Deep Learning and Convolutional Neural Network (42028)

Week 4: Lecture

Neural Network in details

Logistic Regression – Recap

Problem of Binary Classification:
Dog? → 1
Cat (≈ not Dog)? → 0

Input (x)

Model
ANN Architecture
+
Parameters

0
Output (y)

1
Target

Loss/Cost function

Error/ Loss

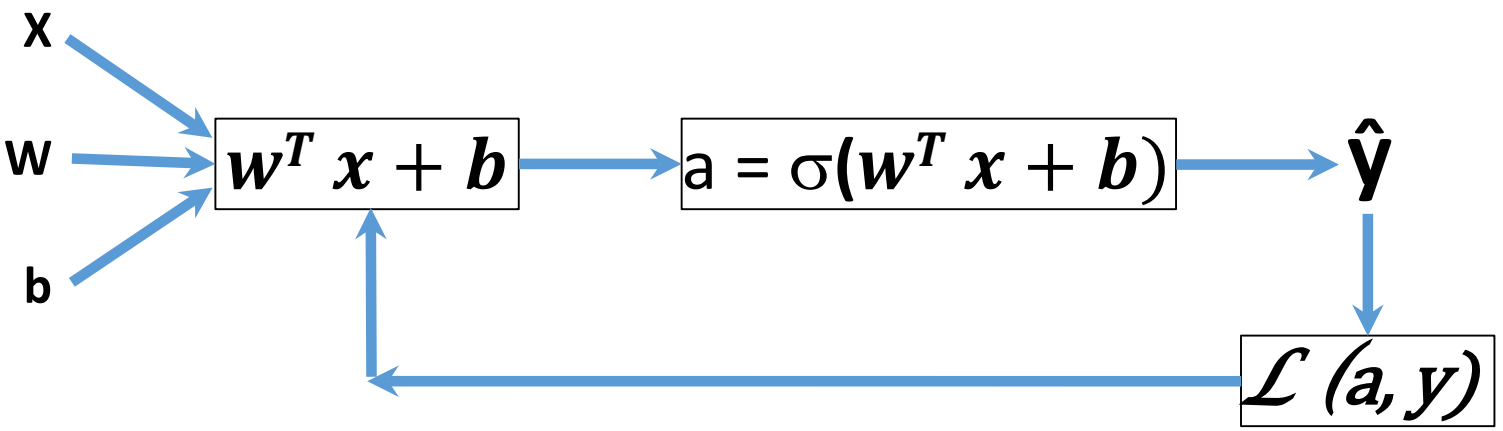
Gradient

Gradient Decent

Activation function

Logistic Regression as Neural Network

Logistic Regression pipeline with the math looks like:



Where,

W → Weights

X → Inputs

b → Bias term

σ → Activation function

Parameters:

1. w (weight)
2. b (bias)
3. Output a = $\sigma(w^T x + b)$

Activation function

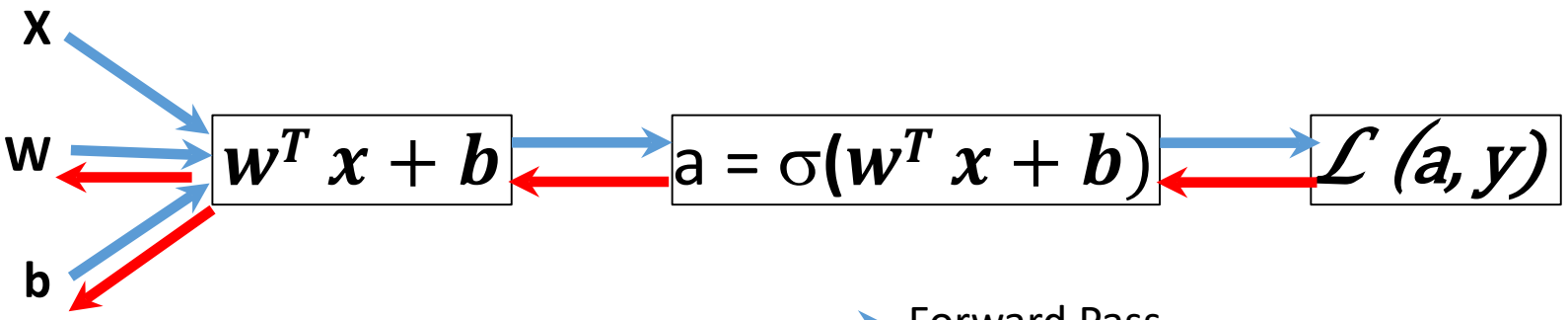
$$a = \sigma = \left(\frac{1}{1+e^{-x}}\right)$$

Loss function for Logistic Regression:

$$\mathcal{L}(a, y) = -(y \log a + (1 - y)\log(1 - a))$$

Logistic Regression as Neural Network

Logistic Regression pipeline with the math looks like:



repeatedly adjust the weights to minimize the difference between actual output and desired output

Where,

W → Weights

X → Inputs

b → Bias term

σ → Activation function

Activation function

$$a = \sigma = \left(\frac{1}{1+e^{-x}}\right)$$

Parameters:

1. w (weight)
2. b (bias)
3. Output a = $\sigma(w^T x + b)$

Loss function for Logistic Regression:

