# Grand Experiment 3: Stacked Autoencoders Pretraining

## Why choosing this experiment and what we are trying to achieve

**Autoencoders** are useful in many problems either to remove noise from images or other representations or when you want to do dimensionality reduction either because PCA is not enough because it lacks non-linearities or just because you want to plot high dimensional data in a 2D space in a way that makes sense. They are a way of representing an input with many features into another dimensional space that makes sense.

**Stacked Autoencoders** is a way to pretrain a neural network by making each layer be the autoencoded representation of the previous layer. In other ways each layer is like a distorted version of the input which is actually better than having the weights be randomly initialized.

In this experiment we are using stacked autoencoders to optimize the pretraining of our neural network.

Our intentions is a pretraining that is going to have two positive effects:

1) For large architectures with many layers there is the issue that the backpropagation is more "slow". This is understandable because we have many layers and in order for the error to propagate properly backwards and set the weights at a good level we might have saturations or other effects that cause very slow learning at the first epochs.

We are going to test that by running an experiment with six or more layers where the slow-learning effect is apparent and then repeat the experiment with our stacked autoencoders pretraining. The expectation is that our pretraining will not go through this phase of slow learning and the learning process will converge more quickly.

2) We should get a very good performance always in comparison to random initialization of the weights.

We are going to test that by repeating the same experiment a few times for various random initializations of the weights. Then for the same architecture we are going to pretrain the network using stacked autoencoders and run to see the final error and accuracy of both training and validation set. We must see that this pretraining yields better or equally good results as the best of the cases with the random initializations.

We will experiment with two kinds of stacked autoencoders: **Linear Stacked Autoencoders** and **Non-Linear Stacked Autoencoders**.

## Neural Network Architecture specifications

We are going to be using **Gradient Descent Learning Rule** because the adaptive learning rules might disguise the effects that we are hoping to get with and without pretraining.

For simplicity and because of low resources we will not do any **regularization** other than early stopping.

We are using **glorot initialization**, a special parameter initialisation scheme for the weights which makes the scale of the random initialisation dependent on the input and output dimensions of the layer, with the aim of trying to keep the scale of activations at different layers of the network the same at initialisation.
We do this because we do not want the weights to get too big too soon and saturate the network.
We also initialise the **biases to zero** as this is not going to affect the gradients.

For all experiments **we use a batch size of 50** and we use **affine transformations** which are interleaved with **Sigmoid** nonlinearities, and at the end we always have a **softmax output layer**.

We are going to use multiple architectures in terms of how many layers we use in order to see and explore the effects of pretraining in deep neural networks. Based on our experiences from the experiments of this and the past coursework we will use the **dimensionality of the hidden layer** to always be **100**.

# Implementing Stacked Autoencoders

It is easy with our current implementations to create an Autoencoder because we already have a data provider to provide the input as the 784 pixels of a 28x28 image.

But in order to create a second layer in our neural network pretrained by an autoencoder we need the first layer to play the role of the input (and output) for this autoencoder and this requires a new kind of Data Provider.

## StackedAutoEncoderDataProvider

This is a subclass of MNISTDataProvider and it provides as input and output the batch of the inputs after they have been processed through a neural network up until the level we want.

To achieve that our Data provider has access to the current model that we are pretraining and it uses its fprop method to get a representation of the image for the layers that we have so far pretrained.

As an example if we want to pretrain the second layer of our network then the model passed as parameter in the Data Provider class must have the first layer already pretrained and this layer is used to transform the 784 dimensional inputs of our image to the representation of that layer which it will have features with 100 dimensionality in our case.
Recall that this layer has been pretrained to provide an as accurate as possible representation of the input. And this representation is that we want to have as input the second iteration of our stacked autoencoder where an autoencoder is used to build the second layer by using the first.

If model is set as *None* in python then the StackedAutoEncoderDataProvider falls back to a simple MNISTDataProvider which returns the inputs as inputs and ouputs.

## Linear Stacked Autoencoder

The Linear Stacked Autoencoder does not include any non-linearities.
It is composed by two affine layers where the input and output layer have the same dimensionality since we want the input to match the output and the hidden layer is the layer that we want to pretrain.

As an error we are using the Sum of Squared Differences Error because we want to match the input with the output.

We are using our best Adaptive Learning Rule which is Adam Learning Rule with default parameters.

When pretraining is finished the affine layer that refers to the ouput is of no concern to us and we are discarding it to keep first affine layer as our pretrained layer.

## Non-Linear Stacked Autoencoder

The Non-Linear Stacked Autoencoder is composed by two affine layers interleaved by two non-linearities which here we have chosen the Sigmoids.

The input and output layer have the same dimensionality since we want the input to match the output and the hidden layer is the layer that we want to pretrain.

