[Wejchert and Tesauro, 1990] Jakub Wejchert and Gerald Tesauro. Neural Network Visualization. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 465–472. Morgan Kaufmann, San Mateo, CA, 1990.

[Werbos, 1974] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, 1974.

[Widrow and Winter, 1988] Bernard Widrow and Rodney Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Computer*, pages 25–39, March 1988.

[Xu, 1990] Xiaoshu Xu. *Acceleration of the learning of artificial neural networks utilizing a thermodynamic and kinetic model with applications in sensor data processing.* PhD thesis, Colorado School of Mines, Golden, CO 80401, October 1990.

[Tesauro and Janssens, 1988] Gerald Tesauro and Robert Janssens. Scaling relationships in back-propagation learning: dependence on predicate order. Technical Report CCSR-88-1, Center for Complex Systems Research, University of Illinois at Urbana-Champaign, February 1988.

[Thrun and Mitchell, 1993] Sebastian B. Thrun and Tom M. Mitchell. Integrating inductive neural network learning and explanation-based learning. In C.L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[Thrun *et al.*, 1991] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K De Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Konenenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The monk's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, December 1991.

[Toates, 1980] F. M. Toates. *Animal behavior: a systems approach.* Wiley, New York, 1980.

[Towell *et al.*, 1990] Geoffrey G. Towell, Jude W. Shavlik, and Michiel O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of AAAI-90*, pages 861–866. Morgan Kaufmann, July 1990.

[Utgoff, 1986] P. E. Utgoff. Shift of bias for inductive concept learning. In Jaime G. Carbonell, Ryszard S. Michalski, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 107–148. Morgan Kaufmann, 1986.

[Utgoff, 1988] Paul E. Utgoff. Id5: An incremental ID3. In *Proceedings of the fifth international workshop on Machine Learning*, University of Michigan, June 1988. Machine Learning.

[Waibel *et al.*, 1989] Alexander Waibel, Hidefumi Sawai, and Kiyohiro Shikano. Modularity and scaling in large phonemic neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12):1888–1898, December 1989.

[Waibel, 1989] Alexander Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1:39–46, 1989.

[Watrous, 1991] Raymond L. Watrous. Current status of peterson-barney vowel formant data. *Journal of the Acoustical Society of America*, 89(3), March 1991.

[Weber, 1990] Bruce Weber. The scary 64. *The New York Times*, 1990. October 30.

[Weigend *et al.*, 1991] Andreas S. Weigend, David E. Rumelhart, and Bernardo A. Huberman. Generalization by weight-elimination with application to forecasting. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 875–882. Morgan Kaufmann, San Mateo, CA, 1991.

[Weiss and Kulikowski, 1991] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems that Learn.* Morgan Kaufmann, 1991.

Intelligence (AAAI86), pages 564–569. The American Association for Artificial Intelligence, Morgan Kaufmann Publishers, Inc., 1986.

[Schlimmer and Granger, 1986] Jeffrey C. Schlimmer and Richard H. Granger. Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI86)*, pages 502–507. The American Association for Artificial Intelligence, Morgan Kaufmann Publishers, Inc., 1986.

[Schwartz, 1992] Evan I. Schwartz. Catalog merchandising. *Business Week*, March 2 1992.

[Shannon, 1949] Claude E. Shannon. *The mathematical theory of communication.* University of Illinois Press, Urbana, IL, 1949.

[Shapiro, 1988] Alan D. Shapiro. *Structured Induction in Expert Systems.* Addison-Wesley, 1988.

[Sharkey and Sharkey, 1992] Noel E. Sharkey and Amanda J. C. Sharkey. Adaptive generalisation and the transfer of knowledge, 1992. Working paper, Center for Connection Science, University of Exeter, 1992.

[Shavlik *et al.*, 1989] Jude W. Shavlik, Raymond J. Mooney, and Geoffrey G. Towell. Symbolic and neural net learning algorithms: An experimental comparison. Technical Report 857, Department of Computer Science, University of Wisconsin, Madison, Wisconsin, June 1989.

[Shavlik *et al.*, 1991] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural net learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, 1991.

[Shrager and Blumenthal, 1992] Jeff Shrager and D. Blumenthal. Transfer in recurrent networks: a preliminary report and request for advice. *Electronic mail message <92Jun29.011830pdt.38019@huh.parc.xerox.com>, archived by Connectionists-Request@cs.cmu.edu.*, 1992.

[Sontag, 1989] Eduardo Sontag. Sigmoids distinguish more efficiently than heavisides. *Neural Computation*, 1:470–472, 1989.

[Sony Corporation, 1992] Sony Corporation. Neural network with normalized learning constant for high-speed stable learning, 1992. U.S. Patent number 5,093,899 awarded 3/92.

[Soucek and The IRIS Group, 1992] Branko Soucek and The IRIS Group. *Fast learning and Invariant Object Recognition: The Sixth Generation Breakthrough.* John Wiley and Sons, Inc., New York, 1992.

[Tenorio and Lee, 1989] Manoel Fernando Tenorio and Wei-Tsih Lee. Self organizing neural network for optimum supervised learning. Technical Report TR-EE 89-30, Purdue Univ. School of Elec. Eng., June 1989.

[Tenorio *et al.*, 1990] M. Fernando Tenorio, M. Daniel Tom, and Richard G. Schwartz. Adaptive networks as a model for human speech development. Technical Report (unnumbered), Parallel Distributed Structures Laboratory, Purdue University, 1990.

[Quinlan, 1983] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine Learning*, pages 463–482. Palo Alto, CA: Tioga Publishing Company, 1983.

[Quinlan, 1987a] J. R. Quinlan. Generating production rules from decision trees. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, 1987.

[Quinlan, 1987b] J. R. Quinlan. Simplifying decision trees. *International Journal of Man Machine Studies*, 27, 1987.

[Quinlan, 1988] J. R. Quinlan. An empirical comparison of genetic and decision-tree classifiers. In *Proceedings of the fifth international workshop on Machine Learning*, University of Michigan, June 1988. Machine Learning.

[Ramachandran and Pratt, 1992] Sowmya Ramachandran and Lorien Pratt. Information measure based skeletonisation. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 1080–1087. Morgan Kaufmann, San Mateo, CA, 1992.

[Ratcliff, 1990] Roger Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308, 1990.

[Reilly *et al.*, 1992] Rosemarie Reilly, Xiaoshu Xu, and Jerry Jones. Neural network application to acoustic emission signal processing. In *Corrosion 92 conference proceedings*, Nashville, TN, June 1992. Proceedings of the National Association of Corrosion Engineers.

[Rosenblatt, 1959] Frank Rosenblatt. Two theorems of statistical separability in the perceptron. In *Proceedings of a Symposium on the Mechanization of Thought Processes*, pages 421–456, London, 1959. Her Majesty's Stationary Office.

[Rumelhart *et al.*, 1987a] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press: Bradford Books, 1987.

[Rumelhart *et al.*, 1987b] David E. Rumelhart, James L. McClelland, and The PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press: Bradford Books, 1987.

[Rutenberg, 1992] M. R. Rutenberg. PAPNET: A neural net based cytological screening system. In *Advances in Neural Information Processing Systems 4*, San Mateo, CA, 1992. Morgan Kaufmann.

[Schaffer, 1992] Cullen Schaffer. Sparse data and the effect of overfitting avoidance in decision tree induction. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, San Jose, CA, 1992.

[Schlimmer and Fisher, 1986] J. Schlimmer and D. Fisher. A case study in incremental concept learning. In *Proceedings of the Fifth National Conference on Artificial*

[Oppenheim and Schafer, 1975] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[Parker, 1982] D. Parker. Learning logic. Invention Report S81-64, File 1, Office of Technology licensing, Stanford University, 1982.

[Pazzani, 1989] Michael J. Pazzani. Detecting and correcting errors of omission after explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 713–717, August 1989.

[Peterson and Anderson, 1987] Carsten Peterson and James R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 995, 1987.

[Peterson and Barney, 1952] Peterson and Barney. Control methods used in a study of the vowels. *J. Acoust. Soc. Am.*, 24(2):175–184, 1952.

[Piramuthu, 1992] Selwyn Piramuthu. Constructing efficient features for back propagation. In Brank Soucek and The IRIS Group, editors, *Fast learning and Invariant Object Recognition: The Sixth Generation Breakthrough*, chapter 4. John Wiley and Sons, Inc., New York, 1992.

[Port, 1986] Otis Port. Computers that come awfully close to thinking. *Business Week*, June 2, 1986.

[Pratt and Kamm, 1991] Lorien Y. Pratt and Candace A. Kamm. Improving a phoneme classification neural network through problem decomposition. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN-91)*, pages 821–826, Seattle, WA, July 1991. IEEE.

[Pratt and Norton, 1990] Lorien Y. Pratt and Steve W. Norton. Neural networks and decision tree induction: Exploring the relationship between two research areas, 1990. Workshop Summary, Neural Information Processing Systems (NIPS).

[Pratt *et al.*, 1990] Lorien Y. Pratt, Kathleen D. Cebulka, and Peter A. Clitherow. Residual speech signal compression: An experiment in the practical application of neural network technology. In *Proceedings of Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE '90*, pages 1063–1072. Association for Computing Machinery, July 1990.

[Pratt *et al.*, 1991] Lorien Y. Pratt, Jack Mostow, and Candace A. Kamm. Direct transfer of learned information among neural networks. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 584–589, Anaheim, CA, 1991.

[Pratt, 1993] Lorien Y. Pratt. Experiments in the transfer of knowledge between neural networks. In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems, Constraints and Prospects*, chapter 4.1. MIT Press, 1993. In Press.

[Qian and Sejnowski, 1988] N. Qian and T. J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202:865–884, 1988.

[Michalski, 1986] Ryszard S. Michalski. Understanding the nature of learning: Issues and research directions. In Ryszard S. Michalski, Ryszard S. Michalski, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 3–25. Morgan Kaufmann Publishers, Inc., 1986.

[Minai and Williams, 1992] Ali A. Minai and Ronald D. Williams. Fast back propagation with adaptive decoupled momentum. In Branko Soucek and the IRIS group, editors, *Fast learning and Invariant Object Recognition: The Sixth Generation Breakthrough*, chapter 2. John Wiley and Sons, Inc., New York, 1992.

[Mingers, 1989] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989.

[Minsky and Papert, 1969] Marvin L. Minsky and Seymour A. Papert. *Perceptrons.* The MIT Press, Cambridge, MA, 1969.

[Mitchell *et al.*, 1985] Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Explanation-based generalization: A unifying view. Technical Report ML-TR-2, Rutgers University Department of Computer Science, August 1985.

[Mitchell, 1980] Tom M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, New Brunswick, NJ, 1980.

[Moody and Darken, 1989] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[Moody *et al.*, 1992] John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors. *Advances in Neural Information Processing Systems 4.* Morgan Kaufmann, San Mateo, CA, 1992.

[Mooney and Ourston, 1989] Raymond Mooney and Dirk Ourston. Induction over the unexplained: Integrated learning of concepts with both explainable and conventional aspects. In *Proceedings of the Sixth International Workshop on Machine Learning, Cornell University*, pages 5–7, June 1989.

[Mooney *et al.*, 1989] Raymond J. Mooney, J. W. Shavlik, G. G. Towell, and A. Gove. An experimental comparison of symbolic and connectionist learning algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 775–780, August 1989.

[Naik *et al.*, 1992] D. K. Naik, R. J. Mammone, and A. Agarwal. Meta-neural network approach to learning by learning. In *Intelligence Engineering Systems through Artificial Neural Networks*, volume 2, pages 245–252. The American Society of Mechanical Engineers, ASME Press, 1992.

[Nilsson, 1965] N. J. Nilsson. *Learning Machines: Foundations of trainable pattern-classifyig systems.* Mcgraw-Hill, New York, 1965.

[Nowlan, 1991] Steve Nowlan. Evaluation of adaptive mixtures of competing experts. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 774–780. Morgan Kaufmann, San Mateo, CA, 1991.

[Lang, 1990] Kevin Lang. Variable resolution learning techniques for speech recognition. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann, San Mateo, CA, 1990.

[Laws, 1991] Ken Laws. Resources – commercial neural-network software. *The computists' communique*, 1(40), December 27, 1991.

[Le Cun, 1986] Yann Le Cun. Learning proesses in an asymmetric threshold network. In E. Beinenstock, Souli Fogelman, and G. Weisbuch, editors, *Disordered Systems and Biological Organization.* springer, Berlin, 1986.

[Litzkow *et al.*, 1988] M. Litzkow, M. Livny, and M. W. Mutka. Condor – a hunter of idle workstations. In *Proceedings of the Eighth International Conference on Distributed Computing Systems*, 1988.

[Lukowiak and Freedman, 1983] K. Lukowiak and L. Freedman. The gill withdrawal reflex is suppressed in sexually active *aplysia*. *Journal of Comparitive Physiology*, 117, 1983.

[Lukowiak, 1980] K. Lukowiak. CNS control over gill reflex behaviors in *aplysia*: Satiation causes an increase in the suppressive control in older but not in young animals. *Journal of Neurobiology*, 11, 1980.

[Maren *et al.*, 1990] Allianna J. Maren, Craig T. Harston, and Robert M. Pap. *Handbook of neural computing applications.* Academic Press, 1990.

[Martin, 1988] Gale Martin. The effects of old learning on new in hopfield and backpropagation nets. Technical Report ACA-HI-019, Microelectronics and Computer Technology Corporation (MCC), 1988.

[Matsushita Electric Industrial, 1992] Matsushita Electric Industrial. Neural network with selective error reduction to increase learning speed, 1992. U.S. Patent number 5,129,039, awarded 7/92.

[McClelland and Rumelhart, 1988] James L. McClelland and David E. Rumelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.* The MIT Press, Cambridge, MA, 1988.

[McCloskey and Cohen, 1989] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: the sequential learning problem. *The psychology of learning and motivation*, 24, 1989.

[McCulloch and Pitts, 1943] W. S. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943. Reprinted in W. S. McCulloch, *Embodiments of Mind*, pp. 19-39 (MIT Press, Cambridge, Mass., 1965).

[Michalski and Larson, 1978] R. S. Michalski and J. B. Larson. Selection of most representative training examples and incremental generation of vl1 hypotheses: The underlying methodology and the description of programs ESEL and AQ11. Technical Report UIUCDCS-R-78-867, Department of Computer Science, University of Illinois and Urbana-Champaign, May 1978.

[Hertz *et al.*, 1991] John A. Hertz, Richard G. Palmer, and Anders S. Krogh. *Introduction to the theory of neural computation.* Addison-Wesley, Redwood City, CA, 1991.

[Hinton *et al.*, 1984] Geoffrey E. Hinton, Terrence J. Sejnowski, and David H. Ackley. Boltzmann machines: Constraint satisfaction networks that learn. Technical Report CMU-CS-84-119, Carnegie-Mellon University, Pittsburgh, PA 15213, 1984.

[Hinton, 1985] Geoffrey E. Hinton. Learning in parallel networks. *BYTE*, April 1985.

[Hirsh, 1988] Haym Hirsh. Empirical Techniques for Repairing Imperfect Theories. In *Proceedings of the EBL symposium*, June 1988.

[Holte *et al.*, 1989] Robert C. Holte, Liane E. Acker, and Bruce W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 813–818, Detroit, MI, 1989. Morgan Kaufmann.

[Hopfield and Tank, 1986] John J. Hopfield and David W. Tank. Computing with neural circuits: A model. *Science*, 233:625–633, August 1986.

[Hopfield, 1982] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of scientists, USA*, 79:2554–2558, April 1982.

[Hornik *et al.*, 1989] K.M. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[IEEE, 1991] IEEE. *Proceedings of the International Joint Conference on Neural Networks (IJCNN-91).* IEEE, Seattle, WA, July 1991.

[Intel, 1990] inc. Intel. Intel has the answers, 1990. (flyer describing neural network chip).

[Jacobs, 1988] Robert A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1, 1988.

[Judd, 1988] Stephen Judd. Learning in neural networks. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 2–8. Morgan Kaufmann, 1988.

[Kolen and Pollack, 1990] John F. Kolen and Jordan Pollack. Scenes from exclusive-or: Back propagation is sensitive to initial conditions. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, page 868, Cambridge, MA, July 1990.

[Kruschke, 1990] John K. Kruschke. ALCOVE: A connectionist model of category learning. Technical Report 19, Indiana University Cognitive Science Department, June 1990.

[Landauer *et al.*, 1987] T. K. Landauer, C. A. Kamm, and S. Singhal. Teaching a minimally structured back-propagation network to recognize speech sounds, 1987.

[Feldman, 1985] Jerome A. Feldman. Connections; massive parallelism in natural and artificial intelligence. *Byte*, April 1985.

[Fisher and McKusick, 1989] Douglas H. Fisher and Kathleen B. McKusick. An empirical comparison of ID3 and back-propagation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 788–793, Detroit, MI, August 1989.

[Fisher, 1936] R. A. Fisher. The use of multiple measurements in taxonomic problems. *American Eugenics*, 7(1):179–188, 1936. Also in *Contributions to Mathematical Statistics*, John Wiley, New York, 1950.

[Flann and Dietterich, 1989] Nicholas S. Flann and Thomas G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226, 1989.

[Fukunaga, 1972] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, Harcourt Brace Jovanovich, San Diego, CA, 1972.

[Fukunaga, 1990] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic press, second edition, 1990.

[Gallant, 1988] Stephen I. Gallant. Connectionist Expert Systems. *Communications of the ACM*, 31(2):152–169, 1988.

[Gennari *et al.*, 1989] J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[Gorman, 1988] Christine Gorman. Putting Brainpower in a Box: Neural networks could change the way computers work. *Time*, August 1988.

[Hall, 1988] R. P. Hall. Computational approaches to analogical reasoning: a comparative analysis. *Artificial Intelligence*, 1988.

[Hanson and Burr, 1987] Stephen José Hanson and David J. Burr. Knowledge Representation in Connectionist Networks. Technical report, Bell Communications Research, Morristown, New Jersey 07960, 1987.

[Hanson and Pratt, 1989] Stephen José Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 177–185. Morgan Kaufmann, San Mateo, CA, 1989.

[Hataoka and Waibel, 1989] Nobuo Hataoka and Alex H. Waibel. Speaker-independent phoneme recognition on TIMIT database using integated time-delay neural networks (TDNNs). Technical Report CMU-CS-89-190, Carnegie-Mellon University, November 1989.

[Hayes-Roth *et al.*, 1983] Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat. *Building Expert Systems.* Addison-Wesley, Reading, Massachusetts, 1983.

[Hecht-Nielson, 1989] Robert Hecht-Nielson. Knowledgenet sales flyer, 1989.

[DARPA, 1988] DARPA. DARPA Neural Network study: October 1987-February 1988, Executive Summary, 1988.

[Dattatreya and Kanal, 1985] G. R. Dattatreya and L. N. Kanal. Decision Trees in Pattern Recognition. In L. Kanal and A. Rosenfeld, editors, *Progress in Pattern Recognition 2*. North-Holland, 1985.

[Detrano *et al.*, 1989] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J. Schmid, S. Sandhu, K. Guppy, S. Lee, and V. Froelicher. International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, 64:304–310, 1989.

[Dewan and Sontag, 1990] Hasanat M. Dewan and Eduardo Sontag. Using extrapolation to speed up the backpropagation algorithm. In *Proceedings of the International Joint Conference on Neural Networks, Washington, DC*, volume 1, pages 613–616. IEEE Publications, Inc., January 1990.

[Dietterich and Bakiri, 1991] Thomas G. Dietterich and Ghulum Bakiri. Error-correcting output codes: a general method for improving multiclass inductive learning programs. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 572–577, Anaheim, CA, 1991.

[Dietterich *et al.*, 1990] Tom G. Dietterich, Hermann Hild, and Ghulum Bakiri. A comparative study of ID3 and backpropagation for English text-to-speech mapping. Technical Report (unnumbered), Department of Computer Science, Oregon State University, Corvallis, OR 97331-3902, 1990.

[Duda and Hart, 1973] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[Durbin *et al.*, 1989] R. Durbin, C. Miall, and G. Mitchison, editors. *The Computing Neuron*. Addison Wesley, Reading, Massachusetts, 1989.

[Eberhart and Dobbins, 1990] Russell C. Eberhart and Roy W. Dobbins, editors. *Neural network PC tools: a practical guide*. Academic Press, 1990.

[Fahlman and Lebiere, 1990] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA, 1990.

[Fahlman, 1988] Scott Fahlman. Faster learning variations on back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, San Mateo, CA, 1988. Morgan Kaufmann.

[Fahlman, 1990] Scott Fahlman. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon Computer Science Department, Pittsburgh, PA, February 1990.

[Fawcett and Utgoff, 1991] Tom E. Fawcett and Paul E. Utgoff. A hybrid method for feature generation. In *Proceedings of the Eighth International Workshop on Machine Learning, 1991*, pages 137–141, Evanson, IL, 1991.

[Belsie, 1991] Laurent Belsie. Smart machines 'learn'. *The Christian Science Monitor*, 1991. April 11.

[Breiman *et al.*, 1984] L. Breiman, J. H. Friedman, R. A. Olshen, and J. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[Brent, 1990] Richard P. Brent. Fast training algorithms for multi-layer neural nets. Technical Report NA-90-03, Numerical Analysis Project, Computer Science Department, Stanford University, March 1990.

[Buchanan and Mitchell, 1978] B. G. Buchanan and T. Mitchell. Model directed learning by production rules. In *Pattern-Directed Inference Systems*, pages 297–312. Academic Press, New York, 1978.

[Buntine and Niblett, 1992] Wray Buntine and Tim Niblett. Technical note: A further comparison of splitting rules for decision tree induction. *Machine Learning*, 8(1), 1992.

[Callan and Utgoff, 1991] James P. Callan and Paul E. Utgoff. Constructive induction on domain information. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 614–619, Anaheim, CA, 1991.

[Carpenter and Grossberg, 1991] Gail A. Carpenter and Stephen Grossberg. *Pattern Recognition by self-organizing neural networks.* MIT Press, 1991.

[Carpenter *et al.*, 1991] Gail A. Carpenter, Stephen Grossberg, and John H. Reynolds. ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4:565–588, 1991.

[Chan and Fallside, 1987] L.W. Chan and F. Fallside. An adaptive training algorithm for back propagation networks. *Computer Speech and Language*, 2:205–218, 1987.

[Cohen, 1990] William Cohen. Learning approximate control rules of high utility. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 268–276, Austin, Texas, July 1990.

[Cole *et al.*, 1990] R. A. Cole, Y. K. Muthusamy, L. Atlas, T. Leen, and M. Rudnick. Speaker-independent vowel recognition: Comparison of backpropagation and trained classification trees. In *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences, Kailua-Kona, Hawaii, January 2-5*, pages 132–141, 1990.

[Cover and Hart, 1967] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[Cybenko, 1989] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4), 1989.

[Danyluk, 1989] Andrea Pohoreckyj Danyluk. Finding new rules for incomplete theories: explicit biases for induction with contextual information. In *Proceedings of the Sixth International Workshop on Machine Learning, Cornell University*, pages 34–36, June 1989.

References

[Abramson, 1963] Norman Abramson. *Information Theory and Coding*. Mcgraw-Hill, New York, 1963.

[Ackley *et al.*, 1985] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[Agarwal *et al.*, 1992] A. Agarwal, R. J. Mammone, and D. K. Naik. An on-line training algorithm to overcome catastrophic forgetting. In *Intelligence Engineering Systems through Artificial Neural Networks*, volume 2, pages 239–244. The American Society of Mechanical Engineers, ASME Press, 1992.

[Aha *et al.*, 1991] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[Allman, 1986] William F. Allman. Mindworks. *Science86*, 7(4), May 1986.

[Atlas *et al.*, 1990a] L. E. Atlas, J. Connor, D. Park, A. Lippman, R. Cole, M. El-Sharkawi, R. J. Marks, Y. Muthusamy, and M. Rudnick. A performance comparison of trained multi-layer perceptrons and trained classification trees. In *IEEE International Conference on Systems, Man, and Cybernetics, Cambridge, Massachusetts, November 14-17*, 1990.

[Atlas *et al.*, 1990b] Les Atlas, David Cohn, and Richard Ladner. Training connectionist networks with queries and selective sampling. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 566–573. Morgan Kaufmann, San Mateo, CA, 1990.

[Atlas *et al.*, 1990c] Les Atlas, Ronald Cole, Yeshwant Muthasamy, Alan Lippman, Jerome Connor, Dong Park, Mohamed El-Sharkawi, and Robert J. Marks II. A performance comparison of trained multi-layer perceptrons and trained classification trees. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 622–629, San Mateo, CA, 1990. Morgan Kaufmann.

[Atlas *et al.*, 1990d] Les Atlas, Ronald Cole, Yeshwant Muthusamy, A. Lippman, Jerome Connor, D. Park, M. El-Sharkawi, and Robert J. Marks II. Performance comparisons between backpropagation networks and classification trees on three real-world applications. *Proceedings of the IEEE*, 78(10):1614–1619, October 1990.

[Barnard and Cole, 1989] Etienne Barnard and Ronald A. Cole. A neural-net training program based on conjugate-gradient optimization. Technical Report CSE 89-014, Oregon Graduate Center, July 1989.

[Barr *et al.*, 1989] Avron Barr, Paul R. Cohen, and Edward A. Feigenbaum. *The Handbook of Artificial Intelligence*, volume IV. Addison-Wesley, Reading, MA, 1989.

with the input or output representation in the source network, instead of discarding the entire layer because of incompatibilities. The most straightforward approach to this technique is to randomize weights when output or input unit identities are different in the source and in the target. If an input attribute is not available in the target data, but was available in the source data, then weights to that unit are removed before further training is done. If there are known relationships between source and target attributes and output classes (perhaps a target class is a superset of two source classes), then more sophisticated problem-dependent processing might be done to, for instance, combine two source input nodes into a single target network input node that receives some function of the values in the source nodes.

6.4 Conclusion

This dissertation has presented the idea of transfer between neural networks as a way of improving their learning speed. We've described and motivated the Discriminability-Based Transfer (DBT) algorithm, and presented empirical results on a variety of benchmark domains that demonstrate DBT's effectiveness. The primary contributions of this dissertation are as follows:

- A formulation of the transfer problem (Section 1.4).
- A demonstration that literal transfer is not robust (Chapter 3).
- The DBT algorithm (Chapter 4).
- Insights about the effect of hyperplane magnitudes on transfer (Chapter 3, plus background study in [Pratt, 1993]).
- A conceptual model of why DBT works. This model makes our assumptions about the method explicit, which facilitates verification and extension of DBT (Chapter 5).
- Indirect support (via the success of DBT) for these assumptions (Section 5.2).
- An analysis of the effectiveness of using the average IM to predict DBT's effectiveness (Chapter 5).
- The design of a number of transfer benchmark tasks that can be used as a baseline for evaluating future transfer methods (Chapter 4).

This dissertation has formulated the problem of transfer between neural networks and demonstrated that this approach can substantially improve learning on new tasks. We believe that mechanisms for transfer between learning tasks will serve an important role in machine learning systems of the future, allowing them to take advantage of previous knowledge while building new skills.

6.3.4 More complex transfer functions

As discussed in Section 3.3.2, DBT calculates target magnitudes M_t to be proportional to the product $M_s \times IM_t$, so $M_t = s \times M_s \times IM_t$, where M_t is the magnitude of a target network hyperplane, s is a constant of proportionality, M_s is the magnitude of a source network hyperplane, and IM_t is the discriminability of a source network hyperplane on the target training data. Other functional forms are possible, and may give DBT more flexibility. For instance, we might consider $M_t = s_0 + s_1 \times M_s + s_2 \times IM_t$, thus allowing an arbitrary linear relationship. Background studies on back-propagation dynamics (especially those that explore the effect of weight magnitudes on hyperplane movement) would help to constrain the choice of this functional form. It's worth exploring whether such approaches could improve the DBT algorithm – they might obviate the need for separate S and C values for every domain (or allow them to be found faster, via search in a smaller space) and so reduce the time that the DBT algorithm itself requires to run.

6.3.5 Other paradigms for DBT

The DBT algorithm allows information about the target data to be merged with weights from the source classifier. Such a procedure is applicable in contexts other than transfer, such as on-line learning (where training data change frequently, cf. [Carpenter *et al.*, 1991]), active learning schemes (where new training data are obtained by querying the environment, cf. [Atlas *et al.*, 1990b]), concept drift [Schlimmer and Granger, 1986] (where source and target may not be related) and in pedagogical training methods (where training data are segregated into stages in order to speed up learning) (cf. [Michalski and Larson, 1978]). DBT may also be used as a postprocessor to a method such as KBANN [Towell *et al.*, 1990] to adapt a knowledge source to the training data prior to learning.

6.3.6 Transfer between networks with different features and output classes

There are two general categories of techniques to consider for handling source and target networks that have different input and/or output representations. The first category contains techniques in which we omit entire layers of weights from the transfer. For example, if the input representation differs, but the output representation does not, only the hidden-to-output weights might be included in the transfer. The effectiveness of this idea was supported by [Sharkey and Sharkey, 1992], who showed that HO weights may carry important information via transfer. Conversely, if only the output representation is different, it may be possible to transfer weights from the input-to-hidden layer (this is the approach taken by [Waibel, 1989], [Waibel *et al.*, 1989]). Finally, if neither input nor output representations are the same, but there is more than one hidden layer, then it may be useful to transfer hidden-to-hidden weights between networks.

The second set of techniques for handling differing input and output would consist of methods for coercing weights within a layer of the target network to be compatible

hyperplanes don't separate the target data), the target task is given precedence (hyperplanes are given low magnitudes, and so are allowed to move out of position). DBT was designed this way because the primary baseline against which it was evaluated was networks that were randomly initialized on the target task training data. This allowed DBT to achieve its goal of never doing worse than this baseline, as evaluated through cross-validation on the target task training data that was available.

A different baseline might have been used for evaluating DBT. If we had evaluated the target network's performance on the source as well as the target data, then the target task data, when impoverished, would have been less important. In Section 1.4.6, we made the binary distinction between retaining and not retaining source task performance. Figure 6.3 suggests a continuous distinction along these lines: the more that we are sure that the source task overlaps with the target task, the more we should enforce source task performance on the target task (despite, perhaps, what's indicated by the target data). If we have a model of *what parts* of the source network are relevant to the target task (i.e. IH hyperplanes), then this procedure could be made more sophisticated. We might also have a model of *what I/O patterns* in the source network should be retained in the target task. For example, in our speech recognition transfer task, perhaps low-voiced female speakers should also be used to train the target network, because these overlap with the target task. Also, the results of Section 4.4 indicated that the source and target tasks did have some I/O mappings in common (P and P' overlap). Developing such models, and determining domains in which this sort of information might be available, is an important future direction for this research.

6.3.2 Speed up DBT running time

The DBT subroutine for determining S and C parameters stops its search according to fairly arbitrary conditions. It seems that good values of these parameters could still be obtained, while using a more stringent cutoff on the number of S and C combinations tried. This improvement could considerably speed up the time involved in using the DBT algorithm.

6.3.3 Hyperplane combinations

Section 5.2 described the DBT assumption that the hyperplanes that make up a decision region in a neural network can be effectively evaluated individually, for purposes of transfer. Hyperplanes don't act alone, however, but instead form nonlinear decision regions in feature space by acting in groups. Therefore, the IM of individual hyperplanes is an imperfect predictor of their usefulness. A logical extension to the current formulation of DBT is to analyze the discriminability of these groups. An appropriate approach may be one like that used in the skeletonization algorithm of [Ramachandran and Pratt, 1992], where a decision tree was induced over a set of learned hyperplanes, to determine which were and were not necessary for the separation of training data. That algorithm made a binary decision (keep/don't keep) over the hyperplanes, however. For transfer, it would need to be modified to determine target network hyperplane magnitudes.

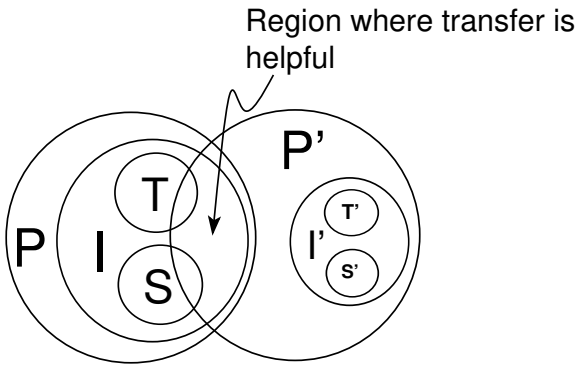


Figure 6.3: How transfer from a source to a target function can be helpful.

The impoverished set of target network training data causes its implementation of P' to give a high error rate. This can be ameliorated by using parts of a trained network for a related task, P, whose network implements the function I, which overlaps with P' as indicated.

Here, the network training process has ignored the source classifier, and has built an inadequate target network based only on the impoverished target training data.

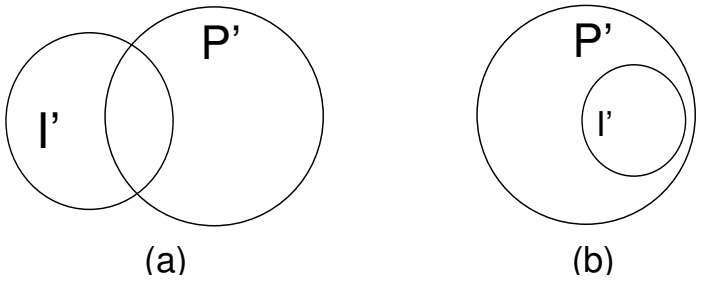


Figure 6.4: Problems that come from mis-balancing source and target information when the target data is impoverished.

In (a), we have used the source network I to implement the target network I', without regard to the target task. In (b), we have used the target training data to implement I', without regard to the source network. By balancing the information from source and target more effectively, performance on the target task might be improved.

Therefore, additional information is needed, beyond the source network, if performance is to be improved for impoverished target data. This information would specify the amount of relative balancing that should be done between the source and target information. This could be done by changing the algorithm for setting source weights, or by reducing the target network training time in an amount proportional to the weight from the source network.

The DBT transfer algorithm presented here assumes that the target training data is representative of the target task. Where the source conflicts with the target (source

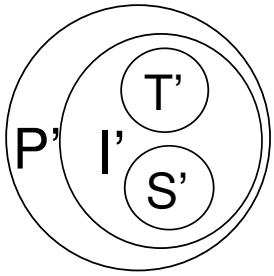


Figure 6.1: Illustration of source network training.

Our goal is to build a network that implements the function P'. By training on the training set S', and cross-validating on the testing set T', we obtain a network that implements some function $I' \subseteq P'$. The difference between I' and P' is the true generalization error. This error is estimated by the error found in the test set T'.

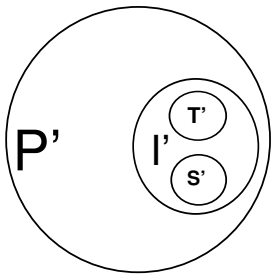


Figure 6.2: Illustration of source network training when training data is impoverished

Because of a shortage of available data, the training set S' and the testing set T' are small relative to the overall function P' to be implemented. Therefore, a network trained using these values only implements the function I', which does not reflect all of P'.

transfer is that it leads to faster learning. In situation (2), DBT transfer may produce a higher asymptotic score, because a randomly initialized network (the baseline) reaches a local minimum.

Which of these two situations holds depends critically on the quality of the baseline search method. Randomly initialized back-propagation networks are subject to local minima if the wrong number of hidden units, or especially if the wrong η and/or α parameters, are used for training.

It was our original goal for the DBT algorithm to lead not only to improved learning speed, but also to improved asymptotic performance. Before performing our benchmark trials, we expected that situation (2) would hold. However, examination of the 700 random baseline runs that we performed indicated the contrary: asymptotic scores for randomly initialized networks had a very small difference (at most around 2%) between the worst and best final score. We suspect that our careful methodology for choosing η, α, and the hidden unit count is responsible.

Although, as discussed in Section 1.4.3, improving learning speed is a very important goal for transfer, it is also worth asking how transfer might be applied differently so as to achieve a performance improvement as well. According to the model above, this may be accomplished in one of two ways. First, transfer tasks could be found that do indeed lead to local minima when a randomly initialized network is used. Secondly, assuming that randomly initialized networks continue to avoid local minima, asymptotic improvement via transfer is possible when the target training data is impoverished in some way. We'll explore this latter situation in the remainder of this section.

By "impoverished" training data, we mean that it doesn't adequately represent the full generalization task. To visualize this situation, first consider the situation where training data is not impoverished. Let P' be a set of <input,desired output> pairs that define a function to be learned. Let $S' \subset P'$ be a set of training data used to represent the source task, and $T' \subset P'$ be a set of testing data for that task. By training on S' and cross-validating based on T', we construct a network that implements some function $I' \subseteq P'$, as illustrated in Figure 6.1. This follows the model of [Sharkey and Sharkey, 1992], as was introduced in Section 5.2.

Figure 6.2 uses this same representation to illustrate the situation where the training data is "impoverished" relative to the full task P'. In this case, the function I' that's implemented by the network is not the full function P', and the network will achieve a poor generalization score when tested on P'.

Now consider how transfer can help this situation. Assume that a source network I has been trained using source training and test sets S and T, in an attempt to implement the full function P, which is somehow related to P'. This situation is shown in Figure 6.3.

The problem with this scenario is that we are now given two sources of information as to the proper function P': the source network that implements I and the target training data S' that can implement I'. If we were to weight the information from the source network too heavily, we might have, in the extreme, the situation shown in Figure 6.4 (a). Here, the classifier I' that's been constructed is only relevant to the target task P' where two tasks overlap. If, on the other hand, we were to rely too heavily on the target task information, we'd have the situation shown in Figure 6.4 (b).

- By running randomly initialized networks over several different initial conditions, we demonstrated that our results were not dependent on the particular initial weights that were chosen.
- We performed systematic cross-validation to obtain a good estimate of the true error rate.
- We evaluated not only the differences in learning speed between the methods studied, but also tested the degree of statistical significance in the difference between various curves, for each condition studied. This established that, in a real-world situation where only one run is possible, it is unlikely that a randomly initialized network will learn faster than DBT.

6.3 Future directions

There are many future directions in which this work can be taken. Ideas for future work can be generated by generalizing any of the aspects of our problem formulation described in Section 1.4. Also, the assumptions listed in Section 5.2 can be relaxed – each one leads to a different approach for addressing our formulation of transfer.

In this section, we focus on just a few of the directions that we find most interesting and promising for future work. We begin with a discussion of how we might build transfer techniques that produce asymptotic performance improvement.

6.3.1 Performance improvement and the power vs. generality tradeoff

One assumption of our problem formulation that was described in Section 1.4.3 was that the goal of transfer was to speed up learning. This section considers what happens when we generalize this goal to also include achieving asymptotic performance improvement via transfer.

If we consider back-propagation training as a search process through the set of weight vectors, then setting initial conditions via transfer can be viewed as addressing one of two scenarios:

1. Even without information from a source (i.e. using a randomly initialized network instead), the target search process always reaches its goal. In this case, initializing weights via transfer speeds up the search process because training starts closer to the goal.
2. In some situations, a global error minimum (the "goal") may not be reached – target search may halt in a local minimum instead. In this case, initializing weights via transfer in a part of the search space that's close to a global minimum increases the probability that that minimum will be found.

In situation (1), there can be no asymptotic performance improvement relative to random initialization, because the baseline search process always reaches the best performance possible, given the training data. In this case, the advantage of using

- Section 5.3.1 showed a variety of degrees of relationship, in terms of the average IM measure, between source and target benchmark tasks. DBT demonstrated a learning speed-up in all but one of these transfer tasks.

An additional claim in support of the success of DBT is that:

> **Additional claim:** Networks initialized via literal transfer demonstrated worse average asymptotic performance than randomly initialized networks on some tasks, and so this approach is less robust than DBT.

In support of this claim, Section 4.8.1 described how literal networks gave worse asymptotic performance for some problems, as well as slower learning speeds, in comparison to randomly initialized networks. Literal transfer is therefore not a reliable method, and so the use of the DBT algorithm is justified in comparison to it.

6.2 Validity of empirical tests

For the results in support of the thesis claim listed above to hold, it's important that the tests used to evaluate DBT were methodologically correct.

Chapter 4 described the methodology we used to evaluate DBT on the benchmark tasks. In that chapter we argued for the validity of several aspects of our empirical method. Important aspects of that method are as follows:

- We compared DBT to three different baselines: the performance of randomly initialized networks, that of networks initialized via literal transfer, and the score of source networks on the target task, without further training. These comparisons demonstrated that the success of DBT could not have been achieved by these less complicated methods, and so justified the use of this algorithm.

- By using several different source networks for DBT and literal conditions, we demonstrated that obtaining an average overall speed-up via DBT is not dependent on a particular set of source weights, but rather on the source problem itself, as represented in the source training data.[1]

- We used an automated method for determining the proper number of hidden units in each network, so that we were not able to bias the results by choosing how many to use.

- We used an automated procedure for determining back-propagation parameters η and α, so that we were not able to bias the results by changing these values, or by choosing a fixed set of values that favored our method.

---

[1] Note that the example shown in Figure 5.21 in Section 5.3.3 illustrated that estimating the average IM for purposes of predicting DBT's effectiveness must use the same source network for estimation as is used in transfer. That example doesn't contradict this point, which says that DBT can be effective on several different source networks for the same problem.

Chapter 6
Conclusion

Section 1.5 presented the thesis claims of this dissertation. In Section 6.1, we revisit the thesis claims, summarizing the support for them that has been described in more detail previously. Section 6.2 summarizes the key features of the methodology we used to evaluate DBT. Section 6.3 will then describe several directions for future work, and we'll conclude the dissertation in Section 6.4.

6.1 Review of support for thesis claims

Chapter 4 presented empirical results from tests of the DBT algorithm; Section 4.8.1 showed how these results support the thesis claim.

The primary claim is as follows:

> **Primary claim:** DBT demonstrates a learning speed improvement relative to randomly initialized networks.

In support of this claim, we showed that:

- Over the seven tasks that were studied, the mean (over 100 trials) DBT performance level was never asymptotically significantly worse than the performance of a randomly initialized network.
- DBT demonstrated faster average learning than randomly initialized networks on six out of seven benchmark tasks; on the remaining task its mean learning curve wasn't statistically significantly different from the randomly initialized baseline network.

We qualify the primary claim as follows:

> **Qualification:** The primary claim holds for a non-trivial set of relationships between the source and target tasks.

Note that these relationships have not been precisely identified. However, the following points support this qualification:

- The relationships between source and target networks in the seven benchmark tasks included both the source as a subtask of the target and the source as a more generally related task to the target.
- Section 4.4 showed that the source networks were not able to solve the target tasks directly, without further training, and so one way that there could be a "trivial" relationship between source and target was avoided in our empirical trials.

Method	**Cost**	**Problem**
Use the source data to estimate a classifier that will be constructed from the source training data.	Time to build a simple classifier.	Source hyperplanes from only an approximate source classifier may be misleading.
Use source weights to calculate the average hyperplane IM on the target training data.	*Time to calculate average IM.*	*Ignores effect of S & C, as well as hidden unit interactions.*
Run DBT to determine the initial target network weights; use their magnitudes to estimate DBT learning speed.	Time to run DBT	It is hard to predict final from initial conditions of training. Also, hidden unit interactions aren't taken into account.
Run DBT for a few epochs and use its early learning speed to estimate the remainder of the learning curve.	Time to run DBT, plus time to train for a few epochs.	It's hard to predict final from intermediate conditions.

Table 5.3: Table showing costs and advantages of the various methods for predicting DBT's effectiveness that are reviewed here.

magnitudes. In general, if DBT produces high magnitudes, then this indicates higher relevance of the source to target task, and so predicts better DBT performance. Low magnitudes predict performance that is more similar to that which would be obtained via random initialization.

This approach suffers from two important problems. First, the criterion for a weight magnitude to be "large" varies from problem to problem – we have no *a priori* model of what makes a weight magnitude big enough to cause retention in a particular problem.

Second, and more fundamentally, as discussed above, it is not known in general how to map initial conditions of back-propagation networks to their final positions. As demonstrated by [Kolen and Pollack, 1990], small perturbations in initial conditions can have a large effect on the final outcome. Until this problem is solved (or at least conditions are identified in which it is less of an issue), no method will be able to precisely predict the outcome of DBT training (or any back-propagation method, including random and literal training, for that matter).

Using TSS after some epochs to estimate learning curve after many epochs

The same problems that apply to predicting the DBT learning curve from initial conditions apply to predicting it from the state of the network after training for a few epochs. However, the more epochs that are trained, the more likely will the performance level observed reflect the asymptotic performance level.[6] Unfortunately, since this process involves extensive training, it becomes less practical as an estimation procedure.

5.3.4 Summary

We have presented several increasingly expensive, but potentially more accurate, methods of estimating DBT learning speed. These methods, along with their costs and reasons that they are approximate, are illustrated in Table 5.3.

Even the best of these methods suffers from the fact that we have no model by which to predict a DBT learning curve from initial, or even intermediate, conditions [Kolen and Pollack, 1990]. Though, for this reason, none of these methods are perfect, we have presented empirical evidence in Section 5.3.1 that measuring the average IM of source network hyperplanes may correlate somewhat with the speed-up that will be achieved via DBT.

5.4 Conclusion

This chapter has built on the empirical results of Chapter 4 by asking more detailed questions about why and when DBT works. We have discussed the assumptions upon which the design of DBT depended, and explored methods for quantifying the relationship between two tasks in order to predict the effectiveness of DBT transfer between them.

---

[6] Though note that back-propagation learning is sometimes nonmonotonic – it occasionally loses a performance level it had once achieved.

