Supervised Learning

GE 461, Spring 2020

Çiğdem Gündüz Demir
Department of Computer Engineering
Bilkent University
gunduz@cs.bilkent.edu.tr

Learning

- In our lives, we take actions based on
  - What we observe in our environments
  - What we have previously learned

Face recognition

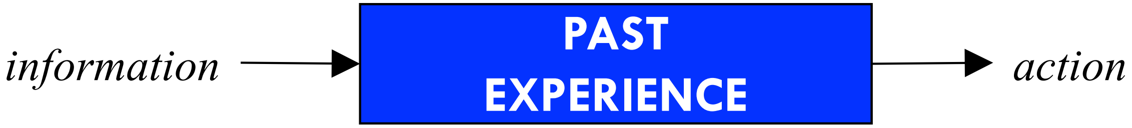
Handwritten character recognition

Chess playing

Car driving

Stock price prediction

- In order to achieve a task, we should
  - Have relevant information representing the environment
  - Know the possible set of actions
  - Know the process to take an action based on the information
    - This process relies on our past experience



information → **PAST EXPERIENCE** → *action*

Handwritten Letter Recognition

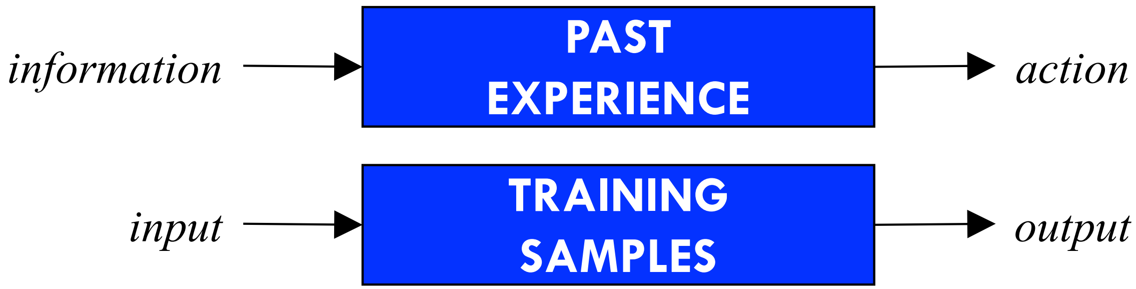
- Obtain information representing the environment
  - Letter to be recognized
  - Preferably its adjacent letters

- Know the possible set of actions
  - Number of letters
  - Language

- Take an action, which is affected by whether or not
  - You have seen that letter before
  - You know the alphabet of that language
  - You understand the context of that language

Machine Learning

- The goal of machine learning is to design systems that
  - Automatically achieves tasks (output) similar to us
  - Depending on the environment (input)
  - Based on the past experience (training samples)
  - With respect to some performance measures (e.g., accuracy)



Supervised learning:

- **There is a teacher providing a label (output) for each training sample**
- **The task is to map an input space to an output space**

Unsupervised learning:

- **There is no explicit teacher that provides sample labels (outputs)**
- **The task is to find regularities (clusters) in the input space**

Input/Output

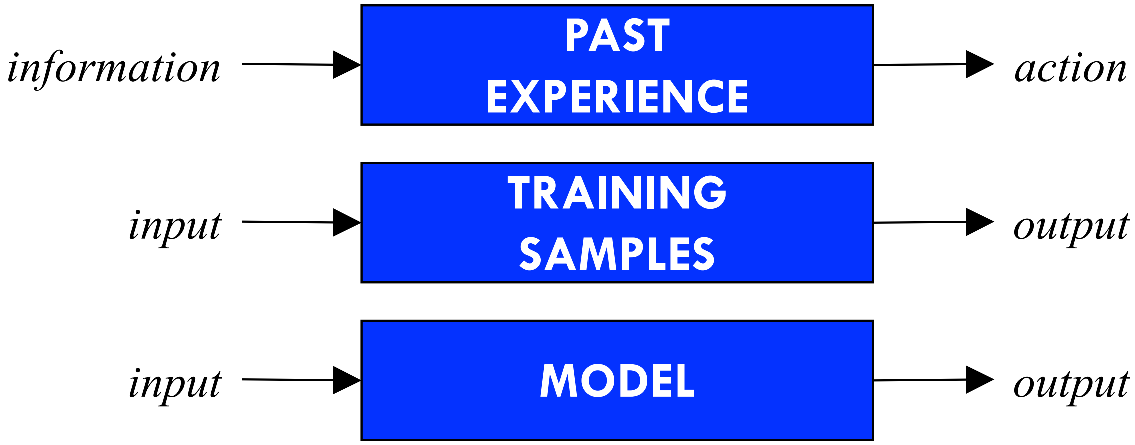
- We reduce the input measuring its certain properties (features), which can be numerical or non-numerical
  - Mileage (e.g., 34187)
  - Condition (e.g., poor, average, excellent)

- Deep learning aims at learning these features directly from data

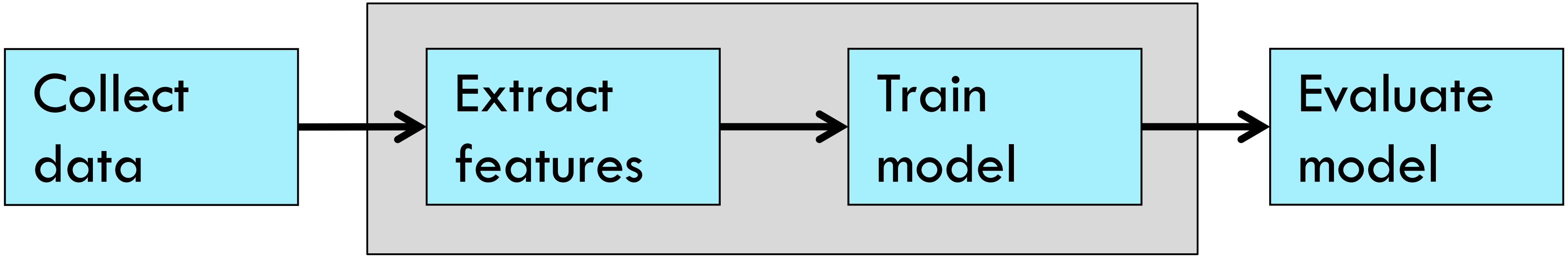
- The output can be discrete or continuous
  - A, C, Z for letter recognition (**classification**)
  - 25999 TL for car price prediction (**regression**)

Supervised Learning

- We believe that there is a process underlying training samples (their inputs and outputs)
  - We may not identify this process completely
  - But we can construct a model approximating the process
    - A function that distinguishes discrete outputs (**classification**)
    - A functional description of output in terms of inputs (**regression**)
  - **Supervised learning focuses on constructing such models**



Supervised Learning



Deep learning aims at learning the features directly from data while learning the model (thus, it combines these two steps)

The goodness of a model depends on

- How well its approximation is
  - No model fits all problems
  - Different models have different assumptions
- How well training samples represent the real-world
  - There may exist noise and exceptions in the samples
  - Some parts may not be covered by the samples

Model Evaluation

Accuracy:

- Percentage of correctly classified samples
- We may also want to consider class-based accuracies, especially when there is an unbalanced distribution among classes

Confusion matrix:

		Predicted class			
		C_1	C_2	…	C_C
True class	C_1				
	C_2				
	…				
	C_C				

Do not use the same set of (training) samples both for learning a model and its evaluation!!!

