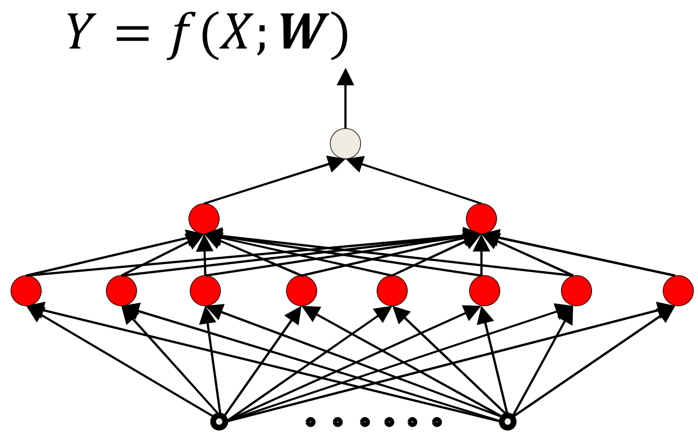
# Neural Networks
# Learning the network: Backprop

## 11-785, Fall 2023

## Lecture 4

# Recap: Empirical Risk Minimization



- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
  – Divergence on the i-th instance: $div(f(X_i; W), d_i)$
  – Empirical average divergence on all training data:

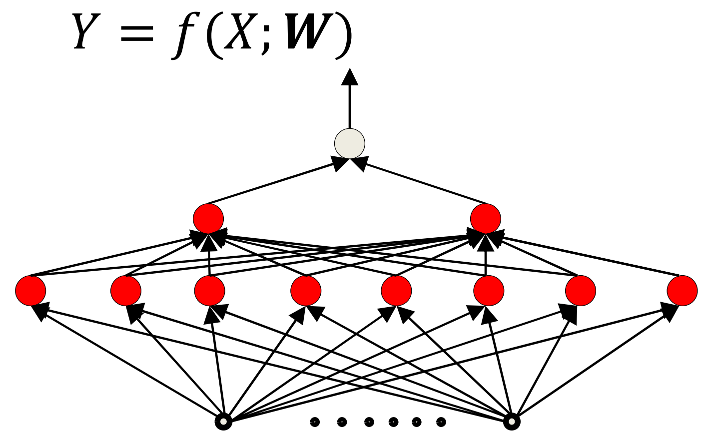
$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

- Estimate the parameters to minimize the empirical estimate of expected divergence

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

  – I.e. minimize the *empirical risk* over the drawn samples

# Recap: Empirical Risk Minimization




This is an instance of function minimization (optimization)

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \dots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
  - Error on the i-th instance: $div(f(X_i; W), d_i)$
  - Empirical average error on all training data:

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

- Estimate the parameters to minimize the empirical estimate of expected error

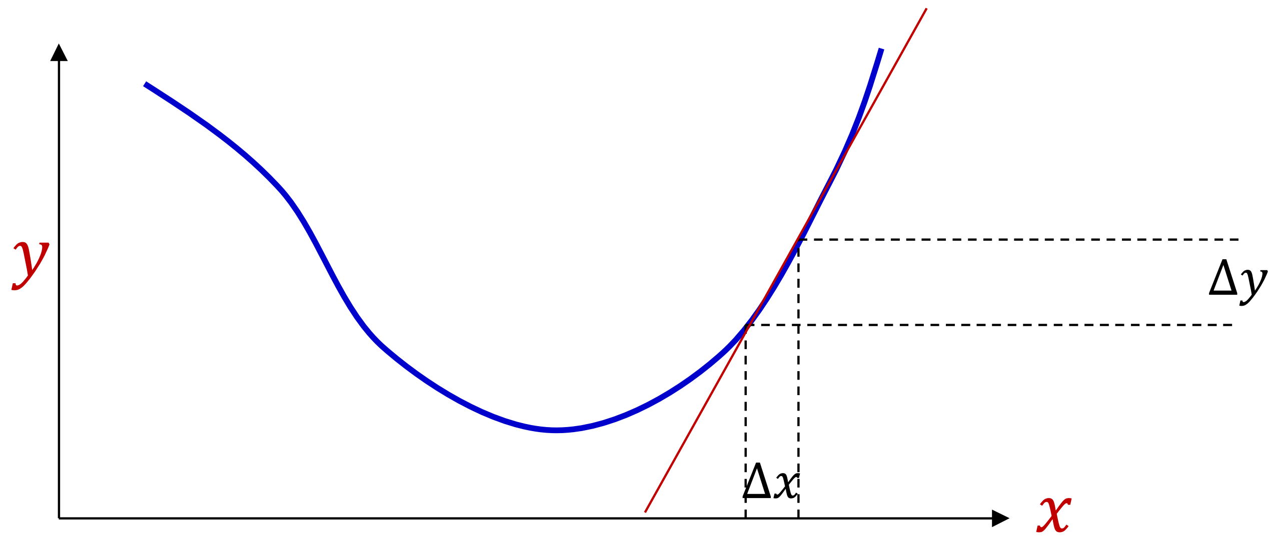
$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}}\, Loss(W)$$

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

# A quick intro to function optimization

with an initial discussion of derivatives

# A brief note on derivatives..



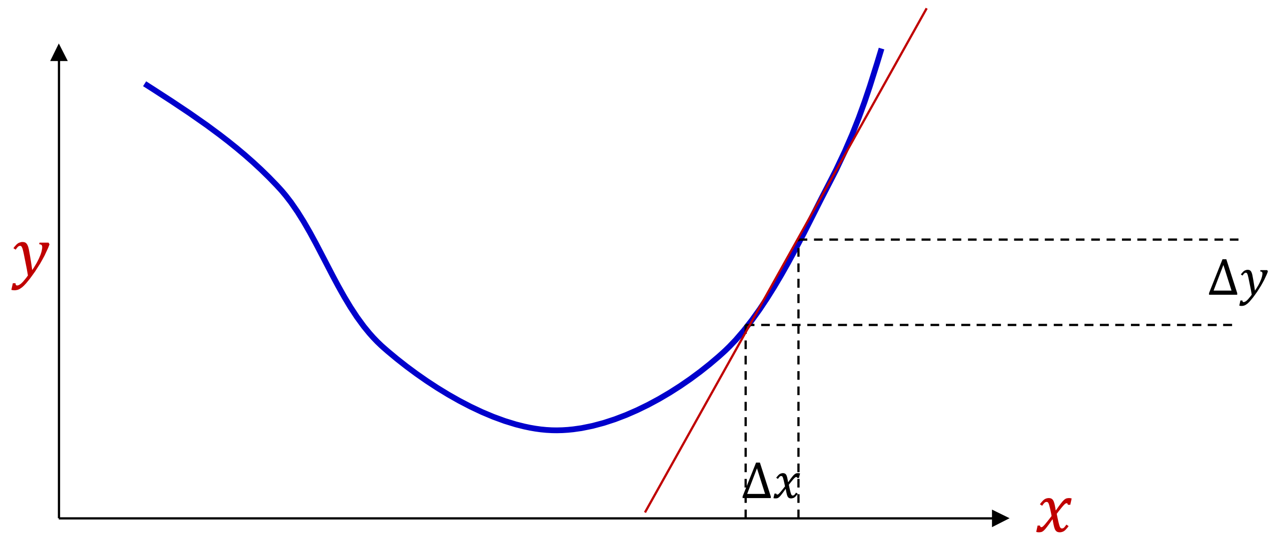

derivative

- 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

$$\Delta y = \alpha \Delta x$$

  - Based on the fact that at a fine enough resolution, any smooth, continuous function is locally linear at any point

# Scalar function of scalar argument




- When $x$ and $y$ are scalar

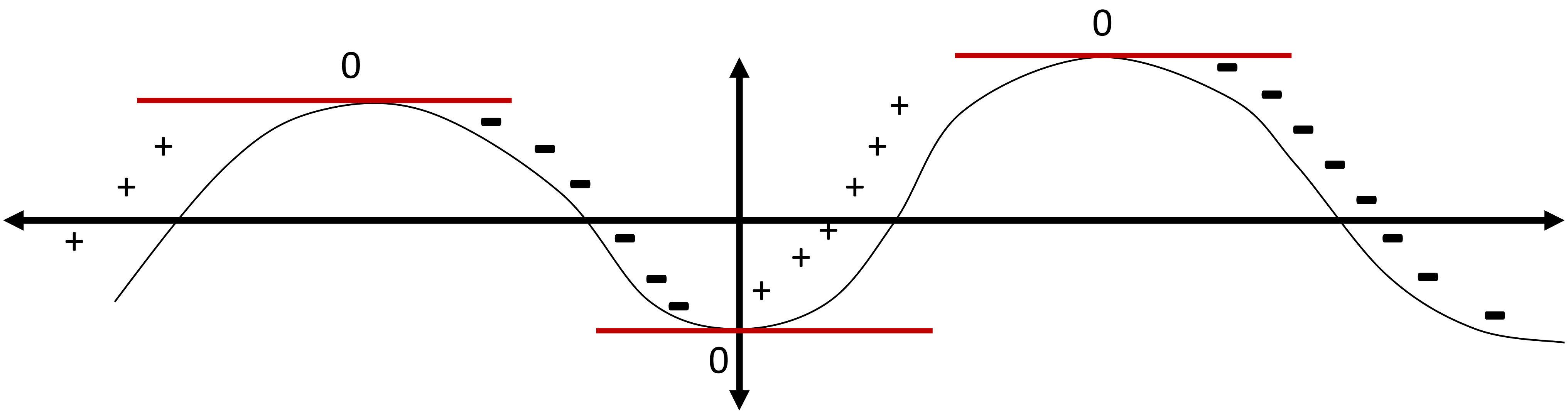
$$y = f(x)$$

  - Derivative:

$$\Delta y = \alpha \Delta x$$

  - Often represented (using somewhat inaccurate notation) 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 $x$
  - How fast it increases with increasing $x$
  - The magnitude of f'(x) gives you the steepness of the curve at x
    - Larger |f'(x)| → the function is increasing or decreasing more rapidly

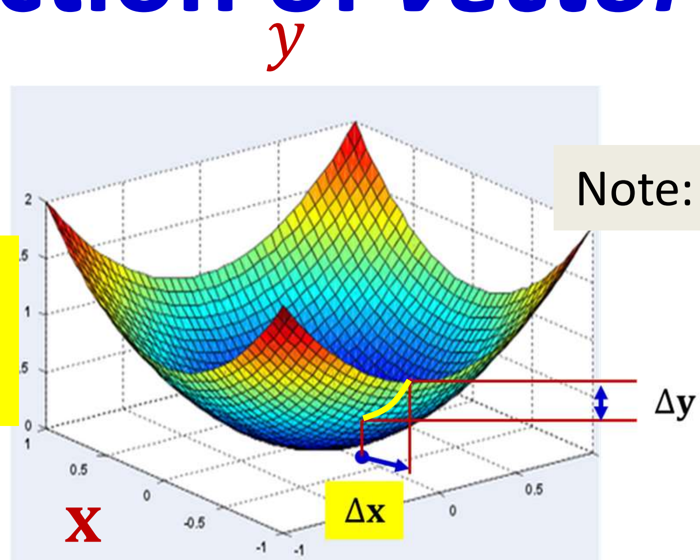
- It will be positive where a small increase in x results in an *increase* of f(x)
  - Regions of positive slope
- It will be negative where a small increase in x results in a *decrease* of f(x)
  - Regions of negative slope

- 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

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

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

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

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

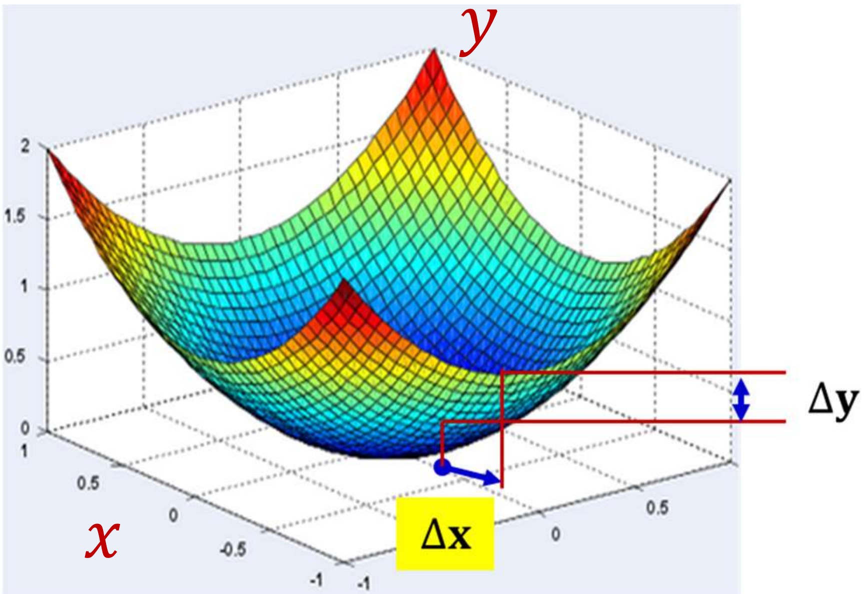
- The *partial* derivative $\alpha_i$ gives us how $y$ increments when *only* $x_i$ is incremented

- Often represented as $\frac{\partial y}{\partial x_i}$

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

# Multivariate scalar function:

# Scalar function of *vector* argument




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

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

$$\Delta y = \nabla_{\mathbf{x}} y \Delta\mathbf{x}$$

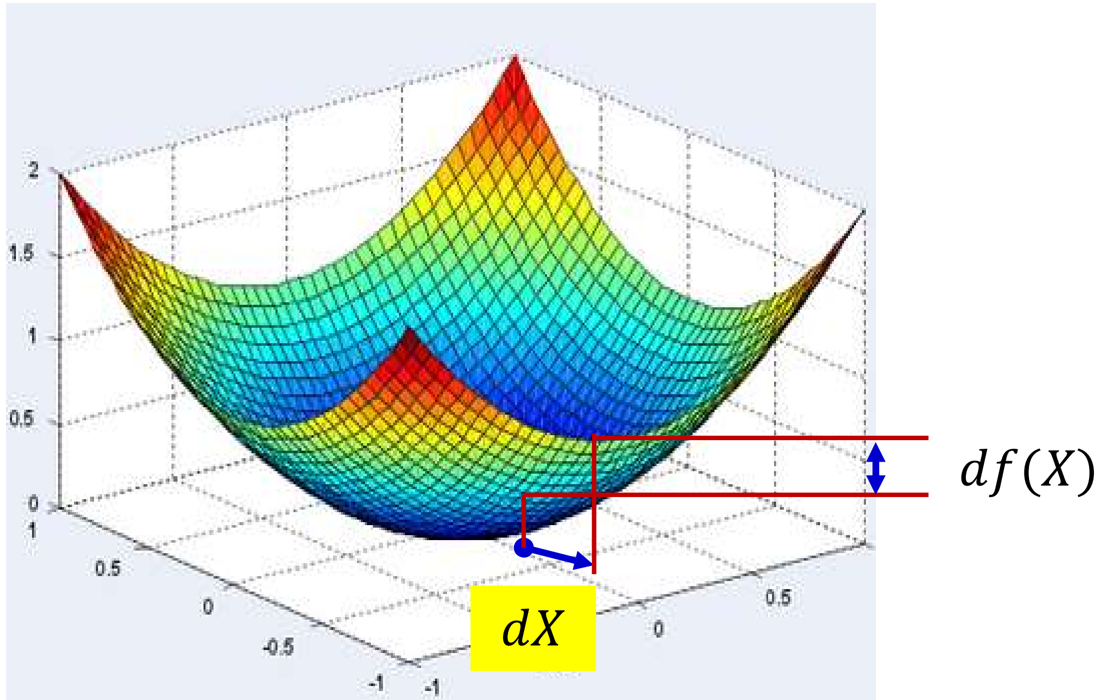
We will be using this symbol for vector and matrix derivatives

- Where

$$\nabla_{\mathbf{x}} y = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \cdots & \frac{\partial y}{\partial x_D} \end{bmatrix}$$

  - You may be more familiar with the term "gradient" which is actually defined as the transpose of the derivative

# *Gradient* of a scalar function of a vector



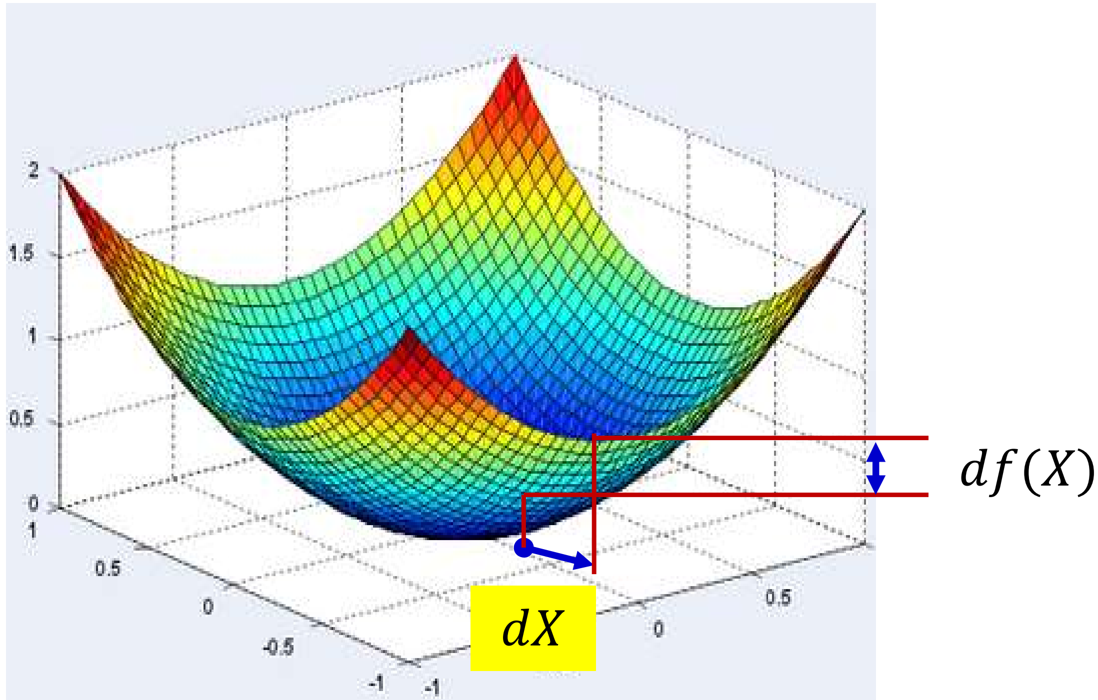

- The *derivative* $\nabla_X f(X)$ of a scalar function $f(X)$ of a multi-variate input $X$ is a multiplicative factor that gives us the change in $f(X)$ for tiny variations in $X$

$$df(X) = \nabla_X f(X) dX$$

  – $\nabla_X f(X) = \left[ \frac{\partial f(X)}{\partial x_1} \quad \frac{\partial f(X)}{\partial x_2} \quad \cdots \quad \frac{\partial f(X)}{\partial x_n} \right]$

- The ***gradient*** is the transpose of the derivative $\nabla_X f(X)^T$
  – A column vector of the same dimensionality as $X$

# *Gradient* of a scalar function of a vector



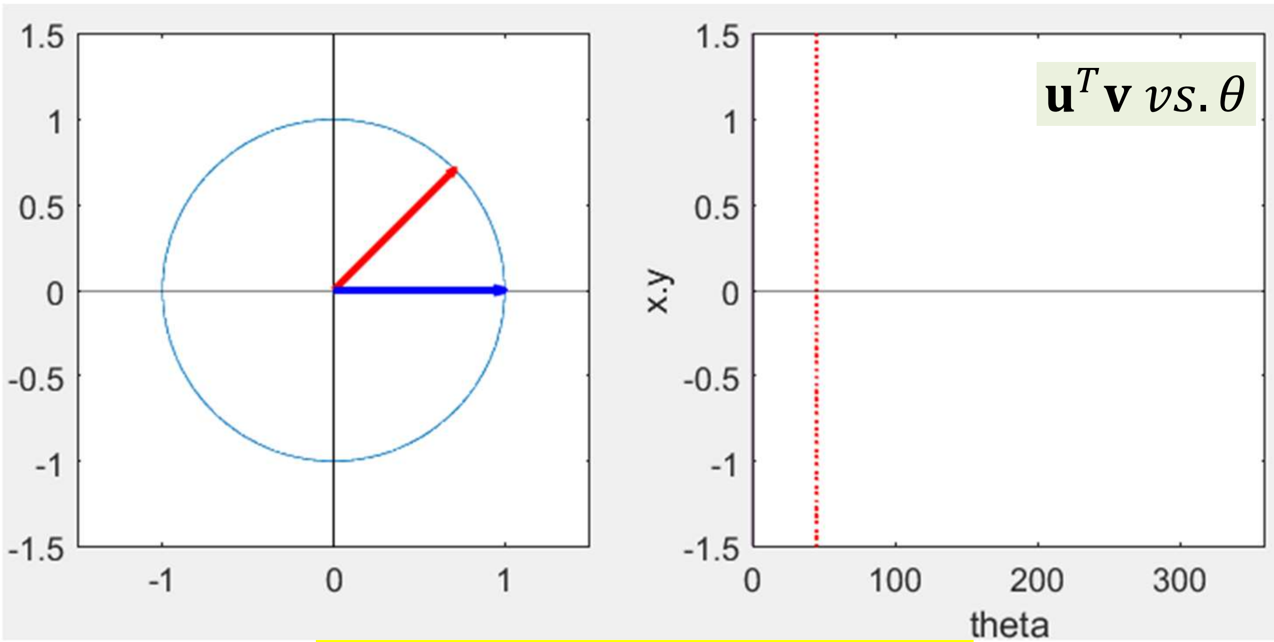

- The *derivative* $\nabla_X f(X)$ of a scalar function $f(X)$ of a multi-variate input $X$ is a multiplicative factor that gives us the change in $f(X)$ for tiny variations in $X$

$$df(X) = \nabla_X f(X) dX$$

  - $\nabla_X f(X) = \left[\frac{\partial f(X)}{\partial x_1} \quad \frac{\partial f(X)}{\partial x_2} \quad \cdots \quad \frac{\partial f(X)}{\partial x_n}\right]$

This is a vector inner product. To understand its behavior lets consider a well-known property of inner products

