Multitask Learning: A Knowledge-Based Source of Inductive Bias[1]

Richard A. Caruana
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
caruana@cs.cmu.edu

Abstract

This paper suggests that it may be easier to learn several hard tasks at one time than to learn these same tasks separately. In effect, the information provided by the training signal for each task serves as a domain-specific inductive bias for the other tasks. Frequently the world gives us clusters of related tasks to learn. When it does not, it is often straightforward to create additional tasks. For many domains, acquiring inductive bias by collecting additional teaching signal may be more practical than the traditional approach of codifying domain-specific biases acquired from human expertise. We call this approach Multitask Learning (MTL). Since much of the power of an inductive learner follows directly from its inductive bias, multitask learning may yield more powerful learning. An empirical example of multitask connectionist learning is presented where learning improves by training one network on several related tasks at the same time. Multitask decision tree induction is also outlined.

1 INTRODUCTION

Inductive bias is anything that causes an inductive learner to prefer some hypotheses over other hypotheses. Bias-free learning is impossible; in fact, much of the power of an inductive learner follows directly from the power of its inductive bias (Mitchell 1980). Though many different sources of inductive bias have been used, these do not yet appear to be powerful enough to enable machine learning to develop inductive systems that scale to complex real-world tasks.

The standard methodology in machine learning is to break large problems into small, reasonably independent subproblems, learn the subproblems separately, and then recombine the learned pieces (Waibel 1989). This paper suggests that part of this methodology may be wrong. The reductionist method has caused us to attempt learning on simple, isolated tasks before earnestly attempting learning on larger, richer tasks. By adhering to this method we may be ignoring a critical source of inductive bias for real-world problems: the inductive bias inherent in the similarity of related tasks drawn from the same domain. If an inductive learner is given several related tasks at the same time, these tasks can be used as valuable sources of inductive bias for each other. This may make learning faster or more accurate and may allow hard tasks to be learned that are not learnable in isolation. We call this approach Multitask Learning (MTL).

First we introduce multitask learning at an abstract level and explain how related tasks can be used as sources of mutual inductive bias. Then we present a concrete example of MTL applied to artificial neural networks. After this demonstration we discuss multitask connectionist learning in more detail. Then we briefly describe multitask decision-tree learning. After this we discuss related work, most notably that on giving hints to networks and on transferring knowledge between related tasks learned one at a time. Finally, we summarize the MTL methodology and highlight the research problems that will have to be tackled for MTL to succeed on complex, real-world tasks.

2 MULTITASK LEARNING AND INDUCTIVE BIAS

We learn embedded in a world that simultaneously tasks us to learn many related things. These things obey the same physical laws, derive from the same human culture, are preprocessed by the same sensory hardware, etc. Perhaps it is the similarity of the tasks we learn that enables us to learn so much with so little experience.

A child trained from birth on a single, isolated, complex task—and nothing else—would be unlikely to learn it. If you were trained from birth to play tennis—and nothing else—you would probably not learn tennis. Instead, you learn to play tennis in a world that tasks you to learn many other things. You also learn to walk, to run, to jump, to exercise, to grasp, to throw, to swing, to recognize objects, to

[1]To appear in The Proceedings of the Tenth International Conference on Machine Learning

predict trajectories, to rest, to talk, to study, to practice, etc. These tasks are not all the same: running in tennis is different from running on a track, yet they are related. Perhaps the similarities between the thousands of tasks you learn are what enable you to learn any one of them—including tennis.

An artificial neural network (or a decision tree) trained tabula rasa on a single, isolated, complex task is unlikely to learn it. For example, a neural network with a 1000 by 1000 pixel input retina will be unlikely to learn to recognize complex objects in the real world given the number of training patterns and the amount of training time we likely can afford. Instead, it may be better to embed the inductive learner in an environment that simultaneously tasks it to learn many related tasks. If these tasks share computed subfeatures or learnable subprocesses, the inductive learner may find that it is easier to learn them together than to learn them in isolation. Thus if we simultaneously train the neural network to recognize objects and object outlines, shapes, edges, regions, subregions, textures, reflections, highlights, shadows, text, orientation, size, distance, etc., it may be better able to learn to recognize complex objects in the real world. This is Multitask Learning (MTL).