- If available, use an independent test set
- If not, create multiple independent training and test sets by partitioning samples many times
  - ✓ Bootstrapping (draw samples with replacement)
  - ✓ K-fold cross-validation (split samples into k folds)
  - ✓ Leave-one-out (form partitions, each contains a single sample)

Bayesian Decision Theory

Bayesian Decision Theory

- It is the fundamental statistical approach in classification

- Here it is assumed that
  1. The decision problem is posed in probabilistic terms and
  2. All relevant probability values are known

Bayesian Decision Theory

- A simple decision problem: **Fish classification**

- Let's assume that a fish emerges nature in one of the following states

State of nature: $C = \begin{cases} C_1 & \text{for } hamsi \\ C_2 & \text{for } barbun \end{cases}$

- To predict what type will emerge next, we consider C as a random variable, which is described probabilistically

Prior probabilities (a priori probabilities): P(C_1) and P(C_2) reflect our previous knowledge before the fish appears

$P(C_1) + P(C_2) = 1$ *(if no other species exist)*

Bayesian Decision Theory

- Decide whether a fish is hamsi or barbun when
  1. We are not allowed to see the fish
  2. We know the prior probabilities
  3. The cost is the same for all incorrect decisions

Decision rule: Select $\begin{cases} hamsi & \text{if} \quad P(C_1) > P(C_2) \\ barbun & \text{otherwise} \end{cases}$

In this case, we always make the same decision!!!

Bayesian Decision Theory

- Fortunately, we usually have more information for making our decisions
  - E.g., we can see the fish, measure its color intensity
  - We make this measurement relying on the fact that hamsi and barbun emerge nature in different colors

- This difference can be expressed in probabilistic terms, considering color intensity x as a continuous random var, whose distribution depends on the state of nature

Class-conditional probability density functions (likelihoods):
$P(x|C_1)$ and $P(x|C_2)$ are the probability of observing color intensity x when the state of nature is C_1 and C_2, respectively

Bayesian Decision Theory

- Now let’s combine this measurement with our previous knowledge

Joint probability

$$P(C_j, x) = P(C_j | x) \cdot P(x) = P(x | C_j) \cdot P(C_j)$$

BAYES FORMULA

$$\underbrace{P(C_j | x)}_{Posterior} = \frac{\overbrace{P(x | C_j)}^{Likelihood} \cdot \overbrace{P(C_j)}^{Prior}}{\underbrace{P(x)}_{Evidence}}$$

$$P(x) = \sum_{j=1}^{N} P(x | C_j) \cdot P(C_j)$$

$$\sum_{j=1}^{N} P(C_j | x) = 1$$

Posterior probabilities (a posteriori probabilities): $P(C_1|x)$ and $P(C_2|x)$ reflect our beliefs of having a particular fish species when the color intensity of the fish is measured as x

Bayesian Decision Theory

- Now decide whether a fish is hamsi or barbun when
  1. We can see the fish and measure its color x
  2. We know the prior probabilities and likelihoods
  3. The cost is the same for all incorrect decisions

Decision rule: Select $\begin{cases} hamsi & \text{if} \quad P(C_1 \mid x) > P(C_2 \mid x) \\ barbun & \text{otherwise} \end{cases}$

We use the Bayes' decision rule to minimize the probability of error

$$P(error) = \int_{-\infty}^{\infty} P(error,x)\ dx = \int_{-\infty}^{\infty} P(error \mid x)\ P(x)\ dx$$

For every x, keep P(error|x) as small as possible, by selecting the state of nature (class) with the highest posterior probability

Bayesian Decision Theory

- Now decide whether a fish is hamsi or barbun when
  1. We can see the fish and measure its color x
  2. We know the prior probabilities and likelihoods
  3. The cost is the same for all incorrect decisions

Decision rule: Select $\begin{cases} hamsi & \text{if} \quad P(C_1 | x) > P(C_2 | x) \\ barbun & \text{otherwise} \end{cases}$

Select $\begin{cases} hamsi & \text{if} \quad P(x | C_1) \cdot P(C_1) > P(x | C_2) \cdot P(C_2) \\ barbun & \text{otherwise} \end{cases}$

- Evidence is unimportant since it is the same for all states of nature
- *Equal priors* → Observing each state of nature is equally likely
- *Equal likelihoods* → Measurement x gives no information

Bayesian Decision Theory

- Now let’s generalize the decision problem

States of nature	$\{C_1 , C_2 , \ldots C_c\}$
Possible actions	$\{\alpha_1 , \alpha_2 , \ldots \alpha_a\}$
Loss function	$\lambda(\alpha_i \mid C_j)$

Let $x \in R^d$ be a feature vector in a d - dimensional space

For this x, we would take the action α_i that minimizes

the loss $\lambda(\alpha_i \mid C_j)$ if we knew C_j is its true state of nature

However, we do not know the true state of nature

Thus, we will take the action based on expectation

Bayesian Decision Theory

- The expected loss associated with taking action α_i

$$\underbrace{R(\alpha_i \mid x)}_{\text{Conditional risk}} = \sum_{j=1}^{C} P(C_j \mid x) \cdot \lambda(\alpha_i \mid C_j)$$

$$P(C_j \mid x) = \frac{P(x \mid C_j) \cdot P(C_j)}{P(x)}$$

- We take the action that minimizes the conditional risk

$$\underbrace{\alpha^*}_{\text{Optimal action}} = \arg\min_i R(\alpha_i \mid x)$$

The resulting minimum risk R* is called *Bayes risk*

Minimum Error-Rate Classification

- In multi-class classification
  - Each state of nature is usually associated with a class
  - Each action is usually interpreted as deciding on a class (sometimes other actions —e.g., reject action, are defined)
  - Zero-one loss function is commonly used

$$\lambda(\alpha_i \mid C_j) = \begin{cases} 0 & \text{if } i = j \quad \text{(correct classification)} \\ 1 & \text{if } i \neq j \quad \text{(all incorrect classifications)} \end{cases}$$

The optimal action is

$$\alpha^* = \arg\max_i P(C_i \mid x)$$

When zero-one loss function is used, selecting the action that minimizes conditional risk is equivalent to selecting the action that maximizes posterior probability

Classifiers and Discriminant Functions

- A classifier is represented with a set of discriminant functions $g_j(x)$ for $j = 1, 2, \ldots C$

- A given instance x is then classified with the class C_j for which the discriminant function $g_j(x)$ is the maximum

1. **Likelihood-based approaches**
2. **Discriminant-based approaches**

Likelihood-Based Approaches

- They estimate class probabilities on training samples and then use them to define the discriminant functions

- Bayes classifier
  - Defines a discriminant function using the conditional risk

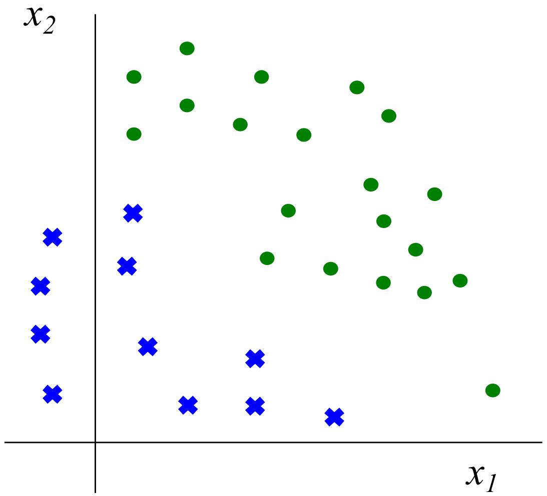
$$g_j(x) = -R(\alpha_j \mid x)$$

$$g_j(x) = \underbrace{P(C_j \mid x)}_{\textit{when 0-1 loss function is used}}$$

