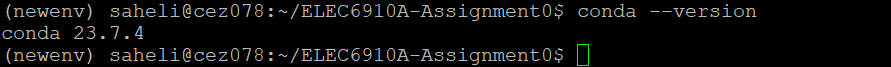
**ELEC6910A Assignment 0**

The assignment has been done on Linux environment where python environment was already set up. To verify I have checked the conda version:



**Note: I have run the code for two types of optimizers. There are separate folders for each optimizer. Inside each folder there is the main.py and Results folder which contain sample images, label distribution, the accuracy comparison, accuracy for MLP, CNN and Lenet using both Cross Entropy and L2 loss. Code Screenshots are provided for Adam Optimizer. The whole combined code log is attached:**

****

**电脑萤幕的截图

描述已自动生成Loading the MNIST dataset:**

Task 0:

we are required to: (1) Randomly pick some samples and view individual images and corresponding labels from the dataset; (2) Analyze the distribution of digits in the dataset; (3) Generate statistical summaries of the dataset.

**(1) Randomly pick some samples and view individual images and corresponding labels from the dataset**

I have defined a function named **view\_samples** to visualize 10 random samples. It generates a random index between 0 and the length of the images dataset minus one. This index is used to select a random image and its corresponding label.

Output:

文本

中度可信度描述已自动生成

**(2) Analyze the distribution of digits in the dataset**

I have defined a function named **digit\_distribution** to visualize the distribution of digit labels in the MNIST dataset which takes a single parameter “label”. It converts the PyTorch tensor of labels to a NumPy array and specifies that the data should be divided into 10 bins, one for each digit (0 through 9). Parameters like edgecolor (to set the color of the bin edges) and alpha (to set the transparency level of the bars) have been specified.

Output:

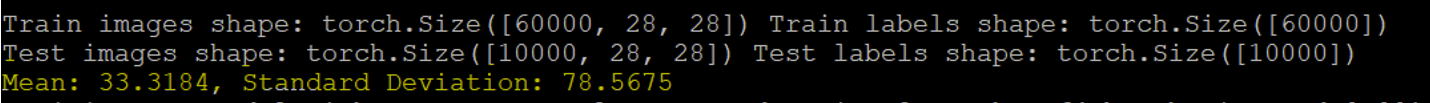
图表, 直方图

描述已自动生成

**(3) Generate statistical summaries of the dataset**

I have defined a function named **generate\_statistical\_summary** to calculate the mean and standard deviation of the pixel values in the images.

Output:



References:

<https://github.com/MrDataScience/tutorials/blob/master/Data/MNIST/How%20To%20Plot%20MNIST%20Digits%20Using%20Matplotlib.ipynb>

<https://blog.devgenius.io/understanding-handwritten-digit-recognition-using-k-nearest-neighbors-knn-da677e87c8ac>

<https://github.com/ageron/handson-ml2/blob/master/03_classification.ipynb>

Task 1:

This requirement document outlines the specifications for implementing three different models: Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and LeNet-5 for the classification of the MNIST dataset.

We are required to develop the following vision models on the MNIST dataset: (1)MLP:A simple feedforward neural network with one or more layers of nodes; (2)CNN: A deep learning model that excels at image recognition tasks; (3)LeNet-5: A classic convolutional neural network architecture designed specifically for handwritten digit recognition. For the MLP and CNN, you are allowed to use ANY hyperparameters like the number of the hidden layers.

**(1)MLP: A simple feedforward neural network with one or more layers of nodes**

Part of the code was already provided. I have defined the first and second hidden layer and the output layer that transforms the features from the last hidden layer to the number of classes.

The forward method, after ensuring the input shape, flattens the 2D image into a 1D vector. I have applied ReLU activation function to the output of each layer except the last one. The final output is passed through the output layer.

**(2) CNN: A deep learning model that excels at image recognition tasks**

Part of the code was already provided. I have defined First convolutional layer which takes a single-channel input and outputs 32 feature maps. Kernel size is 3x3 with padding 1 to maintain the spatial dimensions.

Defined Second convolutional layer which takes 32 feature maps and outputs 64 feature maps. **Kernel size is 3x3 with padding 1. Also defined Max-pooling layer** which reduces the spatial dimensions by a factor of 2. First fully connected layer which takes the flattened tensor from the convolutional layers and outputs 128 features and Output Layer have been defined.

