NUS National University of Singapore | iss INSTITUTE OF SYSTEMS SCIENCE

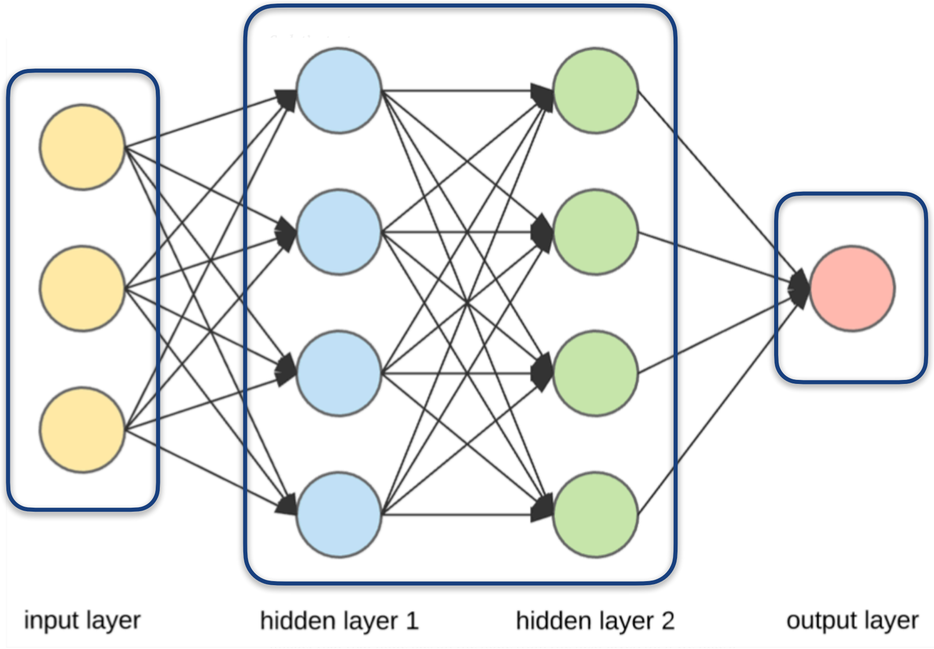
Machine Learning Applications

Neural Network

Introduction

Neural Network

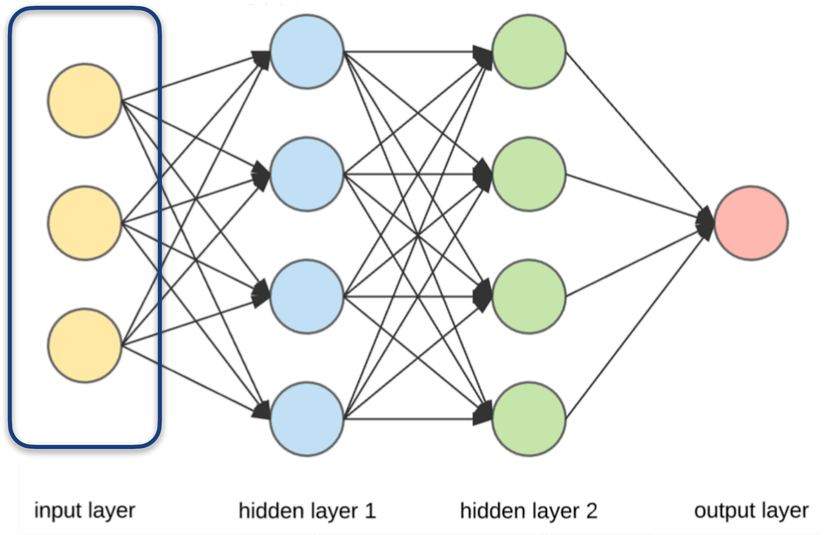
- A Neural Network is made of interconnected layers of computing units
- Common layers – Input, Hidden and Output



A Simple Neural Network

Input Layer

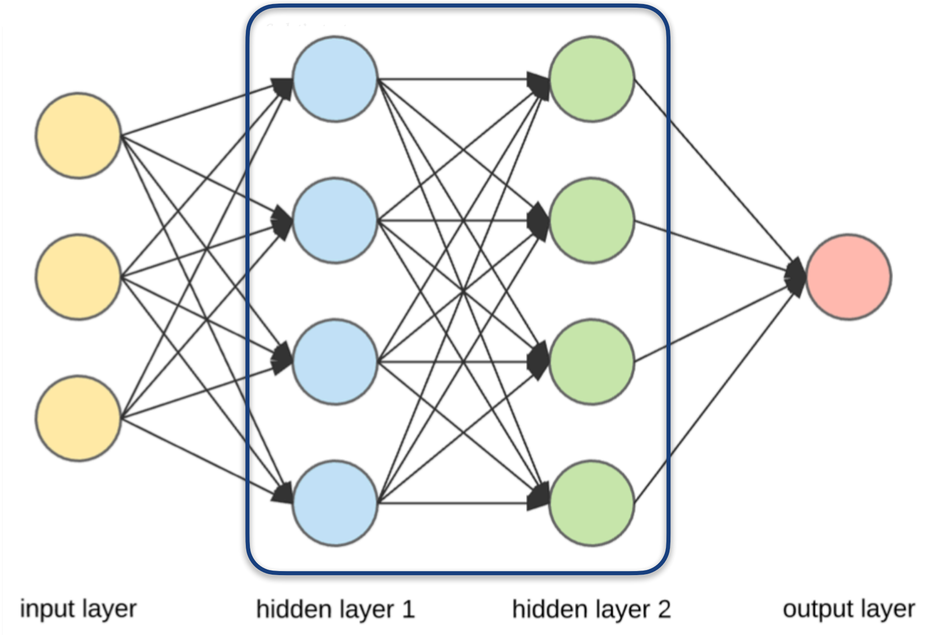
- The Input layer contains the inputs for computation
- If we are training a neural network to predict if a person is overweight, then the inputs could be Age, Weight and Height



A Simple Neural Network

Hidden Layers

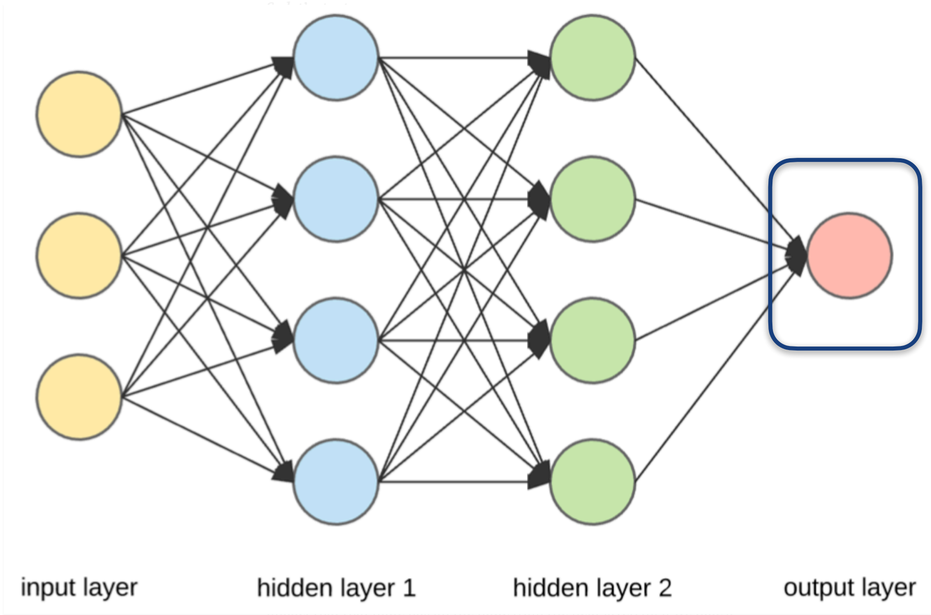
- Hidden layers are the layers between the Input and Output layers
- Bulk of computations happens in the Hidden layers



A Simple Neural Network

Output Layer

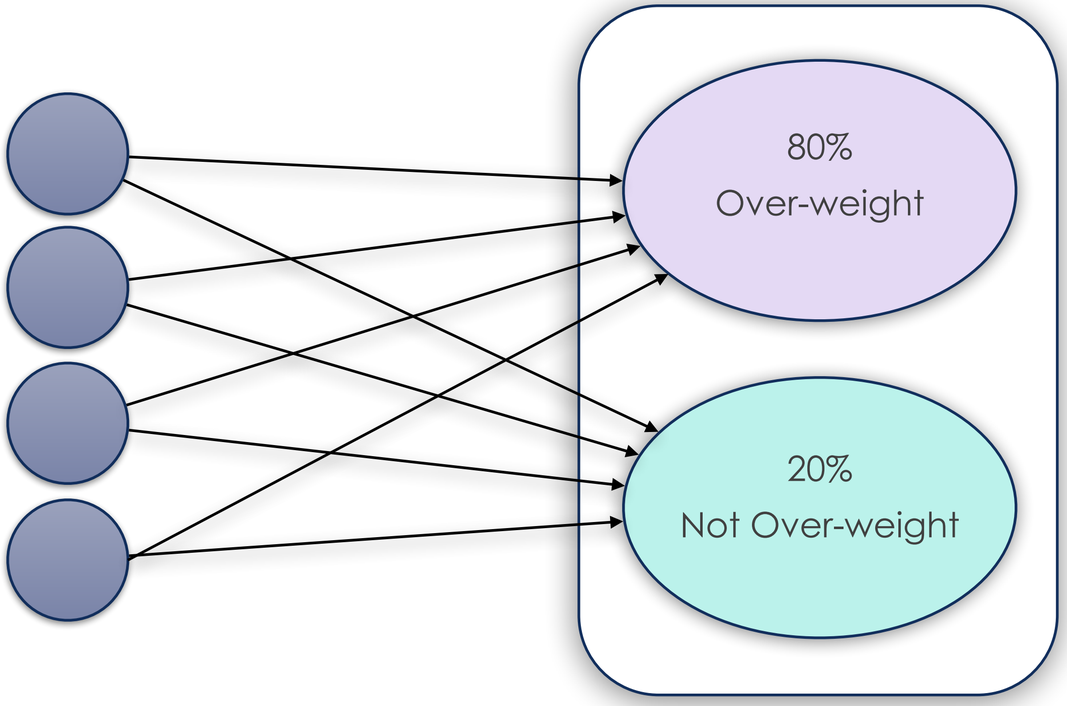
- The Output layer gives the computed prediction from a neural network
- For predicting if a person is overweight, the output can be a value between 0 and 1, where any value above 0.5 denotes the person is overweight



