A systematic comparison of flat and standard Cascade-Correlation using a student-teacher network approximation task

FREDERIC DANDURAND[*]†, VINCENT BERTHIAUME†, THOMAS R. SHULTZ†

† McGill University, Department of Psychology, 1205 Dr. Penfield Ave., Montréal, Québec, H3A 1B1, Canada

Cascade-correlation (cascor) networks grow by recruiting hidden units to adjust their computational power to the task being learned. The standard cascor algorithm recruits each hidden unit on a new layer, creating deep networks. In contrast, the flat cascor variant adds all recruited hidden units on a single hidden layer. Student-teacher network approximation tasks were used to investigate the ability of flat and standard cascor networks to learn the input-output mapping of other, randomly initialized flat and standard cascor networks. For low-complexity approximation tasks, there was no significant performance difference between flat and standard student networks. Contrary to the common belief that standard cascor does not generalize well due to cascading weights creating deep networks, we found that both standard and flat cascor generalized well on problems of varying complexity. On high-complexity tasks, flat cascor networks had fewer connection weights and learned with less computational cost than standard networks did.

Keywords: Neural network architectures, cascade-correlation, network depth, connectivity, function approximation

AMS Subject Classification: 62M45, 92B20

1 Introduction

Backpropagation is a popular neural network algorithm, and a standard to which other learning techniques are often compared. The number and arrangement of hidden units and hidden layers in backpropagation networks is generally set by the network designer based on domain-specific expertise or heuristic rules (e.g. see Rafiq *et al.*, 2001).

By contrast, in constructive algorithms, hidden unit topology is determined automatically as part of the learning process. Constructive networks can be initialised without any hidden units and they accumulate hidden units until they have enough computational power to solve the target task.

[*] Contact email: fdandu@ego.psych.mcgill.ca (Frederic Dandurand)

Several constructive neural networks techniques have been proposed. Cascade-Correlation (cascor) (Fahlman and Lebiere 1990) is a general purpose constructive technique capable of learning both classification and continuous function approximation tasks.

In this paper, we address a common criticism of standard cascor: a poor ability to generalize due to its tendency to build deep networks of many hidden layers. We compare generalization of standard cascor with another variant of cascor that builds shallow, one-hidden-layer networks. This question is important because neural network architectures are often evaluated and compared based primarily on their ability to generalize.

In contrast to cascor, most other constructive neural network algorithms (e.g. Extentron (Baffes and Zell 1992), Upstart (Frean 1990), Tiling (Mezard and Nadal 1989), Divide & Conquer (Romaniuk and Hall 1993), and Pyramid (Parekh *et al.* 2000)) are limited to classification tasks because their learning relies on the partitioning of training patterns as correctly and incorrectly classified, or on the positioning of hyperplanes to divide the input space into discrete regions. Consequently, these algorithms generally build neural networks composed exclusively of discontinuous, binary threshold units.

Classification can be considered as a special case of function approximation where outputs are binary or *m-ary*. In contrast to binary classifications, function approximation can be more complex when the quality of the fit is assessed using a continuous (real-valued) error function requiring finer discriminations. To our knowledge, the only important constructive neural network algorithm besides cascor capable of learning function approximation tasks is Constraint Based Decomposition (CBD) (Draghici 1996, 2001). When used for function approximation, this algorithm first converts the original task where targets are real-valued into a simpler, discrete classification problem using the sign of those patterns. For example, if the original problem has targets -2.5, 1.3, -0.5, the discrete classification problem would be -1, +1, -1. This classification problem is learned using the standard CBD algorithm, and the resulting network (topology and weights) is used to initialize a standard backpropagation neural network to be trained on the original (function approximation) task. Although interesting, CBD appears less cognitively and biologically plausible than cascor because of the additional complexity and processing required to implement the discrete classification task, and because there is no psychological or neurological evidence for a mechanism to switch between classification and function approximation tasks.

1.1 Cognitive modelling

Constructive neural network algorithms are interesting for modelling psychological processes because they are naturally able to model learning and cognitive development in a single unified system (Shultz 2003). Cascor has been successfully applied to the modelling of many cognitive developmental tasks such as the integration of distance, time, and velocity concepts (Buckingham and Shultz 2000), the balance-scale problem (Shultz *et al.* 1994) and personal pronouns (Takane *et al.* 1995). Cascor has several advantages in modelling cognition. For one thing, there is neuroscience evidence for network growth through neuro- and synapto-genesis under the control of learning (Shultz *et al.* 2007). Also, the Knowledge-Based Cascade-Correlation (KBCC) variant of the algorithm (Shultz and Rivest 2001) can model how relevant prior knowledge can be recruited to facilitate the learning of a new task, a phenomenon that is very common in human learning (Wisniewski 1995).

1.2 The cascor algorithm

The cascor algorithm begins with a simple network topology consisting of input and output units only, and recruits hidden units to provide the network with additional computational power for learning. Units (typically with sigmoid transfer functions) are recruited from a pool of candidates. Learning in cascor proceeds in an alternation of two phases:

- In input phase, input weights of all candidates units in the pool are trained to maximize the covariance (S) between their outputs (V) and the residual error (E):

  $$S = \sum_o \left| \sum_p (V_p - \overline{V})(E_{p,o} - \overline{E_o}) \right|$$

  where *o* is the network output and *p* is the training pattern. $\overline{V}$ and $\overline{E}_o$ represent the mean values of *V* and *E* over all patterns. The candidate unit with the highest covariance is then inserted into the network at the end of an input phase.
- In output phase, all the weights connected to the output layer are trained to minimize residual network error.

In input and output phases, learning is done using an algorithm for training feed-forward networks, such as QuickProp (Fahlman 1988). Changes of phase occur when covariance maximization or error reduction stagnates in the current phase.

In its original, standard form described by Fahlman and Lebiere (1990), the cascor algorithm cascades all recruited hidden units. That is, in addition to being fed by input units, hidden units are fed by all previously recruited hidden units. As a result, cascor inserts each new unit into a new layer, creating deep networks with as many hidden layers as recruited units.

By contrast, the flat variant of cascor, described by Sjogaard (1991), adds new recruited units onto a single layer (i.e. cascaded connections are eliminated), thus limiting the depth of the network. Except for this difference in hidden unit connectivity, flat and standard cascor are identical.

1.3 Generalization in cascor networks

Generalization in cascor networks is sometimes cited as problematic, but the literature on cascor generalization provides contradictory results. In psychology simulations, cascor appears to generalize better than backprop. Shultz (2006) compared backpropagation and cascor on ability to learn and cover several cognitive-developmental phenomena. Cascor was consistently superior, but the limitations of backpropagation were not restricted to generalization problems. Sometimes backpropagation failed to learn, and other times it failed to cover developmental stages seen in children.

On the other hand, Adams and Waugh (1995) reported that, in a Gaussian-function approximation problem, cascor produced an uneven stairlike output function exhibiting poor generalization compared to a flat backpropagation network. The constructive nature of cascor may cause sigmoid units to saturate, as Adams and Waugh observed: 'the steepness of the steps reflects the fact that many of the sigmoids in the net are being driven hard on or hard off' (p. 945). Possibly this is caused by the fact that learning in output phase continues until output error stagnates, at which point the learning algorithm switches to input phase to recruit a new hidden unit. Output error stagnation usually happens after sigmoids saturate. Furthermore, the cascor learning algorithm freezes input weights of recruited units, which keeps them saturated. This allows cascor to build powerful feature detectors, but also tends to generate highly nonlinear outputs. As a possible solution to problems with sigmoid units in cascor, Hwang *et al.* (1996) proposed trainable nonlinear nodal activation functions. Such saturation effects are not unique to cascor; they are potentially present in any neural network using sigmoid units, including standard backpropagation networks.

Excessive network depth due to hidden-unit cascading is also considered detrimental to generalization (Prechelt 1997). Deep cascading in standard cascor has been criticized for biasing cascor towards nonlinearity (Prechelt 1997) and for harming the learning of more linear problems (Sjogaard 1991). A variety of techniques restrict the number of connection weights, for example, pruning (Hansen and Pedersen 1994), restricting fan-in (Phatak and Koren 1994, Klagges and Soegtrop 1992) and limiting cascaded connections between layers (Waugh and Adams 1994). These techniques can each help to limit network depth.

Sjogaard (1991) proposed a modification in cascor connectivity, called flat cascor, to restrict network depth to a single hidden layer. He found that the flat variant of cascor generalized better than standard cascor on the single artificial problem he tested. Prechelt (1997) compared, among other things, standard cascor with a version of flat cascor that uses error minimisation for recruiting hidden units on his PROBEN1 test bed. He measured generalization ability and number of recruited units and found that, in most cases, there was no significant difference between the two algorithms. In the cases where there were differences in generalization, 'not cascading hidden units is superior to cascading them on some problems and inferior on others; the former case occurred more often' (Prechelt 1997; p. 895). Littman and Ritter (1993) found that, on a task consisting of time series prediction based on the Mackey-Glass differential equation, deeply cascaded network architectures[†] tend to overfit small data sets less than shallow, broad architectures (i.e. flat cascor) containing the same number of nodes.

