DATA MINING AND NEURAL NETWORKS

Assignment Report

Abdirahman Mohamed Yassin

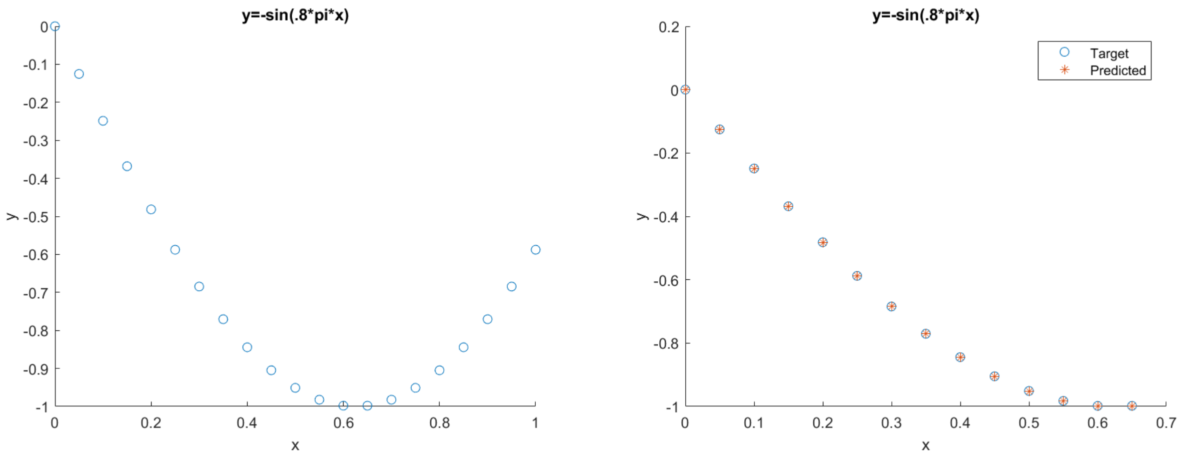
January 2021

Contents

1 Assignment 1

1.1 The preceptron and beyond

Linear model for regression, as described in the question, is supervised learning model that constructs the prediction y through a linear combination of the d-dimensional input vector. The problem becomes a least-square approximation with the objective of minimizing the sum of residuals by determining the best weights $w_1, w_2, w_3, ..., w_d$ and bias b. The perceptron consists of a simple neuron that maps the linear combination of the input vector onto the output by an activation function $a = f(w_1x_{1i} + w_2x_{2i} + ... + w_dx_{di} + b)$. The activation function for linear regression is a fucntion that returns the same linear combination $w_1x_{1i} + w_2x_{2i} + ... + w_dx_{di} + b$. This achieved by setting the activation function as the identity operator.
Figure 1 shows the non-linear mapping of the given dataset. Clearly, the generated sinusoidal output cannot be approximated by a linear model throughout the whole range of the dataset. Consequently a linear learning function will perform poorly on the complete training dataset resulting in a large error (bias) which amounts to underfitting. However, a linear model would capture the pattern of the first two-third of datapoints. This is generally true as the function $f(x) = \sin x$ is almost linear for sufficiently small x.



(a) **(b)**

Figure 1: I/O Relationship (a) Original data, (b) Predicted data by neural network

A non-linear model would be more suitable for this approximation problem. If the selected non-linear model performs too well on the given dataset, it may fail to generalize on test samples due to overfitting.

1.2 Backpropagation in feedforward mulit-layer networks

The first step of forward propagation is performed as follows

$$y = 7\overrightarrow{\mathbf{1}} + 8\sigma_1(\overrightarrow{\mathbf{1}} + 3x_1 + 5x_2) + 9\sigma_2(2\overrightarrow{\mathbf{1}} + 4x_1 + 6x_2)$$

Where $\sigma(t) = 1/(1 + exp(-t))$. Evaluating in x_1 and x_2 gives $y = (24, 14, 24)$. The next step involves the computation of the generalized delta starting at the output

$$\delta_3 = y - y_{desired}$$

After evaluating with $y_{desired} = (3, 0, 3)$, this becomes (21, 14, 21). The obtained δ is used to update the weights between the hidden layer and output layer. For simplicity η is assumed to be 0.1.

$$\triangle w_{b2,y} = \eta\delta_3 \overrightarrow{\mathbf{1}}$$

$$\triangle w_{1,y} = \eta\delta_3\sigma_1$$

$$\triangle w_{2,y} = \eta\delta_3\sigma_2$$

The updates become $(w_{b2,y}, w_{1,y}, w_{2,y}) = (7.6, 12.6, 13.9)$. The next step is to backpropagate δ_3 to obtain the $\delta_{i,h}$ in the hidden layer with corresponding initial weigths $w_{1,y} = 8$ and $w_{2,y} = 9$.

$$\delta_{1,h} = \sigma_1(\overrightarrow{\mathbf{1}} - \sigma_1)w_{1,y}\delta_3 = (33, 21, 33)$$

$$\delta_{2,h} = \sigma_2(\overrightarrow{\mathbf{1}} - \sigma_2)w_{2,y}\delta_3 = (47, 31, 47)$$

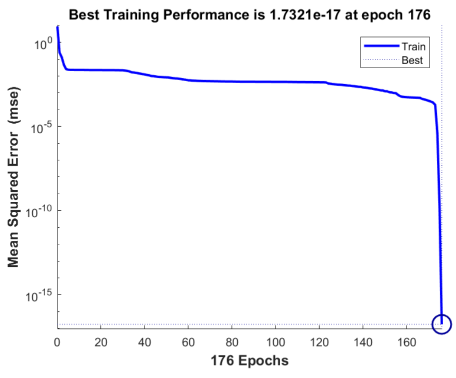
The weights between the input layer and the hidden layer are finally updated using these results using $\triangle w_{i,\sigma_j} = \eta\delta_{j,h}x_i$. The following table summarizes the results of the updates

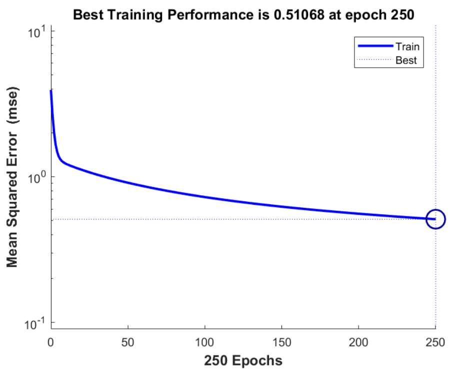
Table 1: Weight updates between input and hidden layer

	σ_1	σ_2
b$_1$	9.8	14.5
x$_1$	15.1	21.2
x$_2$	22.7	31.3

Function approximation: Comparisons of different algorithms

The gradient algorithm is compared with Levenberg-Marquardt (lm) algorithm. From lecture, it is known that the steepest descent is cheap to compute but takes longer iterations to converge. For the default setting, the training time was 2.2s with 0 noise level and 2.8s with noise level of 0.5. On the other hand Newton's method is expensive to compute due to the inverse Hessian but converges fast when initiated close enough to the optimal point. lm can be seen as regularized newton where positive value is added to the Hessian and converges faster than the gradient algorithm. Figure 2 shows lm reaches best performance faster at Epoch 176 as opposed to the gradient reaching it at full epochs. The training time was 0.9s with no noise and 1.3s with noise level 0.5. The speed is significantly higher than the gradient algorithm which is beneficial specially if the model is scaled up. Quasi-Newton method also reached its best performance at a similar to gradient method but takes fewer iterations to reach the performance of the gradient method. This is explained by the fact that the quasi-Newton tries to behave as Newton's method.

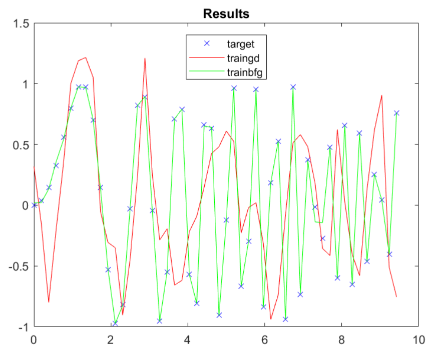


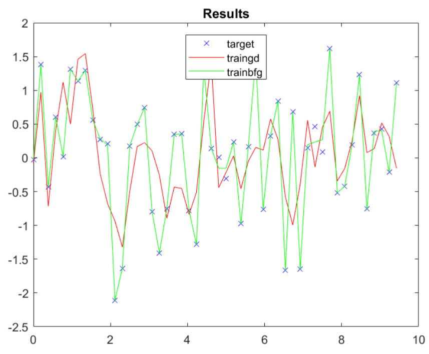


(a) **(b)**

Figure 2: Comparison of epochs and timing(a) Gradient algorithm, (b) Levenberg-Marquardt algorithm

Overall, the better performance on the training set were achieved by the above-mentioned algorithms than gradient algorithm on mean squared error basis. Since performance is not evaluated on validation set overfitting might have occured resulting in the low error.





