Report

Machine Learning Practical

Experiments with Deep Neural Networks

Author: Georgios Diakidis
University Number: s1673624

Contents

General Information

In this report I present the experiments I did for the topics I chose, concerning the hand written digit classification. In all my experiments I use a batch size of 50 and train for a total of 50 epochs, except for certain situations where I mention so. The 3 topics I chose where "reaction to different non-linear layers", "topics in Autoencoders" and "Batch Normalization". I also implemented a convolutional layer but due to the limited time I could not research it any further so instead I chose to combine it with the previous topics and see how it reacts to different situations.
Notation: -Affine-Relu, means a model with 2 layers, an Affine layer and a rectified linear layer. - 400-200 Stacked autoencoder, means the combination of 2 autoencoders, the forst one compresses the input at 400 points, and the second compresses the 400 input at 200 points.

.

1 Topic 1: How does the choice of the non-linear transformation used between affine layers

In this section, through an experimental process, we examined how the performance in training and validation sets get affected by the non-linear transformation used between affine layers in a model.

The initial experiments we did were with Rectified Linear models.

As we can see from the results, the rectified linear layer had the best results, followed by the other two. The sigmoid had the worst results both in the 3-layer models and the 5-layer models.

1.1 Performance of 3-layer models

Our models in this section are of the form Affine-NonLinearLayer-Affine.

Model	Error(train)	Error(val)	Accuracy(train)	Accuracy(val)
3-layer-Sigmoid	1.12e-01	1.24e-01	9.69e-01	9.65e-01
3-layer-Tanh	2.77e-04	1.14e-01	1.00e+00	9.77e-01
3-layer-Relu	3.42e-02	8.11e-02	9.93e-01	9.76e-01

Table 1: Error and Accuracy of our models

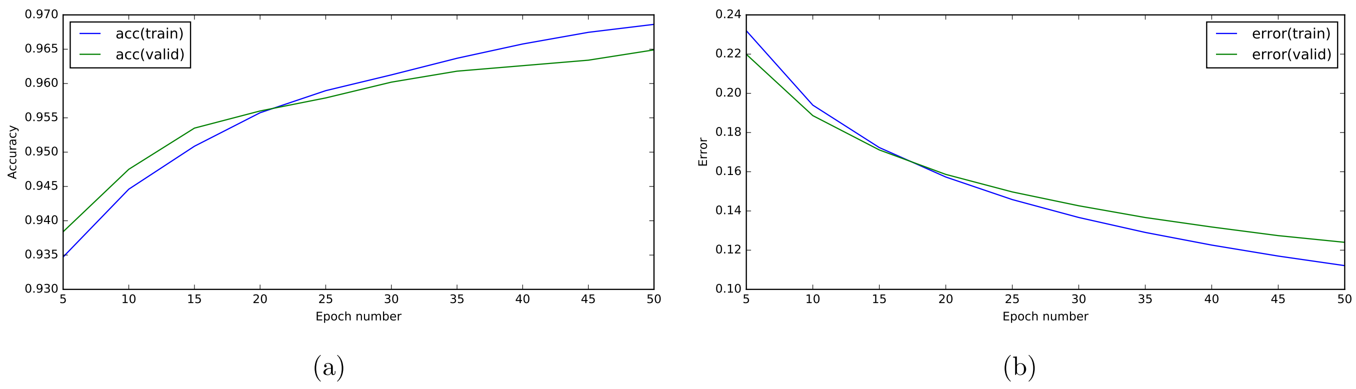
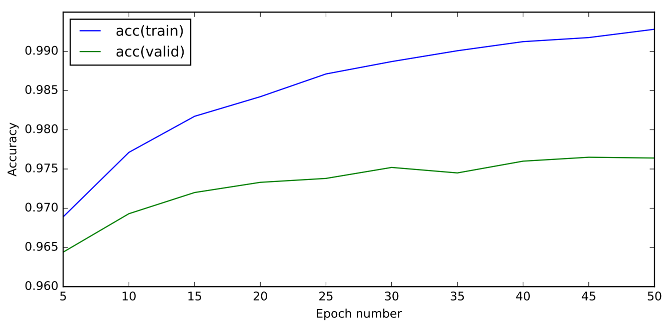
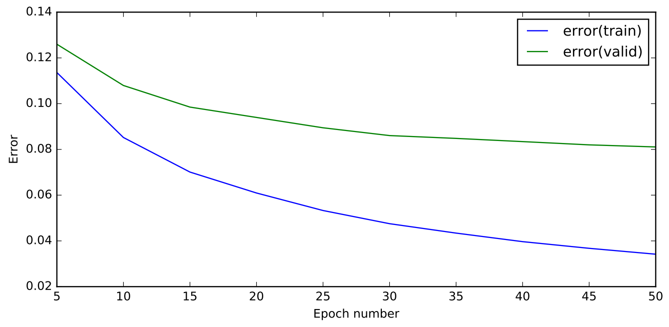


Figure 1: Accuracy and Error of a 3-layer Sigmoid NN

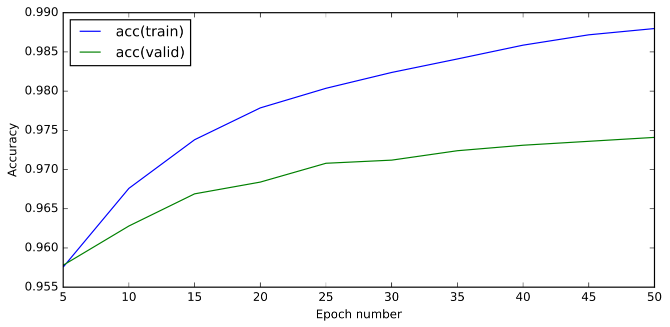


(a)

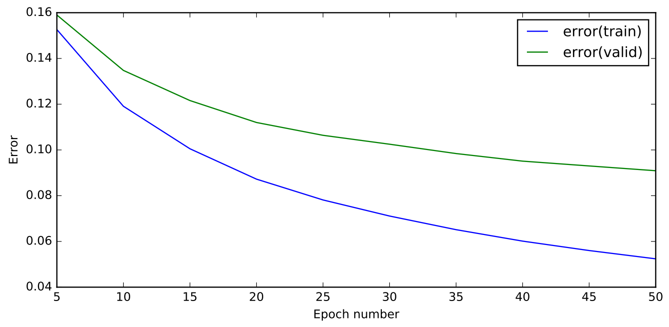


(b)

Figure 2: Accuracy and Error of a 3-layer Relu NN



(a)



(b)

Figure 3: Accuracy and Error of a 3-layer Tanh NN

1.2 Performance of 5-layer models

We experimented with 5-layer models, they consisted of 3 affine layers and 2 non-linear layers (Relu,Sigmoid,Tanh). We got the following results for each one.

