CS 760: Machine Learning
Neural Networks

University of Wisconsin-Madison

Outline

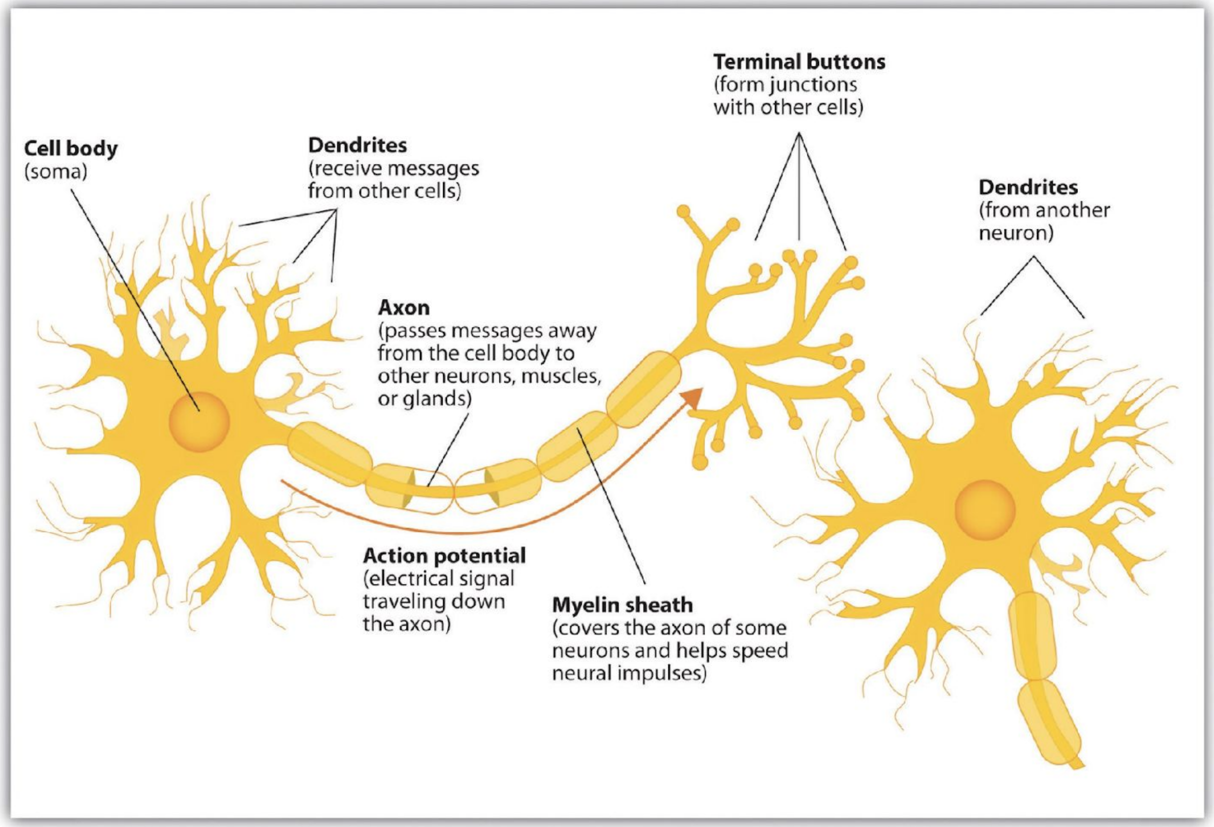
- **Origins: The Perceptron Algorithm**
    - Definition, Training, Loss Equivalent, Mistake Bound
- **Neural Networks**
    - Introduction, Setup, Components, Activations
- **Training Neural Networks**
    - SGD, Computing Gradients, Backpropagation

Outline

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

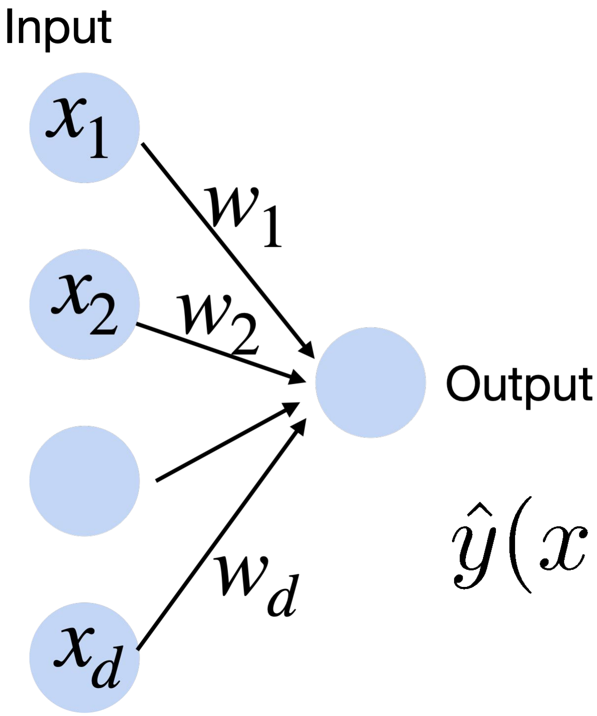
Neural networks: Origins

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



Terminal buttons
(form junctions with other cells)
Cell body
(soma)
Dendrites
(receive messages from other cells)
Dendrites
(from another neuron)
Axon
(passes messages away from the cell body to other neurons, muscles, or glands)
Action potential
(electrical signal traveling down the axon)
Myelin sheath
(covers the axon of some neurons and helps speed neural impulses)

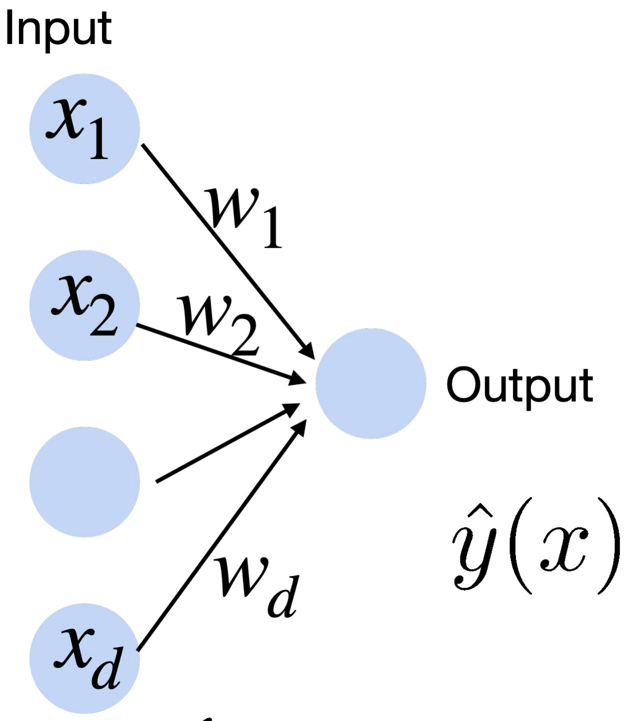
Perceptron: a single "neuron"



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

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

Perceptron: Components



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

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

Linear Transformation **+** **Activation Function**

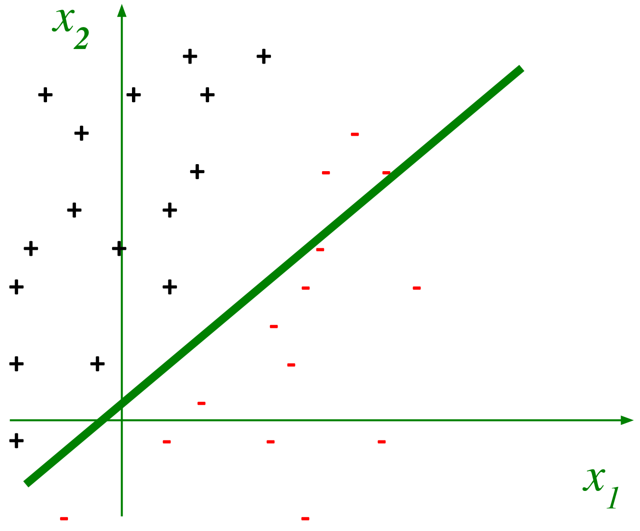
[McCulloch & Pitts, **1943**; Rosenblatt, **1959**; Widrow & Hoff, **1960**]

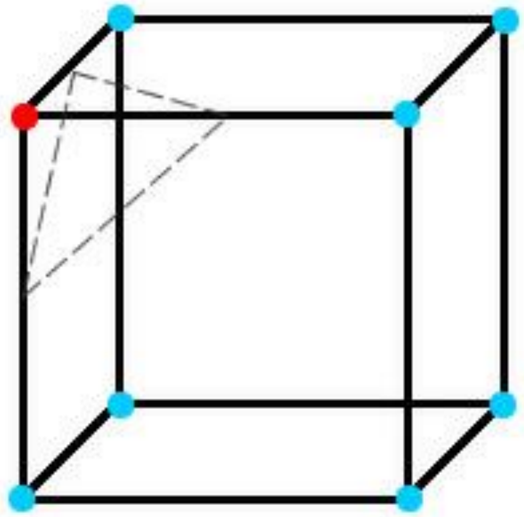
Perceptron: Representational Power

- 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:

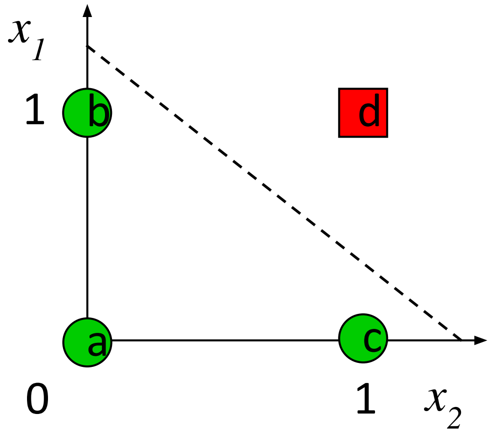




Which Functions are **Linearly Separable?**

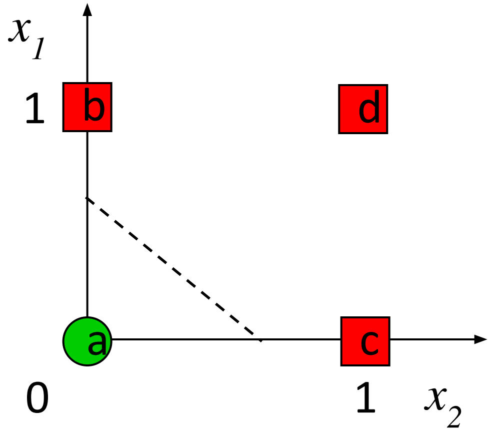
AND

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



OR

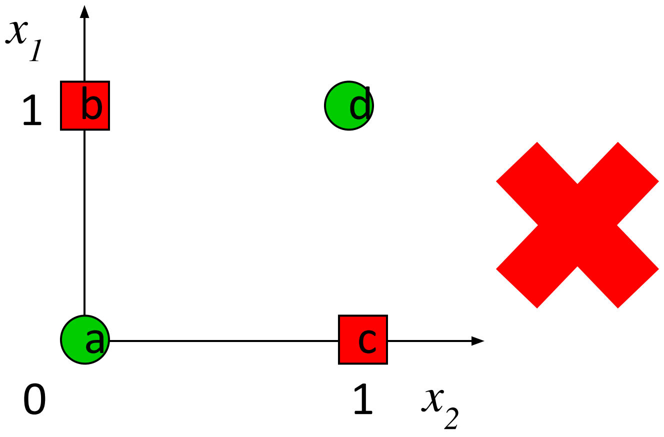
	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	1



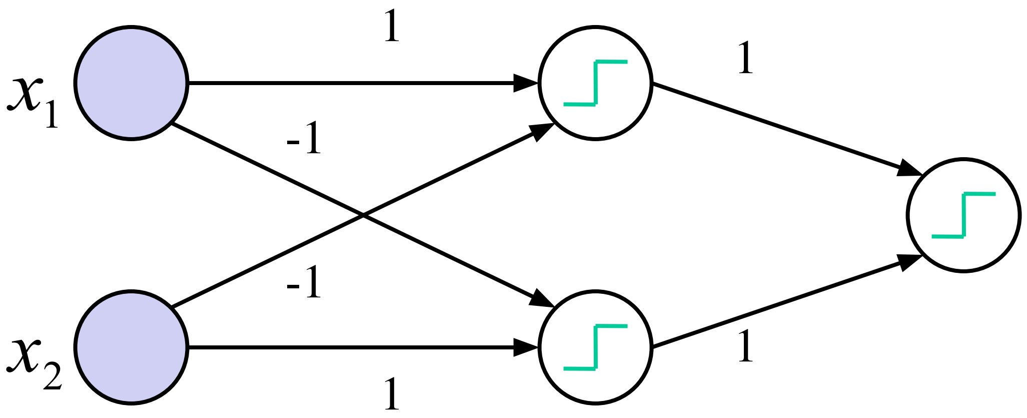
Which Functions are **Linearly Separable?**

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!



