Katholieke
Universiteit
Leuven

**Master of
Artificial Intelligence**

ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING

Course Report

Daichen Li (r0867950)

Academic year 2022-2023

Contents

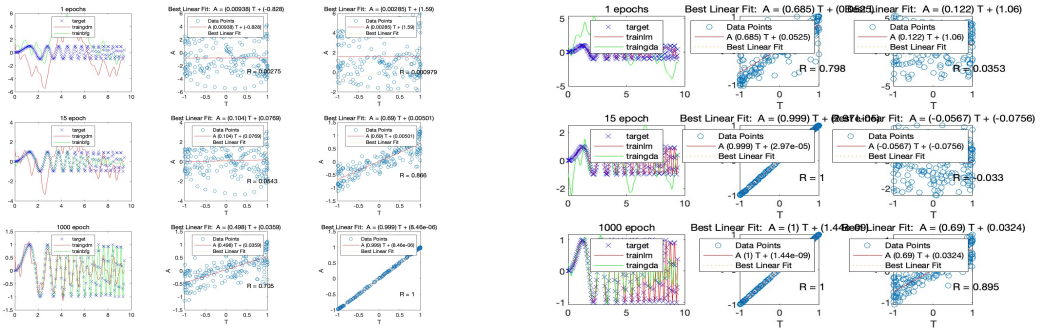
1 Supervised learning and generalization

1.1 Comparing between different training algorithms

The study compared the performance of two different neural network training algorithms with a hidden layer consisting of 50 neurons. Two training groups were established, each utilizing a distinct training algorithm. In the first group, the algorithms employed were 'traingdm' for two epochs, followed by one epoch of 'trainbfg'. The second group employed the 'trainglm' and 'trainda' algorithms.

在没有噪声的情况下观察和比较这些不同训练算法的结果

The aim of the study was to observe and compare the outcomes of these different training algorithms in the absence of noise. A comparison between the two algorithm groups was made, focusing on their respective performances. The results, as depicted in Figure 1, provided insights into the relative effectiveness of the algorithms and highlighted any variations in their performance characteristics.



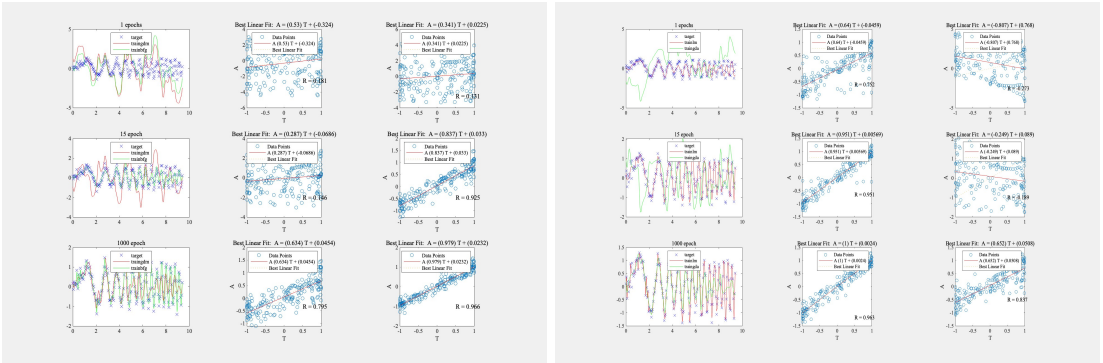
(a) traingdm and trainbfg (b) trainlm and traingda

Figure 1: Comparison Between Different Algorithms

观察和分析不同训练方法对神经网络性能的影响。

The objective is to observe and analyze the impact of different training approaches on the neural network's performance. By varying the number of training epochs for each training step, the study aims to examine the relationship between training duration and the improvement in training accuracy.

In this study, I used the sine function to generate examples and targets for training a neural network. The input data was created within the range of 0 to 3pi, with a step size of 0.05. To simulate real-world scenarios, I added Gaussian noise with a standard deviation of 0.2 to the output target signal. The original target signals were stored in the variable 'y', while the corresponding noisy targets were stored in the variable 'yn', as depicted in Figure 2.



(a) traingdm and trainbfg (b) trainlm and traingda

Figure 2: Comparison Between Different Algorithms (add noisy)

I conducted this experiment to examine how the presence of noise in the input data affects the performance of the neural network. By comparing the network's ability to learn and predict the original targets versus the noisy targets, I aimed to draw conclusions about the network's robustness to noise.

Through my analysis, I observed that the addition of Gaussian noise had a noticeable impact on the neural network's training and prediction accuracy. The presence of noise introduced uncertainty and made it more challenging for the network to accurately learn and generalize from the examples.

Despite the noise, the neural network still exhibited the capability to approximate the underlying sine function to a certain extent. However, the training and prediction errors were higher when compared to the scenario without added noise. This suggests that noise in the input data can adversely affect the network's performance and increase the difficulty of learning the underlying patterns.

1.2 Personal Regression

A new dataset was defined using the student number r0867950 to compute a weighted average of different datasets. The surface of the training set was plotted and is displayed in Figure 3.

