COMP9444
Neural Networks and Deep Learning

1d. Backpropagation

Textbook, Sections 3.10, 4.3, 5.1-5.2, 6.5.2

Outline

- Supervised Learning (5.1)
- Ockham's Razor (5.2)
- Multi-Layer Networks
- Continuous Activation Functions (3.10)
- Gradient Descent (4.3)
- Backpropagation (6.5.2)

Types of Learning (5.1)

- Supervised Learning
  - agent is presented with examples of inputs and their target outputs
- Reinforcement Learning
  - agent is not presented with target outputs, but is given a reward signal, which it aims to maximize
- Unsupervised Learning
  - agent is only presented with the inputs themselves, and aims to find structure in these inputs

Supervised Learning

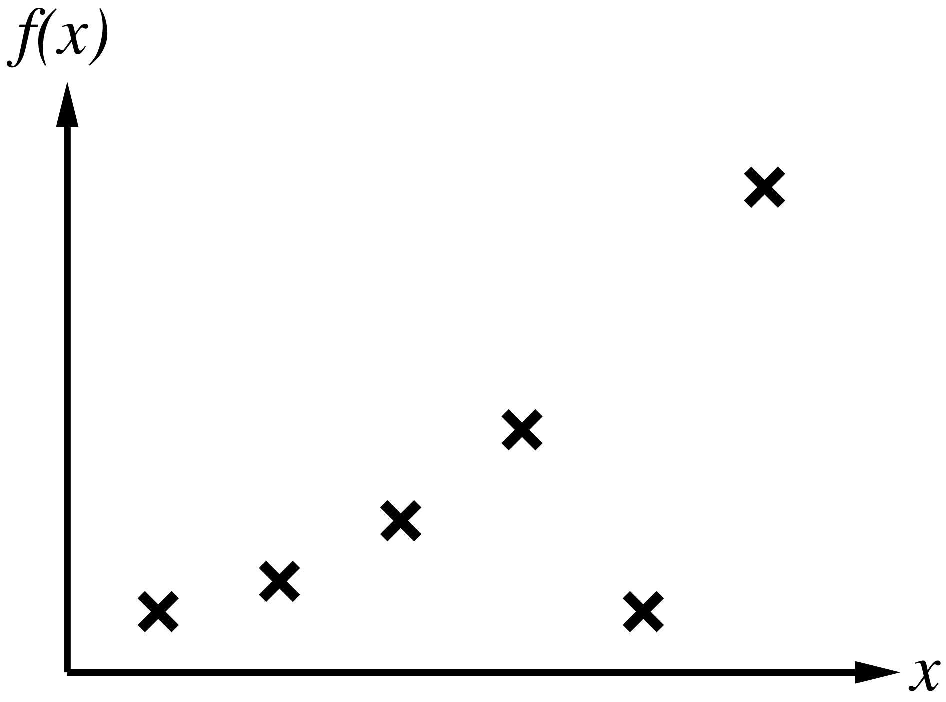
- we have a training set and a test set, each consisting of a set of items; for each item, a number of input attributes and a target value are specified.
- the aim is to predict the target value, based on the input attributes.
- agent is presented with the input and target output for each item in the training set; it must then predict the output for each item in the test set
- various learning paradigms are available:
  - Neural Network
  - Decision Tree
  - Support Vector Machine, etc.

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 and testing sets)

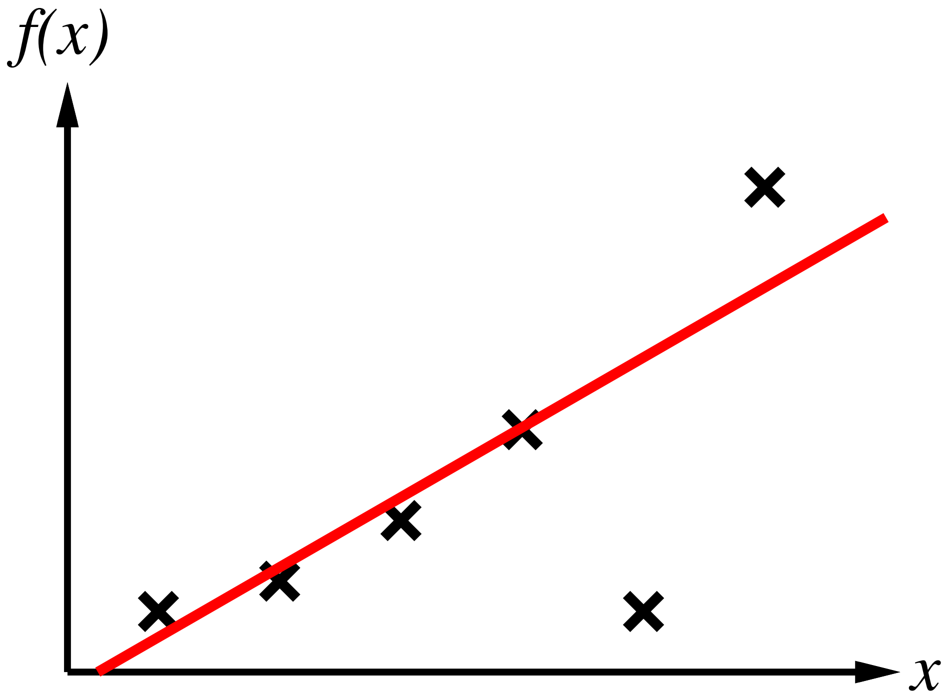
Curve Fitting

Which curve gives the "best fit" to these data?



