

Harsh Arora
AE-1218
22001570006

DEEP LEARNING PRACTICAL ASSIGNMENT

Q1. Implement a deep-neural feed-forward network for estimating the price of house, given real-estate data (Boston Housing Price)

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.datasets import boston_housing
from sklearn.preprocessing import StandardScaler
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
model = Sequential([
    Dense(64, activation='relu', input_shape=(x_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)
])
model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae']
)
history = model.fit(
    x_train, y_train,
```

```

epochs=50,
batch_size=16,
validation_split=0.2,
verbose=1
)

loss, mae = model.evaluate(x_test, y_test, verbose=0)

print("Test MAE (Mean Absolute Error):", mae)

predicted = model.predict(x_test[:5])

print("Predicted Prices:", predicted.flatten())

print("Actual Prices:", y_test[:5])

```

```

Downloaded data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/boston_housing.npz
... 57028/57028 [=====] 100% 0s
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/'input_dim' argument to a layer. When using Sequential models, prefer using an `Input` super().init_(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
21/21    2s 18ms/step - loss: 566.7692 - mae: 22.1489 - val_loss: 590.5991 - val_mae: 22.5375
Epoch 2/50
21/21    0s 8ms/step - loss: 474.1444 - mae: 28.2055 - val_loss: 509.9660 - val_mae: 20.6792
21/21    0s 9ms/step - loss: 422.9601 - mae: 18.5496 - val_loss: 386.0253 - val_mae: 17.6021
Epoch 4/50
21/21    0s 7ms/step - loss: 291.5919 - mae: 14.9336 - val_loss: 231.7836 - val_mae: 12.8186
Epoch 5/50
21/21    0s 7ms/step - loss: 161.9209 - mae: 10.4975 - val_loss: 117.3767 - val_mae: 8.8070
Epoch 6/50
21/21    0s 7ms/step - loss: 73.8933 - mae: 6.5951 - val_loss: 75.5354 - val_mae: 6.2819
21/21    0s 8ms/step - loss: 51.5436 - mae: 5.4783 - val_loss: 55.0989 - val_mae: 5.3246
Epoch 8/50
21/21    0s 9ms/step - loss: 37.2354 - mae: 4.4883 - val_loss: 43.0592 - val_mae: 4.6703
Epoch 9/50
21/21    0s 10ms/step - loss: 30.5888 - mae: 3.9590 - val_loss: 35.8480 - val_mae: 4.2925
21/21    0s 5ms/step - loss: 24.7880 - mae: 3.4284 - val_loss: 31.5105 - val_mae: 4.0856
21/21    0s 5ms/step - loss: 23.5587 - mae: 3.4041 - val_loss: 28.6835 - val_mae: 3.9786
Epoch 12/50
21/21    0s 5ms/step - loss: 22.8079 - mae: 3.3406 - val_loss: 27.1340 - val_mae: 3.8716
Epoch 13/50
21/21    0s 7ms/step - loss: 21.4461 - mae: 3.1401 - val_loss: 25.3352 - val_mae: 3.7955
21/21    0s 5ms/step - loss: 19.8312 - mae: 3.1045 - val_loss: 23.8011 - val_mae: 3.6728
Epoch 15/50
21/21    0s 5ms/step - loss: 21.8813 - mae: 3.1131 - val_loss: 23.0022 - val_mae: 3.6168
Epoch 16/50
21/21    0s 5ms/step - loss: 17.7288 - mae: 2.9338 - val_loss: 22.1607 - val_mae: 3.5486
Epoch 17/50
21/21    0s 7ms/step - loss: 14.2610 - mae: 2.8192 - val_loss: 21.2438 - val_mae: 3.4587
Epoch 18/50
21/21    0s 5ms/step - loss: 15.3126 - mae: 2.7126 - val_loss: 20.4527 - val_mae: 3.4301
Epoch 19/50
21/21    0s 5ms/step - loss: 12.6238 - mae: 2.5895 - val_loss: 19.5799 - val_mae: 3.3361