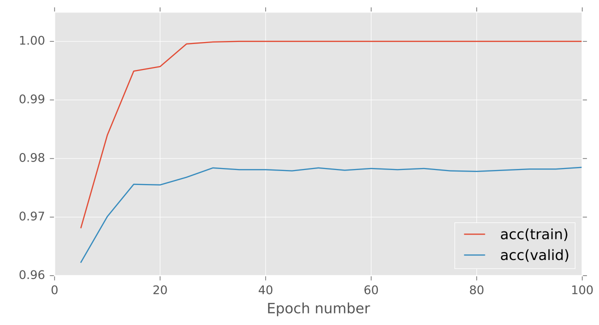
Model	**Error(train)**	**Error(val)**	**Accuracy(train)**	**Accuracy(val)**
5-layer-Sigmoid	9.03e-03	9.32e-02	9.99e-01	9.75e-01
5-layer-Tanh	2.77e-04	1.14e-01	1.00e+00	9.77e-01
5-layer-Relu	1.39e-04	1.31e-01	1.00e+00	9.79e-01

Table 2: Error and Accuracy of our models

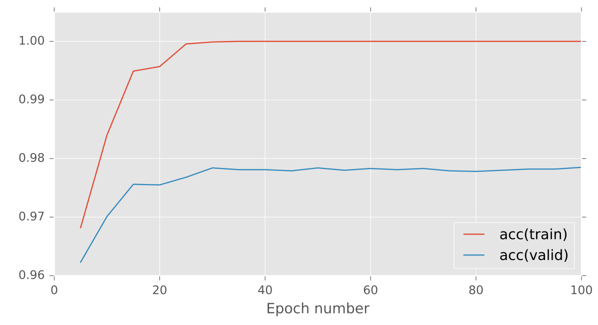
We can see that the Neural network with the 2 Rectified Linear layers has better performance than the other two. We can safely assume thet for the task of image classification, a rectified linear layer shows better performance and it is to be preferred.

2 Topic 2: Preprocessing with stacked Autoencoders and Deep Autoencoders

In this section, we will use Autoencoders as a preprocessing method to discover how the new features generated by them will affect our final result. We use a simple non-linear model for comparison.It's architecture is: Affine Layer-ReluLayer-AffineLayer-SigmoidLayer . It uses the Adagrad adaptive learning

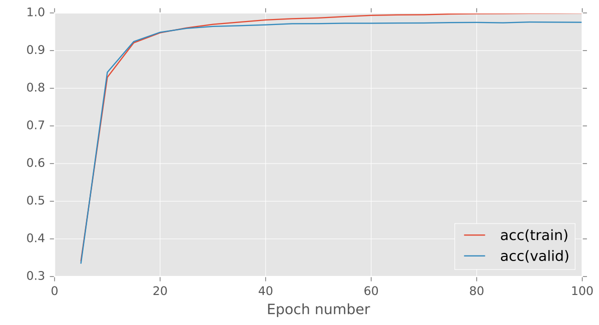


(a)

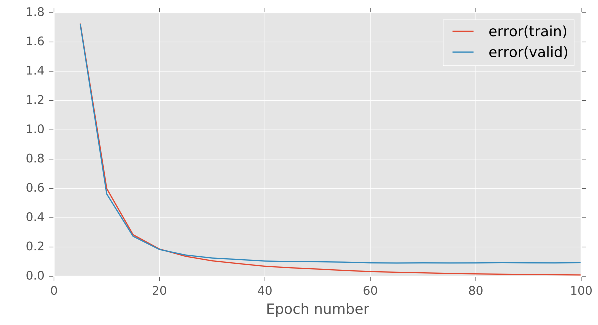


(b)

Figure 4: Error and Accuracy of a simple non-linear multi-layered model with tanh layers

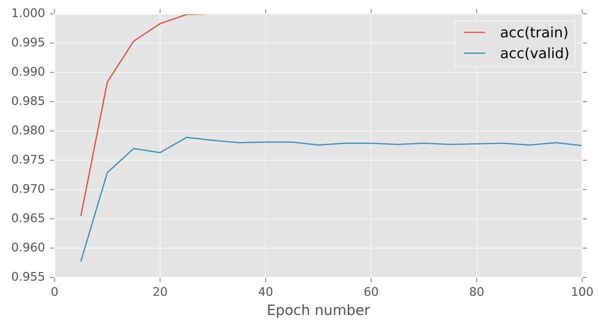


(a)

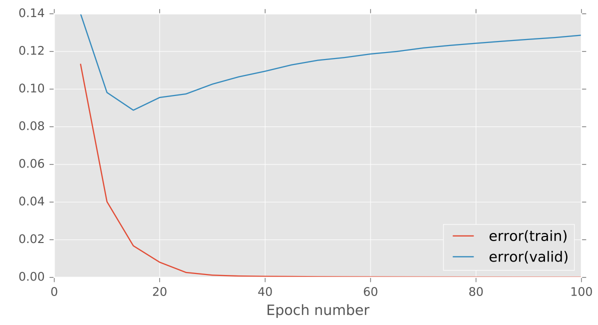


(b)

Figure 5: Error and Accuracy of a non-linear multi-layered model with Sigmoid layers



