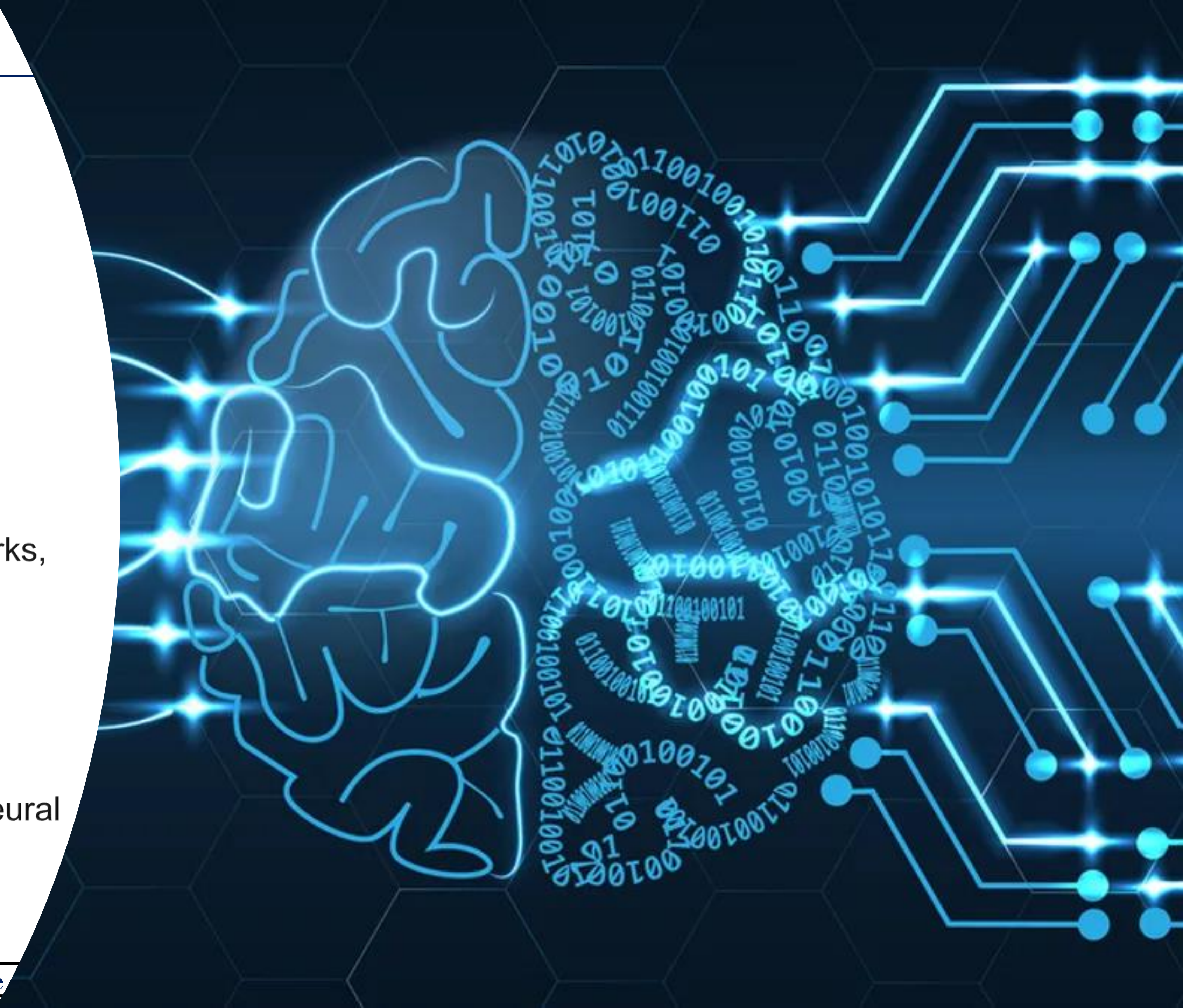
Deep Neural Networks

Loss Function, Building Neural Networks, Cross Entropy loss,

Last Lecture:

Perceptron model, Fully Connected Neural Network Activation function,



Loss Function

What is Loss Function?

Loss function is a method of evaluating how well your algorithm is modelling your dataset.