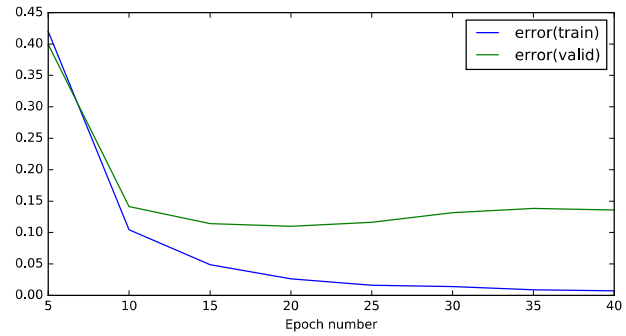
As an error we are using the Sum of Squared Differences Error because we want to match the input with the output.

We are using our best Adaptive Learning Rule which is Adam Learning Rule with default parameters.

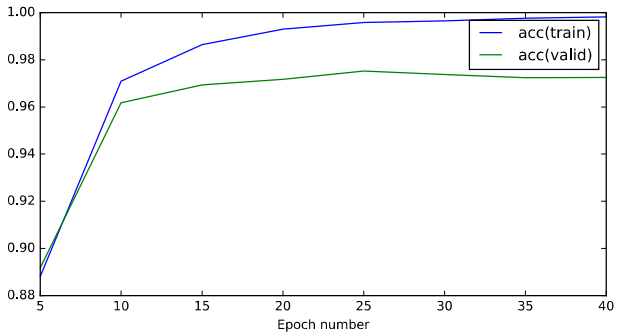
When pretraining is finished the affine layer and the sigmoid non-linearity that refers to the ouput is of no concern to us and we are discarding both of them to keep only the first affine layer as our pretrained layer. Note here that if we keep or not the Sigmoid non-linearity does not make a difference since this layer does not contain parameters, does not store any state.

# Experiment A: avoiding slow learning effects on multi-layer deep neural networks

## Six Layers without pretraining




*Graph 1: Training & Validation Error – Six Layers without Pretraining*




*Graph 2: Training & Validation Accuracy – Six Layers without Pretraining*

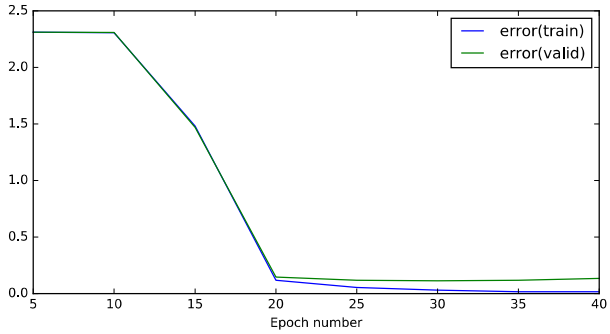
Runtime: `111.428107977 seconds`

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.97249999999999903` |
| `Final Testing Error` | `0.13592290449311858` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.99816000000000116` |
| `Final Training Error` | `0.0069894214987917744` |

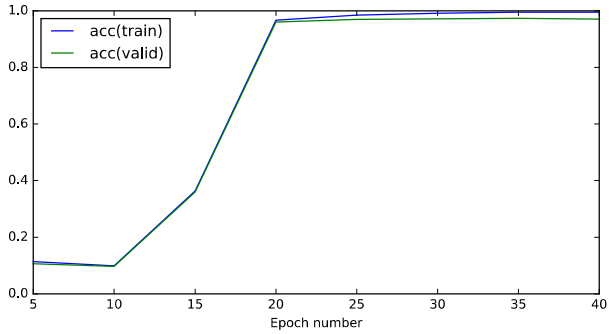
We see in the plots that we start from low accuracy and higher error because of the too many layer of the model. Actually we peak at the 25 epochs and then because of overfitting the metrics are worse resulting in a relatively bad final accuracy.

Now we will construct a model of six layers with 100 dimensionality in each hidden layer via a Linear Stacked AutoEncoder.
This will give us six affine layers which we interleave with Sigmoid Layers in order to have a proper neural network with non-linearities.

# Six Layers with Linear Stacked Autoencoder Pretraining




*Graph 3: Training & Validation Error – Six Layers with Linear Stacked Autoencoder Pretraining*




*Graph 4: Training & Validation Accuracy – Six Layers with Linear Stacked Autoencoder Pretraining*

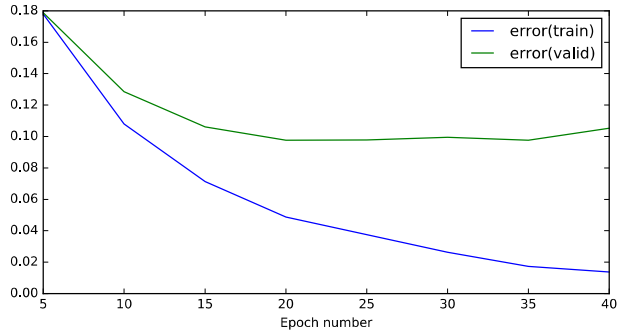
Runtime: `110.861319065` seconds

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.9709999999999992` |
| `Final Testing Error` | `0.135004313417807` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.99542000000000253` |
| `Final Training Error` | `0.015544239697404282` |

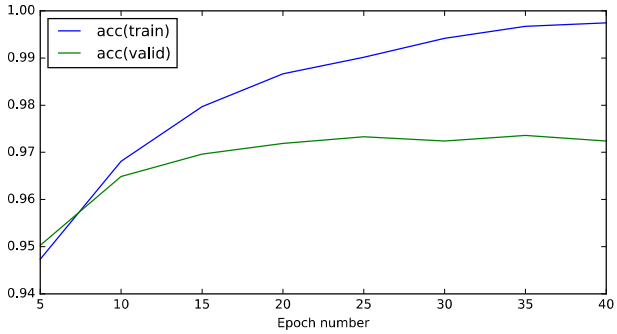
Our linear stacked autoencoder seems to have done a poor job in pretraining the network. It seems that we have evoked the effect that we were trying to avoid!

