# 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

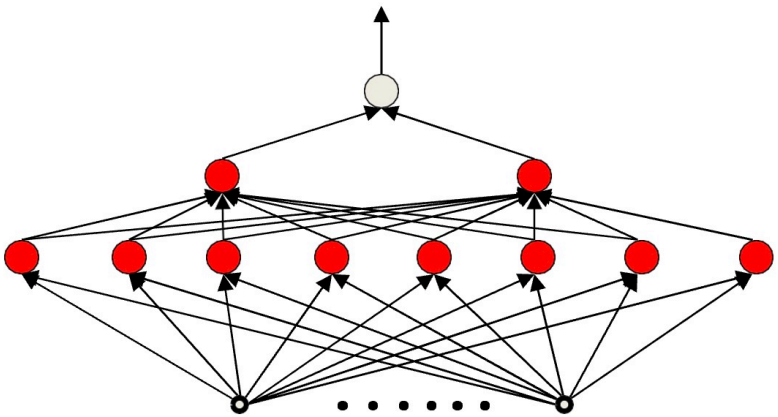
2023 年 3 月 21 日

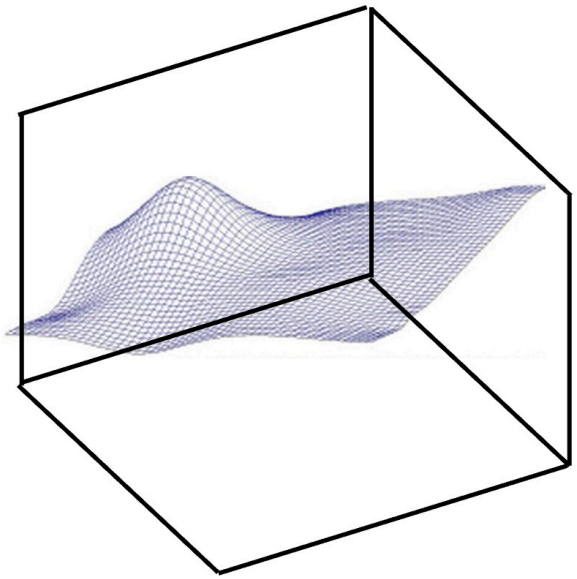
# Lecture 4 Neural Networks Training II: Back Propogation

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

# Recap

# Empirical risk minimization





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

# 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 (经验平均误差) on training data:

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

# 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 (经验平均误差) on training data:

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

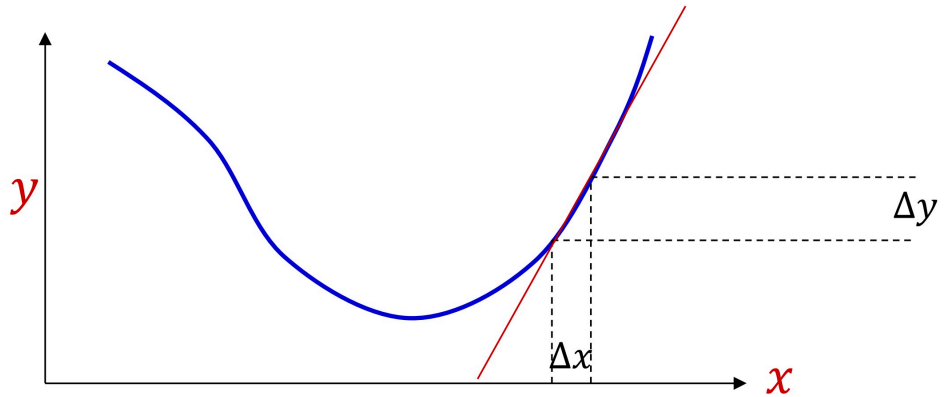
- Estimate the parameters to minimize the empiricial risk over the drawn samples

$$\hat{W} = argmin_W Loss(W)$$

  - I.e. minimize the *empirical error* over the drawn samples

# Lecture 4.1 Derivatives 导数

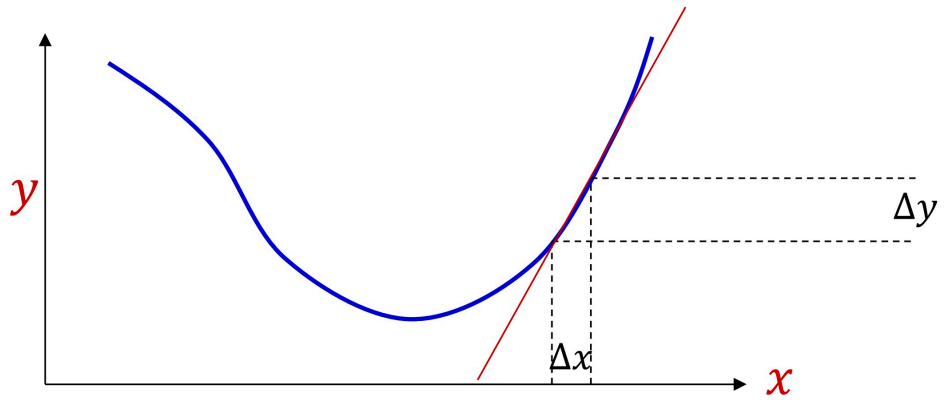
# Derivatives 导数




- A derivative of a function at any point tells us how much a minute increment to the argument of the function will increment the value of the function
  - For any $y = f(x)$, expressed as a multiplier $\alpha$ to a tiny increment $\Delta x$ to obtain the increments $\Delta y$ to the output

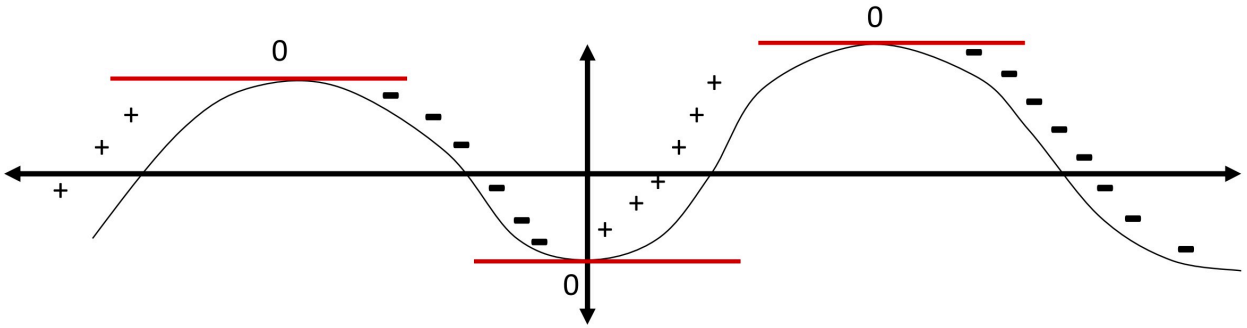
$$\Delta y = \alpha \Delta x \tag{1}$$

# Scalar (标量) function of scalar argument




- When $x$ and $y$ are scalar (标量): $y = f(x)$
  - Derivative
    $$\Delta y = \alpha \Delta x$$
  - Often represented as $\frac{dy}{dx}$
  - Or alternately (and more reasonably) as $f'(x)$

# Scalar function of scalar argument

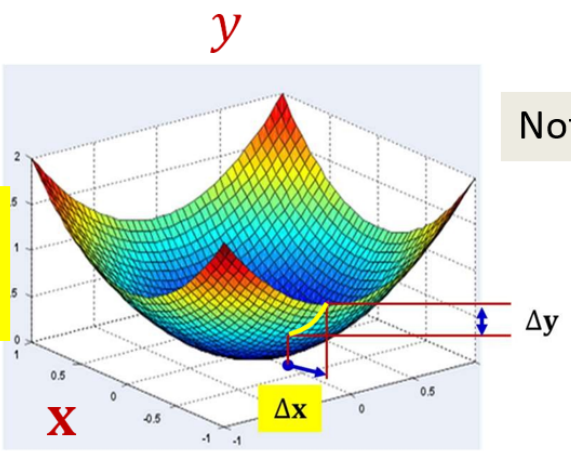



- Derivative $f'(x)$ is the rate of change of the function at $x$
- It will be positive where a small increase in $x$ results in an increase of $f(x)$
- It will be negative where a small increase in $x$ results in an decrease of $f(x)$
- **It will be 0 where the function is locally flat (neither increasing nor decreasing)**

# Multivariate scalar function: Scalar function of vector argument

$y = f(\mathbf{x})$

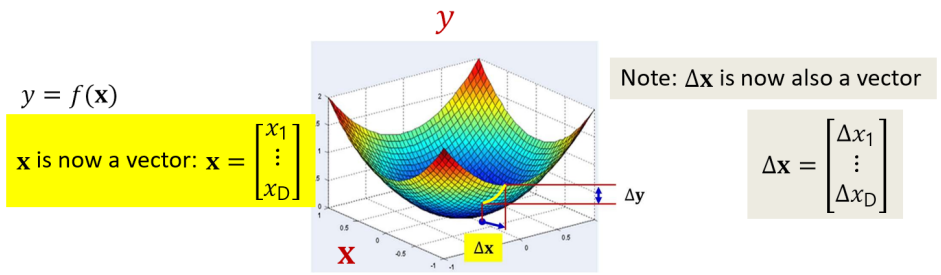
$\mathbf{x}$ is now a vector: $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix}$




Note: $\Delta\mathbf{x}$ is now also a vector

$$\Delta\mathbf{x} = \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix}$$

# Multivariate scalar function: Scalar function of vector argument




$$\Delta y = \alpha \Delta \mathbf{X}$$

- Giving us that $\alpha$ is a row vector: $\alpha = [\alpha_1, \cdots, \alpha_D]$

$$\Delta y = \alpha_1 \Delta x_1 + \alpha_2 \Delta x_2 + \cdots + \alpha_D \Delta x_D$$

- The partial derivative (偏导数) $\alpha_i$ tells us how $y$ increments when only $x_i$ is incremented
- Often represented as $\frac{\partial y}{\partial x_i}$

$$\Delta y = \frac{\partial y}{\partial x_1} \Delta x_1 + \frac{\partial y}{\partial x_2} \Delta x_2 + \cdots + \frac{\partial y}{\partial x_D} \Delta x_D$$

# A well-known vector property

$$\mathbf{u}^T\mathbf{v} = |\mathbf{u}||\mathbf{u}|\cos\theta$$

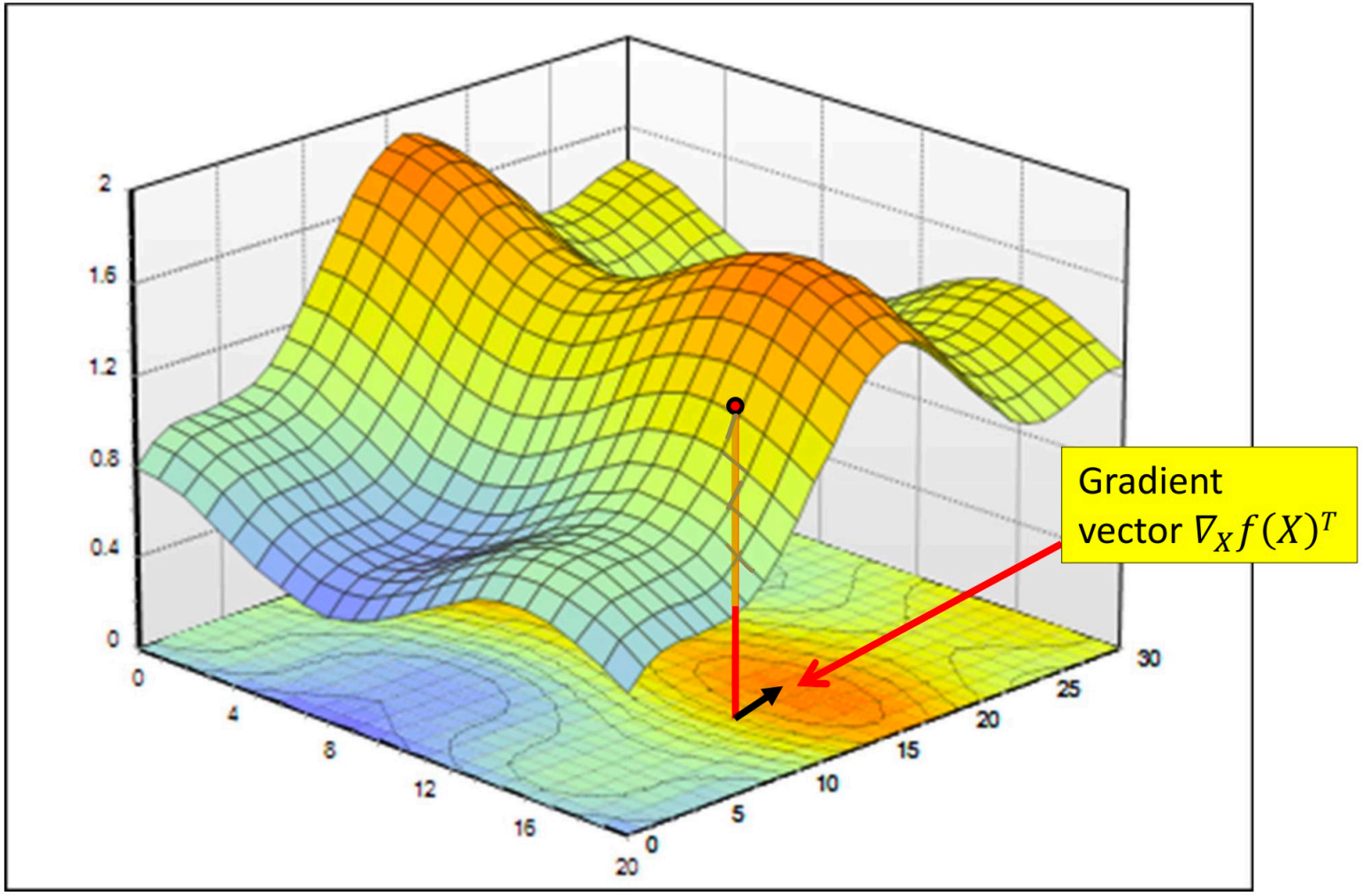
- The inner product (内积/点积) between two vectors of fixed lengths is maximum when the two vectors are aligned
  - i.e. when $\theta = 0$

# Properties of Gradient (梯度)

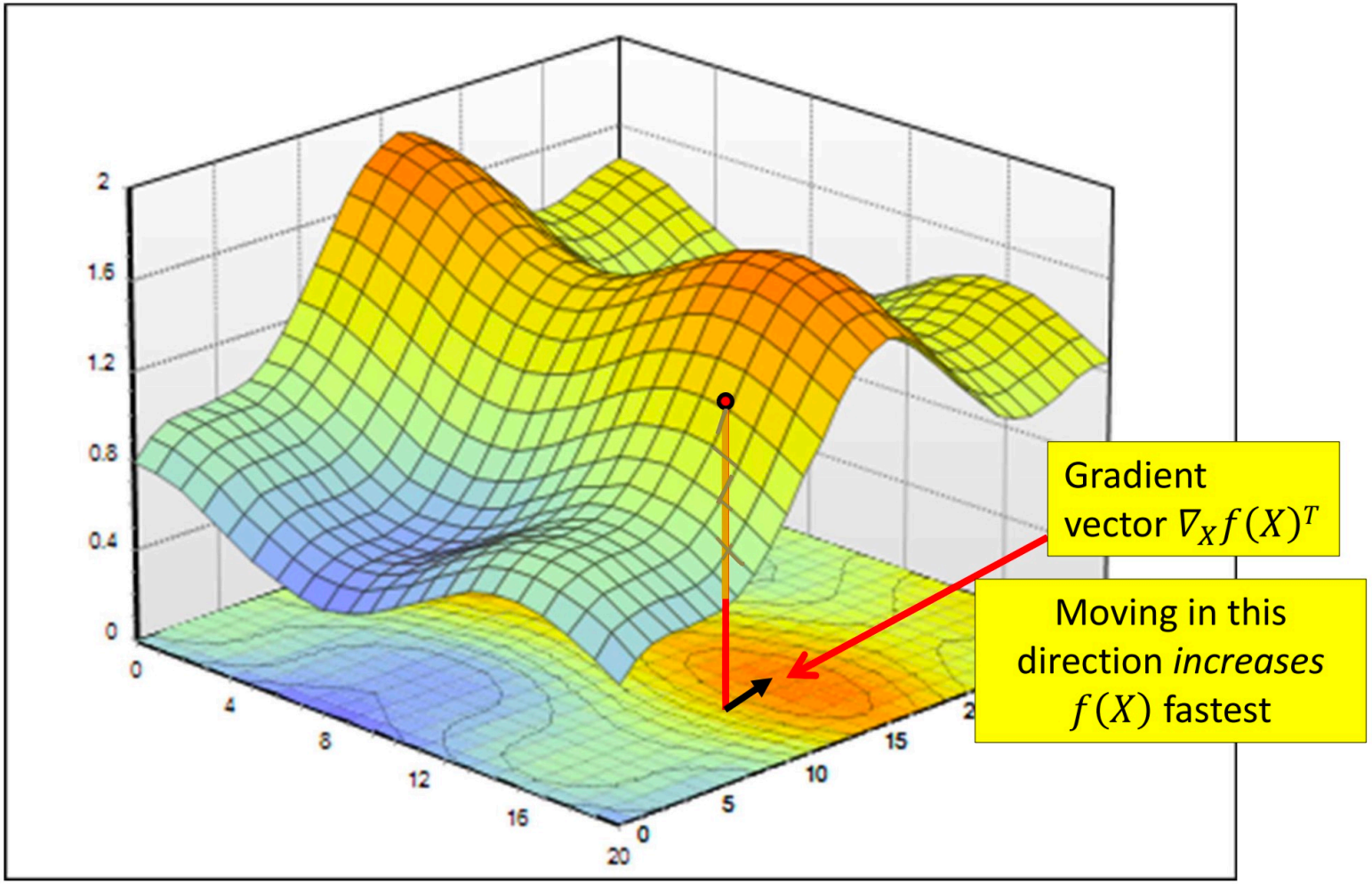
$$df(X) = \Delta_X f(X)\, dX$$

- For an increment $dX$ of any given length, $df(X)$ is max if $dX$ is aligned with $\Delta_X f(X)$
  - The function $f(X)$ increases most rapidly if the input increment $dX$ is exactly in the direction of $\Delta_X f(X)$
- The gradient is the direction of fastest increase in $f(X)$