Curve Fitting

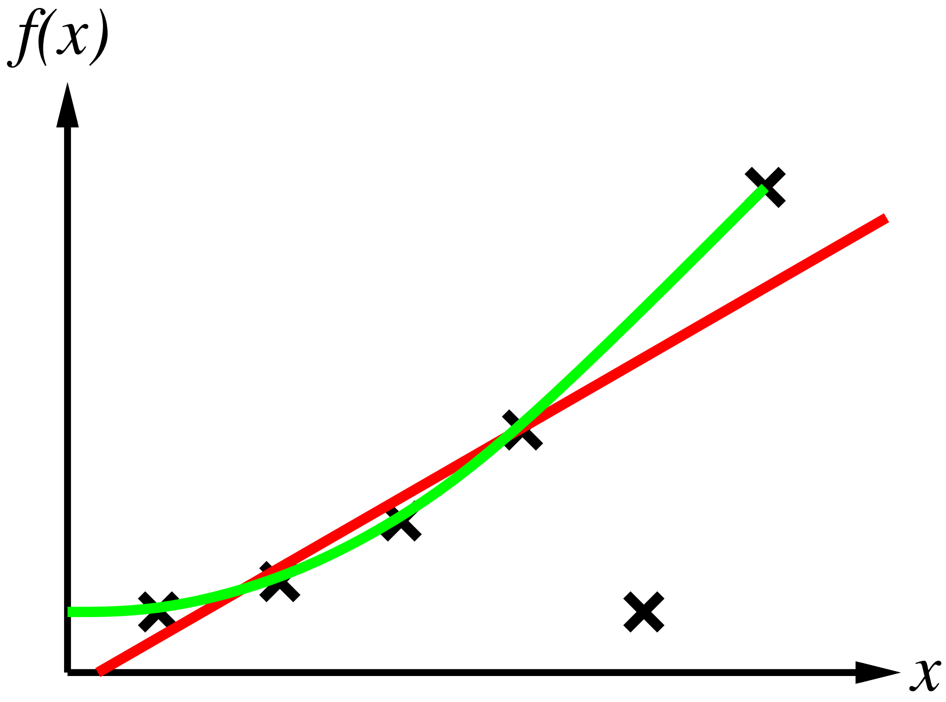
Which curve gives the “best fit” to these data?



straight line?

Curve Fitting

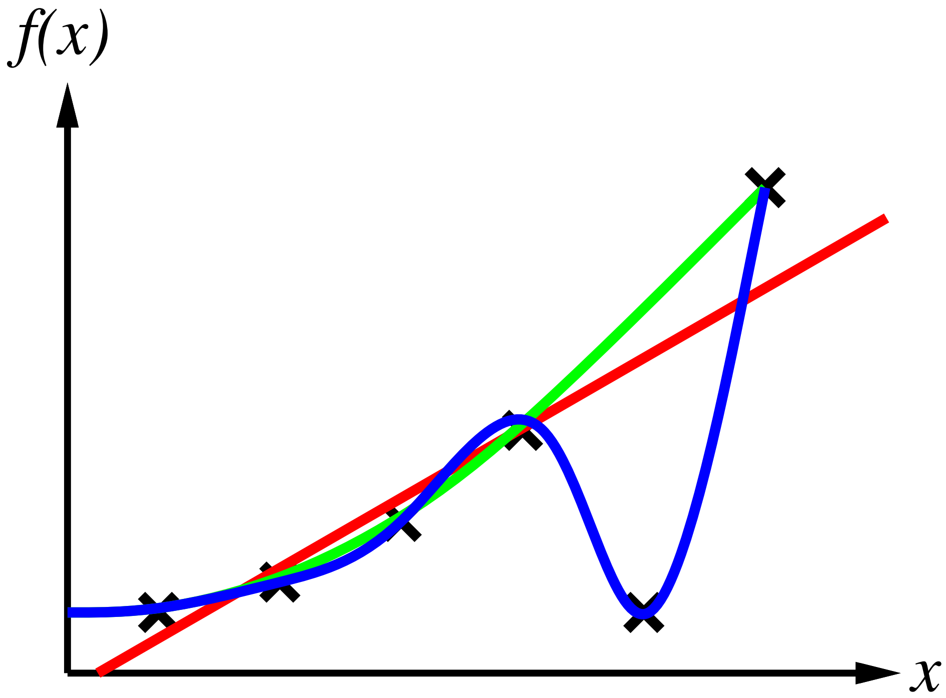
Which curve gives the "best fit" to these data?



parabola?