# A well-known vector property

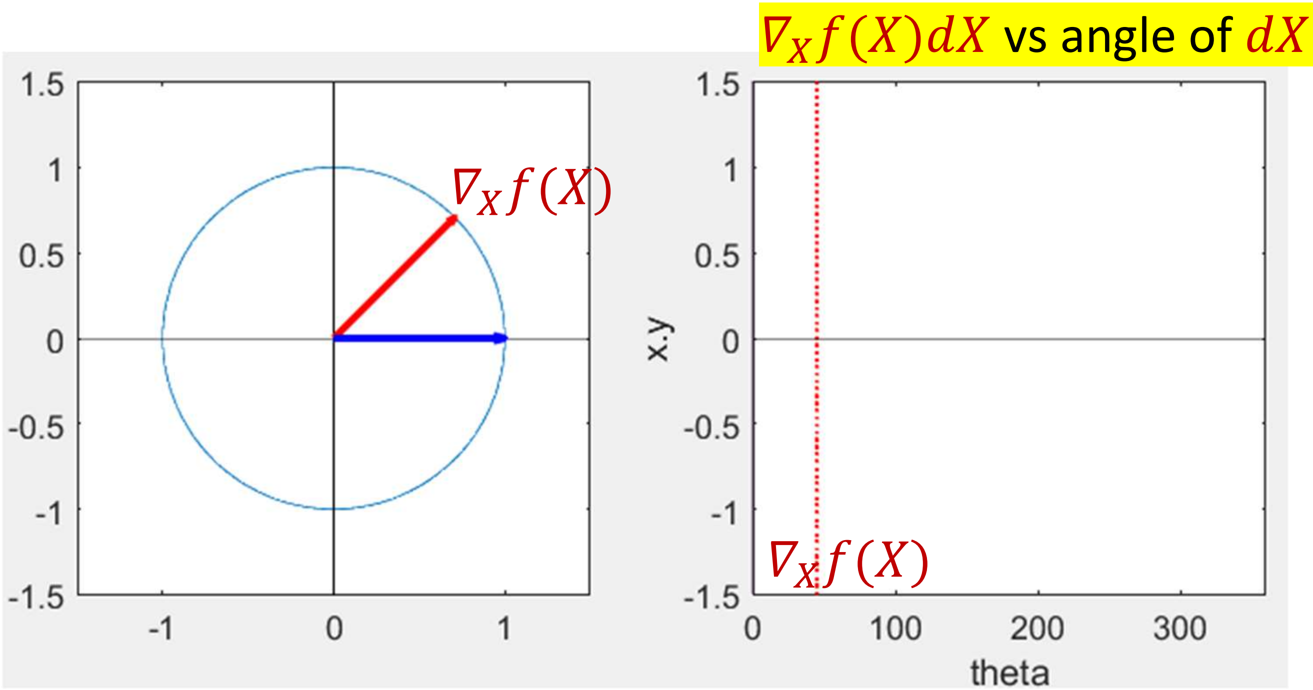



$$\mathbf{u}^T\mathbf{v} = |\mathbf{u}||\mathbf{v}|\cos\theta$$

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

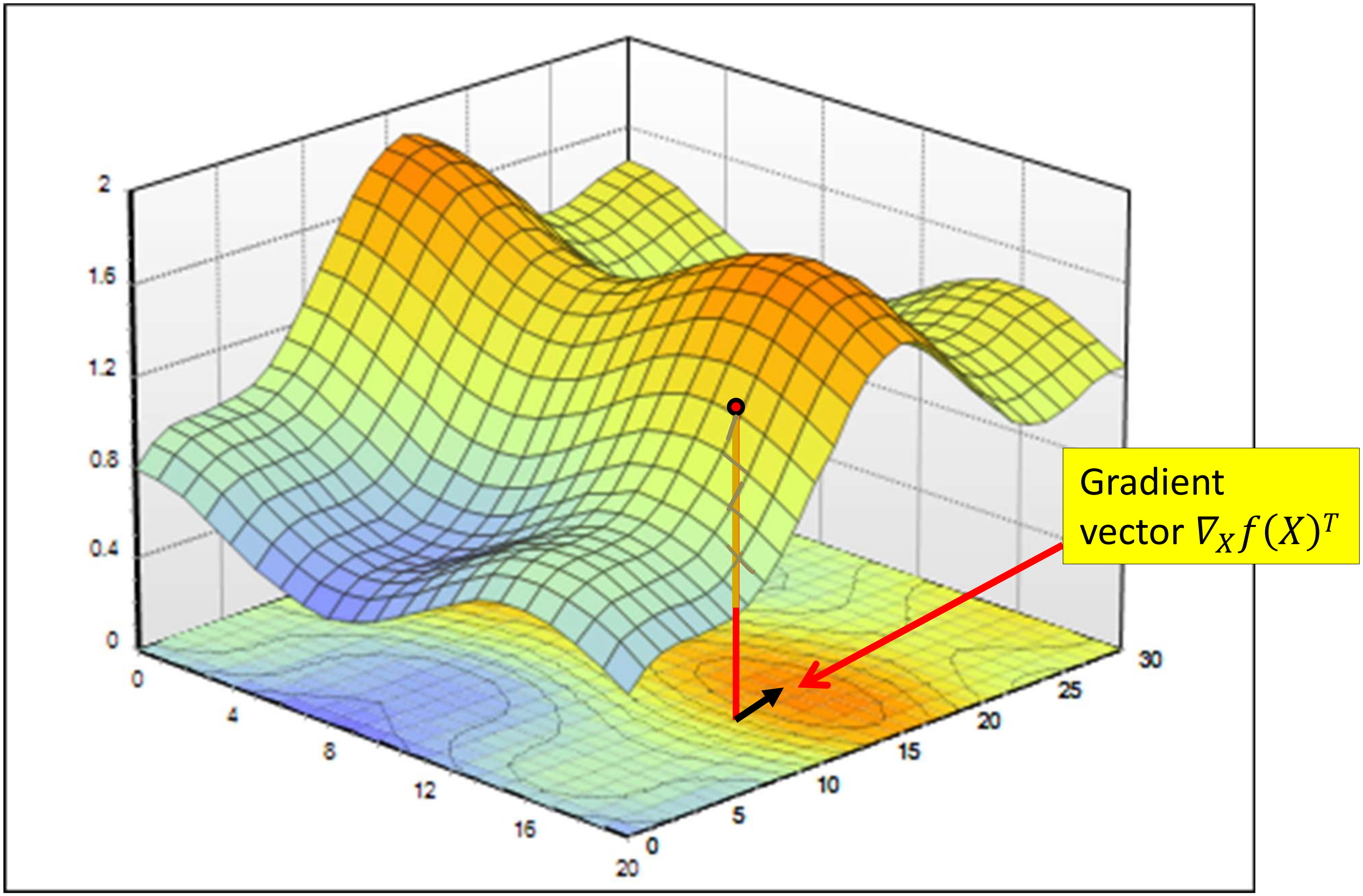
# Properties of Gradient

Blue arrow is $dX$

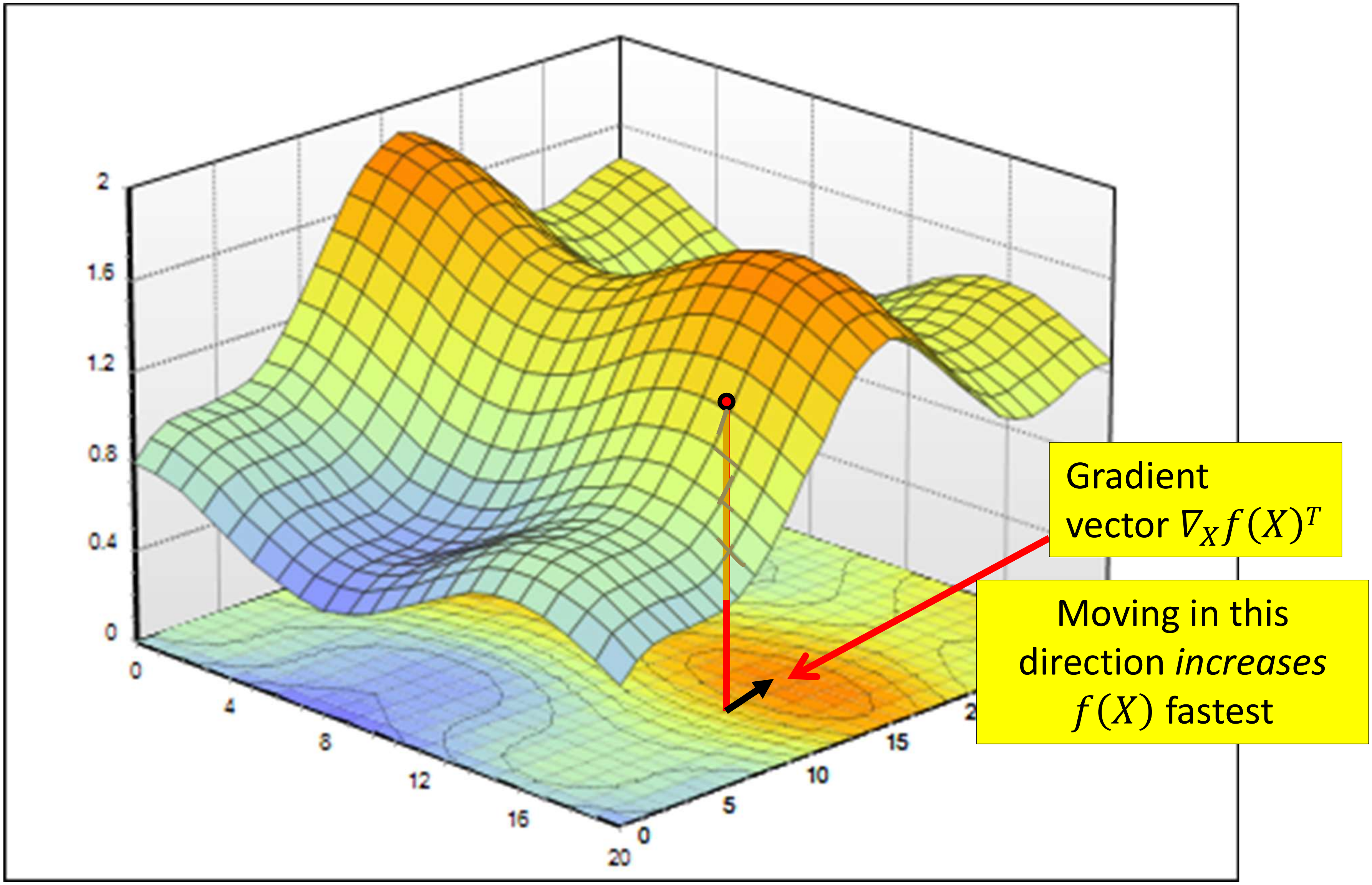



- $df(X) = \nabla_X f(X) dX$
- For an increment $dX$ of any given length $df(X)$ is max if $dX$ is aligned with $\nabla_X f(X)^{\mathrm{T}}$
  - The function $f(X)$ increases most rapidly if the input increment $dX$ is exactly in the direction of $\nabla_X f(X)^{\mathrm{T}}$
- The gradient is the direction of fastest increase in $f(X)$

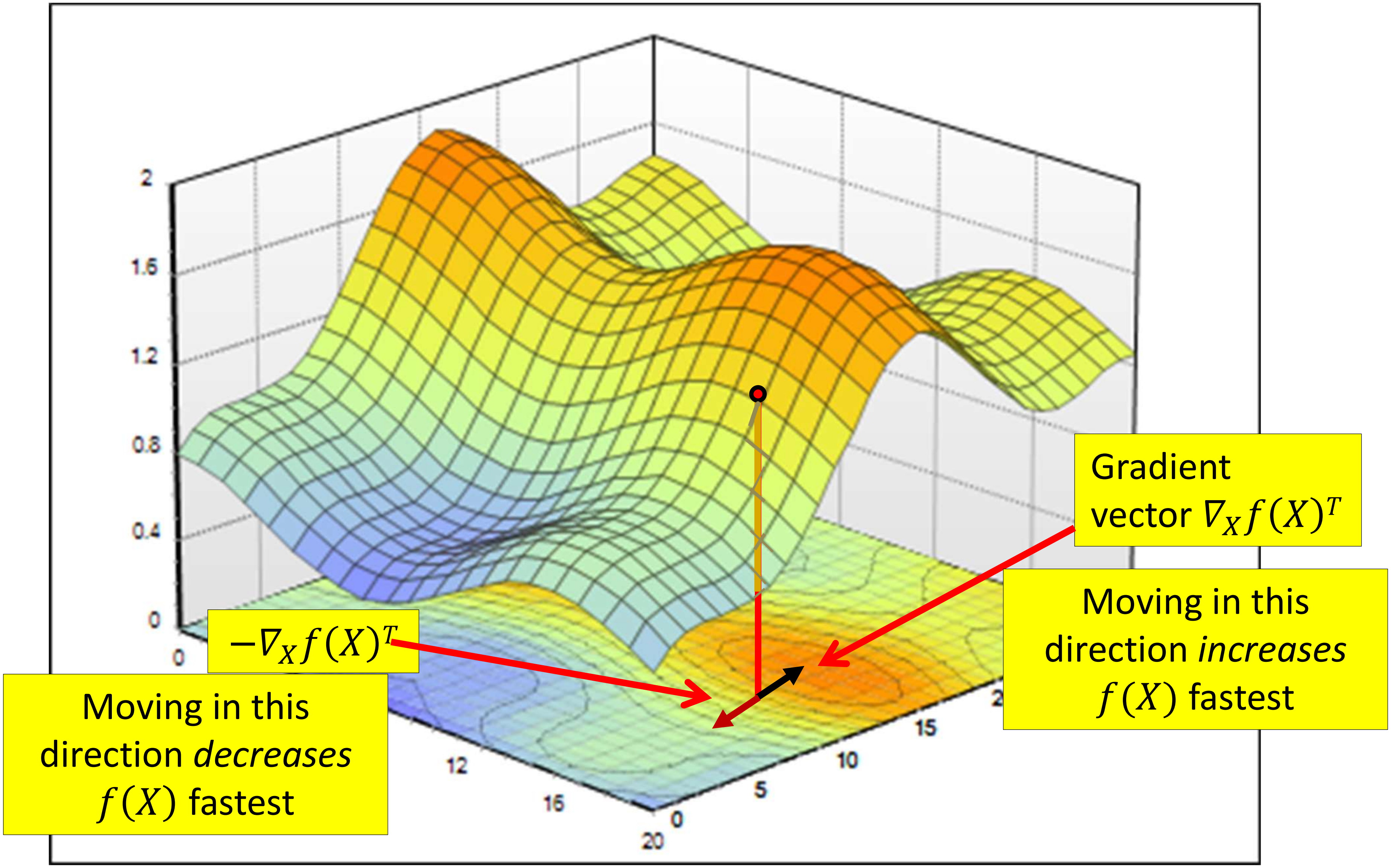
# Gradient


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

# Gradient

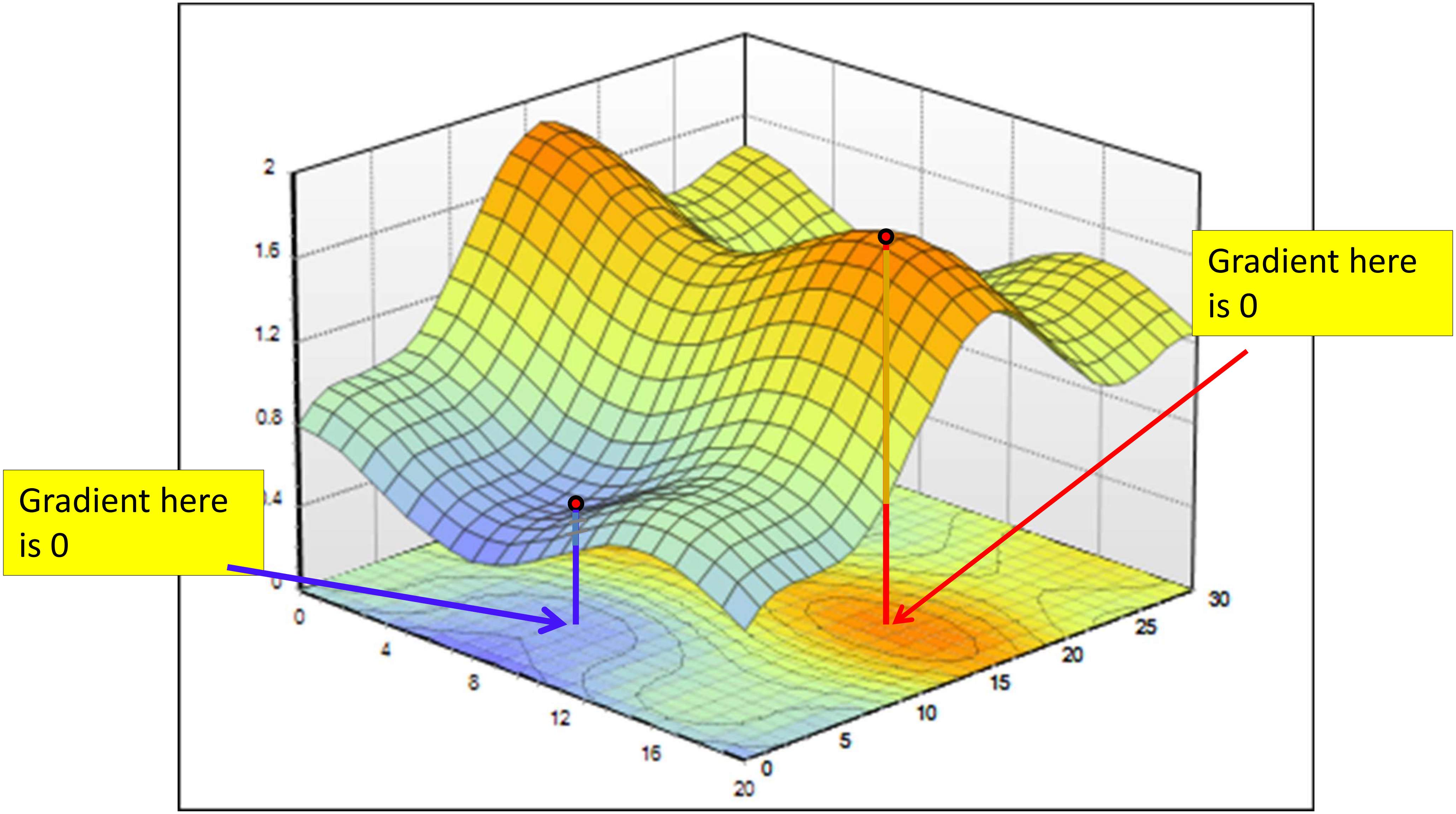

2
1.6
1.2
0.8
0.4
0
0
4
8
12
16
20
0
5
10
15
Gradient vector $\nabla_X f(X)^T$
Moving in this direction *increases* $f(X)$ fastest

# Gradient

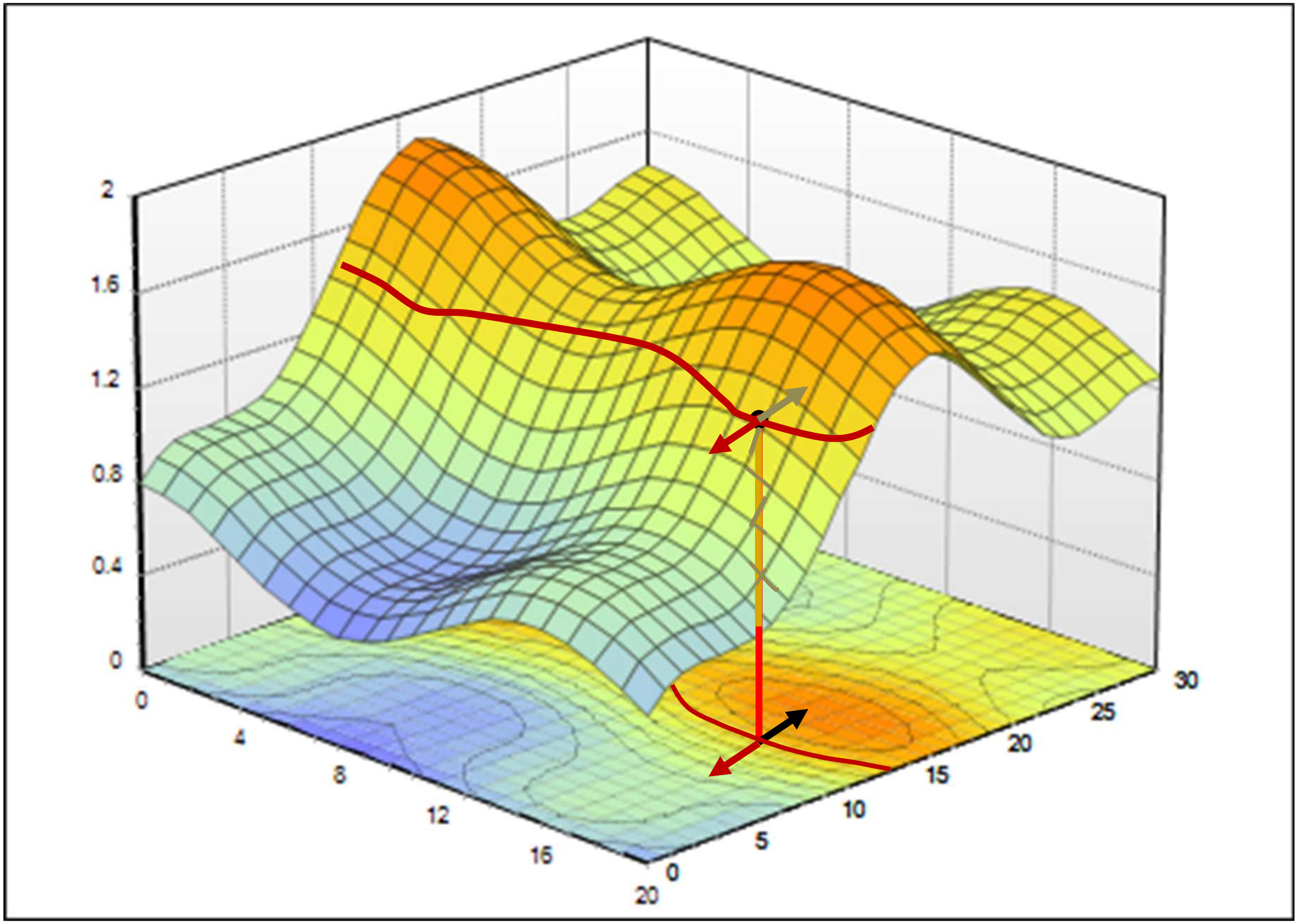


2
1.6
1.2
0.8
0.4
0
0
12
16
20
0
5
10
15
Gradient vector $\nabla_X f(X)^T$
Moving in this direction *increases* $f(X)$ fastest
$-\nabla_X f(X)^T$
Moving in this direction *decreases* $f(X)$ fastest

# Gradient


2
1.6
1.2
0.8
0.4
0
0
4
8
12
16
20
0
5
10
15
20
25
30
Gradient here is 0
Gradient here is 0

# Properties of Gradient: 2




- The gradient vector $\nabla_X f(X)^T$ is perpendicular to the level curve

# The Hessian

- The Hessian of a function $f(x_1, x_2, \ldots, x_n)$ is given by the second derivative

$$\nabla_X^2 f(x_1, \ldots, x_n) := \begin{bmatrix} \frac{\partial^2 f}{\partial x_1{}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdot & \cdot & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2{}^2} & \cdot & \cdot & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdot & \cdot & \frac{\partial^2 f}{\partial x_n{}^2} \end{bmatrix}$$

