A Brief Introduction to Deep Learning

Baojian Zhou, University at Albany, SUNY

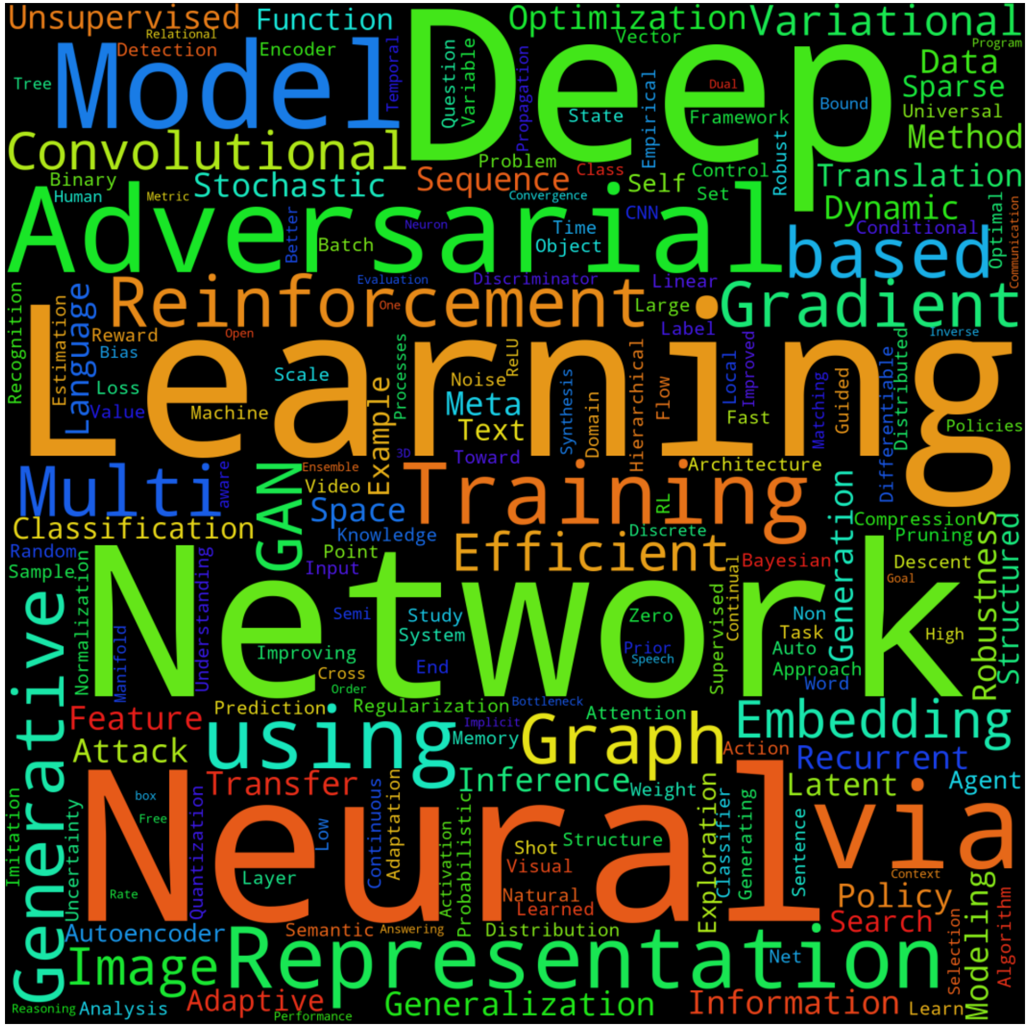
Outline

- **Successful stories of DL (2 minutes)**
- **DL Basics (28 minutes)**
- **Build two simple neural networks (45 minutes)**
- **TensorFlow and Others (5 minutes)**

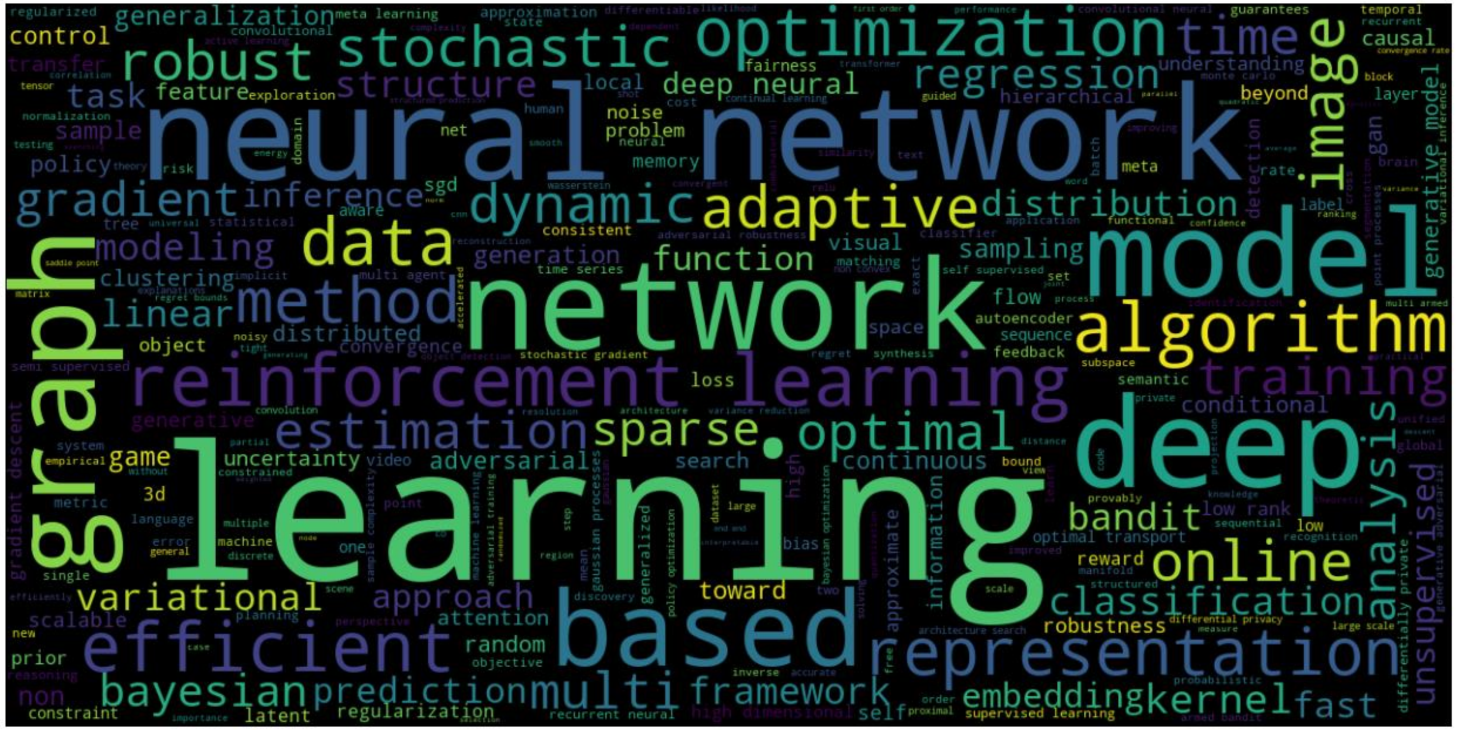
Outline

- **Successful stories of DL (2 minutes)**
- **DL Basics (28 minutes)**
- **Build two simple neural networks (45 minutes)**
- **TensorFlow and Others (5 minutes)**

AI, machine learning, deep learning, …



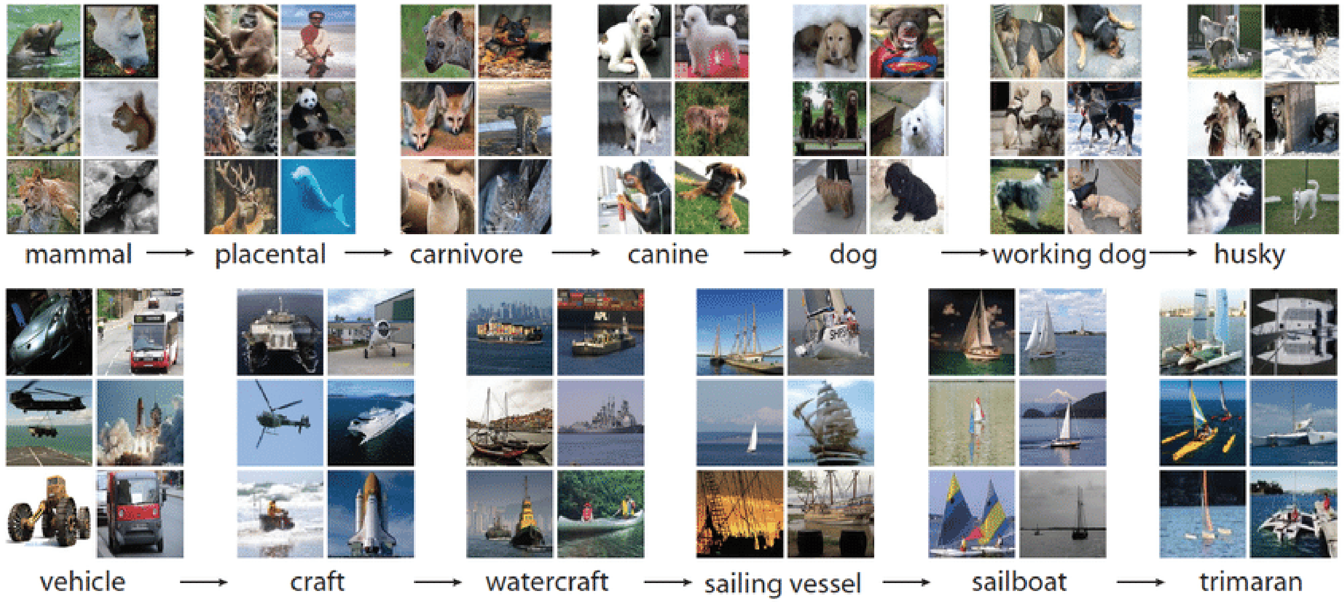
Word Cloud - ICLR, 2019

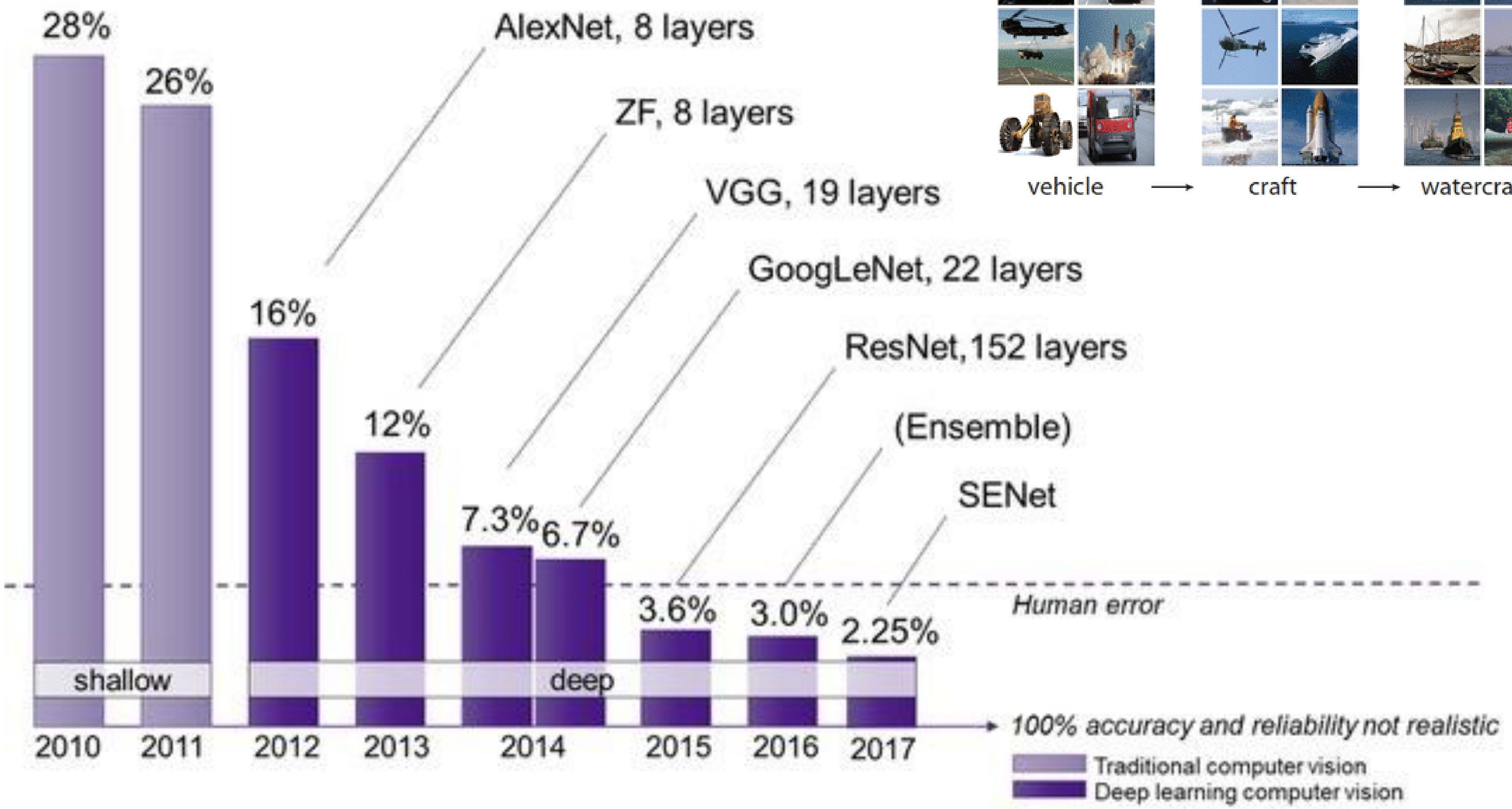


Word Cloud - NeurIPS, 2019

Successful stories of DL

ImageNet Classification





Successful stories of DL

Google DeepMind

AlphaGo

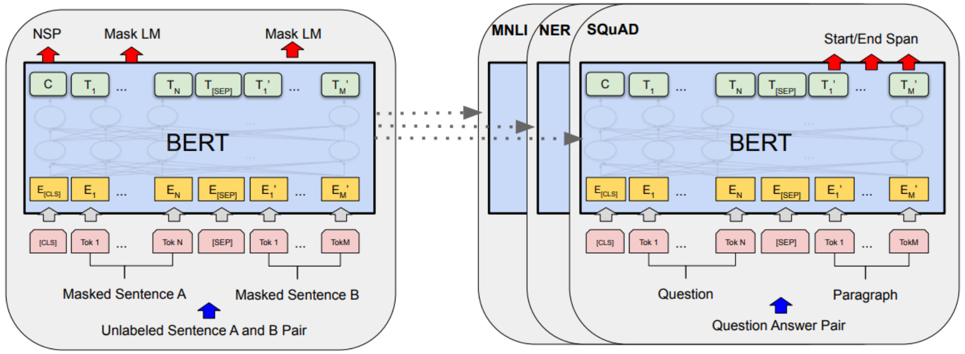
Successful stories of DL

Self-driving

WAYMO

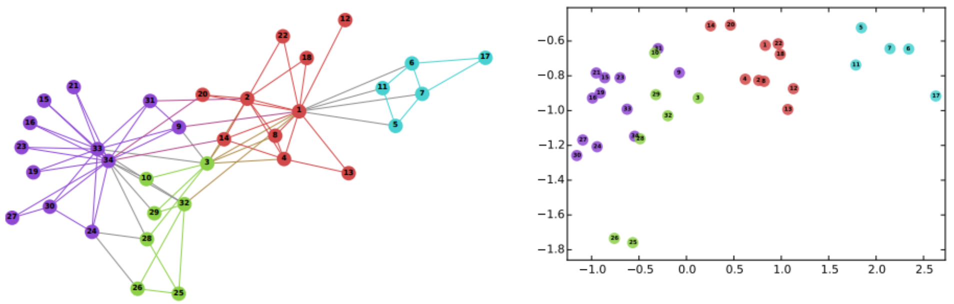
Successful stories of DL

Generative adversarial network, graph embedding, deep reinforcement learning, and many others …



Pre-training

Fine-Tuning



(a) Input: Karate Graph

(b) Output: Representation

Word2vec and BERT in NLP

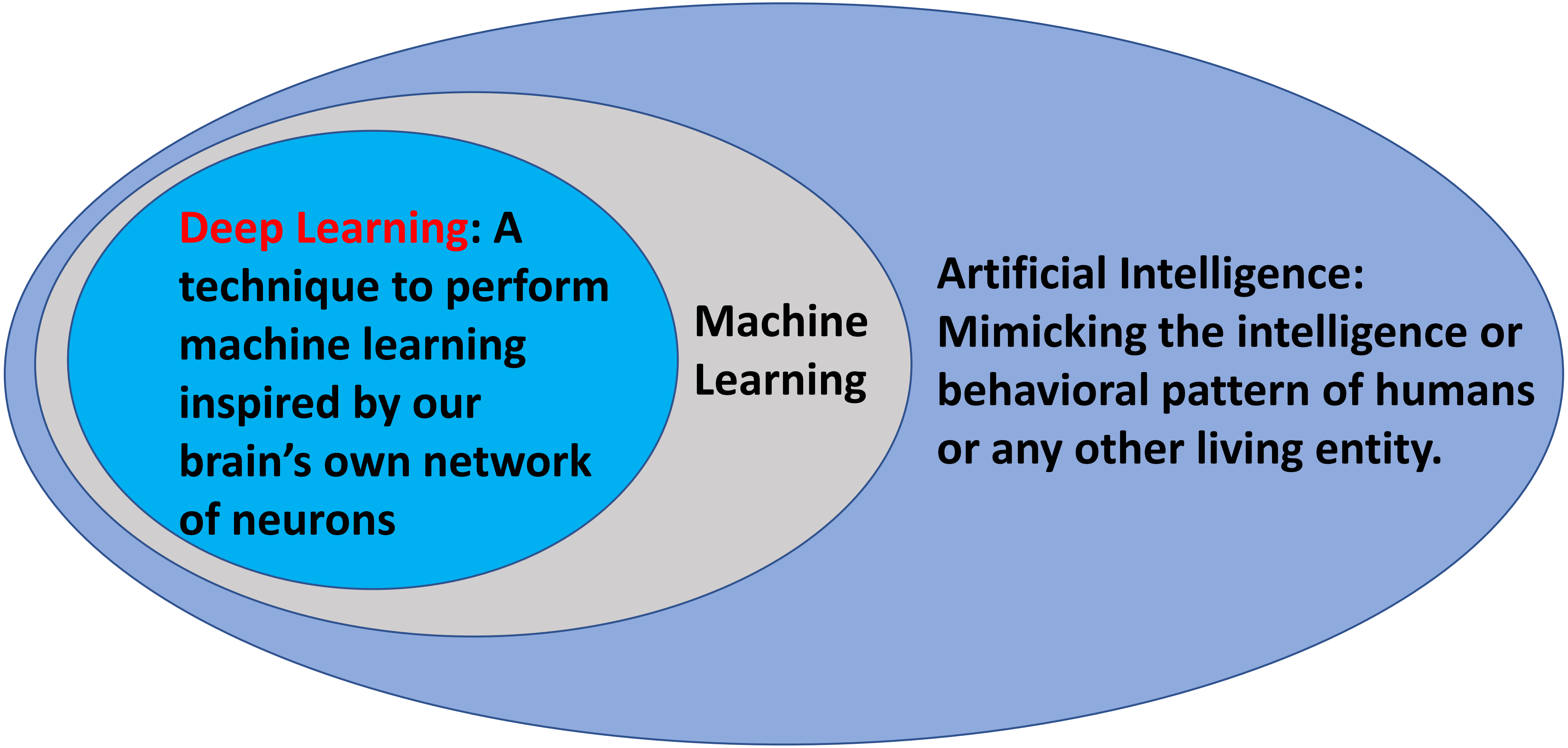
DeepWalk

Outline

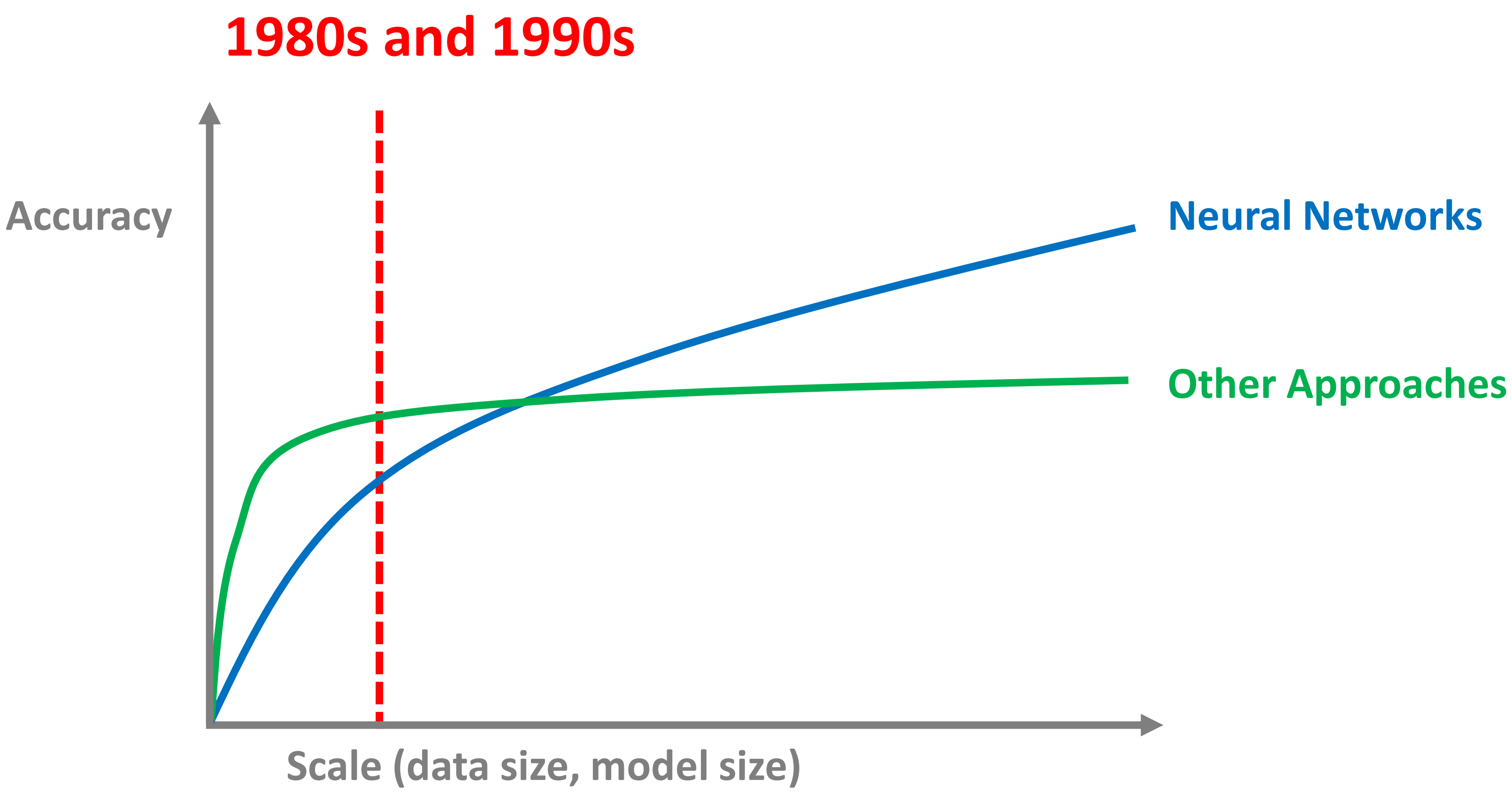
- Successful stories of DL (2 minutes)
- **DL Basics (28 minutes)**
- Build two simple neural networks (45 minutes)
- TensorFlow and Others (5 minutes)

Deep Learning – Definition

Deep learning is part of a broader family of machine learning methods based on **artificial neural networks**. Learning can be supervised, semi-supervised or unsupervised.