Curve Fitting

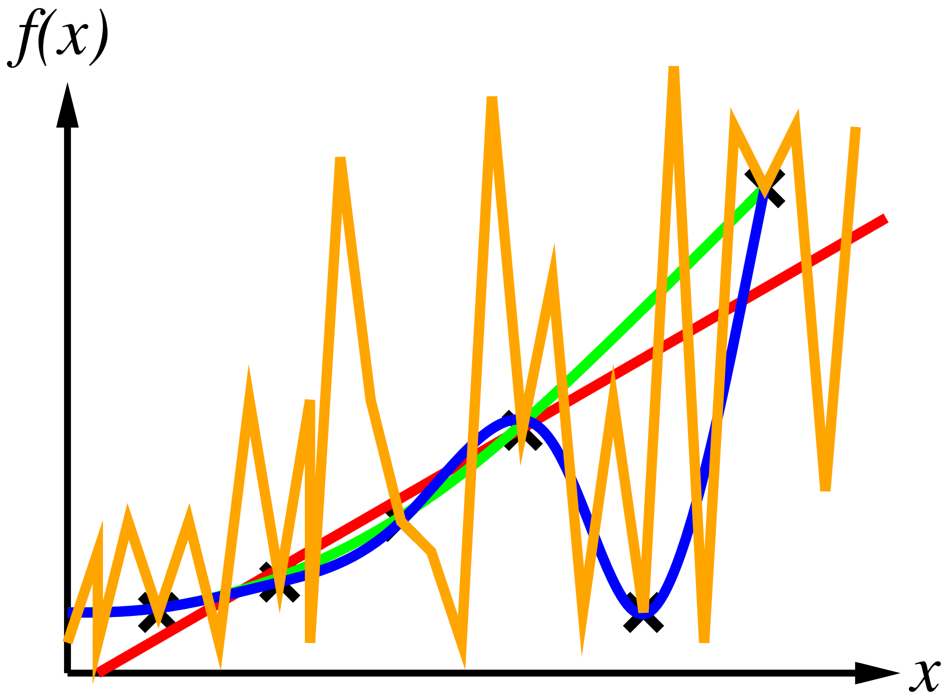
Which curve gives the "best fit" to these data?



4th order polynomial?

Curve Fitting

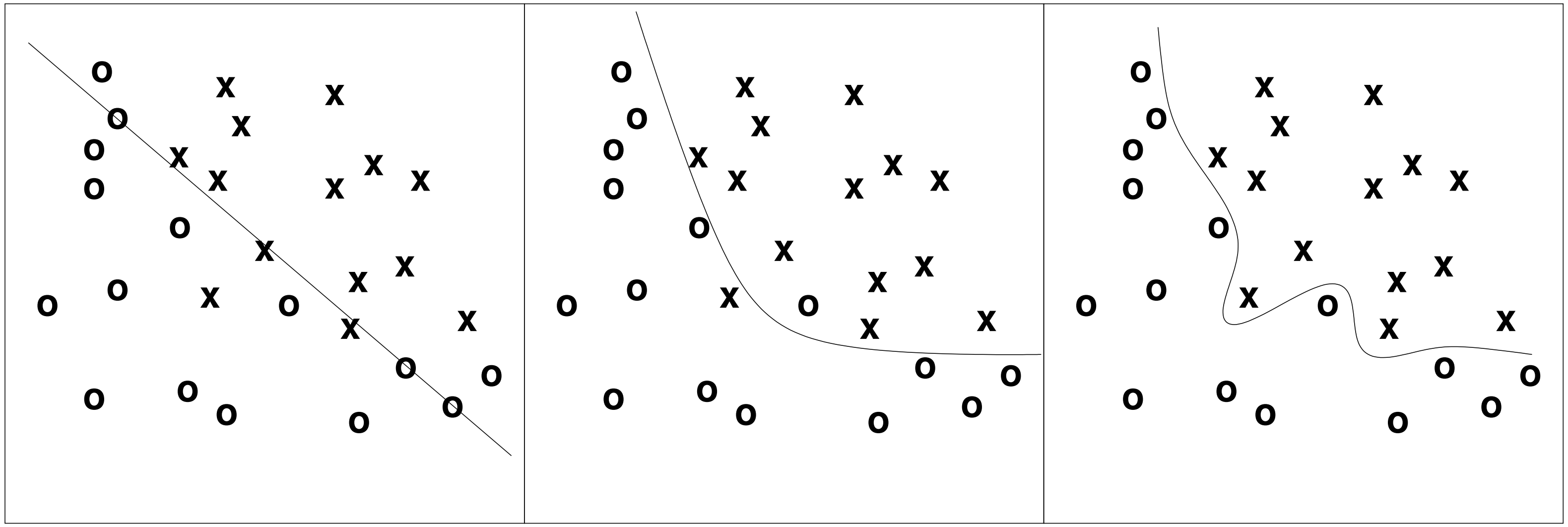
Which curve gives the “best fit” to these data?



Something else?

Ockham's Razor (5.2)