Loss Function value	Algorithm Performance
High	Poor
Low	Great

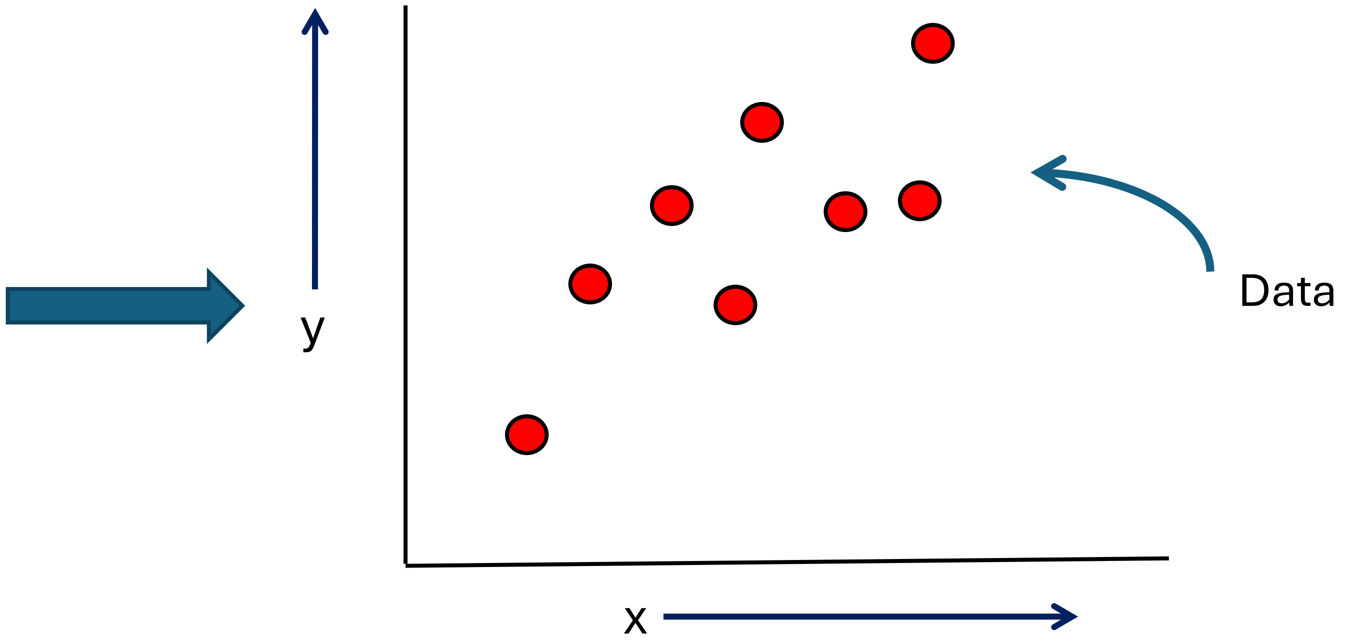
Loss Function

Why Loss Function is Important?

You can't improve what you can't measure

-Peter Druker (*Austrian American management consultant, educator, and author)*

Fitting of line in Linear Regression

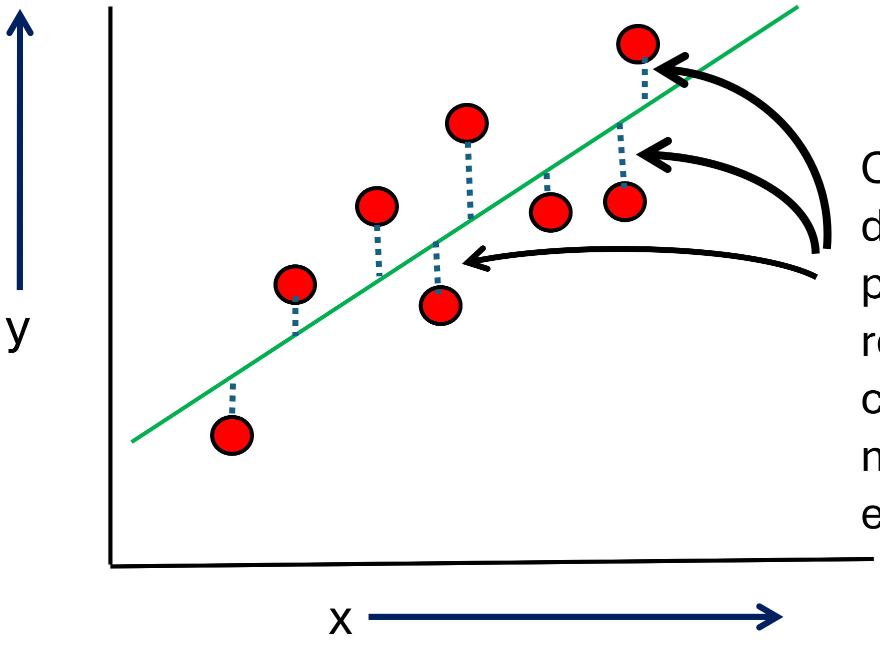


Loss Function

Root Mean Square Error (RMSE) Calculation

$$\sqrt{(d_1{}^2 + d_2{}^2 + d_3{}^2 + \ \dots\dots + d_n{}^2)} \quad = \quad 0.354 \quad \text{(assumption for line 1)}$$

