EECS 545: Machine Learning

Lecture 12. Neural Networks

Honglak Lee

2/16/2011

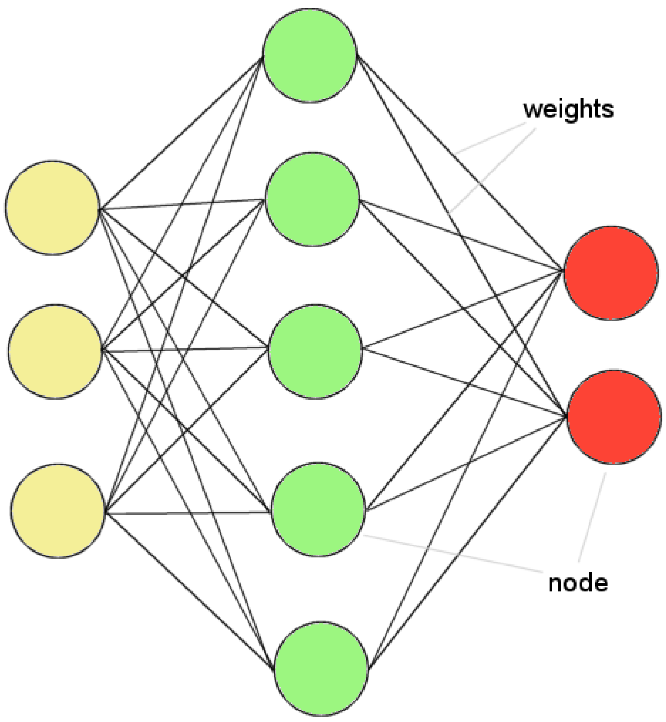
Outline

- Overview of Neural Networks
- Formal Definition
  - Feed-forward neural networks
- Probabilistic interpretation
- Training algorithm for neural networks
  - Backprogation

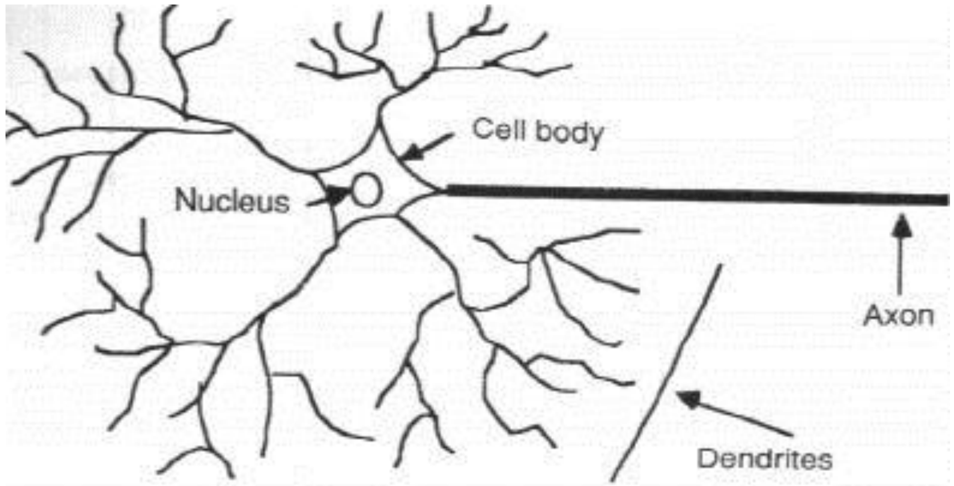
ANNs – The basics

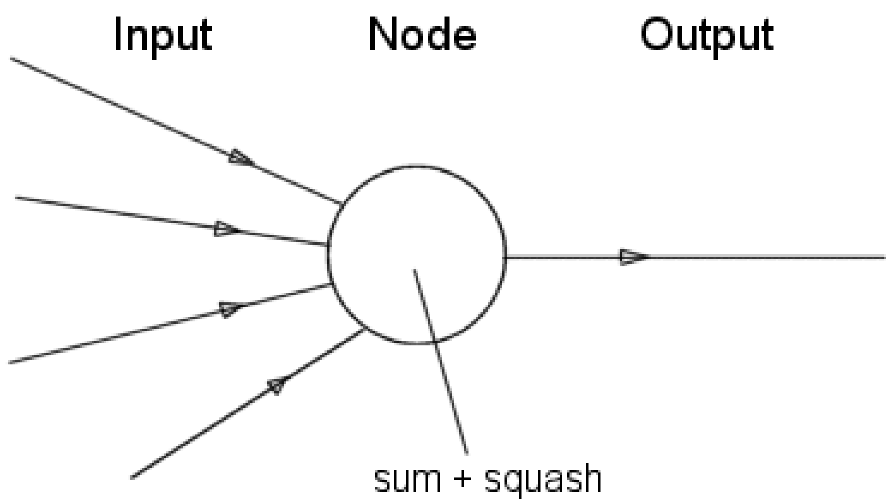
- ANNs incorporate the two fundamental components of biological neural nets:

