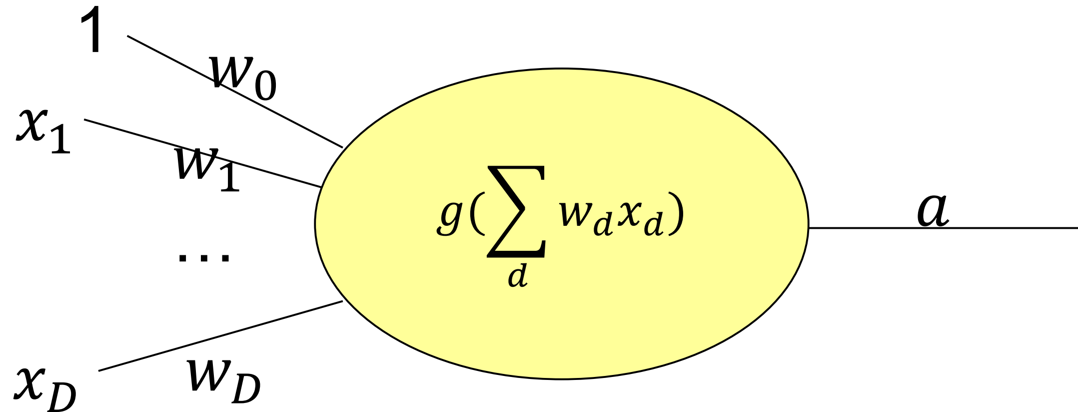
Neural Networks
Part 1

Yin Li
yin.li@wisc.edu
University of Wisconsin, Madison

[Based on slides from Yingyu Liang, Jerry Zhu, Mohit Gupta]

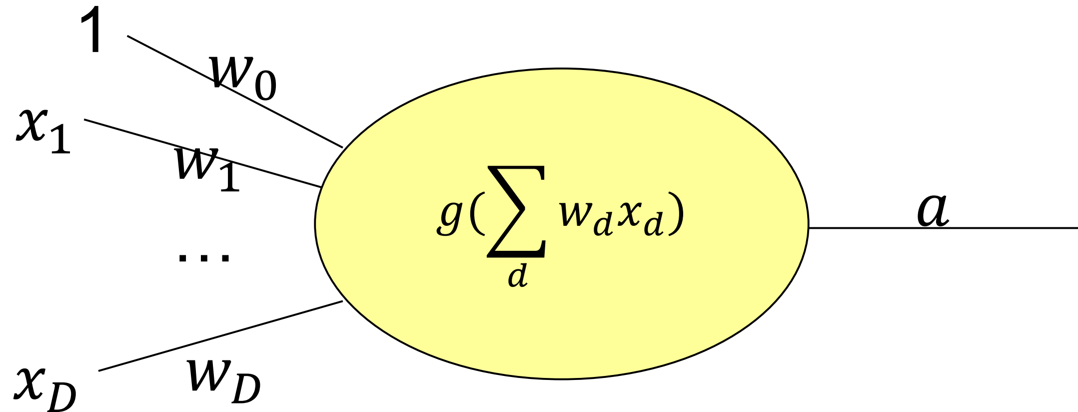
Limited power of one single neuron

- Perceptron: $a = g(\sum_d w_d x_d)$
- Activation function g: linear, step, sigmoid, …

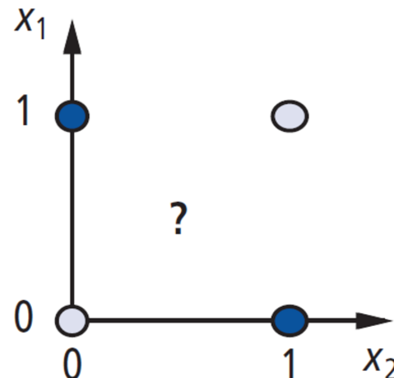


Limited power of one single neuron

- Perceptron: $a = g(\sum_d w_d x_d)$
- Activation function g: linear, step, sigmoid, …

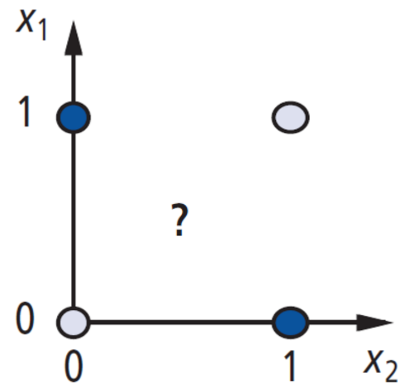


- Decision boundary linear even for nonlinear g
- XOR problem



Limited power of one single neuron

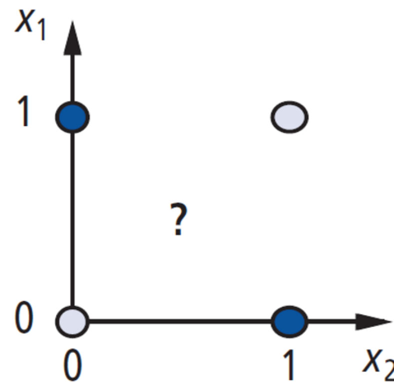
- XOR problem



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

Limited power of one single neuron

- XOR problem



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

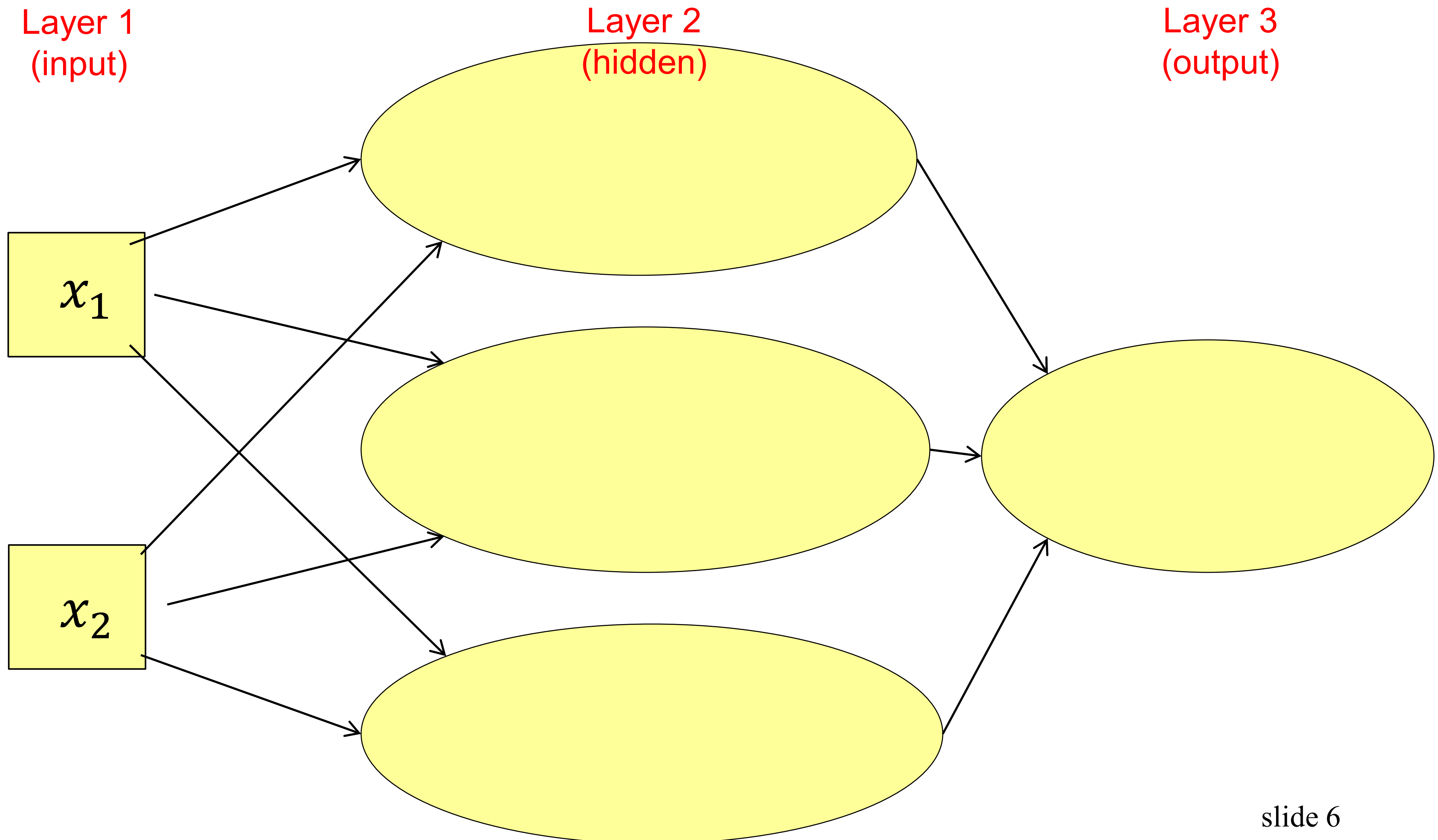
- Wait! If one can represent AND (∧), OR (∨), NOT (¬), one can represent any logic circuit (including XOR), by connecting them

$$XOR(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x1 \wedge x_2)$$

Question: how to represent XOR using Perceptron?

