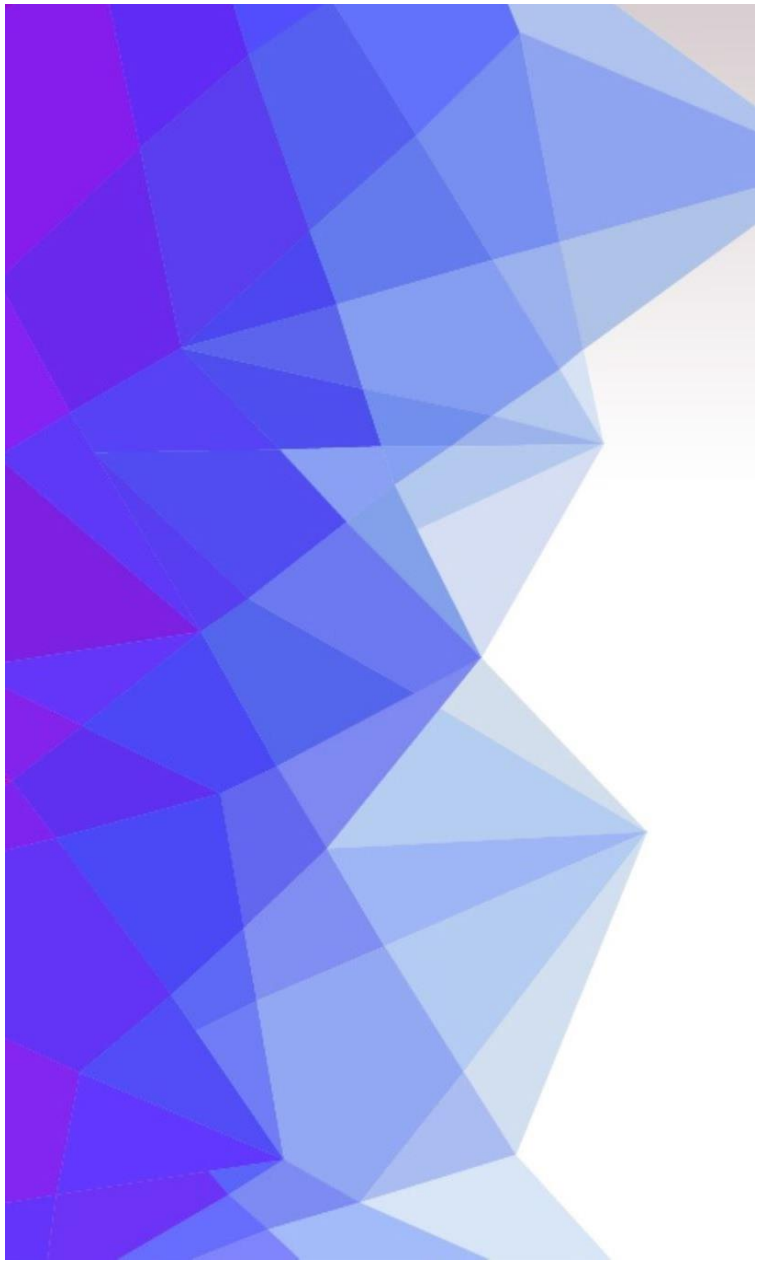
Dansby Lecture

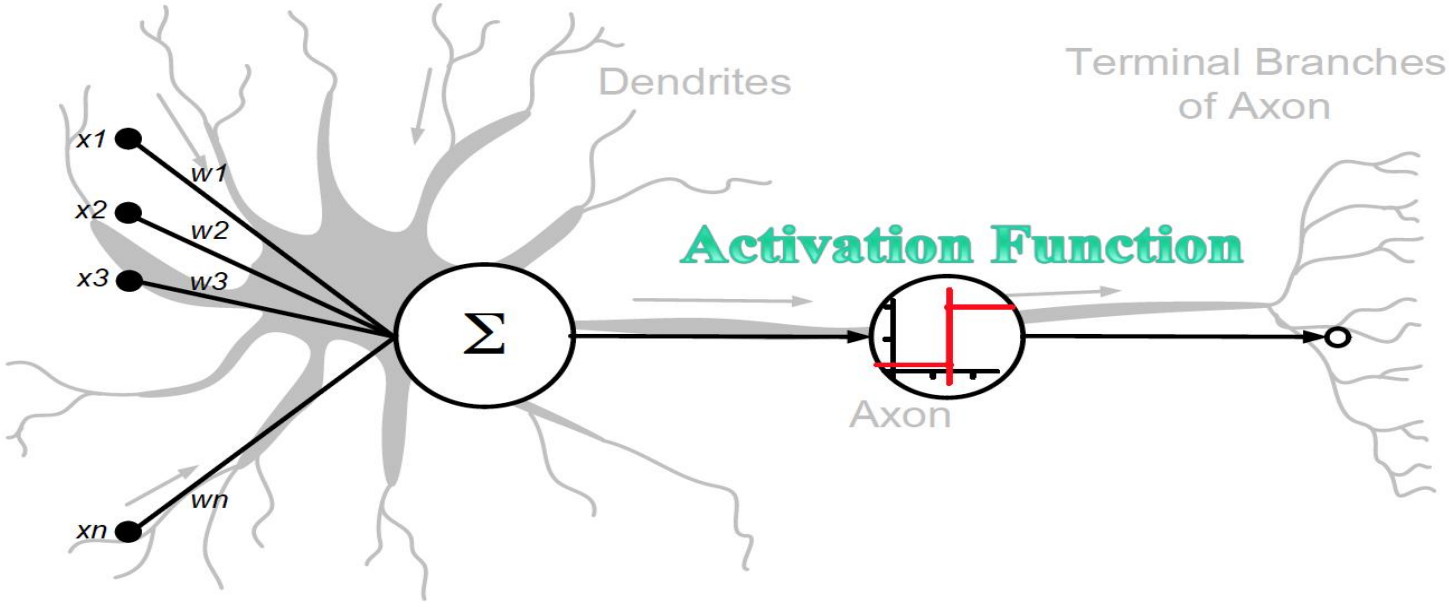
Thursday February 29, 2024

Structure of the Deep Neural Network model

By Dr. A. Brania, Morehouse Mathematics Department



Neural Network



- Neural Networks are a branch of Machine Learning models (sets of algorithms) inspired by the biological brain.
- A Deep Neural Network (DNN) is a NN that has an input layer, an output layer, and at least one layer in between.

Machine Learning

1. Machine Learning (ML) is the science/art of programming a computer so it can *learn from data*.

2. (A. Samuel, 1959) ML is the field of study that gives a computer the ability to *learn without being explicitly programmed*.

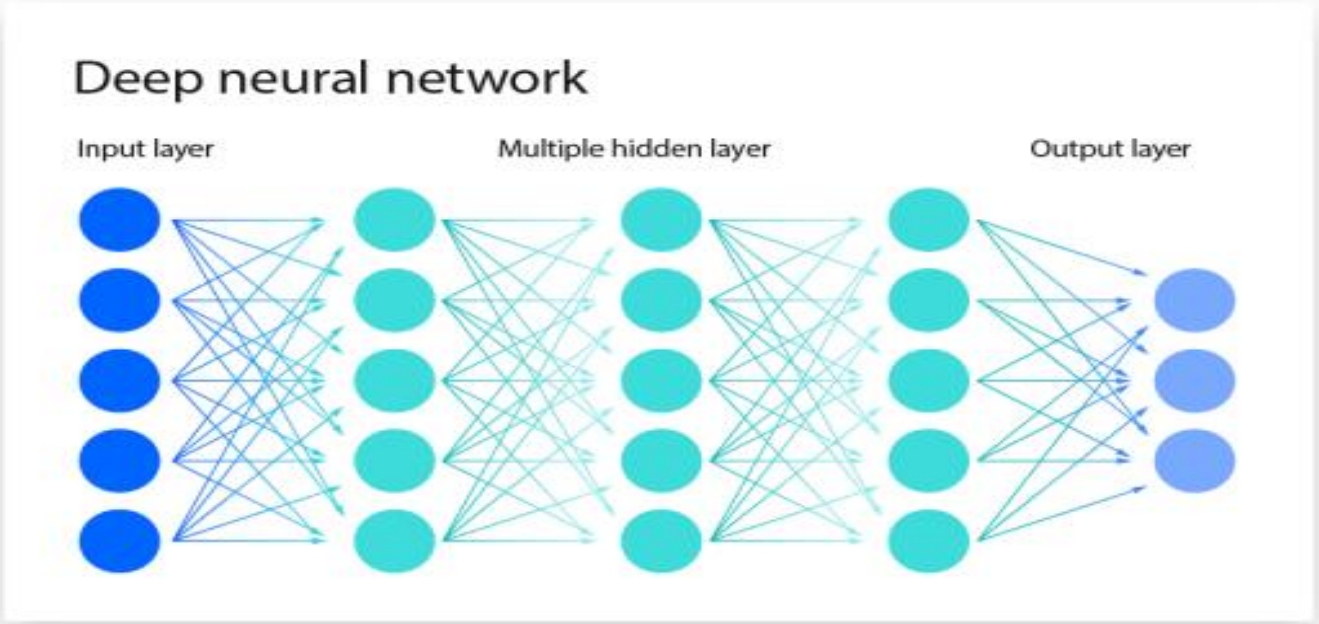
3. (T. Mitchell, 1997) A computer program *learns from experience* E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Example:

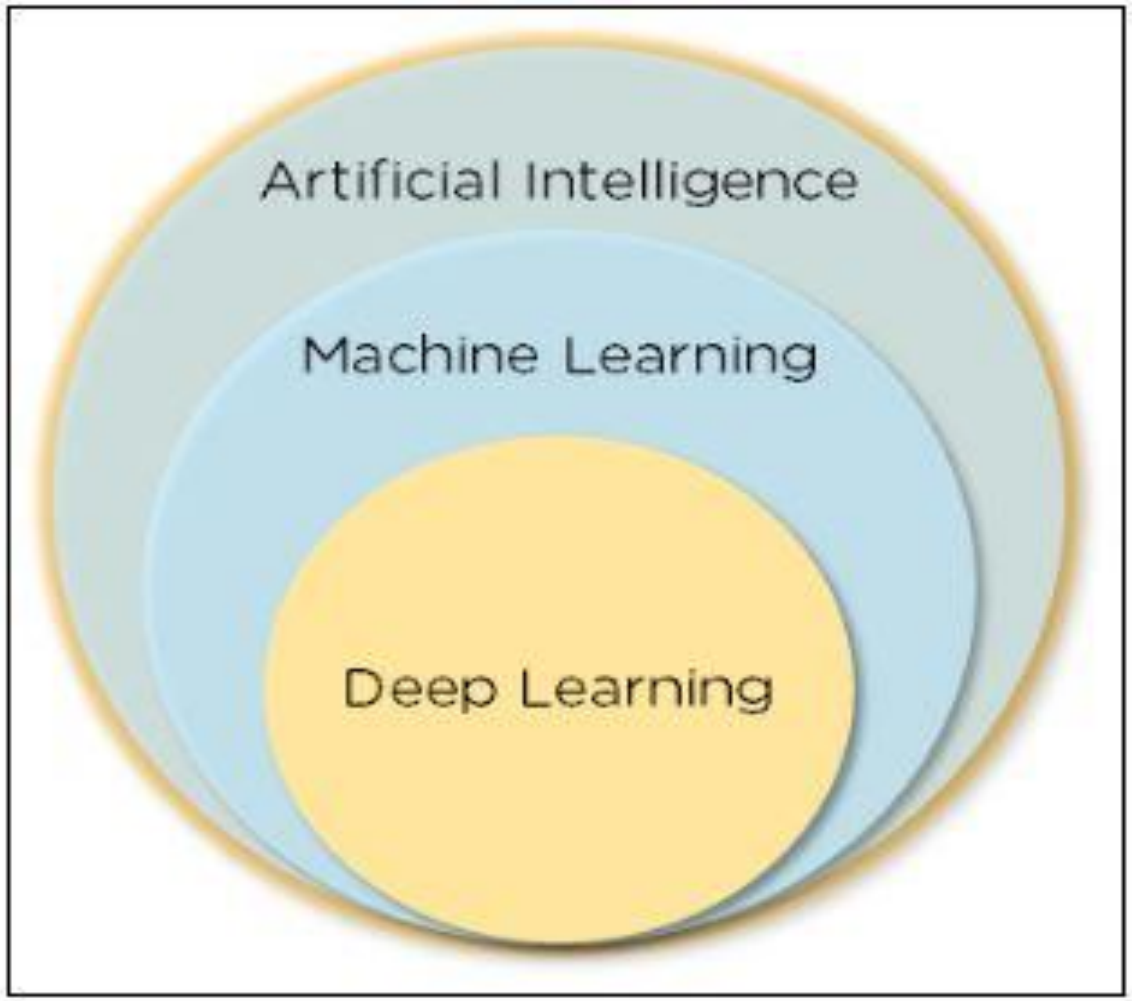
A spam filter is a ML program that, given examples of spam emails (flagged by users) and examples of non-spam emails, can learn to flag spam. The examples that the system uses to learn are called *the training set*.

The part of the ML system that learns and makes predictions is the *model*.

DNN or Deep Learning



Deep Learning is a subset of Machine Learning which uses multi-layered neural networks.



Machine learning is a subset of AI

Why use ML?

Machine Learning is great for:

- Problems for which existing solutions require a lot of fine-tuning or long lists of rules (ML model can often simplify code and perform better than traditional approaches, e.g spam filter)
- Complex problems for which traditional approaches yield no good solution (the best ML techniques can (perhaps) find a solution, e.g. speech recognition)
- Fluctuating environments (a ML system can easily be retrained on new data, always keeping it up to date)
- Getting insights about complex problems and large amounts of data (data mining)

Examples of applications

- Analyze images of products on a production line to automatically classify them (image classification using convolution neural network (CNN) or transformers)
- Detect tumors in brain cells (semantic image segmentation, each pixel is classified using CNN, transformers)
- Automatically classify news articles (natural language processing (NLP) using recurrent neural networks (RNN) and CNN, or transformers)
- Automatically flag offensive comments on discussion forums (text classification like for NLP tools)
- Create a chatbot or a personal assistant (NLP components)

(Continued)

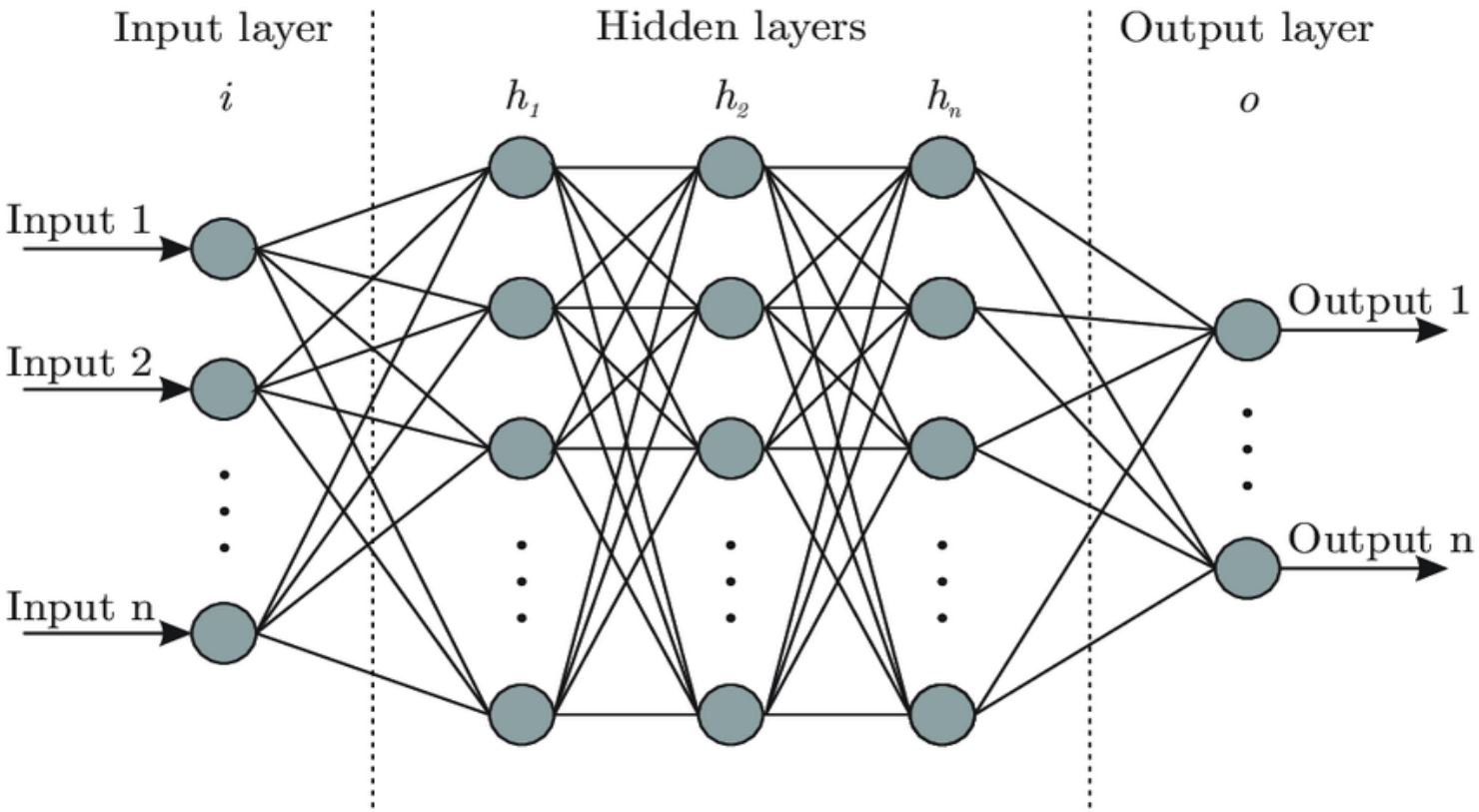
- Forecast next year's company's revenue based on several performance metrics (regression task that predicts a value)
- Detect credit card fraud (anomaly detection)
- Segment clients based on purchasing patterns to design a different marketing strategy for each segment (system clustering).
- Build an intelligent bot for a game (use reinforcement learning)

