



CS 760: Machine Learning **Neural Networks**

Ilias Diakonikolas

University of Wisconsin-Madison

Oct. 8, 2022

Logistics

- **Announcements:**

- HW 3 out soon

- **Class roadmap:**

Thursday, Oct. 6	Neural Networks I
Tuesday, Oct. 11	Neural Networks II
Thursday, Oct. 13	Neural Networks III
Tuesday, Oct. 18	Neural Networks IV
Thursday, Oct. 20	Neural Networks V



Outline

- **Review & Perceptron Algorithm**
 - Definition, Training, Loss Equivalent, Mistake Bound
- **Neural Networks**
 - Introduction, Setup, Components, Activations
- **Training Neural Networks**
 - SGD, Computing Gradients, Backpropagation

Outline

- **Review & Perceptron Algorithm**
 - Definition, Training, Loss Equivalent, Mistake Bound
- **Neural Networks**
 - Introduction, Setup, Components, Activations
- **Training Neural Networks**
 - SGD, Computing Gradients, Backpropagation

Review: Function Representations

- How much does it cost to “store” a function?
 - Last time in the context of PMFs
- Two representations for f

Lookup Table

x	f(x)
0	0
1	1
2	4
3	9
4	16
5	25
⋮	

Expression

$$f(x) = x^2$$

Cost: if x is an integer between 0 and 99:

- Lookup table: **100** entries
- Expression: if quadratic, **3** coefficients ax^2+bx+c : store $(1,0,0)$

Review: Probability Functions

- Say our domain is $\{-1, +1\}^d$
- Function is $P(x_1, x_2, \dots, x_d) \dots$
 - Suppose they're **independent** $P(x_1)P(x_2)\dots P(x_n)$

Lookup Table

x	f(x)
-1,-1,-1,...,-1,-1	0.002
-1,-1,-1,...,-1,1	0.0032
-1,-1,-1,...,1,-1	0.12
-1,-1,-1....1,-1	0.00003
...	
1,1,1,...,1	0.328

Cost:

Lookup table: 2^d entries

Factorized Lookup Table

x_1	$f(x)$	x_2	$f(x)$	\dots	x_d	$f(x)$
-1	0.42	-1	0.12	...	-1	0.2
1	0.58	1	0.88		1	0.8

Cost: Lookup tables: 2^d entries

$$0.002 = P(-1, -1, \dots, -1) = 0.42 \times 0.12 \times \dots \times 0.2$$

Review: Probability Functions

- Say our domain is $\{-1,+1\}^d$
- Function is $P(x_1, x_2, \dots, x_d) \dots$
 - If **independent** $P(x_1)P(x_2)\dots P(x_n)$ and **identically distributed**

Lookup Table

x	f(x)
-1,-1,-1,...,-1,-1	0.002
-1,-1,-1,...,-1,1	0.0032
-1,-1,-1,...,1,-1	0.12
-1,-1,-1....1,-1	0.00003
...	
1,1,1,...,1	0.328

Cost:

Lookup table: 2^d entries

Factorized Lookup Table

x_1	f(x)	x_2	f(x)	\dots	x_d	f(x)
-1	0.42	-1	0.12	• • •	-1	0.2
1	0.58	1	0.88		1	0.8

Cost: Lookup tables: $2d$ entries

x_i	f(x)
-1	0.42
1	0.58

Cost: Lookup table: 2 entries

Review: Naïve Bayes Core Assumption

- Conditional **independence** of features:

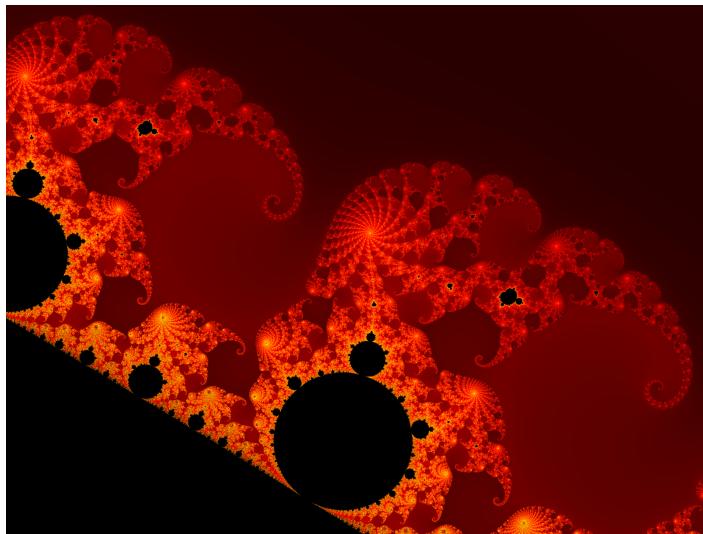
$$\begin{aligned} P(X_1, \dots, X_K, Y) &= P(X_1, \dots, X_K | Y)P(Y) \\ &= \left(\prod_{k=1}^K P(X_k | Y) \right) P(Y) \end{aligned}$$

- What do we gain? With binary features, get 2 entries per feature
- So, number of probabilities

$$2^k \rightarrow 2k$$

Review: Function Representations

- More general: consider some object O
 - We did pairs (x,y) for function f
 - **Kolmogorov complexity** of O : shortest program that outputs O



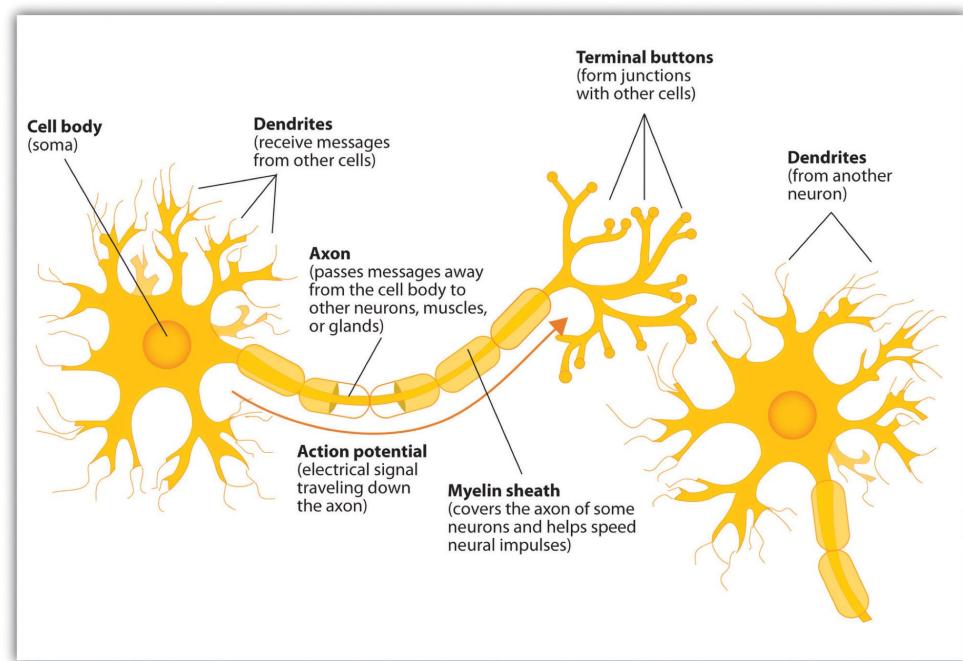
23 MB to store image (ie, bitmap)
A few bytes for program



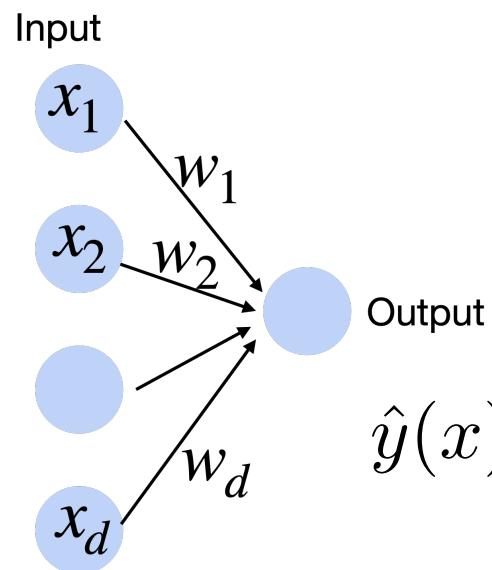
A. N. Kolmogorov

Neural networks: Origins

- *Artificial neural networks, connectionist models*
- Inspired by interconnected neurons in biological systems
 - Simple, homogenous processing units



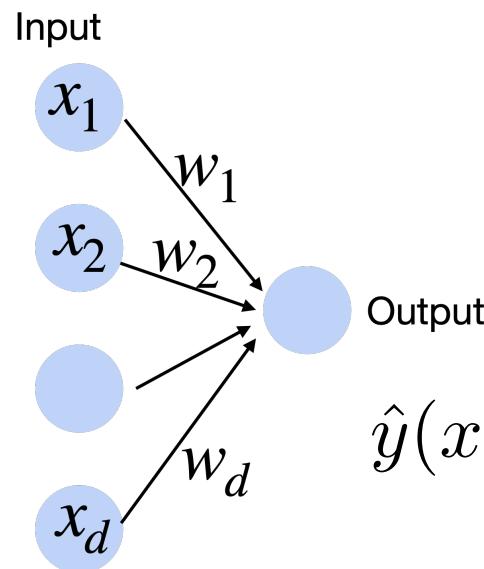
Perceptron: Simple Network



$$\hat{y}(x) = \begin{cases} 1 & w^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

[McCulloch & Pitts, 1943; Rosenblatt, 1959; Widrow & Hoff, 1960]

Perceptron: Components



$$\hat{y}(x) = \begin{cases} 1 & w^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$f = w^T x \quad \sigma(a) = \begin{cases} 1 & a \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad \hat{y}(x) = \sigma(w^T x)$$

Activation Function

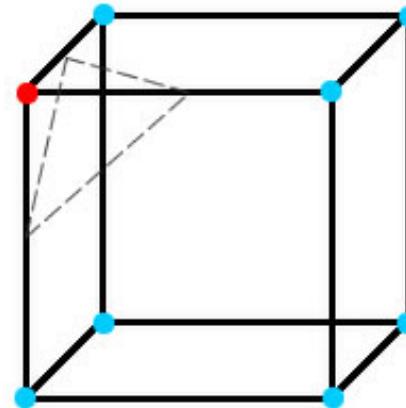
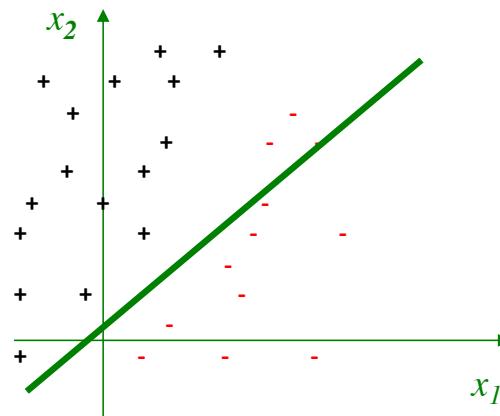
[McCulloch & Pitts, 1943; Rosenblatt, 1959; Widrow & Hoff, 1960]

Perceptron: Representational Power

- Perceptrons can represent only *linearly separable* concepts

$$\hat{y}(x) = \begin{cases} 1 & w^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

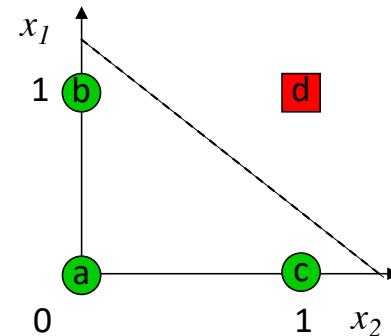
- Decision boundary given by:



Which Functions are Linearly Separable?