# Poll 1

- Select all that are true about derivatives of a scalar function f(X) of multivariate inputs
  - At any location X, there may be many directions in which we can step, such that f(X) increases
  - The direction of the gradient is the direction in which the function increases fastest
  - The gradient is the derivative of f(X) w.r.t. X

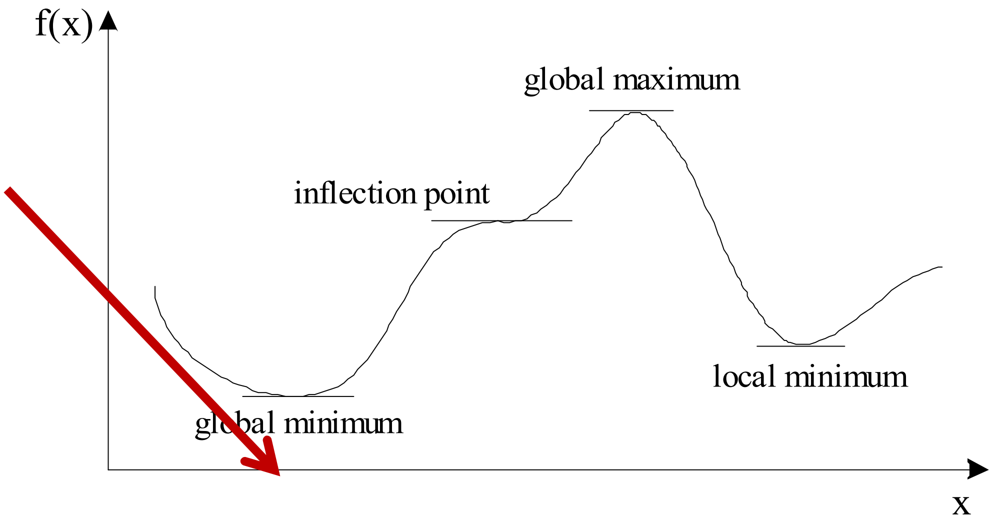
- y = f(x) is a scalar function of an Nx1 column vector variable x. What is the shape of the derivative of y with respect to x
  - Scalar
  - N x 1 column vector
  - 1 x N row vector
  - There is insufficient information to decide

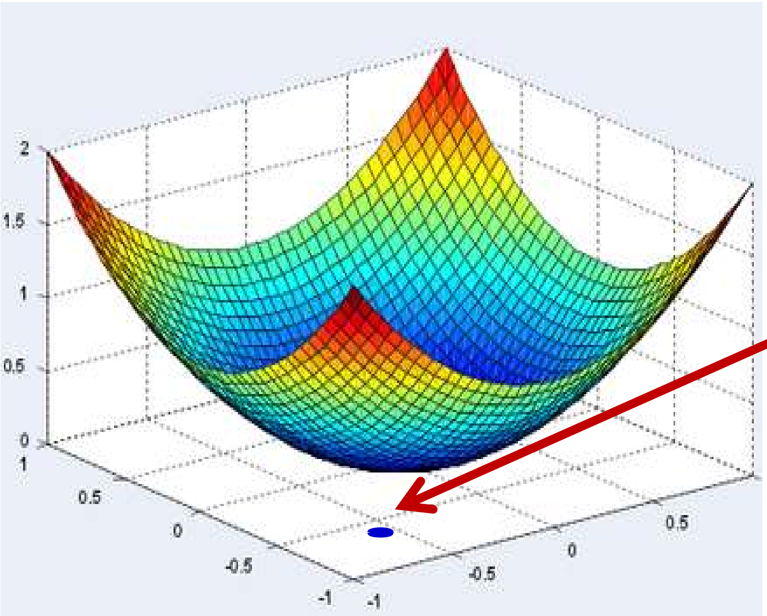
# Poll 1

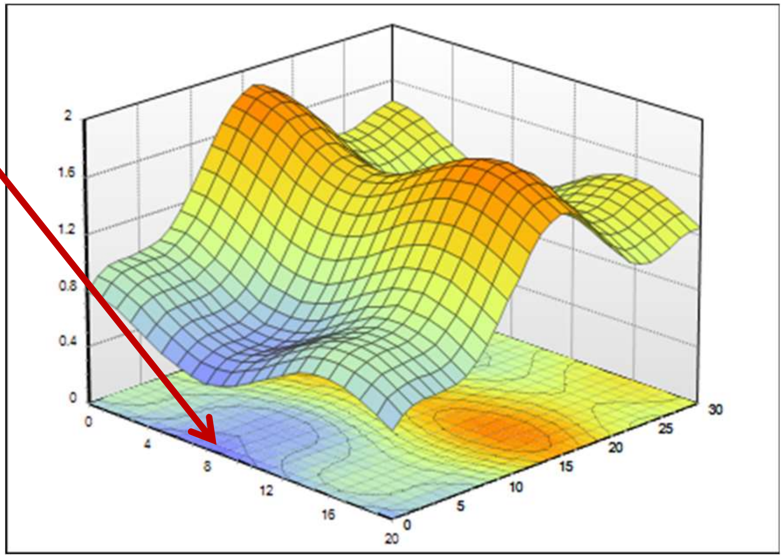
- Select all that are true about derivatives of a scalar function f(X) of multivariate inputs
  - **At any location X, there may be many directions in which we can step, such that f(X) increases**
  - **The direction of the gradient is the direction in which the function increases fastest**
  - The gradient is the derivative of f(X) w.r.t. X

- y = f(x) is a scalar function of an Nx1 column vector variable x. What is the shape of the derivative of y with respect to x
  - Scalar
  - N x 1 column vector
  - **1 x N row vector**
  - There is insufficient information to decide

# The problem of optimization

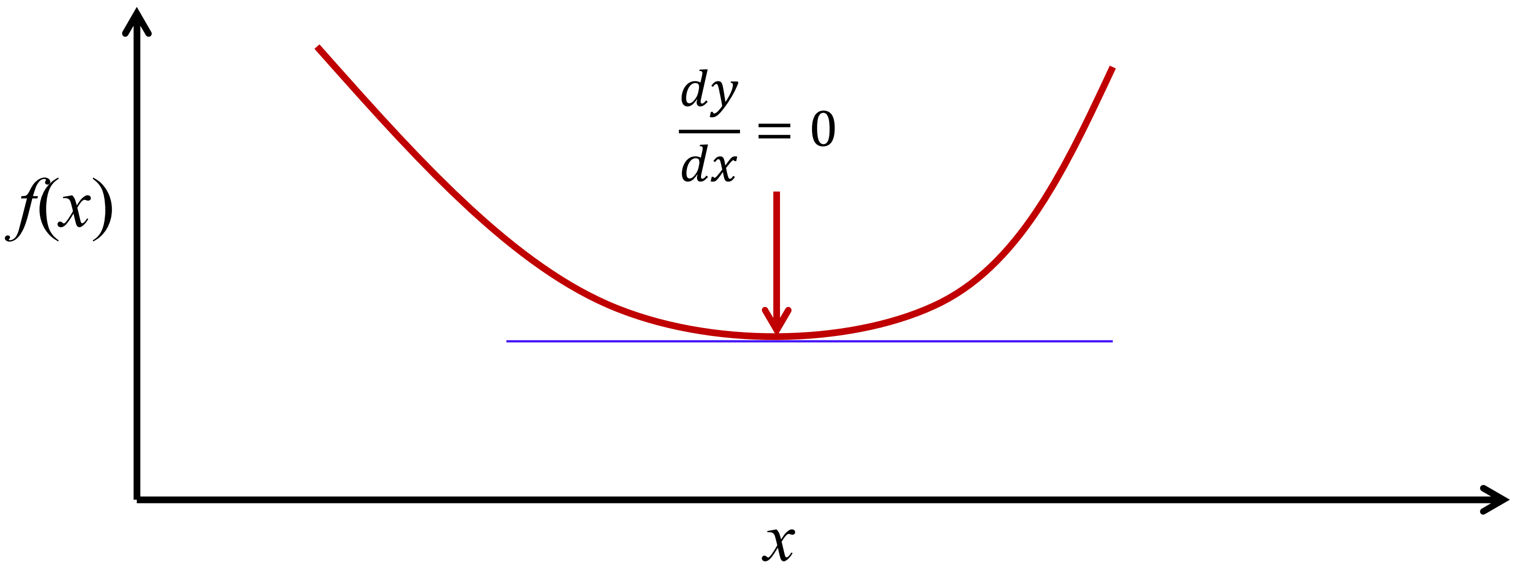




- General problem of optimization: Given a function $\mathbf{f}(\boldsymbol{x})$ of some variable $\boldsymbol{x}$ …
- Find the value of $\boldsymbol{x}$ where $\mathbf{f}(\boldsymbol{x})$ is minimum

# Finding the minimum of a function




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

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

- The solution is a "turning point"
  - Derivatives go from positive to negative or vice versa at this point
- But is it a minimum?

# Poll 2

Which of the following is true (choose only one) about the minimum of a function f(x)

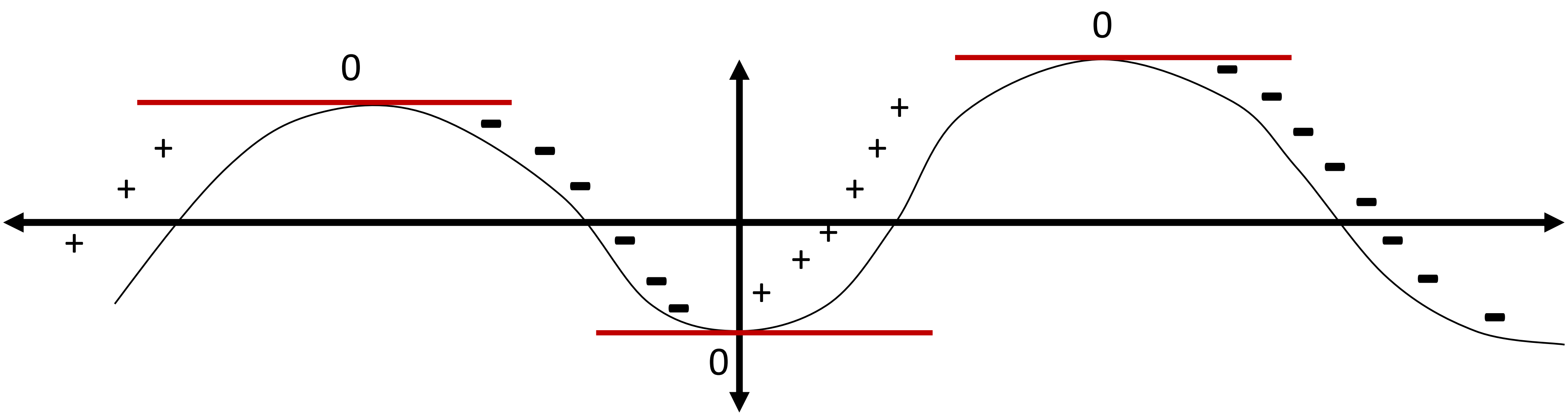
1. The derivative $f'(x) = 0$ at the minimum. This is the only condition to be satisfied
2. $f'(x) = 0$ and the second derivative $f''(x)$ is negative
3. $f'(x) = 0$ and the second derivative $f''(x)$ is positive

# Poll 2

Which of the following is true (choose only one) about the minimum of a function f(x)

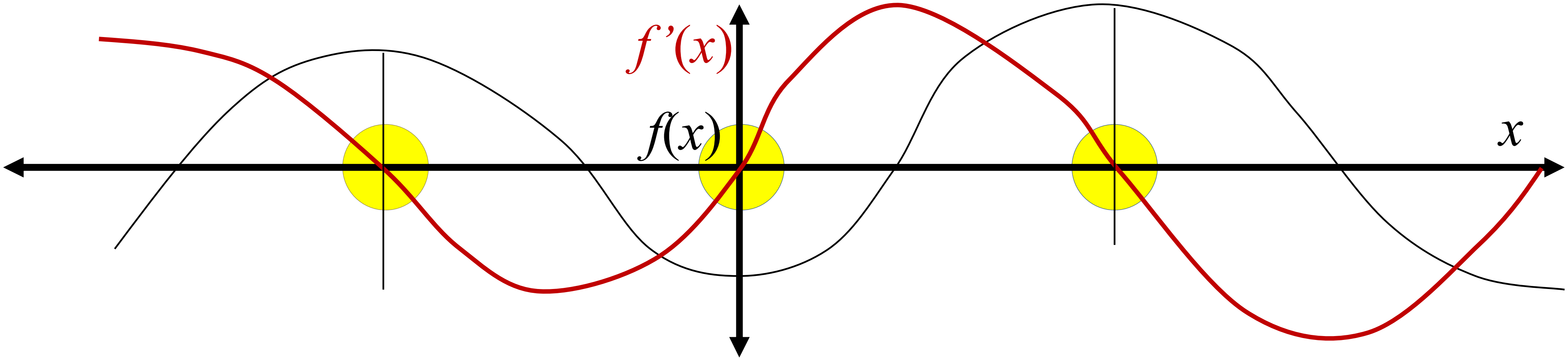
1. The derivative f'(x) = 0 at the minimum. This is the only condition to be satisfied
2. f'(x) = 0 and the second derivative f''(x) is negative
3. **f'(x) = 0 and the second derivative f''(x) is positive**

# Turning Points



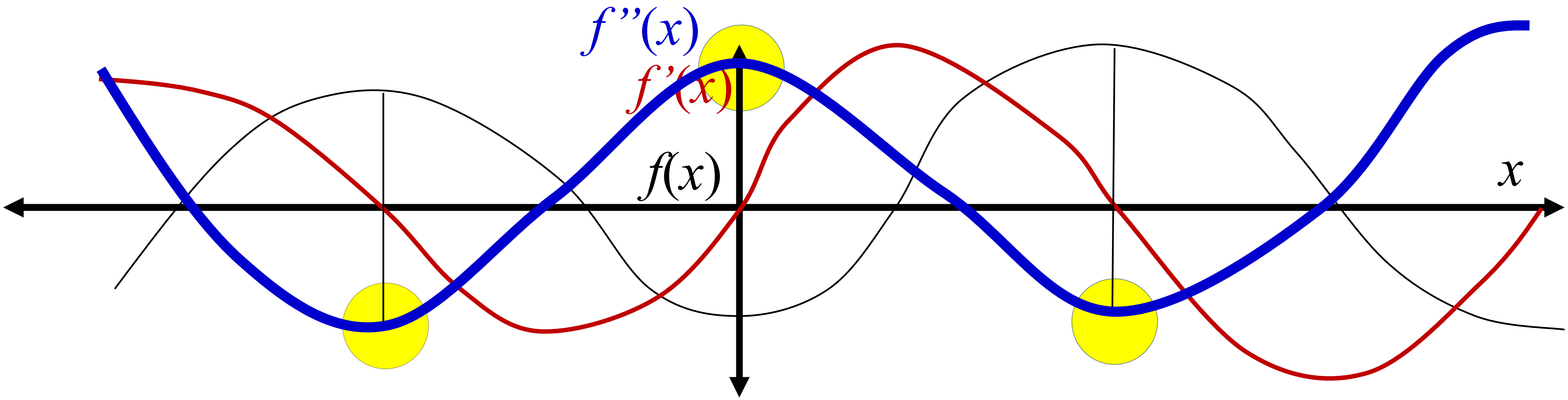

- Both *maxima* and *minima* have zero derivative
- Both are turning points

# Derivatives of a curve




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

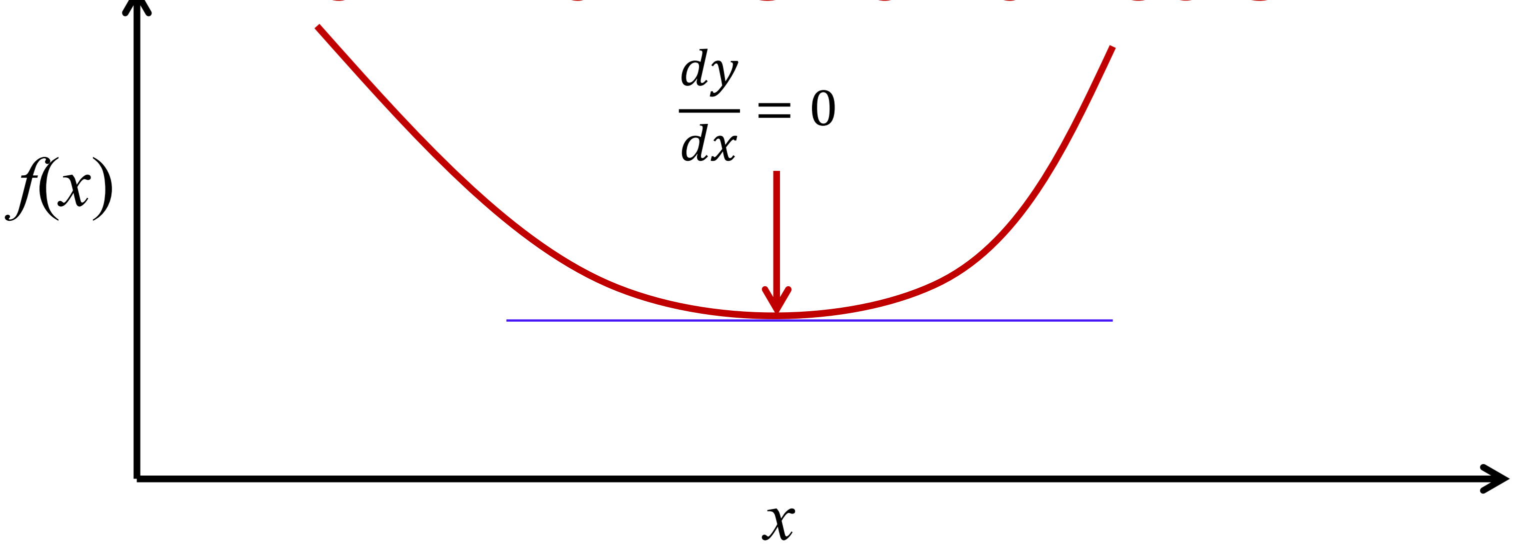
# Derivative of the derivative of the curve




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

- The *second derivative f''*(*x*) is –ve at maxima and +ve at minima!

# Solution: Finding the minimum or maximum of a function




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

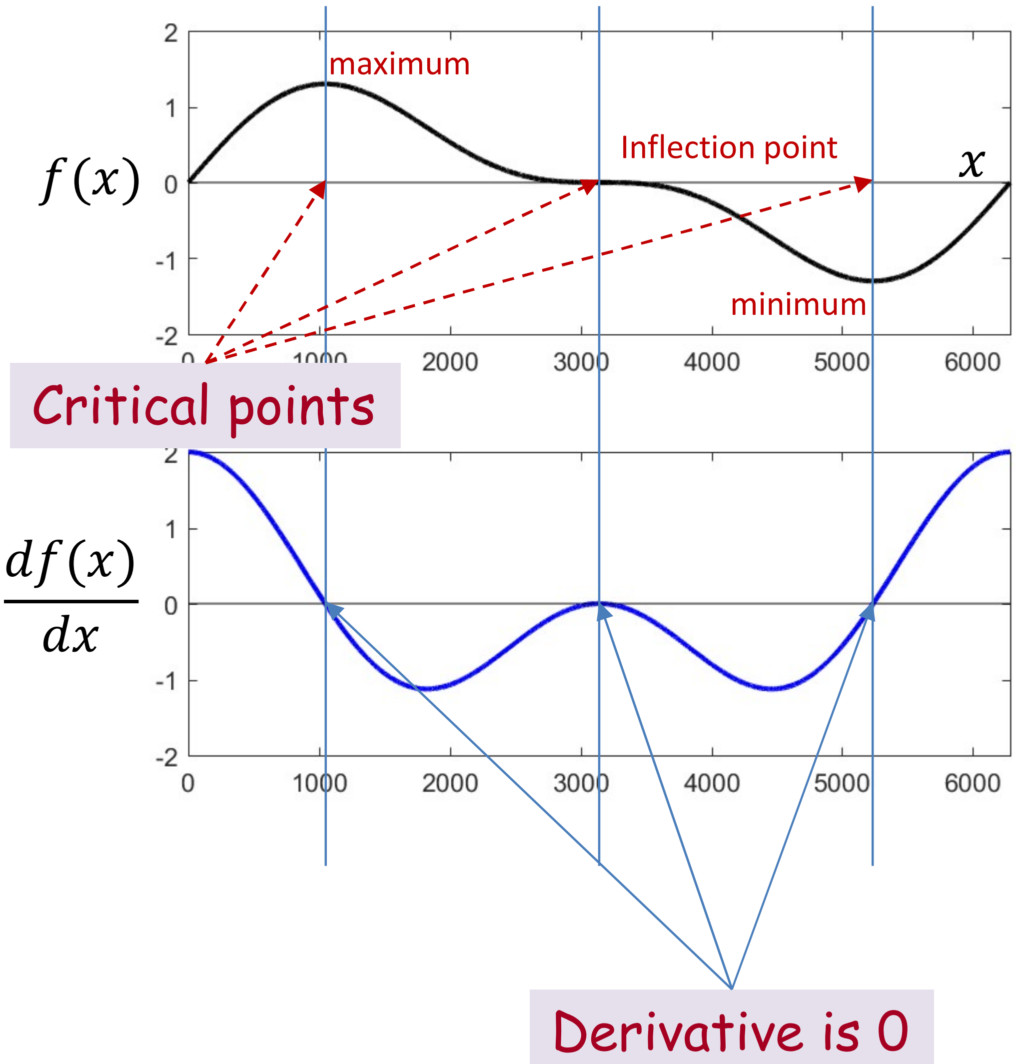
$$\frac{df(x)}{dx} = 0$$

- The solution $x_{soln}$ is a ***turning point***
- Check the double derivative at $x_{soln}$ : compute

$$f''(x_{soln}) = \frac{df'(x_{soln})}{dx}$$

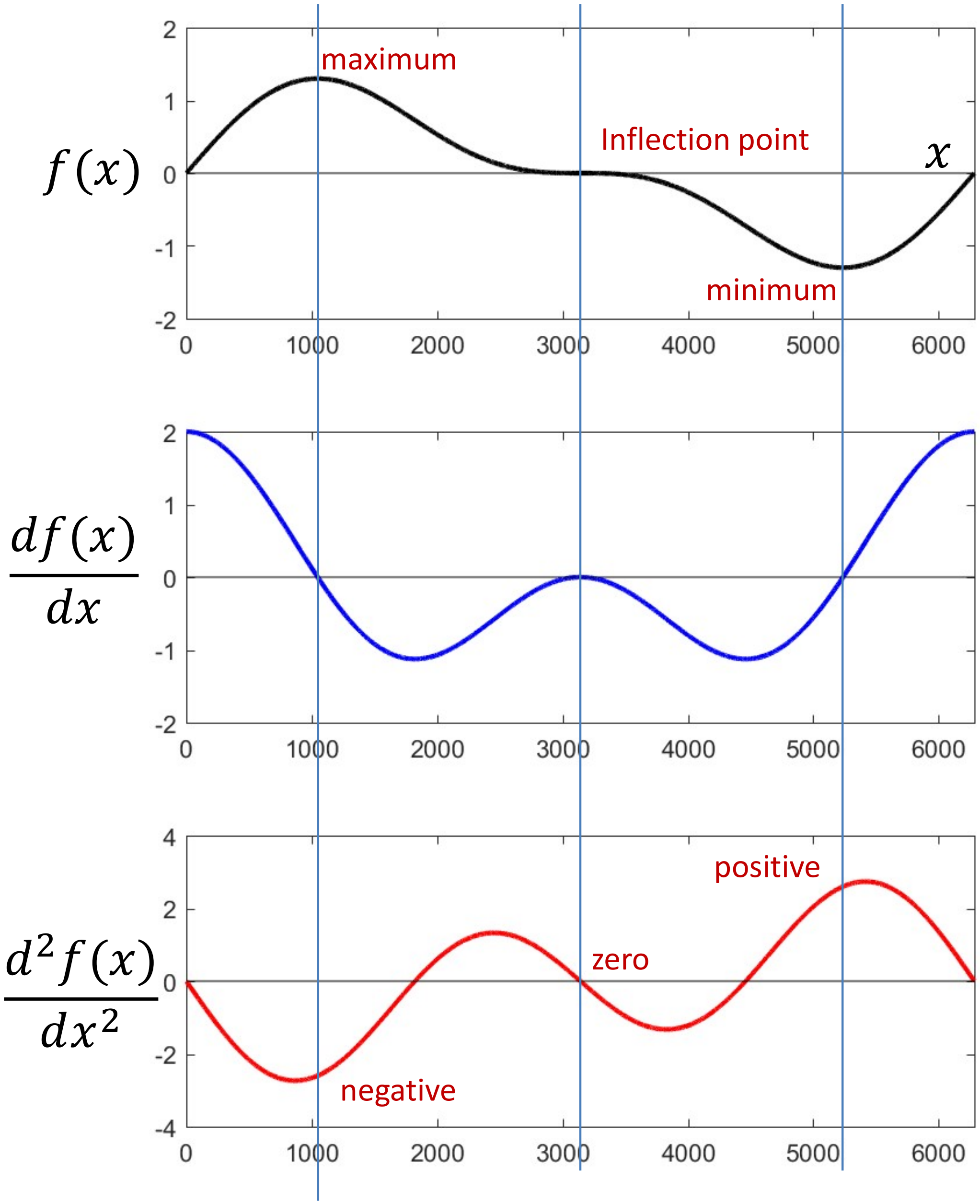
- If $f''(x_{soln})$ is positive $x_{soln}$ is a minimum, otherwise it is a maximum