MTL is predicated on the notion that tasks can serve as mutual sources of inductive bias for each other. Inductive bias is anything that causes an inductive learner to prefer some hypotheses over others. MTL is one particular kind of inductive bias. MTL uses the information contained in the training signal of related tasks to bias the learner to hypotheses that benefit multiple tasks. One does not usually think of training data as a bias; but when the training data contains the teaching signal for more than one task, it is easy to see that, from the point of view of any one task, the other tasks may serve as bias. For this multitask bias to exist, the inductive learner must also be biased (typically via a simplicity bias) to prefer hypotheses that have utility across multiple tasks.

Inductive biases can be categorized by their strength, explicitness, and domain specificity. The strength of the multitask bias depends on how the tasks are related and how strongly the inductive learner is biased towards sharing learned structure across the tasks. The multitask bias is explicit in that the information provided in the teaching signal is explicit. It can also be implicit, however, because the simplicity bias it depends on is, in many inductive systems, only implicitly specified. The multitask bias is domain specific if the tasks are drawn from one domain. It is a *knowledge-based* inductive bias because the training signal for the tasks can be knowledge rich. That is, from the point of view of any one task, the information contained in the training signal for the other tasks can provide additional domain knowledge to the learner that it would not have received if it were working on one task at a time.

Bias-free learning is impossible; much of the power of an inductive learner follows directly from its inductive bias (Mitchell 1980). Most inductive systems rely on weak biases because there exist weak biases with broad applicability that are easy to use on new problems (e.g., syntactic simplicity). Strong inductive biases are usually domain specific. Unfortunately, domain-specific inductive biases are often costly, typically requiring a domain theory manually encoded by a human expert. MTL may provide an effective source of domain-specific, knowledge-rich, inductive bias that requires less effort from the domain expert.

3 AN EXAMPLE OF MULTITASK CONNECTIONIST LEARNING

The simplest way to introduce multitask connectionist learning is with an example. Consider the following four boolean functions defined on 8 bit inputs:

```
Task 1  =  B1   OR   Parity(B2-B8)
Task 2  =  B1   OR   Parity(B2-B8)
Task 3  =  B1  AND   Parity(B2-B8)
Task 4  =  B1  AND   Parity(B2-B8)
```

where Bi represents the ith bit (leftmost bit is B1), and Parity(B2–B8) is the parity (number of bits set, mod 2) of bits 2–8

For example, if the input pattern is 10001101 then Task1(10001101) = 1 OR Parity(0001101) = 1 OR 1 = 1. Similarly Task4(10001101) = 0.

These four tasks are related in several ways. First, all the tasks are defined on the same inputs: eight binary bits. Second, each task is defined using a common computed subfeature: the parity of bits 2 through 8. Third, on those inputs where Task 1 must compute the parity of bits 2 through 8, Task 2 does not need to compute parity, and vice versa. That is, if B1 = 0, then Task 1 = Parity(B2–B8) but Task 2 = 1 independent of the value of Parity(B2–B8). Task 3 and Task 4 are related similarly: Task 3 needs Parity(B2–B8) when B1 = 1, but Task 4 does not, etc.

Backpropagation (Rumelhart et al. 1986) can be used to train artificial neural networks to learn these tasks. Figure 1 shows the networks one would typically use. Since the tasks are independent, a separate network is used for each task. This is single task learning (STL). But it is also possible, as shown in Figure 2, to use a single network to learn the four tasks at the same time. This is multitask learning (MTL).

The four graphs in Figure 3 show the generalization performance for single and multitask networks trained on Tasks 1–4. Each graph contains three generalization performance curves, one for that task trained in isolation, one for that task trained paired with another task (i.e., MTL with two tasks), and one for that task when all four tasks are trained together (i.e., MTL with four tasks). To simplify the presentation of the two-task multitask data, we present curves only for Tasks 1 and 2 trained together and for Tasks 3 and 4 trained together.

All runs use fully connected feed-forward networks with 8 input units, 160 hidden units, and 1–4 output units. (Networks with fewer than 20 hidden units are adequate for these

