Model Selection and Evolutionary Optimization

Neural Computation : Lecture 20

1. Model Selection
2. Statistics and Statistical Significance
3. Presenting Comparative Results
4. Evolutionary Optimization
5. Evolving Weights/Parameters/Architectures
6. Artificial Life Simulations

Model Selection

This module has studied various different neural network approaches, each of which has many parameters that need setting. So, how should we select which approach is best for a given problem, or which parameter values are best within a given approach?

We have seen how one can estimate generalization performance using a validation set or cross-validation, and other performance measures (such as learning speed) can be determined directly. However, these measures will usually depend on random factors (such as the random initial network weights, or the random order of training patterns) and so we will get a distribution of results for the chosen performance measure.

Comparing the mean performances across a number of runs makes sense, but how do we know how reliable those mean values are? Also, if there is high variance or skewed distribution across runs, are the mean values the appropriate thing to look at?

Generally, we need rigorous statistical procedures for comparing/selecting models.

Means, Variances and Standard Deviations

Suppose we run/train our neural network (or other model) n times, and measure the resulting performance x_i for each run, i.e. find $\{x_1, x_2, \ldots, x_n\}$. The obvious overall measure of performance is then the simple average or ***mean*** performance:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

That is fine, but it may be problematic if there is a high variance between runs – we might prefer a more reliable approach over one that has the best average. To measure this we compute the ***variance*** s^2 or ***standard deviation*** s of the sample:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}$$

This is a root-mean-squared measure of how much individual results deviate from the mean. The "n-1" rather than "n" in it reflects that $\bar{x}$ is a biased estimate of the mean, but this difference is small for reasonably large numbers n.

Standard Errors and Error Bars

Having computed the mean, one often wants to know how reliable or accurate that mean is. In other words, how much variance can we expect across different sets of n runs? Doing the algebra shows this to be simply related to the standard deviation:

$$S = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (x_i - \bar{x})^2} = \frac{1}{\sqrt{n}} s$$

This S is the ***standard error*** of the mean. It makes sense that the more runs we have, the better our estimate of the mean performance.

It is often said that "a graph is meaningless without error bars". If the graph (whether it be a plot of performance against time, or a histogram of final performance) doesn't have error bars indicating the standard error or standard deviation, then one has no way of knowing whether the graph is showing anything that can be relied on. Obviously, when it isn't obvious, one needs to say what the error bars represent.

t Tests and Statistical Significance

Intuitively, if one neural network approach has a better mean performance than another, and the difference is larger than the sum of the standard errors, then it is likely that the difference is significant. To measure the significance of differences more rigorously, there are numerous standard ***statistical tests***, each valid under particular ***conditions***.

The simplest such test for comparing neural network models is usually the ***t test***, which is implemented in most statistical and spreadsheet packages (e.g., Microsoft Excel). Given two sets of performance values $\{x_i : i = 1, \ldots, n\}$, this test will return a probability value p that (roughly speaking) tells us how likely it is that the two sets of results are not significantly different. One usually sets a threshold (e.g., 0.01), and if the returned p is less than that, we say that the difference is ***statistically significant*** at that level.

Note that a statistically significant difference does not necessarily mean that the difference is important. For example, one model may be so slightly better than the other that the extra computation involved is not worth the effort.

Distributions, Modes, Medians and Quartiles

If one plots the values $\{x_i : i = 1, \ldots, n\}$ as a histogram showing the number of times each value occurs, possibly after binning them, one gets an idea of the ***distribution*** of results. The peak of that distribution, or the most common value, is called the ***mode***.

Sometimes one finds the distribution of results over numerous runs is ***skewed*** – e.g. there are a very small number of very bad runs with very high errors that dominate the calculations of the mean and standard deviation. Or the distribution might be ***bimodal*** or ***multimodal***. In these cases, the means, standard deviations and t tests can be misleading, and other ***non-parametric*** measures are usually more useful.

