Coursework 2: Logistic Regression

Multi-Class Classification with Multiple Loss Functions

Course: CS3317: Artificial Intelligence
University: Shanghai Jiao Tong University

Overview

You will implement a **multi-class logistic regression** classifier from scratch using NumPy on the **FashionMNIST** dataset. You will implement four different loss functions and compare their effects on classification performance.

- **Dataset:** FashionMNIST — 10 classes of 28×28 grayscale clothing images, flattened to 784-dimensional vectors.
- **Framework:** NumPy only (no PyTorch for this coursework).
- **Model:** Single linear layer followed by softmax: $\mathbf{p} = \text{softmax}(\mathbf{XW} + \mathbf{b})$.

Learning Objectives

By the end of this coursework you should be able to:

1. Implement a numerically stable softmax function.
2. Implement four classification loss functions and derive their gradients analytically.
3. Implement mini-batch gradient descent to train a linear classifier.
4. Compare the convergence, stability, and accuracy of different loss functions.
5. Visualize learned model weights and interpret the confusion matrix.

Setup

```
cd code

# Install dependencies
pip install -r requirements.txt

# Download and prepare the FashionMNIST dataset
python setup_data.py

cd cw2_logistic_regression
```

The dataset will be saved as `.npy` files under `code/data/fashionmnist/`.

What You Need to Implement

All `TODO` locations are clearly marked in the code with hints. You need to implement the following:

File	Function / Method	Description
`losses.py`	`softmax(logits)`	Numerically stable softmax
`losses.py`	`cross_entropy_loss(logits, labels, C)`	Loss + gradient
`losses.py`	`hinge_loss(logits, labels, C)`	Loss + gradient
`losses.py`	`exponential_loss(logits, labels, C)`	Loss + gradient
`losses.py`	`squared_loss(logits, labels, C)`	Loss + gradient
`model.py`	`LogisticRegression.softmax(logits)`	Softmax (same as above)
`model.py`	`LogisticRegression.forward(X)`	Compute logits: $\mathbf{XW} + \mathbf{b}$
`model.py`	`LogisticRegression.predict(X)`	Predicted class indices
`optimizer.py`	`GradientDescent.__init__(lr)`	Store learning rate
`optimizer.py`	`GradientDescent.step(model, grad, X)`	Compute & apply updates

Do not modify the following provided files: `run.py`, `config.py`, `trainer.py`.

Tasks

Task 1 — Implement Softmax

Implement the softmax function in both `losses.py` and `model.py`:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{1}$$

Important: Numerical Stability

Directly computing e^{z_i} can overflow when logit values are large. You must subtract the maximum logit from each row before computing the exponential. This produces mathematically equivalent results while preventing overflow.

Task 2 — Implement Loss Functions

Implement four loss functions in `losses.py`. Each function receives logits, labels, and the number of classes as input, and returns a tuple `(loss, grad)`, where `grad` is the gradient of the loss w.r.t. logits (shape: batch_size × num_classes).

Cross-Entropy Loss

$$L = -\frac{1}{N}\sum_{i=1}^{N} \log p_{i,y_i} \tag{2}$$

where p_{i,y_i} is the predicted probability of the correct class for sample i.

Hinge Loss (Crammer–Singer)

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i} \max(0,\ 1 + z_j - z_{y_i}) \tag{3}$$

Exponential Loss

$$L = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{p_{i,y_i}} \tag{4}$$

Squared Loss

$$L = \frac{1}{2N} \sum_{i=1}^{N} \|\mathbf{p}_i - \mathbf{y}_i^{\text{one-hot}}\|^2 \tag{5}$$

For each loss, you must also derive and implement the **analytic gradient** with respect to the logits. Hints are provided in the code comments.

Task 3 — Implement Gradient Descent

Implement the optimizer in `optimizer.py`. The `step()` method receives the gradient of the loss with respect to logits (from Task 2) and the input batch $\mathbf{X}$, and must:

1. Compute parameter gradients:

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^\top \frac{\partial L}{\partial \mathbf{z}}, \qquad \frac{\partial L}{\partial \mathbf{b}} = \sum_i \frac{\partial L}{\partial \mathbf{z}_i} \tag{6}$$

2. Update parameters: $\mathbf{W} \leftarrow \mathbf{W} - \eta \cdot \frac{\partial L}{\partial \mathbf{W}}$, $\mathbf{b} \leftarrow \mathbf{b} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}}$

Looking Ahead to CW3

The computation $\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^\top \nabla_{\mathbf{z}} L$ is actually the **backward pass** of a single linear layer. In CW3, you will generalize this idea to chain multiple layers together via backpropagation.