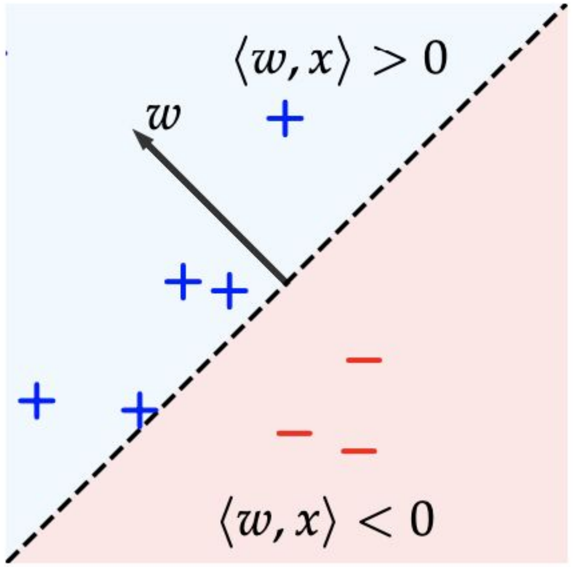
(assume activation is $\sigma(x) = 1_{\{x>0\}}$)

Perceptron: Training

- when are we correct?

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

  - i.e. **signs** of prediction and label match



- could also require a “margin”:

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

  - notion of robustness / easiness of classification

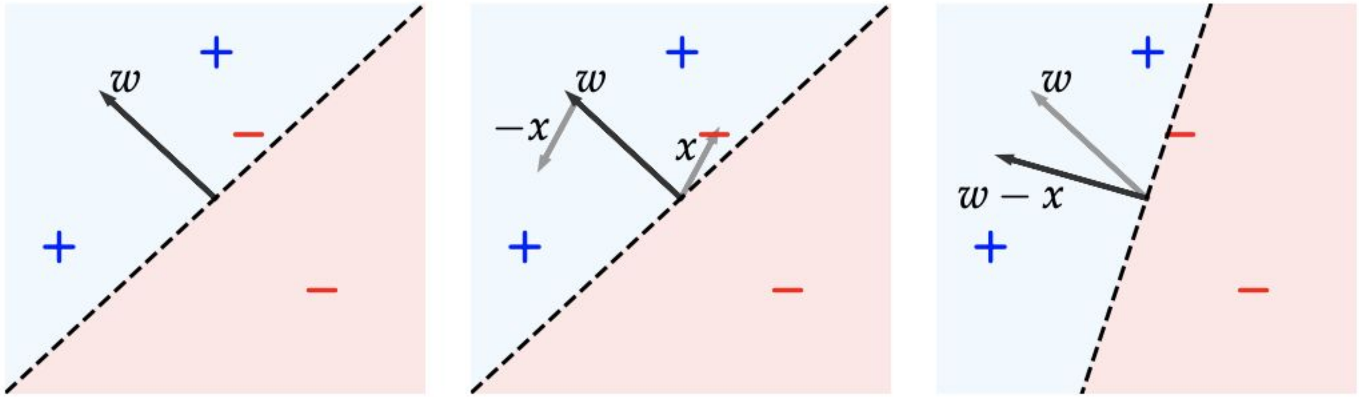
Perceptron: Training

- **Algorithm**:
  - Initialize at $w_0 = [0, \ldots, 0]^T$
  - At step t = 0,…
  - Select a datapoint i (randomly or cyclically)

  - If $y^{(i)} w^T x^{(i)} < c$ then do $w_{t+1} = w_t + y^{(i)} x^{(i)}$

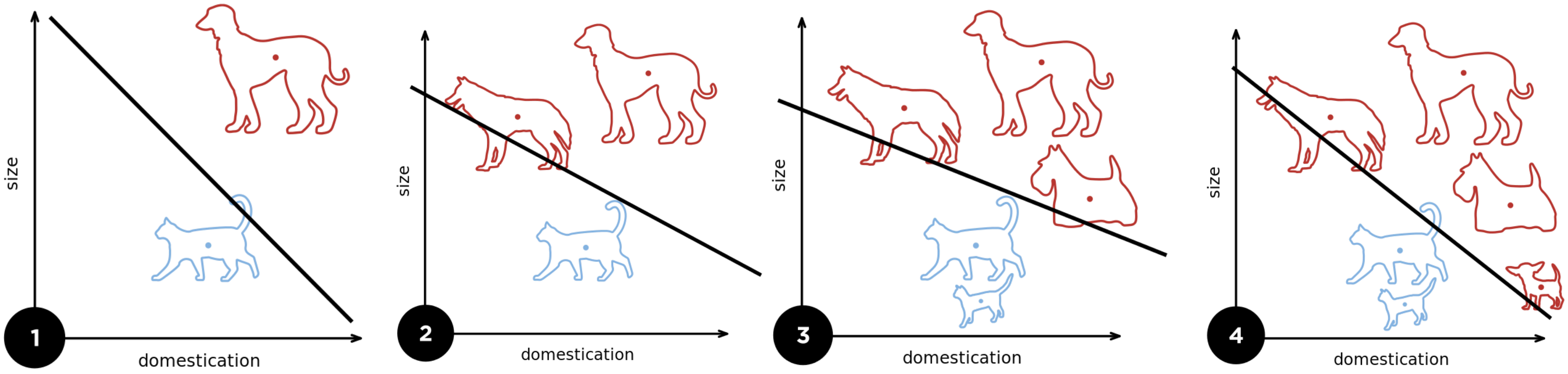
  - Else, $w_{t+1} = w_t$

margin



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 is the update to our prediction?

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

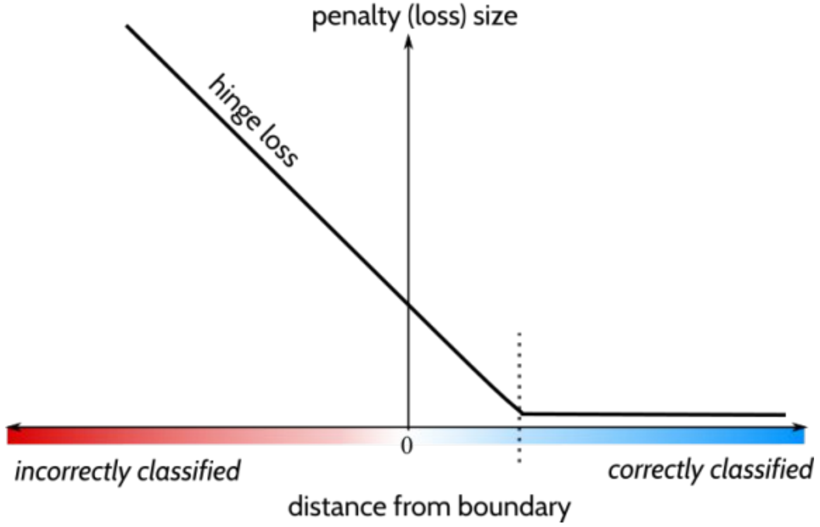
Perceptron: Training Comparison

- 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



Hinge loss!

Perceptron: Analysis

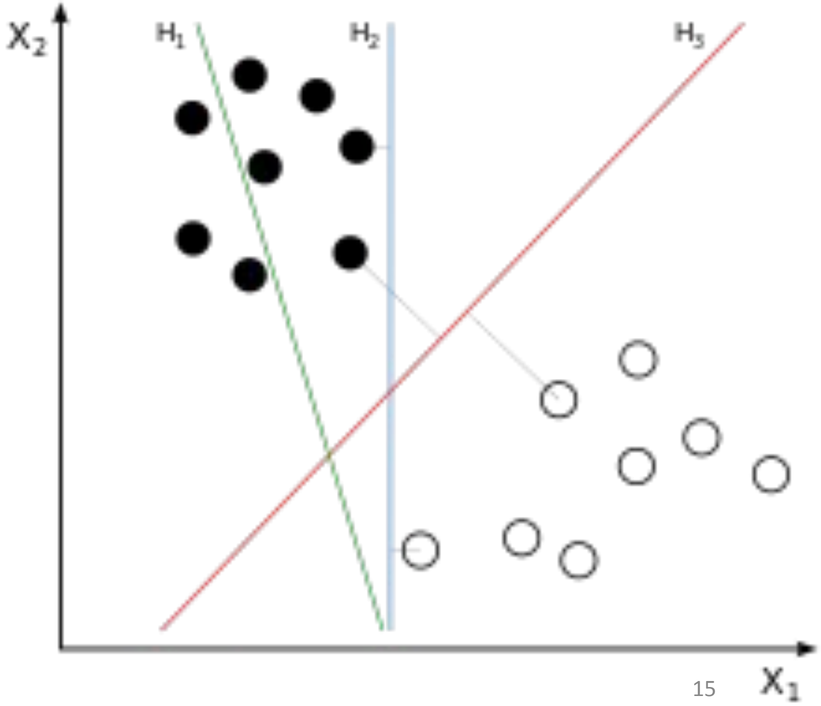
- **How many mistakes** does the Perceptron algorithm make?
- Key quantity needed: **data margin**
  - Hyperplane $H_w = x : w^T x = 0$
  - Margin

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

$$\downarrow$$

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

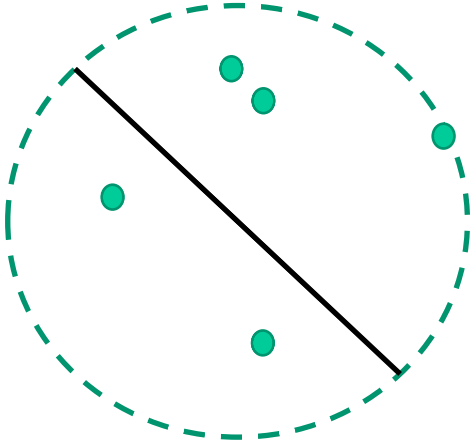
$$\gamma(S) = \max_{\|w\|=1} \gamma(S, w)$$



Perceptron: Mistake Bound

- Another quantity needed: **data diameter**

$$D(S) = \max_{(x,y)\in S} \|x\|$$



Mistake Bound Result: (Perceptron with $c = 0$)

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

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

Perceptron: Mistake Bound Interpretation

Mistake Bound Result:

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

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

smaller means harder
to find separator

Implications?

- running over a dataset S repeatedly until # mistakes stops changing gives you a perfect separator
- says nothing about **generalization** (without further work)

Mistake Bound: Proof 1

- 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

$$\|w_{t+1}\|^2 = \|w_t\|^2 + 2\underbrace{(y^{(i_t)} w_t)^T x^{(i_t)}}_{\text{negative on every mistake, which is the only time an update is made}} + \|x^{(i_t)}\|^2$$

diameter

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

Mistake Bound: Proof 2

- This is true for each mistake

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

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

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

  - This was also a telescoping argument, like we used for gradient descent

Mistake Bound: Proof 3

- Now we'll also *lower bound* the norm
- Let *w* be a unit-norm **separating** hyperplane

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

w classifies correctly

= 1

- But this is the margin for $x^{(i_t)}$, 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 use 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,
$\gamma(S, w_*)$ for mistake

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

Mistake Bound: Proof 6

- So, $m_t \gamma(S) \le \|w_t\|$ $\quad \|w_t\| \le D(S)\sqrt{m_t}$

- thus

$$m_t \gamma(S) \le \|w_t\| \le D(S)\sqrt{m_t}$$

- Easy algebra gets us to

$$m_t \le \frac{D(S)^2}{\gamma(S)^2}$$ ✓

Break & Quiz

Q: 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.

