Carnegie Mellon University
Language Technologies Institute

Cascade-Correlation and Deep Learning

Scott E. Fahlman

Professor Emeritus
Language Technologies Institute

February 27, 2019

Two Ancient Papers

- Fahlman, S. E. and C. Lebiere (1990) "The Cascade-Correlation Learning Architecture", in NIPS 1990.
- Fahlman, S. E. (1991) "The Recurrent Cascade-Correlation Architecture" in NIPS 1991.

Both available online at http://www.cs.cmu.edu/~sef/sefPubs.htm

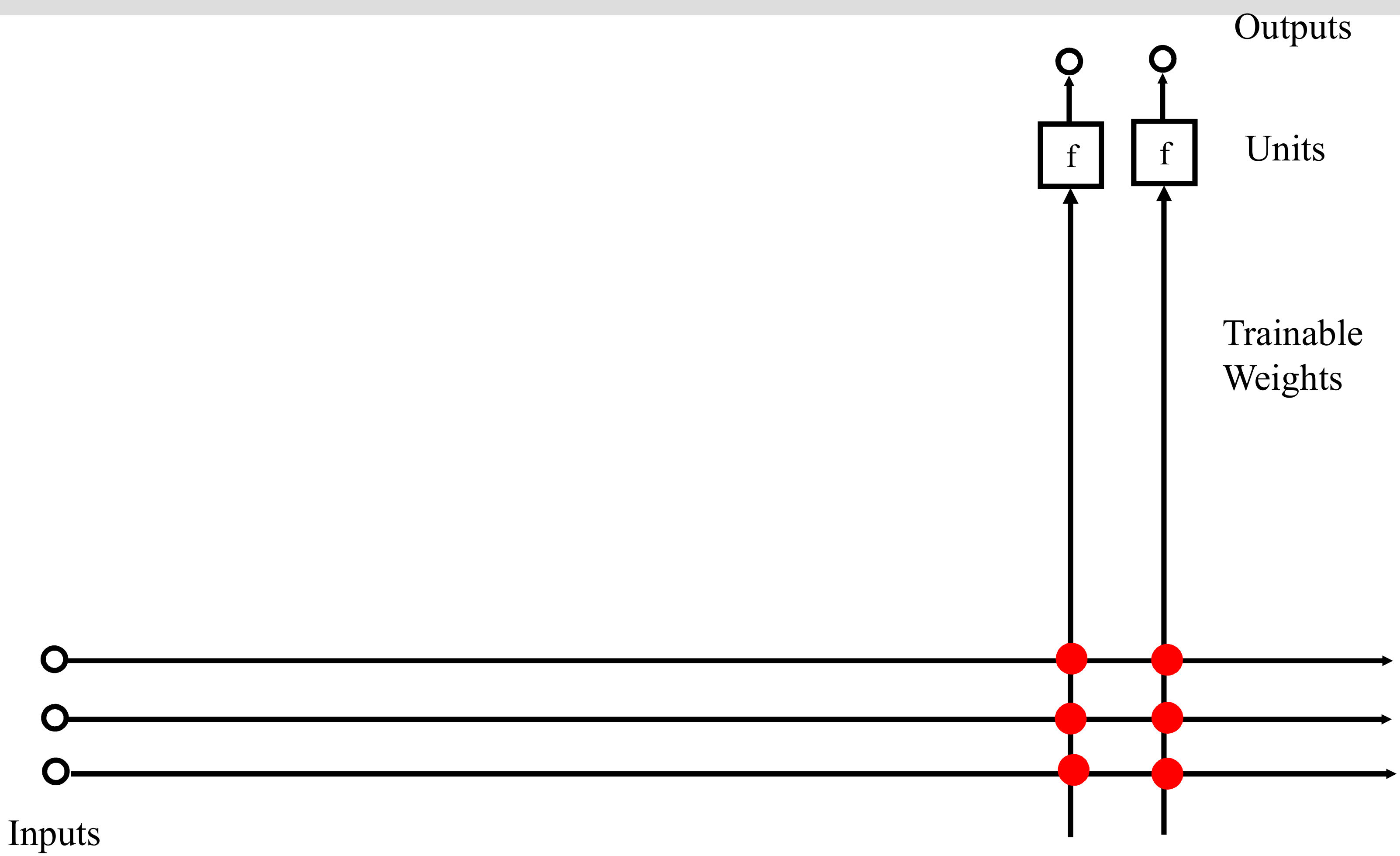
Deep Learning 28 Years Ago?