(a) **(b)**

Figure 3: Influence of noise in gd-bfg comparison(a) Noiseless, (b) 0.5 Noise level

To evaluate the influence of noise, the noise content is gradually increased. It can be observed that the increase in noise likely prevents the function approximator to capture the underlying behavior. Figure 3 illustrates the influence of noise as both algorithms fit some of the noise and fail to distinguish it from the underlying function. As reported earlier, adding noise results in higher training time as it is difficult to capture the true function.

1.3 Personal Regression Example

The goal of this excercise is to approximate a non-linear function using unique dataset constructed by using personal student number as follows

$$T_{new} = \frac{7T_1 + 5T_2 + 4T_3 + 3T_2}{19}$$

Out of the 13600 total datapoints of the two inputs and the target, 1000 training datapoints were uniformly sampled at random. As a rule of thumb, 60% of the datapoints are used for training set. The remaining are equally divided into validation set to acquire a representative set and a test test to evaluate the model performance in the event that the model is not general enough. Figure 4 shows the surface of the training set with sharp spikes around boundary points that may be seen as outliers.

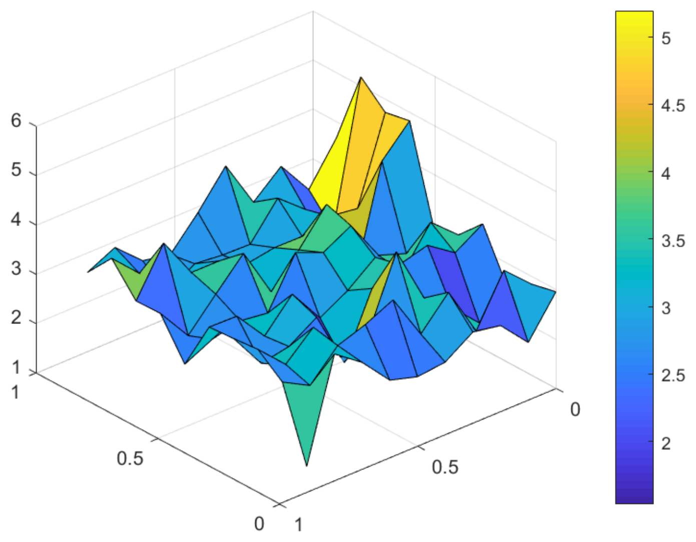


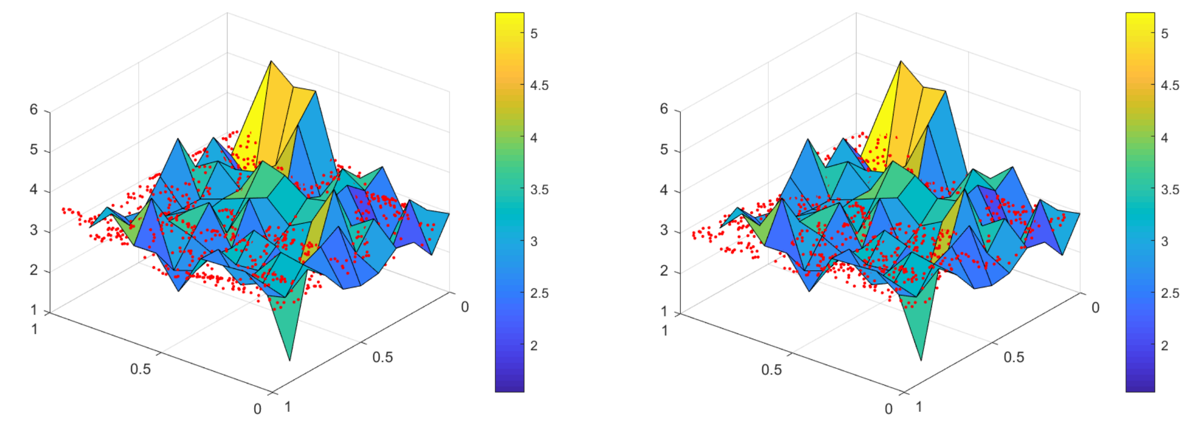
Figure 4: Surface of the training set

For the selection of the appropriate model several feedforward neural networks are trained with varying number of neurons and hidden layers. A suitable number of neurons is selected based on the performance on the validation set. This validation set is also used to decide when to stop early to avoid overfitting. Generally, a simple model with the least neurons in the hidden layer does not offer the necessary complexity to learn sophisticated patterns in the data and will likely result in underfitting. If on the other hand the complexity is increased, the model will be able to capture the underlying pattern more accurately resulting in better performance.
Table 2 shows the performance of different models based on the approximation errors on the validation set. It can seen that the errors are relatively close. As the numbers of neurons is increased, the performance is generally expected to improve. From part 1.2, the best learning algorithm was Levenberg-Marquardt (lm). We proceed with the same assumption that lm will provide the best learning result due to higher convergence speed. Linear activation function is used for the output while the hidden layer activation is the tansig transfer function.

Table 2: Performance of different models measured by approximation errors of the validation set

Number of neurons	5	20	35	50
Performance	0.3059	0.3232	0.3203	0.3248

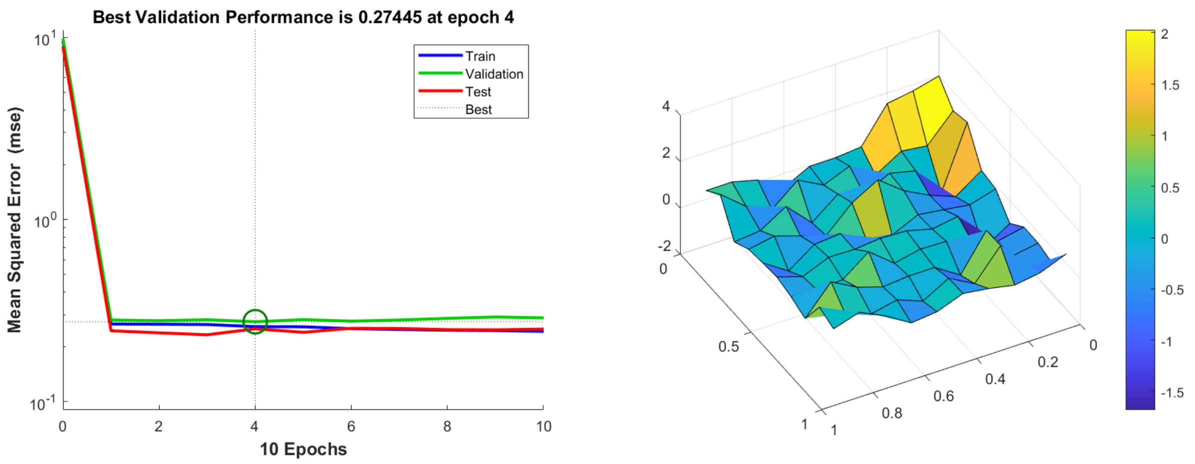
A second model (Model 2) is selected consisting of two hidden layers with 20 neurons and 5 neurons in the respective layers. Figure 5 shows the comparison between a simple model and Model 2. The two models seem similar in first instance. However, after simulating the network several times the average mse of Model 2 dropped to 0.2522 which explains the fact that higher complexity results in better performance.



(a) (b)

Figure 5: Surface plot with simulation result (a) Simple network of one hidden layer and 5 neurons, (b) Model 2: 3 Layers with 20 and 5 neurons

Figure 6 shows the loss curves and error surface on test set using Model 2. The mse on the test set using model 2 is 0.2007 which is relatively lower than the performance on the validation set. This is perhaps attributed to the fact that the model is sufficiently complex resulting in adequate generalization.
To improve the performance, regularization term is introduced. To do this, equally spaced numbers between 0 and 0.5 were used for regularization coefficient. For certain coefficients, significant change in performance on the validation set. For example the corresponding performance using the regularization coefficient [0, 0.25, 0.5] was [0.29, 0.19, 0.13]. Further increasing the coefficients results over regularization of the objective function. Another possibility is to increase the size of samples to use more data but would come at a cost of higher computational burden.

