Optical Character Recognition applied to MNIST dataset

Lilian Rouzaire
Statistical Machine Learning - Project

I. Introduction

In the field of machine learning and computer vision, one of the most fundamental and historically significant tasks is the classification of handwritten digits. The MNIST [4] (Modified National Institute of Standards and Technology) database has played a important role in this domain, serving as a benchmark dataset for evaluating the performance of various image processing systems. Classifying this 10-class dataset of handwritten digits is indeed an easy job in comparison with broader classification tasks, such as labelling the vast ImageNet [3] database. Still, the core techniques apply in both cases and therefore, classifying MNIST is an interesting stepping stone to more complex works.

The MNIST dataset, introduced in 1998 by LeCun et al. [4], has become a basis in the field of machine learning. It consists of a collection of grayscale images. Each image is a representation of handwritten digits from 0 to 9. The simplicity of the dataset, along with its reasonable size, makes it ideal for testing and comparing various network architectures.

The significance of the MNIST dataset lies not only in its utility as a benchmark but also in its historical context. It was one of the first datasets to be widely used for training and testing in the field of machine learning, and it has contributed significantly to the development and validation of various techniques in image processing and classification. From simple linear classifiers like logistic regression to more complex models such as convolutional neural networks (CNNs [2]), the MNIST dataset has been a baseline in demonstrating the efficiency and performance differences of these methods.

This project aims to explore and compare the performances of different types of neural networks in classifying the images of the MNIST dataset. By examining various architectures, mostly traditional fully connected networks, this study seeks to understand to which extent varying the parameters of different models can help or not to process the features of handwritten digits and how these approaches affect classification accuracy, with a focus on detecting and avoiding overfitting and underfitting issues.

II. Methodology and approach

A. Dataset description

The dataset we will work on is the MNIST dataset. It is composed of 70'000 1-channel images of 28x28 pixels. The dataset is divided into 60'000 train samples and 10'000 evaluation samples. For the sake of simplicity, the evaluation of the models during training and the accuracy test afterwards are both performed on this evaluation dataset of 10'000 images. In the context of real research, it would be a bad practice to merge the evaluation and the test dataset, because the accuracy results would therefore be biased and would give a false impression of goodness. However, as this work focuses on controlling overfitting and underfitting during training, the cost of testing the models on unseen data is not worth the unbiasness.

Each image is associated to a class between 0 and 9. In our implementation, this class is represented by a one-hot vector of size 10, and the corresponding label is the argmax of this one-hot vector.

The MNIST dataset is acquired from Pytorch utils. Pixels are grayscale and are encoded with dynamic range [0-255]. Figure 1 shows an example of such images.

For most networks, it is convenient to deal with 1-dimensional data, therefore the input tensors are flattened into 1x784 so that it matches the needs of the fully connected layers. Only convolutional networks will be given square input (because CNN use 2D spatial correlation through 2-dimensional convolution operators).

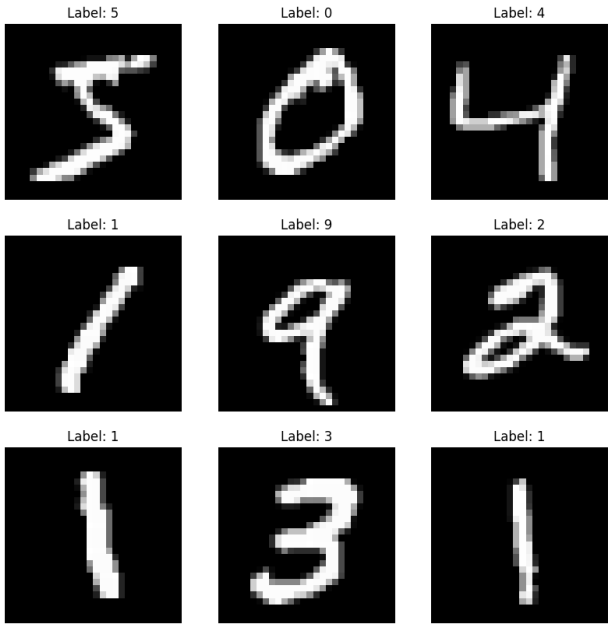


Fig. 1: Example of MNIST images

B. Chosen architectures

As stated in section I, the goal of this project is to setup a few networks, vary their parameters, and analyse their impact in terms of efficiency and CPU computation time. We will explore 3 different types of networks, from the most simple ones to more complex constructions : shallow networks, deep architectures, and finally, convolutional neural nets.