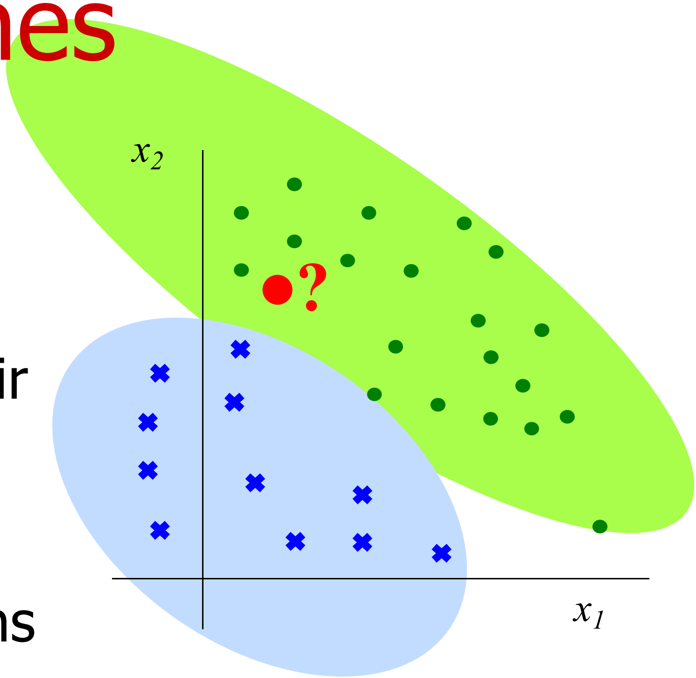
$$g_j(x) = \hat{P}(C_j \mid x)$$

$$= \frac{\hat{P}(x \mid C_j) \cdot \hat{P}(C_j)}{\hat{P}(x)}$$

$$\equiv \underbrace{\hat{P}(x \mid C_j) \cdot \hat{P}(C_j)}_{\textit{For each class, estimate the prior and the likelihood on the training samples that belong to this class}}$$



Likelihood-Based Approaches



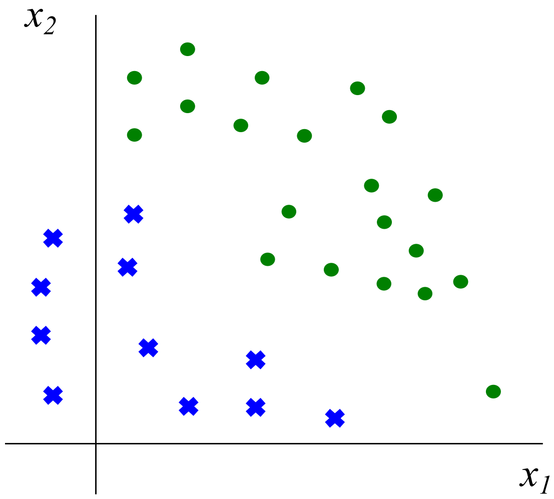
- **Parametric approach**
  - Assumes a parametric form on the probability distributions and estimate their parameters on training samples
  - For a given instance x, it estimates its class probabilities using these distributions
  - Maximum likelihood estimation and Bayesian estimation
- **Nonparametric approach**
  - Does not have such assumption
  - It estimates the class probabilities of the instance x using the nearby points of this instance
  - Parzen windows, k-nearest neighbors

Discriminant-Based Approaches

- They learn discriminant functions directly on training samples

- They make an assumption on the form of discriminant functions and learn their parameters on training samples without estimating class probabilities

- **Linear discriminants assume that each discriminant function is a linear combination of the input features**



Linear Discriminants

Linear Discriminants

- They define $g_j(x)$ as a linear combination of the input features

number of input dimensions

$$g_j(x \mid W_j) = \sum_{i=1}^{d} W_{ij}\, x_i + W_{0j}$$

weight vector for the j-th class

let's define $x_0 = 1$

$$g_j(x \mid W_j) = \sum_{i=0}^{d} W_{ij}\, x_i$$

- Learning involves learning the parameters (weights) W_j for each class C_j from the training samples
- For that, we will define a criterion function and learn the weights that minimize/maximize this function

Linear Discriminants

- They yield hyperplane decision boundaries

Consider two-class classification

$$g_1(x) = \sum_{i=1}^{d} W_{i1}\, x_i \; + \; W_{01}$$

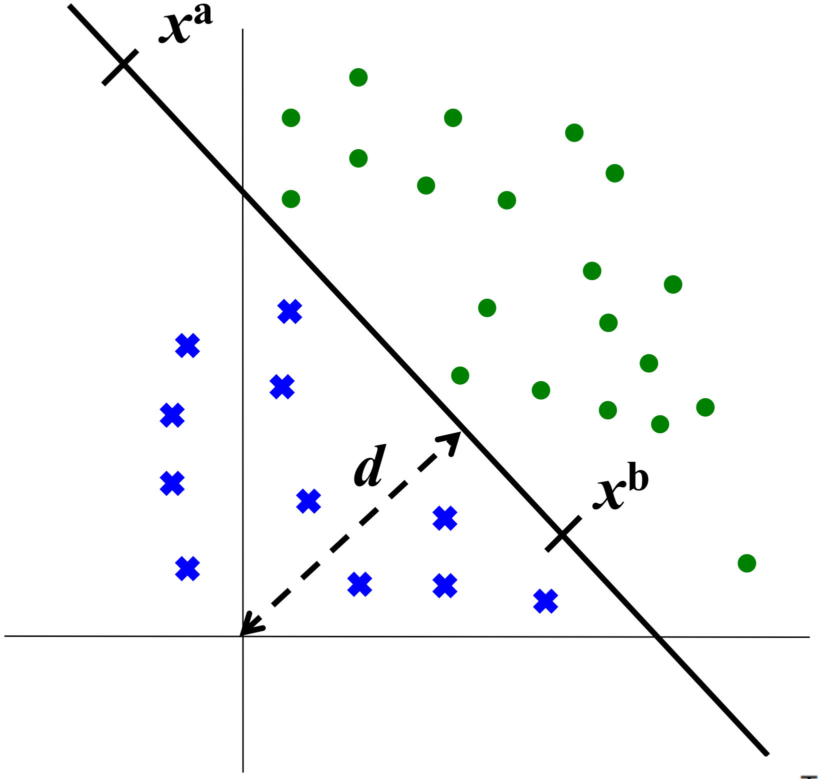
$$\left.\begin{aligned} g_1(x) &= W_1^{\mathsf{T}} x \; + \; W_{01} \\ g_2(x) &= W_2^{\mathsf{T}} x \; + \; W_{02} \end{aligned}\right\} \quad g(x) = g_1(x) - g_2(x) \qquad \text{choose} \quad \begin{aligned} &C_1 \quad \text{if } g(x) \geq 0 \\ &C_2 \quad \text{otherwise} \end{aligned}$$

$$g(x) = \left(W_1 \; - \; W_2\right)^{\mathsf{T}} x + \left(W_{01} \; - \; W_{02}\right)$$

$$g(x) = \underbrace{W^{\mathsf{T}} x \; + \; W_0}_{\textit{This is another linear function}}$$

Linear Discriminants

Consider two-class classification



$$g(x) = W^{\top} x + W_0$$

Let's take two points on the decision plane

$$g(x^a) = g(x^b)$$

$$W^{\top} x^a + W_0 = W^{\top} x^b + W_0$$

$$\underbrace{W^{\top}(x^a - x^b) = 0}$$

W determines the hyperplane's orientation
(W is normal to any vector on the hyperplane)