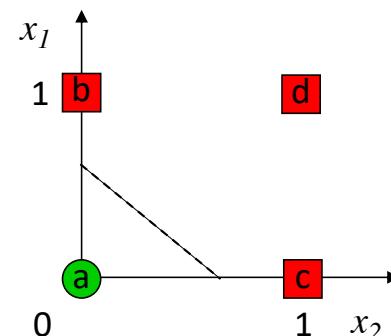
AND

	x_1	x_2	y
a	0	0	0
b	0	1	0
c	1	0	0
d	1	1	1



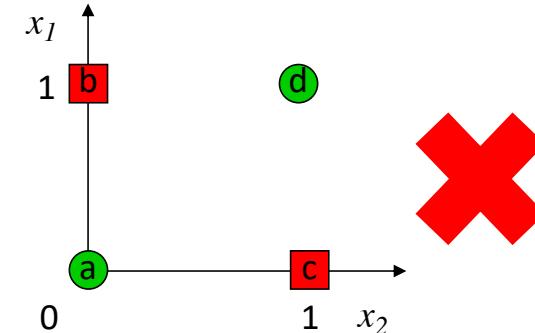
OR

	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	1

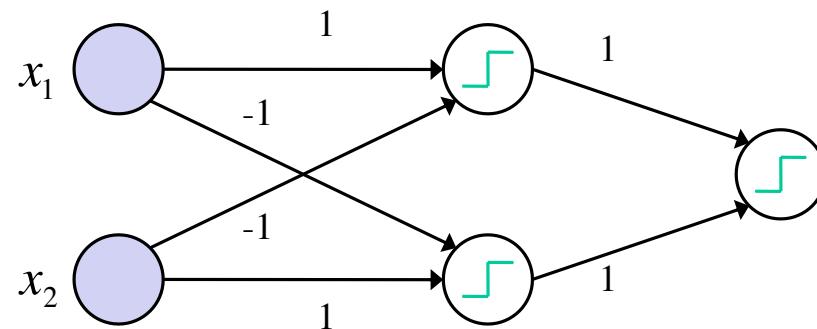


Which Functions are Linearly Separable?

	<u>XOR</u>		
	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	0



A multilayer perceptron
can represent XOR!



assume $w_0 = 0$ for all nodes

Perceptron: Training

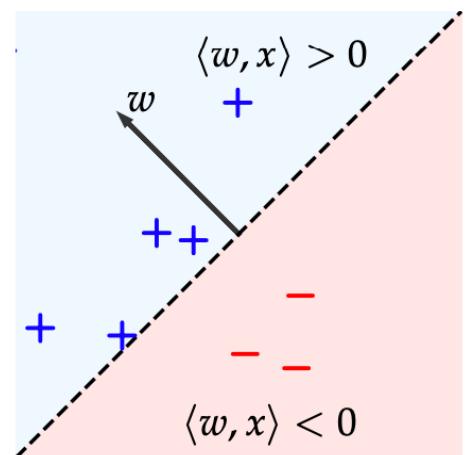
- When are we correct?

$$y^{(i)} w^T x^{(i)} > 0$$

- I.e., **signs** of prediction and label match
- In training, could ask for “margin”: insist

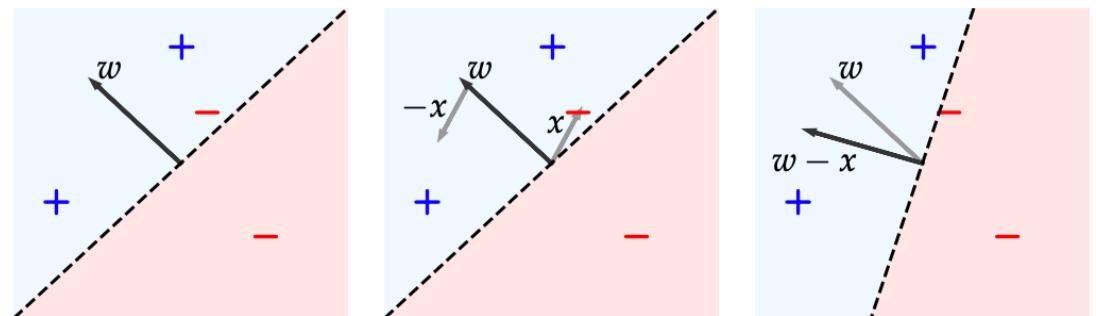
$$y^{(i)} w^T x^{(i)} \geq c$$

- More than we needed



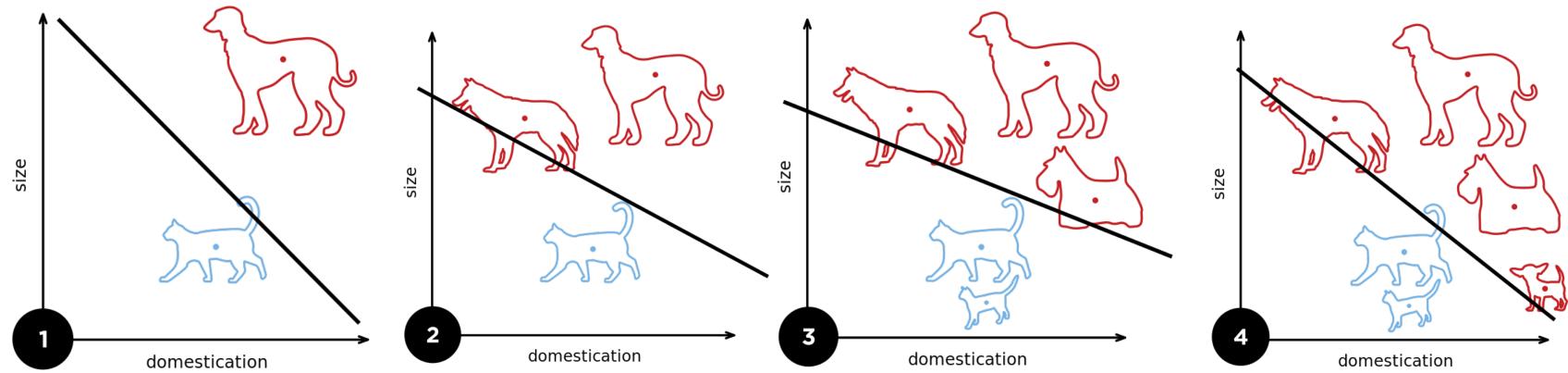
Perceptron: Training

- **Algorithm:**
 - Initialize at $w_0 = [0, \dots, 0]^T$
 - At step $t = 0, \dots$
 - Select index i ,
 - If $y^{(i)} w^T x^{(i)} < 1$ then do $w_{t+1} = w_t + y^{(i)} x^{(i)}$
 - Else, $w_{t+1} = w_t$



Perceptron: Training

- Algorithm training example:



Perceptron: Training Comparison

- We're used to minimizing some loss function...
- Taking one example at a time...
 - Stochastic Optimization (like SGD!)
- **Step:** $w_{t+1} = w_t + y^{(i)}x^{(i)}$
 - What update to our prediction?

$$w_{t+1}^T x^{(i)} = w_t^T x^{(i)} + y^{(i)} \|x^{(i)}\|^2$$

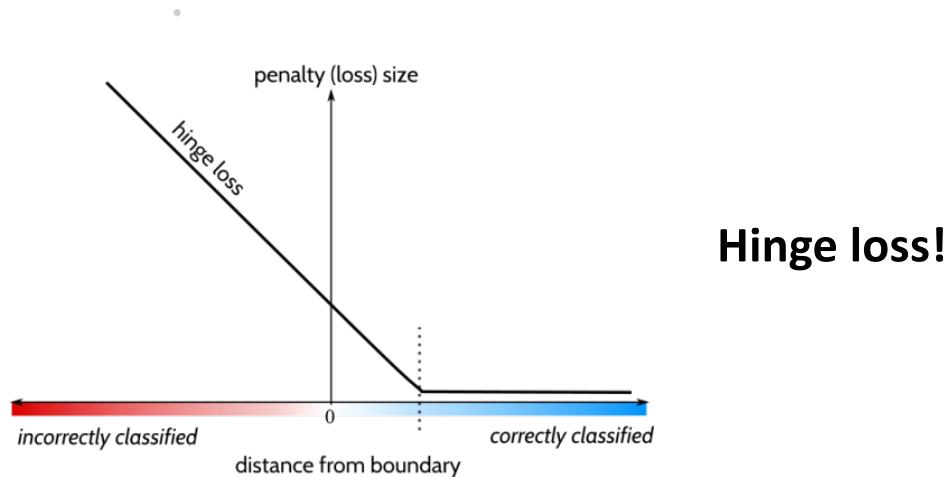
Perceptron: Training Comparison

- So: looks like **SGD** with a loss function L

SGD $w_{t+1} = w_t - \alpha \nabla L(f(x^{(i)}, y^{(i)})$

Perceptron $w_{t+1} = w_t + y^{(i)} x^{(i)}$

- Need: gradient is 0 when we're right, $y^{(i)}x^{(i)}$ on mistakes



Perceptron: Analysis

- Two aspects to analysis: **fitting training data** + **generalization**
- **Mistake bound:**

