



## **CS4248: Natural Language Processing**

### Lecture 5 — Introduction into Connectionist Machine Learning

# Announcements

- Project

- Project teams announced (check your team and update us per our announcement if you see anything amiss)
- Project's Intermediate Update Rubric / Template is available  
(Find in "Canvas > Files > Project" or live version (best bet) at <https://bit.ly/cs4248-2320-iu-template>)

- Assignment

- A2: Text Classification competition, restricted to ML algorithms taught (Naive Bayes and Logistic Regression)
- Emphasis on Natural Language Feature Engineering

- Questions, help, and communications

- The Teaching Team is here to help but we cannot do the assessment for you
- Please acknowledge that TA cannot answer every question about assignments and the project  
(of you really think there is a problem with the communications, you can always send us (Min/Chris) an email)
- If an email or Canvas post does not get replied to in 2-3 days, you are welcome to follow up

# Outline

- **Generative vs. Discriminative Classifiers**
- **Logistic Regression**
  - Setup as Probabilistic Classifier
  - Cross-Entropy Loss Function
  - Gradient Descent
  - Overfitting & Regularization
  - Multiclass Logistic Regression
- **Towards Neural Networks**
  - Motivation: XOR Problem
  - Basic Neural Network Architecture

# Text Classification (well, for classification, in general)

*•  $Y$  is fixed*

*1 article bunch of news articles*

- Formal setup

- $X$  — set of all documents;  $x \in X$  — a single document
- $Y$  — set of all classes (or class labels);  $y \in Y$  — a single class (or class label)
- Mapping  $h$  from input space  $X$  to output space  $Y \rightarrow h : X \rightarrow Y$

*hypothesis?*

→ Find best  $\hat{h}$  to approximate the true mapping  $h$

We find  $\hat{h}$  by learning  $\hat{h}$  from the data

→ **Supervised (Machine) Learning**

- Probabilistic Classifiers (e.g., Naive Bayes)

(Instead of)  $\hat{h} : X \rightarrow Y$ , learn  $\hat{P}(Y|X)$  (or  $\hat{P}(y|x)$  for an  $\langle x, y \rangle$  pair)

# Text Classification — Probabilistic Classifiers

- Common goal: Learn  $P(y|x)$  ←

- Learn  $P(y|x)$  from the data

- Two basic approaches

## (1) Generative Classifiers

*Naive Bayes*

- Learn joint probability  $P(x, y)$
- Apply Bayes Rule to get  $P(y|x)$

$$\rightarrow \hat{y} = \operatorname{argmax}_{y \in Y} \underbrace{P(x, y)}_{= P(x, y) \propto P(y|x)} P(y)$$

## (2) Discriminative Classifiers

- Learn  $P(y|x)$  directly

*e.g. Logistic Regression*

$$\rightarrow \hat{y} = \operatorname{argmax}_{y \in Y} P(y|x)$$

# Generative vs. Discriminative Classifiers — Intuition

- Task: Train a classifier to distinguish zebra from elephants images



# Generative vs. Discriminative Classifiers — Intuition

- Generative classifier

- Builds 2 models what zebra and elephant images “look like”

Feature $x_i$	$P(x_i \text{ zebra})$	$P(x_i \text{ elephant})$
"is grey"	0.32	0.95
"is striped"	0.99	0.08
"long nose"	0.40	0.98
"four legs"	0.88	0.99
...	...	...

NB

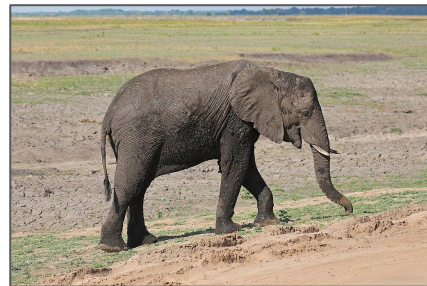
- Models allow to assign a "zebra probability" and an "elephant probability" to any image (using Bayes Rule)
- Given a new image:  
Run both models and see which fits better



# Generative vs. Discriminative Classifiers — Intuition

- Discriminative classifier
  - Tries to distinguish zebra and elephant images
  - Does not model how zebra and elephant images "look like"

Question: How could we quickly distinguish zebras from elephants?

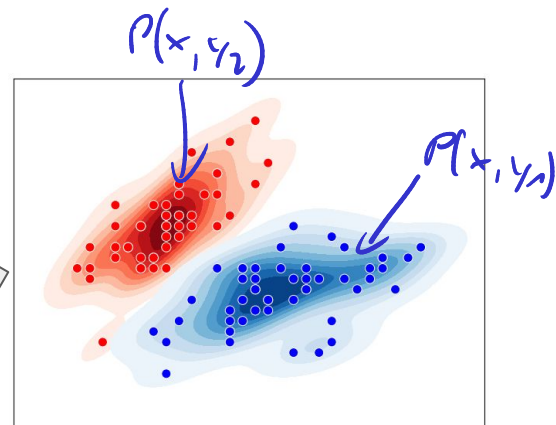
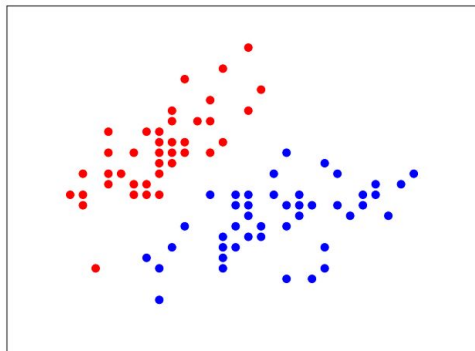




# Generative vs. Discriminative Classifiers

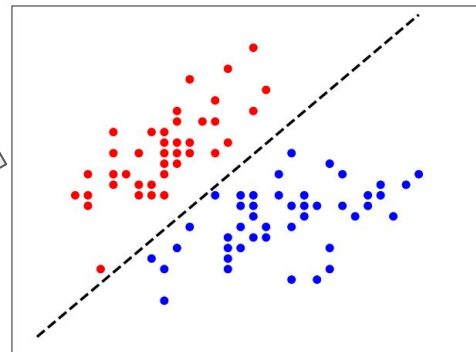
## Generative classifier

- Learn data distribution of each class
- Classifies new data item by comparing the item with each class distribution



## Discriminative classifier

- Learn the decision boundaries between classes
- Classifies new data item based on in which "region" the new item falls



# Outline

- Generative vs. Discriminative Classifiers
- **Logistic Regression**
  - Setup as Probabilistic Classifier
  - Cross-Entropy Loss Function
  - Gradient Descent
  - Overfitting & Regularization
  - Multiclass Logistic Regression
- **Towards Neural Networks**
  - Motivation: XOR Problem
  - Basic Neural Network Architecture

# Linear Models

formula for line:

$$y = m \cdot x + b$$

slope

intercept with y-axis

$$y = \theta_1 \cdot x + b$$

- Underlying assumption:

- There exists linear relationship between  $x^{(j)}$  and dependent variable  $y^{(j)}$

$$\hat{y}^{(j)} = h_{\theta}(x^{(j)}) = f(b + \theta_1 x_1^{(j)} + \theta_2 x_2^{(j)} + \dots + \theta_n x_n^{(j)})$$

offset bias

weighted slope

TF-IDF weight

$$= f\left(\left[\sum_{i=1}^n \theta_i x_i^{(j)}\right] + b\right)$$

Predicted value which is hopefully close to  $y^{(j)}$

$$\theta = \{b, \theta_1, \theta_2, \dots, \theta_n\}, \quad b \in \mathbb{R}, \quad \theta_i \in \mathbb{R}$$

These are the parameters we need to learn  
→ Learning = finding the "right" parameter values

# Linear Models — More User-Friendly Notation

- Vector representation

- **Bias Trick:** Introduce constant feature  $x_0^{(j)}$

$$h_{\theta} \left( x^{(j)} \right) = f \left( \underbrace{\theta_0 x_0^{(j)}}_{=1} + \theta_1 x_1^{(j)} + \theta_2 x_2^{(j)} + \cdots + \theta_n x_n^{(j)} \right)$$

- Represent  $x^{(j)}$  with new constant feature

$$x^{(j)} = \left( \underline{1}, x_1^{(j)}, x_2^{(j)}, \dots, x_n^{(j)} \right)$$

- Rewrite linear relationship using vectors representing  $x^{(j)}$  and  $\theta$

$$h \left( x^{(j)} \right) = f \left( \underline{\theta^T x^{(j)}} \right) \quad \theta = \{\theta_0, \theta_1, \theta_2, \dots, \theta_n\}, \theta_i \in \mathbb{R}$$

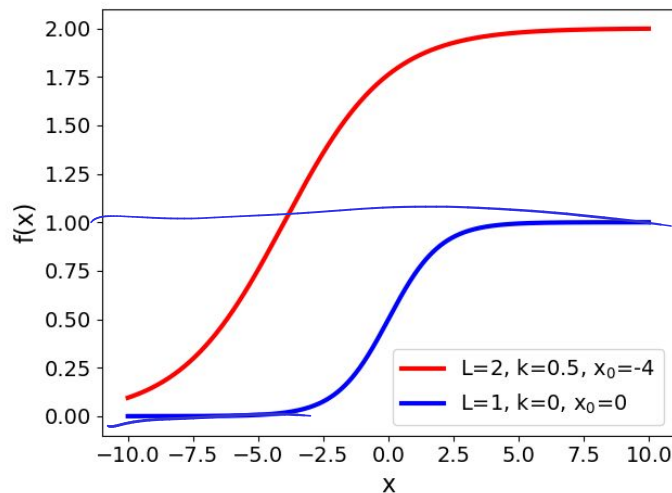
**Note:** Throughout the rest of the slide, we drop to superscript in  $x^{(j)}$  and  $y^{(j)}$  if there is no ambiguity.

# Logistic Regression

- Logistic Regression → Real-valued predictions interpreted as probability

- Function  $f$  is the standard **Logistic Function** (Sigmoid function)

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \xrightarrow[L = 1, k = 1, x_0 = 0]{\text{slope}} f(x) = \frac{1}{1 + e^{-x}}$$



$L=2$

# Logistic Regression — Probabilistic Interpretation

binary  
classification  
class 0, class 1

- $\hat{y}$  <sup>estimate / prediction</sup> interpreted as a probability

$$\hat{y} = h_{\theta}(x) = f(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad \text{with } \hat{y} \in [0, 1]$$

→  $\hat{y} = h_{\theta}(x)$  is the estimated probability that  $y = 1$  given  $x$  and  $\theta$

$$\hat{y} = P(y = 1 | \underline{x}, \theta)$$

→ Given only discrete 2 outcomes:  $P(y = 1 | x, \theta) + P(y = 0 | x, \theta) = 1$

$$\hat{y} = 1 - P(y = 0 | x, \theta)$$

# Outline

- Generative vs. Discriminative Classifiers
- **Logistic Regression**
  - **Setup as Probabilistic Classifier**
  - Cross-Entropy Loss Function
  - Gradient Descent
  - Overfitting & Regularization
  - Multiclass Logistic Regression
- **Towards Neural Networks**
  - Motivation: XOR Problem
  - Basic Neural Network Architecture

# Logistic Regression — Runthrough (Part 1)

- Sentiment Analysis for movie reviews

*"It's hokey. There are no surprises, the writing is poor. So why was it so enjoyable? For one thing, the cast is great. Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you."*

Feature	Description	Value
$x_1$	Number of positive words	
$x_2$	Number of negative words	
$x_3$	1 if "no" in text; 0 otherwise	
$x_4$	Number of 1st & 2nd person pronouns	
$x_5$	1 if "!" in text; 0 otherwise	
$x_6$	$\ln$ of word/token count	

**Side notes:**

- Naive Bayes and Logistic Regression require feature engineering as they do not combine primitive features into composite ones.
- The 6 features on the left are chosen for simplicity; in practice these can be the tf-idf weights



# Logistic Regression — Runthrough (Part 1)

- Step 1: Extract feature values

*"It's **hokey**. There are **no** surprises, the writing is **poor**. So why was it so **enjoyable**? For one thing, the cast is **great**. Another **nice** touch is the music. **I** was overcome with the urge to get off the couch and start dancing. It sucked **me** in, and it'll do the same to **you**."*

Feature	Description	Value
$x_1$	Number of positive words	3
$x_2$	Number of negative words	2
$x_3$	1 if "no" in text; 0 otherwise	1
$x_4$	Number of 1st & 2nd person pronouns	3
$x_5$	1 if "!" in text; 0 otherwise	0
$x_6$	$\ln$ of word/token count	$\ln(66) = 4.19$

# In-Lecture Activity (5 mins)

- Question: What might be other useful features for a sentiment classifier?
  - Bonus: Briefly discuss how easy/difficult it would be to extract your features
  - Post your features to the Canvas Discussion  
(individually or as a group; include all group members' names in the post)

# Logistic Regression — Runthrough (Part 1)

- Step 2: Factor in weights  $\theta$ 
  - Let's assume some oracle gave us those weights
  - It's time to include the bias using the "bias trick"

Feature	Description	Value	Weight $\theta_i$
$x_0$	Bias $b$	1	0.1
$x_1$	Number of positive words	3	2.5
$x_2$	Number of negative words	2	-5.0
$x_3$	1 if "no" in text; 0 otherwise	1	-1.2
$x_4$	Number of 1st & 2nd person pronouns	3	0.5
$x_5$	1 if "!" in text; 0 otherwise	0	2.0
$x_6$	$\ln$ of word/token count	4.19	0.7

# Logistic Regression — Runthrough (Part 1)

- Step 4: Compute linear signal (sum of weighted features)

Feature	Description	Value	Weight $\theta_i$	$\theta_i x_i$
$x_0$	Bias $b$	1	0.1	0.1
$x_1$	Number of positive words	3	2.5	7.5
$x_2$	Number of negative words	2	-5.0	-10.0
$x_3$	1 if "no" in text; 0 otherwise	1	-1.2	-1.2
$x_4$	Number of 1st & 2nd person pronouns	3	0.5	1.5
$x_5$	1 if "!" in text; 0 otherwise	0	2.0	0
$x_6$	$\ln$ of word/token count	4.19	0.7	2.933

$0.1 = 0.1$   
 $+ 2.5 \cdot 3$   
.

