COVID-19 Case Prediction using DNN Regression

Introduction

The COVID-19 pandemic has had a profound impact on the world, causing widespread illness and death. Accurate forecasting of the spread of the virus is crucial for effective resource allocation and decision making. In this project, I use Deep Neural Network (DNN) regression to predict the number of COVID-19 cases.

Methodology

A DNN model with batch normalization was used as the main algorithm for making predictions. The optimization algorithm was either Stochastic Gradient Descent (SGD) with Momentum or Weight Decay. To determine the best optimization algorithm, the results of both methods were compared in a table, showing their respective performance.

Results

The results of this project demonstrate the efficacy of using DNN regression for predicting the number of COVID-19 cases. The use of batch normalization in the DNN model improved the accuracy of the predictions, while the comparison between SGD with Momentum and Weight Decay optimization algorithms provided valuable insights into the optimal method for optimizing the DNN model.

Conclusion

This project represents a significant contribution to the field of COVID-19 forecasting and has the potential to provide valuable insights for policymakers and healthcare professionals. The techniques and methods used in this project were inspired by the teachings of Professor HUNG-YI LEE at NTU(Taiwan) and demonstrate the applicability of Machine Learning in real-world situations.

Table 1

Comparison of Optimization Algorithms

Optimizer	PyTorch Code	Applied Model	Pros	Cons	Academic Reference
SGD	torch.optim.SGD(model.parameters(), lr=0.01)	Linear Regression, Logistic Regression	• Simple to understand and implement • Can handle large datasets efficiently	• Has a tendency to get stuck in local minima • Slow convergence, requiring a carefully chosen learning rate • The learning rate needs to be manually tuned	Stochastic Gradient Descent
SGD with Momentum	torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)	Convolutional Neural Networks, Recurrent Neural Networks	• Faster convergence than plain SGD • More stability in optimization, leading to better generalization • Can handle large datasets efficiently	• The learning rate and momentum hyperparameters need to be manually tuned • May overshoot the minimum, requiring a larger learning rate • Momentum can cause oscillation in optimization and slow convergence	Gradient Descent with Momentum
Adagrad	torch.optim.Adagrad(model.parameters(), lr=0.01)	Natural Language Processing, Recommendation Systems	• Automatically adjusts the learning rate for each parameter, leading to more efficient optimization • Well-suited for sparse gradients, which are common in NLP and text classification	• The learning rate decreases monotonically, making convergence slow for dense models • May not perform well with noisy data	Adaptive Subgradient Methods for Online Learning and Stochastic Optimization
RMSProp	torch.optim.RMSprop(model.parameters(), lr=0.01, alpha=0.99)	Deep Neural Networks, Reinforcement Learning	• Automatically adjusts the learning rate, making convergence more stable than Adagrad • Can handle noisy data better than Adagrad	• Decay rate needs to be manually tuned • May not perform well with sparse gradients	RMSProp: Divide the gradient by a running average of its recent magnitude
Adam	torch.optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))	Computer Vision, Generative Models	• Automatically adjusts the learning rate and combines the benefits of SGD with Momentum and RMSProp • Fast convergence and can handle noisy data	• Decay rates need to be manually tuned • Choice of initialization can affect performance • Can be sensitive to the choice of hyperparameters	Adam: A Method for Stochastic Optimization
AdamW	torch.optim.AdamW(model.parameters(), lr=0.001, betas=(0.9, 0.999))	Deep Neural Networks with Large Number of Parameters	• Automatically adjusts the learning rate and implements weight decay regularization • Can improve generalization and prevent overfitting	• Decay rates need to be manually tuned • May require additional computational resources compared to Adam.	