Artificial Neural Networks and Deep Learning

Exercise Report

Fan Zhou
r0822921
fan.zhou@student.kuleuven.be

May 31, 2023

The codes and images (.fig and .png formats) are available at https://github.com/louisefz/ANN

Exercise Session 1: Supervised Learning and Generalization

1.1 Backpropagation in feedforward multi-layer networks

1.1.1 Learning Algorithms Comparisons on Noiseless and Noisy Data

Eight learning algorithms are employed in feedforward networks. Prior to making comparisons, the optimal number of epochs and hidden units for the networks are explored. Figure 1 and Figure 2 exhibit two feedforward networks, one with 10 neurons and the other with 50 neurons, respectively, spanning 10 epochs. The range of epochs considered spans from 100 to 1000, with intervals of 100 epochs, all based on the training dataset. Upon fixing the number of epochs at 1000, the feedforward network comprising 50 neurons exhibits superior approximations when making predictions based on the test dataset, in comparison to the network with 10 neurons. Consequently, we proceed to investigate and compare various learning algorithms implemented in the feedforward network with 50 hidden units, trained over 1000 epochs.

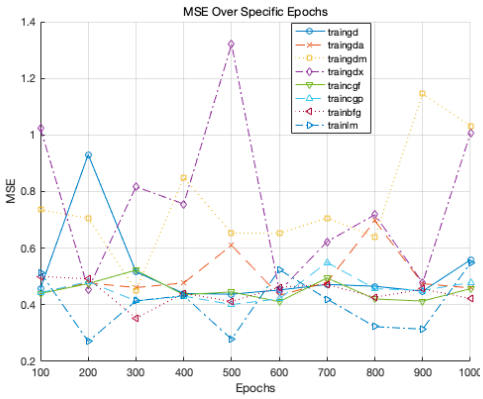

Figure 1 MSEs generated by a network with 10 neurons and 8 different algorithms over 1000 epochs on train dataset

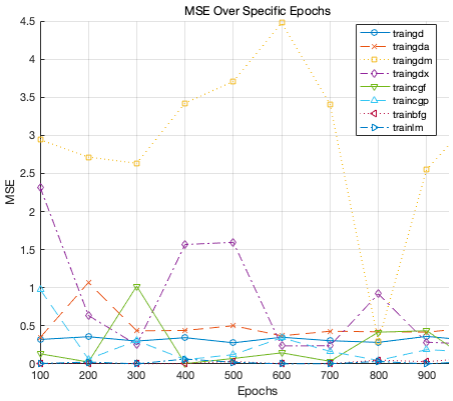

Figure 2 MSEs generated by a network with 50 neurons and 8 different algorithms over 1000 epochs on train dataset

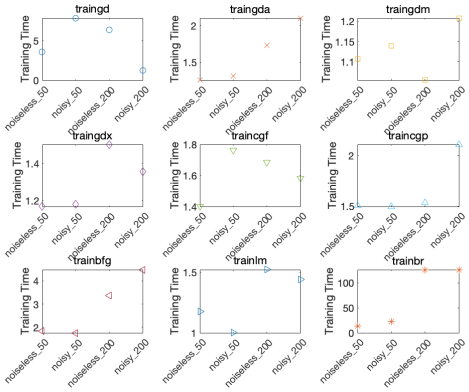

Figure 3 Convergence time of different algorithms

The noiseless data is generated according to the function y = sin(x^2), where x ranges from 0 to 3π with an interval of 0.05. In addition to the noiseless data, Gaussian noise with a standard deviation of 0.2 is incorporated to create the noisy data, based on the aforementioned formula. Both the noiseless and noisy datasets are divided into the default proportions of 70% for training data, 15% for validation data, and 15% for test data. During the evaluation process, the training time is measured using the training dataset. The Mean Squared Error (MSE) is calculated using the validation dataset, providing a measure of the network's performance. Lastly, the predictions made on the test dataset are analyzed to assess the model's effectiveness.

The training process exhibits variations in speed across different learning algorithms. In Figure 3, nine subplots are presented, specifically comparing the "noiseless_50" and "noisy_50" variables. These variables represent the time consumed by the training process on the noiseless and noisy training datasets, respectively, within the feedforward network comprising 50 hidden units over 1000 epochs. Based on these observations, several conclusions can be drawn: (1) When trained with the noiseless dataset, the traingda, traingdm, traingdx, and trainlm algorithms demonstrate high efficiency in convergence, taking approximately 1.3 seconds. On the other hand, traincgf, traincgp, and trainbfg require slightly longer durations, ranging from 1.5 to 2 seconds. Notably, traingd exhibits the longest training time of around 4 seconds. (2) Comparing the training on noiseless and noisy datasets, it is evident that many learning algorithms consume more time when trained on the noiseless data. Notably, traingd, traingda, traingdm, and traincgf algorithms display this behavior. The presence of noise introduces additional complexity and uncertainty into the learning process, potentially leading to suboptimal solutions and slowing down the convergence rate. However, the traingdx and traincgp algorithms exhibit the same training time on both the noisy and noiseless data. Furthermore, the trainbfg and trainlm algorithms require less time when trained on the noisy data. This can be attributed to the fact that noisy data can act as a form of regularization, simplifying the learning problem and facilitating quicker convergence. Moreover, these algorithms may focus on more robust features, avoiding overfitting and leading to faster convergence. In the

presence of noise, these algorithms adapt more readily to varying patterns, resulting in faster convergence when compared to the noiseless data.

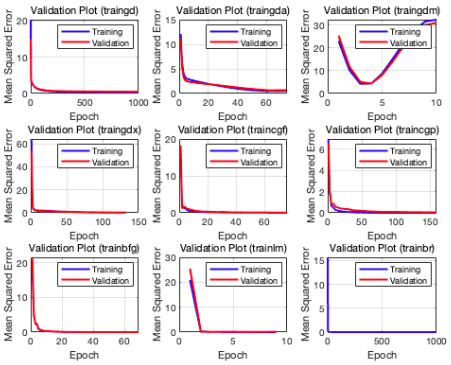

Figure 4 MSEs of different algorithms on the training and validation noiseless data sets.

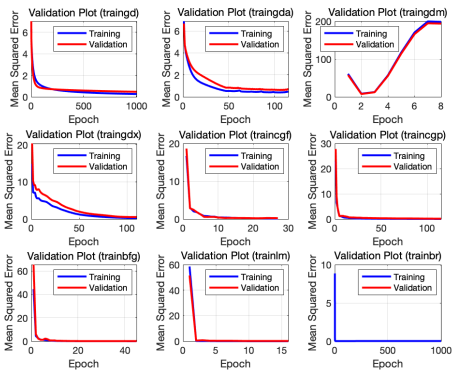

Figure 5 MSEs of different algorithms on the training and validation noisy data sets.

The feedforward network, employing different learning algorithms, exhibits varying MSEs on the validation dataset. Figure 4 and Figure 5 provide insights into these MSEs over both noiseless and noisy validation datasets, leading to several conclusions: (1) Regardless of the dataset being noiseless or noisy, the trainlm, trainbfg, traincgp, traincgf, and traingdx algorithms demonstrate strong convergence abilities with fewer epochs. The traingd algorithm requires more epochs for convergence, while traingda does not necessitate epochs for convergence but maintains relatively higher MSE values. Notably, the MSE of traingdm increases significantly with more epochs. (2) In the noiseless dataset, the traincgp algorithm initializes with a small MSE value, followed by trainlm, trainbfg, traincgf, traingda, traingdm, and traingd. Only traingdx's MSE starts significantly higher, exceeding 50. (3) When compared to the noiseless dataset, the network trained on noisy data requires more epochs to converge. (4) MSE values in the noisy datasets initially start higher than those in the noiseless datasets for certain algorithms, including traingdm, traincgp, trainbfg, and trainlm. Conversely, traingd, traingda, traingdx, and traincgf exhibit lower initial MSE values in the noisy dataset compared to the noiseless dataset. This observation can be attributed to certain algorithms having adaptive learning rates capable of adjusting to the noise present in the data. The introduction of noise as a form of regularization can facilitate better generalization to unseen data, resulting in lower initial MSE values.