W_0 determines the hyperplane's location with respect to the origin

Linear Discriminants

We consider multi-class classification as

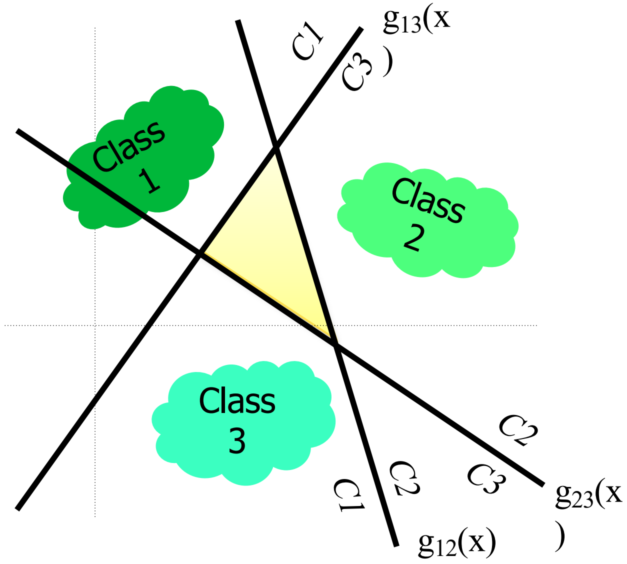
1. one-against-one **OR** **2. one-against-all**

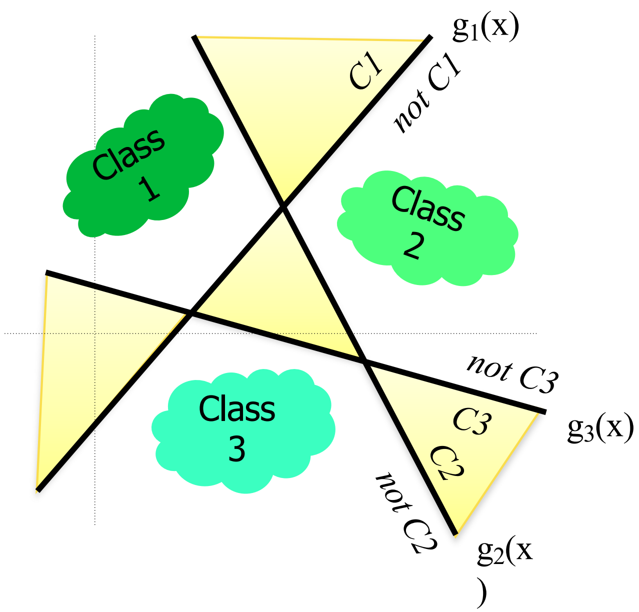
C (C – 1) / 2 discriminants

$$g_{kj}(x) = \begin{cases} \geq 0 & \text{if} \quad x \in C_k \\ < 0 & \text{if} \quad x \notin C_j \\ \text{don't care otherwise} \end{cases}$$

C discriminants

$$g_k(x) = \begin{cases} \geq 0 & \text{if} \quad x \in C_k \\ < 0 & \text{if} \quad x \notin C_k \end{cases}$$



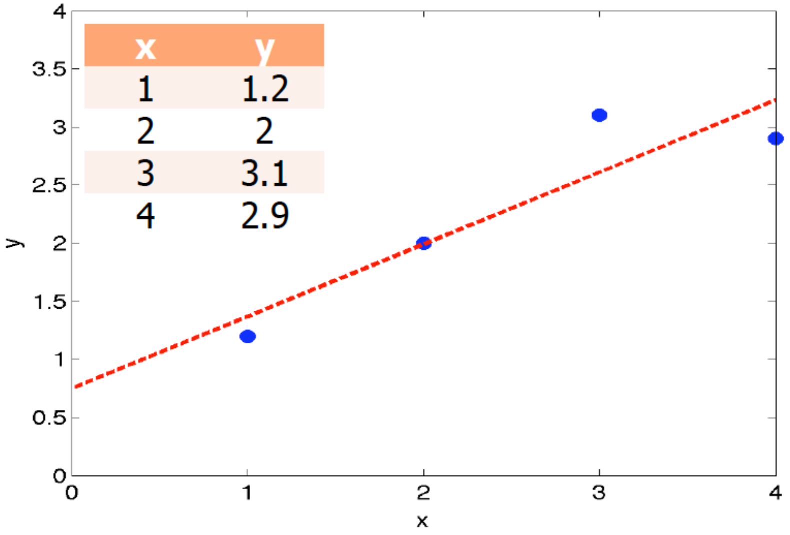


The most common way to resolve ambiguities is to select the class for which the discriminant is highest (one may also take the reject action for ambiguities)

How to Learn?

Although we will use linear discriminants for classification, let's first consider a linear regression problem

Construct a linear model on the following data points



x	y
1	1.2
2	2
3	3.1
4	2.9

<u>construct a linear model</u>

$$f(x) = W\,x + W_0$$

<u>define a criterion function</u>

$$\underbrace{loss(W,\ W_0)}_{\textbf{Sum of squared errors}} = \frac{1}{2}\sum_t \left(f(x^t) - y^t\right)^2$$

select W and W_0 that minimize this error on the training samples

$$\frac{\partial loss}{\partial W} = 0 \qquad \frac{\partial loss}{\partial W_0} = 0$$

Analytical Solution

$$loss(W\,,\,W_0) = \frac{1}{2}\sum_t \left(f(x^t) - y^t\right)^2$$

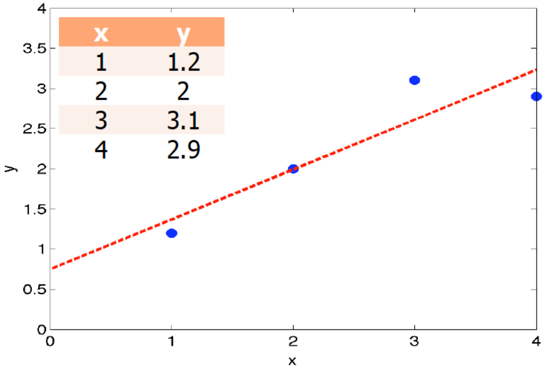
$$f(x) = W\,x + W_0$$

$$\frac{\partial loss}{\partial W} = \frac{1}{2}\,2\sum_t \left(W\,x^t + W_0 - y^t\right)x^t = W\sum_t x^{t2} + W_0\sum_t x^t + \sum_t x^t\,y^t = 0$$

$$\frac{\partial loss}{\partial W_0} = \frac{1}{2}\,2\sum_t \left(W\,x^t + W_0 - y^t\right) = W\sum_t x^t + W_0\,N + \sum_t y^t = 0$$

In our example

$$\sum x^t = 10$$
$$\sum y^t = 9.2$$
$$\sum x^{t2} = 30$$
$$\sum x^t y^t = 26.1$$
$$N = 4$$



x	y
1	1.2
2	2
3	3.1
4	2.9

$$\left.\begin{aligned} &30\ W + 10\ W_0 - 26.1 = 0 \\ &10\ W + 4\ W_0 - 9.2 = 0 \end{aligned}\right\} \quad \begin{aligned} &\text{2 unknowns} \\ &\text{2 equations} \end{aligned} \Rightarrow \begin{aligned} W &= 0.62 \\ W_0 &= 0.75 \end{aligned}$$

$$f(x) = 0.62\ x\ +\ 0.75$$

Sometimes it is hard to analytically solve
OR
There may be no analytical solution at all (if the linear system has a singular matrix, no solution or multiple solutions exist)

↓

ITERATIVE OPTIMIZATION METHODS

Gradient Descent Algorithm

- One commonly used iterative optimization method

- Goal is to find the parameters that minimize the loss
  - Starting with random parameters, it iteratively updates them in the direction of the steepest descent (in the opposite direction of the gradient) until the gradient is zero (or small enough)

```
start with random weights $W_i$
do
    $\Delta W_i = -\eta \frac{\partial loss_{ALL}}{\partial W_i}$    for all $i$
    $W_i = W_i + \Delta W_i$    for all $i$
until convergence
```

η is the learning rate, which determines how much to move in the direction of the steepest descent

