Machine Learning and Financial Applications

Lecture 6
Deep neural networks

Liu Peng
liupeng@smu.edu.sg

Video tutorial

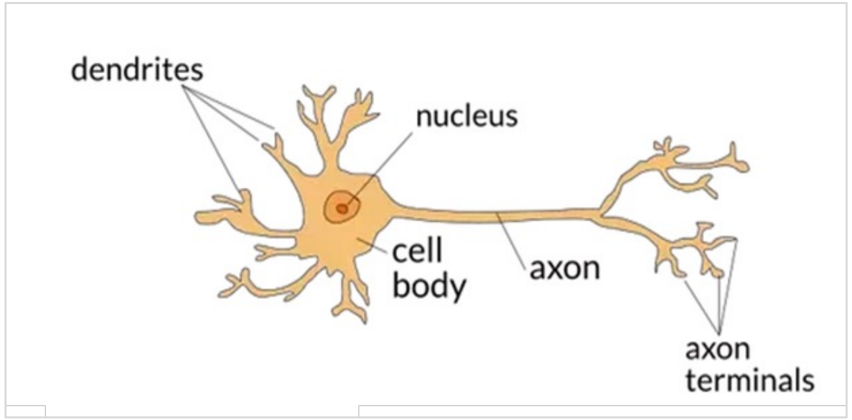
- https://youtu.be/zKN9HOnAByQ

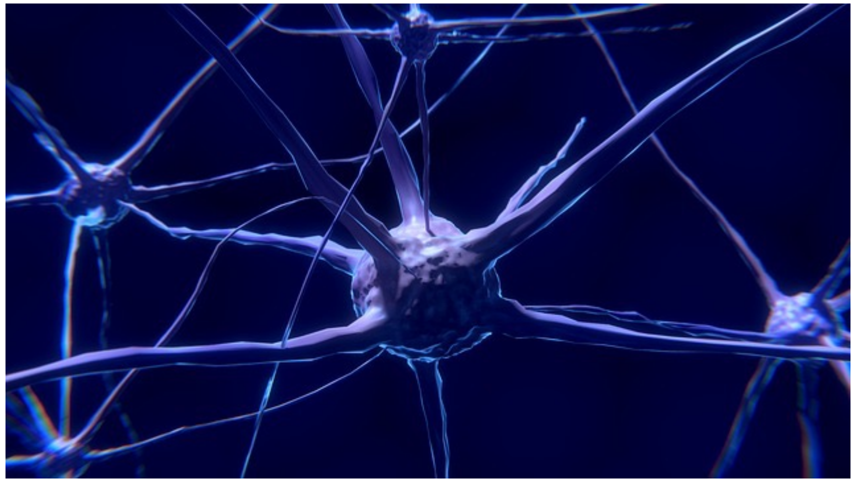
What is Artificial Neural Network (ANN)?

- 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

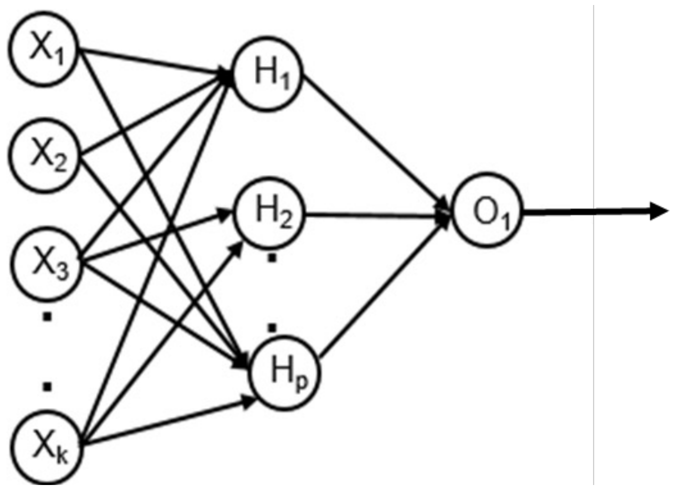
- Dendrite: **input** wire
- Axon: **output** wire (If this connects to muscle, muscle might contract)
- Human brain contains around 100 billion (10^{11}) neurons
- Brain's speed of operation (in milliseconds) is slower than digital computers (switch in nanoseconds)
- Why can brain perform complex tasks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





How does ANN mimic the brain?

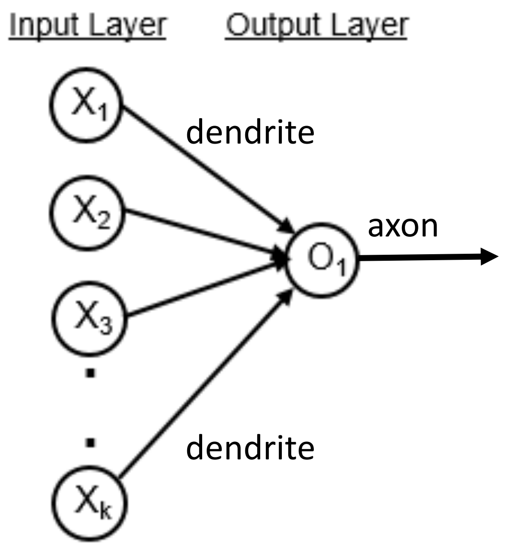
- Neural network is based on a collection of connected units of nodes called ‘artificial neurons’
- Nodes loosely model the neurons in a biological brain, by forming highly **interconnected and weighted network structure**
  - nodes are arranged in **layers**
  - different layers are connected via **links**, and resemble the neurotransmissions in brain
  - All links have weights, indicating the importance of each incoming signal



