ANN Exercise Session 1
Dimitrios Chatzakis r0977164

1.3.1 What is the impact of the noise (parameter noise in the notebook) with respect to the optimization process?

By adding random noise to the data of various degrees, we skew the data points by adding nonlinearities that the models will have to sieve through, up to a point that the original data distribution could be almost imperceptible. This noise will create many local minima that we would have to tackle in our selection of optimizer so as not to get trapped in one. The optimizer takes more epochs to train and achieves worse results for the reasons mentioned above. Results are displayed on the left.

	NOISE=0	NOISE=0.15	NOISE=0.5
TRAIN ERROR	*1.1e-4*	*16e-3*	*0.17*
TEST ERROR	*1.1e-4*	*8e-3*	*0.09*

1.3.2 How does (vanilla) gradient descent compare with respect to its stochastic and accelerated versions?

The original gradient descent, takes in all the data as input before updating. Therefore, it causes a very smooth conversion, but is inefficient computationally for large datasets and is slow to converge. Its stochastic version, uses batches to update the model parameters, thus it is much faster computationally, handling also large datasets efficiently, however, the gradients are noisier due to their reduced size and introduce wrong bias in the training process, but in practice, its inherent stochasticity helps escape local minima in cases where vanilla GD cannot. Its accelerated versions, with momentum, reach even faster convergence especially in cases of high curvature and reduce the impact of noise, enabling the model to escape more local minima and oscillations around them. However, the extra hyperparameter for the momentum needs careful tuning to avoid overshooting the global minimum. The Nesterov alteration of the algorithm, that looks at the gradient at the "predicted" next iteration of the parameters via a "look-ahead" part in the input to the gradient making it even better at handling noise and overshooting, but requires even more careful selection of hyperparameters. Adam utilizes momentum both on the first and second moments which are then bias-corrected before being parsed in the model in an adaptive fashion, updating weights individually, inhibiting or enhancing weights. The bias correction makes the optimizer also more robust to hyperparameter initializations. The LBFGS optimizer is an advanced method that does not utilize a learning rate as a hyperparameter but computes the direction of the update by an approximation of the second-derivative matrix and then performs a line search in that direction. The Strong-Wolfe method searches in a way that ensures that the objective function as well as the slope has decreased sufficiently at each iteration. This

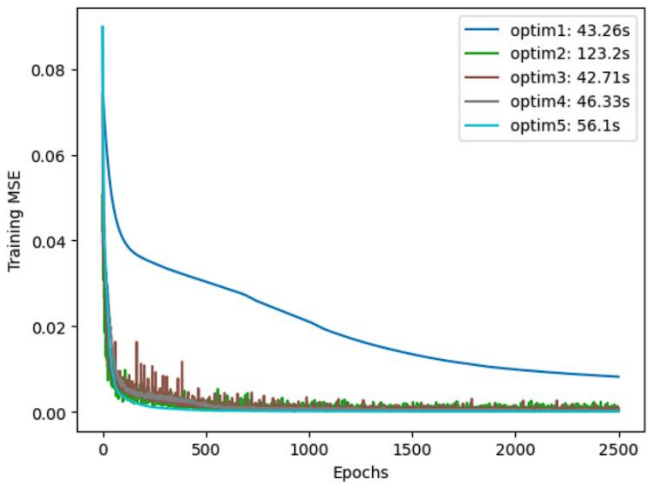

Figure 1 Noiseless Training

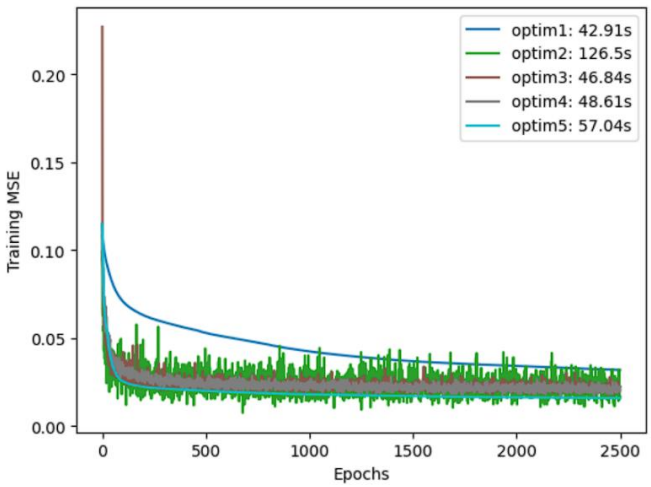

Figure 2 Noisy training (0.15 noise)

optimization algorithm is more complex to compute, but due to its much faster convergence achieves not only faster results but also better.

The addition of noise (Figure 2) to the original data (Figure 1) gives us some important insights. First, the GD takes the longest and noise does not impact considerably training time. In the original data where the curve is smoother the LBFGS achieves the best results. However, in the context of noisy data, the Nesterov algorithm seems to be the most robust followed by Adam, probably due to the look ahead and adaptive nature of the algorithms, while in both cases, the worst is the vanilla GD. Noteworthy is also the fact that in the original data, with smoother curve, the Nesterov SGD has more aberrant behavior than SGD, noting that momentum may be best fitted for noisy data. Results shown below, where the first cell data point is for the noiseless and the second for the noisy with 0.15 noise level.

	GD	SGD	NESTEROV SGD	ADAM	LBFGS
TRAINING TIME (S)	*43.3/42.9*	*123.2/126.5*	*42.7/46.8*	*46.3/48.6*	*56.1/57.0*
TEST MSE (E-4)	*82.6/94.4*	*7.45/37.16*	*5.26/33.5*	*3.78/38.44*	*1.42/72.28*

1.3.3 How does the size of the network impact the choice of the optimizer?

While training times are not substantially impacted for so small networks, we notice that all testing errors increase, and that in the case of 10 units per layer the SGD is the best performing optimizer, I would guess probably that the hyperparameters that the other optimizers introduce are not suited for so small models, leading to overfitting or overshooting.

UNITS IN E-4	GD	SGD	NESTEROV SGD	ADAM	LBFGS
MSE 10 UNITS	*143*	*17.8*	*23.2*	*29.2*	*19.3*
MSE 30 UNITS	*99.4*	*8.63*	*10.4*	*8.67*	*4.74*
MSE 50 UNITS	*82.6*	*7.45*	*5.26*	*3.78*	*1.42*

1.3.4 Discuss the difference between epochs and time to assess the speed of the algorithms. What can it mean to converge fast?

Time refers to the duration of an experiment, from the first to the last epoch, which is needed to compare algorithms. The epoch time is the time it takes for the model to pass through one iteration of the training dataset and is the metric which we compare to gauge whether the model converges fast. In an example where a model with SGD achieves 95% accuracy with 100 epochs in 500 seconds while with Adam it achieves the same results with 50 epochs in 600 seconds, the model with adam converges faster, since it needs less epochs but it takes more for the algorithm to train.

1.3.5 How many parameters does the model have?

Convolutional layers yield parameters of size (kernel width*kernel height*number of channels + 1(bias)) * number of output filters while dense layers yield (number of input layers+1)*number of output layers. This way the 1st convolutional layer yields (3x3+1)*32 = 320parameters, the second (3*3*32+1)*64 = 18496 and the dense one (1600+1)*10 = 16010 for a total of 34.826.

1.3.6. Replace the ADAM optimizer by a SGD one. Can you still achieve excellent performances? Try then the Adadelta optimizer. What is its particularity?

AdaDelta aims to mitigate the problem of the diminishing learning rate of AdaGrad by restricting the window of accumulated past gradients to a fixed size. It requires only the finetuning of a momentum-similar hyperparameter but no learning rate, leaning to slower, but more robust learning of models. The lack of momentum and bias correction make it though a subpar overall

	SGD	ADAM	ADADELTA
TEST ACCURACY	*0.976*	*0.993*	*0.567*
TEST LOSS	*0.082*	*0.020*	*2.094*

algorithm to Adam. SGD achieves subpar results to adam for the reason stated in 1.3.2. The results are displayed in the table to the left.

1.4.1 Define your training and testing dataset using respectively 2000 and 1000 samples drawn independently. Explain the point of having different datasets for training and testing. Plot the surface associated to the training set.

The purpose of having a different training and testing set is to test your model on a dataset other than the one it was trained on to see whether the model has generalized well on the underlying distribution. While you can prevent underfitting and overfitting on the validation set using only training and validation datasets it may be the case that your training data is not representative of the underlying distribution and yields bad results on the unknown test data overfitting on the test set.