[†] The deeply cascaded architectures used by Littman and Ritter (1993) are not strictly speaking standard cascor because they allow multiple units per hidden layer, but they are nonetheless deep.

In short, deciding whether or not to cascade hidden units is not straightforward. Whether standard (deep) cascor networks are better or worse than flat cascor networks is controversial and may well depend on the particular task being learned. Furthermore, performance on small collections of problems may not be representative of overall performance, meaning that further experimentation is required before a conclusion can be reached.

1.4 A novel approach to studying cascading hidden units in cascor

Because previous research, with its variable and inconsistent results, may be limited to the particularities of the specific problems investigated, our work introduces a problem-neutral empirical approach for evaluating the effect of cascading hidden units. Our method is based on having networks learn the input/output functions of other cascor networks, also known as the student-teacher task (Saad 1999). More specifically, standard and flat cascor networks were trained to approximate the output of either standard or flat, randomly-initialized cascor networks. Our goal was to determine whether flat or standard cascor networks make a better function approximator on a task where the biases are known and strictly controlled. The characteristic difference between standard and flat cascor networks is the respective presence vs. absence of connection weights between hidden units.

The following terms are used throughout the article:

1. Network's I/O function: Function implemented by a neural network that maps its inputs onto its outputs. The complexity of this function is related to the number of hidden units in the network, to network connectivity and to the magnitudes and signs of connection weights.
2. Teacher network: a cascor network (standard or flat) that is used to generate training and test patterns. It is initialized with a predetermined number of hidden units, and random connection weights. It is not trained on any task; instead its network I/O function depends only on those random weight values. Teacher networks' role is limited to providing a random input/output function on which student networks will be trained.
3. Student network: a cascor network (standard or flat) that learns the training set produced by the teacher network, using the cascor training algorithm. In other words, student networks' I/O functions are trained to approximate I/O functions of teacher networks.

Student-teacher tasks have been used for studying various characteristics of neural networks, including overfitting (Amari *et al.* 1997, Lawrence *et al.* 1997), the probability distribution of performance parameters (Lawrence *et al.* 1997), and learning algorithms (Park *et al.* 2004).

In our context, the student-teacher method affords a direct comparison of the functional mappings (inputs onto outputs) that the two architectures are capable of representing. Unlike real-world problems with unknown biases, biases here are known and controlled. The I/O functions produced by standard teachers are known to be more nonlinear than those produced by flat teachers. This method also allows systematic exploration of large areas of problem space because weights are randomly initialized and the complexity of the network's I/O function is varied by manipulating the number of hidden units in the teacher network. We compare standard and flat cascor networks to investigate the impact of cascading weights on different performance measures, at different complexity levels. Are cascades detrimental to generalization? In other words, do flat cascor networks generalize better than standard networks? Also, are there differences between the two architectures in terms of size and training efficiency?

2 Experimental design

This section introduces experimental design parameters. This experiment uses a three-way one-repeated-measure design with two independent factors (*teacher-network type* with 2 levels (flat and standard) and *teacher hidden count* , h_t, with 10 levels (2, 4, 6, 8, 10, 12, 14, 16, 18 and 20)) and one repeated factor (*student-network type* with 2 levels (flat and standard)). This design affords all combinations of teacher and student types (flat-flat, standard-flat, flat-standard and standard-standard) at different levels of complexity.

A relatively large number (six) of performance measures of student networks are recorded:

- Network size after training
  - Number of recruited hidden units
  - Number of weights
- Training effort
  - Number of training epochs
  - Computational cost for training
- Approximation quality
  - Error on train set (accuracy)
  - Error on test set (generalization)

In this experiment teacher networks are constructed (as opposed to being trained) with a given number of hidden units and a given connectivity (flat or standard). Network connections are initialized with random weights. Teacher networks had no direct input-output connections: input activations had to pass through hidden units to get to the outputs. This maximized the difference between flat and standard teachers because their characteristic difference lies in the way hidden units are connected. To explore large portions of the possible I/O function space, and to provide sufficient power for statistical analyses, experimental conditions were repeated 20 times with different teacher random weights for each combination of teacher network type (flat or standard) and number of hidden units h_t.

2.1 Teacher I/O function complexity

The number of hidden units in the teacher network, h_t, influences task complexity. Fig. 1 shows examples of low (2 hidden units) and high (20 hidden units) complexity network I/O functions generated using 2 inputs and 1 output. Compressed file size can be used as a measure of objective information content (Donderi 2006; Donderi and McFadden 2005). Mean file size (N=20 networks per condition) is presented in Fig. 2 as a function of number of hidden units. Images were compressed using Portable Network Graphics (PNG), an open, lossless image compression format similar to JPEG.

Insert fig.1 about here

We performed a 2 by 2 ANOVA to examine the effects on file size of teacher type (2 levels: standard and flat cascor) and of the number of hidden units in the teachers, h_t, (2 levels: 2 and 20 hidden units). We found three significant effects. First, file size increases with the number of hidden units h_t (F(1,396) = 743, $p < 0.001$), indicating that the more hidden units, the more information content, and thus greater complexity. Second, we found a main effect of teacher type (F(1,396) = 28, $p < 0.001$), indicating that standard teachers generated larger files, and therefore more complexity, than flat teachers. Finally, we found an interaction between teacher and h_t (F(1,396) = 37, $p < 0.001$) suggesting that standard teachers are more complex than flat teachers only when the teacher network has many hidden units.

Insert fig.2 about here

2.2 Number of connection weights and computational cost

Table 1 shows how the number of connection weights is calculated for each student network.

Insert table 1 about here

Computational cost measures the number of weight adjustments required for training the network, as shown in Table 2. To compute this value, as training alternates between input and output phases (beginning in output phase), the sum of number of weights to train is multiplied by the number of epochs in each specific phase.

Insert table 2 about here

2.3 Task parameters

In contrast to most real-world tasks where the problem determines certain neural network parameters such as number of inputs and outputs, student-teacher tasks have several free parameters. In this section, these parameters are explained and their settings are justified. The task parameters included number of inputs and outputs, levels of the h_t factor, the limit on the number of hidden units in the student ($h_{s\ max}$), the number of training patterns, and the range over which random connection weights in teacher networks vary.

2.3.1 Inputs and outputs

We used continuous coding for input values. We selected an input range of [-1,1] because it is representative of typical encodings. Our choice of number of inputs was the result of a compromise between choosing a realistic value (i.e. representative of real-world problems) and limiting the number of inputs to minimize computational cost. In Prechelt's (1997) real-world benchmarks, where a mixture of binary and continuous inputs are used for each function approximation problem, the mean number of continuous inputs is 5 (min=0, max=14) and the mean number of total inputs is 27 (min=8, max=125). We chose 6 continuous inputs as a compromise value. A single continuous output with an asigmoid activation function (range 0 to 1) was used to allow visualization of training data using gray scale coding (see Fig. 1).

2.3.2 Number of hidden units in teacher networks (h_t) and maximum number of recruitments in student networks ($h_{s\ max}$)

In order to study the effect of complexity, the range of [2,20] was chosen for the number of hidden units in the teacher network (h_t). This range yields a wide coverage of problem complexity while being compatible with real-world tasks, as the average number of hidden units recruited for learning tasks in the PROBEN1 problem set was 8 (Prechelt 1997).

Because task difficulty can vary depending on teacher networks' random initialization, we expected a large variance in the number of student network recruits. To limit computational cost, we imposed a ceiling (called $h_{s\ max}$) on the maximum number of hidden units student networks are allowed to recruit. We empirically determined that a maximum of 10 times the number of hidden units in the teacher network (h_t) was a suitable value for $h_{s\ max}$. The argument is twofold. First, as explained in section 2.3.3, the number of training patterns must exceed the number of connections in the student network. To establish a fixed upper bound on the number of training patterns, we need to impose a limit on the maximum number of hidden units in student networks. Second, if students did not learn the task with ten times more hidden units than h_t, they may be stuck in local error minima perhaps due to unfavourable initial conditions. Thus, we specify that student networks can recruit at most 200 units, occurring when the teacher network contains 20 hidden units.

2.3.3 Number of training patterns

With this artificial task, we can generate as much data as we like, i.e. we can use the teacher network to produce any number of training patterns. If we select too few training patterns, there is a risk of overfitting. Indeed, if student networks have too many connections for a given training set size, they may have enough computational power to rote-memorize training patterns. Such overfitting usually results in poor generalisation. A common guideline to prevent overfitting is to make sure there are more training patterns than there are connections to train.