# A note on derivatives of functions of single variable



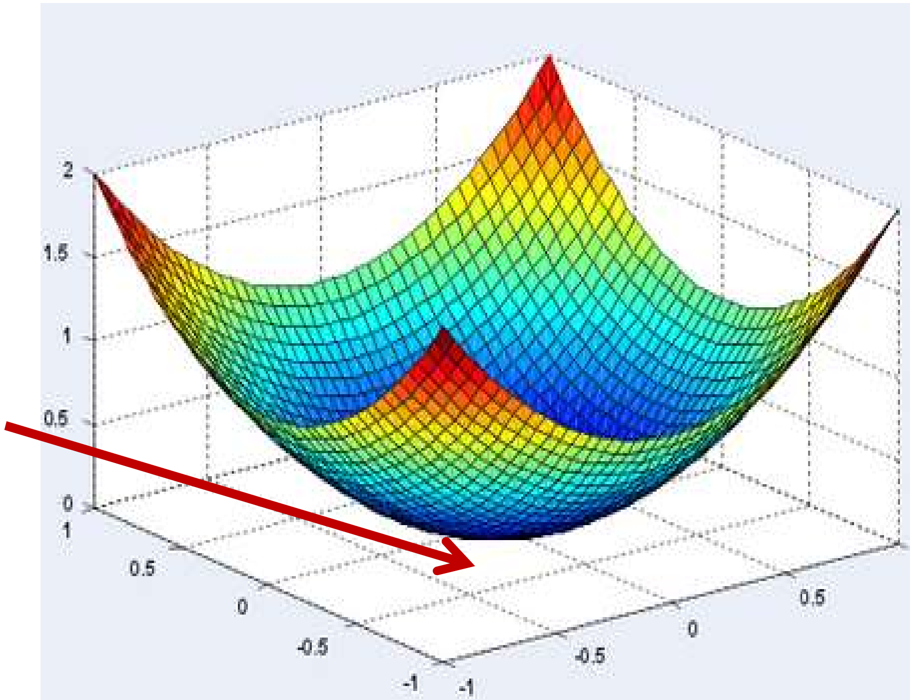

- All locations with zero derivative are *critical* points
  - These can be local maxima, local minima, or inflection points
- The *second* derivative is
  - Positive (or 0) at minima
  - Negative (or 0) at maxima
  - Zero at inflection points

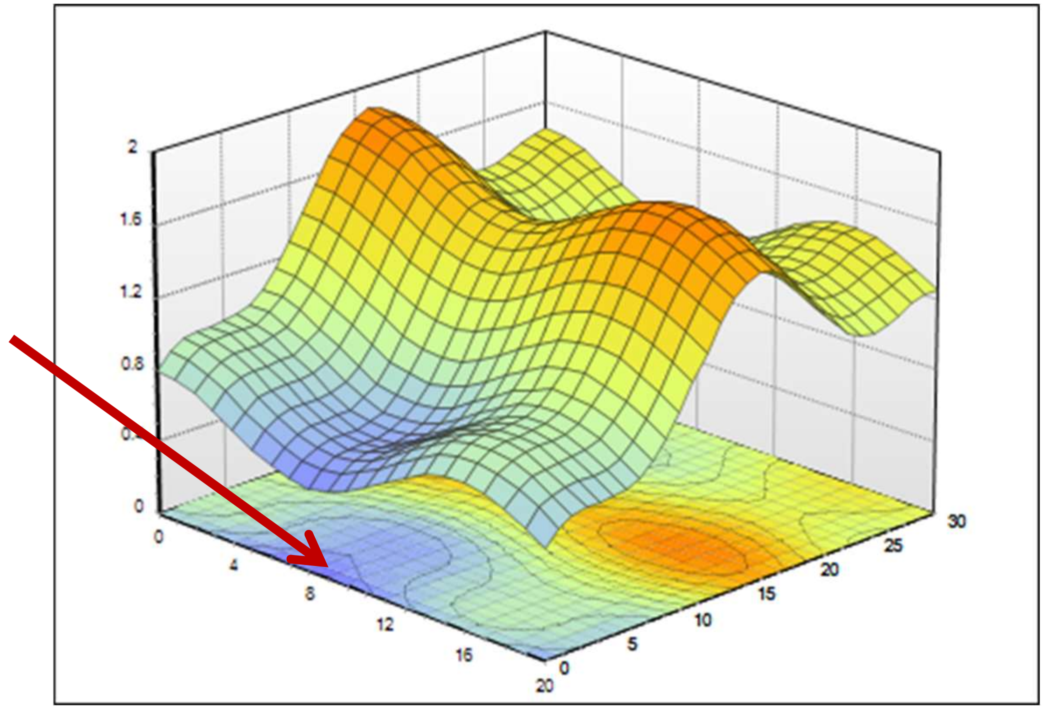
# A note on derivatives of functions of single variable




- All locations with zero derivative are *critical* points
  - These can be local maxima, local minima, or inflection points
- The *second* derivative is
  - $\geq 0$ at minima
  - $\leq 0$ at maxima
  - Zero at inflection points
- It's a little more complicated for functions of multiple variables..

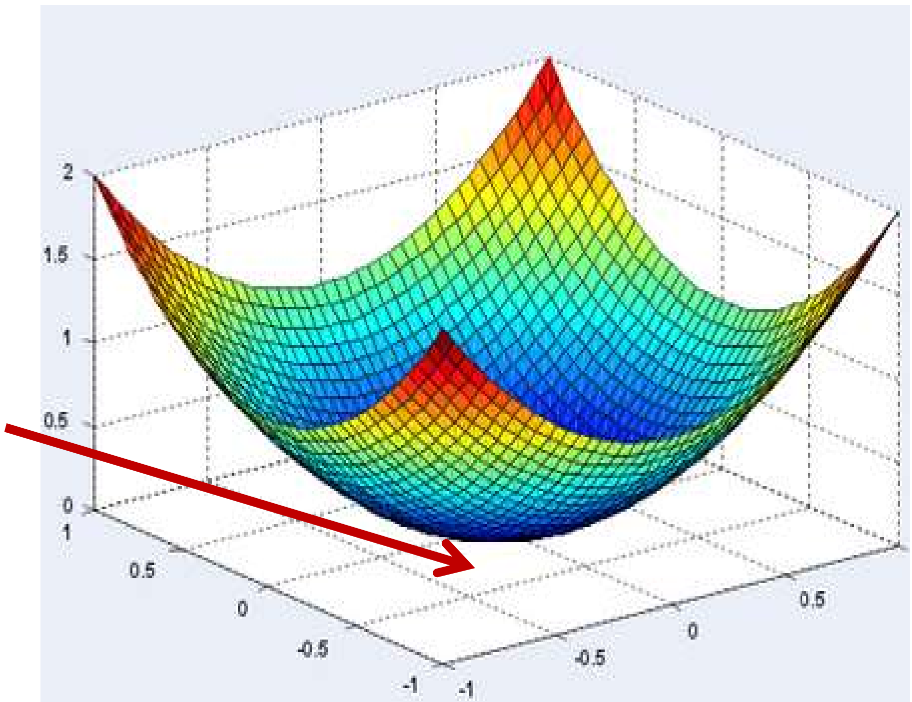
# What about functions of multiple variables?

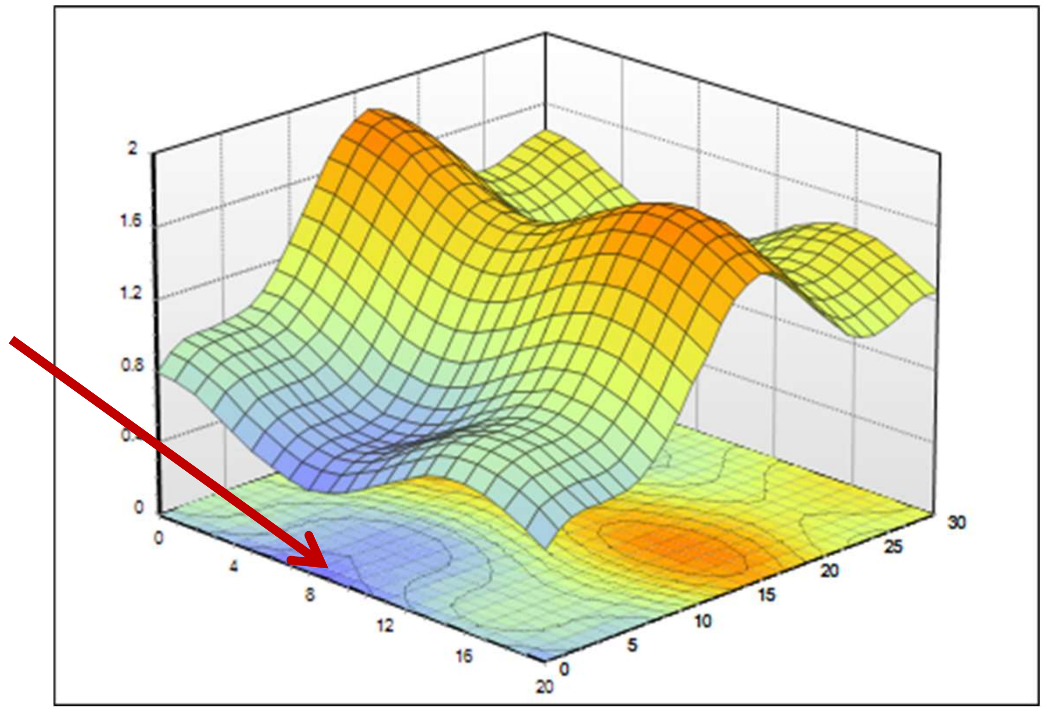




- The optimum point is still "turning" point
  - Shifting in any direction will increase the value
  - For smooth functions, at the minimum/maximum, the gradient is 0
    - Really tiny shifts will not change the function value

# 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

# Unconstrained Minimization of function (Multivariate)

1. Solve for the $X$ where the derivative (or gradient) equals to zero

$$\nabla_X f(X) = 0$$

2. Compute the Hessian Matrix $\nabla_X^2 f(X)$ at the candidate solution and verify that
   – Hessian is positive definite (eigenvalues positive) -> to identify local minima
   – Hessian is negative definite (eigenvalues negative) -> to identify local maxima

# Unconstrained Minimization of function (Example)

- Minimize

$$f(x_1, x_2, x_3) = (x_1)^2 + x_1(1 - x_2) + (x_2)^2 - x_2 x_3 + (x_3)^2 + x_3$$

- Gradient

$$\nabla_X f^T = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix}$$

# Unconstrained Minimization of function (Example)

- Set the gradient to null

$$\nabla_X f = 0 \Rightarrow \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- Solving the 3 equations system with 3 unknowns

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

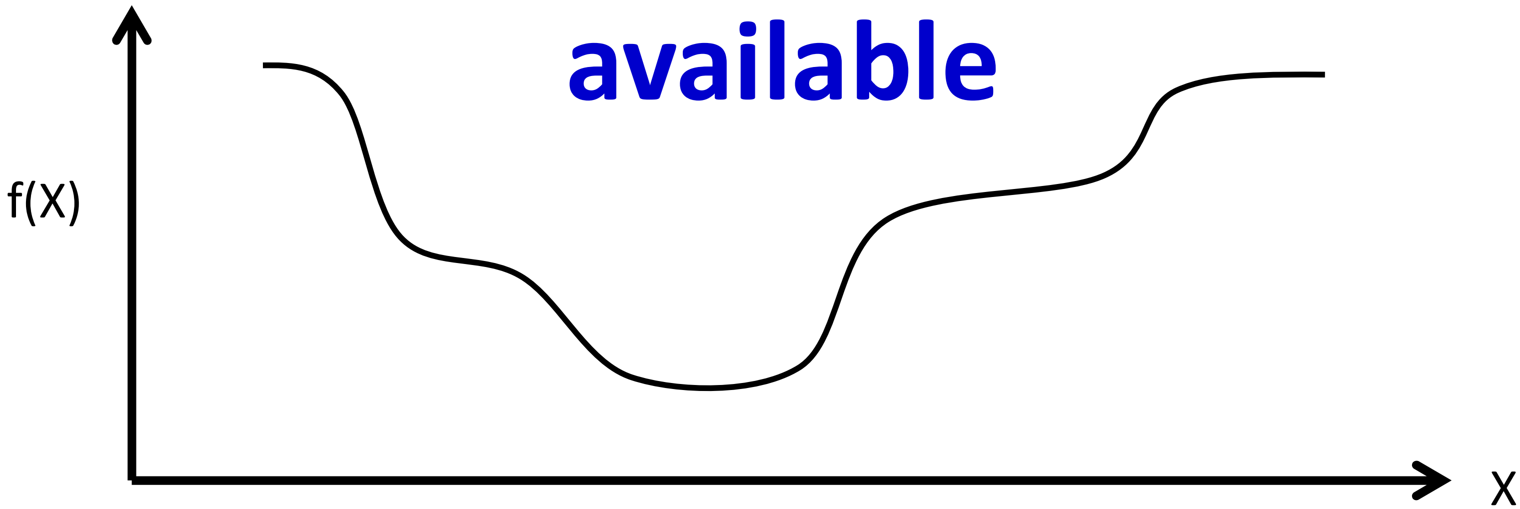
# Unconstrained Minimization of function (Example)

- Compute the Hessian matrix $\nabla_X^2 f = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$
- Evaluate the eigenvalues of the Hessian matrix

$$\lambda_1 = 3.414, \quad \lambda_2 = 0.586, \quad \lambda_3 = 2$$

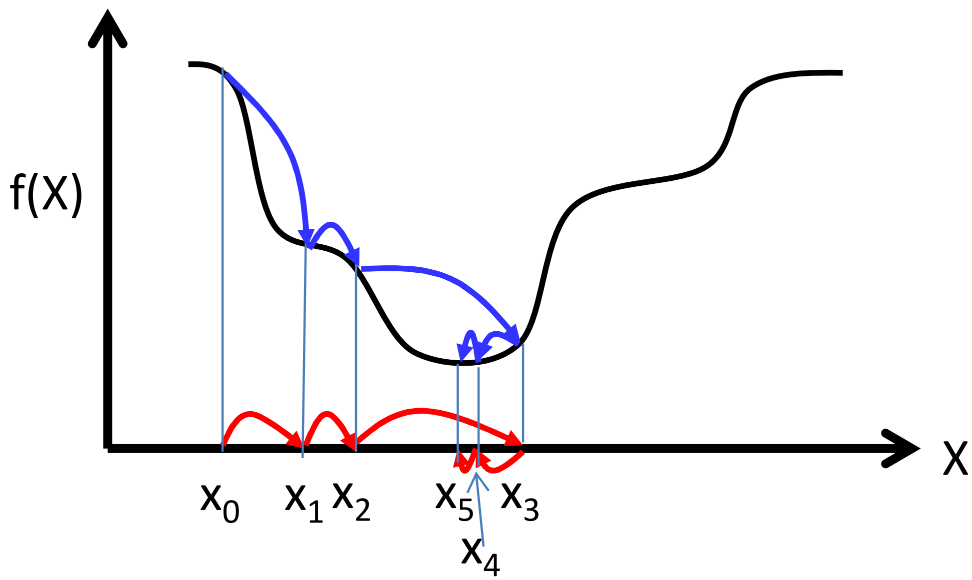
- All the eigenvalues are positives => the Hessian matrix is positive definite
- The point $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$ is a minimum

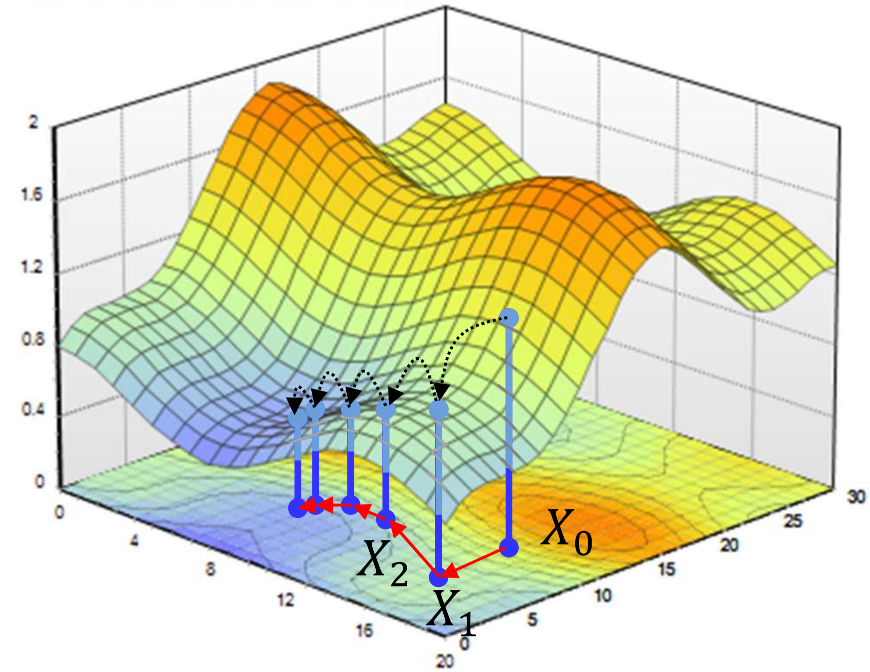
# Closed Form Solutions are not always available




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

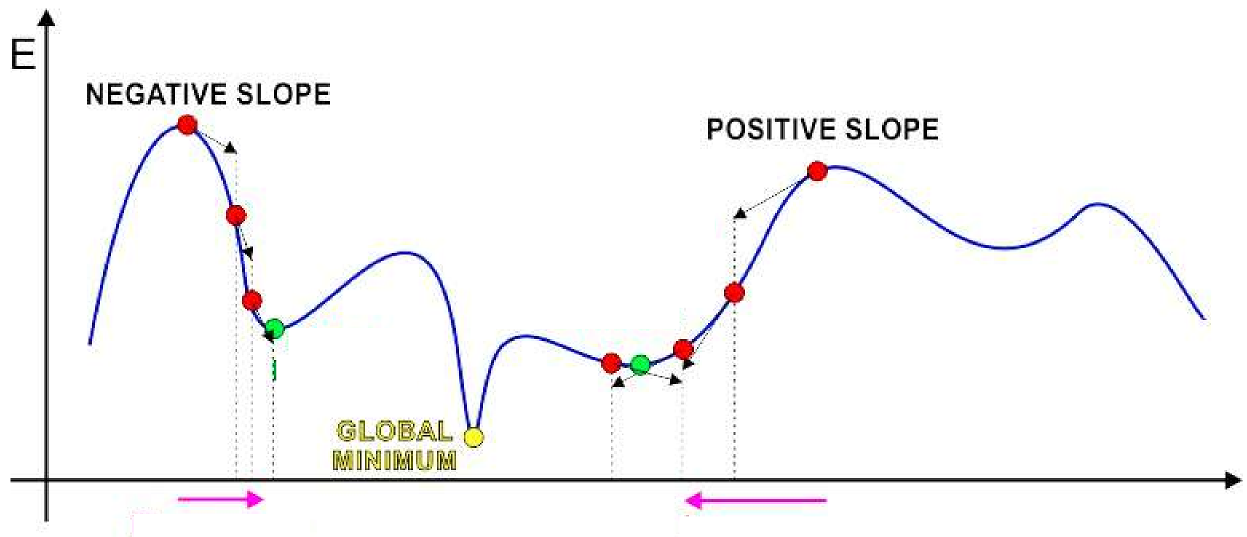
# Iterative solutions



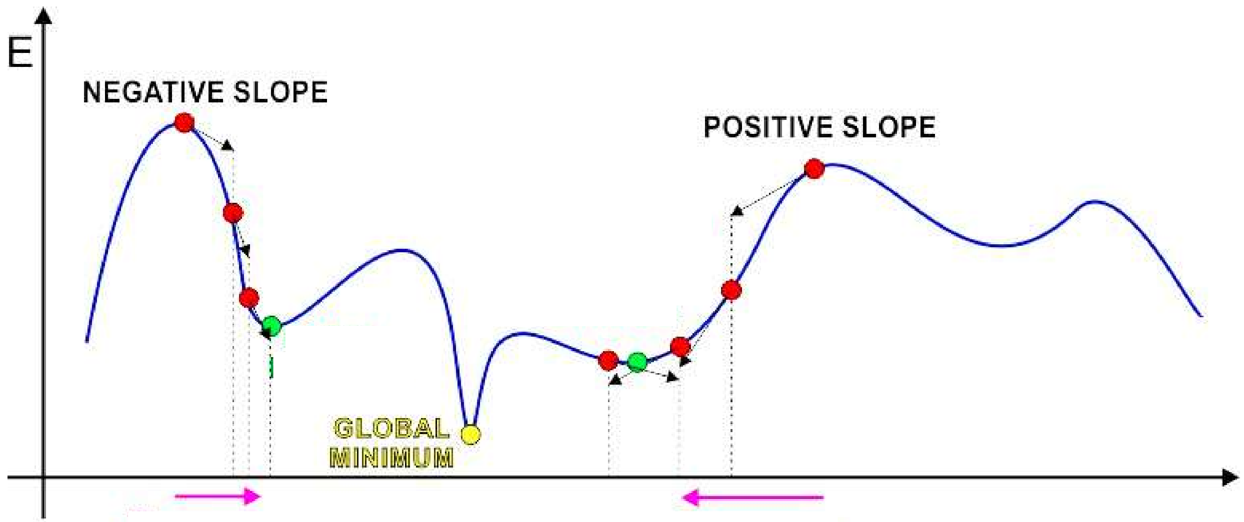

- Iterative solutions
  - Start from an initial guess $X_0$ for the optimal $X$
  - Update the guess towards a (hopefully) "better" value of $\mathrm{f}(X)$
  - Stop when $\mathrm{f}(X)$ no longer decreases
- Problems:
  - Which direction to step in
  - How big must the steps be

