Artificial Neural Networks
人工神经网络

权小军教授
中山大学计算机学院

quanxj3@mail.sysu.edu.cn

2023 年 3 月 13 日

Lecture 3 Neural Networks Training I: Overview

(*Part of the slides are adapted from CMU 11-785*)

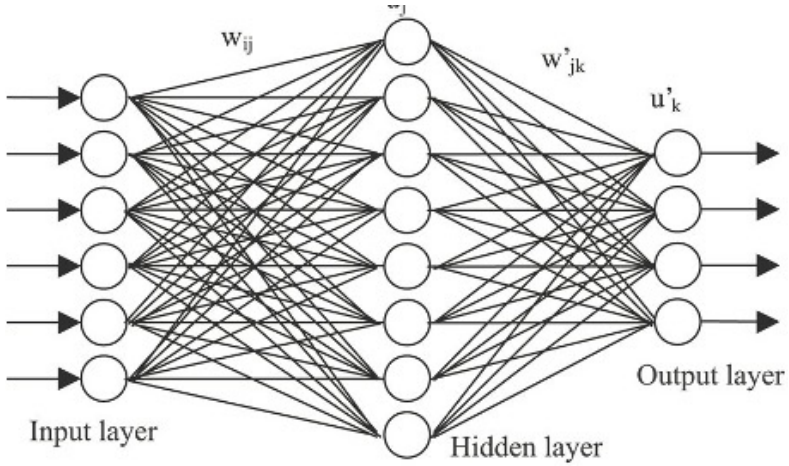
Recap

Recap

Suppose we have a function of any form:

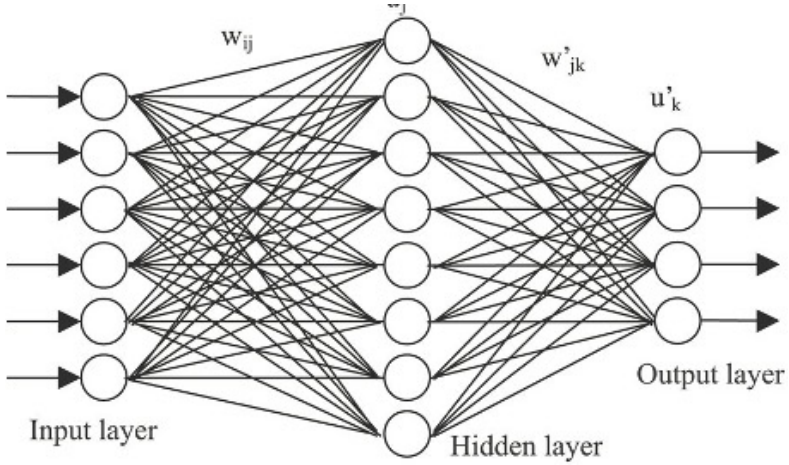
$$y = f(x)$$

Recap: neural networks are universal function approximators 通用函数逼近器



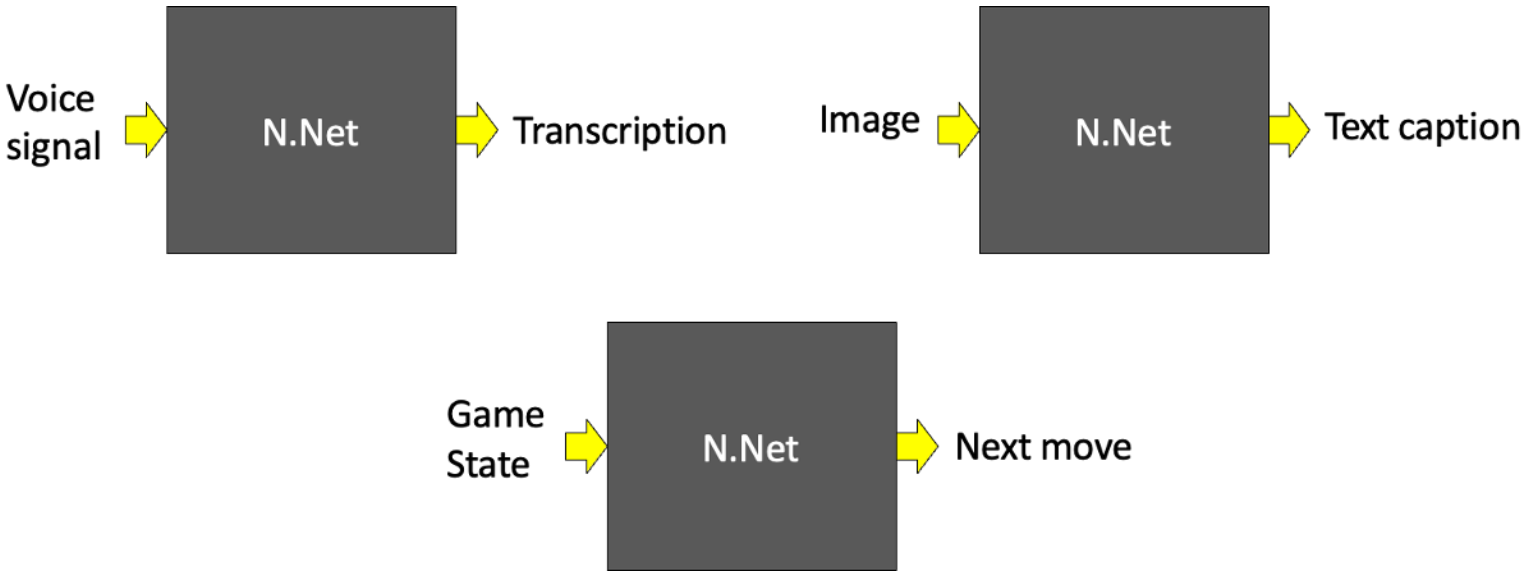
- Neural networks are universal function approximators
  - Can model any Boolean function (布尔函数)
  - Can model any classification boundary (分类边界)
  - Can model any continuous valued function (连续值函数)

Recap: neural networks are universal function approximators 通用函数逼近器



- Neural networks are universal function approximators
  - Can model any Boolean function (布尔函数)
  - Can model any classification boundary (分类边界)
  - Can model any continuous valued function (连续值函数)
- Provided networks satisfies minimal architecture constraints
  - Networks with fewer than the required number of parameters can be very poor approximators

Recap: what are neural networks?



- Take an input
- Produce an output
- Can be modeled by a neural network!

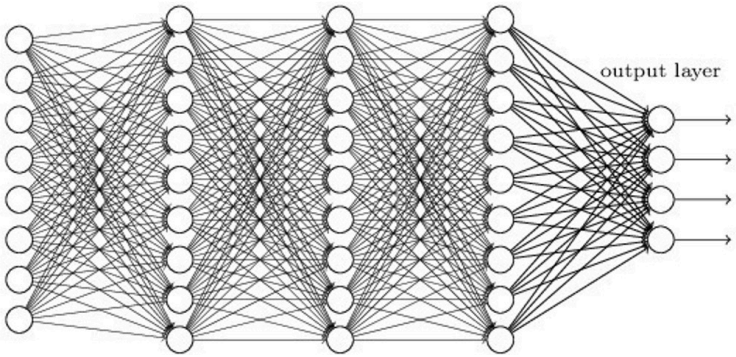
Questions

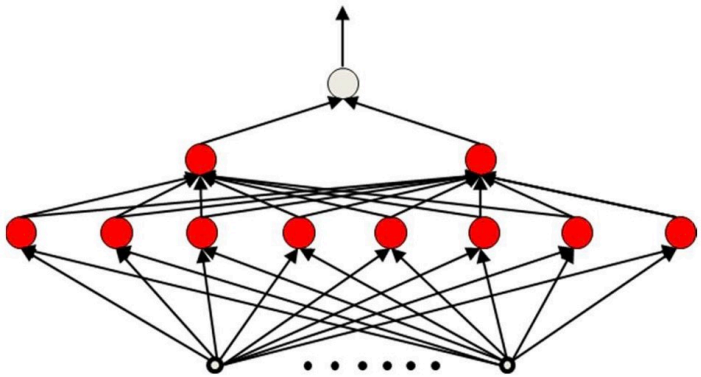
- Preliminaries:
  - How do we represent the input?
  - How do we represent the output?

Questions

- Preliminaries:
  - How do we represent the input?
  - How do we represent the output?
- How do we compose the network that performs a function?
  如何构建拟合函数的神经网络？

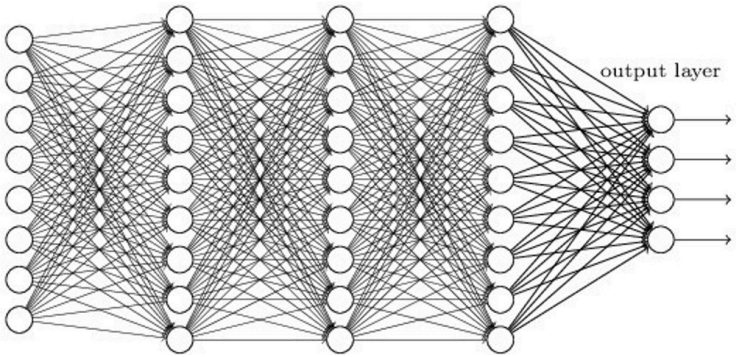
The structure of the network

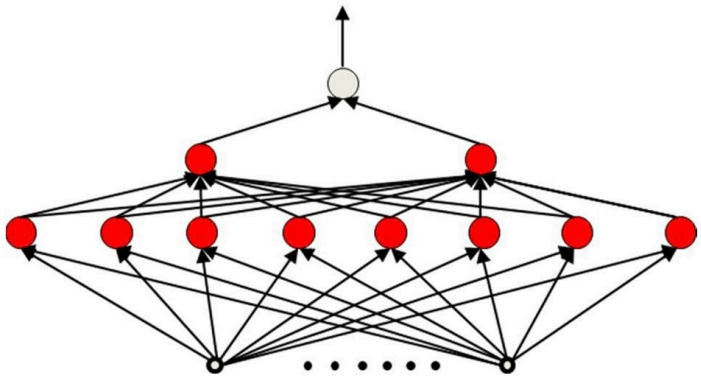




- We will assume a feed-forward network (前馈神经网络)
  - No loops (没有循环): Neuron outputs do not feed back to their inputs directly or indirectly

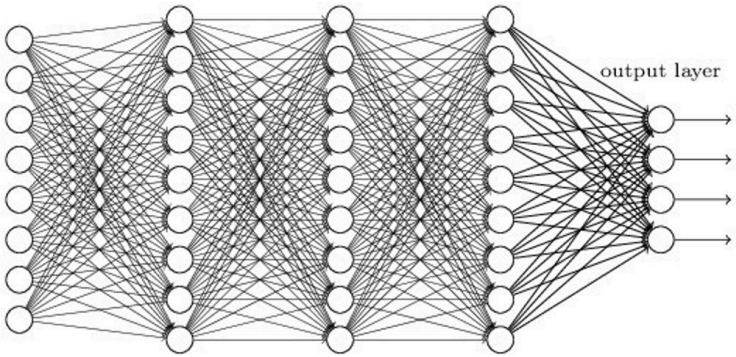
The structure of the network

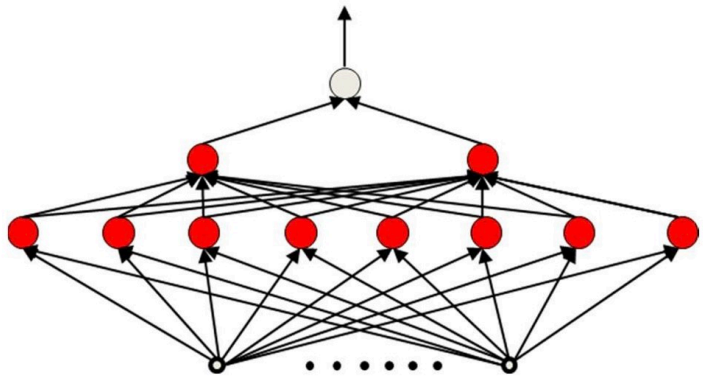




- We will assume a feed-forward network (前馈神经网络)
  - No loops (没有循环): Neuron outputs do not feed back to their inputs directly or indirectly
- **The design of a network: the architecture**
  - How many layers/neurons, which neuron connects to which and how, etc.

