CS 760: Machine Learning
Neural Networks

Kirthi Kandasamy

University of Wisconsin-Madison

February 22, 2023

Outline

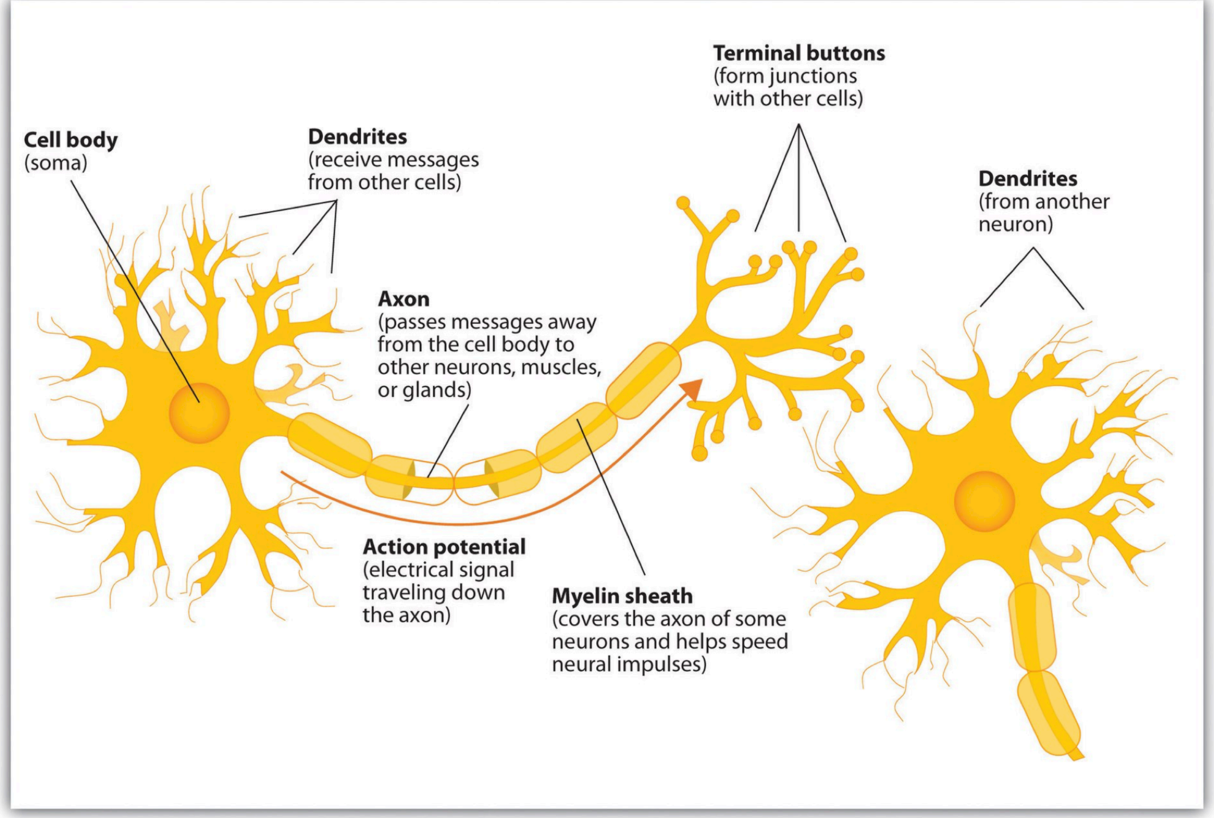
- **Perceptron Algorithm**
  - Definition, Training, Loss Equivalent, Mistake Bound
- **Neural Networks**
  - Introduction, Setup, Components, Activations
- **Training Neural Networks**
  - SGD, Computing Gradients, Backpropagation

Outline

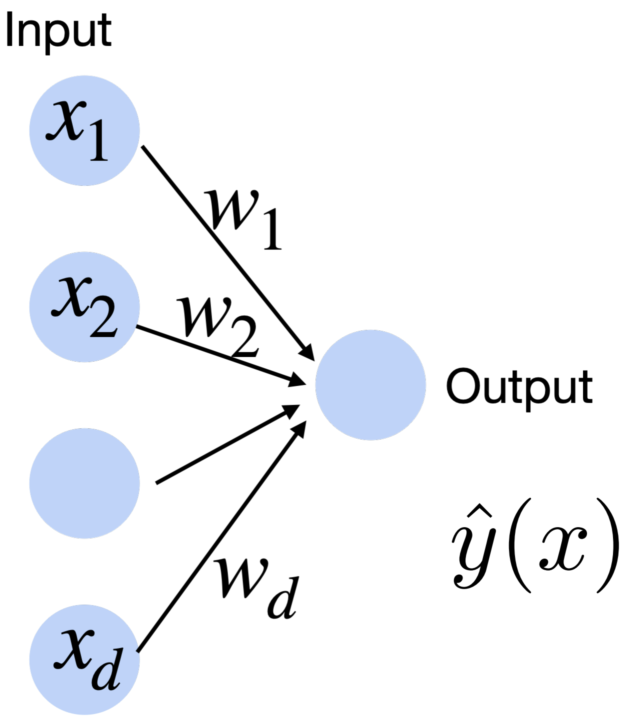
- **Perceptron Algorithm**
  - Definition, Training, Loss Equivalent, Mistake Bound
- **Neural Networks**
  - Introduction, Setup, Components, Activations
- **Training Neural Networks**
  - SGD, Computing Gradients, Backpropagation

Neural networks: Origins

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



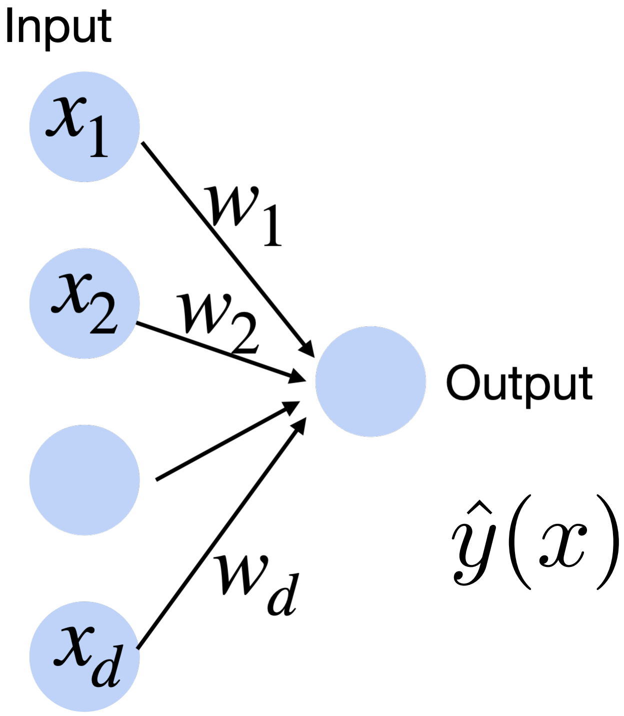
Perceptron: Simple Network



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

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

Perceptron: Components



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

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

Activation Function

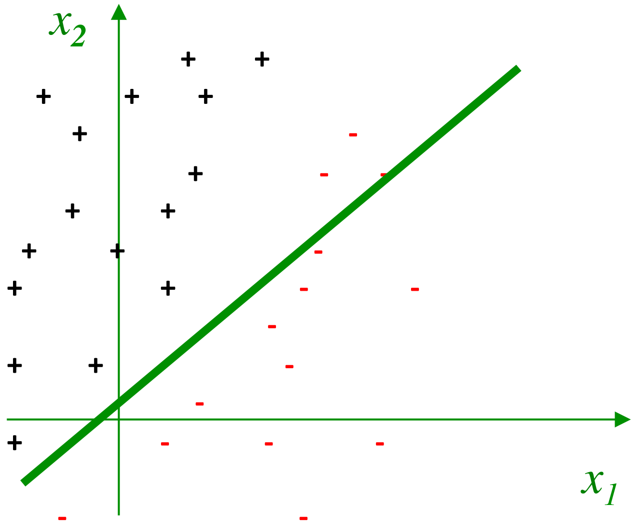
[McCulloch & Pitts, **1943**; Rosenblatt, **1959**; Widrow & Hoff, **1960**]

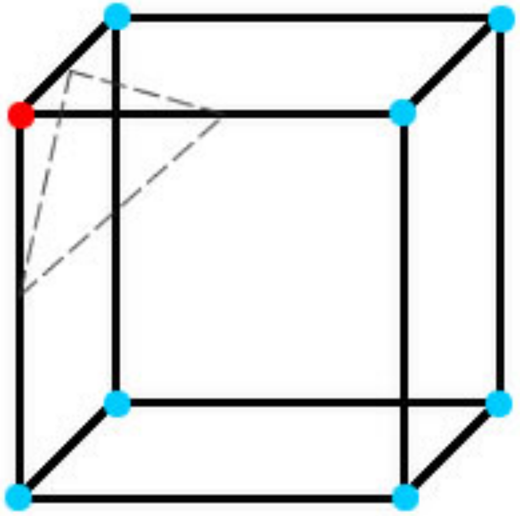
Perceptron: Representational Power

- 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:

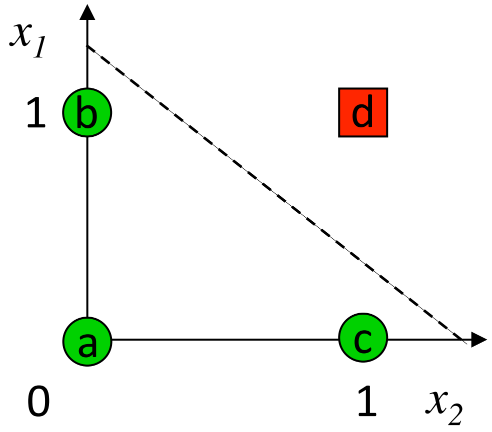




Which Functions are **Linearly Separable?**

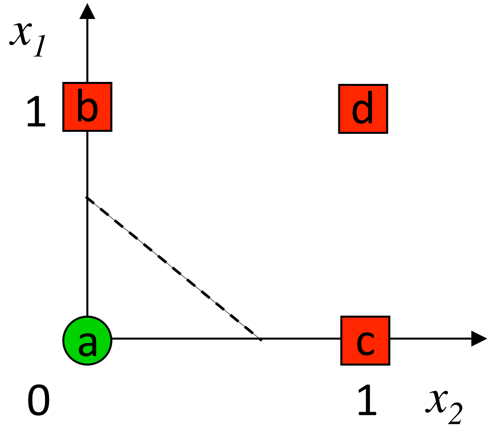
AND

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



OR

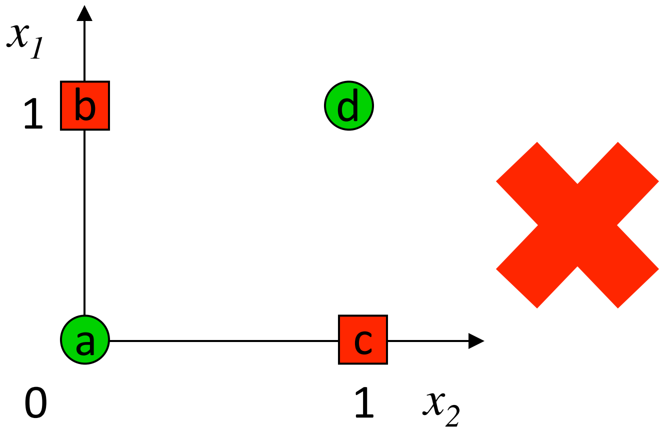
	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	1



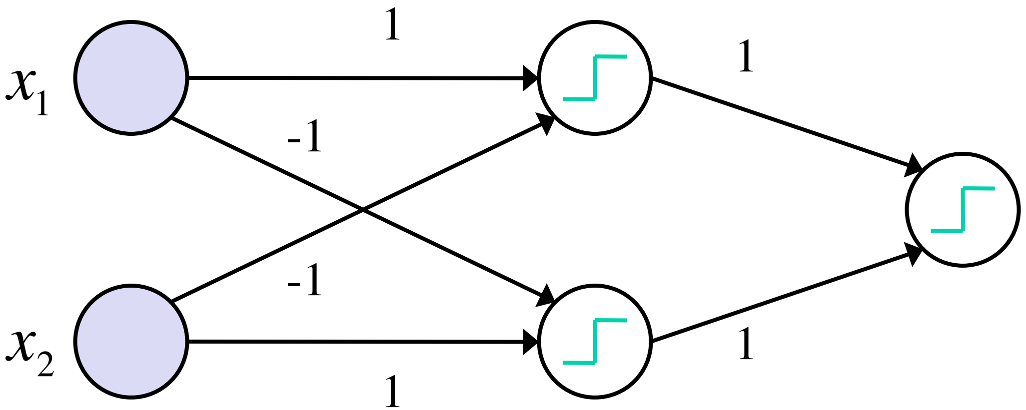
Which Functions are **Linearly Separable?**

XOR

	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!



