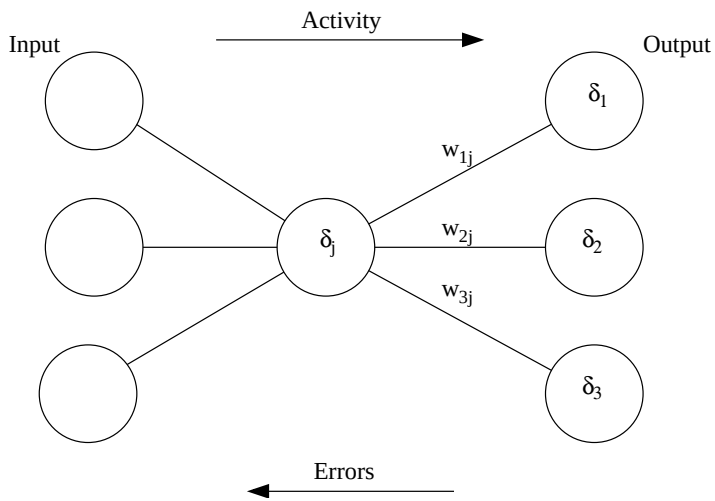
Backpropagation

Learning in multi-layer perceptrons

How to solve the credit assignment problem? i.e. what is the "delta" for hidden units, that have no desired output?

$$\delta_j = g'(h_j) \sum_i w_{ij} \delta_i$$

h_j is total input to unit j; g' is 1st derivative of activation function.
See extra handout.



No guarantee (unlike PCT) that this will converge due to local minima.

Autograd tools

1. We would like to find $\frac{\partial E}{\partial w_i}$ for every weight in the network.
2. Autograd tools can now do the hard work for us. They do not calculate the symbolic derivative.
3. Assumes that our function E is composed of more primitive functions that it can take the derivative of.
4. (R does not have this, but relies on a python package)
5. `https://en.wikipedia.org/wiki/Automatic_differentiation`

Example: MNIST http://yann.lecun.com/exdb/mnist/

https://github.com/cazala/mnist

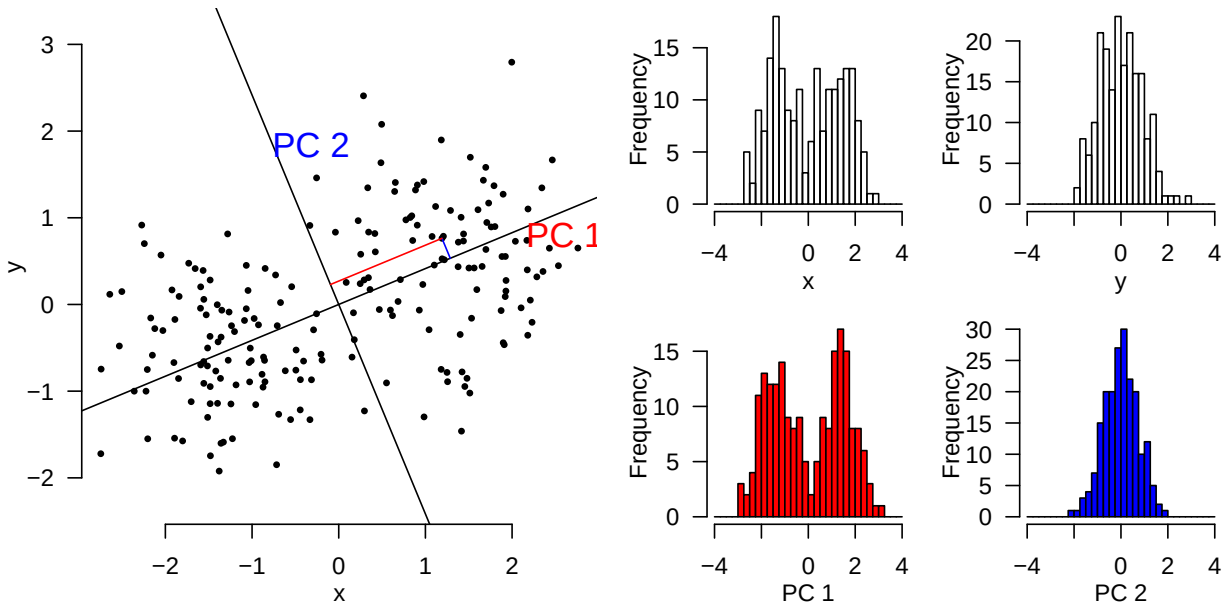
- The “hello world” of deep networks.
- MNIST ZIP Code handwritten images.
- Input 28x28 pixel images [0,255].
- Output: label as integer classes [0-9]
- Training set: 60,000 samples
- Test set: 10,000 samples

What does 784-D space look like?

