

# CSC 561: Neural Networks and Deep Learning

## Learning (part 2)

Marco Alvarez

Department of Computer Science and Statistics  
University of Rhode Island

Spring 2025



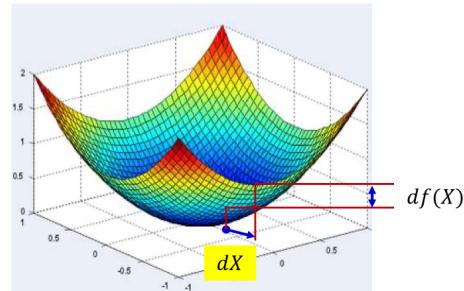
# Neural Networks

## Learning the network: Backprop

11-785, Spring 2024

Lecture 4

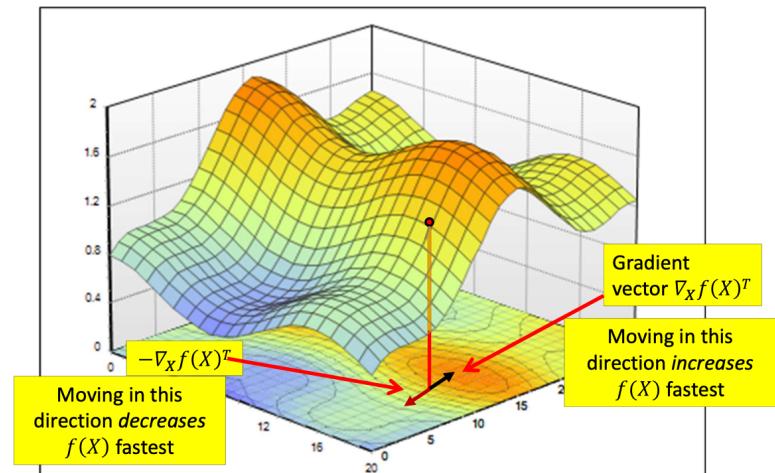
### Gradient of a scalar function of a vector



- 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  $X$   
$$df(X) = \nabla_X f(X) dX$$
  - $\nabla_X f(X) = \begin{bmatrix} \frac{\partial f(X)}{\partial x_1} & \frac{\partial f(X)}{\partial x_2} & \dots & \frac{\partial f(X)}{\partial x_n} \end{bmatrix}$
- The **gradient** is the transpose of the derivative  $\nabla_X f(X)^T$ 
  - A column vector of the same dimensionality as  $X$

