KU LEUVEN

Artificial Neural Network and Deep Learning

[H00G8a]

Advanced Master in Artificial Intelligence

Option Engineering and Computer Science

Supervisor: Prof. Johan Suykens

Student name: Zhiqi Wang

Student ID: r0822618

August, 2023

Session 1 - Supervised learning and generalization

1. Compare different algorithms

The objective of this exercise is to train multiple neural networks using diverse learning algorithms for a simple function approximation task and subsequently compare their performance while controlling relevant parameters. The designated hyperparameters include a dense hidden layer comprising 30 neurons, initialized weights, and a specified transfer function. The target function for approximation is $y = \sin(x^2)$, where x ranges from 0 to 3π, and data points are sampled at intervals of 0.05. The dataset, containing 189 samples, is partitioned randomly into training (70%), validation (15%), and testing (15%) subsets. To curb overfitting, an early stopping mechanism is implemented, whereby training halts if the Mean Squared Error (MSE) on the validation set rises continuously for 6 epochs.

This exercise tries to perform a multifaceted evaluation of algorithmic performance. Firstly, it scrutinizes the convergence speed of the algorithms, draws fitted curve and assesses training time across each algorithm. It also examines the regression efficacy of each algorithm at different epochs (3, 30, 100). Additionally, the study extends its investigation to assess algorithmic robustness in the presence of noise. This is achieved by introducing additive white Gaussian noise to the original function, with a signal-to-noise ratio of 10. The neural networks are trained under identical conditions, enabling the recording of average MSE, training time, and regression performance. A comparative analysis is then conducted between algorithm performance on the unaltered data and data affected by noise, to ascertain the algorithms' relative robustness.

The six distinct training algorithms applied in this exercise are: gradient descent (GD), gradient descent with adaptive learning rate (GDA), gradient descent with momentum and adaptive learning rate (GDX), Conjugate gradient backpropagation with Fletcher-Reeves updates (CGF), BFGS quasi Newton algorithm (BFG), Levenberg-Marquardt algorithm (LM). Through this comprehensive testing, the study aims to provide clear insights into algorithmic efficacy and resilience, thereby contributing to informed decision-making in selecting appropriate algorithms for function approximation tasks.

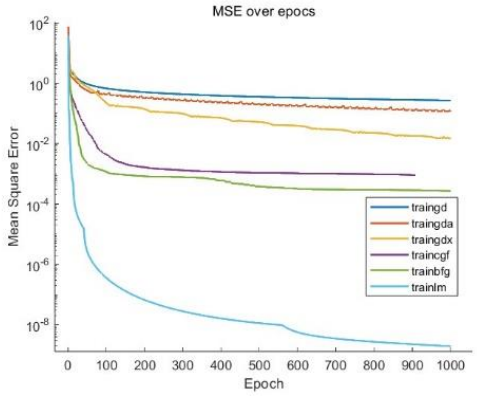

Figure 1 - MSE over epochs with original data

The convergence speed becomes evident through the MSE plot, where training extends to 1000 epochs. Algorithms showcasing sharper initial declines imply swifter convergence towards lower error levels. In Figure 1, it is clear that across all tested training algorithms, the MSE systematically diminishes over training epochs for the datasets, and no signs of overfitting manifest. Among these algorithms, CGF, BFG, and LM exhibit more pronounced declines on

the original data, indicating their superiority. Notably, LM demonstrates converging faster with lower MSE, standing out as the superior algorithms when original dataset. In addition, GDA's introduction leads to oscillations, effectively addressed by GDX, which shows improved performance compared to both GD and GDA.

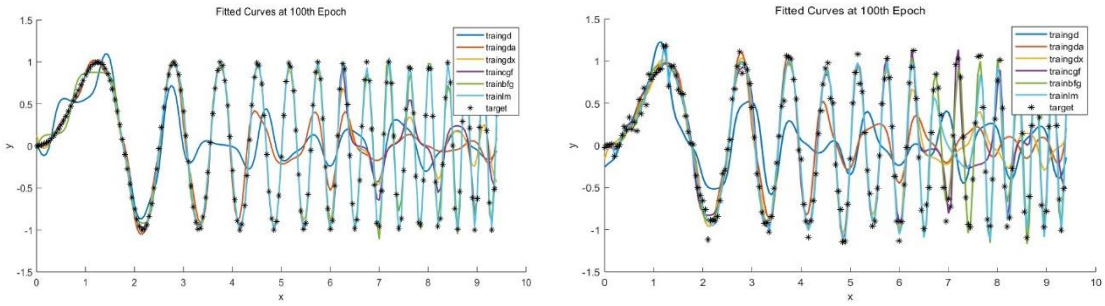

Figure 2 - The fitted curve at 1000 epochs without (left) and with noise (right)

Continuing, we proceed to evaluate different algorithms using the fitted curve plot, which facilitates a direct visual comparison between the algorithms' generated predictions and the actual target values. This comparative analysis reveals the accuracy of each algorithm by gauging how closely the fitted curves align with the target data. From the left plot in Figure 2, GDA displays the poorest alignment with the target curve, while the BFG and LM algorithms showcase relatively superior performance with a higher match stability, signifying higher accuracy. In contrast, the remaining algorithms exhibit suboptimal performance.

Furthermore, the comparison of fitted curves between the original and noisy data illuminates the algorithms' noise robustness. When comparing the left and right plots in Figure 2, the BFG and LM algorithms demonstrate consistent and stable alignment with the target curve, showcasing very good robustness to noise. In contrast, other algorithms exhibit diminished alignment with the curve when noise is introduced. This suggests a susceptibility to noise influence, particularly notable in GD's poor robustness performance.

Comparing the regression efficiency across various algorithms, performance metrics were evaluated for networks trained over 3, 30, and 100 epochs, yielding results displayed in Figure 3 and Figure 4.

