Machine Learning

A. Hees

Quick reminder of the main idea

- construct a model to make the link between some input variables and some output variables

$$\boldsymbol{x} = (x_1, x_2, \ldots x_n) \xrightarrow{\text{Model}} \boldsymbol{y} = (y_1, y_2, \ldots y_n)$$

- based on a very large number of observations (the training set), i.e.

$$\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right)$$

- each $\boldsymbol{x}^{(i)}$ is a set of input variables corresponding to one observation and each $\boldsymbol{y}^{(i)}$ is a set of output variables corresponding to the observation related to $\boldsymbol{x}^{(i)}$.

- once the model is trained, we can use it to make predictions

Multi-Layer Neural Network

- generic form of the model used

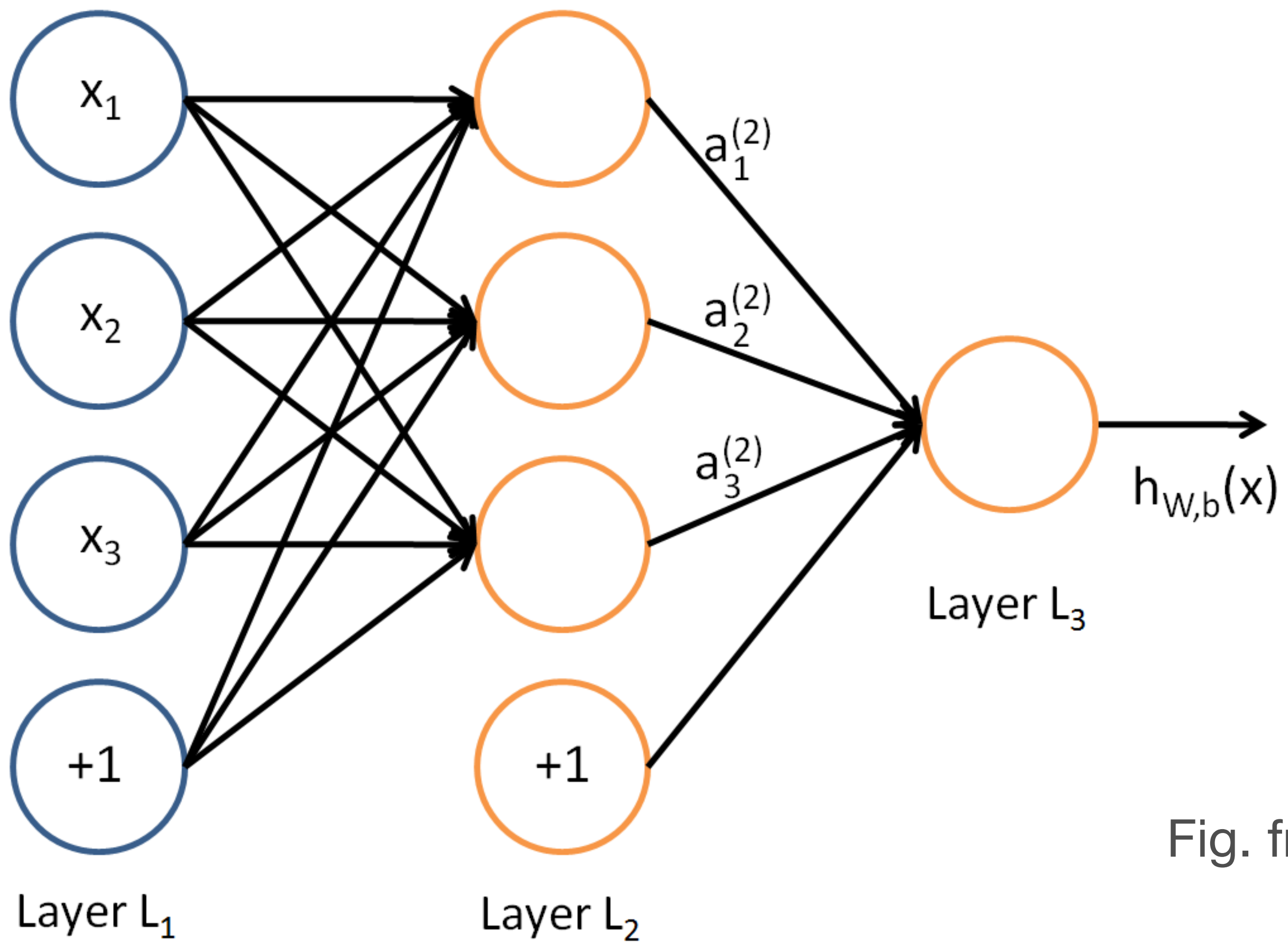


Fig. from UFLD tutorial, Standford

- each neuron is extremely simple ; the complexity of the model is coming from the network structure.

What does one neuron do?

- a neuron is a computational unit that takes a vector and return a scalar

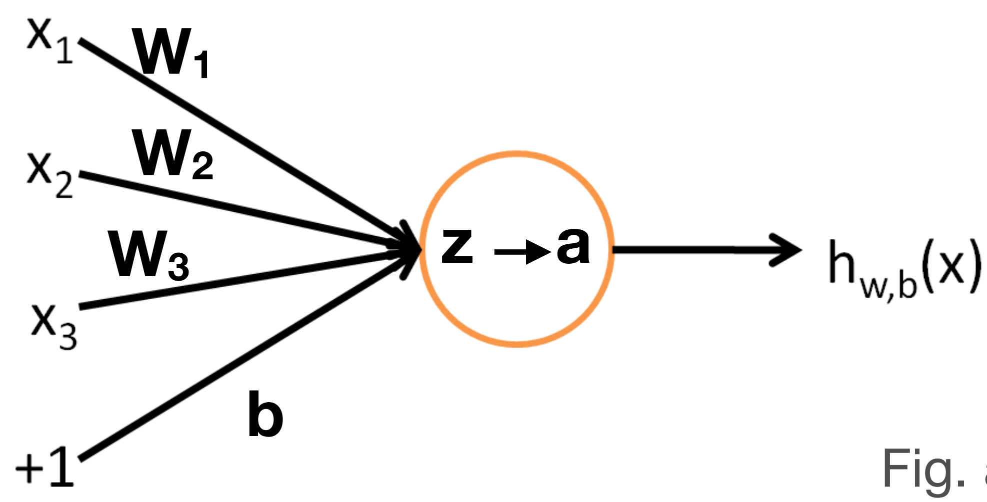


Fig. adapted from UFLD tutorial, Standford

W_i are weight

$$z = \sum_i W_i x_i + b = W_i x_i + b$$

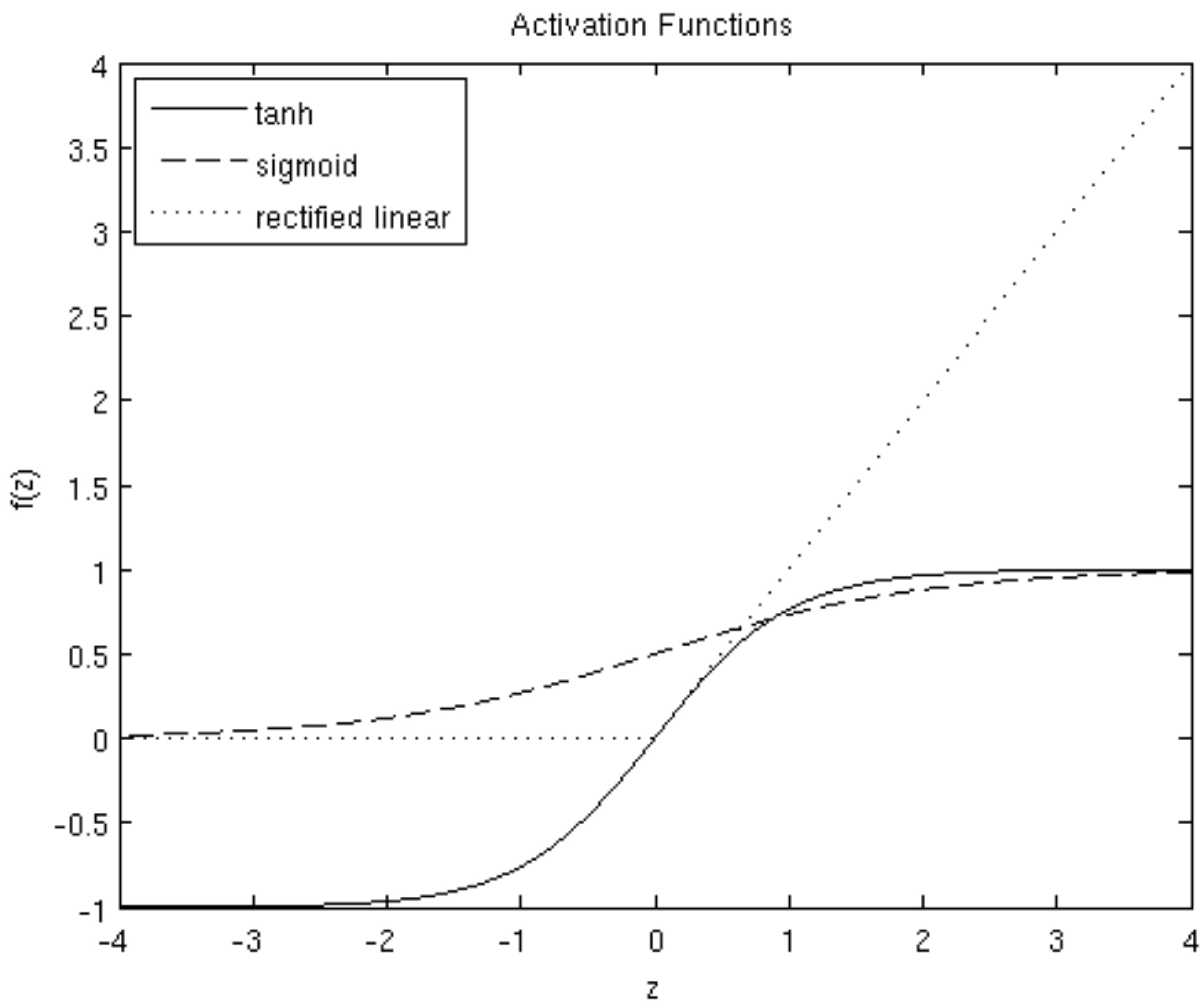
Convention: repeated indices are implicitly summed over

$$a = f(z)$$

- f(z) is the activation function
- One neuron is characterized by "n" +1 weights (n being the number of input)