Types of ML Systems

Many different types of machine learning systems classified in 3 broad categories:

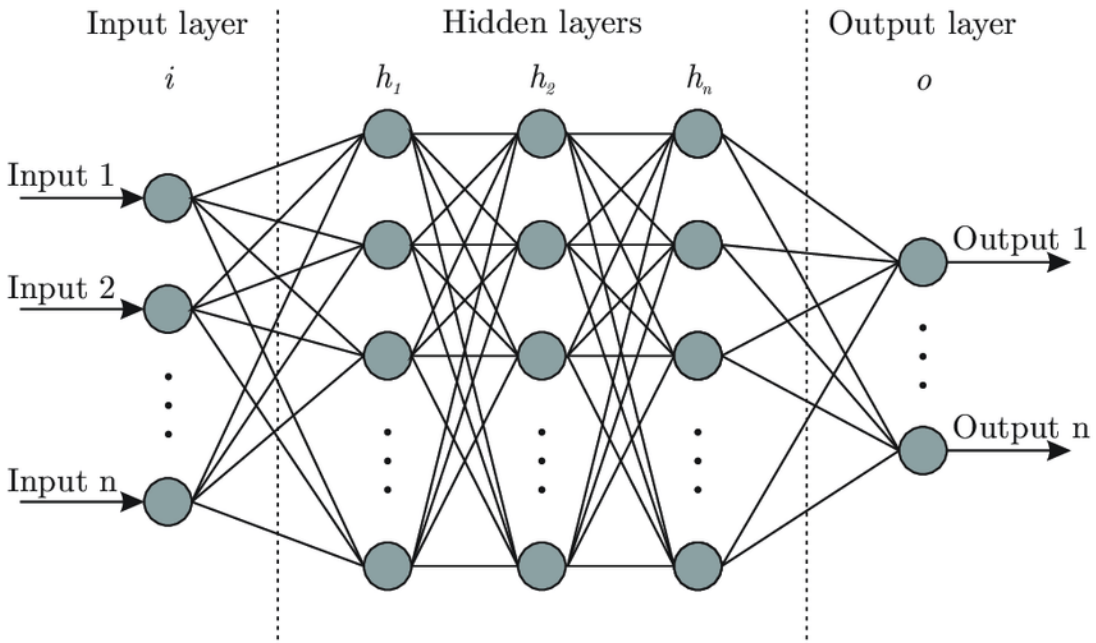
1. How they are supervised during training (supervised, unsupervised, semi-supervised, self-supervised, etc…)
2. Whether or not they can learn incrementally on the fly (online versus batch learning)
3. Whether they work by simply comparing new data points to known data points, or by detecting patterns in the training data and building a predictive model (instance-based versus model-based learning).

How does a neural network work?



NN Architecture of MLP: Input, Hidden, Output
Layers of interconnected nodes (Neurons)

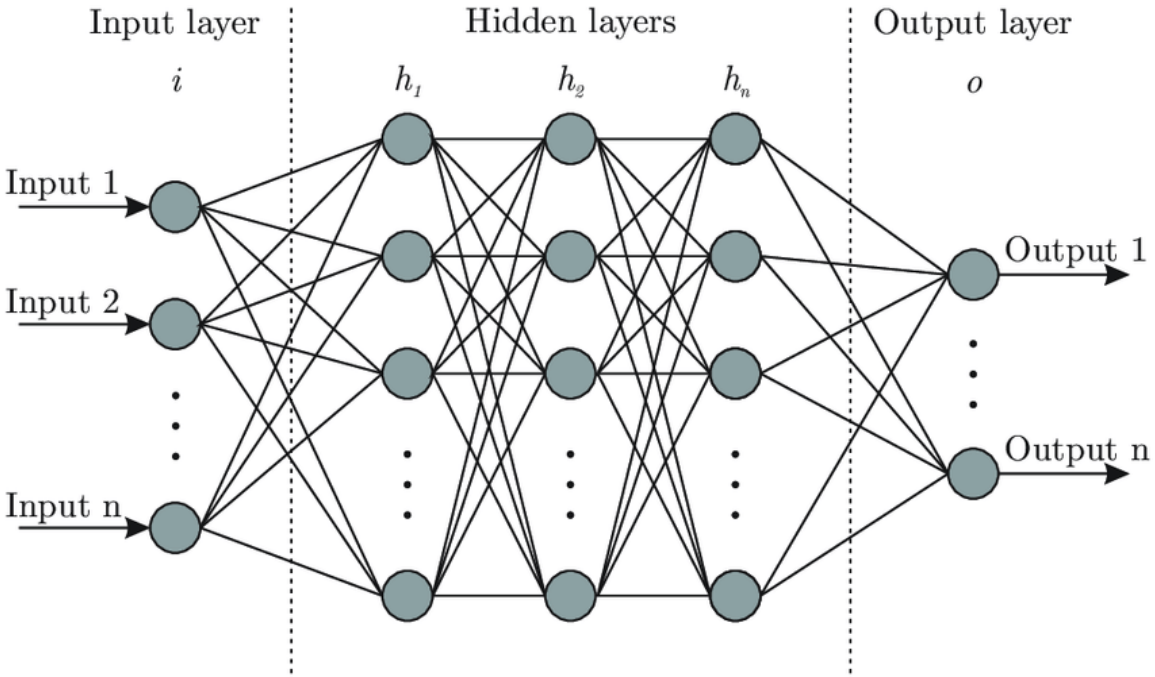
The Input Layer

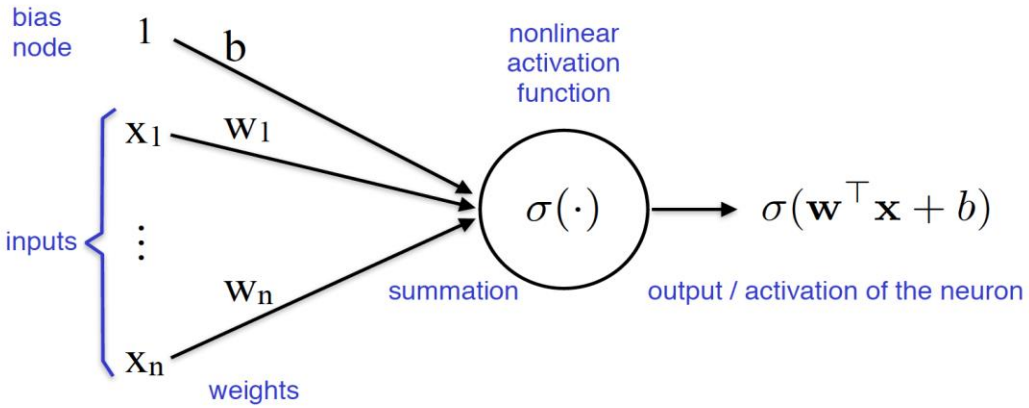


- The nodes of the input layer get data values and do not change the data.
- The nodes of the input are connected to the nodes of the first hidden layer.
- The input layer is different from the other layers (nodes are passive)

The Hidden Layers

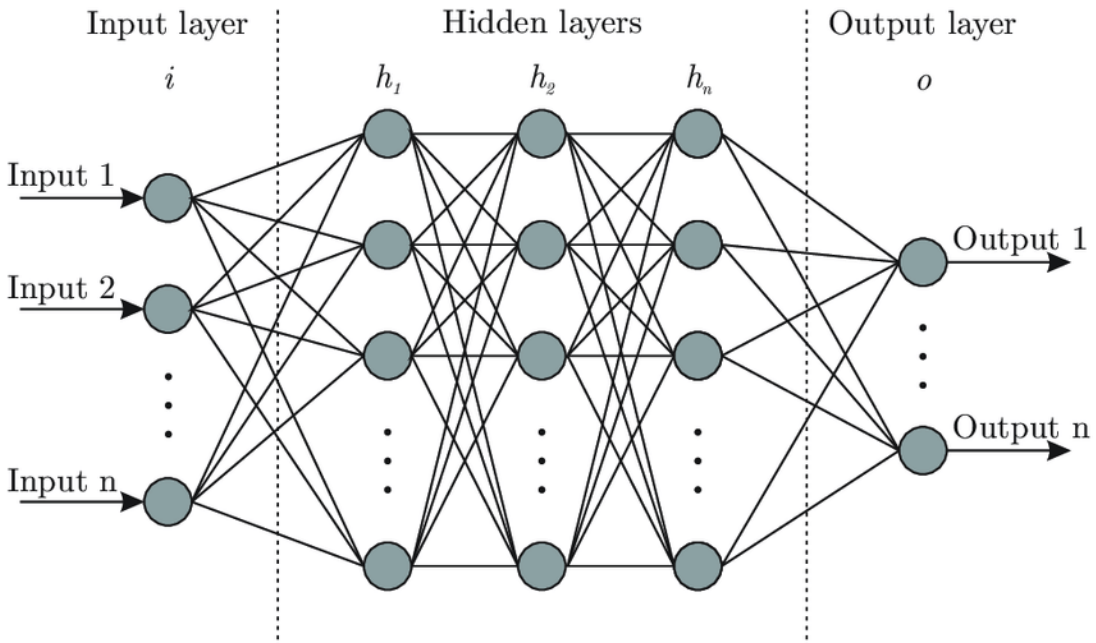
- The nodes of a hidden layer are connected to the nodes of the previous layer by links representing weights (w) and biases (b).
- Each node of a hidden layer is a non-linear function (activation function).
- The input of an activation function is a linear combination of the outputs of the previous layer and the weights and bias.





$$w^T x + b = w_1 x_1 + w_2 x_2 + \cdots w_n x_n + b$$

The Output Layer



- The nodes of the output are connected to the nodes of the last hidden layer.
- Each node is an activation (non-linear) function whose values correspond to a probability distribution.
- The input of the activation function is a linear combination of the outputs of the last hidden layer with weights and bias represented by the links.

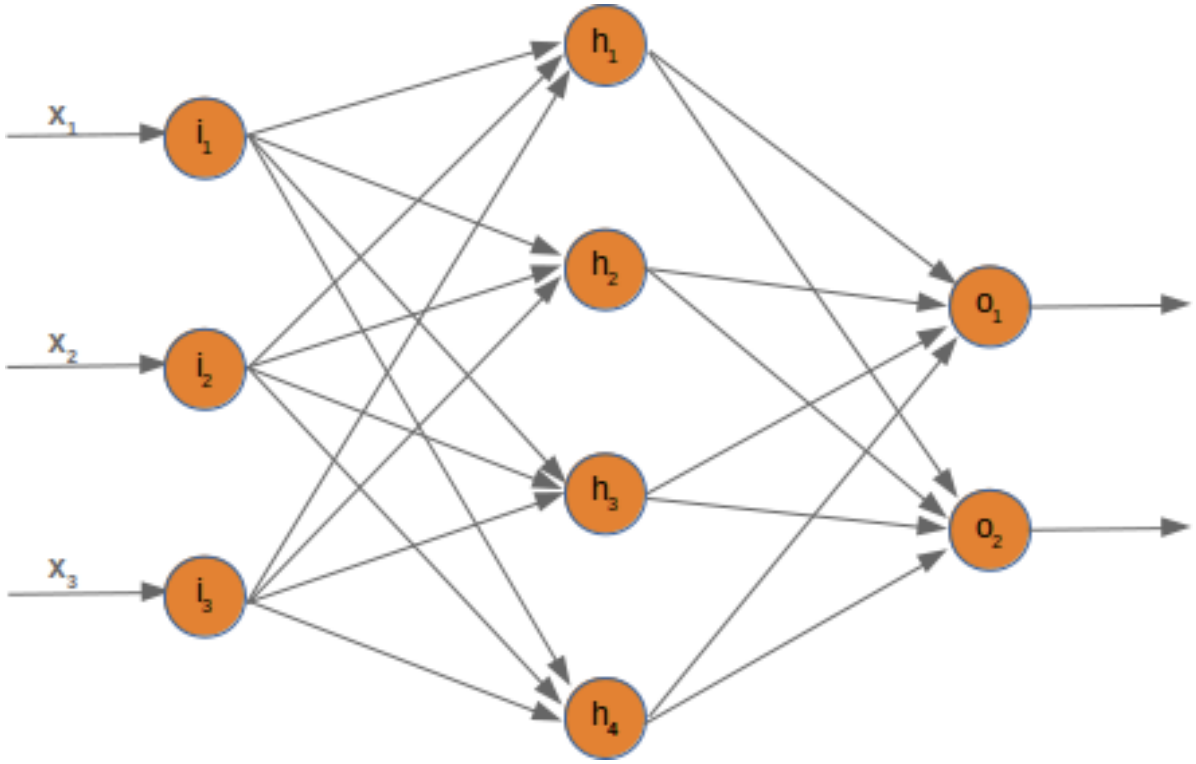
Example of a Neural Network

A neural network with:

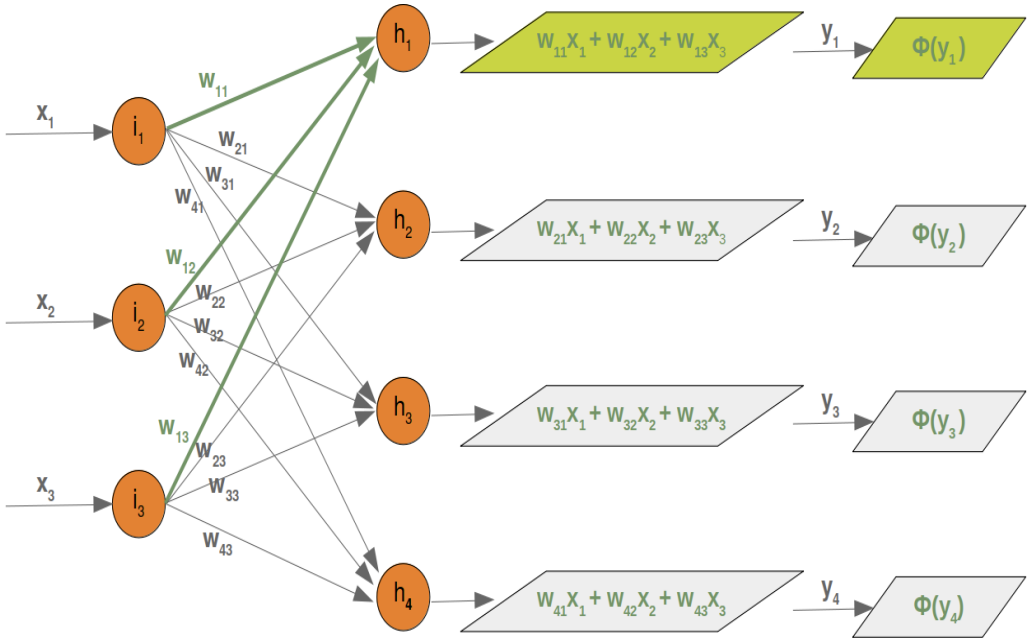
- 1 input layer (3 nodes)
- 1 hidden layer (4 nodes)
- 1 output layer (2 nodes)