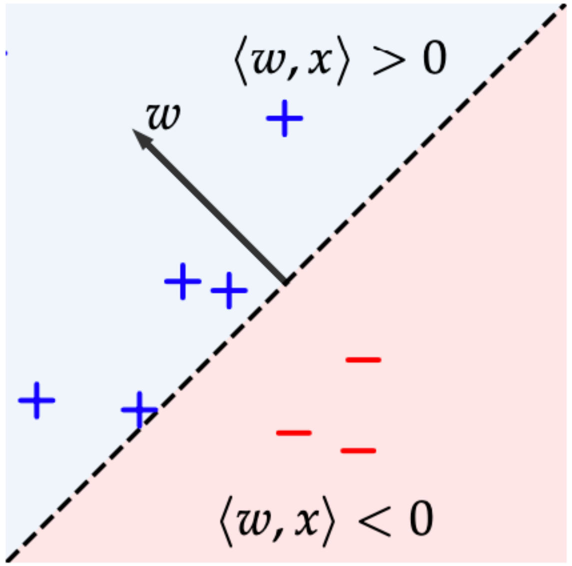
assume $w_0 = 0$ for all nodes

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

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

  - A little more than what we really need

Perceptron: Training

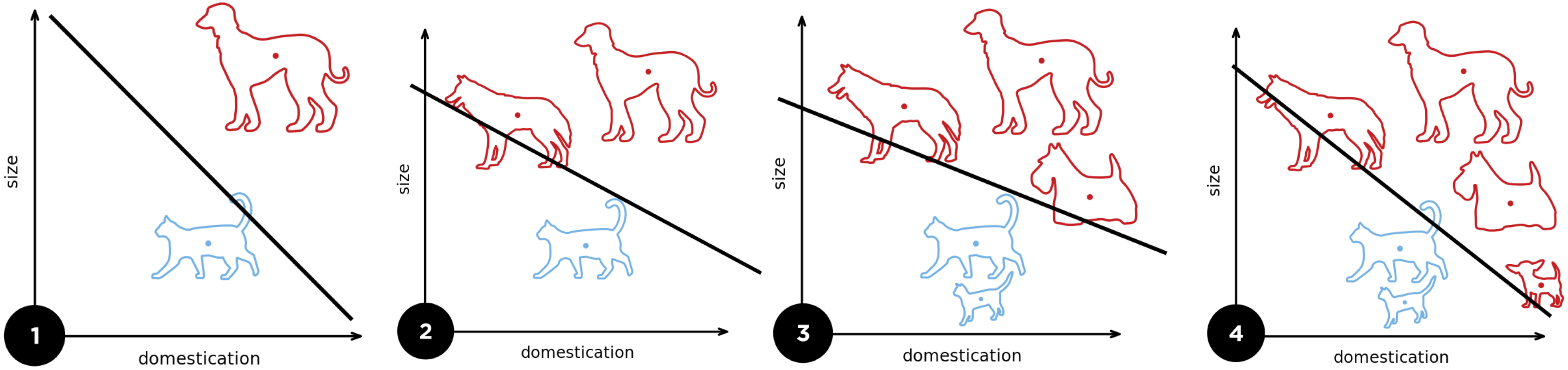
Going forward assume labels are +1 or -1. $y^{(i)} \leftarrow 2y^{(i)} - 1$

- **Algorithm**:
  - Initialize $w_0 = 0$.
  - At step t = 0,…
  - Select index i,
  - If $y^{(i)} w^T x^{(i)} < 1$ then do $w_{t+1} = w_t + y^{(i)} x^{(i)}$
  - Else, $w_{t+1} = w_t$
- What is the update to our prediction?

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

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)

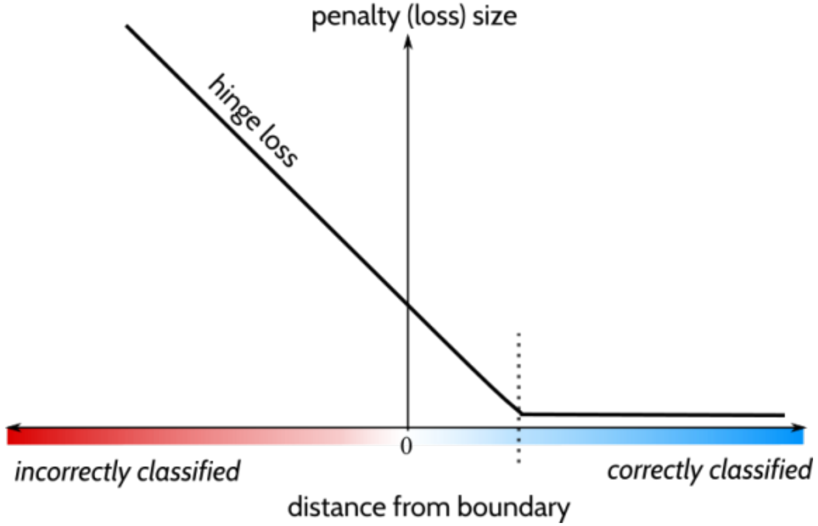
- **Step**: $w_{t+1} = w_t + y^{(i)} x^{(i)}$

Perceptron: Training Comparison

- So: Does this look like **SGD** with some loss function L?

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

Perceptron $$w_{t+1} = w_t + y^{(i)} x^{(i)}$$ (if there is an error)



Hinge loss!

Perceptron: Analysis

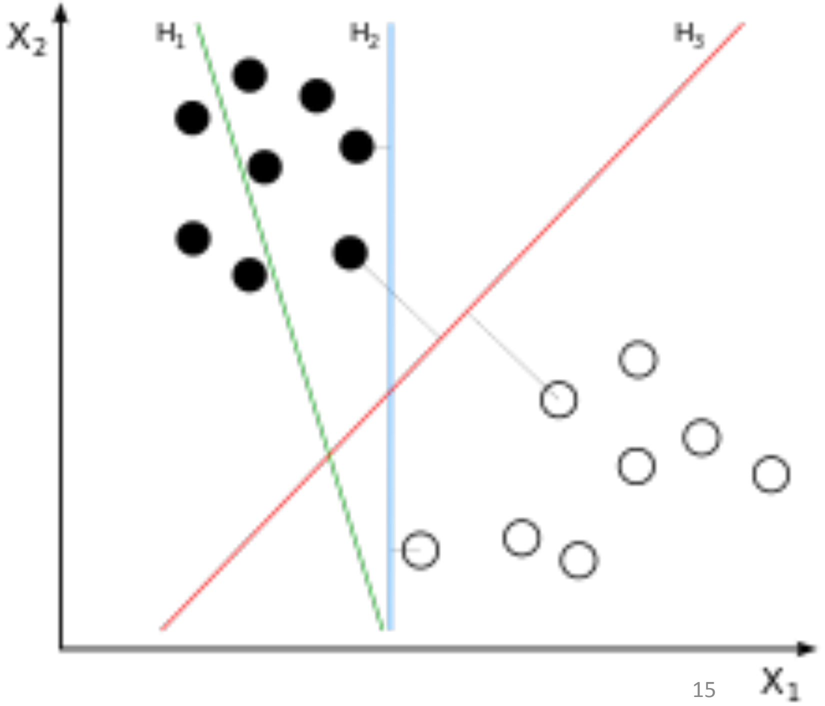
- Two aspects to analysis: **fitting training data** + **generalization**
- **Mistake bound**:
  - Hyperplane $H_w = x : w^T x = 0$
  - Margin (for a dataset S)

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

$$\downarrow$$

$$|x^T w| / \|w\|$$

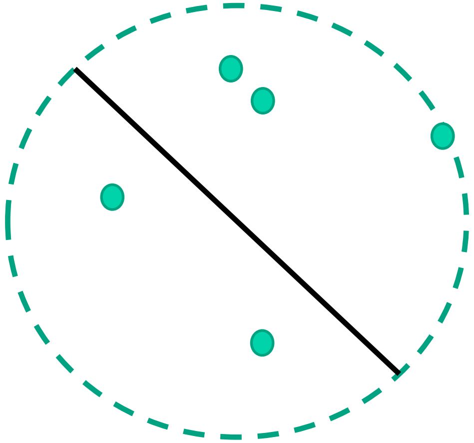
$$\gamma(S) = \max_{\|w\|=1} \gamma(S, w)$$



Perceptron: Mistake Bound

- Need some information about our data:

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



- **Mistake Bound Result**:
  - The total # of mistakes on a linearly separable set S is at most

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

  - How is this result different from our SGD result?

Perceptron: Mistake Bound Interpretation

- **Mistake Bound Result**:
  - The total # of mistakes on a linearly separable set S is at most

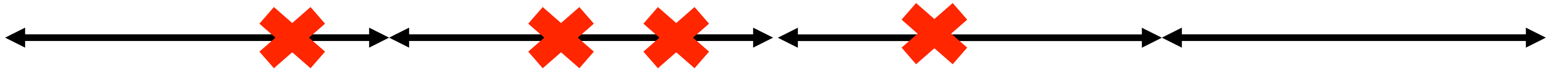
$$(2 + D(S)^2)\gamma(S)^{-2}$$

Diameter: Controls our biggest step.

Margin: Smaller means harder to find separator

- Scaling?

- **Implications**?
  - Run over dataset D repeatedly. # mistakes doesn't change
    - If we keep running it, eventually we get perfect separation on a copy of D



Mistake Bound: Proof 1

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

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

If mistake

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

Margin

Diameter

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

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

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

Mistake Bound: Proof 3

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

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

w classifies correctly ← $|w^T x^{(i_t)}|$

Norm 1 ← $\|w\|$

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

- Let's look at our best unit norm solution: w_*, i.e one with the maximum margin w
- From Cauchy-Schwartz $\|w_t\|\|w_*\| \geq w_*^T w_t$

- Let's set up a telescoping sum:

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

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

- I.e.,

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

- Easy algebra gets us to

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

✓

- Result holds for any t!

Break & Quiz

Q1-1: Select the correct option.

A. *A perceptron is guaranteed to perfectly learn a given linearly separable dataset 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 separable dataset within a finite number of training steps.*

B. *A single perceptron can compute the XOR function.*

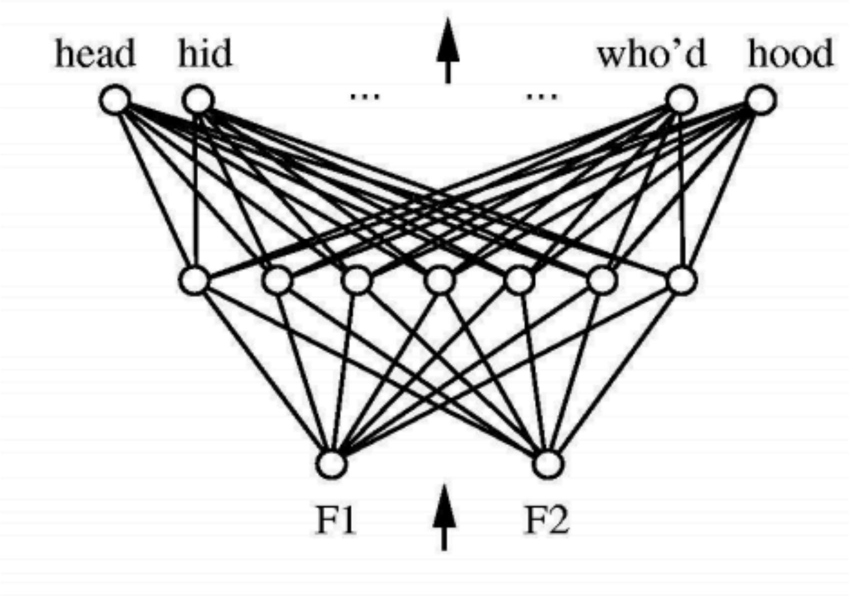
1. Both statements are true.
2. [✓ Both statements are false.]
3. [✓ Statement A is true, Statement B is false.]
4. Statement B is true, Statement A is false.