1) Shallow networks: Let's start with simple networks : a linear one, and a shallow one.

The linear network is composed of a unique linear layer, linking the 784 pixels of the image to a 10x1 vector representing one-hot labels.

The shallow network has one hidden layer, with variable number of neurons, set by default to 100. The non-linearity between the input layer and the hidden layer is ReLU [1], defined by $f(x) = \max(0, x)$.

In both cases, the output is normalized using the softmax function :

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

, where N is the vector length. This way, output values can be considered as probabilities, and the index highest probability within the 10x1 output vector is kept as the predicted class.

2) Deep networks: It is probable that linear and shallow networks will be limited in their capacity to infer the label of an image with so few layers. In order to overcome this issue, we construct a network that is deeper, with 3 hidden layers.

The number of neurons is variable in each layer and will be tuned. Each layer is followed by an activation function (ReLU), and the networks ends with a softmax normalization.

In part III-E, dropout will also be a added before each dense layer (but not represented on figure 2).

Figure 2 represents visually the deep network architecture.

C. Training phase of the models

Each one of the models will undergo the same training phase. The training loop is composed of a predefined number of epochs. Each epoch consists of the following :

- The train dataset is split into mini-batches of variable size (by default 100 samples).
- The forward pass of the model is computed : layers are applied one after the other.
- The loss value is computed. It is defined, in our case, by the cross-entropy loss function, to which we may add a regularisation term.
- The loss function is minimised during the backward pass, and the best coefficients found are propagated back into the model. In our case, the optimiser is the stochastic gradient descent.
- The last train loss is stored, as well as the test loss (over test data), and the accuracies.

The training phase ends after the last epoch.

All training processes are conducted on CPU, no GPU acceleration was setup for this work (there was no need given the depth of the networks).

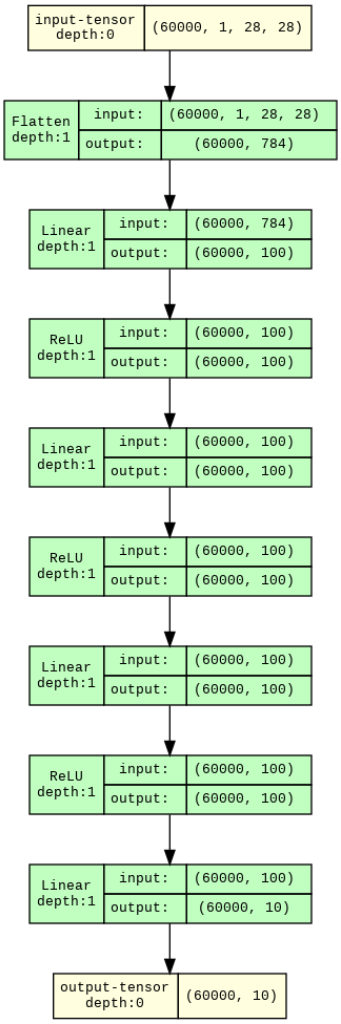


Fig. 2: Architecture of the deep network

III. Results

This section presents the results achieved by the networks presented in subsection II-B. Each subsection focuses on a specific implementation or parameter change. The results are presented without interpretation and will be discussed in section IV. The goal of this section is to test various modifications, mainly in order to see which ones will overfit/underfit. There are a lot of other tunable parameters to be tuned, but the selected ones are those which are most likely to create overfitting issues.

A. Shallow VS deep models

Our first experiments aims at comparing elementary networks with deeper ones, keeping all parameters fixed.

At a first run, let's evaluate the performances of the linear model. This model is trained over 100 epochs, without any form of regularisation, with a learning rate $\eta = 0.1$, and mini-batches of 100 samples.

The considered metrics are the train loss, the test loss, and the accuracy on test dataset evaluated during training. Figure 3 shows the evolution of these metrics during the training phase.

