1

Q1.Build a Single Layer Neural Network Model and find net input of different input and weight vectors.

Sol:

```
import numpy as np
x = []
w=[]
# number of elements as input
n = int(input("Enter number of elements : "))
# iterating till the range
print("enter input vector")
for i in range(0, n):
ele = float(input())
# adding the element
x.append(ele)
print("enter weight vector")
for i in range(0, n):
ele = float(input())
w.append(ele)
print("input vector=",x)
print("weight vector=",w)
bias= float(input("Enter Bias value: "))
X=np.array(x)
W=np.array(w)
y=X.dot(W)
print("Net Input=",y)
```

2.Build a Single Layer Neural Network Model and find Net input of Network using different input and weight variables including bias value.

Sol:

```
import numpy as np
x = []
w=[]
# number of elements as input
n = int(input("Enter number of elements : "))
# iterating till the range
print("enter input vector")
for i in range(0, n):
ele = float(input())
# adding the element
x.append(ele)
print("enter weight vector")
for i in range(0, n):
ele = float(input())
w.append(ele)
print("input vector=",x)
print("weight vector=",w)
bias= float(input("Enter Bias value: "))
X=np.array(x)
W=np.array(w)
y=X.dot(W)+bias
print("Net INput=",y)
```

3.Build a Single Layer Neural Network Model and find output of different input and weight variables including bias value using Activation function.

Sol:

```
import numpy as np
x = []
w=[]
# number of elements as input
n = int(input("Enter number of elements : "))
# iterating till the range
print("enter input vector")
for i in range(0, n):
ele = float(input())
# adding the element
x.append(ele)
print("enter weight vector")
for i in range(0, n):
ele = float(input())
w.append(ele)
print("input vector=",x)
print("weight vector=",w)
bias= float(input("Enter Bias value: "))
X=np.array(x)
W=np.array(w)
y=X.dot(W)+bias
print("Output=",y)
binary=1/(1+np.exp(-y))#binary sigmoid
print("binary sigmoid",binary)
bipolar = (np.exp(y)-1)/(np.exp(y)+1) #bipolar sigmoid
print("bipolar sigmoid",bipolar)
```

Q4.Build a single layer Perceptron,Train the network on a simple dataset, such as the XOR problem, and analyse the model's performance.

Sol:

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Define the dataset for the XOR problem
x_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_train = np.array([[0], [1], [1], [0]])
# Define the Single Layer Perceptron model
model = Sequential([
 Dense(units=1, input_shape=(2,), activation='sigmoid')])
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# Train the model
model.fit(x_train, y_train, epochs=1000, verbose=0)
# Evaluate the model
loss, accuracy = model.evaluate(x_train, y_train)
print("Loss:", loss)
print("Accuracy:", accuracy)
X=([[0, 1]])
predictions = model.predict(X)
print("Predictions:")
print(predictions)
```

Q5.Build a Multi Layer Perceptron, Train the network on a simple dataset, such as the XOR problem, and analyse the model's performance.

Sol:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Define the dataset (XOR problem)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
# Define the Multi Layer Perceptron model
model = Sequential([
 Dense(2, input_shape=(2,), activation='relu'),
 Dense(1, activation='sigmoid')
])
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# Train the model
model.fit(X, y, epochs=1000, verbose=0)
# Evaluate the model
loss, accuracy = model.evaluate(X, y)
print("Model performance:")
print("Loss:", loss)
print("Accuracy:", accuracy)
X=([[0, 1]])
predictions = model.predict(X)
print("Predictions:")
print(predictions)
```

2

Q1. Build a simple CNN model using MNIST dataset and display model summary and test model Accuracy.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import numpy as np
import cv2
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
# Reshape data to add a channel dimension (required by Conv2D layers)
x_train = x_train.reshape((-1, 28, 28, 1))
x_test = x_test.reshape((-1, 28, 28, 1))
# Build the CNN model
model = models.Sequential([
 layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.Flatten(),
 layers.Dense(64, activation='relu'),
 layers.Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
# Display model summary
model.summary()
# Train the model
model.fit(x_train, y_train, epochs=2, batch_size=64, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Q2. Build a CNN model for handwritten digit recognition using MNIST dataset and predict the give digit based on image.