- These algorithms routinely built useful feature detectors 15-30 layers deep.
- Build just as much network structure as they needed – no need to guess network size before training.
- Solved some problems considered hard at the time, 10x to 100x faster than standard backprop.
- Ran on a single-core, 1988-vintage workstation, no GPU.
- But we never attacked the huge datasets that characterize today's "Deep Learning".

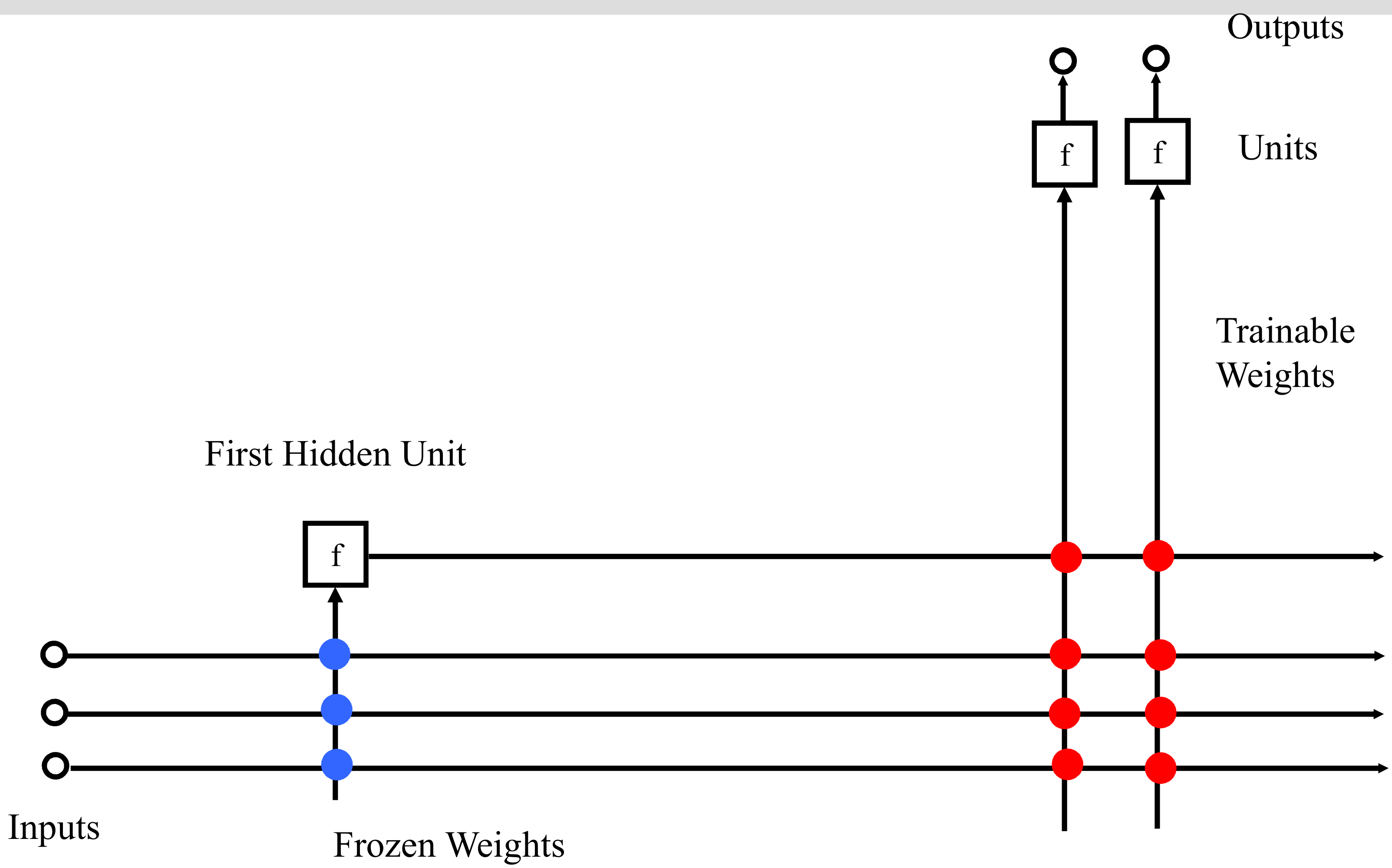
Why Is Backprop So Slow?

