Solution ① → [220621 Manish Kumar] [Assignment - 5]

Logistic Regression VS classic perceptron ?

→ So generally logistics regression preferable to use a logistic regression classifier rather than a classic perceptron because -

- Logistic regression outputs probabilities (values between 0 and 1), making it suitable for probabilistic interpretation and gradient based optimization.
- The classic perceptron uses hard threshold (step function), which is not differentiable and does not provide probability estimates.
- Logistic regression uses a smooth, differentiable sigmoid activation, which enables effective training using gradient descent.

→ Tweaking perceptron to match Logistic Regression

- Replace the perceptron's step activation function with sigmoid (logistics) activation function
- Train the model using a loss function such as cross-entropy, instead of the perceptron's error-counting loss.

Solution - (2) → Output layer design

- Spam Vs. Ham.
  - Neurons : → 1
  - Activation → sigmoid
- MNIST (10 digits)
  - Neurons → 10
  - Activation → Softmax.

Solution: (3) → ∴ Given

(a) Input layer : 10 neurons (pass through)
Hidden layer : 50 neurons (ReLU)
ouput layer : 3 neurons (ReLU)
ReLU activation function in hidden layer

(a) Shape of input matrix → (batch size, 10)

(b) Shape of weight matrix → $(w_h) = (10, 50)$
Shape of bias vector $(b_h) = 50$

(c) Shape of output layer weight matrix (w_o)
$= (50, 3)$

(d) output shape = (batch size, 3)

(e) Equation ⇒ $\boxed{y = \text{ReLU}(X \cdot w_h + b_h) \cdot w_o + b_o}$

Solution (4) →

$$\therefore E(w) = -\sum_{n=1}^{N} \{ t_n \ln y_n + (1-t_n) \ln(1-y_n) \}$$

$\therefore a_n$ = logistic sigmoid activation

$$y_n = \sigma(a_n)$$

$$y_n' = \sigma(a_n)(1-\sigma(a_n)) \quad \left[\sigma(a_n) = \frac{1}{1+e^{-a_n}} \right.$$

$$E = -\sum_{n=1}^{N} [t_n \ln \sigma(a_n) + (1-t_n) \ln(1-\sigma(a_n))]$$

$$E' = -\sum_{n=1}^{N} \left[t_n \frac{y_n'}{\sigma(a_n)} - (1-t_n) \cdot \frac{y_n'}{1-\sigma(a_n)} \right]$$

$$= -\sum_{n=1}^{N} [t_n (1-\sigma(a_n)) - (1-t_n)\sigma(a_n)]$$

$$= -\sum_{n=1}^{N} [t_n - t_n \sigma(a_n) - \sigma(a_n) + t_n \sigma(a_n)]$$

$$= -\sum_{n=1}^{N} [t_n - \sigma(a_n)]$$

$$\boxed{\frac{\partial E}{\partial a_k} = y_k - t_k}$$

Solution (5) →

2	5	-3	0
0	6	0	-4
-1	-3	0	2
5	0	0	3

A

$\times$

-2	0
4	6

B

$= ?$

Let

$$A \times B = C = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$a_{11} = -2 \times 2 + 5 \times 0 + 0 \times 4 + 6 \times 6 = 32$

Similarly

$a_{12} = 14$, $a_{13} = -18$

$a_{21} = -22$, $a_{22} = -24$, $a_{23} = 12$

$a_{31} = 22$; $a_{32} = 6$, $a_{33} = 18$

So

$$C = \begin{bmatrix} 32 & 14 & -18 \\ -22 & -24 & 12 \\ 22 & 6 & 18 \end{bmatrix}$$

Solution (6) →

1D Strided convolutional layer with input having four units with activations (x_1, x_2, x_3, x_4) which 3 padded with zero to given

$(0, x_1, x_2, x_3, x_4, 0)$, & a filter with parameters (w_1, w_2, w_3), stride = 2