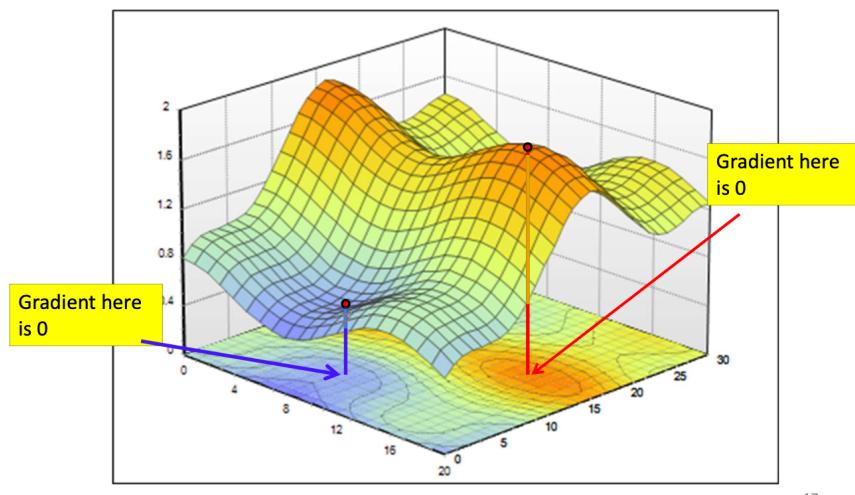
10

### Gradient

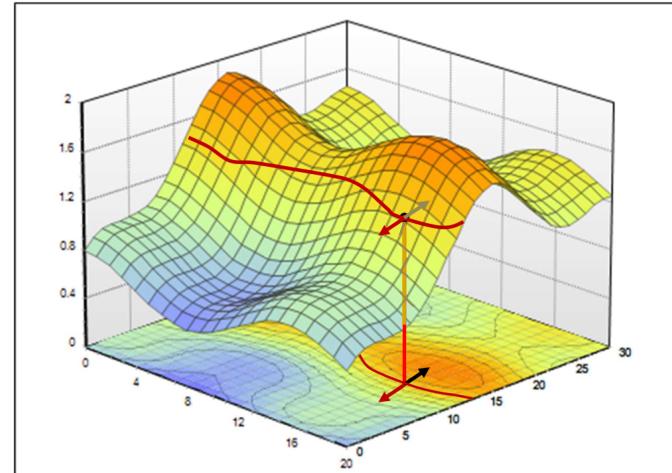


16

## Gradient



## Properties of Gradient: 2



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

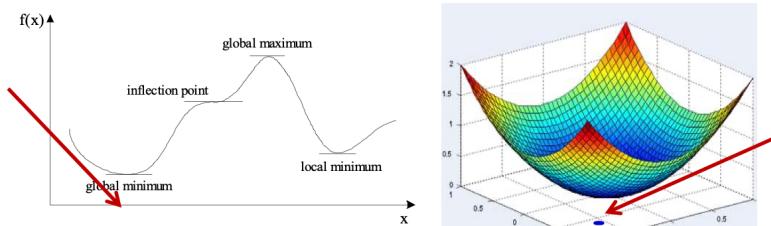
## The Hessian

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

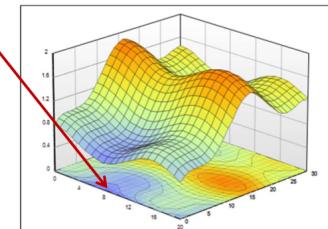
$$\nabla_x^2 f(x_1, \dots, x_n) := \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

19

## The problem of optimization

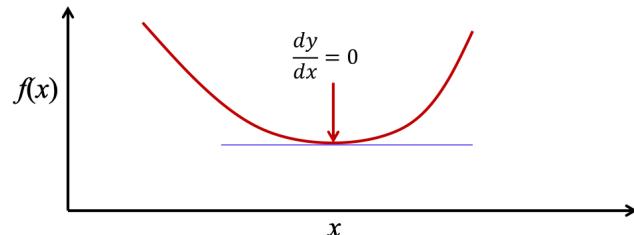


- General problem of optimization: Given a function  $f(x)$  of some variable  $x$  ...
- Find the value of  $x$  where  $f(x)$  is minimum



22

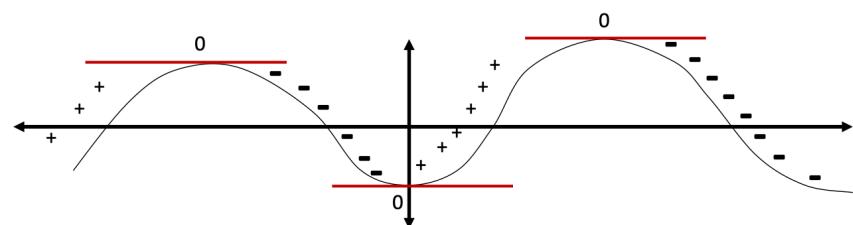
## Finding the minimum of a function



- 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?

23

## Turning Points

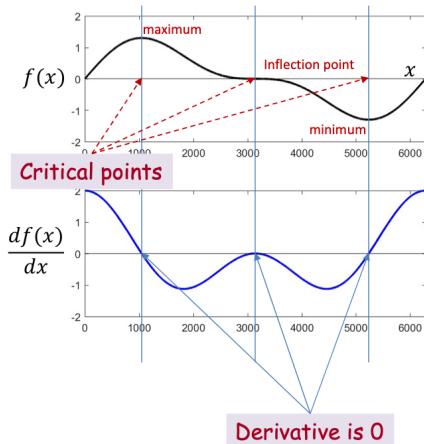


- Both *maxima* and *minima* have zero derivative
- Both are turning points

26

0

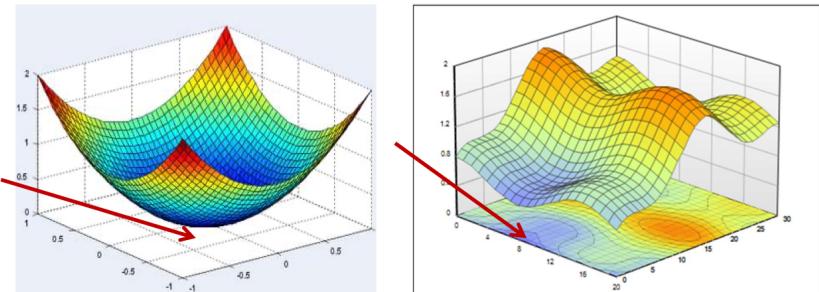
## A note on derivatives of functions of single variable



