Eindhoven University Of Technology

Computer Vision and 3D Data Processing

Fundamentals of Neural Networks

2064472 – R.Regalo

February 3, 2024

I. Multi-Layer Perceptron

I.1. Implementation

The images in the MNIST dataset are 28 by 28 pixels, so the first layer needs to have 784 neurons. Because the output is one hot encoded and there are 10 classes, the output layer needs 10 neurons. We decided to start simple for the hidden part of the network with a single layer of 128 neurons. Training the model led to good accuracy, so this architecture was chosen moving forward.

I.2. Training and testing

I.2.1. Question 1

After training the model using Mini-batch Stochastic Gradient Descent, the final accuracy was 99.9% on the train set and 97.5% on the test set. Intuitively, it makes sense that the model performs better on data that it has seen when compared to data that it has not seen. By plotting the loss and accuracy over time, as seen in figure 1, we can better analyse this behaviour. We can see that the training accuracy keeps increasing while the validation accuracy remains constant, which is a sign that the model is overfitting on the training data.

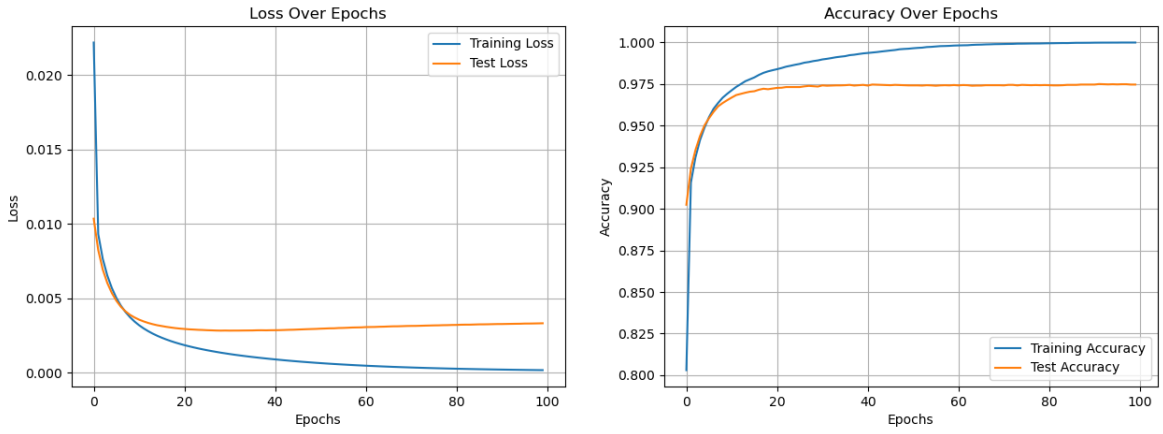

Figure 1: Loss and Accuracy of Epoch

I.2.2. Question 2

In a fully connected layer, each neuron has a weight connecting it to every neuron from the previous layer plus one bias term. Therefore, for an MLP with L layers, where each layer l has N_l neurons and receives input from N_{l-1} neurons from the previous layer (N_0 represents the model input dimension), the expression for the total number of parameters P is:

$$P = \sum_{l=1}^{L} ((N_{l-1} \times N_l) + N_l) \tag{1}$$

Our model has:

- Input layer $N_0 = 784 = 28 \times 28$ pixels (MNIST)
- Hidden layer $N_1 = 64$
- Output layer $N_2 = 10$ (One for each ouput class)

Filling the values in 14 we obtain a final value of 50890 parameters

I.2.3. Question 3

A convolutional layer often has fewer parameters than a fully connected layer, so the total number of parameters would greatly decrease. The layer has a set of filters that are applied to the full image so the number of parameters is not dependent on the size of the input image. The Batch normalization process also typically requires only two parameters, one for scaling and one for shifting. On the contrary, a fully connected layer requires each neuron to have a number of parameters equal to the input size, which scales very fast with the input size.

A convolutional layer can act as a fully connected layer if the kernel size is the same as the input size. This would make the convolution operation equivalent to a weighted sum over the entire input, which requires a large number of parameters. Conversely, a fully connected layer can be seen as a convolutional layer if each neuron looks at only a local region of the input and shares weights with neurons looking at other regions. This is, however, a very atypical architecture and does not take advantage of the different types of layers.

