Financial Data Science

Lecture 7
Neural Networks

Video tutorial:
https://www.youtube.com/watch?v=zKN9HOnAByQ

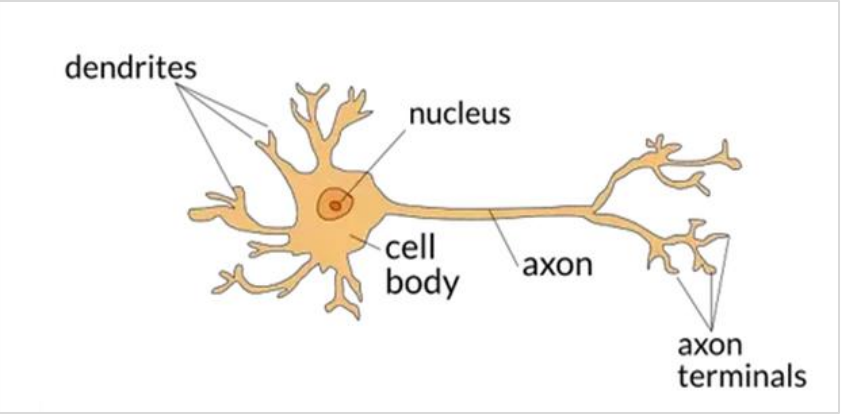
Liu Peng
liupeng@smu.edu.sg

What is Artificial Neural Network (ANN)?

- Algorithm that tries to **imitate the human brain**
- ANN's inception in 1943
  - proposed by neurophysiologist Warren McCulloch and mathematician Walter Pitts
  - inspired by the observation of **biological learning** systems
  - different from conventional computers that only receive commands
- History
  - The early successes until the 1960s led to the widespread belief that we would soon have truly intelligent machines
  - It became clear that this was not possible; hence funding went elsewhere and ANN's popularity **diminished in late 1990s**
  - Recent **resurgence** due to huge quantity of data available to train ANN and **tremendous increase in computing power**

Neuron in the brain

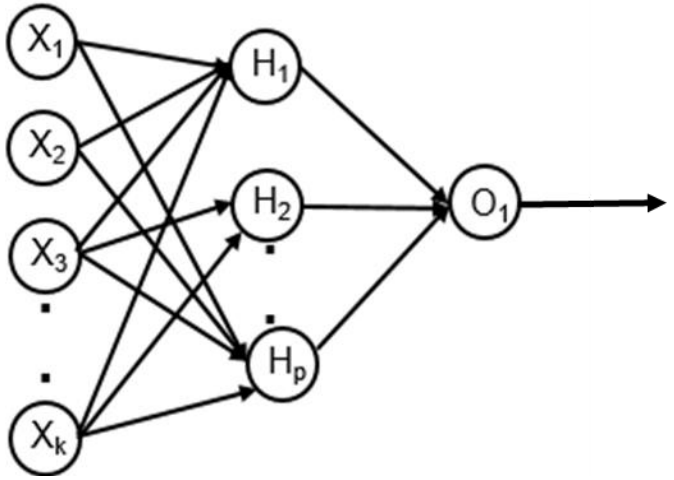
- Dendrite: **input** wire
- Axon: **output** wire (If this connects to muscle, muscle might contract)
- Human brain contains around 100 billion (10^{11}) neurons
- Brain's speed of operation (in milliseconds) is slower than digital computers (switch in nanoseconds)
- Why can brain perform complex tasks faster and more accurately than computer?
  - massive **parallelism**
  - 100 trillion (10^{14}) **interconnections** between 100 billion neurons



How does ANN mimic the brain?

- Neural network is based on a collection of connected units of nodes called ‘artificial neurons’

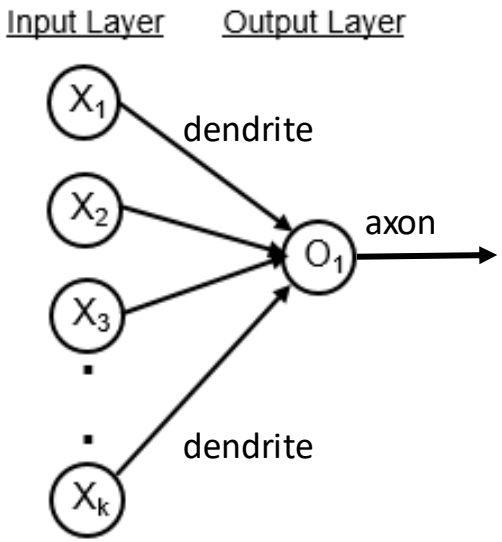
- Nodes loosely model the neurons in a biological brain, by forming highly **interconnected and weighted network structure**
  - nodes are arranged in **layers**
  - different layers are connected via **links**, and resemble the neurotransmissions in brain
  - All links have weights, indicating the importance of each incoming signal



