Mach Learn (2008) 73: 313–336
DOI 10.1007/s10994-008-5088-0

Inductive transfer with context-sensitive neural networks

Daniel L. Silver · Ryan Poirier · Duane Currie

Received: 25 February 2007 / Revised: 8 September 2008 / Accepted: 17 September 2008 /
Published online: 21 October 2008
Springer Science+Business Media, LLC 2008

Abstract *Context-sensitive* Multiple Task Learning, or *cs*MTL, is presented as a method of inductive transfer which uses a single output neural network and additional contextual inputs for learning multiple tasks. Motivated by problems with the application of MTL networks to machine lifelong learning systems, *cs*MTL encoding of multiple task examples was developed and found to improve predictive performance. As evidence, the *cs*MTL method is tested on seven task domains and shown to produce hypotheses for primary tasks that are often better than standard MTL hypotheses when learning in the presence of related and unrelated tasks. We argue that the reason for this performance improvement is a reduction in the number of effective free parameters in the *cs*MTL network brought about by the shared output node and weight update constraints due to the context inputs. An examination of IDT and SVM models developed from *cs*MTL encoded data provides initial evidence that this improvement is not shared across all machine learning models.

Keywords Inductive transfer · Artificial neural networks · Context-sensitive learning · Context attributes · Task relatedness · Machine lifelong learning

1 Introduction

Multiple task learning (MTL) is well recognized as a method of inductive transfer that is able to develop superior performing hypotheses from impoverished training sets. Research on MTL has occurred using traditional machine learning methods (Bakker and Heskes 2003; Caruana 1997; Baxter 1997; Heskes 2000; Thrun and Pratt 1997; Ben-David and Schuller 2003), statistical regression methods (Greene 2002; Zellner 1962; Breiman and Friedman 1998), Bayesian methods involving constraints such as hyper priors (Allenby and Rossi 1999; Arora et al. 1998; Bakker and Heskes 2003), and most recently kernel methods such as support vector machines (SVMs; Jebara 2004; Allenby and Rossi 2005). All of these

Editor: Risto Miikkulainen

D.L. Silver (✉) · R. Poirier · D. Currie
Jodrey School of Computer Science, Acadia University, Wolfville, NS, Canada B4P 2R6
e-mail: danny.silver@acadiau.ca

approaches rely upon the development of multiple hypotheses under a constraint or regularization that characterizes a similarity or *relatedness* between the tasks.

Multiple task learning (MTL) neural networks are one of the better documented methods of inductive transfer of task knowledge (Caruana 1997; Silver and Mercer 1996). An MTL network has an output for each task and develops internal representations that are shared by all tasks. Inductive transfer occurs as a function of sharing these representations. We have investigated the use of MTL networks for the purpose of developing a *machine lifelong learning* system capable of retaining learned knowledge for use in future learning. A key finding has been problems introduced by multiple outputs of MTL systems, such as the build-up of redundant outputs from different sets of training examples for the same task (O'Quinn et al. 2005; Silver and Poirier 2005). Perhaps, more importantly, MTL transfer is dependent upon an elusive *measure of task relatedness* for selecting the more related tasks for optimal inductive transfer (Caruana 1997; Thrun 1996). Lacking a solution to these problems, we have investigated methods of MTL that do not require multiple outputs nor depend upon a method of measuring task relatedness.

This paper presents *context-sensitive* multiple task learning, or *cs*MTL, as a method of inductive transfer that uses a standard back-propagation single-output neural network and additional contextual inputs for learning multiple tasks. The goal of this paper is not to introduce a new learning algorithm, but rather to study the effects of the encoding method on existing algorithms, particularly, artificial neural networks. Our contribution is to demonstrate the strengths and weaknesses of this approach in contrast to existing methods.

Toward this end, we show that the number of free parameters of csMTL networks are lower than their MTL counterparts and that the shared output weights force an additional constraint over a *cs*MTL network's hypothesis space. Experiments on two synthetic and five real-world domains demonstrate that *cs*MTL networks develop models that often perform better than MTL networks. Most notably, we show empirically that *cs*MTL hypotheses for a primary task outperform MTL hypotheses when the examples from the secondary tasks are drawn from the same function as the primary task.

Potentially, the *cs*MTL approach allows multiple tasks to be learned by any single task learning (STL) method. In the later portion of the paper we investigate this possibility. Specifically, we show that Inductive Decision Trees and Support Vector Machines (with linear and radial basis kernels) do not benefit from the introduction of context attributes when learning multiple tasks. The resulting models are equivalent to STL models developed only from the primary task examples. This suggests that neural networks have an ability to capture regularities in the examples across the various tasks and associate these regularities with the context inputs in a way that, at least, these two other methods cannot.

2 Background and notation

Standard classification or concept learning can be formalized as follows. 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 single task learning (STL), the target concept is a function f that maps the set X to the set Y, $f : X \to Y$, 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, $S_{\mathrm{STL}} = \{(\mathbf{x}, f(\mathbf{x}))\}$. The objective of the STL algorithm is to find a hypothesis, h within its hypothesis space, H_{STL}, that minimizes the objective function, $\sum_{x \in S_{\mathrm{STL}}} error[f(\mathbf{x}), h(\mathbf{x})]$. The assumption is that

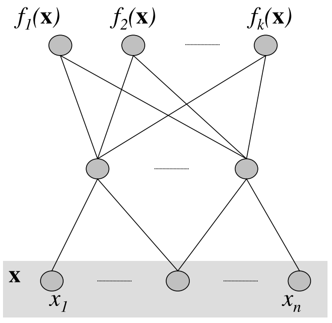


Fig. 1 A multiple task learning (MTL) network

$H_{\text{STL}} \subset \{f \mid f : X \to Y\}$ contains a sufficiently accurate h. Typically, a validation or tuning set of examples is used to prevent over-fitting to the training data and promote generalization.

An MTL network is a feed-forward multi-layer network with an output for each task that is to be learned (Caruana 1997). 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 in this common layer is the method by which inductive transfer occurs within an MTL network (Baxter 1996). MTL is proven to be a good method of knowledge transfer because it allows two or more tasks to share portions of the common feature layer to the extent to which it is mutually beneficial. The more that tasks are related, the more they will share representation and create positive inductive bias (Silver and Mercer 1996).

MTL can be defined as learning a set of target concepts, $\mathbf{f} = \{f_1, f_2, \dots f_k\}$, such that each $f_i : X \to Y$ 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. Therefore, Q 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, $S_{\text{MTL}} = \{(\mathbf{x}, \mathbf{f}(\mathbf{x}))\}$. The objective of the MTL algorithm is to find a set of hypotheses, $\mathbf{h} = \{h_1, h_2, \dots, h_k\}$, within its hypothesis space, H_{MTL}, that minimizes the objective function $\sum_{x \in S_{\text{MTL}}} \sum_{i=1}^{k} error[f_i(\mathbf{x}), h_i(\mathbf{x})]$. The assumption is that H_{MTL} contains sufficiently accurate h_i for each f_i being learned. Typically $|H_{\text{MTL}}| > |H_{\text{STL}}|$ in order to represent the multiple hypotheses. As with STL, a validation or tuning set of examples is used to prevent over-fitting and promote generalization.

3 Motivation for exploring an alternative to MTL

Machine lifelong learning, or ML3, a relatively new area of machine learning research, is concerned with the persistent and cumulative nature of learning (Thrun 1996). Lifelong learning considers situations in which a learner faces a series of different tasks and develops methods of retaining and using prior knowledge to improve the effectiveness (more accurate hypotheses) and efficiency (shorter training times) of learning.

Previously, we have investigated the use of MTL networks as a basis for developing an ML3 system and have found them to have several limitations related to the multiple outputs of the network (Silver and Mercer 2002; Silver and Poirier 2004; O'Quinn et al. 2005). First, the MTL approach 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 *virtual* target values but have found weaknesses related to the differences in the distribution of examples over the input space for various tasks (Silver and Mercer 2002; O'Quinn et al. 2005). We have also found a way to train multiple tasks with just one output target specified per example (Silver and Mercer 2002). The solution is to assign a special *unknown* target value to all other task outputs and have the back-propagation algorithm ignore these outputs when computing the cost function being minimized by the back-propagation algorithm.[1] This avoids having to generate corresponding virtual target values for the multiple tasks; however, the approach does not eliminate redundant task outputs.

