42ctu50dx

April 26, 2025

1 MNIST Digit Classification with Feedforward Neural Network

1.1 1. Design a basic deep learning model suitable for this image classification task

For this task, I designed a **simple feedforward neural network** (also called a fully connected neural network).
The model architecture is:

- Input layer: Flatten the 28x28 pixel image to a 784-dimensional vector.
- Hidden layer 1: Dense layer with 128 neurons and ReLU activation.
- Hidden layer 2: Dense layer with 64 neurons and ReLU activation.
- Output layer: Dense layer with 10 neurons (for digits 0–9) and softmax activation.

1.2 2. Explain why you chose the specific model architecture

I chose a simple **fully connected feedforward network** because:

- MNIST is a relatively **simple and well-studied dataset**, and dense layers can achieve good baseline performance without much complexity.
- **ReLU activation** introduces non-linearity and helps in faster training without vanishing gradient issues.
- **Softmax activation** in the output ensures that the model outputs probabilities across the 10 digit classes.
- The architecture is light enough to **train quickly** and is easy to improve later if needed.

This design sets a good foundation for quick experimentation and improvement.

1.3 3. Implement the code in Python using TensorFlow/Keras

(Implementation is done below following each step.)

1.3.1 a. Import the libraries

[1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
```

1.3.2 b. Load and preprocess the MNIST dataset

[2]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434             3s
0us/step
```

1.3.3 c. Normalize the images to the range [0, 1]

[3]:
```
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

1.3.4 d. Reshape for the dense model (Flatten the 2D 28x28 images)

[4]:
```
x_train = x_train.reshape((60000, 28 * 28))
x_test = x_test.reshape((10000, 28 * 28))
```

1.3.5 e. One-hot encode the labels

[5]:
```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

1.3.6 f. Build a simple feedforward neural network

[6]:
```
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
C:\Users\anves\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n
2kfra8p0\LocalCache\local-packages\Python312\site-
packages\keras\src\layers\core\dense.py:87: 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)
```

1.3.7 g. Compile the model

[7]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 10)	650

```
Total params: 109,386 (427.29 KB)

Trainable params: 109,386 (427.29 KB)

Non-trainable params: 0 (0.00 B)
```

1.3.8 h. Train the model

[8]:
```
history = model.fit(x_train, y_train, epochs=5, batch_size=64,␣
↪validation_split=0.1)
```

```
Epoch 1/5
844/844             4s 4ms/step -
accuracy: 0.8470 - loss: 0.5433 - val_accuracy: 0.9598 - val_loss: 0.1398
Epoch 2/5
844/844             3s 3ms/step -
accuracy: 0.9606 - loss: 0.1335 - val_accuracy: 0.9703 - val_loss: 0.0941
Epoch 3/5
844/844             3s 3ms/step -
accuracy: 0.9742 - loss: 0.0854 - val_accuracy: 0.9753 - val_loss: 0.0803
Epoch 4/5
844/844             3s 3ms/step -
accuracy: 0.9809 - loss: 0.0623 - val_accuracy: 0.9773 - val_loss: 0.0782
Epoch 5/5
844/844             3s 4ms/step -
accuracy: 0.9865 - loss: 0.0435 - val_accuracy: 0.9790 - val_loss: 0.0728
```

1.3.9 i. Evaluate the model

[9]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
```

```
313/313             1s 2ms/step -
```

```
accuracy: 0.9717 - loss: 0.1056
Test accuracy: 0.9761
```

1.4 4. Briefly describe how you would improve the model for better accuracy

To improve the model's accuracy:

- **Use Convolutional Neural Networks (CNNs):** CNNs are better suited for image data as they can capture spatial hierarchies.
- **Data Augmentation:** Apply transformations like rotation, zoom, and shift to artificially expand the dataset and improve generalization.
- **Increase Model Depth:** Add more hidden layers or neurons to increase the model's capacity to learn complex patterns.
- **Regularization:** Apply dropout or L2 regularization to avoid overfitting.
- **Learning Rate Scheduling:** Adjust the learning rate dynamically during training to improve convergence.
- **Use Batch Normalization:** Normalize the outputs of each layer to accelerate training and improve stability.