Epoch 46/50
21/21    0s 5ms/step - loss: 7.3015 - mae: 2.0333 - val_loss: 15.0017 - val_mae: 2.8338
Epoch 47/50
21/21    0s 5ms/step - loss: 11.1160 - mae: 2.3967 - val_loss: 15.1789 - val_mae: 2.8156
Epoch 48/50
21/21    0s 5ms/step - loss: 8.8135 - mae: 2.1348 - val_loss: 15.1890 - val_mae: 2.8027
Epoch 49/50
21/21    0s 5ms/step - loss: 8.6671 - mae: 2.0805 - val_loss: 15.1672 - val_mae: 2.8359
Epoch 50/50
21/21    0s 5ms/step - loss: 7.7427 - mae: 1.8857 - val_loss: 15.0507 - val_mae: 2.8117
Test MAE (Mean Absolute Error): 3.21854305267334
1/1      0s 68ms/step
Predicted Prices: [ 7.4365306 16.83254 20.794758 31.96136 25.366772 ]
Actual Prices: [ 7.2 18.8 19. 27. 22.2]

```

Q2. Implement a feed-forward neural networks for classifying movie reviews as positive or negative (using IMDB dataset)

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Flatten
from tensorflow.keras.preprocessing.sequence import pad_sequences

vocab_size = 5000 # keep top 5000 words
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

max_length = 200
x_train = pad_sequences(x_train, maxlen=max_length)
x_test = pad_sequences(x_test, maxlen=max_length)

model = Sequential([
    Embedding(vocab_size, 32, input_length=max_length), # Word embeddings
    Flatten(),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid') # Binary classification
])

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

history = model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=128,
    validation_split=0.2
)

loss, acc = model.evaluate(x_test, y_test, verbose=0)
print("Test Accuracy:", acc)

predictions = model.predict(x_test[:5])
print("\nPredicted Sentiments:", predictions.flatten())
print("Actual Labels:", y_test[:5])
```

```

Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
    warnings.warn(
157/157      5s 27ms/step - accuracy: 0.6040 - loss: 0.6258 - val_accuracy: 0.8334 - val_loss: 0.3764
Epoch 2/5
157/157      4s 21ms/step - accuracy: 0.9080 - loss: 0.2344 - val_accuracy: 0.8456 - val_loss: 0.3681
Epoch 3/5
157/157      3s 19ms/step - accuracy: 0.9793 - loss: 0.0822 - val_accuracy: 0.8492 - val_loss: 0.4082
Epoch 4/5
157/157      4s 24ms/step - accuracy: 0.9973 - loss: 0.0219 - val_accuracy: 0.8448 - val_loss: 0.4821
Epoch 5/5
157/157      3s 20ms/step - accuracy: 0.9985 - loss: 0.0102 - val_accuracy: 0.8480 - val_loss: 0.4950
Test Accuracy: 0.8475199937820435
1/1          0s 76ms/step

Predicted Sentiments: [0.09684932 0.9999653 0.82193196 0.00419315 0.99968827]
Actual Labels: [0 1 1 0 1]

```

Q3. Implement a deep-neural network for classifying news wires by topic (Reuters dataset).

```

import numpy as np
from tensorflow.keras.datasets import reuters
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

num_words = 10000
(x_train, y_train), (x_test, y_test) = reuters.load_data(num_words=num_words)

def vectorize_sequences(sequences, dimension=num_words):
    results = np.zeros((len(sequences), dimension))
    for i, seq in enumerate(sequences):
        results[i, seq] = 1.0
    return results

x_train = vectorize_sequences(x_train)
x_test = vectorize_sequences(x_test)

y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)

model = Sequential([
    Dense(64, activation='relu', input_shape=(num_words,)),
    Dense(64, activation='relu'),
    Dense(46, activation='softmax')
])

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

history = model.fit(x_train, y_train_cat, epochs=8, batch_size=512, validation_split=0.2)

loss, acc = model.evaluate(x_test, y_test_cat, verbose=0)
print("Test Accuracy:", acc)

```

```

predictions = model.predict(x_test)
predicted_classes = np.argmax(predictions, axis=1) # Convert to class index
actual_classes = y_test

print("\nSample Predictions vs Actual:")
for i in range(10):
    print(f"Sample {i+1}: Predicted = {predicted_classes[i]}, Actual = {actual_classes[i]}")

```

```

...
Epoch 1/8
15/15 3s 80ms/step - accuracy: 0.2554 - loss: 3.6220 - val_accuracy: 0.5142 - val_loss: 2.7730
Epoch 2/8
15/15 1s 57ms/step - accuracy: 0.5384 - loss: 2.5063 - val_accuracy: 0.6116 - val_loss: 1.8510
Epoch 3/8
15/15 1s 70ms/step - accuracy: 0.6534 - loss: 1.6424 - val_accuracy: 0.7051 - val_loss: 1.4229
Epoch 4/8
15/15 1s 56ms/step - accuracy: 0.7454 - loss: 1.1986 - val_accuracy: 0.7401 - val_loss: 1.2291
Epoch 5/8
15/15 1s 69ms/step - accuracy: 0.7829 - loss: 0.9997 - val_accuracy: 0.7496 - val_loss: 1.1371
Epoch 6/8
15/15 1s 72ms/step - accuracy: 0.8274 - loss: 0.7990 - val_accuracy: 0.7707 - val_loss: 1.0577
Epoch 7/8
15/15 2s 114ms/step - accuracy: 0.8631 - loss: 0.6530 - val_accuracy: 0.7813 - val_loss: 1.0120
Epoch 8/8
15/15 1s 67ms/step - accuracy: 0.8930 - loss: 0.5362 - val_accuracy: 0.7896 - val_loss: 0.9686
Test Accuracy: 0.7796081900596619
71/71 0s 3ms/step

Sample Predictions vs Actual:
Sample 1: Predicted = 3, Actual = 3
Sample 2: Predicted = 10, Actual = 10
Sample 3: Predicted = 1, Actual = 1
Sample 4: Predicted = 4, Actual = 4
Sample 5: Predicted = 13, Actual = 4
Sample 6: Predicted = 3, Actual = 3
Sample 7: Predicted = 3, Actual = 3
Sample 8: Predicted = 3, Actual = 3
Sample 9: Predicted = 3, Actual = 3
Sample 10: Predicted = 3, Actual = 3