**Vector notation:**

$$\left. \begin{aligned} x &= (1, 3, 2, 1, 3, 0, 4.19)^T \\ \theta &= (0.1, 2.5, -5.0, -1.2, 0.5, 2.0, 0.7)^T \end{aligned} \right\} \rightarrow \theta^T x = 0.833$$

$$\sum = 0.833$$

↑  
not a prob

# Logistic Regression — Runthrough (Part 1)

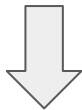
$$\theta^T x = 0.833$$

- Step 4: Compute probabilities

$$P(+|x) = P(y = 1|x, \theta) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} = \frac{1}{1 + e^{-0.833}} = \underline{0.7}$$

$$P(-|x) = P(y = 0|x, \theta) = 1 - P(y = 1|x, \theta) = 0.3$$

$$1 - 0.7 = 0.3$$



$$P(+|x) > 0.5 \rightarrow \hat{y} = + \text{ (positive)}$$

Classify movie review as "positive"

# Logistic Regression

- So, where did the values for  $\theta$  come from?

(in the example, they were simply given to us)

- Of course, different  $\theta$  values would have resulted in different probabilities

- Break down into 2 questions

(1) *How can we quantify how good a set of  $\theta$  values is?*

→ **Loss function** (also: cost function, error function)

(2) *How can we systematically find the best  $\theta$  values?*

→ **Gradient Descent** (numerical method to minimize loss function)

# Outline

- Generative vs. Discriminative Classifiers
- **Logistic Regression**
  - Setup as Probabilistic Classifier
  - **Cross-Entropy Loss Function**
  - Gradient Descent
  - Overfitting & Regularization
  - Multiclass Logistic Regression
- **Towards Neural Networks**
  - Motivation: XOR Problem
  - Basic Neural Network Architecture

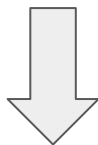
# Logistic Regression — Loss Function

- Intuition: A set of values for  $\theta$  is good if

- the correct label  $y$  (0 or 1; coming from the dataset)
- the model's estimated label  $\hat{y} = \sigma(\theta^T x)$

are similar for all  $\langle x, y \rangle$  pairs

→ Find  $\theta$  that **minimizes the difference** between  $\hat{y}$  and  $y$



$L$  is large  $\rightarrow \theta$  is bad  
 $L$  is small  $\rightarrow \theta$  is good

$L(\hat{y}, y)$  = how much  $\hat{y}$  differs from  $y$



# Logistic Regression — Loss Function

$$\hat{y} = \frac{1}{1 + e^{-\theta^T x}}$$

- Goal: Maximize probability of the correct label  $P(y|x)$

$$\hat{y} = P(\underline{y=1}|x, \theta) = 1 - P(y = 0|x, \theta)$$

- Intermediate step: Combine both case into one formula
  - $P(y|x)$  is a Bernoulli distribution (2 discrete outcomes)

$$P(y|x) = \begin{cases} \hat{y} & , y = 1 \\ 1 - \hat{y} & , y = 0 \end{cases}$$

→ Combine into:

$$P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

$$P(y|x) = \hat{y}^0 (1 - \hat{y})^{1-0} = 1 - \hat{y}$$

# Logistic Regression — Loss Function

$$\hat{y} = \frac{1}{1 + e^{-\theta^T x}}$$

- Goal: Maximize probability of the correct label  $P(y|x)$

- Find  $\theta$  that **maximizes**

$$P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

$$\log P(y|x) = \log [\hat{y}^y (1 - \hat{y})^{1-y}]$$

$$= y \log \hat{y} + (1 - y) \log (1 - \hat{y})$$

$$\begin{aligned} \log ab &= \log a + \log b \\ \log a^b &= b \log a \end{aligned}$$

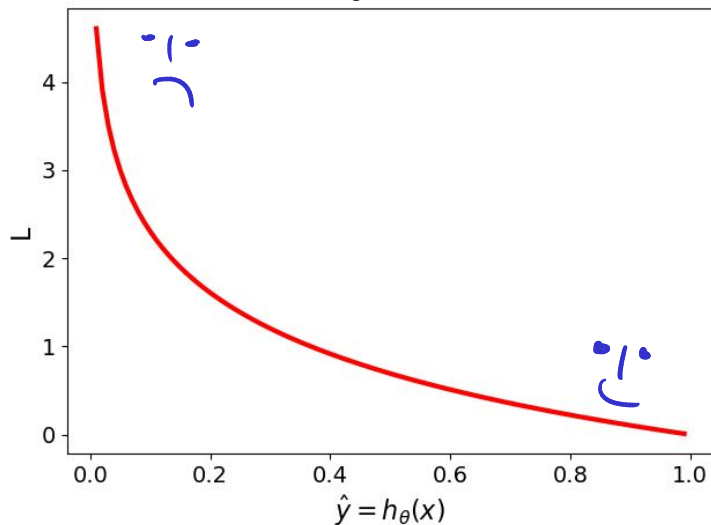
- Find  $\theta$  that **minimizes**

$$L_{CE}(\hat{y}, y) = -P(y|x) \cancel{y} - \underbrace{[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]}_{\text{Cross-Entropy Loss}}$$

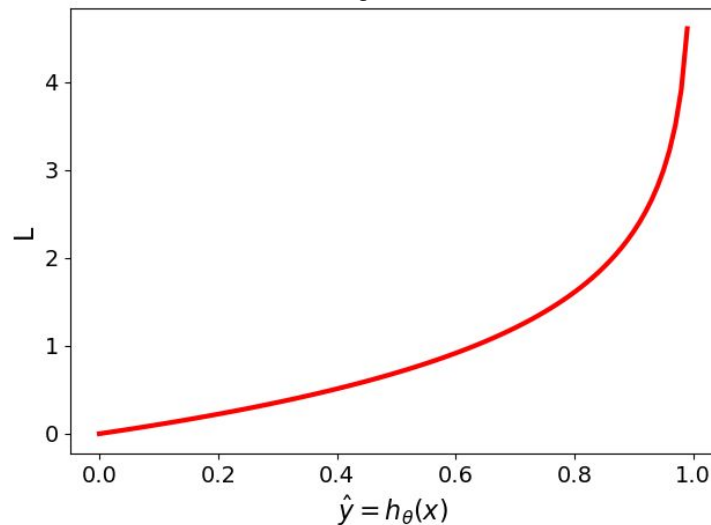
# Cross-Entropy Loss — Visualization

$$L_{CE}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

if  $y = 1$



if  $y = 0$



# Cross-Entropy Loss — Runthrough Example (Part 2)

Recall:

$$P(+|x) = \sigma(\theta^T x) = \underline{0.7}$$

$$P(-|x) = 1 - \sigma(\theta^T x) = \underline{0.3}$$

$\} \Sigma = 1$

Feature	Description	Value	Weight $\theta_i$	$\theta_i x_i$
$x_0$	Bias $b$	1	0.1	0.1
$x_1$	Number of positive words	3	2.5	7.5
$x_2$	Number of negative words	2	-5.0	-10.0
$x_3$	1 if "no" in text; 0 otherwise	1	-1.2	-1.2
$x_4$	Number of 1st & 2nd person pronouns	3	0.5	1.5
$x_5$	1 if "!" in text; 0 otherwise	0	2.0	0
$x_6$	$\ln$ of word/token count	4.19	0.7	2.933

$$L_{CE}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

Assume the model was right ( $y = \underline{1}$ )



$$L_{CE}(\hat{y}, y) = ???$$

Assume the model was wrong ( $y = \underline{0}$ )



$$L_{CE}(\hat{y}, y) = ???$$

# Cross-Entropy Loss — Runthrough Example (Part 2)

$$P(+|x) = \sigma(\theta^T x) = 0.7$$

$$P(-|x) = 1 - \sigma(\theta^T x) = 0.3$$

$$L_{CE}(\hat{y}, y) = - [y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

Assume the model was right ( $y = 1$ )



$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[\log \hat{y}] \\ &= -[\log 0.7] \\ &= \underline{0.36} \end{aligned}$$

Assume the model was wrong ( $y = 0$ )

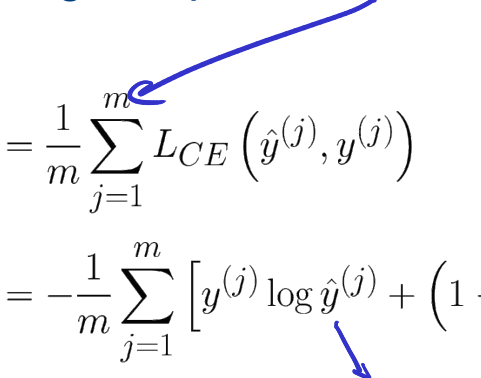


$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[\log (1 - \hat{y})] \\ &= -[\log 0.3] \\ &= \underline{1.2} \end{aligned}$$



# Cross-Entropy Loss — Total Loss

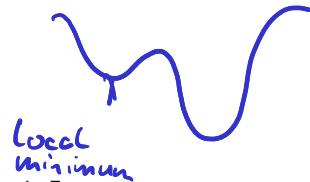
- Loss for all training samples (given  $m$  data samples)


$$\begin{aligned} L_{CE} &= \frac{1}{m} \sum_{j=1}^m L_{CE}(\hat{y}^{(j)}, y^{(j)}) \\ &= -\frac{1}{m} \sum_{j=1}^m \left[ y^{(j)} \log \hat{y}^{(j)} + (1 - y^{(j)}) \log (1 - \hat{y}^{(j)}) \right] \\ &= -\frac{1}{m} \sum_{j=1}^m \left[ y^{(j)} \log \sigma(\theta^T x^{(j)}) + (1 - y^{(j)}) \log (1 - \sigma(\theta^T x^{(j)})) \right] \\ &= -\frac{1}{m} \sum_{j=1}^m \left[ y^{(j)} \log \frac{1}{1 + e^{\theta^T x^{(j)}}} + (1 - y^{(j)}) \log \left( 1 - \frac{1}{1 + e^{\theta^T x^{(j)}}} \right) \right] \end{aligned}$$

# Outline

- Generative vs. Discriminative Classifiers
- **Logistic Regression**
  - Setup as Probabilistic Classifier
  - Cross-Entropy Loss Function
  - **Gradient Descent**
  - Overfitting & Regularization
  - Multiclass Logistic Regression
- Towards Neural Networks
  - Motivation: XOR Problem
  - Basic Neural Network Architecture

# Learning — Minimizing the Loss Function



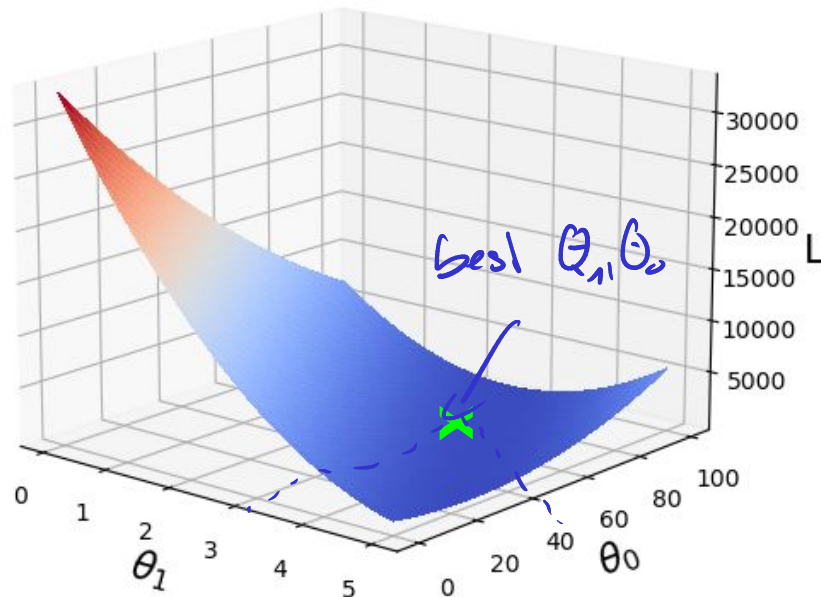
$$L_{CE} = -\frac{1}{m} \sum_{j=1}^m \left[ y^{(j)} \log \frac{1}{1 + e^{\theta^T x^{(j)}}} + \left(1 - y^{(j)}\right) \log \left(1 - \frac{1}{1 + e^{\theta^T x^{(j)}}}\right) \right]$$

## Visual illustration of loss function

- Just 1 feature  $\theta_1$  and bias  $\theta_0$
- Good news:  $L_{CE}$  for Logistic Regression is a convex function  $\rightarrow$  1 global minimum

$\rightarrow$  How to find the minimum of  $L_{CE}$ ?

