Table Of Contents

R0779613
Cemal Yetismis

LAB-1

Personal Regression

1. Data set was split into %80 train, %10 validation, and %10 test set. Since the number of given data is limited, the validation and test set were chosen %10. When the number of data points becomes bigger, the validation and test set can be chosen %20 or %15 for each. As it is known, the validation set is pretty important to overcome overfitting. Artificial neural networks are very powerful tools; however, they are prone to overfitting easily. For this reason, the early stopping mechanism was chosen to eliminate overfitting in this example. If the error in the validation set starts to increase while the error is decreasing in the training set, the learning process stops. In other words, when the distance between the validation curve and training curve starts to grow, overfitting, namely variance increases. Hence, the learning process cannot go further from this point.

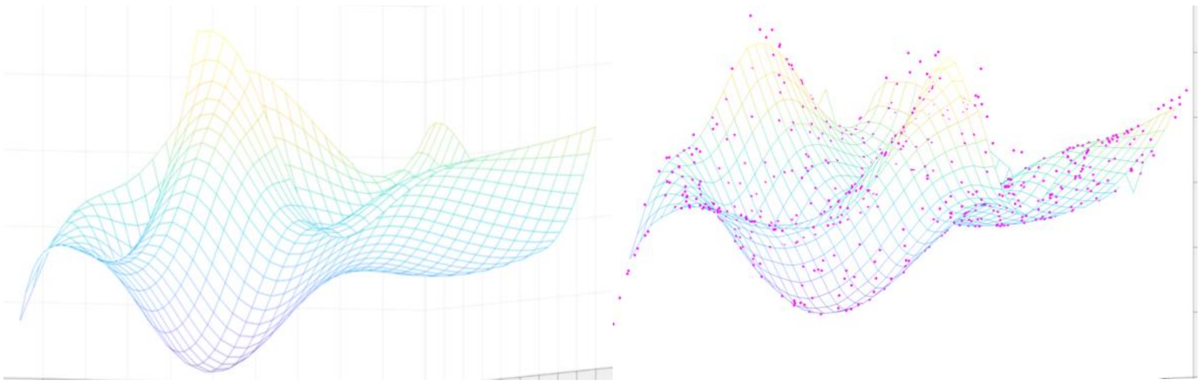


figure-1 Surface of the data training data set

2. When the surface of the data *(figure 1)* is examined, it is easily seen, the shape of the surface is non-convex. For this reason, one concludes that choosing more than one hidden layer is a convenient option for the problem. Hence, the two hidden layers were chosen.

Considering the activation functions in hidden layers, generally using ReLU is the first choice nowadays. Because when the z values are very large or very small then the derivative of the function becomes very small. This slows down the gradient descent.

With regards to the given data set, however, tanh function ends up with a better result in a validation set with the combination of the Levenberg-Marquardt algorithm. It gives the $RMSE$ between e^{-7} and e^{-10} . But this is not the case with the other algorithms such as Quasi newton and Conjugate gradient descent.

Considering the ReLU function, the results are almost same for Quasi newton and Conjugate gradient descent. One possible reason for the bad results for ReLU can be the range of the input data which is between [0,1]. Due to fact that the z values cannot be very large or small. On the other hand, concerning the output layer the linear activation function, which is the default in MATLAB, is chosen. It works efficiently for a regression problem.

After trying different number of neurons, approximately 10 neurons for each hidden layer suited best for the problem. Although 2-7 neuron gave e^{-1} and e^{-4} ,10 neurons gave e^{-7} and e^{-10} $RMSE$ in Levenberg-Marquardt. Therefore 10 neurons for each hidden layer structure were chosen. The more neuron given, the more powerful algorithm got, but after one point it starts to overfit data. In this example after 10 neurons, the model started to overfit.

The three possible learning algorithms were experimented such as Levenberg-Marquardt, Quasi newton, and Conjugate gradient descent. The Levenberg-Marquardt achieved the best results. Normally, Levenberg-Marquardt takes more time and steps to converge for large neural nets because of the computation of second-order derivatives and the hessian matrix. Yet, in this example, it is suited since the neural net and dataset are small.

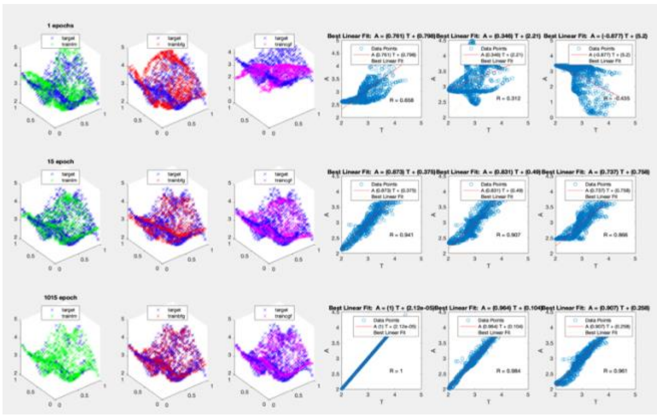
Quasi-Newton took around 60-150 iterations to converge which was far lower than Levenberg-Marquardt 500-1000 iteration since Quasi-Newton did not calculate second derivatives. However, the results on the validation set for Quasi-Newton were worse. One possible reason can be the approximation of the hessian matrix instead of the exact Hessian matrix.

With regards to Conjugate gradient descent, the obtained results are almost the same with the Quasi newton, but it took less step and time to convergence. Another advantage of the conjugate gradient descent is that it solves the storage of the hessian matrix problem by using a vector instead of matrixes. Therefore, it is well suited for large-scale neural nets.

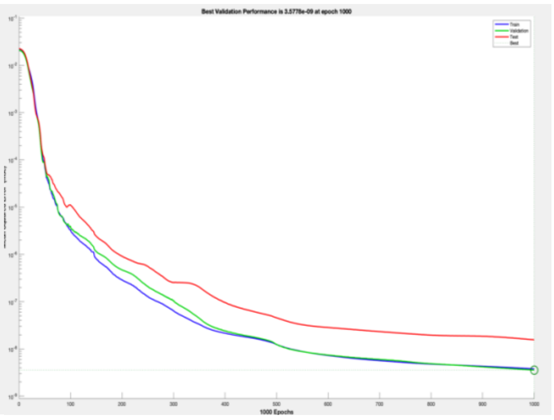
Algorithm	Activation function	Hidden Layer	Hidden Units	iteration	RMSE Validation
trainlm	Tanh	2	5	$60 < i < 120$	$e^{-3} < x < e^{-4}$
trainbfg	Tanh	2	5	$30 < i < 100$	$e^{-1} < x < e^{-2}$
traincgf	Tanh	2	5	$10 < i < 90$	$e^{-1} < x < e^{-2}$
trainlm	Tanh	10	10	$400 < i < 1000$	$e^{-7} < x < e^{-10}$
trainbfg	Tanh	10	10	$60 < i < 150$	$e^{-2} < x < e^{-4}$
traincgf	Tanh	10	10	$20 < i < 120$	$e^{-1} < x < e^{-2}$
trainlm	Poslin	10	10	$30 < i < 80$	$e^{-1} < x < e^{-3}$
trainbfg	Poslin	10	10	$20 < i < 70$	$e^{-1} < x < e^{-2}$
traincgf	Poslin	10	10	$10 < i < 60$	$e^{-1} < x < e^{-2}$

Table-1

During the training process, the results were varied since the random initialization of the weights and different local optima. According to *table 1* Levenberg-Marquardt algorithm 2 hidden layers, 10 hidden neurons with tanh activation function is the best combination for the problem. When the hidden layer and hidden unit number are increased model starts to overfitting. Among the learning algorithm Levenberg-Marquardt achieved a far better result.



