Grand Experiment 2: Annealed Dropout with Maxout

Why choosing this experiment and what we are trying to achieve

Dropout originally is exploited as a method for regularization since the inputs of each layer change randomly in a probabilistic way and therefore it is more "difficult" for the neural network to ever fit exactly to the peculiarities of the training data, or in other words overfit.
Quote from the paper Annealed Dropout Training of Deep Networks: "Dropout training discourages the detectors in the network from co-adapting, which limits the capacity of the network and prevents overfitting."

However in annealing dropout the point is to start from a low inclusive probability, probability of input being included in output in dropout layer, and **gradually increase the probability** as the epochs go by until 100%.

So in this case if we let the training run for lots of epochs for 100% after annealing dropout is finished then indeed it is expected to overfit.

We know that **dropout simulates approximately an exponential number of neural networks** so as we start from a lower probability we have only a few neurons, randomly chose, play the role of adapting to the model each time. Of course these small models are expected to fail but they are going to be highly pretrained ancestors of the new more complex models when we start increasing the probability. **Unnecessary co-adaptation between neurons** that we might have if we start training the entire network altogether **is now avoided**.

The ultimate goal is to see if annealing dropout always yields the **highest performance and the lowest error rate** in comparison with a simple dropout procedure.

Note that in this grand experiment **we are are not optimizing for accuracy** neither optimizing for performance. Rather we want to see how annealing dropout yield higher results in comparison with a simple dropout procedure.

Neural Network Architecture specifications

We are going to be using **Momentum Learning Rule** and **RMSProp Learning Rule** and it is going to be mentioned which one, and with which parameters is used in each case. We will use Momentum Learning Rule because we have already executed experiments with Momentum Learning Rule using Max Pooling that we will use as a baseline. Secondly we are going to use the RMSProp Adaptive Learning Rule because it yielded very good and quick results in the experiments of coursework 1 and needed little attention to the careful selection of its parameters.

Dropout itself will handle the **regularization** part but in the annealing dropout case early stopping is necessary.

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 **max pooling** nonlinearities, and at the end we always have a **softmax output layer**.

Note that we are going to use an architecture that is in consice with the conclusions from Grand Experiment 1.
We are going to use the **two layer model** which is much faster in terms of performance and it will allow us to execute more experiments. This model is going to have a **dimensionality of 140 for the hidden layer**.

Implementing Annealing Dropout as a Scheduler named AnnealedDropoutScheduler

Because annealing dropout changes the inclusion probability as a function of the epochs we introduced a minor hack where we created a new Scheduler class that has a reference to the model and can change the probability of the dropout layers.

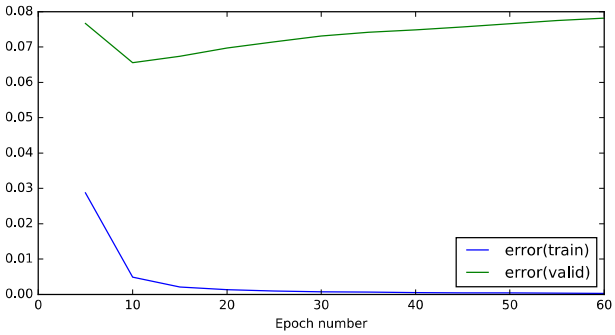
The **update_learning_rule** method which is essential in the interface of all the schedulers as it is called inside the optimizer is being exploited because it provides as a parameter the **epoch number**. We neglect the learning rule parameter of the method as it is not necessary.