The forward method, after ensuring the input shape applies the first convolutional layer followed by the ReLU activation function. Then it applies the second convolutional layer, followed by ReLU activation and max-pooling. It flattens the tensor into a 1D vector while keeping the batch size. It applies the first fully connected layer followed by the ReLU activation function and at last the output layer.

**(3)LeNet-5: A classic convolutional neural network architecture designed specifically for handwritten digit recognition**

Part of the code was already provided. I have defined First convolutional layer which takes a single-channel input (grayscale image) and outputs 6 feature maps. The kernel size is 5x5, Second convolutional layer which takes 6 feature maps and outputs 16 feature maps. **The kernel size is 5x5 followed by Average pooling layer** which reduces the spatial dimensions by a factor of 2. After that First fully connected layer which takes the flattened tensor from the convolutional layers and outputs 120 features, Second fully connected layer which takes the 120 features from the first fully connected layer and outputs 84 features and Output layer which takes the 84 features from the second fully connected layer have been defined.

In the forward method, the assert statement ensures that the input tensor has the expected shape. Then we have applied the first convolutional layer to the input tensor x, the ReLU activation function to introduce non-linearity and the average pooling layer to reduce the spatial dimensions by a factor of 2. Applied the second convolutional layer, the ReLU activation function and the average pooling layer. Flattened the output tensor from the convolutional and pooling layers into a 1D vector, while keeping the batch size.

The first fully connected layer transforms the flattened tensor into a 120-dimensional feature vector. The second fully connected layer further transforms the 120-dimensional feature vector into an 84-dimensional feature vector. The output layer maps the 84-dimensional feature vector to the number of classes specified by num\_of\_classes.

References:

<https://pytorch.org/tutorials/beginner/introyt/modelsyt_tutorial.html>

<https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/>

Zhang, Jingsi, et al. "A novel deep LeNet-5 convolutional neural network model for image recognition." *Computer Science and Information Systems* 19.3 (2022): 1463-1480.

Task 2:

Cross-entropy loss, also known as log loss, is a crucial function used in classification problems with two or more classes. It measures the performance of a classification model whose output is a probability value between 0 and 1. You can also choose the L2 loss for supervision.

I have used two different loss functions: **CrossEntropyLoss and L2 Loss**. It also compares the performance of two optimization algorithms (Adam, RMSprop).

**CrossEntropyLoss:** Measures the performance of a classification model whose output is a probability value between 0 and 1.

**L2 Loss:** Measures the mean squared difference between the predicted probabilities and the one-hot encoded true labels.

The accuracy for each combination of model, loss function (both cross entropy and L2), and optimizer is recorded.

Cross Entropy Loss Works directly with logits and integer labels. It internally handles the softmax conversion and computes the loss efficiently.

L2 Loss Requires the outputs to be probabilities and the labels to be one-hot encoded vectors. This allows for direct computation of the mean squared error between the predicted probabilities and the true labels. Since L2 Loss doesn't include a built-in softmax function, it is necessary to manually apply softmax to the logits to convert them into probabilities before comparing them to the one-hot encoded labels.

References:

<https://discuss.pytorch.org/t/which-loss-function-for-one-hot-encoded-labels/29920>

Mao, Anqi, Mehryar Mohri, and Yutao Zhong. "Cross-entropy loss functions: Theoretical analysis and applications." *International conference on Machine learning*. PMLR, 2023.

Observation:

图表, 条形图

描述已自动生成I have used both Cross Entropy loss and L2 loss. For both losses, we can see Lenet5 performed the best. In the below plot we can see the comparison between the accuracy of MLP, CNN and Lenet-5 for Cross Entropy loss and L2 loss. For the below plot, I have used Adam Optimizer.

Comparison between Adam and RMSprop

|  |  |
| --- | --- |
| Adam | RMSprop |
| Cross Entropy L2  MLP 0.9414 0.9048  CNN 0.9788 0.9642  Lenet-5 0.9864 0.9853 | Cross Entropy L2  MLP 0.9516 0.9301  CNN 0.9647 0.9735  Lenet-5 0.9869 0.9868 |