

# CSC 561: Neural Networks and Deep Learning

## Learning

Marco Alvarez

Department of Computer Science and Statistics  
University of Rhode Island

Spring 2025



# Neural Networks

## Learning the network: Part 1

11-785, Spring 2025

Lecture 3

1

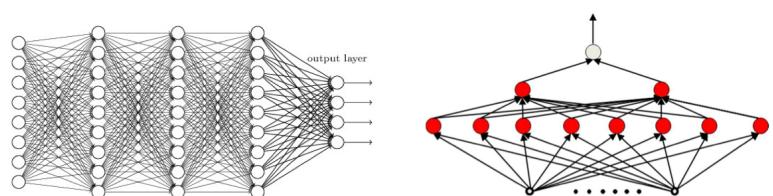
2

## Topics for the day

- The problem of learning
- The perceptron rule for learning individual perceptrons
  - And its inapplicability to multi-layer perceptrons
- Greedy solutions for classification networks: ADALINE and MADALINE
- Learning through Empirical Risk Minimization
- Intro to function optimization and gradient descent

2

## First: the structure of the network



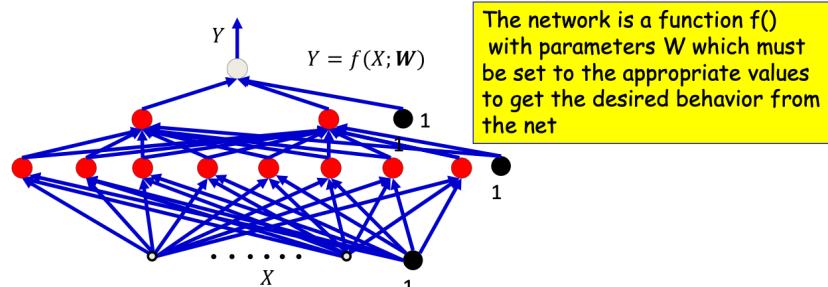
- We will assume a *feed-forward* network
  - No loops: Neuron outputs do not feed back to their inputs directly or indirectly
  - Loopy networks are a future topic
- **Part of the design of a network: The architecture**
  - How many layers/neurons, which neuron connects to which and how, etc.
- **For now, assume the architecture of the network is capable of representing the needed function**

3

10

11

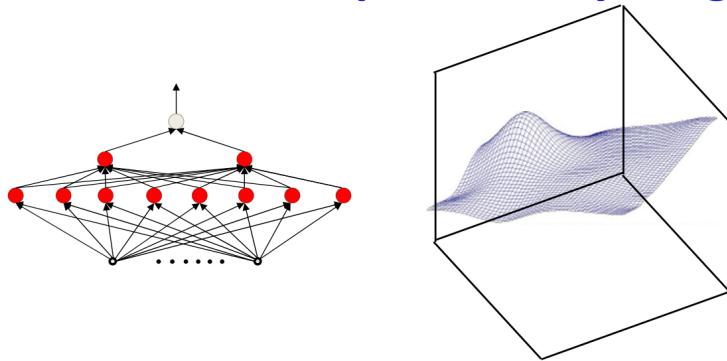
## What we learn: The parameters of the network



- Given: the architecture of the network
- The parameters of the network: The weights and biases
  - The weights associated with the blue arrows in the picture
- Learning the network**: Determining the values of these parameters such that the network computes the desired function