# The Approach of Gradient Descent




- 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 → moving right decreases error
      - A positive derivative → moving left decreases error
  - Shift point in this direction

# The Approach of Gradient Descent




- Iterative solution: Trivial algorithm
  - Initialize $x^0$
  - While $f'\left(x^k\right) \neq 0$
    - If $sign\left(f'\left(x^k\right)\right)$ is positive:

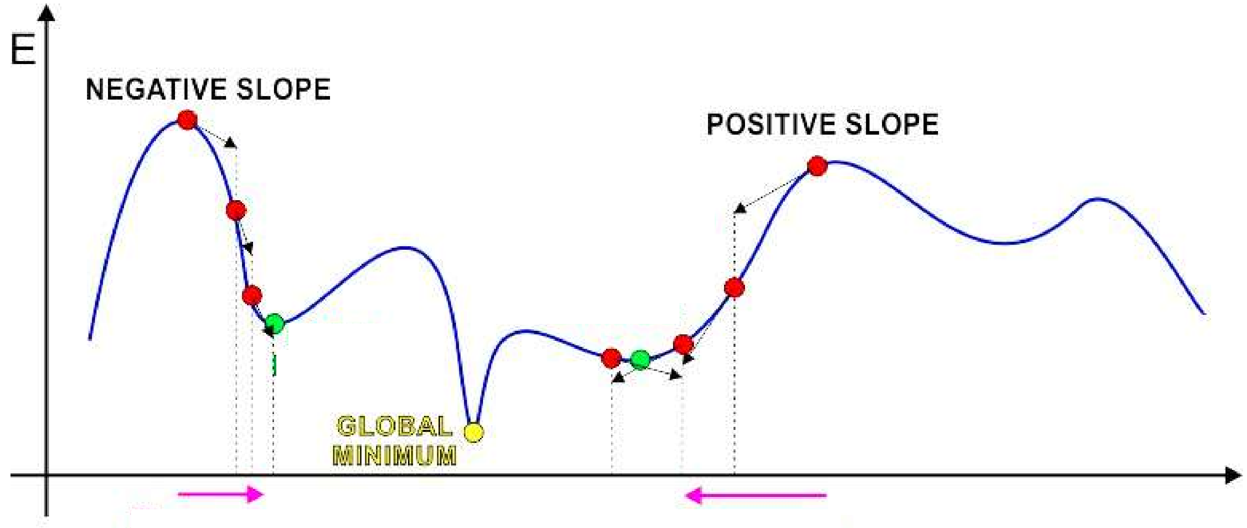
$$x^{k+1} = x^k - step$$

    - Else

$$x^{k+1} = x^k + step$$

– What must step be to ensure we actually get to the optimum?

# The Approach of Gradient Descent

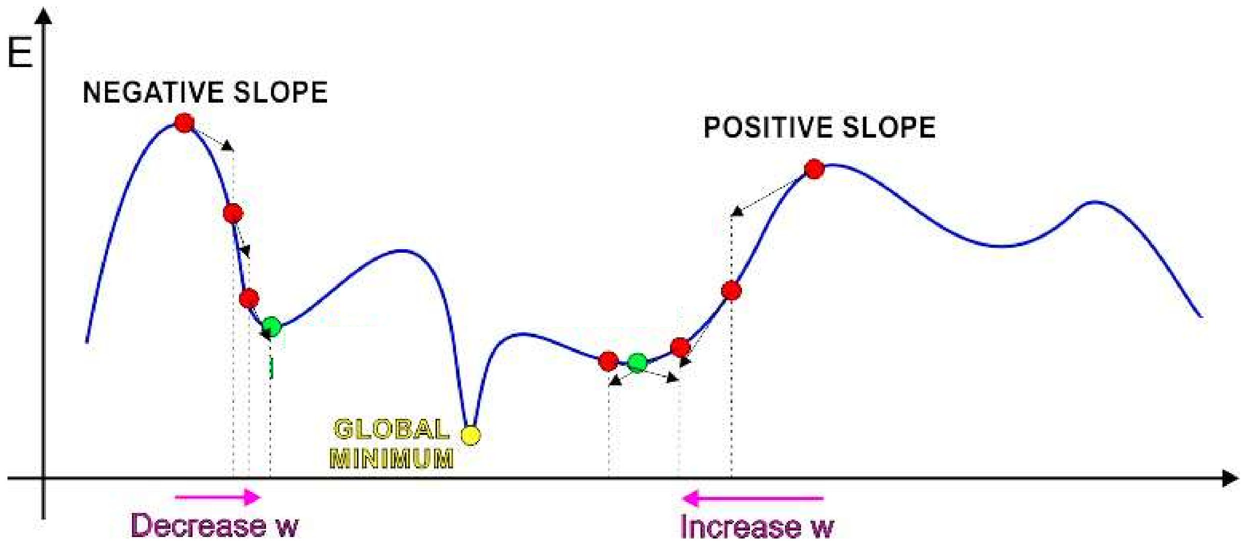



- Iterative solution: Trivial algorithm
  - Initialize $x^0$
  - While $f'(x^k) \neq 0$

$$x^{k+1} = x^k - sign\left(f'(x^k)\right).step$$

- Identical to previous algorithm

# The Approach of Gradient Descent




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

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

- $\eta^k$ is the “step size”

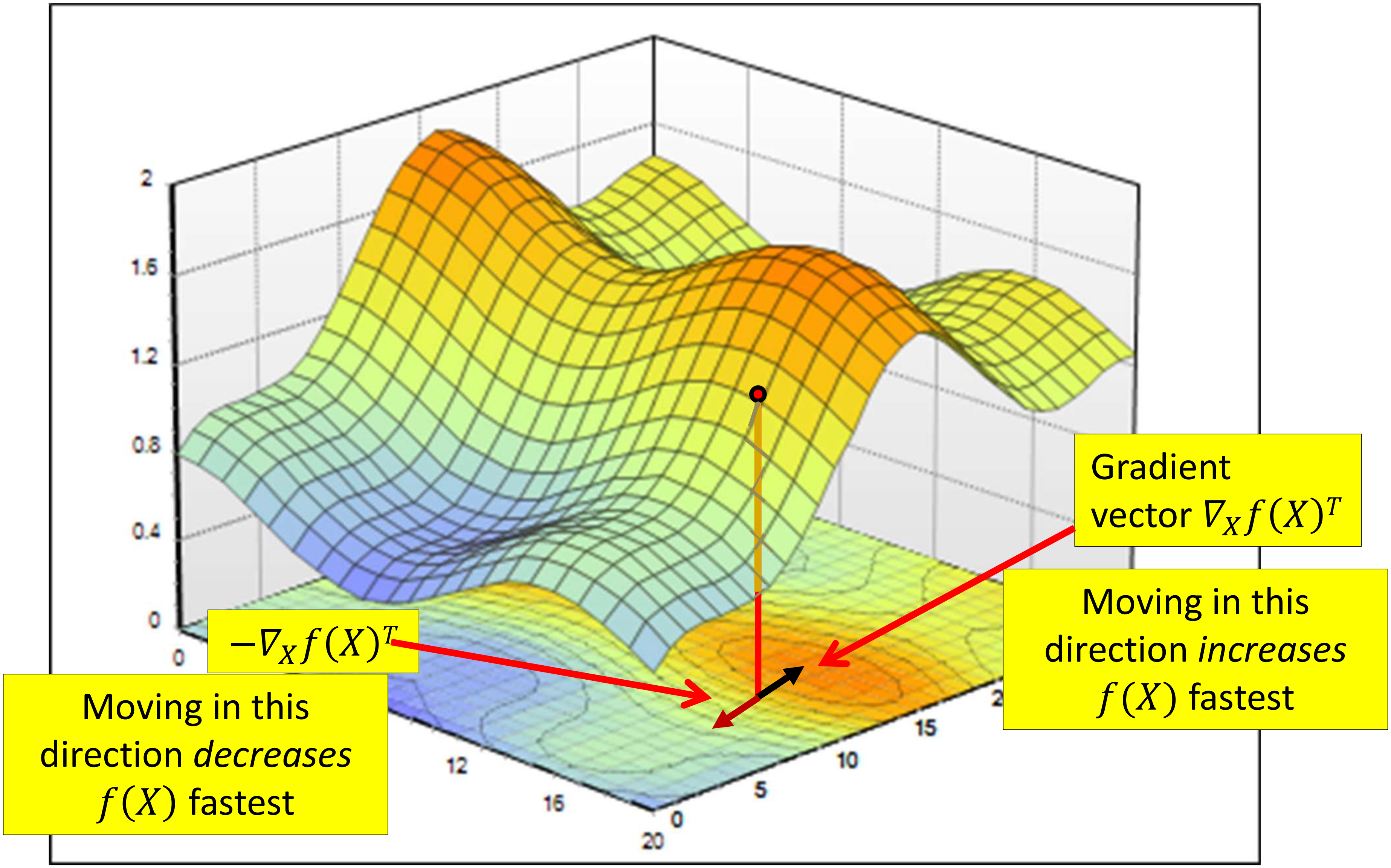
# Poll 3: Multivariate functions

- Select all that are true about derivatives of a scalar function f(X) of multivariate inputs
  – At any location X, there may be many directions in which we can step, such that f(X) increases
  – The direction of the gradient is the direction in which the function increases fastest
  – The gradient is the derivative of f(X) w.r.t. X

# Poll 3: Multivariate functions

- Select all that are true about derivatives of a scalar function f(X) of multivariate inputs
  - **At any location X, there may be many directions in which we can step, such that f(X) increases**
  - **The direction of the gradient is the direction in which the function increases fastest**
  - The gradient is the derivative of f(X) w.r.t. X

# Gradients of multivariate functions



# 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*

$$x^{k+1} = x^k + \eta^k \nabla_x f\left(x^k\right)^T$$

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

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

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

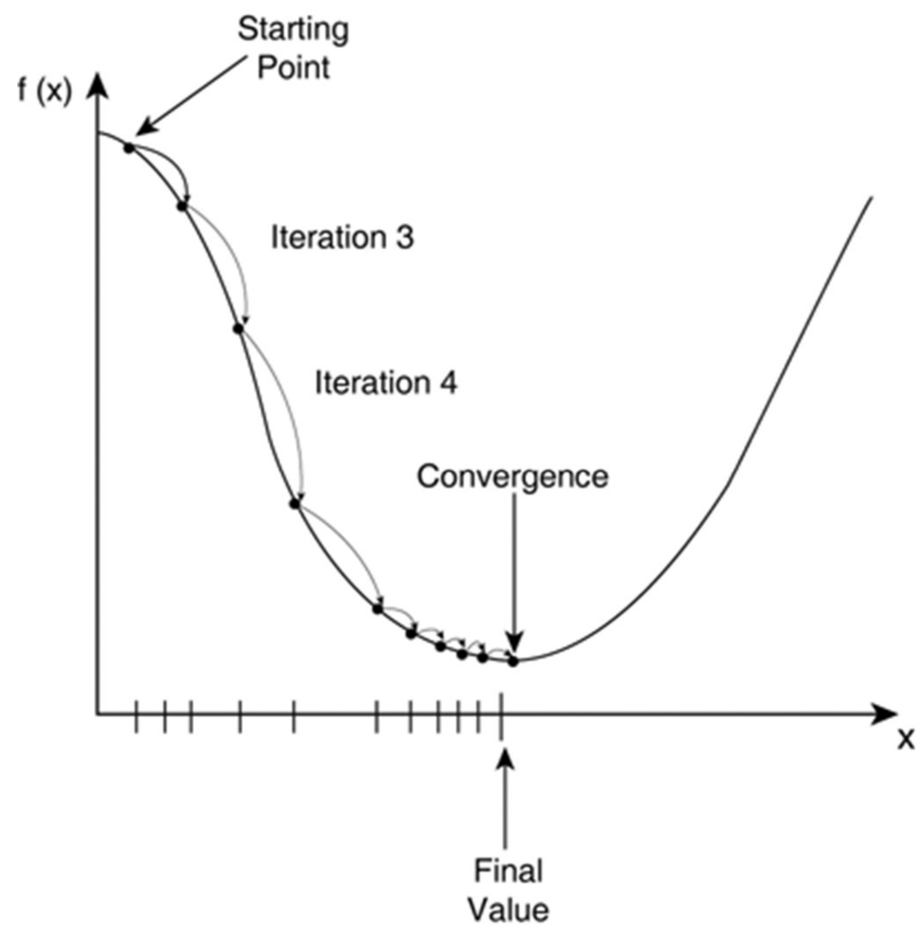
# Gradient descent convergence criteria

- The gradient descent algorithm converges when one of the following criteria is satisfied

$$\left| f(x^{k+1}) - f(x^k) \right| < \varepsilon_1$$

- Or

$$\left\| \nabla_x f(x^k) \right\| < \varepsilon_2$$

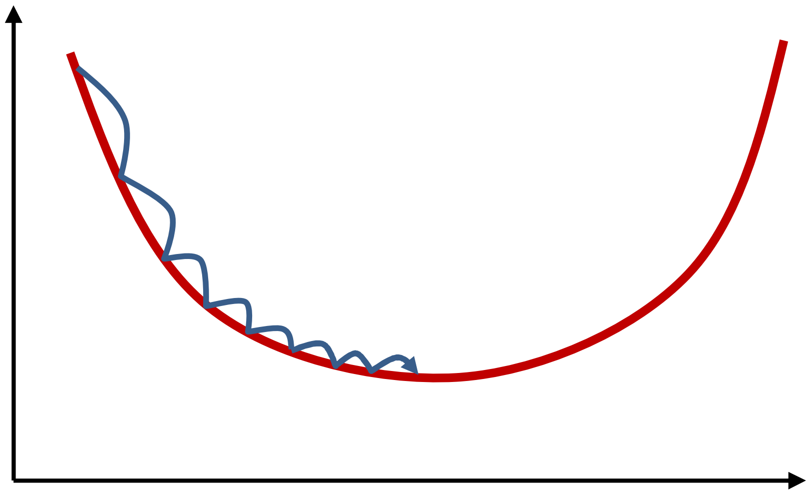


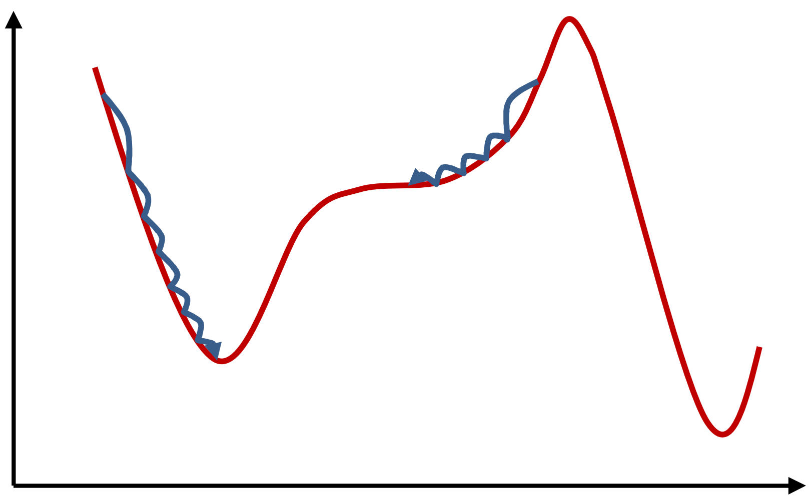
# Overall Gradient Descent Algorithm

- Initialize:
  - $x^0$
  - $k = 0$

- do
  - $x^{k+1} = x^k - \eta^k \nabla_x f\left(x^k\right)^T$
  - $k = k + 1$
- while $\left|f\left(x^{k+1}\right) - f\left(x^k\right)\right| > \varepsilon$

# 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

# Poll 4

- y = f(x) is a scalar function of an Nx1 column vector variable x. Starting from $x = x_0$, in which direction must we move in the space of x, to achieve the maximum decrease in f()?
  - Exactly in the direction of the gradient of f(x) at $x_0$
  - Exactly perpendicular to the direction of the gradient of f(x) at $x_0$
  - Exactly opposite to the direction of the gradient of f(x) at $x_0$
  - Exactly perpendicular to the direction of the gradient of f(x) at $x_0$.

# Poll 4

- $y = f(x)$ is a scalar function of an Nx1 column vector variable x. Starting from $x = x_0$, in which direction must we move in the space of x, to achieve the maximum decrease in f()?
  - Exactly in the direction of the gradient of $f(x)$ at $x_0$
  - Exactly perpendicular to the direction of the gradient of $f(x)$ at $x_0$
  - **Exactly opposite to the direction of the gradient of $f(x)$ at $x_0$**
  - Exactly perpendicular to the direction of the gradient of $f(x)$ at $x_0$.

- Returning to our problem from our detour..

# Problem Statement

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

- Minimize the following function

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

  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

- $W^{k+1} = W^k - \eta^k \nabla Loss(W^k)^T$
- $k = k + 1$

while $\left|Loss\left(W^k\right) - Loss\left(W^{k-1}\right)\right| > \varepsilon$

# Preliminaries

- Before we proceed: the problem setup

# Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

- Minimize the following function

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

w.r.t $W$

# 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 $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

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

What is the divergence div()?

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

# Problem Setup: Things to define

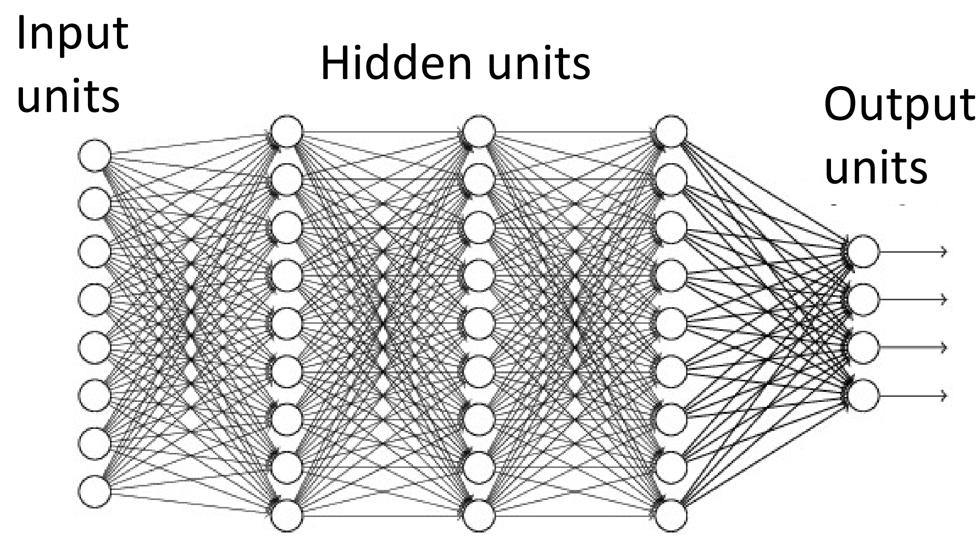
- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

- Minimize the following function

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

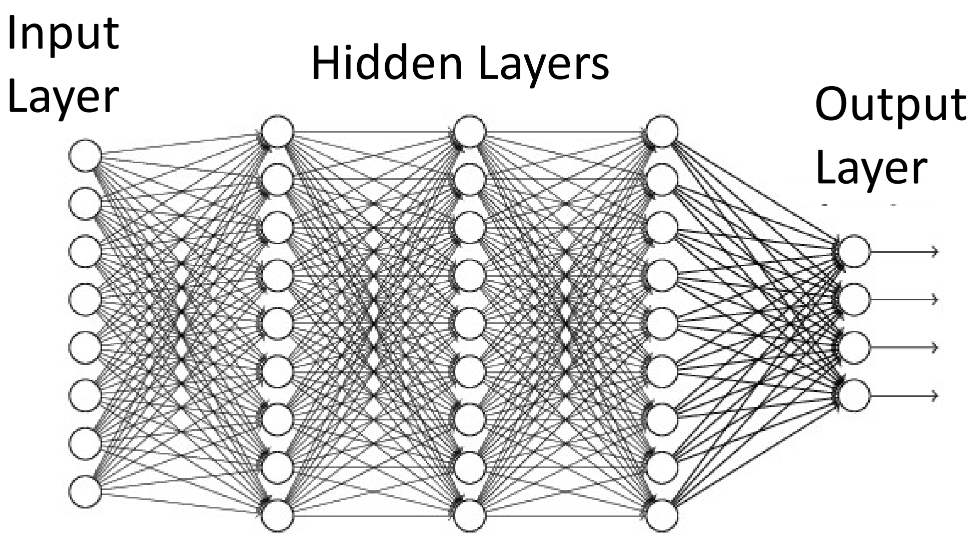
What is f() and what are its parameters W?

