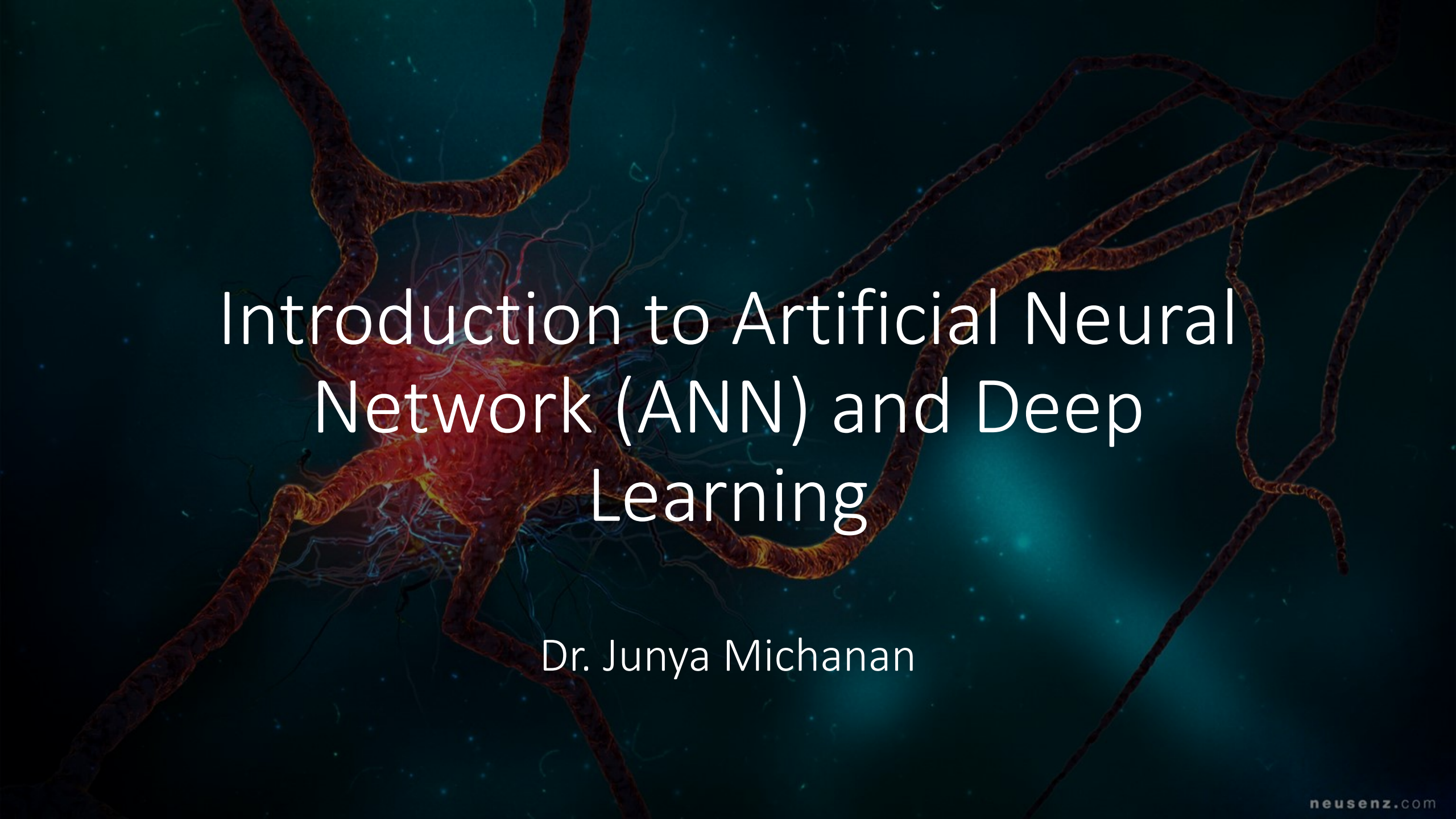


Introduction to Artificial Neural Network (ANN) and Deep Learning

Dr. Junya Michanan

What is Deep Learning?

- It's a technique that teaches computers to do what comes naturally to humans: learn by example.
- Deep Learning is the most exciting and powerful branch of Machine Learning.
- It's a technique that teaches computers to do what comes naturally to humans: learn by example.
- Deep learning is a key technology behind
  - driverless cars, enabling them to recognize a stop sign or to distinguish a pedestrian from a lamppost.
  - It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers.
- Models are trained by using a large set of labeled data and **neural network architectures** that contain many layers.

Deep Learning models can be used for a variety of complex tasks:

- Artificial Neural Networks(ANN) for Regression and classification
- Convolutional Neural Networks(CNN) for Computer Vision
- Recurrent Neural Networks(RNN) for Time Series analysis
- Self-organizing maps for Feature extraction
- Deep Boltzmann machines for Recommendation systems
- Auto Encoders for Recommendation systems.

What is Artificial Neural Networks or ANN?