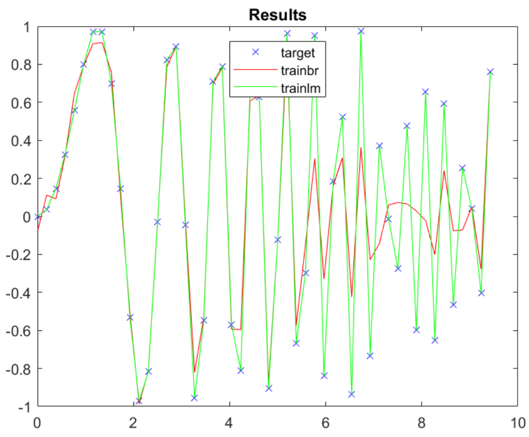


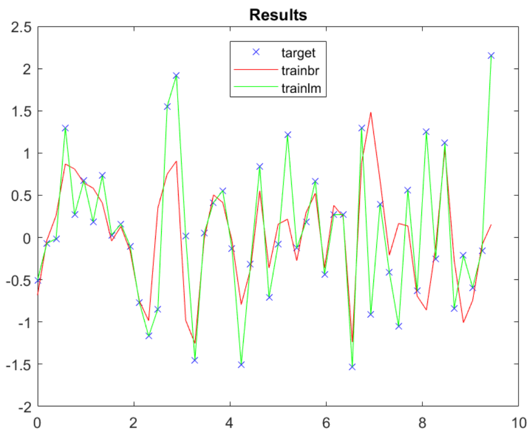
(a) (b)

Figure 6: Performance plots of Model 2 (a) Simple network of one hidden layer and 5 neurons, (b) Model 2: 3 Layers with 20 and 5 neurons

1.4 Bayesian Inference

The training time for `'trainbr'` was 3.1s while `'trainlm'` took 0.87s indicating that convergence slower due to the bayesian regularization. The training time increased to 5.7s when the noise level was set 0.5. The noise perhaps limits the bayesian regularization effect of the algorithm as it becomes difficult to find the true function behavior. Compared to `'trainlm'` , `'trainbr'` avoids overfitting due to the bayesian regularization as shown in the region around x=8 in Figure 7a while also updating the weights according to lm optimization. An overparameterized network of 100 neurons is also considered. `'trainbr'` seems to overly fit the data suggesting some loss in regularization power as the model becomes complex. As the complexity increases it becomes much more challenging for the algorithm to find the right hyperparameters in maximum likelihood framework leading to diminishing effect of Bayesian regularization.





(a) **(b)**

Figure 7: Comparison of trainbr and trainlm (a) Noiseless case (b) Noisel level 0.5

Previously, the regularization was performed according to

$$\tilde{E} = E + \nu\Omega(w)$$

where E is MSE and Ω is the regularization term with the corresponding regularization constant ν. The modeler is tasked with selecting the right ν which can be challenging to find a suitable constant. `'trainbr'` algorithm however includes finding the hyperparameters α and β needed to avoid overfitting in the algorithm itself.

$$M(w) = \beta E_D(w) + \alpha E_D(w)$$

2 Assignment 2

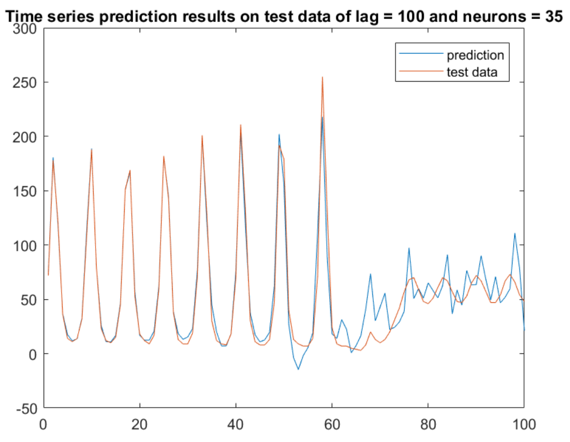
2.1 Time-Series Prediction

Time-series prediction is a common machine learning task which is analyzed in this exercise. Given a dataset consisting of 1000 training datapoints, the objective is to build a feedforward neural network that also uses previously predicted data as input to the network. The resulting network would be able carry out recurrent prediction of time elements with varying lags. These inputs are built by the `getTimeSeriesTrainData.m` which creates a matrix of with delayed time elements based on the preferred lag.
Several models with varying lags and number of neurons are simulated for the selection of the suitable hyperparameters. The mse on the test data is chosen as a performance assessment to select the most suitable model. For this, three lags of [50 75 100] and three neurons [20 35 50] is selected. Levenberg-Marquardt is used as the training function although the quasi-newton BFGS method performed in a similar fashion.

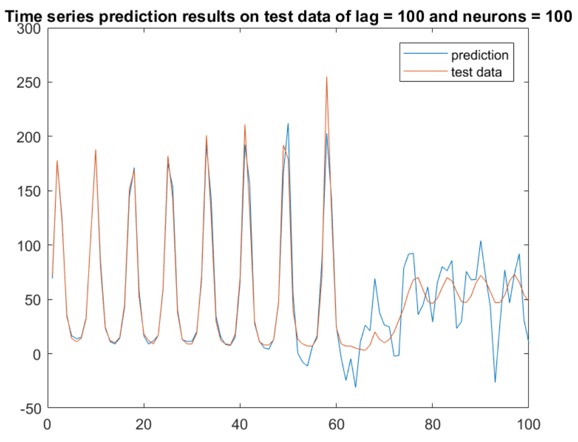
Table 3: Performance of models varying number of units in hiddel layer (N) and lags measured by approximation errors of the prediction test set

	N		
Lags	20	35	50
50	3205	4818	18455
75	4892	2229	32806
100	4229	2244	1632

Table 4 shows the result average mse after several iterations. The results show large errors the best performance with lag of 100 and 50 neurons although 35 neurons performs relatively well with lag of 75 and 100. The model with 35 units in the hidden layer is selected. The lag is is set 100 is the mse on the test data is 252. Figure 8 shows the two models [35,100] and [50,100]. It can be seen how the model [35,100] captures the underlying behaviour better in the last portion of the test data.



(a)



(b)

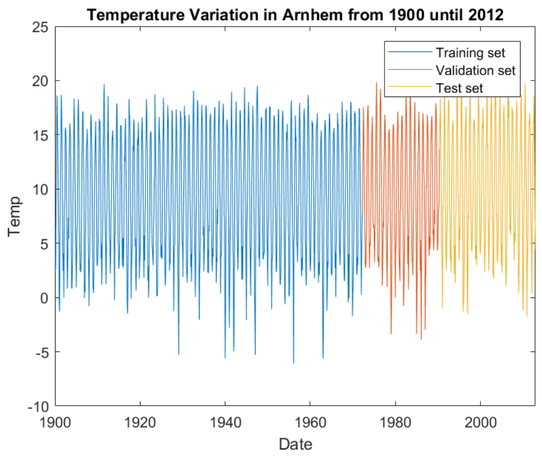
Figure 8: Prediction result on test data laserpred of the two best performing models

A certain caution should be applied when optimizing the hyperparameters using the performance on the validation set. Due to the temporal aspect of time-series, it is crucial to maintain the chronological structure when selecting a validation set. Nonetheless, a validation set that a certain timestep ahead can be selected and time-series data lagging upto a certain timestep can be used for training. The performance can then be used to to select the most suitable hyperparameters.

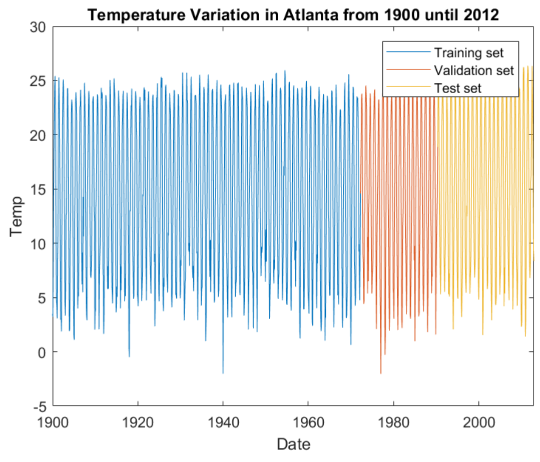
Table 4: Performance of models varying number of units in hiddel layer (N) and lags measured by approximation errors of the prediction test set

	N		
Lags	20	35	50
50	62147	46592	29786
75	4517	7193	4202
100	5147	4927	6542

Arnhem and Atlanta are considered for time-series prediction of temperature variations. Figure 9 shows the training, validation and test sets. For the analysis, three lags of [50 75 100] and three neurons [30 50 70] is selected.



