Problem Set 2: Linear Classifiers and Gradient Descent

Po Yu Chen

October 14, 2024

1 Introduction

The objective of this assignment is to implement a linear classifier using Support Vector Machine (SVM) with hinge loss and a Softmax classifier with cross-entropy loss. We calculate the score function, apply L2 regularization to reduce overfitting, and use gradient descent to optimize the loss. The classifier is tested on a spiral dataset, which highlights the challenges of linear models in complex classification tasks.

2 Python Code

Below is the Python code for the assignment:

2.1 Task 1: Score Function

```python
def score_function(X, W, b):
    scores = np.dot(X, W) + b
    return scores
```

2.2 Task 2: Loss Function

```python
def svm_hinge_loss(X, y, W, b, delta=1.0):
    scores = score_function(X, W, b)
    N = X.shape[0]
    correct_class_scores = scores[np.arange(N), y]
    margins = np.maximum(0, scores - correct_class_scores[:, np.newaxis] +
                         delta)
    margins[np.arange(N), y] = 0
    loss = np.sum(margins) / N

    return loss
```

```
def softmax_cross_entropy_loss(X, y, W, b):
    scores = score_function(X, W, b)
    scores -= np.max(scores, axis=1, keepdims=True)
    exp_scores = np.exp(scores)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    N = X.shape[0]
    correct_logprobs = -np.log(probs[np.arange(N), y])
    loss = np.sum(correct_logprobs) / N

    return loss
```

2.3 Task 3: Regularization

```
def add_l2_regularization(W, reg=0.01):
    return reg * np.sum(W * W)

def svm_hinge_loss(X, y, W, b, delta=1.0, reg=0.01):
    scores = score_function(X, W, b)
    N = X.shape[0]
    correct_class_scores = scores[np.arange(N), y]
    margins = np.maximum(0, scores - correct_class_scores[:, np.newaxis] +
                         delta)
    margins[np.arange(N), y] = 0
    loss = np.sum(margins) / N
    loss += add_l2_regularization(W, reg)

    return loss

def softmax_cross_entropy_loss(X, y, W, b, reg=0.01):
    scores = score_function(X, W, b)
    scores -= np.max(scores, axis=1, keepdims=True)
    exp_scores = np.exp(scores)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    N = X.shape[0]
    correct_logprobs = -np.log(probs[np.arange(N), y])
    loss = np.sum(correct_logprobs) / N
    loss += add_l2_regularization(W, reg)
    return loss
```

2.4 Task 4: Gradient Descent

```
def svm_hinge_loss_gradient(X, y, W, b, delta=1.0, reg=0.01):
    N, D = X.shape
    K = W.shape[1]
    scores = score_function(X, W, b)
    correct_class_scores = scores[np.arange(N), y][:, np.newaxis]
    margins = scores - correct_class_scores + delta
    margins[np.arange(N), y] = 0
    binary = margins > 0
    binary = binary.astype(float)
    row_sum = np.sum(binary, axis=1)
    binary[np.arange(N), y] = -row_sum
    dW = np.dot(X.T, binary) / N + 2 * reg * W
    db = np.sum(binary, axis=0) / N
    return dW, db

def softmax_cross_entropy_loss_gradient(X, y, W, b, reg=0.01):
    N, D = X.shape
    scores = score_function(X, W, b)
    scores -= np.max(scores, axis=1, keepdims=True)
    exp_scores = np.exp(scores)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    dscores = probs
    dscores[np.arange(N), y] -= 1
    dscores /= N
    dW = np.dot(X.T, dscores) + 2 * reg * W
    db = np.sum(dscores, axis=0)
    return dW, db

num_iterations = 200
learning_rate = 1.0

D = X.shape[1]
K = np.max(y) + 1
W_svm = 0.001 * np.random.randn(D, K)
b_svm = np.zeros(K)
W_softmax = 0.001 * np.random.randn(D, K)
b_softmax = np.zeros(K)

svm_losses = []
softmax_losses = []

print("Training SVM Classifier:")
for i in range(num_iterations):
    loss = svm_hinge_loss(X, y, W_svm, b_svm)
    svm_losses.append(loss)
    dW, db = svm_hinge_loss_gradient(X, y, W_svm, b_svm)
    W_svm -= learning_rate * dW
    b_svm -= learning_rate * db
    if (i+1) % 10 == 0 or i == 0:
        print(f"Iteration {i+1}/{num_iterations}, Loss: {loss:.4f}")

print("\nTraining Softmax Classifier:")
for i in range(num_iterations):
    loss = softmax_cross_entropy_loss(X, y, W_softmax, b_softmax)
    softmax_losses.append(loss)
    dW, db = softmax_cross_entropy_loss_gradient(X, y, W_softmax, b_softmax)
    W_softmax -= learning_rate * dW
    b_softmax -= learning_rate * d
    if (i+1) % 10 == 0 or i == 0:
        print(f"Iteration {i+1}/{num_iterations}, Loss: {loss:.4f}")

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(range(1, num_iterations+1), svm_losses, label='SVM Loss')
plt.scatter(range(1, num_iterations+1, 10),
            [svm_losses[i-1] for i in range(1, num_iterations+1, 10)]
            , color='red')
for i in range(0, num_iterations, 10):
    plt.text(i+1, svm_losses[i], f"{svm_losses[i]:.2f}", fontsize=8,
             verticalalignment='bottom')
plt.title('SVM Hinge Loss over Iterations')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(range(1, num_iterations+1), softmax_losses,
         label='Softmax Loss', color='orange')
plt.scatter(range(1, num_iterations+1, 10),
            [softmax_losses[i-1] for i in range(1, num_iterations+1, 10)],
            color='red')
for i in range(0, num_iterations, 10):
    plt.text(i+1, softmax_losses[i], f"{softmax_losses[i]:.2f}",
             fontsize=8, verticalalignment='bottom')
plt.title('Softmax Cross-Entropy Loss over Iterations')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.grid(True)

plt.tight_layout()
plt.show()

def plot_decision_boundary(X, y, W, b, title):
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    h = 0.01

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    grid = np.c_[xx.ravel(), yy.ravel()]
    Z = score_function(grid, W, b)
    Z = np.argmax(Z, axis=1)
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(6, 5))
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.6)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40,
                cmap=plt.cm.Spectral, edgecolors='k')
    plt.title(title)
    plt.show()

plot_decision_boundary(X, y, W_svm, b_svm, 'SVM Decision Boundary')
plot_decision_boundary(X, y, W_softmax, b_softmax,
                       'Softmax Decision Boundary')
```