Consistent with this guideline, we selected a number of training patterns equal to the maximum possible number of connections in a student network. As seen in section 2.3.2, the maximum number of hidden units in student networks is 200. Thus, the maximum number of connections in a student network with 6 inputs is 21507 (see Table 1). Our legacy code represented the input space as a square grid, so it was convenient to use a squared integer to compute sample size. The closest squared integer was 147. Thus, we used 21609, the square of 147, as training and testing set sizes in all simulations.

2.3.4 Connection weight range

We determined the range over which random connection weights in the teacher network would vary. In general, the complexity (or difficulty) of the task increases with weight range because output units saturate when the weighted sum of their inputs is large. We empirically selected a weight range of [-1,1] to make the number of recruits in the student network to be approximately equal to the number of hidden units in the teacher network (i.e. $h_t \approx h_s$).

2.3.5 Summary of simulation parameters

Table 3 summarizes simulation parameters used for student networks, both standard and flat.

Insert table 3 about here

2.4 Learning parameters

We trained student networks using QuickProp (Fahlman 1988). Table 4 summarizes the learning parameters used for student networks. Except for score threshold, we used the default learning parameter settings of Fahlman's original LISP implementation of cascor. No weight change is allowed to be greater in magnitude than *maximum growth factor* times the previous step for that weight (Fahlman 1988). Cascor changes phase if error reduction (or correlation increase) has been lower than *change threshold* across *patience* epochs (Fahlman 1990). Cascor will also switch phase after having reached *max epochs* in the current phase. *Decay* is used to keep weights from growing too big. The *learning rate* controls the amount of gradient descent used in updating weights. See Fahlman (1990) and his LISP code in the CMU AI repository for additional details of learning parameters.

Score threshold controls how close network output values must be to target values for learning to be successful. In output phase, training finishes when, for all outputs $o_{i,j}$ and targets $t_{i,j}$, $| o_{i,j} - t_{i,j} | <$ *score threshold*. Fahlman's default value for score threshold is 0.4. For target values in a range of [0,1], such a score threshold can be suitable for binary classification tasks, but not for function approximation because it allows for a tolerance on approximation error of 80% (+/- 0.4 on a range of [0,1]). Here, we selected a score threshold of 0.05 to be within 10% (i.e. +/- 0.05) of target values.

Insert table 4 about here

3 Results

Using a General Linear Model (GLM), we performed univariate tests to investigate which independent factors had significant influences on the dependent measures. We further analysed significant interactions using simple main effects tests. This section presents the important and statistically significant results.

To improve normality, we performed all our analyses with log-transformed data. However, for ease of interpretation, all reported means and SDs and Figs. 5 to 8 use untransformed data. Consequently, error bars in these figures are not indicative of statistical significance. Table 5 summarizes results of the univariate GLM analyses.

Insert table 5 about here

3.1 Number of recruited hidden units

The number of hidden units is a measure of student network size. Results are presented in Fig. 3 for standard teachers and Fig. 4 for flat teachers. As noted in Table 5, for this dependent measure we found significant main effects of h_t and teacher type, and a significant interaction between student and teacher.

Insert fig. 3 about here

Insert fig. 4 about here

First, the main effect of the number of hidden units in teacher networks (h_t) reflects that student networks recruited about 50 times more hidden units with complex tasks (i.e. teacher I/O functions generated with $h_t = 20$ hidden units) than with simple tasks ($h_t = 2$ hidden units). At $h_t = 20$ they recruited a mean of 15.6 (SD=18.3) units, and at $h_t = 2$, student networks recruited a mean of 0.3 (SD=0.7) units.

Second, the significant main effect of teacher type reflects that student networks recruited about twice as many hidden units to learn standard-teacher tasks (M=7.0, SD=15.2) than flat-teacher tasks (M=3.6, SD=8.6). This is further evidence that standard-teacher tasks are more complex than flat-teacher tasks.

Finally, the significant interaction between teacher and student types suggests that, although students did not require significantly different numbers of recruits with flat teachers (flat student: M=3.4, SD=7.8; standard student: M=3.8, SD=9.3; difference not significant at p=0.344), standard students recruited fewer units than flat students with standard teachers (standard students: M=5.7, SD=10.7; flat students M=8.4, SD=18.6, difference significant at p=0.012). In short, with standard teachers, standard students recruited fewer units than did flat students.

3.2 Number of connection weights

Number of connection weights is another measure of network size. Results on number of weights are presented in Fig. 5 for standard teachers and Fig. 6 for flat teachers. As noted in Table 5, there were significant main effects of student type, h_t and teacher type, and a significant interaction between student and h_t.

Insert fig. 5 about here

Insert fig. 6 about here

First, the main effect of student type reflects that standard students built networks with about twice as many connection weights (M=104, SD=281) as flat students (M=54, SD=116).

Second, the main effect of the number of hidden units in teacher networks (h_t) reflects that complex tasks (h_t = 20) required about 25 times more connections (M=251, SD=413) than simple tasks (for h_t = 2, M=9.4, SD=5.8 connections).

Third, the main effect of teacher type reflects that student networks had about 1.6 times more connections when learning a standard-teacher task (M=98, SD=227) than a flat-teacher task (M=60, SD=204).

Finally, the h_t x student interaction reflects that student networks did not differ for simple tasks (for h_t = 2, flat students M=9.4, SD=6.8; standard students M=9.5, SD=4.7; p=0.8), but that standard students had more connections (M=363, SD=542) than flat students (M=138, SD=160; significant difference at p<0.001) when task complexity was high (h_t = 20).

In short, an analysis of number of connection weights in student networks suggests that student networks did not differ in size for simple tasks, but that flat networks were smaller when learning complex tasks. Furthermore, those results support our hypothesis that task complexity increases with the number of hidden units in the teacher (h_t), and that standard-teacher tasks are more complex than flat tasks.

3.3 Number of epochs

The number of total epochs to train is a measure of training effort. Results are presented in Fig. 7 for standard teachers and Fig. 8 for flat teachers. As noted in Table 5, we found significant main effects of h_t and teacher type, and a significant interaction between student and teacher.

Insert fig. 7 about here

Insert fig. 8 about here

First, the main effect of number of hidden units in the teacher (h_t) reflects the fact that student networks took about 25 times longer to learn complex teacher functions (M=640, SD=786 epochs for $h_t = 20$) than simpler functions (M=24, SD=50 epochs for $h_t = 2$).

Second, the main effect of teacher type reflects that students needed about 1.8 times more epochs to learn standard-teacher functions (M=287, SD=570) than flat-teacher functions (M=162, SD=388).

Finally, the interaction between student and teacher types reflects that flat students took fewer epochs to learn the flat-teacher functions (M=133, SD=261) than standard students (M=191, SD=482; difference significant at p=0.01), but that standard and flat students did not differ (p=0.8) in the number of epochs needed to learn standard-teacher functions (standard students: M=274, SD=523; flat students: M=300, SD=615).

In short, students took more epochs to learn more complex tasks and complexity increased with standard teachers and the number of hidden units in the teacher network. Also, flat students learned flat-teacher tasks faster than standard students did, while flat and standard students learned standard-teacher tasks at similar speeds.

3.4 Computational cost for training

Computational cost is another measure of training effort. As noted in Table 5, there were significant main effects of student type, h_t and teacher type. Figs. 9 and 10 present results of computational training cost for standard and flat teacher networks respectively.

Insert fig. 9 about here

Insert fig. 10 about here

First, the main effect of student reflects the fact that flat students trained about 1.5 times more efficiently (M=7735, SD=26229) than standard students (M=11616, SD=32423). The size and direction of this main effect are similar to that of student type on number of connection weights, suggesting those additional computations are used to train the extra cascaded weights in standard students,.

Second, the main effect of h_t reflects that complex teacher tasks (M=30188, SD=49511 for $h_t = 20$) took about 50 times more computation to train than simpler teacher tasks (M=627, SD=1467 for $h_t = 2$).

Finally, the main effect of teacher type reflects that standard-teacher functions took about twice as much computation to learn (M=12631, SD=34012) than flat-teacher functions (M=versus 6720, SD=23927).

3.5 Student network error

Fig. 11 and Fig. 12 present results of error on test set (generalization) for standard and flat teacher networks respectively. As was seen in Table 5, we found no significant main or interaction effects on accuracy (error on train set) or generalization (error on test set). Accuracy and generalization graphs were very similar, thus for conciseness, only generalization graphs are displayed here.

The similarity between accuracy and generalization suggests two things. First, flat and standard cascor networks generalised well because network error was not larger on unseen (test) data than on data used for training. Second, the density of training data was probably high enough to get a good function approximation in all regions of the problem space.

