X = np.array([-7,-5,-1,2,5,8])

y = np.array([3,6,9,12,15,18])

plt.scatter(X,y)

#Building Model

# Create a model using the Sequential API

model = tf.keras.Sequential([

tf.keras.layers.Input(shape=(1,)), #incase of showing dim errors

tf.keras.layers.Dense(1)

])

# Compile the model

model.compile(loss=tf.keras.losses.mae, # mae is short for mean absolute error

optimizer=tf.keras.optimizers.SGD(), # SGD is short for stochastic gradient descent

metrics=["mae"])

# Fit the model

model.fit(X, y, epochs=5)

Changing Hyperparameter

tf.random.set\_seed(42)

model = tf.keras.Sequential([

tf.keras.layers.Input(shape=(1,)),

tf.keras.layers.Dense(100,activation=None),

tf.keras.layers.Dense(1)

])

model.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.Adam(lr=0.01),

metrics=["mae"])

model.fit(X,y,epochs=100)

Adam optimizers help in reducing the mae

Evaluating model

X = tf.range(-100,100,4)

y = X+10

import matplotlib.pyplot as plt

plt.scatter(X,y)

Three Data Model

X\_train = X[:40]

X\_test = X[40:]

y\_train = y[:40]

y\_test = y[40:]

len(X\_train),len(X\_test),len(y\_train),len(y\_test)

Visualizing Data

plt.figure(figsize=(10,7))

plt.scatter(X\_train,y\_train,c="b",label="Train Set")

plt.scatter(X\_test,y\_test,c="g",label="Test set")

plt.legend()

Model Summary

tf.random.set\_seed(42)

model = tf.keras.Sequential([

tf.keras.layers.Dense(1,input\_shape=[1])

])

model.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.SGD(),

metrics=["mae"])

#model.fit(X\_train,y\_train,epochs=100)

model.summary()

summary() will throw error if input\_shape is not defined

* **Total params** - total number of parameters in the model.
* **Trainable parameters** - these are the parameters (patterns) the model can update as it trains.
* **Non-trainable parameters** - these parameters aren't updated during training (this is typical when you bring in the already learned patterns from other models during transfer learning).
* **\*\*Trainable parameters : Weights and biases\*\***\*

Trainable parameter: Weights and biases

Verbose

model.fit(X\_train,y\_train,epochs=100, verbose=0)

Plot Model

from tensorflow.keras.utils import plot\_model

plot\_model(model,show\_shapes=True)

tf.random.set\_seed(42)

model = tf.keras.Sequential([

tf.keras.layers.Dense(10,input\_shape=[1],name="input\_layer"),

tf.keras.layers.Dense(1,name="output\_layer")

],name="model\_1")

model.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.SGD(),

metrics=["mae"])

model.summary()

plot\_model(model,show\_shapes=True)

Visualize Models Predictions

def plot\_prediction(train\_data=X\_train,train\_labels=y\_train,

test\_data=X\_test,test\_labels=y\_test,

predictions=y\_preds):

plt.figure(figsize=(10,7))

plt.scatter(X\_train,y\_train,c="b",label="Train Data")

plt.scatter(X\_test,y\_test,c="r",label="Test Data")

plt.scatter(X\_test,y\_preds,c="g",label="Predictions")

plt.legend()

plot\_prediction(X\_train,y\_train,X\_test,y\_test,y\_preds)

Models Predictions with Evaluation Metrics

model.evaluate(X\_test,y\_test)

Mean absolute Error

tf.metrics.mean\_absolute\_error(y\_test,tf.constant(y\_preds))

y\_preds are in arrays, convert to tensors and squeeze as per y\_test shape

Mean Squared Error

mse = tf.metrics.mean\_squared\_error(y\_test,tf.squeeze(y\_preds))

mse

Function for MAE and MSE

def mae(y\_test,y\_pred):

return tf.metrics.mean\_absolute\_error(y\_test=y\_test,y\_pred=y\_pred)

def mse(y\_test,y\_pred):

return tf.metrics.mean\_squared\_error(y\_test=y\_test,y\_pred=y\_pred)

Create 3 different Models

tf.random.set\_seed(42)

model\_3 = tf.keras.Sequential([

tf.keras.layers.Dense(1,input\_shape=[1]),

tf.keras.layers.Dense(1,input\_shape=[1])

