Dynamic Nets
Cascade-Correlation

<table>
<thead>
<tr>
<th>Aims</th>
<th>• to consider the concept of dynamic nets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• to look briefly at the cascade-correlation architecture/algorithm</td>
</tr>
</tbody>
</table>


| Keywords     | cascade, correlation |

<table>
<thead>
<tr>
<th>Plan</th>
<th>• dynamic nets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• problems with backprop</td>
</tr>
<tr>
<td></td>
<td>• cascade-correlation</td>
</tr>
</tbody>
</table>

Introduction

• Standard feedforward networks are static, in the sense that the number of units is fixed, along with the locations of connections between units (though not the weight values)

• As biological brains grow, at least, new neurons do get added (and axons grow and create synapses with the dendrites of other neurons). New synaptic connections grow even in adult human brains. Vast numbers of neurons and connections also disappear in infancy, as part of normal development.

• In biological brains, there is a range of different neuron types (pyramidal cells, bushy cells, purkinje cells, etc.), not just a single model of neuron

• There are different types of synapse with different neurotransmitters (some excitatory, like serotonin and acetylcholine, some inhibitory, like gamma amino butyric acid (GABA)). The excitatory/inhibitory distinction is captured in artificial neural networks by having both positive and negative weights.

• So fixed network topologies using homogeneous neurons are not biologically reasonable.

Dynamic Networks

• A dynamic network is one in which the number of units changes as time goes on

• In a dynamic net, when a new unit is added, it must be connected to existing units, and the values of the weights determined in some manner

• Weight values involving a new unit might be determined iteratively, and input connections might be trained in a different way to output connections

• New units and their weights might be trained in different manners.

• A single new unit could be trained in several different ways (in parallel) and the way that gave the greatest reduction in error could determine the “type” of neuron that it became.

Problems with Backprop-trained Networks

The Step-Size Problem

• Backprop is only guaranteed to find a minimum if step size is infinitesimal (in which case training time is infinite). In practice, reasonable learning rates work out OK, but ...

• Various techniques have been used to enhance the speed of backprop - e.g. momentum, conjugate gradient methods, dynamic learning rates, and use of approximations to the second derivatives of the error function.

The Moving Target Problem

• “Each unit in the interior of the network is trying to evolve into a feature detector that will play some useful role in the network’s overall computation, but its task is greatly complicated by the fact that all the other units are changing at the same time. The hidden units in a given layer of the net cannot communicate with one another directly; each unit sees only its inputs and the error signal propagated back to it from the network’s outputs. The error signal defines the problem that the unit is trying to solve, but this problem changes constantly. Instead of a situation in which each unit moves quickly and directly to assume some useful role, we see a complex dance among all the units that takes a long time to settle down.” (Fahlman & Lebiere, 1990)
The Herd Effect

- Several neurons may “set out” to solve the same computational sub-task as part of a learning problem, instead of each one tackling a different aspect of the problem.
- When the sub-task is solved, something else must be done to reduce error, and so all the neurons may modify their weights so as to make progress on the next goal sub-task, which is likely to reduce performance on the original sub-task.
- Eventually the neurons will split up and specialise on a sub-task of their own, but meantime learning is slowed.
- (This is the reason for the random initialisation of weights in backprop - with uniform initialisation, all the neurons would head off in the same direction as learning began).

Solving the Herd Effect Problem

- Just modify one neuron at a time! (Or maybe just a few at a time)

Training Method

- Begin with a candidate unit that has trainable input connections from all input units and all previous hidden units. Output is not yet connected.
- Train so as to maximise $S$, the sum over all output units $o$ of the magnitude of the covariance between $V$, the candidate unit’s value, and $E_o$, the residual output error at unit $o$:

$$S = \sum_{o,\text{outputs}} \sum_{p,\text{patterns}} (V_p - V_{\text{avg}})(E_{p,o} - E_{o,\text{avg}})$$

- Maximisation is done by a gradient descent method. When $S$ stops improving, install the candidate as a unit, freeze its input weights, train its output weights, and then if necessary recruit another hidden unit.

Cascade-Correlation

- Initially, there are just input units and output units
- These are trained (as a two layer net) by, e.g. backprop, though as no backpropagation is needed in a two-layer net, it doesn’t have to be backprop
- When error no longer reduces, a hidden unit is added as follows:

Training Method (continued)

Variations

- Instead of a single candidate unit, use a pool of candidate units, trained in parallel
- Random variations between the initialisation of the pool members means that you might get a better solution – pick the best, install it in the net, freeze its weights, and discard the pool nodes.
- Different pool members could be trained to be of a different nature - e.g. some could be standard backprop units with a sigmoid activation function, some could have Gaussian activation functions, some with radial activation functions, ...
Notes

- when output layer weights are being trained, other weights in the network are frozen
- while candidate units are being trained, the other weights in the network are frozen
- there is a recurrent version of the Cascade Correlation algorithm, too, described in Fahlman’s paper in Advances in Neural Information Processing Systems 3, and available locally at http://www.cse.unsw.edu.au/~billw/cs9444/rec-cascor-fahlman-91.pdf

The “two-spirals” Problem

This problem was attempted and solved with Cascade Correlation – it is an extremely hard problem for algorithms of the back-propagation family to solve. The net has two continuous-valued inputs and a single output. The training set consists of 194 X·Y values, half of which are to produce a +1 output and half a –1 output, as shown in Figure 2a.

![Figure 2a: Training points for the two-spirals problem, and output pattern for one network trained with Cascade-Correlation.](image)

Straight backprop had never solved this: variants had solved it in 12,000 epochs (8,000 with Quickprop). Cascade Correlation did it in 1700 epochs on average.

Evolution of the Solution

![Figure 3 from Fahlman & Lebiere 1991](image)

Evolution of the Solution (continued)

![Figure 4 from Fahlman & Lebiere 1991](image)
<table>
<thead>
<tr>
<th>Scott Fahlman</th>
<th>Christian Lebiere</th>
</tr>
</thead>
<tbody>
<tr>
<td>• found QuickProp &amp; Cascade Correlation</td>
<td>• found Cascade Correlation</td>
</tr>
<tr>
<td>• also credited with inventing the first emoticons, :) and :-( in an email dated 19 Sep 1982</td>
<td>• interest is cognitive architectures and their applications to psychology, artificial intelligence, economics, decision theory and human-computer interaction</td>
</tr>
<tr>
<td>• works at Carnegie-Mellon University</td>
<td>• works at Carnegie-Mellon University</td>
</tr>
</tbody>
</table>