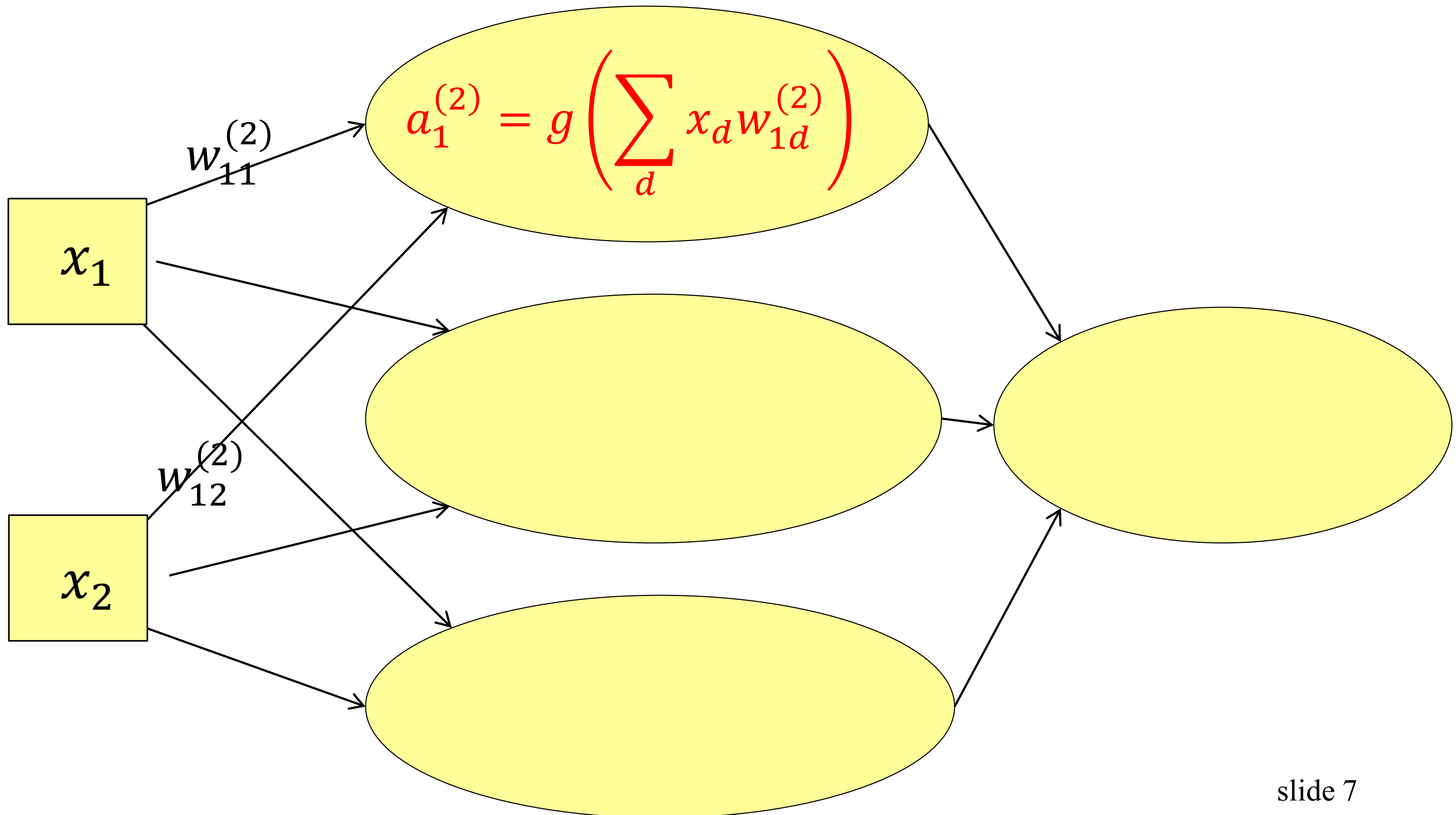
Multi-layer Perceptron: neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



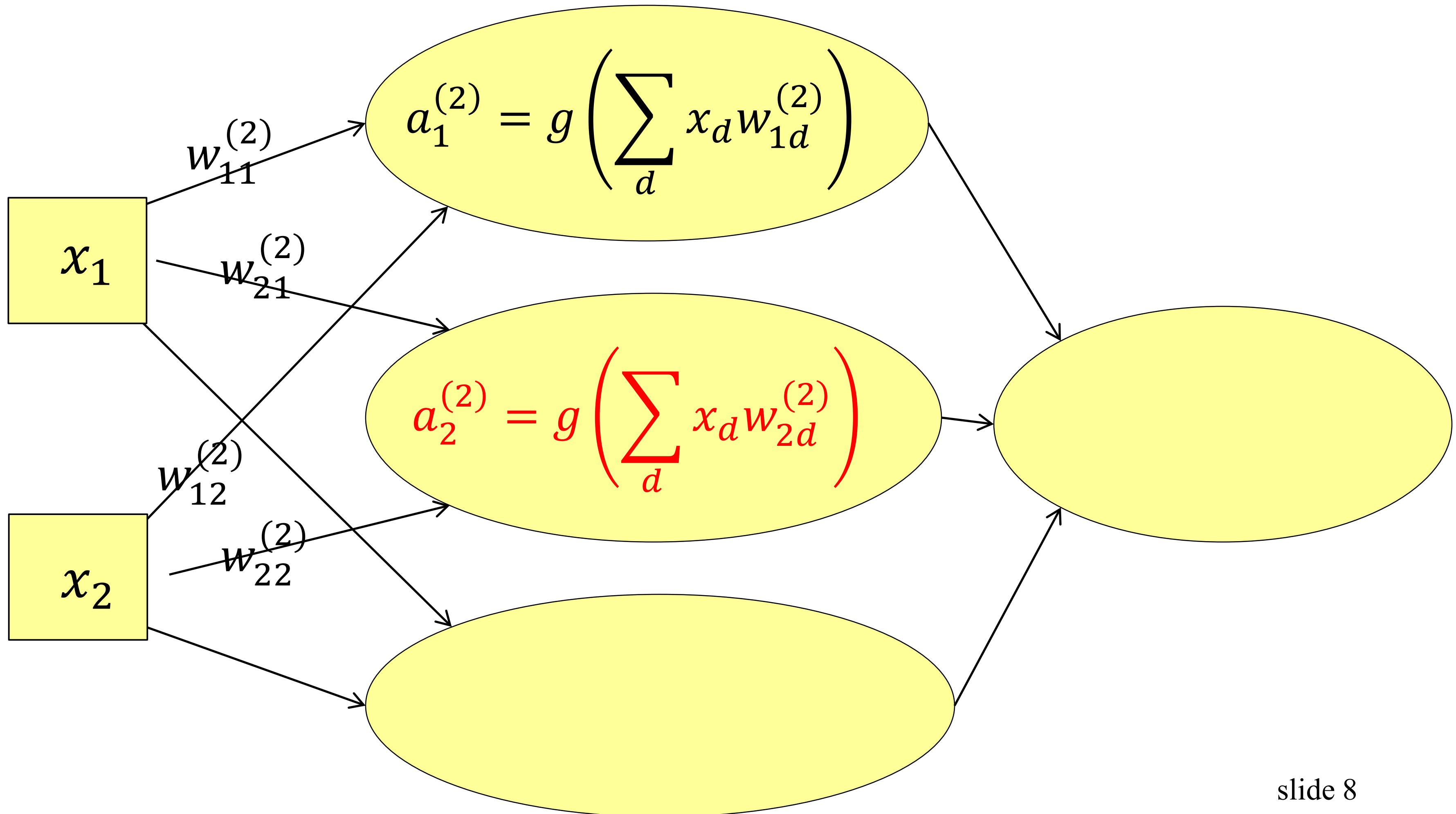
Multi-layer Perceptron: neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



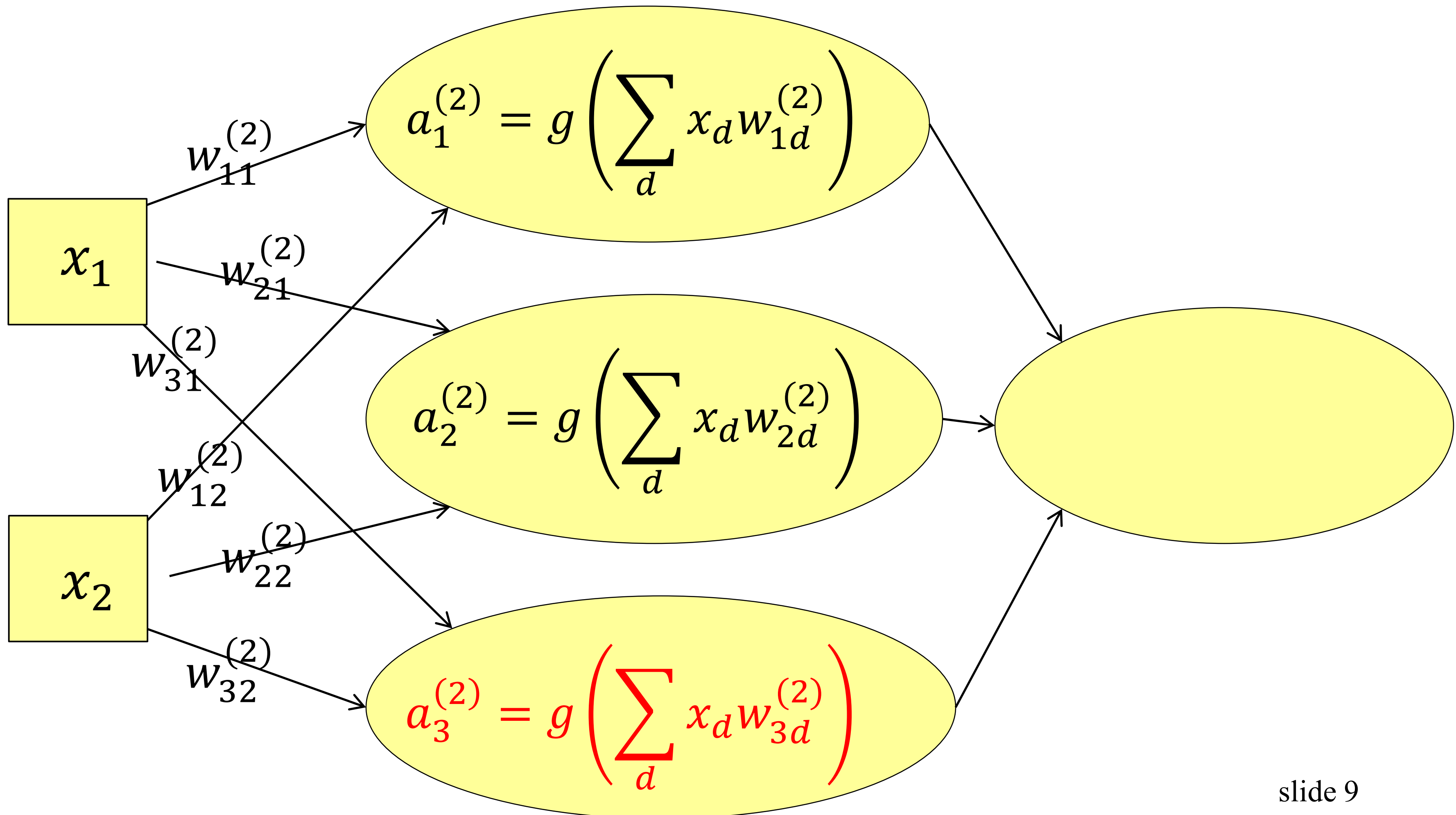
Multi-layer Perceptron: neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