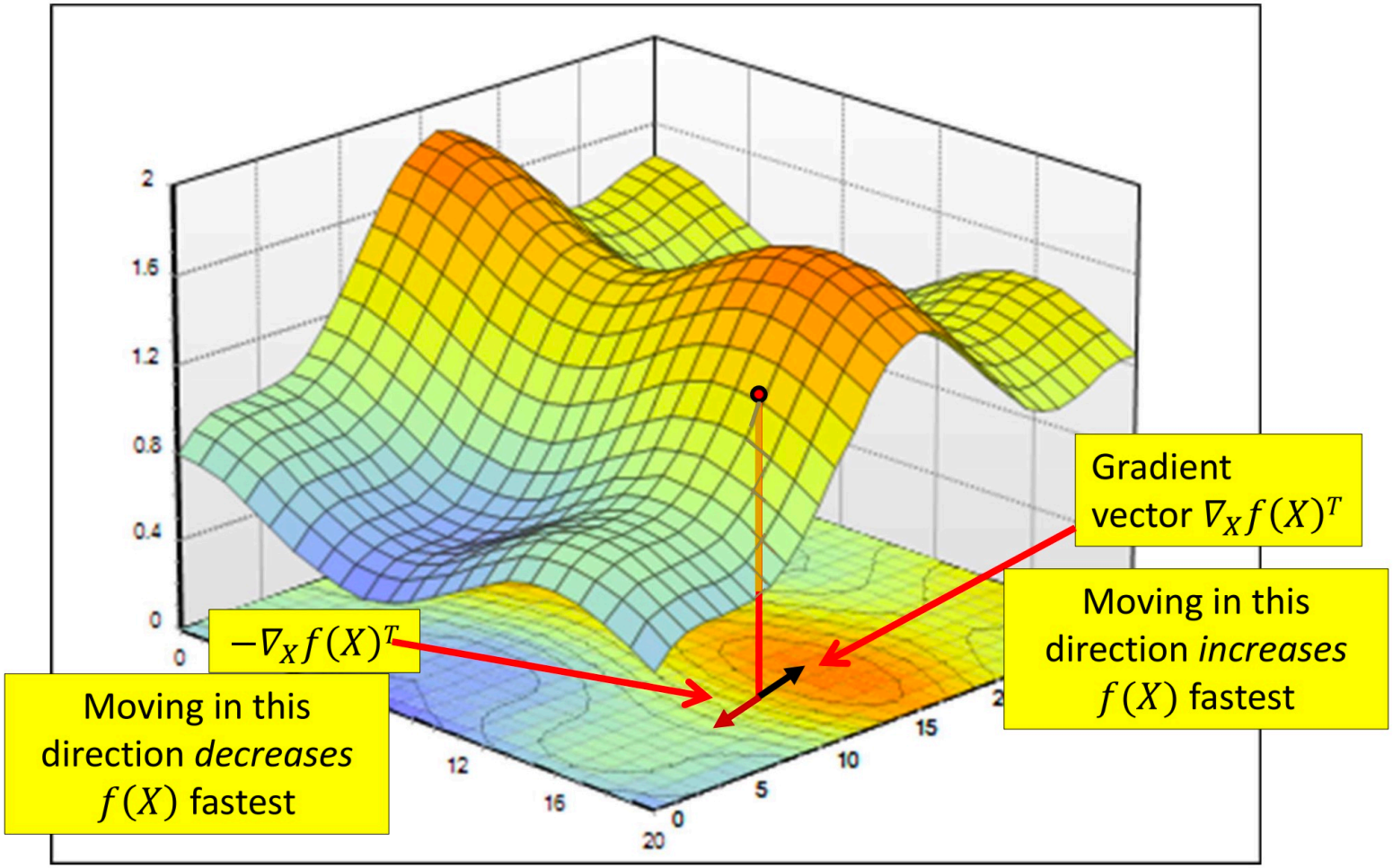
# Gradient


2
1.6
1.2
0.8
0.4
0
0
4
8
12
16
20
0
5
10
15
20
25
30
Gradient
vector $\nabla_X f(X)^T$

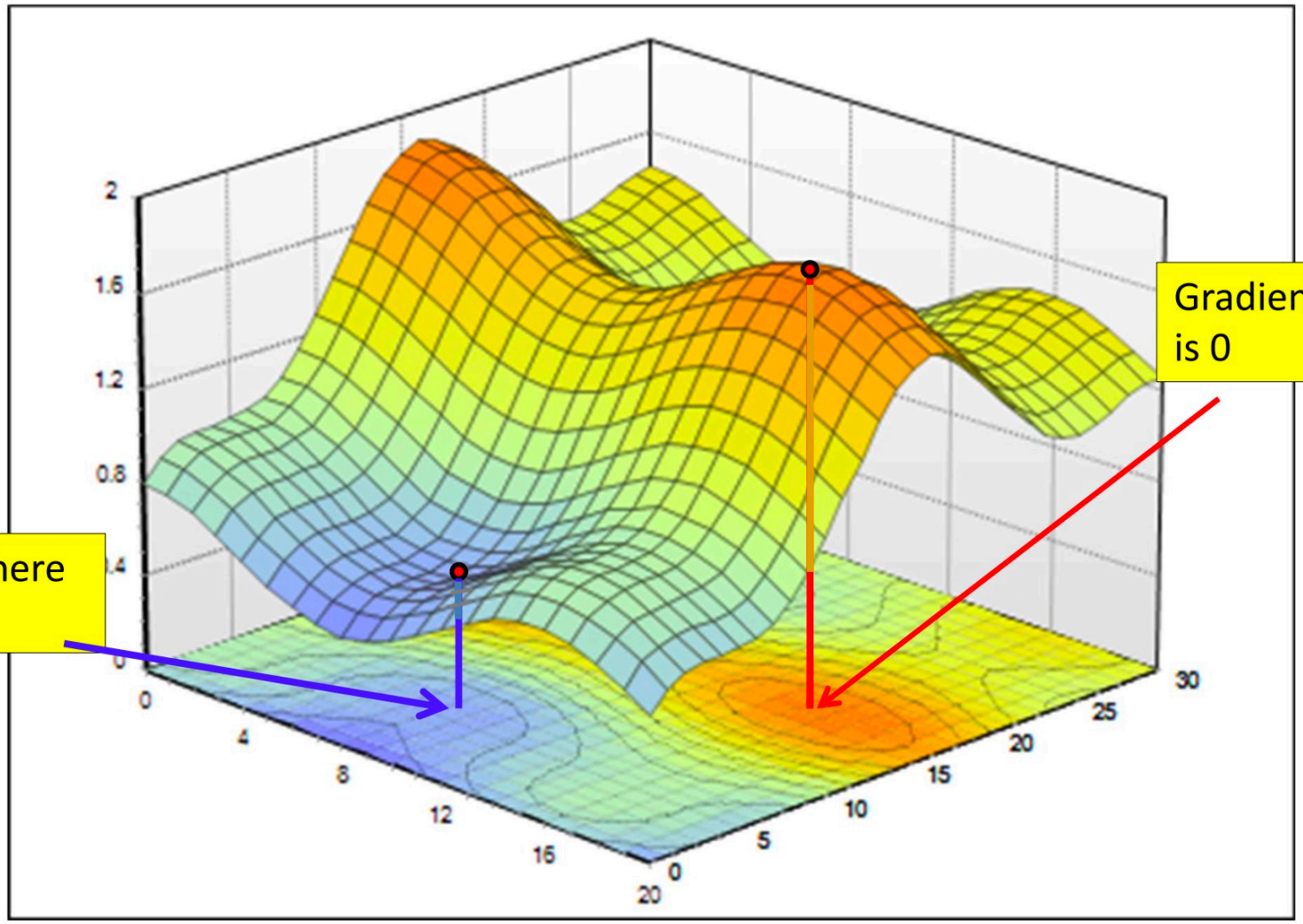
# Gradient


2
1.6
1.2
0.8
0.4
0
0
4
8
12
16
20
0
5
10
15
Gradient vector $\nabla_X f(X)^T$
Moving in this direction *increases* $f(X)$ fastest

# Gradient

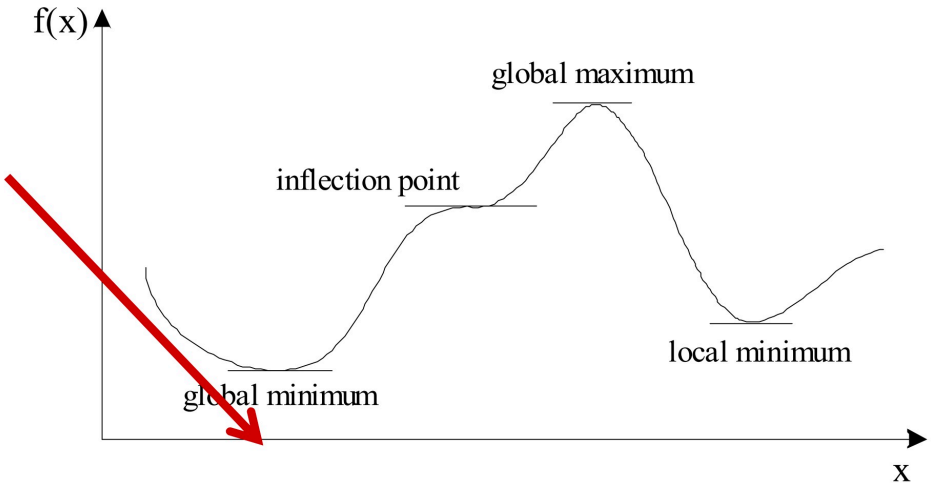

2
1.6
1.2
0.8
0.4
0
0
12
16
20
0
5
10
15
Gradient vector $\nabla_X f(X)^T$
Moving in this direction *increases* $f(X)$ fastest
$-\nabla_X f(X)^T$
Moving in this direction *decreases* $f(X)$ fastest

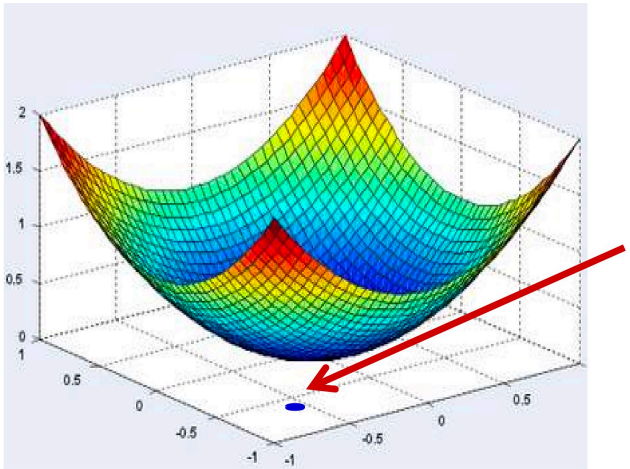
# Gradient


Gradient here is 0
Gradient here is 0
2
1.6
1.2
0.8
0
4
8
12
16
20
0
5
10
15
20
25
30

