Cross-Domain Knowledge Transfer Using Structured Representations

Samarth Swarup and Sylvian R. Ray
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
swarup@uiuc.edu

Abstract

Previous work in knowledge transfer in machine learning has been restricted to tasks in a single domain. However, evidence from psychology and neuroscience suggests that humans are capable of transferring knowledge across domains. We present here a novel learning method, based on neuroevolution, for transferring knowledge across domains. We use many-layered, sparsely-connected neural networks in order to learn a structural representation of tasks. Then we mine frequent sub-graphs in order to discover sub-networks that are useful for multiple tasks. These sub-networks are then used as primitives for speeding up the learning of subsequent related tasks, which may be in different domains.

Introduction

We are interested in the design of agents that work for an extended period of time (Swarup *et al.* 2005). During this "lifetime", an agent may encounter several, possibly related, learning tasks. It is desirable that the agent should be able to improve its learning performance with each new task encountered, i.e., it should gain some experience about "how to learn" each time it has to learn a new task. Techniques for accomplishing this are called, variously, cumulative learning, lifelong learning, meta-learning, learning to learn etc. (Vilalta & Drissi 2002).

Most methods for knowledge transfer in machine learning assume that all tasks come from the same domain. A domain is an input-output space. There are several reasons for this. Inputs can be thought of as coming from sensors, e.g. in robots, which then have to learn several tasks in a given domain. Thus the input space is fixed, and knowledge transfer is done by, e.g, learning with a neural network in which the first layer is shared by several tasks (Caruana 1997). However, for more higher-level, "cognitive", tasks it is unreasonable to try to learn directly from sensory inputs. Instead learning can be thought to occur in some state space, or feature space, that is extracted from the sensory inputs. It would still be very useful to be able to transfer knowledge across tasks. However, we may now have to deal with input-output spaces of different dimensions. It is relatively difficult, though, to imagine how to transfer information between two neural networks of different input and hidden dimensions.

Evidence from psychology, however, suggests that humans can, and often do, transfer information across domains. For example, Dunbar and Blanchette studied analogies used in political speeches, and analogies used by scientists in labs (Dunbar & Blanchette 2001). In a study conducted by videotaping and audiotaping molecular biologists and immunologists over the course of several lab meetings, they found both within-domain and cross-domain analogies being made. While the majority of analogies (75%) were within-domain and involved similarity of superficial features, cross-domain analogies dominated when the scientists were trying to formulate new hypotheses. 80% of the analogies used in this case were cross-domain and used structural features (i.e. relationships). In an analysis of analogies used by politicians and journalists during a referendum campaign in Quebec in 1995, they found that only 24% were from politics, i.e. over 75% of the analogies were based on structural comparisons with other domains. In contrast, experiments on analogy making conducted in laboratory settings often find that people mostly make analogies based on superficial features. This is not surprising perhaps, because these experiments are generally timed, and structure mapping is a more computationally intensive procedure. The point is that people transfer knowledge across domains quite naturally.

Support for this idea also comes from the study of mirror neurons (Rizzolatti *et al.* 1996). Mirror neurons are neurons that are active both when an action is performed and when it is observed. They have also been implicated in empathy, the evolution of language, and multi-modal learning.

These studies suggest that knowledge transfer in machine learning should be extended from dealing with multiple tasks in a single domain to tasks in multiple domains. We present here a learning method for achieving this goal. The key idea is to use a structured representation, which allows extraction of domain-independent knowledge. Note that we are not specifically attempting to model data from psychology or neuroscience, nor are we making any claim about how knowledge is transferred across domains in humans. Our technique also avoids making explicit analogies. It is possible to transfer knowledge across domains effectively

based on just the sub-task decomposition achieved by using a structured representation.

The rest of this article is organized as follows. In the next section we describe some prior attempts at transferring knowledge across domains. Then we present our own representation and learning algorithm. This is followed by a set of experiments that demonstrate the benefit of knowledge transfer across domains in a set of Boolean-function learning tasks. Finally, we discuss some application areas and directions for future work.

