Previous work on knowledge transfer in machine learning has been mostly 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 structural representations of tasks. Then we mine frequent sub-graphs in order to discover sub-networks that are useful for multiple tasks. These sub-networks are used as primitives for generating candidate solutions to subsequent related tasks, which may be in different domains. We observe improvements in speed of convergence, testing error, and variance.

Abstract

To capture the notion of the structure of a task, we use many-layered, sparsely-connected neural networks. These are constructed out of a set of primitives, which are parts of networks (i.e. sub-networks or sub-graphs). To begin with, the set of primitives is small and contains just single nodes. Each node computes a fixed function of its inputs. A network is also represented by a genome, as shown in the figure on the right. Learning is done with a genetic algorithm, which creates networks from the set of primitives, and modifies them by adding, deleting, and replacing mutations. Networks can also be combined using a one-point crossover.

Representation and Learning

A genome (or network) does not correspond uniquely to a function. The figure shows two different ways to compute 3-bit parity using AND, OR, and NOT functions at the nodes. The network on the left in the figure has a substructure which is repeated. This sub-network 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.

Basic Idea

We use the CloseGraph graph mining algorithm to extract a set of frequent sub-networks from the set of learned networks. These sub-networks are added to the set of initial primitives and are used to generate candidate networks for the next task.

Extracting Knowledge

We did experiments in a set of Boolean function domains, where in each task, the output is 1 if there are at least k adjacent 1’s in the input. The left box in the figure shows the networks learned on the first three tasks: 4inputs-2adjOnes, 8inputs-2adjOnes, and 8inputs-3adjOnes. The middle box shows some of the frequent sub-graphs extracted by CloseGraph, and the right box shows the network learned for the 12inputs-2adjOnes task using this augmented set of primitives.

Learning Curves

The figures show learning curves on nine tasks. The learning curve for the first task (4inputs, 2 adj-ones) is not shown because there is no transfer in that case. Tasks were learned in left to right, top to bottom order. We see that there is always an improvement in the speed of convergence and in the standard deviation, with transfer of knowledge from previous tasks.

Results and Discussion

The figure shows the improvement in testing error with transfer of knowledge. The error bars show 95% confidence intervals.

A benefit of cumulative learning is that when we learn by transferring knowledge from previous tasks we discover solutions that are similar to the ones we already know (for other 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 the cumulative advantage.

Inductive Transfer: 10 Years Later, NIPS 2005