ANN in different complexity level

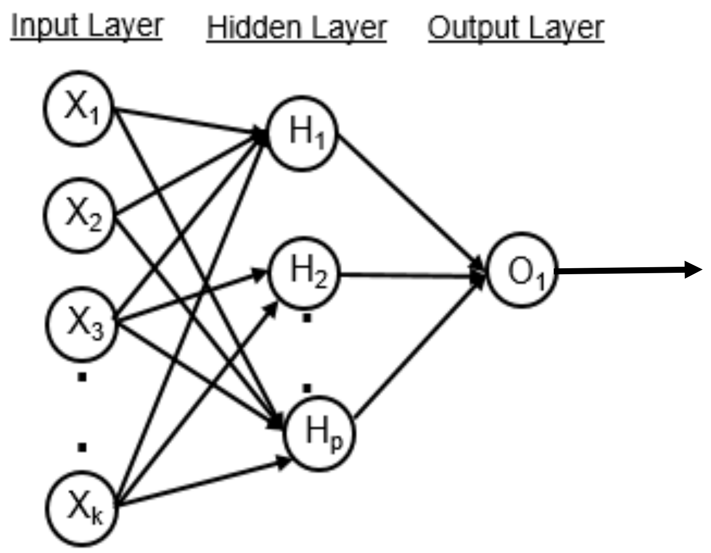
Single-layer

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



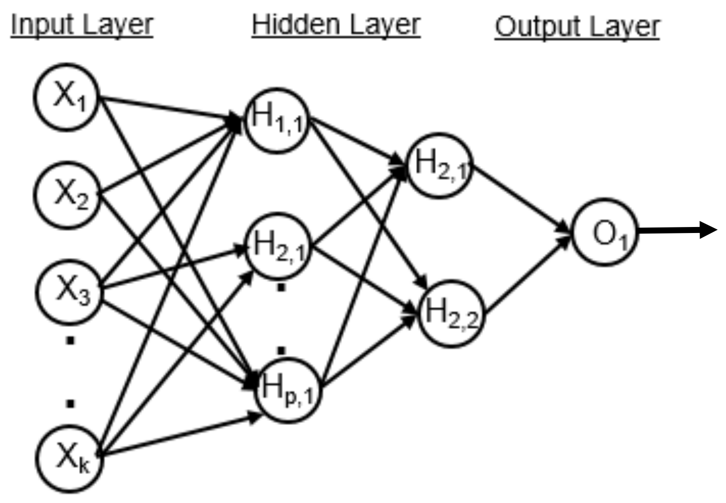
Two-layer

- 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.

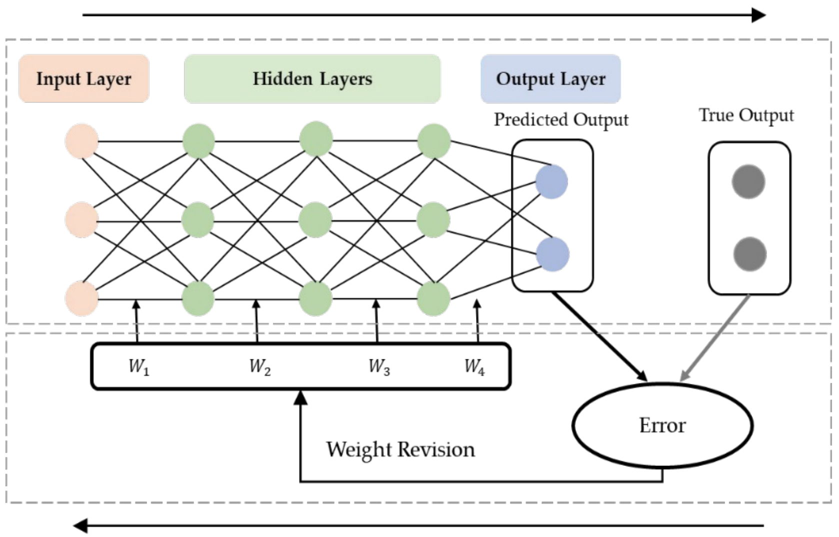
- In other words, linear regression can be thought of as the simplest type of neural network, one with only one layer (not counting the input layer) and no activation function.

- 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.

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

Forward propagation



Backpropagation

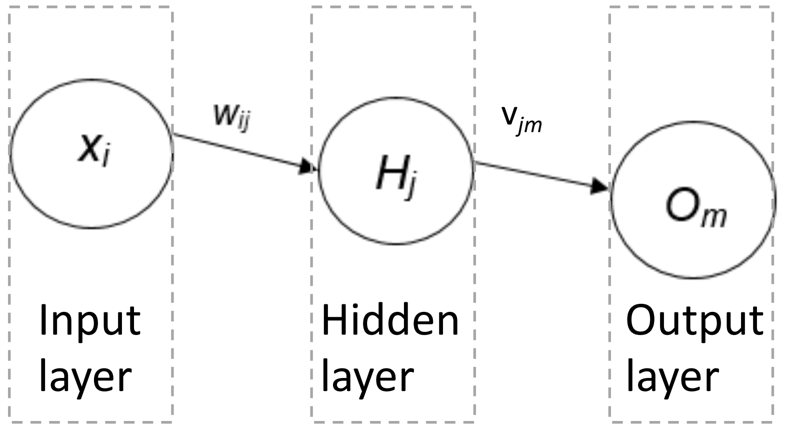
- Compute the prediction error at the output nodes
- Update the weights between the last hidden layer and output layer
- Compute the signal error at the hidden layer
- Update the weights between the last hidden layer and the second last hidden layer
- Similarly repeat for all the hidden layers until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

More on data set for the ANN model

How is the training data set fed into the process?

- Randomly initialize all weights
- Select an observation
- Forward propagation to calculate the output signals
- Backpropagation to revise all weights
- Repeat until all observations are fed into the ANN

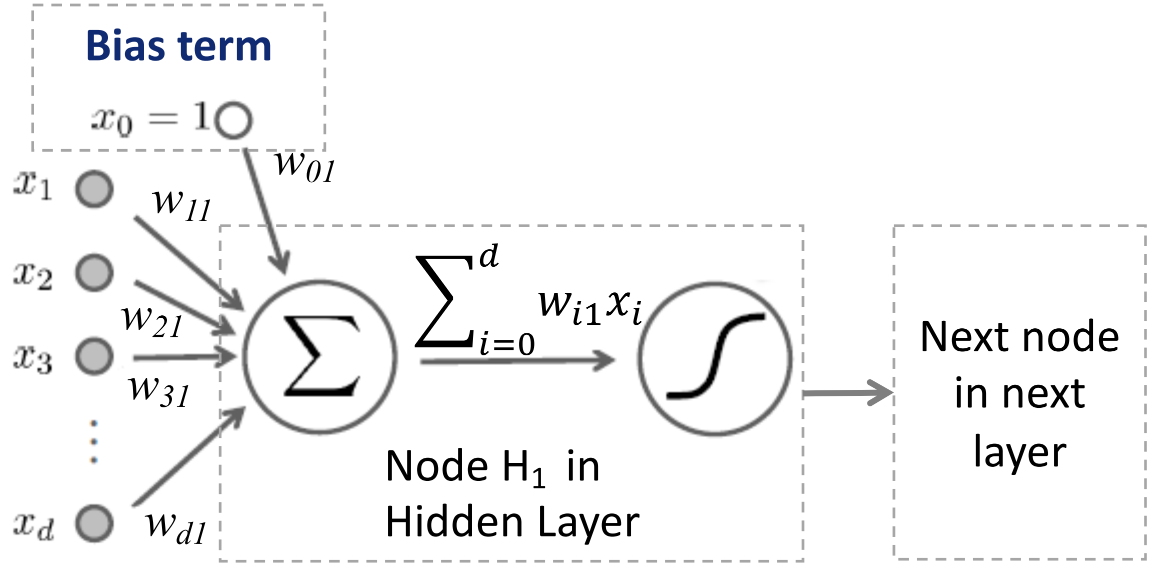
Notations for a TWO-layer ANN



- x_i : *i-th* input variable node, $i = 1, 2, \ldots, d$
- H_j : *j-th* hidden node, $j = 1, 2, \ldots, p$
- O_m : *m-th* output variable node, $m = 1, 2, \ldots, c$
- w_{ij} : weight between x_i and H_j
- v_{jm} : weight between H_j and O_m