Figure 6 illustrates the prediction curves generated by the feedforward network employing the aforementioned learning algorithms on the test dataset. The plots present a comparison between the predicted values for both the noiseless and noisy test datasets and their corresponding original test data. Several observations can be made: (1) In both the noiseless and noisy datasets, trainlm and trainbfg exhibit the highest prediction accuracy. (2) Traincgp performs better on the noiseless data compared to the noisy data. (3) Traincgf, traingdx, and traingd initially demonstrate good approximation capabilities, but their performance deteriorates as the prediction progresses. (4) Traingdm and traingda exhibit the poorest performance throughout the entire prediction process.

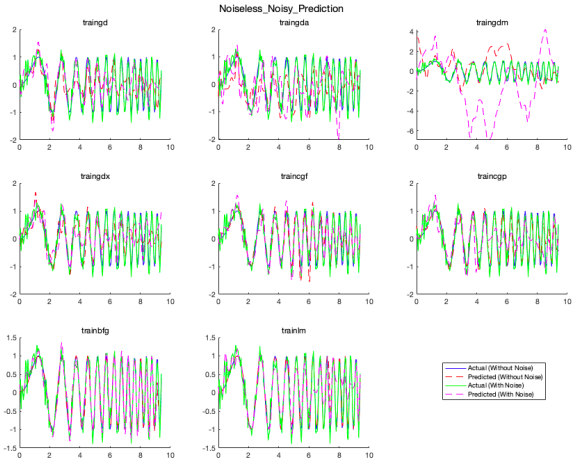

Figure 6 predicted test data on noiseless and noisy data sets comparing to original noiseless and noisy test data sets

1.1.2 Personal Regression

The data used in the experiment is a random sample obtained from a larger dataset consisting of 13,600 data points. For this specific experiment, a subset of 3,000 data points is selected as the experimental dataset. The training, validation, and test datasets are each composed of 1,000 data points, carefully chosen to ensure there is no overlap between the datasets. In Figure 7, a visualization of the surface of the training dataset is presented.

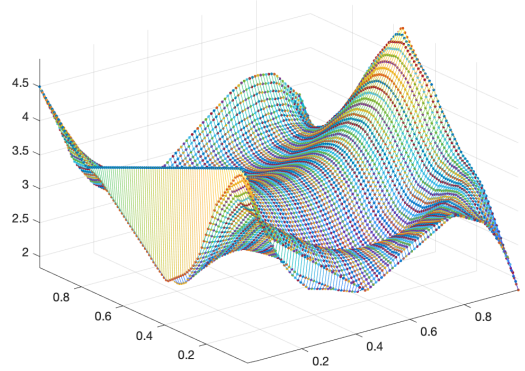

Figure 7 the surface of train data set

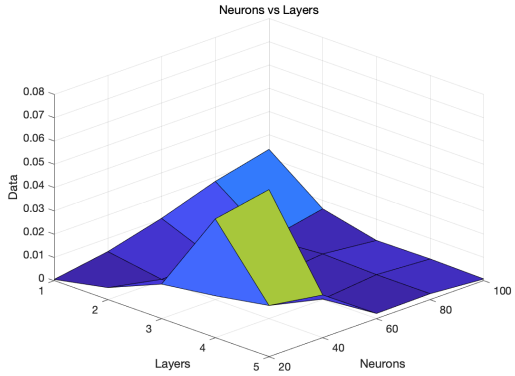

Figure 8 grid search for best numbers of neurons and layers

To construct and train a feedforward neural network, several parameters need to be set, including the learning algorithm, the number of hidden layers and hidden units, and the choice of transfer function or activation function. (1) The Levenberg-Marquardt algorithm (trainlm) is selected as the learning algorithm for the network. This choice is based on previous analyses that revealed three advantages: faster convergence, adaptive learning rate adjustment based on the error surface curvature, and the use of the Jacobian matrix to approximate the Hessian matrix for more accurate weight updates. The trainlm algorithm is also known for its robustness in avoiding local minima and finding optimal solutions. (2) A grid search is performed to determine the optimal number of neurons and layers, guided by the MSEs on the validation sets. The grid search encompasses 5 layers ranging from 1 to 5, and 5 neurons ranging from 20 to 100 with an interval of 20. Therefore, a total of 25 combinations of layer and neuron pairs are tested, each run 10 times to obtain average MSE values. Based on this grid search analysis, an optimal set of hyperparameters is determined: 4 hidden layers and 60 hidden units, resulting in an MSE of 2.5477e-04 as depicted in Figure 8. (3) For the nonlinear regression problem at hand, the tanh transfer function is chosen for the network. The tanh function offers several advantages. Firstly, it is zero-centered, aiding in faster convergence during training and reducing bias in the network. Additionally, the smooth gradient of the tanh function helps alleviate the vanishing gradient problem, resulting in more stable and effective training.

After determining the optimal parameters, learning algorithm, and transfer function, the final performance of the model can be evaluated on the test dataset. The evaluation includes a visual comparison between the original test data surface and the predicted test data surface, as shown in Figure 9 and Figure 10, respectively. Visually, the two surfaces appear to be highly similar. However, when the squared Mean Squared Error (MSE) between the two surfaces is calculated numerically, it becomes apparent that there are certain data points on the periphery that do not perform well. Figure 11 presents a three-dimensional visualization of the errors observed in the predicted test data, highlighting the areas where the model exhibits deviations from the original data surface.

In order to make improvements for the network, there are some methods: (1) The proportion of the training, validation and test dataset can be adjusted, so that more data can be trained for feature learning. (2) Bayesian regularization or dropout regularization can be added so as to avoid overfitting.

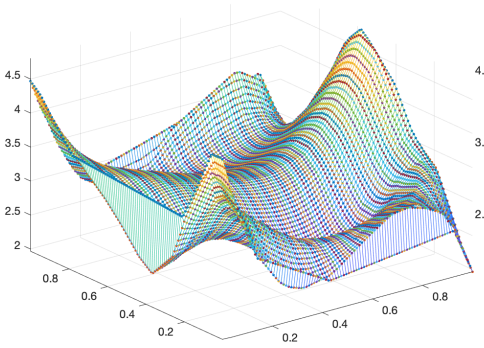

Figure 9 the surface of original test data

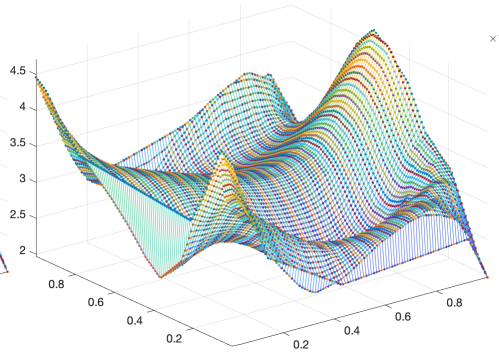

Figure 10 the surface of predicted test

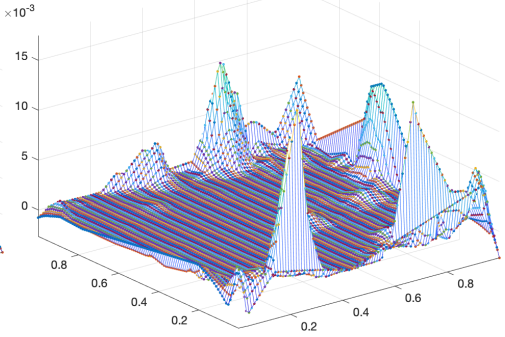

Figure 11 MSEs between original and predicted test data sets

2.1 Bayesian Inference of Network Hyper-parameters

Levenberg-Marquardt with Bayesian Regularisation (trainbr) is compared with previously analyzed eight learning algorithms in terms of converging time, MSE on validation sets, and the number of neurons.

