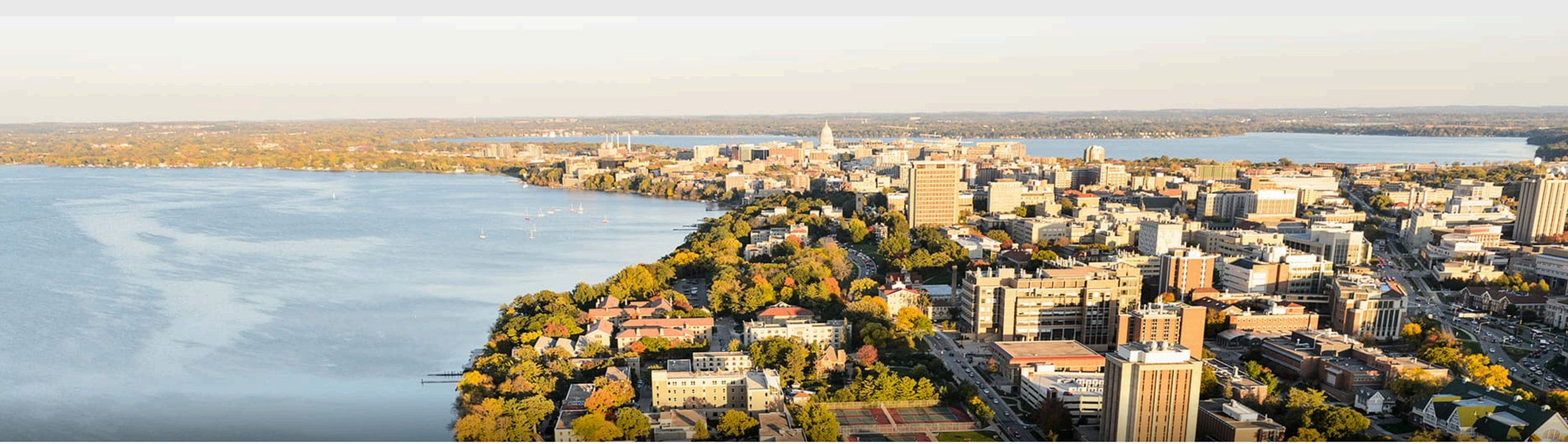


CS 540 Introduction to Artificial Intelligence

Neural Networks (II)

Yudong Chen
University of Wisconsin-Madison

Oct 21, 2021

Slides created by Sharon Li [modified by Yudong Chen]

Announcement

- Midterm: Oct 28 (Thursday)
  - Details can be found on https://piazza.com/class/ktahk2n2c6wkp?cid=161

Today's outline

- Single-layer Perceptron: Recap
- Multi-layer Perceptron
  - Single output
  - Multiple output
- How to train neural networks
  - Gradient descent

Last lecture:
single layer perceptron.

Today:
multi-layer perceptron.

Next Tuesday: → many layers.
deep neural networks.

After midterm:
specific types of DNN applications.

Review: Perceptron

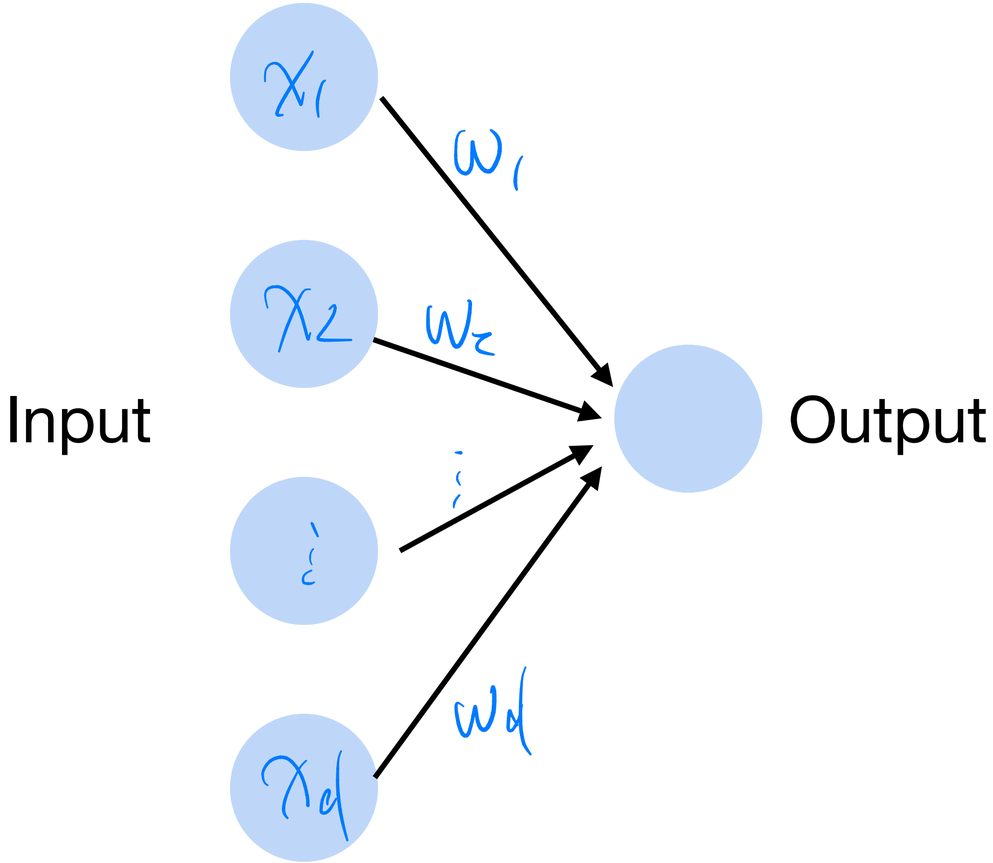
- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$f(\mathbf{x}) = \sigma\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right)$$

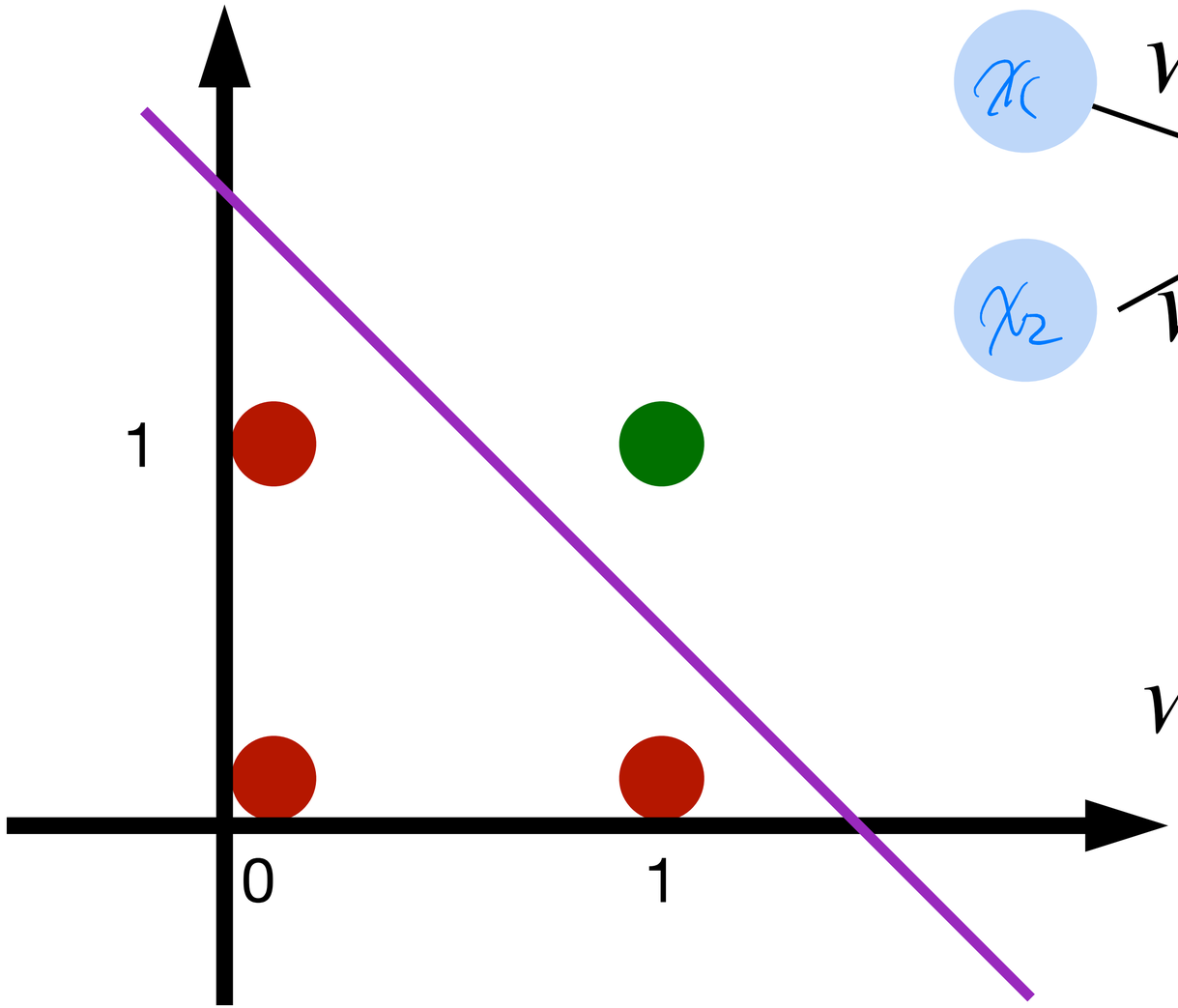
$$\sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

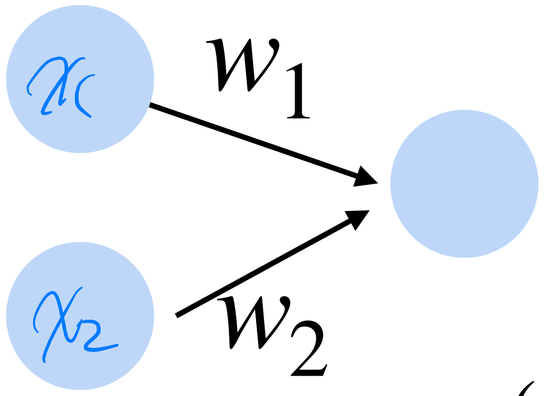
Activation function

Cats vs. dogs?



Perceptron can learn AND function



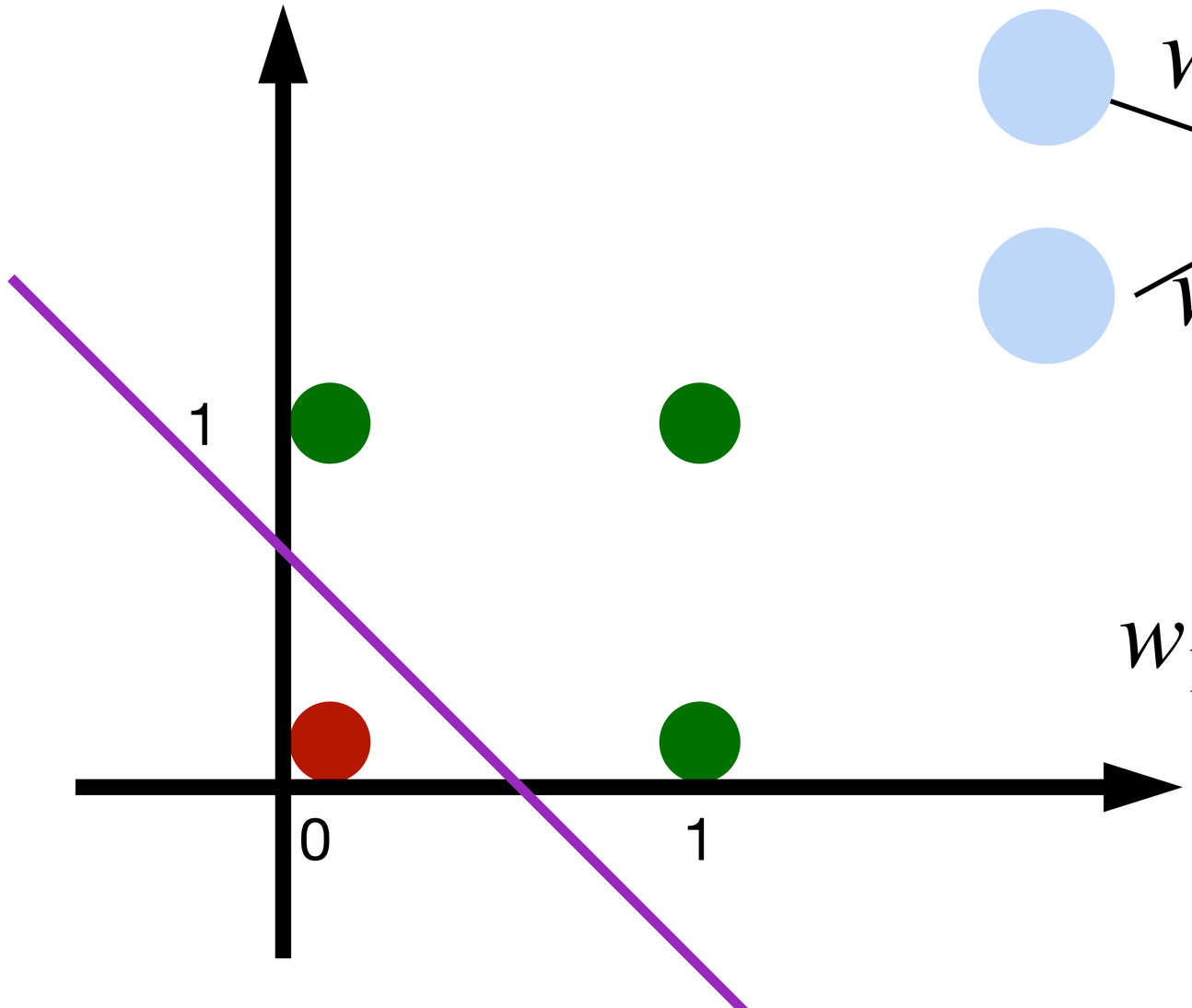


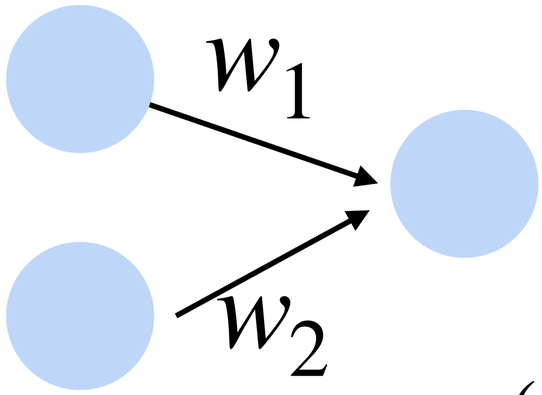
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

$$\sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -1.5$$

Perceptron can learn OR function



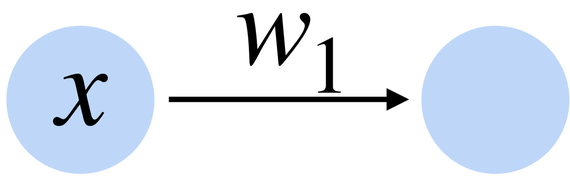


Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

$$\sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -0.5$$

Perceptron can learn NOT function

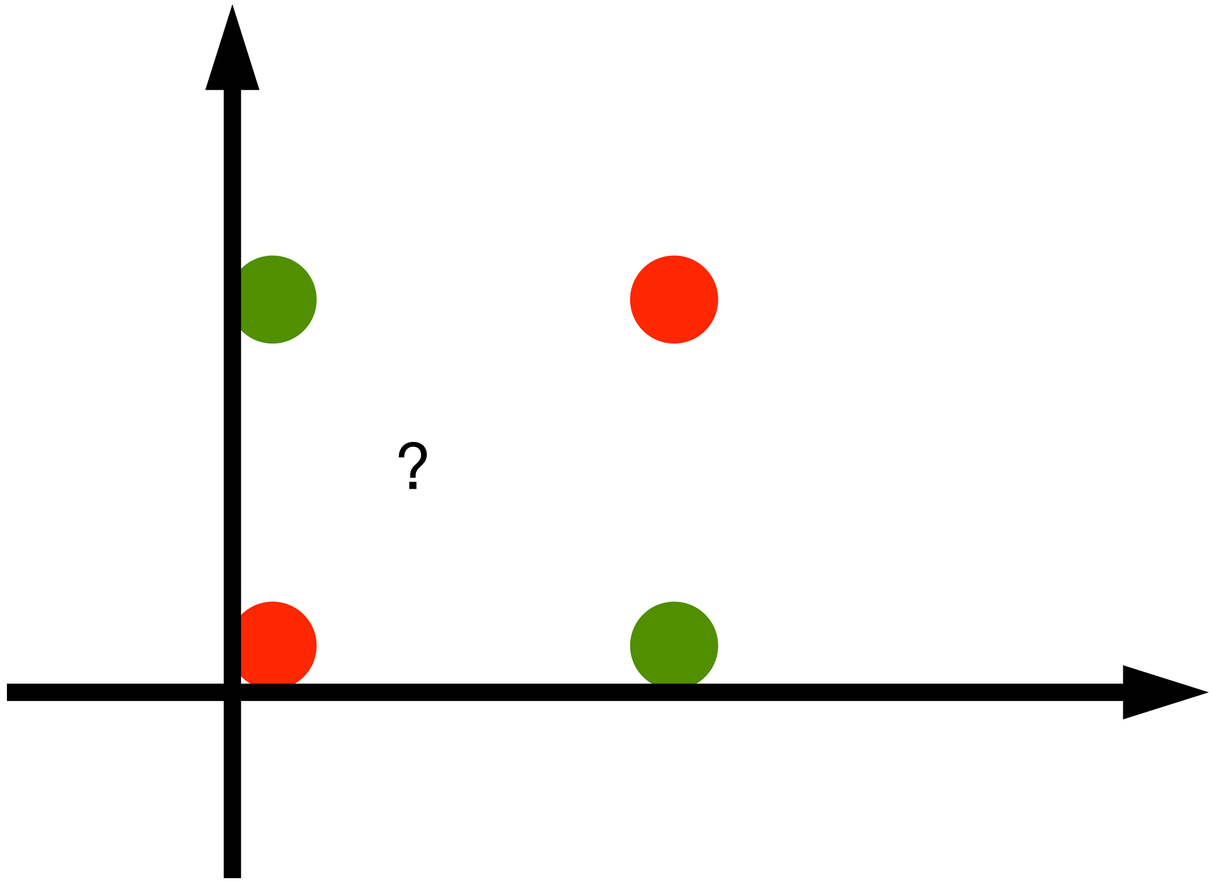


$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = -1, b = 0.5$$

The limited power of a single neuron

The perceptron cannot learn an **XOR** function
(neurons can only generate linear separators)