The activation function

- has been discussed by Bernie 2 weeks ago



$$f(z) = \frac{1}{1 + e^{-z}}$$

Fig. from UFLD tutorial, Standford

- in practice the rectified linear often works better
- we will need the derivative of that function

Neural Network Model

- Is simply a collection of layers of neurons…

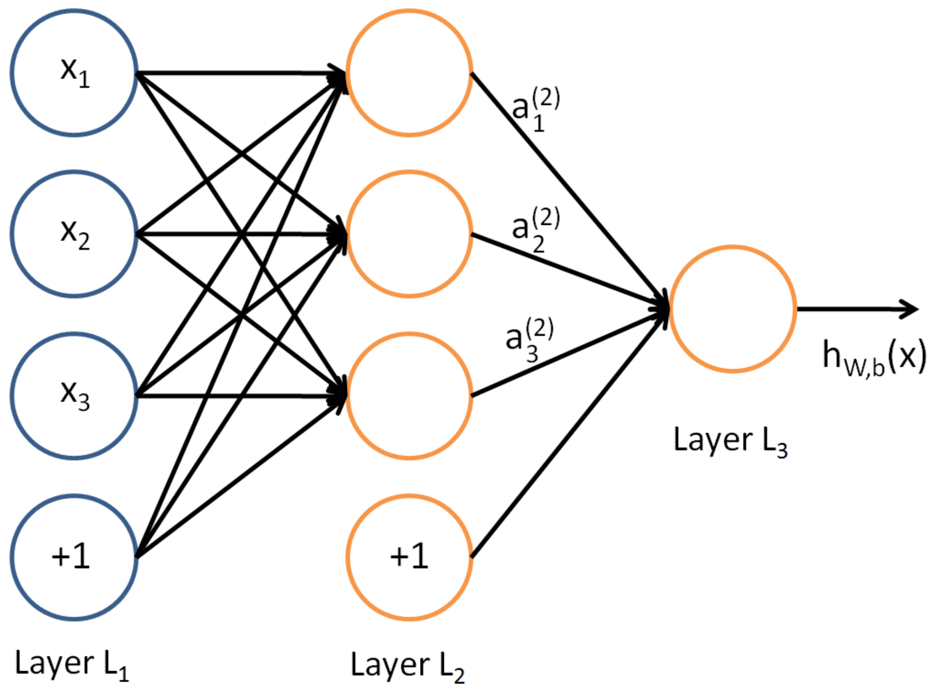


Fig. from UFLD tutorial, Standford

- Iterative description:

$$z_i^{(k)} = W_{ij}^{(k-1)} a_j^{(k-1)} + b_i^{(k-1)}$$

$$a_i^{(k)} = f(z_i^{(k)})$$

- Input layer: $a_j^{(1)} = x_j$
- Output layer: $y_i = a_i^{(n)}$ with n the number of layers

Feedforward neural network

- Each layer i has input from the layer i-1 only (no "jump" between layers)
- No loop

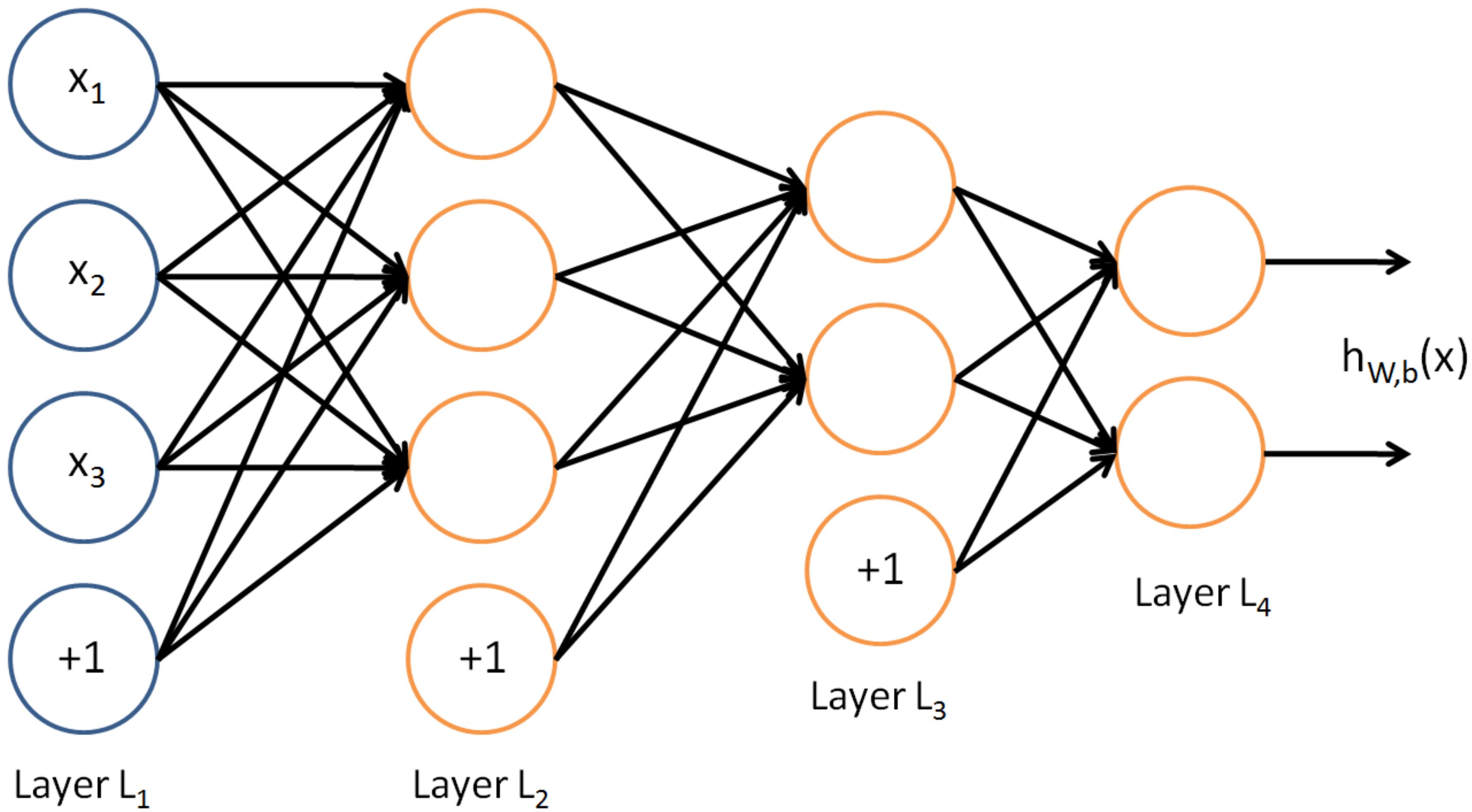


Fig. from UFLD tutorial, Standford

- Forward propagation: we apply a set of weight to the data and calculate the output.

How many parameters?

- Is simply a collection of layers of neurons…

$$z_i^{(k)} = W_{ij}^{(k-1)} a_j^{(k-1)} + b_i^{(k-1)}$$

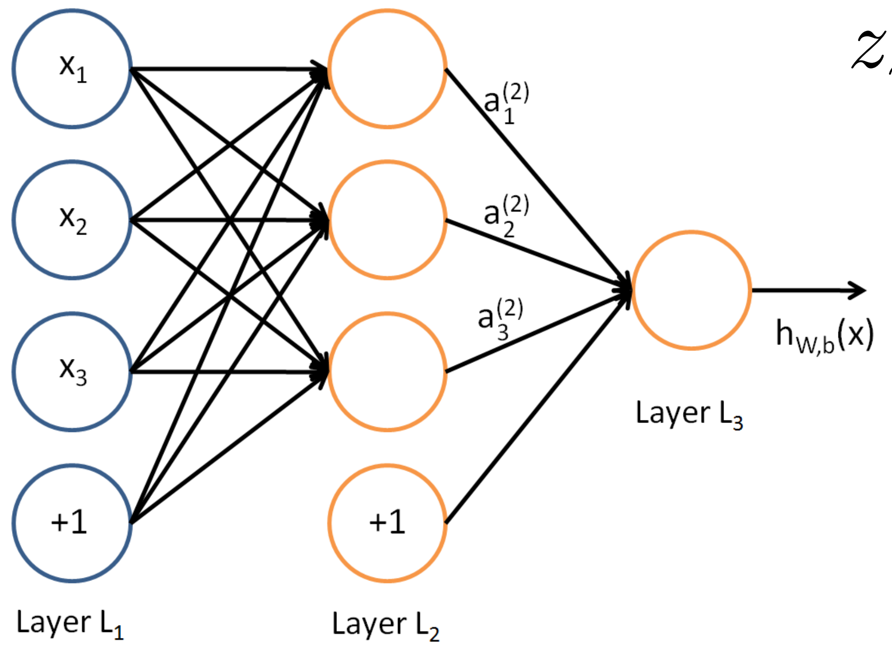


Fig. from UFLD tutorial, Standford

- For each layer: $(n_{in}+1) \times n_{neurons}$
- Total number of parameters $\sum_{i=2}^{n} n_{i-1} \times n_i + \sum_{i=2}^{n} n_i$

