PROJECT 6

OBJECT-ORIENTED PROGRAMMING AND MACHINE LEARNING

STUDENT NAME: Meenakshi Sridharan Sundaram
HAWK NUMBER: A20592581
PROFESSOR NAME: Dr. Won Jae Yi
SUBJECT CODE: ECE590
PROJECT DUE: December 4th, 2024

Acknowledgement

I acknowledge all works, including figures, codes, and writings, belong to me and/or persons who are referenced. I understand if any similarity in the code, comments, customized program behaviour, report writings, and/or figures are found, both the helper (original work) and the requestor (duplicated/modified work) will be called for academic disciplinary action.

Date and Time: December 4th, 2024, 11:59 PM
Signature: Meenakshi Sridharan Sundaram

I. INTRODUCTION

In modern machine learning, the ability to effectively train neural networks represents a critical milestone in developing intelligent systems that can solve complex tasks. This project implements a Convolutional Neural Network from scratch in Python using NumPy to classify handwritten digits from the MNIST dataset into one of ten classes (0–9). The MNIST dataset, because of its simplicity, structured format, and broad applicability, has turned into a benchmark in machine learning.

a) Objective

The key goal of this project is to implement and train a CNN, making use of the fundamental ideas of convolutional layers, pooling operations, backpropagation, and Stochastic Gradient Descent. This assignment is not only supposed to give one a high classification accuracy but also allow one to explore how various architectural choices and hyperparameters impact the performance of a model.

b) Motivation

The CNN introduced a revolution to computer vision by providing a framework that automatically learns the spatial hierarchies of features directly from raw data. Compared to fully connected MLPs, CNNs require fewer parameters, and the inherent structure makes them much better for image data. This makes them indispensable in tasks related to handwriting recognition, object detection, and image classification.

c) Implementation Approach

The code does not use any other modern libraries like TensorFlow or PyTorch to go deep into the basics of CNNs. The major components include:

- Convolutional Operations: Shared kernels that extract spatial features from the input images
- Pooling Layers: Down-sample feature maps to retain significant information while reducing dimensionality
- Activation Functions: Introduction of non-linearity to allow complex patterns to be learned
- Loss Function: Measures the difference between the predicted probabilities and the true labels.
- Backpropagation: Calculate the gradients of the loss concerning network parameters .
- SGD Optimization: Iterative update of weights and biases to converge on the minimum of the loss function.

d) Challenges Addressed

- Forward and backward pass for convolutional and pooling layers.
- Efficient design of a training loop for stable convergence.

e) Investigation of effects of important hyperparameters:

- Bound: Upper limit of initialization range for weights and biases.
- Epsilon: Step size for the SGD.
- Batch Size: Controls the trade-off between computational efficiency and accuracy of gradient estimation. Scope This project will train the CNN on 20,000 training samples and validate the model using 1,000 samples of the MNIST dataset. The model is trained over five epochs to evaluate the performance while making sure it's computationally feasible and achieves competitive accuracy.

Thus, understanding all these aspects of a CNN, this project does much more than implement the simple CNN but builds ground to apply such techniques for much-advanced models and datasets.

---

2. BODY

a) Training Process

The training pipeline for the CNN model involves the following:

- Data Preprocessing: The MNIST dataset is split into a training set of 20,000 samples and a validation set of 1,000 samples.
- Initialization: Weights and biases are initialized uniformly in the range [0.01,0.09]
- Forward Pass: Images pass through convolutional layers, ReLU activations, max-pooling layers, and a fully connected layer to compute predictions.
- Loss Calculation: Cross-entropy loss is computed for each batch.
- Backpropagation: Gradients of the loss w.r.t. weights and biases are computed by the chain rule.
- Parameter Update: Weights and biases are updated by SGD with a learning rate in the range of 0.0001 to 0.001

b) Hyperparameter Analysis

Hyperparameter tuning is a fundamental part of training neural networks to assure peak performance. This project systematically varied the hyperparameters-bounds, epsilon, and batch size across five epochs to examine their impact on model convergence and accuracy. The values of these parameters for each epoch, and the final validation accuracy obtained with them, are shown in the table above.

1. Bounds

The bounds parameter is a way to control the range from which weights and biases are initially set. It affects the starting point that the model begins with in the optimization landscape.

In the first epoch, the bounds were 0.09, giving a relatively wide range for initialization. This makes the model explore a larger space of potential solutions, thus converging faster. However, such wide bounds do risk greater variance in the initial predictions.

While training, the bounds gradually decreased to 0.01 in the last epoch. This narrowed the range and helped in stabilizing the gradients, reducing overfitting, hence fine-tuning the model's parameters to a high accuracy of 97.9% reached at the third epoch.

2. Epsilon (Learning Rate)

defines the size of the update of the parameters in each step of optimization. A good balance between speed and stability is important for effective learning.

During the first epoch, the learning rate was 0.0009, which allowed for quick improvements in accuracy as the model learned core features of the dataset. At the end of this epoch, the accuracy was 96.7%.

The learning rate decreased from across subsequent epochs to 0.0001, thus ensuring that as the model approaches the optimal solution, updates become more precise. Such refinement enabled the model to achieve consistently 97.9% accuracy in the final epochs without overshooting or oscillations.

3. Batch Size

Batch size refers to the number of samples processed before updating the model parameters. It influences computational efficiency and the stability of gradient estimates.