Test result for three algorithms



Learning Curve Levenberg-Marquardt

According to test results, the model fitted on the data and gave satisfying results when the Levenberg-Marquardt algorithm was used. The reasons behind this were explained in the previous section. The obtained result from the training and validation set are close to each other. But the test result is a little bit worse. More data might be gathered to increase performance. Besides the training process is not stopped because of the validation checks. It is stopped since it reached to chosen iteration number 1000. By choosing a larger iteration number, performance might be improved.

Additionally, instead of using early stopping, *10-fold cross-validation* or *drop-out* generalization technique might be chosen. These can end up with better results. For instance, if only one validation set is used as in this example, your results will be very specific for your validation set. It can be too misleading. However, in *10-fold cross-validation,* you divide data into 10 segments, and you train 9 segments for each run. Your results will be more reliable since all observations are used for both training and validation, and each observation is used for validation only once. The downside of *cross-validation* is that it needs more computation power.

Bayesian Method

Algorithm	*Hidden Units*	*Training Error*	*Test Error*	*Noise*
trainlm	*50*	$1.49567\ \mathrm{e}^{-9}$	$1.1884\ \mathrm{e}^{-6}$	-
trainbfg	*50*	$1.1317\ \mathrm{e}^{-4}$	$1.8929\ \mathrm{e}^{-4}$	-
trainbr	*50*	$6.3136\ \mathrm{e}^{-8}$	$6.3136\ \mathrm{e}^{-8}$	-
trainlm	*50*	0.026867	0.0671	+
trainbfg	*50*	0.028037	0.0877	+
trainbr	*50*	0.031545	0.041045	+
trainlm	*350*	$2.2638\ e^{-8}$	0.027	-
trainbfg	*350*	$2.6043\ e^{-5}$	0.0019	-
trainbr	*350*	$3.9535\ e^{-8}$	$1.1467\ e^{-4}$	-

Table-1

On the one hand, the Bayesian method is a very powerful algorithm. It follows the data distribution instead of data points as gradient descent. Besides, it minimizes not the only error in the training data but also an error in the weights $\min M(w) = \beta E_D(w) + \alpha E_w(w)$. In other words, it decides how much regularization the model needs, based on the probabilistic assumption during the training stage. On the other hand, the Bayesian method depends heavily on some mathematical assumptions and the prior distribution of data. That leads to sometimes bad results.

According to $table - 2$ we can easily see that the Bayesian method gives better test results. Furthermore, test results are close to the training results. It means that it automatically adjusts the regularization of the weights. Hence, the overfitting is eliminated without the need for any validation set. When the results are compared with the other algorithm such as Levenberg-Marquardt, it is easily seen that it is more prone to overfitting since the training result and test result are far from each other. The validation data needs to be reserved or regularization needs to be applied for Levenberg-Marquardt and Quasi-Newton. Since the collected data is pretty valuable for neural nets, not needing a validation set is a very important feature of Bayesian especially when the number of data is limited.

However, the Bayesian model is not successful enough when the noise is applied to data. It gives worse results for noisy data since it depends on the likelihood procedure and distribution of data.

When the model is over parametrized by increasing the hidden unit to 350, Bayesian learning gives again the best result. Undoubtedly that the gap between training and test results is going up because of exaggeration of the parameter. Yet, the obtained result is far better than Quasi-Newton and Levenberg-Marquardt. The model is heavily overfitting with Quasi-Newton and Levenberg-Marquardt method in over parametrized circumstances. Despite the exaggerated parameters, it can adjust the weights which can be seen as the other advantage of Bayesian learning.

LAB-2

2-D network

Attractors: (-1,1), (-1, -1), (1, -1), (1,1), (1,0.31), (1,0.128), (1, -0.128)→ 10 iterations→4 unwanted

Attractors: (-1,1), (1, -1), (1,1), (1, -0.43), (1, -0.62), (-0.82,1), (1,0.31) → 15 iterations→4 unwanted

Attractors: (-1,1), (-1, -1), (1, -1), (1,1), (1, -0.62), (-0.82,1) → 25 iterations→3 unwanted

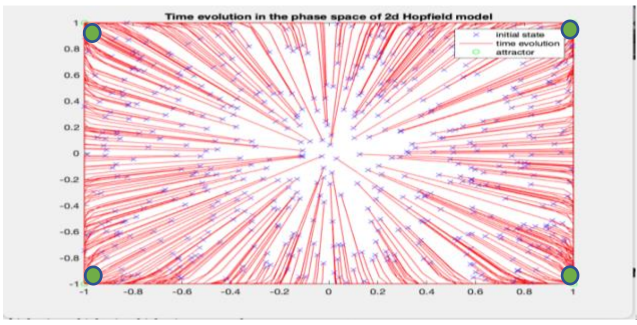
Attractors: (-1,1), (-1, -1), (1, -1), (1,1), (1, -0.73) → 42 iterations→2 unwanted

Attractors: (-1,1), (-1, -1), (1, -1), (1,1)→ 50 iterations→1 unwanted

Attractors: (-1,1), (-1, -1), (1, -1), (1,1)→ 80 iterations→1 unwanted

Attractors: (-1,1), (-1, -1), (1, -1), (1,1)→ 100 iterations→1 unwanted

Table-1



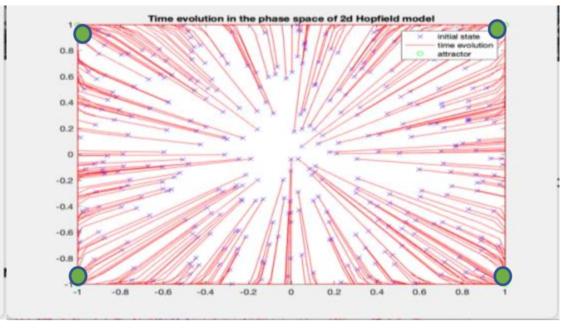


figure-1 50 multiple iteration *figure-2 350 multiple iteration*

1. As seen from $table-1$, more than one different unwanted attractor's points are observed before reaching the 50 iterations point. We observed these extra attractors since the number of iteration steps is insufficient. Hence, the network cannot converge to the optimum point. After 50 iterations, however, the only extra attractor's point is (-1,1). Firstly, for each stored pattern ξ^v automatically you have negation $-\xi^v$ and secondly, any linear combination of an odd number of stored patterns gives rise to the so-called mixture states. Thirdly, for large p, we get local minima that are not correlated to any linear combination of stored patterns when the memory capacity of the network is exceeded. If we start at a state close to any of these spurious attractors, then we will converge to them. However, they will
have a small basin of attraction.

2) It takes to 50 iterations to reach the attractor as seen from the $figure-1$ and $figure-2$ above.

3) The attractors are stable because there are always the same attractors and the given point ends up with one of the equilibrium points which means that the energy function is decreasing in each iteration and converge to one of the local minima. In other words, the dynamic model will show no chaotic behavior.