The class is initialized with five attributes (the constructor has five parameters):

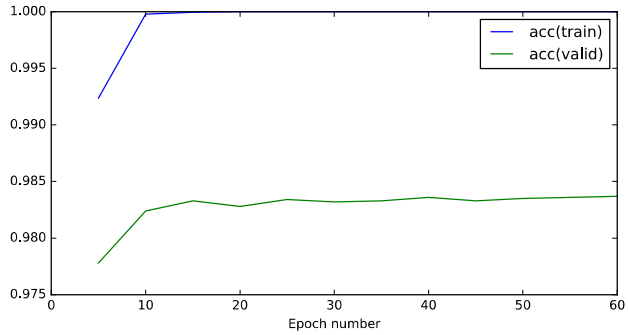
- **model**: this must the a reference to the model with all the layers. This scheduler has no effect if the model does not contain any dropout layers at all
- **startInputInclProb**: This is the initial inclusive probability for the input. We want to treat input with a different probability than the output
- **startHiddenInclProb**: This is the initial inclusive probability for all the hidden layers of the neural network.
- **epochs**: The total number of epochs of the experiment that the model is going to be used
- **holdOnPercent**: The final probability is always 100% or 1 and this percentage expresses how much percent of all epochs the probability is going to be equal to 1.

Probabilities are being increased in a linear way as the epochs go by. This is true both for the input probability and the rest of the probabilities for the hidden layers.

Baseline: Max Pooling
pool size: 2
Momentum Learning Rule learning rate: 0.02 and coefficient: 0.9



Graph 1: Training & Validation Error - Max Pooling - pool size: 2 - Momentum Learning Rule - learning rate: 0.02 and mom coefficient: 0.9



Graph 2: Training & Validation Accuracy - Max Pooling - pool size: 2 - Momentum Learning Rule - learning rate: 0.02 and mom coefficient: 0.9

Runtime: 389.950199127 seconds

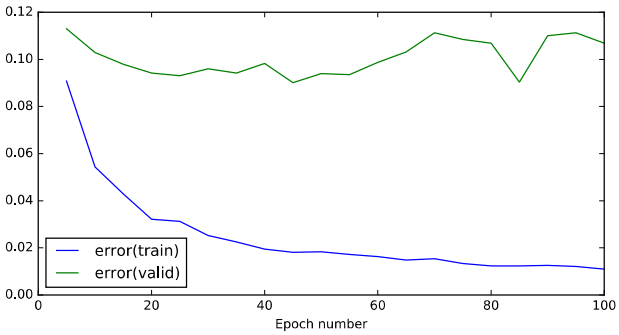
Final Testing Accuracy (percentage 0 to 1)	0.98369999999999902
Final Testing Error	0.078179863515805789
Final Training Accuracy (percentage 0 to 1)	1.0
Final Training Error	0.00032944897215494046

Even with smaller learning rate the momentum drives the learning properly to achieve an accuracy of higher than ~98% within 10 epochs.

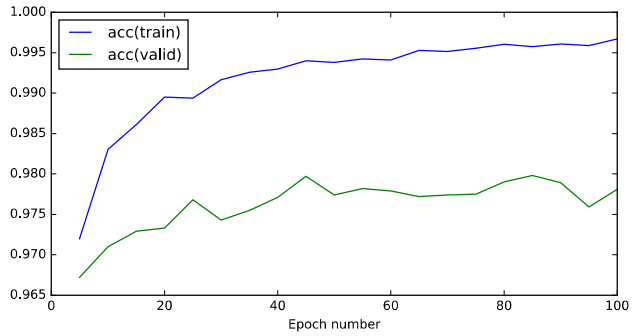
Validation error has increased a lot within 60 epochs while training accuracy has reached perfection and training error is almost zero which means that we have overfitted.

If we care for faster learning of the neural network then we might increase the learning rate and this will result to more overfitting. That's why we need to use a regularization method and here we use dropout to compare later its performance with Annealed Dropout.

