Output and Hidden Unit behaviour of Neural Networks

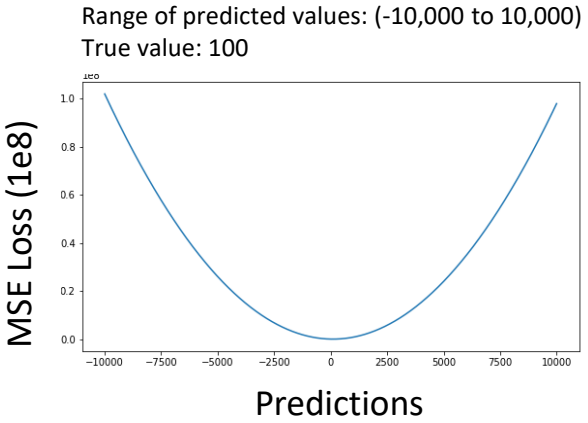
- Output activation functions and cost functions
- Disadvantages of Neural Networks
- Rules of thumb
- Optimisation functions
- Nettalk and MNIST Softmax example
- Hidden Unit Behaviour
  - Assumptions
  - Pruning networks
    - by inspection
    - taxonomy of undesirable units
    - Distinctiveness
  - Robustness and damage resistance
  - Plateaux during training
  - Image compression example

Output activation functions and cost functions

- The output activation function is determined by the target data
  - Sigmoid can be used for 2 class classification
    - e.g. spam vs not spam
  - Softmax for multiple mutually exclusive class classification
    - e.g. classifying handwritten digits
  - Can use linear (no activation function) when predicting real values.
    - e.g. predicting share values.

Output activation functions and cost functions 2

- The cost function is generally determined by the output activation function.
  - We have seen the squared error loss, this is usually used with real valued outputs (no activation function)

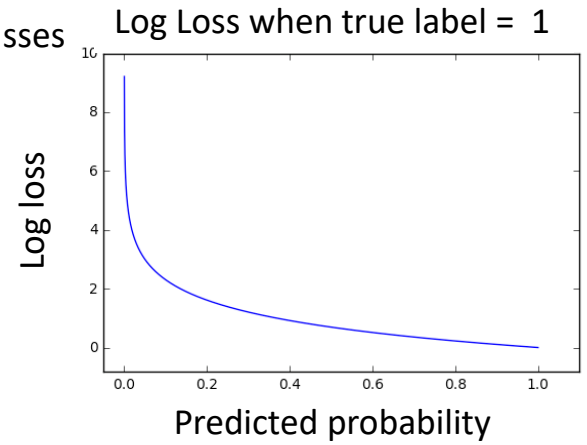


  - The cross-entropy/negative log likelihood loss is generally used with the sigmoid or softmax activation functions for classification problems

$$L(y,t) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log y_{nk}$$

where N = number of training examples and K = number of classes

  - The Cross Entropy/Log Loss produces a high penalty and a steep gradient for outputs that are very wrong. This counteracts the small gradients in the tails of the softmax and sigmoid activation functions and results in much faster learning for these examples.



Disadvantages of back propagation

- Slow to train
- Architecture not easily determinable *a priori*
  - input units – easy
  - hidden units – hard
  - Output units – easy
  - Use neural networks for hard problems
  - More hidden units required during learning
- Moving target
- Herd effect
- Step size

Some rules of thumb

- Selecting number of neurons in a layer:

  (for a vanilla feed forward fully connected NN)

  - input layer : depends on data available
  - 1st hidden layer: ≥ number of neurons in 2nd hidden layer
  - last hidden layer : ~ number of convex or disconnected regions, i.e. number of higher order features expected
  - output layer : depends on classification or prediction task
  - in general – successive layers of processing neurons should have fewer neurons

- When a network does not learn acceptably due to underfitting:

  (the loss on the training set doesn't reduce as much as expected)

  - add an extra hidden neuron
  - add 10% extra hidden neurons
  - double number of hidden neurons

Optimisation algorithms and tricks

Optimisation algorithms determine how the weights are updated based on their gradients

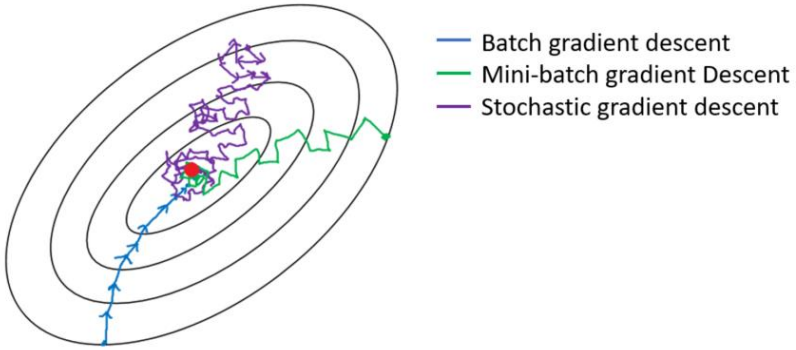
- The simplest optimisation algorithm is stochastic gradient descent which was introduced in the back-prop algorithm:

$$w_{kl}(t+1) = w_{kl}(t) + \eta(-\frac{dL}{dw_{kl}}) \quad 0 < \eta \leq 1$$

  - Stochastic gradient descent updates the weights a small amount for every training example and in real world problems results in a very noisy and slow training path.
  - Gradient descent can also be done in batches of either the entire training dataset or part of it.
  - On simple convex loss surfaces (full) batch gradient descent results in the shortest path to the minima.
  - On complex real world error surfaces mini-batch gradient descent can help get the algorithm out of local minima.

Optimisation algorithms 2

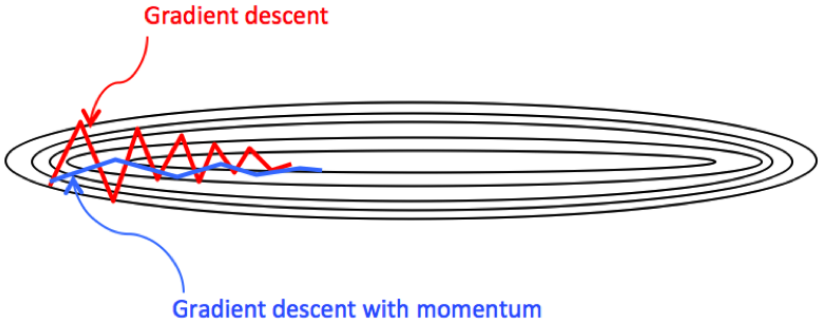
There are a number of other tricks to speed up training and help the model get out of local minima that can be added to the weight update formula or the cost function



- Momentum: adds a fraction of the previous update to the current update. Dampens wild swings, especially when the error surface is steep in one dimension and not in others.

$$w_{kl}(t+1) = w_{kl}(t) + \eta\left(-\frac{dL}{dw_{kl}}\right) + \alpha\left(w_{kl}(t) - w_{kl}(t-1)\right)$$

$0 < \eta \leq 1 \quad 0 < \alpha \leq 1$



Optimisation algorithms 3

- Weight decay: adds an L2 penalty to the cost function to help stop the model overfitting the dataset and encourage generalisation. e.g. for the cross-entropy cost function:

$$L(y,t) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\log y_{nk} + \frac{1}{2}\lambda\sum_{i=1}^{W} w_i^2$$

where W is the total number of weights in the network so w^2 is summed over every weight in the network.

