Comparative Study of Learning Rates, Batch Sizes, and Dropout Rates in Neural Networks for MNIST Classification

Authors Name:
Sneha Naragundkar
Dept of Electronics and Communication

Authors Name:
Dr.Anil Gavade
Dept of Electronics and Communication

ABSTRACT

This paper explores the impact of hyperparameter tuning on the performance of a feedforward neural network for image classification using the MNIST dataset. The study focuses on the effects of four key hyperparameters: learning rate, batch size, dropout rate, and number of hidden units. A series of experiments were conducted by varying these hyperparameters and evaluating their impact on test accuracy, test loss, and training time. The model was trained using different configurations, with early stopping employed to prevent overfitting. Results indicate that hyperparameter choices significantly influence model performance, with specific combinations leading to faster training times and higher accuracy. This research highlights the importance of careful hyperparameter selection and provides insights into how different settings affect neural network training. The findings contribute to a better understanding of how to optimize deep learning models for image classification tasks, offering practical guidance for future work in this area.

KEYWORDS

1. Adam
2. Batch size
3. Dropout rate
4. Learning rate
5. MNIST
6. Neural network
7. Optimizer
8. Test accuracy
9. Training time

INTRODUCTION

In deep learning, selecting the right hyperparameters is crucial for training effective neural networks, particularly in image classification tasks. The performance of models often depends on the choice of hyperparameters such as the learning rate, batch size, dropout rate, and the number of hidden units in the network. Despite their importance, finding the optimal configuration for these hyperparameters is a challenging and time-consuming task, typically requiring extensive experimentation and fine-tuning. This paper investigates the influence of different hyperparameter settings on the performance of a feedforward neural network trained on the MNIST dataset, a widely used benchmark for handwritten digit classification. Specifically, the study examines how variations in learning rate, batch size, dropout rate, and hidden units impact key performance metrics such as test accuracy, test loss, and training time. By conducting a series of experiments with different hyperparameter combinations, the study aims to identify the most effective settings for achieving high accuracy while minimizing training time and overfitting. The results highlight the significant role of hyperparameter tuning in optimizing neural network performance and provide valuable insights for practitioners looking to enhance the efficiency and generalization of deep learning models in similar tasks.Finally, complete content and organizational editing beforeformatting. Please take note of the following items when proofreading spelling and grammar:

METHODOLOGY

The methodology for this study involves conducting a series of experiments to evaluate the impact of various hyperparameters on the performance of a feedforward neural network trained on the MNIST dataset. The process consists of the following steps:

1.Dataset Preparation:

The MNIST dataset, consisting of 28x28 grayscale images of handwritten digits (0–9), is loaded using TensorFlow's built-in keras.datasets.mnist.load_data() function. The dataset is then normalized by dividing the pixel values by 255.0, ensuring that the input features are in the range [0, 1].

2.Model Architecture:

A simple feedforward neural network is built using TensorFlow's Keras API. The network consists of:

A Flatten layer to reshape the 28x28 input images into a 1D vector.

Two fully connected Dense layers, each with hidden units and ReLU activation functions.

Two Dropout layers, applied after each hidden layer to reduce overfitting, with a configurable dropout rate.

A final Dense layer with softmax activation to output the class probabilities (10 possible digits).

3.Hyperparameter Tuning:

A set of hyperparameters is varied across experiments:

Learning rate: The step size used by the Adam optimizer during training, tested at values of 0.001 and 0.01.

Batch size: The number of samples processed in each training step, with values 32 and 64 tested.

Dropout rate: The probability of setting units to zero during training to prevent overfitting, tested at values of 0.2 and 0.3.

Hidden units: The number of neurons in the hidden layers, set to 128 for all experiments.

4.Model Compilation and Training:

For each combination of hyperparameters, the model is compiled using the Adam optimizer with the selected learning rate, and the sparse categorical cross-entropy loss function (suitable for multi-class classification). The model is trained using the fit() method for 5 epochs, with early stopping based on validation loss to prevent overfitting.

5.Performance Evaluation:

After training, the model is evaluated on the test set using the evaluate() method, which returns the test loss and test accuracy. Additionally, the training time for each experiment is recorded.

6.Results Recording:

The performance metrics for each experiment (test accuracy, test loss, final validation accuracy, final validation loss, and training time) are stored in a pandas DataFrame for further analysis.

7.Visualization and Analysis:

The results are visualized using matplotlib:

Validation accuracy and validation loss are plotted across different experiments to identify the most effective hyperparameter combinations.

Comparative bar charts are created to illustrate the relationship between training time, test accuracy, and test loss for different learning rates.

