Cascade correlation NNs

- Background
- Grow topology to match problem complexity

What is Cascade Correlation ?

- Cascade-correlation (Cascor) is an architecture and generative, feed-forward, supervised learning algorithm for artificial neural networks.
- Cascor begins with a minimal network, then automatically trains and adds new hidden units one by one creating a multi-layer structure.

Cascor Architecture and learning Algorithm

- Cascor combines two ideas:
  - The first is the cascade architecture, in which hidden units are added only one at a time and do not change after they have been added.
  - The second is the learning algorithm, which creates and installs the new hidden units. For each new hidden unit, the algorithm tries to maximize the magnitude of the correlation between the new unit's output and the residual error signal of the network.

The Algorithm

1. Cascor starts with a minimal network consisting only of an input and an output layer. Both layers are fully connected.
2. Train all the connections ending at an output unit with a usual learning algorithm until the error of the net no longer decreases.
3. Generate the so-called candidate units. Every candidate unit is connected with all input units and with all existing hidden units. Between the pool of candidate units and the output units there are no weights.

The Algorithm

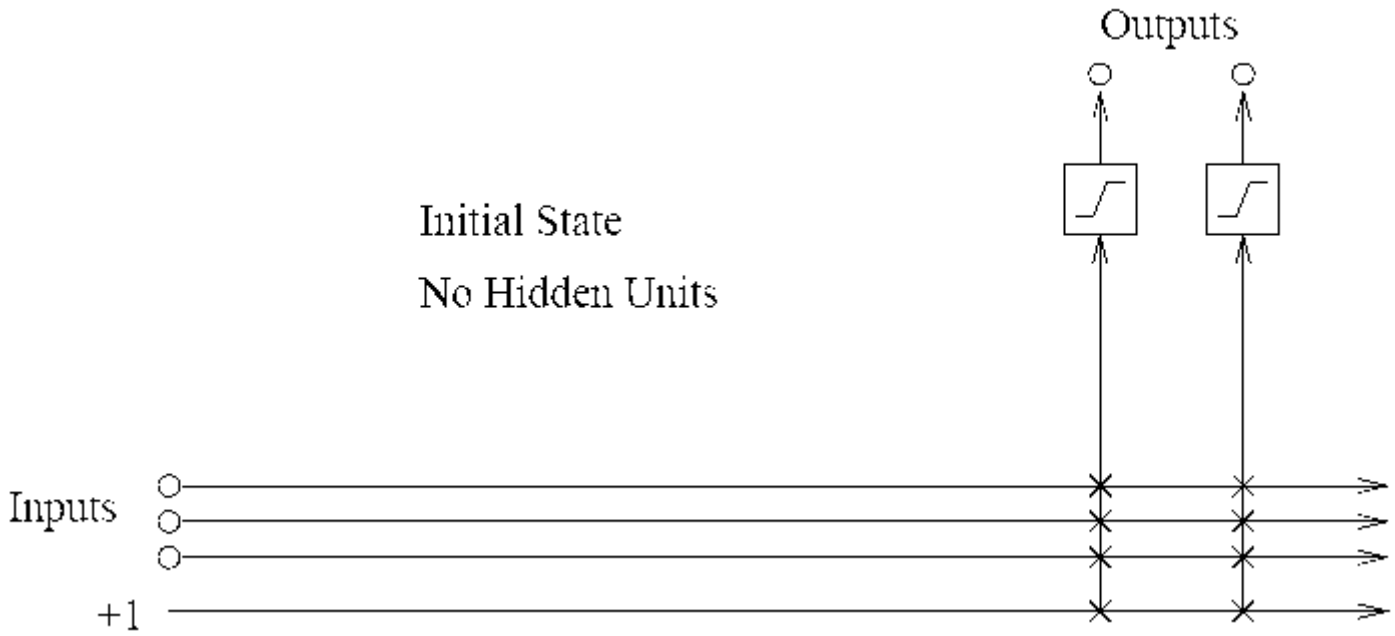
4. Try to maximize the correlation between the activation of the candidate units and the residual error of the net by training all the links leading to a candidate unit. Learning takes place with an ordinary learning algorithm. The training is stopped when the correlation scores no longer improves.
5. Choose the candidate unit with the maximum correlation, freeze its incoming weights and add it to the net.

The Algorithm

5. To change the candidate unit into a hidden unit, generate links between the selected unit and all the output units. Since the weights leading to the new hidden unit are frozen, a new permanent feature detector is obtained. Loop back to step 2.
6. This algorithm is repeated until the overall error of the net falls below a given value