- 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

30

## 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

33

2

## Unconstrained Minimization of function (Multivariate)

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

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

- 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

34  
3

## Unconstrained Minimization of function (Example)

- Minimize

$$f(x_1, x_2, x_3) = (x_1)^2 + x_1(1-x_2) + (x_2)^2 - x_2x_3 + (x_3)^2 + x_3$$

- Gradient

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

35  
4

## 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

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

36  
5

## 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

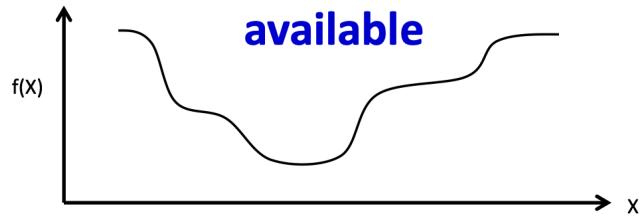
$$\lambda_1 = 3.414, \lambda_2 = 0.586, \lambda_3 = 2$$

- 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

37  
6

## Closed Form Solutions are not always available

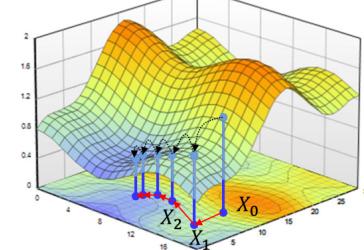
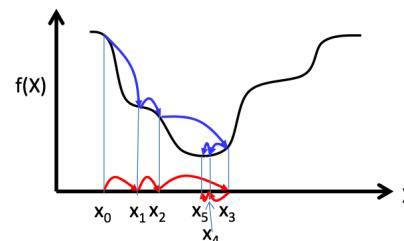


- Often it is not possible to simply solve  $\nabla_X f(X) = 0$ 
  - The function to minimize/maximize may have an intractable form
- In these situations, iterative solutions are used
  - Begin with a “guess” for the optimal  $X$  and refine it iteratively until the correct value is obtained

38

7

## Iterative solutions

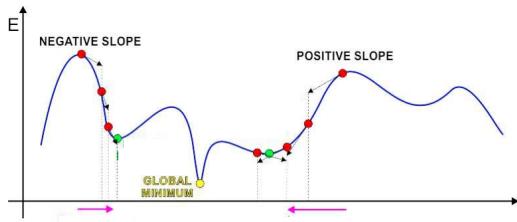


- Iterative solutions
  - Start from an initial guess  $X_0$  for the optimal  $X$
  - Update the guess towards a (hopefully) “better” value of  $f(X)$
  - Stop when  $f(X)$  no longer decreases
- Problems:
  - Which direction to step in
  - How big must the steps be

39

8

## The Approach of Gradient Descent

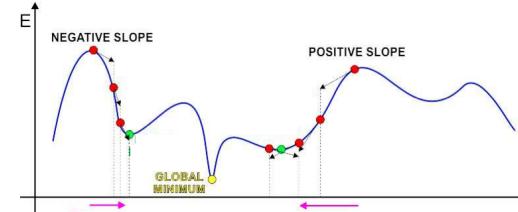


- Iterative solution:
  - Start at some point
  - Find direction in which to shift this point to decrease error
    - This can be found from the derivative of the function
      - A negative derivative  $\rightarrow$  moving right decreases error
      - A positive derivative  $\rightarrow$  moving left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

40

9

## The Approach of Gradient Descent

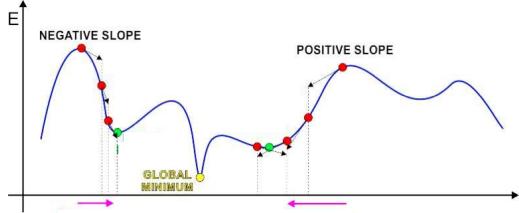


- Iterative solution: Trivial algorithm
  - Initialize  $x^0$
  - While  $f'(x^k) \neq 0$ 
    - If  $\text{sign}(f'(x^k))$  is positive:
 
$$x^{k+1} = x^k - \text{step}$$
    - Else
 
$$x^{k+1} = x^k + \text{step}$$

41

0

## The Approach of Gradient Descent



- Iterative solution: Trivial algorithm
  - Initialize  $x^0$
  - While  $f'(x^k) \neq 0$
$$x^{k+1} = x^k - \text{sign}(f'(x^k)) \cdot \text{step}$$
- Identical to previous algorithm

