# Homework 2

Step 1: Data

1. **How many data samples are included in the dataset?**
    - The MNIST dataset consists of 70,000 grayscale images of handwritten digits (0 through 9). It is split into 60,000 training images and 10,000 test images. Each image is 28x28 pixels.
2. **Which problem will this dataset try to address?**
    - The MNIST dataset is typically used to address the problem of classifying handwritten digits. The task is to assign a label from 0 to 9 to each image, which corresponds to the digit represented by the image.
3. **What is the minimum value and the maximum value in the dataset?**
    - The pixel values in the original dataset range from 0 to 255. However, in the provided code, these pixel values are normalized to be within the range of 0 to 1 by dividing by 255.
4. **What is the dimension of each data sample?**
    - Each image in the MNIST dataset is 28x28 pixels, so the dimension of each data sample is 28x28.
5. **Does the dataset have any missing information? E.g., missing features.**
    - The MNIST dataset is quite clean and is widely recognized for having no missing values. It's a well-prepared dataset, which is one of the reasons it is often used for educational purposes.
6. **What is the label of this dataset?**
    - The label of each image in this dataset is an integer from 0 to 9, representing the digit displayed in the image. In the provided code, these labels are one-hot encoded into vectors of length 10, with a 1 at the index corresponding to the digit and 0s elsewhere.
7. **How many percent of data will you use for training, validation, and testing?**
    - From the provided code, it seems that:
        - The first 50,000 samples of the original training set (out of 60,000) are used for training, which is about 83.33% of the training data and 71.43% of the total data.
        - The last 10,000 samples of the original training set are used for validation, which is about 16.67% of the training data and 14.29% of the total data.
        - The test set provided by the MNIST dataset, which consists of 10,000 samples, is used for testing, which is 14.29% of the total data.

8. **What kind of data pre-processing will you use for your training dataset?**
   - From the provided code, the following preprocessing steps are used:
     - Normalization: The pixel values are normalized to be between 0 and 1 by dividing by 255.
     - One-hot encoding: The labels are one-hot encoded into vectors of length 10, with a 1 at the index corresponding to the label and 0s elsewhere.
     - Data split: The original training data is split into a smaller training set and a validation set to evaluate the model during training.

Step 2: Model

| Model | Accuracy |
|---|---|
| DNN | 0.9771 |
| ConvNet | 0.9930 |
| ResNet | 0.9933 |

Step 3: Objective

I have used Cross-entropy as the loss function to train my models.

Step 4: Optimization

The optimization algorithm selected for training the deep learning models is Adam.

**Reasons for Using Adam:**

1. **Adaptive Learning Rates:** Adam automatically adjusts the learning rate during training. It computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients, which can be particularly useful when optimizing problems with noisy or sparse gradients.
2. **Suitable for Problems with Noisy Gradients:** Adam can be particularly effective in problems with noisy or sparse gradients, making it a good choice for various neural network training scenarios.
3. **Efficiency:** Adam is computationally efficient and has relatively low memory requirements, which is crucial for problems with large datasets or parameter space.

4. **Empirical Success:** Adam has been widely adopted in the machine learning community and has been successful in various empirical evaluations, suggesting that it's a robust choice for a wide range of problems.
5. **Applicability:** Adam is appropriate for non-convex optimization problems, which is typically the case in training deep learning models.

Step 5: Model Selection

| Model | LR: 0.1 | LR: 0.01 | LR: 0.001 | LR: 0.0001 |
|---|---|---|---|---|
| DNN | 0.1042 | 0.9132 | 0.9632 | 0.9771 |
| ConvNet | 0.1030 | 0.9670 | 0.9897 | 0.9930 |
| ResNet | 0.9159 | 0.4527 | 0.9933 | 0.9928 |

**1. Best Performance:**

- The **ResNet** with a learning rate of **0.001** yields the highest accuracy of **0.9933**, making it the top-performing model among those tested.

**2. Experience with Different Learning Rates:**

- **DNN:**
    - At LR=0.1, it underperforms dramatically, suggesting it might be overshooting the minimum.
    - It consistently improves with smaller learning rates.
- **ConvNet:**
    - Again, LR=0.1 seems too high.
    - Performs exceedingly well at smaller learning rates, especially 0.001 and 0.0001.
- **ResNet:**
    - Quite robust even at LR=0.1, but disastrous at 0.01.
    - Outperforms other models at LRs of 0.001 and 0.0001.

**3. Trying Other Learning Rates:**

- Given the table, it might be worthwhile to explore learning rates between 0.01 and 0.001, such as **0.005**, since for some models (like ResNet), 0.01 performed poorly, while 0.001 was optimal. Exploring the intermediate values could unearth a better compromise for stability and convergence speed.

**4. Other Learning Rate Techniques:**

- **Adaptive learning rates** (like AdaGrad, RMSProp) could be experimented with, as they dynamically adjust the learning rate during training.
- **Learning rate schedules** or **learning rate warm-up**: Gradually increasing the learning rate might improve convergence in some cases. Likewise, learning rate annealing (gradually decreasing it) could be helpful.
- **Cyclical Learning Rates** (CLR): Instead of keeping the learning rate constant or decreasing it, CLR varies cyclically within a range, which sometimes enables faster convergence.

**5. Avoiding Overfitting and Underfitting:**

- **Overfitting:**
    - **Regularization:** Applying L1, L2, or Dropout regularization.
    - **Data Augmentation:** Increasing the effective size of the training data.
    - **Early Stopping:** Monitor validation loss and stop training when it starts increasing.
    - **Reduce Model Complexity:** Using a simpler model.
    - **Ensembling:** Using techniques like bagging can also help to mitigate overfitting.
- **Underfitting:**
    - **Increasing Model Complexity:** Sometimes underfitting occurs due to the model being too simple to capture underlying patterns.
    - **Feature Engineering:** Including additional relevant features.
    - **Reducing Regularization:** If the model is regularized, reducing the regularization term might allow the model to learn better from the training data.
    - **Increasing Training Time:** Allowing the model more epochs to learn from the data if it doesn’t start overfitting.

Homework 2

Step 6: Model Performance

A. Accuracy Table:

| Model | LR: 0.1 | LR: 0.01 | LR: 0.001 | LR: 0.0001 |
|---|---|---|---|---|
| DNN | 0.1042 | 0.9132 | 0.9632 | 0.9771 |
| ConvNet | 0.1030 | 0.9670 | 0.9897 | 0.9930 |
| ResNet | 0.9159 | 0.4527 | 0.9933 | 0.9928 |

B. F1 Score Table:

| Model | LR: 0.1 | LR: 0.01 | LR: 0.001 | LR: 0.0001 |
|---|---|---|---|---|
| DNN | 0.0186 | 0.8973 | 0.9707 | 0.9747 |
| ConvNet | 0.0183 | 0.9583 | 0.9894 | 0.9892 |
| ResNet | 0.9152 | 0.3859 | 0.9933 | 0.9928 |

Find all the plots in the Github repository.