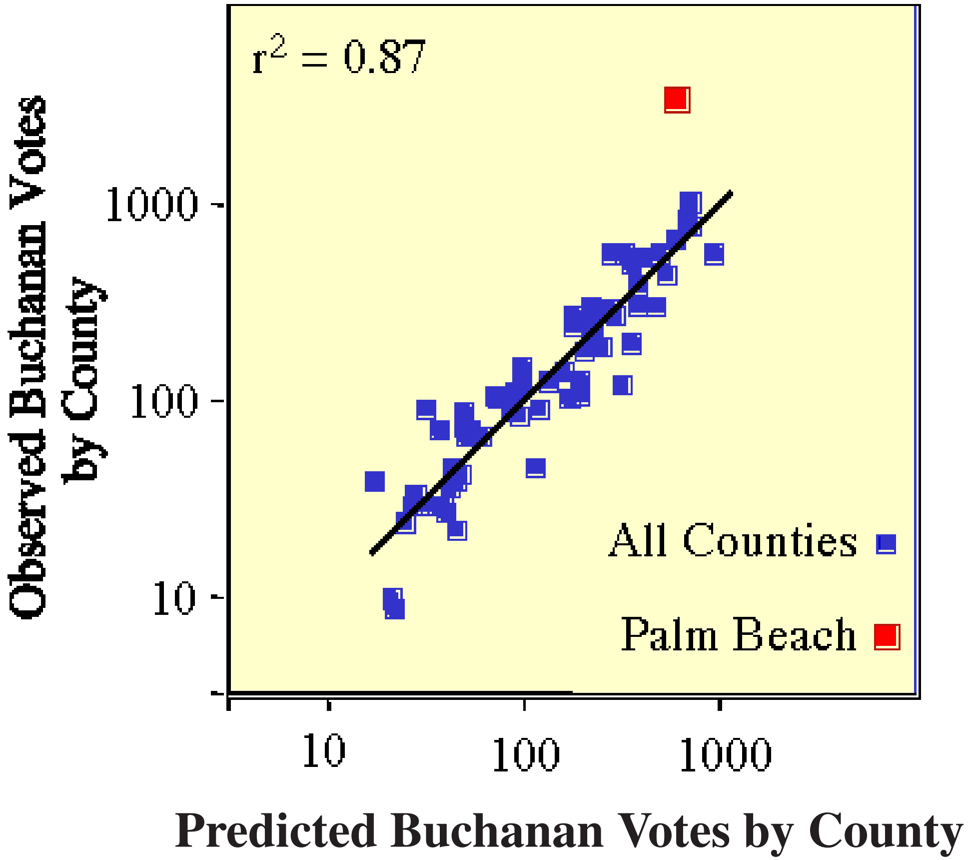
"The most likely hypothesis is the simplest one consistent with the data."



inadequate good compromise over-fitting

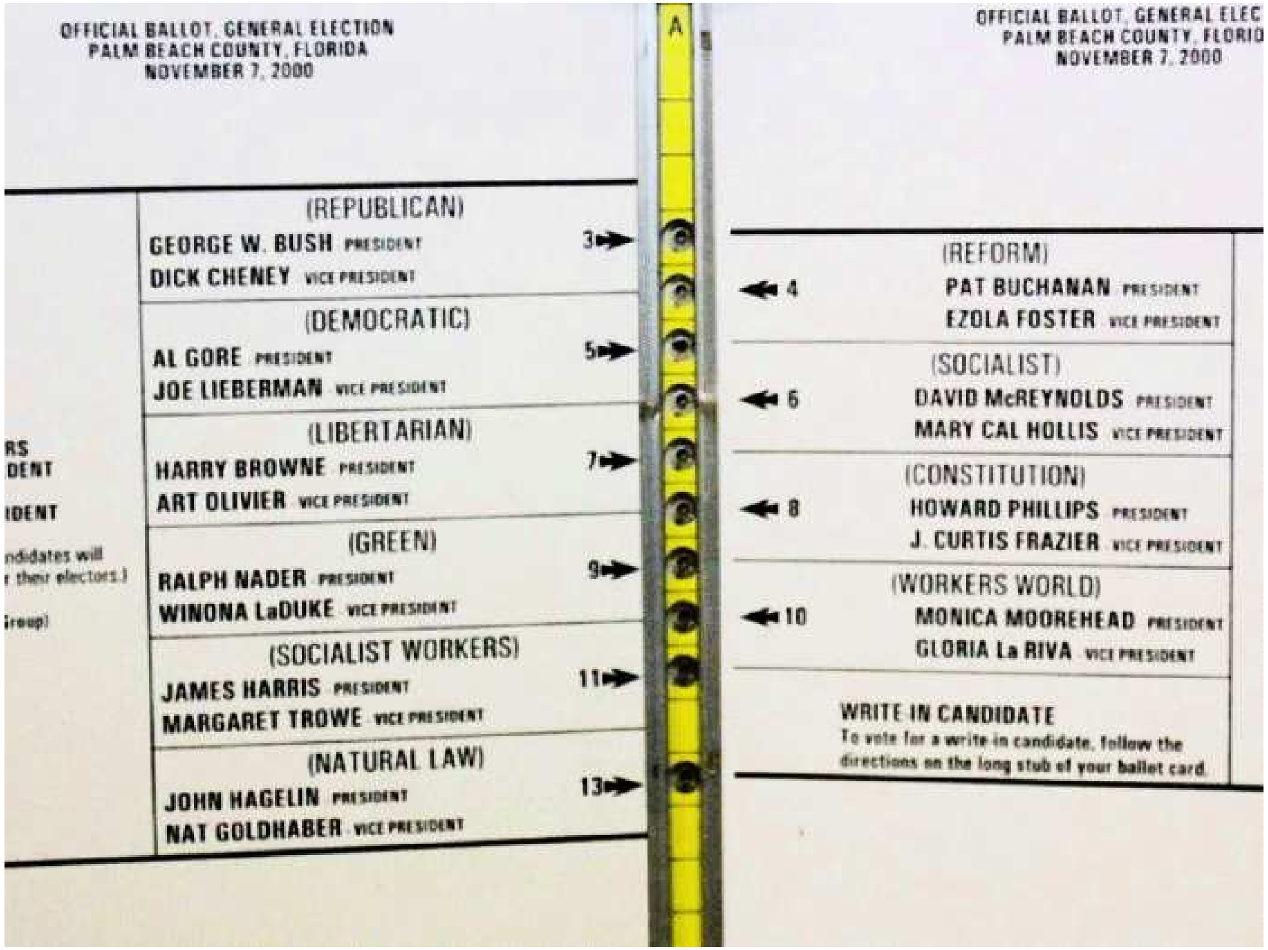
Since there can be noise in the measurements, in practice need to make a tradeoff between simplicity of the hypothesis and how well it fits the data.