Related Work

There has been very little work in cross-domain knowledge transfer. Previous attempts have generally approached the problem by using two representations, one that is problem-specific and another that is problem-independent, such as neural networks and symbolic rules. However, intuitively it seems better to avoid transformations of representation because there is inevitably some loss of precision due to the transformation.

(Madden & Howley 2004) studied cross-domain knowledge transfer in a reinforcement learning setting, using a technique they call *Progressive Reinforcement Learning*. In their system, an agent has to solve a series of mazes of increasing difficulty. The agent, called Theseus, has to reach the exit while avoiding another agent, called the Minotaur, which follows a fixed policy (which is unknown to Theseus). Theseus engages alternately in *experimentation*, i.e. learning a policy for a particular maze, and *introspection*, where it uses a symbolic learner to extract rules that might be beneficial in solving the next maze. These rules are used when a novel state is encountered while solving the new maze. The high-level features used in the rules were defined by the experimenters, such as *distance to Minotaur*, *direction to Exit*, etc. They reported significant speed-ups, and increases in reward over standard Q-learning. The main limitation of their system is that the high-level features may not be the appropriate descriptors for knowledge transfer. However, the advantage of using symbolic rules is that it is easy for the user to add domain knowledge.

The idea of extracting symbolic rules from a learned controller has also been studied by others. (Shavlik 1994), e.g., extracted rules from neural networks. (Omlin & Giles 1996) extracted rules in the form of a regular grammar from recurrent neural networks.

These strategies all suffer from the fact the there is inevitably some noise introduced by the transformation of representation. The second representation may also not have the same representational power. Recurrent neural networks, for example, are capable of representing more complex grammars than regular ones (Bodén & Wiles 2001).

In contrast, using a structured representation allows us to transfer parts of the learned solution to the new task, and avoids the problem of transforming the representation and possibly losing information.

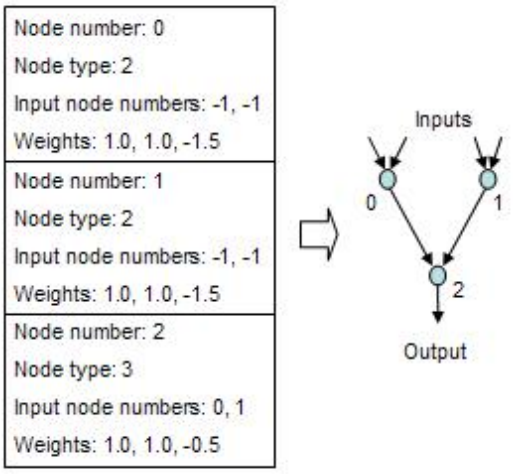

Figure 1: An example genome and the corresponding network. Each node computes the weighted sum of its inputs (the weights include a bias weight), and then squashes it using a logistic function. Inputs designated with -1 are external inputs. The highest numbered node is assumed to be the output node.

Representation, Learning, and Knowledge Transfer

Representation: To capture the notion of the structure of a task, we use many-layered, sparsely-connected neural networks (Utgoff & Stracuzzi 2002). These are constructed out of a set of *primitives*, which are parts of networks (i.e. subnetworks or subgraphs). To begin with, the set of primitives is small and contains just single nodes. In our implementation, each node of a network computes a sigmoid function of the weighted sum of its inputs, but this does not have to be the only case. A node (or a primitive) might compute any function of its inputs. The weights are fixed, and are not updated during learning. A network is a composition of primitives into a directed acyclic graph. There is no reason recurrent connections should not be allowed, but they have been disallowed here for simplicity.

A network is also represented by a *genome*, as shown in figure 1. A genome, unlike a network, is linear. Each *gene* in the genome is a structure that corresponds to a node in the network. It describes the type of the node (i.e. the function it computes), the position of the node in the network, and which nodes provide input to this node. Some nodes get external inputs; these are simply denoted by a -1 for the input number in the corresponding gene. One node is designated as the output node. For simplicity, again, we are only considering tasks with a single output.

