CS587 - Neural Networks & Learning of Hierarchical Representation

Spring Semester 2024
Assignment 3

Papageridis Vasileios - 4710
csd4710@csd.uoc.gr

April 26, 2024

Introduction

In this assignment our goal is to implement, train and test a Multi-Layer Feed Forward Neural Network in order to tackle the task of image classification. We are going to use the Stochastic Gradient Descent (SGD) algorithm and automatic differentiation based on PyTorch framework. The dataset that we are going to use is the Digit dataset. This dataset is made up by 1797 images of handwritten digits from 0 to 9. Each image has 8x8 dimensions and each class consists of approximately 180 samples.
In the last part of the assignment we are trying to maximize the performance on the dataset. To achieve this goal, we define a new architecture of a Neural Network and we will provide the different settings used in order to end up with the final model and hyperparameters.

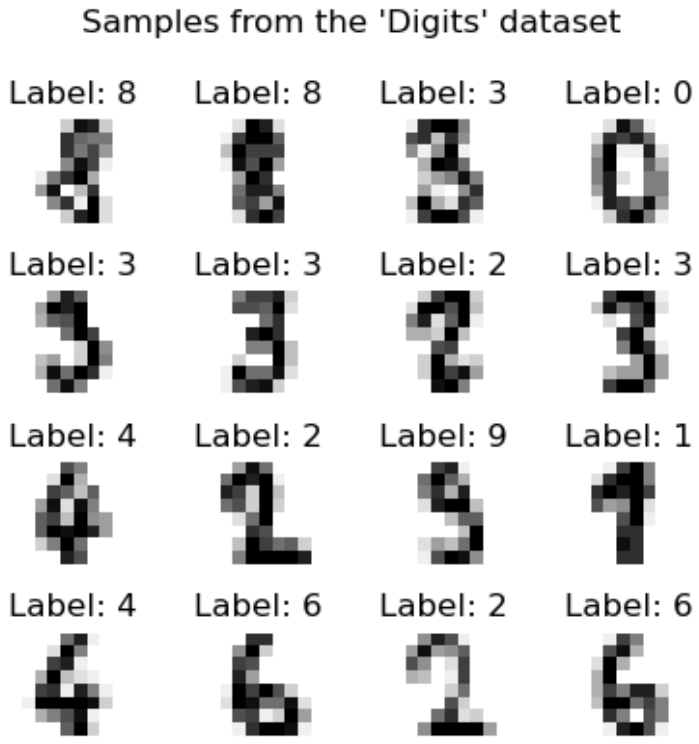


Figure 1: Sample images from the Digits dataset, depicting the ten different classes/digits.

Dataset Analysis and Visualization

The training set comprises 1,500 samples, each represented by a 64-dimensional feature vector. These dimensions correspond to the pixel values of 8x8 images of the digits, flattened into a single array. The labels are integers from 0 to 9, matching the digit depicted in each image.

```
========== Train Set ==========
Number of samples: 1500
X: torch.Size([1500, 64])   |   y: torch.Size([1500])
```

The test set, designed to evaluate model performance, contains an equal distribution of digit classes to prevent class imbalance from biasing our results.

Data Visualization

Histograms of the labels in both the training and test sets show a uniform distribution across the different digit classes, ensuring that our models are exposed to and tested on an equal number of examples from each class.

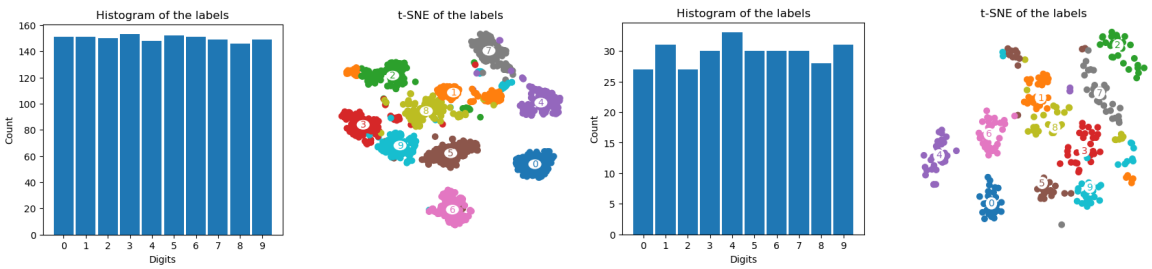


Figure 2: Label distribution in the train (left) and test (right) sets.