Q: 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.*

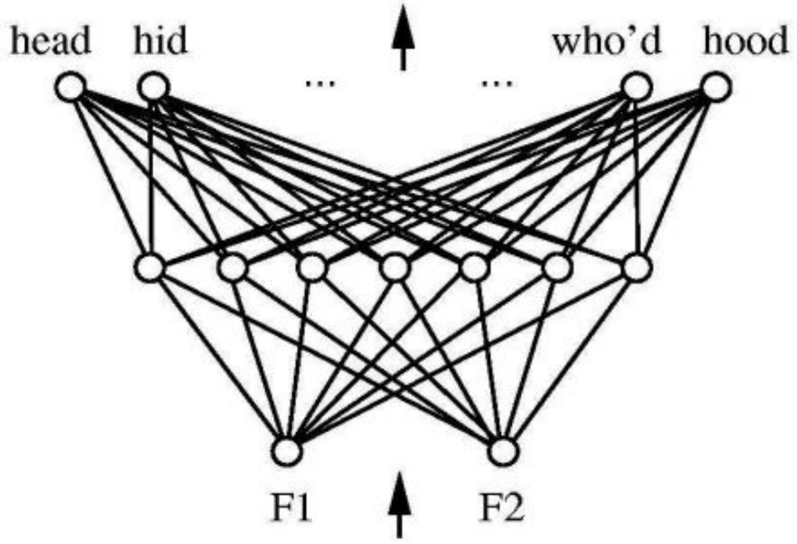
1. Both statements are true.
2. Both statements are false.
3. [✓ Statement A is true, Statement B is false.]
4. Statement B is true, Statement A is false.

Outline

- **Origins: The 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, *NeurIPS* 1988

Neural Network **Decision Regions**

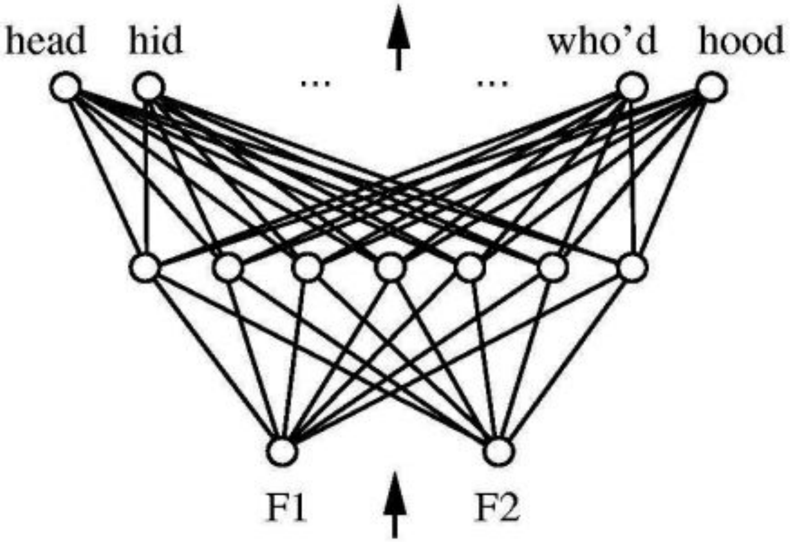
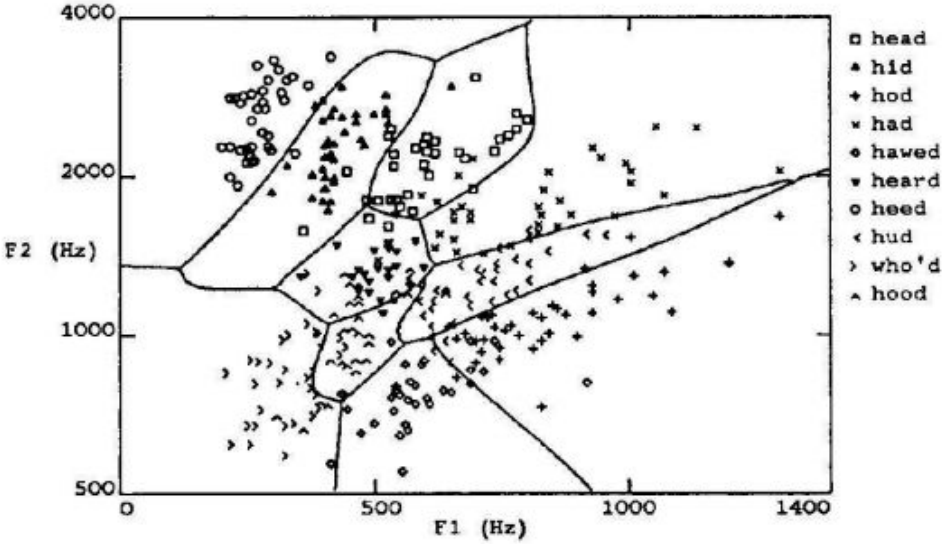
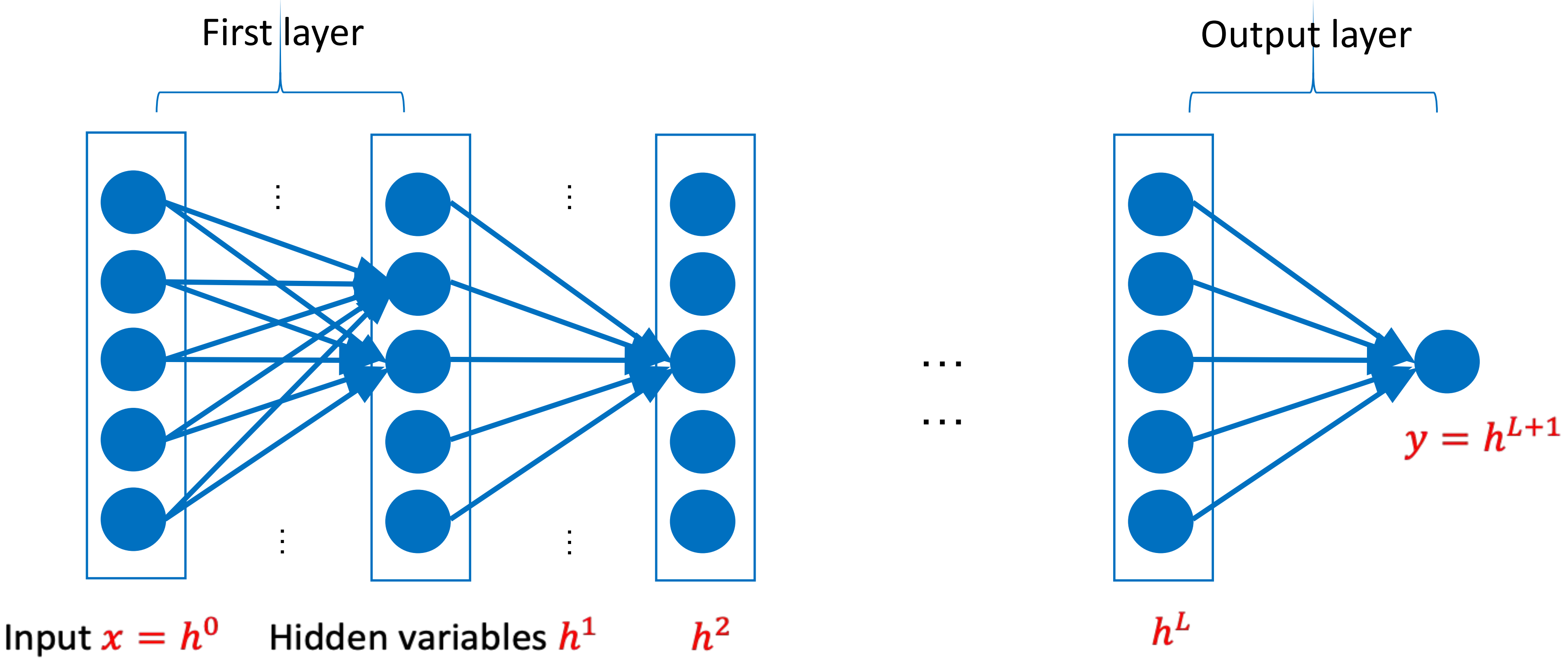


Figure from Huang & Lippmann, *NeurIPS* 1988



Neural Network Components

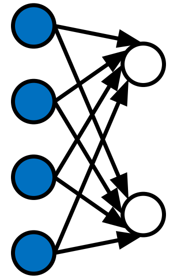
- An $(L+1)$-layer network



Feature Encoding for NNs

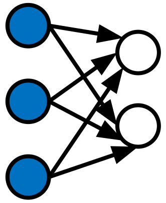
- Nominal features usually a one hot encoding

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



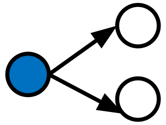
- Ordinal features: use a *thermometer* encoding

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



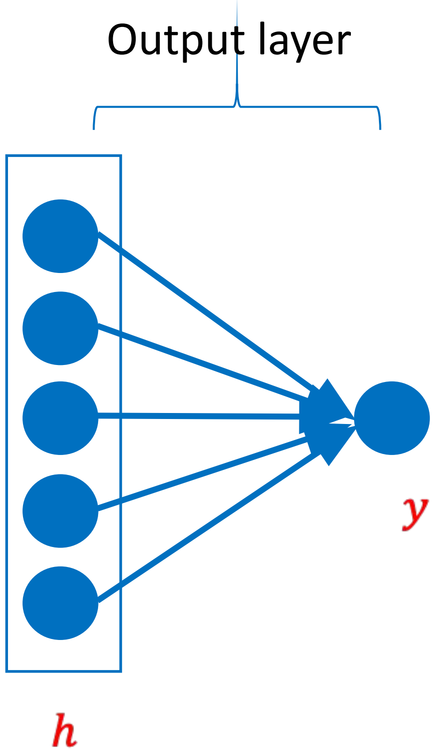
- Real-valued features use individual input units (may want to scale/normalize them first though)

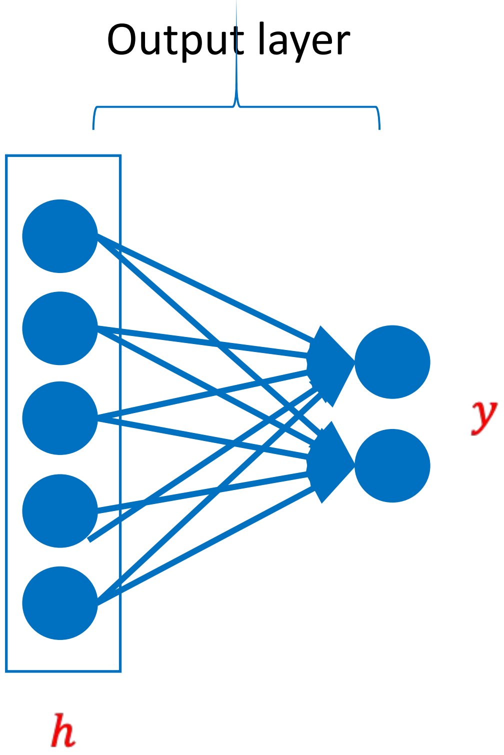
$$\text{precipitation} = \begin{bmatrix}0.68\end{bmatrix}$$



Output Layer: Examples

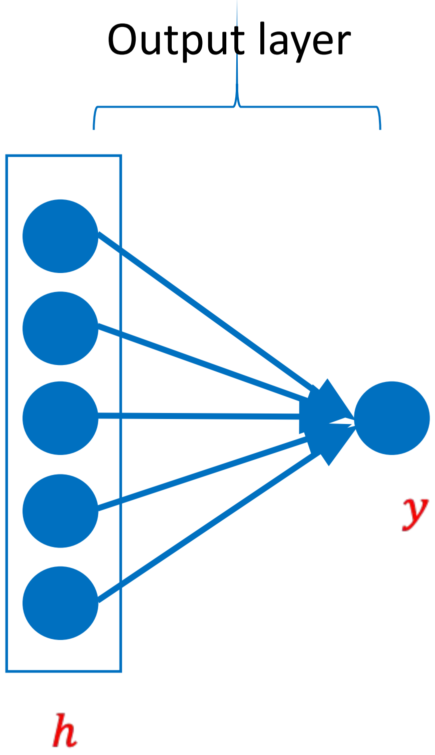
- Regression: $y = w^T h + b$
  - Linear units: no nonlinearity
- Multi-dimensional regression: $y = W^T h + b$
  - Linear units: no nonlinearity