The structure of the network





- We will assume a feed-forward network (前馈神经网络)
  - No loops (没有循环): Neuron outputs do not feed back to their inputs directly or indirectly
- **The design of a network: the architecture**
  - How many layers/neurons, which neuron connects to which and how, etc.
- For now, assume the architecture of the network is capable of representing the needed function

What we learn: The parameters of the network

The network is a function $f(x)$ with parameters $\boldsymbol{W}$ which must be set to the appropriate values to get the desired output

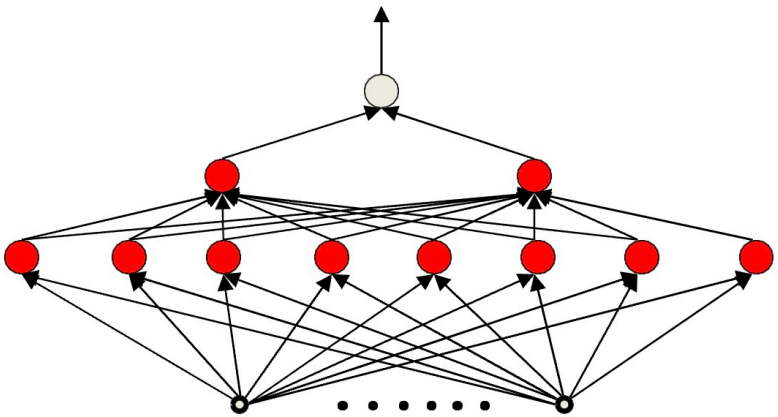
What we learn: The parameters of the network

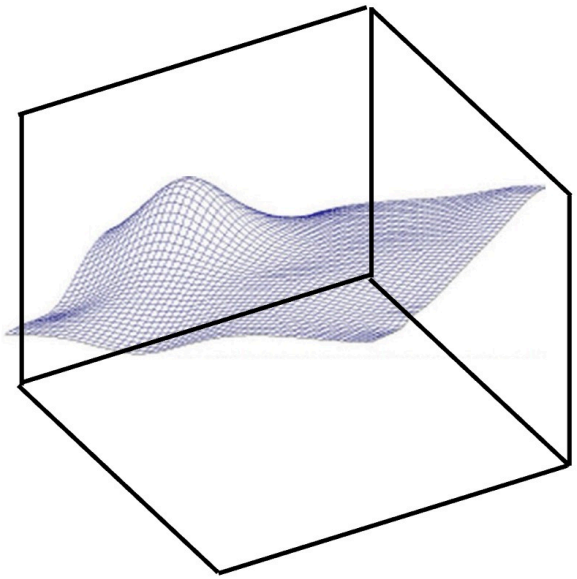
The network is a function $f(x)$ with parameters $\boldsymbol{W}$ which must be set to the appropriate values to get the desired output

- Given: the architecture of the network
- The parameters of the network: The weights and biases
- Learning: Determining the values of these parameters such that the network computes the desired function

Lecture 3.1 Neural Network Training: General

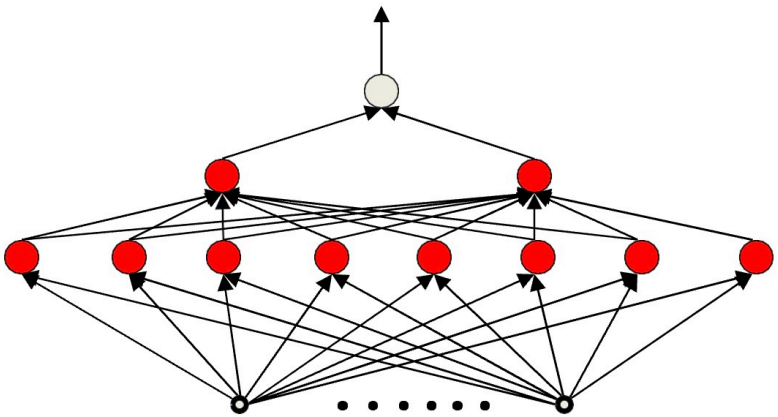
The MLP (多层感知机) can represent anything

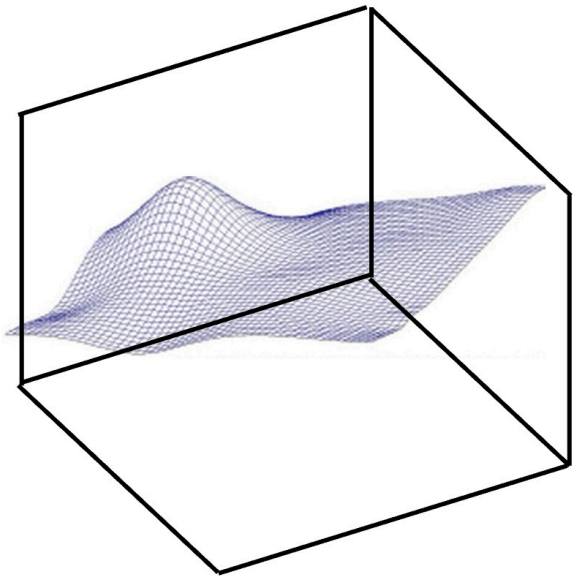




- The MLP can be constructed to represent anything
- But how do we construct it?

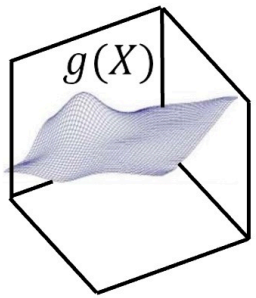
Automatic estimation of a MLP



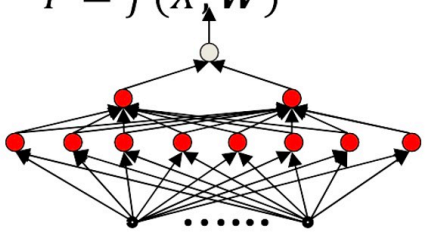


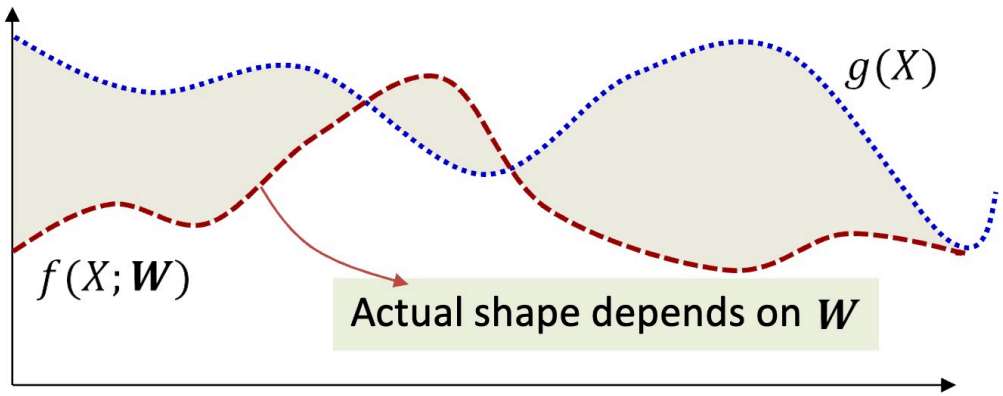
- Generally, given the function $g(X)$ to model, we can derive the parameters of the network to model it, through computation

How to learn a network?



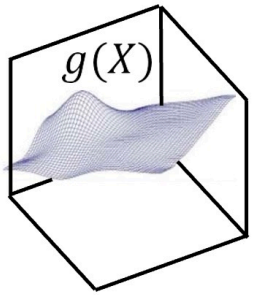
$Y = f(X; \boldsymbol{W})$

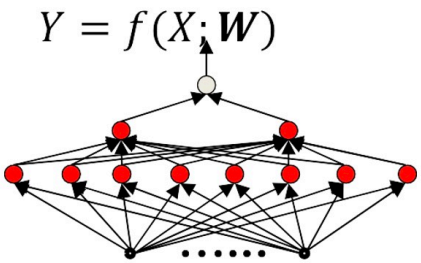


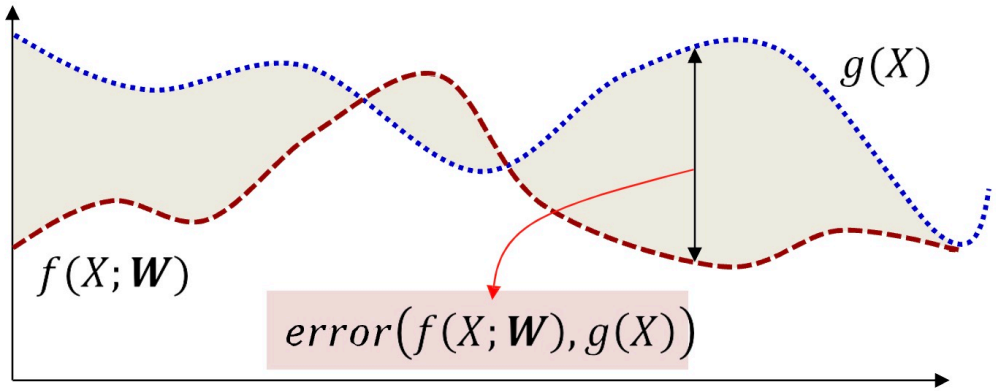


- Solution: Estimate parameters to minimize the error between the target function $g(X)$ and the network function $f(X, \mathbf{W})$
  - Find the parameter $\mathbf{W}$ that minimizes the shaded area

How to learn a network?



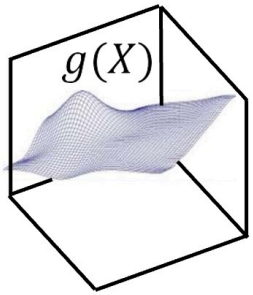


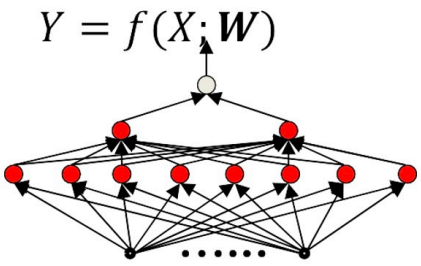


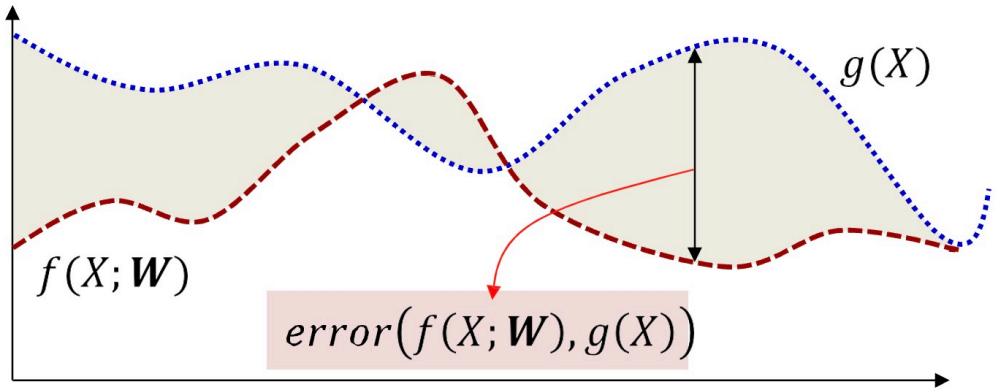
- The shaded area

$$totalerr(\boldsymbol{W}) = \int_{-\infty}^{\infty} error\big(f(X; \boldsymbol{W}), g(X)\big) dX$$

How to learn a network?