  - This leads to a new term in the in the weight update equation:

$$w_{kl}(t+1) = w_{kl}(t) + \eta\left(-\frac{dL}{dw_{kl}}\right) - \lambda\, w_{kl}$$

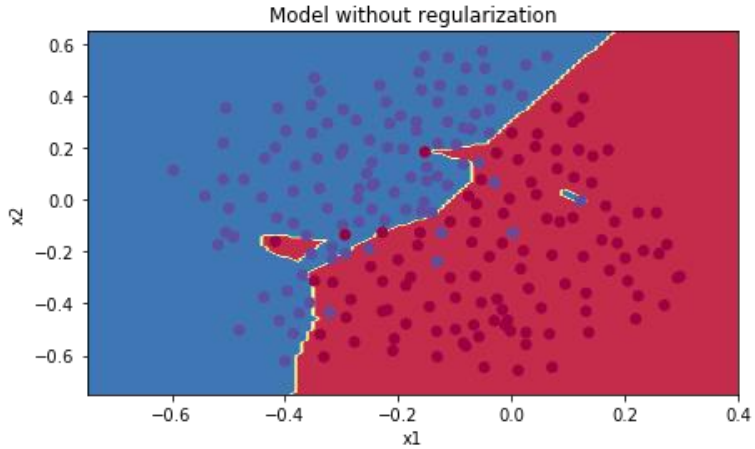


Figure 1: An unregularized model can overfit to noise/outliers in the training data

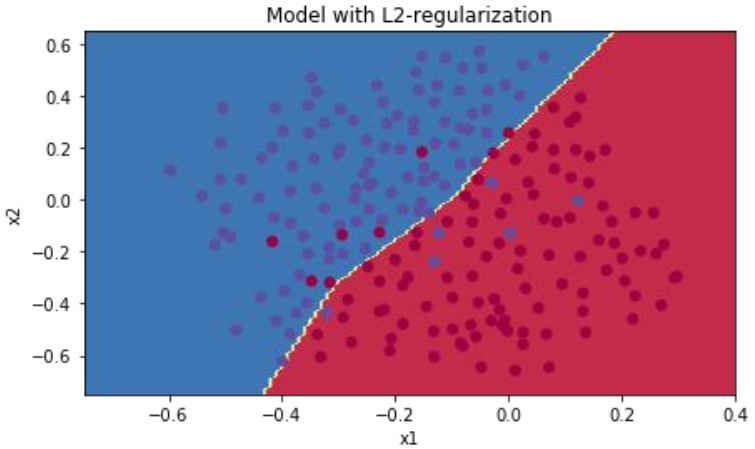
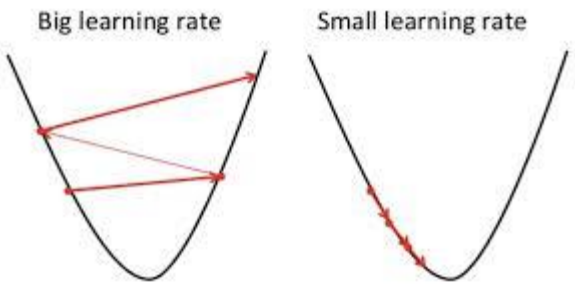


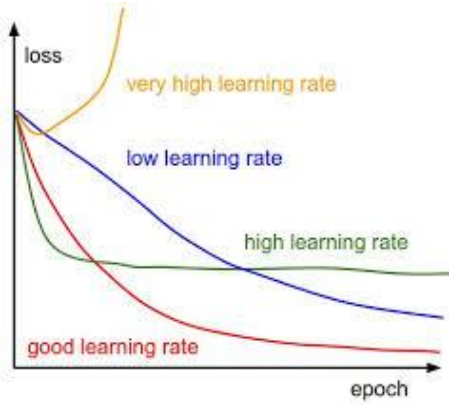
Figure 2: Applying regularization can help prevent overfitting on the training data

Optimisation algorithms 4

- Learning rate η:
  - Too big and you'll overshoot the minimum.
  - Too small and learning will be very slow and get stuck in local minima.
  - Usually start off higher (but not too high) and reduce it when learning plateaus or goes backwards
  - Modern neural network languages have learning rate schedulers which allow you to specify upfront when and how you want the learning rate to reduce

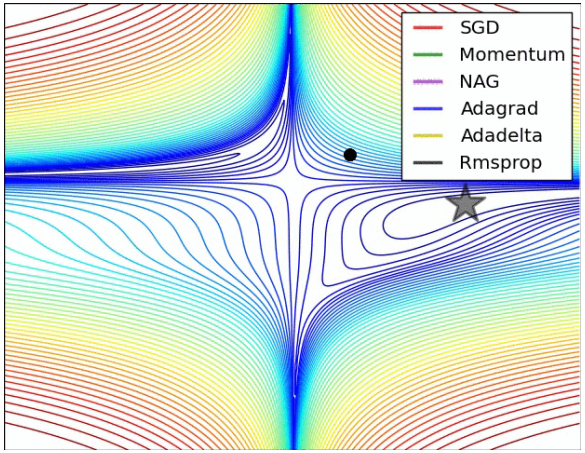


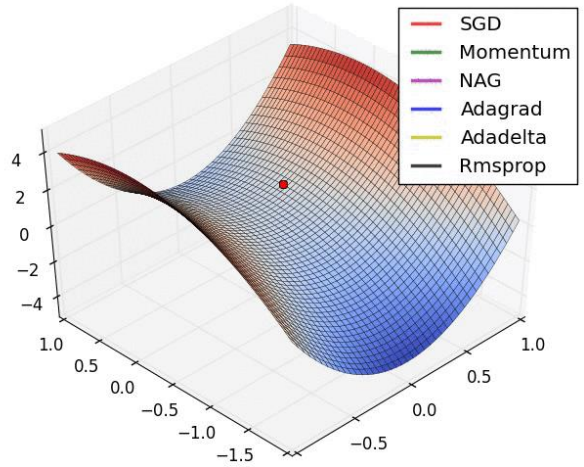
Although the red line is a good learning rate overall, the ideal learning rate schedule would start on the green line until learning plateaued then switch to the red line, then eventually the blue line.



