Charlie Misbach

CSCI-3470

Assignment #2

Due: 3/10/25

import torch

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

# Charlie Misbach

# Machine Learning - CSCI3470

# Assignment #2

# Due: 3/10/25

# Question #1: Download MNIST and create subplots of

# normalized grayscaled images

# Convert the image to a tensor of floats in [0, 1] (pixel values)

transform = transforms.Compose([

transforms.ToTensor() # converts [0,255] -> [0,1] (pixel values)

])

# Download AND load the MNIST training dataset with defined transform

train\_dataset = torchvision.datasets.MNIST(

root='./data', # folder to download/extract

train=True,

transform=transform,

download=True

)

# Download AND load MNIST test dataset

test\_dataset = torchvision.datasets.MNIST(

root='./data',

train=False,

transform=transform,

download=True

)

# Create a dataloader for the training dataset

train\_loader = torch.utils.data.DataLoader(

dataset=train\_dataset,

batch\_size=64,

shuffle=True

)

# Create 4x4 grid of subplots to display 16 images

fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(6,6))

axes = axes.flatten()

# loop through all 16 samples of the training set

for i in range(16):

img, label = train\_dataset[i] # each item is (image\_tensor, label)

# img is a tensor shape [1,28,28] (grayscale). Convert to numpy for plotting:

img\_np = img.squeeze().numpy()

# Display image

axes[i].imshow(img\_np, cmap='gray')

axes[i].set\_title(f"Label: {label}")

axes[i].axis('off')

plt.tight\_layout()

plt.show()

# Question #2: Implement LeNet-5 with Keras functional API

import tensorflow as tf

from tensorflow import keras

from keras import layers

# LeNet-5 (classic CNN arch) consists of

# 1. Convolutional Layers (feature extraction)

# 2. Subsampling (Pooling) layers (Reduces dimensions)

# 3. Fully connected layers (classification)

# Define the input shape for MNIST images (28x28 grayscale)

inputs = keras.Input(shape=(28, 28, 1))

# First Convolutional Layer (6 filters, 5x5 kernel, ReLU activation)

x = layers.Conv2D(filters=6, kernel\_size=(5,5), activation="relu", padding="same")(inputs)

x = layers.AveragePooling2D(pool\_size=(2,2))(x) # Subsampling (Average Pooling)

# Second Convolutional Layer: 16 filters, 5x5 kernel, ReLU activation

x = layers.Conv2D(filters=16, kernel\_size=(5,5), activation="relu")(x)

x = layers.AveragePooling2D(pool\_size=(2,2))(x) # Subsampling layer

# Flatten the extracted features before passing them into Dense layers

x = layers.Flatten()(x)

# Fully Connected Layer: 120 neurons

x = layers.Dense(units=120, activation="relu")(x)

# Fully Connected Layer: 84 neurons

x = layers.Dense(units=84, activation="relu")(x)

# Output Layer: 10 neurons for digit classification (Softmax activation)

outputs = layers.Dense(units=10, activation="softmax")(x)

# Create the Model

lenet\_model = keras.Model(inputs=inputs, outputs=outputs, name="LeNet-5")

# Print the model structure

lenet\_model.summary()

# Compile the model

lenet\_model.compile(

loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=False),

optimizer=keras.optimizers.Adam(),

metrics=["accuracy"]

)

# Load MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data()

# Normalize images to [0,1] range

x\_train = x\_train.astype("float32") / 255.0

x\_test = x\_test.astype("float32") / 255.0

# Reshape images to add the channel dimension (grayscale → (28,28,1))

x\_train = x\_train[..., None]

x\_test = x\_test[..., None]

# Process images in batches of 64

# Train for 10 epochs (10 passes through entire dataset)

# Use 20% of the train data for validation data (throughout the training)

training = lenet\_model.fit(

x\_train, y\_train,

batch\_size=64, # Process images in batches of 64

epochs=10, # Train for 10 epochs

validation\_split=0.2 # Use 20% of training data for validation

)

test\_scores = lenet\_model.evaluate(x\_test, y\_test, verbose=2)

print("Test loss:", test\_scores[0])

print("Test accuracy:", test\_scores[1])

# Question #3: Train the implemented LeNet5 model and plotting the

# training results (for test and validation)

# Extract loss values from training history

train\_loss = training.history['loss'] # Training loss

val\_loss = training.history['val\_loss'] # Validation loss

epochs = range(1, len(train\_loss) + 1) # Epoch numbers

# Plot Training vs Validation Loss

plt.figure(figsize=(8,6))

plt.plot(epochs, train\_loss, 'b-', label='Training Loss') # Blue Line - Training Loss

plt.plot(epochs, val\_loss, 'r-', label='Validation Loss') # Red Line - Validation Loss

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training vs. Validation Loss')

plt.legend()

plt.grid()

plt.show()

**2. Required Displays:**

A screenshot of a computer screen

AI-generated content may be incorrect.

A graph with a line graph

AI-generated content may be incorrect.

This graph shows the training process validation vs training loss, showing the accuracy throughout the training process on both the validation and the training set. It encounters data way more from the training dataset so obviously its going to be more trained on that specific set of images and labels.

A screenshot of a computer program

AI-generated content may be incorrect.

This image shows the training process, each epoch going over every image in the dataset once, it also shows the val and training loss which are what help the model to adjust its parameters to improve accuracy. After training the models accuracy (based upon the test dataset) is a solid 98% which is really good.

A screenshot of a computer

AI-generated content may be incorrect.

This image shows all of the layers and output shape of those layers of the model I created using Keras API.

**3. Discussion:**

I learned a lot from this assignment, beginning with how to download and load datasets from MNIST training datasets. I then learned how to prepare this dataset to properly train a model WELL. I also learned a lot about why splitting these datasets into train test and validation is very important. The training dataset is obviously what the model uses to train itself, the validation set is what makes sure it keeps itself in check during the training process (updates parameters, learning rate, etc, according to validation loss). I’ve already learned a good amount about test datasets use but in this case, it really showed me how important it is to include in the process, you never know how good or bad a model really is until you let it go through information it has yet to see. Also learned more about how different batch sizes affect the training process, as well as epochs. Also a loss function is very important as it can show whether a model is over / under fitting or just not properly learning. The most important part of this assignment was definitely using Keras API to create the model which was used. It really allowed me to understand the ordering of certain layers why they’re in the order they are and how they all work together.