1. How many weight parameters ?

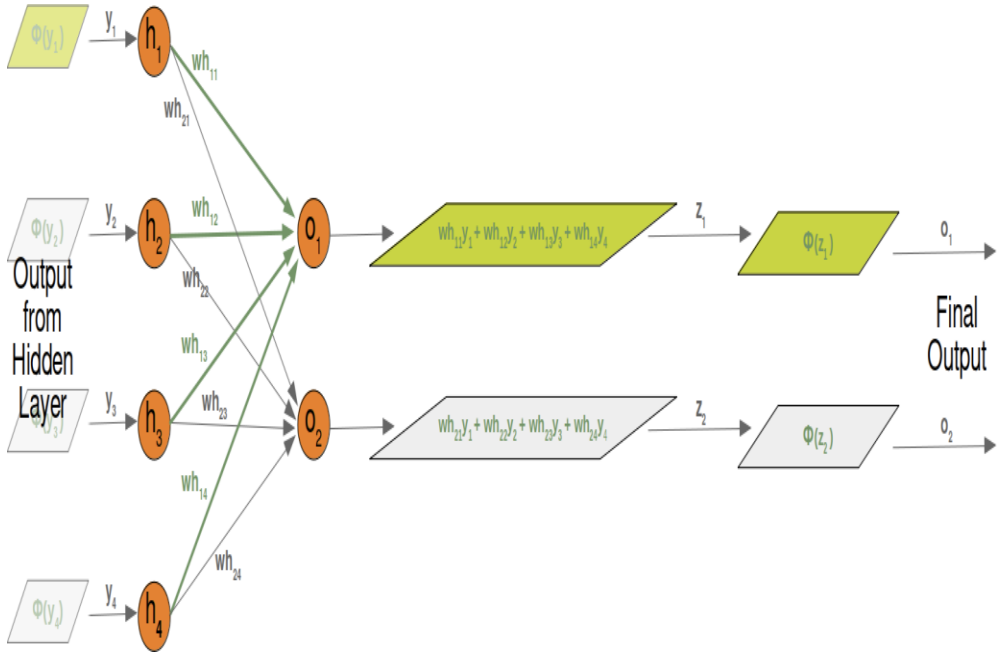
2. Output o_1 is a function of how many weight parameters?



Forward Pass: Input to Output



Input to hidden layer



Hidden to output layer

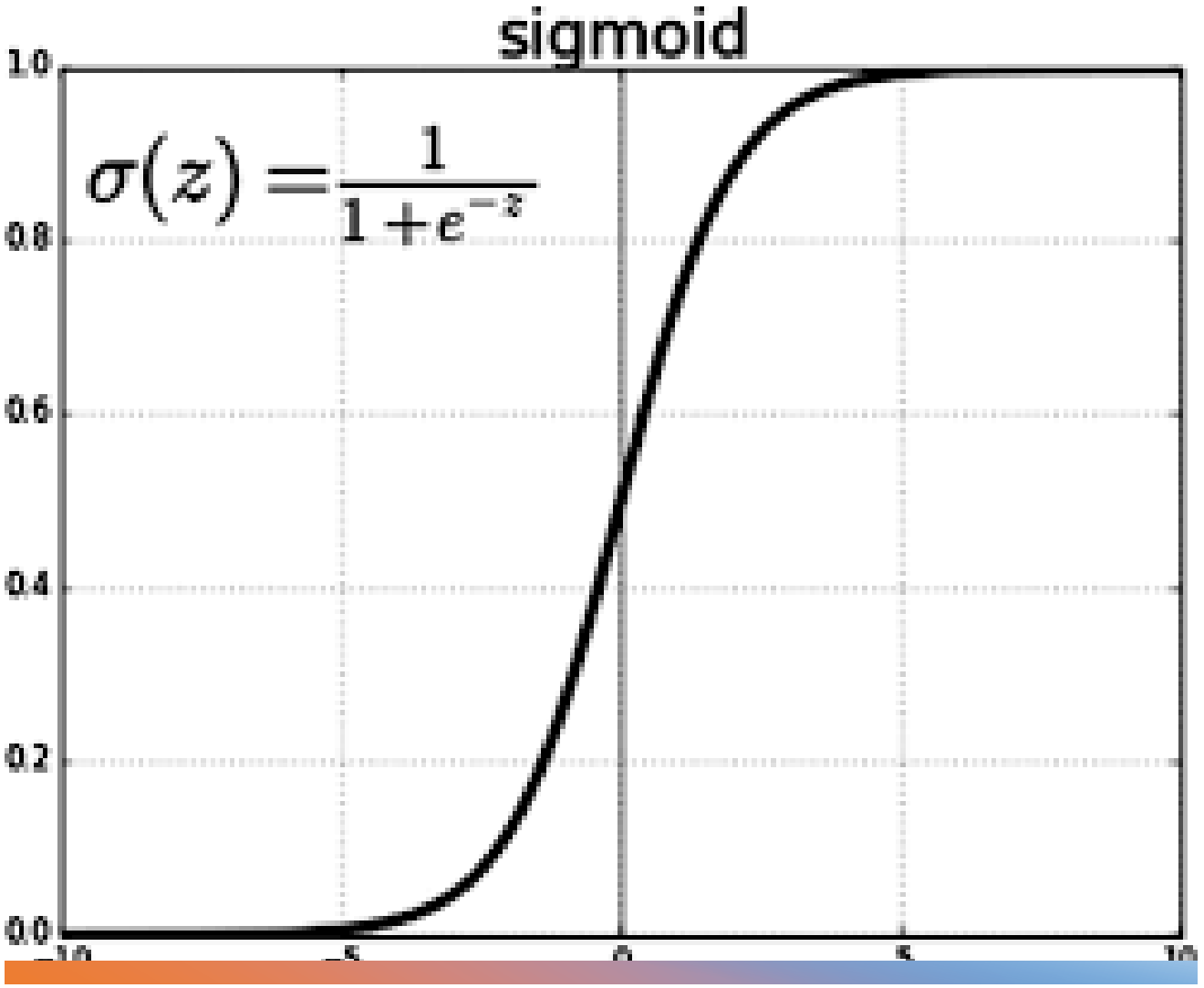
Activation Functions for the Hidden Layers

The Sigmoid (logistic activation function:

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

Derivative:

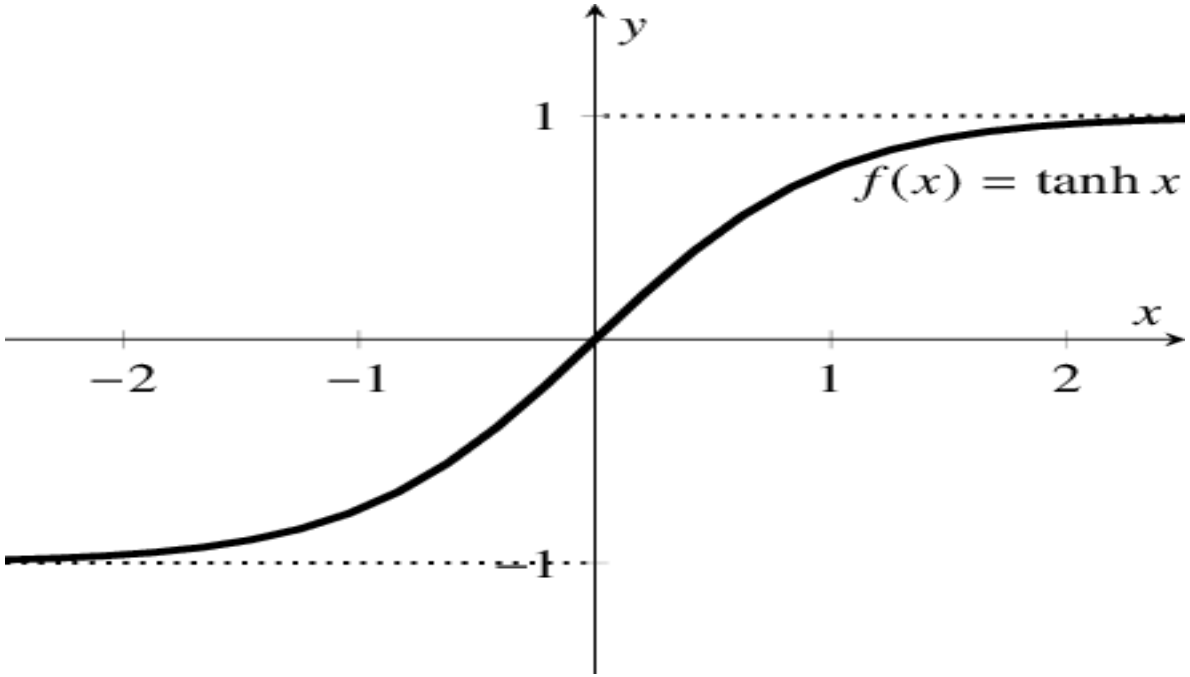
$\sigma'(x) = \sigma(x)(1 - \sigma(x))$



Activation Functions for the Hidden Layers

The Hyperbolic Tangent : $f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

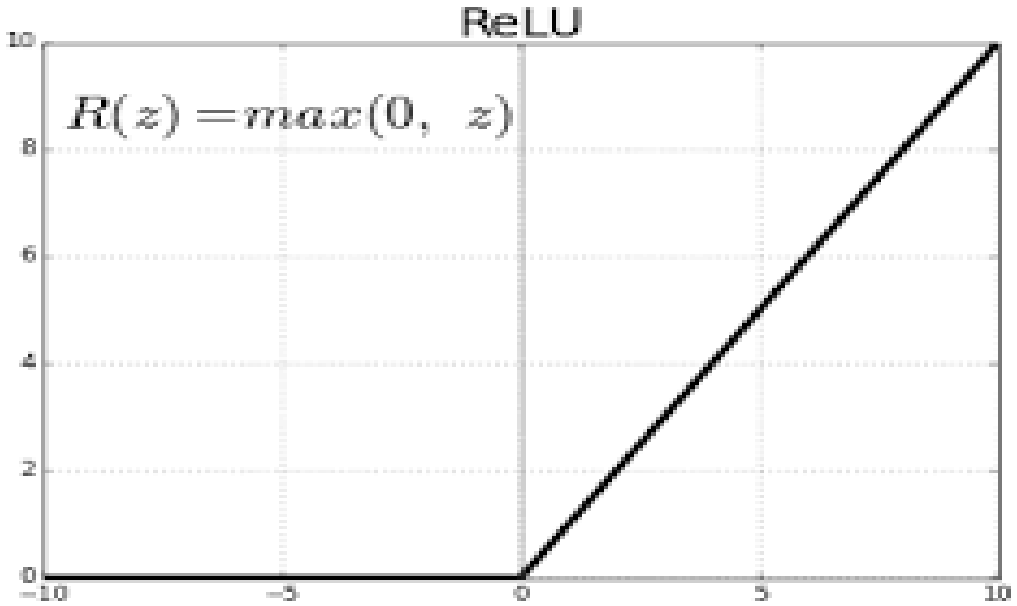
Derivative: $f'(x) = 1 - (f(x))^{2}$



Activation Functions for the Hidden Layers

The ReLU (Rectified Linear Unit): $\text{ReLU}(x) = \begin{cases} 0, & x < 0 \\ x, & x > 0 \end{cases} = max\{0, x\}$

Derivative: $\text{ReLU'}(x) = \begin{cases} 0, x < 0 \\ 1, x > 0 \end{cases}$



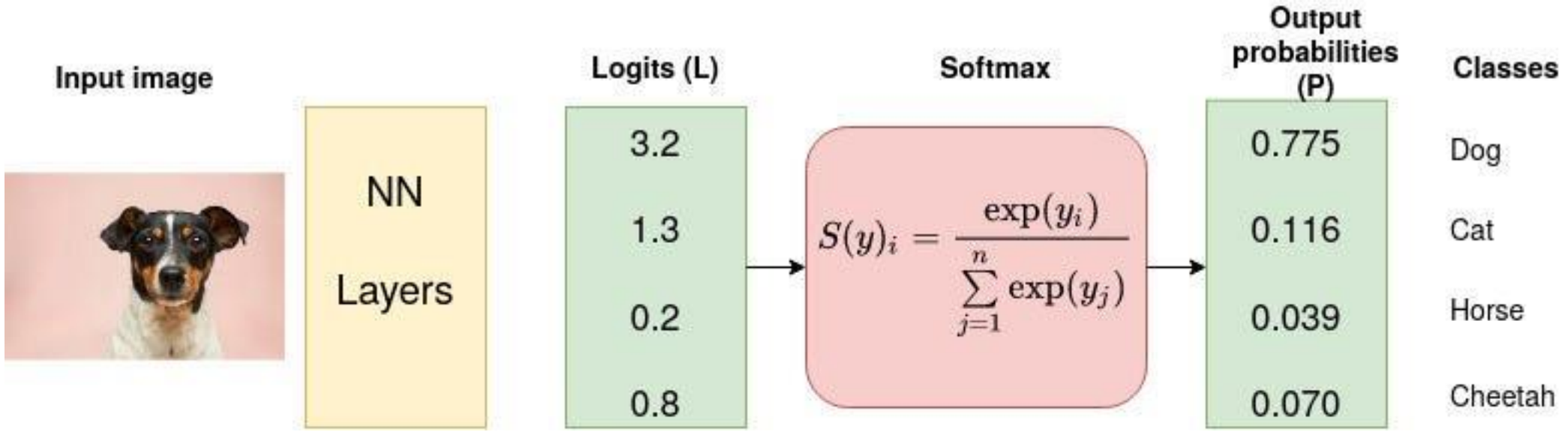
Activation Functions for the Output Layer

- For the output layer, the most common are the Sigmoid and the Softmax activation functions.
- The Sigmoid is used when there is a binary classification like: cat or dog.
- The Softmax is used when there is a multiclass classification like: cat, dog, horse, car, cow, person etc...
- Both give predictions on the output values by assigning a probability that an input belongs to one of a set of output classes.
- Both functions have range values in the interval (0, 1).