Fitting of line in Linear Regression



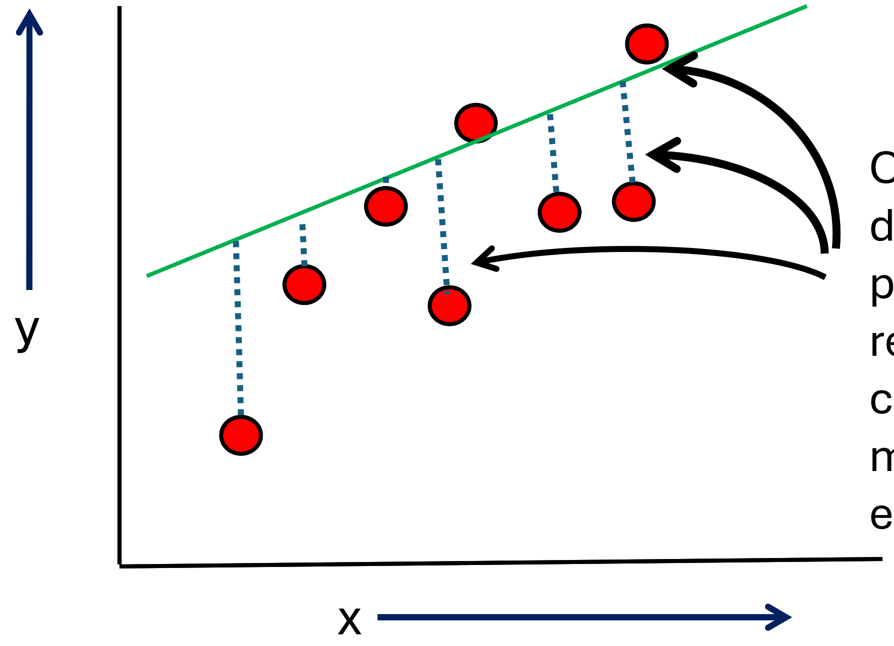
Calculation of distance of data point from a reference line to compute root mean square error

Loss Function

Root Mean Square Error (RMSE) Calculation

$$\sqrt{(d_1{}^2 + d_2{}^2 + d_3{}^2 + \; \ldots \ldots . + d_n{}^2)} \quad = \quad 1.454 \quad \text{(assumption for line 2)}$$

Fitting of line in Linear Regression

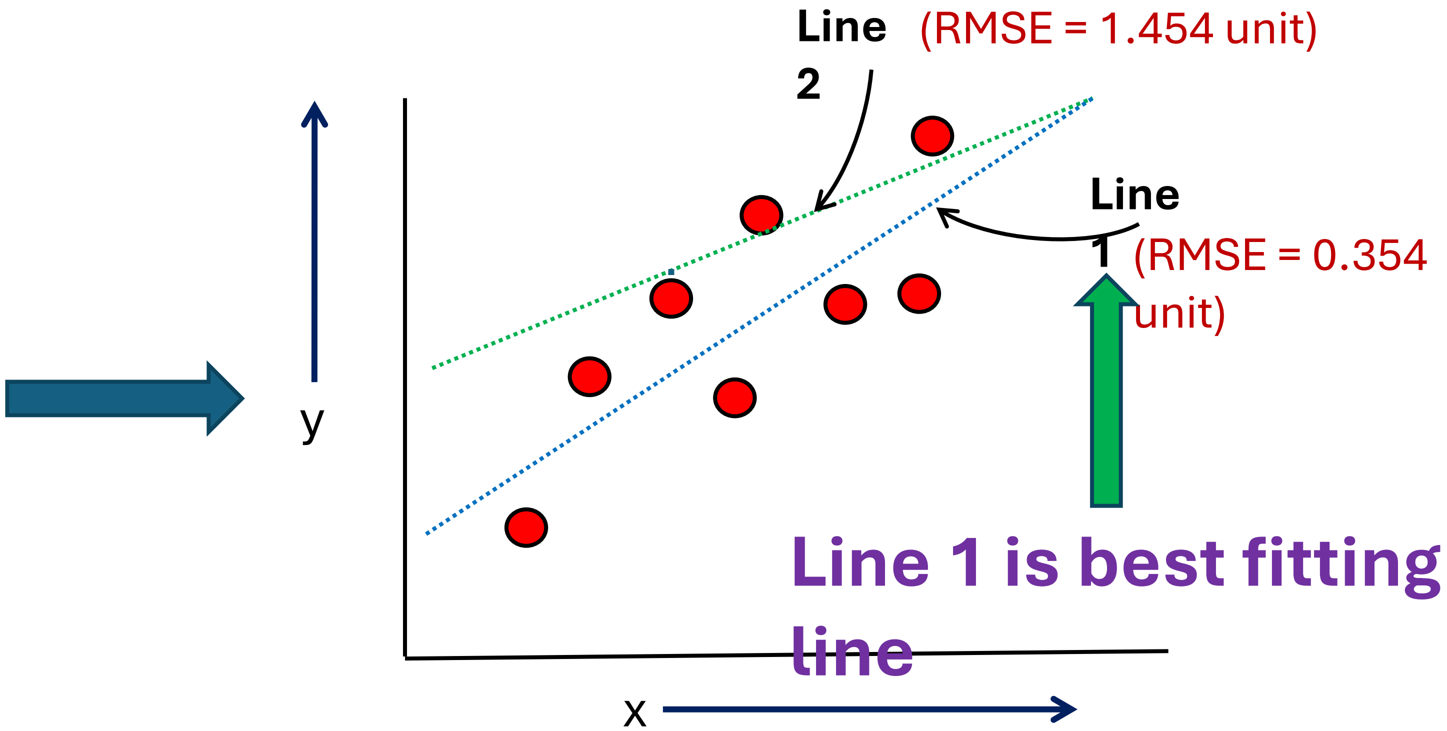


Calculation of distance of data point from a reference line to compute root mean square error