- Each node is an autonomous computational unit
- Each layer has a unique **activation function**
- Each connection has a **weight** indicating the **importance** of each input signal

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

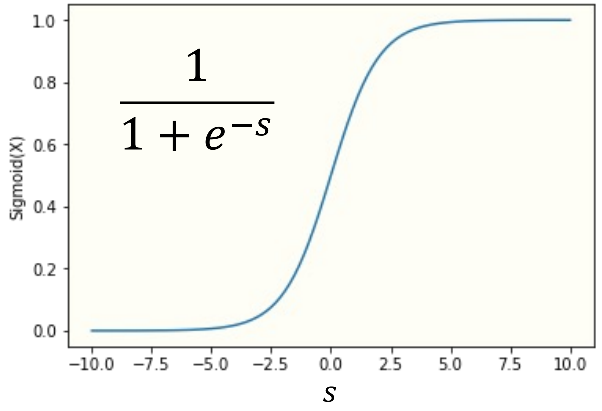
- Sigmoid (a.k.a. logistic)
- Tanh
- ReLU (Rectified Linear Unit)

Non-linear activation functions and loss functions

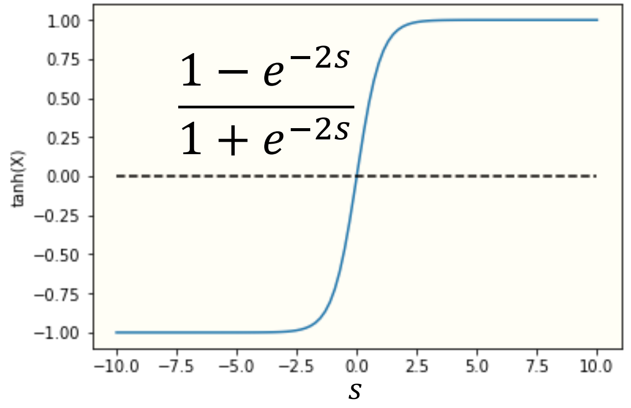
$$s = \sum w_{ij} x_i \text{ or } \sum v_{jm} H_j$$

Activation function

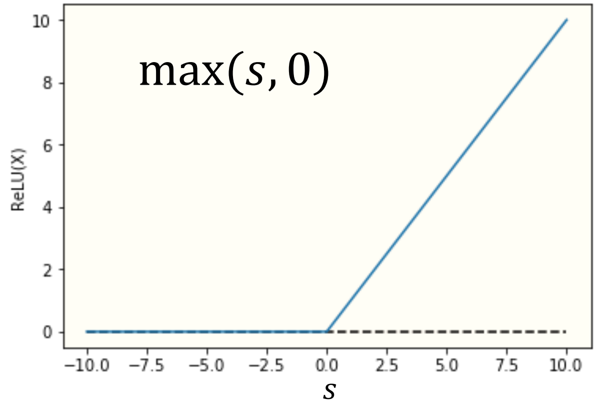
Sigmoid (a.k.a. logistic)



Tanh



ReLU (Rectified Linear Unit)



How to select activation function: Low loss; Fast convergence

Loss function

Regression loss:

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

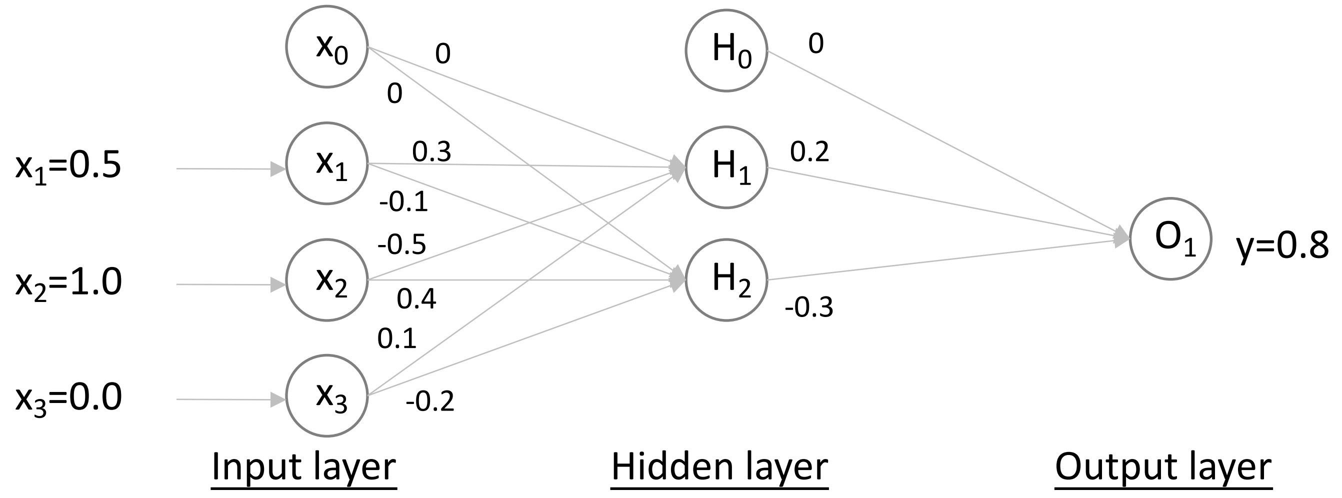
Binary classification loss:

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

Exercise: Create ANN for classification problem

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* v_{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

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

$$\sum_{i=0}^{3} w_{i1}x_i = 0\times1 + 0.3\times0.5 - 0.5\times1 + 0.1\times0 = -0.35$$

$$\sum_{i=0}^{3} w_{i2}x_i = 0\times1 - 0.1\times0.5 + 0.4\times1 - 0.2\times0 = 0.35$$

- 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

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

$$\sum_{j=0}^{2} v_{j1}H_i = 0\times1 + 0.2\times0.4133 + 0.3\times0.5866 \approx -0.0933$$