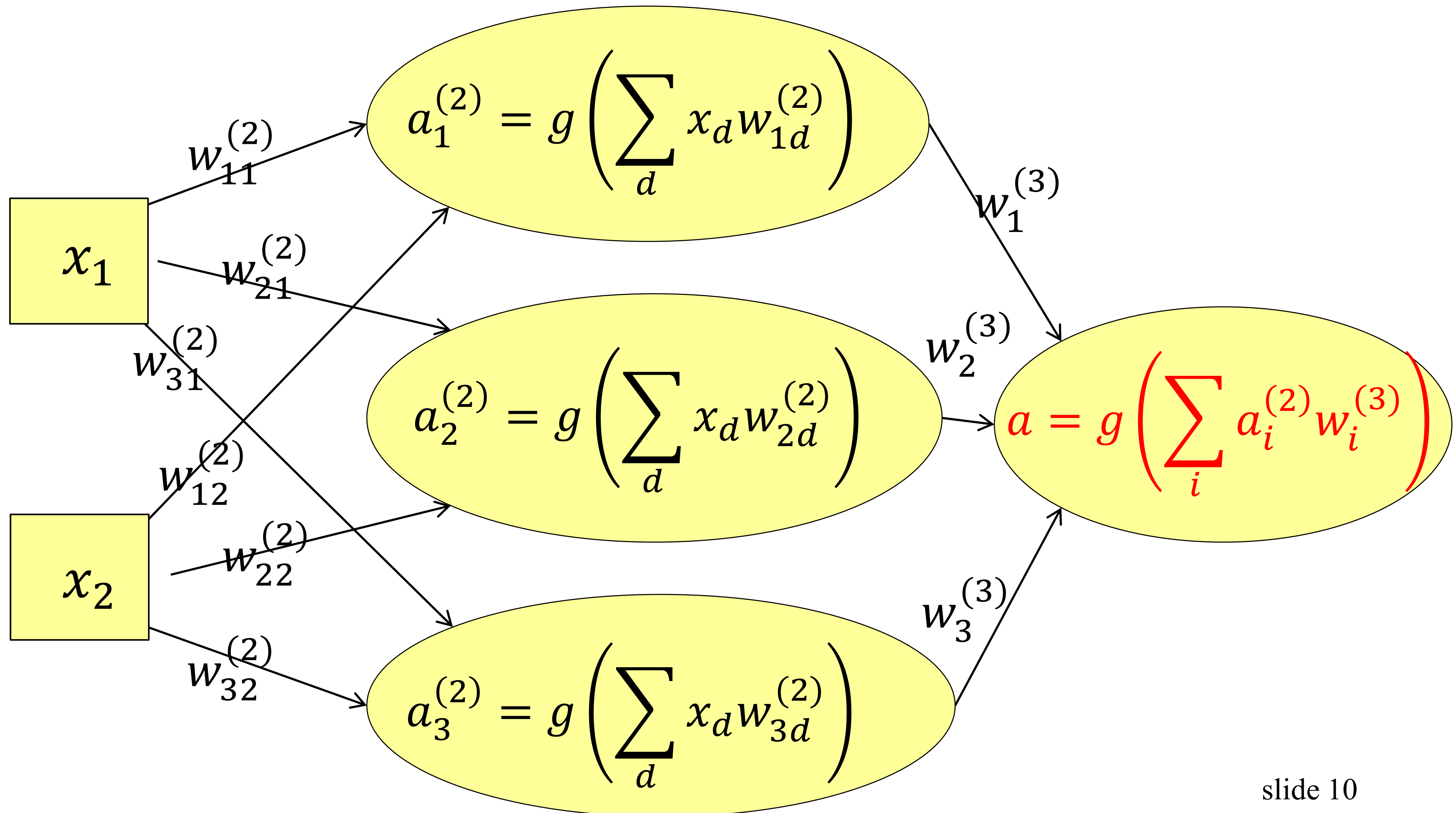
Multi-layer Perceptron: neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



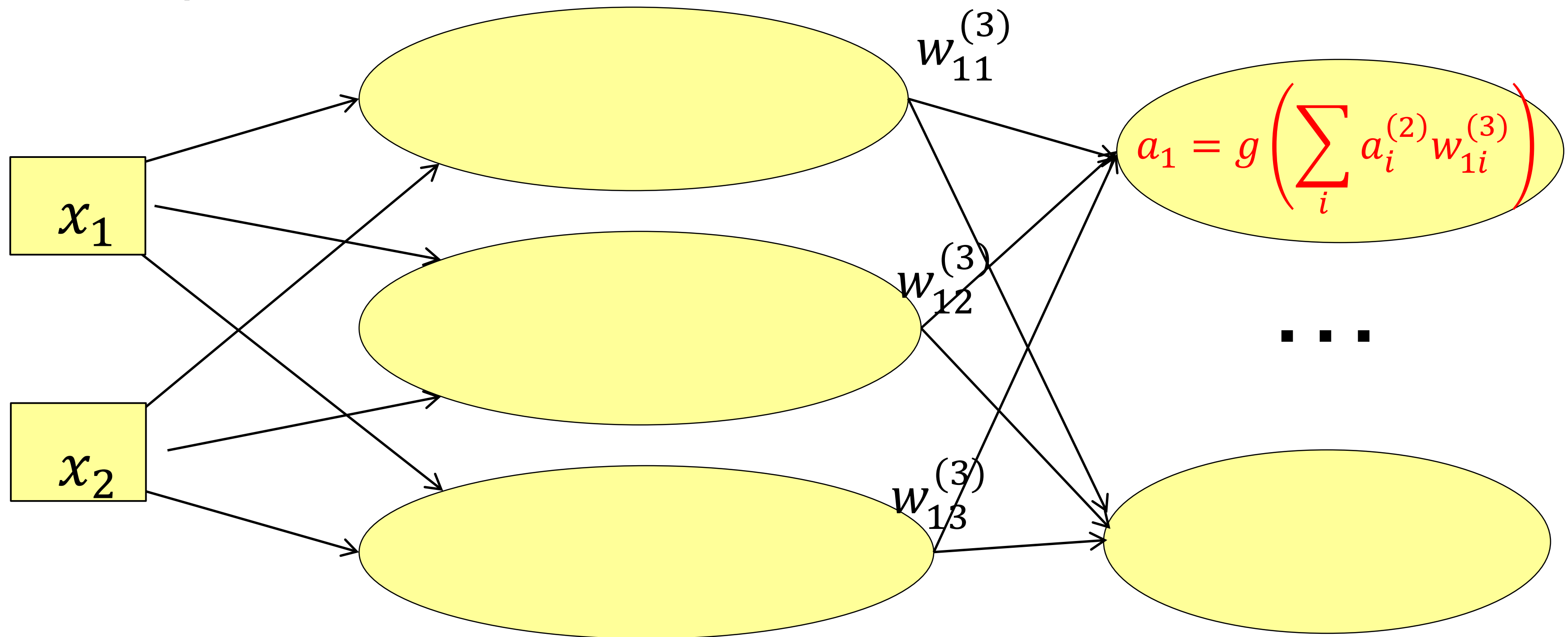
Multi-layer Perceptron: neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



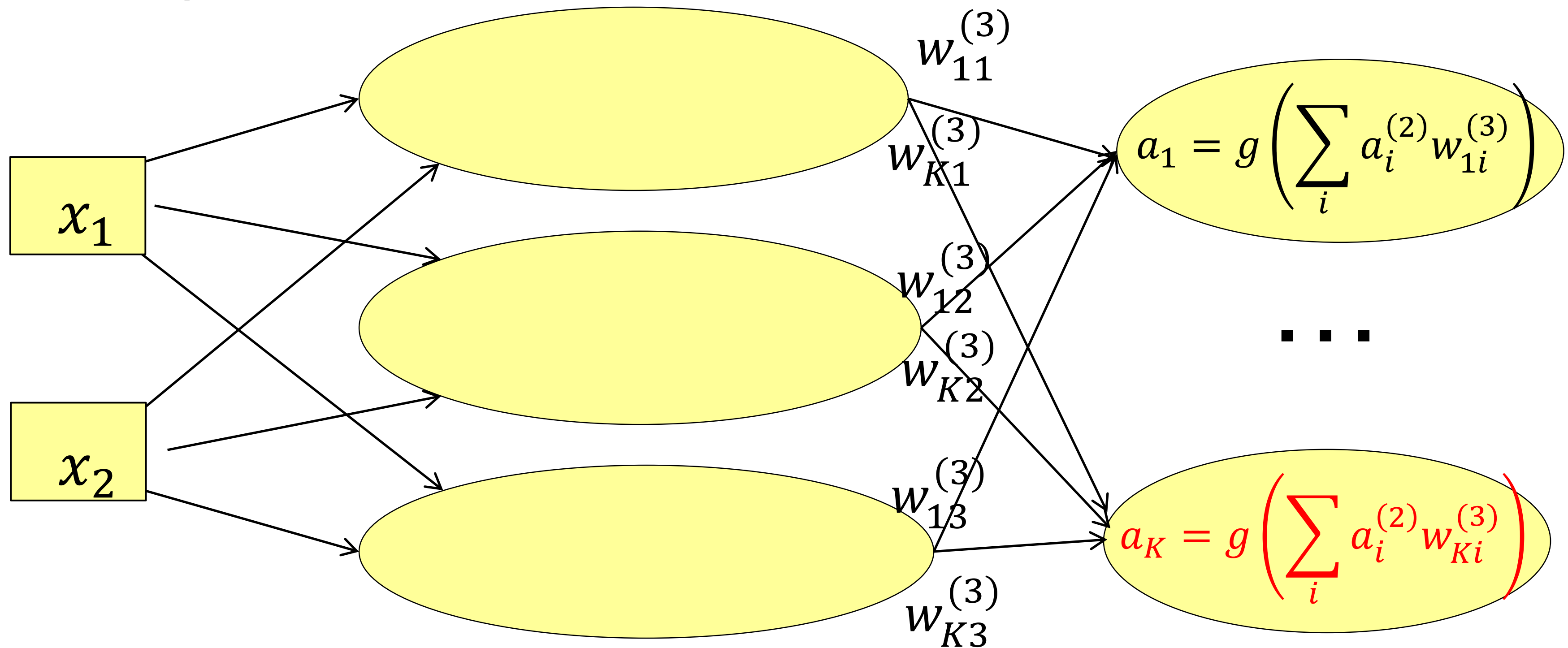
Neural net for K-way classification

- Use K output units
- Training: encode a label y by an indicator vector
  - class1=(1,0,0,…,0), class2=(0,1,0,…,0) etc.
- Test: choose the class corresponding to the largest output unit