I.2.4. Question 4

a)

In order to achieve more than 99.0% accuracy, we need to go with a more complex architecture than an MLP, starting by introducing convolutional layers. These are often used in image-related tasks for their efficiency and ability to capture spatial hierarchies. They use fewer parameters through weight sharing, reducing computation and letting them learn from different parts of the image equally. This setup helps learn various aspects of the image, from simple edges in the initial layers to more complex patterns deeper in the network. It is common to use pooling layers after convolutional layers because they reduce the dimensionality of the feature maps and hence help learn the high-level features.

Using a more complex model brings a problem of over-fitting when trying to get more than 99% accuracy. To fight this, two very easy changes that we can apply to our model are adding dropout layers and batch normalization layers. Dropout layers randomly inactivate some neurons while training. This makes the network generalize better by simply not relying too much on any particular neurons to produce an output. Batch-normalization layers normalize inputs, reducing internal covariance shift,

which helps each layer learn more independently and efficiently.

After searching online for different image classification models and experimenting with different architectures, the final model was chosen and is represented in the figure below

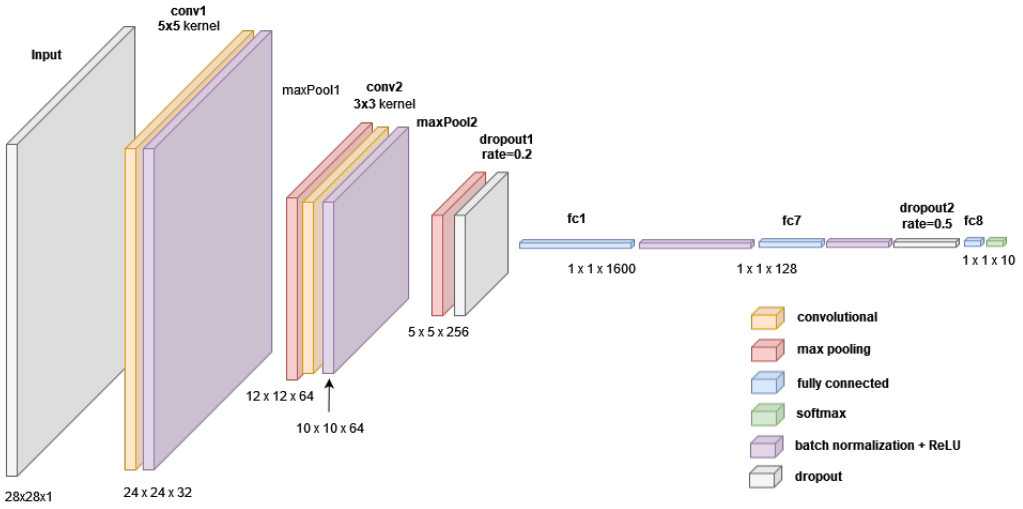

Figure 2: Diagram of the implemented model

Using this model with the original data already yielded very good results achieving a validation accuracy of 99.3% as seen in the image below.

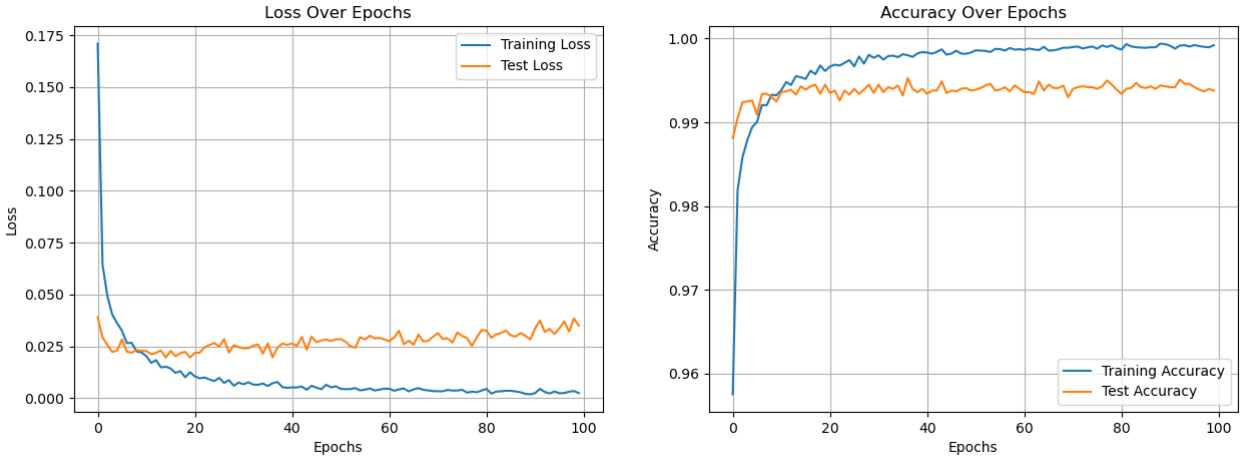

Figure 3: Loss and Accuracy over epochs

Apart from changing the model architecture, another change that we can make to improve performance working on the dataset itself. Firstly, we should normalize all images by subtracting the mean and dividing by the standard deviation from each pixel across the training set, ensuring that pixel values range more consistently rather than spanning from 0 to 255. This normalization reduces variance in inputs and speeds up convergence when training.