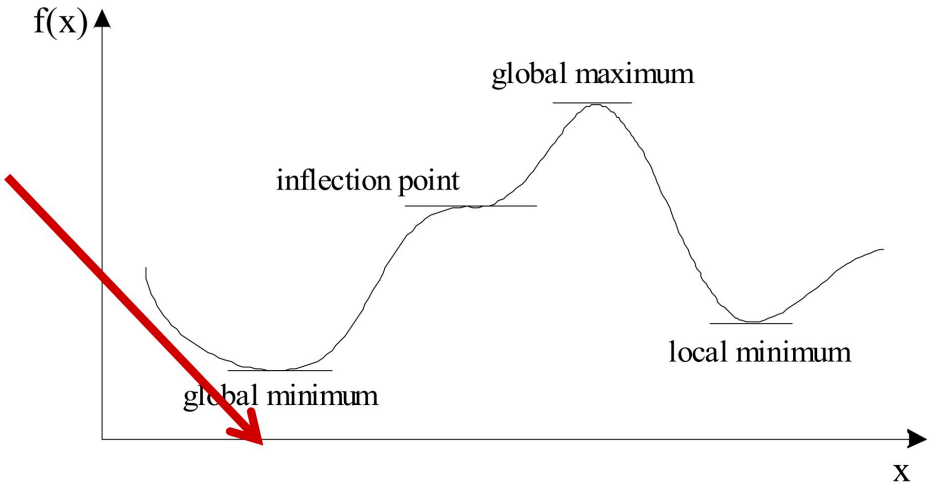
# Lecture 4.2 Optimization 优化

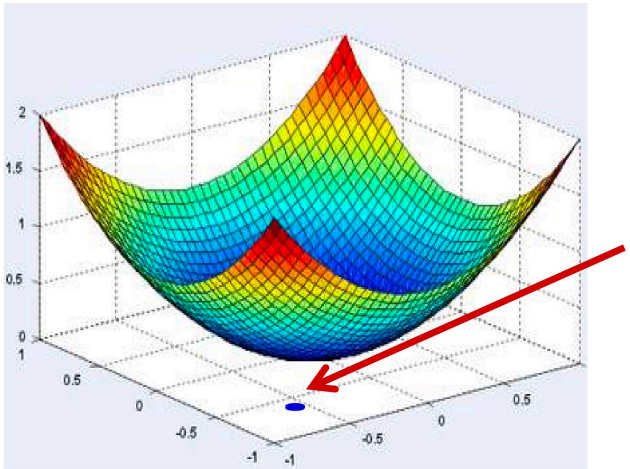
# The problem of optimization



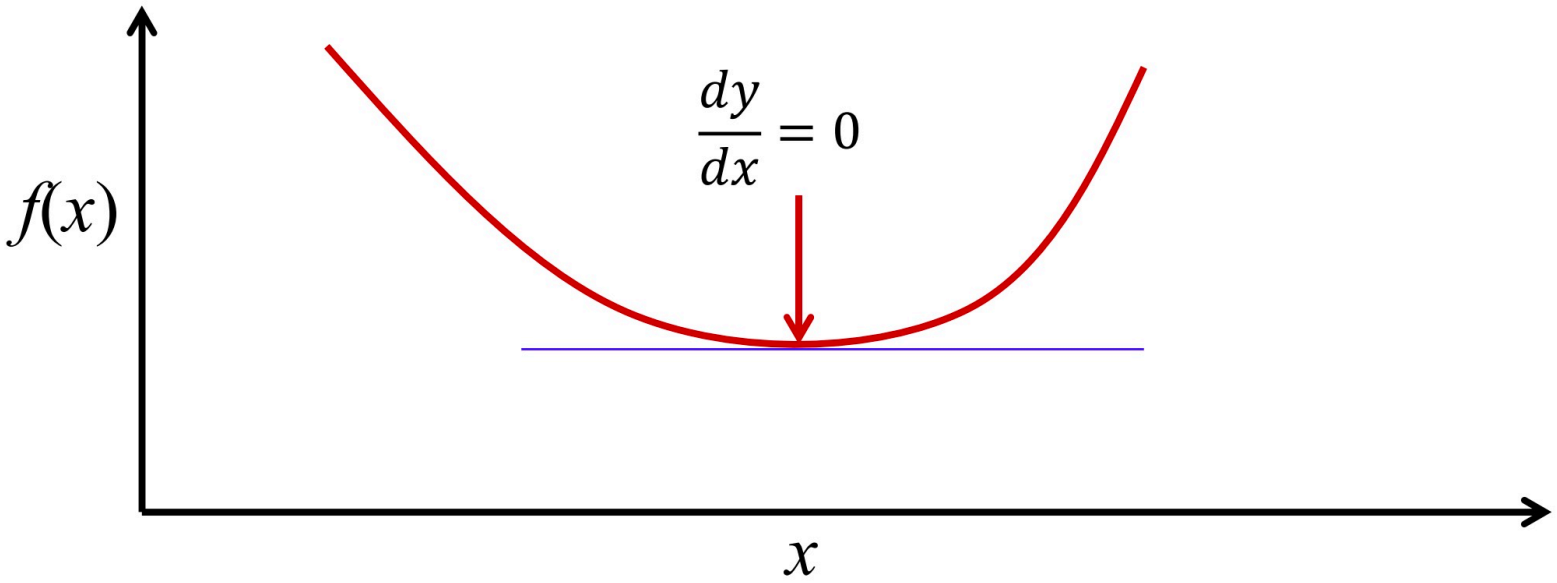
# The problem of optimization



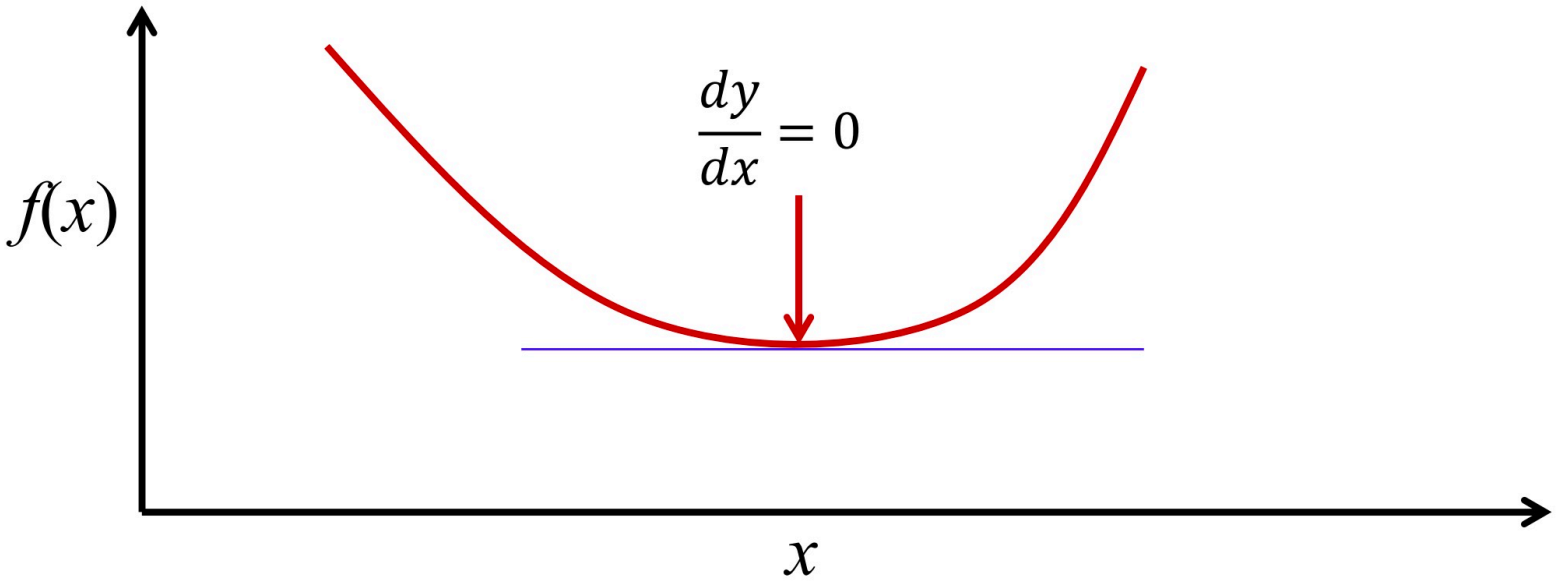




- General problem of optimization: Given a function $f(\mathbf{x})$ of some variable $\mathbf{x}$ ...
- Find the value of $\mathbf{x}$ where $f(\mathbf{x})$ is minimum (最小値)

# Finding the minimum of a function



# Finding the minimum of a function

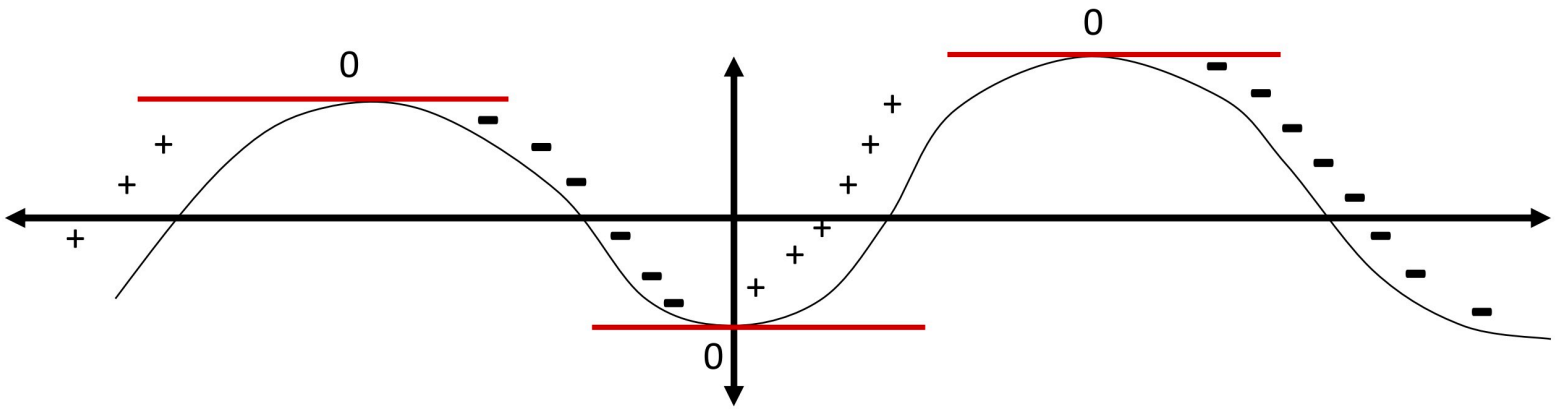



- Find the value $x$ at which $f'(x) = 0$
  - Solve

$$\frac{df(x)}{dx} = 0 \tag{2}$$

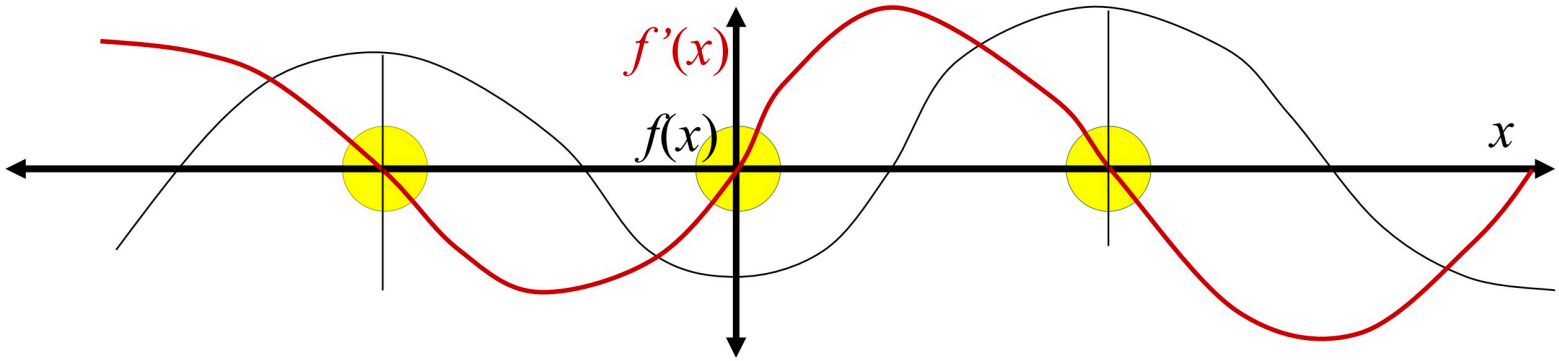
- The solution is a "turning point" (转折点)
  - Derivatives go from positive to negative at this point
- But is it a minimum?

# Turning Points



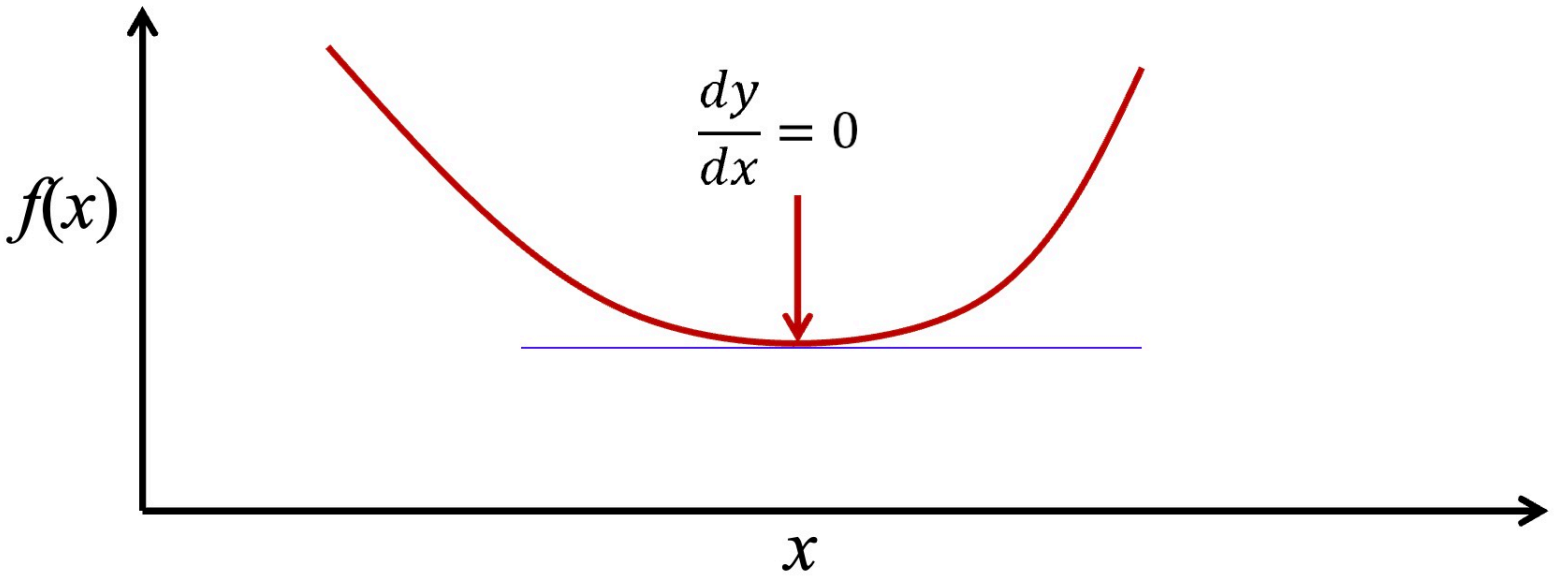

- Both *maxima* and *minima* have zero derivative
- Both are turning points

# Derivatives of a curve




- Both *maxima* and *minima* are turning points
- Both *maxima* and *minima* have zero derivative

