CsMTL MLP For WEKA: Neural Network Learning with Inductive Transfer

Liangliang Tu and **Benjamin Fowler** and **Daniel L. Silver**
Jodrey School of Computer Science
Acadia University
Wolfville, Nova Scotia, Canada B4P 2R6
{053498t, 074771f, danny.silver}@acadiau.ca

Abstract

We present context-sensitive Multiple Task Learning, or *cs*MTL as a method of inductive transfer embedded in the well known WEKA machine learning suite. *cs*MTL uses a single output neural network and additional contextual inputs for learning multiple tasks. Inductive transfer occurs from secondary tasks to the model for the primary task so as to improve its predictive performance. The WEKA multi-layer perceptron algorithm is modified to accept *cs*MTL encoded multiple tasks examples. Testing on three domains of tasks demonstrates that this WEKA-based version of *cs*MTL provides modest but beneficial performance increases. Our on-going objective is to increase the availability of transfer learning systems to students, researchers and practitioners.

Introduction

Inductive transfer learning involves the use of prior knowledge of one or more related secondary tasks while learning a primary task, so as to develop a more effective model (greater generalization accuracy) than one developed from the primary task examples only (Thrun 1997). One of the more common methods of inductive transfer is Multiple Task Learning (MTL) (Caruana 1997). *cs*MTL is a form of MTL that encodes data from multiple tasks using context inputs rather than multiple outputs (Silver, Poirier, and Currie 2008). This paper documents our testing of the well known WEKA machine learning suite (Witten and Frank 2005) modified to allow its multi-layer perceptron algorithm to accept *cs*MTL encoded data. Our objective is to increase the availability of inductive transfer systems to students, researchers and practitioners.

Multiple Task Learning

Multiple task learning (MTL) neural networks are one of the better documented methods of inductive transfer of task knowledge (Caruana 1997; Silver and Mercer 1997). An MTL network is a feed-forward multi-layer network with an output for each task that is to be learned. The standard back-propagation of error learning algorithm is used to train all tasks in parallel. Consequently, MTL training examples are composed of a set of input attributes and a target output for each task. Figure 1 shows a simple MTL network containing a hidden layer of nodes that are common to all tasks. The sharing of internal representation is the method by which inductive bias occurs within an MTL network (Baxter 1996). The more that tasks are related, the more they will share representation and create beneficial inductive transfer.

Formally, let X be a set on $\Re^n$ (the reals), Y the set of $\{0, 1\}$ and $error$ a function that measures the difference between the expected target output and the actual output of the network for an example. Then for standard single task learning (STL), the target concept is a function f that maps the set X to the set Y, $f : X \rightarrow Y$, with some probability distribution P over $X \times Y$. An example for STL is of the form $(\mathbf{x}, f(\mathbf{x}))$, where $\mathbf{x}$ is a vector containing the input values $x_1, x_2, \ldots, x_n$ and $f(\mathbf{x})$ is the target output. A training set S_{STL} consists of all available examples, $S_{STL} = \{(\mathbf{x}, f(\mathbf{x}))\}$. The objective of the STL algorithm is to find a hypothesis h within its hypothesis space H_{STL} that minimizes the objective function, $\sum_{x \in S_{STL}} error\,[f(\mathbf{x}), h(\mathbf{x})]$. The assumption is that $H_{STL} \subset \{f | f : X \rightarrow Y\}$ contains a sufficiently accurate h.