(a)



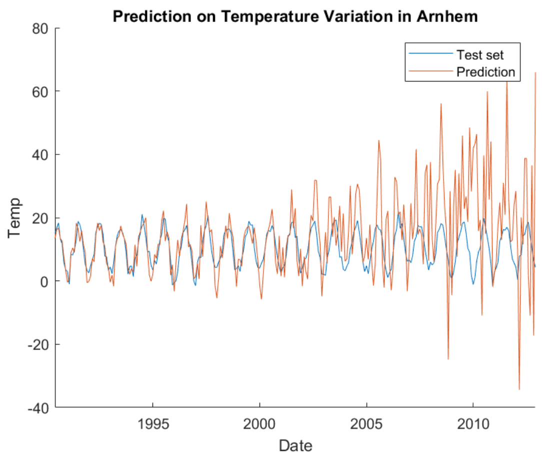
(b)

Figure 9: Temperature time-series data of Arnhem aand Atlanta between 1900 and 2012

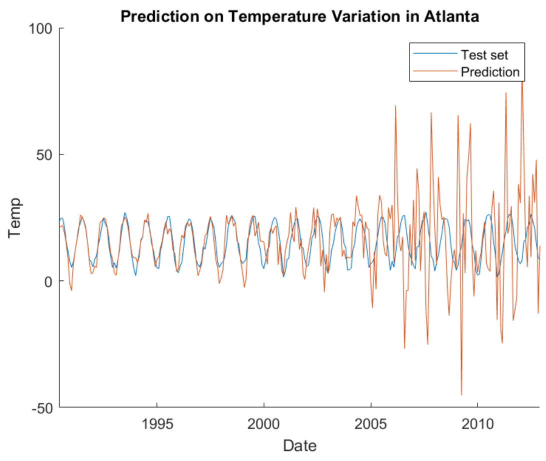
The simulation is performed once to tune the hyperparameters and then run for 5 iterations to average the MSE. Table 5 summarizes the performances of the various models. The best performing model in both cities is of 70 hidden units size and lag of 100. The error on test set was 169 and 229 for Arnhem and Atlanta respectively.
Figure 10 shows the selected model performance on the test set. The prediction in both cases under-performs in the last time steps while the error on the validation set using lag of 100 for all hidden layer sizes was relatively low. This suggests that the model is not able to generalize enough. Several options may be considered to improve the model. The complexity of model the may further be increased to achieve better performance on the test set. The lag could also be varied as this is used for the actual predicting. This however may lead to certain time delay better suited for the test set but not on new and unrelated test set.

Table 5: Performance of models of varying number of units in hiddel layer (N) and lags measured by approximation errors of the prediction validation set

	ARN			ATL		
			N			
Lags	30	50	70	30	50	70
50	112	371	565	39	474	645
75	25	57	6	78	296	93
100	7	36	4	101	27	12



(a)



(b)

Figure 10: Performance on Test set a) Arnhem b) Atlanta

2.2 Classification

The purpose of this part is to apply feedforward neural network for classifcation using the real Breast Cancer Wisconsin (Diagnostic) Data sets. The data set consists of 30 variables of 400 data points that is intended to classify breast cancer incidence by binary values {-1,1}. For exploratory analysis, t-SNE visualization is employed. t-SNE is basically non-linear dimensionality reduction which captures the non-linearity structure in the data set. The technique uses student t-distribution to project into lower dimension in non-linear fashion.

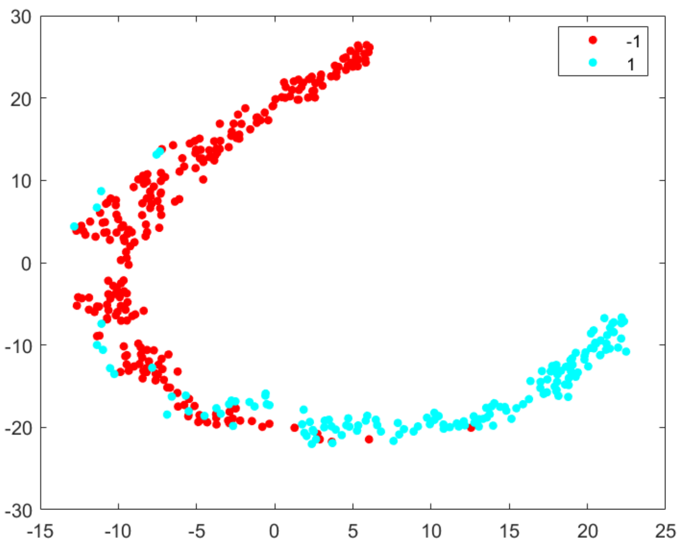
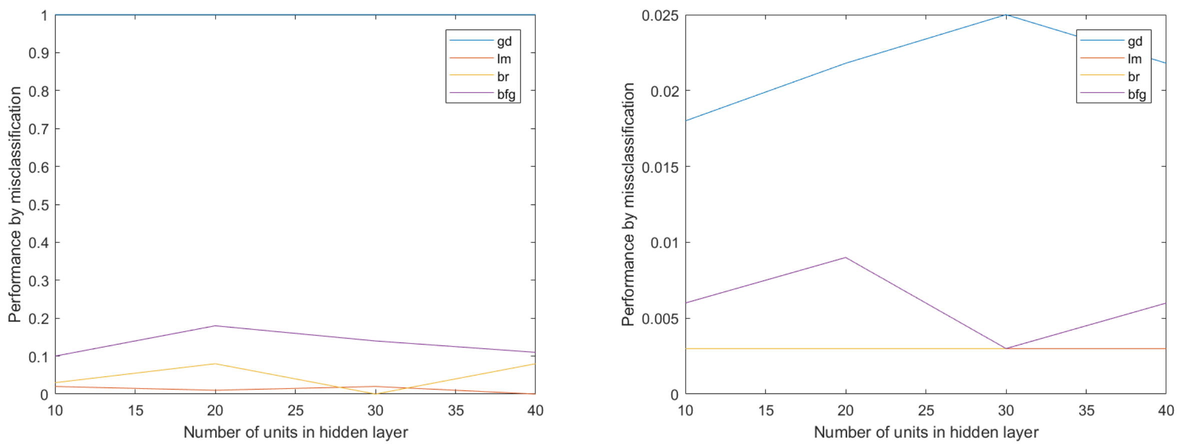


Figure 11: t-SNE Visualization of breast data set

Figure 11 shows the t-SNE 2D projection of the 30 dimensional training set using the target binary label. The two clusters can be clearly distinguished although a transition phase is also visible. The fraction of incidence 1 in the train labels is 37.5% and this distinction can be seen in the visualization as the blue points amount to nearly one third of the projection in one cluster.
A feedforward neural network with varying number of hidden units together with different learning algorithms was explored as to optimize the hyperparameters and select the most suitable model for classification. The learning algorithms are gradient descent (gd), Levenberg-Marquardt (lm), Bayesian regularization backpropagation (br) and BFGS quasi Newton (bfg) The training is carried out both with 20% validation as well 5-fold cross-validation. The performance is assessed on fraction of missclassified incidences based on the validation set. The test set is consequently used to assess the performance of the selected model.



(a) (b)

Figure 12: Performance of different models based on misclassified incidences using a) 20% fixed validation set b) 5-fold cross-validation

Performance of the explored models is shown in Figure 12. The figure shows gd as an outlying training function where every incidences were incorrectly classified. This is perhaps attributed to coding error as the same training function gd resulted in misclassification rate within certain range of other algorithms when 5-fold cross-validation was applied. But gd is overall the worst performing algorithm. The training functions lm and br resulted in the lowest misclassification rate while the quasi-newton algorithm had a higher misclassification rate. The optimal hyperparameter is then 30 hidden units. The misclassification rate for the test labels 30% which is considerably higher than the rates on the validation sets. This suggests that the model performed well on the training set perhaps resulting in overfitting. After applying regularization with λ of 0.1 and 0.4 there was no significant change in misclassification rates on the validation set. Increasing λ further resulted in twice higher misclassification rate than on the test indicating higher weights given to minimizing the parameters rather than the error.

2.3 Automatic Relevance Determination

`demard.m` demonstrates how relative importance of variable is determined. The demo illustrates that inputs with less Gaussian noise are more relevant in determining the target than those with more Gaussian noise. The demo also estimates hyper-parameters from the cost function. Each hyperparameter was associated with an interconnection weight of the network. The relevance of the inputs was determined from the hyperparameter associated with each input. The highest inverse hyper-parameter (highest variance) is selected as that corresponding to the most relevant input. As for the demo `demev1.m`, it involved Bayesian learning in which the hyperparmeters that result in increased posterior probability and minimized cost function is selected.
The Breast Cancer Wisconsin (Diagnostic) Data is once again considered for selection inputs according to relative importance using the concept of automatic relevance determination (ARD). As in the demo's, the hyperparameters are all initialized 0.01 with 2 hidden units. The selected learning algorithm is scaled conjugate gradient backpropagation (scg). After initially training the network, the most relevant inputs are selected based on the values of the hyperparameters. Table 6 summarizes the selected inputs with the corresponding hyperparameter values. After training the network model from previous exercise using these 7 inputs, the performance was on the test set was 37% misclassification rate which is close to model's performance on the test set using all 30 inputs.