$$x_1 = 1, x_2 = 1, y = 0$$

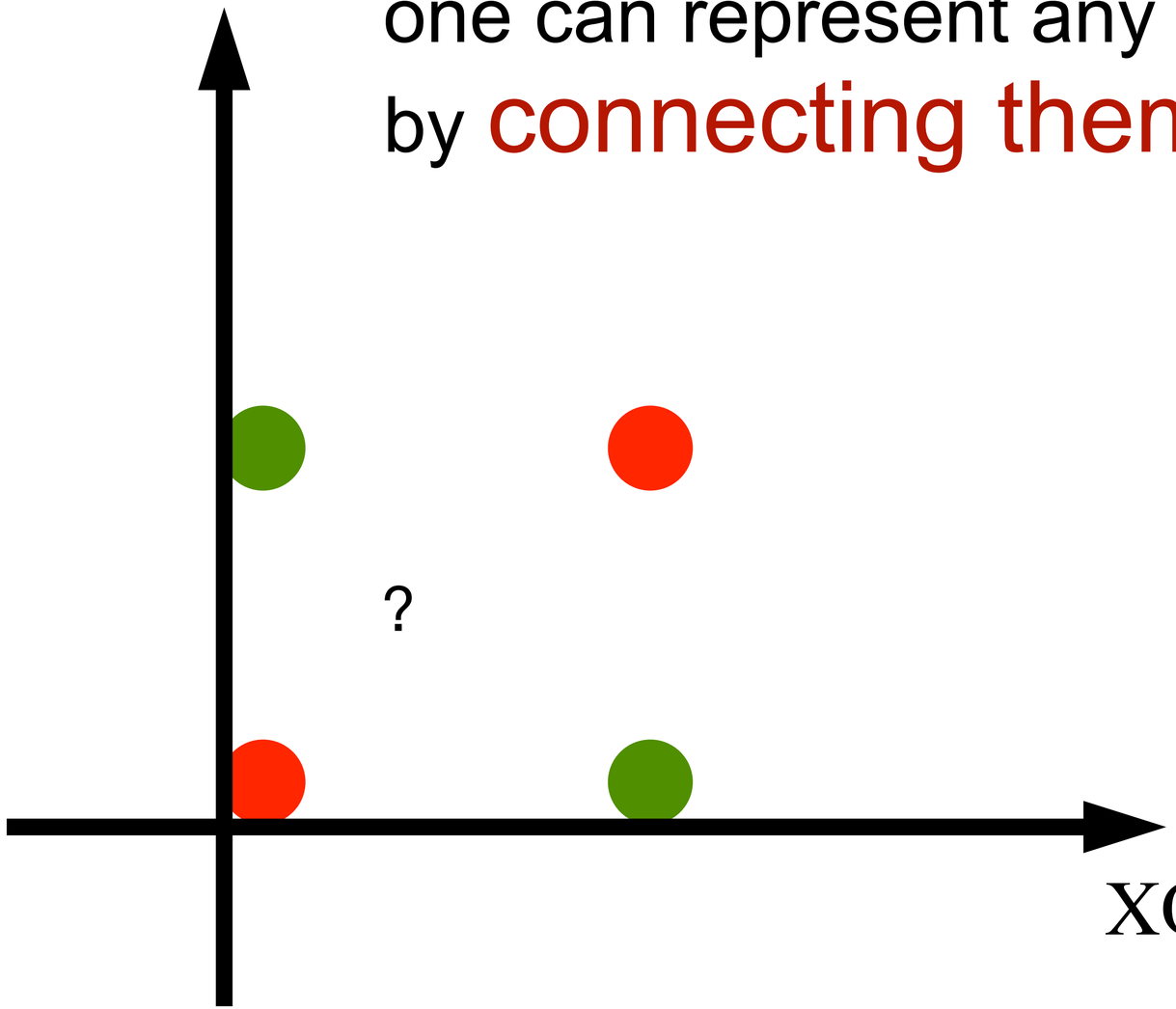
$$x_1 = 1, x_2 = 0, y = 1$$

$$x_1 = 0, x_2 = 1, y = 1$$

$$x_1 = 0, x_2 = 0, y = 0$$

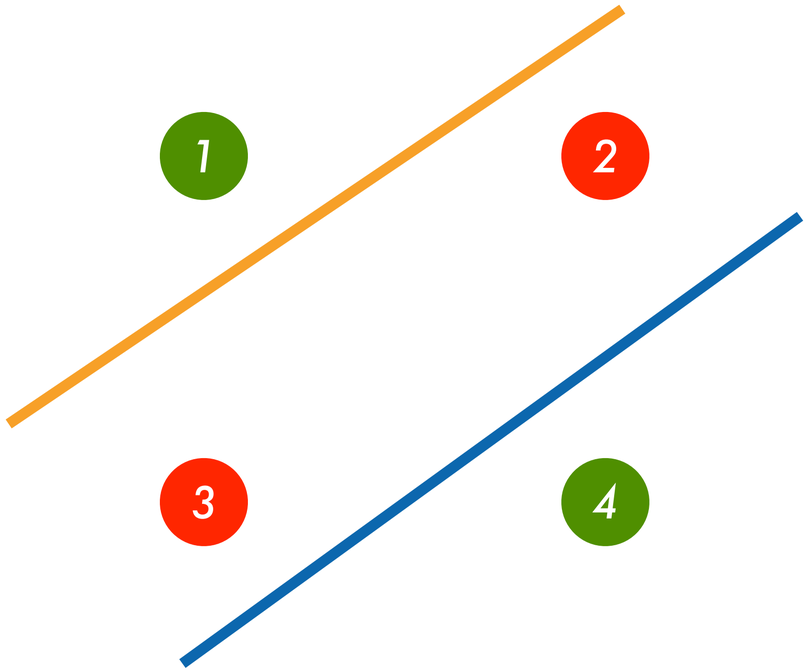
Overcoming the limit of a single neuron

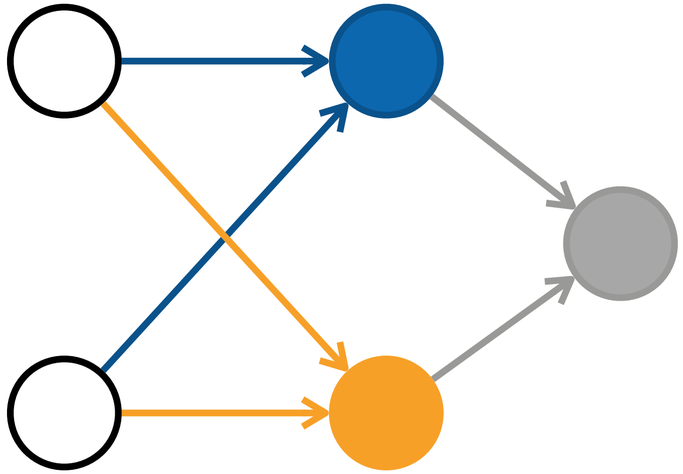
If one can represent AND, OR, NOT,
one can represent any logic circuit (including XOR),
by connecting them



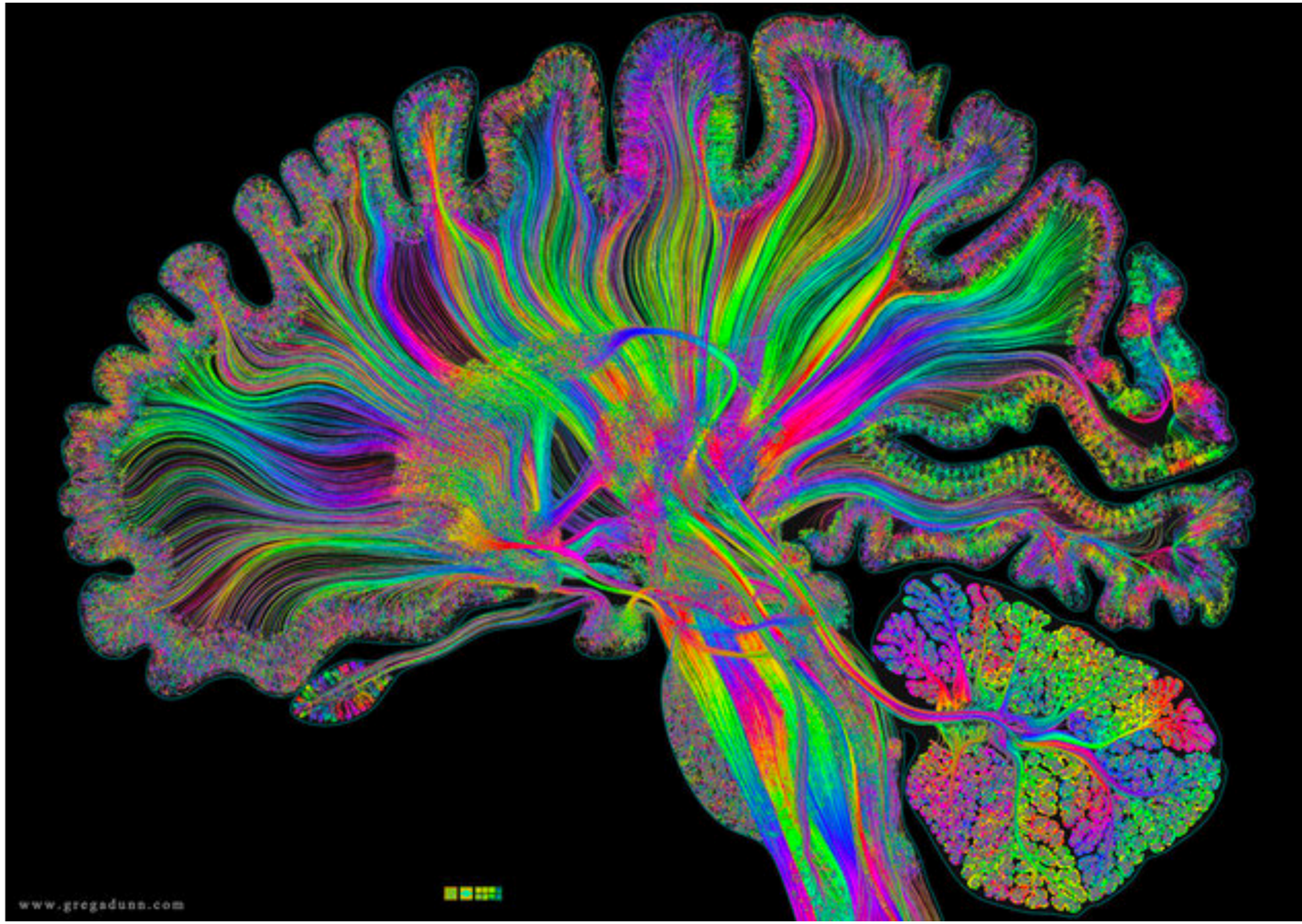
$$XOR(x1, x2) = (x1 \wedge \neg x2) \vee (\neg x1 \wedge x2)$$

Learning XOR



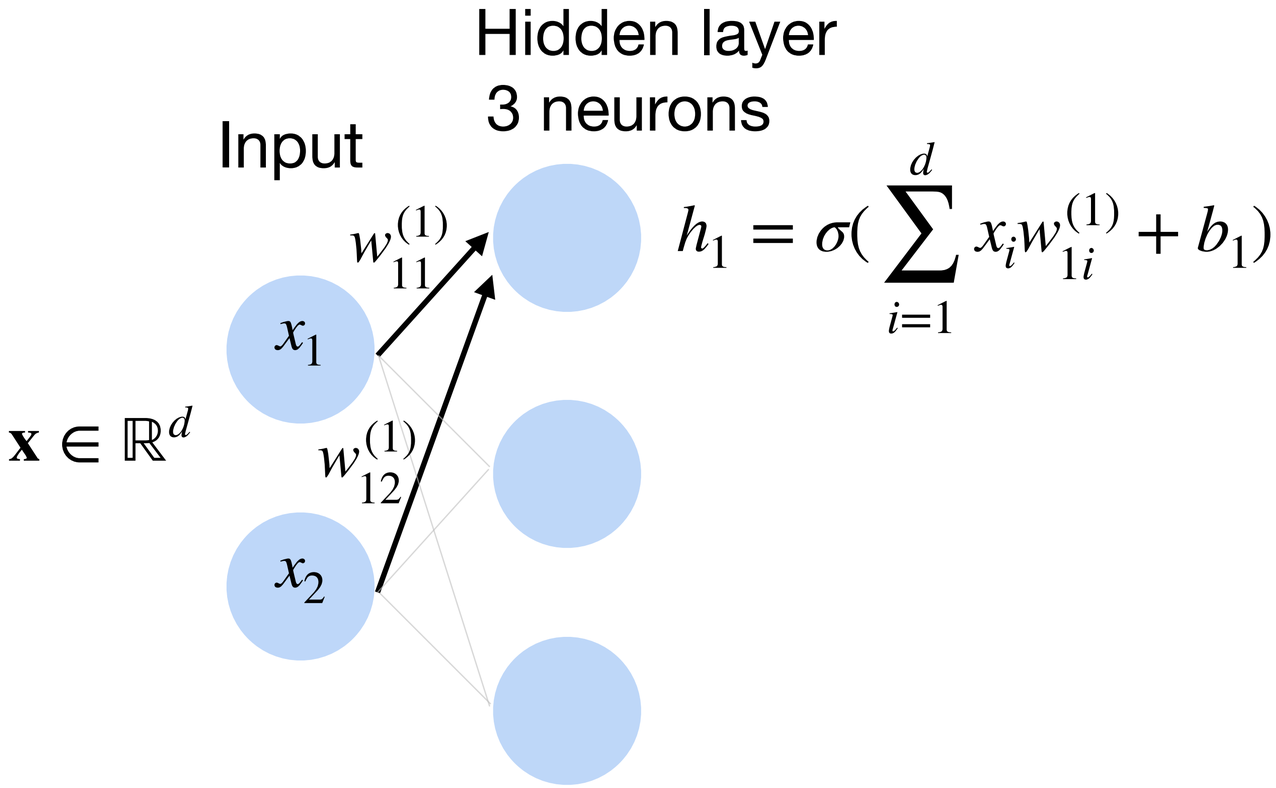


Multilayer Perceptron



Multi-layer perceptron: Example

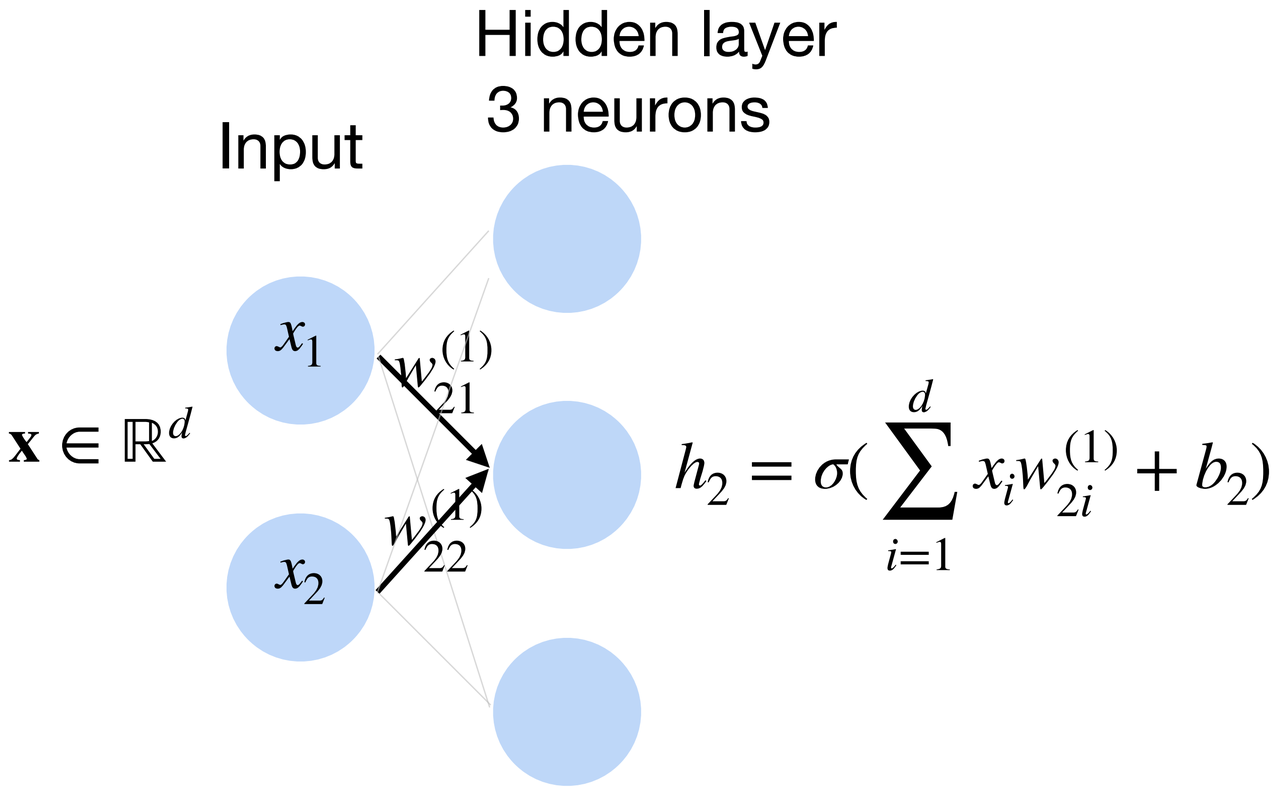
- Example: 1 hidden layer, 1 output layer (depth = 2)



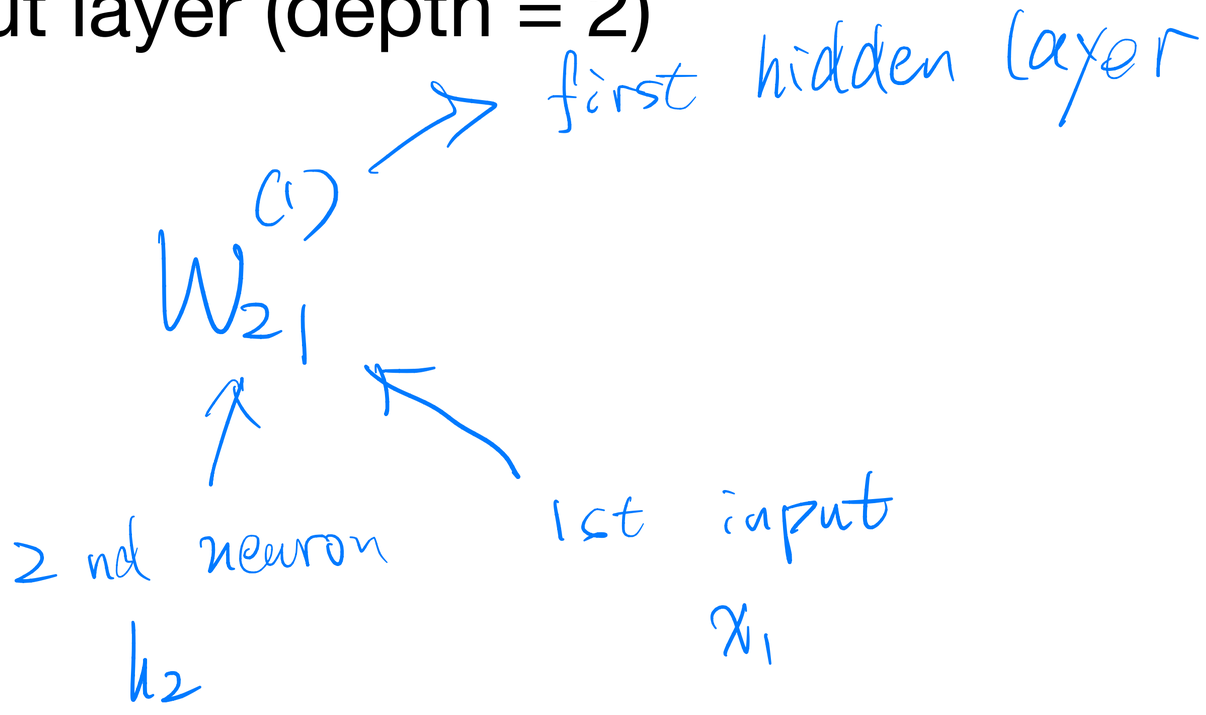
$$h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1)$$