The ***median*** is the middle value in the set $\{x_i : i = 1, \ldots, n\}$ when put in rank order, i.e. there are $(n\text{-}1)/2$ values smaller and $(n\text{-}1)/2$ values larger. The ***quartiles*** are the values one-quarter and three-quarters through the ranked set, i.e. there are $(n\text{-}1)/4$ values smaller and $(n\text{-}1)/4$ values larger. The median and quartiles can give a less skewed indication of the typical result and the variability across runs.

Presenting Comparative Results

We now have the key statistical tools for comparing and selecting the best models. One will typically run numerous (e.g., 10 to 30) independent runs (e.g., with different random number seeds) for a range of models (e.g., with different network architectures or parameter values). Then a validation set (or cross-validation) is used to select the models that are expected to work best on the test set. The overall performances of the selected models are then presented as means and standard deviations of their performance on the test set. The significance of any differences in the means can then be established by presenting p values from a two-tailed t test.

Normally, the comparisons should be carried out on a varied range of datasets, to provide an indication of how general any differences are.

When comparing different approaches/algorithms (e.g. a new algorithm against standard back-propagation) it is important that both algorithms are independently optimized (e.g., optimal back-propagation learning parameters are used for the chosen datasets).

Evolutionary Optimization

Trying different models and parameter values, comparing the resulting performances, and selecting the best, is fine if there are only a few models or parameters. However, neural networks usually have many interacting parameters and other details that need setting appropriately for good performance (e.g., architecture, learning parameters, and so on). In such cases, that have numerous interacting parameters requiring simultaneous optimization, the model selection process involves a search in a high dimensional space, and that usually proves difficult and time consuming to do "by hand".

For all but the simplest cases, some automated search process is required. Moreover, even with automation, simply sweeping sequentially through all possible parameter value combinations is generally far too computationally intensive.

In biological systems, such parameter values have evolved so that the systems perform well. So, the idea of using simulated ***evolution by natural selection*** to generate high performance neural networks is becoming increasingly popular.

Simulating Natural Selection

The general idea is to take a whole population of neural networks, each specified by some ***genotypic encoding*** of its architecture and other parameters. Each individual network is then tested on its chosen task and its ***fitness*** determined (e.g., estimated generalization performance or speed of learning). Then some form of ***selection*** process finds the best/fittest individuals to survive and be the parents of the next generation.

Simulated ***cross-over*** and ***mutation*** at the genotypic level creates children, which leads to a revised or completely new population, and the whole process is repeated.

By selecting the best individuals of each generation to survive and "breed", the average individual performance levels tend to improve from one generation to the next. If done well, we eventually end up with a population of very fit neural networks.

This general idea is simple, but there are a number of details that need thinking about if the best possible results are to be achieved [8].

What Neural Network Details Can Be Evolved?

The obvious thing to evolve would be the neural network ***weights***. That can certainly be done, but when there are known target outputs it is usually more efficient to use them with a more directed gradient descent based learning approach such as back-propagation [8]. However, if target outputs are not available, but some measure of overall fitness of the whole network is, then evolving the weights can be the most effective way to proceed, by a kind of reinforcement style learning.

Real brains, of course, are not born with all their connection weights specified innately. Their basic structure, learning processes and maybe a few key abilities are innate, but mostly their weights are set by lifetime learning [6].

Many researchers therefore concentrate on evolving the neural network ***architectures*** and ***learning parameters***, and use the evolved details with a non-evolutionary algorithm (e.g., back-propagation) for learning the weights. This is a good way to ensure that the parameters of different learning algorithms are optimized to allow fair comparisons.

Evolving Neural Network Weights

The idea is simple – the genotype is the set of weights, and the fitness is the estimated generalization based on a validation set. What is not so simple is specifying appropriate cross-overs and mutations. They need to be set up to respect the network architecture.