8.Comparative Analysis:

A final analysis is presented, summarizing the results in a table and discussing the trade-offs between the different hyperparameter configurations in terms of training time, accuracy, and overfitting

LITERATURE REVIEW

Once upon a time, in the early days of artificial intelligence, a small group of brilliant minds laid the foundations for what would become one of the most transformative fields in technology: **deep learning**. It all began in the 1940s with **McCulloch and Pitts**, who introduced the concept of a simple artificial neuron, inspired by the human brain. This idea sparked curiosity and further exploration into how machines could learn from data.

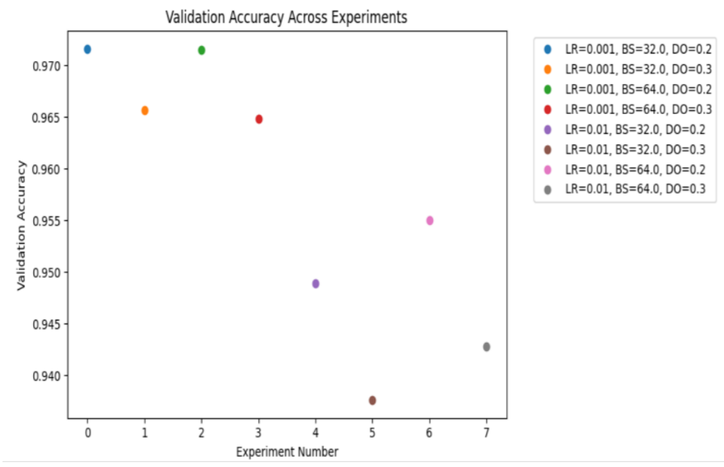
As the decades passed, **Frank Rosenblatt** created the **Perceptron** in 1958, an early form of neural networks that could classify patterns. However, it wasn't until the 1980s that deep learning truly began to take shape. **Geoffrey Hinton**, **Yann LeCun**, and **Yoshua Bengio**, often referred to as the "Godfathers of Deep Learning," revolutionized the field with their work on **backpropagation** and **deep neural networks**. LeCun’s **LeNet** architecture, developed in the 1990s, was the first to successfully use convolutional neural networks (CNNs) to recognize handwritten digits, leading to the creation of the iconic **MNIST dataset**—a collection of images of handwritten numbers that would go on to become a benchmark for machine learning models.

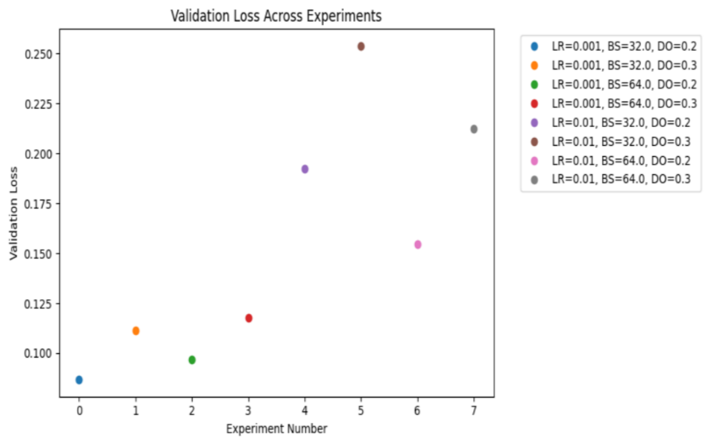
In the 2010s, **AlexNet** by **Alex Krizhevsky** broke new ground by winning the ImageNet competition using deep CNNs, showing the world that neural networks could achieve superhuman accuracy in image classification. This marked the beginning of the deep learning revolution. With the introduction of **dropout** and **early stopping** by researchers like **Geoffrey Hinton**, overfitting was minimized, making it easier to train large models on massive datasets.

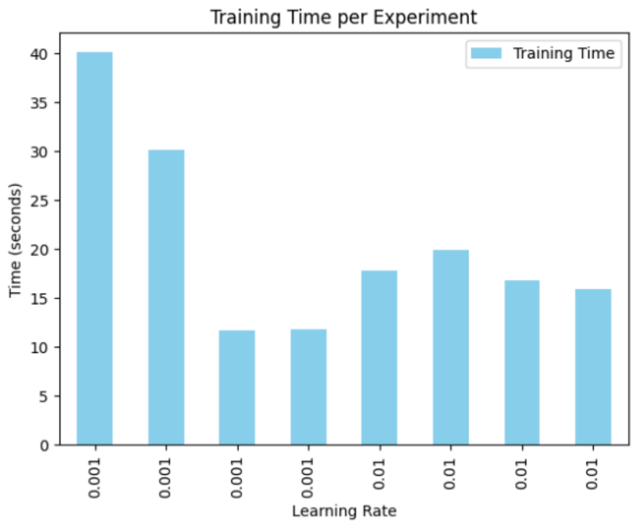
Today, deep learning is everywhere, from **Google’s DeepMind** mastering complex games like Go, to AI assistants like **OpenAI’s GPT** transforming natural language processing. Behind these innovations lie the same principles and tools used in your experiment: the **MNIST dataset**, **Adam optimizer**, **early stopping**, and **dropout** regularization. These techniques, pioneered by the best minds in AI, have made it possible for computers to learn and understand images, patterns, and even human language with incredible accuracy.