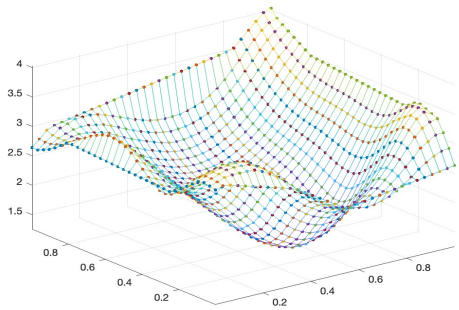


Figure 3: Training Set Surface

In this experiment, a feedforward neural network was trained using the Levenberg-Marquardt algorithm with different numbers of neurons in the hidden layer. Six neural networks were trained, each with a different number of neurons: 10, 15, 20, [5 5], [6 6], and [7 7 7]. The training process involved 1000 epochs using the training set. After each epoch, the mean squared error (MSE) was calculated for both the training and validation sets. The MSE values were used to evaluate the performance of each neural network and compare their results. The resulting MSE values were stored in the variables 'mseTraining' and 'mseValidation' for each neural network, as illustrated in Figure 4.

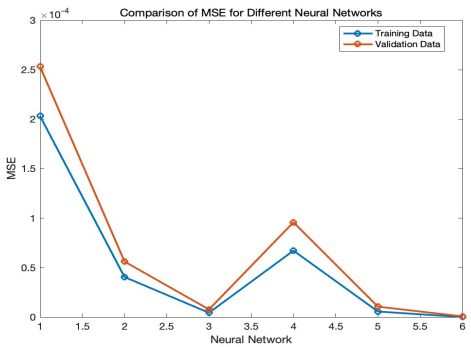


Figure 4: MSE Values for Each NN

By comparing the MSE values, the study aimed to analyze the performance of the neural networks with different numbers of neurons in the hidden layer. The MSE provides a measure of how well the network fits the

training data and generalizes to the validation data. Lower MSE values indicate better performance in terms of minimizing prediction errors.

The performance evaluation of each neural network was conducted using the training and validation sets. Mean squared error (MSE) was used as a loss function to calculate the average squared difference between the predicted value and the target value. During training, the neural network attempts to minimize the MSE by adjusting its parameters. A plot was generated to compare the MSE values of different neural networks on both sets. Based on the plot in Figure 5, the [7 7 7] neural network achieved the smallest MSE value.

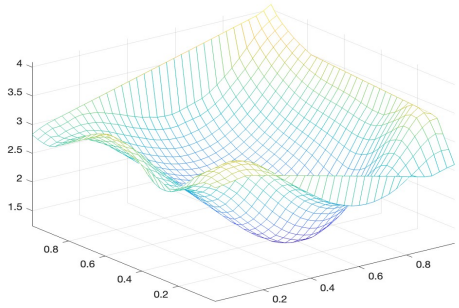
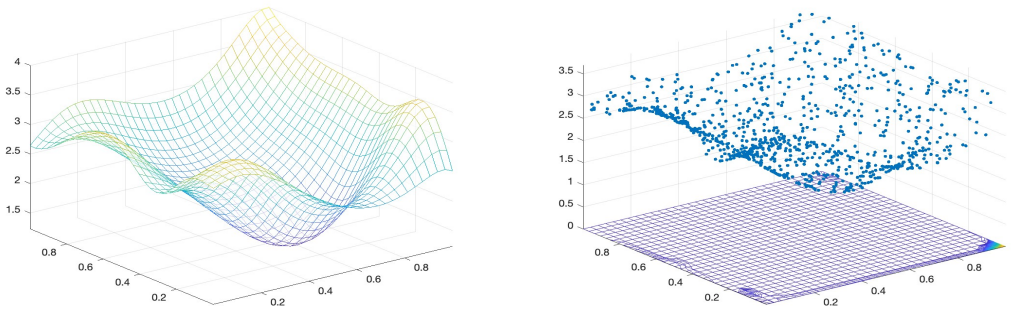


Figure 5: Training Set Surface: [7 7 7]

To plot the performance of the neural network on the test dataset, the following steps were taken. First, a grid-like input space (X, Y) was generated by the input variables x and y of the test dataset. For each point on the input space, the neural network model was used to predict the output and the predicted value was saved in the Z matrix to visualize the approximate surface. Next, the error surface was calculated using scattered interpolation. This method calculates the error at each grid point using the square of the difference between the true output in the test dataset and the output predicted by the neural network as an interpolation parameter. The error surface and error curve were then plotted to visualize the performance of the neural network on the test dataset. Additionally, the scatter data on the training dataset was also plotted on the same graph to visually compare the performance of the neural network on the test and training datasets. The results are shown in Figure 6. It should be noted that removing too many rows in the image preprocessing stage can result in an image that is too blurry to accurately reconstruct.



(a) Approximation Surface

(b) Error Surface

Figure 6: Performance of The Neural Network: [7 7 7]

1.3 Bayesian Inference

Neural networks are a type of machine learning algorithm that aim to simulate the behavior of the human brain. They consist of interconnected nodes, or neurons, that process information and make predictions or classifications based on that information. During the training process, a variety of optimization algorithms such

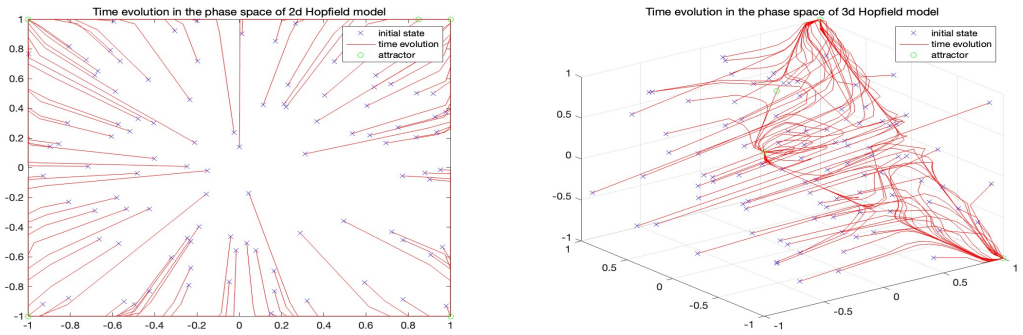
1. Explain regularization

as gradient descent and backpropagation are used to minimize the difference between the predicted outputs and the actual outputs in the training set. One challenge in neural network training is overfitting, where the model becomes too focused on the training data and performs poorly on new data. Techniques such as regularization and early stopping can be used to prevent overfitting.