In Figure 3, it can be observed that when employing the trainbr algorithm in a feedforward network with 50 hidden units and training over 1000 epochs, the training process requires a longer duration compared to other learning algorithms. Specifically, it takes approximately 8 seconds to train on noiseless data and 18 seconds on noisy data. The extended training time can be attributed to the utilization of the "trainbr" function in MATLAB, which implements the Bayesian regularization algorithm. This algorithm involves additional computations and iterations to strike a balance between fitting the training data effectively and maintaining small network weights to prevent overfitting. To achieve this, a regularization term is incorporated into the error function, penalizing large weight values. The algorithm iteratively adjusts the weights while considering the regularization term to find the optimal weights. Additionally, when the network is over-parameterized with 200 neurons, trainbr takes even longer to converge, reaching approximately 120 seconds.

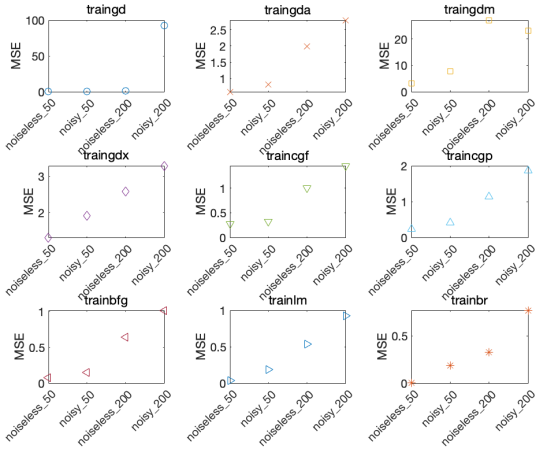

Figure 12 MSEs on noiseless and noisy test data sets to compare normal and over-parameterized networks

When the network consists of 50 neurons, the trainbr algorithm demonstrates performance that is comparable to the best algorithm, trainlm, in terms of MSEs on the test dataset, as depicted in Figure 12. However, when the network is over-parameterized with 200 neurons, trainbr exhibits the lowest MSEs on the test dataset among all the learning algorithms. Specifically, the MSE generated by the network with trainbr, trained on the noiseless data, is approximately 0.25, whereas on the noisy data, it is approximately 0.7. These values are lower than those obtained by trainlm and trainbfg. This outcome can be attributed to the Bayesian regularization algorithm employed by trainbr, which incorporates a regularization term into the error function. By penalizing large weight values and promoting simpler models, this regularization aids in mitigating overfitting, even in the case of an over-parameterized network.

Exercise Session 2: Recurrent Neural Networks

2.1 Hopfield Networks

2.1.1 Approximate Attractors

In the two-dimensional space, 50 randomly initialized points in the Hopfield network exhibit convergence towards both the desired attractors and an undesired one ([-1,1]) over 19 iterations, as depicted in Figure 13. Several factors contribute to the generation of these unwanted attractors: (1) The presence of noise or external perturbations within the network can disrupt convergence towards the desired attractors and introduce additional attractors. These extraneous attractors may arise due to the amplification or propagation of noise through the network's dynamics. (2) The architecture and parameters of the Hopfield network, including the number of neurons, connectivity pattern, and choice of activation function, can influence the emergence of undesired attractors. Improper configuration of these parameters can result in unintended dynamics and attractor landscapes. (3) When the attractors in the network are not perfectly orthogonal or independent, they can overlap within the state space. Such overlap can lead to the emergence of novel attractors that are combinations or superpositions of the original ones. Although unintended, these new attractors arise due to the non-linear dynamics of the network. (4) In a Hopfield network, the weights are typically learned using a Hebbian learning rule, where weight updates depend on the correlation between the desired attractors. If the training process is insufficient or lacks thoroughness, the learned weights may fail to accurately capture the desired attractors, thereby resulting in the emergence of unwanted attractors.

In a three-dimensional space, 50 randomly initialized points in the Hopfield network converge to all the desired attractors within 46 iterations, as depicted in Figure 14.

The stability of attractors in a Hopfield network is influenced by various factors: (1) Accurate capture of the desired attractors is facilitated by well-trained weights; (2) The presence of minimal initial state perturbations or noise; (3) Attractors with larger basins of attraction exhibit higher stability, as they can accommodate a broader range of initial states.

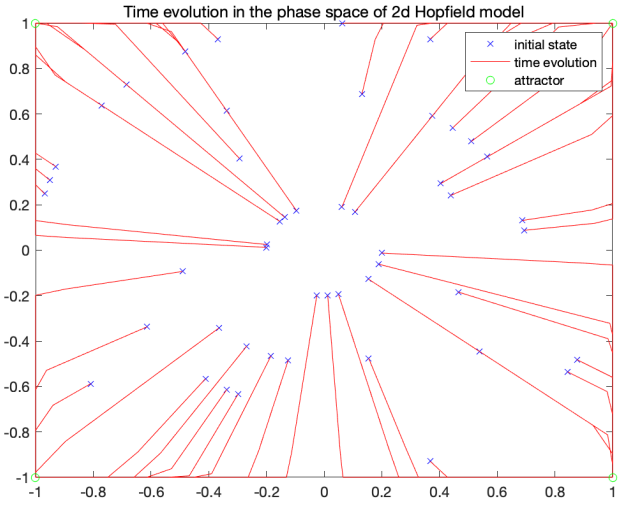

Figure 13 2D trajectory evolutions

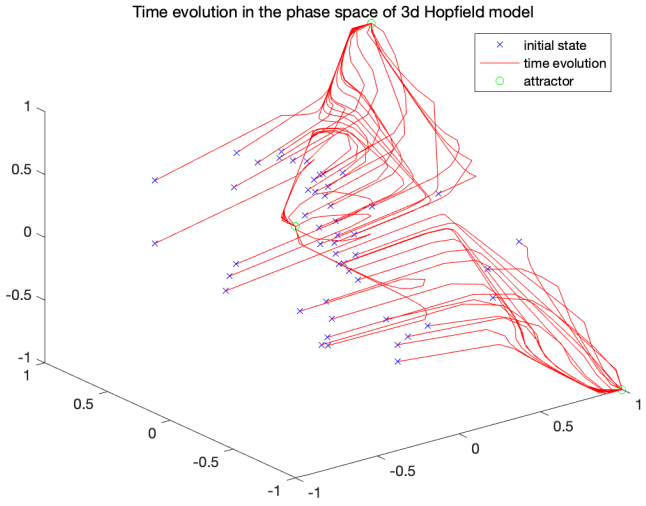

Figure 14 3D trajectory evolutions

2.1.2 A Hopfield Network for Handwritten Digits

The introduction of higher noise levels poses challenges for Hopfield network's ability to reconstruct digits. Specifically, when the noise exceeds a threshold of 4, the quality of reconstruction significantly deteriorates. Conversely, increasing the number of iterations can enhance the network's performance. Nevertheless, an extended number of iterations does not guarantee precise reconstruction. Figure 15 illustrates the reconstruction of digits obtained from the original ones, considering six levels of noise, after undergoing 1000 training iterations.

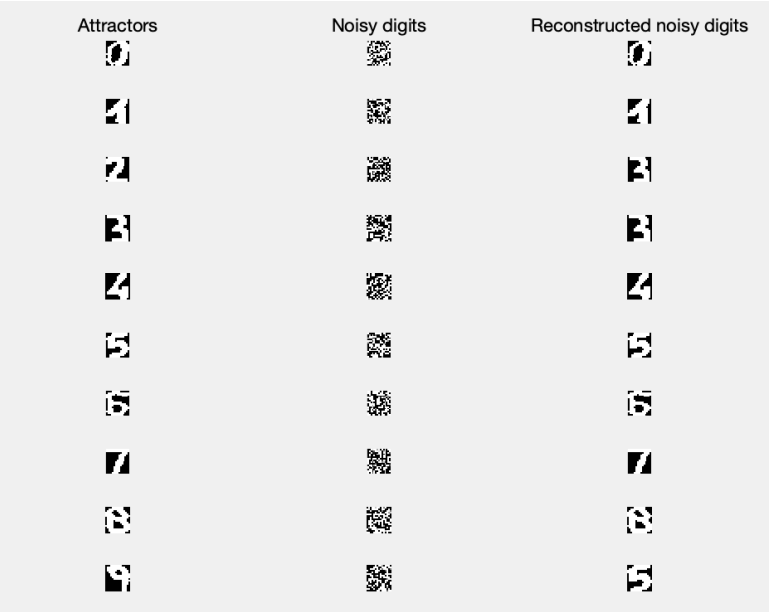

Figure 15 digit reconstruction with noise of 6 and iteration of 1000

2.2 Long Short-Term Memory Networks

2.2.1 Time-Series with a Multi-layer Perceptron

A one-layer multi-layer perceptron is trained using the Levenberg-Marquardt algorithm (trainlm), known for its accuracy and fast convergence. The network undergoes 1000 epochs of training, exploring different lags and varying numbers of neurons in the hidden layer.