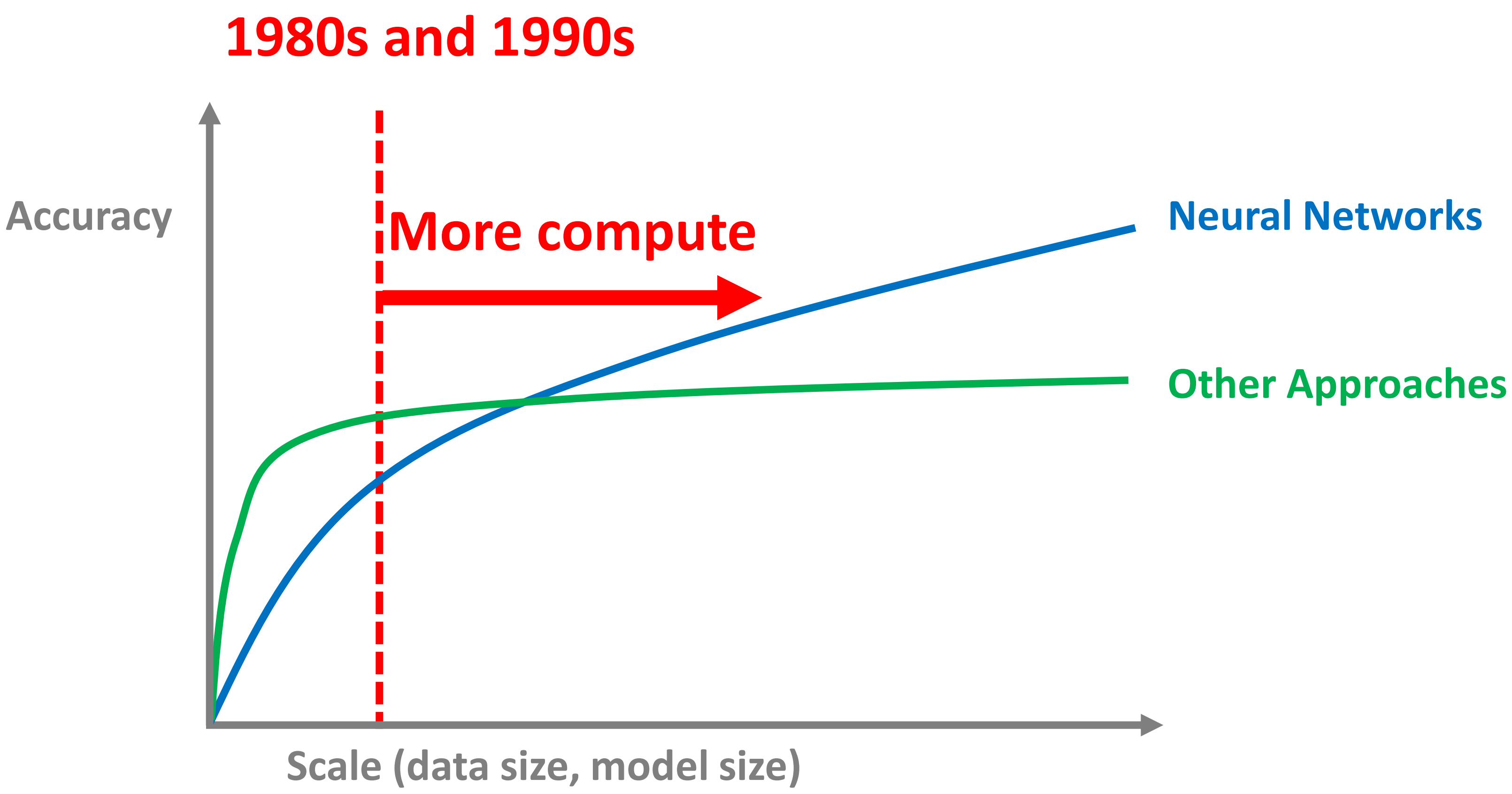


Why deep learning ?



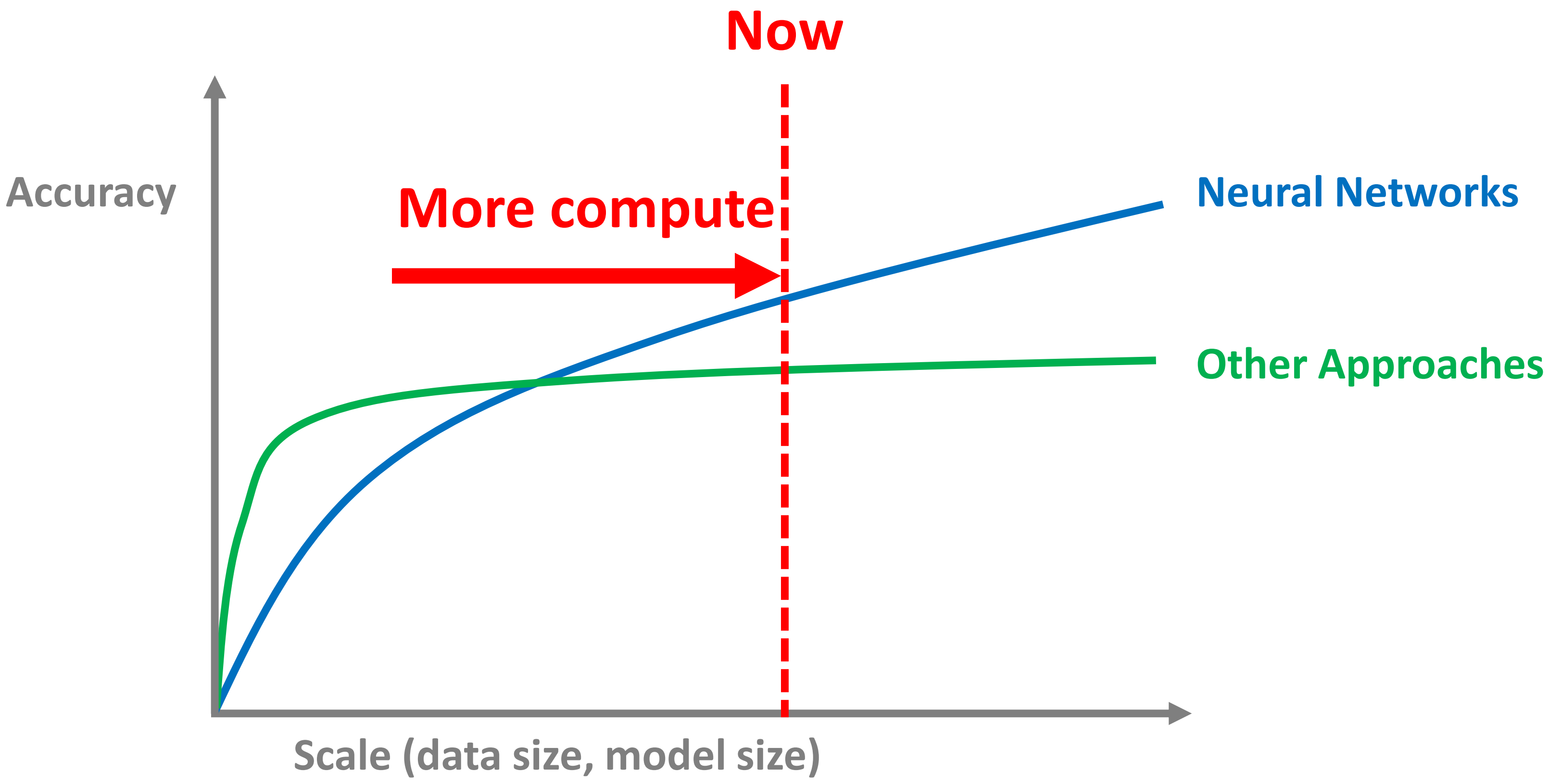
Jeff Dean's Lecture for YC AI

Why deep learning ?



Jeff Dean's Lecture for YC AI

Why deep learning ?



Jeff Dean's Lecture for YC AI

Artificial Neural Networks – Basics

What is a neural network

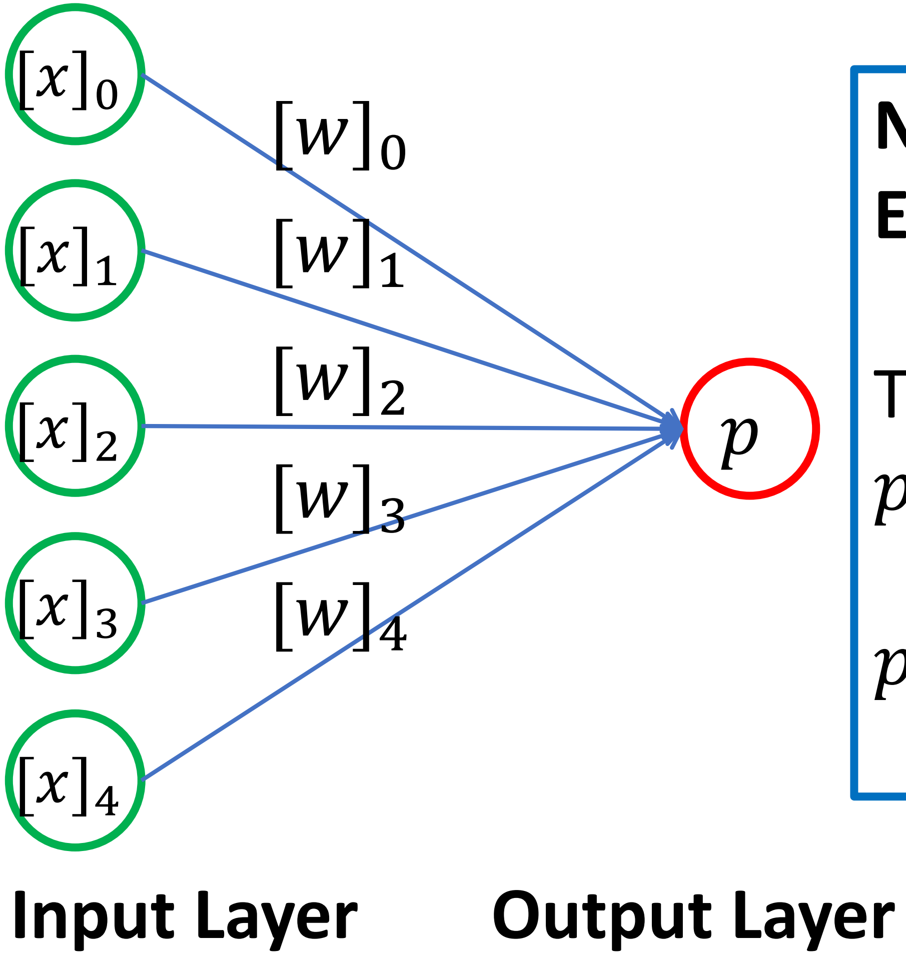
- A brain ?
- A computational graph ?
- A function ?

Input x → (deep) neural network with parameters w → Output $p(x, w)$

We can train the neural network, i.e., make it learn, so that, given any input x, the output $p(x; w)$ is what we want to see.

Artificial Neural Networks – Basics

A simple neural network



Nodes (Neurons) to represent input/output values.
Edges to represent multiplication by a weight.

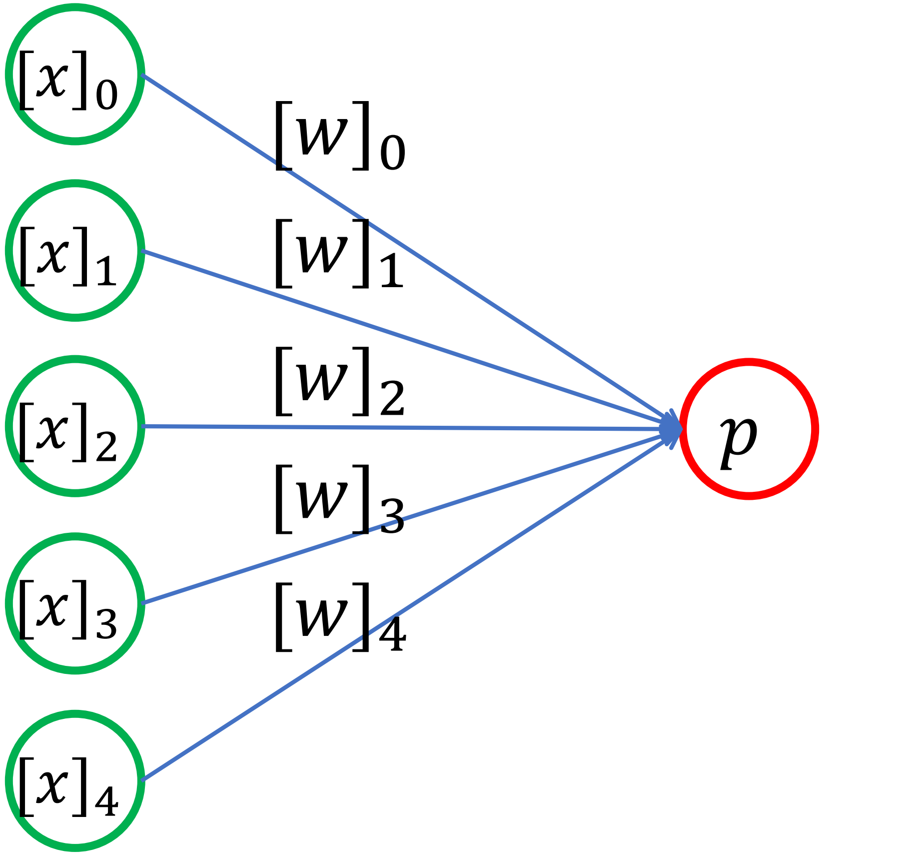
The output value is given by

$$p(w; x) = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + x_5 w_5$$

$$p(w; x) = \sum_{i=1}^{5} x_i w_i = \boldsymbol{x}^\top \boldsymbol{w}$$

Artificial Neural Networks – Basics

A simple neural network for **regression**



Input Layer **Output Layer**

Suppose we have a set of training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$ and suppose we want the weights so that

$$\boldsymbol{x}_i^\top \boldsymbol{w} \approx y_i \ \forall\, i \in \{1, \ldots, n\}.$$

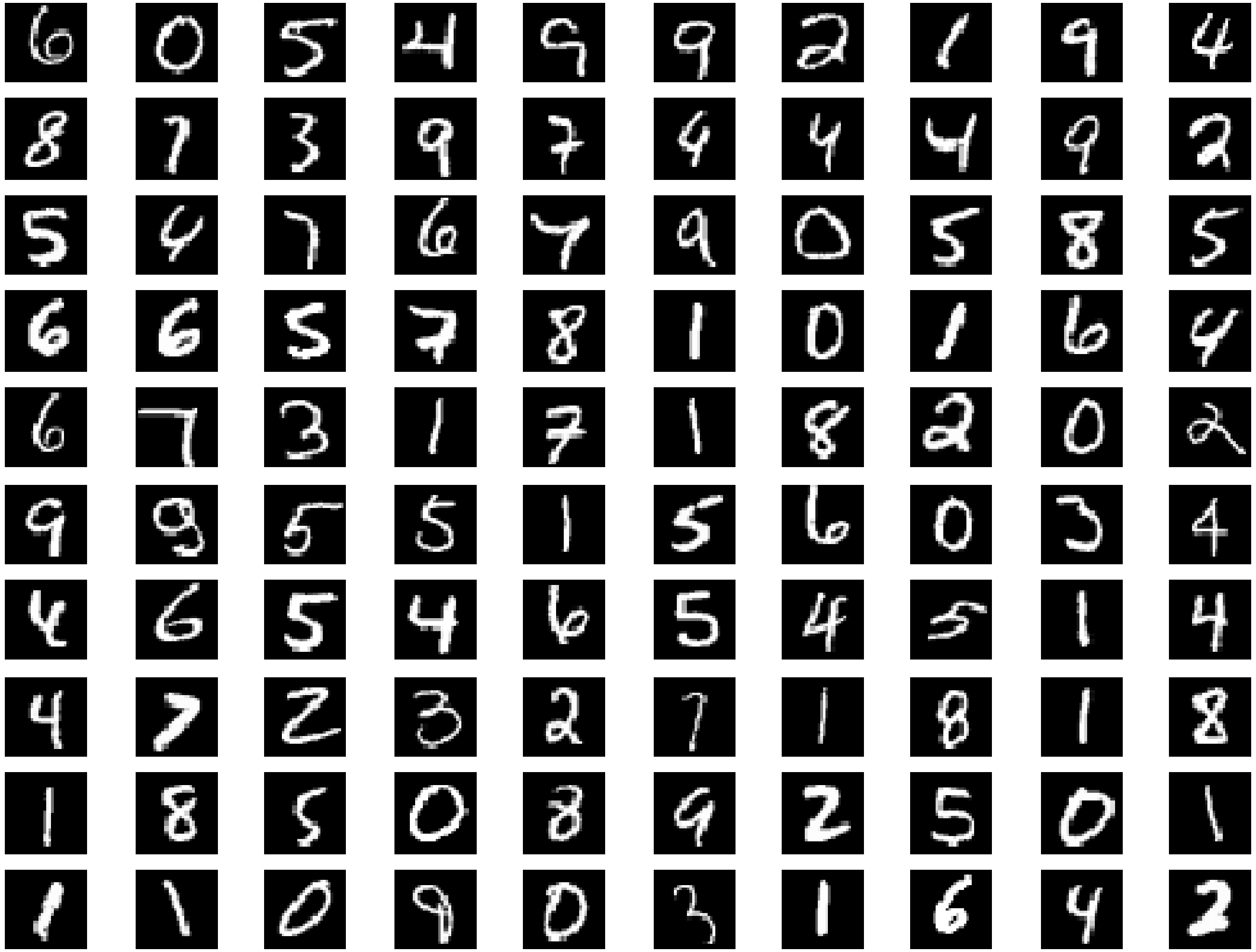
Then, we our goal is

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\boldsymbol{x}_i^\top \boldsymbol{w}, y_i),$$

for some loss ℓ. This is one way to describe linear regression using a network.