It has been observed that inductive transfer in MTL networks is most beneficial when it comes from related tasks (Baxter 1996; Caruana 1997; Silver and Mercer 1996). Thus a measure of task relatedness is required in order for it to work optimally. This has remained an open question for over 20 years (Utgoff 1986). With MTL, shared representation is limited to the hidden node layer and is not considered at the output nodes. The theory is that optimal inductive transfer occurs when related tasks share the same hidden nodes (Baxter 1996). This perspective does not consider the sharing of knowledge at the example level 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. However, from an example level perspective, the tasks are partially related in that half of their examples are identical. Perhaps relatedness would be better judged at the example level of detail versus the task level.

There is also the practical problem of how an MTL-based lifelong learning agent would know to associate an example with a particular task. Clearly, the learning environment should provide the contextual cues; however, this suggests additional inputs and not outputs. A more subtle, but related, problem is managing redundant representation that can develop for nearly identical tasks in an MTL network. A lifelong learning system should be capable of practising a task and closely related tasks and improving its models with new examples over time. We have not been able to architect a solution that will scale up when there are multiple outputs per task. It is unclear how the build-up of redundant task outputs over time can be handled.

In response to the above problems, we have developed *context-sensitive* MTL, or *cs*MTL. The *cs*MTL method uses examples that have only one target output, but additional inputs that indicate the example *context*, such as the task to which it is associated. Section 4 describes the *cs*MTL network and the theory of how it functions.

3.1 Prior work on context input attributes

The idea of characterizing a portion of the input attributes as context is not new; however, it has not been studied as a component of inductive transfer until now. Related work on context-sensitive machine learning can be found in Matwin and Kubat (1996), Turney (1996a, 1996b). In Matwin and Kubat (1996) the importance of context to practical applications of machine learning for data mining is raised. The authors point out that context

[1] This technique is used in the experiments of Sect. 5 for several of the MTL comparisons.

information is often needed for the transfer of learned diagnostic rules from one group of patients to another. For a nice summary of approaches to identifying context attributes see Turney (1996a).

The survey article (Turney 1996b) discusses five strategies for managing context-sensitive features in supervised machine learning. Of relevance to this paper are the strategies of *context classifier selection*, *context classification adjustment* and *context weighting*. Contextual classifier selection considers the selection of a specialized classifier from a set of classifiers using the context inputs. The chosen classifier is used to classify the primary inputs. Contextual classification adjustment uses the same steps but in reverse order—first a classification is made by the primary inputs, then this base classification is adjusted by a context-level model that accepts the context inputs and the base classification. Finally, contextual weighting uses contextual features to weight the primary inputs, prior to classification. More importance is assigned to primary inputs that, in a given context, are more useful for classification. An extreme form of this can be used to contextually ignore certain primary input attributes, thus reducing the dimensionality of the input space.

Prior work on neural network based reinforcement leaning has used an approach similar to that proposed in this paper but did not refer to it as a method of inductive transfer (Santamaria et al. 1998; Gross et al. 1998). Typically, a separate Q function is learned for predicting the value of an action given the current state. Alternatively, it is possible to learn a single function, Q', for all actions over all states of an agent. This is particularly useful when both the states and actions are continuous in nature. In this case, the actions can be considered contextual inputs that select over the hypothesis space given the current state or vice versa. Previous researchers have recognized that this approach can develop models that generalize across both state and action spaces.

4 *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 tasks.[2] Similar to standard MTL neural networks, there are 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

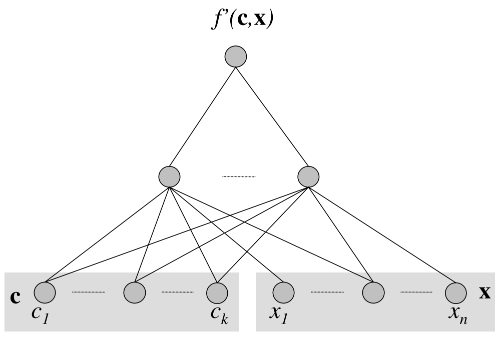


Fig. 2 A context-sensitive multiple task learning (*cs*MTL) network

[2]It is important to note that more outputs could be used for predicting a vector of values for each task. This is a subject for future work.

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.

Formally, let C be the set, $C = \{c | c \in \{0,1\}^t \wedge \sum_{i=1}^{t} c_i = 1\}$, representing the context of the primary inputs from X as described for MTL. The *cs*MTL method can be defined as learning a target concept, $f' : C \times X \to Y$; 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, $S_{cs\mathrm{MTL}}$, consists of all available examples for all tasks and includes the additional context inputs, $S_{cs\mathrm{MTL}} = \{(\mathbf{c}, \mathbf{x}, f'(\mathbf{c}, \mathbf{x}))\}$. The objective of the *cs*MTL algorithm is to find a hypothesis, h', within its hypothesis space, $H_{cs\mathrm{MTL}}$, that minimizes the objective function, $\sum_{x \in S_{cs\mathrm{MTL}}} error[f'(\mathbf{c}, \mathbf{x}), h'(\mathbf{c}, \mathbf{x})]$. The assumption is that $H_{cs\mathrm{MTL}} \subset \{f' | f' : C \times X \to Y\}$ contains a sufficiently accurate h'. Typically, $|H_{cs\mathrm{MTL}}| = |H_{\mathrm{MTL}}|$ 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. As with STL and MTL, a validation or tuning set of examples is used to prevent over-fitting and promote generalization.

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. During learning, $\mathbf{c}$ selects an inductive bias over $H_{cs\mathrm{MTL}}$ relative to the examples of secondary tasks being learned in the network. Once f' is learned, if $\mathbf{x}$ is held constant, then $\mathbf{c}$ indexes over the hypothesis space $H_{cs\mathrm{MTL}}$. Hence, $\mathbf{c}$ differentiates between otherwise conflicting examples and selects internal representation used by related tasks.

4.1 Constraint over the *cs*MTL hypothesis space

We have identified two important constraints that act on the *cs*MTL network to promote inductive transfer across the tasks so as to often produce superior hypotheses to standard MTL. The first is because of a relationship between back-propagation changes to the base bias weight of a hidden node and the changes to context weights of that same hidden node. The second is because of a relationship between the changes to the context weights of a hidden node and changes in the weight from that hidden node to the single output node.

4.1.1 Constraint between context and bias weights

Within sigmoid feed-forward neural networks, the output of a hidden node, j, is given by the sigmoid equation $o_j = 1/(1 - \exp^{\sum_i w_{ij} o_i + b_j})$; where w_{ij} is the weight from input node i (in previous layer) to hidden node j; o_i is the output from node i, and b_j is the bias term. If the inputs are divided into a set of primary inputs, $\{x_1, x_2, \ldots, x_p\}$, and a mutually exclusive set of contextual inputs, $\{c_1, c_2, \ldots, c_t\}$, the summation in the output equation can be expanded to $\sum_i w_{ij} o_i + b_j = c_1 w_{c_1 j} + c_2 w_{c_2 j} + \cdots + c_t w_{c_t j} + x_1 w_{x_1 j} + x_2 w_{x_2 j} + \cdots + x_p w_{x_p j} + b_j$.

Since the contextual inputs form a vector in which $c_z = 1$ when task z is selected, and all other contextual inputs are 0, the summation for an example for task z can be reduced to:

$$\sum_i w_{ij} o_i + b_j = x_1 w_{x_1 j} + x_2 w_{x_2 j} + \cdots + x_p w_{x_p j} + w_{c_z j} + b_j. \tag{1}$$

Thus, *cs*MTL networks operate by selecting a bias, $b_j + w_{c_z j}$, in each hidden node, according to the current task, z. Alternatively, one could say that they choose an appropriate offset, $w_{c_z j}$, to the base bias, b_j, in each node in the hidden layer. In this way $w_{c_z j}$ can be considered a *selective bias* for examples of task z.

Let us consider the updates to the context weights under the back-propagation algorithm in a *cs*MTL network. The algorithm adjusts weights between a node i, in one layer, and the node j in the layer above it, according to the equation, $\Delta w_{ij} = -\eta \frac{\partial E_k}{\partial w_{ij}} = \eta \delta_j o_i$; where η is the learning rate parameter, o_i is the output from the lower layer node i, and δ_j is a function of the error, E_k, observed at the output node, k, of the network. For a context input node, c_z, the notation becomes $\Delta w_{c_z j} = \eta \delta_j o_{c_z}$.