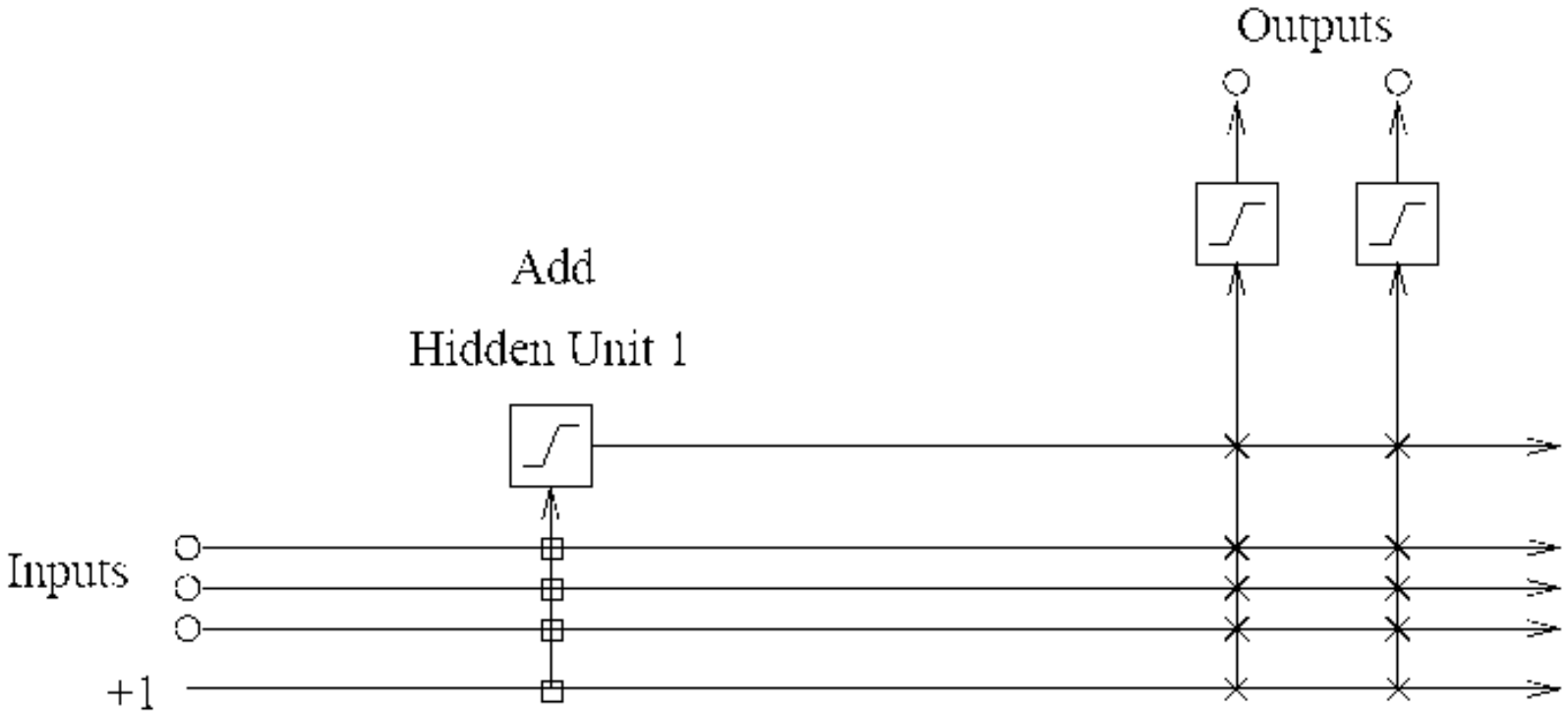
Cascade Correlation

- Automatically finds optimal network structure
- Start with a network with no hidden units
- Grow it by one hidden unit at a time
- Cascade correlation adds units that model the current error in the system



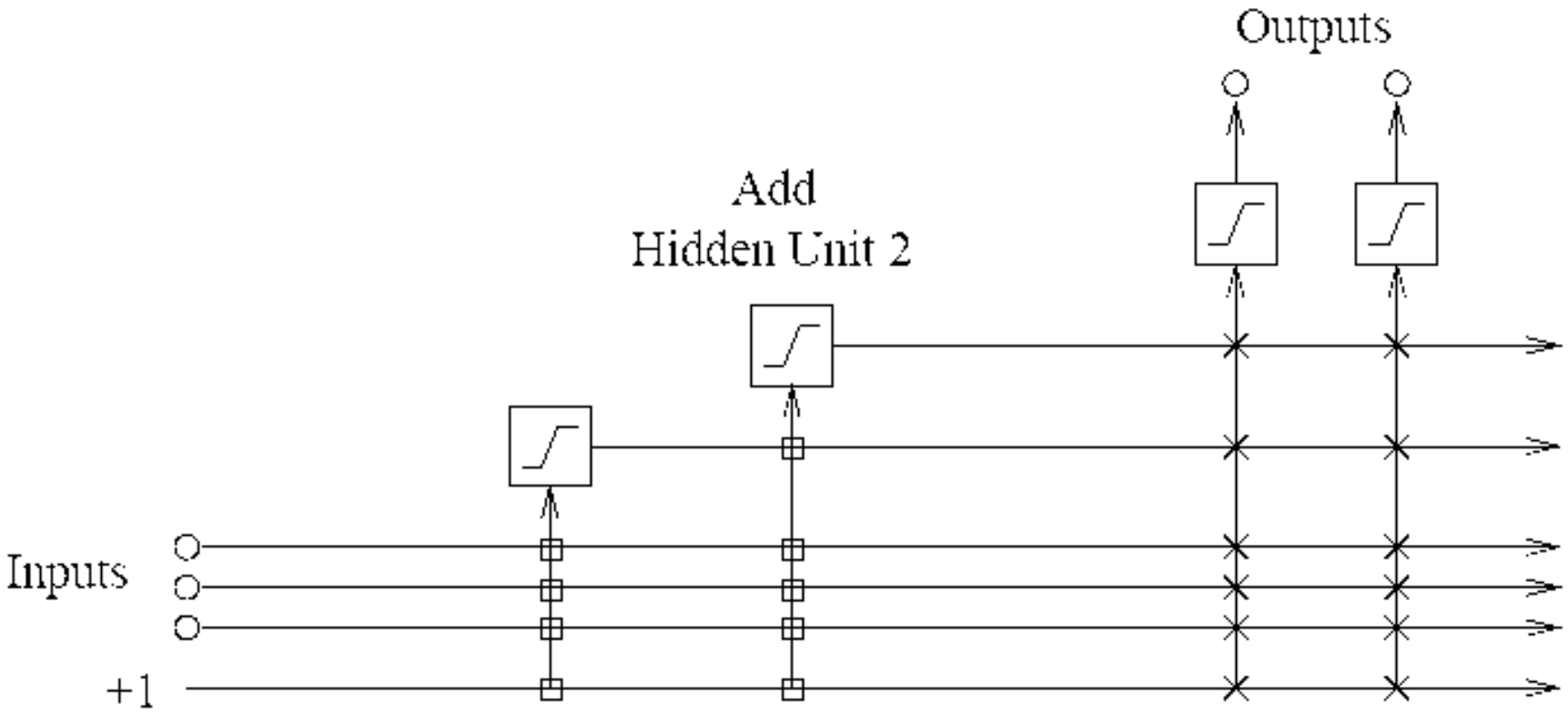
Cascade Correlation

- Freeze the weights of the inputs and pre-existing hidden units to the new hidden unit
- Weights between inputs/hidden units and output units still live
- Repeat cycle of training with modified network until error