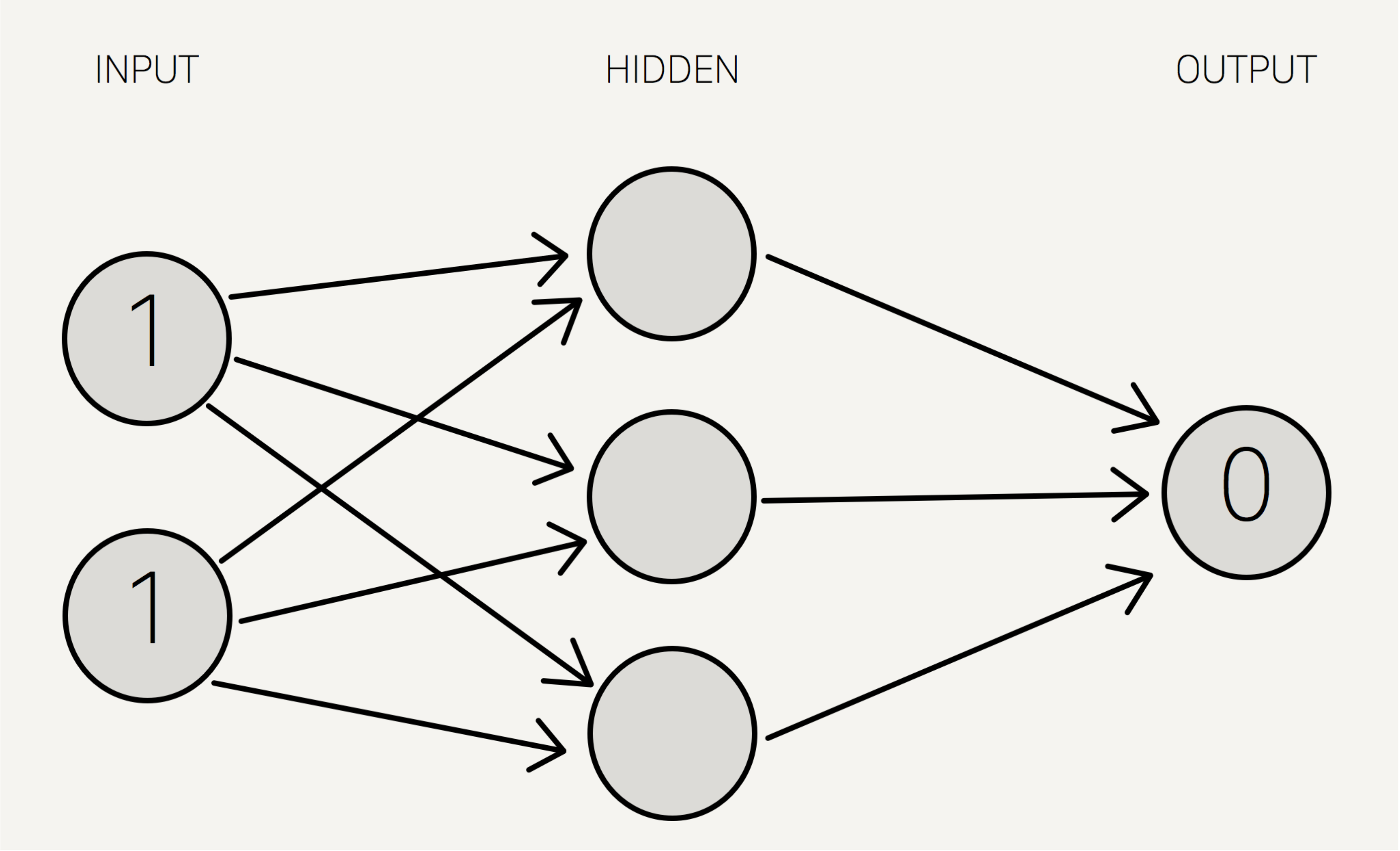
with n_i the number of neurons on layer i

- Example above: $n_1=3$, $n_2=3$, $n_3=1$. Number of parameters: 16

Example of forward propagation

- The famous XOR (cfr Bernie's talk)
- We'll use the sigmoid function $f(z) = \dfrac{1}{1+e^{-z}}$
- A 3 layers network with 3 neurons in the hidden layer

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

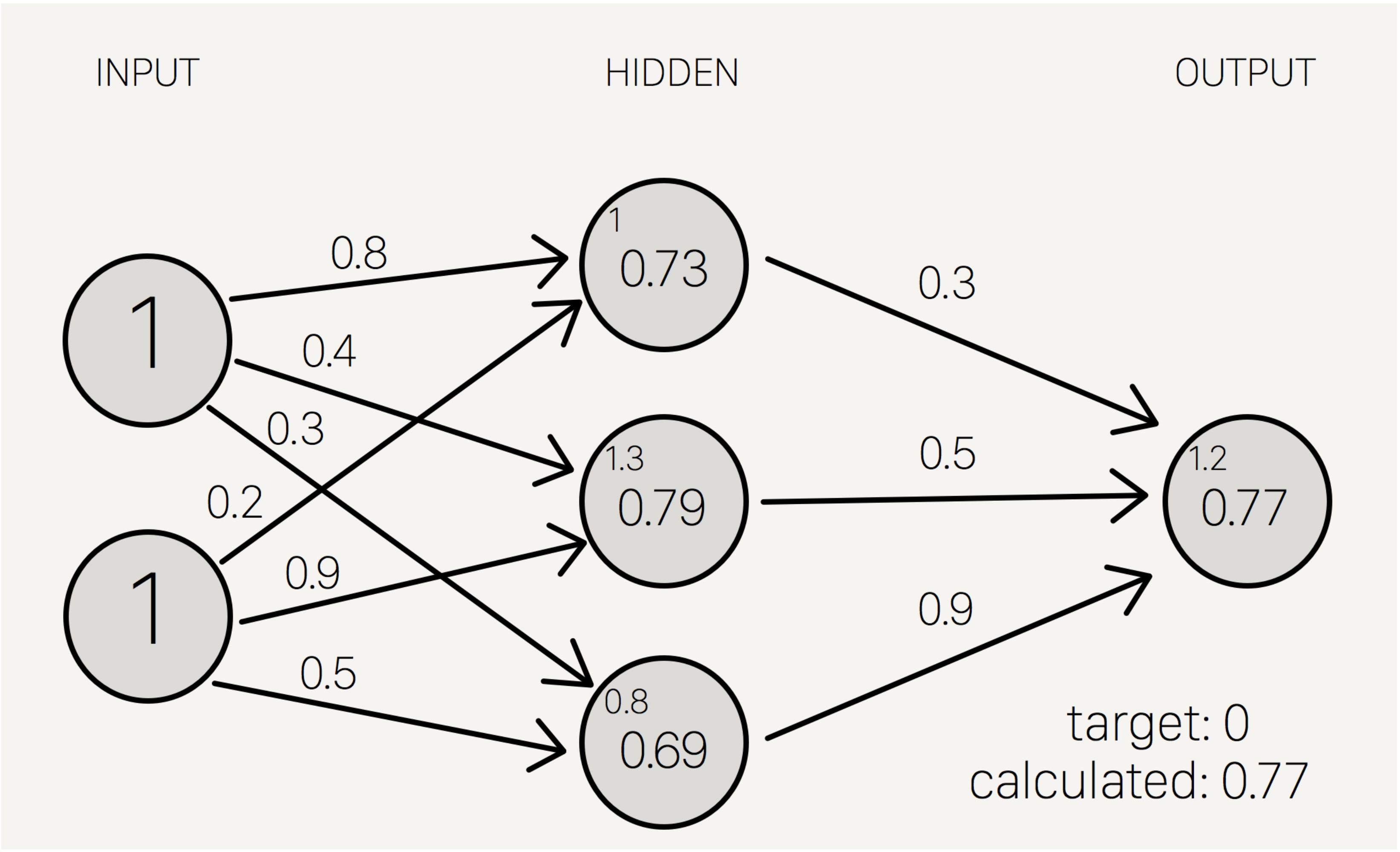


Example and fig. from S. Miller's github

Example of forward propagation

- The famous XOR (cfr Bernie's talk)
- We'll use the sigmoid function $f(z) = \dfrac{1}{1+e^{-z}}$
- A 3 layers network with 3 neurons in the hidden layer
- Assign randomly the weight and compute the output