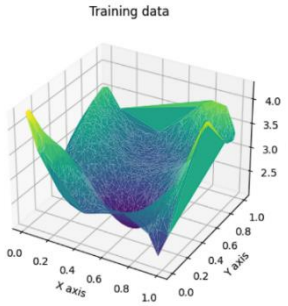

Figure 1 Training Data Visualization

1.4.2 Build and train your feedforward neural network. To that end, you must perform an adequate model selection on the training set. Investigate carefully the architecture of your model: number of layers, number of neurons, learning algorithm and transfer function. How do you validate your model? I created the following encoder-decoder architecture with the idea that I since I split my data into 80% training *2000 = 1600 I will have my first layer have that number of neurons, with relu for nonlinearity, then it will map into a 5**5 layer which would hopefully learn the associations between all combinations of the 5 nonlinear functions given the points, and then it will output into another 1600 neurons to try to recreate those points and give in the end the final z coordinate score. I trained with vanilla adam, sgd with Nesterov and varied my activation between relu and sigmoid. Another thing I added was callback for early stopping given 7 patience and a delta of 1e-4, along with a reduce learning rate callback when the validation loss didn't improve for 3 epochs with a factor of 1/np.sqrt(10) to make an order of magnitude a 2-jump process. The original architecture was tried with l1 and l2 regularization parameters, dropout and batch normalization but achieved subpar results with any kind of regularization. My initial model achieved a training error of 3.4e-3 and a validation error of 4.5e-3. (Code partly generated by GPT4).

```
l1_value = 1e-4
l2_value = 1e-4
dropout_rate = 0.5

net = keras.Sequential(
    [
        keras.layers.Input(shape=[x_train.shape[1]]),
        keras.layers.Dense(
            units=1600,
            kernel_regularizer=keras.regularizers.l1_l2(l1=l1_value, l2=l2_value)
        ),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(dropout_rate),

        keras.layers.Dense(
            units=5**5,
            kernel_regularizer=keras.regularizers.l1_l2(l1=l1_value, l2=l2_value)
        ),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(dropout_rate),

        keras.layers.Dense(
            units=1600,
            kernel_regularizer=keras.regularizers.l1_l2(l1=l1_value, l2=l2_value)
        ),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dense(units=1)
    ]
)
```

1.4.3 Evaluate the performance of your selected network on the test set. Plot the surface of the test set and the approximation given by the network. Explain why you cannot train further. Give the final MSE on the test set.

My results on the test set are 2.8e-3. There is no point in continuing training since the model reached a plateau on both training but mainly validation set. Any more training, would introduce overfitting.

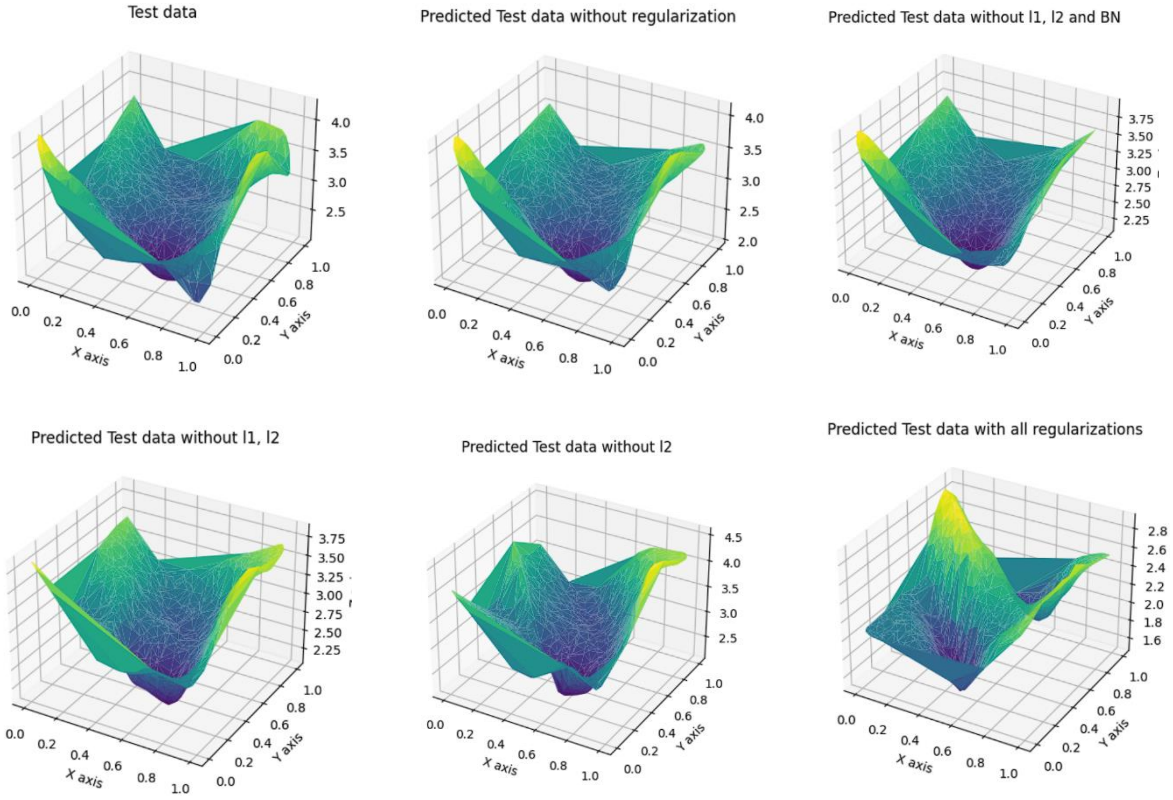

Figure 2 Visualization of training under different regularization paradigms.

Describe the regularization strategy that you used to avoid overfitting. What other strategy can you think of?

I tried l1, l2 and dropout and batch normalization. L1 for its feature selection via sparse representation since it is analogous to the absolute of the weight, L2 to reduce possible overfitting on the dataset since its proportionality punishes large weights since it is analogous to the weights squared. Dropout regularizes the network by deactivating certain neurons at each pass in a probabilistic manner. This way we prevent the model on relying too much on any one neuron and makes it more robust by making it learn more complex associations. Batch normalization adds a mild regularization effect by adding noise due to biased batch statistics that are used for it regularization. Some visualization of removing one regularization at a time is shown above.

ANN Exercise Session 2
Dimitrios Chatzakis r0977164

2.1.1 Create a Hopfield network with target patterns [1,1], [−1,−1] and [1,−1] and the corresponding number of neurons. Simulate the state evolution for various input vectors (e.g. random points or points of high symmetry) and note down the obtained attractors after a sufficient number of iterations. Are the real attractors the same as those used to create the network? If not, why do we get these unwanted attractors? How many iterations does it typically take to reach the attractor? What can you say about the stability of the attractors?