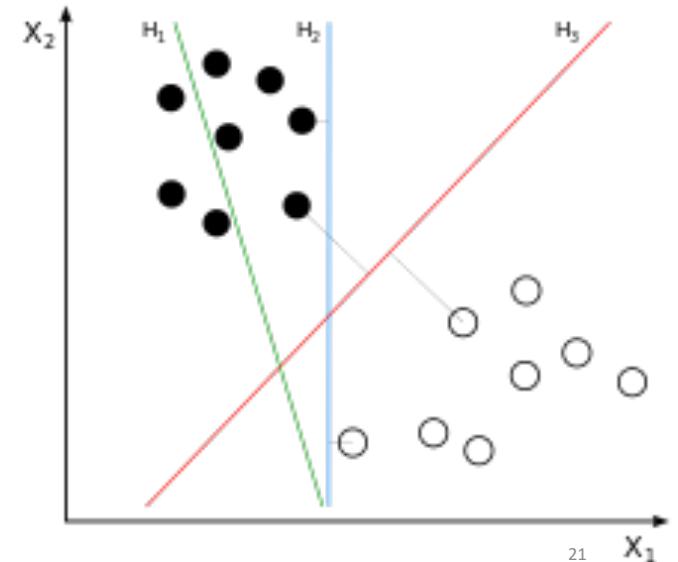
- Hyperplane $H_w = \{x : w^T x = 0\}$

- Margin

$$\gamma(S, w) = \min_{1 \leq i \leq n} \text{dist}(x^{(i)}, H_w)$$

↓
 $|x^T w| / \|w\|$

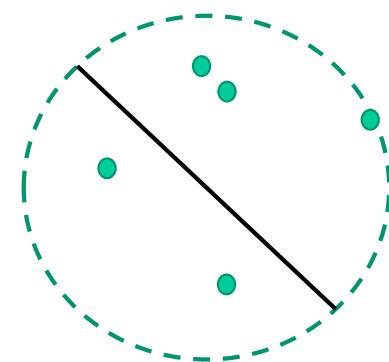
$$\gamma(S) = \max_{\|w\|=1} \gamma(S, w)$$



Perceptron: Mistake Bound

- Need some information about our data:

- “**Diameter**”: $D(S) = \max_{(x,y) \in S} \|x\|$



- **Mistake Bound Result:**

- The total # of mistakes on a linearly separable set S is at most

$$(2 + D(S)^2)\gamma(S)^{-2}$$

Perceptron: Mistake Bound Interpretation

- **Mistake Bound Result:**

- The total # of mistakes on a linearly separable set S is at most

$$(2 + D(S)^2)\gamma(S)^{-2}$$

↑
Diameter: Controls our
biggest step.

← →
Margin: Smaller
means harder to find
separator

- **Scaling?**

- **Implications?**

- Run over dataset D repeatedly. # mistakes doesn't change
 - Eventually get perfect separation on a copy of D



Mistake Bound: Proof 1

- Let's prove the result.
 - Intuitive idea we exploit: **norm of weight vector** <-> # mistakes
- Start with changes in weight norm

$$\|w_{t+1}\|^2 = \|w_t + y^{(i_t)} x^{(i_t)}\|^2 \quad \text{If mistake}$$

$$\|w_{t+1}\|^2 = \|w_t\|^2 + 2(y^{(i_t)})^T x^{(i_t)} + \|x^{(i_t)}\|^2$$



$$\boxed{\|w_{t+1}\|^2 \leq \|w_t\|^2 + 2 + D(S)^2}$$

Mistake Bound: Proof 2

- This is true for each mistake

$$\|w_{t+1}\|^2 \leq \|w_t\|^2 + 2 + D(S)^2$$

- Let m_t be # mistakes by t step. Start at w_0 (norm 0). By w_t

$$\|w_t\| \leq \sqrt{m_t(2 + D(S)^2)}$$

- This was also a telescoping argument; recall proof of GD convergence

Mistake Bound: Proof 3

- Now we'll also lower bound norm
- Let w be a hyperplane that **separates, unit norm**. $\|w\| = 1$

$$w^T(w_{t+1} - w_t) = w^T(y^{(i_t)}x^{(i_t)}) = \frac{|w^T x^{(i_t)}|}{\|w\|}$$

mistake

\leftarrow **w classifies correctly**
 \leftarrow **Norm 1**

- But this is the margin for $x^{(it)}$, so:

$$\frac{|w^T x^{(i_t)}|}{\|w\|} \geq \gamma(S, w)$$

Mistake Bound: Proof 4

- So:

$$w^T(w_{t+1} - w_t) \geq \gamma(S, w)$$

- Let's look at our best solution: w_* , the maximum margin w
- From Cauchy-Schwartz $\|w_t\| \|w_*\| \geq w_*^T w_t$
- Let's set up a telescoping sum:

$$\|w_t\| \geq w_*^T w_t = \sum_{k=1}^t w_*^T (w_k - w_{k-1})$$

Mistake Bound: Proof 5

- Have: $w^T(w_{t+1} - w_t) \geq \gamma(S, w)$

$$\|w_t\| \geq w_*^T w_t = \sum_{k=1}^t w_*^T (w_k - w_{k-1})$$

- Combine:

$$\|w_t\| \geq w_*^T w_t = \sum_{k=1}^t w_*^T (w_k - w_{k-1}) \geq m_t \gamma(S)$$

0 for no mistake,
Purple for mistake

- Note: $\gamma(S, w_*) = \gamma(S)$

Mistake Bound: Proof 6

- So, $m_t \gamma(S) \leq \|w_t\|$ $\|w_t\| \leq \sqrt{m_t(2 + D(S)^2)}$

- I.e., $m_t \gamma(S) \leq \sqrt{m_t(2 + D(S)^2)}$

- Easy algebra gets us to

$$m_t \leq \frac{2 + D(S)^2}{\gamma(S)^2} \quad \checkmark$$



Break & Quiz

Q1-1: Select the correct option.

- A. *A perceptron is guaranteed to perfectly learn a given linearly well-separable function within a finite number of training steps.*
 - B. *A single perceptron can compute the XOR function.*
-
- 1. Both statements are true.
 - 2. Both statements are false.
 - 3. Statement A is true, Statement B is false.
 - 4. Statement B is true, Statement A is false.

Q1-1: Select the correct option.

- A. *A perceptron is guaranteed to perfectly learn a given linearly well-separable function within a finite number of training steps.*
 - B. *A single perceptron can compute the XOR function.*
-
- 1. Both statements are true.
 - 2. Both statements are false.
 - 3. Statement A is true, Statement B is false.
 - 4. Statement B is true, Statement A is false.
- 

Outline

- **Review & Perceptron Algorithm**
 - Definition, Training, Loss Equivalent, Mistake Bound
- **Neural Networks**
 - Introduction, Setup, Components, Activations
- **Training Neural Networks**
 - SGD, Computing Gradients, Backpropagation

Multilayer Neural Network

- Input: two features from spectral analysis of a spoken sound
- Output: vowel sound occurring in the context “h__d”