3-D network

Attractors:(1,1,1), (-1, -1,1), (1, -1, -1), (0.36,0.48,1), (0.65, -1,0,77),(0.57,-1,-0600),4 more→ 10 iteration

Attractors: (1,1,1), (-1, -1,1), (1, -1, -1), (1, -0.13,-0.23),(1,0.85,0.73),3 more→ 20 iteration

Attractors: (1,1,1), (-1,-1,1),(1,-1,-1),(-0.62,-1,0.5),(-0.26,-1,0.077) → 30 iteration

Attractors: (1,1,1), (-1,-1,1),(1,-1,-1),(-0.35,-1,0.22)→40 iteration

Attractors: (1,1,1), (-1,-1,1),(1,-1,-1),(-1,-0.91,1)→50 iteration

Attractors: (1,1,1), (-1,-1,1),(1,-1,-1)→100 iteration

Table-2

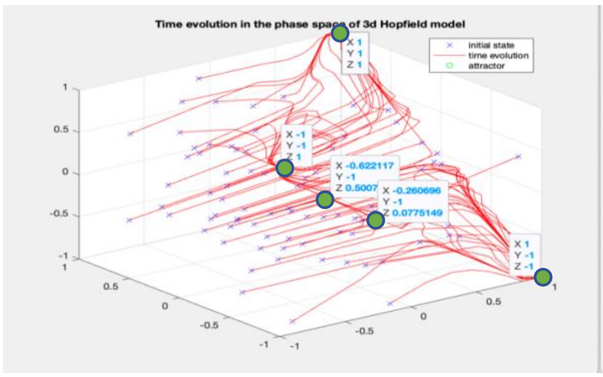


figure-3 30 multiple iteration

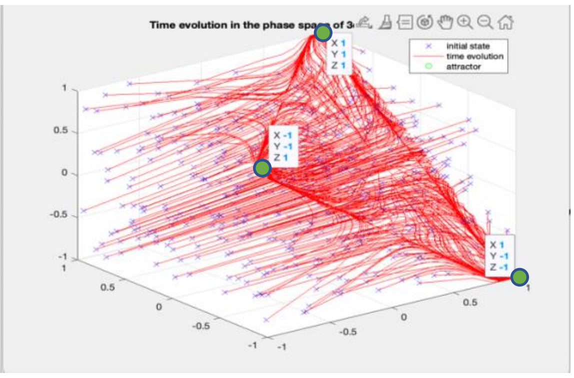


figure-4 100 multiple iteration

1. After 40 iteration the network reached a stable state. In 50 attempts for 50-60 iteration the only one extra unwanted attractor is observed. The second network is more stable than the first network since the second network has more neurons which decreases the p/N value. It leads to fewer errors in the network.

HOPDIGIT

Although the Hopfield network cannot always reconstruct the images correctly, it is doing a great job. Especially when the noise becomes bigger. Even if the iteration number is made so large, generally two or three numbers end up with the wrong reconstruction. The memory capacity decreases as the noise in the training patterns increases. The reason behind this is that $sgn\left(\frac{1}{N}\xi_i^0 \sum_j \xi_j^0 \xi_j^0 + \frac{1}{N}\sum \xi_i^u \sum_j \xi_j^u \xi_j^0\right) = sgn\left(\xi_i^0 + \frac{1}{N}\sum \xi_i^u \sum_j \xi_j^u \xi_j^0\right)$ if one of the memories is close to ξ_i^0 then $\xi_i^0 * \xi^l \approx N$ and $v_i(t+1) = sgn(\xi_i^0 + \xi^l) \neq \xi_i^0$. Hence, your result is the combination of two, not a real attractor.

NEURAL NETWORK

Algorithm	Lag Value	Number of Neurons	Best Rmse Result	Mean Rmse	Rmse<30
trainlm	50	50	12.2308	60.3513	12 result
traincgp	50	50	33.780	187.9378	0 result
trainscgp	35	25	23.8823	91.543	1 result
trainlm	25	20	29.6908	148.10	1 result
trainlm	15	15	49.5559	155.70	0 result
trainlm	10	10	48.4738	128.5779	0 result

Table-3 With Validation set 50 try

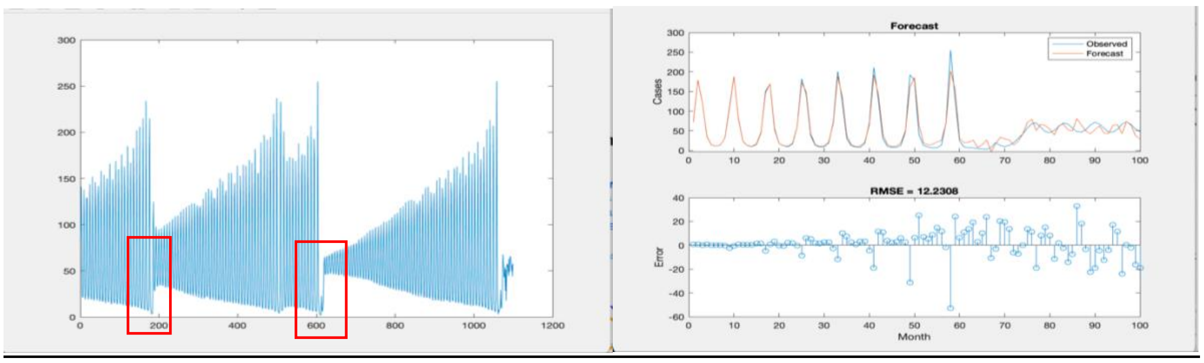


figure-5 Best result one hidden layer-trainlm lag:50 neuron:50

Firstly, the training data is divided into train and validation sets. The chosen validation set is shown as red color on the $figure - 5$. This part is chosen deliberately in order to obtain a better test result. Because there is a similar disruption (chaotic behavior) at the end of the test data.

Besides, the lag value is chosen between 15 and 50, and four different back-prop algorithms are used to find the best result. When the lag value has increased the dimension of the input is also increased. Hence, the dimension of the interconnection matrixes is getting bigger. This can cause the derivatives to disappear or explode which is known as the vanishing gradient-exploding gradient. To overcome this problem weight regularization technique is used instead of early stopping in the second test case ($table - 4$). Moreover, the number of neurons in the hidden layer is increased.

The obtained results illustrate that when the lag value is chosen higher, the test results are improved. Because you feed the network with more previous data (historical data). Thus, the network can learn better. However, aforementioned it can cause overfitting and immense interconnection weights and neuron's number (vanishing gradient-exploding gradient).

Another issue related to one hidden layer network structure is that it cannot estimate disruption at the end of the test data. It is well known that neural network models can predict more complex structures with more than one hidden layer. To prove it extra test cases with two hidden layers were applied. Thus, the model predicted better and smoother in the test set as seen below images (*traincgp two hidden layers*).