# What is f()? Typical network



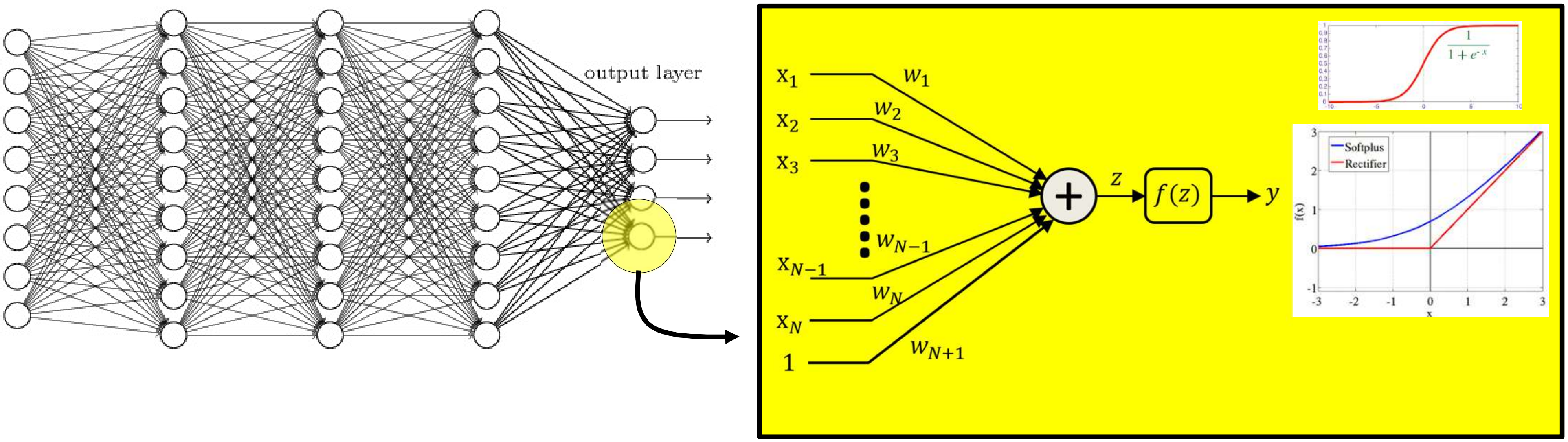

- Multi-layer perceptron
- A *directed* network with a set of inputs and outputs
  – No loops

# 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



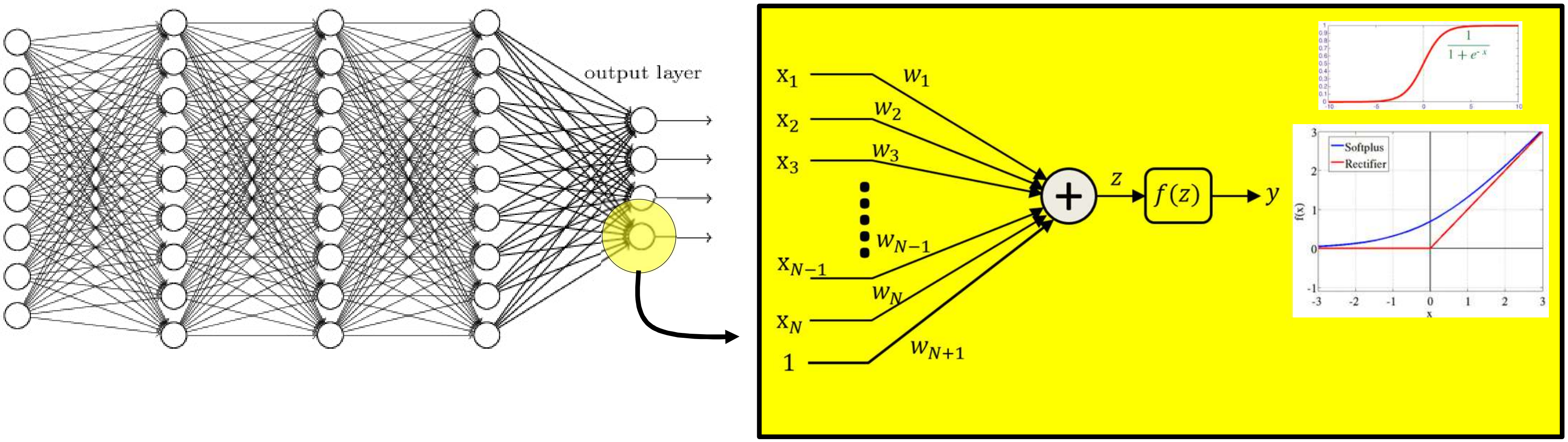

- Individual neurons operate on a set of inputs and produce a single output
  - **Standard setup:** A continuous activation function applied to an affine function of the inputs

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

  - More generally: *any* differentiable function

$$y = f(x_1, x_2, \ldots, x_N; W)$$

# The individual neurons




- Individual neurons operate on a set of inputs and produce a single output
  - **Standard setup:** A continuous activation function applied to an affine function of the inputs

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

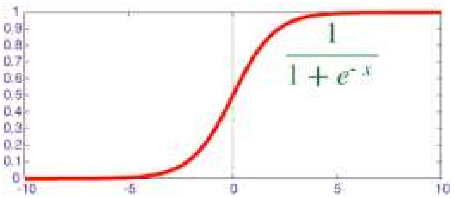
  - More generally: *any* differentiable function

$$y = f(x_1, x_2, \ldots, x_N; W)$$

We will assume this unless otherwise specified

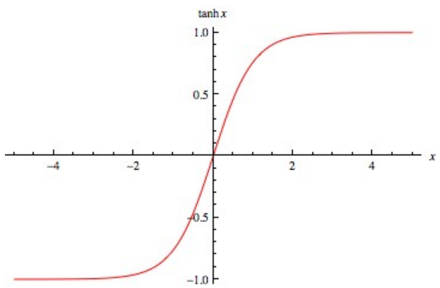
Parameters are weights $w_i$ and bias $b$

# Activations and their derivatives



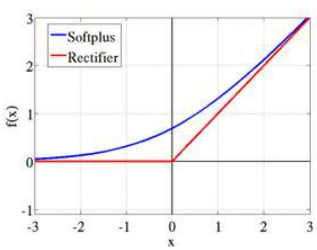

$$f(z) = \frac{1}{1+\exp(-z)}$$

$$f'(z) = f(z)(1-f(z))$$




$$f(z) = \tanh(z)$$

$$f'(z) = (1-f^2(z))$$




$$f(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

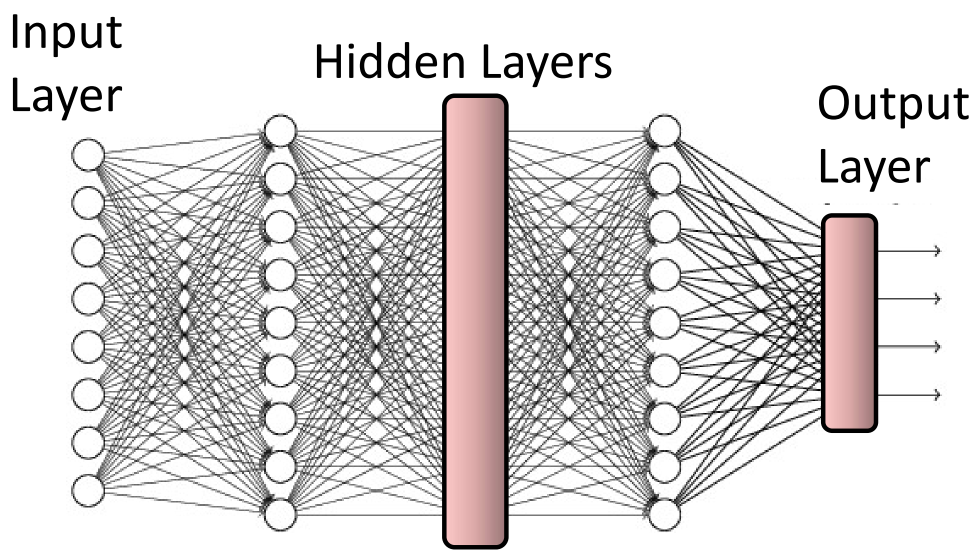
[*] $$f'(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$$

$$f(z) = \log(1+\exp(z))$$

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

- Some popular activation functions and their derivatives

# Vector Activations

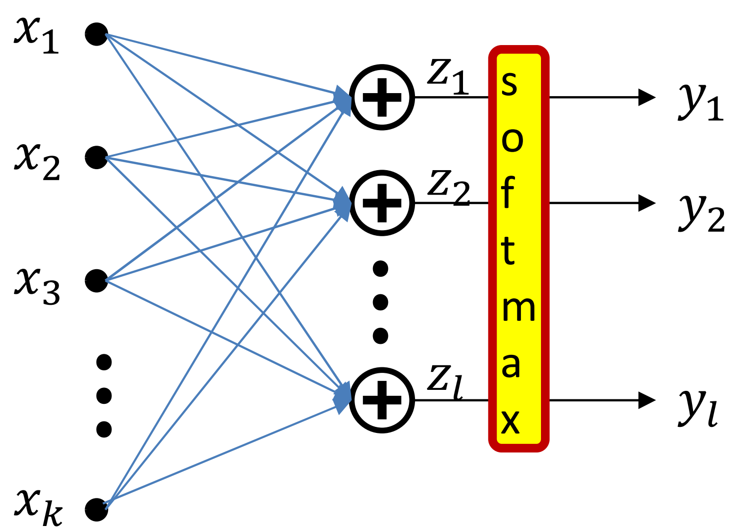



- We can also have neurons that have *multiple coupled* outputs

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

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

# Vector activation example: Softmax



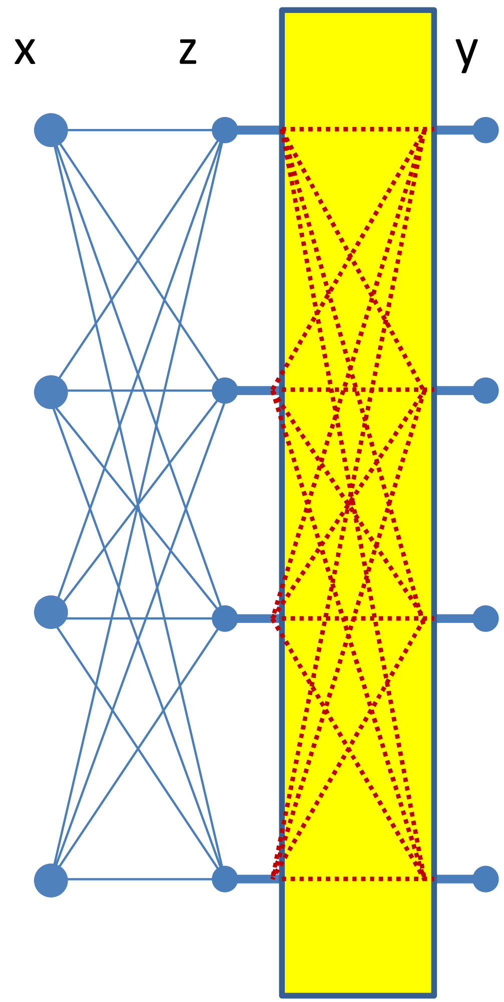

- Example: Softmax *vector* activation

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

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

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

# Multiplicative combination: Can be viewed as a case of vector activations



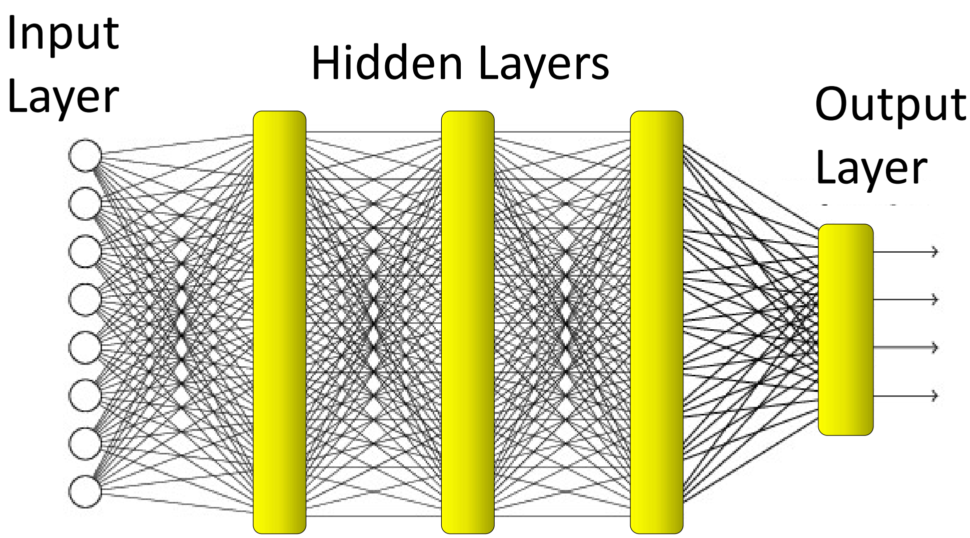

$$z_i = \sum_j w_{ji} x_j + b_i$$

$$y_i = \prod_l (z_l)^{\alpha_{li}}$$

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

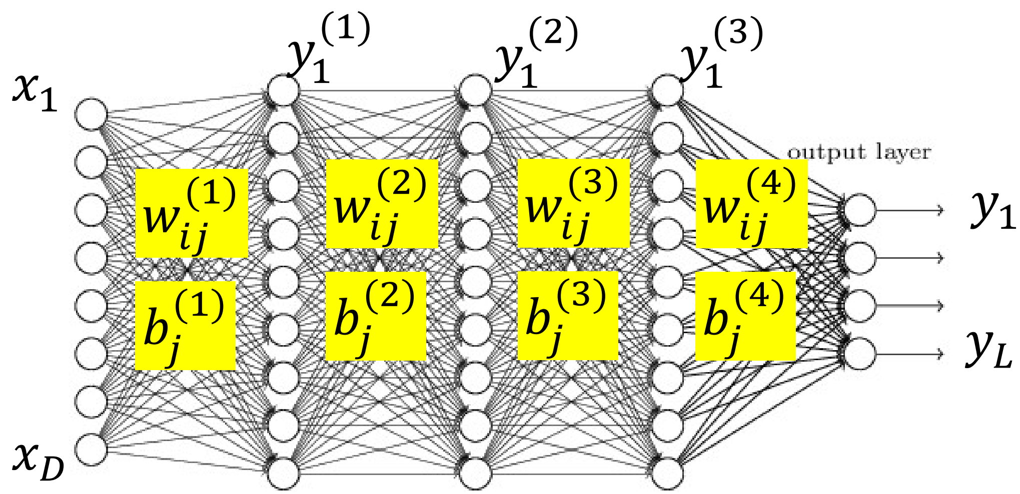
- A layer of multiplicative combination is a special case of vector activation

# 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 $b_j^{(k)}$

# Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

- Minimize the following function

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

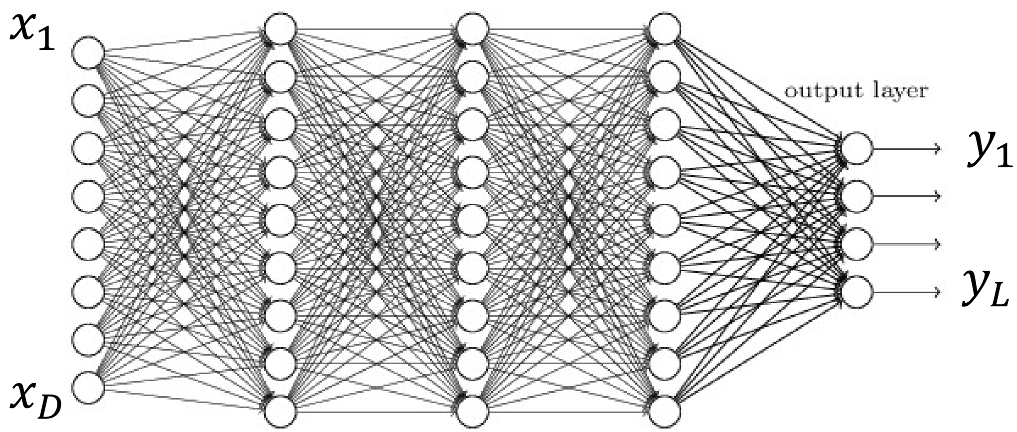
What is f() and what are its parameters W?

# Problem Setup: Things to define

- Given a training set of input-output pairs
  $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

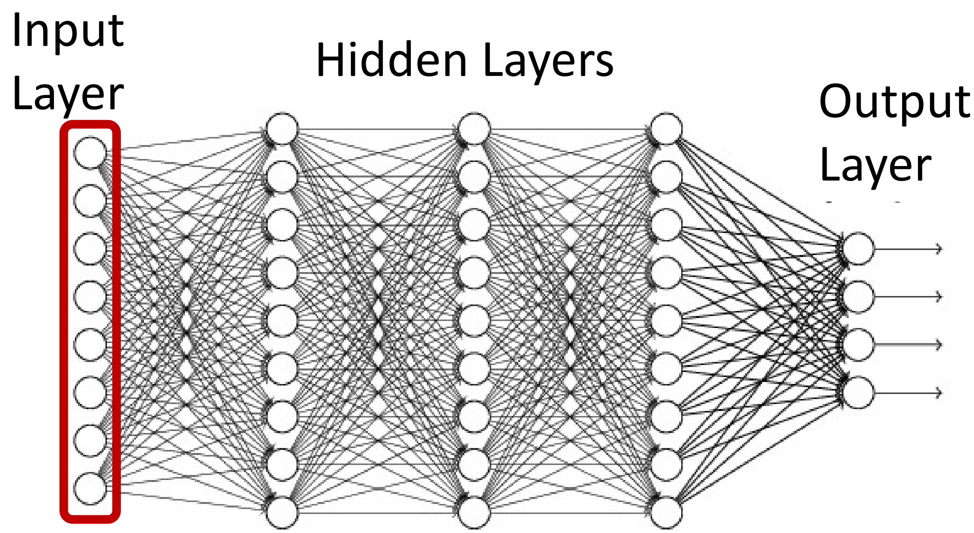
$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

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



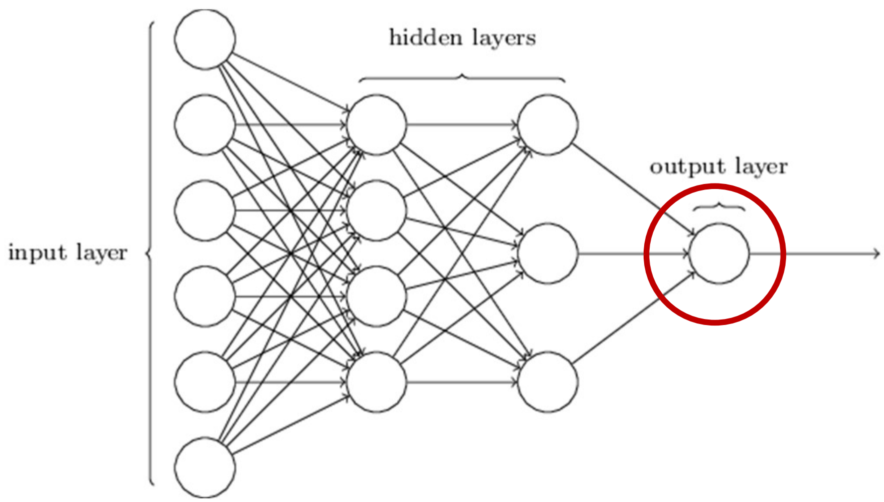

- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- $X_n = [x_{n1}, x_{n2}, \dots, x_{nD}]^\top$ is the nth input vector
- $d_n = [d_{n1}, d_{n2}, \dots, d_{nL}]^\top$ is the nth desired output
- $Y_n = [y_{n1}, y_{n2}, \dots, y_{nL}]^\top$ is the nth vector of *actual* outputs of the network
  - Function of input $X_n$ and network parameters
- We will sometimes drop the first subscript when referring to a *specific* instance

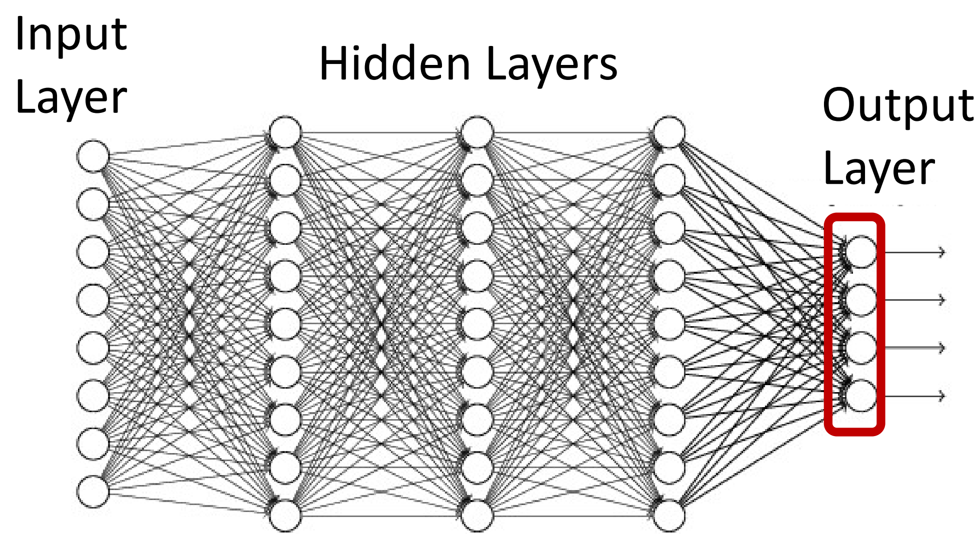
# Representing the input




- 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
    - We will see how this happens later in the course
  - Other real valued vectors

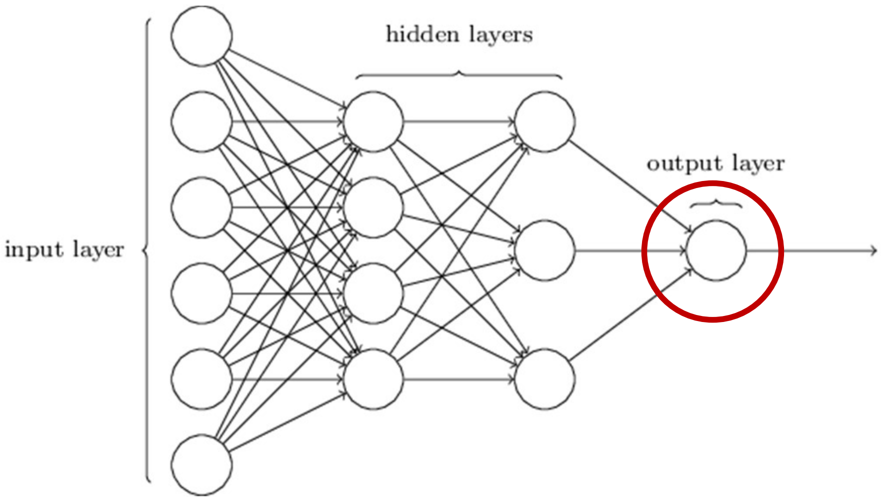
# Representing the output



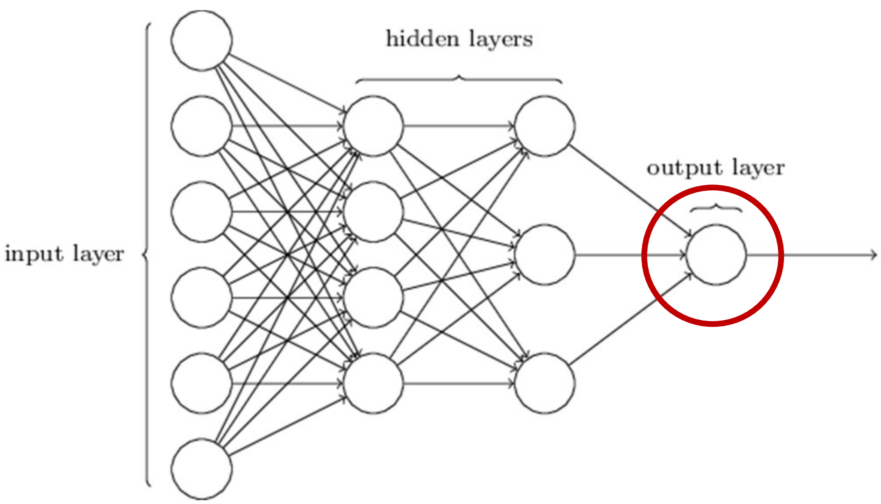

- If the desired *output* is real-valued, no special tricks are necessary
  - Scalar Output : single output neuron
    - d = scalar (real value)
  - Vector Output : as many output neurons as the dimension of the desired output
    - d = [$d_1$ $d_2$ .. $d_L$] (vector of real values)

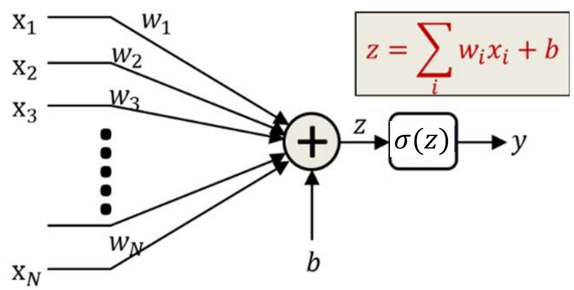
# Representing the output

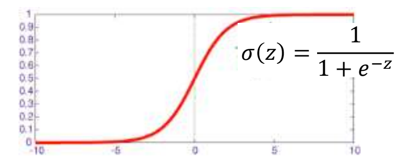



- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
  - 1 = Yes it's a cat
  - 0 = No it's not a cat.