Table 6: Relevant input and corresponding hyperparameters

Input	7	8	17	18	26	27	28
α	0.0019	0.0035	0.0024	0.0067	0.0046	0.0016	0.0046

3 Assignment 3

3.1 Self-Organizing Map

The data `banana.m` is considered for unsupervised learning using SOM. Firstly, the data topology is explored. Figure 14 highlights the topology of the data. The topology can be seen as clusters of two banana shaped distribution of datapoints facing in opposite direction.

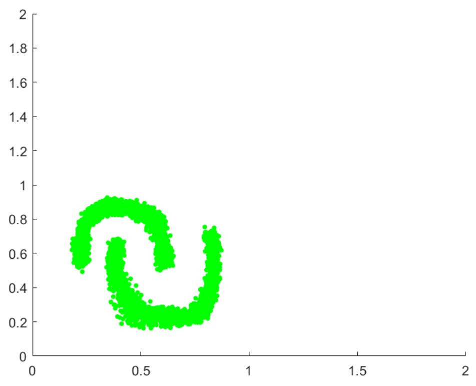


Figure 13: Topology of the dataset `banana.m`

Consesequently, a hexagonal pattern is selected for the original locations of the neurons after exploring gridtop configuration. The structure of Figure 13 needs to be maintained after training by finding the appropriate weight vector for each neuron in the hextop setting.

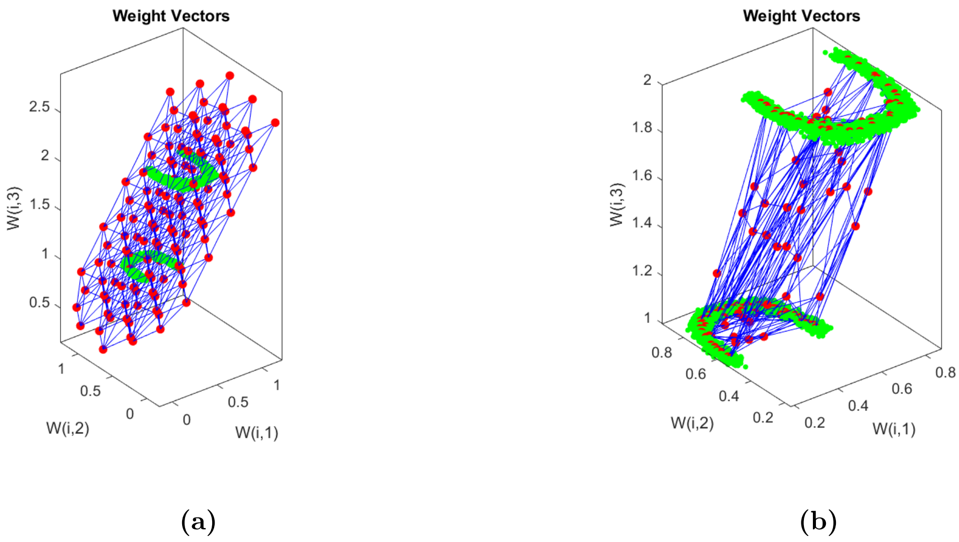


Figure 14: SOM on synthetic data `banana.m`: a) Prototype distribution prior to training, b) After training

The original distribution of the prototypes is shown Figure 14a.The topology of the data set is also visible. The SOM mapping is shown in Figure 14b. It can be seen the best matching units are rearranged according to the topology of the data set while other neurons remain visible in their previous locations.
The next part of this excercise concerns the Covertype data set. This is large real data set of 54 dimensions and 581012 samples. For computational purposes, a 5x5 hextop grid is considered for training. Additional 9x5 and 10x10 grids are then considered with increasing number of epochs of 100 and 200. The performance is assessed on Rand Index (RI) which adjusts for the probability of chance agreement of clusterings.

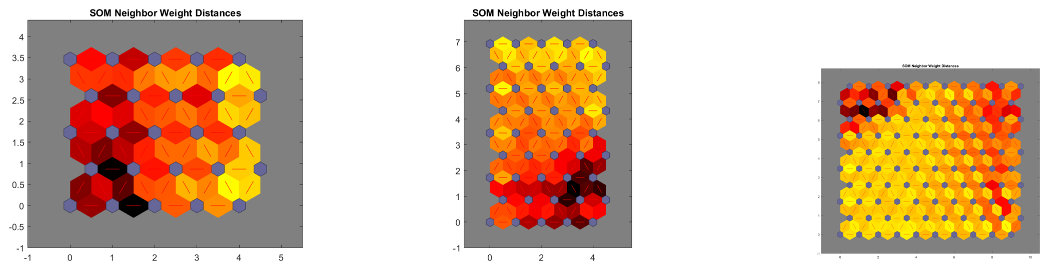
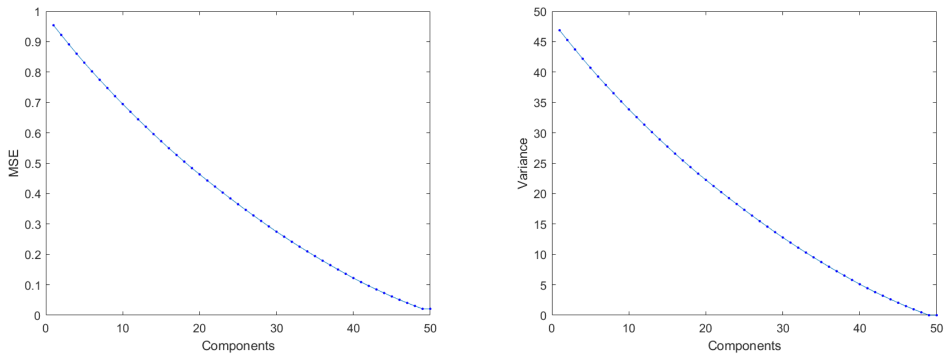


Figure 15: SOM on Covtype dataset

Comparing the three prototype configurations, four different clusters can be visualized although these clusters are less distinguishable from the 5x5 grid. This is perhaps attributed to the fact a smaller grid is less likely to capture the non-linearity resulting in lower resolution. The RI of the three grid configuration was similar fluctuating between 0.6 and 0.7.

3.2 Principal Component Analysis

The first part of this exercise concerns a PCA analysis of generated Gaussian random numbers. The generated data set consists of 50 columns of 500 samples. Firstly, a preprocessing step is considered in which the magnitude of the eigenvalues with corresponding component is visualized. Secondly, the data is centered and all 50 components are considered for dimensionality reduction to assess the reconstruction error after decentering. `linearpca.m` is used for the projections.
From the set of eigenvalues, there was no clear corresponding eigenvector which is dominant enough in which most of the present variantion in the data set is contained. The eigenvalues decrease almost linearly with the components suggesting the reconstruction error will follow the same trend. For the latter, the MSE of the original and the reconstructed matrix is considered.



(a) **(b)**

Figure 16: Reconstruction error a) MSE of increasing components, b) Total variation contribution with increasing components

The MSE, as shown in Figure 16 and expected from preprocessing phase, is almost linearly and inversely proportional to increasing component. The marginal decrease in the error is significant as more components are considered for the reduction.

Similar steps are followed for the `cholesall` data set. The data set contains highly correlated variables with at least 4 variables nearing perfect correlation.This can seen from either the covariance or correlation matrix. Consequently, some components are expected to contain most of the variation and should be visible in the preprocessing plot of eigenvalues versus components.