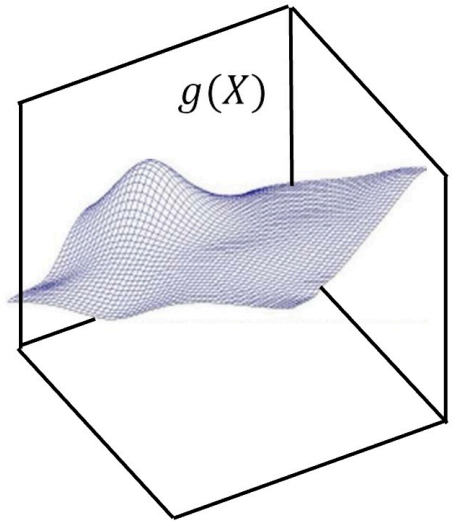
- The shaded area

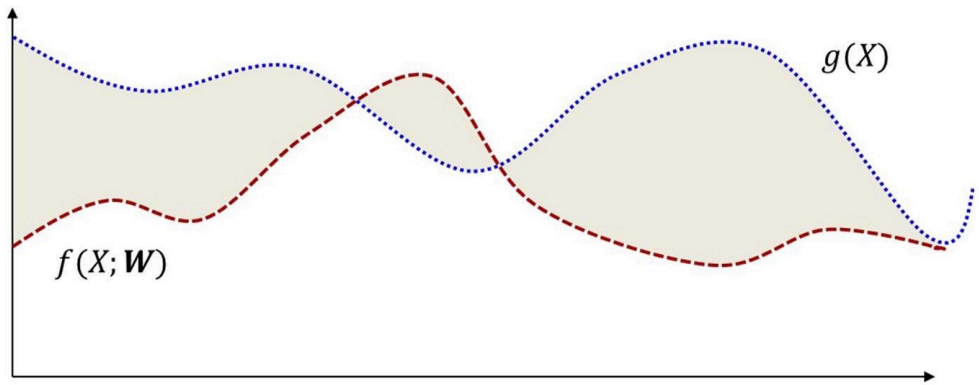
$$totalerr(\boldsymbol{W}) = \int_{-\infty}^{\infty} error\big(f(X; \boldsymbol{W}), g(X)\big) dX$$

- The optimal $\mathbf{W}$

$$\widehat{\boldsymbol{W}} = \underset{\boldsymbol{W}}{\operatorname{argmin}}\ totalerr(\boldsymbol{W})$$

Problem: $g(X)$ is unknown

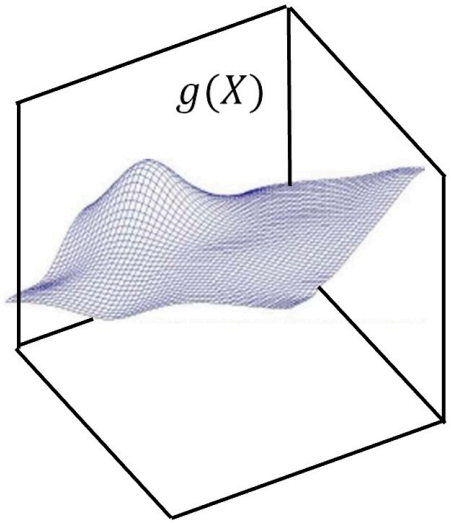


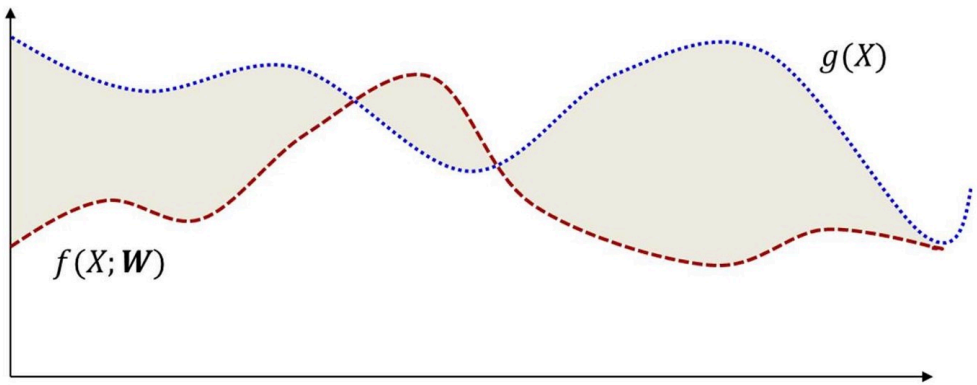


- Function $g(X)$ must be fully specified in order to compute

$$\int_{-\infty}^{\infty} error\big(f(X;\boldsymbol{W}), g(X)\big)dX$$

Problem: $g(X)$ is unknown



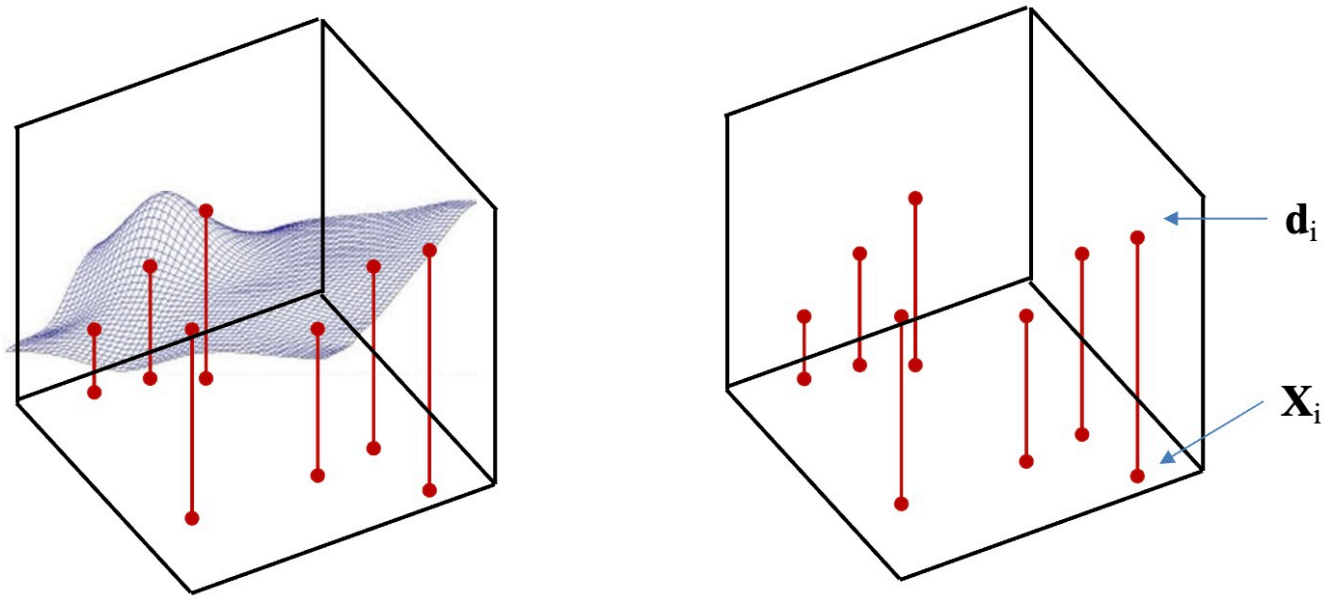


- Function $g(X)$ must be fully specified in order to compute

$$\int_{-\infty}^{\infty} error\big(f(X;\boldsymbol{W}), g(X)\big)dX$$

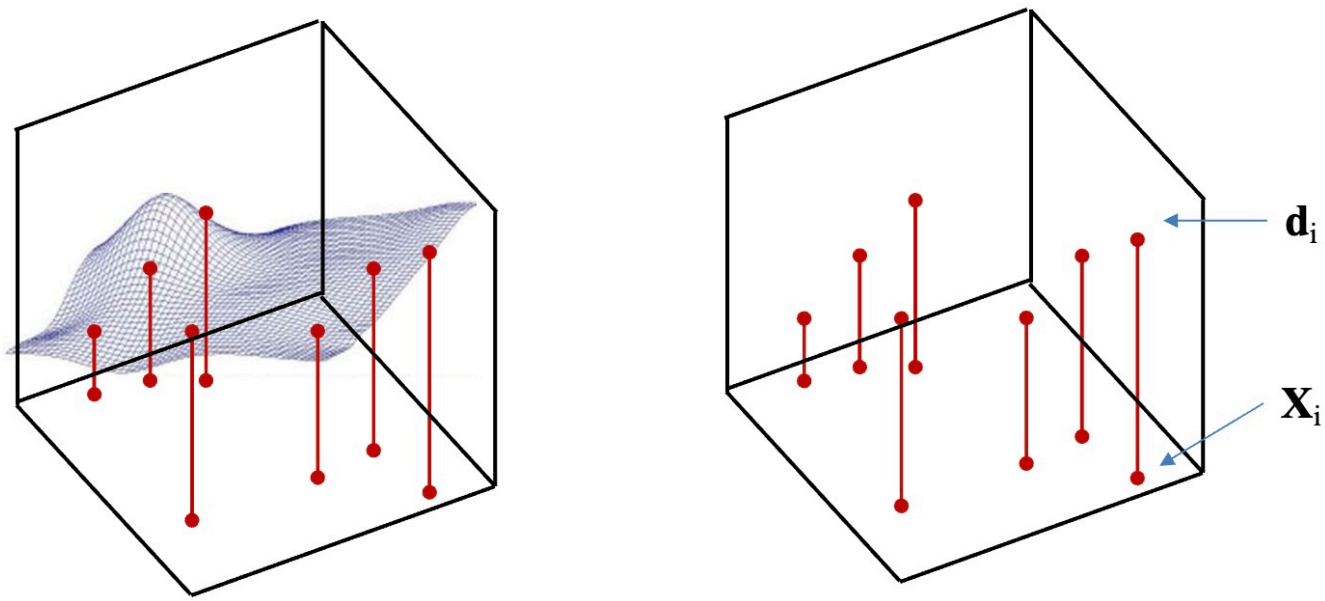
- **In practice we will not have such specification**

Sampling the function



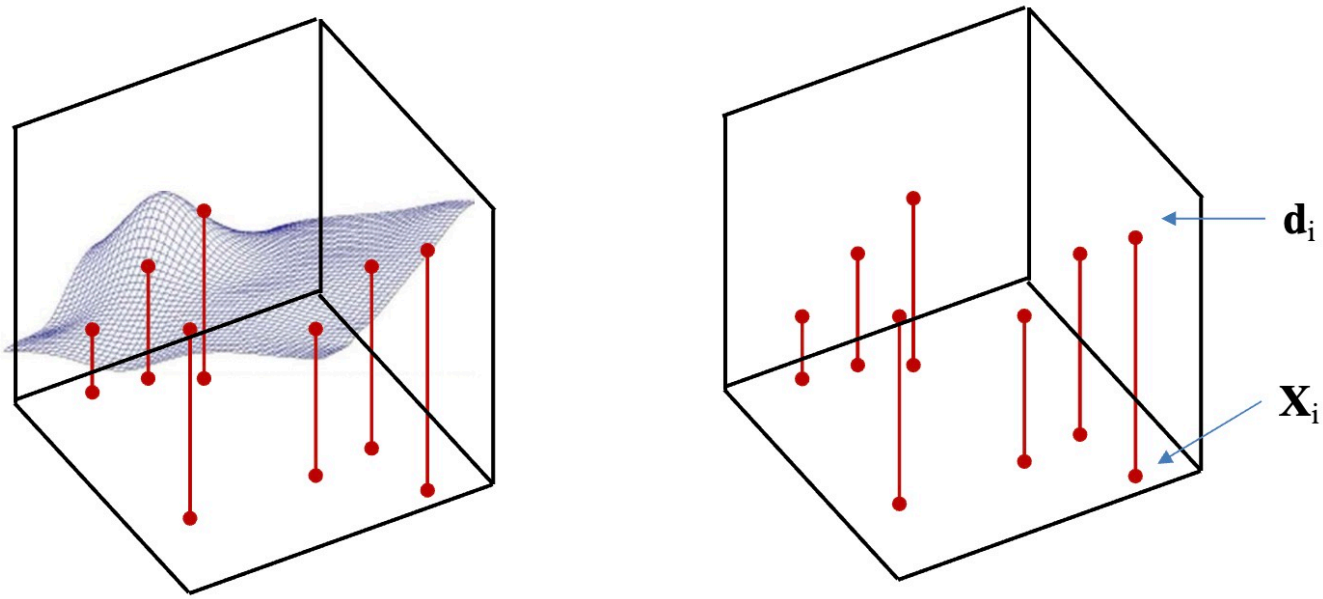
- Sample $g(X)$
  - Get input-output pairs for a number of samples of X_i
    - Many samples (X_i, d_i), where $d_i = g(X_i) + noise$