# Finding the minimum or maximum of a function

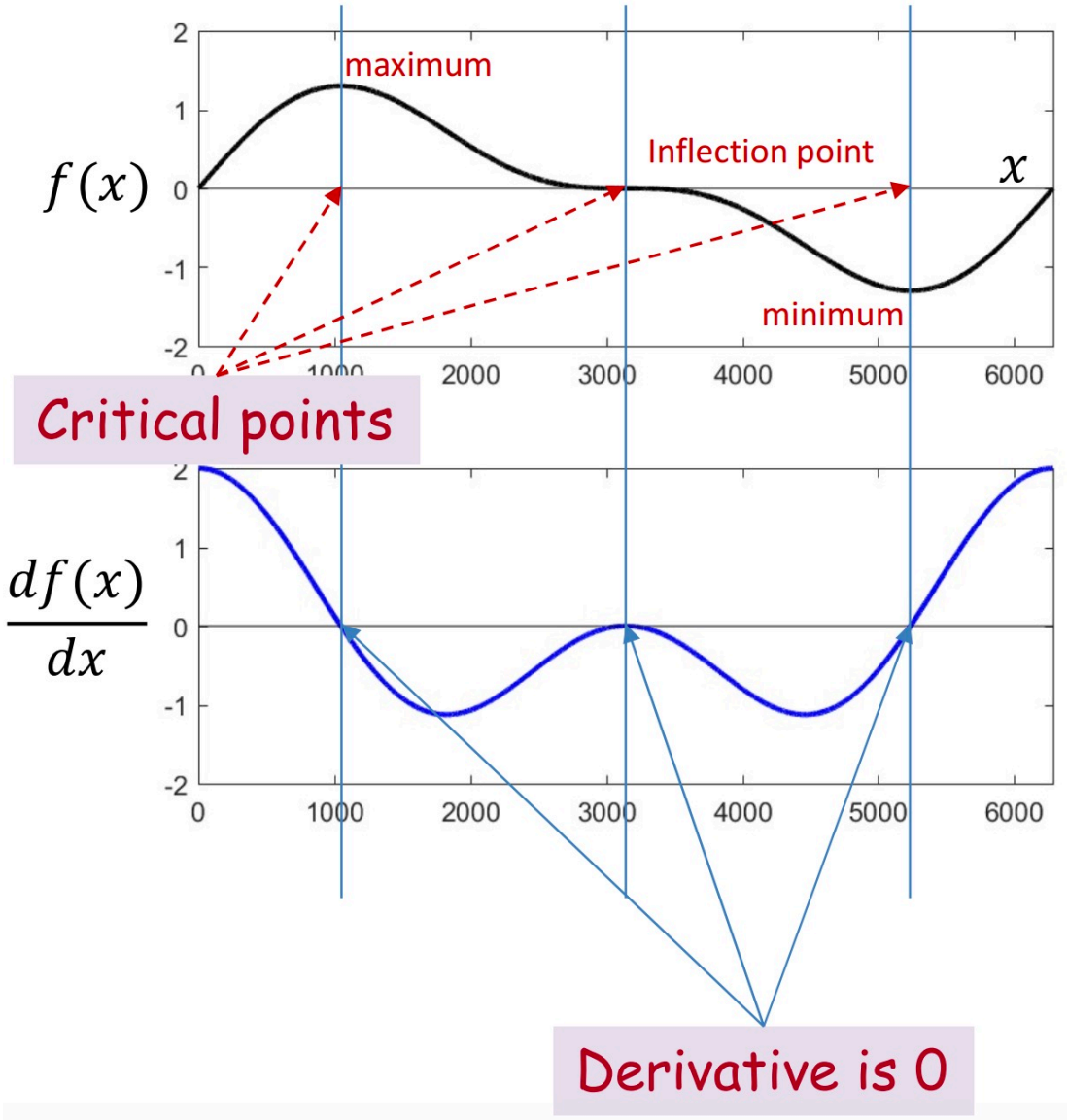



- Find the value $x$ at which $f'(x) = 0$: Solve

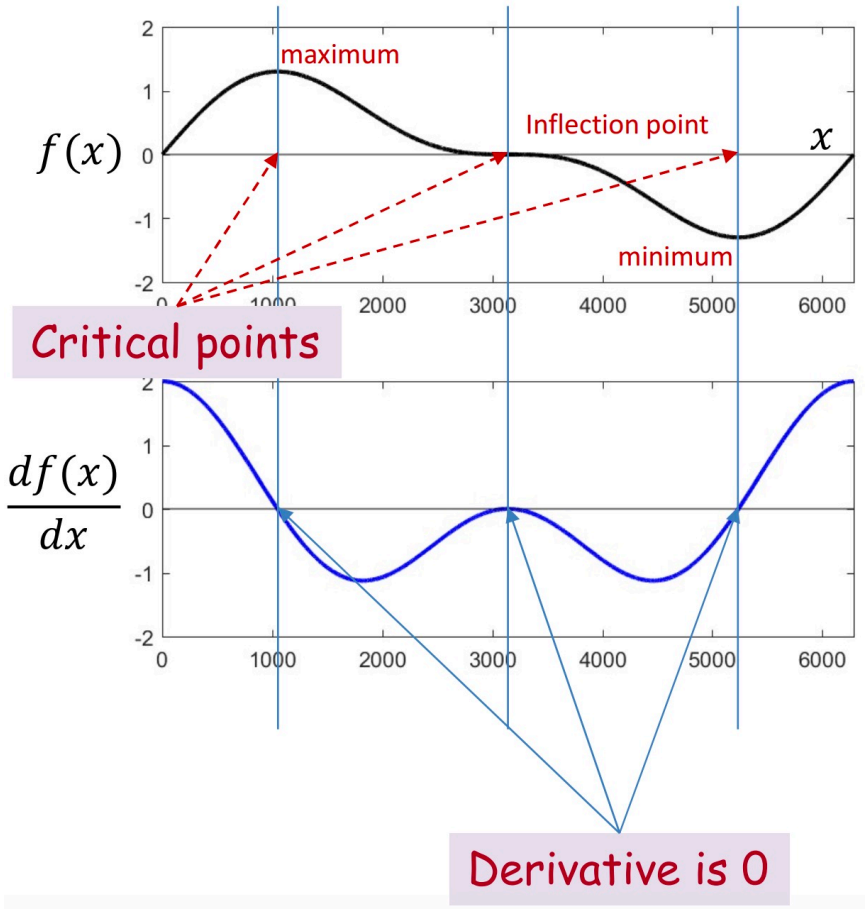
$$\frac{df(x)}{dx} = 0 \tag{3}$$

- The solution $x_{soln}$ is a **turning point**

# A note on derivatives of functions of single variable

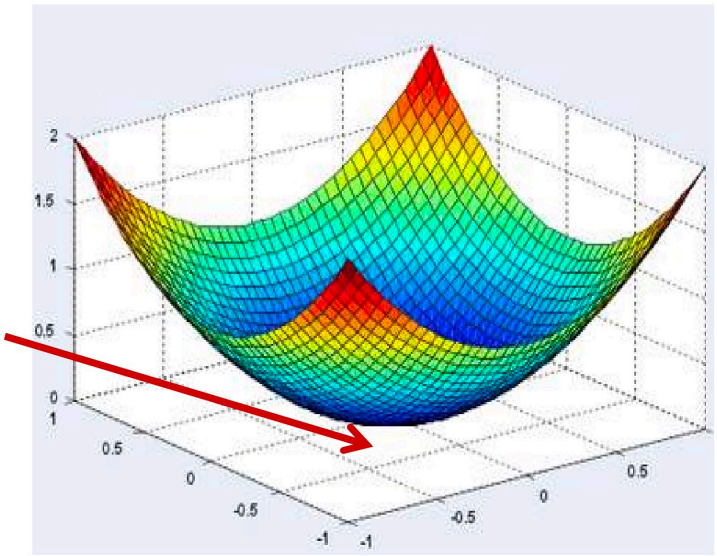


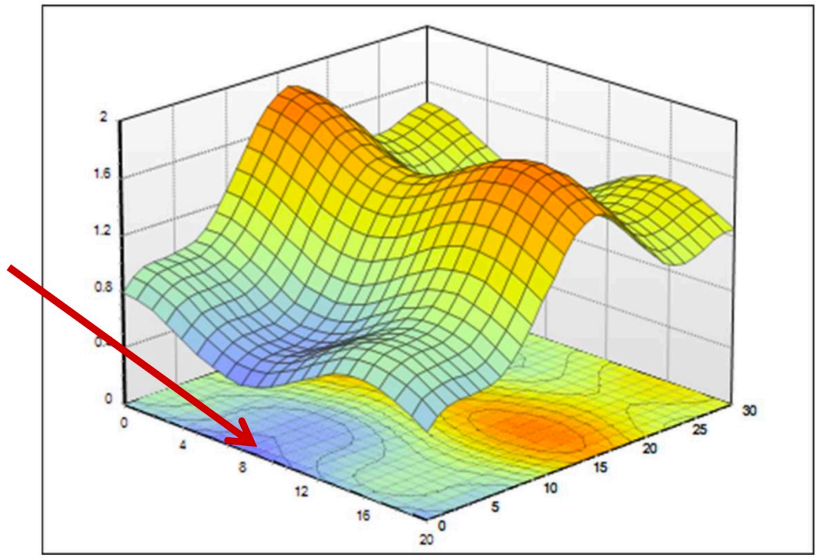
# A note on derivatives of functions of single variable




- All locations with zero derivative are critical points (临界点)
  - Can be local maxima, local minima, or inflection points (拐点)

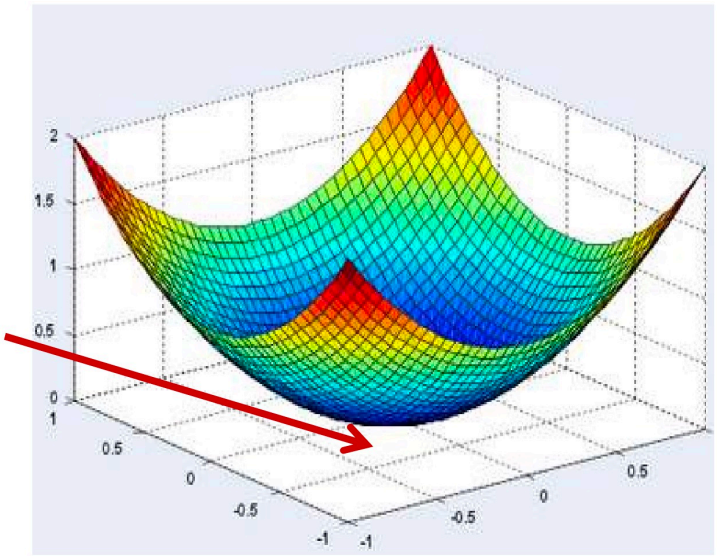
# What about functions of multiple variables?

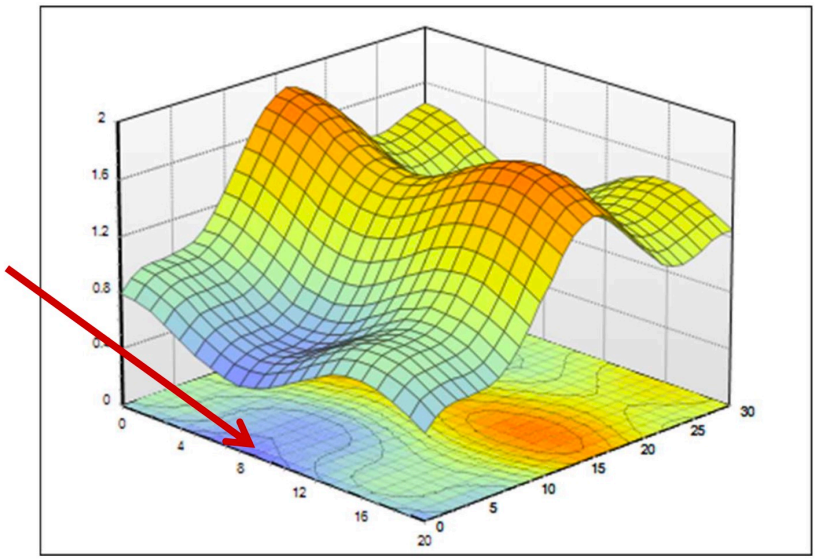



- The optimum point (最优点) is still "turning" point
  - Shifting in any direction will increase the value
  - For smooth functions, at the minimum/maximum, the gradient is 0 points

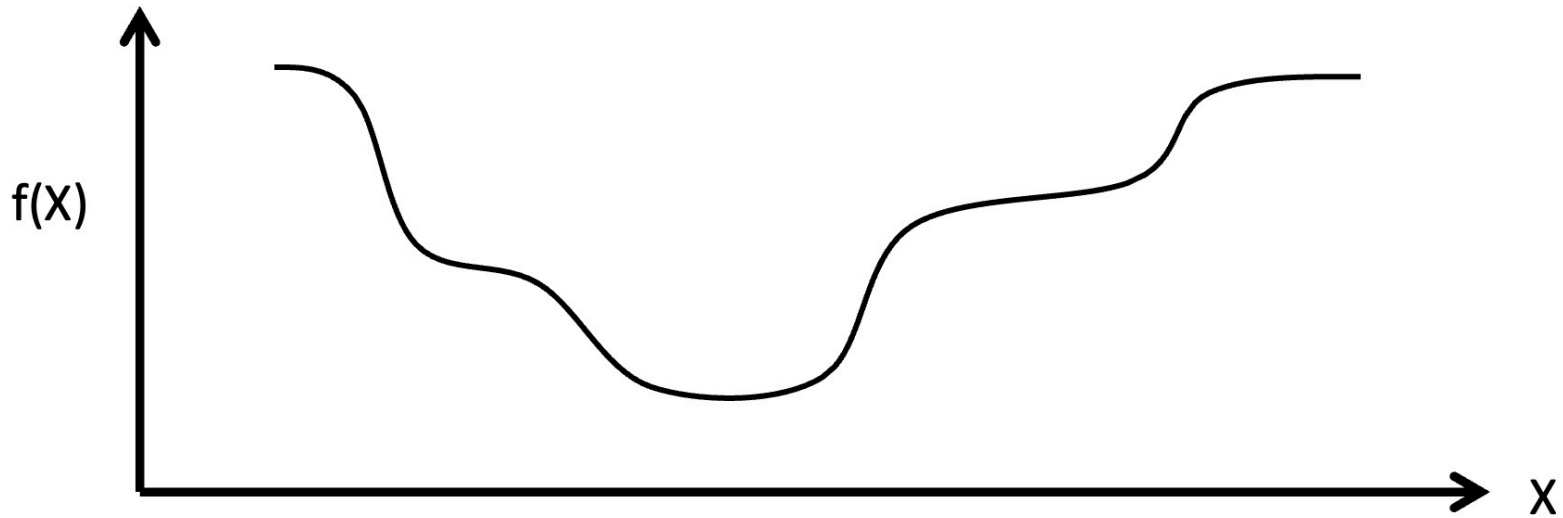
# Finding the minimum of a scalar function of a multivariate input



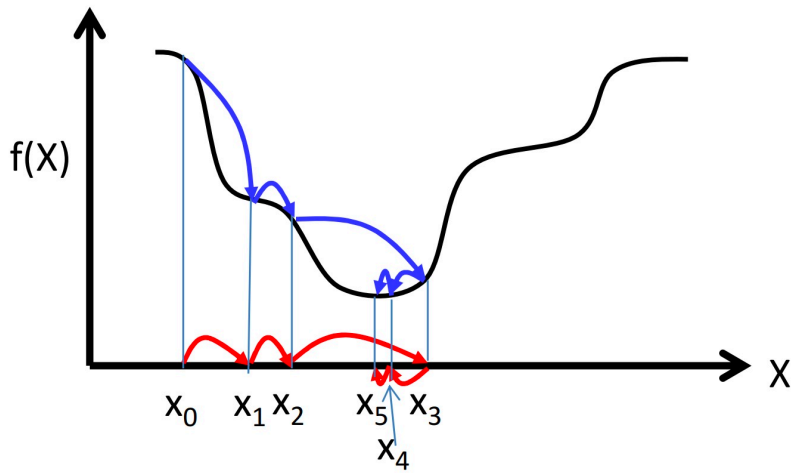

- The optimum point is a turning point –the gradient will be 0
- Find the location where the gradient is 0

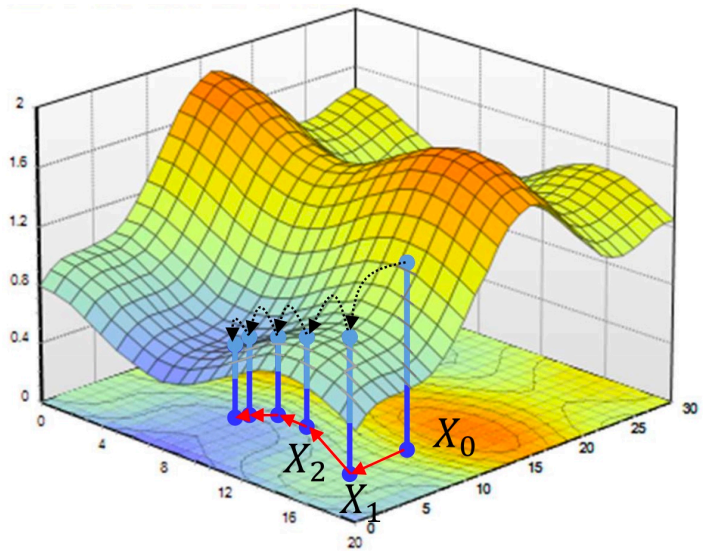
# Closed Form Solutions (闭合解) are not always available




- Often it is not possible to simply solve $\Delta_X f(X) = 0$
  - The function to minimize/maximize may be intractable
- In these situations, iterative solutions are used
  - Begin with a "guess" for the optimal and refine it iteratively until the correct value is obtained

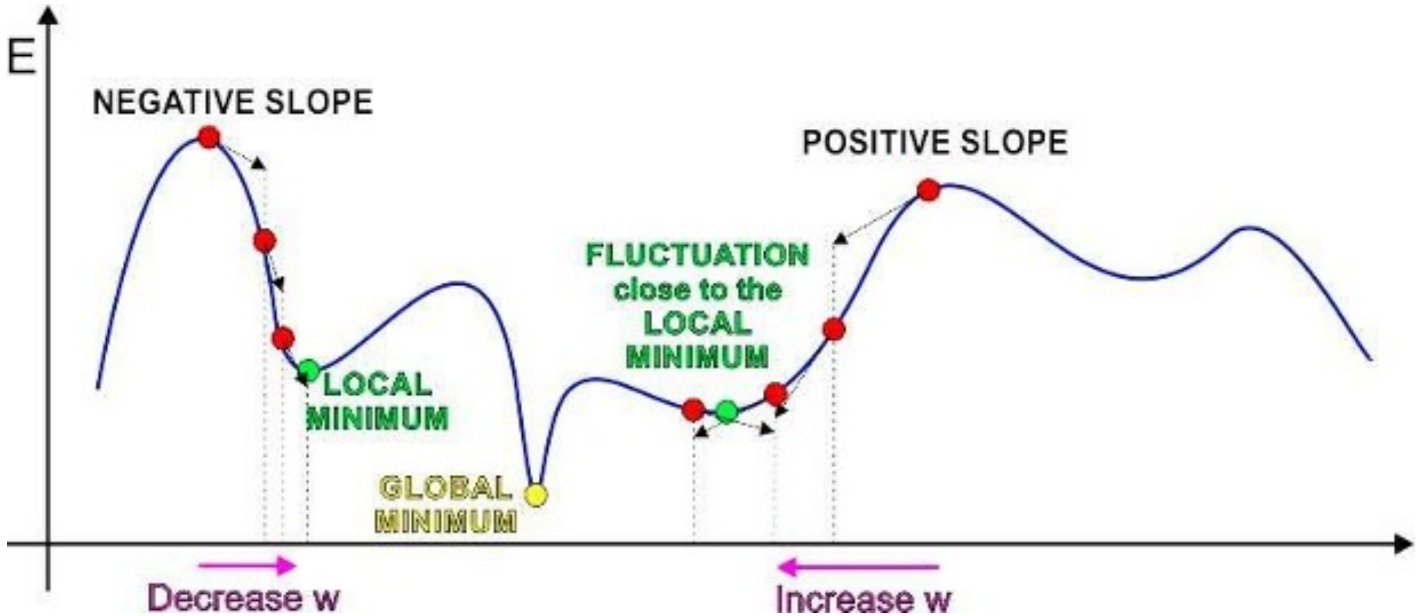
# Iterative solutions 迭代解



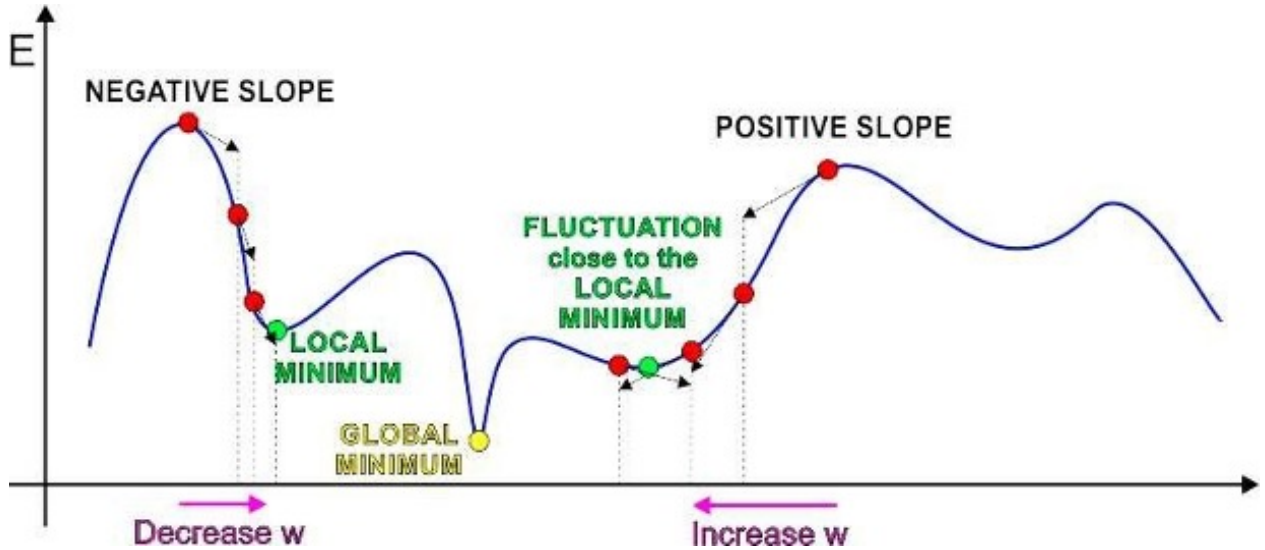

- Iterative solutions
  - Start from an initial guess $X_0$ for the optimal $X$
  - Update the guess towards a (hopefully) "better" value of $f(X)$
  - Stop when $f(X)$ no longer decreases
- Problems:
  - Which direction to step in
  - How big must the steps be