During training, input from the context nodes are all 0, except for the context node corresponding to the task for the given example, which has a value of 1. Therefore, for an example for task z, context node c_z has an output of $o_{c_z} = 1$, which leads to $\Delta w_{c_z j} = \eta \delta_j o_{c_z} = \eta \delta_j$, For all other context nodes t, $o_{c_t} = 0$ and $\Delta w_{c_t j} = \eta \delta_j o_{c_t} = 0$. Therefore, only context weight $w_{c_z j}$ is updated for an example for task z. In contrast, the bias term is updated for every example, and by the same amount, since it can be viewed as a weight connected to an input node with a permanent value of 1. Therefore, for each example of a task z, $\Delta b_j = \Delta w_{c_z j}$. And over all n training examples for all tasks z,

$$\sum_n \Delta b_j = \sum_n \sum_z \Delta w_{c_z j}. \tag{2}$$

4.1.2 *Constraint between context to hidden and output weights*

Given the sum of squared errors as the cost function used by the back-propagation algorithm, it can be shown that a change to the weight between hidden node j and output node k is given by

$$\Delta w_{jk} = -\eta \frac{\partial E_k}{\partial w_{jk}} = \eta \delta_k o_j; \tag{3}$$

where δ_k depends on the cost function. Similarly, it can be shown that the change to a weight between input node i and hidden node j is given by

$$\Delta w_{ij} = -\eta \frac{\partial E_k}{\partial w_{ij}} = \eta \delta_j o_i; \tag{4}$$

where $\delta_j = o_j(1 - o_j) \sum_k \delta_k w_{jk}$. The summation proportions the error for each of the outputs, k, in accord with the current weight, w_{jk}, leading to that output. In the case of a *cs*MTL network, when there is only one output node k, then $\delta_j = o_j(1 - o_j)\delta_k w_{jk}$ and, therefore,

$$\Delta w_{ij} = \eta \delta_j o_i = \eta[o_j(1 - o_j)\delta_k w_{jk}]o_i. \tag{5}$$

Rearranging (5) and substituting in Δw_{jk} from (3) we have:

$$\Delta w_{ij} = (1 - o_j)[\eta \delta_k o_j]w_{jk} o_i = (1 - o_j) w_{jk} \Delta w_{jk} o_i \tag{6}$$

or for a context input node, c_z, when $o_{c_z} = 1$ we have or

$$\Delta w_{c_z j} = (1 - o_j) w_{jk} \Delta w_{jk}. \tag{7}$$

Therefore, after several iterations through all training examples for all tasks, the change in any input to hidden node weight of a *cs*MTL network is constrained by the weight of the associated hidden to output node. In this way a change in all weights associated with a context node is constrained by the output weights shared by all tasks.

4.2 Comparing the number of free parameters of MTL and *cs*MTL

In a three-layer neural network, let us assume k is the number of tasks, h is the number of nodes in the hidden layer, and x is the number of input nodes. Each node in the hidden and output layers contains a number of parameters equal to the number of nodes in the lower layer plus one for the bias term. This leads to the following estimate of the number of free parameters of an MTL network, P_{MTL}:

$$P_{\mathrm{MTL}} = k(h+1) + h(x+1). \tag{8}$$

For a *cs*MTL network, assuming all parameters are free:

$$P_{cs\mathrm{MTL}} = (h+1) + h(k+x+1). \tag{9}$$

However, due to the relationship described in (2), we may express the value of the bias term as a function of the context weights, as follows:

$$b_j = b_j^0 + \sum_n \sum_z \Delta w_{c_z j} - \sum_z w_{c_z j}^0 \tag{10}$$

where b_j^0 is the initial bias in node j, and $w_{c_z j}^0$ is the initial value of the context weight for task z in node j. Therefore, the $k+1$ parameters in each hidden node for the context weights and the bias term represent, effectively, only k free parameters.

Revisiting our calculation of $P_{cs\mathrm{MTL}}$, given the knowledge that the context weights and bias in each node form only k free parameters, we have:

$$P_{cs\mathrm{MTL}} = (h+1) + h(k+x) \tag{11}$$

which suggests that for $k > 1$, we can expect $P_{cs\mathrm{MTL}} < P_{\mathrm{MTL}}$ by a difference of $k-1$. This suggests that fewer training examples will be required for learning under *cs*MTL than MTL.

4.3 Context inputs and relatedness between csMTL hypotheses

Consider that the vector of context inputs, **c**, is a set of mutually exclusive task identifiers per example. An early conjecture of our research was that relatedness between tasks could be measured by the similarity of the weight vector leading forward from each context input to the networks hidden nodes. The theory was that the single output node would constrain the back-propagation algorithm to develop similar weights for the context inputs of similar tasks. We now realize this is not correct.

If there is a large set of hidden nodes, and thus a large representational space, the back-propagation algorithm will develop a rich set of internal features via the input to hidden node weights. It is possible for two functionally equivalent hypotheses for identical tasks, trained from the same set of examples, to have different context to hidden node weight values. An examination of the weights from hypotheses for the same task, as described in the experiment of Sect. 5.4, has demonstrated this to be the case. We conclude that the context to hidden node weights cannot be used as a basis for judging task relatedness.

One might also conclude from the above that related *cs*MTL hypotheses do not provide inductive bias to each other during learning. This is not the case. The constraint $\Delta w_{c_z j} = (1 - o_j) w_{jk} \Delta w_{jk}$ will force the features chosen by a context input to produce an accurate output for examples that are in common across two or more tasks. Training examples that are the same except for a differing context input will generate hidden features that differ only by their context bias weight, $w_{c_k j}$. Over time, the back-propagated weight changes for the two examples will encourage a set of features that generate the same summed weight of their inputs to the networks output node. This functional constraint can be see as an additional form of inductive transfer. In Sect. 5.4 we will demonstrate that this constraint often produces models of superior performance as compared to MTL.

For the use of *cs*MTL in an agent or robot, if the vector of context inputs, **c**, is a set of real-valued inputs from the environment, then it can provide a grounded sense of relatedness for learning a new task with transfer from secondary tasks. Although we do not explore this in this paper, there is evidence from prior research (Matwin and Kubat 1996; Turney 1996a, 1996b) that real-valued contextual cues can make a significant difference during learning.

4.4 Does *cs*MTL work with other ML methods?

Sections 4.2 and 4.3 suggest a mathematical basis for why *cs*MTL neural networks require less data than MTL networks to develop a hypothesis of equal accuracy, where both networks have sufficient representation to develop such hypotheses. In Sect. 5, we provide empirical evidence to demonstrate that *cs*MTL networks develop models that perform better than MTL models. Given these positive results, we were curious to see if the *cs*MTL encoding approach would work with other machine learning algorithms. To explore this question, we have undertaken experiments using Inductive Decision Trees and Support Vector Machines. The results of these experiments are provided in Sect. 6.

5 Experimentation

This section presents a set of experiments that compares *cs*MTL and MTL neural networks in their ability to transfer knowledge with single task learning (STL) neural networks when there is no transfer. A set of seven task domains are explored—two synthetic and five real-world.

The first experiment compares the performance of hypotheses developed for the primary task of the synthetic Logic domain as the number of hidden nodes varies from 5 to 200. Similar preliminary experiments were carried out for all domains to ensure that each method had sufficient number of hidden nodes (free parameters) to develop the very best models. The second experiment examines the performance of hypotheses for the primary task of one domain as the number of training examples varies from 10 to 200. This focuses on the effectiveness of the methods as a function of sample complexity. The third experiment tests *cs*MTL's and MTL's ability to perform inductive transfer when the training examples for all tasks are drawn from the same function. This will determine each method's ability to transfer knowledge in the most optimistic of conditions. The final experiment compares the three neural network methods across all seven domains. This provides an assessment of the methods' abilities on a diversity of task domains.

It is important to note that the focus is on developing models from a small number of training examples for the primary task so as to observe the effect of inductive transfer. Under this condition, for several of the domains, a cross-validation approach would require

Table 1 Statistics on the five domains of tasks used in the experiments

Name	Tasks	Inputs	Primary Task		Secondary Task	
			Train	Tune	Test	Train
Logic	6	10	20	10	1000	200
Band	7	2	10	5	200	50
fMRI	2	24	48	8	24	48
CoverType	6	10	30	20	5714	50
Dermatology	6	33	20	10	358	40
Glass	6	9	19	10	150	35
HeartDisease	3	5	14	6	79	130

hundreds of repetitions and promote unbalanced training sets. For this reason, repeated trials using a random sampling approach was taken that ensured an even mix of positive and negative examples in the training and tuning sets.