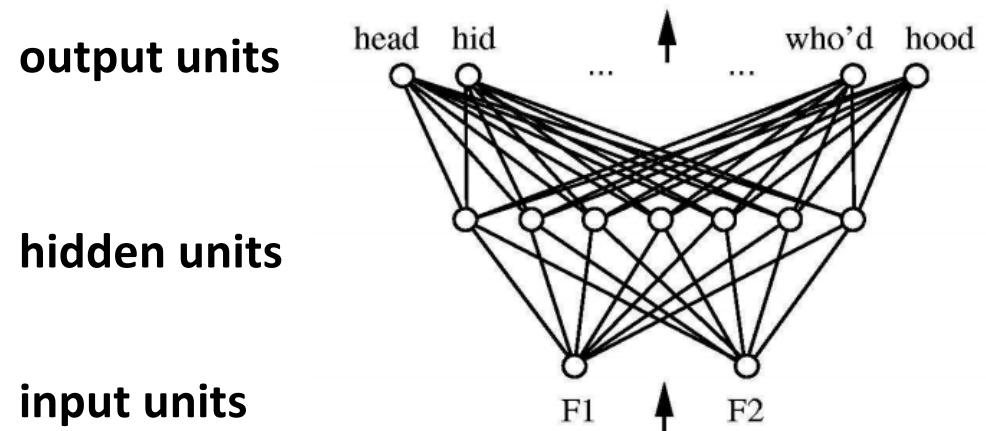


figure from Huang & Lippmann, NIPS 1988

Neural Network Decision Regions

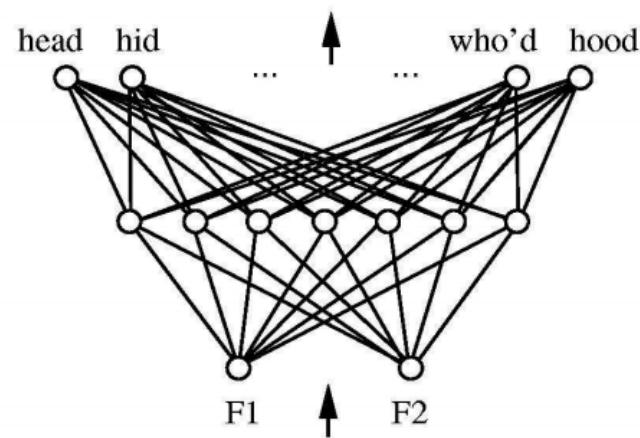
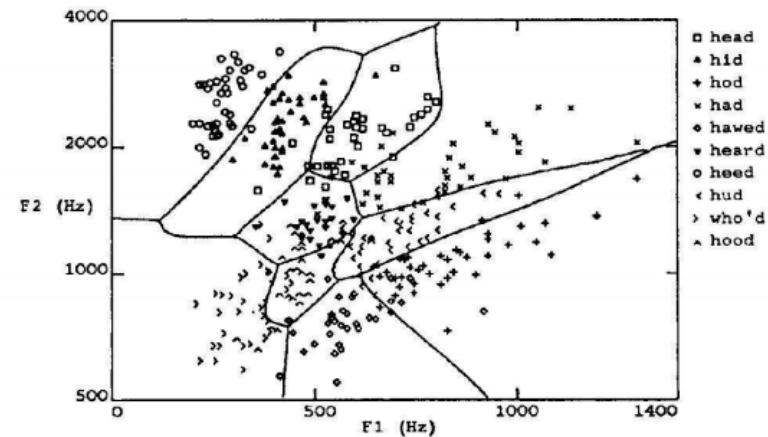
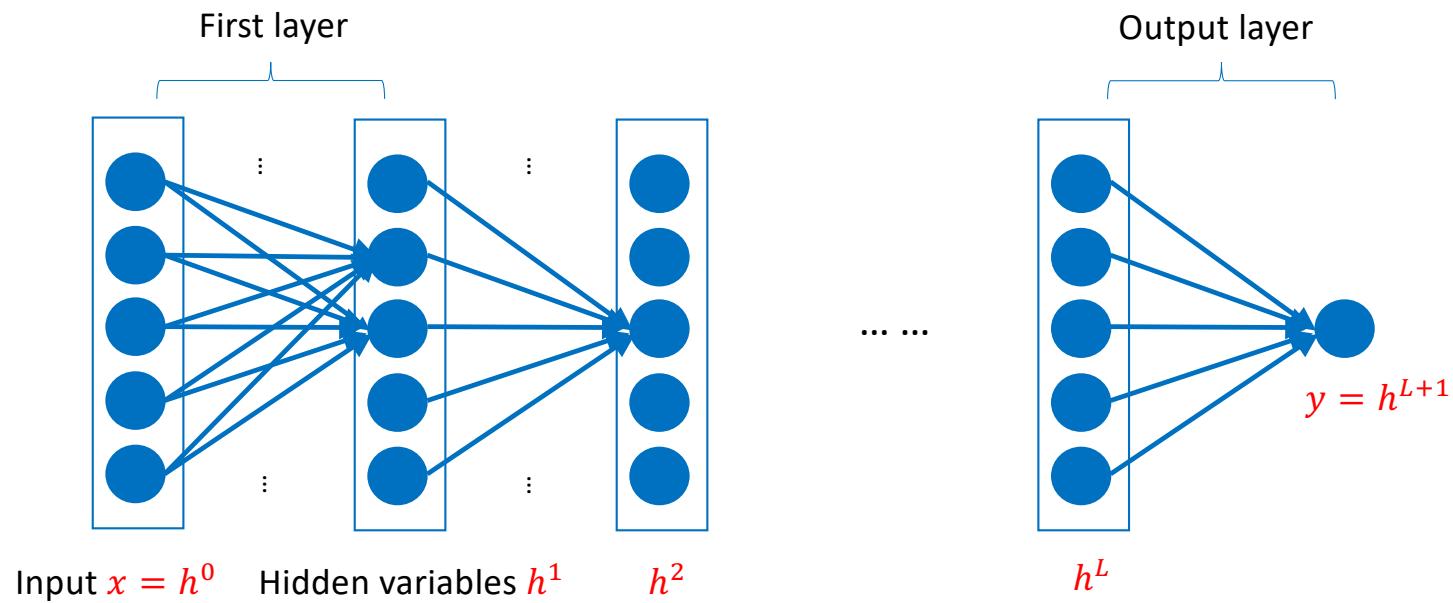


Figure from Huang & Lippmann, NIPS 1988



Neural Network Components

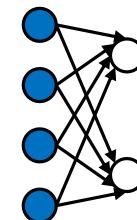
An $(L + 1)$ -layer network



Feature Encoding for NNs

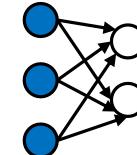
- Nominal features usually a one hot encoding

$$A = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad G = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad T = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$



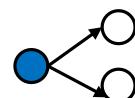
- Ordinal features: use a *thermometer* encoding

$$\text{small} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{medium} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad \text{large} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$



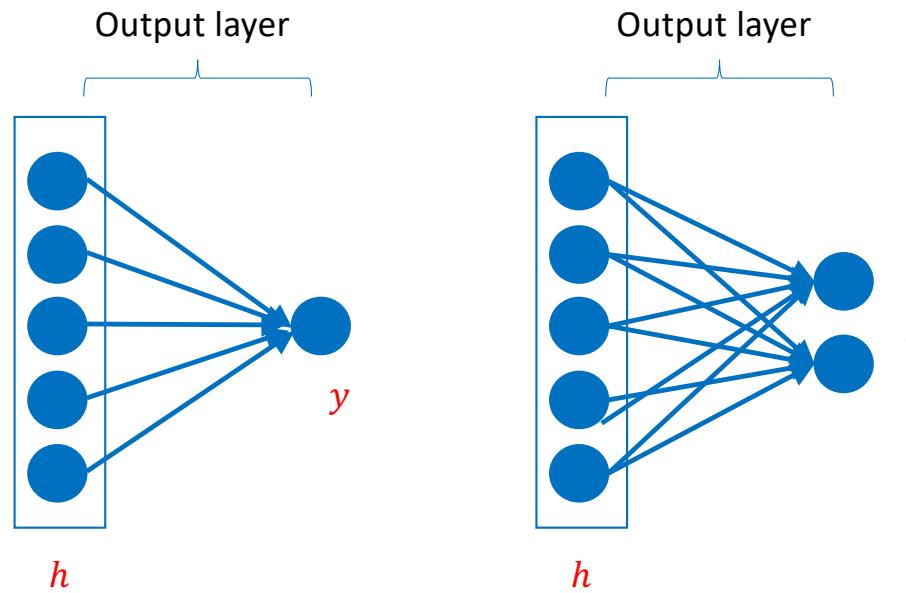
- Real-valued features use individual input units (may want to scale/normalize them first though)

$$\text{precipitation} = [0.68]$$



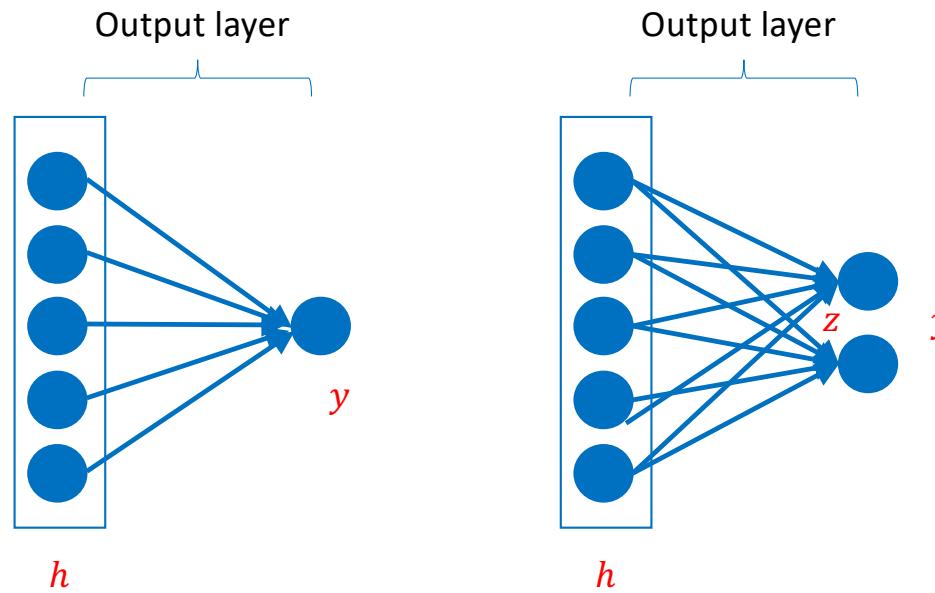
Output Layer: Examples

- Regression: $y = w^T h + b$
 - Linear units: no nonlinearity
- Multi-dimensional regression: $y = W^T h + b$
 - Linear units: no nonlinearity



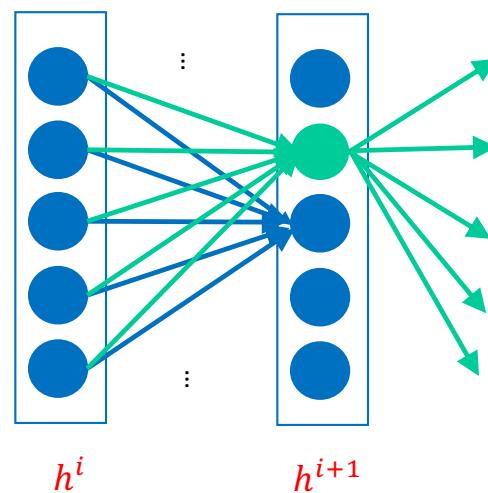
Output Layer: Examples

- Binary classification: $y = \sigma(w^T h + b)$
 - Corresponds to using logistic regression on h
- Multiclass classification:
 - $y = \text{softmax}(z)$ where $z = W^T h + b$



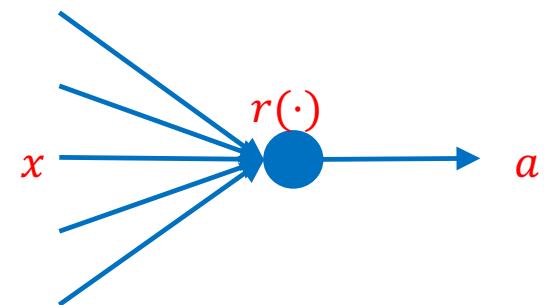
Hidden Layers

- Neuron takes weighted linear combination of the previous representation layer
 - Outputs one value for the next layer