Let’s try and repeat the same experiment using the Non-Linear Stacked Autoencoder for Pretraining

# Six Layers with Non-Linear Stacked Autoencoder Pretraining




*Graph 5: Training & Validation Error – Six Layers with Non-Linear Stacked Autoencoder Pretraining*




*Graph 6: Training & Validation Accuracy – Six Layers with Non-Linear Stacked Autoencoder Pretraining*

Runtime: `110.105275154` seconds

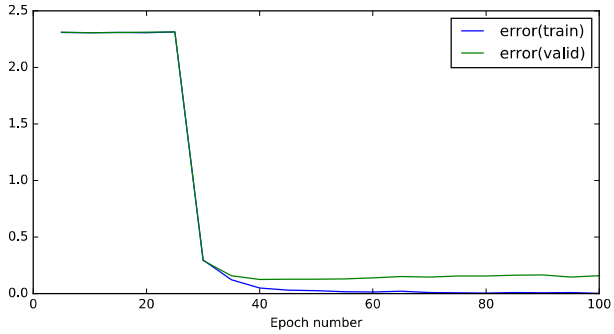
| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.97239999999999904` |
| `Final Testing Error` | `0.10528195949162089` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.99748000000000137` |
| `Final Training Error` | `0.01367461917368043` |

The Non-Linear Stacked Autoencoder Pretraining did a much better job than the Linear Stacked Autoencoder.

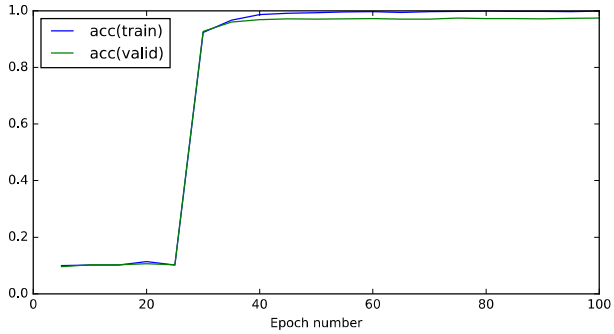
Gladly this experiment is in consice with our original intentions. Here we don't see a slow learning rate at the beginning but rather an immediate increase of the accuracy and decrease of the error.

The final accuracy is better than the experiment with the Linear Stacked Autoencoder.

## Seven Layers without pretraining




*Graph 7: Training & Validation Error – Seven Layers without Pretraining*




*Graph 8: Training & Validation Accuracy – Seven Layers without Pretraining*

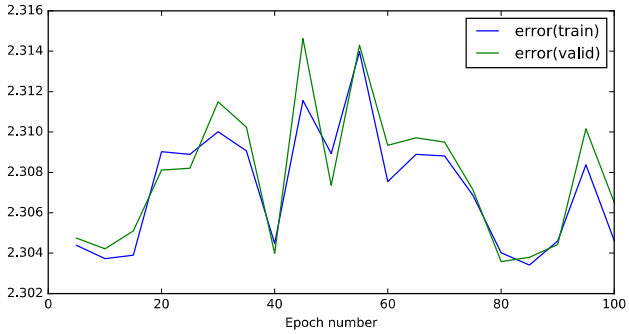
Runtime: `319.806434155` seconds

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.97459999999999836` |
| `Final Testing Error` | `0.15829059585158961` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.99946000000000024` |
| `Final Training Error` | `0.002222706411481464` |

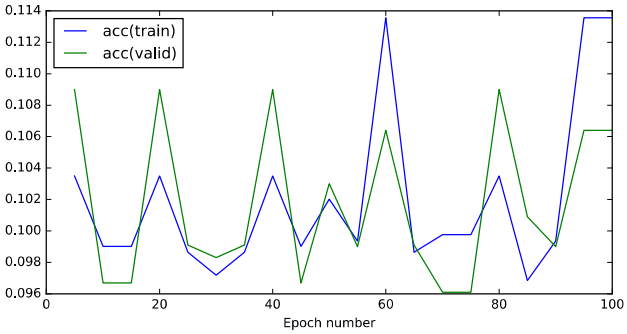
With 7 layers the effect is more severe. The learning is really slow in the beginning and we have to go to more than 40 epochs so that the neural network really starts learning

Now let's construct a model of seven layers with 100 dimensionality in each hidden layer via a Linear Stacked AutoEncoder and see if we are going to alieviate this unwanted effect.