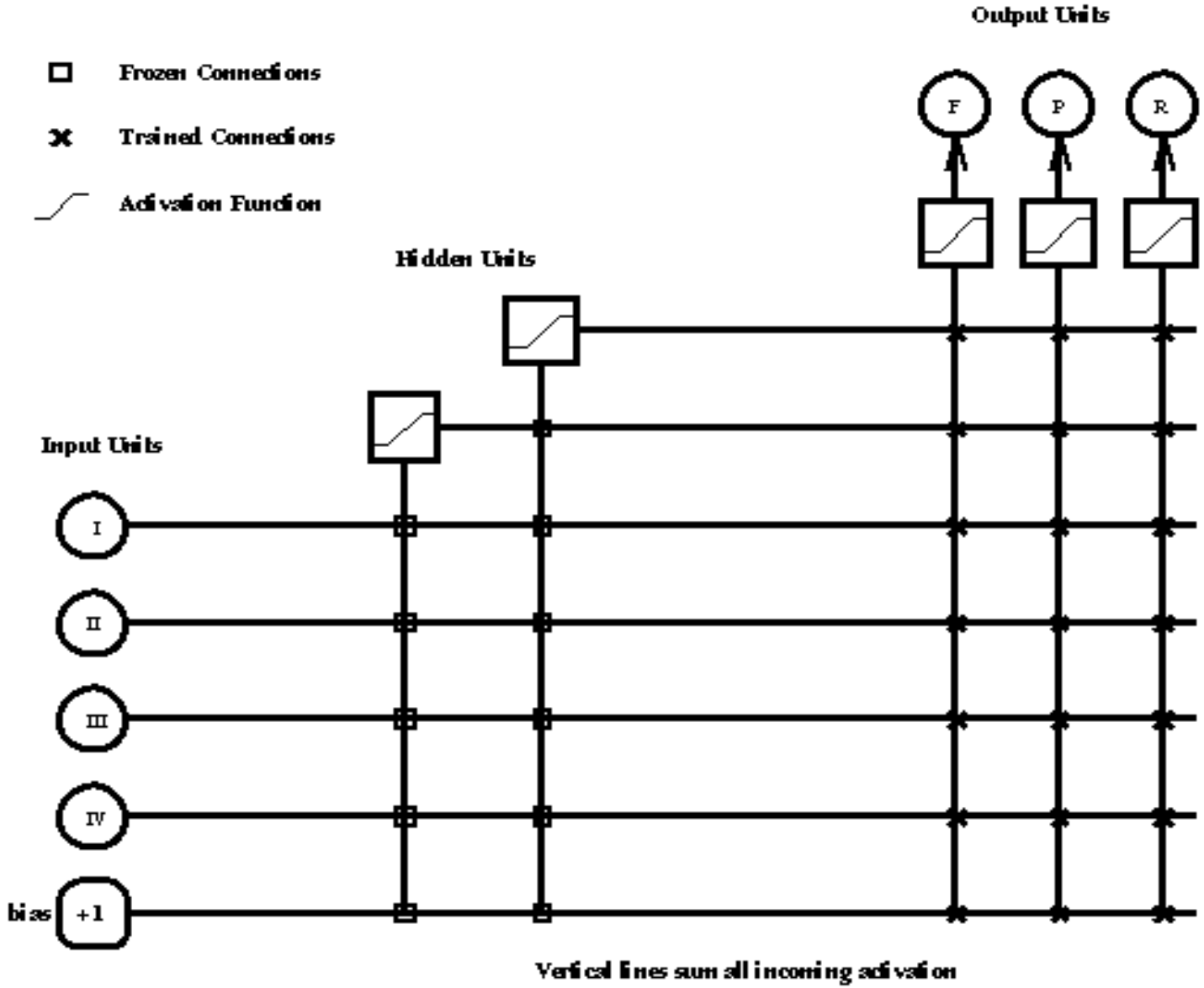
Box = weight frozen, X = weight live

Cascade Correlation next step



Box = weight frozen, X = weight live

A Cascade Correlation Neural Net



Advantages of Cascor

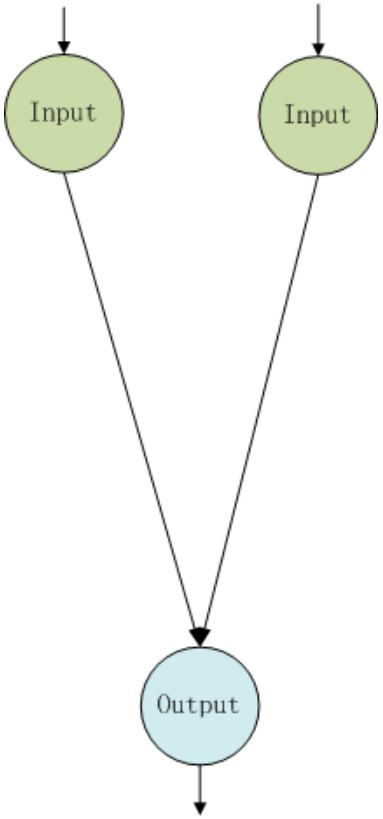
- It learns at least 10 times faster than standard Back-propagation Algorithms.
- The network determines its own size and topologies.
- It is useful for incremental learning in which new information is added to the already trained network.

Cascor algorithm - Fahlman

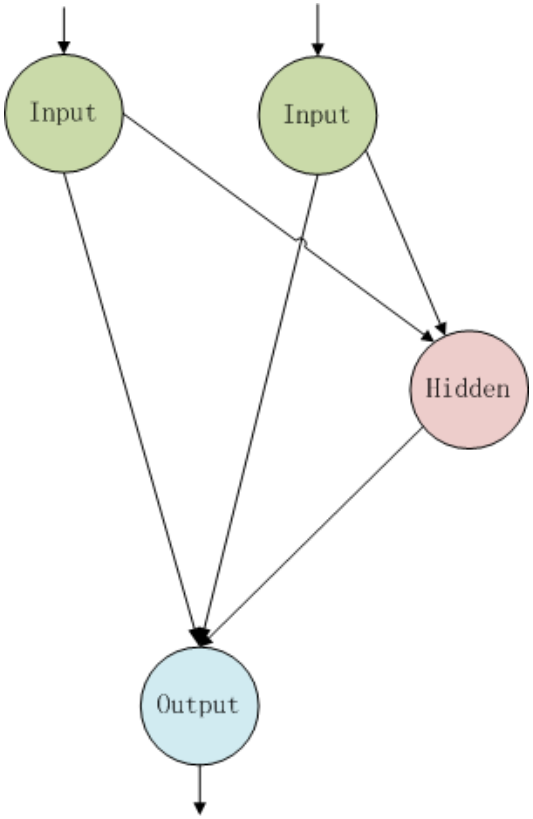
- a single input layer
  - connected directly to the output layer
- neurons are added one at time
  - connected to all previous hidden + input neurons
  - producing a ‘cascade’ network
  - old weights frozen when adding new neurons
  - new neurons come from a ‘candidate pool’
    - maximize the magnitude of the correlation between the new unit’s output and the residual error signal