A Simple Neural Network

Output Layer

- Alternatively, our Output layer could have been designed with 2 output nodes in mind, where each node carries a probability of the person being overweight

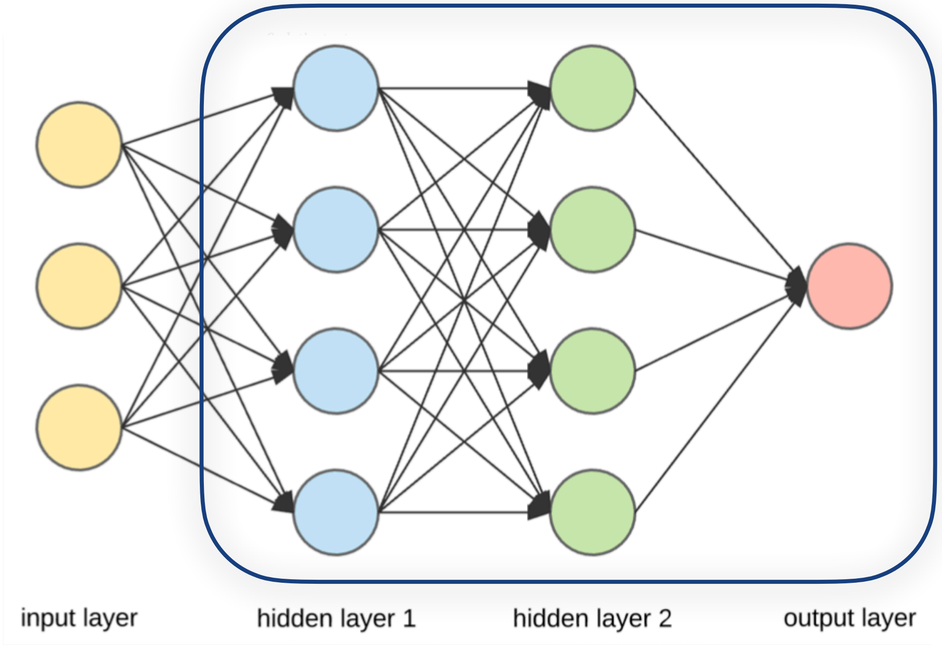


hidden layer 2

output layer (with 2 neurons)

A N-layer Neural Network

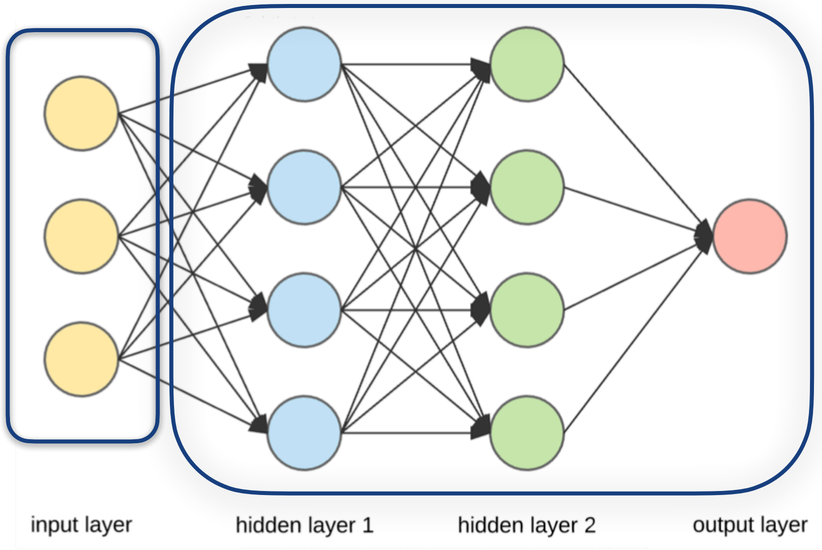
- A Neural Network is called N-layer if the sum of hidden and output layers is N
- The diagram depicts a 3-layer Neural Network



A Simple Neural Network

Neurons

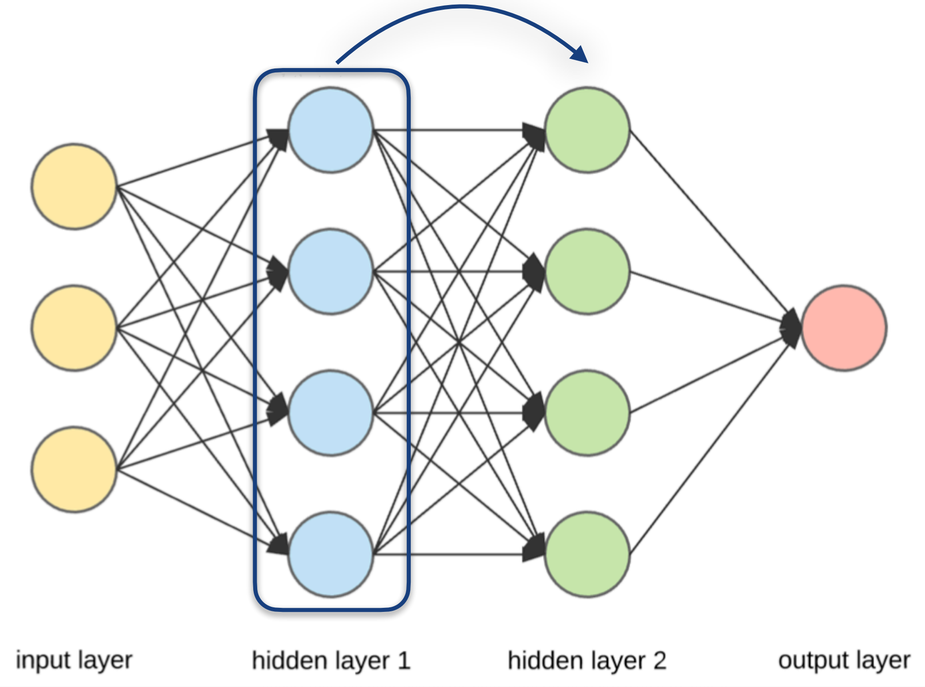
- The nodes in a neural network are called neurons
- The **blue**, **green** and **red** nodes are neurons
- The **yellow** nodes are not neurons, they are just input values



A Simple Neural Network

Neurons

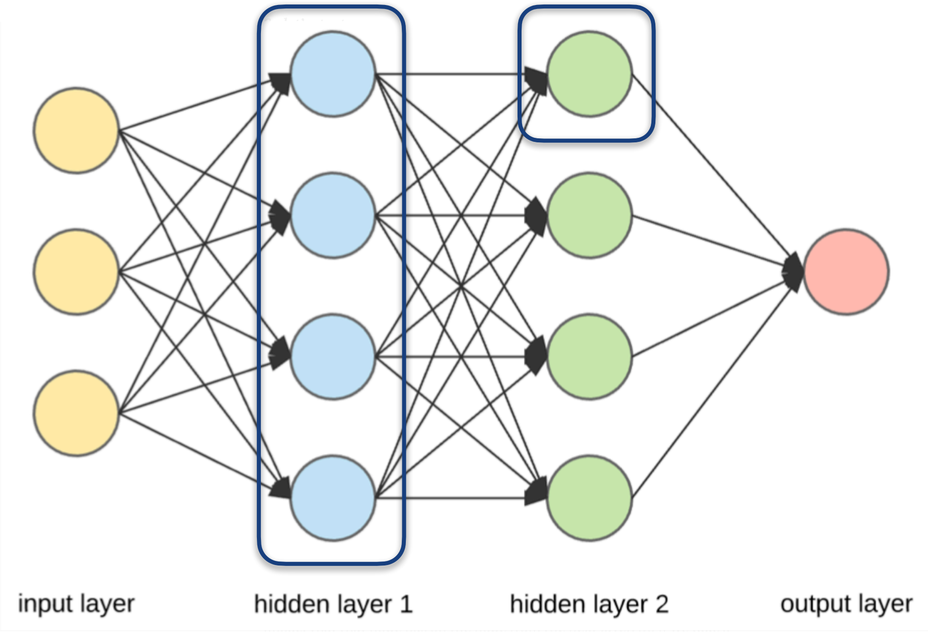
- Each neuron contains mathematical functions
- Each neuron applies its mathematical functions on values that comes from a previous layer