2 Recurrent neural networks

2.1 Hopfield Network

First, according to the matrix T, a two-dimensional Hopfield network net is created. Then, by generating random initial points, running the simulation for 50 time steps, and storing the results in record variables, the evolution results are plotted as points in the time series. In order to compare the evolution in different cases, similar simulation and drawing operations were carried out for points on the x-axis, y-axis and two diagonal directions, respectively, and for different initial points, the evolution process and the final attractor point were plotted. In the second half, using a 3D Hopfield network, a longer simulation is performed with random initial points and the average number of iterations is calculated as shown in the Figure 7 below.



(a) 2D (b) 3D

Figure 7: Performance of different neurons

For different initial points, the time evolution process of the Hopfield model is simulated through the sim function, and the evolution trajectory is drawn.The evolution trajectories of the initial points parallel to the x-axis, parallel to the y-axis, positive slope and negative slope directions are plotted respectively. The attractors in the Hopfield network are stable, meaning that if the initial state is close enough to the attractor, it will eventually converge to the attractor after some iterations. However, if the initial state is far from any of the attractors or if there are spurious states, the network may converge to an unintended attractor or oscillate between states. The function hopdigit(noise, iterations) is a digital recovery algorithm based on the Hopfield network, which accepts two parameters: noise and iterations. When input a digital image that has been added with noise, the Hopfield network will try to find the closest pattern from the stored patterns, and output this pattern as the restored digital image, as shown in the Figure 8 below.

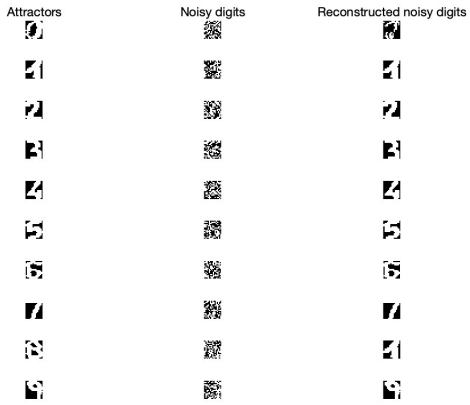


Figure 8: Hopfield Network Restoration of Noisy Digital Images: noise=1,numiter=4

To accomplish this, the network iterates through multiple iterations, updating the state of the neurons on each iteration. After many iterations, the network converges to a stable state representing the restored digital image. Higher noise levels and iterations may result in longer computation times, but may also result in better restorations.

2.2 Long short-term memory network

To analyze the performance of the neural network, the data from 'lasertrain' was split using the function 'getTimeSeriesTrainData'. This resulted in the creation of an input sequence named 'trainingX' and a target sequence named 'trainingY', with each training sample consisting of 10 time steps. The same procedure was applied to the combined 'lasertrain' and 'laserpred' data to obtain an input sequence called 'testX' and a target sequence called 'testY'. Both the input and target sequences were converted into cell vectors and used for training the network.

For each value of hidden units ranging from 11 to 20, a feed-forward neural network was created with the corresponding number of hidden units. These networks were trained using the 'trainscg' algorithm to minimize the mean squared error. During training, the mean squared error and root mean squared error were computed for both the training and validation sets, and these values were stored for further analysis. Additionally, the mean squared error and root mean squared error were calculated on the test set for each hidden unit, and these results were also stored.To visualize the relationship between the number of hidden units and the root mean squared error, a Figure 9 was plotted. The horizontal axis represented the number of hidden units, while the vertical axis represented the root mean squared error. The figure included the results obtained from the training set and the validation set.

2 Explain why you use a validation set

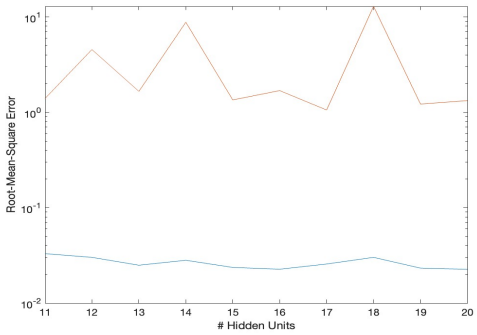
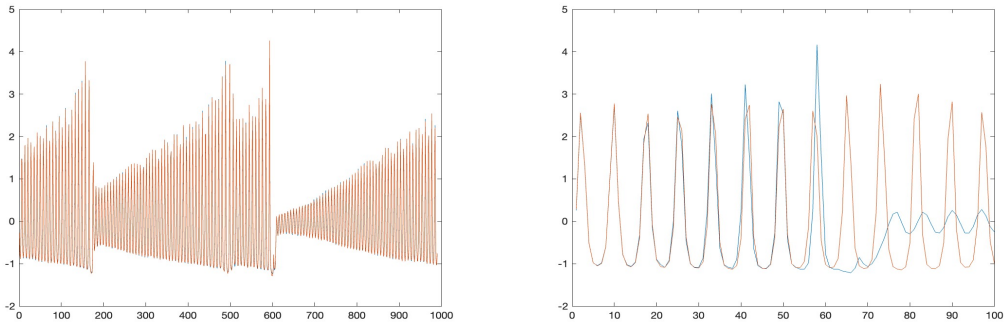


