Problem Set 3 Python and Latex Practice

Dennis Kwadzode

November 25, 2024

1 Introduction

The primary objective of this assignment is to explore the foundational concepts of neural networks through a hands-on approach. This involves understanding the theoretical underpinnings of neural networks while also implementing and training a simple neural network in Python. Specifically, the goals include:

1. Understanding the Basic Architecture of Neural Networks: Learn the structure of a neural network, including input, hidden, and output layers. Understand the role of weights, biases, and the flow of data through layers during forward propagation.

2. Exploring Activation Functions: Study common activation functions like Sigmoid, ReLU, and others to introduce non-linearity. Implement these functions in Python and understand their impact on network learning.

3. Implementing Forward Propagation: Learn how data moves through the network to generate predictions. Write Python code to calculate activations for each layer based on the inputs and weights.

4. Addressing Overfitting Through Regularization: Understand the concept of overfitting and how it hampers generalization. Implement L2 regularization to penalize large weights, promoting simpler models.

5. Adaptive Learning with Optimization Algorithms: Study the importance of learning rates and their impact on training. Implement adaptive learning methods like the Adam optimizer to enhance the efficiency of gradient descent. Hands-on Coding Exercise:

6. Gain practical experience by implementing neural network components in Python, such as: Initialization of weights and biases. Forward and backward propagation. Loss computation with regularization. Parameter updates using gradient descent and Adam optimizer. Train a basic neural network on a small dataset to observe how the network learns.

7. Developing a Conceptual and Practical Understanding: Use Python coding exercises to bridge the gap between theory and practice. Analyze training results, loss reduction, and the network's ability to generalize. Expected Outcomes

8. By completing this assignment, you should: Have a solid grasp of the foundational concepts of neural networks. Be able to implement a simple neural

network from scratch in Python. Understand how different components (activation functions, regularization, optimizers) influence learning. Be equipped to extend your knowledge to more complex neural network architectures and larger datasets. This assignment aims to provide both theoretical knowledge and practical skills, preparing you to tackle real-world problems using neural networks effectively.

2 Methodology

1. Understanding Neural Network Architecture Objective: Learn the structure of a neural network, including input, hidden, and output layers. Method: Study the role of neurons, weights, biases, and layers. Implement a basic 3-layer neural network (input layer, one hidden layer, and an output layer) using Python. Randomly initialize weights and biases to prepare the network for training.

2. Implementing Activation Functions Objective: Introduce non-linearity into the neural network, enabling it to learn complex patterns. Method: Study and implement key activation functions, such as: Sigmoid: For outputs in the range [0, 1], ideal for binary classification. ReLU (Rectified Linear Unit): A popular choice for hidden layers due to its simplicity and efficiency. Leaky ReLU: To address the vanishing gradient problem in ReLU. Write Python functions for each activation function and their derivatives (for backpropagation).

3. Forward Propagation Objective: Compute the output of the neural network for given inputs by passing data through the layers. Method: Implement forward propagation, which includes: Calculating the weighted sum of inputs and biases for each layer. Applying the activation function to produce outputs for the next layer. Ensure the final output reflects the target format (e.g., probabilities for binary classification).

4. Regularization Techniques Objective: Prevent overfitting by penalizing large weights and improving the model's ability to generalize. Method: Implement L2 regularization, adding a penalty term to the loss function based on the sum of squared weights. Incorporate regularization into the gradient calculations during backpropagation.

5. Adaptive Learning Techniques Objective: Use an optimizer that adapts the learning rate during training for faster and more stable convergence. Method: Implement the Adam optimizer, which combines the advantages of momentum and RMSprop: Use exponentially weighted averages for gradients and their squares. Update parameters using bias-corrected estimates of these averages. Integrate Adam into the neural network's weight update step.

6. Coding and Training Objective: Train the neural network on a sample dataset, ensuring it learns to map inputs to outputs effectively. Method: Write Python code to: Perform forward propagation to compute predictions. Calculate the loss using a loss function (e.g., Mean Squared Error). Compute gradients via backward propagation. Update weights and biases using the Adam optimizer. Train the network for multiple epochs, printing the loss periodically to monitor progress.

7. Evaluate Results Objective: Assess the performance of the trained network. Method: Evaluate the network on the training data to ensure it learned the mapping effectively. Visualize loss reduction over epochs (e.g., using a loss curve). Analyze the final outputs to ensure they align with the expected results.

8. Reporting Objective: Document the entire process, results, and conclusions. Method: Write a LaTeX report summarizing: The neural network architecture and design choices. Implementation details of forward propagation, regularization, and adaptive learning. Training results, including loss curves and example predictions. Observations and potential areas for improvement.

Summary The methodology involves a systematic approach to implementing and training a neural network while understanding its foundational components. The practical exercises reinforce theoretical knowledge and provide hands-on experience in building machine learning models. This assignment lays the groundwork for more advanced neural network concepts and architectures.