Optimisation algorithms 5

Many optimisation algorithms that incorporate the above terms in different ways have emerged since the resurgence of neural networks this decade. Each has pros and cons and different problems will be best suited to different algorithms. Modern neural network languages like Pytorch have implementations of all best performing optimisation algorithms as well as the ability to design your own. https://pytorch.org/docs/stable/optim.html



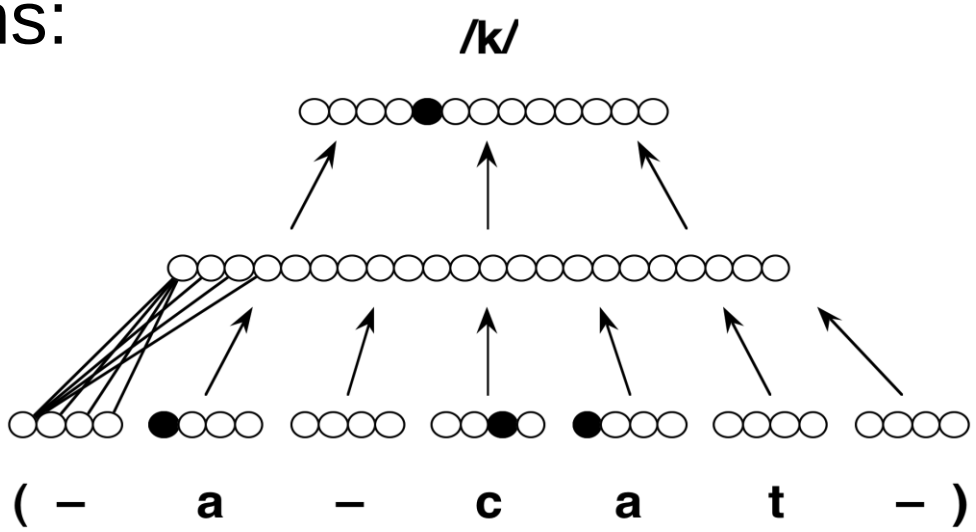


Figures: ability of different optimisation algorithms to navigate loss surfaces and get out of critical points.

http://ruder.io/optimizing-gradient-descent/

NETtalk: A network that learns to pronounce English text

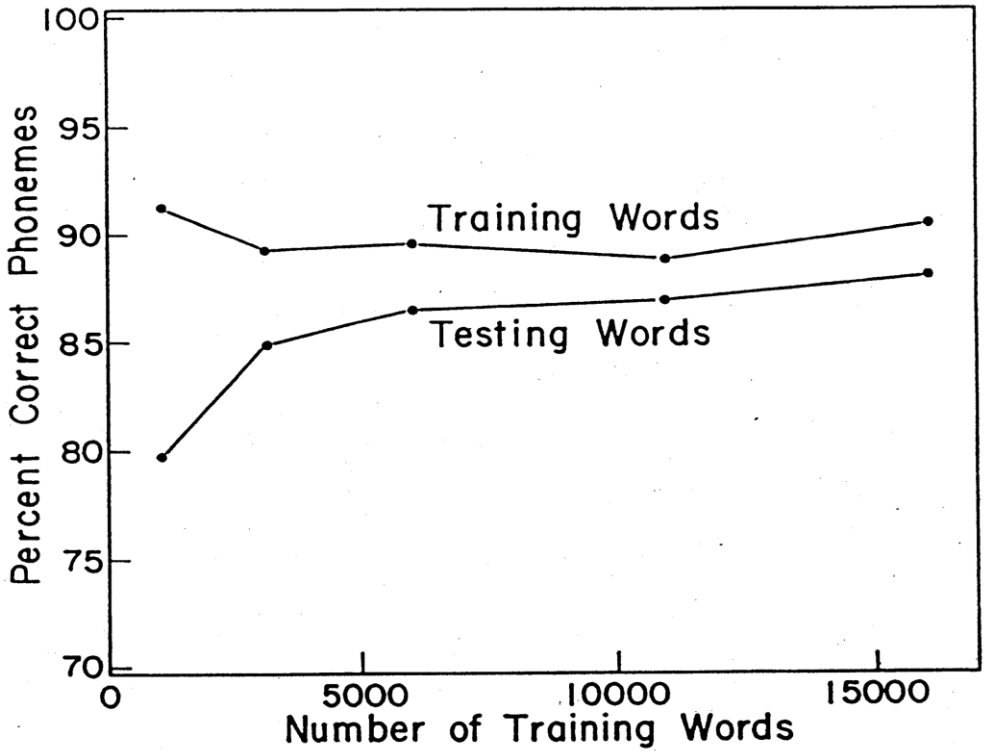
- Network 309 neurons:
  - 7 x 29 input neurons
  - 80 hidden neurons
  - 26 output neurons
- There are 8,629 connections.



- A window 7 letters wide is moved over the text, and the network pronounces the middle letter.

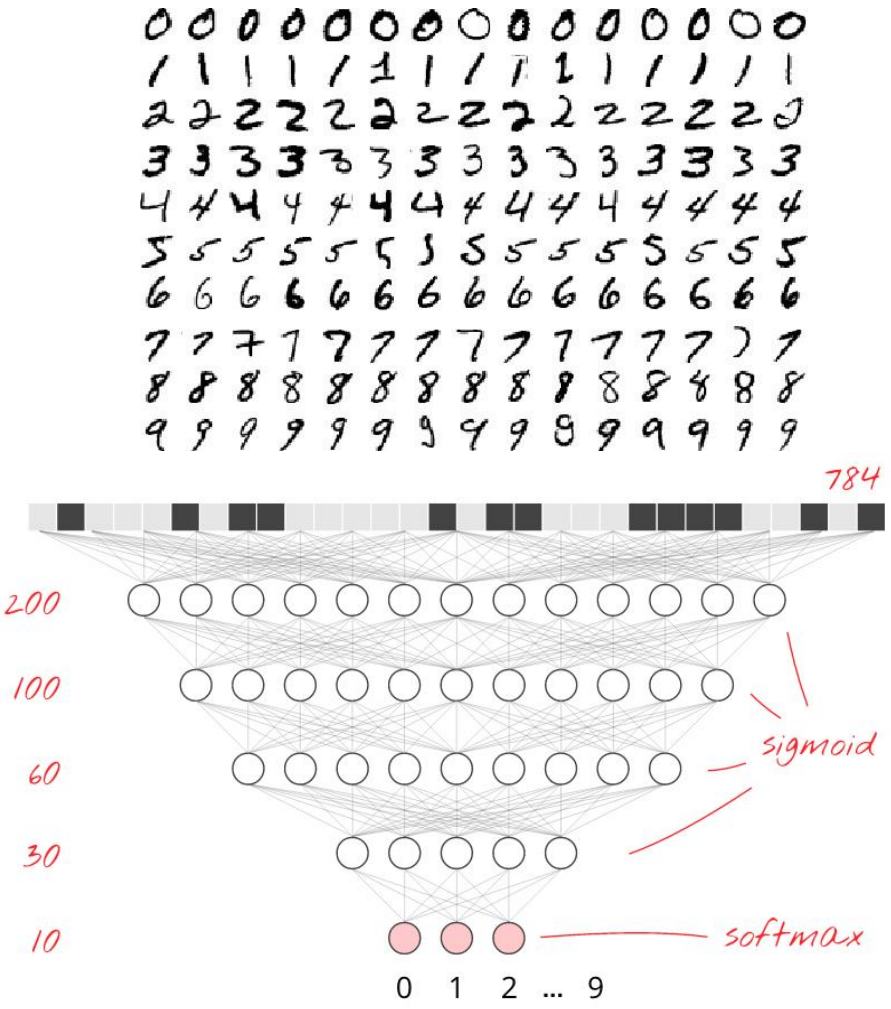
NETtalk: Generalisation to new text

- Separate networks were trained on subsets of 1000, 3000, 11000 and 16000 words from a dictionary.
- Testing used 1000 words new to network, and not in the training set.