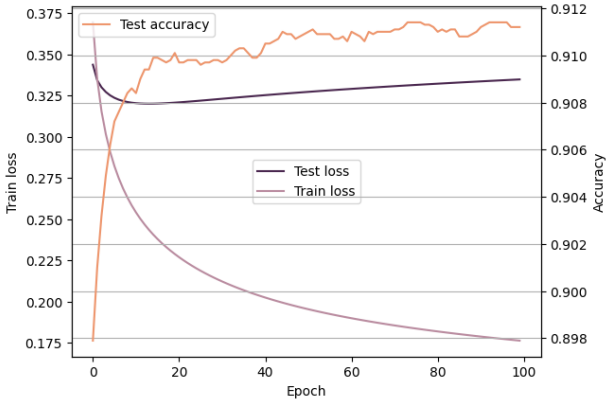


Fig. 3: Evolution of train loss, test loss and accuracy during training phase of a linear model

The shallow model is trained under the same conditions. Figure 4 presents the evolution of the same metrics. The reader may notice that the accuracy and loss scales have changed. In particular, the highest accuracy is now 98% and train loss tends to zero.

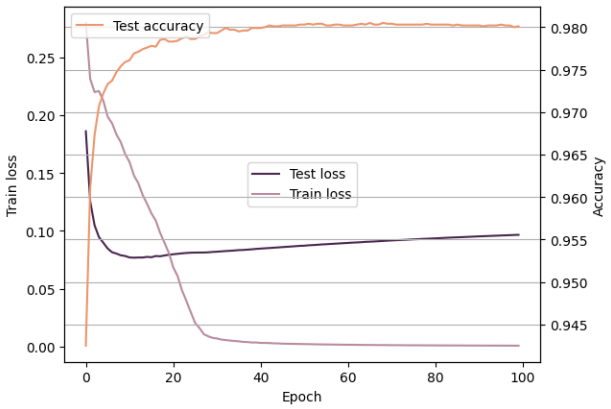


Fig. 4: Evolution of train loss, test loss and accuracy during training phase of a shallow model

As a last comparison, the deep model is trained, still under the same conditions. Results are presented in figure 5.

B. Impact of L2 regularisation

Previously, the models training was conducted without any form of regularisation. As this can lead to overfitting, we explore in this section the difference of performances with and without the L2 penalty. This penalty consists of adding a term to the cross-entropy loss function. This term is defined as the squared magnitude of the parameters :

$$\mathrm{L2}(\lambda) = \lambda \sum_{i=1}^{p} \alpha_i^2 \tag{1}$$

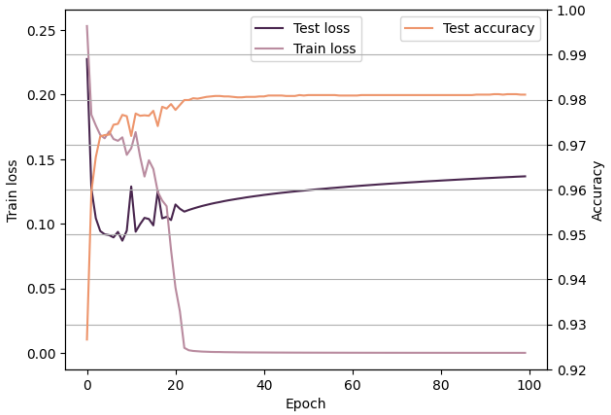


Fig. 5: Evolution of train loss, test loss and accuracy during training phase of a deep model

where α is the descriptor parameter and λ is a parameter controlling the intensity of the penalty.

In this subsection, we want to inspect the effect of λ, for 10 values on the logarithmic scale between 10^{-5} and 10^{-1}.

For each value of λ, a deep model with the structure defined in II-B2 is defined and trained over 50 epochs, with mini-batches of size 100 and learning rate $\eta = 0.1$.

Each time, the loss on test set is recorded during training and displayed on figure 6.

