COMP9444: Neural Networks

2. Perceptrons and Backpropagation

Outline

- Neurons – biological and artificial
- Perceptrons
- Linear separability
- Multi-layer neural networks
- Backpropagation
- Variations on Backprop
- Training tips

Biological Neurons

The brain is made up of neurons (nerve cells) which have

- a cell body (soma)
- dendrites (inputs)
- an axon (output)
- synapses (connections between cells)

Synapses can be exitatory or inhibitory and may change over time.

When the inputs reach some threshhold an action potential (electrical pulse) is sent along the axon to the outputs.

Artificial Neural Networks

(Artificial) Neural Networks are made up of nodes which have

- inputs edges, each with some weight
- outputs edges (with weights)
- an activation level (a function of the inputs)

Weights can be positive or negative and may change over time (learning).

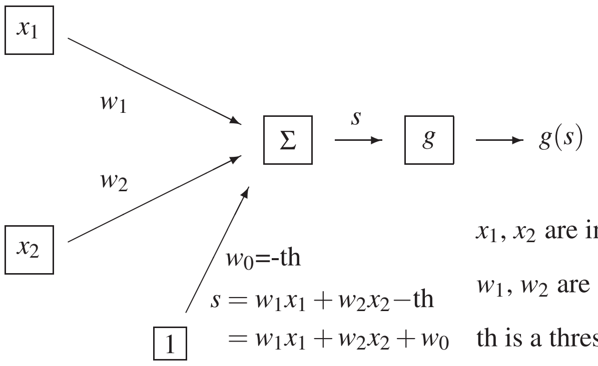
The input function is the weighted sum of the activation levels of inputs.

The activation level is a non-linear transfer function g of this input:

$$\text{activation}_j = g(s_j) = g(\sum_i w_{ji} x_i)$$

Some nodes are inputs (sensing), some are outputs (action)

Rosenblatt Perceptron



$$s = w_1x_1 + w_2x_2 - \text{th}$$
$$= w_1x_1 + w_2x_2 + w_0$$

x_1, x_2 are inputs

w_1, w_2 are synaptic weights

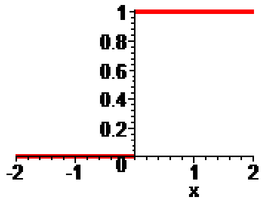
th is a threshold

w_0 is a **bias** weight

g is transfer function

Transfer function

Originally, a (discontinuous) step function was used for the transfer function:



$$g(s) = \begin{cases} 1, & \text{if} \quad s \geq 0 \\ 0, & \text{if} \quad s < 0 \end{cases}$$

(Later, other transfer functions were introduced, which are continuous and smooth)

Linear Separability

Q: what kind of functions can a perceptron compute?

A: linearly separable functions

Examples include:

AND	$w_1 = w_2 = 1.0,$	$w_0 = -1.5$
OR	$w_1 = w_2 = 1.0,$	$w_0 = -0.5$
NOR	$w_1 = w_2 = -1.0,$	$w_0 = 0.5$

Q: How do we train it to learn a new function?

Perceptron Learning Rule

Adjust the weights as each input is presented.

recall: $s = w_1x_1 + w_2x_2 + w_0$,

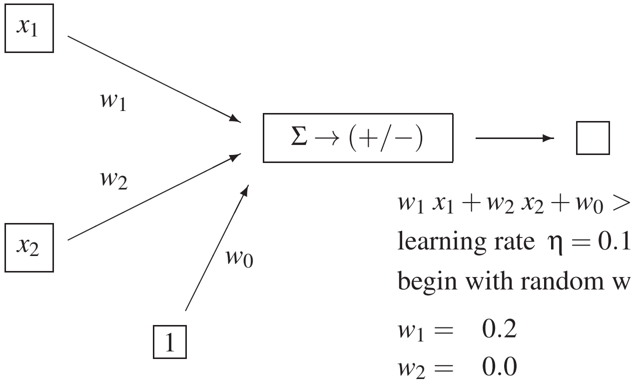
$\eta > 0$ is called the **learning rate**

if $g(s) = 0$ but should be 1,	if $g(s) = 1$ but should be 0,
$w_k \leftarrow w_k + \eta x_k$	$w_k \leftarrow w_k - \eta x_k$
$w_0 \leftarrow w_0 + \eta$	$w_0 \leftarrow w_0 - \eta$
so $s \leftarrow s + \eta(1 + \sum_k x_k^2)$	so $s \leftarrow s - \eta(1 + \sum_k x_k^2)$

otherwise, weights are unchanged.