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

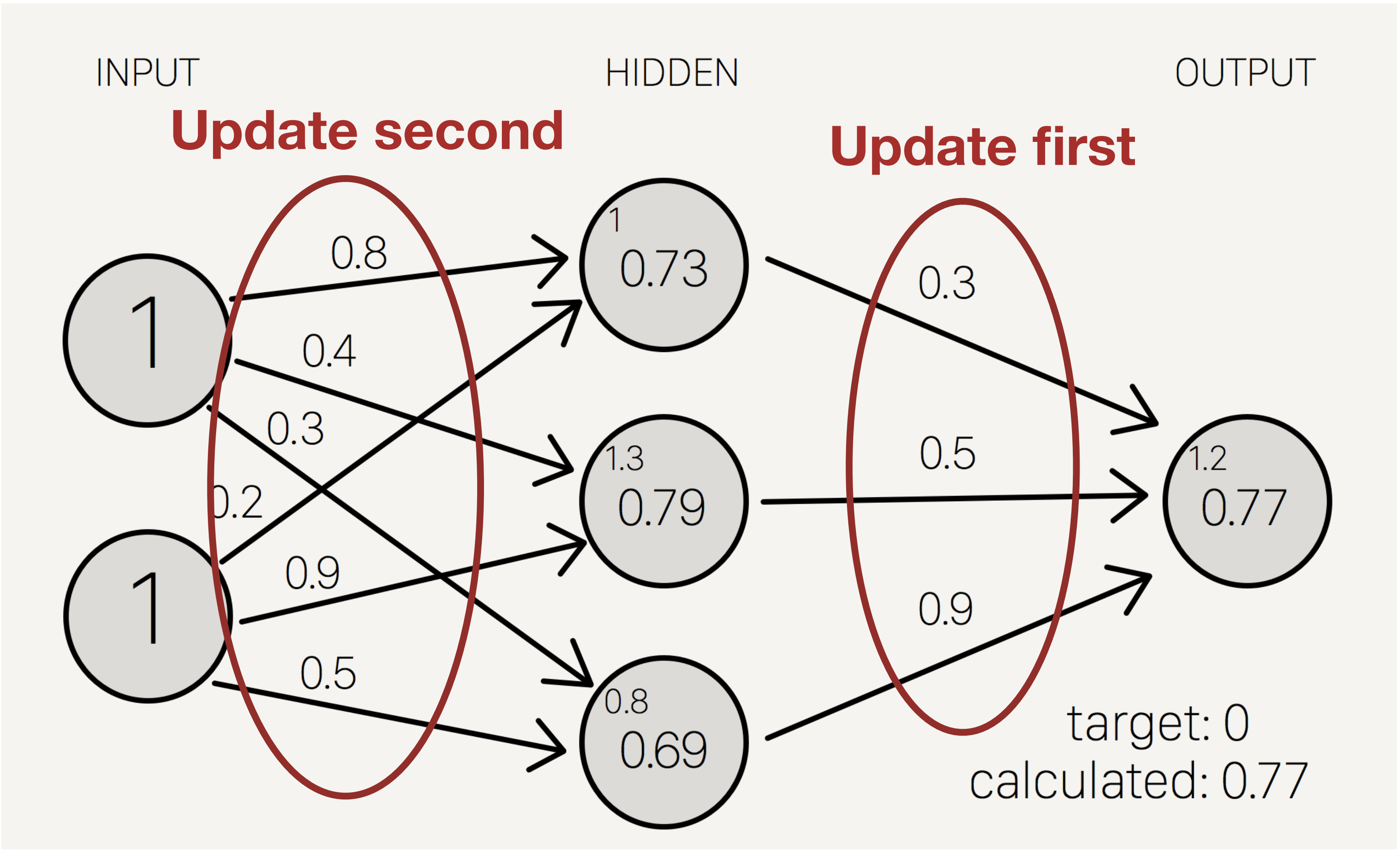


A model that is terrible

Example and fig. from S. Miller's github

All the game is to improve this

- By optimizing the weights using the observations in the training set
- Optimizing on what?
- Backpropagation algorithm: we start upgrading the weights from the end of the layers and move backward.



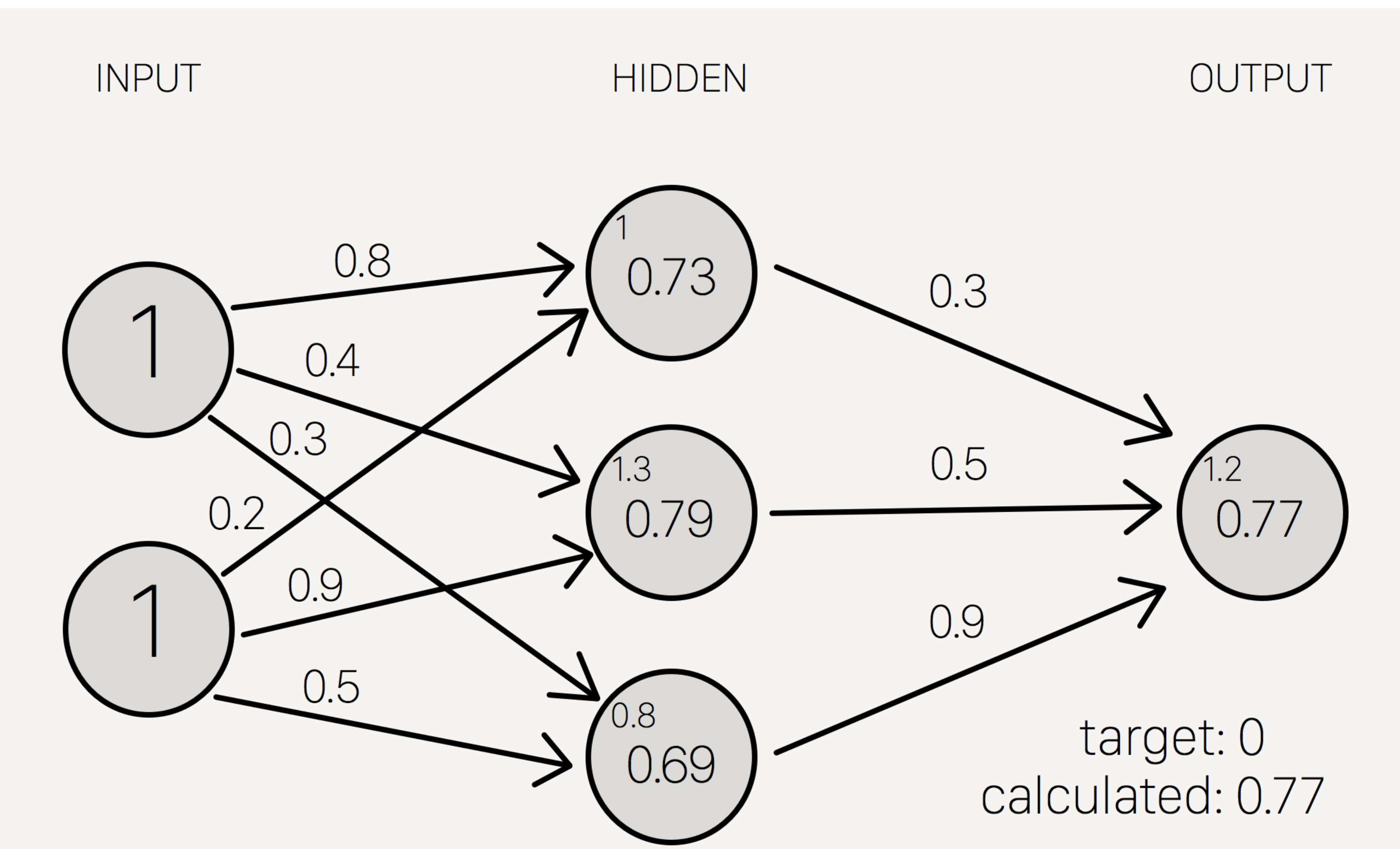
Example and fig. from S. Miller's github

Optimization on the cost function

- Most naive choice: the chi2 (but not the only one, e.g. cross entropy, …)
- chi2 for one observation:

$$J\left(\{W\},\{b\};\boldsymbol{x},\boldsymbol{y}\right)=\frac{1}{2}\left\|\boldsymbol{a}^{(n)}\left(\{W\},\{b\};\boldsymbol{x}\right)-\boldsymbol{y}\right\|^{2}$$

Prediction from the model



- J=0.296

Example and fig. from S. Miller's github

Optimization on the cost function

- Most naive choice: the chi2 (but not the only one, e.g. cross entropy, …)
- chi2 for one observation:

$$J\left(\{W\},\{b\};\boldsymbol{x},\boldsymbol{y}\right) = \frac{1}{2}\left\|\boldsymbol{a}^{(n)}\left(\{W\},\{b\};\boldsymbol{x}\right) - \boldsymbol{y}\right\|^2$$

Prediction from the model

- Total cost function

$$J\left(\{W\},\{b\}\right) = \left[\frac{1}{n_{\mathrm{obs}}}\sum_{i=1}^{n_{\mathrm{obs}}} J\left(\{W\},\{b\};\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)}\right)\right] + \frac{\lambda}{2}\sum_{l,i,j}\left(W_{ij}^{(l)}\right)^2$$

Cost function on all the training set

Regularization term: helps to prevent overfitting ; depends on lambda that needs to be chosen

Optimization on the cost function

- A simple way: Gradient descent

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial J(\{W\}, \{b\})}{\partial W_{ij}^{(l)}}$$

- And similar for the b.
- alpha is the learning rate from the Gradient descent:
  - either a small positive number
  - linear search for the smallest value
  - a lot of variation

How to compute the gradient?

- For each instance of the training set: start from the end of the network and move backward: backpropagation algorithm

$$J\left(\{W\},\{b\};\boldsymbol{x},\boldsymbol{y}\right)=\frac{1}{2}\left\|\boldsymbol{a}^{(n)}\left(\{W\},\{b\};\boldsymbol{x}\right)-\boldsymbol{y}\right\|^{2}$$

$$\boxed{\frac{\partial J}{\partial z_i^{(n)}}=+(a_i^{(n)}-y_i).f'(z_i^{(n)})}$$

$$\frac{\partial J}{\partial z_i^{(n-1)}}=\frac{\partial J}{\partial z_k^{(n)}}\frac{\partial z_k^{(n)}}{\partial z_i^{(n-1)}}=\frac{\partial J}{\partial z_k^{(n)}}W_{ki}^{(n-1)}f'(z_i^{(n-1)})$$

$$\frac{\partial J}{\partial z_i^{(n-2)}}=\frac{\partial J}{\partial z_k^{(n-1)}}\frac{\partial z_k^{(n-1)}}{\partial z_i^{(n-2)}}=\frac{\partial J}{\partial z_k^{(n-1)}}W_{ki}^{(n-2)}f'(z_i^{(n-2)})$$

$$\boxed{\frac{\partial J}{\partial z_i^{(l)}}=\frac{\partial J}{\partial z_k^{(l+1)}}W_{ki}^{(l)}f'(z_i^{(l)})}$$

How to compute the gradient?