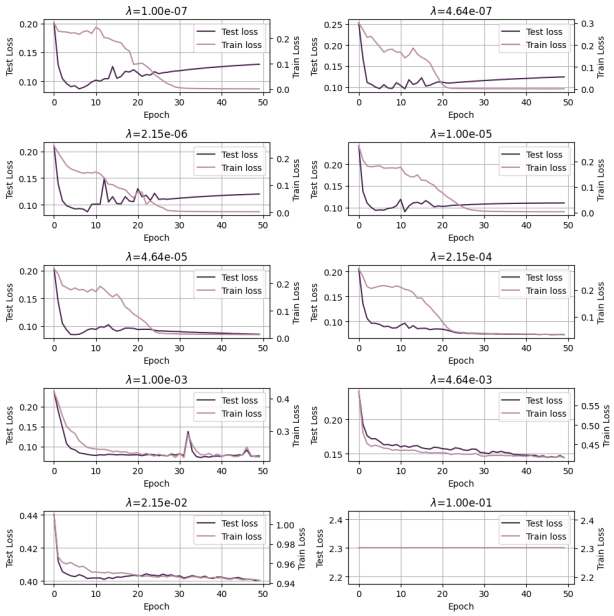
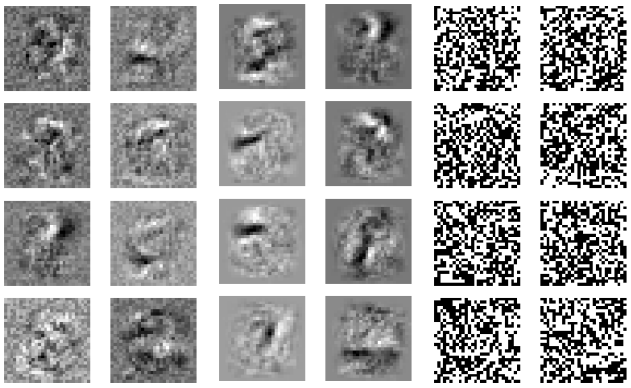


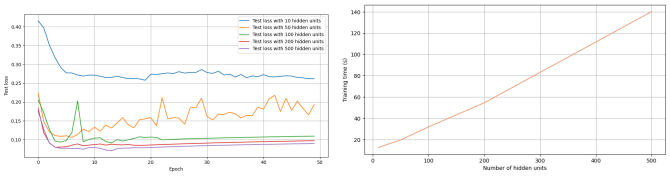
Fig. 6: Impact of regularisation over test loss

It is also interesting to see visualise how weights change under stronger or lighter regularisation pressure. Figure 7 is a representation of the weights of the first layer, reshaped into 28x28 images, right after the training process. For three different regularisation levels (no regularisation, moderate regularisation, and extremely high penalty), 8 sample weights are chosen at random and displayed as grayscale images.



(a) $\lambda = 0$ (b) $\lambda = 10^{-3}$ (c) $\lambda = 1$

Fig. 7: Representation of the deep model weights after training with different L2 regularisation coefficients



(a) Evolution of test loss (b) Training time

Fig. 8: Evolution of test loss (a) and execution time (b) for different configurations of number of neurons

C. Tuning number of neurons

For neural networks with hidden layers, the number of neurons in these hidden layers was, until now, set to 100. In this subsection, we try and vary this parameter to see if and how it affects the loss function and the prediction accuracy.

Let's try with 10, 50, 100, 200 and 500 neurons. These values apply to every hidden layer, without no distinction between the layers themselves. It could also be interesting to change the size of each layer, but we already have a lot of parameters to test, so we limit to this study.

The other parameters stay constant : $\eta = 0.1$, batch size of 100 and training over 50 epochs. λ is set to zero so that distinguishing the overfitting due to the number of hidden units is easier. Figure 8 highlights the impact of the number of neurons with two different metrics : the evolution of test loss during training (figure 8a), but also the time taken for training (on CPU).

D. Impact of learning rate

Another lever to be tuned is the learning rate η. This parameter controls the step size of the stochastic gradient descent algorithm. Intuitively, the tradeoff is the following : a very low learning rate will allow SGD to explore very precise regions of the loss landscape, with low probability to jump over a minima without seeing it ; but it will take a lot of time to explore a given region. On the contrary, a high learning rate will help cover a fixed part of the loss landscape quickly, but will probably miss local minima. So, it's all supposed to be about convergence speed, but the goal of this section is to see if it can also create overfitting issues.

For this experiment, 10 values of η were selected on the logarithmic scale from 10^{-4} to 1. The metric that was chosen to present the results on figure 9 is the test loss.