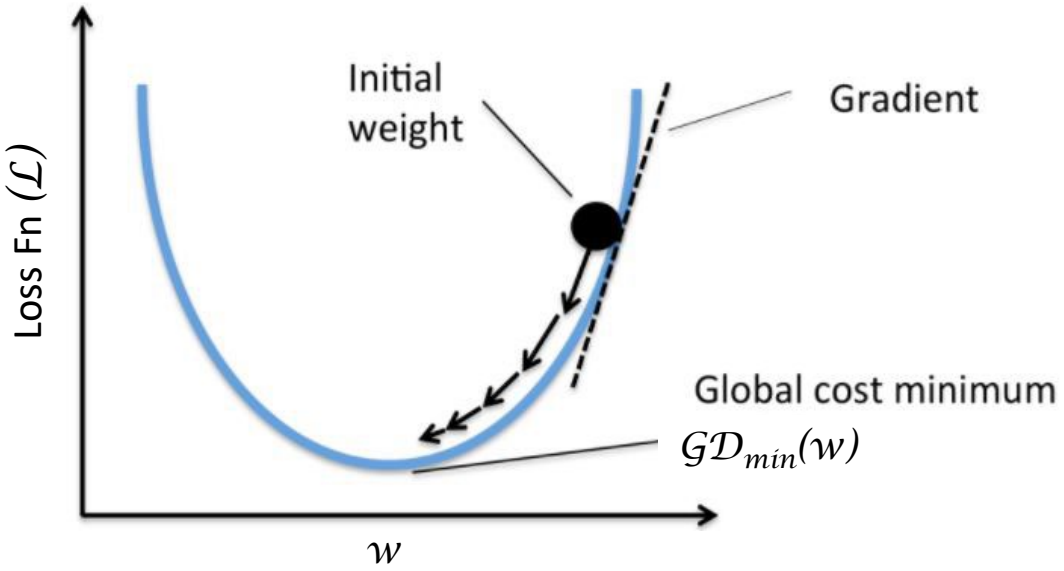
$$\mathcal{L}(a, y) = -(y \log a + (1 - y)\log(1 - a))$$

Gradient Descent

Gradient Descent for learning parameters:
It is an iterative approach for error correction in a machine learning model.

Question: Find w and b that will minimize $\mathcal{GD}(w, b)$

Required: Loss/cost function



Generic Algorithm:

Step 1: Initialize **w** and **b**
Step 2: Perform Forward pass operation/calculations
Step 2: Compute Loss/Cost function $\mathcal{L}(a, y)$
Step 3: Compute change in **w** and **b**
(Take the partial derivative of the cost function with respect to Weights and bias (**dw** and **db**).
Step 4: Update **w** and **b**
w := w – α**dw**
b := b – α**db**
Step 5: Repeat from Step 2 with new values of **w** and **b** for 'n' number of iterations.

Example the loss function is:

$$\mathcal{L}(a, y) = -(y \log a + (1 - y)\log(1 - a))$$

α → Learning rate

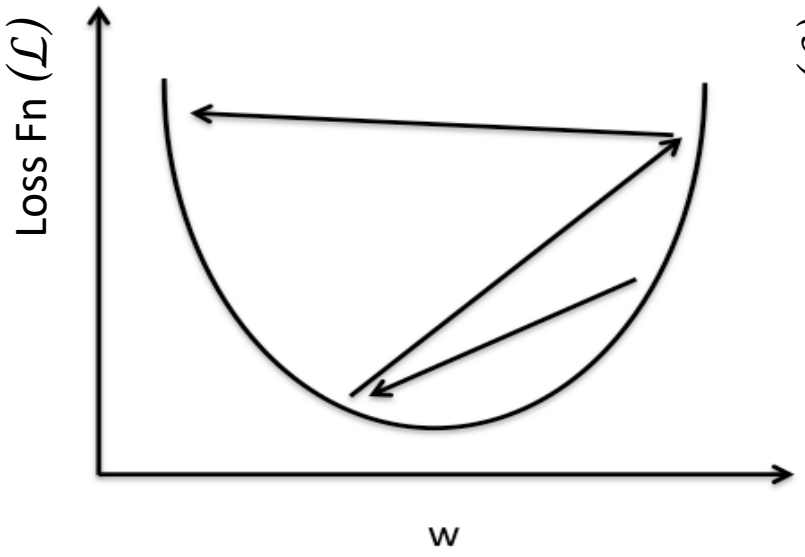
Image Source: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788397872/1/ch01lvl1sec22/gradient-descent
Source and Reference: http://cs230.stanford.edu/files/C1M2.pdf

Gradient Descent

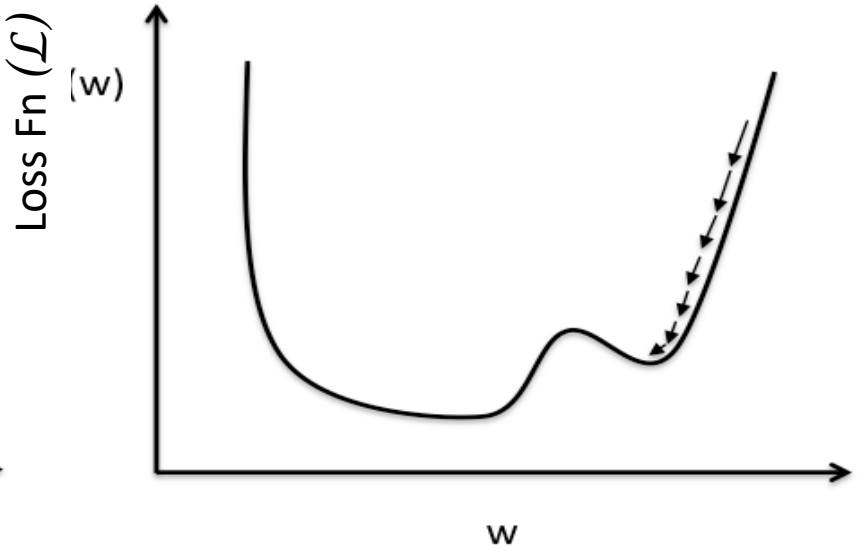
Gradient Descent for learning parameters:

Learning rate(α) issues:

- It is a ***hyper-parameter***



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

Image Source: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788397872/1/ch01lvl1sec22/gradient-descent
Source and Reference: http://cs230.stanford.edu/files/C1M2.pdf