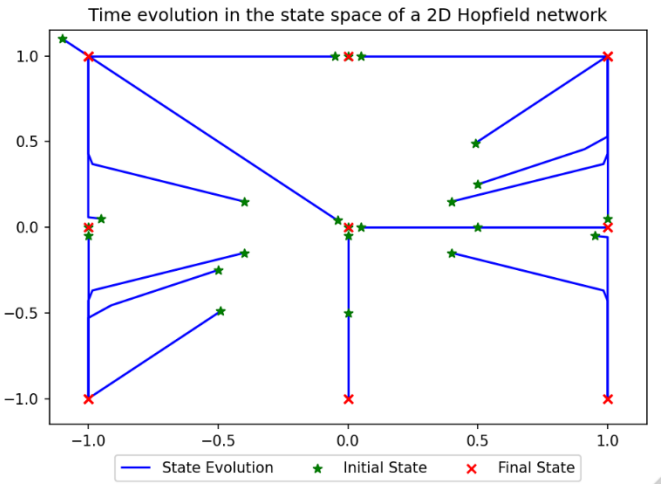

Figure 1 Final state of Hopfield network 2D

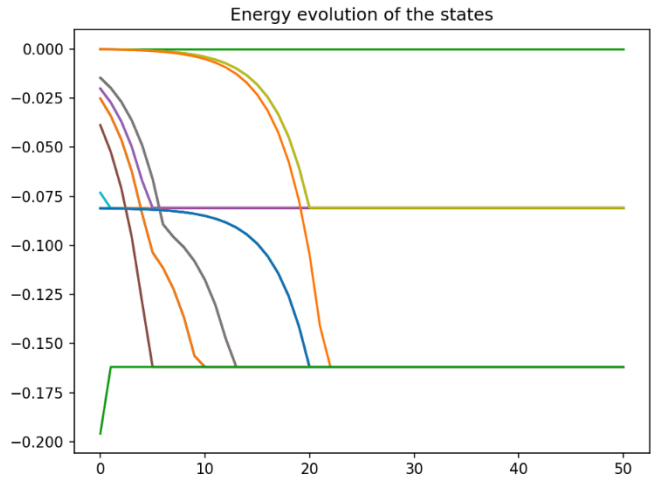

Figure 2 Energy evolution of Figure 1

The global attractors we obtain in the end are the initial 3 targets along with the [-1,1] point representing the 2 neurons, the first deactivated. Let's do some math to discern why: the matrix of weight sums after the 3 initial outer products for the targets if x1=[1,1], x2=[1,-1] and x3=[-1, -1] is $W = \frac{1}{N_{targets}}\left(\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} + \begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix} + \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\right) = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$ after we normalize and remove the self-connections. For random state [a,b] the activation of neuron 1, 2 will be sign(a+b). Practically, this separates the state space into 4 quadrants. Every point with a, b > 0 will end in x1, a > 0 and b < 0 will end in x2, a < 0, b < 0 in x3 and finally if a < 0 and b > 0 the state will end up in the uninitialized attractor. This specific attractor is due to the symmetry of the 3 original target states along with the symmetry of the problem due to the 2 neurons giving 4 possible states. After this initialization, due to the inertia the Hopfield network has, it will update almost all new states to these attractors. Of note are the points [0, 1], [1, 0], [-1, 0], [0, -1] which are saddle points since one neuron gets a sign(0) while the other gets activated/deactivated. The last attractor is the [0,0] point where both neurons yield 0 and is a local maximum since any deviation from it along any direction yields to one attractor. From the number of iterations needed in my experiments around 23 iterations for my 22 points. The final state of the Hopfield network is in figure 1 and figure 2 shows its energy evolution.

2.1.2 Do the same for a three neuron Hopfield network, this time for the target patterns [1,1,1], [−1,−1,1] and [1,−1,−1].

The 18 points I input in the symmetric case, arrive in 7 different, apparently stable points, symmetrical to the 3 apexes of the triangle that the original triangle created and reached full training in 5 steps. The reduction from the possible 8 to the realistically 7 stable points occurs due to the symmetry between the neurons. The final state is shown in Figure 3 and its energy evolution in Figure 4.

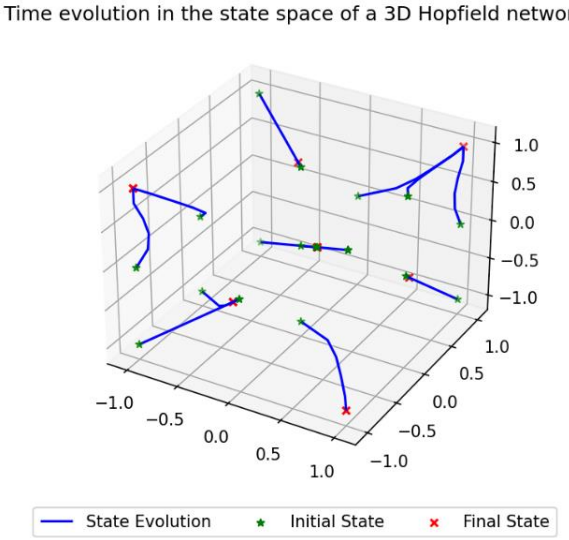

Figure 3 Final state of Hopfield network 3D

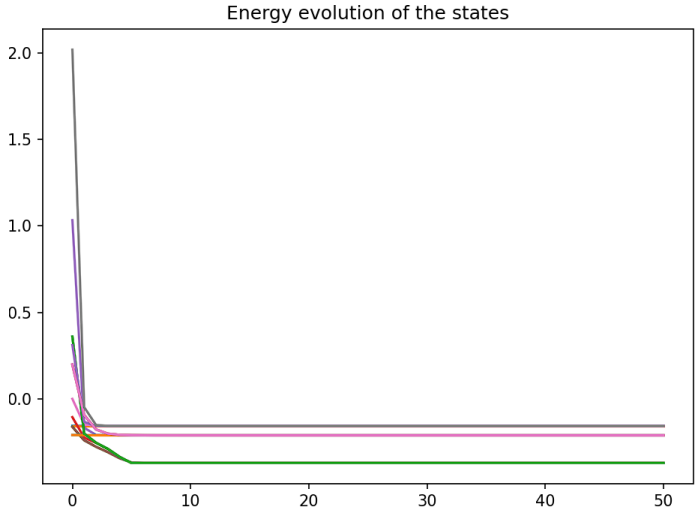

Figure 4 Energy evolution of Figure 3

2.1.3. Create a higher dimensional Hopfield network which has as attractors the handwritten digits from 0 to 9. Test the ability of the network to correctly retrieve these patterns when some noisy digits are given as input to the network. Try to answer the below questions by playing with these two parameters: noise level represents the level of noise that will corrupt the digits and is a positive number. num iter is the number of iterations the Hopfield network (having as input the noisy digits) will run. Is the Hopfield model always able to reconstruct the noisy digits? If not why? What is the influence of the noise on the number of iterations?

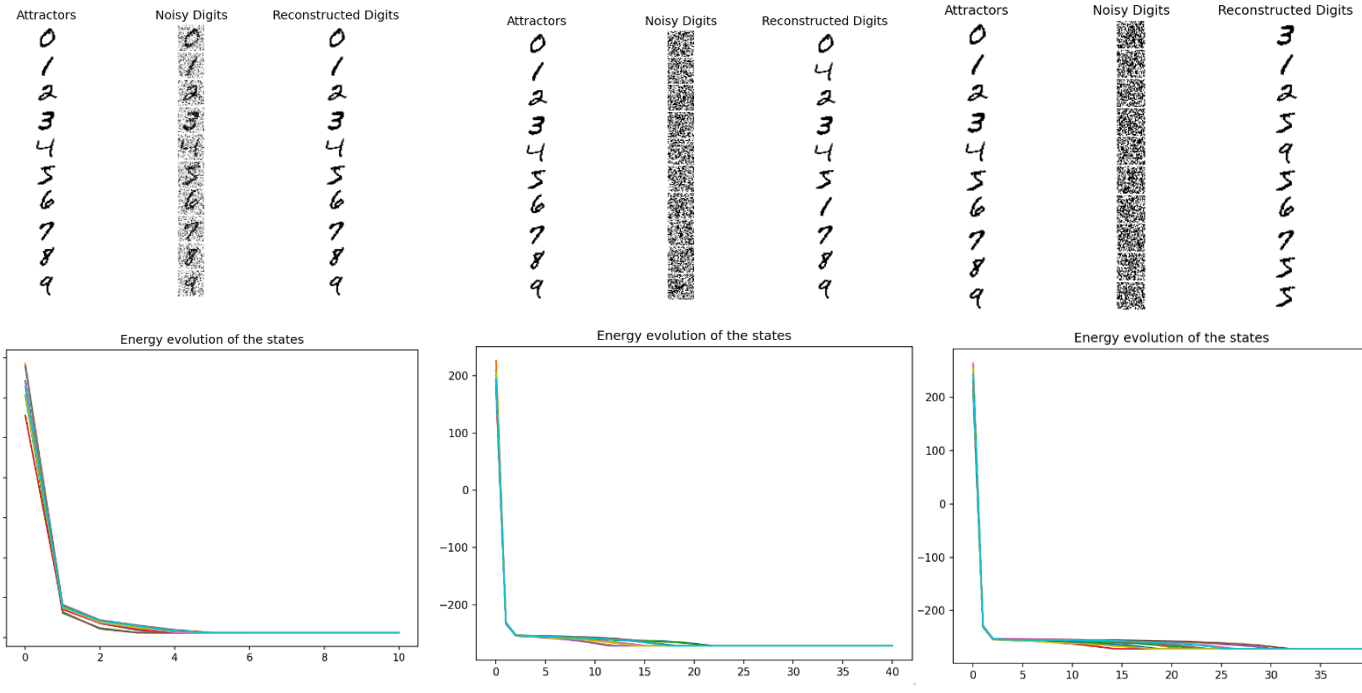

Figure 5 (Upper) Digits at noise levels 1(left), 5(middle) and 10(right). (Lower) Energy evolution of the respective states above.

We notice that the number of iterations to reach a plateau increases as the noise increases. If we consider the 10 digits images as the number of neurons, we have 28x28=784 number of neurons trying

to reach only 10 initial stable states. Adding noise to the digits moves their representation in the 784-dimensional space and makes it more likely, the more noise they have, for the Hopfield network to converge on another stable attractor. Above are figures with noise levels of 1, 5 and 10 for reference. We see as the noise level increases so the do the iterations and the worse is the accuracy.