# Representing the output

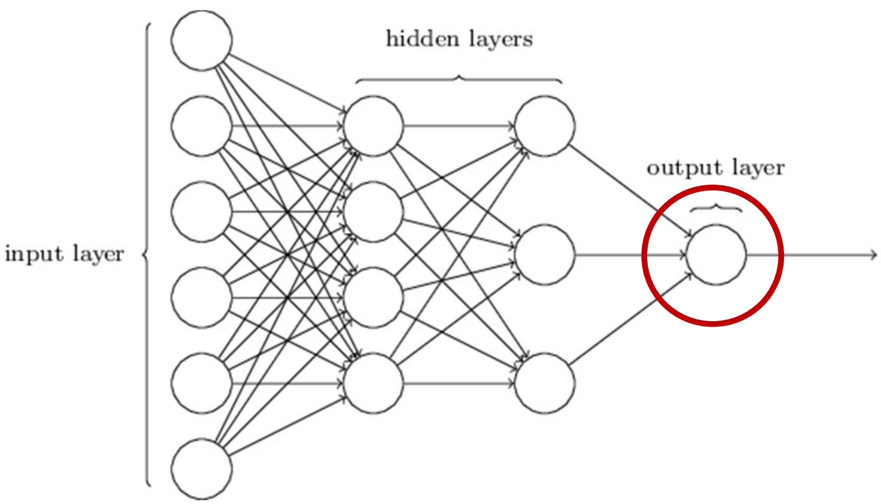


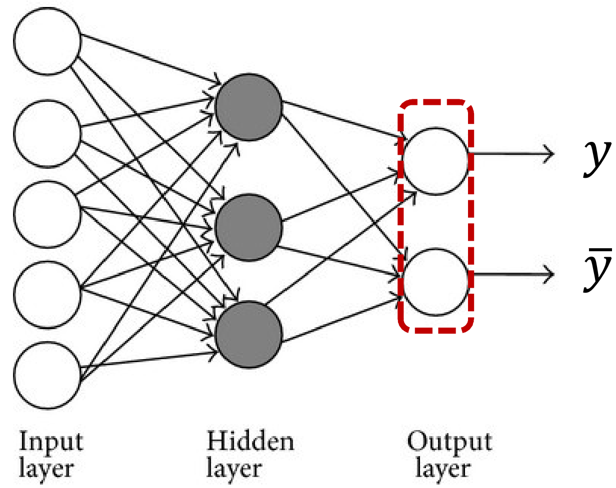




- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
- Output activation: Typically a sigmoid
  - Viewed as the *probability* $P(Y = 1|X)$ of class value 1
    - Indicating the fact that for actual data, in general a feature value X may occur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

# 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* of the desired output
  - Yes: → [1 0]
  - No: → [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:

  $[\text{cat} \;\; \text{dog} \;\; \text{camel} \;\; \text{hat} \;\; \text{flower}]^T$

- For inputs of each of the five classes the desired output is:

  cat: $[1\ 0\ 0\ 0\ 0]^T$

  dog: $[0\ 1\ 0\ 0\ 0]^T$

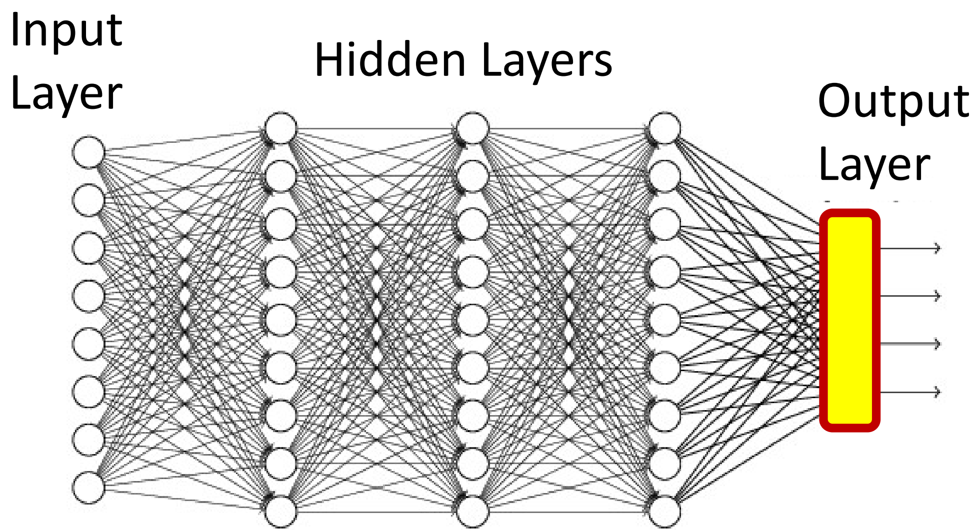
  camel: $[0\ 0\ 1\ 0\ 0]^T$

  hat: $[0\ 0\ 0\ 1\ 0]^T$

  flower: $[0\ 0\ 0\ 0\ 1]^T$

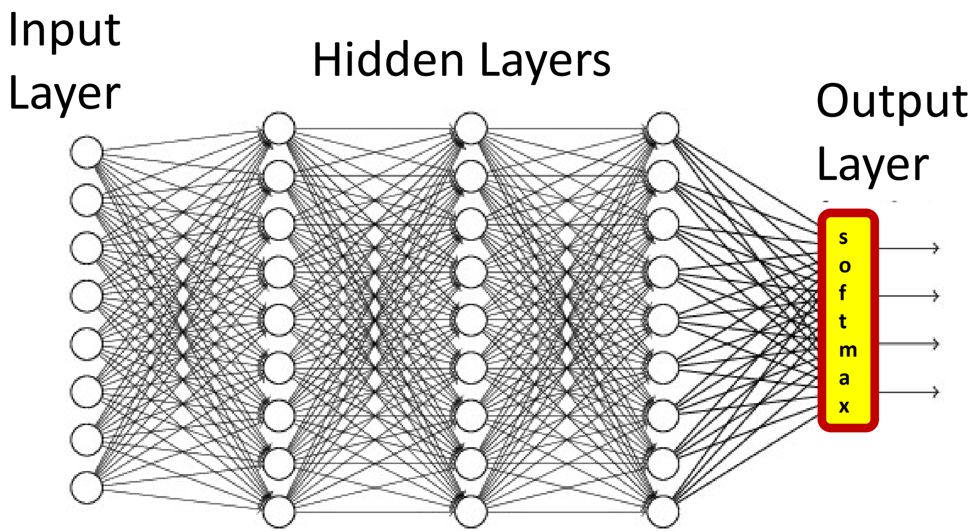
- For an input of any class, we will have a five-dimensional vector output with four zeros and a single 1 at the position of that class
- This is a *one hot vector*

# Multi-class networks




- For a multi-class classifier with N classes, the one-hot representation will have N binary target outputs
  - The **desired** output $d$ 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 classification: Output




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

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

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

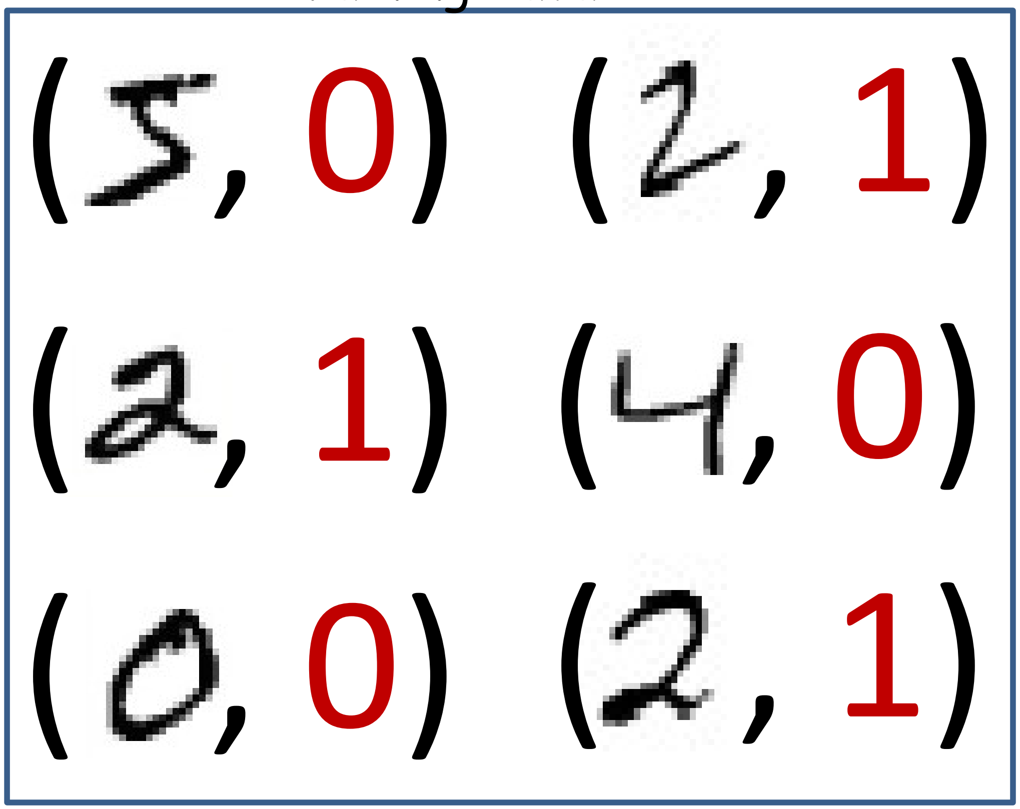
- This can be viewed as the probability $y_i = P(class = i|X)$

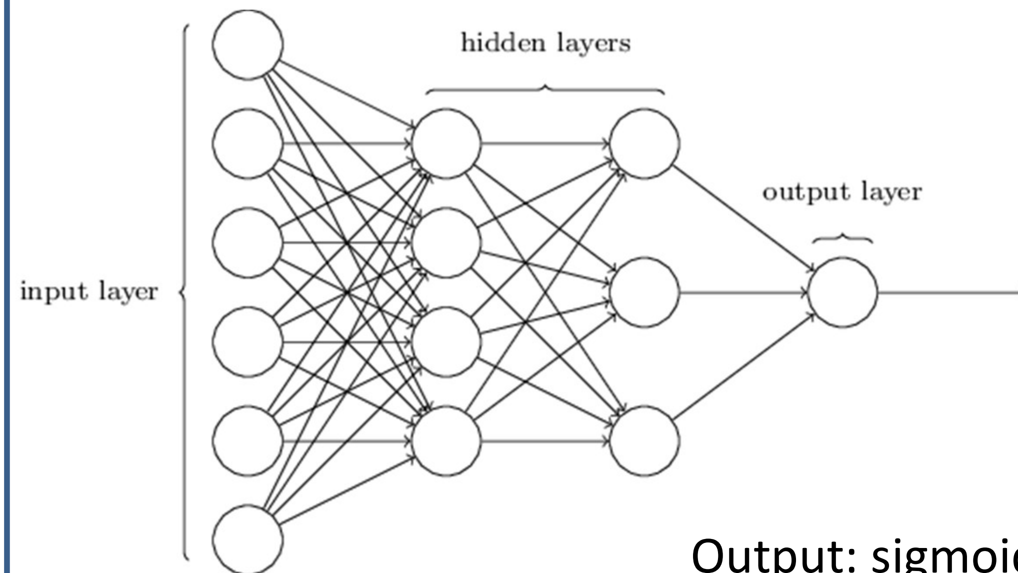
# Inputs and outputs: Typical Problem Statement

- 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

Training data



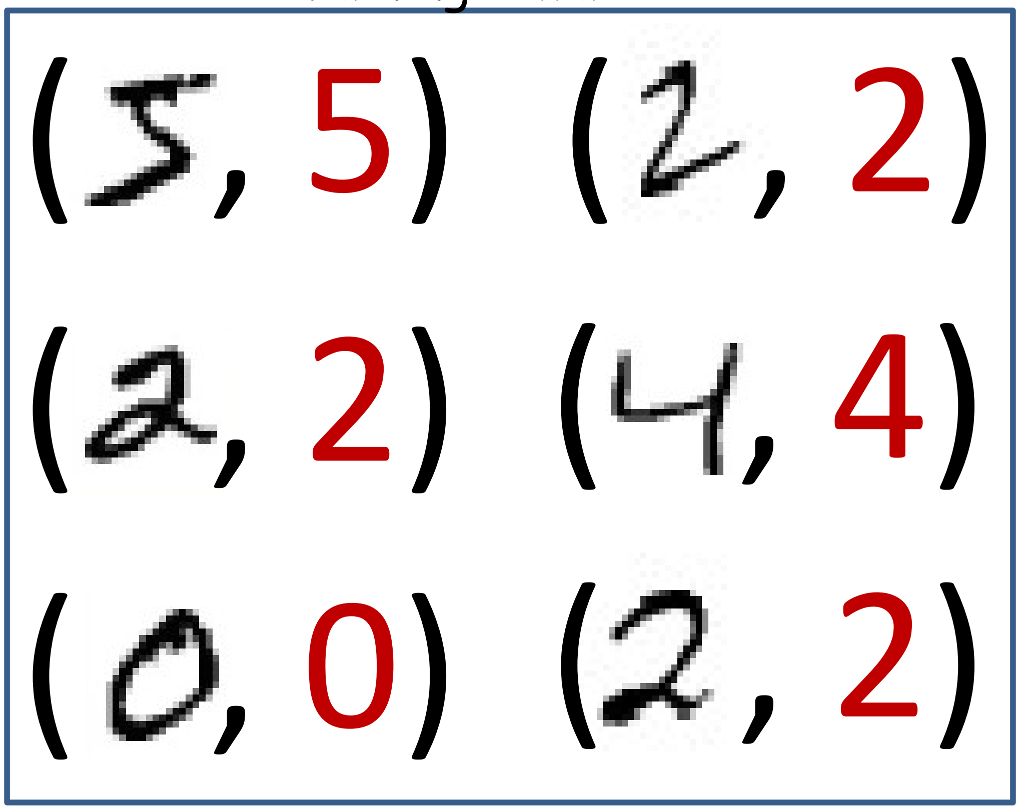

Output: sigmoid

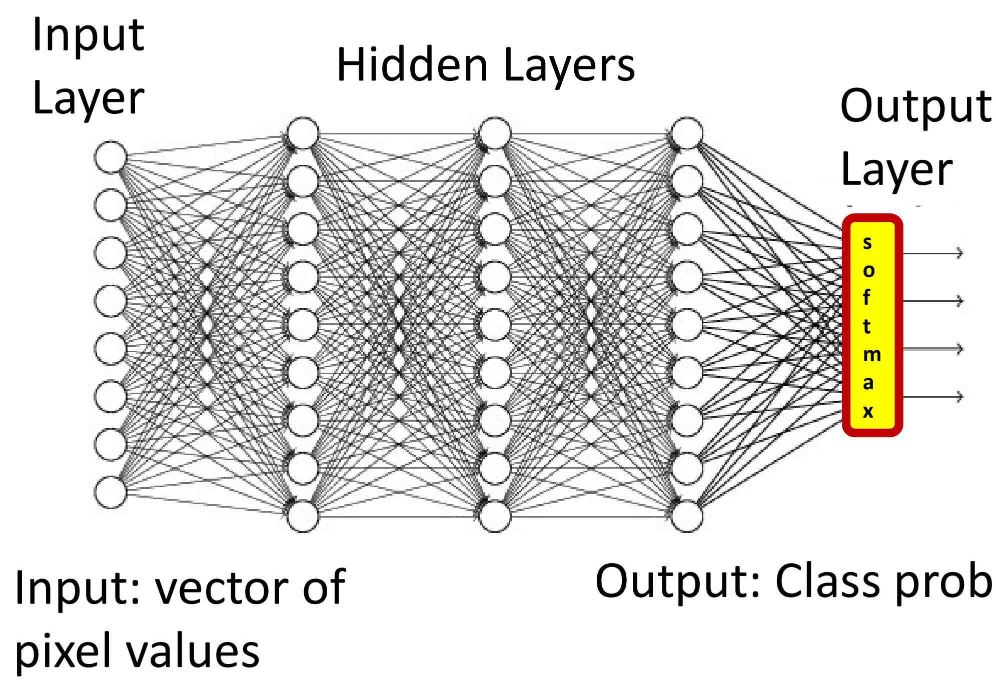
Input: vector of pixel values

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

# Typical Problem statement: multiclass classification

Training data




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

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

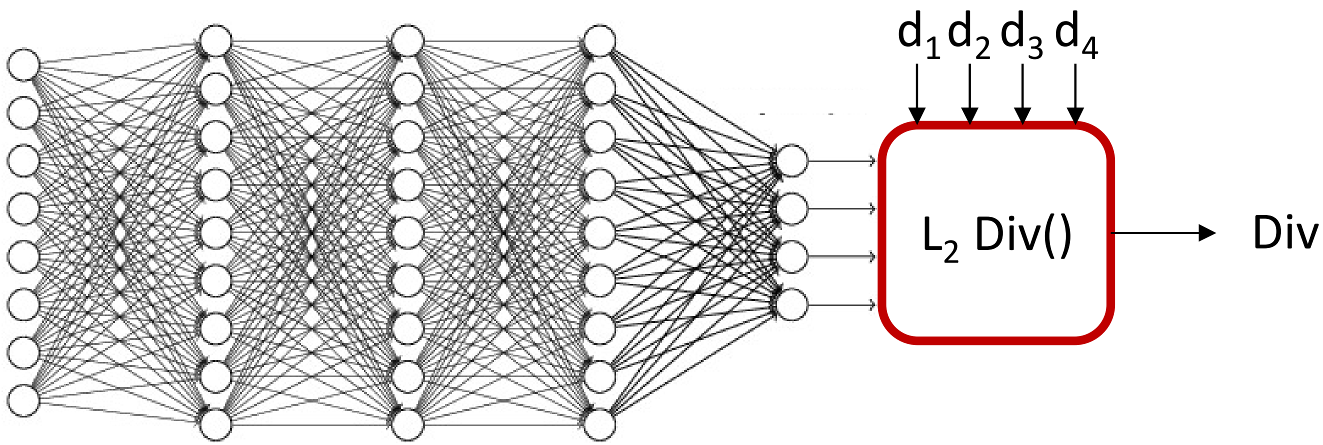
- Minimize the following function

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

What is the divergence div()?