Theorem: This will learn to classify the data correctly, as long as they are **linearly separable**.

Perceptron Learning Example



$w_1\,x_1 + w_2\,x_2 + w_0 > 0$

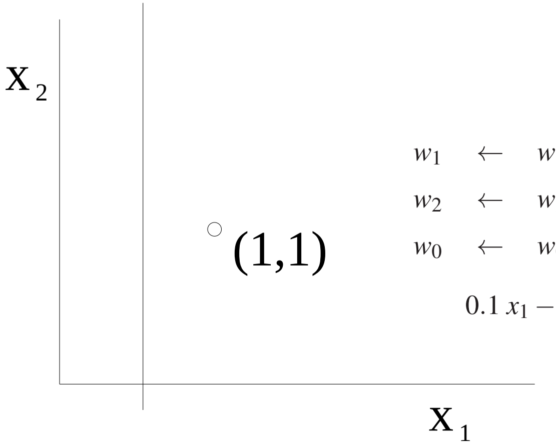
learning rate $\eta = 0.1$

begin with random weights

$$\begin{aligned} w_1 &= 0.2 \\ w_2 &= 0.0 \\ w_0 &= -0.1 \end{aligned}$$

$0.2\,x_1 + 0.0\,x_2 - 0.1 > 0$

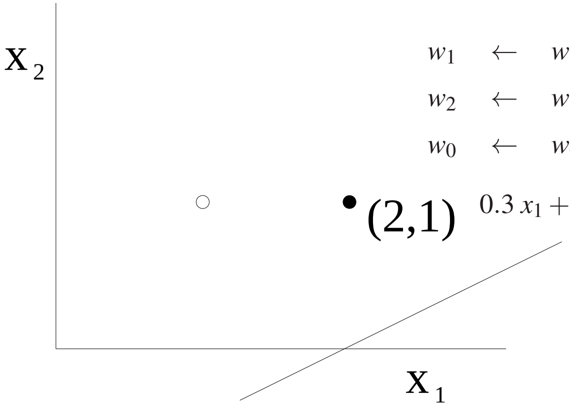
Training Step 1



$$\begin{aligned} w_1 &\leftarrow w_1 - \eta\,x_1 = 0.1 \\ w_2 &\leftarrow w_2 - \eta\,x_2 = -0.1 \\ w_0 &\leftarrow w_0 - \eta = -0.2 \end{aligned}$$

$0.1\,x_1 - 0.1\,x_2 - 0.2 > 0$

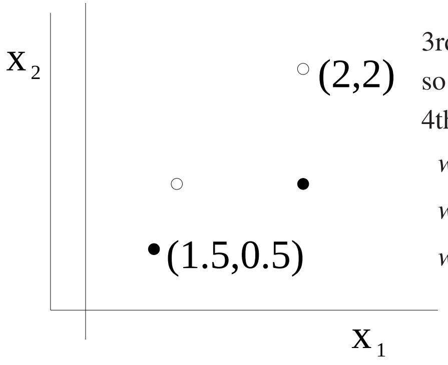
Training Step 2



$$\begin{aligned} w_1 &\leftarrow w_1 + \eta\,x_1 = 0.3 \\ w_2 &\leftarrow w_2 + \eta\,x_2 = 0.0 \\ w_0 &\leftarrow w_0 + \eta = -0.1 \end{aligned}$$

$0.3\,x_1 + 0.0\,x_2 - 0.1 > 0$

Training Step 3



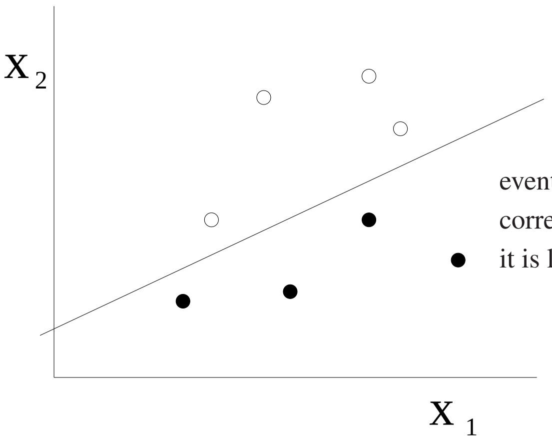
3rd point correctly classified,
so no change