Multi-layer perceptron: Example

- Example: 1 hidden layer, 1 output layer (depth = 2)

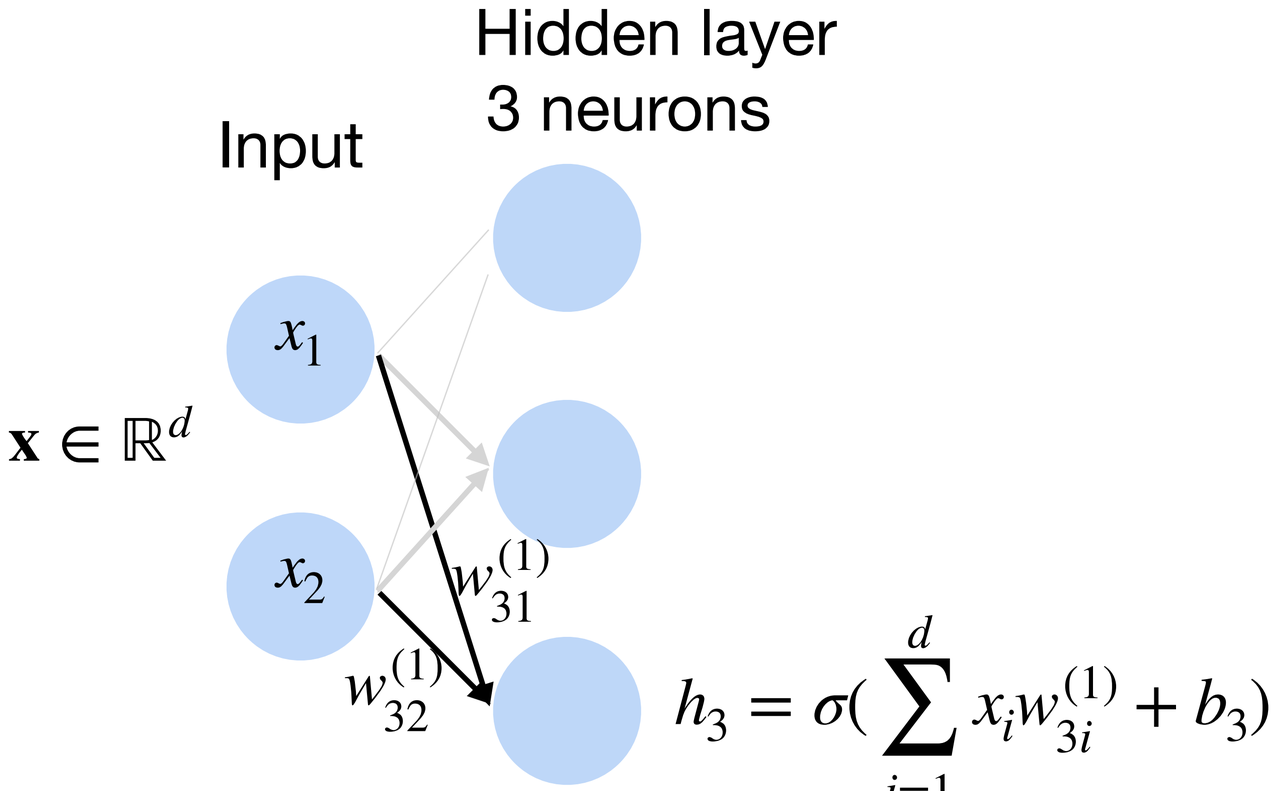


$$h_2 = \sigma\left(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2\right)$$



Multi-layer perceptron: Example

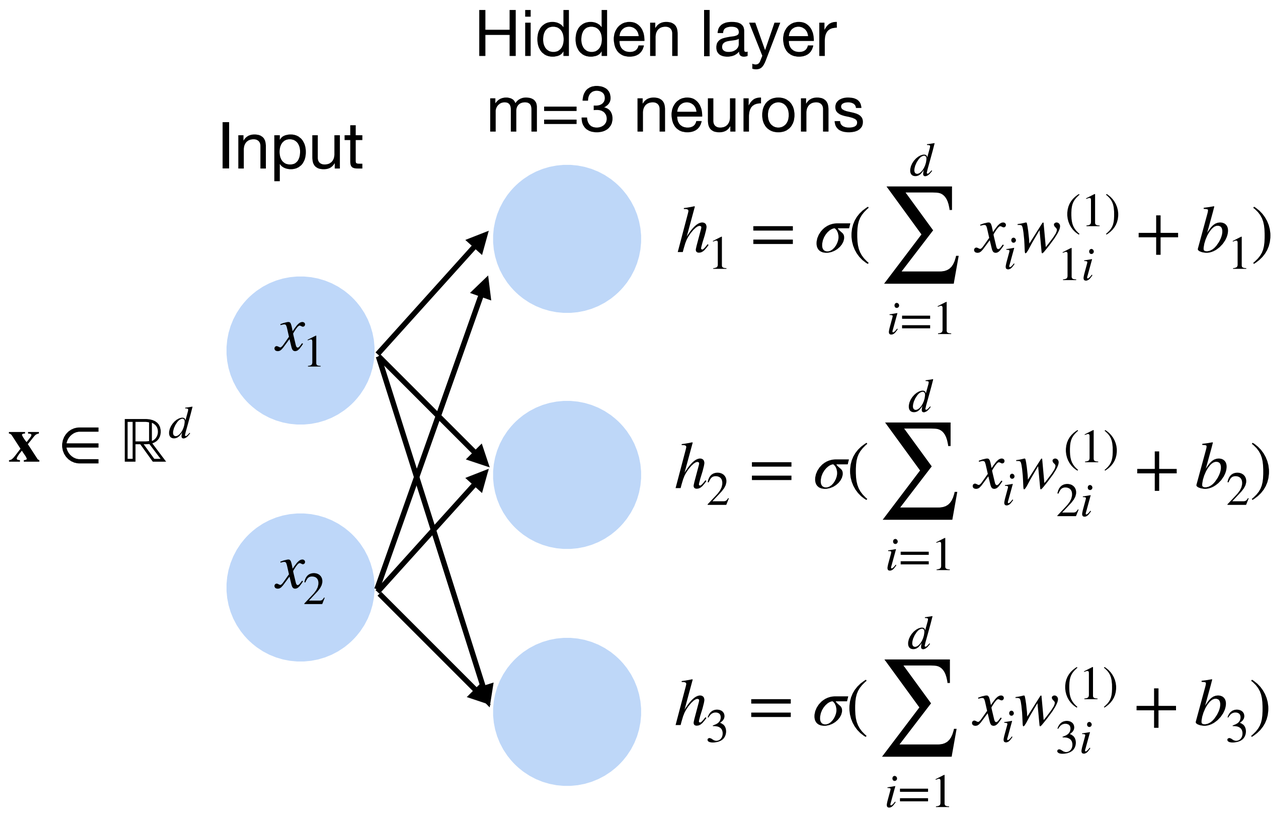
- Example: 1 hidden layer, 1 output layer (depth = 2)



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

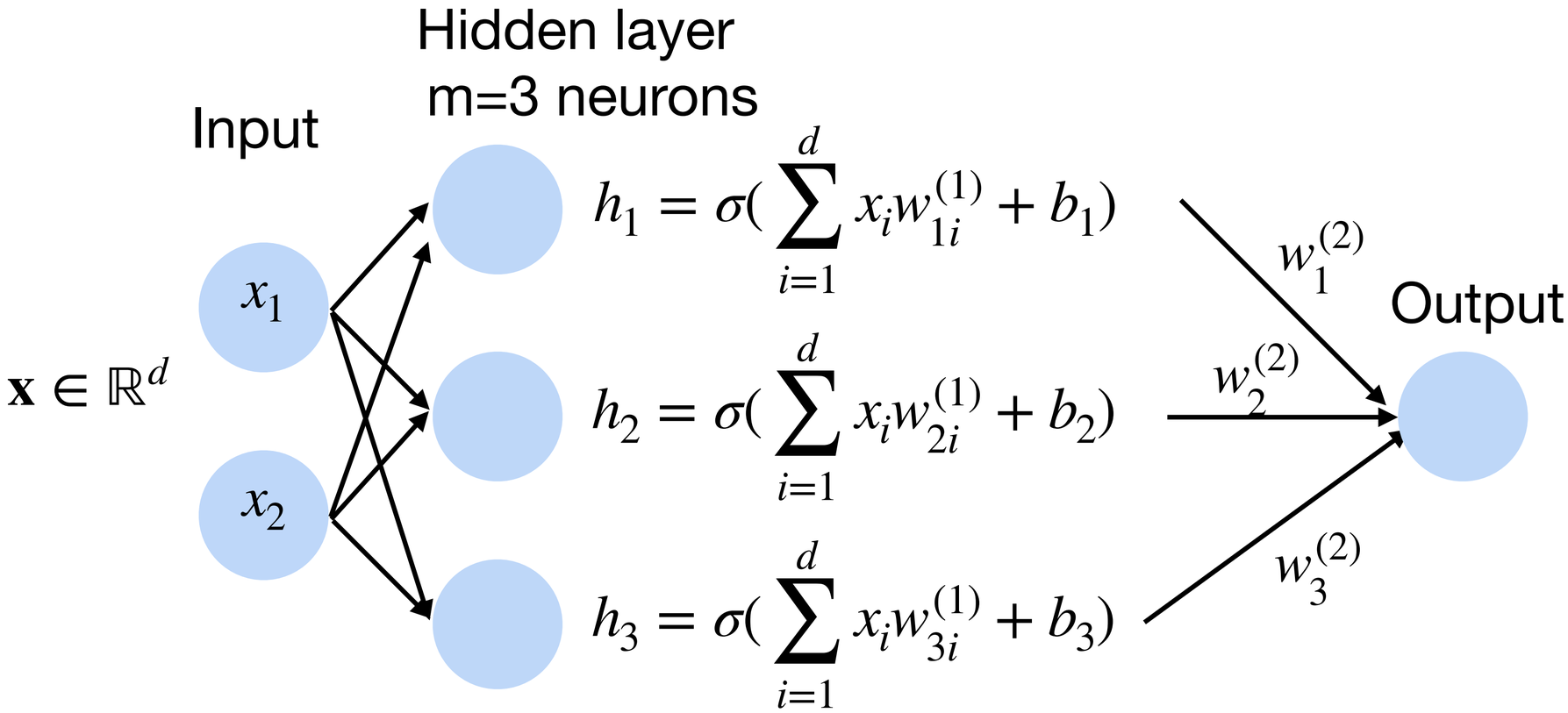
Multi-layer perceptron: Example

- Example: 1 hidden layer, 1 output layer (depth = 2)



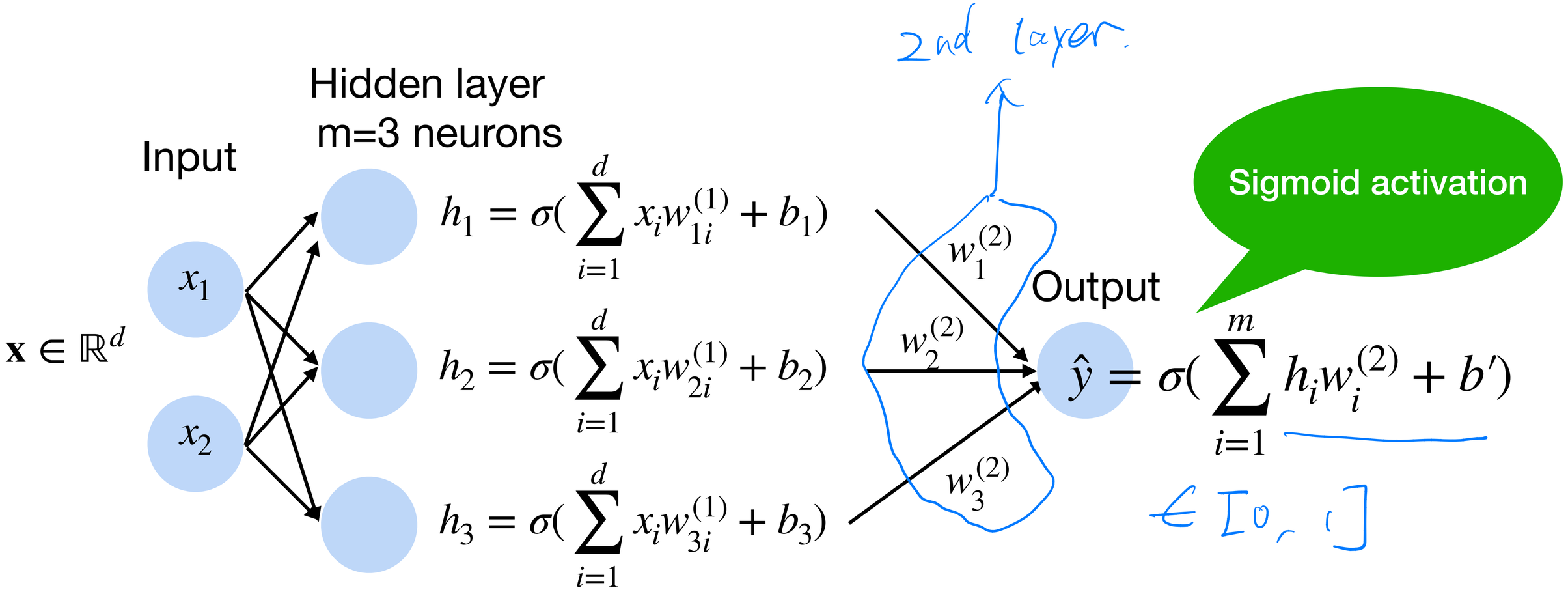
Multi-layer perceptron: Example

- Example: 1 hidden layer, 1 output layer (depth = 2)



Multi-layer perceptron: Example

- Example: 1 hidden layer, 1 output layer (depth = 2)



Multi-layer perceptron: Matrix Notation

- Input $\mathbf{x} \in \mathbb{R}^d$
- Parameters for hidden layer

$$\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$$

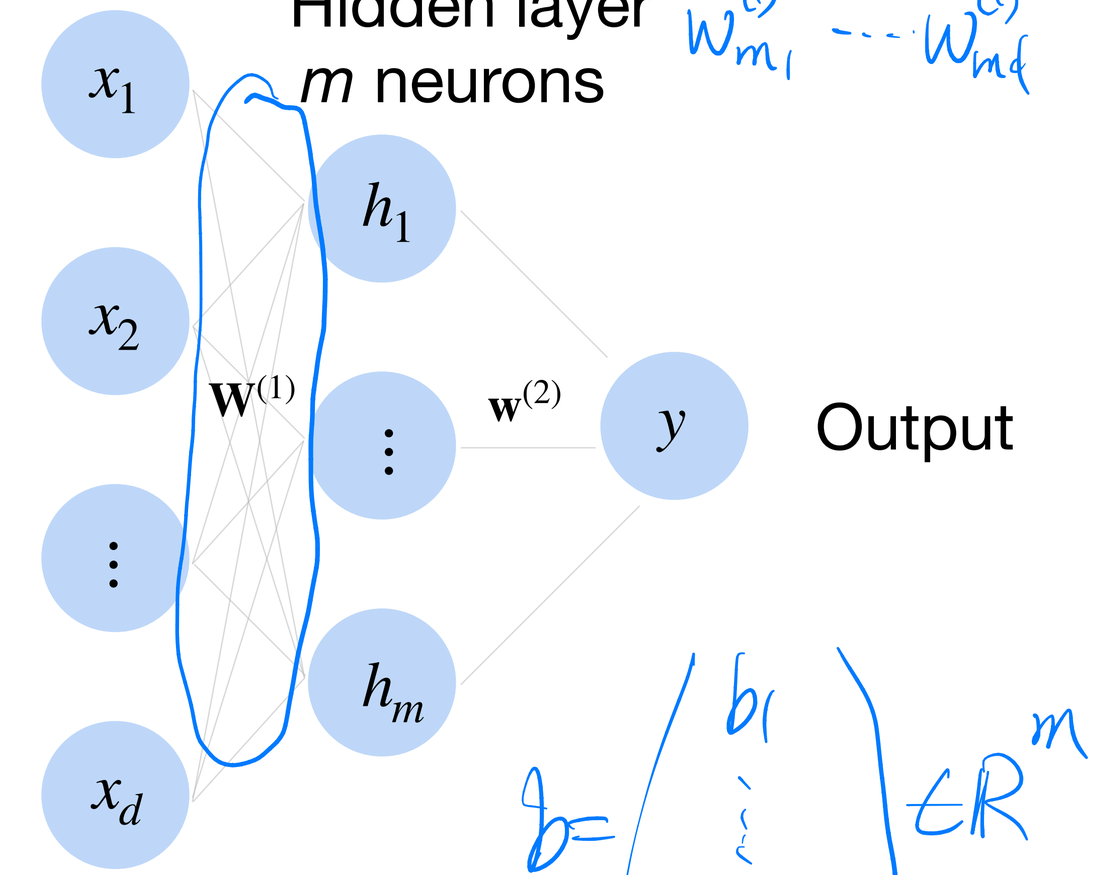
- Output from hidden layer

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$

$$\mathbf{h} \in \mathbb{R}^m$$

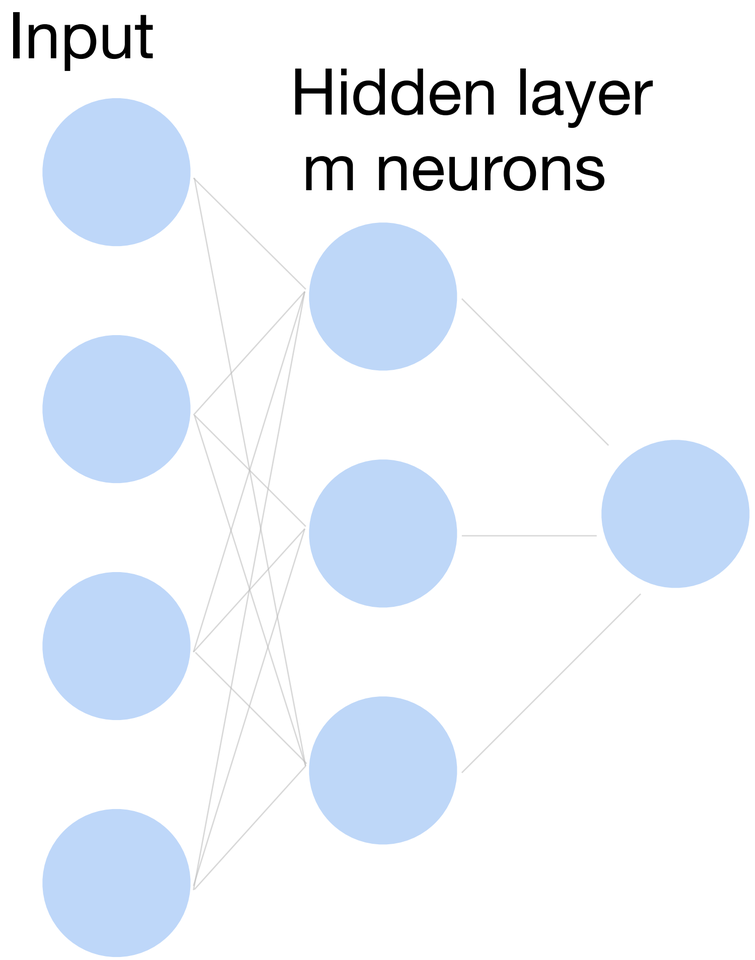
- Final output

$$y = \sigma(\mathbf{w}^{(2)T}\mathbf{h} + b')$$



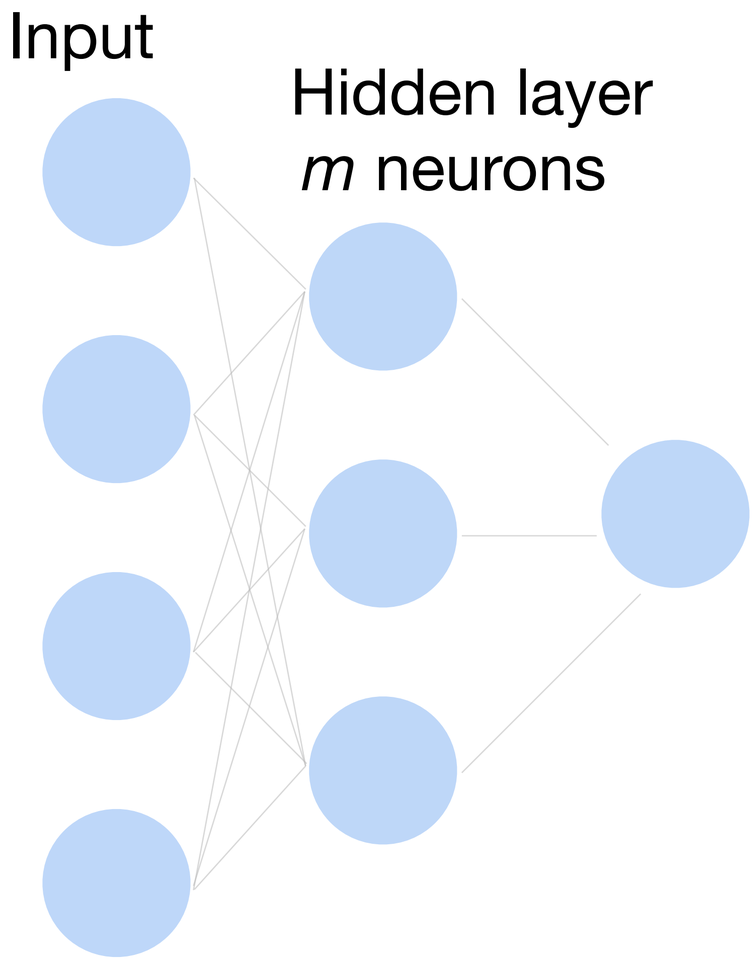
Multi-layer perceptron

$\sigma(z) = z$



Why do we need an a nonlinear activation?