Loss Function

After rotating a line for multiple angles, RMSE is calculated for each line

Fitting of line in Linear Regression

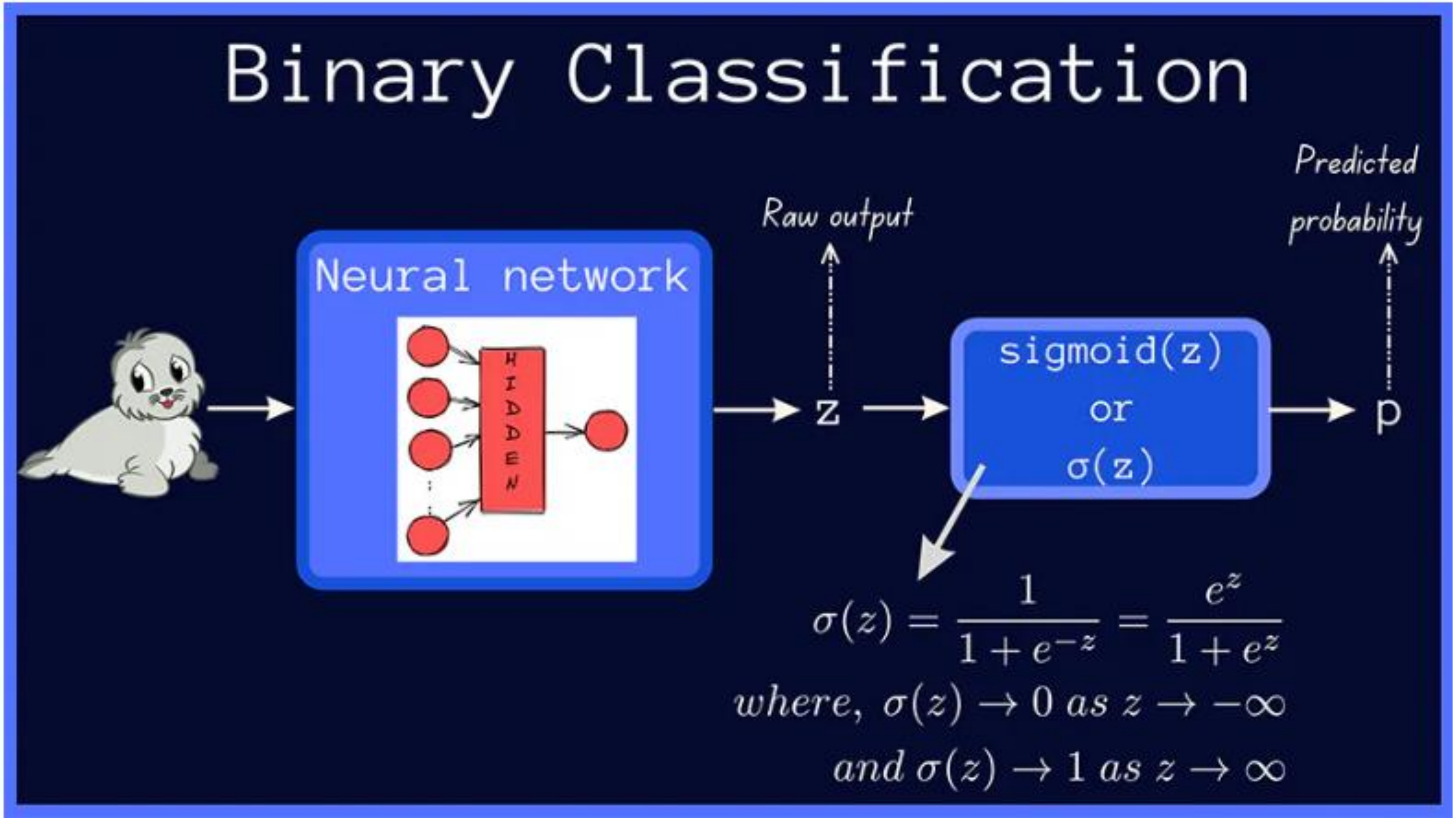


Loss Function

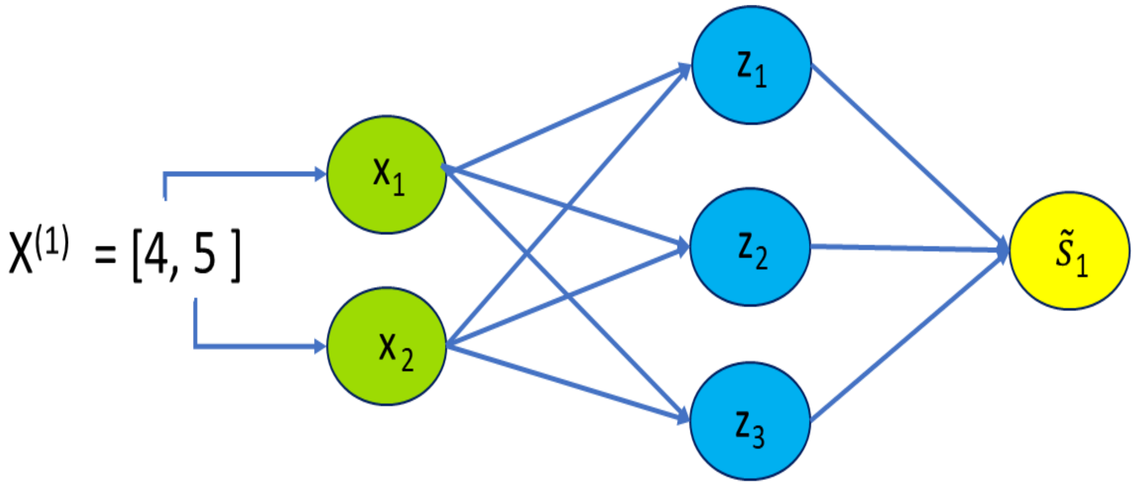
Objective of any Machine Learning/Deep Learning Model → Decrease the value of Loss Function

