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-mode** 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	Procedure

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)`

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

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

```
    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 (v̇)'],
errors='coerce')
    reverse = pd.to_numeric(trace_df['Reverse Adjoint (v̄)'],
```


```
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 (v̇)")
    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()
```

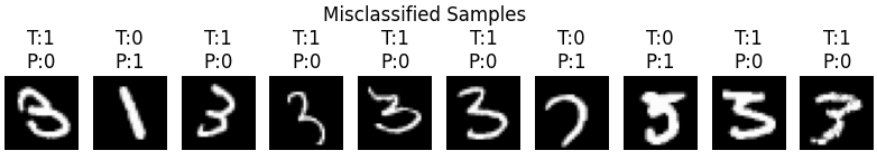
4

```
# ========== 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}")
```

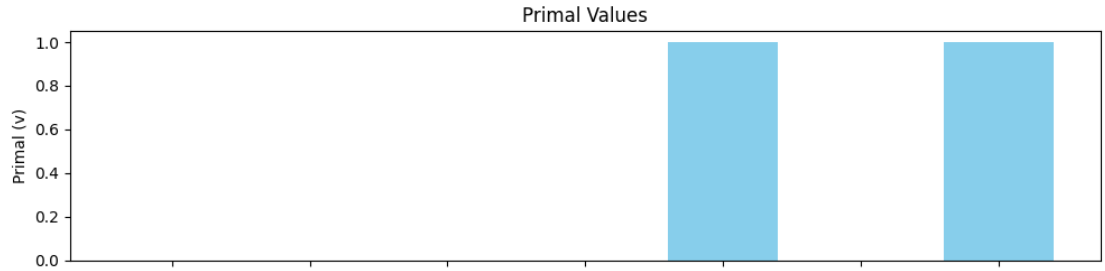

```
    # === Show Misclassifications ===
    # show_misclassified(X_test, y_test_bin, preds)

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

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

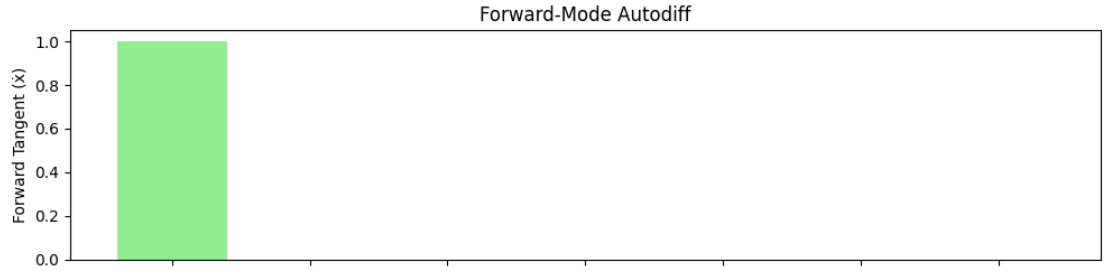
5

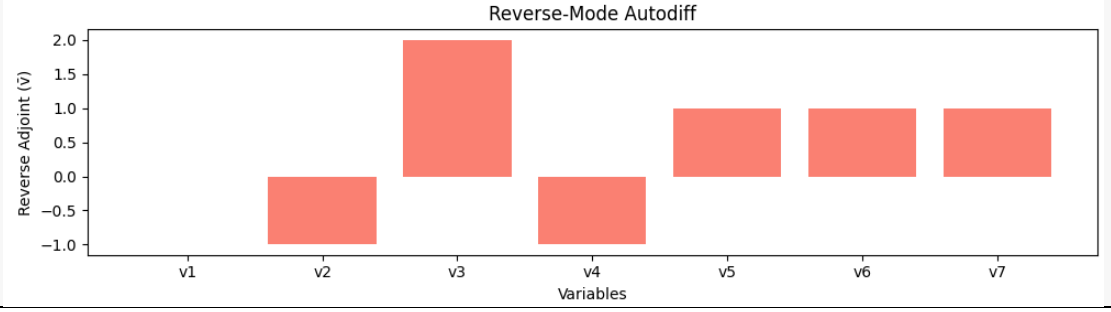
```
#Example Output:
Test Accuracy (is 3 or not): 0.9774
```




```
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:

Loading MNIST data...