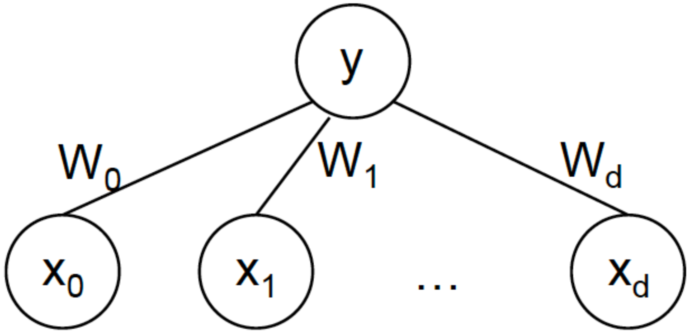
→ if it is too small, convergence is slow

→ if it is too large, we may overshoot the minimum (divergence might occur)

This method finds the nearest minimum, which could be local. It does not guarantee to find the global minimum

Regression

Let's derive the update rules for regression



$$loss_{ALL}(W) = \sum_t loss^t$$

$$loss = \frac{1}{2}\left(f(x) - y\right)^2$$

$$f(x) = net$$

$$net = \sum_i x_i W_i$$

$$\Delta W_i = -\eta \frac{\partial loss_{ALL}(W)}{\partial W_i}$$

$$\frac{\partial loss}{\partial W_i} = \frac{\partial loss}{\partial net} \frac{\partial net}{\partial W_i}$$

$$\frac{\partial loss}{\partial W_i} = \delta x_i$$

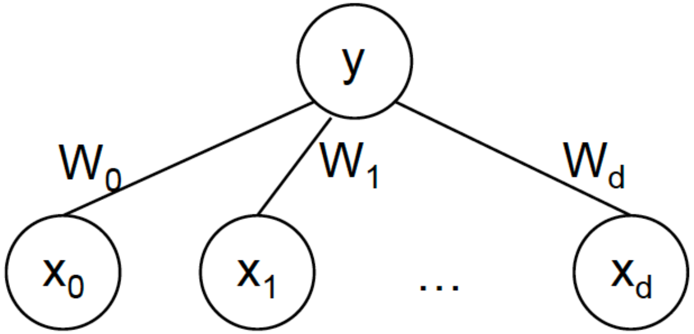
$$\delta = \left(net - y\right)$$

$$\Delta W_i = -\eta \sum_t \delta x_i$$

$$\boxed{\Delta W_i = \eta \sum_t \left(y^t - net^t\right) x^t{}_i}$$

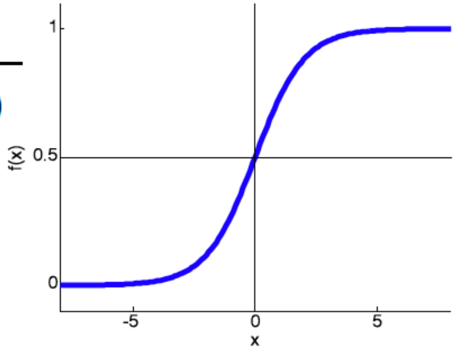
Classification (Logistic Regression)

Let’s derive the update rules for 2-class classification



Logarithmic sigmoid function

$$\sigma(net) = \frac{1}{1+\exp(-net)}$$



$$y = 1 \quad \text{if } x \in C_1$$

$$y = 0 \quad \text{if } x \in C_2$$

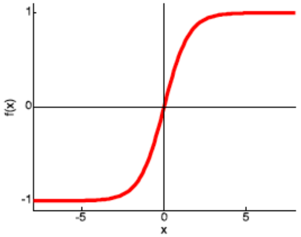
$$\boxed{f(x) = \sigma(net)}$$

$$net = \sum_i x_i\, W_i$$

$$\sigma'(net) = \sigma(net)\,\big(1-\sigma(net)\big)$$

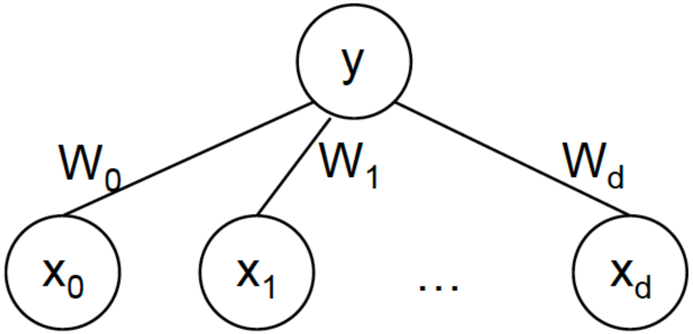
Hyperbolic tangent sigmoid function

$$f(x) = a\ \tanh(b\,x) = a\left[\frac{\exp(b\,x)-\exp(-b\,x)}{\exp(b\,x)+\exp(-b\,x)}\right]$$



Classification (Logistic Regression)

Let’s derive the update rules for 2-class classification



$$\boxed{f(x) = \sigma(net)}$$

$$net = \sum_i x_i \, W_i$$

$$\frac{\partial loss}{\partial W_i} = \frac{\partial loss}{\partial net} \; \frac{\partial net}{\partial W_i}$$

$$\frac{\partial loss}{\partial W_i} = \delta \, x_i$$

$$loss_{ALL}(W) = \sum_t loss^t$$

$$\underbrace{loss = -y \, \log f(x) \; -}_{\text{Cross entropy}} \; (1-y) \, \log \, (1 - f(x))$$

$$\underbrace{loss = \frac{1}{2} \left(f(x) - y\right)^2}_{\text{Squared error}}$$

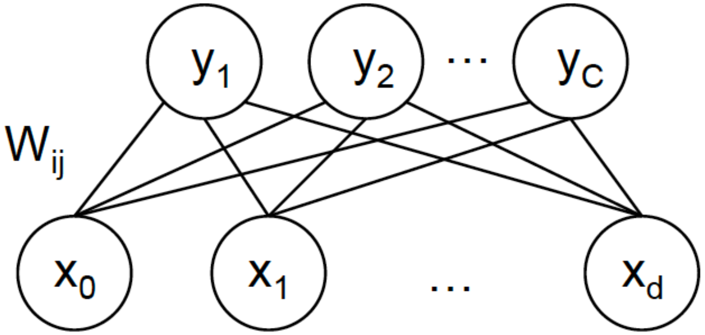
When squared error is used

$$\boxed{\delta = \left(\sigma(net) - y\right) \, \sigma'(net)}$$

$$\boxed{\Delta W_i = \eta \sum_t \left(y^t - \sigma(net^t)\right) \sigma(net^t) \; (1 - \sigma(net^t)) \; x^t{}_i}$$

Classification

Let’s derive the update rules for multiclass classification



Define output as a C-dimensional vector

$$y_j = 1 \quad \text{if } x \in C_j$$
$$y_j = 0 \quad \text{if } x \notin C_j$$
$$f_j(x) = softmax(net_j)$$
$$net_j = \sum_i x_i W_{ij}$$

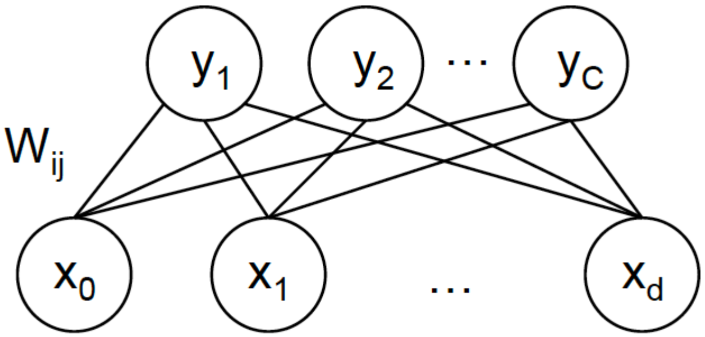
$$softmax(net_j) = \frac{\exp(net_j)}{\sum_m \exp(net_m)}$$

$$\frac{\partial\, softmax(net_k)}{\partial\, net_j} = softmax(net_j)\left(\delta_{jk} - softmax(net_k)\right)$$

$$\left.\delta_{jk} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases}\right\} \text{Kronecker delta}$$

