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`).

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan

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 $\mathbf{w}$

$n \leftarrow 1$
repeat
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_{n:n+B-1}(\mathbf{w})$ `// weight vector update`
  $n \leftarrow n + B$
  if $n > N$ **then**
    shuffle data
    $n \leftarrow 1$
  end if
until convergence
return $\mathbf{w}$

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 $\mathbf{w}$

$n \leftarrow 1$
$\Delta\mathbf{w} \leftarrow \mathbf{0}$
repeat
  $\Delta\mathbf{w} \leftarrow -\eta \nabla E_{n:n+B-1}(\mathbf{w}) + \mu \Delta\mathbf{w}$ `// calculate update term`
  $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$ `// weight vector update`
  $n \leftarrow n + B$
  if $n > N$ **then**
    shuffle data
    $n \leftarrow 1$
  end if
until convergence
return $\mathbf{w}$

Algorithm 7.4: Adam optimization

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
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
  $\hat{\mathbf{s}} \leftarrow \mathbf{s}/(1 - \beta_1^\tau)$ // bias correction
  $\hat{\mathbf{r}} \leftarrow \mathbf{r}/(1 - \beta_2^\tau)$ // bias correction
  $\Delta\mathbf{w} \leftarrow -\eta \dfrac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$ // 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 :

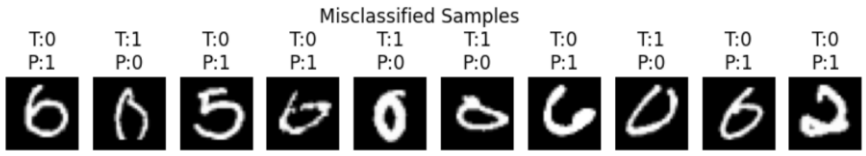
1. Report: Answer all conceptual questions. Include screenshots of your results in the last pages of this PDF File.
2. Code: Submit your complete Python script in either .py or .ipynb format.
3. Upload both your report and code to the E3 system (**Labs4_Homework_Assignment**). Name your files correctly:
   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:

Target digit for binary classification: 4

=== Training Mini-batch SGD ===

Iteration 0, Loss: 0.6372, Accuracy: 0.9026

Iteration 100, Loss: 0.2554, Accuracy: 0.9026

Iteration 200, Loss: 0.2146, Accuracy: 0.9029

Iteration 300, Loss: 0.1887, Accuracy: 0.9189

Iteration 400, Loss: 0.1715, Accuracy: 0.9373

Iteration 500, Loss: 0.1589, Accuracy: 0.9436

Iteration 600, Loss: 0.1500, Accuracy: 0.9483

Iteration 700, Loss: 0.1422, Accuracy: 0.9547

Iteration 800, Loss: 0.1357, Accuracy: 0.9595

Iteration 900, Loss: 0.1308, Accuracy: 0.9600

Iteration 1000, Loss: 0.1264, Accuracy: 0.9626

Iteration 1100, Loss: 0.1226, Accuracy: 0.9630

Iteration 1200, Loss: 0.1194, Accuracy: 0.9638

Iteration 1300, Loss: 0.1164, Accuracy: 0.9658

Iteration 1400, Loss: 0.1137, Accuracy: 0.9674

Iteration 1500, Loss: 0.1112, Accuracy: 0.9673

Iteration 1600, Loss: 0.1089, Accuracy: 0.9685

Iteration 1700, Loss: 0.1070, Accuracy: 0.9687

Iteration 1800, Loss: 0.1054, Accuracy: 0.9710

Iteration 1900, Loss: 0.1035, Accuracy: 0.9701

Iteration 2000, Loss: 0.1021, Accuracy: 0.9705

Iteration 2100, Loss: 0.1010, Accuracy: 0.9692

Iteration 2200, Loss: 0.0994, Accuracy: 0.9720

Iteration 2300, Loss: 0.0982, Accuracy: 0.9725

Iteration 2400, Loss: 0.0970, Accuracy: 0.9725

Iteration 2500, Loss: 0.0959, Accuracy: 0.9721

Iteration 2600, Loss: 0.0948, Accuracy: 0.9732

Iteration 2700, Loss: 0.0939, Accuracy: 0.9729

Iteration 2800, Loss: 0.0929, Accuracy: 0.9737

Iteration 2900, Loss: 0.0922, Accuracy: 0.9744

SGD Test Accuracy: 0.9723

=== Training SGD with Momentum ===

Iteration 0, Loss: 0.6314, Accuracy: 0.9026

Iteration 100, Loss: 0.1250, Accuracy: 0.9575

Iteration 200, Loss: 0.1018, Accuracy: 0.9691

Iteration 300, Loss: 0.0905, Accuracy: 0.9738

Iteration 400, Loss: 0.0846, Accuracy: 0.9760

Iteration 500, Loss: 0.0815, Accuracy: 0.9744

Iteration 600, Loss: 0.0773, Accuracy: 0.9775

Iteration 700, Loss: 0.0751, Accuracy: 0.9767

Iteration 800, Loss: 0.0735, Accuracy: 0.9770

Iteration 900, Loss: 0.0714, Accuracy: 0.9786

Iteration 1000, Loss: 0.0702, Accuracy: 0.9784

Iteration 1100, Loss: 0.0691, Accuracy: 0.9790

Iteration 1200, Loss: 0.0683, Accuracy: 0.9786

Iteration 1300, Loss: 0.0677, Accuracy: 0.9788

Iteration 1400, Loss: 0.0665, Accuracy: 0.9800

Iteration 1500, Loss: 0.0659, Accuracy: 0.9800

Iteration 1600, Loss: 0.0652, Accuracy: 0.9804

Iteration 1700, Loss: 0.0647, Accuracy: 0.9804

Iteration 1800, Loss: 0.0647, Accuracy: 0.9803

Iteration 1900, Loss: 0.0639, Accuracy: 0.9804

Iteration 2000, Loss: 0.0634, Accuracy: 0.9805

Iteration 2100, Loss: 0.0630, Accuracy: 0.9806

Iteration 2200, Loss: 0.0627, Accuracy: 0.9806

Iteration 2300, Loss: 0.0622, Accuracy: 0.9808

Iteration 2400, Loss: 0.0621, Accuracy: 0.9811

Iteration 2500, Loss: 0.0618, Accuracy: 0.9811

Iteration 2600, Loss: 0.0614, Accuracy: 0.9811

Iteration 2700, Loss: 0.0611, Accuracy: 0.9816

Iteration 2800, Loss: 0.0609, Accuracy: 0.9814

Iteration 2900, Loss: 0.0606, Accuracy: 0.9814

SGD with Momentum Test Accuracy: 0.9802

=== Training SGD with Nesterov Momentum ===

Iteration 0, Loss: 0.6386, Accuracy: 0.9026

Iteration 100, Loss: 0.1255, Accuracy: 0.9624

Iteration 200, Loss: 0.1019, Accuracy: 0.9698

Iteration 300, Loss: 0.0912, Accuracy: 0.9728