1D activation vector.

$$\text{Output} = \begin{pmatrix} w_2 x_1 + w_3 x_1 \\ w_1 x_2 + w_2 x_3 + w_3 x_4 \end{pmatrix}$$

$$= A \times \begin{pmatrix} 0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ 0 \end{pmatrix}$$

$$A = \begin{bmatrix} 0 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \end{bmatrix}$$

upsampling Convolutional

(z_1, z_2) – input layer activation

output stride = 2

output size = 6 = (input size − 1) × stride + filter size

= (2 − 1) × 2 + 3 = 5

∴ zero padding gives us 6 dimensions

Inserting zero blw Input elmt ($z_1, 0, z_2$

z_1 gets spread out using the filter as positio

0 1 ϕ 2 : $w_1 \cdot 2, , w_2 z_1, , w_3 z_1$

z_2 gets placed 2 steps (becomes of stride = 2)

at position 2, 3, 4

$w_1 z_2, w_2 z_2, w_3 z_3$

If overlapping happens, values are added

6th dimension

output vector $\Rightarrow y = \begin{bmatrix} w_1 z_1 \\ w_2 z_2 \\ w_3 z_1 + w_1 z_2 \\ w_2 z_2 \\ w_3 z_2 \end{bmatrix}$

matrix from using transpose A

$$A = \begin{bmatrix} 0 & w_1 & w_2 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 0 & 0 & 0 \\ w_1 & 0 & 0 \\ w_2 & w_1 & 0 \\ 0 & w_2 & 0 \\ 0 & w_3 & w_1 \\ 0 & 0 & w_2 \end{bmatrix} \Rightarrow y = A^T \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

Solution (7) →

$$\text{Input} = \begin{bmatrix} 1 & 1 & 2 & 1 \\ 2 & 1 & 2 & 0 \\ 3 & 2 & 4 & 1 \\ 1 & 1 & 2 & 1 \end{bmatrix}$$

Encoder: $W = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ -1 & 0 & 1 & 0 \end{bmatrix}$ $b = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$

Encoder output:

$$= \text{ReLU}(W \cdot \text{Input} + b)$$

$$W \cdot \text{Input} + b = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 2 & 1 \\ 2 & 1 & 2 & 0 \\ 3 & 2 & 4 & 1 \\ 1 & 1 & 2 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 \end{bmatrix}$$

$$= \text{ReLU}\left(\begin{bmatrix} 2 & 2 & 4 & 2 \\ 5 & 3 & 6 & 1 \\ 1 & 0 & 1 & -1 \end{bmatrix}\right) = \begin{bmatrix} 2 & 2 & 4 & 2 \\ 5 & 3 & 6 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Battle Recti $= \text{ReLU}\left(\begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & 2 & 4 & 2 \\ 5 & 3 & 6 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} + b\right)$ Encoder output

$b \downarrow [0]$

Zero Matrix given

$$= \text{ReLU}\left(\begin{bmatrix} 3 & 2 & 5 & 2 \\ 3 & 1 & 2 & -1 \end{bmatrix}\right) = \begin{bmatrix} 3 & 2 & 5 & 2 \\ 3 & 1 & 2 & 0 \end{bmatrix}$$

Decoded output

$$= \text{ReLU}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 3 & 2 & 5 & 2 \\ 3 & 1 & 2 & 0 \end{bmatrix}\right.$$

Calculated bottleneck

$$\left. + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}\right)$$

bias

$$= \begin{bmatrix} 3 & 2 & 5 & 2 \\ 4 & 2 & 3 & 1 \\ 0 & 1 & 3 & 2 \end{bmatrix}$$ (decoded output)

Reconstructed output $\Rightarrow$ $\text{ReLU}\left(\begin{bmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}\begin{bmatrix} 3 & 2 & 5 & 2 \\ 4 & 2 & 3 & 1 \\ 0 & 1 & 3 & 2 \end{bmatrix}\right.$

$$\left. + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ -3 & -3 & -3 & -3 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 3 & 1 & 2 & 0 \\ 0 & 0 & 2 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 0 & 3 & 0 \end{bmatrix}$$