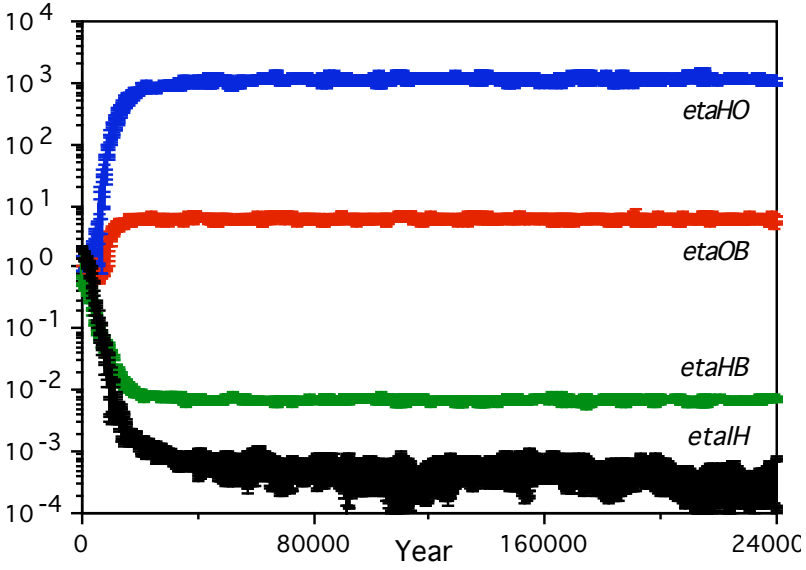
A particular issue is the so-called ***permutation problem*** – the network hidden units can be permuted without affecting the network outputs, but crossing-over between different equivalent networks can be catastrophic. It is very easy to end up with an algorithm that is essentially just performing an inefficient form of stochastic gradient descent.

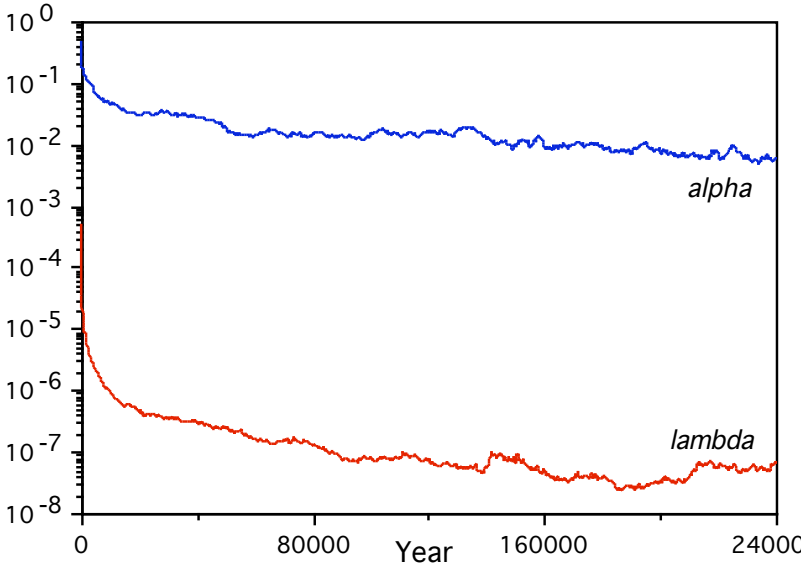
It is often claimed that evolving the weights can avoid the problem that gradient descent algorithms have of getting stuck in ***local optima***. However, it is hard to find solid evidence to support that claim based on large neural networks and realistic datasets.

One can evolve innate weights that form the starting point for lifetime evolution, and learned weights can be assimilated into them by the ***Baldwin Effect*** [2].

Evolving Learning Parameters

All the standard learning parameters can be evolved (initial weight range ρ, learning rate η, momentum α, regularization parameter λ, epochs, etc.). Many more parameters can be evolved than would be feasible to select by hand, e.g., allowing different weight ranges and learning rates for distinct subsets of weights and thresholds [1, 4].