Max Pooling and Dropout
pool size: 4
Momentum Learning Rule learning rate: 0.02 and coefficient: 0.9
Initial Inclusive Input Probability: 0.9
Inclusive Probability for hidden layers: 0.6



Graph 3: Training & Validation Error - Max Pooling and Dropout - pool size: 4 - Momentum Learning Rule - learning rate: 0.02 and coefficient: 0.9 - Initial Inclusive Input Probability: 0.9 - Inclusive Probability for hidden layers: 0.6



Graph 4: Training & Validation Accuracy - Max Pooling and Dropout - pool size: 4 - Momentum Learning Rule - learning rate: 0.02 and coefficient: 0.9 - Initial Inclusive Input Probability: 0.9 - Inclusive Probability for hidden layers: 0.6

Runtime: 775.867121935 seconds

`Final Testing Accuracy (percentage 0 to 1)`	`0.97809999999999919`
`Final Testing Error`	`0.10682848345968722`
`Final Training Accuracy (percentage 0 to 1)`	`0.99670000000000214`
`Final Training Error`	`0.01097044801313264`

With a much higher learning rate dropout managed to achieve good regularization and stabilized the validation error instead of leaving it to increase. However we didn't manage to achieve a high validation accuracy as before over 98%.

Note that we used a max pooling layer with pool size of 4 which was expected to yield optimal results as the previous experiments in the labs have shown. However this must be accounted as a significant factor for making the run time even a bigger number.

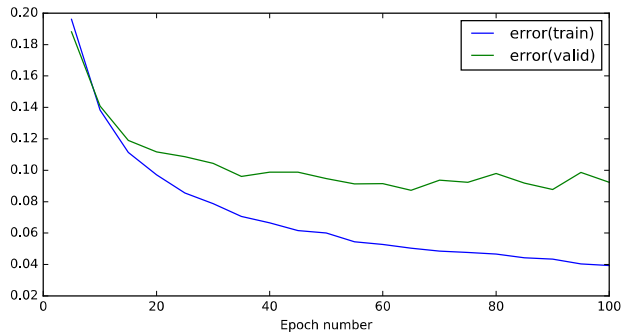
One major problem is that we have introduced two more parameters to our problem here which is choosing the best inclusive probability for the input as well choosing the best inclusive probability for the rest of the hidden layers given that we have a dropout layer in all cases.

One more important factor is that we have iterated for 100 epochs, 40 epochs more than before, but the

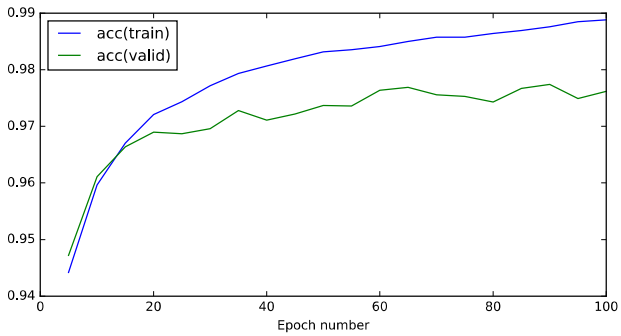
learning is still increasing in a slow rate which means we need even more epochs to achieve optimal results which makes the entire procedure very slow.

Next we are going to remove the momentum learning rule and use the RMSprop learning rule instead which proved to be better overall in coursework 1. Meaning that this adaptive learning rate was flexible enough to allow even small learning rates to be adapted quickly and efficiently.

Max Pooling and Dropout
pool size: 4
RMSProp Adaptive Learning Rule, learning rate: 1e-4, beta: 0.9
Initial Inclusive Input Probability: 0.9
Inclusive Probability for hidden layers: 0.6



Graph 5: Training and Validation Error - Max Pooling and Dropout - pool size: 4 - RMSProp Adaptive Learning Rule – learning rate: 1e-4 – beta: 0.9 - Initial Inclusive Input Probability: 0.9 - Inclusive Probability for hidden layers: 0.6



Graph 6: Training and Validation Accuracy - Max Pooling and Dropout - pool size: 4 - RMSProp Adaptive Learning Rule - learning rate: 1e-4 – beta: 0.9 - Initial Inclusive Input Probability: 0.9 - Inclusive Probability for hidden layers: 0.6