Neural net for K-way classification

- Use K output units
- Training: encode a label y by an indicator vector
  - class1=(1,0,0,…,0), class2=(0,1,0,…,0) etc.
- Test: choose the class corresponding to the largest output unit



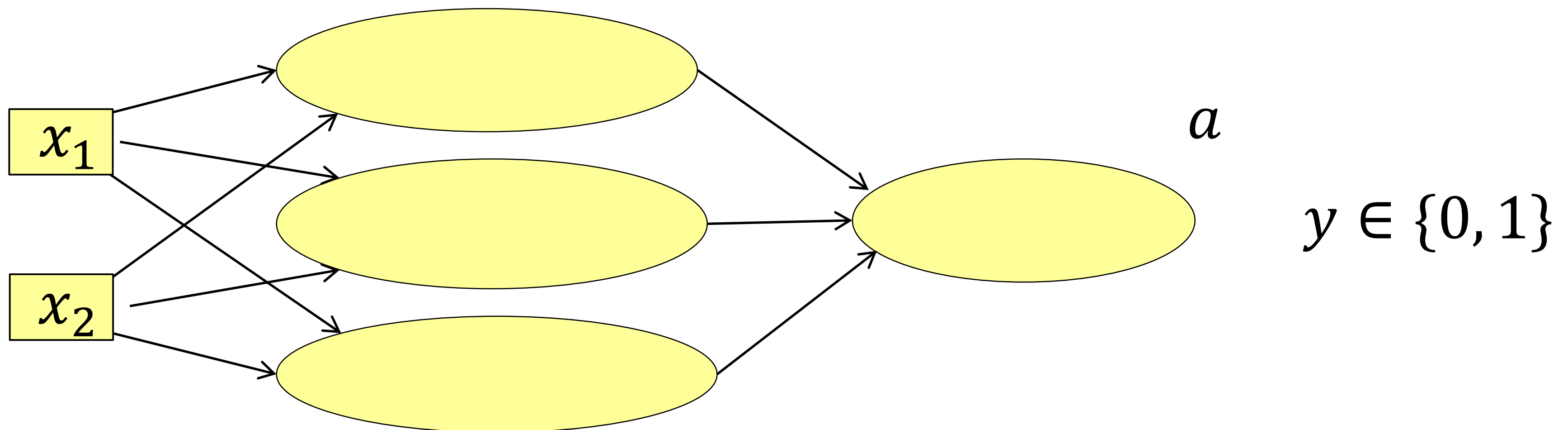
The (unlimited) power of neural network

- In theory
  - We don't need too many layers:
  - 1-hidden-layer net with enough hidden units can represent any continuous function of the inputs with arbitrary accuracy
  - 2-hidden-layer net can even represent discontinuous functions

- In practice
  - A neural network often has many layers (e.g., 50)
  - Each layer has many hidden units (hundreds/thousands)

Learning in neural network

- Consider the XOR problem again
  - x: one training point (2-dim vector) in the training set D
  - a: the output for the training point x
  - y: the binary label for x

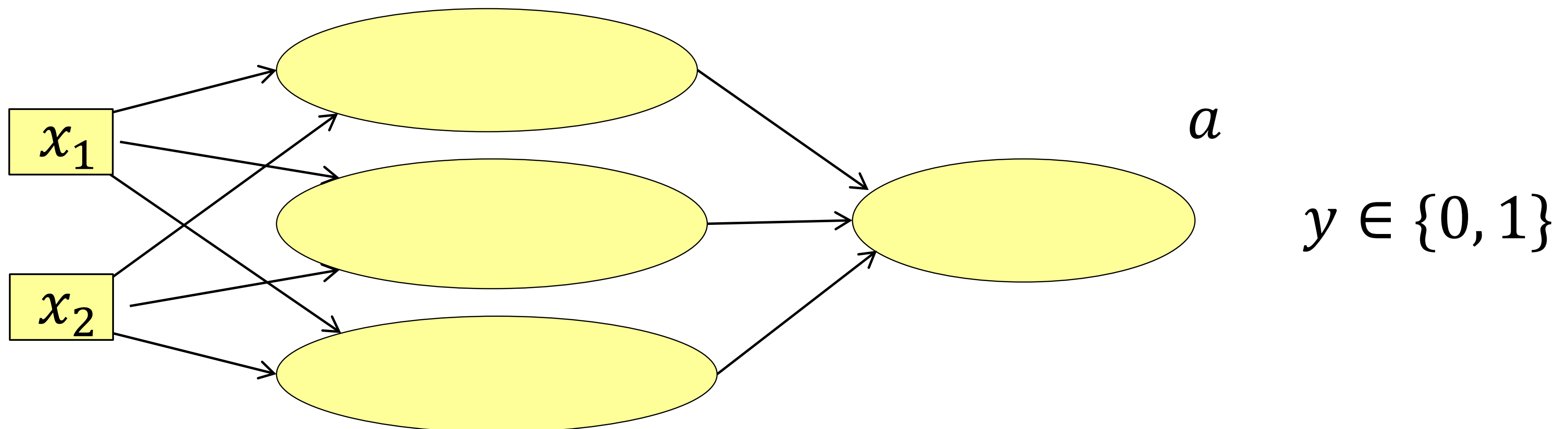


Learning in neural network

- Learning by matching the output a to the label y
- We want $a \to 1$ when $y = 1$, and $a \to 0$ when $y = 0$
- Define a loss function (similar to an error function)

$$E = \frac{1}{|D|} \sum_{x \in D} E_x \qquad E_x = -(y \log(a) + (1 - y) \log(1 - a))$$

  - $a \to 1$ and $y = 1$, $a \to 0$ and $y = 0$, $E_x \to 0$
  - $a \to 0$ and $y = 1$, $a \to 1$ and $y = 0$, $E_x \to +\infty$



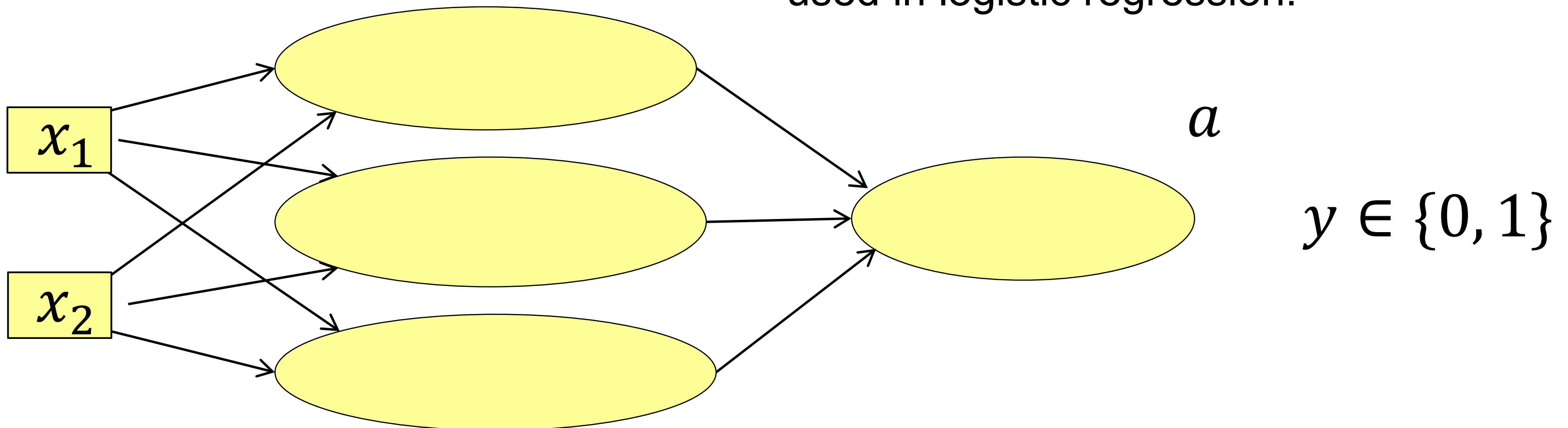
Learning in neural network

- Learning by matching the output a to the label y
- We want $a \to 1$ when $y = 1$, and $a \to 0$ when $y = 0$
- Define a loss function (similar to an error function)

$$E = \frac{1}{|D|} \sum_{x \in D} E_x \qquad E_x = -(y \log(a) + (1 - y) \log(1 - a))$$

(Binary) Cross Entropy Loss

*When a is the output of a sigmoid function, this is the same as the loss used in logistic regression!