A Simple Neural Network

Neurons

- Each **green** neuron applies its mathematical functions on values coming from its connected 4 **blue** neurons in the previous layer

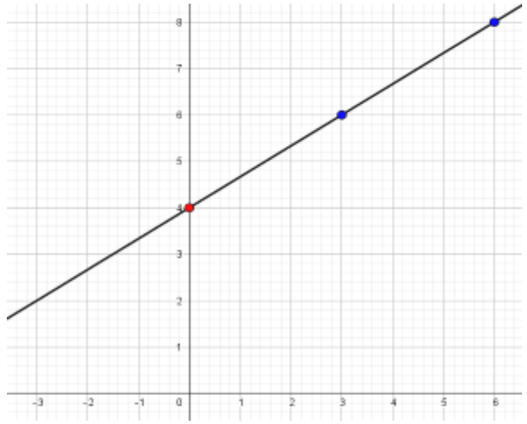


A Simple Neural Network

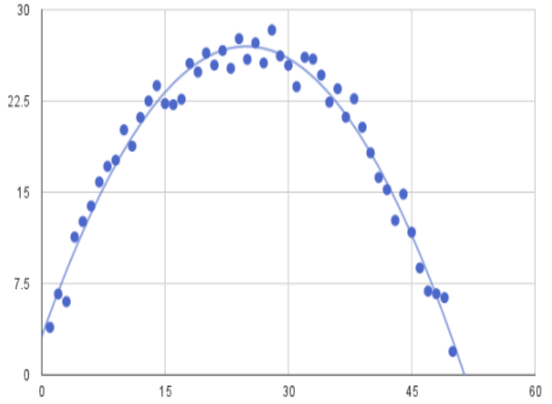
Weights and Biases

Universal Function Approximator

- Neural Networks are Universal Function Approximators that can approximate any mathematical function (linear or non-linear functions)



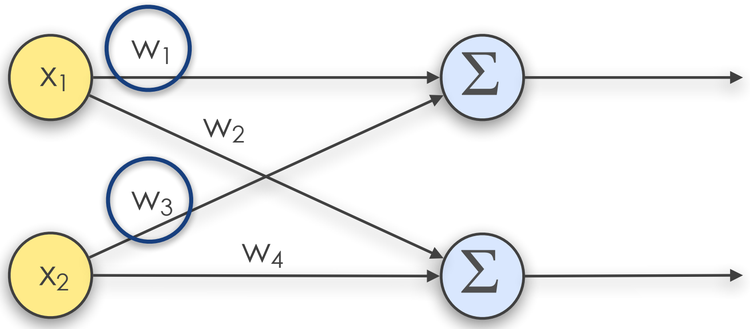
A linear equation



A non-linear equation

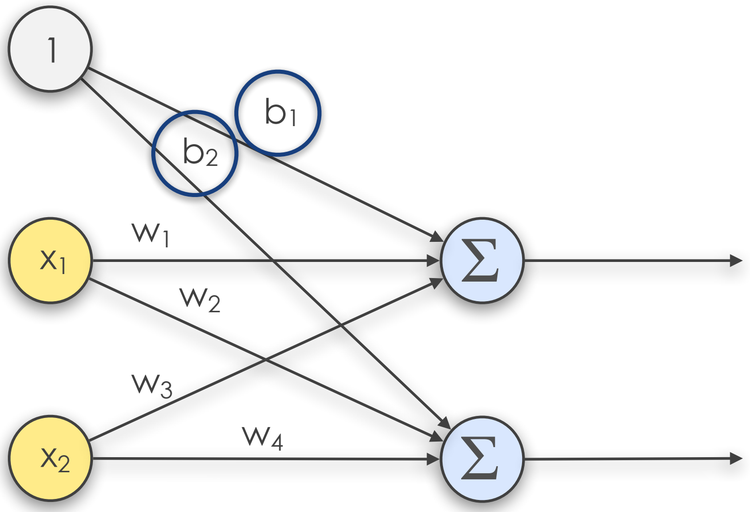
Weights

- Weights are numerical values attached to connections between neurons, and they control the impact of incoming values to the next neuron



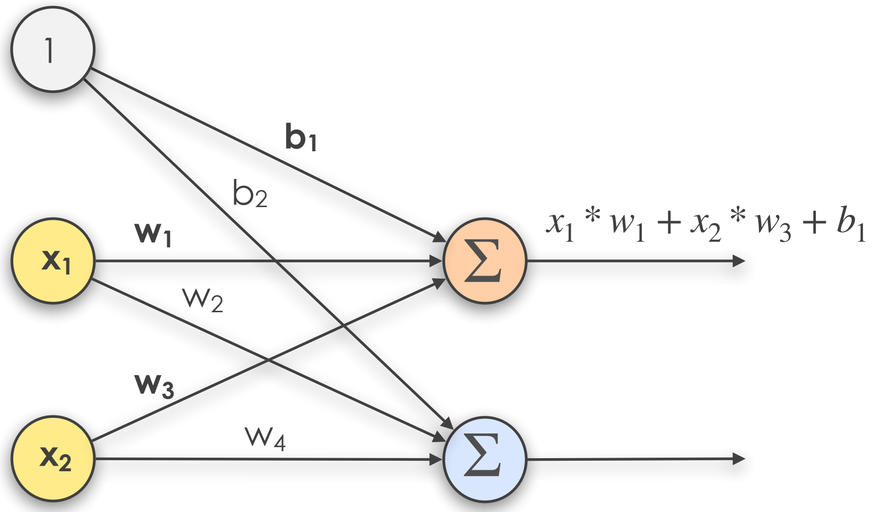
Biases

- Biases are special weights that allow finer adjustments to computed equations



Weighted Sum

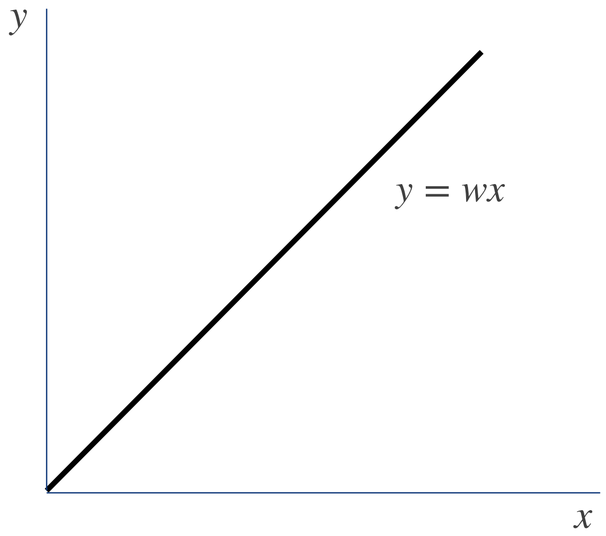
- Weighted Sums are computed by multiplying and adding weights and biases with inputs from previous layers



Weighted Sums calculation

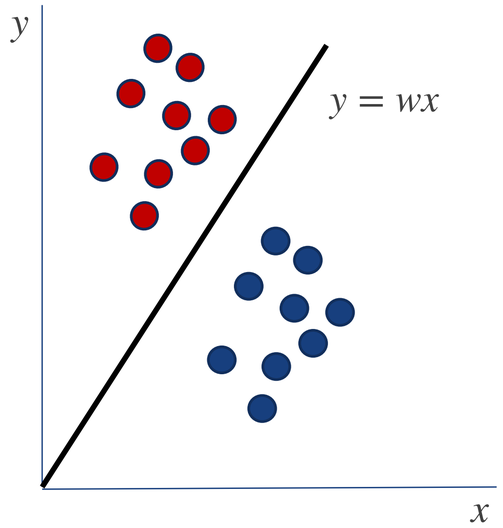
Weights only

- Let's look at a neural network that approximates a linear function that separates a set of data
- A straight line is defined as $y = wx$, where w is the gradient



Weights only

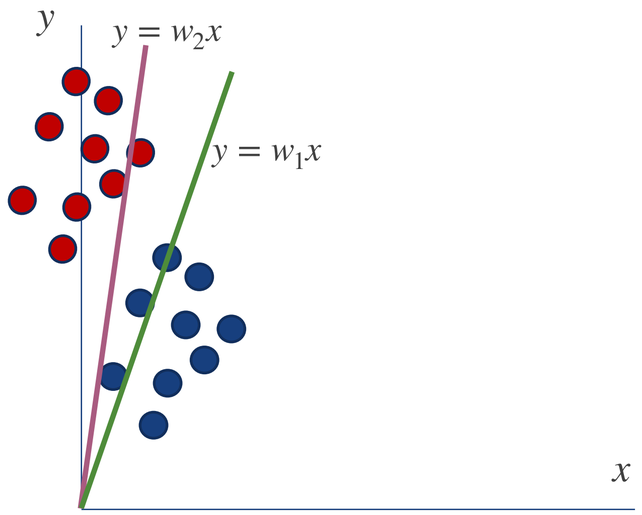
- We can only solve those classification problems whose data points can be separated by a line that intercepts the origin with Weights only