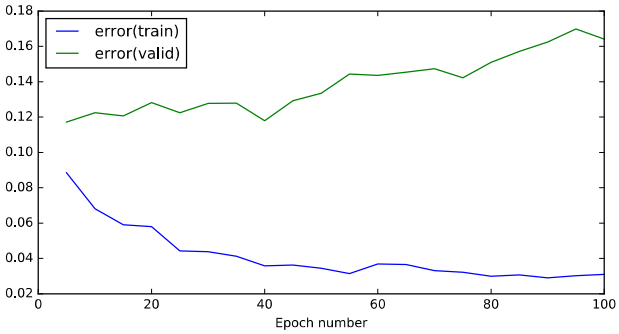
Runtime: 1281.10038996 seconds

Final Testing Accuracy (percentage 0 to 1)	0.97619999999999896
Final Testing Error	0.092352949149768368
Final Training Accuracy (percentage 0 to 1)	0.98880000000000567
Final Training Error	0.039424900776501695

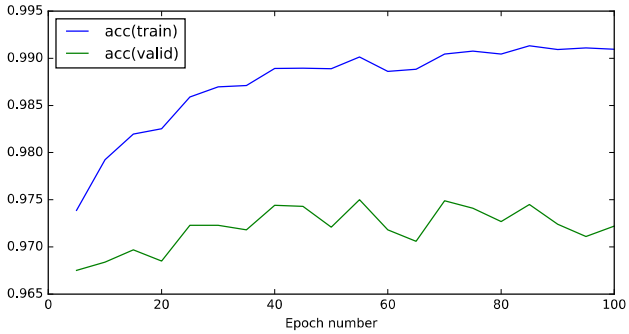
We see that adding the RMSProp did not produce any better results in comparison with the previous experiment and the execution time was increased significantly.
However we must note that the performance is more stable, the plot is more smooth than the previous experiment.

Let's repeat the experiment for max pooling size of two because it is very important to be able and run the experiments as fast as possible to be able and experiment a lot and derive some meaningful conclusions.
We will also increase the learning rate parameters of RMSProp by a factor of ten to give the model a better chance of reaching optimum validation accuracy and error within 100 epochs. This is because a max pooling size of two is expected to perform worse than the max pooling size of four.

Max Pooling and Dropout
pool size: 2
RMSProp Adaptive Learning Rule, learning rate: 1e-3, beta: 0.9
Initial Inclusive Input Probability: 0.9
Inclusive Probability for hidden layers: 0.6



Graph 7: Training and Validation Error - Max Pooling and Dropout - pool size: 2 - RMSProp Adaptive Learning Rule - learning rate: 1e-3 - beta: 0.9 - Initial Inclusive Input Probability: 0.9 - Inclusive Probability for hidden layers: 0.6



Graph 8: Training and Validation Accuracy - Max Pooling and Dropout - pool size: 2 - RMSProp Adaptive Learning Rule - learning rate: 1e-3 - beta: 0.9 - Initial Inclusive Input Probability: 0.9 - Inclusive Probability for hidden layers: 0.6

Runtime: 572.393479109 seconds

`Final Testing Accuracy (percentage 0 to 1)`	`0.97219999999999918`
`Final Testing Error`	`0.16408024330393034`
`Final Training Accuracy (percentage 0 to 1)`	`0.9909600000000045`
`Final Training Error`	`0.030960291560478217`

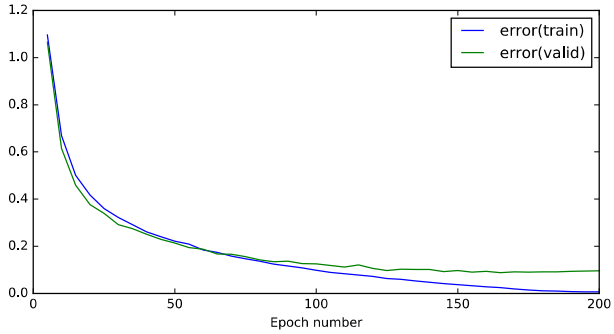
As expected with smaller max pooling size of two we have a significant increase in performance but the model does not seem flexible enough to reach the better accuracy of both training and validation set of the previous experiment where max pooling size was four.