The investigation includes lags ranging from 1 to 50 while keeping the number of neurons fixed at 50. In Figure 16, the root mean square error (RMSE) exhibits minor fluctuations between 0.5 and 2, except for pronounced RMSE values at lags 10, 13, and 44. Additionally, in Figure 17, when the lag is set to 50 and the number of neurons varies from 1 to 50, the RMSE ranges from 0.6 to 1.5. Notably, when the number of neurons is 45, the RMSE reaches a minimum value of 0.6.

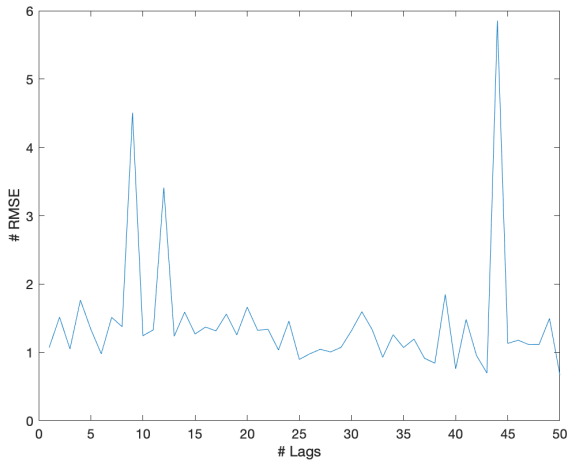

Figure 16 RMSEs over lags when neurons is fixed at 50

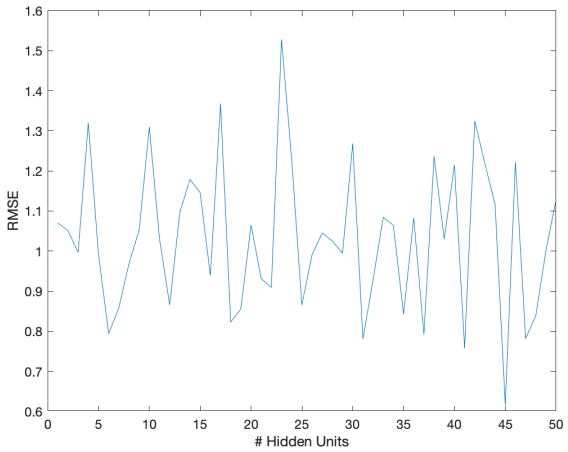

Figure 17 RMSEs over neurons when lags is fixed at 50

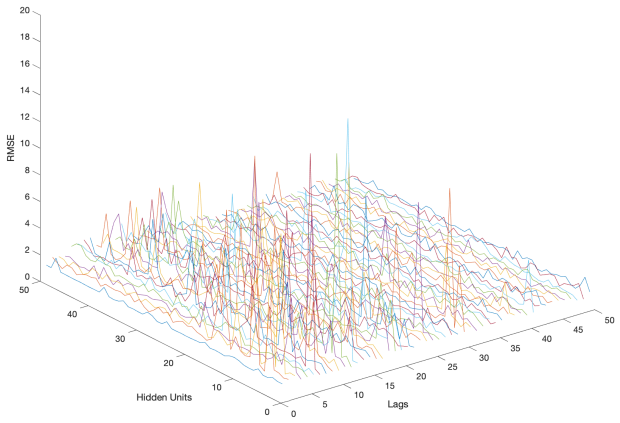

Figure 18 RMSEs over different hidden units and lags

Considering the combined investigation of different lags (1-50) and the number of neurons (1-50), a total of 2500 RMSE calculations are performed. As depicted in Figure 18, a reduction in either the number of lags or the number of neurons, or a reduction in both, leads to higher RMSE values. Figure 19 and Figure 20 illustrate

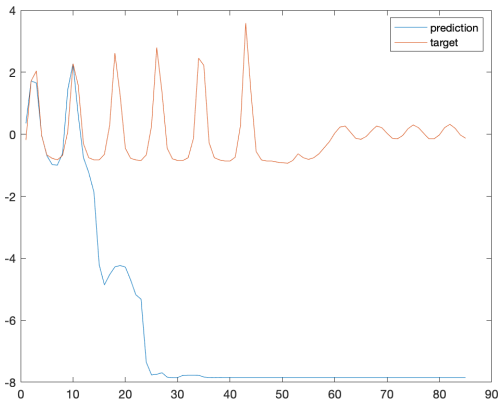

Figure 19 bad prediction and target curves comparison when lags is 15 and neurons is 19, RMSE is 6.8513

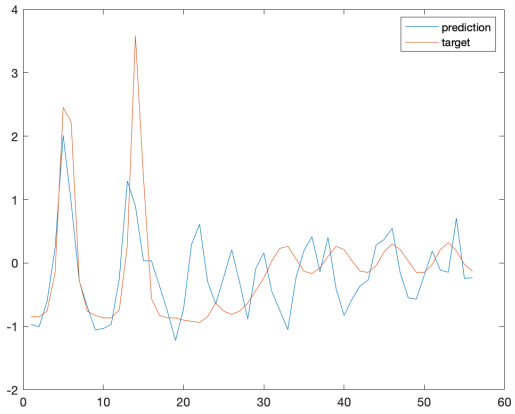

Figure 20 good prediction and target curves comparison when lags is 44 and neurons is 46, RMSE is 0.6832

a case of both poor and good predictions on the test dataset.

2.3 Time Series with LSTM

LSTM, short for Long Short-Term Memory, is an architectural variant of recurrent neural networks (RNNs) designed to address the limitations of traditional RNNs in effectively capturing and retaining long-term dependencies in sequential data. Its core concept involves the incorporation of a memory cell, enabling selective retention or forgetting of information over time. The functioning of LSTM involves several key components. The input gate determines the elements from the current input and the previous hidden state that should be stored in the cell state. The forget gate decides which information in the cell state should be discarded. The cell state is then updated by combining the input and forget gates with the previous cell state. Finally, the output gate determines the portions of the updated cell state that should be output as the hidden state for the current time step. This hidden state carries relevant information for subsequent time steps or the final prediction.

To effectively leverage LSTM, several parameters can be fine-tuned: (1) the number of LSTM layers, which determines the model's depth and capacity to capture intricate patterns; (2) the number of LSTM units, which influences the model's ability to learn and represent complex relationships in the data; (3) the activation function applied to the output of each LSTM unit or gate, such as tanh or ReLU; (4) the learning rate, which

governs the step size utilized by the optimization algorithm to update the model's weights during training; (5) the optimizer, responsible for updating the model's weights based on the calculated loss, commonly including methods like SGD, Adam, and RMSprop; (6) the lag value, indicating the number of past observations employed for future prediction and impacting the model's capacity to capture short-term or long-term dependencies within the data; (7) the number of epochs; and (8) the batch size.

Increasing the lag value in LSTM allows for a broader inclusion of past observations, enhancing the model's ability to capture intricate patterns and dependencies in the data, particularly in tasks demanding long-term memory. However, this necessitates a higher memory capacity and computational complexity. Conversely, employing an excessively large lag value may lead to overfitting, particularly when the data exhibits noise or when the underlying patterns undergo changes over time. In such cases, the model may excessively rely on past observations, impeding its generalization to unseen data.