Sampling the function



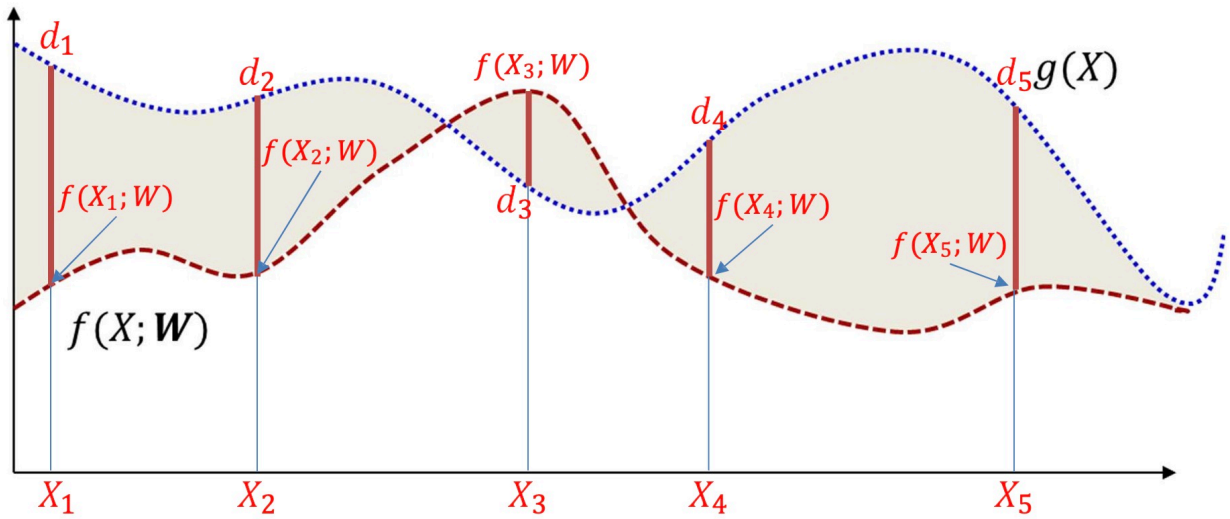
- Sample $g(X)$
  - Get input-output pairs for a number of samples of X_i
    - Many samples (X_i, d_i), where $d_i = g(X_i) + noise$
- Very easy to do in most problems: just gather training data
  - E.g. set of images and their class labels
  - E.g. speech recordings and their transcription (转录)

Sampling the function



- We must learn the entire function from these few examples
  - The "training" samples

The empirical error



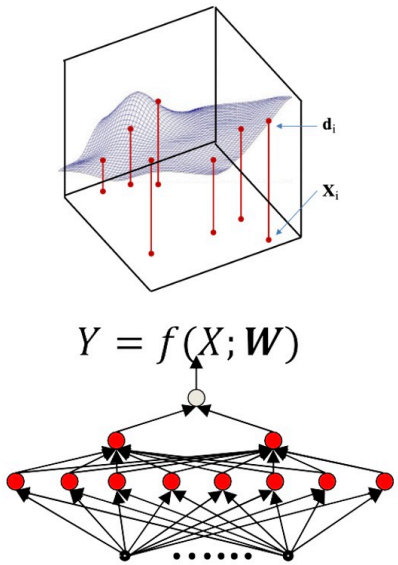
- The empirical estimate of the error (误差经验估计) is the average error over the training samples

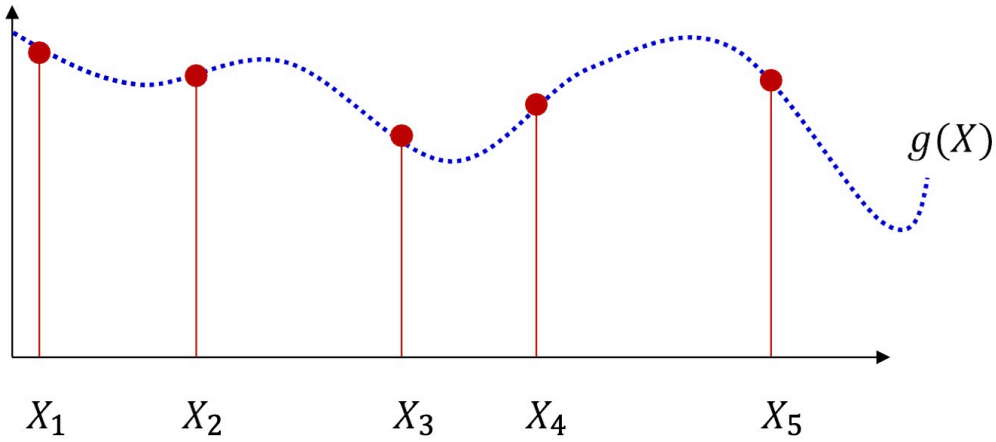
$$EmpiricalError(\boldsymbol{W}) = \frac{1}{N}\sum_{\mathrm{i}=1}^{\mathrm{N}} error(f(X_{\mathrm{i}};\boldsymbol{W}), d_{\mathrm{i}})$$

- Estimate network parameters to minimize this average error

$$\widehat{\boldsymbol{W}} = \underset{\mathrm{W}}{\operatorname{argmin}}\, EmpiricalError(\boldsymbol{W})$$

Learning the function from training samples





- Aim: Find network parameters that fit training points exactly

$$\mathbf{W} : EmpiricalError(\mathbf{W}) = 0$$

- And hopefully the resulting function is also correct where we don't have training samples

Summary

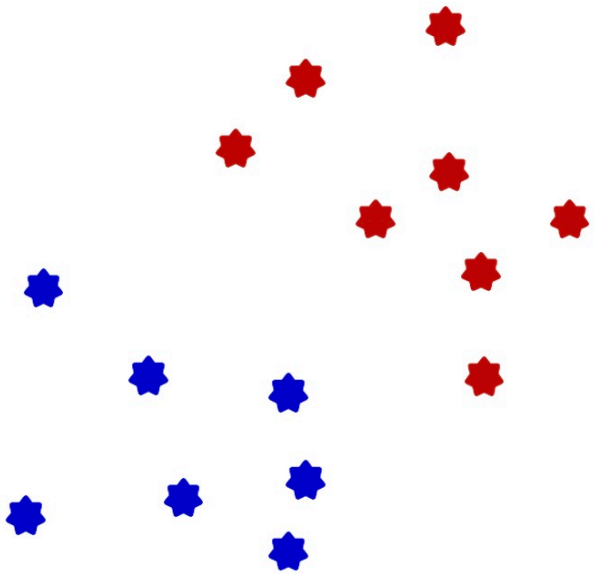
- "Learning" a neural network = determining the parameters of the network (weights and biases) required to model a function
- Ideally, we would like to optimize the network to represent the desired function
- However, this requires knowledge of the function
- Instead, we draw "input-output" training instances from the function and estimate network parameters to "fit" the input-output relation at these instances
  - And hope it fits the function as well

Lecture 3.2 An Example

An example

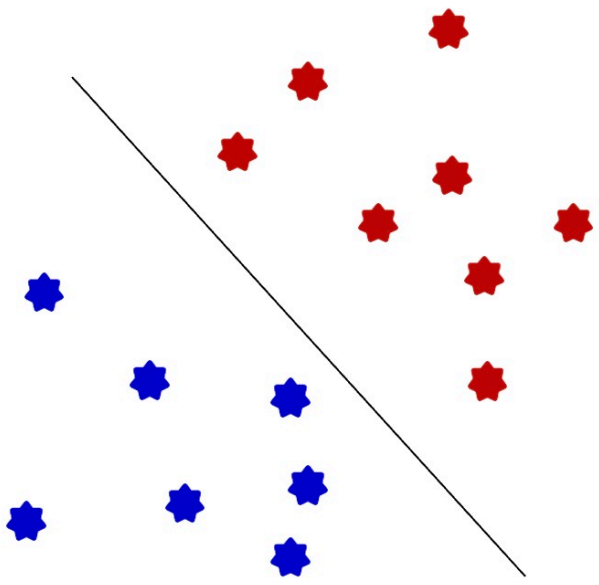
- Learning a classifier
  - Simpler than regressions
- This was among the earliest problems addressed using MLPs
- Specifically, consider binary classification
  - Generalizes to multi-class

An example



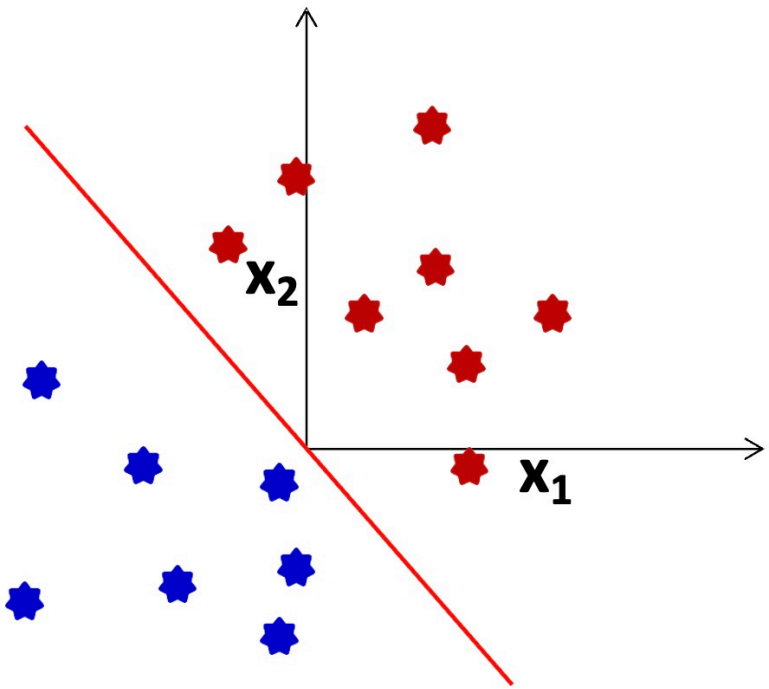
- Task: Learn a function (network) to clasify the set of samples

An example



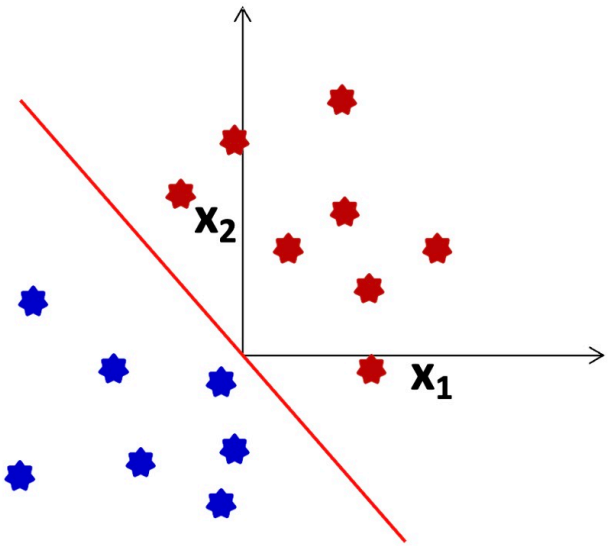
- Task: Learn a function (network) to clasify the set of samples
  - Find a hyperplane (超平面) in the space

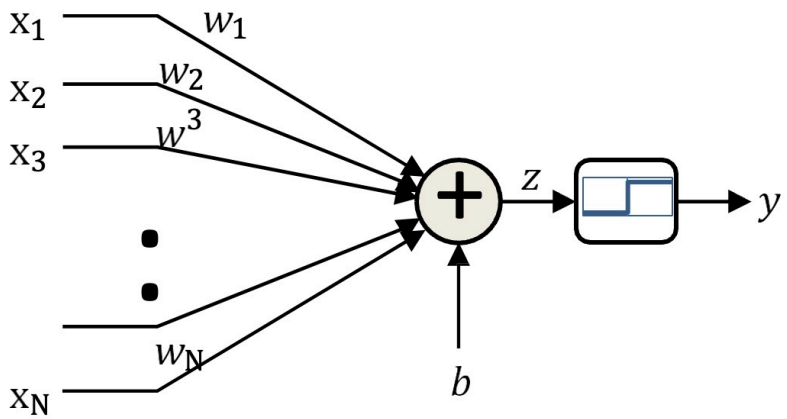
An example



- Task: find a hyperplane (超平面) in the space

Learning the perceptron 感知机

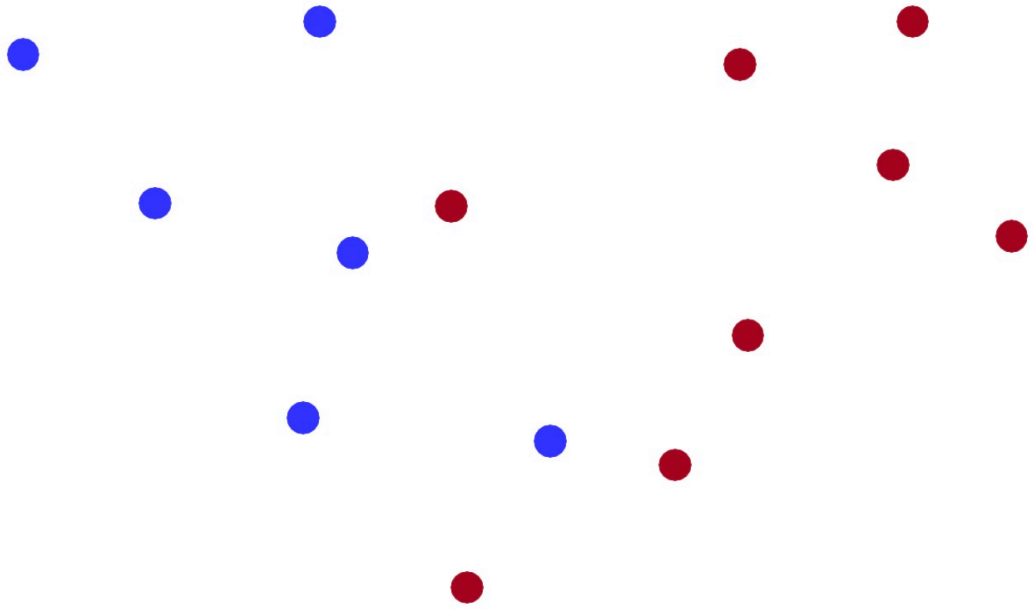




- Given several input-output pairs, learn the weights and bias
  - $y = \begin{cases} 1 & if \ \sum_{i=1}^{N} w_i X_i + b \geq 0 \\ 0 & otherwise \end{cases}$