- For each instance of the training set: start from the end of the network and move backward: backpropagation algorithm

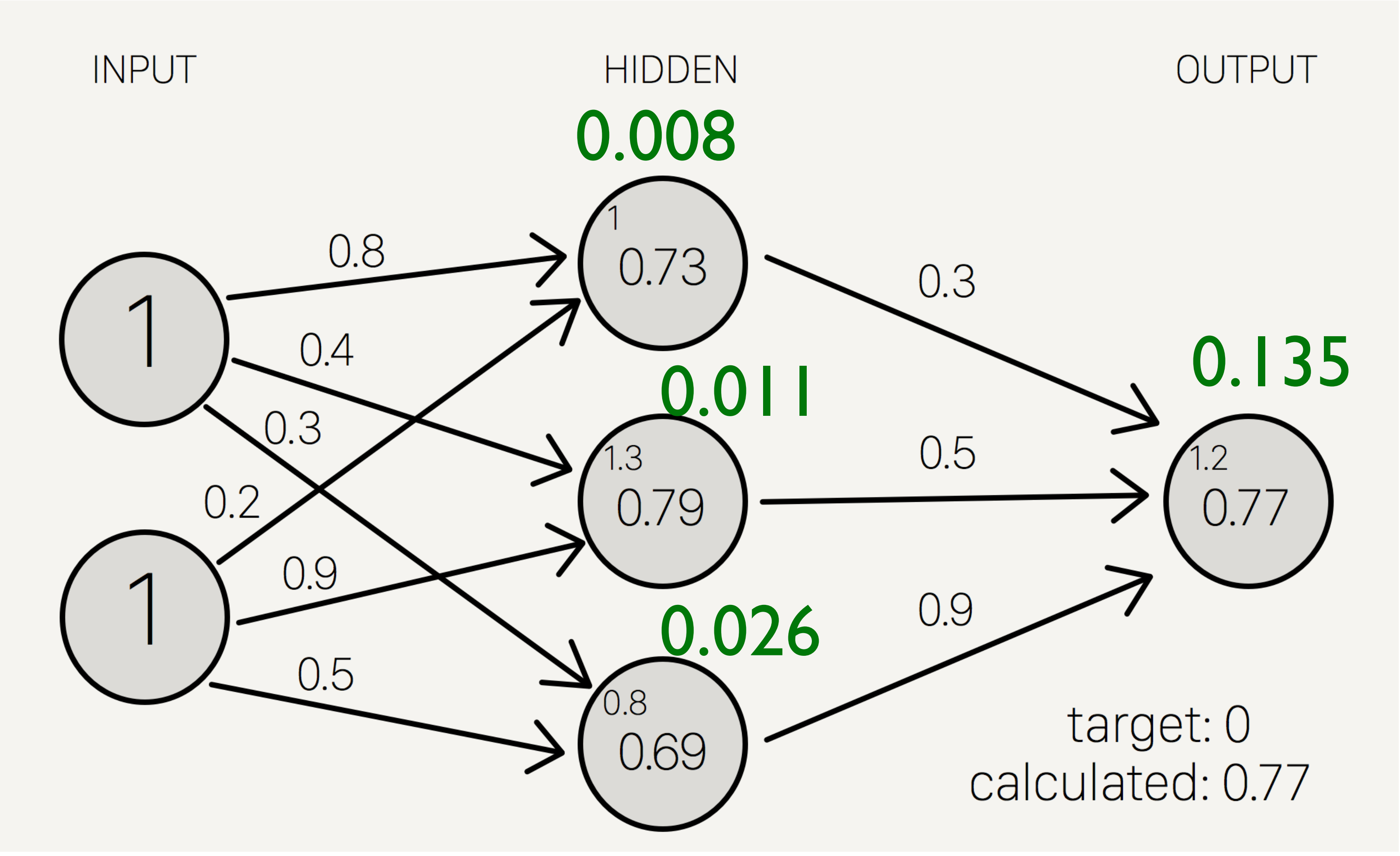
$$J\left(\{W\},\{b\};\boldsymbol{x},\boldsymbol{y}\right)=\frac{1}{2}\left\|\boldsymbol{a}^{(n)}\left(\{W\},\{b\};\boldsymbol{x}\right)-\boldsymbol{y}\right\|^{2}$$

$$\boxed{\begin{aligned}\frac{\partial J}{\partial W_{ij}^{(l)}}&=\frac{\partial J}{\partial z_i^{(l+1)}}a_j^{(l)}\\ \frac{\partial J}{\partial b_i^{(l)}}&=\frac{\partial J}{\partial z_i^{(l+1)}}\end{aligned}}$$

An example

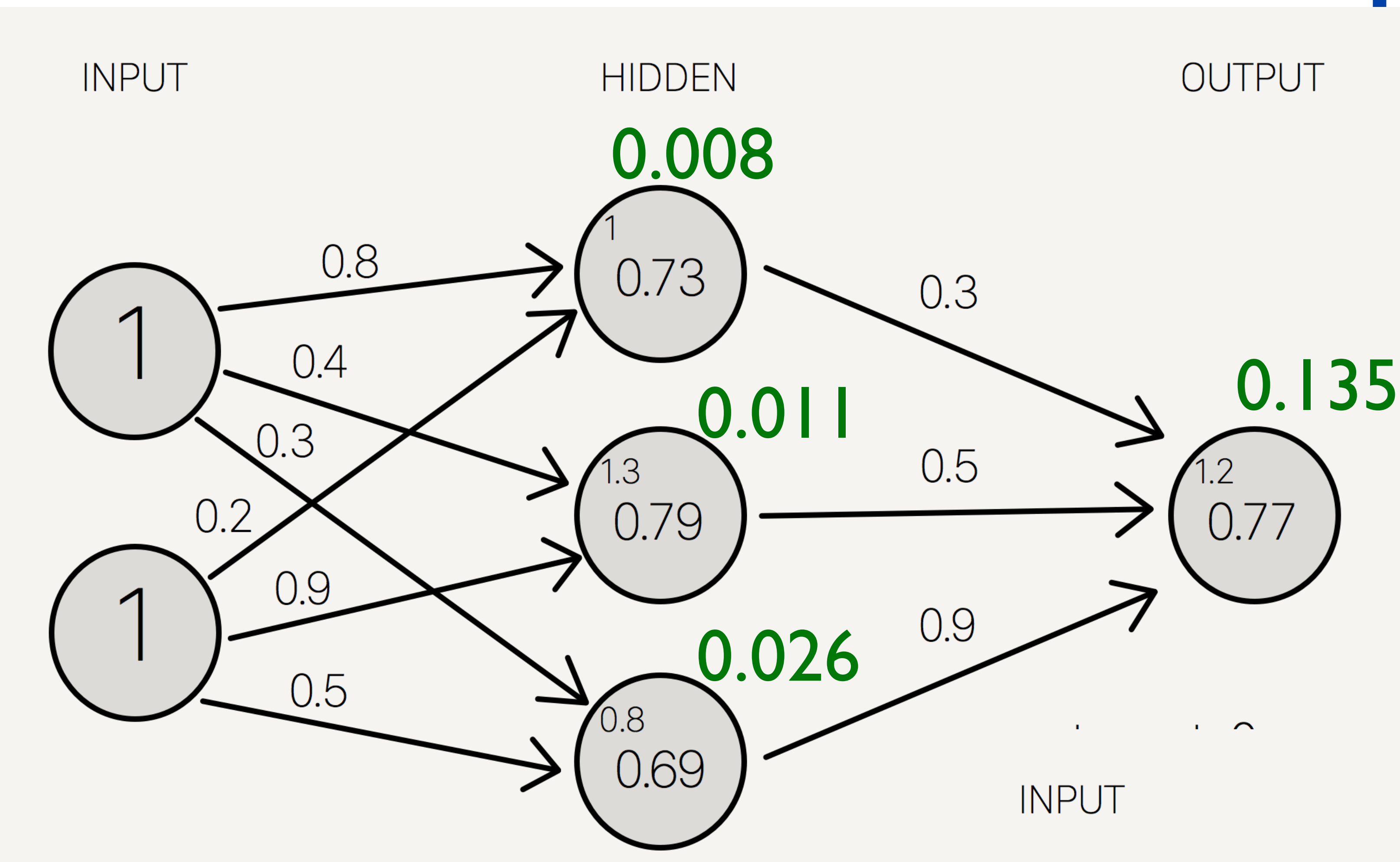
$$\frac{\partial J}{\partial z_i^{(n)}} = +(a_i^{(n)} - y_i).f'(z_i^{(n)})$$