Multi-layer perceptron



Why do we need an a nonlinear activation?

$$\mathbf{h} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}$$

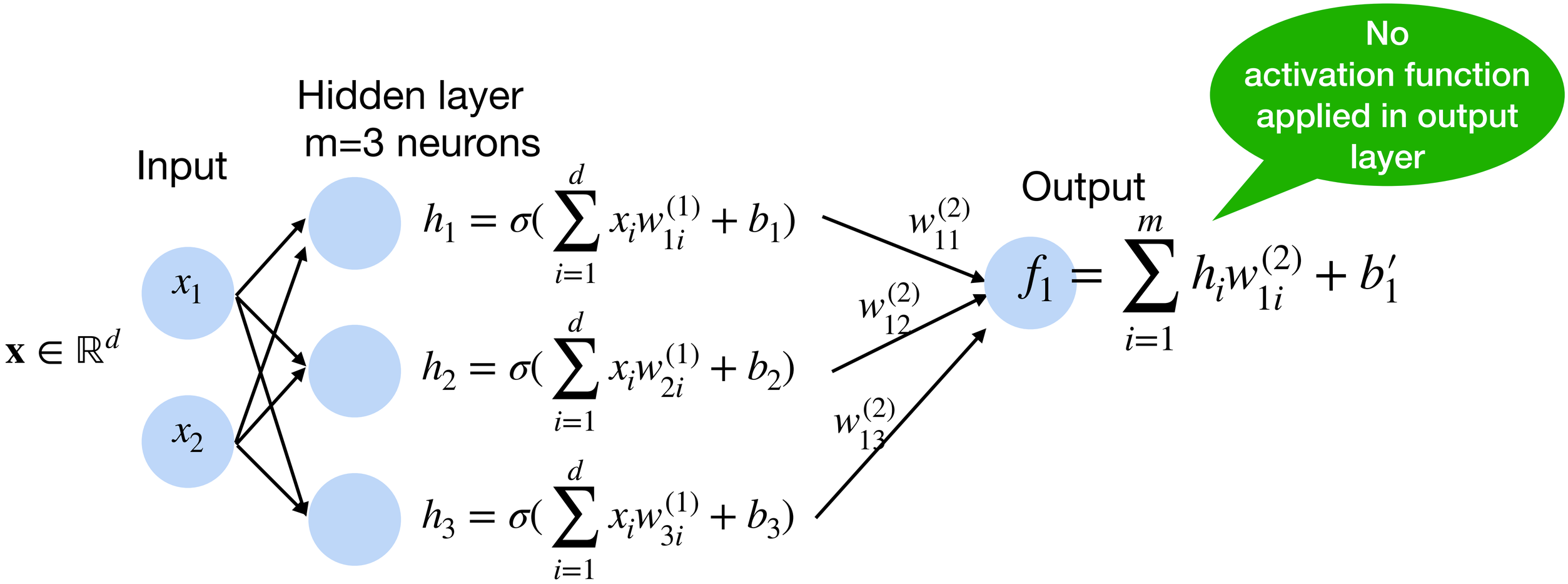
$$f(x) = \mathbf{w}^{(2)\top}\mathbf{h} + b'$$

.

hence $f(x) = \mathbf{w}^{(2)\top}\mathbf{W}^{(1)}\mathbf{x} + b'$

Neural network for *k*-way classification

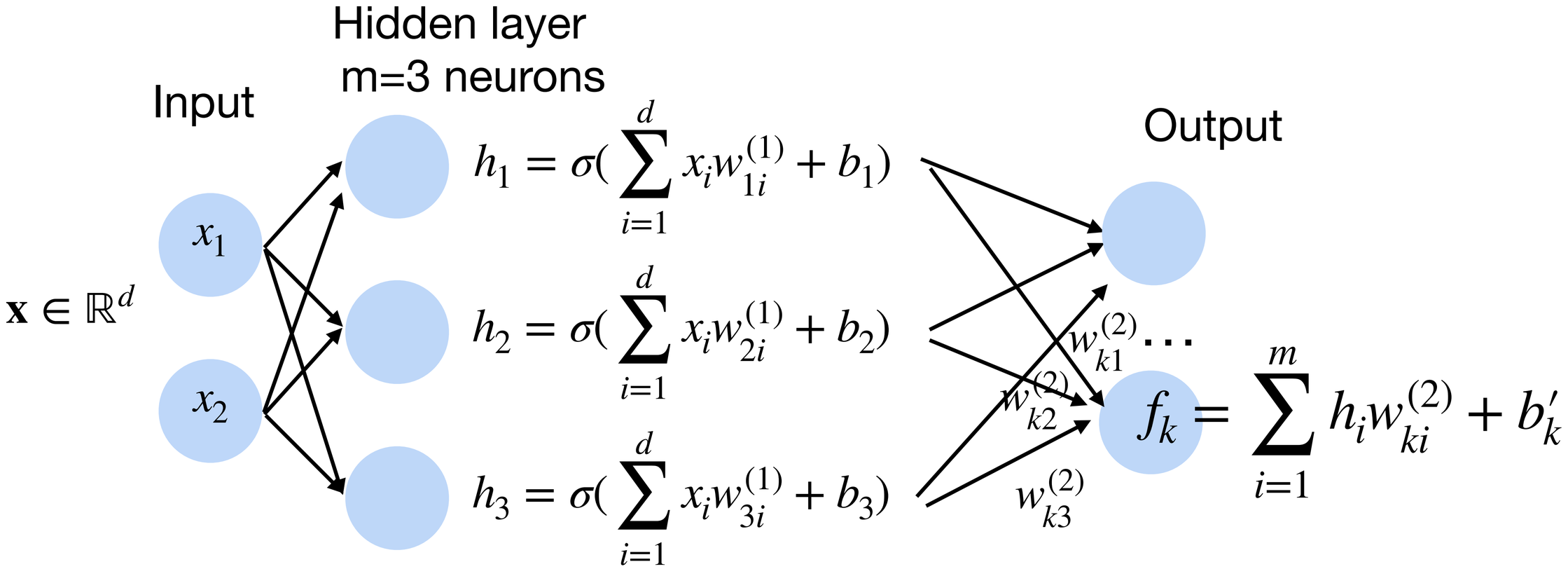
- *k* outputs in the final layer



Neural network for *k*-way classification

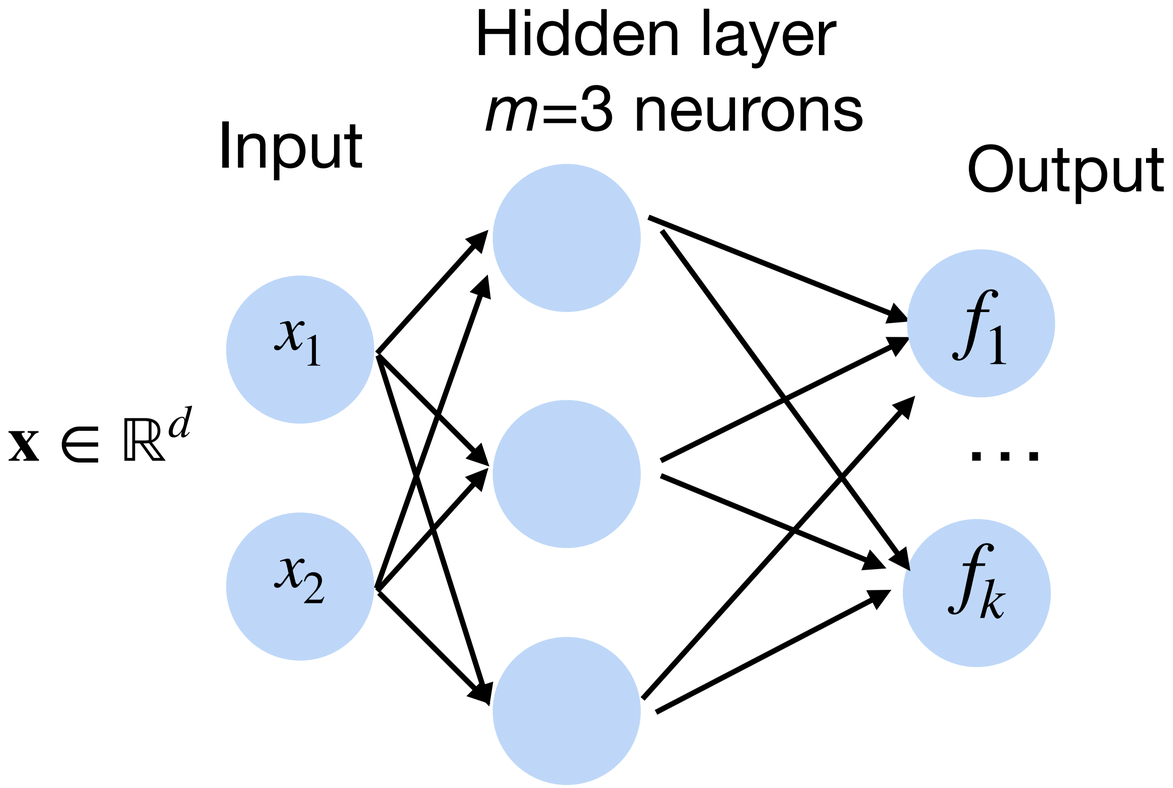
- *k* outputs units in the final layer

Multi-class classification (e.g., ImageNet with *k*=1000)



Softmax

Turns outputs *f* into probabilities (sum up to 1 across *k* classes)



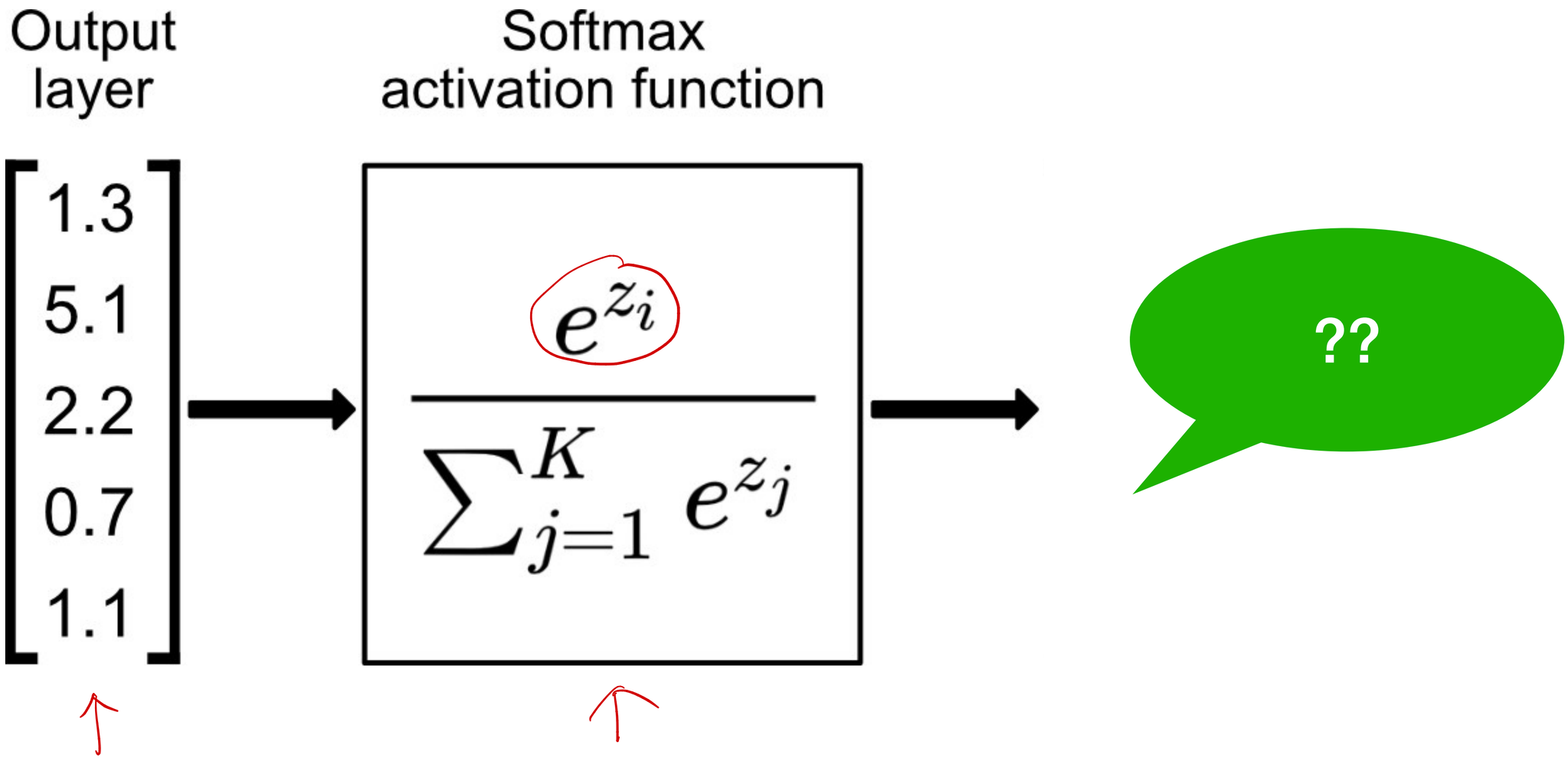
$$p(y = i \mid \mathbf{x}) = \text{softmax}(\vec{f}) = \frac{\exp[f_i(x)]}{\sum_{j=1}^{k} \exp[f_j(x)]}$$

$$\vec{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_k \end{pmatrix}$$

$$\begin{pmatrix} P(y=1 \mid x) \\ P(y=2 \mid x) \\ \vdots \\ P(y=k \mid x) \end{pmatrix}$$

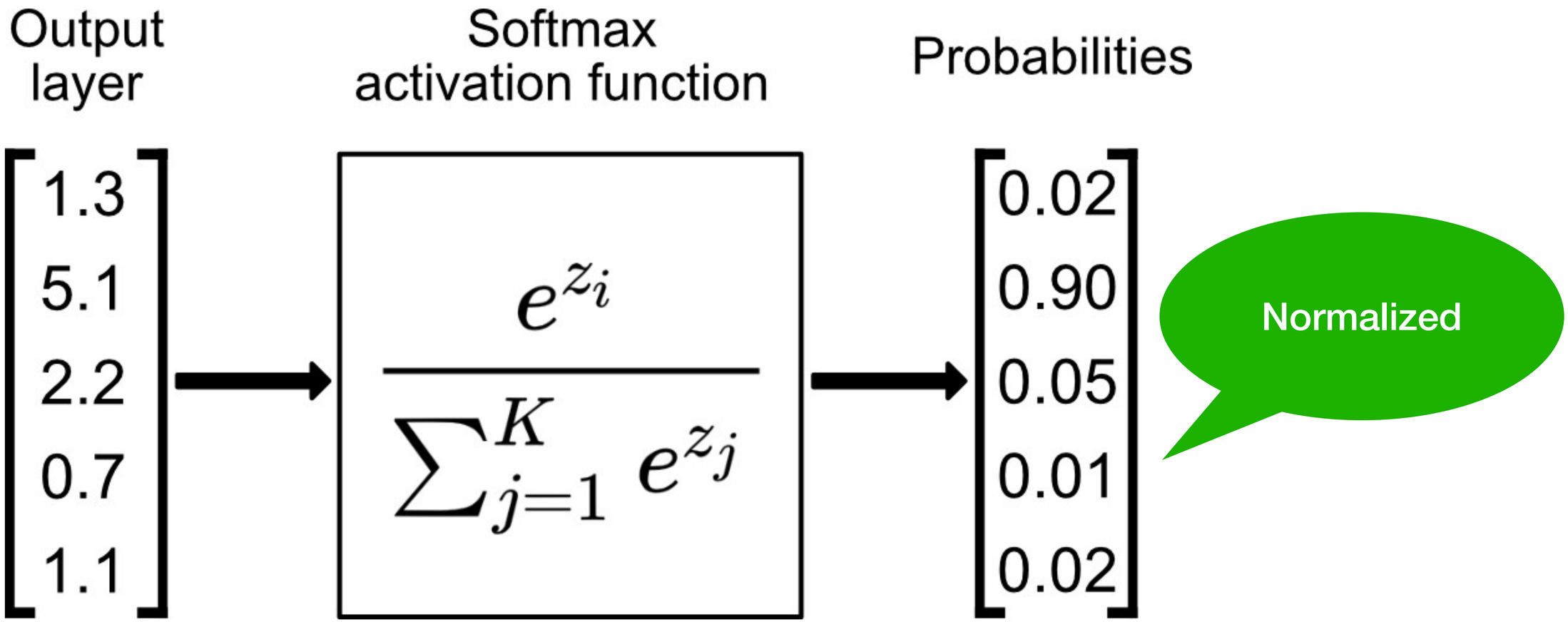
Softmax

Turns outputs *f* into probabilities (sum up to 1 across *k* classes)