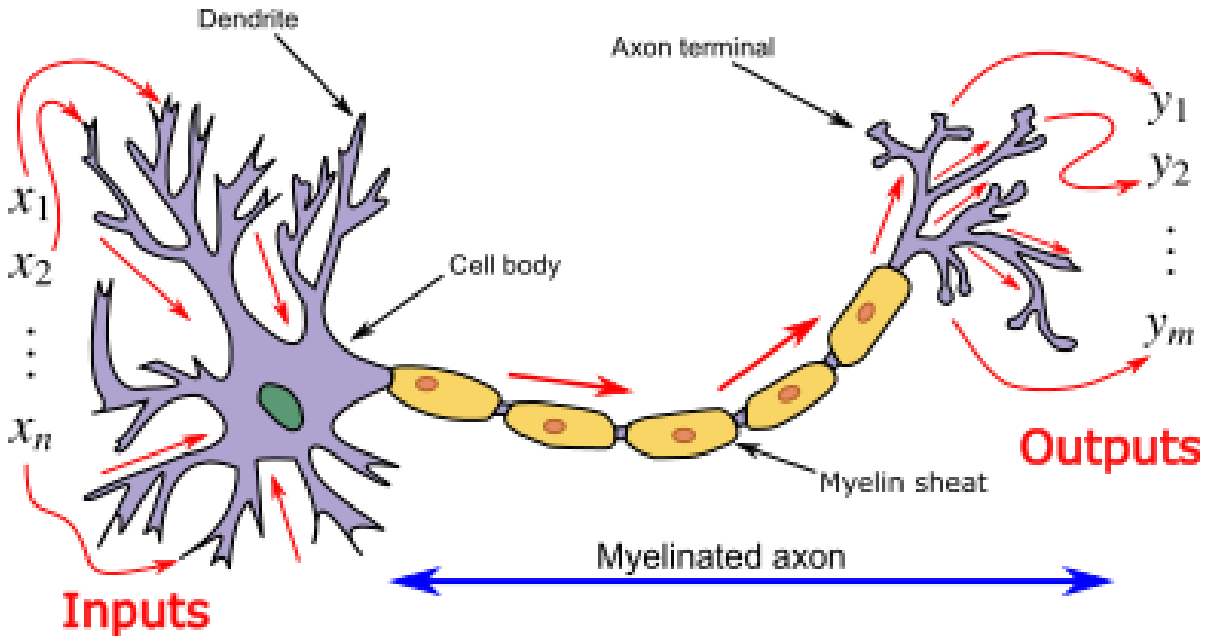
“Artificial Neural Networks or ANN is an information processing paradigm that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements(neurons) working in unison to solve a specific problem.”

Things to know about ANN:

- Neurons
- Activation Functions
- Types of Activation Functions
- How do Neural Networks work
- How do Neural Networks learn(Backpropagation)
- Gradient Descent
- Stochastic Gradient Descent
- Training ANN with Stochastic Gradient Descent

Neurons

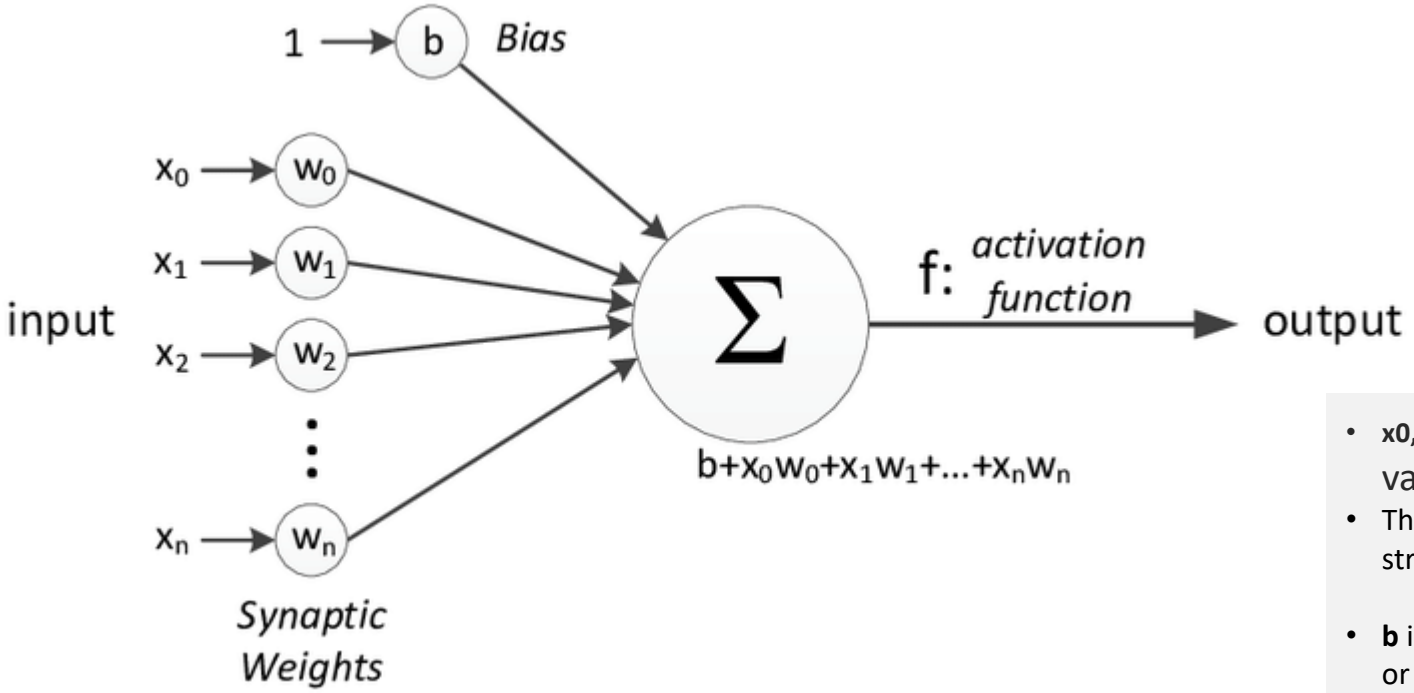
- Biological Neurons (also called nerve cells) or simply neurons
  - are the fundamental units of the brain and nervous system,
  - the cells responsible for receiving sensory input from the external world via dendrites, process it and gives the output through Axons.