Algorithm	Lag Value	Number of Units	Best Rmse Result	Mean Rmse	Regularization	Rmse<30
traincgp	50	50	9.0178	95.8057	0.001	10
trainscg	50	50	10.634	91.450	0.001	10
trainlm	50	25-20	20.9057	107.7139	0.001	2

Table-4

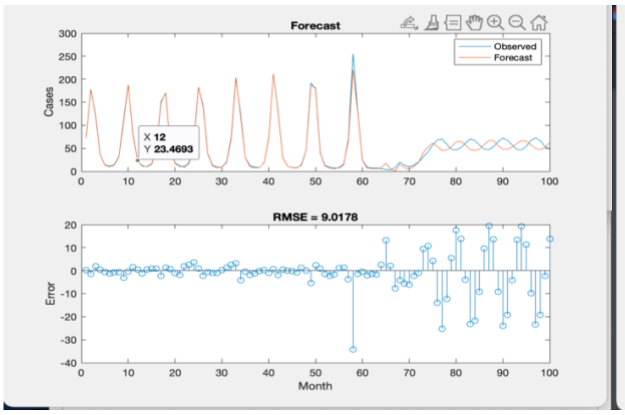


figure-6 traincgp-two hidden layer

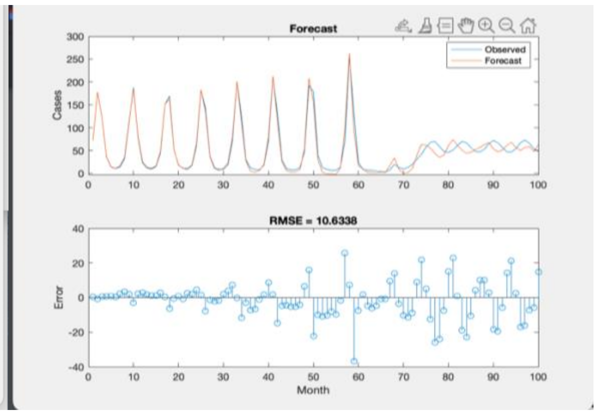


figure-7 trainscg with one hidden layer

As seen $table - 4$ by using regularization, better test results are obtained and, also the model predicted future values better than the early stopping. Even if the mean of the $RMSE$ does not show so difference, there are a lot of $RMSE$ value which is smaller than 30 especially in the traincgp algorithm. Considering lag value, 50 leads to a generally better test result. *The best-obtained test result is traincgp,0.001 regularization, two hidden layers 25-20, lag value=25.*

LONG SHORT-TERM MEMORY NETWORK

Lag	Epoch	Hidden units	Mean Rmse	Nr of tries	Matched
5	650	20	72.567	5	Bad
5	250	20	65.345	5	Bad
5	500	50	50.45	5	2 good
10	650	50	14,78	5	3 very good 2 good
10	650	100	9,4	5	4 very good 1 good
15	650	50	10.765	5	4 very good 1 good
15	650	100	10.4	5	4 very good 1 good
25	650	50	18,4	5	3 very good 2 good
25	650	100	12,5	5	5 good
50	650	50	12,25	5	5 good
50	650	100	10.3	5	4 very good 1 good

Table-5

1. First of all, Adam optimizer was used because of the adaptive learning rate feature. Besides, it can handle better the complex training dynamics of recurrent networks than the plain gradient descent. Like other neural networks, overfitting is also a problem for LSTM's. To avoid overfitting drop-out regularization method has been applied. Max epoch was considered 250 in the beginning. However higher epoch gave better results.

The number of hidden units was chosen around 50-100. Because the higher number of units, the better fit for non-linear data. As seen from the figure when the model has a bigger number of neurons it can estimate more precisely the chaotic side of the test set.

Furthermore, the LSTM contains cells to decide based on the strength and importance of the information. When the number of units is increased these number of cells also increased. Hence, the network can make a better decision.

By taking into consideration RNN, the lag value was chosen higher to feed the network more historical data. However, it is realized that after $lag = 5$, the test results were not improved and more or less stay the same in the LSTM. The reason can be that the LSTM can update and forget the data according to its importance and select the most appropriate ones. Therefore, there is no need for a lot of historical data.

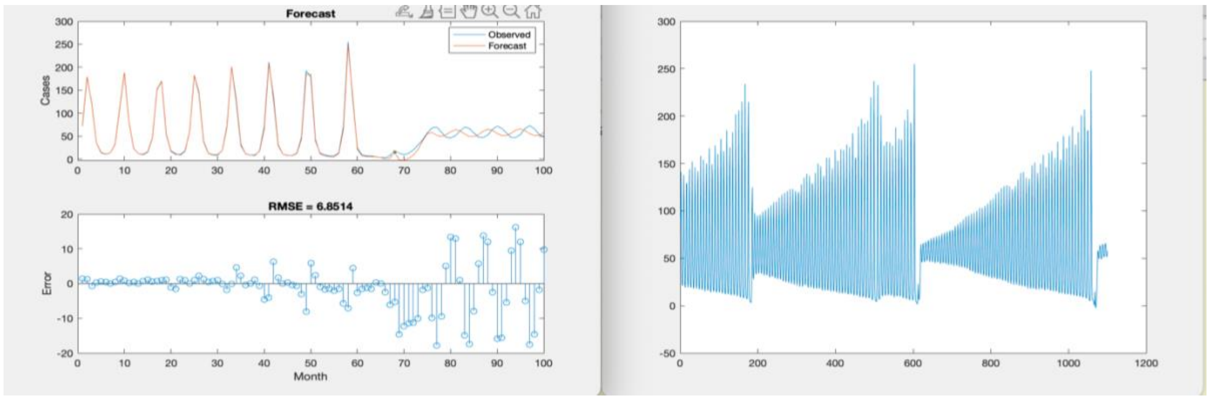


Figure-8 LSTM 100 hidden neurons $lag = 10$

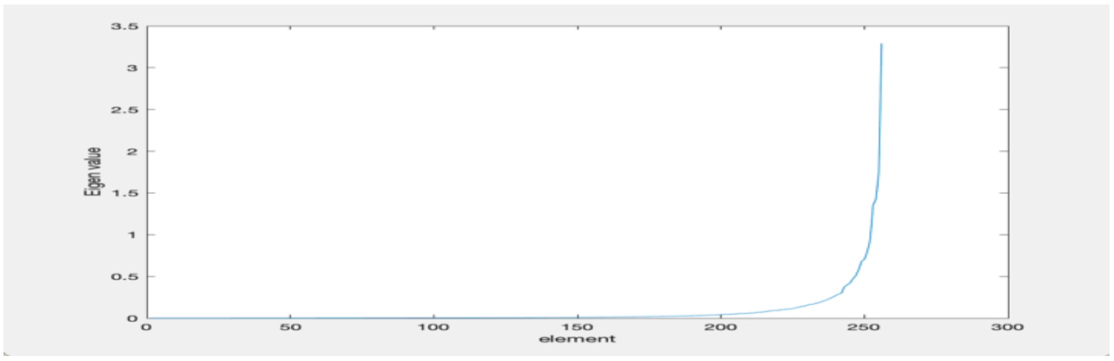
When the results are compared, the LSTM gives much better mean $RMSE$ results and also it reaches a better $RMSE$ value. Besides, it can also handle the chaotic behavior of between 70 and 100 timestamps. In other words, it gives a smoother wave shape which is much harder to achieve with one-layer RNN. In the RNN you have to initialize weights again and again to obtain the best results. It is well known that RNN suffers from vanishing-exploding gradients problem. These problems lead to an unstable network as it is seen from table 3, the networks give incompatible results after each initialization. To some extent, you can refrain from the problem by using regularization. However, test results prove that even if regularization is used, the improvement is insufficient. The LSTM is much more efficient to overcome the problem. Because the gate mechanism provides more efficient learning through gate structure and, as seen from the $table - 5$, by using less amount of input(lag) the LSTM ends up with not only better results but also a much more stable network.

