Applied data science coursework

Alberto Plebani
ap2387

Number of words:

1 Training a diffusion model

I started the coursework by training a regular denoising diffusion probabilistic model on MNIST.

The MNIST dataset is a dataset including coloured images of numbers, of which an example can be seen in Figure 1. The goal of this part of the project was to build a denoising diffusion model (DDM). The DDM works by adding noise to the images, with the encoder mapping the input x through a series of latent variables $z_1, \dots, z_T$. The result of this process is having an image with only noise. Afterwards, the reverse process is learned by a decoder, which passes the data back through the latent variables removing noise at each stage. In the end, if we sample an image from the output model, we expect to see the same images that were used to train the model, without the noise.

The training process is iterative, and starts by adding noise to the image at different levels. In the training, both real and noisy images are used, and for each noisy image the model tries to predict the "true" (not-noisy) image that would have come in the previous step of the diffusion process. The prediction is then compared to the actual clean image, and by doing so the model learns how to remove the noise. This is used in the decoding process, where a noisy image is passed through the latent variables, removing the noise at every stage, eventually generating an image that resembles the original one.

In our case, our model takes a Convolutional Neural Network (CNN) to estimate the diffusion process, with this CNN which is then used to generate the images.

I trained the model using two different sets of hyperparameters, described in Table 1. The results are presented above

Set 1

For the first set, I decided to use a simpler model (see Table 1). In this model, the CNN has been built with 4 hidden layers, with 32,64,64, and 32 nodes each. For the learning rate, β, which specifies how noise is added and then removed during the encoding and decoding steps respectively, and the number of diffusion steps N_T, representing how many discrete steps are used to add or remove the noise from the data, the default values provided by

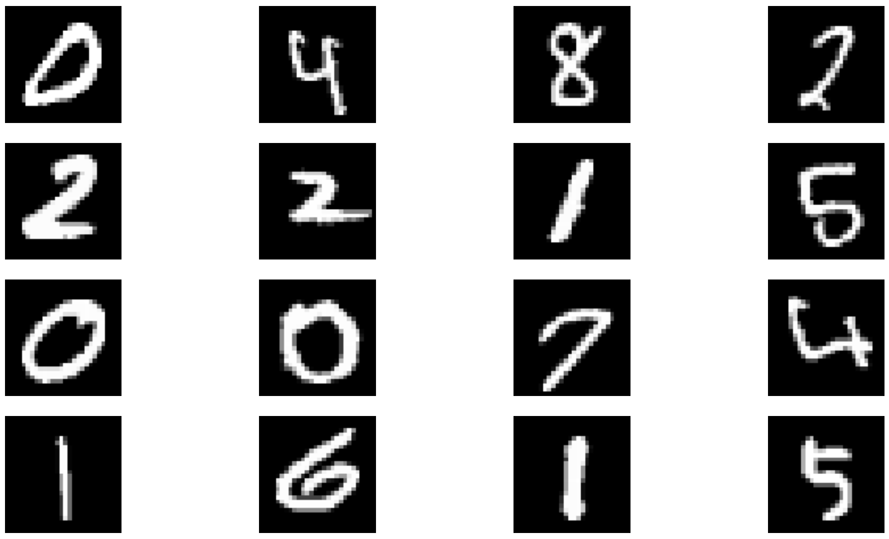


Figure 1: Example MNIST pictures

Hyperparameter	Set 1	Set 2
Hidden layers	32,64,64,32	32,64,128,64,32
Learning rate	2×10^{-4}	5×10^{-4}
β	$[10^{-4},\ 0.02]$	$[10^{-5},\ 0.01]$
N_T	1000	2000
Epochs	100	100
Patience	15	15
Δ	0.0005	0.0005
Optimiser	Adam	Adam
Activation functions	GELU	GELU
Loss function	MSE	MSE

Table 1: Hyperparameters for the two different models tested.

the code were used. The NN was allowed to train for up to 100 epochs, with the early stopping technique which was implemented in order to stop the training once it reached convergence. The early stopping was applied once the difference in loss function between epoch i and $i-15$ was lower than Δ, chosen to be 0.0005. Between the different layers, the Gelu activation function was used, defined as $GELU(x) = x/2 \cdot \left(1 + \text{erf}(x/\sqrt{2})\right)$.