Next we are going to use Annealed Dropout.

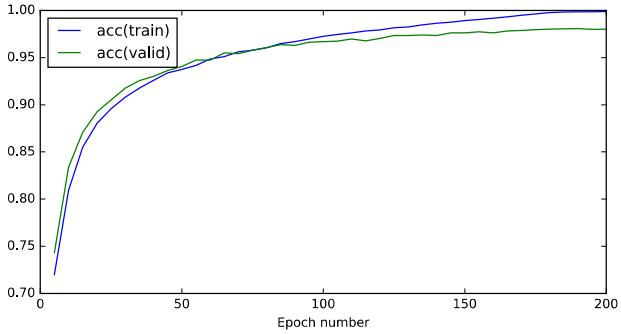
Note that Annealed dropout at first training iterations it learns simpler explanations of the data and gradually increases the capacity of the model to learn more complex explanations for all these details that cannot easily be explained.

With annealed dropout we are expecting to reach at an optimal level with minimal validation error and maximum validation accuracy. We are expecting not to have the issues of the simple dropout where the accuracy was not optimized or the optimal place required too many epochs to reach.

Starting input inclusive probability from 0.8 to 1
Starting hidden layer inclusive probability from 0.1 to 1
Percentage of epochs in the end where probability is equal to 100% constantly is 0.1
Number of epochs: 200
Max Pooling size 2



Graph 9: Training & Validation Error - Starting input inclusive probability 0.8 - Starting hidden layer inclusive probability 0.1 - Percentage of epochs in the end where probability is equal to 100% constantly is 0.1 - Max Pooling size 2



Graph 10: Training & Validation Accuracy - Starting input inclusive probability 0.8 - Starting hidden layer inclusive probability 0.1 - Percentage of epochs in the end where probability is equal to 100% constantly is 0.1 - Max Pooling size 2

Runtime: 1354.82056618 seconds

`Final Testing Accuracy (percentage 0 to 1)`	`0.98029999999999862`
`Final Testing Error`	`0.09619017831096531`
`Final Training Accuracy (percentage 0 to 1)`	`0.99876000000000054`
`Final Training Error`	`0.0061450748950512037`

Here we are training for 200 epochs which seem a lot but give the annealing this slow rate of learning to optimize the final result.

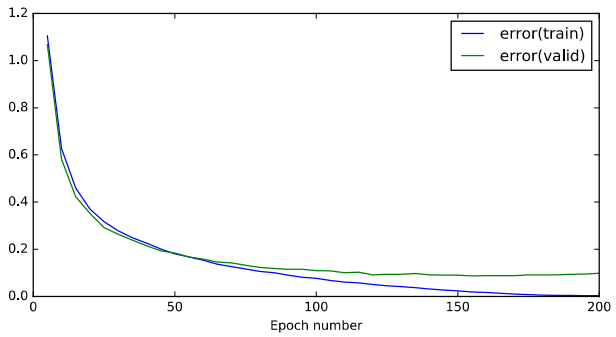
The final validation accuracy is over 98% which is an indicator of a good finally tuned model even with a max pooling of size two. Remember from previous experiments that we had to increase max pooling size in order to optimize the performance.

In addition the testing error is one of the lower testing errors in comparison to previous experiments.

