

Task 13: Building and Training Neural Networks for MNIST Classification

In this assignment, you will implement two different neural network models to classify handwritten digits from the MNIST dataset using TensorFlow and Keras. The first model will be a simple neural network, and the second will be a deep neural network. Both models will be evaluated using training and validation accuracy and loss curves.

Dataset: MNIST

MNIST is a dataset of 70,000 28x28 grayscale images of handwritten digits (0-9), split into 60,000 training images and 10,000 test images.

Part 1: Simple Neural Network for MNIST Classification

1. Load and Preprocess the Data:

- Use TensorFlow's `tf.keras.datasets.mnist` to load the MNIST dataset.
- Normalize the pixel values of the images to be between 0 and 1 by dividing by 255.0.

```
X_train_full, X_test = X_train_full / 255.0, X_test / 255.0
```

- Split the training data into training and validation sets (e.g., 80 percent training, 20 percent validation).

2. Define the Model:

- Use the Keras Sequential API to create a simple neural network model with:
 - An input layer that flattens the 28x28 input images to a 784-dimensional vector.
 - Two dense layers with 16 and 8 neurons, respectively, using ReLU activation.
 - An output layer with 10 neurons (one for each digit class) using softmax activation to output class probabilities.

3. Compile the Model:

- Set up the model using the Adam optimizer with a learning rate of 0.001.

- Use categorical cross-entropy as the loss function and track accuracy as the evaluation metric.

4. Train the Model:

- Train the model using the fit() function for 10 epochs with a batch size of 32.
- Provide the validation set as part of the training to monitor validation accuracy and loss during training.

```
# Define the neural network model

model = Sequential([

    Flatten(input_shape=(28, 28)),    # Flatten the 28x28 images into 784-dimensional vectors

    Dense(16, activation='relu'),      # First hidden layer with 128 neurons

    Dense(8, activation='relu'),       # Second hidden layer with 64 neurons

    Dense(10, activation='softmax')    # Output layer with 10 neurons (for each digit class)

])

# Compile the model

model.compile(optimizer=Adam(learning_rate=0.001),

              loss='categorical_crossentropy',

              metrics=['accuracy'])

# Train the model

history = model.fit(X_train, y_train,

                   epochs=10,

                   batch_size=32,

                   validation_data=(X_val, y_val))

# Evaluate the model on test data

test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)

print(f"Test Accuracy: {test_accuracy:.4f}")
```

5. Plot Training and Validation Curves:

- After training, plot the training and validation accuracy and training and validation loss over the epochs.

```
# Plot the training and validation accuracy and loss over epochs

epochs = range(1, len(history.history['accuracy']) + 1)

# Plot accuracy
plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, history.history['accuracy'], label='Training Accuracy')
plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(epochs, history.history['loss'], label='Training Loss')
plt.plot(epochs, history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Part 2: Deep Neural Network for MNIST Classification

1. Load and Preprocess the Data:

- Repeat the data loading and preprocessing steps as in Part 1. You may reuse the code here.

2. Define the Deep Neural Network Model:

- Create a deeper neural network model with the following architecture:
 - Flatten the 28x28 input images to a 784-dimensional vector.
 - Add four dense layers with 512, 256, 128, and 64 neurons, respectively, all using ReLU activation.
 - After each of the first three dense layers, include a Dropout layer with a rate of 0.2 to prevent overfitting.
 - Add an output layer with 10 neurons and softmax activation to output the probabilities for each digit class.

```
# Define the deep neural network model
model = Sequential([
    Flatten(input_shape=(28, 28)),      # Flatten the 28x28 images into 784-dimensional
    vectors
    Dense(512, activation='relu'),       # First hidden layer with 512 neurons
    Dropout(0.2),                       # Dropout layer to reduce overfitting
    Dense(256, activation='relu'),       # Second hidden layer with 256 neurons
    Dropout(0.2),                       # Another Dropout layer
    Dense(128, activation='relu'),       # Third hidden layer with 128 neurons
    Dropout(0.2),                       # Dropout to regularize the network
    Dense(64, activation='relu'),        # Fourth hidden layer with 64 neurons
    Dense(10, activation='softmax')     # Output layer with 10 neurons (one for each
    class)
])
```

3. Compile the Deep Model:

- Compile the model with the Adam optimizer (learning rate 0.001) and categorical cross-entropy as the loss function.
- Track accuracy as the evaluation metric.

4. Train the Deep Model:

- Train the model for 30 epochs with a batch size of 64.
- Again, use the validation data to monitor validation accuracy and loss during training.

5. Plot Training and Validation Curves for the Deep Model:

- Plot the training and validation accuracy and training and validation loss for the deep neural network over the epochs.

Submission Requirements

Submit your Python code implementing both models, with clear comments explaining each step on the LMS under **Assignment 04_Artificial Neural Networks**.