1. **Cell body (Soma):** The body of the neuron cell contains the nucleus and carries out biochemical transformation necessary to the life of neurons.
2. **Dendrites:** Each neuron has fine, hair-like tubular structures (extensions) around it. They branch out into a tree around the cell body. They accept incoming signals.
3. **Axon:** It is a long, thin, tubular structure that works like a transmission line.
4. **Synapse:** Neurons are connected to one another in a complex spatial arrangement. When axon reaches its final destination it branches again called terminal arborization. At the end of the axon are highly complex and specialized structures called synapses. The connection between two neurons takes place at these synapses.

Perceptron.

- A model of ANN which is inspired by a biological neuron.
- A single layer neural network .



- **x0, x1, x2, x3…x(n)** represents various inputs(independent variables) to the network
- The weights are represented as **w0, w1, w2, w3….w(n)** . Weight shows the strength of a particular node.
- **b** is a bias value. A bias value allows you to shift the activation function up or down.

Mathematically, $x1.w1 + x2.w2 + x3.w3 ..... xn.wn = \sum xi.wi$
f: activation function is applied $\phi(\sum xi.wi)$

Activation functions

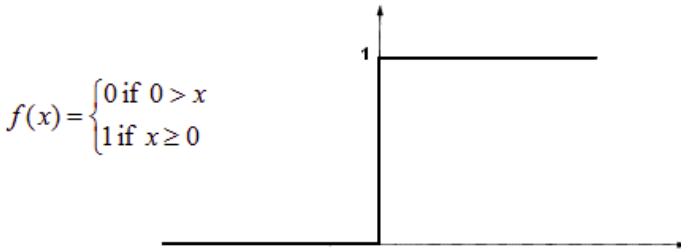
“Activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The motive is to introduce non-linearity into the output of a neuron.”

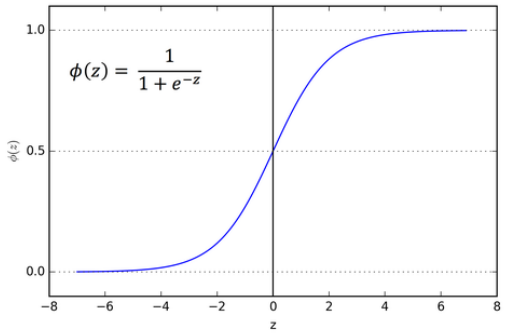
- Their main purpose is to convert an input signal of a node in an ANN to an output signal. This output signal is used as input to the next layer in the stack

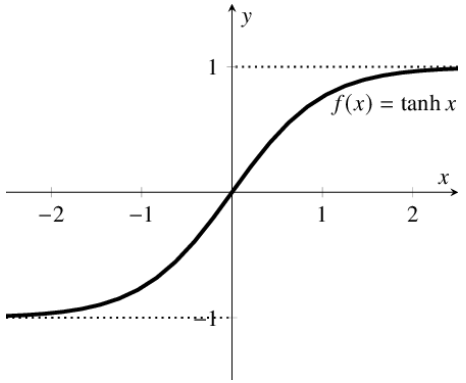
Neural Network is considered “Universal Function Approximators”. It means they can learn and compute any function at all.

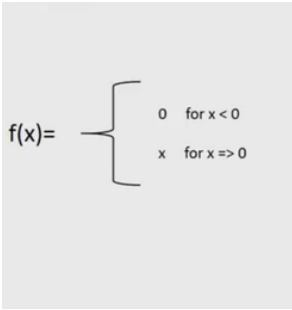
Types of Activation Functions

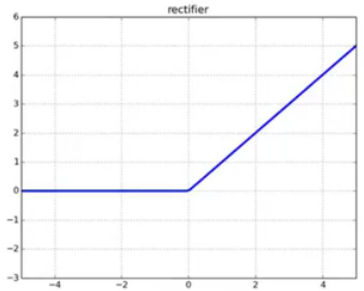
1. Threshold Activation Function — (Binary step function)
2. Sigmoid Activation Function — (Logistic function)
3. Hyperbolic Tangent Function — (tanh)
4. Rectified Linear Units — (ReLu)











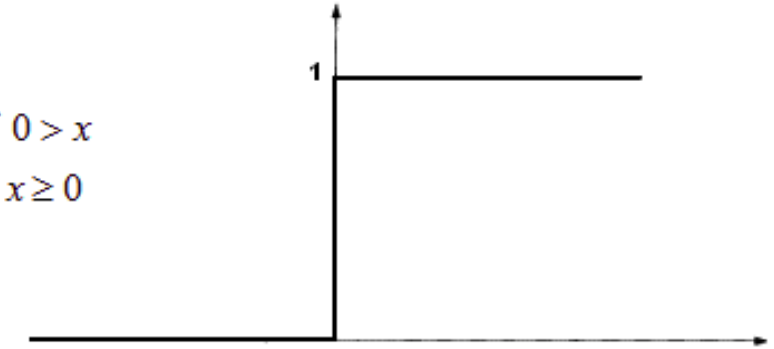
1. Threshold Activation Function (Binary step function)

- A Binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.

Activation function A = “activated” if Y > threshold

else not or A=1 if y>threshold 0 otherwise.