Insert fig. 11 about here

Insert fig. 12 about here

4 Discussion

Our results suggest that:

1. Both flat and standard networks were able to learn and generalise well on functions generated by either standard or flat teacher networks.
2. There were no differences between standard and flat cascor in generalization and accuracy at any level of complexity studied.
3. When the task was simple[‡] (for teacher networks of fewer than about 14 hidden units), there were no performance differences between flat and standard student networks.
4. When there were student differences on training effort, flat networks trained more efficiently than standard networks. First, flat students trained with less computational cost than standard students. Second, flat students learned flat tasks, but not standard tasks, in fewer epochs than standard students did.
5. When there were student differences on network size, flat students required fewer connections than standard students did. Furthermore, standard students recruited fewer units than flat students, but only when learning standard teacher tasks.
6. Task complexity, which depends on teacher networks' I/O functions, was larger for standard networks than for flat networks having the same numbers of hidden units, as shown by larger compressed file size. Standard teacher networks may have been able to build more complex I/O functions due to the cascaded connection weights between hidden layers. This was reflected in our results showing that student networks took more computation to train and built larger networks when learning standard teacher functions than flat teacher functions.
7. Task complexity also increased with the number of hidden units in the teacher network. Again, this was reflected in our results showing that student networks took more computation to train and built larger networks when learning teacher functions of more hidden units (h_t).

Our research examined a comprehensive set of six dependent measures. In contrast, other researchers employed fewer measures. For instance, Sjogaard (1991) considered a set of five measures similar to ours but without the computational cost of training. Adams and Waugh (1995), Littmann and Ritter (1993), and Lahnajarvi et al.(2002) all employed four measures, and finally Prechelt (1997) focused on two measures only, generalization ability and number of recruited units.

4.1 Generalization

The answer to our main experimental question is therefore that both flat and standard cascor generalise well on problems with wide ranges of complexity. This may help to settle a controversy in the literature on cascor. Previous research did not find a conclusive answer about generalization abilities of flat and standard cascor. Prechelt (1997) studied many variants of cascor. His flat variant of cascor was trained in input phase using error minimization and not covariance maximization, so a direct

[‡] Figure 1 presents examples of data sets generated by low and high complexity networks.

comparison with our results is difficult. In general, he found no differences in generalization on most of the real-life problems in the PROBEN1 test bed between the cascor variants he studied, which included differences in unit cascading (network connectivity). Furthermore, he found that when there were differences, the architecture that performed better depended on the problem.

Although they did not use strictly flat cascor, the results of Phatak and Koren (1994) suggest that limiting network depth may improve generalization, but they note that no conclusion can be drawn based on their analysis.

Other work supports contradictory conclusions: that flat cascor generalizes better than standard cascor (Sjogaard 1991) or that standard cascor generalizes better than flat cascor (Littman and Ritter 1993). Our experiment shows that generalization differences between flat and standard cascor may not exist with tasks that control bias. When such differences are found in real-world problems, they may be due to unknown particularities of the problems used that may favour one topology over another.

Our findings differ from those of Adams and Waugh (1995). On a simple and smooth one-dimensional function approximation problem ($1 / (1+x^2)$), they found that cascor generalized poorly because sigmoids in the network tended to saturate, thus producing uneven stairlike outputs. In contrast, our protocol reduces this problem by using cascor networks to generate training data, yielding problems whose level and type of nonlinearity are more compatible with cascor networks. Besides being six-dimensional, our networks' I/O functions are much more complex than the one used by Adams and Waugh, especially when the number of hidden units in the teacher network is large.

4.2 Network size and training effort

In this work, we also considered network size and training effort. We only found differences when task complexity was very high, higher than would normally be found in real-world problems (e.g. Prechelt 1994).

When task complexity was high, flat networks trained more efficiently than standard students did, using fewer epochs on flat-teacher tasks, and with about 1.5 times less computational cost on both standard and flat tasks. For network size, standard cascor recruited fewer hidden units than flat cascor on standard teacher problems, but flat networks generally had about half as many connection weights as standard networks had. In sum, flat cascor may learn more efficiently because there are fewer connections to train. We further discuss implications of cascaded weights in section 4.4.

In Prechelt's (1997) work, the average number of recruits for regression (i.e. function approximation) problems was 8 (min=0, max=92), which corresponds to problems of low complexity in our task. Some of the PROBEN1 tasks were actually linearly separable, something that cascor can readily detect by not needing any recruits. We found no differences in student network size and training efficiency when task complexity was that low. This is compatible with Prechelt's results on PROBEN1 real-world problems where he found no performance difference on most of these tasks.

Furthermore, Prechelt used a non-standard threshold (alpha value) for detecting statistical significance. While we used $\alpha = 0.05$ (and Bonferonni adjusted alpha levels when appropriate), Prechelt used a more liberal value of 0.1, so more results could turn out significant in his study simply due to random effects. In fact, some of his results are contradictory: different architectures perform significantly better on the same task depending on the version used, i.e. how data were split into train and test sets. Such contradictions suggest those results could be false positives. Therefore, we can suspect that there might be even fewer actual (practical) differences between variants of cascor on real-world problems than what Prechelt reported.

Other factors may also interact with network topology to determine cascor performance. For example, Phatak and Koren's (1994) results suggest an interaction between connectivity and sample size: standard cascor may construct smaller networks that train faster than networks of restricted depth when there are many training patterns compared to the number of inputs and outputs. However, restricting cascor depth may result in an overall better performance when less data are available for a given number of inputs and outputs. Unfortunately, none of those differences were tested for statistical significance, so their conclusions are uncertain. Furthermore, because small and large sample size data were drawn from different problems, problem characteristics may be confounded with sample size.

Our simulations, based on very large sample sizes, may support a conclusion opposite to Phatak and Koren's (1994) because flat cascor built networks with fewer connection weights that trained with less computational cost. With its strictly controlled bias and uniform tasks across conditions, our teacher-student method could be used to systematically investigate other issues, such as whether cascor connectivity interacts with the amount of training data.

In short, the choice of cascor architecture might have little importance for many real-world problems (see section 4.4 for discussion on the usefulness of cascading weights). However, by contrast with many real-world problems, our artificial task is complex enough to discriminate subtle architectural effects on performance.

4.3 Differences with real-world problems

In what ways does our student-teacher approximation task differ from real-world problems? First, as argued in the previous section, real-world problems tend to be simple and even sometimes linearly separable whereas the complexity of our task was manipulated from simple (compatible with real-world problems) to much more complex.

Second, we were able to generate as much data as we needed because this is an artificial task. In contrast, real-world problems often have much less data because such data can be costly or difficult to obtain.

Third, we argued that if neural networks have more connections to train than there are training patterns available, they may be too powerful and overfit data. Consequently, neural networks trained on real-world tasks may generalize more poorly than neural networks trained on our problem because real-world problems often have little data available.

Fourth, as described earlier, biases in our tasks are systematically controlled. Target values exactly represent the underlying process to approximate. In contrast, real-world data can suffer from unknown biases, have measurement errors, and other sources of noise.

Finally, although our problem used 6 inputs, which is typical of many real-world tasks, some real-world problems have more inputs.

4.4 Why use cascading weights?

Our results raise questions about the utility of cascading weights, especially because we found no difference in generalization between standard and flat students. At lower, similar to real-life problem complexities, using cascaded weights did not do any harm in our experiments but neither did it provide any apparent benefit.

At higher task complexity, our results suggest that cascading weights may result in larger networks that train less efficiently. Compared to flat student networks, standard networks have extra connections, in the form of cascaded weights between hidden units. If these extra, cascaded weights do some useful work, standard networks may well require fewer recruits than do flat networks as, for example, on the standard-teacher tasks used here. However, some of those extra, cascaded weights seem not to be useful, resulting in fewer connection weights for flat networks than standard networks. Also, the extra connections in standard student networks increase training effort.

Perhaps cascaded weights are more useful on problems in which later, more subtle representations can build on earlier and simpler representations. It is possible that the random functions generated by our teacher networks did not have this characteristic.

Baluja and Fahlman (1994) proposed an elegant solution to the problem of having to choose whether or not to cascade weights. In their extension to cascor called Sibling/Descendant Cascade-Correlation (SDCC), the learning algorithm is modified so that the pool of recruits contains both units to be installed on the current top-most hidden layer (sibling units) and units to be installed on a new hidden layer (descendant units). This solves the problem of network depth by allowing sibling and descendant units to directly compete during each recruitment. Standard and flat cascor network architectures can be seen as two limit cases on a continuum of cascor architectures where the number of hidden units per hidden layer varies. In standard cascor, each unit is installed on a different layer, whereas in flat cascor all units are installed onto a single layer. Optimal solutions for specific tasks might require varying ratios of hidden units per hidden layers, so in most cases, the SDCC algorithm can be expected to recruit a mixture of unit types. SDCC also might be a more biologically plausible model than flat or standard cascor. Indeed, its ability to generate networks of moderate depth might be better suited to modelling the limited layers of human cortex.

As noted, Phatak and Koren (1994) proposed another method for limiting network depth, without completely flattening the network. The caveat of their method is that the number of hidden units per hidden layer has to be manually determined. Unfortunately, this undermines cascor's ability to automatically create network topology, and reintroduces designer expertise and trial-and-error experimentation into the process of network design.

