Machine Learning
LABORATORY: Backpropagation In Class

NAME:	STUDENT ID#:

Objectives:

- Understand the **core concept of backpropagation** as used in training neural networks.
- Simulate and visualize **forward-mode and reverse-mode automatic differentiation** to trace how gradients are propagated.
- Interpret how gradient values are calculated during backprop through a computational graph.

Part 1. Instruction

- In this assignment, please **train a logistic classifier** to recognize whether an MNIST digit image is the target digit (e.g., "Is it a 3?") or not. ***(Last week)***
- You will integrate a backpropagation autodiff mechanism into the SGD training loop to compute gradients used for weight updates.
- Integrate an **autodiff module** that traces:
  - **Primal values** through the forward pass (e.g., intermediate variables like $v_3 = x_1 x_2$).
  - **Forward-mod**e using the chain rule from inputs to output.
  - **Reverse-mode** representing the backpropagation path from output to inputs.
- You will complete the code template provided in the in-class assignment.
- Use only NumPy for all computations. Do not use libraries like scikit-learn or PyTorch.
- Evaluate your results and answer the questions.

Part 2. Code Template

Step 1:

```
# ========== Load Dataset ==========
def load_images(filename):
    with open(filename, 'rb') as f:
        _, num, rows, cols = struct.unpack(">IIII", f.read(16))
       data = np.frombuffer(f.read(), dtype=np.uint8).reshape((num,
rows * cols))
        return data.astype(np.float32) / 255.0

def load_labels(filename):
    with open(filename, 'rb') as f:
        _, num = struct.unpack(">II", f.read(8))
        return np.frombuffer(f.read(), dtype=np.uint8)
```

Step 2:

```
# ========== 1. Sigmoid Function ==========
def sigmoid(z):
    # TODO: Implement sigmoid function (optional)
```

```
    pass

def sigmoid_derivative(z):
    pass

TODO: Implement Backprop Autodiff
# ====== 3. Forward and Reverse Autodiff Trace ========
def trace_autodiff_example(x1, x2):
    # Primal

    # Forward tangent

    # Reverse adjoint

    return table

TODO: Implement SGD (use your codes last week), then use the backprop
inside
# ========== 2. SGD: Algorithm 7.1 ==========
def your_sgd_logistic(X, y, eta, max_iters):

      for i in range(max_iters):

        if i == 0:
            trace = trace_autodiff_example(_,_)
    return w, trace
```

3

```
# ==========Show Misclassified Samples ==========
def show_misclassified(X, y_true, y_pred, max_show=10):
    mis_idx = np.where(y_true != y_pred)[0][:max_show]
    if len(mis_idx) == 0:
        print("No misclassifications!")
        return
    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:{y_true[idx]}\nP:{y_pred[idx]}")
    plt.suptitle("Misclassified Samples")
    plt.show()
```

```
# ========== Plot Trace Graph ==========
def plot_autodiff_traces(trace_df):
    variables = trace_df['Variable']
    primal = trace_df['Primal (v)'].astype(float)
    forward = pd.to_numeric(trace_df['Forward Tangent (ẋ)'],
errors='coerce')
    reverse = pd.to_numeric(trace_df['Reverse Adjoint (v̄)'],
```

```python
errors='coerce')

    fig, ax = plt.subplots(3, 1, figsize=(10, 8), sharex=True)

    ax[0].bar(variables, primal, color='skyblue')
    ax[0].set_ylabel("Primal (v)")
    ax[0].set_title("Primal Values")

    ax[1].bar(variables, forward, color='lightgreen')
    ax[1].set_ylabel("Forward Tangent (ẋ)")
    ax[1].set_title("Forward-Mode Autodiff")

    ax[2].bar(variables, reverse, color='salmon')
    ax[2].set_ylabel("Reverse Adjoint (v̄)")
    ax[2].set_title("Reverse-Mode Autodiff")
    ax[2].set_xlabel("Variables")

    plt.tight_layout()
    plt.show()
```

