

# 2024

**RAYYAN MASOOD.**

Enrollment no: **01-134212-151**

Class: BSCS 6-C

Date: May 24 ,2024



**[“AI LAB”]**

**[“ASSIGNMENT NO:11”]**

# Lab 11

## Software Installation:

To begin, you will need to install Python and TensorFlow on your computer. Follow the official documentation for TensorFlow installation instructions based on your operating system. Make sure to install the compatible version of TensorFlow that supports the functionalities required for the lab.

Use the following command to install TensorFlow using pip in Jupyter notebook:

```
!pip install tensorflow
```

## Dataset Preparation:

For this lab, you can either use an existing dataset or create a custom dataset. If using an existing dataset, ensure it is compatible with TensorFlow. If creating a custom dataset, generate the data and organize it into appropriate training and testing sets. Perform any necessary preprocessing steps such as normalization, scaling, or one-hot encoding.

```
import tensorflow as tf

# Example code for generating data

from sklearn.datasets import make_classification

# Generate a synthetic classification dataset

X, y = make_classification(n_samples=1000, n_features=10, n_classes=2)
```

## Building a Multilayer Perceptron Model:

### Network Architecture:

Define the structure of your MLP model using TensorFlow. Specify the number of layers, the number of neurons in each layer, and the activation functions to be used. TensorFlow provides various layers, such as Dense, that you can stack to create the desired architecture.

```
model = tf.keras.Sequential([

    tf.keras.layers.Dense(64, activation='relu', input_shape=(input_dim,)),

    tf.keras.layers.Dense(64, activation='relu'),

    tf.keras.layers.Dense(num_classes, activation='softmax')])
```

## Activation Functions:

In TensorFlow, you can choose from a range of activation functions. For example, `tf.nn.sigmoid`, `tf.nn.relu`, and `tf.nn.tanh` are commonly used activation functions. Select the appropriate activation functions based on the requirements of your model.

```
model.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
```

## Forward Propagation:

In TensorFlow, the forward propagation step involves passing the inputs through the layers of the MLP model. Use TensorFlow's `tf.keras.Sequential` model or the lower-level TensorFlow API to define the forward propagation process.

```
output = model(X)
```

## Backpropagation Algorithm:

TensorFlow automatically handles the backpropagation algorithm when training the model. During the training process, TensorFlow calculates the gradients and performs weight updates using the optimization algorithms specified.

## Weight Initialization:

When using TensorFlow, the default weight initialization schemes are typically appropriate. However, if desired, you can customize weight initialization by specifying different initialization techniques using TensorFlow's initializer functions, such as `tf.keras.initializers.RandomNormal` or `tf.keras.initializers.GlorotUniform`.

## Training the Model:

Utilize TensorFlow's training API to train your MLP model. Specify the loss function (e.g., mean squared error, categorical cross-entropy), optimizer (e.g., stochastic gradient descent, Adam), learning rate, and the number of epochs. Use the training data prepared in the previous steps to fit the model to the data.

```
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

## Experimental Procedure

### Data Preprocessing

Preprocess your data using TensorFlow's preprocessing functions, such as `tf.data.Dataset`, `tf.data.preprocessing`, or `tf.keras.preprocessing`. Perform any required transformations on the data, including normalization, scaling, or encoding.

## Model Initialization

Use TensorFlow's API to initialize your MLP model. Define the architecture, activation functions, and weight initialization techniques as discussed.

```
model = tf.keras.Sequential()  
  
model.add(tf.keras.layers.Dense(64, activation='relu', input_shape=(input_dim,)))
```

## Training the Model

Train your model using the prepared training dataset. Use TensorFlow's `model.compile` function to specify the loss function, optimizer, and metrics. Then, use `model.fit` to train the model with the training data, specifying the number of epochs and batch size.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
  
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

**Optimizer:** The optimizer determines the algorithm used to update the weights and biases of the neural network during training. In this case, the optimizer chosen is 'adam'. Adam stands for Adaptive Moment Estimation and is a popular optimization algorithm that combines ideas from both AdaGrad and RMSProp. It adapts the learning rate based on the gradients of the parameters, making it efficient and effective for a wide range of deep learning tasks.

Other commonly used optimizers include 'sgd' (Stochastic Gradient Descent), 'rmsprop' (RMSProp optimizer), and 'adagrad' (AdaGrad optimizer). The choice of optimizer depends on the specific problem and the characteristics of the dataset, and it can have an impact on the convergence and performance of the model.

**Loss:** The loss function is used to quantify the difference between the predicted output of the model and the true output. It serves as a measure of how well the model is performing during training. In this case, the chosen loss function is 'binary\_crossentropy'. Binary cross-entropy is commonly used for binary classification problems, where the model outputs a single probability value for a binary class.

If you have a multi-class classification problem, you can choose a different loss function such as 'categorical\_crossentropy'. Other loss functions, such as mean squared error ('mse') or mean absolute error ('mae'), can be used for regression tasks.

## Model Evaluation

After training, evaluate your model's performance using the testing dataset. Use TensorFlow's `model.evaluate` function to calculate relevant metrics such as accuracy, precision, recall, or mean squared error.

*loss, accuracy = model.evaluate(X\_test, y\_test)*

Remember to consult the TensorFlow documentation and explore the available resources for detailed implementation and usage instructions.

## Example

```
import tensorflow as tf
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt

# Generate a synthetic classification dataset
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Perform data preprocessing: feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the MLP model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Visualize the model architecture
plot_model(model, show_shapes=True, show_layer_names=True,
to_file='model_architecture.png')

# Display the model architecture in the notebook
model_img = plt.imread('model_architecture.png')
plt.figure(figsize=(10, 10))
plt.imshow(model_img)
plt.axis('off')
plt.show()
```