A genome (or network) does not correspond uniquely to a function. There are several different ways of computing the same function. However, all such networks may not be equally useful from the point of view of transfer of knowledge. Figure 2, for example, shows two different ways to compute 3-bit parity using AND, OR, and NOT functions at

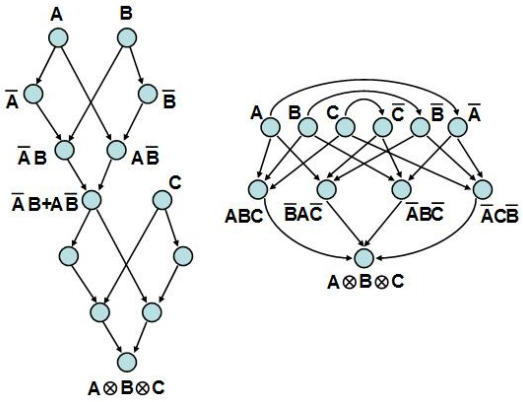

Figure 2: Two networks that each compute 3-bit parity. The one on the left has a repeated structure (corresponding to 2-bit parity) and can therefore can be generated by reusing the solution to the 2-bit parity problem, if it is known beforehand.

the nodes. The network on the left in the figure is more conducive to knowledge transfer, because it has a substructure which is repeated. This subnetwork computes 2-bit parity, i.e. the XOR function. If this network is known beforehand (if it has been learned beforehand, say), then it can be reused to generate a solution to the 3-bit parity problem quickly.

This is the basic idea in this work: in general, whatever the representation, there are multiple ways to solve a problem. The key is discover a set of similar solutions, so that the similarity information can be used to generate solutions to new similar problems quickly.

Learning and Knowledge Transfer: Learning is done by an evolutionary algorithm that constructs a population of networks from the primitives. Each network in the population is evaluated on the training set and assigned a fitness corresponding to its accuracy on the training set. A subset of the population is chosen to create the next generation via mutation and crossover. There can be three kinds of mutations: *adding*, *deleting*, and *replacing* mutations. A deleting mutation simply removes a gene from the genome of a network. An adding mutation adds a *primitive* to the genome. Remember, a primitive can be a sub-network, or even an entire network. Similarly, a replacing mutation replaces a gene with a primitive. A new network is created from two parents of high fitness by a one-point crossover. This is followed by a mutation with some small probability. Some of the high fitness networks are carried over unchanged to the next generation, and the process is repeated until some accuracy or time criterion is met.

The set of primitives is augmented by running a graph mining algorithm on the solutions to the set of tasks that have been solved, to discover frequent sub-networks. These represent partial solutions and are common to several tasks. This allows transfer of knowledge across tasks in different domains because these sub-networks are not bound to a particular input space.

Experiments

We did experiments with a set of Boolean functions where the output is 1 if there are k *adjacent* 1's in the input. For example, in one set of tasks, the output is 1 if there are 2 adjacent 1's anywhere in the input vector. The initial set of primitives were nodes that compute the AND, OR, and NOT functions, as well as an INPUT node that just copies its input unchanged to its output.

The tasks were defined over four domains, i.e. input spaces of four different dimensions: 4, 8, 12, and 16 inputs. Ten tasks were learned over the four domains, corresponding to values of k from 2 to 5, and the CloseGraph graph mining algorithm (Yan & Han 2003) was used to extract a set of common sub-networks from all the previous networks after each new task network was learned. Figure 3 shows an intermediate stage in one run through the set of tasks. Figure 4 shows all the learning curves. The 4-inputs, 2-adj-ones task was learned first, though the learning curve for that task is not shown (since there is no knowledge transfer in that case). The number of examples in the training set is mentioned on the y-axis in each case (200 for 8-input tasks, and 1000 for others). The tasks were learned in left to right, top to bottom order (note that fig. 4 is rotated, so the top of the figure is on the right of the page). We see that the error with transfer of knowledge drops far more quickly in each case than the error without transfer. Also, the learning curve with transfer shows much less variance (the error bars in the figure show one standard deviation). Thus learning is much more *robust* with transfer. Since these domains are quite small, we could also generate all the examples and evaluate the true error of the learned classifier. This is shown in figure 5. We see that the true error is always significantly lower with transfer, and in some cases results in a very large benefit. Again, the error bars (which show 95% confidence intervals) are much smaller with transfer of knowledge. This is another example of the robustness due to transfer of knowledge.