Cascading topology

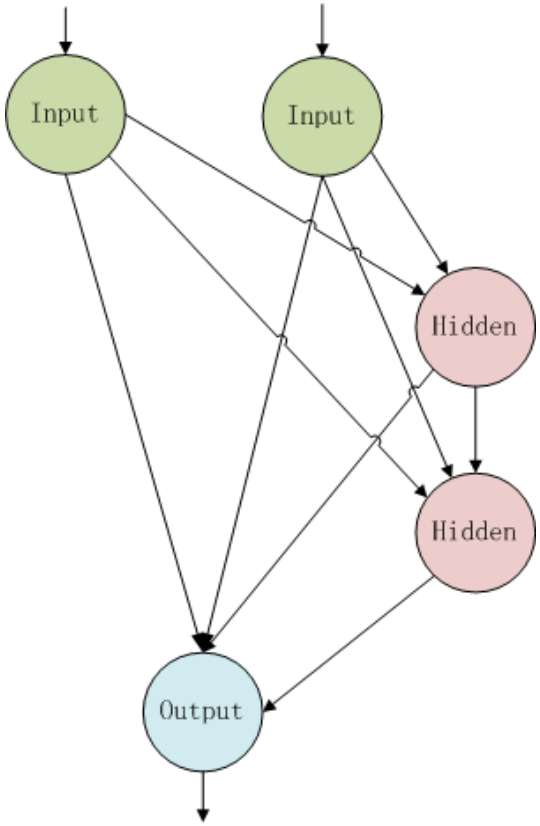
Initial



Stage 1



Stage 2



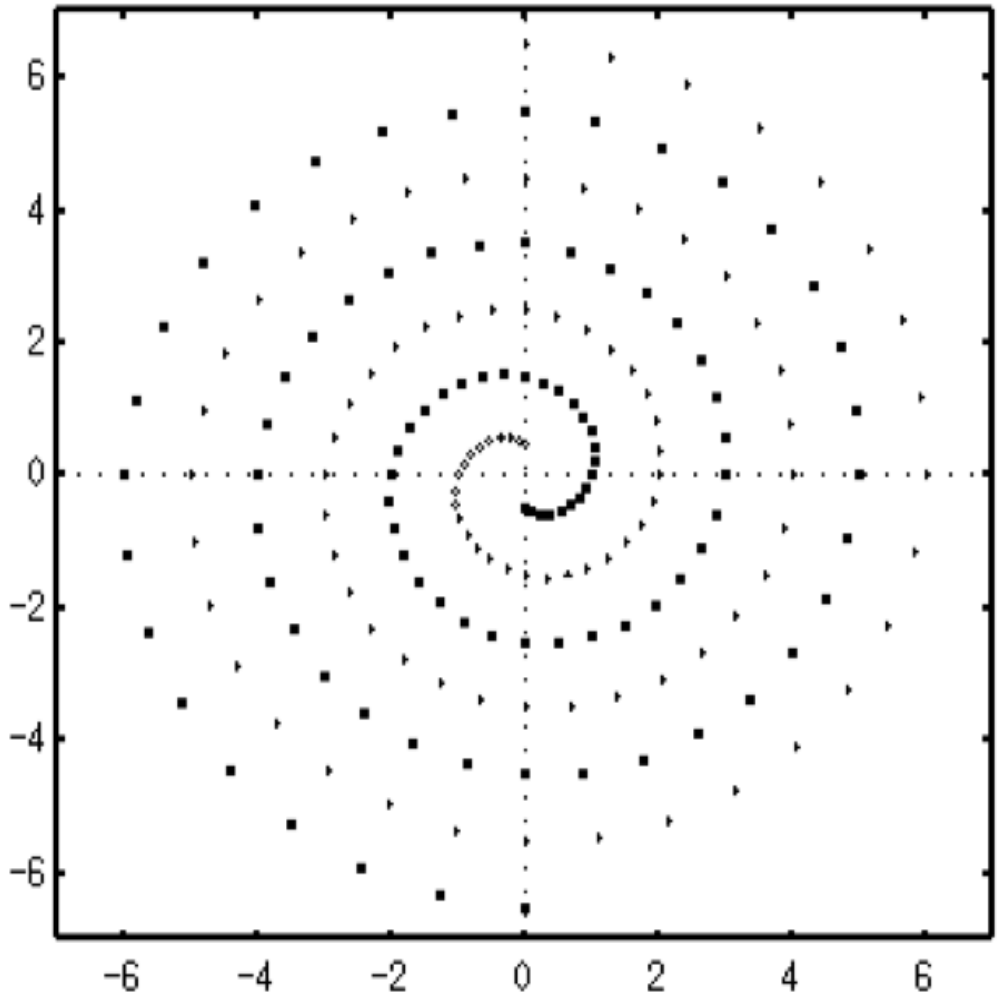
...

Problems with Cascor

- Weight freezing: saves training time BUT can lead to “too large” networks
  - Early hidden neurons can be poor feature detectors
  - Fixed by later neurons
- Use of correlation measure
  - forces the hidden units to saturate, which produces jagged edges in the network outputs

Example: 2 spirals training set

- Interlocked spirals
- Each 97 points for training
- Very difficult for normal back-propagation NNs



Cascor results on 2 spirals

- Best results from 100 runs

14 neurons

17 neurons

18 neurons

The CasPer Algorithm

- A constructive algorithm designed to overcome the bad generalisation property of CasCor.
- CasPer uses a Cascade architecture, like CasCor.
- CasPer uses variation of RPROP, termed **P**rogr**e**ssive RPROP, to train the network. Hence the name **CasPer.**
- CasPer does not use a correlation measure or weight freezing,
  - CasPer uses RPROP to train the whole network.
- CasPer
  - Smaller networks than CasCor
  - Better generalisation than CasCor

The CasPer Algorithm

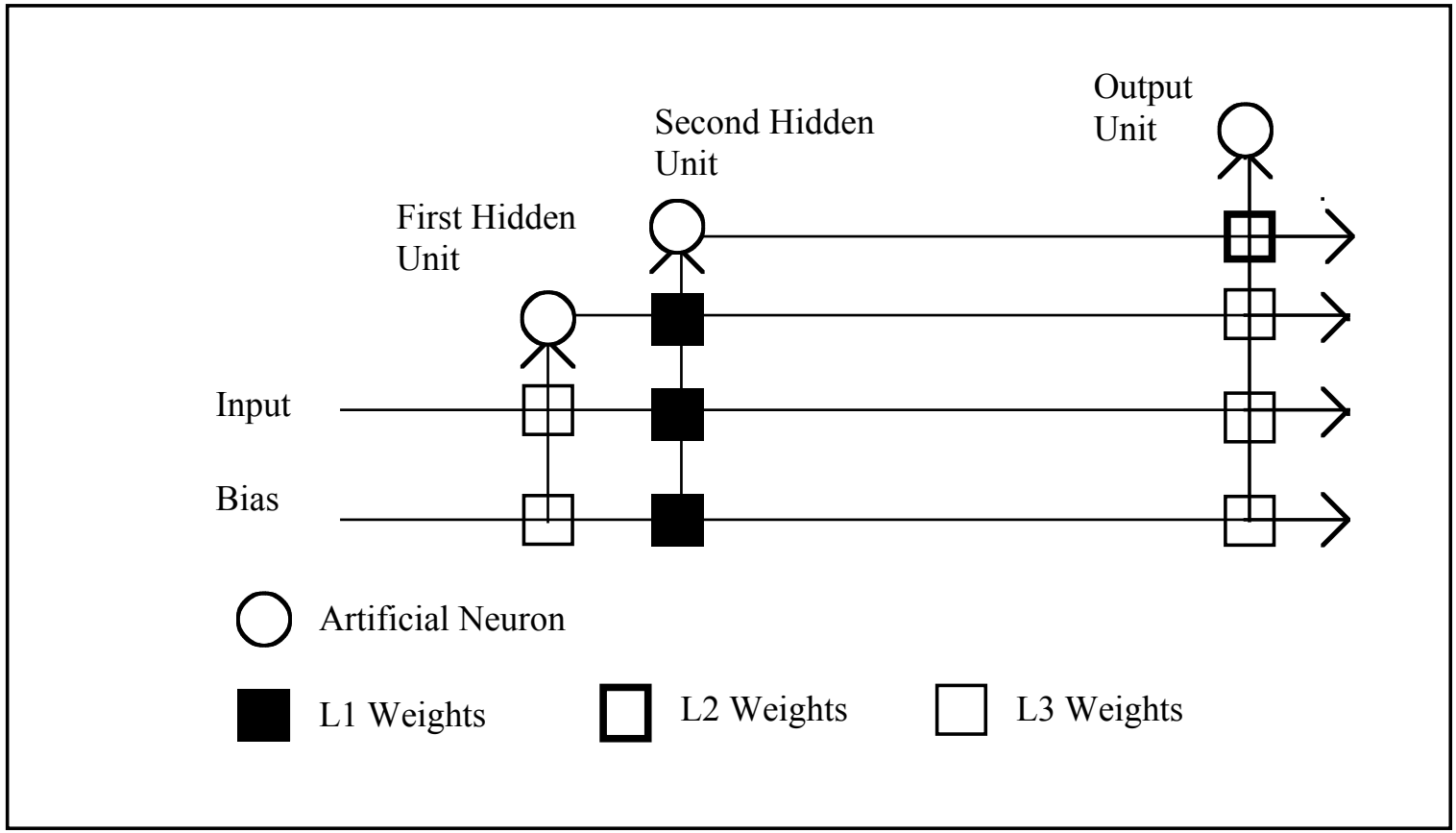
- New neuron addition: initial RPROP learning rates set by weight location.