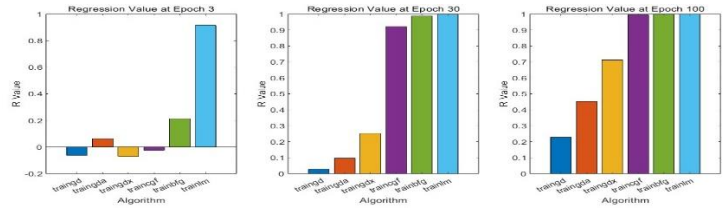

Figure 3 - Regression value for original data with 3 (left), 30 (middle), and 100 (right) epochs.

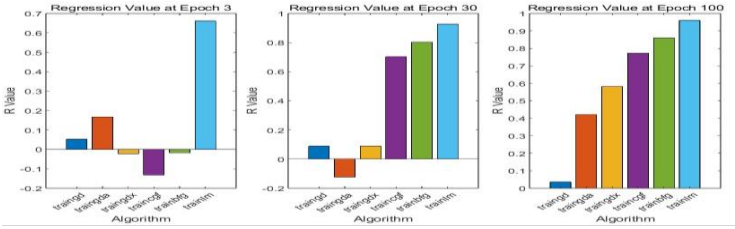

Figure 4 - Regression value for data with noise with 3 (left), 30 (middle), and 100 (right) epochs.

From Figure 3, it's evident that GD consistently exhibits inferior performance across different epochs. In contrast, LM consistently outshines other algorithms, even achieving an R value above 0.9 within just 3 epochs. GD, GDA, and GDX exhibit marginal improvements over each other, with relatively lower performance. Notably, CGF and BFG consistently outperform the former three, demonstrating substantial improvement as the number of epochs increases, R values surpass 0.9 at epoch=30 and draw closer to unity at epoch=100.

Upon introducing noise, observations from Figure 4 align with expectations, almost all algorithms experience a slightly worse performance in regression than with original data. However, LM stands out by exhibiting minimal deterioration in its regression performance. This robustness to noise highlights LM's adaptability.

In the final analysis, a comprehensive comparison involving regression, Mean Squared Error (MSE), and training time is presented in Figure 5, shedding light on algorithmic performance and noise resilience.

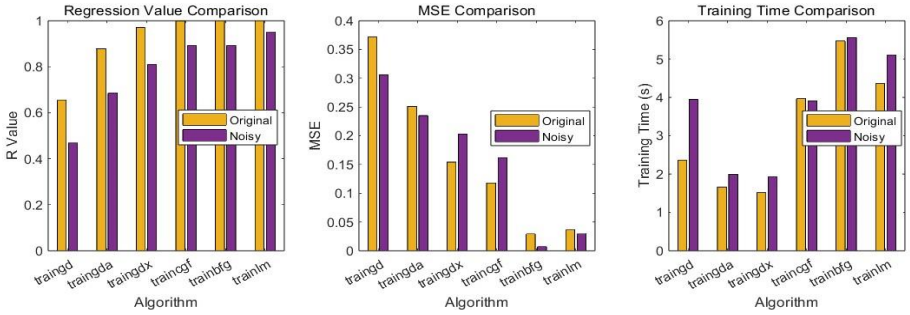

Figure 5 - Compare algorithms performance on original data and noisy data from three aspects: regression (left), MSE (middle) and training time (right) respectively.

Overall, the performance of GD, GDA, and GDX algorithms falls short, particularly evidenced by GD's R value below 0.7 and MSE exceeding 3.5, which surpass other algorithms by over 30%. However, they do boast relatively short training times. Importantly, their performance slightly diminishes when confronted with noise-introduced data, with GD being the most adversely affected, followed by GDA and GDX.

Instead, BFG and LM emerge as top performers, and demonstrate remarkable immunity to noise. Both algorithms achieve R values exceeding 0.9 for both original and noisy data. Moreover, they maintain impressively low MSE values below 0.05. While their execution times are extended, the trade-off for enhanced accuracy and noise resilience is apparent.

This performance discrepancy can be attributed to algorithmic design and principles. GD, GDA, and GDX might struggle due to limited adaptability and sensitivity to noise, affecting their ability to accurately approximate the target function. BFG and LM, on the other hand, are better equipped to capture underlying patterns, contributing to their superior regression and noise resistance.

2. Approximating a 2D Function using a Feed-forward Network

2.1 Hyperparameters selection

In this analysis, a multilayer feed-forward network is employed to approximate a two-dimensional function. The process of selecting network hyperparameters is examined. Data is randomly sampled and divided into training, validation, and testing sets. The 3D surface plot illustrates how target values (trainingY) change with varying input features (x, y), showcasing

regions of higher or lower surface values. The LM algorithm, a top performer from prior exercises, is selected for comparison with Bayesian regularization backpropagation (BR). The validation set determines hyperparameters, revealing insights into out-of-sample and generalization errors and assisting in network structure selection.

The chosen neural network employs 'tansig' activation for the hidden layer and 'purelin' for the output layer. 'tansig' introduces non-linearity for learning complex patterns, while 'purelin' suits regression tasks. Utilizing 1000 epochs in training capitalizes on effective in-sample error reduction with trainlm. Determining optimal hidden layers and neurons involves a grid search. Initially, layers are varied while maintaining 20 neurons to determine effective layer count. Subsequently, neurons are varied with a single hidden layer. Deeper networks showed minimal improvement, leading to a final choice of 1 hidden layer with 20 units using 'tansig'.

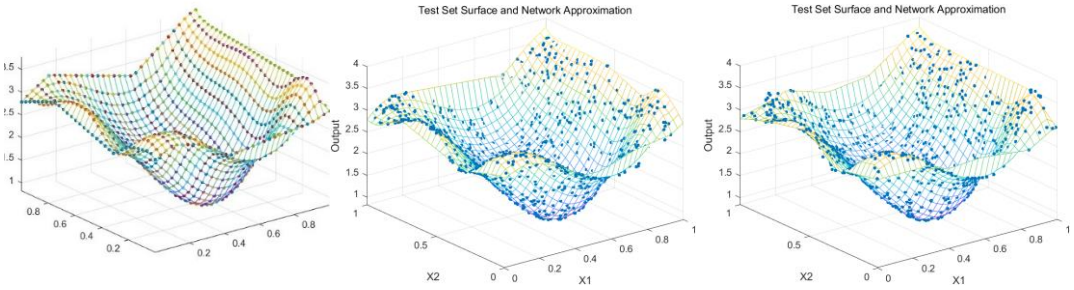

Figure 6 - Training set surface (left), test set surface for trainlm (middle) and for trainbr (right).