Dimensionality reduction recap.

Dimensionality reduction / PCA

What: Maximise variance of encoding.
How: Eigenvectors of covariance matrix of inputs. Fractions of variance given by eigenvalues.



Multidimensional scaling (MDS)

- For each point X_i in some high-dim space, we have an equivalent point $Y_i = (y_{i1}, \ldots y_{id})$ in some low (d= 2 or 3) dim space.

$$o_{ij} = dist(X_i, X_j) \qquad \text{(fixed)}$$

$$d_{ij} = dist(Y_i, Y_j) = (\sum_{k=1}^{d} (y_{ik} - y_{jk})^2)^{0.5}$$

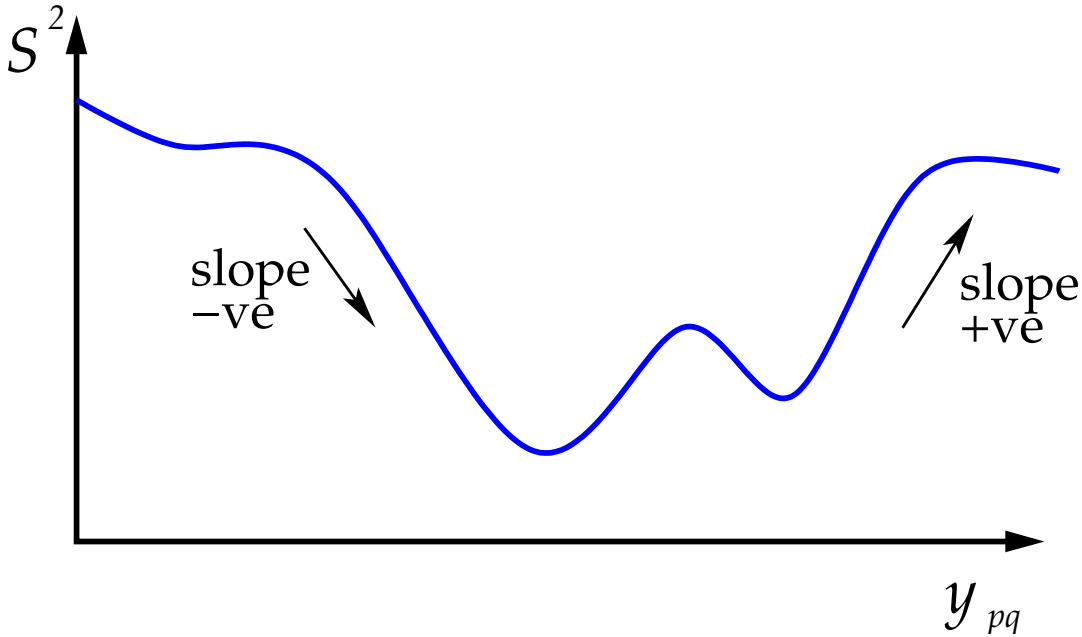
- Project down into lower dimension so that $o_{ij} \approx d_{ij}$.
- o_{ij} are fixed, so we vary points y_{ij} to minimise stress term:

$$S^2 = \frac{\sum_i \sum_j (d_{ij} - o_{ij})^2}{\sum_i \sum_j o_{ij}^2}$$

MDS: Minimisation of Stress term

- Starting from some guess for initial points o_{ij}, calculate S^2 and then evaluate gradient for each parameter y_{pq}.