ANN in different complexity level

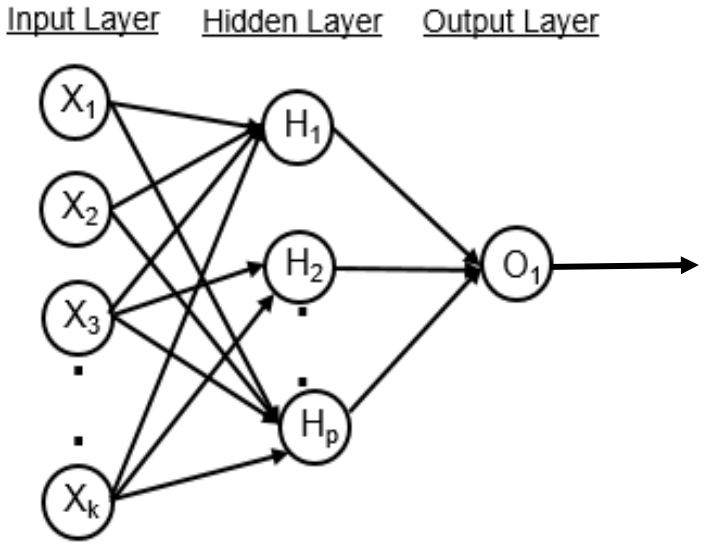
Single-layer

- One layer being the output layer
- O_1 acts as a neuron with dendrite and axon



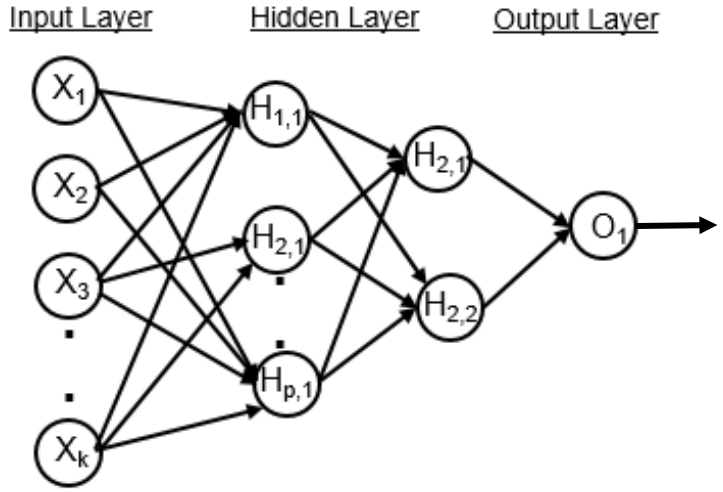
Two-layer

- ONE hidden layer consisting of H_i, autonomous computational units



Three-layer

- TWO hidden layers before the signals reach output layer

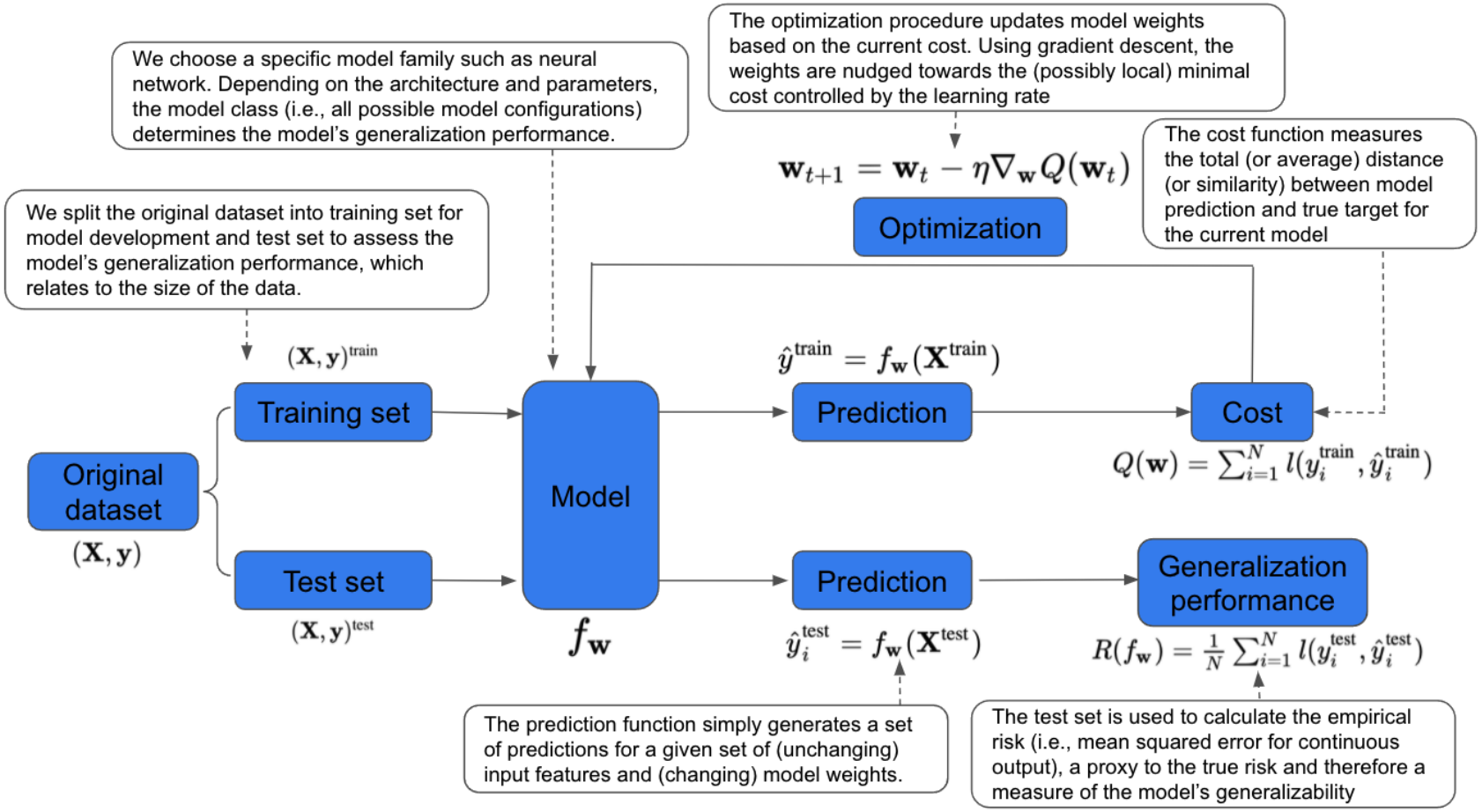


Input layer does **NOT** count as the number of layers in a network

Connection with linear regression

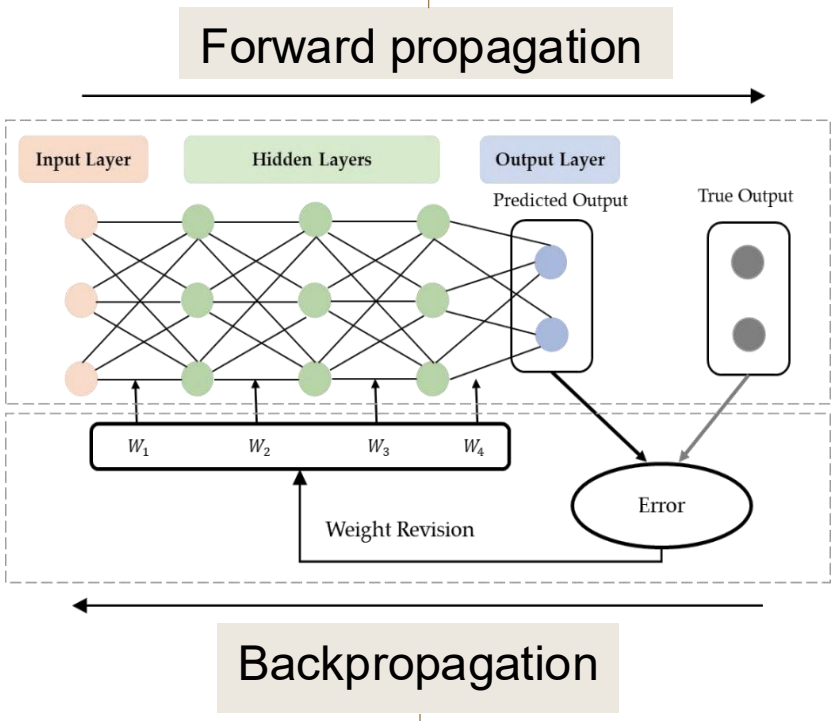
- A single-layer neural network without an activation function is mathematically equivalent to linear regression.
- In other words, linear regression can be thought of as the simplest type of neural network, one with only one layer (not counting the input layer) and no activation function.
- The weights in the neural network correspond to the regression coefficients in the linear regression model, and the task of learning the model is to find the best set of weights that minimize the difference between the predicted and actual values, just as in linear regression.