Figure 9: Effect of Hidden Units on RMSE

From the analysis of the figure, it can be observed that as the number of hidden units increased, the root mean squared error initially decreased. However, after reaching a certain point, further increasing the number of hidden units resulted in diminishing improvements in the root mean squared error. This indicates that adding more hidden units beyond a certain threshold does not significantly enhance the network's performance. Moreover, comparing the performance on the training set and the validation set, it was observed that the root mean squared error was generally lower on the training set. This suggests that the network might have a higher degree of overfitting, where it performs well on the training data but fails to generalize effectively to unseen validation data.

The model was configured to have 17 hidden units, and the training and test datasets were used to generate predictions using the model. The actual output values and the predicted values were then compared for both the training and test sets. This was done in order to evaluate the performance of the model and to see how well it was able to generalize to new data. The results were plotted in Figure 10, which shows the actual output values and the predicted values for both the training and test sets.

It performed well in the early stage, but not well in the later stage.



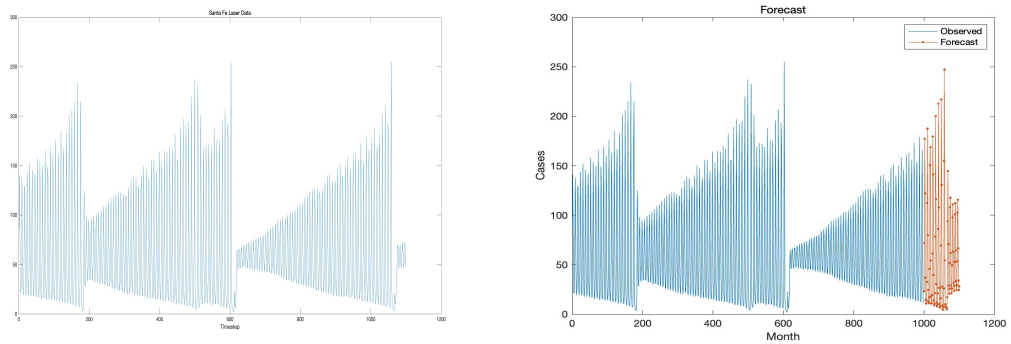
(a) Training Set Performance (b) Test Set Performance

Figure 10: Model Performance Comparison Plot for Hidden Units = 17

3 Explain difference between LSTM and other recurrent neural network

In this study, an LSTM neural network model was employed to predict the Santa Fe Laser dataset. The dataset was divided into training and test data, with the first 1000 data points used for training and the remaining points for testing. To normalize the data, the mean and standard deviation of the training data were calculated separately.

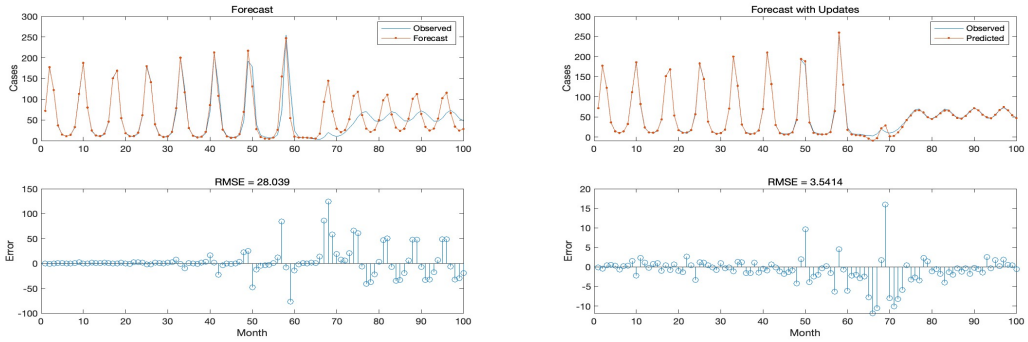
To visually represent the results, Figure 11 was plotted, including the observed values, predicted values, and errors. This figure provides a comprehensive overview of the model's predictive capabilities, allowing for a visual comparison between the actual and predicted values, as well as an assessment of the prediction errors.



(a) (b)

Figure 11: Prediction Performance of LSTM Model

The figure helps us understand the general trend of the time series data. By observing the graph, it can be determined whether the model is converging and whether adjusting the hyperparameters can improve the performance of the model. From the figure, it is evident that the LSTM model has achieved prediction performance. The observed values align with the predicted values, indicating that the model effectively captures the underlying patterns in the Santa Fe Laser dataset. These findings validate the effectiveness of the LSTM model in predicting the Santa Fe Laser dataset and highlight its capability to capture and replicate the complex temporal dynamics present in the data.



(a) (b)

Figure 12: Model Prediction Performance Comparison Plot for Future Data

Figure 12 illustrates the performance of the model in predicting future data and the deviation from the actual values. This information can provide insights into the accuracy of the model's predictions at different time steps, as well as identifying which time steps have higher or lower prediction accuracy. Additionally, this figure can also assist in identifying whether the model suffers from underfitting or overfitting issues. It can be seen that Figure 12(a) shows underfitting, while Figure 12(b) shows overfitting. And the blue graph shows the distance between each point and the target.