# Seven Layers with Linear Stacked Autoencoder Pretraining




*Graph 9: Training & Validation Error – Seven Layers with Linear Stacked Autoencoder Pretraining*




*Graph 10: Training & Validation Accuracy – Seven Layers with Linear Stacked Autoencoder Pretraining*

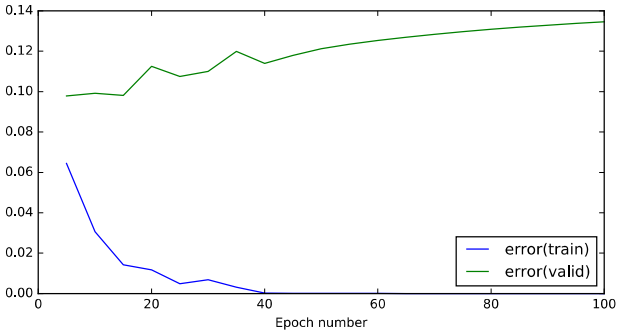
Runtime: `345.946224928` seconds

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.10639999999999988` |
| `Final Testing Error` | `2.3065173136848172` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.11355999999999987` |
| `Final Training Error` | `2.3046307375203297` |

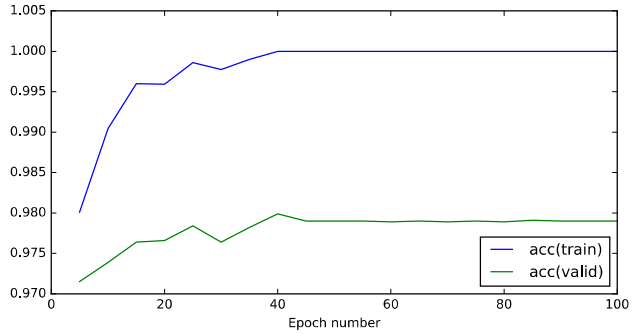
On this experiment the Linear Stacked Autoencoder Pretraining resulted in really bad results for the seven layers. The model is not able to converge and the metrics show that we have not really succeeded any training. The training and testing accuracy is near to 10% which is like the prior probability.

Next we will try again with the Non-Linear Stacked Autoencoder to try and yield better results.

## Seven Layers with Non-Linear Stacked Autoencoder Pretraining




*Graph 11: Training & Validation Error - Seven Layers with Non-Linear Stacked Autoencoder Pretraining*




*Graph 12: Training & Validation Accuracy - Seven Layers with Non-Linear Stacked Autoencoder Pretraining*

Runtime: `337.20222497` seconds

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.97899999999999876` |
| `Final Testing Error` | `0.13459701113216441` |
| `Final Training Accuracy (percentage 0 to 1)` | `1.0` |
| `Final Training Error` | `2.2924539333227827e-05` |

We see that the non linear stacked autoencoder yields better results. Even with 7 layers there is no phase where we have slow-training. The training reachs optimal error and accuracy at ~40 epochs.

## Conclusions on Experiment A

The above experiments show that the Linear Stacked Autoencoder is not a reliable system to use. The massive failure of our experiments could be explained by our architecture where we have consecutive hidden layers of the exact same dimensionality.
So for example in our case if you have as input and output for a Linear Autoencoder 100 features and you are asking to train the optimal hidden layer of again 100 features you are more likely to get the identity matrix which means that lots of weights are saturated. This approach could cause the unwanted effects we experienced above.

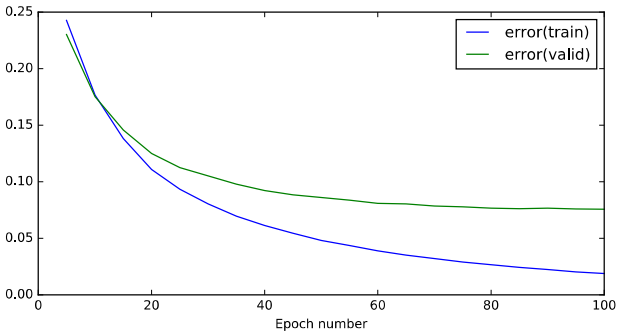
On the other hand a Non-Linear Stacked Autoencoder alleviated the issue of slow learning for deep neural network and the pretraining brought the initial state of the neural network at a place that is suitable for our classification task. Therefore the Non-Linear Stacked Autoencoder is recommened for multilayer deep neural networks.

# Experiment B: yielding smaller error and higher accuracy by pretraining instead of randomly initializing weights

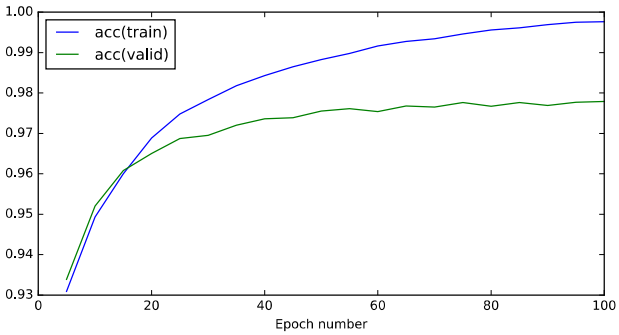
Our goal is to initialize the randomly generated weights with different seeds and see what kind of results we are getting. Then use Pretraining with Stacked Autoencoders and compare the final performance and accuracy.