“Loss Function is the Eye of any AI algorithm”

- **Loss function** is a method of evaluating “how well the algorithm models on the input dataset”.
- If your predictions are totally off, your loss function will output a higher number. If they’re pretty good, it’ll output a lower number.
- As you tune your algorithm to try and improve your model, your loss function will tell you if you’re improving or not.
- ‘Loss’ helps us to understand how much the predicted value differ from actual value


Binary Classification
Raw output
Predicted probability
Neural network
HIDDEN
z
sigmoid(z)
or
σ(z)
p
$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$
$where,\ \sigma(z) \rightarrow 0\ as\ z \rightarrow -\infty$
$and\ \sigma(z) \rightarrow 1\ as\ z \rightarrow \infty$

Quantifying Loss

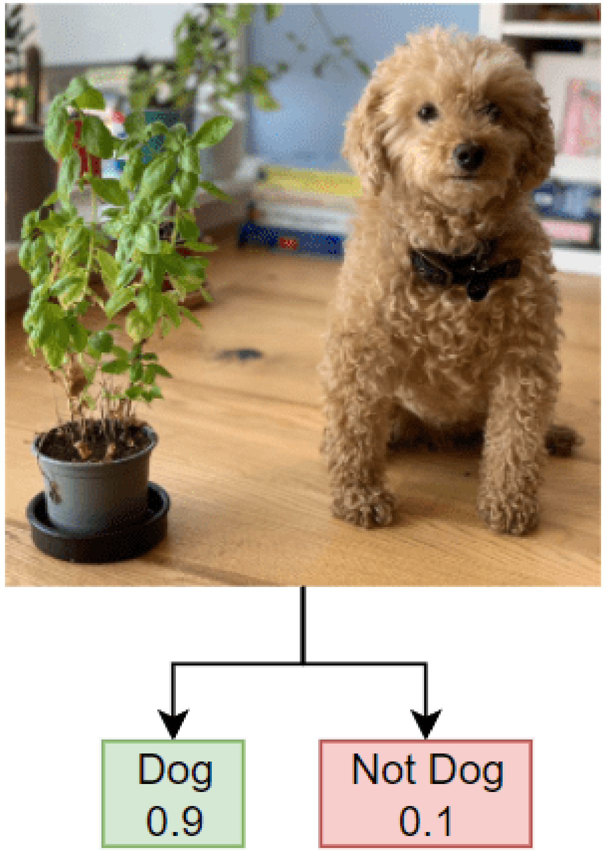


$$L\,(\underbrace{f\,(xi\,;\boldsymbol{W})}_{\text{Predicted}},\ \underbrace{s^i}_{\text{Actual}}\,)$$

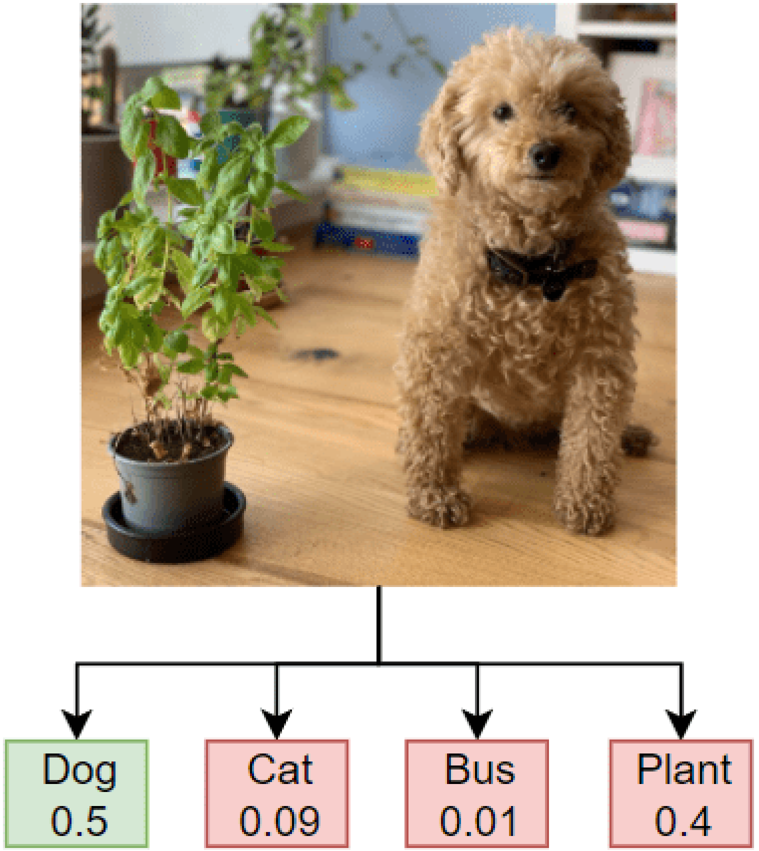
- To train the model to perform correct prediction we have to tell the network when it is wrong so we have to quantify its loss or its error.
- The loss of our network measures the cost incurred from incorrect predictions.
- ***How to quantify the loss***???
  - By comparing the prediction by network with the actual result.
  - If there is a big discrepancy between the prediction and the true result than there is some fault occurred.