FORMULA: The Softmax takes an input vector $z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix}$ and outputs a vector of probabilities

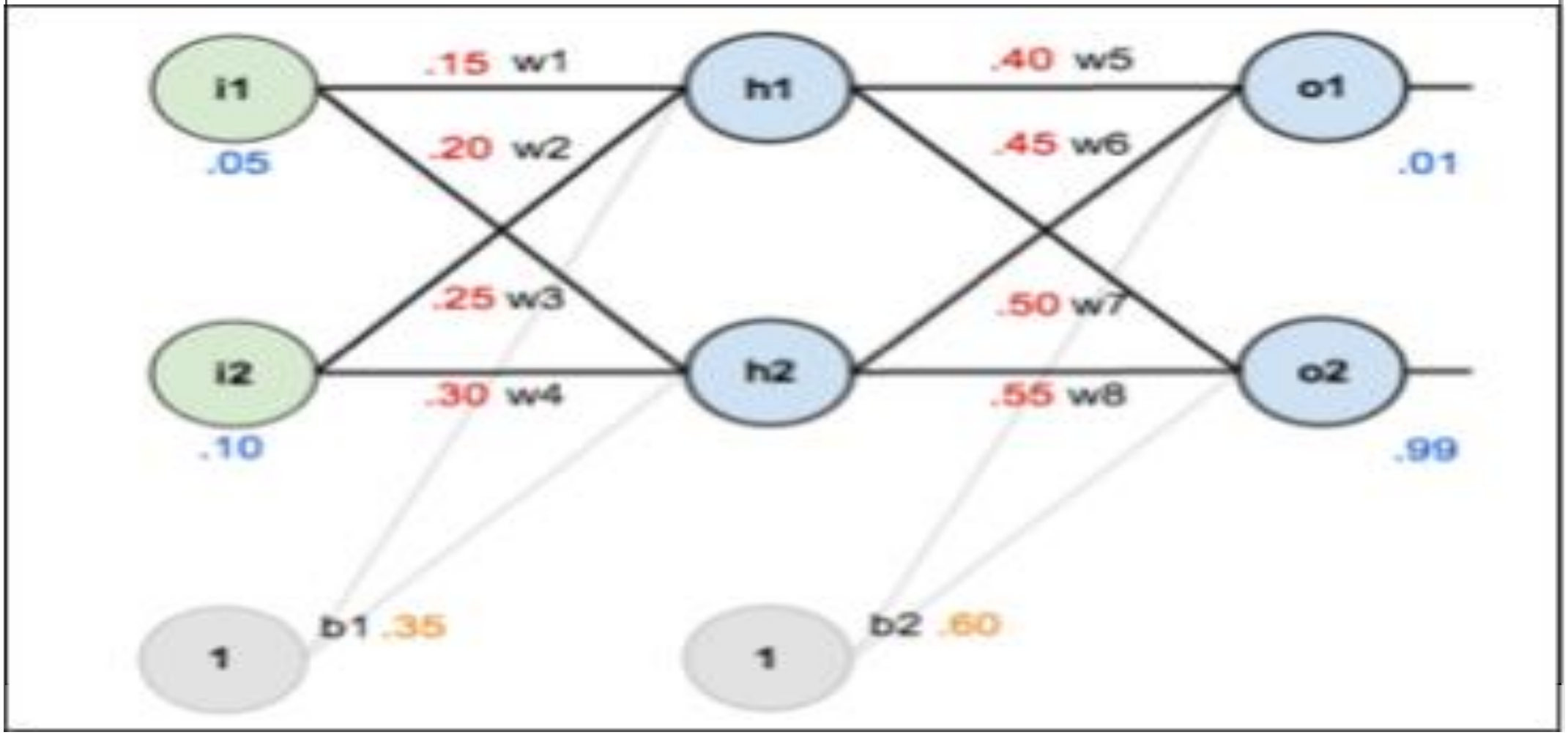
$\mathrm{S}(z) = \begin{bmatrix} S(z_1) \\ S(z_2) \\ \vdots \\ S(z_k) \end{bmatrix}$, where $S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

Softmax Example



DNN Example: forward Pass

Compute the outputs and the Mean Square Error for the given labels
Use activation function Sigmoid.



```
# e constant
e = 2.7182818284
# initial values
i1 = 0.05
i2 = 0.10
# initial weights
w1 = 0.15
w2 = 0.20
w3 = 0.25
w4 = 0.30
w5 = 0.40
w6 = 0.45
w7 = 0.50
w8 = 0.55
# bias
b1 = 0.35
b2 = 0.60
# targets
To1 = 0.01
To2 = 0.99
```

```
# forward propagation
h1 = 1/(1+e**(-(w1*i1 + w2*i2+b1)))
print("h1: " + str(h1))

h2 = 1/(1+e**(-(w3*i1 + w4*i2+b1)))
print("h2: " + str(h2))

o1 = 1/(1+e**(-(w5*h1 + w6*h2+b2)))
print("o1: " + str(o1))

o2 = 1/(1+e**(-(w7*h1 + w8*h2+b2)))
print("o2: " + str(o2))
```

h1: 0.5932699921052087 h2: 0.5968843782577157 o1: 0.751365069547507 6
o2: 0.772928465316421

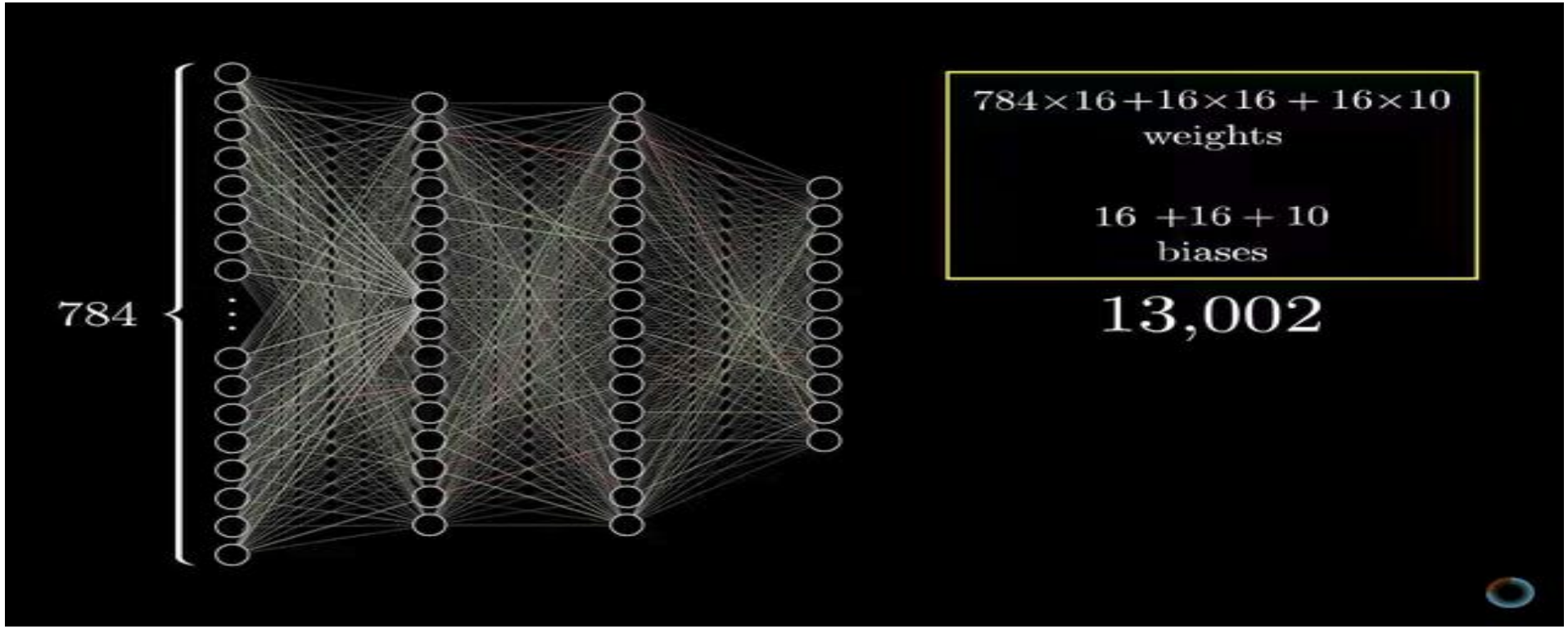
```
    # Error
    Eo1 = 0.5*(To1-o1)**2
    print("Error o1: " + str(Eo1))
    Eo2 = 0.5*(To2-o2)**2
    print("Error o2: " + str(Eo2))
    E = Eo1 + Eo2
    print("Total Error: " + str(E))
```

Error o1: 0.2748110831725904 Error o2: 0.023560025584942117
Total Error: 0.2983711087575325 3

Weights and Biases

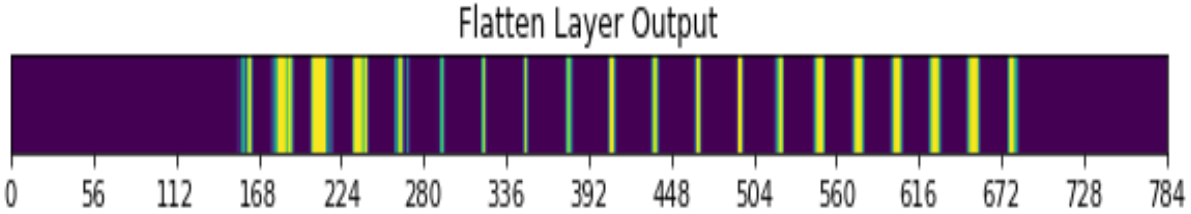
- The weights and biases are the parameters that need to be optimized before implementing the model.

- This model is used to read and classify images of digits: 0,1,2,….,9.

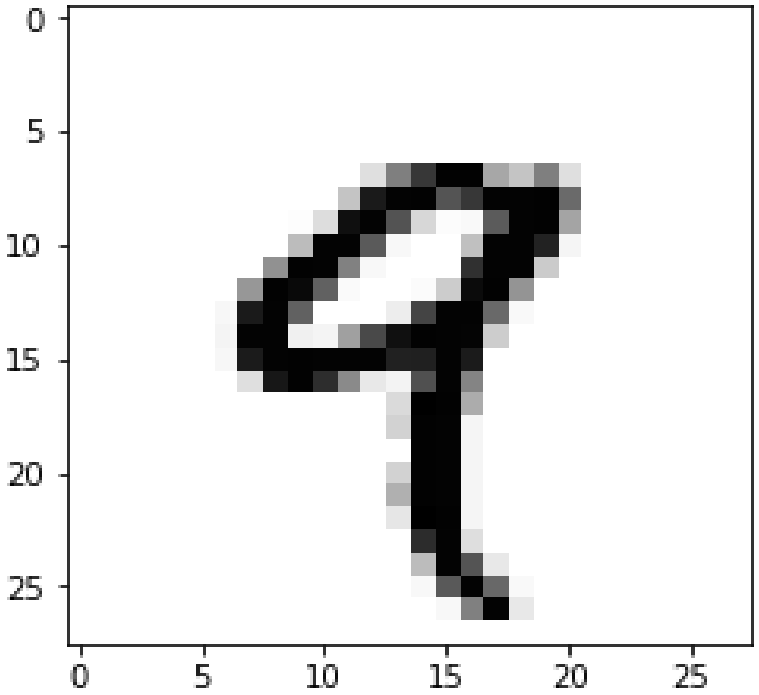


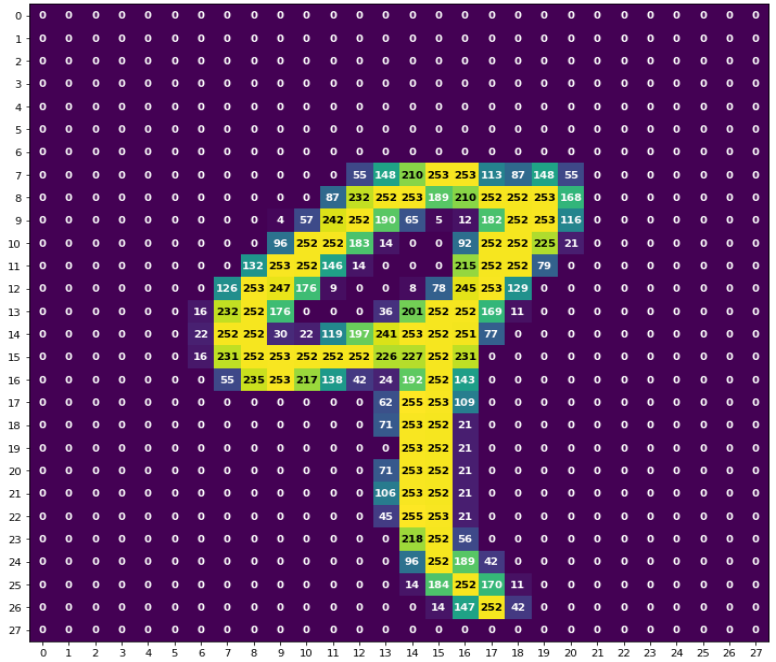
How computers see images

Pixelized image

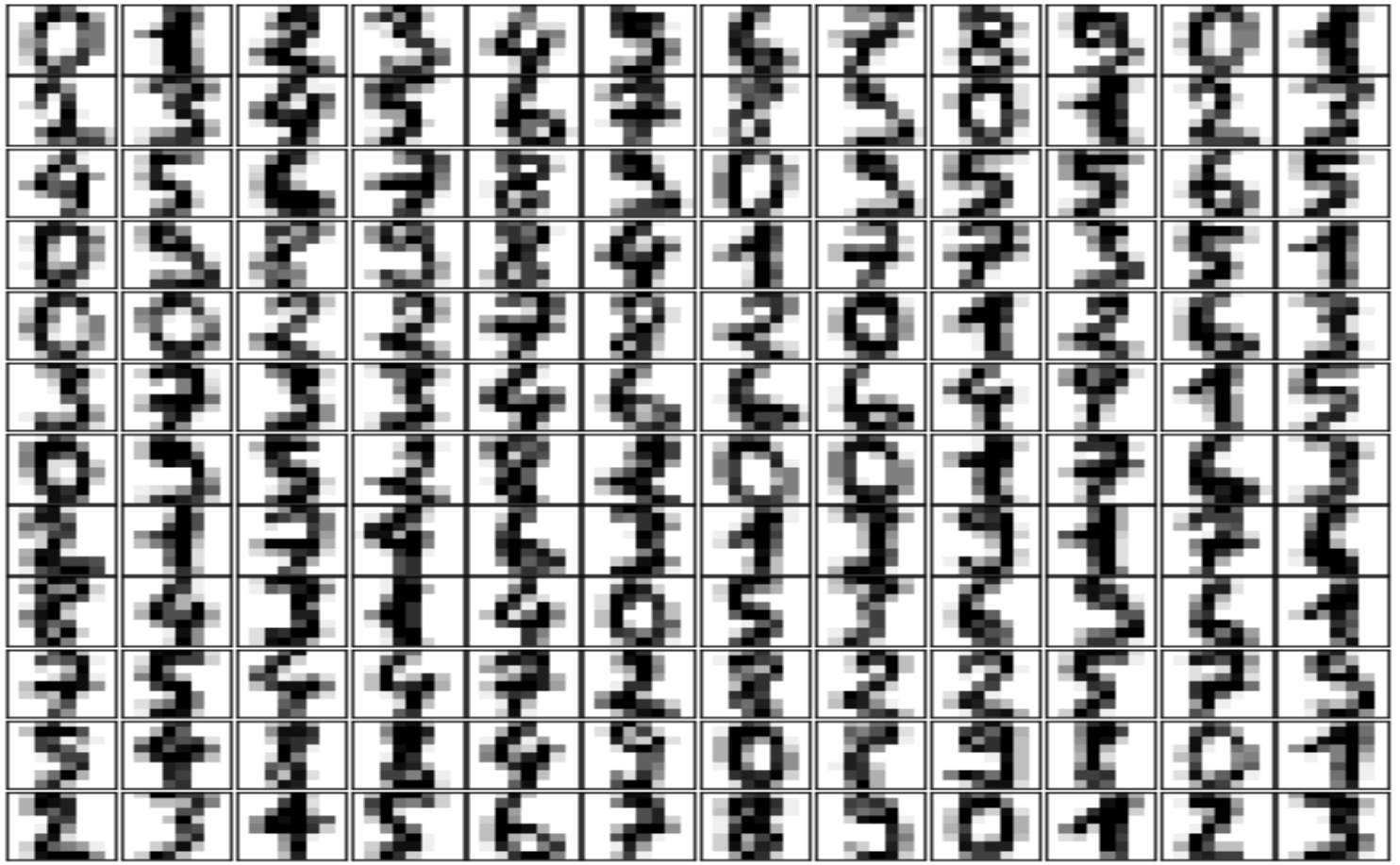


Grayscale image of 28x28 pixels



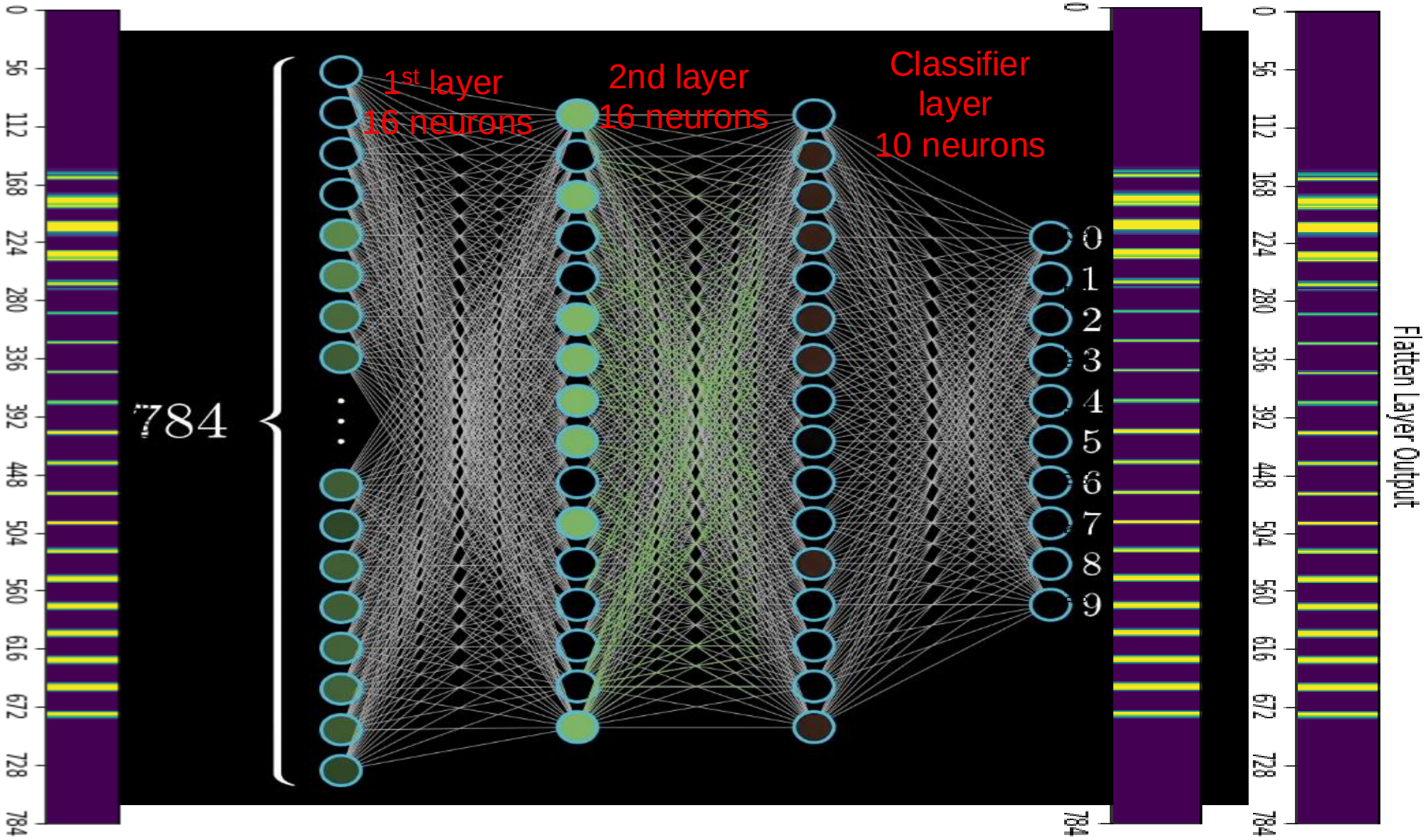


MNIST Data Sets



MNIST has 60,000 images of digits (0 through 9) primarily used to train DNN and 10,000 images to test the DNN model.