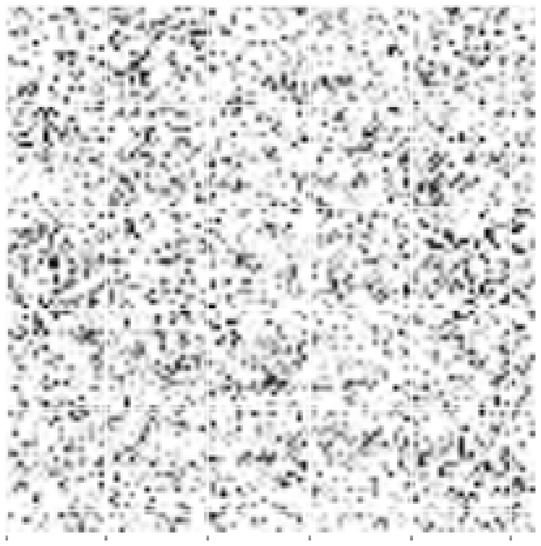
(a)



(b)

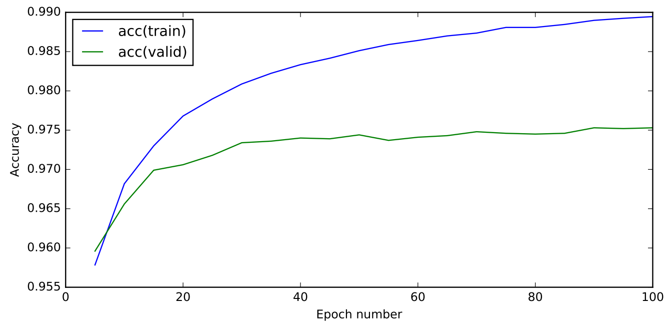
Figure 6: Error and Accuracy of a non-linear multi-layered model with Rectified linear layers

rule with learning rate 0.01. After being trained for 50 epochs it reached a maximum validation accuracy of 9.79% and each epoch took approximately 10 seconds to run.

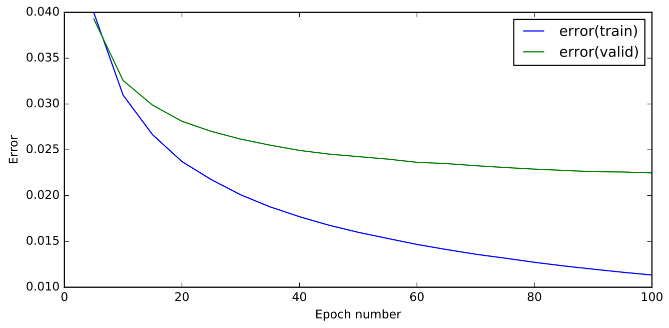


(a)

Figure 7: The output of the middle layer of an autoencoder shown as an 15x15 images



(a)



(b)

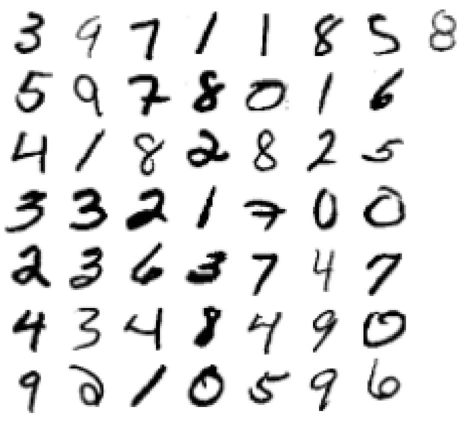
Figure 8: Error and Accuracy of a simple non-linear multi-layered model

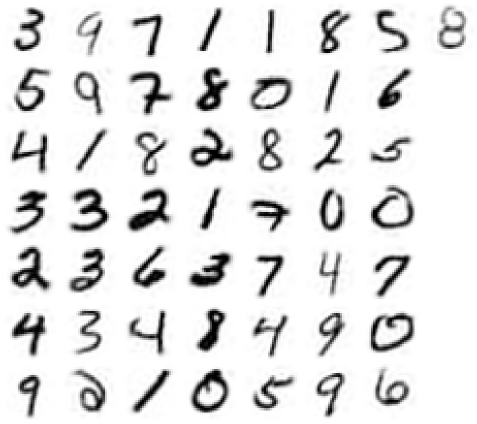
2.1 Initial results with simple models

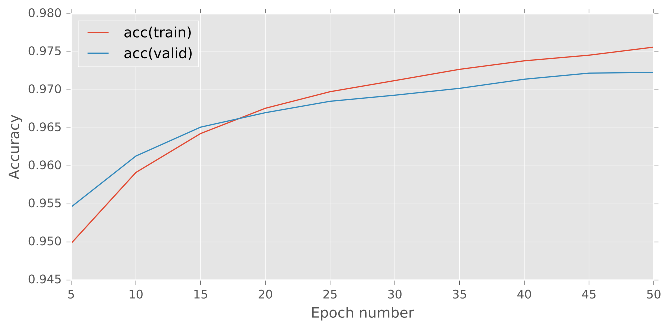
Firstly we will use a simple linear contractive autoencoder with a hidden layer of size 100, that will use the input of an image(784 data points) and will try to recreate it in the output. Our final trained (for 50 epochs, approx 4 seconds per epoch) autoencoder had an error of 2,64. Next step we will use the 100-dimensional output of the hidden layer and try to train our sample model from above.

In Figure 16 we can see the error and accuracy diagrams our non-linear model had, but this time trained with preprocessed data from an autoencoder, which reduced the dimensions from the initial 784 to 100. Our model had an final accuracy of 9.72% and the training time was approximately 1 second per epoch.