1. Neurones (nodes)
2. Synapses (weights)

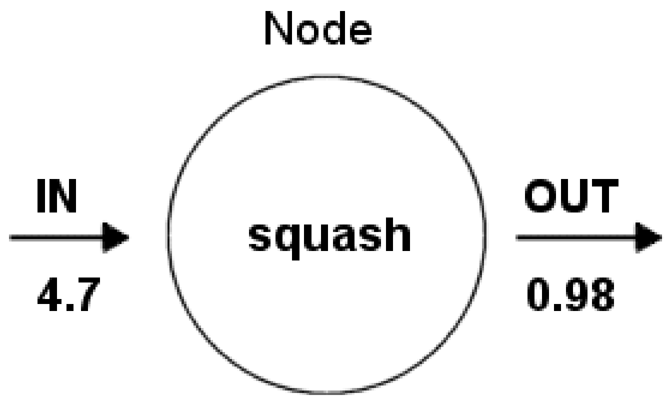


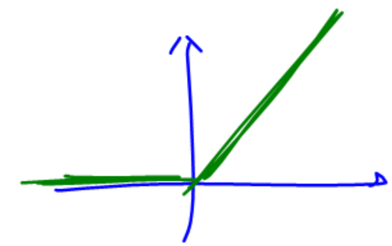
Neurone vs. Node





Structure of a node



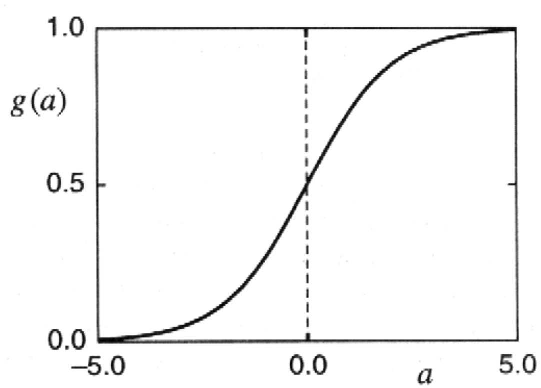


$\tanh = \tanh(s)$

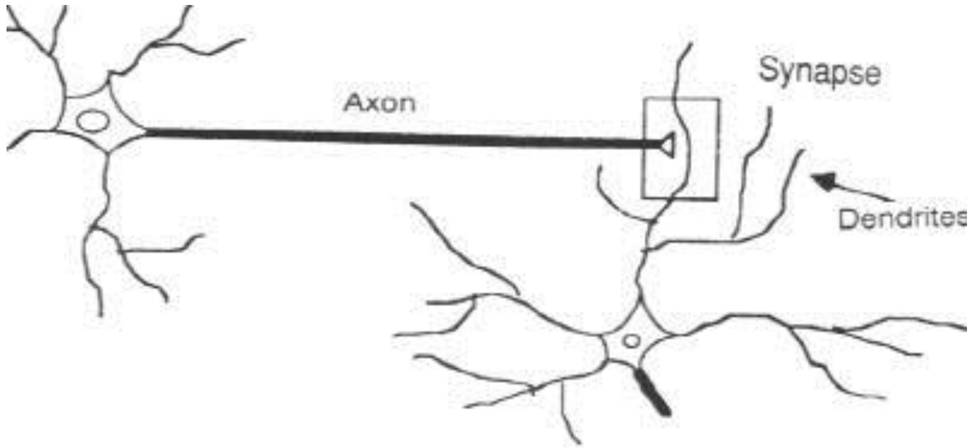
rectified linear$(s) = \max(s, 0)$

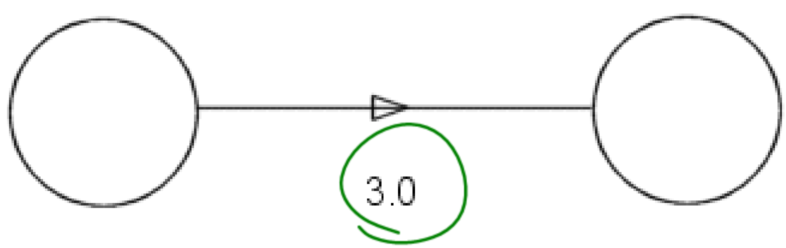
$\text{sigmoid}(s) = \dfrac{1}{1+\exp(-s)}$

Squashing function limits node output:



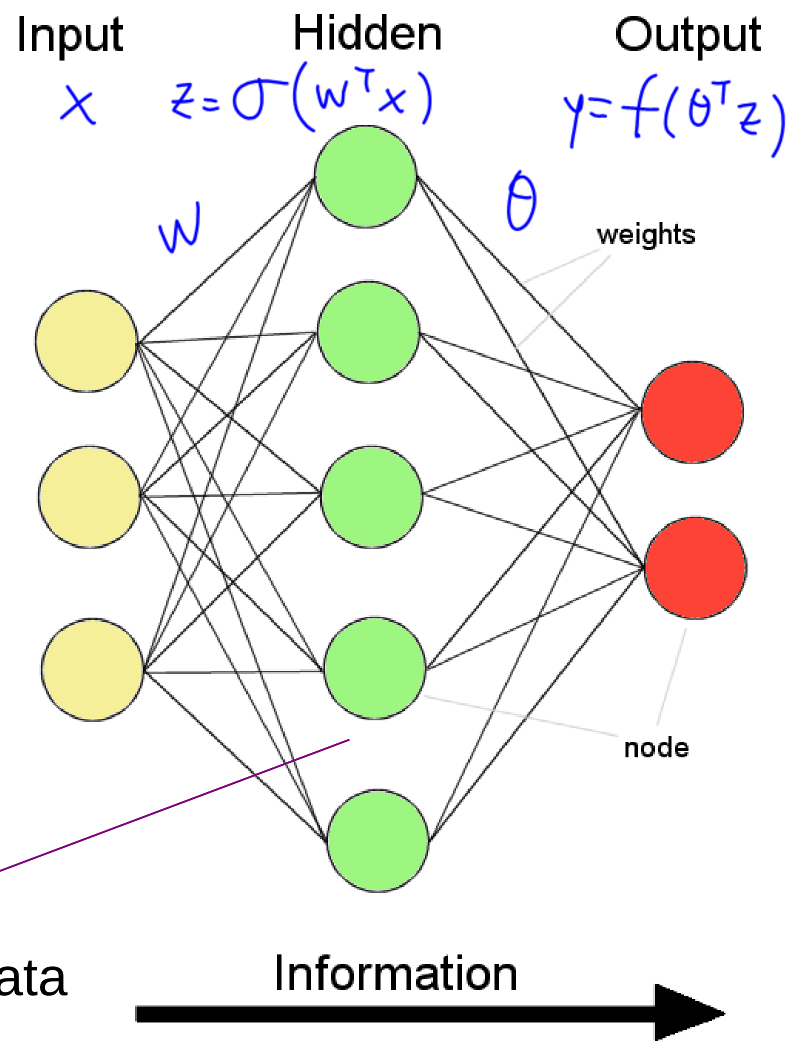
Synapse vs. weight





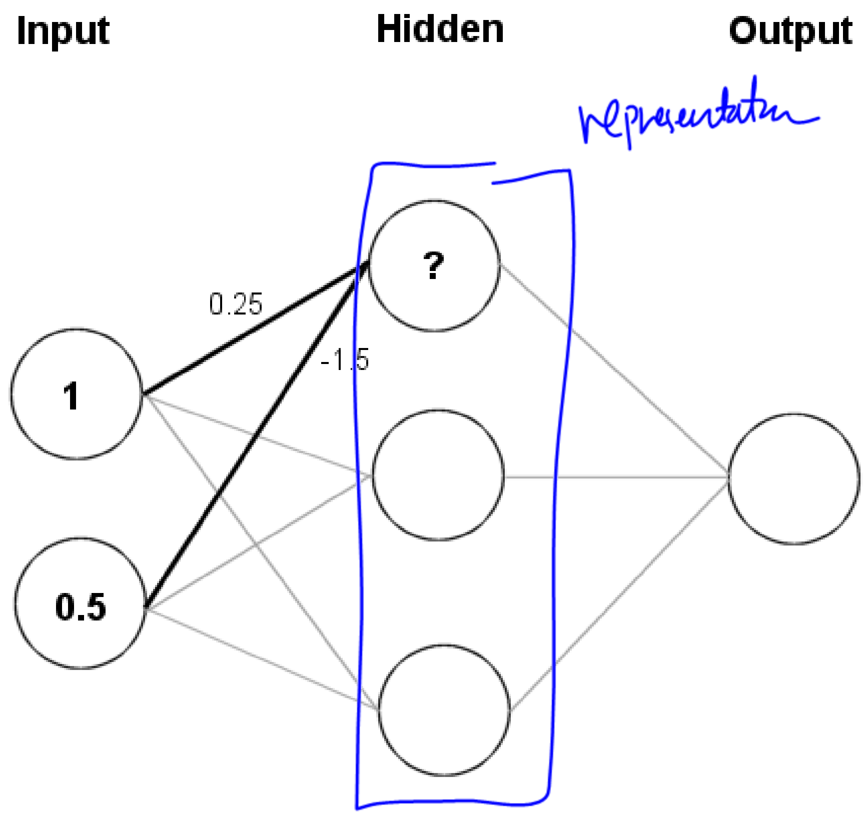
Feed-forward neural nets

- Information flow is unidirectional
  - Data is presented to Input layer
  - Passed on to Hidden Layer
  - Passed on to Output layer
- Information is distributed
- Information processing is parallel