Gradient Descent

Learning rate(α): more intuitions

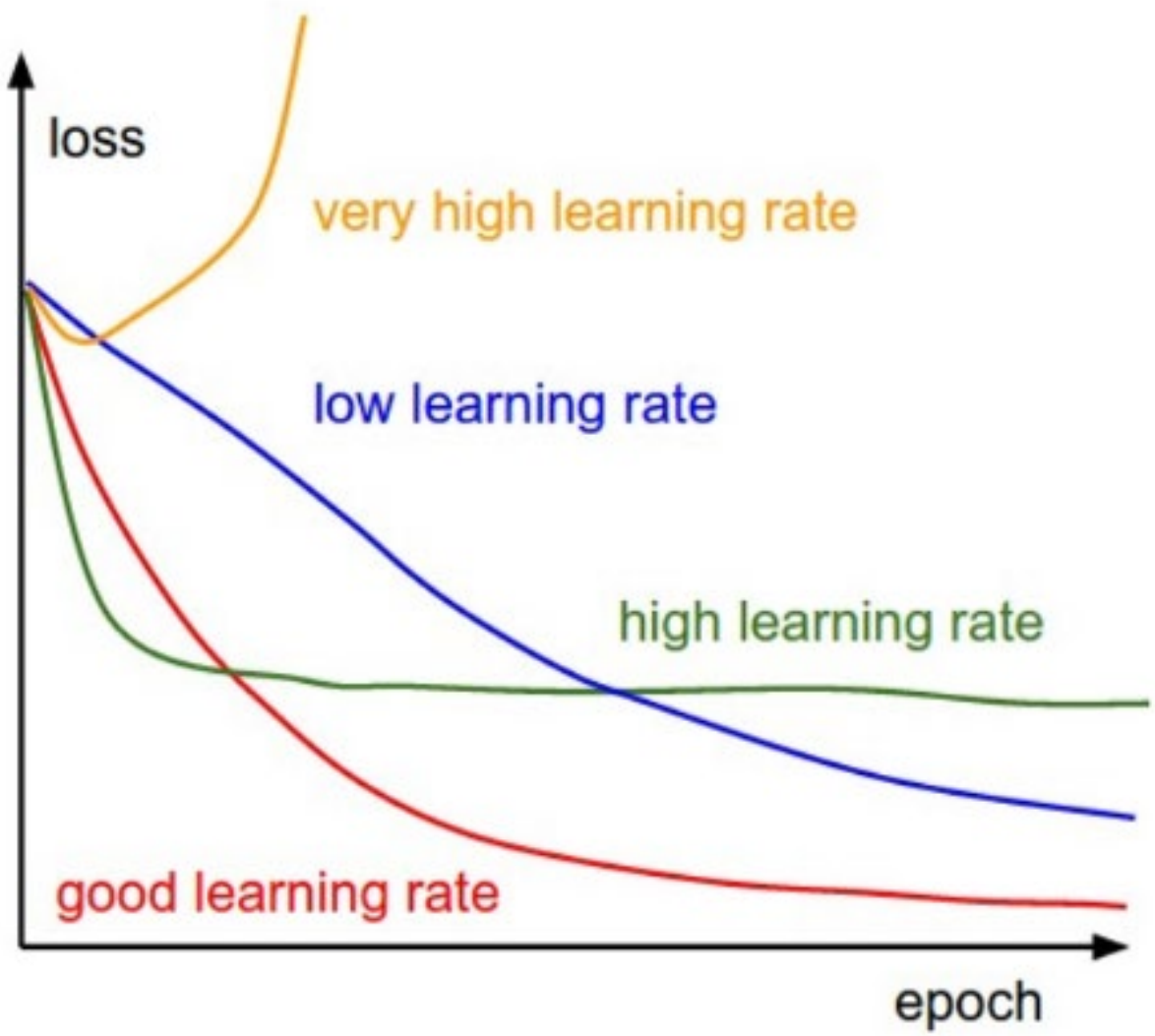


Image Source: http://cs231n.github.io/neural-networks-3/

Gradient Descent Types

There are three main types of Gradient Descent Algorithms:

1. Batch Gradient Descent (BGD)

2. Stochastic Gradient Descent (SGD)

3. Mini-Batch Gradient Descent (MBGD)

Batch Gradient Descent

Generic steps:

-Process each input sample and find the cost

-Find the average cost over all input samples

-Update **w** and **b**, and

-repeat the steps for ‘n’ epochs(iterations)

Issues:

1. It uses the complete dataset to calculate the gradients at every steps
2. Slow when training set is large
3. Difficult to find the learning rate
4. Difficult to ascertain the number of epochs(iterations)

Stochastic Gradient Descent (SGD)

Stochastic → Random

Due to the random nature, the Algorithm is much less regular than BGD

Generic steps:
-Process a random input sample and find the cost

-Update **w** and **b**, and

-repeat the steps for ‘n’ iterations on the training samples

Advantage:

1. Computes gradient based on single input sample: memory efficient
2. Much faster compared to BGD
3. Possible to train on large dataset
4. Randomness is a good escape from local minima problem

Issues:

1. Might not reach the optimal value, but very close to it.

Stochastic Gradient Descent (SGD)

Issues: Might not reach the optimal value, but very close to it.

Possible solution: Reduce the learning rate gradually → Stimulated annealing

Create a Learning Schedule to determine the learning at each iteration.

Epoch: One round through the complete training set.

Iterations: Process in multiple subsets of the training set, say, 'm' iterations my form 1 epoch

Mini-Batch Gradient Descent (SGD)

Generic steps:
-Divide the training set into mini-batches (set of random samples on fixed number)

-Process all the samples in a Mini-batch and find the average cost

-Update **w** and **b**, and

-repeat the steps for ‘n’ iterations/epochs on the training samples

Advantage:

1. Computes gradient based on small sets of input sample
2. Much faster compared to BGD
3. Possible to train on large dataset
4. Performance boost on matrix operations using GPUs!
5. Might not reach the optimal value, but very close to it, and possibly better than SGD

Issues:

1. It may be harder to escape the local minima.

Gradient Descent (SGD) - intuition

Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

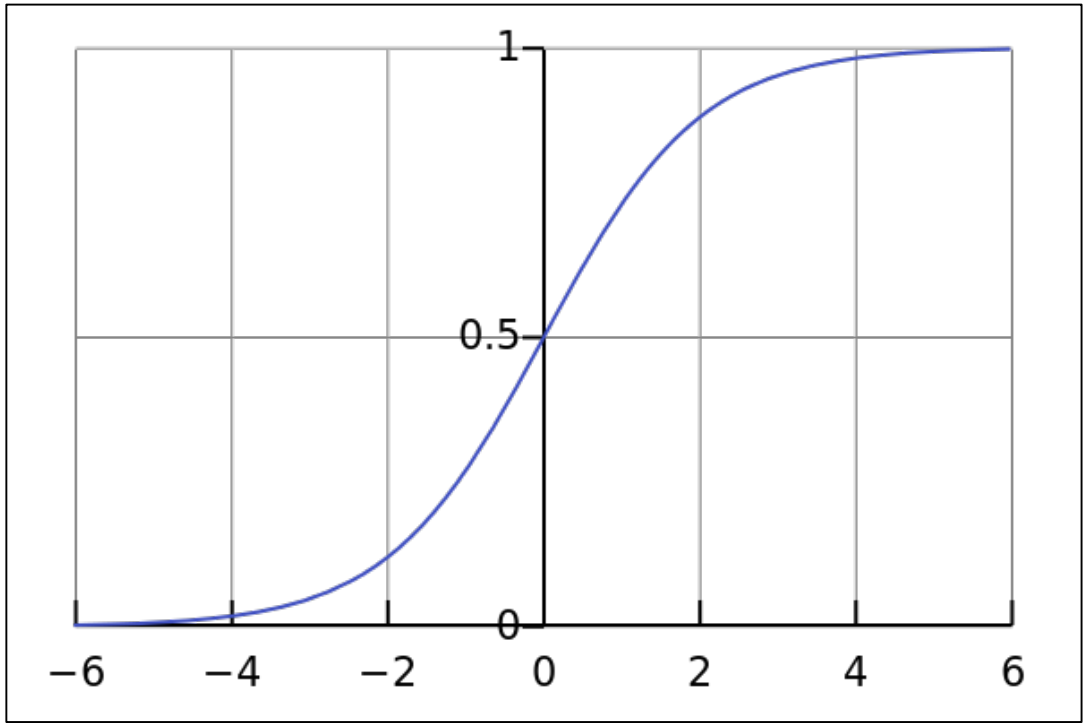
Image Source: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

More Activation Functions

Sigmoid function: $\sigma = \left(\frac{1}{1+e^{-x}}\right)$

Characteristics:

- **Non-linear in nature**
- **Range(0, 1)**
- **Tends to bring the activations to either side of the curve: good for a classifier**
- **Suffers from vanishing gradient problem**



Vanishing Gradient: Towards to the end of the curve, the value of Y change very less to the changes in X values. Hence gradient at the region will be very small. The network will refuse or learning extremely slowly.

Source: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

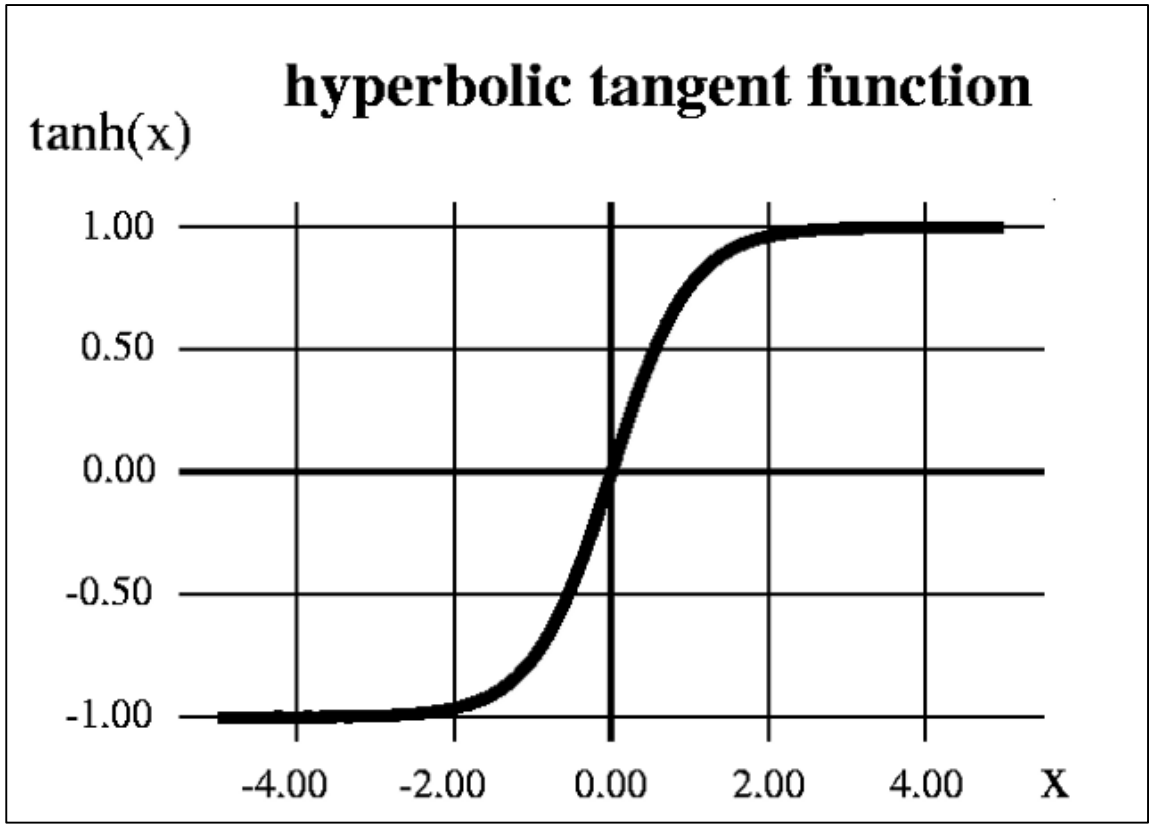
More Activation Functions

Hyperbolic tangent:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Characteristics:

- **Non-linear in nature**
- **Range(-1, 1)**
- **Stronger gradient than sigmoid**
- **Also suffers from vanishing gradient problem**