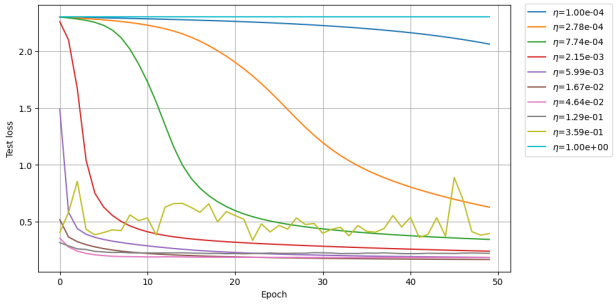


Fig. 9: Impact of learning rate over test loss

E. Dropout

Finally, we also investigate a last technique, namely dropout [6].

The idea of dropout is to tackle overfitting by randomly "dropping out" a subset of neurons in the network during training. At each training step, a certain percentage p of neurons are temporarily removed, along with all their incoming and outgoing connections. This forces the network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. By not relying too heavily on any one neuron, dropout encourages the network to distribute the learned information across multiple pathways, increasing its generalization capabilities. This randomness also mimics having a large number of different network architectures, but in a computationally efficient manner. When the network is evaluated on new data, all neurons are used, but their outputs are scaled down proportionally to the dropout rate, maintaining a balanced contribution to the final prediction.

In this subsection, we train the same deep network than before, to which we added a dropout operation before each dense layer.

The tuned parameter here is the dropout probability p. With $p = 0$, the network is the same as the deep network in II-B2, and with $p = 1$, the network is annihilated.

In order to visualise correctly the impact of dropout, we set $\lambda = 0$ so that there is no other form of regularisation. The other parameters are kept as usual.

Three values of p are tested : $p = 0$ (baseline), $p = 0.25$ and $p = 0.5$. Figure 10 shows the evolution of the test loss for these 3 choices of p over 50 epochs.

F. Performance of the best model

After having presented the impact of each parameter's tuning and their effect on over/under fitting, let's come up with the best combination of parameters in order to have a

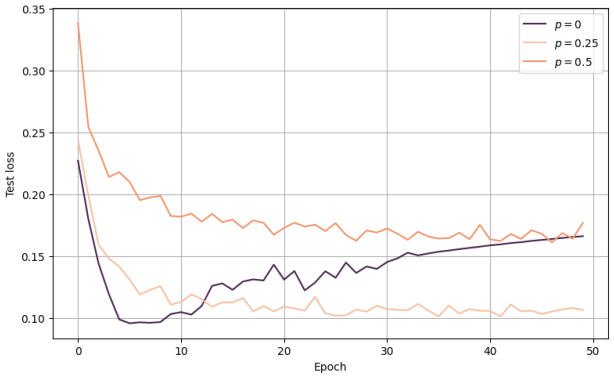


Fig. 10: Impact of dropout over test loss

fully connected network that suits well the given task, namely classifying MNIST samples.

Based on the previous results, the following parameters are selected :

- $\lambda = 10^{-3}$
- $\eta = 0.1$
- 100 hidden units
- dropout with probability $p = 0.25$

These parameters are applied to the deep network presented in section II-B2. After 100 epochs of training, the model is tested over the whole test dataset (10'000) samples and has an accuracy of 0.980. Which means 2% of the samples are misclassified. Figure 11 presents the confusion matrix of this model.

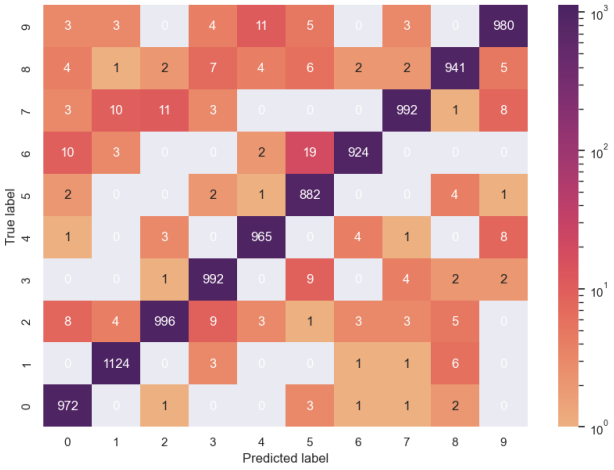


Fig. 11: Confusion matrix of the model presented in III-F over the MNIST test dataset. The numbers in the boxes indicate the number of samples classified as such.