$$\Delta y_{pq} = -\alpha \frac{\partial S^2}{\partial y_{pq}}$$

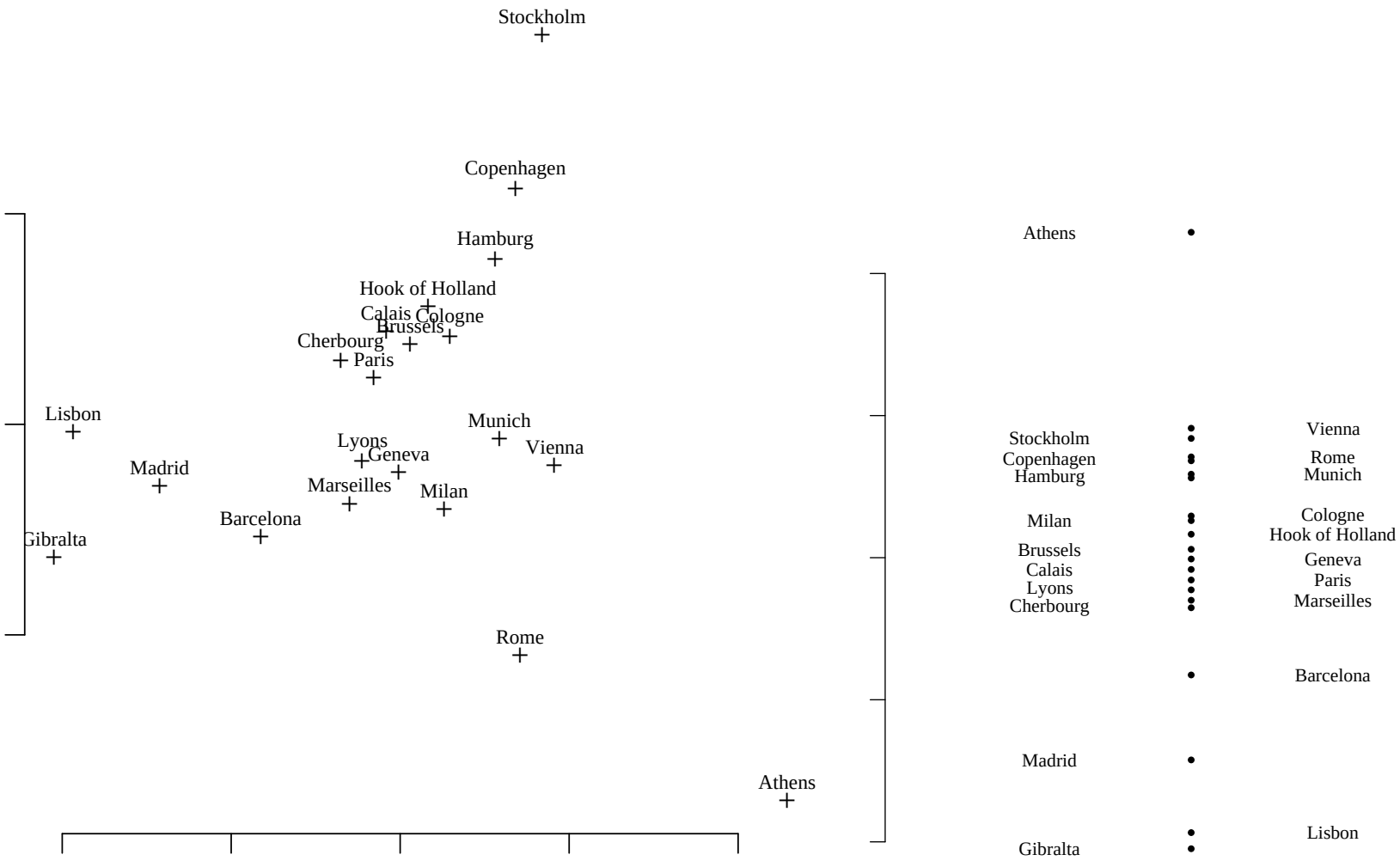


- Iteratively update y_{pq} until local minimum (gradient is zero).

Relationship between MDS and PCA

- MDS and PCA are equivalent when using Euclidean distance measure with stress measure: $S^2 = \sum_{i,j}(d_{ij} - o_{ij})^2$.
- Other stress measures can be used (e.g. Sammon) to emphasise certain aspects of data.
- Non-metric versions (NMDS) do not consider absolute distances, but preserve only ranking of distances.

MDS example: European distances



t-SNE: t-distributed stochastic neighbour embedding

1. Probabilistic approach, with distances scaled by local density of data (perplexity). Measure $p(i,j)$ as probability of being neighbours in high-D space, and fix. $q(i,j)$ is adjusted in low-D space by gradient descent on KL divergence.
2. Computationally expensive (UMAP has mathematical and computational advantages). Used in genomics a lot, but beware of limitations about intepretation. essential reading: Wattenberg et al (2016) *How to use t-SNE effectively* `https://distill.pub/2016/misread-tsne/`).