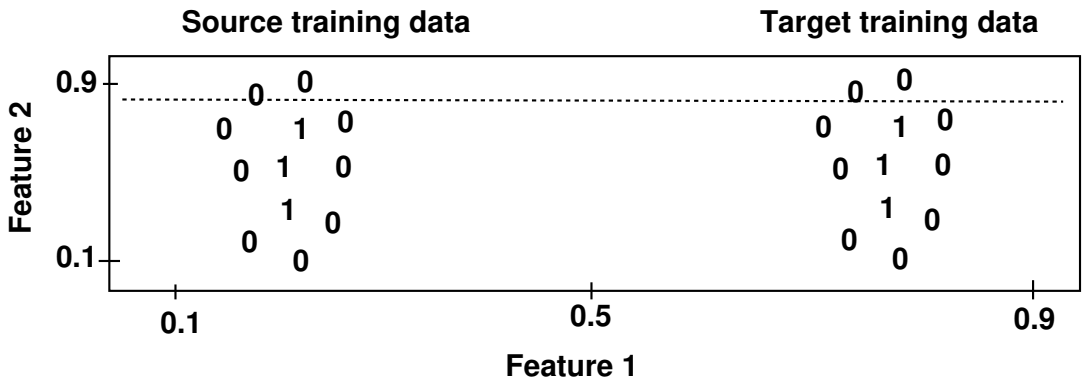


Figure 5.20: Second problem that can occur when estimating the source classifier: this situation will produce an optimistic prediction of DBT performance.

regions is able to generate a classifier with the same complexity as the actual back-propagation classifier, as shown in Figure 5.21. However, the generated classifier in this case is not the same one as will be used for transfer. Here, though the source training data indicates that the source task is indeed potentially relevant to the target task for some solutions to the source problem, this particular source problem solution is not relevant. In general, the source hyperplanes generated may indicate relevance when there is none, or no relevance when it does indeed exist. It is therefore more reliable to use the actual source network when predicting DBT's effectiveness.

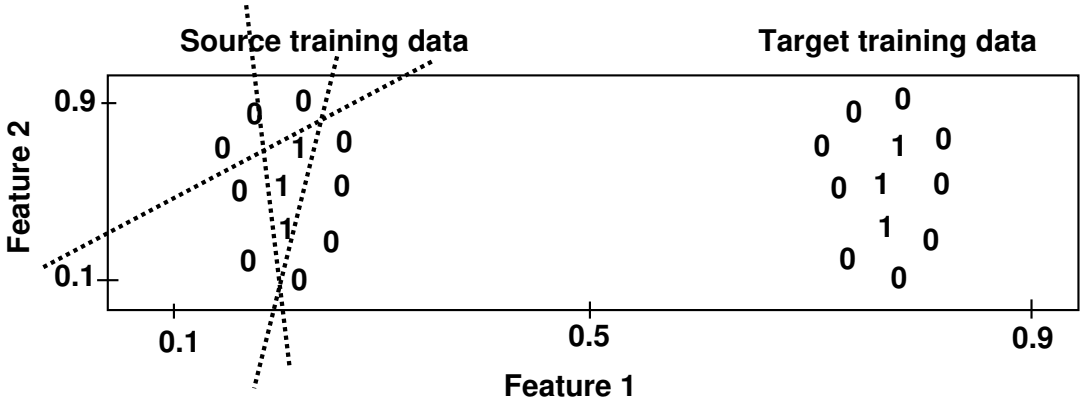


Figure 5.21: Third problem that can occur with estimating the source classifier: since a different source classifier is used for measurement than for transfer, the estimation may be misleadingly optimistic or pessimistic.

Estimating DBT performance from initial conditions

Section 5.3.2 described a variety of reasons that the IM metric may imperfectly predict the improvement in learning speed to be obtained via DBT. These problems are partly caused by the fact that DBT does not use IM directly, but instead uses the initial weight magnitudes to initiate target training. We can address this issue by using DBT-generated initial weight magnitudes to estimate the DBT learning curve. DBT is run, starting with the IM values, and is allowed to determine S and C, and then the initial

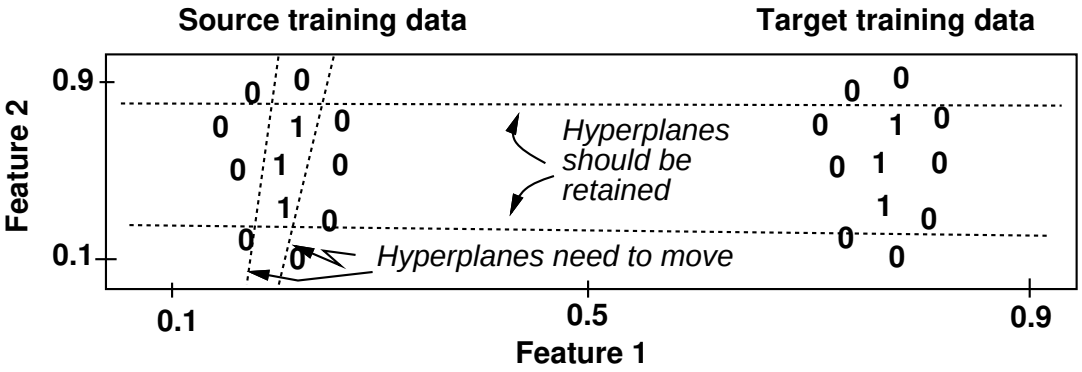


Figure 5.18: Problem illustrating the need for DBT.

Recall that this figure was used to illustrate why literal transfer won't work: some hyperplanes need to move during target training, some don't. In this problem, the source and target tasks are identical, except that the target task has been shifted slightly along one axis, as represented by the training data shown here.

Figure 5.19 illustrates a linear discriminant that might be used to approximate weights that would be obtained on the source task. Any measure of this single source hyperplane on the target training data will indicate its lack of relevance, even though a back-propagation network trained on this source task might very well be helpful. In this case, then, the estimate of task relatedness could be pessimistic relative to DBT's true performance, because better source weights might be generated by actual source training (as in Figure 5.18).

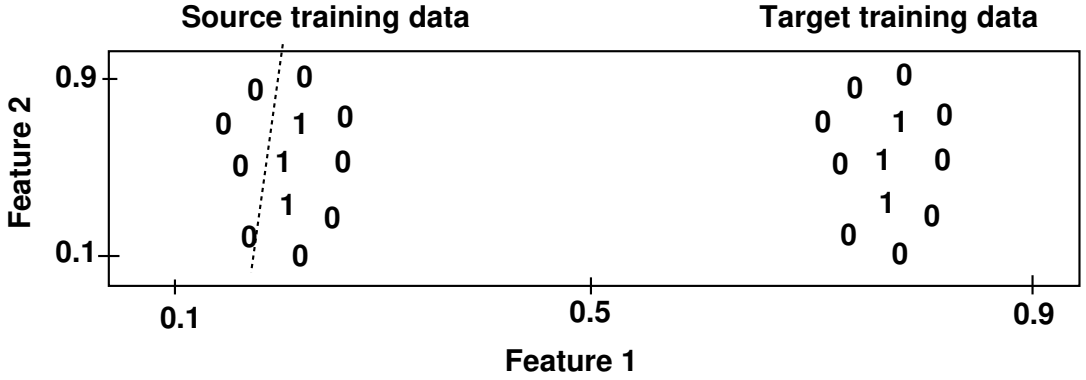


Figure 5.19: First problem that can occur when estimating the source classifier: this situation will produce a pessimistic prediction of DBT performance.

Now consider another situation that might be caused by using a simplified discriminant function, as shown in Figure 5.20. Here, the only hyperplane generated will have a high IM on the target task. In this case, the estimate could optimistically report that DBT would perform better than it actually does. This is because several irrelevant hyperplanes might also be created by back-propagation training on the source task.

Finally, consider the case where the method used to estimate the source decision

Summary

This section has indicated a number of situations where the average IM may not be a good predictor of DBT's effectiveness in speeding up learning. In many cases (one exception is that shown in Figure 5.14), the "unknown" factors that are partially responsible for the approximation are the values of S and C that DBT chooses, which affect hyperplane retentiveness. Therefore, a more expensive, but more accurate measure, would be to use the actual hyperplane magnitudes generated by the full DBT algorithm, which takes the IM as well as S and C into account. This approach is addressed in the next section.

5.3.3 Other methods for estimating source/task relatedness

Estimating DBT's effectiveness via the average IM is only one in a series of methods that make a cost/accuracy trade-off in the prediction of DBT's effectiveness in speeding up learning. These methods roughly correspond to the chronological steps of the DBT that are illustrated in Figure 5.17. We explore the remainder of these methods below.

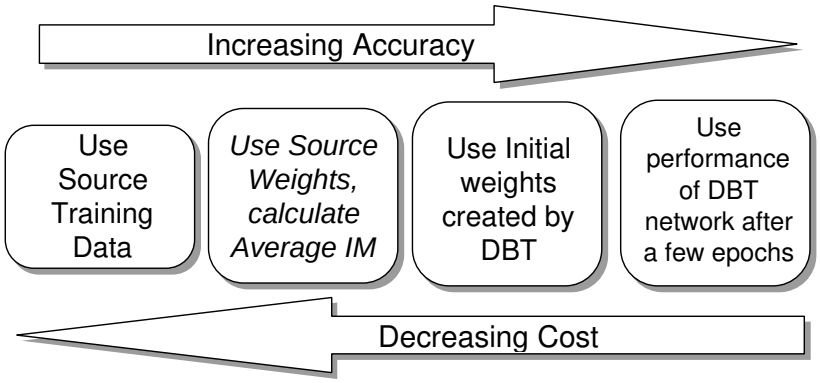


Figure 5.17: Cost/Accuracy tradeoff in prediction methods for DBT.

Estimating DBT performance from only source training data

The least expensive,[5] but most error prone, method for predicting the amount of speed-up to be gained by using DBT uses only the source training data, but does not require that the source network be trained. In this case, source weights might be approximated cheaply by building a simple classifier, such as a linear discriminant, over the source data. To illustrate the problems that can occur with this approach, recall an example situation that we used to motivate the design of DBT, whose illustration is repeated in Figure 5.18. The source task in Figure 5.18 is somewhat related to the target. Some decision regions are in helpful positions and some are not.

[5]Note that this technique is only the least expensive if the time for source training is taken into account, which is not in general the model studied in this dissertation. If we don't assume that this time matters, then this approach may be much more expensive than using source weights, if there's lots of source data available.

back-propagation dynamics, but we haven't proven that it cannot happen.

To illustrate this point further, consider again the situation in Figures 5.13 and 5.14. In these cases, the IM metric may not necessarily be pessimistic, because DBT depends not only on IM, but also on the S and C values that are used in determining target weight magnitudes. If, for example, in Figure 5.13, DBT chose to set weight magnitudes to very low values, any advantage gained by being close to a final position would be lost, because the hyperplane would most likely drift out of position. If, however, this hyperplane received a very high magnitude, then an estimate that said that DBT would do badly might very well be accurate. If it received a moderate magnitude, with which it moved out of its initial position only enough to move into a good position, then DBT might do very well.

A final example of what can go wrong with the IM metric is illustrated by Figure 5.16. Here, all hyperplanes have the same IM, but the ones on the left figure are redundant, and so, together, they carry less useful information to the target task. The average IM is the same in both situations. However, since it uses S and C searching as well as IM, DBT may be able to learn faster for the situation on the right. This is because the S and C search process receives feedback from networks that have been trained for only a few epochs. This feedback may reflect that using the hyperplanes on the right leads to faster training than using the redundant hyperplanes on the left-hand side of Figure 5.16. [4]

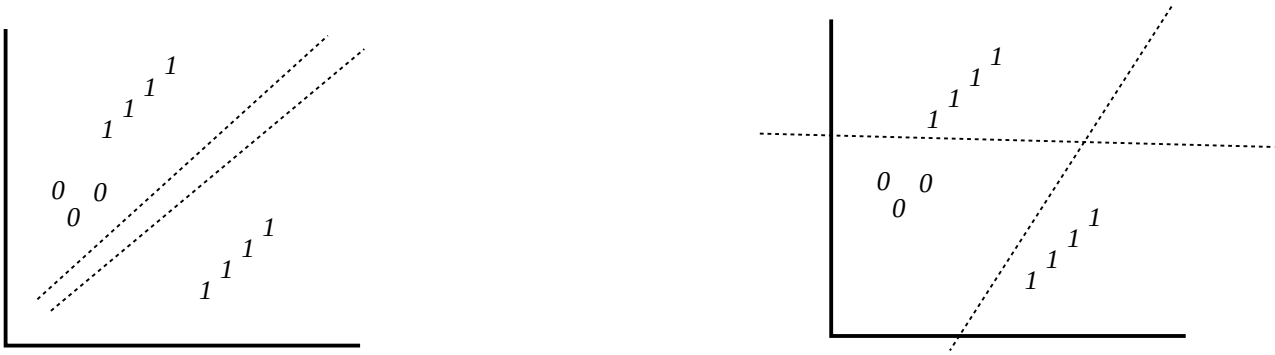


Figure 5.16: Fourth problem that can occur with using just IM for DBT prediction. All four hyperplanes have the same IM value, so both situations have the same average IM. DBT can, however, adjust magnitudes on the problem on the right so that it gives faster learning than in the problem on the left. So using just the average IM is misleading.

[4]Of course, in this particular example, DBT may learn faster than average IM would predict for the situation on the right-hand side for other reasons than the one given, such as that illustrated in Figure 5.14. Learning may also be faster in the right-hand side situation because the hyperplane positions shown have a lower average distance to good positions. The point here is that, since redundant hidden units are erroneously recognized as being potentially useful when the average IM metric is used, this metric may be misleading.

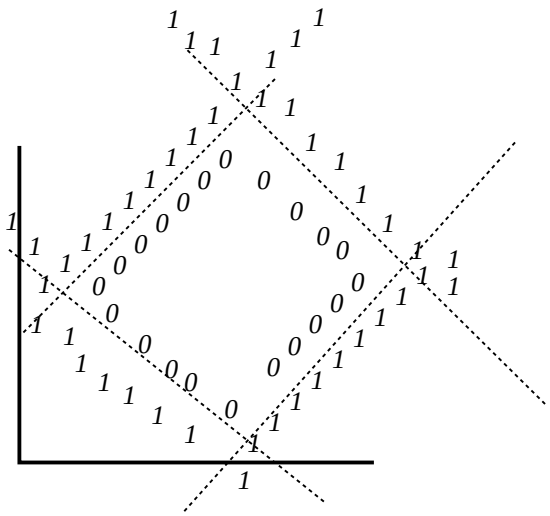


Figure 5.14: Second problem that can occur with using just IM for DBT prediction: this may produce a pessimistic measure.

understanding of back-propagation dynamics, this seems unlikely, but since the S and C values are determined based on a possibly errorful process, it is in principle possible. For example, the initial S and C search might indicate that a low hyperplane magnitude is adequate to retain this hyperplane's position. However, dynamics may arise during later epochs that move the hyperplane out of place, meaning that it actually required a higher S, value. This is more likely in more complicated situations than in this simple example, however.

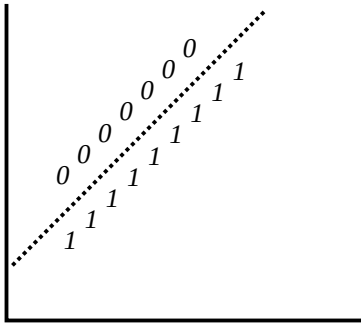


Figure 5.15: Third problem that can occur with using just IM for DBT prediction. If the hyperplane moves out of position, then this may provide an optimistic measure.

This case, along with the previous examples, illustrates the fact that using just the IM metric omits critical information: the retentiveness that DBT chooses for hyperplanes, through its choice of S and C. Although our model of DBT is that it will in general choose higher magnitudes for higher IM hyperplanes, there may be reasons that prevent it from doing so in certain situations. These may include potential error in the S and C estimation process, which determines initial magnitudes via only a small amount of empirical testing on the target task. In particular, it's possible that the small number of epochs that are used to determine S and C would indicate that a low magnitude is better in this situation. This seems unlikely, given our understanding of

resulting learning speed improvement after DBT transfer. These studies indicate that relative IM may be a good way to compare source networks for a particular target on average, although variances among individual runs may obscure this relationship.

We have also shown how average IM relates to learning speed improvement in the benchmark studies of Chapter 4. We have shown again that, despite an apparent average relationship between these two random variables, the variance caused by differences in individual initial weight sets can be substantial.

We now turn to a set of counterexamples to this approach that demonstrate why average IM can be only a rough predictor of DBT effectiveness.

5.3.2 Counterexamples to using the average IM as a predictor of DBT effectiveness

We showed in the studies of the previous section that, on average, high IM corresponds to faster learning. In this section, we explore the situations in which this relationship may not hold.

Figure 5.13 illustrates what can, in principle, go wrong with using the average IM to predict DBT effectiveness. In this situation, there is a low IM (because the hyperplane separates no data), but DBT may nonetheless be able to use the nearby hyperplane by moving it into position. Therefore, the IM measure in this situation could be pessimistic, if the weight magnitudes are set by DBT in such a way that it can take advantage of the fact that the source hyperplane is near to its correct position, despite the fact that it does not separate the data.

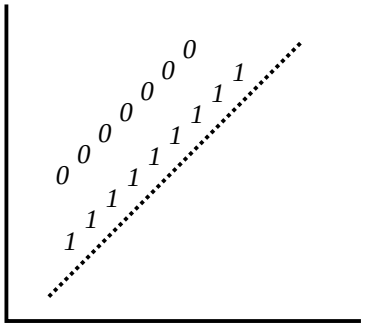


Figure 5.13: First problem that can occur with using just IM to predict the effectiveness of DBT. This might give a pessimistic prediction of DBT performance.

Figure 5.14 demonstrates another situation where the average IM measure may be overly pessimistic. Here, though each hyperplane has a relatively low IM (and so the average IM is low), in combination they do a good job of separating the training data. DBT might therefore choose a high S value, allowing these hyperplanes to remain in place. Target learning after DBT would then be fast, and the IM metric would again be pessimistic.

Figure 5.15 illustrates a situation in which the average IM value may be overly optimistic. It's in principle possible that, despite the high IM value, DBT will choose to loosen this hyperplane such that it is not utilized in further training. Given our

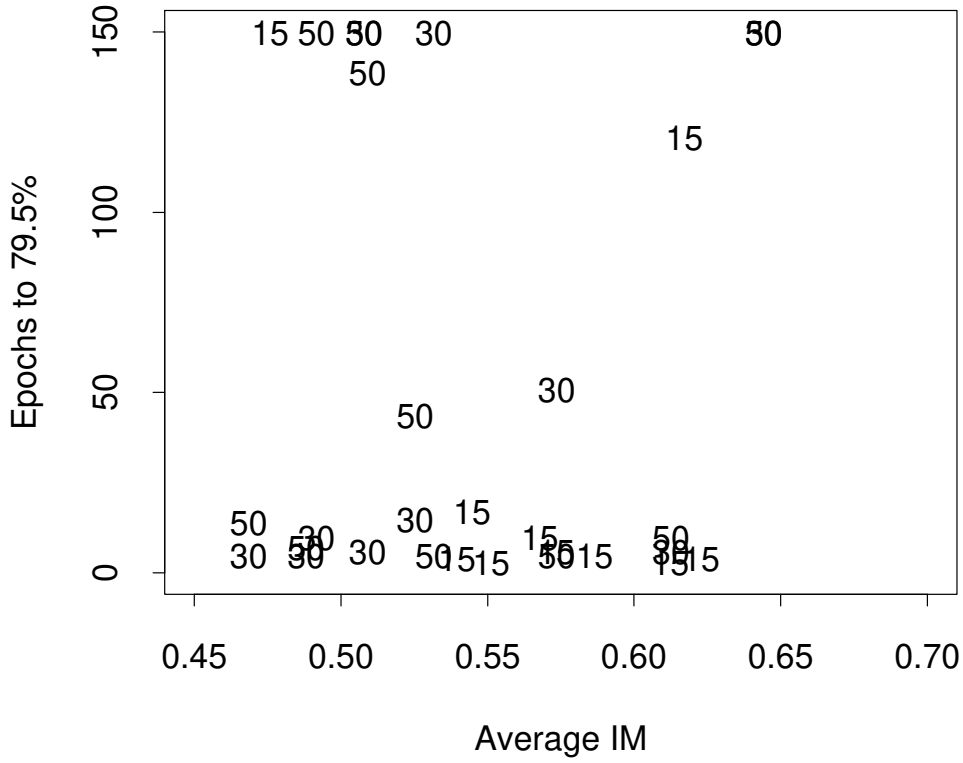


Figure 5.12: Learning time vs average IM for individual Onemale runs.

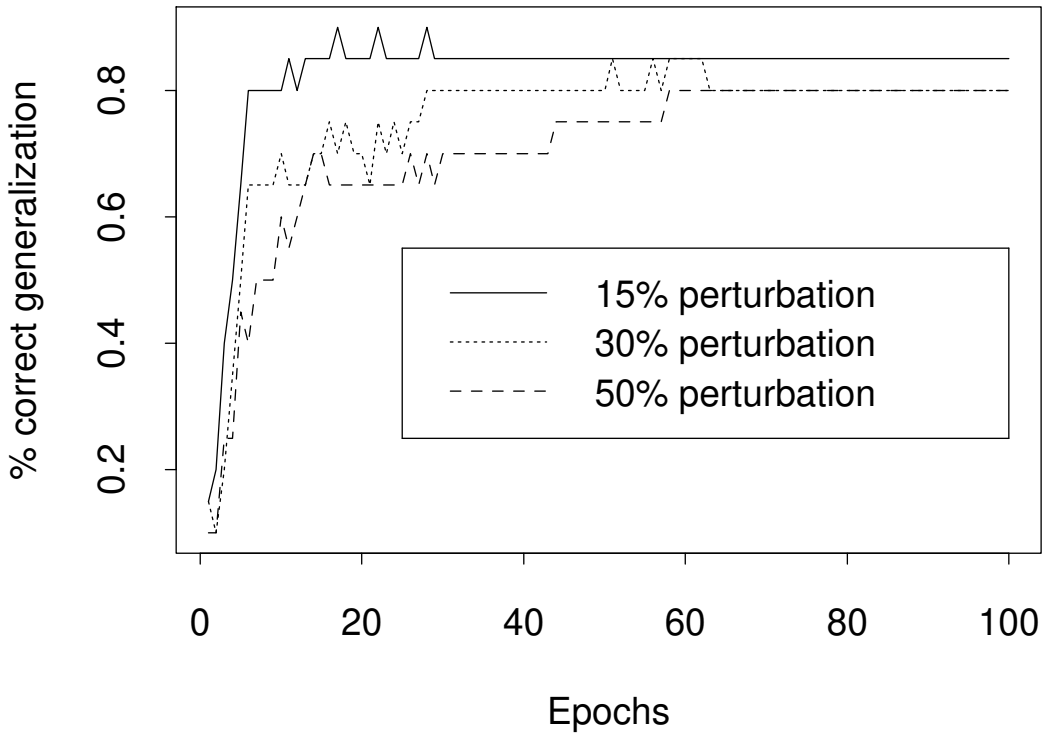


Figure 5.11: Generalization curves for Onemale networks initialized with varying-IM hyperplanes.
Note that larger amounts of perturbation (which correspond to lower IM values) lead to slower learning. In particular, it takes more time to reach the 79.5% criterion.

this same criterion against which to measure the learning speed of networks initialized with different perturbation factors, and that were then run through DBT. The numbers of epochs[3] that our networks required to reach this criterion are shown in Table 5.2. Note that, as the IM decreases, the number of epochs to reach the criterion increases (though there's little difference in average IM between the 15% and 30% perturbations). The average generalization curves for these networks are also shown in Figure 5.11.

Perturbation Factor	Average IM	Epochs to 79.5%
0%	0.59	5
15%	0.58	7
30%	0.57	28
50%	0.46	58

Table 5.2: Number of epochs to reach criterion for networks with different IM levels
The first row of this table shows the IM and epochs to 79.5% performance for the single network from which the 30 remaining perturbed networks were derived. Remaining rows each show the average IM over 10 networks and the number of epochs for the average curve to reach 79.5%

We also plotted each individual network's average IM vs. the number of epochs to reach the 79.5% level, as shown in Figure 5.12. This plot shows the average IM for each of the individual runs summarized in Table 5.2, along with the corresponding number of epochs that were required for each run to reach a score of at least 79.5%. Points are labelled to indicate whether they represented networks generated by 15%, 30%, or 50% random weight perturbation. Several of the individual networks did not reach a score over 79.5% – these are shown at the top of the graph along the time=150 epochs line.

Figure 5.12 shows that, because of the variances in individual runs for this problem, the relationship between average IM and amount of learning time is not as clear. We would expect to see these numbers clustered towards the bottom right of this plot (high average IM, fast learning), and towards the upper left (low average IM, slow learning). However, the fast runs seem to be somewhat uniformly distributed among different IM levels. More data points would need to be generated for a graph like this one to show a significant correspondence between average IM and number of epochs. This plot shows that the differences between different initial weight sets account for a large amount of variance among runs, which confuses the relationship between average IM and learning time, despite the fact that on average the learning time decreased with rising average IM.

Summary

In this section, we have presented four illustrative studies and one more systematic study that explored the relationship between average IM of a source network and the

[3]Not "normalized epochs" as in Section 4.5.

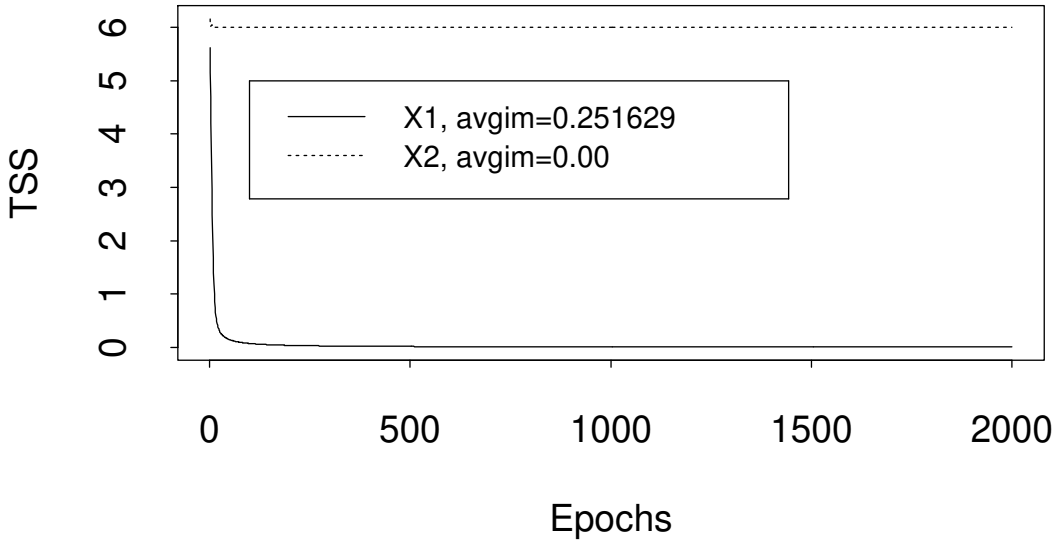


Figure 5.10: Training set curves for the problem in the previous figure showing that, as the IM reduces, so does the learning speed.

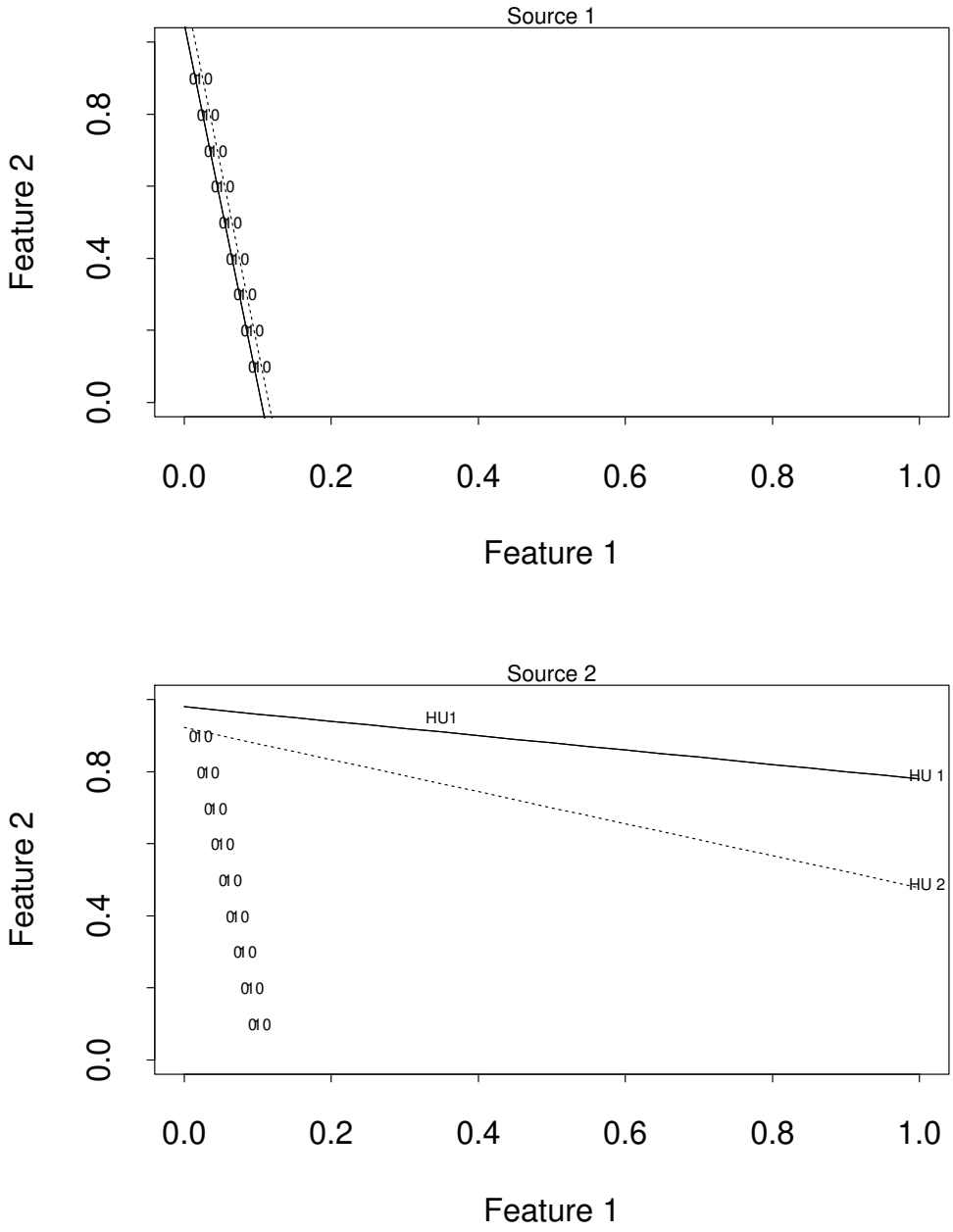


Figure 5.9: A problem which, similar to the previous one, has different feature scaling and hyperplanes that are not axis-parallel.

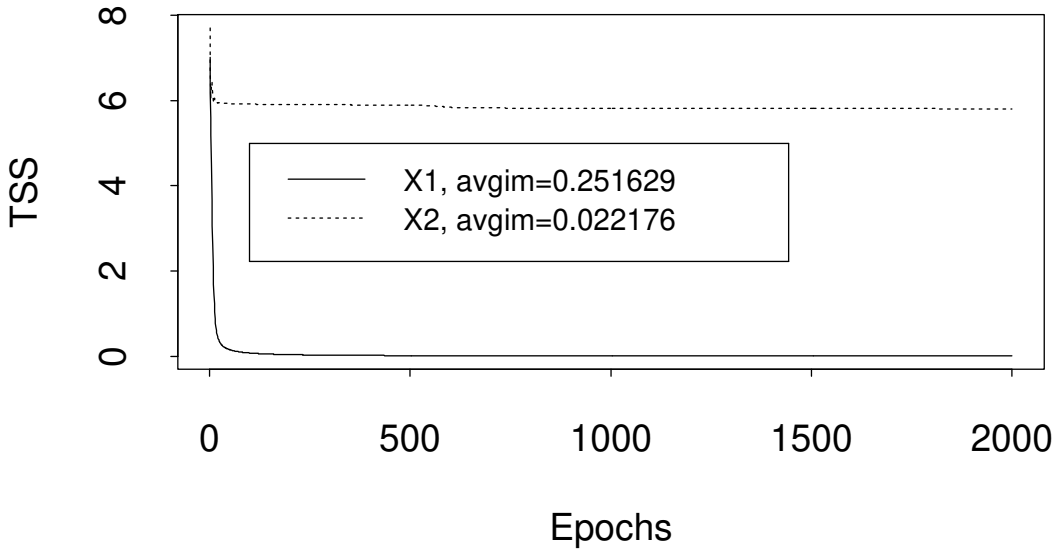


Figure 5.8: Training set curves for the problem in the previous figure showing that lower IM is associated with lower speed of convergence.

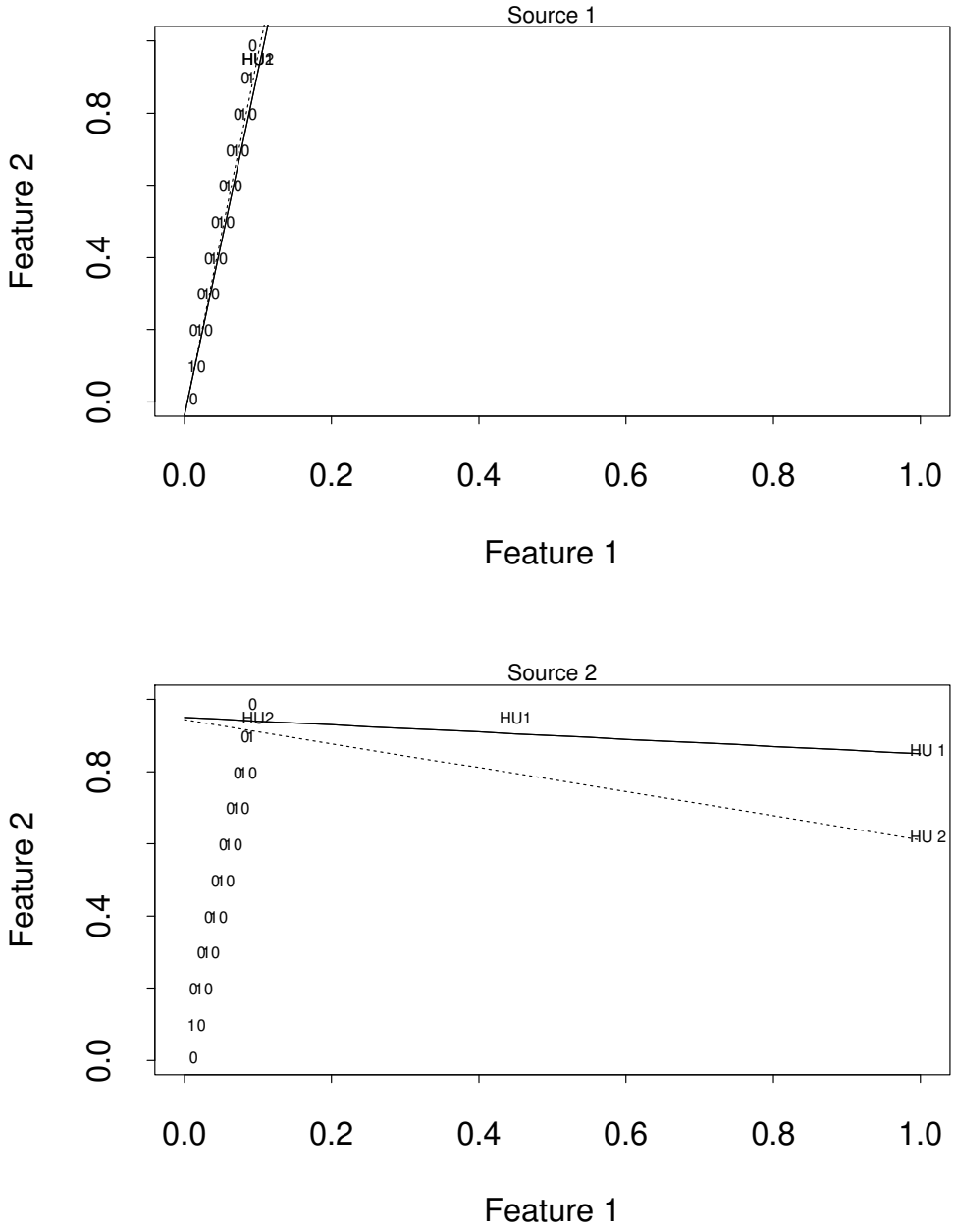


Figure 5.7: Yet another problem on which to measure the relationship between IM and learning speed.
This problem involves hyperplanes that are not axis-parallel, and also demonstrates that transfer in problems where one feature is scaled very differently from the other is effective. In both plots, the training data is the same. In the top image, the initial hyperplanes separate the very closely placed 1's and 0's. In the bottom image, the hyperplanes are not helpful. HU2's hyperplane is shown with a dotted line.

DBT chose the same S and C values for each of these three transfer scenarios. These were: $S = 98.0, C = 0.1$. The resulting TSS curves are shown in Figure 5.6 (b). It shows, again, that low-IM hyperplanes are associated with slower convergence.

Artificially constructed tasks designed to test IM's ability to predict DBT learning speed where hyperplanes are not axis-parallel and data on different dimensions is scaled differently

The studies in the previous two sections were on tasks with axis-parallel hyperplanes. It's reasonable to ask whether the relationship between average IM and learning speed observed there holds up for tasks where the hyperplanes are not axis parallel. Furthermore, it is also interesting to study the behavior of DBT when the scaling factor for different features is different. In the normalized [0.0,1.0] range into which we typically normalize back-propagation inputs (Section 1.1.1), very different scaling factors can translate into a clustering of data points along one of the axes.

We designed two further studies to illustrate the relationship of DBT and average IM in these situations. Figures 5.7 and 5.9 show the training data used for two such problems; Figures 5.8 and 5.10 show the resulting learning curves. For both of these tasks, a 2-2-1 network was used, with automatically chosen η, α, S, and C values. As shown, in these situations, higher IM again corresponds to faster error reduction.

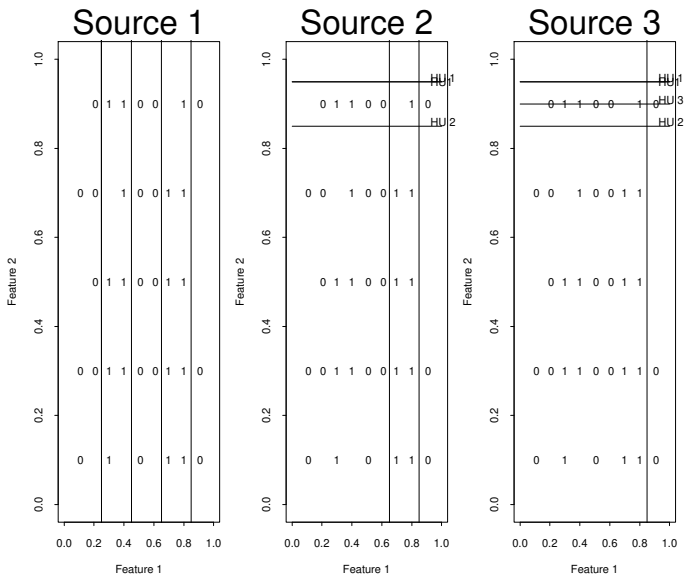
Experiments with systematically altering the "PB-Onemale" problem

The studies presented so far in this section illustrate the relationship between average IM and learning speed improvement obtained via DBT. To more rigorously establish this point would require that random source and target tasks were systematically generated, and that they were then observed to show a correspondence between high average IM and DBT learning speed improvement.

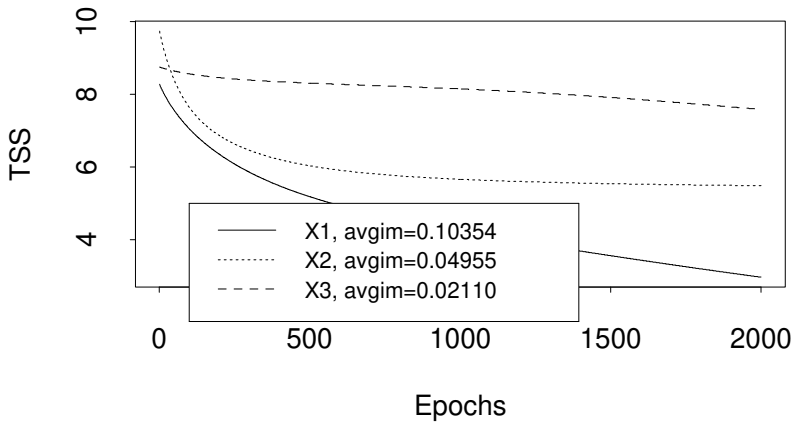
As a step in this direction, we performed a more systematic study on the "PB-Onemale" benchmark task (Section 4.1.3). We randomly perturbed source network weights in this task by varying amounts. This resulted in networks with different IM levels. For these studies, we used source network number 1 of the benchmark study, with the first holdout condition used as a generalization test. We then modified each weight in the source network by setting each weight w to $w + w \times r$, where r was a random number in the range: $[-.15, .15]$. Randomly perturbing the weights in this way resulted in a network that often had a reduced IM. We repeated this procedure for ten different random weight seeds, resulting in ten networks with weights perturbed by up to 15%.

In order to compare the learning speed of networks with different IM values, we repeated this procedure with maximum perturbation amounts of 30% and 50%. We then calculated the average IM of hyperplanes in each set of 10 networks; the result is shown in Table 5.2. As can be seen, the average IM decreased as the perturbation factor was increased.

The period of significance in the benchmark study for the Onemale problem ended at epoch 415 (Table 4.3), at which point the generalization score of the randomly initialized network was 79.5%. So that these results are comparable to the previous ones, we chose



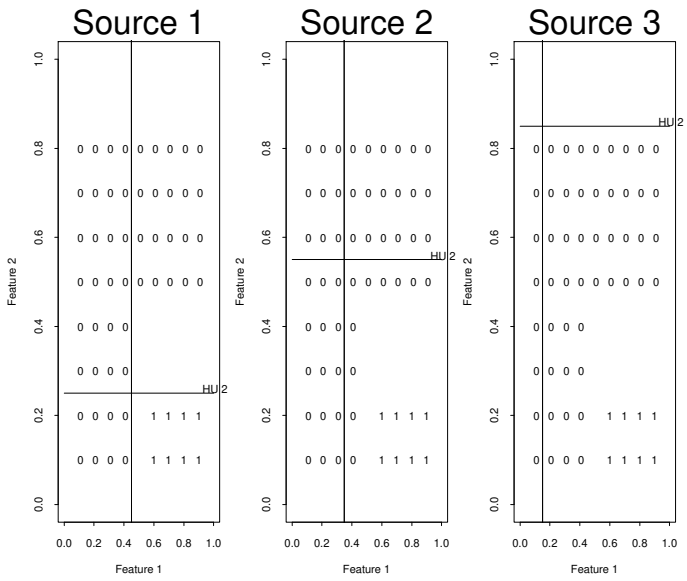
(a)



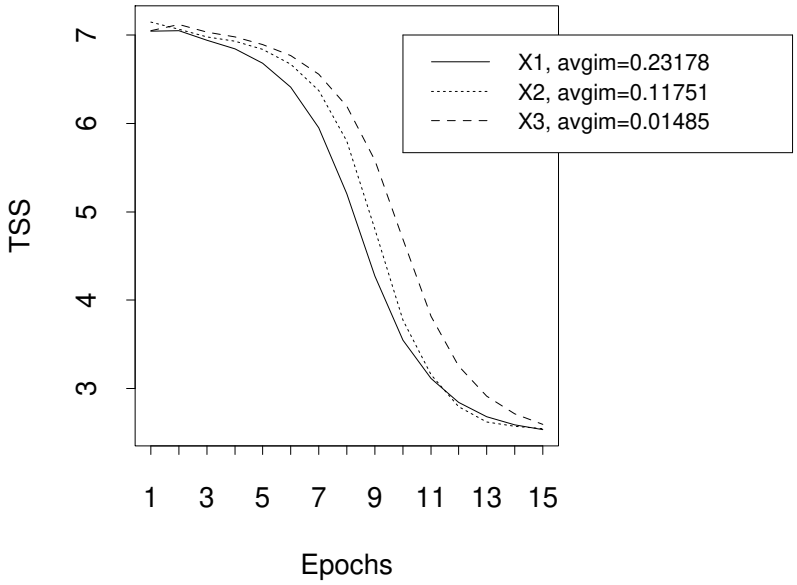
(b)

Figure 5.6: (a) Target training data and source hyperplanes for second illustration of the relationship between IM and learning speed. (b) Training set curves showing that, as the IM decreases, so does the learning speed.

Note in (a), again, that the same target training data is shown in each figure, but that the degree to which the source hyperplanes separate this data decreases in source network 2, and then again in source network 3. (b) shows a more substantial difference in learning speeds based on IM than the previous one, perhaps because the task to be learned is more complex.



(a)



(b)

Figure 5.5: (a) Three different source tasks that were used to test whether lowering the source network IM led to slower training times. (b) Training curves showing how the average IM value related to learning speed.

In (a), the hyperplane corresponding to hidden unit 1 is indicated by the vertical line on each of these plots. Note that the hyperplanes for source network 1 separate the training data completely. Source network 2 hyperplanes only partially separate the data. Source network 3 hyperplanes are only marginally helpful in solving the target task.
In (b), note that, as the average IM is lowered, the networks learn more slowly (TSS drops more slowly).