Internal representation (interpretation) of data

Feeding data through the net



$(1 \times 0.25) + (0.5 \times (-1.5)) = 0.25 + (-0.75) = -\mathbf{0.5}$

Squashing: $0 \leq \dfrac{1}{1+e^{0.5}} = 0.3775 \leq 1$

Representation

- Data is presented to the network in the form of activations in the input layer
- Examples
  - Pixel intensity (for pictures)
  - Share prices (for stock market prediction)
- Data usually requires preprocessing
  - Analogous to senses in biology
- How to represent more abstract data, e.g. a name?
  - Choose a pattern, e.g.
  - 0-0-1 for “Alice”
  - 0-1-0 for “Ben”

Formal definition of Neural Networks

Introduction

- The aim is, as before, to find useful decompositions of the target variable;

$$t(\mathbf{x}) = y(\mathbf{x}, \mathbf{w}) + \epsilon(\mathbf{x})$$

- $t(x_n)$ and x_n are the observations, n = 1, . . . ,N.
- e(x) is the residual error.

Linear models

- For example, recall the (Generalized) Linear Model:

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right)$$

$\boldsymbol{\phi} = (\phi_0, \ldots, \phi_M)^\top$ is the fixed model basis.

$\mathbf{w} = (w_0, \ldots, w_M)^\top$ are the model coefficients.

For regression: $f(\cdot)$ is the identity.

For classification: $f(\cdot)$ maps to a posterior probability.

Feed-Forward Networks

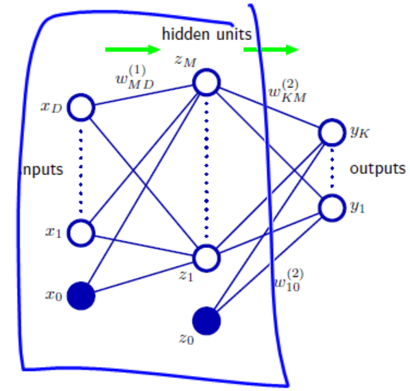
- Feed-forward Neural Networks generalize the linear model

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right)$$

  - The basis itself, as well as the coefficients w_j , will be adapted.
  - Roughly: the above function will be used twice; once to define the basis, and once to obtain the output.

Activations

- Construct M linear combinations of the inputs $(x_1, \ldots, x_D)$:



$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i$$

a_j are the activations, $j = 1, \ldots, M$.

$w^{(1)}_{ji}$ are the layer one weights, $i = 1 \ldots D$.

$w^{(1)}_{j0}$ are the layer one biases.

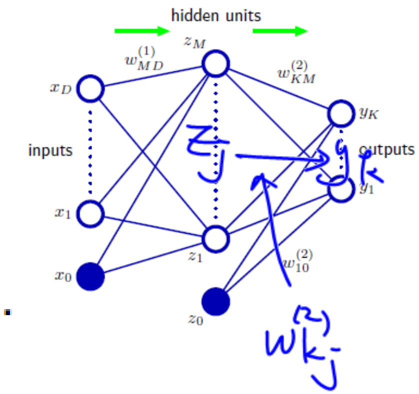
- Each linear combination aj is transformed by a (nonlinear, differentiable) activation function:

$$z_j = h(a_j)$$

h: identity
sigmoid
tanh
rectified linear

Output Activations

- The hidden outputs $z_j = h(a_j)$ are linearly combined in layer two:



$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

a_k are the output activations, $k = 1, \ldots, K$.

$w_{kj}^{(2)}$ are the layer two weights, $j = 1 \ldots D$.

$w_{k0}^{(2)}$ are the layer two biases.

- The output activations ak are transformed by the output activation function: $$y_k = \sigma(a_k)$$

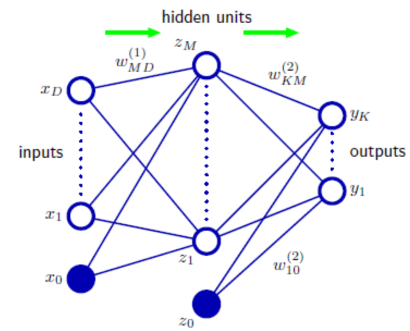
y_k are the final outputs.

$\sigma(\cdot)$ is, like $h(\cdot)$, a sigmoidal function.

Forward propagation: Complete Two-layer model

- The model $y_k = \sigma(a_k)$ is, after substituting the a_j:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\underbrace{\sum_{j=0}^{M} w_{kj}^{(2)}\, h\left(\underbrace{\sum_{i=0}^{D} w_{ji}^{(1)} x_i}_{a_j}\right)}_{a_k}\right)$$



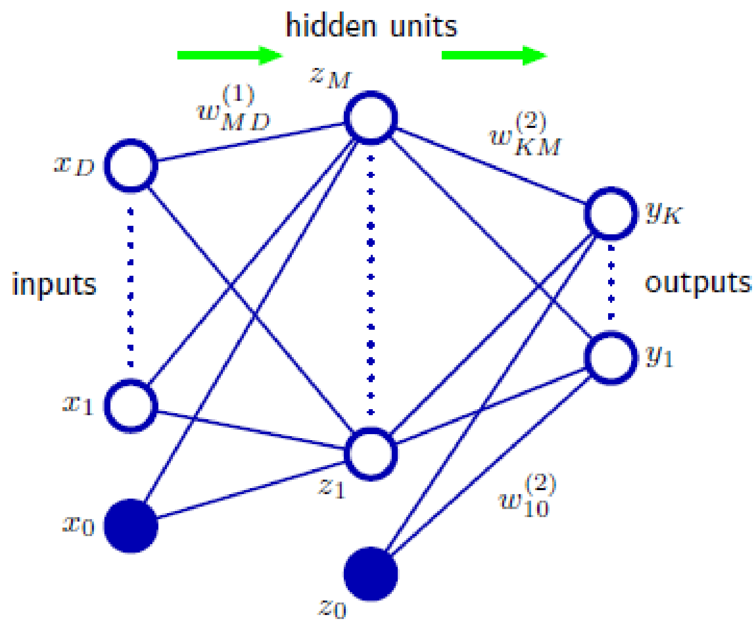
- $h(\cdot)$ and $\sigma(\cdot)$ are a sigmoidal functions, e.g. the logistic function.

$$s(a) = \frac{1}{1 + \exp(-a)} \qquad s(a) \in [0, 1]$$

  - If $\sigma(\cdot)$ is the identity, then a regression model is obtained.
  - This function evaluation is called forward propagation.