We suggest that further research is necessary to determine the utility of cascading weights in cascor networks. In this paper, we investigated two extreme connection schemes, one fully cascaded (standard) and another completely flat. Further research could explore more flexible connectivity schemes such as SDCC. In addition, pruning algorithms (e.g. Thivierge *et al.* 2003) could be used to remove surplus cascading weights, and result in size and training performances closer to that of flat cascor, while keeping the benefits of useful cascading weights in building powerful, cascaded feature detectors.

4.5 Other follow-up experiments

Another possible follow-up experiment would be to compare cascor with backpropagation neural network architectures. Getting a meaningful comparison with backpropagation is not straightforward because backpropagation architectures are designed by the experimenter, and therefore there are potentially unlimited numbers of architectures to compare with. A careful and systematic approach would be necessary to study performance comparison. Based on psychology simulations in which standard cascor or SDCC is compared to backpropagation on the same phenomena (Shultz 2006), we expect that cascor would learn and generalize better than backpropagation.

We found that the student-teacher task technique was powerful for comparing two variants of cascor. We believe this approach could be similarly useful for comparing machine learning algorithms and techniques, with bias and complexity controlled and large areas of problem space explored.

In conclusion, the choice of standard or flat cascor does not seem to be crucial in practice because differences in network size and cost emerge only for high complexity tasks. Our results contradict the common criticism that standard cascor does not generalize well – we found no generalisation differences due to cascading weights at any level of complexity.

5 Acknowledgements

We thank Geoffrey Hinton for providing some seminal ideas for this project, and for his feedback and comments. We also thank François Rivest for ideas and technical advice, Yoshio Takane for help with statistical analyses, and Kristin Laurin for proofreading an earlier version of this document. The research was supported by a McGill Major scholarship to F. D. and a grant to T. R. S. from the Natural Sciences and Engineering Research Council of Canada.

6 References

A. Adams and S. Waugh, “Function evaluation and the cascade-correlation architecture”, in *The IEEE International Conference on Neural Networks*, 1995, pp. 942-946.

S. Amari, N. Murata, K. R. Muller, M. Finke and H. H. Yang, “Asymptotic statistical theory of overtraining and cross-validation”, *IEEE Transactions on Neural Networks*, 8(5), pp. 985-996, 1997.

P. T. Baffes and J. M. Zell, “Growing layers of perceptrons: Introducing the extentron algorithm”, *Proceeding of the International Joint Conference on Neural Networks 2,* 1992, pp. 392-397.

S. Baluja and S. E. Fahlman, “Reducing network depth in the cascade-correlation”, Technical report CMU-CS-94-209, Pittsburgh, Carnegie Mellon University, 1994.

D. Buckingham and T. R. Shultz, “The developmental course of distance, time, and velocity concepts: A generative connectionist model”, *Journal of Cognition and Development,* 1, pp. 305-345, 2000.

D. C. Donderi, “An information theory analysis of visual complexity and dissimilarity”, *Perception,* 35(6), pp. 823-835, 2006.

D. C. Donderi and S. McFadden, “Compressed file length predicts search time and errors on visual displays”, *Displays,* 26, pp. 71-78, 2005.

S. Draghici, “Improving the speed of some constructive algorithms by using a locking detection mechanism”, *Neural Network World,* 6(4), pp. 563-575, 1996.

S. Draghici, “The constraint based decomposition (CBD) training architecture”, *Neural Networks,* 14, pp. 527-550, 2001.

S. E. Fahlman, “Faster-learning variations on back-propagation: An empirical study”, in *Proceedings of the 1988 Connectionist Models Summer School*. T. J. Sejnowski, G. E. Hinton and D. S. Touretzky, Ed., San Mateo, CA: Morgan Kaufmann, 1988.

S. E. Fahlman and C. Lebiere, “The cascade-correlation learning architecture”, in *Advances in neural information processing systems 2,* D. S. Touretzky, Ed., Los Altos, CA: Morgan Kaufmann, 1990, pp. 524-532.

M. Frean, “The upstart algorithm: a method for constructing and training feedforward neural networks”, *Neural Computation*, 2(2), pp. 198-209, 1990.

L. K. Hansen and M. W. Pedersen, “Controlled growth of cascade-correlation nets”, in *Proceedings of ICANN*, 1994, pp. 797- 800.

J. Hwang, S. You, S. Lay and I. Jou, "The cascade-correlation learning: A projection pursuit learning perspective", *IEEE Transactions in Neural Networks,* 7, pp. 278-289, 1996.

H. Klagges and M. Soegtrop, “Limited fan-in random wired cascade-correlation”, *Neuroprose archive*, Munich, IBM Research Division, Physics Group, 1992.

J. J. T. Lahnajarvi, M. Lehtokangas and J. Saarinen, “Evaluation of constructive neural networks with cascaded architectures”, *Neurocomputing,* 48(1-4), pp. 573-607, 2002.

S. A. Lawrence, A. Back, A. C. Tsoi and C. L. Giles, "On the distribution of performance from multiple neural network trials", *IEEE Trans. on Neural Networks*, 8(6), pp. 1507-1517, 1997.

S. Lawrence, C. L. Giles and A. C. Tsoi , "Lessons in neural network training: Overfitting may be harder than expected", in *Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97,* 1997, pp. 540-545.

E. Littmann and H. Ritter, "Generalization abilities of cascade network architectures", in *Advances in Neural Information Processing System 5*, S. J. Hanson, J. Cowan and C. L. Giles, Ed., Morgan Kaufmann, 1993, pp. 188-195.

M. Mezard and J. Nadal, "Learning feedforward networks: The tiling algorithm", *J. Phys. A: Math. Gen.*, 22, pp. 2191-2203, 1989.

R. Parekh, J. Yang and V. Honavar, "Constructive neural network learning algorithms for pattern classification", *IEEE Trans. Neural networks,* 11(2), pp. 436-451, 2000.

H. Park, N. Murata and S. Amari, "Improving generalization performance of natural gradient learning using optimized regularization by NIC", *Neural Computation*, 16, pp. 355-382, 2004.

D. S. Phatak and I. Koren, "Connectivity and performance tradeoffs in the cascade correlation learning architecture", *IEEE Transactions on Neural Networks,* 5(6), pp. 930-935, 1994.

L. Prechelt, "PROBEN1 - A set of neural network benchmark problems and benchmarking rules", Technical report, Karlsruhe, Germany (1994).

L. Prechelt, "Investigation of the CasCor Family of Learning Algorithms", *Neural Networks*, 10(5), pp. 885-896, 1997.

M. Y. Rafiq, G. Bugmann, and D. J Easterbrook, "Neural network design for engineering applications", *Computers and Structures*, 79, pp. 1541-1552, 2001.

S. R. Quartz, "The constructivist brain", *Trends in Cognitive Sciences,* 3, pp. 48-57, 1999.

S. G. Romaniuk and L. O. Hall, "Divide and conquer neural networks", *Neural Networks,* 6, pp. 1105-1116, 1993.

D. Saad (Ed.), *On-line learning in neural networks,* Cambridge: Cambridge University Press, 1999.

T. R. Shultz, *Computational developmental psychology*. Cambridge, MA: MIT Press, 2003.

T. R. Shultz, Constructive learning in the modeling of psychological development. In *Processes of change in brain and cognitive development: Attention and performance XXI,* Y. Munakata and M. H. Johnson, Eds., Oxford: Oxford University Press, 2006, pp. 61-86.

T. R. Shultz, S. P. Mysore, and S. R. Quartz, S. R. Why let networks grow? In *Neuroconstructivism: Perspectives and prospects,* D. Mareschal, S. Sirois, G. Westermann and M. H. Johnson, Eds., Oxford: Oxford University Press, 2007, Vol. 2, pp. 65-98.

T. R. Shultz, D. Mareschal and W. C. Schmidt, "Modeling cognitive development on balance scale phenomena", *Machine Learning,* 16, pp. 57-86, 1994.

T. R. Shultz and F. Rivest, "Knowledge-based cascade-correlation: Using knowledge to speed learning", *Connection Science,* 13, pp. 43-72, 2001.

S. Sjogaard, “A conceptual approach to generalization in dynamic neural networks”. Ph.D. thesis, Computer Science Department, Aarhus University, Denmark (1991).

J. P. Thivierge, F. Rivest, and T. R. Shultz, “A dual-phase technique for pruning constructive networks”, in *Proceedings of the 2003 IEEE International Joint Conference on Neural Networks*, 2003, pp. 559-564.

Y. Takane, Y. Oshima-Takane and T. R. Shultz, “Network analyses: The case of first and second person pronouns”, in *Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics,* 1995, pp. 3594-3599.

S. Waugh and A. Adams, “Connection strategies in cascade-correlation”, in *Proceedings: The Fifth Australian Conference on Neural Networks*, 1994, pp. 1-4.