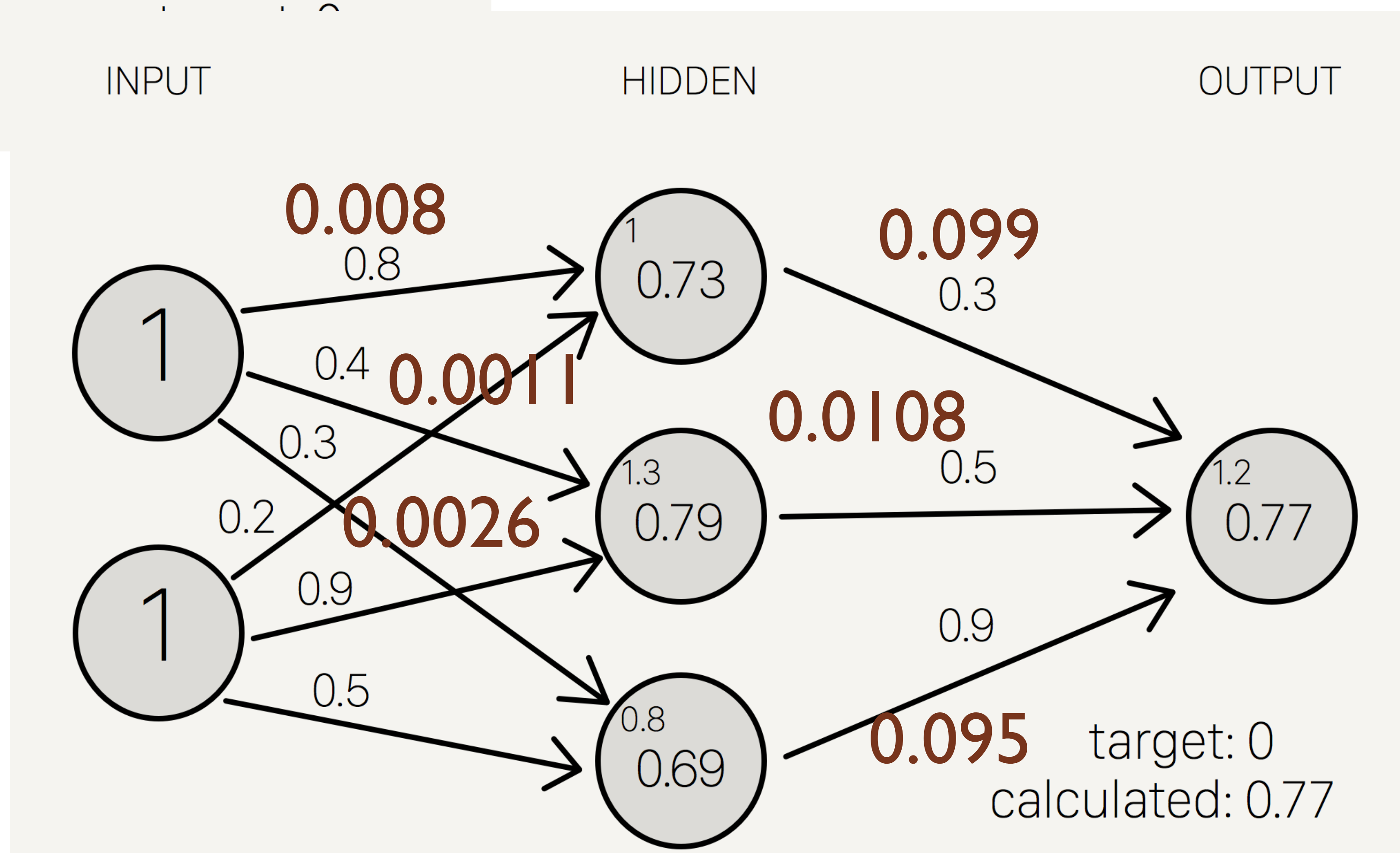
$$\frac{\partial J}{\partial z_i^{(l)}} = \frac{\partial J}{\partial z_k^{(l+1)}} W_{ki}^{(l)} f'(z_i^{(l)})$$



Example and fig. from S. Miller's github

An example





$$\frac{\partial J}{\partial W_{ij}^{(l)}} = \frac{\partial J}{\partial z_i^{(l+1)}} a_j^{(l)}$$

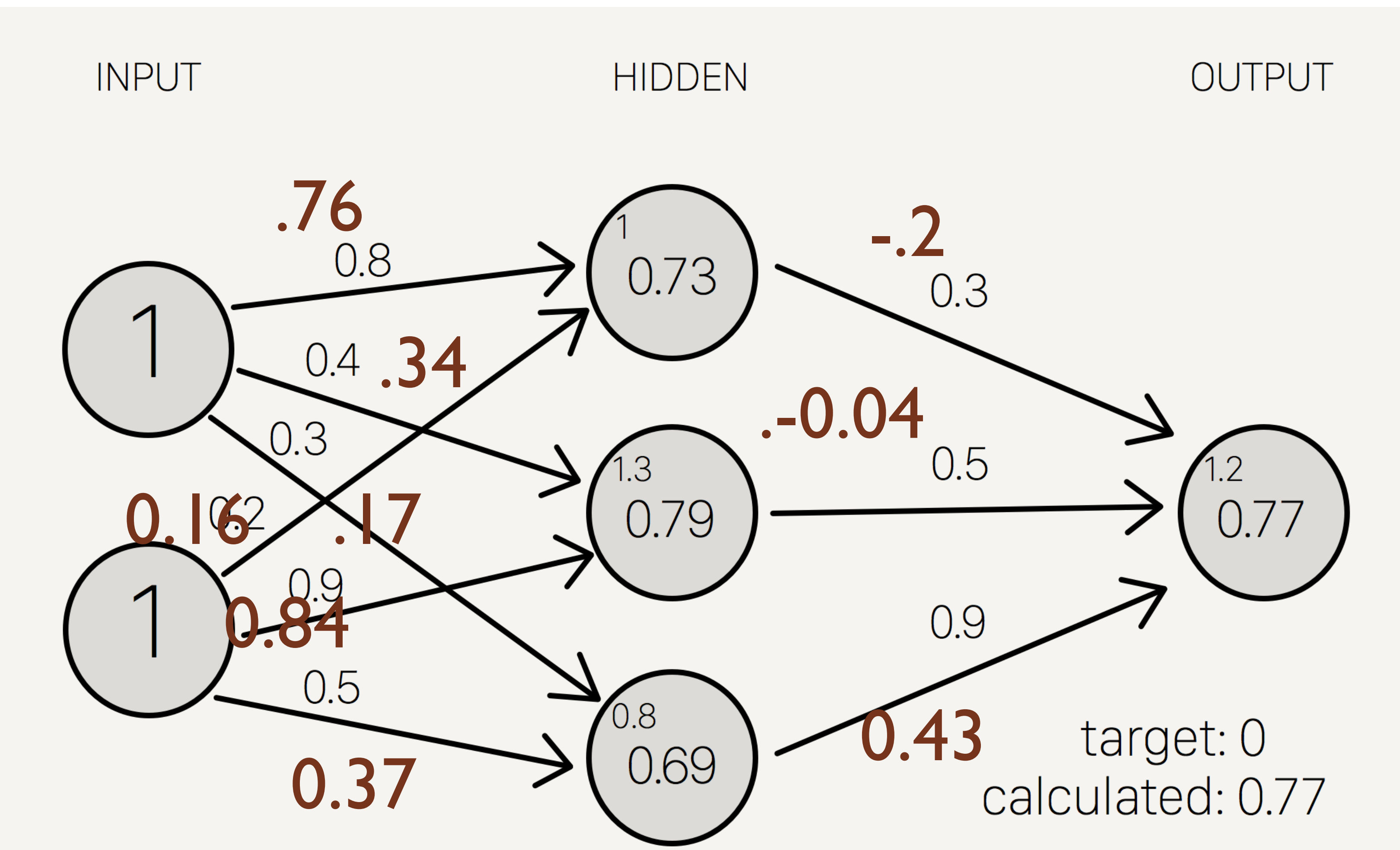
We have the gradient !

We have the gradient !

An example

- We can update te weights (alpha = 5)

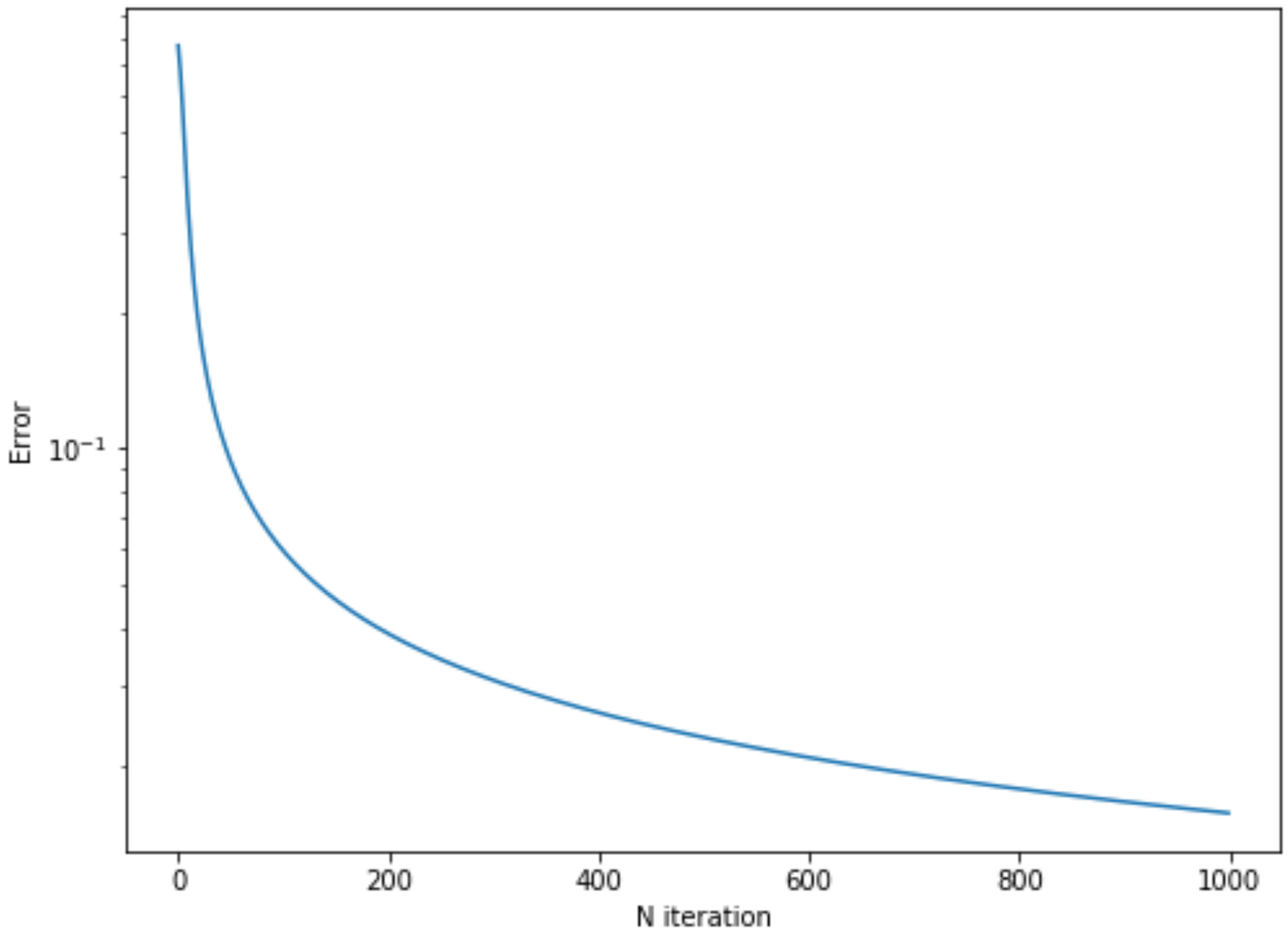
$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial J(\{W\},\{b\})}{\partial W_{ij}^{(l)}}$$