Classification

Let’s derive the update rules for multiclass classification



$$f_j(x) = softmax(net_j)$$

$$net_j = \sum_i x_i W_{ij}$$

$$\frac{\partial loss}{\partial W_{ij}} = \frac{\partial loss}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}$$

$$\frac{\partial loss}{\partial W_{ij}} = \delta_j x_i$$

$$loss_{ALL}(W) = \sum_t loss^t$$

$$\underbrace{loss = -\sum_k y_k \log f_k(x)}_{\text{Cross entropy}}$$

$$\underbrace{loss = \frac{1}{2} \sum_k \left(f_k(x) - y_k\right)^2}_{\text{Squared error}}$$

When squared error is used

$$\boxed{\delta_j = \sum_k \left(softmax\,(net_k) - y_k\right) \frac{\partial\, softmax(net_k)}{\partial\, net_j}}$$

$$\boxed{\Delta W_{ij} = \eta \sum_t \sum_k \left(y_k^t - \underset{\overset{\uparrow}{softmax(net^t_k)}}{s(net_k^t)}\right) s(net_j^t)\ (\delta_{jk} - s(net_k^t))\ x^t{}_i}$$

Batch learning algorithm

```
start with random weights $W_{ij}$
do
    compute $f_j(x^t) = softmax\left(\sum_i x^t{}_i\, W_{ij}\right)$        for all $t$ and $j$
    compute $\Delta W_{ij} = -\eta \sum_t \delta_j\, x_i^t$                  for all $i$ and $j$
    update $W_{ij} = W_{ij} + \Delta W_{ij}$                                for all $i$ and $j$
until convergence
```

EPOCH

Stochastic learning algorithm

```
start with random weights $W_{ij}$
do
    for all $(x^t , y^t)$ in random order
        compute $f_j(x^t) = softmax\left(\sum_i x^t{}_i\, W_{ij}\right)$    for all $j$
        compute $\Delta W_{ij} = -\eta\, \delta_j\, x_i^t$                  for all $i$ and $j$
        update $W_{ij} = W_{ij} + \Delta W_{ij}$                            for all $i$ and $j$
until convergence
```

EPOCH

Mini-batch stochastic learning algorithm is a good tradeoff

Adding Nonlinearity

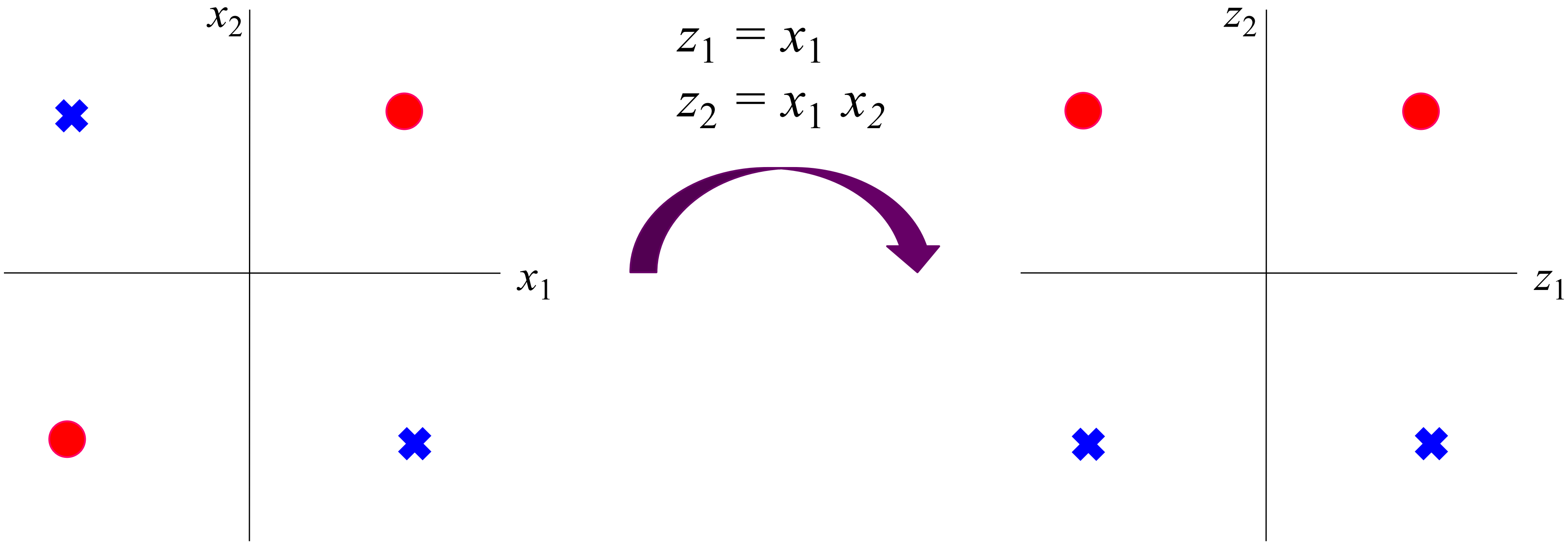
- Linear discriminants yield hyperplane decision boundaries

- If they are not sufficient to construct a “good” model

  1. Transform the space into a new one using nonlinear mappings and construct linear discriminants on the transformed space → **Support vector machines**