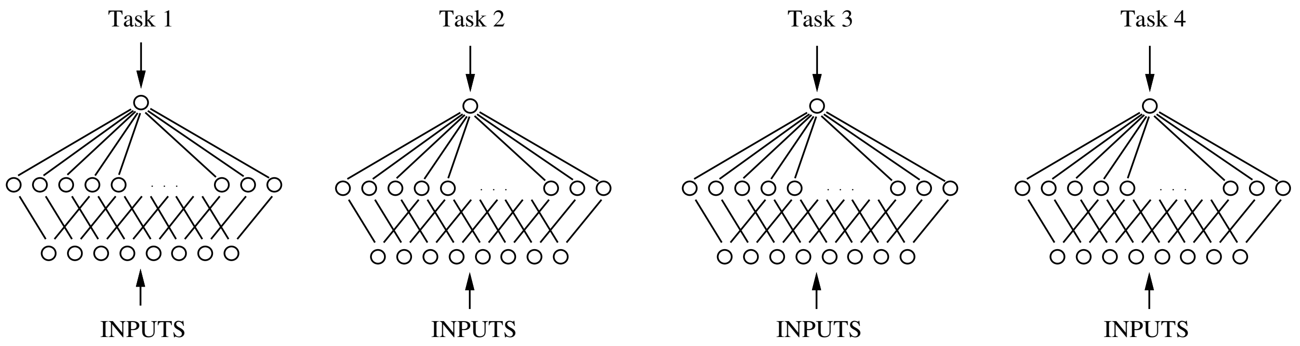

Figure 1: Connectionist Single Task Learning (STL) of Four Functions Defined on the Same Inputs

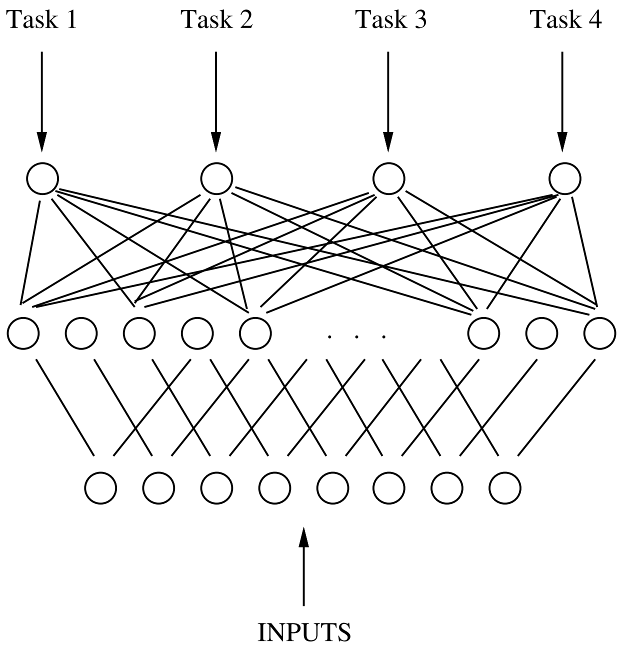

Figure 2: Connectionist Multitask Learning (MTL) of Four Related Functions Defined on the Same Inputs

tasks. The networks are large to minimize capacity effects when different *numbers* of tasks are trained on one network. Other experiments we have run suggest that the generalization performance on these tasks is not injured by the excess capacity.) All networks were trained with backpropagation using MITRE Corporation's Aspirin/MIGRAINES 5.0 with learning rate = 0.1, momentum = 0.9, and per-pattern updating. The training sets contain 64 patterns selected randomly from the 256 possible patterns. The remaining 192 patterns are used as a cross-validation hold-out set to measure generalization performance. The training set was kept small to make generalization challenging, to prevent a training set from including most of the 128 possible examples of the parity subfunction, and to insure that the hold-out set is large enough to yield reliable generalization measures. Each plot is the average of 20 runs.

Let's carefully examine the graphs in Figure 3. The first graph shows the average generalization performance during training for Task 1. The three curves represent the performance for Task 1 when trained alone, when trained with Task 2, and when trained with Tasks 2, 3, and 4. The poorest performance occurs when Task 1 is trained by itself. This is single task neural network learning. Better generalization performance on Task 1 is achieved when a single network is trained on both Tasks 1 and 2 at the same time. And even better generalization performance is achieved by training all four tasks together on a single network. Similar results are found for the other tasks: for each task, better performance is achieved if that task is trained on a network that is trained simultaneously on the other tasks. Note also that better generalization performance is obtained without the need for additional training passes. (In fact, the MTL run is computationally more efficient than the four STL runs it replaces.) To check the results presented graphically in Figure 3, an analysis of variance (ANOVA) of the peak generalization performances from each of the 20 training runs was also performed. This analysis confirms that MTL statistically outperforms STL at the .05 level.

Why is each task learned better if it is trained in the context of a network that is learning the other related tasks at the same time? We have run experiments to verify that the improved generalization performance of MTL is the result of multitask inductive bias. Specifically, we have done experiments to rule out two effects that do not depend on the tasks being related. The first effect is that adding noise to neural net learning sometimes improves generalization performance (Holmstrom & Koistinen 1992). To the extent that the tasks are uncorrelated, their contribution to the aggregate gradient (the gradient that sums the error fed back from each layer's outputs) will appear as noise for other tasks. Thus the uncorrelatedness of the tasks might improve generalization performance by acting as a source of noise. To test for this effect we trained nets on the tasks using random additional tasks. For example, we trained a four output net on Task 1 and three different random tasks. (Each random task was was held constant during training—the random tasks are true functions.)