Additionally, a t-Distributed Stochastic Neighbor Embedding (t-SNE) visualization reveals the natural clustering of different digit classes in the feature space, providing insights into the complexity and separability of the classes.

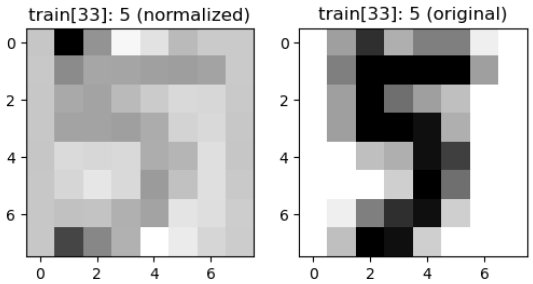


Figure 3: Visualization of the 33rd sample in normalized (left) and original (right) form.

Neural Network Model and Training

In this section we are going to explain the foundations of the assignment in order to construct the Neural Network and to understand the method we need to implement and use.

Architecture

The input layer is fully connected to a hidden layer with size$hid_size = 15$). The hidden layer transforms the inputs using a set of weights W_h and biases b_h, and applies a non-linear activation function. We will use the Sigmoid function for activation, which introduces non-linearity and allows the network to capture complex patterns:

$$h(x) = \frac{1}{1 + e^{-x}}. \tag{1}$$

The output layer produces a set of scores for each class $j \in \{0, \ldots, 9\}$, computed as:

$$z^o(x) = W_o h(x) + b_o, \tag{2}$$

where $h(x)$ denotes the activations from the hidden layer, and W_o, b_o are the weights and biases associated with the output layer, respectively.

Softmax Cross-Entropy Loss

The scores from the output layer are passed through a softmax function to obtain a probability distribution over the classes. The softmax function is defined as:

$$o_j = \frac{e^{z_j^o}}{\sum_k e^{z_k^o}}, \tag{3}$$

where o_j represents the probability that the input belongs to class j, and the denominator normalizes these probabilities.
To measure the performance of the model, we use the softmax cross-entropy loss function, which quantifies the difference between the predicted probabilities and the true distribution. The true class labels are encoded using one-hot encoding, where $t_j = 1$ if j is the correct class, and 0 otherwise. The cross-entropy loss for a single sample is given by:

$$CE(t, o) = -\sum_j t_j \log(o_j), \tag{4}$$

and the overall loss across all samples in a training batch is:

$$loss = \sum_{i=1}^{N} CE(t^{(i)}, o^{(i)}), \tag{5}$$

where N is the number of samples in the batch.

Stochastic Gradient Descent

The model is trained using the Stochastic Gradient Descent (SGD) method, which updates the model parameters iteratively based on the gradient of the loss with respect to each parameter.

Model Analysis and Interpretation

After constructing the neural network as described in the assignment, the network was trained using the Stochastic Gradient Descent algorithm. The following observations are made upon analyzing the training dynamics and the network's performance:

Model Architecture Visualization

The architecture of the neural network is illustrated in Figure 7, which provides the graph of the model using Tensorboard. We can clearly observe data flow from the input layer through the hidden layer and finally to the output layer.

Numerical Analysis of Model Performance

The network was initialized with an input layer size of input_size $= 64$, a hidden layer size hid_size $= 15$, and an output layer size output_size $= 10$. It was trained across num_epochs $= 10$ with a batch_size $= 32$ and a learning rate $\alpha = 0.01$.

- The training accuracy increased from 16.9% to 58.9%, and the test accuracy improved from 17.8% to 55.6% over the 10 epochs. This indicates that the model was learning and generalizing to unseen data, but with a slight difference between training and test performance.
- The gradual convergence of the loss function, as seen in Figure 4, and the monotonic increase in accuracy, as seen in Figure 11, suggest that the chosen hyperparameters facilitated effective learning without signs of overfitting or catastrophic forgetting.

Analysis of Training Progression

A closer examination of the weight histogram and loss and accuracy plots yields further insights:

- **Weight Histogram**: Figure 6 shows the distribution of the learned weights W_h. A wide and diverse distribution of weights typically indicates that the network is leveraging a broad "spectrum" of features to make decisions, rather than relying on a limited set of input patterns.
- **Training Loss Plot**: The plot in Figure 4 demonstrates a typical descent, starting with high variance in the loss reductions which gradually stabilizes as the epochs progress. This behavior is characteristic of the SGD optimizer's operation.
- **Accuracy Plot**: As shown in Figure 11, the accuracy on both the training (blue) and test (red) sets show an asymptotic approach towards higher values. The non-divergent nature of these plots indicates that the model has not yet reached its capacity and suggests that either additional epochs or further architectural optimizations may yield improved performance.