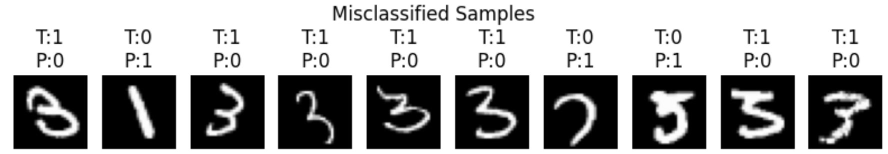
4

```python
# ========== 3. Main ==========
def main():
    # === Load MNIST 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__")

    # === Binary Classification ===
    TARGET_DIGIT = 3 # TODO: Fill in (0 to 9) based on your student number
    y_train_bin = np.where(y_train == TARGET_DIGIT, 1, 0)
    y_test_bin = np.where(y_test == TARGET_DIGIT, 1, 0)

    # === Add Bias ===
    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])

    # === Train ===
    # w, autodiff_trace =

    # === Predict ===
    # pred_probs =
    # preds =

    # === Accuracy ===
    # acc = np.mean(preds == y_test_bin)
    # print(f"\nTest Accuracy (is {TARGET_DIGIT} or not): {acc:.4f}")
```

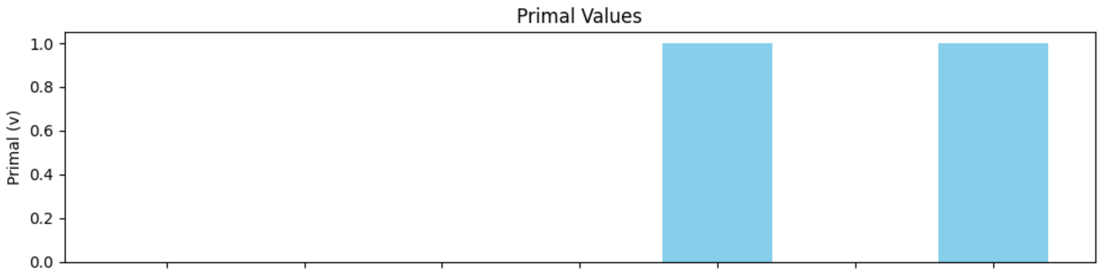
```python
    # === Show Misclassifications ===
    # show_misclassified(X_test, y_test_bin, preds)

    # === Visualize Autodiff Trace ===
    # print("\nAutodiff Trace Table (sample features):")
    # print(autodiff_trace)
    # plot_autodiff_traces(autodiff_trace)

if __name__ == "__main__":
    main()
```