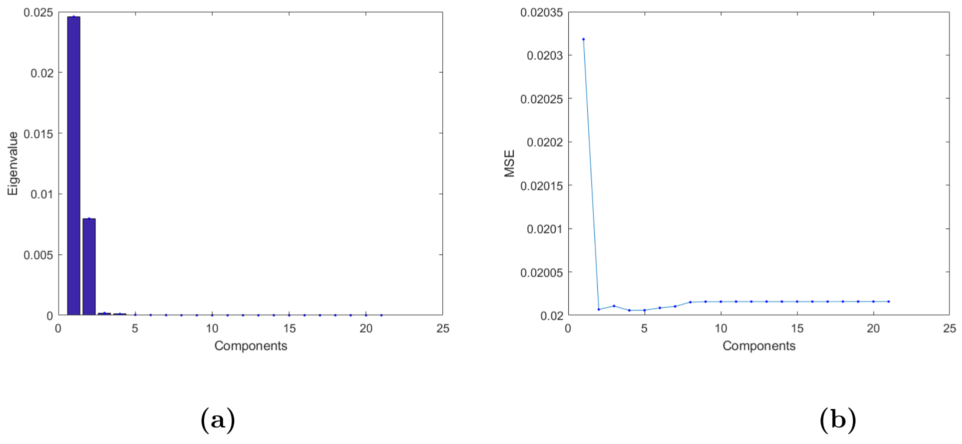


Figure 17: a) Eigenvalues versus components, b) MSE of original and reconstructed matrix

Compared to the random data in the previous part, two eigenvalues stand out with one of the corresponding components or both containing most of the variation in the matrix. Moreover, reducing to two dimension results in insignificant marginal change in the MSE as shown in Figure 17b. If single component is used, about 25% of the information is lost. The loss reduces to 1% if projected on both components.
The previous results are also compared with `preprocesspca` after preprocessing with `mapstd` and carrying out the reverse operation for both. Four components are used to eliminate variations that contribute less than 0.1%. The reconstruction error was signifcantly lower than using four components with `linearpca.m` . This is perhaps attributed to the difference in zero-meaning the data as was the case with linear PCA and standardization in `mapstd`.

3.3 Autoencoder

This section concerns autoencoder as an unsupervised learning algorithm that has the properties of typical neural networks previously considered but with key differences. Firstly, the input size is set equal to target size. Additionaly, minimazion of reconstruction error in the cost function is supplemented by sparsity constraint as multi-objective optimization problem.
A handwritten digits dataset consisting of 1x5000 cells each containing 28x28 pixel representation is considered for reconstruction using autoencoder. The hyperparamters of the given example `aereconstruction.m` were intitially set at 'MaxEpochs', 1000, ... 'L2WeightRegularization',0.004,... 'SparsityRegularization',4,... 'SparsityProportion',0.15 with hidden layer size of 50. The reconstruction of this setting does capture primitive patterns such as circular en straight lines features and but fails to reconstruct slight sophisticated patterns such as multiple crossings of lines or curvatures. This might suggest that the hidden layer is too small for the given input size. The sparsity proportion might also be insufficient as ideally sparse network is preferred to meet the optimization criteria. The high reconstruction error could also be attributed to high regularization of the weights.

The hidden layer size is gradually increased to 60 and 70 while also increasing the sparsity proportion to 0.2 and 0.3. While relatively better resolution is observed, the reconstruction improvement is not significant as shown in Figure 18.

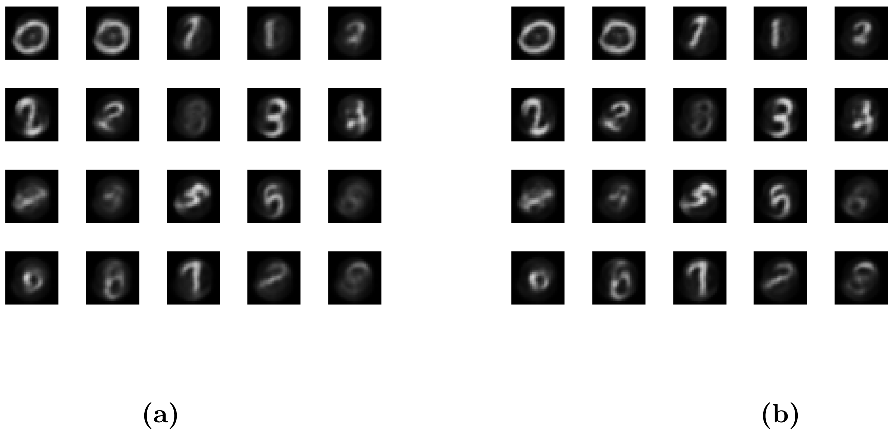


Figure 18: a) Default reconstruction b) Reconstruction using hidden layer size 70 and sparsity proportion 0.3

The previous results suggest that some hyperparameters need to change considerably to improve ther reconstruction error but also avoid the identity mapping. For this, the hidden layer size increased to 200 but the maximum number of epochs is reduced to 500. The l_2-regularization of the weights is reduced to 0.0004 and the sparsity proportion is set to 0.3. The results are depicted in Figure 19. As it can be seen, the reconstruction significantly improved while perfect reconstruction is avoided.

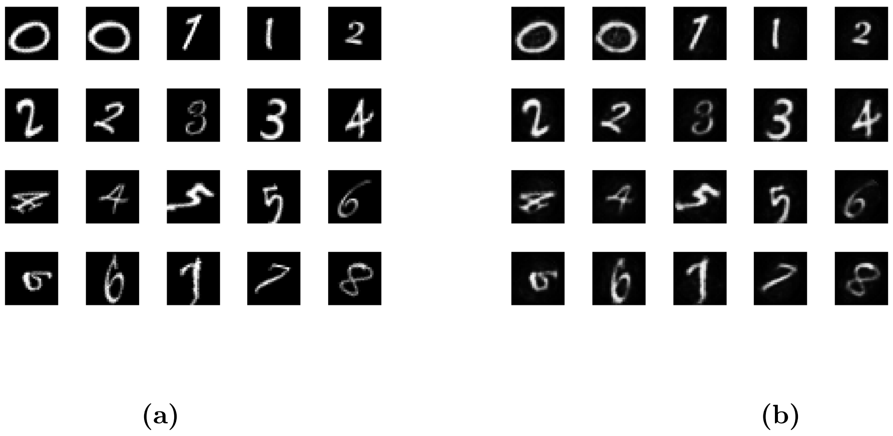


Figure 19: a) Original handrwritten digits b) Reconstructed handwritten digits

3.4 Stacked Autoencoders

The fourth exercise involves the study of stacked autoncoder for digit classification. The `StackedAutoEncodersDigitClassification.m` consists of 2 stacks of autoencoders for unsupervised learning and a softmax classifier for the supervised learning part. The default setting is observed to have misclassification rate of 12.8% without fine-tuning and 0.3% with fine-tuning indicating high accuracy.
Deeper network is likely to capture more complex features. To explore this, a third layer is added with hidden layer size of 25 and similar hyperparameters as the second encoder

resulted in a significant accuracy reduction to 27% without fine-tuning while the results remained the same in case of fine-tuning. The individual layer were further tuned to [200 100 50] for the hidden layer size respectively and [0.25 0.15 0.1] for the sparsity proportion followed by the same softmax classifier. The accuracy increased by 10% without fine-tuning the deep network. Fine-tuning resulted in similar results as in the case of 2 stacked layers. This suggests that applying fine-tuning improves the accuracy by applying backpropagation on already initialized and pre-trained layers as opposed to applying it from the beginning with random initialization. The latter might result in slow optimization due to diffusional grandients. Furthermore, it can be concluded that 2 layers is sufficient to achieve better performance than normal neural network.

4 Assignment 4

4.1 Weight Initialization & Batch-Normalization

Figure 20 shows the result van of a single run of 8 layer network with and without batch-normalization to study the interaction of batch normalization and weight initialization. Two effects of batch-normalization can be observed as it pertains to how the weight initialization affects the model. Firstly, the loss curve shows that the loss is infinitely high at relatively large scaling indicating that initialization has considerable impact on final results. Batch-normalization compensates for this high training loss. Large scaling of the weight initialization might result in output of the non-linear activation function to remain in the extreme regions of the activation function. This in turn results in gradients approaching zero. With batch-normalization, the gradient remains unchanged upon updating as the output of each layer is an approximated Gaussian distribution. As a consequence, the effect of weight initialization is not significant.