# The Approach of Gradient Descent 梯度下降方法



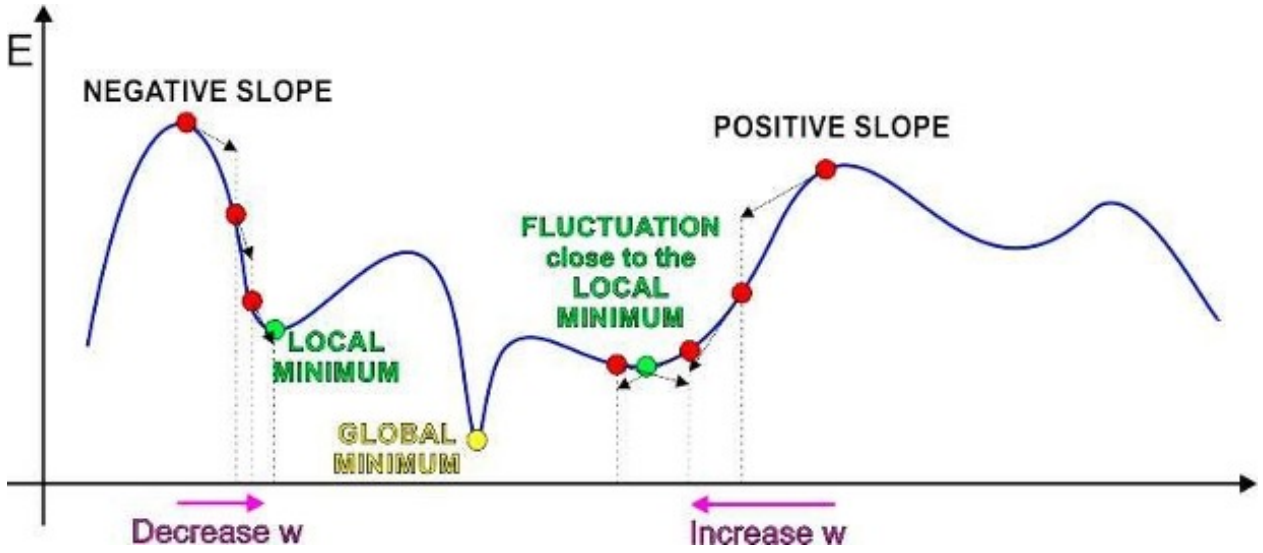

- Iterative solution:
  - Start at some point
  - Find direction in which to shift this point to decrease error
    - This can be found from the derivative of the function
    - A negative derivative $\rightarrow$ moving right decreases error
    - A positive derivative $\rightarrow$ moving left decreases error
  - Shift point in this direction

# The Approach of Gradient Descent 梯度下降方法




- Iterative solution: Trivial algorithm
  - Initialize $x^0$
  - While $f'(x^k) \neq 0$
    - If sign$(f'(x^k))$ is positive: $x^{k+1} = x^k - step$
    - Else: $x^{k+1} = x^k + step$
- Question: How to set the step?

# The Approach of Gradient Descent

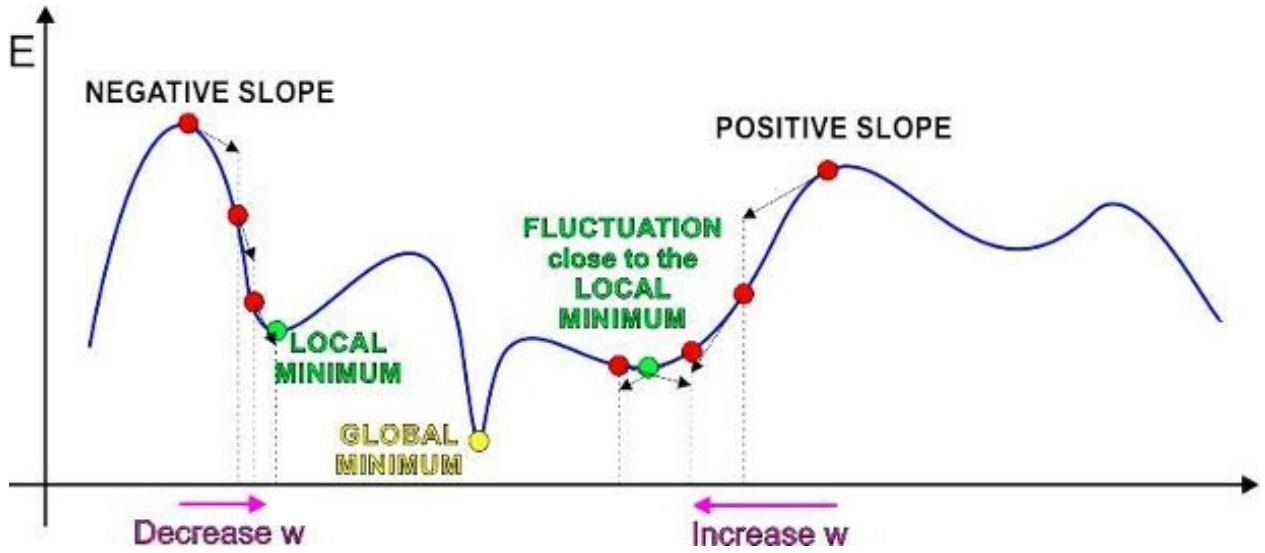



- Iterative solution: Trivial algorithm
  - Initialize $x^0$
  - While $f'(x^k) \neq 0$

$$x^{k+1} = x^k - \text{sign}(f'(x^k)) \cdot step \tag{4}$$

- Identical to previous algorithm

# The Approach of Gradient Descent




- Iterative solution: Trivial algorithm
  - Initialize $x^0$
  - While $f'(x^k) \neq 0$

$$x^{k+1} = x^k - \eta^k f'(x^k)$$

- $\eta^k$ is the "step size" (步长). Why is it necessary?

# Gradient descent/ascent (multivariate)

- The gradient descent/ascent method to find the minimum or maximum of a function $f$ iteratively
  - To find a *maximum* move in the direction of the gradient

$$x^{k+1} = x^k + \eta^k \Delta_x f(x^k)^T$$

  - To find a *minimum* move exactly opposite the direction of the gradient

$$x^{k+1} = x^k - \eta^k \Delta_x f(x^k)^T$$

- Many solutions to choosing step size $\eta^k$

# Gradient descent convergence (收斂) criteria

- How to decide when gradient descent algorithm converges?

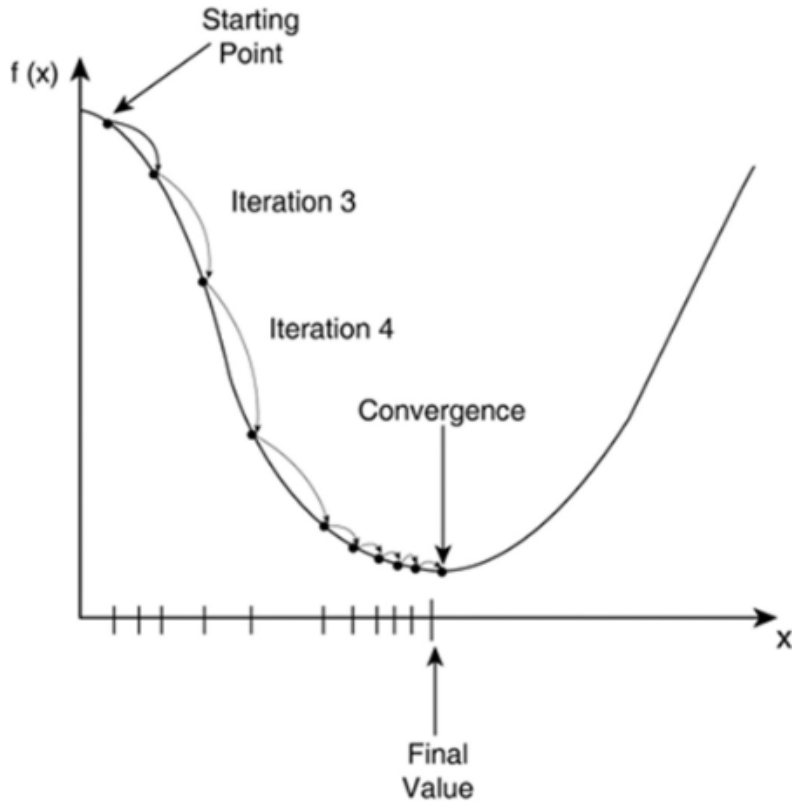
# Gradient descent convergence (收敛) criteria

- How to decide when gradient descent algorithm converges?
- The gradient descent algorithm converges when one of the following criteria is satisfied

$$|f(x^{k+1}) - f(x^k)| < \epsilon_1$$

Or

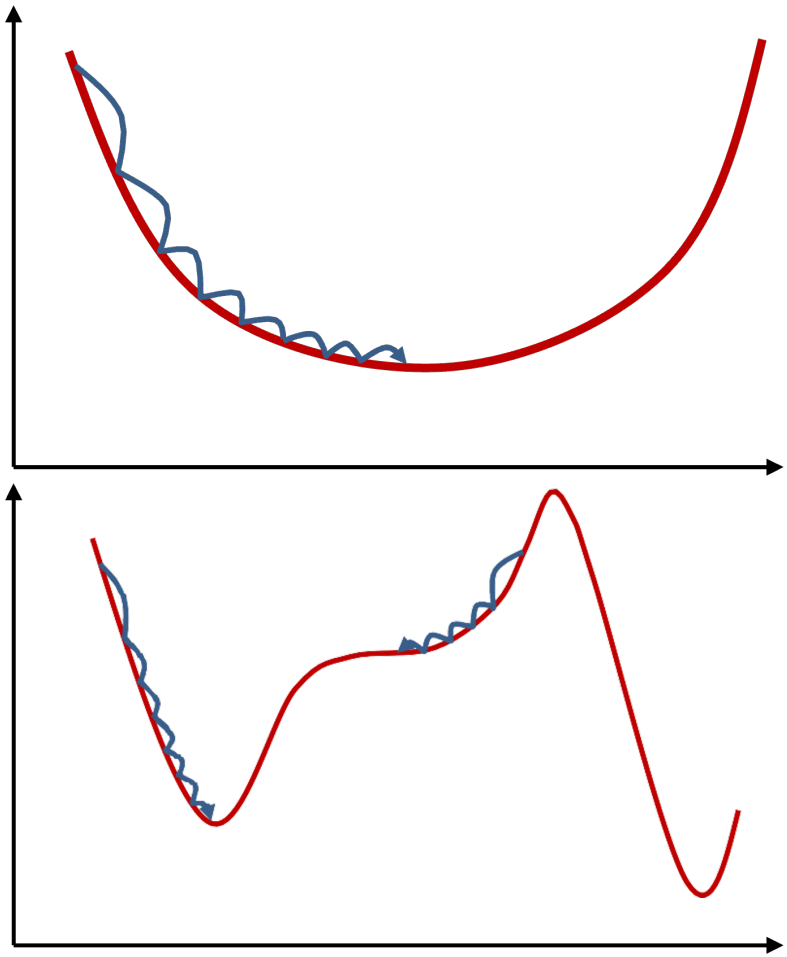
$$\|\Delta_x f(x^k)\| < \epsilon_2$$



# Overall Gradient Descent Algorithm

- Initialize:
  - $x^0$
  - $k = 0$
- do
  - $x^{k+1} = x^k - \eta^k \Delta_x f(x^k)^T$
  - $k = k + 1$
- while $|f(x^{k+1}) - f(x^k)| < \epsilon$

# Convergence of Gradient Descent



- For appropriate step size, for convex (bowl-shaped) functions gradient descent will always find the minimum.
- For non-convex functions it will find a local minimum or an inflection point

# Lecture 4.3 Training a Neural Network

# Problem Statement

- Given a training set of input-output pairs $(X_1, d_1)$, $(X_2, d_d)$, ..., $(X_N, d_N)$
- Minimize the following function

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

w.r.t $W$

- This is problem of function minimization
  - An instance of optimization

# Gradient Descent to train a network

- Initialize:
  - $W^0$
  - $k = 0$
- do
  - $W^{k+1} = W^k - \eta^k \Delta_W f(W^k)^T$
  - $k = k + 1$
- while $|Loss(W^{k+1}) - Loss(W^k)| < \epsilon$

# Problem Setup: Things to define

- Given a training set of input-output pairs

$$(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$$

What are these input-output pairs?

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

# Problem Setup: Things to define

- Given a training set of input-output pairs

$$(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$$

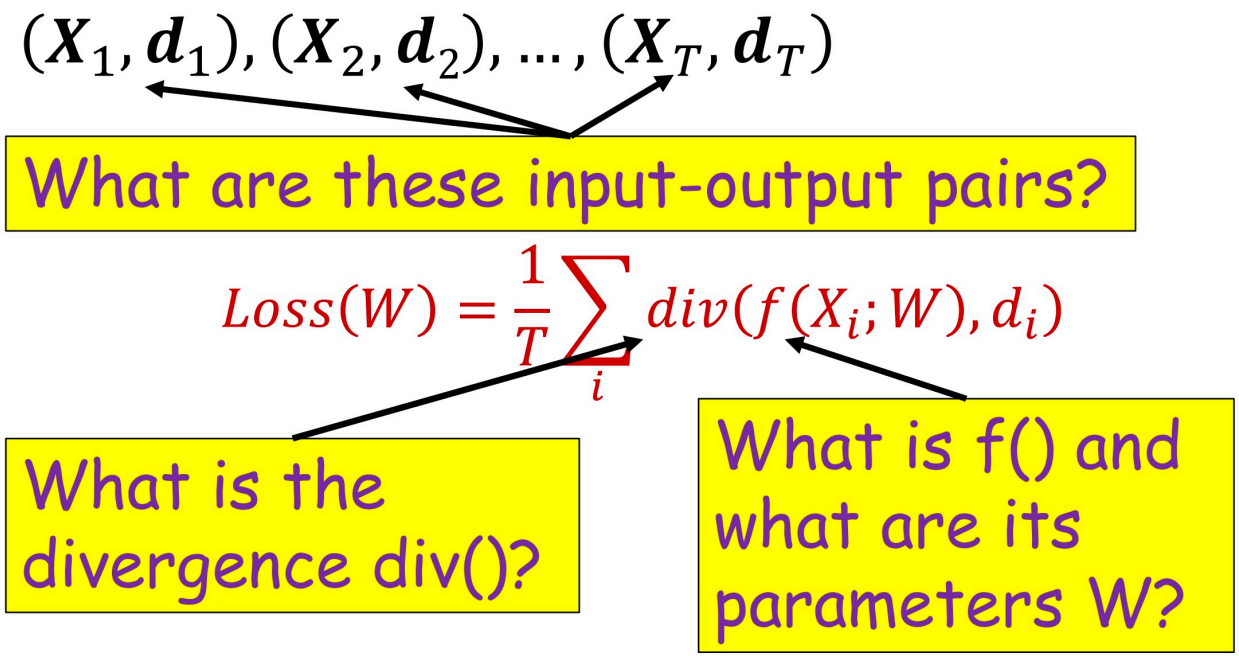
What are these input-output pairs?