Classification Task

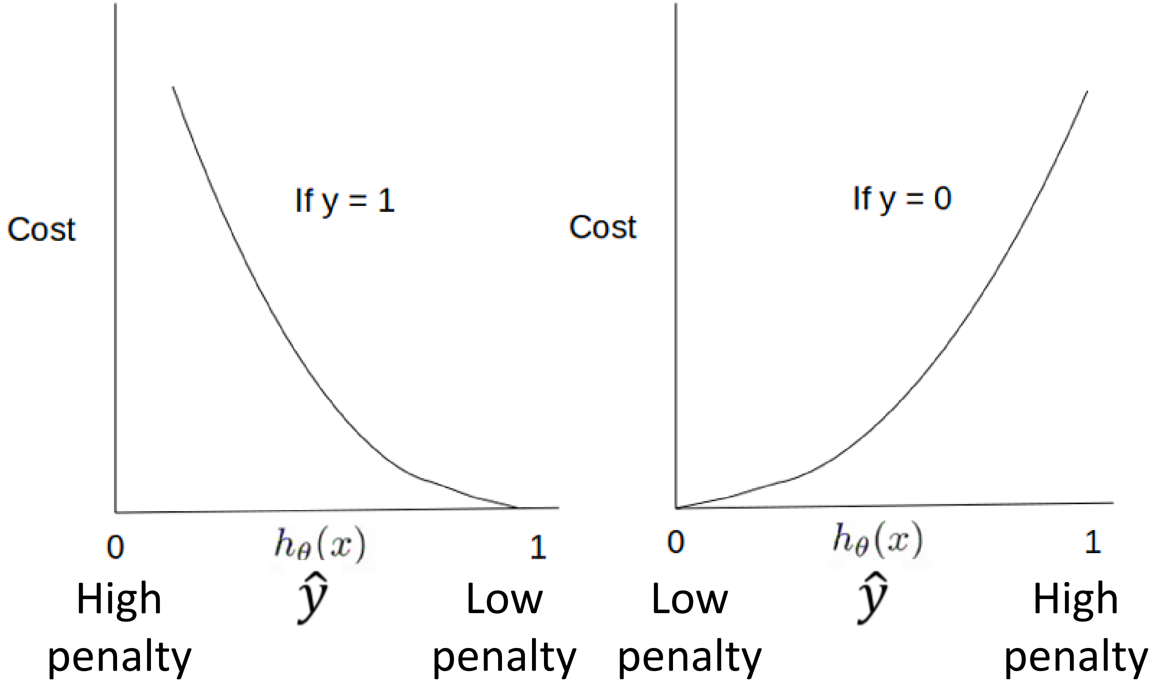
Binary Classification



Multiclass Classification



Binary cross-entropy/Log Loss



$$CE\ Loss = \frac{1}{n}\sum_{i=1}^{N} -\left(\underbrace{y_i}_{\text{Actual}} \cdot \underbrace{log(p_i)}_{\text{Predicted}} + (1 - \underbrace{y_i}_{\text{Actual}}) \cdot \underbrace{log(1 - p_i)}_{\text{Predicted}}\right)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

- Binary cross entropy compares each of the predicted probabilities to the actual class output which can be either 0 or 1.
- It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.
  - *Advantage* –A cost function is a differential.
  - *Disadvantage* –Multiple local minima, Not intuitive