42

1

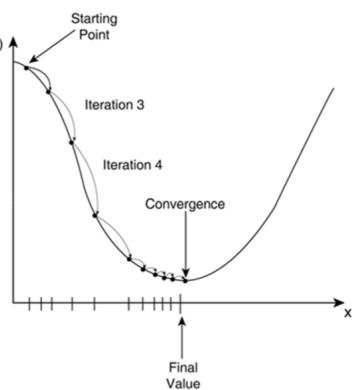
## Gradient descent convergence criteria

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

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

- Or

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



48

3

## 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*

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

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

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

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

47

2

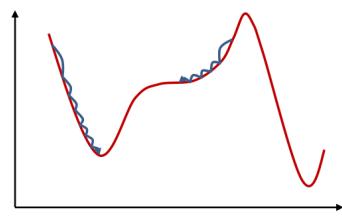
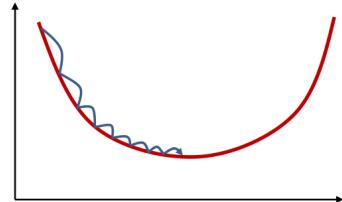
## Overall Gradient Descent Algorithm

- Initialize:
  - $x^0$
  - $k = 0$
- do
  - $x^{k+1} = x^k - \eta^k \nabla_x f(x^k)^T$
  - $k = k + 1$
- while  $|f(x^{k+1}) - f(x^k)| > \varepsilon$

49

4

## 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

50

5

- Returning to our problem from our detour..

53

6

## Problem Statement

- Given a training set of input-output pairs  $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- Minimize the following function

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

w.r.t.  $W$

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

54

7

## Gradient Descent to train a network

- Initialize:

–  $W^0$

–  $k = 0$

do

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

–  $k = k + 1$

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

11-755/18-797

55

8

## Problem Setup: Things to define

- Given a training set of input-output pairs  $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- Minimize the following function

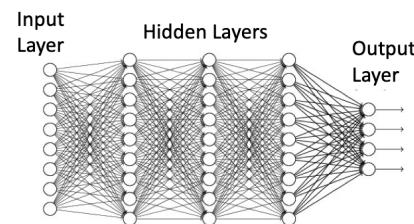
$$\text{Loss}(W) = \frac{1}{T} \sum_i \text{div}(f(X_i; W), d_i)$$

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

61

9

## Typical network

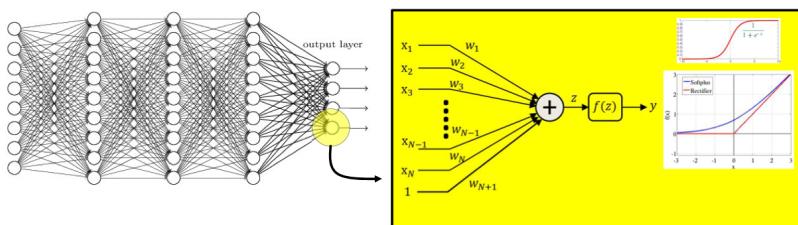


- We assume a “layered” network for simplicity
  - Each “layer” of neurons only gets inputs from the earlier layer(s) and outputs signals only to later layer(s)
  - We will refer to the inputs as the **input layer**
    - No neurons here – the “layer” simply refers to inputs
  - We refer to the outputs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