Network Diagram

- This function evaluation can be represented by a network:



Nodes are input, hidden and output units. Links are corresponding weights.

Information propagates 'forwards' from the explanatory variable $\mathbf{x}$ to the estimated response $y_k(\mathbf{x}, \mathbf{w})$.

Properties & Generalizations

- Typically $K \leq D \leq M$, which means that the network is redundant if all $h(\cdot)$ are linear.
  - K: number of output nodes (output dimension)
  - D: number of input nodes (input dimension)
  - M: number of hidden nodes (number of features)
- Network structure
  - There may be more than one layer of hidden units.
  - Individual units need not be fully connected to the next layer.
  - Individual links may skip over one or more subsequent layers.
- Networks with two or more layers are universal **approximators**.
  - Any continuous function can be uniformly approximated to arbitrary accuracy, given enough hidden units.
  - This is true for many definitions of $h(\cdot)$, but excluding polynomials.
- There may be **symmetries** in the weight space
  - different choices of w may define the same mapping from input to output.

$$x \xrightarrow{w} \underset{\sigma(wx)}{\overset{h}{\bigcirc}} \xrightarrow{\theta} y$$

$-w \qquad -\theta$

Probabilistic Interpretation

Maximum likelihood

- The aim is to minimize the residual error between $y(x_n, w)$ and t_n ($n=1,\ldots,N$).
  - Suppose that the target is a scalar-valued function, which is Normally distributed around the estimate:

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t \,\middle|\, y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

- Then, consider the sum of squared-errors

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \Big(y(\mathbf{x}_n, \mathbf{w}) - t_n \Big)^2$$

- The maximum-likelihood estimate of w can be obtained by minimization:

$$\mathbf{w}_{\text{ML}} = \min_{\mathbf{w}} \; E(\mathbf{w})$$

Maximum likelihood for precision

- Having obtained the ML parameter estimate w_{ML} , the precision, β can also be estimated. E.g. if the N observations are IID, then their joint probability is

$$p\Big(\{t_1, \ldots, t_N\}\Big|\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}, \mathbf{w}, \beta\Big) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

- The negative log-likelihood, in this case, is

$$-\log p = \beta E(\mathbf{w}_{\mathsf{ML}}) - \frac{N}{2}\log\beta + \frac{N}{2}\log 2\pi$$

The derivative $\mathrm{d}/\mathrm{d}\beta$ is $E(\mathbf{w}_{\mathsf{ML}}) - \frac{N}{2\beta}$ and so

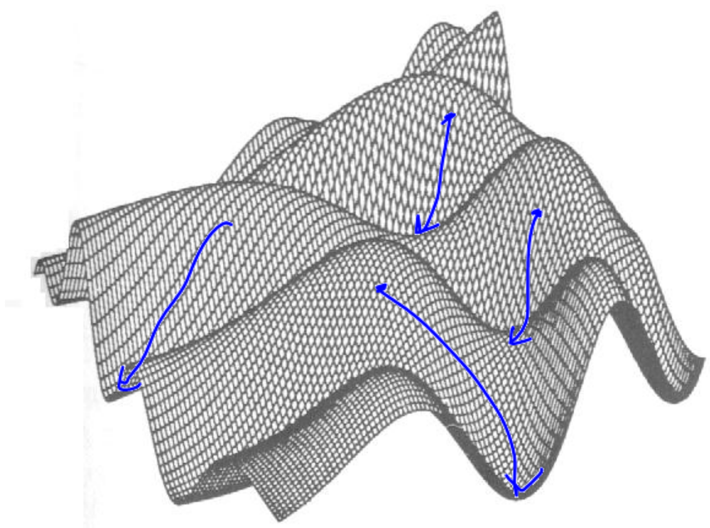
$$\frac{1}{\beta_{\mathsf{ML}}} = \frac{1}{N} 2E(\mathbf{w}_{\mathsf{ML}})$$

And $1/\beta_{\mathsf{ML}} = \frac{1}{NK} 2E(\mathbf{w}_{\mathsf{ML}})$ for K target variables.

Training Neural Networks

Training the neural networks

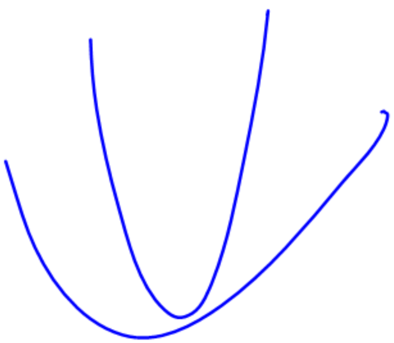
- Backpropagation
  - Requires training set (input / output pairs)
  - Starts with small random weights
  - Error is used to adjust weights (supervised learning)
- Gradient descent on error landscape



Error Surface

- The residual error E(w) can be visualized as a surface in the weight-space:

$f = x^2 + y^2$



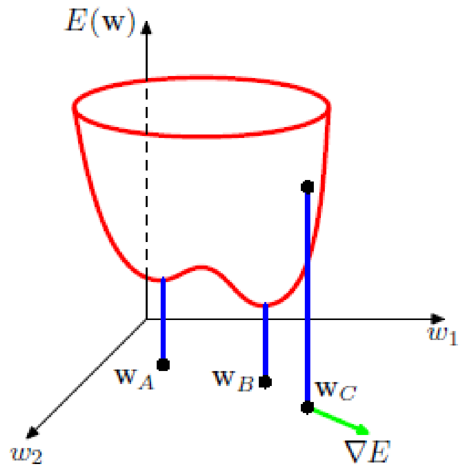
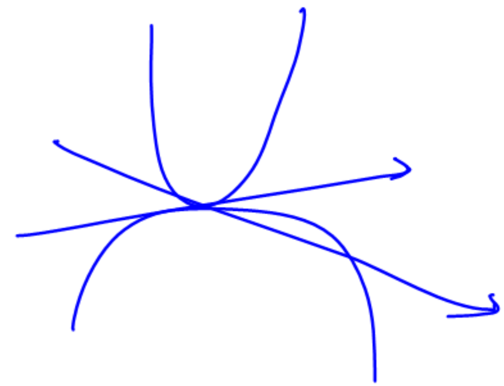


Figure: 5.5

$f = x^2 - y^2$



  - The error will, in practice, be highly nonlinear, with many minima, maxima and saddle-points.
  - There will be inequivalent minima, determined by the particular data and model, as well as equivalent minima, corresponding to weight-space symmetries.

Parameter Optimization

- Iterative search for a local minimum of the error:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$$

∇E will be zero at a minimum of the error.

τ is the time-step.

$\Delta\mathbf{w}^{(\tau)}$ is the weight-vector update.

The definition of the update depends on the choice of algorithm.