11

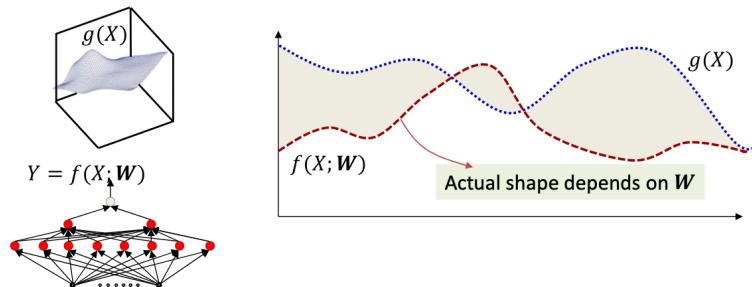
## The MLP can represent anything



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

13

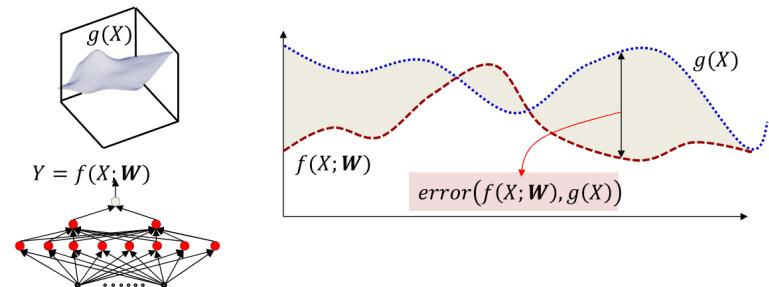
## How to learn a network?



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

22

## How to learn a network?



- The shaded area

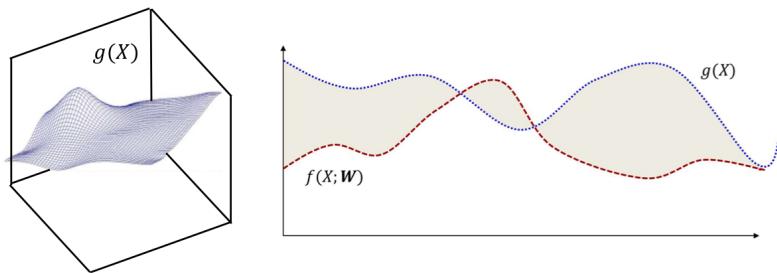
$$\text{totalerr}(\mathbf{W}) = \int_{-\infty}^{\infty} \text{error}(f(X; \mathbf{W}), g(X)) dX$$

- The optimal  $\mathbf{W}$

$$\widehat{\mathbf{W}} = \operatorname{argmin}_{\mathbf{W}} \text{totalerr}(\mathbf{W})$$

23

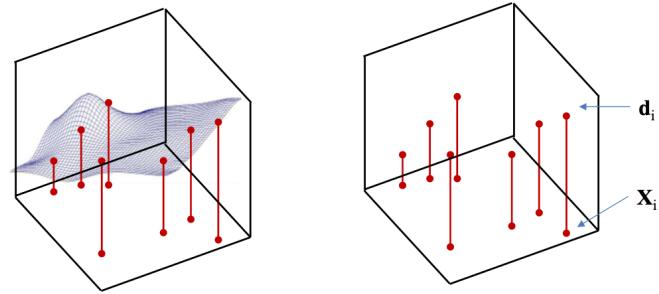
## Problem: $g(X)$ is unknown



- Function  $g(X)$  must be fully specified in order to compute
 
$$\int_{-\infty}^{\infty} \text{error}(f(X; \mathbf{W}), g(X)) dX$$
  - Known *everywhere*, i.e. for every input  $X$
- In practice we will not have such specification

24

## Sampling the function

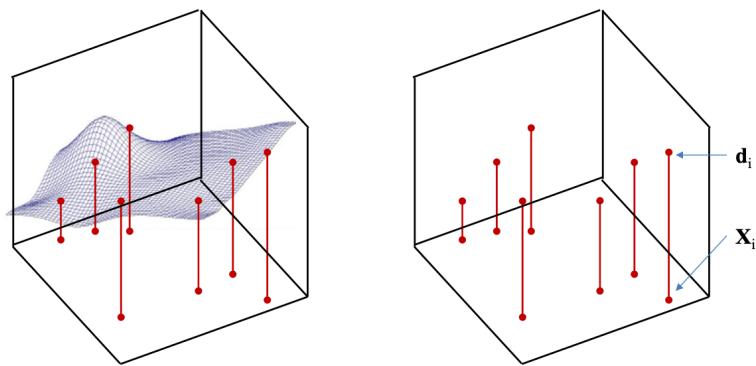


- Sample  $g(X)$ 
  - Basically, get input-output pairs for a number of samples of input  $X_i$ 
    - Many samples  $(X_i, d_i)$ , where  $d_i = g(X_i) + \text{noise}$
- Very easy to do in most problems: just gather training data
  - E.g. set of images and their class labels
  - E.g. speech recordings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

