Machine Learning
LABORATORY: Gradient Descent Homework

NAME:	STUDENT ID#:

Objectives:

- Understand and implement Mini-batch SGD (Algorithm 7.2).
- Extend the algorithm to support Momentum (Algorithm 7.3) and Adam optimization (Algorithm 7.4).
- Apply each optimizer to a binary classification task using the MNIST dataset.
- Evaluate and compare model behavior through accuracy and misclassified samples.
- Practice implementing mathematical update rules directly from textbook equations using NumPy.

Part 1. Instruction

- In this assignment, you will implement **Mini-batch Stochastic Gradient Descent (SGD)** and its extensions using **Algorithms 7.2, 7.3, and 7.4**.
- Your task is to build a **binary classifier** to determine whether an MNIST image matches a specific digit or not (e.g., "Is this a 4 or not?").
- You will implement **three** different methods: **Mini-batch SGD** (Algorithm 7.2), **SGD with Momentum** (Algorithm 7.3) and **Adam Optimizer** (Algorithm 7.4)
- You may write all algorithms in **one file with selectable modes**, or in **three separate files**.
- The code must be implemented **entirely with NumPy**. Do not use external machine learning libraries (e.g., scikit-learn, PyTorch).
- The model should output:
    - Final **accuracy** on the test set.
    - At least **five misclassified test samples**, with true and predicted labels shown.
- Use the **last digit of your student ID** as the `TARGET_DIGIT` for binary classification (e.g., ID ending in 7 → `TARGET_DIGIT = 7`).

Part 2. Arithmetic Instructions.

Algorithm 7.2: Mini-batch stochastic gradient descent

Input: Training set of data points indexed by $n \in \{1, \ldots, N\}$
Batch size B
Error function per mini-batch $E_{n:n+B-1}(\mathbf{w})$
Learning rate parameter η
Initial weight vector $\mathbf{w}$
Output: Final weight vector $\mathbf{w}$

```
n ← 1
repeat
    w ← w − η∇E_{n:n+B−1}(w) // weight vector update
    n ← n + B
    if n > N then
        shuffle data
        n ← 1
    end if
until convergence
return w
```

Algorithm 7.3: Stochastic gradient descent with momentum

Input: Training set of data points indexed by $n \in \{1, \ldots, N\}$
Batch size B
Error function per mini-batch $E_{n:n+B-1}(\mathbf{w})$
Learning rate parameter η
Momentum parameter μ
Initial weight vector $\mathbf{w}$
Output: Final weight vector $\mathbf{w}$

```
n ← 1
Δw ← 0
repeat
    Δw ← −η∇E_{n:n+B−1}(w) + μΔw // calculate update term
    w ← w + Δw // weight vector update
    n ← n + B
    if n > N then
        shuffle data
        n ← 1
    end if
until convergence
return w
```

Algorithm 7.4: Adam optimization

Input: Training set of data points indexed by $n \in \{1, \dots, N\}$
Batch size B
Error function per mini-batch $E_{n:n+B-1}(\mathbf{w})$
Learning rate parameter η
Decay parameters β_1 and β_2
Stabilization parameter δ
Output: Final weight vector $\mathbf{w}$

$n \leftarrow 1$
$\mathbf{s} \leftarrow \mathbf{0}$
$\mathbf{r} \leftarrow \mathbf{0}$
repeat
  Choose a mini-batch at random from $\mathcal{D}$
  $\mathbf{g} = -\nabla E_{n:n+B-1}(\mathbf{w})$ `// evaluate gradient vector`
  $\mathbf{s} \leftarrow \beta_1 \mathbf{s} + (1 - \beta_1)\mathbf{g}$
  $\mathbf{r} \leftarrow \beta_2 \mathbf{r} + (1 - \beta_2)\mathbf{g} \odot \mathbf{g}$ `// element-wise multiply`
  $\widehat{\mathbf{s}} \leftarrow \mathbf{s}/(1 - \beta_1^\tau)$ `// bias correction`
  $\widehat{\mathbf{r}} \leftarrow \mathbf{r}/(1 - \beta_2^\tau)$ `// bias correction`
  $\Delta\mathbf{w} \leftarrow -\eta \dfrac{\widehat{\mathbf{s}}}{\sqrt{\widehat{\mathbf{r}}} + \delta}$ `// element-wise operations`
  $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$ `// weight vector update`
  $n \leftarrow n + B$
  if $n + B > N$ **then**
    shuffle data
    $n \leftarrow 1$
  end if
until convergence
return $\mathbf{w}$

Part 3. Code Template.