MTL can be defined as learning a set of target concepts $\mathbf{f} = \{f_1, f_2, \ldots f_k\}$ such that each $f_i : X \rightarrow Y$ with a probability distribution P_i over $X \times Y$. We assume that the environment delivers each f_i based on a probability distribution Q over all P_i. Q is meant to capture some regularity in the environment that constrains the number of tasks that the learning algorithm will encounter. Q therefore characterizes the domain of tasks to be learned. An example for MTL is of the form $(\mathbf{x}, \mathbf{f}(\mathbf{x}))$, where $\mathbf{x}$ is the same as defined for STL and $\mathbf{f}(\mathbf{x}) = \{f_i(\mathbf{x})\}$, a set of target outputs. A training set S_{MTL} consists of all available examples, $S_{MTL} = \{(\mathbf{x}, \mathbf{f}(\mathbf{x}))\}$. The objective of the MTL algorithm is to find a set of hypotheses $\mathbf{h} = \{h_1, h_2, \ldots, h_k\}$ within its hypothesis space H_{MTL} that minimizes the objective function $\sum_{x \in S_{MTL}} \sum_{i=1}^{k} error\,[f_i(\mathbf{x}), h_i(\mathbf{x})]$. The assumption is that H_{MTL} contains sufficiently accurate h_i for each f_i being learned. Typically $|H_{MTL}| > |H_{STL}|$ in order to represent the multiple hypotheses.

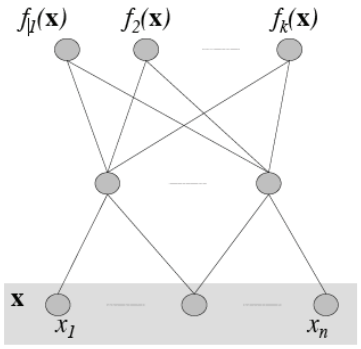

Figure 1: A multiple task learning (MTL) network with an output node for each task being learned in parallel.

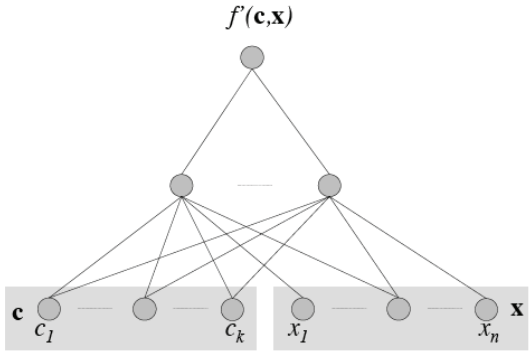

Figure 2: Proposed system: *cs*MTL

Limitations of MTL for Lifelong Learning

Previously, (Silver and Mercer 2002; Silver and Poirier 2004; O'Quinn, Silver, and Poirier 2005) have investigated the use of MTL networks as a basis for developing a machine lifelong learning (ML3) system and have found them to have several limitations related to the multiple outputs of the network[1]. First and foremost, MTL requires that training examples contain corresponding target values for each task; this is impractical for lifelong learning systems as examples of each task are acquired at different times and with unique combinations of input values. We have examined methods of generating corresponding target values but have found weaknesses related to the differences in the distribution of examples over the input space for various tasks.

With MTL, shared representation is limited to the hidden node layer and is not possible at the output nodes. The theory is that optimal inductive transfer occurs when related tasks share the same hidden nodes. This perspective does not consider the sharing of knowledge at the example level and in the context of unrelated tasks. Consider two concept tasks where half of the MTL training examples have identical target values. From a MTL task-level perspective, using most statistical and information theoretic measures, these sets of training examples would be considered unrelated and of little value to each other for inductive transfer.

There is also the practical problem of how a MTL based lifelong learning agent would know to associate an example with a particular task. Clearly, the learning environment should provide the contextual queues, however this suggests additional inputs and not outputs. A related problem is managing the redundant representation that can develop for the same task in an ML3 system based on MTL. A lifelong learning system should be capable of practising a task and improving its model with new examples over time. However, because there is no task context queue, an MTL based ML3 system requires a separate output for each new set of training examples. It is unclear how the build up of redundant task outputs could be managed over time.

[1]ML3 systems are capable of learning a set of tasks over a lifetime such that early learning facilitates the learning of tasks later in life.