Local Quadratic Approximation

- The truncated Taylor expansion of E(w) around w:

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{b} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

  - where

$\mathbf{b} = \nabla E|_{\mathbf{w}=\hat{\mathbf{w}}}$ is the gradient at $\hat{\mathbf{w}}$.

$(\mathbf{H})_{ij} = \left.\frac{\partial E}{\partial w_i \partial w_j}\right|_{\mathbf{w}=\hat{\mathbf{w}}}$ is the Hessian $\nabla\nabla E$ at $\hat{\mathbf{w}}$.

- Gradient can be approximated as form:

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

where $\frac{1}{2}\left((\mathbf{H} + \mathbf{H}^\top)\mathbf{w} - \mathbf{H}\hat{\mathbf{w}} - \mathbf{H}^\top\hat{\mathbf{w}}\right) = \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$, as $\mathbf{H}^\top = \mathbf{H}$.

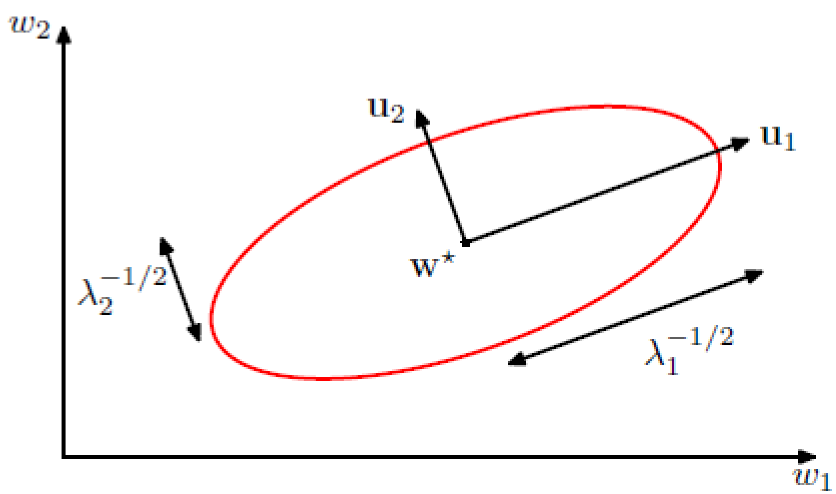
Characterization of a Minimum

- Eigenvalues λ_i of H characterize the stationary point w.

If all $\lambda_i > 0$, then $\mathbf{H}$ is positive definite ($\mathbf{v}^\top \mathbf{H} \mathbf{v} > 0$).

This is analogous to the scalar condition $\left.\frac{\partial^2 E}{\partial w^2}\right|_{w^\star} > 0$.

Zero gradient and positive principle curvatures mean that $E(\mathbf{w}^\star)$ is a minimum.



Gradient Descent

- The simplest approach is to update w by a displacement in the negative gradient direction.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right)$$

This is a steepest descent algorithm.

η is the learning rate.

This is a batch method, as evaluation of ∇E involves the entire data set.

Conjugate gradient or quasi-Newton methods may, in practice, be preferred.

A range of starting points $\{\mathbf{w}^{(0)}\}$ may be needed, in order to find a satisfactory minimum.

Error Backpropagation

Optimization Scheme

- An efficient method for the evaluation of $\nabla E(\mathbf{w})$ is needed.

  Each iteration of the descent algorithm has two stages:

  I. Evaluate derivatives of error with respect to weights (involving backpropagation of error though the network).

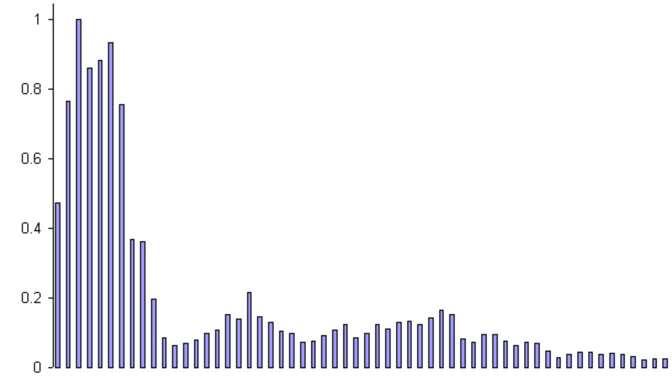
  II. Use derivatives to compute adjustments of the weights (e.g. steepest descent).

- Backpropagation is essentially **chain rule**!
- Backpropagation is a general principle, which can be applied to many types of network and error function.

Example: Voice Recognition

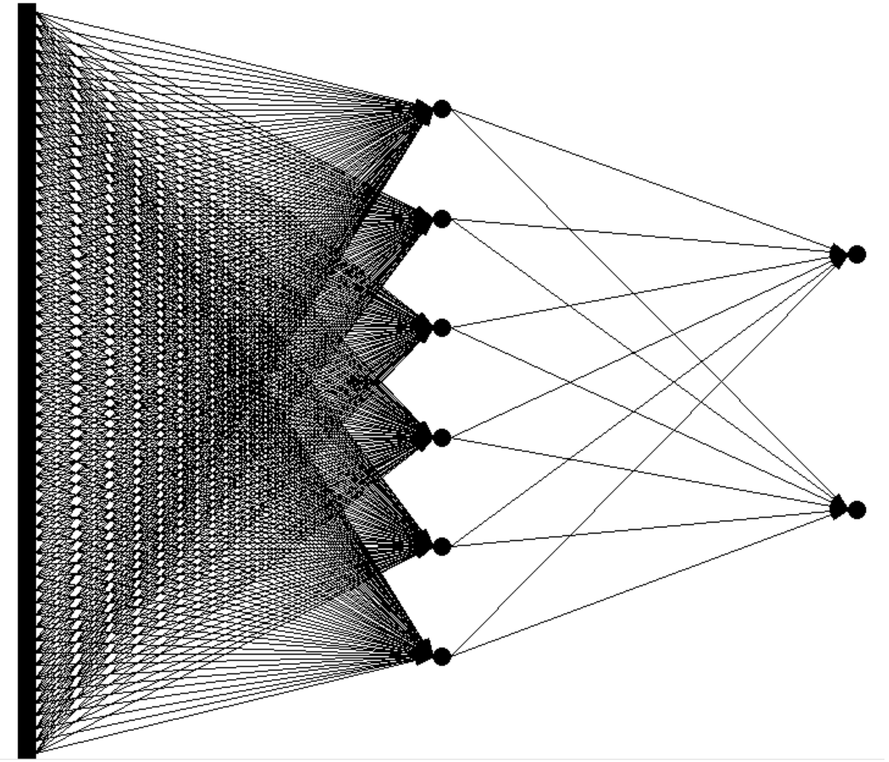
- Task: Learn to discriminate between two different voices saying “Hello”

- Data
  - Sources
    - Alice
    - Ben
  - Format
    - Frequency distribution (60 bins)
    - Analogy: cochlea