2.2.1 Train an MLP with one hidden layer. Investigate the model performance with different lags and number of neurons. Discuss how the model looks and explain clearly how you tune the parameters and what the influence on the final prediction is. Which combination of parameters gives the best performance (MSE) on the test set?

Mean/Std	10 Neurons	30 Neurons	50 Neurons	75 Neurons	100 Neurons	Mean
10 Lag	*3760/678*	*3460/244*	*4055/1846*	*4524/ 1508*	*5046/2108*	***4169***
30 Lag	*4101/320*	*2533/1013*	*2499/1174*	*1305/908*	*2400/1628*	***2568***
50 Lag	*2192/485*	*930/369*	*1028/268*	*627/218*	*836/99*	***1122***
75 Lag	*1777/148*	*1191/306*	*820/651*	*655/586*	*675/539*	***1024***
100 Lag	*1536/845*	*909/721*	*751/262*	***471/60***	*499/208*	***833***
Mean	***2673***	***1805***	***1831***	***1516***	***1891***	

Test were done with a validation size of 50 on 4 folds with 1e-2 learning rate. Early stopping of 7 and reducing learning rate on plateau with patience of 3 by a factor of $1/\sqrt{10}$ was used. I achieved my best results with 75 neurons and a lag of 100 timesteps. We can see in general the more the lag is the better the model learns, something which is expected given the more associations it can make. However, no such trend is observed clearly with the neurons. My best experiments came from the model with 75 neurons, which seems to run better than the one with 100 neurons, probably due to overfitting.

2.2.2 Do the same for the LSTM model and explain the design process. What is the effect of changing the lag value for the LSTM network?

Mean/Std	10 Neurons	30 Neurons	50 Neurons	75 Neurons	100 Neurons	Mean
10 Lag	*3720/360*	*3011/1332*	*2956/1217*	*3359/739*	*3140/517*	***3237***
30 Lag	*1217/64*	*822/2961*	*725/591*	*432/221*	*1244/1272*	***888***
50 Lag	*603/135*	*274/56*	*331/95*	*603/564*	*231/35*	***408***
75 Lag	*379/83*	***210/41***	***222/57***	*471/425*	*433/314*	***343***
100 Lag	*741/214*	*451/101*	*466/76*	*652/281*	*483/38*	***559***
Mean	***1332***	***954***	***940***	***1103***	***1106***	

I run the same experiments with the only difference of using a learning rate of 1e-3. We can see that the LSTM is much more resistant to lower lags, due to the memory gating mechanisms it contains. It can discern the important features in the timeseries, store them and use them later, or forget less useful ones. We can see however, that there is an optimal neuron number for the LSTM architecture, mainly because by increasing the number of units, you increase the capacity of the states, allowing to remember more information and leads to overfitting on the training data.

2.2.3 Compare the results of the recurrent MLP with the LSTM. Which model do you prefer and why?

Each MLP does not consider the sequential nature of the time series, thus it is more sensitive to lag, since if it's too little, the model will miss important associations that the larger lag had and will be able to train less. LSTMs given their internal memory states with gates, can capture and update long-term temporal dependencies Due to that, I think they are more resistant to lag. LSTM are also more robust as to the number of neurons. Since they show they can overfit this amount of data with so few neurons, they will able to handle larger datasets with much better ease than the MLP which showed a stable and worse result across the board. Below you will find a graphical representation of a test done on 30 neurons with 75 lag that clearly shows the superiority of LSTM in Figure 6. Below that you will find a

plot of the lags against the neurons that also shows the supremacy of the LSTMs in Figures 7 and 8. The code to produce the plots was produced by GPT4.

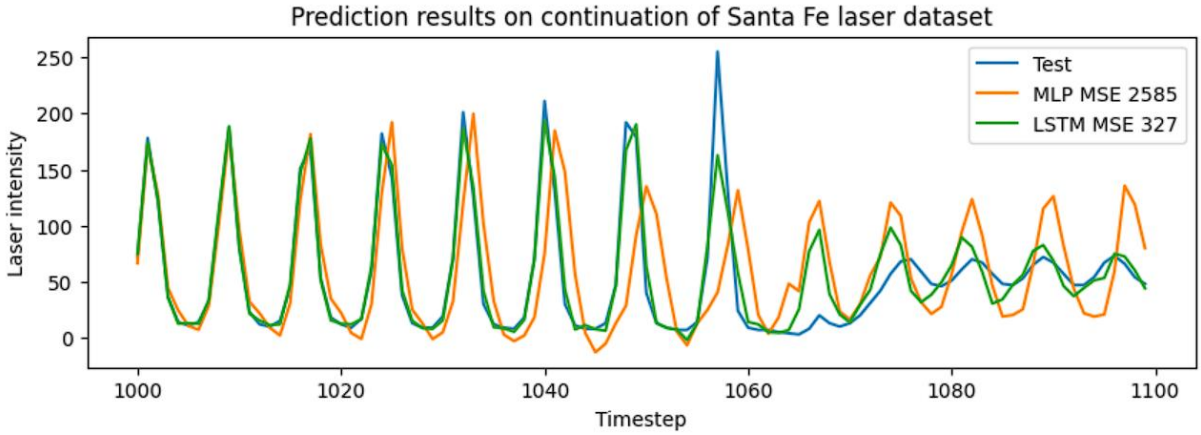

Figure 6 Comparison of the MLP and the LSTM on the test set

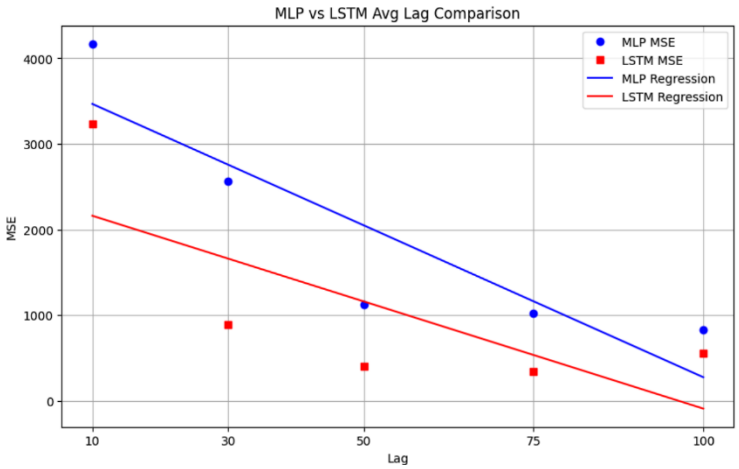

Figure 7 Comparison of the averages of the MLP vs LSTM respective to Lag

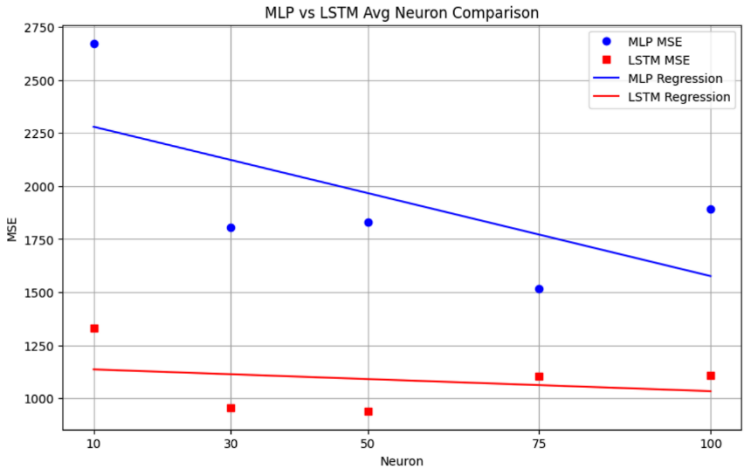

Figure 8 Comparison of the averages of the MLP vs LSTM respective to Neurons

ANN Exercise Session 3
Dimitrios Chatzakis r0977164

3.1.1. Conduct image classification on MNIST using a stacked autoencoder. Are you able to obtain a better result by changing the size of the network architecture? What are the results before and after fine-tuning? What is the benefit of pretraining the network layer by layer?

To discern the behavior of the architectures I tried 4 experiments with the default learning rate, changing the training epochs from 10/10/20 to 20/20/50 and tried the 4 different parameter sets for the first and second hidden layer respectively. The first number of each cell is the mean and the second the std from 4 experiments on each.