Artificial Neural Networks – Basics

A simple neural network for **classification**

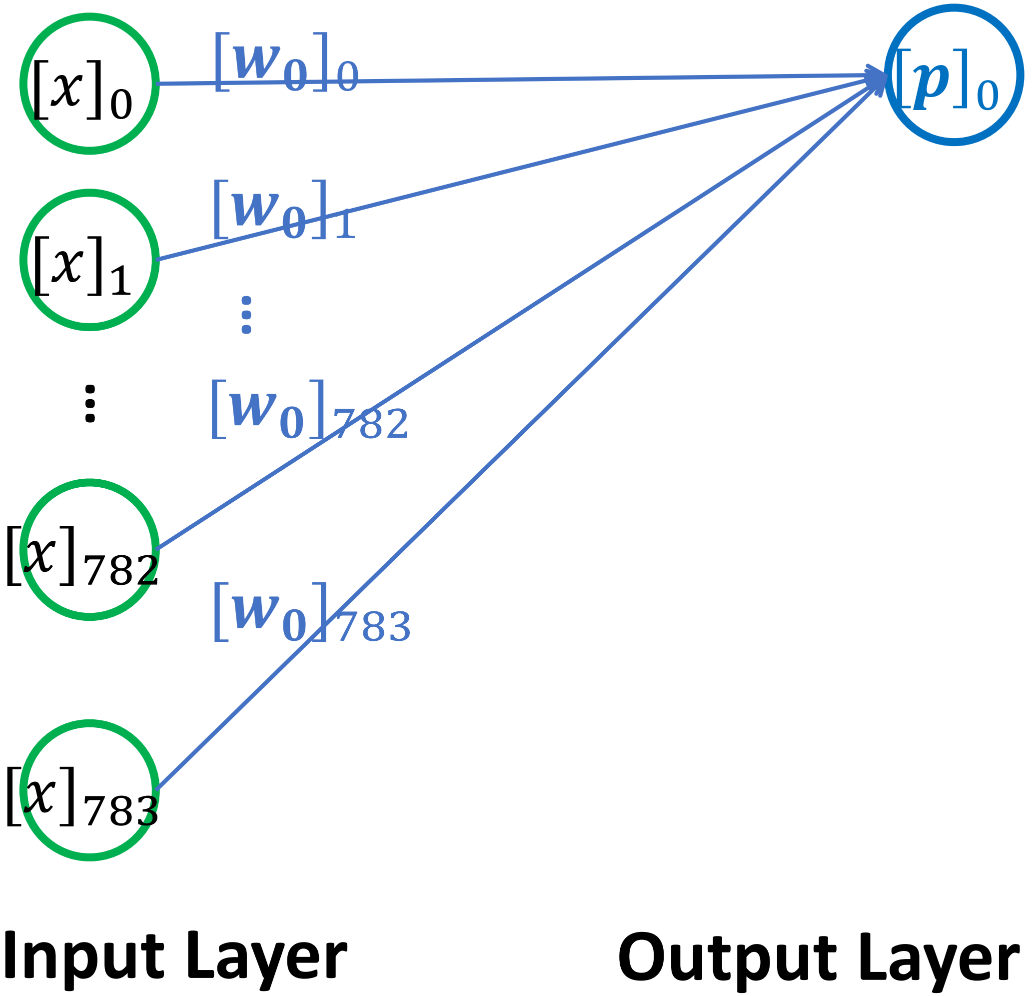


Suppose our inputs are images of digits we want to classify (MNIST dataset). The potential outputs are $0, 1, 2, \ldots, 9$.

Each digit image is $d = 28 \times 28$

Artificial Neural Networks – Basics

A simple neural network for classification



Now, instead of a single vector w, we want 10 weight vectors (one for each digit).

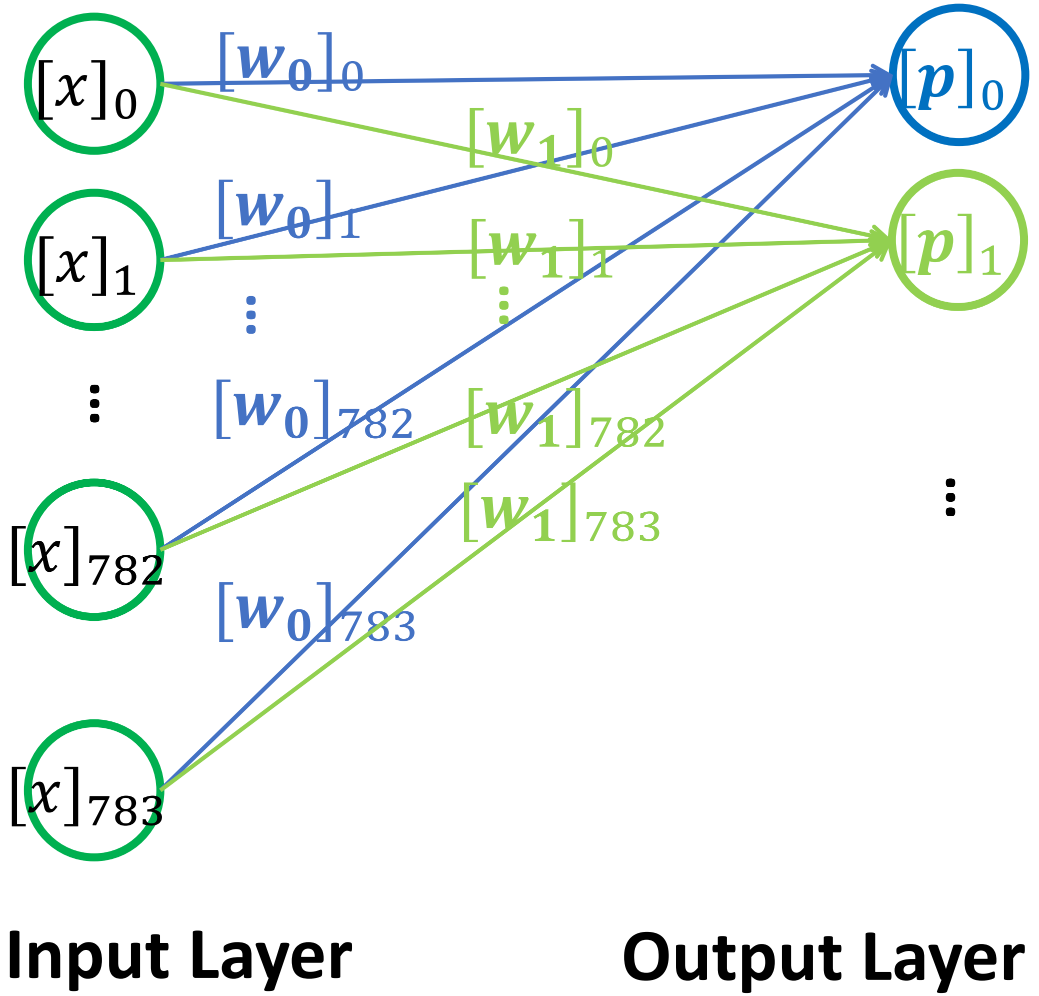
Define the vector $\boldsymbol{w_j}$ for $j \in \{0, 1, 2, \dots, 9\}$.

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_0^\top} \\ \boldsymbol{w_1^\top} \\ \boldsymbol{w_2^\top} \\ \dots \\ \boldsymbol{w_9^\top} \end{bmatrix}. \text{ Then, } \boldsymbol{p} = \boldsymbol{W}\boldsymbol{x}.$$

We want to choose $\boldsymbol{W}$ such that, when the correct digit is j $\in \{0,1,2, \dots, 9\}$, $[\boldsymbol{p}]_j > [\boldsymbol{p}]_i$ for $i \neq j$.

Artificial Neural Networks – Basics

A simple neural network for **classification**



Now, instead of a single vector w, we want 10 weight vectors (one for each digit).

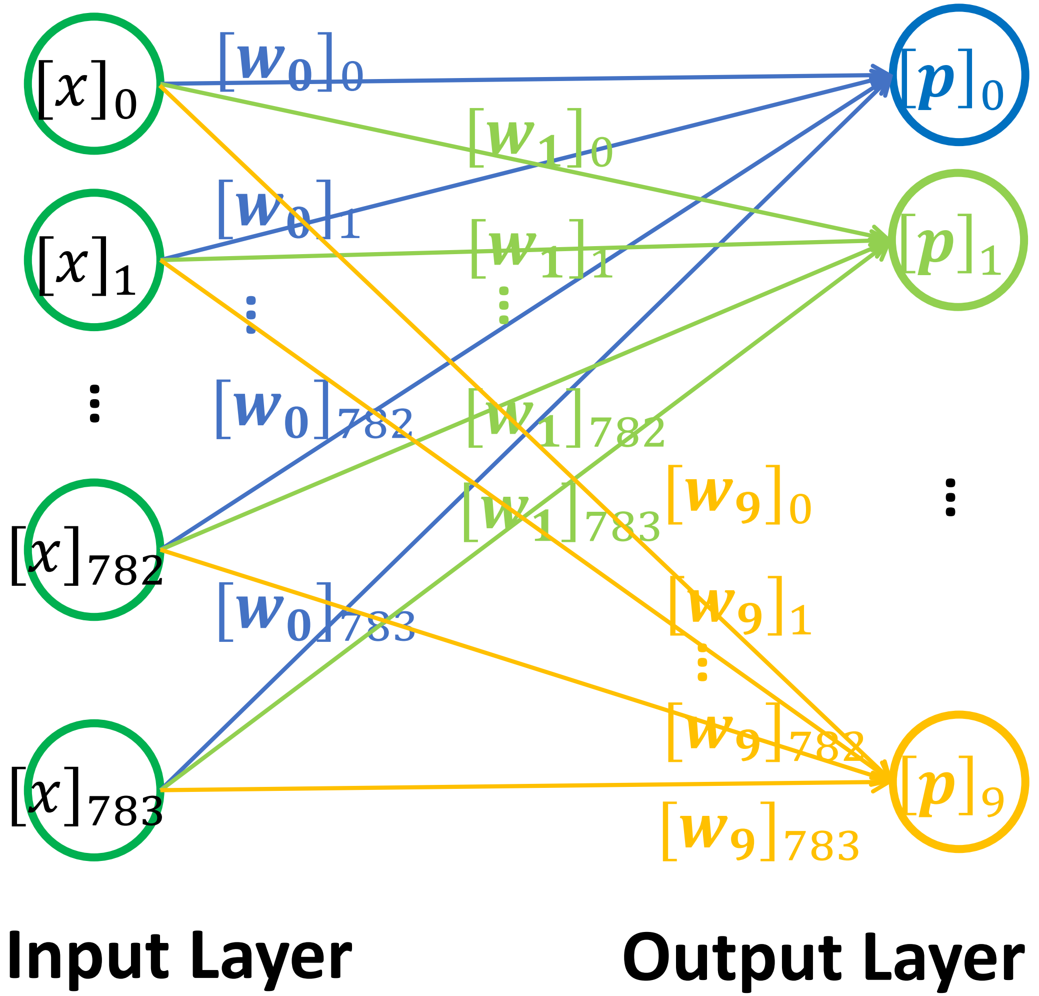
Define the vector $\boldsymbol{w_j}$ for $j \in \{0, 1, 2, \dots, 9\}$.

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_0}^\top \\ \boldsymbol{w_1}^\top \\ \boldsymbol{w_2}^\top \\ \dots \\ \boldsymbol{w_9}^\top \end{bmatrix}. \text{ Then, } \boldsymbol{p} = \boldsymbol{W}\boldsymbol{x}.$$

We want to choose $\boldsymbol{W}$ such that, when the correct digit is j $\in \{0,1,2, \dots, 9\}$, $[\boldsymbol{p}]_j >$ $[\boldsymbol{p}]_i$ for $i \neq j$.

Artificial Neural Networks – Basics

A simple neural network for classification



Now, instead of a single vector w, we want 10 weight vectors (one for each digit).

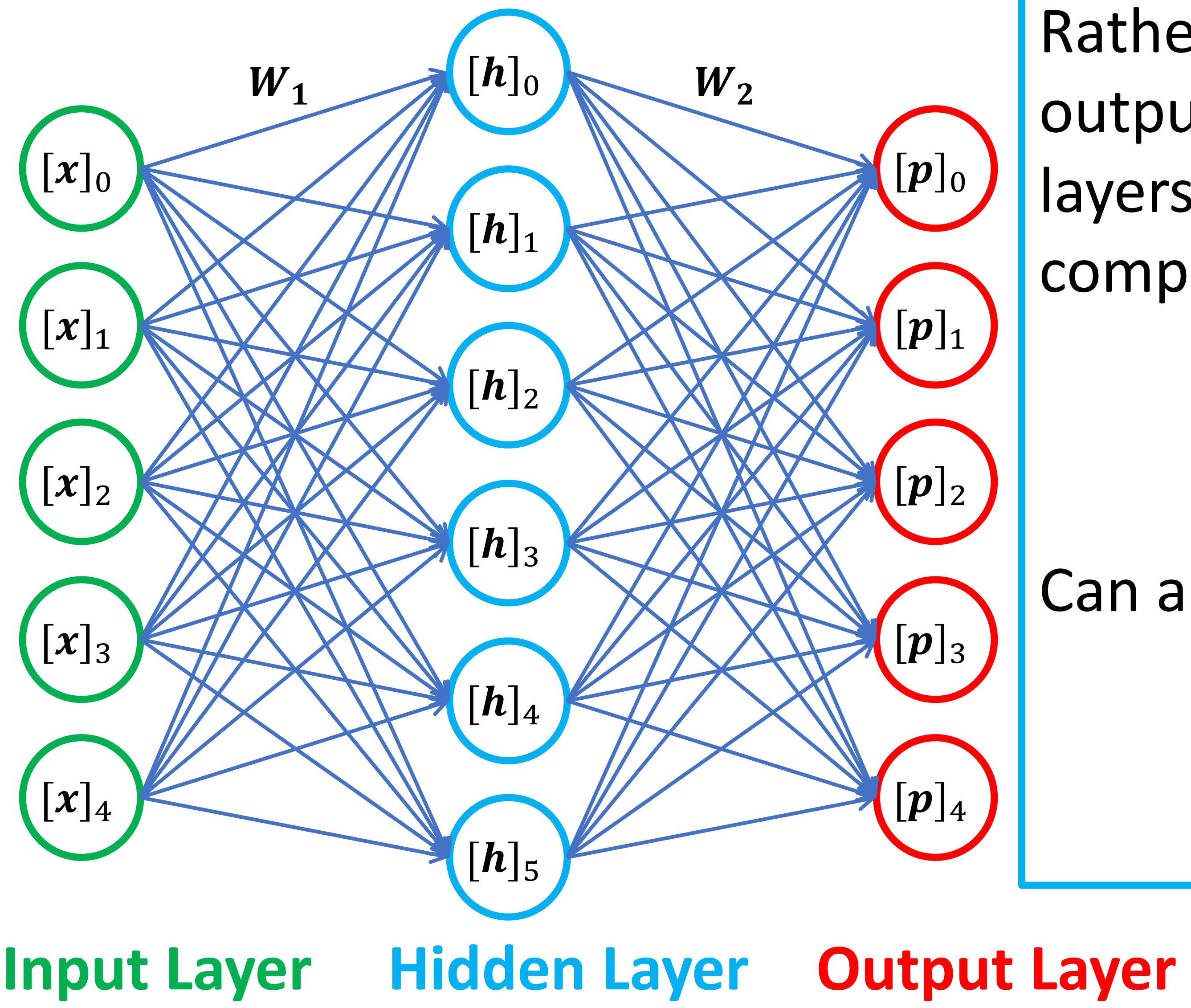
Define the vector $\boldsymbol{w_j}$ for $j \in \{0, 1, 2, \dots, 9\}$.

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_0^\top} \\ \boldsymbol{w_1^\top} \\ \boldsymbol{w_2^\top} \\ \dots \\ \boldsymbol{w_9^\top} \end{bmatrix}. \text{ Then, } \boldsymbol{p} = \boldsymbol{Wx}.$$

We want to choose $\boldsymbol{W}$ such that, when the correct digit is j $\in \{0,1,2, \dots, 9\}$, $[\boldsymbol{p}]_j > [\boldsymbol{p}]_i$ for $i \neq j$.

Artificial Neural Networks – Basics

Hidden Layers



Rather than go directly from inputs to outputs. We can add one or more hidden layers. Now the network performs the computation. Then,

$$\boldsymbol{p} = \boldsymbol{W}_2 \boldsymbol{h}(\boldsymbol{W}_1 \boldsymbol{x}).$$

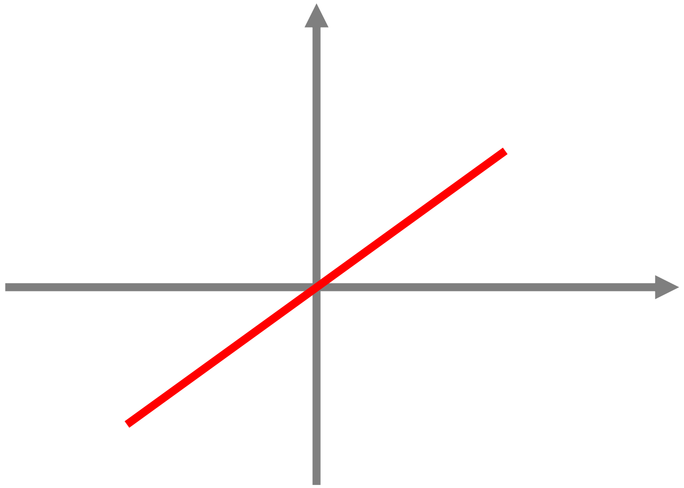
Can also include bias terms such that

$$\boldsymbol{p} = \boldsymbol{W}_2 \boldsymbol{h}(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_0) + \boldsymbol{b}_1.$$

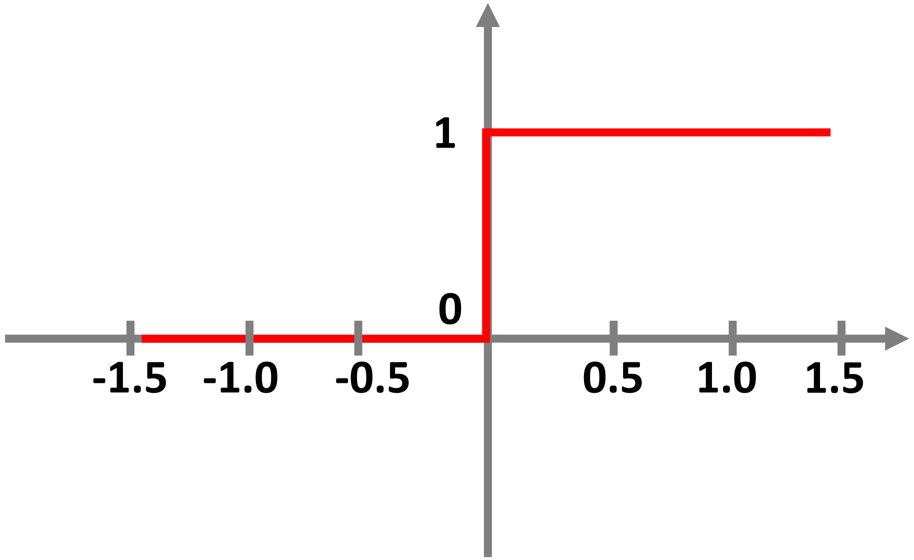
Artificial Neural Networks – Basics

Activation functions h

Inspired by neuroscience, use activation functions to approximate the "firing or not" of a neuron. We want to approximate a 0-1 step function.