E. J. Wisniewski, “Prior knowledge and functionally relevant features in concept learning”, *Journal of Experimental Psychology: Learning, Memory, and Cognition,* 21, pp. 449-468, 1995.

Figures and tables

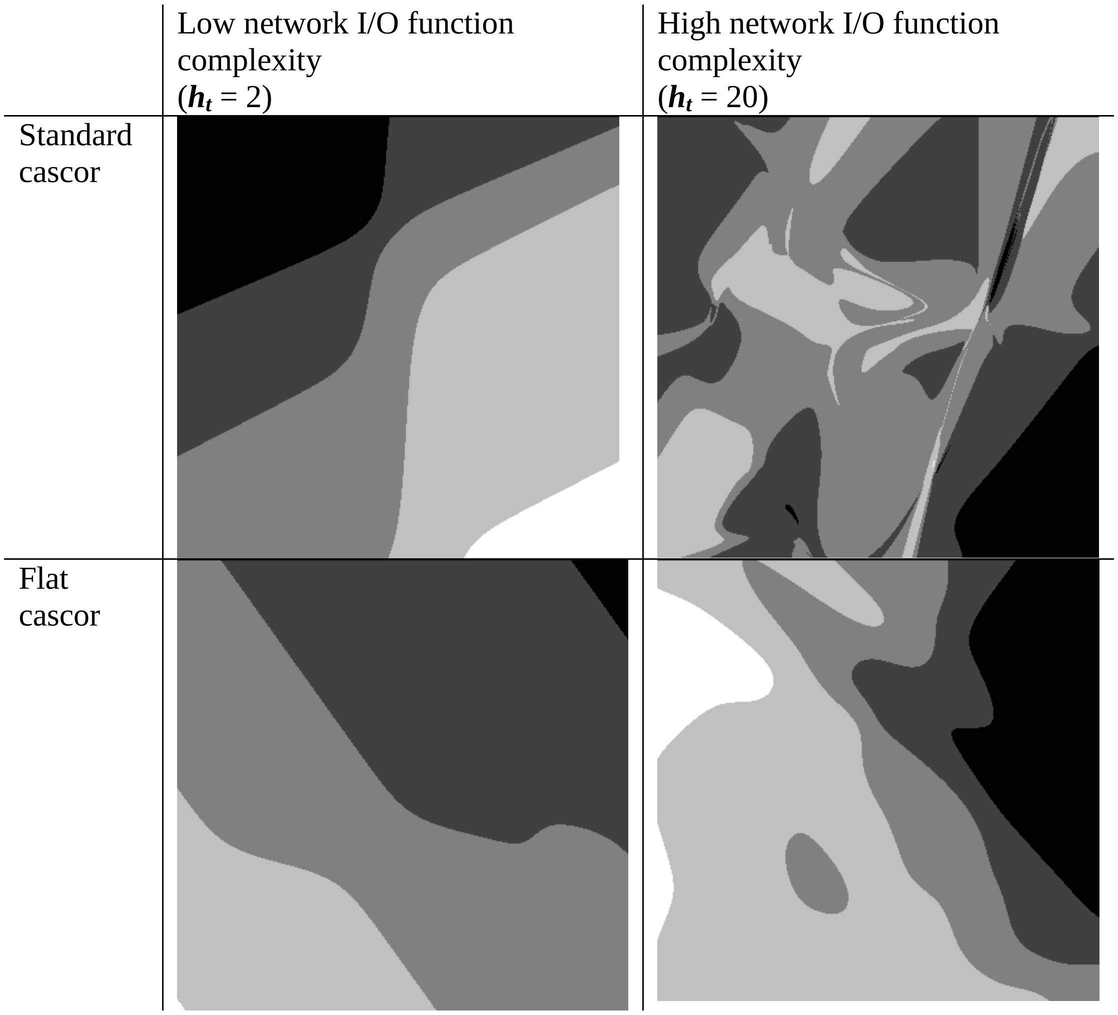

Fig. 1. Examples of low and high complexity network I/O functions generated using standard and flat cascor teacher networks over the $[-1,1]^2$ plane. Gray levels encode output values (z).

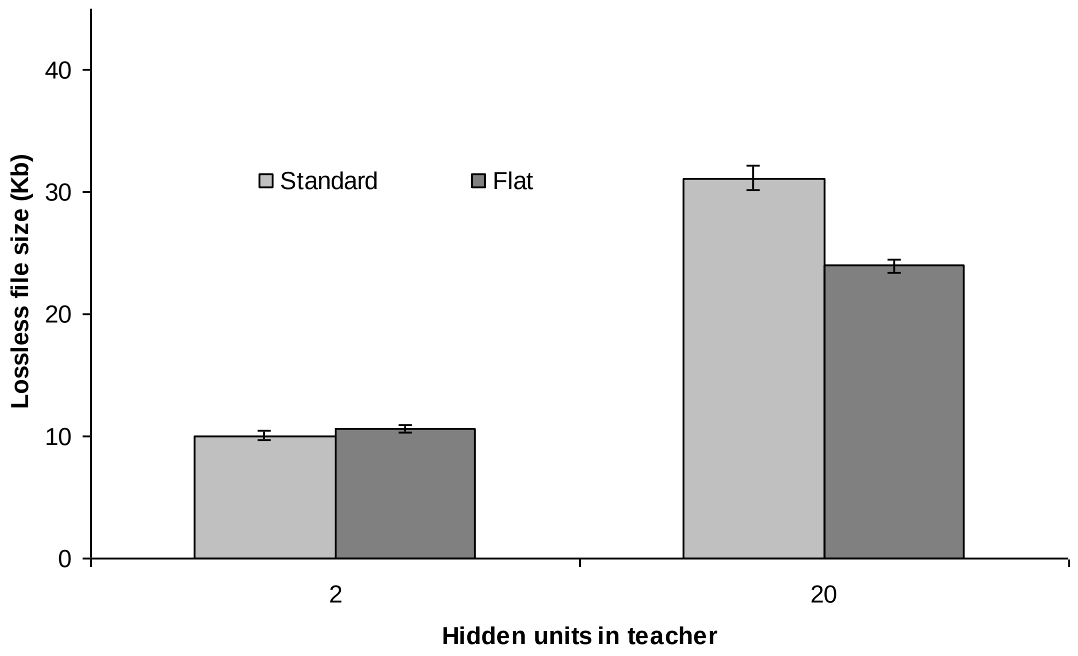

Fig. 2 Mean function complexity and SE bars measured using lossless compressed file size as a function of hidden unit count in the teacher network.

Table 1 – Calculation of number of connection weights

Network type	Number of weights after training
Flat	(inputs + 1) x (outputs + hidden) + hidden x outputs
Standard	0.5 x hidden x (hidden + 1) + inputs x hidden + outputs x (inputs + hidden + 1)

Table 2 - Calculation of computational cost for training

Network type	Phase	Weight changes
Flat	Input	Candidates x (inputs + 1) x Epochs $_{\text{input phase}}$
Standard	Input	Candidates x ((inputs + 1) + (hidden – 1)) x Epochs $_{\text{output phase}}$
Flat and standard	Output	Outputs x ((inputs + 1) + hidden) x Epochs $_{\text{output phase}}$

Table 3 - Simulation Parameters Summary

Parameter	Value
Inputs count	6
Outputs count	1
Input transfer function type	Linear
Hidden and output transfer function type	Asigmoid (range [0,1])
Train and test sample size	21609
Maximum hidden unit recruitments allowed	10 x number of hidden units in teacher network
Weight range	[-1,1]

Table 4 - Learning Parameters Summary (QuickProp)

Parameter	Output Phase	Input Phase
Learning rate	0.175	1.0
Decay	0.0002	0.0
Maximum growth factor	2.0	2.0
Max epochs	100	100
Change threshold	0.01	0.03
Patience	8	8
Score threshold	0.05	NA

Table 5 – GLM Statistical significance of univariate main effects ('-' when $p > 0.05$). Degrees of freedom for all F tests in this table are 1, 380.

