The Role of Hyperparameter Optimization in Enhancing MNIST Model Performance

- Thabang Mokoena

- CSC3022S (Machine Learning)
- 15/04/24 – 22/04/24 (Experiment Date)

Aim:

This study explores the optimization of neural network architectures for improving digit recognition accuracy on the widely-used MNIST dataset. Our primary aim was to evaluate how various configurations and hyperparameters impact the performance of neural networks in image classification tasks. I focused on several key design choices, including the number of layers, the number of neurons per layer, the type of activation functions.

1. Introduction

- **Background Information**

  The MNIST dataset, short for the Modified National Institute of Standards and Technology dataset, is one of the most extensively used datasets for benchmarking machine learning algorithms. Composed of 70,000 images of handwritten digits from 0 to 9, the dataset is split into a training set of 60,000 images and a test set of 10,000 images. Each image is a 28x28 pixel grayscale representation of a digit, designed to allow for straightforward processing while still presenting challenges typical of real-world data. Since its creation, MNIST has become a fundamental resource for researchers testing new machine learning models, particularly in the field of image recognition.

- **Objectives**

  The primary objective is to experiment with different neural network architectures and training methodologies to enhance the accuracy and efficiency of digit recognition systems on the MNIST dataset. This involves assessing various configurations of layers, neurons, and activation functions.

2. Topology of the Networks

- **Layer Structure**

  The experiments conducted involved testing multiple neural network architectures to determine optimal configurations for digit recognition on the MNIST dataset. Initial models began with simpler structures, comprising a few layers to establish baseline performance. For instance, one

model utilized a basic three-layer architecture consisting of an input layer, a single hidden layer with 256 neurons, and an output layer. This approach was chosen to understand the minimal complexity required to achieve reasonable accuracy.

Despite the capability of deeper networks to handle more complex information through additional layers, for the MNIST dataset, extensive layering is often unnecessary. The dataset comprises relatively simple patterns (handwritten digits), which means that adding many hidden layers may lead to diminishing returns in performance improvement and increased computational costs. In fact, very high accuracy rates can often be achieved with just one hidden layer, indicating that while deeper networks are capable of capturing more complex hierarchical patterns, the simplicity of the MNIST digits does not require this capability for effective performance.

- **Activation Functions**

  In the hidden layers, the **Rectified Linear Unit (ReLU)** was the primary activation function used due to its ability to mitigate the vanishing gradient problem commonly encountered with traditional sigmoid activation functions. ReLU is favored in deep learning architectures because it introduces non-linearity without affecting the gradients significantly, thereby supporting efficient training of deeper models.

  For the output layer, although the softmax function is typically described as part of the activation, in practice, it is combined with the cross-entropy loss during training. This integration is efficient as the softmax function converts logits into probabilities by normalizing the output of the network, making it suitable for multi-class classification problems like MNIST. The cross-entropy loss function then measures the difference between the predicted probabilities and the actual distribution (one-hot encoded labels), making it highly effective for training classification models where outputs are probabilities corresponding to class membership.

- **Network Connectivity**

  The decision to use a fully connected (dense) network architecture was guided by the nature of the MNIST dataset and the objectives of the study. Fully connected layers are advantageous in scenarios where the relationship between all input features needs to be captured to make accurate predictions. In the context of digit recognition, each pixel in an image can contain important information about the digit's structure that might be lost if not considered in conjunction with other pixels.

  Dense networks ensure that every neuron in one layer connects to every neuron in the subsequent layer, maximizing the network's ability to learn intricate patterns from the data. This architecture choice is particularly effective for the MNIST dataset, where the spatial arrangement and relationship between pixels are crucial for recognizing handwritten digits. However, it is also acknowledged that fully connected networks can be prone to overfitting, especially as the complexity (number of layers and neurons) increases. Therefore, careful consideration was given to balance the network's depth and breadth to optimize performance without unnecessary computation overhead or overfitting.

3. Data Preprocessing

- **Normalization**

Pixel values of MNIST images are scaled from 0 to 255 to a range of 0 to 1 using transforms.ToTensor(), which also converts them to tensors. This standardizes input data, aiding in faster and more stable network training.

- **Transformation**

  Further standardization is achieved with **transforms.Normalize((0.5,), (0.5,)),** adjusting pixel values to have a mean of zero and standard deviation of one. This helps in normalizing the data distribution, enhancing model learning efficiency.

- **Data Splitting**

  The MNIST dataset is split into training (60,000 images) and testing (10,000 images) sets. Additionally, 80% of the training set is reserved as a validation set using random_split to help tune and validate the model during training.

  The DataLoader is used to efficiently load and batch**(64 Size)** the data during training. Batching helps in managing memory usage and can also impact the training process by providing a more generalized gradient update.