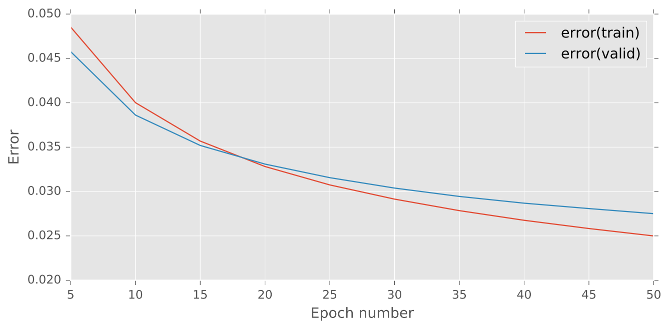
Our initial results show the impact that the autoencoder had in our model. The training time was a lot less and the final model, even though not as good as the previous, had only a 7% decline. The new 100-dimensional data contain the biggest chunk of information that was in the 784-dimensional data. For our next experiment we can try with a little more dimensions, lets say 300, to see how much our model


(a) Numbers from the MNIST dataset


(b) Same numbers created by the autoencoder

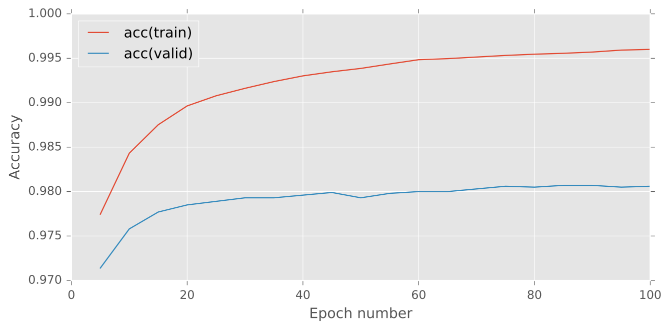


(c)

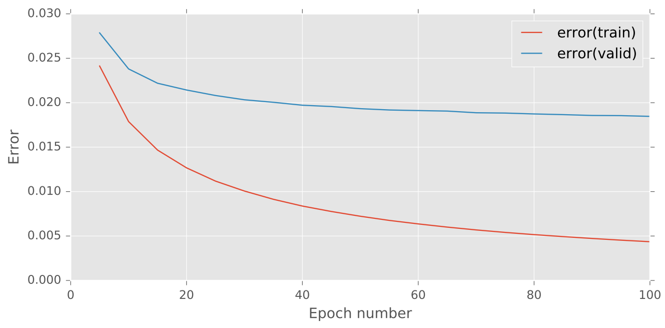


(d)

Figure 9: Error and Accuracy of our model with input from an autoencoder which produces 100-dimensional data



(a)



(b)

Figure 10: Error and Accuracy of our model with input from an autoencoder which produces 300-dimensional data

might improve. The autoencoder had en error of 4.85e-01, a small number which indicates that it can represent the input with the contractive layer pretty well.

After that our final model has validation accuracy equal to 9.81%, which is even better than our initial model without preprocessing. That means that our new input of 300 dimensions represents better our information, allowing it to achieve better results. The training time was aproximately 3.2 seconds per epoch, which logically is a lot faster than the first one. As we can see from 23, the convergence of the model is also faster, allowing us to train more efficiently our models.

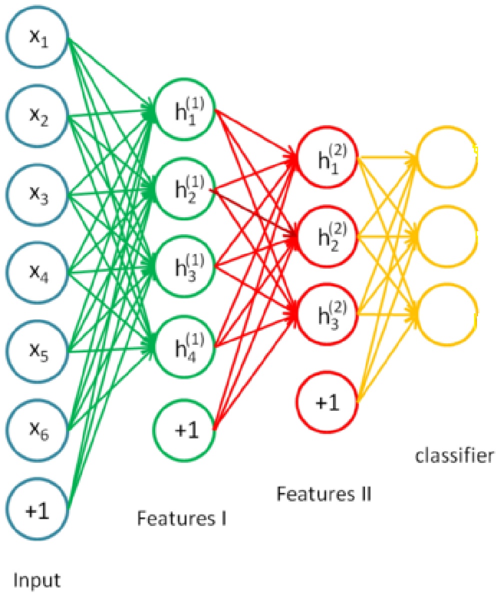
Autoencoder	**Error(train)**	**Error(val)**	**Accuracy(train)**	**Accuracy(val)**
No autoencoder	0.0524	0,163	0.983	0.963
100-dimensions	0.00031	0.108	1.00	0.979
300-dimensions	4.36e-03	1.85e-02	9.96e-01	9.81e-01

Table 3: Error and Accuracy of our models

2.2 Stacked Autoencoders

In this part we will use stacked autoencoders to see how they will affect our final model. At first we will preprocess our data with 2 stacked autoencoders. The features that are produced from the middle layer of the first autoencoder will be used as inputs and outputs in the process of training our second autoencoder. After that we will forward propagate our input from the middle trained layers of the first autoencoder, next from the second autoencoder, and we will use these data as inputs at our non-linear model to see how it reacts. We will use first an autoencoder with a hidden layer with size of 300. After it is trained, we will produce 300-dimensional features with it and use them as input to train a second autoencoder with a 100 sized hidden layer.

The error of the second autoencoder was 1,45.



(a)

Figure 11: 3-stacked Autoencoders

The result of our stacked autoencoder was 9.72% accuracy on the validation set, the same as when we used just the one 100-dimensional autoencoder. This maybe indicate that we cant fit any more information in 100 dimensions. Lets see if we will have better success by using different dimensions in our 2 stacked autoencoded preprocess.

Second experiment with 300-150 dimensional autoencoders. Error of the second autoencoder was 5.80e-01.