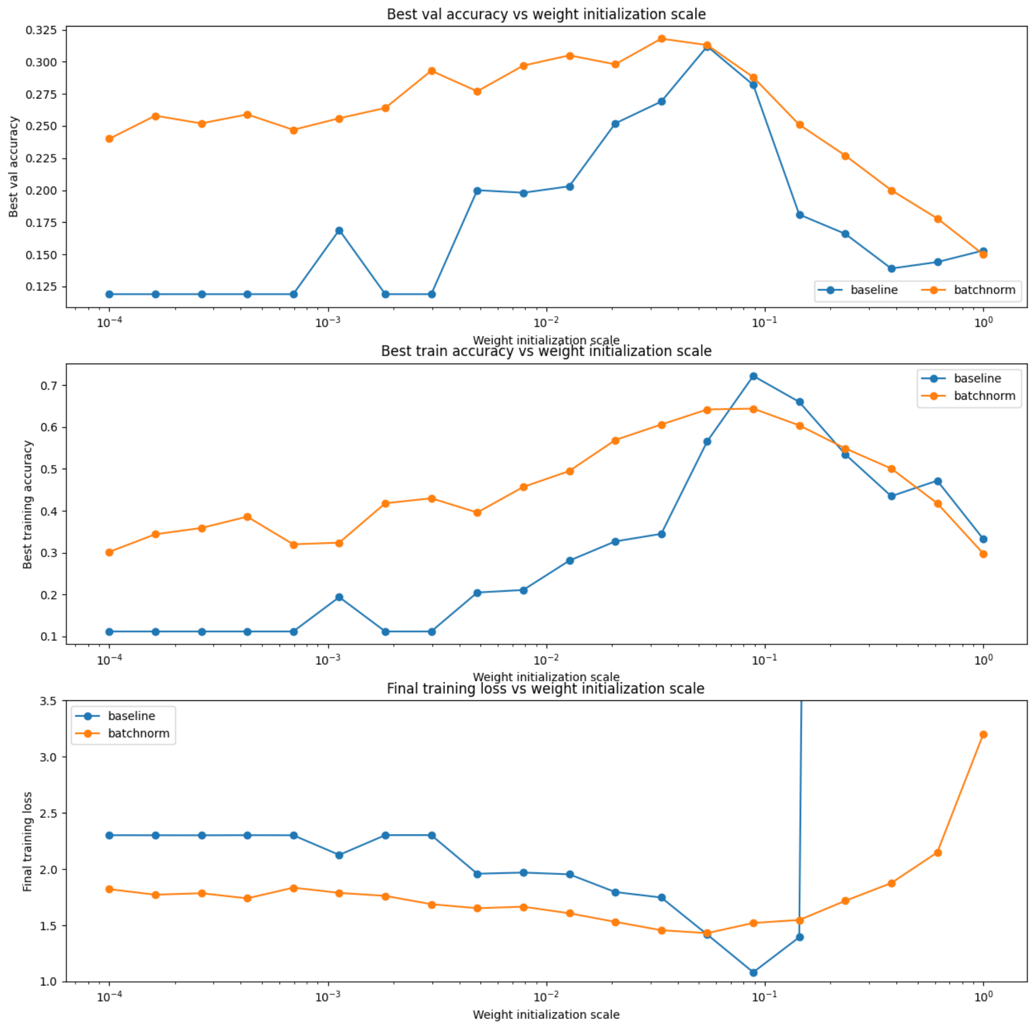


Figure 20: Results of weight scale experiment

Secondly, the baseline over performs the batch-normalized model in the training accuracy around scale 0.1. However, the validation accuracy is higher with batch-normalized model indicating certain degree of overfitting with the baseline. This can seen as l_2 regularization if batch-normalization is applied resulting in less overfitting. The analogy of regularization can also be deduced from the curves as it favors smaller weight scaling which explains the higher accuracy in both training and validation if the weight scaling is low.
Figure 21 shows the effect of batch size on the on the batch-normalized model's accuracy. The training accuracy increases almost directly proportional to batch size specially in later epochs which is likely due to overfitting. But the accuracy proportionality decreases in the validation set indicating that there is some regularization. This is perhaps explained by the roles of the mini-batch statistics as these are directly related to the batch size. It is possible that increasing the batch size diminishes the significance of the μ and σ thus approaching the baseline characteristics. As the size decreases, the regularization effect leads to higher validation accuracy.

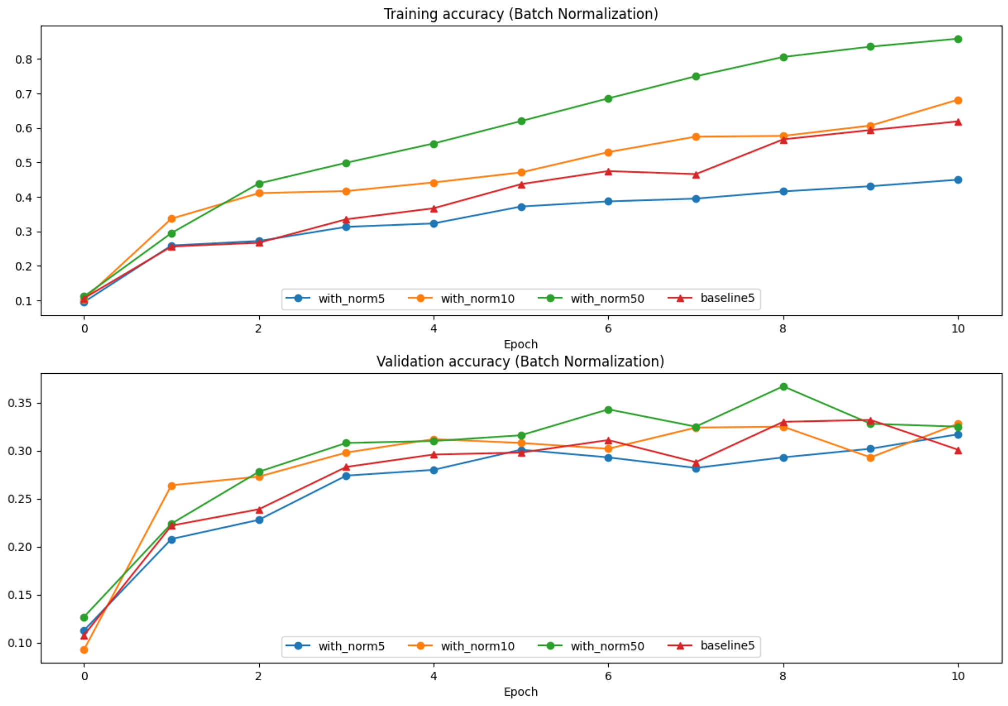


Figure 21: Interaction of batch normalization and batch size

4.2 Variational Autoencoders

Figure 18 is once again considered and compared with the output of the VAE experiments and highlighetd in Figure 22. Without first considering the reconstruction method in each case, the reconstruction error is visualized differently. Specially in contrast to VAE, the stacked autoencoder seems to have less to no noise as it tries to mimic the input. If the reconstruction error is large, we would see no output or completely unrelated output as shown in zero digit reconstruction in top right corner of Figure 23. The reconstructed image looks almost like 8. Poor VAE reconstructions are similar to stacked autoencoders in the sense they both to try gradually build up to the output using primitive features. The difference here is however the output contains much more noise (more added features) if reconstruction error is large suggesting that some sampling from underlying distribution is involved. According to the mean and the variance of the distribution, VAE reconstructs some features that are similar to the input using varying input values with mean and deviation. Digit 2 in position 3,2 in Figure 22a illustrates this phenomena where noise content is high likely indicating reconstruction from samples that are far from the needed values.

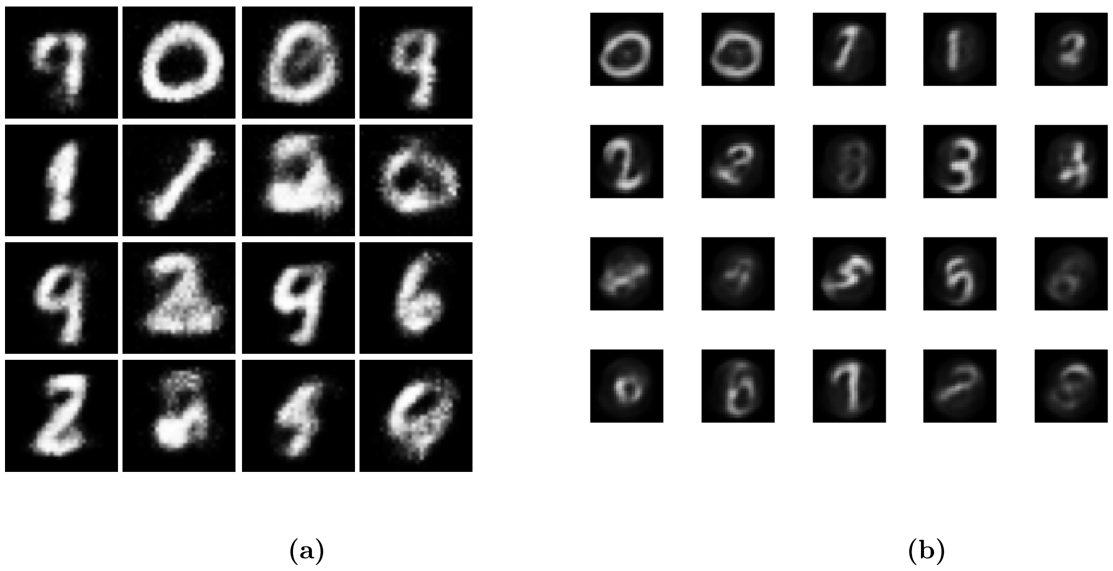


Figure 22: a) Reconstruction using VAE b) Reconstruction using stacked autoencoder