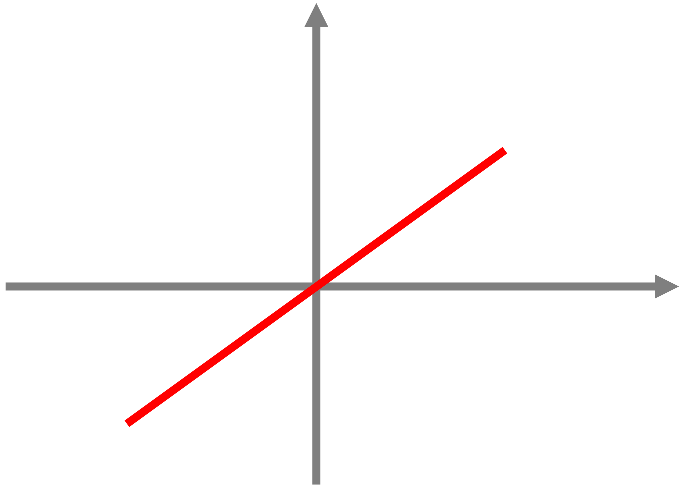
Identity (a.k.a Linear)



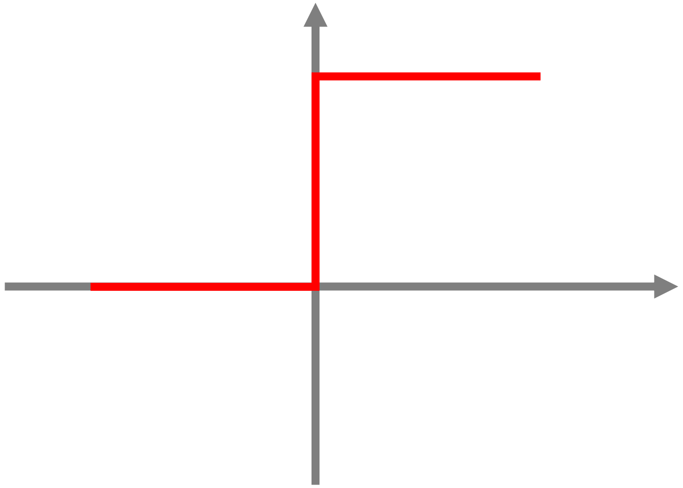
Binary Step

Artificial Neural Networks – Basics

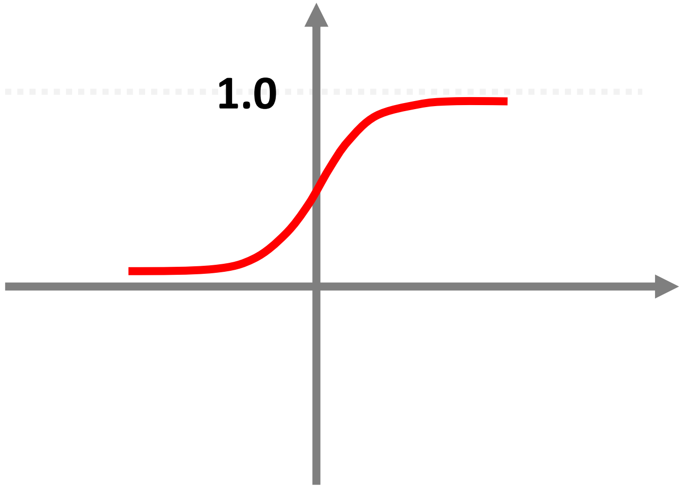
Activation functions h



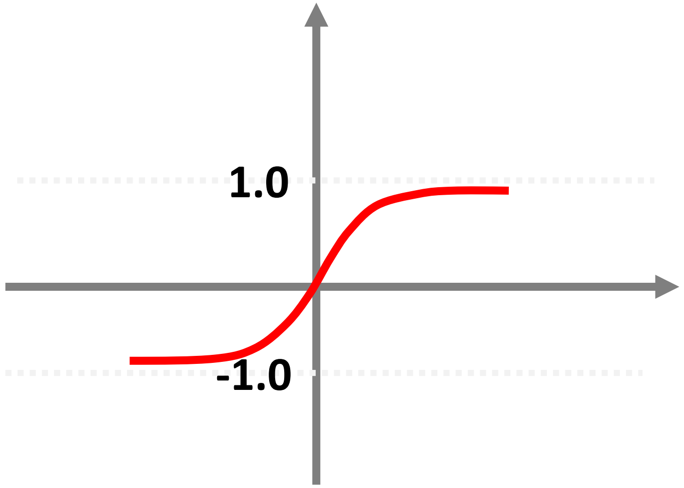
Identity (a.k.a Linear)



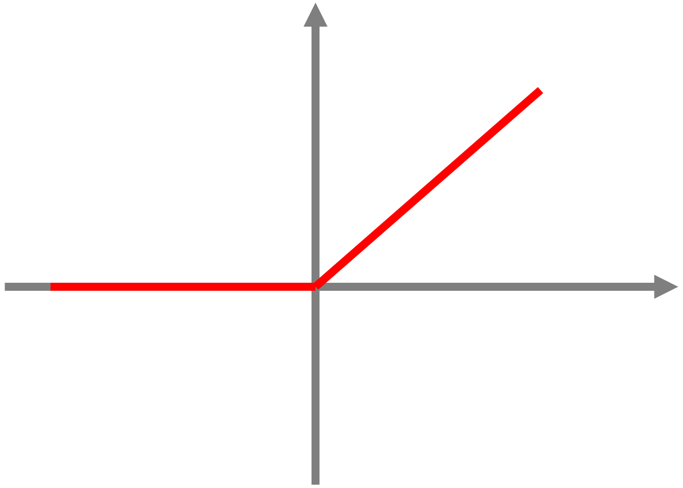
Binary Step



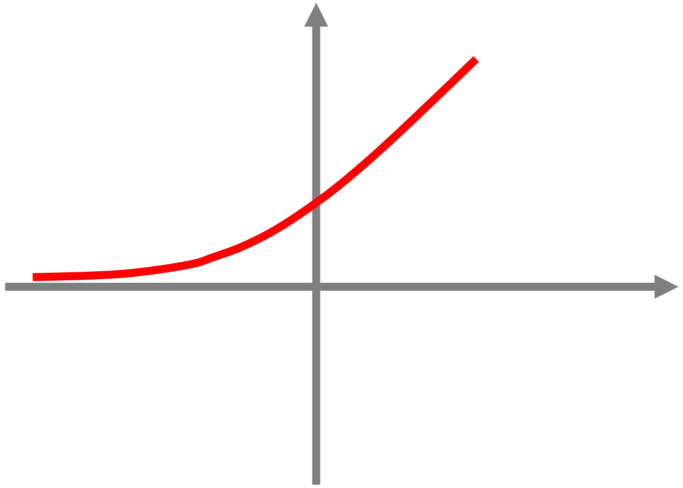
Logistic (a.k.a Sigmoid)



TanH



Rectified Linear Unit (ReLU)



SoftPlus

Artificial Neural Networks – Basics

To design a neural net, you need to

Even for a basic Neural Network, there are many design decisions to make:

- **Number of hidden layers (depth)**
- **Number of units per hidden layer (width)**
- **Type of activation function (nonlinearity)**
- **Form of objective function**
- **Recurrent or feed-forward?**

These parameters will affect the overall performance such as computation cost and accuracy.

Artificial Neural Networks – Basics

Computation cost

For example, I want to design a neural network for classifying image digits (MNIST). Let us consider the following neural network:

- **$28 \times 28 = 784$ neurons in the input layer**
- **512 neurons in the first hidden layer (fully connected)**
- **256 neurons in the second hidden layer (fully connected)**
- **10 neurons in the output layer (fully connected)**

Question: How many optimization variables are there?

Artificial Neural Networks – Basics

Computation cost

For example, I want to design a neural network for classifying image digits (MNIST). Let us consider the following neural network:

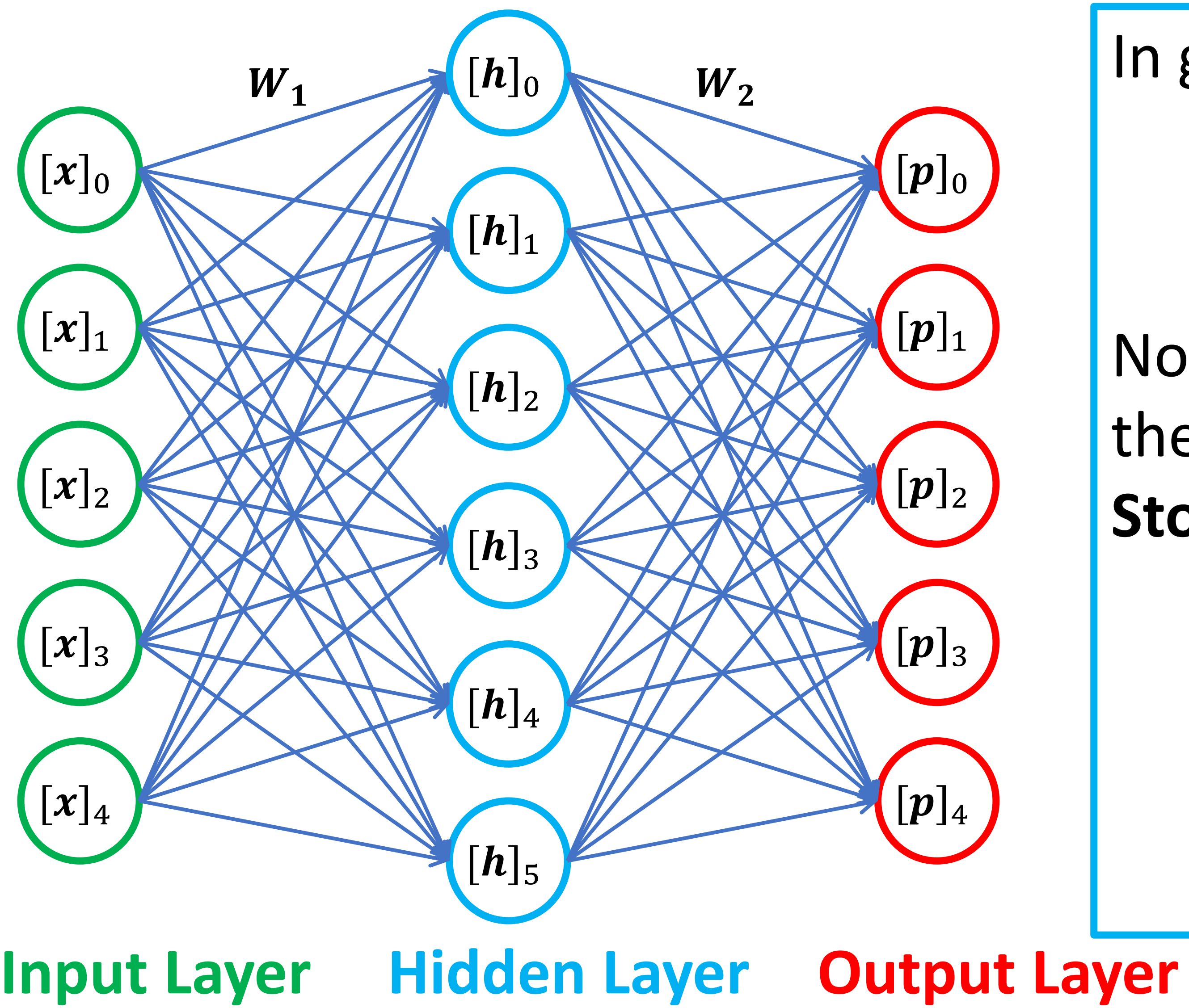
- **$28 \times 28 = 784$ neurons in the input layer**
- **512 neurons in the first hidden layer (fully connected)**
- **256 neurons in the second hidden layer (fully connected)**
- **10 neurons in the output layer (fully connected)**

Question: How many optimization variables are there?

In the first layer: 784×512
In the second layer: 512×256
In the third layer: 256×10
$784 \times 512 + 512 \times 256 + 256 \times 10 = \mathbf{535040}$.

Artificial Neural Networks – Basics

How to update the weights $\boldsymbol{W}_1, \boldsymbol{W}_2$?



In general, we minimize a loss

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2} \frac{1}{n} \sum_{i=1}^{n} \ell(\boldsymbol{W}_1, \boldsymbol{W}_2; \boldsymbol{x}_i, \boldsymbol{y}_i)$$

Now the question is how to optimize the loss w.r.t $\boldsymbol{W}_1, \boldsymbol{W}_2$.

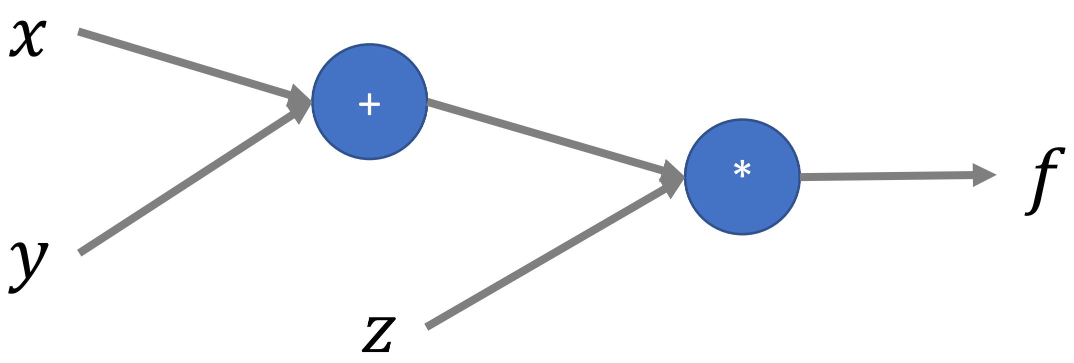
Stochastic Gradient Descent:

$$[\boldsymbol{W}_1]_{ij}^{t+1} = [\boldsymbol{W}_1]_{ij}^{t} - \eta_t \frac{d\ell}{d[\boldsymbol{W}_1]_{ij}^{t}}$$

$$[\boldsymbol{W}_2]_{ij}^{t+1} = [\boldsymbol{W}_2]_{ij}^{t} - \eta_t \frac{d\ell}{d[\boldsymbol{W}_2]_{ij}^{t}}$$

Artificial Neural Networks – Basics

Backpropagation: a simple example



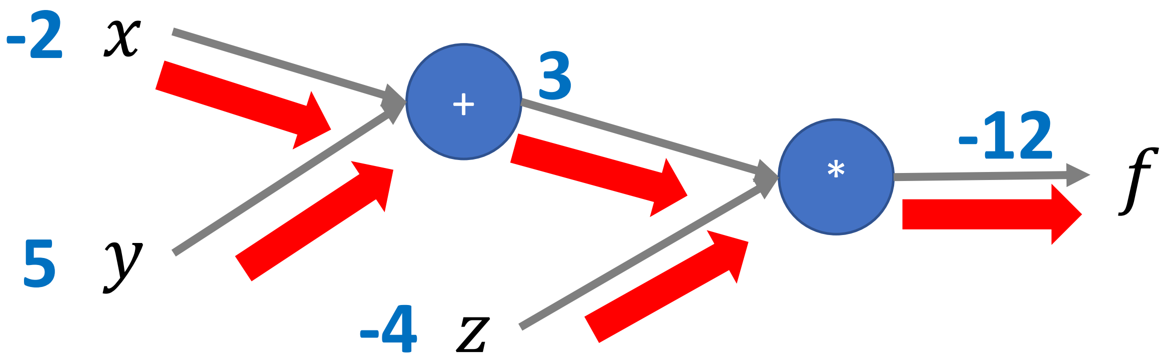
$$f(x, y, z) = (x + y) * z$$

We want to calculate:

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$$

Artificial Neural Networks – Basics

Backpropagation: a simple example



$$f(x, y, z) = (x + y) * z$$

We want to calculate:

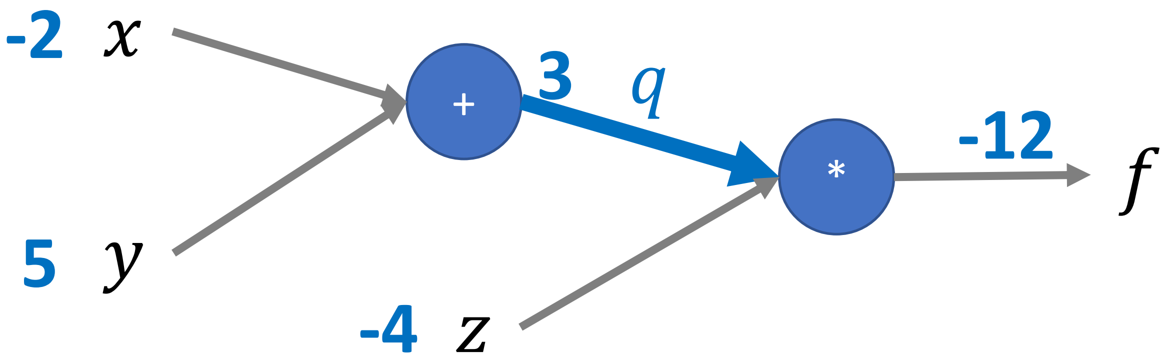
$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$$

Let $x = -2, y = 5, z = -4$,
then $f(x, y, z) = -12$.

Feed Forward

Artificial Neural Networks – Basics

Backpropagation: a simple example



$$f(x, y, z) = (x + y) * z$$

We want to calculate:

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$$

Let $x = -2, y = 5, z = -4$,
then $f(x, y, z) = -12$.

We let the intermediate result be a function

$$q(x, y) = (x + y).$$

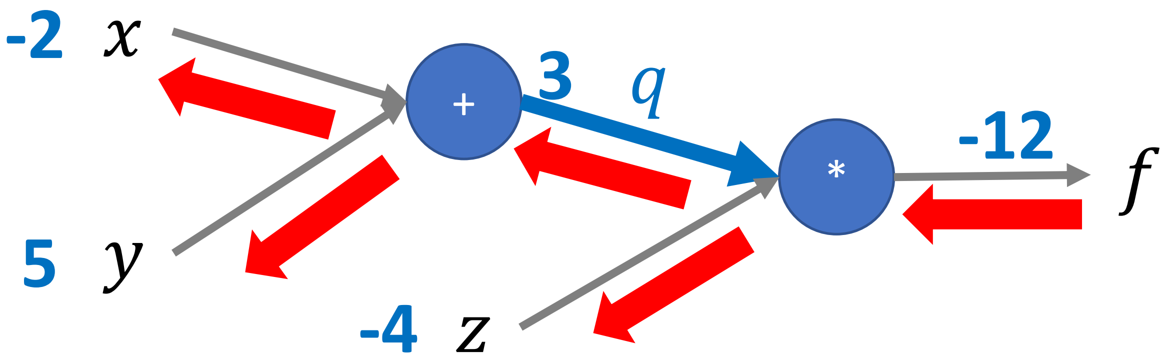
Then $f(x, y, z) = q(x, y)z$.

$$\frac{\partial q}{\partial x} = 1, \qquad \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial q} = z, \qquad \frac{\partial f}{\partial z} = q$$

Artificial Neural Networks – Basics

Backpropagation: a simple example



$$f(x,y,z) = (x+y) * z$$

We want to calculate:

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$$

Let $x=-2, y=5, z=-4$,
then $f(x,y,z) = -12$.

We let the intermediate result be a function

$$q(x,y) = (x+y).$$

Then $f(x,y,z) = q(x,y)z$.

$$\frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

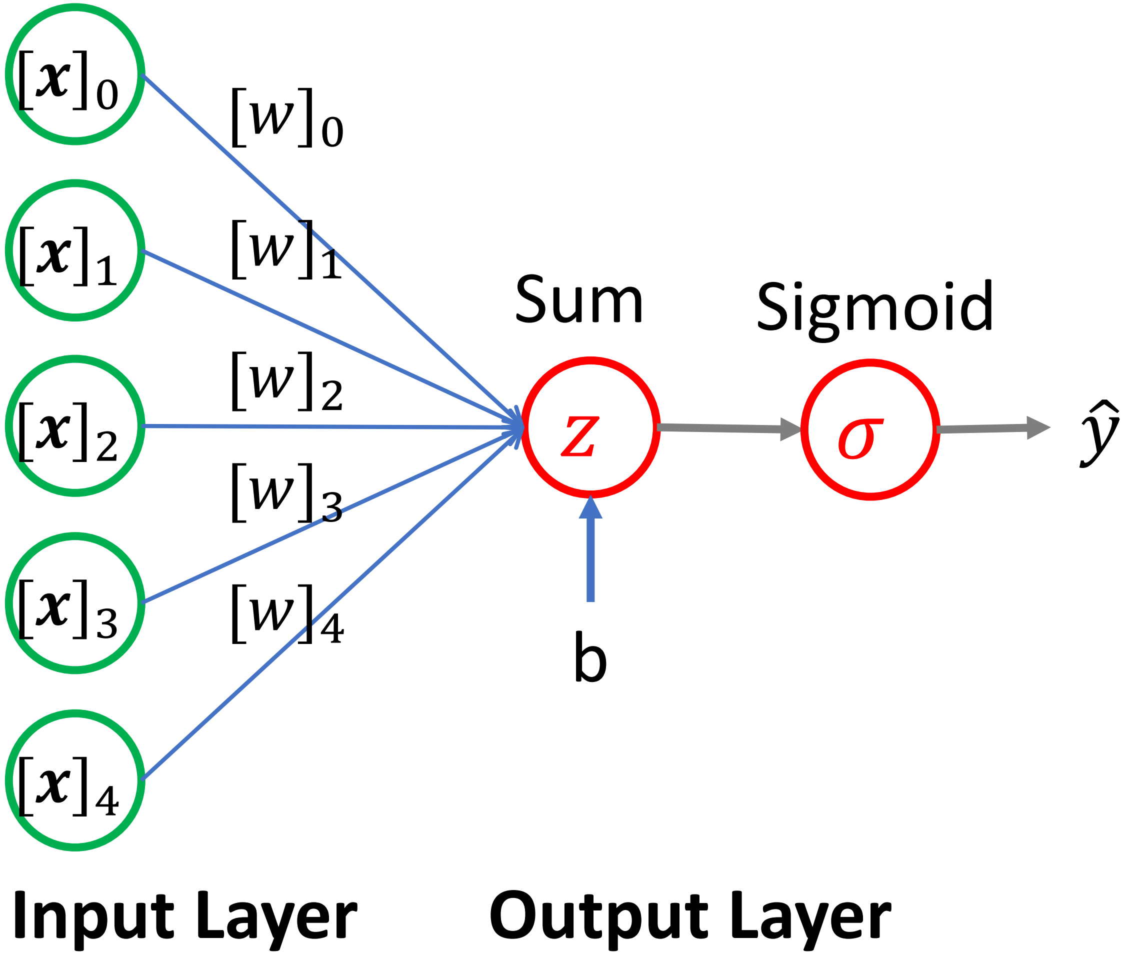
$$\frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