Reconstruction loss $= \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{i,j} - \hat{x}_{i,j})^2$

$= \frac{1}{16} (4+0+0+1+4+1+0+1+4+0+4+0+1+1+1)$

$= 1.375$

MSE loss gradient

$$\frac{\partial L}{\partial y} = \frac{2}{n} (y_{wrong} - y_{true})$$

$$= \sum_i \sum_j y_{i,j} - \sum_{i \text{ for } j} y_{i,j \text{ true}}$$

$$= \frac{2}{16} \left| (23 - 25) \right| = \frac{1}{8} \times 2 = 0.25$$

Programming Questions:

Solution 1:

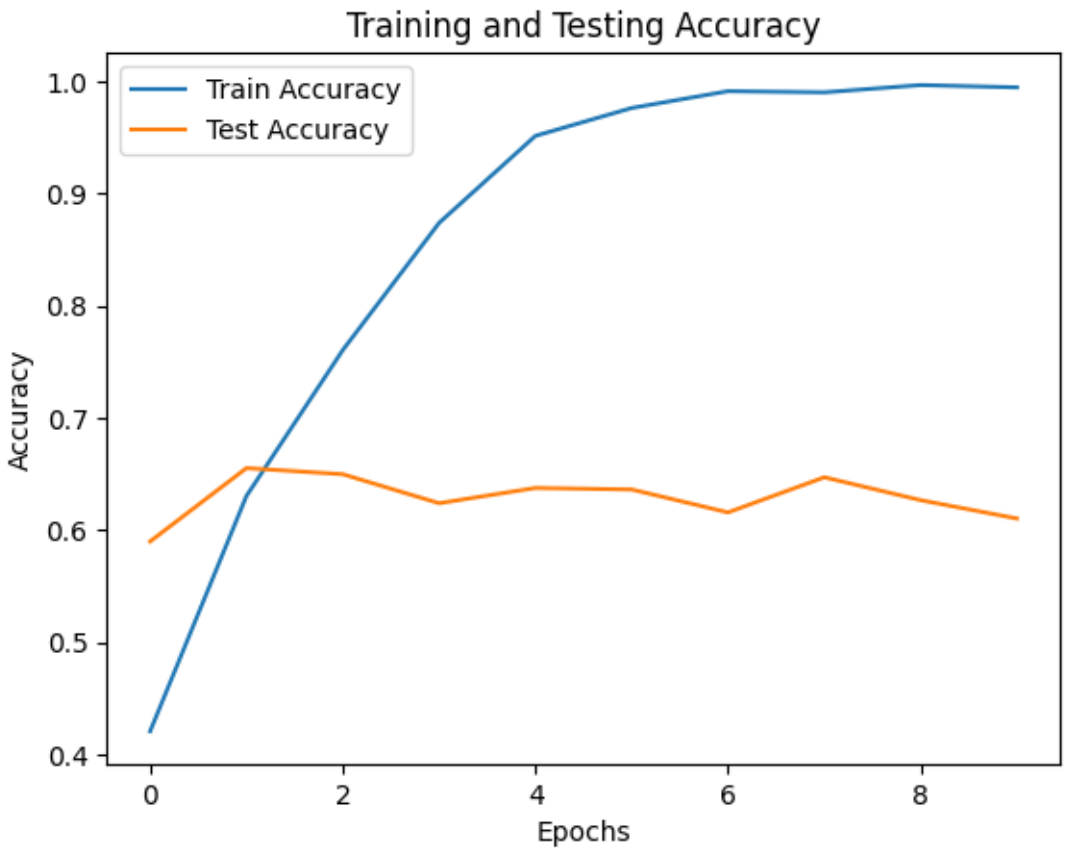
July 11, 2025

[5]:
```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np


dataset, info = tfds.load('tf_flowers', with_info=True, as_supervised=True)
num_classes = info.features['label'].num_classes


train_size = int(0.8 * info.splits['train'].num_examples)
test_size = info.splits['train'].num_examples - train_size

train_ds = dataset['train'].take(train_size)
test_ds = dataset['train'].skip(train_size)


def preprocess(image, label):
    image = tf.image.resize(image, [128, 128])
    image = tf.cast(image, tf.float32) / 255.0
    return image, label


BATCH_SIZE = 32
train_ds = train_ds.map(preprocess).shuffle(1000).batch(BATCH_SIZE).prefetch(tf.
 ↪data.AUTOTUNE)
test_ds = test_ds.map(preprocess).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)


model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128,␣
 ↪128, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
```

```
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

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


history = model.fit(train_ds, validation_data=test_ds, epochs=10)


plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.title("Training and Testing Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()


y_true = []
y_pred = []

for images, labels in test_ds:
    preds = model.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(np.argmax(preds, axis=1))


print("\nClassification Report:")
print(classification_report(y_true, y_pred))


print("Confusion Matrix:")
print(confusion_matrix(y_true, y_pred))
```

```
/usr/local/lib/python3.11/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
92/92             112s 1s/step -
accuracy: 0.3366 - loss: 2.0018 - val_accuracy: 0.5899 - val_loss: 1.0314
Epoch 2/10
```

```
92/92            93s 1s/step -