Simple MNIST MLP NN



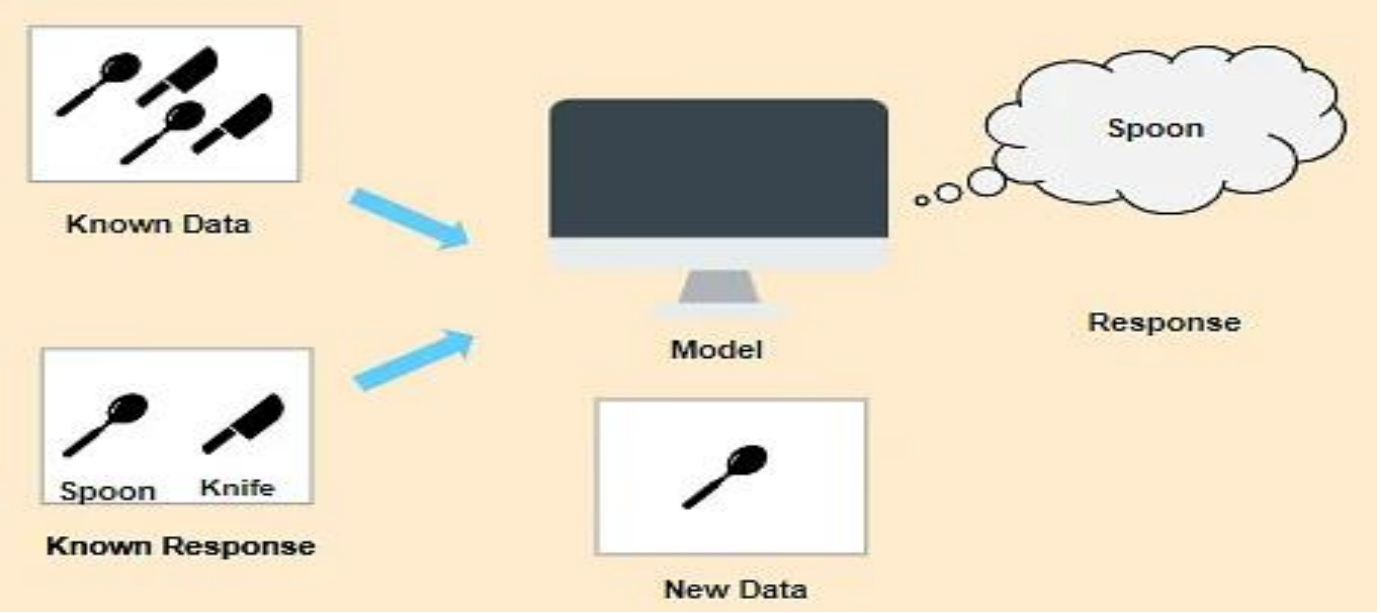
How a DNN Learns

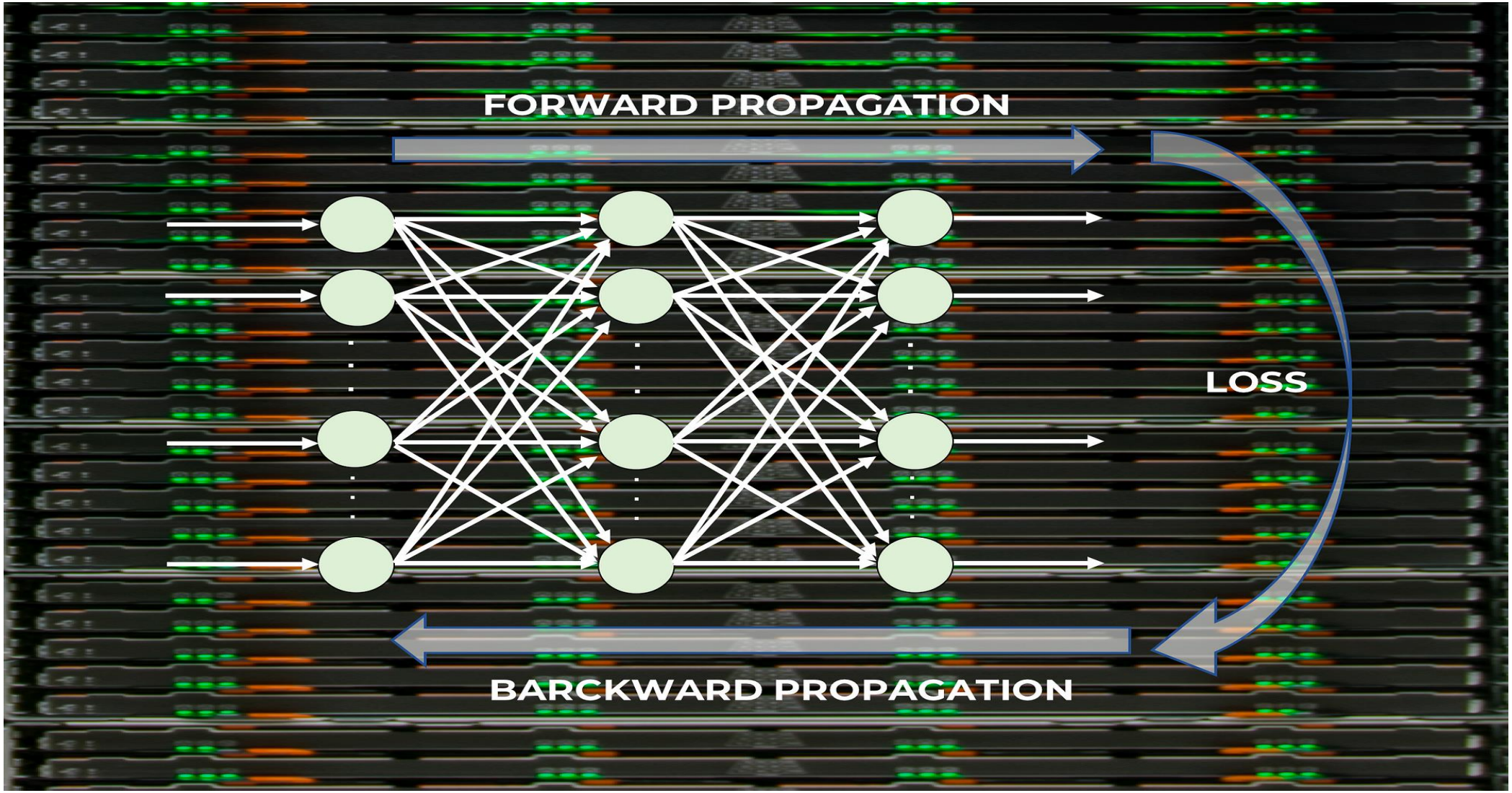
- A neural network learns to perform a task by building knowledge from training data sets so that when new data comes in, they are able to make predictions/decisions with a certain degree of accuracy based on past (learned) data.

- Modes of learning: Supervised, Unsupervised, and Reinforcement.

Supervised Learning

- Learning approach that utilizes labeled data sets to train the model to recognize unseen data.
- Learn by minimizing the loss/cost function which measures the difference between the correct outcome and the prediction.
- Supervised Learning is used mainly for classification problems and regression problems.





Supervised learning: Learns through corrective feedback loops to improve predictions by adjusting weights and biases iteratively.

Minimize the loss through BACK PROPAGATION.

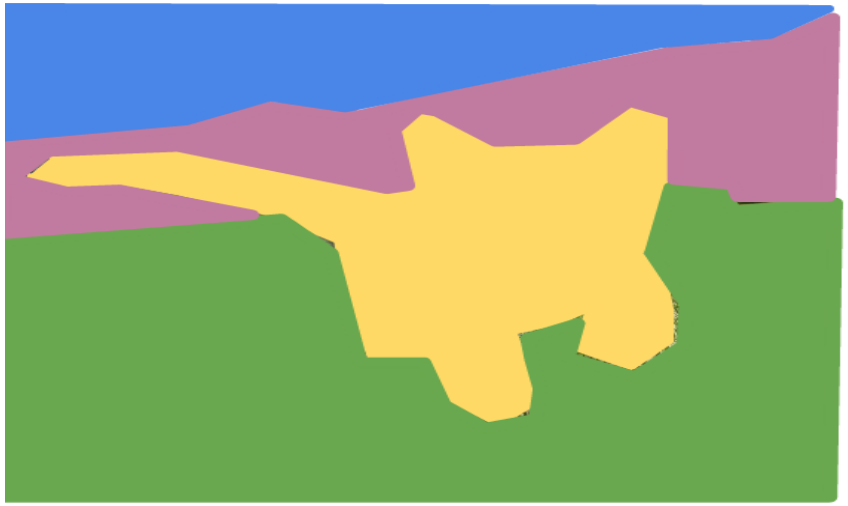
Supervised Learning Data:
(x, y) x is data, y is label

Goal:
Learn a function to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

CAT
classification

DOG, DOG, CAT
Object Detection



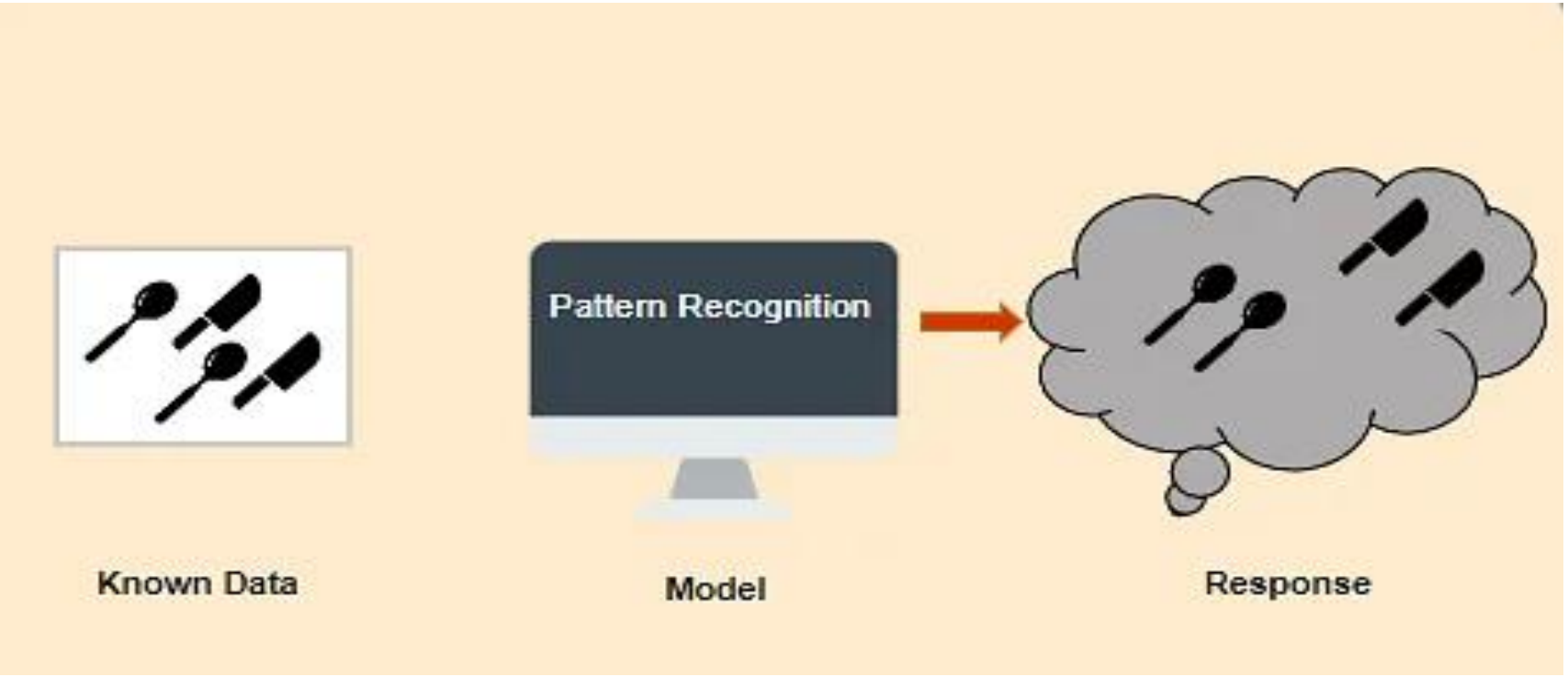
GRASS, CAT,
TREE, SKY

Semantic
Segmenattion

Unsupervised Learning

- Learning approach that utilizes data sets that contain no labels (with no explicit instructions) to train the model to find a structure in the input data on its own.

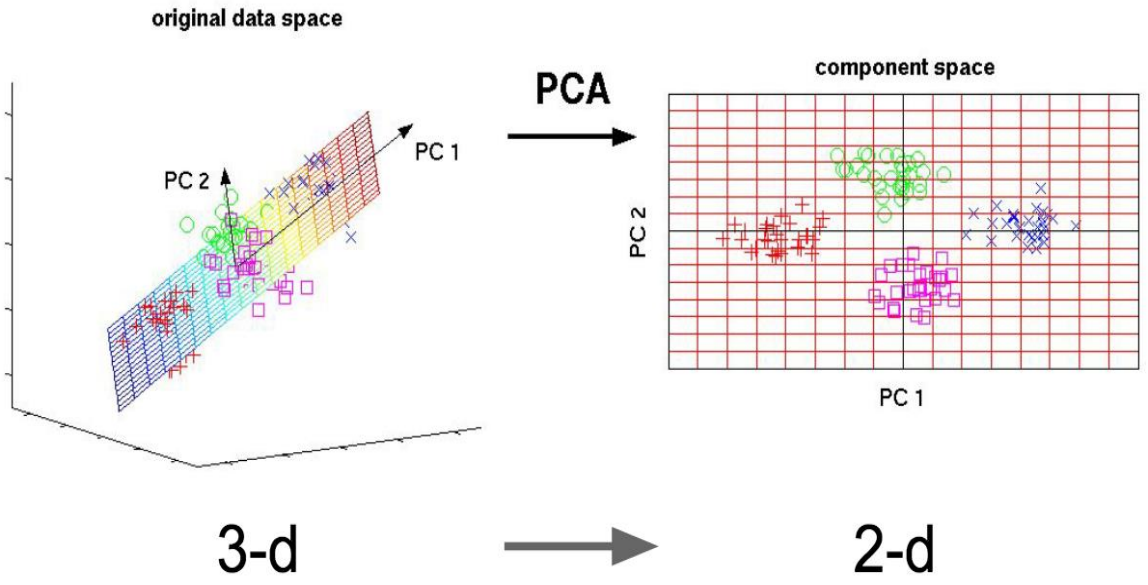
- Unsupervised Learning is used mainly for Clustering, association (NLP), density of distribution problems.