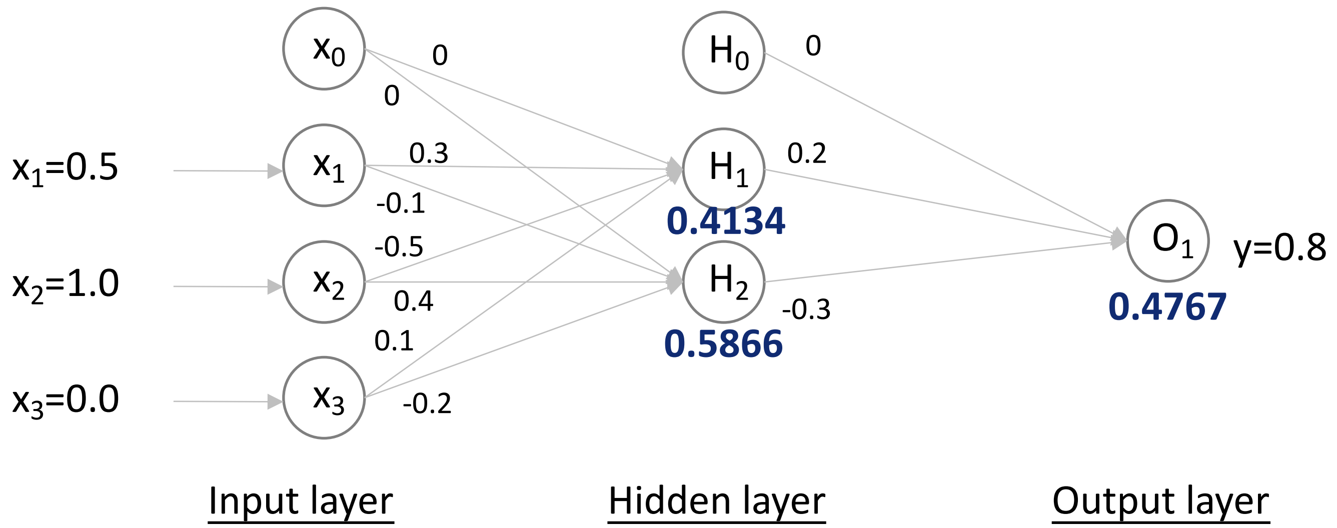
- The final output transmitted from O_m

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

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

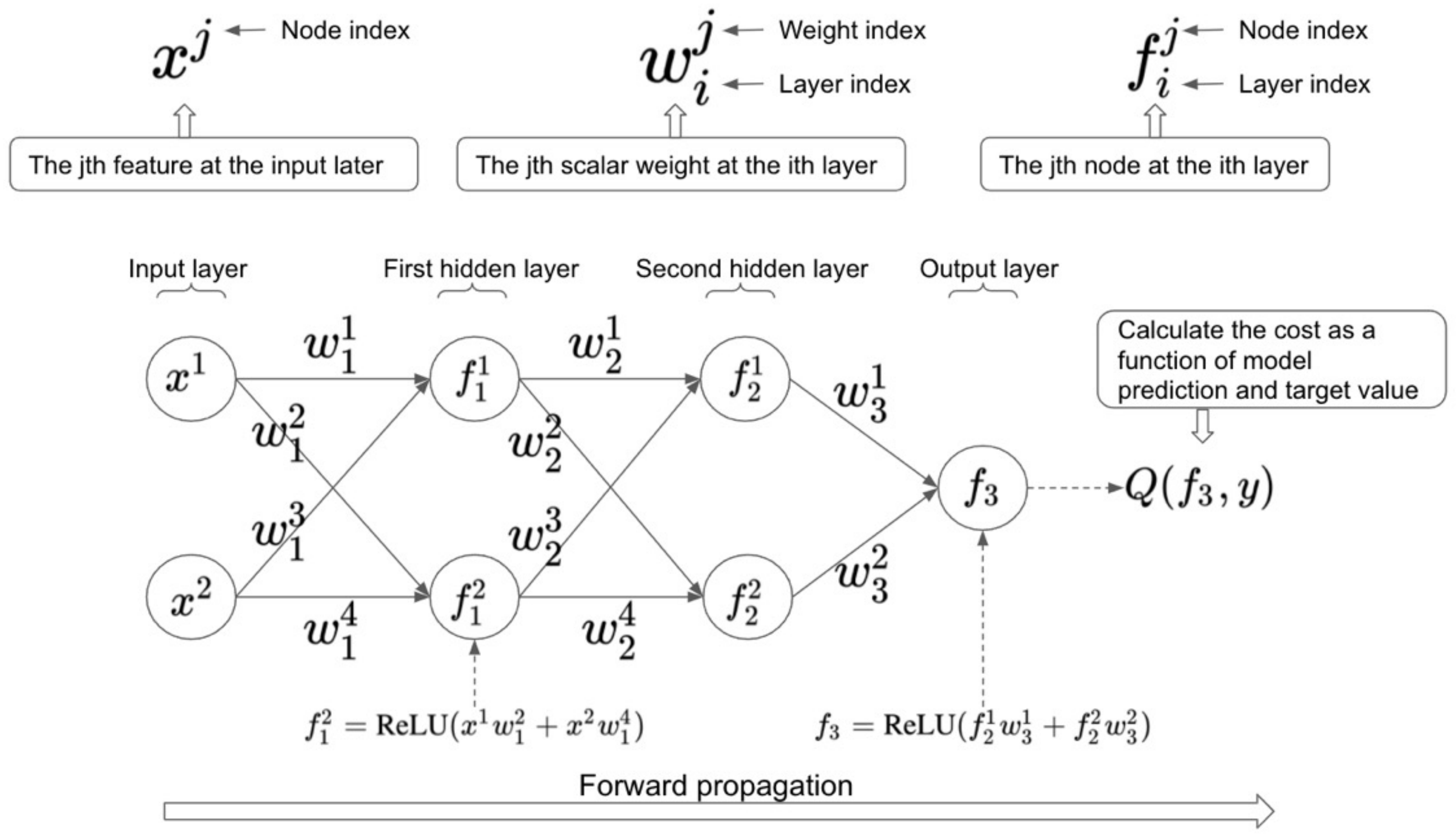
Prediction $\hat{y} = 0.4767$

Forward propagation – result



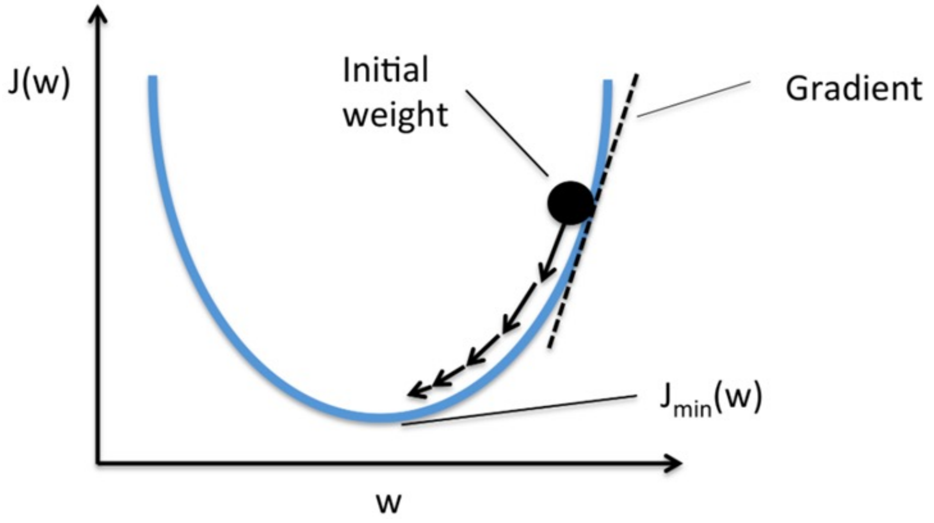
Loss J due to prediction error (difference between 0.4767 and 0.8)
must be propagated from the output layer all the way back to train all the weights