25

0

## Drawing samples

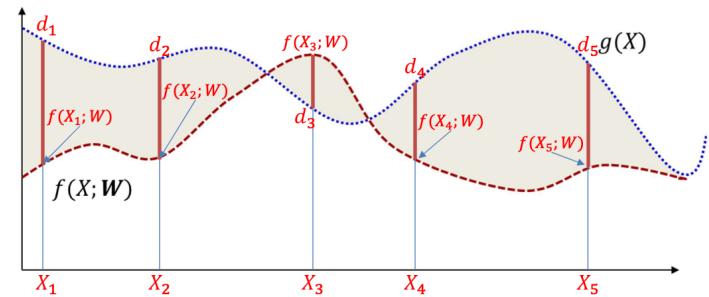


- We must **learn** the *entire* function from these few examples
  - The “training” samples

26

1

## The Empirical error



- The **empirical estimate** of the error is the *average* error over the training samples

$$\text{EmpiricalError}(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^N \text{error}(f(X_i; \mathbf{W}), d_i)$$

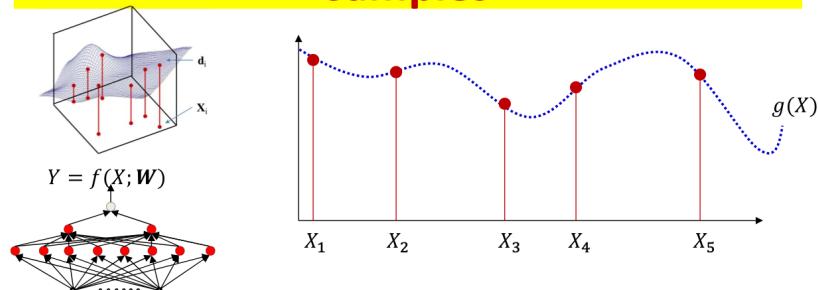
- Estimate network parameters to minimize this average error instead

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \text{EmpiricalError}(\mathbf{W})$$

27

2

## Learning the function from training samples



- Aim: Find the network parameters that “fit” the training points exactly  
 $\mathbf{W}$ :  $\text{EmpiricalError}(\mathbf{W}) = 0$ 
  - Assuming network architecture is sufficient for such a fit
  - Assuming unique output  $d$  at any  $\mathbf{X}$
- And hopefully the resulting function is also correct where we *don't* have training samples

28

3

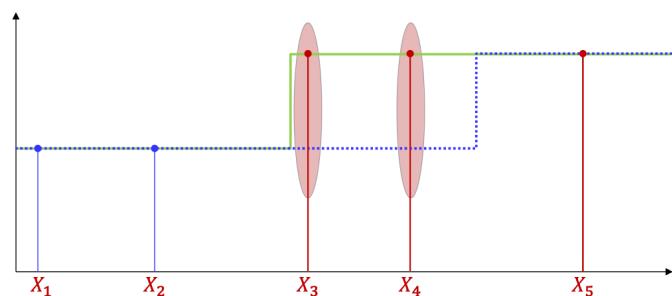
## Let's begin with a simple task

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

32

4

## The *Empirical Classification* error



- The obvious error metric in a classifier is binary
  - The classifier is either right (error=0) or wrong (error=1)
    - Either  $f(\mathbf{X}; \mathbf{W}) = d$ , or  $f(\mathbf{X}; \mathbf{W}) \neq d$

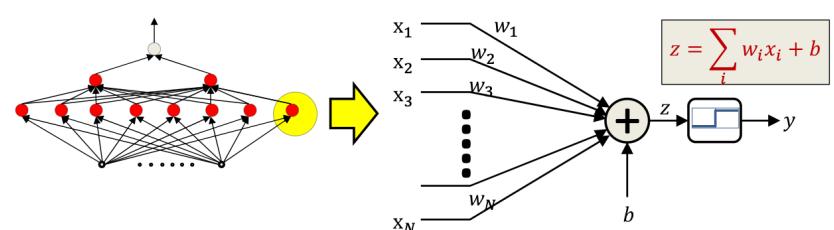
$$\text{EmpiricalError}(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^N \mathbb{1}(f(\mathbf{X}_i; \mathbf{W}) \neq d_i)$$