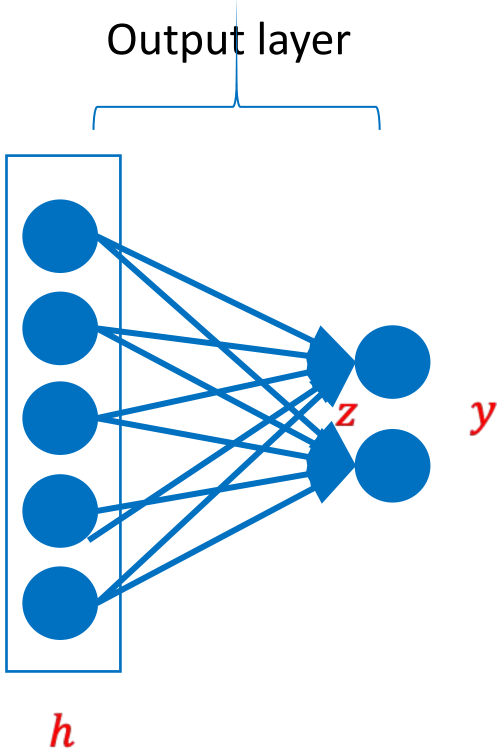




Output Layer: Examples

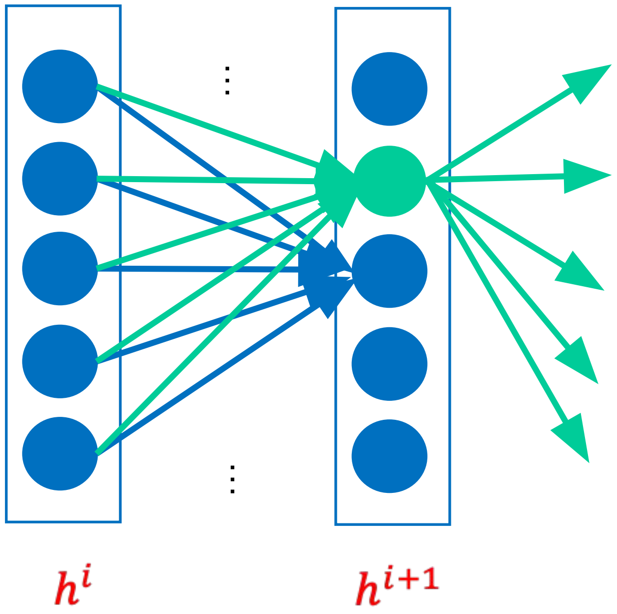
- Binary classification: $y = \sigma(w^T h + b)$
  - Corresponds to using logistic regression on h
- Multiclass classification:
  - $y = \text{softmax}(z)$ where $z = W^T h + b$





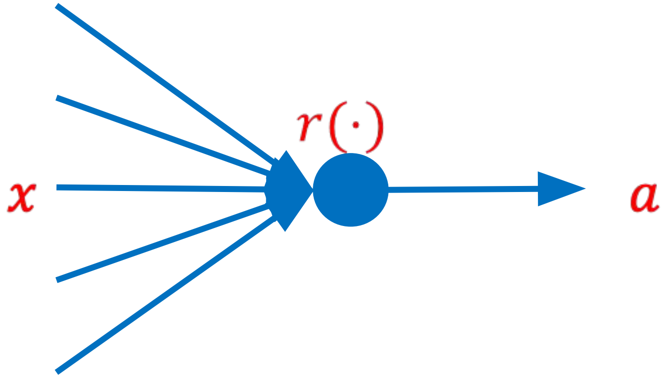
Hidden Layers

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

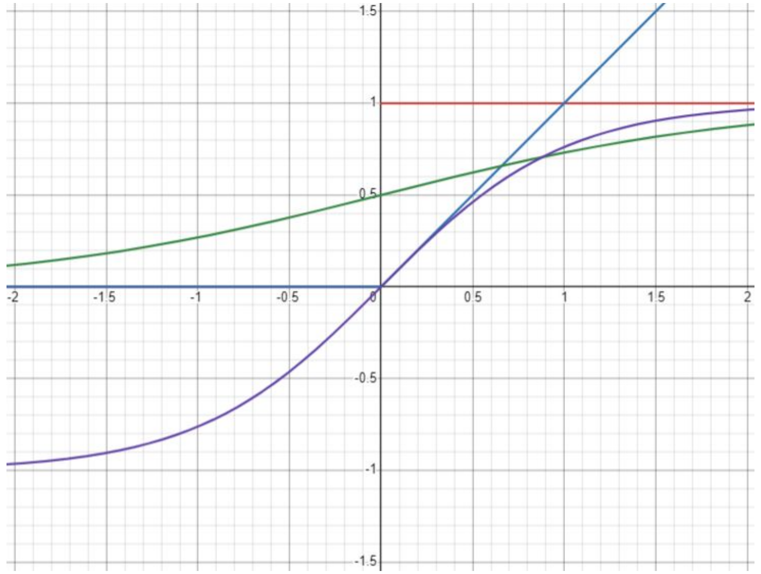


Hidden Layers

- Outputs $a = r(w^T x + b)$



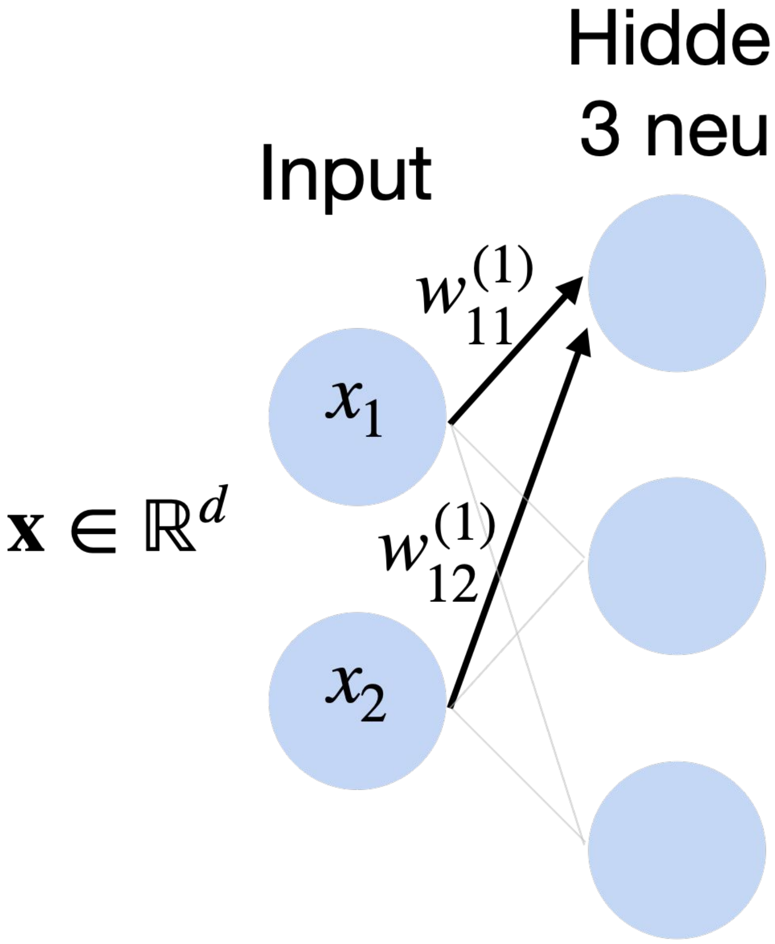
- Typical activation function r
  - threshold $\mathrm{h}(z) = 1_{\{z \geq 0\}}$
  - ReLU $\mathrm{ReLU}(z) = z \cdot t(z) = \max\{0, z\}$
  - sigmoid $\sigma(z) = 1/(1 + \exp(-z))$
  - hyperbolic tangent $\tanh(z) = 2\sigma(2z) - 1$



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

MLPs: Multilayer Perceptron

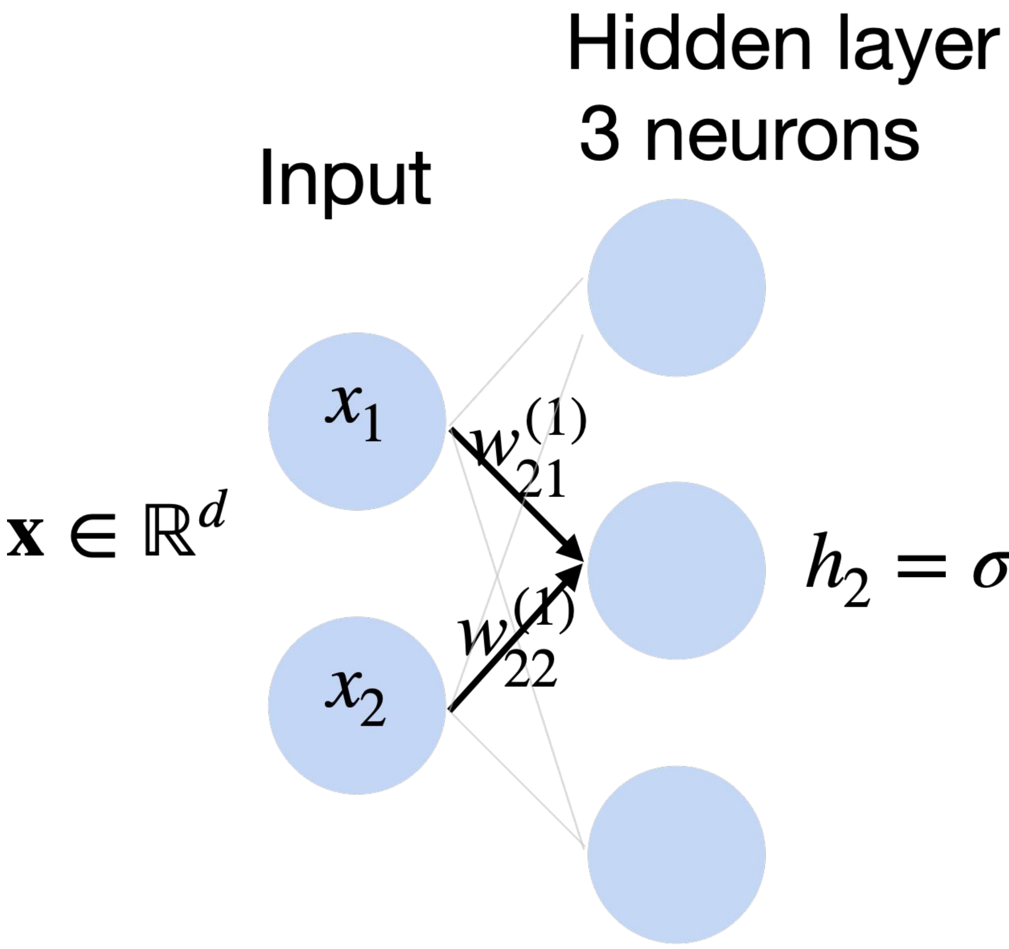
- **Ex**: 1 hidden layer, 1 output layer: depth 2



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

MLPs: Multilayer Perceptron

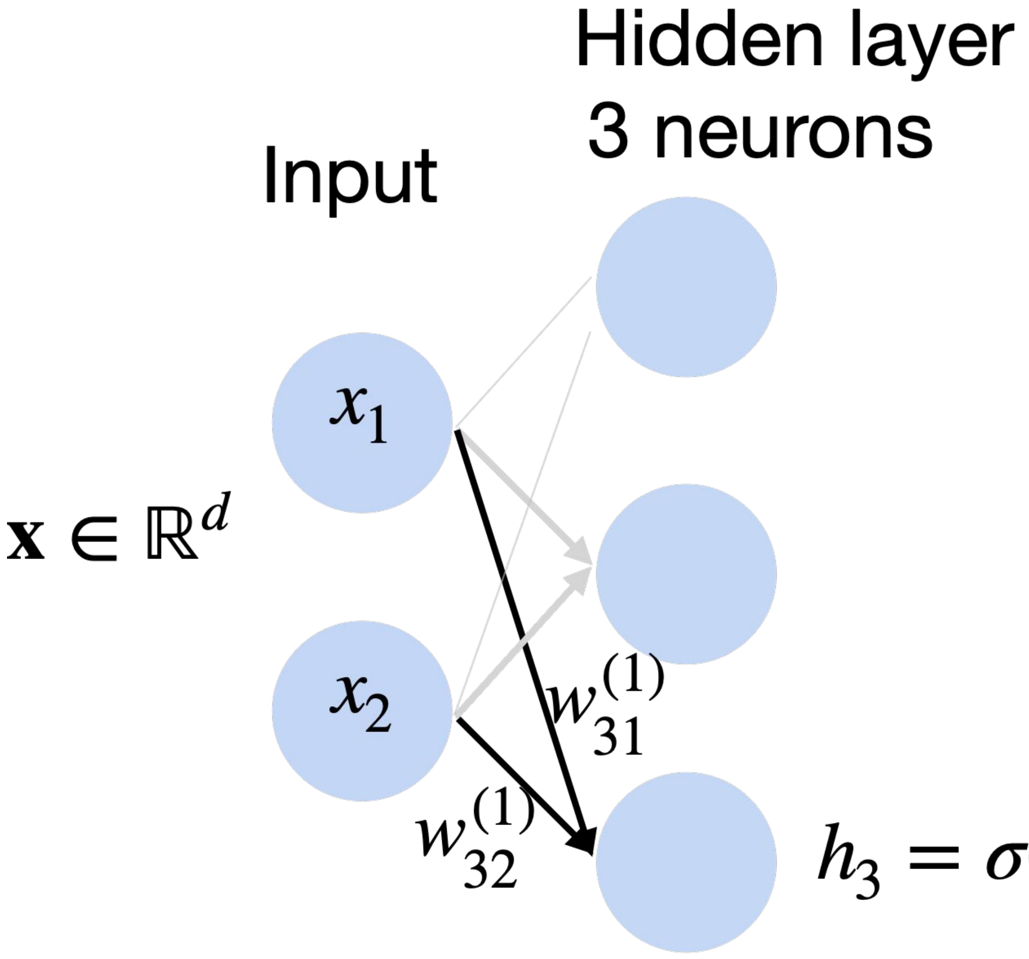
- **Ex**: 1 hidden layer, 1 output layer: depth 2