Forward propagation



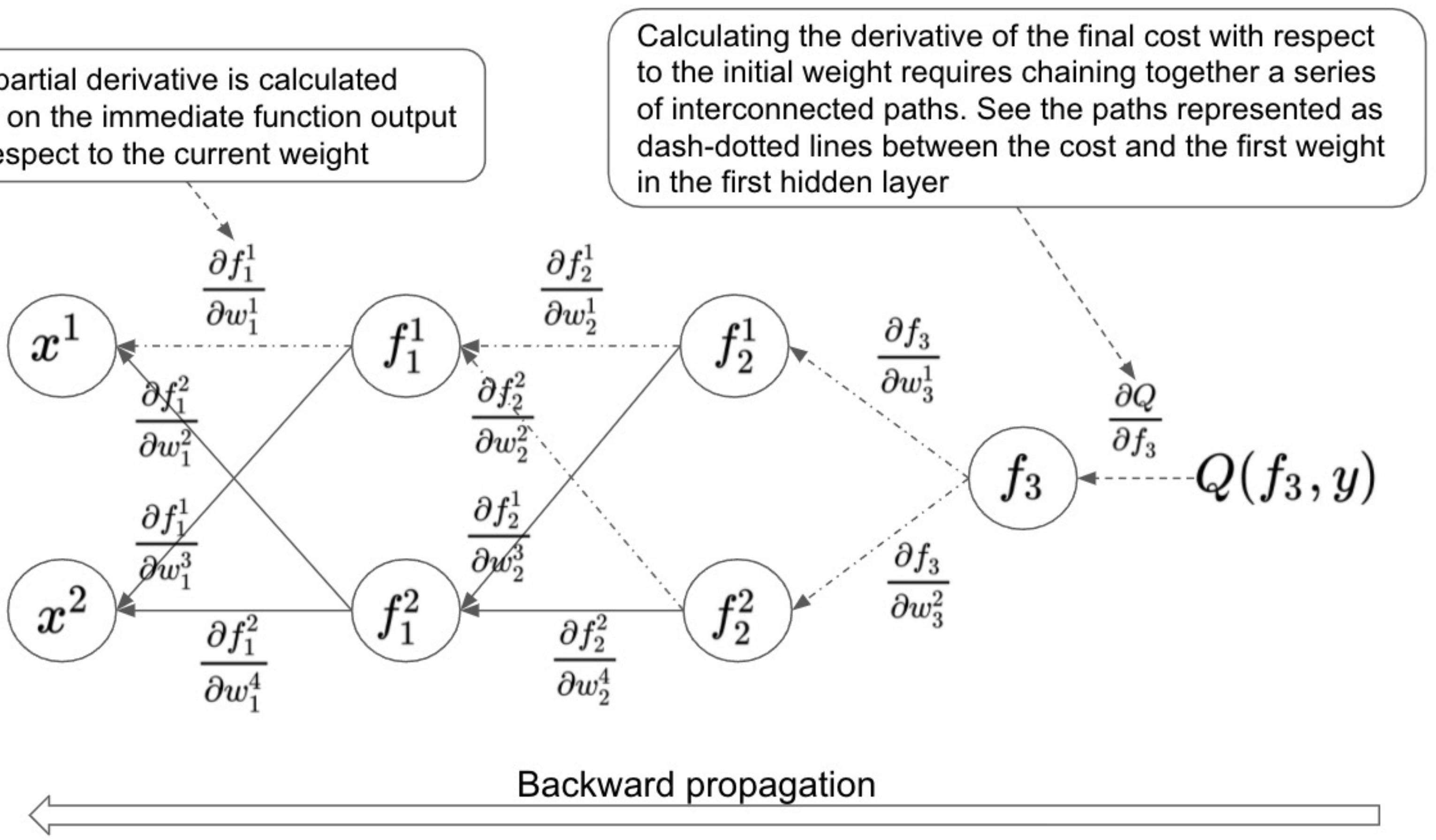
Gradient descent

```
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
```