FILTERS	EXPERIMENT TYPE	64/16	128/32	256/64	512/128
ACCURACY AFTER	*20/20/50*	*94.2/0.9*	*94.0/0.7*	*94.0/0.7*	*94.0/0.8*
FINETUNING	*10/10/20*	*92.1/0.8*	***92.7/1.1***	*91.4/1.2*	*91.0/0.9*
ACCURACY	*20/20/50*	*23.7/8.6*	***29.6/8.1***	*17.0/4.7*	*16.7/4.8*
BEFORE	*10/10/20*	*19.6/5.2*	*14.7/4.8*	*20.3/11.8*	***20.8/7.8***

The results before finetuning are of course vastly worse, since there has been no training with backward connections of the whole network therefore layers not connected have no way to transmit information between them.

The model by itself seems to reach the same accuracy when trained by a large enough number of epochs, but it achieves it faster apparently in smaller models. In the case of pretraining accuracy, it achieves higher results in smaller models. We can see the bigger models reducing in accuracy over so many epochs, probably due to overfitting, while the smaller ones train better.

The benefits of pretraining the network are I think first and foremost in improved weights being initialized, which then it helps escape local minima and reach faster convergence, also it hinders the vanishing gradient problem (as the number of hidden layers is increased, the amount of error information propagated back to earlier layers is dramatically reduced). Lastly, it is efficient computationally, since it is much easier to train more shallow networks than a fully connected bigger one.

3.2.1 Calculate the output of a convolution with the following 2x2 kernel with no padding and a stride of 2. How do you in general determine the dimensionality of the output of a convolutional layer? What benefits do CNNs have over regular fully connected networks.

The dimensionality is given by the equation of: $dim_{out} = \frac{dim_{in}+2*padding-kernel\ size}{stride} + 1$

With the convolution operation given by:

$$Conv(img1, img2) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} img1(u,v) img2(i-u, j-v)$$

We have $\begin{bmatrix} 2 & 5 & 4 & 1 \\ 3 & 1 & 2 & 0 \\ 4 & 5 & 7 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 6 & 11 \end{bmatrix}$.

The benefits of using convolutional layers are many. Firstly, they increase computational efficiency, since for a similar size of units e.g. 16 concerning an image of size (32,32,3) with a dense layer with 16 units would create (32 * 32 * 3 + 1) * 16 = 49168 parameters with the +1 for the bias while a convolutional layer with a 5x5 kernel would output (5 (kernel width) * 5 (kernel height) * 3 (number of channels) + 1(bias) * 16(filters) = 1216 parameters, a huge reduction in complexity.

Secondly, it can detect features irrespective of the place inside the image due to the translational invariance of the operation.

3.2.2 The file cnn.ipynb runs a small CNN on the hand written digits dataset (MNIST). Use this script to investigate some CNN architectures. Try out different amounts of layers, combinations of different kinds of layers, number of filters and kernel sizes. Note that emphasis is not on experimenting with batch size or epochs, but on parameters specific to CNNs. Pay close attention when adjusting the parameters for a convolutional layer as the dimensions of the input and output between layers must align. Discuss your results. Please remember that some architectures will take a long time to train.

Paradigms tested:
P1: Vanilla,
P2: Kernel 5x5
P3: Removing the last odd-kernel convolutional layer, kernel 3x3
P4: Removing the last odd-kernel convolutional layer, kernel 5x5
P5: Adding a convolutional layer with kernel 3x3 same in units as the last one with odd kernel
And then redid the above again for double the units and then the half.

PEAK ACCURACY	P1	P2	P3	P4	P5	MEAN
X2 UNITS	*98.45*	*98.50*	*98.25*	*98.80*	*98.60*	***98.52***
INITIAL	*98.00*	*97.85*	*98.00*	*98.50*	*98.30*	***98.13***
%2	*97.80*	*98.10*	*97.55*	*98.15*	*97.40*	***97.80***
MEAN	***98.08***	***98.15***	***97.93***	***98.48***	***98.10***	

From the above the means for the 3x3 kernel: 98.01 and for the 5x5: 98.32.
Doing the same for number of layers yielded for 3 layers: 98.21, 4 layers: 98.12, 5 layers: 98.1.
The overall trends are that as the units increase so does accuracy because the model can create more filters, capturing more features, and thus learning better the underlying distribution. As the kernels increased, so did the accuracy, which is reasonable since the filters saw a bigger region of the input region and could augment/inhibit the parts that were (un)important. Lastly, regarding the number of layers, the model achieved pretty similar results. I would expect that as the model increased in depth the results would hit a peak and then fall off due to overfitting, so this behavior I would argue is due to there being too few experiments or due to the simplicity of the dataset.

3.3.1 Please run both the NumPy and PyTorch implementations of the self-attention mechanism. Can you explain briefly how the dimensions between the queries, keys and values, attention scores and attention outputs are related? What do the query, key and value vectors represent?
If we start from a sequential data of seq_len (to match the notebook), we input it into the transformer, the transformer creates embeddings out of it of shape (seq_len, dim_input) for key, query and value inputs. These are then mapped into (seq_len, dim_q) for both value query and key pairs and (seq_len, dim_v) for value. These are then multiplied as in the manner before (QK^T) normalized by the root of dim_q for stabilizing the gradients and passes through a softmax for normalization, creating the attention matrix of shape (seq_len, seq_len) which practically is the context dependent attention. That is then multiplied by the values matrix of length (seq_len, dim_v) to create the a context-dependent attention output that will be processed further on. This is mathematically depicted in (1).

$$SelfAttention(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1),$$

The multihead part comes into play, much like the convolutional layers have multiple filters (neurons) to create more filters. Practically many such attention mechanisms called heads are downsized (dim_input/n_heads) and calculated in parallel in order to capture different dependencies.

This architecture was inspired by how databases work in information retrieval. The query vector represents the element of interest or the context that you want to obtain information about, The key vectors are used to compute how relevant each element in the input sequence is to the query and the value is the information returned based on the previous two. I found this short that sums it up very well: “The Query seeks relevant information; the Key identifies it, and Value provides it.”. This mechanism has shown great success relative to older models such as LSTMs because it can capture long-term dependencies better, since it does not suffer from the vanishing gradient problem.

3.3.2 Please train the Transformer on the MNIST dataset. You can try to change the architecture by tuning dim, depth, heads, mlp dim for better results. You can try to increase or decrease the network size and see whether it will influence the prediction results much. Note that ViT can easily overfit on small datasets due to its large capacity. Discuss your results under different architecture sizes.

I kept the default settings of initial query/key dimensions dim=64, number of transformer layers depth=6, heads=8 for the multi-head attention mechanism and mlp_dim=128, the dimensionality of the feedforward layer after each multihead attention, and learning rate. I changed the callback of the learning rate decay to 0.912 and set the epochs to 50 so that at the 50th epoch the learning rate would be 1 hundredth of its original. Then I varied each one of them as the following tables show:

QUERY/KEY DIMS	**32**	**64**	**128**
BEST ACCURACY	*98.34*	*98.72*	***98.83***

TRANSFORMER DEPTH	**3**	**6**	**12**
BEST ACCURACY	*98.36*	*98.54*	***98.77***

HEADS	**4**	**8**	**16**
BEST ACCURACY	*98.56*	***98.64***	*98.46*

MLP DIMS	**64**	**128**	**256**
BEST ACCURACY	*98.48*	*98.71*	***98.84***

We can observe that as each parameter increases, the accuracy value increases as well. That is to be expected since the model has more complex representations to learn from (heads), more units to process the data (depth, mlp_dims), however the impressive is that increasing the heads parameter from the default dropped the accuracy. I think this is due to the fact that the model has too many “filters” for such a small (each image has 28*28) initial parameters and simple dataset (the digits have simple geometries). I would expect the accuracy to continue increasing in bigger architectures.

ANN Exercise Session 4
Dimitrios Chatzakis r0977164

4.1.1 In the notebook, the training algorithm refers to the pseudo-likelihood. Why is that? What are the consequences regarding the training procedure?

Due to the intractability of the partition function in the case of RBMs, the idea came that it would be possible to compute the loss function of a conditional probability, which can be written as a ratio of a joint and a marginal probability, thus bypassing the intractable original partition function. The pseudolikelihood function: $p(x) = \prod_{i=1}^{N} p(x_i|x_{-i}) \Longrightarrow logp(x) = \sum_i^n logp(x_i|x_{-i})$ requires k*n evaluations where k is the possible states of each neuron and n number of neurons instead of the k^n you would normally need for the partition function. You calculate the value of a neuron, given its Markov blanket. This has been proven to be asymptotically consistent (regresses to the true distribution as the data samples approach infinity). Regarding the consequences to the training procedure it converges much faster but does not guarantee the true distribution, given that it is an approximation.