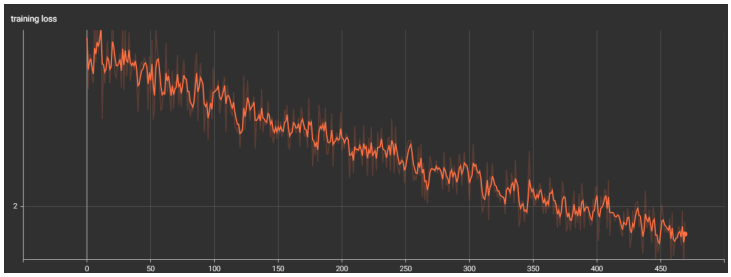


Figure 4: The plot of the training loss over time. The loss decreases, reflecting the learning process of the network.

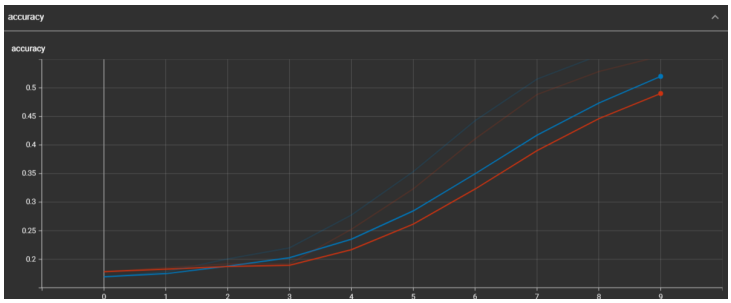


Figure 5: Accuracy on the training and test sets over epochs. The blue line represents training accuracy, while the red line represents test accuracy.

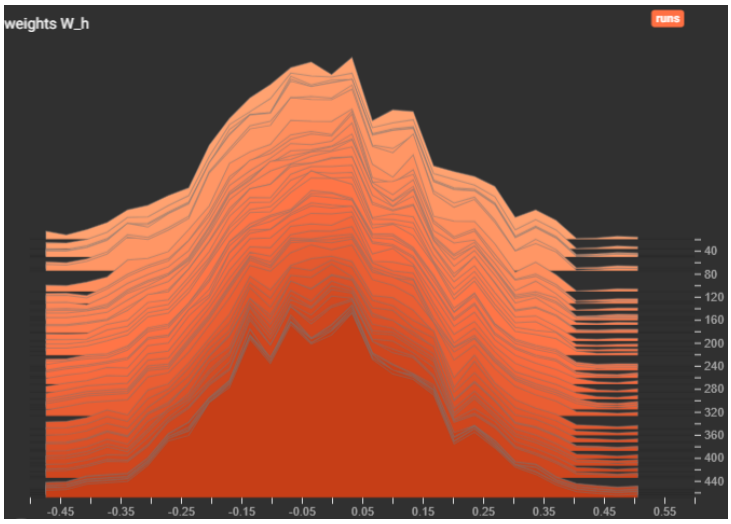


Figure 6: Histogram of the weights W_h of the hidden layer.

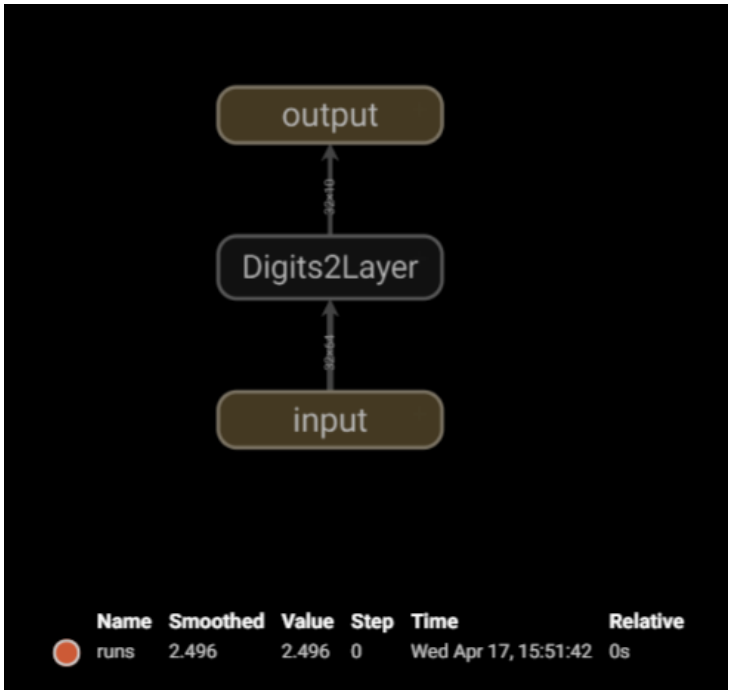


Figure 7: The graph of the neural network model showing the data flow from the input to the output through the Digits2Layer.