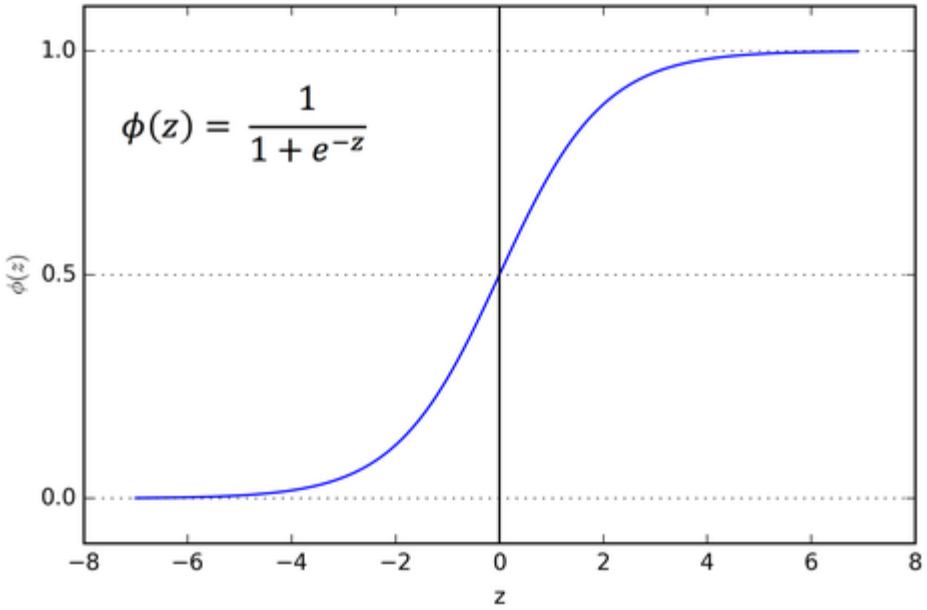
$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



- The problem with this function is for creating a binary classifier (1 or 0), but if you want multiple such neurons to be connected to bring in more classes, Class1, Class2, Class3, etc. In this case, all neurons will give 1, so we cannot decide.

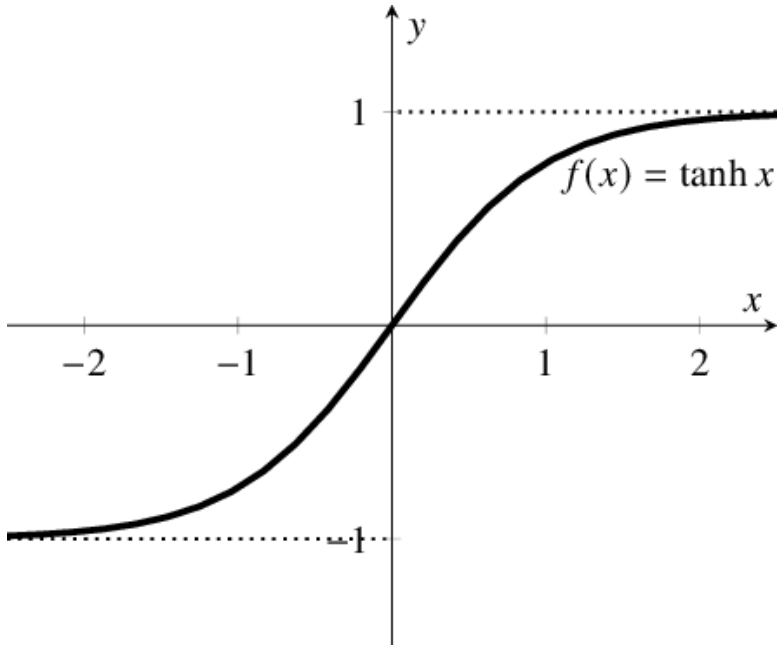
2. Sigmoid Activation Function (Logistic function)

- A Sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve which ranges between 0 and 1.
- It is used for models where we need to predict the probability as an output.
- The drawback of the Sigmoid activation function is that it can cause the neural network to get stuck at training time if strong negative input is provided.



3. Hyperbolic Tangent Function (tanh)

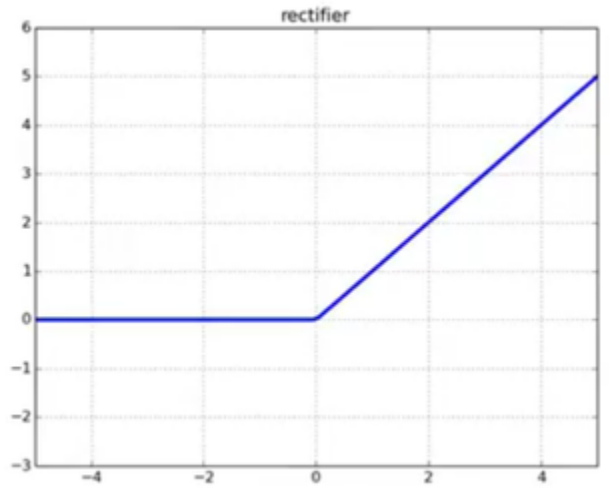
- It is similar to Sigmoid but better in performance.
- It is nonlinear in nature, so great we can stack layers.
- The function ranges between (-1,1).
- The main advantage of this function is that strong negative inputs will be mapped to negative output and only zero-valued inputs are mapped to near-zero outputs. So less likely to get stuck during training.

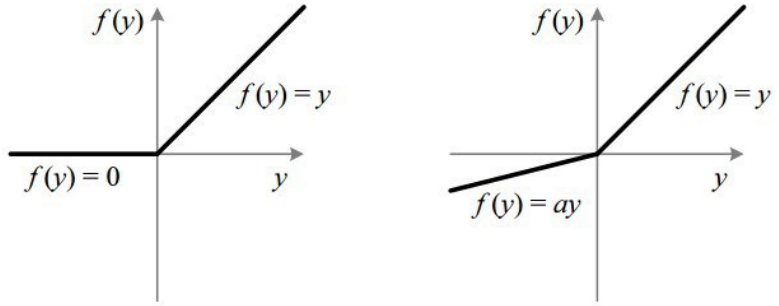


4. Rectified Linear Units (ReLu)

- ReLu is the most used activation functio in CNN and ANN which ranges from zero to infinity: [0,∞)
- It gives an output ‘x’ if x is positive and 0 otherwise.
- ReLu is 6 times improved over hyperbolic tangent function.
- It should only be applied to hidden layers of a neural network.
- Leaky ReLu was introduced. So, Leaky ReLu introduces a small slope to keep the updates alive. Leaky ReLu ranges from -∞ to +∞.