63

0

## 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

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

We will assume this unless otherwise specified

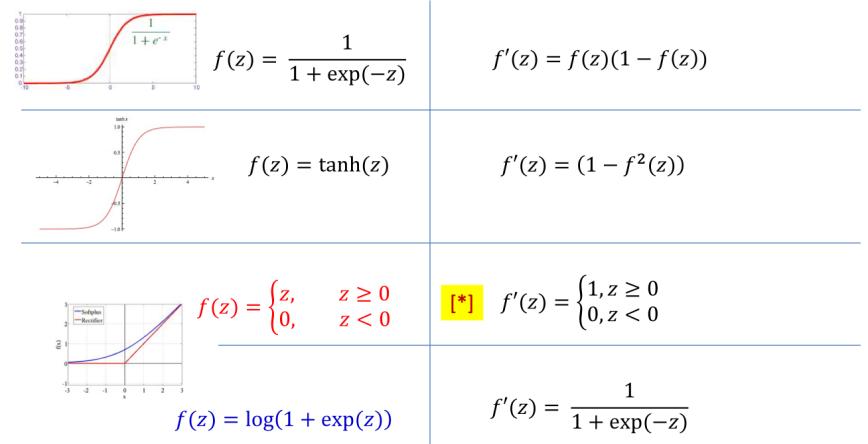
Parameters are weights  $w_i$  and bias  $b$

– More generally: any differentiable function

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

65

## Activations and their derivatives

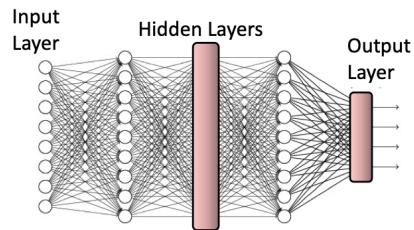


- Some popular activation functions and their derivatives

66

2

## Vector Activations



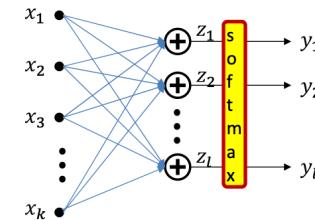
- We can also have neurons that have *multiple coupled outputs*

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

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

67

## Vector activation example: Softmax



- Example: Softmax *vector* activation

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

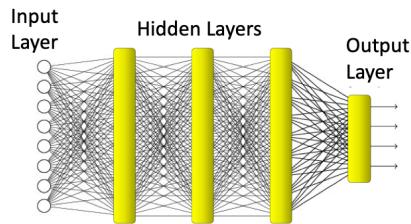
$$y = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

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

68

4

## Typical network

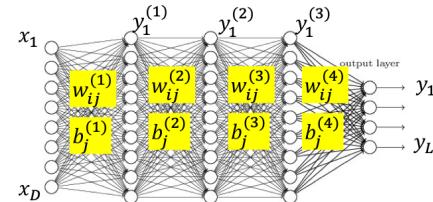


- In a layered network, each layer of perceptrons can be viewed as a single vector activation

70

5

## Notation



- The input layer is the 0<sup>th</sup> layer
- We will represent the output of the i-th perceptron of the k<sup>th</sup> 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-1<sup>th</sup> layer and the jth unit of the k<sup>th</sup> layer as  $w_{ij}^{(k)}$ 
  - The bias to the jth unit of the k<sup>th</sup> layer is  $b_j^{(k)}$

71

6

## Problem Setup: Things to define

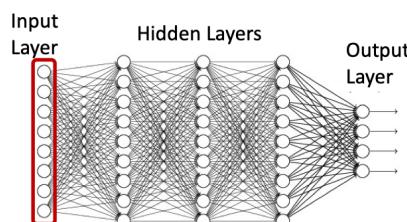
- Given a training set of input-output pairs  $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- What are these input-output pairs?

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

73

7

## Representing the input

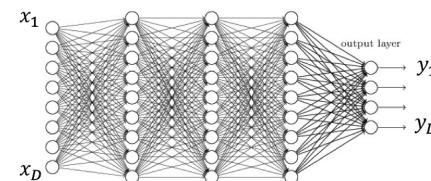


- Vectors of numbers
  - (or may even be just a scalar, if input layer is of size 1)
  - E.g. vector of pixel values
  - E.g. vector of speech features
  - E.g. real-valued vector representing text
    - We will see how this happens later in the course
  - Other real valued vectors

75

9

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

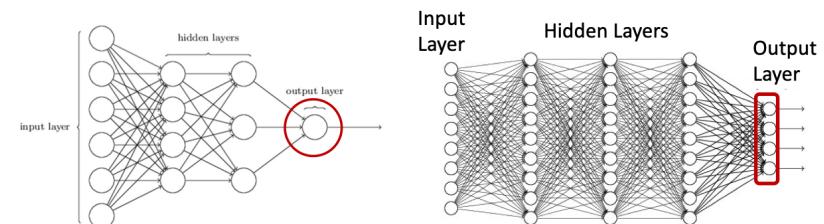


- Given a training set of input-output pairs  $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- $X_n = [x_{n1}, x_{n2}, \dots, x_{nD}]^T$  is the nth input vector
- $d_n = [d_{n1}, d_{n2}, \dots, d_{nL}]^T$  is the nth desired output
- $Y_n = [y_{n1}, y_{n2}, \dots, y_{nL}]^T$  is the nth vector of *actual*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