...this should cause a flashback to your calculus classes :)





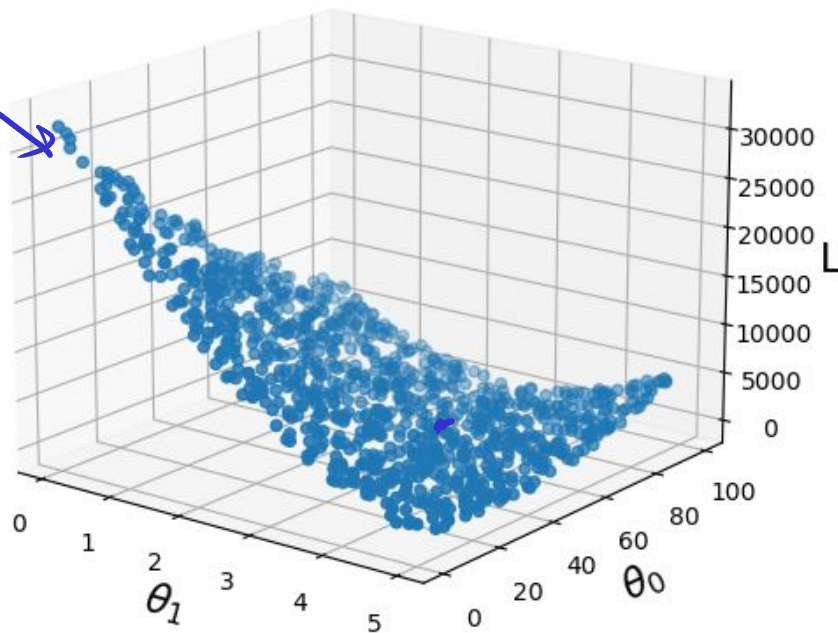
# Method 1: Random Search (the "stupid" way)

- Repeat "enough" times
  - Select random values for  $\theta = \{\theta_0, \theta_1, \theta_2, \dots, \theta_n\}$
  - Calculate loss  $L$  for current  $\theta$
- Return  $\theta$  with smallest loss

- Limitation:
  - Not practical beyond toy examples

→ Don't do that! :)

Plot of 1,000 losses



# Method 2: Using Calculus (the proper way)

- Minimum of loss function  $L \rightarrow$  Calculus to the rescue!

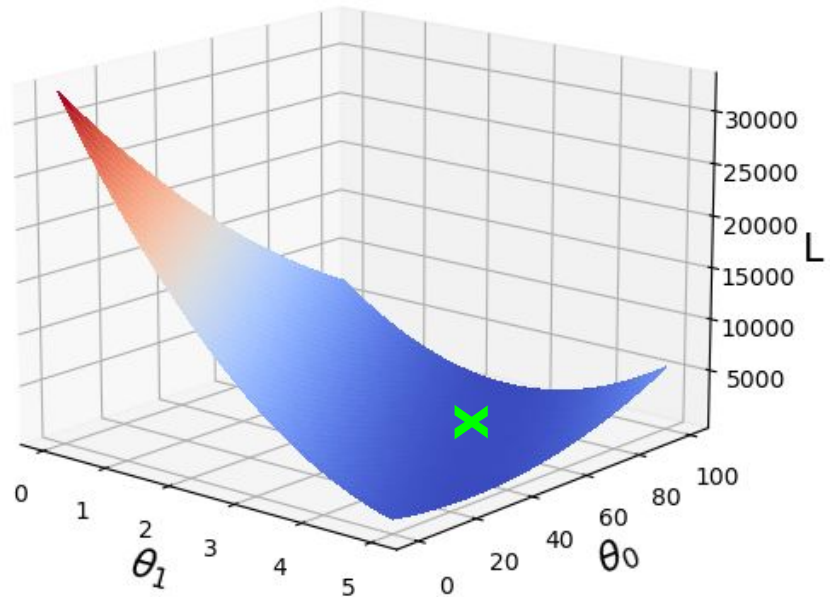
- Partial derivatives w.r.t. to all  $\theta_i$  are 0

$$\frac{\partial L}{\partial \theta_0} = 0, \frac{\partial L}{\partial \theta_1} = 0, \frac{\partial L}{\partial \theta_2} = 0, \dots, \frac{\partial L}{\partial \theta_n} = 0$$

- $n+1$  equations with  $n+1$  unknowns  
( $\rightarrow$  1 unique solution  $\rightarrow$  1 global minimum)

$\rightarrow$  What we need:

$$\frac{\partial L}{\partial \theta}$$



# Loss Function — Derivatives

$$L_{CE} = -\frac{1}{m} \sum_{j=1}^m \left[ y^{(j)} \log \sigma \left( \theta^T x^{(j)} \right) + \left( 1 - y^{(j)} \right) \log \left( 1 - \sigma \left( \theta^T x^{(j)} \right) \right) \right]$$



...lots of tedious math here...



$$\frac{\partial L_{CE}}{\partial \theta_i} = \frac{1}{m} \sum_{j=1}^m \left[ \sigma \left( \theta^T x^{(j)} \right) - y^{(j)} \right] x_i^{(j)}$$

↗

$$\frac{\partial L_{CE}}{\partial \theta} = \frac{1}{m} X^T [\sigma(X\theta) - y]$$

↗ matrix / vector notation

Basic approach to find the minimum

- (1) Set derivative to 0  $\rightarrow \frac{1}{m} X^T [\sigma(X\theta) - y] \stackrel{!}{=} 0$
- (2) Solve for  $\theta$

↘  
 $\theta = \text{something}$

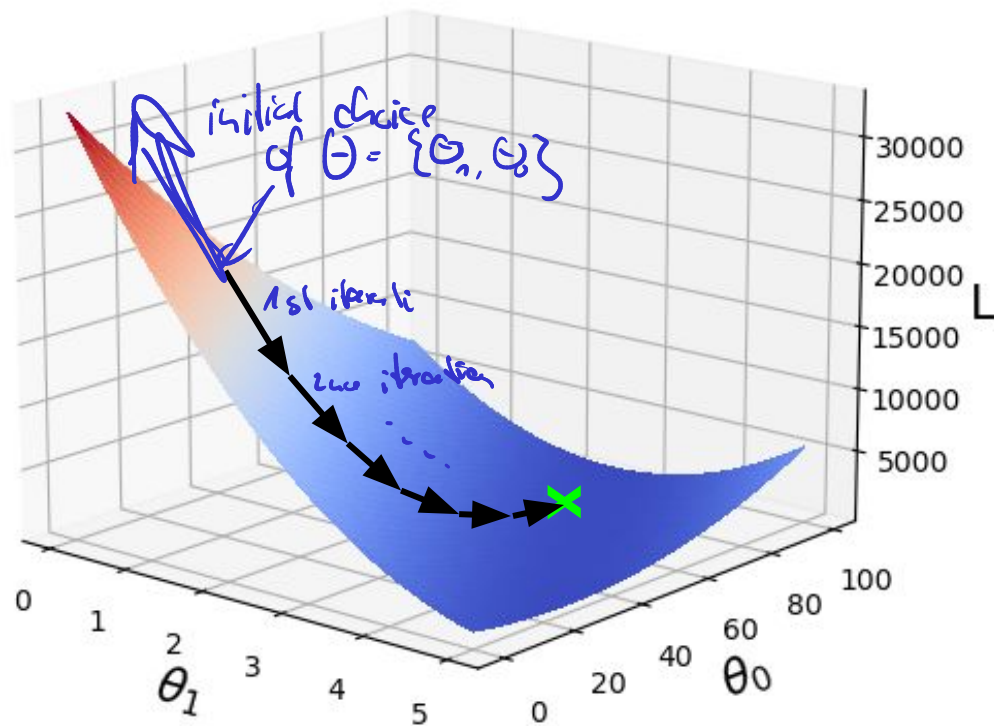
So are we done here?

# Gradient Descent

- Problem:  $\frac{1}{m} X^T [\sigma(X\theta) - y] \stackrel{!}{=} 0$  has no closed-form solution for  $\theta$   $\nRightarrow \theta = \text{some } \theta_0$

## → Gradient Descent

- Start with a random setting of  $\theta$
- Adjust  $\theta$  iteratively to minimize  $L$



# Gradient — Quick Refresher

- Gradient

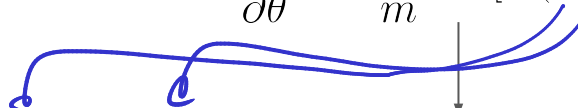
- Vector of partial derivatives of a multivariable function (e.g.,  $\theta_0, \theta_1, \dots, \theta_n$ )
- Partial derivative: slope w.r.t. to a single variable given a current set of values for all  $\theta_0, \theta_1, \dots, \theta_n$
- Points in the direction of the steepest ascent

$$\nabla_{\theta} L = \frac{\partial L}{\partial \theta} = \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \vdots \\ \frac{\partial L}{\partial \theta_n} \end{bmatrix}$$

*partial derivative* →

# Gradients — Runthrough Example (Part 3)

- Calculate Gradients (assuming  $y = 1$ )

$$\frac{\partial L_{CE}}{\partial \theta} = \frac{1}{m} X^T [\sigma(X\theta) - y]$$


Feature	Description	Value	Weight $\theta_i$	$\theta_i x_i$	Gradients
$x_0$	Bias $b$	1	0.1	0.1	-0.30
$x_1$	Number of positive words	3	2.5	7.5	-0.91
$x_2$	Number of negative words	2	-5.0	-10.0	-0.61
$x_3$	1 if "no" in text; 0 otherwise	1	-1.2	-1.2	-0.30
$x_4$	Number of 1st & 2nd person pronouns	3	0.5	1.5	-0.91
$x_5$	1 if "!" in text; 0 otherwise	0	2.0	0	0.0
$x_6$	$\ln$ of word/token count	4.19	0.7	2.933	-1.27

$$\rightarrow \nabla_{\theta} L_{CE} = \begin{bmatrix} -0.30 \\ -0.91 \\ -0.61 \\ -0.30 \\ -0.91 \\ 0.0 \\ -1.27 \end{bmatrix}$$

# Gradients — Runthrough Example (Part 3)

$$\begin{aligned} \theta_0 &\Rightarrow 0.01 \cdot (-0.30) \\ \eta &= 0.003 \\ 0.1 &\rightarrow 0.1003 \end{aligned}$$

- Interpretation of gradients

- Negative values: a small increase in, e.g.,  $\theta_0$  or  $\theta_1$  will decrease the loss
- A small change in  $\theta_1$  affects the loss more than the same change in  $\theta_0$   
(since the absolute value of  $\theta_1$  is larger than the one of  $\theta_0$ )
- Absolute values of gradient not a direct indicator of how to update  $\theta$

$$\theta_1: 0.1 \rightarrow \underline{0.12}$$

$\hookrightarrow$  decreases  $L$

→ So how do we adjust  $\theta$  to decrease the loss?