In response to these problems, we have developed *context-sensitive* MTL, or *cs*MTL. *cs*MTL is based on standard MTL with two major differences; only one output is used for all tasks and additional inputs are used to indicate the example *context*, such as the task to which it is associated. The following section describes the *cs*MTL network.

*cs*MTL

Figure 2 presents the *cs*MTL network. It is a feed-forward network architecture of input, hidden and output nodes that uses the back-propagation of error training algorithm. The *cs*MTL network requires only one output node for learning multiple concept tasks (more outputs could be used for predicting a vector of values for each task). Similar to standard MTL neural networks, there is one or more layers of hidden nodes that act as feature detectors. The input layer can be divided into two parts: a set of *primary* input variables for the tasks and a set of inputs that provide the network with the *context* of each training example. The context inputs can simply be a set of task identifiers that associate each training example to a particular task. Alternatively, they can offer more specific environmental information (such as latitude and elevation) and in this way index over a continuous domain of tasks. Related work on context-sensitive machine learning can be found in (Turney 1996).

Formally, let C be a set on $\Re^n$ representing the context of the primary inputs from X as described for MTL. Let $\mathbf{c}$ be a particular instance of this set where $\mathbf{c}$ is a vector containing the values $c_1, c_2, \ldots, c_k$; where $c_i = 1$ indicates that the example is associated with function f_i. *cs*MTL can be defined as learning a target concept $f' : C \times X \rightarrow Y$; with a probability distribution P' on $C \times X \times Y$ where P' is constrained by the probability distributions P and Q discussed in the previous section for MTL. An example for *cs*MTL takes the form $(\mathbf{c}, \mathbf{x}, f'(\mathbf{c}, \mathbf{x}))$, where $f'(\mathbf{c}, \mathbf{x}) = f_i(\mathbf{x})$ when $c_i = 1$ and $f_i(\mathbf{x})$ is the target output for task f_i. A training set S_{csMTL} consists of all available examples for all tasks, $S_{csMTL} = \{(\mathbf{c}, \mathbf{x}, f'(\mathbf{c}, \mathbf{x}))\}$. The objective of the *cs*MTL algorithm is to find a hypothesis h' within its hypothesis space H_{csMTL} that minimizes the objective function, $\sum_{x \in S_{csMTL}} error\,[f'(\mathbf{c}, \mathbf{x}), h'(\mathbf{c}, \mathbf{x})]$. The assumption is that $H_{csMTL} \subset \{f | f : C \times X \rightarrow Y\}$ contains a suffi-

ciently accurate h'. Typically, $|H_{csMTL}| = |H_{MTL}|$ for the same set of tasks because the number of additional context inputs under *cs*MTL matches the number of additional task outputs under MTL.

With *cs*MTL, the entire representation of the network is used to develop hypotheses for all tasks, $f'(\mathbf{c}, \mathbf{x})$, following the examples drawn according to P'. The focus shifts from learning a subset of shared representation for multiple tasks to learning a completely shared representation for the same tasks. This presents a more continuous sense of domain knowledge and the objective becomes that of learning internal representations that are helpful to predicting the output of similar combinations of the primary and context input values. Therefore, the important concept of relatedness shifts from the task level to the example level. During learning, **c** selects an inductive bias over H_{csMTL} relative to the secondary tasks being learned in the network. Once f' is learned, if **x** is held constant, then **c** indexes over the hypothesis base H_{csMTL}. If **c** is a vector of real-valued inputs and from the environment, it provides a grounded sense of task relatedness. If **c** is a set of task identifiers, it differentiates between otherwise conflicting examples and selects internal representation used by related tasks.

*cs*MTL overcomes the limitations of standard MTL for construction of a machine lifelong learning system (Silver, Poirier, and Currie 2008). First, *cs*MTL eliminates redundant outputs for the same task making it easier to accumulate knowledge of a task through practice. Secondly, *cs*MTL examples are associated directly with a task via the context inputs, **c**. Finally, we conjecture that relatedness between tasks can be measured by the similarity of the context **c**, if **c** is environmentally grounded (such as the relative light level).