Outliers



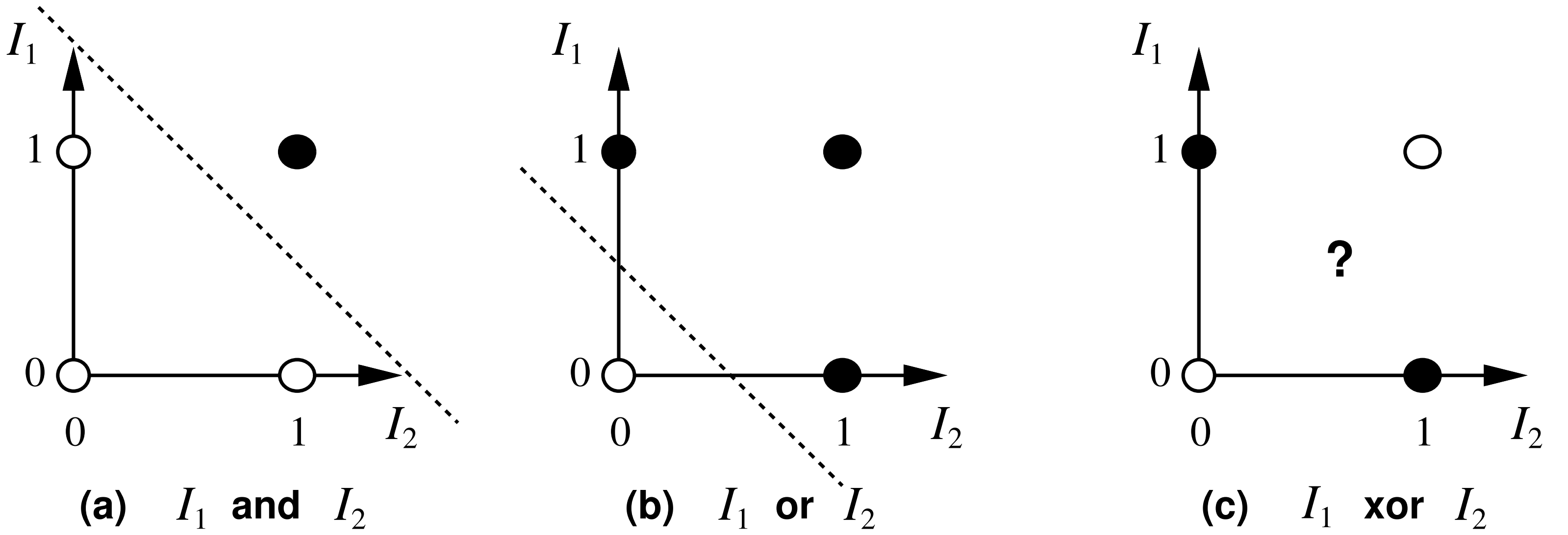
[faculty.washington.edu/mtbrett]

Butterfly Ballot



Recall: Limitations of Perceptrons

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

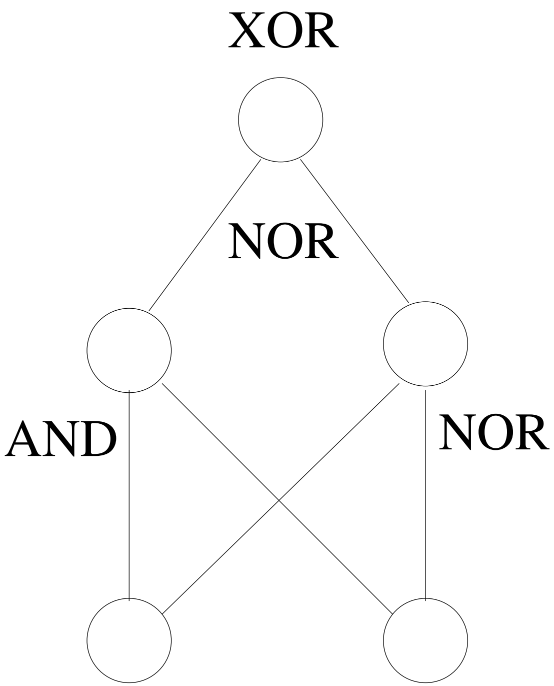


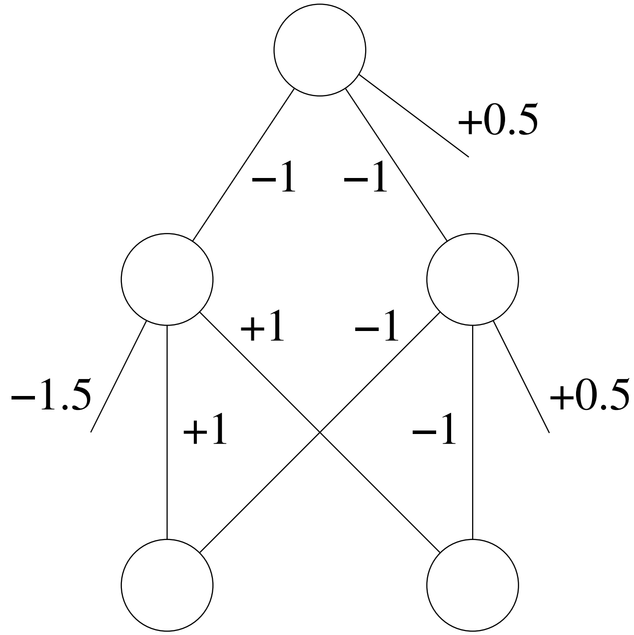
Possible solution:

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.