Autoencoder	Error(train)	Error(val)	Accuracy(train)	Accuracy(val)
No autoencoder	0.0524	0,163	0.983	0.963
300-100 stacked autoencoders	2.17e-02	2.78e-02	9.80e-01	9.72e-01
300-150 stacked autoencoders	2.17e-02	2.78e-02	9.80e-01	9.72e-01

Table 4: Error and Accuracy of our models with stacked autoencoders

Having seen the posibilities of two consecutive stacked autoencoders, we can now try with 3 stacked autoencoders and see if there will be some improvement. The first autoencoder has input/output of 784 and a hidden layer of size 400. The second autoencoder has input/output of 400 and a hidden layer of size 200, and the third autoencoder has input/output of 200 and a hidden layer of size 100. We train them, and they show error of 3.13e-01, 4.06e-01 and 1.28e+00 respectively. After trying the output of the third autoencoder to our model, the result is 9.71%, and the training required 0.6 seconds per epoch approximately. (error(train)=2.35e-02, acc(train)=9.77e-01, error(valid)=3.04e-02, acc(valid)=9.71e-01)

Lets try a more interesting 3-stacked autoencoder, with hidden layers 400,300 and 200 respectively.

$error(train) = 8.23e-03, acc(train) = 9.92e-01, error(valid) = 2.39e-02, acc(valid) = 9.78e-01, params_penalty = 0.00e+00$ Approx 2 secs per epoch

2.3 Deep Autoencoders

For our first experiment with deep autoencoders, we create an autoencoder with 4 affine layers, with output from each layer 1-400,2-100,3-400,4-784. The error of that autoencoder was 3.19e+00.

We input the features produced from the last autoencoder at our non-linear model. We get the following results. As we can see we get somewhat better results of validation accuracy of 9.77%.

Autoencoder	Error(train)	Error(val)	Accuracy(train)	Accuracy(val)
400-100-400 DA	1.59e-02	9.66e-02	9.97e-01	9.77e-01
400-200-100-200-400 DA	6.69e-02	1.02e-01	9.80e-01	9.72e-01

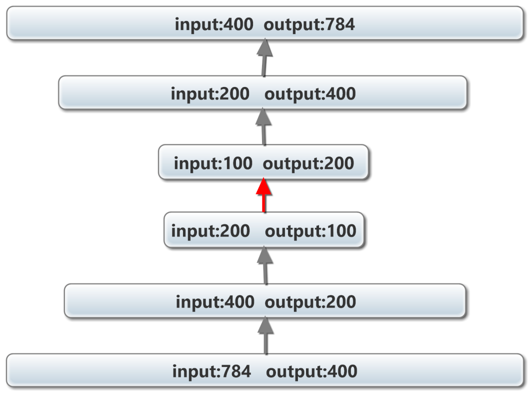
Table 5: Error and Accuracy of our models with deep autoencoders

We also tried a deeper autoencoder with 11 layers. It consisted of 6 affine layers and 5 Relu layers between them. Its format can be seen in Firgure 12. After the autoencoder is trained we get the output of size 100 (the red arrow in Figure 12), and we try to train our simple model with the 100-dimensional input produced. As we can see the result was not as good, maybe because larger networks require some more training before they are tuned completely.

2.4 Autoencoders and Convolutional Neural Network

We implemented a convolutional neural network and we want to see how its training time and its total performance will improve if we feed it pretrained data from an autoencoder.

In 6 we can see that we get a pretty good result of 9.89%. The best result we have had so far with autoencoders. Convolutional Neural Networks seem to work really good at this task even without data augmentation techniques. Furthermore the performance has been greately increased since we use the data from the autoencoder representation, and so the training time has been decreased significantly.

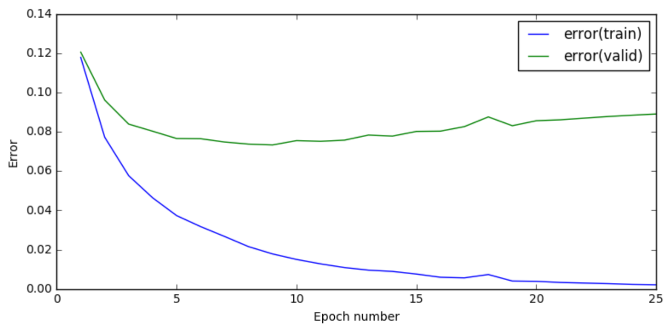


(a)

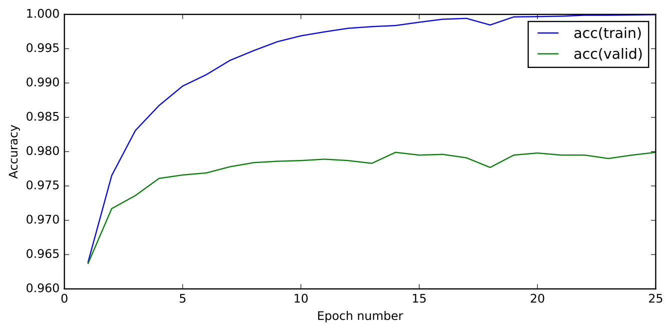
Figure 12: 11-layer Deep Autoencoder

Autoencoder	**Error(train)**	**Error(val)**	**Accuracy(train)**	**Accuracy(val)**
CNN+Autoencoder 225	8.09e-06	9.63e-02	1.00e+00	9.89e-01

Table 6: Error and Accuracy of our models with stacked autoencoders



(a)



(b)

Figure 13: Error and Accuracy of our CNN model with input from an autoencoder which produces 225-dimensional data

3 Topic 3: Batch Normalization

In this section, we will use the Batch Normalization Method to see how effectively we can reduce overfitting. Lets see some comparisons first of the same models being trained at first without and later with batch normalization.

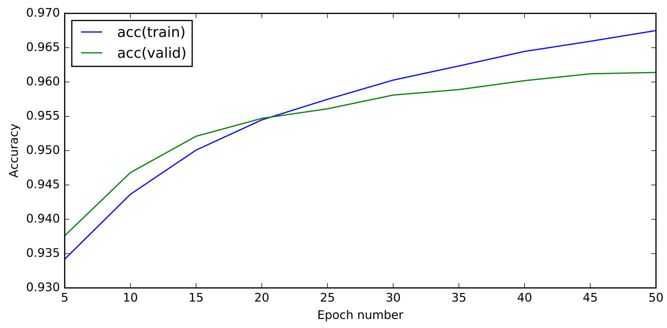
Our experiments were with learning rates equal to 0,01 and 0,1. Our first model consists of Affine-Sigmoid-Affine. Our second deeper model consists of Affine-Sigmoid-Affine-Sigmoid-Affine-Sigmoid-Affine. Our third deeper model consists of Affine-Relu-Affine-Relu-Affine-Relu-Affine.