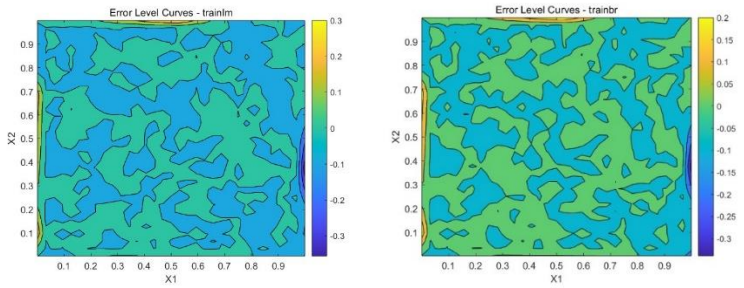

Figure 7 - Error level curve for trainlm (left) and for trainbr (right).

With the parameters chosen, a good level of generalization can be seen on the test set compared to the training set (Figure 6). Error peaks near the border are highlighted, which shows the error surface on the test set. We can observe the performance of the two algorithms more intuitively by drawing the error level curve. It can be seen from Figure 7 that both perform very well. Especially trainbr combines the best of the results with regularization, which helps to avoid overfitting, but its speed is relatively slow. In addition, on the boundary of the two the error is higher because the density of training points is lower there, and the result comes from the interpolation that forms the grid, not from the data set itself, so the error is often larger. If we could use more training points, especially near the boundaries, the performance of this network would improve significantly.

2.2 Levenberg-Marquardt (LM) VS. LM with Bayesian regularization

For the comparison of the performance of LM and BR in noisy and noise-free data, due to limited space, the main conclusion is that LM with Bayesian regularization (BR) performs better than LM on the test set. In particular, Bayesian regularized networks generalize better to noisy data and have better generalization capabilities.

Session 2 - Recurrent Neural Networks

1. Hopfield Network

1.1 Two-neuron and Three-neuron Hopfield Network

The Hopfield network, a type of recurrent artificial neural network, finds applications in tasks like pattern recognition, optimization, and associative memory. It operates by minimizing an energy function and consists of fully connected neurons that update synchronously, producing an output sequence from a single input. In our exercise, we focus on simple Hopfield networks with two and three neurons.

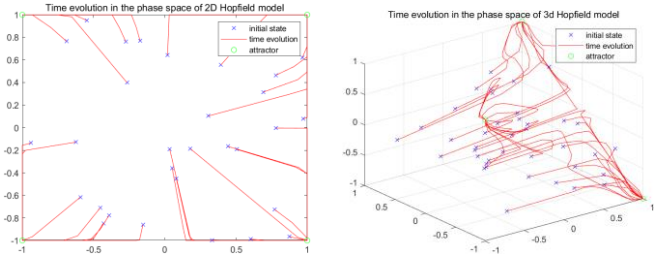

Figure 8 - 2D time evolution (left) and 3D time evolution (right)

To start, we construct a Hopfield network with two neurons and with the attractor matrix T = [1 1; -1 -1; 1 -1]. Using 30 initial points, we generate random vectors within the range [-1, 1] using the function -1 + 2 * rand(2, n). By simulating the network for 50 timesteps, we can visualize the results of the 2D Hopfield model in Figure 8. The green hollow points indicate attractors, all located at the graph's corners. In a Hopfield network, an attractor corresponds to a steady state to which the network converges iteratively. When all attractors fall on the corners, it's likely due to the experiment starting from random initial states. The network tends to converge to the nearest attractor in terms of Hamming distance.

Furthermore, this network usually reached a stable state after about 20 iterations. However, this number varies with distance: the greater the distance, the more iterations needed, but convergence is faster. Transitioning to 3D Hopfield networks, we notice similar behavior. After generating random points and allowing convergence, due to the higher dimensionality, around 55 iterations are required on average. The 3D network accurately stores its expected attractors without any extra ones. This is attributed to the increased network dimensionality, which enhances the network's ability to correctly store a certain number of attractors compared to a network with fewer neurons. In summary, this analysis sheds light on Hopfield network behavior, convergence tendencies, and their response to dimensionality changes.

1.2 Handwritten digits

The performance of the Hopfield network is explored within the context of noise introduction, shedding light on its capabilities when faced with corrupted data. Initially, the network's attractors are established using handwritten numbers 0 to 9. Subsequently, the network's ability to accurately recover these patterns is tested by subjecting it to noisy input.

During the reconstruction of corrupted digits with a noise level of 10 and 1000 iterations, the trends are evident in Figure 9. As noise escalates, the network's capacity to rectify distorted numbers rapidly diminishes, particularly when numbers share similar shapes (like 2 and 0). Thus, Hopfield networks don't consistently manage to reconstruct corrupted numbers. Overall, as noise increases, the required iterations for reconstruction rise due to noise pushing the input

away from attractors. While amplifying the iterations can enhance performance, even exceeding 1000 iterations doesn't guarantee successful reconstruction for heavily noisy cases.

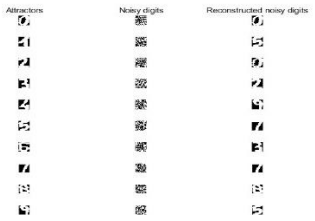

Figure 9 - Digits recognition after introduced noise

Fundamentally, the network relocates input towards the nearest attractor in a finite iteration count, resembling the behavior of nearest neighbor algorithms. However, sporadic spurious attractors may emerge where input hasn't reached, influencing network accuracy.

Overall, network performance remains commendable, as errors arise mainly when noise surpasses recognition thresholds for digits. The network might potentially fare better with higher-resolution images featuring more pixels. This effectively augments the "distance" between attractor states, possibly enhancing performance.