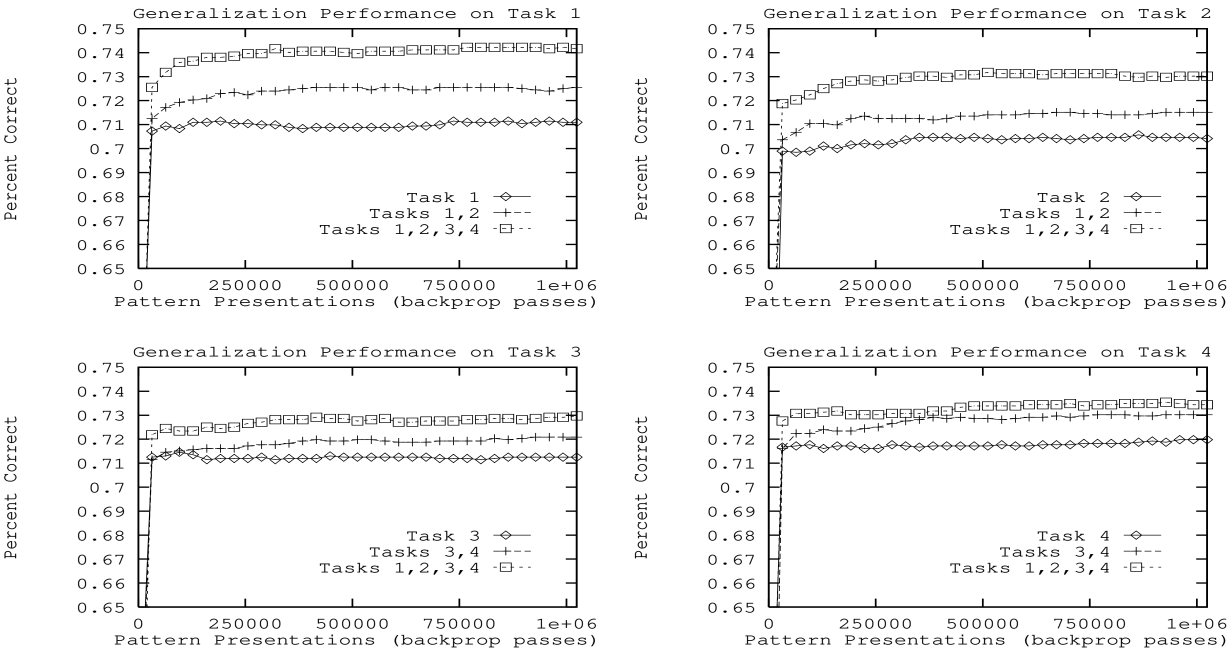

Figure 3: Generalization Performance of Single Task Learning and Multitask Learning on Tasks 1–4

The second effect we tested for is the possibility that adding tasks merely changes the weight updating dynamics to somehow favor nets with more tasks. To test for this we trained nets on four "replicated" tasks at one time. For example, a single net was trained on four copies of Task 1; each of its four outputs received exactly the same training signal. In this degenerate form of MTL, each task correlates perfectly with the others and contributes no additional information, thus cannot serve as a source of knowledge-based inductive bias.

Figure 4 shows the generalization performance for Task 1 when it is multitask trained with three random functions. The performance for Task 1 when it is trained alone, with Task 2, and with Tasks 2–4 is shown for comparison. The performance for Task 1 when it is trained on networks simultaneously trained on random functions is similar to the performance for Task 1 trained alone. (When the other tasks are multitasked with random functions their performance actually drops a little.) We conclude that it is unlikely that the MTL effect is due to the uncorrelatedness of the multiple tasks serving as a beneficial source of noise. (To rule out the possibility that net capacity might artificially limit performance when learning random functions, we verified that networks with as few as 20 hidden units are able to train to 100 percent accuracy on one task and three random tasks.)