Theoretical Questions

(a) Sigmoid Activation Function

The sigmoid activation function, denoted as $\sigma(x)$, is a nonlinear function that maps any real-valued number into the range (0, 1), defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{6}$$

It is commonly used in neural networks to introduce nonlinearity and to model probability-like outputs. The sigmoid function has historically been used widely in binary classification problems.

Advantages:

- Smooth gradient, preventing ”jumps” in output values.
- Output values bound between 0 and 1, representing probabilities.
- Easy to understand and implement.

Disadvantages:

- Prone to vanishing gradient problem, where gradients become very small for large positive or negative inputs, slowing down learning.
- Outputs are not zero-centered which can make the optimization harder.
- Computationally expensive due to the exponential function.

(b) Softmax Cross Entropy Loss Function

The Softmax Cross Entropy loss, often simply referred to as cross-entropy loss, combines the Softmax activation and the Cross Entropy loss into a single function. This loss function is typically used in multi-class classification problems. The softmax function converts the raw score outputs of a network into a probability distribution:

$$o_j = \frac{e^{z_j}}{\sum_k e^{z_k}}, \tag{7}$$

where o_j is the predicted probability of the j-th class, z_j is the raw score for the j-th class, and the denominator is the sum of the exponentiated scores for all classes.
The cross-entropy loss is then computed as the negative log probability of the correct class:

$$CE(t, o) = -\sum_j t_j \log(o_j), \tag{8}$$

where t_j is the target probability for class j, which is 1 for the correct class and 0 for all others in one-hot encoding.
This loss function is advantageous as it penalizes incorrect class predictions with a large margin, and its gradient is proportional to the error in prediction, which contributes to faster learning.

Hyperparameter Tuning and Model Evaluation

We used grid-search to optimize hyperparameters for several neural network architectures. The hyperparameter space explored included the following:

- learning rates: 0.001, 0.01, and 0.1
- batch sizes: 32, 64, and 128
- hidden layer sizes: (10, 5), (20, 10), and (25, 15) and
- epoch counts 10, 50, and 150

Model Comparisons

The `EnhancedDigitsModel` provided a standard architecture for comparison. The `ConfigurableDigitsModel` offered flexibility in terms of layer sizes. Models employing dropout, like `MLPwithBatchRELUDropout`, introduced regularization that helped mitigate overfitting, evident in their performance gains. `DeeperMLP` and `WiderMLP` explored the effects of increasing depth and width, respectively, which tend to improve the model's capacity. The use of alternative activation functions, as seen in `MLPWithELU`, and normalization techniques in `MLPWithBatchNorm`, showcased the impact of these modifications on model convergence and accuracy.

Model	LR	Batch	Hidden	Train Acc.	Test Acc.
EnhancedDigitsModel	0.1	32	(25, 15)	0.9913	0.9125
ConfigurableDigitsModel	0.1	64	(25, 15)	1.0000	0.9259
MLPwithBatchRELUDropout	0.01	32	(25, 15)	0.9960	0.9192
DeeperMLP	0.1	64	(20, 10)	1.0000	0.9327
WiderMLP	0.1	32	(10, 5)	1.0000	0.9226
MLPWithELU	0.1	32	(20, 10)	1.0000	0.9125
MLPWithDropout	**0.1**	**32**	**(20, 15)**	**1.0000**	**0.9460**
MLPWithBatchNorm	0.1	32	(10, 5)	1.0000	0.9293

Table 1: Comparison of Neural Network Architectures

Optimal Model Configuration

The `MLPWithDropout` model emerged as the top performer, achieving the highest test accuracy of 94.60%. It effectively leveraged dropout regularization to combat overfitting, as indicated by its high accuracy on unseen data. The best parameters for this model were:

- learning rate: 0.1
- batch size: 32
- hidden layer sizes: (20, 15)

Analysis of Best Performing Model

The superior performance of the `MLPWithDropout` model can be attributed to its architecture, which incorporates dropout after both the first and second layers. This dropout rate of 0.5 during training appears to be an optimal choice in this context, effectively reducing overfitting while still allowing the network to learn complex patterns from the data.