Several points are worth discussing in relation to these two latter figures. First, note that there is a great deal of vertical striation – average IM values seem clustered around only some points on the X axis. This behavior comes from the fact that, although each task is represented by 100 points, these data were drawn from only 10 source networks, and so only 10 different average IM values are represented. Second, many networks, both random and DBT, didn't reach the indicated criteria. Chapter 4 showed that, on average, DBT networks reached various criteria faster than randomly initialized networks. However note from Table 5.1 that for some tasks there were more random runs that reached a criterion at all. Since DBT was so overwhelmingly faster when it was faster, these few runs were overshadowed when the mean generalization curves were measured. Also, although it appears that there was a high variance in the amount of speed-up to the three criteria, a 1-sided T-test between the amount of learning time for each run (in epochs) and for each criterion indicated that, despite these variances, the populations were significantly different at the 99.0% confidence level, except for the two Heart tasks to the 66% criterion (all runs, DBT and random, reached this criterion after the first epoch) and the DNA task to the 95% and 98% criteria (where no significant difference was found in learning time to these criteria).

Overall, these results indicate a relationship between the average IM and the speed-up factor obtained by using DBT . We now explore it further by studying several artificially constructed tasks that highlight the relationship between IM and DBT learning speed.

First artificially constructed task designed to illustrate IM's ability to predict DBT learning speed

In order to further explore the relationship between average IM and the speed-up factor, we constructed an artificial problem for which the IM of several source networks was varied. The training data used is shown in Figure 5.5 (a), along with the hyperplanes that were formed by hand-coding three source networks. Note that the degree to which the source hyperplanes separate the target data decreases in source network 2, and then again in source network 3.

Networks used for this study had a 2-2-1 architecture, and were trained on 60 target task examples. Our automated procedure chose $\eta = 0.35, \alpha = 0.9$. DBT chose S and C values of $(10, 0.1), (1.0, 1.0), (12.0, 0.0)$ for the three source networks, respectively. The resulting TSS curves for the first few epochs are shown in Figure 5.5 (b). As can be seen, the lower-IM networks converged more slowly.

Second artificially constructed task designed to test IM's ability to predict DBT learning speed

In order to further demonstrate that low IM correlates with slower learning rate after transfer, we constructed a second, more complicated task (requiring more hidden units to solve) with varying-IM source hyperplanes, shown in Figure 5.6 (a). Note again that the degree to which the source hyperplanes separate the target data decreases over the three source networks. Networks used for this study had a 2-2-1 architecture, and were trained on 36 target task examples. Our automated procedure chose $\eta = 0.01, \alpha = 0.35$.

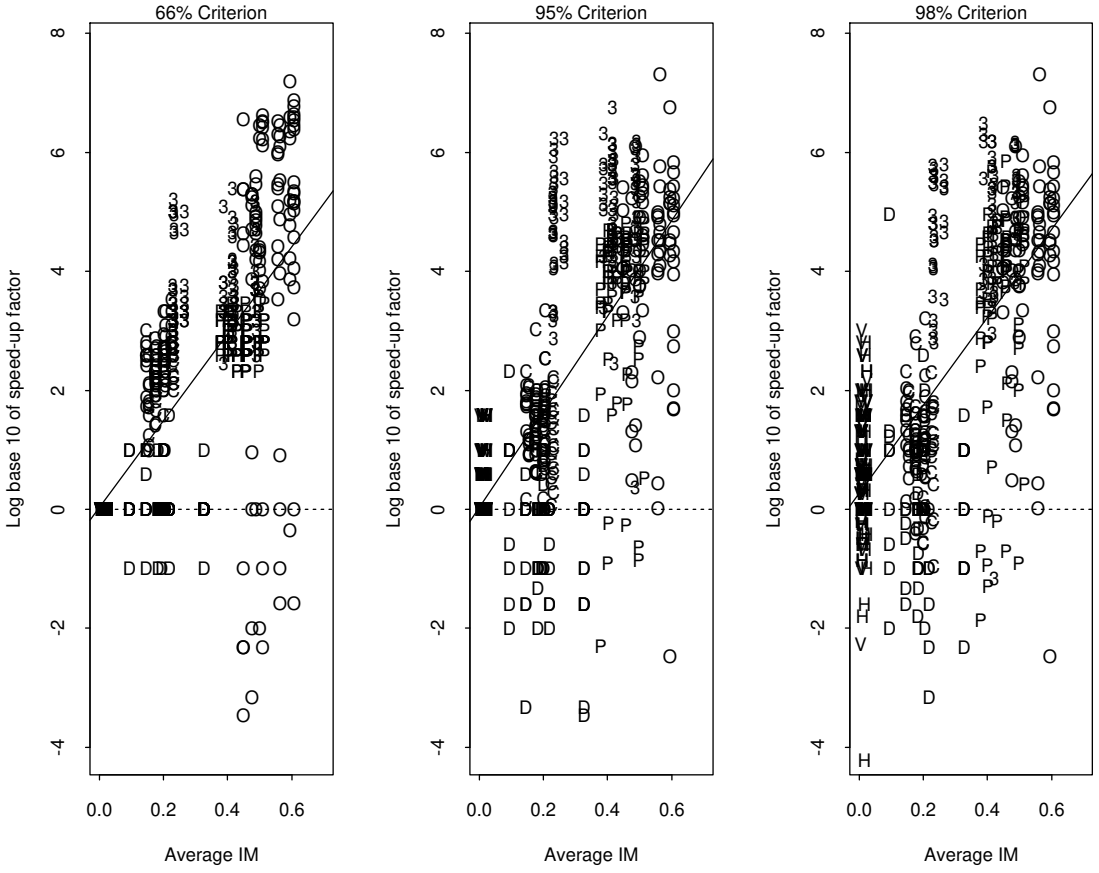


Figure 5.4: Average IM vs. log of speed-up factor observed in the seven benchmark tasks, for 3 different stopping criteria.

This plot shows the same data as the previous figure, but on a semilog scale. A least squares fit line has been shown to indicate the general trend of the data.

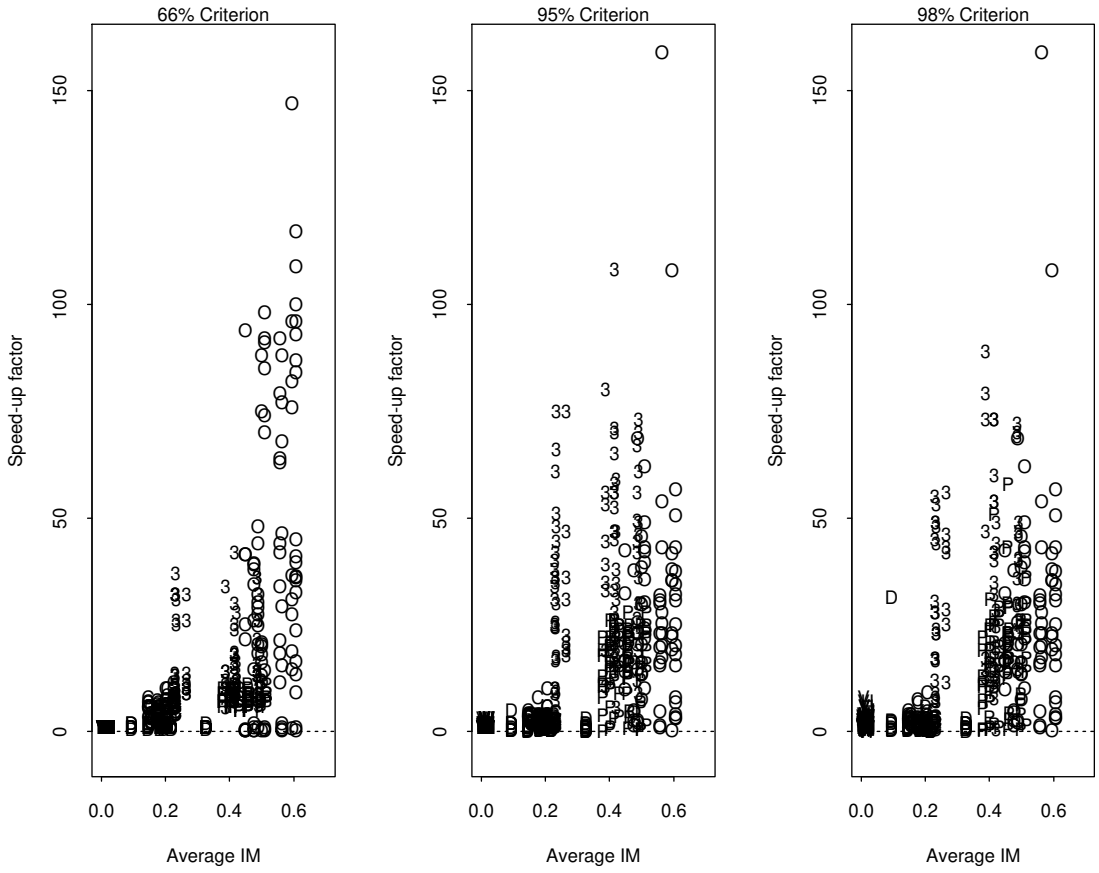


Figure 5.3: Average IM vs. speed-up factor observed in the seven benchmark tasks, for 3 different stopping criteria. This shows all 100 runs for every task.

This figure shows the studies that led to the average time savings indicated in the previous figure. Points are from the seven studies as follows: "O": Onemale, "3": PB123, "D": DNA, "P": PB, "C": Chess, "V": Heart-VAS, "H": Heart-HS. These latter two categories are mostly overlapping, in the bottom left portion of the image. See the following figure for a better view of their positions.

the first random network within each holdout group to the first source network that was used for DBT. This process continued for each pair of initial conditions within the holdout group, and for all the holdout groups, for all tasks. For each pair of randomly initialized and DBT initialized networks so obtained, we measured the amount of time (in epochs) required to reach each of the 66%, 95%, and 98% criteria (as defined with respect to the average curves, as was done in the previous plot). Some DBT and some random runs did not reach some of the criteria, as shown in Table 5.1, which will be discussed in more detail below.

Task	R66	R95	R98	D66	D95	D98
Chess	100	100	90	100	100	91
PB	100	90	90	100	92	**88**
PB123	100	100	90	100	**90**	**74**
Onemale	100	98	98	**99**	**82**	**82**
DNA	100	98	90	100	99	**78**
Heart-HS	100	100	100	100	100	100
Heart-VAS	100	100	100	100	100	100

Table 5.1: Number of runs that reached criterion for randomly initialized and DBT transfer networks.

Columns labelled R66, R95, R98 show the number of randomly initialized networks, out of 100, that reached the 66%, 95%, and 98% criteria, respectively. Columns labelled D66, D95, D98 show the number of networks initialized via DBT, out of 100, that reached the 66%, 95%, and 98% criteria, respectively. Shown in boldface are the criteria and tasks for which fewer DBT runs converged than random runs.

A list of relative times to reach criterion for DBT and randomly initialized networks was collected; whenever either DBT or the corresponding randomly initialized run didn't reach criterion, that pair was omitted from the list.[2] For pairs that were included, the number of random epochs to criterion was divided by the number of DBT epochs, to obtain a "speed-up factor". These results are plotted in the vertical axis in Figure 5.3. The horizontal axis of this figure shows the average IM value for hidden unit hyperplanes in the DBT networks (the horizontal axis reflects nothing about the random network other than the fact that the random network was trained on the same holdout as the DBT network that had that IM value).

Figure 5.4 shows the same information as in Figure 5.3, except that the vertical axis has been plotted on a log base 10 scale, to emphasize data whose speed-up factor was near to 1. A least squares fit line is shown on each plot. The mean distance to this line on this plot was 1.91, 2.42, and 2.52 for the three criteria, respectively, indicating variances on the order of 100's of weight updates.

[2]As can be seen in Table 5.1, this biased the resulting plot. For some problems and criteria, DBT converged less often than randomly initialized networks.

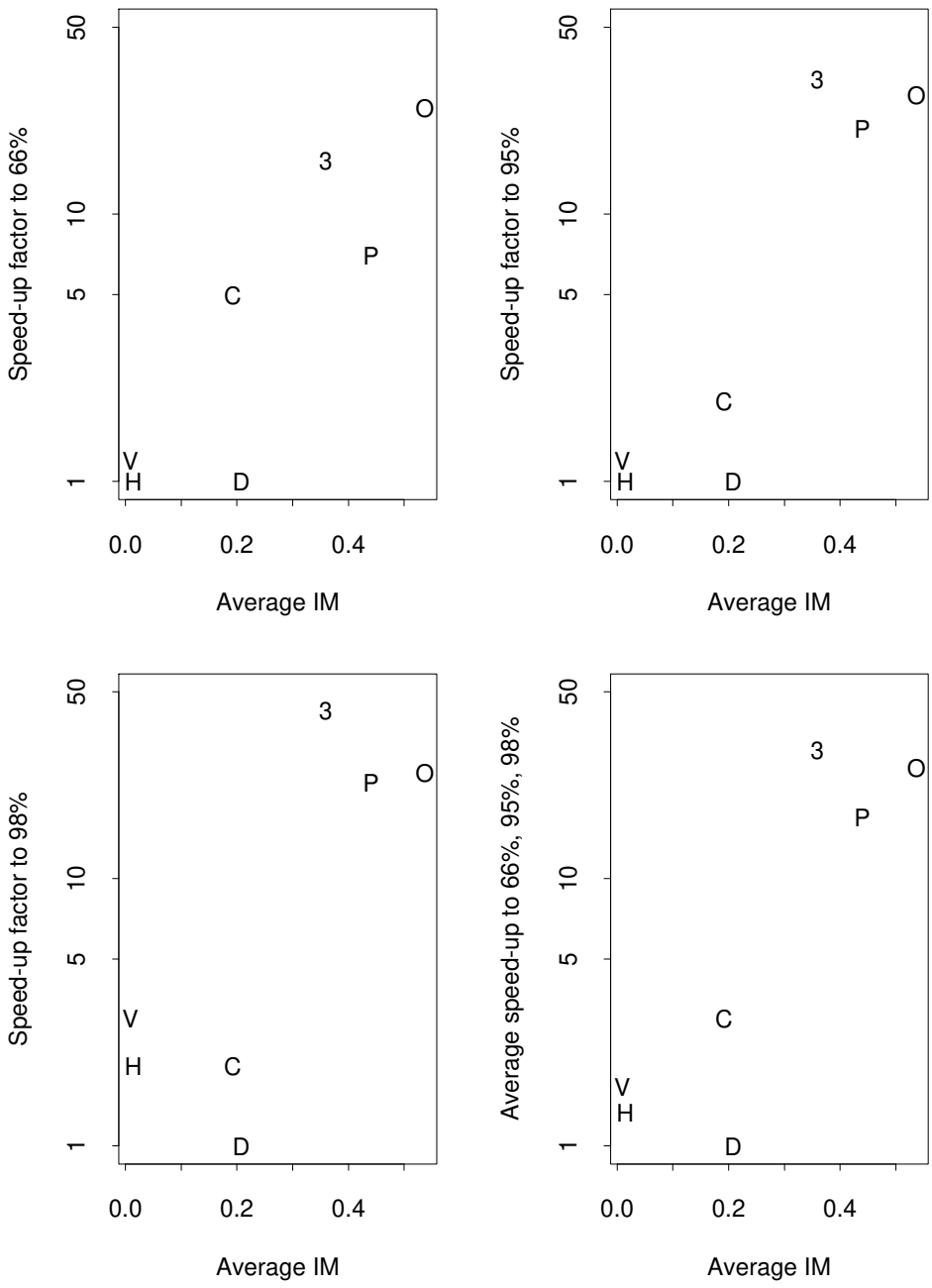


Figure 5.2: Average IM vs. speed-up factor observed in the seven benchmark tasks, for 3 different stopping criteria, along with their average.

Points are from the seven studies as follows: "O": Onemale, "3": PB123, "D": DNA, "P": PB, "C": Chess, "V": Heart-VAS, "H": Heart-HS. These plots show that, roughly speaking, higher average IM values correspond to a larger speed-up factor to criterion. Note that the vertical axis in each plot is logarithmic. Also, to avoid overlapping labels, an offset of 0.02 was added to the speed-up factor for the Heart-VAS task for the 66% and 95% cases.

Formal domain knowledge is often expensive or difficult to obtain, however, and it's not clear how to use domain knowledge to assess task relatedness. As an alternative, we propose, in Section 5.3.1, to use the average IM of trained hyperplanes from the source network on the target data to assess source/target relatedness. We present the relationship between the average IM of the tasks studied in Chapter 4 to the amount of learning speed-up observed, along with a set of more systematic studies that demonstrate how lower IM values correspond to slower learning. The idea of developing more systematic methods for evaluating task relationships for the purpose of predicting the effectiveness of DBT can be viewed as a relaxation of Assumption A of the main procedure of the transfer process, as described in Section 5.2, which is that *Human suspicion of task relatedness can indicate which source networks have approximately positioned or identical hyperplane positions to those needed in the target network.* This section proposes an approach that doesn't rely on the somewhat nebulous process of applying human "suspicion".

Using the average IM is actually only one in a series of increasingly more accurate (and time consuming) methods that can be used to assess task relatedness. We'll describe these techniques in Section 5.3.3.

5.3.1 Using the IM metric to assess source/target relatedness

Since DBT uses the IM of source hyperplanes to assess their relevance to (and hence utility in solving) the target task, it makes sense to use the average source IM as a rough predictor of DBT's "effectiveness" before the full DBT algorithm is run. In this section, we explore the accuracy of this estimate in two ways. First we show how the average IM of the benchmark tasks of Chapter 4 corresponds to the amount of DBT speed-up. Second we present a series of illustrative experiments in which the IM was systematically varied. These experiments indicate that lower average IM's correspond to slower learning. We then describe the limitations of this approach, which explain why the average IM can only produce an approximate estimate of the reduction in learning time to be achieved via DBT. Section 5.3.3 will recommend some procedures that might overcome these problems.

Measuring benchmark task IM

Figure 5.2 shows how the average IM relates to the amount of speed-up observed in the benchmark tasks of Chapter 4. In order to obtain this plot, the IM for all the hidden units in all 100 networks for each of the seven benchmark tasks was averaged. This number was plotted against the speed-up factors in comparison to random initialization that were shown in Table 4.3 in Section 4.6. Separate plots were done for each of the speed-up criteria: 66%, 95%, and 98%. This plot seems to show a rough correspondence between high average IM and an increase in learning speed via DBT.

To obtain more information about the relationship between average IM and the speed-up factor to be obtained via DBT, we compared corresponding runs of DBT and randomly initialized networks. Recall from Chapter 4 that within each group of 10 train/test holdouts, 10 different initial random networks were tested. We matched

Assumptions:

A. The "estimated" target task is a good approximation of the true target task.

Intuition for assumption A: Assumption A is necessary to make the connection between the precondition of this step and the goal: if faster learning is to obtained via use of the weights from an estimated target task, then that estimated task must be a good approximation of the true target task.

Many training and generalization curves that we've observed have changed much faster in earlier epochs than in later ones. This could indicate that the performance in early learning can be used to estimate performance later in learning. Furthermore, the procedure that was described in Section 4.2.2 to determine η and α using estimation based on the first few epochs was very effective in avoiding local minima in randomly initialized networks (as shown by the fact that asymptotic scores differed by no more than a few percentage points).

TSS is a widely used metric for evaluating network performance (see, for example, [McClelland and Rumelhart, 1988]), and is a logical choice for evaluating the approximate network, since it sums the error in each training pattern and each output unit.

5.2.1 Assumptions of generality

Note that the assumptions given above address why our formulation of DBT works for the problems we studied (i.e. leads to at least as fast learning and at least as good performance as random and literal networks overall, and faster learning on several tasks). The long-term success of this research program depends on further assumptions about the applicability of the particular paradigm studied here to a more general situation. For example, we hope that DBT can be extended to work with recently developed back-propagation methods like Conjugate Gradient training. The practical usefulness of this method also depends on the assumption that the results given here hold for larger and more complex problems that take so long to train that learning speed-up is an important issue (as discussed in Section 4.7.2).

5.3 When does DBT work? Quantifying task relatedness

It's important to determine the circumstances under which DBT is effective. A method that takes as input information about the source and target tasks and produces as output an estimate of the speed-up to be gained via DBT would be valuable in assessing whether to apply DBT. Such an estimator might be applied before source training is performed, or after source weights are available. We might automate the process of evaluating task relatedness by using formal domain knowledge about task similarities and differences, as well as information about the performance level of the source network on the source task, if available, to predict DBT's learning speed.

F. Good values for S,C may be in many different parts of the S, C space. The space is continuous enough that a local search will be adequate, starting with a few (4) initial seeds.

G. If the same scaling factor is used for all hyperplanes, then some won't move during further target training (for this reason, a C value is used as a cutoff against which to measure IM values). Hyperplanes with IM values less than C will be given very low random weights.

Method:

A: Begin search in 4 positions of S, C space.

B: Iterate via hill climbing search to find minimum local value of TSS.

Intuition behind these assumptions: These assumptions are all necessary to justify the method that we used to search for S and C, so that it does indeed accomplish the three goals listed above.

The choice of the function to use to determine target weight magnitudes was not motivated by any intuitions about the behavior of transfer or of back-propagation, but was rather developed over time via a series of empirical experiments. For instance, we originally used only the IM of a hyperplane to determine its target magnitude, but found empirically that adding the source magnitude as part of the function to determine this value gave faster target training after transfer. We then found that different problems needed different functions of the IM and source magnitude, so we introduced the multiplier S. We then found that, when scaling using only S, some hyperplanes didn't receive low enough magnitudes, so we added the C cutoff to produce hyperplanes that were very flexible. These assumptions are all based on empirical experimentation on different versions of a transfer algorithm.

1.3 Apply S, C, IM to obtain target magnitudes

Inputs: S, C, IM, source weight magnitudes

Outputs: Target weight magnitudes

Goal: Produce initial weights to be used by transfer method

Preconditions: S, C, IM have been calculated.

Assumptions: (no new assumptions)

2. Post Transfer training

Inputs: Initial weights for transfer training

Outputs: A trained network

Goal: Train the network as quickly or faster than random initialization and literal transfer, to as good or a better performance level.

Preconditions: Initial weights have been set so as to speed up learning on the estimated task.

Method: The method associated with this step involves general back-propagation training. Our methodology for this process was described in Section 4.2.

In particular, it seems unlikely that a particular set of HO weights will be useful without the presence of IH weights to determine the activations of hidden units upon which the HO weights depend (though see [Sharkey and Sharkey, 1992] for some preliminary counterexamples).

1.1 Calculate source IM

Inputs: Source hyperplanes.

Outputs: An IM value for every source hyperplane.

Goal: To obtain a number for each hyperplane that reflects how well it separates the target training data.

Method: See Section 3.2.

Assumptions:

A. IM is an accurate measure of hyperplane separation.

Justification for Assumption A: IM gives a high value for hyperplanes that split the data well, and is well grounded in information theory (Section 3.2.1). Other metrics (entropy, GINI) give similar values to IM, as discussed in Section 1.4.1.

1.2 Hill climbing in scale-up (S), cut-off (C) space

Inputs: Source hyperplane, target patterns, IM

Outputs: S, C values

Goals:

1. Locate a minimum in training data TSS (total sum of squared error) space on the estimated task, by trying various values of S and C in a hill climbing (actually, descending) search.
2. Obtain a scaling factor, S, that will be used in the equation $M_T = M_S \times S \times IM$ to obtain target magnitudes, for hyperplanes with IM values $> C$. This scaling factor should be such that good hyperplanes will be more retentive, and bad hyperplanes should be able to move out of place.
3. Determine C, the IM level below which hyperplanes must be effectively randomized, so that they are free to move into position.

Assumptions:

A. Multiplicative scaling via S will lead to retentive/non-retentive hyperplanes.

B. Hill climbing will lead to good S values.

C. More than one hyperplane will in general need complete loosening.

D. Hill climbing will lead to good C values.

E. The local minimum that's found via this procedure will indicate S and C values that are close to a global minimum (i.e. shape of this space is fairly monotonic).

C. Hidden unit (and IH layer) behavior can be evaluated independently of output unit layer behavior (and HO layer) to give a substantial characterization of the behavior of the entire network.

Intuition for Assumption A (*A hyperplane is a good approximator of a hidden unit's behavior*): Assumption A is necessary to support the goal of this step because the method used here approximates a hidden unit's behavior as a hyperplane. This hyperplane is used in part to determine the weights that help to produce good performance on the estimated target task.

As described in Section 3.1, weight magnitudes often grow during neural network training. High IH weight magnitudes lead to a large magnitude input to each hidden node, which in turn leads to hidden unit activations that are close to 0 or 1. This assumption models feature space as being separated into the regions where hidden unit activations are above 0.5 and below 0.5, for purposes of evaluating hyperplanes for transfer. As weight magnitudes grow, this model is more and more accurate, because hidden units are less likely to have activations near 0.5. Since we evaluate trained source networks for purposes of transfer, the hyperplane assumption is approximately correct for this reason.

We have also observed the relationship between hyperplane position and network output on several small tasks, and it seems that network output contours approximately mimic hyperplane positions. This point was also addressed in Section 1.2.1.

Intuition for Assumption B (*Evaluating each hyperplane individually provides a reasonable estimation of the quality of the decision region that it makes up in a group with other hyperplanes*): Assumption B is necessary to support the goal of this step because the method used here evaluates hyperplanes individually.

We based DBT on this assumption because it allowed a simpler implementation of the algorithm, not because of any intuition that hyperplanes act individually to classify the data (except in linearly separable problems, which tend to train very fast, and so are of less interest here). Extending DBT to measure the region formed by a group of hyperplanes is an important future direction for this work.

Intuition for Assumption C *(Hidden unit (and IH layer) behavior can be evaluated independently of output unit layer behavior (and HO layer) to give a substantial characterization of the behavior of the entire network):* Assumption C is necessary to support the goal of this step because this method generates the weights for the estimated target task based on the input-to-hidden unit weights and not the hidden-to-output layer.

In the two-layer networks we've studied, the hidden layer usually comprises at least half of the weights. Therefore, a substantial amount of the entire network's behavior should be transferable from a source network's hidden layer. Furthermore, since the hidden-to-output layer depends on the input-to-hidden weights, the hidden-to-output layer seems less critical for transfer.

that, in turn, define a function. P is the full function to be implemented by the source task. P' is the target task function. By training on the source training set S, and testing on the source generalization set T, we attempt to ensure that we have constructed a network that implements the full source task P. One view of task relatedness is that the source and target tasks have some I/O mappings in common, as shown by the intersection of P and P'. If our goal in source training is to get a good generalization score on the set T, then the source network is more likely to also be relevant to the target task P' than if, for example, we had only focused on the source training set S.

Assumption A is also based on intuitions gained from examining input-to-hidden layer hyperplanes for related tasks in feature space, and observing that related tasks sometimes share approximate hyperplane positions in common. This can also be seen by examining problems we judge to be related, and seeing that groups of training data of the same class are modified from source to target using transformations in which the same or nearby hyperplanes may be useful for separation (recall Figures 4.1 and 4.2).

Intuition for Assumption B (*Source training is done well enough that the source hyperplanes actually reflect the source/target relationship*): Assumption B supports the goal of achieving faster training through transfer because it's necessary to obtain a set of weights that solves the source network well in order for them to be worthwhile for transfer. Assumption B depends on the fact that the source network classifier reflects the source training data. This can be verified by evaluation of the source classifier on the source training data. This process was described in Section 4.2.

1. DBT Transfer

Inputs: Source weights, target training data

Outputs: Initial weights for further training

Goal: To find initial conditions that give good or as good performance and/or learning speed as random and literal networks (in the sense described in the goal for step 1) on an "estimated" target task (i.e. a target network that's trained for only a few epochs, see Step 1.2, below).

Method:

1.1: **Calculate source IM**

1.2: **Hill climbing in scale-up (S), cut-off (C) space:** For each hyperplane, do hill climbing in S, C space to find reasonable values for these scaling factors.

1.3: **Apply S, C, IM to give initial weights for each hyperplane**

Assumptions:

A. A hyperplane is a good approximator of a hidden unit's behavior.

B. Evaluating each hyperplane individually provides a reasonable estimation of the quality of the decision region that it makes up in a group with other hyperplanes.

A. Human suspicion of task relatedness can indicate which source networks have approximately positioned or identical hyperplane positions to those needed in the target network.

B. Source training is done well enough that the source hyperplanes actually reflect the source/target relationship.

Intuition for assumption A: Assumption A supports the goal of achieving speed-up via transfer because it allows for the possibility of a user being able to identify a task relationship that facilitates transfer. If it was impossible to find related tasks, only part of the goal (where an unrelated source task leads to no learning speed degradation) could be achieved.

For an example of the kind of reasoning that was used to choose related tasks, consider transfer from female to male speakers. Since we can learn a word spoken by a female, and recognize that same word when spoken by a male, it seems that, for purposes of recognition, we are able to ignore differences in pitch. Therefore, decision regions in feature space for the two tasks should be similar along the dimensions that matter to speech recognition, though they may be different along irrelevant dimensions. Hyperplanes that are correctly positioned according to some features and not others should be able to move into place during target network training.

If training on the source task results in good source generalization performance, and the target task is related to the source task, then parts of the source network will be useful on the target task training data. Following the model of [Sharkey and Sharkey, 1992], we can use Venn diagrams to illustrate this assumption, as shown in Figure 5.1. [1]

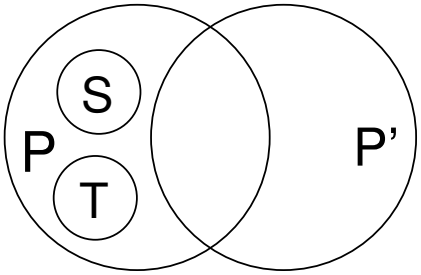


Figure 5.1: Venn diagrams illustrating why obtaining good source task generalization should help with target task generalization.

See text for discussion.

In this figure, each circle represents a set of <input,desired output> pairs

[1] Note that this model of transfer is somewhat simplified in that it indicates that the only way that two tasks P and P' can be related is if they have I/O pairs in common. DBT transfer is also able to take advantage of situations where, although the source and target tasks do not have I/O in common, some generalization of the source does have I/O in common with the target. This generalization is to other source networks with a subset of hyperplanes in the same or nearby positions. Note also that there is no reason that S and T can't overlap P' – it was arbitrarily chosen in Figure 5.1 to show them as not overlapping.

5.2 Assumptions and intuitions behind DBT: Why does it work?

We now present a hierarchical decomposition of the steps used in training a network using DBT. The whole procedure is called "Neural network training, with transfer from a source network", which contains the substeps of "DBT transfer" and "Post-transfer training". This decomposition continues in this manner, until all the steps of the algorithm are described. This formulation facilitates explication of the assumptions associated with each step. These are listed, along with the inputs, outputs, goals, preconditions, and assumptions associated with each stage of DBT. We also describe the intuitions and justifications on which the assumptions are based.

A note on terminology: We have used the word "assumption" in this section to mean the rationale underlying any step in the DBT algorithm that might benefit from further validation, relaxation, or might be used elsewhere, as described above in Section 5.1. Some assumptions are based on our intuition, some are based on stronger evidence, which we've labelled as "justifications". This distinction between intuitions and justifications is somewhat artificial, however, since the amount of evidence gathered to support an "assumption" varies over a continuum. Also, it can be argued that any assumption that's given a "justification" should no longer be called an assumption. Since the point of this section is not to make these sorts of distinctions, but rather to elucidate aspects of the DBT algorithm, we haven't bothered to develop and use a more precise hierarchy of terms to describe the various evidence levels gathered for each aspect of DBT.

Neural network training, with transfer from a source task

Inputs: Source weights, target training data

Outputs: Trained target network

Goal: If, despite the user's suspicion (see below), the source task has no relationship to the target task, then the goal of this method is to train a neural network no slower than a randomly or literal initialized network, to a performance level whose expected value is not significantly worse. Otherwise, the goal is to train a target network faster than would be required if it was initialized with random weights or by literal transfer, and to an expected performance level that is not worse than that of a randomly initialized network or of a network initialized via literal transfer.

Preconditions: The user suspects that the source weights that are input to this step solve a task that is related to the target.

Method:

1. **DBT transfer:** Find initial conditions based on an estimated target task, using source network weights. Do this step using the DBT transfer method.
2. **Post-transfer training:** Train a back-propagation network using these initial conditions.

Assumptions:

Chapter 5
Why and when does DBT work?

5.1 Introduction

In Chapter 3 we motivated the DBT algorithm by showing how it overcomes the shortcomings of literal transfer. DBT was described in detail, and then Chapter 4 presented empirical experiments and their results in support of the claim that DBT speeds up learning.

Given the failure of literal transfer to function reliably, however, there were many non-literal transfer schemes that could in principle have been developed, besides DBT. In Section 5.2, we'll explore the assumptions that led to DBT as a non-literal transfer method. There are at least three reasons for doing so. First, by calling these properties of neural network training "assumptions", we are saying that we have not formally validated them. By making them explicit in Section 5.2, we hope to facilitate the extension of DBT – each assumption can now be systematically verified (or perhaps even disproven!). Such studies would give insights into the sources of DBT's power, addressing the question of "why" DBT works.

Second, DBT might also be extended by relaxing one or more assumptions. For example, we assume that a hidden unit's behavior can be characterized in terms of the hyperplane along which its activation crosses 0.5. If we relaxed this assumption, and characterized hidden units in terms of a more complex model, DBT might be improved. In this way, this list of assumptions can be viewed as an "idea generator" for follow-on projects.

Finally, and perhaps most importantly, making these assumptions explicit allows them to be used in methods other than DBT. For instance, we have used the hyperplane simplification described above, along with the idea of measuring a hyperplane's information measure IM (as described below), for network skeletonisation [Ramachandran and Pratt, 1992]. The success of DBT, which depends intimately on the validity of these assumptions, constitutes indirect evidence for their usefulness in more general settings.

As was described in Sections 1.4.7 and 4.1, we chose the transfer tasks of Chapter 4 because, based on our knowledge of the domains from which they were drawn, it appeared that the source and target problems were related enough that the source weights should help to improve learning speed on the target task. In a practical situation, obtaining a source network might be expensive, so it might be helpful to develop more formal methods for assessing source/target task relevance, in order to estimate the amount of learning time speed-up to be achieved via DBT. This question of "when" (for what task relationships) DBT works will be addressed in Section 5.3.

applied to only male patients over age 50. It seems that there are many transfer situations in which a network trained on a super-population might be helpful in learning a sub-population.

Some source networks are better than others

DBT demonstrated 194 epochs of superior average performance on the Heart-HS task, but only 5 epochs of average superior performance on the Heart-VAS task. Both these tasks had the same target set of Swiss patients. This suggests that, in general, some source networks may be better for transfer than others. We'll return to this point in Section 5.3.1, where we'll discuss how we might evaluate the utility of a source network for a particular target task.

4.9 Conclusion

This chapter has presented empirical results of applying DBT, literal transfer, and random initialization to each of seven benchmark tasks. We've shown how the results of these experiments support the claims of this thesis. Chapter 5 now presents an analysis of the DBT method.

The speed-up factors shown above (by which, for some criteria, literal is actually faster than DBT) were achieved at the expense of the *robustness* of the method. We consider the cost of using a method that's as brittle as literal transfer, where on several tasks it achieved an inferior asymptotic performance score to a randomly initialized network, to be too high to justify its use.

4.8.2 Insights gained from these results

The results on these seven benchmark tasks are suggestive of the situations in which DBT transfer, or perhaps transfer in general, might be most helpful.

DBT helps more when there is room for improvement

DBT demonstrated the least learning speed improvement in cases where a random network already did well. Learning to the asymptotic level was fast (under 20 epochs) for randomly initialized networks on the DNA task. Since this task was linearly separable, only one hyperplane needed to be positioned, which the random networks were able to do easily. DBT networks weren't able to improve much on this already very fast learning speed.

Another way that random networks can do well is to produce a high asymptotic score. This was the case for the Heart-HS and Heart-VAS tasks. In these cases, even when DBT gave significantly superior results, the amount of difference was a small percentage of the random asymptotic score.

These results suggest that, in general, when there is less "room for improvement", DBT is less helpful. Room for improvement can be either in terms of epochs of speed-up that are possible, or amount of performance improvement that can potentially be made. It seems reasonable to expect that any transfer method, not just DBT, can be more helpful in situations in which learning is slow or achieves a low asymptotic score. The failure of DBT to help much for these tasks is partially a result of our choice of unrealistic tasks upon which to test DBT. To benchmark this algorithm, we didn't choose tasks that have demonstrated notoriously long training times. We suspect that, in situations where training sets are large and networks have many nodes, DBT will be more helpful.

DBT does well when a large amount of information is brought to bear on a small task

DBT did best on the Onemale task, where a network for a complex problem (vowel recognition from several female speakers) was used to initialize a network for a simpler related problem (vowel recognition from a single male speaker). In this situation, it's possible that the parts of the more general network were extracted and specialized for this task. It seems that this process might apply in general to any situation where a network is trained for a complex task and then transferred to a related simpler task. Perhaps a speaker-independent recognition network could be used via DBT to initialize a speaker-dependent network. Or a general heart disease diagnosis network could be

were randomly initialized networks on average significantly superior to DBT (though individual runs were sometimes inferior). On the one task (DNA) in which DBT did not produce epochs of superior performance, DBT and randomly initialized networks gave results that weren't significantly different.

Secondly, DBT didn't achieve learning speed improvement at the expense of another important criterion: asymptotic performance. In all seven tasks studied, DBT networks did not display statistically inferior performance to randomly initialized networks at the end of training.

In support of the additional claim, we have evaluated a simpler transfer method – literal transfer – using the same tasks and methodology as for DBT. DBT is superior to literal transfer because literal transfer produced significantly inferior asymptotic scores to random initialization on some tasks (DBT did not). Literal networks also displayed slightly slower learning on some epochs in some tasks (another problem that DBT didn't display). In particular, for the PB-onemale, Heart-HS and DNA tasks, literal networks gave significantly worse performance asymptotically; for PB and DNA the asymptotic score was lower, but not significantly so. Literal networks also produced fewer epochs for which they were statistically significantly superior. In particular, the number of epochs of statistically significant literal superiority over randomly initialized networks, averaged over our 7 benchmark tasks, was 108, compared to 134 for DBT vs. random.

Normalized curve shapes were used as a further criterion to characterize the speed with which DBT, literal, and randomly initialized networks reached their last epoch of significant difference. We found that, in DBT comparisons to randomly initialized networks:

- DBT networks converged up to 25 times faster to the 66% level than random nets.
- DBT networks converged up to 32 times faster to the 95% level than random nets.
- DBT networks converged up to 43 times faster to the 98% level than random nets.

In comparisons between literal and random networks, we found that:

- Literal networks converged up to 59 times faster to the 66% level than random networks.
- At the 95% level, one literal network never reached this point; the remaining literal networks were up to 59 times faster than random networks to this criterion.
- At the 98% level, 2 literal networks never reached this point; the remaining literal networks were up to 77 times faster to reach this point than random networks.

Overall, DBT curves not only displayed a number of significantly superior epochs over random networks during learning, but the amount of significant difference in performance during those epochs was often considerable. Literal networks also demonstrated some learning speed improvement (for some criteria and problems literal networks were two or even three times faster than randomly initialized networks), but this result is overshadowed by the fact that literal transfer was not reliable asymptotically. We consider this latter point to be a critical flaw with literal transfer in comparison to DBT.

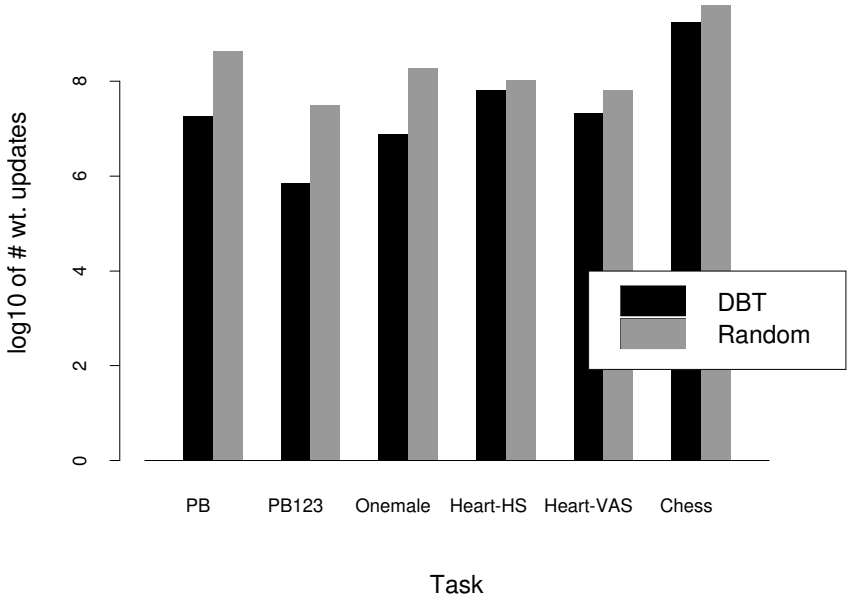


Figure 4.13: Summary of DBT and random initialization time to 98% criterion

These values were obtained by weighting the number of epochs in the columns for DBT and random initialization under the 99% criterion in Table 4.3 by the number of weight updates per epoch. Note that the y axis is logarithmic, so, for example, over 30 million weight updates were saved by using DBT instead of random initialization in the PB123 problem to reach this criterion. We have omitted the DNA task from this graph, since there was no significant difference in its learning curves.

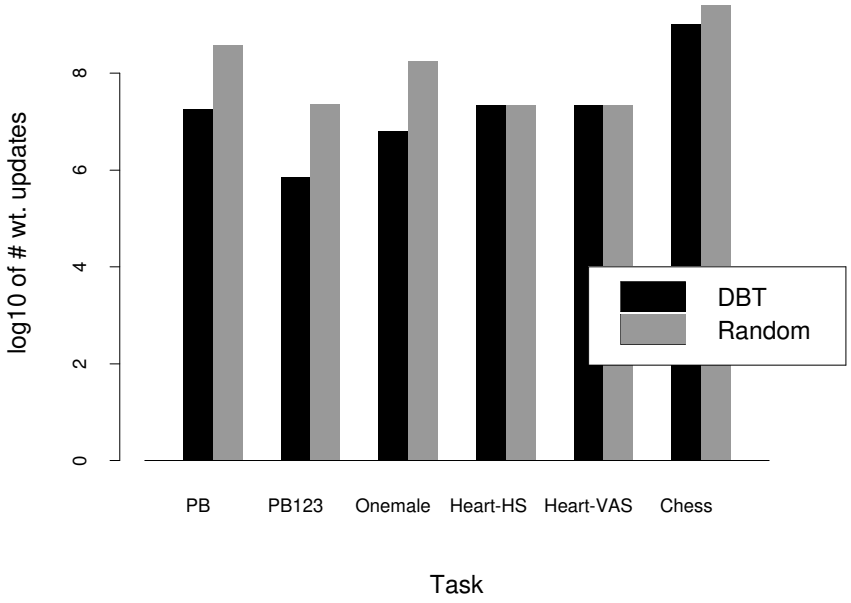


Figure 4.12: Summary of DBT and random initialization time to 95% criterion

These values were obtained by weighting the number of epochs in the DBT and random columns for the 95% criterion in Table 4.3 by the number of weight updates per epoch. Note that the y axis is logarithmic, so, for example, over three million weight updates were saved by using DBT instead of random initialization in the PB problem to reach this criterion. We have omitted the DNA task from this graph, since there was no significant difference in its learning curves.

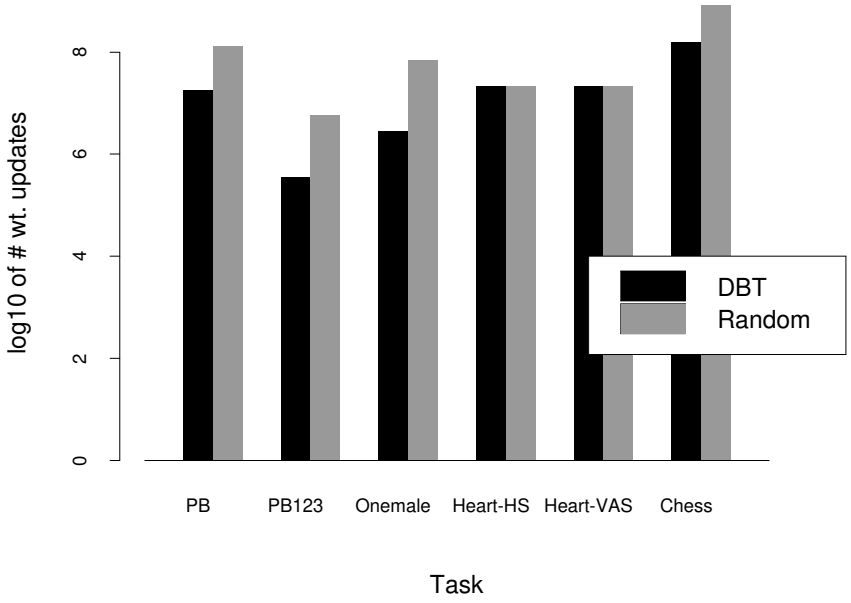


Figure 4.11: Summary of DBT and random initialization time to 66% criterion

These values were obtained by weighting the number of epochs in the DBT and random columns for the 66% convergence criterion in Table 4.3 by the number of weight updates per epoch. Note that the y axis is logarithmic, so, for example, over six million weight updates were saved by using DBT instead of random initialization in the Chess problem to reach this criterion. We have omitted the DNA task from this graph, since there was no significant difference in its learning curves.

may be effective on much larger tasks, in which improvement in learning speed may indeed be important.

When literal networks were compared to randomly initialized networks, they were also superior for some epochs, but were also inferior for some tasks for some epochs.