Iteration 400, Loss: 0.0849, Accuracy: 0.9746

Iteration 500, Loss: 0.0820, Accuracy: 0.9739

Iteration 600, Loss: 0.0773, Accuracy: 0.9762

Iteration 700, Loss: 0.0750, Accuracy: 0.9775

Iteration 800, Loss: 0.0730, Accuracy: 0.9782

Iteration 900, Loss: 0.0722, Accuracy: 0.9775

Iteration 1000, Loss: 0.0706, Accuracy: 0.9789

Iteration 1100, Loss: 0.0692, Accuracy: 0.9788

Iteration 1200, Loss: 0.0685, Accuracy: 0.9788

Iteration 1300, Loss: 0.0674, Accuracy: 0.9796

Iteration 1400, Loss: 0.0665, Accuracy: 0.9798

Iteration 1500, Loss: 0.0659, Accuracy: 0.9801

Iteration 1600, Loss: 0.0656, Accuracy: 0.9804

Iteration 1700, Loss: 0.0650, Accuracy: 0.9801

Iteration 1800, Loss: 0.0643, Accuracy: 0.9807

Iteration 1900, Loss: 0.0638, Accuracy: 0.9804

Iteration 2000, Loss: 0.0633, Accuracy: 0.9806

Iteration 2100, Loss: 0.0634, Accuracy: 0.9808

Iteration 2200, Loss: 0.0629, Accuracy: 0.9808

Iteration 2300, Loss: 0.0623, Accuracy: 0.9811

Iteration 2400, Loss: 0.0619, Accuracy: 0.9811

Iteration 2500, Loss: 0.0618, Accuracy: 0.9811

Iteration 2600, Loss: 0.0617, Accuracy: 0.9809

Iteration 2700, Loss: 0.0613, Accuracy: 0.9811

Iteration 2800, Loss: 0.0609, Accuracy: 0.9811

Iteration 2900, Loss: 0.0615, Accuracy: 0.9807

SGD with Nesterov Momentum Test Accuracy: 0.9800

=== Training Adam Optimizer ===

Iteration 100, Loss: 0.2219, Accuracy: 0.9034

Iteration 200, Loss: 0.1618, Accuracy: 0.9439

Iteration 300, Loss: 0.1318, Accuracy: 0.9571

Iteration 400, Loss: 0.1135, Accuracy: 0.9666

Iteration 500, Loss: 0.1015, Accuracy: 0.9702

Iteration 600, Loss: 0.0934, Accuracy: 0.9734

Iteration 700, Loss: 0.0876, Accuracy: 0.9741

Iteration 800, Loss: 0.0823, Accuracy: 0.9757

Iteration 900, Loss: 0.0780, Accuracy: 0.9782

Iteration 1000, Loss: 0.0763, Accuracy: 0.9795

Iteration 1100, Loss: 0.0720, Accuracy: 0.9796

Iteration 1200, Loss: 0.0703, Accuracy: 0.9790

Iteration 1300, Loss: 0.0688, Accuracy: 0.9790

Iteration 1400, Loss: 0.0666, Accuracy: 0.9809

Iteration 1500, Loss: 0.0654, Accuracy: 0.9814

Iteration 1600, Loss: 0.0638, Accuracy: 0.9816

Iteration 1700, Loss: 0.0629, Accuracy: 0.9811

Iteration 1800, Loss: 0.0619, Accuracy: 0.9816

Iteration 1900, Loss: 0.0613, Accuracy: 0.9813

Iteration 2000, Loss: 0.0603, Accuracy: 0.9815

Iteration 2100, Loss: 0.0594, Accuracy: 0.9820

Iteration 2200, Loss: 0.0588, Accuracy: 0.9822

Iteration 2300, Loss: 0.0582, Accuracy: 0.9823

Iteration 2400, Loss: 0.0577, Accuracy: 0.9826

Iteration 2500, Loss: 0.0573, Accuracy: 0.9825

Iteration 2600, Loss: 0.0567, Accuracy: 0.9825

Iteration 2700, Loss: 0.0563, Accuracy: 0.9826

Iteration 2800, Loss: 0.0578, Accuracy: 0.9823

Iteration 2900, Loss: 0.0572, Accuracy: 0.9825

Iteration 3000, Loss: 0.0554, Accuracy: 0.9825

Adam Test Accuracy: 0.9822

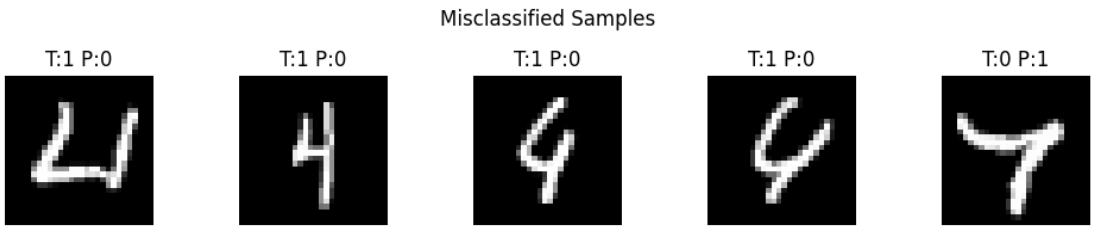
=== Comparison of Test Accuracies ===

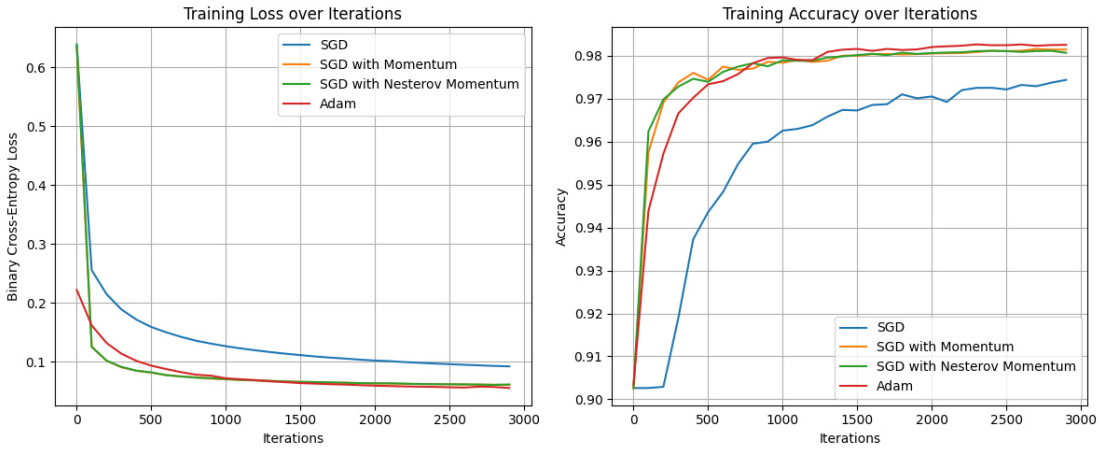
SGD: 0.9723

SGD with Momentum: 0.9802

SGD with Nesterov Momentum: 0.9800

Adam: 0.9822





1. Which optimizer gave you the best test accuracy? Why do you think it performed better than the others?

🏆 **Best Test Accuracy: Adam (0.9822)**

Adam performed **the best** among the four optimizers in terms of test accuracy.