WEKA and the *cs*MTL MLP Algorithm

WEKA is an open source project of the University of Waikato (Witten and Frank 2005). It has been widely used in colleges and universities as well as by data mining practitioners on many real-world domains (McQueen et al. 1994). The WEKA multi-layer perceptron (MLP) was implemented by Malcolm Ware in 2000 (Ware 2000). Its use has been documented in a number of research publications (Klautau 2002). Our challenge is to implement a new version of the WEKA MLP such that it is capable of working with csMTL encoded data.

Two important corrections and five extensions to the existing WEKA MLP are implemented in a new Java class, MultilayerPerceptronCS (MLP_CS). The following subsections will present each of these corrections and extensions. This modified code can be found on the web at `ml3.acadiau.ca`.

Correcting WEKA MLP Early Stopping

The WEKA MLP intends to employ an early stopping approach to prevent over-fitting by using a separate validation set of data for the primary task. The network weights should be saved each time a lower validation error is found. As the model begins to over-fit to the training data, the error on the validation set will start to increase. At the end of the training, but prior to testing the network model, the weights of the neural network are returned to the point of the lowest validation error. More specifically, the WEKA MLP uses two parameters; validationSetSize and validationThreshold. The validationSetSize parameter determines the percentage of training set data use to create the validation set. The validationThreshold value is the number of iterations through the training data without a reduction in validation error before training should be stopped. The MLP documentation implies that the best set of weights will be restored after the validationThreshold is met.

Upon inspection we discovered that the original WEKA MLP software did not save the weights at the point of the lowest validation error and therefore could not restore to this model prior to testing. Instead, the weights used during testing were those of the last iteration completed. This finding has important implications for researchers and practitioners using WEKA and the MLP algorithm, as it would be very difficult to develop accurate neural network models under the current implementation. Furthermore, we discovered that the WEKA MLP software incorrectly changed the value of the lowest validation error and saved the model weights even when there had been a lower validation error encountered earlier in training.

Having identified these problems, we took steps to correct them. A modified version of the standard WEKA MLP was provided to the WEKA development team and integrated into new versions of WEKA. Now, the revised version of WEKA corrects the MLP code so as to properly use the validation set as explained above.

Extensions to the WEKA MLP

Five extensions are designed and implemented for the original WEKA MLP.

-T Parameter The first extension is for the "-T <filename>" parameter which allows the specification of a data file containing training examples for one or more secondary tasks. The file type can be any normally accepted by WEKA, which includes the comma separated values file format (csv) and attribute-relation file format (arff). The secondary task data must contain the same number of input and output values as the primary training data and in the same order. Both the primary task and the secondary task data files will contain all context and primary inputs, however the secondary task data file will contain only secondary task examples.

Duplication of Primary Task Training Examples The second extension concerns the primary task examples which normally are much fewer in number than that of each of the secondary tasks. The primary task training examples are duplicated as many times as needed to ensure an equivalent amount of training examples for each task. Without this duplication, the large number of examples for the secondary tasks would overpower the training of the neural network model such that the resulting model would perform poorly for the primary task.

-X Parameter The third extension is for the "-X <filename>" parameter, which allows the specification of a data file containing a separate set of validation examples for the primary task. This allows greater flexibility than the existing validation option, which selects a percentage of the training set to act as a validation set. The existing validation option is also modified to ensure that only primary examples are used for the validation set in cross-validation mode.

It is important to note that, although training examples for primary and secondary examples are used, only the accuracy of primary task is considered as the performance indicator. The model is developed to minimize the error on the validation data for the primary task and not across all tasks. A sufficiently large number of training examples for secondary examples are used so as to ensure the development of accurate models for these tasks without the need for early stopping via a validation set.