accuracy: 0.6191 - loss: 0.9779 - val_accuracy: 0.6553 - val_loss: 0.9203
Epoch 3/10
92/92            95s 1s/step -
accuracy: 0.7318 - loss: 0.7372 - val_accuracy: 0.6499 - val_loss: 0.9015
Epoch 4/10
92/92            136s 964ms/step -
accuracy: 0.8716 - loss: 0.4013 - val_accuracy: 0.6240 - val_loss: 1.1545
Epoch 5/10
92/92            94s 1s/step -
accuracy: 0.9437 - loss: 0.2114 - val_accuracy: 0.6376 - val_loss: 1.4820
Epoch 6/10
92/92            96s 1s/step -
accuracy: 0.9706 - loss: 0.1066 - val_accuracy: 0.6362 - val_loss: 1.3767
Epoch 7/10
92/92            136s 975ms/step -
accuracy: 0.9907 - loss: 0.0468 - val_accuracy: 0.6158 - val_loss: 1.7748
Epoch 8/10
92/92            94s 1s/step -
accuracy: 0.9909 - loss: 0.0500 - val_accuracy: 0.6471 - val_loss: 1.6884
Epoch 9/10
92/92            144s 1s/step -
accuracy: 0.9955 - loss: 0.0210 - val_accuracy: 0.6267 - val_loss: 2.2096
Epoch 10/10
92/92            134s 961ms/step -
accuracy: 0.9967 - loss: 0.0160 - val_accuracy: 0.6104 - val_loss: 1.9694
```




```
1/1             0s 405ms/step
1/1             0s 252ms/step
1/1             0s 242ms/step
1/1             0s 247ms/step
1/1             0s 265ms/step
1/1             0s 244ms/step
1/1             0s 247ms/step
1/1             0s 263ms/step
1/1             0s 245ms/step
1/1             0s 246ms/step
1/1             0s 302ms/step
1/1             0s 258ms/step
1/1             0s 238ms/step
1/1             0s 293ms/step
1/1             0s 270ms/step
1/1             0s 287ms/step
1/1             0s 251ms/step
1/1             0s 248ms/step
1/1             0s 268ms/step
1/1             0s 261ms/step
```

```
1/1             0s 244ms/step
1/1             0s 263ms/step
1/1             0s 320ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.77      0.66       158
           1       0.75      0.36      0.49       141
           2       0.48      0.66      0.56       146
           3       0.81      0.69      0.74       151
           4       0.61      0.54      0.57       138

    accuracy                           0.61       734
   macro avg       0.64      0.60      0.60       734
weighted avg       0.64      0.61      0.61       734

Confusion Matrix:
[[122   7  12  11   6]
 [ 40  51  30   7  13]
 [ 14   3  97   4  28]
 [ 26   1  19 104   1]
 [ 12   6  43   3  74]]
```