4. Loss Function(s):

- **Choice of Loss Function**

  For your project, you've employed the **Cross-Entropy Loss** function. This loss function is particularly suited for classification tasks that involve multiple classes, like digit recognition with the MNIST dataset. Cross-Entropy Loss measures the performance of a classification model whose output is a probability value between 0 and 1. It provides a low score for diverging predictions and a high score for accurate predictions, effectively quantifying how different the predicted probability distribution is from the actual distribution (the label). It's especially effective because it accentuates differences in the probability distribution, pushing the model toward precise probabilities, rather than just being satisfied with the correct class.

5.Optimizer(s)

- **Optimizer Choices**

  The **Adam optimizer** was selected for training the neural network. Adam stands for Adaptive Moment Estimation; it combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Adam is known for its effectiveness in cases where large amounts of data and parameters are involved, primarily due to its use of adaptive learning rate techniques. Unlike standard stochastic gradient descent, which maintains a single learning rate for all weight updates, Adam maintains a learning rate for each network weight (parameter) and adapts these rates based on how frequently a parameter gets updated during training. This feature helps in achieving faster convergence and requires less tuning of the learning rate.

Parameter Tuning: The initial learning rate for the Adam optimizer was set to 0.001. This is a typical starting value that offers a balance between speed and accuracy in the convergence process. During preliminary experiments, this learning rate was adjusted based on the observed performance on the validation set. If the validation loss failed to decrease, or if training showed signs of stalling, the learning rate might have been reduced to refine the training steps. Adjusting the learning rate based on validation performance is a crucial strategy to avoid overfitting and to ensure that the model generalizes well to new, unseen data.

6. Training and Validation

Epochs and Batch Size: The models are set to train for a total of 15 epochs using a batch size of 64. This setup helps in balancing training efficiency and computational resource utilization.

Training Loop: Each epoch iterates over the training dataset using a DataLoader that shuffles the data, ensuring each batch is different, which helps prevent the model from memorizing the order of the data.

Loss Calculation and Backpropagation: During each batch processing, the model predicts outputs based on the input data, and the loss is computed using the Cross-Entropy Loss function. The gradients are then propagated back through the network (backpropagation) to update the model parameters using the Adam optimizer.

- **Validation Techniques**

  Periodic Evaluation: The validation occurs after each training epoch. The model is evaluated on a held-out validation set that was randomly split from the original training data (80% of the initial dataset).

  No Training Involved: During validation, the model is set to evaluation mode (model.eval()), which disables dropout or batch normalization layers that might affect the deterministic outcome of the model.

  Loss and Accuracy Computation: For each batch in the validation set, the model predicts and computes the loss and accuracy. These metrics are averaged over all batches to determine the overall validation loss and accuracy per epoch.

- **Performance Metrics**

  Accuracy and Validation Loss: These metrics are crucial in monitoring the model's performance. The validation loss provides insight into how well the model is generalizing, while the accuracy metric tells how often the model predicts the correct labels.

7. Evaluation of Architectures and Hyperparameters

- ### Comparative Analysis

  Our examination of different architectures revealed that the MNIST dataset's simplicity allows for high accuracy with a less complex model. Rather than adding many hidden layers, which can lead to overfitting and increased computational load, we found that a modest architecture was sufficient to achieve excellent performance, demonstrating that extra layers were not necessary for this particular task.

- ### Hyperparameter Optimization

During the optimization process, learning rates and neuron counts were meticulously adjusted. While high learning rates led to faster convergence, they were often unstable. Conversely, smaller rates were stable but slow. A learning rate of 0.001 was selected for optimal stability and convergence speed. The exact effect of neuron count per layer was nuanced, but our strategy was to choose a number that offered adequate capacity without excess, aligning with the dataset's complexity, So for the final model I chose to have two hidden layers of 256,128 neurons respectively.

- **Activation Function Choice**

  For the hidden layers, the ReLU activation function was employed and outperformed the sigmoid function. The advantage of ReLU lies in its simplicity—maintaining a constant gradient of 1 for positive values—which mitigates the vanishing gradient issue commonly faced with sigmoid functions. This allows for efficient training of deeper networks since the gradients are less likely to diminish through layers.

- **Best Performing Model**

  The model that delivered the best results on the MNIST dataset employed two hidden layers with 256 and 128 neurons, respectively, and used the ReLU activation function. The Adam optimizer was chosen for its adaptive learning rate, enhancing the training process without the stability issues of larger learning rates. With a learning rate set to 0.0001, the model achieved a balance between efficient training and high accuracy, converging effectively to a robust solution for digit recognition.

8. Conclusion

My exploration into neural network configurations for the MNIST dataset revealed that simplicity often trumps complexity. The ReLU activation function and a careful selection of learning rate and optimizer were instrumental in constructing a model that is both accurate and efficient. The study's results emphasize the effectiveness of these choices for digit recognition tasks, underscoring the balance between network design and generalization.

References

1. LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). *The MNIST Database of Handwritten Digits*. Retrieved from http://yann.lecun.com/exdb/mnist/
2. Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. arXiv preprint arXiv:1412.6980.
3. Nair, V., & Hinton, G. E. (2010). *Rectified linear units improve restricted boltzmann machines*. In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807-814).