📊 **Why Adam performed better:**

1. **Adaptive Learning Rate:**
    - Adam adjusts the learning rate for each parameter individually based on estimates of first and second moments of the gradients.
    - This helps it converge faster and more effectively, especially when gradients vary in scale across features.
2. **Fast Convergence:**
    - In your training curves, Adam reached high accuracy within the first few hundred iterations and then continued improving smoothly.
    - This is visible in the right-hand plot: Adam's curve rises rapidly and flattens out at the top.
3. **Stable and Low Training Loss:**
    - Adam also achieved the **lowest final training loss** among all optimizers, which suggests it fit the training data very well while generalizing well to the test set.

4. **Better Handling of Noisy Gradients:**
    - Adam tends to be more robust to noisy updates (common in mini-batch training), which helps achieve better generalization.

✅ **Summary of Test Accuracies:**

Optimizer	**Test Accuracy**
SGD	0.9723
SGD with Momentum	0.9802
SGD with Nesterov Momentum	0.9800
Adam	**0.9822**

2. What is the differences in learning stability, convergence speed, or misclassification types across all algorithm? Please explain with examples or observation from your results.

🧠 **1. Learning Stability**

🔹 **SGD:**

- **Observation:** Fluctuating accuracy in early iterations. The training accuracy rises slowly and continues to improve steadily.
- **Interpretation:** Classic SGD tends to be less stable because it doesn't account for momentum. It may oscillate or converge slowly, especially in valleys or plateaus of the loss surface.

Example:
SGD takes until **~2900 iterations** to reach ~0.974 accuracy. Training loss also decreases more slowly and remains higher than others (see the left plot).

🔹 **SGD with Momentum / Nesterov Momentum:**

- **Observation:** Much smoother curves than plain SGD. Accuracy improves rapidly and stabilizes after ~1000 iterations.
- **Interpretation:** Momentum reduces oscillation by smoothing the gradient updates, giving better stability in the direction of convergence.

Example:
At **iteration 100**, momentum-based SGD already hits **~0.9575–0.9624 accuracy**, compared to SGD's 0.9026. Later, the accuracy improves with smaller fluctuations.

🔹 **Adam:**

- **Observation:** Very stable training. The curve rises quickly and then levels off without much fluctuation.
- **Interpretation:** Adam uses **adaptive learning rates**, which help maintain stability and avoid overshooting minima.

Example:
Adam reaches **~0.978 accuracy by iteration 1000** and maintains it with very minor changes, with loss steadily decreasing to the lowest point.

⚡ **2. Convergence Speed**

Optimizer	**Accuracy @ Iteration 100**	**Accuracy @ 1000**	**Final Test Accuracy**
SGD	0.9026	0.9626	0.9723
SGD with Momentum	0.9575	0.9784	0.9802
SGD with Nesterov	0.9624	0.9789	0.9800
Adam	0.9034 (low start)	**0.9795**	**0.9822**

🔍 **Observations:**

- **Adam converges the fastest** after a slightly lower initial jump (due to adaptive step sizes).
- **Momentum methods (especially Nesterov)** are also fast, reaching 0.96+ by iteration 300–400.
- **SGD is slowest** to converge, requiring 2000+ iterations to match the performance others achieve in 500–800.

🧩 **Misclassification Pattern Analysis**

Each image shows:

- T:1 P:0 → **True label is '4'**, predicted as **not-4**
- T:0 P:1 → **True label is not '4'**, predicted as **4**

🔹 **1. SGD (1st row)**

- **All errors are T:1 P:0**, meaning the model failed to recognize digit '4'.
- Examples include:
  - A **curvy '4'** with a loop — may resemble a distorted '9'.
  - A **thin slanted '4'** — may resemble a '1' or stylized '7'.
  - **Unusual handwriting** styles with loose curvature or broken strokes.

📌 **Insight:**
SGD underfits in early stages and learns more slowly. It may not develop robust features to capture the variability in how '4' is written.

🔹 **2. SGD with Momentum (2nd row)**

- Mix of:
  - **T:1 P:0**: failed to detect odd or italicized 4s.
  - **T:0 P:1**: mistakenly classifies a **curved 7** as 4.

📌 **Insight:**
Momentum improves fitting, but still struggles with digits that share **similar strokes** — especially **4 vs 7**, which both often have vertical and horizontal segments.

🔹 **3. SGD with Nesterov Momentum (3rd row)**

- Very similar error types to Momentum.
- Misclassifies:
    - A 4 that looks like a mirrored or rounder variant.
    - A digit '7' (with long stroke) predicted as '4'.

📌 **Insight:**
Nesterov helps with smoother convergence, but misclassifications still happen with **ambiguous or slanted digits**.

🔹 **4. Adam (4th row)**

- More **false positives** (T:0 P:1) than the others.
- Misclassified:
    - One digit '9' (confused with 4 — similar lower loops).
    - Several very curly or unusually written 4s.