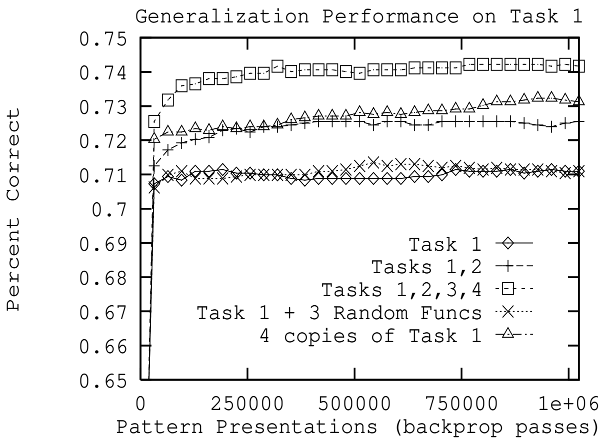

Figure 4: Performance of MTL Variants on Task 1

Figure 4 also shows the performance of nets trained on four copies of Task 1. (The figure shows the performance for one output selected at random.) Surprisingly, performance is better than that for Task 1 trained in isolation. This improvement is gained without any additional information being given to the system. We can think of three explanations: 1) the extra copies may increase the effective learning rate on the weights from the input layer to the hidden layer because their backpropagated error signals add; 2) the four copies may provide a better signal-to-noise ratio early in training when error signals are backpropagated through initially random weights because there are four times as many random weights backpropagating from the output layer to the hidden layer; and 3) having more weights connected to the same signal increases the odds that one of the weights will be close to a "good" value. We have yet to determine what mechanism accounts for the observed performance increase. Fortunately, the replication effect is not large enough to explain the MTL performance increase observed on Tasks 1–4 even if we assume it is somehow able to confer the same magnitude advantage for tasks as different as Tasks 1–4.

One final comment is necessary here. Is the MTL bias effect we observe large enough to be interesting? There are several answers to this. First, performance increased as the number of related tasks increased. In many domains it is surprisingly easy to add more tasks. The accumulated benefit of MTL might be significant in some domains. Second, parity is a hard function to learn, so hard that it is rarely used in generalization studies. (Most studies using parity try to learn parity by training on the complete set of patterns.) The fact that we see improved generalization on tasks that compute parity internally (but are not trained on any explicit information about the parity function itself) is remarkable. Third, the networks have not been "told" that the tasks share a common computable subfeature. Moreover, the tasks differ significantly in how they use this subfeature. This represents a challenging case for MTL because the relationship between the tasks is deeply buried. Yet measurable improvement does occur.

4 MULTITASK CONNECTIONIST LEARNING IN MORE DETAIL

4.1 Advantages and Disadvantages

What are the advantages and disadvantages of using one net to learn several tasks compared with separate nets for each task? Let's begin by examining the possible disadvantages:

1. The MTL net will probably need additional capacity if it is to learn all the tasks.
2. Because the MTL net is doing several potentially competing jobs at the same time, the gradients computed from the error signal for each task may interfere (i.e., cause the aggregate gradient to be flatter or less directional), so learning might be slower for some MTL tasks.
3. The MTL net may begin to overtrain on different tasks at different times. If overtraining is prevented by halting training when the cross-validation generalization performance begins to drop, there may be no one place to stop training the MTL net to achieve peak generalization on all tasks.
4. MTL will sometimes require that new tasks be defined and that training signal for these new tasks be acquired. This places additional burden on the domain expert.
5. If the learning parameters (e.g., learning rate, momentum, initial weight ranges, etc.) must be set differently for different tasks, there may be no one setting appropriate to all the tasks.

The advantages of MTL stem from the possibility that, if the tasks are related, information from the teaching signal of one task might aid learning the other tasks. If the benefit from this mutual inductive bias is strong enough, most of the disadvantages described above can be mitigated:

1. If the tasks can share weights, the MTL net, while bigger than any one net required for the individual tasks, will be smaller than the individual nets combined.
2. Although the MTL aggregate gradient may be flatter, it may be less susceptible to noise. It may also point in better directions because early in search the gradients will point in directions beneficial to several tasks, i.e., towards more generally useful subfeatures.
3. Although tasks will overtrain at different times, this is not really a problem. MTL requires that tasks are trained in the context of other tasks, not that learning produce a single net to use for all tasks. MTL can produce separate nets for each task, each representing the point where generalization performance peaked on that task. (We did this for the ANOVA in Section 3.)
4. Devising new tasks and acquiring the training signal will not always be easy, but it will often be easier than the alternative: acquiring and codifying domain-specific inductive biases acquired from human expertise. We conjecture that MTL provides a convenient "pipe" through which to add domain-specific inductive bias.
5. Where tasks require tweaking the learning parameters, it may be impossible to train the tasks at one time. This may limit MTL's use in some domains.