# Two Layer Model with random initialization of weights with Glorot Uniform Initialization

Let's see how our two layer model behaves within 100 epochs




*Graph 13: Training & Validation Error - Two Layers without Pretraining*

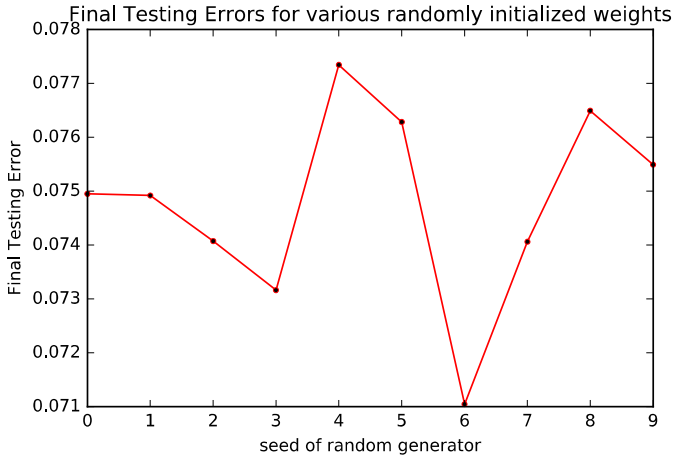



*Graph 14: Training & Validation Accuracy - Two Layers without Pretraining*

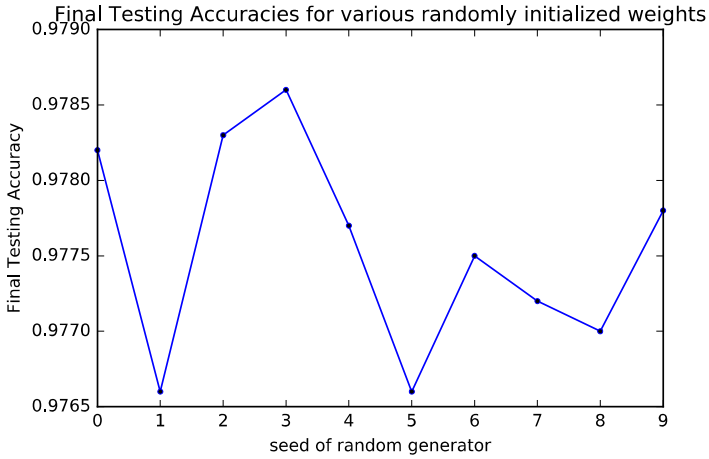
```
Runtime: 126.894783974 seconds
```

| | |
|---|---|
| Final Testing Accuracy (percentage 0 to 1) | 0.97789999999999777 |
| Final Testing Error | 0.075733012679254696 |
| Final Training Accuracy (percentage 0 to 1) | 0.99764000000000153 |
| Final Training Error | 0.018958982953952288 |

# Two Layer Model multiple experiments with various random initializations of weights with Glorot Uniform Initialization




*Graph 15: Final Validation Error for various randomly initialized weights for a two layer deep neural network*




*Graph 16: Final Validation Accuracy for various randomly initialized weights for a two layer deep neural network*

| | |
|---|---|
| Maximum Testing Accuracy | 0.97859999999999903 |
| Mean Testing Accuracy | 0.97754999999999903 |
| Minimum Testing Accuracy | 0.97659999999999914 |
| Maximum Testing Error | 0.077343073407462298 |
| Mean Testing Error | 0.074782804560505595 |
| Minimum Testing Error | 0.071047685963312984 |

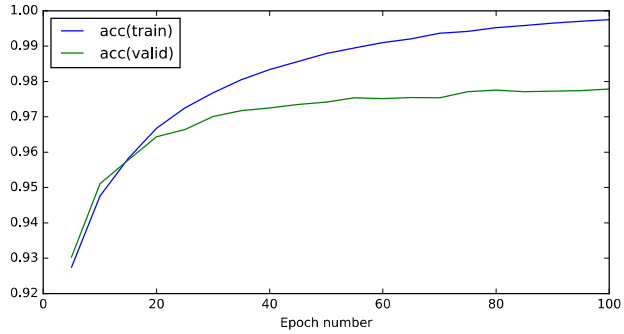
The final accuracy is between ~97.66% to ~97.86%

and the validation error is between ~0.071 to ~0.077

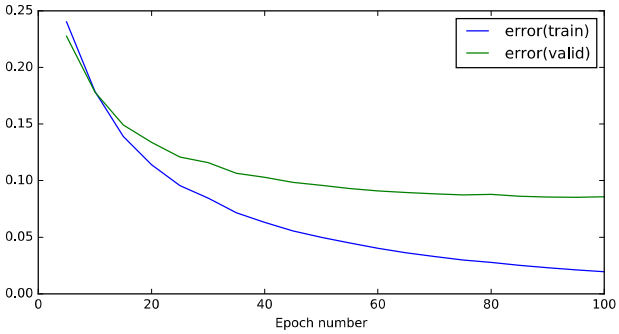
We will use this information for comparisons below

Next we will repeat the experiment by pretraining with a Linear Stacked Autoencoder.

