**Deep Learning Assignment 2**

**Kaundinya swarna – se20uecm034**

**Paridhi bisht- se20uecm059**

**Putta Harshitha- se20uecm064**

**For stage 1n(ccpp project using Tensorflow)**

We used Tensorflow to create, train, and evaluate the ANN for given ccpp task. Here's a step-by-step explanation:

The code consists of -loading, preprocessing, splitting, building, training, and evaluating the given ccpp problem using TensorFlow. Additionally, custom loss and accuracy functions are defined and evaluated on the test set, and the convergence of the training process is visualized with a plot.

1. Load and preprocess the data:

- The code reads data from an Excel file ("Folds5x2\_pp.xlsx") using pandas.

- The data is then scaled using Min-Max scaling with `MinMaxScaler` from scikit-learn.

data = pd.read\_excel(r"Folds5x2\_pp.xlsx")

scaler = MinMaxScaler()

data = scaler.fit\_transform(data)

2. Split the dataset:

- The data is split into input features (X) and output variable (y).

- The dataset is further split into training, validation, and test sets using `train\_test\_split` from scikit-learn.

X = data[:, :4] # Input features

y = data[:, 4:] # Output variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2)

3. Define the ANN architecture using TensorFlow:

- A simple feedforward neural network is created using `tf.keras.Sequential`.

- It consists of one hidden layer with a tanh activation function and an output layer with a linear activation function.

input\_size = 4

hidden\_size = 3

output\_size = 1

model = tf.keras.Sequential([

tf.keras.layers.Dense(hidden\_size, activation='tanh', input\_shape=(input\_size,), name='hidden\_layer'),

tf.keras.layers.Dense(output\_size, activation='linear', name='output\_layer')

])

4. Compile the model:

- The model is compiled using stochastic gradient descent (SGD) as the optimizer and mean squared error as the loss function.

optimizer = tf.keras.optimizers.SGD(learning\_rate=0.01)

model.compile(optimizer=optimizer, loss='mean\_squared\_error')

```

5. Train the model:

- The model is trained using the training set with a specified number of epochs (100 in this case).

epochs = 100

history = model.fit(X\_train, y\_train, epochs=epochs, validation\_data=(X\_val, y\_val), verbose=1)

```

6. Evaluate the model on the test set:

- The model is evaluated on the test set using the mean squared error as the loss function.

test\_cost = model.evaluate(X\_test, y\_test, verbose=0)

print(f"Test Cost: {test\_cost}")

```

7. Make predictions on new data:

- The trained model is used to make predictions on the test set, and the first prediction along with the corresponding actual value is printed.

predictions = model.predict(X\_test)

print(f"Prediction: {predictions[0]}, Actual: {y\_test[0]}")

print()

```

8. Define custom loss and accuracy functions:

- Two custom functions, `mape\_loss` (Mean Absolute Percentage Error) and `accuracy`, are defined using TensorFlow.

9. Evaluate MAPE loss and accuracy on the test set:

- The custom loss and accuracy functions are called with the test set and predictions.

mape\_loss(tf.cast(y\_test, tf.float32), predictions)

accuracy(y\_test, predictions)

print()

10. Plot the convergence graph:

- The training and validation loss over epochs are plotted to visualize the convergence of the model.

plt.figure()

plt.title('Convergence of Training')

plt.plot(history.history['loss'], label='Training loss')