4.2 How MTL Can Improve Generalization

There are several reasons why multitask nets may generalize better. MTL can provide a data amplification effect, can allow tasks to "eavesdrop" on patterns discovered for other tasks, and can bias the network towards representations that might be overlooked if the tasks are learned separately. Let's examine each of these more closely.

The data amplification effect is best explained using Tasks 1–4 from Section 3. Both Task 1 and 2 need to compute the parity subfeature on half of the patterns, but Task 1 needs parity only on those patterns for which Task 2 does not, and vice versa. Thus the teaching signal for Task 1 provides information about parity half of the time and the teaching signal for Task 2 provides information about parity the other half of the time. A net trained on one of these tasks receives information about the parity subfeature only about half the time, whereas a net trained on both Tasks 1 and 2 gets information about the parity subfeature on every training pattern. Both nets see exactly the same training patterns, but the multitask net is getting more information with each training pattern. This can increase the effective sample size for a computed subfeature. To any one task, the other tasks are providing an inductive bias, i.e., a preference for hypotheses that more accurately compute common subfeatures. (This effect cannot explain all the MTL performance observed on Tasks 1–4 because performance increases if Tasks 1 and 2 are trained with Tasks 3 and 4, yet Tasks 1 and 2 already "require" the parity subfeature for all patterns.)

"Eavesdropping" will be a common benefit of MTL. If tasks are related, it is likely that the computed subfeatures needed by some tasks will sometimes be useful to other tasks. These subfeatures may not be learnable by the tasks that eavesdrop on them if they are trained separately, but might prove valuable sources of information when they are learned for other tasks.

At its best, MTL will bias nets to learn internal representations they would not have learned if trained on the tasks in isolation. Consider a set of 100 tasks for which there are 100 training patterns. Assume each task could benefit by computing a function F of its inputs. Further assume that each task needs F for only one of the 100 training patterns and that each task needs F for a different pattern. Nets trained on the tasks one a time will not learn F because they receive insufficient information to recognize F exists. A net trained on all 100 tasks, however, might learn F. MTL may allow nets to discover internal representations that STL nets could not. MTL may also bias nets towards one of several different internal representations if one representation benefits several tasks.

4.3 Where Multiple Tasks Come From

Tasks 1–4 were devised to be related in carefully chosen ways. Where will the additional tasks MTL requires come from for real-world problems? Often the real world gives us related tasks to learn at the same time. For example, in the Calendar Apprentice System (Dent et al. 1990) inductive learning is used to predict the location, time of day, day of the week, and duration for the meetings it schedules. These tasks are related; they are functions of the same data and probably share some common subfeatures. The Calendar Apprentice currently learns these tasks independently. (Actually, it learns the tasks in a specified order so it can *feed* the output of one task as an input feature to the next.) The MTL approach is to induce a single model for all four tasks at the same time. MTL does not need to define an order for task induction to enable tasks to feed information to each other.

Many real-world domains are "natural" multitask domains if one avoids reductionism. But it is not essential that a domain "naturally" present multiple tasks to use MTL. MTL is a means of adding domain-specific inductive bias to an inductive system via additional teaching signal. For many domains it is surprisingly easy to devise additional related tasks. These might be semi-modular pieces of the larger task (e.g., tasking a robot to predict the trajectory of a ball when the full task requires the robot to catch the ball) or might be tasks that appear to require similar abilities as the given task(s). Acquiring the additional training signal is not trivial, but it is probably easier than asking domain experts to provide other forms of domain-specific inductive bias: domain experts excel at using their knowledge, but usually find codifying their knowledge difficult. Note that MTL does not require that the training signals be available at run time. The training signals are needed only during training because they are outputs—not inputs—of the MTL net. Thus one is free to specify any feature or task that might be useful even if there will be no operational procedure to compute it later.

5 MULTITASK DECISION TREES

MTL can be applied to other learning techniques, even to techniques like decision trees that are not usually used for multiple tasks. Typically, a leaf in a decision tree provides a single class assignment to the instances that reach it. Sometimes leaf nodes refer to multiple classes within a single task, e.g., by assigning probabilities for each class, but separate decision trees are usually built for different tasks. This need not be the case. We can create multitask decision trees containing nodes that assign membership for all the tasks the decision tree is trained on. That is, a single decision tree might have a leaf node that corresponds to class A for Task 1, class C for Task 2, etc.