We call this approach *cumulative learning* because the set of sub-networks, which represents a store of abstract or common knowledge, grows in a cumulative way over the course of learning (i.e. over an agent's lifetime).

Discussion

Another major advantage of cumulative learning is that when we learn by transferring knowledge from previous tasks, not only do we learn faster, we also discover solutions that are similar to the ones we already know (for other tasks). There are generally several ways of solving a particular task. The advantage of finding similar solutions is that we then discover and reinforce[1] the patterns that are useful for multiple tasks. This means that in the future we will use these patterns more in solving related tasks, and thus discover solutions that again reinforce these patterns. This is akin to a frequency-dependent effect, and is called the *cumulative advantage* (de Solla Price 1976).

[1]The word *reinforce* is not being used in a technical sense here.

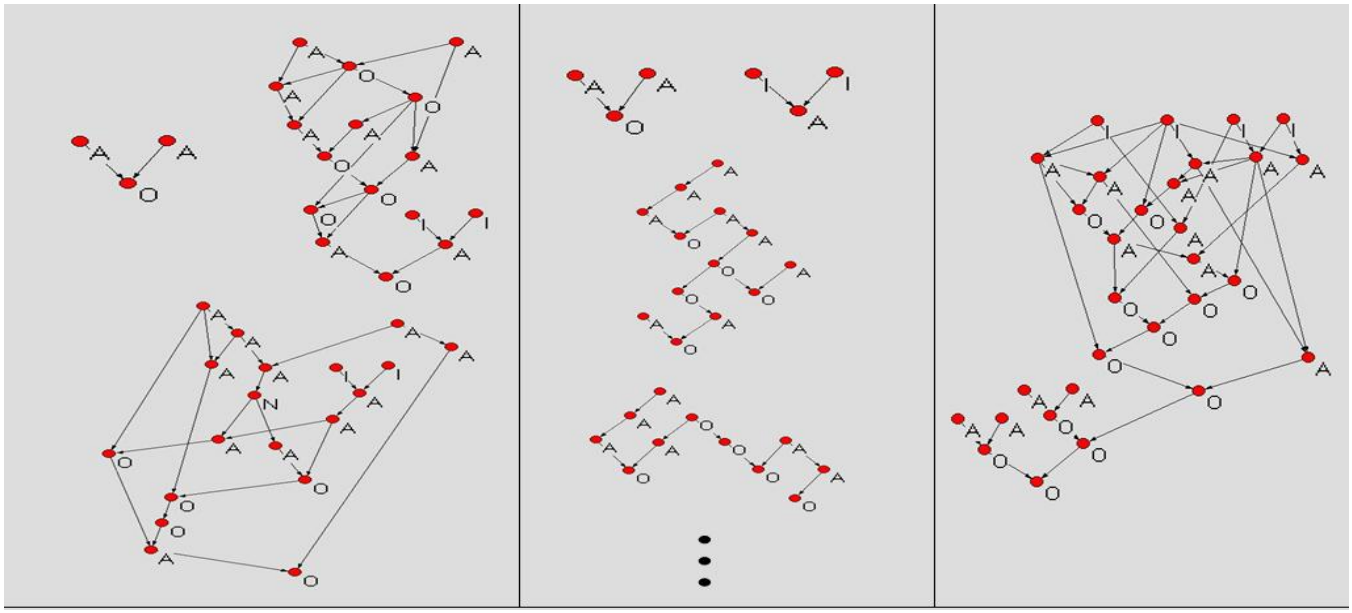

Figure 3: The left box shows networks learned on the first three tasks (4inputs-2adjones, 8inputs-2adjones, and 8inputs-3adjones). The middle box shows 4 of the 18 subnetworks extracted by CloseGraph, which appear in at least two of the networks on the left. The right box shows the network learned for the 12inputs-2adjones task, in which some of the subnetworks are seen to appear several times.