In conclusion, I prefer to use the LSTM structure because of the gate structure leading to powerful learning. Besides, it solves the vanishing-exploding gradient problem. In other aspects, it fitted data and gave more consistent results.

LAB-3

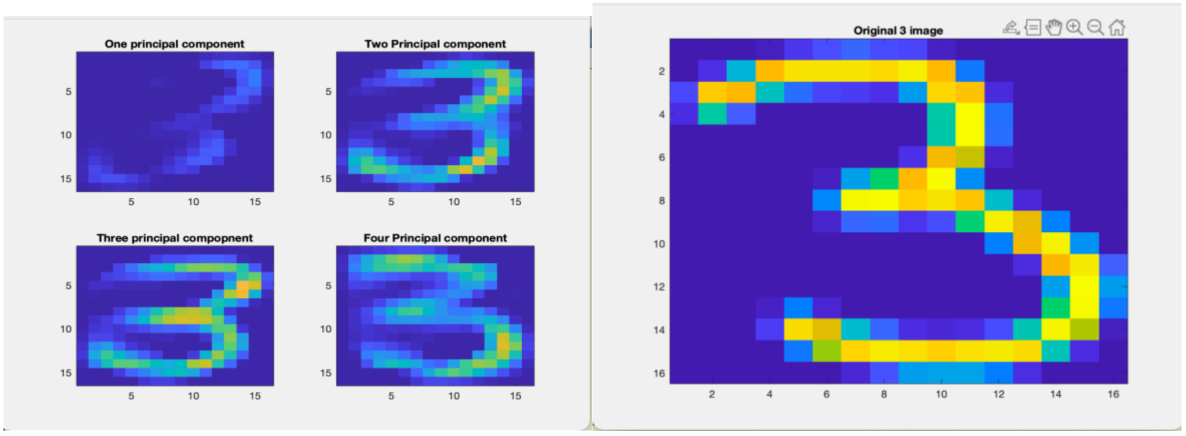
PCA

1. In order to calculate the mean of three, *mean(threes,2)* function was used. Because the function calculates the row's mean which is $500\ x\ 1$ vector.



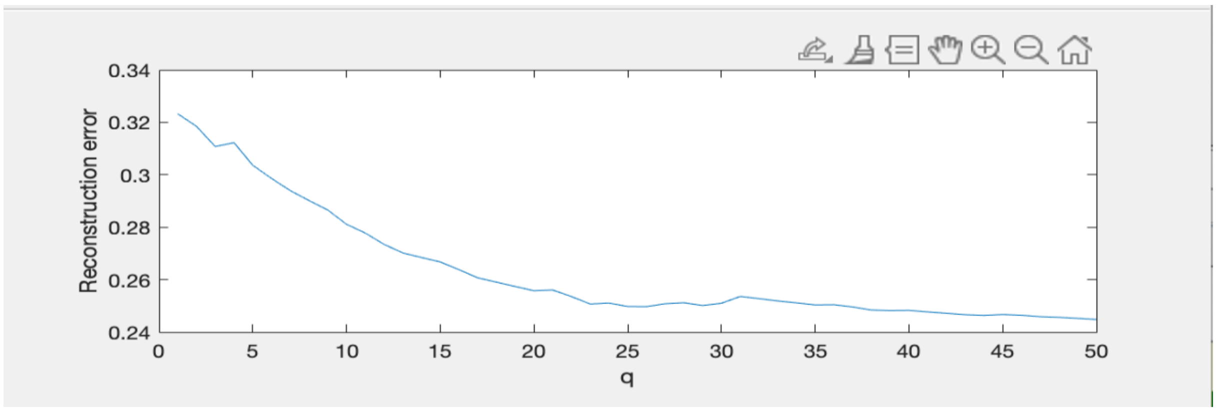
Plot-1

2. As we can see the $Plot-1$ eigenvalues start to increase after 200 points, which means that very few numbers of values affect the performance of the images. For this reason, we can display *threes* that has good image quality, by using a small amount of eigenvalue that have the biggest values.



Plot-2,3

3. As seen $Plot-3$ the three's image can be distinguished easily just using 2 principal components since we used not only the two most valuable elements at the compression stage but also the eigenvalue of these two elements are the biggest ones than the rest.

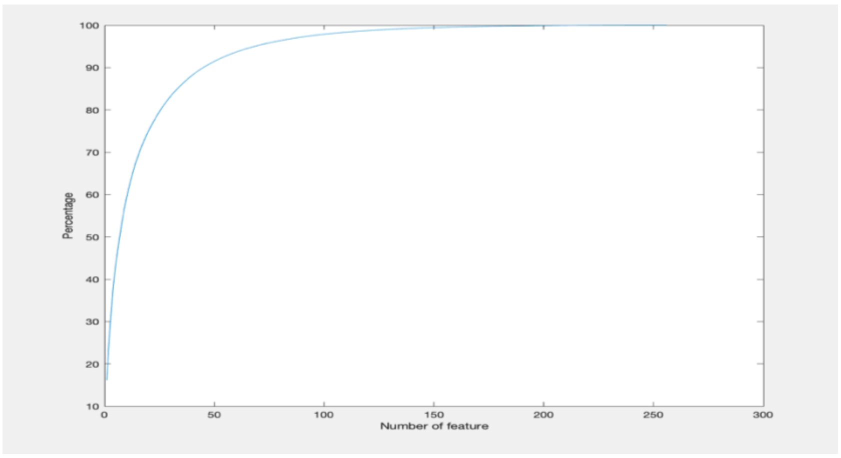


Plot-4

4. *Plot-4* shows that after 25 principal components the error is not changing a lot and it remains almost the same. Aforementioned, after some point the difference between the eigenvalue almost zero and adding a new component is not affecting the image's quality.

5. $error_{256} = 0.2375$. Even if you use 256 principal components, the error is not so different from q=50($error_{50} - error_{256} = 0.0073$). Because the most valuable components were used in the earlier stage and as seen from $Plot-1$, 200 eigenvalues from 256 are almost zero.

6. Figure shows the relation between the number of used principal components and the percentage of the quality of the image. For instance, you can provide more than %90 of the image quality by using only 50 principal components.