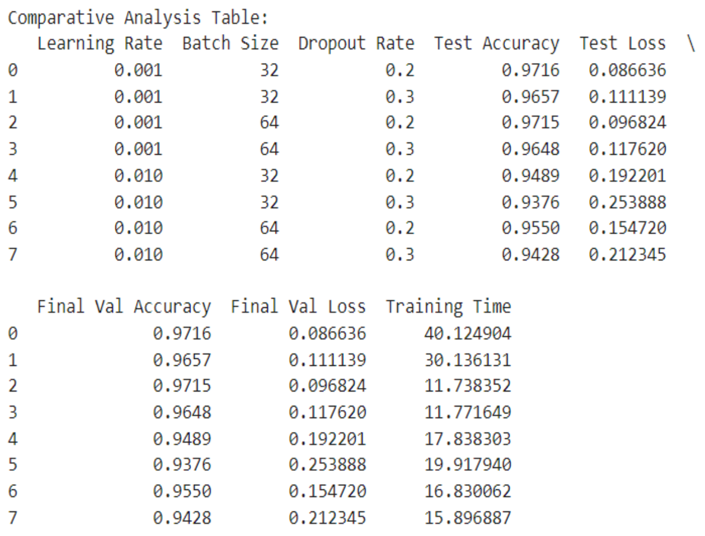
And so, the journey of neural networks continues, as researchers and engineers push the boundaries of what’s possible, shaping a future where AI plays an increasingly pivotal role in our lives.

RESULTS



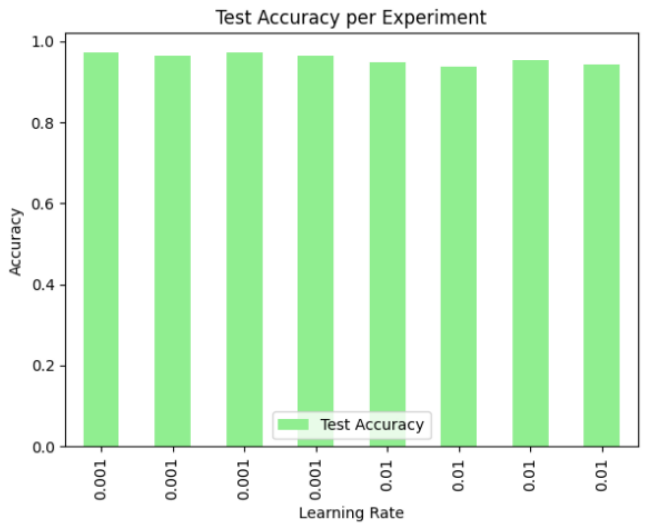


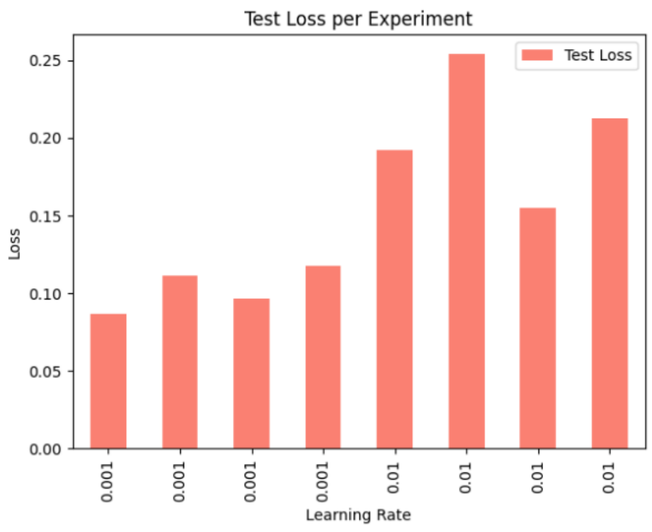




```
Comparative Analysis Table:
   Learning Rate  Batch Size  Dropout Rate  Test Accuracy  Test Loss  \
0          0.001          32           0.2         0.9716   0.086636
1          0.001          32           0.3         0.9657   0.111139
2          0.001          64           0.2         0.9715   0.096824
3          0.001          64           0.3         0.9648   0.117620
4          0.010          32           0.2         0.9489   0.192201
5          0.010          32           0.3         0.9376   0.253888
6          0.010          64           0.2         0.9550   0.154720
7          0.010          64           0.3         0.9428   0.212345

   Final Val Accuracy  Final Val Loss  Training Time
0              0.9716        0.086636      40.124904
1              0.9657        0.111139      30.136131
2              0.9715        0.096824      11.738352
3              0.9648        0.117620      11.771649
4              0.9489        0.192201      17.838303
5              0.9376        0.253888      19.917940
6              0.9550        0.154720      16.830062
7              0.9428        0.212345      15.896887
```