$$\nabla_{\theta} L_{CE} = \begin{bmatrix} -0.30 \\ -0.91 \\ -0.61 \\ -0.30 \\ -0.91 \\ 0.0 \\ -1.27 \end{bmatrix}$$

# Gradient Descent Algorithm

- Important concept: learning rate

- Scaling factor for gradient (typical range: 0.01 - 0.0001)

**Input** : data  $(X, y)$ , loss function  $L$ , learning rate  $\eta$

**Initialization** : Set  $\theta$  to random values

**while true** :

    Calculate gradient  $\nabla_{\theta} L$

$\theta \leftarrow \theta - (\eta \cdot \nabla_{\theta} L)$

↑  
steepest descent

↖ cross-entropy loss

In practice: stop loop  
when  $\theta$  converges



# Gradient Descent — Runthrough Example (Part 4)

- Update weights  $\theta$

- Learning rate:  $\eta = 0.1$

$$\theta \leftarrow \theta - (\eta \cdot \nabla_{\theta} L)$$

Feature	Description	Value	Weight $\theta_i$	$\theta_i x_i$	<u>Gradients</u>	New Weight $\theta_i$
$x_0$	Bias $b$	1	0.1	0.1	-0.30	<b>0.13</b>
$x_1$	Number of positive words	3	2.5	7.5	-0.91	<b>2.59</b>
$x_2$	Number of negative words	2	-5.0	-10.0	-0.61	<b>-4.94</b>
$x_3$	1 if "no" in text; 0 otherwise	1	-1.2	-1.2	-0.30	<b>-1.17</b>
$x_4$	Number of 1st & 2nd person pronouns	3	0.5	1.5	-0.91	<b>0.59</b>
$x_5$	1 if "!" in text; 0 otherwise	0	2.0	0	0.0	<b>2.0</b>
$x_6$	<i>ln</i> of word/token count	4.19	0.7	2.933	-1.27	<b>0.83</b>

→ 1st iteration of Gradient Descent done!

→  $L_{CE} = 0.12$   
(down from 0.36)

# Gradient Descent — Runthrough Example (Part 4)

- Update weights  $\theta$

- Learning rate:  $\eta = 0.1$

$$\theta \leftarrow \theta - (\eta \cdot \nabla_{\theta} L)$$

Feature	Description	Value	Weight $\theta_i$	$\theta_i x_i$	Gradients	New Weight $\theta_i$
$x_0$	Bias $b$	1	0.13	0.13	-0.11	<b>0.14</b>
$x_1$	Number of positive words	3	2.59	7.77	-0.33	<b>2.62</b>
$x_2$	Number of negative words	2	-4.94	-9.88	-0.22	<b>-4.92</b>
$x_3$	1 if "no" in text; 0 otherwise	1	-1.17	-1.17	-0.11	<b>-1.16</b>
$x_4$	Number of 1st & 2nd person pronouns	3	0.59	1.77	-0.33	<b>0.62</b>
$x_5$	1 if "!" in text; 0 otherwise	0	2.0	0	0.0	<b>2.0</b>
$x_6$	$\ln$ of word/token count	4.19	0.83	3.46	-0.46	<b>0.87</b>

→ 2nd iteration of Gradient Descent done!

↪  $L_{CE} = 0.075$   
(down from 0.12)

## Quick Quiz

$$y = 0 \quad / \quad \nearrow$$
$$\hat{y} = 0.95 \dots$$

What happens if  
Logistic Regression gets a  
training sample **correct**?

**A**

No loss will be calculated

**B**

The loss will be 0

**C**



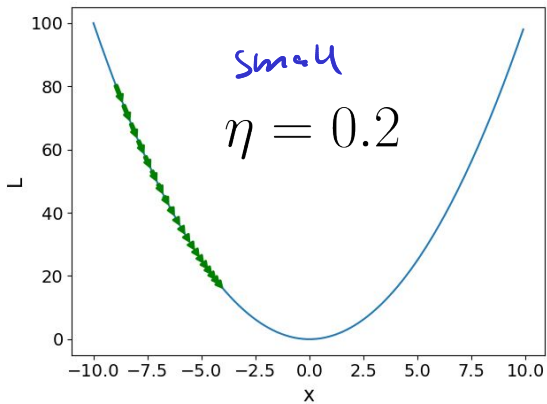
The loss will be small

**D**

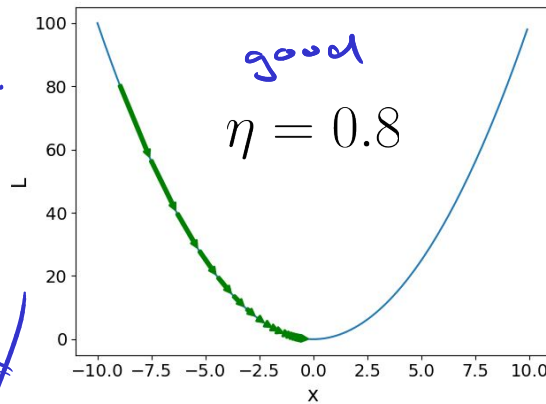
The loss will be large

# Effects of Learning Rate for

$$L = x^2, \frac{\partial L}{\partial x} = 2x, \text{ 20 steps}$$



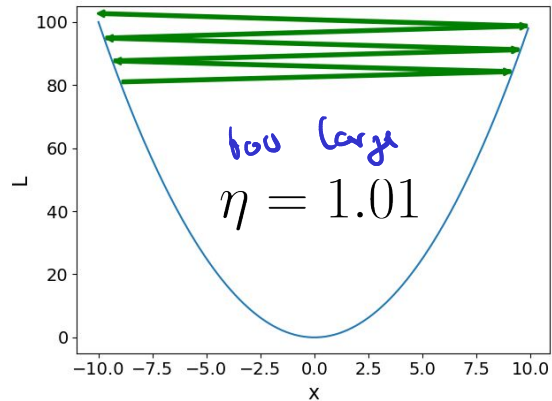
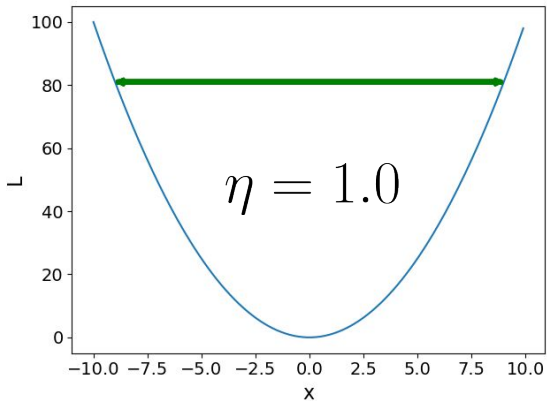
loss  
function



$\eta = 0.58$



A hand-drawn blue diagram of a parabola with several horizontal lines inside, representing a range of learning rates.

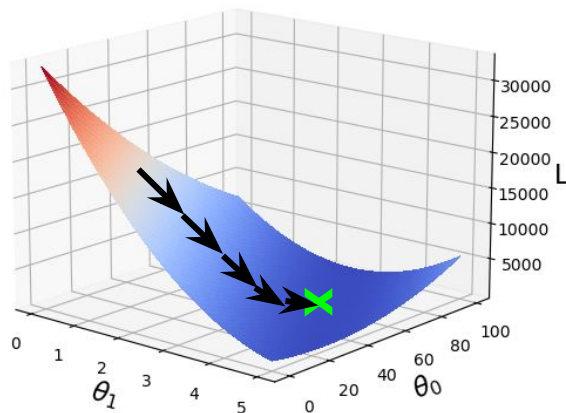


# Gradient Descent — Variations

- (Basic) Gradient Descent
  - Calculate gradient und update  $\theta$  for whole dataset
- Stochastic Gradient Descent (SGD)
  - Calculate gradient und update  $\theta$  for each data sample
- Mini-batch Gradient Descent *← practice*
  - Calculate gradient und update  $\theta$  for batches of sample
  - e.g., batch = 64 data samples
  - In practice often referred to as SGD

# Gradient Descent — Variations

## Gradient Descent

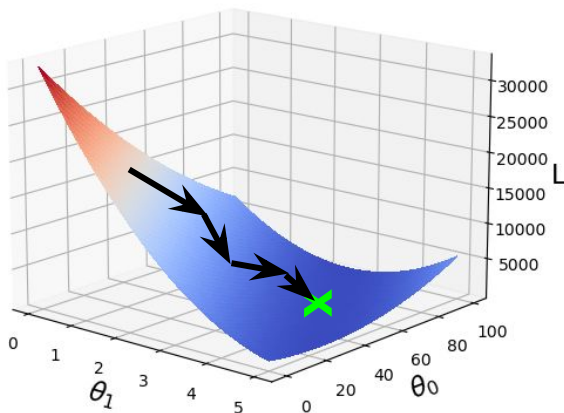


Gradient averaged over all data items

- Smooth descent
- Small(er) gradients
- Small(er) update steps

*higher  $\eta$*

## Mini-Batch Gradient Descent

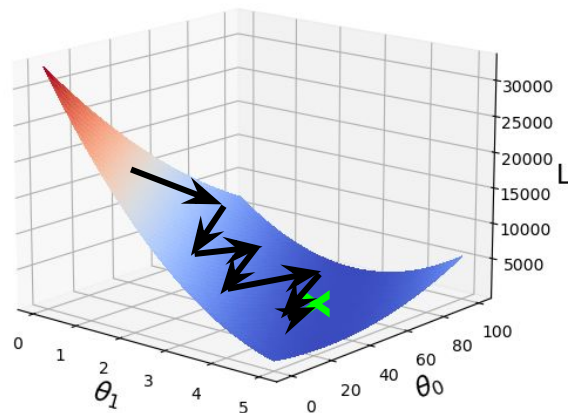


Gradient averaged over some data items

- Well, "somewhere in-between" :)



## Stochastic Gradient Descent



Gradient for each data item considered

- Choppy descent
- Large(r) gradients
- Large(r) steps

# Gradient Descent — When to Stop?

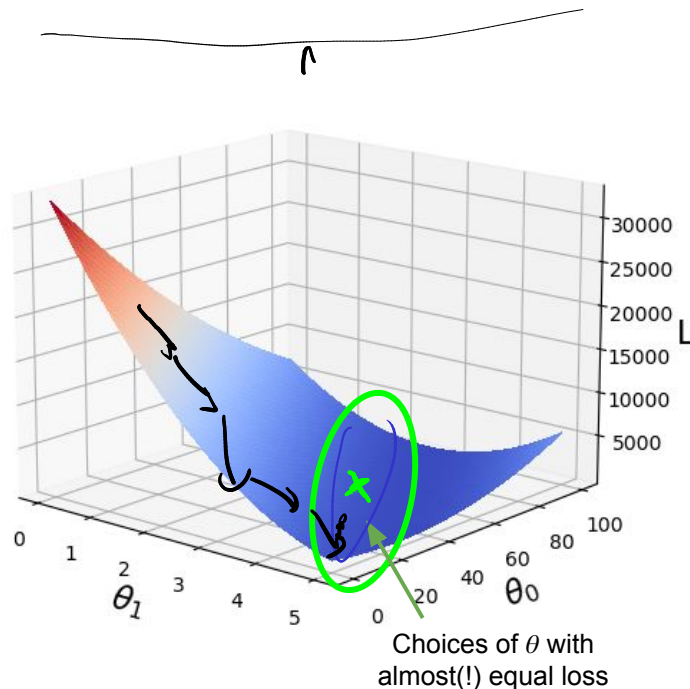
- Intuition:  $\nabla_{\theta} L_{CE} < threshold$

Problem: regions of "near-plateaus":

- Gradient  $\nabla_{\theta} L$  very small
- Step  $\eta \nabla_{\theta} L$  extremely small
- Very slow convergence

- Alternative stop conditions:

- Loss is small (enough)
- Change in loss is small enough
- Max. #iterations reached



**Note:** This problem is much more pronounced for non-convex loss function with multiple local minima

# Outline

- Generative vs. Discriminative Classifiers
- **Logistic Regression**
  - Setup as Probabilistic Classifier
  - Cross-Entropy Loss Function
  - Gradient Descent
  - **Overfitting & Regularization**
  - Multiclass Logistic Regression
- Towards Neural Networks
  - Motivation: XOR Problem
  - Basic Neural Network Architecture