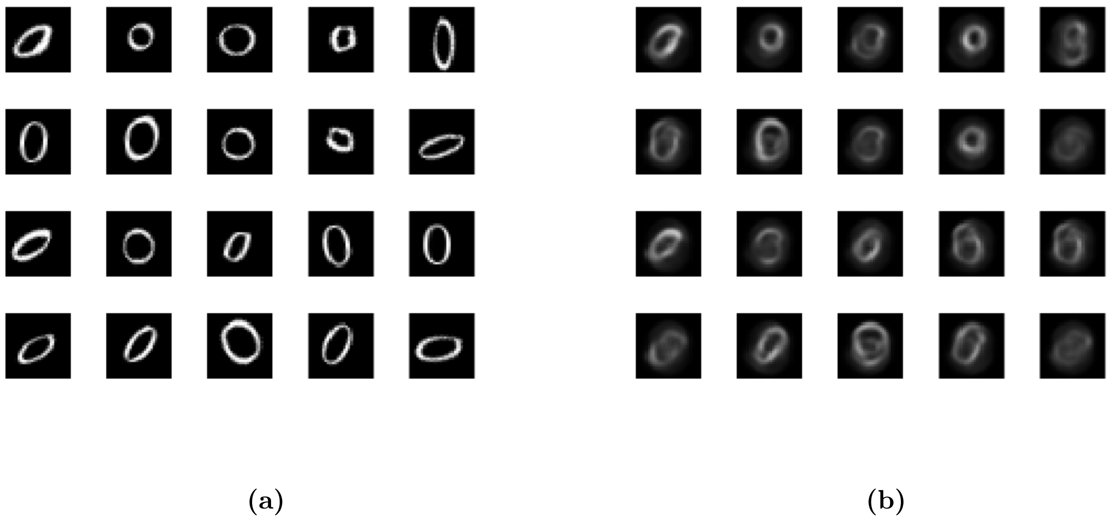


Figure 23: Digit zero reconstruction using Stacked Autoencoder a) Test b) Reconstruction

The stacked autoencoder tries to minimize the difference between the input and the decoded output accompanied by sparsity constraint.

$$\|input - decoder(code)\|^2 + sparsity(code)$$

The reconstructed error is therefore directly related to the difference between the input values and the values of the decoder. The reconstructed might not have common patterns with input if the difference is too large. The above-mentioned procedure is similar to VAE from architectural point of view but the key difference is the input and decoder output characteristics (fixed vector vs distribution) and how they are compared. Following the theory of VAE, the goal is to maximize the likelihood of original input being reconstructed once the appropriate sampling of input given latent variable $(x|z)$ is performed. So the metrics are on one hand distance between two vectors for the stacked autoencoder and maximum likelihood of certain input in case of VAE. The latter can also be seen as regularization as it constrains the way probability distribution is determined by including the prior $p(z)$.

The optimizer used in the stacked autoncoder is scaled conjugate gradient descent. The optimizer is based on conjugate directions but avoids line search methods which is advantageous as the computations become less expensive without line search. This is achieved by combining model-trust region approach of Levenberg-Marquardt algorithm with conjugate gradient approach[1]. A possible drawback as it pertains to training an autoencoder is that it may require more iterations to converge.
The optimizer used in the VAE experiment is ADAM, which is an adaptive method for updating optimization steps. The algorithm scales the learning rate by squared gradients and uses exponential moving averages of the gradients as a warm-start to speed up convergence due to greater weights of recent gradients[2]. Some cons are highlighted in examples of its limts regarding weights in deep learning [3]. Specifically, convergence may fail or the limiting weights may fail to generalize poorly leading to poor performance in test data. Another example is given in [4] in which the algorithm moves in the wrong direction twice and only in the right direction every three steps. However the advances booked in the latter step is scaled down by the exponential decay and the learning rate does not decrease.

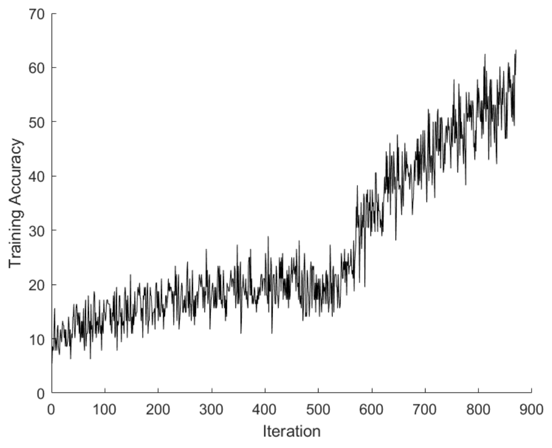
4.3 Convolutional Neural Networks

The given kernel $\in \mathbb{R}^3$ is convolved with x $\in \mathbb{R}^6$ resulting in out y $\in \mathbb{R}^4$. The required toeplitz matrix A $\in \mathbb{R}^{4x6}$ matrix should consist of the kernel and perform row-column inner product with x leading to the desired output. To achieve this, the kernel is should occupy the first 3 positions in the first row and slide one position to the right in the next row and so forth. The convolution is captured by this moving inner product which can be seen as a combination of shift matrices. The required matrix is $A = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

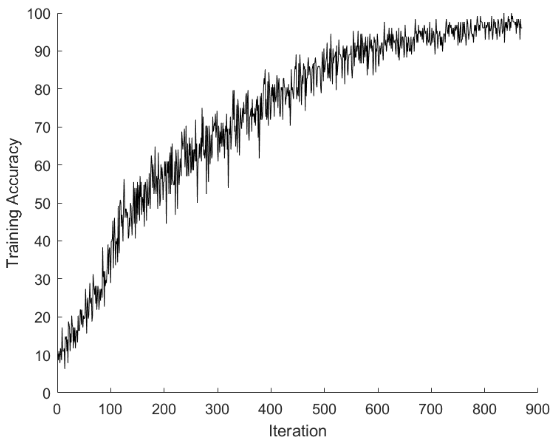
$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

x can be recovered from $A^T y$ if $A^T = A^{-1}$ and which is valid if A is orhtogonal matrix. A is not orthogonal so x not recoverable in this way. A would be orthogonal incase of circular convolution which makes such reconstruction of x possible.
The weights represent 96 11x11x3 convolutions. Moreover, the 227x227x3 images are convolved with 96 filters with dimensions 11x11x3. The demo uses stride 4 and no zero padding, resulting in 96 outputs with dimensions of 55x55x3. The fifth layer consists of 3x3 max pooling with stride 2. The operation doesn't change the depth. The final dimensions of 96 outputs are 27x27x3. Matlab allows for the Number of class names for ImageNet classification task. The size is 1000. As a result, 1000 neurons are used to for classification in the last fully connected layer. The size is considerably smaller than initial dimensions of 227x227x3

To explore digit classification, different configurations were simulated. Firstly, filter dimensions was decreased to 3x3x12 to boost local feature extraction. The max pooling dimensions were increased to 3x3 to reduce number of computations (Setting 1). The training accuracy reduced to 88%. The filter dimensions were increased to 7x7x12 and 7x7x24 to see if the accuracy would further reduce (Setting 2). In the contrary, the training accuracy increased back to 96% which relatively the same as the default filter sizes of 5x5x12 and 5x5x24. This indicates the default setting is highly accurate on the training set but perhaps overfitted. Another option to improve performance was to configure the network in the first step deeper. For this, an intermediate convolutional layer of 7x7x12 is added before the first layer followed by 3x3 maxpooling and ReLU (Setting 3). The setup resulted in accuracy reduction to 55%. This may be attributed to the fact that some local features were lost in the added layer. The result can be seen in Figure 24 where the accuracy kept oscillating bet 10% and 20% in the first 500 iterations.



(a)



(b)

Figure 24: Digit reconstruction using Convolutional Neural Network a) Setting 3 b) Setting 2

5 References

[1] Moller, M. F. A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning, Neural Networks, Vol. 6, 1993, pp. 525–533

[2] Kingma, D. Jimmy, a. Adam: A method for stochastic optimization. ICLR, 2015

[3] Hardt, M., Recht, B., Singer, Y. (2016, June). Train faster, generalize better: Stability of stochastic gradient descent. In International Conference on Machine Learning (pp. 1225-1234). PMLR.