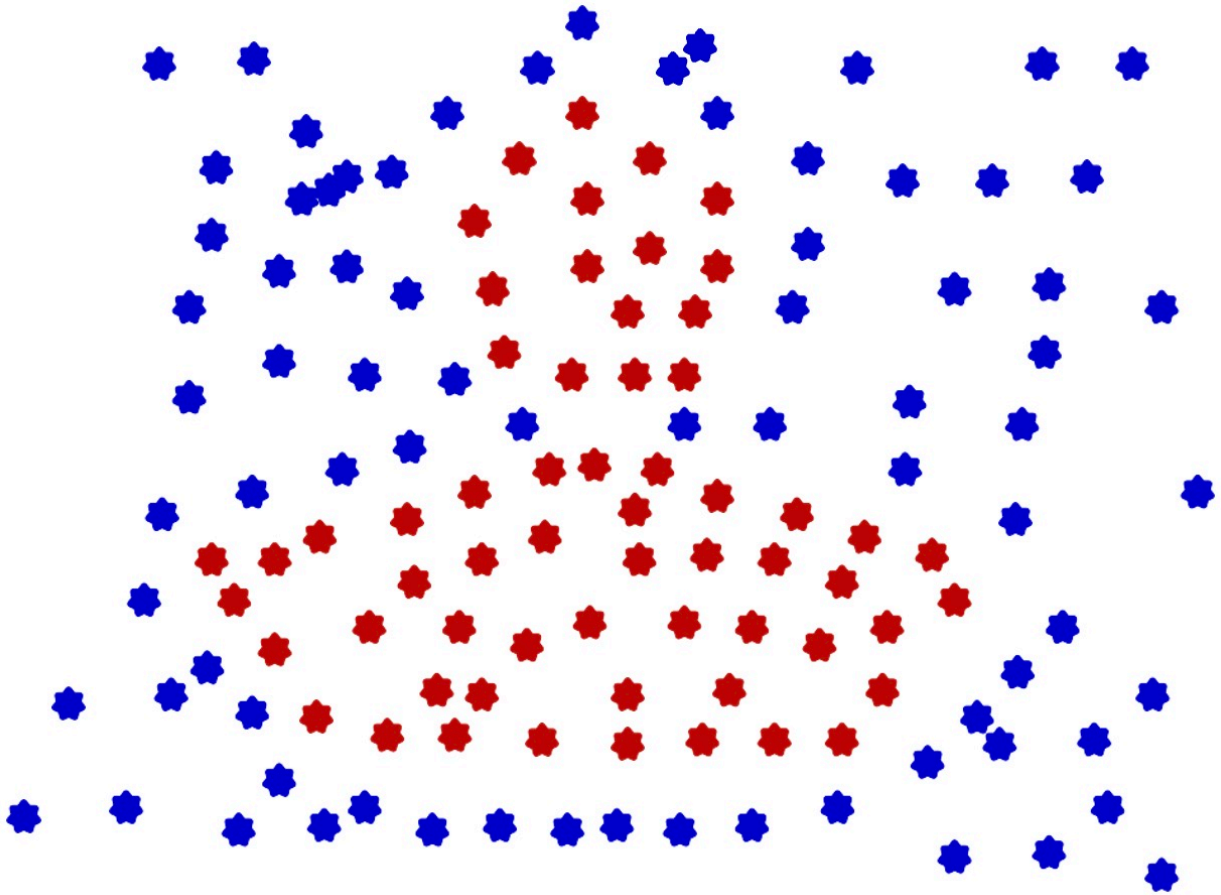
    Boundary: $\sum_{i=1}^{N} w_i X_i + b = 0$
  - Learn $W = [w_1 .. w_N]^T$ and b, given several (X, y) pairs

When classes are not linearly separable

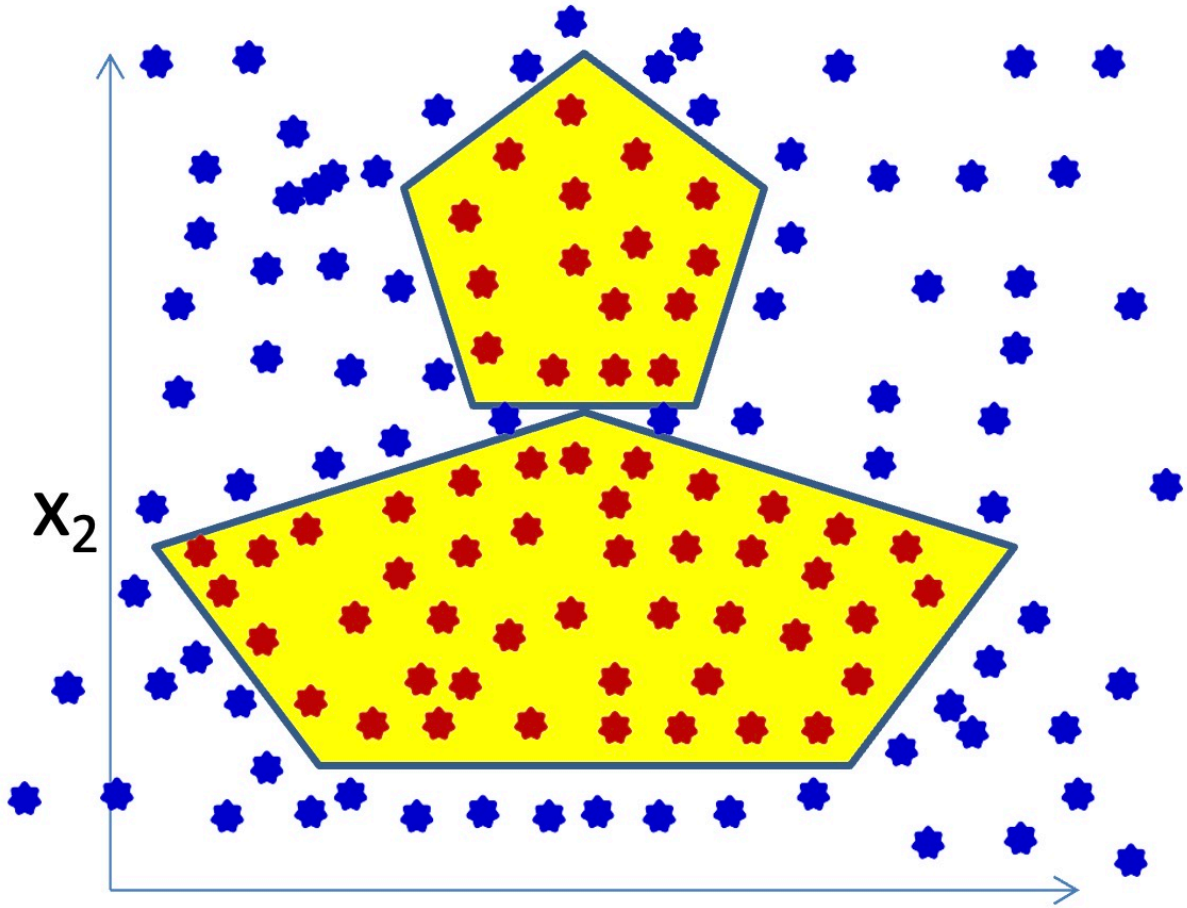


- When classes are not linearly separable, not possible to find a separating hyperplane
  - No "support" plane for reflected data
  - Some points will always lie on the other side
- Model does not support perfect classification of this data

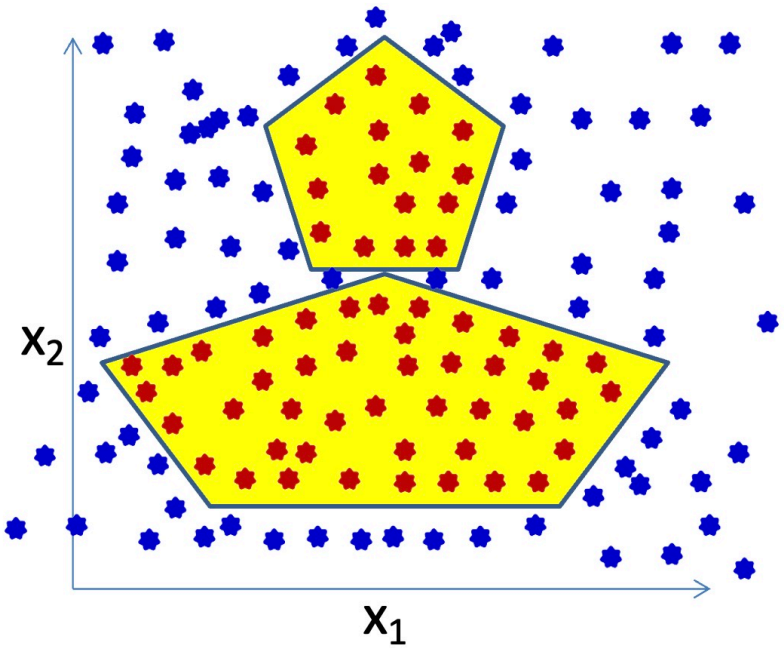
A more complex problem

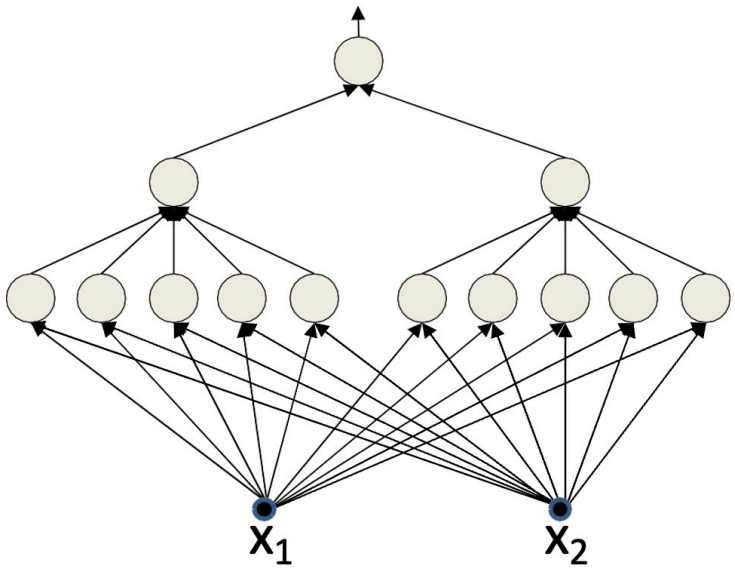


A more complex problem (cont'd)