Cross-validation Statistics An additional feature added for cross-validation is the reporting of the test set statistics for each of the k models developed. This allows researchers to compute the standard deviation over the models from the repeated studies and therefore the confidence interval for producing a model of the mean accuracy.

Additional GUI Components The fifth extension displays additional validation error information in the neural network graphical user interface (GUI) while the system is training. If the GUI is chosen, the resulting interface shows not only the current iteration number and validation set error, but also the lowest validation set error and the iteration at which it occurred. This addition facilitates error and model monitoring as learning occurs.

Use of the New WEKA MLP_CS Algorithm

The WEKA MLP_CS algorithm has a number of modes of operation that include the creation of a validation set, choice of a specific test set, or alternatively cross-validation using all available training data. Cross-validation was used for all studies presented in this paper and below we explain the changes made for this mode of operation. Additional details can be found at `ml3.acadiau.ca`.

When testing inductive transfer methods, one normally uses an impoverished set of data for the primary task so as to demonstrate the value of knowledge transferred from the secondary tasks. Our testing simulates this by providing a small set of data for the primary task from which training, validation, and test instances are drawn. In comparison, an abundance of examples are provided for the secondary tasks. A 10-fold cross-validation is used to thoroughly test the learning algorithm's ability to develop accurate models. For each cross-validation run, 10% of the primary task data is used for testing. If the amount option for the validation set is specified as 30%, then 30% of the remaining primary task data is used for validation, and the rest is used for training. In order to ensure the very best transfer of knowledge, 100% of the secondary task data is used for training.

Experimentation

The following compares the predictive accuracy of learned models developed under STL using only primary task examples with models developed under *cs*MTL with inductive transfer from secondary tasks. All models are developed with our modified WEKA MLP_CS. The objective is to develop models for the primary task from a small set of training examples so as to observe the effect of inductive transfer from the secondary tasks.

Task Domains

Three domains are studied. The *Band domain*, described in (Silver and Mercer 2002), consists of seven synthetic tasks. Each task has a band of positive examples across a 2-dimensional input space. The tasks were synthesized so that the primary task T_0 would vary in its relatedness to the other tasks based on the band orientation.

The *Logic domain*, described in (McCracken 2003) consists of six synthetic tasks. Each positive example is defined by a logical combination of 4 of the 10 real-valued inputs of the form, $T_0 : (I_0 > 0.5 \vee I_1 > 0.5) \wedge (I_2 > 0.5 \vee I_3 > 0.5)$. The tasks of this domain are more or less related in that they share zero, one or two features such as $(I_0 > 0.5 \vee I_1 > 0.5)$ with the other tasks. The Band and Logic domains have been designed so that all tasks are non-linearly separable; each task requires the use of at least two hidden nodes of a neural network to form an accurate hypothesis.

The *Covertype domain* can be found at the UCI ML repository. The Covertype domain contains data from four wilderness areas in northern Colorado. Each example contains 10 input values representing information such as elevation and soil type, and one output value that classifies the cover type, that is, the species of tree that grows there. There are six types of cover in the data, including various types of spruces, firs, and pines. This is a large and noisy data set for which previous methods have had a difficult time.

Table 1 shows the relevant statistics for each task domain and for each cross-validation run: the number of tasks in the domain, the number of primary input attributes, the number of primary task training, validation, and test set examples, and the number of training examples for each secondary task. To clarify, the first row is for a cross-validation run using 20 primary task examples and 200 secondary task examples for the Logic domain. The 20 primary task examples are used to complete a 10-fold cross-validation with 13 training, 5 validation and 2 test examples in each fold.

Method

STL and *cs*MTL networks were configured using four-layers of nodes. The input layer receives the primary and context input values for each task. The first hidden layer contains 20 nodes with sigmoid transfer functions for the Logic domain and 30 nodes for both the Band and the CoverType domains. The second hidden layer contains only one sigmoid node. This second hidden layer is necessary because it provides shared internal representation between the first hidden layer and output layer. Also, the networks were configured to generate nominal outputs; two possible class values. Therefore,

Table 1: Statistics for the three domains of tasks used in the experiment.