An example will clarify why one would do this. Suppose we wish to learn several related medical diagnosis tasks. Task 1 is to determine if the patient has retinal degeneracy, Task 2 is to determine if the patient has diabetes, Task 3 is to determine if the patient has reduced peripheral circulation, and Task 4 is to determine if the patient has pulmonary disease. These tasks are strongly related, but the relationship between them is convoluted. Diabetes often causes retinal degeneration and reduced peripheral circulation, and increases the risk of pulmonary disease. Retinal degeneration and reduced peripheral flow are common first symptoms of diabetes. Pulmonary disease, if it is linked to diabetes, is usually a long-term consequence of diabetes, not a diagnostically useful symptom. But any of these conditions can also derive from other dysfunctions and they also often occur in isolation.

The web relating these conditions is tangled. And we can easily imagine a larger web that would bring in dozens of other related symptoms and ailments. The classic decision tree induction approach (Quinlan 1986) is to induce a separate decision tree for each malady. If it is known that some diagnoses might benefit from other diagnoses, we might learn the decision trees in some order and use the outputs from earlier trees as input features to subsequent trees. This is awkward given the number of tasks and the complex web that relates them. The MTL approach is to grow a single decision tree that predicts all the conditions at the same time. To do this we must devise a new greedy splitting rule so that multitask decision trees may be efficiently grown in traditional top-down fashion. One such rule is the average information gain (or entropy decrease) over all the tasks. This will favor attribute splits high in the decision tree that provide discrimination for several tasks. Thus early in its search the decision tree will be biased towards categorizations that have broad utility. As in MTL networks, this bias will cause learning to develop clusters that represent more meaningful subgroups of the domain. In the diabetes example, the MTL decision tree will be more likely to learn the class *brittle diabetics* because this class is useful to all the tasks.

6 RELATED WORK

It is common to train neural nets with many outputs. Usually these outputs encode a single task. For example, in classification tasks it is common to use one output for each class, the output with the highest activation being the classification. But using one net for a few strongly related tasks is also not new. The classic NETtalk (Sejnowski & Rosenberg 1986) application uses one net to learn both phonemes and their stresses. This multitask approach is natural for problems like NETtalk because the goal of NETtalk is to learn to control a synthesizer that needs both phoneme and stress commands at the same time for every input. NETtalk is an example of multitask learning. (Experiments are currently underway to see if STL NETtalk would perform worse than MTL NETtalk.)

Transferring learned structure between related tasks is not new. The main differences between this work and previous work on knowledge transfer are the emphasis here on learning the tasks at the same time and the realization that using many additional tasks might make learning substantially easier. These differences are significant. The work on transferring learned structure between neural nets trained on similar tasks (Pratt et al. 1991)(Pratt 1992)(Sharkey & Sharkey 1992) demonstrates one can use what is learned on one task as a bias for another task. The problem with this is that nets are trained on the tasks sequentially. The operator must devise an appropriate training sequence. One also needs a means of causing the net to retain what it learned before while still enabling it the flexibility to learn anew and modify what was learned before. Because of these difficulties it is unlikely this approach would scale to many tasks. And even if it could, there is no opportunity for a system that trains tasks one at a time to discover representations apparent only when several tasks are viewed together.

This work is most similar to that on providing hints to networks (Abu-Mostafa 1989)(Suddarth & Kergosien 1990)(Suddarth & Holden 1991). The hints approach uses additional tasks that are semi-modular decompositions of the main task. This is definitely MTL. This paper advances the work on hints in four ways: 1) by recognizing that multiple tasks serve as mutual sources of inductive bias; 2) by showing that tasks can be related in more diverse ways than hints yet still yield beneficial bias; 3) by further exploring the ways in which tasks can bias each other; and 4) by suggesting that creating new related tasks may be an efficient way of providing domain-specific inductive bias to learning systems. The empirical results we present demonstrate the first three of these contributions: Tasks 1–4 are not easily viewed as providing hints for each other, yet generalization performance increases when they are trained together.

MTL decision trees are related to some conceptual clustering techniques, most notably COBWEB (Fisher 1987). In fact, a simple modification to the indices in COBWEB's probabilistic information metric yields a metric suitable for judging multitask decision tree splits. COBWEB considers all features as tasks to predict. MTL decision trees allow the user to specify which signals are features and which are training signal, thus making it easier to create additional tasks without committing to extra training information being available later at run time.

7 SUMMARY

Acquiring domain-specific inductive bias is subject to the knowledge acquisition bottleneck. We suggest that domain-specific inductive bias can more easily be acquired in some domains by acquiring teaching signal for additional tasks in that domain. We describe several mechanisms by which this teaching signal can serve as an inductive bias when combined with a bias that prefers hypotheses that benefit several tasks.