])

model\_3.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.SGD(),

metrics=["mae"])

model\_3.fit(X\_train,y\_train,epochs=500,verbose=0)

y\_preds\_3 = model\_3.predict(X\_test)

plot\_prediction(predictions=y\_preds\_3)

mae\_3 = mae(y\_test,y\_preds\_3)

mse\_3 = mse(y\_test,y\_preds\_3)

mae\_3, mse\_3

Comparing Models with Panda

model\_results = [["model\_1", mae\_1, mse\_1],

["model\_2", mae\_2, mse\_2],

["model\_3", mae\_3, mae\_3]]

import pandas as pd

all\_results = pd.DataFrame(model\_results,columns=["model","mae","mse"])

all\_results

Tracking Experiment

Weights and biases

Tensorboard

Saving Models

Saved Model Format

model\_3.save("best\_model\_3")

HDF5 format

model\_3.save("best\_model\_3.h5")

Loading Models

loaded\_model\_3 = tf.keras.models.load\_model("/content/best\_model\_3")

loaded\_model\_3.summary()

test\_y\_preds\_3 = model\_3.predict(X\_test)

loaded\_y\_preds\_3 = loaded\_model\_3.predict(X\_test)

test\_y\_preds\_3 == loaded\_y\_preds\_3

Downloading files from Colab

Through console

Through Code

from google.colab import files

files.download("/content/best\_model\_3")

By mounting drive to colab

!cp /content/best\_model\_3.h5 /content/drive/MyDrive/Colab-Files

Larger Example with Kaggle Dataset

import tensorflow as tf

import pandas as pd

import matplotlib.pyplot as plt

insurance = pd.read\_csv("https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-datasets/master/insurance.csv")

insurance.head()

Onehot encoding for Categorical features

insurance\_onehot = pd.get\_dummies(insurance)

insurance\_onehot.head()

Getting X & Y

X = insurance\_onehot.drop("charges",axis=1)

y = insurance\_onehot["charges"]

X.head()

Splitting X & Y

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=42)

len(X\_train),len(X\_test),len(X)

Create Model and Evaluate

tf.random.set\_seed(42)

insurance\_model = tf.keras.Sequential([

tf.keras.layers.Dense(10),

tf.keras.layers.Dense(1)

])

insurance\_model.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.SGD(),

metrics=["mae"])

insurance\_model.fit(X\_train,y\_train,epochs=100)

insurance\_model.evaluate(X\_test,y\_test)

Optimize Model With Adam Optimizer

If model is not getting trained and throwing nan then try using Adam optimizer

tf.random.set\_seed(42)

insurance\_model\_2= tf.keras.Sequential([

tf.keras.layers.Dense(100),

tf.keras.layers.Dense(10),

tf.keras.layers.Dense(1)

])

insurance\_model\_2.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.Adam(),

metrics=["mae"])

insurance\_model\_2.fit(X\_train,y\_train,epochs=100,verbose=1)

insurance\_model\_2.evaluate(X\_test,y\_test)

Fit model with history

tf.random.set\_seed(42)

insurance\_model\_3 = tf.keras.Sequential([

tf.keras.layers.Dense(100),

tf.keras.layers.Dense(10),

tf.keras.layers.Dense(1)

])

insurance\_model\_3.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.Adam(),

metrics=["mae"])

history=insurance\_model\_3.fit(X\_train,y\_train,epochs=200,verbose=0)

Plot History(Loss curv or training curv)

pd.DataFrame(history.history).plot()

plt.ylabel("Epochs")

plt.xlabel("Losses")

Early Stopping Callback::: TBC

Preprocess Data(Normalization and Standardization)

insurance = pd.read\_csv("https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-datasets/master/insurance.csv")

insurance["bmi"].plot(kind="hist")

Use Scikit learn to Normalize and Onehot encode the Data

from sklearn.compose import make\_column\_transformer

from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from sklearn.model\_selection import train\_test\_split

ct = make\_column\_transformer(

(MinMaxScaler(),["age","bmi","children"]),

(OneHotEncoder(handle\_unknown="ignore"),["sex","smoker","region"])

)

X = insurance.drop("charges",axis=1)

y = insurance["charges"]

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=42)

ct.fit(X\_train)

X\_train\_normal = ct.transform(X\_train)

X\_test\_normal = ct.transform(X\_test)

X\_train\_normal[0]

X\_train.loc[0]

X\_train.shape, X\_train\_normal.shape

Create a model with normalize data and Evaluate

tf.random.set\_seed(42)

insurance\_model\_4 = tf.keras.Sequential([

tf.keras.layers.Dense(100),

tf.keras.layers.Dense(10),

tf.keras.layers.Dense(1)

])

insurance\_model\_4.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.Adam(),

metrics=["mae"])

insurance\_model\_4.fit(X\_train\_normal,y\_train,epochs=100)

insurance\_model\_4.evaluate(X\_test\_normal,y\_test)