4.7.3 Shape of normalized curve

The column "Eps of nrm crve" in Table 4.3 shows the number of epochs used to generate the "normalized" curve shown in the second column of Figures 4.4 through 4.7, from which measures reflecting curve shape during epochs of significance (to be shown in Tables 4.3 (b) and 4.4 (b)) were derived. As described in Section 4.3, the number in this column was determined by taking all epochs starting at epoch 1, up to and including the last epoch for which a significant difference was detected. Discrepancies between values in this and the previous column represent "false convergence" – where the test condition (DBT or literal) and random networks seemed to have reached a point of no significant difference, but they later diverged again.

Columns to the right of the fourth give the percentage of normalized epochs required for DBT and random networks to reach different criteria, along with the ratio of these times. Each criterion is expressed in terms of percent of highest random score at the end of the normalized epochs. The speed-up factors to different criteria (66%, 95%, 98%) on the normalized curves are shown. Ratios are rounded to the nearest whole number. The 66% criterion is the point at which the curve would begin to flatten out if was exponentially shaped, as is often hypothesized for natural learning systems (cf. [Ratcliff, 1990]). Figure 4.11 summarizes the results for this criterion for DBT vs. random initialization. The 95% criterion shows the percent convergence time differences near to the top performance score. Figure 4.12 summarizes the results for this criterion on DBT vs. randomly initialized networks. Note that it shows no speedup to the 95% level for the Heart-HS and Heart-VAS for DBT tasks. The 98% criterion addresses this problem – at this level the DBT curves for the Heart-HS and Heart-VAS tasks show speedup compared to randomly initialized networks. Figure 4.13 summarizes the results for this criterion.

4.8 Discussion

4.8.1 Support for the thesis argument

The empirical results presented in this chapter support the primary thesis claim of this dissertation: that DBT demonstrates a learning speed improvement compared to randomly initialized networks. These results also support the additional claim that networks initialized via literal transfer demonstrated worse asymptotic performance than randomly initialized networks on some tasks. The evidence in support of these claims is summarized below.

First, for six of the seven benchmark tasks studied, DBT demonstrated some number of epochs of significantly superior performance. In no epoch in any of the seven tasks

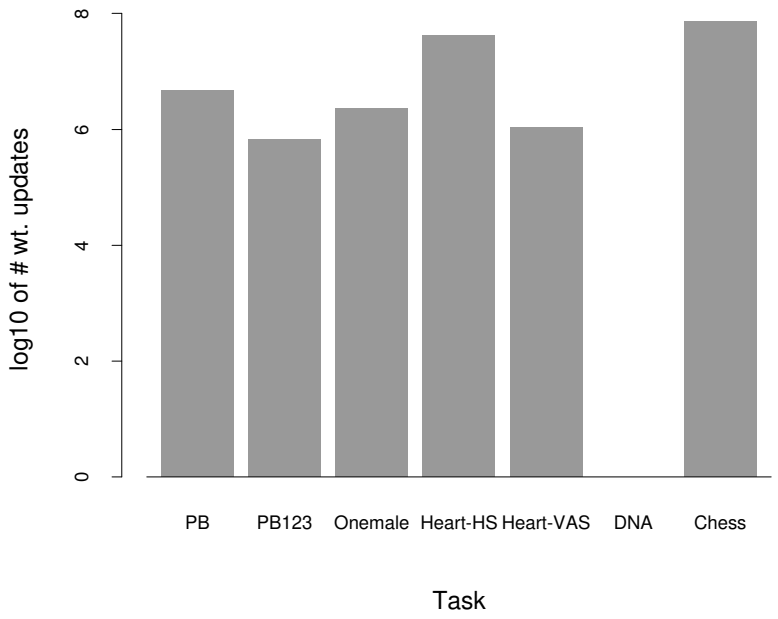


Figure 4.10: Summary of amount of time in period of DBT statistically superior performance over random initialization.

These values were obtained by weighting the number of epochs in column 3 of Table 4.3 (a) by the number of weight updates per epoch. Note that the y axis is logarithmic.

number of epochs of statistically significant difference, and the speed within that period that different methods take to reach different criteria.

4.6.2 Comparison to common baseline of random initialization

Table 4.3 shows the results of comparing DBT to randomly initialized networks. Table 4.4 compares literally initialized to randomly initialized networks. We chose to make these comparisons, instead of comparing DBT directly to literally initialized networks, so that the results in the two tables were directly comparable. Also, future transfer methods can be compared to the same baseline of randomly initialized networks so that inter-method comparisons are feasible. As discussed in Section 4.3, we also designed our measurement scheme to facilitate intra-method as well as inter-task comparison of learning speed.

4.7 Specific comments about how to read results tables

4.7.1 Asymptotic performance

The first two columns of both tables show the final performance of each condition (after the number of epochs determined as described in Section 4.2.4). The 99.0% confidence interval is also shown. Note that, in the DBT-to-random comparison, the final performance score was never significantly different. However, literal networks did give asymptotically worse performance than randomly initialized networks on some tasks (significantly so on PB-onemale, Heart-HS; not significantly on PB and DNA).

4.7.2 Number of statistically superior performance epochs

The column "Eps sig. DBT sp'y" in Table 4.3 shows for how many epochs DBT was found to be statistically significantly superior to a random network, with 99.0% confidence. This column reflects the central empirical result of this dissertation: **when compared to randomly initialized networks, DBT networks often demonstrated statistically superior performance.** In particular, for all but the DNA task, this period lasted for more than 10,000 weight updates. These results are summarized in Figure 4.10.

As an example of the amount of real time that is represented by the periods of significance in Figure 4.10, it's possible to demonstrate that the Chess network requires approximately 5 arithmetic operations per weight update. A typical high-end workstation operates at between 10 and 30 MIPS (million operations per second). Figure 4.10 shows over 73 million weight updates that differ significantly in performance for the Chess problem. This represents $73,000,000 * 5/30,000,000 = 12.16$ seconds for this problem on a 30 MIP workstation. Note that this amount of time savings isn't really adequate to justify the use of transfer in a practical setting like those described in Section 4.6.1; rather this and the remaining benchmark tasks indicate that this method

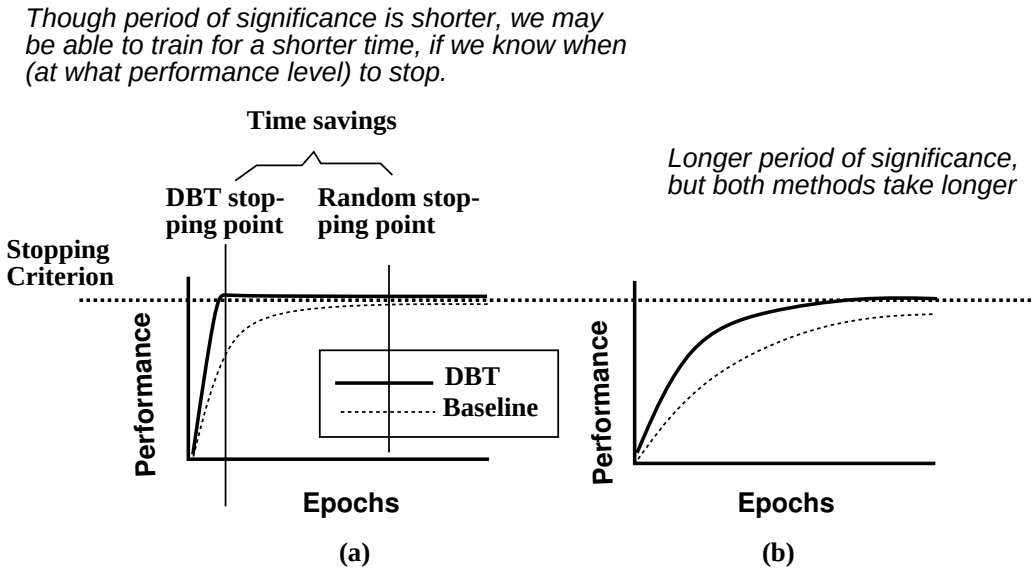


Figure 4.9: Illustration that if we know when to stop, length of significant period is less important.

If we are able to detect when a network has trained to criterion, then we may be able to stop training when it reaches that criterion, thus saving training time. In this case, the length of the period of statistically superior performance is less important than the relative amounts of time required to reach criterion by DBT and the baseline network.

4.6.1 Scenarios for evaluation

There are at least two different practical situations in which we may want to speed up a learning process. First, consider the situation where we have a limited amount of computer time, and that we will use all of this time because we have no way of detecting when a network's performance has reached some criterion and is therefore adequate for some applied task. In this situation, if our speed-up method (i.e. DBT) is significantly superior to a baseline (i.e. a randomly initialized network) for a large proportion of epochs during training, then the probability that we'll have to stop during that period of significant superiority is higher than if the period of significance had been shorter.

If we do stop at an epoch when our method is significantly better, then this justifies our method over the baseline, because the resulting network has better performance. This kind of comparison is illustrated in Figure 4.8. If the amount that DBT is superior to the baseline method during the significant period is so small as not to be relevant in a particular situation, however, then the length of the significant period is less important. We're assuming here that any overall performance improvement obtained via DBT makes it worthwhile, even if this improvement is small.

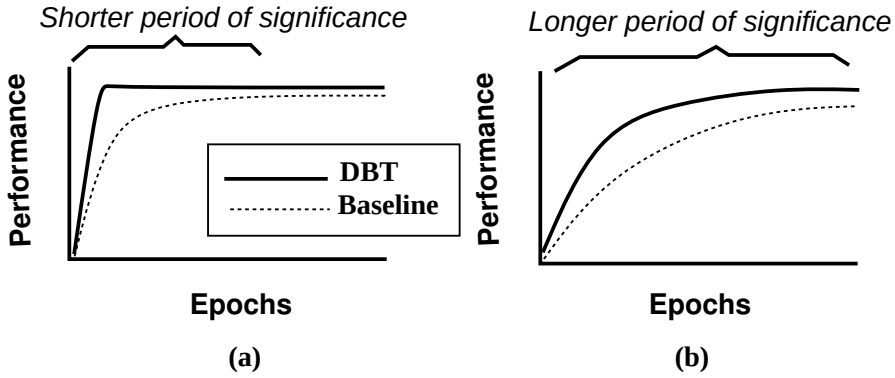


Figure 4.8: Illustration of length of periods of significant superiority between a speed-up method and a baseline.

These curves are relevant to the situation where we don't have any other criterion for when to stop training than that we've run out of training time. In this case the use of DBT is more justified when DBT is significantly better than the baseline method for a larger number of epochs, as in (b).

A second way to evaluate DBT (or any speed-up method) is when we have some way of detecting when performance is "good enough" for the application. This may occur, for example, in problems in which performance reaches close to 100%, and so we may know, *a priori* that we can stop when our network score reaches, say, 99%. In contrast to the above situation, a DBT network may be run for a shorter time than a baseline network, because it reaches this performance criterion faster. In this case, the number of epochs for which DBT is statistically significantly superior is less important than the speed with which it achieves our criterion. Figure 4.9 illustrates this scenario.

Both of these scenarios are addressed in the tables above, which report both the

Task	Literal Last Perf	Rand Last Perf	Eps of lsig /rsig	Eps of nrm crve
PB	.84±0.01	.86±0.01	24/0	24
PB123	.90±0.02	.90±0.02	107/0	116
PB-onemale	.80±0.06	.95±0.00	254/516*	1000*
Heart-VAS	.99±0.00	.99±0.00	2/531†	700†
Heart-HS	.97±0.00	.99±0.00	0/398†	400†
DNA	.87±0.03	.89±0.02	0/11†	26†
Chess	.98±0.01	.97±0.01	367/0	373
Mean			108/208	

(a)

	% Conv. time to 66% of random			% Conv. time to 95% of random			% Conv. time to 98% of random		
Task	DBT	Rand	ration	DBT	Rand	ration	DBT	Rand	ration
PB	4.17	25.00	1:6	8.33	83.33	1:10	8.33	91.67	1:11
PB123	0.86	12.93	1:15	0.86	50.86	1:59	0.86	66.38	1:77
PB-onemale	0.30	17.60	1:59	∞	53.70	∞:1	∞	64.90	∞:1
Heart-VAS	0.14	0.14	1:1	0.14	0.14	1:1	3.00	1.00	3:1
Heart-HS	0.25	0.25	1:1	0.25	0.25	1:1	34.50	1.50	23:1
DNA	3.85	3.85	1:1	42.31	19.23	2:1	∞	23.08	∞:1
Chess	0.27	4.29	1:16	4.56	14.48	1:3	7.77	27.35	1:4

(b)

Table 4.4: Results of literal comparisons to randomly initialized networks.
The second two columns of Table (a) show the final mean performance level of each condition, along with the 99.0% confidence interval. "Eps of lsig/rsig' shows the number of epochs for which the literal mean generalization curve was significantly better than the random curve, followed by the number of epochs that the random curve was significantly better than the literal curve. "Eps of nrm curve" shows the number of epochs in the "normalized" curve, which is taken to be all epochs up to the last epoch of statistically significant difference. A "†" indicates that the literal network was significantly asymptotically inferior to the random network; remaining entries in that row and in the corresponding row of Table (b) were for the whole curve. The "*" indicates that, for the PB-onemale task, curves crossed during learning, from literal being superior to random being superior. Remaining measures in (a) and (b) for PB-Onemale are over entire curve. ∞ entries in this and other rows of this table indicate that the literal network never reached the indicated criterion. Table (b) shows the convergence time in epochs for DBT and randomly initialized networks to reach the 66%, 95%, and 98% criteria, along with the ratio of the times for the two methods.

Task	DBT Last Perf	Rand Last Perf	Eps sig. DBT sp'y	Eps of nrm crve
PB	.85±0.01	.86±0.01	26	26
PB123	.91±0.02	.90±0.02	185	185
PB-onemale	.89±0.06	.95±0.00	391	415
Heart-HS	.99±0.00	.99±0.00	194	254
Heart-VAS	.99±0.00	.99±0.00	5	6
DNA	.89±0.02	.89±0.02	0*	0*
Chess	.98±0.01	.97±0.01	140	140
Mean			134	

(a)

	% Conv. time to 66% of random			% Conv. time to 95% of random			% Conv. time to 98% of random		
Task	DBT	Rand	ration	DBT	Rand	ration	DBT	Rand	ration
PB	3.85	26.92	1:7	3.85	80.77	1:21	3.85	88.46	1:23
PB123	0.54	8.65	1:16	1.08	34.05	1:32	1.08	45.95	1:43
PB-onemale	1.20	29.40	1:25	2.65	74.22	1:28	3.13	77.11	1:25
Heart-VAS	0.39	0.39	1:1	0.39	0.39	1:1	1.18	1.97	1:2
Heart-HS	16.67	16.67	1:1	16.67	16.67	1:1	16.67	50.00	1:3
DNA	1.00	1.00	1:1	5.00	4.00	1.3:1	8.00	6.00	1.3:1
Chess	2.14	11.43	1:5	14.29	35.00	1:2	23.57	55.71	1:2

(b)

Table 4.3: Results of DBT comparisons to randomly initialized networks
The second two columns of Table (a) show the final mean performance level of each condition, along with the 99.0% confidence interval. "Eps sig. DBT sp'y" shows the number of epochs that DBT was found to be statistically significantly superior to randomly initialized networks. "Eps of nrm curve" shows the number of epochs in the "normalized" curve, which is taken to be all epochs up to the last epoch of statistically significant difference. The "*" in Table (a) indicates that there were no epochs of significant difference between DBT and random for the DNA task. Remaining entries in the row and in the corresponding row of Table (b) are for the entire curve, though there was no significance there. Table (b) shows the convergence time in epochs for DBT and randomly initialized networks to reach the 66%, 95%, and 98% criteria, along with the ratio of the times for the two methods.

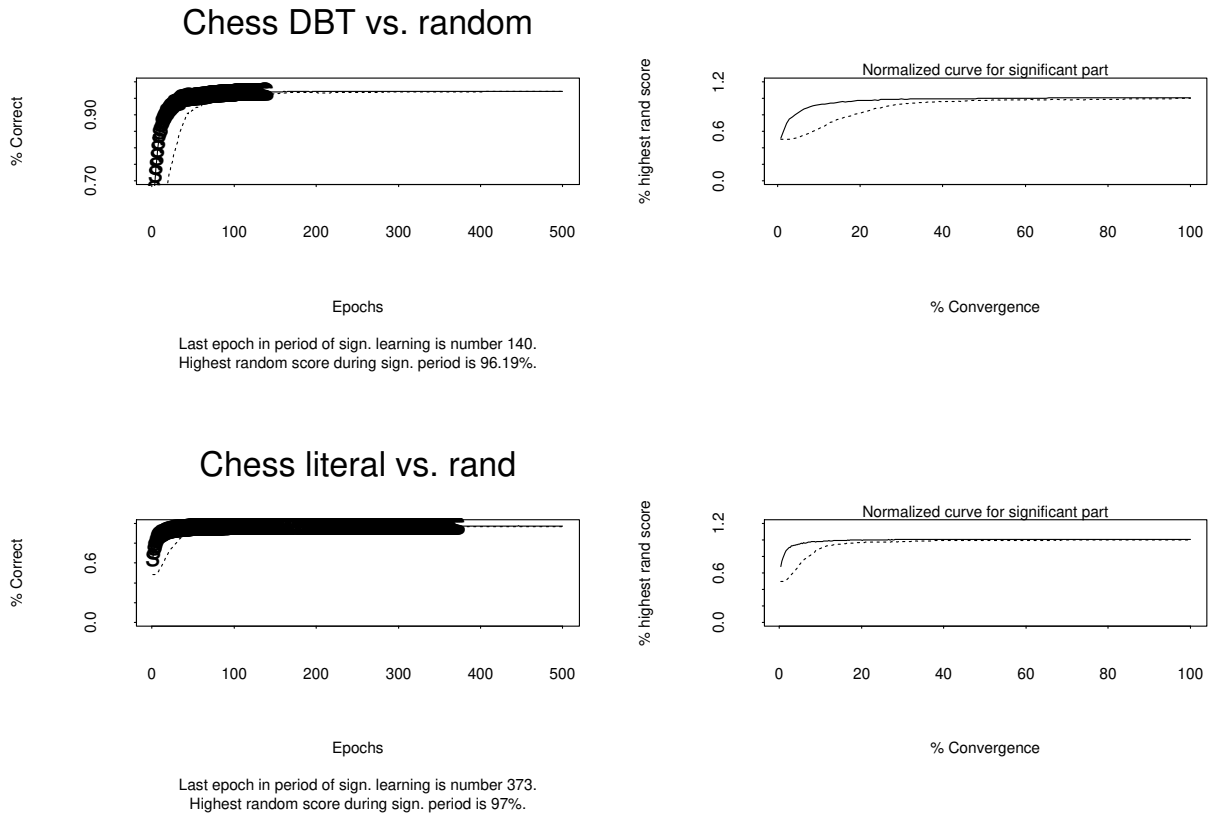


Figure 4.7: Mean and normalized mean generalization curves for DBT/random and literal/random comparisons on the Chess problem.

These curves relate the percent correct on a holdout set to the number of epochs of training. All comparisons are to randomly initialized networks, whose curves are shown as a dotted line. DBT and literal curves are solid lines. Epochs at which a significant performance difference was measured are labelled with the letter "S" on the curve with higher performance.
Note that both DBT and literal curves are significantly better in some epochs than randomly initialized networks.

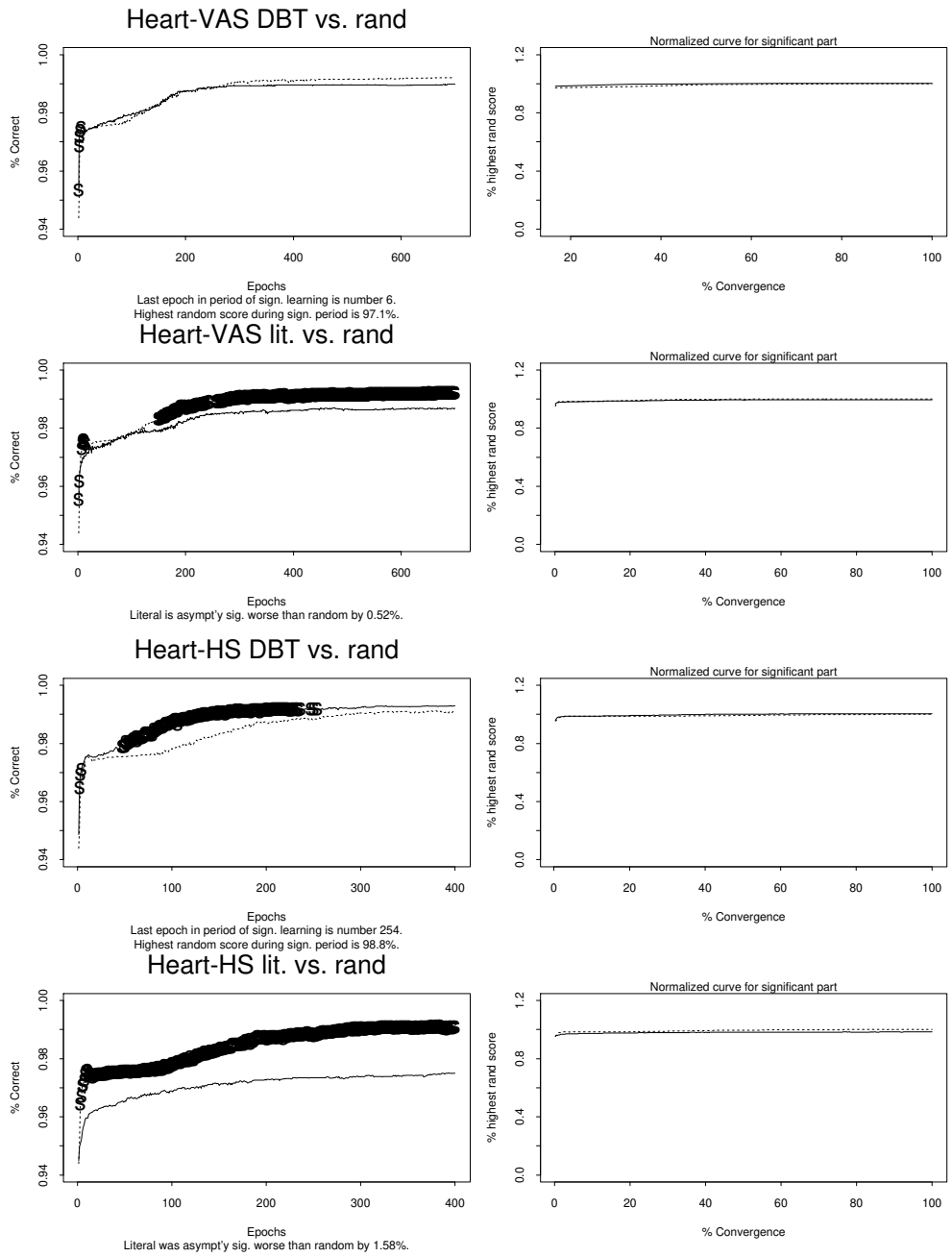


Figure 4.6: Mean and normalized mean generalization curves for DBT/random and literal/random comparisons on Heart-VAS and Heart-HS problems.

These curves relate the percent correct on a holdout set to the number of epochs of training. All comparisons are to randomly initialized networks, whose curves are shown as a dotted line. DBT and literal curves are solid lines. Epochs at which a significant performance difference was measured are labelled with the letter "S" on the curve with higher performance.

Note that the mean literal generalization curve is significantly worse asymptotically than the randomly initialized networks for Heart-VAS and Heart-HS. DBT is significantly better than randomly initialized networks during significant epochs on both tasks, though to a smaller degree than in tasks in previous figures.

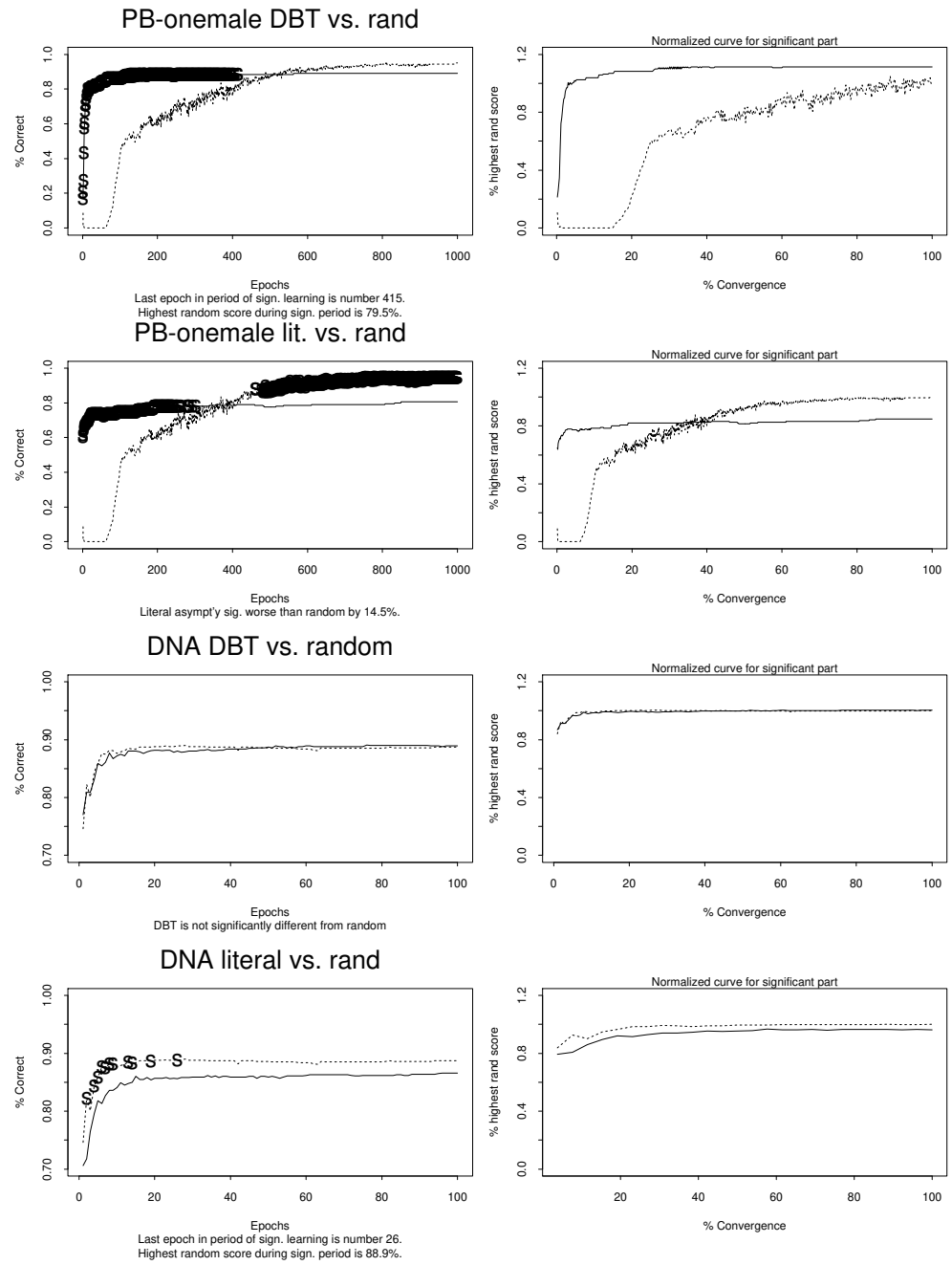


Figure 4.5: Mean and normalized mean generalization curves for DBT/random and literal/random comparisons on the PB-onemale and DNA problems.

These curves relate the percent correct on a holdout set to the number of epochs of training. All comparisons are to randomly initialized networks, whose curves are shown as a dotted line. DBT and literal curves are solid lines. Epochs at which a significant performance difference was measured are labelled with the letter "S" on the curve with higher performance.

Note that literal transfer does asymptotically significantly worse than random initialization on the Onemale task. Literal also does significantly worse for some epochs than randomly initialized networks on the DNA task. The DBT DNA curve is not significantly different from randomly initialized networks in any epoch of learning.

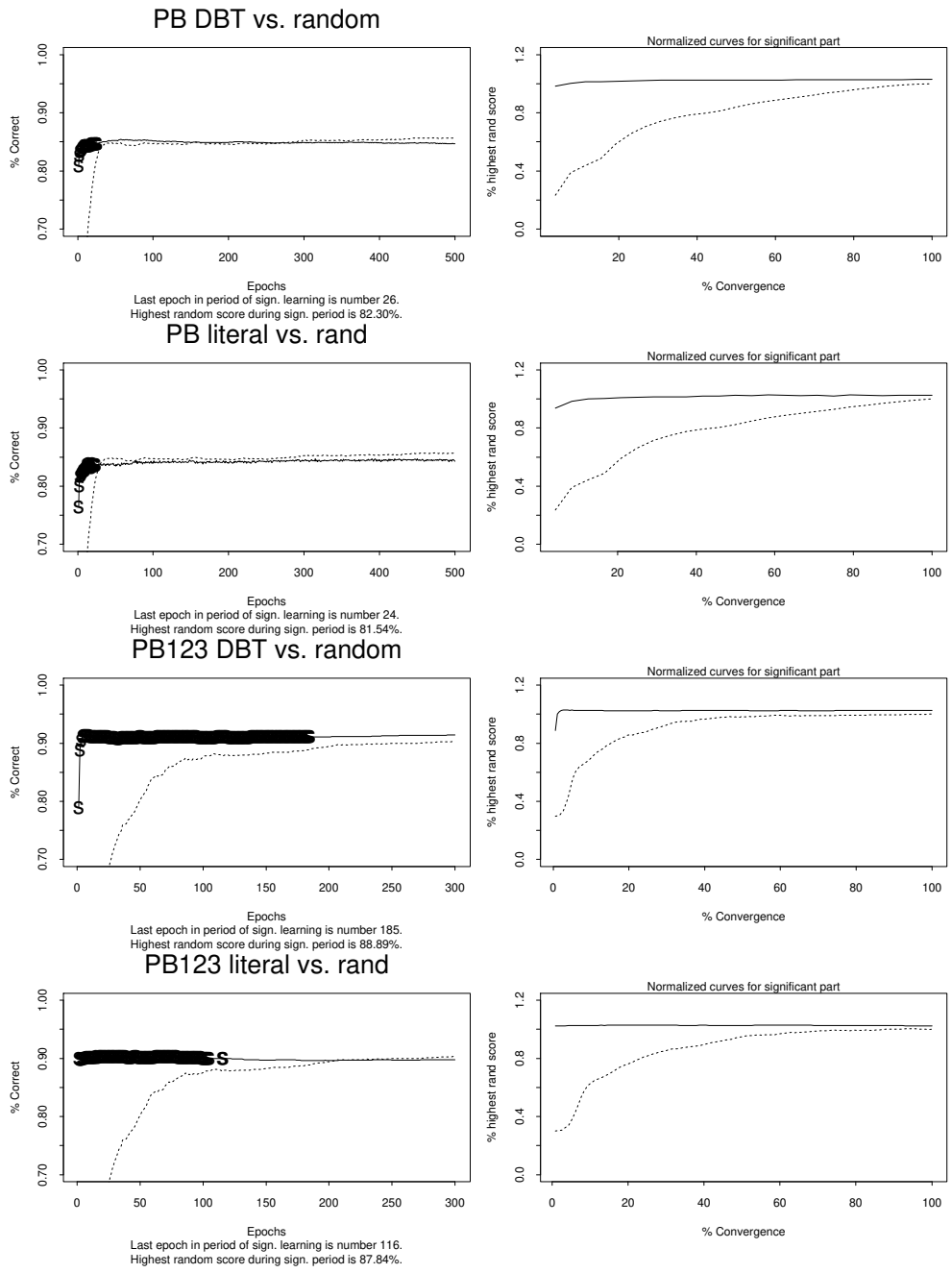


Figure 4.4: Mean and normalized mean generalization curves for DBT/random and literal/random comparisons on the PB and PB123 problems.

These curves relate the percent correct on a holdout set to the number of epochs of training. All comparisons are to randomly initialized networks, whose curves are shown as dotted lines. DBT and literal curves are solid lines. Epochs at which a significant performance difference was measured are labelled with the letter "S" on the curve with higher performance.

Note that for each comparison there is some period of DBT or literal superiority to randomly initialized networks.

Task	Avg. Source Score on Target data	Rand Last Perf
PB	0.51 ± 0.026	0.86±0.01
PB123	0.70 ± 0.033	0.90±0.02
PB-onemale	0.68 ± 0.040	0.95±0.00
Heart-HS	0.42 ± 0.127	0.99±0.00
Heart-VAS	0.72 ± 0.052	0.99±0.00
DNA	0.75 ± 0.063	0.89±0.02
Chess	0.86 ± 0.006	0.97±0.01

Table 4.2: Performance of source networks on target tasks.
Random network scores at the end of target training are shown for comparison. These results are in terms of average percent correct on a holdout set (for randomly initialized networks), and in terms of average percent correct on the entire target data set (for the source networks tested on the target data). Intervals shown are for 99.0% confidence. Note that networks that were randomly initialized and trained on target task data give consistently better asymptotic performance than the source networks tested on the target data.

the source networks achieved acceptable performance on the target task without further target training, then transfer would not have been motivated.

4.5 Curves showing experimental results on benchmark tasks

As described in Section 4.2.1, for each of the seven tasks described in Section 4.1, we generated three mean generalization curves: for DBT-, literal-, and randomly-initialized networks. These generalization curves are shown in Figures 4.4, 4.5, 4.6, and 4.7. Also shown are the normalized curves that reflect relative curve shapes. Normalized curves were obtained by rescaling each generalization curve horizontally so that the last significant epoch was at the position marked 100% convergence. They were rescaled vertically so that the performance of the randomly initialized network at this epoch was at the position marked 100% on the vertical axis (Section 4.3). We will quantify the results shown in these curves in the tables in the next section.

4.6 Quantifying learning curve results

Tables 4.3 and 4.4 quantify the results of the generalization curves shown in the previous section. In the following subsections we present background to these tables (Section 4.6), describe how to read them (Section 4.7), and also discuss how to interpret the results they reflect.

Section 4.6 for more details.

Although this approach makes the choice of stopping criterion more objective, the issue remains that the shape of the curves before achieving that criterion performance level has been lost. Therefore, we also measured the point (in terms of percent of epochs before the stopping criterion) at which curves reached 66%, 95%, and 98% of the highest curves' level at the stopping point. This approach to comparing curve shapes allows for same-method, different-task comparisons (since we measure the same quantities for every task), and for same-task, different-method comparisons (since we can compare multiple methods to the random initialization baseline to see which does relatively better).

Using these measures, we compared DBT-trained networks to those that had been randomly initialized. In order to gather evidence in support of DBT over literal transfer, we also compared networks initialized via literal transfer to randomly initialized networks. These results will be reported starting in Section 4.5.

Note that we did not directly compare DBT curves to those for literal transfer. This was because we view random initialization as a baseline against which any transfer method may be compared. By using this baseline, the amount of speed-up of any new method compared to random initialization may be compared to the same speed-up amount observed in DBT and in literal transfer over the random baseline.

Assumption about shape of random curve

The choice of the final random score as the 100% mark on normalized curves assumes that the random curves are monotonically increasing. If this was not true, we'd call a score that's not the highest score 100%, and so measures relative to this value would be misleading. We didn't choose to just use the highest observed random score as the 100% mark, because this would be tantamount to training on the testing data (using observations on the testing data to choose a stopping criterion) and so might give overly optimistic results.

4.4 Results of comparison to using the source network on the target task

Table 4.2 shows the average score of source networks, as tested on the target data.[4] For comparison, scores of randomly initialized target networks at the end of training are shown. These results show that using the source networks to classify the target data is not a good idea – they do not achieve the same performance scores as randomly initialized networks.

These results support our goal of choosing transfer benchmark tasks on which transfer is actually necessary. If the source and target populations had been so close that

[4]Recall that our cross-validation testing methodology made no single training/testing data split. In order to obtain the most meaningful estimate of the source network's ability to solve the target problems, we therefore used the entire target data set, instead of any single holdout "test" set.

was to establish that the DBT method improves not the performance of the final network, but the learning algorithms' speed in achieving that level (without harming performance), a more complicated analysis of learning curve shapes (recall Section 4.2.1) was required. Previous authors' approaches to this problem have involved choosing a stopping criterion (such as 99.5% performance, cf. [Shavlik *et al.*, 1991]), and measuring the number of training epochs to reach that point. Such an approach was problematic for our benchmarks, because best final performance scores varied widely: some of our tasks were more difficult, and so networks were never able to reach such a high performance score. Furthermore, this measure doesn't capture the shape of the curve leading to the 99.5% level, as illustrated in Figure 4.3.

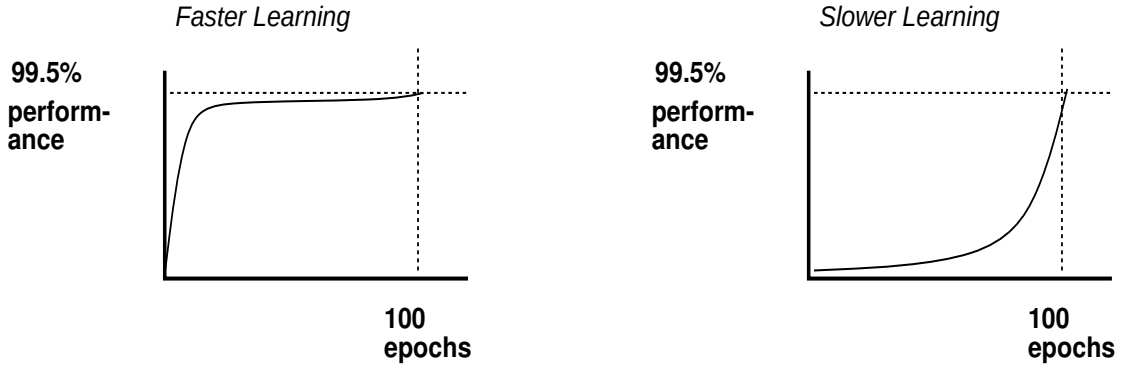


Figure 4.3: Illustration of how a fixed criterion may not capture learning speed.

Though both these curves reach the 99.5% performance level point at the same time, they have different shapes.

One solution that may seem suggested by Figure 4.3 is to simply take the area under the learning curves, but this retains the problem of 99.5% being an arbitrary criterion – would the relative areas have been the same if, say, 80% had been used as the criterion instead?

To address these issues, we used as a stopping point the epoch at which two mean generalization curves to be compared (i.e. DBT and literal for PB) reflected populations that do not have statistically significantly different results. To understand how this was done, recall that each of our curves reflects an average of 100 different runs – 10 for each different holdout condition for each of 10 different initial conditions. Therefore, at each epoch, two populations of 100 numbers can be compared. We performed a one-sided T-test to compare curves at each epoch, and considered the curves to be significantly different if the T-test reflected at least 99.0% confidence that the numbers came from different populations. We then determined the number of epochs for which the method to be tested was statistically significantly different from the baseline method of random initialization. We called the part of the curve up to and including this last significant epoch the "normalized" curve.

This method was designed for runs that achieved asymptotic performance scores that were not significantly different. In cases where there was an asymptotically significant performance difference, we simply reported this fact and did not use this measure. See

```
Best starting point is 1.000000, 0.100000, with tss 161.807000
Searching up and right
1.000000, 1.000000    tss: 382.974000, avgtss: 458.182400
2.000000, 0.100000    tss: 122.442000, avgtss: 159.576600
Best parameters are 2.000000, 0.100000, tss 159.576600

Starting out searching right
3.000000, 0.100000    tss: 124.215000, avgtss: 160.904200
Looking ahead (1) in right direction: 4.000000, 0.100000    tss: 125.310000,
avgtss: 162.749800
Looking ahead (2) in right direction: 5.000000, 0.100000    tss: 126.050000,
avgtss: 165.195200
Looking ahead (3) in right direction: 6.000000, 0.100000    tss: 128.928000,
avgtss: 168.315600
Looking ahead (4) in right direction:
Searching for better direction,  starting at 2.000000, 0.100000, tss 159.576600
10 searches of 5 epochs each done altogether
Final choices: 2.000000 0.100000 159.576600
```

As can be seen in this example, the network chose to start searching at $S = 2.0, C = 0.1$. It then tried some higher values of S, until a pre-set limit caused the search procedure to stop. These values of S and C were used in the DBT example of Section 3.4.

4.2.7 Testing source networks on target task

As discussed in Section 2.2, previous classifiers have implemented a form of transfer by simply using a trained source classifier for a slightly different population than that for which it was originally trained. In order to test our DBT method against this benchmark, and to establish that our transfer tasks did require some form of transfer, we also tested, for each of the seven tasks, the ten source networks on the target training data, without any further target task training. These trials yielded 10 different numbers (one for each holdout condition) for each of the seven tasks. For each task, the 10 numbers were averaged together. These results will also be reported in Section 4.4.

4.3 Learning speed measurement for benchmark problems

Empirical results for neural networks are often reported in terms of relative performance levels obtained on a set of test data (cf. [Weiss and Kulikowski, 1991]). Since our goal

4.2.6 Transfer parameters: cutoff, scaleup

The cutoff and scaleup (C and S) parameters to the DBT algorithm were determined empirically, using a similar procedure to that used for η and α search (Section 4.2.2). The average TSS score of networks that were trained for a small epoch count was used for determining good values of c and s to use.

This small epoch count was chosen to be $\frac{1}{100}$ of the total number of epochs that we had chosen to train the target problem for. In a practical setting, this epoch count would need to be chosen without relying on advance knowledge of the total number of training epochs. Several runs to this epoch count were done for each problem, and searching was stopped when a pre-set limit of runs had been executed. This approach usually produced a total number of epochs for finding C and S that was around 20% of the epoch count described above.

In more detail, the S, C search algorithm begins by running DBT with four S, C pairs:

S	C
10.000000	0.100000
10.000000	0.000010
1.000000	0.100000
1.000000	0.000010

The average TSS over the learning period was recorded for each of these pairs, and then the best value was chosen from this group. In order to determine which direction to search from that point, we searched "up" (higher C) and "right" (higher S). Depending on the resulting average TSS, we chose a first direction in which to search from that point (from the set of directions: higher C, lower C, higher S, lower S). We then searched in the chosen direction until, after 5 lookaheads, no improvement was found, at which point the direction was changed. This procedure was repeated until either we reached a local minimum or a pre-set limit on the number of searches was reached.

Example of determining S and C

We now give an example of finding S and C for the example problem introduced in Section 3.4. The search routine produced the following output, which is presented below to better illustrate the algorithm's behavior:

```
Doing four-corners search
Searching for a good starting point
10.000000, 0.100000    tss: 137.807000, avgtss: 176.336400
10.000000, 0.000010    tss: 137.557000, avgtss: 175.718400
1.000000, 0.100000    tss: 122.621000, avgtss: 161.807000
1.000000, 0.000010    tss: 122.612000, avgtss: 161.823400
```

In an attempt to strike a good balance between ease of use and accurate hidden unit count estimation, we used our Information Measure Based Skeletonisation (IMBS) algorithm [Ramachandran and Pratt, 1992]. This algorithm examines a trained network and identifies "superfluous" hidden units: those whose removal would not impact network performance. Superfluous units may be redundant, or may give the same activation for all training data patterns and so have little discriminability.

In order to use IMBS to find a proper hidden unit count, we trained a single network with a very large number of hidden units on the target problem. IMBS was then run, and suggested a number of hidden units for removal. The count of remaining hidden units was then used to train subsequent networks on that task.

Since this method only counts the number of hidden units required to fit the training data, it may err on the side of providing too many hidden units for subsequent training, allowing overfitting. However, based on the generalization scores were observed on these tasks, we do not believe that they were overfitting the data. Also, since all networks for a problem received this same hidden unit count, they were equally prone to this problem, and so performance and learning speed should have been equally impacted had overfitting occurred.

4.2.4 Target network epoch count

In all of the empirical studies we've done, TRS curves (Section 1.1) level out to a small slope after some number of epochs. Preliminary runs were done for each problem to determine an epoch count at which the TRS curve had been level for several epochs, based on visual examination of training curves. This epoch count was used for all subsequent studies.

4.2.5 Source network training

For each problem, ten different source networks were trained to a large number of epochs using different automatically determined η and α values.[3] Each source network for a problem was trained on the same training patterns, but starting with different initial weights. This approach allows us to make a more general claim about the resulting target network: that its performance was independent of the particular source network used (but depended on the training data set used, which was kept constant). Source networks used the same hidden unit count as determined for target networks for the same problem. Note that, since we tested DBT only on source networks that were trained well (to many epochs, with carefully chosen η and α), we cannot claim based on the results reported in this dissertation that DBT will do well on incompletely trained networks.

[3] We determined different source parameters for each of 10 different random initial weight sets to obtain the best source performance possible.

Move in the current direction repeatedly until a larger TSS value is encountered. Avoid local minima by searching ahead a few iterations before changing direction. Exit main loop of program if a maximum search count has been reached.

Train three networks that pursue different directions to the one just tried (i.e. if we were searching for higher α values, then try lower α, higher η, lower η.)

If none of these directions yield a lower TSS than the best found so far, then exit the loop.

Otherwise, repeat the loop while searching in the new best direction found, of the three just tried.

END;

Increments in η and α in the above procedure are chosen based on the current η or α value. For instance, if $\alpha < 0.1$, we'll next try $\alpha = 0.01, 0.001, etc.$.

For each of the tasks of Section 4.1, a single pair of η, α values was determined, based on a randomly initialized set of weights. These values were used for all runs of all experiments (DBT, literal, random) for a task. This methodology produced runs that were biased to favor the randomly initialized networks, if any, because parameters were determined based on randomly initialized networks.