Weights only

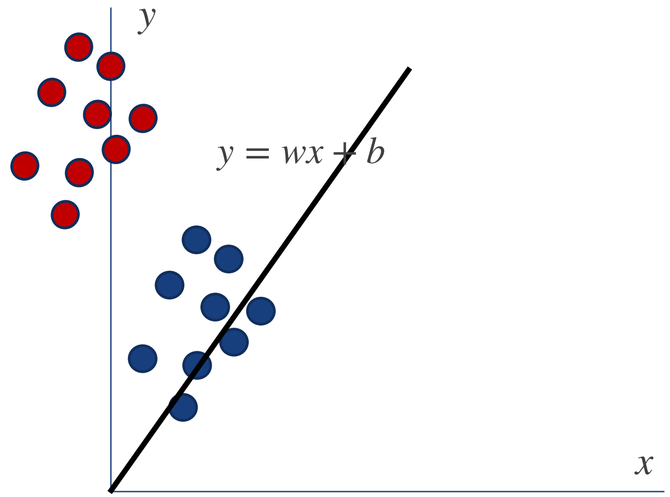
Weights only

- These data do not allow us to separate it cleanly by adjusting the gradient alone (i.e. using only Weights in our computation)



Weights and Biases

- Bias gives us the ability to move away from the origin
- The diagram shows the same problem solved by adding a bias

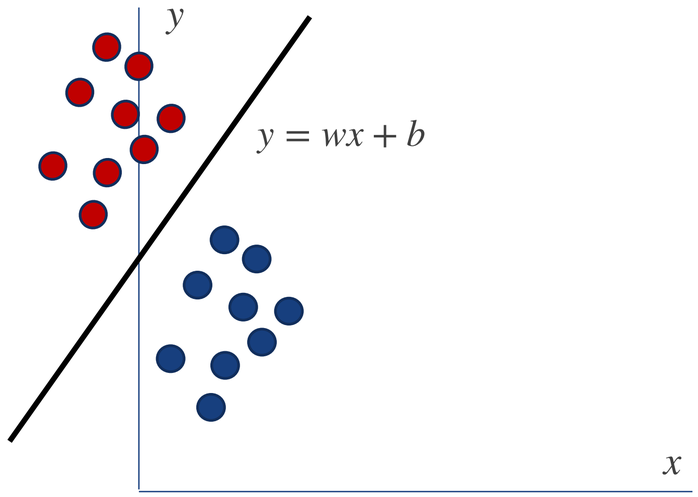


Weights and Biases

Activation functions

Weighted Sums only

- Linear solutions, such as the one provided by Weighted Sums, can solve problems that are linearly separable



Linearly separable data

Weighted Sums only

- A Weighted Sum operation yields a linear equation: $y = Wx + b$
- As the output of a weighted sum operation in one layer is the input to a weighted sum operation of the next, a neural network with only weighted sum operations can only produce linear approximations
- Example:

  $Y_1 = 2X + 1$

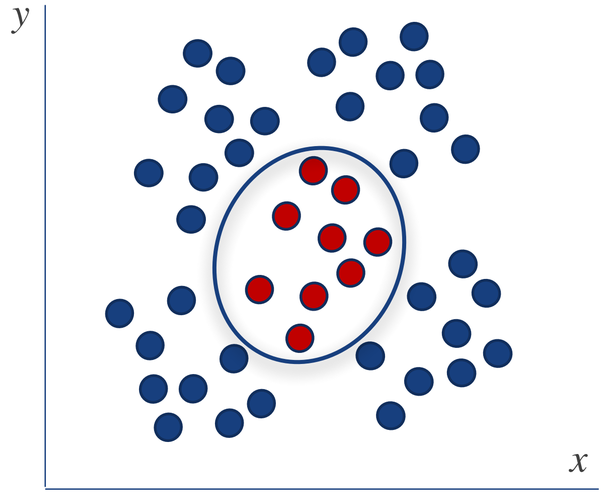
  $Y_2 = 3Y_1 + 4$

  $Y_2 = 3(2X + 1) + 4$

  $Y_2 = 6X + 7$ (still a linear equation)

Weighted Sums only

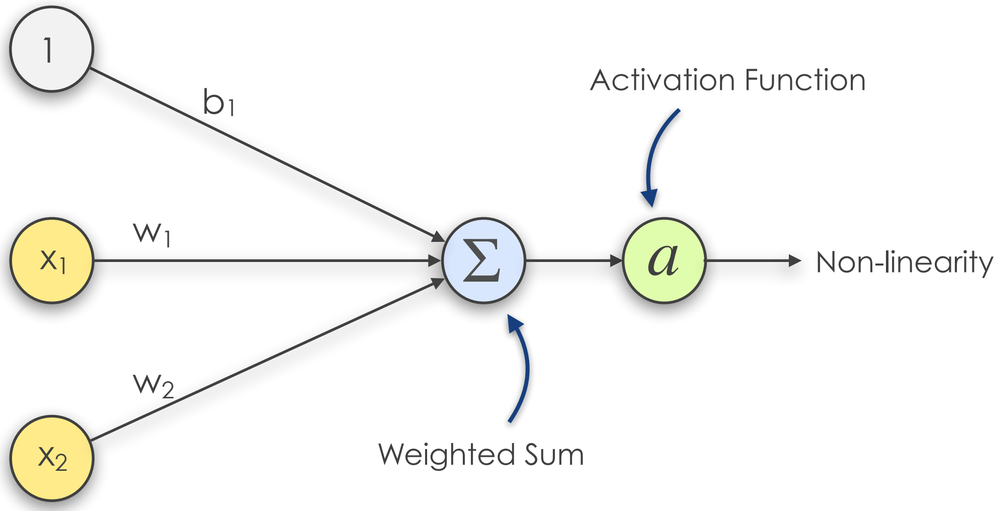
- Data that are non-linearly separable requires the use of non-linear equations
- We shall see how Activation Functions add non-linearity to our models to solve problems like such



Non-Linearly separable data

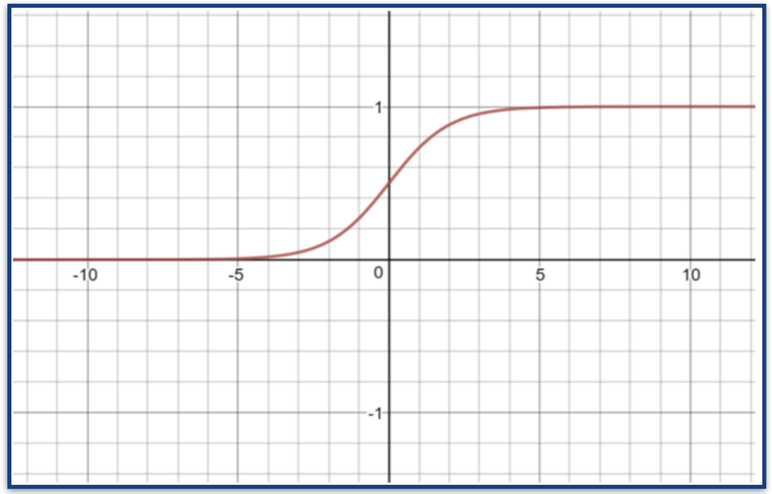
Non-linear transformation

- A Weighted Sum operation, followed by a Non-Linear Transformation applied by an Activation Function



Sigmoid

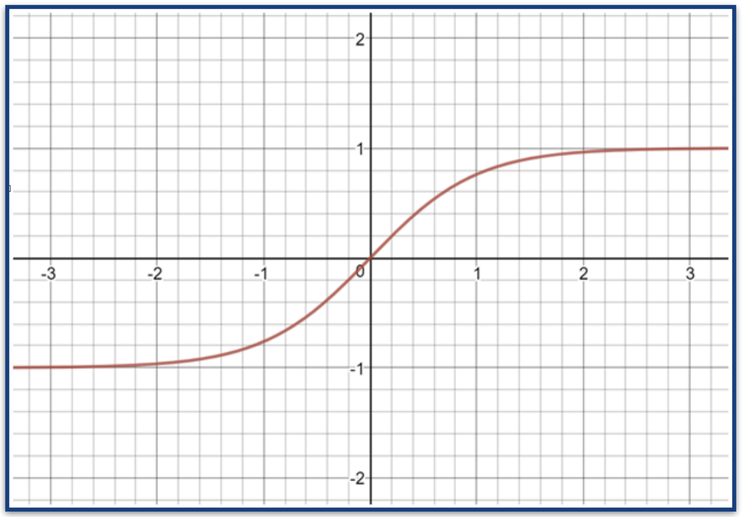
- The Sigmoid activation function, also called the Logistic function, transforms inputs into a value between 0 and 1
- Useful for models that predict a probability as an output



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

Tanh

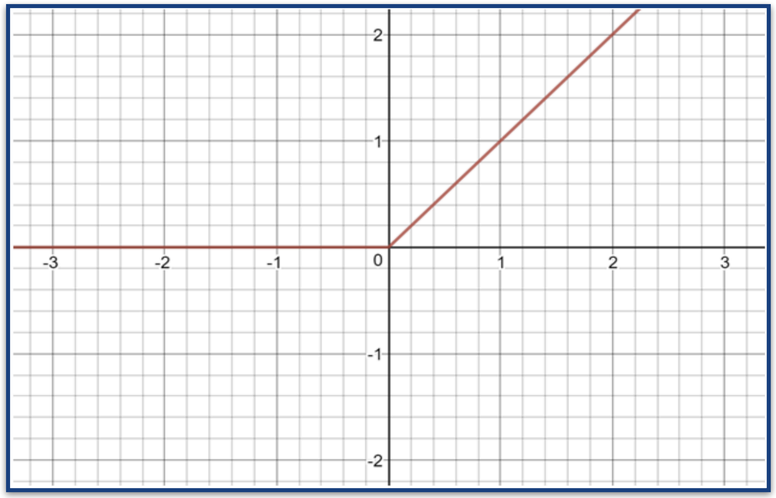
- The Tanh activation function transforms inputs to values between -1 and 1
- Useful for models that yields normalized positive and negative outputs