IV. Discussion

A. Shallow VS deep models

As depicted in figure 3, the linear model, which is composed of a single layer with no non-linearity activation function, displays a specific trend in its training loss, test loss, and accuracy. After a period of 50 epochs, the accuracy stabilises to 92%, which is not a great value (in the sense that 8% of misclassification is often too high for a practical application). The train loss constantly decreases, but still does not reach a good value after a large number of epochs. The test loss finds its optimal value quickly (after 10 epochs) and then increases slightly. The model's simplicity, indicated by its relatively stable but poor test loss and accuracy trends, suggests a potential underfitting scenario. Underfitting occurs when a model is too simple to capture the underlying patterns in the data. This is often characterized by bad performance on both training and validation datasets, which is the case here.

In the shallow and deep models, non-linearity activation functions are introduced and layers are added. Figure 4 (shallow model) and figure 5 (deep model) both depict a similar behavior. In both cases, the best values of test loss and accuracy (98%) are way better than for the linear model. What's more, the train loss is now quasi reaching zero, at the same time that test accuracy stabilises.

What differs between shallow and deep models is, from the one part, the stability of the losses. Figure 5 shows higher fluctuations of the losses during the training part. This is maybe due to the fact that the architecture being more complex, the number of parameter is higher and the stochastic gradient descent [5] has to navigate within a higher-dimensional loss landscape. Each epoch can therefore result in significant changes to the weights, particularly in the early stages of training, before the model begins to converge. On the other part, we can detect in figure 5 a overfitting trend. Indeed after no more than 10 epochs, the minimum of the test loss is found and from there, the loss increases continually. This is typical of overfitting situations : while the train loss is quickly dropping to zero, indicating the model fits very well to the train data, its capacity to predict the test data is limited, because it relied and learned too much from the train data. This is a typical scheme for deep models, which have a higher capability of learning from train datasets, at the cost of lacking generalisation on unseen data.

In short, the tradeoff between shallow and deep models can be summed up as follows : shallow/linear models do not make awesome predictions, and their capacity to infer well test datasets after training is limited (underfitting). On the contrary, the complexity of deeper models give them power to fit data, but without regularisation, they fit train data so well that they become bad at fitting unseen data (overfitting). Thus, the goal is to cut down on this unwanted effect using regularisation.

B. The impact of regularisation

L2 regularisation, as applied in the described models, involves adding a penalty term to the cross-entropy loss function, proportional to the square of the magnitude of the model parameters. This approach penalizes larger weights, encouraging the model to maintain smaller, more distributed weights, thereby reducing model complexity and the risk of overfitting.

Figure 6 shows a deep network trained 10 times, using each time a different value of λ (as defined in equation 1).

The lowest value, $\lambda = 10^{-7}$, can be assimilated to zero : the trends of train and tests loss are very similar to the ones on graph 5. In particular, it shows an overfitting behavior of the network : the train loss reaches low values, while the test loss increases, which highlights the difficulty for the network to generalise to test data.

For the following values of λ, the overfitting trends fades out as λ increases : the train and test loss starts converging towards the same values after some epochs. Around $\lambda = 10^{-3}$, both curves even merge and the overfitting has completely disappeared. Nevertheless, when λ is too high, the convergence is drastically reduced. For $\lambda = 2.15 \times 10^{-2}$, we notice that the magnitude between the initial test loss (0.44) and the test loss after 50 epochs (0.40) is smaller than for previous values of λ, where the difference of magnitude was 4 or 5 times more important. This means that the test loss is quickly bounded by a minimum value, which is a sign of underfitting : the constraint term 1 becomes too heavy in comparison with the loss function, and the stochastic gradient descent algorithm, which aims at minimising the cross-entropy plus 1, suffers from a too high weighting of the constraint. Towards extreme values, for example $\lambda = 0.1$ on figure 1, the underfitting trend is crystal clear : both losses are constant, which means the network becomes tremendously inefficient to make predictions.

It is also interesting to take a look at the weights of the neural network, to see how they differ with respect to the regularisation that is applied. Luckily, this work is about OCR, so the weights are naturally visible objects (handwritten digits, originally). Figure 7 shows these weights under 3 different configurations : 2 two extreme values of λ, as well as $\lambda = 10^{-3}$, which is believed to be the most suited one.