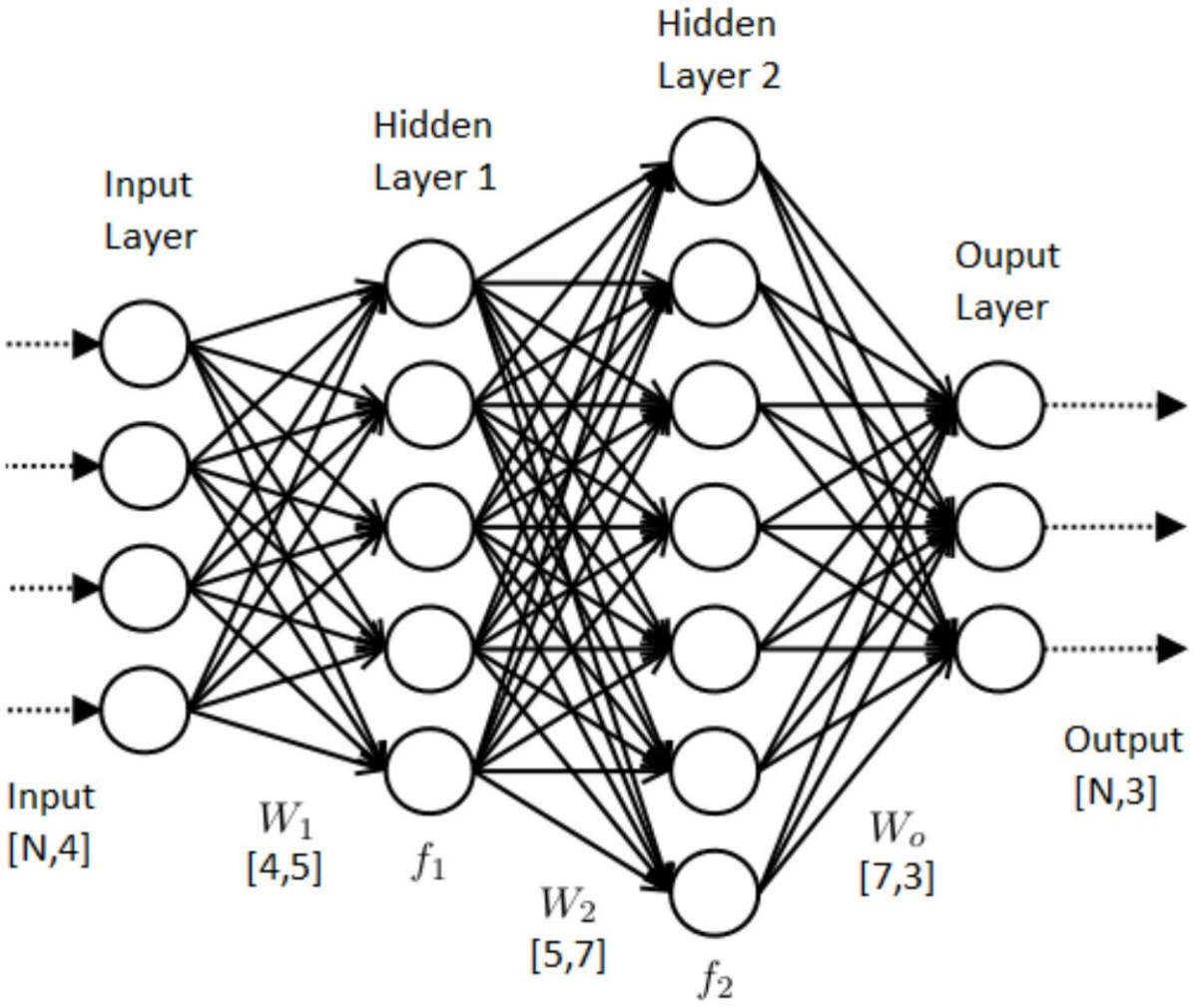
$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \Rightarrow 0 \end{cases}$$