Plot-5

AUTO-ENCODERS

Hidden layer 1	Hidden Layer 2	Hidden-Layer 3	Hidden-Layer 4	Hidden-Layer 5	Max-Epoch	Soft-Max	Result Without Tuning	Result With Tuning	NN 1 Layer Result	NN 2 Layer Result
100	50	-	-	-	1)400 2)100	50	79.1	99.8	95.8	96.6
100	50	-	-	-	1)600 2)300	50	94.9	98.4	95.8	96.6
320	150	-	-	-	1)600 2)300	150	98.9	99.3	96.7	97.8
320	150	50	-	-	1)600 2)300 3)400	50	90.0	99.1	94.2	96.9
320	150	50	-	-	1)600 2)300 3)150	50	72.7	99.8	97.8	95.7
250	150	100	50		1)600 2)300 3)300 4)300	50	61.8	99.5	96.6	97.5
350	200	150	100	50	1)400 2)300 3)250 4)300	50	81.4	68.34		

Table-1

We can make the auto-encoders powerful by increasing nodes per layer and most importantly the code size. Increasing these hyperparameters will let the autoencoder learn more complex coding. But we should be careful to not make it too powerful. Otherwise, the autoencoder will simply learn to copy its inputs to the output, without learning any meaningful representation. It will just mimic the identity function. The autoencoder will reconstruct the training data perfectly, but it will be overfitting without being able to generalize to new instances, which is not what we want. This is why we prefer a “sandwich” architecture, and deliberately keep the code size small

- By increasing just, the epochs, the pre-train stage catches almost the normal neural nets (see second row 94.9)
- When the number of hidden units in the soft-max layer is increased, the pre-train performance is also going up (Third row 98.9). However, the compression size or dimensionality reduction is going down.
- The number of epochs at the last hidden layer affects the learning. When the epoch number is decreased from 400 to 150 the result without tuning is going down to 72.7. As seen above table the more epoch brings the general performance up.
- According to results when the layer size is increased or it is made deeper, the pre-trained performance is getting worse. The deeper architecture is causing a higher error rate.

The pre-train stage provides a better starting point if we compared to normal neural nets since neural nets are started the weights randomly. By using unsupervised learning, the pre-train stage learns how the interconnection weights can be started more efficiently than the multi-layer perceptron. Additionally, by applying fine-tuning stage, you can increase the performance of the network further, since the back-prop algorithm increases the model's learning performance. Hence, you end up with a more efficient model. It can be seen above table, almost all of the result with tuning is higher than the normal neural nets performance even if pre-stages performance is worse.

CNN

CNNex.m

1. The weight's dimension is $11x11x3x96$. It indicates that there are 96 different filters, and each filter has $11x11x3 = 363$ weights. A filter refers to the specific feature of the dataset. Let assume, our data contains the people's faces. We can say that one of the filters can be the edges of the face. Then the weights are each element of this filter. The CNN extracts the meaningful features of the dataset by adjusting the weights on its own. Besides, you can use the pre-extracted feature, when these filters are applied across the whole input, these features can be detected anywhere in the image.

2. For overlapping regions where the pool size is bigger than stride size, the output of a pooling layer is *(Input Size – Pool Size + 2 * Padding) / Stride + 1. Poolsize =[3x3], Stride=[2x2], Inputsize=[11x11x96x3].*
$\frac{([11x11]-[3x3]+2*0)}{[2x2]} + 1 = [5x5]$ $is\ for\ only\ one\ filter. There\ are$ 96 $filter\ then\ result$ $[\mathbf{5x5x96}]$.
Because of the max-pooling layer dimension of the input at the start of layer 6 is decreased. A max-pooling layer performs downsampling by dividing the input into rectangular pooling regions and computing the maximum of each region as another way of avoiding overfitting.

3. The dimension of the input layer before the fully connected layers *($convnet.Layers(17)$ $Name$: $'fc6'$ Hyperparameters→ **InputSize: 9216** OutputSize: 4096)* is **9216**. The input dimension of the images was **[227x227x3] =154587**. As it is noticed, the dimension is decreased a lot at the start of the fully connected layer. That is the one major advantages of CNN since you do not deal with a large amount of interconnection matrix as we compared with the normal neural network. It

is well known that the images have large input dimensions. Besides, when it is combined with huge interconnection weights, the fully connected networks demand much more memory and computational power.

CNNDigits.m

N.	Conv Layer1	Relu Layer-1	maxPoolLayer-1	Conv Layer-2	Relu Layer-2	maxPool Layer-2	fullyConnected Layer-1	fullyConnected Layer-2	Test Acc	Time
1	✓	✓	✓	✓	✓	×	✓	×	0.82	25.25
2	✓	✓	✓	✓	✓	×	✓	✓	0.71	26.95
3	✓	✓	✓	✓	✓	✓	✓	×	0.48	27.18
4	✓	✓	×	✓	✓	×	✓	×	0.96	41.69
5	×	✓	✓	×	✓	×	✓	×	0.52	7.43
6	×	✓	×	×	✓	×	✓	×	0.63	7.46
7	✓	×	✓	✓	×	×	✓	×	0.91	22.68
8	✓	✓	Pool 2x2 Stride 1x1	✓	✓	×	✓	×	0.96	51.72

Table-2

When the different number of layers is chosen training time and test accuracy is varying in some conditions a lot. The results illustrate that the max pool layer is affecting the training time because of the down-sampling operation. Since the dimension of the weight is decreasing, the training time is also going down. Besides without the max-pooling layer the test accuracy increase. The reason is that no data is eliminated by the max-pool layer in order to speed up the learning process. For instance, in row 4 without max-pool layer train time increase two times, and accuracy increase to 0.96.

Considering the convLayer, as seen in row-5 without any convLayer, the training accuracy and training time is going down. As the convolution layer extracts the meaningful features to be used in classification, it enhances the test accuracy. With regards to training time, since the convolution layer demands complex operations, it is time-consuming. Hence, the training time is decreasing without it.

Other meaningful result is that by adjusting *stride* to *1x1* the test accuracy improves %14 percent compared to the starting point as more data points are included in the pooling stage.

As a result, the max-pooling and convolution layers are very important for CNN. Although in this case the max-pooling layer is seen as a performance blocking element, in the real application it prevents overfitting and wasting immense time for training CNN.

LAB-4

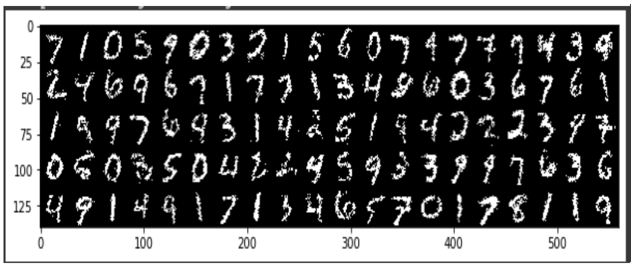
1. RESTRICTED BOLTZMANN MACHINES

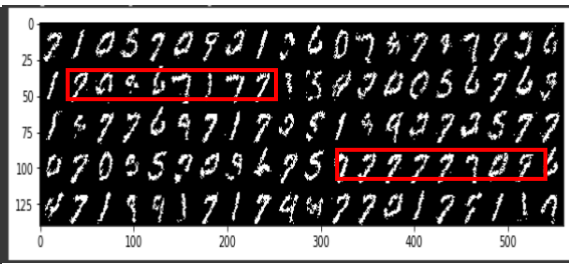
N.	Component	Epoch	Learning rate	Gibbs Step	Reconstructing Quality	Likelihood
1	10	10	0.01	10	Low	-198 to -152
2	20	100	0.01	10	Slightly better	-198 to -120
3	100	50	0.01	10	Improved a lot	-128 to -74
4	100	50	0.01	100	Worse than Gibbs=10	-128 to -74
5	200	20	0.01	10	Good quality	-98 to -65.91
6	200	20	0.01	150	Worse than Gibbs=10	-98 to -65.91
7	100	10	0.1	10	Worser than 0.01	-102 to -95.53
8	250	10	0.01	1	Best quality	-95 to -68.66
9	250	10	0.06	1	Good quality	-87 to -79

Table-1

The number of components affects the learning process undeniably. Firstly, as seen from the table and the reconstructed images, by increasing the number of hidden neurons, the quality of the images is brought up. In other words, the model can learn more efficiently when the number of hidden units is raised due to fact that the larger the number of hidden units, the more features can be extracted.