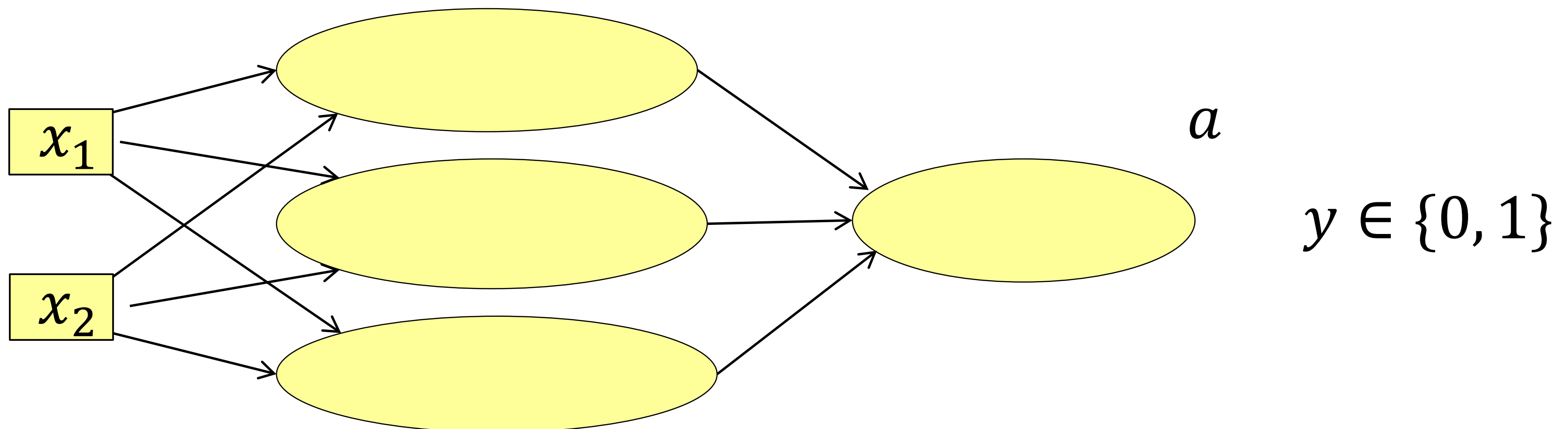


Learning in neural network

- Minimize the binary cross entropy loss

$$E = \frac{1}{|D|} \sum_{x \in D} E_x \qquad E_x = -(y \log(a) + (1-y) \log(1-a))$$

  - x: one training point (2-dim vector) in the training set D
  - a: the output for the training point x
  - y: the binary label for x



Learning in neural network

- Minimize the binary cross entropy loss

$$E = \frac{1}{|D|} \sum_{x \in D} E_x \qquad E_x = -(y \log(a) + (1-y)\log(1-a))$$

  - x: one training point (2-dim vector) in the training set D
  - a: the output for the training point x
  - y: the binary label for x
- Our variables are all the weights w on all the edges
  - Apparent difficulty: how to update the weights for the hidden units?

Learning in neural network

- Minimize the binary cross entropy loss

$$E = \frac{1}{|D|} \sum_{x \in D} E_x \qquad E_x = -(y \log(a) + (1-y) \log(1-a))$$

  - x: one training point (2-dim vector) in the training set D
  - a: the output for the training point x
  - y: the binary label for x
- Our variables are all the weights w on all the edges
  - Apparent difficulty: how to update the weights for the hidden units?
  - It turns out to be OK: we can still do gradient descent. The trick you need is the chain rule
  - The algorithm is known as back-propagation

Gradient descent

- Select a learning rate $\alpha > 0$
- Initialize the model parameters (edge weights) $w^{(0)}$
- For $t = 1, 2, \dots$
  - Compute $\frac{\partial E_x}{\partial w}$ (per sample gradients w.r.t. w) for $\mathrm{x} \in D$
  - Update the parameters

$$w^{(t)} = w^{(t-1)} - \alpha \frac{1}{|D|} \sum_{x \in D} \frac{\partial E_x}{\partial w}$$

- Repeat until E converges

Mini-batch stochastic gradient descent

- Select a learning rate $\alpha > 0$
- Initialize the model parameters (edge weights) $w^{(0)}$
- For $t = 1, 2, \ldots$
  - Randomly sample a subset $\hat{D}$ from D
  - Compute $\frac{\partial E_x}{\partial w}$ (per sample gradients w.r.t. w) for $\mathrm{x} \in \hat{D}$
  - Update the parameters

$$w^{(t)} = w^{(t-1)} - \alpha \frac{1}{|\hat{D}|} \sum_{x \in \hat{D}} \frac{\partial E_x}{\partial w}$$

- Repeat until E converges

The key challenge is to compute $\frac{\partial E_x}{\partial w}$!

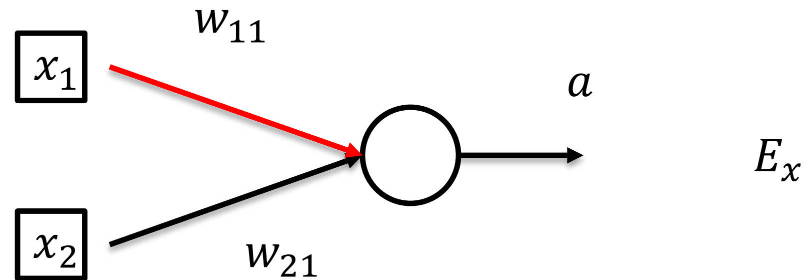
Demo: Learning XOR using neural net

- https://playground.tensorflow.org/

Credits: TensorFlow Team @ Google

Gradient (on one data point)

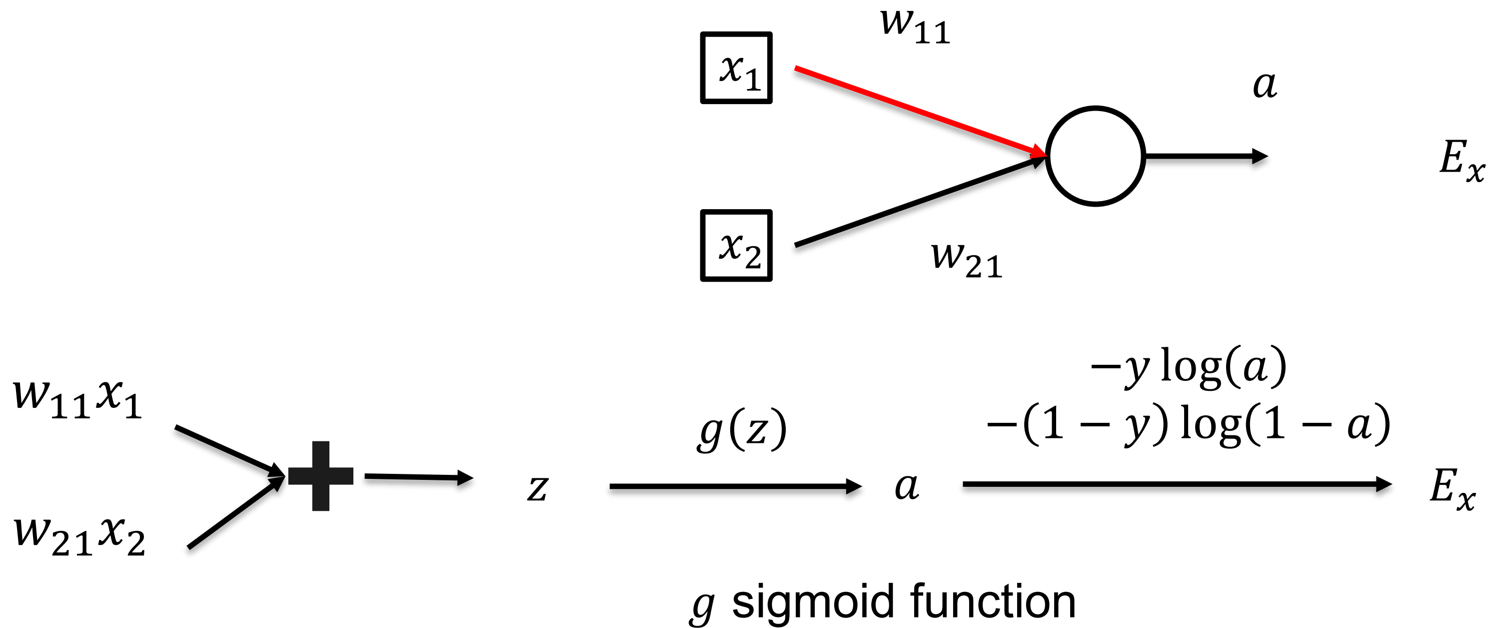
- Again, start with a single Perceptron



want to compute $\frac{\partial E_x}{\partial w_{11}}$

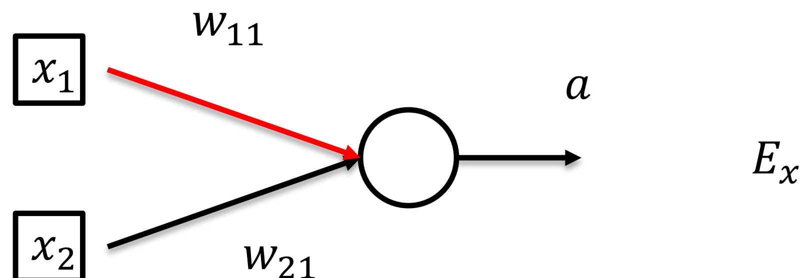
Gradient (on one data point)

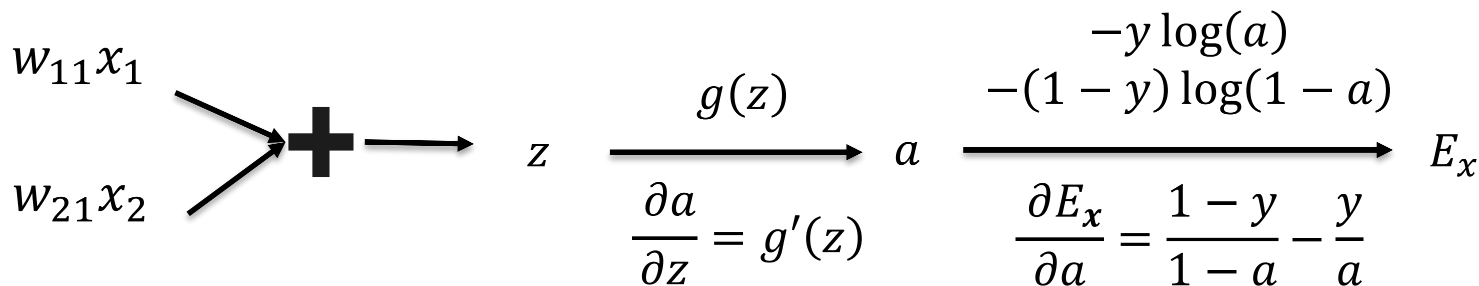
- Again, start with a single Perceptron



Gradient (on one data point)