Categorical cross-entropy

One-hot encoding

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

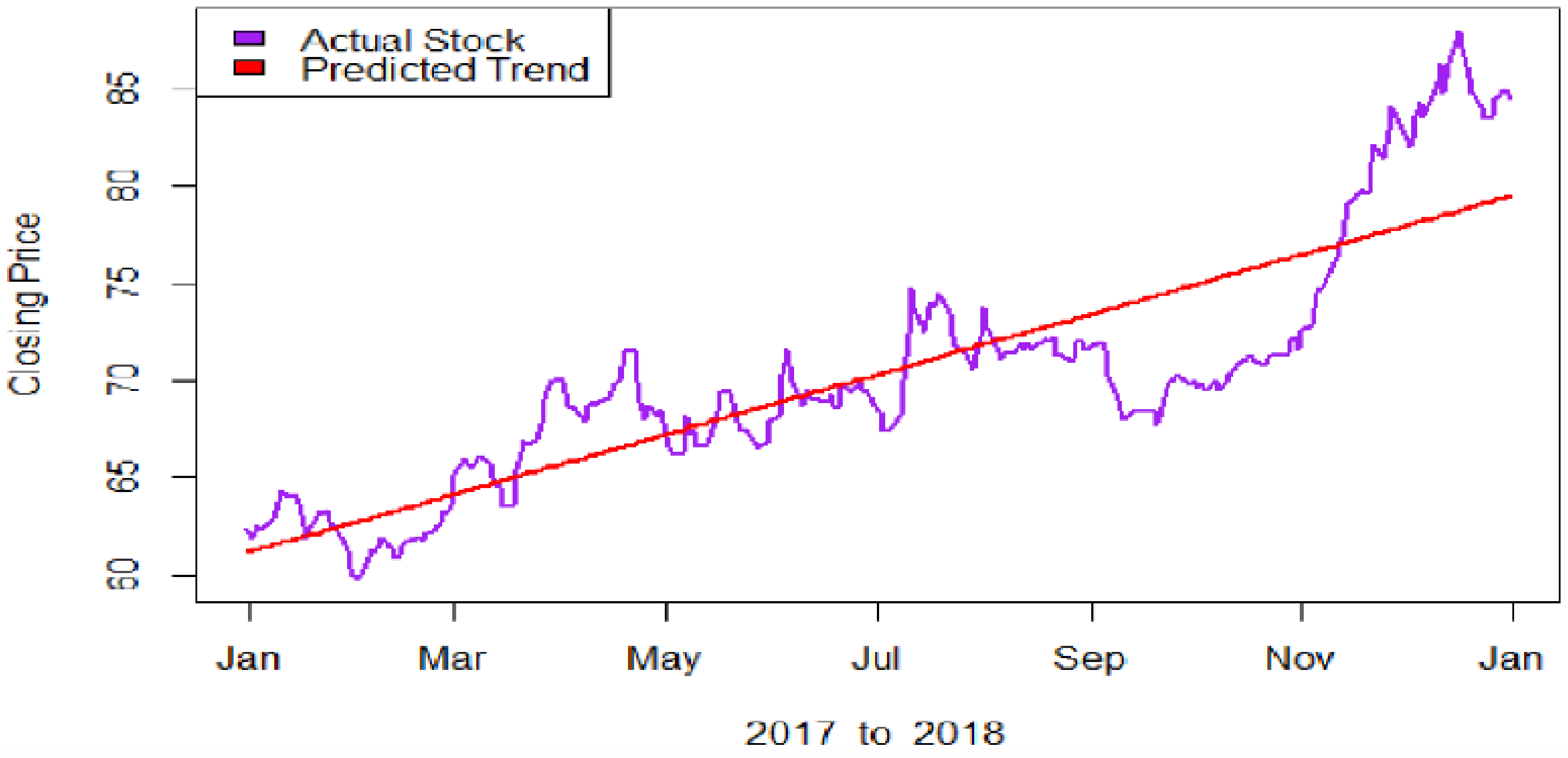
$$CE\ Loss = -\frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\cdot log(p_{ij})$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

- It is used for multi-class classification.
- In the specific (and usual) case of Multi-Class classification the labels are *one-hot encoded*.

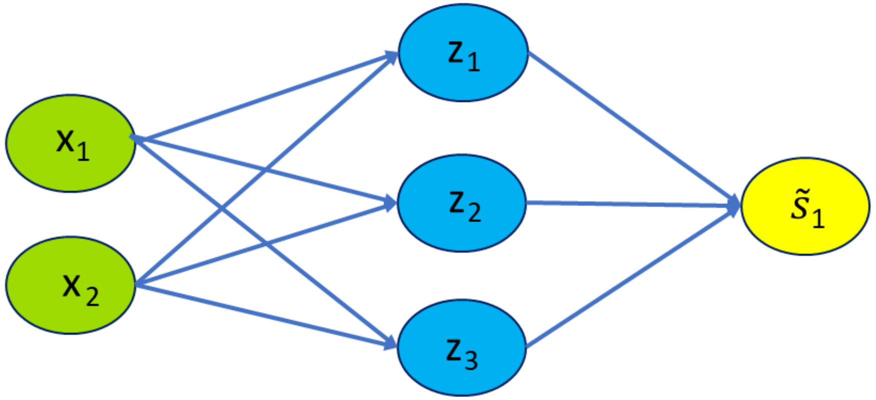
Let y=[1,0] and p=[0.8,0.2]
CE-loss= -1.log(0.8)+0.log(0.2)
=0.2231
Let y=[1,0] and p=[1,0]
CE-loss= -1.log(1)+0.log(0)
=0
Let y=[1,0] and p=[0,1]
CE-loss= -1.log(0)+0.log(1)
=-1. log e, where e=10^{-10}
=10

Regression Task



Mean Squared Error Loss

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{cc} f(X) & y \\ \begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} & \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix} \end{array}$$

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}(\underbrace{si}_{\text{Actual}} - \underbrace{f(xi;W)}_{\text{Predicted}})^2$$

- In case, players wants to know the score of the match regression is performed rather then classification.
- For such problem different type of loss is used, like "***mean squared error loss***" is preferred here.
- Difference between actual and predicted output is taken and their squared error is calculated for optimizing the loss.

