exercise_1

November 5, 2025

1 Exercise-1: Training Deep Neural Network on MNIST

Train a controlled deep neural network on the MNIST dataset. Set random seeds to 42. Load and preprocess MNIST. Build the network using the following configuration: - Flatten input images to 28 × 28 = 784 features - 3 hidden layers, 64 neurons each - ELU activation function - He normal initialization - Output layer: 10 neurons with softmax - Optimizer: Nadam - learning_rate = 0.001, loss=sparse_categorical_crossentropy - EarlyStopping callback: monitor validation loss, patience = 5, restore best weights - epochs = 50, batch size = 32 - Use only the first 1000 training samples and first 200 test samples

1.1 Q1.1 Report the obtained test accuracy.

[ ]:
```python
# Load MNIST dataset
from tensorflow.keras.datasets import mnist

RANDOM_SEED = 42
hidden_layers = 3
hidden_neurons = 64
output_neurons = 10
learning_rate = 0.001

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

# Preprocess data, normalize and reshape
x_train = x_train.reshape(-1, 28 * 28).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28 * 28).astype('float32') / 255.0

# Use only the first 1000 training samples and first 200 test samples
x_train = x_train[:1000]
y_train = y_train[:1000]
x_test = x_test[:200]
y_test = y_test[:200]

import tensorflow as tf
tf.random.set_seed(RANDOM_SEED)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.callbacks import EarlyStopping

# Build the model
model = Sequential()
# Input layer
model.add(Dense(hidden_neurons, activation='elu',␣
 ↪kernel_initializer='he_normal', input_shape=(28 * 28,)))
# Hidden layers, 3 layers with ELU activation and He normal initialization
for _ in range(hidden_layers - 1):
    model.add(Dense(hidden_neurons, activation='elu',␣
 ↪kernel_initializer='he_normal'))
# Output layer with softmax activation
model.add(Dense(output_neurons, activation='softmax'))
model.compile(optimizer=Nadam(learning_rate=learning_rate),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


# Early stopping callback
earl = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
# Train the model
history = model.fit(x_train,
                    y_train,
                    epochs=50,
                    batch_size=32,
                    validation_split=0.2,
                    callbacks=[earl],
                    verbose=1)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Epoch 1/50
25/25                1s 4ms/step -
accuracy: 0.5200 - loss: 1.5309 - val_accuracy: 0.7900 - val_loss: 0.8540
Epoch 2/50
25/25                0s 1ms/step -
accuracy: 0.8425 - loss: 0.5828 - val_accuracy: 0.8600 - val_loss: 0.5678
Epoch 3/50
25/25                0s 1ms/step -
accuracy: 0.8950 - loss: 0.3513 - val_accuracy: 0.8650 - val_loss: 0.5182
Epoch 4/50
25/25                0s 1ms/step -
accuracy: 0.9287 - loss: 0.2478 - val_accuracy: 0.8650 - val_loss: 0.5160
Epoch 5/50
```

```
25/25             0s 1ms/step -
accuracy: 0.9575 - loss: 0.1807 - val_accuracy: 0.8650 - val_loss: 0.5234
Epoch 6/50
25/25             0s 1ms/step -
accuracy: 0.9737 - loss: 0.1324 - val_accuracy: 0.8600 - val_loss: 0.5344
Epoch 7/50
25/25             0s 1ms/step -
accuracy: 0.9837 - loss: 0.0962 - val_accuracy: 0.8500 - val_loss: 0.5520
Epoch 8/50
25/25             0s 1ms/step -
accuracy: 0.9887 - loss: 0.0681 - val_accuracy: 0.8450 - val_loss: 0.5784
Epoch 9/50
25/25             0s 1ms/step -
accuracy: 0.9975 - loss: 0.0476 - val_accuracy: 0.8500 - val_loss: 0.6056
Test Accuracy: 84.50%
```

exercise_2

November 5, 2025

1 Exercise-2: Training Deep Neural Network on CIFAR-10

Train a controlled deep neural network on the CIFAR-10 dataset. Set random seeds to 42. Load and preprocess CIFAR-10. Build the network using the following configuration: - Flatten input images to $32 \times 32 \times 3 = 3072$ features - 4 hidden layers, 256 neurons each - ELU activation function - He normal initialization - Output layer: 10 neurons with softmax - Optimizer: Nadam - learning_rate = 0.001, loss = sparse_categorical_crossentropy - EarlyStopping callback: monitor validation loss, patience = 5, restore best weights - epochs = 50, batch_size = 128 - Use only the first 5000 training samples and first 1000 test samples