  2. Learn nonlinearity at the same time as you learn the linear discriminants → **Neural networks**

XOR Problem

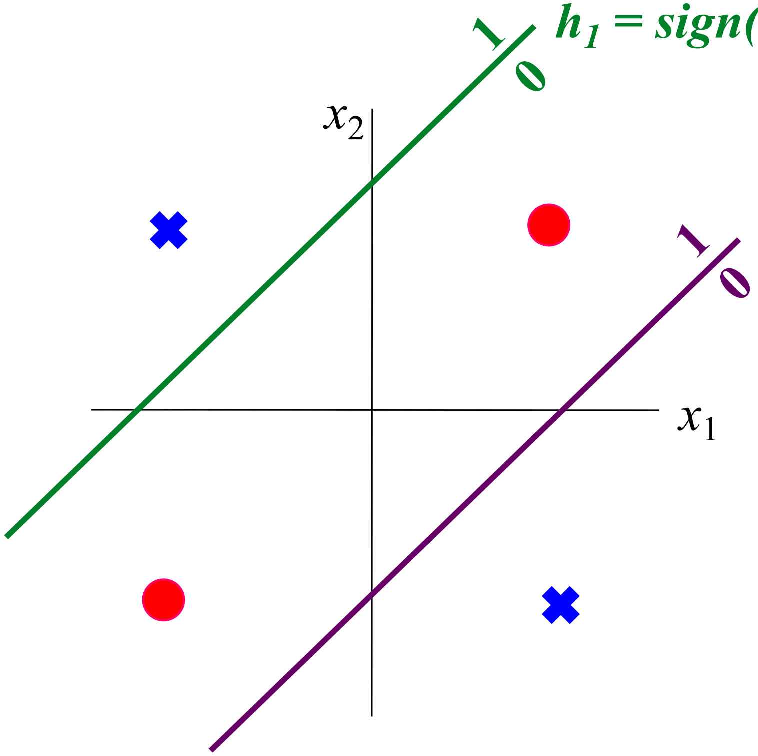


$$z_1 = x_1$$
$$z_2 = x_1\ x_2$$

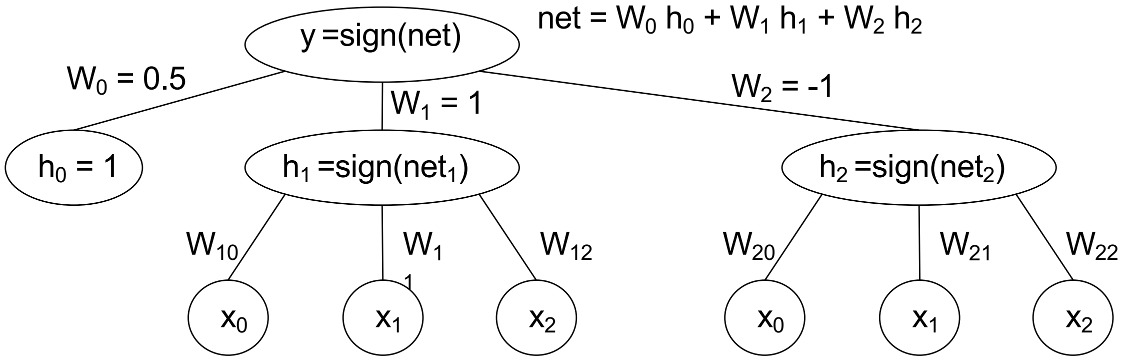
Support vector machines use the idea of nonlinear mapping to find a linearly separable space

XOR Problem

$$net_1 = W_{12}\ x_2 + W_{11}\ x_1 + W_{10}$$
$$h_1 = sign(\ net_1\ )$$



$$net_2 = W_{22}\ x_2 + W_{21}\ x_1 + W_{20}$$
$$h_2 = sign(\ net_2\ )$$

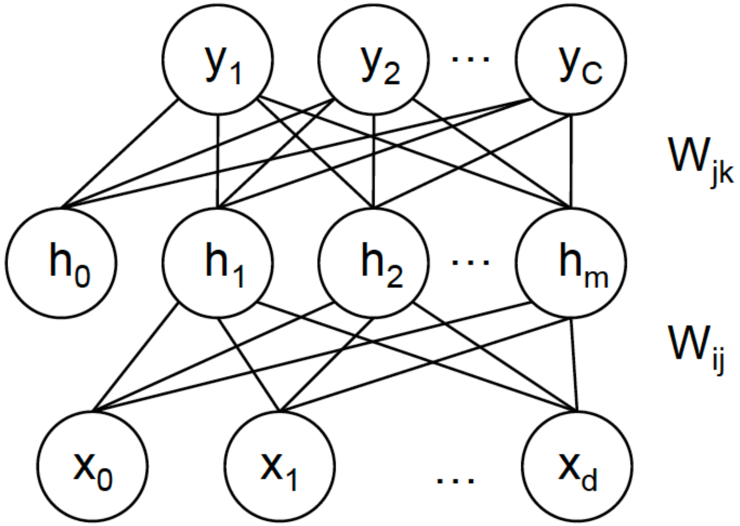


Neural networks learn the nonlinearity at the same time as they learn the linear discriminants
(learn all the weights at the same time)

Neural Networks

Multilayer Perceptrons

- Also contain hidden layers in addition to input and output layers



Hidden units h_j's can be viewed as new "features" obtained by combining x_i's

A deeper architecture with nonlinear activations is more expressive than a shallow one

In this network

1. Each hidden unit computes its net activation $$net_j = \sum_i x_i W_{ij}$$
2. Each hidden unit emits an output that is a nonlinear function of its activation $$h_j = \sigma(net_j)$$
3. Each output unit computes its net activation $$net_k = \sum_j h_j W_{jk}$$
4. Each output units emits an output $$y_k = g(net_k)$$

How to Learn?

- In linear discriminants, we select the weights to minimize a loss function defined on the difference between the actual and computed outputs

- In multilayer structures, we can also select the hidden-to-output-layer weights to minimize a loss function defined on the actual and computed outputs

- However, we cannot select the input-to-hidden-layer weights in a similar way since we do not know the actual values of the hidden units

- Thus, to learn the input-to-hidden-layer weights, we propagate the loss function (defined on the outputs) from the output layer to the corresponding hidden layer
  → **BACKPROPAGATION ALGORITHM**

Backpropagation Algorithm

Let's derive the update rules for multiclass classification

$$net_j = \sum_i x_i W_{ij}$$

$$h_j = \sigma(net_j)$$

$$net_k = \sum_j h_j W_{jk}$$

$$f_k(x) = softmax(net_k)$$

$$loss_{ALL}(W) = \sum_t loss^t$$

$$\underbrace{loss = \frac{1}{2} \sum_k \left(f_k(x) - y_k\right)^2}_{\text{Squared error}}$$

$$\Delta W_{jk} = -\eta \frac{\partial loss_{ALL}(W)}{\partial W_{jk}}$$

$$\frac{\partial loss}{\partial W_{jk}} = \frac{\partial loss}{\partial net_k} \frac{\partial net_k}{\partial W_{jk}}$$

$$\frac{\partial loss}{\partial W_{jk}} = \delta_k h_j$$

$$\delta_k = \sum_m \left(softmax\,(net_m) - y_m\right) \frac{\partial\, softmax(net_m)}{\partial\, net_k}$$

Hidden-to-output-layer weights

Backpropagation Algorithm

Let’s derive the update rules for multiclass classification

$$net_j = \sum_i x_i W_{ij}$$

$$h_j = \sigma(net_j)$$

$$net_k = \sum_j h_j W_{jk}$$

$$f_k(x) = softmax(net_k)$$

$$loss_{ALL}(W) = \sum_t loss^t$$

$$\underbrace{loss = \frac{1}{2}\sum_k \left(f_k(x) - y_k\right)^2}_{\text{Squared error}}$$

$$\Delta W_{ij} = -\eta \frac{\partial loss_{ALL}(W)}{\partial W_{ij}}$$

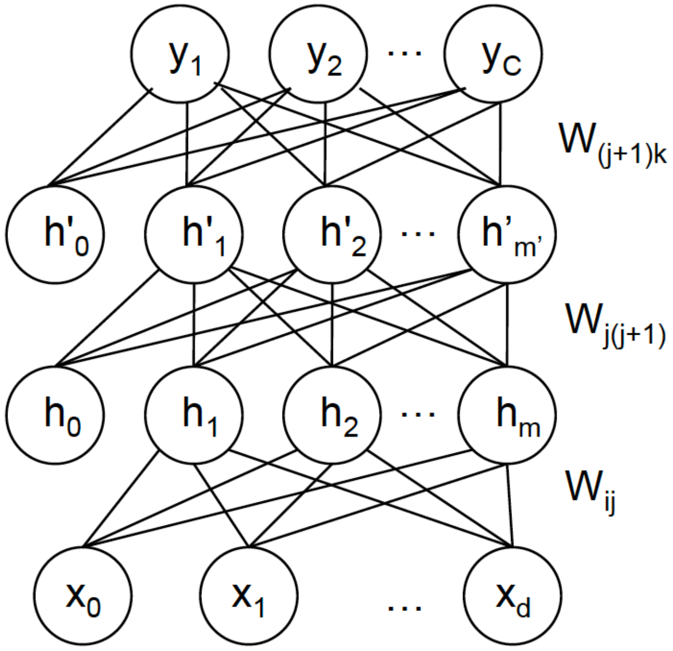
$$\frac{\partial loss}{\partial W_{ij}} = \frac{\partial loss}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}$$

$$\frac{\partial loss}{\partial W_{ij}} = \delta_j x_i$$

$$\delta_j = \sum_k \frac{\partial loss}{\partial net_k} \frac{\partial net_k}{\partial net_j} = \left[\sum_k \delta_k W_{jk}\right] \sigma'(net_j)$$