Considering the epoch, the larger epoch number improves the learning of the RBM. For instance, as the number of epochs is increased from 10 to 100, the likelihood converges to 120 and the image quality is getting slightly better. (*See row 2*)

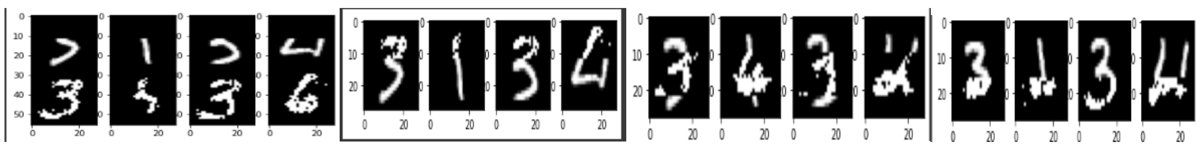

component=100 gibbs=1


component=100 gibbs=50

Concerning Gibb's steps in training, it affects the quality of the reconstructed images as well. When the number of Gibbs steps increase to 50 the reconstructed image quality is going down for digits that include more complex drawing. If it is decreased to 1 improvement is observed in the digit quality. In other words, performing just one sample of *h* and immediately resampling $\tilde{x}$ provides a good procedure for extracting meaningful features.

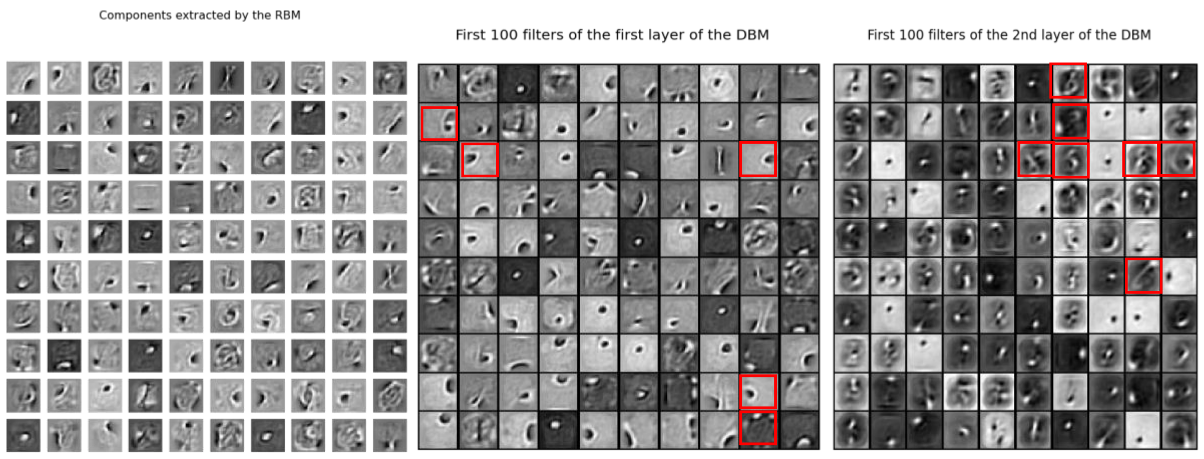
The results indicate that the choice of the λ is crucial If chosen correctly, the divergence problem was solved. But if the hyperparameter λ was too large, learning stagnated on a low log-likelihood level and thus the RBMs did not model the target distribution accurately. And if the parameter was too small, the weight decay term could not prevent divergence.

Concerning training time, more units require more training time. For instance, 250 components take approximately 42 seconds for each iteration since the complexity of the model is enhanced.


Components=200 Gibbs=150 $\lambda = 0.01$

The missing part of the images can be reconstructed to some extent as seen above. When the smaller hidden units are used the performance is getting worse. The reason behind this was explained previously. The Gibbs sampling has an important role in reconstructing missing parts. It is observed that when the Gibbs step is chosen bigger, the model can reconstruct the missing part of the images better. In the case of removing the rows from the middle of the image, the reconstructed images are not well-conditioned. Additionally, when the removed rows number is made larger, the images are not rebuilt as expected.

2. DEEP BOLTZMANN MACHINES


Components extracted by the RBM
First 100 filters of the first layer of the DBM
First 100 filters of the 2nd layer of the DBM

RBM *DBM First Layer* *DBM Second Layer*

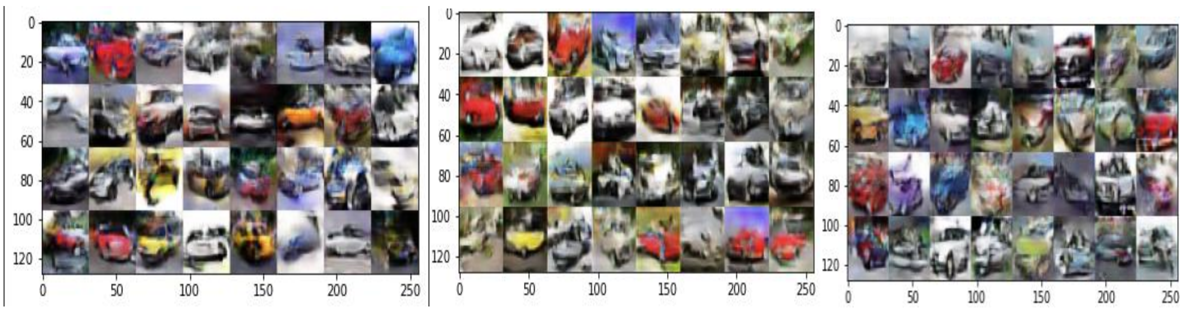
Each square on the images of filters corresponds to one hidden unit. Besides the intensity of the pixel indicates weight's value. For instance, the gray color corresponds to weight equals zero whereas the white pixel corresponds to the positive weight, and black corresponds to negative weights. In other words, if the square includes more white pixels the probability of that hidden units goes up vice versa for the black pixels. It is easily realized that the model is learning the pen strokes of digits.

First of all, at first glance, one can notice that RBM contains more greyscale pixels than DBM. The squares in the RBM include also more mixed shapes or features than DBM. In other words, while in DBM you see mostly the edges of the digits in the first layer, in the RBM you can see also meaningless features than the edges in one square. Additionally, some digits are distinguishable even in the first layer in DBM. (*Indicated with red squares 6-5-5-5-9*).

Considering the difference between the first layer and the second layer, the second layer detects more non-linear edges than the first layer, namely more complex features and the squares have images that seem like 3-D.