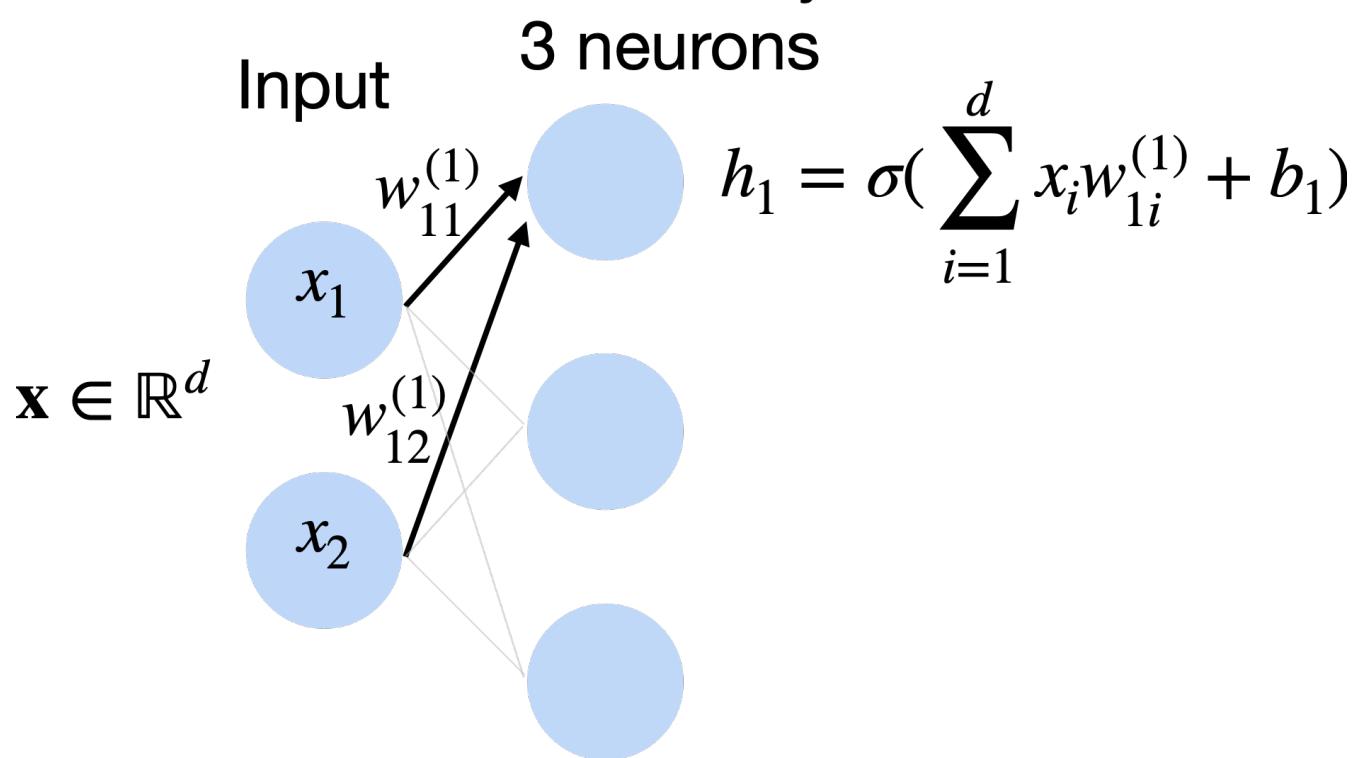
Hidden Layers

- Outputs $a = r(w^T x + b)$
- Typical activation function r
 - Threshold $t(z) = \mathbb{I}[z \geq 0]$
 - Sigmoid $\sigma(z) = 1/(1 + \exp(-z))$
 - Tanh $\tanh(z) = 2\sigma(2z) - 1$
- Why not **linear activation** functions?
 - Model would be linear.



