

# COMP9414/9814/3411: Artificial Intelligence

## 11. Neural Networks

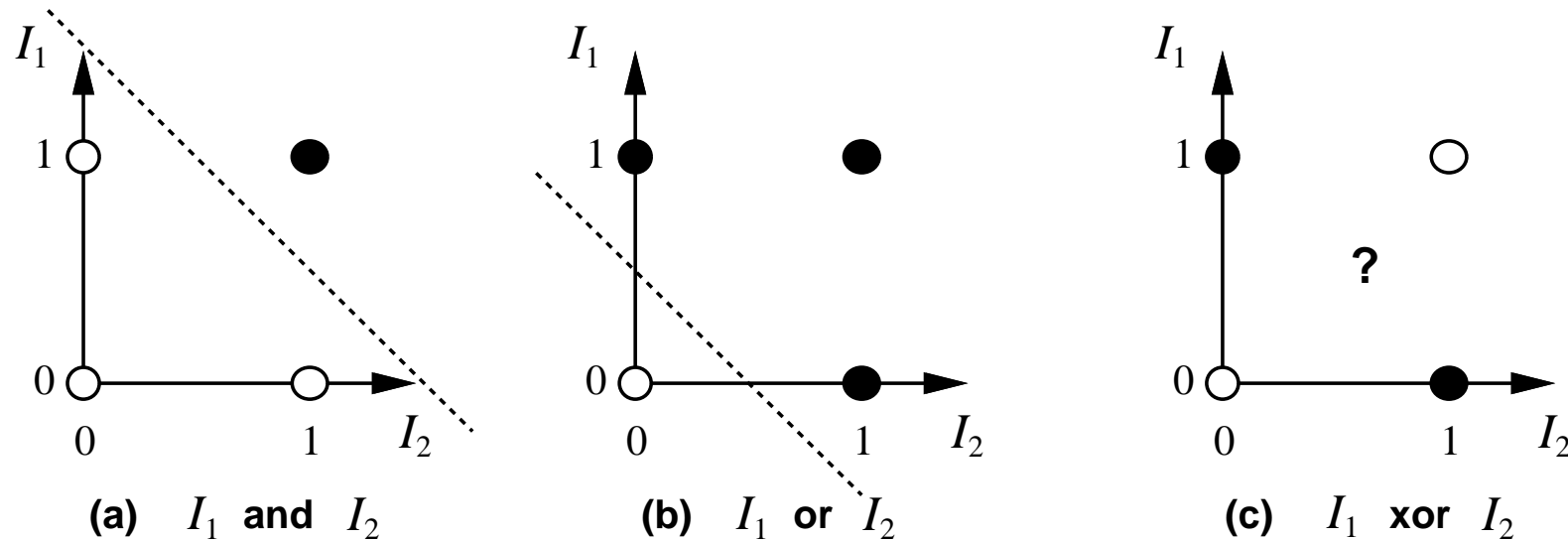
# Outline

---

- Multi-Layer Neural Networks
- Backpropagation
- Application - ALVINN
- Variations on Backprop
  - ▶ Cross Entropy, Weight Decay, Momentum
- Training Tips