4.1.2 What is the role of the number of components, learning rate and number of iterations on the performance? You can also evaluate it visually by reconstructing unseen test images.

The number of components is the number of the hidden neurons, and the number of iterations is the number of epochs, namely the number of times all the data needs to pass through the models. Learning rate is the coefficient that is used to update the neuron weights. Below are some test examples. The last column is of course the original.

Keeping in mind that this technique is very sensitive to the tuning of the learning rate, we can see (Figure 1 – table above) that increasing the learning rate, fits the data less, by overshooting. Increasing it however has the same effect due to slower convergence. More neurons lead to better results as does the increase in iterations.

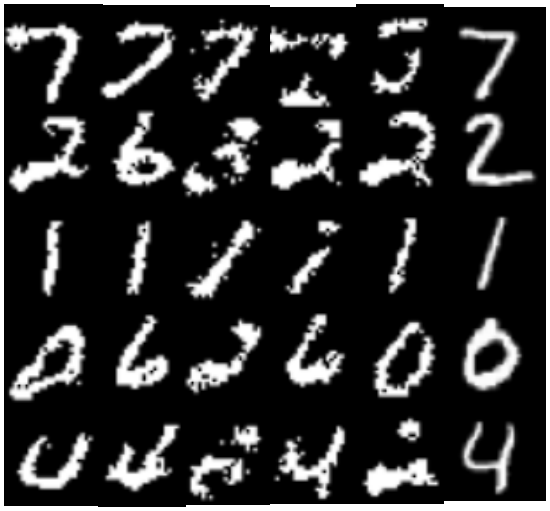

Figure 1 Created images for different settings

COLUMN NUMBER	1ST	2ND	3RD	4TH	5TH
NUM_COMPONENTS	784	784	784	100	100
LEARNING_RATE	0.01	0.1	0.001	0.01	0.01
N_ITER	10	10	10	10	100
PSEUDO VALUE AT END	-64.83	-82.8	-83.73	-82.63	-75.81

4.1.3 Change the number of Gibbs sampling steps. Can you explain the result?

Increasing the number of gibbs sampling, increases the iterations the network will perform in the test cases until it yields the result to us. That means more iterations, but not better results. On the right (Figure 2) is the best performing model (1st) with gibbs of 4, 100 and 1000. Due to having approximated the function, there is bias towards one digit. 1 is the simplest to learn and thus probably the model overfits to it, since it is the simplest building block for most digits, being a line. I checked and its not overrepresented in the train set.

Figure 2 Gibbs sampling steps increase

Figure 3 Removing rows results

4.1.4 – 4.1.5 Use the RBM to reconstruct missing parts of images. Discuss the results. What is the effect of removing more rows in the image on the ability of the net work to reconstruct? What if you remove rows on different locations (top, middle…)?

The accuracy regresses to 3/5 (upper right), 2/5 (lower left), 3/5 (lower right) in Figure 3 and that is to be expected. The middle contains more important information since it makes the image discontinuous and that messes with the model. I would also expect the top to be more important than the bottom, since there is usually more information on the shapes on the top.

4.1.6 Load the pre-trained DBM that is trained on the MNIST database. Show the filters (interconnection weights) extracted from the previously trained RBM (cf. supra) and the DBM, what is the difference? Can you explain the difference between filters of the first and second layer of the DBM?

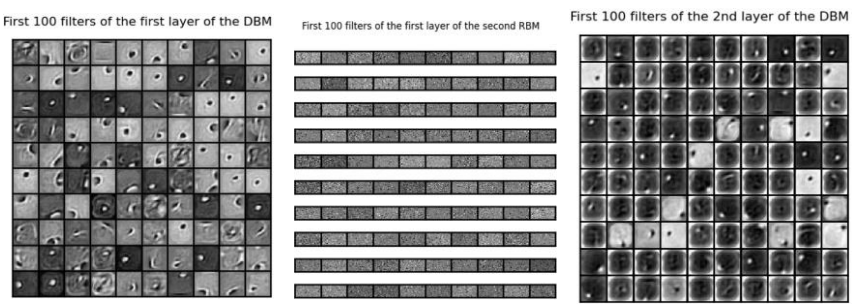

Figure 4 Reprentations of layers

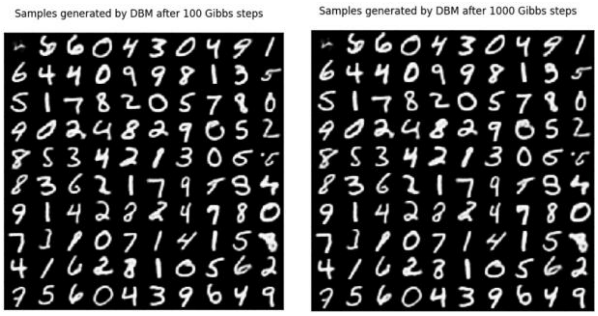

Figure 5 Results of DBM for different Gibbs sampling steps

The first layer of the dbm (Fig. 4 left) is the same as the weight of the rbm1, while the second one is the dot product of the first layer and the weights of the rbm2 which seem very noisy (Fig. 4 right). That is why the product of the two seems clear but less granular. It seems as if the first rbm has overfitted and the second underfitted.

4.1.7 Sample new images from the DBM. Is the quality better than the RBM from the previous exercise? Explain.
Here we can see (Figure 5) how much more robust to both Gibbs sampling variance the networks are as well as the much-increased quality of the reconstructed images given the reasons stated above.

4.2.1 Explain the different losses and results in the context of the GAN framework.
The loss of the discriminator is the unweighted of the binary cross entropies for the real images with the discriminator predictions and for the fake images with the discriminator predictions. The loss of the generator is the binary cross-entropy for the discriminator fake images' predictions and the real labels (wants to make the discriminator mistake fake images for real ones) they play the minimax game as defined by:

$$min_G max_D (E_x[\log(D(x))] + E_z[log(1 - D(G(z)))]$$

where G is the generator, D is the discriminator, x denotes real data, and z the random noise inputs to the generator. For results see below Figure 6. I could not make the vanilla GAN converge or the generator loss converge to zero.

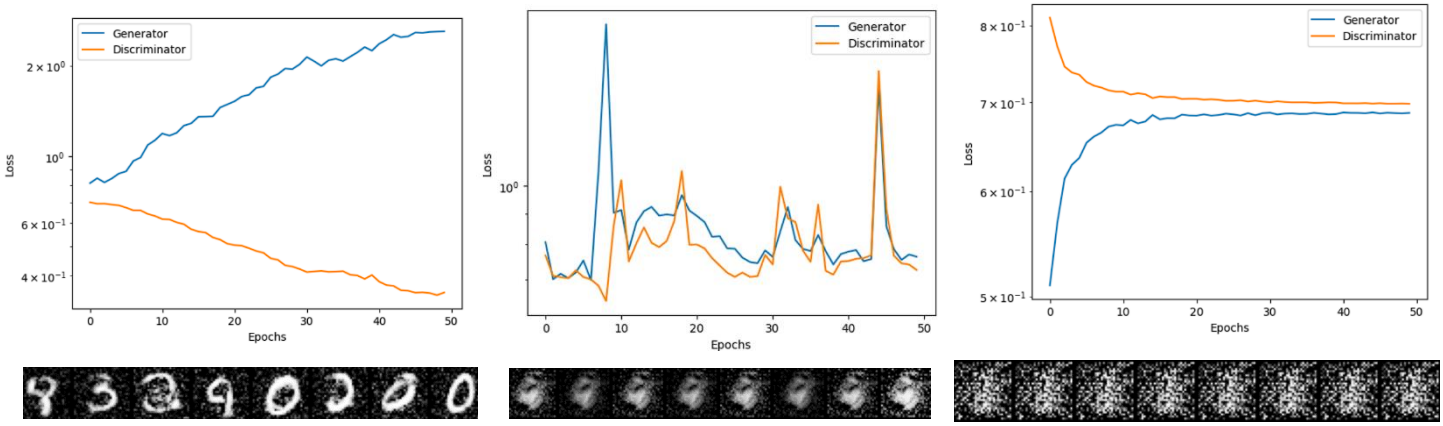

Figure 6 (left) Default settings, (middle) betas [0.5,0.9], (right) discriminator optimizer changed to SGD

The best results occur when the discriminator falls, even when the generator loss increases, resulting in 0s producing the same variety of inputs. When there are oscillations, as in the middle, the model cannot stabilize around a minimum and has haphazard performance. In the right subplot we see when the model has the proper behavior but not enough of it, so it is trapped in a local minima and outputs the same wrong image for every case. Ideally, we would expect the generator loss to rise and cross the discriminator loss before both stabilize.