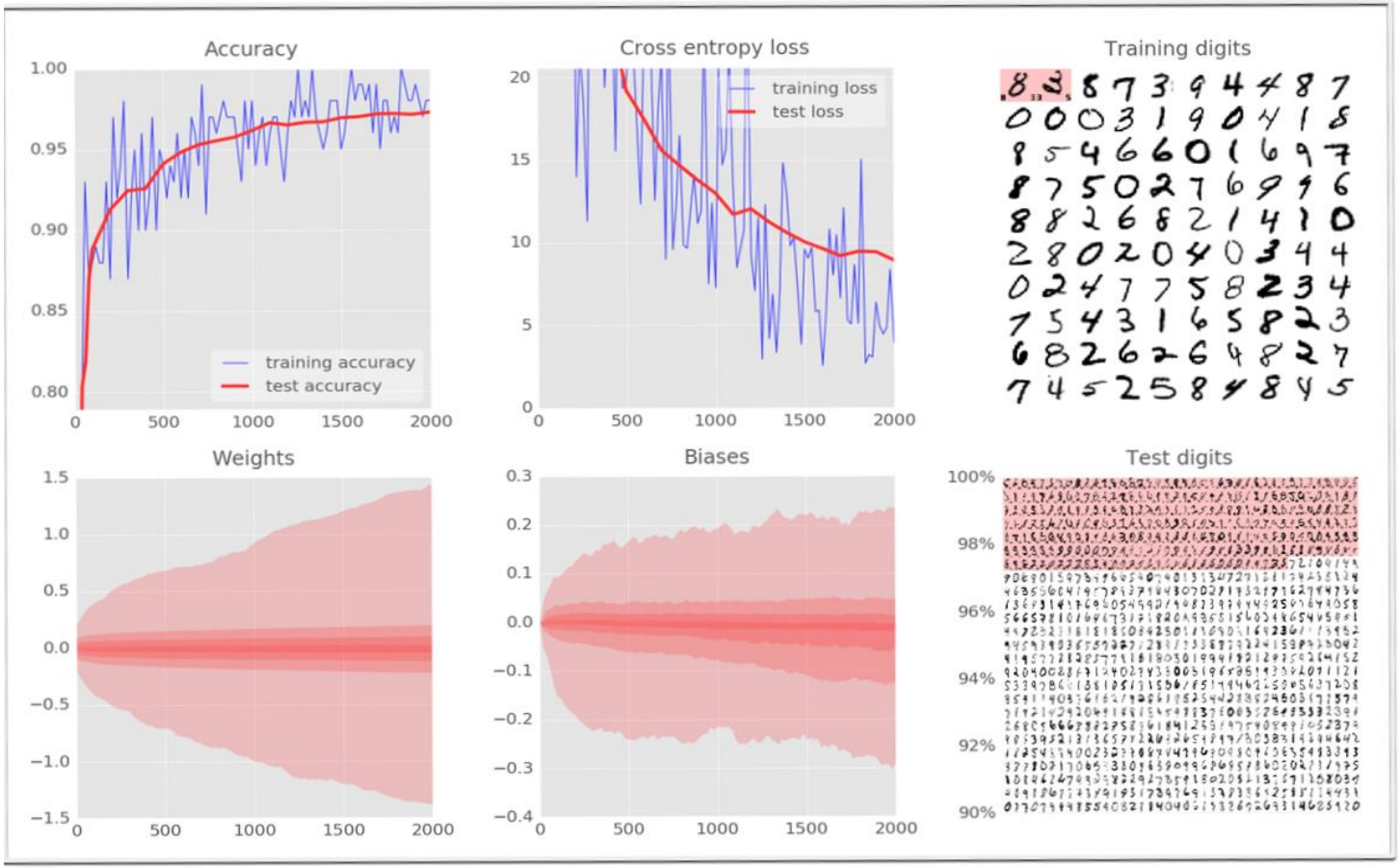
MNIST NN a network that learns to classify handwritten digits

- Modern networks use the Softmax activation function for the output units to achieve better results on multi-class classification problems.
- Modern GPU computing also allows for training of bigger networks with more training examples for better generalisation.



784 (28x28) pixel values feed into a 4 hidden layer NN with 184,900 weights

MNIST NN: training and generalisation



Hidden Unit Behaviour - Assumptions

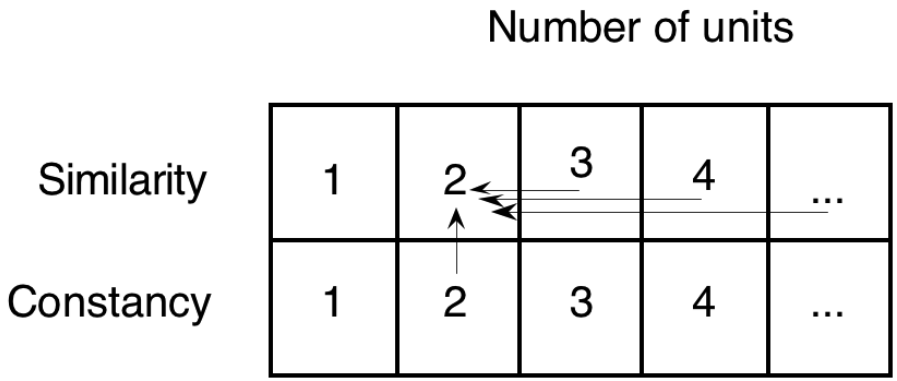
- Feed-forward network
- Trained by back propagation
  - Develop error gradient with respect to weights
  - Not restricted to standard gradient descent
- Three layers
- No connections
  - Laterally
  - Backward
  - multilayer
- Single output unit
- Simple weight connections
- Logistic activation function

Pruning Networks

- By inspection
  - Outputs close to zero
  - banded to 0/1, duplicates/inverses
- Relevance
  - unit in place vs. removed
  - extra back-propagation phase – different error function
- Contributions
  - activation * weight by pattern
  - vast amounts of data
  - non-activated units
- Sensitivity
  - approximate error if removed
  - store incremental changes to weights
- Badness
  - sum of back-propagated error component
  - baddest unit does most learning?