More Activation Functions

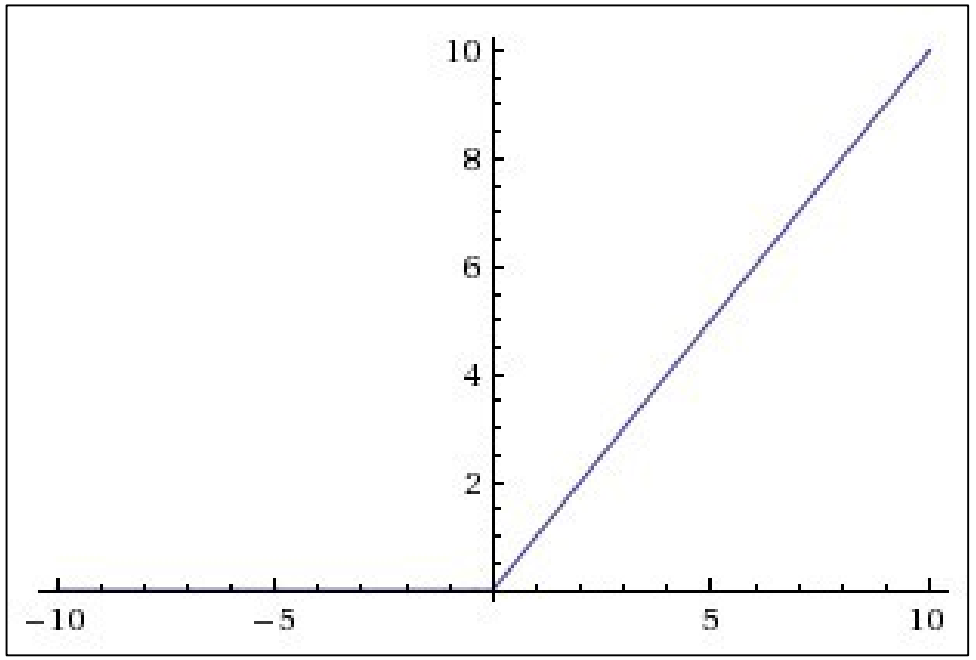
Rectified Linear Unit (ReLu)

$$A(x) = \max(0, x)$$

i.e. : if x < 0, *A(x)* = 0, if x > 0, *A(x)* = x

Characteristics:

- **Non-linear in nature**
- **Range[0, inf]**
- **Stronger gradient than sigmoid**
- **Computationally less expensive than Sigmoid and Tanh**
- **Best used in hidden layers**
- **Dying ReLu problem**



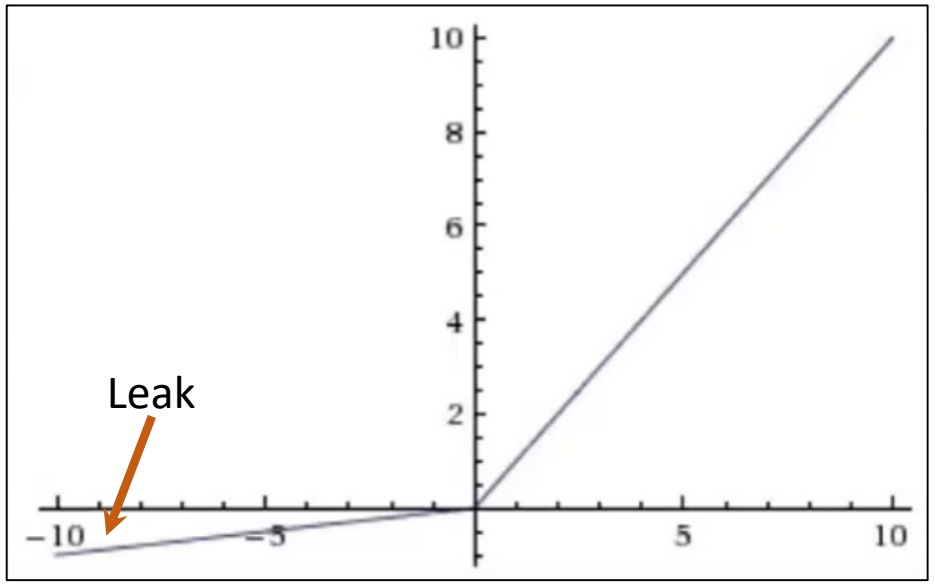
Avoids and rectifies **vanishing gradient** problem

More Activation Functions

Leaky Rectified Linear Unit (Leaky ReLu)

$$A(x) = \max(0.01x, \mathrm{x})$$

i.e. : if x < 0, *A(x)* = 0.01x, if x > 0, *A(x)* = x



Characteristics:

- **Non-linear in nature**
- **Range[0, inf]**
- **Leaky ReLUs are one attempt to fix the "dying ReLU" problem**

Source & Reference : http://cs231n.github.io/neural-networks-1/
https://towardsdatascience.com/activation-functions-in-neural-networks-58115cda9c96

More Activation Functions

$$\text{Softmax} \quad S(y_i) = \frac{e^{y_i}}{\sum_j (e^{y_j})} \quad \textit{for } j = 1, \ldots, K.$$

Characteristics:

- **Non-linear in nature**
- **Turns numbers in probabilities that sum to one.**
- **Useful when we have more than one output**
- **Used for classification in the output layer**
- **Less computationally expensive than Sigmoid and Tanh**

Illustration:

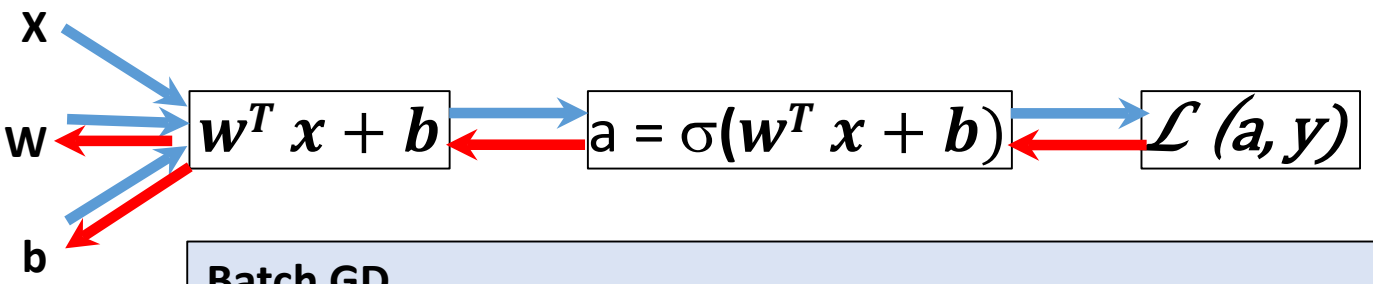
Y = [2.0, 1.0, 0.1]

Softmax(Y) = [0.7, 0.2, 0.1] (approx.)