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

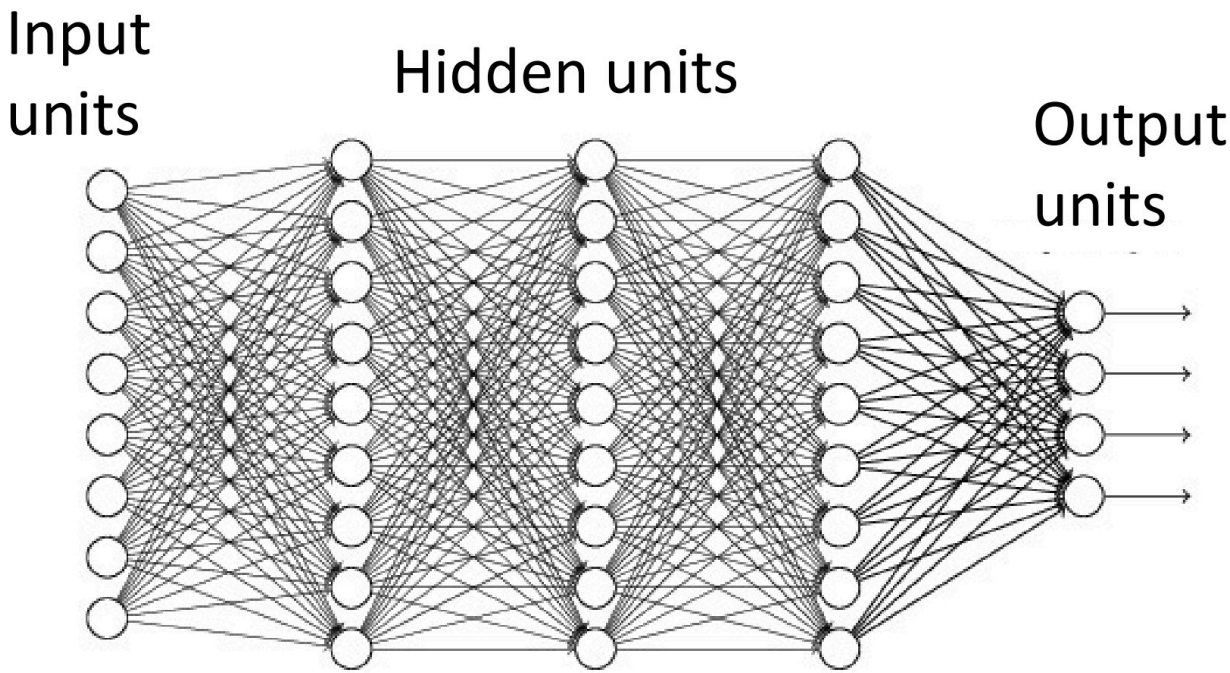
What is f() and what are its parameters W?

# Problem Setup: Things to define

- Given a training set of input-output pairs

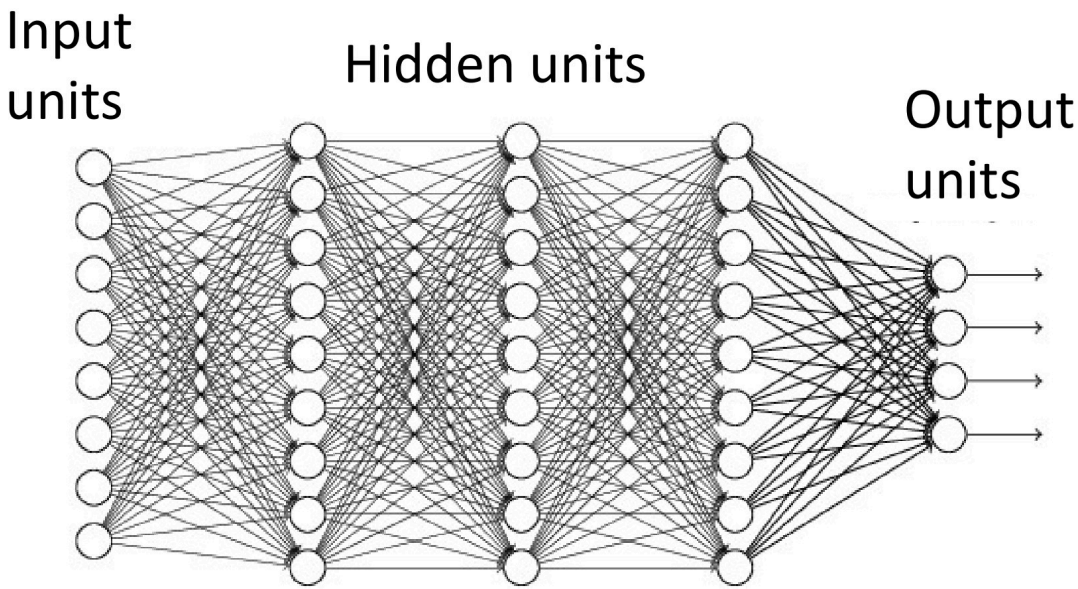


# What is $f()$? Typical network



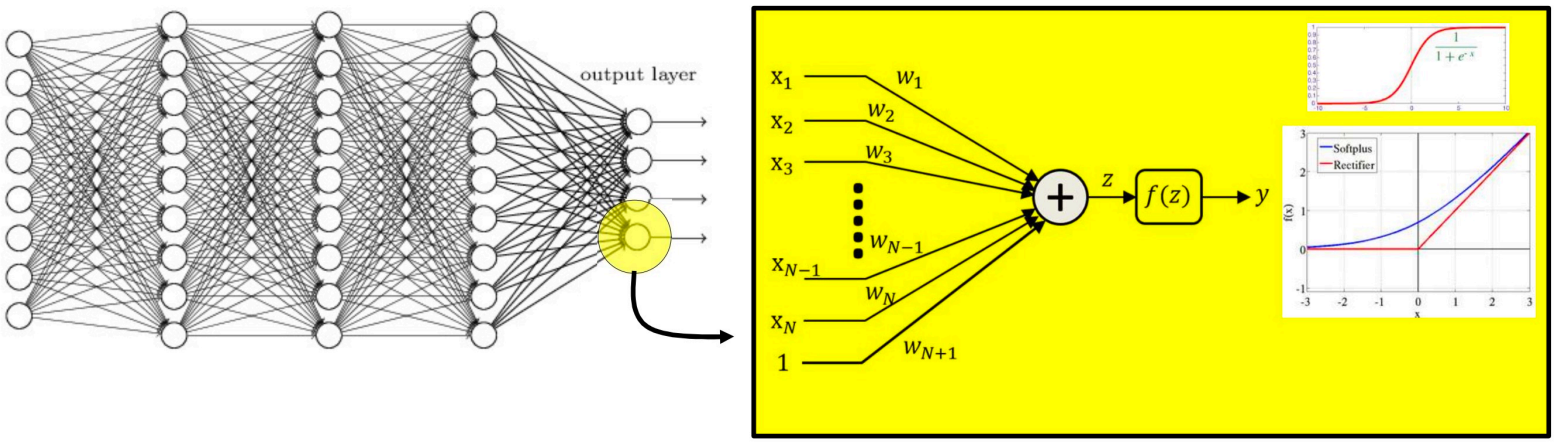

- Multi-layer perceptron
- A directed network with a set of inputs and outputs
  - No loops

# Typical network




- We assume a "layered" network for simplicity
  - Each "layer" of neurons only gets inputs from the earlier layer(s) and outputs signals only to later layer(s)
  - We will refer to the inputs as the input layer
    - No neurons here –the "layer" simply refers to inputs
  - We refer to the outputs as the output layer
    - Intermediate layers are "hidden" layers

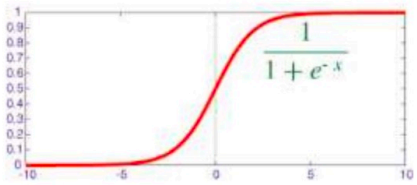
# The individual neurons



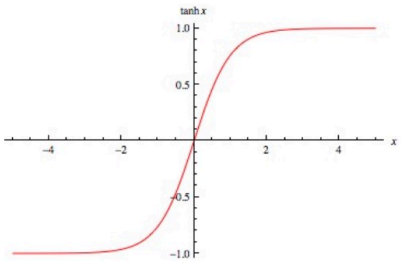

- Individual neurons operate on a set of inputs and produce a single output

$$y = f\left(\sum_i w_i\, x_i + b\right)$$

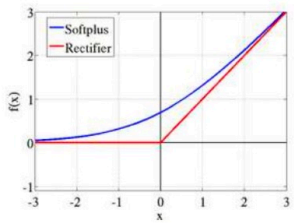
# Activations and their derivatives




$$f(z) = \frac{1}{1+\exp(-z)} \qquad f'(z) = f(z)(1-f(z))$$




$$f(z) = \tanh(z) \qquad f'(z) = (1-f^2(z))$$

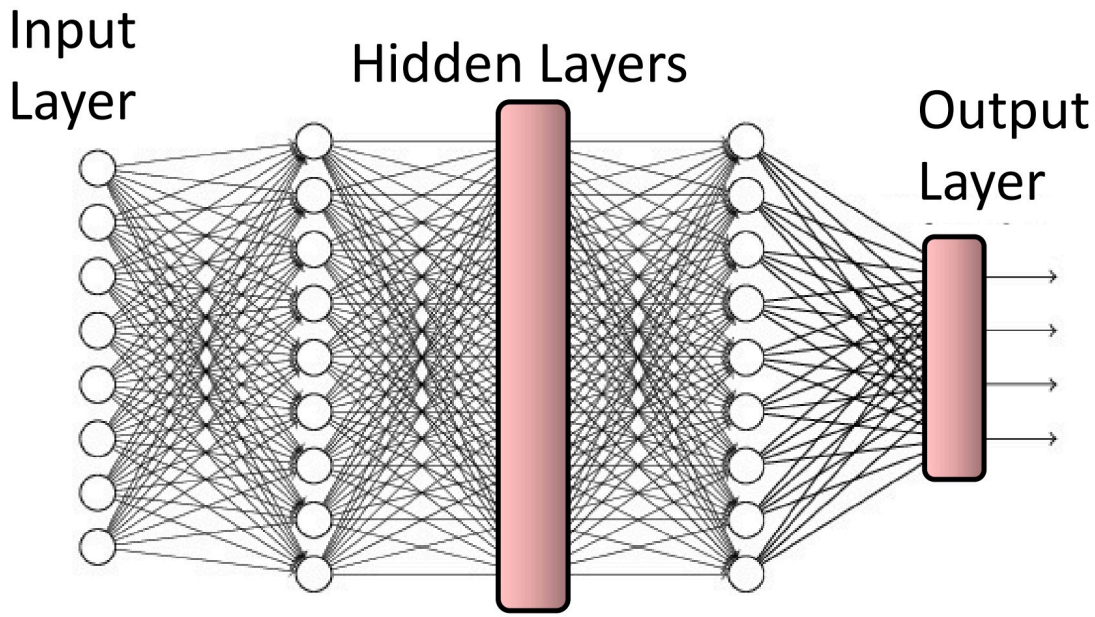



$$f(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases} \qquad [*] \quad f'(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$$

$$f(z) = \log(1+\exp(z)) \qquad f'(z) = \frac{1}{1+\exp(-z)}$$

- Some popular activation functions and their derivatives

# Vector Activations

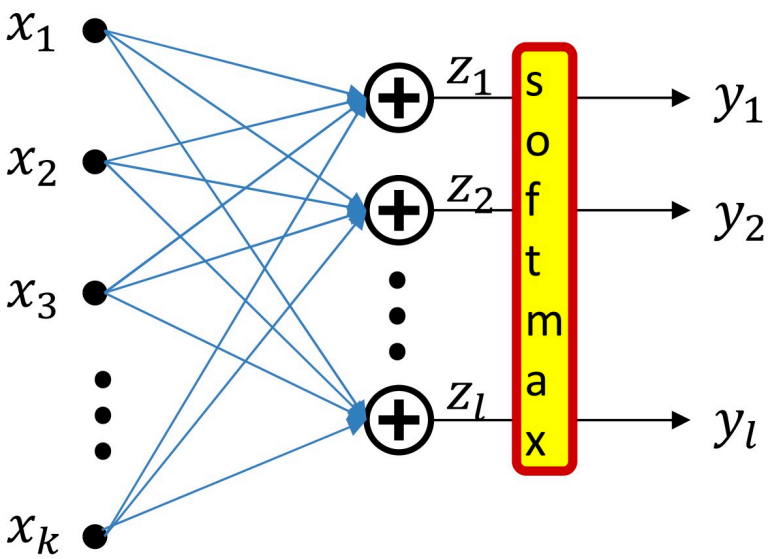



- We can also have neurons that have multiple coupled outputs

$$[y_1, y_2, \ldots, y_l] = f(x_1, x_2, \ldots, x_k; W)$$

  - Function $f(x)$ operates on set of inputs to produce a set of outputs
  - Modifying a single parameter in $W$ will affect all outputs

# Vector activation example: Softmax



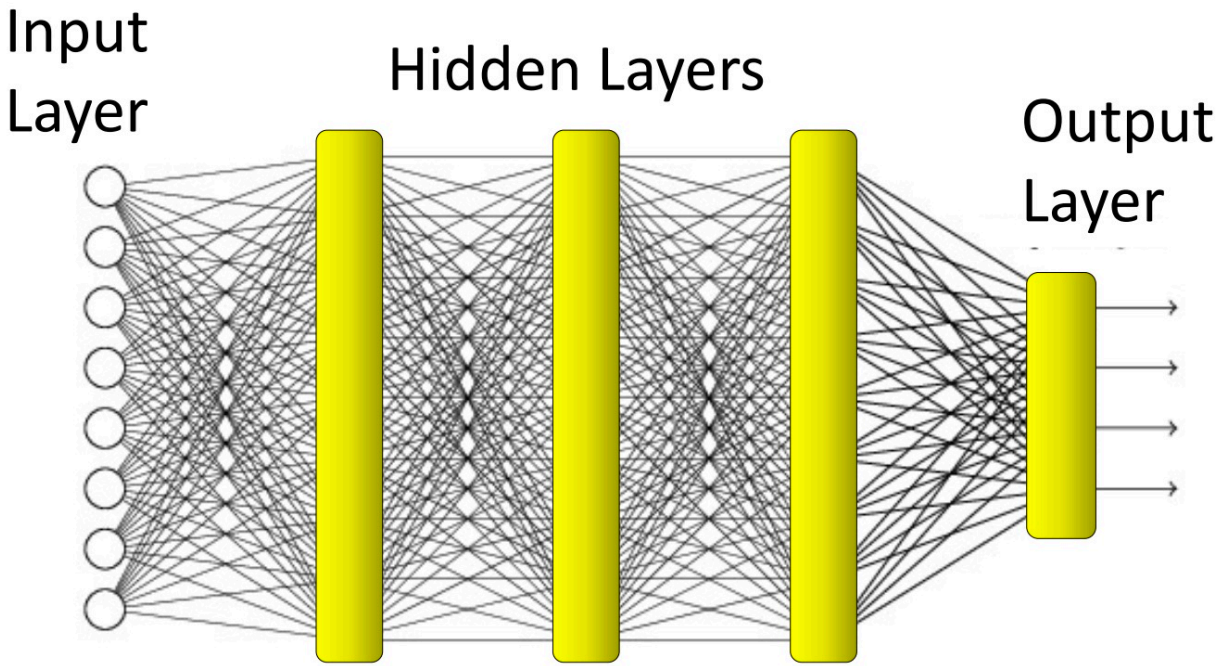

- Example: Softmax vector activation

$$z_i = \sum_j w_{ji} x_j + b_i$$

Parameters are weights $w_{ji}$ and bias $b_i$

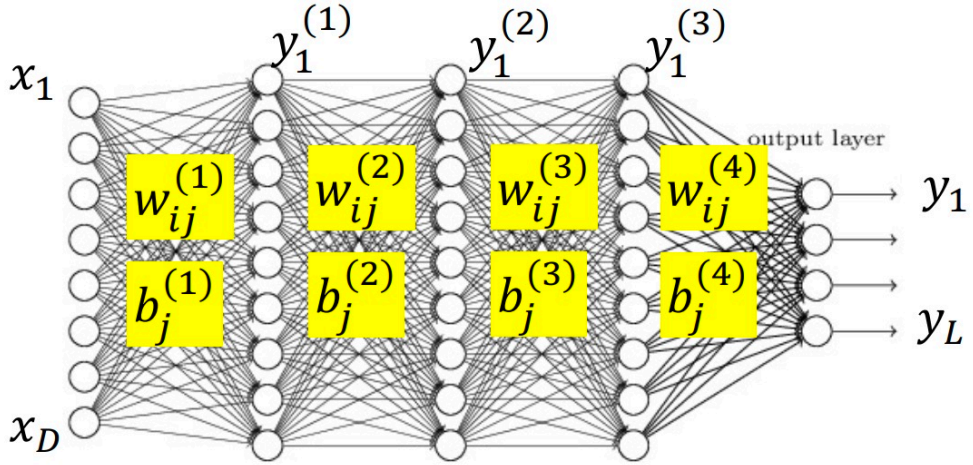
$$y = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

# Typical network



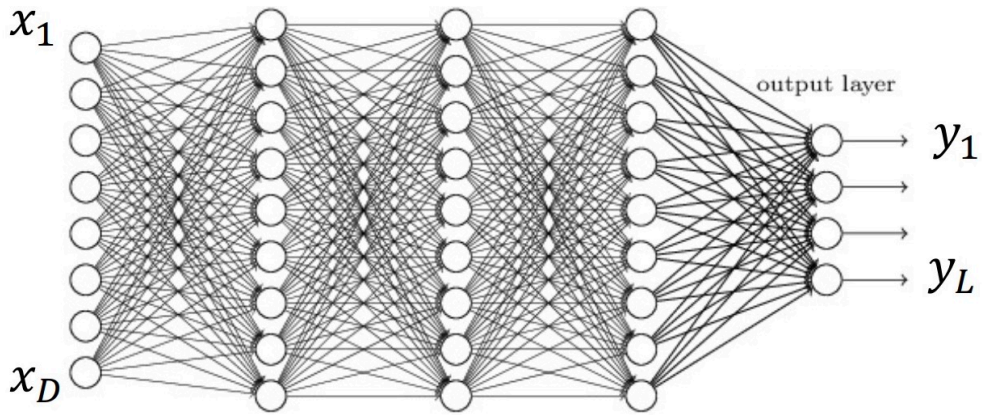

- In a layered network, each layer of perceptrons can be viewed as a single vector activation