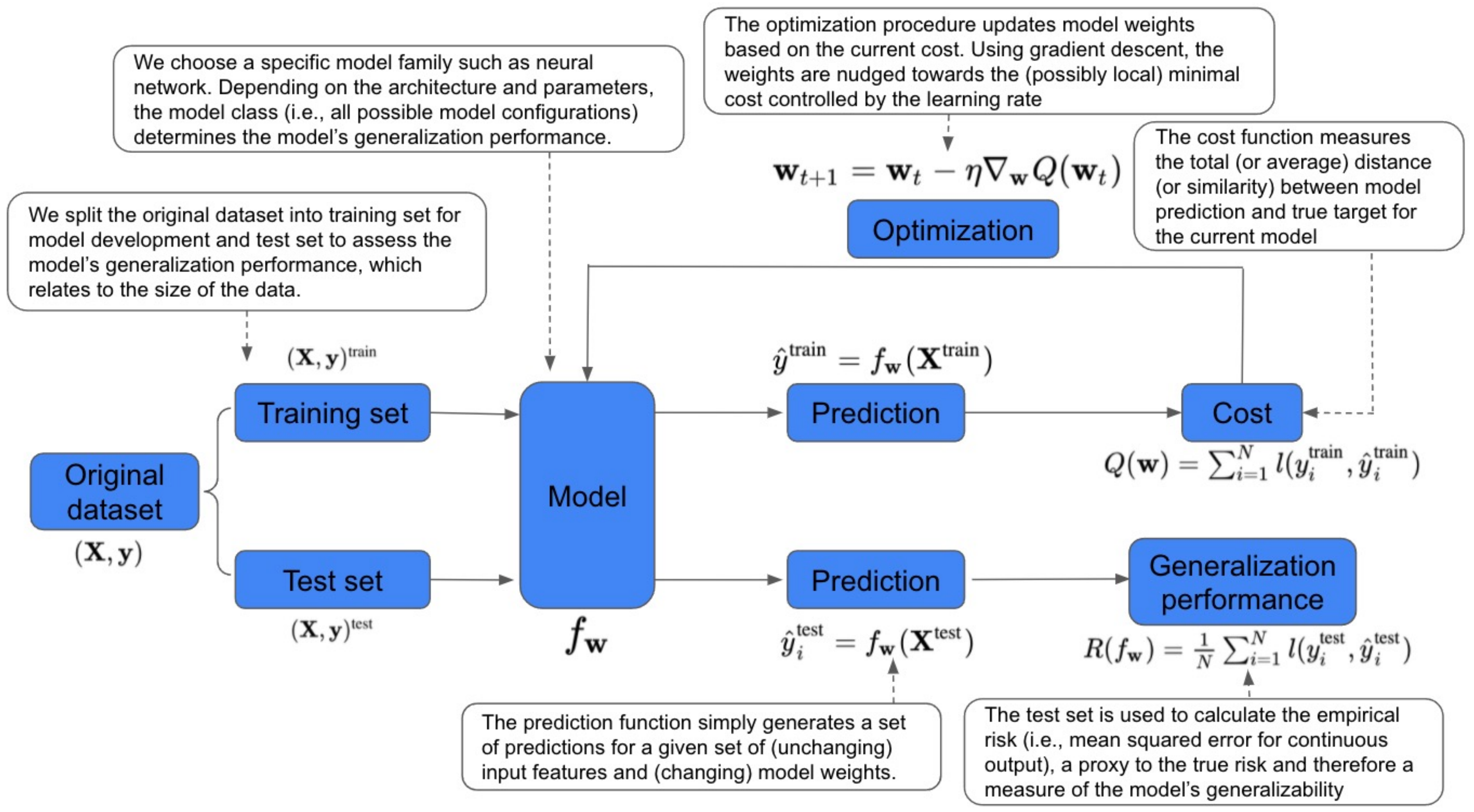


- Gradient descent algorithm
  - η(eta) is the learning rate
  - Old weight minus a portion of the gradient brings us closer to the min of loss function J
  - If learning rate is too big?
  - If learning rate is too small?
- Mathematical derivations of how to calculate $\frac{\partial J}{\partial w^l}$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

Backward propagation

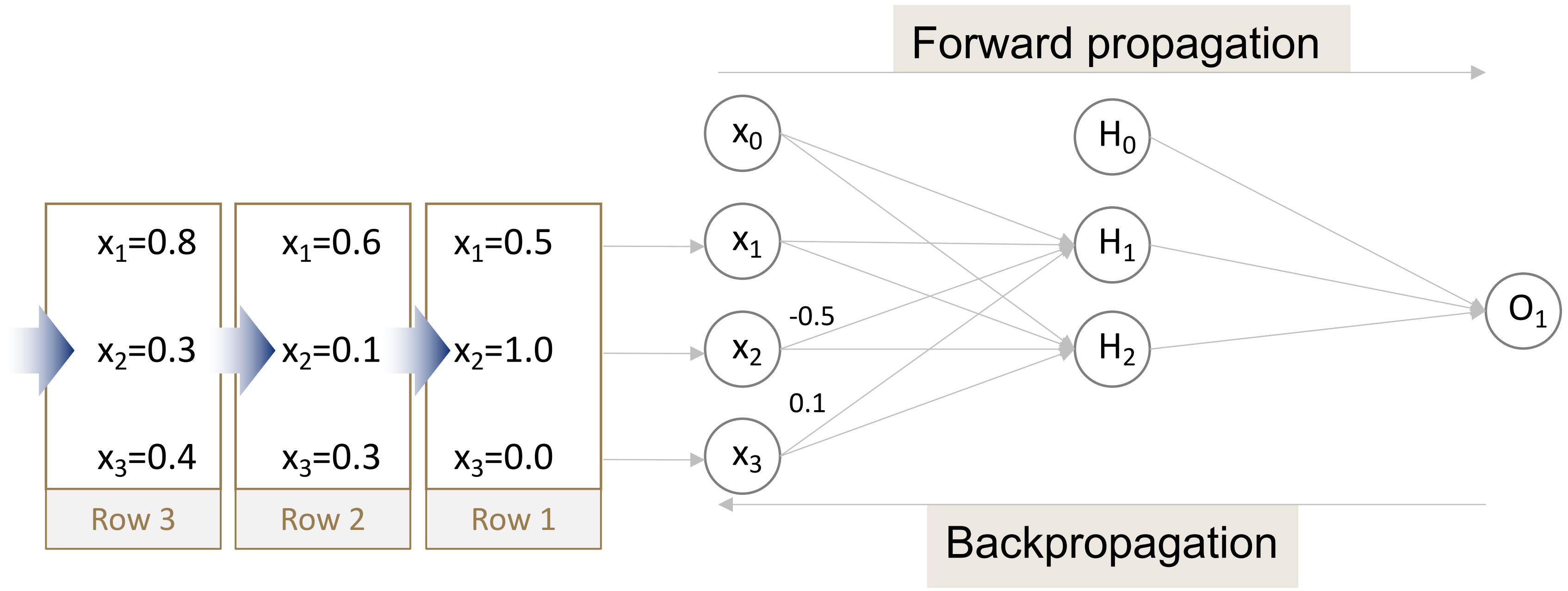

Each partial derivative is calculated based on the immediate function output with respect to the current weight
Calculating the derivative of the final cost with respect to the initial weight requires chaining together a series of interconnected paths. See the paths represented as dash-dotted lines between the cost and the first weight in the first hidden layer
x^1
x^2
f_1^1
f_1^2
f_2^1
f_2^2
f_3
$Q(f_3, y)$
$\frac{\partial f_1^1}{\partial w_1^1}$
$\frac{\partial f_1^2}{\partial w_1^2}$
$\frac{\partial f_1^1}{\partial w_1^3}$
$\frac{\partial f_1^2}{\partial w_1^4}$
$\frac{\partial f_2^1}{\partial w_2^1}$
$\frac{\partial f_2^2}{\partial w_2^2}$
$\frac{\partial f_2^1}{\partial w_2^3}$
$\frac{\partial f_2^2}{\partial w_2^4}$
$\frac{\partial f_3}{\partial w_3^1}$
$\frac{\partial f_3}{\partial w_3^2}$
$\frac{\partial Q}{\partial f_3}$
Backward propagation