Other approaches to automating the setting of η and α are described in [Atlas *et al.*, 1990c], [Barnard and Cole, 1989], [Brent, 1990], and [Chan and Fallside, 1987]). These techniques change these parameters during learning, however, and so fundamentally alter the nature of back-propagation training.

4.2.3 Hidden unit count in target networks

The automatic determination of network topology is an active research area; no single method for determining a suitable number and interconnectivity of units has been shown to work universally. In the absence of problem-specific information as to network topology, most networks are constructed with a single hidden layer, fully connected to input and to output units. Network topology determination in this paradigm then reduces to determining a suitable number of units in the hidden layer. Determination of proper hidden unit count is especially important in a transfer context, where we've observed that bias from the source network can lead to faster overfitting at the same time that it speeds up training.

Previous empirical comparison studies have used somewhat arbitrary methods of determining the proper network hidden unit count. For example, [Shavlik *et al.*, 1989] used a hidden unit count that was 10% of the sum of the input and hidden unit counts. [Weiss and Kulikowski, 1991] used a more systematic, but more time-consuming method, where many networks were trained with different numbers of hidden units.

parameters S and C (Section 4.2.6) and the same hidden unit count. Source and target networks also had the same number of hidden units (recall Section 1.4.7).

4.2.2 Learning rate and momentum

Several research projects exploring back-propagation networks (cf. [Shavlik *et al.*, 1991]) use the same parameters η (learning rate) and α (momentum) for different tasks. The results of these studies may have been misleading, however, since parameters are often task-dependent, and since there's no reason to believe that a learning rate that's good for, say, a chess problem, is appropriate for, say, a speech recognition problem. If, instead of using identical parameters, we use a fixed, parameterless automated parameter determination method, this problem can be avoided while still retaining experimental control. Furthermore, there is no risk of introducing inadvertent bias into the learning method via choice of parameters.

We developed an automated search algorithm to determine locally optimal values of η and α for standard back-propagation. The design of the search method was based on intuitions gained from training a number of different networks with different η and α values, which were chosen by hand. For instance, we have observed that α values greater than 1.0 seem to lead to local minima. However, we've observed that η values greater than 1.0 are often helpful.

Our η, α search method is based on the idea that good performance after a few epochs at a particular learning rate and momentum value predicts good performance after many epochs.[2] Therefore, to find good values of these numbers, we can run several networks to a few epochs and choose the parameters that lead to the best performance over these runs. The method implements a local hill-climbing search in η, α space, with the goal of minimizing TSS. Pseudocode is as follows:

Begin by running four networks with η, α values that are widely
distributed. Record the TSS for each run.
Choose the start position from the above which yields the lowest TSS.
Run a network with a slightly higher η value than the saved one,
and store the resulting TSS.
Run a network with a slightly higher α value than the saved one,
and store the resulting TSS.
Use the three saved TSS values to determine whether to start searching
for a higher α, lower α, higher η, or lower η.
LOOP

[2]Preliminary studies indicate that this assumption holds anecdotally; we are currently doing further work to verify it systematically.

hidden units were used, with $\eta = 1.1, \alpha = 0.55$. The data to represent the target contained 1230 patterns. The Hungarian source network had 294 patterns and was trained for 544 epochs. The VA source networks had 200 patterns and were trained for 544 epochs.

4.1.6 Chess endgame problem (Chess)

This classification task is to determine whether White or Black will win a chess game, based on a set of binary-valued features describing an endgame position in which the pawn is in location a7 and it is white's turn to move [Shapiro, 1988, Holte *et al.*, 1989, Shavlik *et al.*, 1989]. Classification was based on 73 binary inputs representing preprocessed features of the endgame position. Again, we preprocessed this data to duplicate the process used by [Shavlik *et al.*, 1989]. We constructed a transfer scenario from this problem by extracting from the training data all examples in which, if white's pawn were promoted, the king would be attacked in some way. Networks trained on this subproblem were used as transfer sources to a target problem for the full endgame task. 12 hidden units were used, with $\eta = 0.65, \alpha = 0.05$. The target task had 591 patterns. The source networks had 249 patterns and were trained for 500 epochs.

4.2 Methodology used to evaluate benchmark tasks

In this section, we describe techniques used to conduct a series of experiments to compare DBT to three baselines: randomly initialized target networks, target networks initialized via literal transfer, and source networks tested directly on the target training data, without further training.

4.2.1 Cross validation strategy

As described above, a target training data set was determined for each task. Each one was split ten different ways into a training and test set, with 90% and 10% of the data in each, respectively. Each split is called a *holdout* condition. By evaluating multiple networks with the same parameters on different holdout sets, a more reliable estimate of the true error rate of the generalization score can be determined [Weiss and Kulikowski, 1991]. For each holdout condition, 10 different initial weight sets were used for training, so that results didn't depend on any particular set of initial conditions. For randomly initialized networks, there were 10 different initial random weight sets. For literal trials, initial weights each came from a different one of 10 source networks (see Section 4.2.5). For DBT trials, each initial weight set was the result of applying the DBT transfer algorithm to one of the 10 different source networks. This methodology produced 100 different training runs for each condition (random, literal, DBT) and each problem (PB, PB123, onemale, Heart-VAS, Heart-HS, Chess, DNA).

These 100 runs were averaged to obtain 21 mean generalization curves, obtained by averaging the test set performance at each epoch. All runs for each task used the same values of η and α. All runs within each holdout condition used the same values of DBT

The data to represent the target task had 20 patterns. The same source networks used in the PB task were used here.

4.1.4 DNA promoter problem (DNA)

This is the DNA promoter recognition task described in [Towell *et al.*, 1990]. The input data set contained unary-encoded[1] amino acid sequences representing segments of a DNA strand. The target task was to determine whether a *promoter* (a sequence important in the process by which DNA is converted to proteins) was present within the input window. As in [Towell *et al.*, 1990], the network had 233 binary-valued inputs and 106 training patterns, using the same preprocessing scheme described in [Towell *et al.*, 1990]. Since the number of training patterns is less than the size of the input vector, this training data set is linearly separable (which makes it seem simple, though the linear separation created based on the training data may or may not generalize well to new tasks). An artificial transfer situation was created by using a source network trained on an arbitrarily chosen 26-pattern subset of the 106 available positive and negative examples. The data to represent the target training task consisted of the entire 106 patterns. 1 hidden unit was used, with $\eta = 1.05, \alpha = 0.7$. The source networks had 26 patterns, and were trained for 100 epochs.

4.1.5 Heart disease diagnosis for Switzerland patients (Heart-VAS, Heart-HS)

Using a 14-attribute set of diagnosis information, networks were trained for a heart disease diagnosis problem [Schaffer, 1992, Detrano *et al.*, 1989, Gennari *et al.*, 1989, Aha *et al.*, 1991]. Input features included the following attributes: age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved (on a stress test), an exercise-induced angina indicator, ST depression induced by exercise relative to rest, slope of peak exercise stress test segment, number of major vessels colored by fluoroscopy, and a thalamus condition indicator. Some of these attributes were preprocessed by partitioning, others, such as age, were left as ratio-valued numbers. The output classes corresponded to whether a patient did or did not have heart disease.

Two different source problems were used, represented by data from a hospital in Hungary, and from a VA hospital in Long Beach, California. Feature preprocessing was done to duplicate the approach described in [Shavlik *et al.*, 1991]. This included a technique for missing value processing, where activation for missing feature information was distributed evenly over the multiple input units representing that feature in a unary encoding. This procedure resulted in a 25-element input vector to the neural network, for both the training and testing data, in both source and target tasks. The target problem was to train a network to perform the diagnosis task on Swiss patients. 7

[1] Every amino acid was represented by four input units, with only 1 unit of the four receiving an activation 1 to represent an amino acid; the remaining inputs were 0.

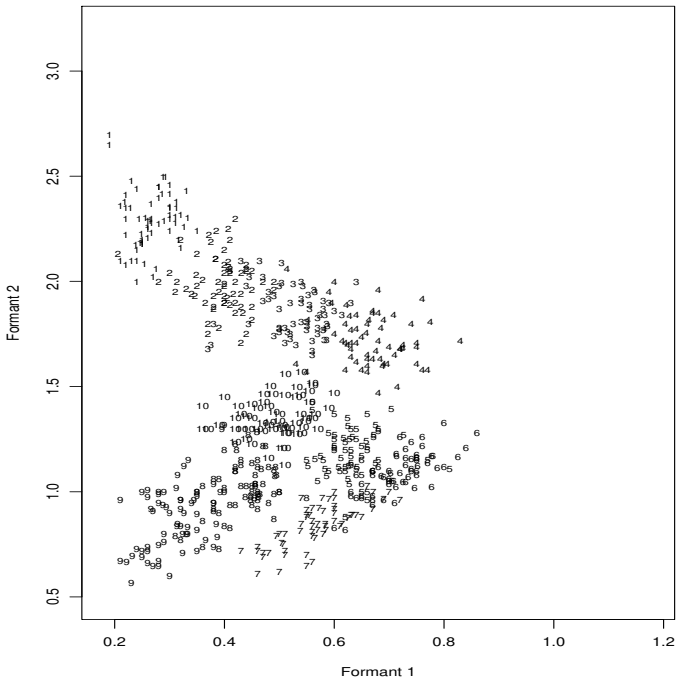


Figure 4.2: Peterson-Barney vowel data for only male speakers.

This was the target task for the PB problem. Note that the data is in the same general shape as in the source task, but shifted down along both axes.

discrimination (up to 80%) is possible with only two features. Therefore, hyperplane position and data arrangement in 2-space can be visualized easily. This problem is illustrated in Figures 4.1 and 4.2.

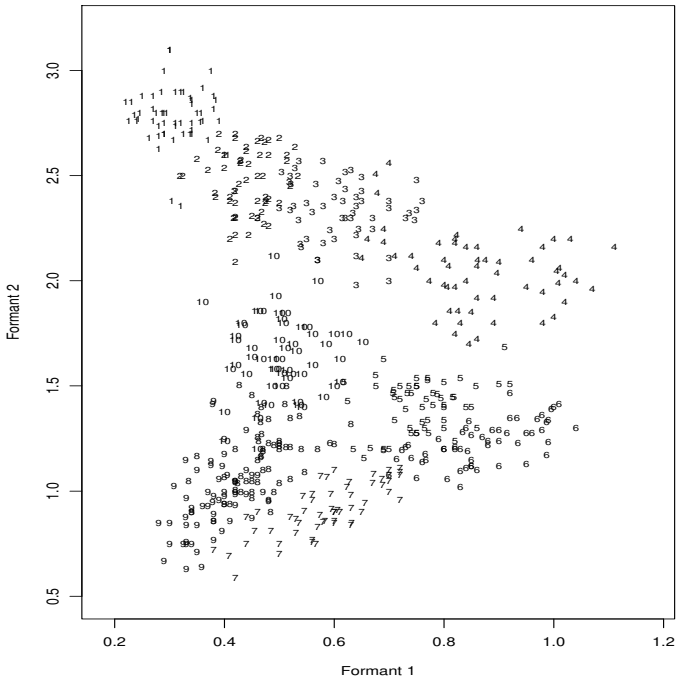


Figure 4.1: Peterson-Barney vowel data for only female speakers.

Used to train source network for the PB problem. Axes represent normalized frequency strengths.

4.1.2 3-class Peterson-Barney subset (PB123)

PB123 is a simplification of the full Peterson-Barney task, where only 3 of the 10 vowels are to be separated. Again, source networks were trained on only female speakers, and the target task was to recognize male speech. 3 hidden units were used, with target training parameters of $\eta = 0.3, \alpha = 0.000005$. The target task had 192 patterns. The source task had 168 patterns and networks were trained for 500 epochs. This problem is the subset of the data in Figures 4.1 and 4.2 for only classes 1, 2, and 3.

4.1.3 Female to single male transfer ("Onemale")

"Onemale" is a simplification of the full Peterson-Barney task in which the target network has only a single male speaker. We evaluated DBT on this task to see if it gave superior performance when transfer was from a complex task to a much simpler one. 24 hidden units were used, with target training parameters of $\eta = 1.1, \alpha = 0.75$.

Task Name	# input units	# output classes	Target Data Set Size	HU Count	η, α	Feature Types
PB	2	10	636	24	0.85, 0.1	ratio
PB123	2	3	192	3	0.3, 0.000005	ratio
Onemale	2	10	20	24	1.1, 0.75	ratio
Heart-VAS	25	2	1230	7	1.1, 0.55	ratio, nom., bin., interval
Heart-HS	25	2	1230	7	1.1, 0.55	ratio, nom., bin., interval
Chess	73	2	591	12	0.65, 0.05	nominal, binary
DNA	233	2	106	1	1.05, 0.7	binary

Table 4.1: Summary of benchmark tasks used to evaluate DBT

"nom." stands for nominal inputs; "bin." stands for binary inputs.

4.1.1 Peterson-Barney vowel recognition (PB)

PB is a 10-vowel sound classification task that was originally described in [Peterson and Barney, 1952] and recently recreated by [Watrous, 1991]. Training data for this problem was collected by asking subjects to pronounce vowels as part of the following words, spoken in isolation: *heed, hid, head, had, hud, hod, hawed, hood, who'd, heard.* The word list was spoken twice by each of 76 speakers, with the words in a different random order for each presentation. The resulting signals were preprocessed to yield frequency strength information at each time step. Vowels can be characterized as bands of energy in this spectro-temporal representation; each band is called a *formant.* The spectrograms were hand-segmented, which reduced this signal to a single time sample, from which the energy levels at the center frequency of each of the two highest-energy formants were extracted. This procedure yielded two real-valued numbers as network input. We duplicated the preprocessing technique on this data that was reported by [Nowlan, 1991]. We divided each formant value by 1000, yielding two values in the ranges [0,1.5] and [0,4.0], respectively.

Source networks were trained on female-only speech, using 560 training patterns and 8000 training epochs. The target task was to recognize speech from males only. Our hypothesis was that, by using a female-trained network as an initial bias, learning on the male-only network could be improved. 24 hidden units were used (this hidden unit count, along with those for the remaining tasks, was determined using the procedure described in Section 4.2.3), with target training parameters of $\eta = 0.85, \alpha = 0.1$. The data contained 636 patterns representing the target task. This problem is of special interest because it is fairly complex and similar to a real-world task, yet reasonable

Chapter 4
Results

This chapter describes how the DBT algorithm was evaluated in order to demonstrate that it produces a learning speed improvement. We begin in Section 4.1 by describing a series of benchmark tasks from a variety of domains. Then in Section 4.2 we describe the methodology we used to evaluate DBT on these tasks. A method for measuring the speed-up attributable to DBT is described in Section 4.3.

Section 4.4 then presents the performance scores of source networks when they were simply tested on target task data. We demonstrate that the final performance is much lower for all 7 tasks studied than that of a randomly initialized network trained on the target data. Then, in Section 4.5, we show the raw mean generalization curves obtained from target task training. Curves will compare both DBT and literal networks to randomly initialized ones. Section 4.6 contains tables that quantify the results shown in these learning curves. These results in general, and the results tables in particular, are described in Sections 4.6 and 4.7, respectively. We summarize the experimental results presented in this chapter in Section 4.8.

4.1 Tasks used to evaluate DBT

This section describes the data sets used for evaluation of the DBT method that was presented in Chapter 3. In order to evaluate DBT's generality, these tasks were chosen from an online repository to represent a variety of situations. Networks had inputs that ranged from binary-valued to ratio, interval, and nominal values and had between 2 and 10 output units and between 2 and 233 input units. Various parameters, such as learning rate, momentum, and hidden unit count, were determined using the methods that will be described below in Section 4.2.

Table 4.1 summarizes the benchmark tasks used to evaluate DBT. Note that the tasks studied have in common the fact that they were all of moderate size, as measured by the size of training sets and the number of nodes in the networks. These tasks were larger than the toy problems on which networks are sometimes measured, but were smaller than larger problems, with many thousands of training patterns, which have been trained in other studies.

Note that Table 4.1 shows tasks that vary by the different kinds of features that are used as network inputs. These inputs were obtained using a variety of preprocessing techniques, which can have an impact on learning. We chose to use preprocessing techniques that were identical to those used in previously published work on these data sets, as described in more detail below.

the average absolute value of the weights and bias for each hidden unit, before DBT was run. The fourth column shows the scale factor chosen by DBT for the given HU. As described above, this was based on the S value of 2.0 and the IM. Note that, because $S = 2.0$, the largest scale factor on a hyperplane is 2.0. The last column shows the average weight magnitude of the weights defining each hyperplane upon output from DBT.

Note that the chosen scale factor of 2.0 means that the hyperplanes whose IM values exceeded the cutoff value of 0.1 (all but 3) received higher magnitudes when they were transferred to the initial conditions of the target task. DBT's role on this problem was, therefore, twofold: (1) DBT identified low-IM hyperplanes and lowered their retentiveness, and (2), for the remaining hyperplanes, DBT raised magnitudes overall, relative to the IM value. This made most hyperplanes more retentive than they would have been in literal transfer.

Chapter 4 will summarize the results of over 2100 additional runs of the DBT method, on a variety of problems.

3.5 Conclusion

This chapter has presented the details of the DBT algorithm, and of the IM metric on which it is based. We have seen that DBT performs two primary roles in modifying source weights – it lowers retentiveness of less useful hyperplanes (either through a low S value or for hyperplanes for which $IM < C$), and it raises retentiveness of more useful hyperplanes (through $S > 1.0$). We now describe a series of experiments to test DBT.

HU#	HU IM	Avg. Mag. on input	Scale Factor	Avg. Mag. on output
1	0.856566	38.665715	2.000000	77.331430
2	0.583557	26.124430	1.362551	35.595872
3	0.851171	33.315121	1.987402	66.210553
4	0.684703	40.850806	1.598717	65.308873
5	0.486894	16.108197	1.136851	18.312621
6	0.690180	35.584755	1.611504	57.344992
7	0.204924	35.789684	0.478479	17.124613
8	0.000000	1.439079	$\star$	0.236713
9	0.658203	29.805623	1.536841	45.806513
10	0.000000	3.520771	$\star$	0.195976
11	0.100478	45.113146	0.234606	10.583797
12	0.671402	25.409403	1.567659	39.833289
13	0.681007	27.474739	1.590087	43.687231
14	0.743190	54.439360	1.735278	94.467408
15	0.534876	35.778146	1.248885	44.682778
16	0.495005	48.504792	1.155789	56.061326
17	0.576382	53.572980	1.345797	72.098369
18	0.661045	35.708491	1.543478	55.115258
19	0.404706	9.805365	0.944949	9.265574
20	0.658697	27.509550	1.537996	42.309574
21	0.477071	21.338561	1.113915	23.769351
22	0.005786	50.251846	$\star$	0.404883
23	0.709666	79.529471	1.657002	131.780487
24	0.620727	27.395267	1.449338	39.704992

Table 3.3: Results of rescaling weights via DBT on the PB problem.

$\star$ indicates that the indicated hyperplane's IM value was below the C cutoff, so the scale factor S was ignored.

In an empirical trial, we observed that hyperplanes defined by some large and some small weights did not correctly move into position, so if a hyperplane has $IM_t < C$ and it has a largest weight magnitude ratio greater than or equal to 100.0, then that hyperplane's weights are randomized in $[-0.5, 0.5]$. For example, in a 2-input network, if a hyperplane with $IM < C$ had weights of $-1000.00, -100.00, +120$, then it would receive random low-magnitude weights. If, however, it had weights of $-1, +1, -2$, it would be rescaled to weights of $-.25, .25, -.5$.

The C parameter is necessary so that some network weights have as much flexibility as if they had been randomly initialized. We found in several experiments that the scaling only via Equation 3.1 was not adequate for this purpose. We observed that if some source hyperplanes were scaled down relative only to IM_t, useful hyperplanes moved out of position; scaling up relative only to IM_t sometimes prevented useless hyperplanes from moving. Therefore, the C parameter was used to set a threshold so that hyperplanes with low discriminabilities would be effectively randomized.

In order to determine S and C for a particular source and target task, we ran DBT several times for a small number of epochs with different S and C values. Assuming that the average TSS over a few epochs is indicative of asymptotic performance, we chose the S and C values that yielded the best average TSS after a few epochs. We used local hill climbing in average TSS space to decide how to move in S, C space. More details of this method will be given in Section 4.2.6, since the procedure for determining S and C is similar to that for determining the η and α parameters to back-propagation.

3.3.3 HO layer weights

DBT also randomizes the weights in the network's hidden-to-output layer. At first it might seem useful to also set hidden-to-output (HO) weights non-randomly. However, we found experimentally that, since HO representations strongly depend on proper IH representations, and since the transfer algorithm creates only approximately positioned IH hyperplanes, it is better to set HO weights to low random values to achieve maximum flexibility.

3.4 DBT example

To demonstrate the DBT algorithm, we ran it on a vowel recognition problem called PB. The network for this problem has 2 input units, 24 hidden units, and 10 output classes (this problem will be described in more detail in Section 4.1). DBT input for this example consisted of weights from one source network, along with training data from one of 10 holdout partitions (the training data used constituted 90% of all available data). Using the automated procedure to be described in Section 4.2.2, we chose a learning rate 0.85 and a momentum of 0.1. S of 2.0, and C of 0.1 were chosen by the procedure described in Section 4.2.6. Table 3.3 shows the values used and calculated by DBT when run on this problem.

The first column of Table 3.3 shows the number of each hidden unit. The second column shows the IM value that DBT calculated for each one. The third column shows

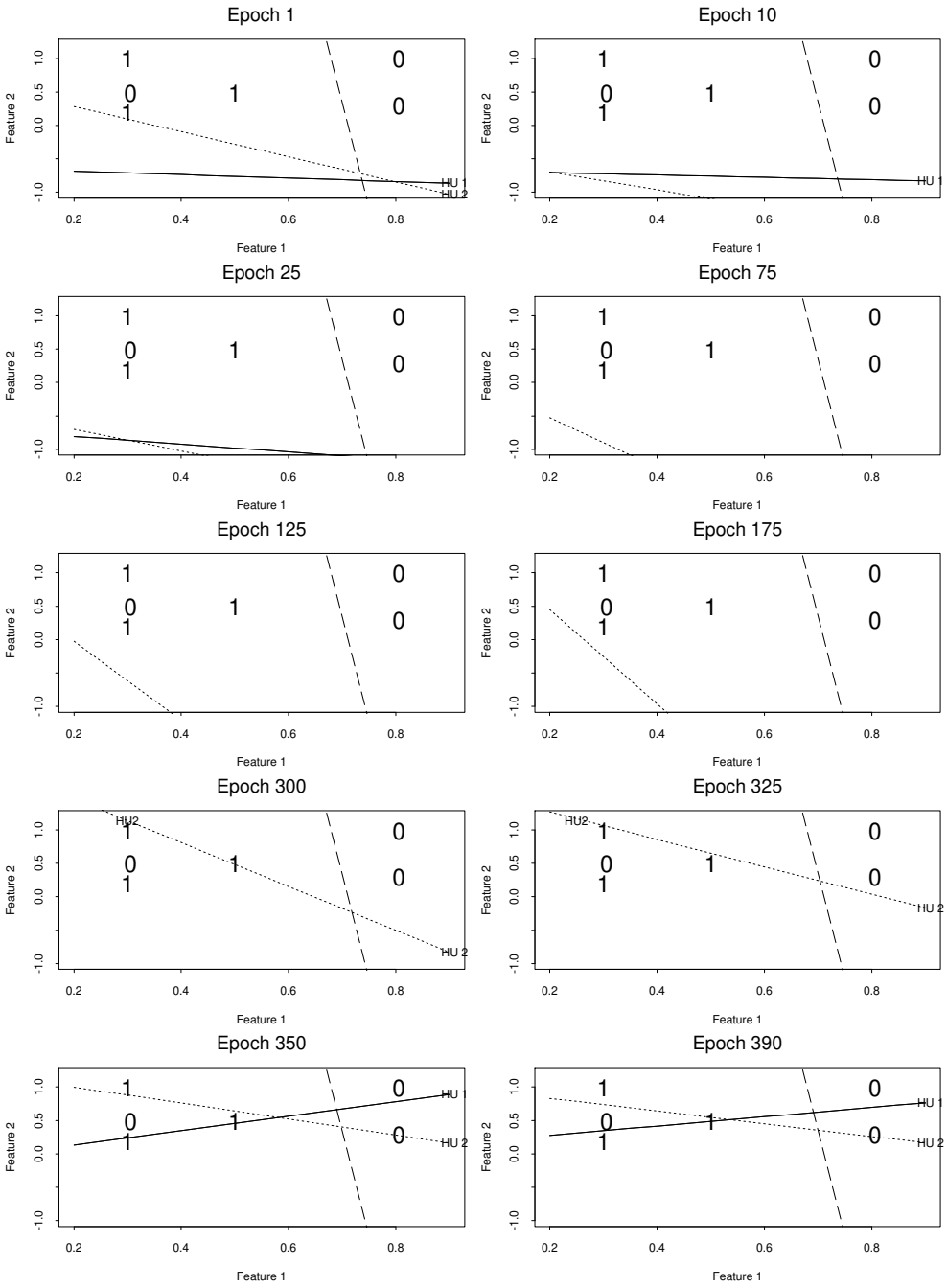


Figure 3.8: Snapshots of hidden unit hyperplane positions, showing how DBT can speed up learning.

This demonstration was created by copying HU2 and HU3's initial positions from their final positions in a trained network. They had average weight magnitudes of 4.93 and 4.99, respectively. Hidden unit 1's hyperplane is a solid line; HU2's is a dotted line, and HU3's hyperplane is shown as a dashed line. HU1 was given a position that did not separate the training data. Its magnitude was 5.83. DBT was run on the resulting network. Note, in contrast to Figure 3.2, that the training data is separated by epoch 390. HU1 was given a small magnitude by DBT, and so it is able to move into place quickly.

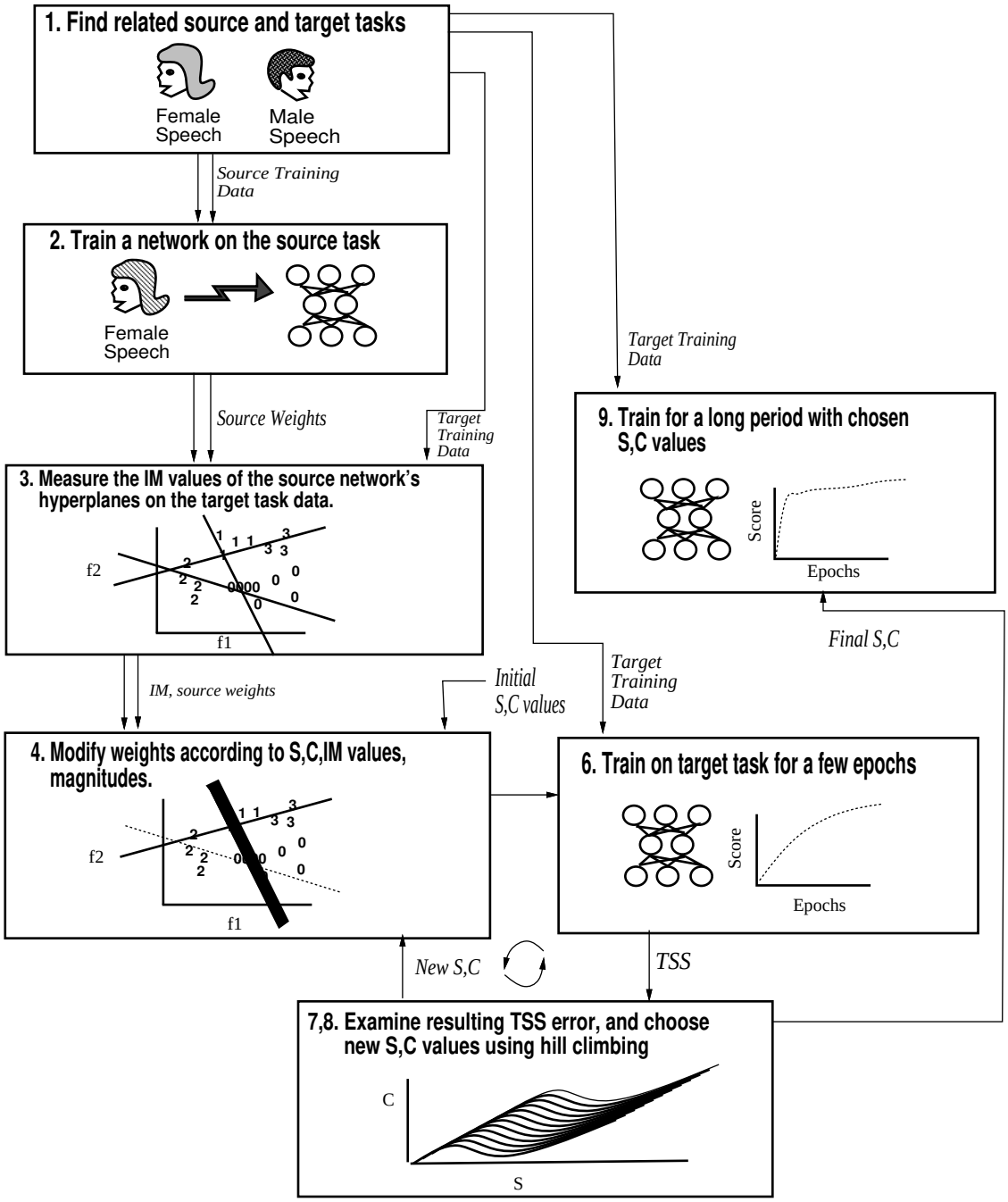


Figure 3.7: Flow of control in DBT.

Source training data (1) is converted into a set of source weights (2). The IM of source weights is measured against the target task (3), and then used, along with S and C and the source magnitude, to modify weight magnitudes for use in the target (4). Networks are trained for a few epochs (6), and TSS feedback is used to adjust S and C in a hill climbing procedure (6-8). When S and C values are finalized, the target network is trained (9).

Input:
Source network weights
Target training data
Parameters: C (cutoff factor), S (scaleup factor)
Output:
Initial weights for target network, assuming same topology as source network
Method:
For each source network hidden unit i
Evaluate the IM value of the source hyperplane with respect to the target training data, calculating IM_{ti} ($\in [0,1]$)
(the "discriminability" of that hyperplane; "t" indicates that IM is calculated over the target training data, not the source; "i" indicates the hyperplane whose discriminability on that data is measured)
Rescale IM_{ti} *values so that largest has value S. Put result in* S_i.
For IM_{ti}*'s that are less than C*
If highest magnitude ratio between weights defining hyperplane i is > 100.0, *reset weights for that hyperplane randomly*
Else uniformly scale down hyperplane to have low-valued weights (maximum magnitude of 0.5*), but in the same position as before.*
For each remaining IH hidden unit i
Modify each source weight w^t_{ji} *to be:* w^t_{ji} = *source weight* $w^s_{ji} \times S_i$
Set hidden-to-output target network weights randomly in $[-0.5, 0.5]$

Figure 3.6: The discriminability-based transfer (DBT) algorithm.

and S are discussed in more detail below. As output, DBT produces initial weights for subsequent back-propagation training on the target task. It attempts to determine these initial weights so that they speed up target learning compared to networks initialized with literal or random weights.

3.3.2 Modifying weights to be proportional to IM

DBT modifies the weights defining each source hyperplane to be proportional to the IM value, according to an input parameter, S. DBT is based on the idea that a good target magnitude M_t for a hyperplane is

$$M_t = S \times M_s \times IM_t \tag{3.1}$$

where M_t is the magnitude of a target network hyperplane, S ("scaleup") is a constant of proportionality, M_s is the magnitude of a source network hyperplane, and IM_t is the discriminability of the hyperplane on the target training data.

We assume that this simple relationship holds over some range of IM_t values. A second parameter, C, determines a cut-off in this relationship – source hyperplanes with $IM_t < C$ receive magnitudes based on a different function: magnitudes are very low, so that they are effectively equivalent to hyperplanes in a randomly initialized network. If the largest ratio between weight magnitudes defining a hyperplane with $IM_t < C$ is larger than 100.0, then those weights are scaled down uniformly, retaining that hyperplane's position.

3.2.2 Calculating IM for a decision surface

Let A be a set of all symbols a_i that can be sent over a channel, and let B be the set of all symbols b_j that can be received. Mutual information (IM) is calculated as: $I(A;B) = H(A) - H(A/B)$. H(A/B) is the notation used to express the equivocation, which was described above. H(A) is the a priori entropy.

From [Shannon, 1949], the formula for $H(A)$ is $\sum_{i=1}^{r} P(a_i)\frac{1}{\log P(a_i)}$, where each a_i is a different class, and $P(a_i)$ is the prior probability of class a_i. $H(A/B)$ is

$$\sum_{a_i \in A, b_j \in B} P(a_i, b_j) \log \frac{1}{P(a_i|b_j)}$$

where a_i is a class, b_j is a received symbol, $P(a_i|b_j)$ is the probability of being in a class, given that the data item is on a particular side of a hyperplane, and $P(a_i, b_j)$ is the joint probability of sent symbol a_i and received symbol b_j. This can be estimated from the training data.

As described in [Mingers, 1989] (see also [Buntine and Niblett, 1992]), this formula for mutual information (IM), for a decision surface in a multi-class problem, based on a set of training data patterns, can be reduced to the following:

$$\text{IM} \quad = \quad \tfrac{1}{N}\Big(\quad \sum\sum x_{ij} \log x_{ij} - \sum x_{i.} \log x_{i.} - \sum x_{.j} \log x_{.j} + N \log N \Big)$$

Here, N is the number of patterns, i is either 0 or 1, depending on the side of a hyperplane on which a pattern falls, j indexes over all classes, x_{ij} is the count of class j patterns on side i of the hyperplane, $x_{i.}$ is the count of all patterns on side i, and $x_{.j}$ is the total number of patterns in class j.

Section 1.4.1 described previous work that uses the IM metric to evaluate decision surfaces in decision tree construction. In contrast to that approach, DBT uses IM to evaluate decision surfaces whose positions have already been determined by back-propagation.[1]

3.3 The DBT algorithm

The DBT algorithm is outlined in Figure 3.6. Figure 3.7 shows the algorithm graphically. Figure 3.8 shows how the problem illustrated in Figure 3.2 was fixed via DBT. Note that, when DBT is used on this simple problem, the hyperplanes move into separating positions much faster (by epoch 390) than in literal transfer as shown in Figure 3.2 (required 3100 epochs). The remainder of this section describes various details of the DBT algorithm.

3.3.1 DBT I/O

DBT takes as input the weights defining the source classifier and the target training data. In addition, two parameters to DBT, called C and S, must be determined. C

[1] We've published one other paper that uses IM evaluatively, this time for the purpose of identifying superfluous back-propagation hyperplanes [Ramachandran and Pratt, 1992].

Information theory	**Decision surface evaluation**
Identity of source symbol a_i	Target class identity
Sending a source symbol	Receiving an input vector in performance mode
Prior probabilities of source symbols	Estimated probabilities of class frequencies, from training data.
Received symbol b_j	Indication of the side of the hyperplane decision surface that the input vector is on. This will be $b_j = 0$ or 1.

Table 3.1: Correspondence between information theory and decision surface evaluation.

Quantity	**What represents it in a classifier**
A priori entropy	Represents the uniformity in class distribution before the decision surface is placed. High a priori entropy corresponds to evenly mixed classes. If all classes had the same prior probabilities, then the a priori entropy would be 0.
Equivocation	Represents the average uniformity of distribution of the classes on both sides of the decision surface, over all classes, and both decision surface sides. For instance, in a 2-class problem, this will be low if all 1's are on one side of the decision surface, all 0's are on the other.
Mutual information	The difference between the a priori entropy and the equivocation. Represents how much difference the decision surface makes. For instance, if there is only one class (zero a priori entropy), then the mutual information must be 0 even with a very low equivocation. The best situation is when the classes are evenly mixed without the decision surface (a priori entropy is 1 in the 2-class case) but the subregions created by the decision surface contain only 0's and 1's.

Table 3.2: Correspondence between calculated quantities in information theory and their meanings when evaluating the IM of a decision surface.

Translating from information theory to decision surface measurement

Consider the situation where we are given a set of training data and a classifier. For simplicity, assume there are two input values, two output classes, and a single decision region (with two sides) in the classifier. Assuming that the training data is representative of any new data that will be input to the classifier, our job is to quantify the decision region's utility (recall Figure 3.3). For this scenario, we can translate the information theory concept of "the amount of information about the symbol a_i conveyed by the possibly noisy symbol b_j" into "the amount of class identity information about an arbitrary input vector conveyed by knowing which side of the decision surface it is on". In our example scenario, a_i is 0 or 1, for the true class, and b_j is 0 or 1, for the decision surface side. This correspondence is shown in Figure 3.5. To use the IM formulation,

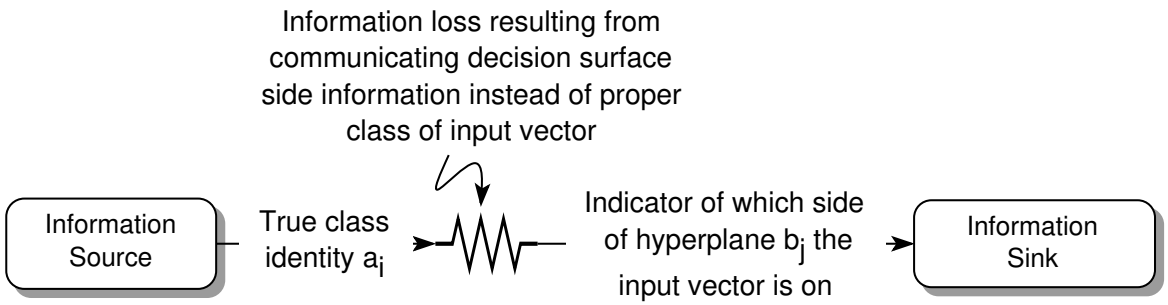


Figure 3.5: Information flow through a hyperplane decision surface.

The class identity for a data vector a_i is sent through a "noisy channel", in the sense that it is re-represented as the side of a particular hyperplane, resulting in a signal b_j = 0 or 1. This is analogous to sending information through a noisy channel, as is studied in information theory.

the set of training data can be used to estimate the prior probabilities of each class, based on observed class frequency. The utility of a hyperplane decision surface then depends on the degree to which it helps to separate a previously mixed group of data (high entropy) into purer subsets. If all class-0 items are on one side of the hyperplane, and all class-1 items are on the other side, then receiving the b_j information (hyperplane side) reduces the entropy to 0 in each of the partitions created by that decision surface. In contrast, if 0's and 1's are evenly mixed on either side of the hyperplane, it's not very helpful in separating the data, and the mutual information is small.

The correspondence between the classical formulation of information through a channel and evaluation of a decision surface in a classifier is summarized in Table 3.1. Table 3.2 summarizes how information theory quantities are used in decision region evaluation. We now provide full details for how to calculate IM.

Basis for IM in information theory

Consider the situation shown in Figure 3.4, where a symbol a_i from an alphabet of possible source symbols A is sent through a possibly noisy channel, and received as b_j, a symbol from an alphabet of possible received symbols B. Ignoring for a moment the

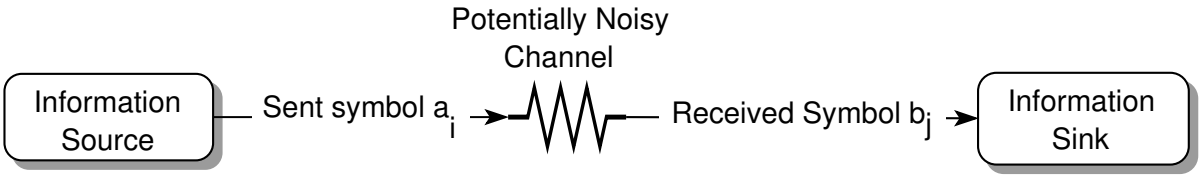


Figure 3.4: Information flow through a channel.

The symbol a_i is sent through a potentially noisy channel, and received as symbol b_j.

benefit of knowledge of b_j's identity, assume we are given a set of prior probabilities of the source emitting each particular symbol, $P(a_i)$. We can characterize the *entropy of the information source*, or the *a priori entropy* as the amount of information we'd need to determine a_i's identity, given only these prior probabilities. For simplicity, assume that we have a *binary* information source, and the only two symbols that can be sent are 0 and 1. If $P(0) = P(1) = 0.5$, then both symbols are equally likely, and we need a full bit of information to determine the signal's identity. In this situation, we say that the a priori entropy is 1. If, on the other hand, $P(0) = .9999$ and $P(1) = .0001$, then in most situations if we guess that the symbol is 0 we'll be right, and so we say that the entropy is very small. A common characterization of the entropy is that it's the amount of information conveyed by being told the symbol's identity. This is very small if we were already pretty sure that the symbol was 0.

Now consider the amount that the entropy changes when the signal b_j is received. In a common interpretation, b_j is meant to indicate the identity of a_i, but may have been modified by noise in the information channel. We can characterize this noise by specifying conditional probabilities $P(a_i|b_j)$ for particular values of i and j. The *equivocation* of the information channel is the average amount of entropy after receiving a symbol b_j, over all possible source and received signals. Intuitively, the equivocation reflects the degree to which we remain uncertain about the identity of a_i, despite having received b_j. The *mutual information* or IM in the channel is defined to be the a priori entropy minus the equivocation. It represents the amount of information that the b_j values convey. If there is no noise in the channel, then $b_j = a_i$ and any entropy previously in the channel is reduced to 0 by receiving b_j, and the IM is equal to the a priori entropy. Intuitively, again, b_j has told us a_i's identity, which is now a certainty. If the channel is very noisy, then b_j may give us little or no information (entropy reduction) about the identity of a_i, and the mutual information (IM) will be low.

3.2 Evaluating classifier components

As discussed in Section 1.3, we borrow a metric for evaluating hyperplanes called Mutual Information, or IM, from decision tree induction [Quinlan, 1983], which took it from information theory [Shannon, 1949] (also see [Abramson, 1963]). IM is also sometimes called "discriminability", hence the name "discriminability-based transfer" (DBT). Given a set of training data and a decision surface that crosses through it, the IM function returns a value between 0 and 1, indicating the degree to which the hyperplane helps to separate the data into different classes. For example, if a hyperplane in a two-class problem separated a mixed group of training data into two regions, each of which contained only patterns of a single class, then that hyperplane would receive a high IM. The IM metric is illustrated in Figure 3.3.

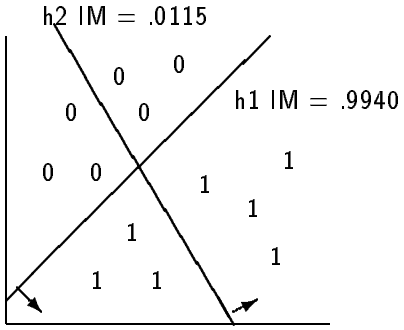


Figure 3.3: Hyperplanes and their IM.

This figure shows the training data for an artificially constructed two-feature, two-class problem and a possible configuration of the hyperplanes formed by each hidden unit at the end of training. Hyperplane $h1$'s higher IM corresponds to the fact that it separates the two classes better than $h2$. (Arrows indicate regions of feature space where hidden units have activations > 0.5.)

We now present the background of the IM metric, along with the method used to calculate its value.

3.2.1 The IM metric: background

IM is a measure of the information gained by knowing the value of an attribute (side of a hyperplane boundary) for the purpose of classification. The higher the IM of an attribute, the greater the uniformity of class data in the subsets of feature space it creates. IM is the same as the *mutual information* $I(A;B)$ of information theory, which characterizes the amount of entropy reduced by receiving a symbol through a channel. We now summarize this theory and show how it can be used to characterize the amount of entropy reduced (information gained) about the class identity of an input data item, given knowledge of the side of a decision surface on which it falls. Similar discussions at varying levels of detail can be found in [Breiman *et al.*, 1984, Mingers, 1989, Quinlan, 1983, Buntine and Niblett, 1992].