All experiments were conducted using a neural network inductive transfer system developed at Acadia. The results for several of the domains have been confirmed by conducting similar runs using the MLP algorithm of the open source WEKA 3 machine learning package (Witten and Frank 2005).

5.1 The task domains

Seven domains have been studied using *cs*MTL. Table 1 shows a number of the statistics for each of these domains. Included is the number of tasks in the domain, the number of input attributes, the number of primary task training, tuning, and test set examples, and the number of examples used for each secondary task.

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 Silver and 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 *fMRI* domain challenges the learning systems to develop models that can classify 24 features extracted from fMRI images as a subject is reading a sentence or viewing a picture.[3] Inductive transfer between two subject models is examined; from subject T_1 for which good models could be developed to a second subject T_0 for which only poor models could be developed.

The *Covertype* and *Dermatology* domain can be found at the UCI ML repository. The Covertype domain contains data from four wilderness areas in northern Colorado. These data, representing information such as elevation and soil type, are to be used to determine the cover type, that is, the species of tree that grows there. There are six types of cover in

[3]Courtesy of the Brain Image Analysis Research Group and CALD, Carnegie Mellon University.

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. The Dermatology domain concerns the classification of six types of skin disease. Each data example contains 33 input skin attributes per patient along with their classification. This domain has under 400 examples but it has the largest input space of all of the domains.

The *Glass* and *Heart Disease* domain also come from the UCI ML repository. The Glass domain classifies 214 examples of glass as being one of six types, such as building windows, vehicle windows, containers, tableware, and headlamps. There are nine input attributes, including the refractive index of the example and its chemical make-up. The Heart Disease domain consists of clinical data from three hospitals in the USA and Hungary. The objective is to develop a hypothesis that can accurately predict the likelihood of a patient having a 50% or greater narrowing of one or more coronary arteries. Five input attributes from the original data was used in the study: age, gender, type of chest pain, resting blood pressure, and resting electrocardiogram.

The Covertype, Dermatology, Glass, and Heart Disease domains were originally single-task classification problems with examples that could be of n classes. Each was converted to a domain of n binary tasks, one task for each class, where each example indicates if it is of that class or not.

5.2 Comparison of methods varying number of hidden nodes

This experiment examines the performance of STL, MTL, and *cs*MTL hypotheses developed for the primary task of the synthetic Logic domain, and the real-world Dermatology, Glass and CoverType domains as the number of hidden nodes vary. Similar experiments were done for all domains to ensure that each method had sufficient number of hidden nodes, and thus free parameters, to develop the very best models.

5.2.1 Method

The STL and MTL networks were configured with the appropriate number of input and output nodes, respectively (see Table 1). The *cs*MTL networks had additional context inputs and only 1 output. The number of hidden nodes for all methods was varied from 5 to 200. The learning rate for each method was optimized between 0.01 and 0.001 through preliminary testing. The momentum term was fixed at 0.9 for all runs.

The objective is to learn the primary task of each domain (*e.g.* T_0 for the Logic domain) using an impoverished training set of examples (20 for the Logic domain) as shown in Table 1. The secondary tasks of each domain have sufficient training examples to develop models with accuracies greater than .75 using a STL network. Tuning sets with small numbers of examples for the primary task (6 for the Logic domain) are used to prevent the neural network from over-fitting.

It is important to note, that in the case of *cs*MTL, the training examples for the primary task are duplicated to ensure an equal number of training examples as that of each secondary task. In the case of the Logic domain, the 20 training examples for the primary task were duplicated nine times to make a total of 200 training examples, the same number as each of the secondary tasks. This is necessary to ensure a fair update to the weights for all tasks being learned by the *cs*MTL network which has a single output node. The duplication of primary examples does not help with MTL which has a separate output for each task. With duplicated examples, MTL develops hypotheses that quickly over-fit to the training data and have less generalization accuracy.

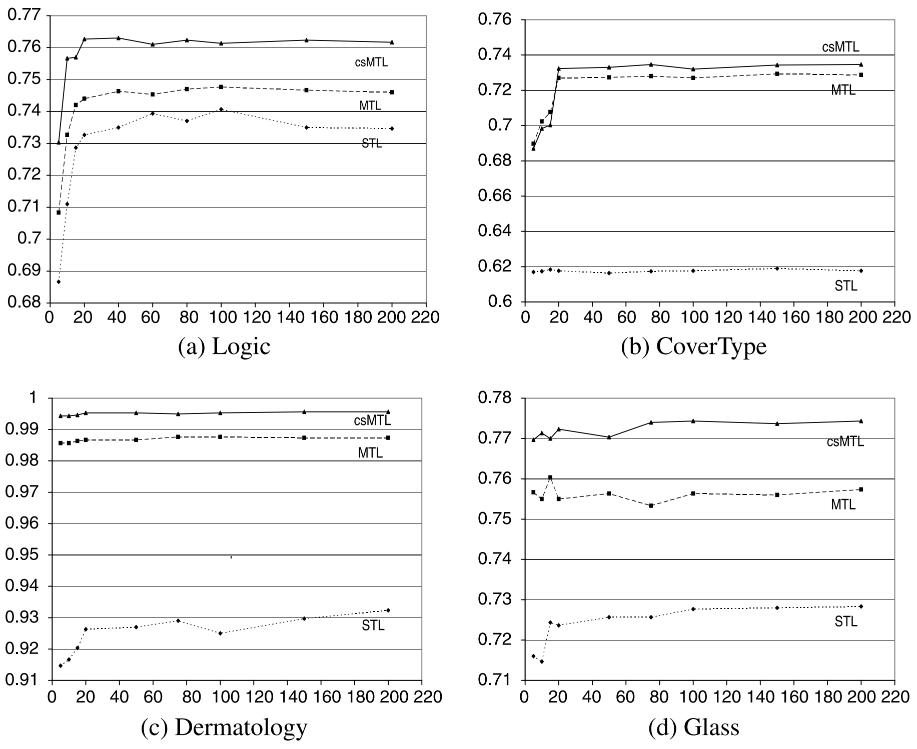


Fig. 3 Performance (test set accuracy) of hypotheses for the Logic, CoverType, Dermatology and Glass domains as a function of number of hidden nodes

Table 2 Results (p-values) from difference of means T-Tests comparing *cs*MTL to STL and MTL models on the four domains

Domain	*cs*MTL *vs* STL	*cs*MTL *vs* MTL
Logic	1.357E-06	5.228E-08
CoverType	2.415E-07	0.251
Dermatology	2.564E-10	2.278E-11
Glass	4.301E-10	4.432E-07

An independent test set of examples is used to determine hypothesis performance (see Table 1). A network output of 0.5 or greater is considered a positive classification. The mean accuracies reported are from 5 repeated trials. For each repeated trial, the examples in the data sets were randomly sampled from the available data.

5.2.2 *Results*

Figure 3 shows the mean accuracy of models for the four tasks developed under the three methods as the number of hidden nodes increases. The p-values from a difference of means T-test over the models for each task is shown in Table 2. All graphs show that the performance of the MTL and *cs*MTL models become statistically stable at 20 hidden nodes or more. The STL models plateau at or before 100 hidden nodes. Therefore, all methods are robust given sufficient representation and early stopping using a validation set to prevent over-fitting.

The MTL and *cs*MTL models perform statistically better than the STL models on all four domains of tasks. The csMTL models perform statistically better than MTL models on the Logic, Dermatology and Glass domains but at the same level of accuracy on the CoverType domain. MTL, which takes advantage of shared representation and knowledge transfer from the secondary tasks, produces better models than STL. *cs*MTL, constrained by the common output node and biased by the context inputs is able to build the most accurate hypotheses for three of the domains.

5.3 Comparison of methods varying number of training examples

This experiment examines the performance of hypotheses developed for the primary Logic domain task as the number of training examples varies from 10 to 200. This focuses on the effectiveness of the methods as a function of sample complexity.

5.3.1 Method

STL, MTL, and *cs*MTL networks were configured for the Logic domain as described in Sect. 5.2, but with a fix number of 20 hidden nodes, sufficient for learning under all methods.