New calculated target: 0.53

An example

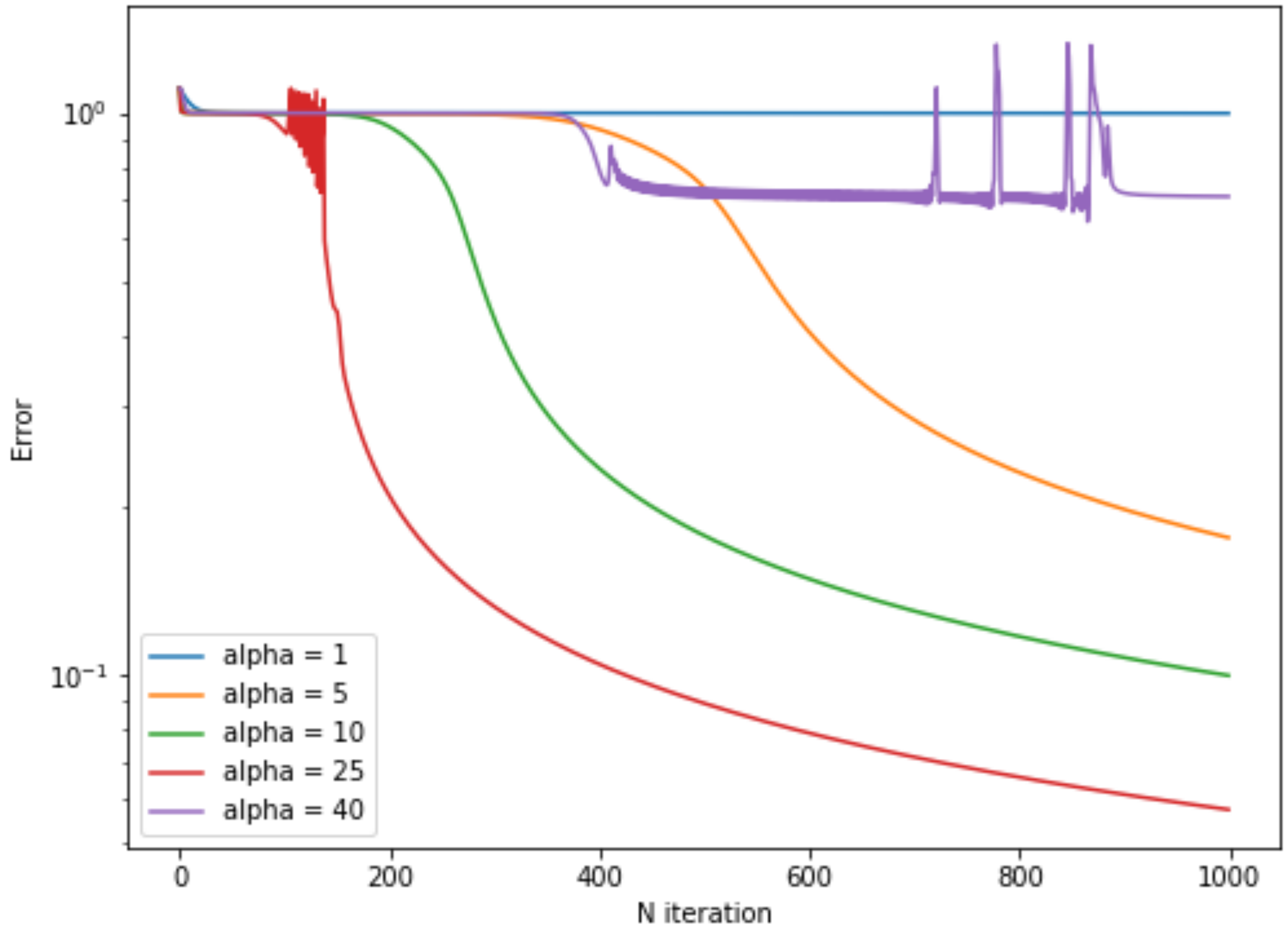
- The same example but we iterate (alpha fixed to 1)



The XOR with a more complete training set

- The sample set:

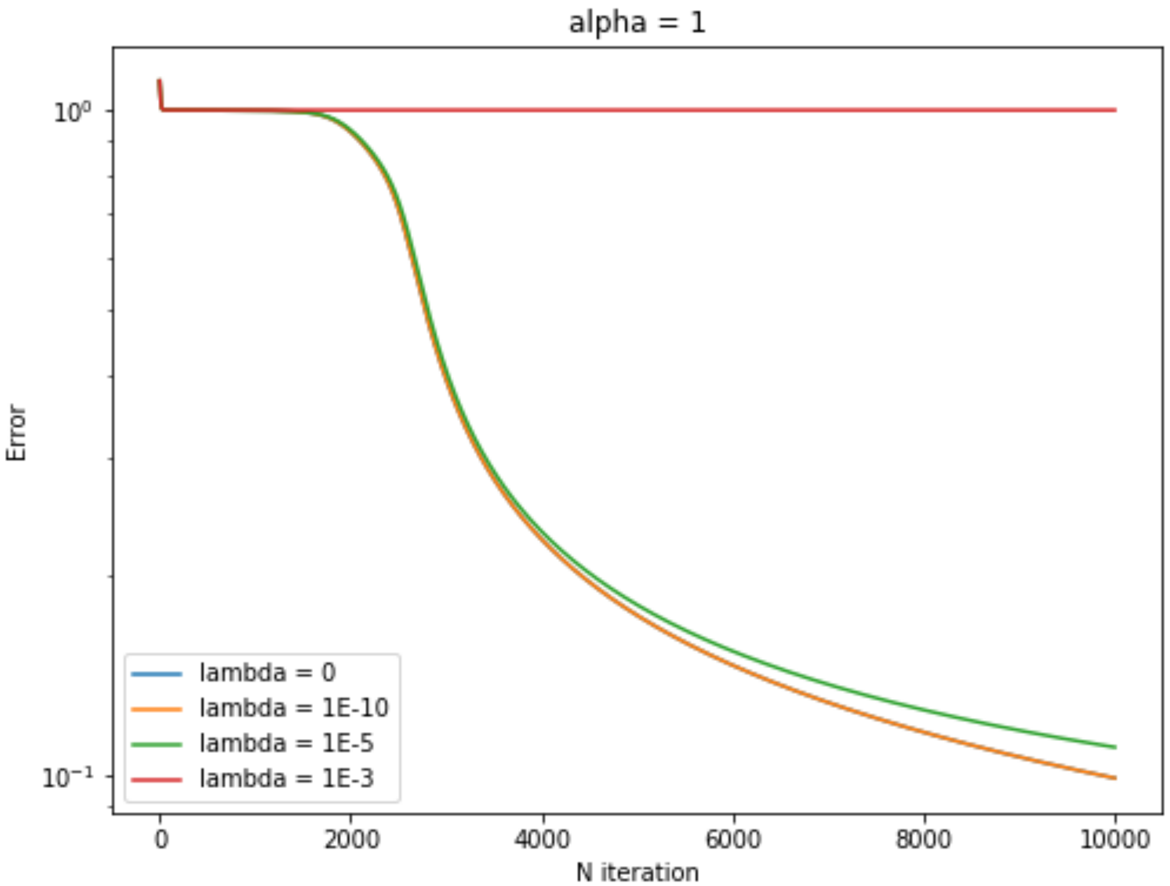
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