- 3 regions in network, each with own learning rate:
  - Region L1: Weights connecting to new neuron.
  - Region L2: Weights connected from new neuron to output neurons.
  - Region L3: Remaining weights (all weights connected to and coming from the old hidden and input neurons).

- Relative values for the learning rates are set: L1>>L2>L3.

The CasPer Algorithm

- CasPer Architecture: The second hidden neuron has just been added.
- **Note:** as with CasCor, vertical lines sum incoming inputs.



The CasPer Algorithm

- Motivation for L1 >> L2 > L3
  - initially new hidden neuron learns remaining error with little interference from other hidden neurons.

- No weight freezing: old weights can be modified, albeit at an initially slower rate than the new weights.

- CasPer uses **weight decay** to improve generalisation.

- Weight decay uses Simulated Annealing

$$\delta E/\delta wij = \delta E/\delta wij - D*sign(wij)*wij^2*2^{-T*Hepoch}$$

Hepoch = epochs since addition of last hidden neuron

D = decay parameter

Resilient Backpropagation (Rprop)

- The Rprop algorithm takes a very different approach to improving backpropagation.
- Rprop only uses the **sign** of the gradient, because its size can be a poor and noisy estimator of required weight updates.
- Furthermore, Rprop assumes that different weights need different step sizes for updates, which vary throughout the learning process.

Resilient Backpropagation (Rprop)

- The basic idea is that if the error gradient for a given weight w_{ij} had the same sign in two consecutive epochs, we increase its step size Δ_{ij}, because the weight's optimal value may be far away.
- If, on the other hand, the sign switched, we decrease the step size.
- Weights are always changed by adding or subtracting the current step size, regardless of the absolute value of the gradient.
- This way we do not "get stuck" with extreme weights that are hard to change because of the shallow slope in the sigmoid function.

Resilient Backpropagation (Rprop)

- Formally, the step size update rules are:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^{+} \cdot \Delta_{ij}^{(t-1)}, & \text{if } \dfrac{\partial E^{(t-1)}}{\partial w_{ij}} \cdot \dfrac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ \eta^{-} \cdot \Delta_{ij}^{(t-1)}, & \text{if } \dfrac{\partial E^{(t-1)}}{\partial w_{ij}} \cdot \dfrac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ \Delta_{ij}^{(t-1)}, & \text{otherwise} \end{cases}$$

Empirically, best results were obtained with initial step sizes of 0.1, $\eta^{+}=1.2$, $\eta^{-}=1.2$, $\Delta_{max}=50$, and $\Delta_{min}=10^{-6}$.

Resilient Backpropagation (Rprop)

- Weight updates are then performed as follows:

$$w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}, & \text{if } \dfrac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ +\Delta_{ij}^{(t)}, & \text{if } \dfrac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ 0, & \text{otherwise} \end{cases}$$

It is important to remember that in Rprop the gradient needs to be computed across all samples (per-epoch learning).

Resilient Backpropagation (Rprop)

- The performance of Rprop considerably accelerates backpropagation learning.
- Compared to the standard backpropagation algorithm, Rprop has one advantage:
- Rprop does not require the user to estimate or empirically determine a step size parameter and its change over time.
- Rprop will determine appropriate step size values by itself and can thus be applied “as is” to a variety of problems without significant loss of efficiency.

Casper algorithm

- a single input layer
  - connected directly to the output layer
- neurons are added one at time
  - connected to all previous hidden + input neurons
  - producing a ‘cascade’ network
- Differences from Cascor
  - Weight Decay
  - Weights use different learning rates
    - Adaptive learning rates (RPROP algorithm)
  - Learning rates reset when new neurons added