2. Long Short-Term Memory Networks

2.1 Neural network

To explore model performance across varying lags and neuron counts, with a focus on parameter tuning's impact on predictions, we conducted a single run on the provided chaotic laser data using Levenberg-Marquardt with Bayesian regularization. The training utilizes a multi-layer perceptron (MLP) framework, and the trainscg algorithm, adept at handling numerous parameters. The network undergoes 1000 training iterations with different lag and hidden layer neuron values, and the root mean square error (RMSE) is employed to identify the optimal parameter combination.

To begin, we set the lag value (P) at 10 and manipulate the neuron count across a range. In Figure 10, we compare target and predicted values for neuron counts of 5, 20, and 40. Notably, increased neuron count doesn't guarantee linear performance improvement, instead, optimal results surface around 20 neurons. This configuration exhibits superior target value capture and minimal RMSE (0.06). However, this network only predicts the initial peak series, failing to accurately forecast the laser's qualitative behavior post-decline. Oversized neuron counts can exacerbate this issue, yielding instability. Conversely, too few neurons hinder correct initial qualitative predictions. Shifting focus, we maintain 40 neurons and vary lag values (P=3, 10, 50), yielding analogous conclusions (Figure 11).

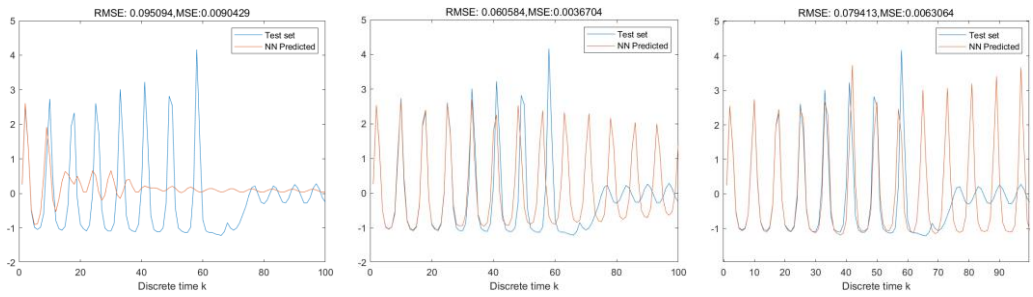

Figure 10 - Compare the RNN prediction with target data, lag value (P) =10, number of neurons (n)=5 (left), 20 (middle), 40 (right)

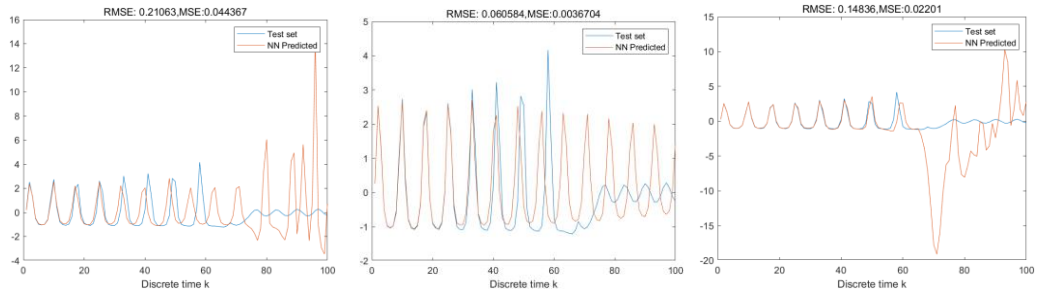

Figure 11 - Compare the RNN prediction with target data, number of neurons (n)=40, lag value (P) = 3 (left), 10 (middle), 50 (right)

Overall, the primary challenge across models is replicating the amplitude drop in the test set at discrete time k around 60. Even with parameter optimization, this struggle persists. This limitation can potentially be addressed by employing the LSTM (Long Short-Term Memory) method.

2.2 Long short-term memory network

An LSTM (Long Short-Term Memory) network stands as a specialized form of Recurrent Neural Network (RNN) capable of learning long-term dependencies. By processing previous inputs to understand its current state and forecast subsequent outcomes, it exhibits enhanced performance in capturing intricate temporal patterns. The LSTM architecture incorporates input gates for selecting relevant inputs to the memory cell, a forget gate for determining retained prior state information, and an output gate that regulates network output. These gates operate on the current input and previous output, thereby influencing state updates through activation functions and weight adjustments facilitated by iterative learning processes, error backpropagation, and gradient descent optimization.

This LSTM model encompasses a sequence input layer, a hidden layer with 50 LSTM units, a fully connected layer, and a regression layer. Leveraging the Adam optimizer for training across 1000 epochs with a learning rate adjustment after 150 epochs and a gradient threshold of 1, this network aims to predict the subsequent 100 time steps. As illustrated in Figure 11, predictions and errors are presented. Initially predicting without updating observations, the left-side forecast demonstrates inaccuracies in forecasting the qualitative laser behavior, similar to RNN results. The approach then evolves by updating the network state after each prediction, yielding accurate predictions for the declining laser signal and low RMSE values.

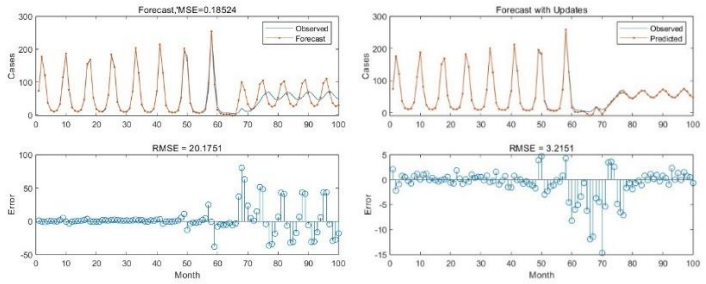

Figure 12 - Compare the LSTM prediction with target data for both the forecast (left) and updating the network with observed values, epochs= 1000, number of neurons =50

Comparing LSTM and RNN fitting outcomes, it becomes evident that LSTM excels in time series prediction, showcasing exceptional performance with 50 units. Moreover, LSTM's ability to accurately forecast with a lag value of 1 significantly reduces training time compared to RNN.