We can see that batch normalization had a bigger impact at the models which consisted of Sigmoid layers. The ones with Rectified Linear layers had some difference but on much smaller scale. We present some indicative plots.

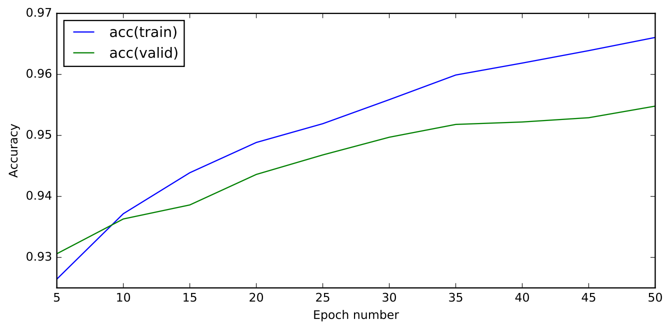
From the figures shown below we can see how our network behaves with batch normalization. According to our results, the batch normalization process improves the behaviour of the neural networks that consist of Sigmoid layers, while we get similar results while using rectified linear layers.

Autoencoder	**Error(train)**	**Error(val)**	**Accuracy(train)**	**Accuracy(val)**
3-layer+0,01 Tanh	1.19e-01	1.35e-01	9.96e-01	9.61e-01
3-layer+0,01 Sigmoid	1.26e-01	1.65e-01	9.96e-01	9.55e-01
3-layer+0,01+Relu	1.26e-01	1.65e-01	9.96e-01	9.74e-01
7-layer+0,01 Tanh	6.29e-02	9.67e-02	9.96e-01	9.69e-01
7-layer+0,1 Relu	3.32e-02	9.97e-02	9.96e-01	9.70e-01
7-layer+0,01 Relu	1.96e-03	1.09e-01	9.96e-01	9.77e-01

Table 7: Error and Accuracy of our CNN model with batch norm

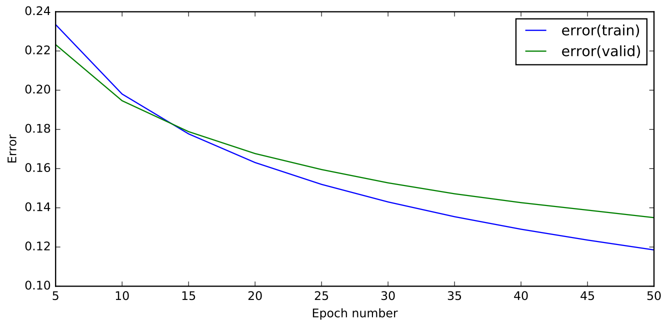


(a) Simple 3-layered sigmoid model

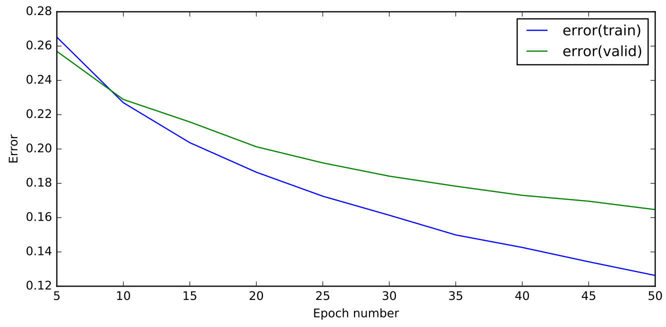


(b) Simple 3-layered sigmoid model with batch normalization

Figure 14: Comparison of accuracies in a model and the same model with batch normalization with learning rate=0,01

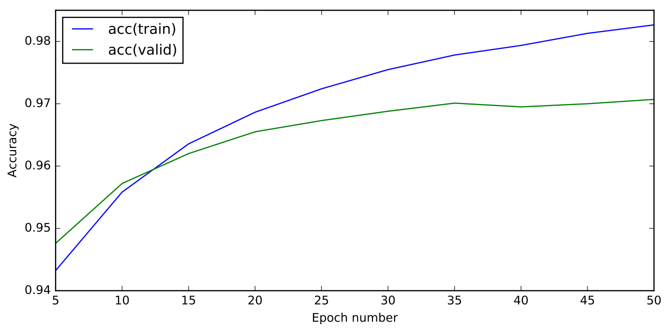


(a)

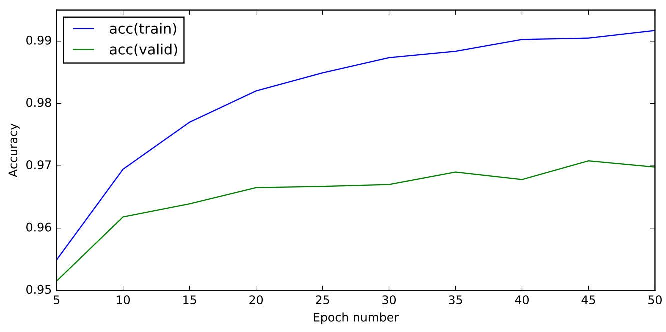


(b)

Figure 15: Comparison of errors in a model and the same model with batch normalization with learning rate=0,01