Revisiting model training pipeline

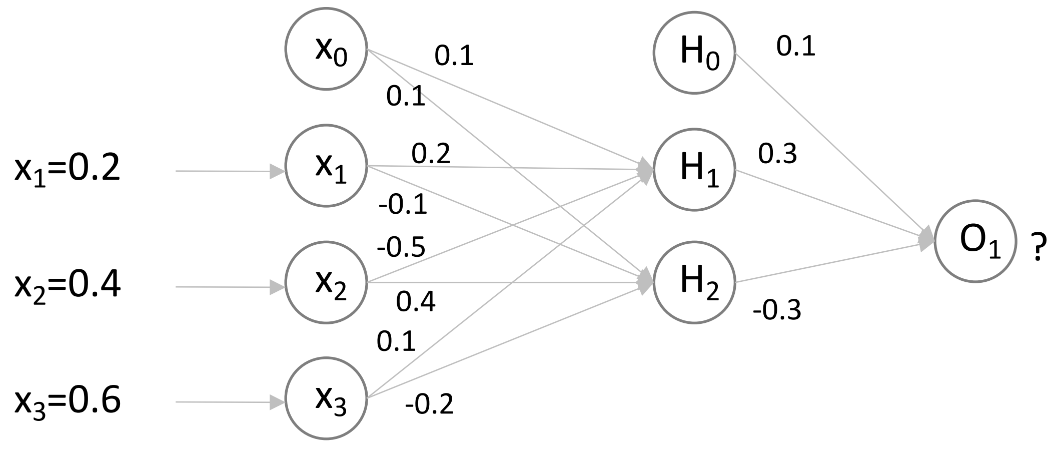

We choose a specific model family such as neural network. Depending on the architecture and parameters, the model class (i.e., all possible model configurations) determines the model's generalization performance.
The optimization procedure updates model weights based on the current cost. Using gradient descent, the weights are nudged towards the (possibly local) minimal cost controlled by the learning rate
$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} Q(\mathbf{w}_t)$
Optimization
The cost function measures the total (or average) distance (or similarity) between model prediction and true target for the current model
We split the original dataset into training set for model development and test set to assess the model's generalization performance, which relates to the size of the data.
$(\mathbf{X}, \mathbf{y})^{\text{train}}$
Training set
Original dataset
$(\mathbf{X}, \mathbf{y})$
Test set
$(\mathbf{X}, \mathbf{y})^{\text{test}}$
Model
$f_{\mathbf{w}}$
$\hat{y}^{\text{train}} = f_{\mathbf{w}}(\mathbf{X}^{\text{train}})$
Prediction
Cost
$Q(\mathbf{w}) = \sum_{i=1}^{N} l(y_i^{\text{train}}, \hat{y}_i^{\text{train}})$
Prediction
$\hat{y}_i^{\text{test}} = f_{\mathbf{w}}(\mathbf{X}^{\text{test}})$
Generalization performance
$R(f_{\mathbf{w}}) = \frac{1}{N} \sum_{i=1}^{N} l(y_i^{\text{test}}, \hat{y}_i^{\text{test}})$
The prediction function simply generates a set of predictions for a given set of (unchanging) input features and (changing) model weights.
The test set is used to calculate the empirical risk (i.e., mean squared error for continuous output), a proxy to the true risk and therefore a measure of the model's generalizability

What is next after one iteration?

- One forward propagation and one backpropagation conducted with one observation
- New weights are not too different from original weights
  - predictive error E_1 is not large
  - learning rate η is intentionally set as 0.5 and not large
- Next observation can be fed into the ANN and update all weights again slowly



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 $\hat{y} = 0.5227$

Exercise: Create ANN for Regression problem

MLPRegressor class from sklearn library

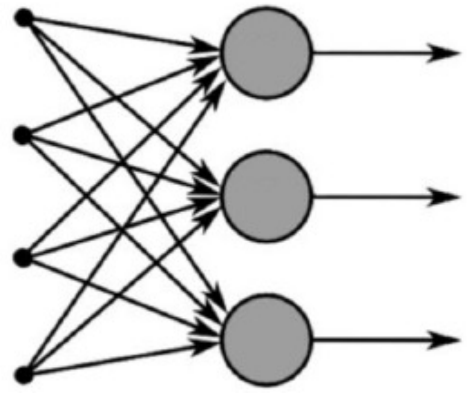
- Experiment hidden_layer_sizes to create different structure for the network
- Use appropriate learning rate

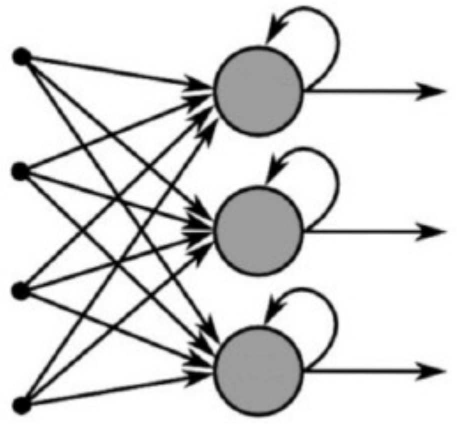
Sequential class from tensorflow library

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

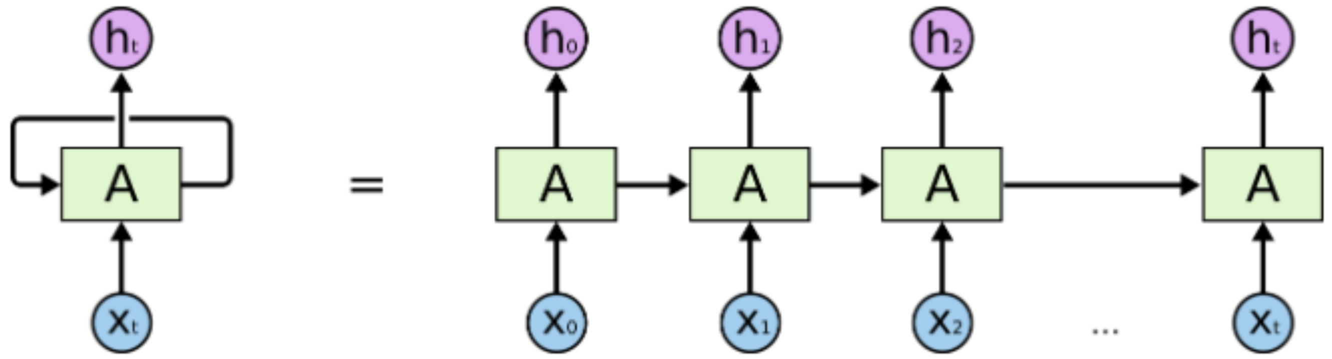
Recurrent Neural Networks (RNN)

- 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



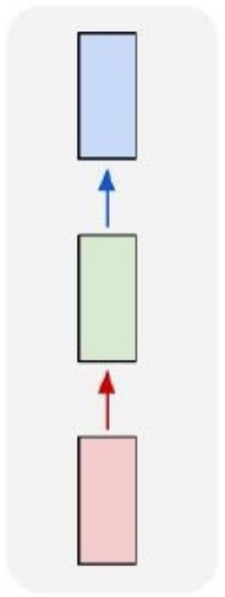
One neuron

- 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

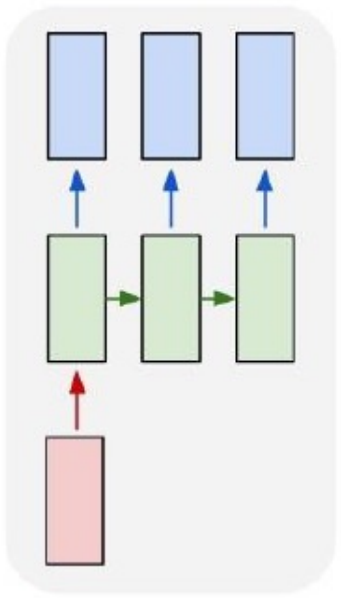
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Different kind of ANN (1/2)



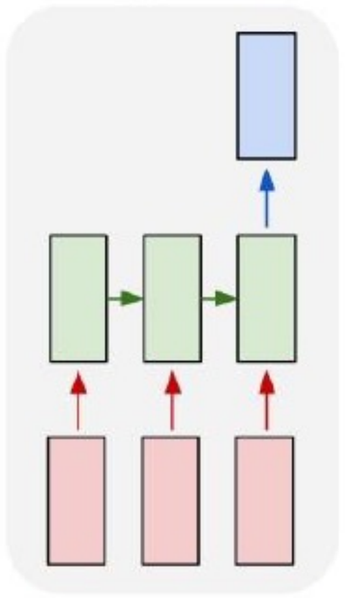
Simplest ANN

- 1 neuron
- 1 input
- 1 output



One to many, e.g.