Reference: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

Logistic Regression with Backpropagation

Logistic Regression pipeline with the math looks like:



Batch GD

Step 1: Initialize **w** and **b**

Step 2: Perform Forward pass operation/calculations

Step 2: Compute Loss/Cost function $\mathcal{L}$ (a, y)

Step 3: Find the average cost over all input samples

(Take the partial derivative of the cost function with respect to Weights and bias (**dw** and **db**).

Step 4: Update **w** and **b**

w := w – α**dw**

b := b – α**db**

Step 5: Repeat from Step 2 with new values of **w** and **b** for 'n' number of iterations.

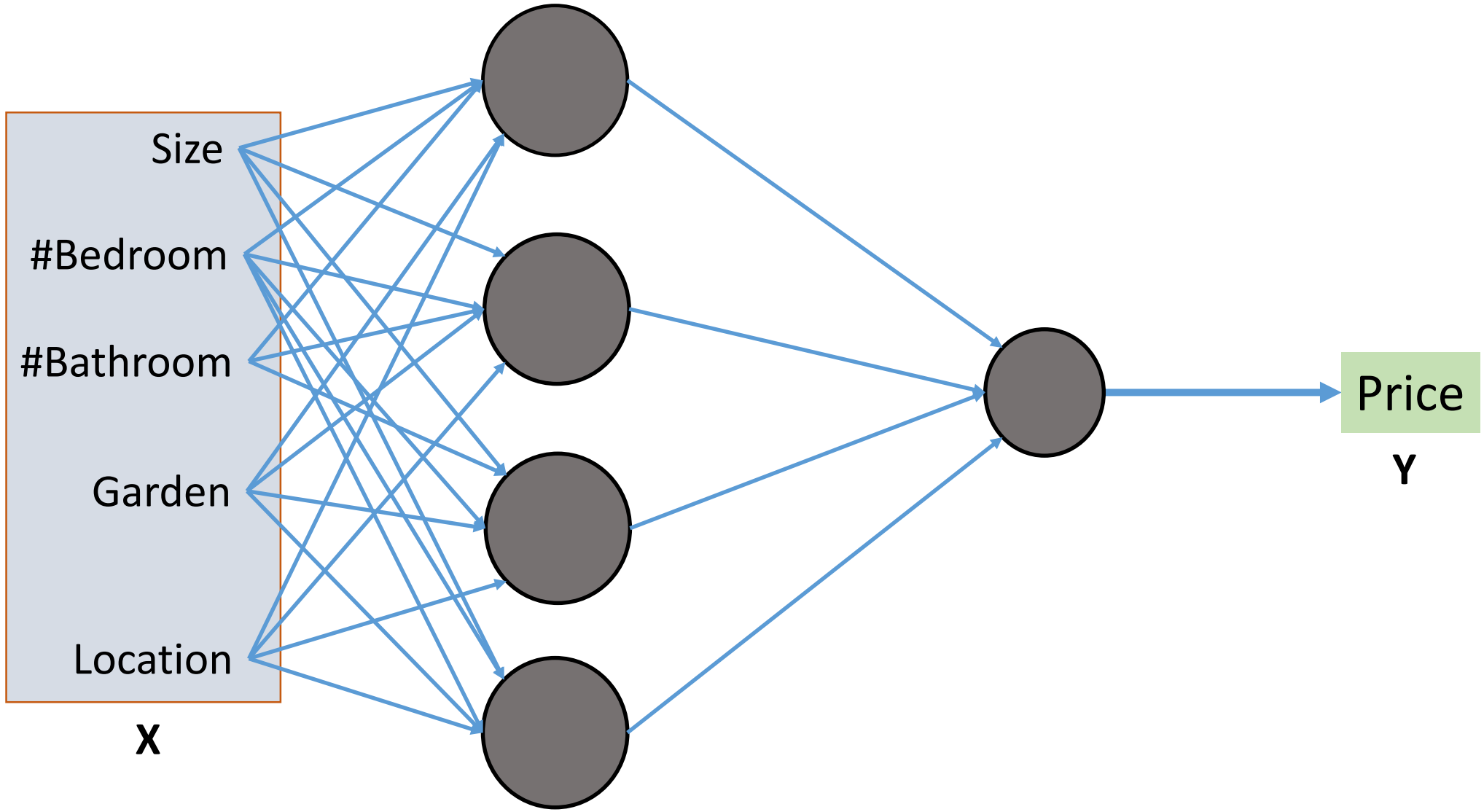
Average cost over all training 'm' samples

$$\text{Avg Loss(J)} = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a_i, y_i)$$

$$\text{dw} = \frac{\partial J}{\partial w}, \qquad \text{db} = \frac{\partial J}{\partial b}$$