3 Mathematical Explanation

3.1 Task 1: Score Function Implementation

The score function computes the raw class scores for a linear classifier. For an input matrix $X \in R^{N\times D}$ (where N is the number of examples and D is the number of features) and a weight matrix $W \in R^{D\times K}$ (where K is the number of classes), the score function computes the following:

$$\text{scores} = X \cdot W + b \tag{1}$$

Where:

- X is the input matrix of shape (N, D), where each row represents an example and each column represents a feature.
- W is the weight matrix of shape (D, K), where each column corresponds to the weights associated with a specific class.
- b is a bias vector of shape $(K,)$, representing the bias for each class.

This score function is used to compute the raw scores for each class, which will be used in the loss function to measure the classifier's performance.

3.2 Task 2: SVM Hinge Loss

The hinge loss, commonly used in Support Vector Machines (SVM), creates a margin between the correct and incorrect classes. For a given training sample i and its correct class y_i, the hinge loss is defined as:

$$L_i = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + 1) \tag{2}$$

Where:

- S_j is the score for class j.
- S_{y_i} is the score for the correct class y_i.

The hinge loss encourages the correct class score to be at least one unit greater than the incorrect class scores. The total hinge loss for the dataset is given by:

$$L_{SVM} = \frac{1}{N} \sum_i L_i + \frac{1}{2} \lambda \sum_{i,j} W_{ij}^2 \tag{3}$$

Where the second term represents the L2 regularization to prevent overfitting by penalizing large weight values.

3.3 Task 3: Softmax Loss

The softmax loss is commonly used for multiclass classification. The softmax function converts the raw class scores into probabilities. The probability of class j for sample i is given by:

$$P_j = \frac{e^{S_j}}{\sum_k e^{S_k}} \tag{4}$$

Where:

- S_j is the score for class j.

The cross-entropy loss for sample i is defined as:

$$L_i = -\log(P_{y_i}) \tag{5}$$

Where P_{y_i} is the probability of the correct class y_i. The total softmax loss for the dataset is:

$$L_{Softmax} = \frac{1}{N} \sum_i L_i + \frac{1}{2} \lambda \sum_{i,j} W_{ij}^2 \tag{6}$$

The second term is the L2 regularization term to prevent overfitting by penalizing large weight values.

3.4 Task 4: Regularization

Both hinge loss and softmax loss use L2 regularization to prevent overfitting. The L2 regularization term is:

$$L_{reg} = \frac{1}{2}\lambda \sum_{i,j} W_{ij}^2 \tag{7}$$

Where:

- λ is a regularization parameter controlling the strength of the regularization.
- W_{ij} are the elements of the weight matrix W.

Regularization helps prevent the classifier from becoming overly complex, thus improving its generalization to unseen data.

3.5 Task 5: Gradient Descent

Gradient descent is an optimization algorithm used to minimize the loss function by iteratively updating the weights W and the bias b. The update rule for gradient descent is:

$$W = W - \eta \cdot \nabla L(W) \tag{8}$$

$$b = b - \eta \cdot \nabla L(b) \tag{9}$$

Where:

- η is the learning rate that controls the step size in each iteration.
- $\nabla L(W)$ and $\nabla L(b)$ are the gradients of the loss function with respect to the weights and bias.

For softmax loss, the gradient with respect to the weights is:

$$\nabla L(W) = \frac{1}{N} X^T \cdot (\text{probs} - y_{\text{one-hot}}) + \lambda W \tag{10}$$

Where:

- probs is the matrix of predicted probabilities for each class.
- $y_{\text{one-hot}}$ is the one-hot encoded label matrix indicating the correct class.
- X^T is the transpose of the input matrix.

Gradient descent iterates over several steps, updating the weights and bias in the direction of the negative gradient to minimize the loss function over time.

4 Conclusion

In this assignment, we successfully implemented and compared two linear classifiers—the SVM with hinge loss and the Softmax classifier with cross-entropy loss—on a synthetic spiral dataset. Both classifiers were optimized using gradient descent with L2 regularization to prevent overfitting.