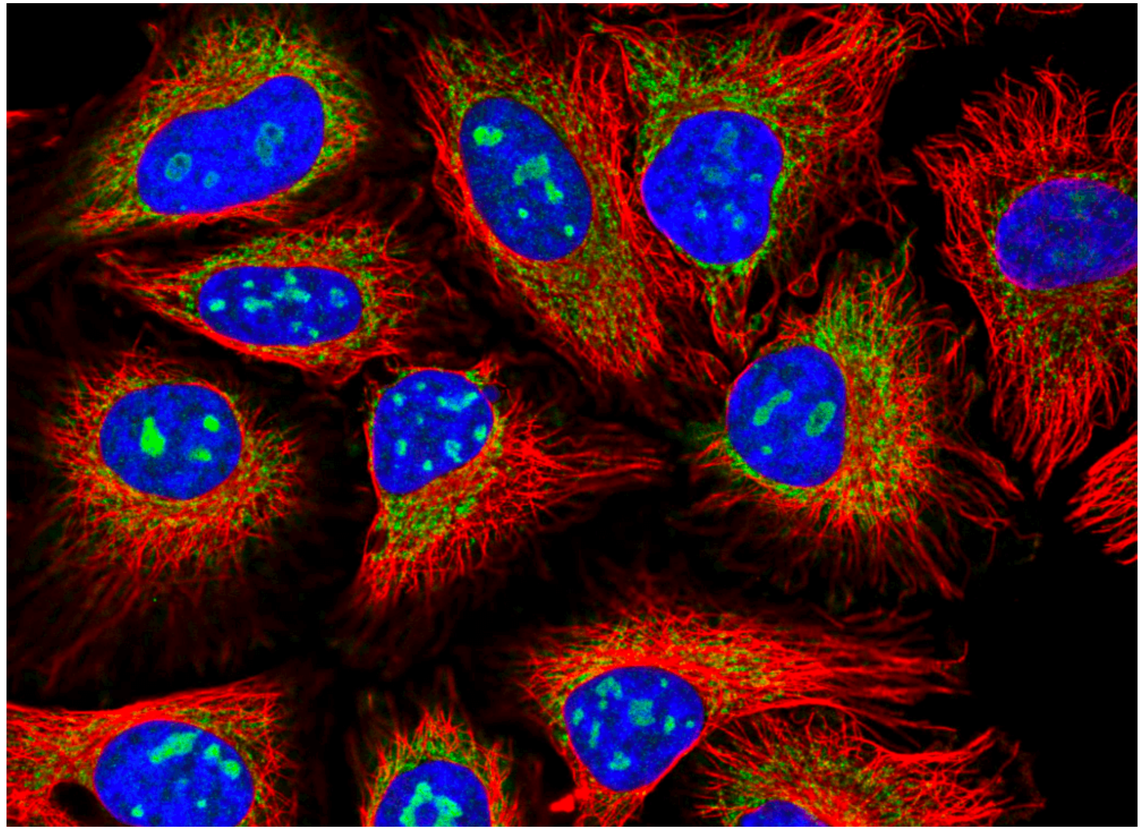
Softmax

Turns outputs *f* into probabilities (sum up to 1 across *k* classes)



Classification Tasks at Kaggle

Classify human protein microscope images into 28 categories

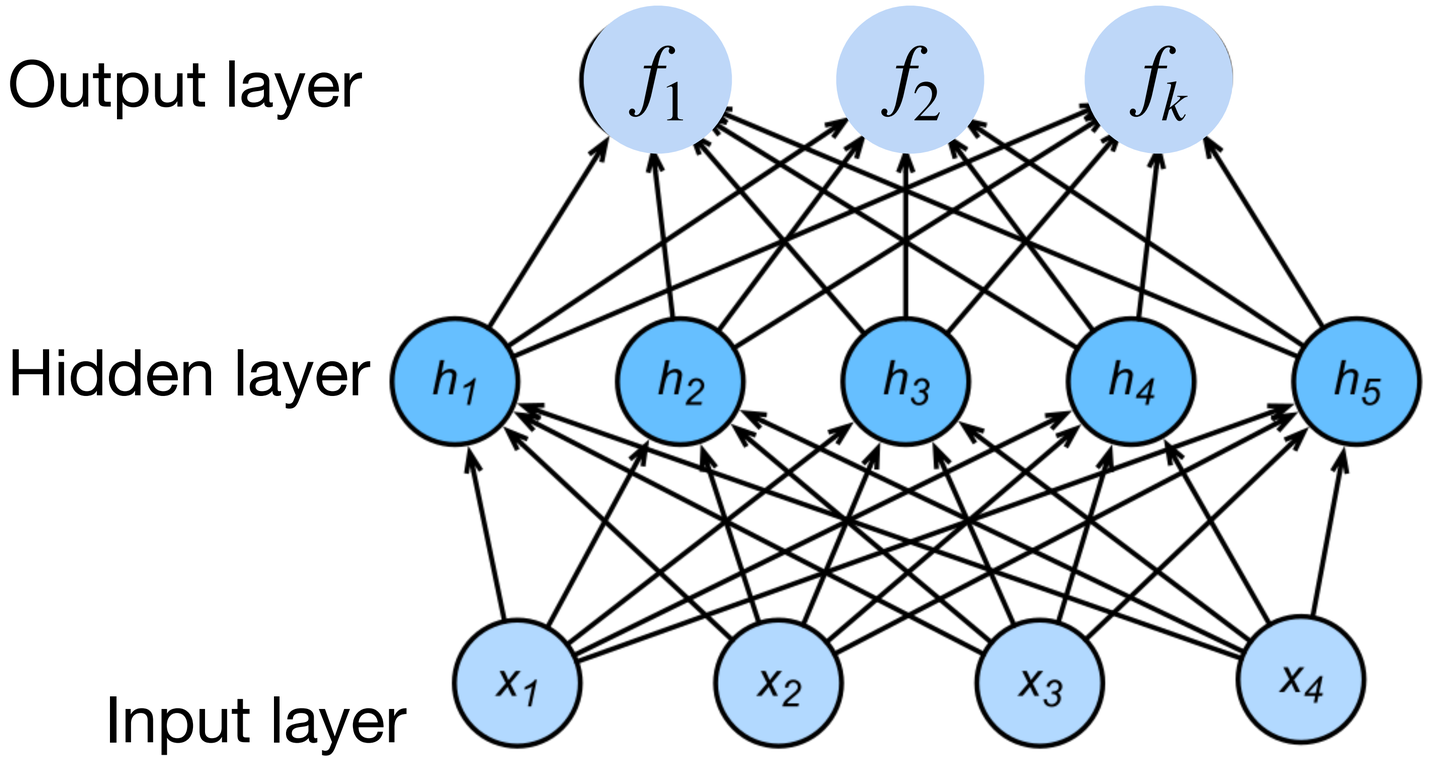


0. Nucleoplasm
1. Nuclear membrane
2. Nucleoli
3. Nucleoli fibrillar
4. Nuclear speckles
5. Nuclear bodies
6. Endoplasmic reticu
7. Golgi apparatus
8. Peroxisomes
9. Endosomes
10. Lysosomes
11. Intermediate fila
12. Actin filaments
13. Focal adhesion si
14. Microtubules
15. Microtubule ends
16. Cytokinetic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

https://www.kaggle.com/c/human-protein-atlas-image-classification

More complicated neural networks

$$y_1, y_2, \ldots, y_k = \text{softmax}(f_1, f_2, \ldots, f_k)$$



$k=3$

$m=5$

$d=4$

More complicated neural networks

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden layer

$$\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$$

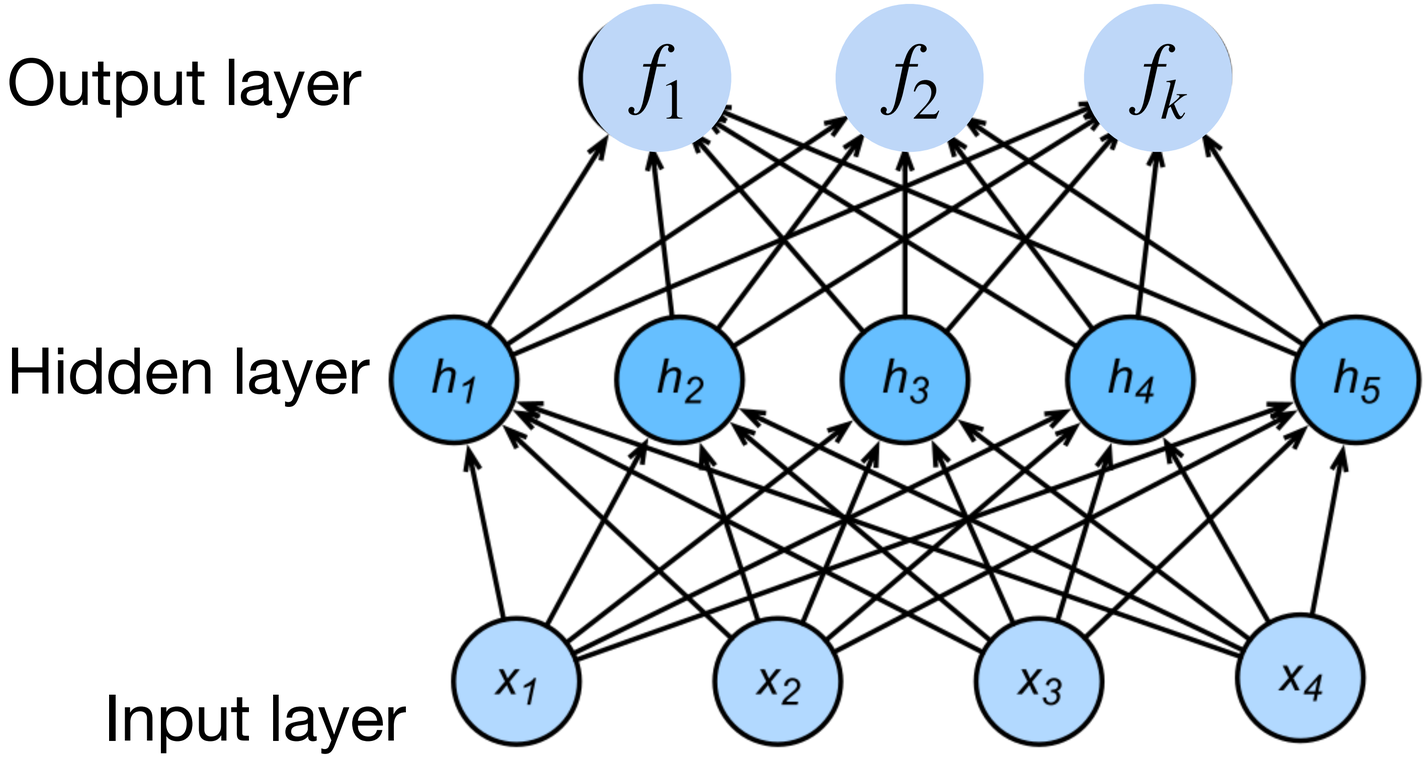
- Outputs:

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{f} = \sigma(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

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

$$y_1, y_2, \ldots, y_k = \text{softmax}(f_1, f_2, \ldots, f_k)$$



More complicated neural networks: 3 hidden layers

depth-4 NN.

$$\mathbf{h}_1 = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

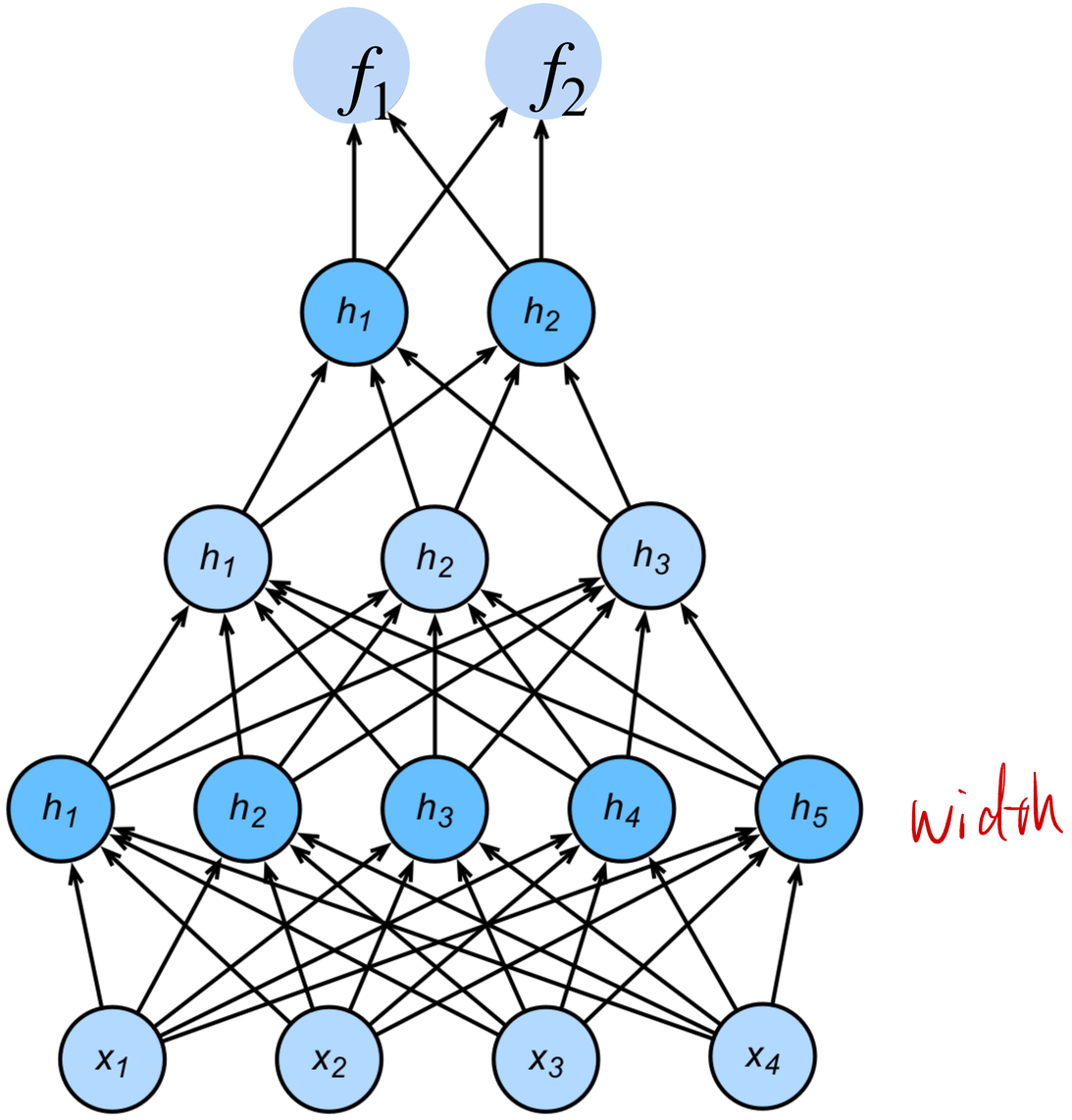
$$\mathbf{h}_2 = \sigma(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)})$$ 3rd

$$\mathbf{h}_3 = \sigma(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)})$$ 2nd

$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}_3 + \mathbf{b}^{(4)}$$

$$\mathbf{y} = \text{softmax}(\mathbf{f})$$ 1st

height / depth = # layers.



Quiz Break

Which output function is often used for multi-class classification tasks?

A Sigmoid function

B Rectified Linear Unit (ReLU)

C Softmax function

D Max function

Quiz Break

Which output function is often used for multi-class classification tasks?

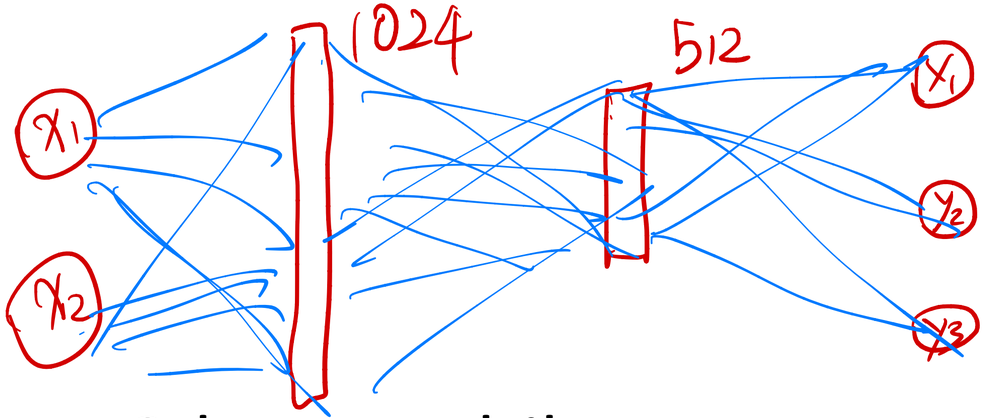
A Sigmoid function

B Rectified Linear Unit (ReLU)

C Softmax function

D Max function

Quiz Break



Suppose you are given a 3-layer multilayer perceptron (2 hidden layers h1 and h2 and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose h1 has 1024 units and h2 has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

1 st : $1024 \times 2 + 1024$

2nd : $512 \times 1024 + 512$

output : $3 \times 512 + 3$

Quiz Break

Suppose you are given a 3-layer multilayer perceptron (2 hidden layers h1 and h2 and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose h1 has 1024 units and h2 has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

1024 * 2 + 1024 + 512 * 1024 + 512 + 512 * 3 + 3 = 529411

Quiz Break

activation linear.

Consider a three-layer network with **linear Perceptrons** for binary classification. The hidden layer has 3 neurons. Can the network represent a XOR problem?

a)Yes

b)No

Quiz Break

Consider a three-layer network with **linear Perceptrons** for binary classification. The hidden layer has 3 neurons. Can the network represent a XOR problem?

a)Yes

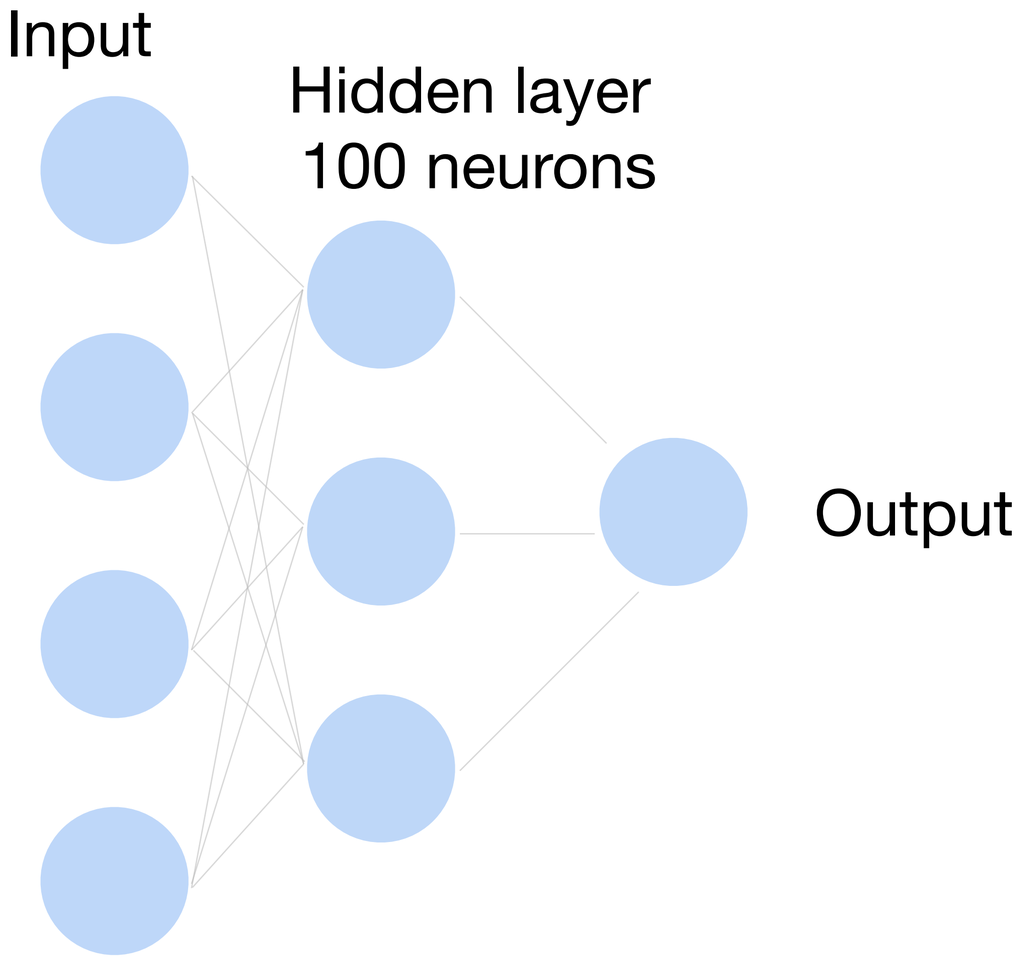
b)No 

Solution:

A combination of linear Perceptrons is still a linear function.