## Recall: Limitations of Perceptrons

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



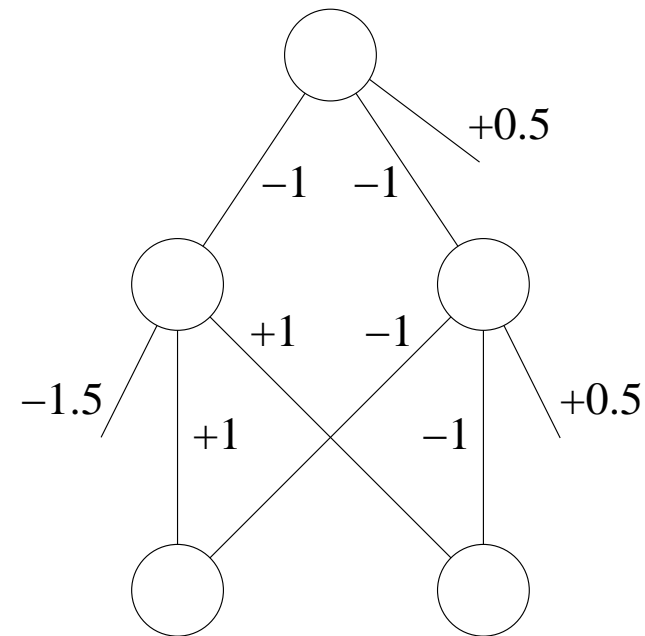
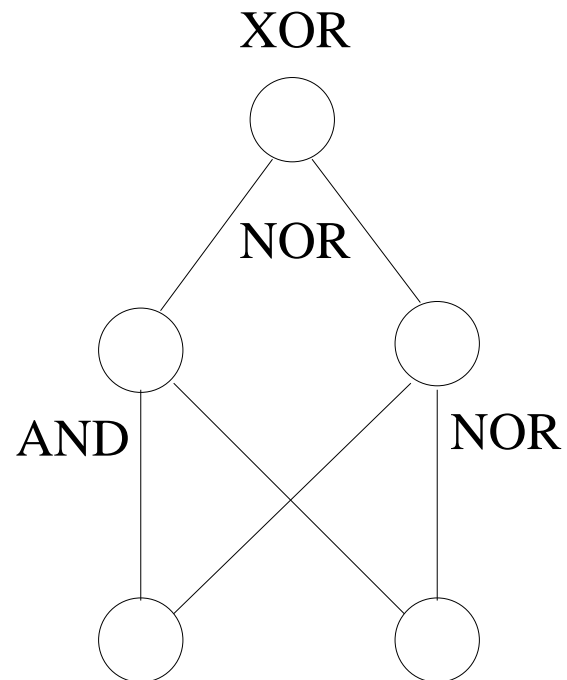
Possible solution:

$x_1$  XOR  $x_2$  can be written as:  $(x_1 \text{ AND } x_2) \text{ NOR } (x_1 \text{ NOR } x_2)$