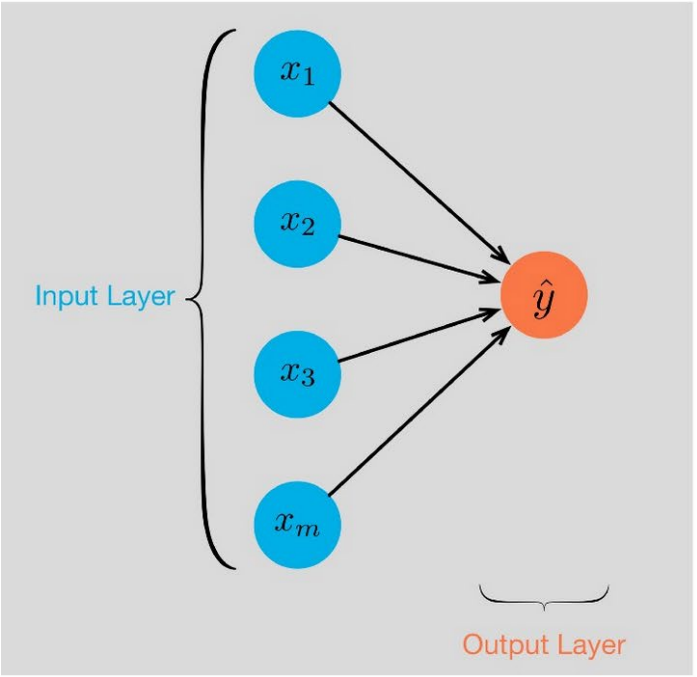
$$w := w - \alpha \mathbf{dw}$$
$$b := b - \alpha \mathbf{db}$$