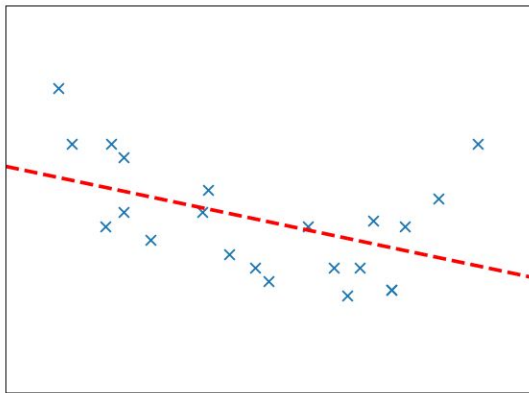


# Overfitting — Basic Intuition

- Overfitting — Visualized using curve fitting

- Task: Find a polynomial for degree  $p$  that best fit the data points

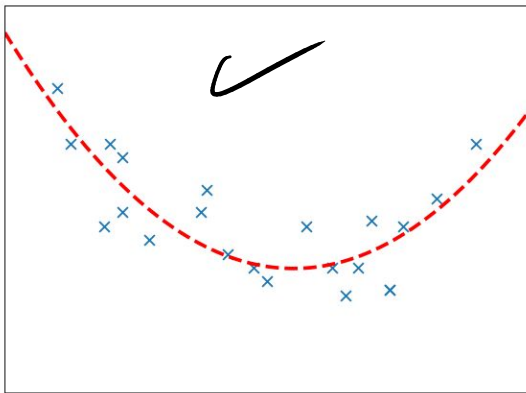
$p = 1$



Underfitting

- Polynomial of degree 1 just a line
- Not capable to fit non-linear data

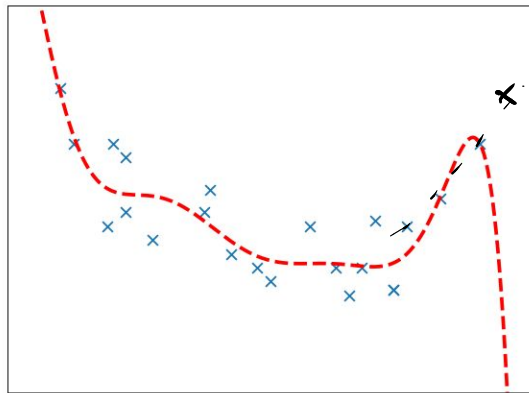
$p = 2$



Good fit

- Model captures the overall trend
- Probably good fit for unseen data

$p = 8$



Overfitting

- Model has too much capacity to exactly fit individual data points
- Probably bad fit for unseen data

# Overfitting — Intuition (Naive Bayes Classifier)

- Scenario — movie reviews
  - (Very) low number of reviews
  - NB classifier based on 4-grams

<i>This movie drew me in, and it'll do the same to you.</i>	positive
<i>I can't tell you how much I hated this movie. It sucked.</i>	negative
...	...

## → Effect of Naive Bayes classifier

- Each 4-gram most likely unique and associated with only 1 class  
(e.g., "tell you how much" only found in a negative review)
- Unseen positive review  $x$  containing "tell you how much" →  $P(\text{positive}|x) = 0$

# Overfitting — Intuition (Logistic Regression Classifier)

- Scenario — movie reviews

- (Very) low number of reviews

- Assume the following artifact

All positive reviews contain many pronouns

Almost no negative reviews contain pronouns

Feature	Description	Value	Weight $\theta_i$	$\theta_i x_i$
$x_0$	Bias $b$	1	0.1	0.1
$x_1$	Number of positive words	3	2.5	7.5
$x_2$	Number of negative words	2	-5.0	-10.0
$x_3$	1 if "no" in text; 0 otherwise	1	-1.2	-1.2
$x_4$	Number of 1st & 2nd person pronouns	3	0.5	1.5
$x_5$	1 if "!" in text; 0 otherwise	0	2.0	0
$x_6$	$\ln$ of word/token count	4.19	0.7	2.933

## → Effect of Logistic Regression classifier

- Classifiers over-emphasizes the importance of pronouns
  - large value for  $\theta_4$  (compared to other  $\theta_i$ )
- Unseen negative review with many pronouns will most likely be misclassified

# Regularization

- Observation

- Model "too powerful"  $\Leftrightarrow$  (very) large  $\theta$  values

→ **Regularization**: Penalize large  $\theta$  values

- Extend loss function by penalty term
- For example, for Cross-Entropy loss

$$L = -\frac{1}{m} \sum_{j=1}^m \left[ y^{(j)} \log \sigma \left( \theta^T x^{(j)} \right) + \left( 1 - y^{(j)} \right) \log \left( 1 - \sigma \left( \theta^T x^{(j)} \right) \right) \right] + \lambda \sum_{i=1}^n \theta_i^2$$

*Cross-Entropy*

$\lambda$ : Regularization Parameter to control the "strength of the regularization"

L2 Regularization  
("Ridge Regression")

$$L = -\frac{1}{m} \sum_{j=1}^m \left[ y^{(j)} \log \sigma \left( \theta^T x^{(j)} \right) + \left( 1 - y^{(j)} \right) \log \left( 1 - \sigma \left( \theta^T x^{(j)} \right) \right) \right] + \lambda \sum_{i=1}^n |\theta_i|$$

L1 Regularization  
("Lasso Regression")

# New Loss → New Gradient

- Since we change  $L$ , the gradient  $\nabla_{\theta} L = \frac{\partial L}{\partial \theta}$  also changes
  - No big deal, regularization is just an added term
  - For example, for L2 Regularization (Ridge Regression)

$$\frac{\partial L_{CE}}{\partial \theta} = \frac{1}{m} X^T [\sigma(X\theta) - y] + \lambda \frac{2}{n} \theta$$

- No changes to Gradient Descent Algorithms

# Quick Quiz

Which of the statements regarding  
Logistic Regression is **True**?

A

It's impossible to overfit given a dataset with only 1 feature

B ✓

Scaling the data will change the values for  $\theta$

C

Gradient Descent can get stuck in local minimum

D

Regularization can improve the training loss/error

# Outline

- Generative vs. Discriminative Classifiers
- **Logistic Regression**
  - Setup as Probabilistic Classifier
  - Cross-Entropy Loss Function
  - Gradient Descent
  - Overfitting & Regularization
  - **Multiclass Logistic Regression**
- Towards Neural Networks
  - Motivation: XOR Problem
  - Basic Neural Network Architecture

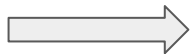
# Binary LR → Multiclass LR

- Multiclass LR: Classification beyond 2 classes

- Let's assume we have  $C$  classes:  $c = 1..C$  *e.g.  $C=10$*
- Separate weights  $\theta_c$  for each classes  $c \rightarrow C$  output probabilities

## Binary Logistic Regression

$$P(y = 1|x) = \sigma(\theta_1^T x)$$



## Multiclass Logistic Regression

$$\underbrace{\begin{bmatrix} P(y = 1|x) \\ P(y = 2|x) \\ \dots \\ P(y = C|x) \end{bmatrix}}_{\text{Probabilities need to sum up to 1}} = f_{\text{mystery}} \left( \begin{bmatrix} \theta_1^T x \\ \theta_2^T x \\ \dots \\ \theta_C^T x \end{bmatrix} \right)$$

Probabilities need  
to sum up to 1

→ How can we ensure that?



# $f_{mystery} \rightarrow \text{Softmax}$

- **Softmax function**

- Converts any vector of scores into a vector of probabilities

$$P(y = c|x) = \frac{\exp(\theta_c^T x)}{\sum_{i=1}^C \exp(\theta_i^T x)}$$

$$\begin{bmatrix} P(y = 1|x) \\ P(y = 2|x) \\ \dots \\ P(y = C|x) \end{bmatrix} = \frac{1}{\sum_{i=1}^C \exp(\theta_i^T x)} \begin{bmatrix} \exp(\theta_1^T x) \\ \exp(\theta_2^T x) \\ \dots \\ \exp(\theta_C^T x) \end{bmatrix}$$

# Example

- Example with 4 classes and 3 input features

$$\begin{array}{c} \text{Weight matrix } \theta \end{array} \quad \begin{array}{c} x \end{array} \quad \begin{array}{c} \theta^T x \end{array} \quad \begin{array}{c} \hat{y} \end{array}$$
$$\begin{array}{c} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{array} \begin{bmatrix} 0.55 & 0.71 & 0.29 \\ 0.51 & 0.89 & 0.90 \\ 0.13 & 0.21 & 0.05 \\ 0.44 & 0.03 & 0.46 \end{bmatrix} \cdot \begin{bmatrix} -0.4 \\ 0.2 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.009 \\ \underline{0.244} \\ 0.005 \\ -0.032 \end{bmatrix} \xrightarrow{\text{Softmax}} \begin{bmatrix} 0.238 \\ \underline{0.296} \\ 0.237 \\ 0.229 \end{bmatrix} \begin{array}{c} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{array}$$

$\eta$   
no probs

$\Downarrow$   
 $\sum = 1$

# Cross-Entropy Loss

## Cross-Entropy Loss for Binary Logistic Regression

$$L_{CE}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

## Generalized Cross-Entropy Loss for Multiclass Logistic Regression

$$L_{CE}(\hat{y}, y) = - \sum_{i=1}^C y_i \log(\hat{y}_i)$$

probability output after Softmax

$y_i = 1$  for correct class, 0 otherwise

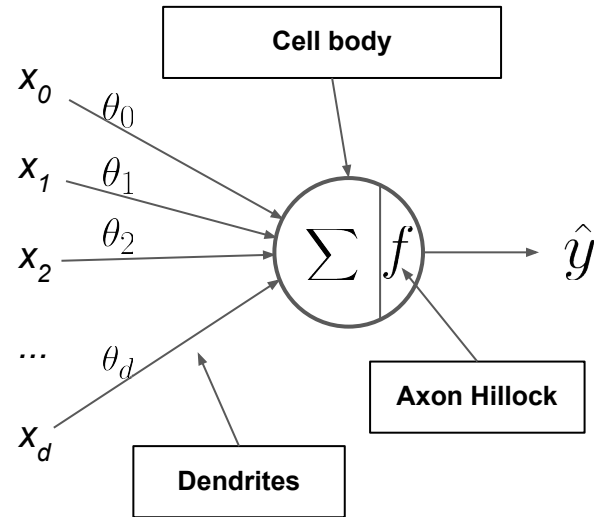
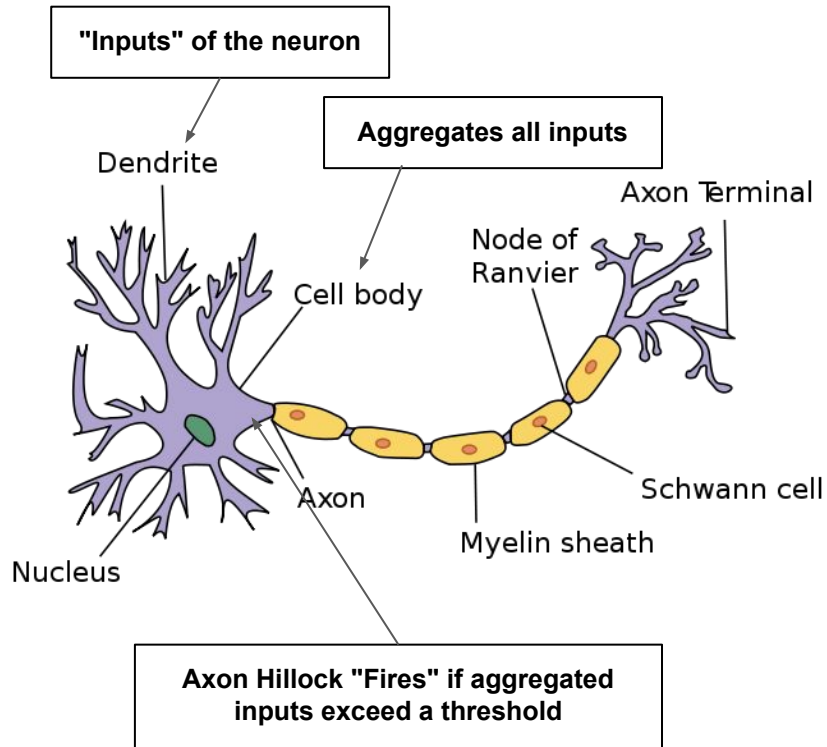
$C=2$

New gradient  $\nabla_{\theta} L_{CE}$  but beyond the scope here.