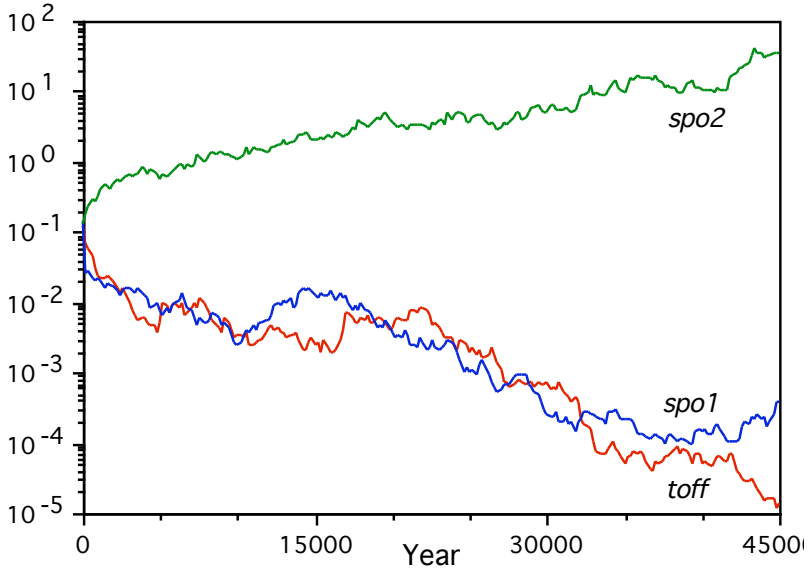
The (often enormous) error bars indicate how crucial the value of each parameter is.

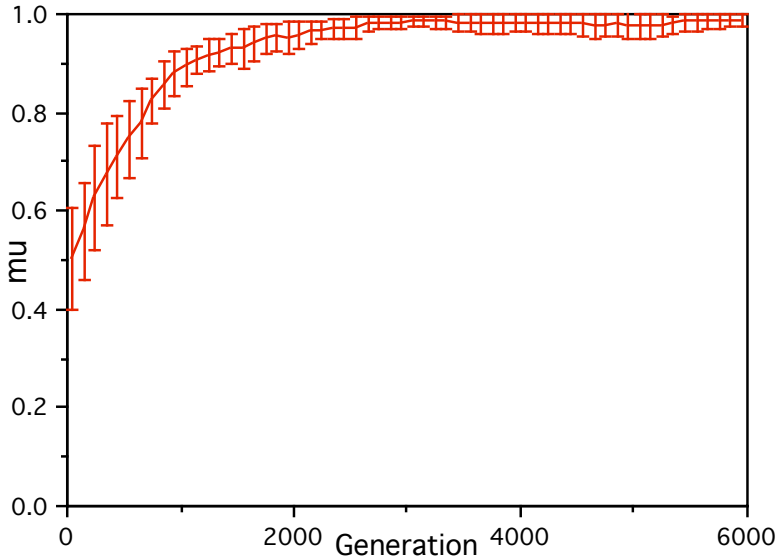
Evolving the Learning Cost Function

Two equivalent ways of evolving an improved learning cost function are:

Add SPO into weight updates: $\Delta w_{kl} = \eta . in_k . (targ_l - out_l) . [out_l . (1 - out_l) + spo2]$