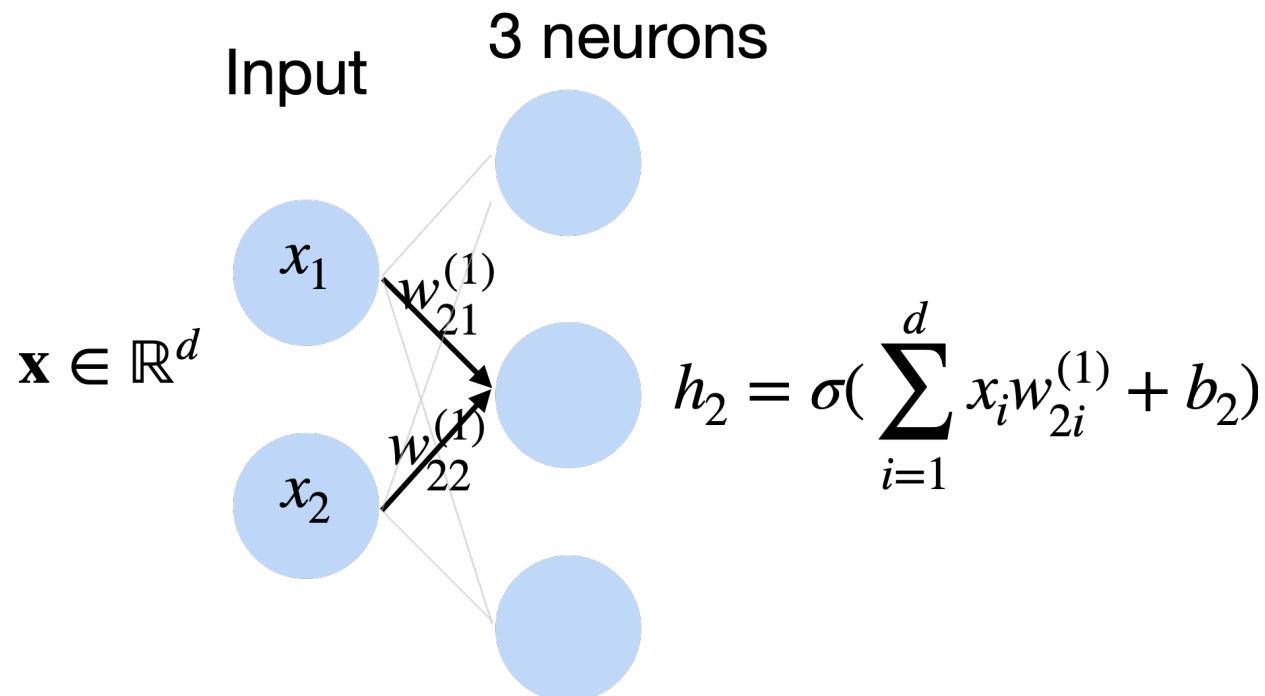
MLPs: Multilayer Perceptron

- Ex: 1 hidden layer, 1 output layer: depth 2



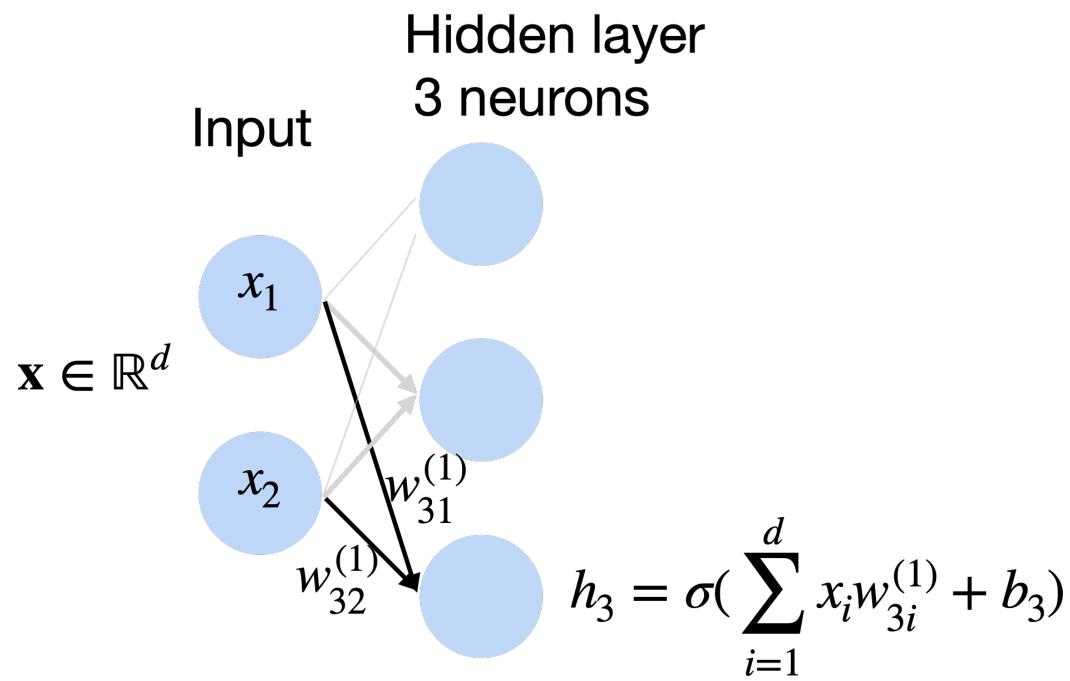
MLPs: Multilayer Perceptron

- Ex: 1 hidden layer, 1 output layer: depth 2



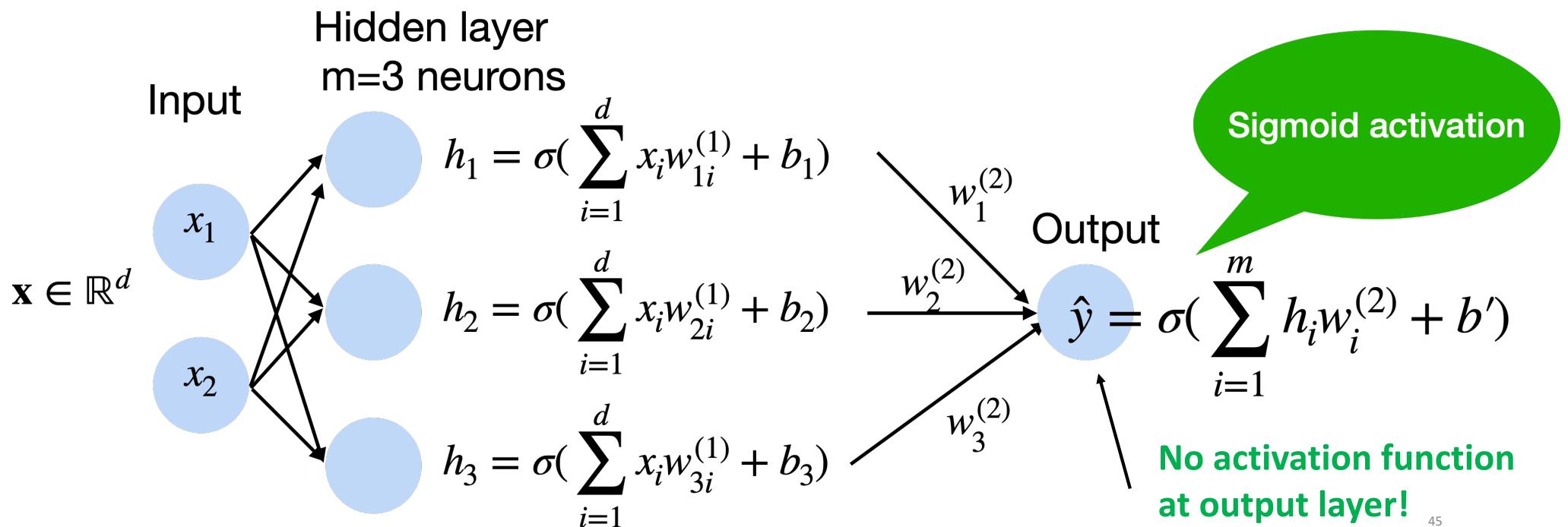
MLPs: Multilayer Perceptron

- Ex: 1 hidden layer, 1 output layer: depth 2



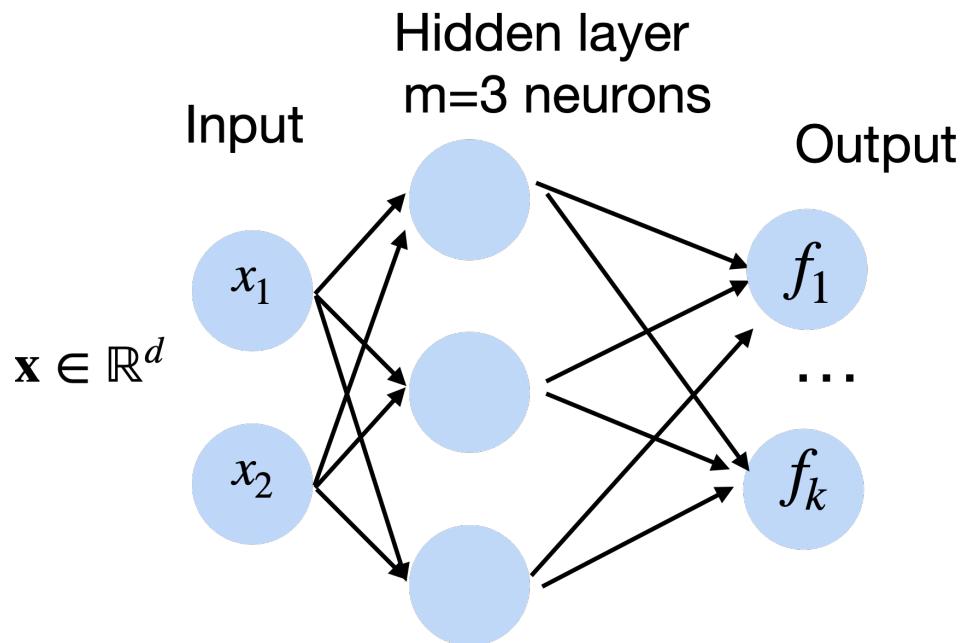
MLPs: Multilayer Perceptron

- Ex: 1 hidden layer, 1 output layer: depth 2



Multiclass Classification Output

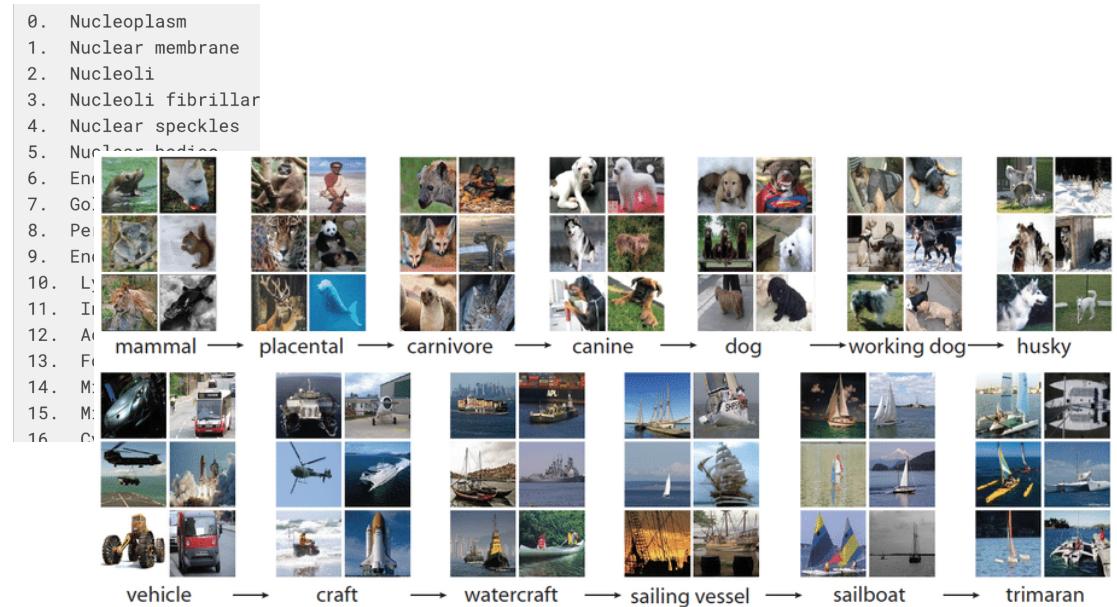
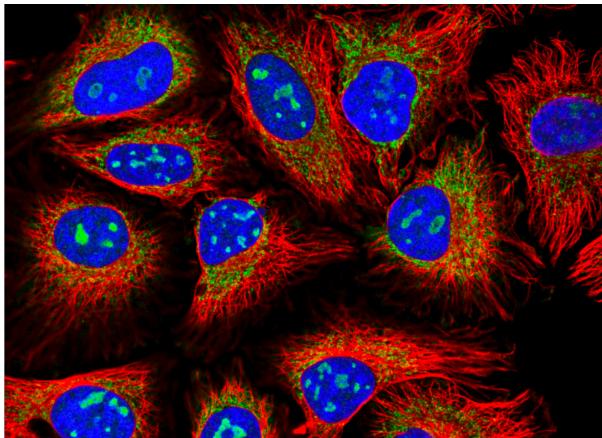
- Create k output units
- Use softmax (just like logistic regression)



$$\begin{aligned} p(y | \mathbf{x}) &= \text{softmax}(f) \\ &= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)} \end{aligned}$$

Multiclass Classification Examples

- Protein classification (Kaggle challenge)
- ImageNet





Break & Quiz

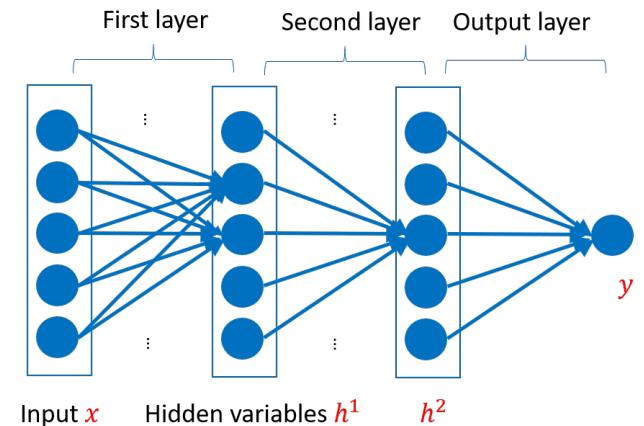
Q2-1: Select the correct option.

- A. *The more hidden-layer units a Neural Network has, the better it can predict desired outputs for new inputs that it was not trained with.*
 - B. *A 3-layers Neural Network with 5 neurons in the input and hidden representations and 1 neuron in the output has a total of 55 connections.*
-
- 1. Both statements are true.
 - 2. Both statements are false.
 - 3. Statement A is true, Statement B is false.
 - 4. Statement B is true, Statement A is false.

Q2-1: Select the correct option.

- A. *The more hidden-layer units a Neural Network has, the better it can predict desired outputs for new inputs that it was not trained with.*
- B. *A 3-layers Neural Network with 5 neurons in the input and hidden representations and 1 neuron in the output has a total of 55 connections.*

- 1. Both statements are true.
- 2. Both statements are false.
- 3. Statement A is true, Statement B is false.
- 4. Statement B is true, Statement A is false.



Outline

- **Review & Perceptron Algorithm**
 - Definition, Training, Loss Equivalent, Mistake Bound
- **Neural Networks**
 - Introduction, Setup, Components, Activations
- **Training Neural Networks**
 - SGD, Computing Gradients, Backpropagation

Training Neural Networks

- Training the usual way. Pick a loss and optimize
- **Example:** 2 scalar weights

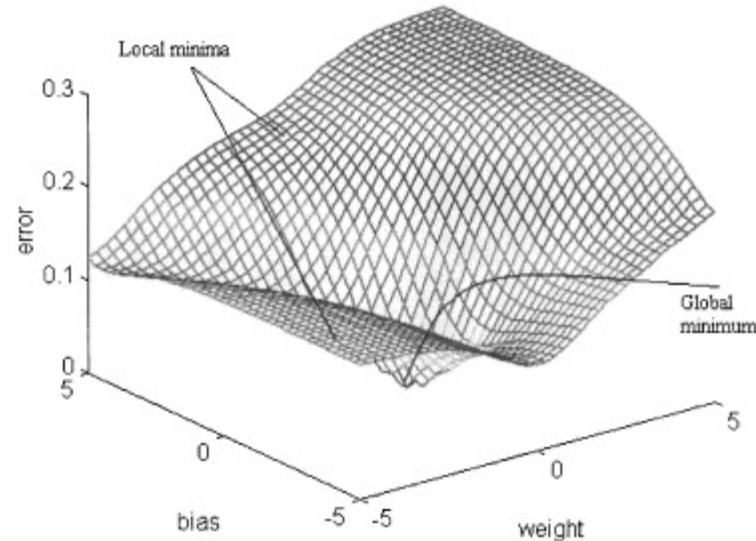
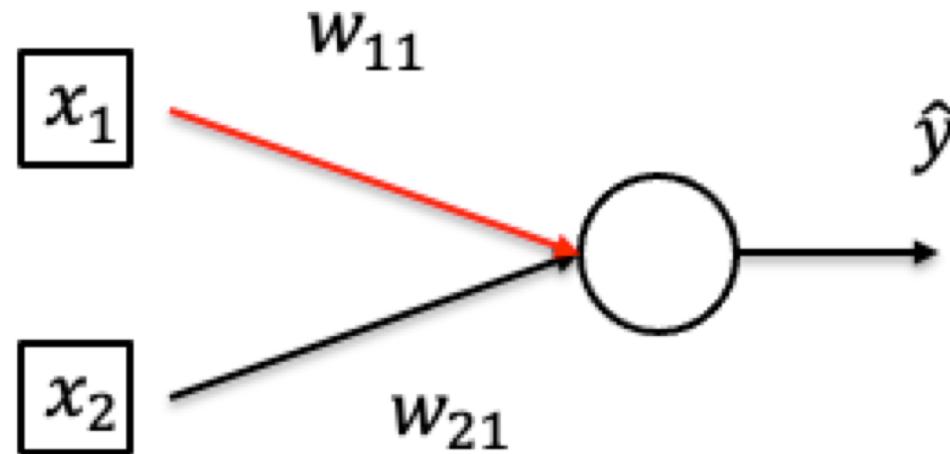