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

# Examples of divergence functions




- For real-valued output vectors, the (scaled) $\mathrm{L}_2$ divergence is popular

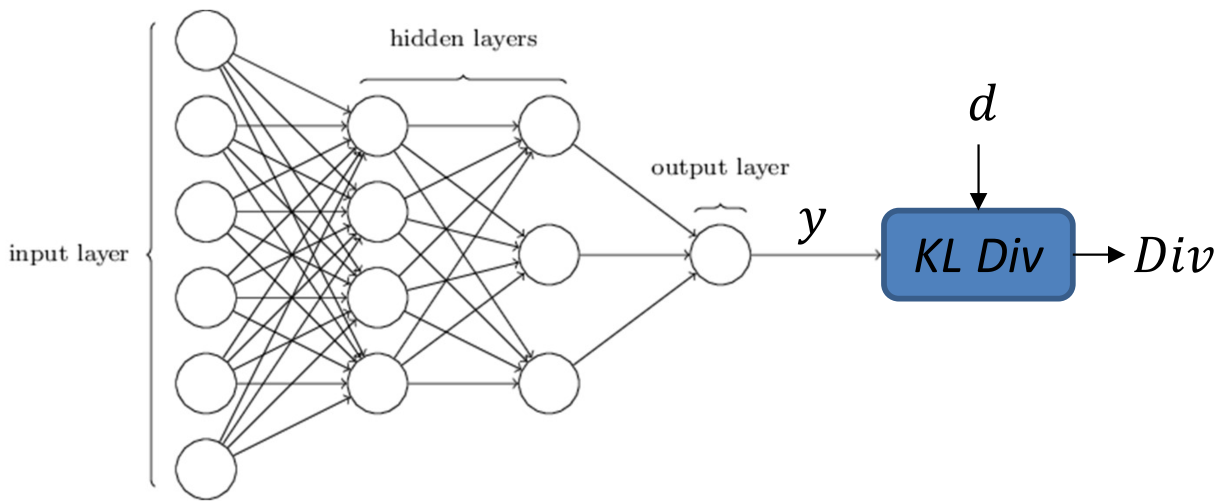
$$Div(Y,d) = \frac{1}{2}\|Y-d\|^2 = \frac{1}{2}\sum_i (y_i - d_i)^2$$

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

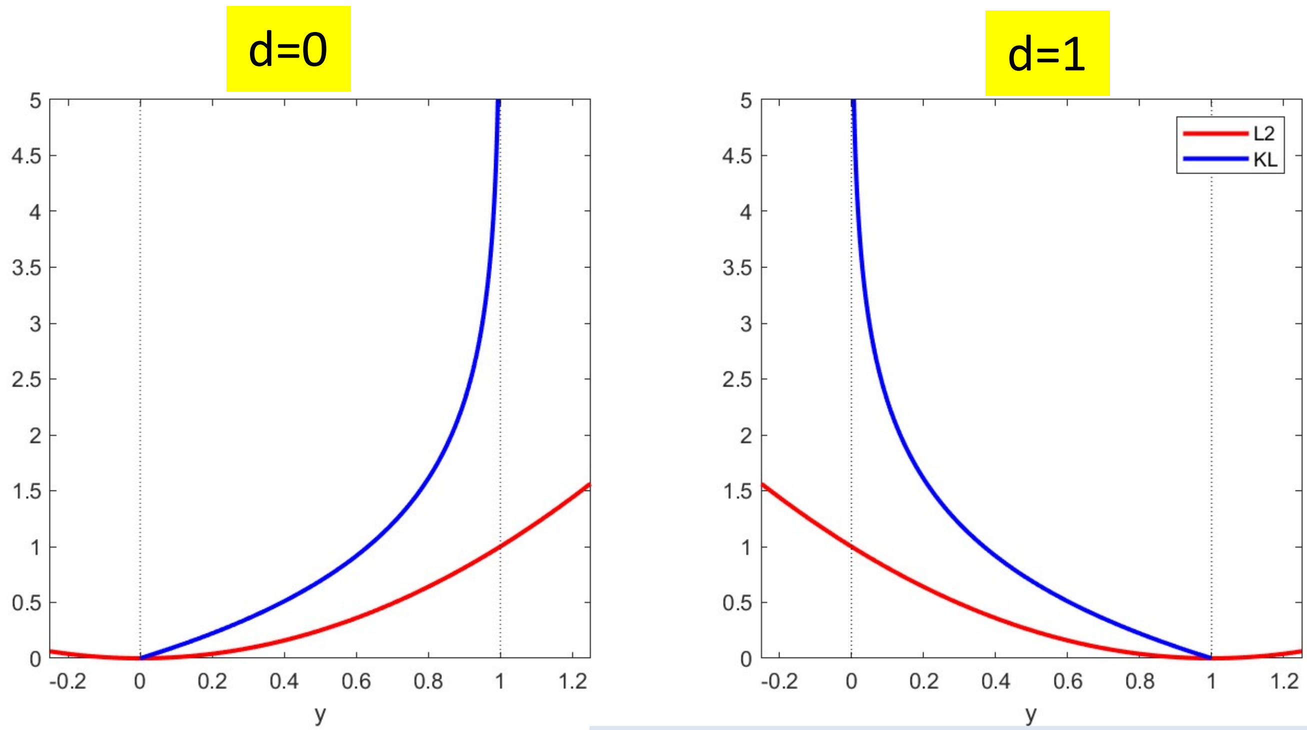
$$Div(Y, d) = -dlogY - (1 - d)\log(1 - Y)$$

  - Minimum when $d = Y$
- Derivative

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

$$\frac{dKLDiv(Y,d)}{dY} = \begin{cases} -\frac{1}{Y} \quad if \;\; d = 1 \\ \frac{1}{1-Y} \quad if \; d = 0 \end{cases}$$

# KL vs L2

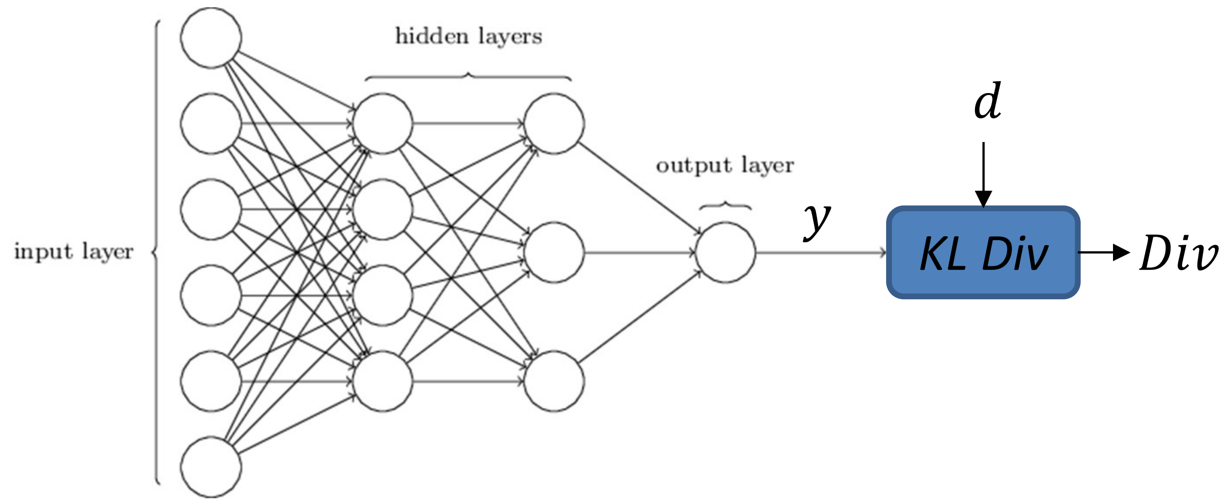



$$L2(Y,d) = (y-d)^2$$

$$KL(Y,d) = -dlogY - (1-d)\log(1-Y)$$

- Both KL and L2 have a minimum when $y$ is the target value of $d$
- KL rises much more steeply away from $d$
  - Encouraging faster convergence of gradient descent
- The derivative of KL is *not* equal to 0 at the minimum
  - It is 0 for L2, though

# For binary classifier




- For binary classifier with scalar output, $Y \in (0,1)$, $d$ is $0/1$, the Kullback Leibler (KL) divergence 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 $\mathrm{d} = Y$

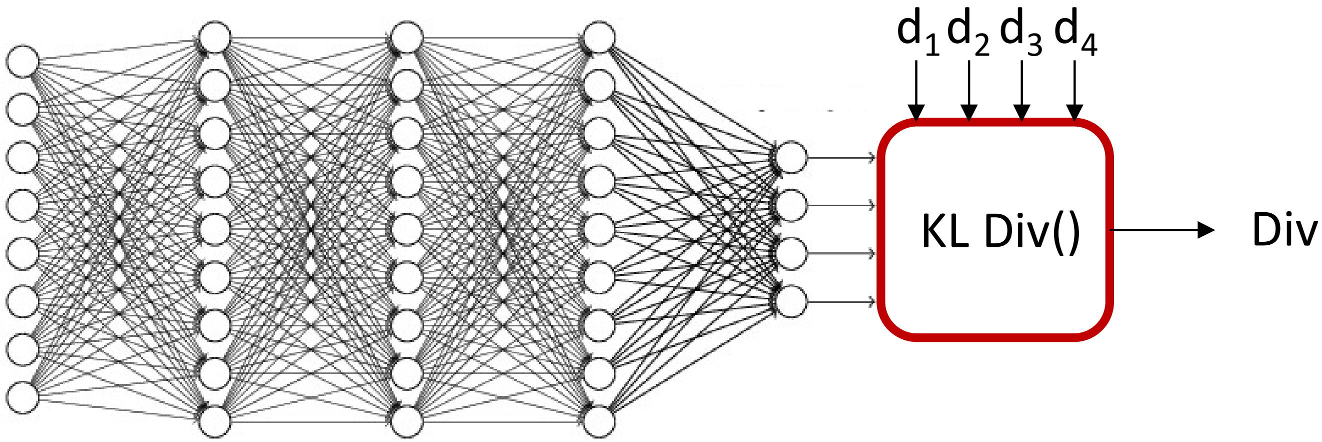
- Derivative

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

Note: when $y = d$ the derivative is *not* 0

*Even though $div() = 0$* (minimum) *when y = d*

# For multi-class classification




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

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

- Derivative

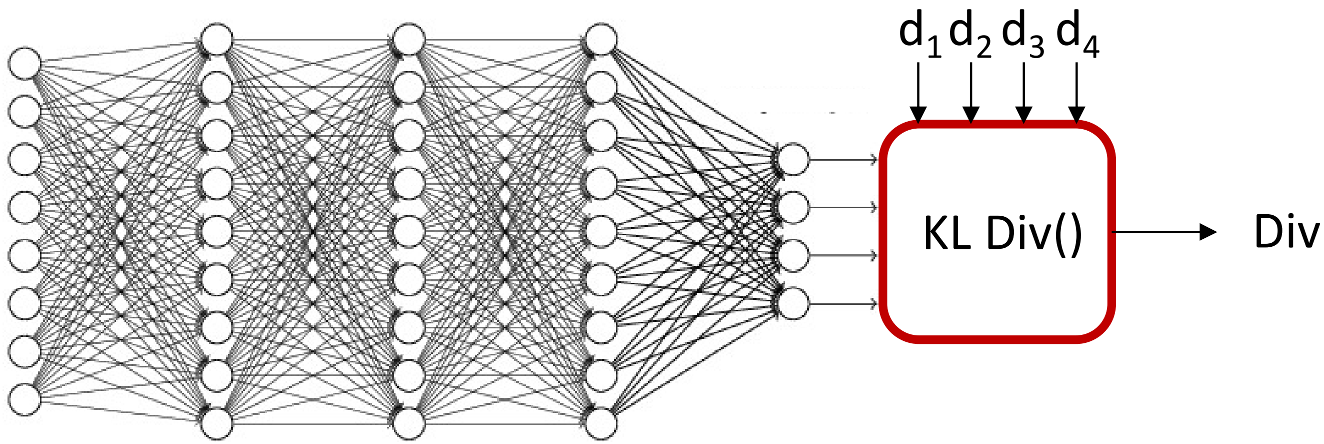
$$\frac{dDiv(Y,d)}{dY_i} = \begin{cases} -\frac{1}{y_c} & for\ the\ c-th\ component \\ 0 & for\ remaining\ component \end{cases}$$

$$\nabla_Y Div(Y,d) = \left[0\ 0\ \dots \frac{-1}{y_c} \dots 0\ 0\right]$$

The slope is negative w.r.t. $y_c$

Indicates *increasing* $y_c$ will *reduce* divergence

# For multi-class classification




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

$$Div(Y,d) = \sum_i d_i \log d_i - \sum_i d_i \log y_i = 0 - \log y_c = -\log y_c$$

Note: when $y\ =\ d$ the derivative is *not* 0

*Even though* $div()\ = 0$ (minimum) *when y = d*

$$\frac{dDiv(Y,d)}{dY_i} = \begin{cases} -\frac{1}{y_c} & for\ the\ c-th\ component \\ 0 & for\ remaining\ component \end{cases}$$

$$\nabla_Y Div(Y,d) = \left[0\ 0\ \ldots \frac{-1}{y_c} \ldots 0\ 0\right]$$

The slope is negative w.r.t. $y_c$

Indicates *increasing* $y_c$ will *reduce* divergence

# 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 $y$:

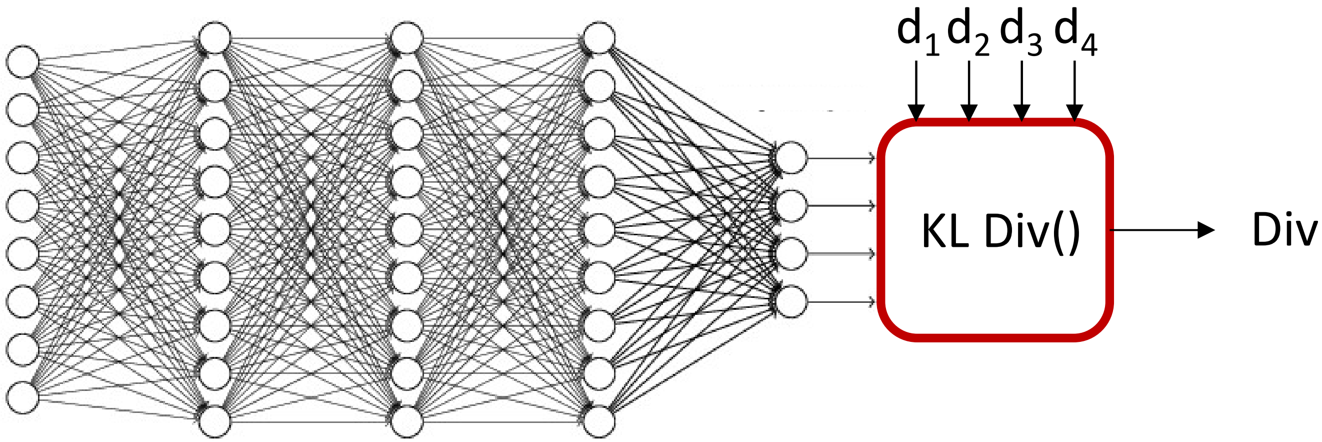
$$Xent(Y,d) = -\sum_i d_i \log y_i$$

- 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
  - since $d$ is the desired output and does not depend on the network, $H(d)$ does not depend on the net
  - In fact, for one-hot $d$, $H(d) = 0$ (and KL = Xent)
- We will generally minimize to the *cross-entropy* loss rather than the KL divergence
  - The Xent is *not* a divergence, and although it attains its minimum when $y = d$, its minimum value is not 0

# "Label smoothing"



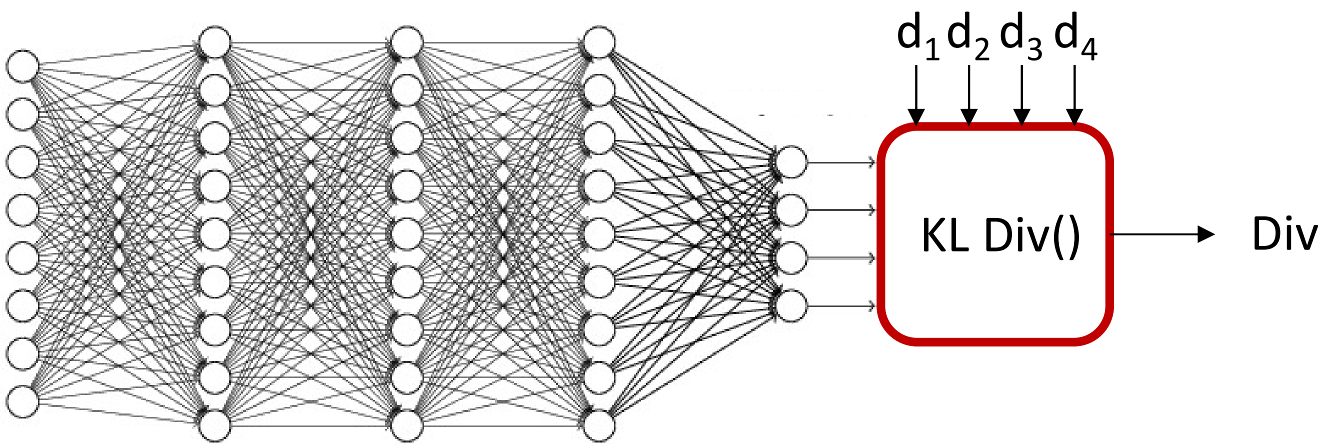

- It is sometimes useful to set the target output to $[\epsilon \;\; \epsilon \; \ldots \; (1-(K-1)\epsilon) \; \ldots \; \epsilon \;\; \epsilon \;\; \epsilon]$ with the value $1-(\mathrm{K}-1)\epsilon$ in the $c$-th position (for class $c$) and $\epsilon$ elsewhere for some small $\epsilon$
  - "Label smoothing" -- aids gradient descent
- The KL divergence remains:

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

- Derivative

$$\frac{dDiv(Y,d)}{dY_i} = \begin{cases} -\dfrac{1-(K-1)\epsilon}{y_c} & for\ the\ c-th\ component \\ -\dfrac{\epsilon}{y_i} for\ remaining\ components & \end{cases}$$

# "Label smoothing"




- It is sometimes useful to set the target output to $[\epsilon \;\; \epsilon \;...\; (1-(K-1)\epsilon) \;...\; \epsilon \;\; \epsilon \;\; \epsilon]$ with the value $1-(\mathrm{K}-1)\epsilon$ in the $c$-th position (for class $c$) and $\epsilon$ elsewhere for some small $\epsilon$
  - "Label smoothing" -- aids gradient descent
- The KL divergence remains:

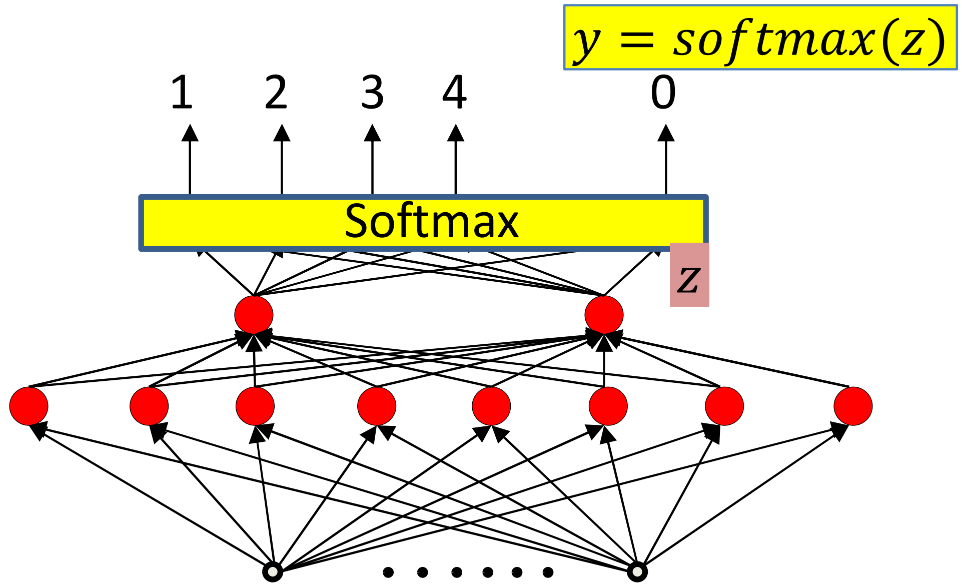
$$Div(Y,d) = \sum_i d_i \log d_i - \sum_i d_i \log y_i$$

- Derivative

$$\frac{dDiv(Y,d)}{dY_i} = \begin{cases} -\dfrac{1-(K-1)\epsilon}{y_c} & for\ the\ c-th\ component \\ -\dfrac{\epsilon}{y_i} & for\ remaining\ components \end{cases}$$

Negative derivatives encourage *increasing* the probabilities of *all* classes, including *incorrect* classes! (Seems wrong, no?)

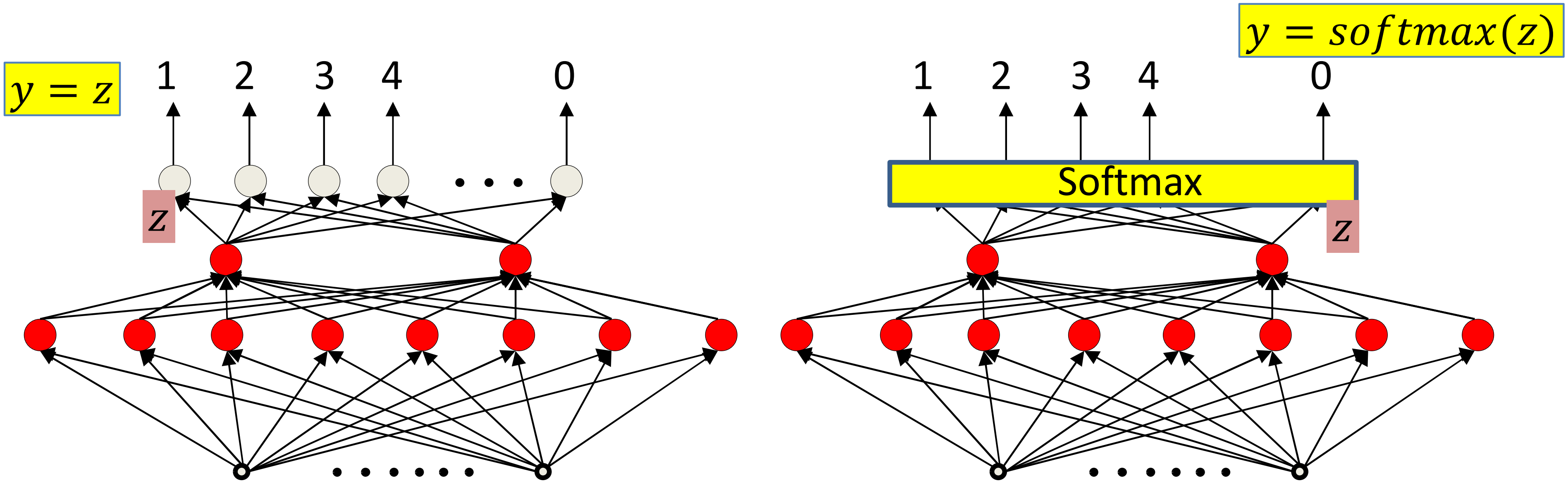
# The derivative of the KL divergence




- The softmax is computed on affine values z to obtain output probabilities y
- The derivative of the KL divergence between the actual output y and target output is as given earlier
- However, the derivative of the KL divergence w.r.t. the *affine* value z at the input of the softmax is just the error

$$\nabla_{\mathbf{z}} KL(\boldsymbol{y}, \boldsymbol{d}) = (\boldsymbol{y} - \boldsymbol{d})^{\top}$$

# A note on derivatives




- Note: For both regression models with linear output layer and L2 divergence, and classification models with softmax output layer and KL divergence the gradient w.r.t. the final affine value of the network is just the error

$$\nabla_{\mathbf{z}} \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{d}\|^2 = (\boldsymbol{y} - \boldsymbol{d})^\top$$

$$\nabla_{\mathbf{z}} KL(\boldsymbol{y}, \boldsymbol{d}) = (\boldsymbol{y} - \boldsymbol{d})^\top$$

# Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

- Minimize the following function

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

ALL TERMS HAVE BEEN DEFINED

# Poll 5

- Select all that are correct
  - The gradient of the loss will always be 0 or close to 0 at a minimum
  - The gradient of the loss may be 0 or close to 0 at a minimum
  - The gradient of the loss may have large magnitude at a minimum
  - If the gradient is not 0 at a minimum, it must be a local minimum

# Poll 5

- Select all that are correct
  - The gradient of the loss will always be 0 or close to 0 at a minimum
  - **The gradient of the loss may be 0 or close to 0 at a minimum**
  - **The gradient of the loss may have large magnitude at a minimum**
  - If the gradient is not 0 at a minimum, it must be a local minimum

# Story so far

- Neural nets are universal approximators

- Neural networks are trained to approximate functions by adjusting their parameters to minimize the average divergence between their actual output and the desired output at a set of "training instances"
  - Input-output samples from the function to be learned
  - The average divergence is the "Loss" to be minimized

- To train them, several terms must be defined
  - The network itself
  - The manner in which inputs are represented as numbers
  - The manner in which outputs are represented as numbers
    - As numeric vectors for real predictions
    - As one-hot vectors for classification functions
  - The divergence function that computes the error between actual and desired outputs
    - L2 divergence for real-valued predictions
    - KL divergence for classifiers

# Next Class

- Backpropagation