The loss function is represented in Figure 2. We can see how the training stops after 88 epochs, and the loss function converges to a value of 0.019.

At each step of the training, a set of 16 images was sampled from the distribution. An example of these images is shown in Figure 3. We can see how after 1 epoch we are able to see only . After roughly 10 epochs, it was possible to see symbols consistently, with these symbols looking like numbers only after epoch 22, even though these numbers aren't always clear, as displayed by Figure 3(b). As the number of epochs increased, I

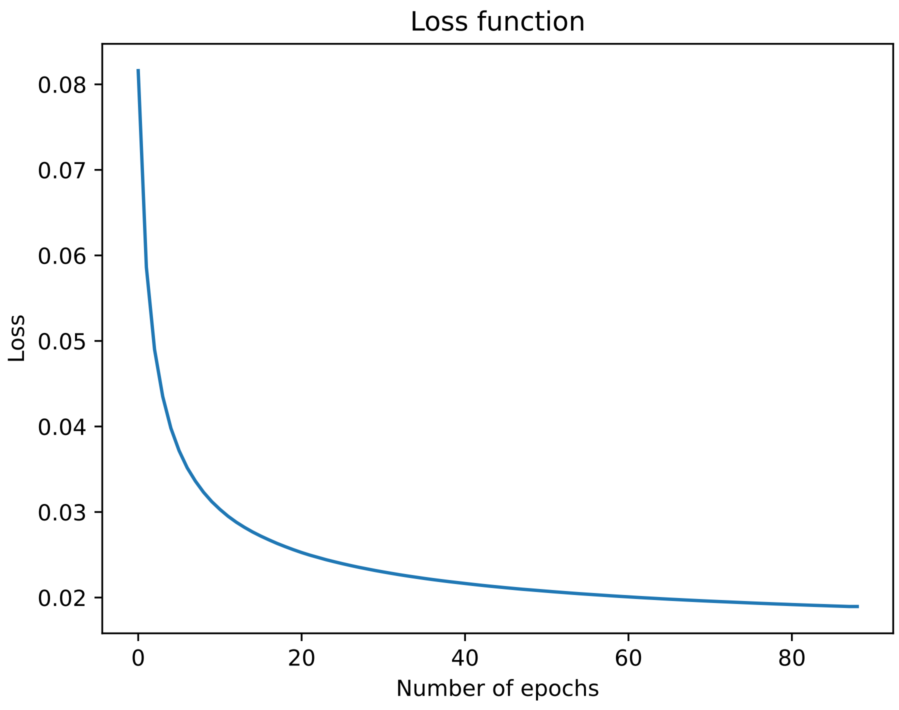


Figure 2: Loss function for Set 1

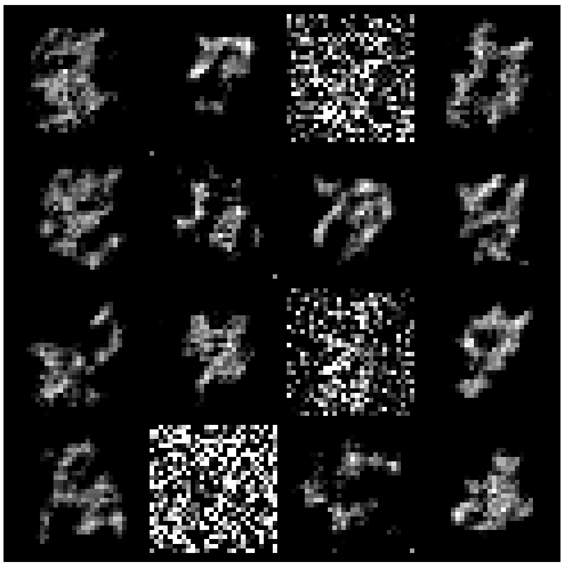
was able to identify always more and more numbers, reaching the final epoch (88) where more than half of the numbers sampled are well defined and can be correctly identified. Nevertheless, for some figures it still hard to say which number they are referring to. For instance, the figure in 3rd row and 3rd column could be either a 4 or an 8. Therefore, I decided to use a different set of hyperparameters, to try and improve the performance of the training.