Revisiting the overall model training pipeline



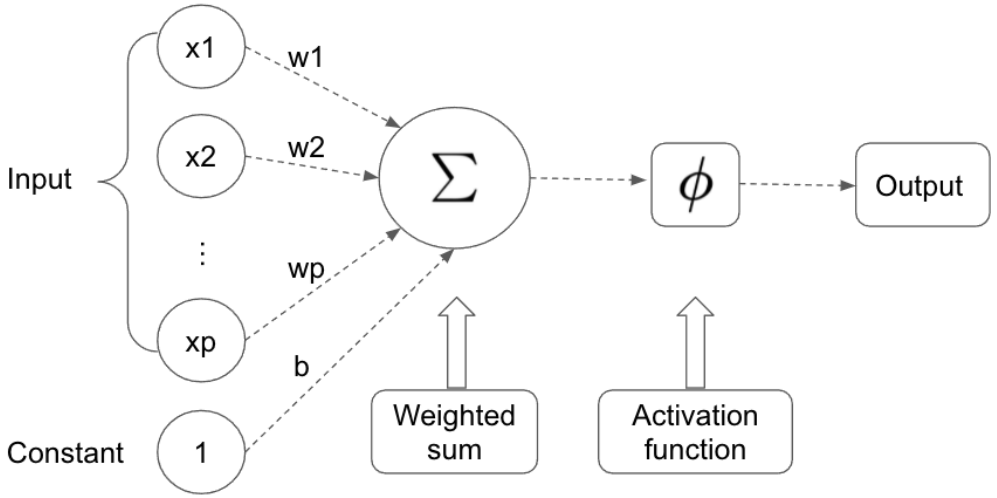
Workflow of ANN Algorithm

- Each input node receives the input variables' values from the data set and sends weighted signals to the hidden nodes
- Each hidden node and output node combines the received signals and pass it through an activation function to generate the output
- The output value of the output nodes are the predicted values



- Compute the prediction error at the output nodes
- Update the weights between the last hidden layer and output layer
- Compute the signal error at the hidden layer
- Update the weights between the last hidden layer and the second last hidden layer
- Similarly repeat for all the hidden layers until we update weights from input layer to first hidden layer

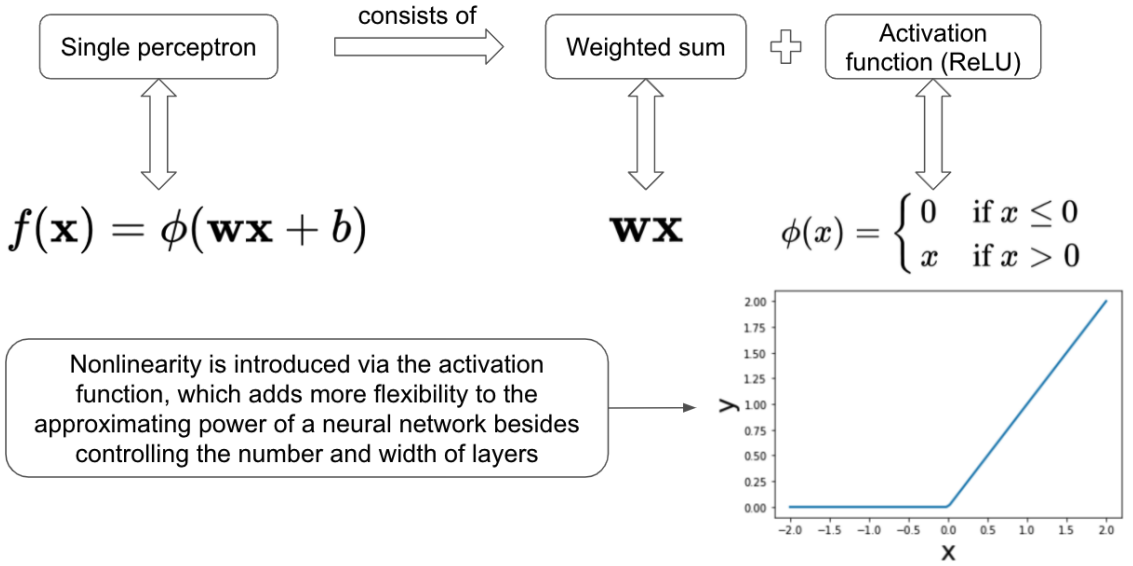
A single perceptron



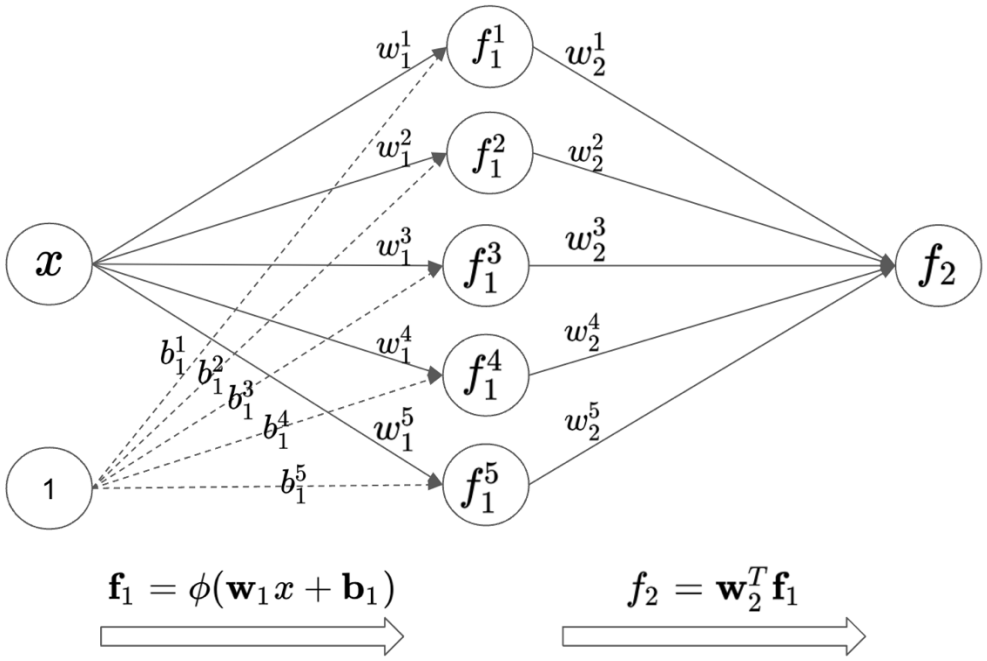
- The process flowchart of a perceptron, which consists of a weighted sum operation followed by an activation function. A column of ones is automatically added to correspond to the bias term in the weight vector.

Decomposing a perceptron

- Decomposing a single perceptron into a weighted sum and an activation function which is often ReLU. The ReLU operation passes through a signal if it is positive and mutes it if it is negative. Such nonlinearlity also introduces great approximating power to the neural networks in addition to the flexibility on designing the number and width of layers.