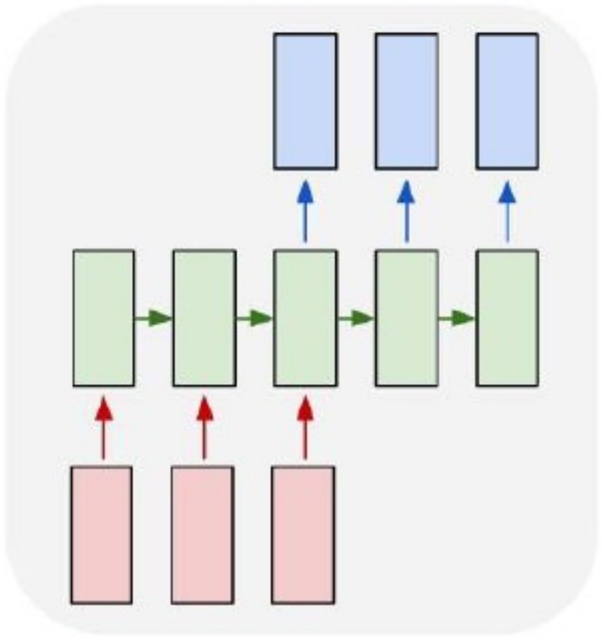
- Input: Image
- Output: Caption



Many to one, e.g.

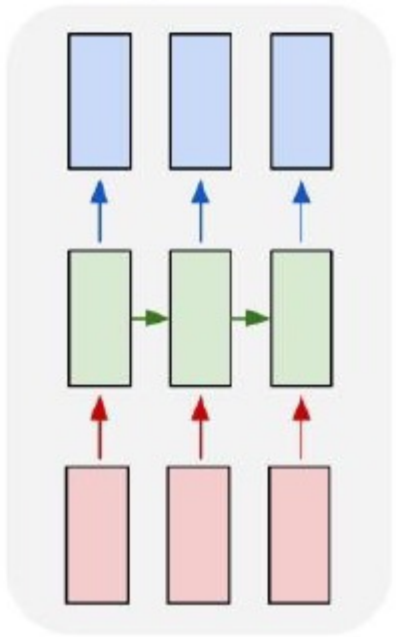
- Input: Words in an email
- Output: Spam or not

Different kind of ANN (2/2)



Many to many with time shift, e.g.

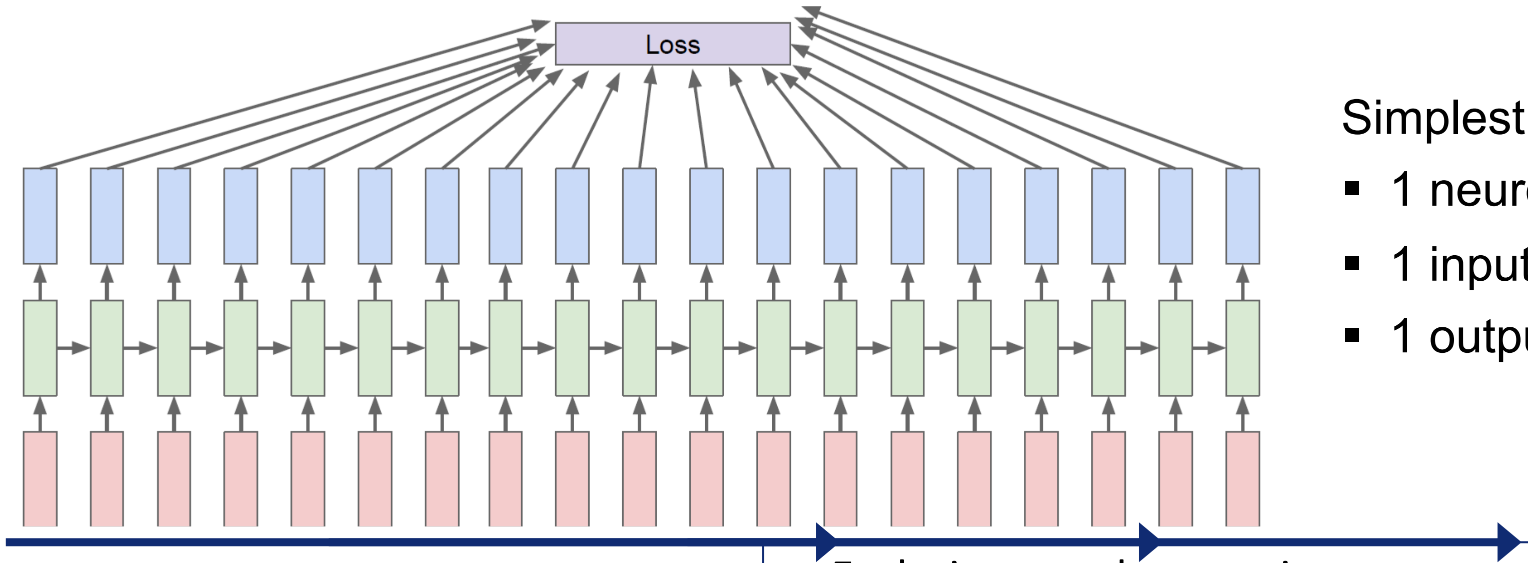
- Input: French sentence
- Output: English sentence



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

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

RNN mechanism for sequential data such as time series



Simplest ANN

- 1 neuron
- 1 input
- 1 output

- Each timestep has one input, one copy of the network, and one output
- Errors are then calculated and accumulated for each timestep, contributing to the overall loss

- The network goes back in time
- Each previous timestep can be considered a layer
- The weights are updated

Long short-term memory (LSTM)

- Since the output of a recurrent neuron at time step t is a function of all the inputs from previous time steps, it has a form of memory, hence the word 'memory'

- **Memory Cell** is the building component of RNN, which preserves some state across time steps

- **Long-Term Dependencies**: when the time difference between the relevant information and the point where it is needed become very large
  - e.g. "Mary's salary was 5000. <1000 words>. She got a raise for 100. At a grand total of ____"
  - How can the model predict the next word?

- LSTM is **capable** of learning long-term dependencies

Recap and next lecture

Recap

- ANN
- Neuron in brain
- Forward propagation
- Backpropagation
- Activation function
- Weight
- Bias term
- Calculation illustration
- Use trained ANN for prediction
- RNN

Next lecture

- Statistical arbitrage with machine learning

Homework

- Post learning reflections and questions if any
- Complete group assignment and submit as a group before week 7 lecture
- Continue to work on final group project