- **Moving Targets:**
  - All hidden units are being trained at once, changing the environment seen by the other units as they train.
- **Herd Effect:**
  - Each unit must find a distinct job -- some component of the error to correct.
  - All units scramble for the most important jobs. No central authority or communication.
  - Once a job is taken, it disappears and units head for the next-best job, including the unit that took the best job.
  - A chaotic game of “musical chairs” develops.
  - This is a very inefficient way to assign a distinct useful job to each unit.

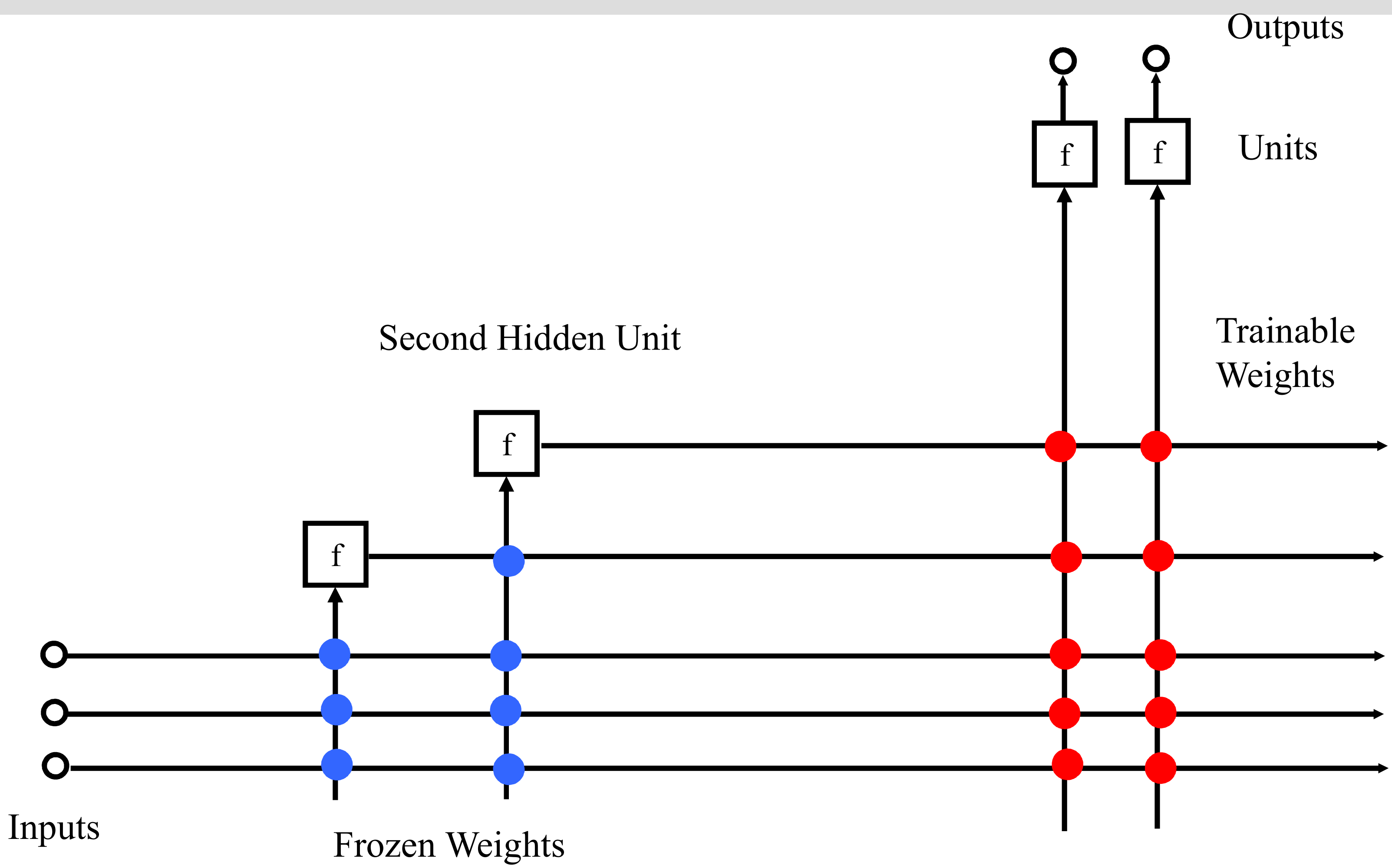
Cascade Architecture



Cascade Architecture