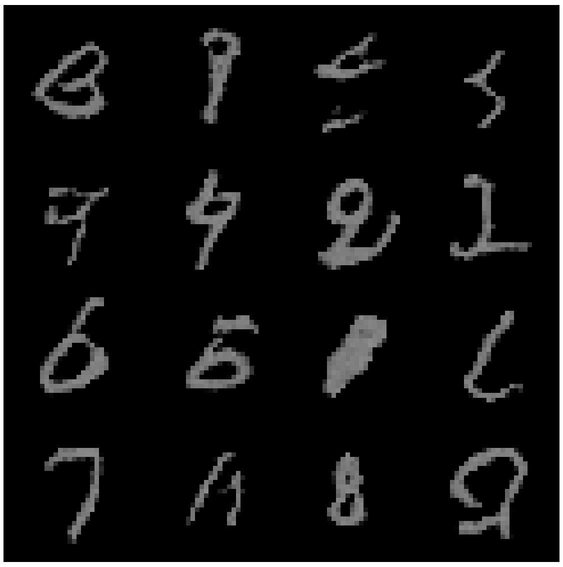
Set 2

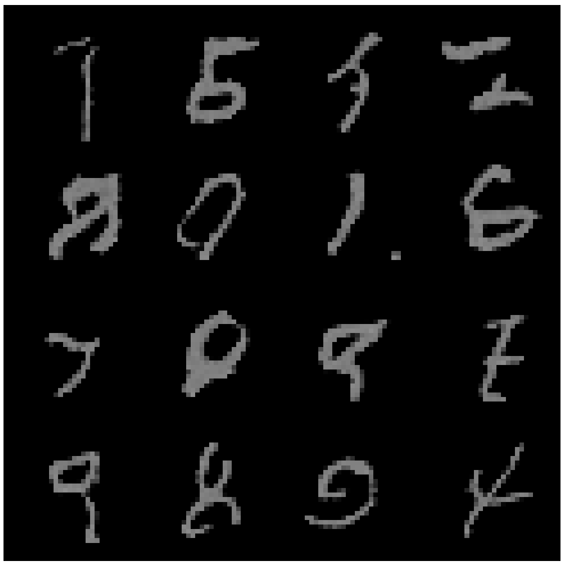
In this set, I decided to build a more complex CNN. This CNN has one additional layer of 128 nodes in the middle, resulting in 5 hidden layers, and I also decided to slightly increase the learning rate. Regarding the DDM hyperparameters, I tested a smaller set of values for β. This can lead to a decrease in the convergence time for the NN, but it has the advantage of improving the quality of the generated images. I also doubled the number of diffusion steps N_T, which allows for a more powerful model at the cost of increasing the time needed for training. The other parameters were kept the same as the ones for Set 1.

The loss function for this set is displayed in Figure 4. Once again, the early stopping technique was called once convergence was reached, this time after epoch 84. The loss function converged to a value of 0.018, slightly lower than the previous set.

Once again, I reported 4 example figures, each containing 16 sampled images. Similarly to before, we notice an increase in the quality of the generated images. However, this time the quality of the generated images is better than in the previous case. In the last image (Figure 5(d)) we can identify all the 16 numbers, ensuring that the NN has learned how to correctly decode the noise better than the otehr one. In this case, we start to see clearer numbers after roughly 16 epochs, comparable with what we obtained with the other set.