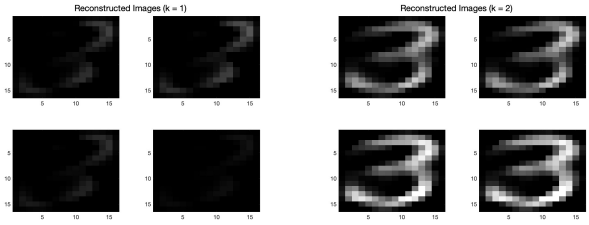
3 Deep Feature Learning

3.1 Principal Component Analysis

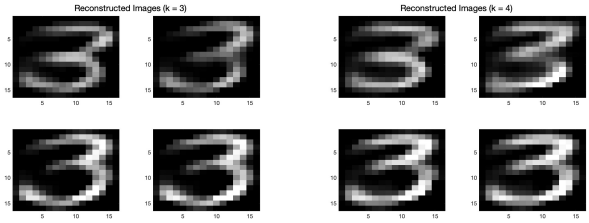
6 autoencoder里面的reconstrution error?

In the reconstruction process after dimensionality reduction using random data, the average reconstruction error is 0.66041. This means that there is a certain degree of difference between the dimensionally reduced data and the original data, and the accuracy of the reconstruction is low. This is because random data has no obvious structure or correlation, so dimensionality reduction cannot be used to capture key feature information. In the reconstruction process after dimensionality reduction using the "cholesall" dataset, the average reconstruction error is 3.5203e-16. This very close to zero reconstruction error indicates that the reduced data can be almost perfectly reconstructed back to the original data. This is because the "cholesall" dataset is highly correlated and structured, and the dimensionally reduced data can retain most of the key feature information, and the reconstruction accuracy is very high.

The Figure 13 below shows the image after the data set projected to different principal components and then reconstructed.



(a) k=1 (b) k=2



(c) k=3 (d) k=4

Figure 13: Different Principal Components

In Figure 14, the horizontal axis represents the number of principal components used (q), and the vertical axis represents the reconstruction error. By choosing different numbers of principal components for data compression and reconstruction, we can observe how the reconstruction error varies.

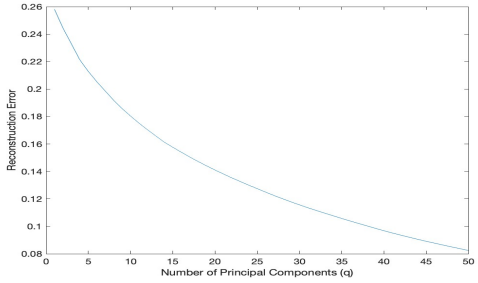


Figure 14: Function

If q = 256, the reconstruction error should ideally be zero because we are using all 256 principal components to reconstruct the dataset. In theory, this should result in an exact reconstruction of the original dataset, leading to a reconstruction error of zero. However, when we actually try it, there might be some numerical precision limitations or noise in the data that can affect the reconstruction process. Therefore, even though the reconstruction error should be close to zero, it may not be exactly zero. The actual reconstruction error when q = 256 would depend on the specific dataset and the quality of the data.

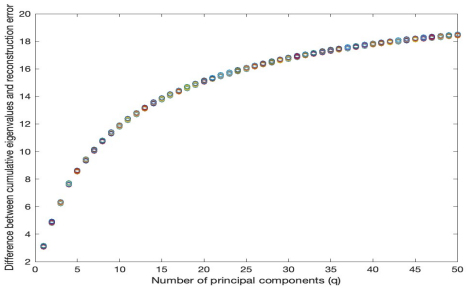


Figure 15: Difference

Figure 15 shows the difference between the accumulated eigenvalues and the reconstruction error when using principal component analysis (PCA) for data compression and reconstruction. In the figure, the horizontal axis represents the number of principal components used (q), and the vertical axis represents the difference between the accumulated eigenvalues and reconstruction errors.

For the data compression and reconstruction process, choosing more principal components (higher q value) can better preserve the information of the original data, thereby reducing the reconstruction error. Therefore, we expect that as the number of principal components increases, the difference between the accumulated eigenvalues and the reconstruction error gradually decreases. The preceding number of principal components (smaller q-values) has a stronger explanatory power for the data, and the difference between the cumulative eigenvalues and the reconstruction error is smaller. As the number of principal components increases, the difference gradually increases, indicating that the improvement effect of adding more principal components on the reconstruction error gradually weakens.

4 Explain which stacked encoder configuration works best

3.2 Stacked Autoencoders

The experiments vary in the number of layers and units of the models and the use of pre-training with autoencoders. The results show that using autoencoders for pre-training improves the performance of the models on the classification task. Specifically, the test performance greatly drops without finetuning if the number of units in the autoencoders is decreased. With fine-tuning, however, the performance is still great, outperforming MLPs with much more units. Even a single autoencoder with 50 units, if pre-trained with unclassified data, performs much better than a straightforward NN.

A deep neural network with autoencoders and softmax layers was trained to compare the performance of regular and deep neural networks with different numbers of layers. The training and test data of the MNIST digit recognition dataset were loaded, and the encoder and decoder were used to train two autoencoders and a softmax layer to build the deep neural network. After training, the test and training accuracy of regular and deep neural networks using different numbers of layers were compared. To visualize the performance and structure of different models, confusion matrix diagrams and network structure diagrams were generated.

