Problem Set 2 - Python and LaTeX Practice

Dennis Kwadzode

October 14, 2024

1 Introduction

The goal of this assignment is to implement a simple linear classifier whiles exploring two different types of loss functions. Namely : **Support Vector Machine (SVM)** This involves using a hinge loss to build a classifier. The hinge loss function is commonly used in training linear SVMs, which aims to create a decision boundary that maximizes the margin between different classes. **Softmax Classifier** This entails using the cross-entropy function to build another classifier. This loss is used in multi-class classification problems to measure how well the model's predicted probability distribution aligns with the actual distribution of the classes.

Finally a gradient descent, which is an optimization technique that helps to minimize the above loss functions and also improve the accuracy of the classifiers will be implemented. This will help me gain a practical experience in designing and optimizing machine learning models through the usage of different loss functions whiles understanding how they influence model performance.

2 Methodology

i)The implementation of the score function for the linear classifier is based on the data generated using the code provided. This creates a spiral dataset for three classes (K=3) with 100 data points per class. Input data (X) - A 2D dataset with 300 data points (since N = 100 and K = 3) is generated where each point is described by two features (dimensionality D = 2). Each classes data points are arranged in a spiral pattern, which is generated using polar coordinates, with the radius(r) and angle(t). Class labels (y) - The class labels are represented by the array y, which contains 0,1, or 2 for the three respective classes. Score Function Implementation - The score function for a linear classifier is a simple linear transformation of the input data. f(X, W, b) = XW + b X: The input data matrix with shape (300,2) - each row is a 2D point. W: The weight matrix with shape (2,3), where the two rows correspond to the input dimensions and the three columns correspond to the classes. b: The bias term with shape (1,3), which is added to the result of the linear transformation. f(x,W,b): The score matrix where each row contains the score for each class for a given input point.

The score function computes the class scores for each point in the dataset by performing a matrix multiplication between the input data and the weights, then adding the bias . These scores are then used to determine the predicted class during classification. Visualizing the Dataset - The scatter plot visualizes the dataset using different colors foe each class. The score function will later classify these points into the correct classes based on their scores for each class.

ii) Hinge Loss (SVM Loss) The hinge loss is mostly used in support vector machines (svm) for classification tasks. It is designed to ensure that the correct class score is higher than the scores of incorrect classes by a margin, typically set to 1.If the score for the correct class is sufficiently higher than the incorrect class scores, the loss will be zero. Otherwise, the loss increases based on how close the incorrect class scores are to the correct class scores. The hinge loss also measures how well a classifier separates data points from different classes. Correct class score: The score for the correct class (indexed by y) is isolated. Margin calculation: A margin of 1 is added to the difference between the scores of the incorrect classes and the correct class score. This margin enforces a separation between the scores of the correct and incorrect classes. Loss Calculation: The function calculates the loss by checking if the score of any incorrect class exceeds the correct class score minus the margin. The loss of the correct class is set to zero. Final Loss: The total loss for a single example is the sum of all positive margins(losses from incorrect classes that are too close to the correct class). Softmax Loss(Cross - Entropy Loss) The softmax loss is used in multi-class classification problems and works together with the softmax function. This loss function penalizes incorrect classifications by using the predicted probabilities in each class. The softmax function converts the raw scores into probabilities by normalizing the exponentiated scores. The cross-entropy loss then compares the predicted probabilities with the actual class.

iii)Discuss the regularization used Adding L2 regularization to the loss functions for both SVM and Softmax classifiers helps prevent overfitting by penalizing large weights. L2 regularization adds a term to the loss function that depends on the square of the magnitude of the coefficients. These updated functions now include a term that penalizes the magnitude of the weights, which helps in reducing overfitting by discouraging the model from fitting the noise in the training data. Adjust the lambda reg parameter based on the level of regularization you need, which typically requires tuning using a validation set or via cross-validation.

iv)Explain the gradient descent method you implemented To implement a gradient descent to minimize a loss function such as the SVM or Softmax with regularization, a systematic approach that iteratively updates the model parameters (in this case, weights W) to reduce the loss was followed. This approach provides a clear view of how the loss decreases over iterations and allows to observe the behavior of the optimization process. Adjust learning rate and lambda reg based on a specific problem to achieve better convergence. Notes: Softmax Probabilities Calculation: It converts logits into probabilities which sum to 1 across classes for each sample. Loss Calculation: Includes both the data loss (cross-entropy) and the regularization term. Gradient Calculation: Computes

how much each weight contributed to the error and adjusts accordingly, also including the effect of regularization. Printing Results: Outputs the loss at every 10th iteration and final results.