MLPWithDropout Architecture

The `MLPWithDropout` model is a neural network designed to balance the model's capacity to learn from complex data against the risks of overfitting. The architecture comprises the following layers:

- **Input Layer**: A fully connected linear layer with 256 neurons.
- **Rectified Linear Unit (ReLU)**: Following the linear transformation, a ReLU activation function introduces non-linearity to the model, allowing for the learning of complex patterns. Its advantage over other activation functions is that it mitigates the vanishing gradient problem, promoting faster learning.
- **Dropout**: A dropout layer with a rate of 0.5 follows the ReLU activation. This rate was chosen as it effectively halves the number of active neurons in the layer during training, which prevents co-adaptation of features and encourages the network to develop a more robust representation of the data.

- **Hidden Layer and Depth**: An additional hidden layer with 128 neurons to increase the depth and the learning capacity of the model.
- **Output Layer**: The final layer is another linear transformation that maps the features learned by the network to the output classes. Since this is a classification task, the size of the output layer corresponds to the number of classes in the dataset.

Model Architecture Visualization

The architecture of MLPWithDropout, is illustrated in Figure 8.

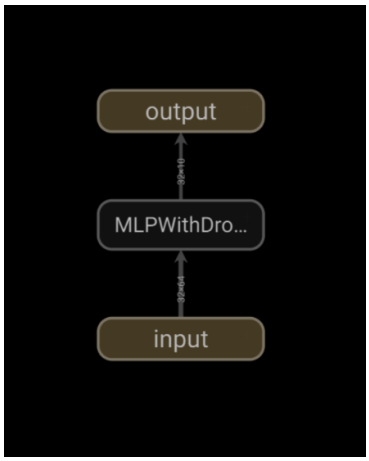
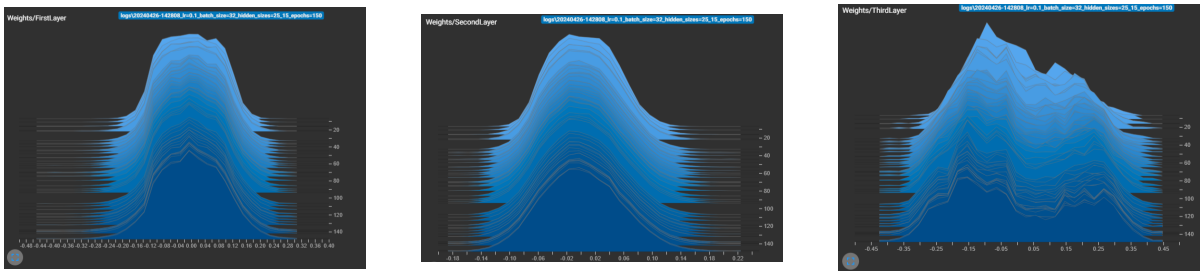


Figure 8: TensorBoard visualization of the MLPWithDropout model.

Observations

The graph provides a high-level overview of the model's structure, indicating the pathway from input to output through the intermediary computational unit, labeled as 'MLPWithDropout'. Despite the minimalist representation, the TensorBoard graph confirms the successful implementation of the intended model design.

Weight Distribution Analysis



(a) First Layer Weights **(b)** Second Layer Weights **(c)** Third Layer Weights

Figure 9: Distributions of the weights across different layers of the network.

In Figure 9, the first layer's weights exhibit a symmetric distribution around zero, which is desirable as it indicates that the network's initial learning is unbiased towards any particular direction of the feature space.

The weights of the second layer, demonstrate a narrower spread compared to the first layer. This is the beginning to form more definitive opinions on the correlations between the features and the outputs.
Lastly, the third layer's weights show a diverse range of values, suggesting that this layer is actively involved in making decisions based on the combinations of features processed by earlier layers.

Observations

Evenly distributed weights around zero in initial layers imply a good initialization that supports balanced learning. As we move deeper into the network, the divergence in the weight values can often reflect a more complex abstraction of the data.

Training Performance Analysis

The training process of the MLPWithDropout model is analyzed by observing the training loss and accuracy over epochs.