Session 3 - Deep Feature Learning

1. Principal Component Analysis (PCA)

1.1 Redundancy and Random Data

Principal Component Analysis (PCA) serves the purpose of dimensionality reduction by minimizing reconstruction errors. It achieves this by projecting high-dimensional data onto lower-dimensional subspaces while retaining the maximum variation present in the data. A practical strategy involves selecting the direction with the greatest variance in the input space and discarding the rest.

To comprehend PCA's characteristics, a comparison is made between the redundancy of random and highly correlated data. Initially, PCA is employed to reduce dimensionality, followed by attempts to reconstruct the original dataset, and the quality of reconstruction is gauged using the Root Mean Square Error (RMSE).

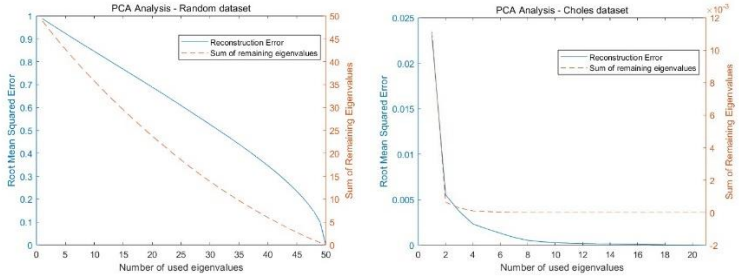

Figure 13 - View the reconstruction error and the sum of remaining eigenvalues of random dataset (left) and Choles dataset (right)

For randomly generated data, a matrix of Gaussian random numbers with dimensions 50x500 is generated. The provided Choles dataset was used as the highly correlated data. As depicted in Figure 13, the distribution of eigenvalues for randomly generated data follows a Gaussian distribution. This indicates that most of the dataset's variance is distributed across various dimensions. The blue solid line illustrates the RMSE for reconstruction, which decreases as the eigenvalues increase, eventually approaching 0 when eigenvalues reach around 50.

The orange dashed line represents unused eigenvalues for random values. These eigenvalues decrease nearly linearly as the number of used eigenvalues increases. This suggests that the input data are largely uncorrelated. In the case of the highly correlated Choles data, the reconstruction error decreases rapidly within the first two dimensions and gradually diminishes in subsequent dimensions, ultimately reaching 0 at dimension 20. This signifies that significant dimensionality reduction can be achieved by using the first two eigenvalues. Notably, PCA performs exceptionally well on highly correlated data compared to uncorrelated data.

It is clear that PCA's ability to reduce dimensionality and capture data variance is influenced by the degree of correlation within the dataset. It performs better on data with strong correlations, as evident from the rapid reduction of reconstruction error in the first few dimensions.

1.2 PCA on Hand-written Digits

We proceed by conducting Principal Component Analysis on handwritten images of the digit "3" from the US Postal Service database. Initially, calculating the average of this dataset doesn't reveal any obvious identifiable pattern. However, upon computing its covariance matrix, we delve deeper into its underlying structure. Figure 15 visually portrays the changes in eigenvalues, illustrating a rapid logarithmic increase beyond the 200th eigenvalue index.

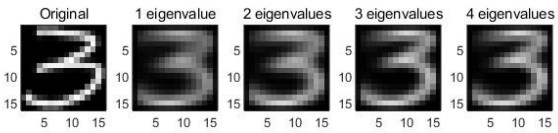

Figure 14 - The original image of "3" and the reconstructions image from one, two, three, and four dimensions

Subsequently, we compress the dataset by projecting it onto one, two, three, and four principal components, respectively, and then reconstruct images from these compressed representations. Figure 14 showcases the reconstructed and original images, where individual features become distinguishable. These reconstructed images convey enough information to form a blurry yet recognizable representation of the digit "3".

Moving forward, we examine the reconstruction error in relation to the cumulative sum of the remaining eigenvalues for this dataset. This inverse relationship is perceptible in Figure 15. Notably, as the number of principal components used for reconstruction increases, a substantial reduction in the reconstruction error is evident. While in theory, the error should approach 0 due to the image's 256 dimensions, practical limitations of numerical precision result in an error of approximately 5.5210e-16 after full reconstruction.

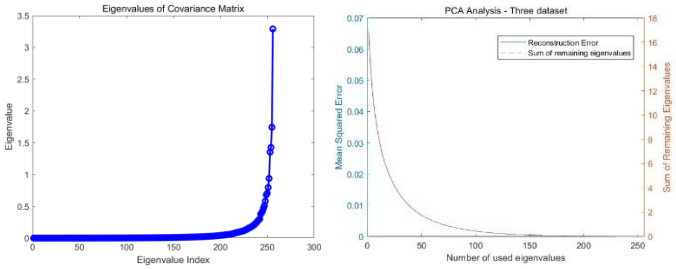

Figure 15 - The plot of eigenvalue of covariance matrix (left) and the PCA analysis of "three "dataset (right)

In general, for this dataset, it is reasonable to conclude that employing around 100 principal components is sufficient for meaningful analysis. In essence, Principal Component Analysis facilitates the identification of essential characteristics within datasets, enabling effective dimensionality reduction and feature extraction.

2. Stacked Autoencoders for Digit Recognition

Autoencoders represent another facet of unsupervised learning, functioning by compressing data using an encoder and then learning the process of data reconstruction through a decoder. Comprising multiple layers, they are constructed with an encoder and decoder stack, where the output of each layer feeds into subsequent layers. In this context, we contrast a stacked autoencoder with a conventional multilayer neural network. Our aim is to explore the impact of network parameters on the classification ability, particularly for recognizing digits subjected to random transformations.

The default stacked autoencoder configuration includes two autoencoders and a softmax layer. The initial autoencoder encompasses 100 hidden units, trained over 400 epochs. Figure 16 visualizes the associated feature weights after training the first autoencoder, depicting fundamental features like curved lines. A second autoencoder follows a similar procedure, with 50 hidden units, intended to learn a more compact input representation and reduce dataset dimensionality. Ultimately, the softmax layer is trained to classify and distinguish various digit classes.