Factors	Dependent measures					
	Recruits	Weights	Epochs	Cost	Train error	Test error
Student	-	F = 15, $p < 0.001$	-	F = 15, $p < 0.001$	-	-
h_t	F = 16, $p < 0.001$	F = 16, $p < 0.001$	F = 16, $p < 0.001$	F = 15, $p < 0.001$	-	-
Teacher	F = 10, $p = 0.002$	F = 9, $p = 0.002$	F = 7, $p = 0.010$	F = 5, $p = 0.024$	-	-
h_t x student	-	F = 2.1, $p = 0.034$	-	-	-	-
Student x Teacher	F = 6, $p = 0.015$	-	F = 4.0, $p = 0.045$	-	-	-
h_t x teacher	-	-	-	-	-	-
Student x h_t x teacher	-	-	-	-	-	-

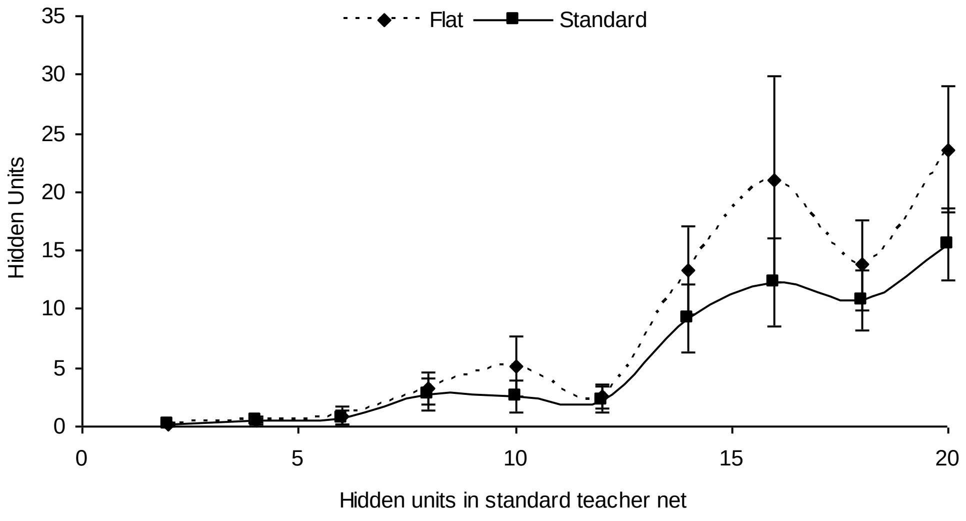

Fig. 3. Recruited hidden units in student networks as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by standard teacher networks.

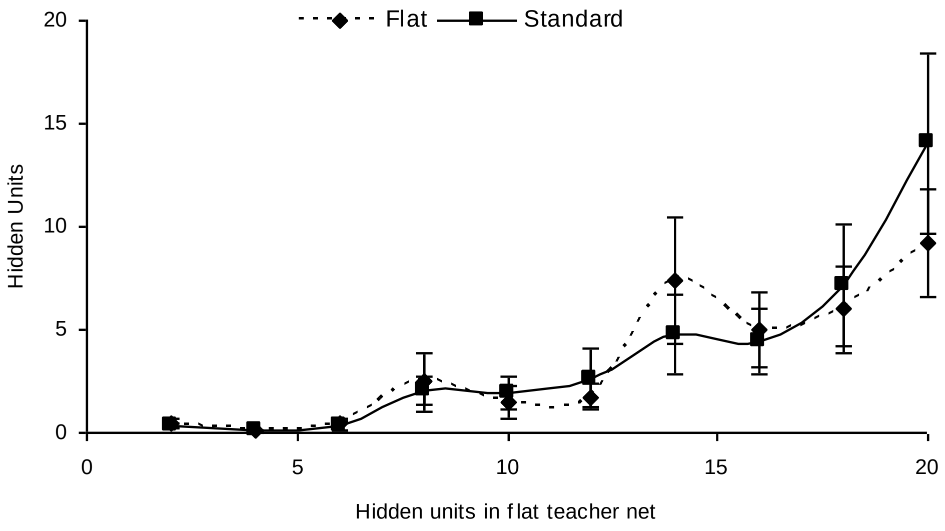

Fig. 4. Recruited hidden units in student networks as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by flat teacher networks. Note that vertical scales are different in Fig. 3 and Fig. 4 .

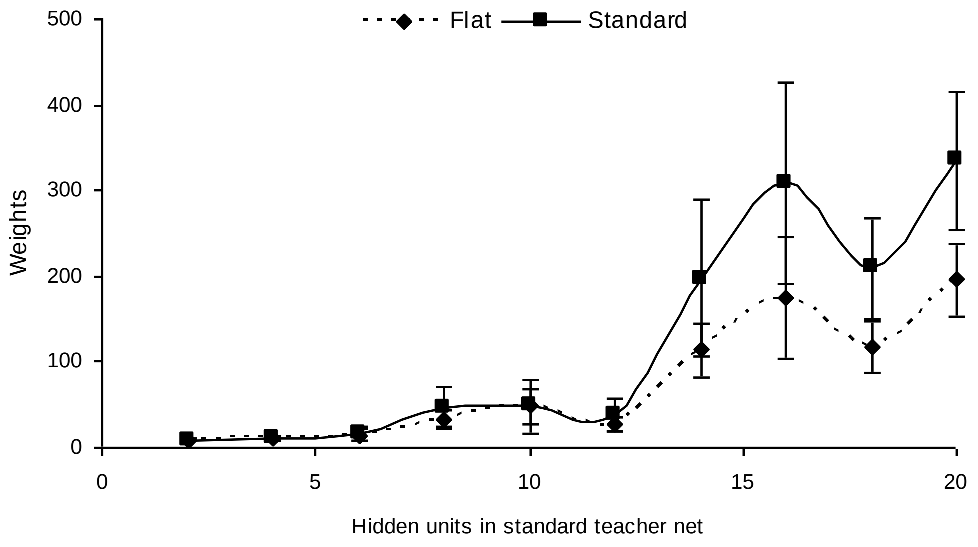

Fig. 5. Mean number of connection weights and SE bars as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by standard teacher networks.

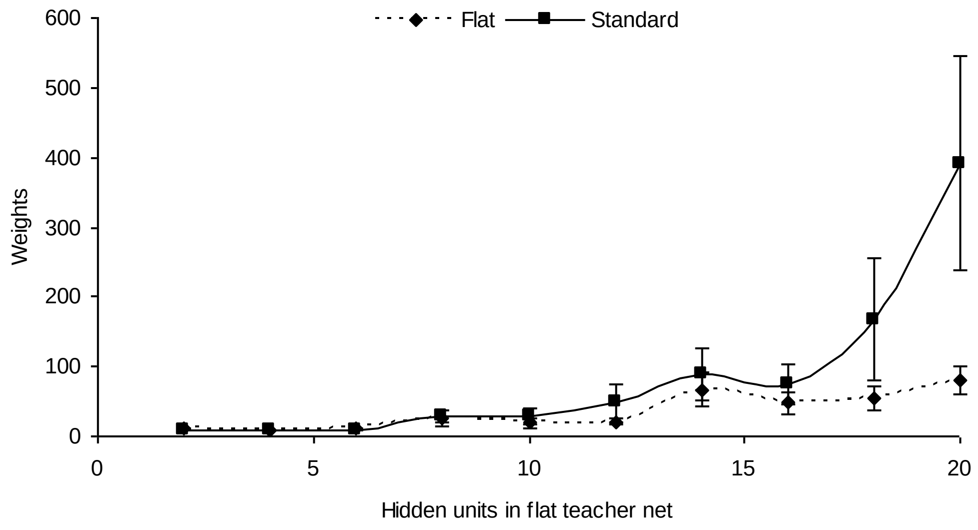

Fig. 6. Number of connection weights and SE bars as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by flat teacher networks.

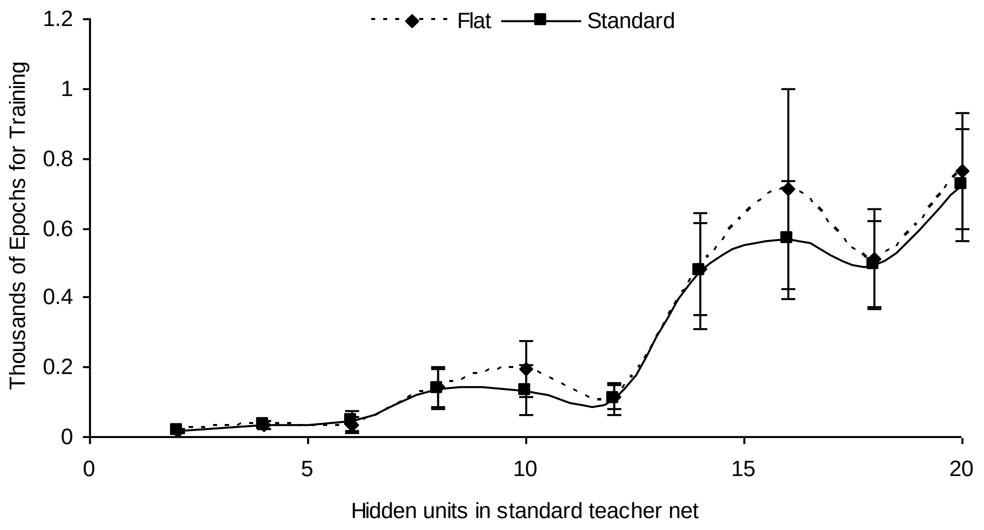

Fig. 7. Number of epochs required for training and SE bars as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by standard teacher networks.