# Two layers with Linear Stacked AutoEncoder Pretraining




*Graph 17: Training & Validation Error - Two layers with Linear Stacked AutoEncoder Pretraining*




*Graph 18: Training & Validation Accuracy - Two layers with Linear Stacked AutoEncoder Pretraining*

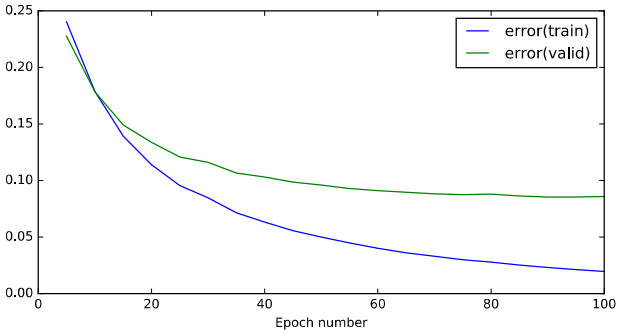
`Runtime: 129.11024189 seconds`

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.9778999999999991` |
| `Final Testing Error` | `0.071578519487560929` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.99748000000000137` |
| `Final Training Error` | `0.019062135396377036` |

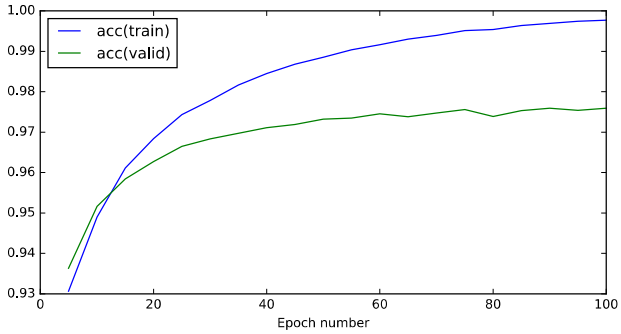
The pretraining with the linear stacked autoencoder yielded in general good results.
The validation accuracy is near the upper end of the range when using random initializations.
Also the validation error is near the lower end of the range when using random initializations.
The results can be described as good but not spectacular.

We will repeat the same experiment using a Non-Linear Stacked Autoencoder

## Two layers with Non-Linear Stacked AutoEncoder Pretraining




*Graph 19: Training & Validation Error - Two layers with Non-Linear Stacked AutoEncoder Pretraining*




*Graph 20: Training & Validation Accuracy - Two layers with Non-Linear Stacked AutoEncoder Pretraining*

Runtime: `131.107201099 seconds`

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.97589999999999943` |
| `Final Testing Error` | `0.085712887076613342` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.99770000000000147` |
| `Final Training Error` | `0.019567442932861615` |

Even though we expected the Non-Linear Stacked Autoencoder to perform better based on previous experiments with Non-Linear Stacked Autoencoder in fact here the metrics after the pretraining are worse.

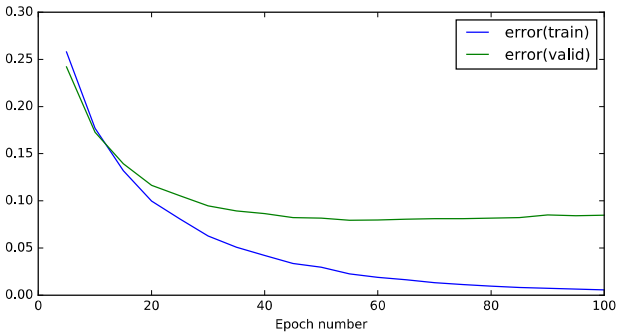
## Comparison Table for Two Layers

| **Two Layers** | Random Initialization of Weights | | Linear Stacked Autoencoder | Non-Linear Stacked Autoencoder |
|---|---|---|---|---|
| Testing Accuracy | `Maximum` | `0.97859999999999` | `0.97789999999999` | `0.97589999999999` |
| | `Mean` | `0.97754999999999` | | |
| | `Minimum` | `0.97659999999999` | | |
| Testing Error | `Maximum` | `0.07734307340746` | `0.07157851948756` | `0.08571288707661` |
| | `Mean` | `0.07478280456050` | | |
| | `Minimum` | `0.07104768596331` | | |

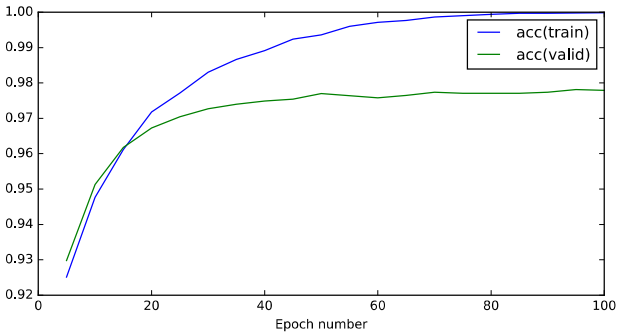
The conclusion is that the pretraining for two layers using either Linear or Non-Linear Stacked Autoencoders does not guarantee better results nor does it have any other significant impact.