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

ReLU

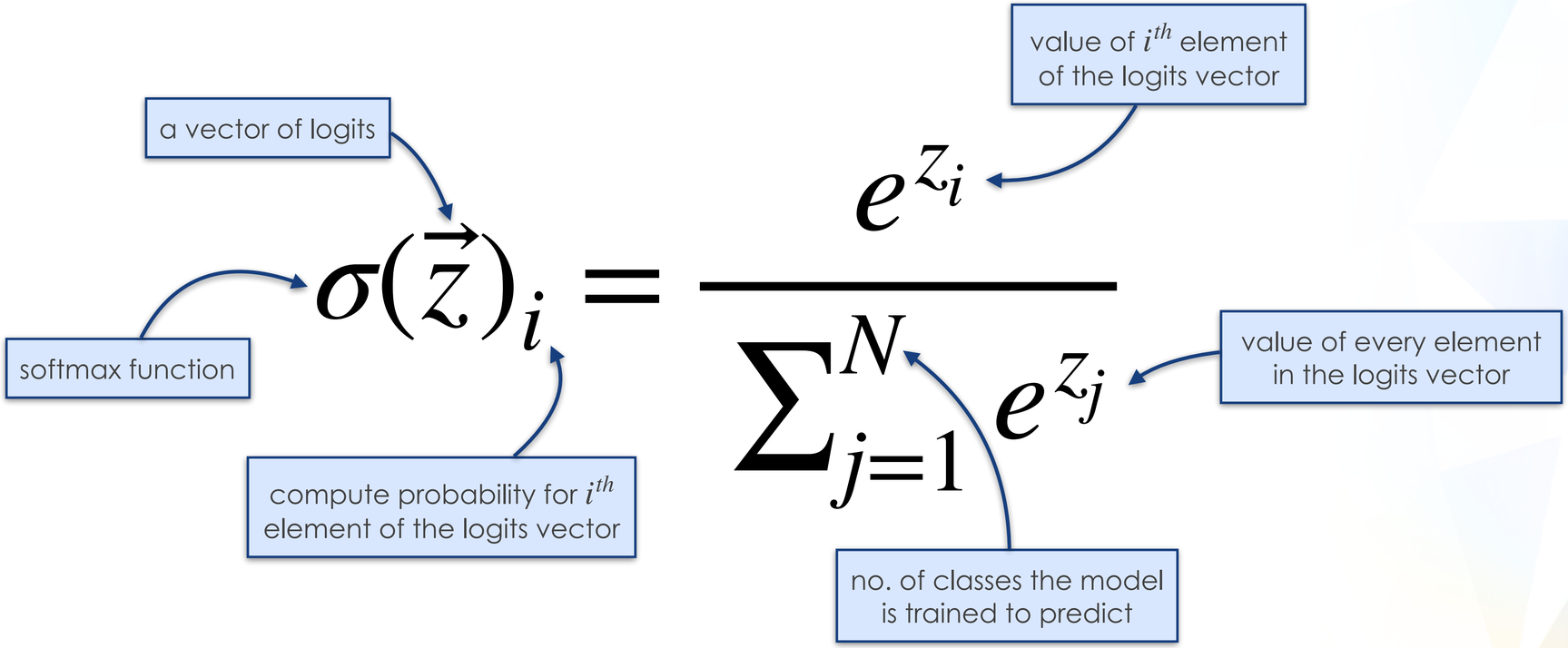
- The ReLU activation function transforms inputs with negative values to 0, but leave other values unchanged



$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$$

Softmax

- The Softmax activation function takes a final computed vector and outputs a vector of probabilities as predictions



Loss Functions

- When training, a loss function helps to determine the error of our neural network
- The error tell us how far our predicted value is from the actual value
- We want to minimize the error in our neural network such that its predictions are accurate
- Loss functions are only used in model training

Loss Function

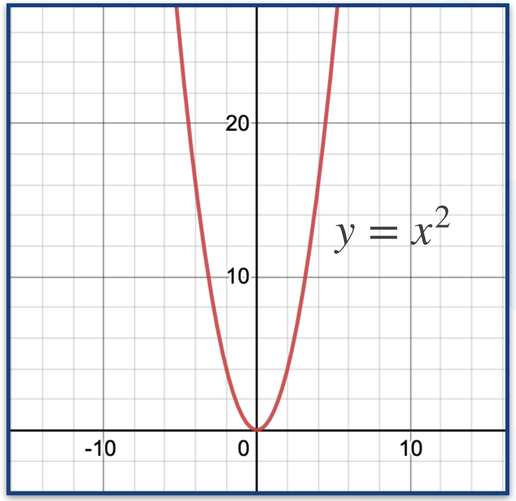
- Mean Square Error (aka L_2 loss) is a loss function that is commonly used

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

Actual Value (Ground Truth)

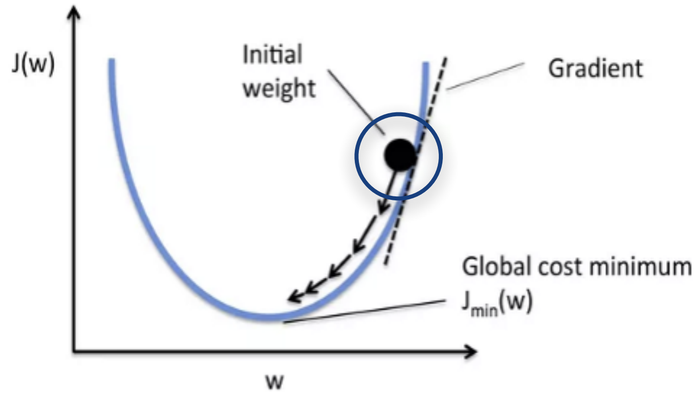
Batch Size

Predicted Value



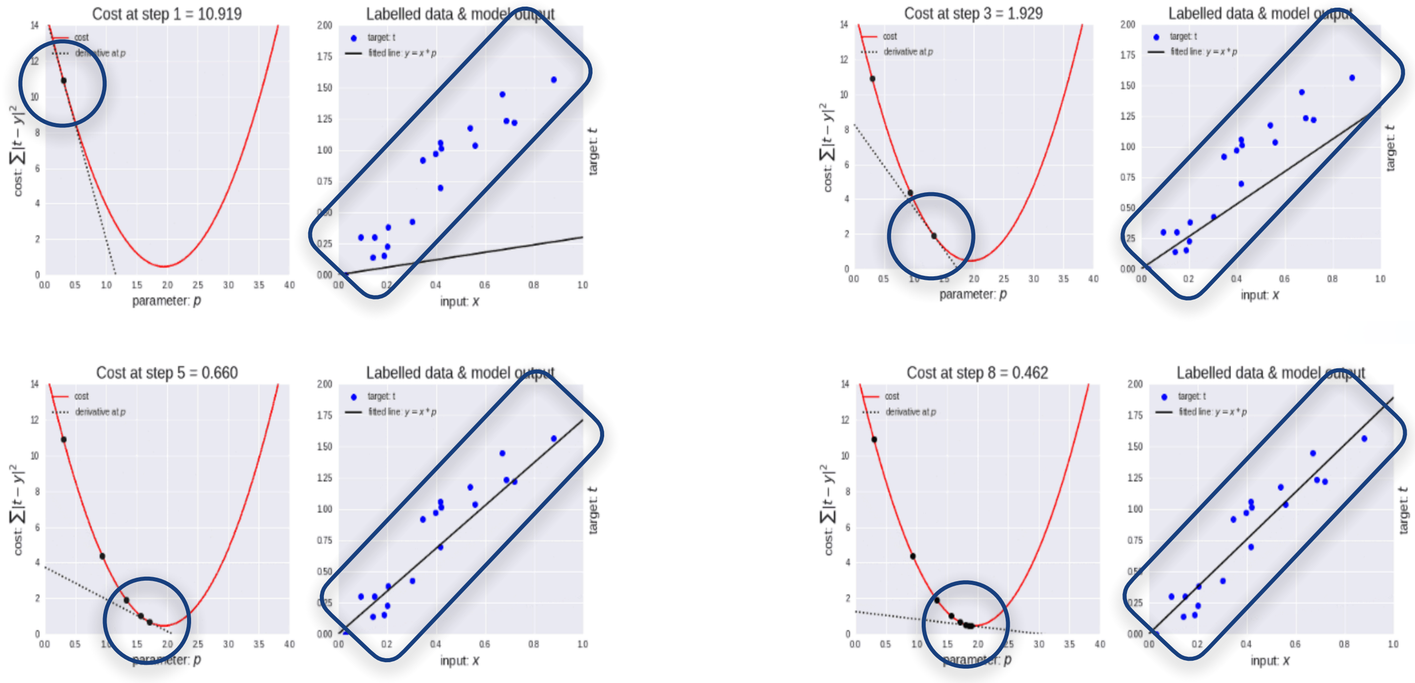
Gradient Descent

- Gradient Descent is an optimization technique to minimize a loss function by iteratively moving in the direction of the loss function's gradient
- It determines the amount of changes to be made to the Weights and Biases for a better prediction (i.e. smaller error) in the next iteration