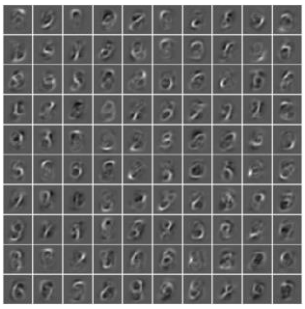

Figure 16 - The plot for features after first autoencoder layer

Upon replication, the default stacked autoencoder yielded an average classification accuracy of 95.7%. To enhance performance, we initiated the optimization of hyperparameters such as hidden layers, epochs, and hidden units. First, varying the number of layers (autoencoders) unveiled that a single hidden layer with 50 neurons reached a peak accuracy of approximately 92.9%. When using three hidden layers, performance dropped significantly compared to two layers. This might stem from overfitting due to added complexity or issues with initialization weights. Further examination of epoch variations revealed that the first autoencoder plateaus after 300 epochs. The second autoencoder demonstrated continuous improvement up to 100 epochs, prompting an increase to 200 epochs. This amalgamation led to an enhanced classification accuracy of 98.1%. Additionally, the influence of altering neuron count in each layer was explored. Interestingly, adding neurons seemed to have little effect, possibly attributed to the problem's moderate complexity. Reducing hidden units in the second autoencoder to 25 notably affected accuracy, dropping to 79%.

Ultimately, after fine-tuning, the network retained a similar structure to the initial setup with improved parameters. The first autoencoder contained 100 hidden units, trained for 300 epochs, while the second featured 50 hidden units, trained for 200 epochs. This refinement resulted in an enhanced classification accuracy of 98.1%. Pretrained autoencoders offer considerable advantages over random initialization, offering a more reasonable starting point and leveraging unsupervised learning for positive generalization effects.

3. Convolutional Neural Networks

Convolutional neural networks (CNNs) are powerful deep learning algorithms renowned for their exceptional performance in image classification. They excel by assigning weights and biases to distinct image features, enabling them to comprehend both spatial and temporal dependencies within images. In this exercise, we deployed a CNN as a class classifier to identify images from the Caltech 101 dataset. The network architecture follows that of AlexNet, comprising five convolutional layers, each succeeded by a ReLU layer and a max pooling layer.

To following formula used to calculate the size of the output:

$$\text{Output width} = \frac{W - F_w + 2P}{S_w} + 1$$

$$\text{Output height} = \frac{H - F_h + 2P}{S_h} + 1$$

The initial input image dimension in the first layer is 227x227x3. In the second layer, an 11x11x3 convolutional filter is applied with a stride of 4 in both directions, shifting each filter by 4 pixels before computing a new output. A visualization of the first convolutional layer's weights is depicted in Figure 17. This reveals that the network has learned 96 convolutional kernels, which serve to identify distinct features. Many of these kernels exhibit frequency and direction selectivity, detecting edges oriented in varying directions and frequencies. The application is both horizontal (55) and vertical, yielding a 55x55 per filter output, resulting in 55x55x96. A 3x3 max pooling layer with a stride of 2 reduces the output to 27x27x96.

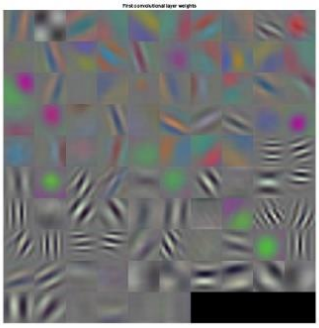

Figure 17 - The first convolutional layer weights of 96 features

By experimenting with different parameter combinations, we sought to enhance performance. Opting for two convolutional layers was prudent due to the dataset's manageable complexity, ensuring accuracy without overfitting. We adhered to the Adam optimization algorithm, along with selecting a connection layer. Varying the number of filters across layers, we found that too few led to substantial performance drops, while excessive filters yielded marginal improvements. Consequently, a 10x10 filter size was deemed optimal. Learning rate significantly influences convergence and computational time, here, a rate of 0.001 struck an effective balance. The default batch size of 128 was adjusted to 32 to achieve stable convergence.

CNNs derive three key advantages—local connections, parameter sharing, and translation invariance. Local connections exploit spatial correlations, reducing computational operations and enhancing efficiency. Parameter sharing implies that units in the same convolution kernel share parameters across layers, effectively decreasing the parameter count. Translation invariance ensures uniform outputs regardless of an object's position in the image, approximating translation invariance, an important property in image processing.

Session 4 - Generative Models

1. Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is a generative stochastic artificial neural network that learns a probability distribution over its input dataset. All input and hidden nodes are interconnected, facilitating the exchange of information among nodes to generate sufficient data for learning autonomously. In this section, we explore the impact of various training parameters on RBM performance when applied to the MNIST dataset, these parameters include the learning rate, number of components, Gibbs sampling step, and more.

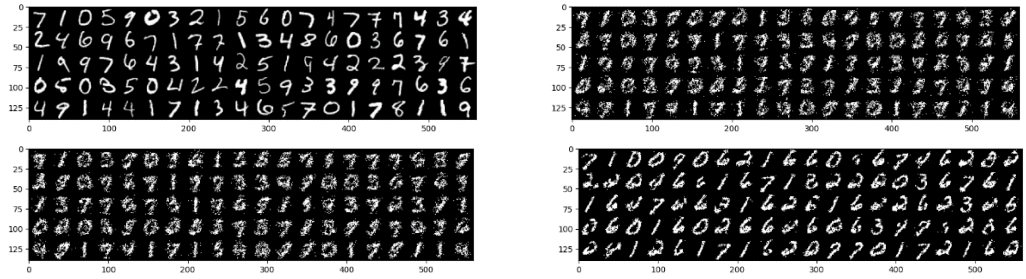

Figure 18 - Results for different Gibbs samples, upper left is the original test image, upper right is the reconstruction image with 10 component and 20 Gibbs, bottom left is with 100 component and 20 Gibbs, bottom right is with 10 component and 100 Gibbs

The term "number of components" pertains to the frequency of training iterations executed on the dataset. As illustrated in Figure 18, with a fixed Gibbs step of 20, varying the number of components yields subtle changes in image resolution, resulting in images that are somewhat more blurred than the original ones. While keeping the number of components fixed at 10 while increasing the Gibbs sampling steps (20 and 100), a noticeable enhancement in RBM's ability to replicate the original image can be observed.