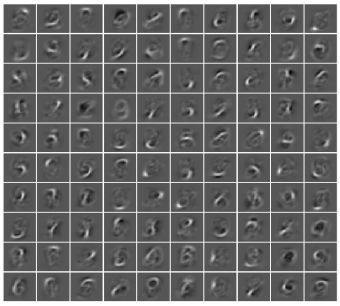
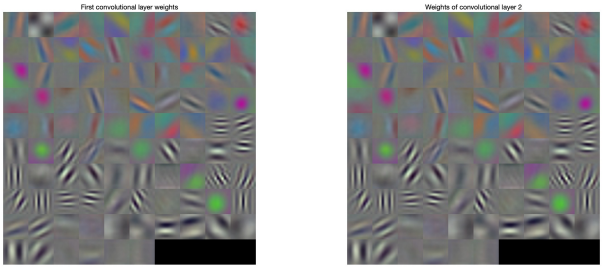


Figure 16: Digit Classification

3.3 Convolutional Neural Network



(a) The first convolutional layer (b) Weights of convolutional layer 2

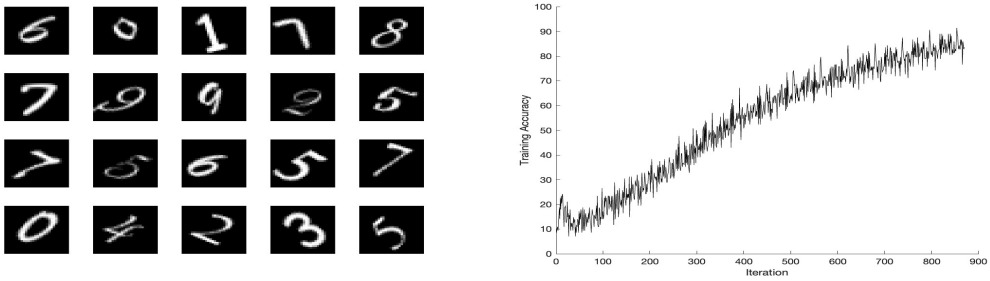
Figure 17: Restoration and Convolutional Weights

11 说一下cnn里面有什么characteristics

The dimension of the inputs before the final classification part of the network is $6 \times 6 \times 256$. This is because the last layer before the fully connected layers is layer 7, which is a convolutional layer with 256 filters, each having a size of 3×3. This dimension is much smaller than the initial dimension of $227 \times 227 \times 3$, which is the size of the input image.

The main advantage of CNNs over fully connected networks for image classification is that CNNs can efficiently capture the spatial structure and local dependencies in images, which are essential for image recognition tasks. By using convolutional and pooling layers, CNNs can extract local features and learn hierarchical representations of the image. This reduces the number of parameters needed to represent the image and makes the network more efficient and effective for image classification tasks. In contrast, fully connected networks require a large number of parameters to represent the image and may suffer from overfitting.

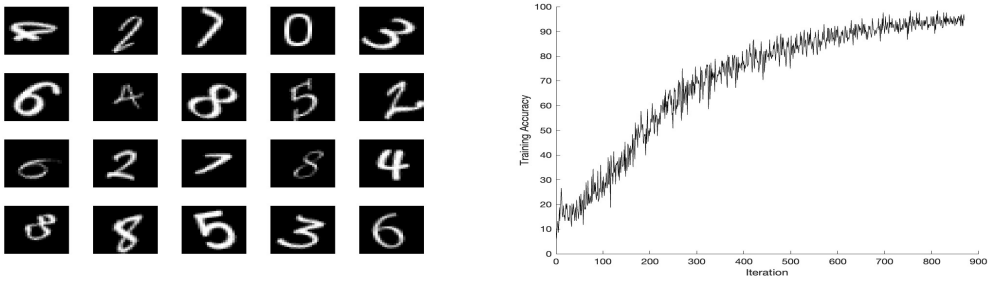
The architecture of the model was a simple convolutional neural network with two convolutional layers followed by a fully connected layer. The input layer is a 28×28 single-channel image, followed by a 5×5 sized 12-filter convolutional layer, followed by a ReLU activation function and a max pooling layer, followed by another 5×5 Convolutional layer with 24 filters of 5 size, ReLU activation function and a fully connected layer, finally a softmax layer and a classification layer.The accuracy rate reached 0.8256 as shown in the Figure 18(b) .



(a) Restoration (b) Accuracy

Figure 18: Unmodified neural network

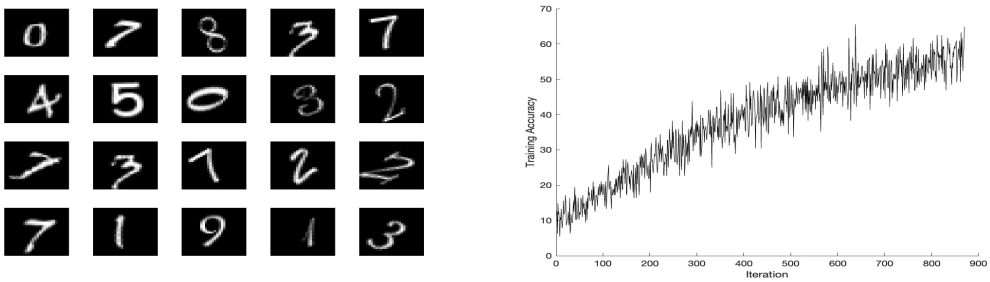
The model was modified by changing the convolution layer of the first layer from 5×5 to 3×3 and increasing the number of filters from 12 to 32. This model performs better with the addition of convolutional layers because it increases the complexity of the model.The accuracy rate reached 0.9082 as shown in the Figure 19(b).