74

8

## Representing the output

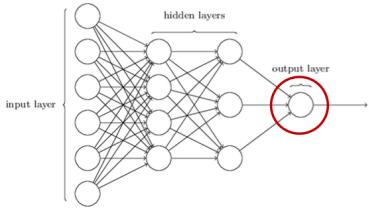


- If the desired *output* is **real-valued**, no special tricks are necessary
  - Scalar Output : single output neuron
    - $d = \text{scalar (real value)}$
  - Vector Output : as many output neurons as the dimension of the desired output
    - $d = [d_1 \ d_2 \ \dots \ d_L]$  (vector of real values)

76

0

## Representing the output

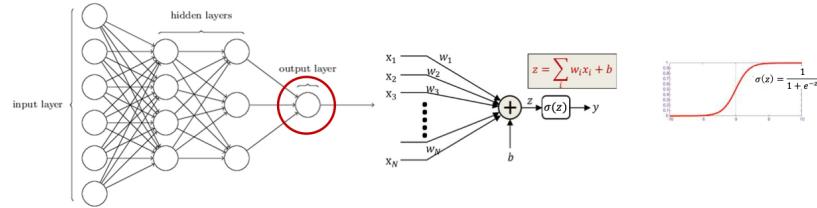


- If the desired output is **binary** (is this a cat or not), use a simple 1/0 representation of the desired output
  - 1 = Yes, it's a cat
  - 0 = No, it's not a cat.

77

1

## Representing the output

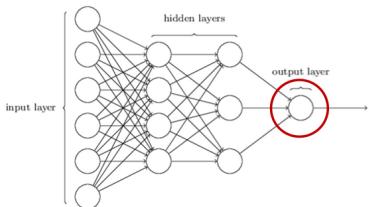


- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
- Output activation: Typically, a sigmoid
  - Viewed as the *probability*  $P(Y = 1|X)$  of class value 1
  - Indicating the fact that for actual data, in general a feature value X may occur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

78

2

## 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:  $\rightarrow [1 0]$
  - No:  $\rightarrow [0 1]$
- The output explicitly becomes a 2-output softmax

79

3

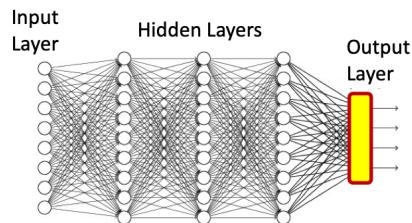
## Multi-class output: One-hot representations

- 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 five-dimensional vector output with four zeros and a single 1 at the position of that class
- This is a *one hot vector*

80

4

## Multi-class networks

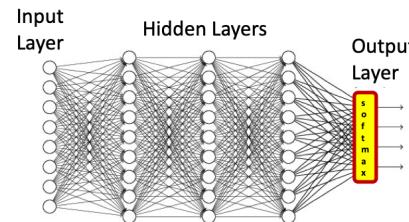


- For a multi-class classifier with  $N$  classes, the one-hot representation will have  $N$  binary target outputs
  - The **desired** output  $d$  is an  $N$ -dimensional binary vector
- The neural network's **actual** output too must ideally be binary ( $N-1$  zeros and a single 1 in the right place)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

81

5

## 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  $y_i = P(\text{class} = i | X)$

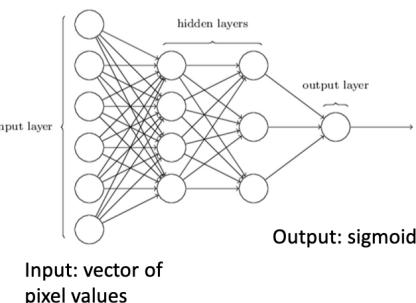
82

6

## Typical Problem statement: binary classification

Training data

(5, 0)	(2, 1)
(2, 1)	(4, 0)
(0, 0)	(2, 1)