Creating binary classification task for digit 4...

Training logistic regression classifier...

Iteration 0, Training Error: 0.2460

Iteration 10, Training Error: 0.0974

Iteration 20, Training Error: 0.0974

Iteration 30, Training Error: 0.0974

Iteration 40, Training Error: 0.0974

Iteration 50, Training Error: 0.0974

Iteration 60, Training Error: 0.0974

Iteration 70, Training Error: 0.0974

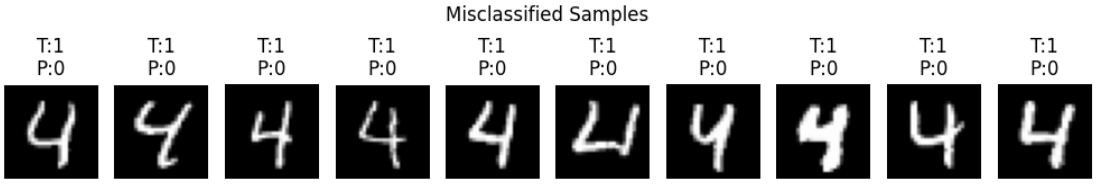
Iteration 80, Training Error: 0.0974

Iteration 90, Training Error: 0.0974

Making predictions...

Test Accuracy (is 4 or not): 0.9018

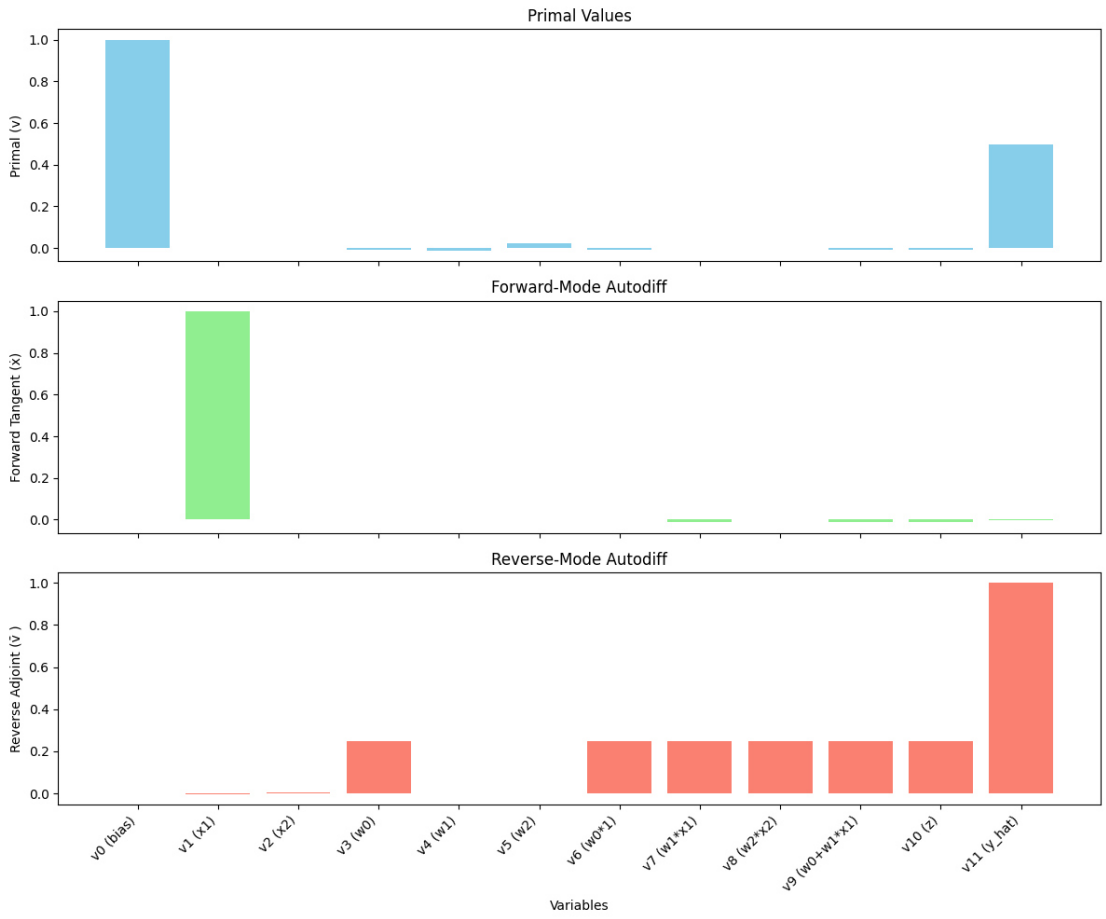
Showing some misclassified examples...