```

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

# Train the model
model.fit(X_train_scaled, y_train, epochs=10, batch_size=32)

# Evaluate the model on the testing set
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

```

## Example 2

```

import tensorflow as tf
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the wine dataset
data = load_wine()
X, y = data.data, data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Scale the input features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the MLP model architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(3, activation='softmax') # 3 classes for wine dataset
])

# Compile the model
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

```

```

# Train the model
model.fit(X_train_scaled, y_train, epochs=10, batch_size=32)

# Evaluate the model on the testing set
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

```

## Lab Journal

### Task1:

Implement a multilayer perceptron (MLP) model using TensorFlow and apply it to a binary classification problem. Choose a suitable dataset from a public repository (e.g., UCI Machine Learning Repository) and preprocess the data as necessary. Design the MLP architecture, compile the model with an appropriate loss function and optimizer, train the model on the training set, and evaluate its performance on the testing set. Experiment with different hyperparameters (e.g., number of hidden layers, number of neurons, learning rate) to improve the model's accuracy. Provide a brief report discussing the dataset, model architecture, hyperparameter choices, and the model's performance.

Code:

```

import tensorflow as tf
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

wine_data = load_wine()
X, y = wine_data.data, wine_data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = Sequential([

```

```

    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(3, activation='softmax')
])

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])







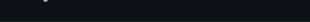
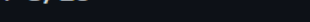
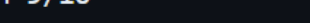
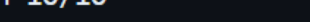

model.fit(X_train_scaled, y_train, epochs=10, batch_size=32)

loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

```

Output:

```

ags.
Epoch 1/10
5/5  0s 3ms/step - accuracy: 0.3975 - loss: 1.1457
Epoch 2/10
5/5  0s 2ms/step - accuracy: 0.4849 - loss: 1.0464
Epoch 3/10
5/5  0s 2ms/step - accuracy: 0.5441 - loss: 0.9819
Epoch 4/10
5/5  0s 3ms/step - accuracy: 0.6097 - loss: 0.8926
Epoch 5/10
5/5  0s 4ms/step - accuracy: 0.6580 - loss: 0.8846
Epoch 6/10
5/5  0s 1ms/step - accuracy: 0.7817 - loss: 0.8361
Epoch 7/10
5/5  0s 4ms/step - accuracy: 0.8081 - loss: 0.8040
Epoch 8/10
5/5  0s 2ms/step - accuracy: 0.8366 - loss: 0.7634
Epoch 9/10
5/5  0s 1ms/step - accuracy: 0.8774 - loss: 0.7416
Epoch 10/10
5/5  0s 2ms/step - accuracy: 0.9072 - loss: 0.7005
2/2  0s 5ms/step - accuracy: 0.9132 - loss: 0.6847
Test Loss: 0.6782
Test Accuracy: 0.9167

```



## Task 2:

Implement a multilayer perceptron (MLP) model using TensorFlow and apply it to a multi-class classification problem. Choose a suitable dataset with more than three classes from a public repository (e.g., UCI Machine Learning Repository) and preprocess the data accordingly. Design the MLP architecture, compile the model with an appropriate loss function and optimizer suitable for multi-class classification, train the model on the training set, and evaluate its performance on the testing set. Experiment with different hyperparameters and network architectures to find the best combination for achieving high accuracy. Provide a detailed analysis of the dataset, model architecture, hyperparameter tuning process, and the model's performance.

Code:

```
import tensorflow as tf
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

wine_data = load_wine()
X = wine_data.data
y = wine_data.target

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)





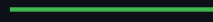
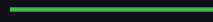

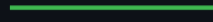
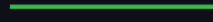
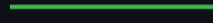

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(wine_data.target_names.shape[0], activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

```
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,
y_test))

loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test loss: {loss:.4f}')
print(f'Test accuracy: {accuracy:.4f}')
```

## Output:

```
Epoch 1/10
5/5  1s 46ms/step - accuracy: 0.5588 - loss: 0.9706 - val_accuracy: 0.8889 - val_loss: 0.7981
Epoch 2/10
5/5  0s 12ms/step - accuracy: 0.8437 - loss: 0.7811 - val_accuracy: 0.9722 - val_loss: 0.6490
Epoch 3/10
5/5  0s 13ms/step - accuracy: 0.9048 - loss: 0.6317 - val_accuracy: 0.9444 - val_loss: 0.5232
Epoch 4/10
5/5  0s 10ms/step - accuracy: 0.9449 - loss: 0.5129 - val_accuracy: 0.9444 - val_loss: 0.4148
Epoch 5/10
5/5  0s 11ms/step - accuracy: 0.9686 - loss: 0.3876 - val_accuracy: 0.9444 - val_loss: 0.3272
Epoch 6/10
5/5  0s 8ms/step - accuracy: 0.9548 - loss: 0.3286 - val_accuracy: 0.9444 - val_loss: 0.2575
Epoch 7/10
5/5  0s 10ms/step - accuracy: 0.9551 - loss: 0.2777 - val_accuracy: 0.9722 - val_loss: 0.2041
Epoch 8/10
5/5  0s 12ms/step - accuracy: 0.9627 - loss: 0.2120 - val_accuracy: 0.9722 - val_loss: 0.1649
Epoch 9/10
5/5  0s 12ms/step - accuracy: 0.9830 - loss: 0.1607 - val_accuracy: 0.9722 - val_loss: 0.1361
Epoch 10/10
5/5  0s 13ms/step - accuracy: 0.9884 - loss: 0.1501 - val_accuracy: 0.9722 - val_loss: 0.1145
2/2  0s 2ms/step - accuracy: 0.9711 - loss: 0.1151
Test loss: 0.1145
Test accuracy: 0.9722
```