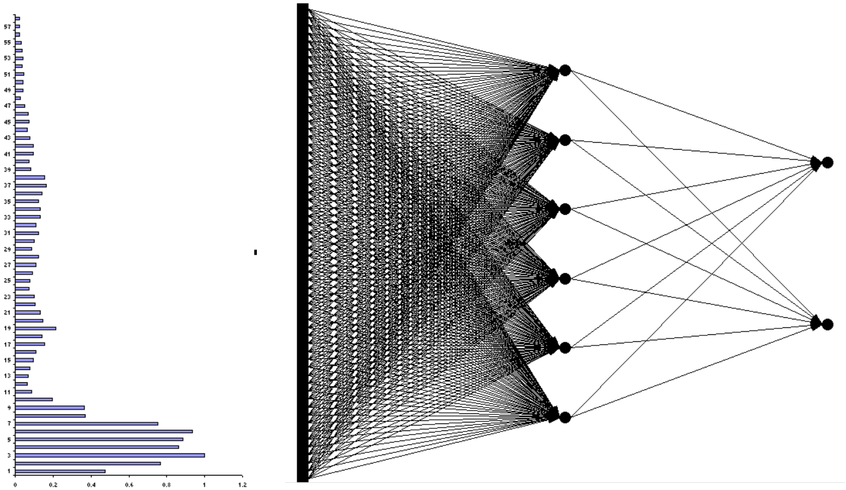
Network architecture

- Feed forward network
  - 60 input (one for each frequency bin)
  - 6 hidden
  - 2 output (0-1 for “Alice”, 1-0 for “Ben”)

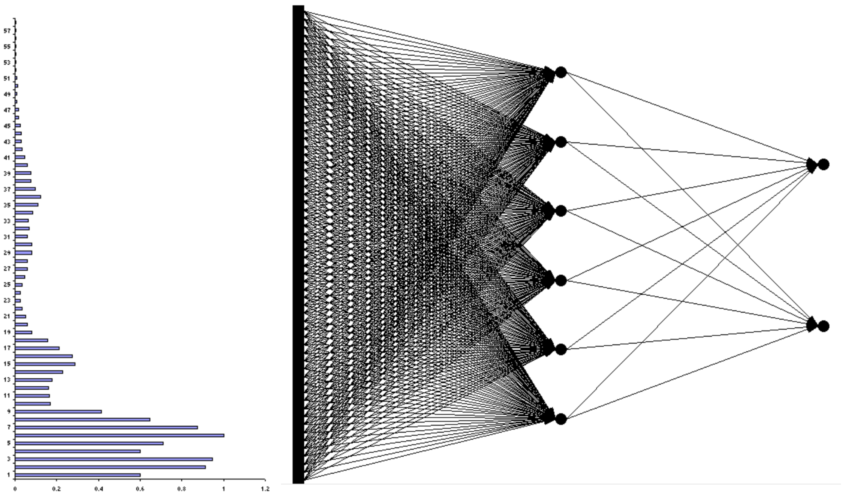


• Presenting the data

Alice

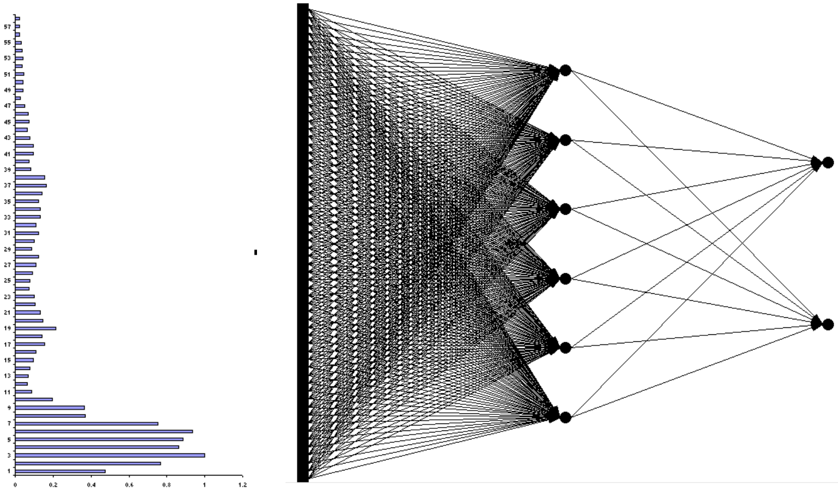


Ben



- Presenting the data (untrained network)

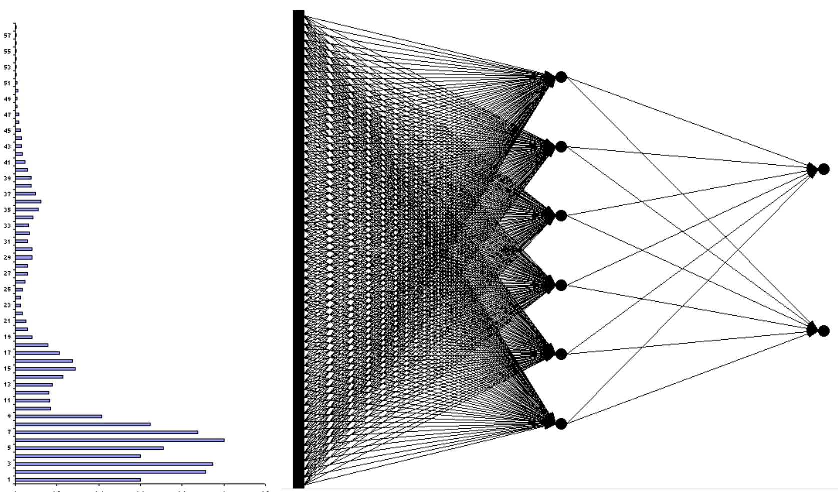
Alice



0.43

0.26

Ben

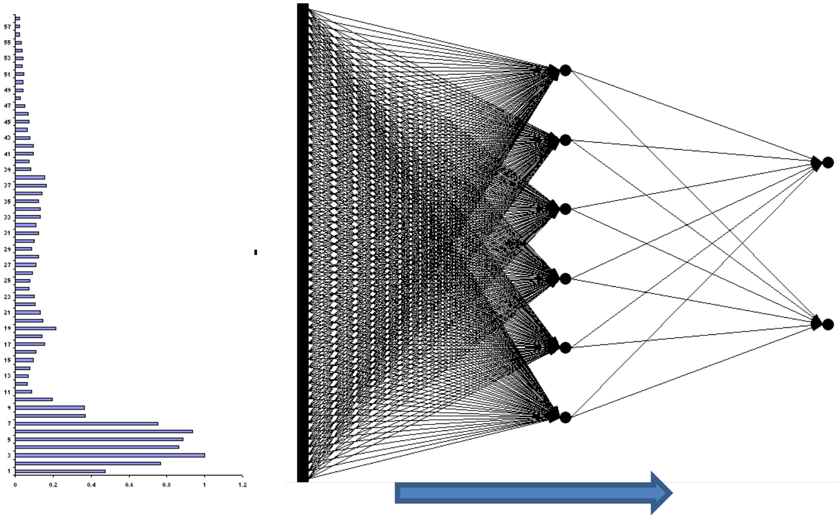


0.73

0.55

- Calculate error (squared error):