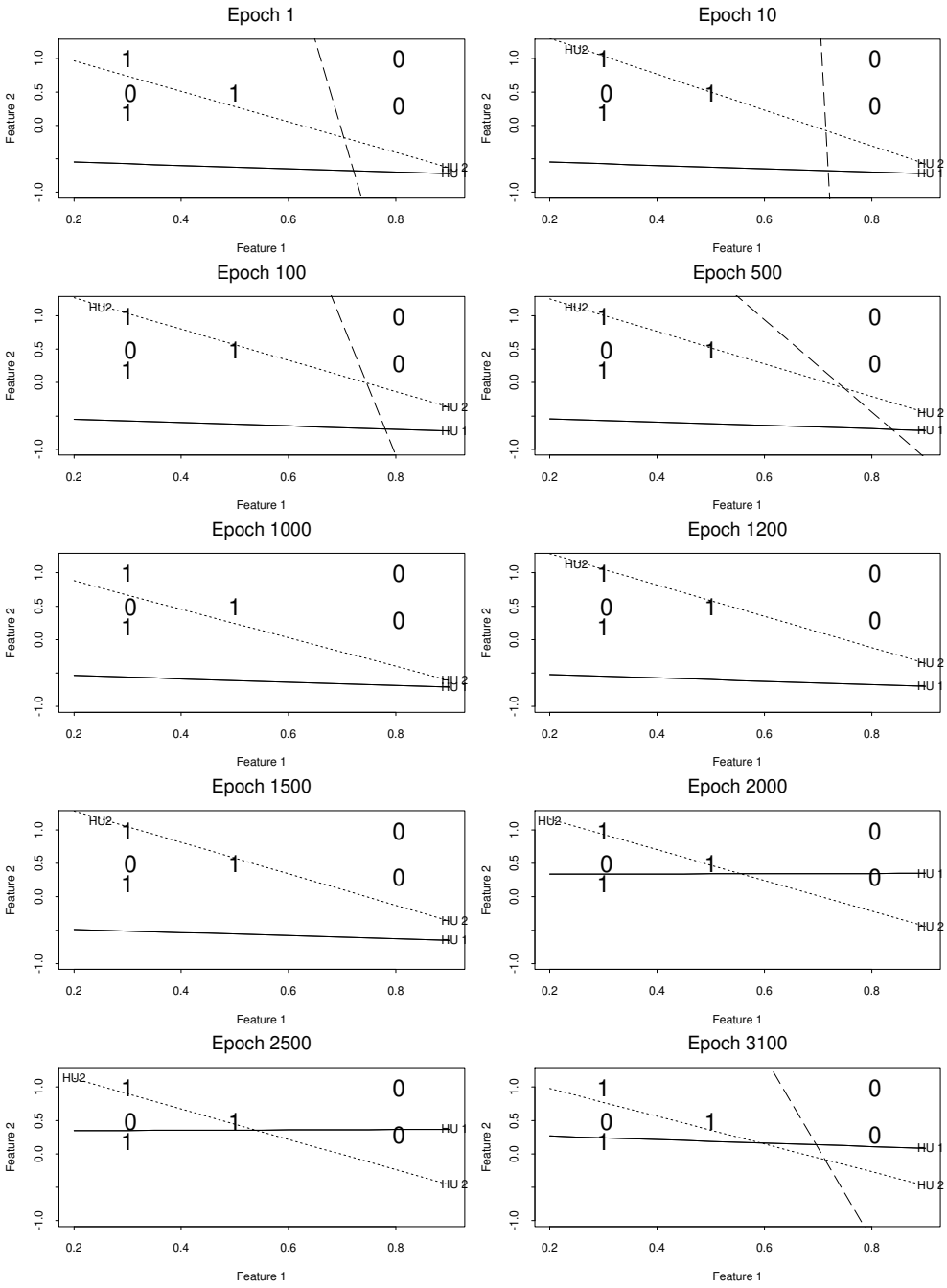


Figure 3.2: Snapshots of hidden unit hyperplane positions, showing how an excessively high magnitude hyperplane (HU1, shown as a solid line) from literal transfer can slow down learning.

This demonstration was created by copying HU2 and HU3's initial positions from their final positions in a trained network. They had average weight magnitudes of 4.93 and 4.99, respectively. Hidden unit 1's hyperplane is a solid line; HU2's is a dotted line, and HU3's hyperplane is shown as a dashed line. HU1 was given a position that did not separate the training data. Its magnitude was 5.83. Note how HU1 (near the bottom of the picture) seems fixed in place. Its high magnitude causes learning to be slow (taking about 3100 epochs to converge to 100% performance on the training data), compared to a randomly initialized network on this problem (which takes 600 epochs).

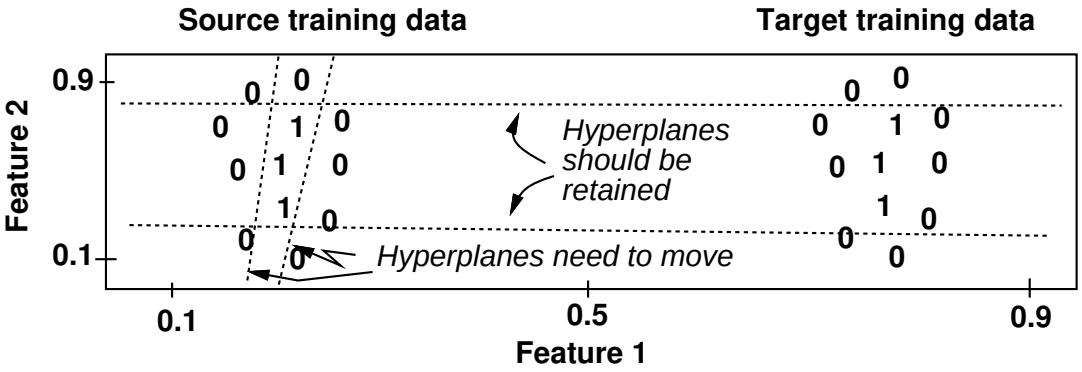


Figure 3.1: Problem illustrating the need for DBT.

The source and target tasks are identical, except that the target task has been shifted slightly along one axis, as represented by the training data shown here. Because of this shift, two of the source network hyperplanes are helpful in separating class-0 from class-1 data in the target task, and two are not.

data. The weights defining such hyperplanes often have high magnitudes [Dewan and Sontag, 1990]. Figure 3.2 shows a simulation of such a situation, where a hyperplane that has a high magnitude, as if it came from a source network, causes learning to be slowed down because it doesn't move into place.

As discussed in Section 1.2.3, high weight magnitudes from the source retard back-propagation learning on the target task because the weight update equation is not normalized for weight magnitudes. Also, this equation contains the factor $y_j(1 - y_j)$ (where y_j is a unit's activation), which is most often small for large weights.

As discussed in Section 1.2.1, weights can be interpreted as implementing hyperplane decision regions in feature space. Recall that the same hyperplane position can be implemented by different weight vectors. Therefore, it's possible to change weights without moving the hyperplane they define.

Since we've seen that high weight magnitudes interfere with learning, and that magnitudes can be altered without moving hyperplanes, it might at first appear that a simple solution to the problem with literal transfer is to uniformly lower all weight magnitudes. However, we have also observed that hyperplanes in good positions will move out of those positions unless they are given high enough weight magnitudes to remain in place.

To address both of these problems, it is necessary to rescale hyperplanes so that useful ones are defined by high-magnitude weights and less useful hyperplanes receive low magnitudes. In order to implement such a method, we must first develop a metric for evaluating hyperplane utility. Figure 3.1 gave an intuition for what it means for a hyperplane to be "separating" (and thus, by assumption, useful) or not. We now describe a metric for quantifying how well the data can be separated.

Chapter 3
The DBT algorithm

As discussed in Chapter 1, shifting distributions of training data are common in practical settings, such as when a method for diagnosing heart disease is developed using patients at one hospital and is to be used at another. It is reasonable to believe both that the previous results are relevant and that they also may need to be changed somewhat to accommodate the new situation. The discriminability-based transfer (DBT) algorithm, introduced in Section 1.3, addresses this situation by providing a mechanism that allows neural network weights to be used on a new task. DBT addresses the specific formulation of the transfer problem that was presented in Section 1.4. It allows weights from a source neural network to be transferred to the initial conditions of a target network for a related task.

In this chapter we present DBT in full detail, along with an example of its application. Then, in Chapter 4, we'll describe how we evaluated the effectiveness of DBT, and give empirical results of tests of the DBT method.

We begin, in Section 3.1, by describing what goes wrong with literal transfer, where weights are copied without modification from the source to the target. In Section 3.2, we describe the IM metric, which is used by DBT to evaluate which parts of the source network are relevant to the target task. We'll then show the DBT algorithm in Section 3.3. The remainder of this chapter will explain more details of this method and give an example of its application.

3.1 What can go wrong with literal transfer

Section 2.2 cited several studies that explored the feasibility of literal transfer, where weights are simply copied from the final conditions of the source network to the initial conditions of the target network. Those studies showed that randomly initializing the target network can give better performance and learning speed than when literal transfer is used to copy the source network weights. This result may seem non-intuitive: why is it that copying weights from a related task may interfere with training?

To understand this problem, consider the situation when only a subset of the source network input-to-hidden (IH) layer hyperplanes are relevant to the target problem, as illustrated by the artificially constructed task shown in Figure 3.1. Here, only two of the hyperplanes used to solve the source problem are relevant for the target task; the remaining two will need to move in order to classify the target data.

As shown in the foundational study for this dissertation that was described in [Pratt, 1993], some hyperplanes initialized by source network training don't shift out of their initial positions, despite the fact that they don't help to separate the target training

System	Source Task	Target Task	Goal: psych. or app?	Entry pt. into target	Targ. method	Retain source perf?	Benefits of xfer	Source training method
KBANN [Towell *et al.*, 1990]	(DNA pattern recognition)	DNA pattern recognition	app	initial wts, topology	BP	N/A	(1),(2)	(K. from expert)
Modular methods (Waibel, Pratt)	various, including phoneme recognition	various tasks with more output classes than source	app	pre-trained weight subset	BP	Y	(1)	BP
EBNN	robot operator	robot Q learning	app	"analytic" network update	BP + analytic method	N	(1)	BP
EBNN*	robot operator	robot Q learning	app	update of instance-based method	nearest-neighbor-like method	N	(1)	BP
Sharkey &Sharkey	3-letter nonsense words	3-letter nonsense words	both	initial wts.	BP	N	(1)	BP
MNN	(various artificially constructed tasks)	(various artificially constructed tasks)	app	modified wt. update rule	modified BP	N	(1),(3)	modified BP
A. Agarwal	2-class, 2-input training data set	same set w/extra training items	app	weights from source	RAP	Y	(1)	RAP
Martin, 1988	Random binary vectors	Random binary vectors	app	initial weights	BP	Y	(2)	BP
Mccloskey	arithmetic problems	arithmetic problems	psych	initial wts	BP	Y	–	BP

Table 2.1: Summary of various studies addressing transfer-like problems in neural networks.
The "System" column shows a variety of methods that address transfer or transfer-like problems. The "Goal: psych or app?" column indicates whether the primary goal of the indicated work was applications or cognitive modelling. The "Entry Pt. into target" column indicates how information transferred from the source was used to affect the target network. The "Retain source perf?" column indicates whether retaining the source task's performance was an important criterion or not. The "Benefit(s) of transfer" column indicates whether transfer was used to improve learning speed (1), performance (2), or to avoid local minima (3).

indicates that it may be possible to transfer HO weights as well as the IH weights, which are all that are addressed in this dissertation.

2.2.10 Summary

We have described several research projects that address problems that might be viewed as transfer, though their problem formulation differs from ours. Table 2.1 summarizes these studies. The primary difference between the method presented in this dissertation and in all of these studies is that our approach is more carefully and thoroughly empirically validated. Our transfer method also differs in that we don't focus on retaining the source task, we are not worried about cognitive modelling, and we retain the back-propagation algorithm.

2.3 Conclusion

This chapter has presented background material for classification learning systems. It has also discussed research that's related to the idea of transfer. Chapter 3 now presents the DBT transfer algorithm in full detail.

[Thrun and Mitchell, 1993] describe two systems: EBNN, where the target learner is a back-propagation network (and so this is a transfer scenario) and EBNN*, where the target learner is a specialized instance-based learning method that explicitly stores each training example that it's given. This latter paradigm is of less interest, because it is very space- and time-intensive for problems with large numbers of training examples. However, empirical studies on EBNN, which did implement transfer, indicated that it is very sensitive to the "quality" of the source networks – when source networks aren't trained well, asymptotic performance and learning speed are negatively affected. Tests of EBNN*, on the other hand, indicated that it learns to the same asymptote at a faster rate when well-trained source networks are used, and that when a badly trained source network is used, it doesn't do worse than if the source network wasn't used at all.

The description of the empirical studies does not address any control for number of parameters, initial conditions, learning rate, or momentum factors. Evaluation was based on a single robot task, and measurement of learning speed was based on training, and not on testing, data. There was no quantification of learning speed-up; instead learning curves were shown in several graphs.

2.2.9 Noel and Amanda Sharkey: Adaptive generalisation and the transfer of knowledge

[Sharkey and Sharkey, 1992] extend the work in this dissertation by presenting the results of experiments on literal transfer between related tasks. In particular, they tested the effectiveness of literal transfer of just HO weights, and showed that this can speed up target task learning, at least in a simple network for an artificially constructed task. Sharkey and Sharkey study a "paired-associate" learning task, where the networks are trained to associate pairs of 3-letter nonsense words represented by a small training set (< 10 patterns). Transfer is between networks that have the same input words, but different output words. Another study explored target networks trained on a task with the same output words, but different input words from the source.

The authors found that, when the output patterns were held constant and the input patterns were changed between source and target tasks, learning speed-up in the target was observed. However, when the input patterns were held constant and the output patterns were changed, this interfered with learning on the target task to such an extent that it was better not to use the source at all (called "negative transfer"). This is consistent with our view of generalization in back-propagation networks: they create a region of feature space around the training data patterns of a particular class that are given the same response, and which include the training data patterns as a subset. In transfer, as in generalization, rough decision regions are retained and are useful in the new task. This explains the observation of positive transfer when input values were perturbed. However, when new output classes are to be associated with the same input regions, negative transfer was observed.

Another study explored the effect on transfer when the input weights were gradually perturbed until they were completely randomized, while HO weights from the source were held constant. Positive transfer was observed in all networks that were run. This

The source network is trained several times from different starting conditions. Each member of a set of MNN's (one for each hidden unit) is given initial and final ("optimal") weights from a trained source network as its training data.

A modified back-propagation rule is used to train a related task. At each epoch, all target training data patterns are analyzed to determine which is most consistent with the direction used in source training. The MNN then provides a step size and direction in which to change weights in the target.

Empirical results showed that the number of target training epochs could be reduced using an MNN, compared to a standard multi-layer perceptron. Evaluation problems were small artificially constructed tasks, and all evaluation was on training, not generalization, data. The authors do not mention any effort to control for HU count, learning rate, or momentum in the networks that were compared, so these results must be viewed as preliminary.

2.2.7 A. Agarwal *et al.*: An On-line algorithm to Overcome Catastrophic Forgetting

[Agarwal *et al.*, 1992] describe an algorithm for perturbing hidden unit hyperplanes in order to accommodate new target data. Their paradigm is that source performance must be retained. The key idea of this work is that it is possible to move a hyperplane so that it continues to properly separate a set of old training data, but that it better accommodates a new set of target data. It's not clear how general or applicable this idea is, however, because the method is only tested on a very simple 2-class, 2-input artificially constructed problem, where it is shown to reduce the number of target training iterations. Hyperplanes are updated individually, in the HO layer, then in the IH layer if a misclassification remains. No mechanism is described for handling the situation where misclassifications exist after IH hyperplane perturbation, or for coordinating hyperplane update. This method also uses a new procedure, called RAP, for weight updates. Evaluation is on training and not testing data, and there is no discussion of how the experiments were controlled for factors like hidden unit count, learning rate, initial conditions, or momentum. Again, therefore, this work must be viewed as preliminary.

2.2.8 S. Thrun and T. Mitchell: Integrating Inductive and Explanation-Based Learning

[Thrun and Mitchell, 1993] describe a method for extracting information from networks that implement robot operators and inserting it into an inductive learner whose task is to estimate how likely a particular action applied to a particular state is to lead to a goal. This target task is called "Q-learning"; a set of "Q" estimators for all operators implements a "policy" for achieving a particular goal. The primary insight in this work is that it is possible to calculate the derivative of input units with respect to output units in a source network, and that this "slope" information may be transferred to a new network to help in Q-function learning.

2. **Glue training:** Combine subnetworks into a larger network using additional "glue" units. These are trained while subnetwork weights are frozen. These two steps correspond to the process of further training on the target task in our transfer framework.

3. **Fine tuning:** All weights are modified during further training.

The performance obtained by [Waibel *et al.*, 1989] for a decomposed network was slightly better than performance of a "monolithic" (non-decomposed) network. In addition, there was a marked decrease in learning time. [Hataoka and Waibel, 1989] extended this work to a network for recognizing all consonants.

[Waibel *et al.*, 1989] and [Hataoka and Waibel, 1989] did not, however, perform tests to evaluate the degree to which learning speed or performance improvement due to problem decomposition was dependent on a network's initial weights. Since it's been shown that networks can give very different behavior dependent on initial conditions (cf. [Kolen and Pollack, 1990]), it is important to determine whether the improvement observed was due to that effect or to problem decomposition.

This work was extended [Pratt and Kamm, 1991, Pratt *et al.*, 1991, Pratt, 1993]. These studies did test for the statistical significance of performance and learning speed results over multiple initial conditions, and found, in support of [Waibel *et al.*, 1989], that there was no significant performance improvement, but that learning speed was substantially improved. Also, these studies used larger networks on different tasks, and a different network architecture than [Waibel *et al.*, 1989].

2.2.5 This study, in comparison to literal transfer and modular network construction

So far, we have described neural network techniques that implement literal transfer, where weights are copied without modification from a source network to be used as the initial conditions for a target task. In comparison to this prior work, this dissertation studies how weights can be adjusted as they're transferred from the source to the target task, in order to overcome the problems encountered in literal transfer. In comparison to work on modular network construction, network input features and output classes are the same in the transfer scenarios that we study.

Since the time that a working paper based on this thesis was first distributed, four papers have been published that pursue the idea of transfer in different problem formulations from ours. We now review each of these papers in turn.

2.2.6 D. K. Naik *et al.*: Meta-Neural Network approach to learning by learning

[Naik *et al.*, 1992] describe a mechanism for a "meta" neural network (MNN) to "observe" neural network training on one task so that it can provide information to a modified weight update rule on a related task, in order to speed up target learning.

2.2.2 Studies of source task interference via literal transfer

Previous exploration of literal transfer has studied the degree to which source learning can interfere with neural network learning on a target task. It has been shown for several problems that the degree of interference is so large that it's better not to use any source information (cf. [Martin, 1988], [Tenorio and Lee, 1989]).

2.2.3 Retaining source task performance

The cascade-correlation architecture of [Fahlman and Lebiere, 1990] addresses this problem by prohibiting nodes from changing their weights after they are learned. Fahlman's more recent work capitalizes on this property by training a network for a Morse code memorization task by presenting data in chunks of increasing complexity. This work differs from ours in that it builds network performance incrementally – previously learned mappings must be retained. It's also questionable how well this technique could be used for transfer, since it does not include a mechanism for discarding newly irrelevant parts of the network.

Fahlman's approach has been called *sequential learning* by previous authors – here the goal is to retain source task performance after literal transfer to the target task, even though target training data doesn't reflect the source task. Other studies of sequential learning [McCloskey and Cohen, 1989, Kruschke, 1990] have been from a cognitive modelling perspective. Although these studies did achieve some success in their goal of modelling cognitive processes, the algorithms described weren't evaluated on real-world tasks nor on simpler problems that were related to real-world tasks in some known way.

2.2.4 Modular network construction

Waibel *et al.* [1989] introduced the idea of problem decomposition in neural networks. That study attempted to extend a network trained to recognize the consonants /b,d,g/ to one that recognized all "stop" consonants /b,d,g,p,t,k/. Waibel *et al.* observed that a straightforward approach to this procedure, where new weights and nodes are added to the smaller network, increased learning time more than linearly with problem complexity. To alleviate this problem, several techniques were used to combine the weights from pre-trained /b,d,g/ and /p,t,k/ subnetworks. This process can be viewed as literal transfer of weights from the smaller to the larger network, which handles the full set of output classes. In contrast to the sequential learning methods described above, here the new network has different output classes and the target training data is a superset of the source.

Waibel *et al.*'s best performance was obtained when the weights on links between the input layer and the first hidden layer of the pre-trained subnetworks were used to initialize a larger network with two hidden layers of nodes. The first hidden layer was supplemented by several "glue" units whose purpose was to combine the results from the subnetworks. The following training steps were used:

1. **Subnetwork training**: Divide network into subsets; train each individually. This approach corresponds to source training in our transfer framework.

2.2 Previous approaches to transfer in neural networks

Section 2.1 included a discussion of related work in machine learning that provided a basis for the idea of transfer. Here, we briefly discuss the relationship of transfer to statistical pattern recognition, and then turn to recent work in neural networks that can be viewed as studying different formulations of the transfer problem than our own.

To the best of our knowledge, the problem of how best to incorporate information from a previously trained classifier in a new task has not been explicitly explored in the pattern recognition community. However, despite this fact, it is nonetheless true that classifiers are often used on populations that differ somewhat from those on which they were originally constructed.

Aside from the work on analogy discussed in Section 2.1, we are not aware of studies in the symbolic machine learning community that explicitly address transfer. We view this as a significant open problem, with opportunities for further research for other algorithms besides neural networks.

In the field of neural networks, some studies have been performed that can be characterized as studying transfer. Several of these studies were for the purpose of cognitive modelling via neural networks, and not for applications. Therefore, their criteria for success were not that transfer helped in target task learning, but rather that the same kind of transfer interference effects viewed in humans were observed in their models. For example, [Shrager and Blumenthal, 1992] studied transfer in simple back-propagation networks for simulating a child learning how to bake muffins from its parent. More studies of this nature are cited below.

A widely explored related paradigm is that of "adaptive" learning, where the network is able to incorporate new training data from the same population as it was originally trained (cf. [Carpenter and Grossberg, 1991]). In these systems, use of the classifier alternates with further training. In contrast, the transfer formulation considered here involves only one movement of information from the source to the target task, and the target task is different from the source (it's not just the source with more training data added). Furthermore, we focus specifically on the criterion of training on the target task; source task performance is not important in our formulation.

2.2.1 Literal transfer

Back-propagation is an iterative learning method, meaning that it constructs a full classifier at each step of the training process. During training, the classifier gradually improves its performance. A straightforward way to do transfer with an iterative method is simply to change the training data, and let the network continue to adjust to the new concept. We call this process *literal transfer*, because there is no modification to the weights to accommodate the new concept to be learned. Instead the weights can be viewed as being copied, without change, from a network for the source, to one for the target task.

domain theory.

2.1.2 Analogy

The idea of transfer presented here has strong conceptual ties to the idea of analogy that's studied in symbolic machine learning (see [Hall, 1988] for a survey). There are a wide variety of methods in that field that have been proposed for using information from a source problem to aid in solving a related target task. This is exactly the intuition behind transfer. The main point of contrast between the two bodies of work is that the approach described here uses analogy to aid in inductive learning, meaning that it assumes the availability of a set of training examples that represent the target task. In contrast, machine learning analogy methods are not integrated with an inductive component. In our formalism, this allows the system to be robust against a faulty analogy source – the inductive component can "take over" when the source information is not adequate. A secondary point of contrast is that we use a neural network representation whereas symbolic machine learning analogy methods are usually based on logical representations.

2.1.3 KBANN vs. Transfer

A paradigm that's related to transfer is embodied by the KBANN system of Towell *et al.* [1990]. Instead of a source network, KBANN uses a formal domain theory (represented in propositional logic) to determine network topology and to pre-set network weights. It improves both performance and learning speed on the target task.

There are at least two potential limitations to the KBANN approach:

1. Domain knowledge for a problem may not be readily available (the *knowledge acquisition bottleneck* ([Barr *et al.*, 1989, page 178], [Hayes-Roth *et al.*, 1983]) is a well-known problem in AI research).

2. Useful knowledge may not be readily expressible in terms of the propositional logic rules required as input to KBANN. In particular, KBANN inputs must be binary valued.

The neural network transfer paradigm studied here suggests a way to avoid these problems. As classifiers are constructed for many tasks, they should be able to build on previous results instead of starting from scratch with every training run. Therefore, the source of information used to help with neural network training – namely other neural networks – should be more readily available. Our approach deals with the second problem by avoiding the representational shift and reliance on binary-coded inputs required by systems that require the use of logical rules. Instead, the source network's representational formalism of neural network weights (see Section 1.1) is the same as the representation used in the target network.

less quantifiable) criterion, as well.

The study of automated methods for constructing classifiers began in the field of statistics (cf. [Fisher, 1936], who introduced the use of discriminant functions). The study of classifiers continued to be addressed by statistical approaches within the field of pattern recognition [Duda and Hart, 1973, Fukunaga, 1972].

Less statistically oriented approaches like that of [McCulloch and Pitts, 1943] studied classification devices that fit specific constraints meant to mimic information processing in the human brain. These models allowed for parallel computation, and performed simple thresholding operations. In contrast to many statistical techniques, where a discriminant function was analytically calculated from a set of training data in a single step, these methods were iterative – they improved their performance over time.

The statistical and neural approaches were combined through the 1960's, and were synthesized in two important books: [Nilsson, 1965, Minsky and Papert, 1969]. Around 1969, interest in the neural approach waned because of the limited results that could be obtained using the methods of the time. A new wave of research began in the 1980's, starting with the publication of [Hopfield, 1982] (which used neural networks for optimization instead of classification). Chapter 7 of [Rumelhart *et al.*, 1987b] presented the now popular back-propagation algorithm, which had previously been independently developed by several other researchers [Werbos, 1974, Parker, 1982, Le Cun, 1986]. Today, the level of interest in neural approaches remains high.

2.1.1 Machine learning research on supplementing classifier learning with additional information

The idea of neural network transfer was inspired by studies in the field of symbolic machine learning that explore how a learner's training data can be supplemented by using *domain knowledge* – a set of logical rules describing aspects of the problem at hand. For example, a rule for a "chair recognition" classifier might express that all chairs have seats, and that some are made of wood. Several systems include both an Explanation-Based Learning (EBL) [Mitchell *et al.*, 1985] component that facilitates the use of domain knowledge, as well as an inductive component that learns from training data. These components can communicate in two directions, in which case an EBL domain theory is used to focus the use of the training data, and the inductive learner is capable of generating new rules in the language of the domain theory (OCCAM [Pazzani, 1989], GEMINI [Danyluk, 1989], IOU [Mooney and Ourston, 1989]). Several systems also transfer knowledge in a single direction, using domain knowledge to provide input to a process that creates new features for inductive learning. These include STABB [Utgoff, 1986], Zenith [Fawcett and Utgoff, 1991] and CINDI [Callan and Utgoff, 1991]. Also, the INTSUM procedure of META-Dendral [Buchanan and Mitchell, 1978] uses a "half-order" domain theory as input to inductive learning; IOE [Flann and Dietterich, 1989] uses both EBL and inductive methods in a single process to develop a concept; and Cohen [1990] does induction over generalizations of EBG rules. Each of these approaches studies how training data can be supplemented through the use of a

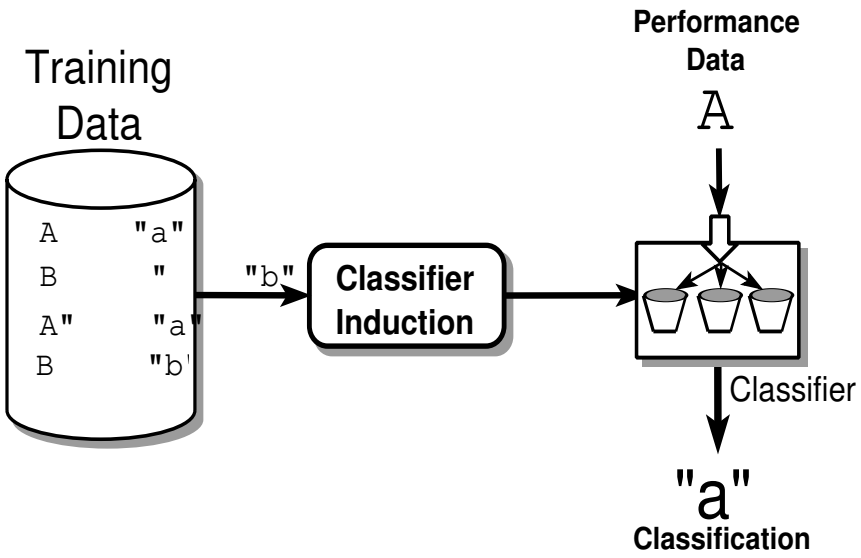


Figure 2.1: A general framework for classifier induction.

See text for discussion.

should be able to classify new instances drawn from the same distribution as the original training data, but not necessarily contained in that dataset. For example, a letter "A" written in a new hand should be recognizable as an A. The process of using the trained classifier is called *performance* or *classification*. The ability to correctly classify data not in the training set is called *generalization*. We can estimate the generalization error rate of a classifier by evaluating the classifier on a set of data called a *test set*, which is distinct from the training set.

The general principle used by classifier induction algorithms is that they should learn how to give the response specified by training data patterns, but that learning is constrained, usually by restricting memory, so that the classifier is prevented from simply memorizing the training data in a look-up table. Such a solution would also frequently (though not always) produce bad generalization performance (estimated by scores on testing data); this is called *overfitting*. Instead, the learner must extract more general regularities that distinguish the different classes; these regularities should then hold for testing as well as training data. Good generalization performance is the goal of classifier induction, because in general we are interested in obtaining a classifier that can handle data that wasn't in the training set. If the only data that needs to be handled is in the training set, then a simple look-up table would have sufficed, and classifier induction is not needed.

There are several criteria for evaluating a classifier induction method. Most important is classifier *performance* on the testing data, measured as the number or proportion of test data patterns correctly classified. Performance on training data is sometimes measured as an (often inaccurate) indicator of test data performance. A second criterion for evaluating the induction algorithm is the amount of time required for learning. Ease of construction and training of the classifier induction system is an important (if

Chapter 2
Background and Related work

This chapter reviews systems for classifier induction and their relationship to the idea of transfer (Section 2.1) as well as neural network research that's related to transfer (Section 2.2).

2.1 Classifier induction

Classifiers take as input a set of *features*[1] representing some object, and map them into one of several discrete *classes*. A feature is a measurable aspect of an object, like a patient's blood pressure or the nucleic acid present at a particular position on a DNA strand. Classes are groupings of objects, based on feature information, that are of interest for some task. An example class might be all patients at risk for heart disease, or all DNA subsequences that serve to delimit genes. Examples of domains that may include a classifier component are speech recognition (where a classifier might be used for mapping a speech signal into a phoneme (discrete speech sound)), the recognition of handwritten characters, and various diagnosis tasks (like converting a set of clinical results into a determination of the presence or absence of heart disease in a patient). *Classifier Induction* algorithms automate the construction of classifiers. Classifier induction allows a classifier to be built from a set of examples of classifier input and output – an approach that's potentially less costly than traditional programming or expert system design. A framework for classifier induction is shown in Figure 2.1, which is discussed further below.

The induction process is usually referred to as *learning* or *training*[2] and so the data used as input are called *training data*, a *training set*, or *training patterns*. The training set contains example classifier input values (features) and the corresponding correct desired output category.[3] Figure 2.1 shows a set of training data that might be used for learning to distinguish the handwritten characters "A" and "B". Attributes for this task might be as simple as individual pixel values, or the results of more complicated routines that detect edges or line intersections. The classifier resulting from the training process

---

[1] Features are also sometimes called *attributes* or *inputs*. Note that the classifiers we consider here take a fixed set of features as input, though this isn't true in all classifier learning systems.

[2] Classifier induction also goes by the names of *classifier learning*, *empirical learning* (cf. [Hirsh, 1988]) and *similarity-based learning* [Michalski, 1986].

[3] Note that classifier induction can be sped up, or classifier performance improved, by *preprocessing* the input data to change its representation. For example, a common data preprocessing method in speech recognition is to apply a Fourier transform to an acoustic waveform to convert it into a spectro-temporal representation [Oppenheim and Schafer, 1975].

1.5 Thesis of the dissertation

The primary claim made by this dissertation is as follows:

> **Primary claim:** DBT demonstrates a learning speed improvement relative to randomly initialized networks.

We'll demonstrate this claim via the empirical results to be presented in Chapter 4. We qualify the primary claim as follows:

> **Qualification:** The primary claim holds for a non-trivial set of relationships between the source and target tasks.

We'll discuss these relationships in Chapters 4 and 5. We'll also show evidence to support an additional claim:

> **Additional claim:** Networks initialized via literal transfer exhibit worse average asymptotic performance than randomly initialized networks on some tasks, and so this approach is less robust than DBT.

This claim will also be supported by the results of Chapter 4. Section 4.8.1 will show how the results of Chapter 4 support these claims. We'll also summarize the support we've gathered in Section 6.1.

1.6 Reader's guide to the remainder of the thesis

Chapter 2 now presents more background to classification methods in general, as well as a review of work that's related to transfer. Chapter 3 then presents the DBT algorithm in full detail. Chapter 4 describes how DBT was evaluated in order to demonstrate that it reliably improves learning speed. We analyze these results in Chapter 5, and then conclude in Chapter 6.

high IM values, simply measuring the average IM of source network hyperplanes on target training data can give a rough determination of relevance. For the seven tasks studied, we observed some correlation between IM and eventual DBT superiority over randomly initialized networks, as reported in Chapter 5. These results are not significant, however, since they were only over the seven tasks studied. Even with a more extensive body of test results, however, the correlation between IM and eventual performance will remain, at best, an approximate predictor of DBT success until methods are developed to estimate final backpropagation performance, given initial conditions. Chapter 5 will argue that the development of more accurate methods to predict the effectiveness of transfer via DBT awaits more fundamental results on back-propagation that characterize learning time, based on initial conditions.

Relative complexity

In the empirical studies in this dissertation, we will focus on source and target tasks that have the same complexity, measured in terms of the number of decision surfaces required for accurate classification. This choice was made to simplify the analysis and empirical methodology used for training, and not because we believe that it adequately represents the full range of situations in which transfer should be useful.

Although we have not done more than preliminary empirical testing for source and target networks with different complexities, it's our belief that the current formulation of DBT should speed up learning in these cases. If there are too few hyperplanes provided by the source for the target task, then the target network should be able to utilize the ones that it is given, using extra randomly initialized hyperplanes for additional complexity. These will appear to the target network just as do irrelevant source hyperplanes from a larger source network – they will have low magnitudes and will be able to move into position. If the target task requires fewer hyperplanes than the source task, it should still be able to utilize the ones that it is actually given.

The consequences of our choice to test on only same-complexity source and targets is that the above assumptions could be wrong – there could be unanticipated interactions that arise when source and target tasks are of different complexities. We believe, based on preliminary testing and our understanding of hyperplane dynamics, that this is not the case.

1.4.8 Summary of problem formulation

To summarize, this dissertation explores the problem of *transfer* between classifiers, where information from a trained **source network** is used to supplement **target network** training. We focus on the goal of improving the **learning speed** for a **neural network** inductive learning algorithm called **back-propagation**. We use the transferred information to modify the **initial weights** of the target task, which are then trained further using back-propagation. Our goal is to speed up learning on the target task, even at the expense of target network performance on the source task. The DBT algorithm does **non-literal** transfer from networks trained on related tasks.

more so) as changing initial conditions. However, the effect and degree of effect of these entry points are even less well understood than that of the initial weights.

1.4.6 Retention of source performance

In a transfer scenario, we may or may not be interested in retaining performance on the source task. We focus on the latter paradigm here, where our goal is simply to speed up training on the target task, not to supplement the source task network. Since source task performance may interfere with the target task, for some situations it's necessary and appropriate to abandon the criterion of retaining source task performance.

Ignoring the criterion of performing well on the source task is also a reasonable assumption in situations where both source and target tasks are important. In this case, two networks can be used – one specialized to each of the source and target problems. A consequence of this decision is that we've precluded situations where it is not easy to determine whether incoming data should be routed to the source or the target network. In these cases, a single network would probably be more feasible, though it could be argued that such a network would embody an ability to tell source from target task. Another approach to using the method presented here for retaining source performance is to define the target task to be a superset of the source task, and to provide training data accordingly. This is in fact the case in two of our benchmark problems of Chapter 4 – "Chess" and "DNA". This approach may be more costly, however, than if the network was constrained to retain the source task without the necessity of continued source training data presentation.

1.4.7 Relationship of source and target tasks

Quantifying relevance of source to target task

The seven transfer tasks for which we'll present empirical results in Chapter 4 were chosen because, based on our knowledge of the domains from which they were drawn, it appeared that there were at least two closely related tasks between which transfer might be possible. For instance, our intuition tells us that we use the same mechanisms to understand male and female speech, and so we might consider these to be related tasks. Yet since male and female speech are most frequently clearly different, we suspect that a network trained on only male speech might have trouble if simply applied to females, without further training.

In a practical situation, obtaining a source network might be expensive, so it might be helpful to develop more principled methods than this use of human intuition for assessing source/target task relevance before source training is performed or source weights are available. This might be done using formal domain knowledge about task similarities and differences.

An alternative approach is to analyze source and target training data and source weights, if available, to ascertain relevance. Recall from Section 1.3 that we use a metric called IM as part of the transfer process, to measure the degree of relevance of each source hyperplane to the target task. Since DBT attempts to retain hyperplanes with

1.4.5 Initial weights as entry point to Back-propagation

When people construct back-propagation networks for classifier induction, they bring to bear both their understanding of the particular domain and experience with networks for related tasks in order to make various choices, such as:

- Network topology (cf. [Lang, 1990], [Towell *et al.*, 1990])
- Representation of input unit values (preprocessing)
- Node activation functions (cf. [Moody and Darken, 1989])
- Objective function (cf. [Hanson and Pratt, 1989], [Lang, 1990], [Thrun and Mitchell, 1993], [Naik *et al.*, 1992])
- Representation of target unit values (postprocessing) (cf. [Dietterich and Bakiri, 1991])
- Training data order (cf. [Kruschke, 1990])
- Learning parameters η, α (many studies, where these parameters are chosen based on their success on previous problems)
- **Initial network weights** [Towell *et al.*, 1990] (this is the approach used here)

Various research efforts can be characterized as removing the necessity for the system designer to make these choices (cf. [Fahlman and Lebiere, 1990], [Chan and Fallside, 1987]) by choosing good values based on the training data.

However, to the extent that information is available to supplement the training data (such as, in our case, a trained source network), it makes sense not to automate the means by which these choices are made based on training data alone, but to investigate methods for translating this supplemental information into these choices. In this case, the above aspects of BP constitute the entry points by which learning bias (i.e. information that supplements the training data [Mitchell, 1980]) can be incorporated into both the source and the target network. For example, we may have an automated scheme for determining network topology by estimating problem complexity from the target training data (cf. [Fahlman and Lebiere, 1990], [Ramachandran and Pratt, 1992]), but if we are given information that supplements a possibly misleading set of training data that specifies the proper complexity, then it might make sense to ignore the complexity indicated by the training data.

Each of these entry points can be used to inject information into a target network in a transfer scenario. As indicated, our focus is on changing the initial weights of the target network. We might have chosen any of the other entry points as well (for example, KBANN [Towell *et al.*, 1990] uses domain knowledge to change both initial weights and network topology). We chose to use initial weights in our study because of previous work (cf. [Kolen and Pollack, 1990]) that demonstrated huge differences in back-propagation training based on initial conditions. Furthermore, in an iterative method like back-propagation, "initial conditions" can be arbitrarily close to a good solution!

The consequences of this choice for transfer is that it may ignore other important ways of inserting knowledge into neural networks, which may be as effective (if not

- **Reduced number of training examples:** Information from a learned classifier might substitute for a subset of training examples, thereby eliminating their necessity.

- **Improved performance:** Classifier information may supplement a sparse (not enough data to adequately represent the concept) or noisy set of training examples in order to boost classification accuracy. We'll discuss this extension to our formulation of the problem in more detail in Section 6.3.1.

The consequences of not pursuing these other potential uses for transfer is that the relevance of the results of this dissertation is limited. There are many scenarios, especially for small problems, where learning speed is less of an issue than obtaining a network that gets the best performance possible. Our results are much more important for larger networks, where learning speed is more of a problem. If networks trained on related tasks could be used to help train a new network and to result in performance improvement as well, then this work would be more broadly helpful.

1.4.4 Transfer algorithm design decision: setting initial conditions

As described above, many classifier induction techniques are iterative in nature. They learn by constructing a series of classifiers, each of which gives better classification performance than the last. This property leads to an approach to transfer that is integrated with the induction algorithm: the transfer algorithm converts weights from the source classifier into initial conditions (weights) to be used in learning the target task. Target task training data is then incorporated via continued training using the same iterative learning method that was used for source training.

This is the approach taken in this dissertation. It can be contrasted with a hypothetical approach to transfer that only uses the learning algorithm for the source task, and then, through analysis of the target training data, modifies the source classifier directly into a target classifier, without further iterative training.

As described above, a guiding force in the design of the transfer methodology used here was to preserve as much as possible whatever characteristics of back-propagation have made it successful. This is why our method only changes back-propagation's initial weights. The downside of this choice is that we didn't consider alternative methods that don't involve iterative training on the target training data, and which may have resulted in even faster target task learning. For example, a transfer method that compared source and target training data and, in one step, moved the source hyperplanes to follow that data, might be faster than DBT followed by further training, as described here. The work of [Agarwal *et al.*, 1992] (reviewed in Section 2.2.7), though preliminary, is along these lines. It's our suspicion, however, that such a removal of iterative training on the target task would cause such a method to be be less robust – it might work in only limited domains. The empirically demonstrated benefit of using DBT (Chapter 4) is that DBT's average value is never worse than the baseline of random initialization on the target training data. So it's a more general, but perhaps a weaker, method than may in principle be possible.

other methods, they consistently trained slower than those methods. See, for example, [Shavlik *et al.*, 1991], who thanks the Condor system [Litzkow *et al.*, 1988] (which executes programs in parallel on idle workstations) for providing "many of the large number of cycles needed for these experiments. The equivalent of over two Sun-4/110 CPU *years* was provided by Condor"[emphasis in original].

Other authors have made this point as well. [Maren *et al.*, 1990, Page 92] says that:

> The need for lengthy training ... is one of the major limitations of [Back-propagation and Boltzmann machine] networks.

And, from [Waibel *et al.*, 1989]:

> A problem that emerges as we attempt to apply neural network models to the full speech recognition problem is the problem of scaling. Simply extending our networks to ever larger structures and retraining them soon exceeds the capabilities of even the fastest and largest of today's supercomputers

Many researchers have sought to speed up back-propagation. This work is summarized by [Hertz *et al.*, 1991, Page 120], who says that

> The basic [back-propagation] algorithm ... is exceedingly slow to converge in a multi-layer network, and many variations have been suggested to make it faster.

Fahlman's "quickprop" method is one such method that addresses back-propagation's slow learning speed. From his article [Fahlman, 1988]:

> The greatest single obstacle to the widespread use of connectionist learning networks in real-world applications is the slow speed at which the current algorithms learn.

In an applied setting, [Xu, 1990] describes a learning speed acceleration method for an network applied to sensor data processing. From [Xu, 1990, Page 11]:

> The learning is very slow in the standard back-propagation algorithm. For example, in the case of the [application discussed here], over one million examples need to be learned without one detection error, which could take an extremely long time, perhaps as much as *two years or more*, to "train" the net using the standard back-propagation algorithm on a high speed workstation [emphasis added].

Our experience [Pratt and Kamm, 1991] supports this claim– during the course of attempting to train a large network for speaker independent, continuous-speech, large-vocabulary phoneme recognition, the lengthy times to train networks (on the order of days) were a substantial impediment to research.

In addition, two recent U.S. patents have been awarded for learning speed improvements [Matsushita Electric Industrial, 1992, Sony Corporation, 1992].

Other research efforts for speeding up back-propagation include [Barnard and Cole, 1989], [Jacobs, 1988], [Piramuthu, 1992], and [Minai and Williams, 1992]. The book *Fast Learning and Invariant Object Recognition* [Soucek and The IRIS Group, 1992] also contains several papers on this subject.

Other potential benefits that might come from transfer, but which aren't studied in this dissertation, include:

Boltzmann Machine learning

Another neural network algorithm upon which transfer might in principle be based is the Boltzmann Machine [Ackley *et al.*, 1985], which learns weights in a network topology called a Hopfield network [Hopfield and Tank, 1986]. This algorithm handles classifier induction as well as pattern completion problems, since the identity of input and output neurons can change over time.

The primary difference between a Boltzmann Machine and a Back-Propagation network is in the learning algorithm. The Boltzmann Machine learning rule is based on thermodynamics, and involves a stochastic update rule that changes over time as a simulated temperature (representing the probability that an activation will take on a value not consistent with activations on connected nodes) decreases over time. Unfortunately, the Boltzmann machine learning method runs very slowly, and so it is not as widely used as back-propagation. Recent developments, however [Peterson and Anderson, 1987] have produced a faster Boltzmann Machine learning rule, though thorough empirical tests have yet to establish its superiority in learning speed, compared to back-propagation or other methods

1.4.3 Goal of transfer: improving learning speed

There are a number of target task goals that transfer from learned classifiers might help to achieve.

As mentioned above, our focus is on achieving faster learning through transfer. The general idea is that information from a classifier for a related task may be used to bias search through concept space by initializing an incremental learner close to a solution, or by introducing constraints that limit search.

Speeding up learning, even without asymptotic performance improvement, is important in some situations. Whether it is important depends on the particular practical setting in which learning is to take place. If a person or critical process must wait during learning, fast learning may make an important difference. For example, quick speaker adaptation may be required for a system (such as a telephone network) that recognizes speech from a constantly changing user population. Also, in a research setting, where several learning runs may need to be done, even small improvements in learning speed can multiply to produce a large improvement. For example, a neural network for protein structure determination (cf. [Qian and Sejnowski, 1988]) might require days to train without transfer, and only hours with transfer.

Furthermore, Judd [1988] has shown that neural network learning is NP-complete,[14] and so may take a long time in the worst case for large problems. It is therefore essential to develop heuristic methods to limit search.

Learning speed is also widely acknowledged to be a central problem in neural network training. As mentioned above, empirical comparison studies on back-propagation have found that, though networks often achieved as good or better performance than

[14] Judd's proof provides a worst-case analysis, so we must appeal to practical experience to back up these results.

Improved versions of back-propagation

Recently, several improvements to the back-propagation algorithm have been developed. One popular technique is to use an optimization procedure called Conjugate Gradient (CG) [Barnard and Cole, 1989] instead of the gradient descent procedure used to update back-propagation weights. Because CG only specifies a different ΔW value, learning is over the same architecture as a back-propagation network. Another recently popularized algorithm is Scott Fahlman's Cascade Correlation method [Fahlman and Lebiere, 1990], which determines its own network topology by constructing deep networks, with specialized, fixed hidden units.

Note that these and other methods all assume that the only source of information used during learning is a set of training data. This is in contrast to transfer, which uses an additional source of information for learning.

There were three reasons that we chose not to base our transfer method on these improved versions of back-propagation in this study:

1. By limiting our transfer method to only be compatible with, say, a conjugate gradient implementation of back-propagation, we would have also been limiting its generality. By using the design that we did, where only the initial conditions of a back-propagation network are changed, our transfer method can now be combined with any other method (CG or others) that learns over the same architecture.