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

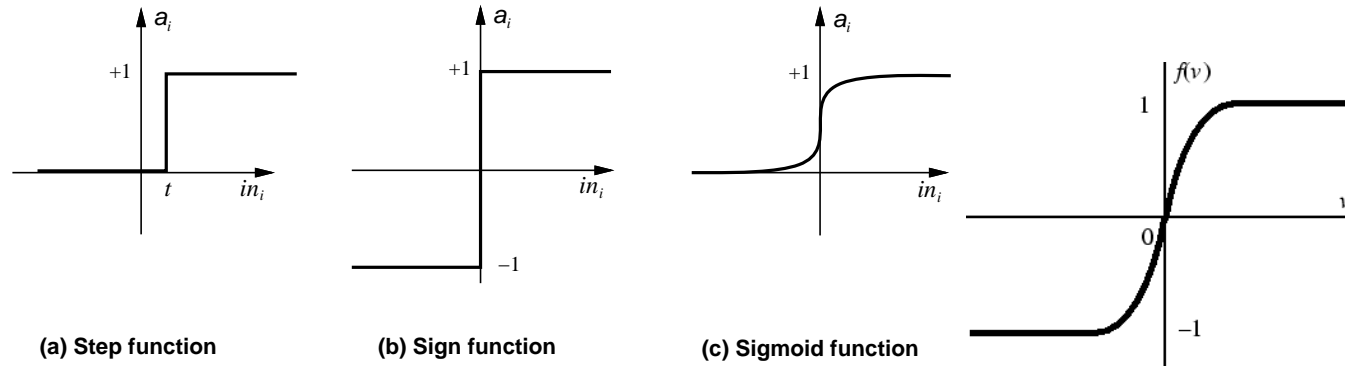
# Multi-Layer Neural Networks

---



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

# Key Idea



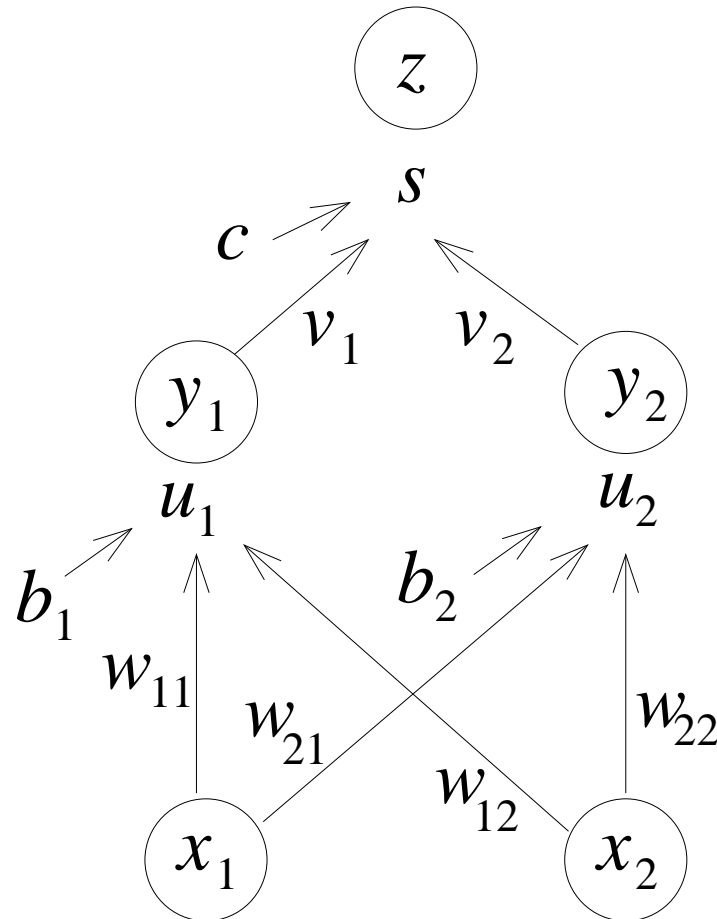
Replace the (discontinuous) step function with a differentiable function, such as the sigmoid:

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

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



$$u_1 = b_1 + w_{11}x_1 + w_{12}x_2$$

$$y_1 = g(u_1)$$

$$s = c + v_1y_1 + v_2y_2$$

$$z = g(s)$$

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

# 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

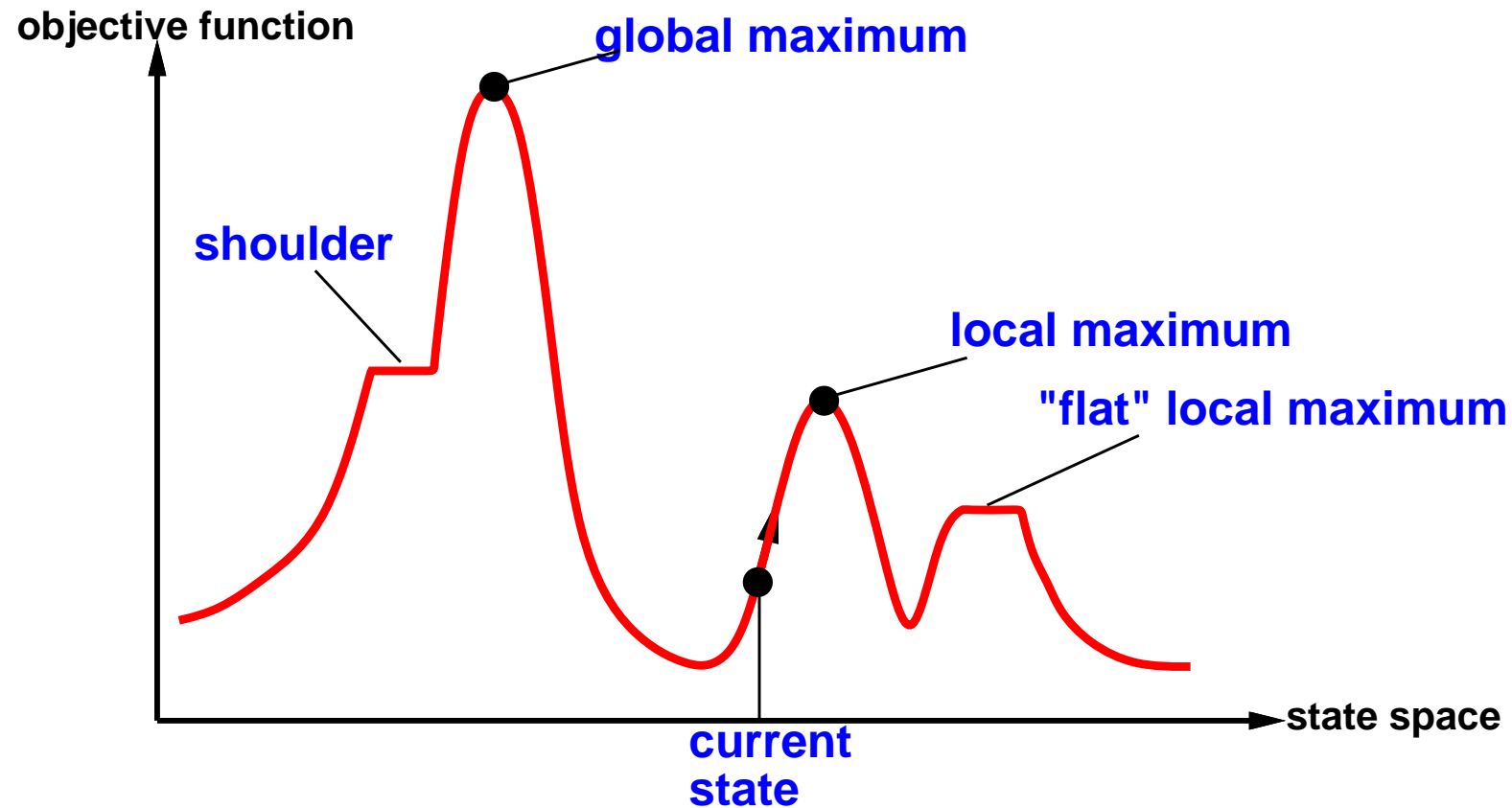
$$E = \frac{1}{2} \sum (z - t)^2$$

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  $\eta$  is called the **learning rate**.

# Local Search in Weight Space





# 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).

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

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

# 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

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

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

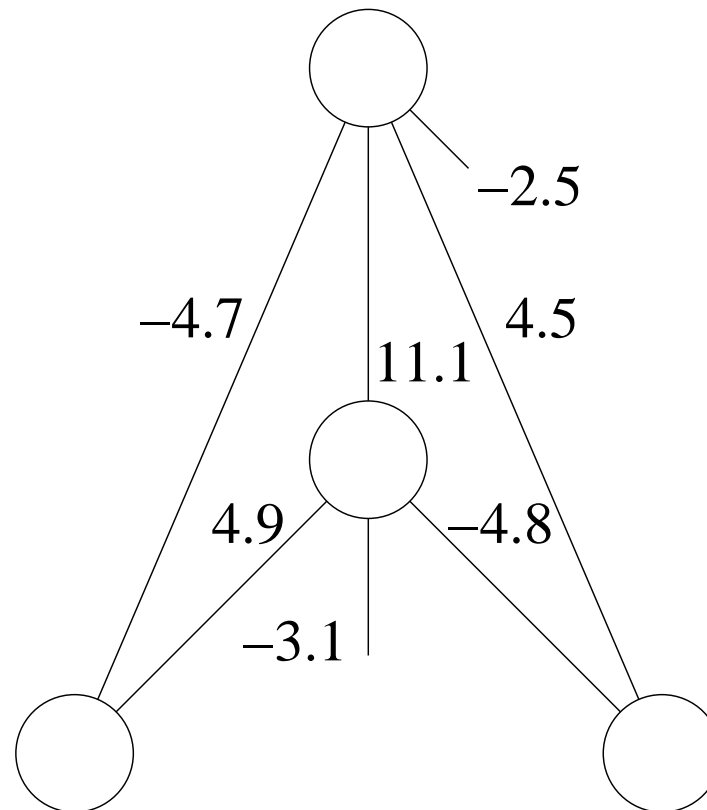
$$\delta_1 = \delta_{\text{out}} v_1 y_1 (1 - y_1)$$