Overfitting

As primitives get larger, the networks that are generated by the evolutionary algorithm get larger as well. This leads to overfitting. There are several ways to control for this. One possibility is to include a cost for the size of the network in the fitness function, thus penalizing larger candidate networks. Another way is to include a preference for smaller sized primitives during mutations. Yet another possibility is to choose primitives based on their frequency of occurrence in the previously learned tasks. This is similar to having a preference for smaller-sized primitives because these tend to have higher support. We took a simpler approach, and made the probability of a deleting mutation much larger than the probabilities of adding or replacing mutations, which turns out to work very well in practice. The problem of overfitting in general is related to Ockham's razor, which suggests that simpler solutions that fit the data are to be preferred. This line of thought suggests a route to a theoretical analysis of cross-domain transfer via the algorithmic complexity framework. In this framework, there is the idea of a universal prior (Solomonoff 1964) which gives higher probability to smaller hypotheses (represented as strings). There have been several attempts to contruct computable versions of the universal prior (Schmidhuber 2002), typically by adding in the time to compute the hypothesis. When we are learning a set of related tasks (and not looking at the entire universe of possible tasks), it should be possible to do better than the universal prior. Intuitively, this is exactly what our technique attempts to do by extracting frequently used sub-networks (or substrings if we consider the genomes instead of the networks) which effectively tell us which of the small networks are very unlikely to appear as solutions in our set of related tasks. This is a promising avenue for further research.

Task Relatedness

One of the most important questions facing the study of inductive transfer is how to define and measure the relatedness of two tasks. Our technique offers a potentially well-founded measure of the relatedness of two tasks. We can construct a metric based on the size of the largest common sub-network. This is well-founded in the following sense. The informational distance between two strings can be seen as the difference between the sum of the minimum description length of the two strings separately, and the minimum description length of the two strings taken together (by appending one to the other). The largest common subgraph (or the largest common substring of the two genomes) is related to the minimum description length of the two networks taken together. Thus one possible measure of the distance between two networks (i.e. two tasks), is

distance(network1, network2) = sizeof(network1) + sizeof(network2) - sizeof(largest common subgraph).

This should be normalized by dividing by the sum of the sizes of the two networks. Once we have pairwise distances between a set of learned tasks, we can cluster the previously learned tasks and do selective transfer of knowledge by comparing networks generated relatively early in the search to prototypes of clusters, and then only using primitives from one cluster to guide the generation of new candidate networks.

The importance of starting small

Since there are many possible networks for a given problem, an important issue is, how do we ensure that problems that are solved early will result in networks with reusable subparts? In particular, networks which are very good from the point of view of knowledge transfer might be suboptimal in terms of error on the particular task. One answer is to begin by solving small problems, where very small networks (having few small variants) solve the problem. These networks then form the primitives for solving larger related problems. In our experiments, we solve tasks in increasing order of size of the input space. The idea is that for small input spaces, the set of networks of comparable size that have high fitness is small, and so we are more likely to learn a network which has reusable subparts. However a more principled solution to this problem is also a promising area of future research.

Conclusions

We believe that cross-domain knowledge transfer is a very important direction for extending multi-task learning. It is one of the key steps towards building sophisticated intelligent agents that can exist for a lifetime. There has been very little previous work, however, that looks at this problem.

We have presented a technique for transferring knowledge across domains which avoids the problem of transformation of representation and the consequent possibility of loss of knowledge. The key idea is to use a structured representation which allows transferring parts of solutions.