figure from Cho & Chow, *Neurocomputing* 1999

Training Neural Networks

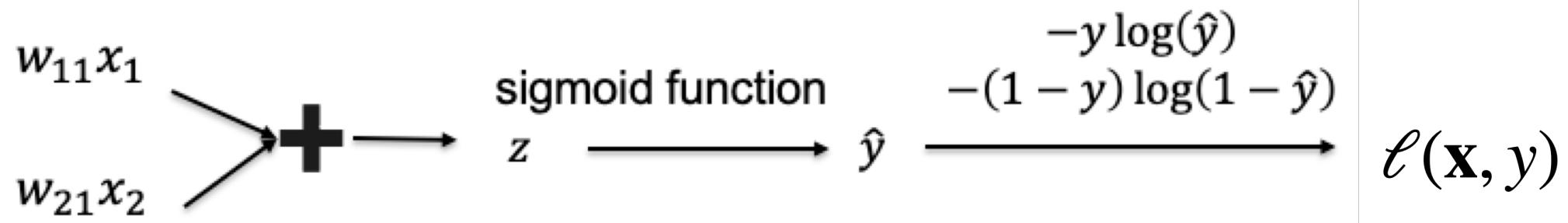
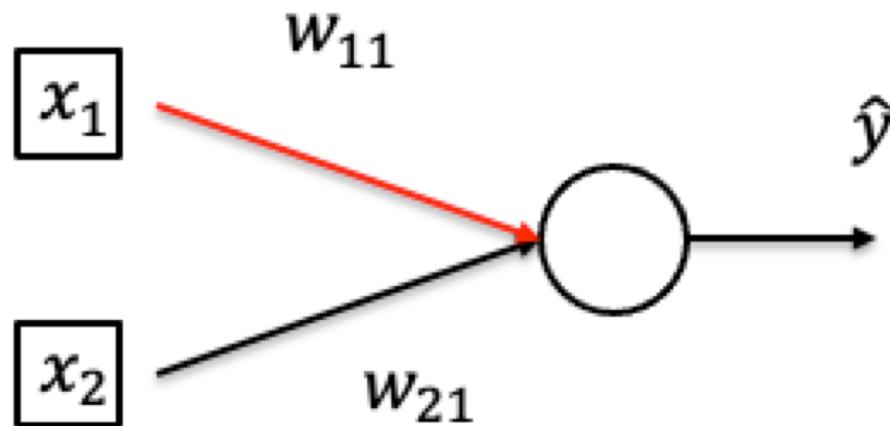
- Algorithm:
 - Get $D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$
 - Initialize weights
 - Until stopping criteria met,
 - For each training point $(x^{(i)}, y^{(i)})$
 - Compute: $f_{\text{network}}(x^{(d)})$ ← **Forward Pass**
 - Compute gradient: $\nabla L^{(i)}(w) = \left[\frac{\partial L^{(d)}}{\partial w_0}, \frac{\partial L^{(d)}}{\partial w_1}, \dots, \frac{\partial L^{(d)}}{\partial w_m} \right]^T$ ← **Backward Pass**
 - Update weights: $w \leftarrow w - \alpha \nabla L^{(i)}(w)$

Computing Gradients

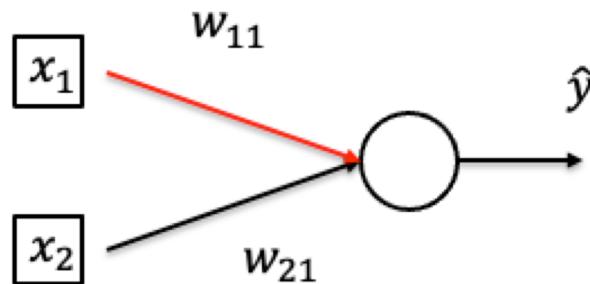


- Want to compute $\frac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

Computing Gradients



Computing Gradients

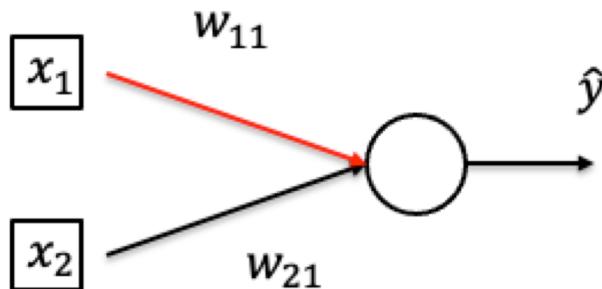


$w_{11}x_1$	$+ w_{21}x_2$	$\rightarrow z$	$\xrightarrow{\text{sigmoid function}}$	\hat{y}	$\xrightarrow{-y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})}$	$\ell(\mathbf{x}, y)$
				$\frac{\partial \hat{y}}{\partial z} = \sigma'(z)$	$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}}$	

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}}$$

Computing Gradients

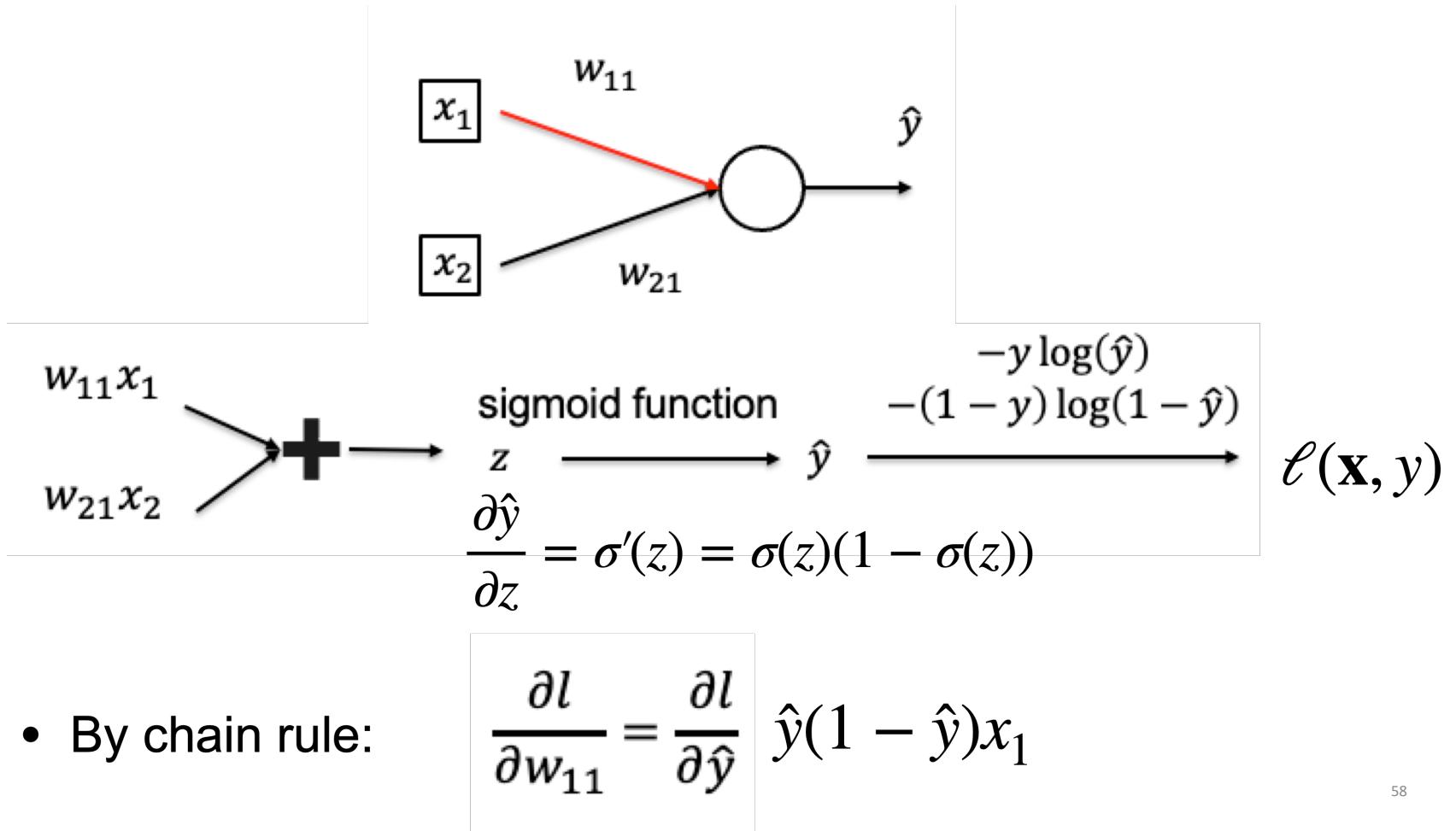


$$\begin{array}{ccccc} w_{11}x_1 & \xrightarrow{\quad\quad\quad} & z & \xrightarrow{\text{sigmoid function}} & \hat{y} \\ w_{21}x_2 & \xrightarrow{+} & & & \xrightarrow{-y \log(\hat{y})} \\ & & & & \xrightarrow{-(1-y) \log(1-\hat{y})} \\ & & & & \ell(\mathbf{x}, y) \\ \frac{\partial \hat{y}}{\partial z} = \sigma'(z) & & & & \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \end{array}$$

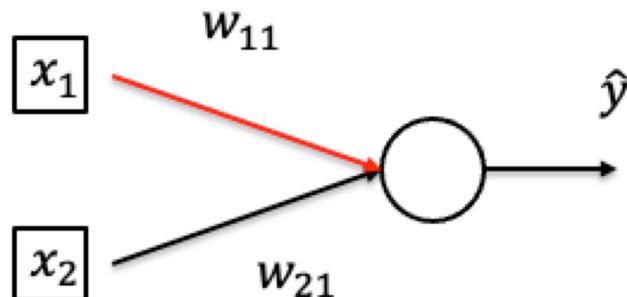
- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} |_{x_1}$$

Computing Gradients



Computing Gradients

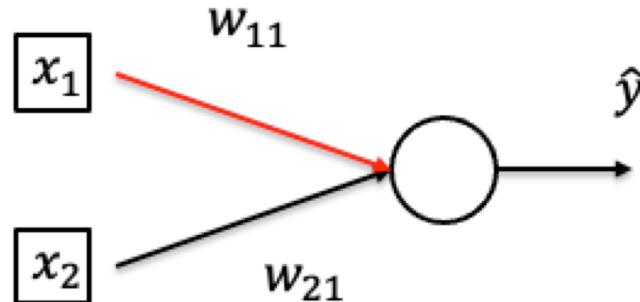


$$\begin{array}{c} w_{11}x_1 \\ w_{21}x_2 \end{array} \rightarrow \text{+} \rightarrow z \xrightarrow{\text{sigmoid function}} \hat{y} \xrightarrow{-y \log(\hat{y}) - (1 - \hat{y}) \log(1 - \hat{y})} \ell(\mathbf{x}, y)$$
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \left(\frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}} \right) \hat{y}(1 - \hat{y})x_1$$