4th point:

$$\begin{aligned} w_1 &\leftarrow w_1 - \eta\,x_1 = 0.1 \\ w_2 &\leftarrow w_2 - \eta\,x_2 = -0.2 \\ w_0 &\leftarrow w_0 - \eta = -0.2 \end{aligned}$$

$0.1\,x_1 - 0.2\,x_2 - 0.2 > 0$

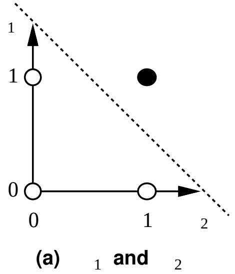
Training Step 4



eventually, all the data will be correctly classified (provided it is linearly separable)

Limitations

Problem: many useful functions are not linearly separable (e.g. XOR)

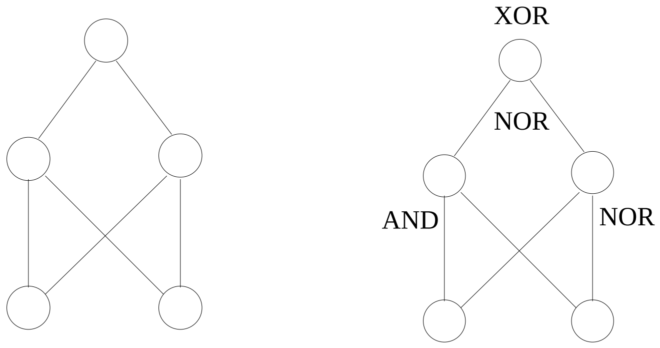


Possible solution:

x_1 XOR x_2 can be written as: (x_1 AND x_2) NOR (x_1 NOR x_2)

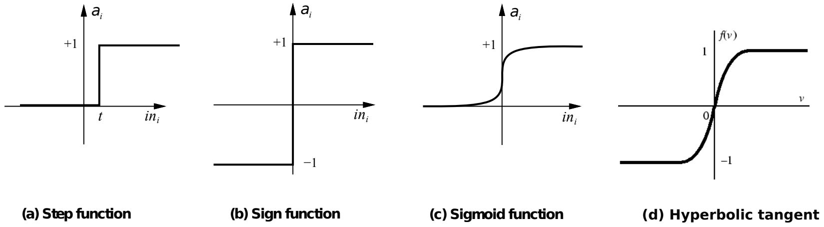
Recall that AND, OR and NOR can be implemented by perceptrons.

Multi-Layer Neural Networks



Problem: How do we train it to learn a new function? (credit assignment)

Key Idea



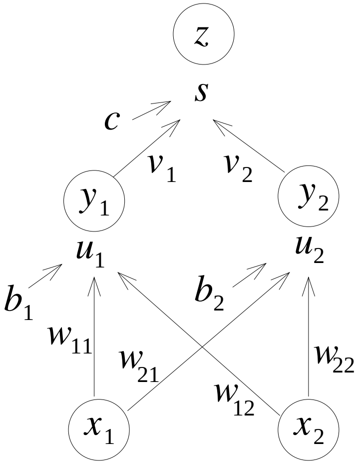
Replace the (discontinuous) step function with a differentiable function, such as the sigmoid:

$$g(s) = \frac{1}{1+e^{-s}}$$

or hyperbolic tangent

$$g(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} = 2\left(\frac{1}{1+e^{-2s}}\right) - 1$$

Forward Pass



$$\begin{aligned}
u_1 &= b_1 + w_{11}x_1 + w_{12}x_2\\
y_1 &= g(u_1)\\
s &= c + v_1 y_1 + v_2 y_2\\
z &= g(s)\\
E &= \frac{1}{2}\sum(z-t)^2
\end{aligned}$$

Gradient Descent

We define an **error function** E to be (half) the sum over all input patterns of the square of the difference between actual output and desired output

$$E = \frac{1}{2}\sum(z-t)^2$$

If we think of E as height, it defines an error **landscape** on the weight space. The aim is to find a set of weights for which E is very low. This is done by moving in the steepest downhill direction.

$$w \leftarrow w - \eta\,\frac{\partial E}{\partial w}$$

Parameter η is called the learning rate.