A two-layer neural network

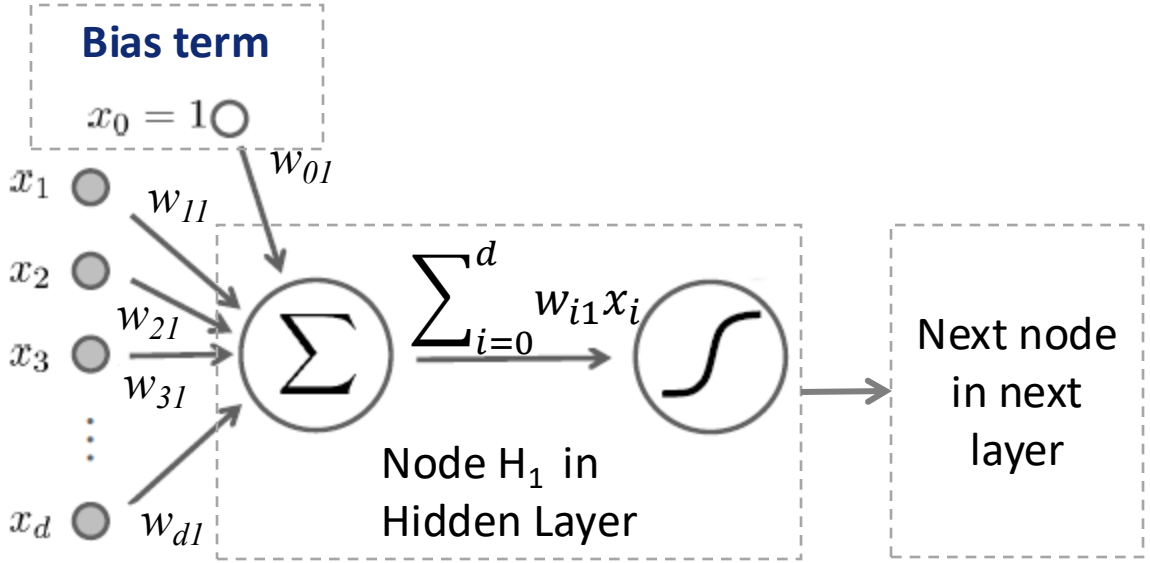


- Depicting the architecture of a two-layer network. The first hidden layer completes the weighted sum and nonlinear transformation operations, resulting in a 5x1 vector of intermediate features. The second hidden layer, i.e., the output layer, performs a weighted sum without the bias term, generating the final scalar output.
- The whole neural network could then be expressed via:

$$f(x) = \mathbf{w}_2^T \phi(\mathbf{w}_1 x + \mathbf{b}_1)$$
$$\mathbf{w}_1 = \{w_1^1, w_1^2, w_1^3, w_1^4, w_1^5\},$$
$$\mathbf{b}_1 = \{b_1^1, b_1^2, b_1^3, b_1^4, b_1^5\}$$
$$\mathbf{w}_2 = \{w_2^1, w_2^2, w_2^3, w_2^4, w_2^5\}$$

Which are all 5×1 weight vectors

Each node in hidden and output layer consists of summation and activation function



Why Bias term?

- Bias in linear equation: y=ax+b
- Without bias, the line always goes through (0,0) and depends on the slope only
- In ANN, bias shifts the model entirely to fit the data set better

Why non-linear activation function?

- It increases the capacity of model
- Without non-linearities, each extra layer is just one linear transform and neural network could be meaningless

Common activation functions

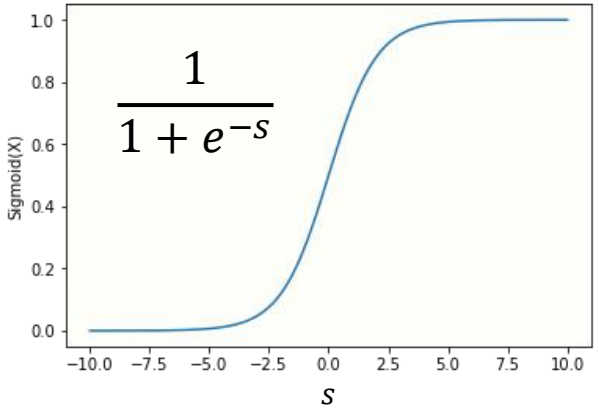
- Sigmoid (a.k.a. logistic)
- Tanh
- ReLU (Rectified Linear Unit)

Non-linear activation functions and loss functions

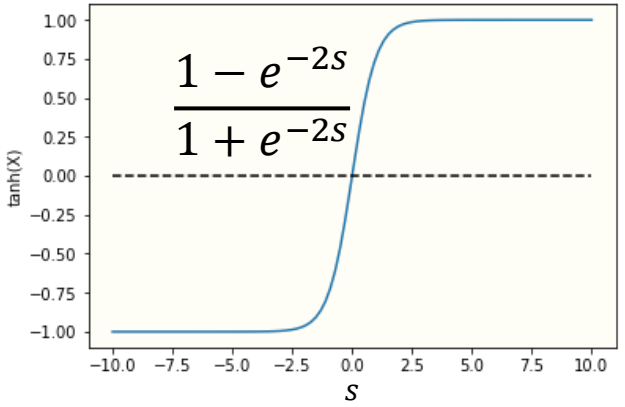
$$s = \sum w_{ij} x_i \text{ or } \sum w_{jm} H_j$$

Activation function

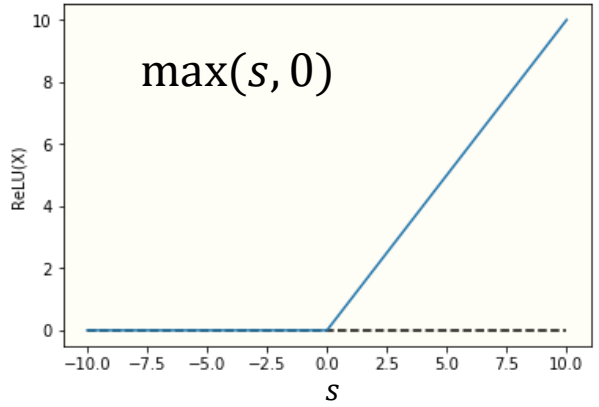
Sigmoid (a.k.a. logistic)



Tanh



ReLU (Rectified Linear Unit)



How to select activation function: Low loss; Fast convergence

Loss function

Regression loss:

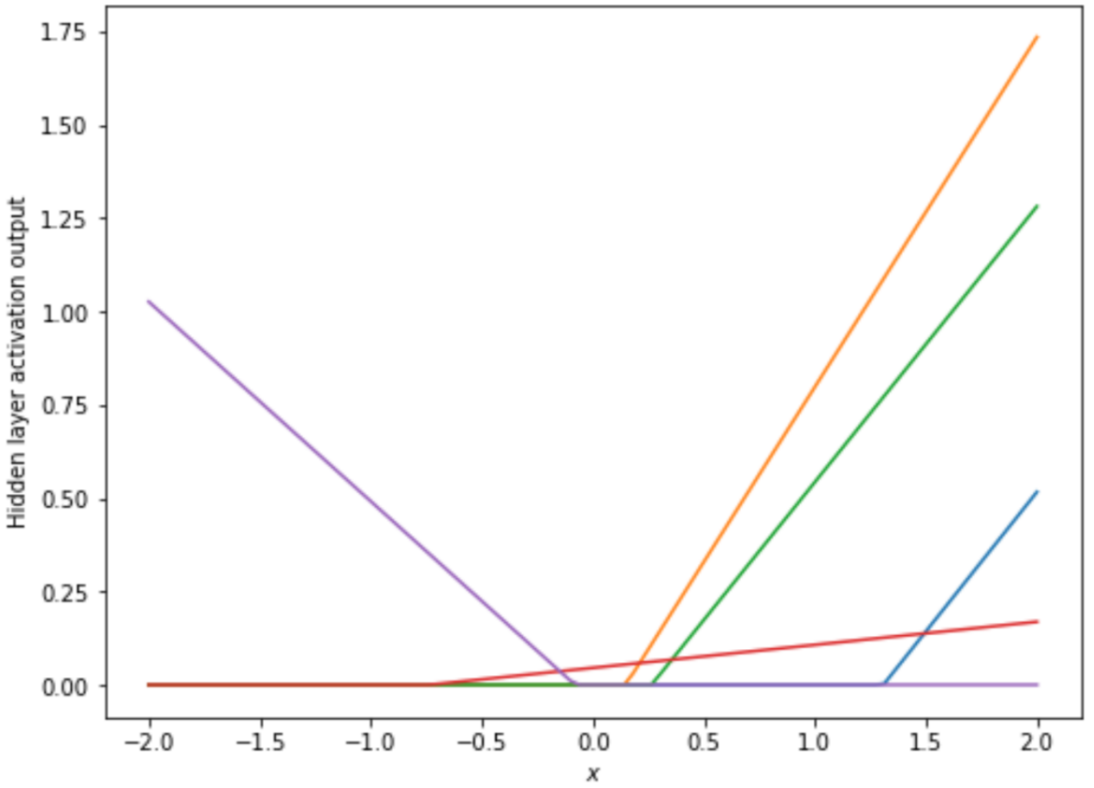
$$J = \frac{1}{2}\sum_{i=1}^{d}(y_i - \hat{y}_i)^2$$