All three networks are presented with training sets of size 10 to 200 for the primary task, increasing the number of examples by 10. For MTL and *cs*MTL, the five secondary tasks had training sets of 200 examples, which is sufficient to build accurate hypotheses for this domain. For *cs*MTL, the training examples for the primary task are duplicated as many times as necessary to ensure 200 training examples for each run—the same number as that of each secondary task. As discussed in Sect. 5.2, this is to ensure a fair update of network weights for each task being learned by the *cs*MTL network. Thus, for *cs*MTL, variation in the “number of training examples” means a variation in the number of unique examples in the training set.

A tuning set of 20 examples for the primary task is used during training, and an independent test set of size 1000 is used to measure accuracy. Mean test set accuracies from three repeated studies each with random draws of examples are compared.

5.3.2 Results

Figure 4 shows the mean accuracy of models developed under the three methods as the number of primary task examples increases. The results show that *cs*MTL hypotheses are consistently better than STL models, and beyond 30 training examples they are consistently better than MTL models. This further suggests that the *cs*MTL method is able to selectively transfer knowledge from the more related secondary tasks, without an explicit measure of relatedness.

5.4 Comparison of methods on a domain of equivalent tasks

This experiment tests the ability of the MTL and *cs*MTL methods to transfer knowledge from five secondary tasks to a primary task when the training examples for all tasks are drawn from the same function. The models are compared to STL models developed from a combination of all the training data. This will assess the ability of each method to transfer knowledge in the most optimistic of conditions. Despite the mutually exclusive context inputs, we expect that *cs*MTL will be able to beneficially combine the examples from the six training sets during model development.

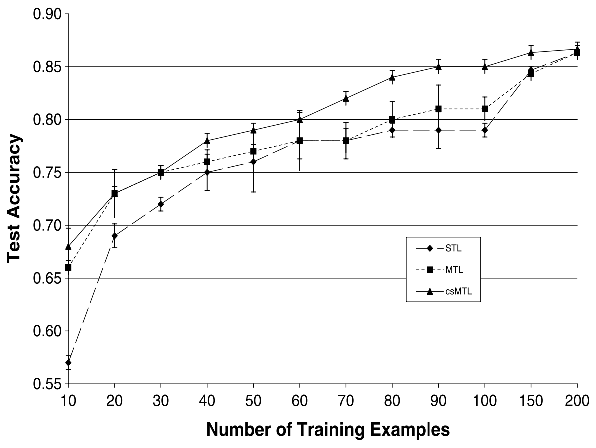


Fig. 4 Performance of hypotheses for the Logic domain as a function of number of training examples

5.4.1 Method

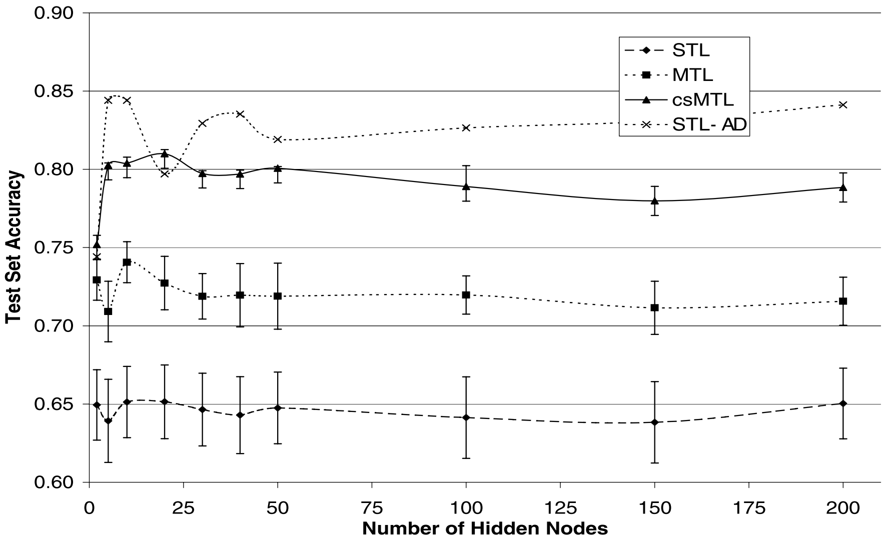
The primary tasks for the synthetic Logic domain and the real-world Glass, Dermatology and CoverType domains were used to generate four new domains called *Logic, Glass, Derm* and *Cover*, each consisting of six tasks. The training examples for each task of a domain were drawn from the original primary task for that domain; 20 examples per task for the Logic domain, 10 for Glass, 20 for Derm and 30 for Cover. For each domain, the six sets of training data were also combined into a large training set call STL-AllData. The STL-AllData training set allowed us to develop the best possible primary task model for each of the new domains using STL.

A *cs*MTL network was configured for each of the domains as described in Sect. 5.3 with one output node, a layer of hidden nodes sufficient for learning all tasks (20 for the Dermatology, Glass domains and 30 for the Covertype domain) and a layer of input nodes equal to the number of primary inputs plus the number of tasks in the domain. For the Logic domain the number of hidden nodes for all methods was varied from 2 to 200. The STL networks used were identical to the *cs*MTL networks less the context inputs. The MTL networks were the same as the STL networks but with one output for each task. Independent test sets for the primary task of each domain (see Table 1) were used to measure model accuracies. Mean accuracies from three repeated studies were computed to compare model performance.

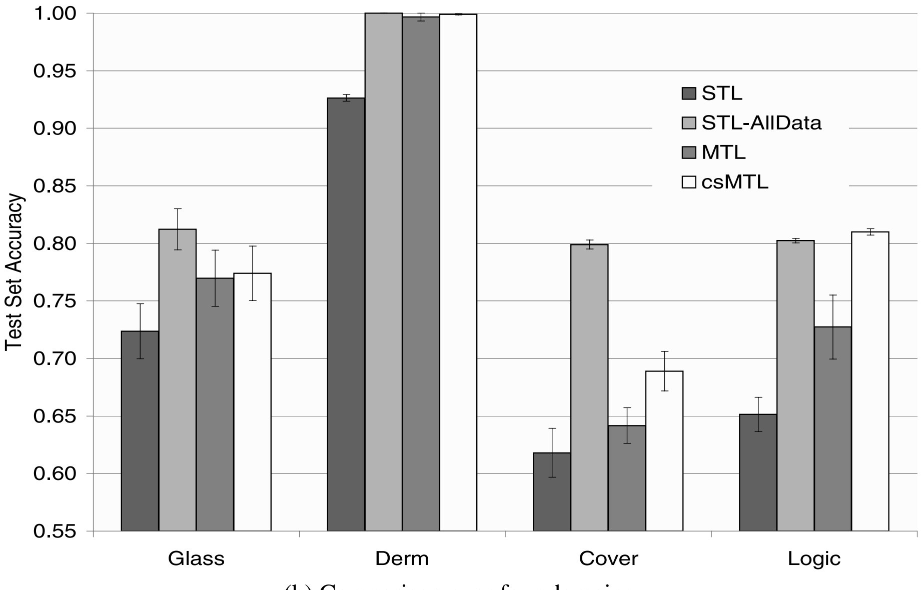
As an additional study, the examples for one of the equivalent secondary tasks was replaced by the same number of examples of another task from each domain. In the case of the synthetic Logic domain, we chose replacement tasks that were known to be least related to the primary task. For the real-world domains, secondary tasks and examples were randomly chosen. The intention was to observe the effect on the MTL and *cs*MTL methods as the diversity of tasks increased. The replacement of an equivalent task was repeated three times until there were only two secondary tasks left with examples drawn from the same function as the primary task. As before, independent test sets for the primary task of each domain were used to compute mean accuracies so as to measure method performance.

5.4.2 Results

Figure 5(a) and (b) show the performance of each of the methods on the test sets for the primary task for each domain. As expected, the combined data of STL-AllData produced



(a) Logic equivalent-task domain.



(b) Comparison over four domains.

Fig. 5 Comparison of STL, STL-AllData, MTL and csMTL models on four equivalent-task domains

the best models. The MTL and *cs*MTL hypotheses are typically much more accurate than the STL models developed from only small numbers of examples. The *cs*MTL hypotheses are always as good as the MTL hypotheses and in the case of the Logic and Cover domains the *cs*MTL hypotheses are statistically more accurate. The results indicate that inductive bias occurs in the *cs*MTL networks between the examples of the tasks to a level that is as good as or better than the MTL networks.