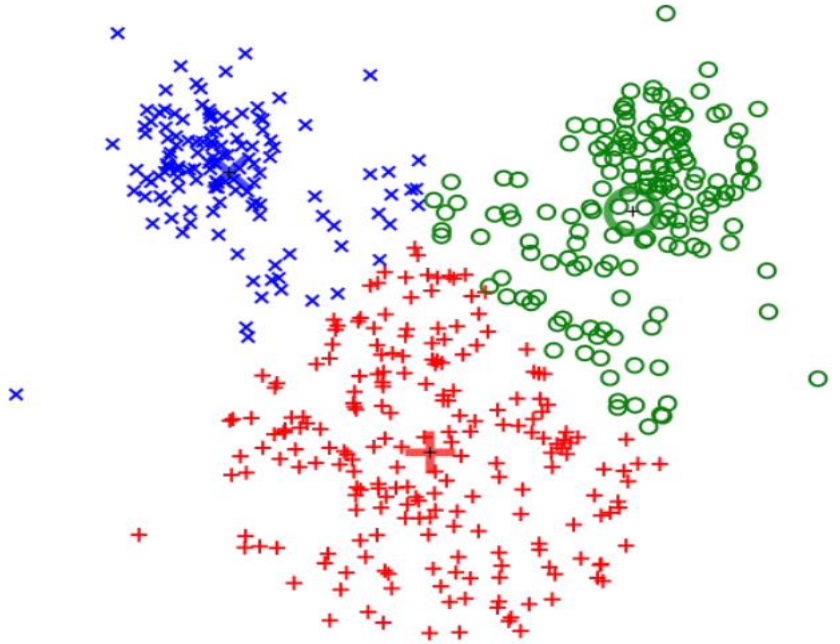


Unsupervised Learning Data:
x Just data, no labels!

Goal:
Learn some underlying hidden structure of the data

Examples: Clustering, dimensionality reduction, density estimation, etc.



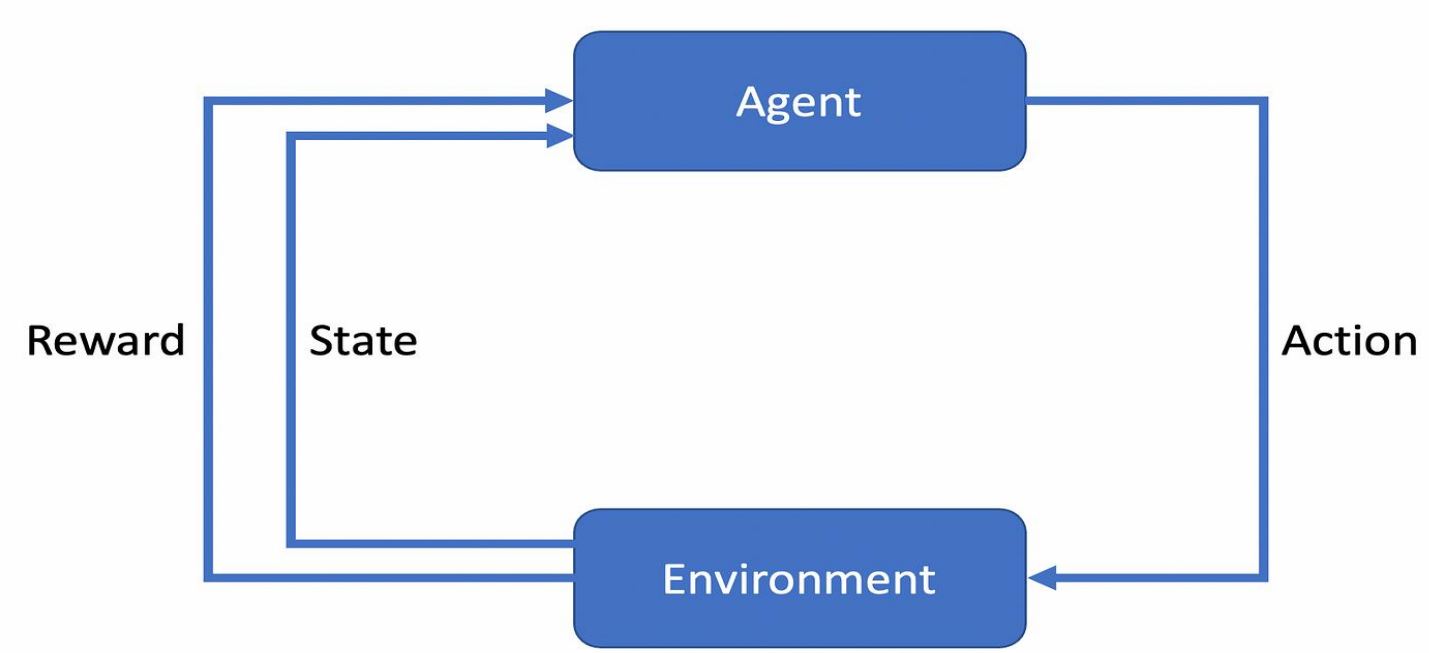


K-means clustering

Principal Component Analysis
(Dimensionality reduction)

Reinforcement Learning

- Learning approach through reward and punishment feedback for the environment to take the correct action/ make the correct decision.



Loss Function and Back Propagation in Supervised Learning

- A Loss/Cost function is needed for evaluating how well the learning algorithm models featured data sets.
- Cost functions measure how good is the model in predicting the expected outcome.

- Loss/Cost functions are generally grouped in two types:

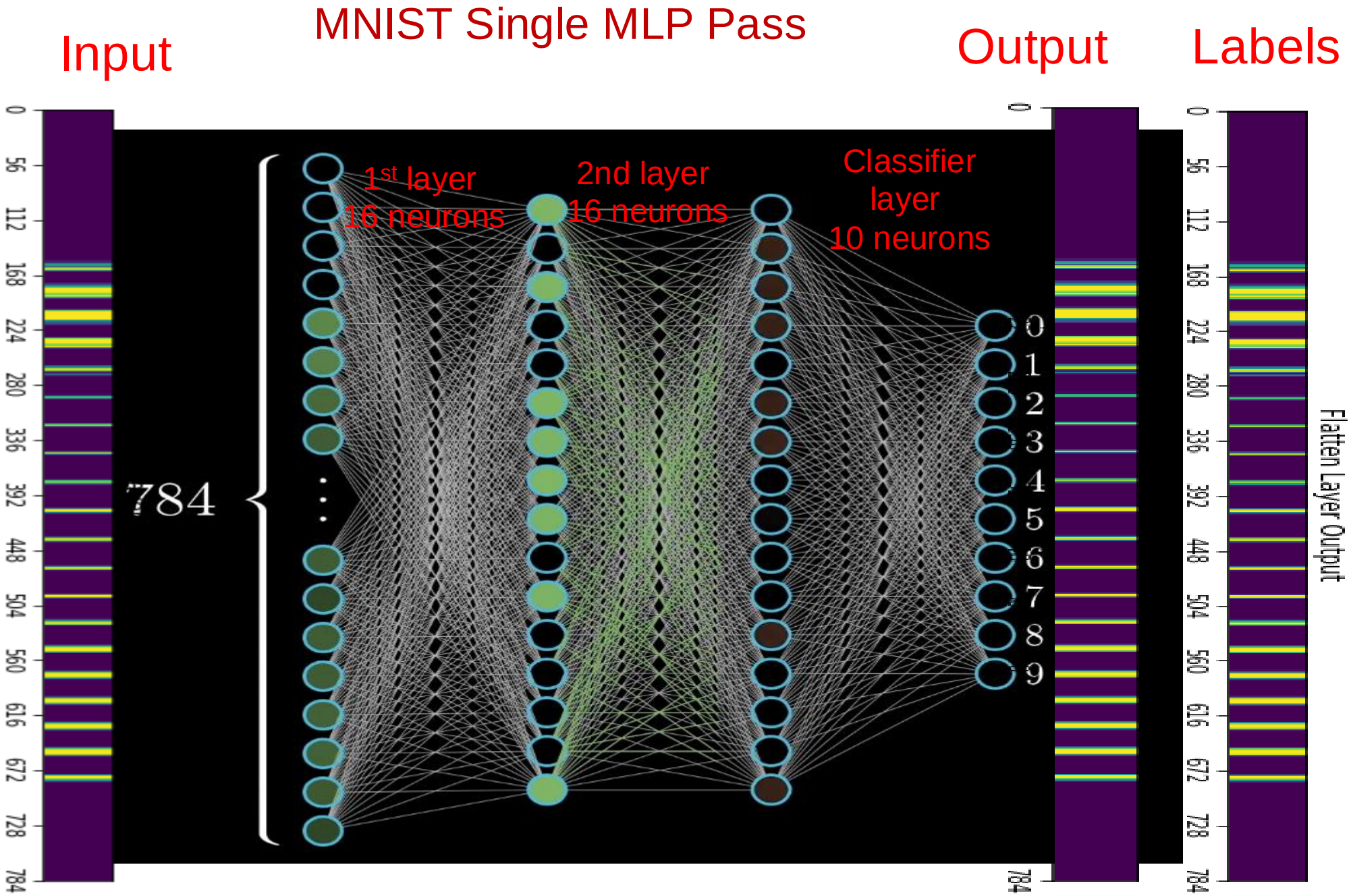
Regression Problems: the loss/cost function is the Mean Square Error function

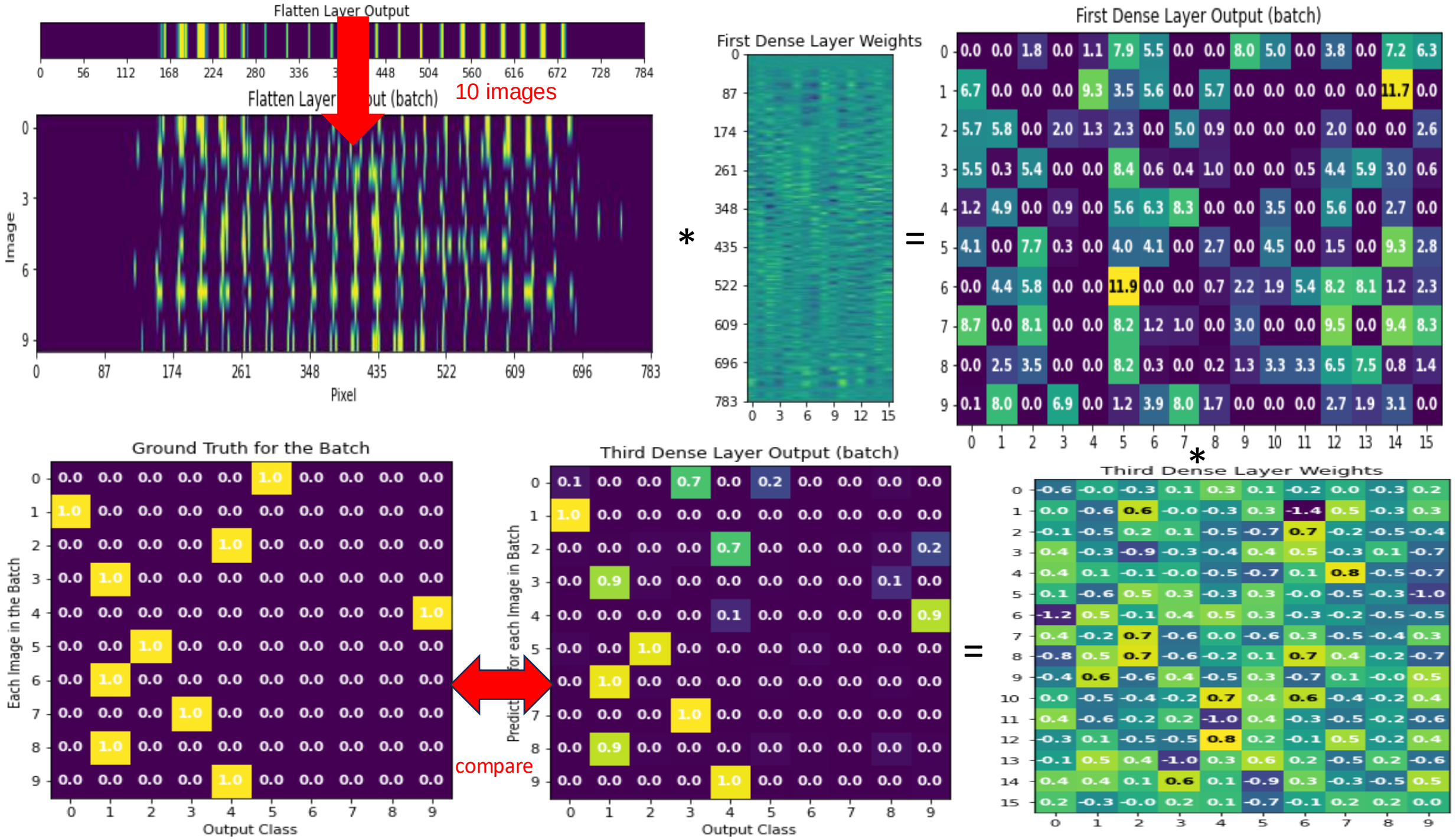
$L = MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}\text{-}y_i)^2$ (outputs are continuous real values)

Classification Problems : the common loss/cost function Categorical Cross Entropy function

$L = -\sum_{i=1}^{n} y_i \cdot \ln(\hat{y}_i)$. (outputs are probability values)

MNIST Single MLP Pass




Flatten Layer Output
Flatten Layer Output (batch)
10 images
Image
Pixel
First Dense Layer Weights
First Dense Layer Output (batch)
Ground Truth for the Batch
Each Image in the Batch
Output Class
compare
Third Dense Layer Output (batch)
Predict for each Image in Batch
Output Class
Third Dense Layer Weights

Minimize the Loss Function

- The Loss Function $L(w; b)$ is a (a.e.) differentiable function of all the weight and bias parameters.

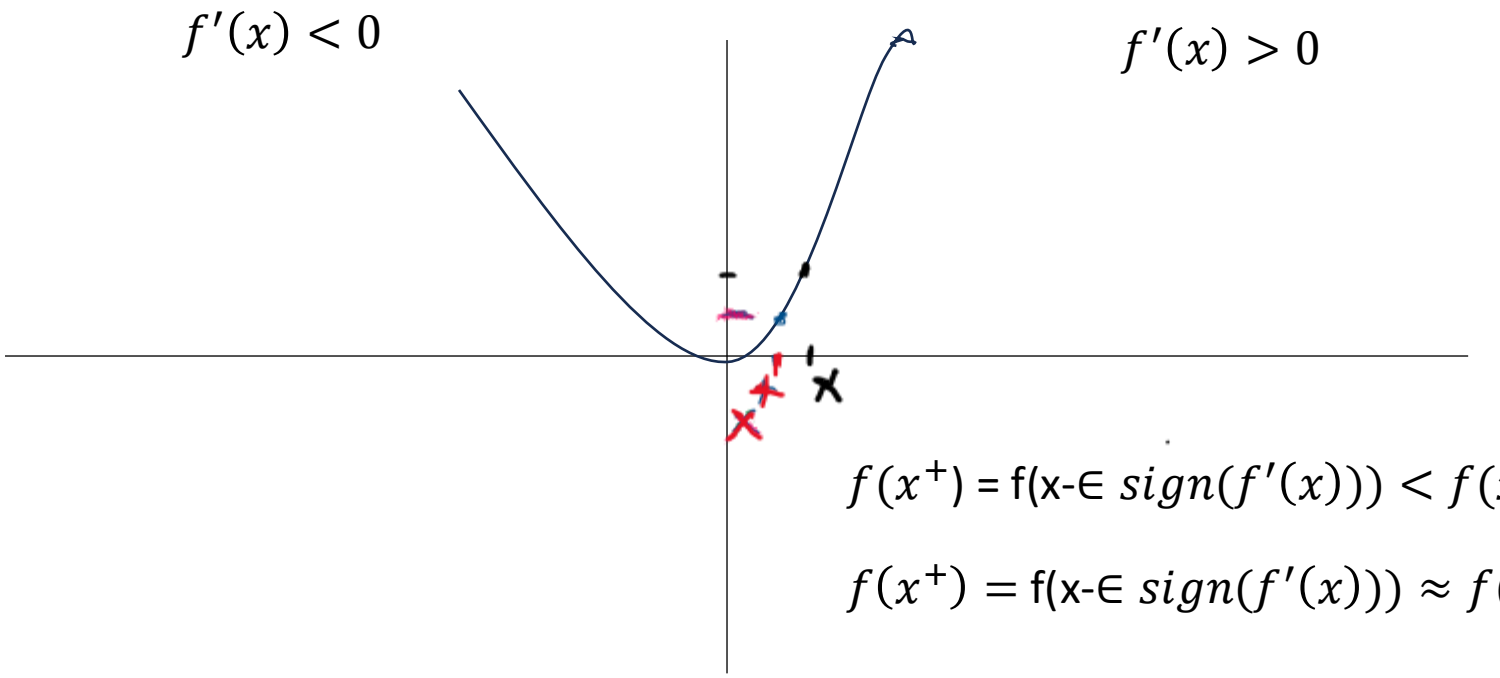
- We seek to optimize it by finding a suitable (local) minimum using the method of gradient descent.