- The weights visualized for $\lambda = 0$ show a lot of detail and variation, indicating that the model might be capturing a lot of the data's complexity and noise. This level of detail in the weights is a sign that the network could be overfitting.
- The weights at the level $\lambda = 10^{-3}$ appear less detailed than the $\lambda = 0$ case, suggesting that the model is less likely to be overfitting. There is a balance between capturing the underlying structure of the data and avoiding the noise.
- As for $\lambda = 1$, the weights look the least detailed, almost binary, which implies that the regularisation may be too strong, leading the model to underfit. The network is likely failing to capture the important patterns in the data, resulting in a loss of predictive performance.

In a nutshell, the results obtained in figure 6 suggest that regularising using L2 regression effectively balances the model's complexity and its ability to generalise. The mathematical framework of L2 regularisation is integral to enhancing the performance of neural networks (on the MNIST classification task, at least). It effectively constrains the model complexity, ensuring that the learning process remains focused on capturing the underlying patterns in the data rather than overfitting to the noise. The regularisation weight λ is to be tuned by trying a bunch of values and comparing for each one of them the train-test losses over epochs, or even the test accuracy metric.

C. Number of neurons

In the goal of optimising neural network performance, the number of neurons within hidden layers is another important hyperparameter. It directly influences the model's capacity to learn from data, shaping its ability to generalize and avoid overfitting or underfitting.

Increasing the number of neurons enhances the network's representational power. This allows the model to capture more complex relationships within the data. However, beyond a certain point, this increased complexity could theoretically lead to overfitting, where the model learns the noise instead of the underlying pattern. Conversely, too few neurons can restrict the model's learning capability, resulting in underfitting, where the model fails to capture the complexity of the data and performs poorly on both the training and test sets.

So let's see if these theoretical considerations apply to our data. Figure 8a shows the test loss of the same network for 5 different configurations of hidden units : 10, 50, 100, 200 and 500.

It seems clear that 10 neurons are not enough to capture the essence of the data : the loss remains high, and does not improve over epochs, which shows underfitting.

As for the other configurations, it is harder to draw a clear conclusion. In the case of 50 hidden units, overfitting occurs : the test loss reaches its minimum within 5 epochs and then increases again. However, for networks with bigger layers, the test loss is increasing very slightly, but maybe not enough to show overfitting. This might be because an other form of regularisation comes into play, although $\lambda = 0$ which kills L2 penalty term.

On the other hand, figure 8b shows that the number of neurons is linearly correlated to the training time for the interval [10, 500] at least. This cost is to be taken into account when balancing the mathematical results of a network and the practical consideration of available CPU resources.

In practice, finding the optimal number of neurons is an empirical process that involves monitoring loss and accuracy while adjusting the model's complexity. It is also influenced by the size and diversity of the training data, and the regularisation techniques employed. For the rest of this work, building layers with 100 hidden units seems to be a reasonable tradeoff between performances and training time.

D. Impact of learning rate

In this subsection, we shall observe the impact of the learning rate and determine to what extent it is responsible for good fitting, and wether it can be used as a regularisation hyperparameter.

The learning rate parameter η operates during the optimisation process of the loss function. Integrated to the optimiser

(stochastic gradient descent for our case), it controls the speed at which the exploration of the loss landscape is conducted. A low learning rate is inefficient to explore a wide part of the space, but will not miss local minima. Conversly, higher learning rates help converging faster without the guarantee to detect subtle asperities of the landscape.

Figure 9 confirms this statement. Among the 10 rates tested, the lower ones converge slower than the high ones. Two values present an interest in our search for regularisation : $\eta = 0.359$ and $\eta = 1$. While all other values of η result in a smooth loss curve, these two outsiders exhibit unusual behaviors.

A learning rate of $\eta = 0.359$ is relatively high. At this magnitude, the learning rate may cause the model's weights to update too aggressively, potentially causing the model to overshoot minima in the loss landscape. This results in the loss not steadily decreasing and instead showing spikes and significant variability. The spikes in the loss curve could also point towards moments where the model starts to overfit the training data. Because no regularisation method is effectively implemented, the model probably picks up noise from the training data, leading to temporary decreases in loss followed by sharp increases when evaluated on the test set.

As for the light blue curve, $\eta = 1$ is also too large for the model to make nuanced adjustments to the weights. Instead of converging to a solution, the model's loss actually diverges.