The main directions for future work are to do a theoretical analysis in the algorithmic complexity framework, including an analysis of task relatedness based on informational distance between networks, and to explicate the technique for clustering tasks based on informational distance and selective transfer.

Another direction for future work is to try this approach in a real-world situation. The idea of primitives carries over easily to robotics and motion planning where the primitives are a set of basic movements or abilities, and complex actions are carried out by composition of primitives (Thoroughman & Shadmehr 2000). Our cumulative learning approach could be used to coordinate multiple limbs and learn complex actions quickly.

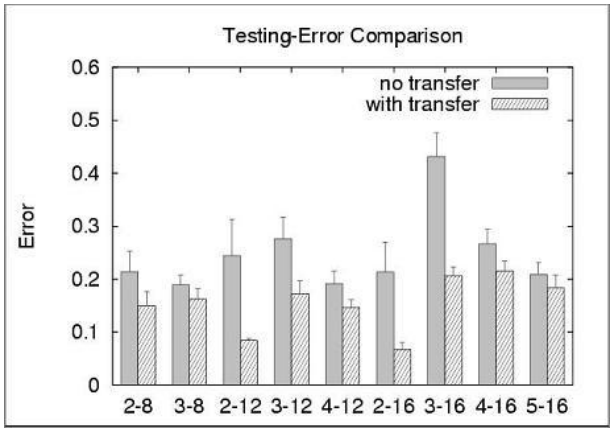

Figure 5: A comparison of the error on the complete data set for each task, with and without transfer of knowledge. The errorbars show 95% confidence intervals.

References

Bodén, M., and Wiles, J. 2001. Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science* 12(3,4):197–210.

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

de Solla Price, D. 1976. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American Society for Information Science* 27:292–306.

Dunbar, K., and Blanchette, I. 2001. The inVivo/inVitro approach to cognition: The case of analogy. *Trends in Cognitive Sciences* 5:334–339.

Madden, M. G., and Howley, T. 2004. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review* 21:375–398.

Omlin, C. W., and Giles, C. L. 1996. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks* 9(1):41–52.

Rizzolatti, G.; Fadiga, L.; Matelli, M.; Bettinardi, V.; Paulescu, E.; Perani, D.; and Fazio, G. 1996. Localization of grasp representations in humans by positron emission tomography: Observation versus execution. *Experimental Brain Research* 111:246–252.

Schmidhuber, J. 2002. The speed prior: A new simplicity measure yielding near optimal computable predictions. In *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT)*.

Shavlik, J. 1994. Combining symbolic and neural learning. *Machine Learning* 14(3):321–331.

Solomonoff, R. J. 1964. A formal theory of inductive inference, parts 1 and 2. *Information and Control* 7:1–22, 224–254.

Swarup, S.; Mahmud, M. M. H.; Lakkaraju, K.; and Ray, S. R. 2005. Cumulative learning: Towards designing cognitive architectures for artificial agents that have a lifetime. Technical Report UIUCDCS-R-2005-2514.

Thoroughman, K. A., and Shadmehr, R. 2000. Learning of action through adaptive combination of motor primitives. *Nature* 407:742–747.

Utgoff, P. E., and Stracuzzi, D. J. 2002. Many-layered learning. *Neural Computation* 14(10).

Vilalta, R., and Drissi, Y. 2002. A perspective view and survey of meta-learning. *Artificial Intelligence Review* 18(2):77–95.

Yan, X., and Han, J. 2003. CloseGraph: Mining closed frequent graph patterns. In *Proceedings of the 9th ACM SIGKDD conference on Knowledge-Discovery and Data Mining (KDD 2003)*.