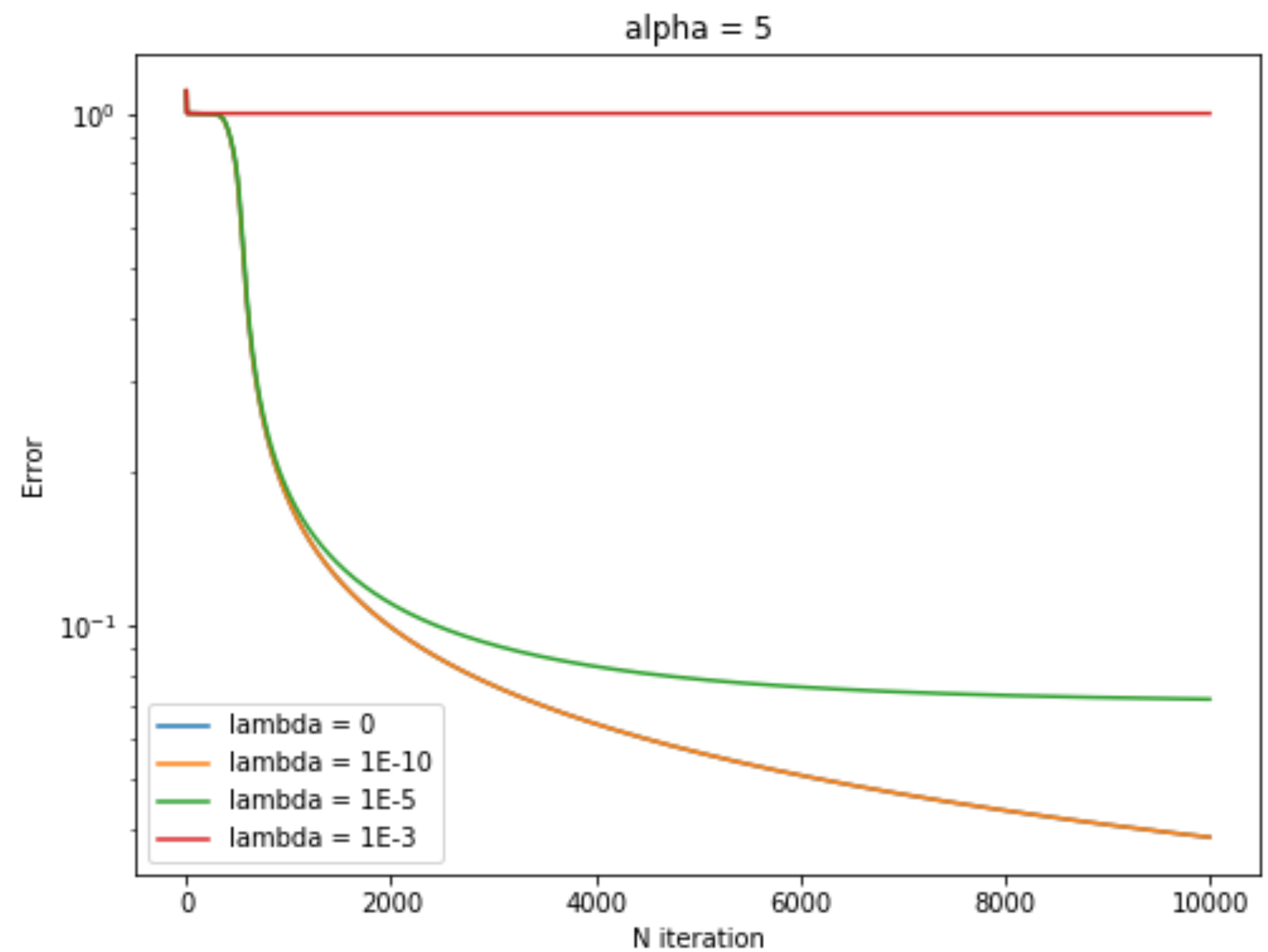


The XOR with a more complete training set

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1

- With the regularization

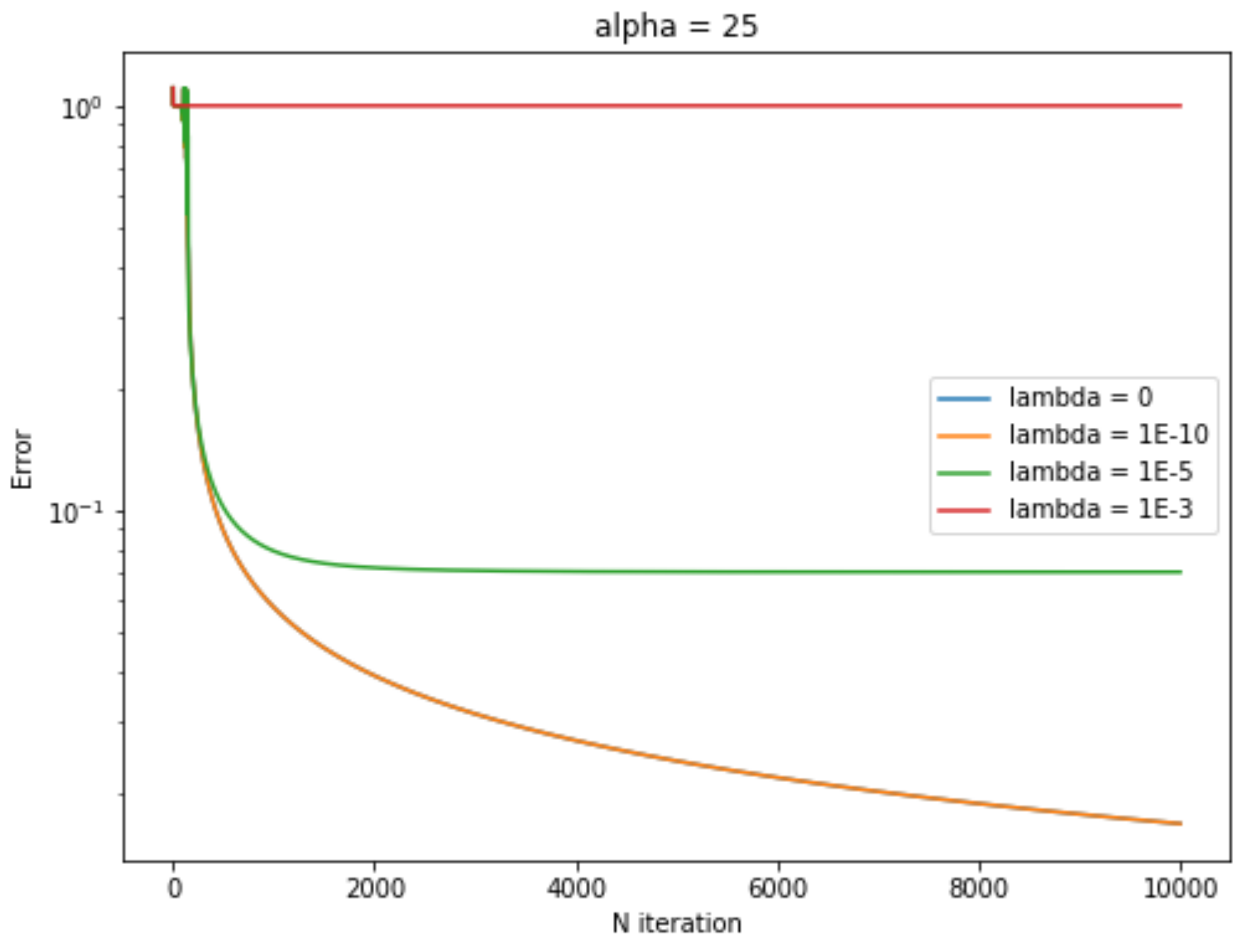


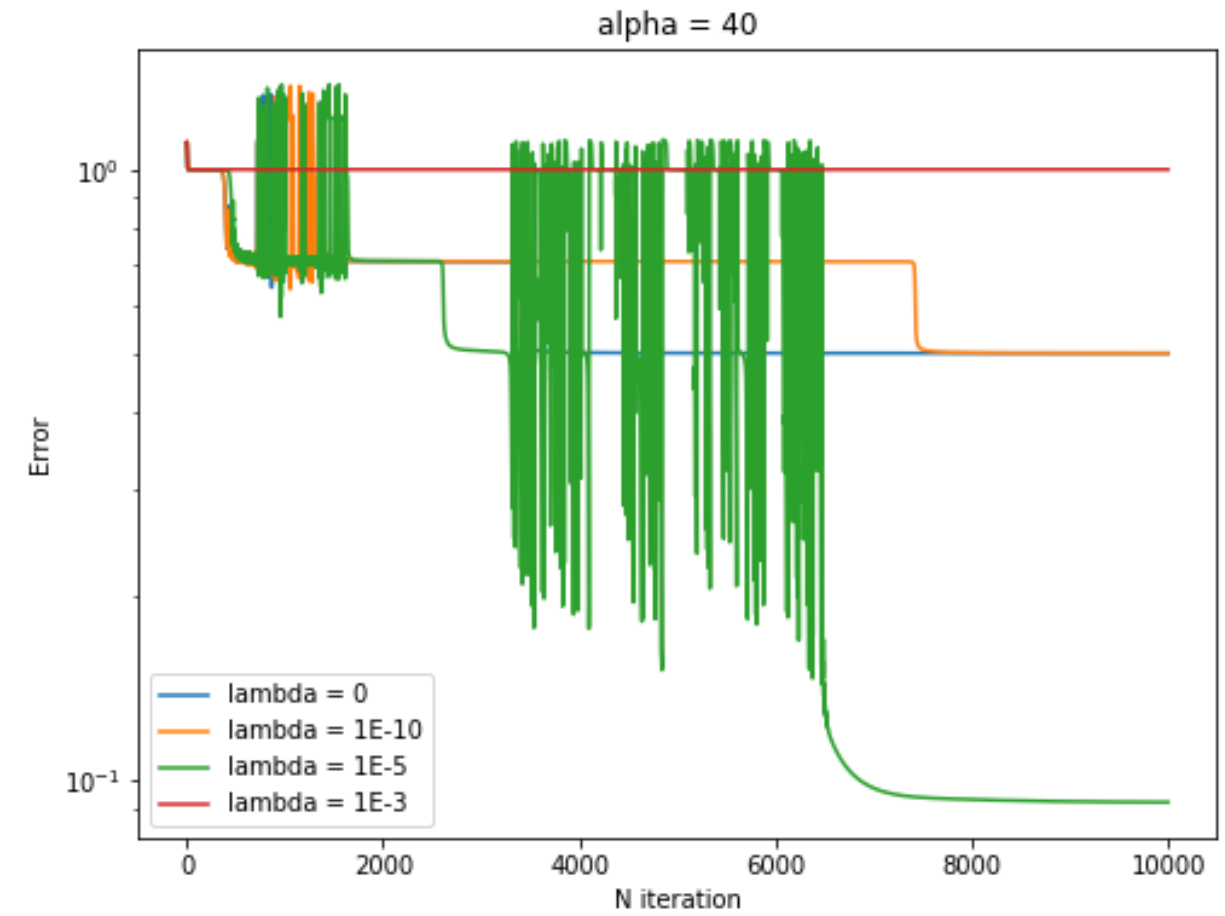


The XOR with a more complete training set

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

- With the regularization





An example of interpolation

$$y = x_1^2 + x_2^2$$

- First generate a training set + rescaled it between 0 and 1
- Choose the structure of the model (number of hidden layers + number of neurons)
- Train the model
- Use a set of "new" data and use the model to predict the output