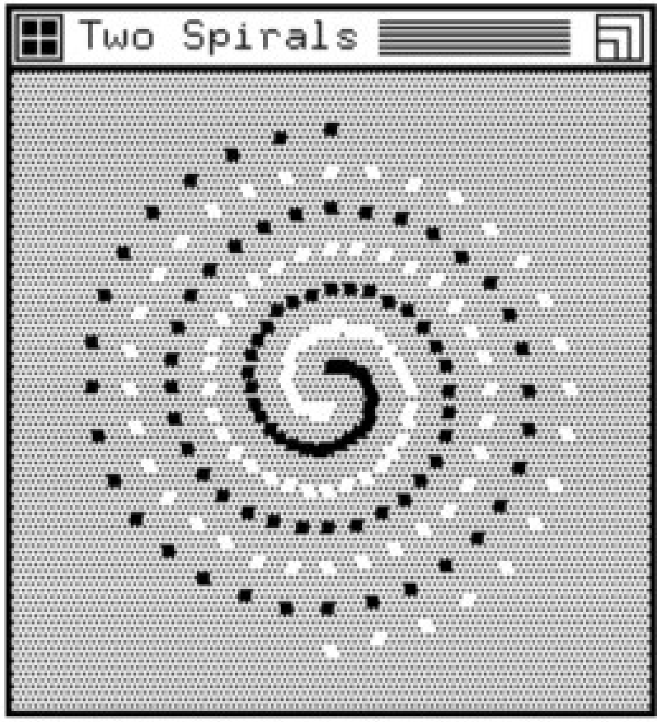
Cascade Architecture



The Cascade-Correlation Algorithm

- Start with direct I/O connections only. No hidden units.
- Train output-layer weights using BP or Quickprop.
- If error is now acceptable, quit.
- Else, Create **one** new hidden unit offline.
  - Create a **pool** of candidate units. Each gets all available inputs. Outputs are not yet connected to anything.
  - Train the incoming weights to maximize the match (covariance) between each unit's output and the residual error:
  - When all are quiescent, **tenure** the winner and add it to active net. Kill all the other candidates.
- Re-train output layer weights and repeat the cycle until done.