- Again, start with a single Perceptron

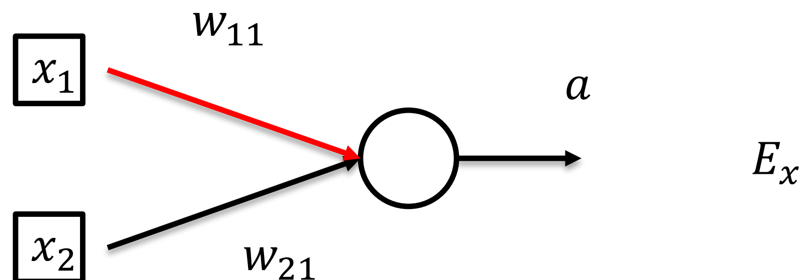


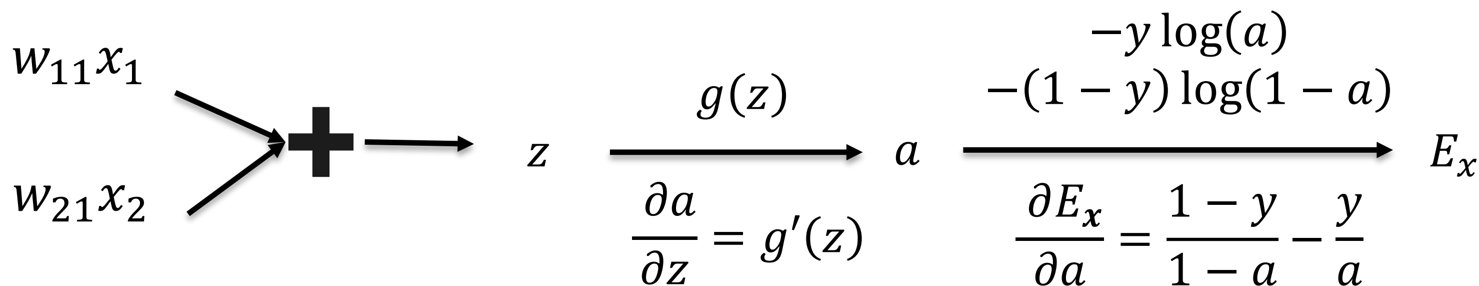


By Chain Rule: $$\frac{\partial E_x}{\partial w_{11}} = \frac{\partial E_x}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w_{11}}$$

Gradient (on one data point)

- Again, start with a single Perceptron

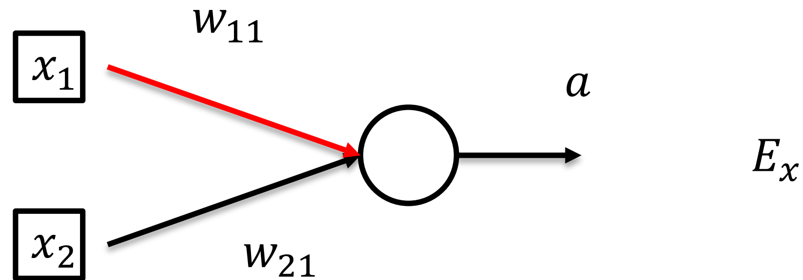


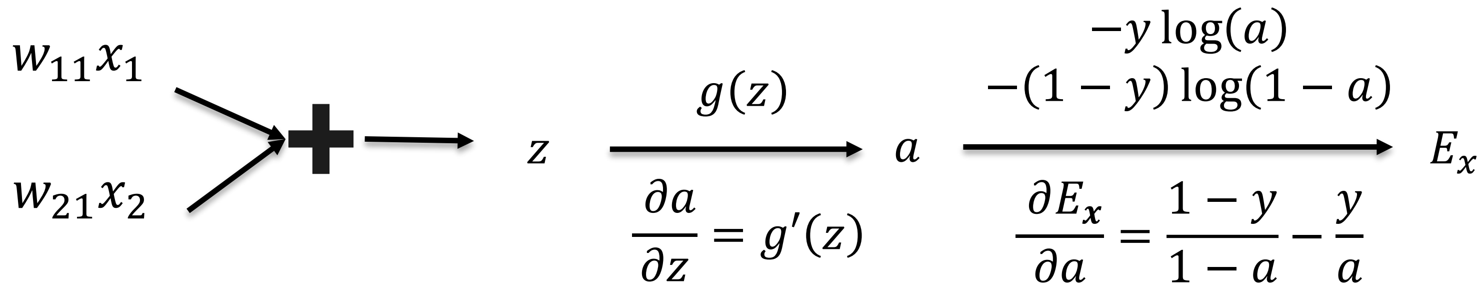


By Chain Rule: $$\frac{\partial E_x}{\partial w_{11}} = \frac{\partial E_x}{\partial a} \frac{\partial a}{\partial z} x_1$$

Gradient (on one data point)

- Again, start with a single Perceptron





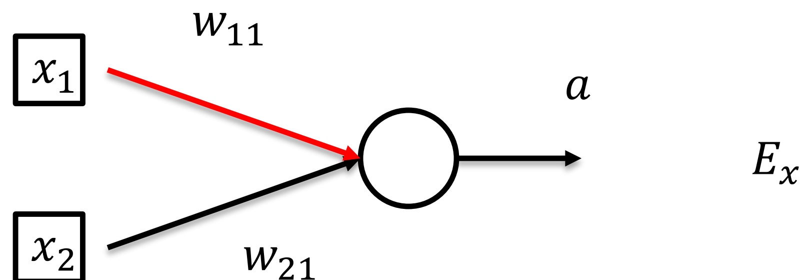
g sigmoid function

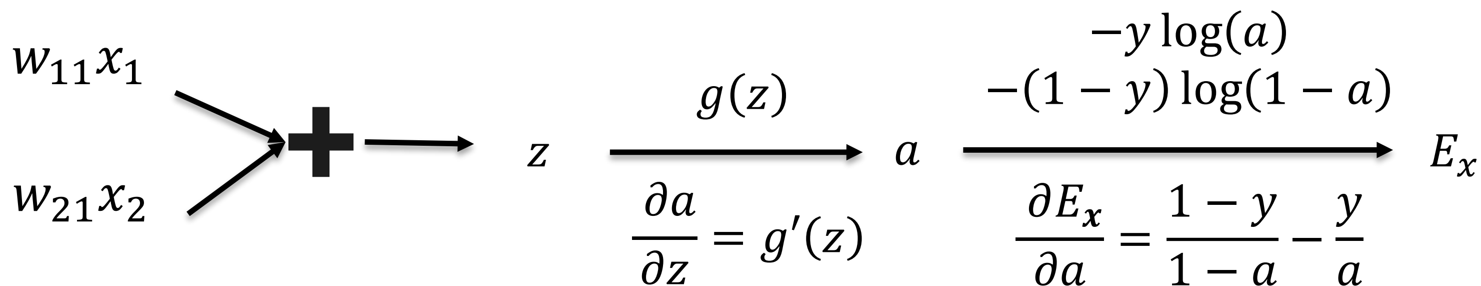
$g'(x) = g(x)(1-g(x))$

By Chain Rule: $\frac{\partial E_x}{\partial w_{11}} = \frac{\partial E_x}{\partial a} a(1-a)x_1$

Gradient (on one data point)

- Again, start with a single Perceptron



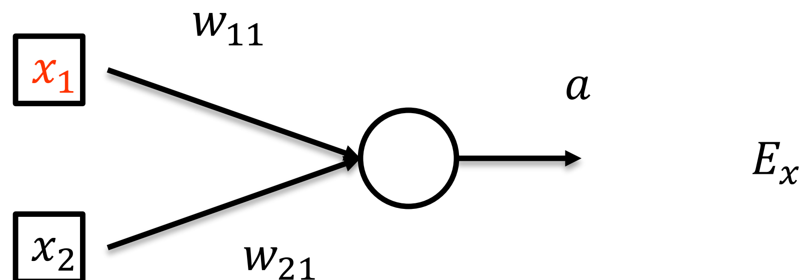


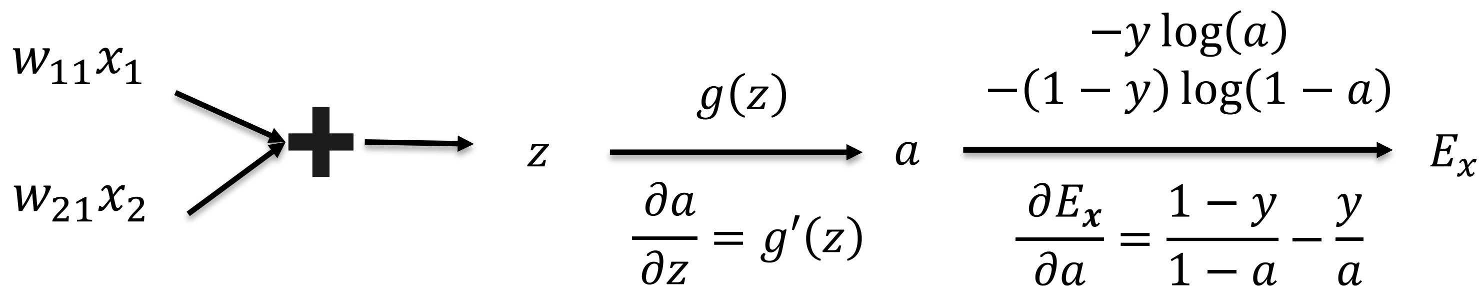
By Chain Rule:

$$\frac{\partial E_{\boldsymbol{x}}}{\partial w_{11}} = \left(\frac{1-y}{1-a} - \frac{y}{a}\right) a(1-a)x_1 = (a-y)x_1$$

Gradient (on one data point)