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"



output units

hidden units

input units

figure from Huang & Lippmann, *NIPS* 1988

Neural Network **Decision Regions**

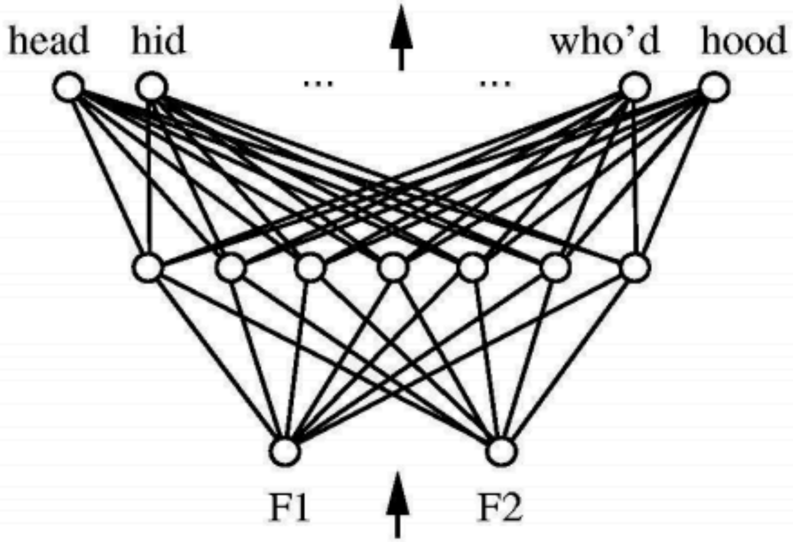
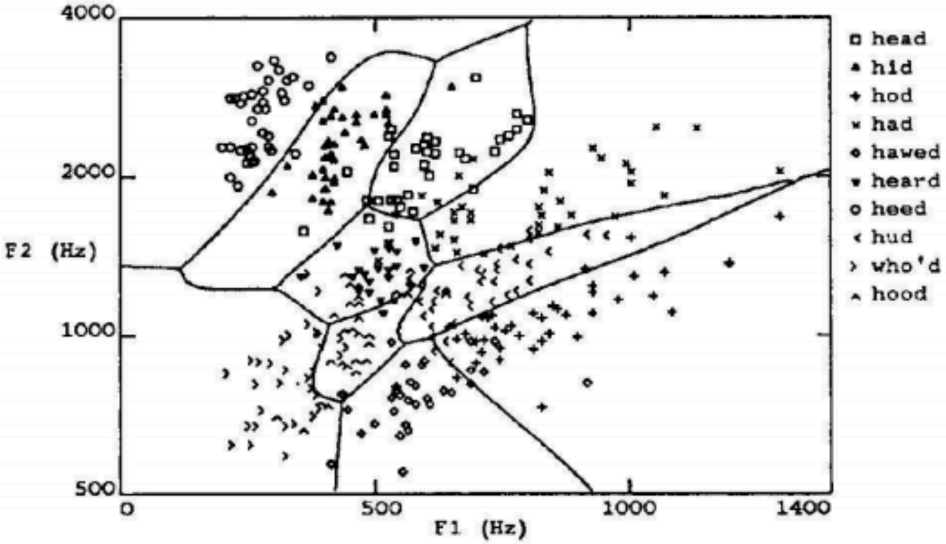
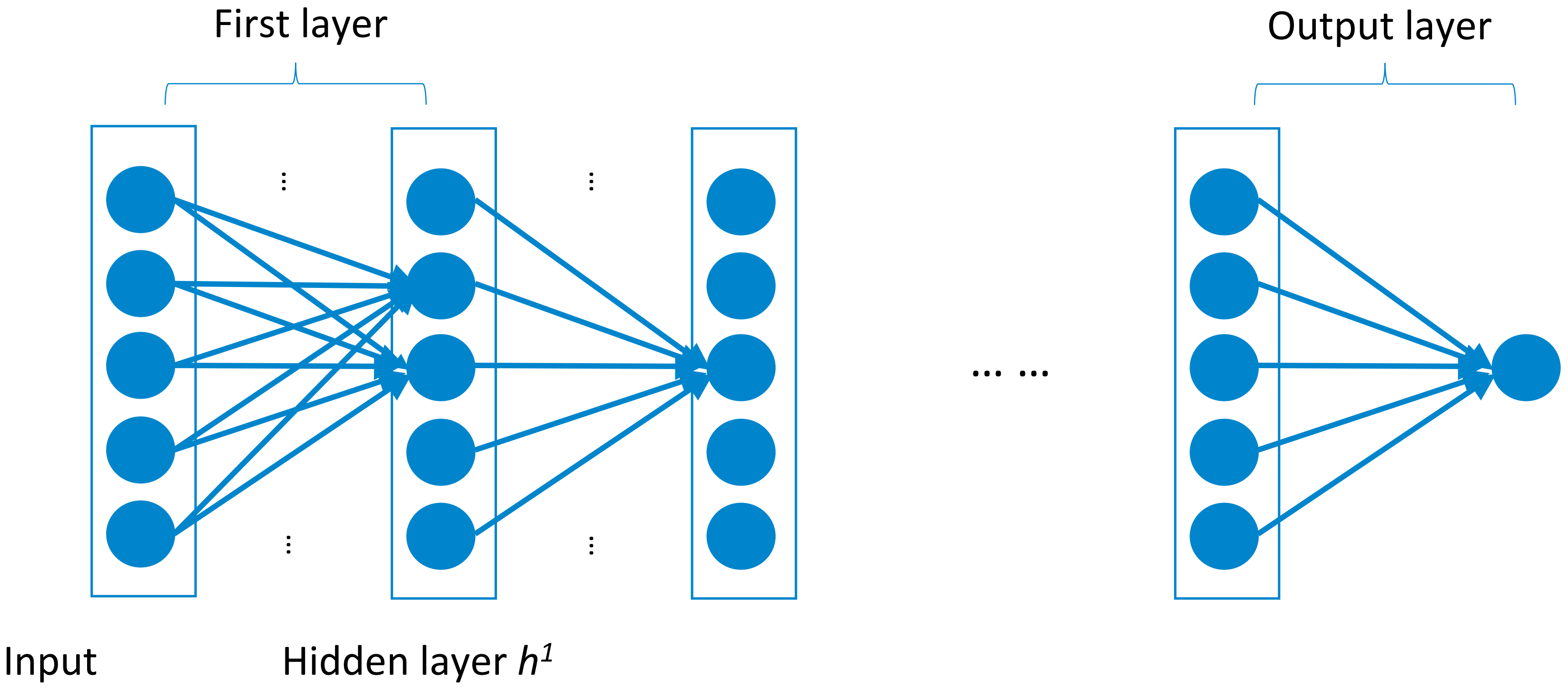


Figure from Huang & Lippmann, *NeurIPS* 1988



Neural Network Components

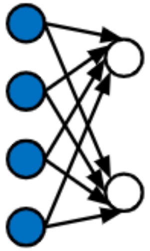
An $(L+1)$-layer network



Feature Encoding for NNs

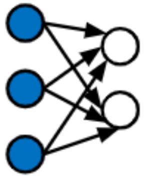
- Nominal features usually a one hot encoding

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



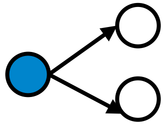
- Ordinal features: use a *thermometer* encoding

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



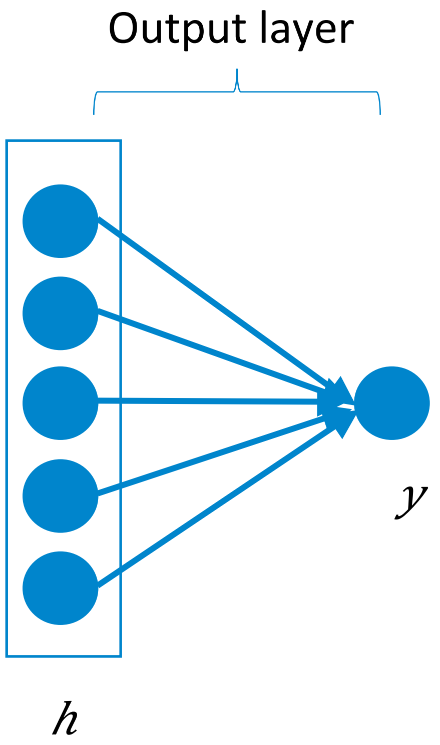
- Real-valued features use individual input units (may want to scale/normalize them first though)

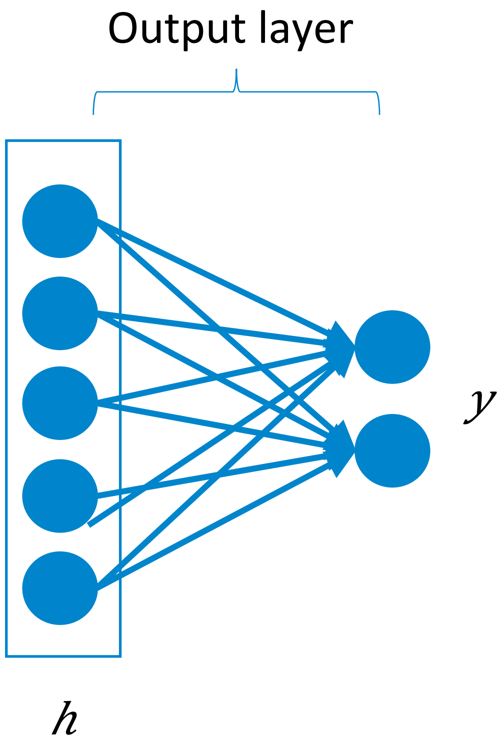
$$\text{precipitation} = \begin{bmatrix}0.68\end{bmatrix}$$



Output Layer: Examples

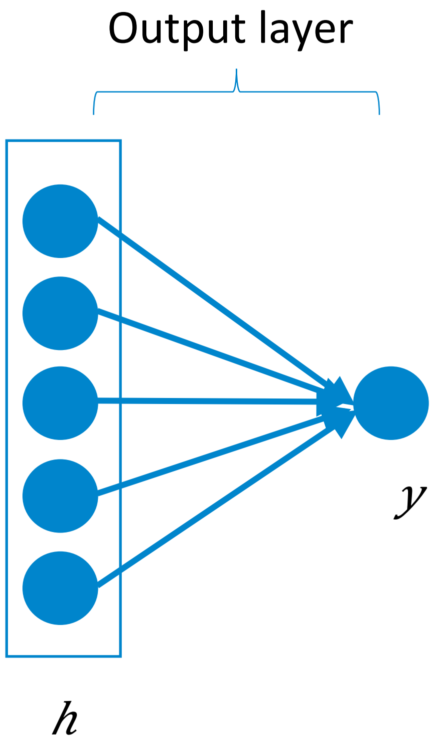
- Regression:
  - Linear units: no nonlinearity
- Multi-dimensional regression:
  - Linear units: no nonlinearity

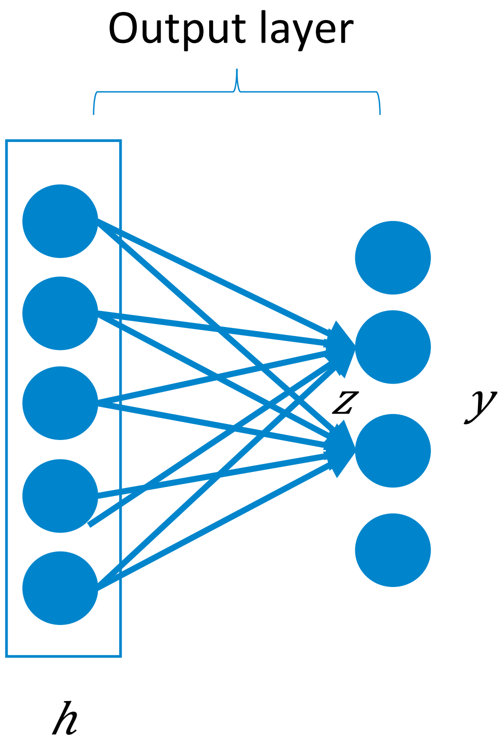




Output Layer: Examples

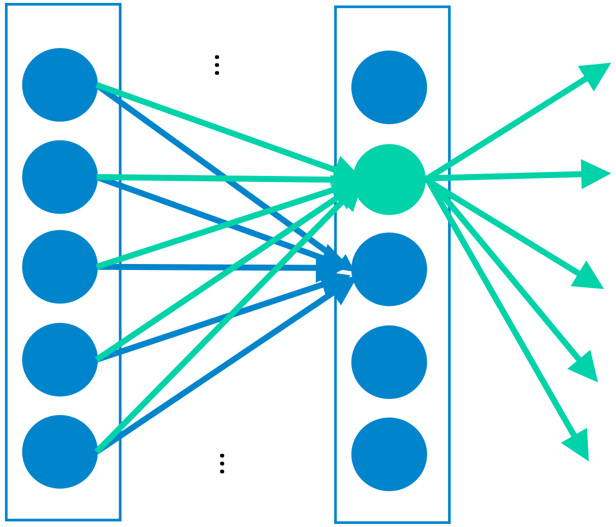
- Binary classification:
  - Corresponds to using logistic regression on
- Multiclass classification:
  - where