Taxonomy of undesirable units

- No function (one unit)
  - weights very small
  - always off
  - always on
- Inverse Units (two units)
  - Opposite output for all patterns
- Too similar (two or more units)
  - same output for all patterns
- Group no function (three or more units)
  - sum of vectors is zero



Example net: unit activs by pattern

- Hard to see any pattern in the units' behaviour on this pattern set

	Hidden Units							
Pattern	1	2	3	4	5	6	Result	Target
p.000	0.330	1.000	0.910	0.582	0.593	0.381	0.000	0.000
p.001	0.091	0.994	0.339	0.101	0.834	0.839	0.000	0.000
p.002	0.098	0.994	0.348	0.130	0.874	0.868	0.000	0.000
p.003	0.022	0.488	0.026	0.012	0.960	0.982	0.025	0.000
p.004	0.094	0.994	0.325	0.128	0.853	0.849	0.000	0.000
p.005	0.021	0.489	0.024	0.012	0.952	0.979	0.025	0.000
p.006	0.023	0.488	0.025	0.016	0.965	0.984	0.025	0.000
p.007	0.005	0.006	0.001	0.001	0.990	0.998	0.805	1.000
p.008	0.117	0.994	0.339	0.132	0.875	0.870	0.000	0.000
p.009	0.026	0.488	0.025	0.012	0.960	0.983	0.025	0.000
p.010	0.029	0.488	0.026	0.016	0.971	0.986	0.025	0.000
p.011	0.006	0.006	0.001	0.001	0.991	0.998	0.805	1.000
p.012	0.027	0.489	0.024	0.016	0.965	0.984	0.025	0.000
p.013	0.006	0.006	0.001	0.001	0.990	0.998	0.805	1.000
p.014	0.006	0.006	0.001	0.002	0.992	0.998	0.805	1.000
p.015	0.001	0.000	0.000	0.000	0.998	1.000	0.816	0.000

Distinctiveness

- Unit output activations over pattern set
- Similar and complementary units
  - calculate angle between vectors in pattern space
  - normalise to 0.5, 0.5
  - angle ≤ 15° – too similar – remove
  - angle ≥ 165° – complementary – remove
  - adjust weights, need not re-train
- No function groups
  - adaptively band to 0, 0.5 and 1.0
  - Gaussian vector pivot
  - Row echelon matrix
  - remove -> may need to re-train

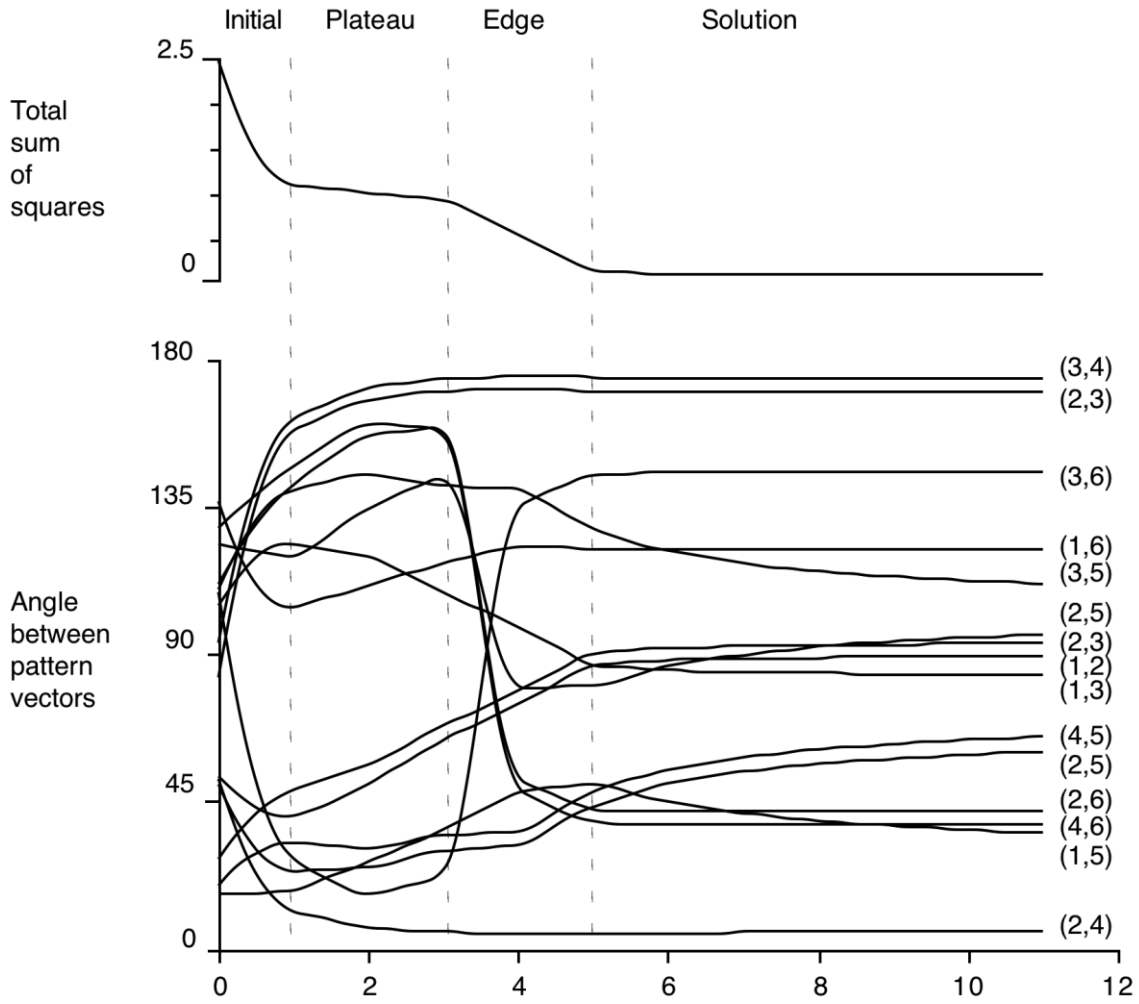
Example network: Vector Angles

- More comparisons to be made
- Much easier to see some similarities in units' behaviour

Pair of units	Vector angle
1 2	81.8
1 3	25.2
1 4	8.4
1 5	176.5
1 6	169.9
2 3	63.0
2 4	77.8
2 5	100.8
2 6	103.7
3 4	18.0
3 5	157.7
3 6	163.7
4 5	173.7
4 6	177.5
5 6	7.3