Training Loss

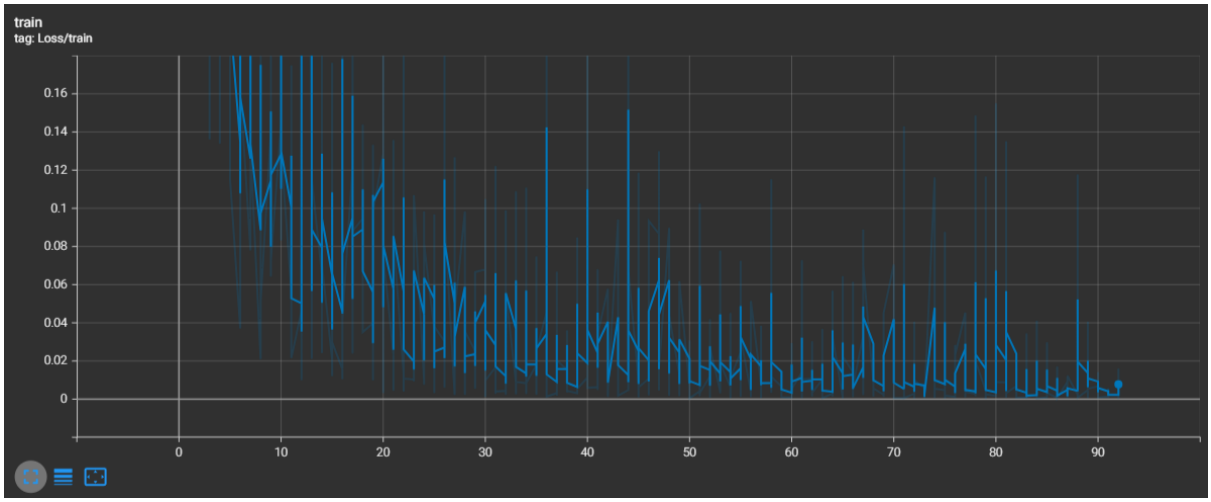


Figure 10: Training loss of the MLPWithDropout model over epochs.

Figure 10 shows the training loss over time. The decreasing trend indicates learning and convergence, while the presence of spikes suggests sensitivity to certain batches of data or learning rate adjustments.

Accuracy

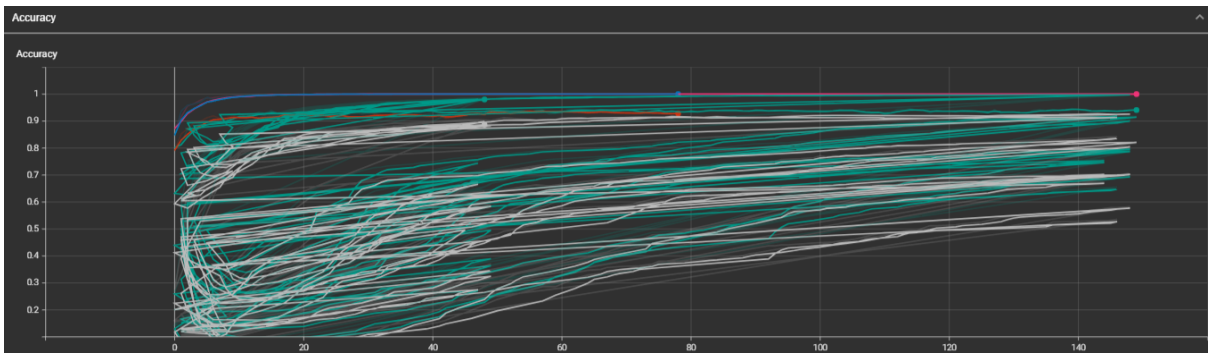


Figure 11: Training and validation accuracy of the MLPWithDropout model.

In Figure 11, we observe the training and validation accuracy over time. The accuracy curves demonstrate an upward trend, suggesting that the model is effectively learning from the training data and is capable of generalizing from what it has learned when evaluated on the validation set. The saturation of accuracy at high values is indicative of the model reaching its learning capacity.

Conclusion

The hyperparameter tuning process underscored the effectiveness of dropout in enhancing the generalization of neural network models. The `MLPWithDropout` model's optimal configuration of learning rate, batch size, and hidden layer sizes led to the highest test accuracy, outperforming other configurations.

References

[1] Simon J.D. Prince, *Understanding Deep Learning*, MIT Press, 2023, https://udlbook.github.io/udlbook/.

[2] Sebastian Nowozin and Christoph H. Lampert, *Structured Learning and Prediction in Computer Vision*, 2012, https://www.nowozin.net/sebastian/cvpr2012tutorial/.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org/.

[4] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[5] Michael A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015, http://neuralnetworksanddeeplearning.com/.