2. Ordinary back-propagation is more widely used and accepted than these advanced methods, so this work is more widely immediately applicable.

3. Our own preliminary results with transfer and conjugate gradient training indicated that it was more prone to local minima than standard back-propagation.

The negative consequence of using unadorned back-propagation for our empirical studies is that our networks don't train as quickly as they might if another method was used. Since improved learning speed is our primary claim, it is important to explore, in future work, how transfer can be combined with faster versions of back-propagation.

The Perceptron Learning Rule

[Minsky and Papert, 1969] described the representational limitations of the Perceptron ([McCulloch and Pitts, 1943], Section 2.1), which were later overcome by multilayer learning methods like back-propagation. If our study of transfer had been based on the Perceptron algorithm, our results would have been limited to networks without a hidden layer, which can only handle problems for which enough feature preprocessing had been done to make them linearly separable.

A consequence of using our method instead of a perceptron-based technique is that there is no convergence proof for back-propagation. In general, it's not as well analyzed, and so it's very difficult to understand back-propagation's learning dynamics.

that the number of hyperplanes in a back-propagation network must be fixed before training – decision trees can in contrast "grow" as many hyperplanes as are necessary (note that some new neural network schemes [Hanson and Pratt, 1989, Fahlman, 1988, Ramachandran and Pratt, 1992] seek to overcome this limitation).

Finally, classification performance in a back-propagation network differs from a decision tree in that it may be much faster – limited to a number of parallel time steps equal to the number of weight layers (typically 2). Decision trees tend to be somewhat deeper. A downside to a neural network representation is that all features in a back-propagation network must be present before classification can be done (though see [Gallant, 1988]) – a decision tree only requires as many features to be input as will lead to a leaf node. [Pratt and Norton, 1990] discusses the learning, representational, and classification differences between decision trees and neural networks in more detail.

In addition to the negatives of back-propagation in comparison to decision trees discussed above is the fact that decision trees tend to be much easier to understand after they are constructed. A consequence of the choice of back-propagation for this research project is therefore that analysis of the resulting networks is harder than it would have been had a decision tree been used. It's our belief, however, that the idea of transfer can be applied to decision trees as well as to neural networks, and that this is an important research area that has yet to be explored.

As discussed above, the benefit of using back-propagation as the underlying algorithm for this dissertation is that it is widely used, and is therefore the subject of much interest in applied and research communities. However, the problems cited above with back-propagation in comparison to decision trees do limit the breadth and applicability of the results of transfer presented in this dissertation. In particular, if it's necessary to explain the behavior of the classifier in question, then a decision tree may be a more appropriate classifier model to use, and transfer techniques have yet to be developed for this formalism. Both neural networks and machine learning methods can be viewed as being based on transferable representations.

IM, instead of other information measures for hyperplanes

The DBT algorithm evaluates individual neural network hyperplanes using the IM metric. The studies described above that compare IM to other metrics such as GINI and twoing indicate that the usual metrics all provide good measures for discriminability, and we therefore picked the one with the longest history.

1.4.2 Back-Propagation, instead of other neural network algorithms

Neural network methods are characterized by the fact that they use a simplified computational model of the neuron cell found in human and animal brains [Rumelhart *et al.*, 1987b]. There are a variety of algorithms that have been constructed from this primitive, only some of which do classifier induction. In the previous section, we described why we chose to study the back-propagation neural network algorithm instead of some other non-neural methods. Here, we describe a few more neural network algorithms, and discuss the consequences of using back-propagation instead of them.

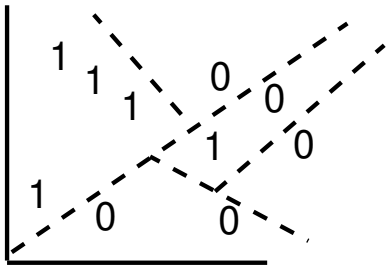


Figure 1.16: A typical arrangement of CART hyperplanes.

Note that, in contrast to ID3, here hyperplanes aren't necessarily axis-parallel.

[Breiman *et al.*, 1984] in terms of the shape of the function relating class proportion to the impurity measure – a good impurity function must be concave downwards, rather than linear. [Mingers, 1989] empirically confirmed the equivalence of several versions of the IM metric, the Chi-squared metric used by C4.5, the GINI criterion, and several other selection measures (see also [Buntine and Niblett, 1992]).

Back-propagation, compared to decision trees

Like ID3 and CART, the first layer of back-propagation weights can be interpreted as implementing hyperplane-shaped decision surfaces in feature space. As in CART, back-propagation decision surfaces can be arbitrary linear combinations of the input features. Unlike CART, hidden units have different activations depending on their distance from the hyperplane, which results in a smoothed decision surface for the network as a whole. As in C4.5, back-propagation is able to determine good splits on numerical features, but it goes a step further in that it also determines arbitrary linear combinations over the output values of these arbitrary splits. This is done by the second weight layer of a back-propagation network. It is here that back-propagation diverges substantially from decision tree methods, which only implement conjunctions of hyperplane decision surfaces. Representationally, this allows a single-hidden-layer back-propagation network to implement an arbitrary function of its inputs (though this may require an impractical number of hidden units) [Cybenko, 1989, Hornik *et al.*, 1989].

Back-propagation networks also differ from decision trees in terms of their learning procedure. Decision trees place and fix hyperplanes until the training data is separated. Back-propagation places a set of hyperplanes in random positions and then moves them iteratively into place. A benefit of this arrangement is that learning can be highly parallel [13], and so may be faster per iteration (though back-propagation takes so many iterations to learn that it usually learns much more slowly). Another downside is

[13] Although ID3 can in principle be written in a parallel language and run on parallel hardware, it's unlikely that the degree of parallelization could be as great as that in a neural network, where each neuron in a layer can be updated simultaneously.

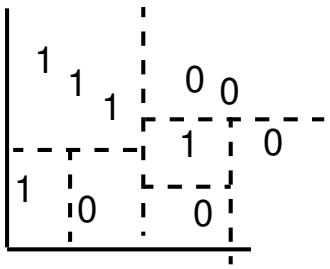


Figure 1.15: A typical arrangement of ID3 hyperplanes.

Note that all hyperplanes are axis-parallel, and that all hyperplanes don't cross the entire feature space -- instead each one subdivides the region created by previous hyperplanes.

Since its introduction in 1983, several improvements to ID3 have been developed. [Schlimmer and Fisher, 1986] and [Utgoff, 1988] present incremental versions (ID4 and ID5, respectively), which are able to handle a training data set that increases in size. Quinlan [Quinlan, 1987a, Quinlan, 1987b, Quinlan, 1988] has more recently created the C4 algorithm, which includes methods for extracting rules from the learned tree, and for pruning the tree (by removing hyperplanes) to handle noisy data. The latest version, C4.5, also includes a method for determining the best split on numerical-valued attributes. [Dattatreya and Kanal, 1985] gives an extensive survey of the use of decision trees in pattern recognition.

CART

The CART algorithm [Breiman *et al.*, 1984] also constructs a decision tree from a set of training data. As in ID3, CART places decision regions iteratively in feature space to separate training data of different classes. CART includes at least two improvements over ID3, however. First, CART includes a method for considering linear combinations of input features as data splits. Therefore some hyperplanes may not be axis-parallel. Secondly, like C4, CART includes a pruning method. A CART tree is pruned when the CART algorithm detects that the data has been overfit. Empirical testing to determine the point of overfitting is therefore an integral part of CART, so is discussed at length in [Breiman *et al.*, 1984]. A typical split of data from CART is shown in Figure 1.16.

The CART study also considered a number of different splitting rules besides the IM ("entropy") metric. The simplest metric considered was based on the idea of labelling a region of feature space with the most frequent class in that region. The proportion of training data elements in each region that would have been mislabelled using this method was then used as a measure of the "impurity" of that region, instead of entropy. At each step of the CART algorithm, the hyperplane which produced the greatest difference between before-split impurity and after-split impurity was chosen. This metric was found to produce inferior trees to the entropy metric, as well as to two others, called GINI and "twoing". No significant difference was found in the performance scores of trees constructed using entropy, GINI, and "twoing". These results are explained by

We now turn to a more detailed discussion of two widely studied machine learning algorithms for decision tree induction. We'll compare them to back-propagation and discuss the consequences of choosing back-propagation as an underlying classifier induction system instead of these methods.

ID3

The ID3 algorithm [Quinlan, 1983] constructs a classifier in the form of a *decision tree* by repeatedly placing linear decision surfaces (hyperplanes) in feature space. Decision surfaces are placed until all training data of different classes is separated by at least one hyperplane. Each hyperplane represents a split along a single input feature (or *attribute*) and so is parallel to a feature axis. A decision tree is created for ID3 by growing a tree down from the root, which originally represents all of the training data. At each tree level, the feature and split to be used for separating the data is chosen by evaluating the *Mutual Information* (IM) of that decision surface on the subset of data that it's to split.

As discussed in Section 1.3, mutual information is a metric borrowed from information theory. It was developed by [Shannon, 1949] (also see [Abramson, 1963]) as a means for quantifying the amount of information received in a coded message, and has its origins in the concept of entropy in thermodynamics. In the context of decision trees, IM is measured as the entropy of a set of training data before a message is received saying which side of a hyperplane split each data item is on, minus the entropy of the data after this information is received. Since high entropy corresponds to an even mix of classes, and low entropy corresponds to class data all in the same category, a hyperplane with a high associated IM value will be one that separates a previously uniform mix of classes into regions that contain only a single class each, as illustrated in Figure 1.14. Figure 1.15 shows a partition of feature space that might have been created by the ID3 algorithm. Note that each hyperplane subdivides the regions created by previous hyperplanes. In contrast, hyperplanes in a back-propagation neural network cross completely through feature space. As discussed in Section 1.3, the DBT algorithm, which forms the core of the transfer method presented in this dissertation, borrows the IM metric from ID3 and from information theory, and uses this metric to evaluate hyperplanes created by a back-propagation neural network. Section 3.2 gives more details about the IM metric.

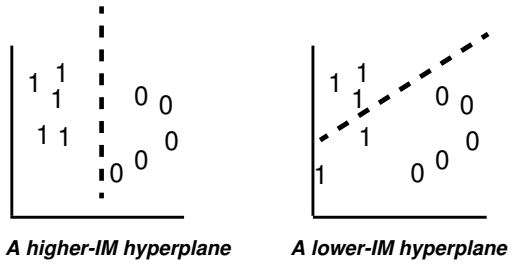


Figure 1.14: Good and bad hyperplanes, according to the IM metric

This representational flexibility is purchased at a cost, however. Disadvantages of back-propagation compared to classical statistical techniques include the fact that there are not clearly developed methods for utilizing prior class probabilities, that no convergence proof is known (as there is for the perceptron learning algorithm), and that training times are often lengthy. Furthermore, the algorithm is not as well understood as some older methods, and so there is a smaller body of theoretical knowledge upon which theorems scoping its effectiveness can be based.

Despite these problems, the back-propagation algorithm has been receiving considerable study over the past decade. Several journals and many recent books are devoted to exploration of the use of neural networks in general, and back-propagation in particular, on practical problems (Section 1.1). Although some of the success of this method can be attributed to technical reasons, there are some sociological factors at play as well. The idea of neural networks has a certain intuitive appeal to the layperson, and has received considerable press attention (cf. [Hinton, 1985, Feldman, 1985, Port, 1986, Allman, 1986, Gorman, 1988, Belsie, 1991]) and interest by funding agencies (cf. [DARPA, 1988]). Furthermore, several PC-based packages have made this technology more accessible to non-statisticians. Manuals for these packages are friendly and accessible to a nontechnical audience (cf. [Eberhart and Dobbins, 1990, Hecht-Nielson, 1989]; also [Laws, 1991] lists over 20 additional currently available PC-based neural network simulator packages).

It is because of this widespread interest in the back-propagation algorithm that it was a reasonable choice for an underlying method for this dissertation. A negative consequence of choosing this method, however, is that analysis of the scope of effectiveness of DBT transfer, since it's based on back-propagation, is very difficult. We will return to this point in Chapter 5, where we will argue that a precise determination of the range of effectiveness of the transfer method presented here awaits further fundamental results on the underlying back-propagation algorithm.

Symbolic machine learning

Transferable representations

Logic is used as an underlying representation in many symbolic machine learning systems. This use of a common language is based in part on a long term goal of the field to allow transfer between different learners. Rules obtained by one system are easily transferable to the next. Information can be transferred between decision trees as well, since they can be interpreted in terms of rules [Quinlan, 1987a].

A mechanism for transfer between neural networks is less immediately obvious, because it's less clear how weights in one network might be relevant to another. Viewing weights as representing decision surfaces addresses this question. Despite the fact that neural networks grew out of pattern recognition, where discriminating surfaces are central, a surprising number of neural network studies ignore this fact, and focus instead on the weights themselves as the units to be manipulated for purposes of skeletonisation, network understanding, etc. Our approach applies the idea of focusing on decision regions coded by weights to the problem of transfer.

To construct a parametric classifier, a set of training data is first examined to estimate parameters to the chosen distribution. In the Gaussian case, for example, these are mean and covariance matrices. These parameters completely determine a model probability distribution for the data. To find a discriminant surface between data of different classes that follows two or more different distributions, parameters are determined as above, then the probability density equations are solved to obtain an equal-density separating surface. This surface is then used as a discriminator for new data.

A *Bayes classifier* is a parametric method that is able to take conditional and prior probabilities of classes of data into account. Conditional densities can be modelled as following a Gaussian or other distribution, and can then be combined, using Bayes' law, to obtain an *a posteriori* density function specifying the probability of a given class, given a particular value for observed data. Discriminant surfaces can then be constructed using this function in order to obtain a classifier.

Nonparametric methods

In contrast, for a nonparametric classifier, a model for the discriminant surface (a line, for example) is chosen first. Then parameters for that discriminator are determined based on optimizing its relationship to the sample data. For example, the Fisher linear discriminant criterion function [Fisher, 1936] attempts to position a linear decision surface so as to maximize the ratio between within- and between-class scatter of points projected onto the line from different classes of data. Another nonparametric method is the nearest neighbor technique [Cover and Hart, 1967], where new patterns are classified based on the category of the closest classified pattern in feature space.

Nonparametric methods can be iterative or non-iterative. Non-iterative methods calculate the discriminant surface analytically in a single step. Iterative methods generate successive approximations to a criterion for good discrimination. A widely known iterative method for generating a linear decision surface to completely separate two classes of linearly separable data is the Perceptron Learning Algorithm [Rosenblatt, 1959]. In this method, the weights defining a linear surface are modified iteratively using training data until a separating line is found. This method is guaranteed to converge for linearly separable data.

Back-propagation, compared to statistical methods

The back-propagation neural network algorithm that we have chosen to study in this dissertation is another iterative nonparametric method. It is based on the perceptron learning algorithm [Rosenblatt, 1959]. Its advantage over that method is that, depending on the resources it is given, it is capable of learning not just a linear but an arbitrarily shaped decision region [Cybenko, 1989, Hornik *et al.*, 1989]. The IH layer in a back-propagation network can be roughly characterized as a piecewise linear discriminating surface, but a further layer of network weights combines the result of these discriminator functions using an additional approximately piecewise linear function, producing a more complicated decision surface.

modified so that they'll be changed during target training.

To quantify the amount that each source hyperplane is relevant to the target task, we borrow a metric, called Mutual Information, or IM, from decision tree induction [Quinlan, 1983], which borrowed it in turn from information theory [Shannon, 1949]. Given a set of training data and a decision surface that crosses through it, the IM function returns a value between 0 and 1 indicating the amount that the hyperplane helps to separate the data into different classes. For example, a high IM would be assigned to a hyperplane in a two class problem that separated a mixed group of training data into two regions, each of which contains only patterns of a single class. A hyperplane's IM value is used to determine its retentiveness during further training on the target task.

We'll discuss the IM metric in the context of decision trees in Section 1.4.1. We'll discuss the use of the IM metric and the DBT algorithm in more detail in Chapter 3. Chapter 4 will then present empirical results that compare the performance of DBT-initialized networks to those initialized via literal transfer, to target networks that have randomly initialized weights, and to the performance of source networks on target tasks.

1.4 Problem Formulation

In this section, we describe several of the choices made in the research project described by this dissertation. These choices include the use of a neural network, the use of back-propagation as a learning algorithm, and various decisions about aspects of the DBT algorithm. For each choice, we describe alternatives, argue for why the choice made was reasonable, and discuss the consequences of that choice.

1.4.1 Neural networks for classifier induction

Several different techniques, from a variety of fields, can be used for classifier learning. As discussed in Section 2.1, this problem was first explored by statisticians. The subfield of statistical pattern recognition emerged from within statistics, and later influenced the fields of neural networks and symbolic machine learning. All of these fields continue to study classifier learning today. In this section, we review classifier learning in each one. We include discussions of our rationale for choosing the back-propagation neural network algorithm as an underlying learning mechanism for transfer, as well as the consequences of that choice.

Statistical methods

Several different statistical methods are used to construct classifiers. Statistical pattern recognition techniques [Duda and Hart, 1973, Fukunaga, 1990] can be grouped into *parametric* and *nonparametric* methods. Parametric methods assume that the data to be classified follows a known distribution (often a Gaussian), whereas nonparametric methods assume, instead, that the classifier has a known form (a linear discriminant is a typical nonparametric method).

1.3 Overview of Discriminability Based Transfer (DBT)

So far, we've described the idea of transfer at a high level, and explored the effect of neural network weights on learning speed. In this section, we introduce the central algorithm of this thesis, called *Discriminability-Based Transfer* (DBT). Complete details of the DBT algorithm will be given in Chapter 3.

Figure 1.13 shows the framework in which DBT is used. We assume that a source network has been trained on some task that's related to the target task (we'll talk about what it means for a task to be related in Section 1.4.7). Weights are extracted from the trained source network, modified via DBT transfer, and used as the initial weights for back-propagation training on the target task. The widely used [Rumelhart *et al.*, 1987a] version of back-propagation is used for target training.

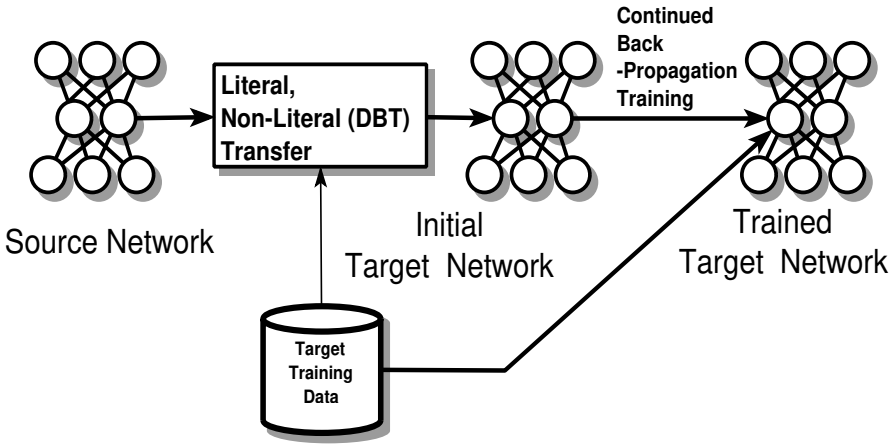


Figure 1.13: Transfer between neural networks.

Literal transfer in this framework refers to the process of simply copying the weights from the source to the initial conditions of target training, without modification. With literal transfer, we've found that hyperplane decision surfaces in the source network that are not relevant to the target task are not able to change to accommodate the target training data, because they have high weight magnitudes. This impedes learning, which usually leads to worse final performance [12] on the target task, as well as slower learning. We can't fix this problem by uniformly lowering all weights, because another problem that may occur is that hyperplane positions from the source network that are relevant to the target task are lost during training.

DBT overcomes these problems by determining which input-to-hidden layer hyperplanes are relevant to the target task, and which not. It modifies the weights associated with each hidden unit to make them more or less *retentive* during subsequent back-propagation training. Hyperplanes that are relevant to the target task are modified so that they'll be retained during further learning; hyperplanes that are not relevant are

[12] The word "performance" in this dissertation usually refers to the performance of a network on data that was not used during training. We usually estimate the true performance via cross-validation testing on a set of test data [Weiss and Kulikowski, 1991].

Several additional points are of interest in the single-hidden-layer network:

Source of information between layers: The "summary" factor (2) in Equation 1.3, is the only source of information passed from the hidden-to-output layer to the formula for Δw_{ij} for the input-to-hidden layer weights. δ_k is weighted by the magnitude of hidden-to-output weights, so the direction of a hidden unit's activation change will be mostly determined by large-magnitude weights in this layer.

There is no separating influence on hyperplanes: Note that there is no provision made in either Equation 1.1 or 1.3 for guaranteeing that two hyperplanes aren't in the same positions relative to training examples (i.e. hyperplane activations are on the same side of 0.5 for all training examples).

Δw_{ij} does not depend directly on w_{ij}: As was true for the hidden-to-output weight change formula, the value of w_{ij} does not have a direct influence on Δw_{ij}. This point has important implications that will be discussed below in Section 1.2.3.

1.2.3 High weight magnitudes retard learning

Recall from Section 1.2.1 that hyperplane positions are determined by the proportions of their defining weights. Thus, the magnitude of weights defining a hyperplane can be arbitrarily adjusted while retaining the same hyperplane position.

The DBT algorithm that will be presented in Chapter 3 is based on the idea of retaining hyperplane position by changing initial network weights to affect hyperplane movement speed. In this section, we demonstrate that hyperplanes that are given initially higher weight magnitudes move more slowly during subsequent training.

How weight magnitudes affect training

There are two reasons to believe that, based on Equation 1.3 alone, larger weight magnitudes will lead to slow hyperplane movement during learning. First, large weight magnitudes tend to lead to large input values I_j, which, after processing through the sigmoidal, lead to hidden unit activations y_j that are close to 0 (for large-magnitude negative inputs) or 1 (for large-magnitude positive inputs). As mentioned above, either value leads to smaller values of term (1). Secondly, this equation does not directly include a factor that scales Δw_{ij} relative to w_{ij}. Therefore, larger weight magnitudes will have smaller relative changes.

Although these input-to-hidden equation factors aren't the only ones affecting the rate of hyperplane movement (error from the output layer and input activations are also factors), we do expect that, all other things being equal, large weight magnitudes lead to slower hyperplane movement, which is desirable for correct transferred weights. [Pratt, 1993] described a series of experiments that tested this hypothesis. That paper presented foundational work for this dissertation that showed that hyperplane magnitudes are an important factor in determining their movement. Even hyperplanes that are helpful in separating the data at the start of learning won't necessarily retain their positions unless their magnitudes are set to an adequately high value.

Several points about this hyperplane movement process are of interest here:

Hyperplane movement is only in response to errors: If there was an additional training example in Figure 1.11 whose desired activation was 0 and whose position was, say, at $(0.0, 1.5)$, then y_k would be so near to 0 that its effect on hyperplane position would be very small. It is only the misclassified desired-1 data item described above that causes hyperplane movement.

There is no hyperplane movement along dimensions whose input is 0: A zero value in factor (3) of Equation 1.1 makes $\Delta w_{jk} = 0$.

Δw_{jk} does not depend directly on the magnitude of w_{jk}: This property has an effect on hyperplane movement speed, to be discussed in Section 1.2.3.

Modification of input-to-hidden layer weights

Now consider a network that has a layer of hidden units (e.g. Figure 1.4 (page 6)). In the equation below for changes to input-to-hidden weights, η is the same value as in Equation 1.1, j indexes hidden units, i indexes input units, k indexes output units, x_i is the activation of input unit i, y_j is the activation of hidden unit j, and w_{ij} is the the weight from input unit i to hidden unit j. Let δ_k be the value shown in Equation 1.1. Δw_{ij} is the modification to each weight calculated on each back-propagation pass. The weight modification equation for the input-to-hidden layer (again, from [Rumelhart *et al.*, 1987a]) is:

$$\Delta w_{ij} = \eta \underbrace{y_j(1 - y_j)}_{(1)} \underbrace{\sum_k \delta_k w_{jk}}_{(2)} \underbrace{x_i}_{(3)} \tag{1.3}$$

As can be seen by examining Equation 1.3 and the formula for δ_k in Equation 1.1, it becomes clear that factor (2), which previously reflected the magnitude and direction of the desired change in activation for output neuron j, as determined by d_j, has been replaced by a cumulated "voting" from the output layer for the direction and magnitude with which the hidden unit's activation should change. As indicated by the δ_k label in Equation 1.1, for each output unit, factors (1) and (2) (reflecting the magnitude change due to proximity to activation of .5 and due to distance from the desired output, respectively) are multiplied together. As shown in Equation 1.3, δ_k is multiplied by w_{jk}, along with the weight to the corresponding output unit. The sums of these three-factor values are then calculated and used to determine both the direction of the desired hidden unit activation change, as well as to partially determine its magnitude.

Similar to Equation 1.1, η is an external control on weight change magnitudes, factor (1) in Equation 1.3 makes the magnitude of Δw_{ij} larger for hidden unit activations close to .5, and factor (3) makes individual weight changes in proportion to the magnitude of the input unit's activation.

Aside from the difference in the source of (2), the geometrical intuition shown for Equation 1.1 holds here as well. As was demonstrated in Section 1.2.1, each hidden unit implements a different hyperplane decision surface in feature space. Their outputs serve as input to the next hyperplane decision surface in the hidden-to-output layer.

It's important to understand the forces acting on this hyperplane during back-propagation learning. Begin by disregarding the effect of η (which is externally controlled and whose effect is just a constant multiplier of the rate of hyperplane movement) and the factor marked (1) in Equation 1.1 (which, as mentioned before, just makes movement faster when activations are near to 0.5). This leaves the weight update equation:

$$\Delta w_{jk} = (d_k - y_k)y_j$$

In the example network, there are three weights modified using Equation 1.1: two from each of F1 and F2 (we'll call them $w1$ and $w2$) and one bias value. $d_j - y_j = 1.0-(0.31) = 0.69$, so Δw_{ij} will also be positive, causing all weights to rise. The position at which the hyperplane intercepts the F1 axis can be obtained by solving $f1w1+f2w2+bias = 0$ for f2=0, to obtain: $-$bias/(w1). The F2 intercept is: $-$bias/(w2). Raising the bias will move the F1 intercept position to the right and the F2 intercept upwards. Raising the weights counteracts this effect to some degree, but the magnitude of the change will be in general smaller, because of the y_j factor, which is 1.0 for the bias, but never as high as 1.0 for a weight update. In general, then, the hyperplane will move up and to the right when $d_j - y_j$ is positive. In this example, the hyperplane's rightward movement makes it move from being above to being below the training point.

Assuming weight updates according to this simplified rule, then, $\Delta w1 = 0.69 \times 0.9 = 0.621$, so the new $w1 = -2.0 + 0.621 = -1.38$. $\Delta w2 = 0.69 \times 0.5 = 0.35$, so the new $w2 = 1.0 + 0.35 = 1.35$. Δbias $= 0.69 \times 1.0 = 0.69$, so the new bias $= 0.5 + .69 = 1.19$. The $F1$ axis intercept is $-\text{bias}/(w1) = -1.19/-1.38 = 0.86$. The new $F2$ axis intercept is $-\text{bias}/(w2) = -1.19/1.35 = -0.88$. The new network is shown in Figure 1.12.

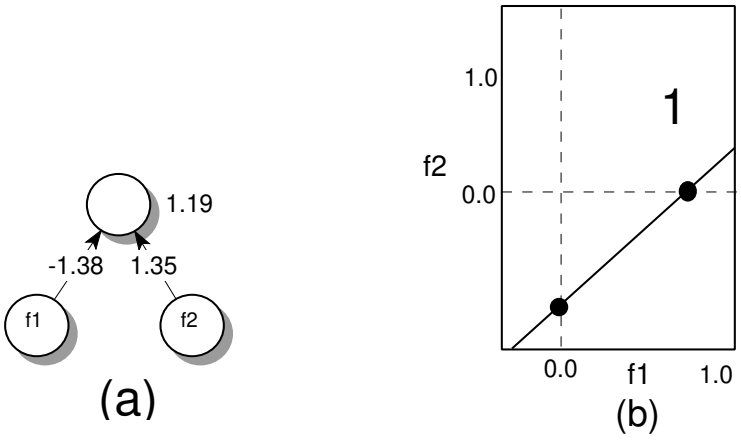


Figure 1.12: (a) The network in the previous figure after a weight update. (b) The associated decision region.

Although it is only easy to understand this movement geometrically in two dimensions, in higher dimensions, it is easy to understand that adding positive values to the coefficients (weights) on the left-hand side of a threshold equation like F1w1 + F2w2 + ... + Fnwn > 0 will increase the possibility that the inequality will be satisfied, and so the resulting activation will be 1.

for y_k that is near to the desired activation d_k will cause a smaller Δw_{jk} than when there is a large error.

The second effect of factor (2) is to determine not the magnitude but the sign of Δw_{jk}. An activation less than the desired value will yield $\Delta w_{jk} > 0$ (greater than desired gives $\Delta w_{jk} < 0$), thus forcing the activation to be higher (lower) on the next pass, so that it will be closer to the desired output.

Finally, the activation of the hidden unit, y_j, also contributes to the magnitude of the weight change. A larger activation of this hidden unit causes Δw_{jk} to be proportionally higher.

In practice, Equation 1.1 is usually modified to include a *momentum* term, as follows:

$$\Delta w_{jk}^{(n+1)} \leftarrow \Delta w_{jk}^{(n+1)} + \alpha \Delta w_{jk}^{(n)} \tag{1.2}$$

Here, n is a presentation number and α is a multiplier (also called the momentum) that determines the effect of the previous Δw_{jk} value on the current one. By encouraging movement in a constant direction through weight space, this term helps gradient descent in error space to escape from local minima. Use of a momentum term has been observed empirically to speed learning considerably, so it was used in the experiments described in this dissertation.

Geometrical intuition for single-layer weight movement

A simple network best illustrates the effect of this equation geometrically. Consider the zero-hidden-layer network and its associated decision region that are shown in Figure 1.11.

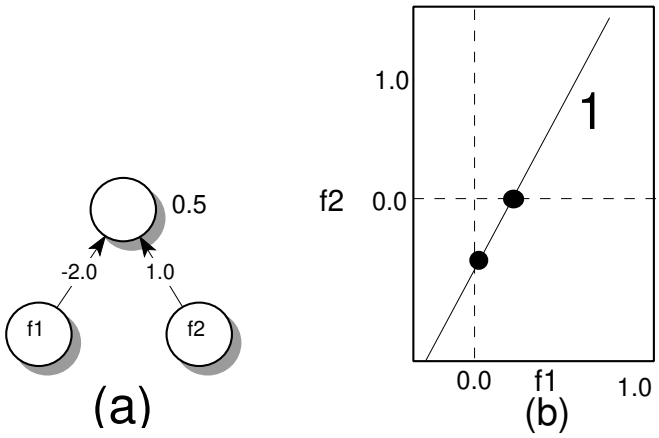


Figure 1.11: (a) A small network with no hidden units. (b) Its associated decision region.

For the training data pattern with desired output of 1 shown in Figure 1.11, the displayed hyperplane and the associated network do not provide the correct output. This training data item is at position $0.9, 0.5$, and its desired output is 1. However, the input to the output unit is $y_j = 0.9 \times -2.0 + 0.5 \times 1.0 + 0.5 = -.8$ on this training data item, resulting, after feeding this value through the sigmoidal, in an output activation of 0.31.

called feedforward.[11] The output layer activations generated by a training data item are calculated. The error is then calculated by comparing output activations to the desired output. For each pattern, the squared difference between the actual and desired output is calculated. This value is used in the back-propagation algorithm to calculate a modification to each weight in the network, such that upon the next presentation of that training data item, the error will be lessened. We now describe the specific equations used to calculate back-propagation weight updates.

Let η be a user-determined learning rate, let k index the output units, and let j index the hidden units. y_j is the activation of hidden unit j, y_k is the activation of output unit k, d_k is the desired output (from the training data) for output unit k, and w_{jk} is the weight from unit j to unit k. Δw_{jk} is the modification to each weight calculated on each back-propagation pass. For the hidden-to-output layer, the standard back-propagation formula (from [Rumelhart *et al.*, 1987a]) is:

$$\Delta w_{jk} = \eta \overbrace{\underbrace{y_k(1-y_k)}_{(1)}\underbrace{(d_k - y_k)}_{(2)}}^{\delta_k} \underbrace{y_j}_{(3)} \tag{1.1}$$

Let us discuss each part of this equation in turn.

First, the learning rate η determines the speed of learning. A larger value for η will make weights change, and hence learning progress, faster. The danger with too large values of η is that the learning process may diverge or oscillate. In practice, too-large η has been seen to correspond to an increase in the frequency of runs that do not converge to a global minimum (see, for example, [Tesauro and Janssens, 1988] and [Pratt *et al.*, 1990]).

The factor labeled (1) comes from taking the derivative of the sigmoidal activation function in the derivation of the back-propagation weight update rule. It is largest for values of y_k near 0.5, smaller for values close to 0 or 1. This function is illustrated in Figure 1.10. Its effect is to force faster weight changes when the output activation is

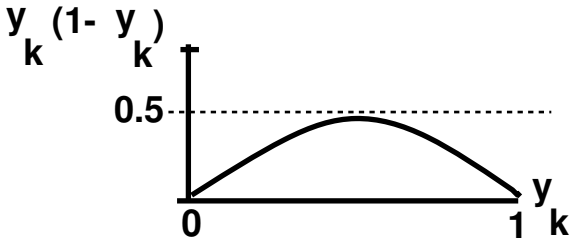


Figure 1.10: The relationship between y_k and $y_k(1-y_k)$

near 0.5.

The factor labeled (2) has two effects. First, along with the other factors in this equation, it contributes to determining the magnitude of Δw_{jk}. An output activation

[11] We will discuss presentation of training data items in the same repeated sequence, although they are randomly permuted in some implementations.

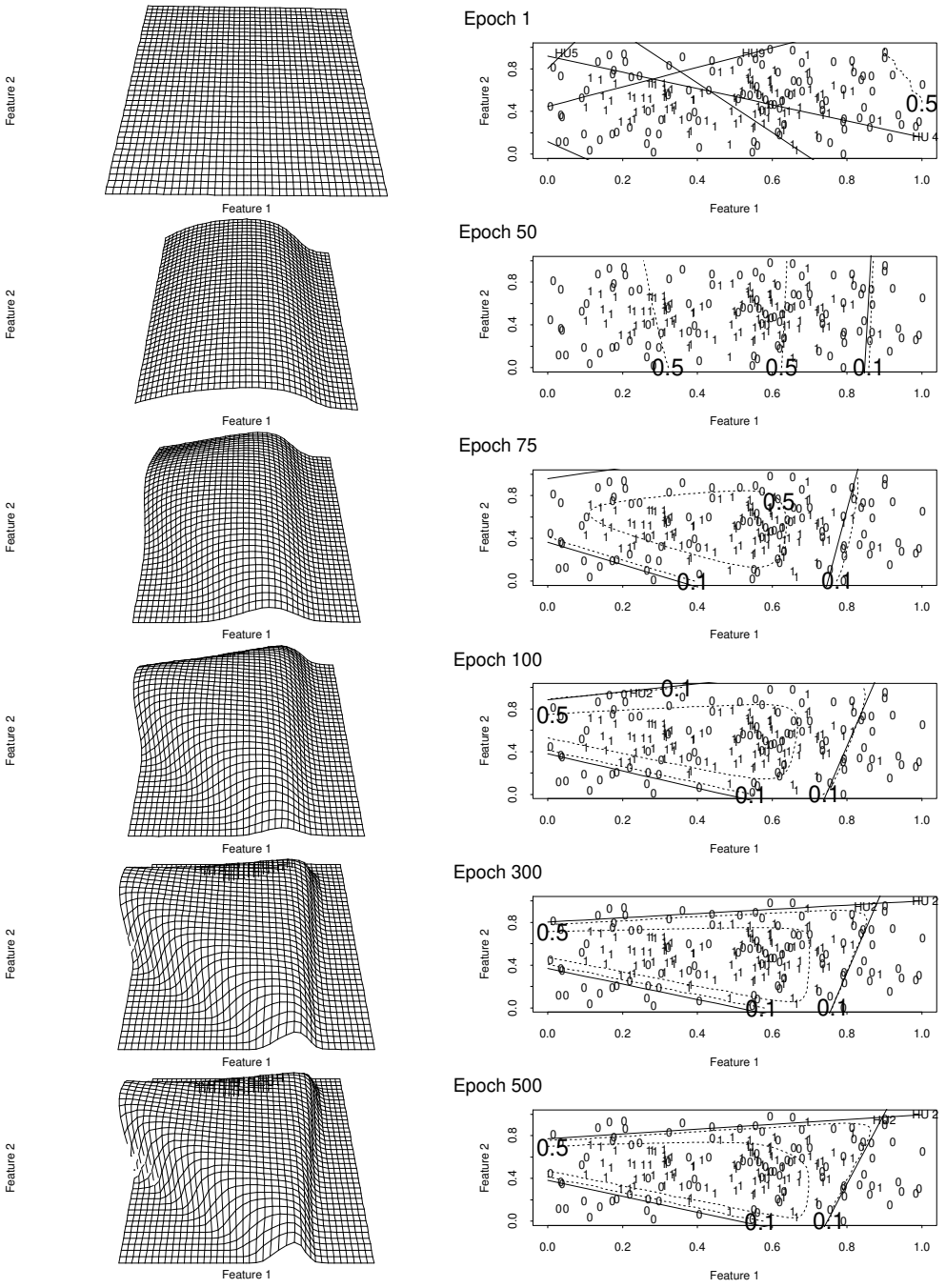


Figure 1.9: Output activation contours during the course of learning on the Gaussian distribution problem.

See text for details.

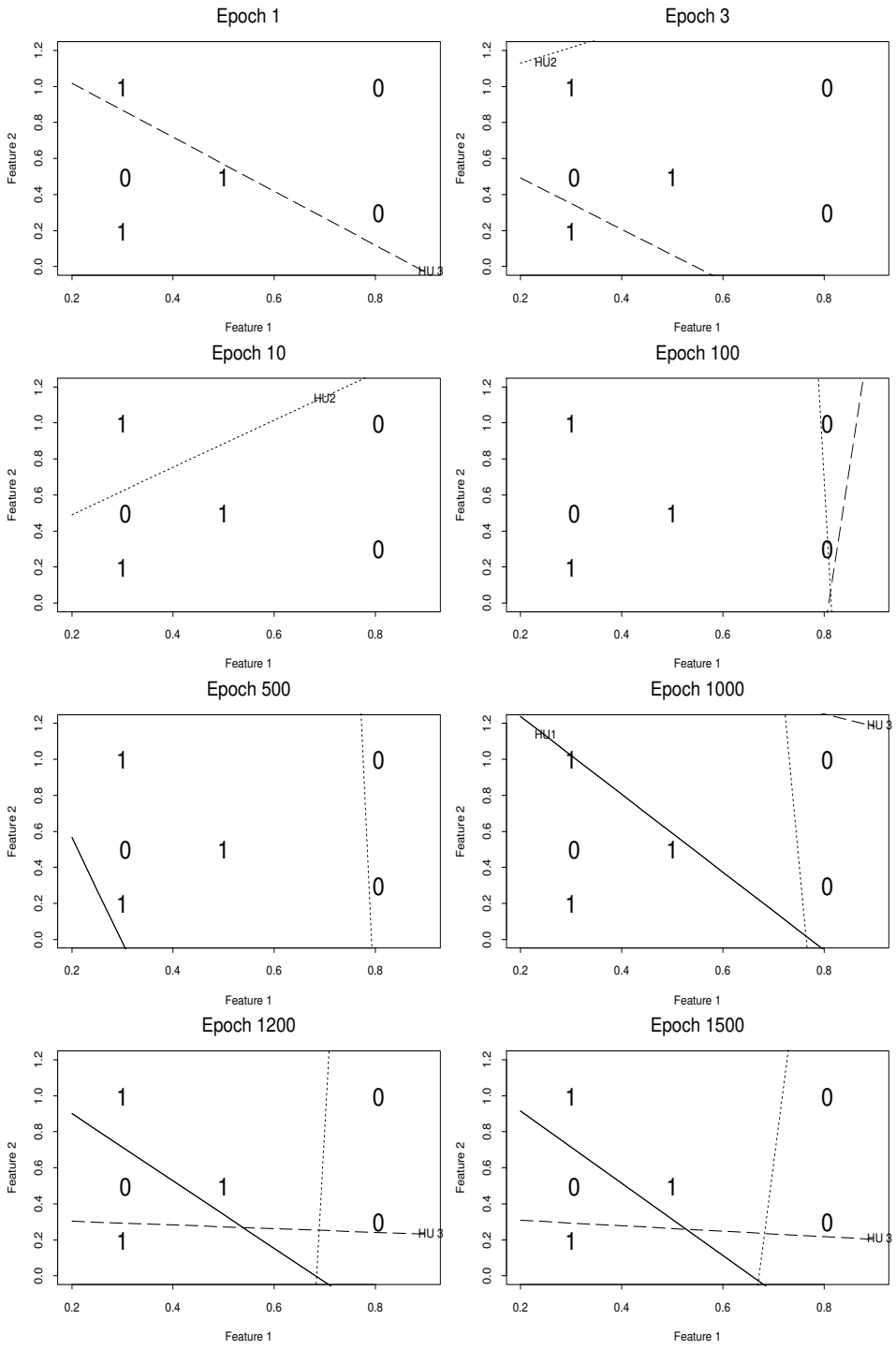


Figure 1.8: Snapshots of hidden unit hyperplane positions during learning

Shown are the three hyperplanes for a network on a small problem, at various epochs during learning. Hidden unit 1's hyperplane is a solid line; HU2's is a dotted line, and HU3's hyperplane is shown as a dashed line. Note that hyperplanes are initially randomly placed, and that they gradually move into positions that separate the data.

([Hinton *et al.*, 1984]), and some sophisticated graphical interface packages (e.g. [Wejchert and Tesauro, 1990]), this method doesn't simply illustrate the value of weights but also provides a means for displaying the relationship of the weights to the training examples that they are meant to classify.

Hyperplane dynamics during learning

Back-propagation learning is an iterative process that begins with random network weights and gradually refines them until training is complete (Section 1.1.2). Under the geometrical interpretation presented here, this process corresponds to random initial placement of hyperplane decision regions, with gradual movement into positions that separate the training data.

Figure 1.8 shows a number of snapshots of this learning process for a slightly more complicated problem than the one shown in Figure 1.6. Note how initial random IH positions eventually move into positions that separate the data.

Figure 1.9 shows the same problem as Figure 1.7, but for a network during the course of learning starting with initial random weights. As can be seen, the network initially outputs activations near 0.5 for all points in feature space[10], and gradually learns to differentiate positions that should receive output activation of 1 from those that should output 0.

Plots such as these can help to gain insights into neural network dynamics. These insights can inform the construction of transfer methods. For example, our formulation of transfer is based on the idea that hyperplanes placed in approximately correct initial positions can be used to speed up learning. This point was explored in more depth in [Pratt, 1993].

1.2.2 Back-propagation weight update equations and their effect on learning

In this section, we consider the relationship between the back-propagation weight update equations and the dynamics of hyperplane movement during learning. We consider networks with a single layer of hidden units; these networks are *minimally structured* ([Landauer *et al.*, 1987]), meaning that each layer is fully connected to adjacent layers, and there are no links that skip layers. Figure 1.4 (page 6) showed an example of such a network.

Modification of hidden-to-output layer weights

Back-propagation learning consists of a number of training *epochs* (Section 1.1.2). During each epoch, each training data pattern is presented to the network in a process

[10]The fact that the initial network outputs activations near 0.5 for all feature space points is a consequence of the form of the activation equation and the fact that the network has small weights throughout. Small weights lead to unit input values close to 0, which lead to activations near 0.5.

properly separate the data. This is because the hidden-to-output layer must implement a linear separation on its input domain, which may not be possible.

The left-hand image in Figure 1.7 plots a trained network's output activation against its two input features, for another problem with more training data. On the right is shown the equivalent contour plot (dotted lines) and the hyperplanes used to separate this problem. From this image it can be seen that the actual contour of the entire network's decision surface, including its HO weights, roughly follows a smoothed conjunction of IH hyperplane positions. This gives evidence that hyperplane positions provide a good indication of the network's overall behavior. Note, however, that more complicated problems over more than two inputs may or may not have such a clear correspondence between IH hyperplanes and overall decision surfaces.

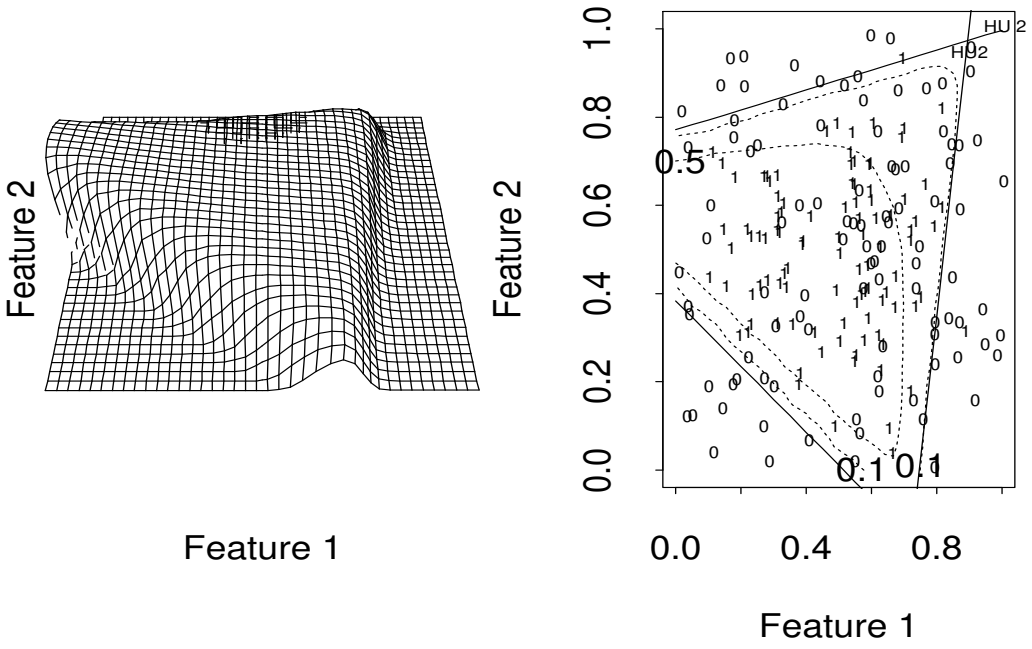


Figure 1.7: Output activation contour and corresponding hidden units for a network trained on Gaussian distributed data.