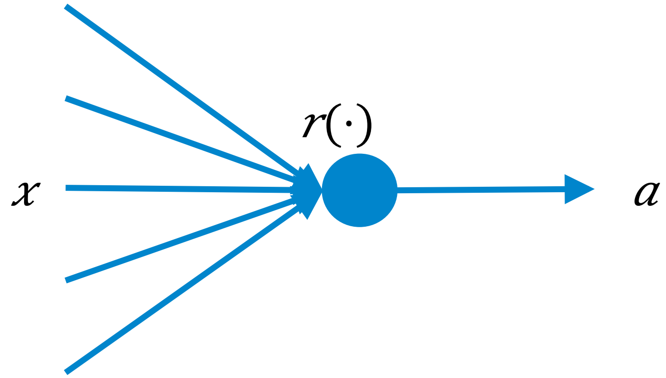
Hidden Layers

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



Hidden Layers

- Outputs

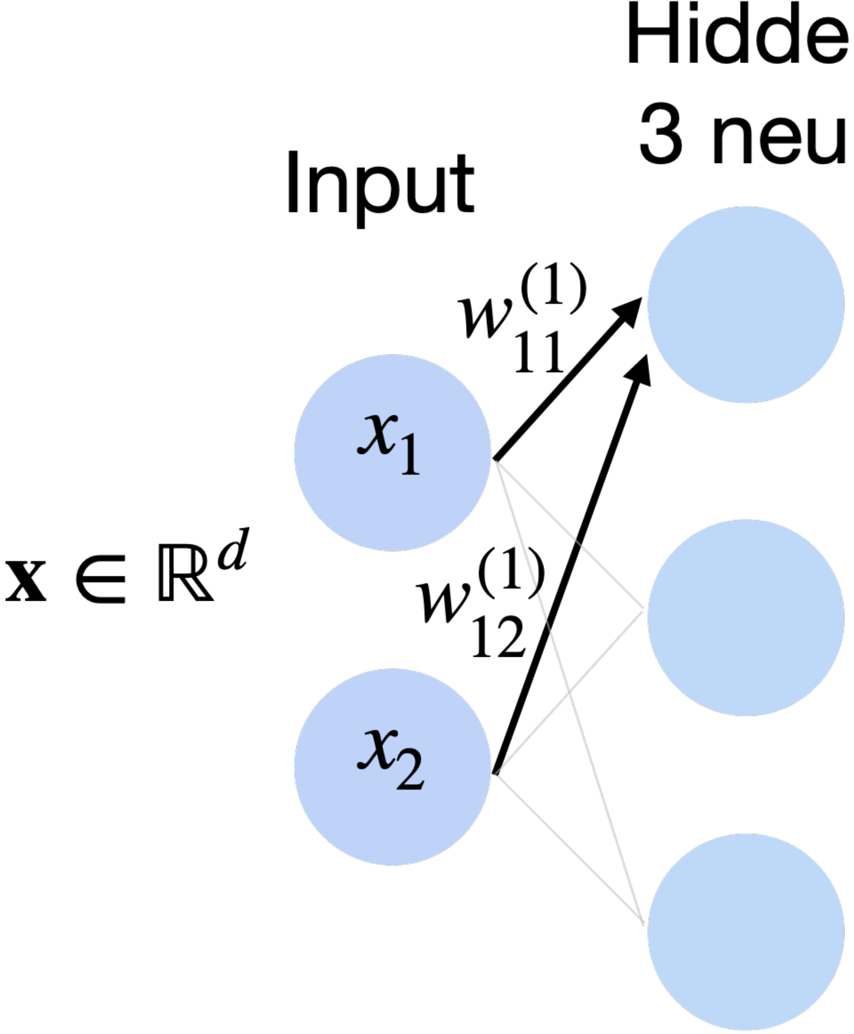


- Typical activation function
  - Threshold:
  - Sigmoid:
  - Tanh:

- Why not **linear activation** functions?
  - Model would be linear.

MLPs: Multilayer Perceptron

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



Hidden layer

3 neurons

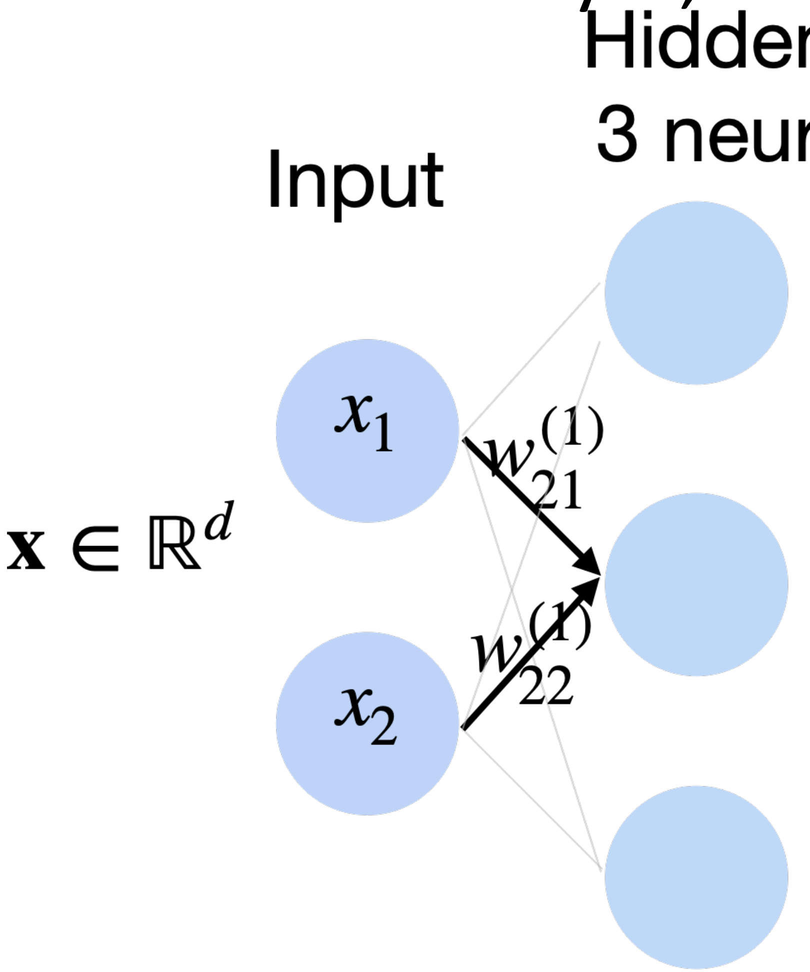
Input

$\mathbf{x} \in \mathbb{R}^d$

$$h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1)$$

MLPs: Multilayer Perceptron

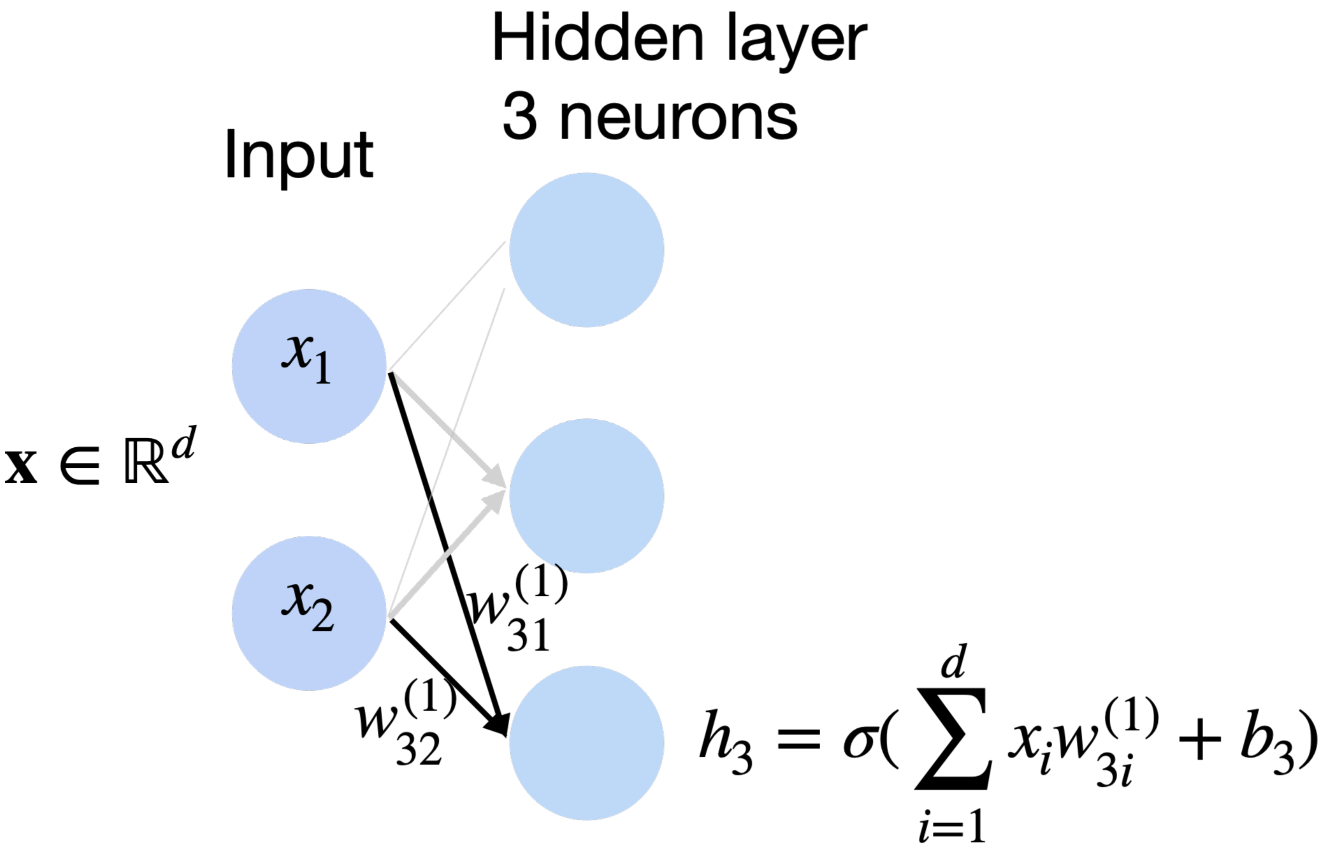
- **Ex**: 1 hidden layer, 1 output layer: depth 2



$$h_2 = \sigma(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2)$$

MLPs: Multilayer Perceptron

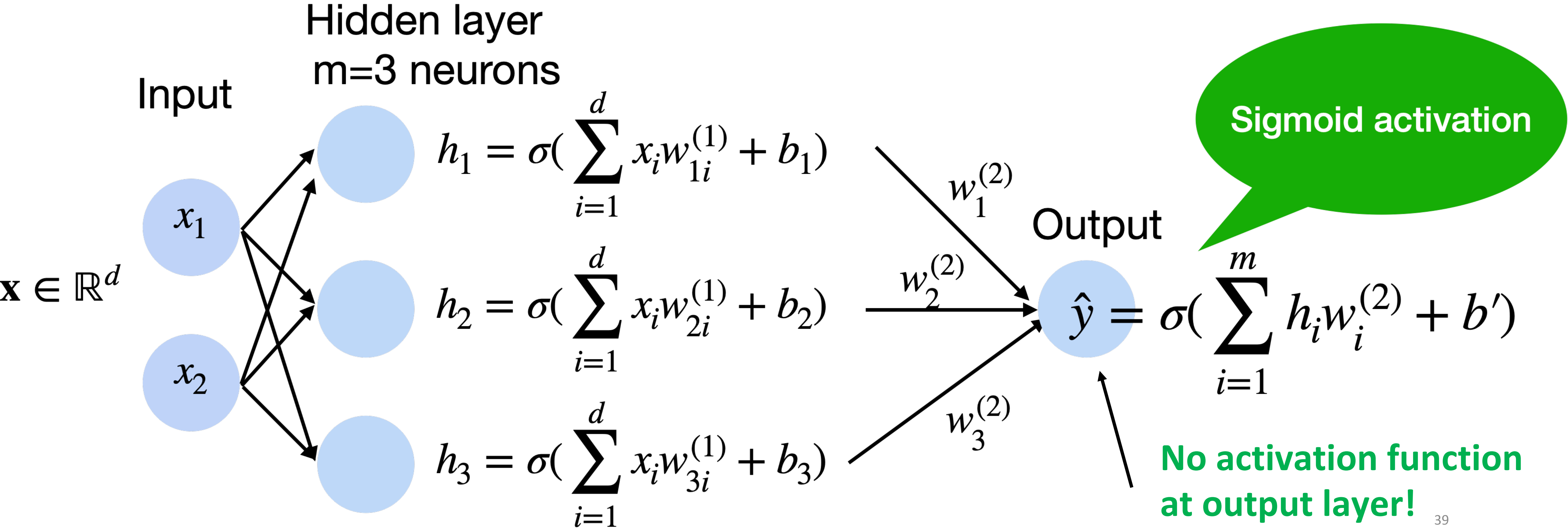
- **Ex**: 1 hidden layer, 1 output layer: depth 2