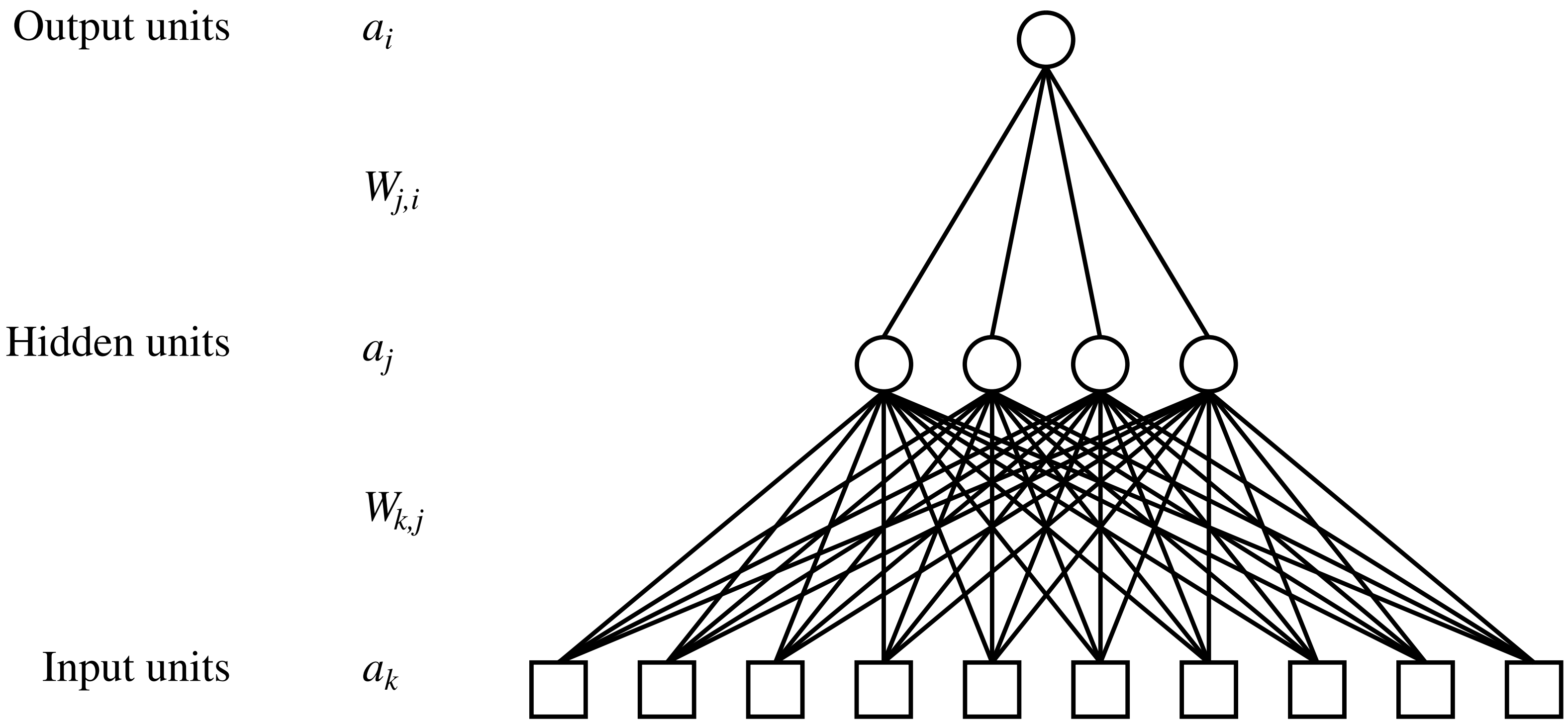
Multi-Layer Neural Networks





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

Two-Layer Neural Network



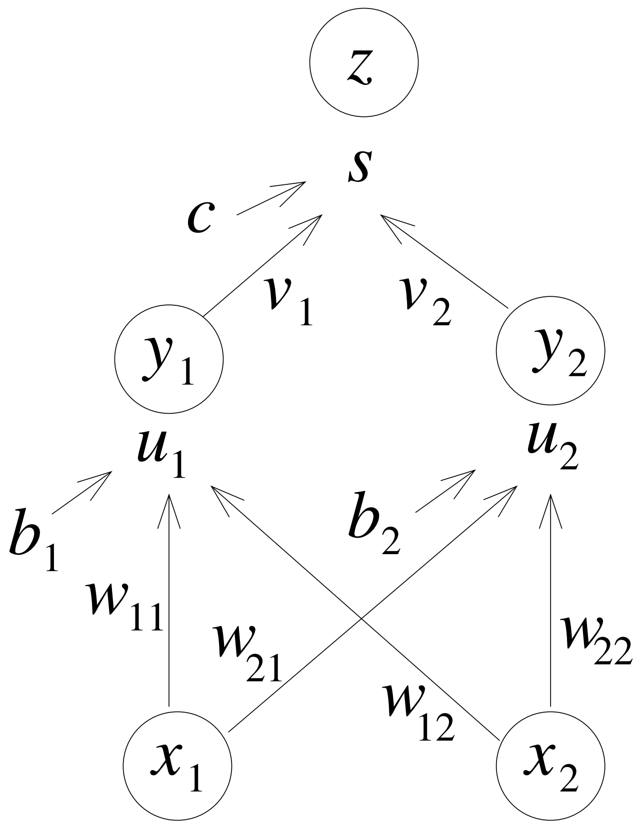
Normally, the numbers of input and output units are fixed, but we can choose the number of hidden units.