- **Learning the classifier:** Minimizing the count of misclassifications

33

5

## History: The original MLP

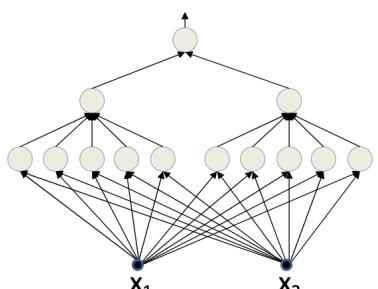


- The original MLP as proposed by Rosenblatt: a network of threshold units
  - But how do you train it?
    - Given only “training” instances of input-output pairs

34

6

## Learning a multilayer perceptron



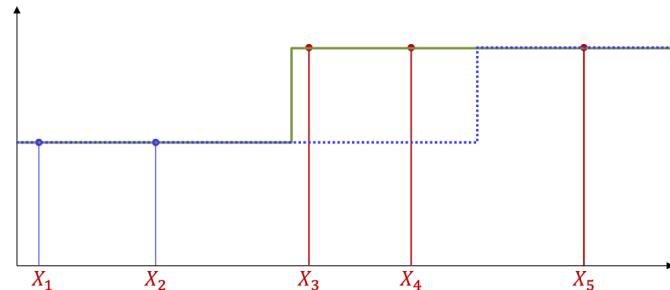
Training data only specifies input and output of network  
Intermediate outputs (outputs of individual neurons) are not specified

- Training this network using the perceptron rule is a combinatorial optimization problem
- We don't know the outputs of the individual intermediate neurons in the network for any training input
- Must also determine the correct output for each neuron for every training instance
- At least exponential (in inputs) time complexity!!!!!!

77

7

## The problem

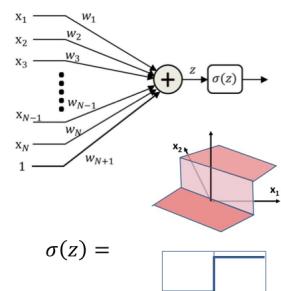
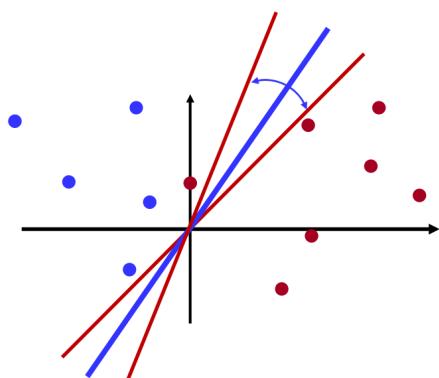


- Our binary error metric is not useful
  - To improve the classifier we must move the blue dotted line left
  - But if we move it only slightly, moving it either right or left results in no change in error

95

8

## Why this problem?

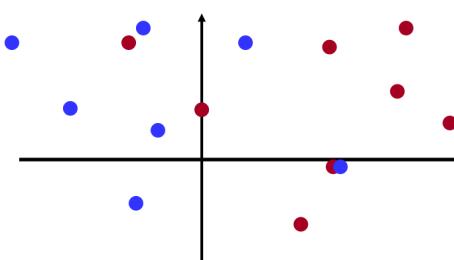


- The perceptron is a flat function with zero derivative everywhere, except at 0 where it is non-differentiable
  - You can vary the weights a lot without changing the error
  - There is no indication of which direction to change the weights to reduce error

96

9

## A second problem: What we *actually* model

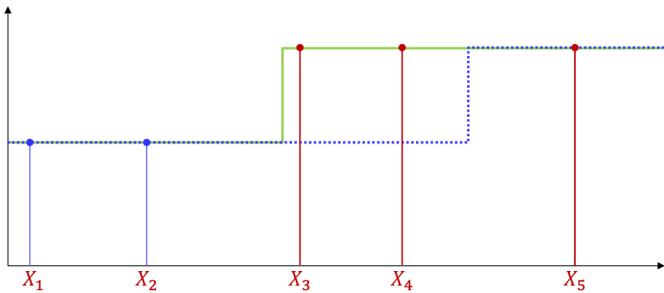


- Real-life data are rarely clean
  - Not linearly separable
  - Rosenblatt's perceptron learning rule wouldn't work in the first place