Solution 2:

July 11, 2025

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split


df = pd.read_csv('A5.csv')
print(df.head())


prices = df.iloc[:, 0].values.reshape(-1, 1)


scaler = MinMaxScaler()
scaled_prices = scaler.fit_transform(prices)


def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    return np.array(X), np.array(y)

seq_len = 60
X, y = create_sequences(scaled_prices, seq_len)

split = int(0.8 * len(X))
X_train, y_train = X[:split], y[:split]
X_test, y_test = X[split:], y[split:]


model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(seq_len, 1)),
```

```
    LSTM(50),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')
history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
 ↪validation_data=(X_test, y_test))


predictions = model.predict(X_test)
predicted_prices = scaler.inverse_transform(predictions)
real_prices = scaler.inverse_transform(y_test)

plt.figure(figsize=(10,6))
plt.plot(real_prices, label='Actual Price')
plt.plot(predicted_prices, label='Predicted Price')
plt.title('Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```

```
       Close
0  24.320436
1  23.635284
2  23.637505
3  23.968964
4  24.889898
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

```
Epoch 1/20
49/49             10s 87ms/step -
loss: 0.0171 - val_loss: 0.0022
Epoch 2/20
49/49             5s 80ms/step -
loss: 4.0919e-04 - val_loss: 0.0021
Epoch 3/20
49/49             5s 75ms/step -
loss: 3.5809e-04 - val_loss: 0.0018
Epoch 4/20
49/49             4s 72ms/step -
loss: 3.4392e-04 - val_loss: 0.0018
```

```
Epoch 5/20
49/49             4s 81ms/step -
loss: 3.4887e-04 - val_loss: 0.0021
Epoch 6/20
49/49             5s 91ms/step -
loss: 3.3972e-04 - val_loss: 0.0028
Epoch 7/20
49/49             3s 71ms/step -
loss: 3.5537e-04 - val_loss: 0.0030
Epoch 8/20
49/49             6s 84ms/step -
loss: 3.5901e-04 - val_loss: 0.0019
Epoch 9/20
49/49             5s 75ms/step -
loss: 2.8566e-04 - val_loss: 0.0024
Epoch 10/20
49/49             4s 74ms/step -
loss: 2.7825e-04 - val_loss: 0.0014
Epoch 11/20
49/49             4s 81ms/step -
loss: 2.5916e-04 - val_loss: 0.0020
Epoch 12/20
49/49             5s 75ms/step -
loss: 2.8406e-04 - val_loss: 0.0013
Epoch 13/20
49/49             4s 73ms/step -
loss: 2.5945e-04 - val_loss: 0.0016
Epoch 14/20
49/49             4s 83ms/step -
loss: 2.8621e-04 - val_loss: 0.0014
Epoch 15/20
49/49             4s 89ms/step -
loss: 3.1127e-04 - val_loss: 0.0026
Epoch 16/20
49/49             4s 73ms/step -
loss: 2.6444e-04 - val_loss: 0.0015
Epoch 17/20
49/49             6s 93ms/step -
loss: 2.6091e-04 - val_loss: 0.0017
Epoch 18/20
49/49             4s 83ms/step -
loss: 2.4419e-04 - val_loss: 0.0010
Epoch 19/20
49/49             4s 73ms/step -
loss: 2.9064e-04 - val_loss: 0.0012
Epoch 20/20
49/49             4s 75ms/step -
loss: 2.0154e-04 - val_loss: 0.0015
```

```
13/13                2s 78ms/step
```