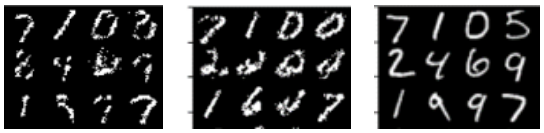

Figure 19 - Original image (right) with reconstruction image with Learning rate: 0.01 (left), 0.05 (middle)

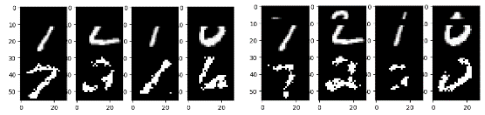

Figure 20 - Reconstruction image with Gibbs =100, remove row 0-12 (left), remove row 8-12 (right)

Another investigation involves altering the number and position of deleted rows, which impacts RBMs' capability to reconstruct the original image. Comparing Figure 20 with and without row removal for lines 0-7, it becomes apparent that reconstruction is improved. This could be attributed to the presence of defining features towards the bottom of the image rather than in the middle. Additionally, changing the learning rate influences reconstruction outcomes by affecting the network's adaptation level per iteration. The comparison in Figure 19, between learning rates of 0.01 and 0.05, reveals substantial enhancement in reconstruction with the latter, images generated with a learning rate of 0.05 are clearer. A lower learning rate slows fitting but

minimizes overshooting, while a higher rate accelerates learning but increases overshooting risk. In general, a slightly higher learning rate (e.g., 0.05) and a higher Gibbs step (e.g., 20) are favored for achieving satisfactory results with this dataset. Excessive row removal diminishes reconstruction ability, and specific digits, like 0, are more recognizable than others.

2. Deep Boltzmann Machines

A deep Boltzmann machine (DBM) can be conceptualized as a stack of restricted Boltzmann machines (RBMs), where each layer of hidden units is organized hierarchically. Connectivity exists between consecutive layers, but not within a layer or between non-adjacent layers. In this exercise, we assess the performance of DBMs and RBMs trained on the MNIST dataset, both utilizing a Gibbs step size of 100. The outcomes of the DBM are illustrated in Figure 21. A comparison with the RBM results from Figure 18 reveals that the DBM yields superior results, characterized by higher resolution and reduced noise. This improvement can be attributed to the DBM's utilization of inference as an approximation method, involving multiple passes through the hidden layers.

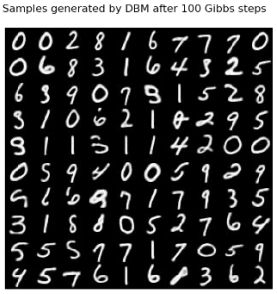

Figure 21 - Image generated by DBM

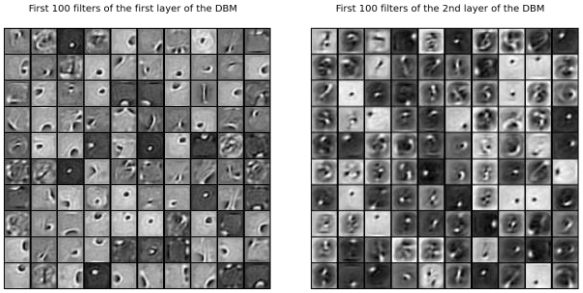

Figure 22 - The filter weights for layer 1 (left) and layer 2 (right)

The visualization of DBM's filter weights for each layer is showcased in Figure 22. Observations can be drawn from this representation. Firstly, the initial layer is responsible for detecting rudimentary features, such as circles, angles, and minor lines. Consequently, it possesses lower feature weights that allow it to identify distinct attributes in the early stages. The second layer demonstrates a more intricate structure, encompassing finer details and appearing slightly blurred in comparison to the first layer. Consequently, its filtering weights are higher. In summary, the training outcomes indicate that DBMs offer a better fit to the training distribution. This advantage arises from the fact that RBMs are limited in capacity as they consist of only one hidden layer and lack hidden units.

3. Generative Adversarial Networks

GANs are comprised of two neural networks, namely a generator and a discriminator, engaged in a zero-sum game where they strive to find equilibrium by enhancing their respective performances. The generator accepts random inputs and produces an image, subsequently fed into the discriminator alongside genuine ground truth data. The discriminator assesses real and fake images and furnishes probabilities, ranging between 0 and 1, denoting authenticity likelihood — 0 indicating a forged image and 1 implying a credible prediction. In this experiment, a deep convolutional generative adversarial network (DCGAN) is trained using the CIFAR dataset. This dataset comprises 10 classes, each with 6000 32 x 32 color images. For training, the "car" class is selected, and the focus is on discerning discriminative and generative loss.

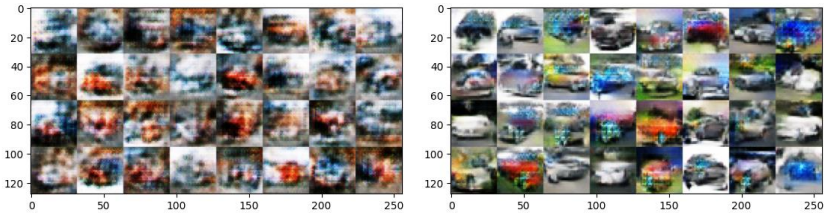

Figure 23 - Car images generated by GAN over 2000 batches (left) and 20000 batches (right)

Examining the images in Figure 23, it's evident that the GAN achieves realistic outputs following training on batches of 2000 and 20,000 images. At 2000 batches, the generated images are somewhat blurry, revealing car outlines and colors, albeit sometimes with irregular proportions. This initial fuzziness can stem from substantial disparities between the generator and the discriminator. However, with extended training, stability is attained, and the cars and their backgrounds, along with improved color fidelity, become discernible. The transition from blurry to detailed images highlights the refinement facilitated by prolonged training.