Casper results on 2 spirals

- Best results from 100 runs

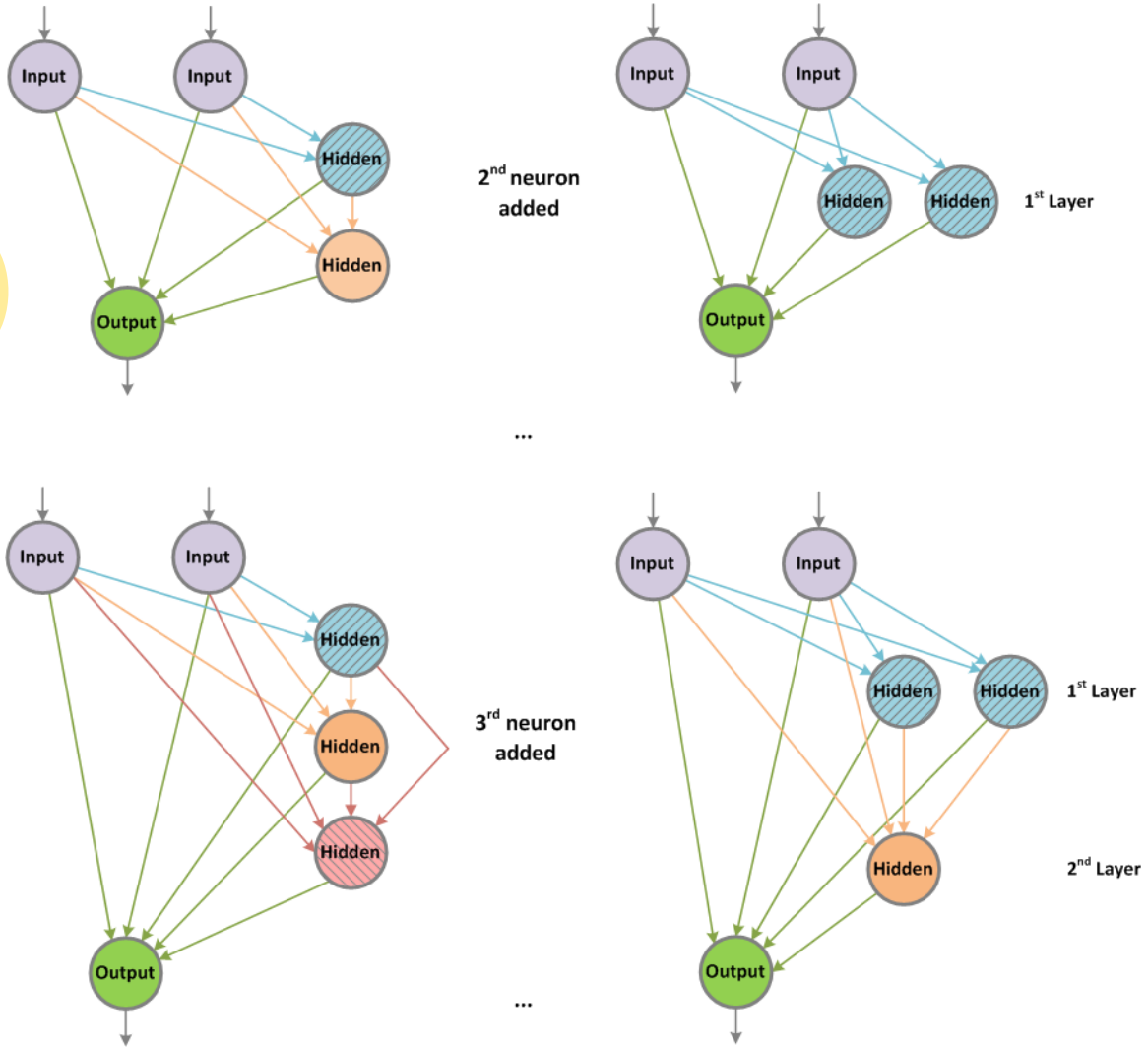
12 neurons

12 neurons

14 neurons

Layered Casper

- new neurons added beside previous ones up to a limit then a new layer neuron is added



Layered Casper on 2 spirals

Fig. 14. The two spirals testing set

Fig. 17. Result of the CasCor algorithn (17 hidden neurons) [3]

Casper

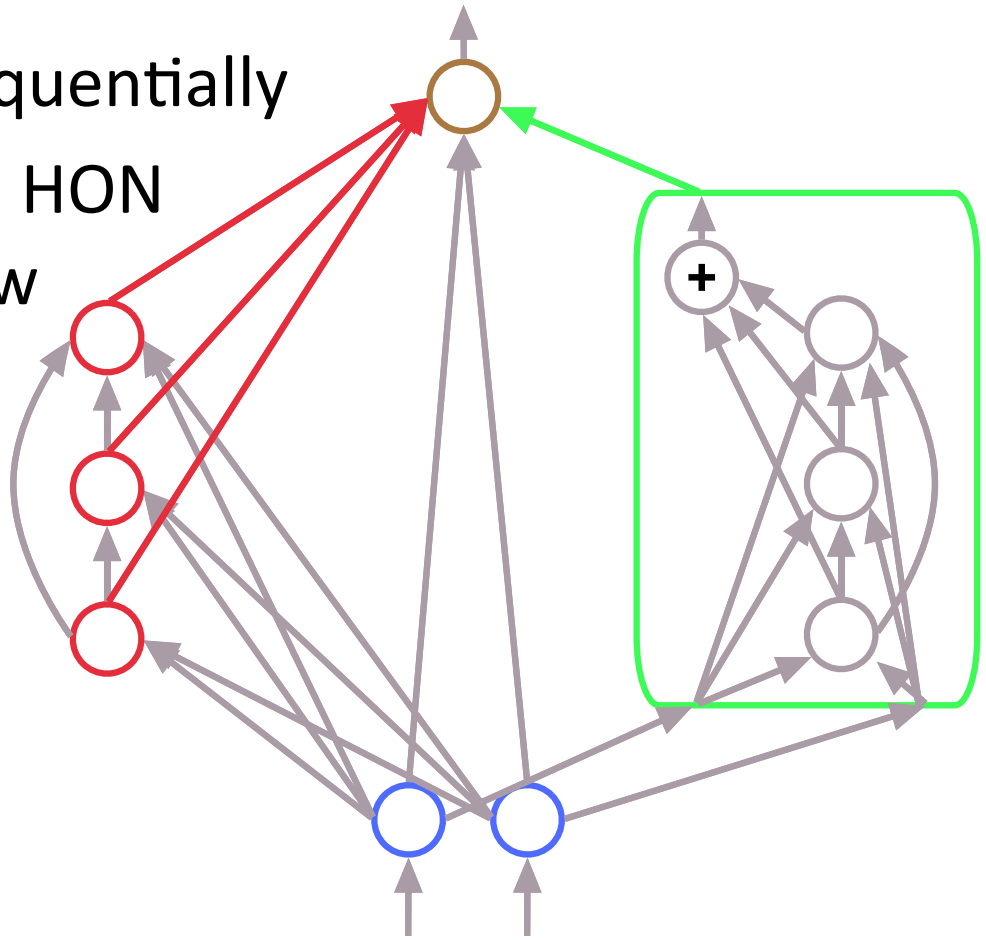
Fig. 16. Result of the CasPer algorithm (12 hidden neurons) [3]

Layered Casper

Fig. 15. Result of the Layered_CasPer algorithm

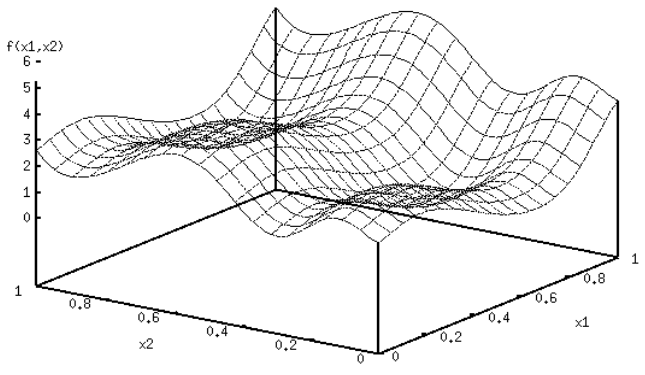
CasPer towers/HON, fan-in

- How is tower/HON constructed?
  - Each HON is built sequentially
  - When the maximum HON size is reached, a new HON is started
- Training Algorithm
  - Same as CasPer

