Joshua Matni
CS 422 Writeup

Linear Regression:

Write-Up (20 points)

Give a brief description of the algorithm. Explain what the algorithm's strengths and shortcomings are. How does gradient descent get expanded into use for neural networks?

Linear regression is a fundamental statistical method used to model the relationship between a dependent variable and one or more independent variables. The algorithm fits a linear equation to observed data by finding the line (in simple linear regression) or hyperplane (in multiple linear regression) that minimizes the sum of the squared differences (residuals) between the observed values and the values predicted by the linear model. This is commonly achieved using the least squares method.

Strengths of Linear Regression:

- Simplicity and Interpretability:
  - Easy to implement and understand.
  - Coefficients directly indicate the influence of each independent variable.
- Computational Efficiency:
  - Requires less computational resources compared to complex algorithms.
- Analytical Solution:
  - Closed-form solution exists, allowing for quick calculations without iterative methods.
- Baseline Performance:
  - Serves as a good starting point for modeling before moving to more complex models.

Shortcomings of Linear Regression:

- Assumption of Linearity:
  - Cannot capture nonlinear relationships between variables.
- Sensitivity to Outliers:
  - Outliers can disproportionately affect the model, leading to misleading results.
- Multicollinearity Issues:
  - High correlation between independent variables can distort the significance of predictors.
- Assumes Homoscedasticity and Normality:
  - Violations of constant variance and normal distribution of errors can impact the validity of the model.

Expansion of Gradient Descent into Neural Networks:

- Backpropagation Algorithm:
  - Gradient descent is extended through backpropagation, allowing the calculation of gradients for each layer in a neural network.

- Adjusting Weights and Biases:
    - Gradually updates the weights and biases in the network to minimize the loss function.
- Handling Non-Linearities:
    - Works with activation functions to model complex, non-linear patterns in data.
- Scalability:
    - Variants like Stochastic Gradient Descent (SGD) and Mini-Batch Gradient Descent enable training on large datasets.
- Optimization Techniques:
    - Advanced methods like Adam, RMSprop, and Momentum build upon gradient descent to improve convergence speed and performance.

Neural Networks:

Write-Up (30 points)
Describe the neural network algorithm provided in the code to the best of your ability. Indicate what your parameters were, and what they do in the final implementation. For example, describe what the learning rate does. For the optimizer functions, you do not have to write out all the math, but give a brief overview of how the two functions differ from one another in a general sense. Describe at a high level what cross entropy loss does.

The provided code implements a simple feedforward neural network using PyTorch for binary classification on a synthetic dataset. The main components of the algorithm are:

- Data Generation:
    - Creates two classes (class0 and class1) with 100 data points each in a 2D space.
    - Data points are generated using normal distributions centered at different means to create separable classes.
    - Labels are assigned as 0 for class0 and 1 for class1.
- Network Architecture:
    - Input Layer: Accepts two features per data point (coordinates in 2D space).
    - Hidden Layer:

        - Contains a specified number of neurons (n_hidden), set via the hidden parameter.
        - Applies the ReLU (Rectified Linear Unit) activation function to introduce non-linearity.
    - Output Layer:
        - Contains two neurons corresponding to the two classes.
        - Outputs raw scores (logits) without an activation function, as the loss function (CrossEntropyLoss) handles that internally.
- Forward Pass (forward method):
    - Data flows through the hidden layer with ReLU activation.
    - The output from the hidden layer is passed to the output layer to obtain the final predictions.
- Training Loop:
    - For each epoch:
        - Prediction: Computes the network's output for the input data.
        - Loss Computation: Calculates the loss using Cross Entropy Loss, comparing predictions with true labels.
        - Backpropagation:
            - Clears previous gradients (optimizer.zero_grad()).
            - Computes gradients of the loss with respect to network parameters (loss.backward()).
        - Parameter Update: Adjusts the network's weights using the chosen optimizer (optimizer.step()).
        - Visualization (every few epochs):
            - Plots the data points colored by predicted class to visualize the decision boundary and learning progress.

Parameters and Their Roles

- Hidden Layer Size (hidden):
    - Value: 4
    - Role: Determines the number of neurons in the hidden layer.
        - A larger size increases the network's capacity to learn complex patterns.
        - Too few neurons may lead to underfitting; too many may cause overfitting.
- Learning Rate (learning_rate):
    - Value: 0.01
    - Role: Controls the step size during the optimization process.
        - A higher learning rate can speed up training but may skip over minima.
        - A lower learning rate ensures more precise convergence but requires more epochs.
- Epochs (epochs):
    - Value: 100
    - Role: Specifies the number of times the entire dataset is passed through the network.

        - More epochs allow the model to learn better but increase computation time.
        - Too many epochs may lead to overfitting on the training data.
- Optimizer:
    - Choice: Adam optimizer (torch.optim.Adam(net.parameters(), lr=learning_rate))
    - Role: Updates the network's weights to minimize the loss function.
        - Uses adaptive learning rates for each parameter.
        - Combines the benefits of two other extensions of SGD, AdaGrad and RMSProp.

Differences Between Optimizer Functions

- Stochastic Gradient Descent (SGD):
    - Mechanism:
        - Updates parameters by moving in the opposite direction of the gradient of the loss function.
        - Uses a constant learning rate throughout training.
    - Characteristics:
        - Simple and easy to implement.
        - Can be slow to converge, especially in the presence of noisy gradients.
        - Sensitive to the choice of learning rate.
    - Usage Scenario:
        - Effective for convex problems and smaller datasets.
        - When computational resources are limited.
- Adam Optimizer:
    - Mechanism:
        - Computes adaptive learning rates for each parameter.
        - Maintains exponentially decaying averages of past gradients and squared gradients.
    - Characteristics:
        - Generally faster convergence than SGD.
        - Less sensitive to initial learning rates.
        - Handles sparse gradients and non-stationary objectives well.
    - Usage Scenario:
        - Suitable for large datasets and complex neural networks.
        - When quick convergence is desired.

Cross Entropy Loss

- Purpose:
    - Measures the performance of a classification model whose output is a probability value between 0 and 1.
    - Quantifies the difference between the predicted probability distribution and the actual distribution (true labels).
- Functionality:
    - High Penalty for Incorrect Predictions:

        - Assigns a high loss value when the predicted probability for the true class is low.
    - Mathematical Operation:
        - Calculates the negative log-likelihood of the true class probabilities.
        - Encourages the model to output high probabilities for the correct class.
- Why It's Used:
    - Suitable for multi-class classification problems with mutually exclusive classes.
    - Works well with logits (raw outputs) from the network, as it internally applies the softmax function.
    - Provides a smooth and continuous loss landscape, which is beneficial for optimization.