Allow a mixed error function: $E = \mu E_{CE} + (1 - \mu) E_{SSE}$

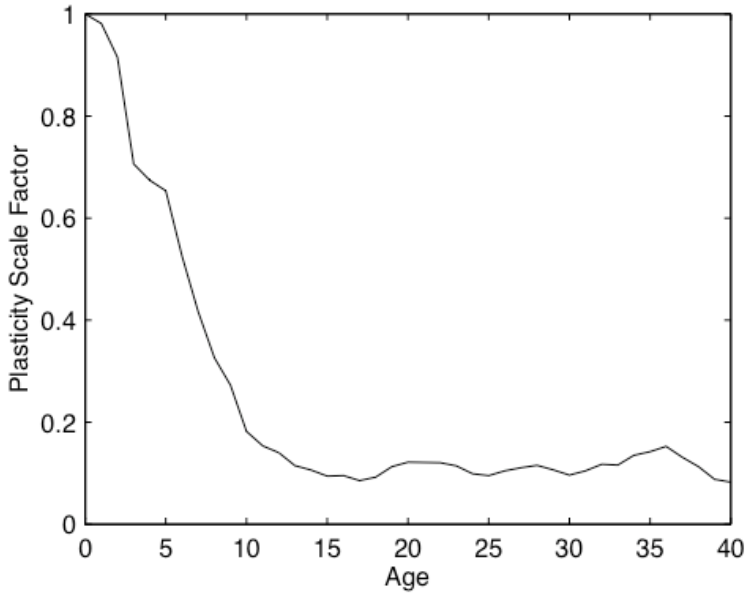


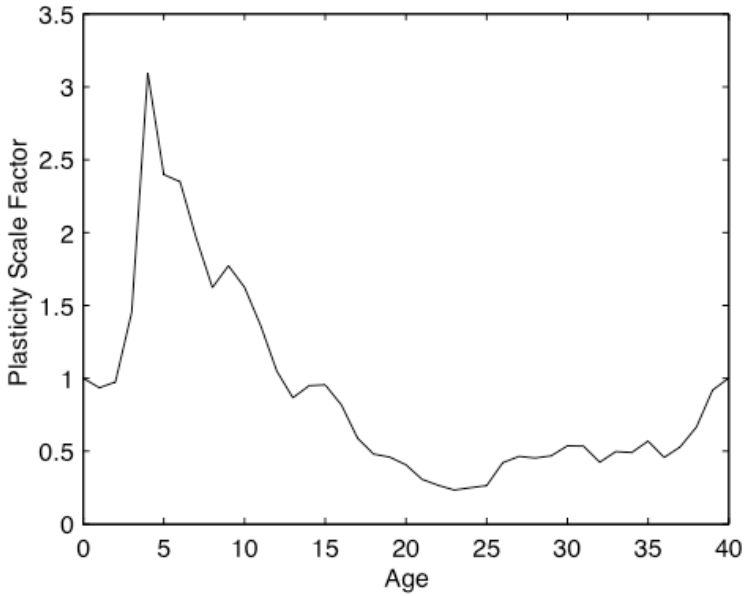


Either way, the SSE cost function evolves into the better CE cost function [1, 3].

Evolving Age Dependent Learning Rates

In many animals, learning becomes harder with age (like teaching old dogs new tricks). One can evolve a piecewise linear function (or simple two/three parameter exponential functions) to see if this idea works well for artificial neural networks [2, 4, 5].