# Outline

- Generative vs. Discriminative Classifiers
- Logistic Regression
  - Setup as Probabilistic Classifier
  - Cross-Entropy Loss Function
  - Gradient Descent
  - Overfitting & Regularization
  - Multiclass Logistic Regression
- **Towards Neural Networks**
  - **Motivation: XOR Problem**
  - Basic Neural Network Architecture

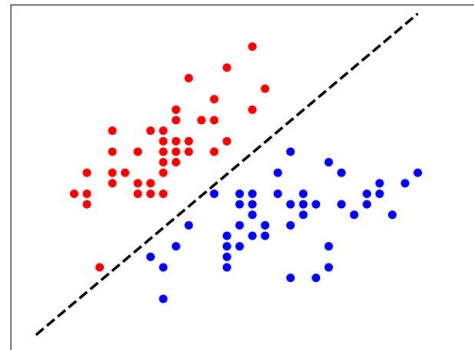
# Biological Inspiration — Neuron



→ Logistic Regression (crudely) a biological neuron

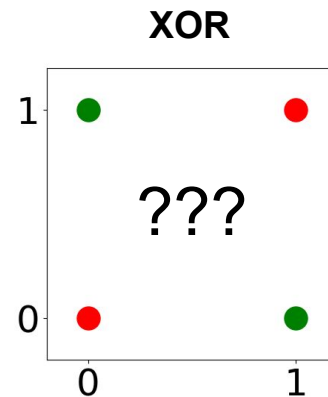
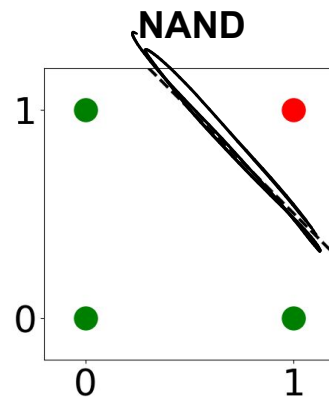
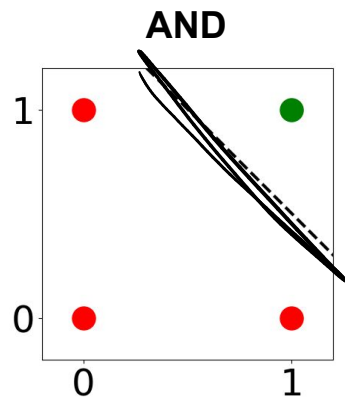
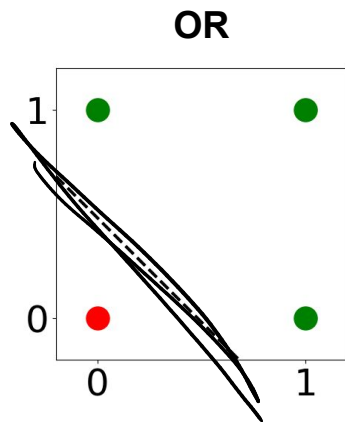
# Logistic Regression — Limitations

- Logistic Regression is a linear model
  - Limited to linear combination of features  
(and a non-linear mapping to a probability)
  - Limited to linear decision boundaries  
(i.e., lines, planes, hyperplanes)
- What if we want or need to represent non-linear relationships between features? We can't!
- Scale up: "Stacked" Logistic Regression
  - Feed input into multiple neurons (i.e., LR units)
  - Use output of neurons as input for other neurons



# XOR

$x_1$	$x_2$	OR	AND	NAND	XOR
0	0	0	0	1	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	1	1	0	0

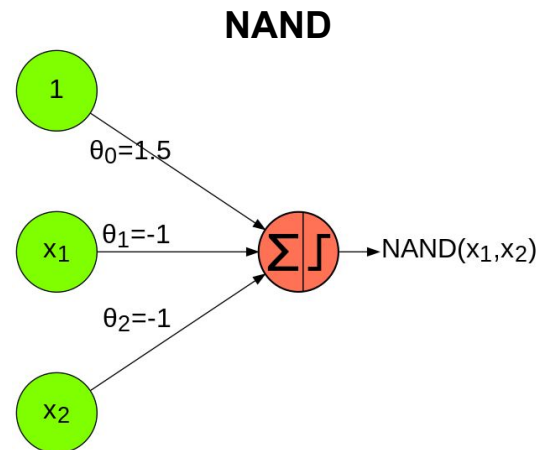
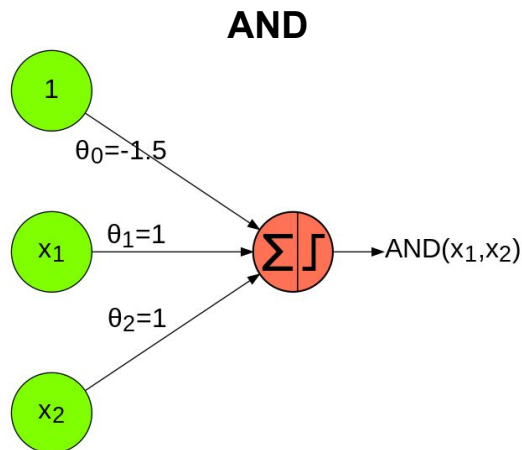
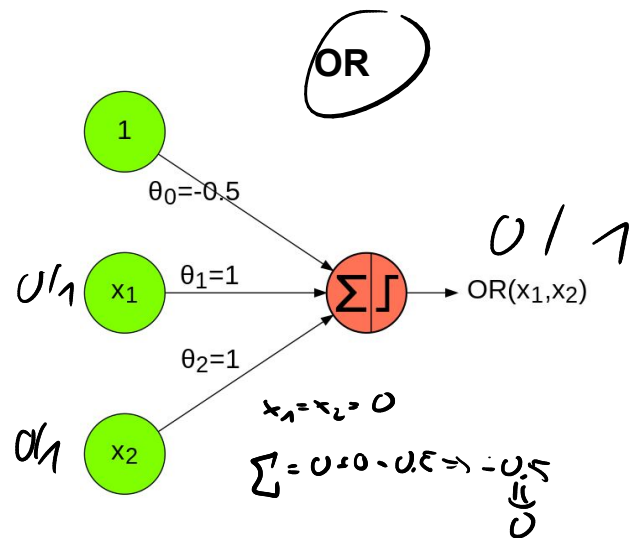


# XOR

## • Learning OR, AND, and NAND

- Finding correct weights simply by "looking hard"  
(the weights are not unique; there are many ways to set  $\theta$ )
- The activation function is the Step Function, not Sigmoid  
(strictly speaking, this makes it a Perceptron not a Linear Regression unit)

$$f_{step} = \begin{cases} 1 & , \text{if } \theta^T x > 0 \\ 0 & , \text{otherwise} \end{cases}$$

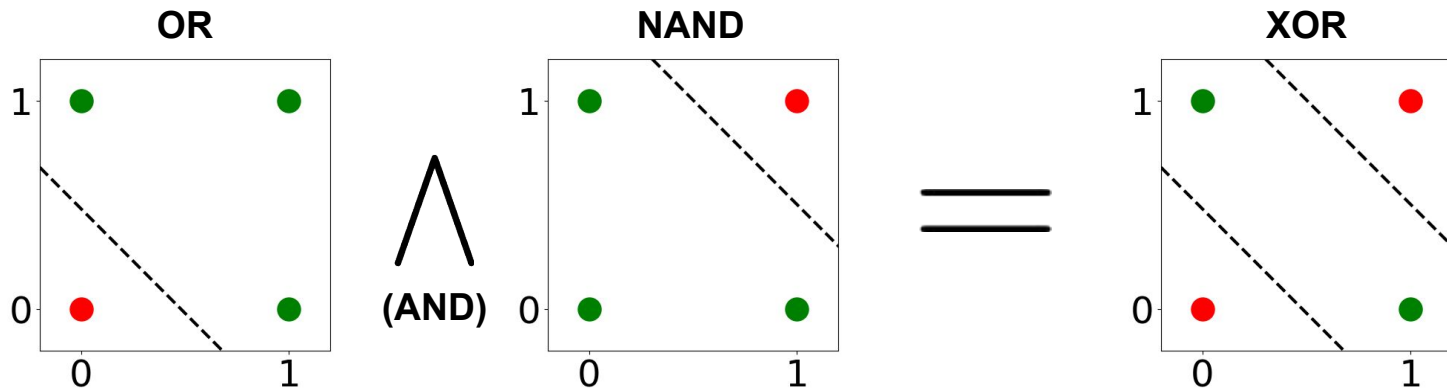




# XOR

- Deriving XOR from simple classifiers
  - Note: this is not the only way, just convenient

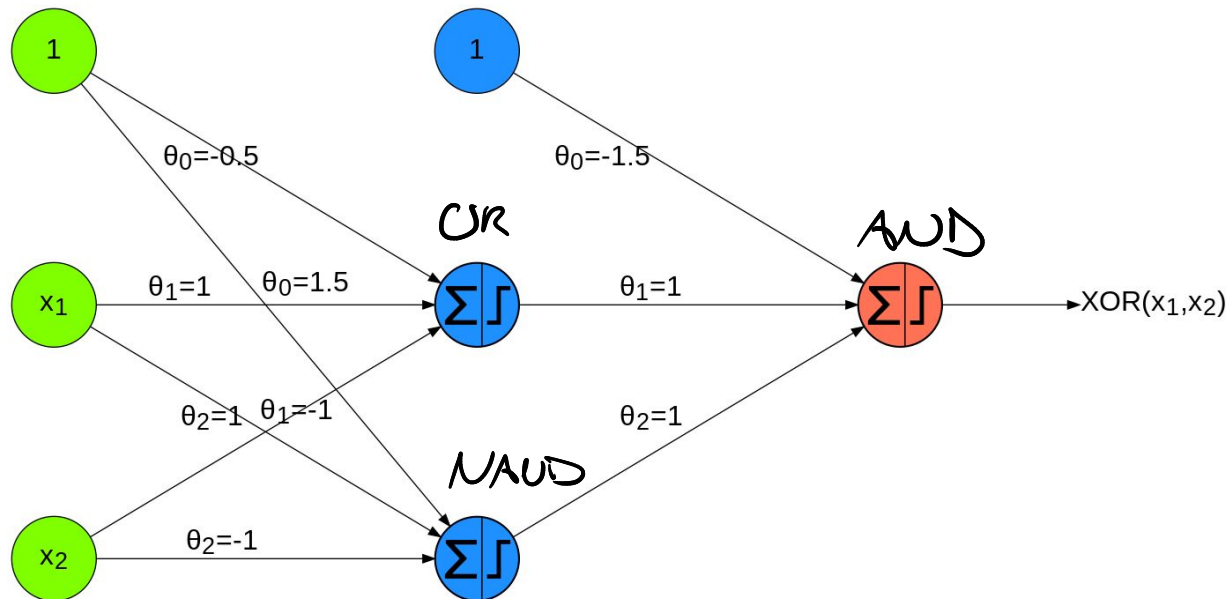
$x_1$	$x_2$	OR	AND	NAND	XOR
0	0	0	0	1	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	1	1	0	0



→ Cool, we know how to do ORs, ANDs and NANDs!