Computing Gradients

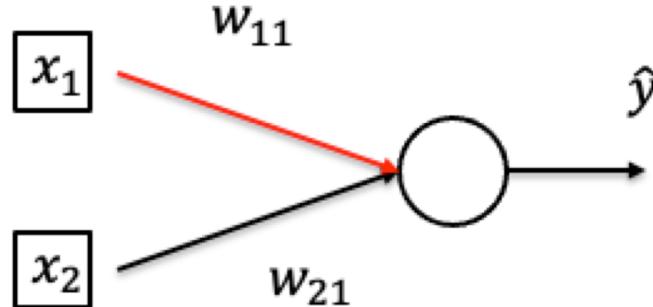


$$\begin{array}{c} w_{11}x_1 \\ w_{21}x_2 \end{array} \rightarrow \text{+} \rightarrow z \rightarrow \text{sigmoid function} \rightarrow \hat{y} \rightarrow \frac{-y \log(\hat{y})}{-(1 - y) \log(1 - \hat{y})} \rightarrow \ell(\mathbf{x}, y)$$
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = (\hat{y} - y)x_1$$

Computing Gradients

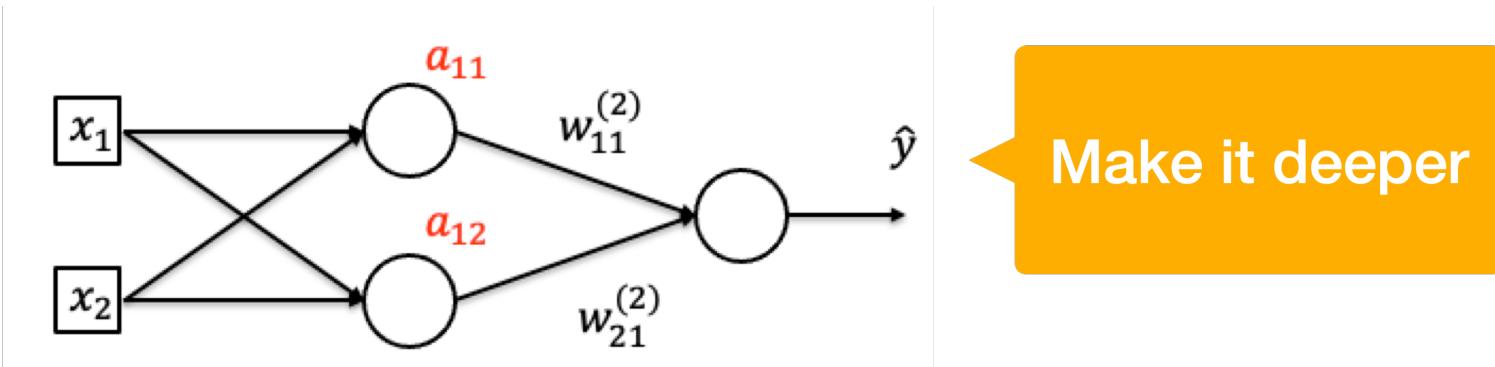


$$\begin{array}{c} w_{11}x_1 \\ w_{21}x_2 \end{array} \rightarrow \text{+} \rightarrow z \xrightarrow{\text{sigmoid function}} \hat{y} \xrightarrow{-y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})} \ell(\mathbf{x}, y)$$
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- By chain rule:

$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} w_{11} = (\hat{y} - y)w_{11}$$

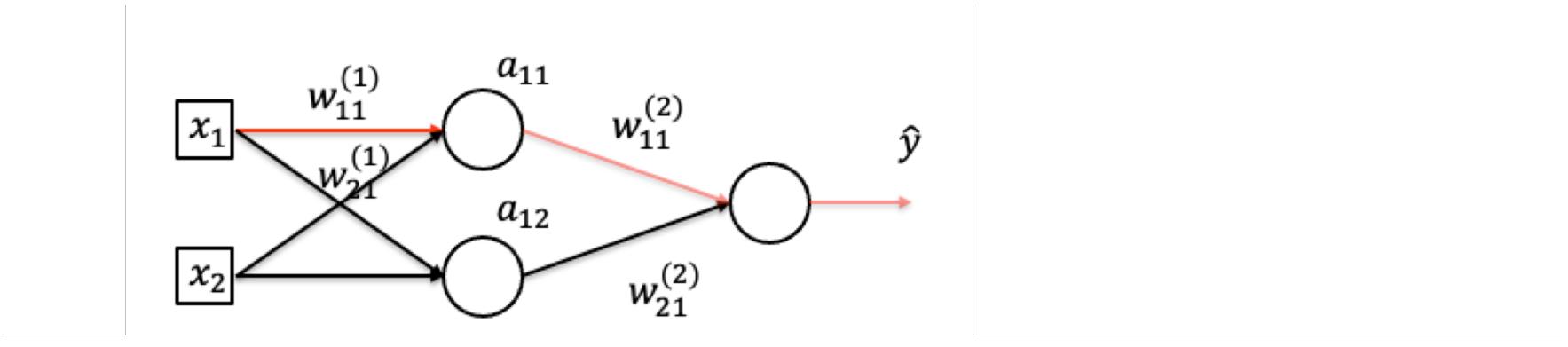
Computing Gradients: More Layers



$$\begin{array}{c} w_{11}^{(2)} \color{red}{a_{11}} \\ w_{21}^{(2)} \color{red}{a_{12}} \end{array} \rightarrow \text{+} \rightarrow \begin{array}{c} \text{sigmoid function} \\ z \longrightarrow \hat{y} \\ \frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z)) \end{array} \rightarrow \begin{array}{c} -y \log(\hat{y}) \\ -(1 - y) \log(1 - \hat{y}) \\ \ell(\mathbf{x}, y) \end{array}$$

- By chain rule: $\frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}, \frac{\partial l}{\partial a_{12}} = (\hat{y} - y)w_{21}^{(2)}$

Computing Gradients: More Layers

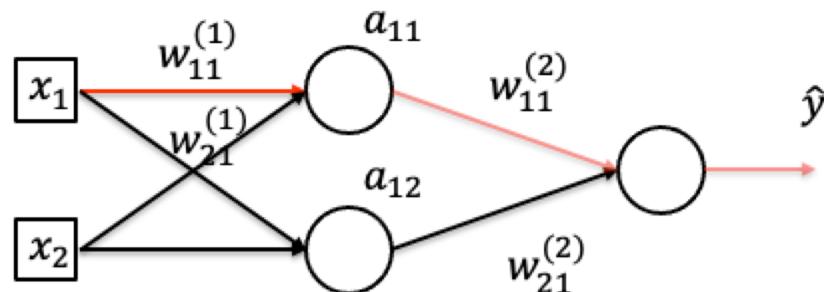


$$\begin{aligned} & w_{11}^{(1)}x_1 \\ & w_{21}^{(1)}x_2 \end{aligned} \rightarrow \text{+} \rightarrow z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \rightarrow l(x, y)$$
$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \quad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)} \frac{\partial a_{11}}{\partial w_{11}^{(1)}}$$

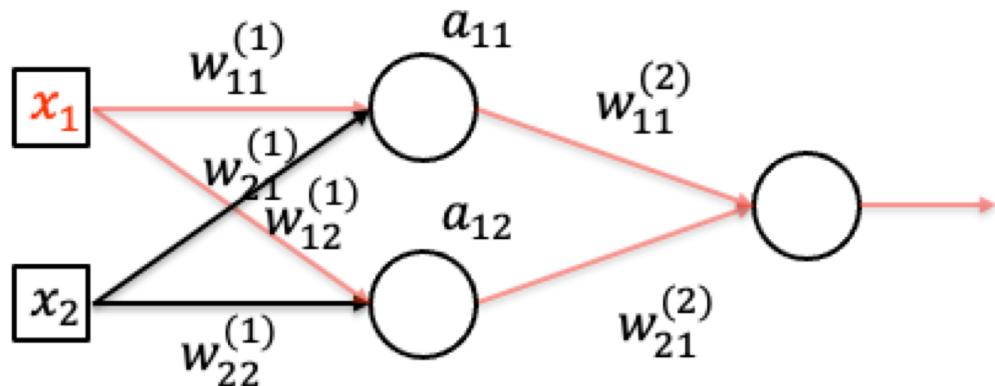
Computing Gradients: More Layers



$w_{11}^{(1)} x_1$ $w_{21}^{(1)} x_2$ $\sigma(z_{11})$ a_{11} $l(x, y)$
 $\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11})$ $\frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$

- By chain rule: $\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)}a_{11}(1 - a_{11})x_1$

Computing Gradients: More Layers



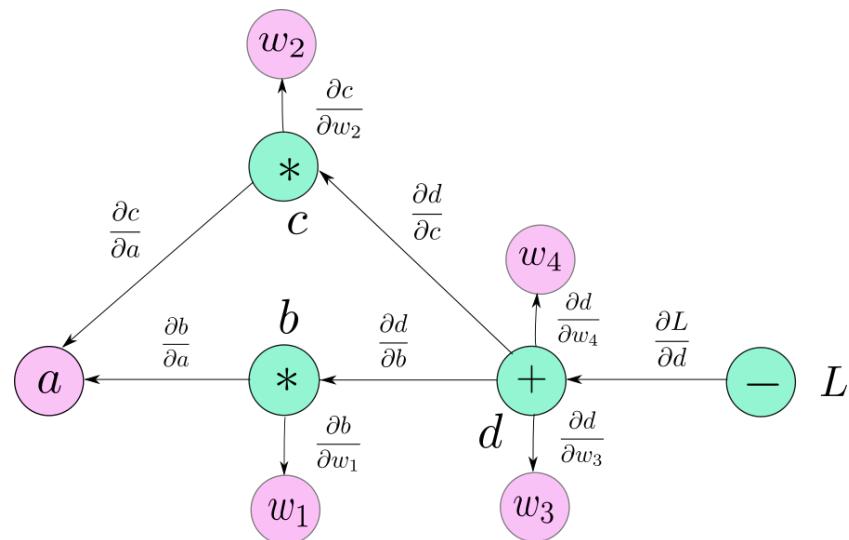
$$\begin{aligned}
 & w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 \rightarrow z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \xrightarrow{} l(x, y) \\
 & \frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \quad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}
 \end{aligned}$$

- By chain rule:

$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}} \frac{\partial a_{12}}{\partial x_1}$$

Backpropagation

- Now we can compute derivatives for particular neurons, but we want to automate this process
- Set up a computation graph and run on the graph





Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, Sharon Li, Fred Sala