(a) Restoration (b) Accuracy

Figure 19: The first modified neural network

The second revision of the model included the addition of a batch normalization layer after the first convolutional layer to accelerate the network's convergence. Furthermore, a batch normalization layer and a pooling layer were added. A fully connected layer was also added, outputting 500 neurons, and a dropout layer was used to avoid overfitting. Surprisingly, this accuracy rate is the lowest among the three, only 0.731, as shown in the Figure 20(b).Regarding the difference in accuracy, this is likely to be caused by different network structures performing differently on data sets and different settings of hyperparameters. If possible, experiment with more datasets and different hyperparameters to find the best combination of network structure and hyperparameters.



(a) Restoration (b) Accuracy

Figure 20: The second modified neural network

4 Generative Models

4.1 Restricted Boltzmann Machine

9 dbm和 rbm区别

In each training, the RBM is learning to model the input data by minimizing the negative log-likelihood (pseudo-likelihood) of the data.

n_components: This controls the number of neurons in the hidden layer in the RBM. If it is set too small, it may cause the model to underfit; if it is set too large, it may cause the model to overfit.

learning_rate: It determines how much each parameter changes with each update. If it is set too large, it may cause the model to fail to converge; if it is set too small, it may cause the model to train slowly.

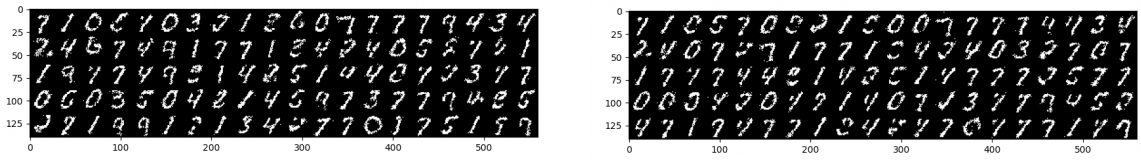
n_iter: It indicates the number of iterations to be performed during training. If it is set too small, it may cause the model to be undertrained; if it is set too large, it may cause the model to overfit.

During performance evaluation, various parameters are set as per requirement, followed by training the model using the fit function. The model is trained with different parameter values, and after the completion of training, it can be utilized to make predictions on new data or generate new samples.

Three different options were designed to perform experiments. The first option used a lower number of hidden variables (`n_components`=10) with a lower learning rate (`learning_rate`=0.01) and fewer iterations (`n_iter`=10). The corresponding pseudo-likelihood measure was -187. The second option kept the same number of hidden variables and iterations as the first option but increased the learning rate by an order of magnitude (`learning_rate`=0.1). The corresponding pseudo-likelihood measure was -148. The third option used a higher number of latent variables (`n_components`=100) with a lower learning rate (`learning_rate`=0.01) and more iterations (`n_iter`=15). The corresponding pseudo-likelihood measure was -77.

It can be seen that these three schemes have significant differences in the setting of model parameters, so they may have certain differences in the training and results of the models. Generally speaking, more hidden variables can improve the expressive ability of the model, but it will also increase the computational complexity; a higher learning rate can speed up the convergence of the model, but it will also increase the instability of the model; and more iterations The number of times can improve the fitting effect of the model, but it will also increase the calculation time. In general, a lower pseudo-likelihood indicates a better fit of the model to the data.

In image processing, Gibbs sampling can be used to sample pixels in an image to generate a new image with a similar distribution. If the *gibbs_steps* variable is set to a small value, fewer Gibbs sampling steps will be performed. This may result in new images that are less different from the original test image, but may also result in new images that are less accurate or lack detail. The Figure 21 below can prove the image after 50 steps and the image after 100 steps generated by RBM.

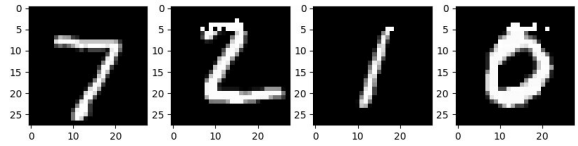


(a) Image after 50 steps.

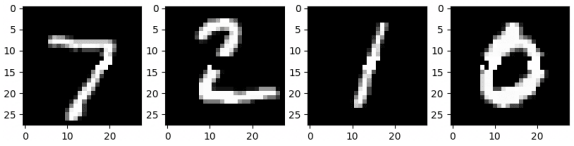
(b) Image after 100 steps.

Figure 21: Images generated with different Gibbs steps.

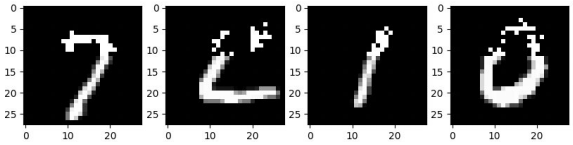
Conversely, if the `'gibbs_steps'` variable is set to a larger value, more Gibbs sampling steps will be performed. This may result in a new generated image that differs significantly from the original test image, but may also result in a new generated image that is more accurate or has more detail.The values of `'start_row_to_remove'` and `'end_row_to_remove'` were adjusted and the results were compared. It was observed that removing too many rows can result in an image that is too blurry to reconstruct accurately, as shown in the Figure 22.