3 Resutls

import numpy as np

def compute$_s oftmax_l oss_a nd_g radient(W, X, y, lambda_r eg) : num_e xamples =$
$X.shape[0] Numberoftrainingexamplesscores = np.dot(X, W) shift_s cores =$
$scores - np.max(scores, axis = 1, keepdims = True) Numericstabilityexp_s cores =$
$np.exp(shift_s cores) softmax_p robabilities = exp_s cores/np.sum(exp_s cores, axis =$
$1, keepdims = True)$

correct$_l og_p robs = -np.log(softmax_p robabilities[range(num_e xamples), y]) data_l oss =$
$np.sum(correct_l og_p robs)/num_e xamplesreg_l oss = 0.5 * lambda_r eg * np.sum(W *$
$W) loss = data_l oss + reg_l oss$

dscores = softmax$_p robabilitiesdscores[range(num_e xamples), y] - = 1 dscores/ =$
$num_e xamplesdW = np.dot(X.T, dscores) dW + = lambda_r eg * W$

return loss, dW

num$_e xamples = 300 dim = 5 num_c lasses = 3$

np.random.seed(42) X = np.random.randn(num$_e xamples, dim) y = np.random.randint(num_c lasses, si$
$num_e xamples) learning_r ate = 1e - 2 num_i terations = 200 lambda_r eg = 0.1$

W = 0.001 * np.random.randn(dim, num$_c lasses$)

for i in range(num$_i terations$) : $loss, grad = compute_s oftmax_l oss_a nd_g radient(W, X, y, lambda_r eg) W -$
$learning_r ate * gradUpdateweights$

if i print(f'Iteration i, loss: loss')

print("Final loss:", loss) print("Optimized weights:", W)

4 Conclusion

In this assignment, I implemented a linear classifier using SVM and Softmax loss functions, along with gradient descent optimization, on a synthetic spiral dataset. The dataset was designed to test the classifier's ability to handle non-linear data distributions due to its challenging spiral layout of three distinct classes.The assignment successfully demonstrated the application of linear classifiers to a complex, non-linear dataset. Key takeaways include the importance of regularization, the effectiveness of gradient descent for optimization, and the challenges posed by complex data distributions. The experience gained from this implementation provides a solid foundation for tackling more sophisticated machine learning tasks involving non-linear and high-dimensional data. The project also emphasized the need for continuous learning and adaptation of strategies in the face of new and challenging datasets.

Implementation Details

Score Function: Created to compute the raw scores from the input data using the initial random weights. Loss Functions: Developed both SVM (hinge

loss) and Softmax (cross-entropy loss) functions to measure the performance of the classifier. The hinge loss was used for its robustness in handling non-linear separations by focusing on misclassified points, while the cross-entropy loss provided a probabilistic measure that is particularly useful for multi-class classification. Regularization: Added L2 regularization to the loss functions to prevent overfitting, which is crucial for maintaining the model's generalization on unseen data. Gradient Descent: Implemented to optimize the loss functions. This iterative process adjusted the model weights based on the computed gradients to minimize the loss, significantly improving the classifier's accuracy over iterations. Data Visualization Visual analysis was performed by plotting the dataset, which revealed the intricate spiral pattern of the classes. This helped in understanding the complexity of the dataset and setting expectations for the classifier's performance.

Challenges Faced

Model Complexity: The non-linear nature of the spiral data required careful tuning of the model parameters and regularization factor to achieve satisfactory results without overfitting. Optimization Difficulties: The gradient descent had to be meticulously configured (e.g., learning rate and number of iterations) to ensure convergence. Initial settings led to either slow convergence or overshooting the minimum. Balancing Loss Components: Combining the loss from the score function with regularization posed a challenge in balancing their contributions to the overall loss, necessitating several experiments to find an optimal regularization strength. Computational Efficiency: The calculation of gradients for all data points was computationally intensive, highlighting the need for efficient vectorized implementations to speed up the training process.