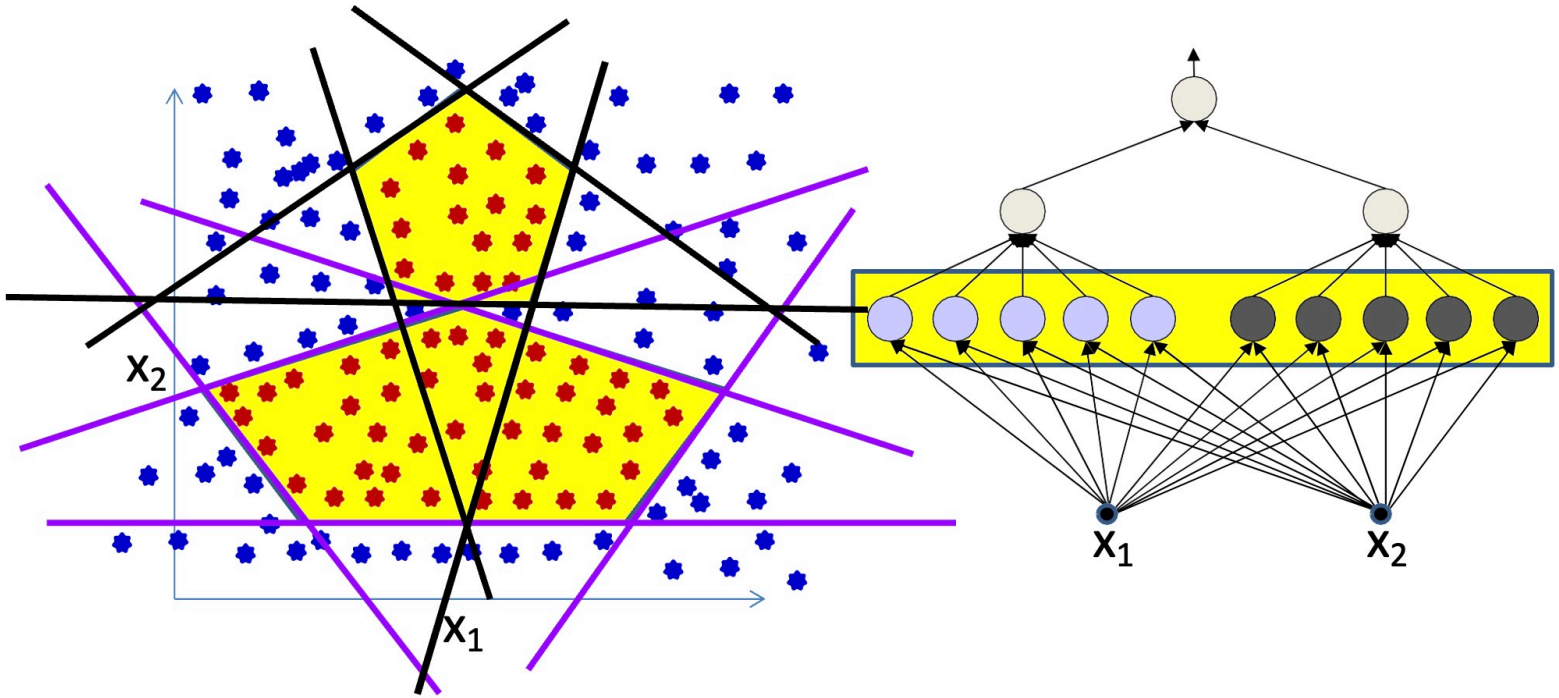
A more complex problem (cont'd)





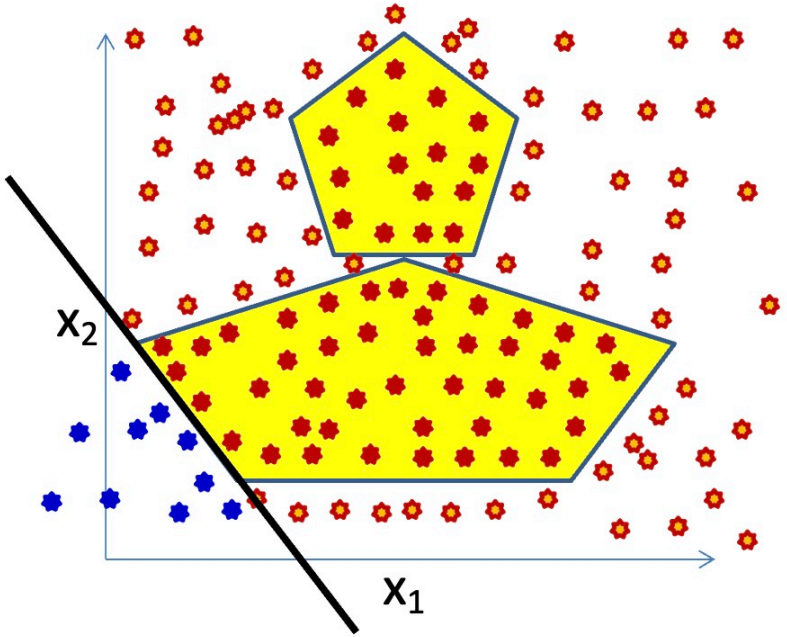
- Learn an MLP for this function
  - 1 in the yellow regions, 0 outside
- Using just the samples
- This can be perfectly represented using an MLP (多层感知机)

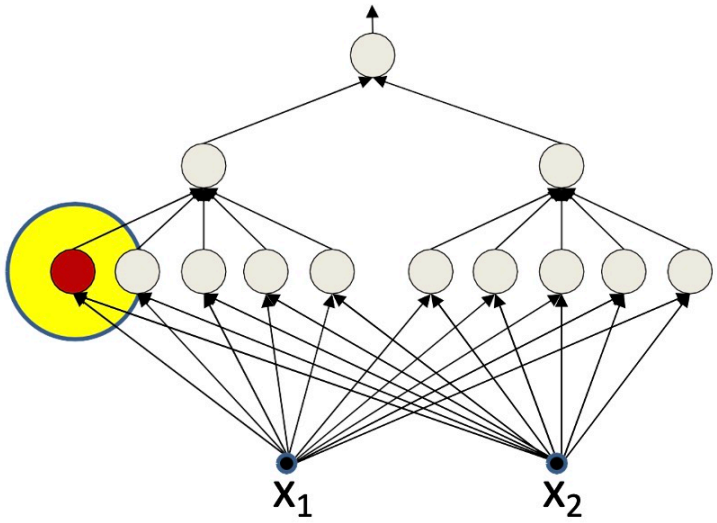
The pattern to be learned at the lower level



- The lower-level neurons are linear classifiers
  - They require linearly separated labels to be learned
  - The actually provided labels are not linearly separated
  - Challenge: Must also learn the labels for the lowest units!

The pattern to be learned at the lower level





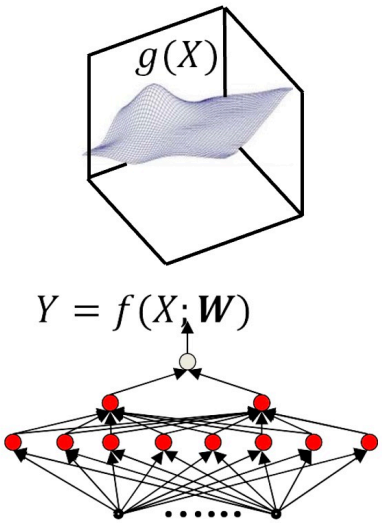
- The lower-level neurons are linear classifiers
  - They require linearly separated labels to be learned
- The individual classifier requires the kind of labelling shown here, which is not given!

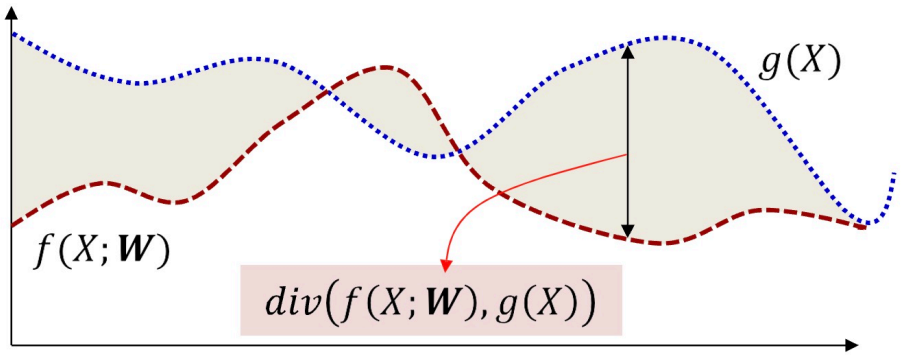
Summary

- "Learning" a network = learning the weights and biases to compute a target function
  - Will require a network with sufficient "capacity"
- In practice, we learn networks by "fitting" them to match the input-output relation of "training" instances drawn from the target function
- A linear decision boundary can be learned by a single perceptron if classes are linearly separable
- Non-linear decision boundaries require networks of perceptrons

Lecture 3.3 How to Train a Network?

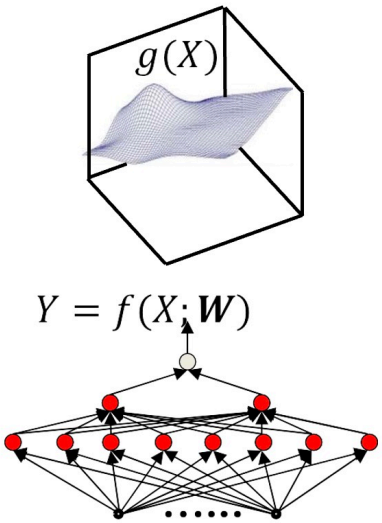
How to learn a network?

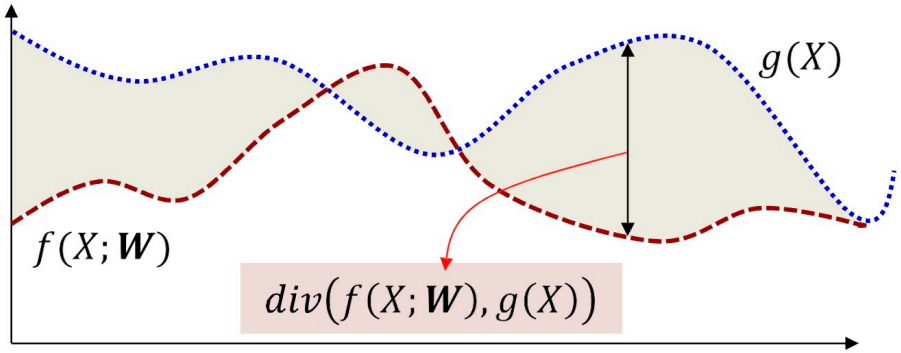




- Define a divergence function (差异函数) $div(f(X; \mathbf{W}), g(X))$ with the following properties
  - $div(f(X; \mathbf{W}), g(X)) = 0$ if $f(X; \mathbf{W}) = g(X)$
  - $div(f(X; \mathbf{W}), g(X)) > 0$ if $f(X; \mathbf{W}) \neq g(X)$
  - $div(f, g)$ is differentiable (可微) with respect to f

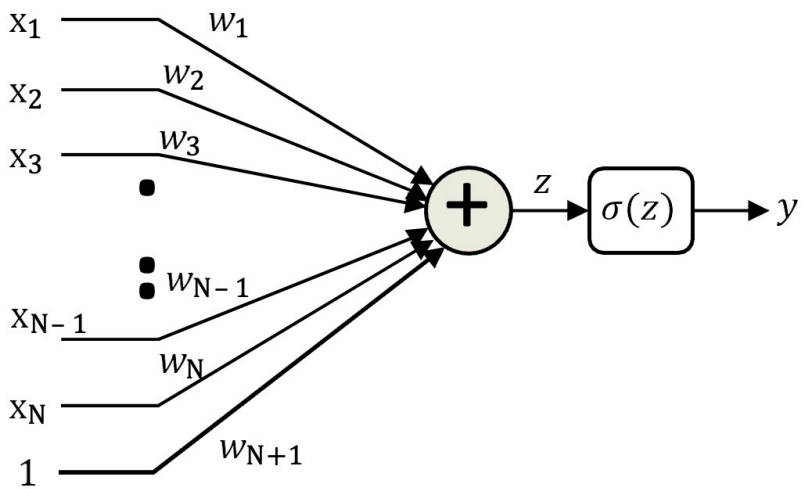
How to learn a network?