3 Results

import numpy as np

Define the architecture of the network input$_size = 784 Example for MNIST dataset hidden_size =$ $128 Number of neurons in the hidden layer output_size = 10 Number of classes for MNIST$

Initialize weights and biases np.random.seed(42) For reproducibility W1 = np.random.randn(input$_size, hidden_size) * 0.01 b1 = np.zeros((1, hidden_size)) W2 =$ $np.random.randn(hidden_size, output_size) * 0.01 b2 = np.zeros((1, output_size))$

import matplotlib.pyplot as plt

Activation functions def sigmoid(x): return 1 / (1 + np.exp(-x))

def relu(x): return np.maximum(0, x)

def leaky$_relu(x, alpha = 0.01) : return np.where(x > 0, x, x * alpha)$

Plotting x = np.linspace(-10, 10, 100) plt.figure(figsize=(10, 6)) plt.plot(x, sigmoid(x), label='Sigmoid') plt.plot(x, relu(x), label='ReLU') plt.plot(x, leaky$_relu(x), label =$ $'Leaky ReLU') plt.title('Activation Functions') plt.xlabel('Input value (x)') plt.ylabel('Output value') plt.legend()$

def forward$_propagation(X) : Hidden layer Z1 = np.dot(X, W1) + b1 A1 =$ $relu(Z1) Using ReLU for hidden layer$

Output layer Z2 = np.dot(A1, W2) + b2 A2 = sigmoid(Z2) Using sigmoid for output layer (for binary classification)

return A2 This is the output of the network

Example usage X$_sample = np.random.randn(1, input_size) Simulated input output =$ $forward_propagation(X_sample) print("Network output:", output)$

def compute$_loss_with_regularization(A2, Y, parameters, lambda_reg) : m =$ $Y.shape[0] number of examples W1, W2 = parameters['W1'], parameters['W2']$

Cross-entropy part cross$_entropy_loss = -np.mean(Y * np.log(A2) + (1 - Y) *$ $np.log(1 - A2))$

L2 Regularization part l2$_loss = (lambda_reg/(2*m)) * (np.sum(np.square(W1)) +$ $np.sum(np.square(W2)))$

loss = cross$_entropy_loss + l2_loss return loss$

def backward$_propagation(X, Y, W1, b1, W2, b2, lambd) : m = X.shape[1]$

Forward propagation Z1 = np.dot(W1, X) + b1 A1 = relu(Z1) Z2 = np.dot(W2, A1) + b2

Loss derivatives (MSE + L2 Regularization) dZ2 = Z2 - Y dW2 = (1 / m) * np.dot(dZ2, A1.T) + (lambd / m) * W2 db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)

dA1 = np.dot(W2.T, dZ2) dZ1 = dA1 * (Z1 ¿ 0) Derivative of ReLU dW1 = (1 / m) * np.dot(dZ1, X.T) + (lambd / m) * W1 db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

Return gradients return dW1, db1, dW2, db2

def gradient$_descent(X, Y, W1, b1, W2, b2, learning_rate = 0.01, epochs = 1000, lambd =$ 0.1) : $Listtostorelossvaluesforplottingloss_history = []$

Gradient Descent Loop for epoch in range(epochs): Forward propagation Z1 = np.dot(W1, X) + b1 A1 = relu(Z1) Z2 = np.dot(W2, A1) + b2

Compute loss with L2 regularization loss = compute$_loss_with_L2(Y, Z2, W1, W2, lambd)loss_history.append(l$

Backward propagation dW1, db1, dW2, db2 = backward$_propagation(X, Y, W1, b1, W2, b2, lambd)$

Update parameters (weights and biases) W1 -= learning$_rate * dW1b1- =$ $learning_rate * db1W2- = learning_rate * dW2b2- = learning_rate * db2$

Print loss every 100 epochs if epoch print(f'Epoch epoch/epochs, Loss: loss")

Return final weights, biases, and loss history return W1, b1, W2, b2, loss$_history$

def initialize$_adam(parameters) : v = s = forkey, valueinparameters.items() :$ $v["d" + key] = np.zeros_like(value)s["d" + key] = np.zeros_like(value)returnv, s$

def update$_parameters_with_adam(parameters, grads, v, s, t, learning_rate = 0.01, beta1 =$ $0.9, beta2 = 0.999, epsilon = 1e - 8) : v_corrected = s_corrected =$

for key in parameters.keys(): v["d" + key] = beta1 * v["d" + key] + (1 - beta1) * grads['d' + key] v$_corrected["d" + key] = v["d" + key]/(1 - beta1 * *t)$

s["d" + key] = beta2 * s["d" + key] + (1 - beta2) * (grads['d' + key] ** 2) s$_corrected["d" + key] = s["d" + key]/(1 - beta2 * *t)$

parameters[key] -= learning$_rate*v_corrected["d"+key]/(np.sqrt(s_corrected["d"+$ $key]) + epsilon)returnparameters, v, s$

4 Conclusion

This assignment provided a comprehensive exploration of the foundational concepts of neural networks. Through theoretical understanding and practical coding exercises, key components of neural networks were implemented, including:

Basic Architecture:

The neural network structure was designed with an input layer, hidden layers, and an output layer, demonstrating how layers of neurons interact to process information and produce predictions. Activation Functions:

Various activation functions, such as Sigmoid and ReLU, were implemented to introduce non-linearity, enabling the network to model complex relationships within the data. Forward Propagation:

The process of passing inputs through the network to compute outputs was explored, reinforcing how weighted sums and activations work together to produce predictions. Regularization:

Techniques such as L2 regularization were integrated to prevent overfitting, ensuring the model generalizes well to unseen data by penalizing large weights. Adaptive Learning:

The Adam optimizer was used to enhance training efficiency by dynamically adjusting learning rates based on past gradients, showcasing modern optimization strategies in deep learning. Coding Implementation:

Neural network components were implemented in Python, enabling hands-on experience in constructing, training, and fine-tuning a simple neural network. Training and Results:

The network was successfully trained on sample data. Observations included a steady decrease in loss over epochs, demonstrating the model's ability to learn and approximate the underlying data patterns effectively.

Key Takeaways Neural networks are powerful tools for modeling non-linear relationships in data. Understanding the interplay of architecture, activation functions, and optimization strategies is critical for building effective models. Regularization and adaptive learning enhance model robustness and training efficiency. Practical implementation solidifies theoretical knowledge, bridging the gap between concepts and real-world applications. This assignment has provided foundational skills and knowledge essential for further exploration of advanced neural network architectures and their applications in fields like image recognition, natural language processing, and beyond.