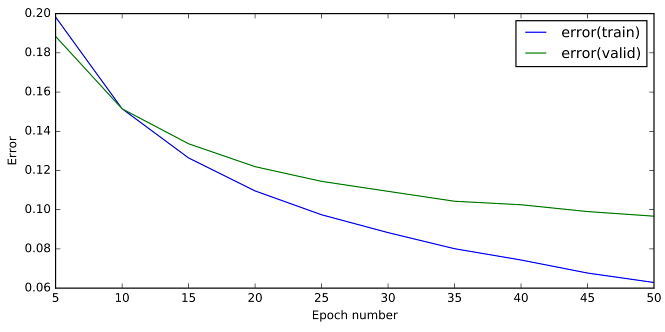


(a) 7-layer sigmoid model

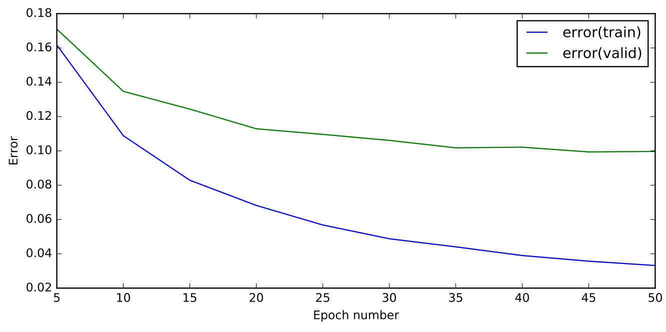


(b) 7-layered sigmoid model with batch normalization

Figure 16: Comparison of accuracies in a model and the same model with batch normalization with learning rate=0,01

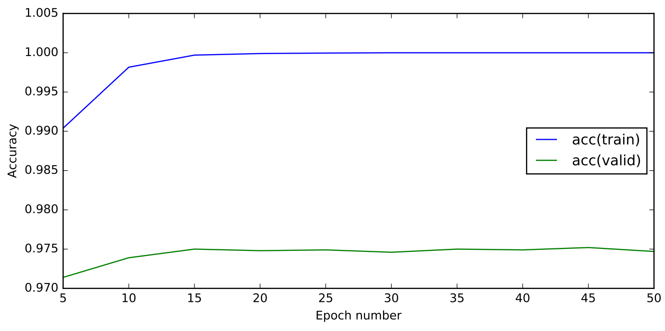


(a) Simple 4-layer sigmoid model

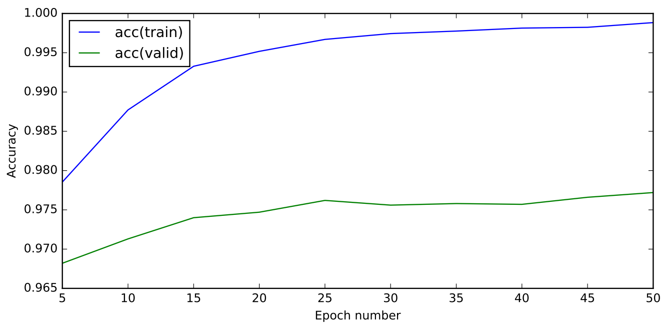


(b) Simple 4-layered sigmoid model with batch normalization

Figure 17: Comparison of errors in a model and the same model with batch normalization with learning rate=0,01

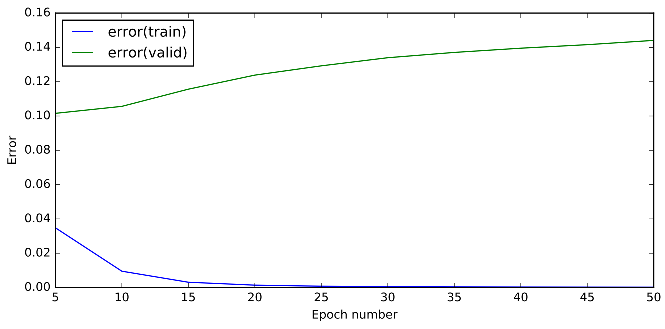


(a) 7-layer sigmoid model

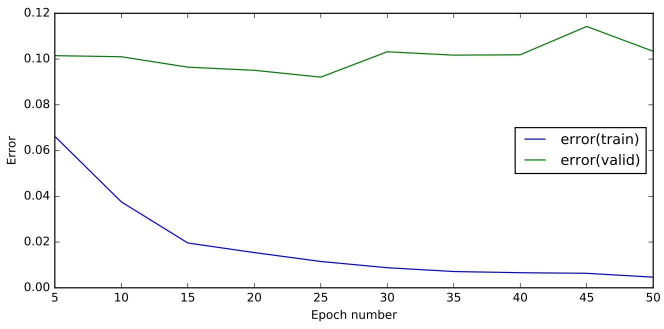


(b) 7-layered sigmoid model with batch normalization

Figure 18: Comparison of errors in a model and the same model with batch normalization with learning rate=0,1

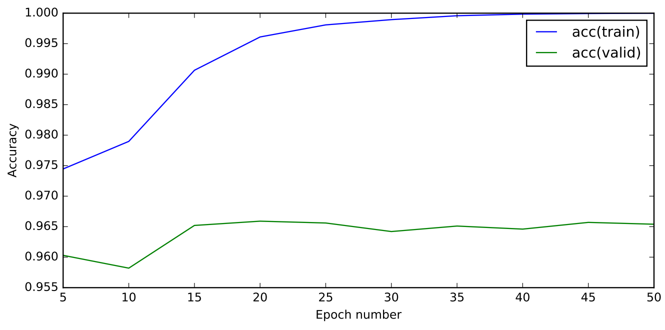


(a) 7-layer sigmoid model

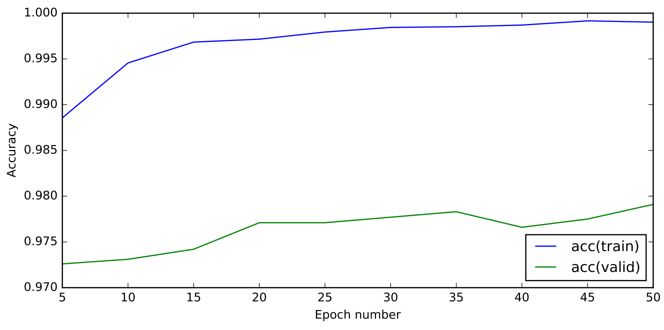


(b) 7-layer sigmoid model with batch normalization

Figure 19: Comparison of errors in a model and the same model with batch normalization with learning rate=0,1