Step	Procedure
1	#Load Dataset `import struct` `import numpy as np` `import matplotlib.pyplot as plt` `# ==========Load IDX Files ==========` `def load_images(filename):` `    with open(filename, 'rb') as f:` `        _, num, rows, cols = struct.unpack(">IIII", f.read(16))` `        images=np.frombuffer(f.read(), dtype=np.uint8)` `        images = images[:(len(images)//(rows * cols)) * rows * cols]` `        return images.reshape(-1, rows * cols).astype(np.float32) / 255.0` `def load_labels(filename):` `    with open(filename, 'rb') as f:` `        _, num = struct.unpack(">II", f.read(8))` `        labels = np.frombuffer(f.read(),`

	`dtype=np.uint8)` `        return labels[:num]`
2	`# ========== 1. Sigmoid Function ==========` `def sigmoid(z):` `    # TODO: Implement sigmoid function` `    pass` `# ====== 2. Mini Batch SGD: Algorithm 7.2 ======` `def sgd_minibatch(X, y, eta=0.01, max_iters=10000,` `batch_size=64):` `    pass` `# ===== 3. Mini Batch SGD with Momentum: Algorithm 7.3 =====` `def sgd_minibatch_momentum(X, y, eta=0.01, max_iters=10000,` `batch_size=64, momentum=0.9):` `    pass` `# ====== 4. Adam Optimizer: Algorithm 7.4 ======` `def sgd_Adam(X, y, eta=0.001, max_iters=10000, batch_size=64,` `beta1=0.9, beta2=0.999, delta=1e-8):` `    pass`
3	`# ==========Show Misclassified Samples ==========` `def show_misclassified(X, true_labels, pred_labels,` `max_show=10):` `    mis_idx = np.where(true_labels !=` `pred_labels)[0][:max_show]` `    plt.figure(figsize=(10, 2))` `    for i, idx in enumerate(mis_idx):` `        plt.subplot(1, len(mis_idx), i + 1)` `        plt.imshow(X[idx, 1:].reshape(28, 28), cmap='gray')` `        plt.axis('off')` `        plt.title(f"T:{true_labels[idx]}` `P:{pred_labels[idx]}")` `    plt.suptitle("Misclassified Samples")` `    plt.show()`
4	`# ========== 3. Main ==========` `if __name__ == "__main__":` `    # === Load Data ===` `    X_train = load_images("train-images.idx3-ubyte__")` `    y_train = load_labels("train-labels.idx1-ubyte__")` `    X_test = load_images("t10k-images.idx3-ubyte__")` `    y_test = load_labels("t10k-labels.idx1-ubyte__")` `    # === Choose binary classification target digit ===` `    TARGET_DIGIT = 0    # TODO: Fill in (0 to 9)` `    y_train_bin = np.where(y_train == TARGET_DIGIT, 1, 0)` `    y_test_bin = np.where(y_test == TARGET_DIGIT, 1, 0)`

	`` # === Add bias term === X_train = np.hstack([np.ones((X_train.shape[0], 1)), X_train]) X_test = np.hstack([np.ones((X_test.shape[0], 1)), X_test]) # === Set parameters === # === Train === # === Predict === # === Evaluate === # === Show Misclassified Samples === ```

Grading Assignment & Submission (70% Max)

Implementation(50%):

1. **Correctly** implemented, runs, shows accuracy and sample misclassification of:
   a. **(15%) Mini-batch SGD (Algorithm 7.2)**
   b. **(10%) SGD with momentum (Algorithm 7.3)**
   c. **(5%) SGD with Nesterov momentum (Eq. 7.34)**
   d. **(15%) Adam Optimizer (Algorithm 7.4)**
2. **(5%)** Compare the accuracy and test sample for each algorithm.

Question(20%):

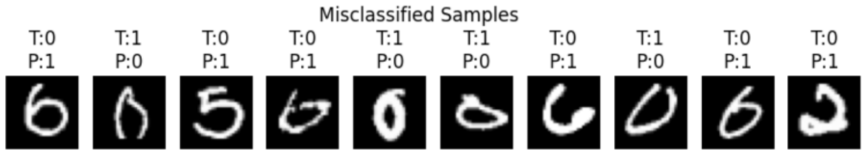
1. (7%) Which optimizer gave you the best test accuracy? Why do you think it performed better than the others?
2. (8%) What is the differences in learning stability, convergence speed, or misclassification types across all algorithm? Please explain with examples or observation from your results.
3. (7%) How did your choice of learning rate, batch size, or momentum affect each optimizer? What values worked best in your experiments?