Binary classification loss:

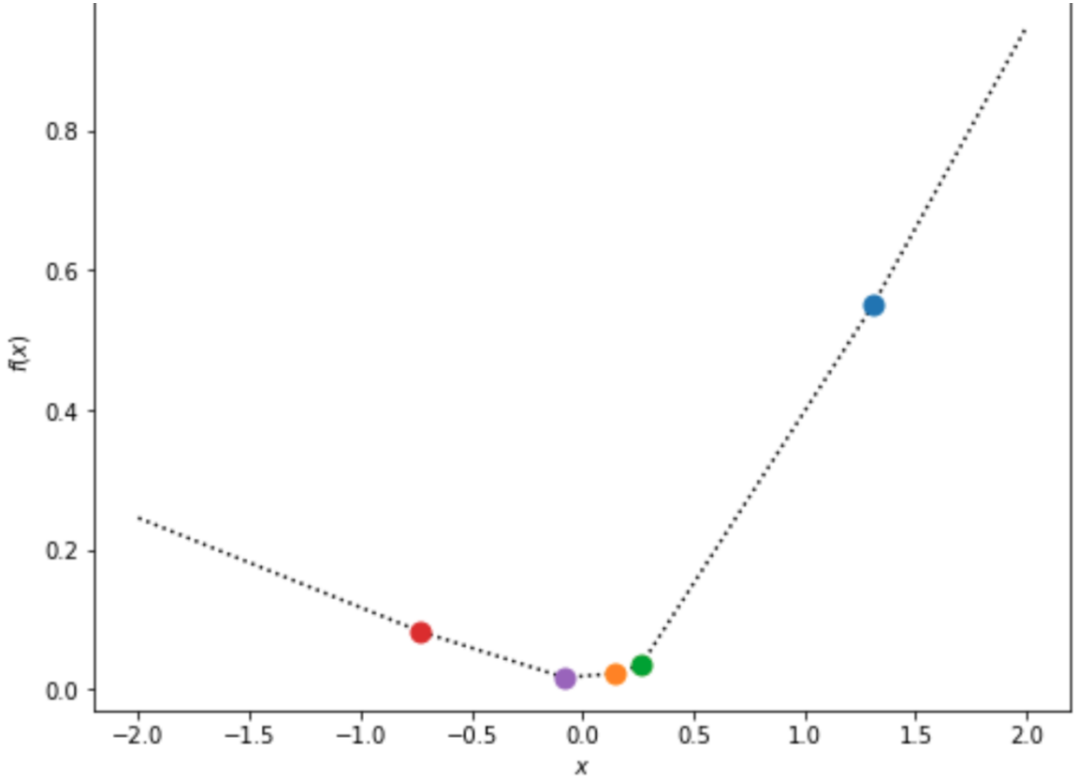
$$J = -\sum_{i=1}^{d} y_i \ln(\hat{y}_i) + (1 - y_i)\ln(1 - \hat{y}_i)$$

Visualizing the activation of individual neurons

- Visualizing the activation output of the five neurons in the hidden layer. Each activation is a ReLU function with similar shape, differing in the location of the turning point, the magnitude and the direction of the slope.
- As optimization proceeds, these features will jointly form a better representation of the original input data, each presenting a different perspective at various levels of granularity. They can also be interpreted as a set of derived basis functions.



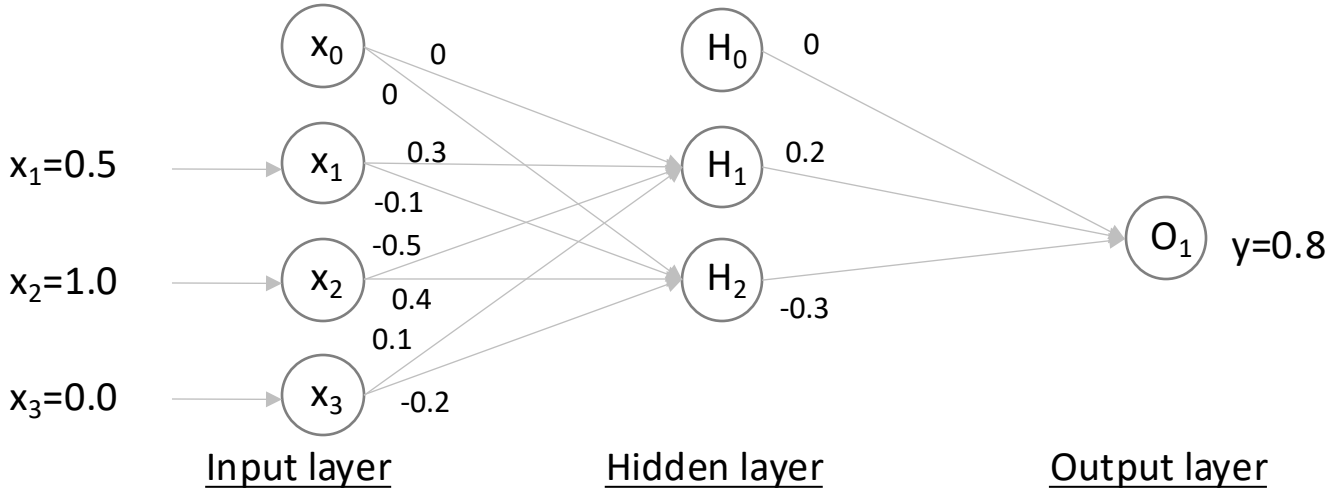
Visualizing the final output of the two-layer neural network



- Visualizing the final layer's output. The fitting model is piecewise linear, with turning points reflected as the intersection between two ReLU activations.
- To have sufficient turning points (i.e., breakpoints) to approximate an elaborate function, a neural network model needs to be wide enough in the hidden layer (i.e., having enough hidden nodes). However, doing so would be computationally inefficient. The computational cost of having more nodes will likely outweigh the benefit of the added modeling flexibility.
- Instead of making the shallow neural network wider, the mainstream approach to improve the representational power of a neural network is to make it deeper by stacking more layers. This is why it is called the deep neural network. In general, a neural network with more than two hidden layers can be called deep, although the depth can grow much more for large models. Building a deep neural network with multiple layers is more efficient in increasing its approximating power than widening a shallow one.