Loss Function

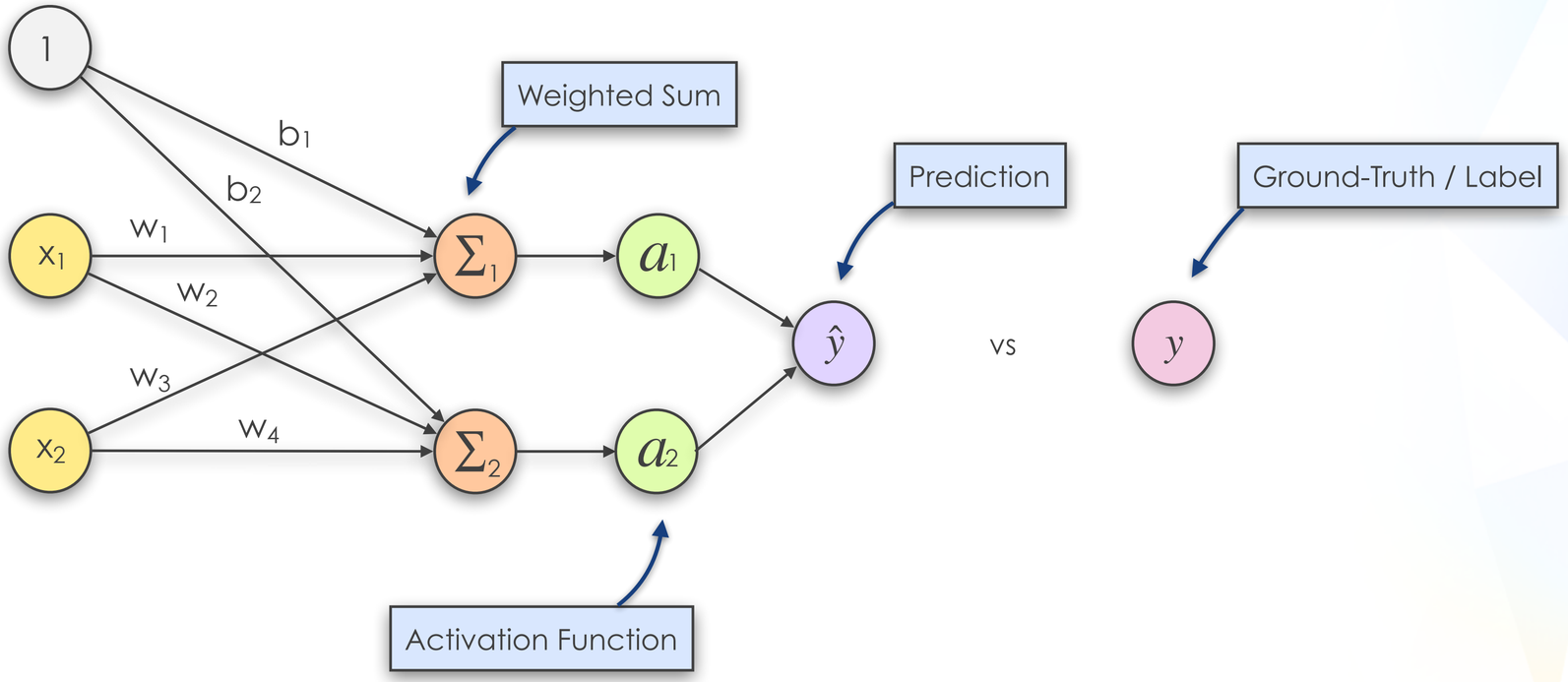
- An illustration to find a line that best fits the set of points by minimizing the MSE loss function



Backpropagation

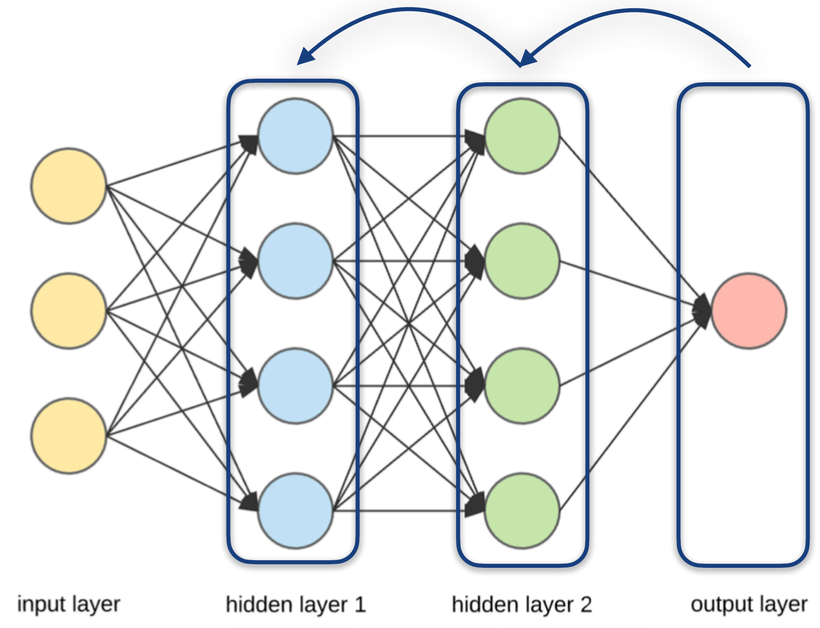
Error

- The difference between the predicted value and the ground-truth is known as the Error



Backpropagation

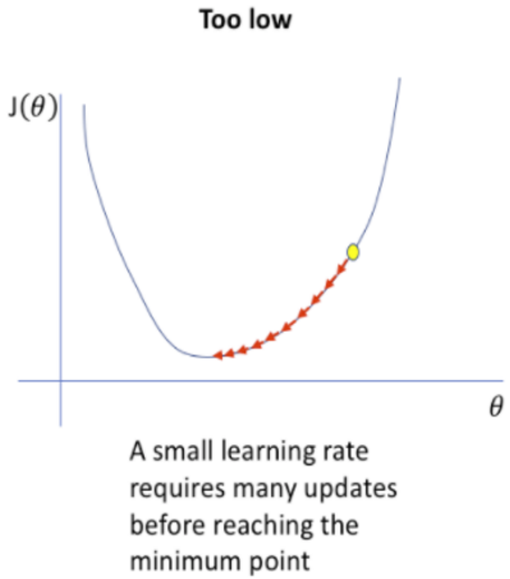
- Backpropagation is the backward propagation of small changes in weights and biases within a Neural Network
- Backpropagation is when “learning” takes place, and only happens when a Neural Network is being trained



A Simple Neural Network

Learning Rate

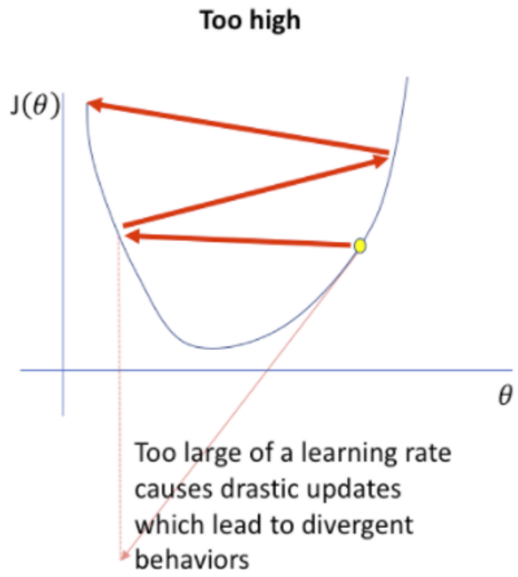
- The learning rate controls the amount of adjustments made to our weights and biases in each backpropagation
- A learning rate that is too small causes a neural network to learn slowly



Source: https://www.jeremyjordan.me/nn-learning-rate/

Learning Rate

- A learning rate that is too large causes a neural network to yield losses that swing wildly from one iteration to the next



Source: https://www.jeremyjordan.me/nn-learning-rate/

Learning Rate

- An optimal learning rate allows a neural network to learn steadily and swiftly

The optimal learning rate swiftly reaches the minimum point

Source: https://www.jeremyjordan.me/nn-learning-rate/

Fitting a Model

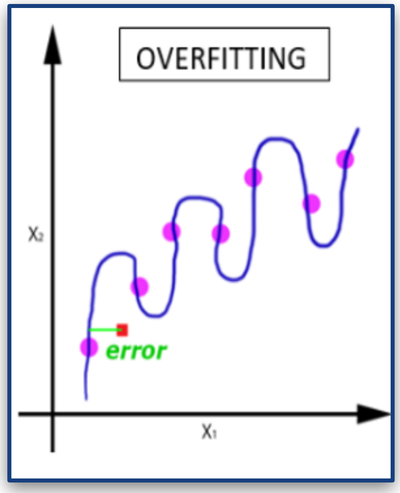
Data

- Training Data is used to train our model over multiple epochs
  - Epochs are the number of times a model has seen the entire set of data
  - Model updates its weights and biases from its prediction errors using these data (back-propagation takes place)

- Validation and Test Data are used to access the accuracy of our model
  - Model does not learn using these data (no back-propagation during validation and testing phase)