Sol:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import numpy as np
import cv2
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
# Reshape data to add a channel dimension (required by Conv2D layers)
x_train = x_train.reshape((-1, 28, 28, 1))
x_test = x_test.reshape((-1, 28, 28, 1))
# Build the CNN model
model = models.Sequential([
 layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.Flatten(),
 layers.Dense(64, activation='relu'),
 layers.Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
# Display model summary
model.summary()
# Train the model
model.fit(x_train, y_train, epochs=2, batch_size=64, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
# Make predictions
img = cv2.imread("3.png")
img = cv2.resize(img, (28, 28)) # Resize image to fit model input shape
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to grayscale
img = img / 255.0 # Normalize pixel values
digit_image = np.expand_dims(img, axis=0) # Add batch dimension
digit_image = digit_image.reshape((1, 28, 28, 1))
# Make predictions
predictions = model.predict(digit_image)
predicted_digit = np.argmax(predictions)
print('Predicted digit:', predicted_digit)
# Display the image
cv2.imshow("Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q3. Build a CNN model for fashion Dress recognition using Fashion MNIST dataset and predict the class based on given image.
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist
import numpy as np
import cv2
# Load Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
# Reshape data to add a channel dimension (required by Conv2D layers)
x_train = x_train.reshape((-1, 28, 28, 1))
x_test = x_test.reshape((-1, 28, 28, 1))
# Build the CNN model
model = models.Sequential([
 layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.Flatten(),
 layers.Dense(64, activation='relu'),
 layers.Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
# Display model summary
model.summary()
# Train the model
model.fit(x_train, y_train, epochs=20, batch_size=64, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
# Load and preprocess the image
img = cv2.imread("boot.png")
img = cv2.resize(img, (28, 28)) # Resize image to fit model input shape
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to grayscale
img = img / 255.0 # Normalize pixel values
digit_image = np.expand_dims(img, axis=0) # Add batch dimension
digit_image = digit_image.reshape((1, 28, 28, 1))
# Make predictions
predictions = model.predict(digit_image)
predicted_class = np.argmax(predictions)
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle
boot']
print("Predicted class:", class_names[predicted_class])
cv2.imshow("Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q4. . Build a simple CNN model using CIFAR -10 dataset and display model summary and test model Accuracy.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
import numpy as np
import cv2
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# Build the CNN model
model = models.Sequential([
 layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.Flatten(),
 layers.Dense(64, activation='relu'),
 layers.Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
# Display model summary
model.summary()
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Q5.Implement a deep learning model CNN to classify images from the CIFAR-10 dataset and predict the class based on given image.
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
Sol:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
import numpy as np
import cv2
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# Build the CNN model
model = models.Sequential([
 layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.MaxPooling2D((2, 2)),
 layers.Conv2D(64, (3, 3), activation='relu'),
 layers.Flatten(),
 layers.Dense(64, activation='relu'),
 layers.Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
# Display model summary
model.summary()
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
from tensorflow.keras.preprocessing import image
img_path = "Desktop\DL Data\cat1.png"
# Load and preprocess the image
img = image.load_img(img_path, target_size=(32, 32))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = img_array / 255.0 # Normalize pixel values
# Make predictions
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions)
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
print("Predicted class:", class_names[predicted_class])
```

3

Q1. Create a simple chatbot using Python and natural language processing libraries to engage in conversation with users.
(pip install nltk)
Sol:

```
import nltk
from nltk.chat.util import Chat, reflections
# Define pairs of patterns and responses
pairs = [
 ["hi|hello|hey", ["Hello!", "Hey there!", "Hi!"]],
 ["how are you?", ["I'm good, thanks!", "Doing well, thank you!", "I'm fine, how
about you?"]],
 ["what's your name?", ["I'm a chatbot.", "You can call me Chatbot.", "I'm
Chatbot!"]],
 ["quit|exit|bye", ["Goodbye!", "Bye!", "See you later!"]],
]
# Create a chatbot
chatbot = Chat(pairs, reflections)
# Start conversation
print("Welcome to the chatbot!")
print("Type 'quit' to exit.")
while True:
 user_input = input("You: ")
 response = chatbot.respond(user_input)
 print("Chatbot:", response)
 if user_input.lower() in ["quit", "exit", "bye"]:
 break
```

Q2. Build a simple LSTM model using TensorFlow and the IMDB Movie Reviews dataset.
Sol:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Load and preprocess the IMDB dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
X_train = pad_sequences(X_train, maxlen=200)
X_test = pad_sequences(X_test, maxlen=200)
# Define and compile the LSTM model
model = tf.keras.Sequential([
 tf.keras.layers.Embedding(input_dim=10000, output_dim=128,
input_length=200),
 tf.keras.layers.LSTM(64, return_sequences=True),
 tf.keras.layers.LSTM(32),
 tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# Train and evaluate the model
model.fit(X_train, y_train, batch_size=128, epochs=5, validation_split=0.2)
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

Q3. Implementation of sentiment analysis using LSTM model and IMDB Movie Reviews dataset. Predict the sentiment whether sentiment is positive or negative based on given text review.

Sol:

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import
pad_sequences
# Load and preprocess the IMDB dataset
(X_train, y_train), (X_test, y_test) =
imdb.load_data(num_words=10000)
X_train = pad_sequences(X_train, maxlen=200)
X_test = pad_sequences(X_test, maxlen=200)
# Define and compile the LSTM model
model = tf.keras.Sequential([
 tf.keras.layers.Embedding(input_dim=10000, output_dim=128,
input_length=200),
 tf.keras.layers.LSTM(64, return_sequences=True),
 tf.keras.layers.LSTM(32),
 tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# Train and evaluate the model
model.fit(X_train, y_train, batch_size=128, epochs=10,
validation_split=0.2)
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
# Make predictions
text="I hate this movie."
# Convert text to sequence
tokenizer = imdb.get_word_index()
sequence = [tokenizer[word] if word in tokenizer and
tokenizer[word] < 10000 else 0 for word in text.split()]
# Padding sequence to match input_length
sequence = pad_sequences([sequence], maxlen=200)
# Predict
prediction = model.predict(sequence)
if prediction >= 0.5:
 print('Sentiment is positive')
else:
 print('Sentiment is negative')
```