Forward propagation – Starting setup

- **Two-layer ANN** with 3 input nodes, 2 hidden nodes, and 1 output node
- Regression model, i.e., y is continuous, hence $J = \frac{1}{2}\sum_{i=1}^{d}(y_i - \hat{y}_i)^2$
- *d*=3, *p*=2, *c*=1
- All the weights w_{ij} *and* w_{jm} are randomly initialized, within [-0.5, 0.5]
- Assume the randomly selected first observation is x_1=0.5, x_2=1.0, x_3=0.0; y=0.8
- Assume bias x_0 and H_0 are selected as 1



Forward propagation – calculation with sigmoid activation function

- Combined signal received at H_j *from* x_i

$$\sum_{i=0}^{d} w_{ij} x_i$$

$$\sum_{i=0}^{3} w_{i1} x_i = 0 \times 1 + 0.3 \times 0.5 - 0.5 \times 1 + 0.1 \times 0 = -0.35$$ For node H_1

$$\sum_{i=0}^{3} w_{i2} x_i = 0 \times 1 - 0.1 \times 0.5 + 0.4 \times 1 - 0.2 \times 0 = 0.35$$ For node H_2

- Processed signal at H_j (using sigmoid activation function)

$$\frac{1}{1 + e^{-\sum_{i=0}^{d} w_{ij} x_i}}$$

$$H_1 = \frac{1}{1 + e^{-(-0.35)}} = 0.4133$$

$$H_2 = \frac{1}{1 + e^{-0.35}} = 0.5866$$

- Combined processed signal transmitted from H_j and received at O_m

$$\sum_{j=0}^{p} w_{jm} H_j$$

For node O_1

$$\sum_{j=0}^{2} v_{j1} H_i = 0 \times 1 + 0.2 \times 0.4133 + 0.3 \times 0.5866 \approx -0.0933$$

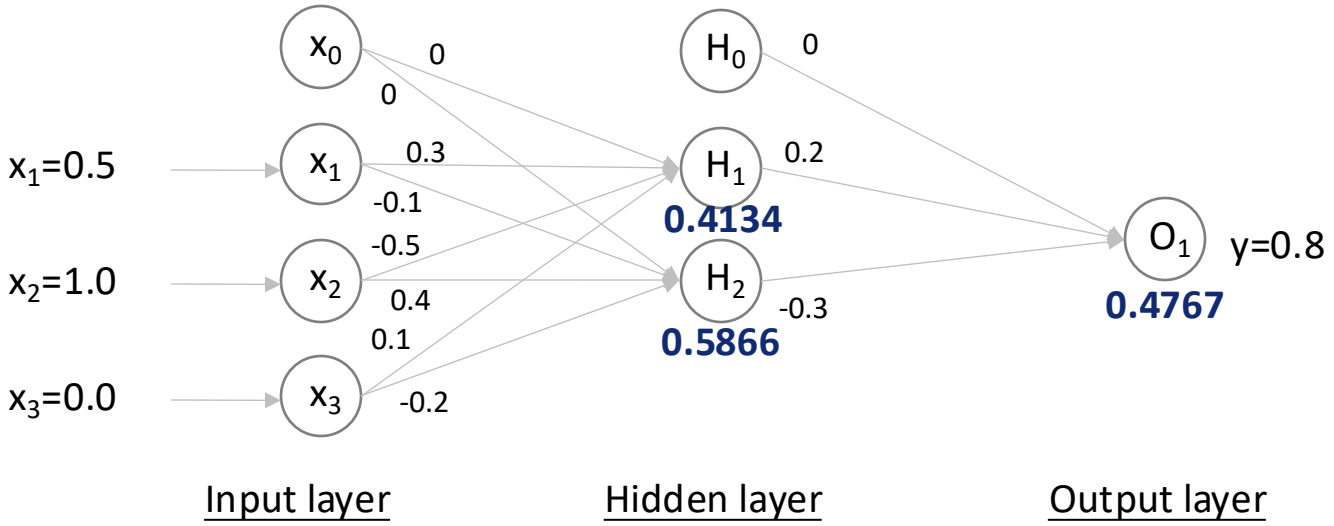
- The final output transmitted from O_m

$$\frac{1}{1 + e^{-\sum_{j=0}^{p} w_{jm} H_j}}$$

$$O_1 = \frac{1}{1 + e^{-(-0.0933)}} = 0.4767$$

Prediction $\hat{y} = 0.4767$

Forward propagation – result



Incur a loss J due to prediction error (squared difference between 0.4767 and 0.8) must be propagated from the output layer all the way back to train all the weights