1.1 Q2.1 Report the obtained test accuracy.

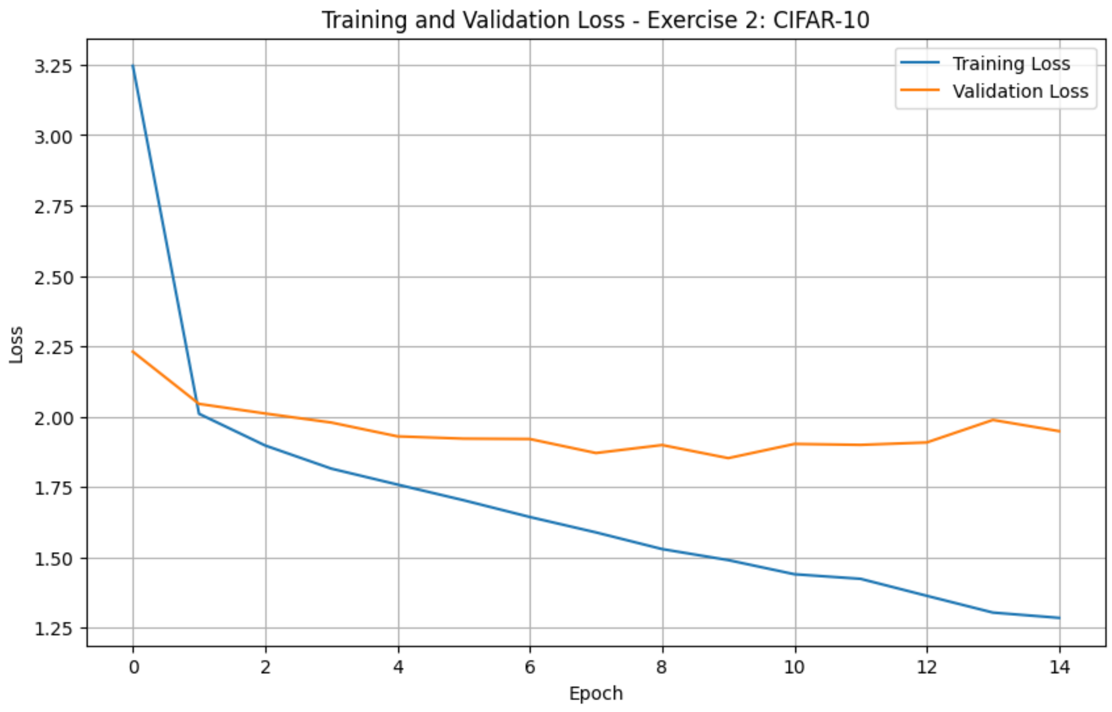
[ ]:
```python
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.layers import Dense
from tensorflow.random import set_seed
import matplotlib.pyplot as plt
import numpy as np

loss_function = 'sparse_categorical_crossentropy'
output_activation = 'softmax'
initializer = 'he_normal'
learning_rate = 0.001
hidden_neurons = 256
output_neurons = 10
optimizer = 'Nadam'
activation = 'elu'
random_seed = 42
batch_size = 128
epochs = 50

# Set random seeds to 42
set_seed(42)
np.random.seed(42)

# Load cifar-10 dataset
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Use only the first 5000 training samples and first 1000 test samples
x_train = x_train[:5000]
y_train = y_train[:5000]
x_test = x_test[:1000]
y_test = y_test[:1000]

# Preprocess the data by flattening and normalizing, 32x32x3 to 3072 features
x_train = x_train.reshape(-1, 32 * 32 * 3).astype('float32') / 255.0
x_test = x_test.reshape(-1, 32 * 32 * 3).astype('float32') / 255.0

# Build the model
model = Sequential()
# Add hidden layers
model.add(Dense(hidden_neurons,
                        activation=activation,
                        kernel_initializer=initializer,
                        input_shape=(32 * 32 * 3,)))
model.add(Dense(hidden_neurons,
                        activation=activation,
                        kernel_initializer=initializer))
model.add(Dense(hidden_neurons,
                        activation=activation,
                        kernel_initializer=initializer))
model.add(Dense(hidden_neurons,
                        activation=activation,
                        kernel_initializer=initializer))

# add output layer
model.add(Dense(output_neurons, activation=output_activation))
# Compile the model, with specified optimizer and loss function
model.compile(optimizer=Nadam(learning_rate=learning_rate),
              loss=loss_function,
              metrics=['accuracy'])

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss',
                     patience=5,
                     restore_best_weights=True)
# Train the model
history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2,
          callbacks=[early_stopping],
          verbose=1)
```

```
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')

# Plot training and validation loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss - Exercise 2: CIFAR-10')
plt.legend()