📌 **Insight:**
Adam fits the training data **more aggressively** and thus **risks overfitting** on some ambiguous features — leading to occasional **overconfident false positives**.

📊 **Summary of Differences Across Optimizers**

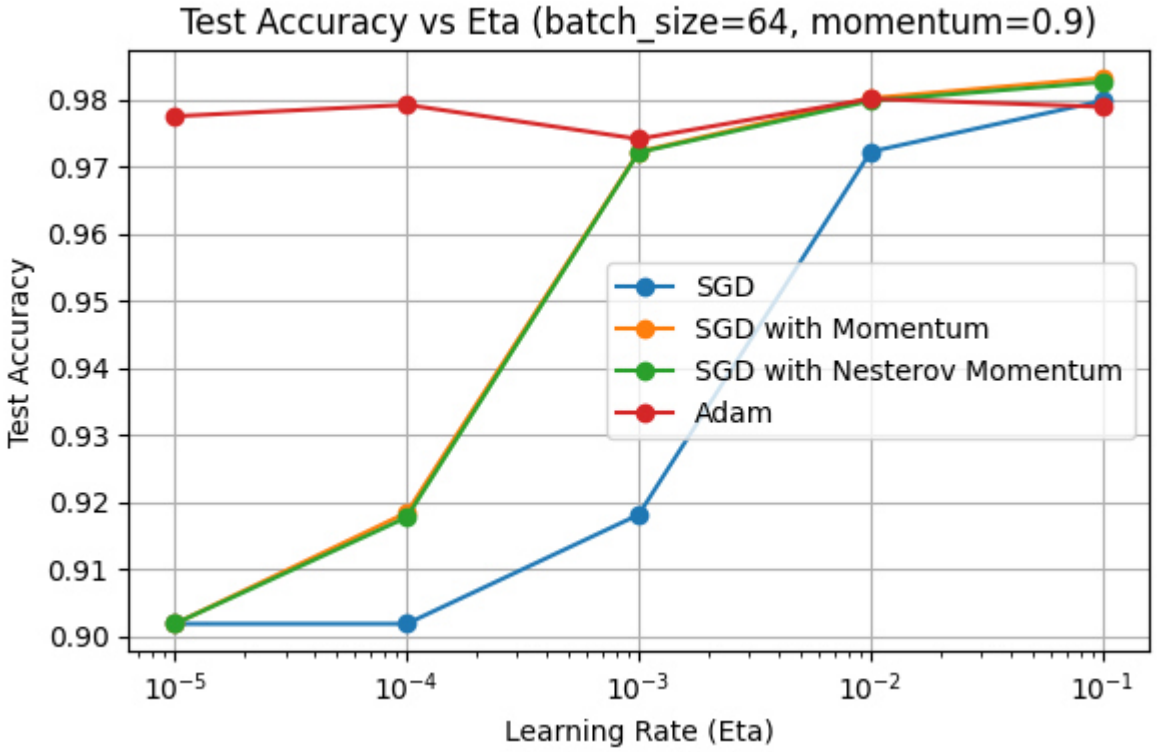
Optimizer	Main Errors	Cause	Strengths	Weaknesses
SGD	All T:1 P:0 (misses 4)	Underfitting, slow learning	Simple & stable	Weak at modeling handwriting variation
Momentum	T:1 P:0 and T:0 P:1	Still struggles with 4 vs 7 or italic fonts	Faster than SGD	Mild overfitting/misgeneralization
Nesterov	Similar to Momentum	Similar challenges with stroke ambiguity	Slightly smoother than Momentum	Same error types as Momentum
Adam	More T:0 P:1 (false 4s)	Stronger fitting, possibly overconfident predictions	Best overall accuracy & convergence	Risks mistaking noisy digits for 4s