Multi-layer Neural Network

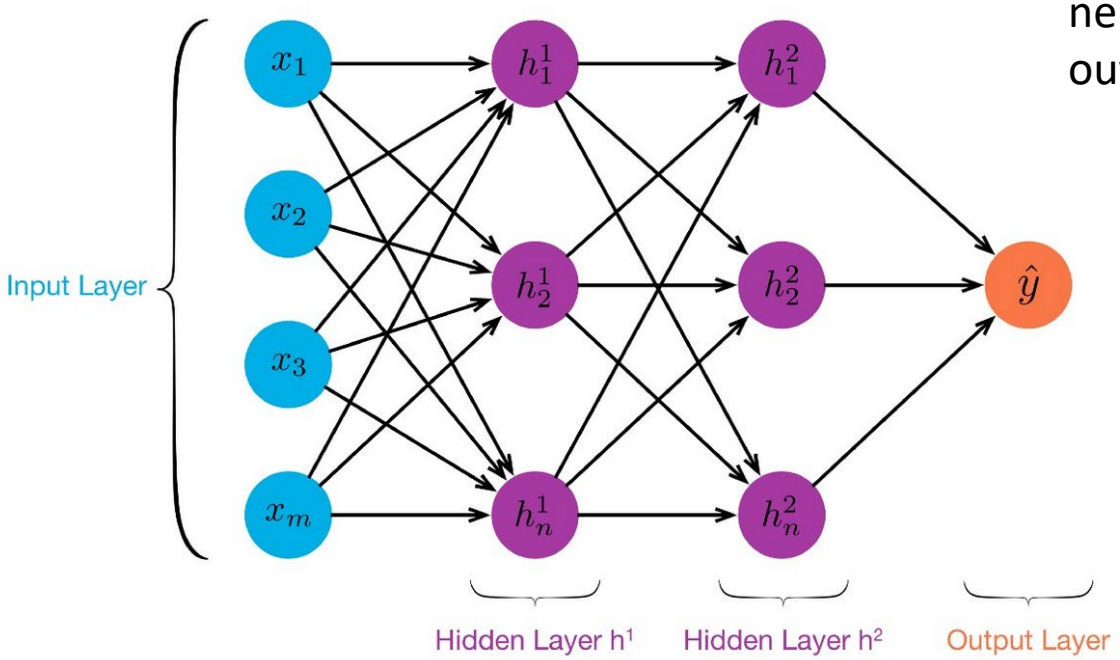


Multi-layer Neural Network

Hidden Layer→ Adding more neurons in between input and output layer



Single layer perceptron

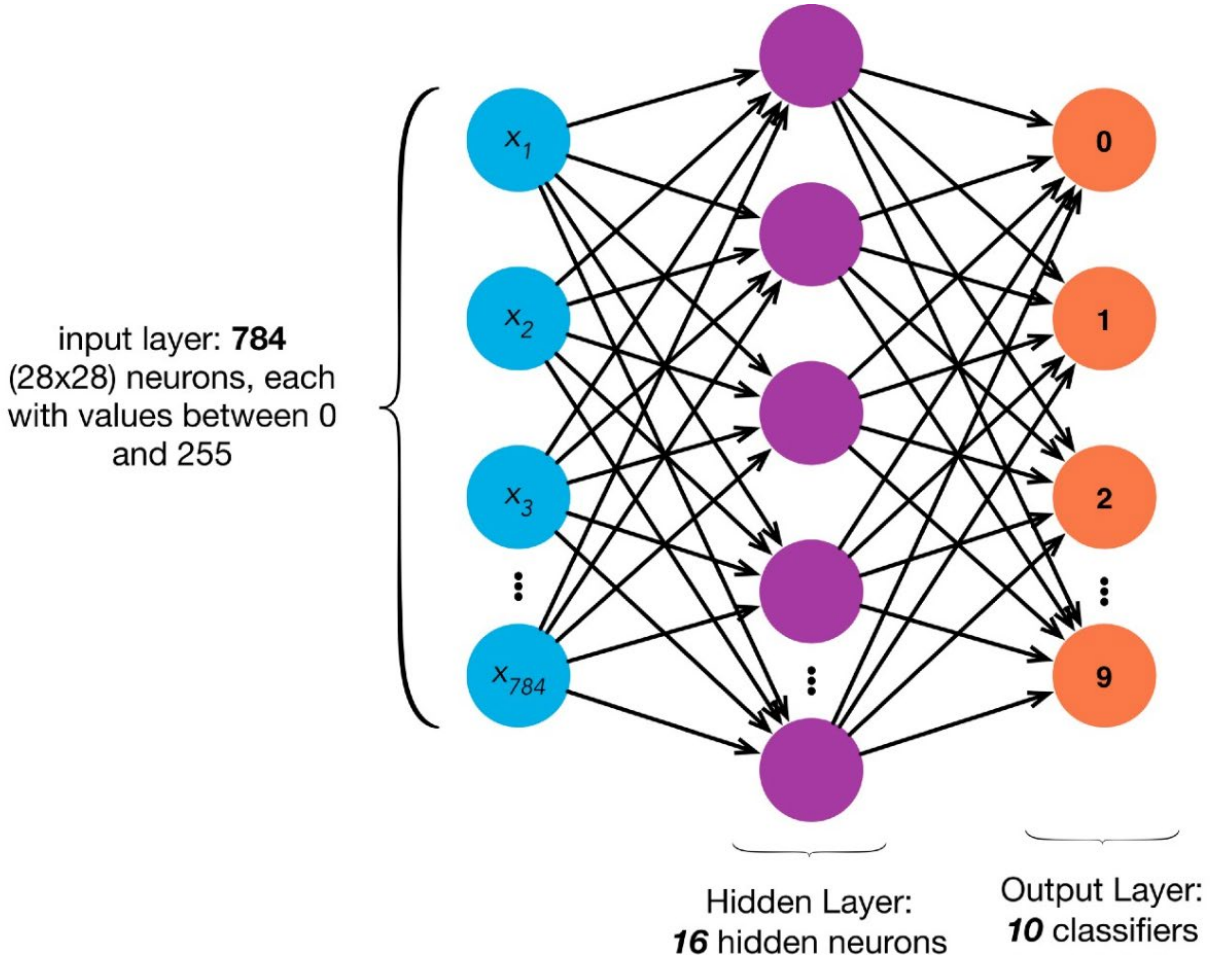


3-layered neural network with 2 hidden layers

Image Source: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Multi-layer Neural Network

Example:
2-layered architecture for multi-class classification
(e.g: Fashion MNIST dataset)

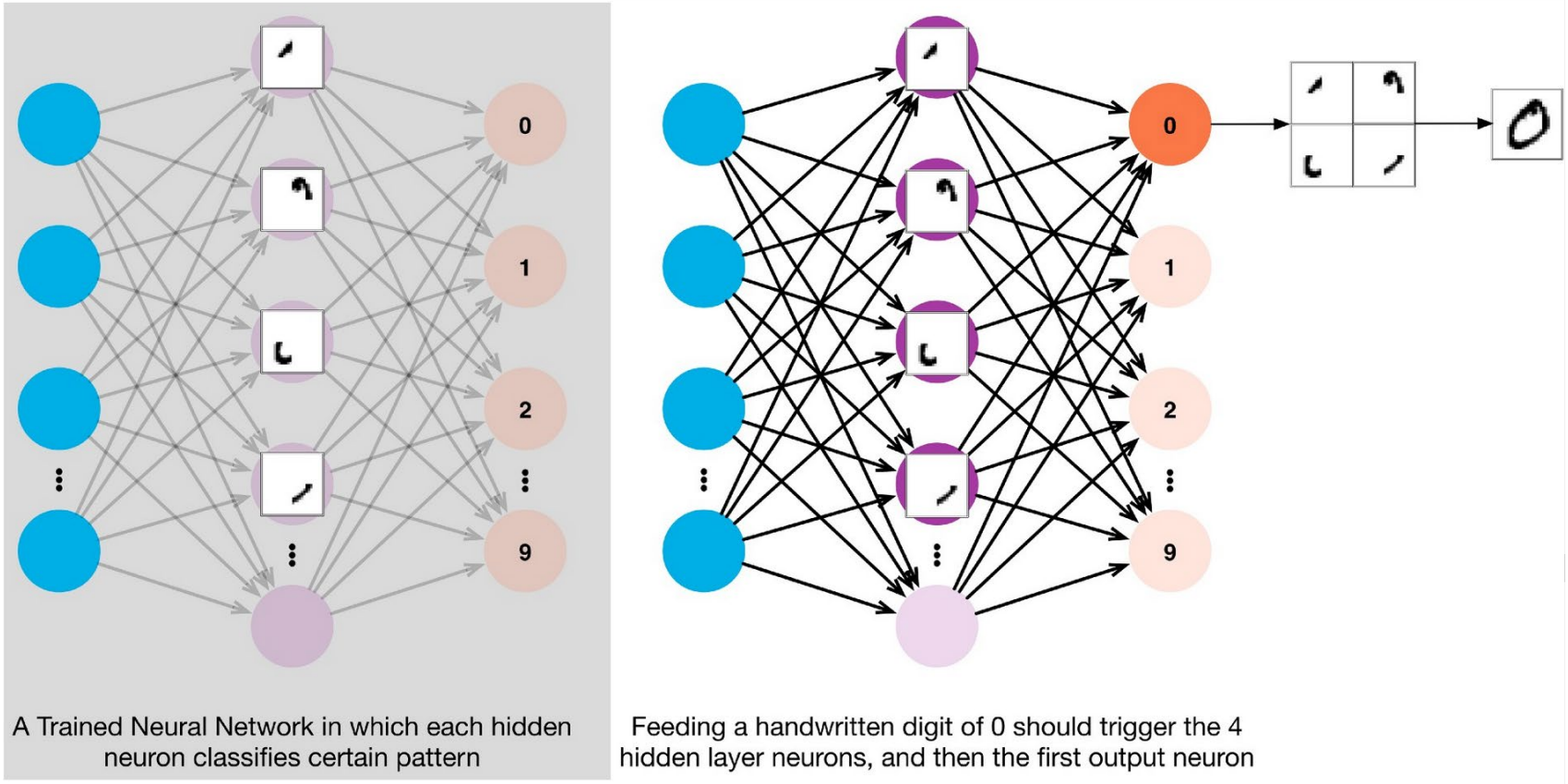


Intuition:
In a multi-layer neural network, the first hidden layer will be able to learn some very simple patterns. Each additional hidden layer will somehow be able to learn progressively more complicated patterns.

Image & content Source (Adapted) and reference to read : https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Multi-layer Neural Network

Example:
2-layered architecture for multi-class classification
(e.g: MNIST digit dataset) intuition



A Trained Neural Network in which each hidden neuron classifies certain pattern

Feeding a handwritten digit of 0 should trigger the 4 hidden layer neurons, and then the first output neuron

Image & content Source (Adapted) and reference to read : https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f