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

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

# Trained XOR Network

---

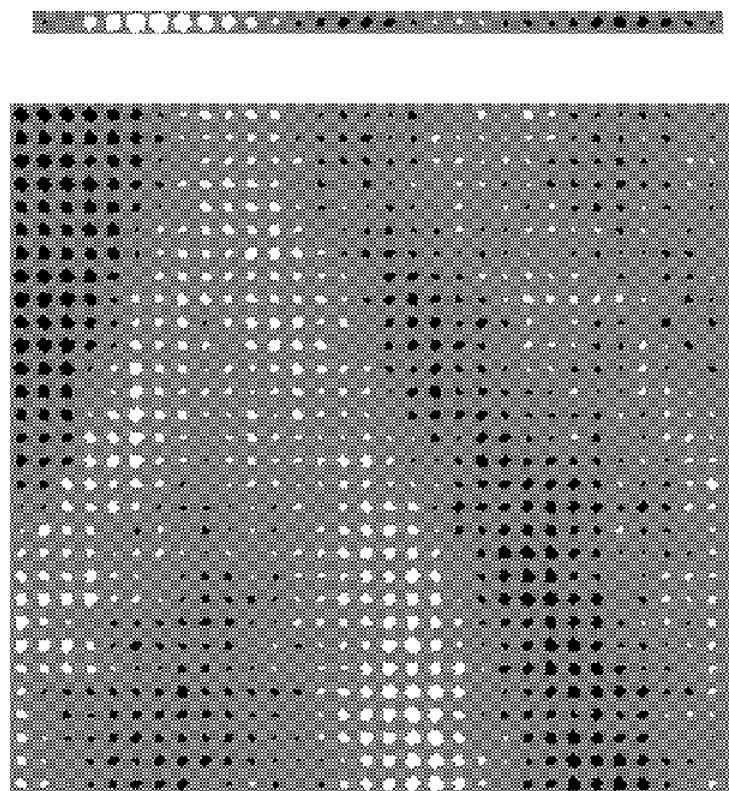
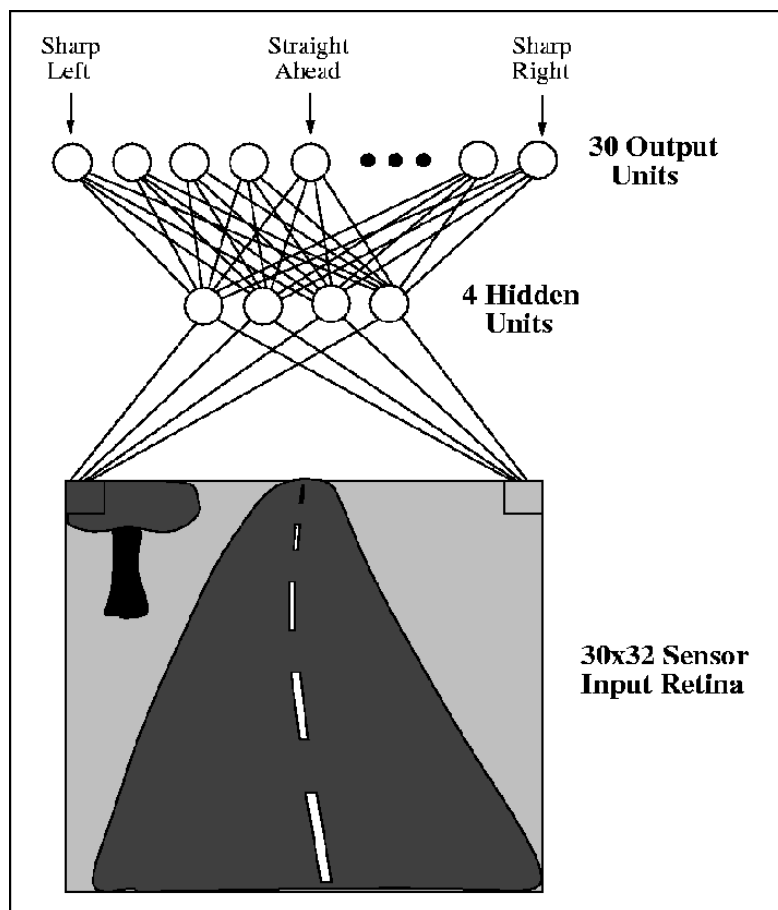


# Neural Network – Applications

---

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

# ALVINN



# ALVINN

---

- Autonomous Land Vehicle In a Neural Network
- later version included a sonar range finder
  - ▶  $8 \times 32$  range finder input retina
  - ▶ 29 hidden units
  - ▶ 45 output units
- Supervised Learning, from human actions (Behavioral Cloning)
  - ▶ additional “transformed” training items to cover emergency situations
- drove autonomously from coast to coast

# 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
- ▶ mathematical theory: Bayes’ rule
- ▶ 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, and works well in practice – especially when negative examples vastly outweigh positive ones.