plt.grid(True)
plt.show()
```

```
Epoch 1/50
32/32                 1s 7ms/step -
accuracy: 0.1497 - loss: 3.2464 - val_accuracy: 0.2000 - val_loss: 2.2310
Epoch 2/50
32/32                 0s 5ms/step -
accuracy: 0.2677 - loss: 2.0104 - val_accuracy: 0.2460 - val_loss: 2.0456
Epoch 3/50
32/32                 0s 6ms/step -
accuracy: 0.3205 - loss: 1.8978 - val_accuracy: 0.2700 - val_loss: 2.0115
Epoch 4/50
32/32                 0s 6ms/step -
accuracy: 0.3537 - loss: 1.8156 - val_accuracy: 0.2830 - val_loss: 1.9793
Epoch 5/50
32/32                 0s 6ms/step -
accuracy: 0.3778 - loss: 1.7591 - val_accuracy: 0.3130 - val_loss: 1.9302
Epoch 6/50
32/32                 0s 6ms/step -
accuracy: 0.3965 - loss: 1.7033 - val_accuracy: 0.3240 - val_loss: 1.9222
Epoch 7/50
32/32                 0s 6ms/step -
accuracy: 0.4123 - loss: 1.6439 - val_accuracy: 0.3370 - val_loss: 1.9208
Epoch 8/50
32/32                 0s 6ms/step -
accuracy: 0.4325 - loss: 1.5891 - val_accuracy: 0.3550 - val_loss: 1.8711
Epoch 9/50
32/32                 0s 6ms/step -
accuracy: 0.4568 - loss: 1.5299 - val_accuracy: 0.3570 - val_loss: 1.8994
Epoch 10/50
32/32                 0s 6ms/step -
accuracy: 0.4642 - loss: 1.4907 - val_accuracy: 0.3630 - val_loss: 1.8529
Epoch 11/50
32/32                 0s 6ms/step -
accuracy: 0.4845 - loss: 1.4405 - val_accuracy: 0.3510 - val_loss: 1.9033
```

```
Epoch 12/50
32/32            0s 6ms/step -
accuracy: 0.4888 - loss: 1.4242 - val_accuracy: 0.3520 - val_loss: 1.9000
Epoch 13/50
32/32            0s 6ms/step -
accuracy: 0.5163 - loss: 1.3640 - val_accuracy: 0.3740 - val_loss: 1.9086
Epoch 14/50
32/32            0s 7ms/step -
accuracy: 0.5383 - loss: 1.3044 - val_accuracy: 0.3620 - val_loss: 1.9885
Epoch 15/50
32/32            0s 6ms/step -
accuracy: 0.5415 - loss: 1.2856 - val_accuracy: 0.3660 - val_loss: 1.9488
32/32            0s 870us/step -
accuracy: 0.3670 - loss: 1.7664
Test accuracy: 0.3670
```



Training loss Training loss is the calculated error when the model makes predictions on the training data. It is updated after every forward and backward pass of the model during the training process.

Validation loss Validation loss evaluates the model's performance on a separate dataset (validation set) that the model has never seen during training. This metric provides an indication of how well the model generalizes to new data.