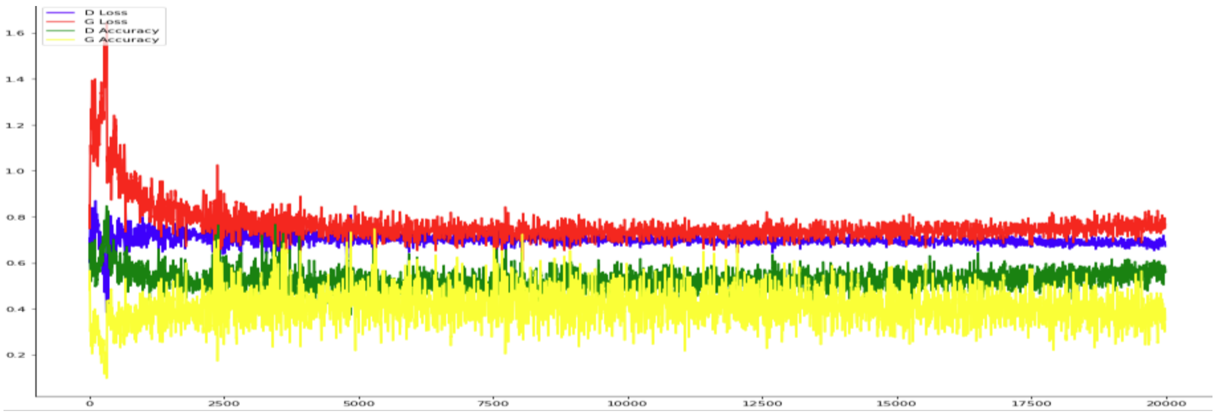
The created digits are noisy in RBM. However, the created images are cleaner in Deep Boltzmann machines. The benefit here adding a new layer provides higher quality images.

3. GENERATIVE ADVERSARIAL NEURAL NETWORKS



Generated Photos GAN

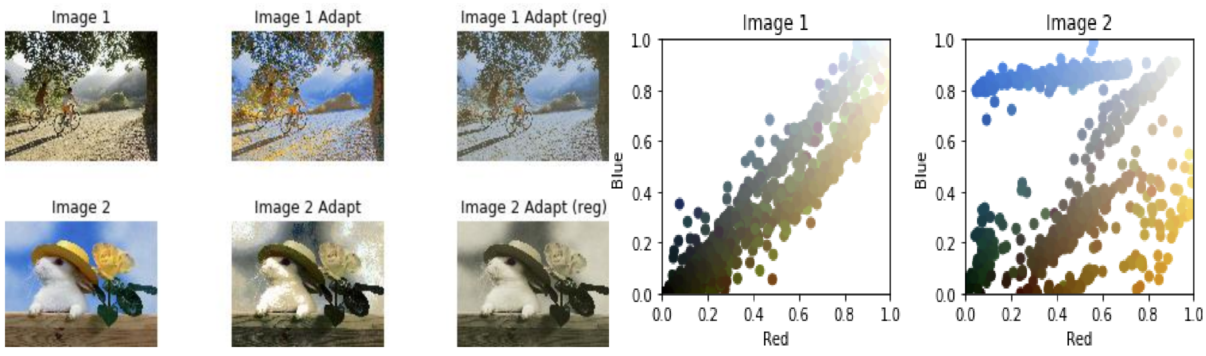
GANs are composed of two networks, Generator and Discriminator, there is a competition between them. Both the generator model and the discriminator model are trained simultaneously in a zero-game context. This means that improvements to one model come at the expense of the other model. Besides, each one of them has its loss function$E_D = [\log(1 - D(x)]$, $E_G = [\log(1 - D(G(z)))]$. The Generator (G) loss $= [log\ (1 - D(G(z)))]$, can lead to the GAN instability, which can be the cause of the gradient vanishing problem when the Discriminator (D) can easily distinguish between real and fake samples. In GANs architecture, the D tries to minimize,$G^* = arg \min_G \max_D v(G, D)$, while the G tries to maximize cross-entropy. When D confidence is high and starts to reject the samples that are produced by G, leads G's gradient vanishes. The existence of local equilibria in the non-convex game might be a reason. In other words, stability can be seen when $\max_D v(G, D)$ is not convex.



Training Curve GAN

Curves illustrate that the model is stable. Because there is not any Nash equilibrium point in the graph. Nash equilibrium happens when one player will not change his action regardless of what the opponent may do. This is the only state where the action of your opponent does not matter. It is the only state that any opponents' actions will not change the game outcome. Besides the loss of the two elements is decreasing during the training and also generator accuracy is increasing, and discriminator accuracy is decreasing. It means that model is generating more realistic fake images during the training.

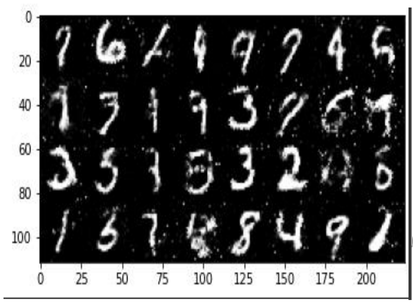
a. OPTIMAL TRANSPORT

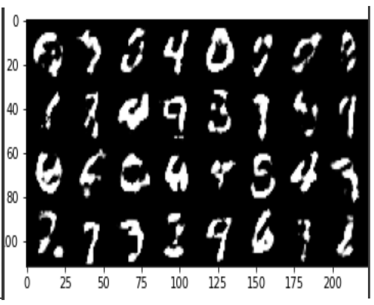


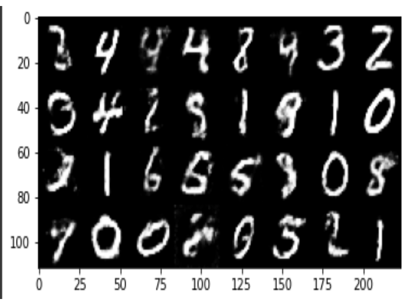
Color Histogram Transportation

The main idea is how costly it is to shift a mass from one observation to another. The RGB color points that are on the graph, can be seen as a mass. Then how these points are optimally transferred to the target based on the optimal transport strategy. As seen from the images, the pixels are not swapped directly without computing the cost. For instance, even if there is not any yellow point on the bottom of image 2, they can be observed on the image 1 adapt version.

b. WASSERSTEIN GAN


GAN


WGAN Weight Clipping


WGAN Penalty

GAN produces not only noisy images but also, we observe the same digits 1 eight times than others which is known as mode collapse. Additionally, the generated image’s quality is not expected level. Hence, we conclude that it has a stability problem as known discussed previous section. To cope with this problem Wasserstein GAN is found out. In the GAN the discriminator plays a classifier role which causes the stability problem since it is bounded with sigmoid function, whereas in WGAN discriminator has the role of critic. In other words, it grades the generated images based on their realness and fakeness. We do not observe in the formula any probability, we only see real numbers which removes the bound of the discriminator. For this reason, as it is noticed from the code, the Linear activation function is used at the output layer instead of the sigmoid.

$$W(D,G) = E_{x\sim pdata}[D(x)] - E_{z\sim p_z}[D(G(z))]$$

$$W(D,G) = -W^{(D)}(D,G)$$

The main expectation is that training the generator should seek a minimization of the distance between the distribution of the data observed in the training dataset and the distribution observed in generated examples based on the optimal transport and Wasserstein theory. So, the discriminator wants to maximize the distance between its grades on the reals versus on the fakes. In short, it's trying to push away these two distributions to be as far apart as possible. Meanwhile, the generator wants to minimize this difference, because it wants the discriminator to think that its fake images are as close as possible to the reals.

As a result, because WGAN is not bounded, the critic is allowed to improve without degrading its feedback back to the generator since it doesn't have a vanishing gradient problem, and this will prevent mode collapse because the generator will always get useful feedback back.

The WGAN with gradient penalizing gives the best results. Because sometimes first WGAN can still generate only low-quality samples or fail to converge since weight clipping enforces a Lipschitz constraint on the critic, which can lead to undesired behavior.