Submission :

1. Report: Answer all conceptual questions. Include screenshots of your results in the last pages of this PDF File.
2. Code: Submit your complete Python script in either .py or .ipynb format.
3. Upload both your report and code to the E3 system (**Labs4_Homework_Assignment**). Name your files correctly:
   a. Report: StudentID_Lab4_Homework.pdf
   b. Code: StudentID_Lab4_Homework.py or StudentID_Lab4_Homeworkipynb
4. Deadline: Sunday, 21:00 PM
5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

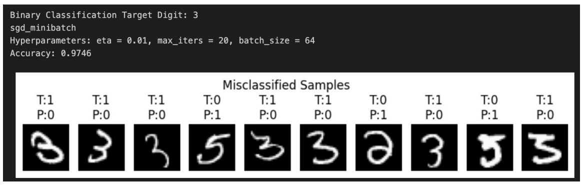
Example Output (Just for reference):

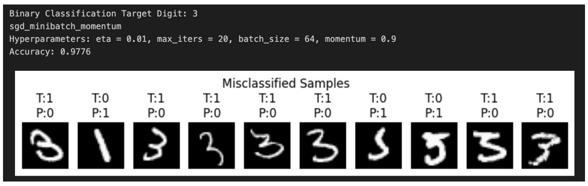
```
[INFO] Header: 60000 images, 28x28
[INFO] Loading 60000 images based on file size
[INFO] Loading 60000 labels based on file size
[INFO] Header: 10000 images, 28x28
[INFO] Loading 10000 images based on file size
[INFO] Loading 10000 labels based on file size

[INFO] Binary classification: '0' vs not-0

Test Accuracy (is 0 or not): 0.9927
```

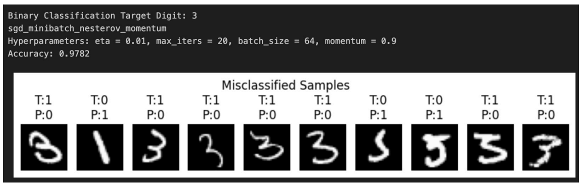


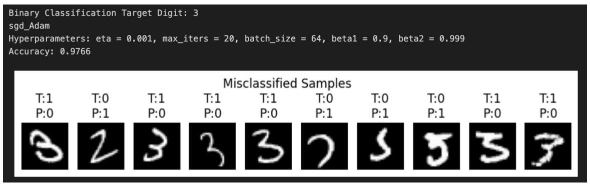
Code Results and Answer:

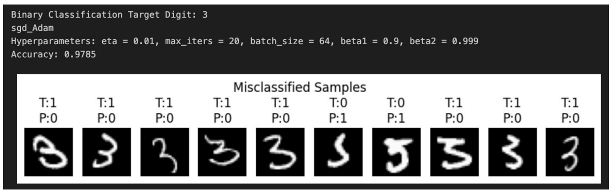
A1: According to my experiments, **Adam** optimizer performs best in terms of test accuracy. In addition to the effect of momentum (momentum), Adam also takes into account the past quarter momentum (second-order momentum) and the learning rate adjustment throughout cognition (adaptive learning rate), which effectively avoids local oscillations and improves the speed and stability of convergence. Therefore, the highest accuracy was obtained in this experiment.

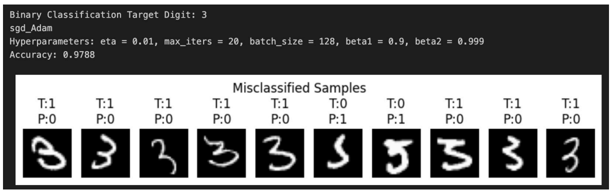








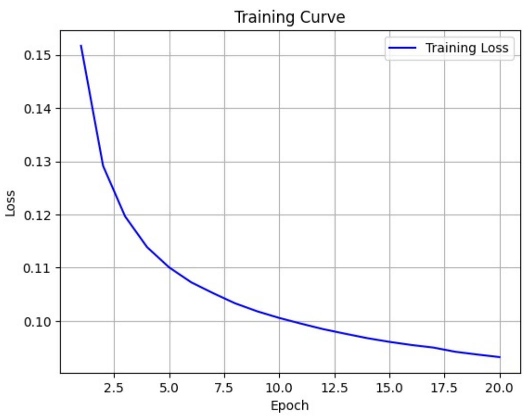




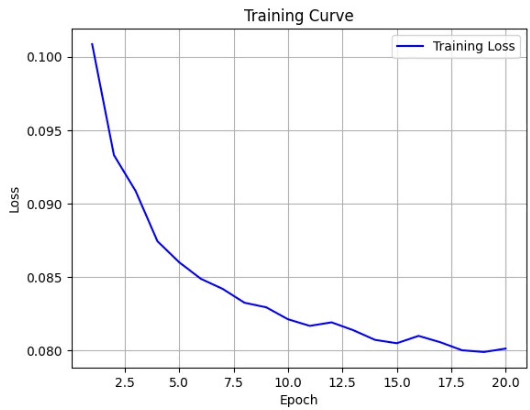
A2:

- Stability: The pure SGD method is prone to oscillations, and the convergence path fluctuates significantly. After using Momentum or Nesterov Momentum, the volatility is significantly reduced, and Adam has the best stability.
- Convergence speed: SGD is the slowest and requires more epochs to converge; Momentum and Nesterov have improved some speed; Adam is the fastest and can usually quickly achieve better accuracy within a few epochs.
- Misclassification types: SGD tends to make random misjudgments, the Momentum series is more likely to misjudge numbers with similar shapes, and Adam makes fewer misjudgments, but occasionally misjudges special handwriting or blurred images. For example, the numbers "3" and "5" are sometimes easily confused.