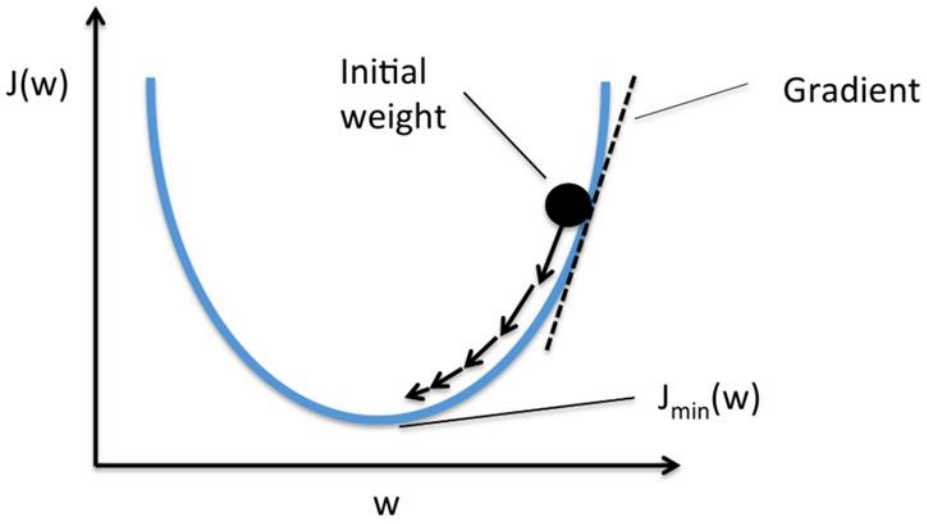
Optimization using gradient descent

For each observation

For each layer l

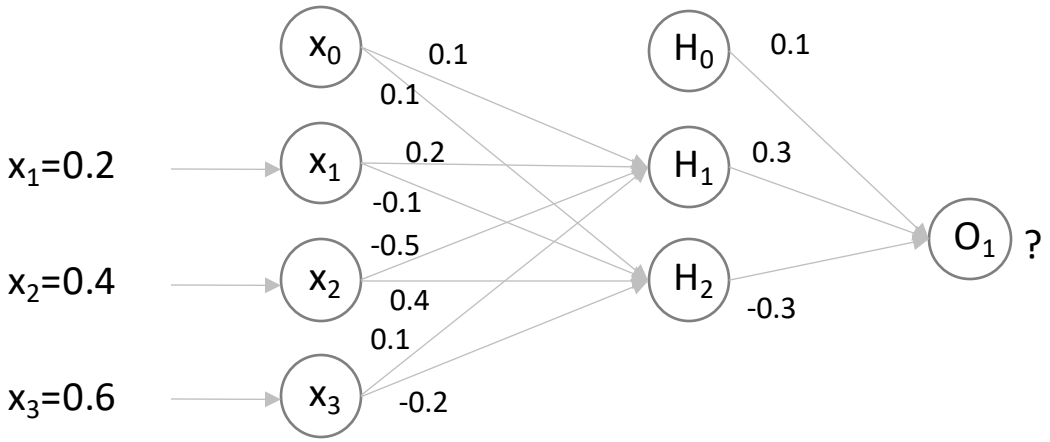
$$w^l \leftarrow w^l - \eta \frac{\partial J}{\partial w^l}$$

$\frac{\partial J}{\partial w^l}$ is the gradient



- Gradient descent algorithm
  - η(eta) is the learning rate
  - Old weight minus a portion of the gradient brings us closer to the min of loss function J
  - If learning rate is too big?
  - If learning rate is too small?
- Mathematical derivations of how to calculate $\frac{\partial J}{\partial w^l}$ for student's own exploration

Use this trained ANN to predict target for an unseen observation



$$\sum_{i=0}^{3} w_{i1}x_i = 0.1 \times 1 + 0.2 \times 0.2 - 0.5 \times 0.4 + 0.1 \times 0.6 = 0$$

$$\sum_{i=0}^{3} w_{i2}x_i = 0.1 \times 1 - 0.1 \times 0.2 + 0.4 \times 0.4 - 0.2 \times 0.6 = 0.12$$

$$H_1 = \frac{1}{1 + e^{-(0)}} = 0.5$$

$$H_2 = \frac{1}{1 + e^{-0.12}} \approx 0.53$$

$$\sum_{j=0}^{2} v_{j1}H_i = 0.1 \times 1 + 0.3 \times 0.5 - 0.3 \times 0.53 \approx 0.091$$

$$O_1 = \frac{1}{1+e^{-(0.091)}} \approx 0.5227$$

Prediction $\hat{y} = 0.5227$

Create ANN for Regression problem

MLPRegressor class from sklearn library

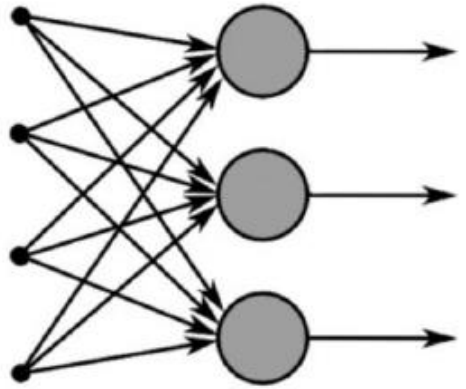
- Experiment hidden_layer_sizes to create different structure for the network
- Use appropriate learning rate

Sequential class from tensorflow library

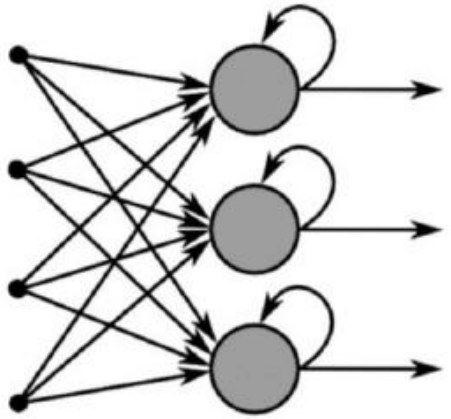
- Create desired network structure by adding layers
- Understand how batch_size and epochs work

Recurrent Neural Networks (RNN)

- A family of neural networks typically for processing sequential data e.g., time series, text, audio
- RNN looks like a normal neural network, except that nodes also have connections pointing back at themselves
- One common sequence model is long short-term memory (LSTM)

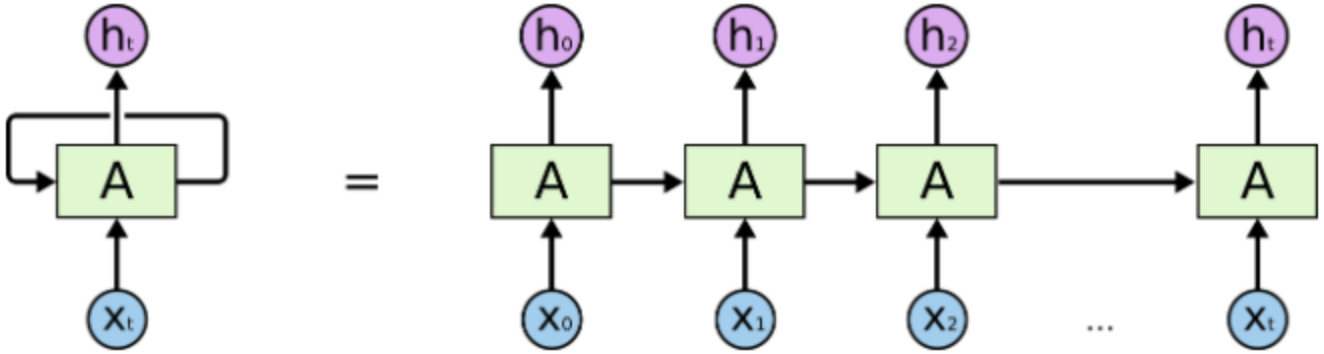


Normal neural network



Recurrent neural network

The simplest possible RNN



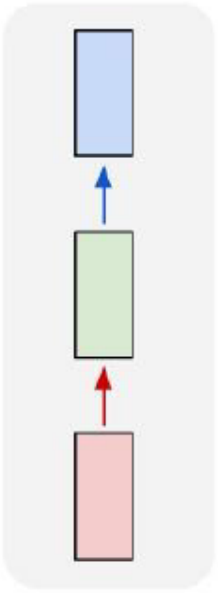
One neuron

- receiving inputs
- producing an output
- sending that output back to itself

- The neuron receives both the input x_t and the hidden output from the previous time step h_{t-1}
- Notice that h_t is a function of x_t and h_{t-1}, and h_{t-1} is a function of x_{t-1} and h_{t-2}, so on so forth
- h_t is therefore a function of all the inputs since t = 0

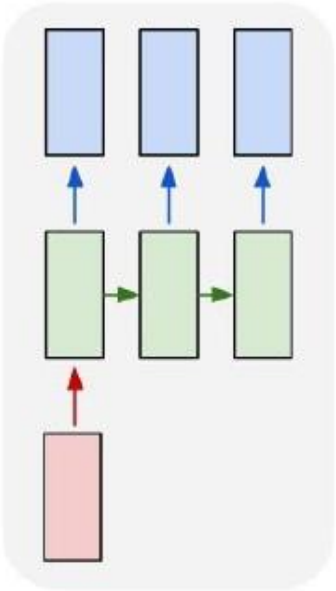
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Different kind of ANN (1/2)



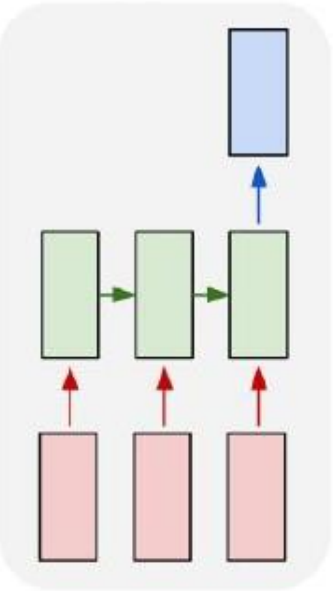
Simplest ANN

- 1 neuron
- 1 input
- 1 output



One to many, e.g.