DISCUSSIONS

The code is designed to perform hyperparameter tuning and evaluate the performance of a neural network model on the MNIST dataset. It begins by loading and normalizing the dataset, then constructs a model with two hidden layers, dropout regularization, and a softmax output for multi-class classification. The run_experiments function conducts multiple experiments, testing different combinations of learning rates, batch sizes, and dropout rates, and records key metrics such as test accuracy, test loss, final validation accuracy, and training time. Early stopping is used to halt training if the validation loss does not improve. The results are visualized using scatter plots for validation accuracy and loss, as well as bar charts comparing training time, test accuracy, and test loss. Finally, a comparative analysis table is printed to summarize the experiment outcomes. This framework offers insights into how different hyperparameters affect model performance, highlighting the trade-offs between accuracy and computational cost, and provides a foundation for further optimization and experimentation with more advanced techniques like grid search or random search for hyperparameter tuning.

CONCLUSION

In conclusion, the code provides a comprehensive framework for evaluating and tuning the hyperparameters of a neural network model on the MNIST dataset. By experimenting with different learning rates, batch sizes, and dropout rates, the code enables a detailed analysis of how these hyperparameters influence the model's performance in terms of accuracy, loss, and training time. The use of early stopping helps to prevent overfitting and saves training time, while the visualizations of validation accuracy, loss, and training time provide an intuitive way to compare the results across experiments. The tabular summary of results further enhances the interpretability of the experiments, allowing for a clear understanding of which configurations work best. Overall, this framework offers a valuable tool for hyperparameter tuning, model evaluation, and comparison, helping users identify optimal configurations for their machine learning tasks.

FUTURE SCOPE

For future improvements, the code can be extended to include a wider range of hyperparameters, such as varying the number of hidden layers, the type of activation functions, or the optimizer used (e.g., SGD, RMSprop, or AdamW). This would provide a more comprehensive exploration of the hyperparameter space and potentially lead to better-performing models. Additionally, more advanced hyperparameter search techniques, like grid search, random search, or Bayesian optimization, could be employed to automate and optimize the search for the best hyperparameters more efficiently. The current model could also benefit from incorporating other regularization techniques like L2 regularization or batch normalization to improve generalization and training stability. Furthermore, extending this framework to larger and more complex datasets, or even exploring other architectures (such as convolutional neural networks for image classification), could make the approach applicable to a wider range of problems. Finally, integrating automated tools such as Keras Tuner or Optuna could facilitate a more systematic and scalable approach to hyperparameter optimization, particularly for larger datasets and more complex models.

REFERENCES

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. - Fundamental textbook covering neural networks, optimization, and regularization techniques like dropout.

2. Srivastava, N., et al. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 15(1), 1929–1958. - Introduces dropout as a regularization method for deep learning.

3.Wilson, J., & Ho, J. (2017). The Marginal Value of Adaptive Methods in Deep Learning. NeurIPS 2017. - Discusses the impact of adaptive learning rates (e.g., Adam) on training efficiency and generalization.

4. Keskar, N. S., et al. (2016). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. ICML 2016. - Investigates the effect of large batch sizes on model generalization.

5.LeCun, Y., & Cortes, C. (2010). MNIST Handwritten Digit Database. [Link](http://yann.lecun.com/exdb/mnist/) - The original MNIST dataset for benchmarking machine learning models.

6.Chollet, F. (2017). Deep Learning with Python. Manning Publications. - Practical guide to implementing deep learning models using Keras and TensorFlow.

7.TensorFlow Keras API. TensorFlow Documentation. [Link](https://www.tensorflow.org/api_docs/python/tf/keras) - Official documentation for building and tuning neural networks in TensorFlow/Keras.

8. ChatGPT