4

Q1. Build a simple Autoencoder model for Image Compression using given image and display model summary and test model Accuracy.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
import cv2
# Load and preprocess the image
image = cv2.imread("dog.jpg")
#image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convert to RGB format
image = cv2.resize(image, (256, 256)) # Resize to target size
image = image.astype(np.float32) / 255.0 # Normalize pixel values
# Define the autoencoder architecture
def autoencoder_model():
 # Encoder
 encoder_input = layers.Input(shape=(256, 256, 3))
 x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoder_input)
 x = layers.MaxPooling2D((2, 2), padding='same')(x)
 x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
 x = layers.MaxPooling2D((2, 2), padding='same')(x)
 # Decoder
 x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
 x = layers.UpSampling2D((2, 2))(x)
 x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
 x = layers.UpSampling2D((2, 2))(x)
 decoder_output = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)
 # Define model
 autoencoder = models.Model(encoder_input, decoder_output)
 return autoencoder
# Create the autoencoder model
model = autoencoder_model()
model.compile(optimizer='adam', loss='mse')
# Reshape the image for training
image = np.expand_dims(image, axis=0) # Add batch dimension
# Train the autoencoder
model.fit(image, image, epochs=50, batch_size=1, verbose=1)
# Encode and decode the image
encoded_image = model.predict(image)
decoded_image = encoded_image[0]
# Display the decoded image
cv2.imshow("Decoded Image", decoded_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
# Save the decoded image
# Convert the decoded image back to BGR
#decoded_image_bgr = cv2.cvtColor(decoded_image, cv2.COLOR_RGB2BGR)
#decoded_image = tf.keras.preprocessing.image.array_to_img(decoded_image)
#decoded_image.save("decoded_image.jpg")
```

Q2. Build a simple Autoencoder model for Image Denoising using given image and display model summary and test model Accuracy.
Sol:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
import cv2
# Load and preprocess the image
image = cv2.imread("dog.jpg")
image = cv2.resize(image, (256, 256)) # Resize to target size
image = image.astype(np.float32) / 255.0 # Normalize pixel values
# Adding Gaussian noise to the image
noise_factor = 0.5
noisy_image = image + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=image.shape)
noisy_image = np.clip(noisy_image, 0., 1.)
# Define the denoising autoencoder architecture
def denoising_autoencoder_model():
 # Encoder
 encoder_input = layers.Input(shape=(256, 256, 3))
 x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoder_input)
 x = layers.MaxPooling2D((2, 2), padding='same')(x)
 x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
 x = layers.MaxPooling2D((2, 2), padding='same')(x)
 # Decoder
 x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
 x = layers.UpSampling2D((2, 2))(x)
 x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
 x = layers.UpSampling2D((2, 2))(x)
 decoder_output = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)
 # Define model
 autoencoder = models.Model(encoder_input, decoder_output)
 return autoencoder
# Create the denoising autoencoder model
model = denoising_autoencoder_model()
model.compile(optimizer='adam', loss='mse')
# Train the denoising autoencoder
model.fit(noisy_image[np.newaxis, ...], image[np.newaxis, ...],
 epochs=50, batch_size=1, verbose=1)
# Display the denoised image
cv2.imshow("Noisy Image", noisy_image)
# Denoise the noisy image
denoised_image = model.predict(noisy_image[np.newaxis, ...])[0]
# Display the denoised image
cv2.imshow("Denoised Image", denoised_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```