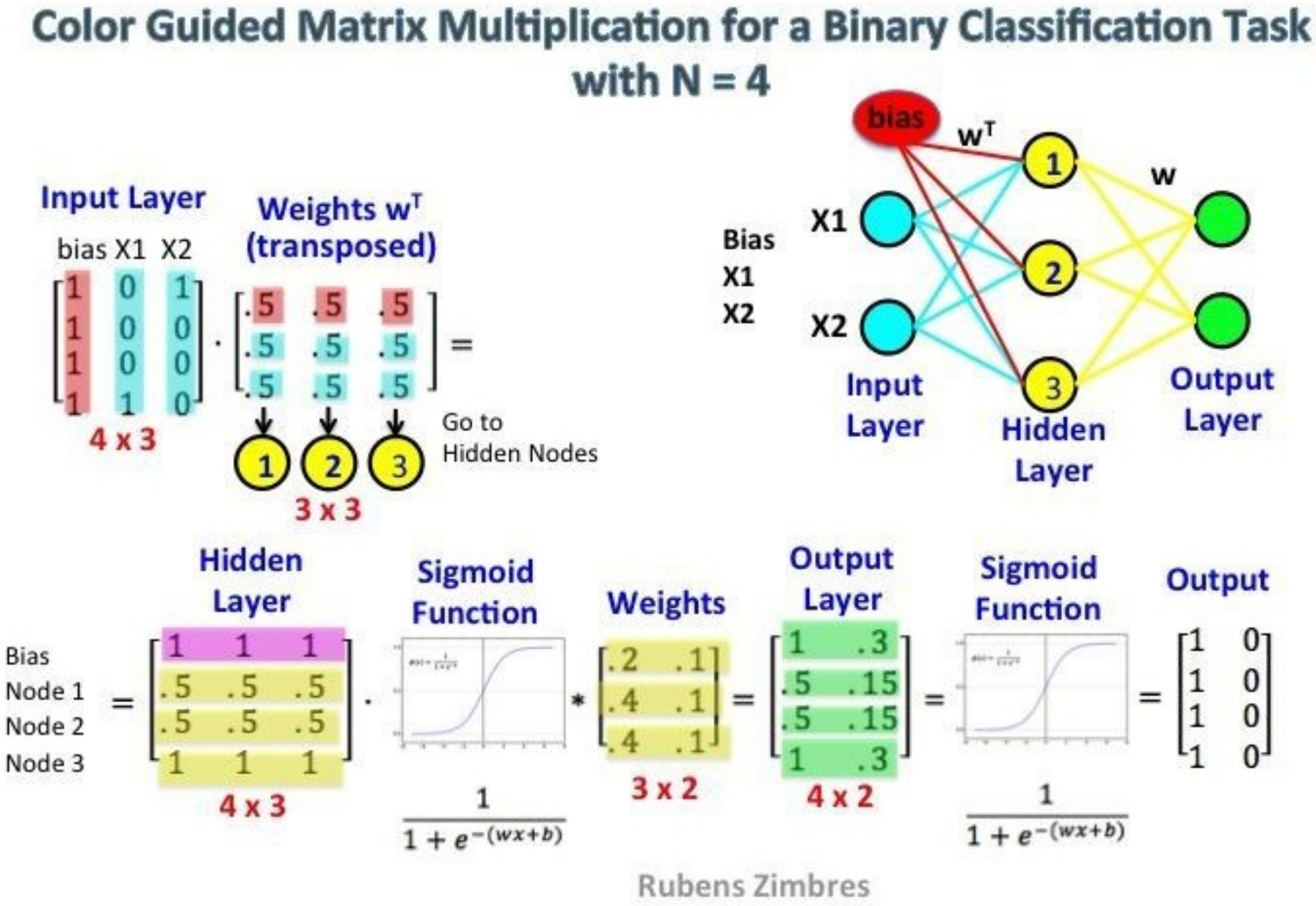


ReLu vs Leaky ReLu

How does the Neural network work?

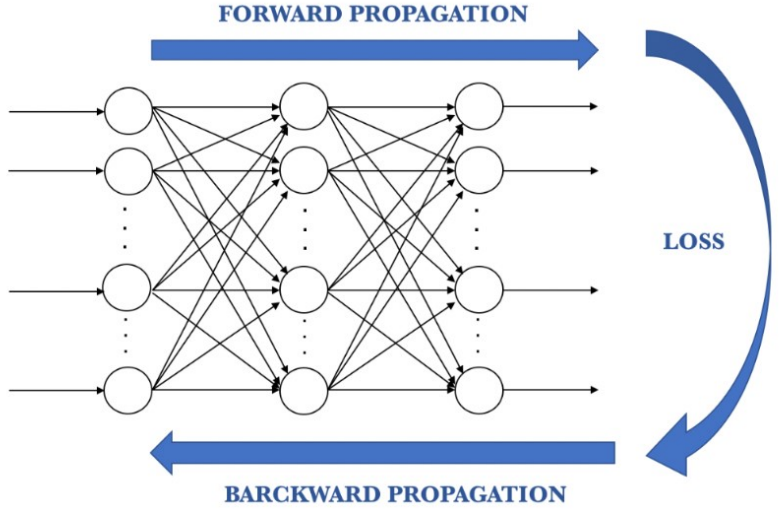


How does the Neural network work?



How do Neural networks learn?

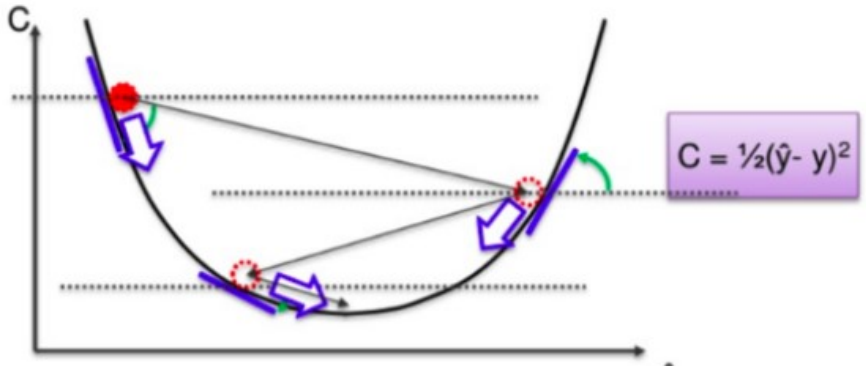
- Learning in a neural network is closely related to how we learn in our regular lives and activities — we perform an action and are either accepted or corrected by a trainer or coach to understand how to get better at a certain task.
- Neural networks require a trainer in order to describe what should have been produced as a response to the input. Based on the difference between the actual value and the predicted value, an error value also called Cost Function is computed and sent back through the system.
- **Cost Function:** One half of the squared difference between actual and output value.