- Again, start with a single Perceptron



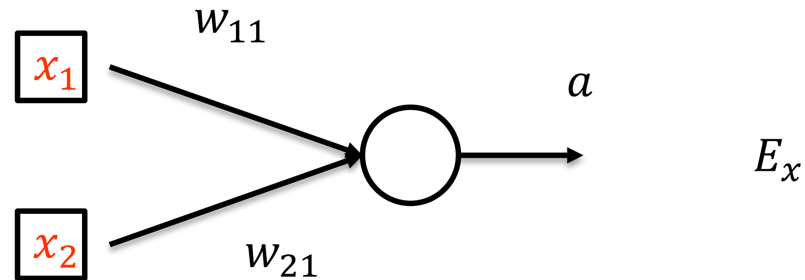


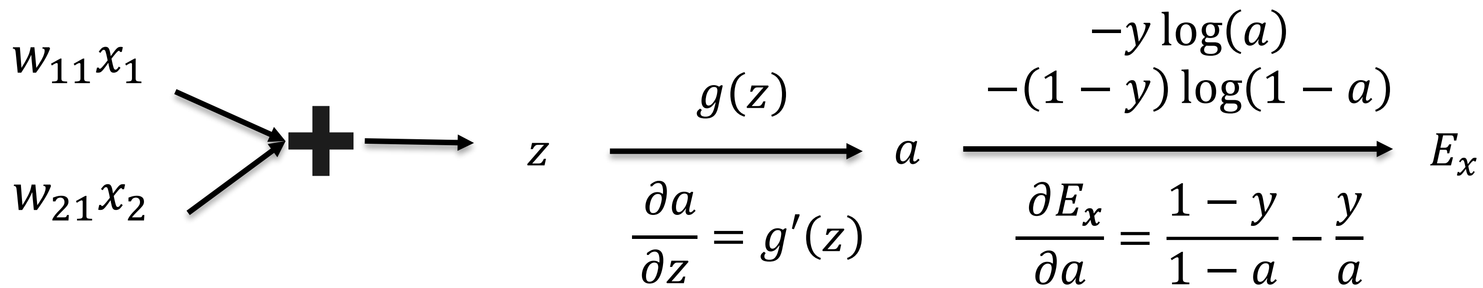
By Chain Rule:

$$\frac{\partial E_x}{\partial x_1} = \frac{\partial E_x}{\partial a}\frac{\partial a}{\partial z}w_{11} = (a-y)w_{11}$$

Gradient (on one data point)

- Again, start with a single Perceptron

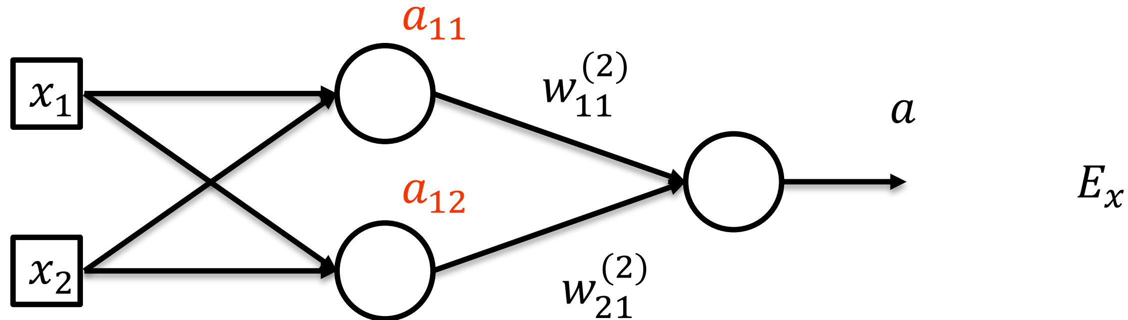


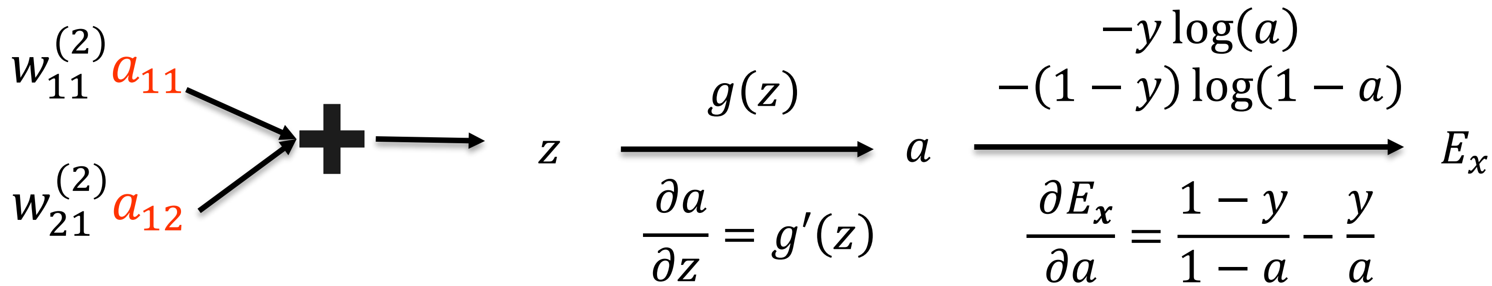


By Chain Rule: $\frac{\partial E_x}{\partial x_1} = (a-y)w_{11}, \; \frac{\partial E_x}{\partial x_2} = (a-y)w_{21}$

Gradient (on one data point)

- Now, make it deeper

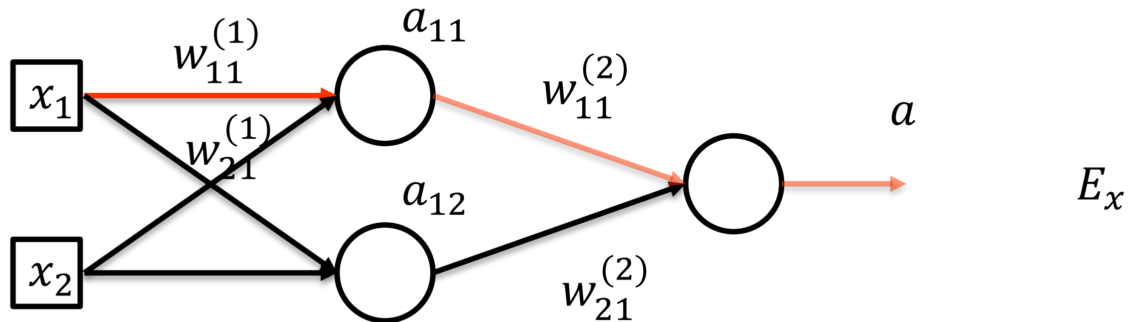


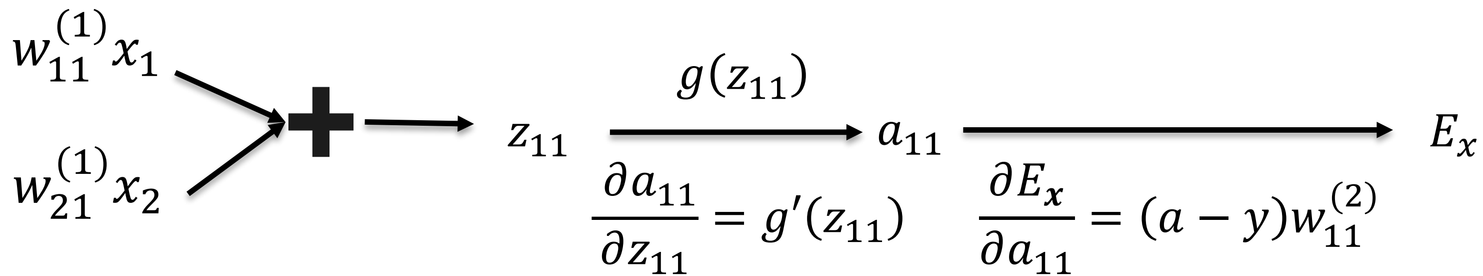


By Chain Rule: $\frac{\partial E_x}{\partial a_{11}} = (a-y)w_{11}^{(2)}$, $\frac{\partial E_x}{\partial a_{12}} = (a-y)w_{21}^{(2)}$

Gradient (on one data point)

- Now, make it deeper

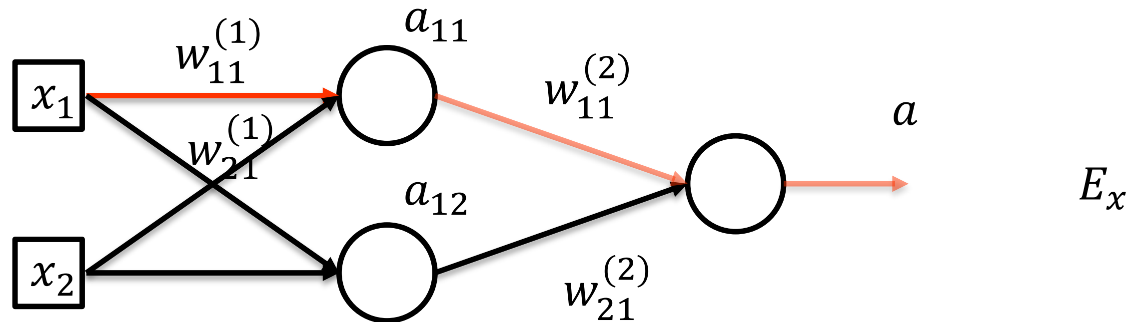


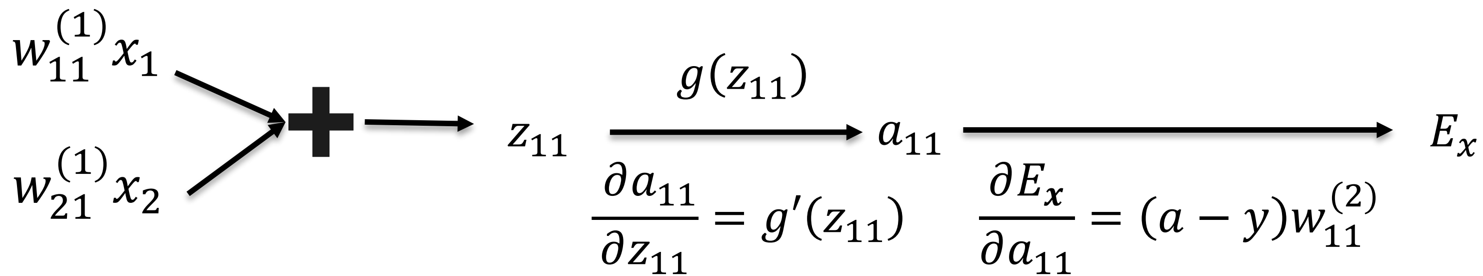


By Chain Rule: $$\frac{\partial E_x}{\partial w_{11}} = \frac{\partial E_x}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (a-y)w_{11}^{(2)} \frac{\partial a_{11}}{\partial w_{11}^{(1)}}$$