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

MLPs: Multilayer Perceptron

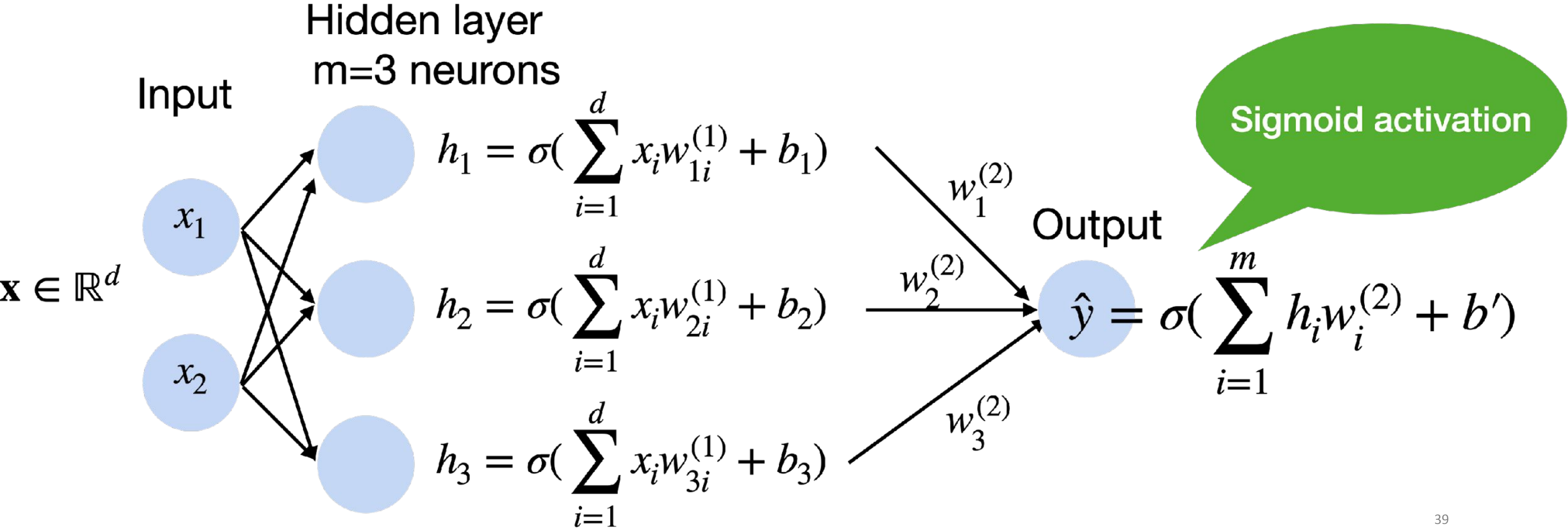
- **Ex**: 1 hidden layer, 1 output layer: depth 2



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

MLPs: Multilayer Perceptron

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



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

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

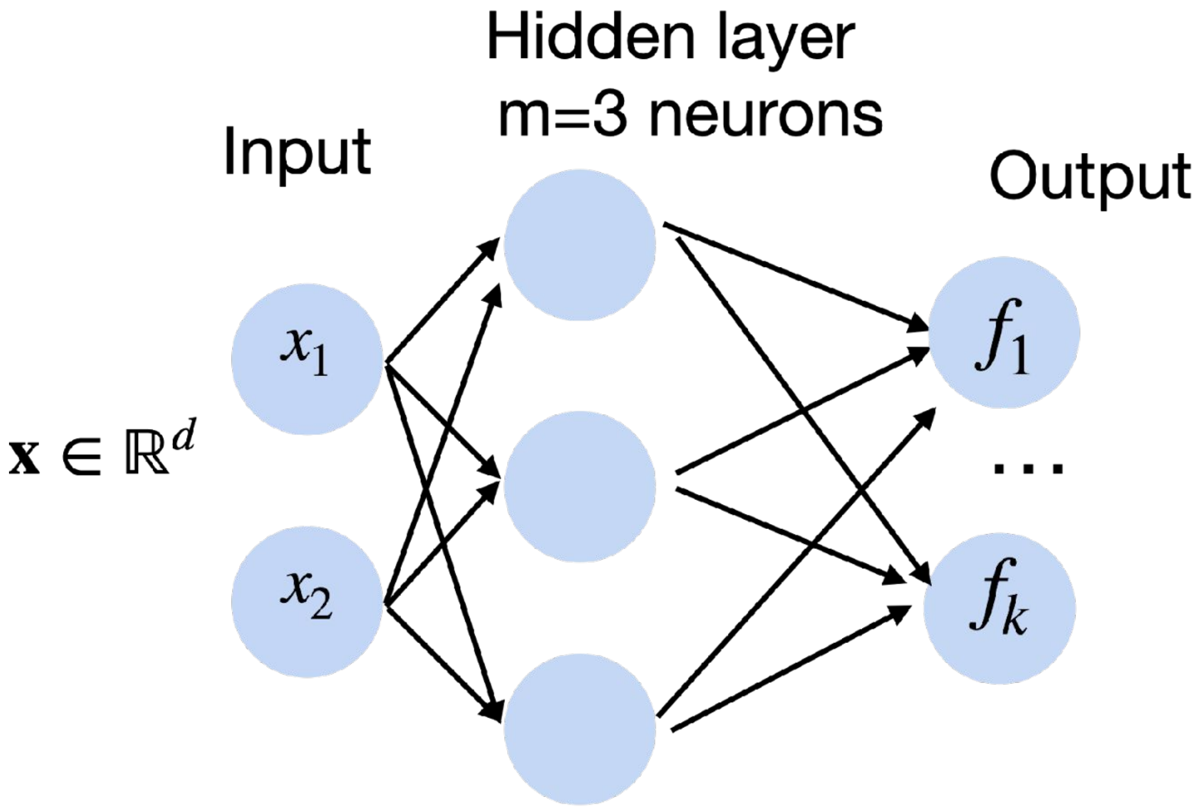
$$h_3 = \sigma(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3)$$

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

Sigmoid activation

Multiclass Classification Output

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

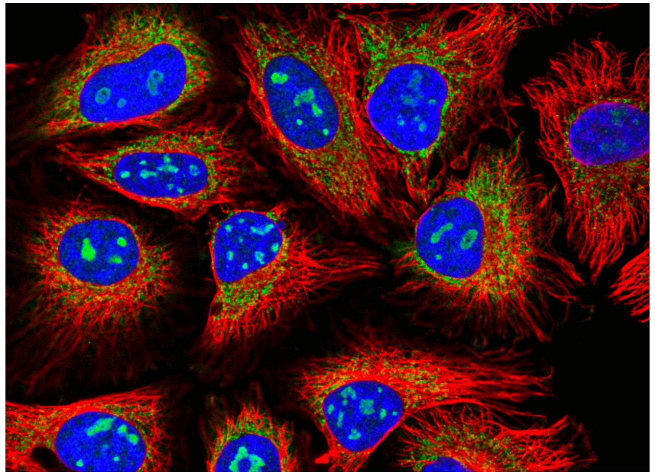


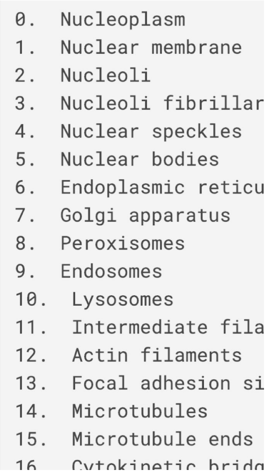
$$p(y \mid \mathbf{x}) = \text{softmax}(f)$$

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

Multiclass Classification Examples

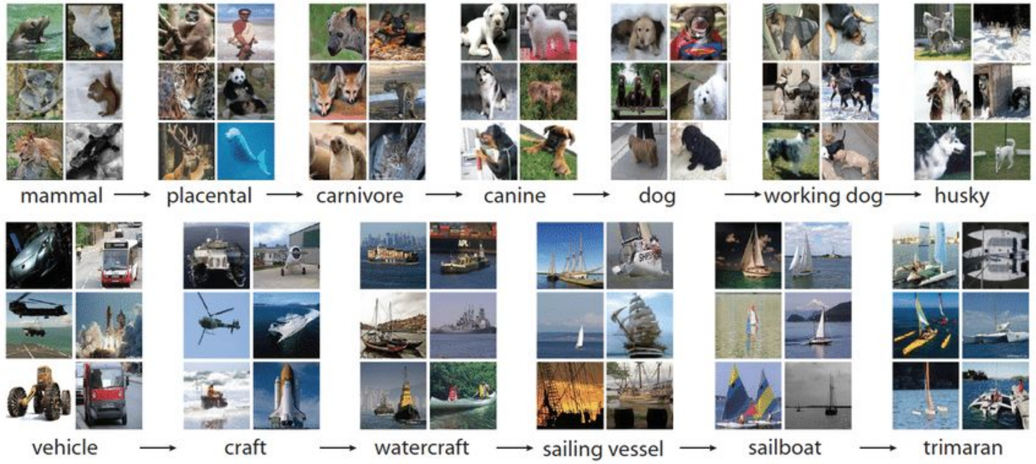
- Protein classification (Kaggle challenge)





```
0.  Nucleoplasm
1.  Nuclear membrane
2.  Nucleoli
3.  Nucleoli fibrillar
4.  Nuclear speckles
5.  Nuclear bodies
6.  Endoplasmic reticu
7.  Golgi apparatus
8.  Peroxisomes
9.  Endosomes
10.  Lysosomes
11.  Intermediate fila
12.  Actin filaments
13.  Focal adhesion si
14.  Microtubules
15.  Microtubule ends
16.  Cytokinetic bridg
```

- ImageNet


mammal → placental → carnivore → canine → dog → working dog → husky
vehicle → craft → watercraft → sailing vessel → sailboat → trimaran

Break & Quiz

Q: 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.*

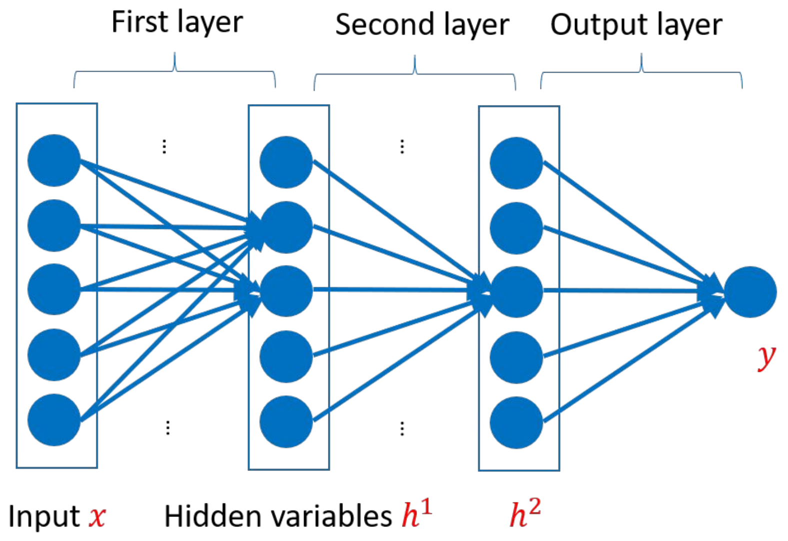
1. Both statements are true.
2. Both statements are false.
3. Statement A is true, Statement B is false.
4. Statement A is false, Statement B is true.