The XOR Problem

x_1	x_2	target
0	0	0
0	1	1
1	0	1
1	1	0

- for this toy problem, there is only a training set; there is no validation or test set, so we don't worry about overfitting
- the XOR data cannot be learned with a perceptron, but can be achieved using a 2-layer network with two hidden units

Neural Network Equations



$$
\begin{aligned}
u_1 &= b_1 + w_{11}x_1 + w_{12}x_2 \\
y_1 &= g(u_1) \\
s &= c + v_1 y_1 + v_2 y_2 \\
z &= g(s)
\end{aligned}
$$

We sometimes use w as a shorthand for any of the trainable weights $\{c, v_1, v_2, b_1, b_2, w_{11}, w_{21}, w_{12}, w_{22}\}$.

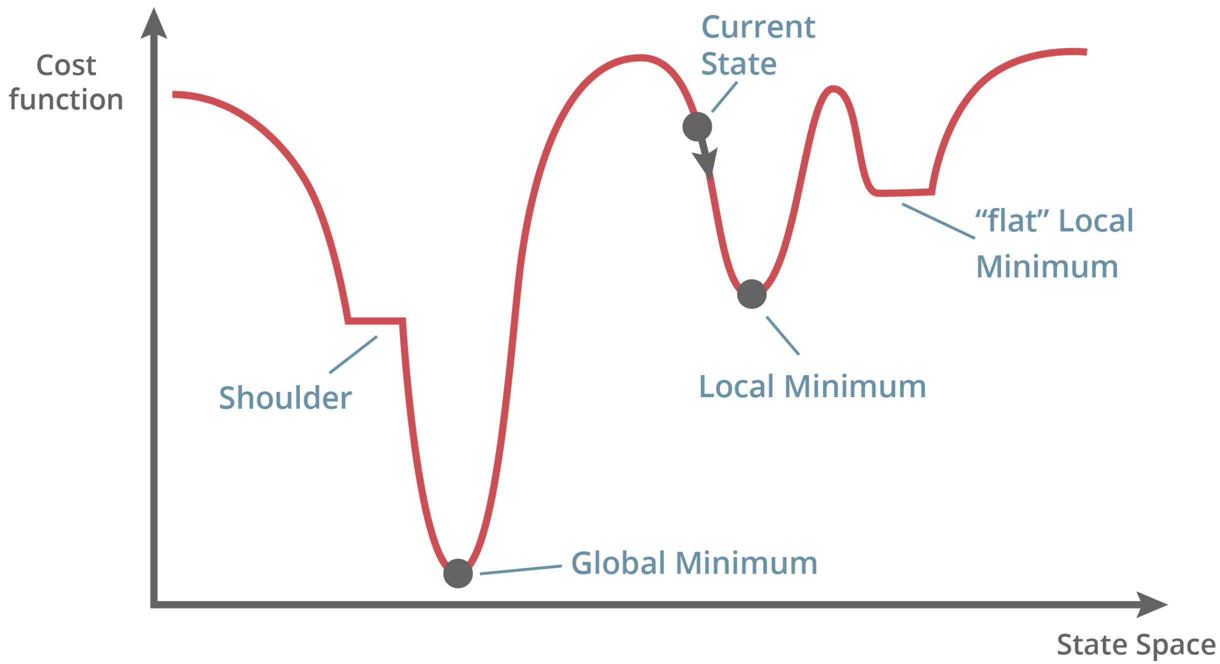
NN Training as Cost Minimization

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

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

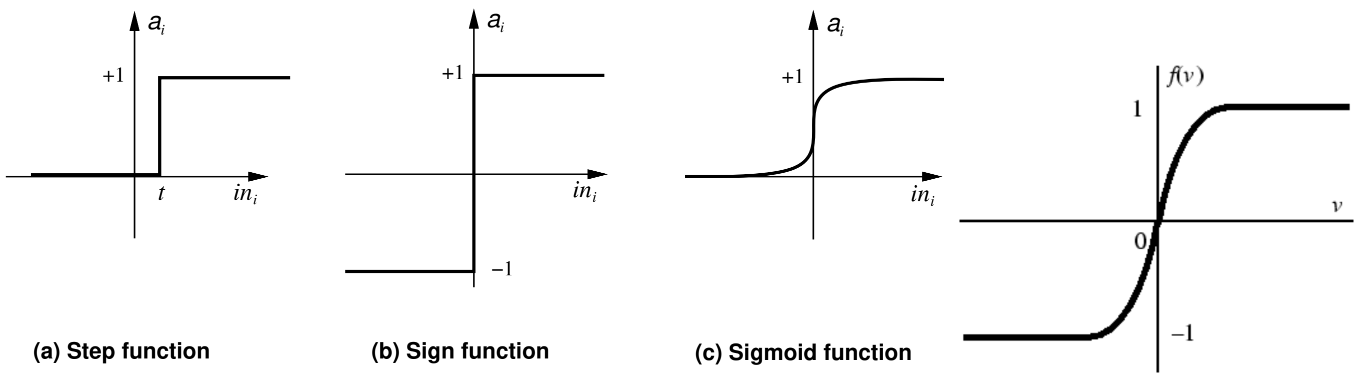
If we think of E as height, it defines an error **landscape** on the weight space. The aim is to find a set of weights for which E is very low.