For achieving a robust fit, an LSTM architecture comprising 50 hidden units, a fully connected layer, and a regression layer is trained using the Adam optimizer over 1000 epochs. The prediction outcomes and associated errors are visually depicted in Figure 21. Notably, the updated network, benefiting from continuous refinement after each prediction, exhibits enhanced accuracy in forecasting the decline of the laser-induced signal, as illustrated in Figure 22.

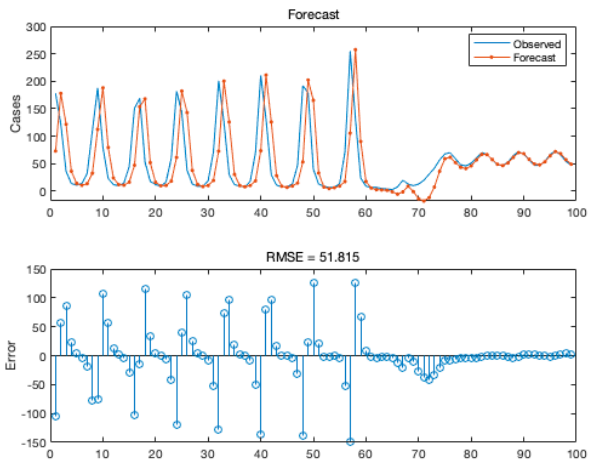

Figure 21 prediction and RMSE by the normal network

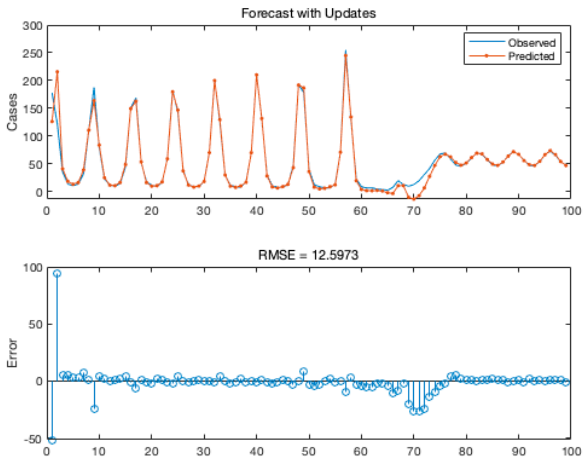

Figure 22 prediction and RMSE by the upgraded network

Compared to RNN, LSTM exhibits superior performance in the following aspects: (1) LSTM yields significantly lower training errors, with an average of approximately 0.08, surpassing the error rates obtained by RNN in the previous exercise. (2) When handling lengthy sequences of information, LSTM demonstrates efficiency by requiring fewer computational resources. This efficiency is achieved through the utilization of memory cells, which selectively retain important information while discarding irrelevant details. (3) RNNs, lacking memory cells, encounter challenges associated with vanishing/exploding gradients. Consequently, they struggle to retain information over extended time intervals. In contrast, LSTM overcomes this issue, enabling efficient processing of shorter information segments while also accommodating larger lag values for extended information sequences.

Exercise Session 3: Deep Feature Learning

3.1 Principal Component Analysis

PCA, an abbreviation for Principal Component Analysis, is a dimensionality reduction technique employed to transform high-dimensional data into a lower-dimensional representation while preserving essential information. By identifying principal components, which are linear combinations of the original variables capturing the highest data variance, PCA effectively reduces data dimensionality. This is accomplished by computing eigenvectors and eigenvalues of the covariance or correlation matrix. Selection of eigenvectors with the largest eigenvalues ensures the retention of vital information while reducing dimensionality. The reconstruction error serves as a metric to evaluate the fidelity of the reduced representation compared to the original data.

When applying PCA to randomly generated data conforming to a Gaussian distribution, where limited correlation among input dimensions is expected, Figure 23 illustrates a notable trend. As the number of dimensions (eigenvalues) utilized for representation increases, the reconstruction error steadily decreases, eventually approaching zero when all dimensions are employed.

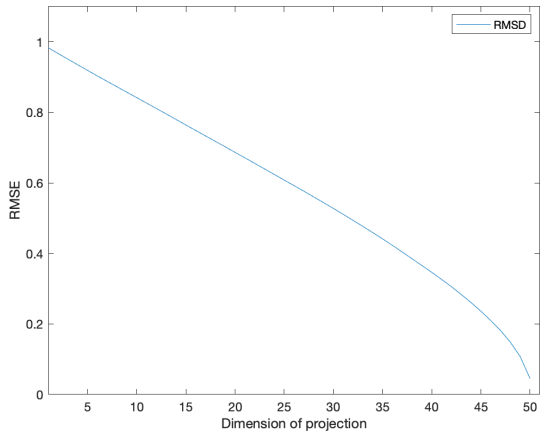

Figure 23 RMSEs over the increasing numbers of eigenvalues

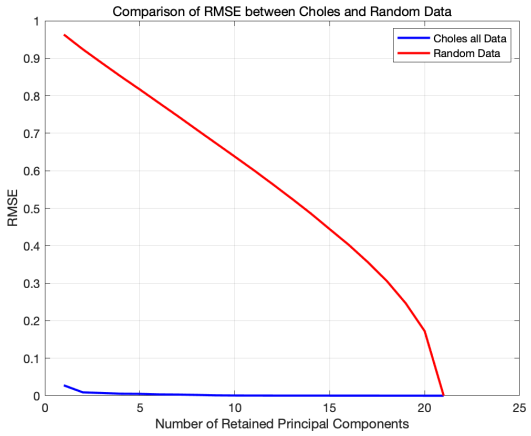

Figure 24 random data V.S. Choles data on RMSEs over the increasing numbers of eigenvalues

Upon employing PCA on correlated data, exemplified by the choles dataset, intriguing observations arise. Figure 24 demonstrates a rapid decline in the reconstruction error, accompanied by a distinct prominence of initial eigenvalues that significantly surpass the subsequent ones. This phenomenon suggests that exploiting these prominent eigenvalues enables a substantial reduction in dimensionality, surpassing what is achievable when dealing with uncorrelated data.

3.1.1 PCA on Handwritten Digits

PCA is utilized for the reconstruction of the digit 3 dataset. Figure 25 visually depicts the impact of incorporating a greater number of eigenvalues on the clarity of the reconstructed digit "3". By including additional eigenvalues during the reconstruction process, PCA enables a more comprehensive consideration of variations and intricate details within the dataset. These supplementary eigenvalues play a pivotal role in capturing finer patterns, edges, and other distinctive features that may have been overlooked when employing a limited number of eigenvalues.

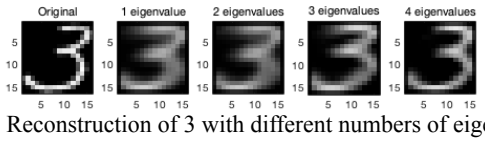

Figure 25 Reconstruction of 3 with different numbers of eigenvalues

Comparing the first 50 elements of the vector to the previously calculated vector of reconstruction errors reveals an intriguing and significant finding. It indicates that when the eigenvalues exhibit rapid decline, projecting the data onto the initial principal components yields smaller reconstruction errors. This phenomenon arises due to the relatively minor cumulative sum of the unused eigenvalues in the reconstruction process. Consequently, the principal components carrying the most informative content capture a substantial portion of the data variability, while excluding the remaining components incurs minimal information loss. As a result, smaller reconstruction errors are achieved.

3.2 Stacked Autoencoders

The provided autoencoder, with default parameters, comprises two layers, where the first layer consists of 100 hidden units trained for 400 epochs, and the second layer has 50 hidden units trained for 100 epochs. After three runs of pre-training and fine-tuning, the autoencoder achieves an impressive average accuracy of 99.73%. Consequently, this serves as a benchmark for evaluating the effects of parameter modifications in the autoencoder, specifically focusing on the number of epochs, hidden units, and layers.