Input-to-hidden layer weights

More Hidden Layers



$$\frac{\partial loss}{\partial W_{ij}} = \frac{\partial loss}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}$$

$$\frac{\partial loss}{\partial W_{ij}} = \delta_j x_i$$

$$\delta_j = \sum_{(j+1)} \frac{\partial loss}{\partial net_{(j+1)}} \frac{\partial net_{(j+1)}}{\partial net_j}$$

$$\delta_j = \left[\sum_{(j+1)} \delta_{(j+1)} W_{j(j+1)} \right] \sigma'(net_j)$$

δ_j may vanish after repeated multiplication.
This makes deep architectures hard to train
(when initial weights are not "good" enough)

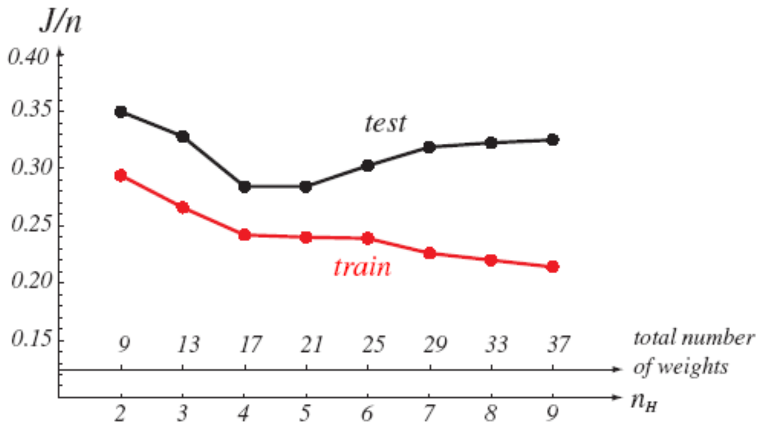
Approaches for alleviating underfitting and overfitting problems

- Better network designs: Sparse connections, weight sharing, convolutional nets, long/short skip connections, activation functions, …
- Better network training: regularization, loss function definitions, larger datasets, data augmentation, …
- Previously, layerwise pretraining (restricted Boltzmann machines, autoencoders)

Network Topology

The number of hidden units and hidden layers

- It controls the expressive power of the network
- Thus, the complexity of the decision boundary
- No foolproof method to set them before training
  - Few hidden units/layers will be enough if samples are well-separated
  - More will be necessary if samples have complicated densities



Hidden units more than necessary

- Network is tuned to the particular training set (overfitting)
- Training error can become small, but test error is unacceptably high

Too few hidden units

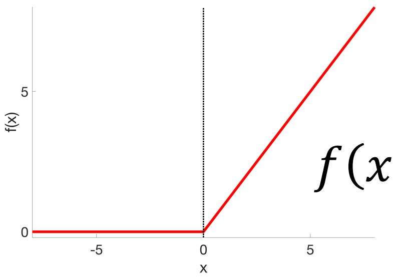
- Network does not have enough free parameters to fit the training set well
- Training and test errors are high

You will see different deep network architectures next week!!!

Some Practical Issues

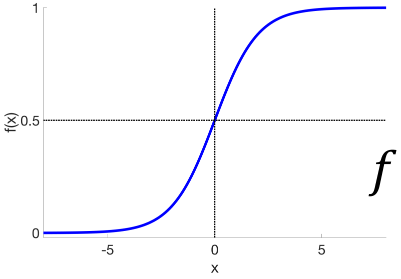
- Commonly used activation functions

Rectified linear unit (ReLU)



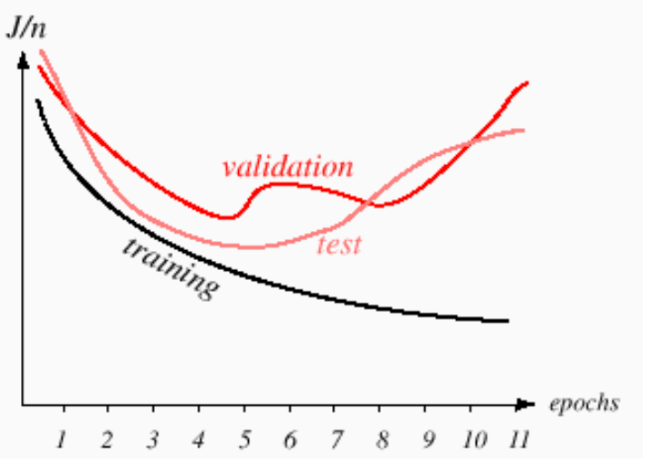
$$f(x) = \max(0, x)$$

Logarithmic sigmoid function



$$f(x) = \frac{1}{1 + \exp(-x)}$$

- When to stop?



Validation data can be used:

- Training error ultimately reaches an asymptotic value
- The error on an independent test set is expected to be higher
  - Although it usually decreases, it can also increase or oscillate

Some Practical Issues

1. Unbalanced class distributions
   - Classifiers typically favor the majority class(es)

2. Small training sets
   - Data augmentation is typically useful

3. Features with different orders of magnitudes
   - A neural network adjusts weights in favor of features with higher magnitudes
   - Normalization/scaling is typically useful

4. High feature values
   - May cause the exploding gradient problem
   - Normalization/scaling is typically useful

Many of them are indeed issues not only for neural networks but also for many other classifiers

Some Practical Issues

Regularization reduces sensitivity to training samples and decreases the risk of overfitting

$$loss_{ALL}(W) = \frac{1}{T}\sum_{t} loss^{t} + \|W\|$$

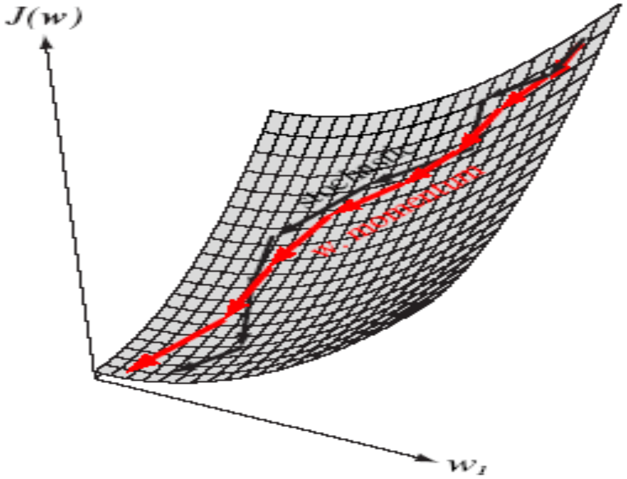
$$loss_{ALL}(W) = \underbrace{\frac{1}{2\,T}\sum_{t}\sum_{k}\left(f_k(x^t) - y_k^t\right)^2}_{\text{Mean squared error}} + \underbrace{\frac{\lambda}{2\,T}\|W\|_2^2}_{\text{L2-regularization term}} \qquad \text{where } \|W\|_2^2 = \sum_{i} W_i^{\,2}$$

Dropout regularization

- During training, in each iteration, randomly drop out units (also their incoming and outgoing connections) with probability p to sample a “thinned” network and train it
- Training can be seen as training a collection of different thinned networks with extensive weight sharing
- In testing, consider the entire network where the weights are scaled down by multiplying them a factor of 1 – p

Some Practical Issues

- Momentum helps speed up learning especially when when there are plateaus in error surfaces



Some fraction of the previous weight updates is included into the current update rule

$$w^{(t+1)} = w^{(t)} + (1-\alpha)\,\Delta w^{(t)} + \alpha\,\Delta w^{(t-1)}$$

Selection of initial weights as well as selection/update of learning rate, momentum constant, dropout factor, etc. may greatly affect learning

For some, optimization methods (e.g., AdaDelta, Adam) are available