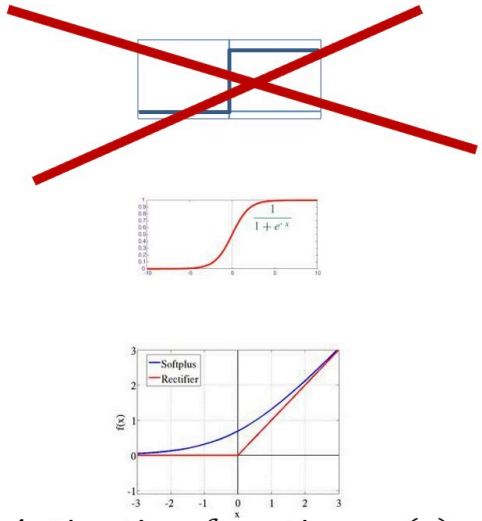




- Define a divergence function (差异函数) $div(f(X; \mathbf{W}), g(X))$ with the following properties
  - $div(f(X; \mathbf{W}), g(X)) = 0$ if $f(X; \mathbf{W}) = g(X)$
  - $div(f(X; \mathbf{W}), g(X)) > 0$ if $f(X; \mathbf{W}) \neq g(X)$
  - $div(f, g)$ is differentiable (可微) with respect to f
- The divergence function quantifies mismatch between the network output and target function
  - For classification, this is usually not the classification error but an approximation to it

Continuous Activations

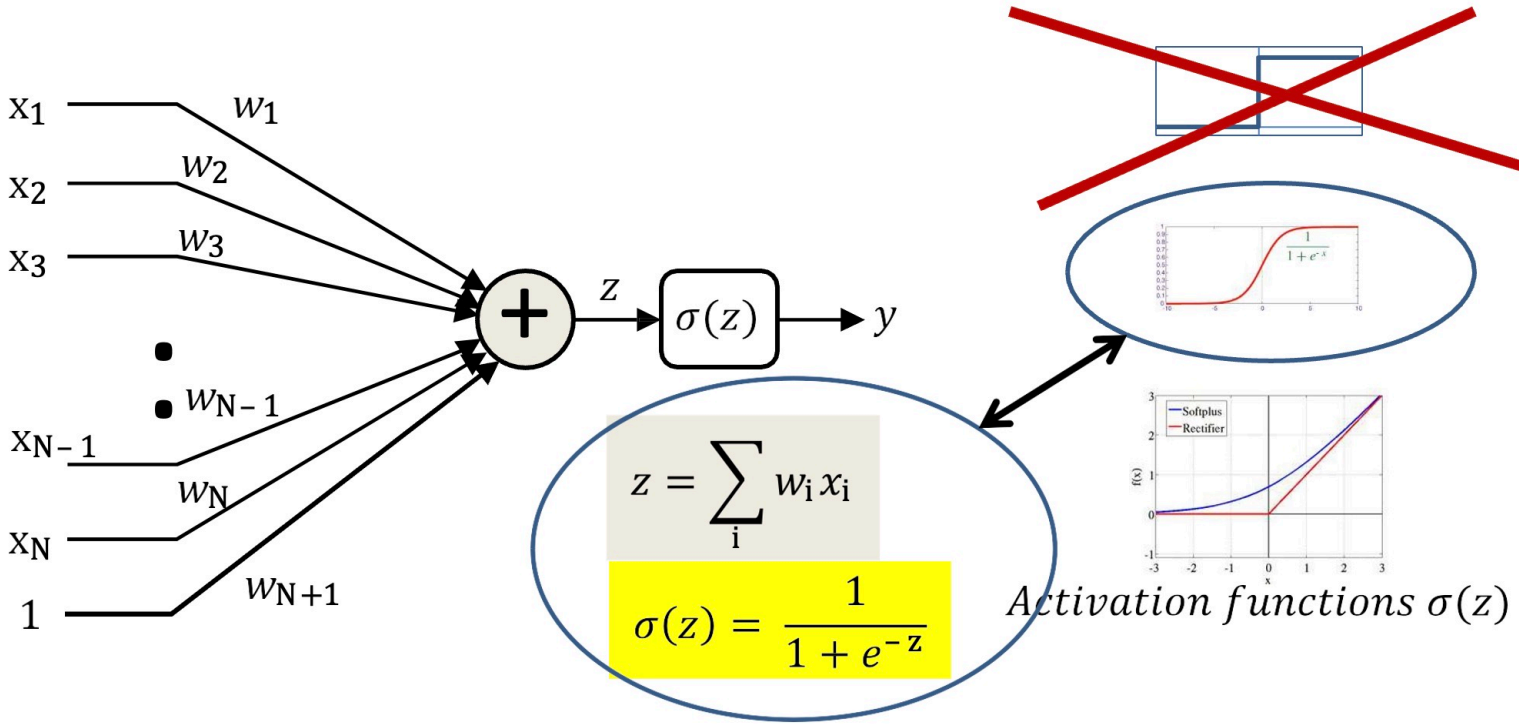




Activation functions $\sigma(z)$

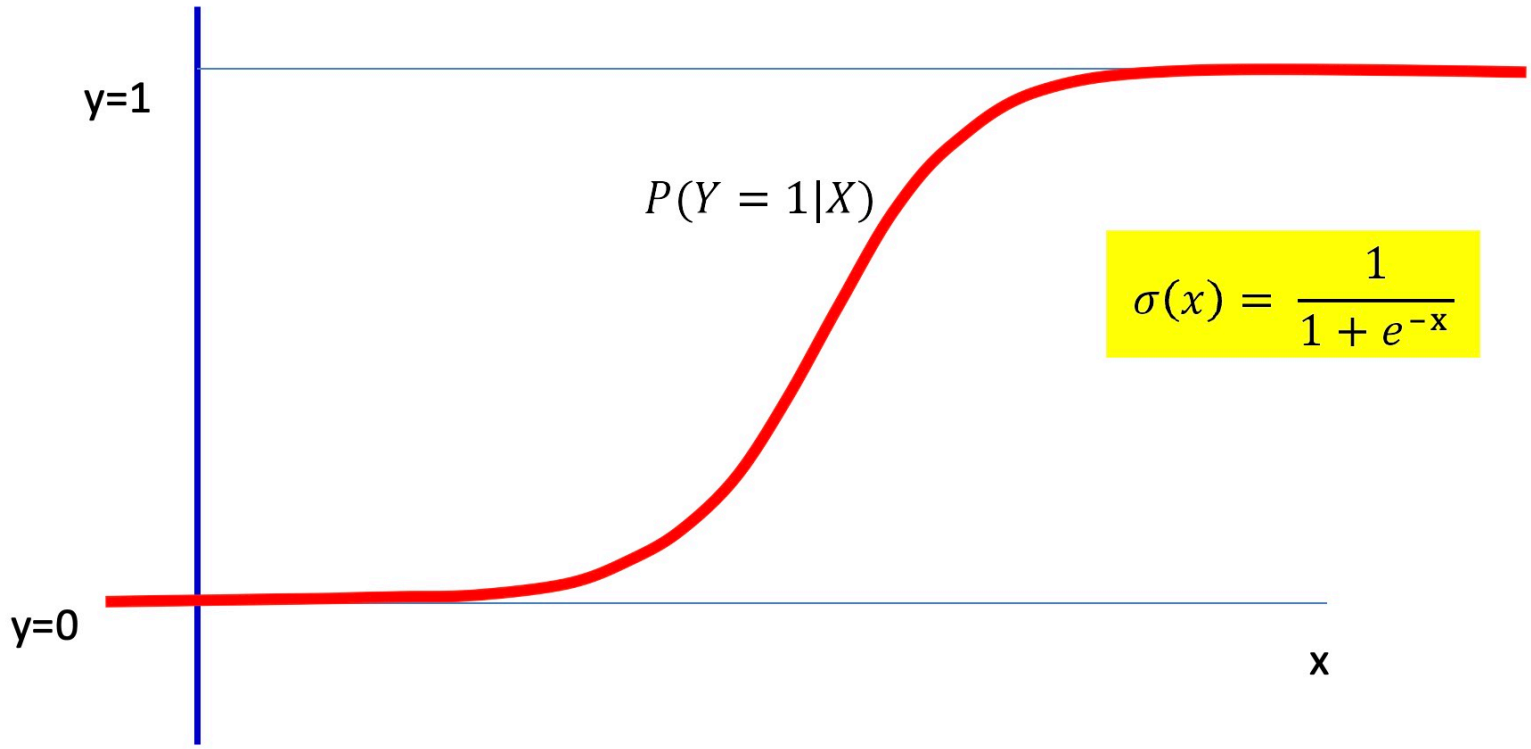
- Replace the threshold activation with continuous activations
  - E.g. RELU, softplus, sigmoid etc.
- The activations are differentiable almost everywhere
  - Have derivatives almost everywhere
  - And have "subderivatives" at non-differentiable corners

The sigmoid activation is special



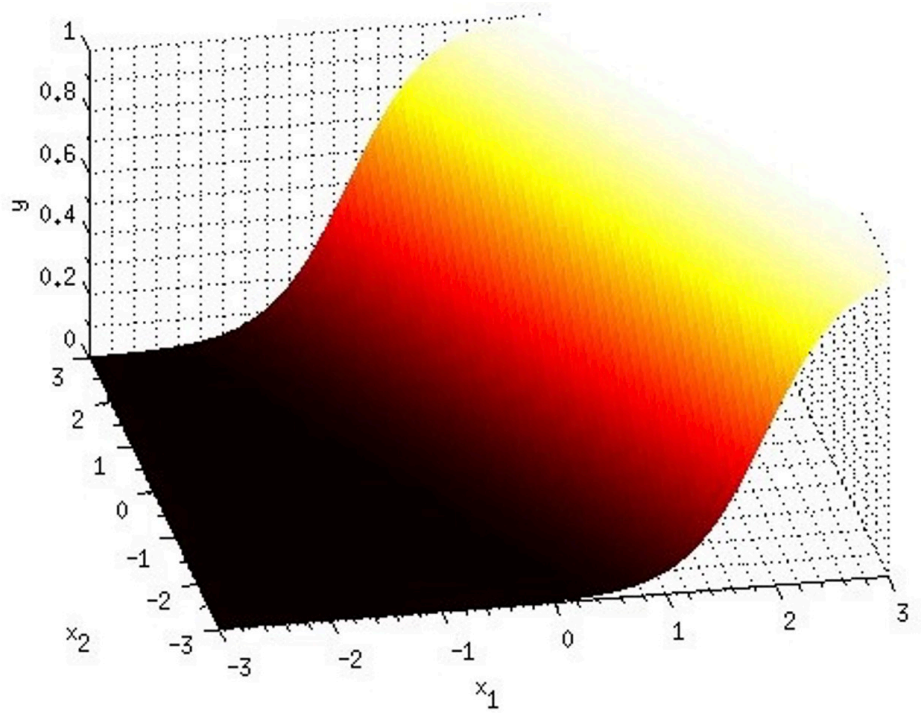
- This particular one has a nice interpretation
  - It can be interpreted as $P(y = 1|x)$

The logistic regression model



- Class 1 becomes increasingly probable going left to right
  - Very typical in many problems

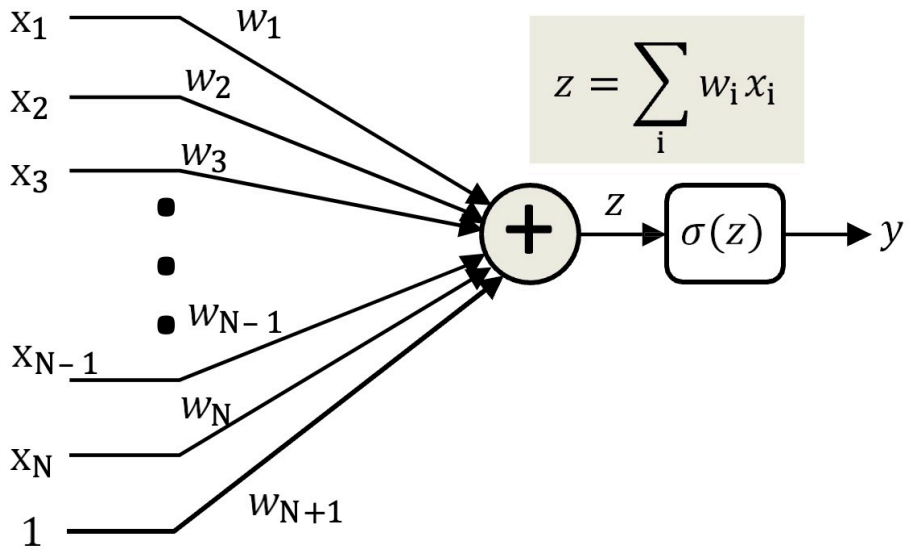
The logistic regression model



When X is a 2-D variable

- This the perceptron with a sigmoid activation
  - It computes the probability that the input belongs to class 1