$$h_3 = \sigma(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3)$$

MLPs: Multilayer Perceptron

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



$$h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1)$$

$$h_2 = \sigma(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2)$$

$$h_3 = \sigma(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3)$$

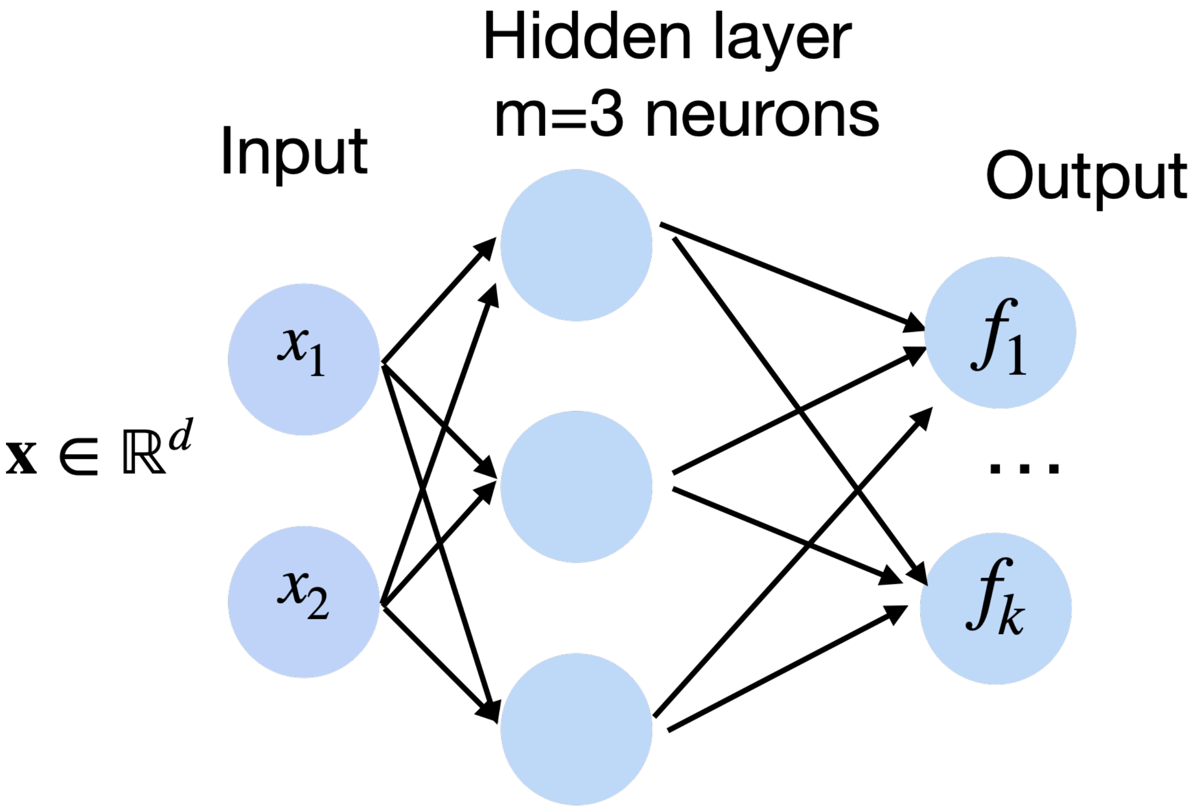
$$\hat{y} = \sigma(\sum_{i=1}^{m} h_i w_i^{(2)} + b')$$

Sigmoid activation

No activation function at output layer!

Multiclass Classification Output

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

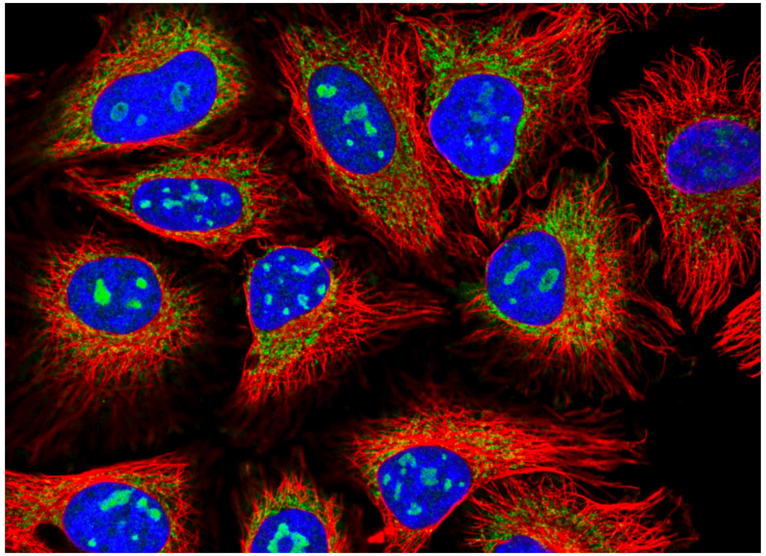


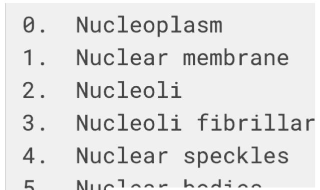
$$p(y \mid \mathbf{x}) = \text{softmax}(f)$$

$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

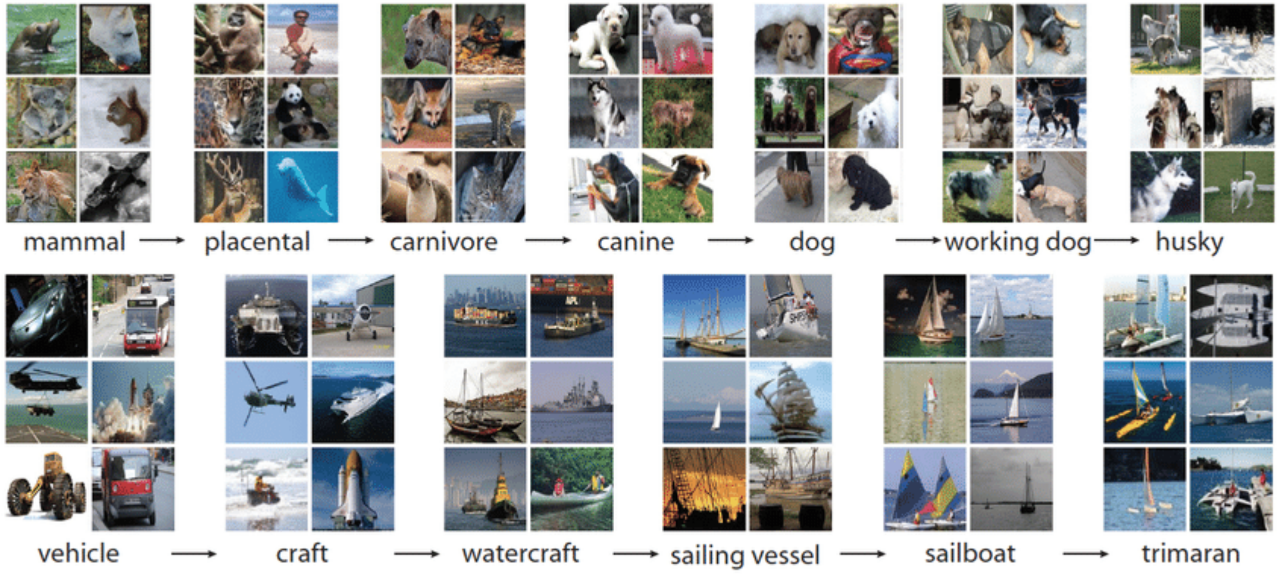
Multiclass Classification Examples

- Protein classification (Kaggle challenge)
- ImageNet





0. Nucleoplasm
1. Nuclear membrane
2. Nucleoli
3. Nucleoli fibrillar
4. Nuclear speckles
5. Nuclear bodies
6. En
7. Gol
8. Pe
9. En
10. L
11. I
12. A
13. F
14. M
15. M
16. C



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.*

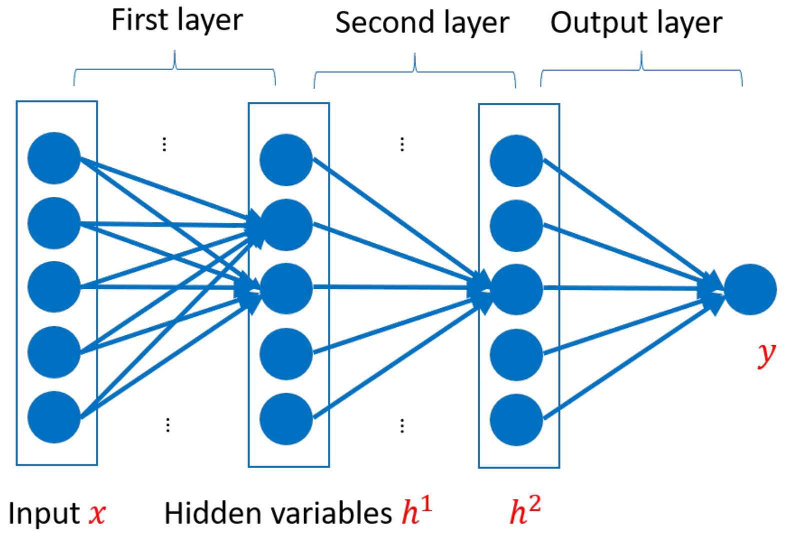
1. Both statements are true.
2. Both statements are false.
3. Statement A is true, Statement B is false.
4. Statement B is true, Statement A is false.