Initially, the investigation focuses on a two-layer autoencoder, varying the number of epochs and hidden units in the first layer. With the number of epochs set to 50 and 1000, the accuracies before fine-tuning are 88.5% and 83.7%, respectively. After fine-tuning, these values improve to 99.2% and 99.8%. Upon restoring the number of epochs to 400 in the first layer and modifying the number of hidden units to 50 and 200, the accuracies before fine-tuning are 57.5% and 89.2%, respectively. Post fine-tuning, these values increase to 99.5% and 99.7%. In the second layer, when epochs are set to 50 and 200, the accuracies before fine-tuning are 52.4% and 94.3%, respectively, which improve to 99.7% and 99.0% after fine-tuning. Similarly, setting the number of neurons to 10 and 100 yields accuracies before fine-tuning of 36.9% and 92.1%, respectively, which significantly improve to 99.9% and 99.8% after fine-tuning. Additionally, the investigation includes the tuning of the number of layers in the autoencoder. Setting the number of layers to 1 and 3 results in accuracies before fine-tuning of 98.2% and 17.9%, respectively, and accuracies after fine-tuning of 98.5% and 99.6%. The analysis of the three parameters reveals the following findings: (1) Increasing the number of epochs and neurons during training enhances the autoencoder's learning capability, leading to improved accuracies. (2) The addition of layers in the autoencoder negatively impacts pre-training performance, while reducing the number of layers yields better pre-training results. (3) Fine-tuning plays a crucial role in significantly enhancing the autoencoder's performance, even when the accuracy during pre-training is initially low.

In comparison, multilayer perceptrons (MLPs) demonstrate inferior performance. MLPs with varying numbers of neurons and layers, trained for 1000 epochs, are generated for comparison. One-layer MLPs with 200, 100, and 50 neurons achieve accuracies of 97.2%, 97.4%, and 95.5%, respectively. For two-layer MLPs, accuracies of 96.5%, 96.6%, 94.4%, and 96.8% are attained when the configurations are [100 100], [50 50], [200 200], and [100 50], respectively.

Comparing these concepts to the earlier argument on training Multilayer Perceptrons (MLPs), it is evident that pre-training autoencoders offer significant advantages over random initialization of MLPs. Firstly, pre-training serves as a beneficial starting point in the vast hypothesis space by reducing input dimensions and mitigating the vanishing gradient problem. Secondly, pre-training employs unsupervised learning, which has a substantial impact on generalization. Despite the seemingly high number of parameters, a significant portion of training occurs in an unsupervised manner. As a result, the hypothesis space is constrained to a smaller region that can be effectively explored through supervised learning. The effect of fine-tuning is crucial as it enables the model to adapt the learned representations from pre-training to the specific task. During pre-training, the model learns general features or patterns from unlabeled data, which may not be directly optimized for the target task. By fine-tuning the model, the learned features can be refined to be more relevant and suitable for the specific

classification or regression task. Fine-tuning allows the model to further optimize its representations, making them more discriminative and aligned with the target labels.
The impact of fine-tuning is often significant, as it bridges the gap between unsupervised learning and supervised learning. It helps the model leverage the pre-trained knowledge while adapting it to the labeled data, resulting in improved performance and generalization.

3.3 Convolutional Neural Networks

3.3.1 Dimension Calculation

The first convolutional layer in the network has weight dimensions of 96 filters, each with a size of 11x11x3. These filters are responsible for detecting specific patterns or features in the input data. The dimensions 11x11x3 indicate that the filters have a width and height of 11 pixels, and the "3" represents the number of input channels, typically corresponding to the RGB color channels in an image.

To determine the dimensions of the input at the beginning of the second convolutional layer (layer 6), two formulas can be used: (1) output_width = (W-F+2P)/S+1 and (2) output_height = (H-F+2P)/S+1. Here, W and H represent the width and height of the input, F denotes the size of the filters in terms of width and height, P represents the padding, and S is the stride size. In this case, the input data has dimensions of 227x227x3, and the first convolutional layer has 96 filters with sizes of 11x11, a stride of 4, and no padding. Applying the formulas, the output width and height of the first convolutional layer can be calculated as (227-11+2*0)/4+1 = 55. Therefore, the output of the first convolutional layer is 55x55x96. After applying the ReLU activation and Cross Channel Normalization layers, and performing pooling operations with 3x3 patches and a stride of 2, the input dimension for the second convolutional layer can be calculated as (55-3+2*0)/2+1 = 27. Thus, the input for layer 6 or the second convolutional layer is 27x27x96.

Prior to the Softmax layer, there are two fully connected layers and one output layer with 4096, 4096, and 1000 neurons, respectively. In comparison to the initial input data with dimensions of 227x227x3, these layers significantly reduce the output size.

Convolutional Neural Networks (CNNs) offer several advantages over fully connected layers for image classification. (1) CNNs are more parameter efficient since they share weights across space. Unlike fully connected networks, where each neuron in a layer is connected to all neurons in the previous layer, CNNs only connect each neuron to a small patch of the input volume. This local connectivity and weight sharing reduce the number of parameters, making the network easier to train. (2) CNNs exploit spatial correlation by preserving the spatial relationship between pixels, which is crucial for handling image data. Fully connected networks do not consider this property. (3) CNNs exhibit translation invariance, meaning once a pattern is learned in one part of the image, it can be recognized in any other part. In contrast, a fully connected network would need to learn the pattern anew if it appears in a different location. (4) Deep CNNs can learn hierarchical levels of representation, enabling the detection of complex patterns. Lower layers may learn simple features like edges, middle layers recognize combinations of edges forming shapes, and higher layers identify collections of shapes constituting objects.

3.3.2 CNNs on Handwritten Digits

The default benchmark CNN model, consisting of 2 convolutional layers and one fully connected layer, achieves an accuracy of 82.36%. To assess the model's performance, nine parameters are tuned individually: mini-batch size, the number of convolutional layers, the number of fully connected layers, initial learning rates, learning algorithms, the number of training epochs, convolutional layer sizes, the number of convolutional filters, and strides. (1) Tuning the mini-batch size (default value: 128) to 4, 8, 16, 32, and 64, resulted in accuracies of 97.04%, 98.88%, 98.94%, 97.76%, and 94.76%, respectively. The best performance was achieved with a batch size of 8. Smaller batch sizes introduce noise in the gradient estimates, acting as a form of regularization, preventing overfitting, and improving generalization. (2) Varying the number of convolutional layers (1, 3, 4, and 5) yielded accuracies of 88.88%, 91.08%, 98.56%, and 98.84%. The results indicate that

increasing the number of convolutional layers improves accuracy, but there is a saturation point where further increasing the layers does not necessarily yield significant improvements. (3) Increasing the number of fully connected layers (2, 3, 4, and 5) led to corresponding accuracies of 88.08%, 93.32%, 94.16%, and 95.00%, following a similar trend as the convolutional layers. (4) Adjusting the initial learning rate from 0.0001 to 0.0003, 0.0005, 0.0008, and 0.0010 resulted in accuracies of 92.48%, 87.24%, 93.32%, and 82.84%. (5) Exploring different learning algorithms, using Adam achieved an accuracy of 86.16%, while RMSprop reached 94.24%. (6) Increasing the number of training epochs showed improved accuracy. With 50 and 100 epochs, the CNN model achieved 97.00% and 97.88% accuracy, respectively. However, increasing the epochs may lead to overfitting. (7) Investigating different convolutional filter sizes, a 3x3 filter size produced the highest accuracy of 91.80%. Larger filter sizes, such as 4x4 and 5x5, resulted in accuracies of 84.56% and 78.72%, respectively. (8) Tuning the number of convolutional filters for the two layers, using [12,12], [24,24], [24,36], [36,48], and [48,60] configurations achieved accuracies of 15.12%, 89.52%, 94.32%, 97.08%, and 97.88%. (9) Adjusting the stride to 1, 3, and 4 resulted in accuracies of 96.84%, 25.20%, and 22.32%, respectively.