Solution: By using the chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x}, \quad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y}, \quad \frac{\partial f}{\partial z} = q$$

Artificial Neural Networks – Basics

Backpropagation



Given a training sample $(\boldsymbol{x}, y)$, compute the loss

$z = \boldsymbol{x}^\top \boldsymbol{w} + b, \hat{y} = \sigma(z)$,

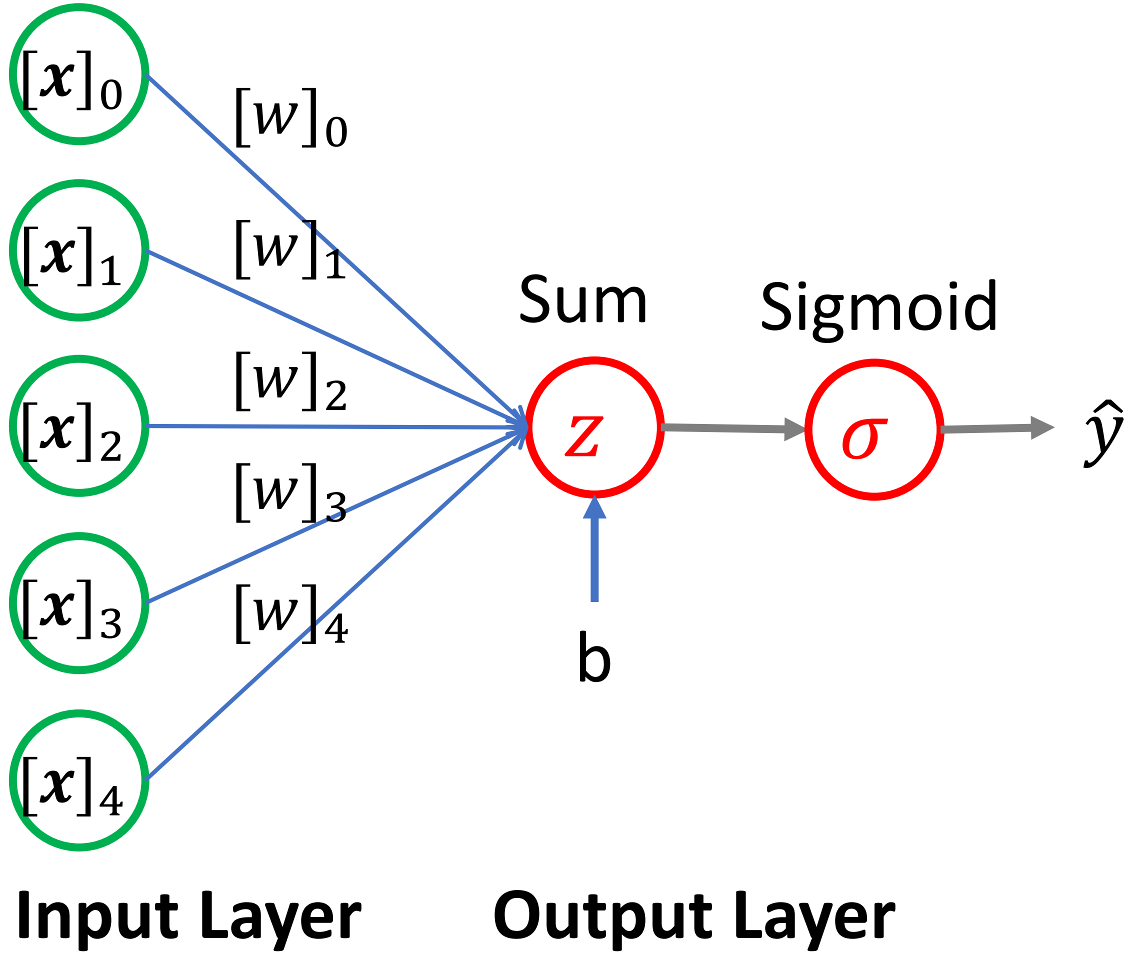
$$\ell = \frac{1}{2}(\hat{y} - y)^2$$

Q: calculate

$$\frac{\partial \ell}{\partial w}, \frac{\partial \ell}{\partial b}$$

Artificial Neural Networks – Basics

Backpropagation



Given a training sample $(\boldsymbol{x}, y)$, compute the loss

$z = \boldsymbol{x}^\top \boldsymbol{w} + b$, $\hat{y} = \sigma(z)$,

$$\ell = \frac{1}{2}(\hat{y} - y)^2$$

Compute the derivatives

$$\frac{d\ell}{d\hat{y}} = (\hat{y} - y),$$

$$\frac{d\ell}{dz} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{dz} = \frac{d\ell}{d\hat{y}} \cdot \sigma(z) \cdot (1 - \sigma(z)),$$

$$\frac{\partial \ell}{\partial w} = \frac{d\ell}{dz} \cdot \frac{dz}{\partial w} = \frac{d\ell}{dz} \cdot \boldsymbol{x},$$

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{db} \cdot \frac{dz}{\partial b} = \frac{d\ell}{dz} \cdot 1.$$

Outline

- Successful stories of DL (2 minutes)
- DL Basics (28 minutes)
- **Build two simple neural networks (45 minutes)**
- TensorFlow and Others (5 minutes)

Build our first neural network

Here is our task

Assume that we have one training example where

$$(\boldsymbol{x} = [0.05, 0.1]^\top, \boldsymbol{y} = [0.01, 0.99]^\top).$$

Given $\boldsymbol{x}$, we want to design a neural network such that it outputs

$$[\boldsymbol{p}]_0 \approx 0.01, [\boldsymbol{p}]_1 \approx 0.99.$$

Obviously, this is a regression problem, we want to minimize the least square loss

$$\min \ell = \frac{1}{2} ||\boldsymbol{p} - \boldsymbol{y}||^2 = \frac{1}{2} \sum_{i=0}^{1} ([p]_i - [y]_i)^2$$

Build our first neural network

Here is our task

Assume that we have one training example where

$$(\boldsymbol{x} = [0.05, 0.1]^\top, \boldsymbol{y} = [0.01, 0.99]^\top).$$

Given $\boldsymbol{x}$, we want to design a neural network such that it outputs

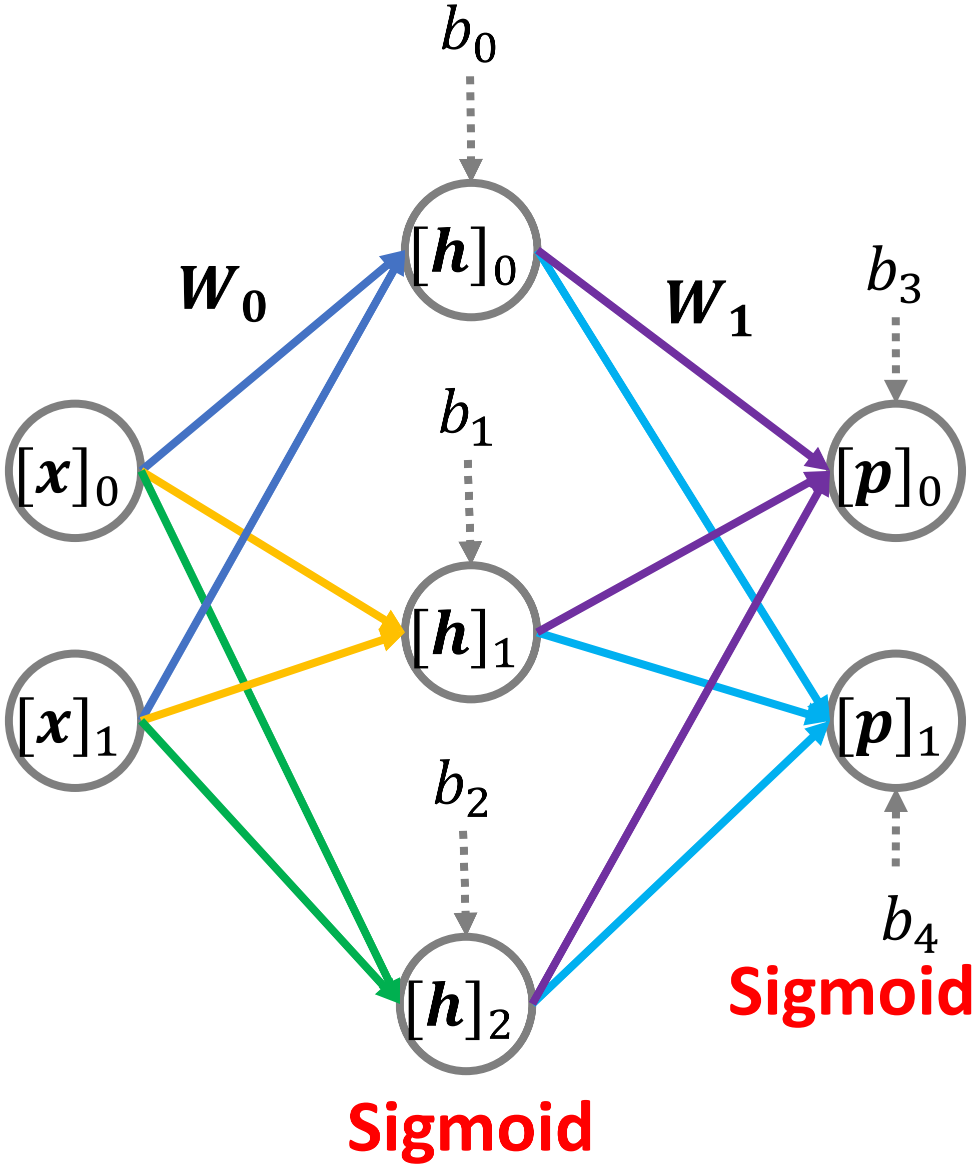
$$[\boldsymbol{p}]_0 \approx 0.01, [\boldsymbol{p}]_1 \approx 0.99.$$

To finish this task, we need to

1. **design a neural net**
2. **implement this neural net**
3. **load the dataset and train the neural net**
4. **check the training losses**

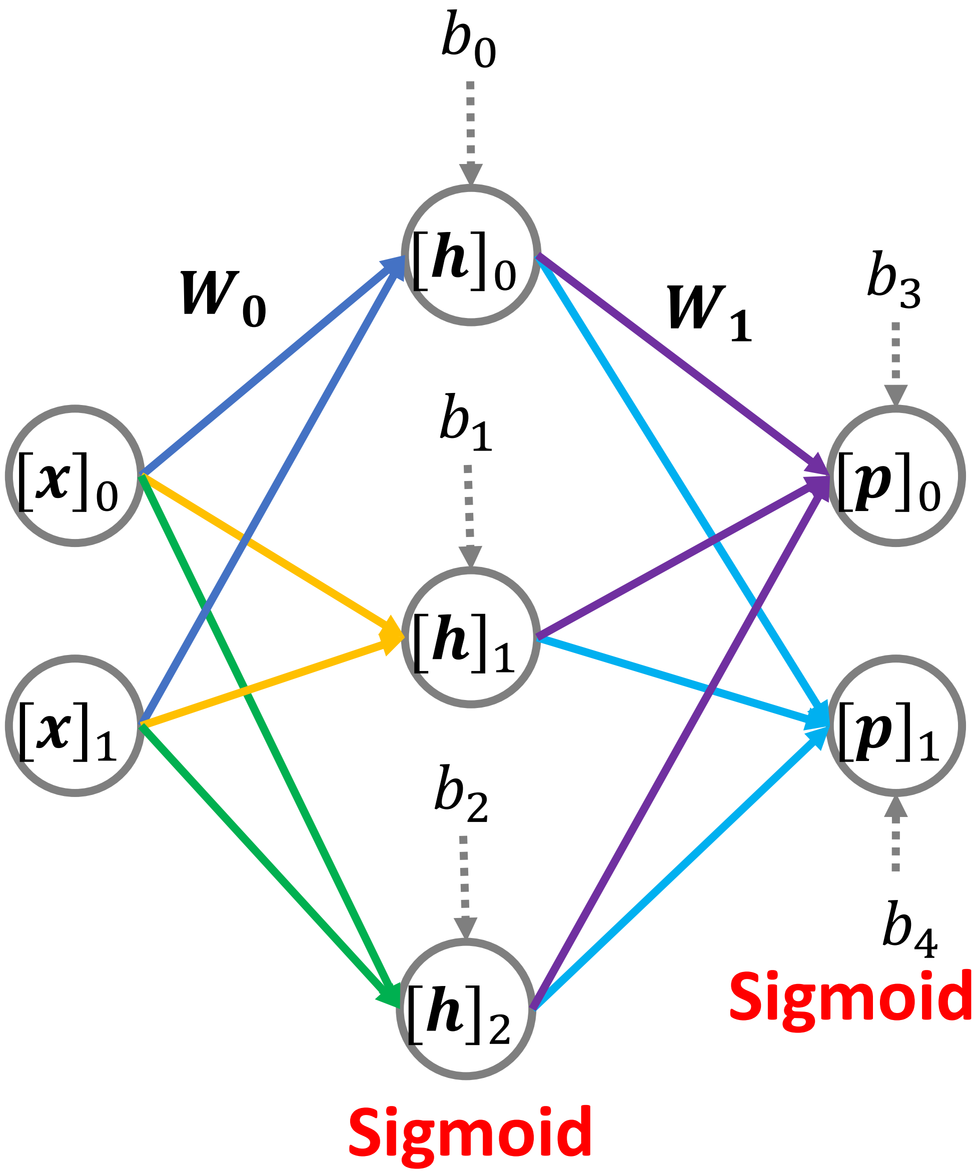
Build our first neural network

1. design a neural network



Build our first neural network

1. design a neural network



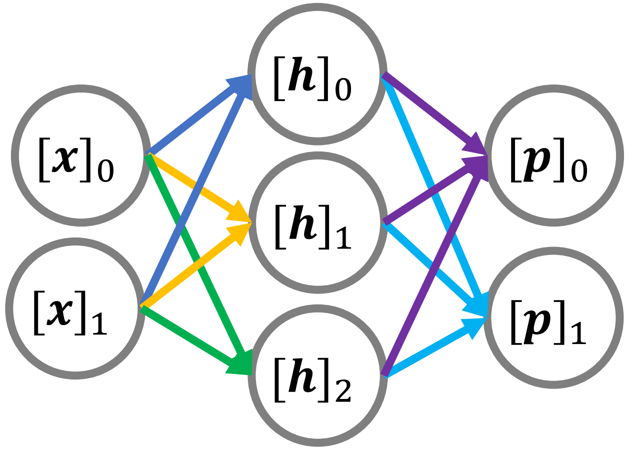
More details:

$$\boldsymbol{W}_0 = \begin{bmatrix} \boldsymbol{w}_0^\top \\ \boldsymbol{w}_1^\top \\ \boldsymbol{w}_2^\top \end{bmatrix}, \ \boldsymbol{W}_1 = \begin{bmatrix} \boldsymbol{w}_3^\top \\ \boldsymbol{w}_4^\top \end{bmatrix}.$$

$$z_0 = [\boldsymbol{x}]_0[\boldsymbol{w}_0]_0 + [\boldsymbol{x}]_1[\boldsymbol{w}_0]_1 + b_0 = \boldsymbol{x}^\top \boldsymbol{w}_0 + b_0,$$
$$z_1 = [\boldsymbol{x}]_0[\boldsymbol{w}_1]_0 + [\boldsymbol{x}]_1[\boldsymbol{w}_1]_1 + b_1 = \boldsymbol{x}^\top \boldsymbol{w}_1 + b_1,$$
$$z_2 = [\boldsymbol{x}]_0[\boldsymbol{w}_2]_0 + [\boldsymbol{x}]_1[\boldsymbol{w}_2]_1 + b_2 = \boldsymbol{x}^\top \boldsymbol{w}_2 + b_2,$$
$$[\boldsymbol{h}]_0 = \sigma(z_0) = \frac{1}{1+e^{-z_0}},$$
$$[\boldsymbol{h}]_1 = \sigma(z_1) = \frac{1}{1+e^{-z_1}},$$
$$[\boldsymbol{h}]_2 = \sigma(z_2) = \frac{1}{1+e^{-z_2}},$$
$$z_3 = [\boldsymbol{h}]_0[\boldsymbol{w}_3]_0 + \cdots + [\boldsymbol{h}]_2[\boldsymbol{w}_3]_2 + b_3 = \boldsymbol{h}^\top \boldsymbol{w}_3 + b_3,$$
$$z_4 = [\boldsymbol{h}]_0[\boldsymbol{w}_4]_0 + \cdots + [\boldsymbol{h}]_2[\boldsymbol{w}_4]_2 + b_4 = \boldsymbol{h}^\top \boldsymbol{w}_4 + b_4,$$
$$[\boldsymbol{p}]_0 = \sigma(z_3) = \frac{1}{1+e^{-z_3}},$$
$$[\boldsymbol{p}]_1 = \sigma(z_4) = \frac{1}{1+e^{-z_4}}.$$

Build our first neural network

2. implement this neural network



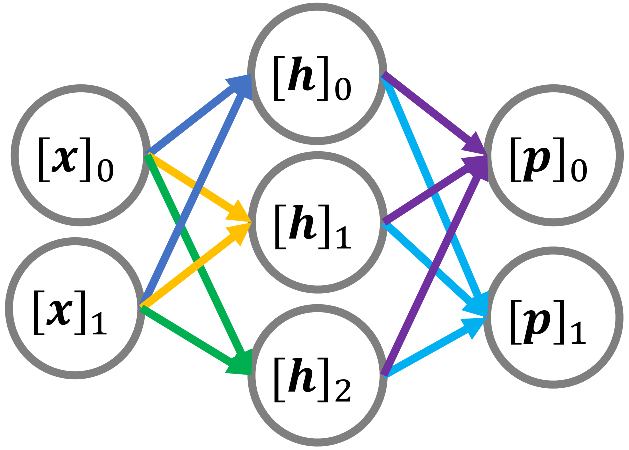
We have the following information about **our network**:

1. The number of inputs (i.e. dimension of the inputs).
2. The number of neurons in the **hidden** layer.
3. The number of neurons in the **output** layer.
4. Hidden layer and output layer use **Sigmoid** as the activation.
5. The loss function used is the **least square loss**.

```
def neural_network_create(num_inputs, num_hidden, num_outputs,
                          hidden_layer_weights=None, hidden_layer_bias=None,
                          output_layer_weights=None, output_layer_bias=None):
    hidden_layer = {'num_neurons': num_hidden,
                    'neurons': [neuron_create(num_inputs, 'sigmoid', hidden_layer_bias, 1, _)
                                for _ in range(num_hidden)]}
    output_layer = {'num_neurons': num_outputs,
                    'neurons': [neuron_create(num_hidden, 'sigmoid', output_layer_bias, 2, _)
                                for _ in range(num_outputs)]}
```

Build our first neural network

2. implement this neural network



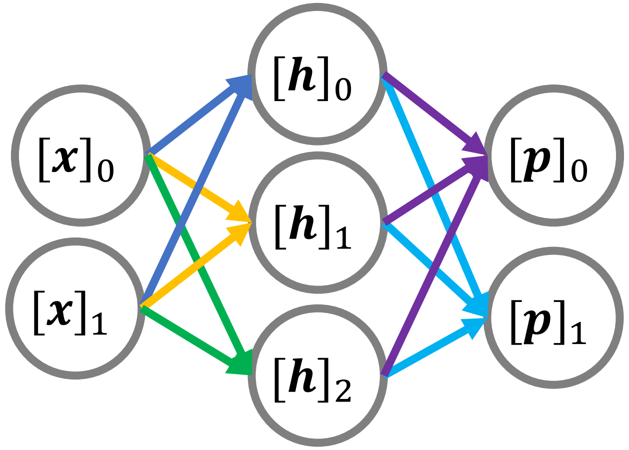
We have the following information about **our neuron**:

1. The number of inputs: 2.
2. Weight vector
3. Input vector (dimension is 2 for hidden layer)
4. Bias
5. Output
6. Activation: Sigmoid

```python
def neuron_create(num_inputs, activation, bias, layer_id, neuron_id):
    return {'weights': np.zeros(num_inputs),
            'inputs': np.zeros(num_inputs),
            'bias': random.random() if bias is None else bias,
            'output': 0.0,
            'activation': activation,
            'layer_id': layer_id,
            'neuron_id': neuron_id}
```

Build our first neural network

2. implement this neural network – Constructed Neural Net




```python
neural_network = {'num_inputs': num_inputs,
                  'num_hidden': num_hidden,
                  'num_outputs': num_outputs,
                  'hidden_layer': hidden_layer,
                  'output_layer': output_layer}
print('created a new neural networks and finished the initialization ...')
return neural_network
```

After we construct the neural net, we need to have the initial weights ...