Name	Tasks	Input Attributes	Primary Task			Secondary Task
			Train	Validation	Test	Training
Logic	6	10	13	5	2	200
		10	16	7	2	200
		10	19	8	3	200
		10	25	11	4	200
		10	32	13	5	200
		10	63	27	10	200
		10	126	54	20	200
Band	7	2	10	4	2	50
CoverType	6	10	28	12	4	300

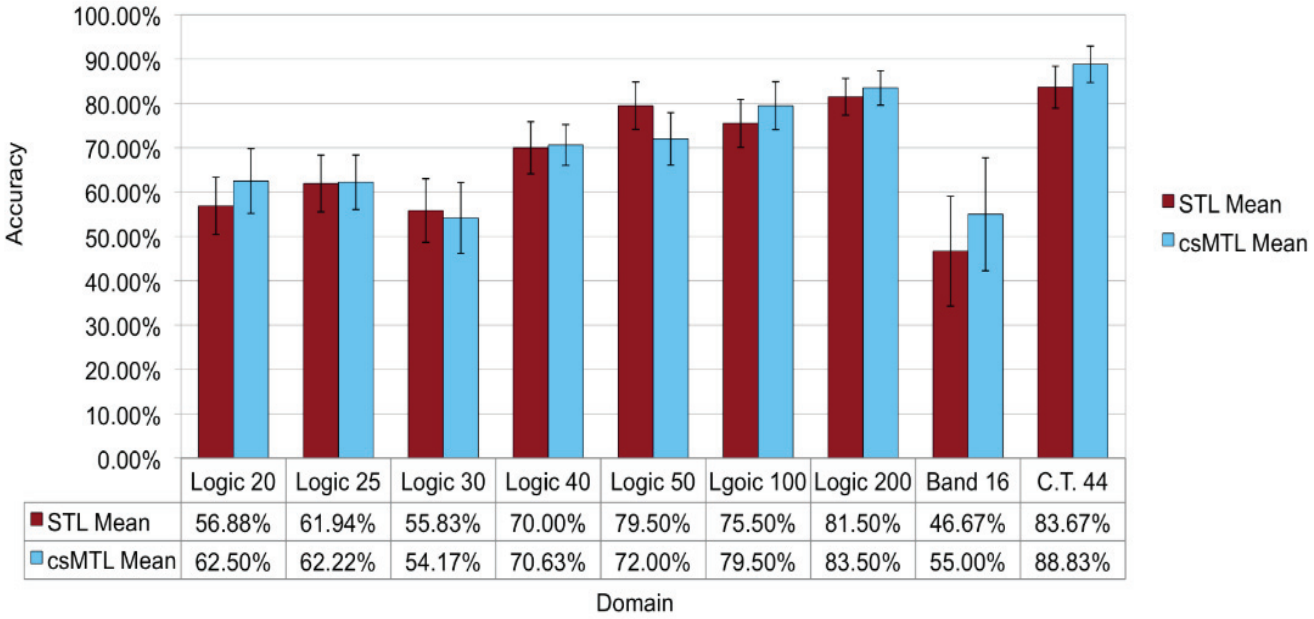

Figure 3: Comparison of STL and csMTL model performance across all domains.

the fourth output layer, contains two nodes, each representing a separate class. As an example, for the Logic domain, we use a 16-20-1-2 network.

For each network, the learning rate is 0.001 and the momentum term is 0.9. All tests are run for 50,000 iterations. The study compares the predictive accuracy over all examples using a 10-fold cross-validation approach.

It is important to note that, under *cs*MTL, the number of training examples for all tasks must be the same so as to ensure that each task has equal opportunity to update the network weights. This requires the primary task training examples to be duplicated until their number equals that of each secondary task.