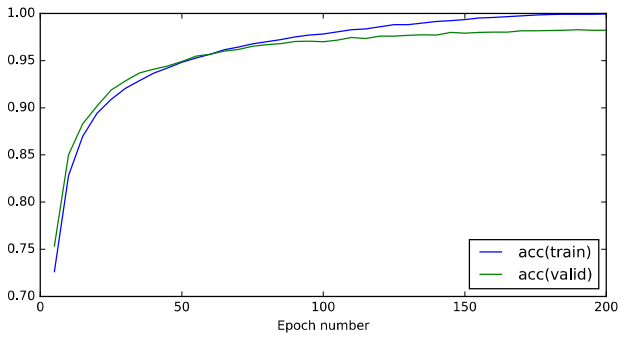
However the training running time was approximately one third of an hour which is significant.

We need to run more experiments before stating any conclusions.
Next let’s execute the same experiment but for max pooling size of 4.
If Annealed Dropout “guarantees” the best possible model we should see an increase in the testing accuracy.

Starting input inclusive probability from 0.8 to 1
Starting hidden layer inclusive probability from 0.1 to 1
Percentage of epochs in the end where probability is equal to 100% constantly: 0.1
Number of epochs 200
Max Pooling size 4



Graph 11: Training & Validation Error - Starting input inclusive probability from 0.8 to 1Starting hidden layer inclusive probability from 0.1 to 1Percentage of epochs in the end where probability is equal to 100% constantly: 0.1Number of epochs 200Max Pooling size 4



Graph 12: Training & Validation Accuracy - Starting input inclusive probability from 0.8 to 1Starting hidden layer inclusive probability from 0.1 to 1Percentage of epochs in the end where probability is equal to 100% constantly: 0.1Number of epochs 200Max Pooling size 4

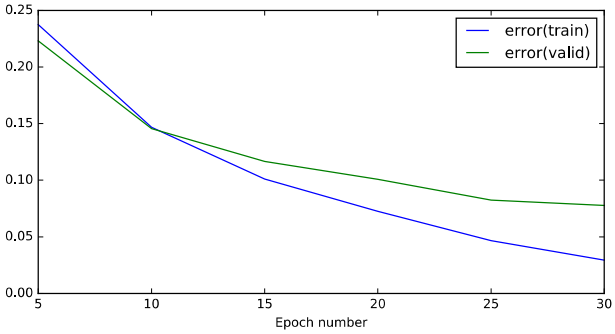
Runtime: 3312.20865393 seconds

Final Testing Accuracy (percentage 0 to 1)	0.9820999999999992
Final Testing Error	0.098154630284018191
Final Training Accuracy (percentage 0 to 1)	0.99938000000000038
Final Training Error	0.0028349326601966609

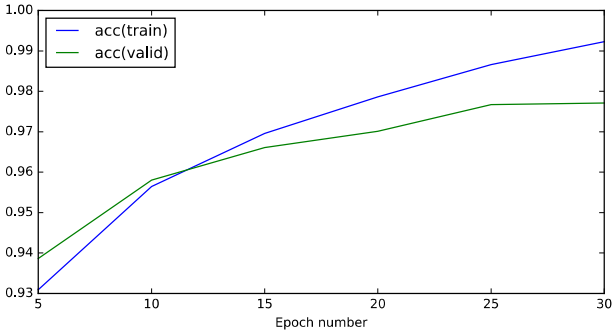
And yes here annealing dropout resulted in one the highest validation accuracies we have seen. We must note that we don't expect annealing dropout to produce remarkable results in comparison to simple dropout nor that it is expected to yield a higher performance because it does nothing special in contributing to the architecture or the complexity or the simplicity of the model. However what annealing dropout does is maximize the potential of the model in hand by avoid the convergence towards local minima.

Because Annealing Dropout is a very computationally intensive procedure it worths doing some experiments with fewer number of epochs and different initial inclusive probabilities to see if we can get the same good results but faster.