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

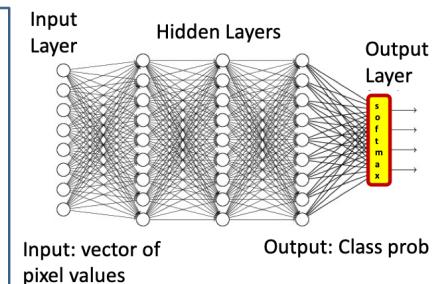
84

7

## Typical Problem statement: multiclass classification

Training data

(5, 5)	(2, 2)
(2, 2)	(4, 4)
(0, 0)	(2, 2)



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

85

8

## Problem Setup: Things to define

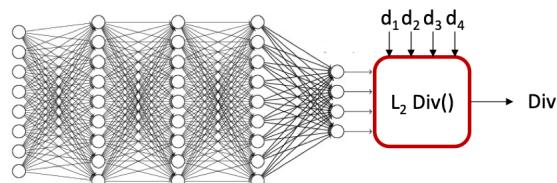
- Given a training set of input-output pairs  $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- Minimize the following function

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

What is the divergence  $div()$ ?

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

## Examples of divergence functions



- For real-valued output vectors, the (scaled) L<sub>2</sub> 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

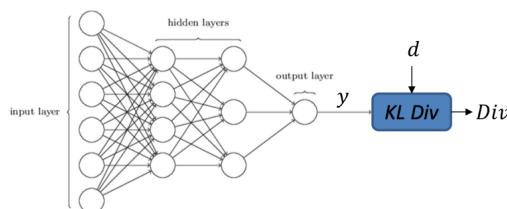
$$\frac{dDiv(Y, d)}{dy_i} = (y_i - d_i)$$

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

88

0

## 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  $d = Y$

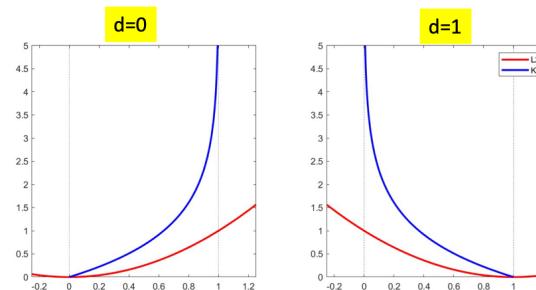
- Derivative

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

89

$$\frac{dKLDiv(Y, d)}{dy} = \begin{cases} -\frac{1}{Y} & \text{if } d = 1 \\ \frac{1}{1-Y} & \text{if } d = 0 \end{cases}$$

## KL vs L2



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

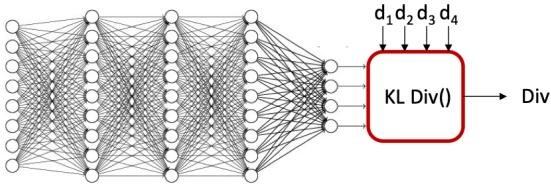
$$KL(Y, d) = -d \log Y - (1 - d) \log(1 - Y)$$

- 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

90

2

## 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

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

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

The slope is negative w.r.t.  $y_c$   
Indicates increasing  $y_c$  will reduce divergence

92

## 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  $y$ :

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

- When the desired target  $d_i$  is one-hot,  $\sum_i d_i \log d_i = 0$ 
  - KL and cross-entropy are identical
  - $Xent(Y, d) = -\log y_i$
  - Minimizing cross-entropy simply maximizes the probability of the target class
- We will continue discussing in terms of KL, but the discussion applies directly to Cross-entropy as well

94

4

## 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  $y$ :

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

- More generally, the cross entropy is merely the KL - entropy of  $d$

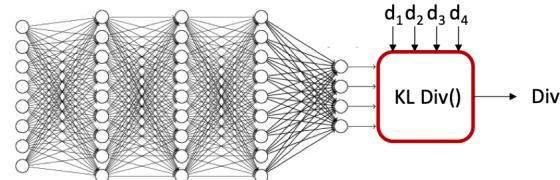
$$Xent(Y, d) = KL(Y, d) - \sum_i d_i \log d_i = KL(Y, d) - H(d)$$

- 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

95

5

## “Label smoothing”



- It is sometimes useful to set the target output to  $[\epsilon \ \epsilon \ \dots (1 - (K - 1)\epsilon) \ \dots \epsilon \ \epsilon \ \epsilon]$  with the value  $1 - (K - 1)\epsilon$  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

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

96

6