Chain Rule

If, say

$$y = y(u)$$

$$u = u(x)$$

Then

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\,\frac{\partial u}{\partial x}$$

This principle can be used to compute the partial derivatives in an efficient and localized manner. Note that the transfer function must be differentiable (usually sigmoid, or tanh).

$$\begin{aligned}
&\text{Note: if}\quad z(s) = \frac{1}{1+e^{-s}}, &\quad z'(s) &= z(1-z).\\
&\text{if}\quad z(s) = \tanh(s), &\quad z'(s) &= 1-z^2.
\end{aligned}$$

Backpropagation

We want to find the way the error function changes with respect to the weights, which allows us to change weights such that the error is reduced.

For output node weights:

$$\frac{\partial E}{\partial v_1} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial v_1}$$

For hidden node weights:

$$\frac{\partial E}{\partial w_{11}} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial y_1}\frac{\partial y_1}{\partial u_1}\frac{\partial u_1}{\partial w_{11}}$$

Backpropagation

Partial Derivatives

$$\frac{\partial E}{\partial z} = z - t$$

$$\frac{dz}{ds} = g'(s) = z(1-z)$$

$$\frac{\partial s}{\partial y_1} = v_1$$

$$\frac{dy_1}{du_1} = y_1(1-y_1)$$

Useful notation

$$\delta_{\text{out}} = \frac{\partial E}{\partial s} \quad \delta_1 = \frac{\partial E}{\partial u_1} \quad \delta_2 = \frac{\partial E}{\partial u_2}$$

Then

$$\delta_{\text{out}} = (z-t)\, z\, (1-z)$$

$$\frac{\partial E}{\partial v_1} = \delta_{\text{out}}\, y_1$$

$$\delta_1 = \delta_{\text{out}}\, v_1\, y_1\, (1-y_1)$$

$$\frac{\partial E}{\partial w_{11}} = \delta_1\, x_1$$

Partial derivatives can be calculated efficiently by packpropagating deltas through the network.

Backpropagation

- Target values need to be within the range of the activation function, otherwise weights will grow unbounded
  - For example when using the logistic (sigmoid) activation function it is better to use 0.05 for a low target and 0.95 for a high target than 0 and 1
- When the output has more than two classes (multinomial encoding), a common method is to create an output node for each class, and set a high target for the correct class and low for all others

---

Variations on Backprop

- Cross Entropy
  - problem: least squares error function unsuitable for classification, where target = 0 or 1
  - mathematical theory: maximum likelihood
  - solution: replace with cross entropy error function
- Weight Decay
  - problem: weights “blow up”, and inhibit further learning
  - solution: add weight decay term to error function
- Momentum
  - problem: weights oscillate in a “rain gutter”
  - solution: weighted average of gradient over time

Cross Entropy

For classification tasks, target t is either 0 or 1, so better to use

$$E = -t\log(z) - (1-t)\log(1-z)$$

This can be justified mathematically, works well in practice, and also makes the backprop computations simpler

$$\frac{\partial E}{\partial z} = \frac{z-t}{z(1-z)}$$

$$\text{if} \quad z = \frac{1}{1+e^{-s}},$$

$$\frac{\partial E}{\partial s} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial s} = z - t$$

Maximum Likelihood

H is a class of hypotheses

$P(D|h) =$ probability of data D being generated under hypothesis $h \in H$.

$\log P(D|h)$ is called the likelihood.

ML Principle: Choose $h \in H$ which maximizes the likelihood,
i.e. maximizes $P(D|h)$ [or, maximizes $\log P(D|h)$]

Derivation of Least Squares

Suppose data generated by a linear function h, plus Gaussian noise with standard deviation σ.

$$\begin{aligned}
P(D|h) &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2} \\
\log P(D|h) &= \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2 - \log(\sigma) - \frac{1}{2}\log(2\pi) \\
h_{ML} &= \text{argmax}_{h\in H}\ \log P(D|h) \\
&= \text{argmin}_{h\in H} \sum_{i=1}^{m} (d_i - h(x_i))^2
\end{aligned}$$

(Note: we do not need to know σ)

Derivation of Cross Entropy

For classification tasks, d is either 0 or 1.
Assume D generated by hypothesis h as follows:

$$\begin{aligned}
P(1|h(x_i)) &= h(x_i) \\
P(0|h(x_i)) &= (1 - h(x_i)) \\
\text{i.e.}\quad P(d_i|h(x_i)) &= h(x_i)^{d_i}(1 - h(x_i))^{1-d_i}
\end{aligned}$$

then

$$\begin{aligned}
\log P(D|h) &= \sum_{i=1}^{m} d_i \log h(x_i) + (1 - d_i)\log(1 - h(x_i)) \\
h_{ML} &= \text{argmax}_{h\in H} \sum_{i=1}^{m} d_i \log h(x_i) + (1 - d_i)\log(1 - h(x_i))
\end{aligned}$$

(Can be generalized to multiple classes.)