We introduce multitask learning (MTL) in connectionism because there is an established methodology for training artificial neural networks that have multiple outputs. In fact, the classic NETtalk application and previous work on supplying hints to neural networks are instances of MTL in connectionism. We provide an empirical demonstration that shows that even when the similarity between multiple tasks is difficult to learn and recognize, MTL can still improve generalization performance. We further demonstrate that this improvement is not easily attributable to effects other than a mutual inductive bias based on the similarity of the tasks.

To show the generality of the MTL methodology, we briefly outline how to do top-down induction of multitask decision trees. This is significant because decision trees are not usually used to learn multiple tasks. Doing this results in a system that generalizes some conceptual clustering techniques so as to make them more useful in domains where the distinction between features (information that will be available in the future) and classes (things that we want to predict) must be retained.

Precisely what conditions are required for MTL to succeed are not yet known. Perhaps it will be sufficient to give multiple tasks to inductive systems and allow their existing simplicity biases to exploit the relatedness. Or perhaps we will have to devise new ways to help the inductive systems recognize common substructure and then "coerce" them to share representations for this substructure. Much work remains to be done. We need more effective ways to "open up" complex networks so that we can see what is inside. In particular, we need to devise methods for detecting when "eavesdropping" is occurring or when a new (or modified) internal representation has been developed.

We need to better understand how tasks should be related for MTL to succeed and how to recognize these traits in real-world tasks. We need to empirically demonstrate that MTL can provide a strong enough inductive bias to be useful in complex tasks. We need to evaluate how hard it will be to build useful additional tasks for domains of interest (e.g., vision). We conjecture that as the complexity of the domain scales, so too will the opportunity for creating additional related tasks. If this is true, MTL may provide a practical method of acquiring domain-specific inductive bias that scales naturally with the complexity of the task.

Acknowledgements

Thanks to Wray Buntine, Lonnie Chrisman, Jeff Jackson, Tom Mitchell, Herb Simon, and Sebastian Thrun for their help in refining the ideas presented here. Special thanks to the reviewers and to Lonnie Chrisman, Rich Goodwin, Tom Mitchell, and Diane Staub for their careful review of drafts of this paper.

This research was sponsored in part by the Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

Abu-Mostafa, Y. S. (1990) Learning From Hints in Neural Networks. *Journal of Complexity* **6**(2):192-198.

Dent, L., Boticario, J., McDermott, J., Mitchell, T. & Zabowski, D. (1992). A Personal Learning Apprentice. *Proceedings of 1992 National Conference on Artificial Intelligence*.

Fisher, D. H.(1987) Conceptual Clustering, Learning From Examples, and Inference. *Proceedings of the 4th International Workshop on Machine Learning*.

Holmstrom, L. & Koistinen, P. (1992) Using Additive Noise in Back-propagation Training. *IEEE Transactions on Neural Networks.* **3**(1):24-38.

Mitchell, T. M. (1980) The Need for Biases in Learning Generalizations. Rutgers University: *CBM-TR-117.*

Quinlan, J. R. (1986) Induction of Decision Trees. *Machine Learning.* **1**:81-106.

Pratt, L. Y., Mostow, J. & Kamm, C. A. (1991) Direct Transfer of Learned Information Among Neural Networks. *Proceedings of AAAI-91.*

Pratt, L. Y. (1992) Non-literal Transfer Among Neural Network Learners. Colorado School of Mines: *MCS-92-04.*

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986) Learning Representations by Back-propagating Errors. *Nature.* **323**:533-536.

Sejnowski, T. J. & Rosenberg, C. R. (1986) NETtalk: A Parallel Network that Learns to Read Aloud. John Hopkins: *JHU/EECS-86/01.*

Sharkey, N. E. & Sharkey, A. J. C. (1992) Adaptive Generalisation and the Transfer of Knowledge. University of Exeter: *R257.*

Suddarth, S. C. & Holden, A. D. C. (1991) Symbolic-neural Systems and the Use of Hints for Developing Complex Systems. *International Journal of Max-Machine Studies* **35**:3:291-311.

Suddarth, S. C. & Kergosien, Y. L. (1990) Rule-injection Hints as a Means of Improving Network Performance and Learning Time. *Proceedings of the 1990 EURASIP Workshop on Neural Networks.* 120-129.

Waibel, A., Sawai, H. & Shikano, K. (1989) Modularity and Scaling in Large Phonemic Neural Networks. *IEEE Transactions on Acoustics, Speech and Signal Processing.* **37**(12):1888-98.