We will try to see if any effects emerge when the neural network is deeper with more layers.

# Three Layer Model with random initialization of weights with Glorot Uniform Initialization




*Graph 21: Training & Validation Error - Three Layer Model with random initialization of weights with Glorot Uniform Initialization*




*Graph 22: Training & Validation Accuracy - Three Layer Model with random initialization of weights with Glorot Uniform Initialization*

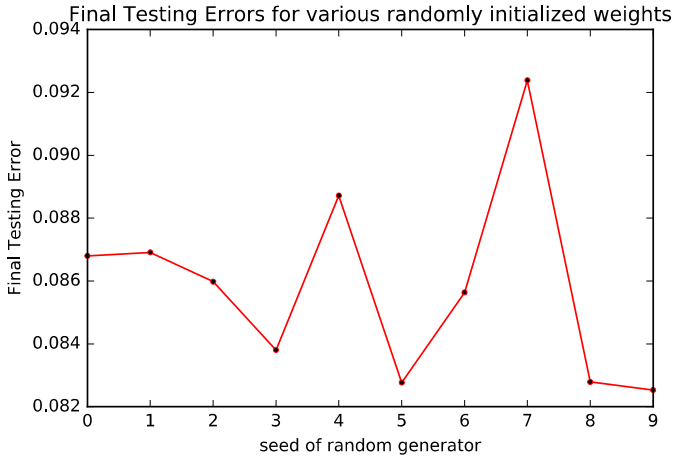
Runtime: `173.823220015 seconds`

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.9778999999999991` |
| `Final Testing Error` | `0.084736826625688905` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.99982000000000015` |
| `Final Training Error` | `0.0055768883911021475` |

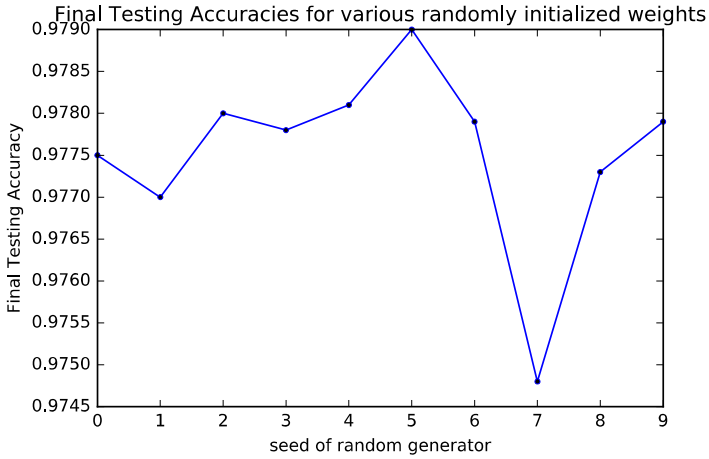
After 100 epochs we have reached at ~97.78% of accuracy for our classification task

We will repeat the experiment for various random initializations and explore the range of testing error and testing accuracy.

# Three Layer Model multiple experiments with various random initializations of weights with Glorot Uniform Initialization




*Graph 23: Final Testing Errors - Three Layer Model multiple experiments with various random initializations of weights with Glorot Uniform Initialization*




*Graph 24: Final Testing Accuracies - Three Layer Model multiple experiments with various random initializations of weights with Glorot Uniform Initialization*

| | |
|---|---|
| Maximum Testing Accuracy | 0.97899999999999876 |
| Mean Testing Accuracy | 0.97752999999999912 |
| Minimum Testing Accuracy | 0.97479999999999867 |
| Maximum Testing Error | 0.092384759482505838 |
| Mean Testing Error | 0.08583166386151371 |
| Minimum Testing Error | 0.082530967717292694 |

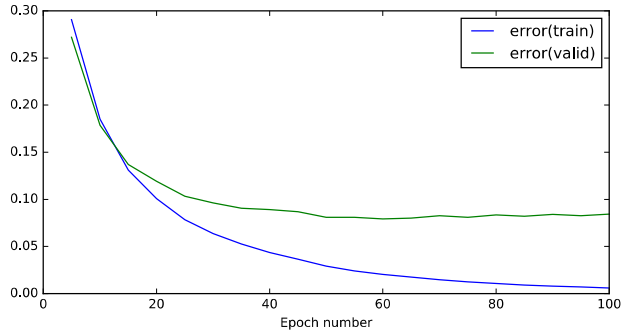
The final accuracy is between ~97.48% to ~97.90%

and the validation error is between 0.082 to 0.092

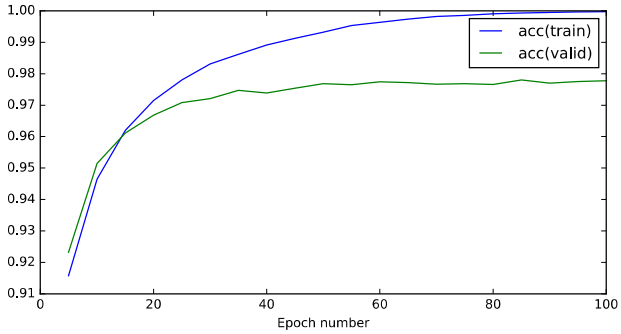
We will use this information for comparisons below

Next we will repeat the experiment by pretraining with a Linear Stacked Autoencoder.