(a) Epoch: 1


(b) Epoch: 22


(c) Epoch: 65

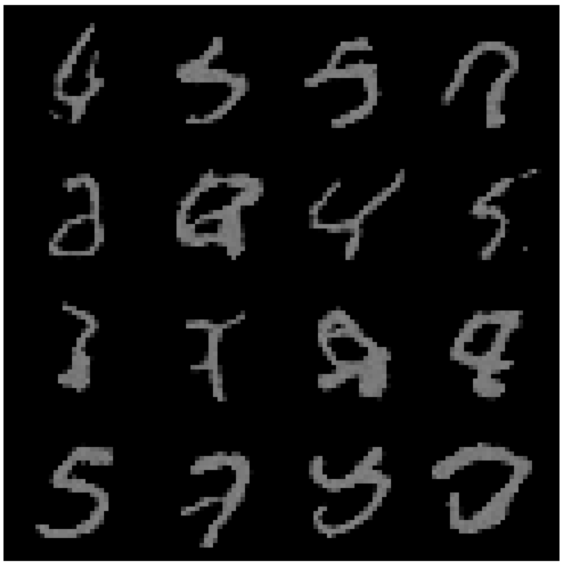

(d) Epoch: 88 (final)

Figure 3: Examples of sampled images for 4 different epochs, set 1.

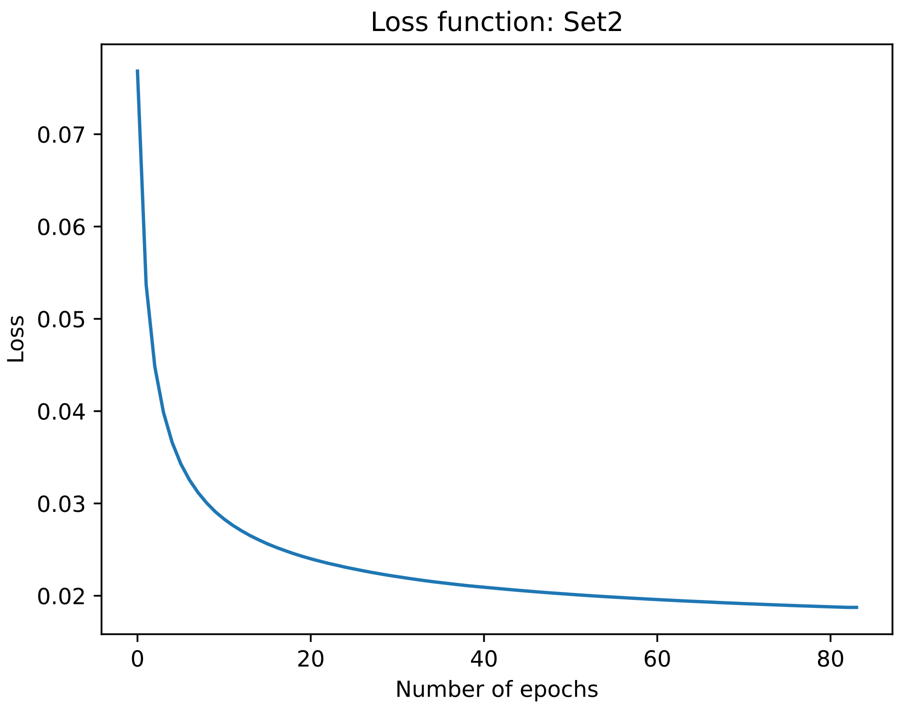
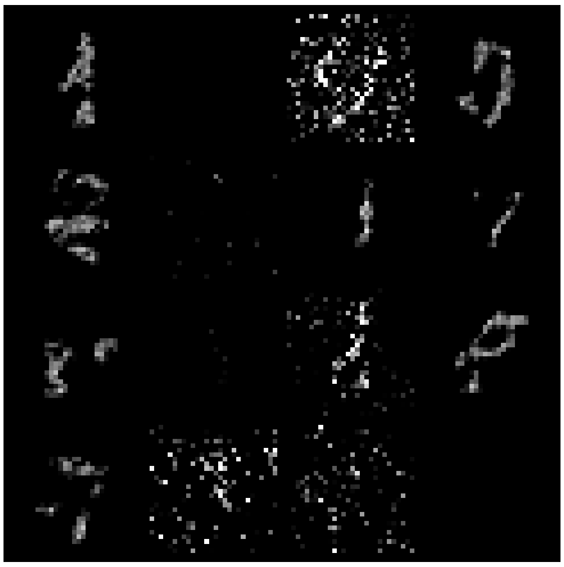
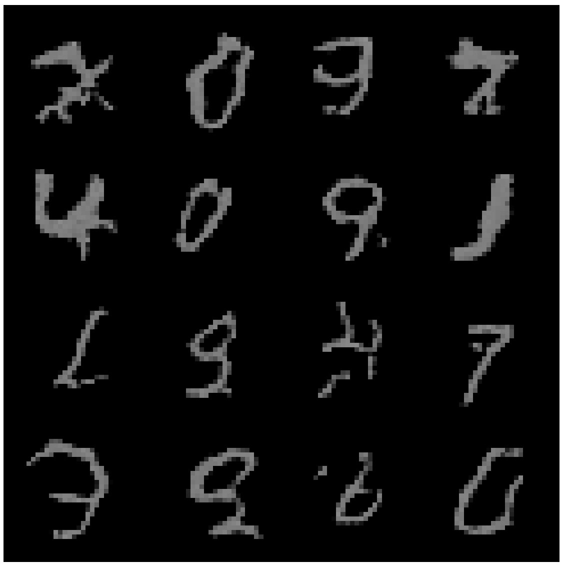
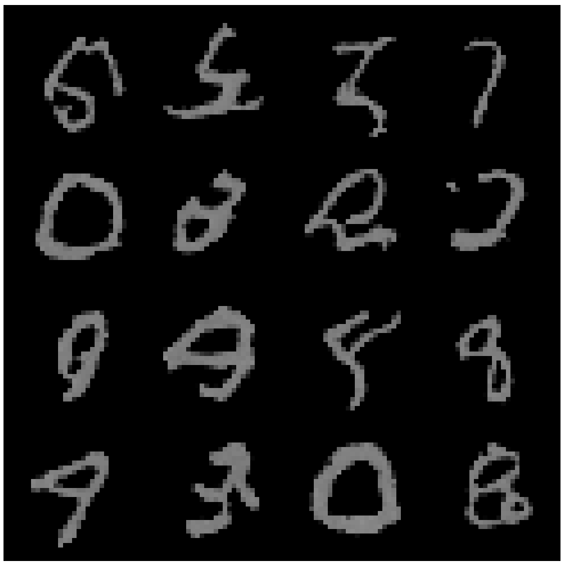