Alice



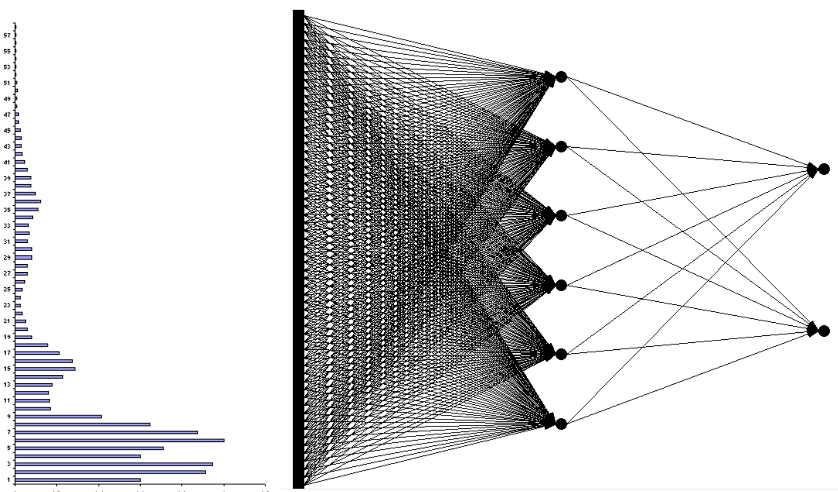
$(0.43 - 0)^2 = 0.185$

$(0.26 - 1)^2 = 0.548$

0.733

Forward-propagate: compute hidden activations and errors

Ben



$(0.73 - 1)^2 = 0.073$

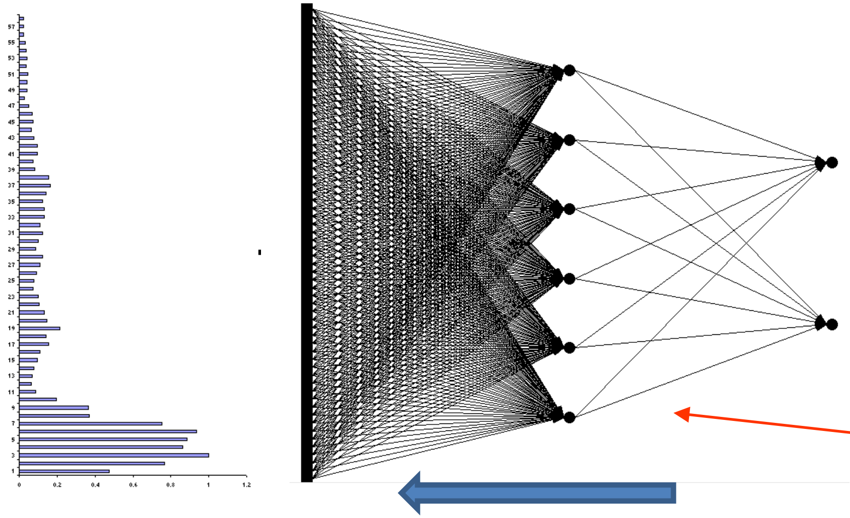
$(0.55 - 0)^2 = 0.303$

0.376

Note: Squared error was used for simple illustration, usually cross entropy is used for classification.

- Backprop error and adjust weights (squared error)

Alice



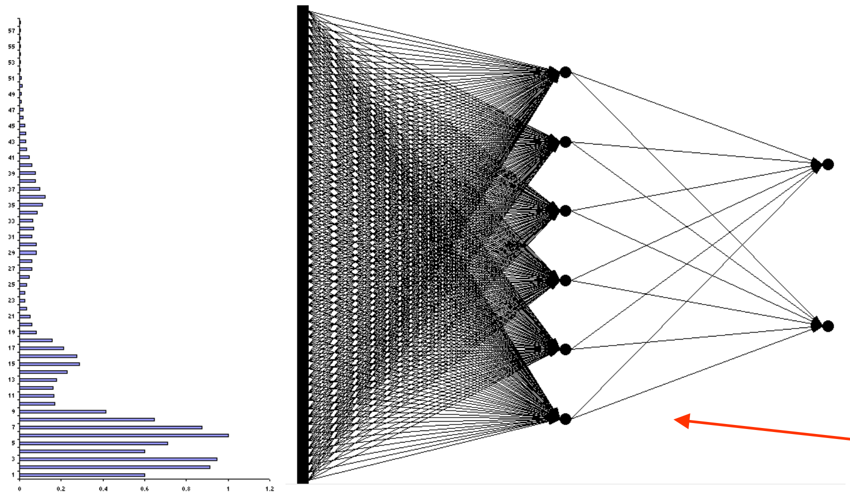
$(0.43 - 0)^2 = 0.185$

$(0.26 - 1)^2 = 0.548$

0.733

Back-propagate: compute gradient and update parameters

Ben



$(0.73 - 1)^2 = 0.27$

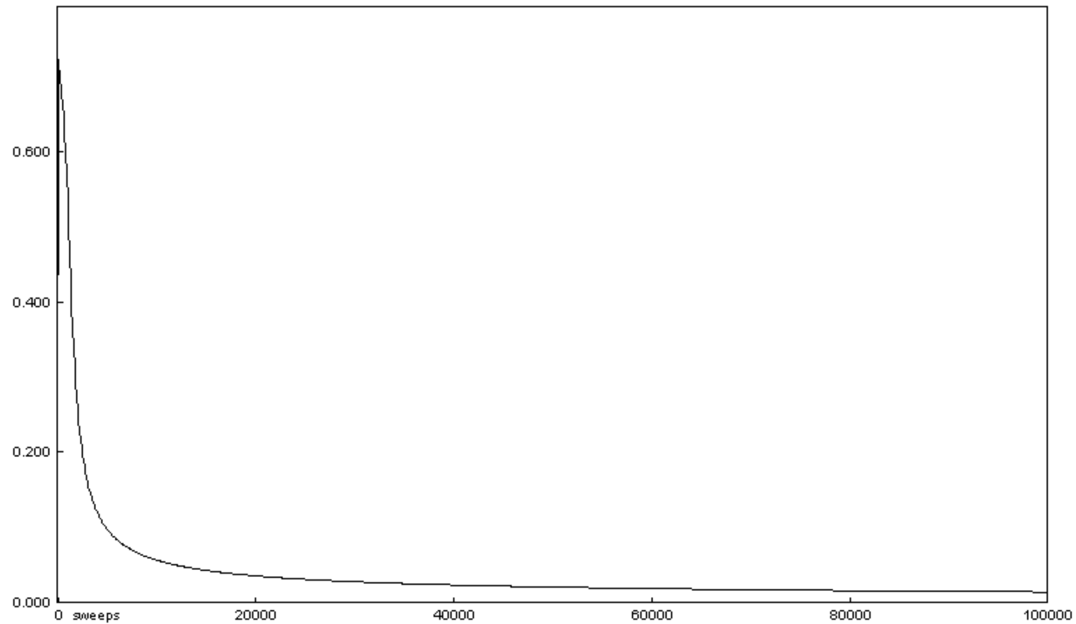
$(0.55 - 0)^2 = 0.55$

0.82

Note: Squared error was used for simple illustration, usually cross entropy is used for classification.