Another operation that we can perform on the dataset is data augmentation. This consists of randomly rotating, scaling, zooming in and out, and translating some images. Such transformations diversify the training set by introducing variations that the model might encounter in the validation, thereby reducing over-fitting and increasing the model's ability to generalize from irregular or less typical handwritten digit representations.

However, after implementing these changes we concluded that the accuracy remained the same as before. This is probably due to the fact that these changes are not as impactful when we are so close to the Bayesian error limit (Error due to ambiguities in the dataset, which is around 0.3% for MNIST).

b) With no limits on computational power, we might be able to push the accuracy further. By thoroughly fine-tuning hyperparameters, we can ensure the model is set up for success from the start. We can also combine different models into ensembles, which helps mitigate any inconsistencies individual models might have, leading to more accurate predictions.

II. Loss functions

II.1. Question 1

In order to derive $\frac{\partial y}{\partial z}$, we must keep in mind that y and z are vectors so in fact we want to compute $\frac{\partial y_i}{\partial z_j}$. Immediately it can be seen that there two different cases for $i = k$ and $i \neq k$. For the first case we get

$$\begin{aligned}
\frac{\partial y_i}{\partial z_i} &= \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_k}}\right) && (2)\\
&= \frac{e^{z_i}\cdot\sum_{j=1}^{K} e^{z_k} - e^{z_i}\cdot e^{z_i}}{\left(\sum_{j=1}^{K} e^{z_k}\right)^2} \\
&= \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_k}} - \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_k}}\cdot\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_k}} \\
&= y_i(1-y_i)\,, && (3)
\end{aligned}$$

and for the second we get

$$\begin{aligned}
\frac{\partial y_i}{\partial z_j} &= \frac{\partial}{\partial z_j}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_k}}\right) && (4)\\
&= \frac{0\cdot\sum_{j=1}^{K} e^{z_k} - e^{z_j}\cdot e^{z_i}}{\left(\sum_{j=1}^{K} e^{z_k}\right)^2} \\
&= -\frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_k}}\cdot\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_k}} \\
&= -y_j y_i\,. && (5)
\end{aligned}$$

II.2. Question 2

Using the chain rule, we can derive $\frac{\partial \mathcal{L}_{ce}}{\partial z}$ from $\frac{\partial \mathcal{L}_{ce}}{\partial y}$ and the previously derived $\frac{\partial y}{\partial z}$. The first is given by

$$\frac{\partial \mathcal{L}_{ce}}{\partial y} = t \cdot \frac{1}{y} + (1-t) \cdot -\frac{1}{1-y} \,. \tag{6}$$

Finally, applying the chain rule we get

$$\begin{aligned} \frac{\partial \mathcal{L}_{ce}}{\partial z} &= \frac{\partial \mathcal{L}_{ce}}{\partial y} \cdot \frac{\partial y}{\partial z} \\ &= \left[t \cdot \frac{1}{y_i} + (1-t) \cdot -\frac{1}{1-y_i} \right] \cdot [y_i(1-y_i)] \\ &= y_i - t \,. \end{aligned} \tag{7}$$

For a multi-class classification problem with one-hot encoded output (like the one developed previously), $t = 1$ for the correct class and $t = 0$ for every other class wich simplifies the expression to

$$\frac{\partial \mathcal{L}_{ce}}{\partial z} = y_i - 1 \,. \tag{8}$$

II.3. Question 3

Similarly to the previous question we'll start by deriving the expression for $\frac{\partial \mathcal{L}_{dice}}{\partial y}$ and then applying the chain rule. First we get

$$\frac{\partial \mathcal{L}_{dice}}{\partial y} = \frac{\partial}{\partial y} \left(1 - \frac{2yt}{y+t} \right) \tag{9}$$