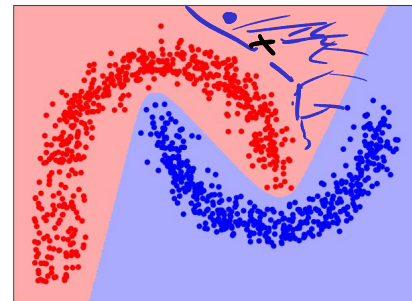
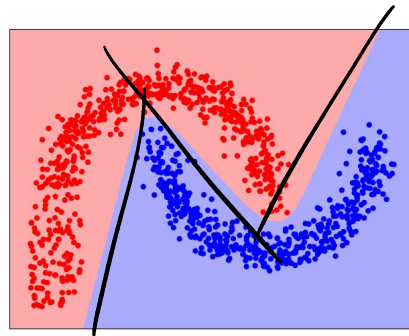
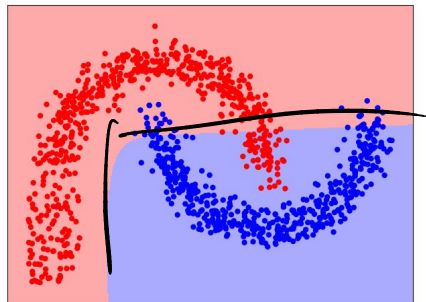
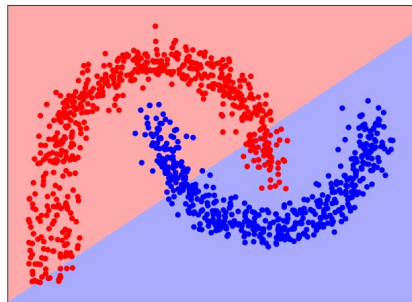
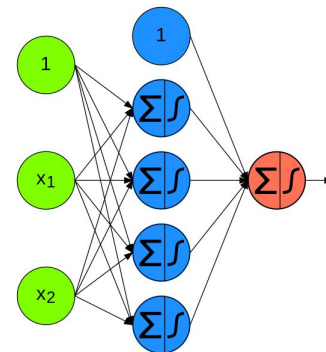
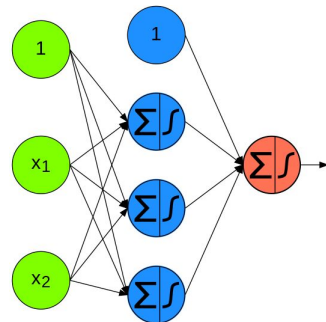
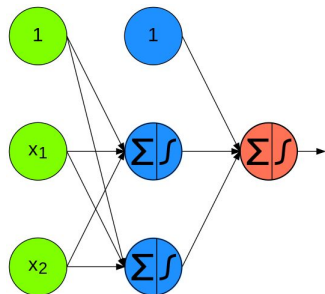
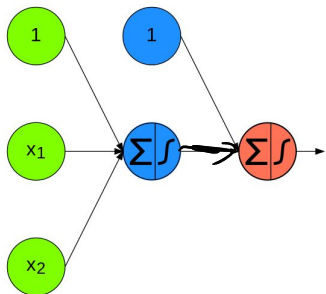
# XOR

- Modeling XOR by "stacking" LR units → **Neural Network (NN)**
  - More specifically, a **Feedforward NN** (i.e., network contains no loops)



# Network Capacity — Intuition

**Quick quiz:** Is there any harm to have too many neurons?



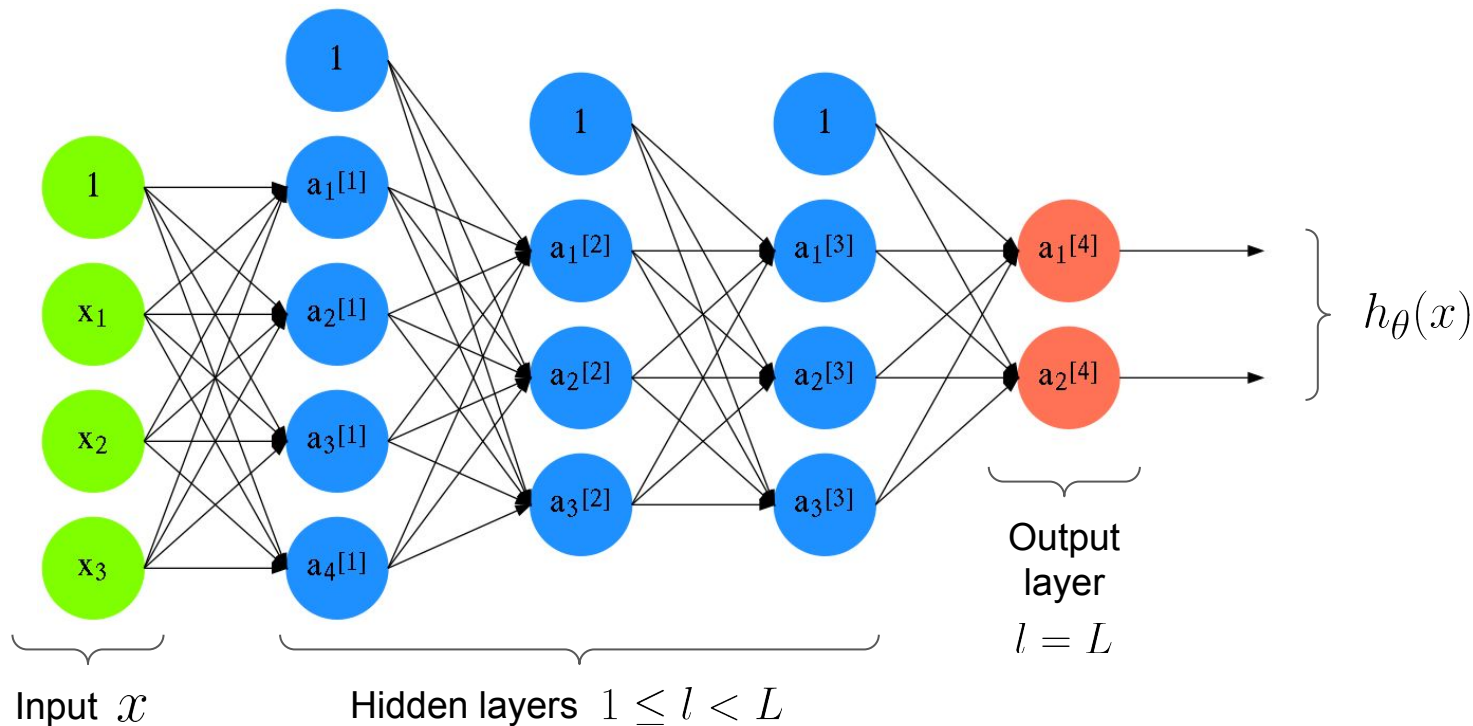
Note: The activation function is the Sigmoid, hence the smooth decision boundaries

# Outline

- Generative vs. Discriminative Classifiers
- Logistic Regression
  - Setup as Probabilistic Classifier
  - Cross-Entropy Loss Function
  - Gradient Descent
  - Overfitting & Regularization
  - Multiclass Logistic Regression
- **Towards Neural Networks**
  - Motivation: XOR Problem
  - **Basic Neural Network Architecture**

# A Neural Network (Feedforward NN)

- Example:  $L$ -layer Feedforward Neural Network (here:  $L = 4$ )



# Neural Network — Indices

Logistic Regression  
 $\theta_0, \theta_1, \dots, \theta_d$

$d^{[l]} = \text{\#neurons/units in layer } l$

$\theta^{[l]} = (d^{[l-1]} + 1) \cdot d^{[l]} = \text{\#weights for layer } l$

layer  $l$

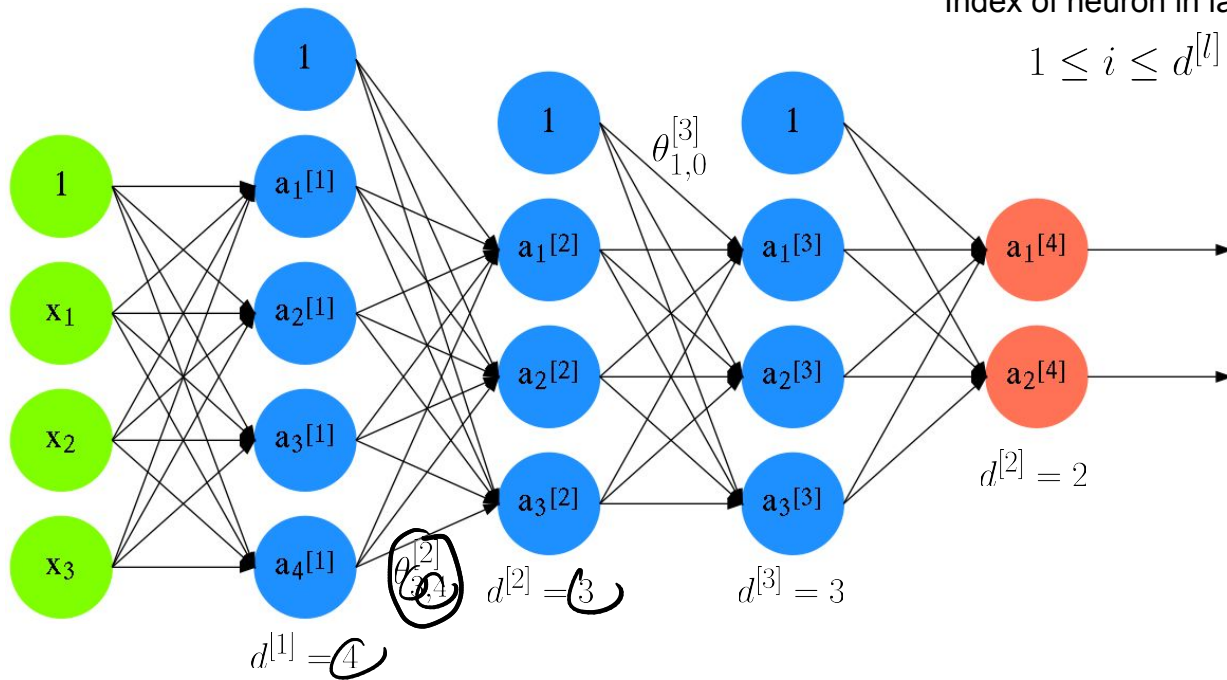
$\theta^{[l]}_{i,j}$

Index of neuron in layer  $l$

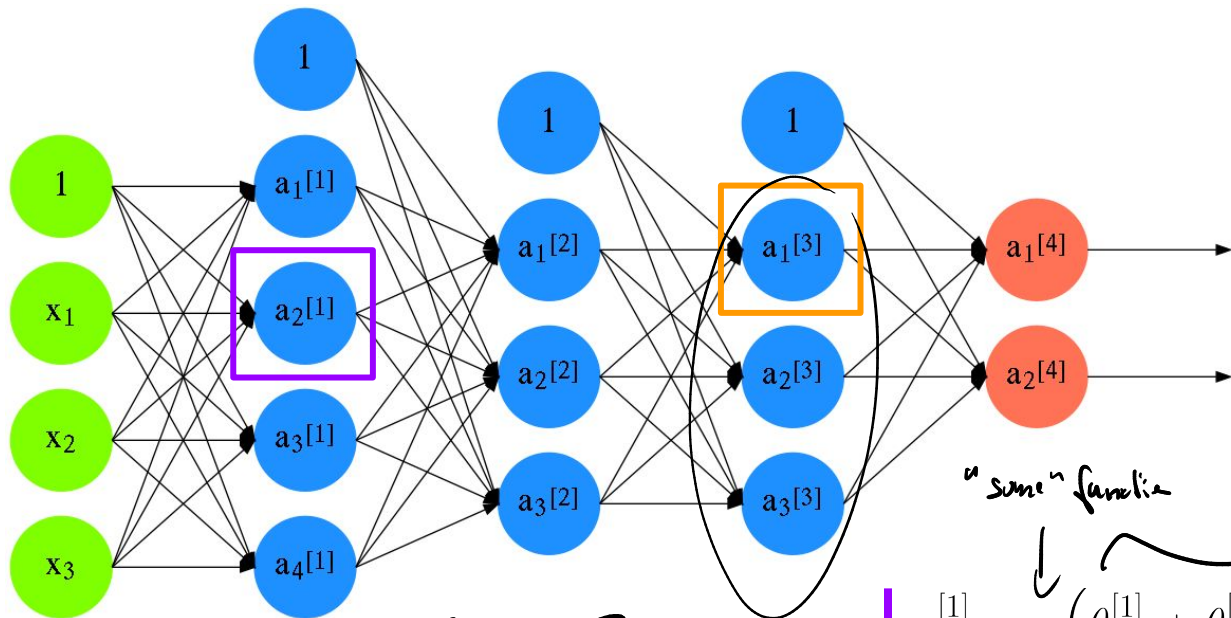
$$1 \leq i \leq d^{[l]}$$

Index of neuron in layer  $l-1$

$$0 \leq j \leq d^{[l-1]}$$



# Neural Network — Activations



$$a_2^{[1]} = g \left( \theta_{2,0}^{[1]} + \theta_{2,1}^{[1]}x_1 + \theta_{2,2}^{[1]}x_2 + \theta_{2,3}^{[1]}x_3 \right)$$

$$a_1^{[3]} = g \left( \theta_{3,0}^{[3]} + \theta_{3,1}^{[3]}a_1^{[2]} + \theta_{3,2}^{[3]}a_2^{[2]} + \theta_{3,3}^{[3]}a_3^{[2]} \right)$$

# Neural Network — Activations

- Layer-wise computations

- Let  $x^{[l]}$  be the output of layer  $l$
- $x^{[0]} = x$  — initial input
- $x^{[L]} = h(x)$  — final output