Weight Decay

Assume that small weights are more likely to occur than large weights, i.e.

$$P(w) = \frac{1}{Z} e^{-\frac{\lambda}{2}\sum_j w_j^2}$$

where Z is a normalizing constant. Then the cost function becomes:

$$E = \frac{1}{2}\sum_i (z_i - t_i)^2 + \frac{\lambda}{2}\sum_j w_j^2$$

This can prevent the weights from "saturating" to very high values.

Problem: need to determine λ from experience, or empirically.

In practise, adjust weights using: $w \leftarrow w(1 - \varepsilon)$

Momentum

If landscape is shaped like a “rain gutter”, weights will tend to oscillate without much improvement.

Solution: add a momentum factor

$$\begin{aligned}\delta w &\leftarrow \alpha\,\delta w + (1-\alpha)\frac{\partial E}{\partial w}\\ w &\leftarrow w - \eta\,\delta w\end{aligned}$$

Hopefully, this will dampen sideways oscillations but amplify downhill motion by $\frac{1}{1-\alpha}$.

Conjugate Gradients

Compute matrix of second derivatives $\frac{\partial^2 E}{\partial w_i \partial w_j}$ (called the Hessian).

Approximate the landscape with a quadratic function (paraboloid).

Jump to the minimum of this quadratic function.

Natural Gradients (Amari, 1995)

Use methods from information geometry to find a “natural” re-scaling of the partial derivatives.

Training and Testing

- Gradient descent will adjust weights so that the network reproduces outputs according to examples it is trained on
- The training set is only a sample, we evaluate usefulness using a separate testing set
- Overfitting is when the network classifies training set examples better than the test set. We can check if learning is producing overfitting by using a validation set

---

Training Tips

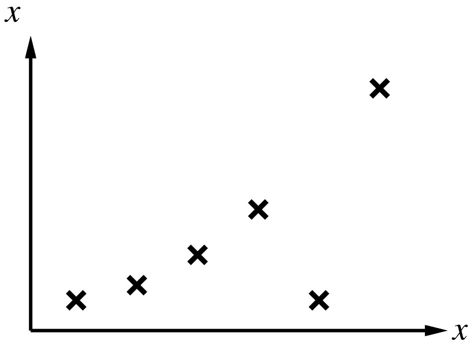
- re-scale inputs and outputs to be in the range 0 to 1 or -1 to 1
- initialize weights to very small random values
- on-line or batch learning
- three different ways to prevent overfitting:
  - limit the number of hidden nodes or connections
  - limit the training time, using a validation set
  - weight decay
- adjust learning rate and momentum to suit the particular task

Supervised Learning – Issues

- framework (decision tree, neural network, SVM, etc.)
- representation (of inputs and outputs)
- pre-processing / post-processing
- training method (perceptron learning, backpropagation, etc.)
- generalization (avoid over-fitting)
- evaluation (separate training, validation, test sets)

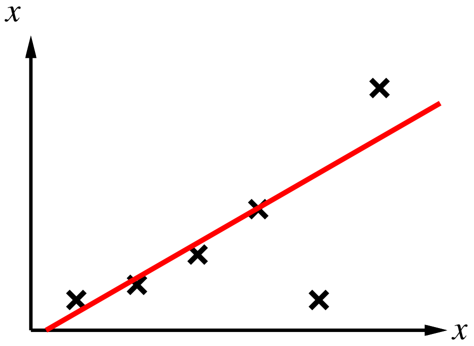
Curve Fitting

Which curve gives the “best fit” to these data?



Curve Fitting

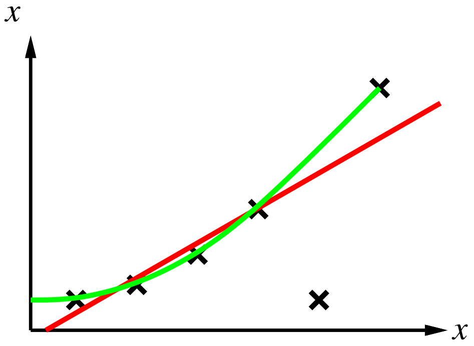
Which curve gives the “best fit” to these data?



straight line?