Thus, we can conclude that extremely high values of learning rate lead to bad or no convergence at all, that we could maybe impute to overfitting. However, unlike λ, the number of neurons, or other hyperparameters, the learning rate is not a key parameter to control the goodness of fit.

E. Regularising using dropout

Dropout is the last regularisation technique we will study. It aims to reduce overfitting in neural networks by preventing complex co-adaptations on training data. It is achieved by randomly disabling a fraction of the neurons during the training process, forcing the network to learn more robust features that are not reliant on any small set of neurons.

During testing or evaluation, dropout is not applied. Instead, the activations are scaled appropriately to account for the reduced number of active neurons during training.

The dropout rate p is a tunable parameter. In figure 10, we tested 3 different values : $p = 0$ as a baseline, $p = 0.25$ and $p = 0.5$. Naturally, when $p = 0$, we observe that the network is overfitting because without regularisation technique, the situation is the same than on figure 5.

By turning off 25% of the hidden units, though, the convergence of the test loss improves drastically. It provides a moderate level of regularisation, forcing the network to learn redundant representations to make good predictions without relying too much on any individual neuron.

A dropout rate of $p = 0.5$ however, turns off half of the neurons on average, which is a more aggressive form of regularisation. It may be too extreme for our model which is maybe not deep enough for such a high dropout probability, leading to underfitting by not allowing the network to learn the patterns in the data sufficiently.

Therefore, dropout seems to be an efficient way to regularise a network. Dropping randomly neurons allows to lighten a network, making it less dense, less fully connected. Indeed, by removing connections between hidden units, it forces the network to rely less on the specificities of the training data, thereby reducing overfitting.

F. Best model performance

The model presented in section III-F outputs the best accuracy within the subset of tested parameters for a fully-connected 3-hidden layers network using dropout. 98% is a rather good performance for a lot of use cases, especially when knowing the confusion matrix 11 which gives insights about false positives and false negatives.

Overall, the diagonal cells of the confusion matrix, which represent correct predictions, show high values, indicating that the model has a strong predictive accuracy for most classes. For example, the model has confidently predicted the majority of the digits for classes 1, 3, 4, and 7 with very high accuracy, as seen by the high counts (1124, 992, 965, and 992 respectively) in the diagonal.

The off-diagonal counts are relatively low for many pairs of digits, which is a good sign. However, certain misclassifications are more frequent, such as digits '5' being predicted as '3' or '6'. This is probably due to similarities in the handwriting styles of these digits, or it could be an indication that the model might benefit from further training, more nuanced feature extraction, or perhaps a more complex architecture like a convolutional neural network that can capture spatial hierarchies in the image data more effectively. Indeed, fully connected networks are often limited in their capacities when it comes to learning subtle spatial features, because dense nets flatten the input data beforehand, whereas the convolution operators of CNN help capturing this two-dimensional relationship.

V. Conclusion

In conclusion, this work explored different network architectures aiming at classifying MNIST dataset. After having explored different models and having fine-tuned the hyperparameters, we have produced a robust model with 98% accuracy. We emphasized, using various indicators, the importance of tuning parameters in order to avoid under/over fitting so that the model generalises well.

Looking forward, there are several possibilities for further work to enhance the model's performance and address the remaining challenges.

Firstly, as stated in IV-F, the misclassification of certain digits suggests that using more sophisticated feature extraction methods could increase the model's accuracy, that reached a plateau around 0.98. In particular, using convolutional neural networks (CNNs) could be efficient, as they often manage to capture spatial relationships in images.

Secondly, we could explore some additional methods during

training to cut down on overfitting. We can mention, for example : batch normalisation, which normalises weights within each batch ; or also exploring the impact of different optimisers, such as Adam that adapts the learning rate dynamically to the loss landscape.
Monitoring convergence can also be done even before training, on the data itself. Researches have shown that performing data augmentation (data pre-processing) or feature selection before the network is trained can help the model learn better from data, by throwing away the most useless and noisy parts of the input.

In conclusion, this work explored a limited range of network architectures, but rather focused on how to tackle overfitting by tuning the available hyperparameters. Further exploration of the model's architecture could certainly lead to more accurate classification results, but this project showed that from "simple" models, it is already possible to limit consistently overfitting just by adjusting a set of parameters.

REFERENCES

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. *Handwritten digit recognition with a back-propagation network*, page 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[5] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.