- Vectorized form

- Calculate  $x^{[l]}$  in practice "in one go"
- Everything becomes matrix\* operations
- GPUs: hardware-supported processing of matrix operations (+ parallelism)

$$x_i^{[l]} = a_i^{[l]} = g \left( \sum_{j=0}^{d^{[l-1]}} \theta_{i,j}^{[l]} x_j^{[l-1]} \right)$$

$$= g \left( \left[ \theta_i^{[l]} \right]^T \cdot x^{[l-1]} \right)$$

Weight vector  $\theta_i^{[l]} \in \mathbb{R}^{d^{[l-1]}}$

$$x^{[l]} = a^{[l]} = g \left( \theta^{[l]} x^{[l-1]} \right)$$

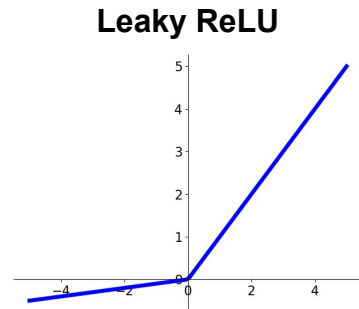
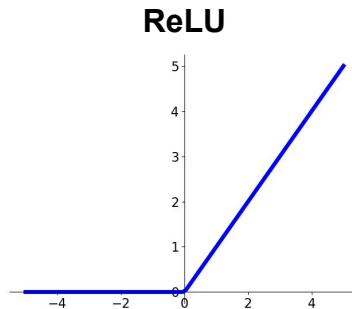
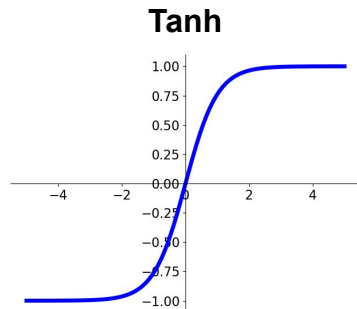
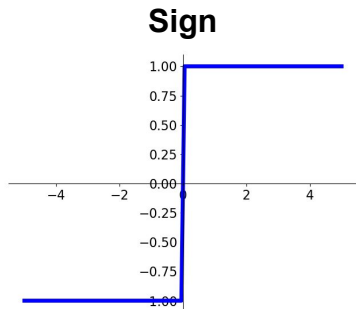
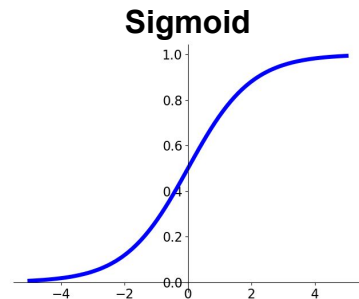
Weight matrix  $\theta^{[l]} \in \mathbb{R}^{d^{[l]} \times d^{[l-1]}}$

\*strictly speaking: tensor operations (tensor  $\approx$  n-dimensional arrays)



# Neural Network — Activation Functions

- Wide range of activation functions
- Activations functions for hidden layers
  - Do not need to have a probabilistic interpretation
  - Only requirement: non-linear function!
  - Examples:

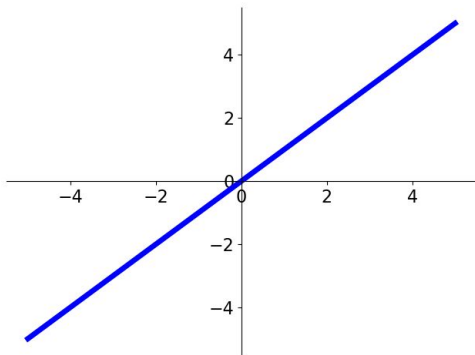


ReLU = Rectified Linear Unit

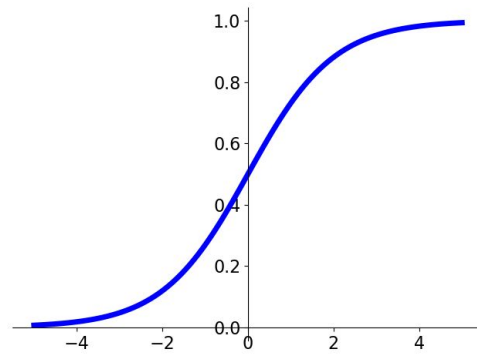
# Neural Network — Activation Functions

- Activations functions for output layers
  - Choice of activation function depending on task  
(mainly: classification or regression)
  - Examples:

**Linear** function for regression tasks

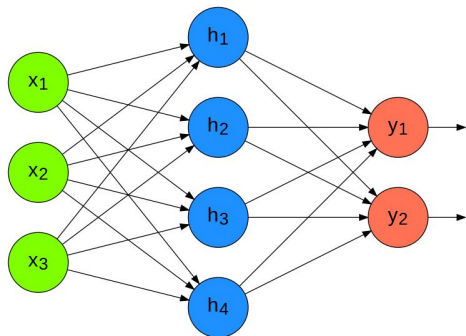


**Sigmoid** function for classification tasks



# Example

Input  $x$       Hidden  $h$       Output  $y$



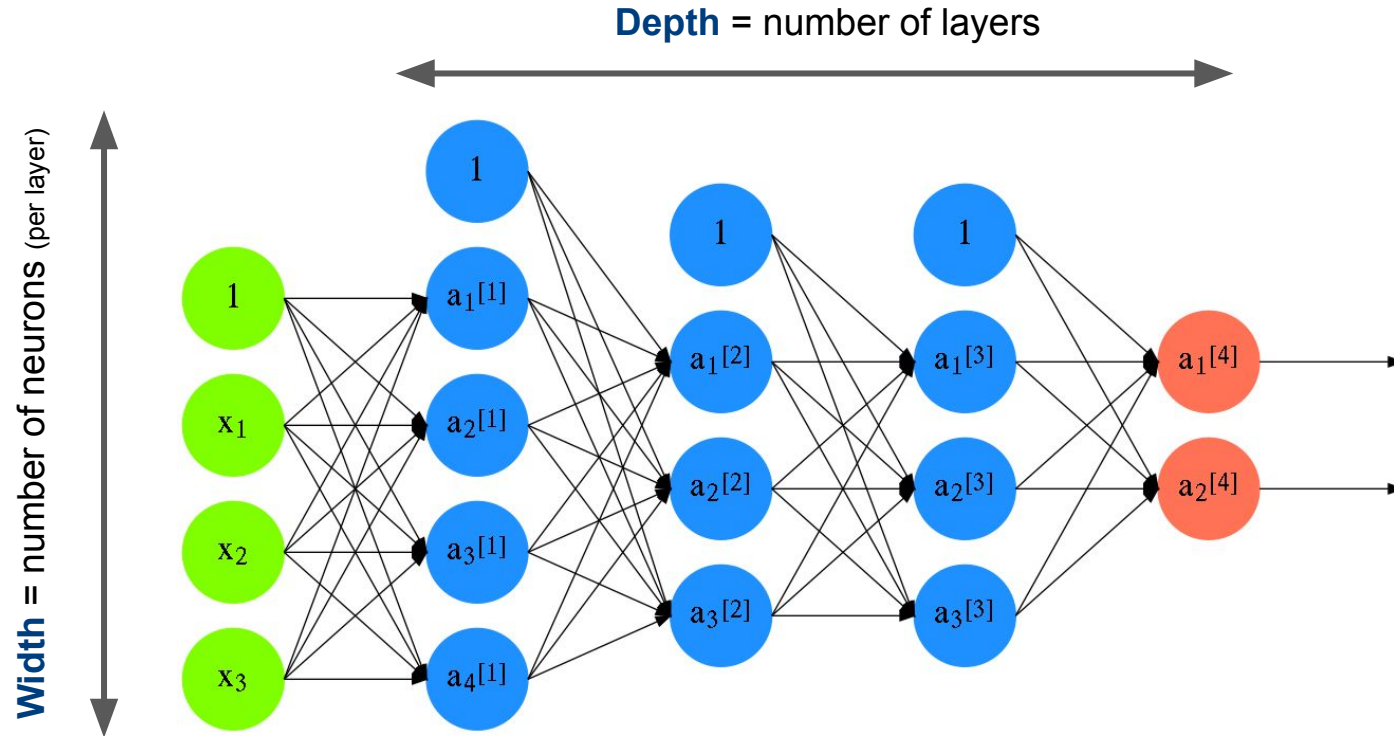
$$h = g_h(\theta_h x) \text{ , with } \theta_h \in \mathbb{R}^{4 \times 3}$$

$$y = g_y(\theta_y h) \text{ , with } \theta_y \in \mathbb{R}^{2 \times 4}$$

$g_h, g_y$  : suitable activation functions

$$\begin{array}{ccccccc}
 \theta_h & x & \theta_h x & & h & \theta_y & h & \theta_y h & y \\
 \left\{ \begin{bmatrix} 0.55 & 0.71 & 0.29 \\ 0.51 & 0.89 & 0.90 \\ 0.13 & 0.21 & 0.05 \\ 0.44 & 0.03 & 0.46 \end{bmatrix} \cdot \begin{bmatrix} -0.4 \\ 0.2 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.009 \\ 0.244 \\ 0.005 \\ -0.032 \end{bmatrix} \right. & \xrightarrow{\text{ReLU}(\theta_h x)} & \begin{bmatrix} 0.009 \\ 0.244 \\ 0.005 \\ 0 \end{bmatrix} & \xrightarrow{\theta_y} & \begin{bmatrix} 0.65 & 0.28 & 0.68 & 0.59 \\ 0.02 & 0.56 & 0.26 & 0.42 \end{bmatrix} \cdot \begin{bmatrix} 0.009 \\ 0.244 \\ 0.005 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.078 \\ 0.138 \end{bmatrix} & \xrightarrow{\text{Softmax}(\theta_y h)} & \begin{bmatrix} 0.48 \\ 0.52 \end{bmatrix} \\
 \text{4 classes} & & & & & \uparrow \text{not probs} & \sum=1
 \end{array}$$

# Neural Networks



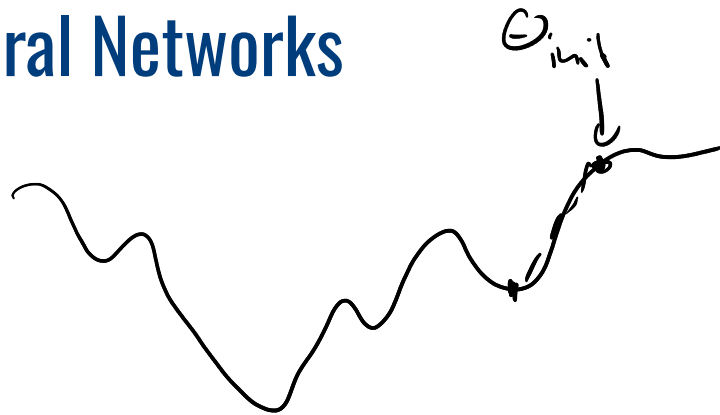
# From Logistic Regression to (Deep) Neural Networks

- Fundamentally, nothing new here:

- A neural network is a function  $h_{\theta}(x)$
- Define a loss function  $L = L(y, \hat{y}) = L(y, h_{\theta}(x))$
- Perform Gradient Descent to minimize  $L$

- Difference: increased complexity

- $h_{\theta}(x)$  and thus  $L(y, h_{\theta}(x))$  are much more complex functions
- Calculation of  $\frac{\partial L}{\partial \theta}$  much more challenging → backpropagation
- $L$  is no longer a convex function → local minima → training more challenging
- Overfitting becomes a bigger issue → regularization (several different approaches)



# Outline

- Generative vs. Discriminative Classifiers
- Logistic Regression
  - Setup as Probabilistic Classifier
  - Cross-Entropy Loss Function
  - Gradient Descent
  - Overfitting & Regularization
  - Multiclass Logistic Regression
- Towards Neural Networks
  - Motivation: XOR Problem
  - Basic Neural Network Architecture

# Summary

- Linear model: **Logistic Regression**

- Very important probabilistic classifier
- Discriminative classifier → linear decision boundaries
- Core unit of neural networks

- "Stacked" Logistic Regression → Neural Network

- Neuron = Linear Regression unit
- Non-convex loss function → global minimum vs. local minima
- Higher risk of overfitting → regularization crucial (but also other methods)

# Pre-Lecture Activity for Next Week

- Assigned Task

- Post a 1-2 sentence answer to the following question into the L2 Discussion  
(you will find the Discussion on Canvas)

*"What do we mean by sparse or dense vectors?"*

*"Are documents characterized by tf-idf sparse or dense?"*

**Side notes:**

- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers but this won't help you learn better



# Solutions to Quick Quizzes

- Slide 43: C

- The ground truth is 0 or 1, while the prediction will only ever be (very) close to 0 or 1
- At least with arbitrary mathematical precision, there will always be some loss calculated

- Slide 54: B

- In case Logistic Regression, scaling does not change the performance of the model
- However, the values of the coefficients do change

- Slide 67

- Too many neurons may lead to overfitting
- Example: the model might try to "learn" an outlier