4.2.2 What would you expect if the discriminator performs proportionally much better than the generator?
See immediately above, left subplot of Figure 6. It is called mode collapse as the model outputs a few samples and not the variety present in the initial dataset as we see above where the model outputs 3s, 0s and 9s.

4.2.3 Discuss and illustrate the convergence and stability of GANs.

GANs due to the inherent nature of minmax are prone to instability during training. If one of the two networks gets stuck in a local minima, it is very easy for the other one to overpower the training process and prevent the model from learning or generalizing well (underfitting and overfitting).

4.2.4 Explore the latent space and discuss.

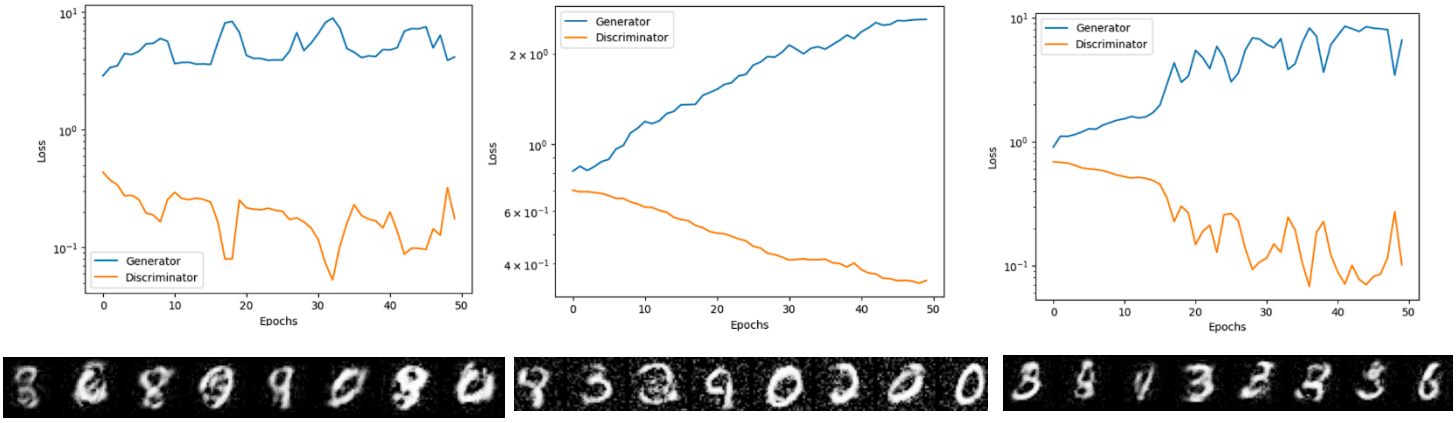

Figure 7 Latent spaces of 3 (left) 10 (middle), 784 (right).

From Figure 7 we can see that as the latent space increases, there is optimal value for the training of the GAN. Too few and the GAN will output only a subsample of the original data, (here for a latent space of 3 it output only the similar 3 digits 8, 9 and 0) while too large and the model will be too complex to converge. However, in the case where the input_dimensions are the same as the output dimensions, we have less sharp images but without noise, since the model undestands which pixels are background with high accuracy, failing to locate the ones consisting the right number.

4.2.5 Try the CNN-based backbone and discuss.

In Figure 8 we discern some interesting results. The GAN after having a very bad curve improves, and ends in a decent local minimum where there is less noise observed but not clear shapes for the digits in many cases.

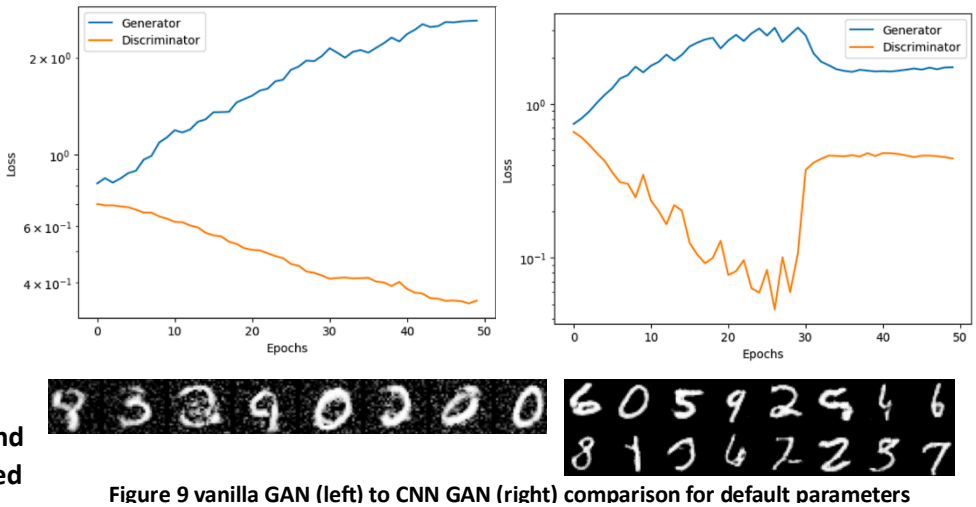

Figure 9 vanilla GAN (left) to CNN GAN (right) comparison for default parameters

4.2.6 What are the advantages and disadvantages of GAN-models compared to other generative models, e.g. the auto-encoder family or diffusion models?

I showcase in the table below the main differences between GANs and the other architectures.

4.3.1 In practice, the model does not maximize the log-likelihood but another metric. Which one? Why is that and how does it work?

VAEs are a probabilistic twist approach to the standard autoencoders. The bottleneck instead of being a dense layer, consists of 2 non-communicating dense layers that try to model the mean and the variance of the initial distribution along with a stochastic scaling Gaussian reparameterization trick to enable stochastic sampling while still allowing for backpropagation. That is why the model uses a loss metric that is the sum of the pixel-wise binary-crossentropy along with the KL divergence to impose a motivation to close the statistical distance between the original and the bottleneck layer's statistics:

$$D_{KL}(P|Q) = \sum_{x \in X} P(x) log \frac{P(x)}{Q(x)}$$

MODELS	DIFFUSION	GAN	AUTO-ENCODERS	DBM
ADVANTAGES	*High diversity images.* *High quality images.* *Stable training.*	*High quality images.*	*High diversity* *Very stable training.* *Relatively low amount of data needed.*	
DISADVANTAGES	*Extremely slow to train due to the diffusion process.* *Needs a lot of data.*	*Slow to train.* *Needs a lot of data.* *Extremely hard to train due to mode collapse and two losses that are not necessarily representative of the final results.* *Non-diverse reconstructions due to mode collapse.*	*Low quality blurred images due to the latent space being much smaller than the image and induces the model to predict an average of dark and bright images.*	*Difficult to scale.* *Slow to train.*

4.3.2 What similarities and differences do you see when compared with stacked auto-encoder from the previous assignment? What is the metric for the reconstruction error in each case?

Both models have the same architecture except the bottleneck. The SAEs use a single linear layer, and is thus less powerful computationally, and uses the cross-entropy loss only, whereas the VAEs use two linear layers that are not communicating and is trying to probabilistically reconstruct the underlying distributions, thus being more interpretable, and utilizing two loss functions. In the training process, I would argue that due to pretraining of the layers separately, SAEs have stabler training performance.

4.3.3 Explore the latent space using the provided code and discuss what you observe.

As observed in Figure 10, best training results lie in the highest latent space and lower as it decreases. Best test results lie between the 10 to 100 cases. The first produces digits more correctly shaped, but less impressed than the second. Lastly, I was impressed that in the lowest latent space, the output was not just 3 numbers as I would expect but many more, 9 in total, probably due to the combinations (3*3) that the models 2 non-communicating layers produce.

4.3.4 Compare the generation mechanism to GANs. You may optionally want to consider similar backbones for a fair comparison. What are the advantages and disadvantages?

GANs are unsupervised, use two models that compete, are unstable but produce sharp images due to the inherent non-probabilistic approach. VAEs are supervised, stable and probabilistic in nature, but produce low quality images due to the averaging in the KL divergence.