- The gradient $\nabla L(w; b)$ is a vector of partial derivatives of the Loss Function with respect to all the weight and bias parameters.
- If $(w^+;\ b^+) = (w; b) - \in \nabla L(w; b)$ then $L(w^+; b^+) < L(w; b)$

$\in > 0$ (small) is called the learning rate.

Gradient and gradient descent

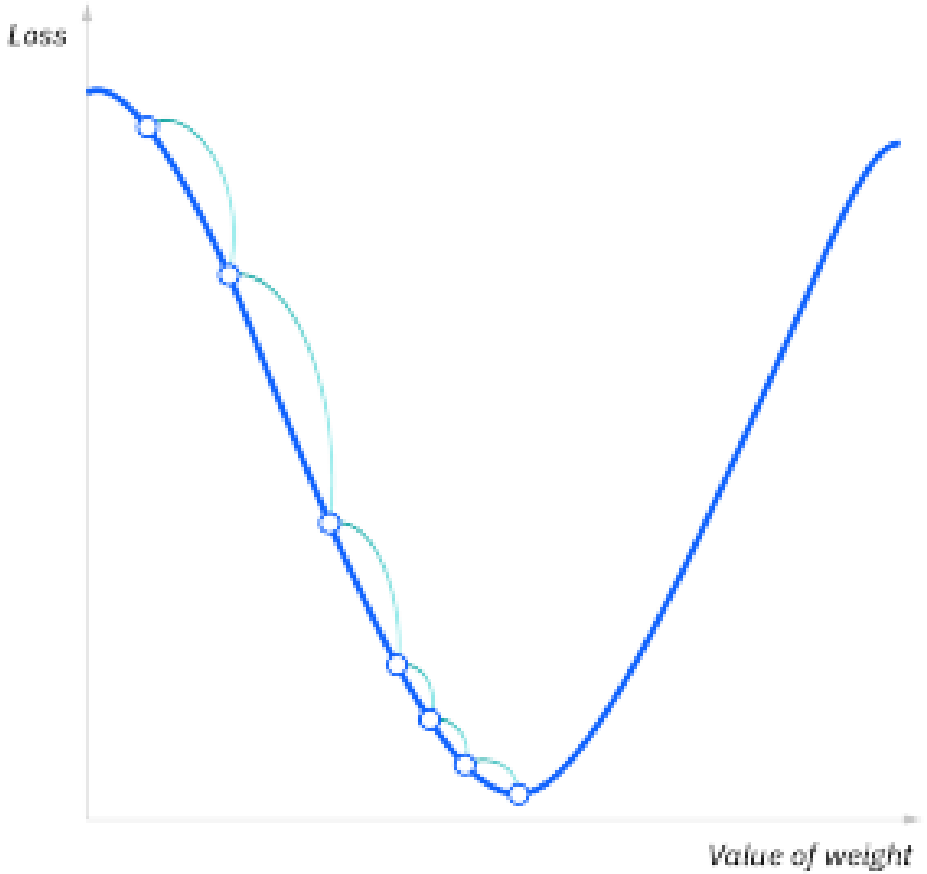
The gradient of a function $y = f(x)$ (single variable) is its derivative $f'(x)$.



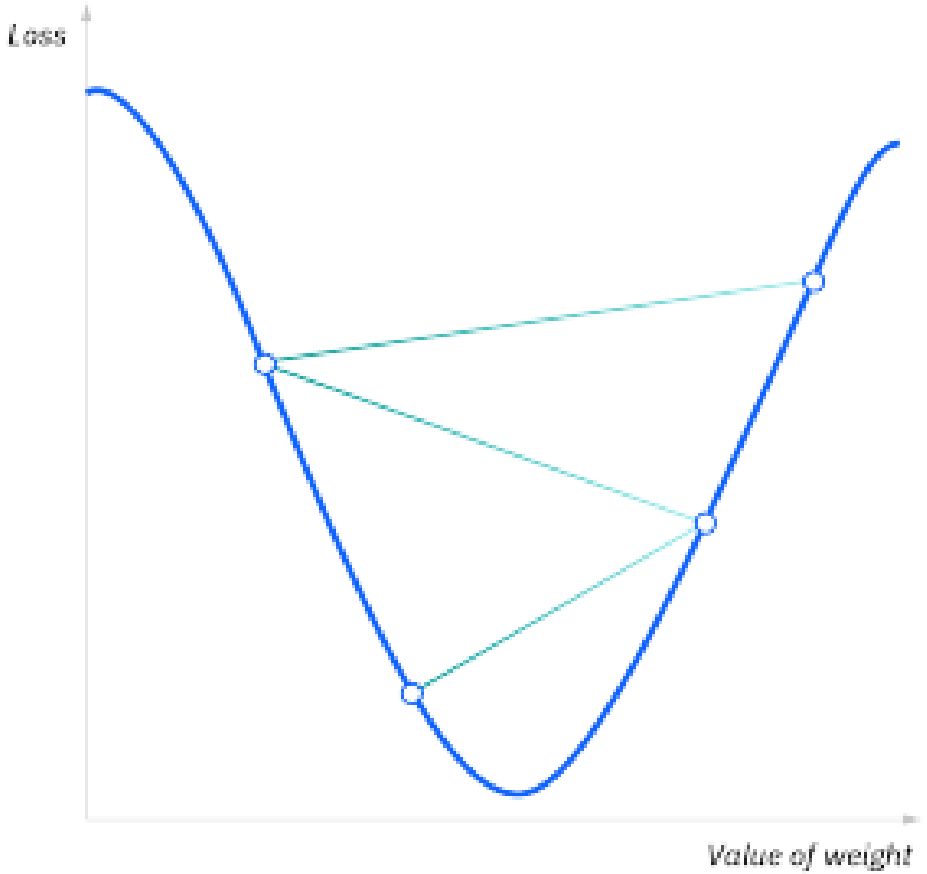
$f(x^+) = f(x - \epsilon\, sign(f'(x))) < f(x)$ For small $\epsilon > 0$

$f(x^+) = f(x - \epsilon\, sign(f'(x))) \approx f(x) - \epsilon\, |f'(x)|$

Small learning rate



Large learning rate



Gradient Descent - multivariable case

For a real-valued function $z = f(x_1, x_2, \dots, x_n)$ of 2 or more variables, its gradient derivative is a vector of partial derivatives:

$\nabla f(x_1, x_2, \dots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}\right)$

Example: $z = F(x, y) = x^2 + y^2;\ F(1,1) = (1)^2 + (1)^2 = 2$

$$\nabla z = \nabla F(x, y) = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}\right) = (2x, 2y);\ \nabla F(1,1) = (2,2)$$

For $\in = 0.25$,

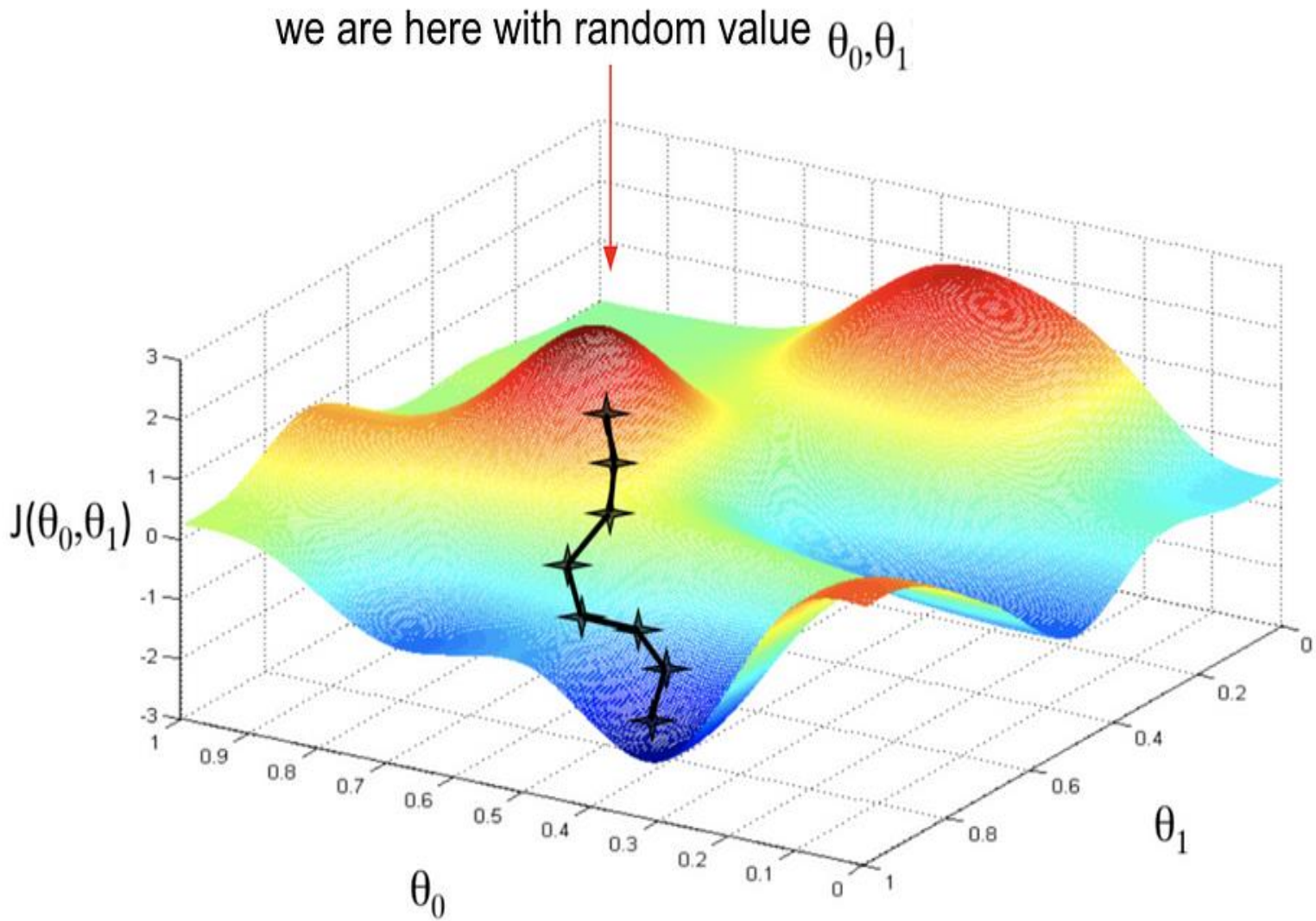
$$(x^+, y^+) = (1,1) - 0.25(2,2) = (0.5,0.5)$$

Then $F(x^+, y^+) = F(0.5,0.5) = (0.5)^2 + (0.5)^2 = 0.5 < 2 = F(x, y)$

Training Neural Networks

- Algorithm: Find the weights and biases to achieve the lowest loss

1. Initialize the weights and biases $(w; b) \sim N(0,1)$
2. Loop until convergence:
3. Compute the gradients of the Loss $\nabla L(w; b) = \frac{\partial L}{\partial (w; b)}$
4. Update the weights and biases: $(w^+; b^+) = (w; b) - \in \nabla L(w; b)$
5. Return weights and biases



- Start with some θ_0,θ_1
- Keep changing θ_0,θ_1 to reduce $J(\theta_0,\theta_1)$