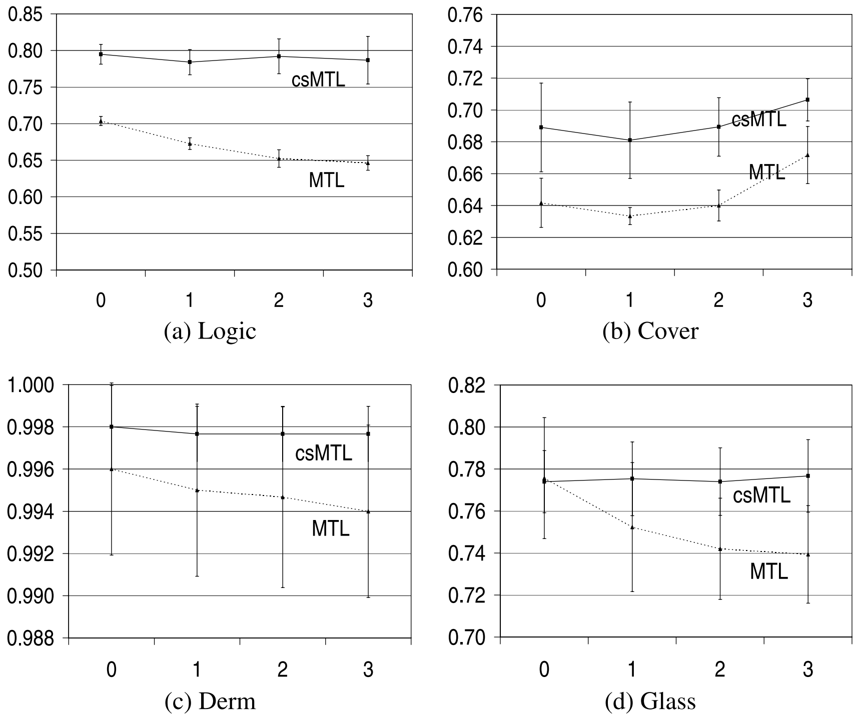


Fig. 6 Test accuracy (95% conf) of MTL and csMTL methods on the equivalent-task domains as the number of less related tasks increases.

Figure 6 shows how the performance of the MTL and *cs*MTL methods vary as the number of less related secondary tasks increases in each domain. Clearly, the MTL method is more affected by the variety of related tasks as compared to the *cs*MTL method. The results suggest that selective inductive bias is occurring in the *cs*MTL network more effectively than in the MTL network. The results on the Cover domain indicate that the last two secondary tasks actually provided beneficial inductive bias to the primary task.

5.5 Comparison of methods on the seven task domains

The final experiment compares the STL, MTL and *cs*MTL neural network methods across the seven domains of tasks. This provides an assessment of the methods' abilities on a diversity of synthetic and real-world task domains.

5.5.1 Method

A *cs*MTL network was configured for each domain with one output node, a layer of hidden nodes sufficient for learning all tasks (30 for the Band, 20 for the Logic, 10 for the fMRI, 30 for the Covertype and 20 for the Dermatology, Glass and HD domains) and a layer of input nodes equal to the number of primary inputs plus the number of tasks in the domain. The STL networks used were identical to the *cs*MTL networks less the context inputs. The MTL networks were the same as the STL networks but with one output for each task.

For all five domains, the objective is to learn the primary task using an impoverished training set of examples as shown in Table 1. Each of the other tasks of the domain have 35

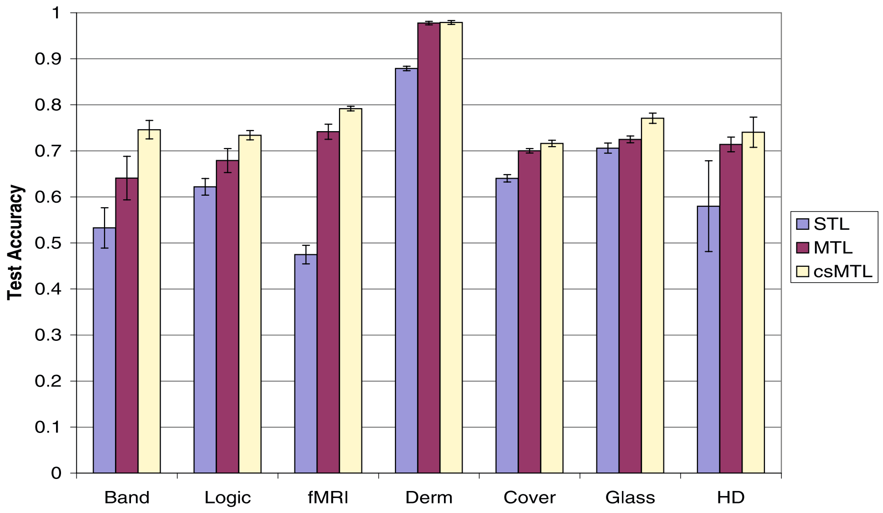


Fig. 7 *cs*MTL compared to STL and previous MTL methods. Shown is the mean test set accuracy for primary task hypotheses for each of seven domains of tasks

or more training examples that have been demonstrated to develop models with accuracies greater than .75 using a STL network. A small tuning set of examples for the primary task of each domain is used to prevent the neural network from over-fitting to the impoverished training sets.

As described in Sect. 5.2, in the case of *cs*MTL, the training examples for the primary task are duplicated as many times as necessary to ensure an equal number of training examples to that of each secondary task. This ensures a fair update to the weights for all tasks being learned by the *cs*MTL network. As discussed in Sect. 5.2, this duplication of primary examples does not help MTL in the same manner.

An independent test set is used to determine hypothesis performance. See Table 1 for details on the number of examples in each of these data sets. A network output of 0.5 or greater is considered a positive classification. The mean accuracies reported are from repeated trials (10 for the Band, fMRI, Heart Disease and Glass domains, 30 for the Logic, Covertype, and Dermatology domains). For each repeated trial, the examples in the data sets were randomly sampled from the available data ensuring a balance of positive and negative examples.

5.5.2 Results

Figure 7 and Table 3 show the results for the seven domains. They compare the mean test set accuracy of the *cs*MTL hypotheses developed for the primary tasks to hypotheses developed with no inductive transfer under STL and with transfer under standard MTL. The MTL and *cs*MTL results demonstrate the advantage of knowledge transfer with mean accuracies that are significantly better than STL models for all domains. Furthermore, *cs*MTL performs significantly better than MTL on all domains except Dermatology and Heart Disease. The results suggest that the *cs*MTL method is at least as good as standard MTL. It is able to transfer knowledge from shared internal representation to a primary task when training on examples of a mixture of tasks that are more or less related.

Table 3 Results (p-values) from difference of means T-Tests comparing *cs*MTL to STL and MTL models on the seven domains

Domain	*cs*MTL *vs* STL	*cs*MTL *vs* MTL
Band	0.00128	0.04805
Logic	1.07276E-12	0.00027
fMRI	6.40698E-06	0.00402
Dermatology	4.10918E-26	0.52559
Covertype	8.47545E-13	0.00200
Glass	3.76788E-06	0.00011
HeartDisease	0.02256	0.20922

6 *cs*MTL and other machine learning methods

The following experiments explore the use of encoding multiple tasks using context inputs with two other machine learning methods: IDTs and SVMs. Single task learning is compared to *cs*MTL learning under the two methods in the hope that inductive transfer is observed.

6.1 *cs*MTL and IDT models

Inductive Decision Trees, or IDTs, are a family of machine learning methods that divides the input space into class regions based on input attribute-values. The classic ID3 and C4.5 algorithms recursively partitions a subspace, initially the entire input space, into two separate subspaces by determining the attribute that provides the greatest gain in properly classify the training examples (Quinlan 1993). IDTs are significantly different from neural networks, in terms of their learning algorithm and model representation. We were curious to see if IDTs could take advantage of *cs*MTL encoding of multiple tasks in the same way as back-propagation neural networks.

6.1.1 Method

The objective is to compare *cs*MTL with STL IDTs using the Logic domain. The same primary task and data used in Sect. 5.3 was used for this experiment. Similarly, we examined the effect of varying the number of primary task examples from 10 to 100. There were 200 examples for each of the secondary tasks in the *cs*MTL scenario. A set of 1000 primary task examples is used to test prediction accuracy of the tree for task T_0.