Task 4 — Implement the Model

Implement `forward()` and `predict()` in `model.py`:

- `forward(X)`: Compute logits $\mathbf{z} = \mathbf{X}\mathbf{W} + \mathbf{b}$.
- `predict(X)`: Return the predicted class index for each sample (call `forward()`, then argmax).

Task 5 — Train, Compare, and Analyze

1. Train with each loss function individually to verify correctness.
2. Run the comparison across all four losses.
3. Analyze the results: convergence speed, final accuracy, training stability.
4. Examine the weight visualization and confusion matrix.

How to Run

```
# Quick mode (for debugging --- uses smaller dataset, fewer epochs)
python run.py --quick

# Train with a specific loss function
python run.py --loss_type cross_entropy
python run.py --loss_type hinge
python run.py --loss_type exponential
python run.py --loss_type squared

# Compare all 4 loss functions (generates comparison plot)
python run.py --compare_all
```

The `-quick` flag uses a 10K-sample subset and fewer epochs for fast debugging. Use the full dataset for your final experiments.

How to Verify Your Implementation

Gradient Checking

Run the provided gradient checking tests:

```
# From the code/ directory
python -m tests.test_cw2
```

This test uses **numerical finite differences** to verify that your analytic gradients are correct. All gradient checks should pass with relative error $< 10^{-5}$. It also tests that your softmax produces valid probabilities (positive, sums to 1, no NaN/Inf).

Training Convergence

After training with cross-entropy loss using default hyperparameters (`python run.py`), you should observe reasonable classification accuracy on the test set.

Outputs

After training, the following files are generated in the `outputs/` directory:

- `training_summary.png` — Training loss and validation accuracy curves (side-by-side).
- `confusion_matrix.png` — Confusion matrix on the test set.
- `sample_predictions.png` — Sample images with predicted and true labels.
- `weight_visualization.png` — Learned 28×28 weight templates for each class.
- `results.json` — Numerical results (accuracy, loss history, etc.).

When using `-compare_all`, an additional `loss_comparison.png` is generated showing all four losses side by side.

Deliverables

Submit a single **zip file** with the following structure:

```
cw2_logistic_regression/   # The entire code directory
  losses.py
  model.py
  optimizer.py
  run.py
  config.py
  trainer.py
  outputs/                 # Your training outputs
  ...
report.pdf                 # Your report
```

Report (use the template in `report_template/`, see example report in `report_examples/`):

- Implementation overview with key equations.

- Loss function comparison with training curves (`-compare_all` output).
- Results table and analysis: convergence speed, final accuracy, stability.
- Weight visualization and confusion matrix discussion.

Grading Rubric (100 points)

Component	Points
Softmax implementation (numerically stable, correct shape, sums to 1)	10
Cross-entropy loss + gradient (passes gradient check)	15
Hinge loss + gradient (correct Crammer–Singer formulation)	15
Exponential loss + gradient (correct, numerically stable)	10
Squared loss + gradient (correct formulation)	10
Gradient descent optimizer (correct update with mini-batches)	10
Model forward + predict (correct logits and argmax)	5
Training convergence (cross-entropy achieves reasonable accuracy)	10
Report: loss comparison with curves and analysis	10
Code quality (clean, commented, follows structure)	5
Total	**100**
Bonus (capped at 100 total): L2 regularization	+5
Bonus (capped at 100 total): Learning rate schedule	+5

Note: The final grade including bonus will not exceed 100 points.

Project Structure

```
cw2_logistic_regression/
  run.py              # Entry point (provided)
  config.py           # Hyperparameter defaults (provided)
  trainer.py          # Training loop (provided)
  model.py            # (*) Logistic regression model
  losses.py           # (*) Four loss functions + softmax
  optimizer.py        # (*) Gradient descent

common/               # Shared utilities (provided)
  data_loader.py      #   Data loading, batching, train/val split
  metrics.py          #   Accuracy, confusion matrix
  visualization.py    #   Plotting functions
  utils.py            #   Seed setting, timer, save/load

tests/
  test_cw2.py         # Gradient checking tests

(*) = files you need to implement
```

Estimated Running Times (CPU)

Task	Quick Mode	Full Run
Single loss function	< 1 min	$\sim$5 min
Compare all 4 losses	$\sim$2 min	$\sim$20 min

Academic Integrity

- You may discuss high-level ideas with other students.
- You must write your own code.
- Do not copy implementations from other students or online sources.