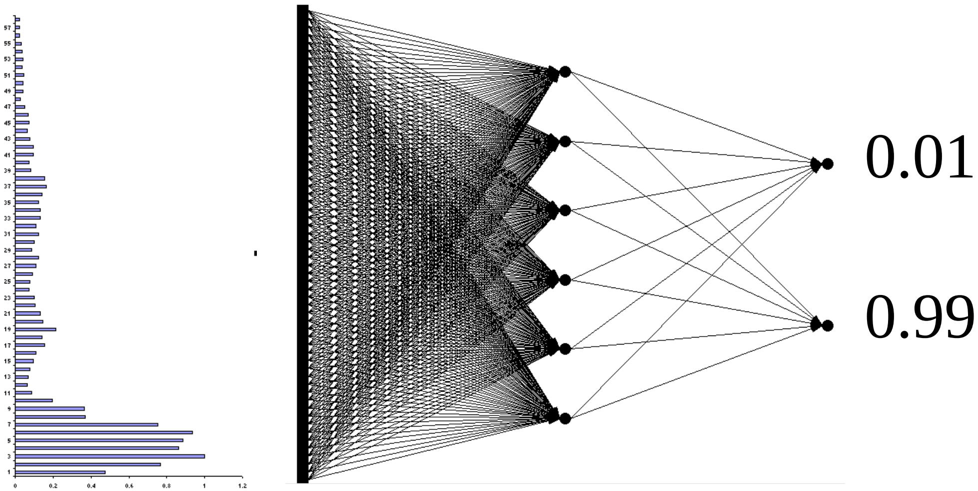
Training procedure

- Repeat process (sweep) for all training examples
  - Present data
  - Calculate error
  - Backpropagate error
  - Adjust weights
- Repeat process multiple times

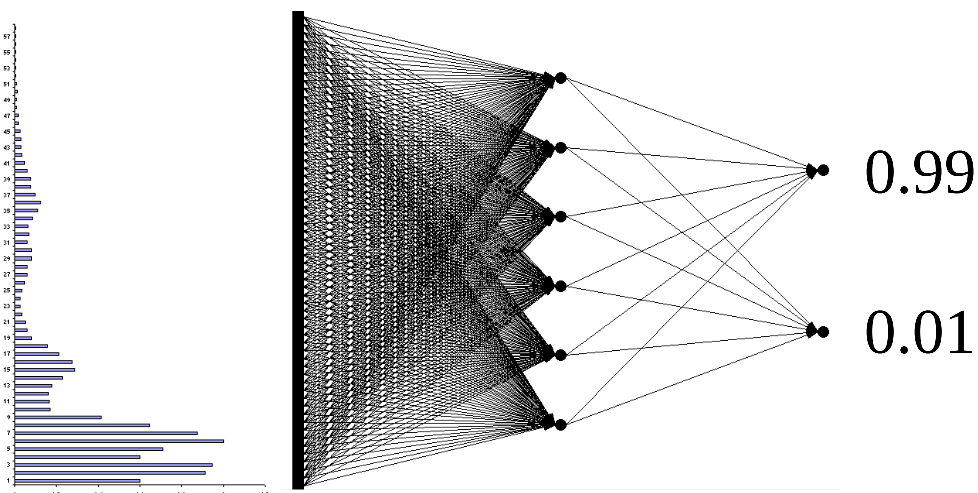


Presenting the data (trained network)

Alice



Ben



Simple Backpropagation

- The error function is, typically, a sum over the data points $E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$. For example, consider a linear model

$$y_k = \sum_i w_{ki} x_i$$

The error function, for an individual input $\mathbf{x}_n$, is

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2, \quad \text{where} \quad y_{nk} = y_k(\mathbf{x}_n, \mathbf{w}).$$

The gradient with respect to a weight w_{ji} is

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})\, x_{ni}$$

  - w_{ji} is a particular link (x_i to y_j).
  - x_{ni} is the input to the link (i-th component of $\mathbf{x}_n$).
  - $(y_{nj} - t_{nj})$ is the error output by the link.

General Backpropagation

$z_i = \sigma(a_i) \to a_j \to z(a_j)$

Recall that, in general, each unit computes a weighted sum:

$$a_j = \sum_i w_{ji} z_i \quad \text{with activation} \quad z_j = h(a_j). \qquad (5.48,5.49)$$

For each error-term:

$$\frac{\partial E_n}{\partial w_{ji}} = \underbrace{\frac{\partial E_n}{\partial a_j}}_{\equiv \delta_j} \frac{\partial a_j}{\partial w_{ji}} \qquad (5.50)$$

Given $\frac{\partial f}{\partial a_j}$,

compute $\frac{\partial f}{\partial a_i}$

So, from 5.48:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \qquad (5.53)$$

In the network:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad \text{where } j \to \{k\} \qquad (5.55)$$

Algorithm:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \quad \text{as} \quad \frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \qquad (5.56)$$

Recursion

$$\frac{\partial f}{\partial a_i} = \sum_j \frac{\partial a_j}{\partial a_i} \boxed{\frac{\partial f}{\partial a_j}}$$

given.

$$a_j = \sum_i w_{ji}\, h(a_i)$$

$$\frac{\partial a_j}{\partial a_i} = w_{ji}\, h'(a_i)$$

Backpropagation Algorithm

The formula for the update of a given unit depends only on the 'later' (i.e. closer to the output) layers:

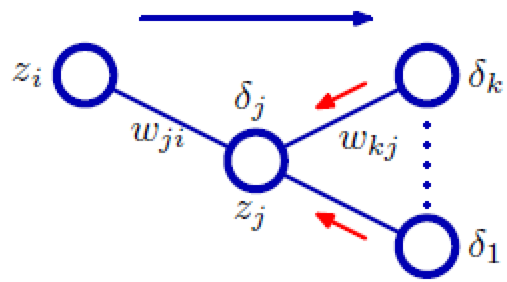


Figure: 5.7

Hence the backpropagation algorithm is:

- Apply input **x**, and forward propagate to find the hidden and output activations.
- Evaluate δ_k directly for the output units.
- Back propagate the δ's to obtain a δ_j for each hidden unit.
- Evaluate the derivatives $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$.

Computational Efficiency

- The back-propagation algorithm is computationally more efficient than standard numerical minimization of E_n.
  - Suppose that W is the total number of weights and biases in the network.
  - Backpropagation: The evaluation is $O(W)$ for large W, as there are many more weights than units.
  - Standard approach: Perturb each weight, and forward propagate to compute the change in E_n. This requires $W \times O(W)$ computations, so the total complexity is $O(W^2)$.