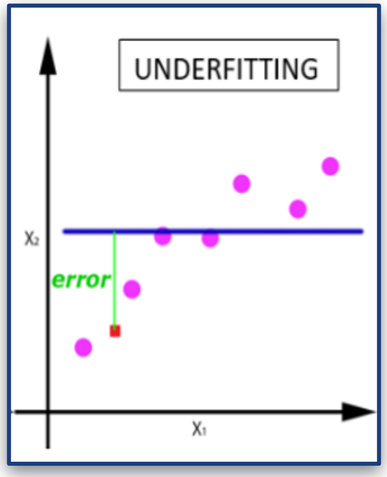
Over-Fitting

- If our model gives good predictions on training data, but bad predictions on validation and/or test data, then it is over-fitting
- To resolve this, use more and better training data, or apply regularization (add an error term) during backpropagation



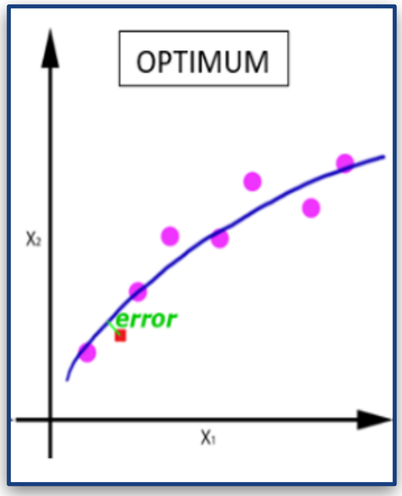
Under-Fitting

- If our model predicts badly in training, validation and/or test data, it is under-fitting
- To resolve this, train our model longer, add more hidden layers to it or feed it with more and better training data



Optimal Fitting

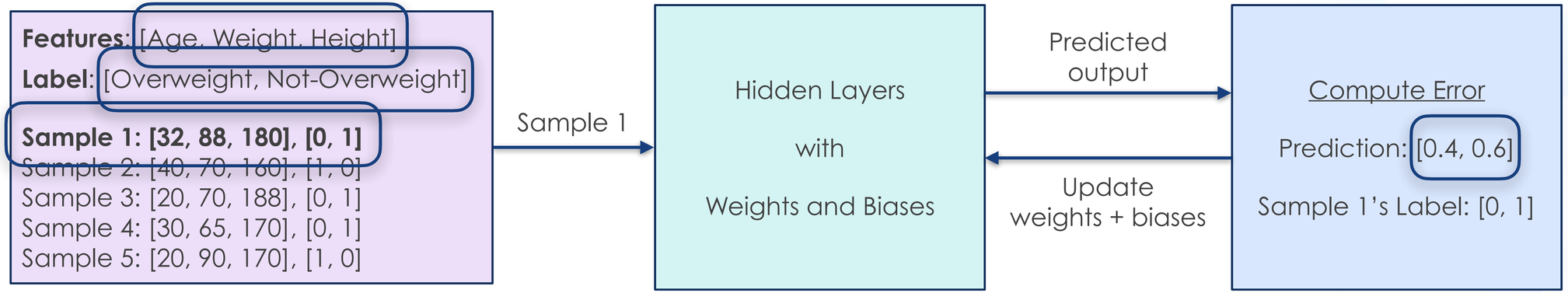
- If a model predicts correctly for its training, validation and/or test data, it has achieved optimal fitting
- Our model is general enough to accommodate new points that follows the distribution of the training data



Training and Testing

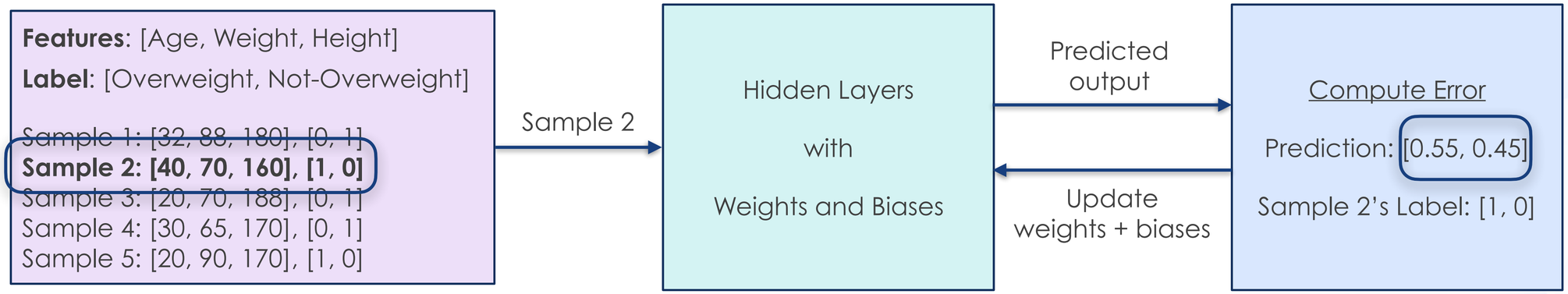
Training a Neural Network

- Train a Neural Network to predict if a person is overweight starting with Training Sample 1



Training a Neural Network

- Train a Neural Network to predict if a person is overweight using the next training-sample and so on



Using a trained Neural Network

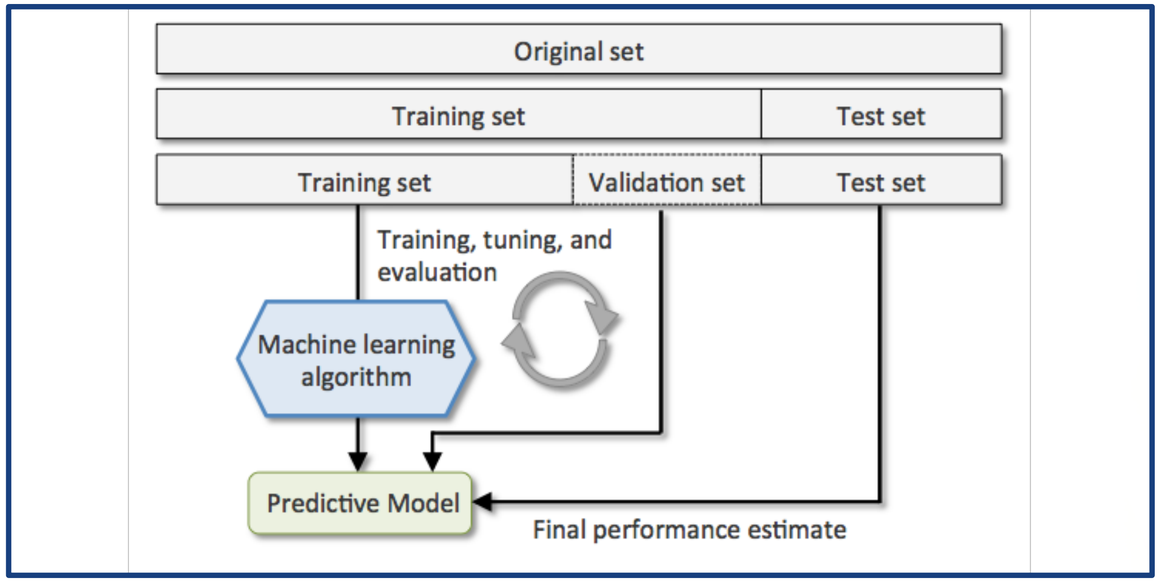
- Using the trained Neural Network to predict if a person is overweight using Test Sample 1

Using a trained Neural Network

- Using the trained Neural Network to predict if a person is overweight using subsequent test samples

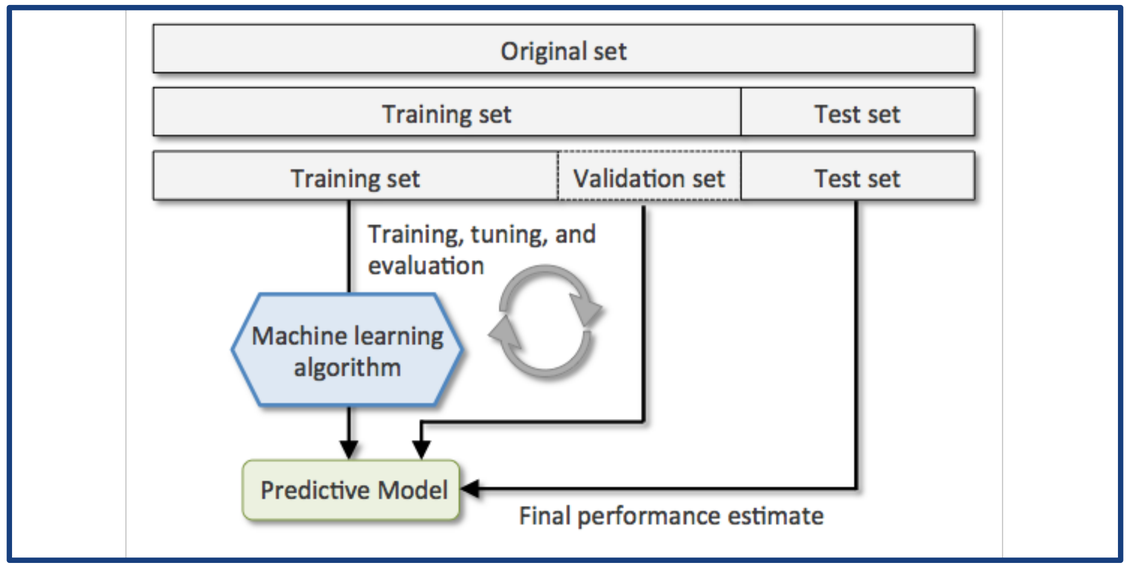
Training Data

- Training Data is used to train our Neural Network to give accurate predictions over multiple epochs (an epoch is when the Neural Network has seen all training data once)
- Weights and Biases are updated during backpropagation (learning)
- Objective is to minimize error (recall "loss function") and train a set of final weights and biases that gives accurate predications