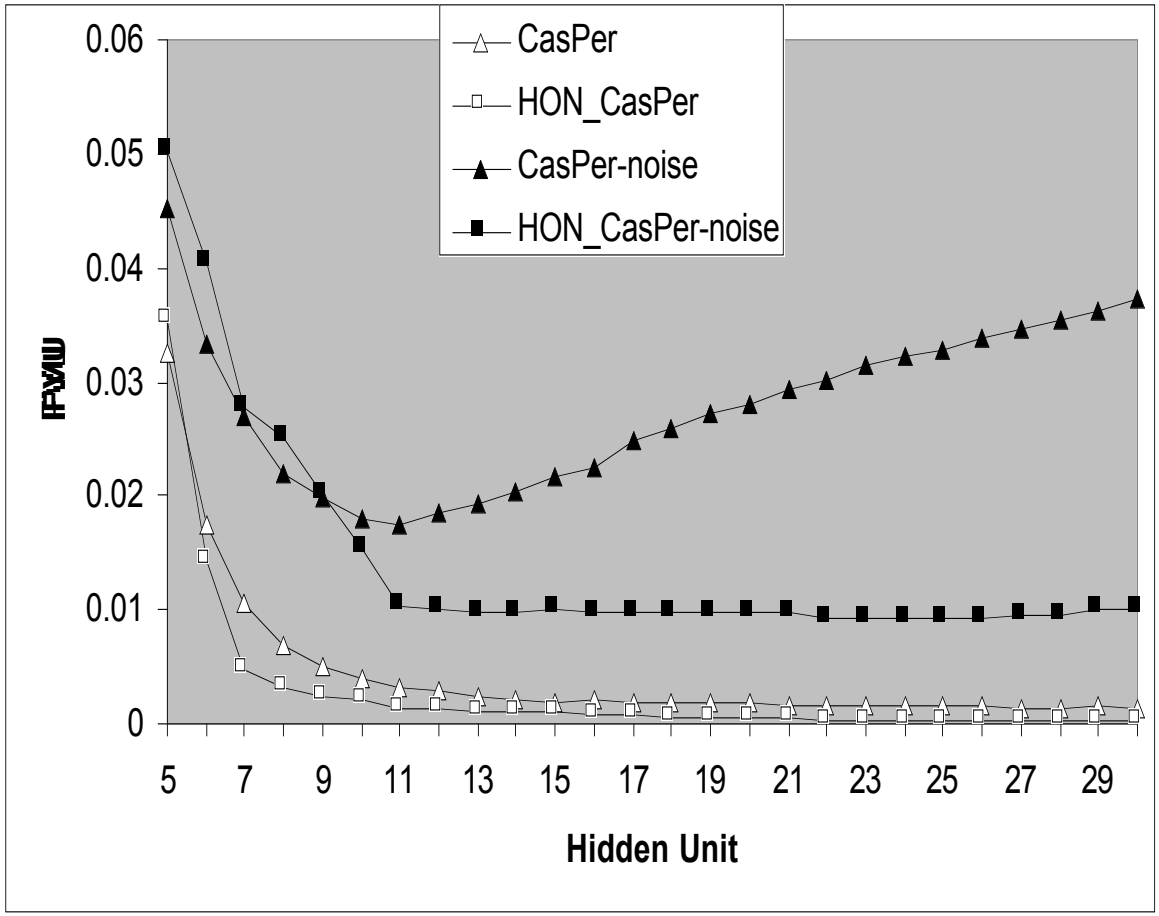


CasPer vs HON_CasPer

- Regression Benchmark: Complex Additive Function



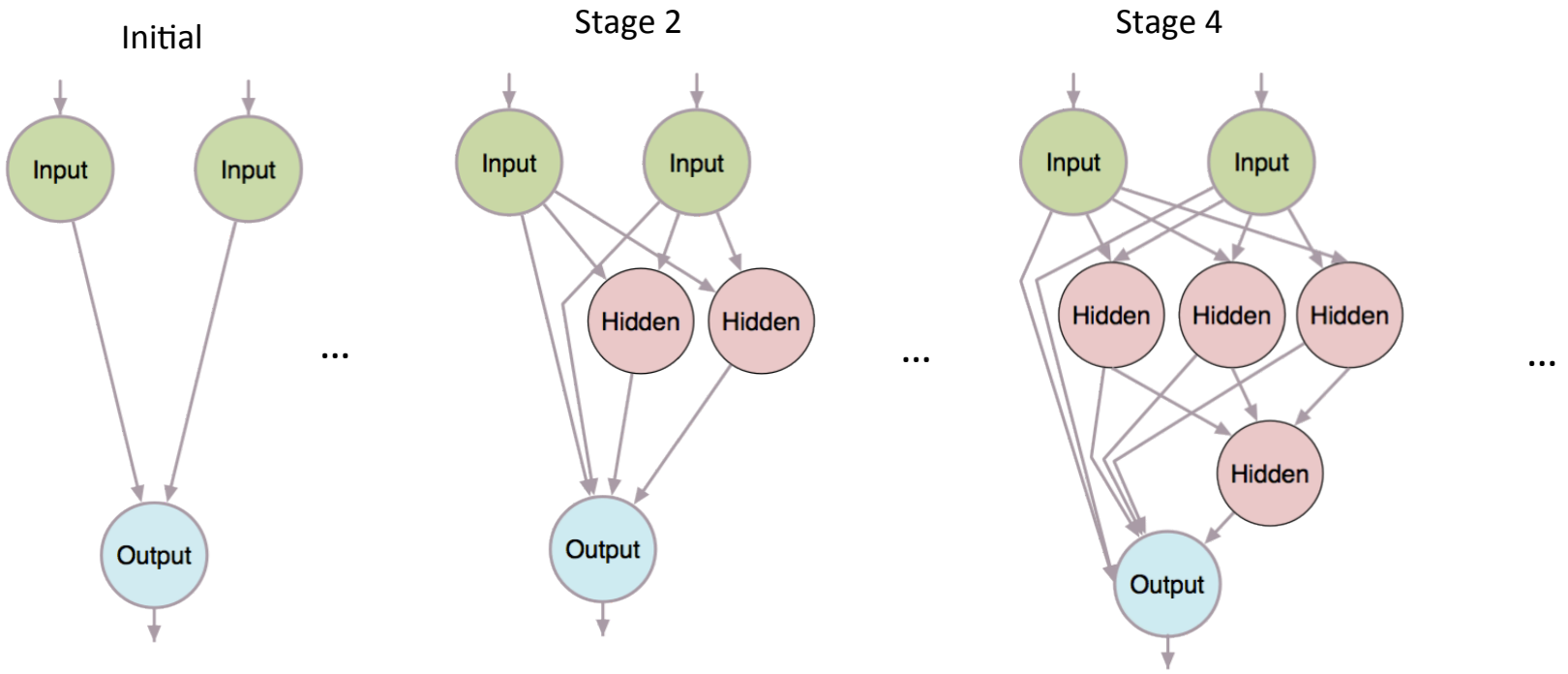
- HON_CasPer stops overfitting
  - Fewer weights



