Bhargavi Dandu

700741185

NEURAL NETWORK AND DEEP LEARNING ASSIGNMENT-6

GITHUB LINK: https://github.com/bhargavidandu/DL-NN-Assign6.git

RECORDINGLINK:
https://drive.google.com/file/d/1ABVELITTh3yOIkpAm0AW14hHtzJWvkVI/view?usp=drive_link

Use Case Description: Predicting the diabetes disease

Programming elements: Keras Basics

In class programming:

1. Use the use case in the class:

a. Add more Dense layers to the existing code and check how the accuracy changes.

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required

changes. Report accuracy of the model.

3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below).

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer

```python
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Loading the diabetes dataset
dataset = pd.read_csv('diabetes.csv')

# Splitting the dataset into the dependent and independent variables
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Normalizing the data using StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Building the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

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

# Training the model
history = model.fit(X_train, y_train, epochs = 100, batch_size = 10, validation_split=0.2)

# Evaluating the model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

```python
# Importing the libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Loading the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Splitting the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Normalizing the data using StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Building the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

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

# Training the model
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.2)

# Evaluating the model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

In class programming: Use Image Classification on the hand written digits data set (mnist)

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.
2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.
3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.
4. Run the same code without scaling the images and check the performance?

```python
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Flatten the images
image_size = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape((x_train.shape[0], image_size))
x_test = x_test.reshape((x_test.shape[0], image_size))

# Scale the pixel values from range [0, 255] to [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# Convert class labels to one-hot encoded vectors
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Define the model
model = Sequential()
model.add(Dense(128, activation='relu', input_dim=image_size))
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

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

# Train the model and save the history object
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(x_test, y_test))

# Plot the loss and accuracy for both training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
import keras
```

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop

# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# reshape the data to a 1D array of pixels
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# convert data type to float32 and normalize the data to a range between 0 and 1
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# convert labels to categorical one-hot encoding
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# define the model architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# print the model summary
model.summary()
```

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

# train the model
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_test, y_test))

# plot the training and validation loss and accuracy
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# select a random image from the test data
idx = np.random.randint(x_test.shape[0])
image = x_test[idx].reshape(28, 28)

# plot the selected image
plt.figure()
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.title('Selected Image')
```

```
  plt.subplot(1, 2, 2)
  plt.plot(history.history['accuracy'], label='Training Accuracy')
  plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
  plt.title('Training and Validation Accuracy')
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy')
  plt.legend()

  # select a random image from the test data
  idx = np.random.randint(x_test.shape[0])
  image = x_test[idx].reshape(28, 28)

  # plot the selected image
  plt.figure()
  plt.imshow(image, cmap='gray')
  plt.axis('off')
  plt.title('Selected Image')

  # do inferencing to check the model prediction on the selected image
  prediction = model.predict(image.reshape(1, 784))
  prediction = np.argmax(prediction)

  # print the predicted label
  print('Predicted label:', prediction)
```

```python
# Load the MNIST dataset
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Convert the pixel values to floats and normalize them to the range 0-1
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# Convert the target variable to a one-hot encoding
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Create a neural network model with 3 hidden layers and tanh activation
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(256, input_dim=784, activation='tanh'))
model.add(Dense(128, activation='tanh'))
model.add(Dense(64, activation='tanh'))
model.add(Dense(10, activation='softmax'))

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

# Train the model
history = model.fit(x_train.reshape(-1, 784), y_train, epochs=10, validation_data=(x_test.reshape(-1, 784), y_test))

# Plot the loss and accuracy for both training and validation data
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```

```python
# Load the MNIST dataset
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Flatten the input images
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# Convert the target variable to a one-hot encoding
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Create a neural network model with 2 hidden layers and Relu activation
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(256, input_dim=784, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

# Plot the loss and accuracy for both training and validation data
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```