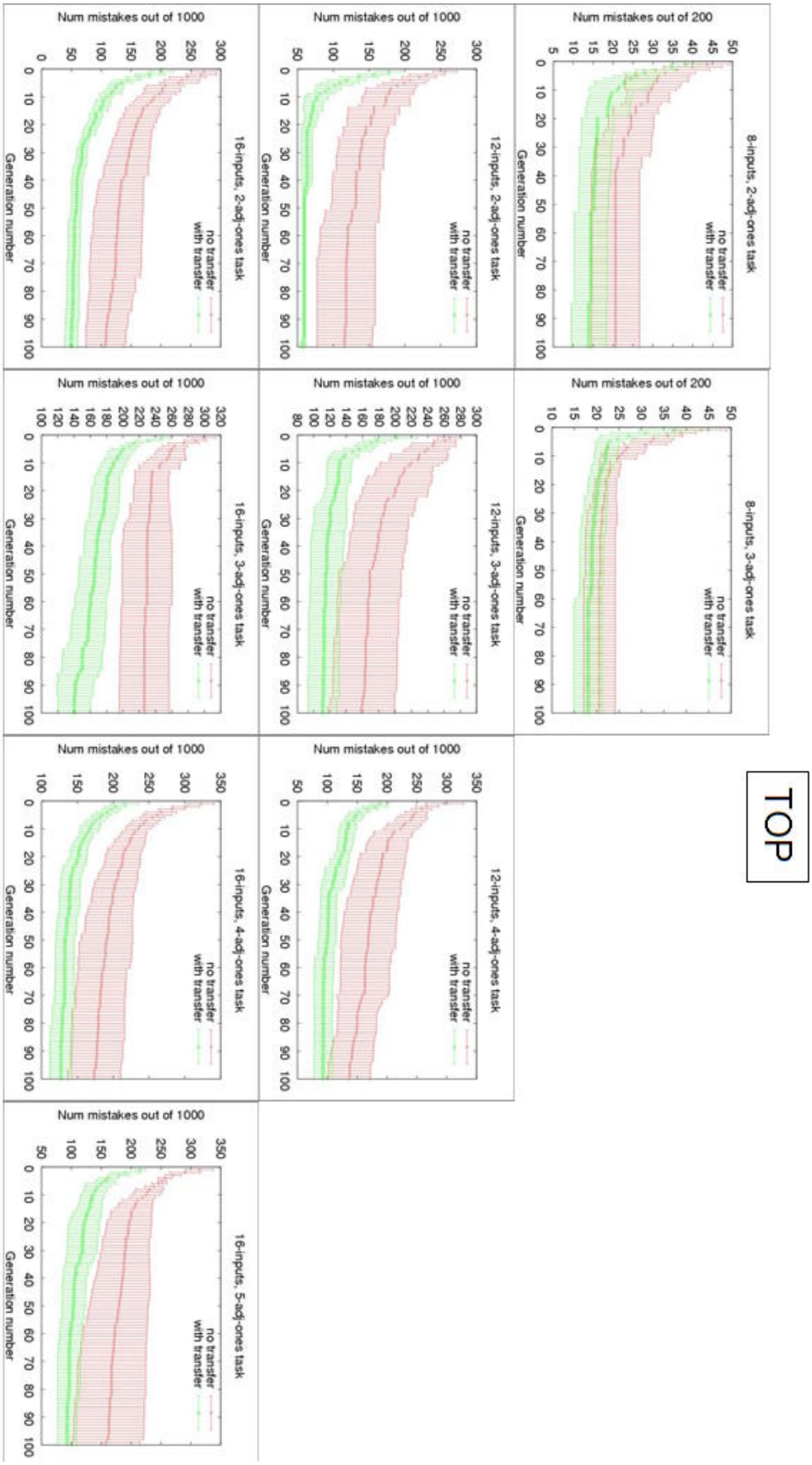

Figure 4: Learning curves with and without transfer of information, for each task. Tasks were learned in left to right, top to bottom order, i.e. 8-inputs-2-adj-ones, 8-inputs-3-adj-ones, 12-inputs-2-adj-ones, etc. Note that the figure is rotated, so the top of the figure is to the right of the page.