The trees were built using the J48 algorithm of open source software WEKA 3 (Witten and Frank 2005). Preliminary experiments were conducted varying the branching confidence level from 0.10 to 0.40 and the minimum number of instances per leaf from 2 to 30. A confidence value of 0.25 and minimum number of instances per leaf of 15 were found to be the best choices for the two methods.

6.1.2 Results

Figure 8 shows no significant difference between the *cs*MTL and STL methods, regardless of the number of primary task examples. Both methods benefit from an increase in the number of training examples, but IDTs do not develop better models from related task examples when they are encoded using additional context inputs.

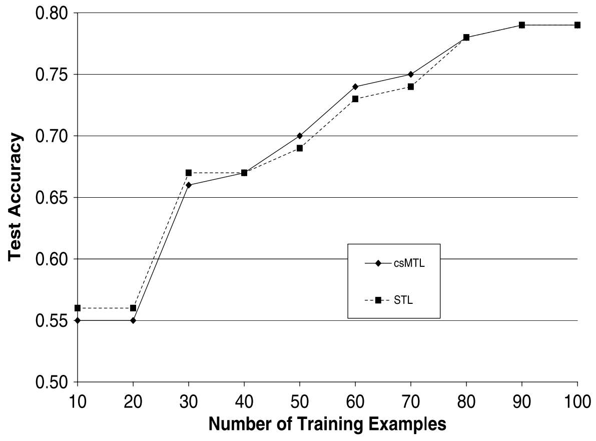


Fig. 8 *cs*MTL and STL IDT performance on the Logic domain. Shown is the mean test set accuracy for T_0 hypotheses

6.2 *cs*MTL and SVM models

Support Vector Machines, or SVMs, bear some resemblance to neural networks in that they develop a multi-layered functional model. The lower layer projects the inputs into a feature space, and computes a value based upon a weighted combination of these features in order to perform classification (Boser et al. 1992; Smola and Schoelkopf 1998).

However, there are also significant differences between the two models. The training in SVMs is an optimization problem which does not depend upon the initial values of free parameters. Also, the combination of kernel mapping functions as well as the selection of a varying set of support points and parameters allows for very complex decision boundaries.

This combination of similarities and differences between SVMs and neural networks make SVMs an appealing machine learning method to study with multiple tasks examples encoded using context attributes.

6.2.1 Method

This experiment compares the mean accuracy of STL and *cs*MTL SVM models for the T_0 task of the Logic domain. The experiment was developed in the R statistical programming language using the SVM implementation in the e1071 library, which is based upon libSVM (Chang and Lin 2001).

Each simulation consisted of selecting 20 examples at random from a pool of 1020 examples, under the constraint that 10 selected examples were of each class. These 20 examples were used as the training set for T_0, and the remaining 1000 examples were retained as the independent test set. For each simulation, two SVMs were trained. The first was a single task SVM using only the 20 training examples. The second was an SVM which used 200 examples from each of 5 other related tasks marked using additional contextual attributes, and the 20 training examples were duplicated another 9 times to provide an equal degree of weighting with the other tasks. The prediction accuracy of each SVM was determined using the 1000 examples remaining in the test set for T_0.

In order to select appropriate values for the kernel type and parameters for the STL and *cs*MTL SVMs, 50 repetitions were performed for:

- linear kernels with constraint violation costs of 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, and 1
- radial basis kernels with combinations of gamma equal 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 1, with constraint violation costs of 0.001, 0.005, 0.01, 0.05 , 0.1, 0.5, and 1

The best mean and standard deviations of prediction accuracy are reported for each method.

6.2.2 Results

The mean accuracy for the best STL SVM model was 71.4% with a 3.3% standard deviation, occurring for a SVM model trained with a radial basis kernel using a gamma of 0.8 and a constraint violation cost of 1.

The mean accuracy for the best *cs*MTL SVM model built using primary and secondary task data with contextual inputs was 72.4% with a 3.9% standard deviation, occurring for a SVM model trained with a radial basis kernel using a gamma of 0.1 and a constraint violation cost of 1.

The results indicate that there is no significant difference between STL and *cs*MTL SVM models that use linear or radial basis kernels. In future work we intend to explore other kernels that may support *cs*MTL encoded examples.

6.3 Discussion

The IDT method demonstrated no significant difference in performance between a single task model, and a model given additional examples for related tasks using contextual attributes. This shows that the IDT method does not experience inductive transfer through the use of contextual attributes with data from related tasks. An analysis of several of the trees that were developed indicate that the reason for this is that IDTs tend to create decision boundaries using the context input attributes that lead to separate representations for each of the tasks.

Similarly, the SVM method, using linear or radial basis kernels, neither improved nor degraded from the use of contextual attributes with additional data from other tasks. The SVM method with the given kernels does not appear to directly experience inductive transfer through the use of contextual attributes in the same way as the neural networks. This may be because with SVM, the radial basis kernel,

$$K(x, x') = \exp(-\gamma \|x - x'\|^2) \tag{12}$$

produces higher values for the support vectors which belong to the same task as the example being tested. Due to the difference in the contextual components of the input vectors, $\|x - x'\|^2$ will be greater by 2 for support vectors from different tasks than for support vectors from the same task. This effect will result in lower values for $K(x, x')$ for support vectors from different tasks, which in at least some situations may limit the degree of inductive transfer from other tasks.

Based on these results, we conjecture that the performance improvement in *cs*MTL over MTL networks is not due to the transformation of the data from a multiple output format to a single output format using contextual inputs, but rather is due to the manner in which a neural network develops a model given this transformation of the data. As shown in Sect. 4, the back-propagation algorithm places functional constraints on the parameters of the model, which reduces the number of examples available to find an accurate hypothesis.

7 Hints to why *cs*MTL works so well

Another area for consideration in explaining the observed benefits to performance under *cs*MTL is Abu-Mostafa's extended VC Dimension, $VC(G;H)$, and the application of *virtual examples* (Abu-Mostafa 1995).

Abu-Mostafa has systematically researched the use of *hints* for reduction of the number of necessary training examples for selecting accurate hypotheses. Hints are defined as properties of the primary task or the task domain that are known to be true although independent of the primary training examples, such as monotonicity or symmetry of the output with respect to the inputs. Hints can be expressed as *virtual examples* that are used to train a single task network. In fact, Abu-Mostafa considers primary task training examples to be just another form of hint. By minimizing the error across all of the hint examples, a more accurate hypothesis for the primary task can be produced.

Within the context of a *cs*MTL network, examples for secondary related tasks could be considered virtual examples for the primary task. In future work, we intend to explore the *cs*MTL approach in the light of Abu-Mostafa's theory.

8 Summary and conclusion

This paper has presented *cs*MTL as a method of inductive transfer that uses a single output neural network and additional *context* inputs for learning multiple tasks. The method was developed in response to problems we had encountered in using MTL networks for developing machine lifelong learning (ML3) systems. Our goal was not to develop a new learning algorithm, but rather to propose a new method of learning multiple tasks with standard single task learning algorithms.

The introduction of context attributes to distinguish task examples has been found to improve predictive performance of the models developed under the back-propagation algorithm using a MSE cost function. Experimentation on seven domains of tasks has demonstrated that *cs*MTL may often produce hypotheses for a primary task that are significantly better than standard MTL hypotheses when learning in the presence of related and unrelated tasks.

The paper provides a theoretical justification for the *cs*MTL performance improvement over MTL. The structure of the *cs*MTL network, with only one output and context inputs, combined with the back-propagation training algorithm results in a network which has a lower number of free parameters than an MTL network with the same number of hidden nodes. This partially explains the positive results we have experienced with using csMTL for inductive transfer—the reduced dimensionality of the free parameter space allows csMTL networks to require less training data than MTL networks to achieve similar levels of accuracy for the same mixture of tasks. We have also shown mathematically that the network configuration may place additional constraints on the remaining parameters as a function of having only one output node and therefore only one backward propagated error signal.

An examination of using classifier IDTs and SVMs (with linear and radial basis kernels) using the same *cs*MTL data has indicated that the improvement in model performance is not shared by other machine learning methods, in general. Inductive transfer occurs in neural networks, at least in part, because of functional constraints imposed on the weights of the neural network model by the learning algorithm and the format of the *cs*MTL input data. It is possible that this phenomenon may occur in other machine learning methods that we have yet to examine.