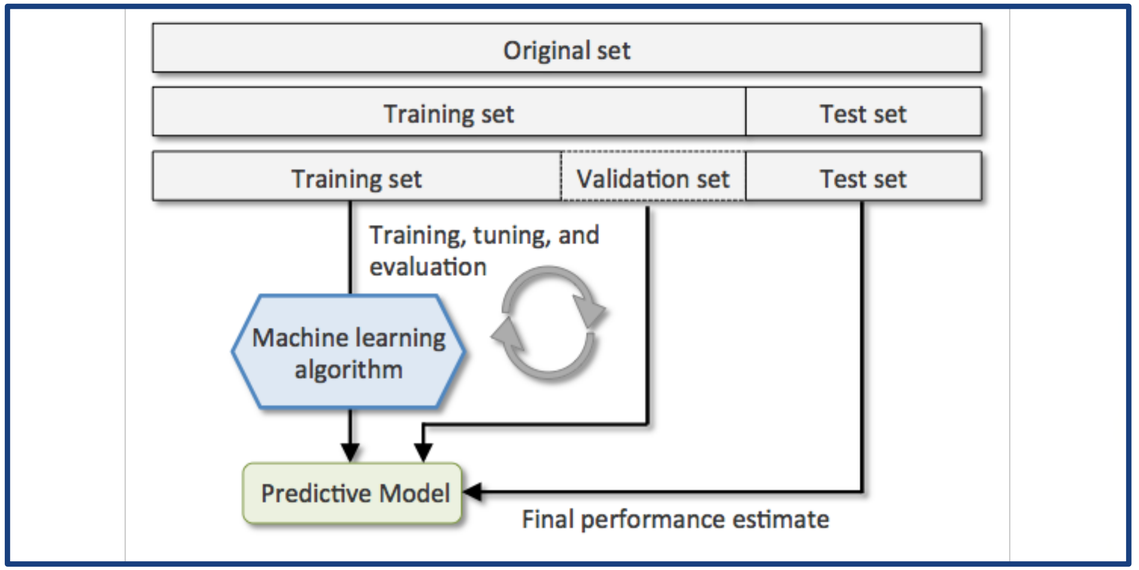
Validation data

- Validation Data is used to assess how accurate our Neural Network is, based on our training so far
- Only forward pass (i.e. prediction); no backpropagation and updating of weights and biases (i.e. no learning takes place)
- Objective is to compare the accuracies of our Neural Networks that have different architectures



Test data

- Test Data is to gauge the accuracy of a trained neural network
- Data points in the training data are usually excluded from the test data
- As a neural network has not seen the Test Data before, Test Data give us an unbiased estimate of its predictive accuracy



Tensorflow and Keras

TensorFlow

- TensorFlow, developed by Google, is a software library for machine learning
- Under the hood, we will leverage on TensorFlow to create, train and test our neural networks (or models)
- Keras is the high-level API of TensorFlow 2 and is a more approachable interface to work with TensorFlow

Keras (Adding example)

- Let's now create a neural network and train it to count 3 numbers
- So, we want to train a neural network by showing it examples of adding 3 numbers
  - E.g. 2 + 3 + 4 = 9
  - E.g. 6 + 2 + 6 = 14
- And when training is over, we will give him 3 numbers
  - E.g. 7 + 10 + 3 = ?
- Our trained neural network should give us a value that is very close to 20

Keras (Adding example)

- First, create our training data by randomly generating tuples of 3 values

```
import numpy as np
import tensorflow as tf

# prepare training features
x_train = np.random.randint(-100, 100, size=(300, 3))
print("x_train.shape =", x_train.shape)
print("x_train =", x_train)
```



```
x_train.shape = (300, 3)
x_train = [[  39  -23  -75]
 [  58  -96  -43]
 [   4   25  -20]
 [   4   11   27]
 [ -35  -23   34]
 [  50   94  -90]
 [  52   44   52]
 [  63  -65   35]
 [  83  -57  -90]
 [ -61  -43  -59]
 [  -9   54   37]
```

Keras (Adding example)

- Next, compute the sums of generated tuples
- These values are the “labels” to our samples

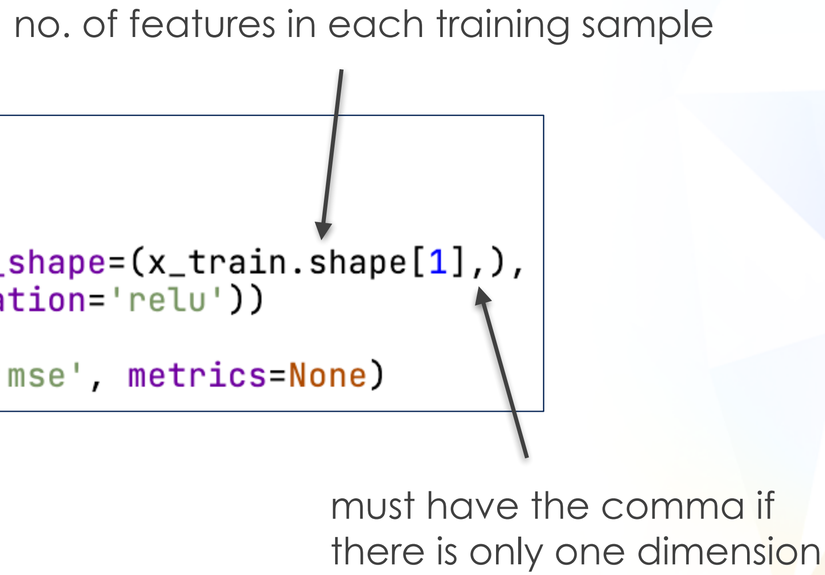
```
y_train = np.array([[sum(x)] for x in x_train])
print("y_train.shape =", y_train.shape)
print("y_train =", y_train)
```

```
y_train.shape = (300, 1)
y_train = [[ -59]
 [ -81]
 [   9]
 [  42]
 [ -24]
 [  54]
 [ 148]
 [  33]
 [ -64]
 [-163]
 [  82]
```

Keras (Adding example)

- Create our model by using Sequential()
- Add hidden layers via Dense(…)
- The loss function is Mean Square Error

```
# construct neural network model
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(500,
                                input_shape=(x_train.shape[1],),
                                activation='relu'))
model.add(tf.keras.layers.Dense(1))
model.compile(optimizer='adam', loss='mse', metrics=None)
```



Keras (Adding example)

- Our model start training when fit(…) is called
- x_train contains training **features**
- y_train contains training **labels**
- Our Neural Network will be trained with our training **samples** 500 times (i.e. epochs=500)

```
# train the model
model.fit(x_train, y_train, epochs=500, verbose=1)
```

Keras (Adding example)

- Once our training has completed, create some test data to test our trained neural network
- Use evaluate(…) to determine the loss (the difference between the ground truth and prediction

```
# generate test data
x_test = np.random.randint(-100, 100, size=(20, 3))
y_test = np.array([[sum(x)] for x in x_test])

# perform auto-evaluation
loss = model.evaluate(x_test, y_test, verbose=1)
print('Loss = ', loss, '\n')
```

```
Loss =  0.026681125164031982
```

Keras (Adding example)

- Examine our model's predicted results

```
# perform prediction
predictions = model.predict(x_test)
for i in np.arange(len(predictions)):
    print('Data: ', x_test[i],
          ', Actual: ', y_test[i],
          ', Predicted: ', predictions[i])
```

```
Data:  [78 60 72] , Actual:  [210] , Predicted:  [209.98482]
Data:  [-29 -33  74] , Actual:  [12] , Predicted:  [12.3939495]
Data:  [72 16 98] , Actual:  [186] , Predicted:  [185.92139]
Data:  [98 53 88] , Actual:  [239] , Predicted:  [239.03915]
Data:  [-24  25 -41] , Actual:  [-40] , Predicted:  [-40.075893]
Data:  [ 47   3 -17] , Actual:  [33] , Predicted:  [32.941364]
Data:  [-77 -15  33] , Actual:  [-59] , Predicted:  [-58.995853]
Data:  [-10  23  82] , Actual:  [95] , Predicted:  [94.7834]
Data:  [-57  61   1] , Actual:  [5] , Predicted:  [4.686406]
Data:  [ -1 -63  38] , Actual:  [-26] , Predicted:  [-26.01451]
Data:  [ 68   3 -94] , Actual:  [-23] , Predicted:  [-23.03826]
Data:  [-66   1  91] , Actual:  [26] , Predicted:  [26.239145]
Data:  [-95 -45  59] , Actual:  [-81] , Predicted:  [-80.65134]
Data:  [74 37 11] , Actual:  [122] , Predicted:  [121.929184]
Data:  [-100  -24   68] , Actual:  [-56] , Predicted:  [-55.926502]
Data:  [-11 -70 -69] , Actual:  [-150] , Predicted:  [-150.02951]
Data:  [-11 -58 -52] , Actual:  [-121] , Predicted:  [-121.001015]
Data:  [ -8  53 -57] , Actual:  [-12] , Predicted:  [-12.055797]
Data:  [-63 -72 -70] , Actual:  [-205] , Predicted:  [-204.99951]
Data:  [-37  86 -78] , Actual:  [-29] , Predicted:  [-29.145395]
```

The End