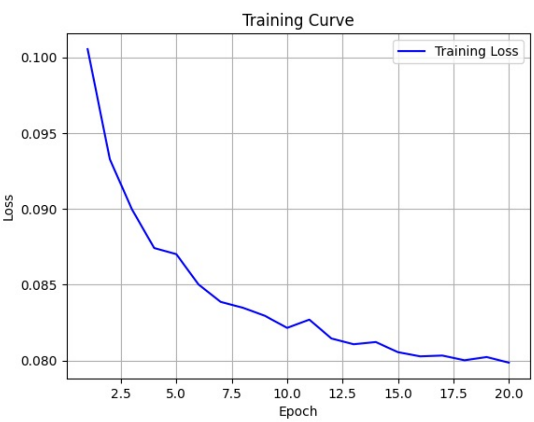
```
Binary Classification Target Digit: 3
sgd_minibatch
Hyperparameters: eta = 0.01, max_iters = 20, batch_size = 64
Accuracy: 0.9746
```



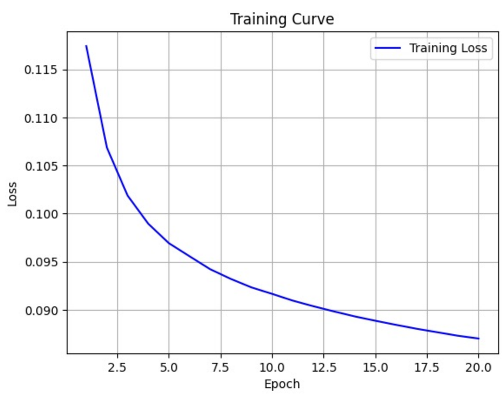

```
Binary Classification Target Digit: 3
sgd_minibatch_momentum
Hyperparameters: eta = 0.01, max_iters = 20, batch_size = 64, momentum = 0.9
Accuracy: 0.9776
```




```
Binary Classification Target Digit: 3
sgd_minibatch_nesterov_momentum
Hyperparameters: eta = 0.01, max_iters = 20, batch_size = 64, momentum = 0.9
Accuracy: 0.9782
```




```
Binary Classification Target Digit: 3
sgd_Adam
Hyperparameters: eta = 0.001, max_iters = 20, batch_size = 64, beta1 = 0.9, beta2 = 0.999
Accuracy: 0.9766
```

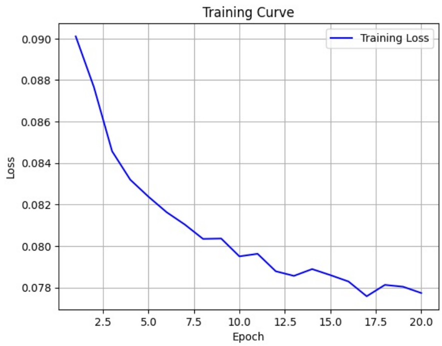


A3: Effect of parameters on optimizer:

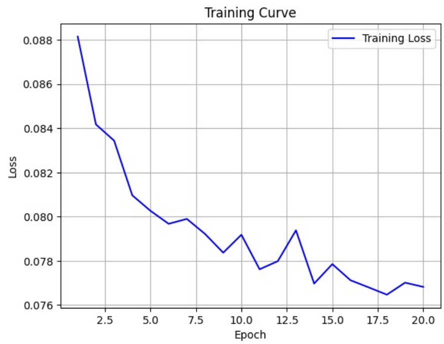
- A learning rate (η) that is too large may result in excessive dynamics and failure to converge; however, Adam can be adaptively adjusted. The best learning rate is about 0.01 which is better than 0.001.
- A good balance is achieved when the batch size is 128. If the batch size is too small, training will be fast but the fluctuation will be obvious.

BEST

```
Binary Classification Target Digit: 3
sgd_Adam
Hyperparameters: eta = 0.01, max_iters = 20, batch_size = 128, beta1 = 0.9, beta2 = 0.999
Accuracy: 0.9788
```




```
Binary Classification Target Digit: 3
sgd_Adam
Hyperparameters: eta = 0.01, max_iters = 20, batch_size = 64, beta1 = 0.9, beta2 = 0.999
Accuracy: 0.9785
```




```
Binary Classification Target Digit: 3
sgd_Adam
Hyperparameters: eta = 0.001, max_iters = 20, batch_size = 64, beta1 = 0.9, beta2 = 0.999
Accuracy: 0.9766
```