# Three layers with Linear Stacked AutoEncoder Pretraining




*Graph 25: Training & Validation Error - Three layers with Linear Stacked AutoEncoder Pretraining*




*Graph 26: Training & Validation Accuracy - Three layers with Linear Stacked AutoEncoder Pretraining*

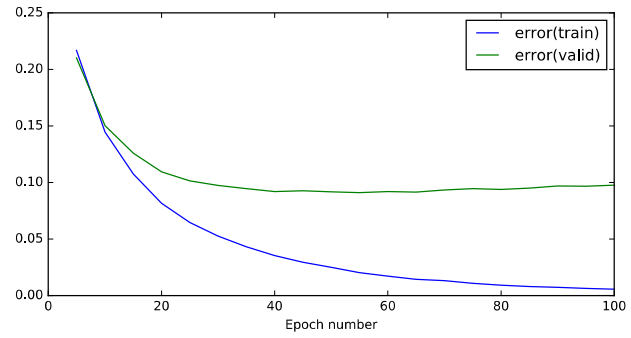
```
Runtime: 168.464905977 seconds
```

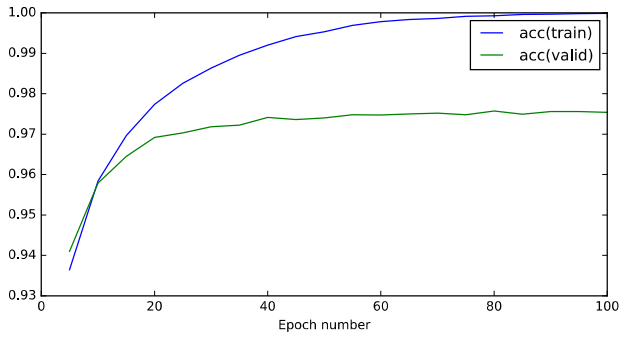
| | |
|---|---|
| Final Testing Accuracy (percentage 0 to 1) | 0.977799999999999 |
| Final Testing Error | 0.084256629397097479 |
| Final Training Accuracy (percentage 0 to 1) | 0.99974000000000007 |
| Final Training Error | 0.0058863214910023351 |

The final testing accuracy and final testing error are above and below the mean respectively in comparison to experiment with various initialization of the weights which is a first but not particularly strong indicator of the effects of pretraining with a linear stacked autoencoder.

We will repeat the same experiment using a Non-Linear Stacked Autoencoder

## Three layers with Non-Linear Stacked AutoEncoder Pretraining




Runtime: `163.474694967 seconds`

| | |
|---|---|
| `Final Testing Accuracy (percentage 0 to 1)` | `0.97539999999999893` |
| `Final Testing Error` | `0.09755379924948486` |
| `Final Training Accuracy (percentage 0 to 1)` | `0.9998800000000001` |
| `Final Training Error` | `0.0055896778343472566` |

Nothing special for the Pretraining with Non-Linear Stacked Encoder in comparison to previous experiments

## Comparison Table for Three Layers

| **Three Layers** | Random Initialization of Weights | | Linear Stacked Autoencoder | Non-Linear Stacked Autoencoder |
|---|---|---|---|---|
| Testing Accuracy | `Maximum` | `0.97899999999999` | `0.97779999999999` | `0.97539999999999` |
| | `Mean` | `0.97752999999999` | | |
| | `Minimum` | `0.97479999999999` | | |
| Testing Error | `Maximum` | `0.09238475948250` | `0.08425662939709` | `0.09755379924948` |
| | `Mean` | `0.08583166386151` | | |
| | `Minimum` | `0.08253096771729` | | |

The conclusion is that the pretraining with either Linear or Non-Linear Stacked Autoencoder failed to yield better performance than the random initialization of the weights.

## Conclusions for Experiment B

For the MNIST dataset and our current architecture we have proven from previous experiments of this coursework that the metrics are optimal when using three layers. But on this Grand Experiment we saw that pretraining made a difference when the architecture is chosen to be more complex than it is required. This is extremely useful when we have enough computational power to execute an experiment with a very deep and wide neural network but we lack the computational power to repeat this experiment multiple times to find which is the optimal architecture. On that perspective Non-Linear Stacked Autoencoder Pretraining is a useful tool.

However the experiments with pretraining did not yield any spectacular metrics in comparison to the ones with the random initialisation of the weights. In other words we did not achieve the outcome that we were aiming for before these experiments.

The next logical step is to test how pretraining would behave to a more complex problem that would require a deeper neural network.