Autodiff Trace Table (Sample Features)

Index	Variable	Primal (v)	Forward Tangent ($\dot{x}$)	Reverse Adjoint ($\bar{v}$)
0	v0 (bias)	1.000000	0.000000	-0.002284
1	v1 (x1)	0.000000	1.000000	-0.001595

Index	Variable	Primal (v)	Forward Tangent ($\dot{x}$)	Reverse Adjoint ($\bar{v}$)
2	v2 (x2)	0.000000	0.000000	0.003436
3	v3 (w0)	-0.009138	0.000000	0.249995
4	v4 (w1)	-0.006378	0.000000	0.000000
5	v5 (w2)	0.013746	0.000000	0.000000
6	v6 (w0 * 1)	-0.009138	0.000000	0.249995
7	v7 (w1 * x1)	-0.000000	-0.006378	0.249995
8	v8 (w2 * x2)	0.000000	0.000000	0.249995
9	v9 (w0 + w1*x1)	-0.009138	-0.006378	0.249995
10	v10 (z)	-0.009138	-0.006378	0.249995
11	v11 (y_hat)	0.497716	-0.001595	1.000000



Misclassified Samples Analysis

The first image shows misclassified digit samples. We can see that all examples are digit "4" that were incorrectly classified as "0" (T:1, P:0). This indicates that the model had difficulty specifically recognizing certain styles of the digit 4. Looking at the misclassified examples, they all appear to be variations of the digit 4, some with more open loops and others with different slants. The model seems to struggle with the variability in how people write the digit 4.

Autodiff Trace Analysis

The three charts display the computational trace of automatic differentiation for a sample data point:

1. Primal Values Chart

- This shows the actual values calculated during the forward pass
- The highest values are at v0 (bias = 1.0) and v11 (y_hat ≈ 0.5)

- Notice that v1 (x1) and v2 (x2) are both 0, indicating this particular sample had zero values for the selected features
- The final output v11 is around 0.498, which is very close to 0.5, suggesting this sample was near the decision boundary

2. Forward-Mode Autodiff Chart

- This represents the partial derivatives with respect to input x1
- The large value at v1 (x1) is because we seeded dx1/dx1 = 1.0
- The small values in the other variables show how a change in x1 would propagate through the computation
- The small value at v11 indicates that a change in x1 would have minimal effect on the output for this sample

3. Reverse-Mode Autodiff Chart

- This shows the gradients flowing backward from the output
- v11 (y_hat) has the highest value (1.0) because we start backpropagation from here
- The gradients flow backward through the network with significant values at v6-v10, showing how the error propagates back
- v3 (w0) has a moderate gradient value, indicating the bias weight needs adjustment
- The very small values at v1 and v2 suggest that for this particular sample, the input features contribute little to the gradient

Training Results Interpretation

The training error stabilized quickly at around 0.097 after just 10 iterations and didn't improve further, suggesting the model reached a local optimum. The final test accuracy of 90.18% is reasonable for a simple logistic regression model on the binary classification task (is digit 4 or not).

The table shows how values and gradients flow through the computation graph.

Notice how in reverse mode (backpropagation), the gradients flow from the output (v11) back to the inputs and weights, which is precisely how neural networks learn through gradient descent. The gradient values for weights (v3, v4, v5) inform the weight updates during training.

The stabilization of training error might indicate that logistic regression has reached its modeling capacity for this problem. A more complex model like a neural network might achieve better performance by capturing more intricate patterns in the digits.