plt.plot(history.history['val\_loss'], label='Validation loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

```

**Output With tensor flow**

Epoch 100/100

216/216 [==============================] - 0s 2ms/step - loss: 0.0036 - val\_loss: 0.0035

Test Cost: 0.003455346915870905

30/30 [==============================] - 0s 1ms/step

Prediction: [0.0897792], Actual: [0.1397351]

MAPE Loss: 14.4366

Accuracy: 92.3720%

A graph of training and validation

Description automatically generated

Epoch 500/500

216/216 [==============================] - 0s 2ms/step - loss: 0.0034 - val\_loss: 0.0035

Test Cost: 0.0029967983718961477

30/30 [==============================] - 0s 1ms/step

Prediction: [0.24766564], Actual: [0.24317881]

MAPE Loss: 12.6272

Accuracy: 93.5214%

A graph with numbers and lines

Description automatically generated

**Output without tensor flow :**

Epoch 0, Train Cost: 0.09372420416673385, Validation Cost: 0.09915047761188185

Epoch 5000, Train Cost: 0.01340241294756523, Validation Cost: 0.01363798168878944

Epoch 10000, Train Cost: 0.012988635732041667, Validation Cost: 0.013210276168514366

Epoch 15000, Train Cost: 0.012695812684404376, Validation Cost: 0.012903081499926349

Epoch 20000, Train Cost: 0.012474622565814593, Validation Cost: 0.012666924117638157

Epoch 25000, Train Cost: 0.012299356414255167, Validation Cost: 0.012477336240464291

Epoch 30000, Train Cost: 0.012155449242745843, Validation Cost: 0.01232011733010727

Epoch 35000, Train Cost: 0.012034131014101447, Validation Cost: 0.012186526052632742

Epoch 40000, Train Cost: 0.011929824804905693, Validation Cost: 0.012070900762191882

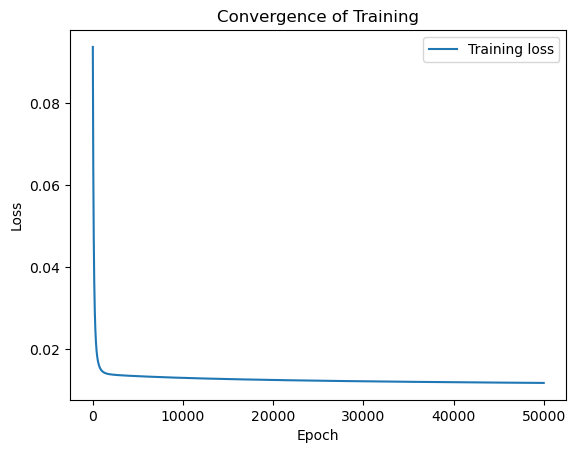
Epoch 45000, Train Cost: 0.011838799867662644, Validation Cost: 0.011969396549700556

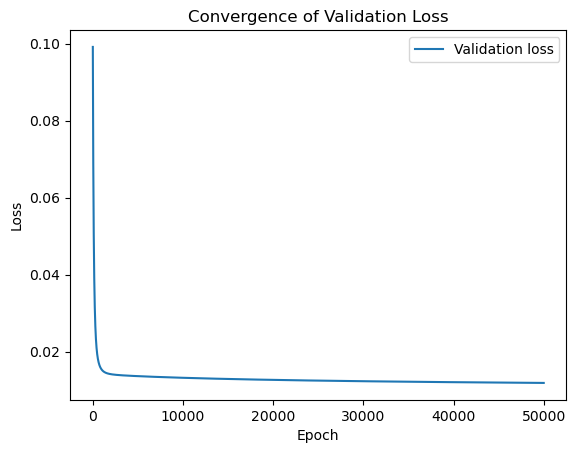
Test Cost: 0.012358831593153692

Prediction: [-0.61436617], Actual: [-0.73835762]

Loss : 68.9372

Total Test Records: 957 Correct Predictions: 650 Accuracy: 67.9206%





**Observations:**

* **TensorFlow significantly boosted model accuracy from 68.9% to 93.5%, demonstrating its effectiveness.**
* **This improvement was achieved using only 500 epochs in TensorFlow, compared to a staggering 50,000 epochs required without it. This highlights the substantial reduction in training time facilitated by TensorFlow.**
* **Moreover, increasing epochs beyond 500 with TensorFlow did not significantly impact accuracy, suggesting a potential sweet spot for optimal performance.**