Two-Spirals Problem & Solution



Cascor Performance on Two-Spirals

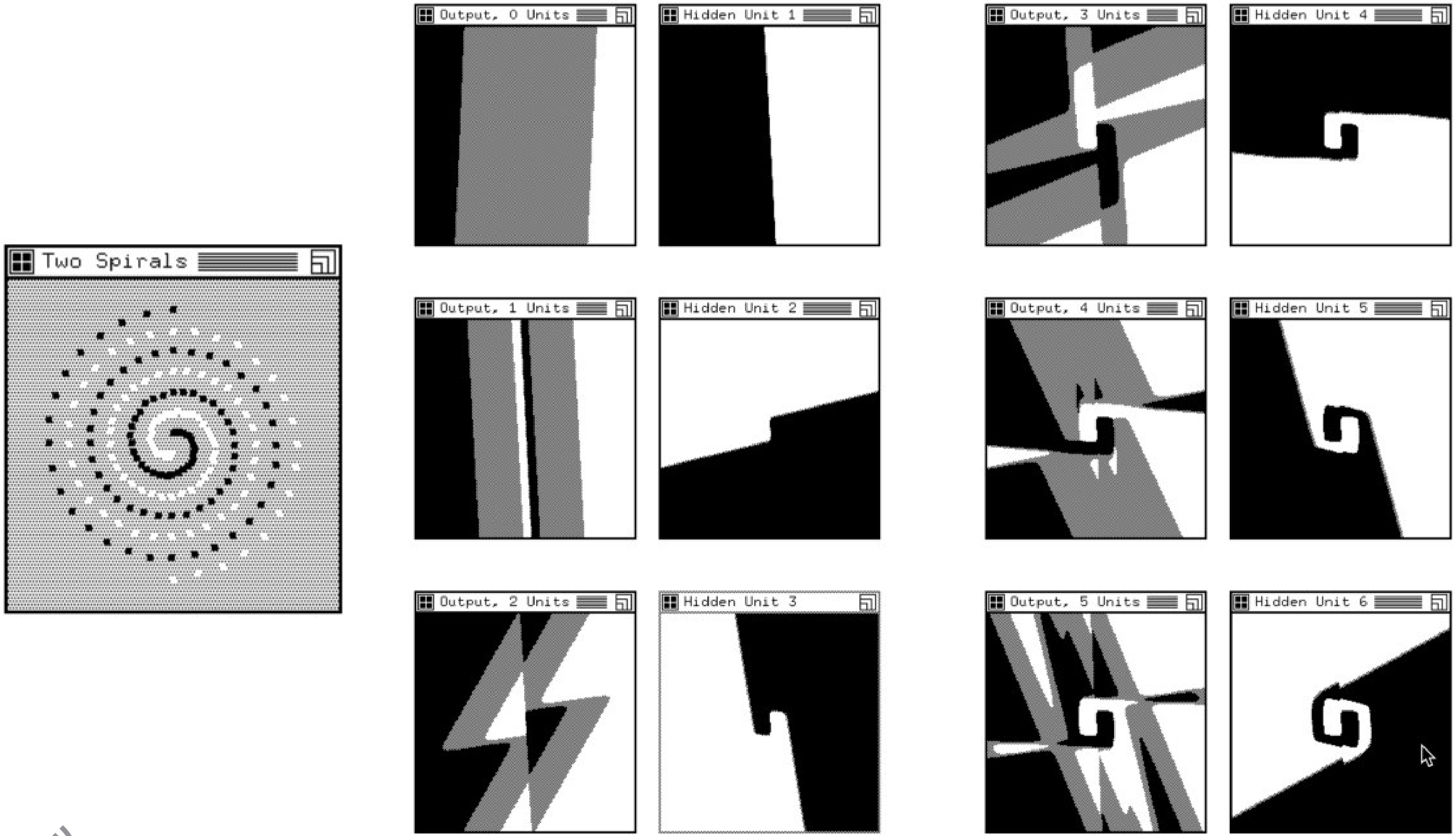
Hidden Units	Number of Trials	
12	4	####
13	9	#########
14	24	########################
15	19	###################
16	24	########################
17	13	#############
18	5	#####
19	2	##