How to train a neural network?

Classify cats vs. dogs



How to train a neural network? 2-class

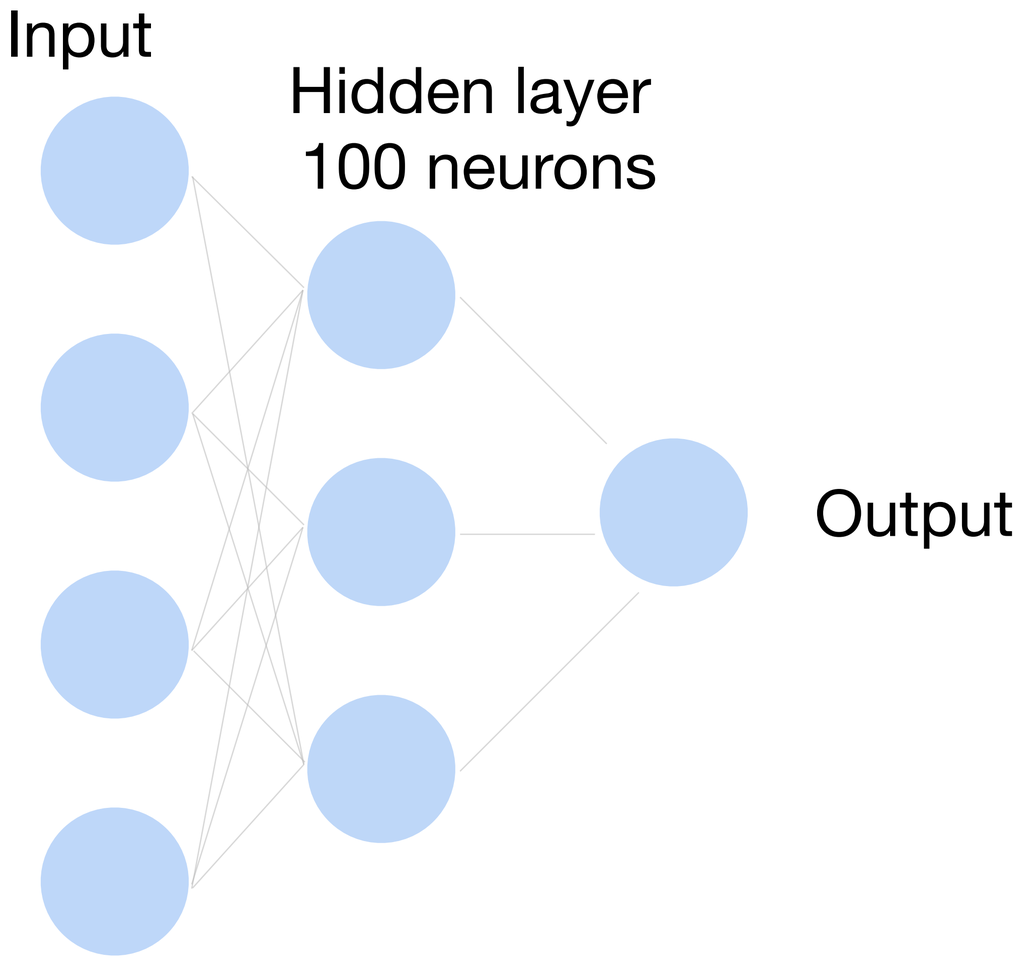
$\mathbf{x}_i \in \mathbb{R}^d$: One training data point in the training set D

$\hat{y}_i$ Model output for $\mathbf{x}_i$

y_i Ground truth label for $\mathbf{x}_i$

Learning by matching the output to the label

We want $\hat{y}_i \to 1$ **when** $y_i = 1$,
and $\hat{y}_i \to 0$ **when** $y_i = 0$



How to train a neural network? 2-class

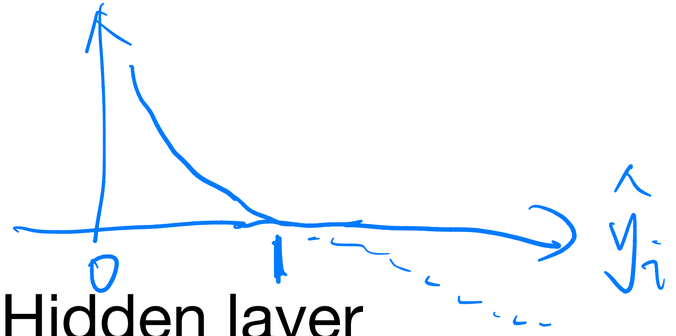
Loss function: $$\frac{1}{|D|}\sum_i \ell(\mathbf{x}_i, y_i)$$

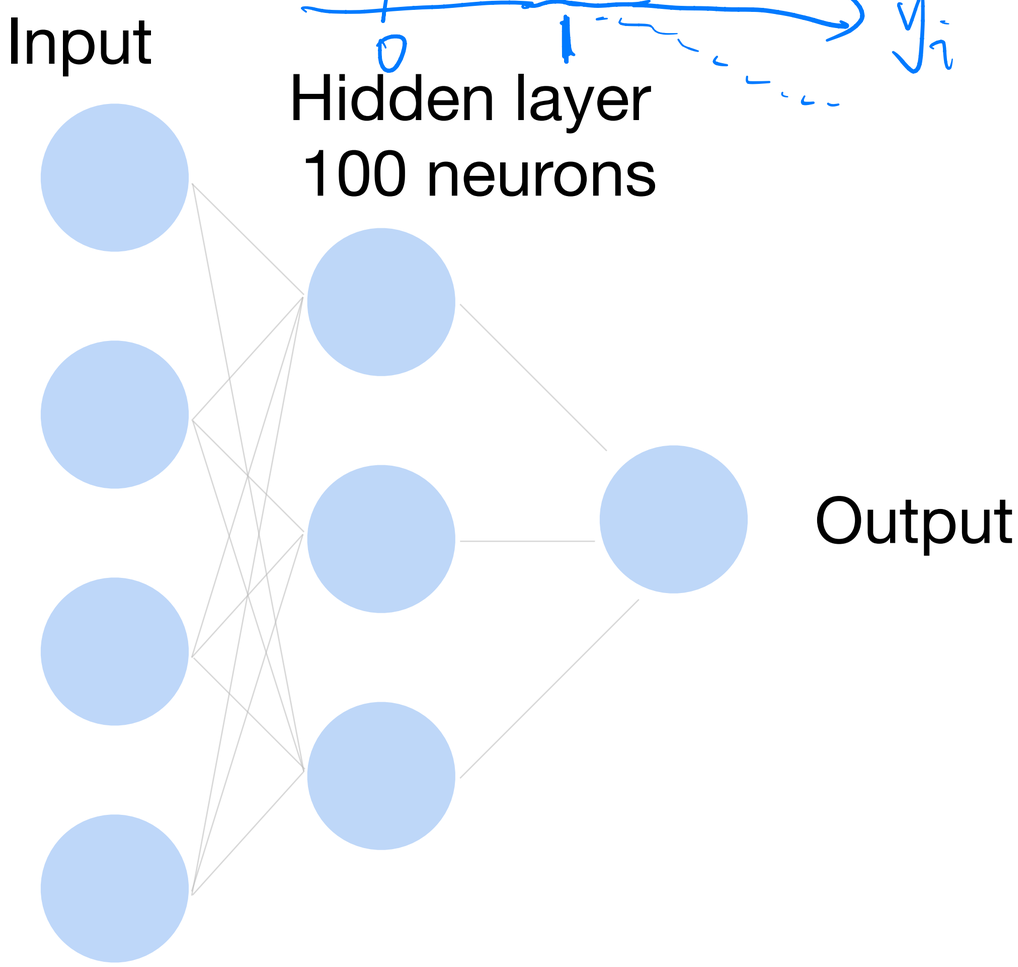
Per-sample loss:

$$\ell(\mathbf{x}_i, y_i) = -y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

$$= \begin{cases} -\log(\hat{y}_i) & y_i = 1 \\ \log(1 - \hat{y}_i) & y_i = 0 \end{cases}$$

Also known as binary cross-entropy loss





How to train a neural network? *K*-class

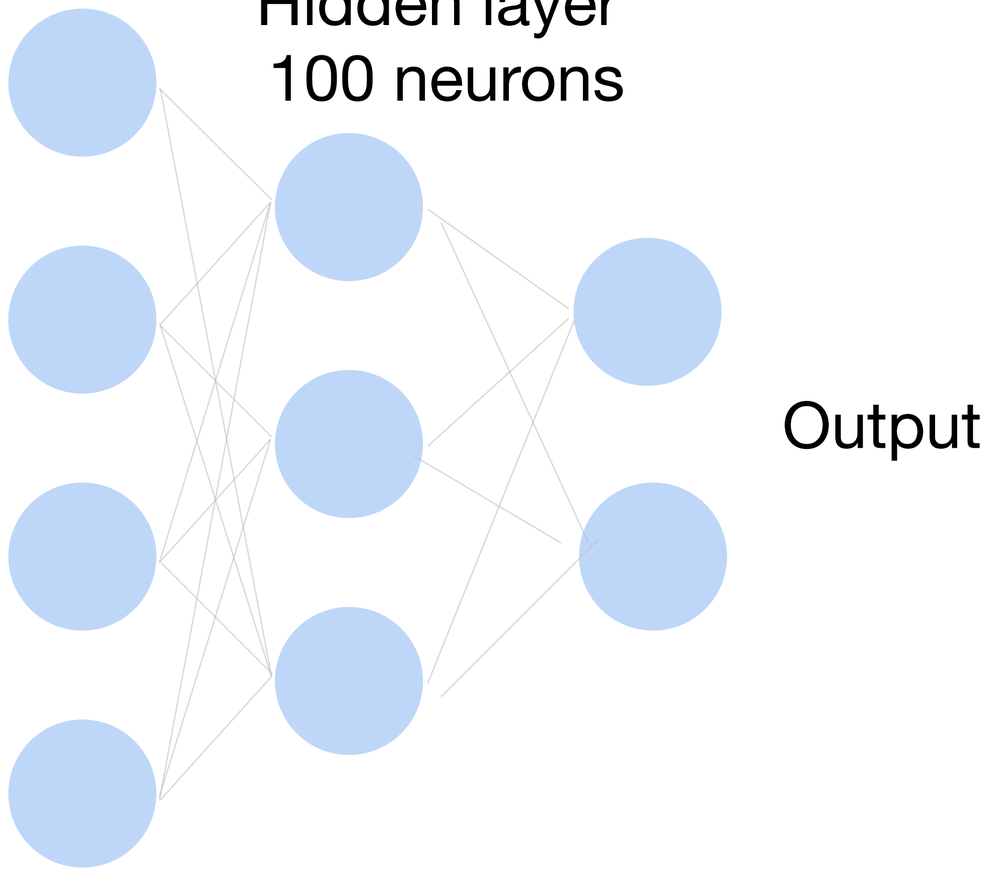
Loss function: $L = \frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$

$y_i = \begin{pmatrix} y_{i1} \\ y_{i2} \\ \vdots \\ y_{iK} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ if x_i is in class i

Per-sample loss:

$$\ell(\mathbf{x}_i, y_i) = \sum_{j=1}^{K} - y_{ij} \log \hat{p}_{ij}$$

Also known as cross-entropy loss or softmax loss

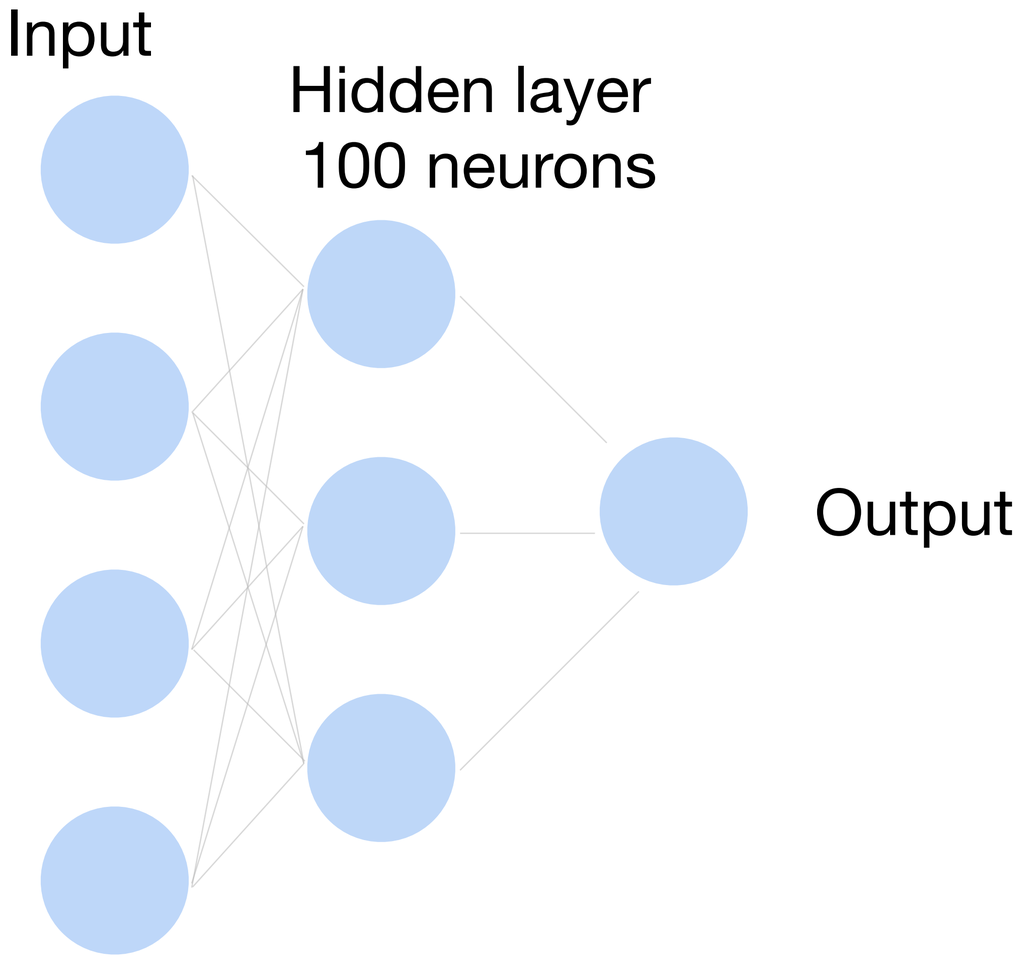


How to train a neural network?

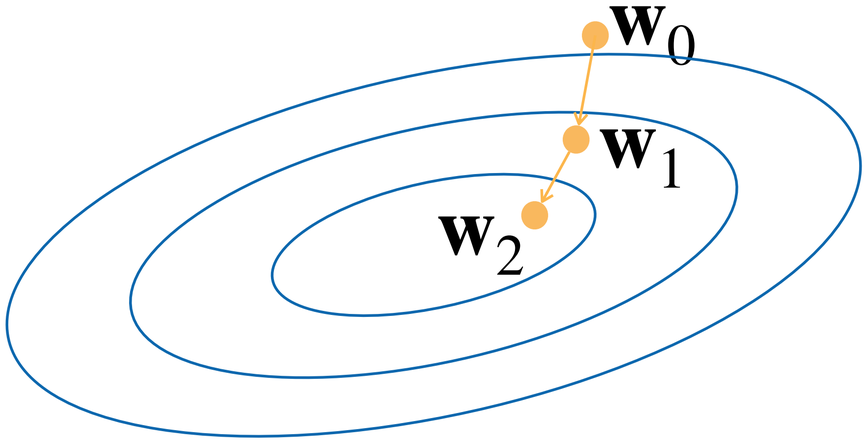
Update the weights *W* to minimize the loss function

$$L = \frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$$

Use gradient descent!



Gradient Descent



- Choose a learning rate $\alpha > 0$
- Initialize the model parameters $\mathbf{w_0}$
- For t =1,2,…
  - Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{\partial L}{\partial \mathbf{w}_{t-1}}$$

D can be very large. Expensive

$$= \mathbf{w}_{t-1} - \alpha \frac{1}{|D|} \sum_{i \in D} \frac{\partial \ell(\mathbf{x}_i, y_i)}{\partial \mathbf{w}_{t-1}}$$

- Repeat until converges

Minibatch Stochastic Gradient Descent

$|\hat{D}| = 1 \Rightarrow$ SGD

$|\hat{D}| = 8, 16, 32.$

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters $\mathbf{w_0}$
- For t =1,2,…
  - **Randomly sample a subset (mini-batch)** $\hat{D} \subset D$
    Update parameters:

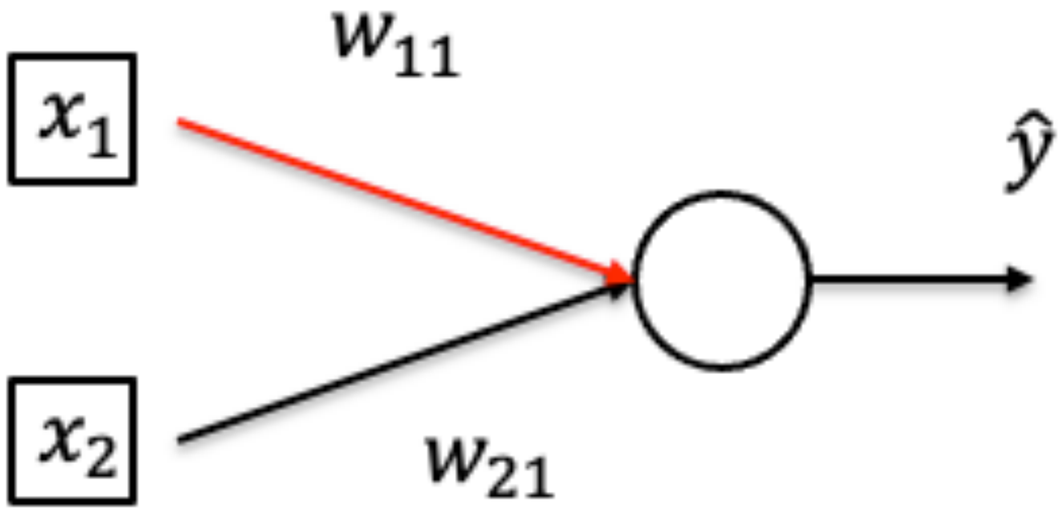
$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{1}{|\hat{D}|} \sum_{i \in \hat{D}} \frac{\partial \ell(\mathbf{x}_i, y_i)}{\partial \mathbf{w}_{t-1}}$$

- Repeat until converges

Calculate Gradient (on one data point)

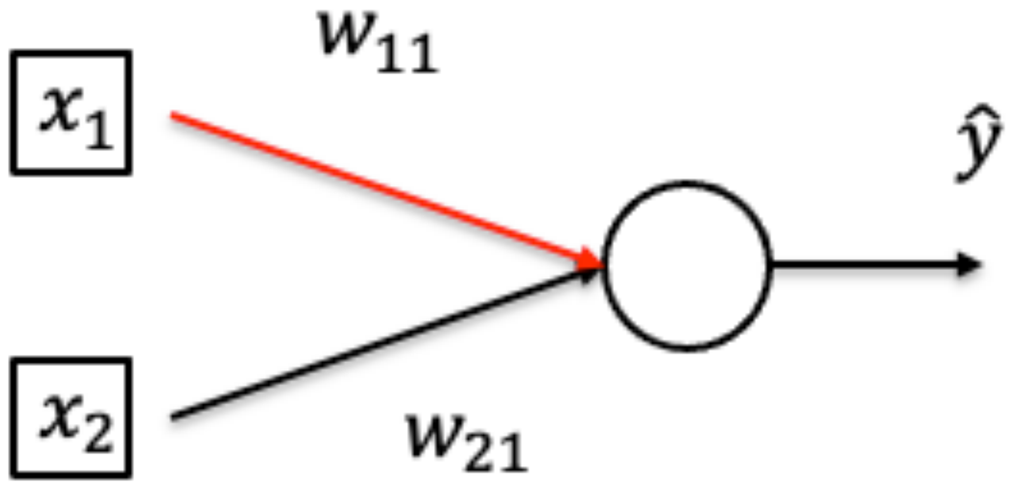
"Back-propagation algorithm"

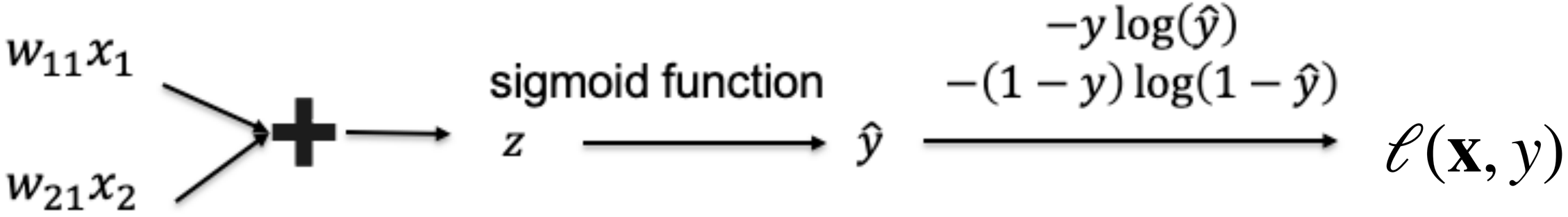
automatic chain rule.