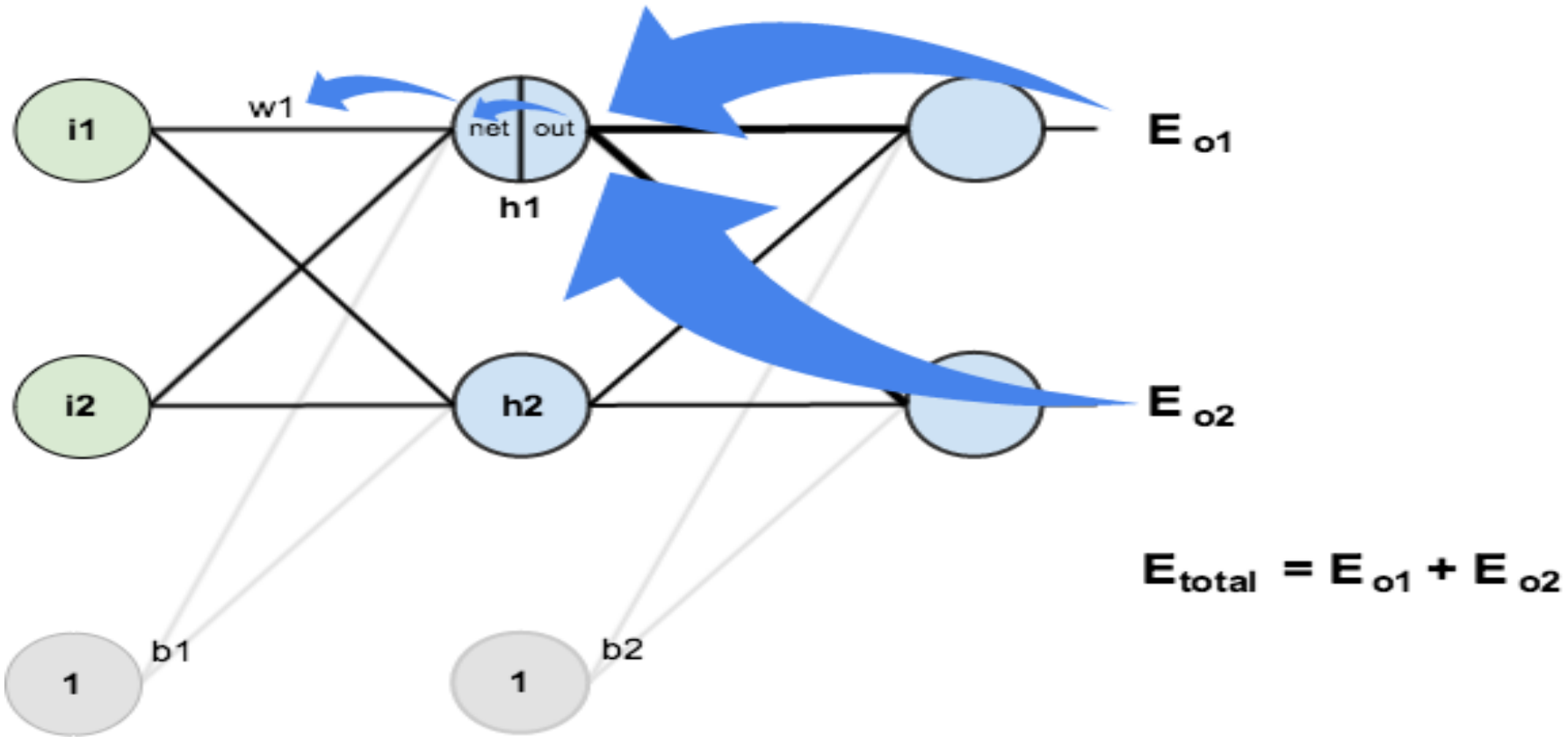
until we hopefully end up at a minimum

Back Propagation to Compute gradients

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

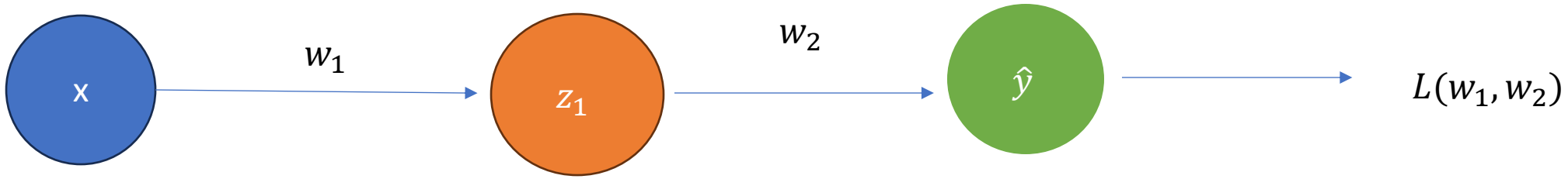


$$E_{total} = E_{o1} + E_{o2}$$

A Simple Example

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$



X=0.3 W_1=0.5 Sigmoid W_2=0.2 Sigmoid Label=true value = 0.9

Assume learning rate =$\in$= 0.1

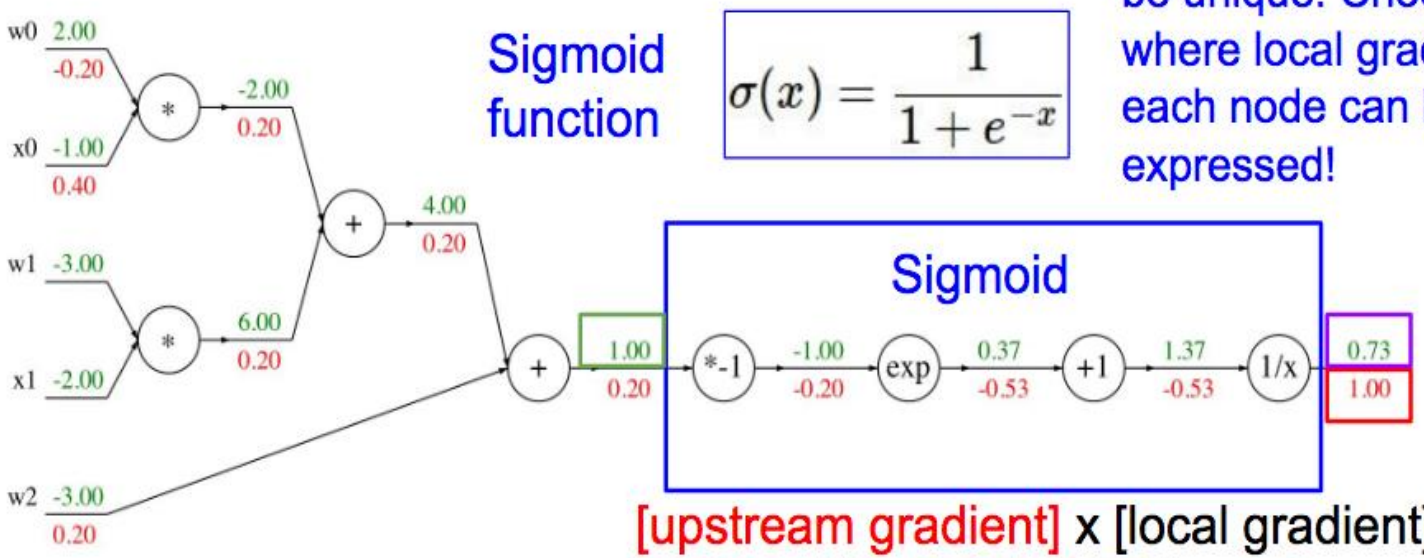
Loss function = MSE

$$(w_1^+, w_2^+) = (w_1, w_2) - \in \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}\right)$$

Computational Graph

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



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

[upstream gradient] x [local gradient]

[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient: $$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Optimization of the Loss/Cost Function

- Loss Functions can be difficult to optimize

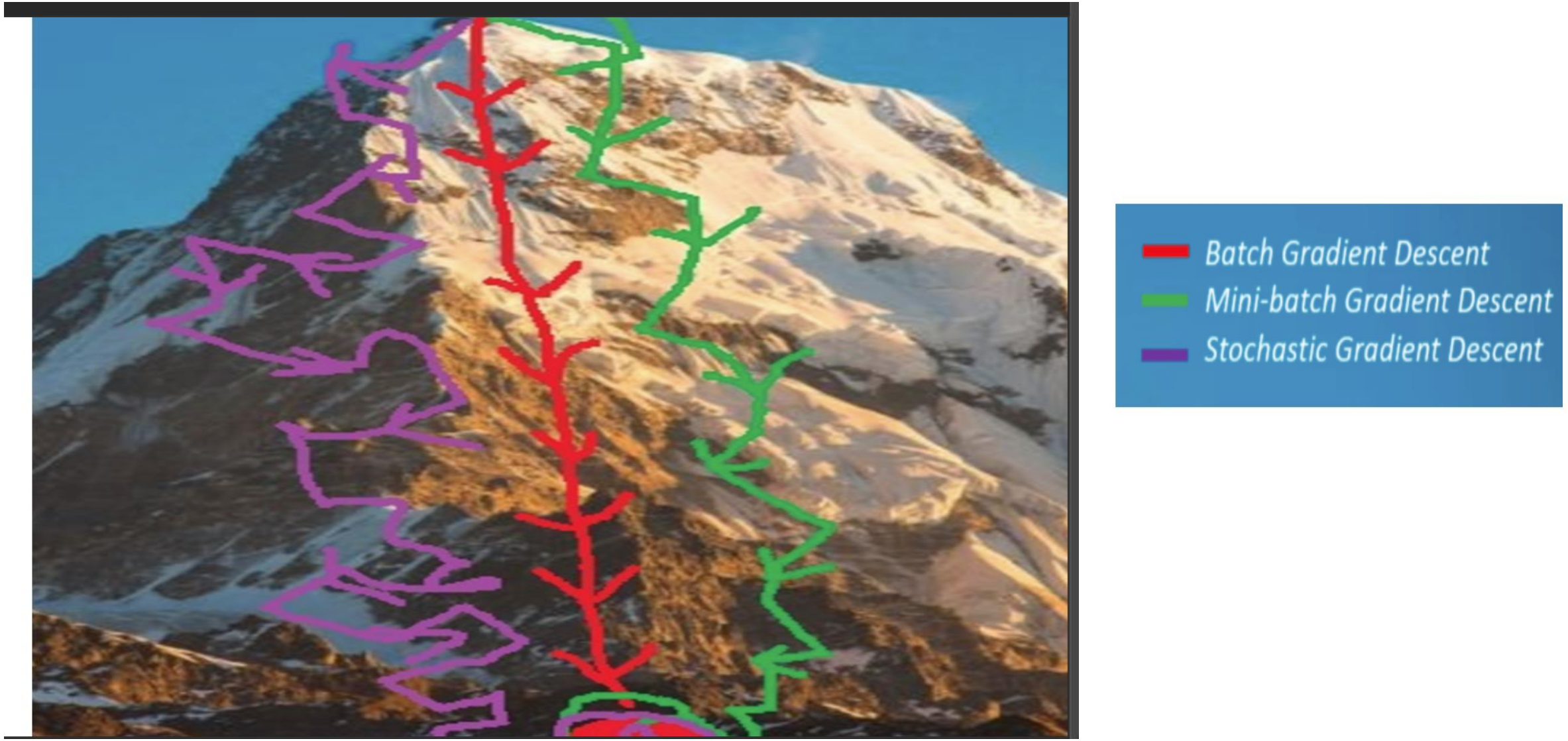
1. Small learning rate causes slow convergence and gets stuck at false local minimum (in flat regions)
2. Large learning rate causes to overshoot local minimum and creates divergence.
3. Best to use Adaptive Learning Rates: learning rates are not fixed; they can be made larger or smaller depending on:
    A. How large the gradient is
    B. How fast learning is happening
    C. Size of particular weights/biases
    D etc…

- Optimization is done through gradient descent

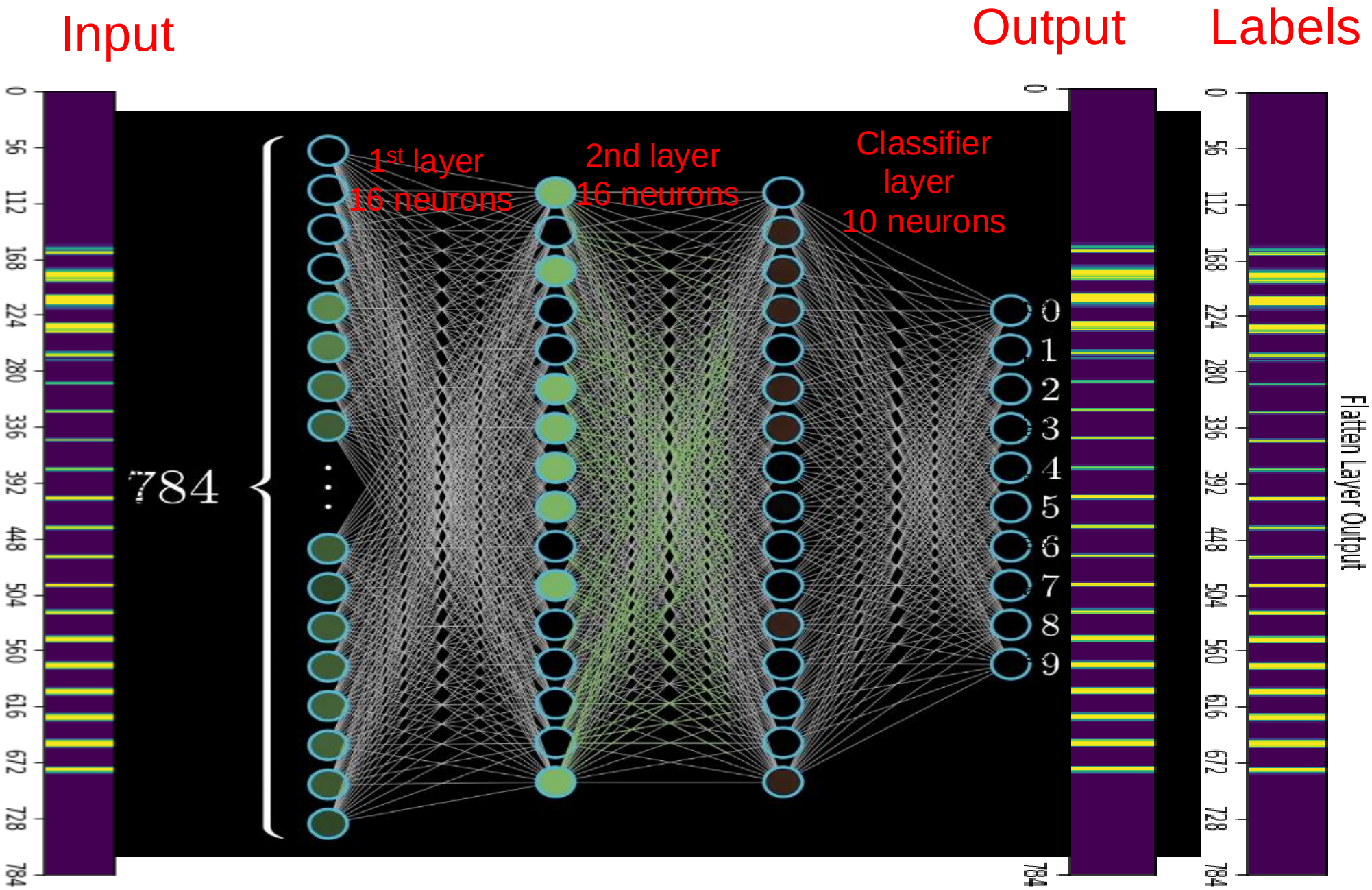
Optimization Schemes

- Batch optimization: (slow, ineffective) requires gradients for the whole training data set
- Stochastic Gradient Descent (SGD): (fast, noisy) requires gradients of single random training data points
- Mini Batch Stochastic Gradient (SGD): requires gradients and a small random batch of training data points
- Stochastic Gradient Descent with Momentum (ADAM)
- Others

Views of Gradient Descent



Train MNIST data with Tensorflow



```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
```

Tensorflow is imported
Sets the mnist dataset to variable “mnist”

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Loads the mnist dataset

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(16, activation='relu’),
 tf.keras.layers.Dense(16, activation='relu'),
  tf.keras.layers.Dense(10, activation=‘softmax’)])
```

Builds the layers of the model
4 layers in this model

```
  model.compile(optimizer=‘sgd', loss=‘SparseCategoricalCrossentropy’,
  metrics=['accuracy’])
  model.summary()
  logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
  tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
```

Loss Function

Compiles the model with the SGD optimizer

Print summary

```
  model.fit(x_train, y_train, epochs=5, batch_size=10, validation_data=(x_test,
  y_test),  callbacks=[tensorboard_callback])
```

Use tensorborad

Adjusts model parameters to minimize the loss

```
  model.evaluate(x_test,  y_test, verbose=2)
  %tensorboard --logdir logs
```

Tests the model performance on a test set

```
 library libcublas.so.10
2020-07-29 19:53:40.101102: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcufft.so.10
2020-07-29 19:53:40.101114: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcurand.so.10
2020-07-29 19:53:40.101126: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcusolver.so.10
2020-07-29 19:53:40.101138: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcusparse.so.10
2020-07-29 19:53:40.101151: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcudnn.so.7
2020-07-29 19:53:40.101222: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.101784: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.102281: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1703] Adding visible gpu devices: 0
2020-07-29 19:53:40.108638: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcudart.so.10.1
2020-07-29 19:53:40.592823: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102] Device interconnect StreamExecutor
 with strength 1 edge matrix:
2020-07-29 19:53:40.592860: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1108]      0
2020-07-29 19:53:40.592871: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1121] 0:   N
2020-07-29 19:53:40.593073: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.593535: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.593973: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.594387: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1247] Created TensorFlow device (/job:lo
calhost/replica:0/task:0/device:GPU:0 with 7219 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1080, pci bus
id: 0000:26:00.0, compute capability: 6.1)
2020-07-29 19:53:40.623075: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x55d981198b80 initialized fo
r platform CUDA (this does not guarantee that XLA will be used). Devices:
2020-07-29 19:53:40.623096: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): GeForce GTX
1080, Compute Capability 6.1
2020-07-29 19:53:52.215159: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcublas.so.10
Epoch 1/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2982 - accuracy: 0.9127
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1433 - accuracy: 0.9571
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1097 - accuracy: 0.9663
Epoch 4/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0893 - accuracy: 0.9730
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0777 - accuracy: 0.9750
313/313 - 0s - loss: 0.0727 - accuracy: 0.9776
(tf) julian@julian-MS-7C02:~/Downloads$
```

Does the Gradient Descent Method always Converge?

Theorem: Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be convex and differentiable and that its gradient is Lipschitz continuous with constant $L > 0$ $(ie. \left|\left|\nabla f(x) - \nabla f(y)\right|\right|_2 \leq L \left|\left|x - y\right|\right|_2$ for all x, y. Then if we run the gradient descent k times with fixed step size $t \leq \frac{1}{L}$, it will yield a solution such that $f\left(x^k\right) - f\left(x^*\right) \leq \frac{\left|\left|x^{(0)} - x^*\right|\right|_2^2}{2tk}$.

$f\left(x^*\right)$: optimal value

Main challenges of ML

In ML the main task is to select a Model and to Train it with some Data prior to deployment. The two things that can go wrong are: “bad model” or “bad data”.

1. **Bad Data**:

- Insufficient Quantity of Training Data (it takes a lot of data to work properly)
- Nonrepresentative Training Data (inaccurate predictions)
- Poor-Quality Data (full or errors, outliers, noise)
- Irrelevant Features (garbage in, garbage out)

(continued)

2. **Bad Model**:

- Overfitting the Training Data
- Underfitting the Training data

No free lunch theorem

(David Wolpert, 1996): *If you make absolutely no assumption about the data, then there is no reason to prefer one model over any other*.

For some datasets the best model is a linear model, while for other datasets it is a neural network.

There is no model that is a priori guaranteed to work better. In practice, one makes some reasonable assumptions about the data and evaluate a few reasonable models.

References

1. Deep learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville

2. Hands-On ML with Scikit-Learn, Keras, and TensorFlow (Aurelien Geron)

3. Machine Learning with Python by Bernd Klein

4. LAPENNA Workshop (UTK) by Kwai Wong

5. 6S191 MIT Deep Learning Lecture 1 by Alexander Amini