Results

Figure 3 shows the comparative results from the cross-validation runs for the three domains. "Logic 20" indicates that the bar graphs above it present the mean results for the Logic domain when only 20 examples were used for the training, validation and test sets of data.

The respective p-values from a difference of means t-Test betwen the STL and *cs*MTL models are: Band domain, 0.716; Logic domain, 0.171; and CoverType, 0.036. The results on the last two domains indicate that there is value in using the *cs*MTL over STL. This having been said, these results are not as strong as that of prior studies using alternative MLP neural network implementations. The reasons for this require further investigation.

Conclusion and Future Work

This paper has described context-sensitive MTL and modifications to the WEKA machine learning suite that allows its MLP alogorithm to accept *cs*MTL encoded data. This modification allows WEKA users to experiment with inductive

transfer from related secondary tasks when faced with few training examples for a primary task. Our intention is to increase the availability of inductive transfer machine learning systems to students, researchers and practitioners.

The results of these initial experiments show that the WEKA *cs*MTL MLP algorithm is transferring knowledge from secondary tasks to the primary task, but not at the same level of performance as seen in prior studies using alternate neural network implementations. We are investigating the reason why the current implementation is not working as well as anticipated.

In the future we intend to use the WEKA *cs*MTL MLP to explore new theory in inductive transfer and apply this theory to a variety of classification and predictive (real-valued) tasks.

Acknowledgments

This research has been funded in part by the Government of Canada through NSERC.

References

Baxter, J. 1996. Learning model bias. In Touretzky, D. S.; Mozer, M. C.; and Hasselmo, M. E., eds., *Advances in Neural Information Processing Systems*, volume 8, 169–175. The MIT Press.

Caruana, R. A. 1997. Multitask learning. *Machine Learning* 28:41–75.

Klautau, A. 2002. Classification of peterson and barney's vowels using weka. Technical report, Universidade Federal do Par.

McCracken, P. 2003. *Selective Representational Transfer Using Stochastic Noise*. Wolfville, NS, Canada: Honurs Thesis, Jodrey School of Computer Science, Acadia University.

McQueen, R.; Neal, D.; DeWar, R.; Garner, S.; and Nevill-Manning, C. 1994. The weka machine learning workbench : Its application to a real world agricultural database. In *Proc Canadian Machine Learning Workshop*.

O'Quinn, R.; Silver, D. L.; and Poirier, R. 2005. Continued practice and consolidation of a learning task. In *Proceedings of the Meta-Learning Workshop, 22nd Int. Conference on Machine Learning (ICML 2005)*.

Silver, D. L., and Mercer, R. E. 1997. The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Learning to Learn* 213–233.

Silver, D. L., and Mercer, R. E. 2002. The task rehearsal method of life-long learning: Overcoming impoverished data. *Advances in Artificial Intelligence, 15th Conference of the Canadian Society for Computational Studies of Intelligence (AI'2002)* 90–101.

Silver, D. L., and Poirier, R. 2004. Sequential consolidation of learned task knowledge. *Lecture Notes in Artificial Intelligence, 17th Conference of the Canadian Society for Computational Studies of Intelligence (AI'2004)* 217–232.

Silver, D. L.; Poirier, R.; and Currie, D. 2008. Inductive tranfser with context-sensitive neural networks. *Machine Learning* 73(3):313–336.

Thrun, S. 1997. Lifelong learning algorithms. *Learning to Learn* 181–209.1 Kluwer Academic Publisher.

Turney, P. D. 1996. The identification of context-sensitive features: A formal definition of context for concept learning. In *13th International Conference on Machine Learning (ICML96), Workshop on Learning in Context-Sensitive Domains*, volume NRC 39222, 53–59.

Ware, M. 2000. *WEKA Documentation*. University of Waikoto.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco: Morgan Kaufmann Publishers, second edition.