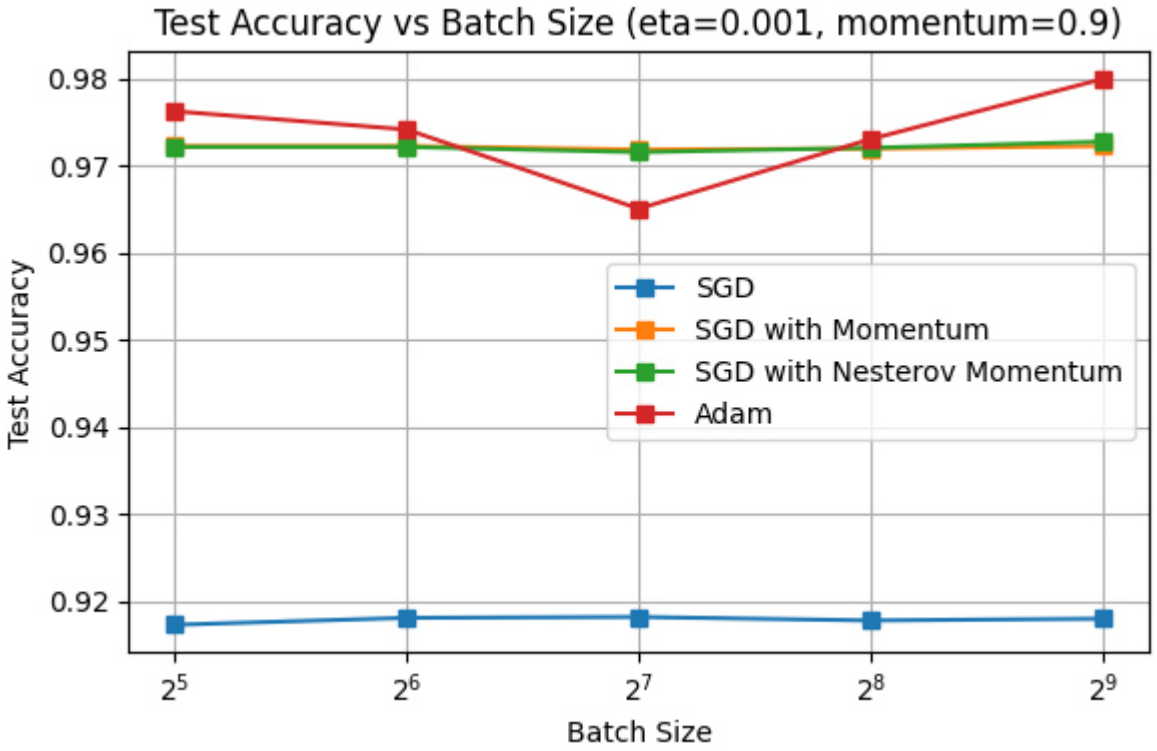
---

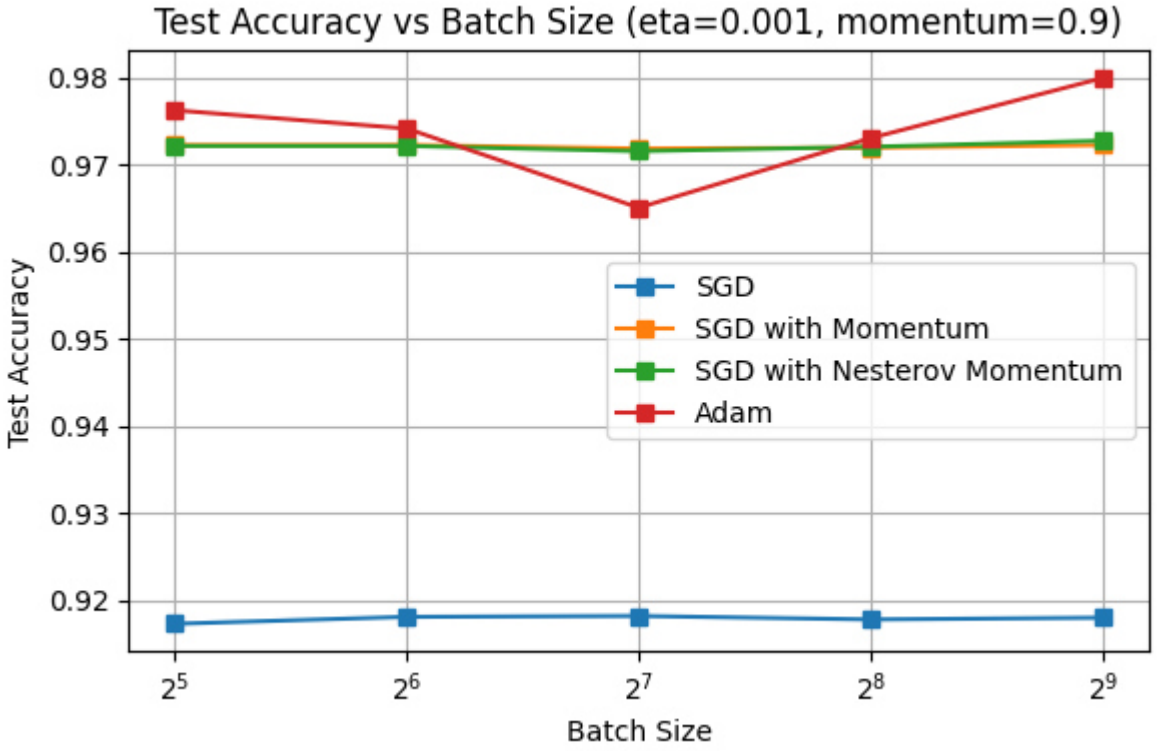
✅ **Final Thoughts**

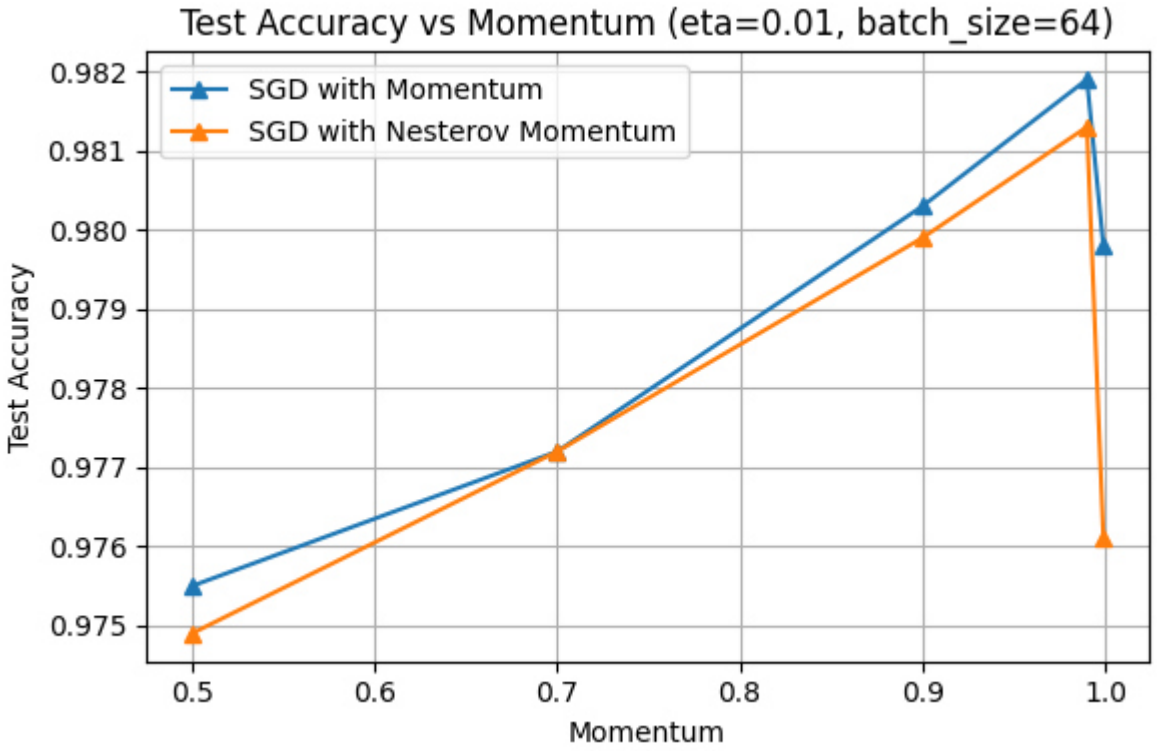
- **Adam learns fast**, but may **overfit** on ambiguous non-4 digits.
- **Momentum variants** are a nice balance — good convergence and moderate generalization.