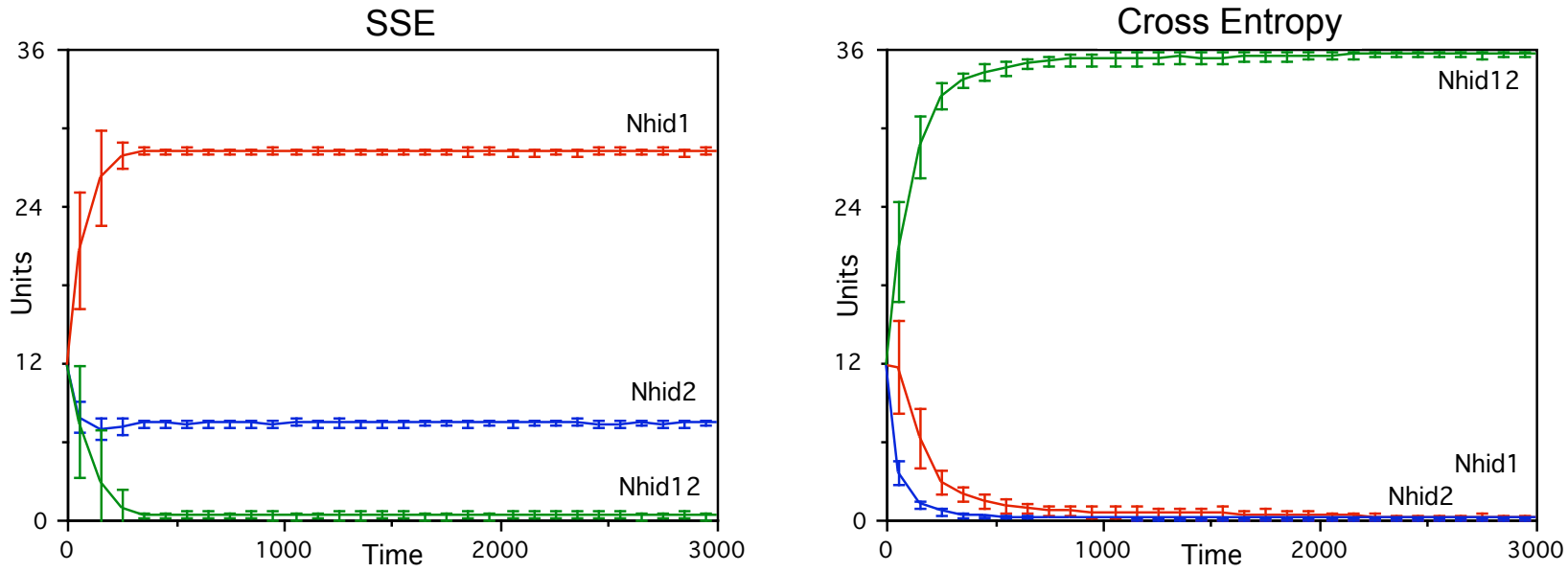




Normally, fast learning initially and slower later is best. However, if useful learning data only comes later, so does the evolved peak in learning rate [2].

Evolving Neural Architectures

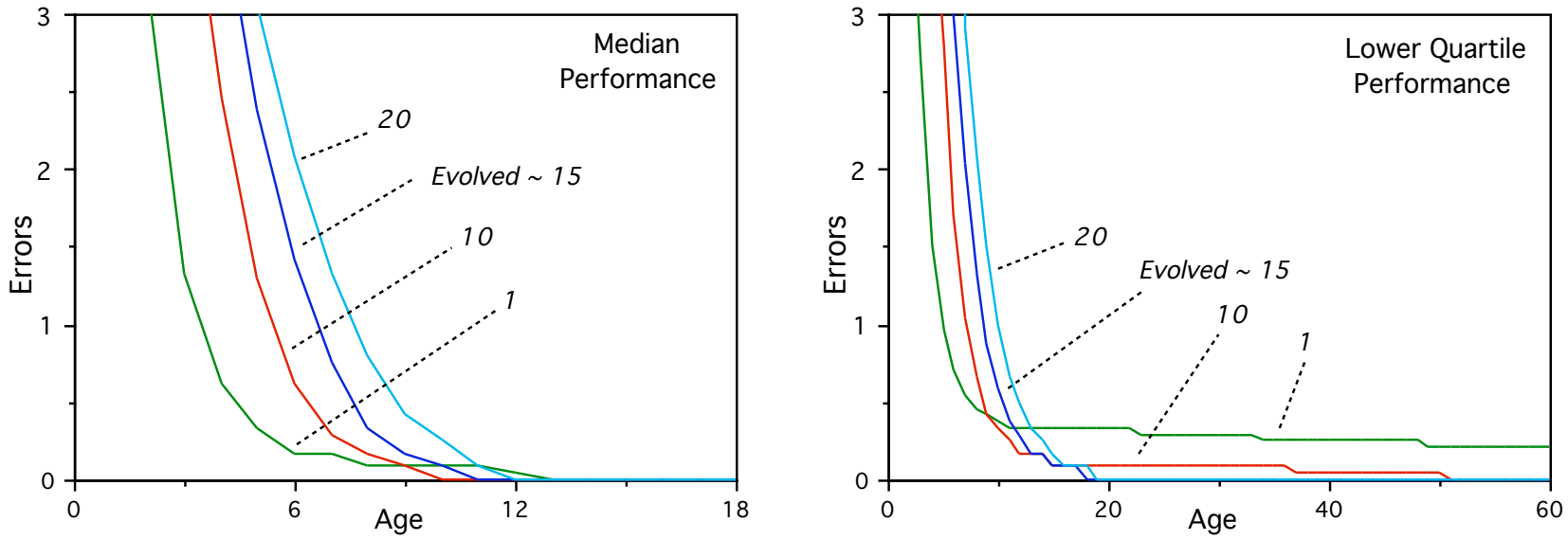
The obvious architectural details are the number of hidden layers, the number of hidden units, and connectivity patterns related to modularity. Two tasks 1 and 2 can evolve their own blocks of hidden units (*Nhid1* and *Nhid2*) and/or a shared block (*Nhid12*):



The results depend on which learning cost function (e.g., SSE or CE) is used, and how well any neurophysiological constraints (e.g., wiring volumes) are modelled [3].