98

9

## The solution

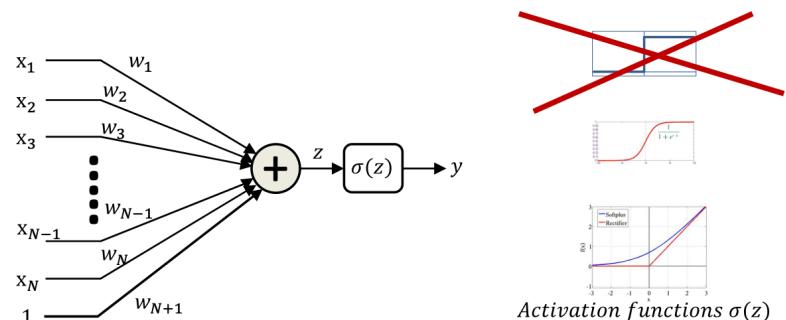


- Change our way of computing the mismatch such that modifying the classifier slightly lets us know if we are going the right way or not
  - This requires changing both, our activation functions, and the manner in which we evaluate the mismatch between the classifier output and the target output
  - Our mismatch function will now not actually count errors, but a *proxy* for it

99

1

## Solution: Differentiable activation

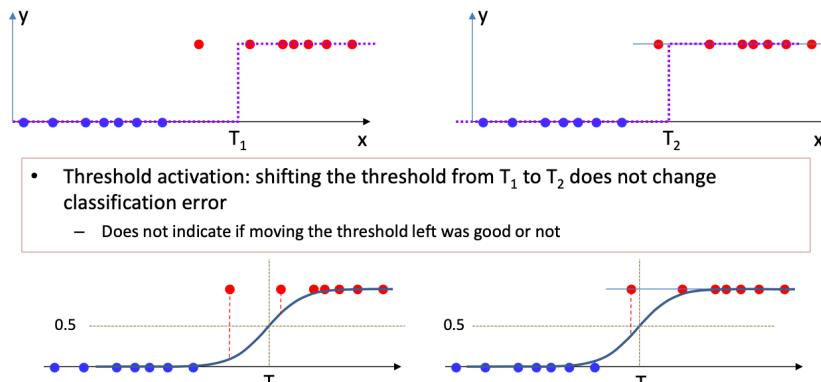


- Let's make the neuron differentiable, *with non-zero derivatives over much of the input space*
  - Small changes in weight can result in non-negligible changes in output
  - This enables us to estimate the parameters using gradient descent techniques..

100

2

## Differentiable Mismatch function

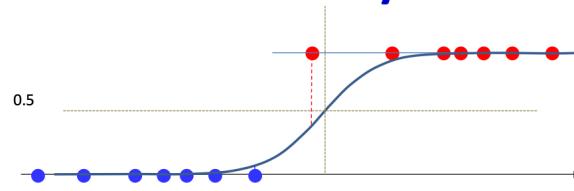


- Smooth, continuously varying activation: Classification based on whether the output is greater than 0.5 or less
  - Quantify *how much* the output differs from the desired target value (0 or 1)
  - Moving the function left or right changes this quantity, even if the classification error itself doesn't change

101

3

## The two key requirements for learnability

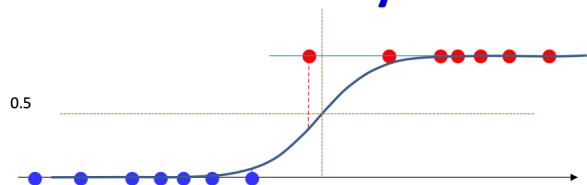


- Continuously varying activation
  - Differentiable
- Continuously varying error function
  - Also differentiable

104

4

## The two key requirements for learnability

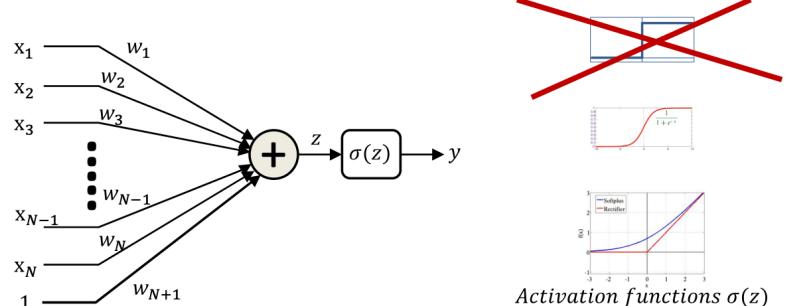


- Continuously varying activation
  - Differentiable
- Continuously varying error function
  - Also differentiable