Figure 4: Loss function for set 2


(a) Epoch: 1


(b) Epoch: 16


(c) Epoch: 65

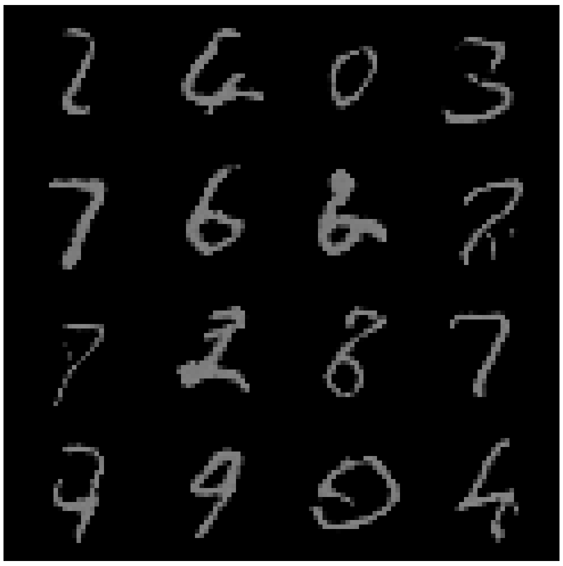

(d) Epoch: 83 (final)

Figure 5: Examples of sampled images for 4 different epochs, set 2.