Hidden unit behaviour

- Visual display of network behaviour
- Effects of multiple units is visible
- 3 cases found



plateau which ends

- the angles between pattern vectors vary significantly during the plateau phase

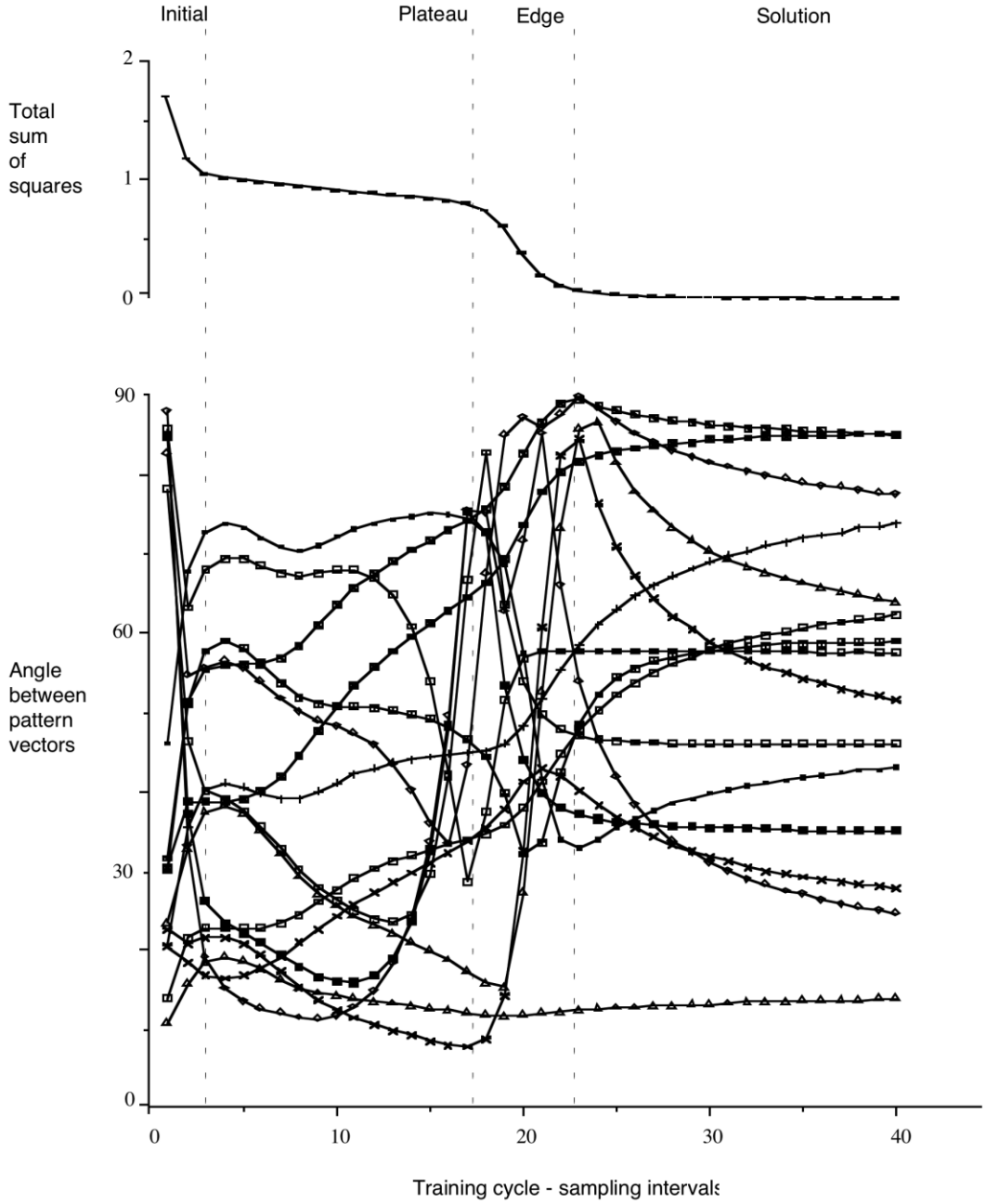


Figure 1: Network illustrating a plateau with a solution

no plateau

- the total sum of squares error measure decreases continually during training

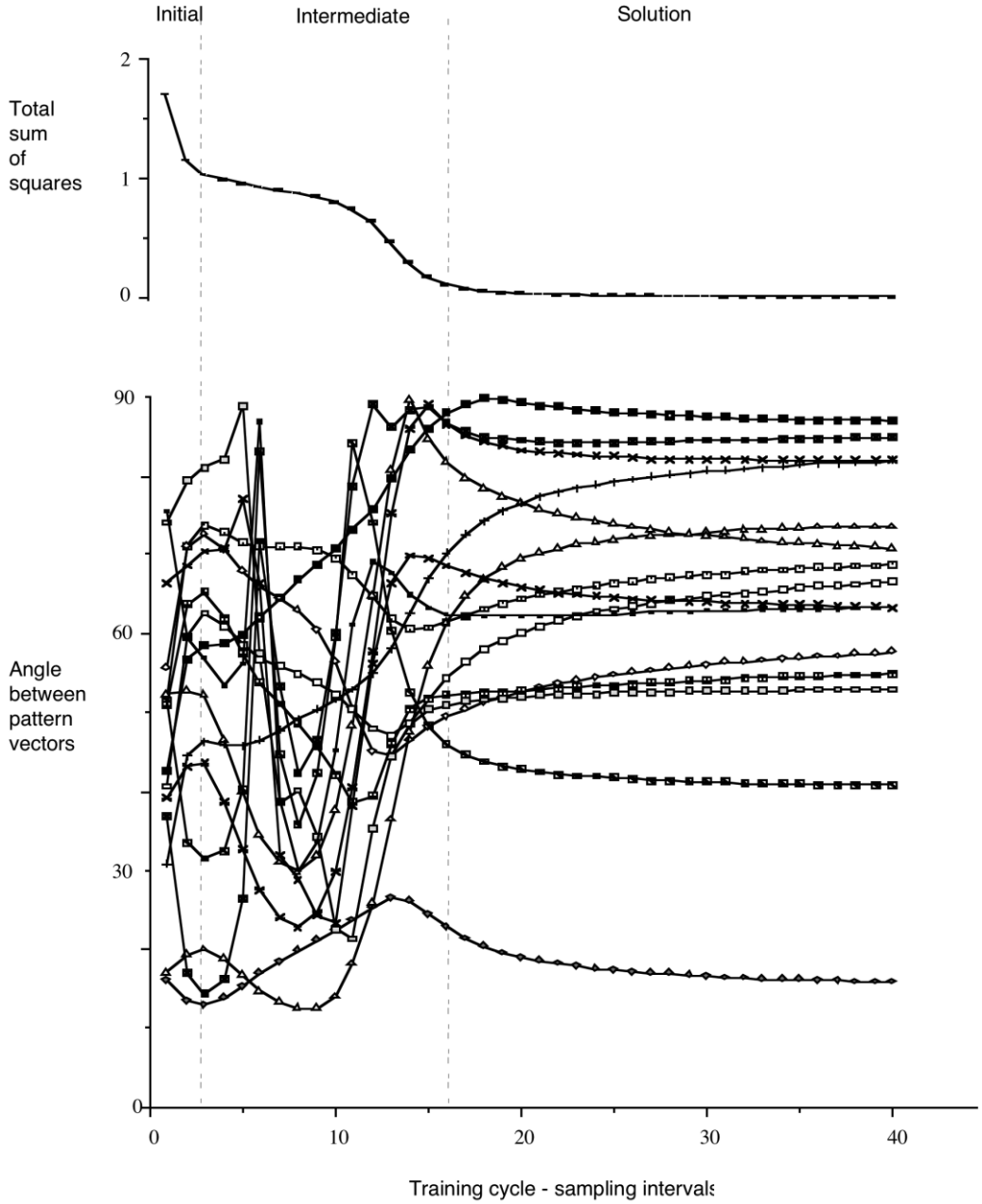


Figure 2: Network illustrating a solution with no plateau stage

local minimum

- never gets off the plateau only small changes in pattern angular separations

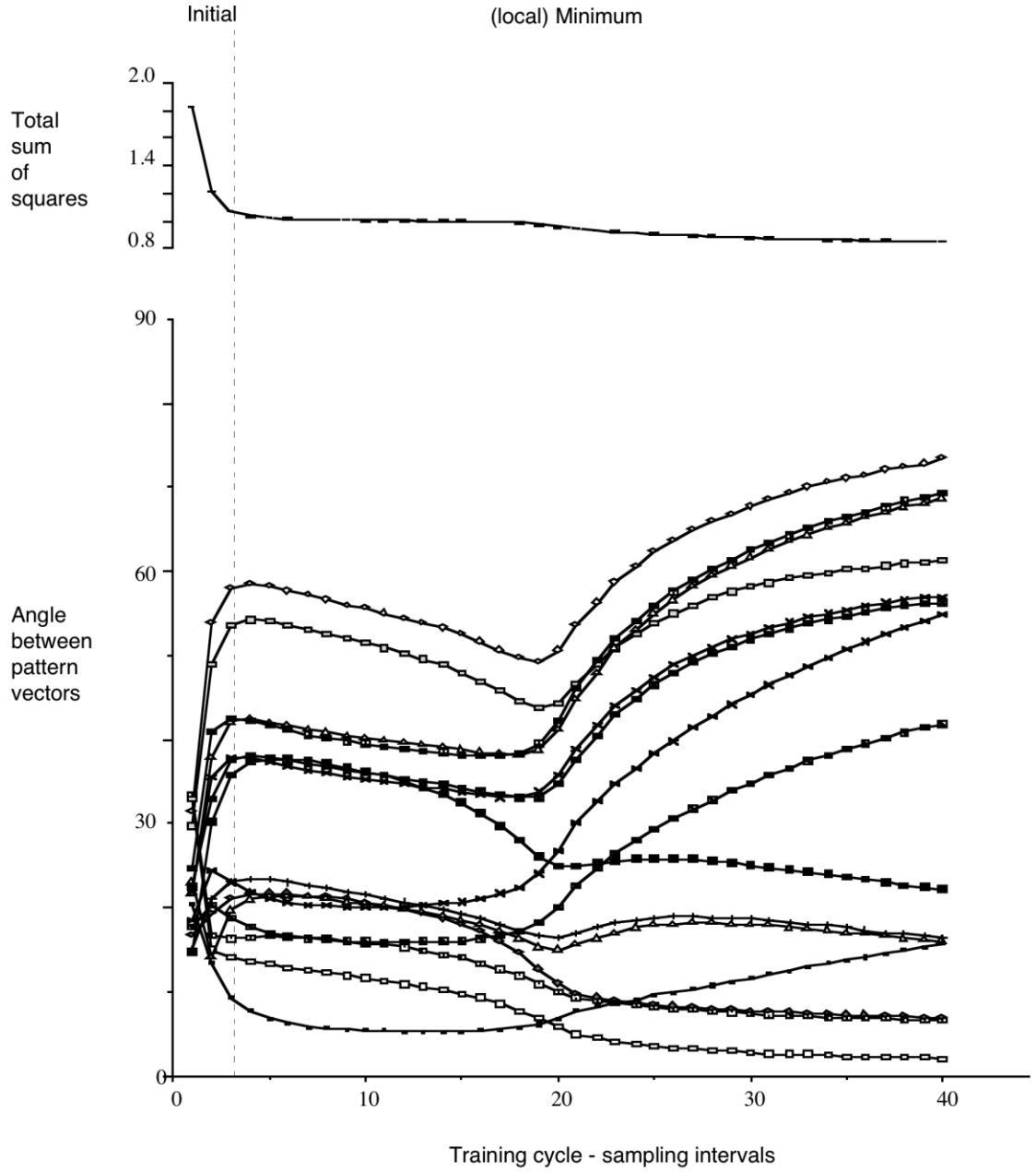


Figure 3: Network illustrating a plateau with no solution

Part of 512 patterns, 6 units.

- Any pattern here?

	Hidden Units							
Pattern	1	2	3	4	5	6	Result	Target
.								
.								
p.044	0.812	0.000	0.741	0.937	0.011	0.000	0.997	1.000
p.045	0.998	0.000	0.588	0.009	0.000	0.000	0.363	0.000
p.046	0.022	0.000	0.123	0.008	0.000	0.000	0.377	0.000
.								
.								
p.336	0.000	0.999	0.000	0.000	0.004	0.978	0.976	1.000
p.337	0.041	0.936	0.000	0.000	0.000	0.049	0.006	0.000
p.338	0.000	0.832	0.000	0.000	0.000	0.998	0.993	1.000
.								
.								

“Part of 512 patterns, 6 units.” 2

- Clearly, all the units are distinct with no redundant functionality.
- Is a candidate for the addition of at least one extra unit.
- The network never does converge if left at the current size.
- Context is important.

Pair of units	Vector angle
1 2	78.3
1 3	69.1
1 4	82.6
1 5	80.5
1 6	89.8
2 3	89.7
2 4	79.5
2 5	81.5
2 6	75.8
3 4	84.5
3 5	87.3
3 6	89.9
4 5	79.9
4 6	89.8
5 6	88.4

Neural knitworks

- http://harbaugh.uoregon.edu/Brain/index.htm
- "It took a year to knit, and the result is astonishing. The cortex of Norberg's larger-than-life brain has realistic folds, while the internal structure is correct down to the nearest stitch. All the parts are properly connected, as can be revealed by undoing a well-concealed zip that connects the two hemispheres." *New Sci. 20/28 Dec. 08*