- **Plain SGD** is the most cautious, but misses too many 4s early on.

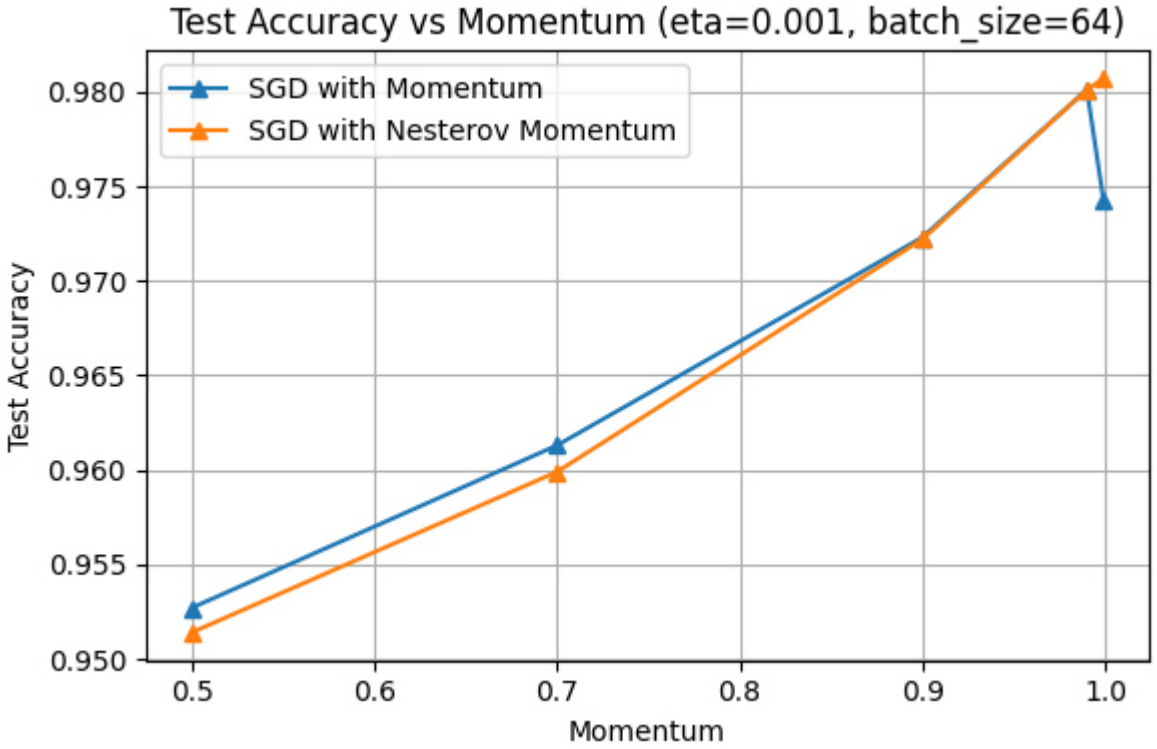
3. How did your choice of learning rate, batch size, or momentum affect each optimizer? What values worked best in your experiments?