Local Search in Weight Space



Problem: because of the step function, the landscape will not be smooth but will instead consist almost entirely of flat local regions and "shoulders", with occasional discontinuous jumps.

Continuous Activation Functions (3.10)



(a) Step function (b) Sign function (c) Sigmoid function

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

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

Gradient Descent (4.3)

Recall that the **loss** function E is (half) the sum over all input patterns of the square of the difference between actual output and target output

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

The aim is to find a set of weights for which E is very low.

If the functions involved are smooth, we can use multi-variable calculus to adjust the weights in such a way as to take us in the steepest downhill direction.

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

Parameter η is called the learning rate.

Chain Rule (6.5.2)

If, say

$$y = y(u)$$

$$u = u(x)$$

Then

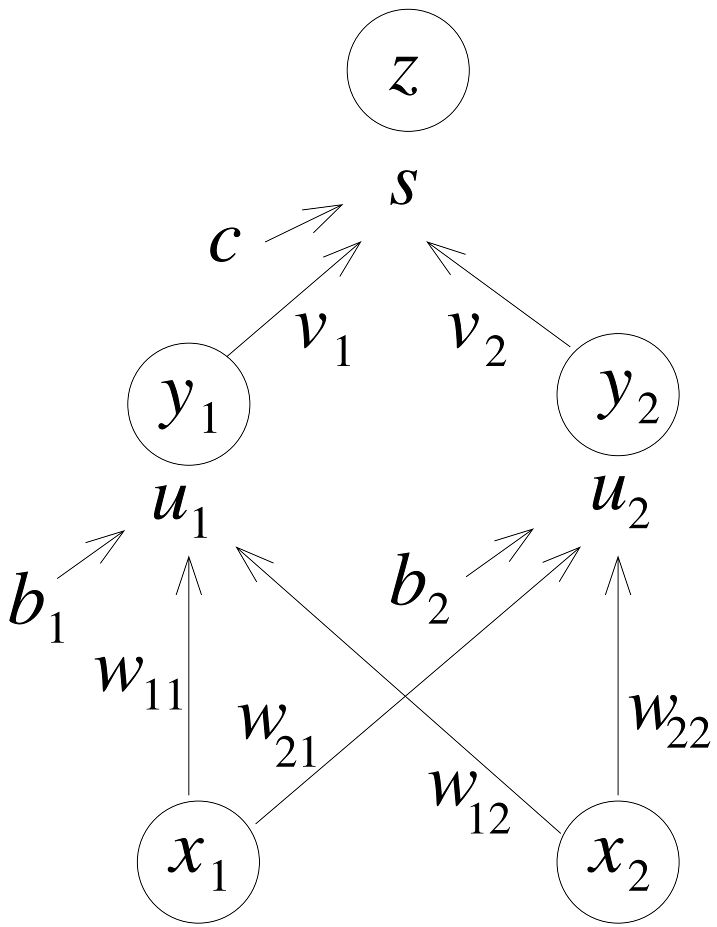
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

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).

$$\text{Note: if} \quad z(s) = \frac{1}{1+e^{-s}}, \qquad z'(s) = z(1-z).$$

$$\text{if} \quad z(s) = \tanh(s), \qquad z'(s) = 1-z^2.$$

Forward Pass



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

Backpropagation

Partial Derivatives

$$
\begin{aligned}
\frac{\partial E}{\partial z} &= z - t \\
\frac{dz}{ds} &= g'(s) = z(1-z) \\
\frac{\partial s}{\partial y_1} &= v_1 \\
\frac{dy_1}{du_1} &= y_1(1-y_1)
\end{aligned}
$$

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

$$
\begin{aligned}
\delta_{\text{out}} &= (z-t)\, z\, (1-z) \\
\frac{\partial E}{\partial v_1} &= \delta_{\text{out}}\, y_1 \\
\delta_1 &= \delta_{\text{out}}\, v_1\, y_1\, (1-y_1) \\
\frac{\partial E}{\partial w_{11}} &= \delta_1\, x_1
\end{aligned}
$$

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

Two-Layer NN's – Applications

- Medical Dignosis
- Autonomous Driving
- Game Playing
- Credit Card Fraud Detection
- Handwriting Recognition
- Financial Prediction

Example: Pima Indians Diabetes Dataset

	Attribute	mean	stdv
1.	Number of times pregnant	3.8	3.4
2.	Plasma glucose concentration	120.9	32.0
3.	Diastolic blood pressure (mm Hg)	69.1	19.4
4.	Triceps skin fold thickness (mm)	20.5	16.0
5.	2-Hour serum insulin (mu U/ml)	79.8	115.2
6.	Body mass index (weight in kg/(height in m)^2)	32.0	7.9
7.	Diabetes pedigree function	0.5	0.3
8.	Age (years)	33.2	11.8

Based on these inputs, try to predict whether the patient will develop Diabetes (1) or Not (0).

Training Tips

- re-scale inputs and outputs to be in the range 0 to 1 or -1 to 1
  - otherwise, backprop may put undue emphasis on larger values
- replace missing values with mean value for that attribute
- initialize weights to small random values
- on-line, batch, mini-batch, experience replay
- adjust learning rate (and momentum) to suit the particular task