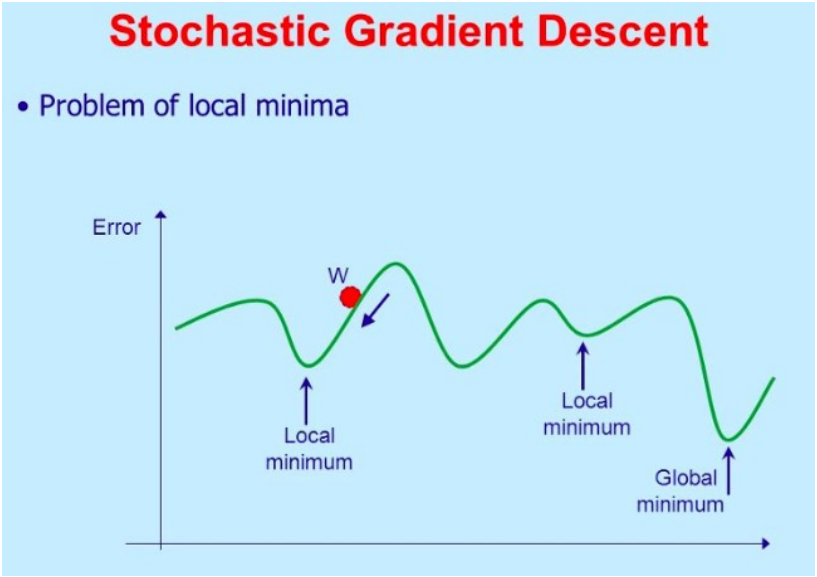
AIM: to minimize the cost function (Error) down to as small as possible.

Cost Optimization

- Methods to adjust weights:
    1. Brute-force method →Problem with “Curse of Dimensionality”
    2. Batch-Gradient Descent
    3. Stochastic Gradient Descent(SGD)



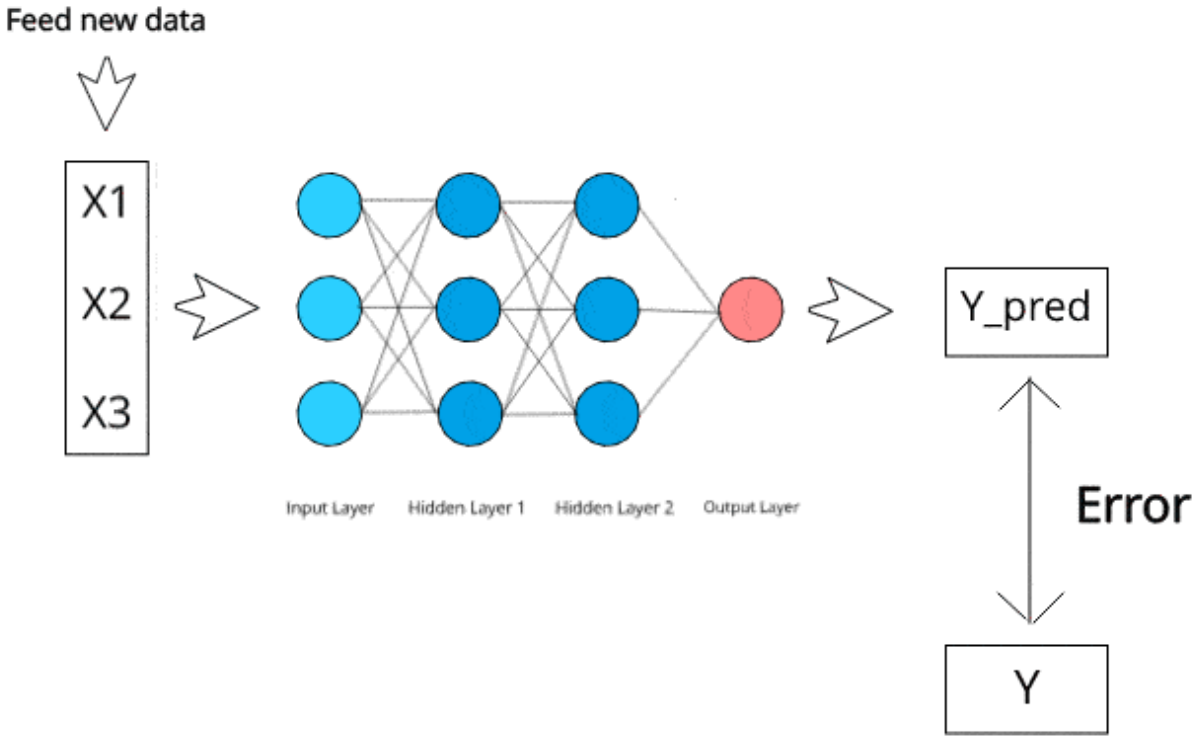
If slope → Negative, that means you go down the curve.
If slope → Positive, Do nothing



- Randomness in its descent
- Computationally much less expensive than typical Gradient Descent

Training ANN with Stochastic Gradient Descent

- **Step-1** → Randomly initialize the weights to small numbers close to 0 but not 0.
- **Step-2** → Input the first observation of your dataset in the input layer, each feature in one node.
- **Step-3** → Forward-Propagation: From left to right, the neuron are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted value.
- **Step-4** → Compare the predicted result to the actual result and measure the generated error(Cost function).
- **Step-5** → Back-Propagation: from right to left, the error is backpropagated. Update the weights according to how much they are responsible for the error. The learning rate decides how much we update weights.
- **Step-6** → Repeat step-1 to 5 and update the weights after each observation(Reinforcement Learning)
- **Step-7** → When the whole training set passed through the ANN, that makes and epoch. Redo more epochs.



Your First Deep Learning Project in Python with Keras Step-By-Step

- Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models.

Keras Tutorial Overview:

- The steps you are going to cover in this tutorial are as follows:
    1. Load Data.
    2. Define Keras Model.
    3. Compile Keras Model.
    4. Fit Keras Model.
    5. Evaluate Keras Model.
    6. Make Predictions

REQUIREMENTS:
1.You have Python 2 or 3 installed and configured.
2.You have SciPy (including NumPy) installed and configured.
3.You have Keras and a backend (Theano or TensorFlow) installed and configured.