(a) Removed the 3rd to 6th lines.



(b) Removed the 12td to 15th lines.



(c) Removed the 0 to 12th lines.

Figure 22: Different removal scenarios.

The hyperparameters of the RBM have a great influence on the reconstruction. When training RBMs, different hyperparameter settings will lead to different model performance and reconstruction quality.

4.2 Deep Boltzmann Machines

9 dbm和 rbm区别

The samples generated by the DBM appear to be of better quality compared to those generated by the RBM in the previous exercise. The images generated by the DBM are sharper and more recognizable as digits, with clear edges and fewer artifacts.

This improvement in quality can be attributed to the fact that the DBM has multiple layers of RBMs, which allows for more complex and hierarchical representations of the data. The first layer of the DBM learns low-level features such as edges and corners, while the second layer learns higher-level features that are composed of these lower-level features. This hierarchical learning allows for the generation of more complex and realistic samples.

4.3 Generative Adversarial Networks

The CIFAR-10 dataset was trained using Deep Convolutional Generative Adversarial Network (DCGAN) to generate new images. After training for 20,000 steps, the change curves of the four indicators during the GAN model training process are drawn. Among them, as shown in the Figure 23, the blue line represents the loss function of the discriminator, the red line represents the loss function of the generator, the green line represents the accuracy of the discriminator, and the yellow line represents the accuracy of the generator.

The loss function of the generator decreases sharply at the beginning.

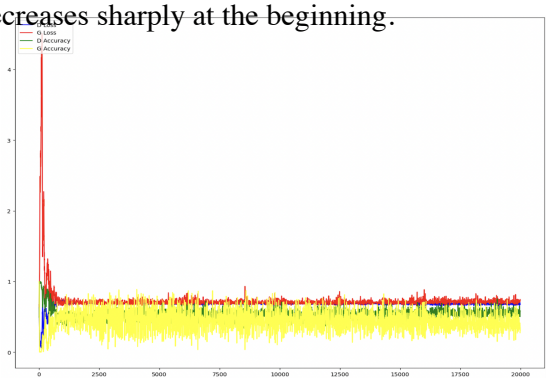
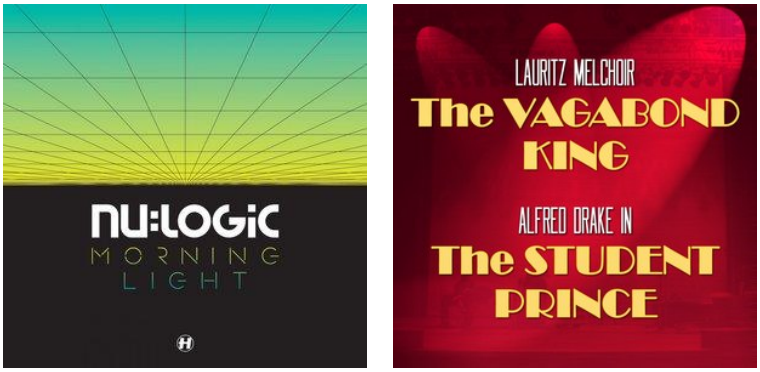


Figure 23: GAN Training Progress on CIFAR-10 Dataset: Loss and Accuracy Curves

4.4 Optimal transport

To conduct testing, two images with a resolution of 300*300 pixels were selected as the initial pictures, as shown in the Figure 24.



(a) Image 1 (b) Image 2

Figure 24: Initial images

The Figure 25 below is the result of a visual comparison of pixels in two images. In the image comparison, each pixel consists of three channels of red, green, and blue, and the value range of each channel is between 0 and 255. Therefore, each pixel can be seen as a point in 3D space whose coordinates consist of the values of the three channels.

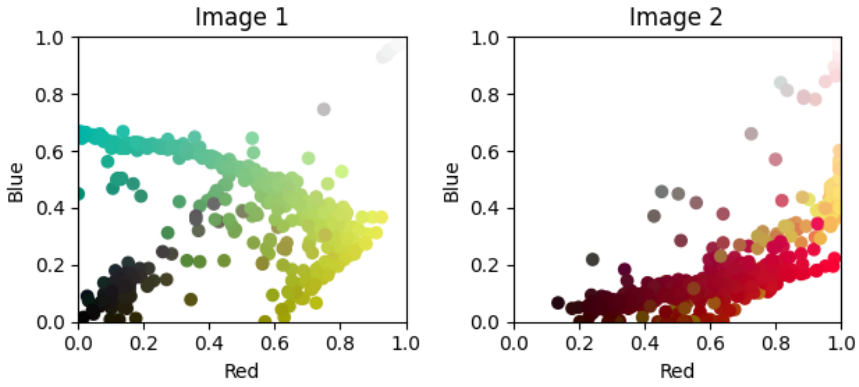


Figure 25: Transfer the colors

The image on the left shows the pixels in Figure 24(a) and the image on the right shows the pixels in Figure 24(b). In each plot, the position of a point is determined by the values of the red and blue channels, and the color of each point is determined by the values of the three channels of the corresponding pixel. The abscissa and ordinate of each graph represent two different channels respectively, so the relationship between the three channels and the difference between pixels in the two images can be compared by observing the two graphs. The Figure 26 can be observed as shown below.

Figure 26: Transfer the colors

In summary, the transport of color histograms is a more sophisticated and accurate method of color swapping that takes into account the distribution of colors in the original image, resulting in a more visually pleasing output compared to non-optimal color swapping.