Class 1 values are chosen according to a Gaussian distribution, clustered around the center of feature space. Class 0 values are also distributed using a Gaussian distribution, with the same mean as the 1 targets, but a higher variance. On the left is a 3D plot, where height corresponds to the entire network's output value for various points of feature space. On the right is shown the IH hyperplanes in the network, along with the 0.5- and 0.1-level contours of the 3D plot. As can be seen, these contours follow the general shape formed by IH hyperplanes, indicating that IH hyperplanes provide an approximate representation of the network's overall decision surface.

Many researchers (cf. [Widrow and Winter, 1988] [Hanson and Burr, 1987], [Fahlman, 1990], [Atlas *et al.*, 1990b]) have used static versions of the graphical display technique described in this section. Unlike several other techniques, such as Hinton Diagrams,

Label	Input	Target	HU1 activation	HU2 activation
p0	0.1, 0.1	1	$< 0.5 \approx 0$	$< 0.5 \approx 0$
p1	0.3, 0.8	0	$< 0.5 \approx 0$	$> 0.5 \approx 1$
p2	0.4, 0.4	0	$< 0.5 \approx 0$	$> 0.5 \approx 1$
p3	0.8, 0.5	1	$> 0.5 \approx 1$	$> 0.5 \approx 1$
p4	0.8, 0.2	1	$> 0.5 \approx 1$	$> 0.5 \approx 1$

Table 1.2: Training data and HU activations for the sample problem
The hyperplane formed by HU1 separates the space of inputs for which that hidden unit will have activation > 0.5 from those for which it will have activation < 0.5. As shown in the rightmost two columns of this table, HU1 will have high activation on training items p3 and p4 but not on p0, p1, or p2.

Note that the position of a hyperplane is not determined by the absolute magnitudes of the weights leading to the associated hidden unit, but by their ratio. Thus the same hyperplanes shown in Figure 1.6 could have been generated by a network other than the one shown, by multiplying a hidden unit's incoming weights (along with its bias) by a constant value.

Conditions for successful classification

For many classifier induction tasks, a necessary condition for a properly trained classifier is that, as in Figure 1.6, hidden unit hyperplanes separate training data of different classes such that no region created by hyperplanes has training data targets of different classes.[9] In order to understand this necessity, note that hidden unit activations tend towards 0 or 1, as described in the previous section. Separation is a necessary condition because input activations that cause the same hidden unit activation vector will, by definition of feedforward, cause identical output unit activation vectors. Therefore, for two different input vectors with different desired outputs, hidden units must have different activation vectors if the network is to produce a proper classification, where the two vectors produce different output classes. In our simplified model, these two vectors must be on different sides of at least one hyperplane, so that it activates > 0.5 for one and < 0.5 for the other. This differing HU vector can then lead to a differing output vector. For example, hidden unit activations for the network of Figure 1.6 were shown in Table 1.2.

Separating training data of different target classes by hyperplanes is not a sufficient condition for learning because it is also necessary that the hidden-to-output weights, which are not represented by these diagrams, have the proper values. It also may be true that for a given set of training data and a given separating positioning of the input-to-hidden hyperplanes, no hidden-to-output weight vector exists that will

[9] See [Sontag, 1989] for exceptions. The consequence of this simplification is that it may appear that more hidden units are necessary to solve a problem than are strictly required.

Figure 1.5 can be interpreted as meaning that, when testing data are input to a classifier that's been trained on the training data shown, it should be classified into the target category of the nearest training data item shown (according to an as of yet unspecified distance criterion). For example, a testing data item whose input features are 0.2, 0.0 should probably be mapped to class 1.

Geometrical interpretation of hidden units in a classifier

Consider a single-hidden-layer back-propagation network that is fully connected in the input-to-hidden (IH) and hidden-to-output (HO) layers, and trained for a classification task. Let the input to unit j be $I_j = \sum_i y_i w_{ij}$, where y_i is the activation of incoming unit i, and w_{ij} is the weight between units i and j. Let the activation function be the sigmoidal: $y_i = \frac{1}{1+e^{(-I_j)}}$ where y_i is the activation of neuron i, and I_i is its input.

For large positive I_j, the sigmoid function is close to 1; it is near 0 for large negative I_j. As I_j approaches 0, y_j approaches 0.5. Because of the shape of the sigmoid, many I_j values yield y_j close to 0 or 1, however, and so a simplified way of understanding y_j is to ask whether it is less than or greater than 0.5. $y_j = 0.5$ when $I_j = \sum_i w_{ij} y_i = 0$. We can plot the equation of this line for each hidden unit in feature space along with the training data, as shown in Figure 1.6. Table 1.2, which will be discussed below, shows HU activations in this network for the training data.

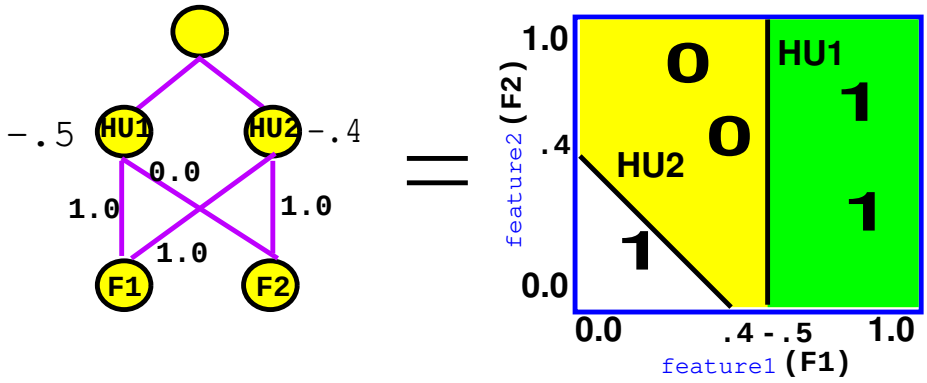


Figure 1.6: A small feedforward network and the corresponding hyperplane interpretation for its hidden units.

Values shown beside arcs are weights, beside nodes are biases.

In the network of Figure 1.6, hidden unit 1 (HU1) has activation = 0.5 when $1.0F1 + 0.0F2 - 0.5 = 0$. Similarly, HU2 has activation = 0.5 when $1.0F1 + 1.0F2 - 0.4 = 0$. In general, networks have more than 2 input features, so hidden units can be represented as $n-1$-dimensional *hyperplanes*[8] separating data in n-space.

[8] In neural networks, the word "hyperplane" has a more specific connotation than in the usual usage – it refers specifically to the decision surface coded by hidden or output units, even when the hyperplane is only a line in 2-dimensional space.

1.2 Understanding back-propagation learning

1.2.1 Graphical display of decision regions in a two-input network

So far, we've introduced the idea of transfer at a high level. Section 1.3 will give more details on the DBT transfer method. In order to understand the argument there as to why DBT is effective, it's necessary to first understand more details of back-propagation learning dynamics than are presented in Chapter 2. Specifically, it's important to understand factors that affect the movement of hyperplane decision surfaces in feature space. This section presents this interpretation.

Geometrical interpretation of training data

A training data set for a classifier induction task is represented as a list of <input, desired output> patterns.[7] If there are only two values in the input vector, then this data can be plotted in two-dimensional *feature space* by labelling points specified by an input value with the corresponding desired output value. For example, Figure 1.5 shows how the training data shown in Table 1.1 would be plotted.

Input	Target
0.1, 0.1	1
0.3, 0.8	0
0.4, 0.4	0
0.8, 0.5	1
0.8, 0.2	1

Table 1.1: Training data for a small example problem

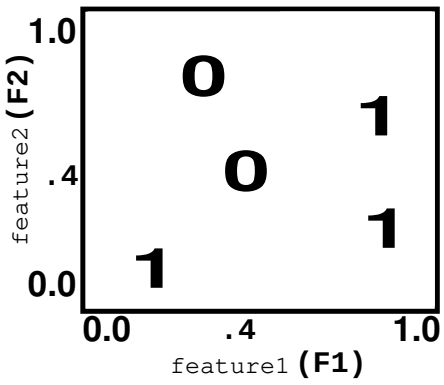


Figure 1.5: A set of training data plotted in feature space

[7] A commonly used term for the desired output that's specified in the training data is the *target*. We've used the phrase "desired output" here instead in order to avoid any confusion with a transfer target task.

In order to avoid overfitting (Section 2.1), back-propagation networks are usually constrained by using a small number of hidden units. More sophisticated approaches to overfitting avoidance consist of modifying the back-propagation weight update equations (Section 1.2.2) so that weights near zero are preferred (cf. [Hanson and Pratt, 1989, Weigend *et al.*, 1991]).

Learning

Learning in a back-propagation network consists of iteratively modifying weights (and biases, which are treated as additional weights) to change the network's classification behavior. The goal of learning is for the activations on the output layer after feedforward to approximate those in the training data desired output for each training data pattern. Learning consists of first randomly initializing network weights, then presenting training data patterns to the network repeatedly. For each pattern, weights are updated to bring the network output closer to the specified desired output value. Each pass through all the training data patterns is called an *epoch*. The formulas for calculating how weights change during back-propagation learning will be presented in Section 1.2.2.

Measuring network performance

Network performance on the training data is often measured by a value called the *Total Sum of Squared error* (TSS). This value is calculated by using the network to classify each training pattern. Each output neuron's value (o_i) is compared to the training data desired output value (d_i), which represents what that neuron's output value should be if the network worked perfectly on that input data item. Then the difference $d_i - o_i$ is computed, squared, and summed for all output neurons. The TSS formula is thus:

$$\text{TSS} = \sum_{j=1}^{m} \sum_{i=1}^{n} (d_{ij} - o_{ij})^2$$

where n is the number of output neurons, m is the number of training patterns, d_{ij} is the desired output for pattern j and output neuron i, and o_{ij} is the corresponding actual output unit activation. Low TSS corresponds to good network performance.

In the networks studied in this dissertation, each output unit represents a different output class. For this representation, we can use another useful error measure, which characterizes the number of patterns for which the highest activating output unit corresponds to the correct classification. We call this measure the *Training* (or testing) *Set Score* (TRS). One way of characterizing speed of training is to plot percent TRS vs. epoch. High TRS values early in learning mean that the network has fit the training data quickly. TSS and TRS may be calculated based on the training set, or, alternatively, on a test set (which provides an estimate of generalization performance, as described in Section 2.1).

---

flow in only one direction.

1.1.2 Back Propagation

Although many algorithms can be constructed using neuron-like computational units, we focus here on the Back-Propagation[4] algorithm for classifier induction ([Rumelhart *et al.*, 1987a], [McClelland and Rumelhart, 1988, Chapter 5] (which includes simulator software)). We chose to study back-propagation because it's one of the most widely applied neural network algorithms. Therefore, results based on it should be of interest to a broader research community than if we'd focused on a less commonly used method.

Classification

Back-propagation uses the same data structure for both classifier induction and for classification. In a back-propagation network, computational units are arranged into a sequence of *layers* (see Figure 1.4).

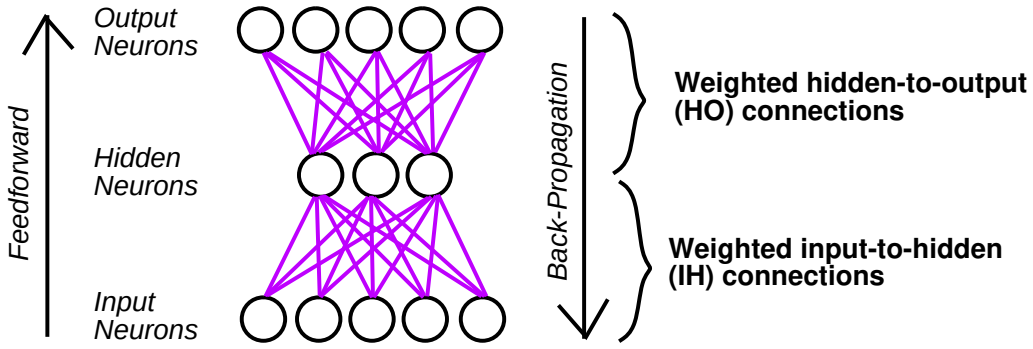


Figure 1.4: A back-propagation network.

During classification, a preprocessed data vector is first copied into input unit activations.[5] Based on these values, and on the values of weights between input and hidden units, hidden units then calculate their activations. Based on hidden unit activations and hidden-to-output weights, output units calculate their activations. The network's chosen classification is then read from the output unit activations, which usually each represent a class into which the input data can be mapped. In this case, the highest activating output unit represents the network's choice. This process is called *feedforward*, and can be done very quickly, especially in special-purpose parallel hardware [Intel, 1990]. Note that the network's reaction to a particular set of input values is determined by the weights and biases on links connecting nodes. [6]

---

[4] Also called the *Generalized Delta Rule*.

[5] Strictly speaking, the inputs are not really computational units, because they do not receive input or calculate activation based on previous unit values. Some projects have used sigmoidals in input units, however [Tenorio *et al.*, 1990] (though the same functionality could have been accomplished by preprocessing training data inputs through a sigmoid so that they're in the range $[0, 1]$).

[6] The architecture shown in Figure 1.4 is not the only possible way to arrange neurons in a back-propagation network. There may be zero or multiple hidden layers, and there may not be full connectivity between layers. There may also be connections that skip layers, such as a link directly from an input to an output neuron. The standard back-propagation algorithm requires only that information

- A unit's *activation* is a number associated with its state. For many neural network algorithms, activations must be $\in [0.0, 1.0]$.

- The *fan-in* of a unit refers to the connections that lead into it from other units in the network.

- Each connection from one unit to another has an associated *weight*, which represents the strength of the connection. Each unit also has an associated *bias*, which acts like an additional weight that is connected to a fan-in unit with a constant value of 1. In later sections, we will indicate the neuron's bias by writing it next to the neuron.[3]

- The *fan-in function* (also sometimes called the *input function*) calculates the input to a unit based on the activations of other units connected to it. A common fan-in function sums the product of activation and weight for each incoming unit. The fan-in to unit j might be defined as $I_j = \sum_i y_i w_{ij}$, where y_i is the activation of unit i, and w_{ij} is the weight from unit i to unit j. For example, for the unit in Figure 1.2, the fan-in is $.1 \times .2 + .3 \times .3 + .7 \times .4 + .05 \times .3 + 0.6 = 1.185$. The value I_j of the fan-in function is often called the unit's *input*.

- The *activation function* converts a unit's input to its activation. A common activation function is given by the *sigmoidal*:

$$y_i = \frac{1}{1 + e^{(-I_i)}}$$

  where y_i is the activation of unit i, and I_i is its input, calculated via the fan-in function. Note that this function maps its input into the interval $(0.0, 1.0)$, as shown in Figure 1.3. Using a sigmoidal, the unit in Figure 1.2 has an activation of $\frac{1}{1+e^{-1.185}} = .765$.

Chapters 1 and 2 of [Rumelhart *et al.*, 1987b] discuss the neuron-like computational unit and its use in neural networks in more detail.

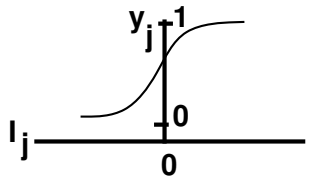


Figure 1.3: Sigmoidal activation function.

This function maps an arbitrary real-valued input into (0.0,1.0).

[3] A related term is the neuron's *threshold*, which is equal to $-bias$.

2. In contrast to many parallel computers, each neural processor does very limited computation: instead of being a full Turing machine, it only computes a threshold (or other simple) function over its input values.

The field of neural networks also goes goes by the names of Neural Information Processing Systems (NIPS), Parallel Distributed Processing systems (PDP), Connectionism, and Artificial Neural Networks (ANN). Neural network research is very active today, spawning a number of large conferences [Moody *et al.*, 1992, IEEE, 1991] and journals (*Neural Computation, Neural Networks, International Journal of Neural Systems*).

Neurophysiologist neural network researchers use computer models based on our understanding of biological neurons to explore theories of information processing in human and animal brains. Psychologists in this field attempt to explain the connection between higher cognitive function and neurophysiology. Philosophers explore the implications of neural networks for models of cognition. Mathematicians explore the novel computational characteristics obtained by computer systems that are based on neuron models. Engineers and computer scientists attempt to use these models to solve problems in a variety of domains.

The dissertation falls within this last camp. Although the idea of transfer is of interest in the psychological and neurobiological literature as well, our goal is not cognitive or biological modelling. Our focus is on using the idea of transfer to improve our ability to solve practical classification problems.

Neuron-like computational units

A computational unit based on neural functionality is shown in Figure 1.2. Some important concepts associated with this simplified neuron are defined below.

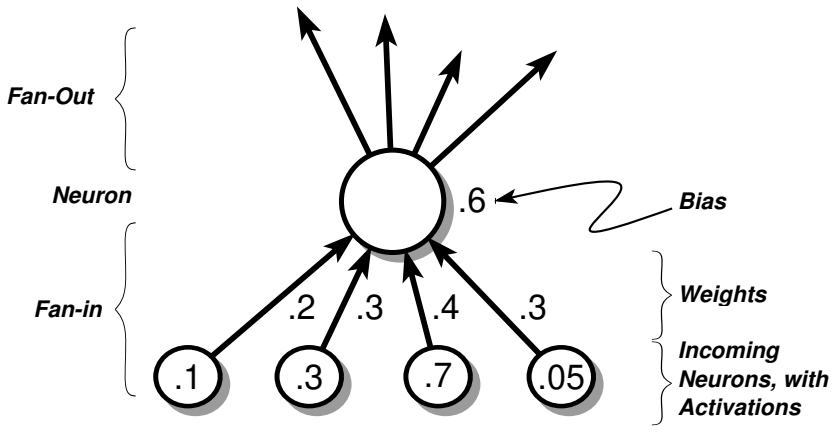


Figure 1.2: Neuron-like computational unit.
This unit omits many of the elements that have been found to exist in biological neurons, including details of the signal that it emits, and complex structure in the interior of the cell. Computational units are also called *nodes*, *threshold units*, and *neurons*.

Section 1.1 now gives further details on the back-propagation neural network learning algorithm. Then Section 1.2 describes the back-propagation weight update equations, and how they can be interpreted in terms of decision surfaces in feature space. Section 1.3 will then introduce the DBT algorithm, which forms the primary contribution of this dissertation (Chapter 3 will present DBT in full detail). Section 1.4 then describes several choices that were made during our formulation of transfer, such as the decision to use a neural network, to use the back-propagation neural network learning algorithm, and other aspects of the DBT algorithm. For each choice made, we'll explore the alternatives and discuss the consequences of the choice made on the impact of the results presented here. Finally, Section 1.5 presents the thesis claims of this dissertation.

1.1 Review of back-propagation neural networks

Neural networks have been used in many applied settings, including such diverse tasks as signal processing [Reilly *et al.*, 1992], catalog merchandising [Schwartz, 1992], medical screening [Rutenberg, 1992, Weber, 1990], and municipal power grid security [Atlas *et al.*, 1990d]. Back-propagation neural networks are an interesting method for classifier induction because they promise highly parallel (and hence very fast) real-time classification. They also provide a novel representational formalism, which leads to improved performance on some tasks, as demonstrated by several recent empirical studies [Fisher and McKusick, 1989, Mooney *et al.*, 1989, Atlas *et al.*, 1990c, Atlas *et al.*, 1990d, Cole *et al.*, 1990, Atlas *et al.*, 1990a, Dietterich *et al.*, 1990, Fahlman and Lebiere, 1990, Pratt and Norton, 1990, Weiss and Kulikowski, 1991, Shavlik *et al.*, 1991, Thrun *et al.*, 1991]. However, these same studies indicate that neural network learning is usually much slower than for competing machine learning and statistical methods. This point is reinforced by our own experience training a very large network [Pratt and Kamm, 1991], and by other authors' comments to this effect (cf. [Maren *et al.*, 1990]). Long training times for networks for practical problems can be prohibitively expensive. Therefore, our primary focus here is on exploring how transfer from related networks can speed up neural network learning time, while achieving comparable or improved performance. We'll elaborate further on this point in Section 1.4.3.

In the following two sections, we give more background information on neural networks in general, and on the back-propagation learning algorithm in particular.

1.1.1 Neural networks

Neural networks represent an approach to classifier induction that is inspired by our understanding of biological neurons in human and animal brains. Two primary features of neural networks distinguish them from other approaches:

1. Computation is massively parallel, with many processing units instead of a single CPU.

which more background will be given in Chapter 2).

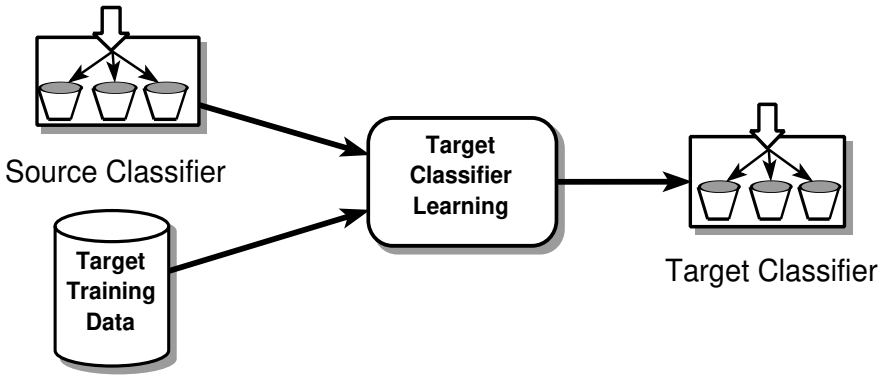


Figure 1.1: Transfer between classifiers.

Instead of using just a set of training data for target training, we also use an already trained classifier that has been trained on a related source task.

A definition of transfer is as follows:

> **Transfer:** The process of using a classifier for one task to aid in classifier learning for a second task.

This dissertation focuses on a particular formulation of the transfer problem. Although there are a number of methods for doing classification and for classifier learning, we focus on transfer between back-propagation neural networks [Rumelhart *et al.*, 1987a]. Also, although transfer might be used for a number of purposes (including improving the performance of the target classifier), we focus here only on using the transferred information to speed up the target classifier construction process. Section 1.4 will describe the particular formulation of the transfer problem studied here in full detail.

Transfer is an important problem to study because:

1. Transfer can help to alleviate an important problem with current back-propagation training schemes – the learning process is often very lengthy for complicated problems.

2. As source classifiers become more widely available in many domains, this approach will become more widely applicable.

The idea of transfer also has strong roots in psychology (as discussed in [Sharkey and Sharkey, 1992]), and is a standard paradigm in neurobiology, where synapses almost always come "pre-wired" (cf. [Durbin *et al.*, 1989], [Toates, 1980], [Lukowiak, 1980], [Lukowiak and Freedman, 1983]).

Chapter 2 reviews background material on classifier learning systems. The reader without a background in this area may wish to skip ahead to Chapter 2 at this point. Chapter 2 also discusses related work. This includes both the foundations for the transfer method presented here, as well as more recent work that builds upon transfer.

Chapter 1
Introduction

When people learn a new task, they build on their ability to solve related tasks. For example, it may require some time to learn to understand the speech of a new acquaintance, especially if their accent is unfamiliar. We don't have to start from scratch in this learning process, but can instead combine our ability to understand speech in general with information we gain by listening to the new speaker.

This dissertation explores how automated learning systems can build on their previous experience in this way. We focus on tasks that can be formulated as *classification problems*, where incoming data is mapped into one of a number of discrete categories. In the example above, part of the speech understanding task can be formulated as that of *phoneme classification*. In this problem a time sample of sound waves might be converted into a phoneme (discrete speech sound); phonemes would then be processed further to determine the words that were spoken. The automation of such classification methods (to provide, for example, a speech transcription device) is an active area of research. The algorithm that does the classification task is called a *classifier*.

Automated construction of classifier algorithms is also widely studied. This process often uses only a set of examples of input and output to the classification process to build a classifier. For instance, a set of pairs of <speech sound, phoneme spoken> examples might be gathered. An automated classifier construction algorithm would use this list to produce a classifier that was able to map an arbitrary speech sound into the correct phoneme. Automated classifier construction using a set of examples is often called classifier *learning*.[1]

In terms of this formulation, then, this dissertation studies how a set of examples of a new problem (e.g. the accented speech) can be combined with a classifier that has been previously constructed to solve a related task (e.g. more general speech recognition). By supplementing the set of examples with information from a previously learned classifier, learning of the new task can be more efficient than if it had to start from scratch.

We call this process the *transfer* of information from one classifier to another. The classifier from which information is extracted is called the *source*. This information will be used to facilitate learning on a *target* classification problem.[2] See Figure 1.1 (for

---

[1] Other terms for classifier learning are *classifier induction*, *inductive learning*, *learning from examples*, and *training*.

[2] Note that we use the word *target* in a different sense than in some inductive machine learning research, where it is used to refer to a target concept, as distinct from training data that represent that concept. In this thesis, we'll use the word "target" as explained here for the transfer framework; we'll say "concept to be learned" for the other meaning. This usage is also consistent with that in the machine learning subfield of analogical reasoning.

5.14. Second problem that can occur with using just IM for DBT prediction: this may produce a pessimistic measure. 117
5.15. Third problem that can occur with using just IM for DBT prediction. If the hyperplane moves out of position, then this may provide an optimistic measure. 117
5.16. Fourth problem that can occur with using just IM for DBT prediction. All four hyperplanes have the same IM value, so both situations have the same average IM. DBT can, however, adjust magnitudes on the problem on the right so that it gives faster learning than in the problem on the left. So using just the average IM is misleading. 118
5.17. Cost/Accuracy tradeoff in prediction methods for DBT. 119
5.18. Problem illustrating the need for DBT. 120
5.19. First problem that can occur when estimating the source classifier: this situation will produce a pessimistic prediction of DBT performance. . . 120
5.20. Second problem that can occur when estimating the source classifier: this situation will produce an optimistic prediction of DBT performance. . 121
5.21. Third problem that can occur with estimating the source classifier: since a different source classifier is used for measurement than for transfer, the estimation may be misleadingly optimistic or pessimistic. 121
6.1. Illustration of source network training. 128
6.2. Illustration of source network training when training data is impoverished . . 128
6.3. How transfer from a source to a target function can be helpful. 129
6.4. Problems that come from mis-balancing source and target information when the target data is impoverished. 129

4.13. Summary of DBT and random initialization time to 98% criterion 88
5.1. Venn diagrams illustrating why obtaining good source task generalization should help with target task generalization. 94
5.2. Average IM vs. speed-up factor observed in the seven benchmark tasks, for 3 different stopping criteria, along with their average. 101
5.3. Average IM vs. speed-up factor observed in the seven benchmark tasks, for 3 different stopping criteria. This shows all 100 runs for every task. . . 103
5.4. Average IM vs. log of speed-up factor observed in the seven benchmark tasks, for 3 different stopping criteria. 104
5.5. (a) Three different source tasks that were used to test whether lowering the source network IM led to slower training times. (b) Training curves showing how the average IM value related to learning speed. 106
5.6. (a) Target training data and source hyperplanes for second illustration of the relationship between IM and learning speed. (b) Training set curves showing that, as the IM decreases, so does the learning speed. . . . 107
5.7. Yet another problem on which to measure the relationship between IM and learning speed. 109
5.8. Training set curves for the problem in the previous figure showing that lower IM is associated with lower speed of convergence. 110
5.9. A problem which, similar to the previous one, has different feature scaling and hyperplanes that are not axis-parallel. 111
5.10. Training set curves for the problem in the previous figure showing that, as the IM reduces, so does the learning speed. 112
5.11. Generalization curves for Onemale networks initialized with varying-IM hyperplanes. 114
5.12. Learning time vs average IM for individual Onemale runs. 115
5.13. First problem that can occur with using just IM to predict the effectiveness of DBT. This might give a pessimistic prediction of DBT performance. 116

3.2. Snapshots of hidden unit hyperplane positions, showing how an excessively high magnitude hyperplane (HU1, shown as a solid line) from literal transfer can slow down learning. 49
3.3. Hyperplanes and their IM. 50
3.4. Information flow through a channel. 51
3.5. Information flow through a hyperplane decision surface. 52
3.6. The discriminability-based transfer (DBT) algorithm. 55
3.7. Flow of control in DBT. 56
3.8. Snapshots of hidden unit hyperplane positions, showing how DBT can speed up learning. 57
4.1. Peterson-Barney vowel data for only female speakers. 63
4.2. Peterson-Barney vowel data for only male speakers. 64
4.3. Illustration of how a fixed criterion may not capture learning speed. 72
4.4. Mean and normalized mean generalization curves for DBT/random and literal/random comparisons on the PB and PB123 problems. 75
4.5. Mean and normalized mean generalization curves for DBT/random and literal/random comparisons on the PB-onemale and DNA problems. . . . 76
4.6. Mean and normalized mean generalization curves for DBT/random and literal/random comparisons on Heart-VAS and Heart-HS problems. . . . 77
4.7. Mean and normalized mean generalization curves for DBT/random and literal/random comparisons on the Chess problem. 78
4.8. Illustration of length of periods of significant superiority between a speed-up method and a baseline. 81
4.9. Illustration that if we know when to stop, length of significant period is less important. 82
4.10. Summary of amount of time in period of DBT statistically superior performance over random initialization. 84
4.11. Summary of DBT and random initialization time to 66% criterion 86
4.12. Summary of DBT and random initialization time to 95% criterion 87

List of Figures

1.1. Transfer between classifiers. 2

1.2. Neuron-like computational unit. 4

1.3. Sigmoidal activation function. 5

1.4. A back-propagation network. 6

1.5. A set of training data plotted in feature space 8

1.6. A small feedforward network and the corresponding hyperplane interpretation for its hidden units. 9

1.7. Output activation contour and corresponding hidden units for a network trained on Gaussian distributed data. 11

1.8. Snapshots of hidden unit hyperplane positions during learning 13

1.9. Output activation contours during the course of learning on the Gaussian distribution problem. 14

1.10. The relationship between y_k and $y_k(1 - y_k)$ 15

1.11. (a) A small network with no hidden units. (b) Its associated decision region. 16

1.12. (a) The network in the previous figure after a weight update. (b) The associated decision region. 17

1.13. Transfer between neural networks. 20

1.14. Good and bad hyperplanes, according to the IM metric 24

1.15. A typical arrangement of ID3 hyperplanes. 25

1.16. A typical arrangement of CART hyperplanes. 26

2.1. A general framework for classifier induction. 37

3.1. Problem illustrating the need for DBT. 48

List of Tables

1.1. Training data for a small example problem 8
1.2. Training data and HU activations for the sample problem 10
2.1. Summary of various studies addressing transfer-like problems in neural networks. 46
3.1. Correspondence between information theory and decision surface evaluation. 53
3.2. Correspondence between calculated quantities in information theory and their meanings when evaluating the IM of a decision surface. 53
3.3. Results of rescaling weights via DBT on the PB problem. 59
4.1. Summary of benchmark tasks used to evaluate DBT 62
4.2. Performance of source networks on target tasks. 74
4.3. Results of DBT comparisons to randomly initialized networks 79
4.4. Results of literal comparisons to randomly initialized networks. 80
5.1. Number of runs that reached criterion for randomly initialized and DBT transfer networks. 102
5.2. Number of epochs to reach criterion for networks with different IM levels . 113
5.3. Table showing costs and advantages of the various methods for predicting DBT's effectiveness that are reviewed here. 123

6.3.6. Transfer between networks with different features and output classes 131
6.4. Conclusion 132
References 133

Second artificially constructed task designed to test IM's ability to predict DBT learning speed 105
Artificially constructed tasks designed to test IM's ability to predict DBT learning speed where hyperplanes are not axis-parallel and data on different dimensions is scaled differently 108
Experiments with systematically altering the "PB-Onemale" problem 108
Summary 113
5.3.2. Counterexamples to using the average IM as a predictor of DBT effectiveness 116
Summary 119
5.3.3. Other methods for estimating source/task relatedness 119
Estimating DBT performance from only source training data . . 119
Estimating DBT performance from initial conditions 121
Using TSS after some epochs to estimate learning curve after many epochs 122
5.3.4. Summary 122
5.4. Conclusion 122

6. Conclusion 124
6.1. Review of support for thesis claims 124
6.2. Validity of empirical tests 125
6.3. Future directions 126
6.3.1. Performance improvement and the power vs. generality tradeoff 126
6.3.2. Speed up DBT running time 130
6.3.3. Hyperplane combinations 130
6.3.4. More complex transfer functions 131
6.3.5. Other paradigms for DBT 131

Assumption about shape of random curve 73
4.4. Results of comparison to using the source network on the target task . . 73
4.5. Curves showing experimental results on benchmark tasks 74
4.6. Quantifying learning curve results 74
4.6.1. Scenarios for evaluation 81
4.6.2. Comparison to common baseline of random initialization 83
4.7. Specific comments about how to read results tables 83
4.7.1. Asymptotic performance 83
4.7.2. Number of statistically superior performance epochs 83
4.7.3. Shape of normalized curve 85
4.8. Discussion 85
4.8.1. Support for the thesis argument 85
4.8.2. Insights gained from these results 90
DBT helps more when there is room for improvement 90
DBT does well when a large amount of information is brought to bear on a small task 90
Some source networks are better than others 91
4.9. Conclusion 91

5. Why and when does DBT work? 92
5.1. Introduction 92
5.2. Assumptions and intuitions behind DBT: Why does it work? 93
5.2.1. Assumptions of generality 99
5.3. When does DBT work? Quantifying task relatedness 99
5.3.1. Using the IM metric to assess source/target relatedness 100
Measuring benchmark task IM 100
First artificially constructed task designed to illustrate IM's ability to predict DBT learning speed 105

Translating from information theory to decision surface measurement . . . 52
3.2.2. Calculating IM for a decision surface . . . 54
3.3. The DBT algorithm . . . 54
3.3.1. DBT I/O . . . 54
3.3.2. Modifying weights to be proportional to IM . . . 55
3.3.3. HO layer weights . . . 58
3.4. DBT example . . . 58
3.5. Conclusion . . . 60

4. Results . . . 61
4.1. Tasks used to evaluate DBT . . . 61
4.1.1. Peterson-Barney vowel recognition (PB) . . . 62
4.1.2. 3-class Peterson-Barney subset (PB123) . . . 63
4.1.3. Female to single male transfer ("Onemale") . . . 63
4.1.4. DNA promoter problem (DNA) . . . 65
4.1.5. Heart disease diagnosis for Switzerland patients (Heart-VAS, Heart-HS) . . . 65
4.1.6. Chess endgame problem (Chess) . . . 66
4.2. Methodology used to evaluate benchmark tasks . . . 66
4.2.1. Cross validation strategy . . . 66
4.2.2. Learning rate and momentum . . . 67
4.2.3. Hidden unit count in target networks . . . 68
4.2.4. Target network epoch count . . . 69
4.2.5. Source network training . . . 69
4.2.6. Transfer parameters: cutoff, scaleup . . . 70
Example of determining S and C . . . 70
4.2.7. Testing source networks on target task . . . 71
4.3. Learning speed measurement for benchmark problems . . . 71

2. Background and Related work 36
2.1. Classifier induction 36
2.1.1. Machine learning research on supplementing classifier learning with additional information 38
2.1.2. Analogy 39
2.1.3. KBANN vs. Transfer 39
2.2. Previous approaches to transfer in neural networks 40
2.2.1. Literal transfer 40
2.2.2. Studies of source task interference via literal transfer 41
2.2.3. Retaining source task performance 41
2.2.4. Modular network construction 41
2.2.5. This study, in comparison to literal transfer and modular network construction 42
2.2.6. D. K. Naik *et al.*: Meta-Neural Network approach to learning by learning 42
2.2.7. A. Agarwal *et al.*: An On-line algorithm to Overcome Catastrophic Forgetting 43
2.2.8. S. Thrun and T. Mitchell: Integrating Inductive and Explanation-Based Learning 43
2.2.9. Noel and Amanda Sharkey: Adaptive generalisation and the transfer of knowledge 44
2.2.10. Summary 45
2.3. Conclusion 45

3. The DBT algorithm 47
3.1. What can go wrong with literal transfer 47
3.2. Evaluating classifier components 50
3.2.1. The IM metric: background 50
Basis for IM in information theory 51

Modification of input-to-hidden layer weights 18
1.2.3. High weight magnitudes retard learning 19
How weight magnitudes affect training 19
1.3. Overview of Discriminability Based Transfer (DBT) 20
1.4. Problem Formulation 21
1.4.1. Neural networks for classifier induction 21
Statistical methods 21
Nonparametric methods 22
Back-propagation, compared to statistical methods 22
Symbolic machine learning 23
Transferable representations 23
ID3 24
CART 25
Back-propagation, compared to decision trees 26
IM, instead of other information measures for hyperplanes 27
1.4.2. Back-Propagation, instead of other neural network algorithms . . 27
Improved versions of back-propagation 28
The Perceptron Learning Rule 28
Boltzmann Machine learning 29
1.4.3. Goal of transfer: improving learning speed 29
1.4.4. Transfer algorithm design decision: setting initial conditions . . . 31
1.4.5. Initial weights as entry point to Back-propagation 32
1.4.6. Retention of source performance 33
1.4.7. Relationship of source and target tasks 33
Quantifying relevance of source to target task 33
Relative complexity 34
1.4.8. Summary of problem formulation 34
1.5. Thesis of the dissertation 35
1.6. Reader's guide to the remainder of the thesis 35

Table of Contents

Acknowledgements ii

Dedication v

List of Tables xiii

List of Figures xiv

1. Introduction 1

1.1. Review of back-propagation neural networks 3

1.1.1. Neural networks 3

Neuron-like computational units 4

1.1.2. Back Propagation 6

Classification 6

Learning 7

Measuring network performance 7

1.2. Understanding back-propagation learning 8

1.2.1. Graphical display of decision regions in a two-input network . . . 8

Geometrical interpretation of training data 8

Geometrical interpretation of hidden units in a classifier 9

Conditions for successful classification 10

Hyperplane dynamics during learning 12

1.2.2. Back-propagation weight update equations and their effect on learning 12

Modification of hidden-to-output layer weights 12

Geometrical intuition for single-layer weight movement 16

Dedication

For John

I greatly appreciate the help of William Cohen, who reviewed a dissertation draft and helped to considerably strengthen parts of its presentation.

I am also grateful to the IEEE, which made it possible for me to attend the NIPS conferences from 1988-1991 and IJCNN91; to the DOE, which funded a trip to the Emergent Computation conference at Los Alamos in 1989; to AAAI, which gave me a student travel scholarship in 1991; and to the Rutgers Computer Science Department for supplemental funding throughout my graduate career.

Thanks to Alex Waibel for suggesting the idea of using transfer for speaker adaptation.

This thesis was completely rewritten during my first semester as a faculty member at the Colorado School of Mines. This was a painful experience, but it might have been unbearable if not for the unfailing support of my colleagues Aaron Gordon, Jean Bell, Manavendra Misra, and Robin Murphy. Department head Ardel Boes and CSM vice president Frank Schowengerdt also supported my decision to continue publishing, and thus delay the thesis even further, producing what we all hope is an eventual positive effect on my career, at the expense of some short-term peace of mind.

The heart disease data were collected by: Matthias Pfister, M.D., of University Hospital in Zurich, Switzerland; Andras Janosi, M.D., of Hungarian Institute of Cardiology, Budapest; and Robert Detrano, M.D., of the V.A. Medical Center, Long beach and Cleveland Clinic Foundation. I used software distributed with [McClelland and Rumelhart, 1988] for many simulations.

No dissertation would be possible without the support of friends. There are many I'd like to thank, but special appreciation is due to Valentine Rolfe, who has always made herself available for caring support. David Cohn has also been a long-distance source of encouragement, and my best graduate student comrade, even before I came to Rutgers. My parents, Annis and Henry Pratt, have given me special insight and motivation in pursuing a graduate career. My sister, Faith Pratt, has also been a good sounding board for my feelings about graduate school. Finally, I've dedicated this dissertation to my husband, John G. Smith, without whose friendship, love, and creative ideas this thesis would not have been possible.

Mark Jones also influenced my work substantially, especially in its early stages. During 1987 and 1988, his interest in exploring the connection between AI and neural networks led me to the research direction that resulted in this dissertation. He has also been constant in his encouragement and support of my work.

I am indebted to Bellcore for employment in the summers of 1989 and 1990, in which I learned about many of the applied issues facing neural networks today. During this time period, Candace Kamm was particularly helpful and supportive, collaborating on a problem that related strongly to my thesis work. Peter Clitherow and Kathy Cebulka were also helpful colleagues; together we published an applied neural network paper outside the scope of transfer [Pratt *et al.*, 1990].

With myself, Steve Norton co-chaired a 1990 NIPS workshop on the relationship between decision trees and neural networks, in which some of the ideas expressed in this dissertation were first laid out. Steve also suggested the idea of reporting significance curves during the course of learning. Steve Nowlan and Ray Watrous helped out with the Peterson-Barney data during January 1991. Dave Ackley commented on the dissertation proposal and gave important guidance into the nature of the systematic testing studies.

Tom Fawcett was very helpful in showing me the connection between my work and that in machine learning that addresses the combination of inductive and theory-based learning methods. Fellow Rutgers students Diane Zimmerman, Dave Loewenstern, Vince Sgro, Paul Hoeper, and Mike Mozer provided valuable and timely feedback on drafts of papers on which this dissertation is based. Matt Dismukes, Paul Hoeper, Mick Noordewier, and Vince Sgro worked on a hyperplane animation system that helped to provide many of the intuitions associated with transfer. Sowmya Ramachandran co-authored a paper on the IMBS procedure for topology determination. Conversations with John Keane explored how best to use IMBS. Dawn Cohen was very helpful with questions about decision trees and information theory. CSM student Asbjorn Christenson reviewed a late draft of Chapter 1, and was helpful in finding some errors in some equations. I'm also indebted to Devon Wiel of the University of Oregon for helping me track down neurobiology references that are related to transfer.

Acknowledgements

This work would not have been possible without the help and encouragement of my PhD thesis committee. Jack Mostow encouraged me to study neural networks in the first place, despite a lack of interest in this area at Rutgers when I started out. He also introduced me to the field of machine learning in particular and to research in general, and suggested many of the ideas that led to this work. Haym Hirsh's incremental version space merging dissertation inspired the idea of neural network transfer. He commented extensively on a proposal draft and helped with many details of the DBT algorithm. Mick Noordewier provided essential help and encouragement, and helped me obtain financial support during 1991 and 1992 (under DOE #DE-FG02-91ER61129, through subcontract #097P753 from the University of Wisconsin), as well as helping with the DNA and other biological problems. Steve Hanson employed me as a TA for a neural network course at Princeton in 1988, and has been a continued source of encouragement and support in my graduate student career. With Siemens, Inc., he provided financial support during 1990. Casimir Kulikowski supported me in the idea of neural networks at Rutgers, and for the neural network colloquium series that I ran in the fall of 1988. He asked tough questions near the end of the dissertation, which strengthened it considerably. Chris Tong chaired my qualifying exam committee and provided constant support and encouragement during my early graduate career.

Discussions with John G. Smith led to the original formulation of the problem addressed in this dissertation, particularly on the general transfer framework. In addition to giving insightful and detailed comments on both proposal and dissertation drafts, he has been an invaluable sounding board for ideas throughout my graduate career. In particular, the discussion of practical scenarios for evaluating transfer in Section 4.6.1 was based on a discussion with John towards the end of the thesis.

© 1993
Lorien Y. Pratt
ALL RIGHTS RESERVED

TRANSFERRING PREVIOUSLY LEARNED BACK-PROPAGATION NEURAL NETWORKS TO NEW LEARNING TASKS

BY LORIEN Y. PRATT

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Jack Mostow and Haym Hirsh (co-advisor)

and approved by

(Jack Mostow)

(Haym Hirsh)

(Casimir Kulikowski)

(Michiel Noordewier)

(Steve Hanson)

New Brunswick, New Jersey

May, 1993

Transferring Previously Learned Back-Propagation Neural Networks to New Learning Tasks

L. Y. Pratt

lpratt@mines.colorado.edu

ML-TR-37

June 1993

When people learn a new task, they often build on their ability to solve related problems. For example, a doctor moving to a new country can use prior experience to aid in diagnosing patients. A chess player can use experience with one set of end-games to aid in solving a different, but related, set.

However, although people are able to perform this sort of skill *transfer* between tasks, most neural network training methods in use today are unable to build on their prior experience. Instead, every new task is learned from scratch.

This dissertation explores how a back-propagation neural network learner can build on its previous experience. We present an algorithm, called *Discriminability-Based Transfer (DBT)*, that facilitates the transfer of information from the learned weights of one network to the initial weights of another. Through evaluation of DBT on several benchmark tasks we demonstrate that it can speed up learning on a new task. We also show that DBT is more robust than simpler methods for transfer.

Computer Science Department

Rutgers University

New Brunswick, NJ 08903