Perceptrons with differentiable activation functions



$$z = \sum_i w_i x_i$$

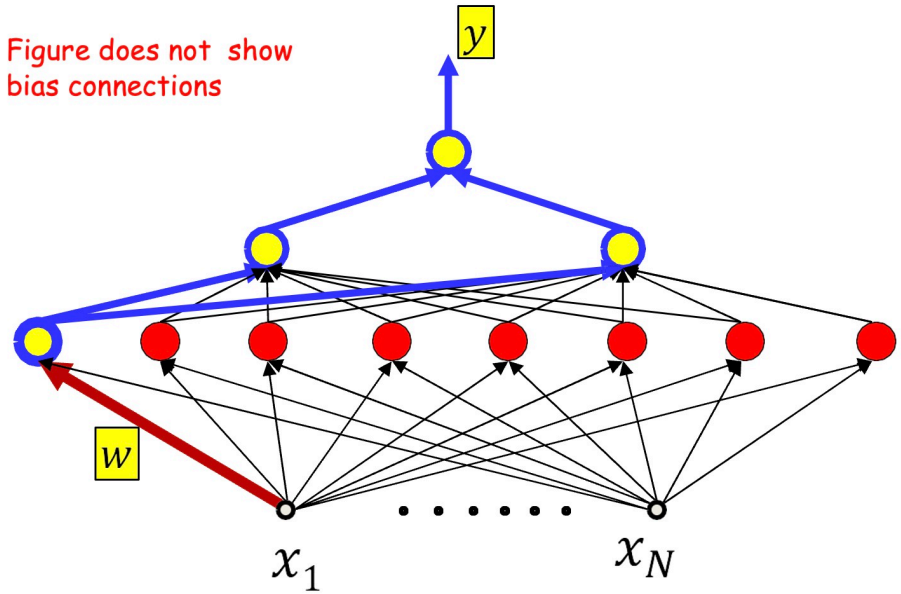
$$\frac{dy}{dz} = \sigma'(z)$$

$$\frac{dy}{dw_i} = \frac{dy}{dz}\frac{dz}{dw_i} = \sigma'(z)x_i$$

$$\frac{dy}{dx_i} = \frac{dy}{dz}\frac{dz}{dx_i} = \sigma'(z)w_i$$

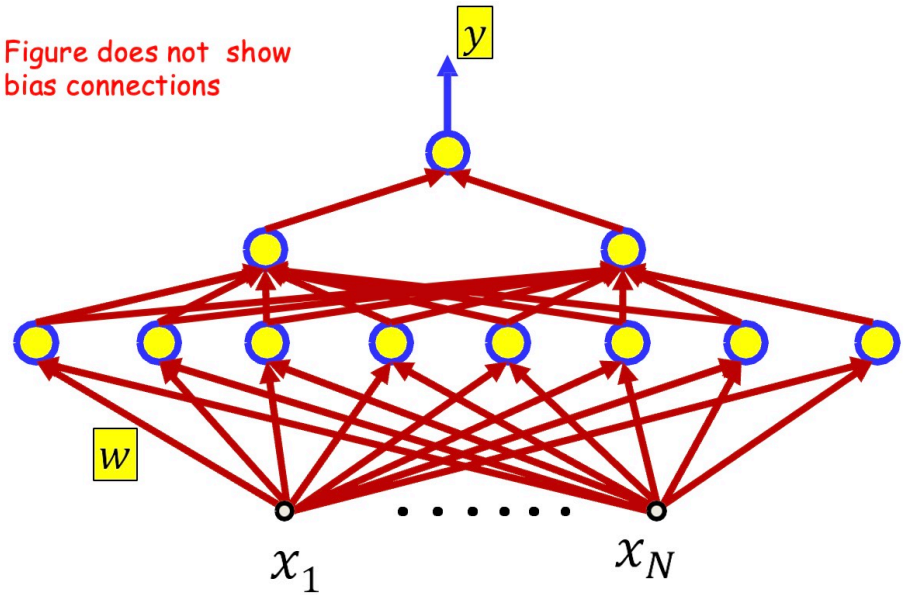
- $\sigma(z)$ is a differentiable function of z
- Using the chain rule (链式法则), y is a differentiable function of both inputs x_i and weights w_i
- This means that we can compute the change in the output for small changes in either the input or the weights

Overall network is differentiable



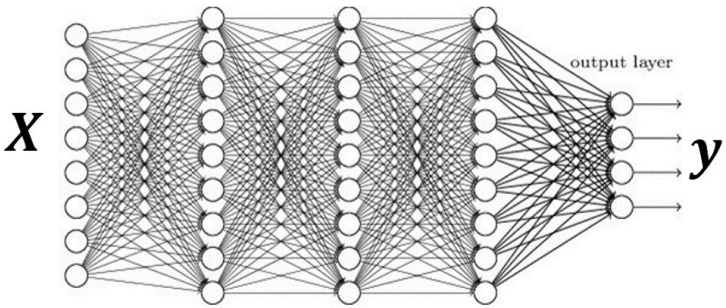
- Every individual perceptron is differentiable w.r.t its inputs and its weights (including "bias" weight)
- Using the chain rule can compute how small perturbations of a parameter change the output of the network
  - The network output is differentiable w.r.t the parameter

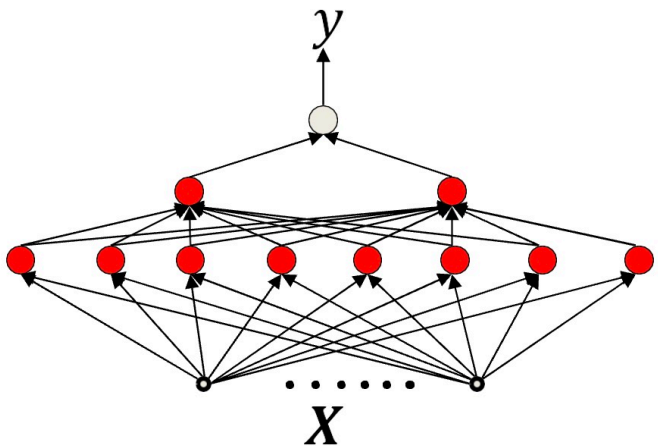
Overall function is differentiable



- By extension, the overall function is differentiable w.r.t every parameter in the network
  - Can compute how small changes in parameters change output
- We will derive the actual derivatives using the chain rule later

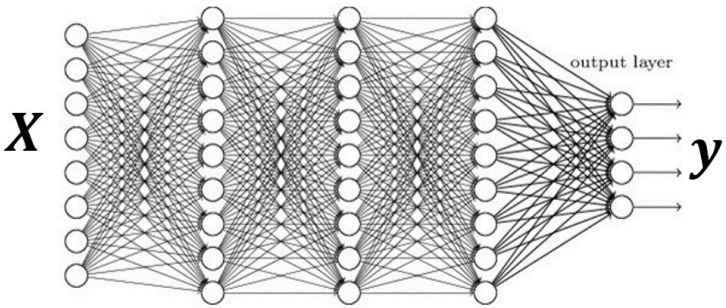
Overall setting for "Learning" the MLP

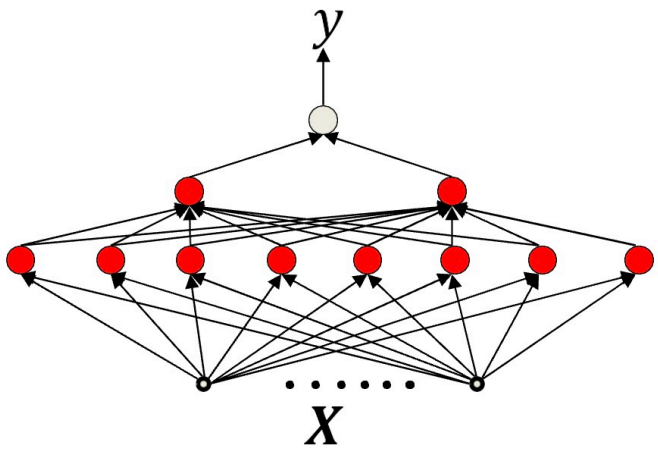




- Given a training set of input-output pairs (X_1, d_1), (X_2, d_d), ..., (X_N, d_N)
  - d is the desired output of the network in response to X
  - X and d may both be vectors

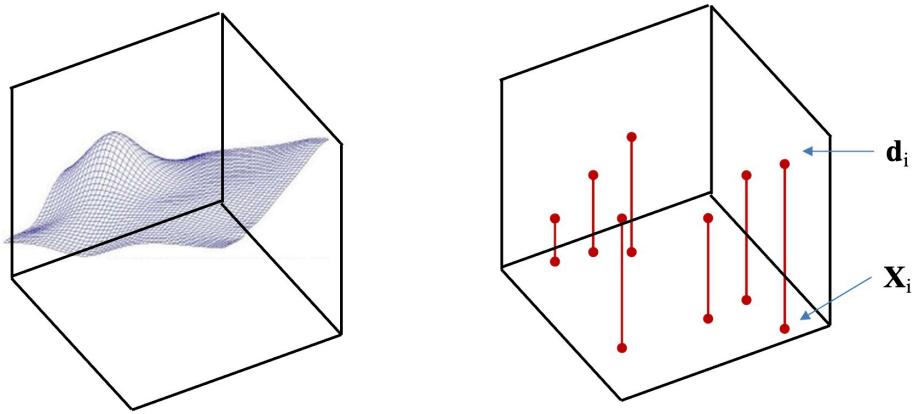
Overall setting for "Learning" the MLP





- Given a training set of input-output pairs (X_1, d_1), (X_2, d_d), ..., (X_N, d_N)
  - d is the desired output of the network in response to X
  - X and d may both be vectors
- We must find the network parameters such that the network produces the desired output for each training input
  - Or a close approximation of it
  - **The architecture of the network must be specified by us**

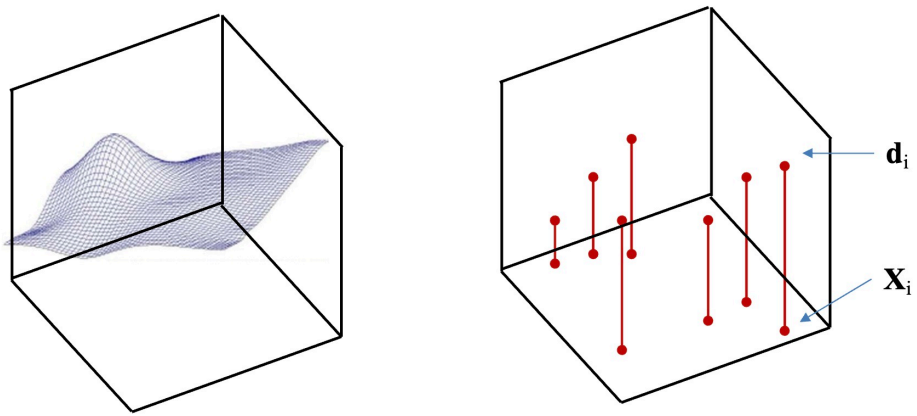
The Empirical risk



- The expected divergence (期望误差) is the average divergence over the entire input space

$$E\big[div\big(f(X;W),g(X)\big)\big] = \int_{\mathrm{K}} div\big(f(X;W),g(X)\big)P(X)dX$$

The Empirical risk



- The expected divergence (期望误差) is the average divergence over the entire input space

$$E\big[div\big(f(X;W),g(X)\big)\big] = \int_{\mathrm{K}} div\big(f(X;W),g(X)\big)P(X)dX$$

- The empirical estimate of the expected risk (误差经验估计) is the average divergence over the samples

$$E\big[div\big(f(X;W),g(X)\big)\big] \approx \frac{1}{N}\sum_{\mathrm{i}=1}^{\mathrm{N}} div(f(X_{\mathrm{i}};W),d_{\mathrm{i}})$$

Empirical risk minimization

- Given training samples (X_1, d_1), (X_2, d_d), ..., (X_N, d_N)
  - Quantification of error on the i-th instance: $div(f(X_i; \mathbf{W}), d_i)$
  - Empirical average divergence on all training data:

$$Loss(\mathbf{W}) = \frac{1}{N} \sum_i div(f(X_i; \mathbf{W}), d_i)$$

  - Estimate the parameters to minimize the empiricial risk over the drawn samples

$$\hat{\mathbf{W}} = \text{argmin}_W Loss(\mathbf{W})$$

Empirical risk minimization

Note 1: It's really a measure of error, but using standard terminology, we will call it a "Loss" (损失)

Note 2: The empirical risk $Loss(\mathbf{W})$ is only an empirical approximation to the true risk $E[div(f(X;\mathbf{W}), g(X))]$, which is our **actual minimization objective**.

Note 3: For a given training set, the loss is only a function of $\mathbf{W}$

Problem Statement

- Given training samples (X_1, d_1), (X_2, d_d), ..., (X_N, d_N)
- Minimize the following function

$$Loss(\mathbf{W}) = \frac{1}{N}\sum_i div(f(X_i; \mathbf{W}), d_i)$$

  w.r.t $\mathbf{W}$
- This is problem of function minimization

$$\hat{\mathbf{W}} = \text{argmin}_W Loss(\mathbf{W})$$

In-class quiz

- 根据通用近似定理，神经网络可以拟合任意函数，但在实践中会遇到什么困难使得这项理论难以实现？
- 什么是期望误差 (expected divergence) 和期望误差经验估计 (empirical estimate of the expected risk)？
- 什么是损失函数（loss function）？
- 采样训练样本进行神经网络训练的理论依据是什么？

Thank you!