Gradient (on one data point)

- Now, make it deeper

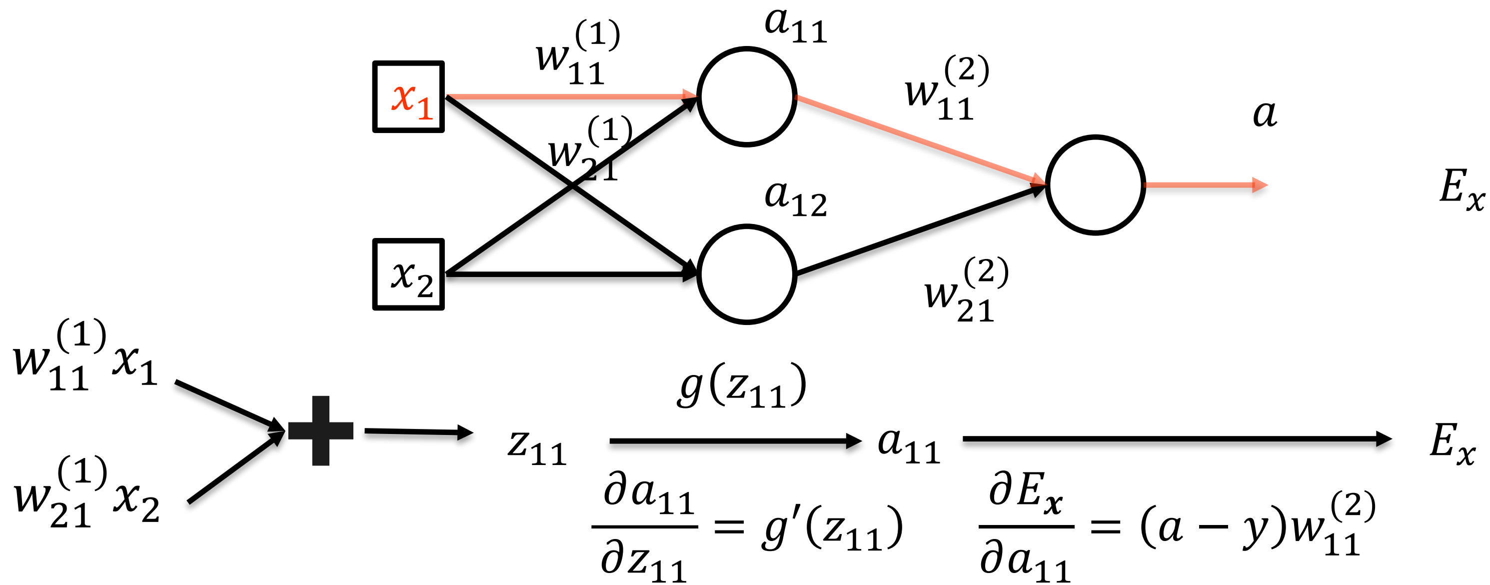




By Chain Rule:
$$\frac{\partial E_x}{\partial w_{11}} = \frac{\partial E_x}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (a-y)w_{11}^{(2)} a_{11}(1-a_{11})x_1$$

Gradient (on one data point)

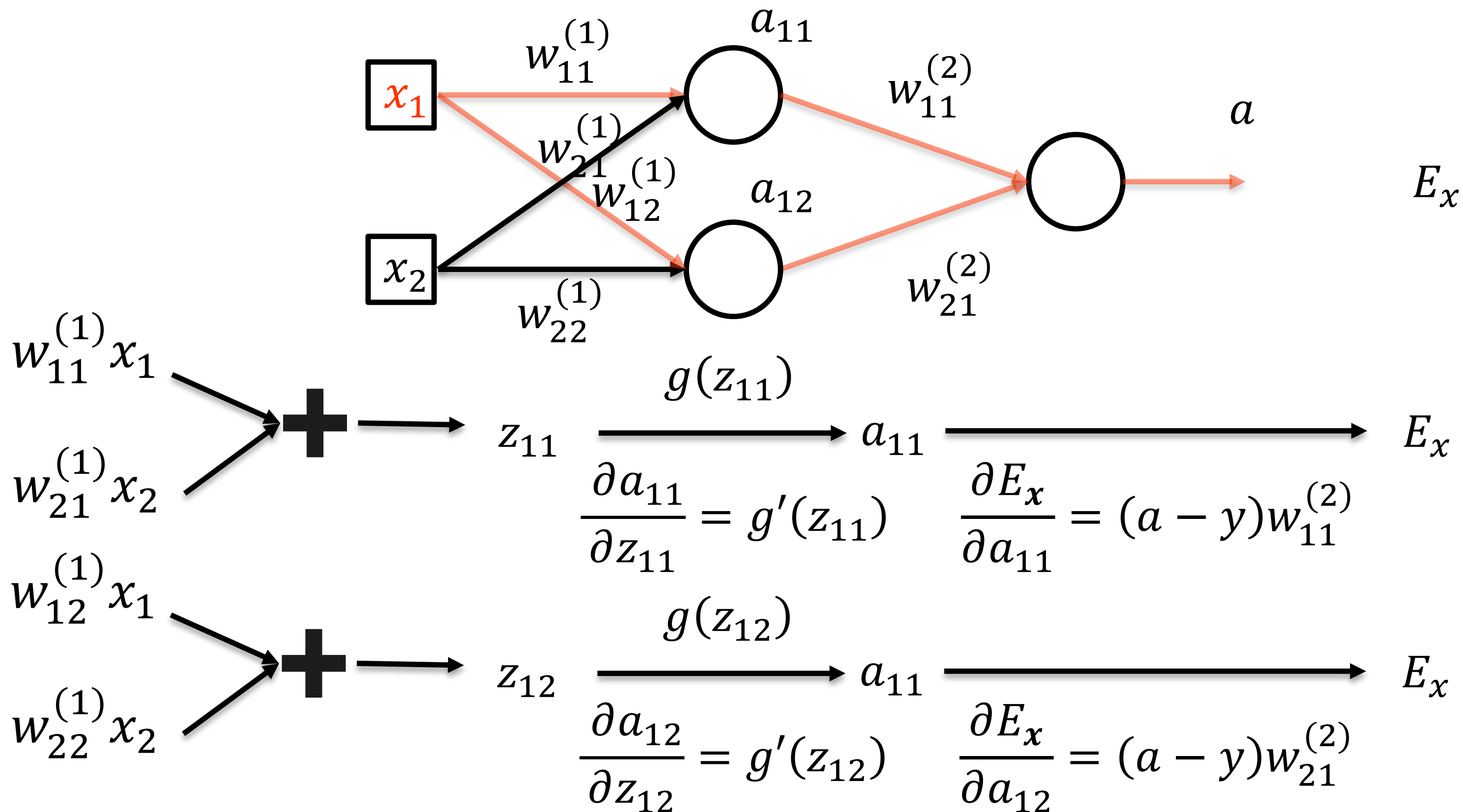
- Now, make it deeper



By Chain Rule: $$\frac{\partial E_x}{\partial x_1} = \frac{\partial E_x}{\partial a_{11}} \frac{\partial a_{11}}{\partial x_1} \; ?$$

Gradient (on one data point)

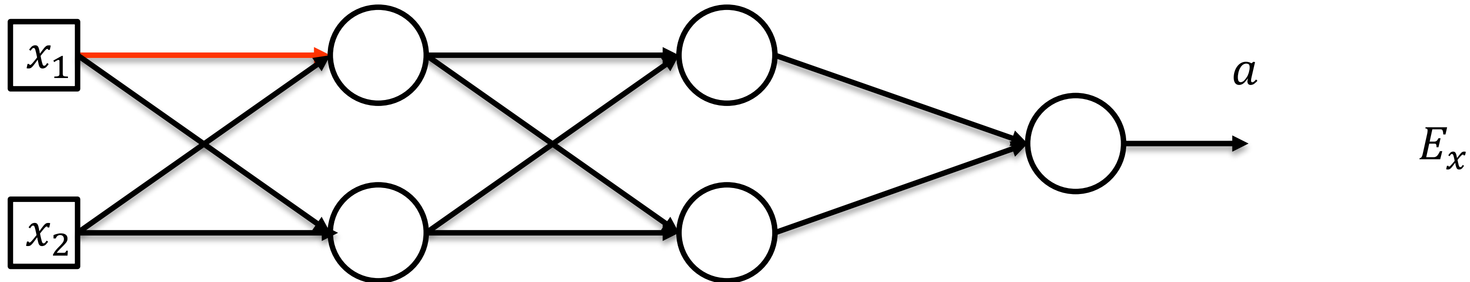
- Now, make it deeper



By Chain Rule:

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

Gradient (on one data point)



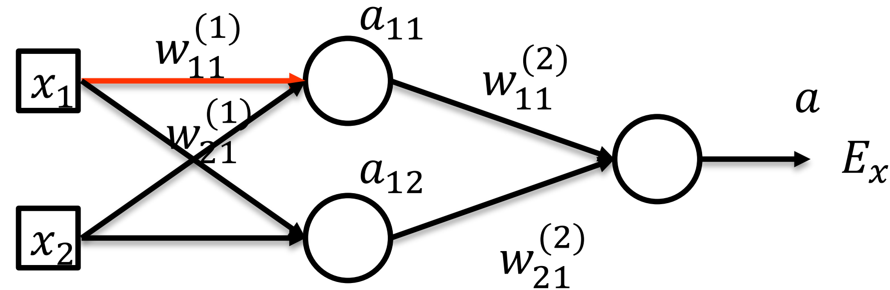
want to compute $\frac{\partial E_x}{\partial w}$ …

Getting tedious?

Back-propagation

- In theory

$$\frac{\partial E_{\boldsymbol{x}}}{\partial w_{11}} = (a-y)w_{11}^{(2)}a_{11}(1-a_{11})x_1$$



- In practice
  - Define the model and the loss function
  - Select the optimization method (e.g., stochastic gradient descent)
  - Back-propagation handled by automatic differentiation (but how?)