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

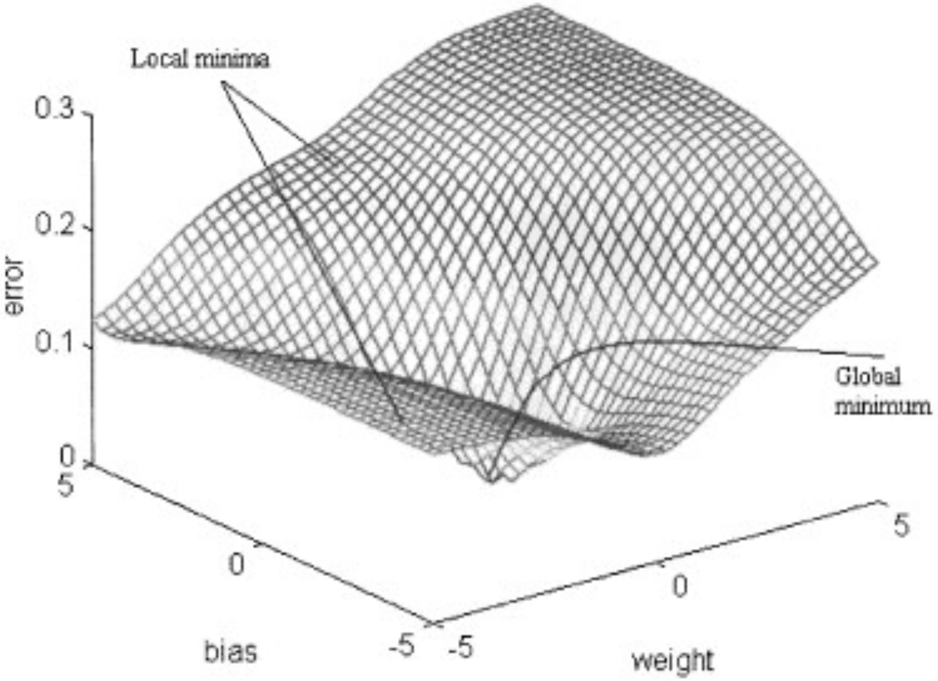
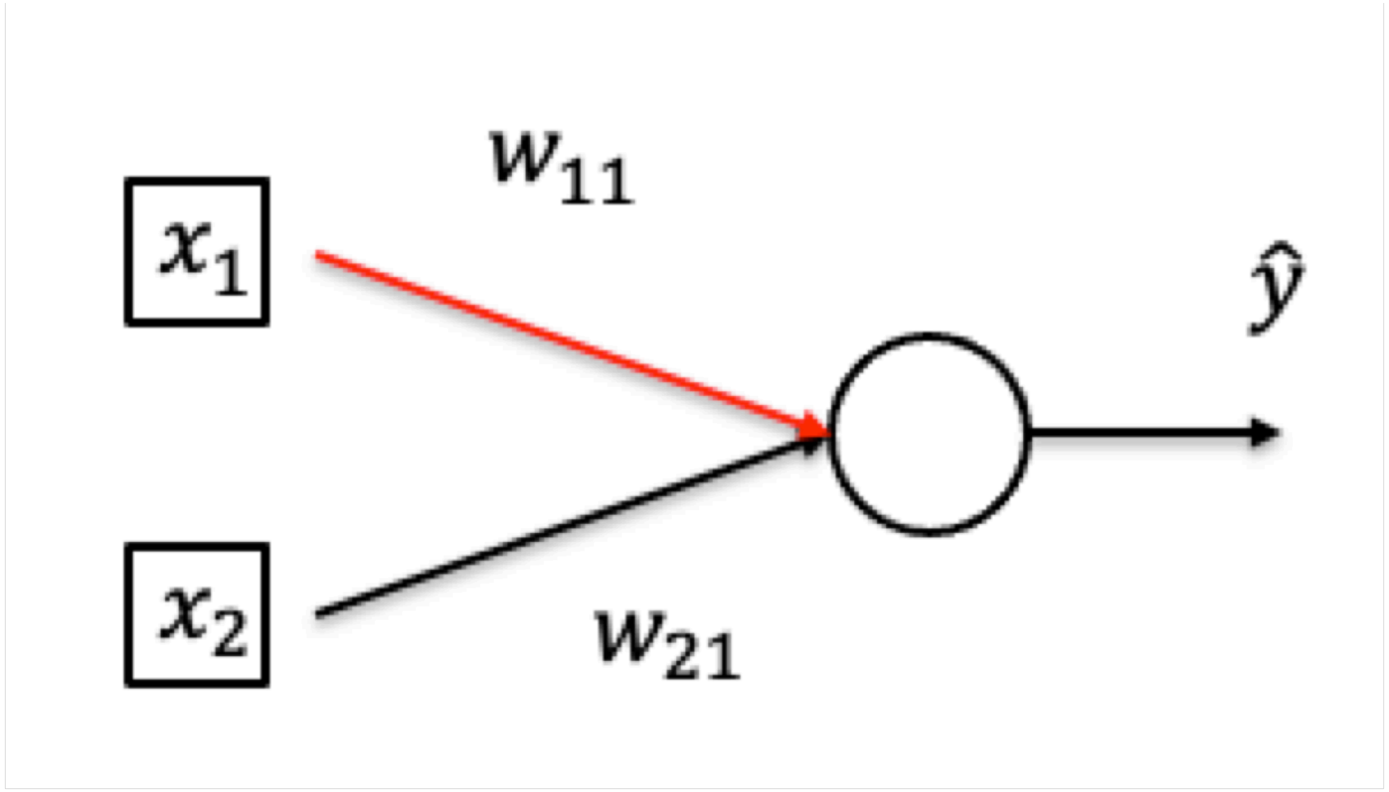


figure from Cho & Chow, *Neurocomputing* 1999

Training Neural Networks

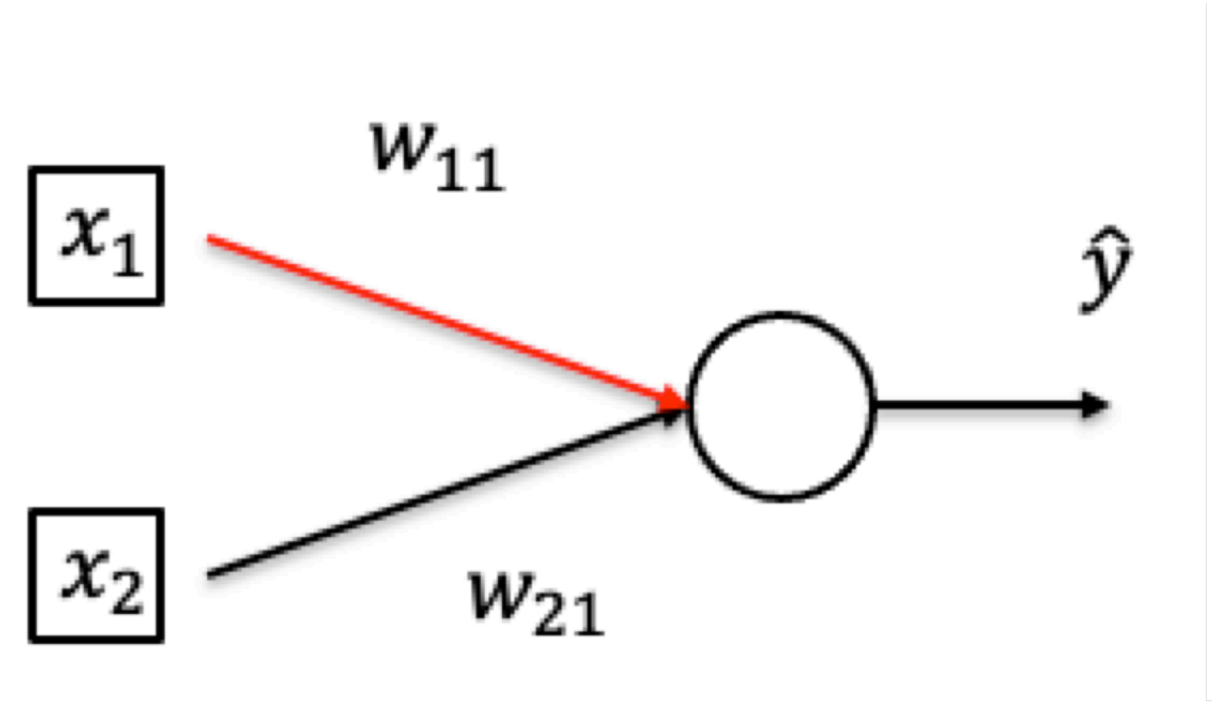
- Algorithm:
  - Get $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$
  - Initialize weights
  - Until stopping criteria met,
    - For each training point $(x^{(i)}, y^{(i)})$
    - Compute: $f_{\text{network}}(x^{(d)})$ ⟵ **Forward Pass**
    - Compute gradient: $\nabla L^{(i)}(w) = \left[\frac{\partial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

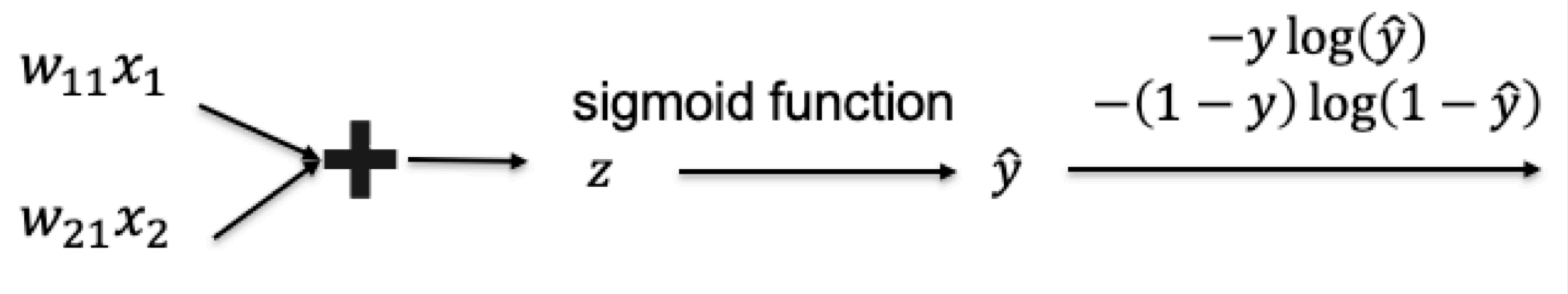
Computing Gradients



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

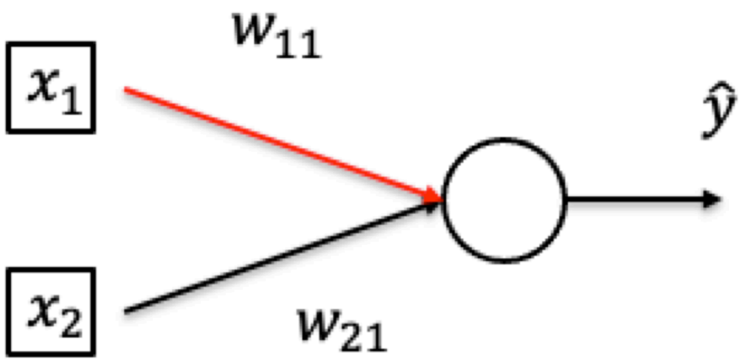
Computing Gradients

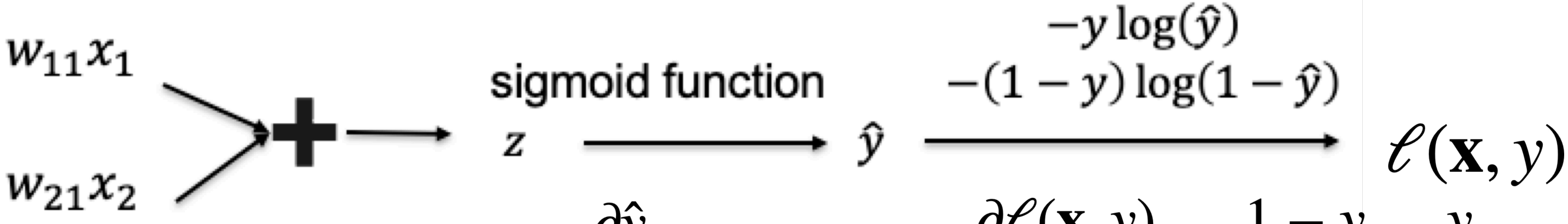




$\ell(\mathbf{x}, y)$

Computing Gradients





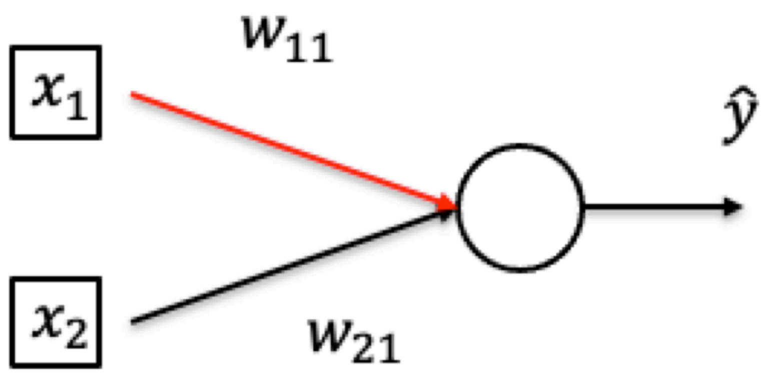
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z)$$

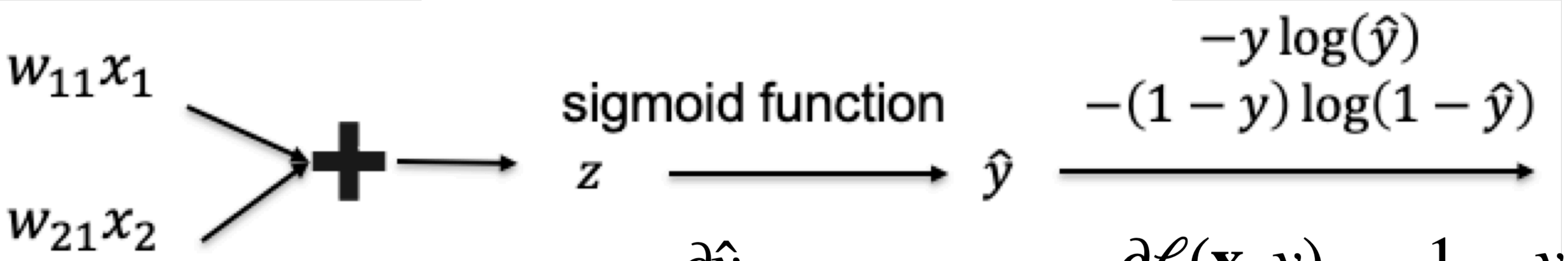
$$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