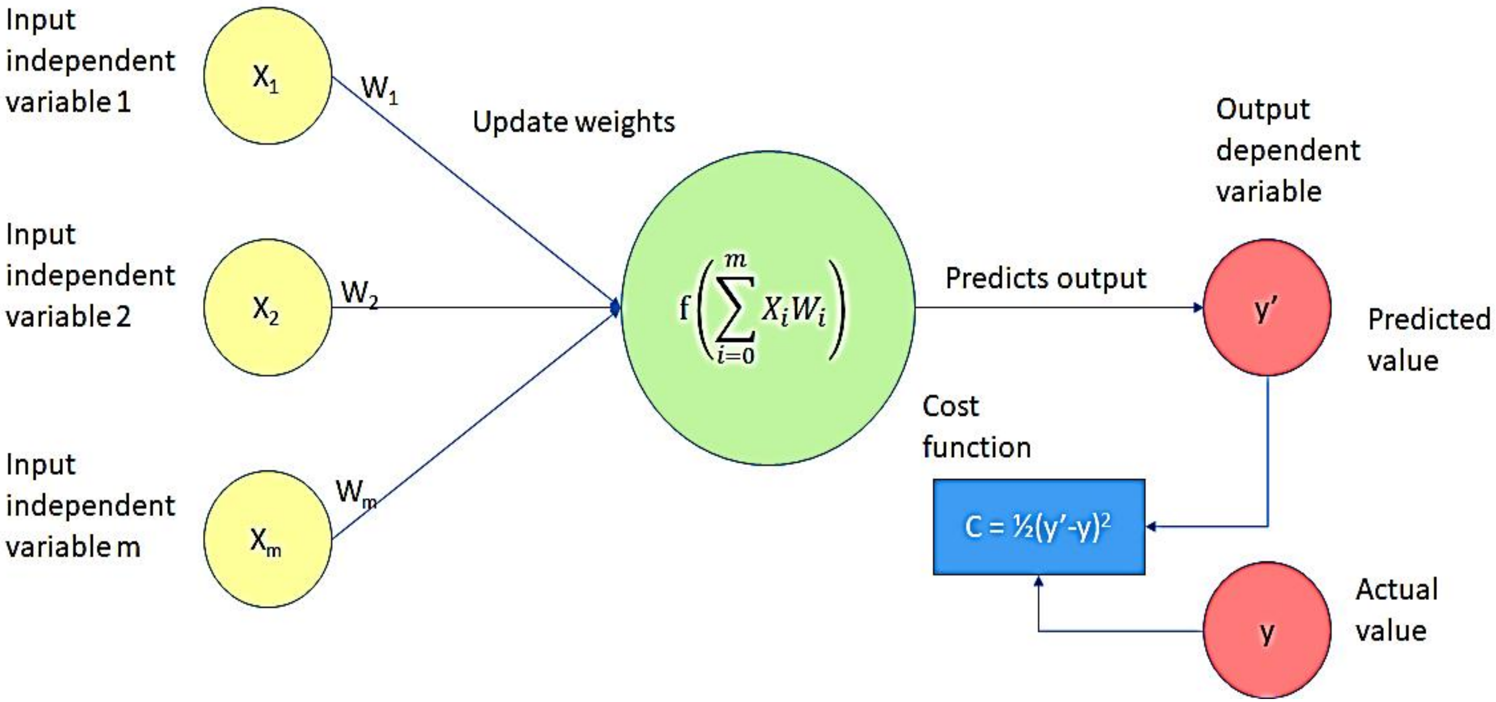
```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred)
```

Cost Function and Loss Function

The loss error is computed for a single training example. If we have ‘m’ number of examples then the average of the loss function of the entire training set is called ‘Cost function’.

Cost function (J) = 1/m (Sum of Loss error for ‘m’ examples)

The shape of cost function graph against parameters (W and b) is a cup up parabola with a single minimum value called ‘local optima’.



The **loss function** (or error) is for a **single training example**, while the **cost function** is over the **entire training set** (or mini-batch for mini-batch gradient descent).

Cost Function and Loss Function

Single Sample

Loss Function/ Error Function

Input			**Output** (Y)	**Model Prediction** ($\hat{Y}$)
Temperature (°C)	**Vegetation Index**	**Elevation (m)**	**Actual Annual Rainfall (mm)**	**Predicted Annual Rainfall (mm)**
30	0.8	500	1200	1100
25	0.7	600	800	750
35	0.6	700	1500	1600
28	0.9	450	1000	950

$\hat{Y}$-Y	$(\hat{Y}\text{-}Y)^2$
100	10000
50	2500
100	10000
50	2500

Entire Data

Cost Function $= \frac{1}{4}((\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + (\hat{y}_3 - y_3)^2 + (\hat{y}_4 - y_4)^2)$

$= \frac{1}{4}(10000 + 2500 + 10000 + 2500)$ =6250

Building Neural Network

Steps in building NN:

1. Load Data
2. Define Keras Model
3. Compile Keras Model
4. Fit Keras Model
5. Evaluate Keras Model
6. Tie It All Together
7. Make Predictions

1. Load Data

```
# first neural network with keras tutorial
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# 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

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

3. Compile Keras Model

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

4. Train Keras Model

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

5. Evaluate Keras Model

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

Putting all code together

```
# first neural network with keras tutorial
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# 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]
# define the keras model
model = Sequential()
model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Any Question ?

Thanks