So, what does 784D space look like?

Excellent online resource for visualisation:
`http://colah.github.io/posts/2014-10-Visualizing-MNIST/`

Classification: modifications

Three tweaks:

1. **1 HOT encoding** of training signal
2. **Softmax** function on output layer:

$$y_i = \frac{\exp(z_i)}{\sum_{j=1}^{N} \exp(z_j)}$$

3. **Cross-entropy** loss function:

$$E = -\sum_{i=1}^{N} t_i \log(y_i)$$

for comparing two probability distributions. Minimal ( ) when t=y.

Some more definitions

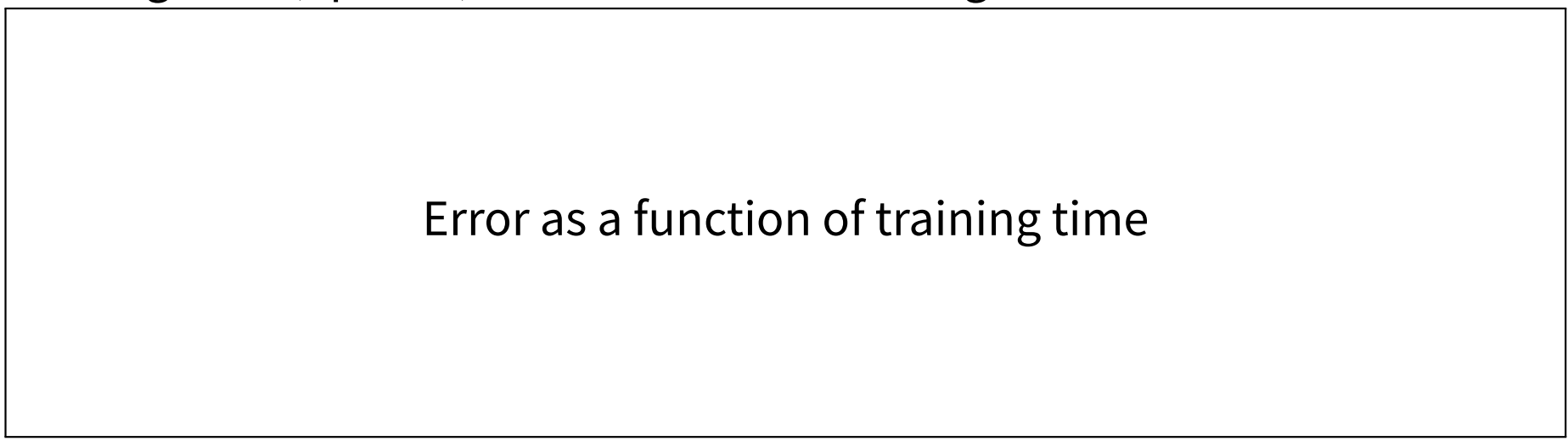
1. **flattening images** 2D image structure collapsed into 1D vector
2. **iteration** presentation of one input vector
3. **mini batch** grouping together input vectors and then learning
4. **epoch** one presentation of all input vectors (in training set)
5. **training set vs test set** Used for learning vs testing.

MNIST: 1 vs 2 hidden layers

See course home page for Rmd and HTML.

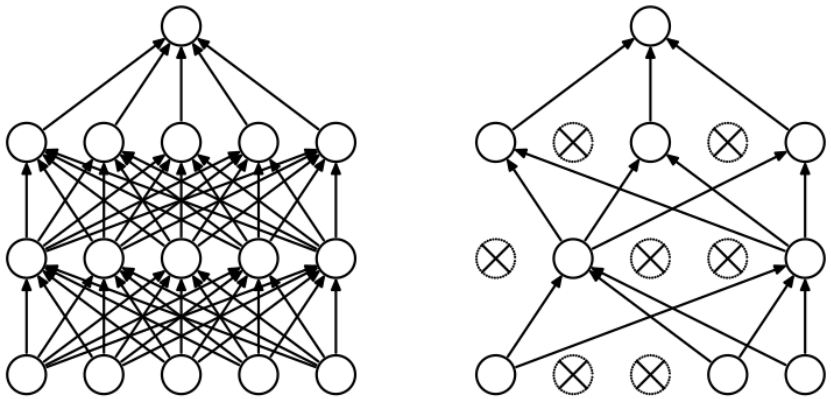
How to assess for over fitting vs under fitting

- Where is the “Goldilocks spot”?
- Run on **validation set** during learning. Plot error as a function of training time (epochs). Assess after on training set.