Standard BP 2-5-5-5-1: 20K epochs, 1.1G link-X

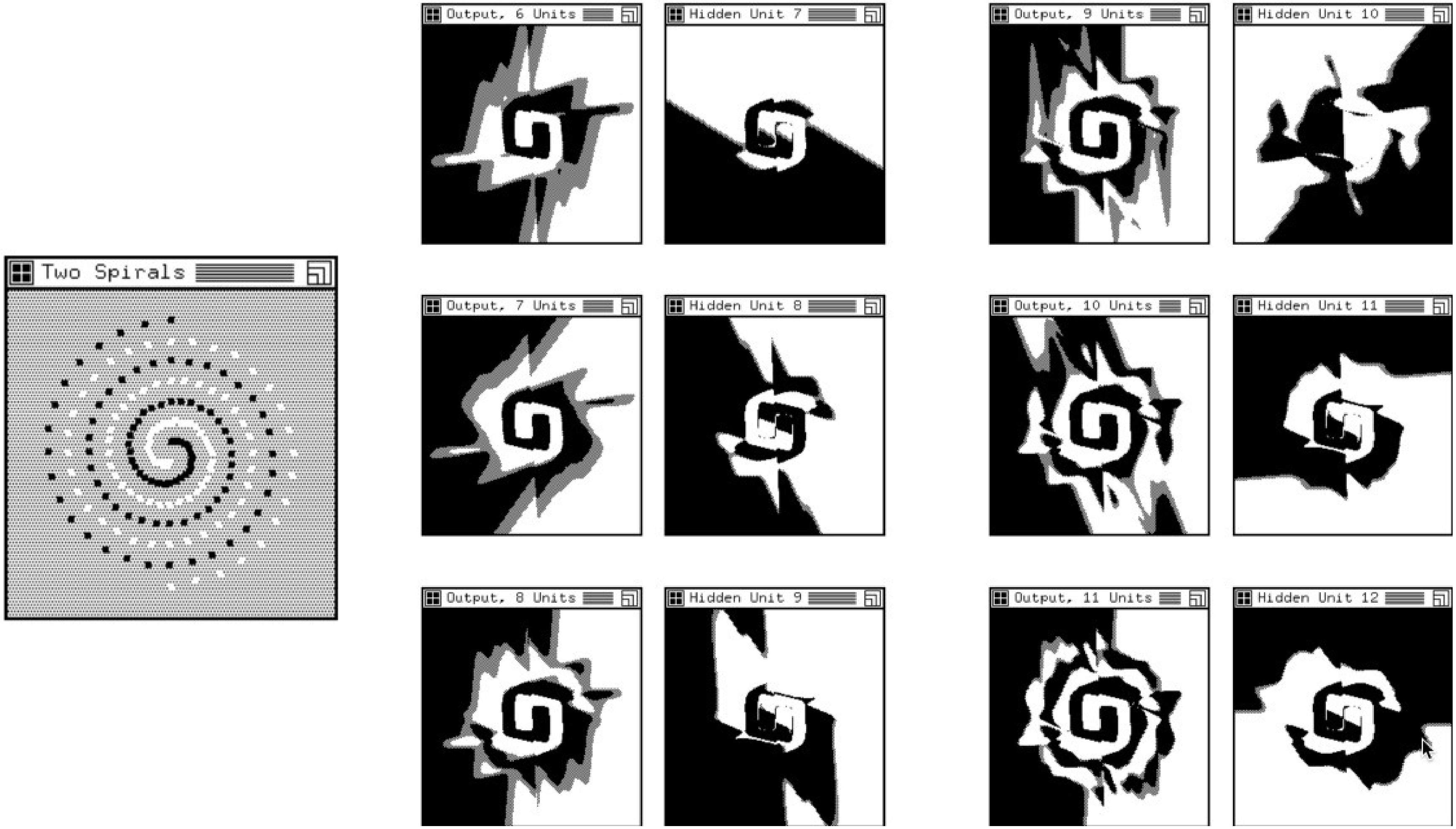
Quickprop 2-5-5-5-1: 8K epochs, 438M link-X

Cascor: 1700 epochs, 19M link-X

Cascor-Created Hidden Units 1-6



Cascor-Created Hidden Units 7-12

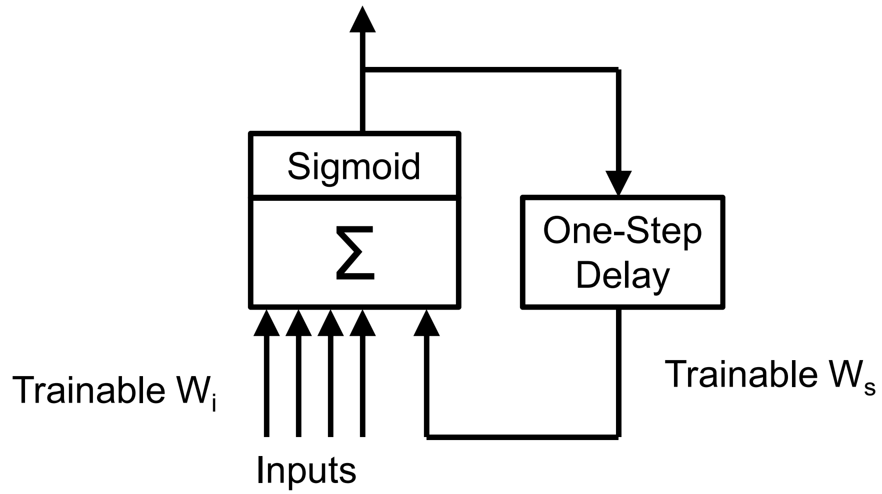


Advantages of Cascade Correlation

- No need to guess size and topology of net in advance.
- Can build deep nets with higher-order features.
- Much faster than Backprop or Quickprop.
- Trains just one layer of weights at a time (fast).
- Works on smaller training sets (in some cases, at least).
- Old feature detectors are frozen, not cannibalized, so good for incremental “curriculum” training.
- Good for parallel implementation.