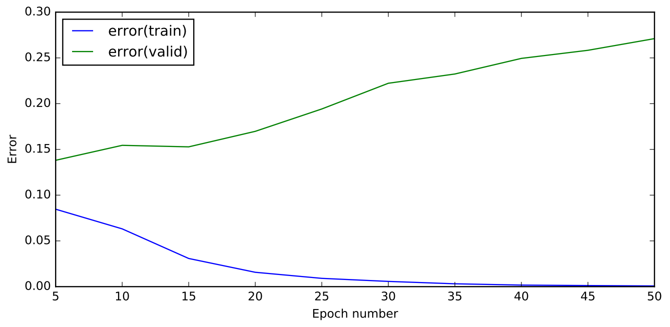


(a) 7-layer Relu model

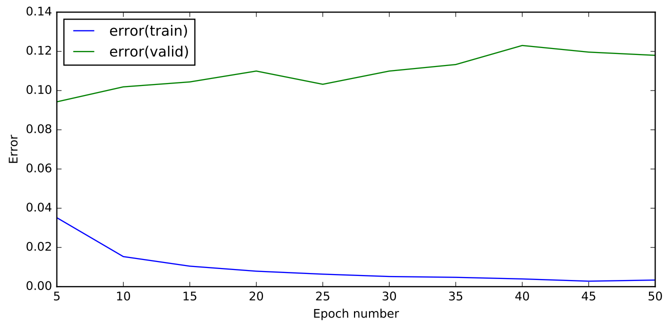


(b) 7-layered sigmoid model with batch normalization

Figure 20: Comparison of errors in a model and the same model with batch normalization with learning rate=0,1



(a) 7-layer Relu model



(b) 7-layer Relu model with batch normalization

Figure 21: Comparison of errors in a model and the same model with batch normalization with learning rate=0,1

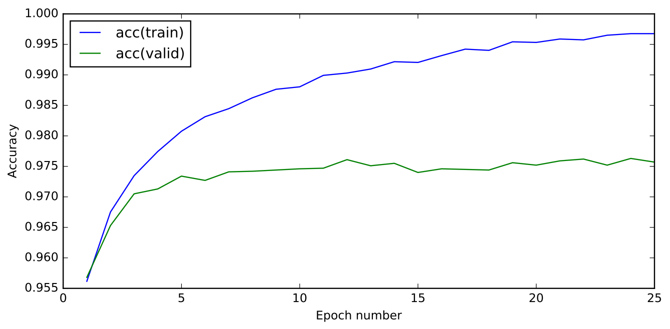
3.1 Batch Normalization in a Convolutional Neural Network

We implemented a convolutional layer which we used to run experiments and see how a neural network with 1 convolutional layer, 1 rectified linear layer and 1 affine layer reacts with batch normalization.

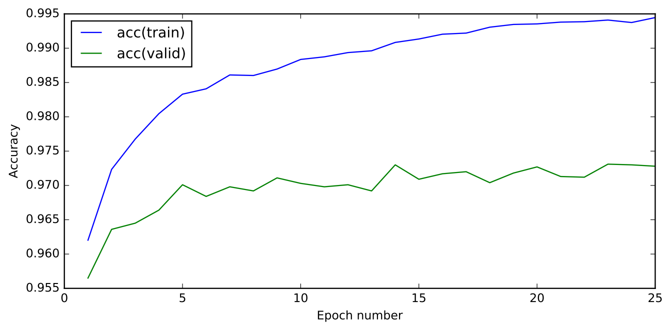
From the figures and the final results, we do not see any major difference in our two models. Maybe a batch normalization was not needed at the current model and so performs similarly. We see though from the figures how batch normalization affects slightly our error and accuracy curves. The same for the convolutional neural network, using batch normalization seems to have similar results. At least in this task, the convolutional neural network seems to not need any form of regularization since it is capable of creating really good representations without any help.

Autoencoder	**Error(train)**	**Error(val)**	**Accuracy(train)**	**Accuracy(val)**
Auto+Batch	2.01e-02	1.03e-01	9.96e-01	9.73e-01
Conv+Batch	1.63e-02	1.73e-02	1.00e-01	9.88e-01

Table 8: Error and Accuracy of our CNN model with batch norm

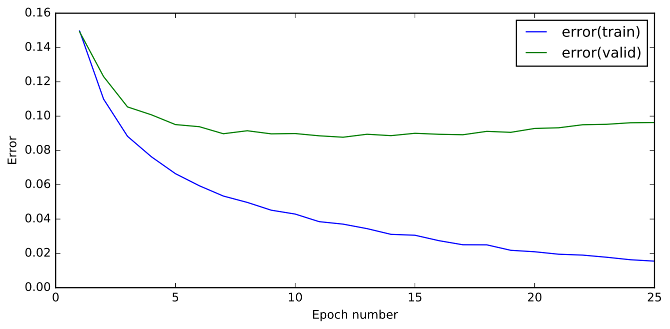


(a) Simple 4-layered sigmoid model

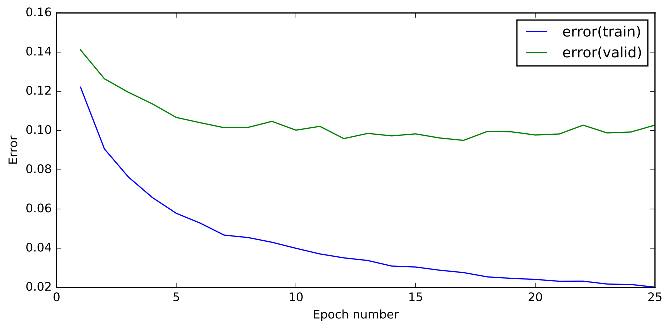


(b) Simple 4-layered sigmoid model with batch normalization

Figure 22: Comparison of errors in convolutional model and the same model with batch normalization with learning rate=0,01



(a) Simple 4-layered sigmoid model



(b) Simple 4-layered sigmoid model with batch normalization

Figure 23: Comparison of errors in a convolutional model and the same model with batch normalization with learning rate=0,01