- Other approaches (k-fold validation) also feasible. Be careful about time-dependencies!

Dropout (Srivastava et al 2014, fig 1)



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- **Dropout** introduced as a way to reduce overfitting. Bank-teller analogy.
- During training, on each iteration silence some fraction of units.
- Implemented during training:
  1. once the activation of a layer calculated, set output of half of neurons to zero.
  2. Double the activation of rest of the neurons.

Additional terms

Many additions have been suggested to back-propagation to help prevent e.g. overfitting, local minima:

- Weight decay/ L2regularization:

$$E = \frac{1}{2}(t - y)^2 + \beta \sum_i w_i^2$$

- Momentum: add history to weight changes:

$$\Delta w_{ij}(t+1) = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t)$$

  where $0 < \alpha < 1$.

Several other methods available (conjugate gradient, rmsprop, adam) that improve convergence.

Hyper parameters

How do you decide on the hyper-parameters of the model?

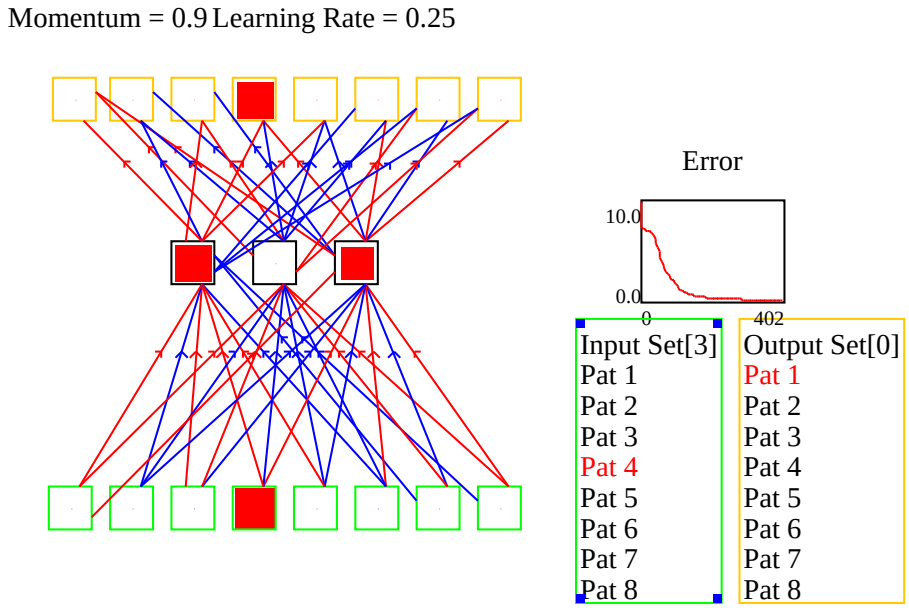
1. Network architecture: how many hidden units? how many hidden layers?
2. Learning rates (and other parameters)
3. Error functions
4. Feature selection
5. Data set size

One approach is to overfit then regularize. Sophisticated methods exist, but often more pragmatic approaches taken.

Multi-layer perceptrons

- No guarantee that solution, even if one exists, can be found.
- Training can take a long time.
- Overlearning of the decision surface to the training data. Network acts as a lookup table, with poor generalisation.
- Role of hidden units: act as feature detectors, like in XOR problem.
- How many hidden units? Too few may cause a bottleneck, but too many and the network may not generalise.

Encoder networks



Inputs: 1 of N units active. Task: reproduce input at output layer, passing through a “bottleneck” hidden layer.

After 400 epochs, activation of hidden units:

Pattern	Hidden units			Pattern	Hidden units		
1	1	1	1	5	1	0	0
2	0	0	0	6	0	0	1
3	1	1	0	7	0	1	0
4	1	0	1	8	0	1	1

Also called “self-supervised” networks.

Projects input onto first M (=3) PCs.

Application: compression. Local vs distributed representations.

Image compression example

Cottrell et al (1987) applied 64-16-64 network with quantization (8 bits per hidden unit) to patches of images. 4:1 compression ratio if input images also 8 bit. Figure 9:

Figure 9. A: The original image of the Intelligent Systems Group (ISG) at UCSD. B: The reconstructed image using eight bits (or 256 quantization values), representing 2 bits/pixel, resulting in NMSE of 0.413%.