Artificial Life Simulations

Evolving neural networks provide a good framework for agent-based artificial life simulations. For example, one can explore how populations of individuals can evolve to learn to perform particular tasks, and the effect that learning can have on the evolution of life history traits such as the periods of protection that parents offer their young.



More protection (15, 20 years) leads to better lower quartile performance [5].

Generational versus Steady State Evolution

Real biological populations tend to be comprised of individuals of many ages, learning and improving their fitness within their lifetimes, with a small proportion of the population dying and being replaced by children at each stage. Such a ***steady state*** approach is a biologically realistic option for evolving good neural networks.

A simpler approach is to only compare the fitness of the neural networks after they have all finished learning, and then replace the whole population in one go. Such a ***generational*** approach is often more straightforward to implement.

The steady state approach has the advantage of encouraging fast learning, as well as high final fitness, because the younger neural networks need to compete with the older networks to breed and survive. The generational approach, however, tends to encourage less risky learning strategies that result in good final performance more consistently. The choice usually depends on what aspects of the neural networks need to be evolved, and how biologically realistic the simulated evolution needs to be [4].

More Sophisticated Approaches

Some researchers aim to do more than just optimize a few neural network parameters. A particularly interesting series of approaches are ***NEAT*** (NeuroEvolution of Augmenting Topologies), ***HyperNEAT*** (Hypercube-based NEAT), and ***ES-HyperNEAT*** (Evolvable Substrate HyperNEAT) [7].

NEAT uses a direct encoding that starts with small simple networks and complexifies them over generations with increasing performance. HyperNEAT uses Compositional Pattern-Producing Networks (CPPNs) which allow compact indirect encoding of patterns with regularities such as symmetry and repetition with variation. This allows the pattern of weights to be generated as a function of geometry. The ES-HyperNEAT extension can also "determine the internal geometry of node placement and density, based only on implicit information in an infinite-resolution pattern of weights" [7].

There are numerous well-written papers and web-pages describing these approaches, and source code to play with in various languages for those with plenty of spare time.

References / Advanced Reading List

1. Bullinaria, J.A. (2003). Evolving Efficient Learning Algorithms for Binary Mappings. *Neural Networks*, **16**, 793-800.
2. Bullinaria, J.A. (2003). From Biological Models to the Evolution of Robot Control Systems. *Philosophical Transactions of the Royal Society of London* A, **361**, 2145-2164.
3. Bullinaria, J.A. (2007). Understanding the Emergence of Modularity in Neural Systems. *Cognitive Science*, **31**, 673-695.
4. Bullinaria, J.A. (2007). Using Evolution to Improve Neural Network Learning: Pitfalls and Solutions. *Neural Computing & Applications*, **16**, 209-226.
5. Bullinaria, J.A. (2009). Lifetime Learning as a Factor in Life History Evolution. *Artificial Life*, **15**, 389-409.
6. Elman, J.L., Bates, E.A., Johnson, M., Karmiloff-Smith, A., Parisi, D. & Plunkett, K. (1996). *Rethinking Innateness: A Connectionist Perspective on Development*. MIT Press.
7. Risi, S. & Stanley, K.O. (2012). An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density and Connectivity of Neurons. *Artificial Life*, **18**, 331–363.
8. Yao, X. (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE*, **87**, 1423-1447.

Overview and Reading

1. We began by considering the need to compare models using rigorous statistical techniques, and outlined the relevant ideas such as means, standard deviations, standard errors, error bars, modes, medians, and quartiles. The *t test* was suggested as being the simplest way to measure the statistical significance of differences between models.
2. We ended with an overview of using simulated evolution by natural selection to optimize and select the best models automatically. A range of examples from recent publications were presented.

Reading

The topics covered in this lecture are not dealt with in detail by any of the recommended books. The statistical ideas will be covered in any good book on statistics. For evolutionary neural networks you will need to look at the papers referenced and search for more recent papers on specific aspects.