# Notation



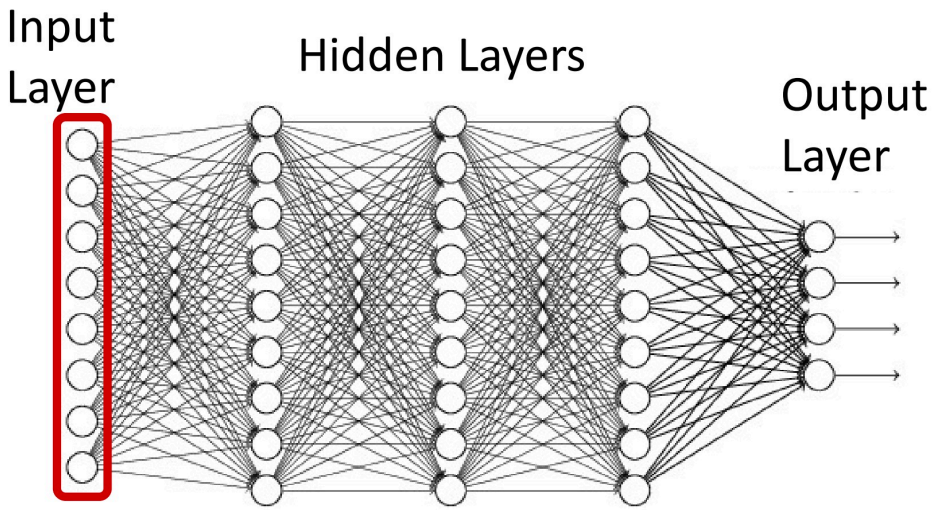

- The input layer is the 0th layer
- We will represent the output of the i-th perceptron of the kth layer as $y_i^{(k)}$
  - **Input to network:** $y_i^{(0)} = x_i$
  - **Output of network:** $y_i = y_i^{(N)}$
- We will represent the weight of the connection between the i-th unit of the k-1th layer and the jth unit of the k-th layer as $w_{ij}^{(k)}$
  - The bias to the jth unit of the k-th layer is $b_j^{(k)}$

# Input, target output, and actual output: Vector notation



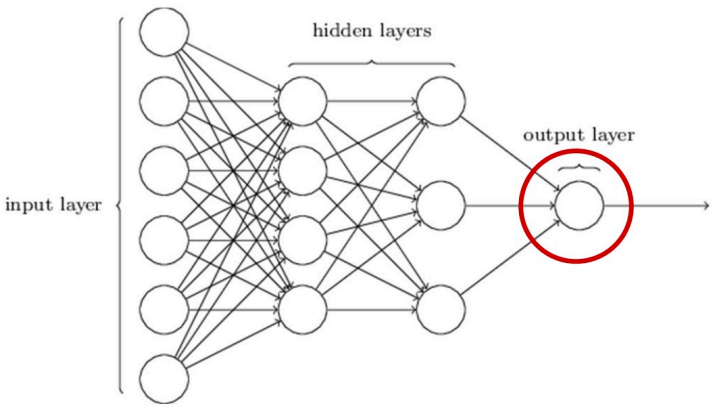

- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)$
- $X_n = [x_{n1}, x_{n2}, \ldots, x_{nD}]^\top$ is the nth input vector
- $d_n = [d_{n1}, d_{n2}, \ldots, d_{nL}]^\top$ is the nth desired output
- $Y_n = [y_{n1}, y_{n2}, \ldots, y_{nL}]^\top$ is the nth vector of *actual* outputs of the network
  - Function of input $X_n$ and network parameters

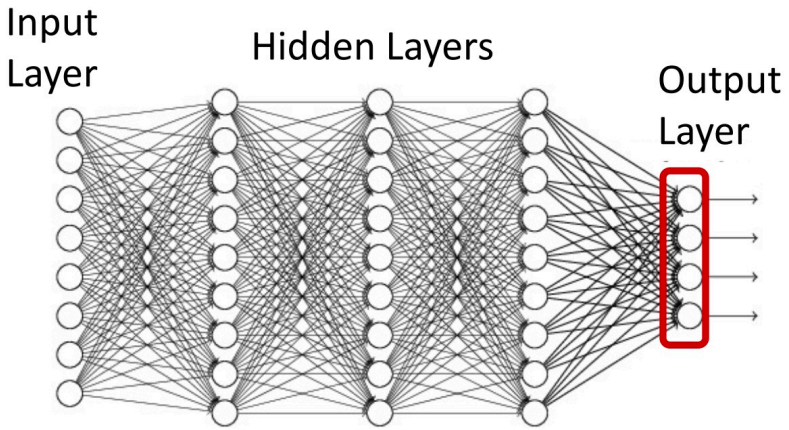
# Representing the input




- Vectors of numbers
  - (or may even be just a scalar, if input layer is of size 1)
  - E.g. vector of pixel values
  - E.g. vector of speech features
  - E.g. real-valued vector representing text
  - Other real valued vectors

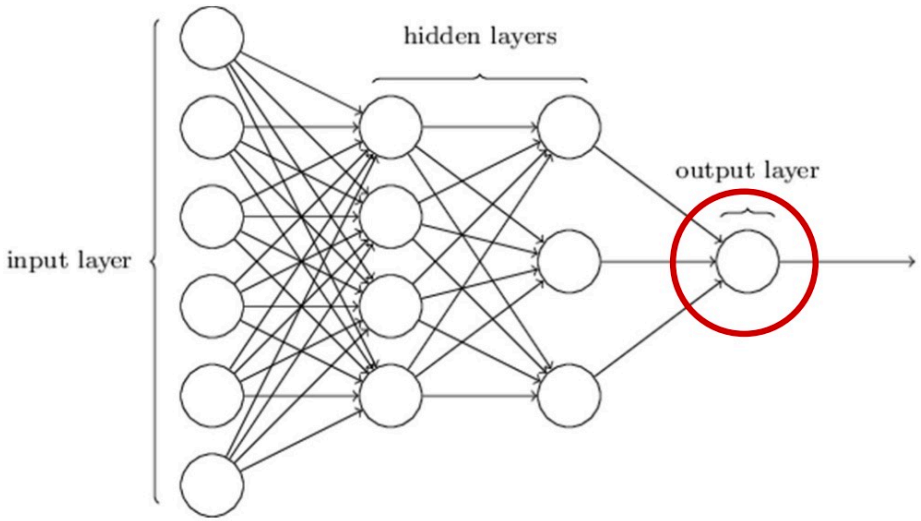
# Representing the output



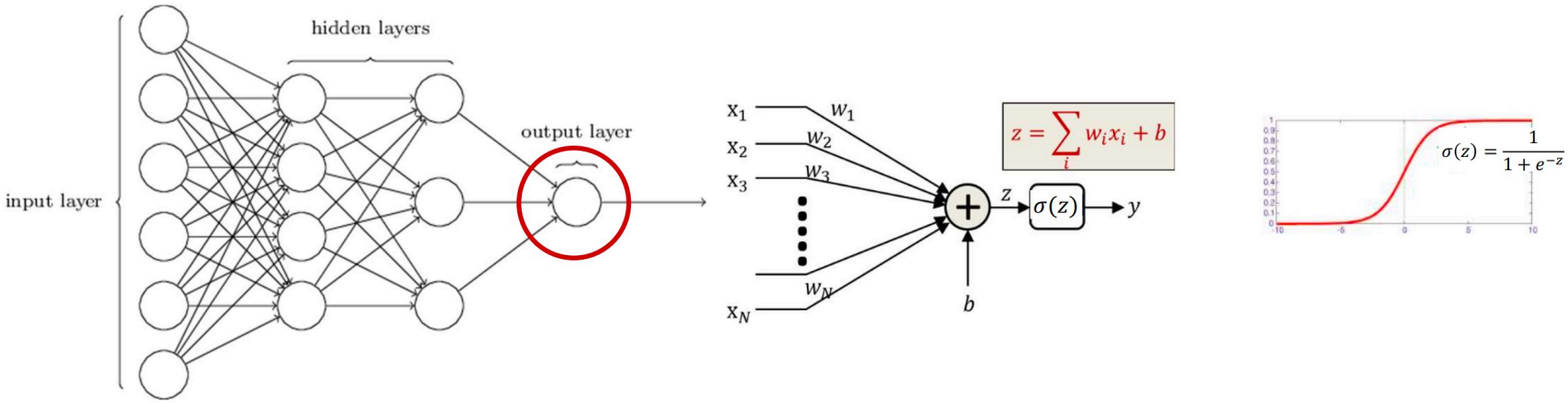

- If the desired output is real-valued, no special tricks are necessary
  - Scalar Output : single output neuron
    - $d =$ scalar (real value)
  - Vector Output : as many output neurons as the dimension of the desired output
    - $d = [d_1\, d_2, \ldots, d_L]$ (vector of real values)

# Representing the output



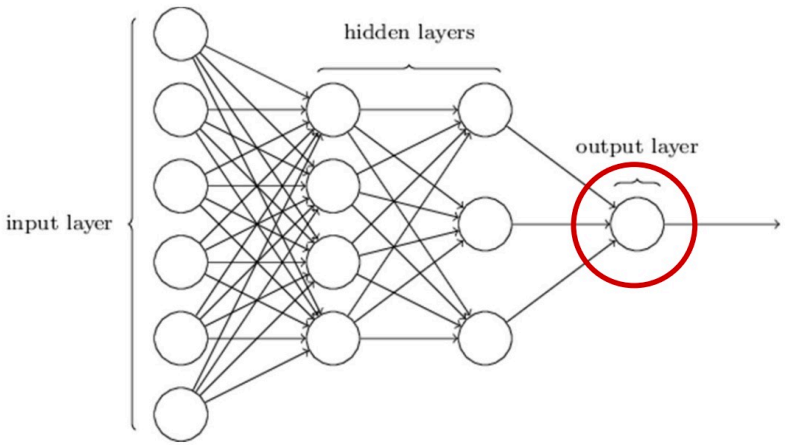

- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
  - 1 = Yes it's a cat
  - 0 = No it's not a cat.

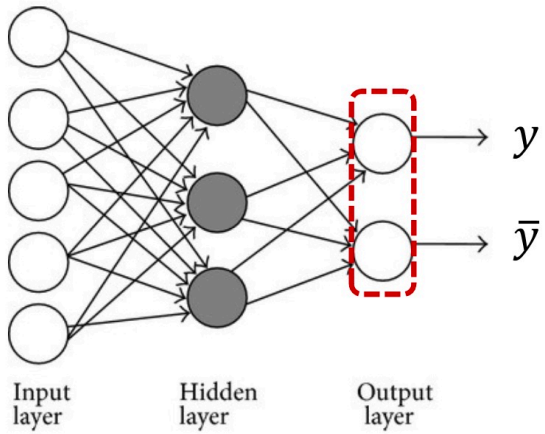
# Representing the output




- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
- Output activation: Typically a sigmoid
  - Viewed as the probability $P(Y = 1|X)$ of class value 1
    - Indicating that a feature value $X$ may occur for both classes, but with different probabilities
    - Is differentiable (可微)

# Representing the output



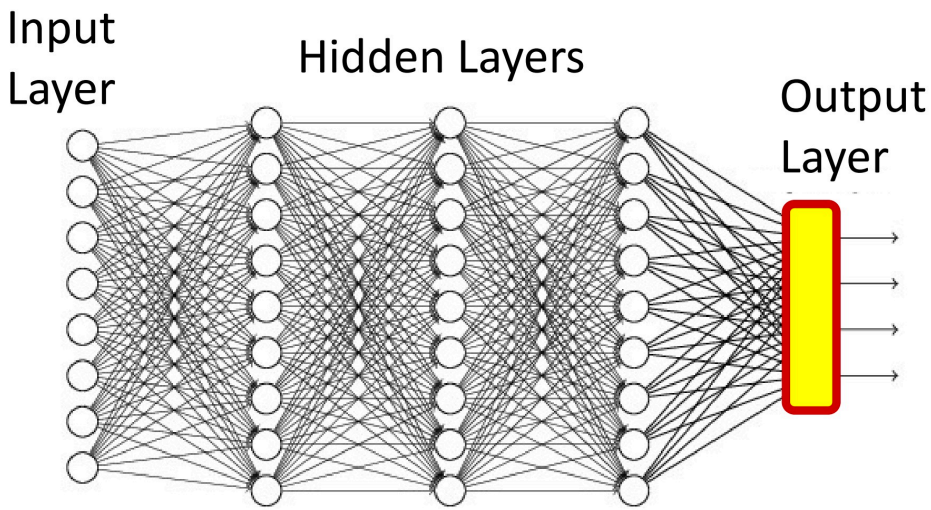




- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
  - 1 = Yes it's a cat
  - 0 = No it's not a cat.
- Sometimes represented by two outputs, one representing the desired output, the other representing the negation
  - Yes: $\rightarrow$ [1 0]
  - No: $\rightarrow$ [0 1]
- The output explicitly becomes a 2-output softmax

# Multi-class output: One-hot representations

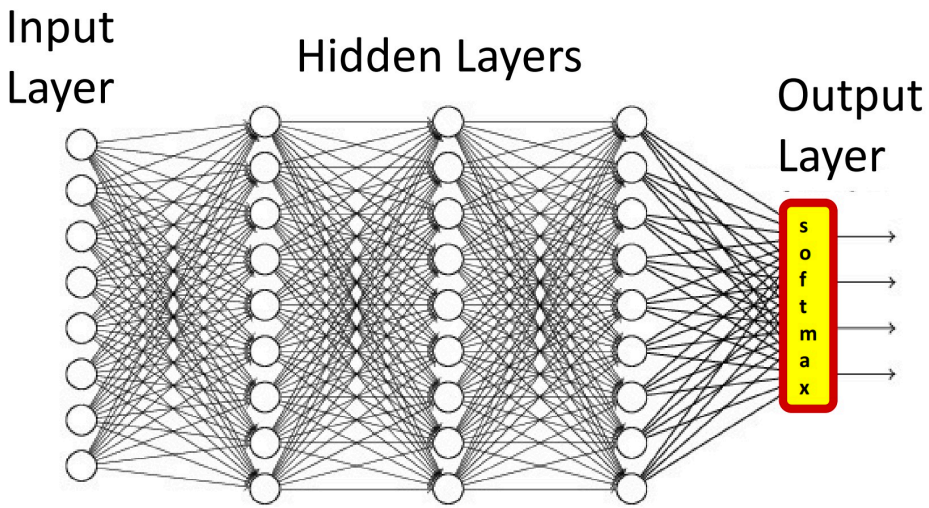
- Consider a network that must distinguish if an input is a cat, a dog, a camel, a hat, or a flower
- We can represent this set as the following vector, with the classes arranged in a chosen order:
  [cat dog camel hat flower]$^T$
- For inputs of each of the five classes the desired output is:
  cat: [1 0 0 0 0] $^T$
  dog: [0 1 0 0 0]$^T$
  camel: [0 0 1 0 0]$^T$
  hat: [0 0 0 1 0]$^T$
  flower: [0 0 0 0 1]$^T$
- For an input of any class, we will have a 5-dimensional vector output with 4 zeros and a single 1 at the position of that class
- This is a one hot vector (独热向量)

# Multi-class networks




- For a multi-class classifier with $N$ classes, the one-hot representation will have N binary target outputs
  - The desired output is an $N$-dimensional binary vector
- The neural network's actual output too must ideally be binary ($N$-1 zeros and a single 1 in the right place)
- More realistically, it will be a probability vector
  - $N$ probability values that sum to 1.