Build our first neural network

2. implement this neural network - Initialization

initialize weights from inputs to hidden layer neurons

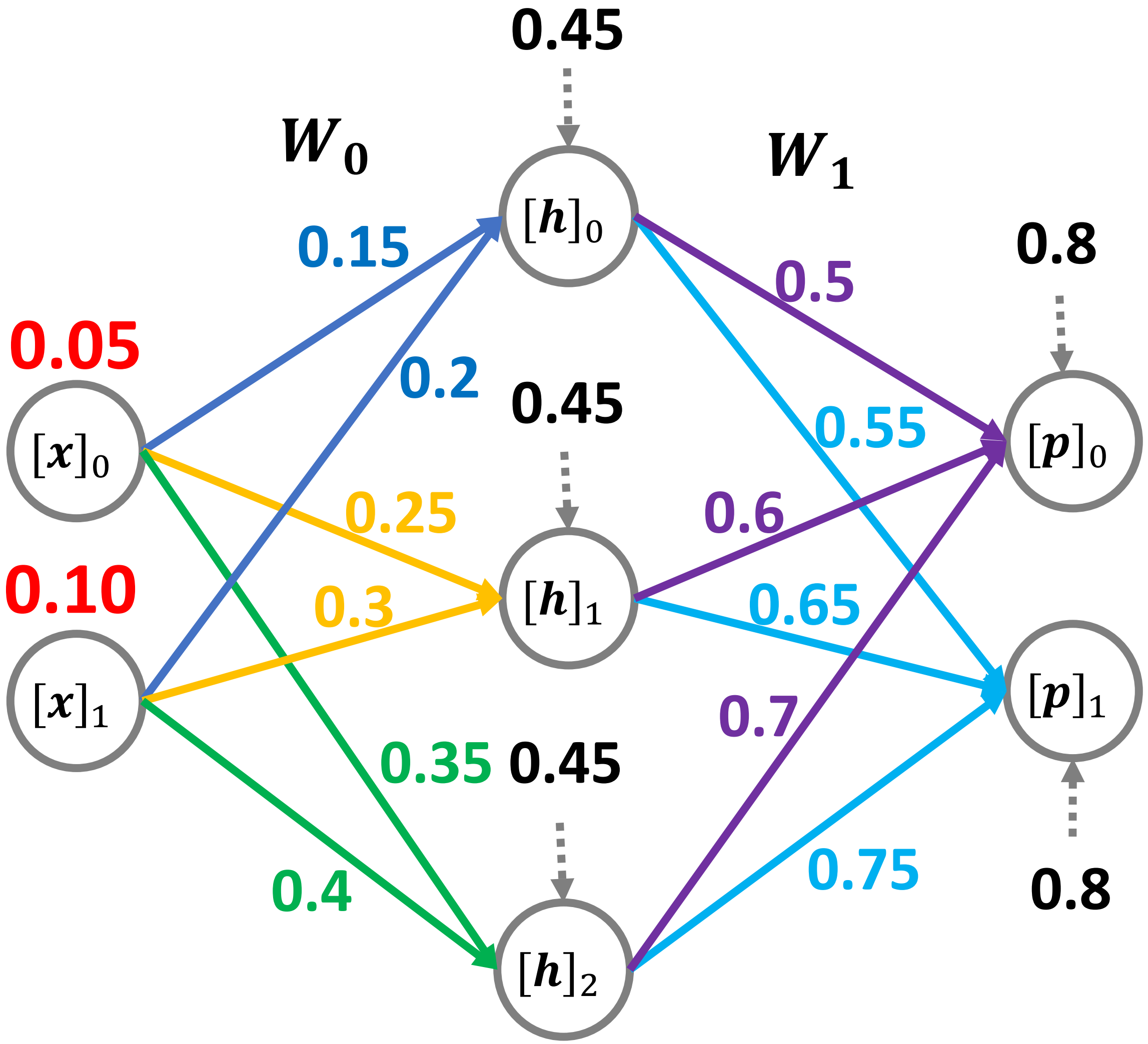
```
weight_num = 0
for h in range(hidden_layer['num_neurons']):
    for i in range(num_inputs):
        if not hidden_layer_weights:
            hidden_layer['neurons'][h]['weights'][i] = random.random()
        else:
            hidden_layer['neurons'][h]['weights'][i] = hidden_layer_weights[weight_num]
        weight_num += 1
```

initialize weights from hidden layer neurons to output layer neurons

```
weight_num = 0
for o in range(output_layer['num_neurons']):
    for h in range(hidden_layer['num_neurons']):
        if not output_layer_weights:
            output_layer['neurons'][o]['weights'][h] = random.random()
        else:
            output_layer['neurons'][o]['weights'][h] = output_layer_weights[weight_num]
        weight_num += 1
```

Build our first neural network

2. implement this neural network – Calculate the square loss



Proposed neural net

$$\boldsymbol{W_0} = \begin{bmatrix} 0.15 & 0.2 \\ 0.25 & 0.3 \\ 0.35 & 0.4 \end{bmatrix}, \quad \boldsymbol{W_1} = \begin{bmatrix} 0.5 & 0.6 & 0.7 \\ 0.55 & 0.65 & 0.75 \end{bmatrix}.$$

$z_0 = 0.05 * 0.15 + 0.10 * 0.2 + 0.45 = 0.4775,$

$z_1 = 0.05 * 0.25 + 0.10 * 0.3 + 0.45 = 0.4925,$

$z_2 = 0.05 * 0.35 + 0.10 * 0.4 + 0.45 = 0.5057,$

$[\boldsymbol{h}]_0 = \frac{1}{1+e^{-z_0}} \approx ?,$

$[\boldsymbol{h}]_1 = \frac{1}{1+e^{-z_1}} \approx ?,$

$[\boldsymbol{h}]_2 = \frac{1}{1+e^{-z_2}} \approx ?,$

$z_3 \approx ?, z_4 \approx ?,$

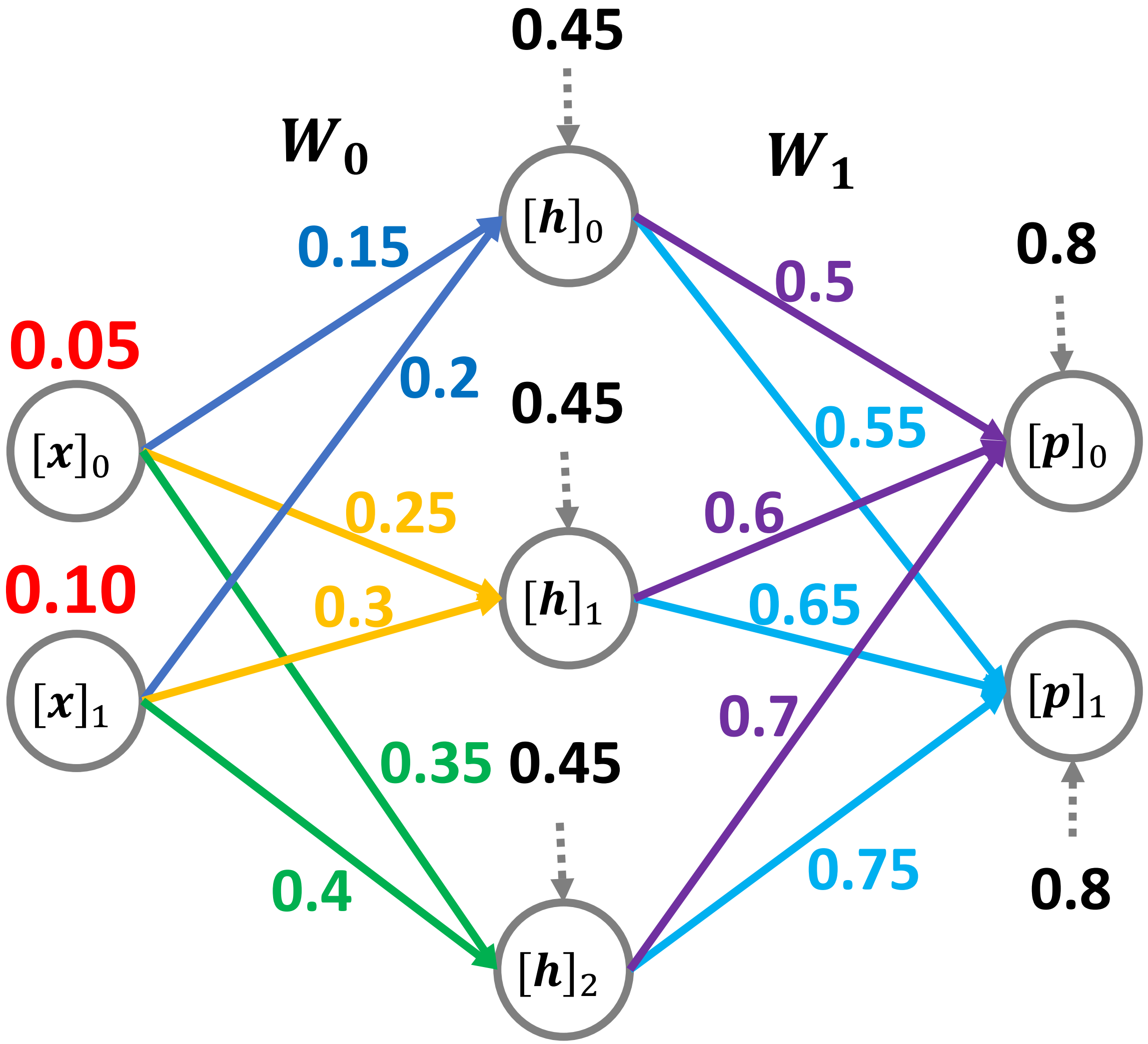
$[\boldsymbol{p}]_0 = \frac{1}{1+e^{-z_3}} \approx ?, [\boldsymbol{p}]_1 = \frac{1}{1+e^{-z_4}} \approx ?$

The loss ℓ ?

Quiz!

Build our first neural network

2. implement this neural network – Calculate the square loss



Proposed neural net

$$W_0 = \begin{bmatrix} 0.15 & 0.2 \\ 0.25 & 0.3 \\ 0.35 & 0.4 \end{bmatrix},\ W_1 = \begin{bmatrix} 0.5 & 0.6 & 0.7 \\ 0.55 & 0.65 & 0.75 \end{bmatrix}.$$

$z_0 = 0.05 * 0.15 + 0.10 * 0.2 + 0.45 = 0.4775,$

$z_1 = 0.05 * 0.25 + 0.10 * 0.3 + 0.45 = 0.4925,$

$z_2 = 0.05 * 0.35 + 0.10 * 0.4 + 0.45 = 0.5057,$

$[h]_0 = \frac{1}{1+e^{-z_0}} \approx 0.6172,$

$[h]_1 = \frac{1}{1+e^{-z_1}} \approx 0.6207,$

$[h]_2 = \frac{1}{1+e^{-z_2}} \approx 0.6242,$

$z_3 \approx 1.8245,\ z_4 \approx 2.1038,$

$[p]_0 = \frac{1}{1+e^{-z_3}} \approx 0.8611, [p]_1 = \frac{1}{1+e^{-z_4}} \approx 0.8913.$

The loss ℓ ≈0.3671

Build our first neural network

2. implement this neural network – Calculate the square loss

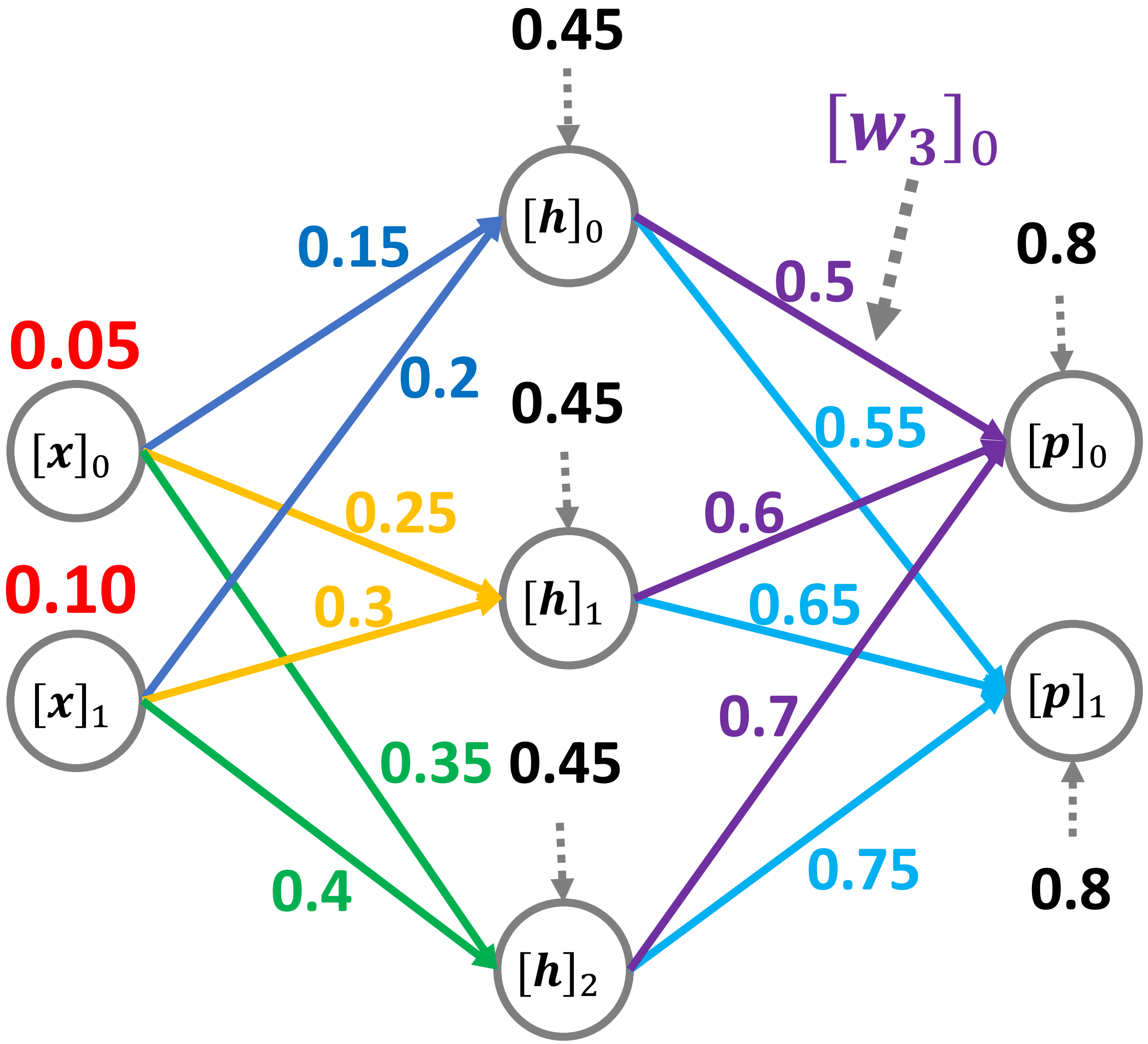
Feed Forward

```
def neural_network_feed_forward(neural_network, train_xi):
    hidden_layer = neural_network['hidden_layer']
    output_layer = neural_network['output_layer']
    # hidden layer feeds forward activation.
    hidden_layer_outputs = np.zeros(hidden_layer['num_neurons'])
    for index, neuron in enumerate(hidden_layer['neurons']):
        hidden_layer_outputs[index] = neuron_cal_output(neuron, train_xi)
    # output layer feeds forward activation.
    outputs = np.zeros(hidden_layer['num_neurons'])
    for index, neuron in enumerate(output_layer['neurons']):
        outputs[index] = neuron_cal_output(neuron, hidden_layer_outputs)
    # final prediction
    return outputs
```

Given the squared loss, we can how calculate the gradient w.r.t w ?

Build our first neural network

2. implement this neural network – Calculate the gradient



Proposed neural net

Given the loss, how can we calculate the gradient w.r.t $\boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{b}$?

Backpropagation!

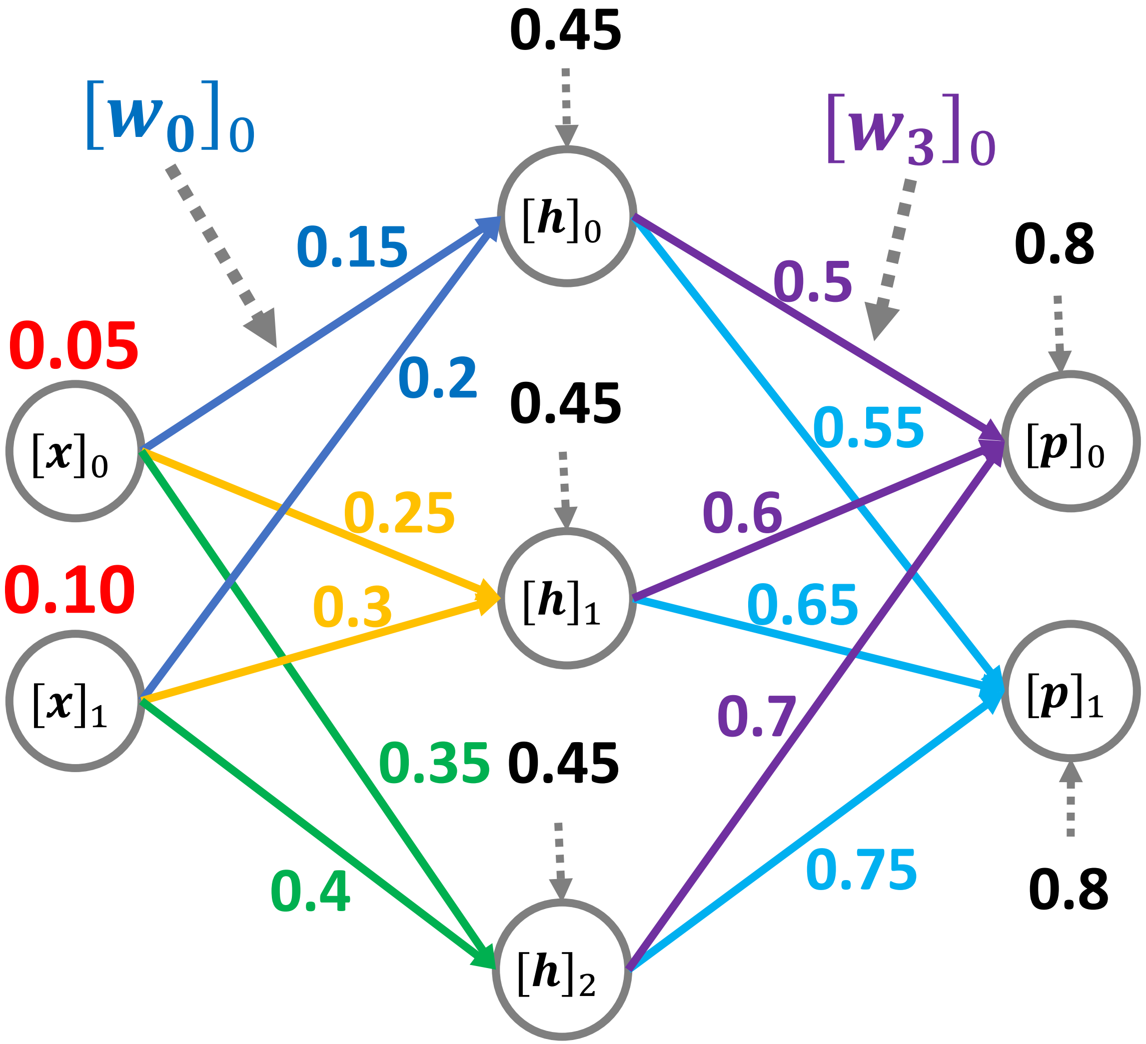
For example, we to calculate the gradient of ℓ w.r.t. $[\boldsymbol{w_3}]_0$.

$$\frac{\partial \ell}{\partial [\boldsymbol{w_3}]_0} = \frac{\partial \ell}{\partial [\boldsymbol{p}]_0} \cdot \frac{\partial [\boldsymbol{p}]_0}{\partial z_3} \cdot \frac{\partial z_3}{\partial [\boldsymbol{w_3}]_0}$$

$$= ([\boldsymbol{p}]_0 - [\boldsymbol{y}]_0) \cdot \left(\sigma(z_3)\big(1 - \sigma(z_3)\big)[\boldsymbol{h}]_0\right).$$

Build our first neural network

2. implement this neural network – Calculate the gradient



Proposed neural net

Given the loss, how can we calculate the gradient w.r.t $\boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{b}$?

Backpropagation!

For example, we to calculate the gradient of ℓ w.r.t. $[\boldsymbol{w_3}]_0$.

$$\frac{\partial \ell}{\partial [\boldsymbol{w_3}]_0} = \frac{\partial \ell}{\partial [\boldsymbol{p}]_0} \cdot \frac{\partial [\boldsymbol{p}]_0}{\partial z_3} \cdot \frac{\partial z_3}{\partial [\boldsymbol{w_3}]_0}$$
$$= ([\boldsymbol{p}]_0 - [\boldsymbol{y}]_0) \cdot \sigma(z_3)(1 - \sigma(z_3))[\boldsymbol{h}]_0$$
$$= ([\boldsymbol{p}]_0 - [\boldsymbol{y}]_0) \cdot [p]_0(1 - [p]_0)[\boldsymbol{h}]_0$$

Use chain rule, to calculate

$$\frac{\partial \ell}{\partial [\boldsymbol{w_0}]_0} = 0.0006$$

Quiz!