105

5

## Continuous Activations

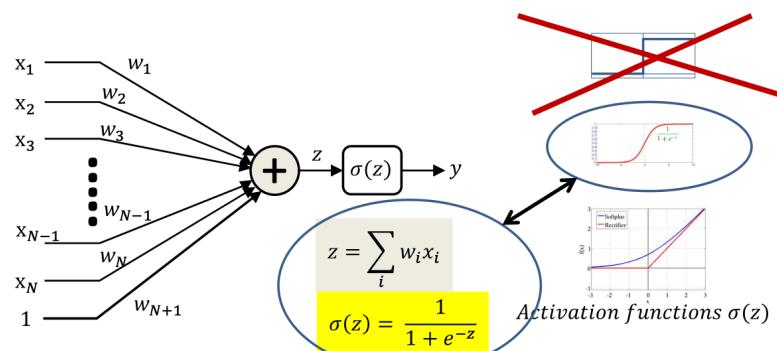


- Replace the threshold activation with continuous graded activations
  - E.g. RELU, softplus, sigmoid etc.
- The activations are *differentiable* almost everywhere
  - Have derivatives almost everywhere
  - And have “subderivatives” at non-differentiable corners
    - Bounds on the derivative that can substitute for derivatives in our setting
    - More on these later

106

6

## The sigmoid activation is special

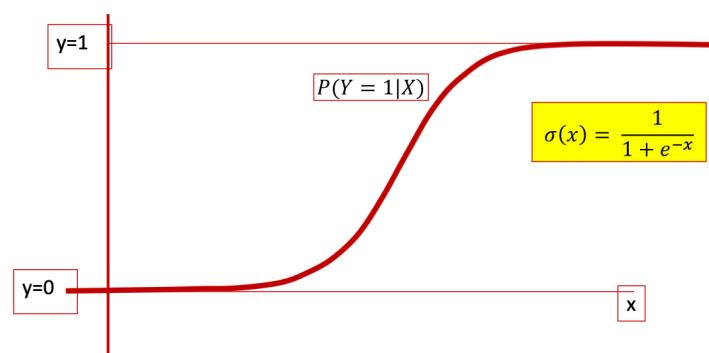


- This particular one has a nice interpretation
- It can be interpreted as  $P(y = 1|x)$

107

7

## The logistic regression model

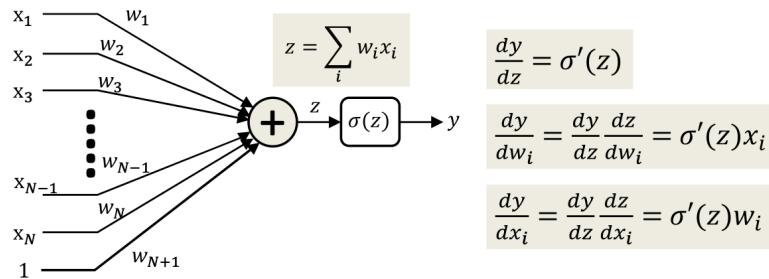


- Class 1 becomes increasingly probable going left to right
  - Very typical in many problems

123

8

## Perceptrons with differentiable activation functions

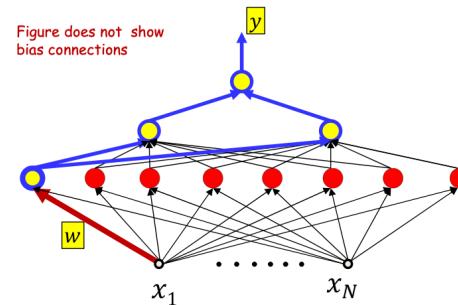


- $\sigma(z)$  is a differentiable function of  $z$ 
  - $\frac{d\sigma(z)}{dz}$  is well-defined and finite for all  $z$
- Using the chain rule,  $y$  is a differentiable function of both inputs  $x_i$  and weights  $w_i$
- This means that we can compute the change in the output for small changes in either the input or the weights