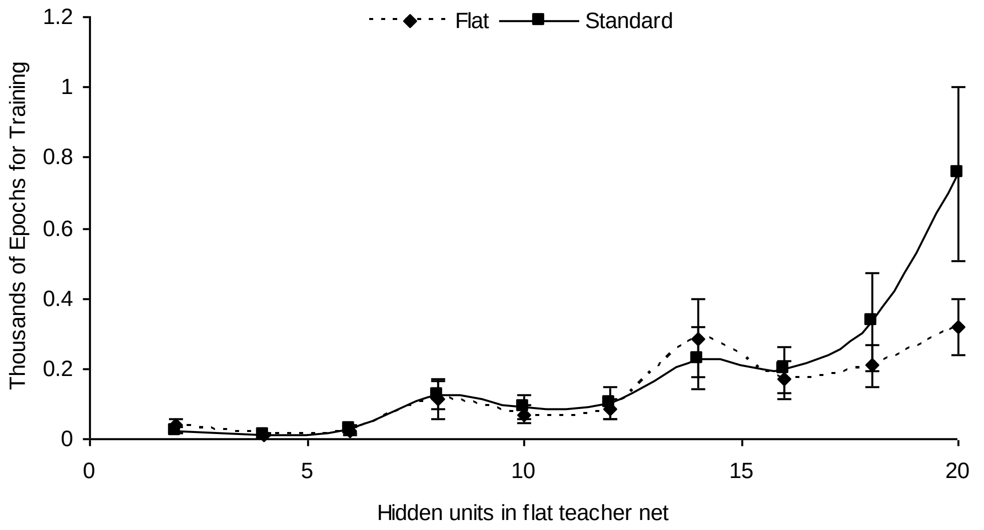

Fig. 8. Number of epochs required for training and SE bars as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by flat teacher networks.

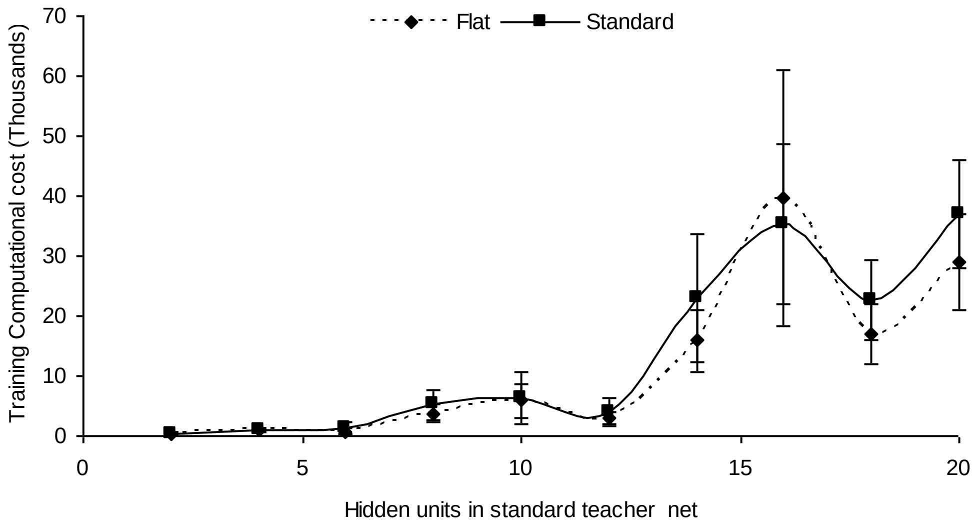

Fig. 9. Computational cost for training and SE bars as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by standard teacher networks.

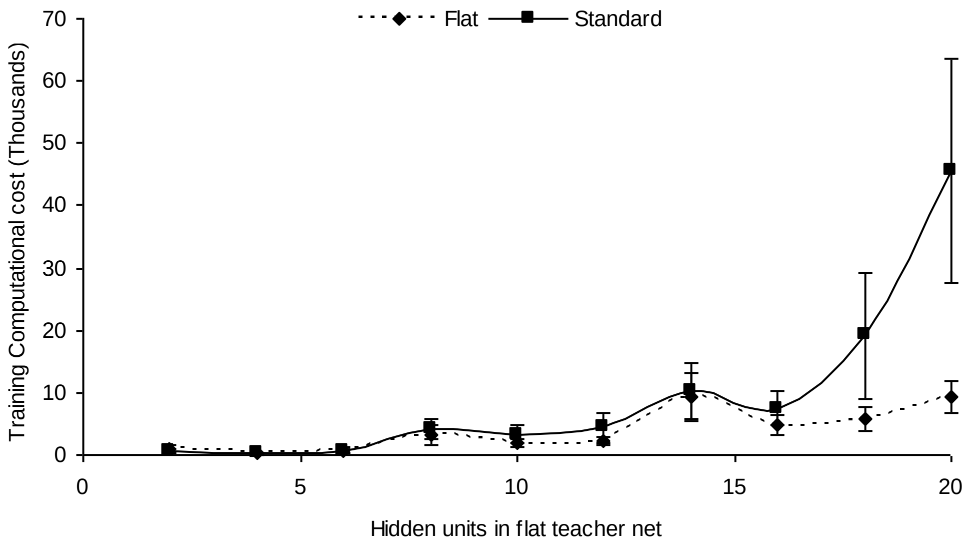

Fig. 10. Computational cost for training and SE bars as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by flat teacher networks.

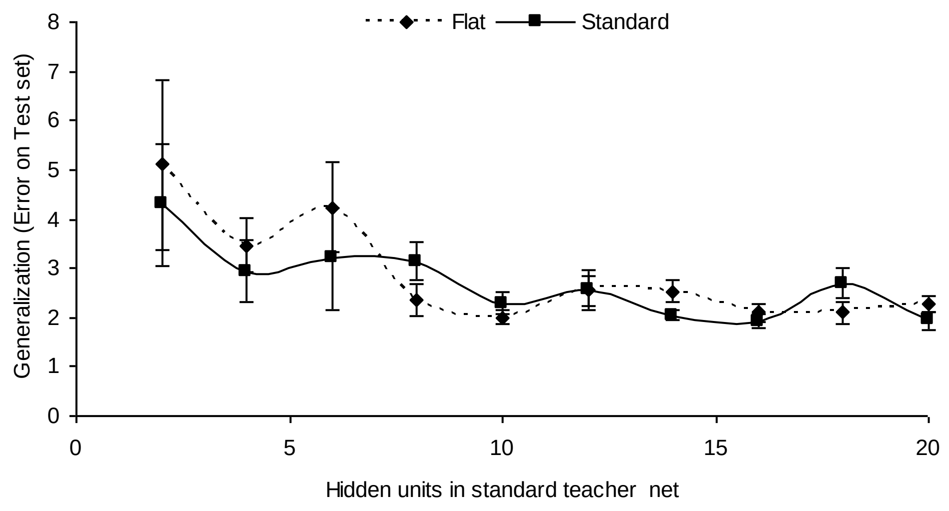

Fig. 11. Generalization (Error on test set after training) and SE bars as a function of the number of hidden units in the teacher network and type of student network. Training samples were generated by standard teacher networks.

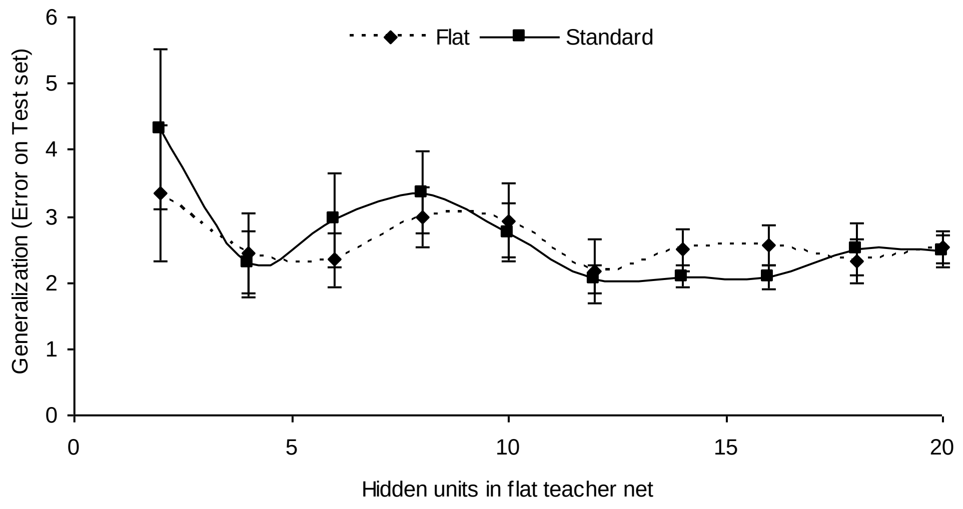

Fig. 12. Generalization (Error on test set after training) and SE bars as a function of the number of hidden units in the teacher network and type of student network. Samples were generated by flat teacher networks.