Build our first neural network

2. implement this neural network – Update the weights

Update all weights by using stochastic gradient descent

$$[w_i]_j^{t+1} = [w_i]_j^t - \eta \frac{\partial \ell}{\partial [w_i]_j^t}, \text{where } \eta = 0.5$$

```
# 3. Update output neuron weights
for o in range(output_layer['num_neurons']):
    for w_ho in range(len(output_layer['neurons'][o]['weights'])):
        # ∂Eⱼ/∂wᵢⱼ = ∂E/∂zⱼ * ∂zⱼ/∂wᵢⱼ
        pd = neuron_cal_pd_total_net_input_wrt_weight(output_layer['neurons'][o], w_ho)
        pd_error_wrt_weight = pd_errors_wrt_output_neuron_total_net_input[o] * pd
        # Δw = α * ∂Eⱼ/∂wᵢ
        output_layer['neurons'][o]['weights'][w_ho] -= learning_rate * pd_error_wrt_weight
```

```
# 4. Update hidden neuron weights
for h in range(hidden_layer['num_neurons']):
    for w_ih in range(len(hidden_layer['neurons'][h]['weights'])):
        # ∂Eⱼ/∂wᵢ = ∂E/∂zⱼ * ∂zⱼ/∂wᵢ
        pd = neuron_cal_pd_total_net_input_wrt_weight(hidden_layer['neurons'][h], w_ih)
        pd_error_wrt_weight = pd_errors_wrt_hidden_neuron_total_net_input[h] * pd
        # Δw = α * ∂Eⱼ/∂wᵢ
        hidden_layer['neurons'][h]['weights'][w_ih] -= learning_rate * pd_error_wrt_weight
```

Build our first neural network

3. load the dataset and train the neural network

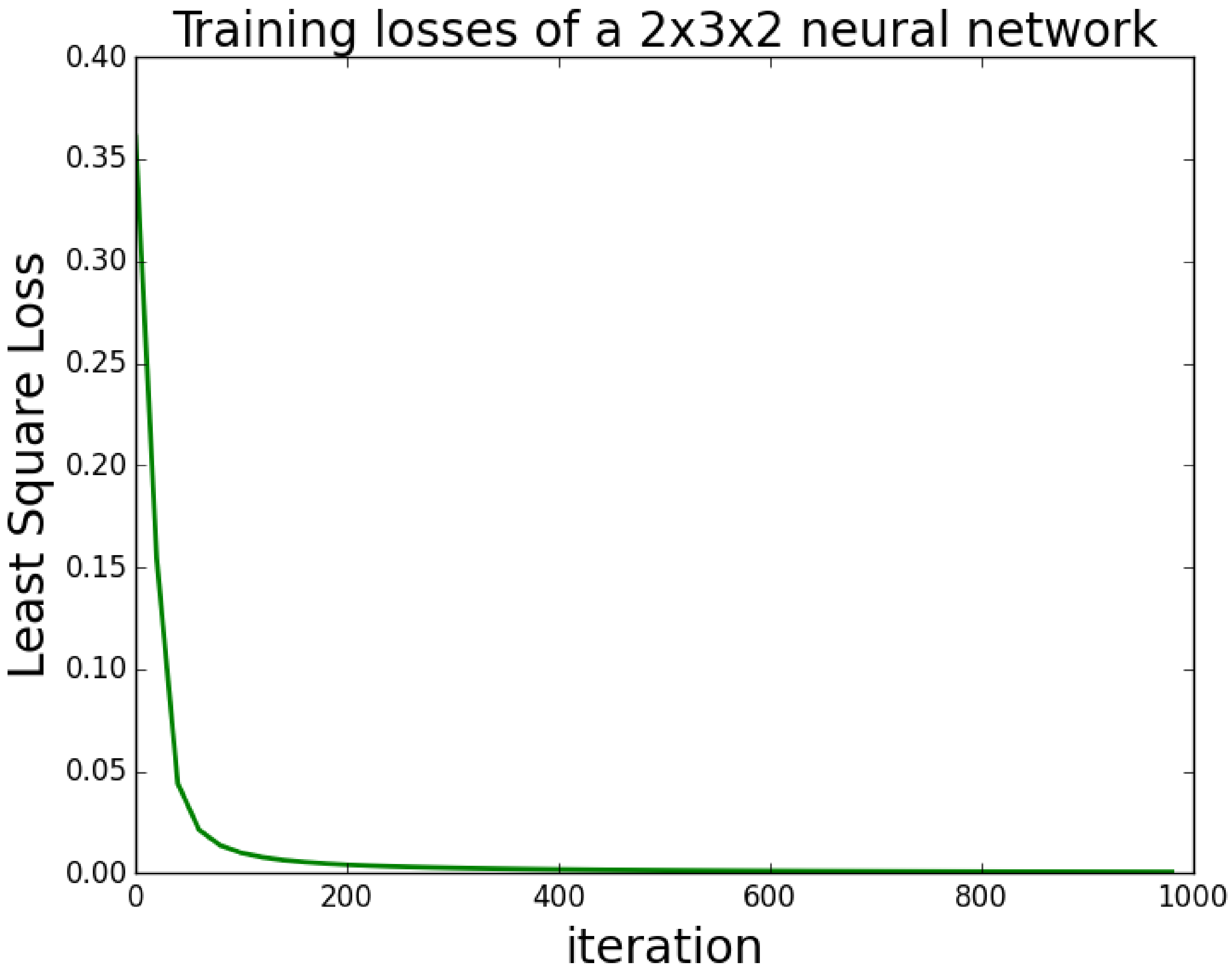
```
train_x = [[0.05, 0.1]]
train_y = [[0.01, 0.99]]
for i in range(1000):
    train_xi, train_yi = train_x[0], train_y[0]
    neural_network_train(nn, train_x=train_xi, train_y=train_yi, learning_rate=.5)
    if i % 5 == 0:
        cur_loss = neural_network_total_error(nn, train_x=train_x, train_y=train_y)
        train_losses.append(cur_loss)
        print(i, train_losses[-1])
```

How can we decide whether this neural network is a good one? There is no a standard criteria, but we can at least take a loot at the losses during the training process.

Important: the total training error will decrease in general.

Build our first neural network

4. check the training losses



As we can see, during the learning process, the total loss is always decreasing.

Quiz!

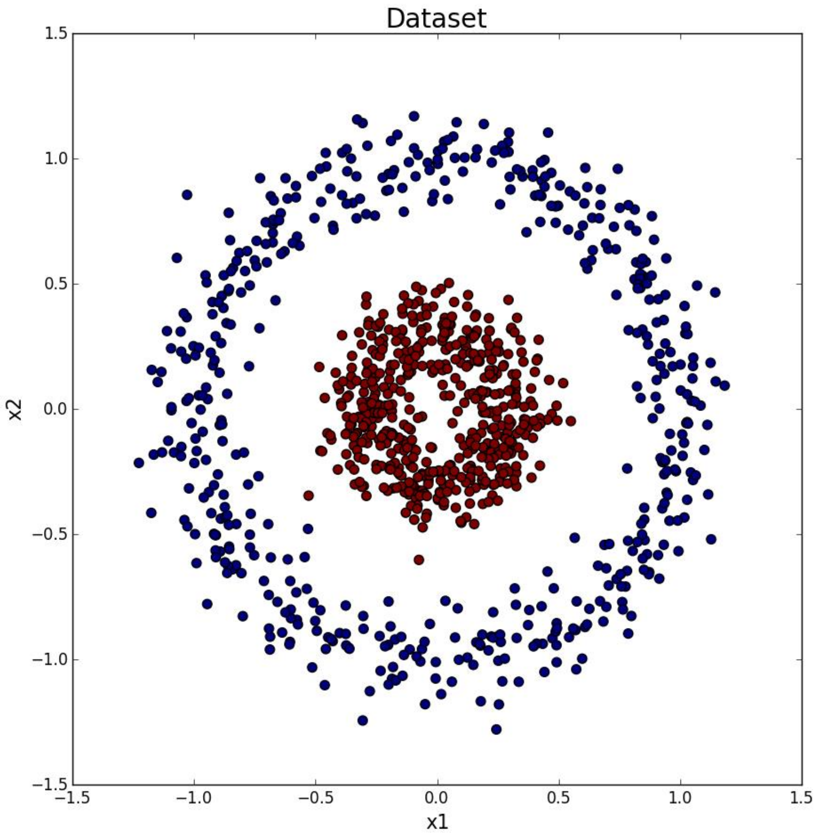
Try to run the program!

Build our second neural network

Here is our task:

Suppose, we have a classification problem.

Each $(\boldsymbol{x_i}, y_i)$ is a training sample from the following distribution



We want to minimize the cross-entropy loss

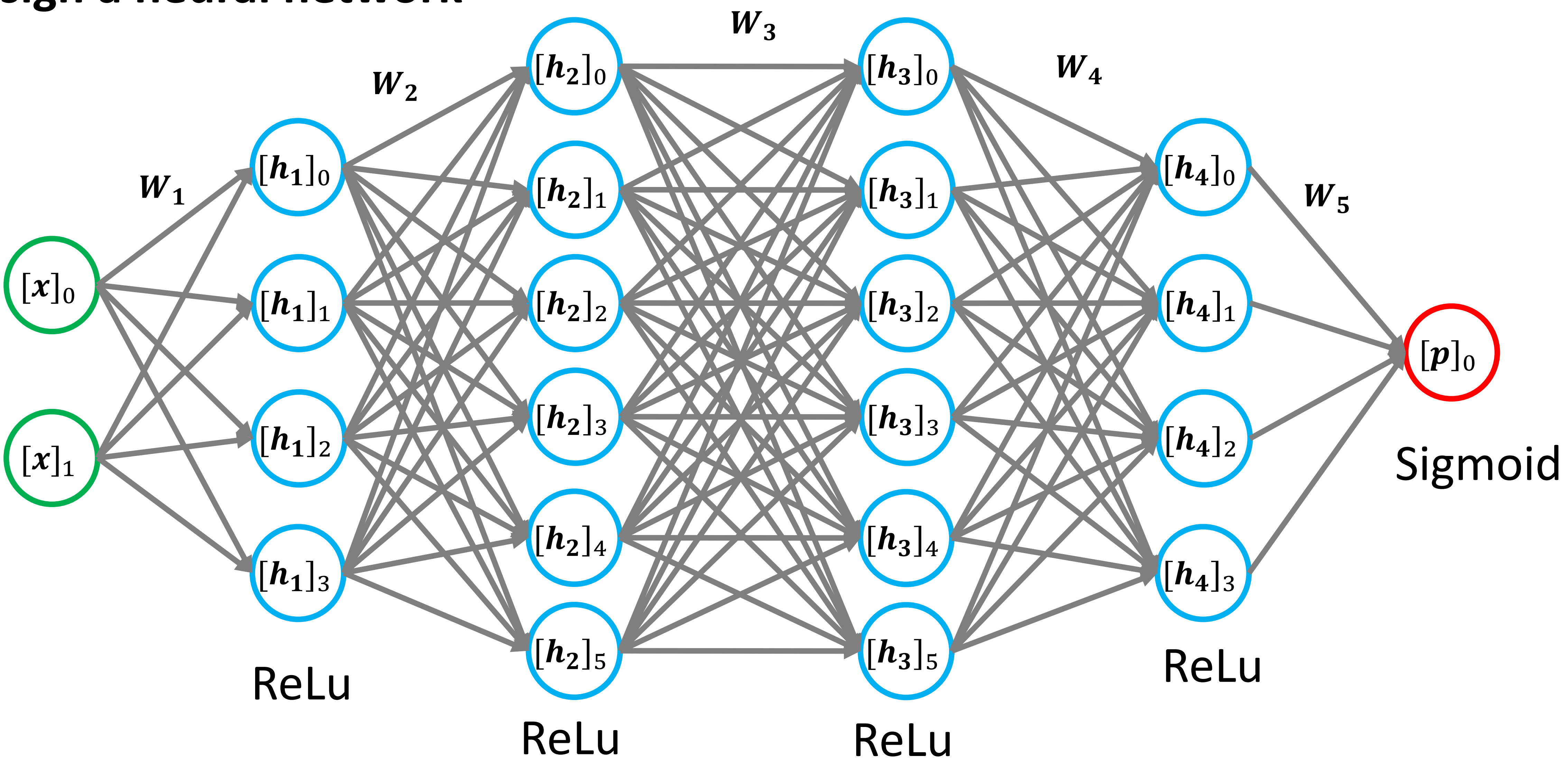
$$\ell = \frac{1}{n}\sum_{i=1}^{n} -y_i \log p_i$$

To finish this task, we need to

1. design a neural network
2. implement this neural network
3. load the dataset and train the neural net
4. check the training losses

Build our second neural network

1. design a neural network



Build our second neural network

2. implement this neural network

```
def neural_network_create(input_dim, list_node_sizes, output_dim=2):
    model = dict()
    # first hidden layer
    model['W1'] = np.random.randn(input_dim, list_node_sizes[0])
    model['b1'] = np.zeros((1, list_node_sizes[0]))
    # second hidden layer
    model['W2'] = np.random.randn(list_node_sizes[0], list_node_sizes[1])
    model['b2'] = np.zeros((1, list_node_sizes[1]))
    # third hidden layer
    model['W3'] = np.random.randn(list_node_sizes[1], list_node_sizes[2])
    model['b3'] = np.zeros((1, list_node_sizes[2]))
    # fourth hidden layer
    model['W4'] = np.random.randn(list_node_sizes[2], list_node_sizes[3])
    model['b4'] = np.zeros((1, list_node_sizes[3]))
    # output layer
    model['W5'] = np.random.randn(list_node_sizes[3], output_dim)
    model['b5'] = np.zeros((1, output_dim))
    return model
```

Build our second neural network

3. load the dataset and train the neural network

```
def test_batch(x_tr, y_tr, x_te, y_te):
    model = neural_network_create(input_dim=x_tr.shape[1], list_node_sizes=[4, 6, 6, 4],
output_dim=2)
    train_batch(model, x_tr, y_tr, num_passes=50, learning_rate=0.001)
    output = neural_network_feed_forward(model, x_te)
    success = 0
    for ind, item in enumerate(output[-1]):
        if y_te[ind] == np.argmax(item):
            success += 1
    print(success / float(len(y_te)))


def test_stochastic(x_tr, y_tr, x_te, y_te):
    model = neural_network_create(input_dim=x_tr.shape[1], list_node_sizes=[4, 6, 6, 4],
output_dim=2)
    train_stochastic(model=model, x_tr=x_tr, y_tr=y_tr, num_passes=50, batch_size=20,
learning_rate=0.001)
    output = neural_network_feed_forward(model, x_te)
    success = 0
    for ind, item in enumerate(output[-1]):
        if y_te[ind] == np.argmax(item):
            success += 1
    print('test accuracy: %.4f' % (success / float(len(y_te))))
```

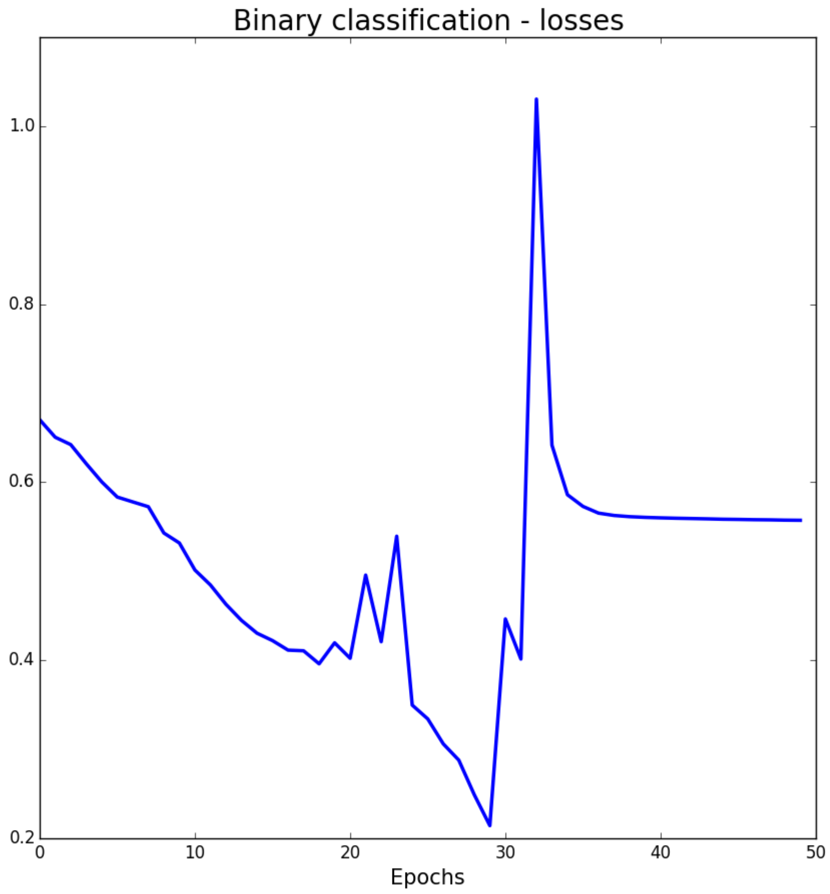
Try to use two different ways:

1. **Batch training**
2. **Stochastic training**

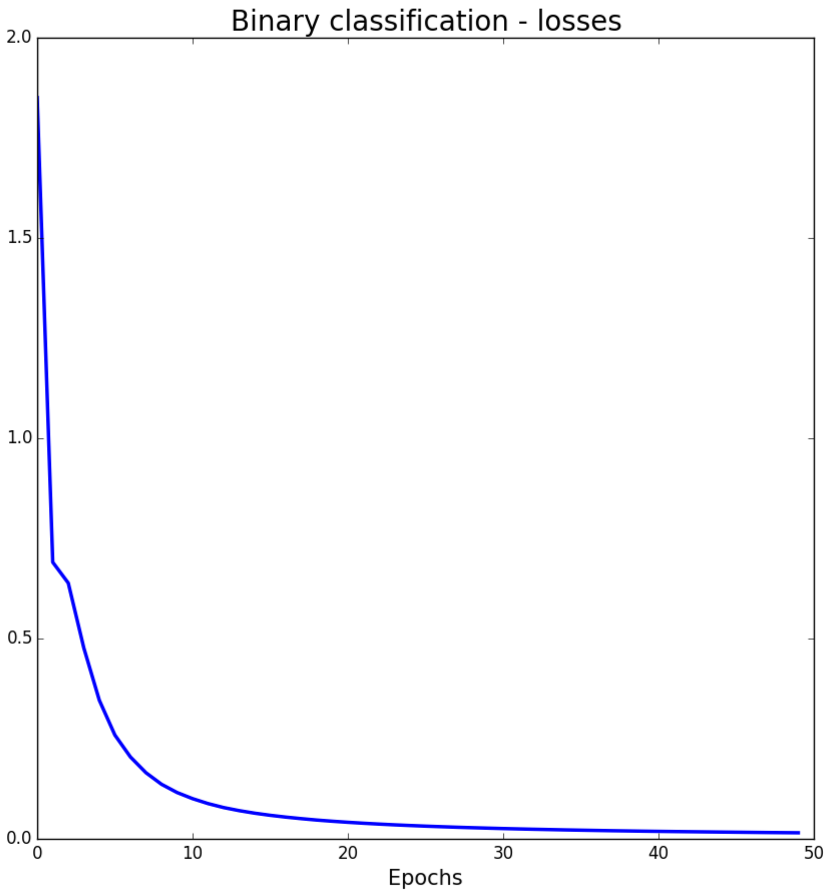
Build our second neural network

4. check the training losses

Batch training



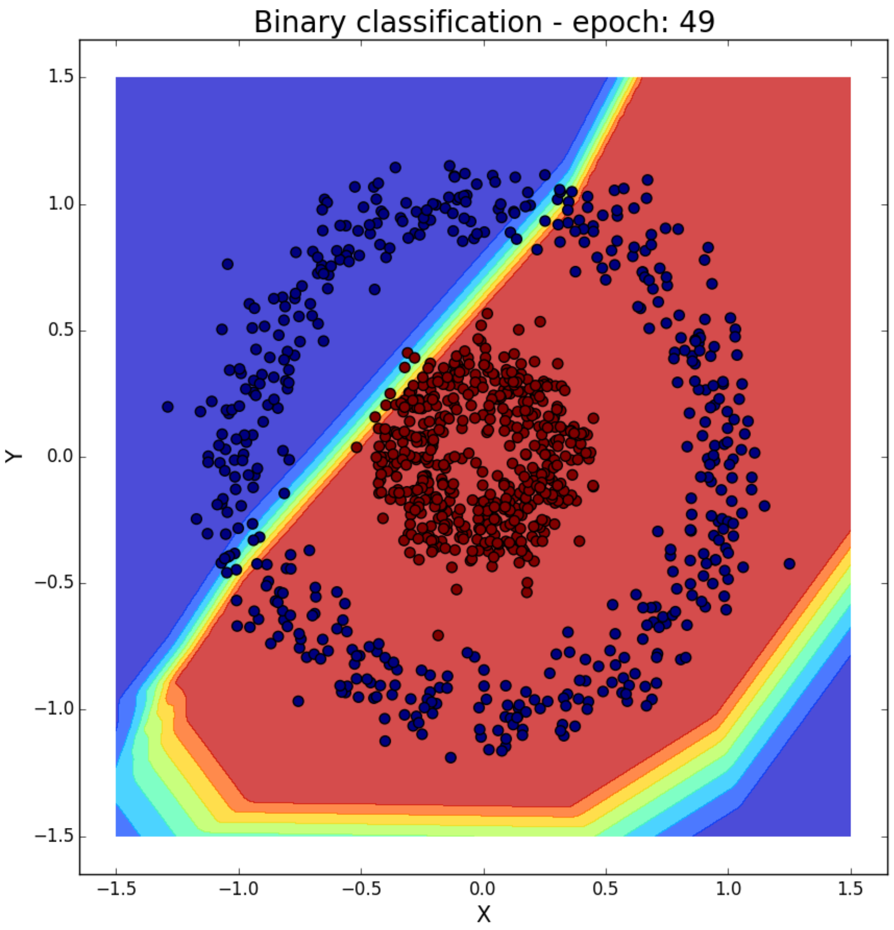
Stochastic training



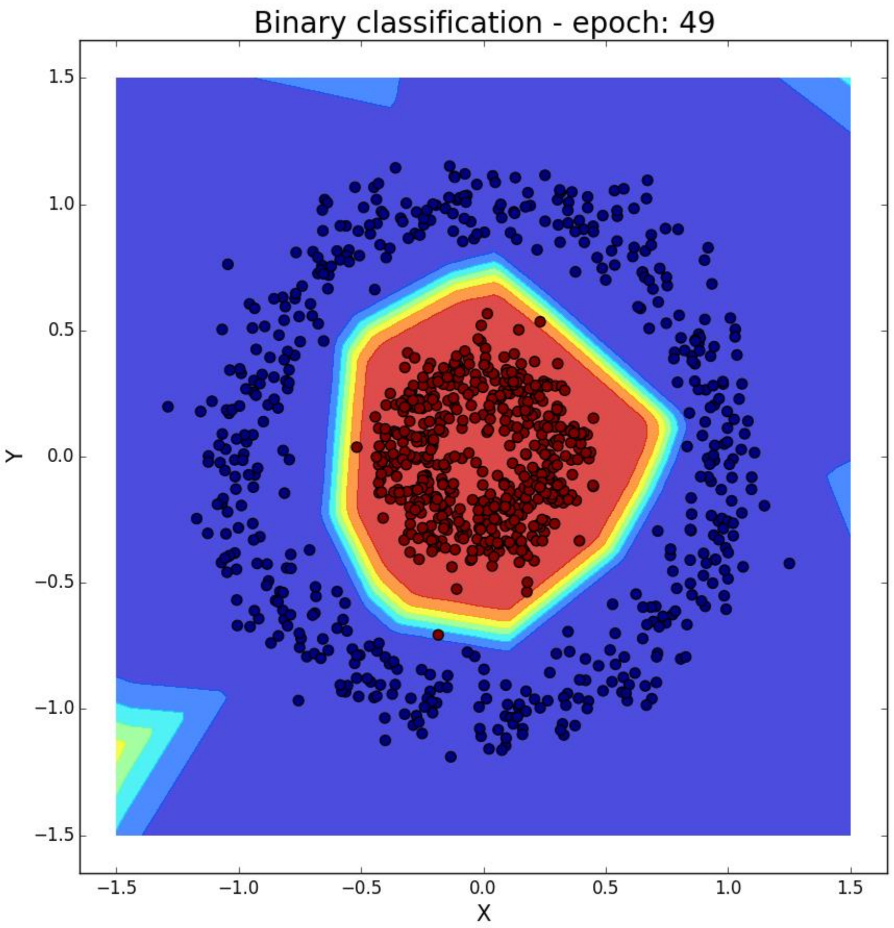
Build our second neural network

4. check the model

Batch training



Stochastic training



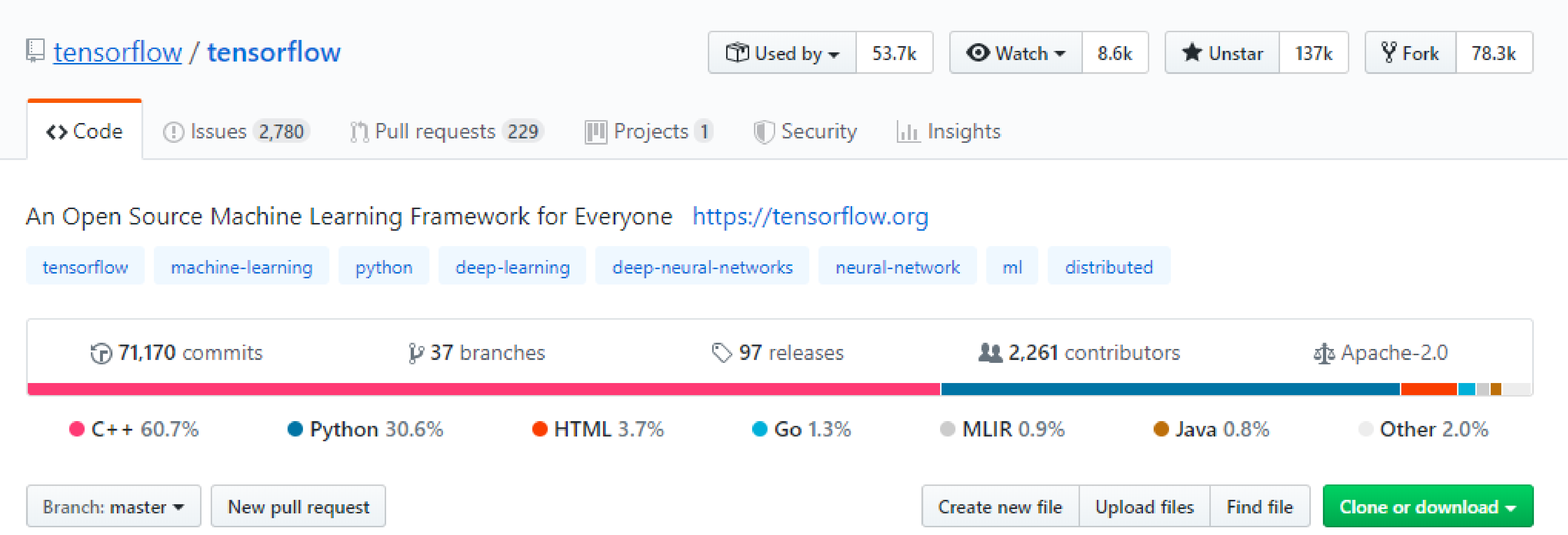
Quiz!

Try to run the code several times to find a potential pattern.

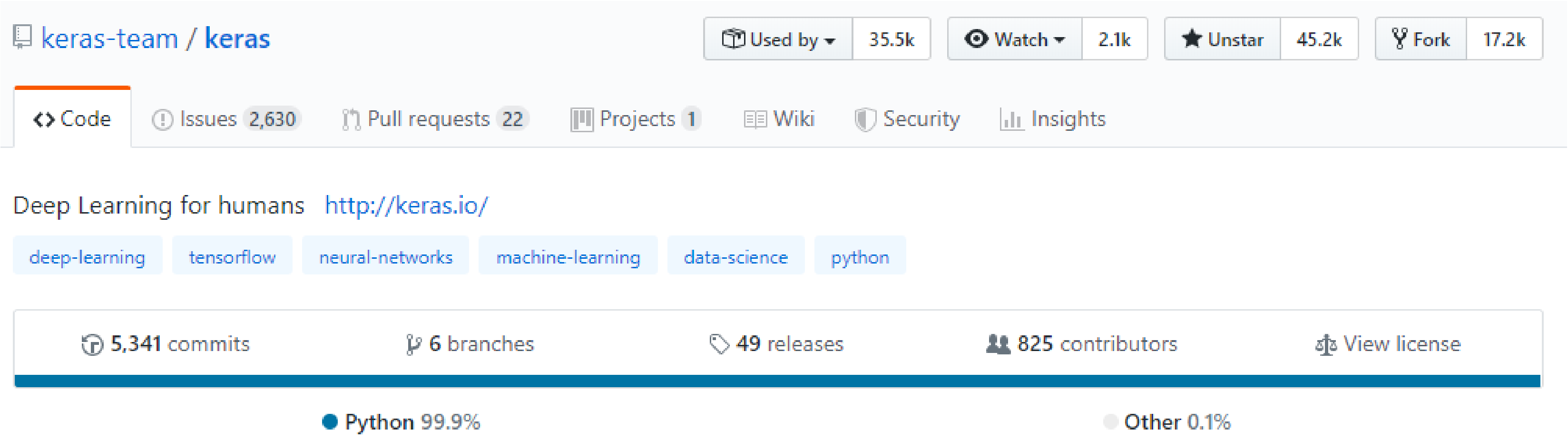
Outline

- **Successful stories of DL (2 minutes)**
- **DL Basics (28 minutes)**
- **Build two simple neural networks (45 minutes)**
- **TensorFlow and Others (5 minutes)**

TensorFlow – Overview

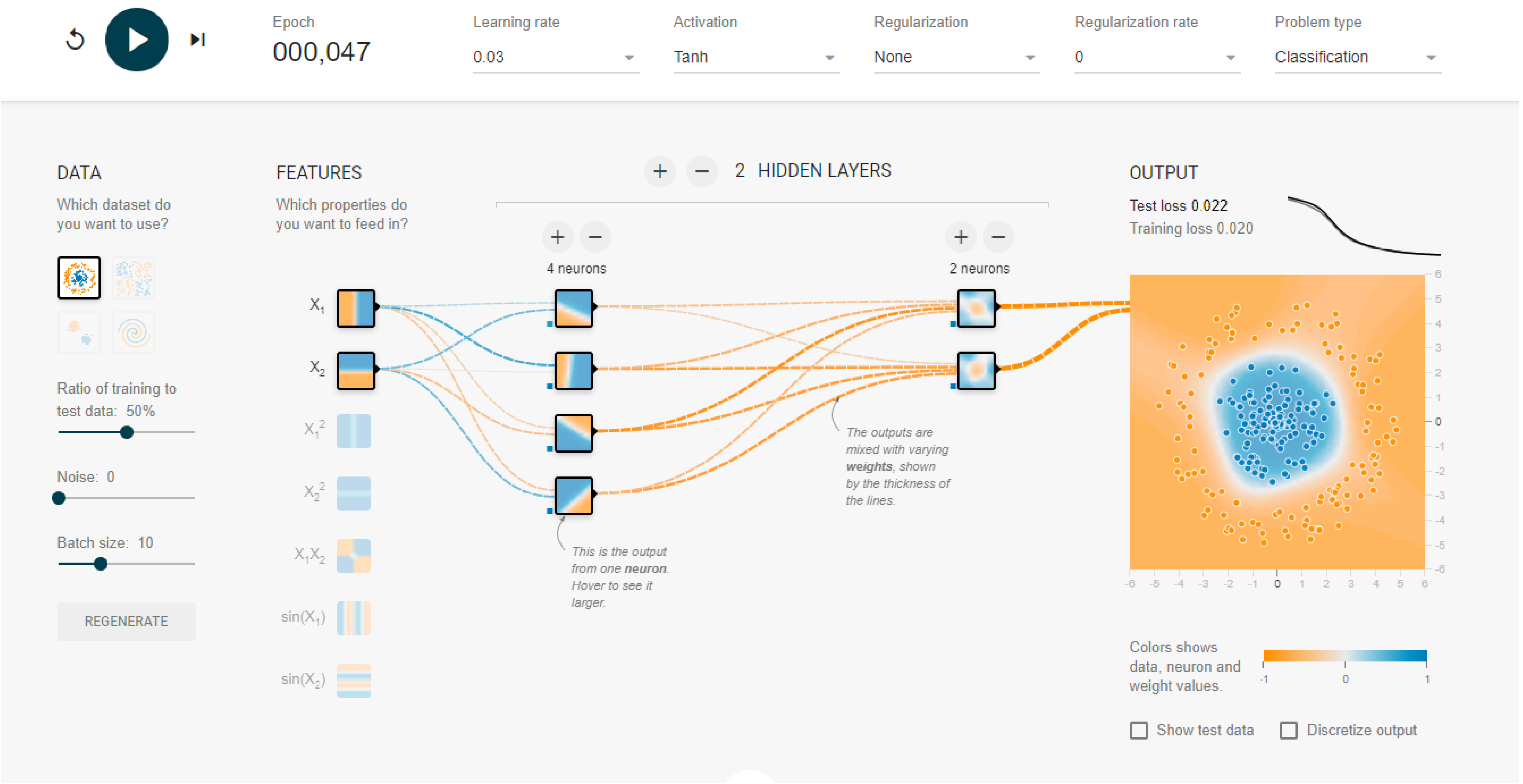


Keras

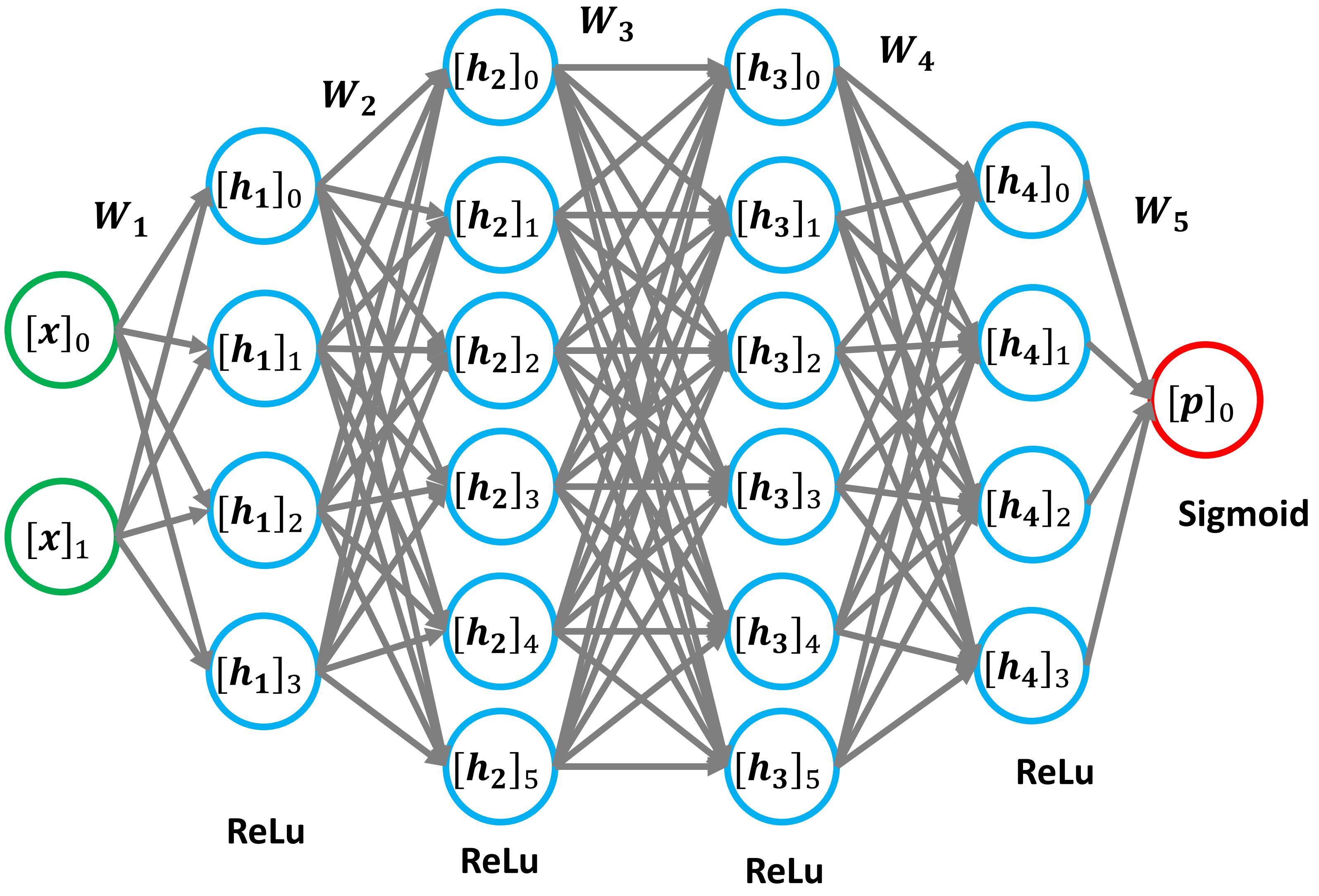


TensorFlow – Playground

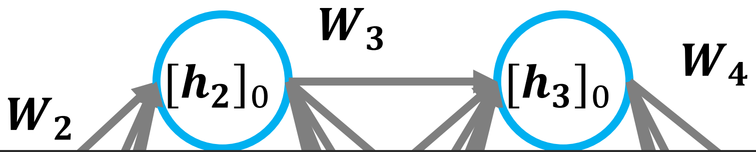
Have fun: https://playground.tensorflow.org



TensorFlow – Build a deep learning model

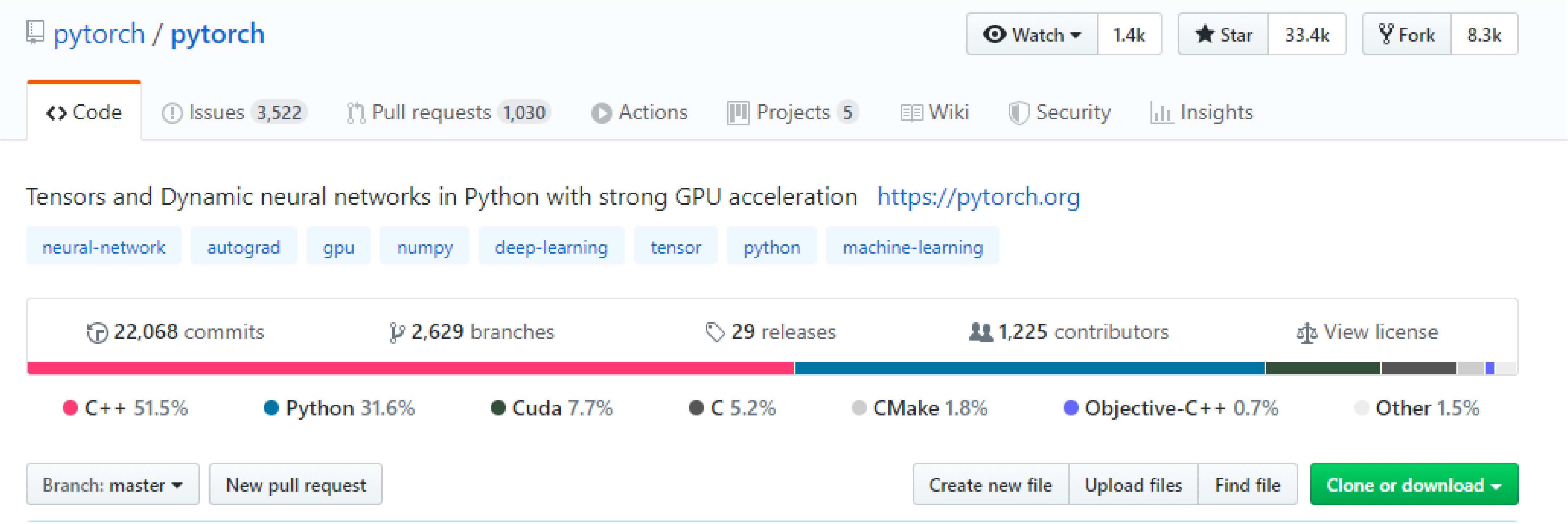


TensorFlow – Build a deep learning model




```python
from keras.layers import Dense
from keras.models import Sequential
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

PyTorch - https://github.com/pytorch/pytorch



References

- https://github.com/HarisIqbal88/PlotNeuralNet
- http://www.emergentmind.com/neural-network
- http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf
- https://medium.com/@dcharrezt/neurips-2019-stats-c91346d31c8f [word cloud, NIPS, 2019]
- https://www.reddit.com/r/MachineLearning/comments/9jhhvb/d_iclr_2019_submissions_are_viewable_which_ones/ [word cloud, ICLR, 2019]
- Jeff Dean's Lecture for YC AI: https://blog.ycombinator.com/jeff-deans-lecture-for-yc-ai/
- Page 4, AlphaGo: https://intellipaat.com/blog/power-of-deep-learning-alphago-vs-lee-sedol-case-study/

A Timeline of AI

- 1943: First neuron, Walter Pitts, a logician, and Warren McCulloch, “A logical calculus of the ideas immanent in nervous activity”.
- 1945: ENIAC, in full Electronic Numerical Integrator and Computer, the first programmable general-purpose electronic digital computer.
- 1950: Alan Turing, “Computing Machinery and Intelligence”, Can machines think?
- 1952: **Machine Learning**, Arthur Lee Samuel, “Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress”, (1959)
- 1957: Frank Rosenblatt, Rosenblatt, a psychologist, submitted a paper entitled “The Perceptron: A Perceiving and Recognizing Automaton” to Cornell Aeronautical Laboratory in 1957.