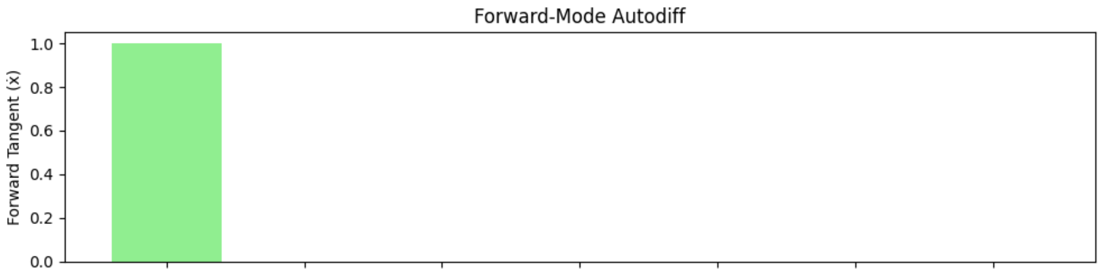
5

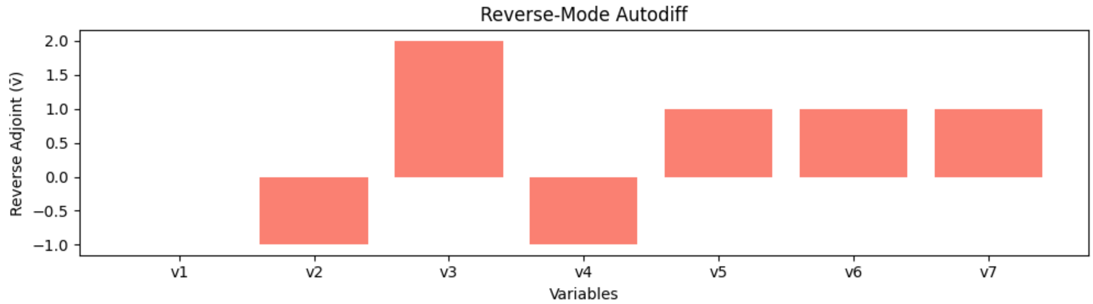
```
#Example Output:
Test Accuracy (is 3 or not): 0.9774
```




```
Autodiff Trace Table (sample features):
  Variable  Primal (v)  Forward Tangent (ẋ)  Reverse Adjoint (v̄)
0       v1         0.0                  1.0                  0.0
1       v2         0.0                  0.0                 -1.0
2       v3         0.0                  0.0                  2.0
3       v4         0.0                  0.0                 -1.0
4       v5         1.0                  0.0                  1.0
5       v6         0.0                  0.0                  1.0
6       v7         1.0                  0.0                  1.0
```







Grading Assignment & Submission (30% Max)

Implementation:

1. (10%) Implement the backpropagation autodiff
2. (10%) The model runs successfully without errors, use the provided MNIST dataset, and output the primal values, forward and reverse mode autodiff
3. (5%) Set the class of binary classification to the last digit of your student ID. (e.g., if your ID ends in 7, use the class '7'). Displays the result as shown as the "Example Output" in the last pages of this document.
4. (5%) Briefly discuss your results. For example, explain what the graph represents and why you obtained those results.

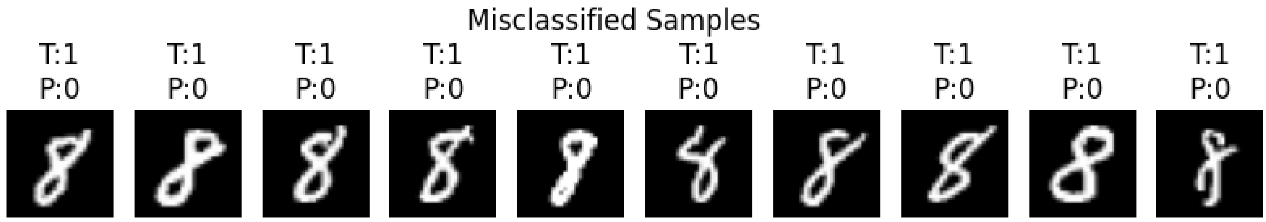
Submission:

1. Report: Provide your screenshots of your results including the discussion 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 (**Labs5 In Class Assignment**). Name your files correctly:
    a. Report: StudentID_Lab5_InClass.pdf
    b. Code: StudentID_Lab5_ InClass.py or StudentID_Lab5_InClass.ipynb
4. Deadline: 16:20 PM
5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

Results and Discussion:

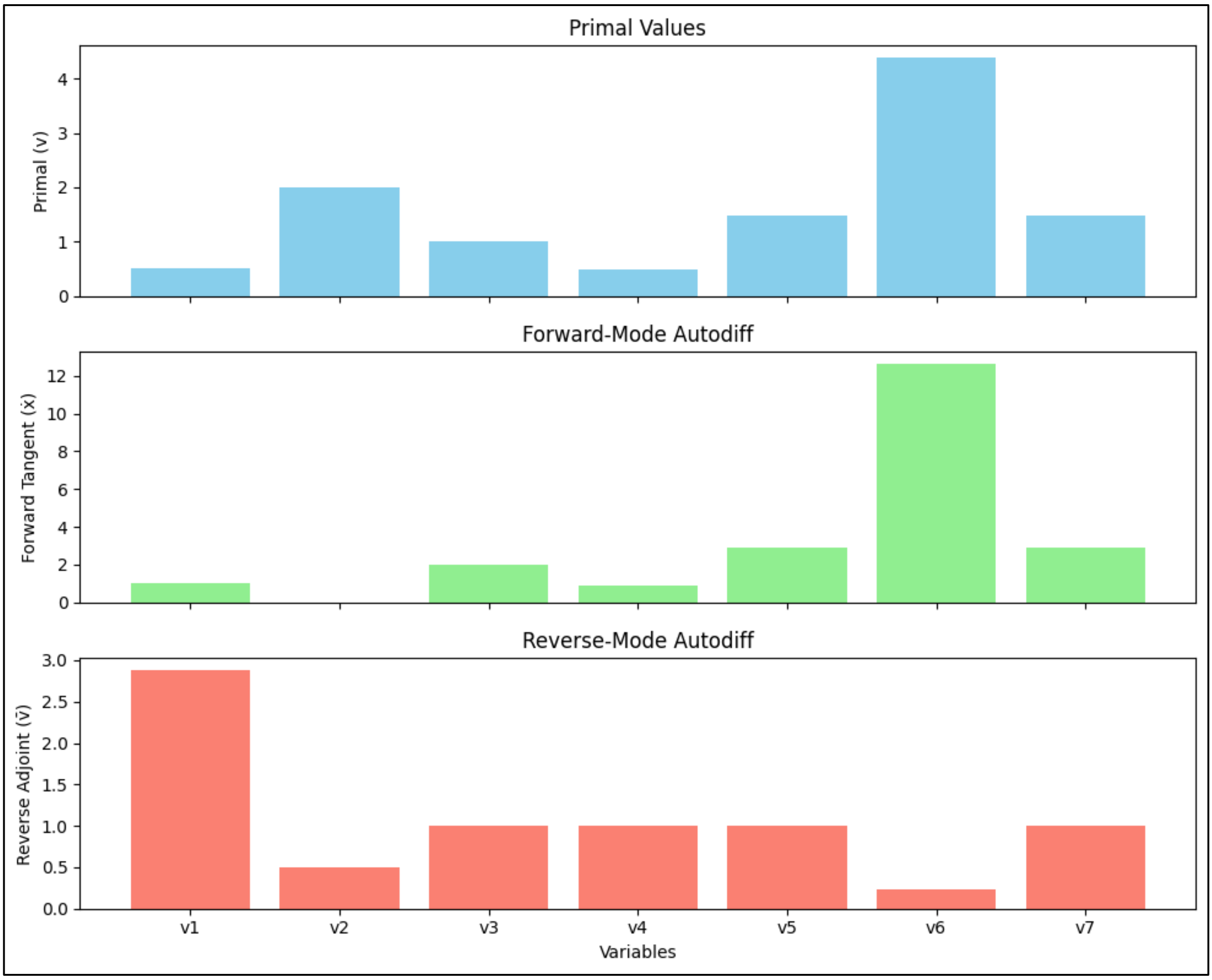
Put Your Code Results Here:

Test Accuracy (is 8 or not): 0.9269



Autodiff Trace Table (sample features):

	Variable	Primal (v)	Forward Tangent ($\dot{x}$)	Reverse Adjoint ($\bar{v}$)
0	v1	0.500000	1.000000	2.877583
1	v2	2.000000	0.000000	0.500000
2	v3	1.000000	2.000000	1.000000
3	v4	0.479426	0.877583	1.000000
4	v5	1.479426	2.877583	1.000000
5	v6	4.390423	12.633804	0.227768
6	v7	1.479426	2.877583	1.000000



What do these graphs mean?

These three graphs show how numbers change when we do some math steps, from the input all the way to the final output.

- The first blue graph shows the normal values we get from each step. For example, we multiply, take sine, add, etc., and get values like v1 to v7.
- The second green graph shows da how much the output will change if we slightly change the input x1. This is called forward-mode autodiff.
- The third graph red one. Which shows how much each value (v1 to v7) affects the final output. This is called reverse-mode autodiff, and it's how deep learning learns by going backward.

Why do the forward-mode and reverse-mode values look like that?

The forward-mode values start from x1 and move forward through the steps. So, if one step has a big function like exp() or log(), it can make the numbers grow bigger.

The reverse-mode values go backward. So, the last number (v7) gives its effect to the steps before it. If one variable is used in many steps (like v1), it gets more "blame" or "credit", so its number gets bigger.

What did you learn from this assignment?

I learned how computers figure out how much each number matters when making a prediction. It's like when we do math step-by-step and also check how changes affect the result.

I think I understand how backpropagation works — it's like going backward through the math to find what caused the result. But I still have to read teacher PPT again when I back to home.
Also, I learned the difference between forward type (goes from input to output) and reverse type (goes from output back to input).