Based on the parameter tuning results, the best performing configurations were identified as follows: (1) 5 convolutional layers; (2) 5 fully connected layers; (3) mini-batch size of 8; (4) initial learning rate of 0.0008; (5) 100 epochs; (6) RMSprop as the learning algorithm; (7) convolutional filter sizes of [12,24,36,48,60]; (8) fully connected layer neuron configuration of [512,256,128,128,10]; (9) convolutional layer sizes set to 5; (10) stride set to 2. Using these optimized parameters, the final accuracy of the CNN model reached 99.72%.

Exercise Session 4: Supervised Learning and Generalization

4.1 Restricted Boltzmann Machines

To evaluate the performance of Restricted Boltzmann Machines (RBMs), several training parameters are tuned, including the number of neurons, iterations, learning rate, and Gibbs sampling steps.

The pseudo-likelihood method is employed as an approximation to the log-likelihood of the RBM model. This method estimates the probability of a training sample by considering it as a missing data point and conditioning on the remaining observed variables. Maximizing the pseudo-likelihood improves the model's ability to reconstruct the training data, although it does not guarantee better generalization or performance on downstream tasks.

Increasing the number of neurons and iterations generally leads to higher pseudo-likelihood values and more accurate generation of digits. Having more neurons allows the RBM to capture complex representations and details in the data. However, an excessively large number of hidden units can result in overfitting and increased computational requirements. In this study, the maximum number of neurons (1000) when Gibbs sampling fixed at 10 achieved the best results, see Figure 26. Similarly, increasing the number of iterations helps the RBM refine its learned representations and enhance performance. However, there is a point at which the pseudo-likelihood value plateaus, indicating diminishing returns in terms of performance improvement.

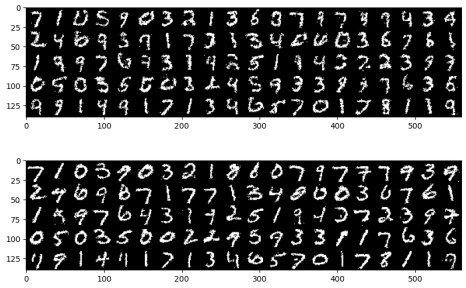

Figure 26 top: 10 neurons; bottom: 1000 neurons

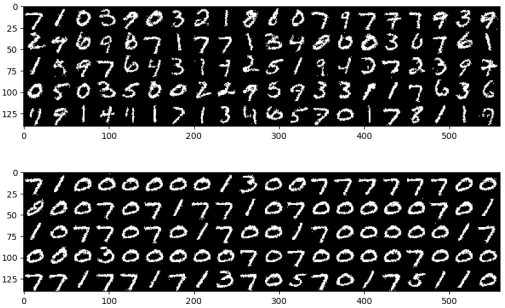

Figure 27 top: 100 Gibbs steps; bottom:1000 Gibbs

The choice of learning rate influences the convergence speed of the RBM. Higher learning rates facilitate faster convergence but can introduce instability and hinder finding a good solution. In this study, a lower learning rate (0.01) yielded higher pseudo-likelihood values for digit reconstruction compared to a higher learning rate (0.1).

Gibbs sampling, employed in RBM training, approximates the distribution of hidden units given the visible units. It updates the hidden and visible units in a Markov Chain Monte Carlo (MCMC) manner, enabling the model to explore the state space and converge to a stationary distribution. Increasing the number of Gibbs sampling steps does not necessarily result in improved performance and can even lead to worse results, see Figure 27. An ideal range for Gibbs steps is found to be 5-10.

Furthermore, when removing different locations in digit images, accurate digit reconstruction is achieved when the removal occurs within certain limits at the top and bottom parts of the image. Beyond a certain threshold, the quality of reconstruction deteriorates. In this study, accurate digit reconstruction was obtained for removal within the range of 0-15 in the top region. Beyond 15, the reconstructions became blurred and difficult to recognize (e.g., incorrectly generated digits such as "4", "9", and “7”). In Figure 28 and 29, three different numbers of neurons 10, 100, 1000 in the network have been tried, and the network with more neurons show better digit reconstruction.

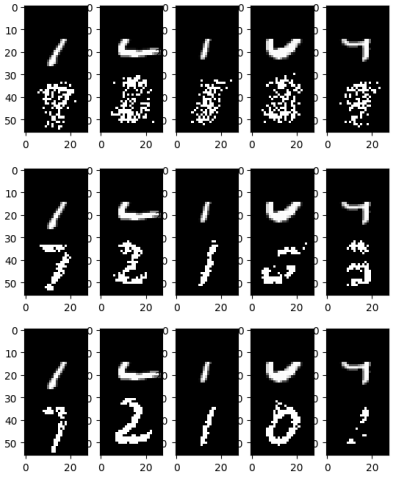

Figure 28 Different numbers of neurons applied to 0-15 masked digit reconstruction

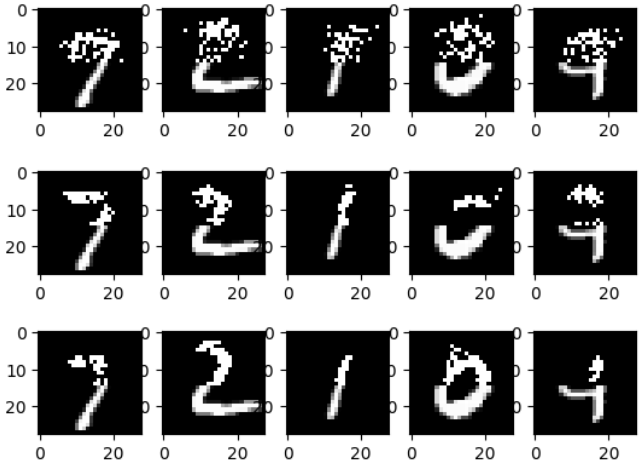

Figure 29 replace the reconstructed part with the masked location

4.2 Deep Boltzmann Machines

In the initial layer of a Deep Boltzmann Machine (DBM), the filters primarily emphasize loops within edges. In contrast, the first layer of a Restricted Boltzmann Machine (RBM) captures a broader range of information, including lines, edges, and loops with varying brightness, as observed in Figure 30. This discrepancy arises from the fact that RBM has a single layer dedicated to capturing as many features as possible. In contrast, DBM consists of multiple layers, with each layer specializing in specific features while allowing subsequent layers to extract other features, as depicted in Figure 31. Additionally, RBM's filters exhibit more noise compared to DBM. This is a consequence of RBM's inability to capture higher-level features and its susceptibility to overfitting, which manifests as increased noise. Figure 32 illustrates the comparison between the second layer's filters in DBM and its first layer's filters. The black circles indicate the areas where digits are recognized. Furthermore, the second layer extracts additional intricate textures and curves, demonstrating its capability to capture more complex patterns and features.

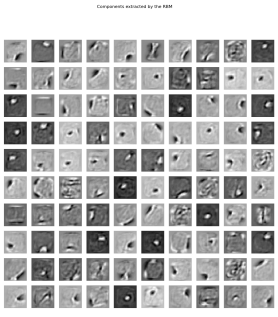

Figure 30 RBM hidden layer

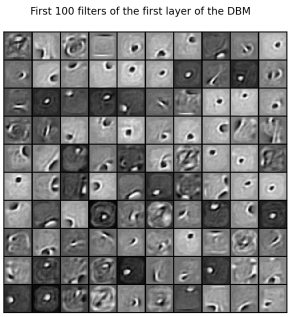

Figure 31DBM layer one

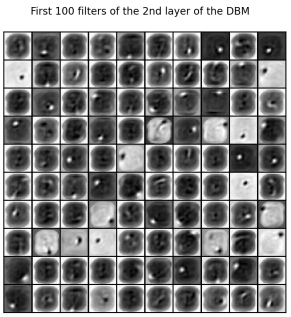

Figure 32 DBM layer two

Figure 33 Samples generated by DBM with 1000 Gibbs steps

The images generated by Deep Boltzmann Machines (DBMs) exhibit a remarkable resemblance to real digits, displaying minimal noise and fewer errors, as illustrated in Figure 33. This indicates that DBMs achieve a superior fit to the training distribution. Moreover, increasing the number of Gibbs sampling steps does not lead to a decline in image quality. This can be attributed to the hierarchical nature of DBMs and the presence of multiple hidden layers. Each hidden layer in a DBM captures increasingly abstract and higher-level features of the data. While the initial layers learn low-level representations, subsequent layers build upon these representations to acquire more intricate and abstract features. This hierarchical structure empowers DBMs to effectively model the data distribution, reducing the reliance on Gibbs sampling steps.