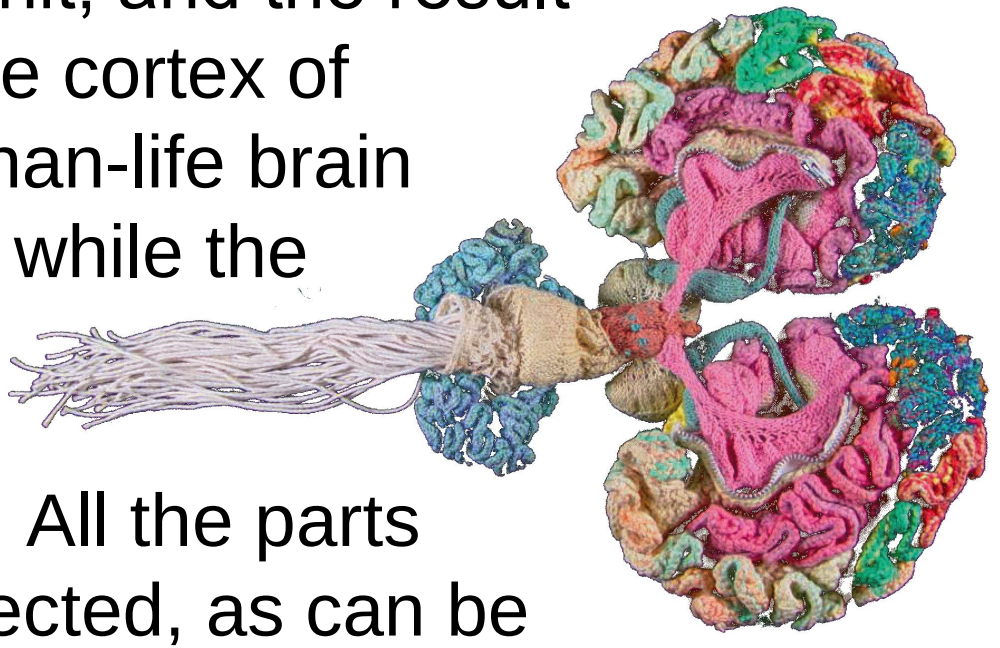
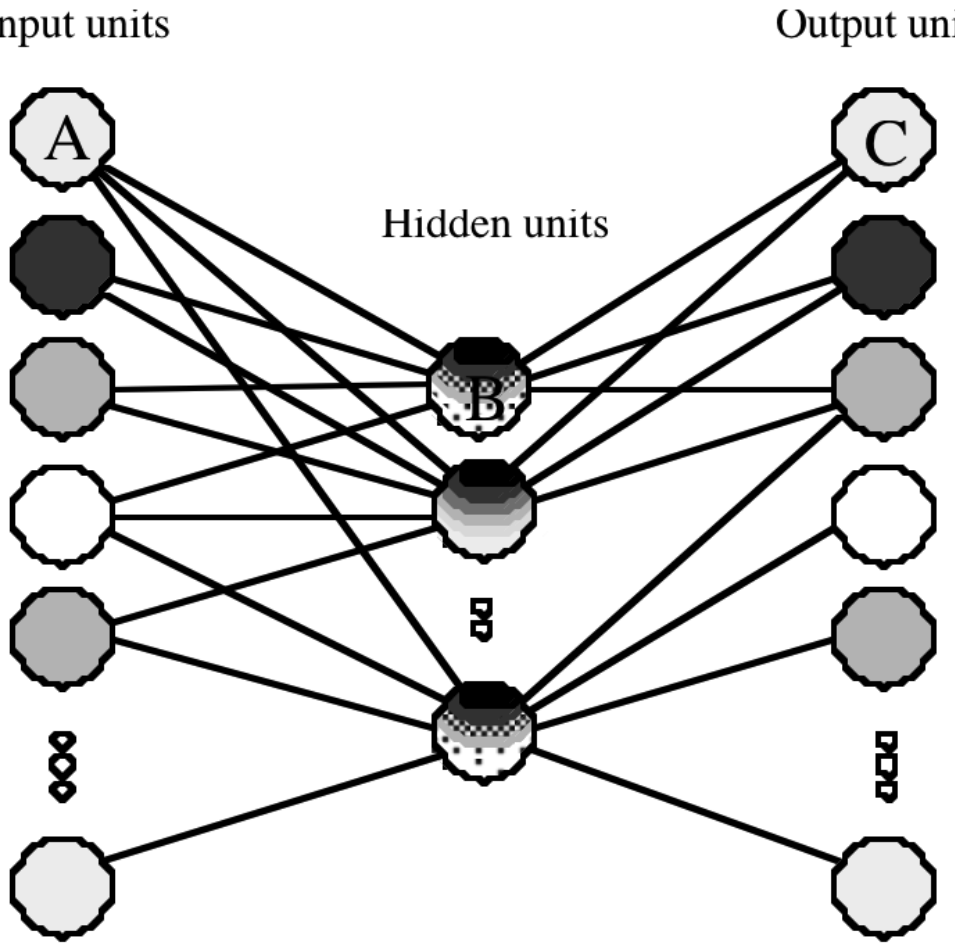


Image compression

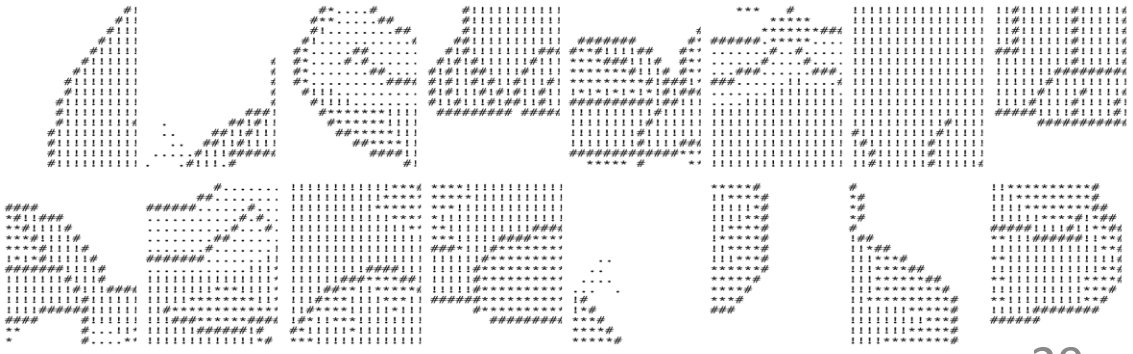
- Autoassociative network
- Input = Output
- Hidden unit activations represent a compressed image
- 256 input/output
- x hidden units



Network connections shown are representative only.
Activation of units is represented by degree of shading.

Image components

- 64x64 image
- Split into 16 16x16 non-overlapping patches
- Ensure generalisation – little similarity of patches
- For transmission over a slow connection, the neural network weights could be transmitted as a fixed initial cost, then the compressed representation formed by the hidden units could be transmitted: the activation of hidden units for the image



Units, image quality & min. angles

Number of units	Total sum of squares	Minimum angle	Compression ratio	image
16	33.0	46	16	excellent
14	43.0	46	18	
12	40.7	47	21	
11	41.4	55	23	good
10	48.3	60	26	ok
9	52.1	61	28	
8	71.0	61	32	
7	97.0	61	37	
6	125.0	63	43	just recog.
5	152.0	65	51	not recog.
4	188.0	66	64	
3	288.9	69	85	nothing

Sample results: units/compr./qual.

16u/16/exc.

12u/21/good

8u/32/ok

Sample results continued

- Networks trained for 200 presentations of the 16 patches
- Potential improvements
  - Overlapping patches
  - More than 1 original image

6u/43/bad

Compression & distinctiveness

Using the distinctiveness of units:

- eliminate unnecessary units
  - maximise compression for a given image quality
- widen degree of distinctiveness (remove significant units)
  - compression at the expense of image quality
- narrow degree of distinctiveness (and add units)
  - improve image quality at the expense of compression