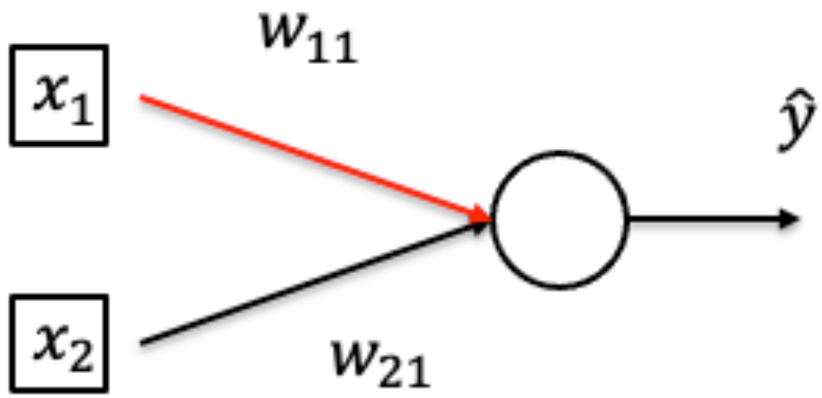
- Want to compute $\dfrac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

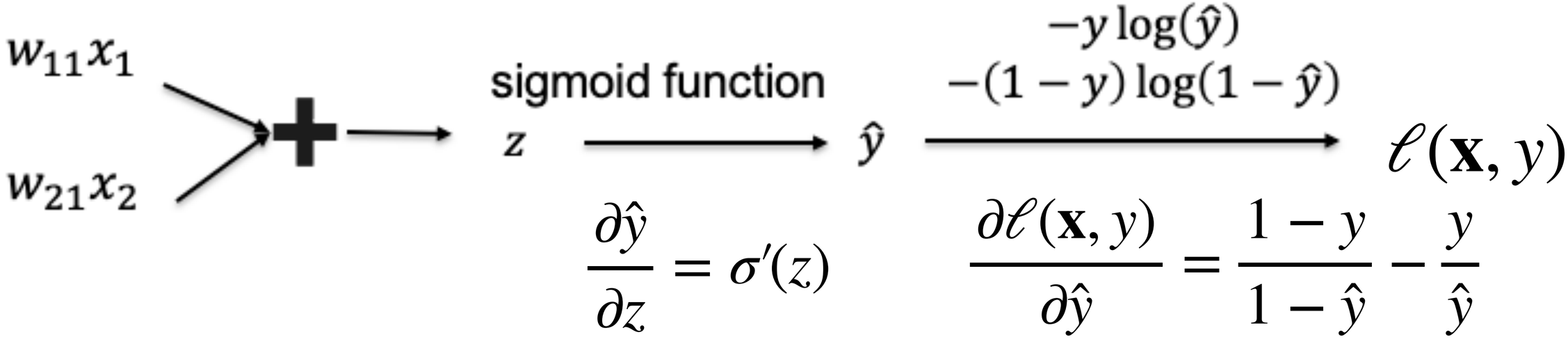
Calculate Gradient (on one data point)





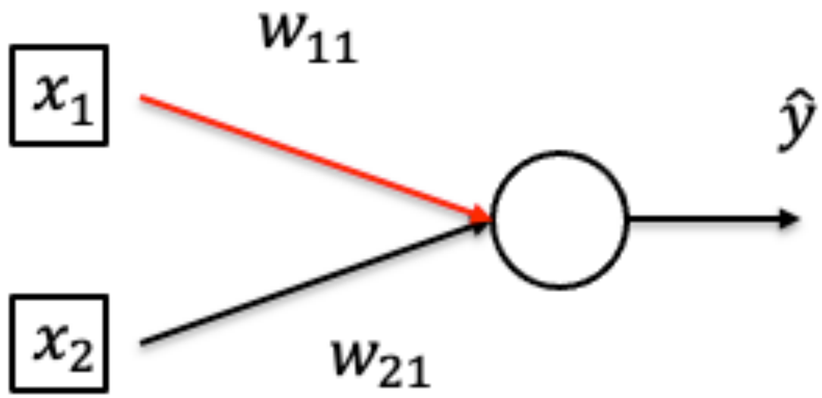
Calculate Gradient (on one data point)





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

Calculate Gradient (on one data point)



$w_{11}x_1$

$w_{21}x_2$

sigmoid function

$z \longrightarrow \hat{y}$

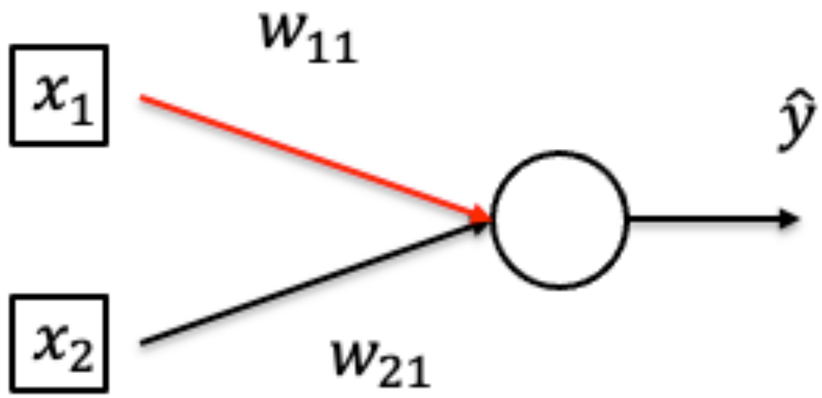
$-y\log(\hat{y})$
$-(1-y)\log(1-\hat{y})$

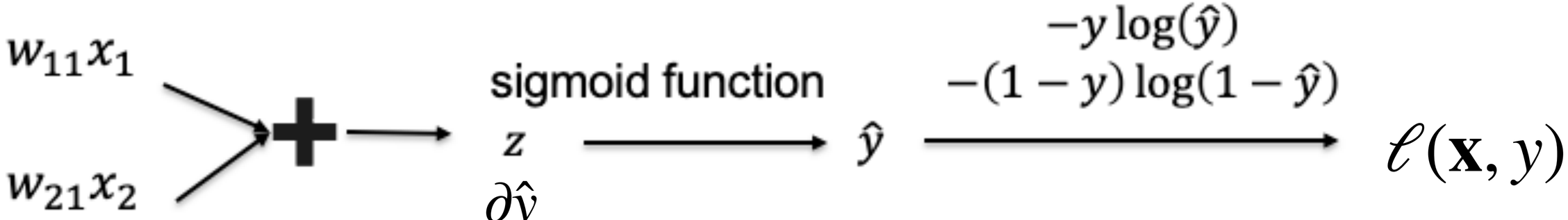
$\longrightarrow \ell(\mathbf{x}, y)$

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

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

Calculate Gradient (on one data point)





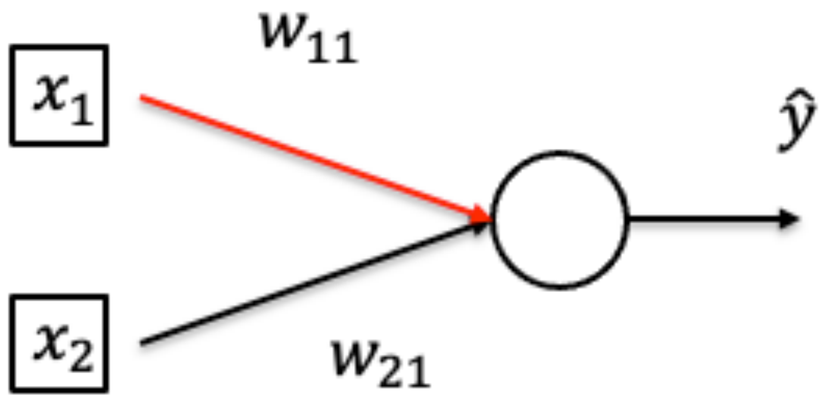
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

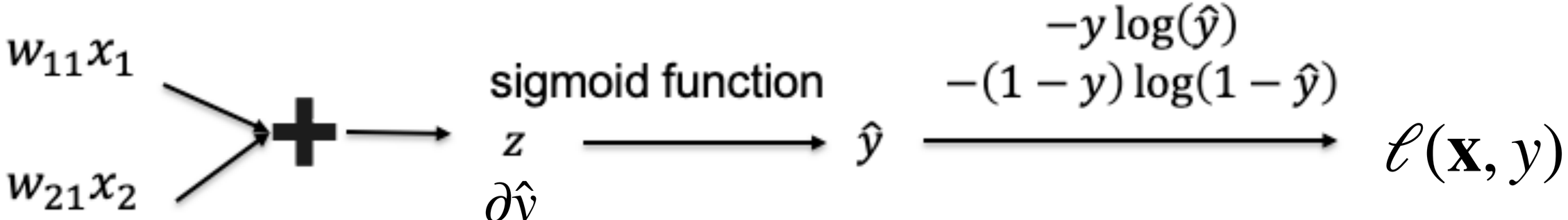
$\hat{y} = \sigma(z)$

- By chain rule:

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

Calculate Gradient (on one data point)

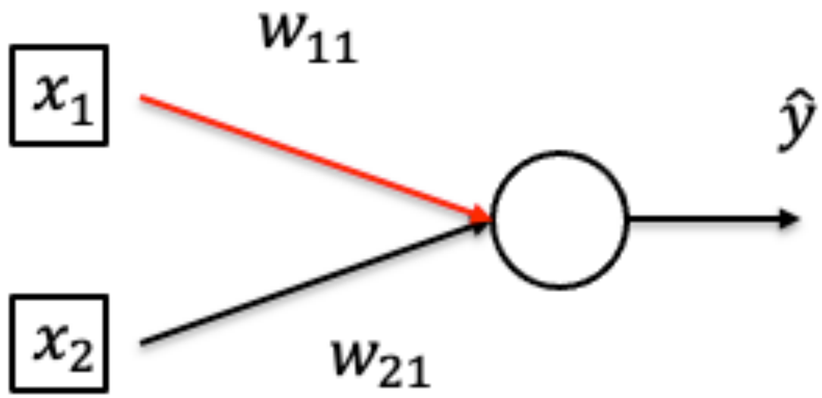


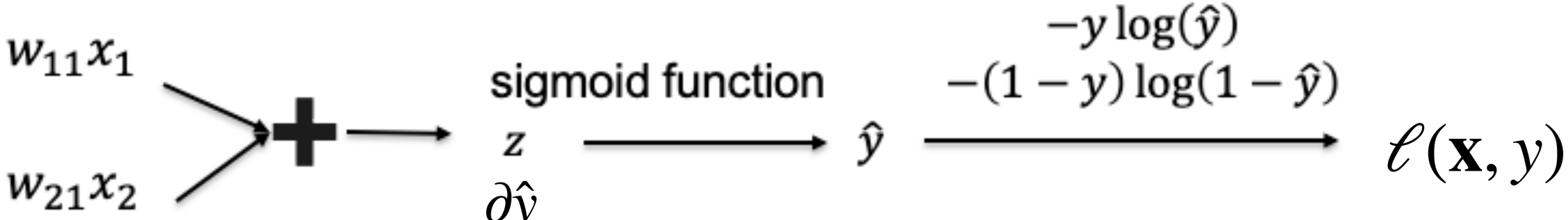


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

Calculate Gradient (on one data point)

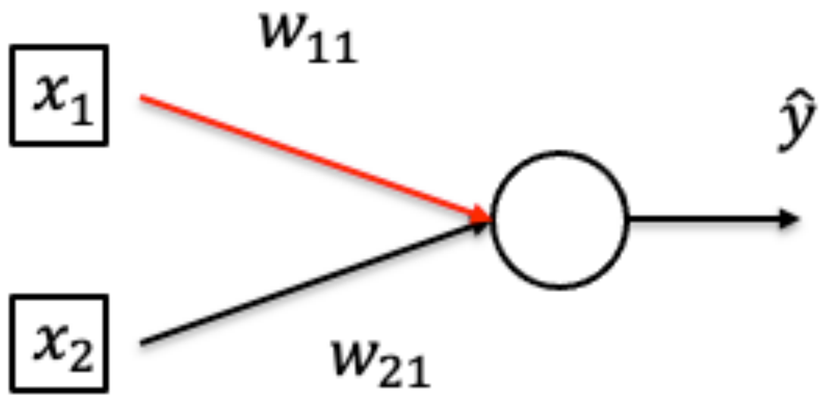




$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

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

Calculate Gradient (on one data point)



$w_{11}x_1$

$w_{21}x_2$

sigmoid function

$z \longrightarrow \hat{y}$

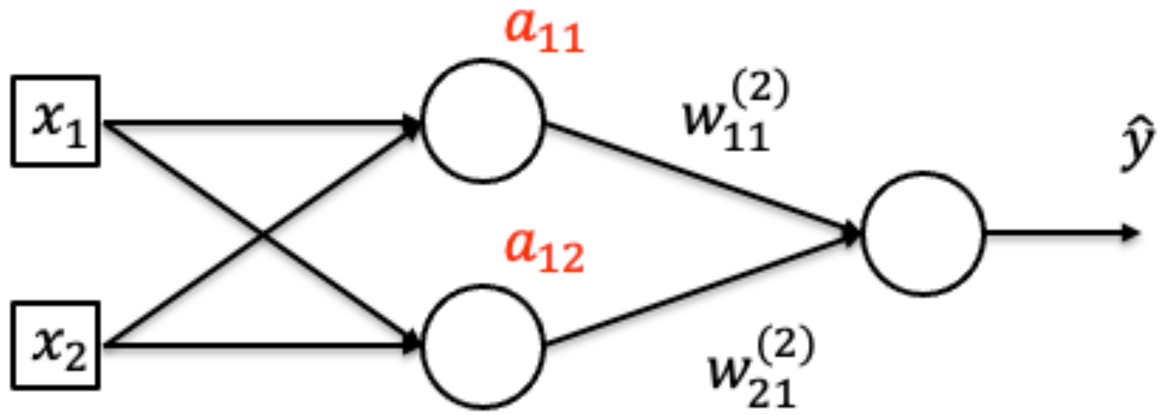
$-y\log(\hat{y})$
$-(1-y)\log(1-\hat{y})$

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

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

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

Calculate Gradient (on one data point)



Make it deeper

$w_{11}^{(2)} a_{11}$

$w_{21}^{(2)} a_{12}$

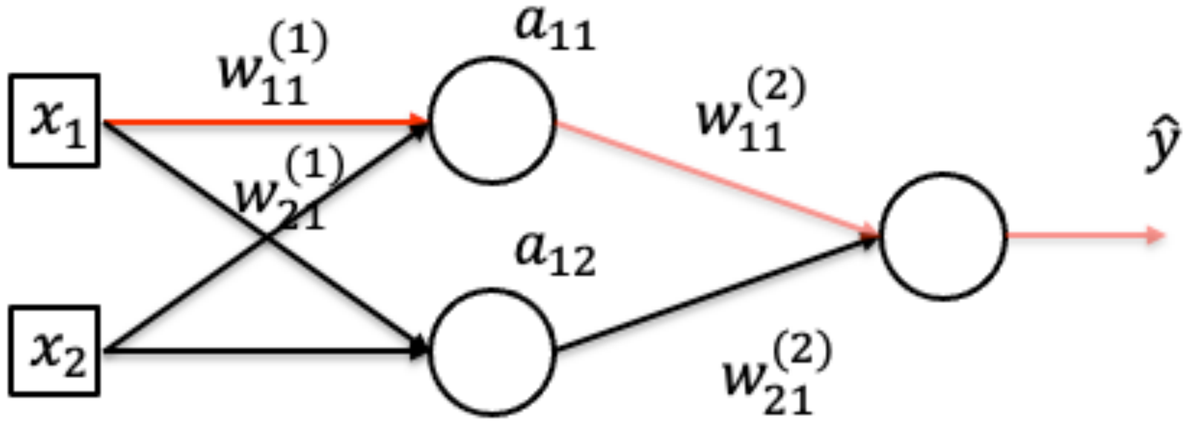
sigmoid function

$z \longrightarrow \hat{y} \xrightarrow{\substack{-y\log(\hat{y}) \\ -(1-y)\log(1-\hat{y})}} \ell(\mathbf{x}, y)$

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

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

Calculate Gradient (on one data point)