# Multi-class networks




- Softmax vector activation is often used at the output of multi-class classifier nets

$$z_i = \sum_j w_{ji}^{(n)} y_j^{(n-1)}$$

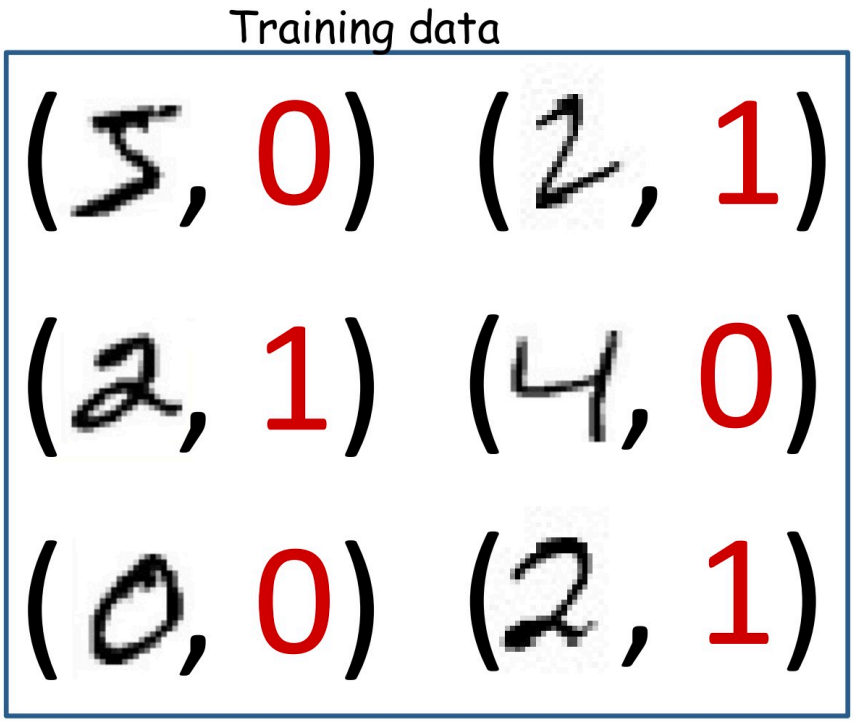
$$y_i = \frac{\exp(z_i)}{\sum_j \exp(z_i)}$$

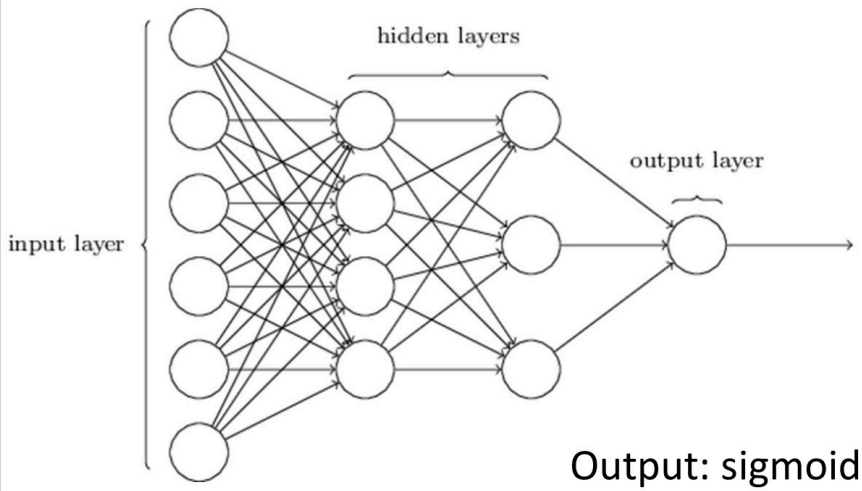
- This can be viewed as the probability $y_i = P(class = i|X)$

# Inputs and outputs: Typical Problem Statement

- We are given a number of "training" data instances
- E.g. images of digits, along with information about which digit the image represents
- Tasks:
  - Binary recognition: Is this a "2" or not
  - Multi-class recognition: Which digit is this?

# Typical Problem statement: binary classification



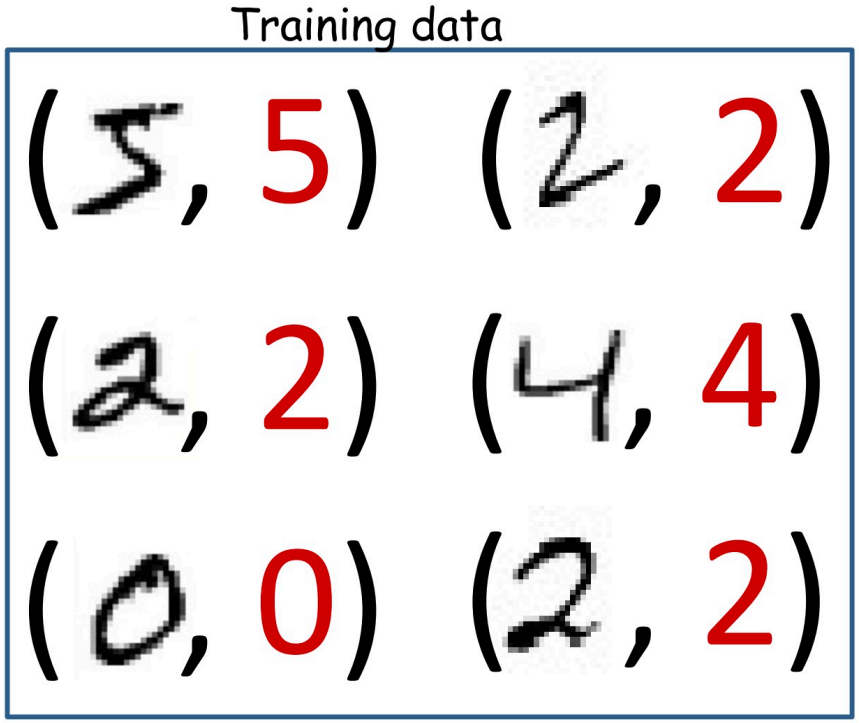



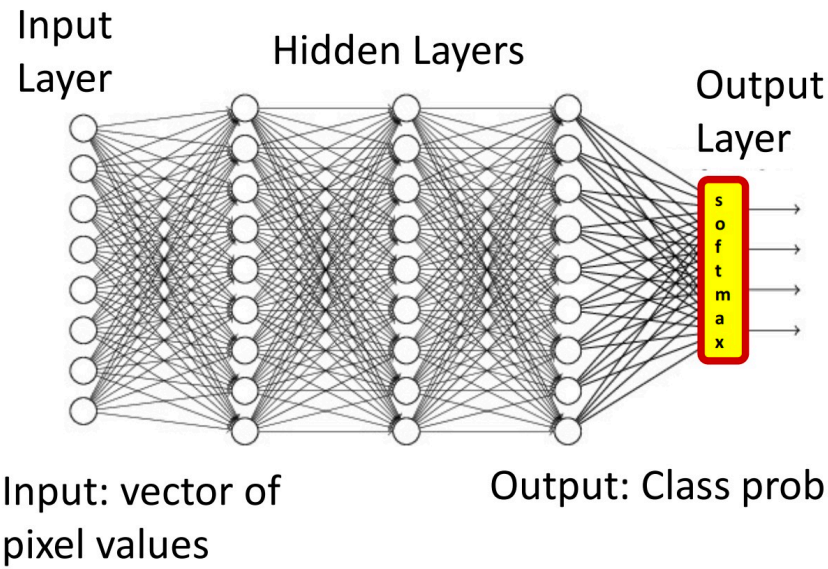

Output: sigmoid

Input: vector of pixel values

- Given, many positive and negative examples (training data),
  - learn all weights such that the network does the desired job

# Typical Problem statement: multiclass classification




- Given, many positive and negative examples (training data),
  - learn all weights such that the network does the desired job

# Problem Setup: Things to define

- Given a training set of input-output pairs $(X_1, d_1)$, $(X_2, d_d)$, ..., $(X_T, d_T)$
- Minimize the following function

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

What is the divergence div()?

# Problem Setup: Things to define

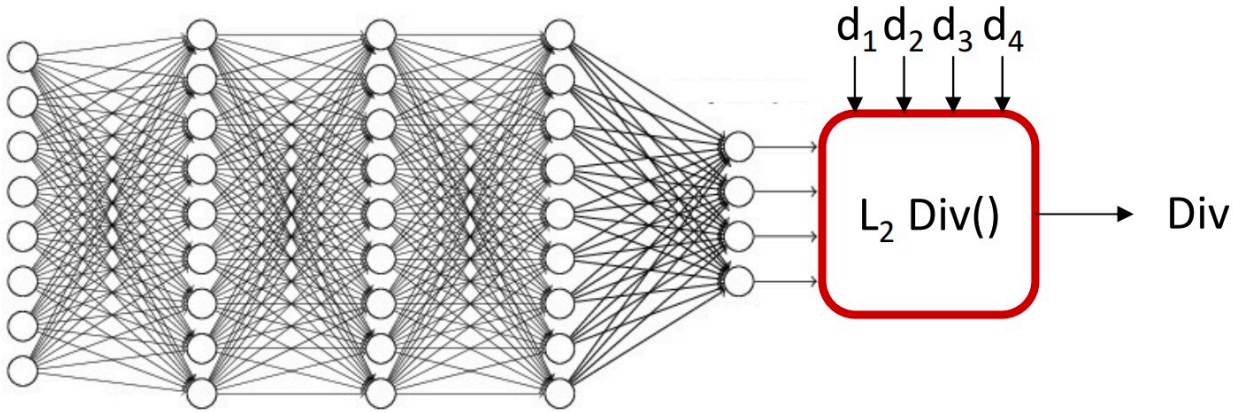
- Given a training set of input-output pairs $(X_1, d_1)$, $(X_2, d_d)$, ..., $(X_T, d_T)$
- Minimize the following function

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

What is the divergence div()?

Note: For Loss(W) to be differentiable w.r.t W, div() must be differentiable

# Examples of divergence function




- For real-valued output vectors, the L2 divergence is popular

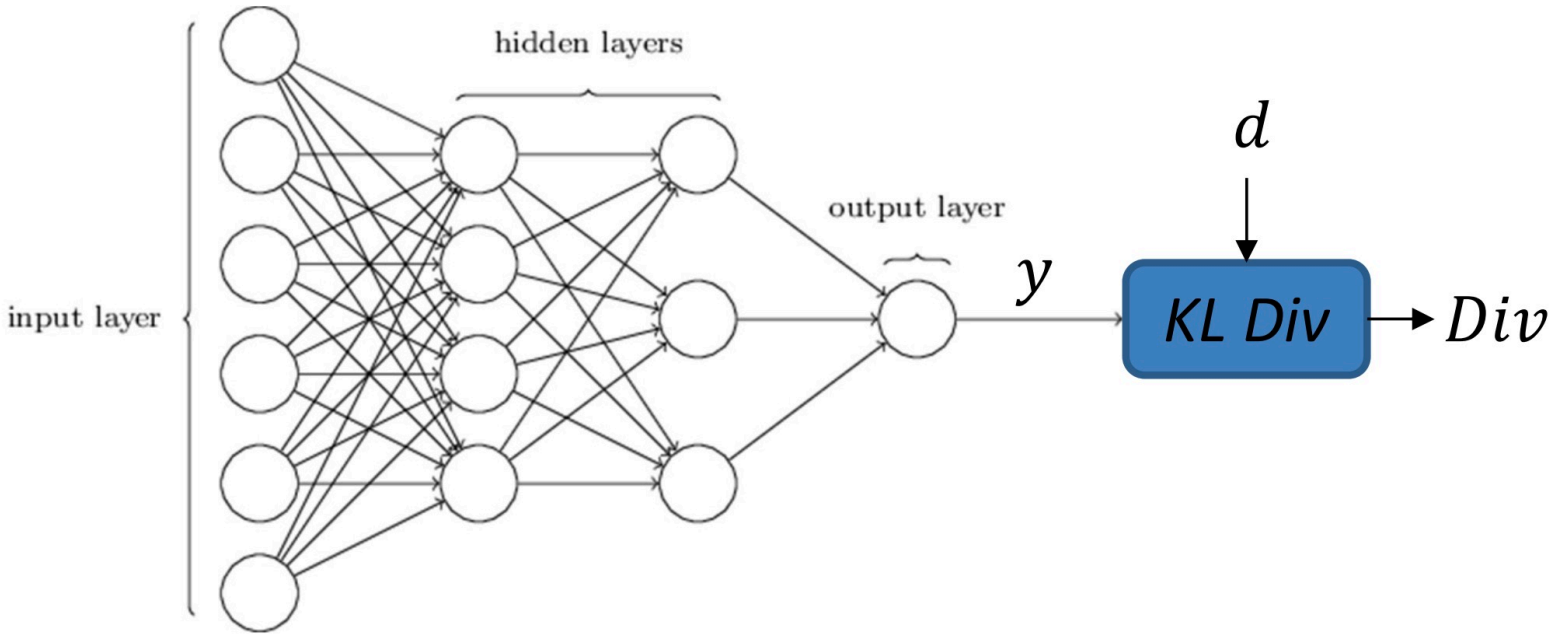
$$Div(Y,d) = \frac{1}{2}\|Y-d\|^2 = \frac{1}{2}\sum_i (y_i - d_i)^2$$

  - Squared Euclidean distance between true and desired output
  - Note: this is differentiable

$$\frac{dDiv(Y,d)}{dy_i} = (y_i - d_i)$$

$$\nabla_Y Div(Y,d) = [y_1 - d_1, y_2 - d_2, \ldots]$$

# For binary classifier



# For binary classifier

- For binary classifier with scalar output, $Y \in (0, 1)$, $d$ is 0/1, the Kullback Leibler (KL) divergence (KL 散度) between the probability distribution $[Y, 1 - Y]$ and the ideal output probability $[d, 1 - d]$ is popular

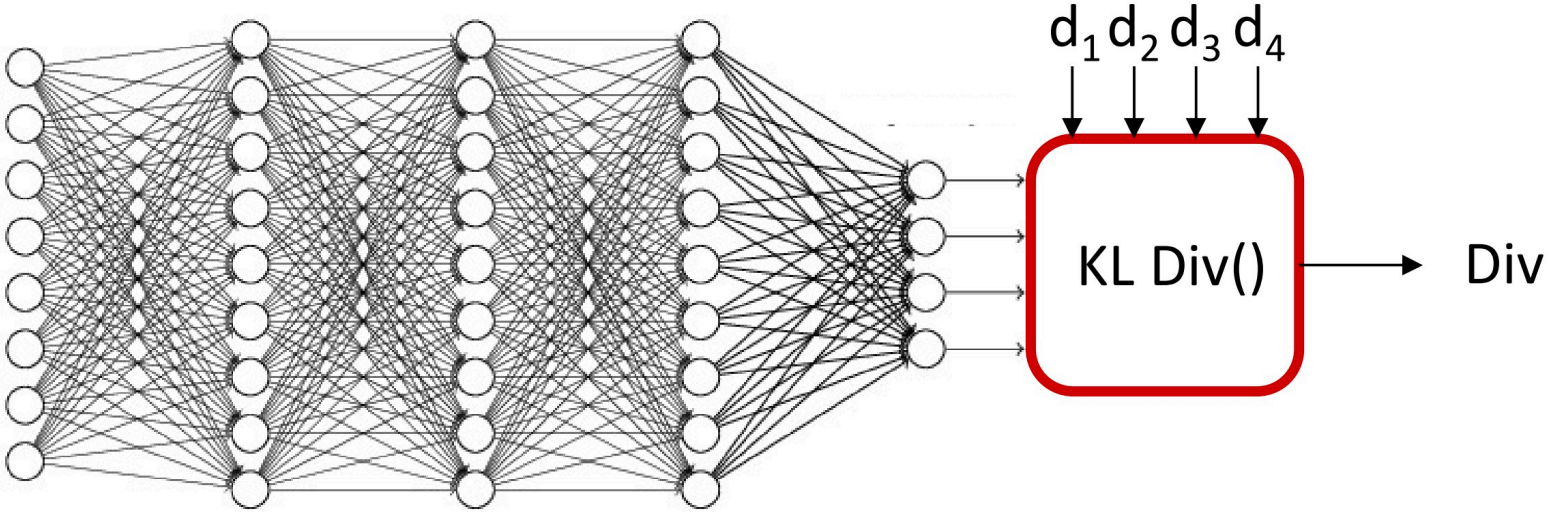
$$Div(Y, d) = -d \log Y - (1 - d) \log(1 - Y)$$

  - minimum when $d = Y$

- Derivative

$$\frac{dDiv(Y, d)}{dY} = \begin{cases} -\frac{1}{Y} & if\ d = 1 \\ \frac{1}{1 - Y} & if\ d = 0 \end{cases} \tag{5}$$

# For multi-class classification



# For multi-class classification

- Desired output $d$ is a one hot vector [0 0 ...1 ...0 0 0] with the 1 in the $c$-th position (for class $c$)
- Actual output will be probability distribution $[y_1, y_2, \cdots]$
- The KL divergence between the desired one-hot output and actual output:

$$Div(Y, d) = \sum_i d_i \log \frac{d_i}{y_i} = \sum_i d_i \log d_i - \sum_i d_i \log y_i = -\log y_c$$

# KL divergence vs cross entropy

- KL divergence between $d$ and $y$ :

$$KL(Y, d) = \sum_i d_i \log d_i - \sum_i d_i \log y_i$$

- Cross-entropy between $d$ and $y$ :

$$Xent(Y, d) = -\sum_i d_i \log y_i$$

- The cross entropy is merely the KL - entropy of $d$

$$Xent(Y, d) = KL(Y, d) - \sum_i d_i \log d_i = KL(Y, d) - H(d)$$

- The $W$ that minimizes cross-entropy will minimize the KL divergence
- We will generally minimize the *cross-entropy* loss rather than the KL divergence

# Problem Setup: Things to define

- Given a training set of input-output pairs $(X_1, d_1)$, $(X_2, d_d)$, ..., $(X_T, d_T)$
- Minimize the following function

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

ALL TERMS HAVE BEEN DEFINED

Thank you!