126

9

## Overall network is differentiable

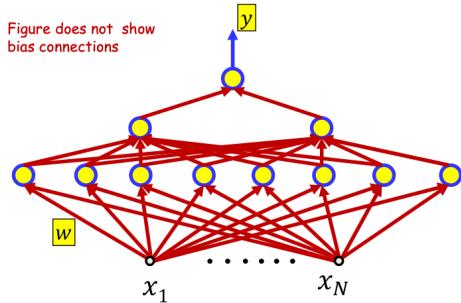


- Every individual perceptron is differentiable w.r.t its inputs and its weights (including “bias” weight)
  - Small changes in the parameters result in measurable changes in output
- Using the chain rule can compute how small perturbations of a parameter change the output of the network
  - The network output is differentiable with respect to the parameter

127

0

## Overall function is differentiable

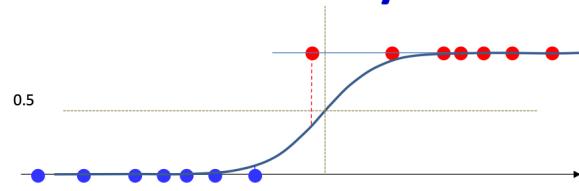


- By extension, the overall function is differentiable w.r.t every parameter in the network
  - We can compute how small changes in the parameters change the output
    - For non-threshold activations the derivative are finite and generally non-zero
- We will derive the actual derivatives using the chain rule later

128

1

## The two key requirements for learnability

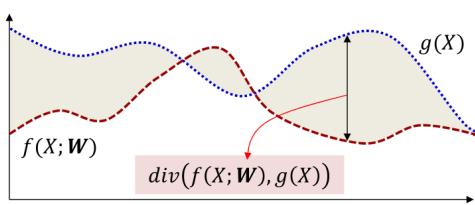
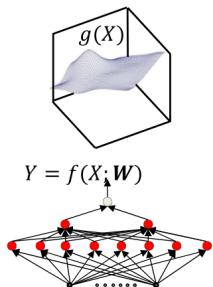


- Continuously varying activation
  - Differentiable
- Continuously varying error function
  - Also differentiable

129

2

## The error function

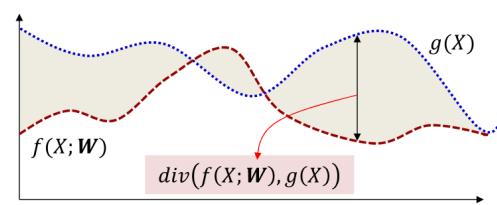
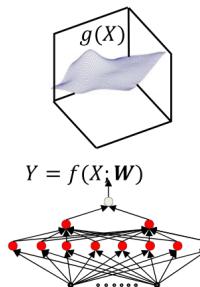


- Define a *divergence* function  $\text{div}(f(X; \mathbf{W}), g(X))$  with the following properties
  - $\text{div}(f(X; \mathbf{W}), g(X)) = 0$  if  $f(X; \mathbf{W}) = g(X)$
  - $\text{div}(f(X; \mathbf{W}), g(X)) > 0$  if  $f(X; \mathbf{W}) \neq g(X)$
  - $\text{div}(f, g)$  is differentiable with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

130

3

## How to learn a network



- When  $f(X; \mathbf{W})$  has the capacity to exactly represent  $g(X)$

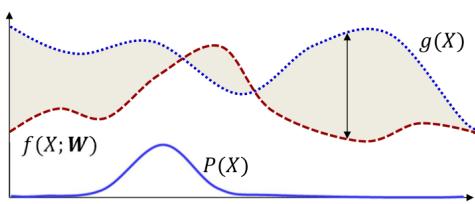
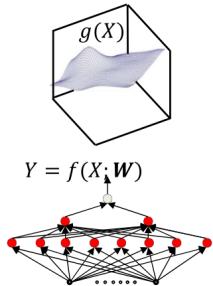
$$\widehat{\mathbf{W}} = \operatorname{argmin}_{\mathbf{W}} \int_X \text{div}(f(X; \mathbf{W}), g(X)) dX$$

- $\text{div}()$  is a divergence function that goes to zero when  $f(X; \mathbf{W}) = g(X)$