$$= -\frac{2t \cdot (y+t) - 2yt}{(y+t)^2}$$

$$= -\frac{2t^2}{(y+t)^2} \,. \tag{10}$$

Using the chain rule and simplifying the expression we get

$$\frac{\partial \mathcal{L}_{dice}}{\partial z} = \frac{\partial \mathcal{L}_{dice}}{\partial y} \cdot \frac{\partial y}{\partial z} \tag{11}$$

$$= \left(-\frac{2t^2}{(y+t)^2} \right) \cdot [y_i(1-y_i)]$$

$$= \frac{y-1}{2y} \cdot \left(\frac{2ty}{y+t} \right)^2$$

$$= \frac{y-1}{2y} \cdot (1 - \mathcal{L}_{dice})^2 \ , \tag{12}$$

like we wanted to show.

II.4. Question 4

In a binary classification problem there are only two classes with labels 0 and 1. The output y becomes a scalar representing the probability that the predicted class is "1" (conversely the probability that the predicted class is "0" is $1-y$). Because there are only two possible values for t, the focal loss can be written as

$$\begin{cases} -(1-y)^\gamma log(y) & \text{if t = 1} \\ -y^\gamma log(1-y) & \text{if t = 0} \end{cases} , \tag{13}$$

which can be combined into just one simple expression as

$$\mathcal{L}_{focal}^{binary} = -t(1-y)^\gamma log(y) - (1-t)y^\gamma log(1-y) \tag{14}$$

II.5. Question 5

In order to achieve the required proof we'll derive the derivative and simplify it in order to reach the desired form.

$$\frac{\partial \mathcal{L}_{focal}^{binary}}{\partial y} = \frac{\partial}{\partial y} \left(-t(1-y)^\gamma log(y) - (1-t)y^\gamma log(1-y) \right)$$

$$= t \left(\frac{\gamma(1-y)^\gamma}{1-y} log(y) - \frac{(1-y)^\gamma}{y} \right) - (1-t) \left(\frac{\gamma y^\gamma}{y} log(1-y) - \frac{y^\gamma}{1-y} \right)$$

$$= \frac{-1}{y(1-y)} \left[t(1-y)^\gamma \cdot (-\gamma y log(y) + (1-y)) + (t-1)(y)^\gamma \cdot (-\gamma(1-y)log(1-y) + y) \right]$$

$$= \frac{-1}{y(1-y)} \left[t(1-y)^\gamma \cdot (\gamma \varepsilon_y + 1 - y) + (t-1)(y)^\gamma \cdot (\gamma \varepsilon_{1-y} + y) \right] \tag{15}$$

This concludes the proof.

II.6. Question 6

By looking at the expressions for the binary versions of both losses we can see that if we set $\gamma = 0$ in the focal loss the expression equals that of the cross-entropy loss. If γ is too high the training process will focus more on the classes that are harder to classify which may lead to under-fitting because the model will not learn how to classify the easier classes. If, on the other hand, γ is too low, the focal loss gets closer to cross entropy, losing its benefits. In an imbalanced dataset, this will lead to the model not being able to learn the harder classes to classify, hindering performance.

II.7. Question 7

Cross-entropy loss is the standard choice for classification tasks where each class is represented equally. It works well in measuring the accuracy of the predictions and is particularly useful when dealing with balanced datasets.

Focal loss is more useful in scenarios with significant class imbalances, such as object detection, where some classes may be much rarer than others. It helps the model focus on harder, often misclassified, cases by decreasing the loss contribution from easy ones. This focus helps to prevent the model from being biased towards the majority class and improves performance on the minority classes that are typically harder to detect.

II.8. Question 8

Cross-Entropy Loss: Optimizing cross-entropy loss tends to increase both precision and recall, as it directly optimizes for classification accuracy. However, if the dataset is imbalanced, it might not effectively improve recall since the model may still be biased towards the majority class.

Dice Loss: Dice loss is designed to optimize for the overlap between the predicted class and the true class. In terms of precision and recall, Dice loss can boost recall by penalizing false negatives more heavily, since it aims to maximize the overlap between prediction and ground truth. Precision might be less affected as it doesn't directly penalize false positives unless they significantly impact the overlap.

Focal Loss: Focal loss aims to address class imbalance by focusing on hard-to-classify examples. With a higher focusing parameter γ, the model is encouraged to correct misclassifications, which can lead to an improvement in recall, especially for the minority class. Precision might decrease if the model starts to classify too many instances as the minority class to improve recall.

References

[1] Pytorch documentation, Pytorch, https://bitly.ws/38Eat