Q: 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 A is false, Statement B is true.



Outline

- **Origins: The 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 is done in the usual way: pick a loss and optimize it

- **Example**: 2 scalar weights

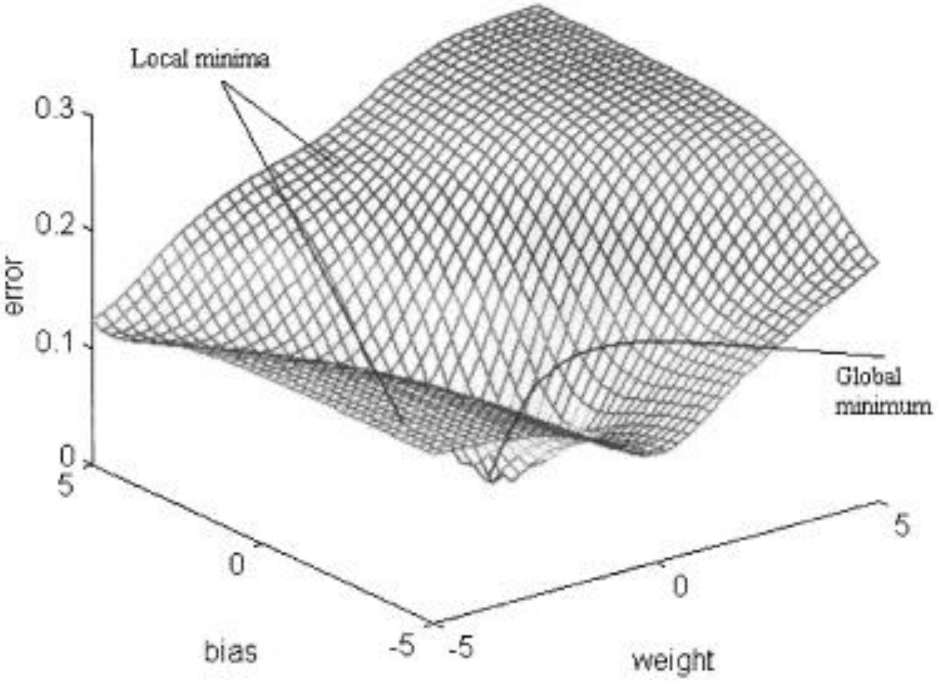


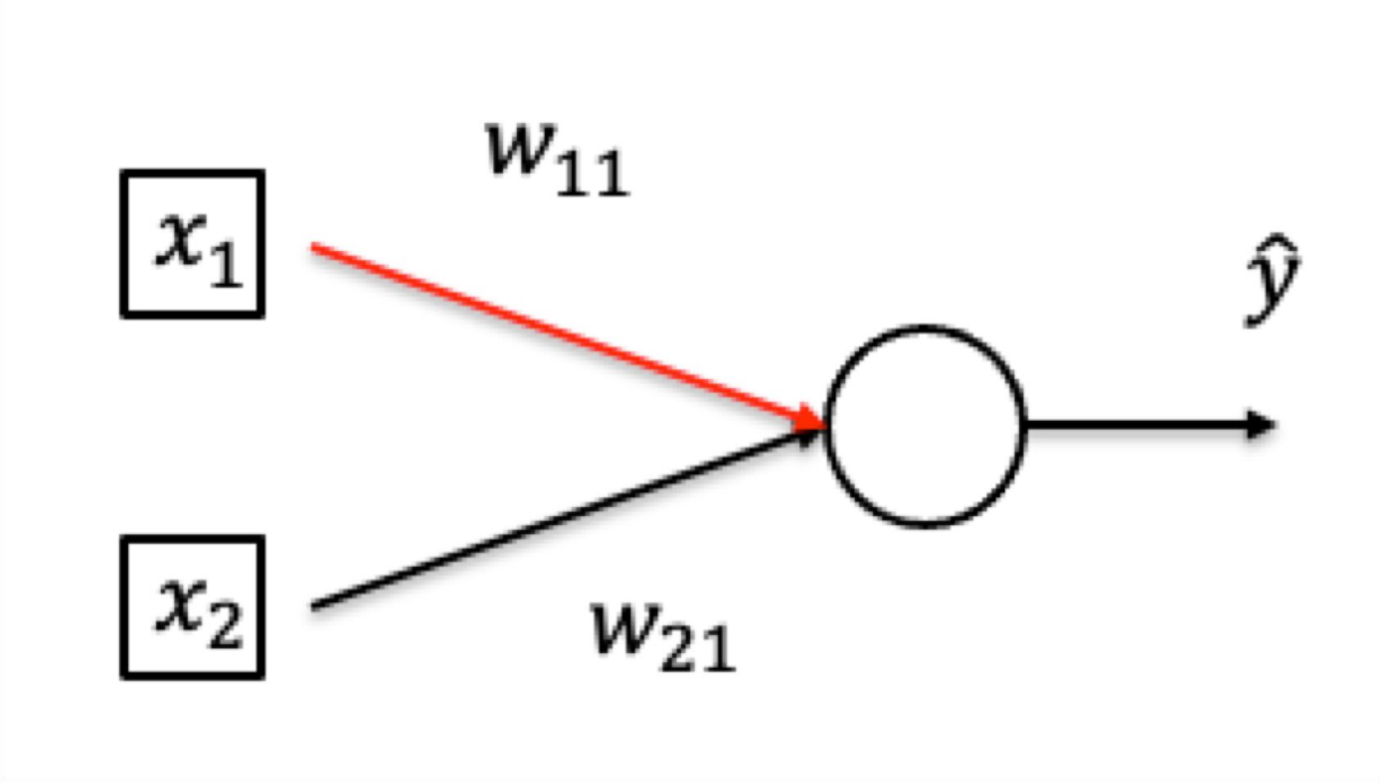
figure from Cho & Chow, *Neurocomputing* 1999

Training Neural Networks with SGD

Algorithm:

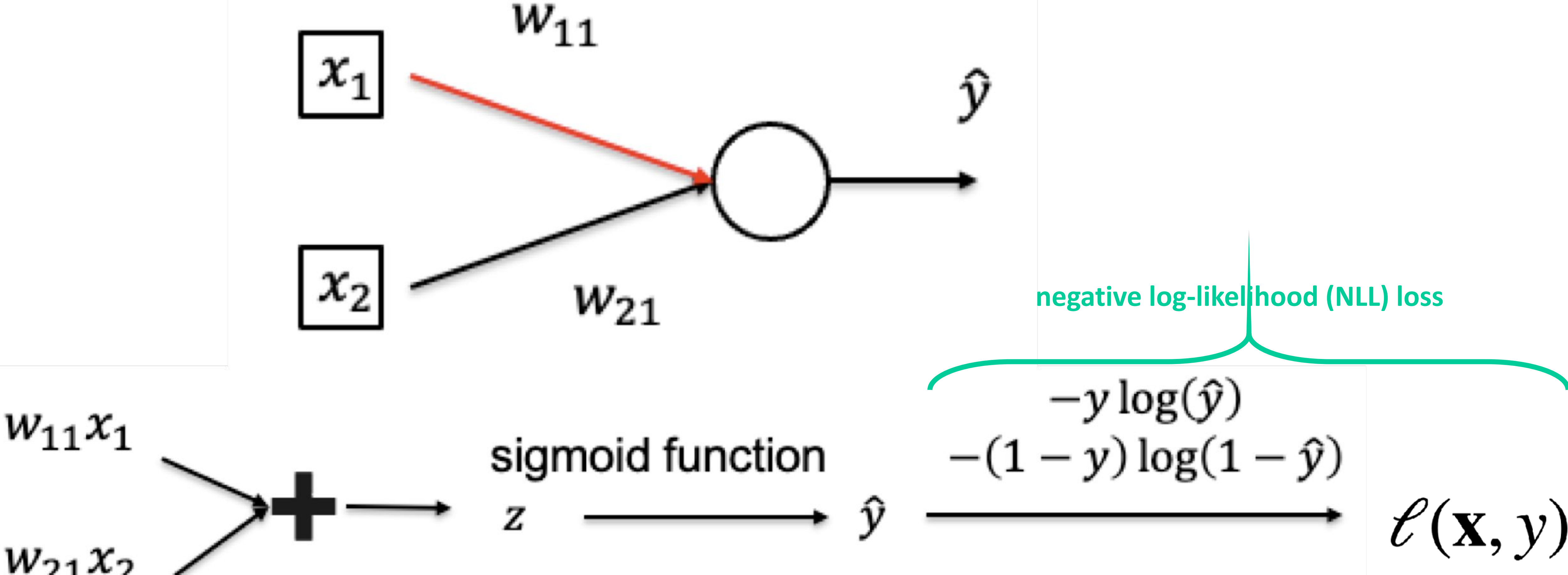
- Input dataset $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$
- Initialize weights
- Until stopping criterion is met:
    - For each training point $(x^{(i)}, y^{(i)})$ do
        - Compute prediction: $\hat{y}^{(i)} = f_w(x^{(i)})$ ← **Forward Pass**
        - Compute loss: $L^{(i)} = L(\hat{y}^{(i)}, y^{(i)})$ ← e.g. negative log-likelihood (NLL) loss $L(\hat{y}, y) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$
        - Compute gradient: $\nabla_w L^{(i)} = \left(\partial_{w_1} L^{(i)}, \partial_{w_2} L^{(i)}, \ldots, \partial_{w_m} L^{(i)}\right)^{\top}$ ← **Backward Pass**
        - Update weights: $w \leftarrow w - \alpha \nabla_w L^{(i)}$ ← SGD step

Computing Gradients

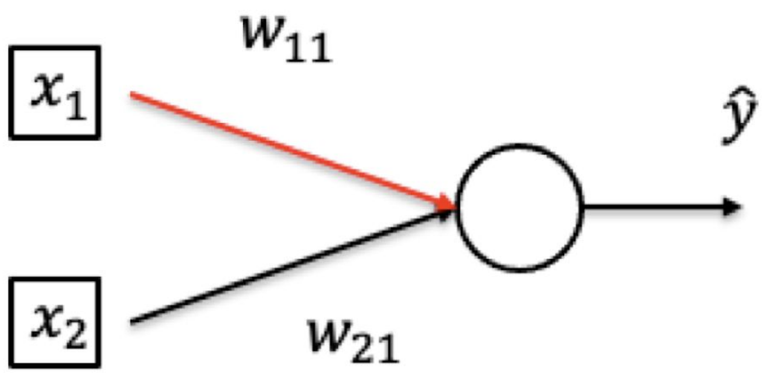


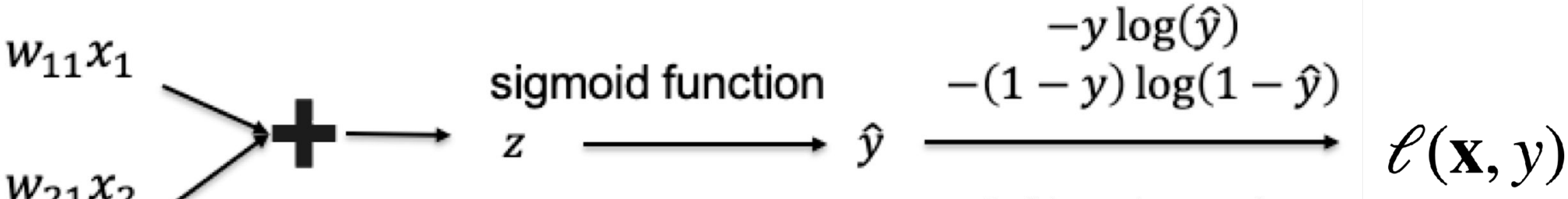
Want to compute $\dfrac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