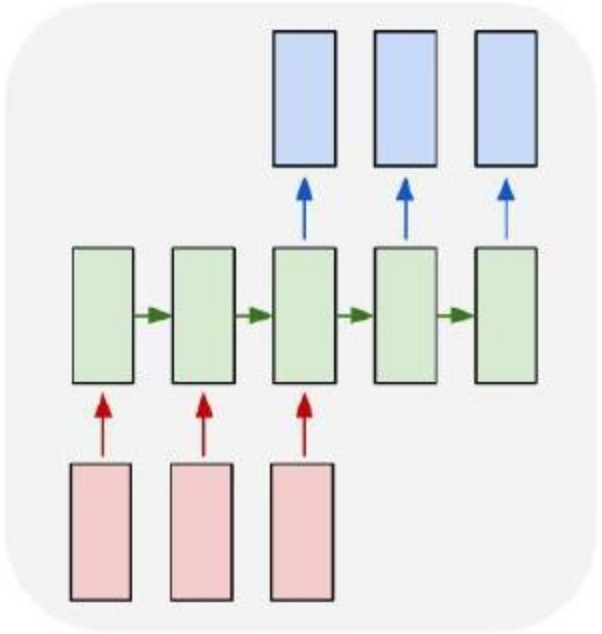
- Input: Image
- Output: Caption



Many to one, e.g.

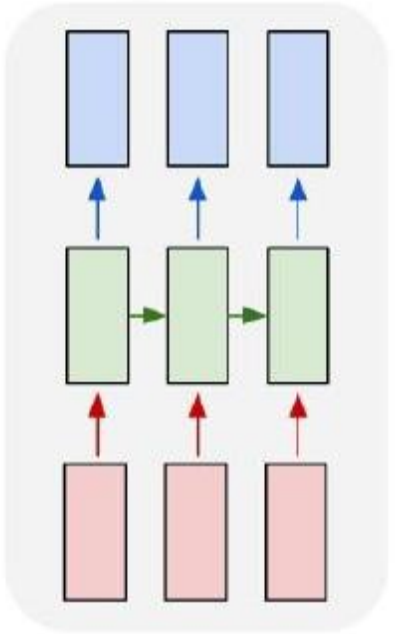
- Input: Words in an email
- Output: Spam or not

Different kind of ANN (2/2)



Many to many with time shift, e.g.

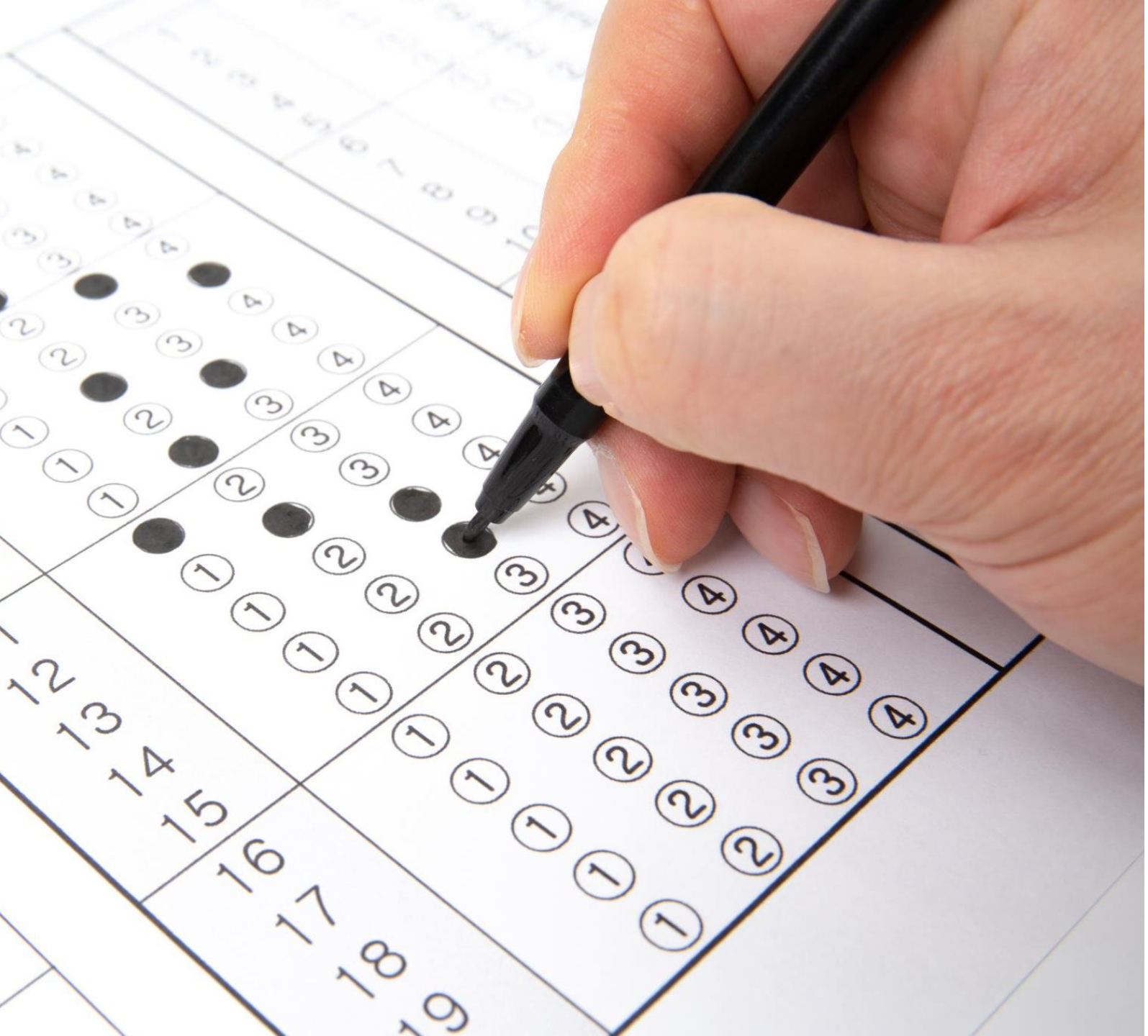
- Input: French sentence
- Output: English sentence



Many to many with no time shift, e.g.

- Input: audio of words pronunciation
- Output: words spelled out


Coding session



In-class quiz