8.1 Benefits and limitations of *cs*MTL

The *cs*MTL method of encoding examples with context attributes was utilized in response to problems that we encountered when using MTL networks to develop an ML3 system. We have discovered that the approach satisfies most of these problems while at the same time delivering superior inductive transfer.

- The *cs*MTL examples discussed in this paper require one context input per task but only one output. No corresponding secondary task target values are required.
- The *cs*MTL method shifts the focus to learning a completely shared representation for all tasks of the domain. The context inputs for each example can be seen as a selective bias that indexes over the domain of tasks during and after model development.
- The *cs*MTL method has been shown to develop superior models to that of MTL in the presence of completely related tasks and a mix of related and unrelated tasks. There is likely a class of tasks for which *cs*MTL does not work as well as MTL. The characterization of this class of tasks will be an important next step in our research.

The *cs*MTL method is not without its limitations. One of the consequences of a more highly constrained hypothesis space is that, under certain conditions, a larger amount of internal representation may be required in order to store the same number of hypotheses as MTL. Although we have not yet observed this effect, we feel it is worthy of further testing.

We have observed increases in training times using *cs*MTL over MTL neural networks when there are equal numbers of training examples. This is because of the larger number of *cs*MTL training examples and the relatively small learning rates required to produce good models. This study has focused on model effectiveness; we leave a thorough examination of the efficiency of model development to future work.

The *cs*MTL approach suffers from the same scaling problems as standard back-propagation neural network systems. The computational complexity of the standard back-propagation algorithm is $O(W^3)$, where W is the number of weights in the network. This provides a strong motivation for discovering other standard machine learning algorithms which can benefit from the use of context attributes in a similar way.

8.2 Future work

Our long-term goal is to develop an ML3 methodology and related system, based on *cs*MTL, which is capable of sequential knowledge retention and inductive transfer. The system is meant to satisfy a number of ML3 requirements including the effective and efficient consolidation of task knowledge into a long-term network using task rehearsal (Silver and Mercer 2002), and effective and efficient inductive transfer during new learning. With this is long-term goal in mind, our short-term directions for future research include:

- Performing a detailed analysis of the contribution of shared output node representation on the reduction of number of necessary training examples. We believe that this does not reduce the number of free parameters, per se, but rather reduces the effective size of hypothesis space. Both mathematical analyses and experimentation will be necessary in order to confirm or deny this belief.
- Examining the conditions under which *cs*MTL networks will produce less accurate hypotheses than MTL networks. The reduction in the number of free parameters, and the effective modification to the functional form of the model leads us to believe this may occur. In order to best apply *cs*MTL, it is important to have an understanding of the conditions under which it is best used.

- Investigating more optimal methods of ensuring that small numbers of examples for the primary task under *cs*MTL are not overwhelmed by large numbers of examples for each of the secondary tasks. In this research we simply duplicate the primary task examples to equal the number for each secondary task. More optimal methods are likely possible.
- Experimenting with other machine learning methods to determine which methods, and more importantly, which attributes of methods lead to better results given multi-task data structured as per *cs*MTL.
- Exploring if Abu-Mostafa's theory of hints can be applied directly, or with some extensions, to explain the effects of *cs*MTL learning, and even of multi-task learning models in general, when the multiple tasks are not the same task.
- Investigating the use of real-valued context inputs from the environment as a grounded source of task relatedness.
- Investigating domains of tasks that have multiple outputs per task, such as image transformation tasks.

References

Abu-Mostafa, Y. S. (1995). Hints. *Neural Computation*, *7*, 639–671.

Allenby, G. M., & Rossi, P. E. (1999). Marketing models of consumer heterogeneity. *Journal of Econometrics*, *89*, 57–78.

Allenby, G. M., & Rossi, P. E. (2005). Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, *6*, 615–637.

Arora, N., Allenby, G. M., & Ginter, J. (1998). A hierarchical Bayes model of primary and secondary demand. *Marketing Science*, *17*(1), 29–44.

Bakker, B., & Heskes, T. (2003). Task clustering and gating for Bayesian multi-task learning. *Journal of Machine Learning Research*, *4*, 83–99.

Baxter, J. (1996). Learning model bias. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Advances in neural information processing systems* (Vol. 8, pp. 169–175). Cambridge: The MIT Press.

Baxter, J. (1997). Theoretical models of learning to learn. *Learning to Learn*, 71–94.

Ben-David, S., & Schuller, R. (2003). Exploiting task relatedness for multiple task learning. In *Proceedings of computational learning theory (COLT)* (pp. 185–192).

Boser, B. E., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Computational learning theory* (pp. 144–152).

Breiman, L., & Friedman, J. H. (1998). Predicting multivariate responses in multiple linear regression. *Royal Statistical Society Series B*, *1*, 3–54.

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

Chang, C., & Lin, C. (2001). *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Greene, W. (2002). *Econometric analysis* (5th ed.). Englewood Cliffs: Prentice-Hall.

Gross, H., Stephan, V., & Krabbes, M. (1998). A neural field approach to topological reinforcement learning in continuous action spaces. In *Procedings of the international joint conference on neural networks (IJCNN'98)* (pp. 1992–1997). Anchorage, IEEE Press.

Heskes, T. (2000). Empirical Bayes for learning to learn. In P. Langley (Ed.), *Proceedings of the international conference on machine learning (ICML'00)* (pp. 367–374).

Jebara, T. (2004). Multi-task feature and kernel selection for svms. In *Proceedings of the international conference on machine learning (ICML'04)* (pp. 185–192).

Matwin, S., & Kubat, M. (1996). The role of context in concept learning. In *Proceedings of ICML-96, workshop on learning in context-sensitive domains* (pp. 1–5). Bari, Italy.

O'Quinn, R., Silver, D. L., & Poirier, R. (2005). Continued practice and consolidation of a learning task. In *Proceedings of the meta-learning workshop, 22nd international conference on machine learning (ICML 2005)*. Bonn, Germany.

Quinlan, R. J. (1993). *C4.5: programs for machine learning*. Los Altos: Morgan Kaufmann.

Santamaria, J., Sutton, R., & Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, *6*, 163–218.

Silver, D. L., & McCracken, P. (2003). Selective transfer of task knowledge using stochastic noise. In Y. Xiang & B. Chaib-draa (Eds.), *Advances in artificial intelligence, 16th conference of the Canadian society for computational studies of intelligence (AI'2003)* (pp. 190–205). New York.

Silver, D. L., & Mercer, R. E. (1996). The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Connection Science Special Issue: Transfer in Inductive Systems*, *8*(2), 277–294.

Silver, D. L., & Mercer, R. E. (2002). The task rehearsal method of life-long learning: overcoming impoverished data. In *Advances in artificial intelligence, 15th conference of the Canadian society for computational studies of intelligence (AI'2002)* (pp. 90–101).

Silver, D. L., & Poirier, R. (2004). Sequential consolidation of learned task knowledge. In *Lecture notes in artificial intelligence, 17th conference of the Canadian society for computational studies of intelligence (AI'2004)* (pp. 217–232).

Silver, D. L., & Poirier, R. (2005). *Requirements for machine lifelong learning* (Jodrey School of Computer Science, TR-2005-009). November.

Smola, A. J., & Schoelkopf, B. (1998). *A tutorial on support vector regression* (Technical Report NC2-TR-1998-030). NeuroCOLT2.

Thrun, S. (1996). Is learning the nth thing any easier than learning the first?. *Advances in Neural Information Processing Systems*, *8*, 8.

Thrun, S., & Pratt, L. Y. (Eds.) (1997). *Learning to learn*. Boston: Kluwer Academic.

Turney, P. D. (1996a). 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* (Vol. NRC 39222, pp. 53–59). Bari, Italy.

Turney, P. D. (1996b). The management of context-sensitive features: A review of strategies. In *13th international conference on machine learning (ICML96), workshop on learning in context-sensitive domains* (Vol. NRC 39222, pp. 60–65). Bari, Italy.

Utgoff, P. E. (1986). *Machine learning of inductive bias*. Boston: Kluwer Academic.

Witten, I. H., & Frank, E. (2005). *Data mining: practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Zellner, A. (1962). An efficient method for estimating seemingly unrelated regression equations and tests for aggregation bias. *Journal of the American Statistical Association*, *57*, 348–368.