Computing Gradients



Computing Gradients



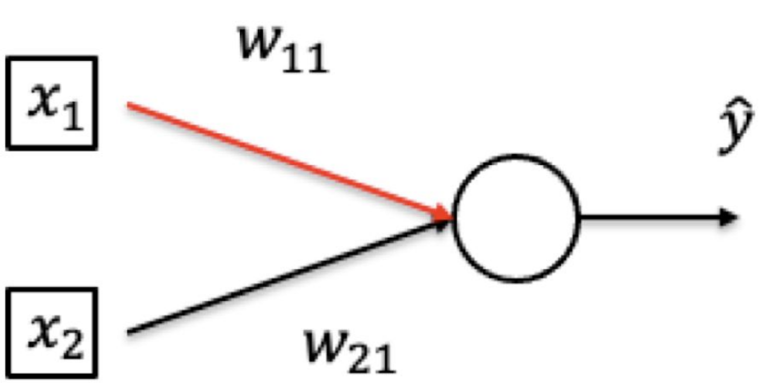


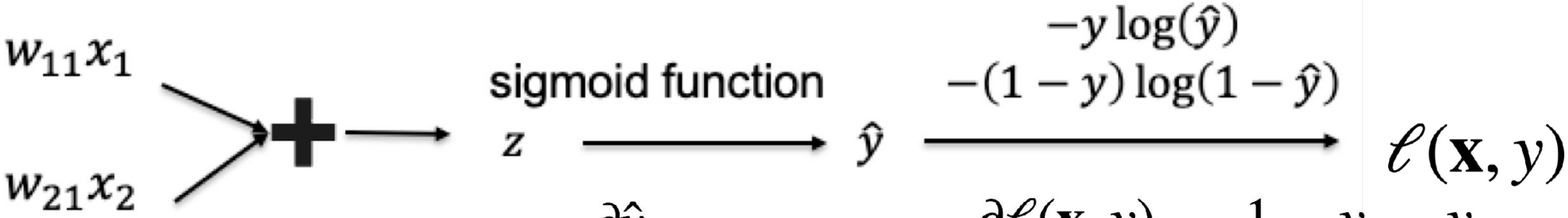
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

By chain rule:

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

Computing Gradients





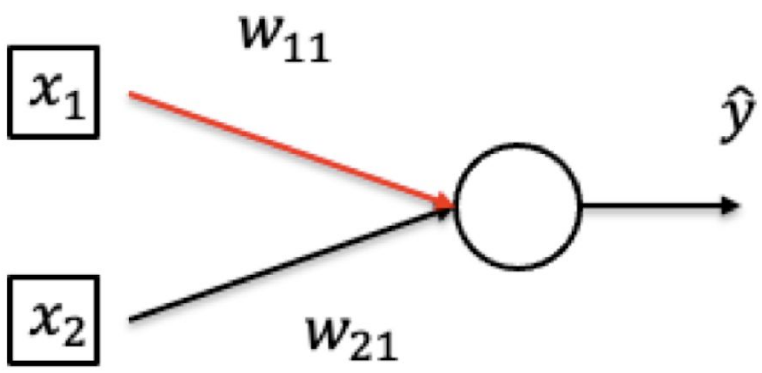
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z)$$

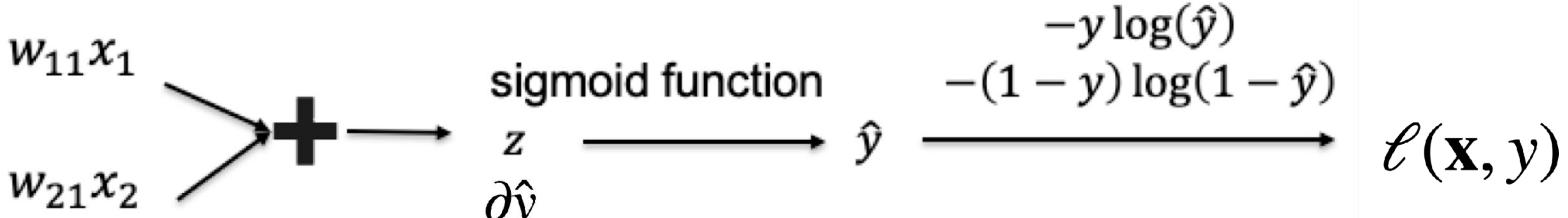
$$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

By chain rule:

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

Computing Gradients

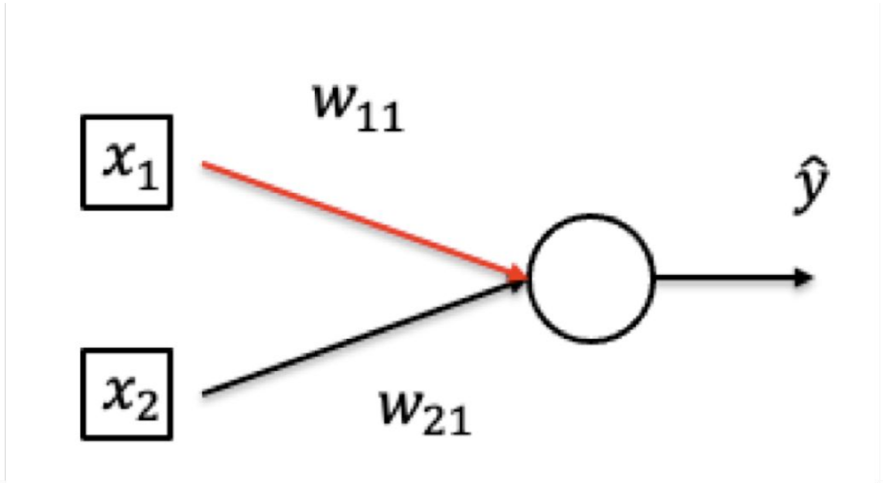


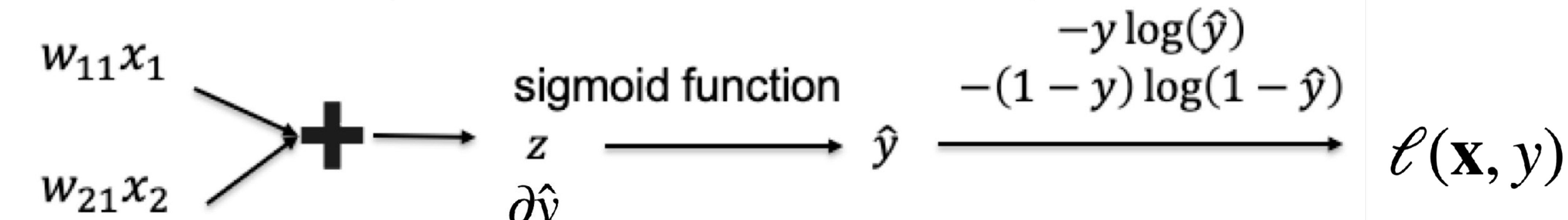


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Computing Gradients

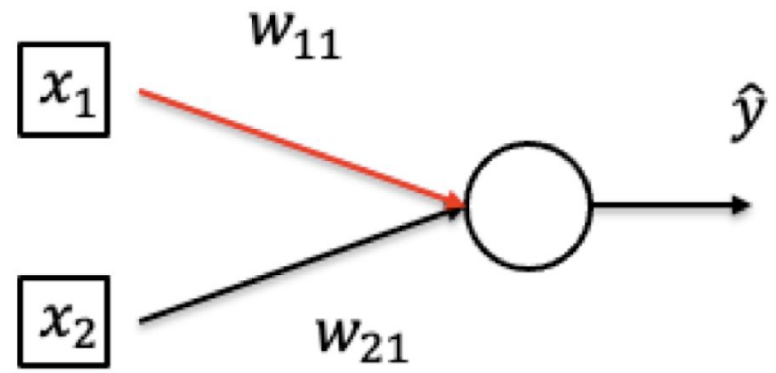


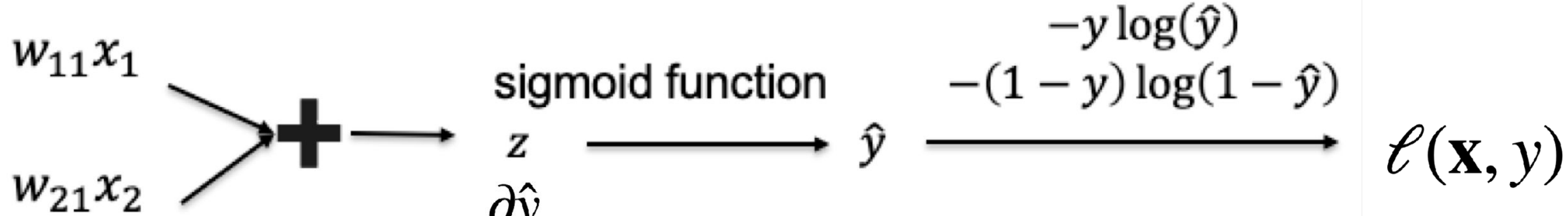


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

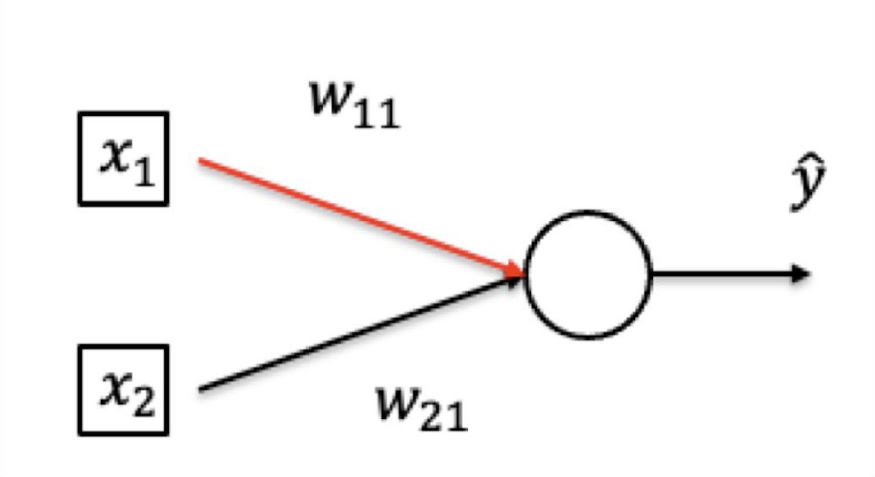


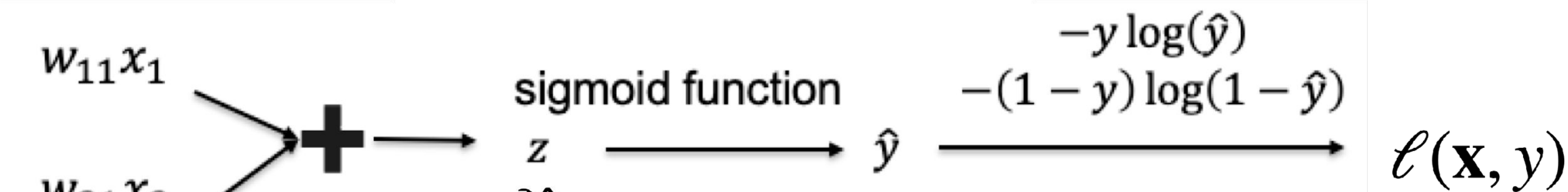


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Computing Gradients



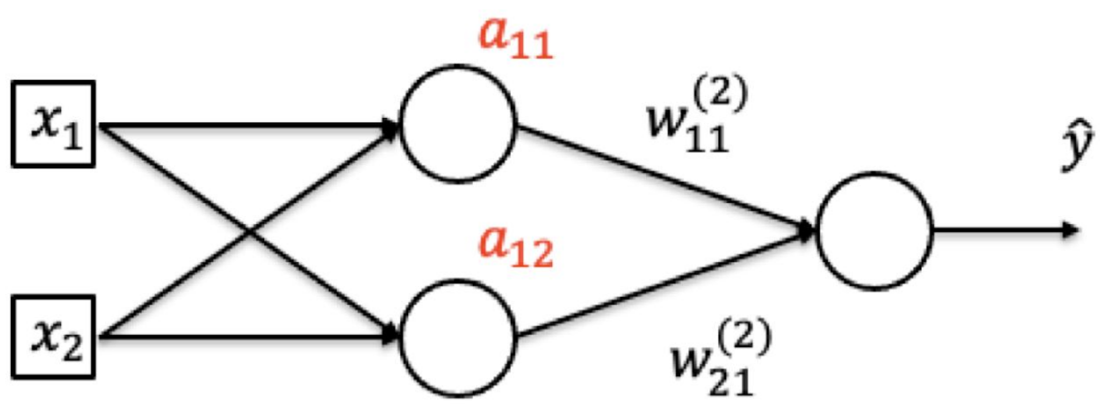


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

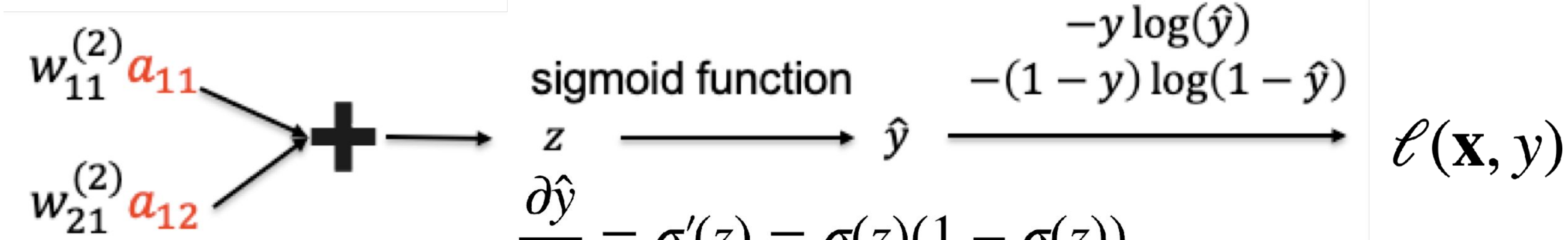
By chain rule:

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

Computing Gradients: More Layers



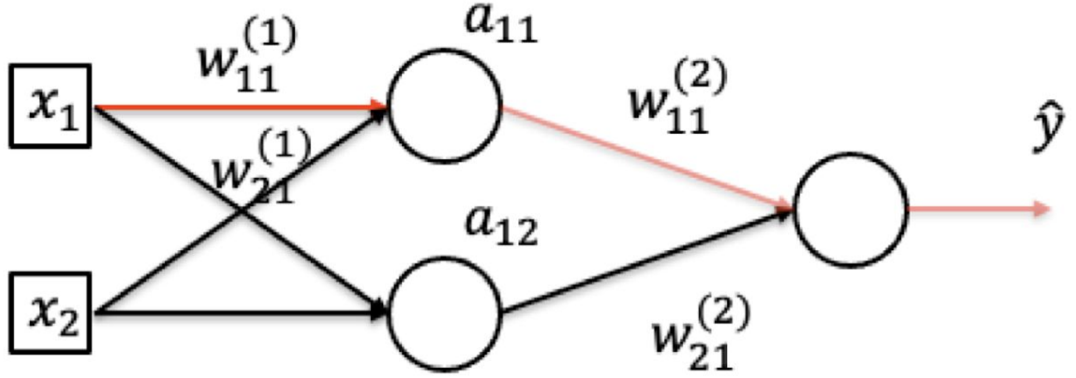
Make it deeper

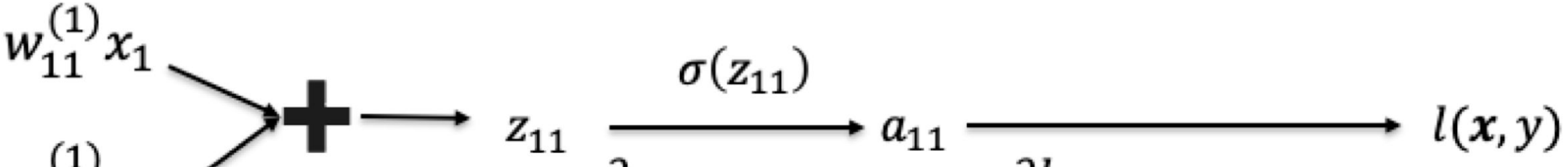


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Computing Gradients: More Layers

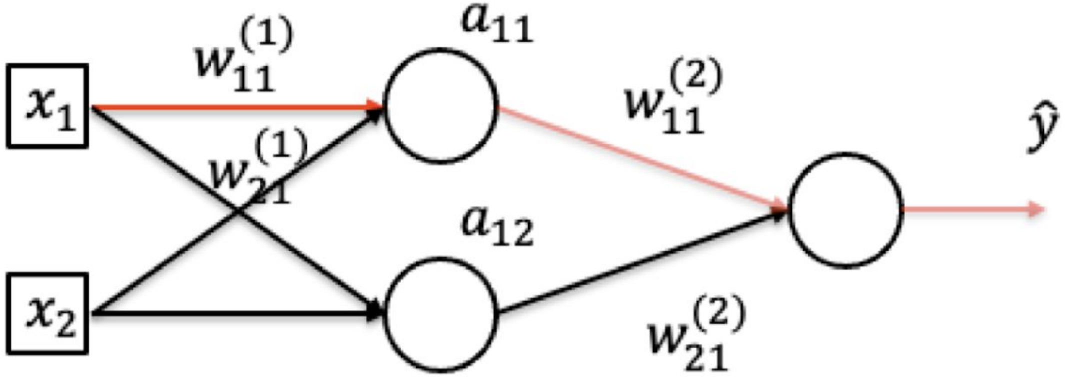


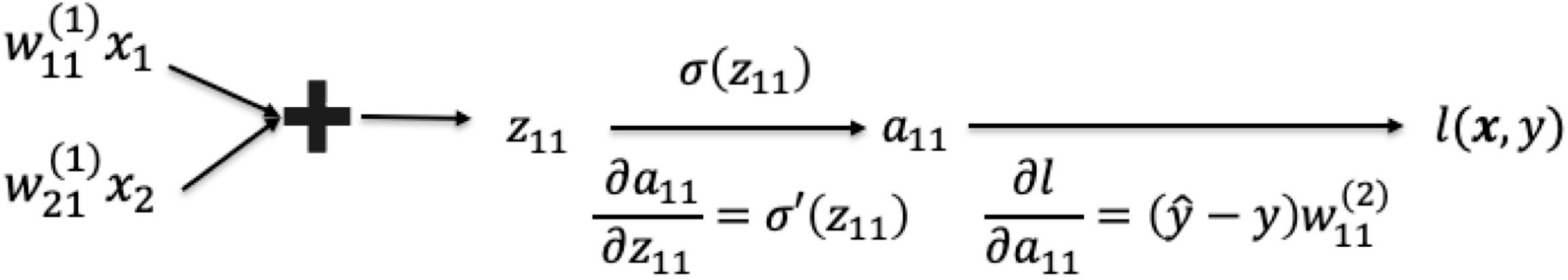


$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w^{(2)}_{11}$$

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

Computing Gradients: More Layers

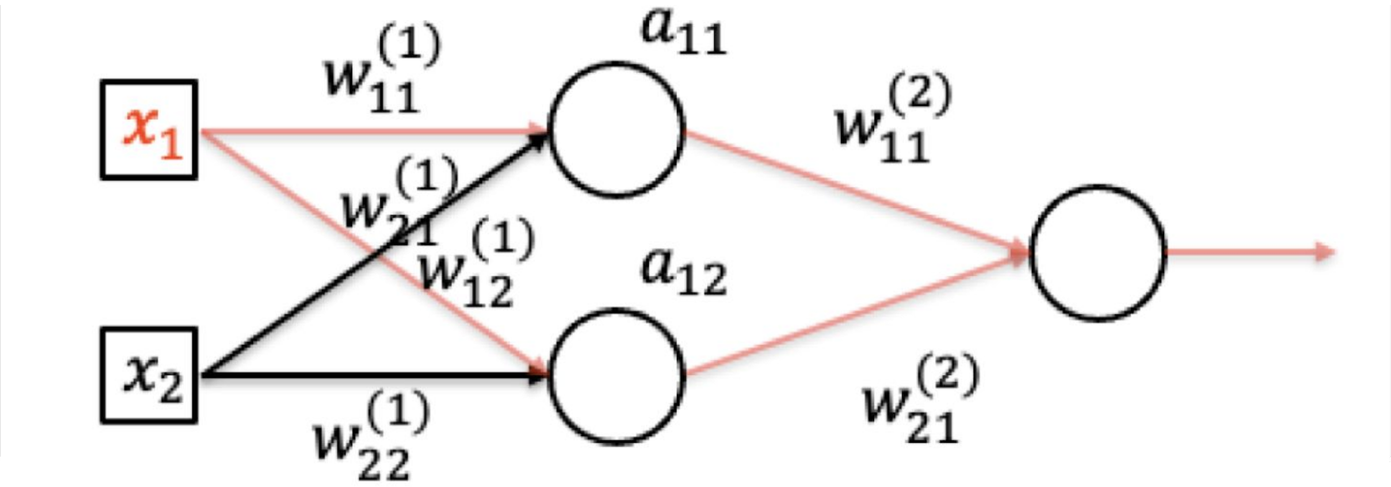




$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

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



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

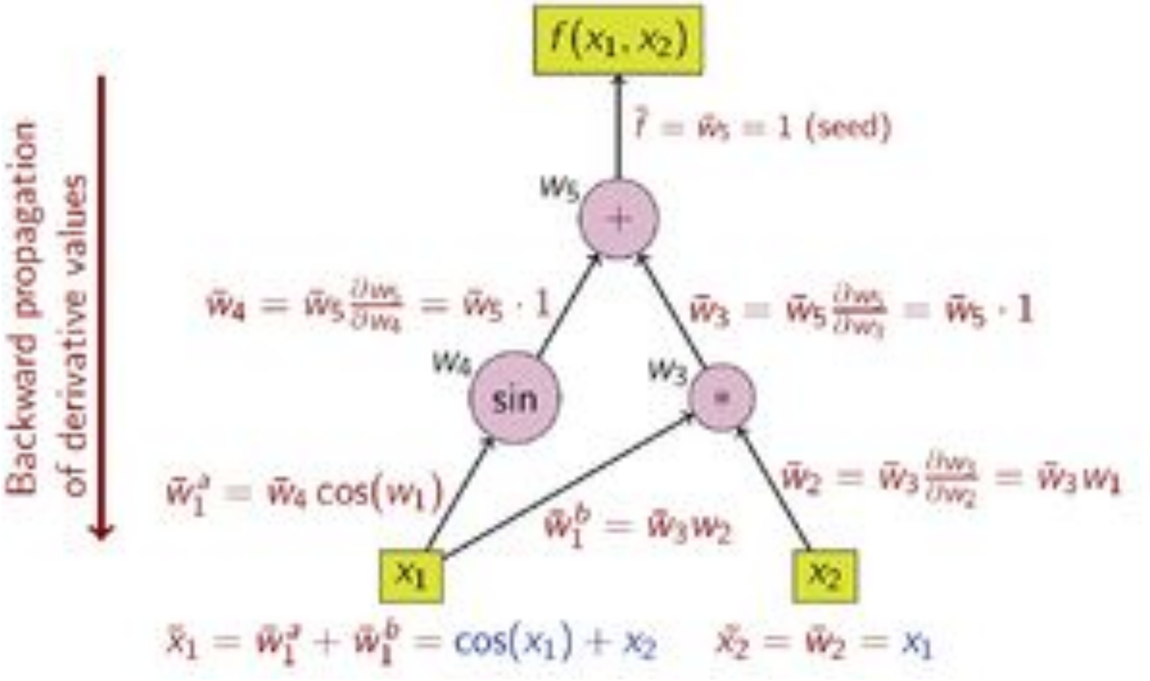
$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

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

- So to compute derivative w.r.t specific weights we **propagate** loss information **back** through the network
- Today we do this by automatic differentiation (**autodiff**) for arbitrarily complex computation graphs
- Go backwards from top to bottom, recursively computing gradients


$f(x_1, x_2)$
$\bar{f} = \bar{w}_5 = 1$ (seed)
w_5
+
$\bar{w}_4 = \bar{w}_5 \frac{\partial w_5}{\partial w_4} = \bar{w}_5 \cdot 1$
$\bar{w}_3 = \bar{w}_5 \frac{\partial w_5}{\partial w_3} = \bar{w}_5 \cdot 1$
w_4
sin
w_3
*
Backward propagation of derivative values
$\bar{w}_1^a = \bar{w}_4 \cos(w_1)$
$\bar{w}_1^b = \bar{w}_3 w_2$
$\bar{w}_2 = \bar{w}_3 \frac{\partial w_3}{\partial w_2} = \bar{w}_3 w_1$
x_1
x_2
$\bar{x}_1 = \bar{w}_1^a + \bar{w}_1^b = \cos(x_1) + x_2$
$\bar{x}_2 = \bar{w}_2 = x_1$

Wiki

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Misha Khodak, 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, Josiah Hanna