It also makes the backprop computations simpler

$$\begin{aligned} \frac{\partial E}{\partial z} &= \frac{z - t}{z(1 - z)} \\ \text{if } z &= \frac{1}{1 + e^{-s}}, \\ \frac{\partial E}{\partial s} &= \frac{\partial E}{\partial z} \frac{\partial z}{\partial s} = z - t \end{aligned}$$



# Maximum Likelihood

---

$H$  is a class of hypotheses

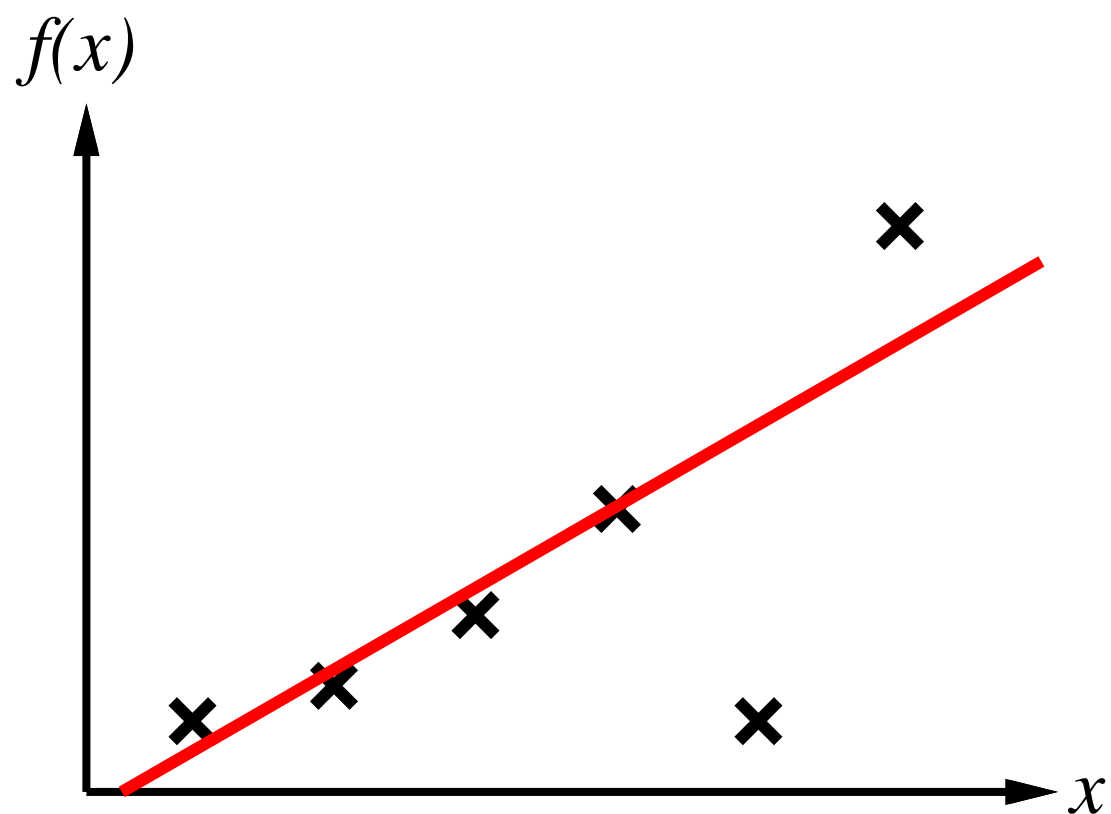
$P(D|h)$  = probability of data  $D$  being generated under hypothesis  $h \in H$ .

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

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

# Least Squares Line Fitting

---



# Derivation of Least Squares

---

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

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

(Note: we do not need to know  $\sigma$ )

# Derivation of Cross Entropy

---

For classification tasks,  $d$  is either 0 or 1.

Assume  $D$  generated by hypothesis  $h$  as follows:

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

then

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

(Can be generalized to multiple classes.)

# Bayes Rule

---

$H$  is a class of hypotheses

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

$P(h|D)$  = probability that  $h$  is correct, given that data  $D$  were observed.

Bayes' Theorem:

$$\begin{aligned} P(h|D)P(D) &= P(D|h)P(h) \\ P(h|D) &= \frac{P(D|h)P(h)}{P(D)} \end{aligned}$$

$P(h)$  is called the **prior**.

## Example: Medical Diagnosis

---

Suppose we have a 98% accurate test for a type of cancer which occurs in 1% of patients. If a patient tests positive, what is the probability that they have the cancer?

# 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  $\lambda$  from experience, or empirically.

# 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 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