4.3 Generative Adversarial Networks

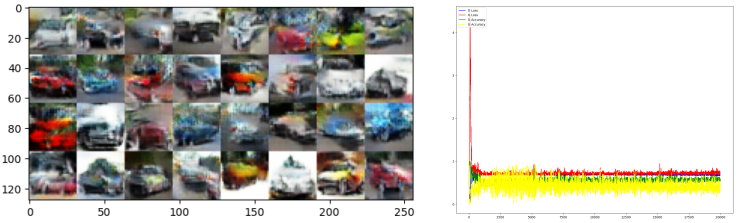

Figure 35 GAN training loss and accuracy

Figure 36 color transfer

During GANs training, Class 1 (car) is chosen as the target class and the training process spans 20000 iterations, as depicted in Figure 34. Throughout the training, oscillations are observed, which stem from a zero-sum competition between the generator and discriminator in forming a minimax structure. This competition creates a trade-off scenario where enhancing one aspect comes at the expense of the other. As a consequence, the network never converges, leading to a phenomenon known as "mode collapse." Analyzing the loss and accuracy metrics, the discriminator's loss fluctuates between 0.5 and 0.8, with an accuracy of approximately 0.6. Conversely, the generator's loss ranges from 0.8 to values exceeding 1, accompanied by an accuracy spanning from 0.1 to 0.5, as illustrated in Figure 35.

4.4 Optimal Transport

4.4.1 OT

To achieve color transfer, a commonly employed technique involves histogram matching between the source and target images. The color histogram represents the distribution of colors in an image, and by aligning these histograms, we can ensure that the target image adopts a similar color distribution as the source image.

In this exercise, optimal transport methods, namely Earth Mover's Distance (EMD) and Sinkhorn, are utilized to compute transport plans between the source and target images. The EMD algorithm determines the optimal transport plan by considering the distances between individual elements of the two distributions, minimizing the overall transportation cost. It ensures that the color transfer respects the spatial relationships of colors in the images. EMD takes into account both color values and their spatial locations when matching histograms. On the other hand, Sinkhorn incorporates an entropic regularization term to enhance the tractability of the optimization. The regularization term promotes a more uniform distribution in the transport plan, avoiding extreme or sparse assignments. This stabilization aids in the optimization process and facilitates convergence. By adjusting the regularization parameter, Sinkhorn Transport allows control over the level of regularization and, consequently, the smoothness of the color transfer.

In comparison, non-optimal color swapping involves a simple exchange of pixel values between the source and target images. This method disregards the overall color distribution and spatial relationships between colors. As a result, non-optimal color swapping can lead to abrupt and unnatural color changes, undermining visual coherence and the structural integrity of the images, as illustrated in Figure 36.

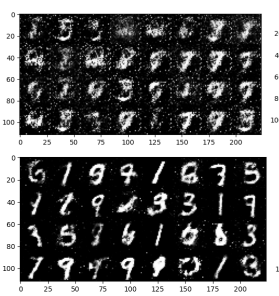

Figure 37 GAN, top: 5k iterations; bottom: 20k iterations

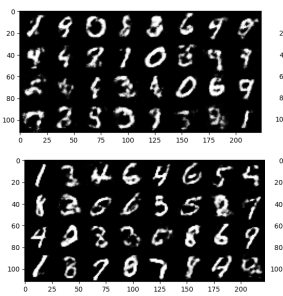

Figure 38 WGAN with GP, top: 5k iterations; bottom: 20k iterations

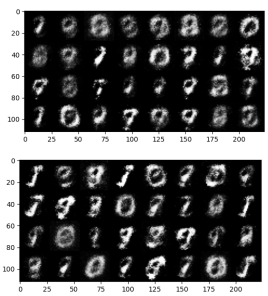

Figure 39 WGAN with WC, top: 5k iterations; bottom:

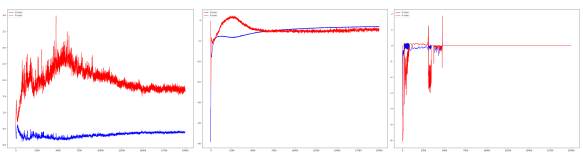

Figure 40 discriminator and generator losses in GAN, WGAN with GP and WGAN with WC

4.4.2 Wasserstein GANs

Training WGANs requires more time compared to GANs due to various factors. These factors include the complexity of the training process, the need to ensure stability, the incorporation of gradient penalty, and the computational overhead associated with evaluating the Wasserstein distance.

In GANs, the discriminator has an advantage over the generator, making it difficult for the generator to improve its performance. Gradient updates based on the reward function in GANs can result in issues such as exploding or vanishing gradients, negatively affecting the generator's training. On the other hand, WGANs exhibit linear gradient behavior, leading to more stable training and cleaner results. WGANs utilize the Wasserstein distance, derived from Optimal Transport, as a metric to compare the distributions of training data and generated samples. The objective of WGANs is to minimize this distance, which provides better gradients compared to GANs. The Wasserstein distance and Optimal Transport offer a more meaningful measure of dissimilarity between the distributions.

Empirical evidence suggests that WGANs tend to be more stable than GANs. WGANs generate images with fewer random white pixels, resulting in well-defined shapes even with a low number of training epochs. However, these shapes may not resemble digits or may exhibit bias towards specific numbers. On the other hand, GANs show gradual improvements over time by reducing noise and generating more realistic images, despite potentially starting with lower quality initially, see Figure 37. The main distinction between WGAN with weight clipping and WGAN with gradient penalty lies in the mechanism used to enforce the Lipschitz constraint. Weight clipping limits the magnitude of the discriminator weights, restricting the Lipschitz constant and ensuring stable gradients. However, weight clipping can lead to optimization problems, such as vanishing or exploding gradients, and may adversely affect the quality of learned representations. On the other hand, gradient penalty introduces a term in the loss function that encourages gradients of the discriminator to have a norm close to 1, effectively enforcing the Lipschitz constraint. This penalty term is computed by evaluating the norm of the discriminator's gradients with respect to interpolated samples along the real and generated data distribution. By incorporating the gradient penalty, WGAN with gradient penalty maintains a more stable and consistent training process without the issues associated with weight clipping, see Figure 38 and 39 and 40.

Use of ChatGPT (or any other AI Writing Assistance) – Form to be completed

Student name: **Fan Zhou**

Student number: **r0822921**

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: [H02C4a]

O This form is related to **my Master thesis**.

Title Master thesis: .....

Promotor: .....

Please indicate with "X":

O **I did not use** ChatGPT or any other AI Writing Assistance.

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

ChatGPT

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

X Assistance purely with the language of the paper

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

X As a search engine to learn on a particular topic

- *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.

O **For literature search**

➢ *Code of conduct*: This use is comparable to e.g. a google scholar search. However, be aware that ChatGPT may output no or wrong references. As a student you are responsible for further checking and verifying the absence or correctness of references.

O **For short-form input assistance**

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

O **To let generate programming code**

➢ *Code of conduct*: Correctly mention the use of ChatGPT and cite it. You can also ask ChatGPT how to cite it.

O **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.

O **To let generate blocks of text**

➢ *Code of conduct*: Inserting blocks of text without quotes from ChatGPT 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, mention it as a citation by using quotes. But this should be kept to an absolute minimum.

O **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 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 template samen met de code of conduct zal in de toekomst nog verdere aanpassingen behoeven. Het schept alvast een kader voor de 2de en de 3de examenperiode van 2022-2023.