131

4

## Minimizing expected divergence



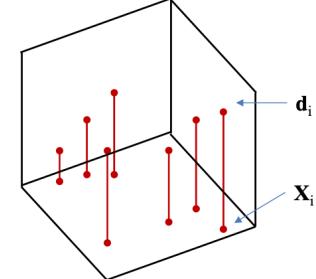
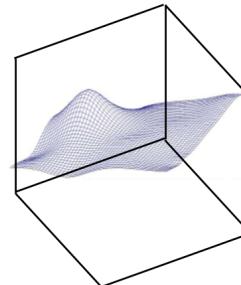
- More generally, assuming  $X$  is a random variable

$$\begin{aligned} \widehat{\mathbf{W}} &= \operatorname{argmin}_{\mathbf{W}} \int_X \text{div}(f(X; \mathbf{W}), g(X)) P(X) dX \\ &= \operatorname{argmin}_{\mathbf{W}} E[\text{div}(f(X; \mathbf{W}), g(X))] \end{aligned}$$

132

5

## The Empirical risk



- The *expected divergence* (or risk) is the average divergence over the entire input space

$$E[\text{div}(f(X; \mathbf{W}), g(X))] = \int_X \text{div}(f(X; \mathbf{W}), g(X)) P(X) dX$$

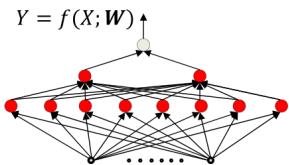
- The *empirical estimate* of the expected risk is the *average* divergence over the samples

$$E[\text{div}(f(X; \mathbf{W}), g(X))] \approx \frac{1}{N} \sum_{i=1}^N \text{div}(f(\mathbf{x}_i; \mathbf{W}), d_i)$$

134

6

## Training the network: Empirical Risk Minimization



- Given a training set of input-output pairs  $(X_1, d_1), (X_2, d_2), \dots, (X_N, d_N)$ 
  - Quantification of error on the  $i^{\text{th}}$  instance:  $\text{div}(f(X_i; W), d_i)$
  - Empirical average divergence (Empirical Risk) on all training data:

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

- Estimate the parameters to minimize the empirical estimate of expected divergence (empirical risk)

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

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

135

7

## Empirical Risk Minimization

$$Y = f(X; W)$$

Note : Its really a measure of error, but using standard terminology, we will call it a "Loss"

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

Note 3: For a given training set the loss is only a function of  $W$

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

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

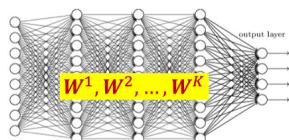
$$\widehat{W} = \underset{W}{\operatorname{argmin}} \text{Loss}(W)$$

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

136

8

## ERM for neural networks



**Actual output of network:**

$$\mathbf{Y}_i = \text{net}(X_i; \{w_{i,j}^k \forall i, j, k\}) \\ = \text{net}(X_i; W^1, W^2, \dots, W^K)$$

**Desired output of network:**  $d_i$

**Error on  $i^{\text{th}}$  training input:**  $\text{Div}(\mathbf{Y}_i, d_i; W^1, W^2, \dots, W^K)$

**Average training error(loss):**

$$\text{Loss}(W^1, W^2, \dots, W^K) = \frac{1}{N} \sum_{i=1}^N \text{Div}(\mathbf{Y}_i, d_i; W^1, W^2, \dots, W^K)$$

- What is the exact form of  $\text{Div}()$ ? More on this later
- Optimize network parameters to minimize the total error over all training inputs

137

9

## Problem Statement

- Given a training set of input-output pairs  $(X_1, d_1), (X_2, d_2), \dots, (X_N, d_N)$

- Minimize the following function

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

w.r.t  $W$

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

138

0

## Story so far

- We learn networks by “fitting” them to training instances drawn from a target function
- Learning networks of threshold-activation perceptrons requires solving a hard combinatorial-optimization problem
  - Because we cannot compute the influence of small changes to the parameters on the overall error
- Instead we use continuous activation functions with non-zero derivatives to enable us to estimate network parameters
  - This makes the output of the network differentiable w.r.t every parameter in the network
  - The *logistic* activation perceptron actually computes the *a posteriori* probability of the output given the input
- We define differentiable *divergence* between the output of the network and the desired output for the training instances
  - And a total error, which is the average divergence over all training instances
- We optimize network parameters to minimize this error
  - Empirical risk minimization
- This is an instance of function minimization

139

1