Differences in image quality arise due to variations between different generators and discriminators, contingent on the objective function's gradient. Generally, the generative loss tends to surpass the discriminative loss, indicating that the generator effectively confounds the discriminator. The temporal evolution of loss and accuracy isn't utterly consistent. The discriminator's loss hovers around 0.6, while the generator's loss rests around 0.9. This consistency signifies that the GAN is fairly steady in generating high-quality samples and has reached a good balance in terms of adversarial training dynamics.

4. Optimal Transport

The essence of the optimal transport problem lies in comparing two probability distributions to identify feasible strategies for matching their contents. However, while attempting to solve this problem, it's possible to encounter situations where the resulting mapping assigns strikingly distinct colors to pixels, leading to images with uneven and irregular color patterns. To address this issue, we introduce a regulalizer. This regularizer seeks to strike a balance between achieving an optimal mapping and maintaining more uniform color distributions. The regularization term essentially influences the smoothness of the color transfer in the image: higher weights of the regularization term result in smoother color transitions, albeit at the expense of the optimal mapping's precision.

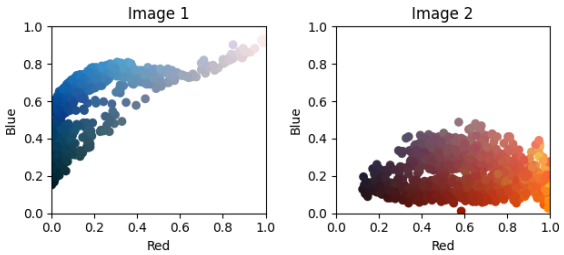

Figure 24 - Scatter plot for Blue VS Red (Adapt) and Sinkhorn

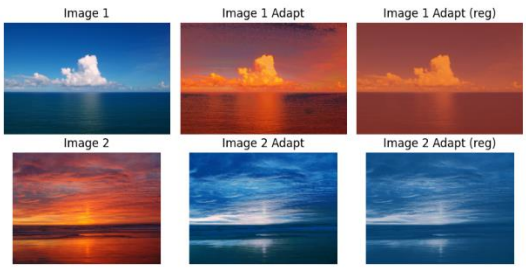

Figure 25 - Original Images, EMD Transport Images

In our experiment, we loaded two local images, "Ocean day" and "Ocean sunset," which display different color tendencies—more bluish and reddish, respectively. We apply two distinct methods of optimal color transfer between these images. The color distribution scatter plot for the images is presented in Figure 24. The initial step involves determining the distance between the two probability distributions, essentially indicating the volume of color changes required to

align the distributions. The second approach utilizes the Sinkhorn distance as the optimal distance metric. This distance metric is a modified version of the Wasserstein distance and boasts advantages in terms of its performance.

Figure 25 illustrates the outcomes of the optimal transportation problem using both the Wasserstein and Sinkhorn distances. Notably, the Sinkhorn distance emerges as the superior choice. This superiority stems from the fact that the Sinkhorn distance doesn't incorporate cost factors in the distribution alignment process. Consequently, this absence of cost factors results in a more even distribution of colors across the image. This indicates that the Sinkhorn distance method offers a more natural and visually pleasing color transfer, thus rendering it more effective for solving the optimal transport problem with color images.

4.4.1 Wasserstein GAN

Wasserstein GAN (Generative Adversarial Network) stands as an extension of the traditional GAN framework, designed to enhance training stability and yield a more informative loss function for assessing image quality. Unlike GANs that employ a discriminator to classify the authenticity of generated images, WGAN replaces this role with a "critic" responsible for scoring image realness. The critical innovation in WGAN is its focus on minimizing the distribution divergence between the training data and the generated samples, thereby mitigating issues like vanishing gradients and ensuring the model's suitability for more prolonged training sessions on robust networks.

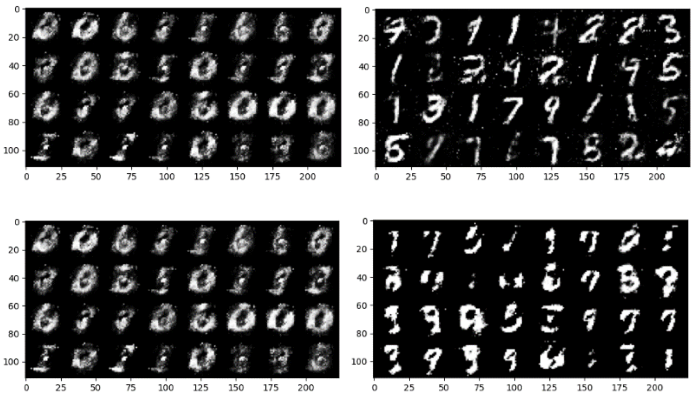

Figure 26 - Comparison of Image Reconstruction using GAN (upper) and WGAN (lower), first column is with 2000 batches, second column with 20000 batches

In a comparative analysis, both GAN and WGAN were trained on the MNIST dataset to gauge their stability and performance disparities. As depicted in Figure 26, the outcomes reveal distinct characteristics. GAN manages to produce satisfactory results after numerous iterations, albeit exhibiting less stability in terms of loss function and accuracy. On the other hand, WGAN demonstrates higher stability and generates images with finer resolutions. Notably, as training batch sizes increase from 2000 to 20000, both models yield clearer, less noisy images with discernible digits.

When weighing the overall picture, WGAN's image reconstruction and stabilization tend to outperform GAN, yielding cleaner outcomes with less noise. While GAN may require a longer training duration, its results tend to improve progressively over time. Moreover, GAN-generated images exhibit better recognition features, making the generated letters easier to distinguish. In general, GAN is a better option for the purposes of this exercise.

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

Student name: Zhiqi Wang

Student number: r0822618

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

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

Course name: Artificial Neural Network and Deep Learning

Course number: [H00G8a] .....

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/...):

.....

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

X **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.