Recurrent Cascade Correlation (RCC)

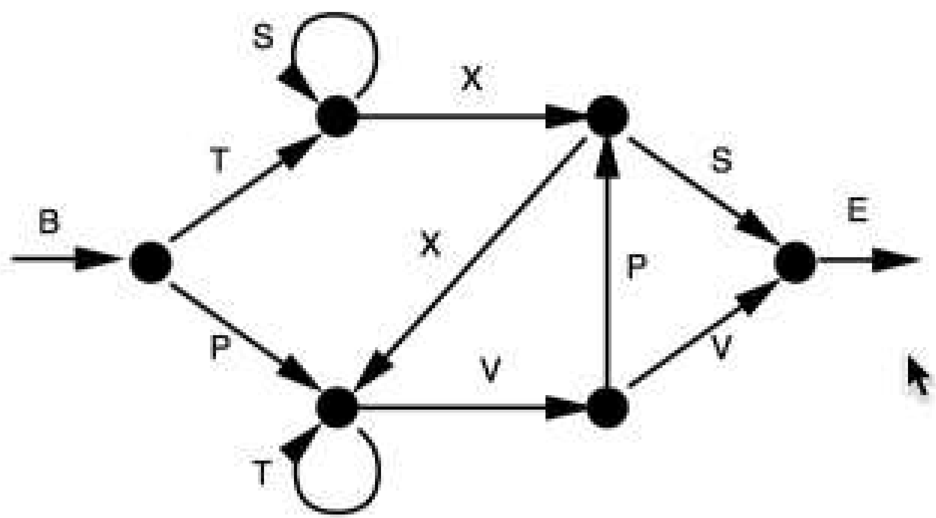
Simplest possible extension to Cascor to handle sequential inputs:



- Trained just like Cascor units, then added, frozen.
- If W_s is strongly positive, unit is a memory cell for one bit.
- If W_s is strongly negative, unit wants to alternate 0-1.

Reber Grammar Test

The Reber grammar is a simple finite-state grammar that others had used to benchmark recurrent-net learning.



Typical legal string: “BTSSXXVPSE”.

Task: Tokens presented sequentially. Predict the next Token.

Reber Grammar Results

State of the art:

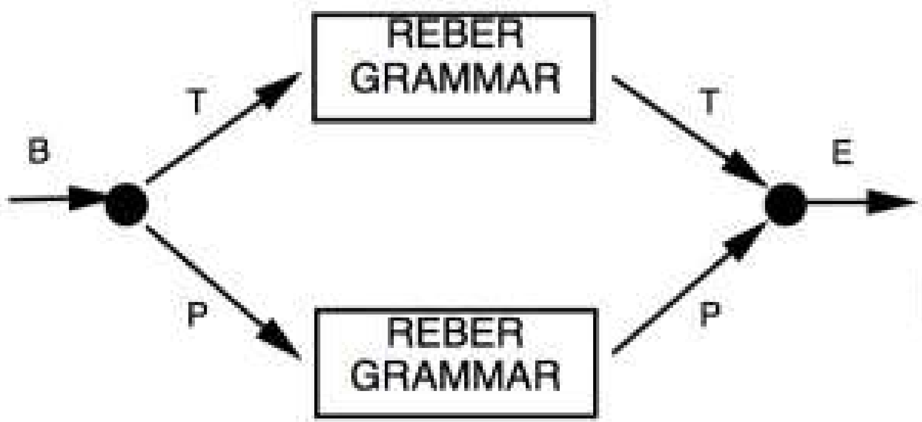
- Elman net (fixed topology with recurrent units): 3 hidden units, learned the grammar after seeing 60K distinct strings, once each. (Best run, not average.)
- With 15 hidden units, 20K strings suffice. (Best run.)

RCC Results:

- Fixed set of 128 training strings, presented repeatedly.
- Learned the task, building 2-3 hidden units.
- Average: 195.5 epochs, or 25K string presentations.
- All tested perfectly on new, unseen strings.

Embedded Reber Grammar Test

The embedded Reber grammar is harder.



Must remember initial T or P token and replay it at the end.

Intervening strings potentially have many Ts and Ps of their own.

Embedded Reber Grammar Results

State of the art:

- Elman net was unable to learn this task, even with 250,000 distinct strings and 15 hidden units.

RCC Results:

- Fixed set of 256 training strings, presented repeatedly, then tested on 256 different strings. 20 runs.
- Perfect performance on 11 of 20 runs, typically building 5-7 hidden units.
- Worst performance on others, 20 test-set errors.
- Training required avg of 288 epochs, 200K string presentations.