```

Q4. Implement CNN for classifying MNIST dataset

```

# Import Libraries
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),

```

```

        MaxPooling2D((2, 2)),  

        Flatten(),  

        Dense(128, activation='relu'),  

        Dense(10, activation='softmax') # 10 classes  

    )  
  

    model.compile(  

        optimizer='adam',  

        loss='categorical_crossentropy',  

        metrics=['accuracy']
    )  
  

    history = model.fit(  

        x_train, y_train,  

        epochs=5,  

        batch_size=64,  

        validation_split=0.1
    )  
  

    loss, acc = model.evaluate(x_test, y_test, verbose=0)
    print("Test Accuracy:", acc)

```

```

    /usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_  

        super().__init__(activity_regularizer=activity_regularizer, **kwargs)  

Epoch 1/5  

844/844 [=====] 47s 53ms/step - accuracy: 0.8860 - loss: 0.3859 - val_accuracy: 0.9845 - val_loss: 0.0560  

Epoch 2/5  

844/844 [=====] 80s 50ms/step - accuracy: 0.9835 - loss: 0.0506 - val_accuracy: 0.9862 - val_loss: 0.0432  

Epoch 3/5  

844/844 [=====] 45s 54ms/step - accuracy: 0.9898 - loss: 0.0343 - val_accuracy: 0.9902 - val_loss: 0.0385  

Epoch 4/5  

844/844 [=====] 42s 50ms/step - accuracy: 0.9921 - loss: 0.0236 - val_accuracy: 0.9910 - val_loss: 0.0332  

Epoch 5/5  

844/844 [=====] 43s 51ms/step - accuracy: 0.9946 - loss: 0.0170 - val_accuracy: 0.9877 - val_loss: 0.0374  

Test Accuracy: 0.9900000095367432

```

Q5. Classify MNIST dataset using any pertained model

```

import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Conv2D
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.utils import to_categorical  
  

(x_train, y_train), (x_test, y_test) = mnist.load_data()  
  

x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0  
  

x_train = tf.expand_dims(x_train, axis=-1)

```

```
x_test = tf.expand_dims(x_test, axis=-1)

x_train = tf.image.grayscale_to_rgb(x_train)
x_test = tf.image.grayscale_to_rgb(x_test)

x_train = tf.image.resize(x_train, (224, 224))
x_test = tf.image.resize(x_test, (224, 224))

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

base_model = MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)
base_model.trainable = False

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(128, activation="relu"),
    Dense(10, activation="softmax") # 10 MNIST digits
])

model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

history = model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=32,
    validation_split=0.1
)
predictions = model.predict(x_test[:10])
predicted_classes = tf.argmax(predictions, axis=1)
actual_classes = tf.argmax(y_test[:10], axis=1)

loss, acc = model.evaluate(x_test, y_test, verbose=0)
print("Test Accuracy:", acc)
```

```
Epoch 1/5
844/844 10s 6ms/step - accuracy: 0.8809 - loss: 0.3967 - val_accuracy: 0.9873 - val_loss: 0.0450
Epoch 2/5
844/844 3s 3ms/step - accuracy: 0.9841 - loss: 0.0506 - val_accuracy: 0.9883 - val_loss: 0.0431
Epoch 3/5
844/844 3s 4ms/step - accuracy: 0.9903 - loss: 0.0324 - val_accuracy: 0.9897 - val_loss: 0.0359
Epoch 4/5
844/844 3s 4ms/step - accuracy: 0.9926 - loss: 0.0239 - val_accuracy: 0.9913 - val_loss: 0.0340
Epoch 5/5
844/844 4s 5ms/step - accuracy: 0.9949 - loss: 0.0159 - val_accuracy: 0.9907 - val_loss: 0.0377
Test Accuracy: 0.9909999966621399
1/1 0s 404ms/step

Predicted vs Actual:
Sample 1: Predicted = 7, Actual = 7
Sample 2: Predicted = 2, Actual = 2
Sample 3: Predicted = 1, Actual = 1
Sample 4: Predicted = 0, Actual = 0
Sample 5: Predicted = 4, Actual = 4
Sample 6: Predicted = 1, Actual = 1
Sample 7: Predicted = 4, Actual = 4
Sample 8: Predicted = 9, Actual = 9
Sample 9: Predicted = 5, Actual = 5
Sample 10: Predicted = 9, Actual = 9
```