Curve Fitting

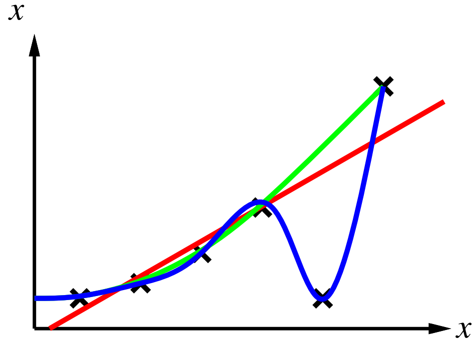
Which curve gives the “best fit” to these data?



parabola?

Curve Fitting

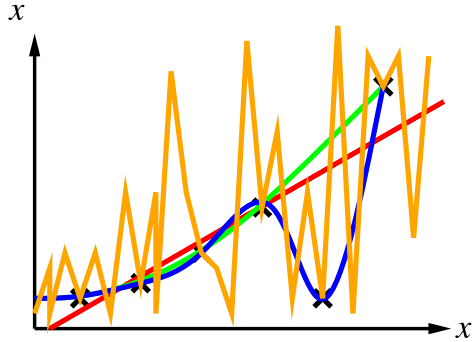
Which curve gives the “best fit” to these data?



4th order polynomial?

Curve Fitting

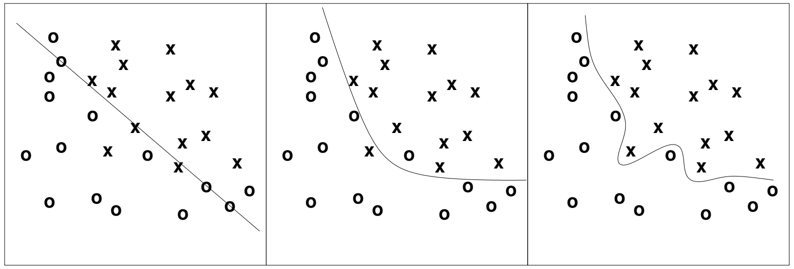
Which curve gives the “best fit” to these data?



Something else?

Ockham’s Razor

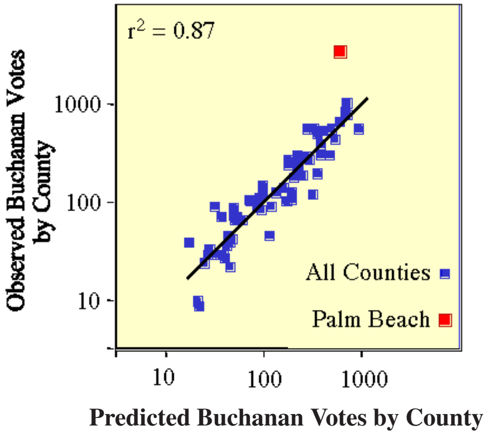
“The most likely hypothesis is the simplest one consistent with the data.”



inadequate good compromise over-fitting

Since there can be noise in the measurements, in practice need to make a tradeoff between simplicity of the hypothesis and how well it fits the data.

Outliers



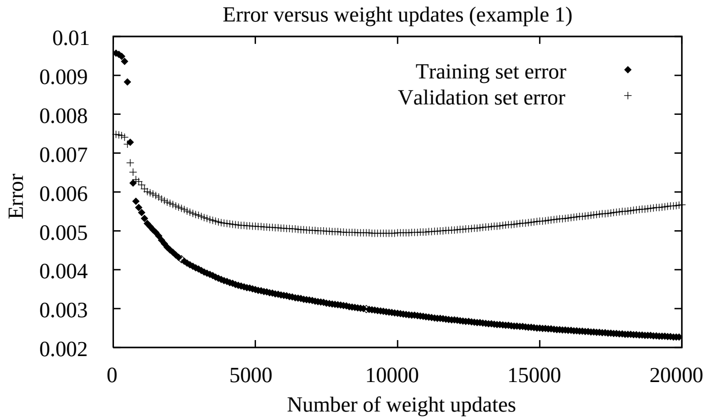
[faculty.washington.edu/mtbrett]

How to Prevent Over-Fitting

- limit the number of hidden nodes or connections
- limit the training time
- keep weights small, using Weight Decay

The appropriate number of hidden nodes or training cycles may be estimated using a Validation Set.

Overfitting in Neural Networks



Overfitting in Neural Networks