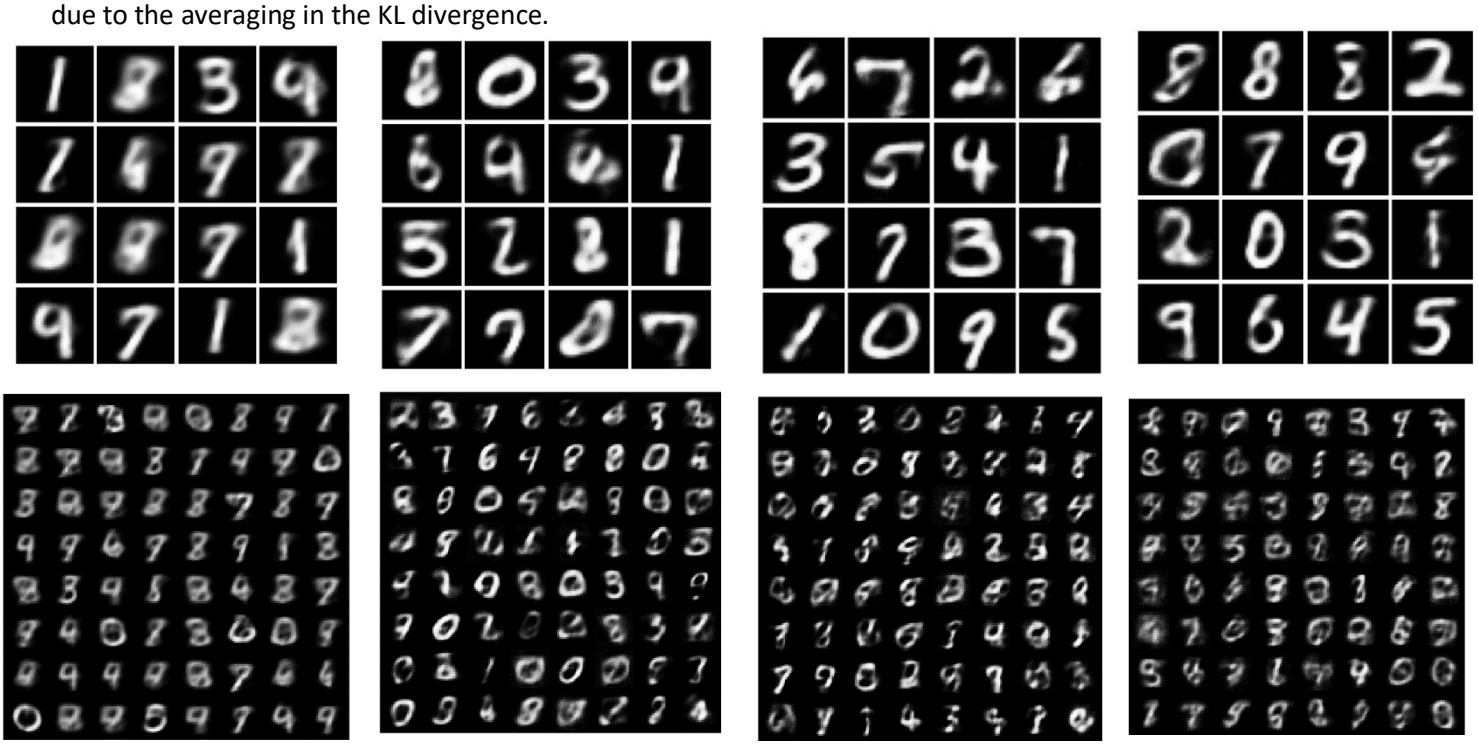

Figure 10 Comparison of different latent spaces (from left to right 3, 10, 100, 1000)

Use of ChatGPT (or any other AI writing assistance tool)

Form to be completed

Student name: **Dimitrios Chatzakis**

Student number: **r0977164**

Please indicate with "X" whether it relates to a course assignment or to the master thesis:

O This form is related to a **course assignment**.

Course name: Artificial Neural Networks and Deep Learning

Course number: H02C04a/H00G8a

Please indicate with "X":

O **I did not use** ChatGPT or any other AI writing assistance tool.

O **I did use** AI Writing Assistance. In this case **specify which one** (e.g. ChatGPT/GPT4/...):

I used GPT4/GPT4o

Please indicate with "X" (possibly multiple times) in which way you were using it:

Assistance purely with the language of the paper

- *Code of conduct*: This use is similar to using a spelling checker

As a search engine to learn on a particular topic -> X

- *Code of conduct*: This use is similar to e.g. a google search or checking Wikipedia. Be aware that the output of Chatbot evolves and may change over time.

For literature search -> X

- *Code of conduct*: This use is comparable to e.g. a google scholar search. However, be aware that some AI writing assistance tools like ChatGPT may output no or wrong references. As a

student you are responsible for further checking and verifying the absence or correctness of references.

For short-form input assistance

- *Code of conduct*: This use is similar to e.g. google docs powered by generative language models

To let generate programming code -> X

- *Code of conduct*: Correctly mention the use of ChatGPT (or other AI writing assistance tool) and cite it. You can also ask ChatGPT how to cite it.

To let generate new research ideas

- *Code of conduct*: Further verify in this case whether the idea is novel or not. It is likely that it is related to existing work, which should be referenced then.

To let generate blocks of text

- *Code of conduct*: Inserting blocks of text without quotes from ChatGPT (or other AI writing assistance tool) to your report or thesis is not allowed. According to Article 84 of the exam regulations in evaluating your work one should be able to correctly judge on your own knowledge. In case it is really needed to insert a block of text from ChatGPT (or other AI writing assistance tool), mention it as a citation by using quotes. But this should be kept to an absolute minimum.

Other

- *Code of conduct*: Contact the professor of the course or the promotor of the thesis. Inform also the program director. Motivate how you comply with Article 84 of the exam regulations. Explain the use and the added value of ChatGPT or other AI tool: ….

Further important guidelines and remarks

- ChatGPT cannot be used related **to data or subjects under NDA agreement.**
- ChatGPT cannot be used related **to sensitive or personal data due to privacy issues**.
- **Take a scientific and critical attitude** when interacting with ChatGPT (or other AI writing assistance tool)and interpreting its output. Don't become emotionally connected to AI tools.
- As a student you are responsible to comply with Article 84 of the exam regulations: your report or thesis should reflect your own knowledge. Be aware that plagiarism rules also apply to the use of ChatGPT or any other AI tools.

- **Exam regulations Article 84**: “Every conduct individual students display with which they (partially) inhibit or attempt to inhibit a correct judgement of their own knowledge, understanding and/or skills or those of other students, is considered an irregularity which may result in a suitable penalty. A special type of irregularity is plagiarism, i.e. copying the work (ideas, texts, structures, designs, images, plans, codes , ...) of others or prior personal work in an exact or slightly modified way without adequately acknowledging the sources. Every possession of prohibited resources during an examination (see article 65) is considered an irregularity.”
- **ChatGPT suggestion about citation**: “Citing and referencing ChatGPT output is essential to maintain academic integrity and avoid plagiarism. Here are some guidelines on how to correctly cite and reference ChatGPT in your Master's thesis: 1. Citing ChatGPT: Whenever you use a direct quote or paraphrase from ChatGPT, you should include an in-text citation that indicates the source. For example: (ChatGPT, 2023). 2. Referencing ChatGPT: In the reference list at the end of your thesis, you should include a full citation for ChatGPT. This should include the title of the AI language model, the year it was published or trained, the name of the institution or organization that developed it, and the URL or DOI (if available). For example: OpenAI. (2021). GPT-3 Language Model. https://openai.com/blog/gpt-3-apps/ 3. Describing the use of ChatGPT: You may also want to describe how you used ChatGPT in your research methodology section. This could include details on how you accessed ChatGPT, the specific parameters you used, and any other relevant information related to your use of the AI language model. Remember, it is important to adhere to your institution's specific guidelines for citing and referencing sources in your Master's thesis. If you are unsure about how to correctly cite and reference ChatGPT or any other source, consult with your thesis advisor or a librarian for guidance.”

Additional reading

ACL 2023 Policy on AI Writing Assistance: https://2023.aclweb.org/blog/ACL-2023-policy/

KU Leuven guidelines on citing and referencing Generative AI tools, and other information: https://www.kuleuven.be/english/education/student/educational-tools/generative-artificial-intelligence

Dit formulier werd opgesteld voor studenten in de Master of Artificial intelligence. Ze bevat een code of conduct, die we bij universiteitsbrede communicatie rond onderwijs verder wensen te hanteren.

Deze code of conduct schept een kader voor academiejaar 2023-2024, aanpassingen kunnen gebeuren in functie van nieuwe evoluties.