- 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



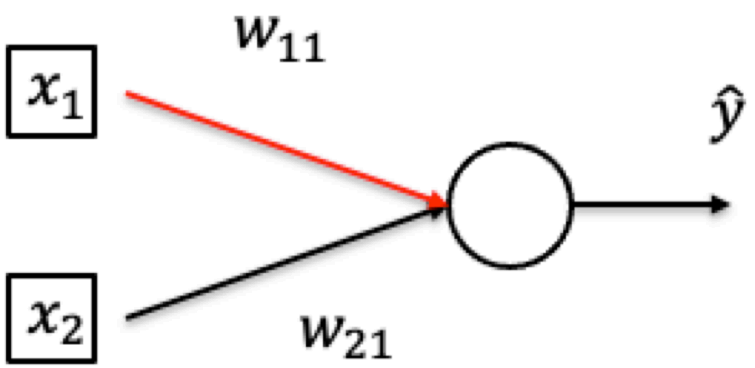


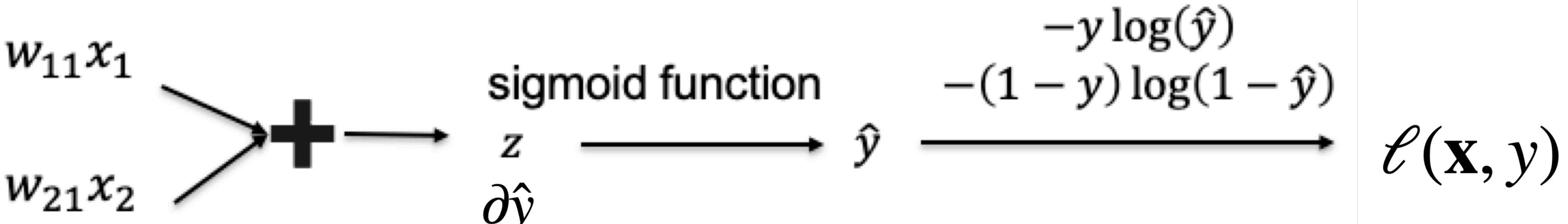
$\ell(\mathbf{x}, y)$

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

- By chain rule: $$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} x_1$$

Computing Gradients



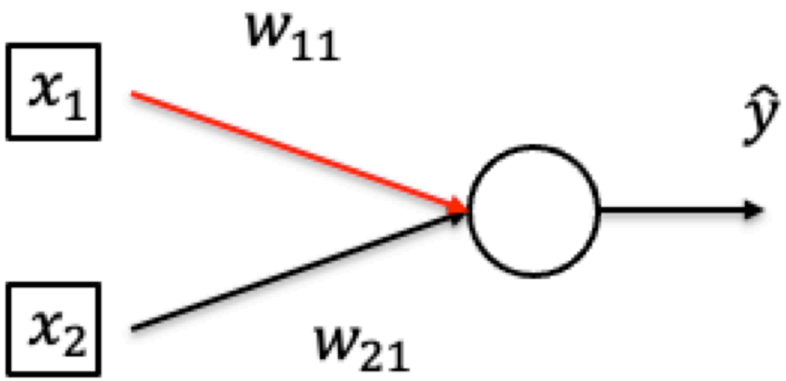


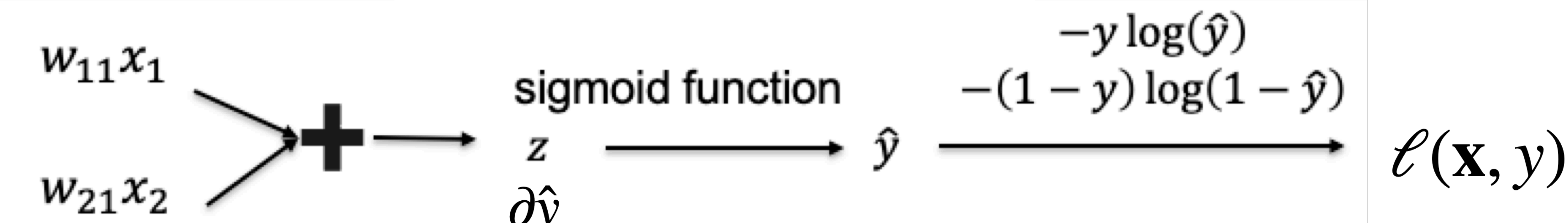
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- By chain rule:

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

Computing Gradients

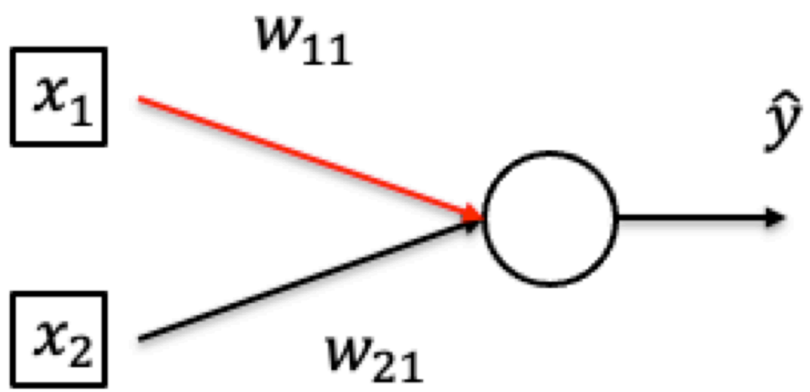


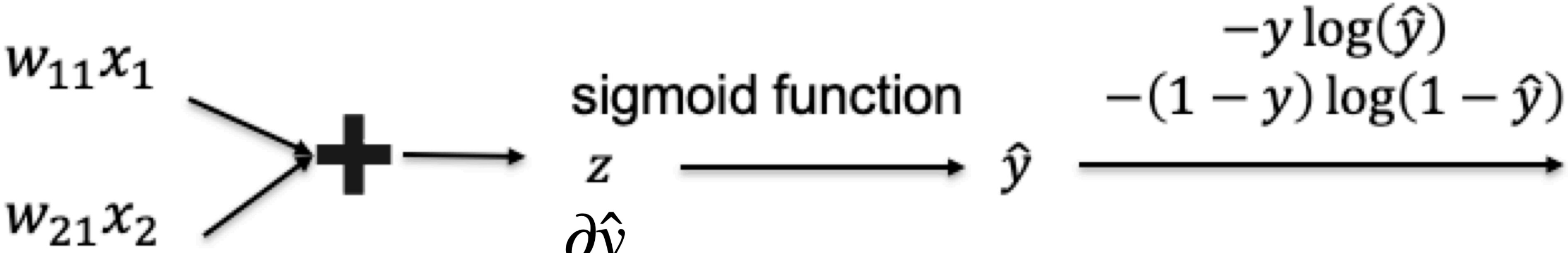


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Computing Gradients



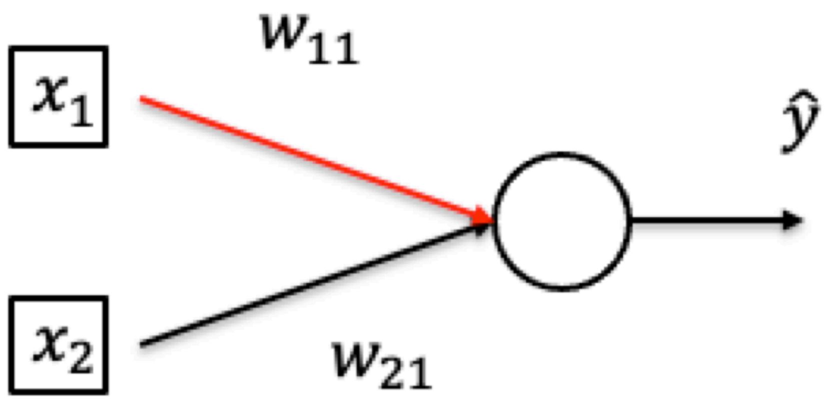


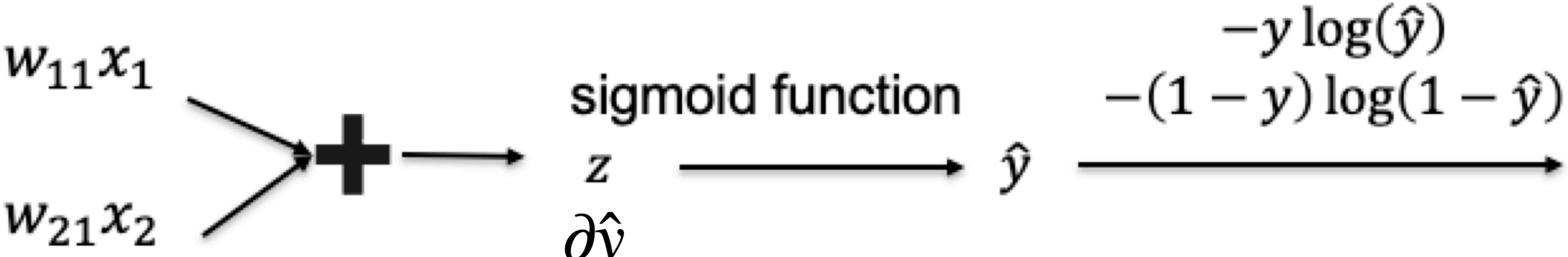
$\ell(\mathbf{x}, y)$

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Computing Gradients





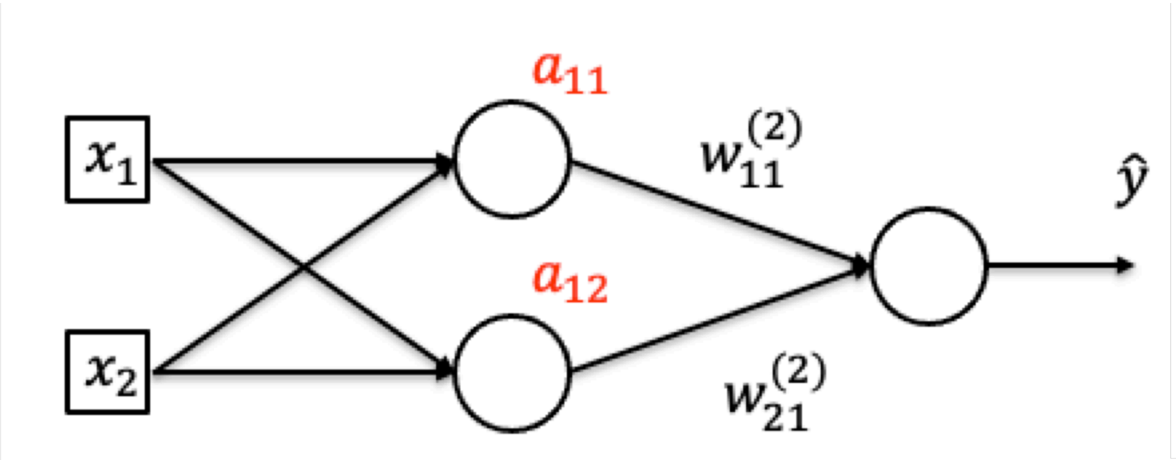
$\ell(\mathbf{x}, y)$

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

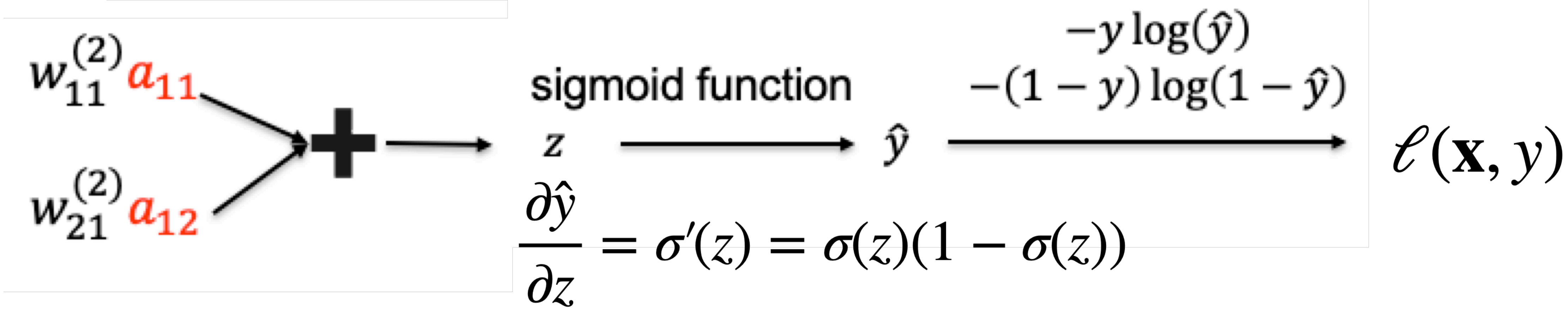
- By chain rule:

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

Computing Gradients: More Layers



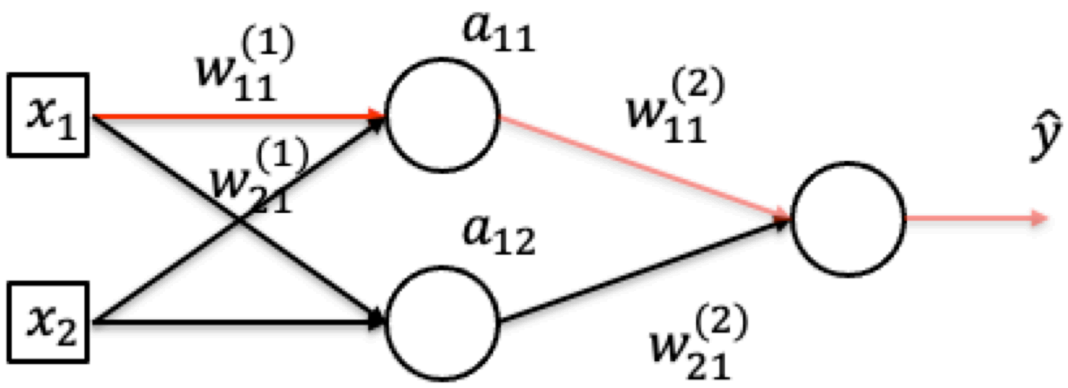
Make it deeper

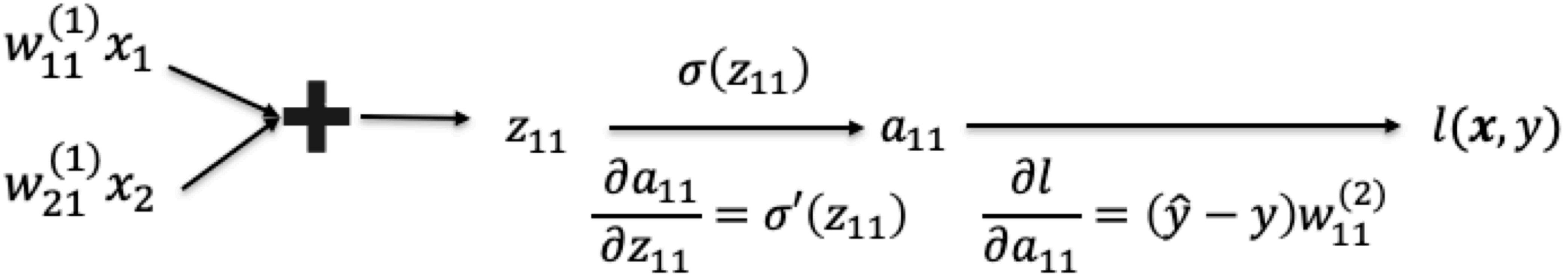


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- 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



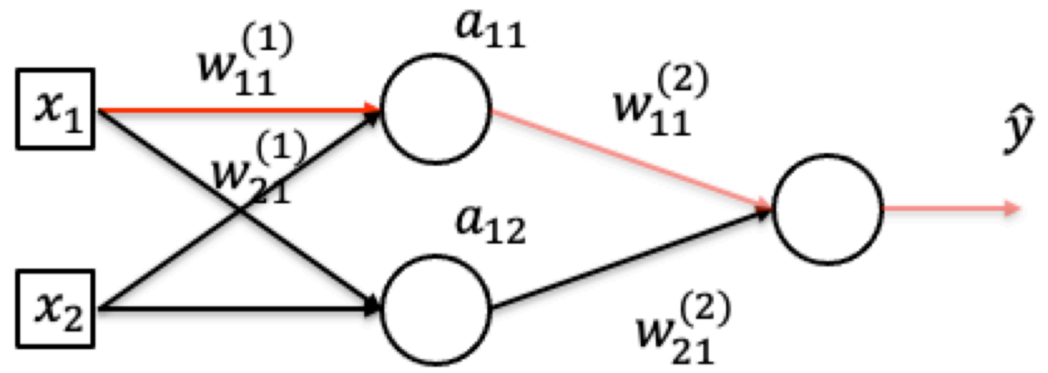


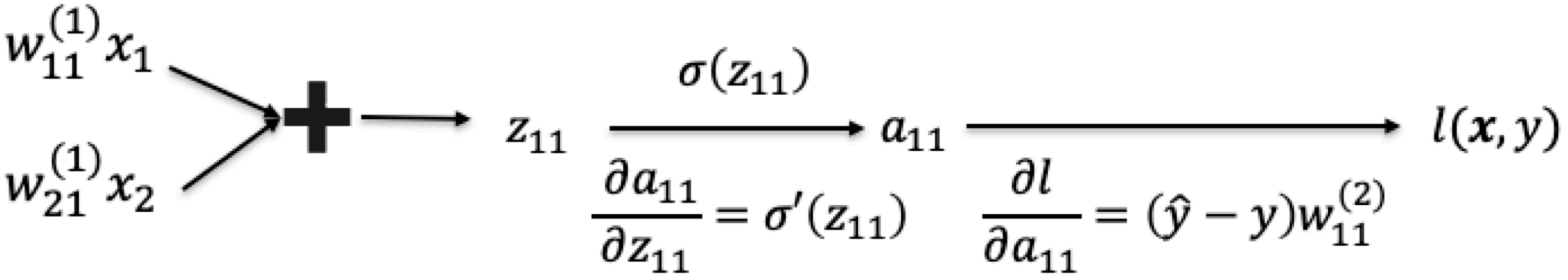
$$\begin{tikzcd} w^{(1)}_{11}x_1 \arrow[dr] & & & & \\ & + \arrow[r] & z_{11} \arrow[r, "\sigma(z_{11})"] & a_{11} \arrow[r] & l(\boldsymbol{x},y) \\ w^{(1)}_{21}x_2 \arrow[ur] & & & & \end{tikzcd}$$

$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y}-y)w^{(2)}_{11}$$

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

Computing Gradients: More Layers

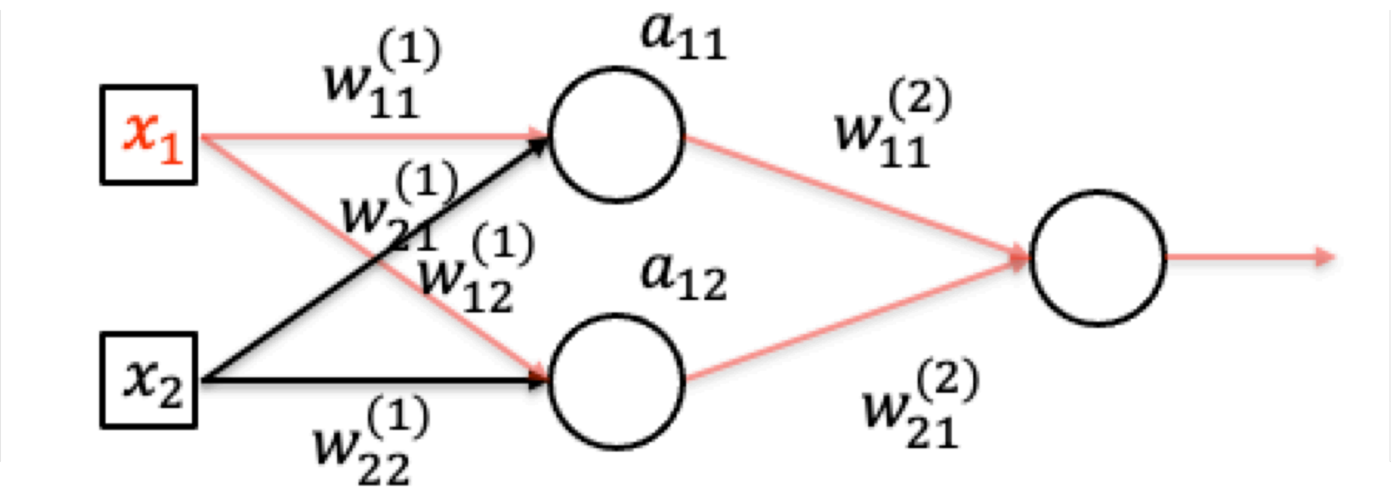


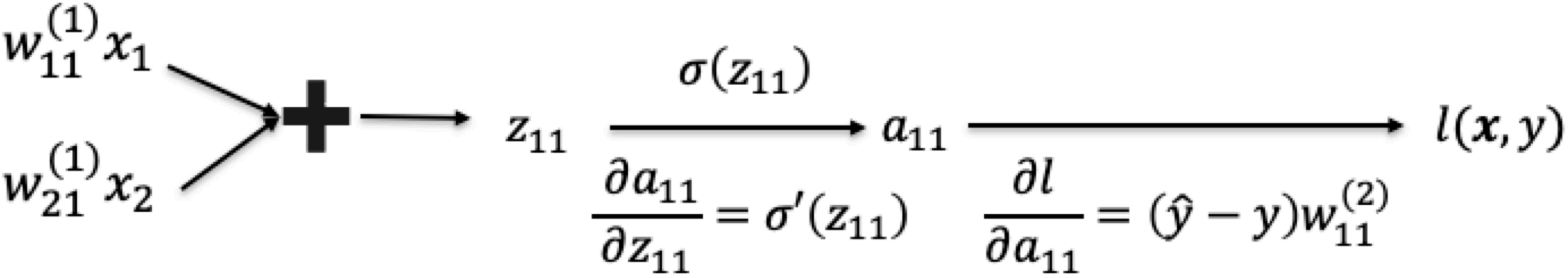


$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

- 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





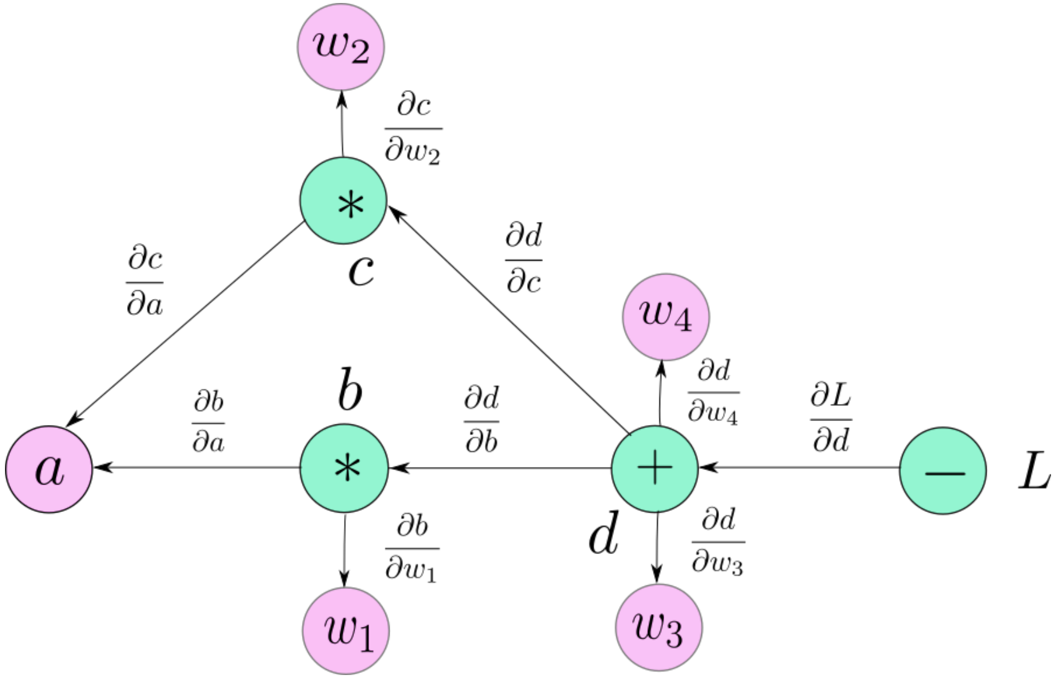
$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

- 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