Morse Code Test

- One binary input, 26 binary outputs (one per letter), plus "strobe" output at end.
- Dot is 10, dash 110, letter terminator adds an extra zero.
- So letter V …- is 1010101100.
  Letters are 3-12 time-steps long.
- At start of each letter, we zero the memory states.
- Outputs should be all zero except at end of letter – then 1 on the strobe and on correct letter.

Morse Code Results

- Trained on entire set of 26 patterns, repeatedly.
- In ten trials, learned the task perfectly every time.
- Average of 10.5 hidden units created.
  - Note: Don't need a unit for every pattern or every time-slice.
- Average of 1321 epochs.

Scott E. Fahlman <sef@cs.cmu.edu>

“Curriculum” Morse Code

Instead of learning the whole set at once, present a series of lessons, with simplest cases first.

- Presented E (one dot) and T (one dash) first, training these outputs and the strobe.
- Then, in increasing sequence length, train “AIN”, “DMSU”, “GHKRW”, “BFLOV”, “CJPQXYZ”. Do not repeat earlier lessons.
- Finally, train on the entire set.

Lesson-Plan Morse Results

- Ten trials run.
- E and T learned perfectly, usually with 2 hidden units.
- Each additional lesson adds 1 or 2 units.
- Final combination training adds 2 or 3 units.
- Overall, all 10 trials were perfect, average of 9.6 units.
- Required avg of 1427 epochs, vs. 1321 for all-at-once, but these epochs are very small.
- On average, saved about 50% on training time.

Cascor Variants

- Cascade 2: Different correlation measure works better for continuous outputs.
- Mixed unit types in pool: Gaussian, Edge, etc. Tenure whatever unit grabs the most error.
- Mixture of descendant and sibling units. Keeps detectors from getting deeper than necessary.
- Mixture of delays and delay types, or trainable delays.
- Add multiple new units at once from the pool, if they are not completely redundant.
- KBCC: Treat previously learned networks as candidate units.

Key Ideas

- Build just the structure you need. Don’t carve the filters out of a huge, deep block of weights.
- Train/Add one unit (feature detector) at a time. Then add and freeze it, and train the network to use it.
  - Eliminates inefficiency due to moving targets and herd effect.
  - Freezing allows for incremental “lesson-plan” training.
  - Unit training/selection is very parallelizable.
- Train each new unit to cancel some residual error. (Same idea as boosting.)

So…

- I still have the old code in Common Lisp and C.
Serial, so would need to be ported to work on GPUs, etc.
- My primary focus is Scone, but I am interested in collaborating with people to try this on bigger problems.
- It might be worth trying Cascor and RCC on inferring real natural-language grammars and other Deep Learning/Big Data problems.
- Perhaps tweaking the memory/delay model of RCC would allow it to work on time-continuous signals such as speech.
- A convolutional version of Cascor is straightforward, I think.
- The *hope* is that this might require **less data** and **much less computation** than current deep learning approaches.

Some Current Work

- One PhD student Dean Alderucci, has ported RCC to Python using Graham Neubig's Dynet toolkit.
  - Dean will be looking at using this for NLP applications specifically aimed at the language in patents.
  - Dean also has done some work on word embeddings, developing a version of word2vec using Scone.
- An undergrad, Ian Chiu, has ported Cascor to Python, running on TensorFlow Eager, which can handle networks that change during processing. Some issues remain.
  - Ian is now looking for good sequential benchmarks to compare the speed of RCC.
  - It's surprisingly hard to find reported results that we can compare for learning speed.

The End

Equations: Cascor Candidate Training

Adjust candidate weights to maximize covariance S:

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

Adjust incoming weights:

$$\partial S / \partial w_i = \sum_{p,o} \sigma_o (E_{p,o} - \overline{E_o}) f'_p I_{i,p}$$

Equations: RCC Candidate Training

Output of each unit;

$$V(t) = \sigma\left(\sum_i I_i(t)\, w_i \;+\; V(t-1)\, w_s\right)$$

Adjust incoming weights:

$$\partial S / \partial w_i = \sum_{p,o} \sigma_o (E_{p,o} - \overline{E_o}) f'_p I_{i,p}$$

$$\partial V(t) / \partial w_s = \sigma'(t)\ \left(V(t-1) + w_s\, \partial V(t-1)/\partial w_s\right)$$