Step 1: Load Data

- Dataset CSV File (pima-indians-diabetes.csv)

Data: Pima Indians Diabetes Database

Number of Instances: 768

Number of Attributes: 8 plus class

For Each Attribute: (all numeric-valued)

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

```
...
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
...
```

2. Define Keras Model

- We can piece it all together by adding each layer:
  - The model expects rows of data with 8 variables (the input_dim=8 argument)
  - The first hidden layer has 12 nodes and uses the relu activation function.
  - The second hidden layer has 8 nodes and uses the relu activation function.
  - The output layer has one node and uses the sigmoid activation function.

```
...
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
...
```

3. Compile Keras Model

- Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware, such as CPU or GPU or even distributed.
- we will use cross entropy as the loss argument. This loss is for a binary classification problems and is defined in Keras as “binary_crossentropy“.
- We will define the optimizer as the efficient stochastic gradient descent algorithm “adam“. This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems.
- Because it is a classification problem, we will collect and report the classification accuracy, defined via the **metrics** argument.

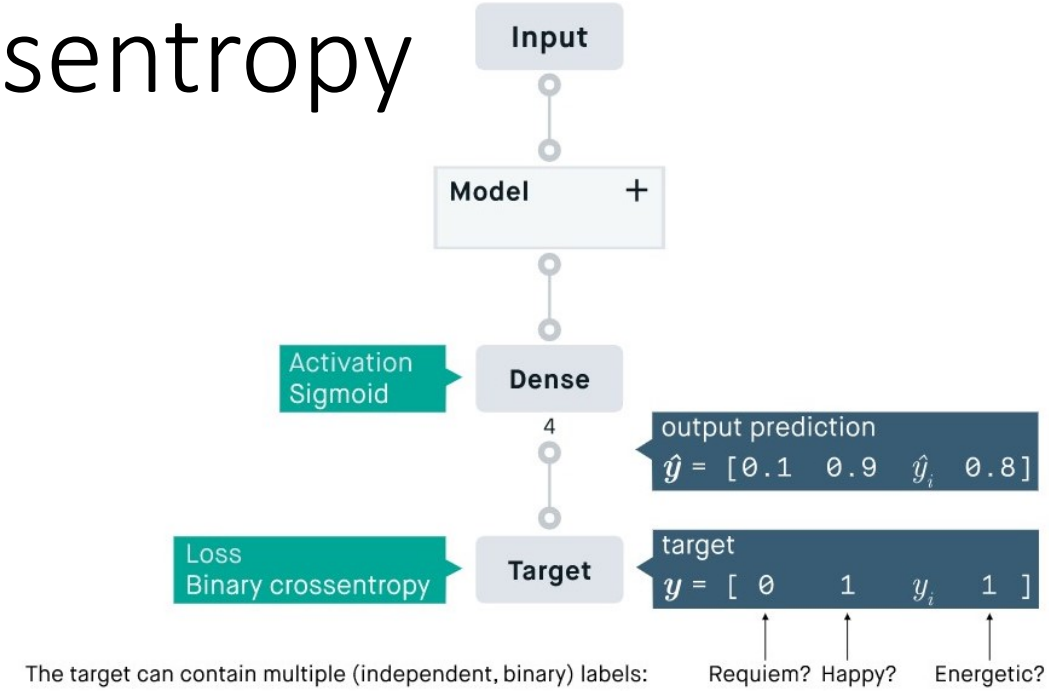
```
...
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
...
```

Types of Loss Functions

1. Regression Loss Functions
    1. Mean Squared Error Loss
    2. Mean Squared Logarithmic Error Loss
    3. Mean Absolute Error Loss
2. Binary Classification Loss Functions
    1. Binary Cross-Entropy
    2. Hinge Loss
    3. Squared Hinge Loss
3. Multi-Class Classification Loss Functions
    1. Multi-Class Cross-Entropy Loss
    2. Sparse Multiclass Cross-Entropy Loss
    3. Kullback Leibler Divergence Loss

Binary vs categorical crossentropy

- Binary crossentropy is a **loss function** that is used in binary classification tasks. These are tasks that answer a question with only two choices (yes or no, A or B, 0 or 1, left or right).
- This loss is equal to the average of the categorical crossentropy loss on many two-category tasks.



$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

where $\hat{y}_i$ is the i-th scalar value in the model output, y_i is the corresponding target value, and `output size` is the number of scalar values in the model output.

This is equivalent to the average result of the categorical crossentropy loss function applied to many independent classification problems, each problem having only two possible classes with target probabilities y_i and $(1 - y_i)$.

4. Fit Keras Model

- We can train or fit our model on our loaded data by calling the fit() function on the model.
- Training occurs over epochs and each epoch is split into batches.
  - **Epoch:** One pass through all of the rows in the training dataset.
  - **Batch:** One or more samples considered by the model within an epoch before weights are updated.
- For this problem, we will run for a small number of epochs (150) and use a relatively small batch size of 10.

```
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
```

5. Evaluate Keras Model

- You can evaluate your model on your training dataset using the evaluate() function on your model and pass it the same input and output used to train the model.
- The evaluate() function will return a list with two values. The first will be the loss of the model on the dataset and the second will be the accuracy of the model on the dataset. We are only interested in reporting the accuracy, so we will ignore the loss value.

```
...
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

6. Make Predictions

- Making predictions is as easy as calling the predict() function on the model. We are using a sigmoid activation function on the output layer, so the predictions will be a probability in the range between 0 and 1.

```
...
# make probability predictions with the model
predictions = model.predict(X)
# round predictions
rounded = [round(x[0]) for x in predictions]
```