Test Accuracy vs Eta (batch_size=64, momentum=0.9)
Test Accuracy
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
Learning Rate (Eta)
SGD
SGD with Momentum
SGD with Nesterov Momentum
Adam


Test Accuracy vs Batch Size (eta=0.001, momentum=0.9)
Test Accuracy
0.98
0.97
0.96
0.95
0.94
0.93
0.92
2^5
2^6
2^7
2^8
2^9
Batch Size
SGD
SGD with Momentum
SGD with Nesterov Momentum
Adam


Test Accuracy vs Batch Size (eta=0.001, momentum=0.9)
Test Accuracy
Batch Size
SGD
SGD with Momentum
SGD with Nesterov Momentum
Adam
0.98
0.97
0.96
0.95
0.94
0.93
0.92
2^5
2^6
2^7
2^8
2^9


Test Accuracy vs Momentum (eta=0.01, batch_size=64)
Test Accuracy
Momentum
SGD with Momentum
SGD with Nesterov Momentum
0.982
0.981
0.980
0.979
0.978
0.977
0.976
0.975
0.5
0.6
0.7
0.8
0.9
1.0



1. Learning Rate (Eta) Effects (First Plot)

- **SGD**:
    - Very sensitive to the learning rate.
    - Low accuracy at small learning rates (e.g., 10−510^{-5}10−5 to 10−310^{-3}10−3).
    - Accuracy improves dramatically with higher learning rates, peaking at η=10−1\eta = 10^{-1}η=10−1.
- **SGD with Momentum & Nesterov Momentum**:
    - Perform much better than plain SGD at low learning rates.
    - Accuracy increases consistently as learning rate increases.
    - Plateau around η=10−2\eta = 10^{-2}η=10−2 and 10−110^{-1}10−1 with ~0.982+ accuracy.
- **Adam**:
    - Very stable across all learning rates.

- Maintains high accuracy even at very low learning rates.
- Slightly outperformed by momentum methods at the highest learning rate.

📌 **Insight**: Adam is robust to learning rate choice, while SGD benefits greatly from momentum at lower learning rates.

---

2. Batch Size Effects (Second & Third Plot — Duplicates)

- **SGD**:
  - Performs consistently poorly regardless of batch size (~91.7–91.8%).
- **Momentum & Nesterov**:
  - High and stable accuracy (~97.2–97.3%) across all batch sizes.
  - Not much sensitivity to batch size changes.
- **Adam**:
  - Slightly more variation across batch sizes.
  - Smallest batch size (25=322^5 = 3225=32) and largest (29=5122^9 = 51229=512) give the highest accuracy (~97.7–97.9%).
  - Dip at medium batch size (27=1282^7 = 12827=128).

📌 **Insight**: Batch size has little effect on momentum-based SGD and minor nonlinear effect on Adam. SGD remains inferior.

---

3. Momentum Effects (Fourth and Fifth Plots)

At η=0.01\eta = 0.01η=0.01:

- **SGD with Momentum**:
  - Accuracy increases as momentum increases up to 0.99.
  - Slight dip at 1.0, possibly due to instability at max momentum.

- **Nesterov**:
    - Similar trend, slightly lower than classic momentum at each point.

At η=0.001\eta = 0.001η=0.001:

- Both optimizers show increasing accuracy with momentum up to 0.99.
- Peak at 0.99, followed by a performance drop at 1.0 (likely due to overshooting).

📌 **Insight**: Momentum improves accuracy significantly, but extremely high momentum (1.0) can harm performance.

Summary of Effects per Optimizer

Optimizer	Learning Rate	Batch Size	Momentum
SGD	Highly sensitive	Consistently low	Not applicable
SGD + Momentum	Improves at all rates	Very stable, high accuracy	Boosts performance steadily
SGD + Nesterov	Similar to Momentum	Very stable	Slightly lower than Momentum
Adam	Very robust	Slight dip at mid-batch	Not applicable