Layered Cascade Neural Network

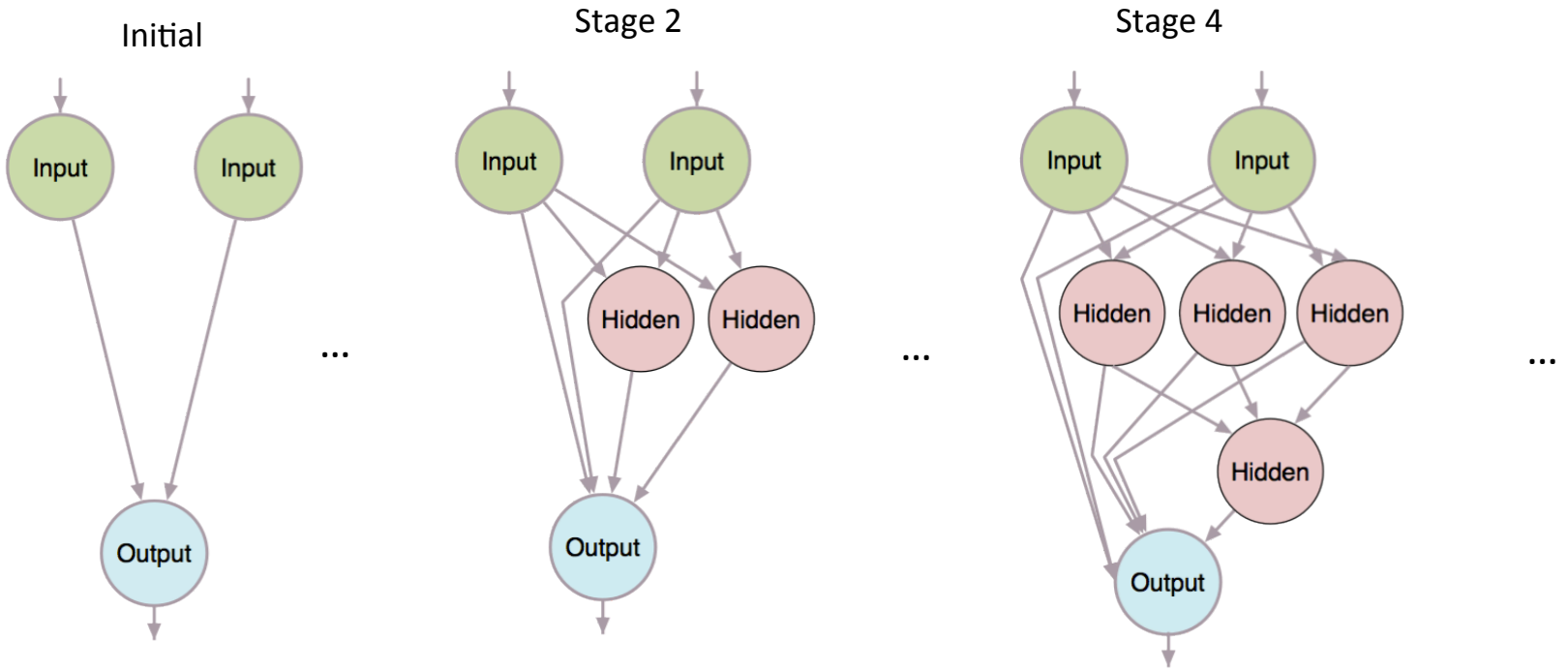
CasPer: each neuron has inputs from all previous neurons

Layered CasPer: new neurons added beside previous neurons up to a limit
then a new layer neuron is added



Layered CasPer

- **CasPer**: a neuron has inputs from all previous neurons
- **Layered CasPer**: new neurons added beside previous neurons up to a limit, then a new layer neuron is added



Connection crossings

- "we measure learning time in *connection crossings,* defined as the number of multiply-accumulate steps necessary to propagate activation values forward through the network and error values backward. This measure leaves out some computational steps, but it is a more accurate measure of computational complexity than comparing epochs of different sizes" (Fahlman)

Results on benchmark datasets

Data Set	Algorithm	Average number of hidden neurons	Average Connection Crossings	Mean of Error Rate
Cancer	AT__CasPer	5.5	2.15E+07	1.29
	A_CasPer	3.9	2.21E+07	**1.15**
	Layered-CasPer1	2.9	9.70E+05	2.13
	Layered-CasPer2	7.1	3.64E+07	1.55
Card	AT__CasPer	7.4	1.03E+08	13.40
	A_CasPer	7.8	1.00E+08	13.72
	Layered-CasPer1	7.6	4.84E+07	13.31
	Layered-CasPer2	6.3	1.23E+08	**12.38**
Diabetes	AT__CasPer	8.2	2.95E+07	**22.88**
	A_CasPer	8.6	7.07E+07	23.14
	Layered-CasPer1	7.7	1.98E+07	23.91
	Layered-CasPer2	7.8	5.09E+07	23.28
Gene	AT__CasPer	5.2	7.25E+08	12.32
	A_CasPer	3.2	4.24E+08	11.72
	Layered-CasPer1	3.8	6.18E+08	8.28
	Layered-CasPer2	6.4	1.96E+09	**7.80**
Glass	AT__CasPer	4.8	7.98E+06	28.42
	A_CasPer	6.2	1.00E+07	27.68
	Layered-CasPer1	4.3	7.95E+06	30.38
	Layered-CasPer2	5.7	1.87E+07	**27.42**

Results continued

Heart	AT__CasPer	3.4	3.68E+07	17.26
	A_CasPer	6.1	7.01E+07	**17.21**
	Layered-CasPer1	7.4	4.26E+07	19.52
	Layered-CasPer2	6.5	1.08E+08	18.83
Heartc	AT__CasPer	2.4	6.84E+06	18.34
	A_CasPer	5	3.43E+07	18.85
	Layered-CasPer1	6.1	4.05E+07	17.87
	Layered-CasPer2	6.3	1.39E+08	**17.67**
Horse	AT__CasPer	3.2	2.36E+07	29.73
	A_CasPer	5.6	5.48E+07	32.45
	Layered-CasPer1	3	1.11E+07	26.26
	Layered-CasPer2	4	4.41E+07	**23.85**
Soybean	AT__CasPer	1.8	1.51E+08	7.26
	A_CasPer	2.4	1.78E+08	7.19
	Layered-CasPer1	7.2	2.59E+08	6.77
	Layered-CasPer2	6.3	6.80E+08	**6.88**
Thyroid	AT__CasPer	9.6	1.16E+09	**1.53**
	A_CasPer	8.6	8.30E+08	1.67
	Layered-CasPer1	4.6	1.50E+08	4.33
	Layered-CasPer2	6.1	4.04E+08	4.22