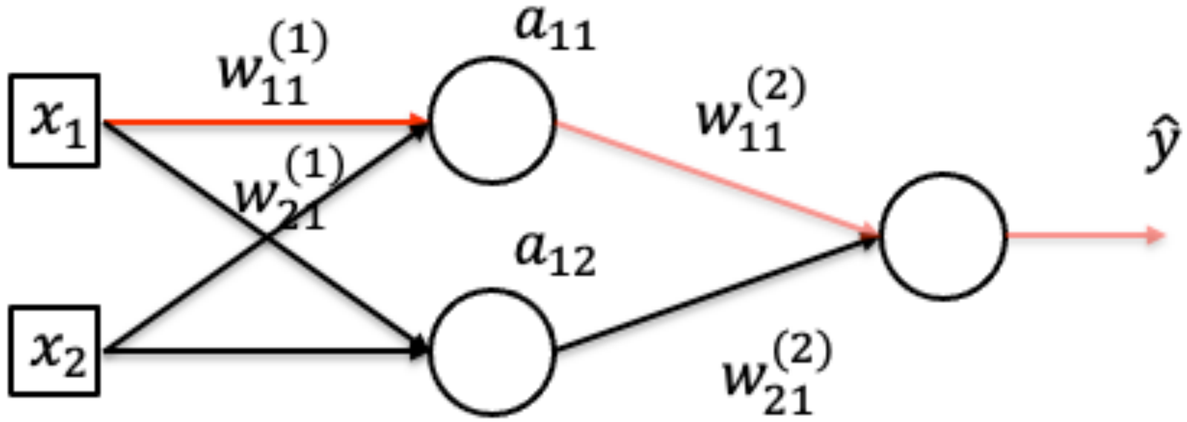


$$w_{11}^{(1)}x_1 \quad w_{21}^{(1)}x_2 \quad \rightarrow + \rightarrow \quad z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \rightarrow l(x,y)$$

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

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

Calculate Gradient (on one data point)



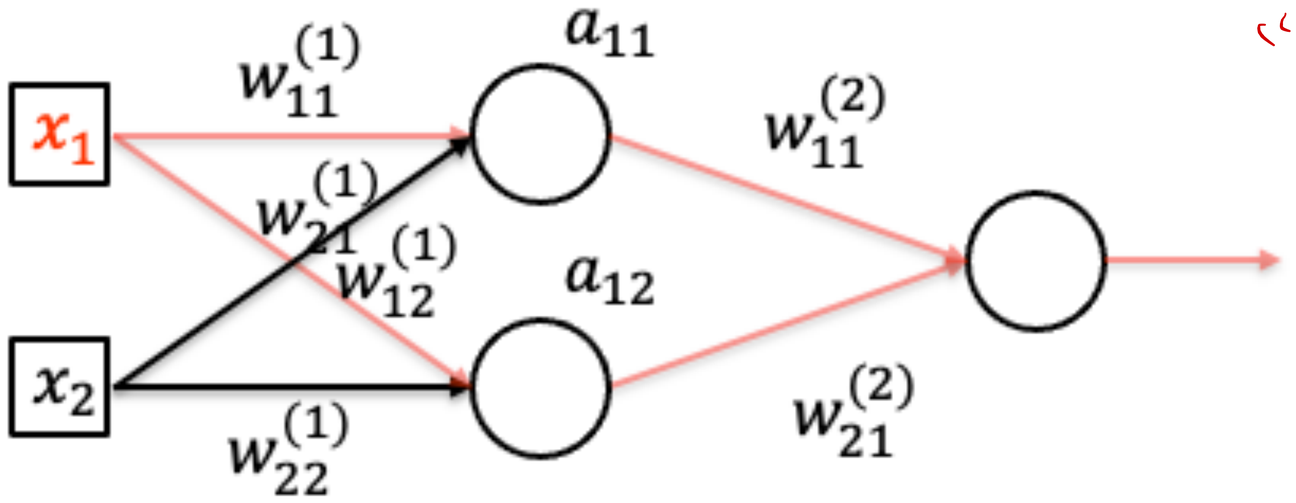
$$w_{11}^{(1)}x_1 \quad w_{21}^{(1)}x_2 \quad \rightarrow + \rightarrow \quad z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \rightarrow l(x,y)$$

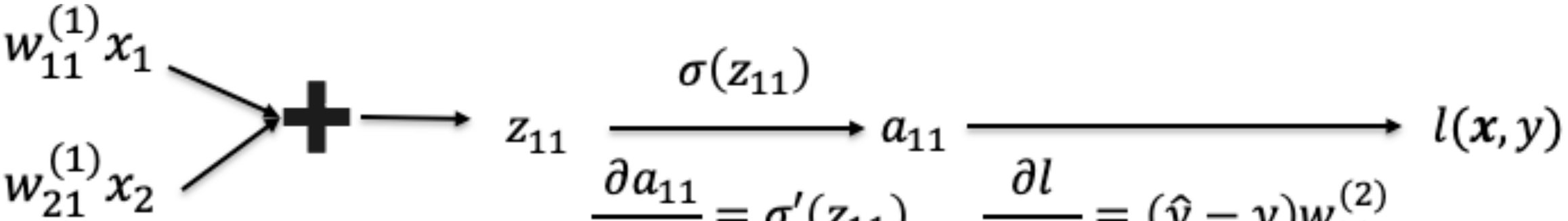
$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

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

Calculate Gradient (on one data point)

"automatic differentiation"





- By chain rule:

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

Quiz Break

Gradient Descent in neural network training computes the _____ of a loss function with respect to the model _____ until convergence.

A gradients, parameters

B parameters, gradients

C loss, parameters

D parameters, loss

Quiz Break

Gradient Descent in neural network training computes the _____ of a loss function with respect to the model _____ until convergence.

A gradients, parameters

B parameters, gradients

C loss, parameters

D parameters, loss

Quiz Break

Suppose you are given a dataset with 1,000,000 images to train with. Which of the following methods is more desirable if training resources are limit but decent accuracy is needed?

A Gradient Descent

B Stochastic Gradient Descent

C Minibatch Stochastic Gradient Descent

D Computation Graph

Quiz Break

Suppose you are given a dataset with 1,000,000 images to train with. Which of the following methods is more desirable if training resources are limit but decent accuracy is needed?

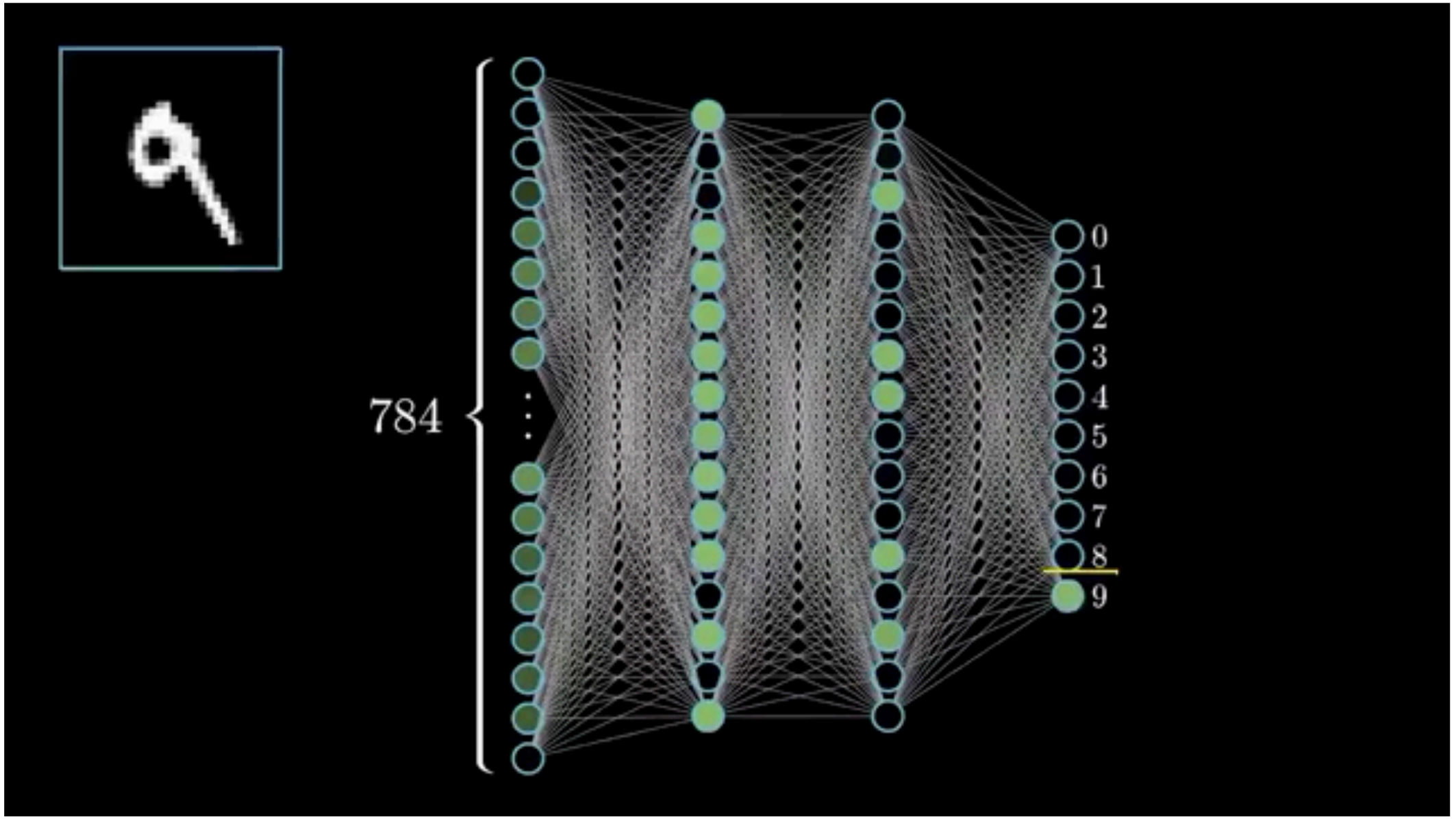
A Gradient Descent

B Stochastic Gradient Descent

C Minibatch Stochastic Gradient Descent

D Computation Graph

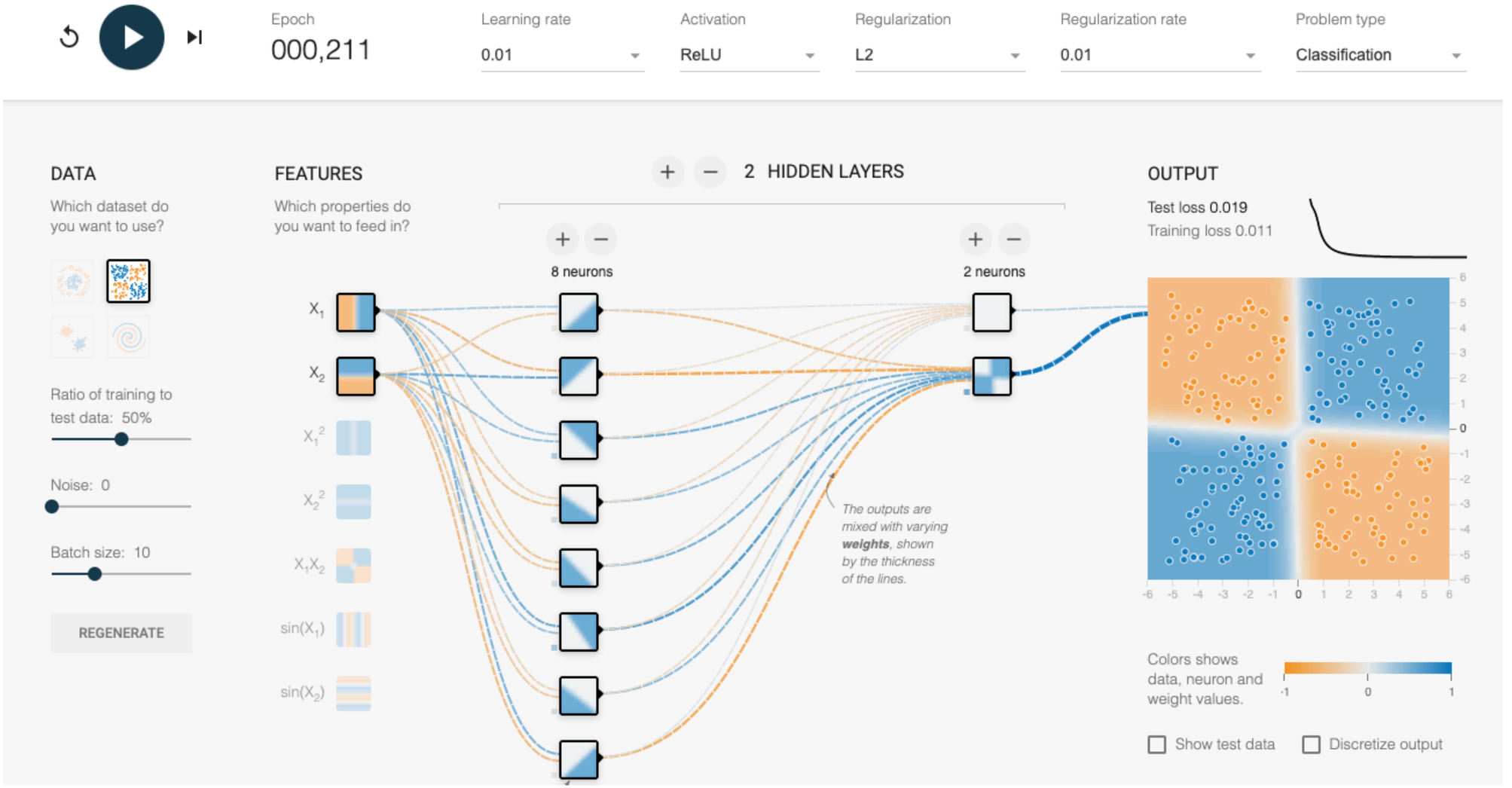
HW6



HW6 (working with MNIST dataset)

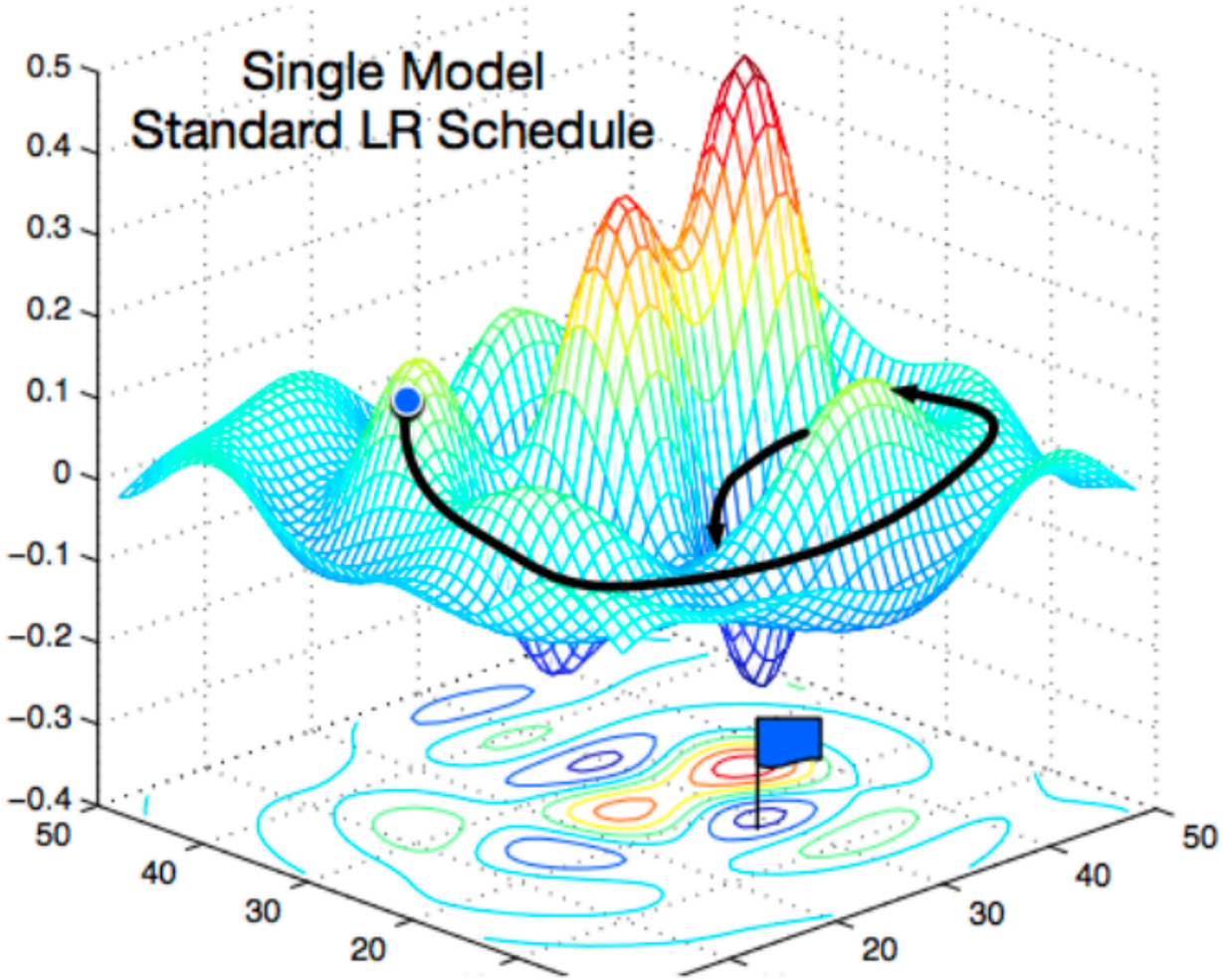
PyTorch
(TensorFlow)

Demo: Learning XOR using neural net



- https://playground.tensorflow.org/

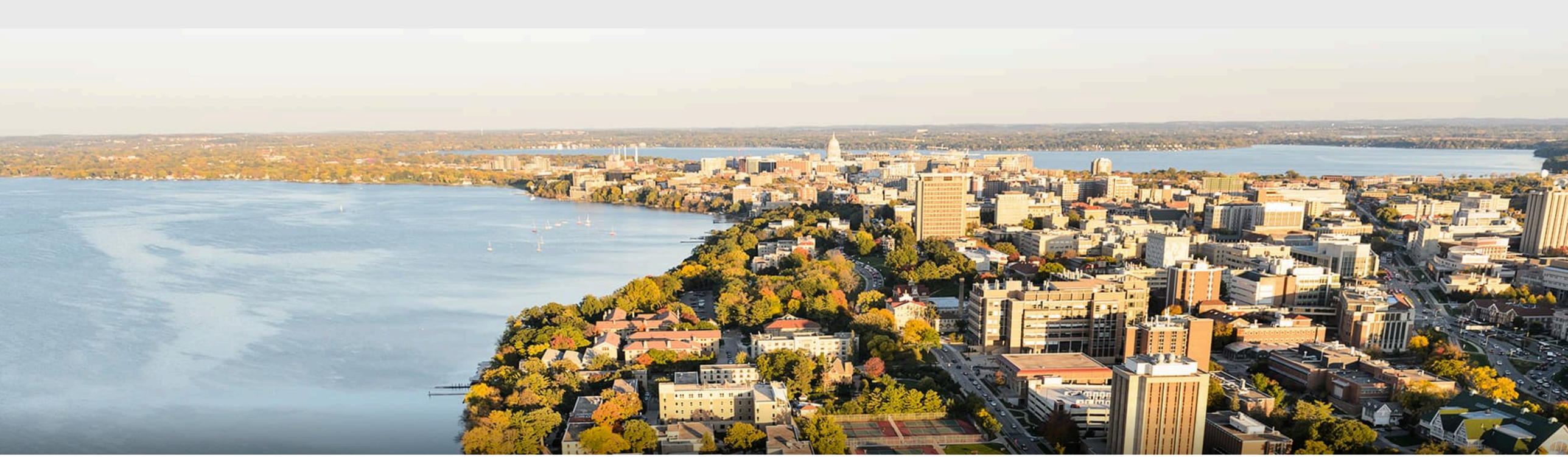
Non-convex Optimization



[Gao and Li et al., 2018]

What we've learned today…

- Single-layer Perceptron Review
- Multi-layer Perceptron
  - Single output
  - Multiple output
- How to train neural networks
  - Gradient descent



Thanks!