Starting with a batch size of 10 in the first epoch, the model achieved a trade-off between computational cost and gradient estimation accuracy. This batch size allowed it to converge rapidly to an initial accuracy of 96.7%.

Over subsequent epochs, the batch size was progressively reduced, reaching 1 in the final epoch. While this granular update method increased computation time, it allowed for highly precise updates, ensuring the model consistently achieved high accuracy.

c) Observations

1. Validation Accuracy Trends

The model quickly reached an accuracy of 96.7% within the first epoch. This reflects the appropriateness of the initial parameters, including a wide bound, moderately high learning rate, and a reasonable batch size.

By the third epoch, the model converged to an accuracy of 97.9%, which reflects successful convergence. Further epochs beyond this maintained this accuracy, reflecting the robustness of the model's learning process.

2. Impact of Hyperparameters

- Bounds: Larger bounds during initial epochs encouraged exploration while small in later epochs refined the parameters and reduced overfitting.
- Epsilon: This reduction in learning rate facilitated quicker learning in the initial steps, followed by more precise updates in later epochs.
- Batch size: Smaller batch sizes in the final epochs improved the accuracy of gradients, adding value to the model's superior results.

3. Stabilization:

It was already converging well by the third epoch, with most of the improvements in accuracy happening in later epochs. This point drives home a very crucial aspect: hyperparameters should be reduced dynamically as the training progresses.

3.1 Efficiency and Accuracy Trade-offs

Smaller batch sizes in later epochs increased computation time but resulted in more accurate parameter updates. This kind of trade-off was therefore helpful to obtain high final accuracy.

4. Comments

The systematic variation of hyperparameters over epochs illustrates some key observations:

4.1 Gradual Improvement: Keeping higher bounds and learning rates for the initial few epochs served to converge fast, while reducing the learning rate and bounds in later epochs just fine-tuned the model and allowed it to converge better.

4.2 Importance of Batch Size: Smaller batch sizes in later epochs proved to be crucial in improving the performance of the model, at the cost of increased computation time.

4.3 Smoothness Over Epochs: The model has achieved an accuracy of 97.9% for the last epochs, which says that the parameters were well tuned, and the model learned strongly.

The results also show that hyperparameter tuning is not a 'one-size-fits-all' approach but rather dynamic and requires careful tuning depending on the training phase.

5. Conclusion

The importance of hyperparameters while training a CNN model using the MNIST dataset has been shown in this project. Some key takeaways are

- Bounds: A higher bound on weight initialization in the beginning promotes faster convergence, while later epochs a reduced bound stabilizes training and overfitting.
- Epsilon: A moderate high learning rate allows rapid learning in the initial stages but smaller values ensure precise updates in later stages.
- Batch Size: Larger batch sizes are computationally efficient for the early epochs, while small batch sizes in later epochs lead to an improvement in the accuracy of gradient estimation and model generalization.

The CNN has achieved a peak validation accuracy of 97.8%, reflecting the effectiveness of the tuning strategy. The gradual reduction of bounds, epsilon, and batch size over epochs created a training regime with a balance between convergence speed, accuracy, and computational efficiency. Future works may also apply adaptive learning rate techniques

or other advanced optimization algorithms such as Adam to further improve the performance.

Epoch Number	*Bounds*	*Epilson*	*Batch Size*	*Final Accuracy*
1	0.09	0.0009	10	0.967
2	0.07	0.0007	08	0.970
3	0.05	0.0005	06	0.979
4	0.03	0.0003	03	0.977
5	0.01	0.0001	01	0.979

5.1 Comparing with MLP (Project 5)

Where Project 5 implemented an MLP, the CNN used in Project 6 is a huge leap ahead. While both models classify handwritten digits from the MNIST dataset, their design and capability differences are fundamental to each other, affecting their performance and suitability for image data.

The MLP processes the MNIST images as flattened 1D arrays of pixels, which means that the spatial structure is lost. In contrast, the CNN directly processes the 2D spatial structure of images, preserving critical information about the arrangement of pixels. This inherent difference allows the CNN to extract meaningful patterns, such as edges, textures, and shapes, which are essential for image classification tasks.

The MLP requires fully connected layers where every neuron is connected to all input features. This design results in a much larger number of parameters, which increases computational cost and, especially for large input sizes, also increases the risk of overfitting. On the other hand, CNN reduces the number of parameters by using shared convolutional filters that slide over the input, enabling efficient learning of spatial features in a computationally less demanding way.

From a performance point of view, CNN shows much better classification performance compared with MLP. Due to the convolutional and pooling layer, CNN can capture image features hierarchically. Because of that, it would be more suitable for image classification problems like MNIST. Also, ReLU activation and max-pooling in the proposed CNN introduce more nonlinearity and dimensionality reduction than in MLP, making this model more robust and invariant.

Another point where the CNN outperforms the MLP is in terms of training speed. While the CNN is much more architecturally complex, sharing filters and a reduction in the parameter space result in faster convergence during training. The MLP, with its dense connections, requires more computational resources and takes longer to train effectively.

In summary, the CNN of Project 6 exploits its architecture to better leverage the spatial structure of image data and, as a result, is more accurate and computationally efficient than the MLP from Project 5. While the MLP was illustrative, the CNN indicates practical benefits of modern neural network architectures for image classification tasks.

5.2 The code was graded with the grading script for project 6 and was pushed into the git server using the command git push and below is the screenshot of it