Starting input inclusive probability from 0.8 to 1
Starting hidden layer inclusive probability from 0.5 to 1
Percentage of epochs in the end where probability is equal to 1 constantly: 0.05
Number of epochs 30
Max Pooling size 4



Graph 13: Training and Validation Error - Starting input inclusive probability from 0.8 to 1 - Starting hidden layer inclusive probability from 0.5 to 1 - Percentage of epochs in the end where probability is equal to 1 constantly: 0.05 - Number of epochs 30 - Max Pooling size 4



Graph 14: Training and Validation Accuracy - Starting input inclusive probability from 0.8 to 1 - Starting hidden layer inclusive probability from 0.5 to 1 - Percentage of epochs in the end where probability is equal to 1 constantly: 0.05 - Number of epochs 30 - Max Pooling size 4

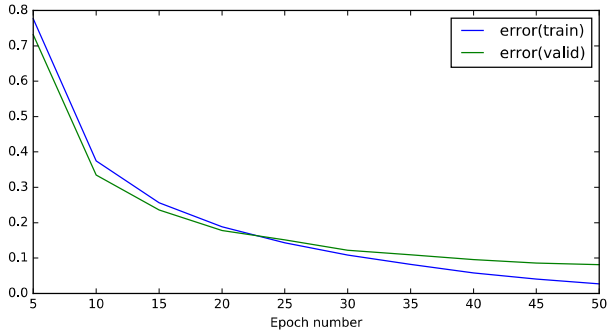
Runtime: 395.63371706

Final Testing Accuracy (percentage 0 to 1)	0.97709999999999908
Final Testing Error	0.077706504433064316
Final Training Accuracy (percentage 0 to 1)	0.99230000000000385
Final Training Error	0.029451786118968395

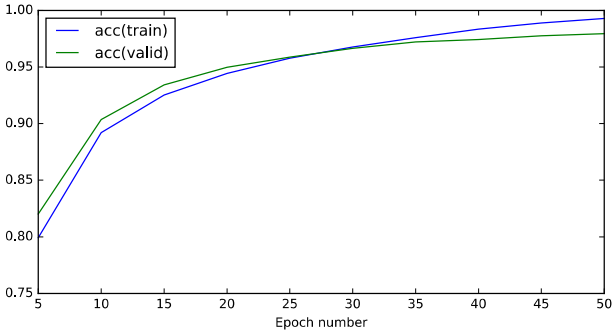
Note that we had chosen to decrease the percentage of epochs where probability is equal to 100% at the end because 30 epochs are not enough for something like that. Also the initial inclusive probability for the hidden layers was chosen to be 0.5 because going from 0.1 to 1 in 30 epochs would have been very quick for annealing to work properly.

This experiment did not yield the expected results. Seems like 30 epochs is not slow enough for the model to go from simpler to more complex representations and annealing did not gave us the best model.

Starting input inclusive probability from 0.8 to 1
Starting hidden layer inclusive probability from 0.1 to 1
Percentage of epochs in the end where probability is equal to 1 constantly: 0.05
Number of epochs 50
Max Pooling size 4



Graph 15: Training and Validation Error - Starting input inclusive probability from 0.8 to 1 - Starting hidden layer inclusive probability from 0.1 to 1 - Percentage of epochs in the end where probability is equal to 1 constantly: 0.05 - Number of epochs 50 - Max Pooling size 4



Graph 16: Training and Validation Accuracy - Starting input inclusive probability from 0.8 to 1 - Starting hidden layer inclusive probability from 0.1 to 1 - Percentage of epochs in the end where probability is equal to 1 constantly: 0.05 - Number of epochs 50 - Max Pooling size 4

`Final Testing Accuracy (percentage 0 to 1)`	`0.97949999999999904`
`Final Testing Error`	`0.081238606177604145`
`Final Training Accuracy (percentage 0 to 1)`	`0.99300000000000366`
`Final Training Error`	`0.027229031752991996`

Here we have not achieved any remarkable results in comparison to previous experiments but it is interesting to note that with 20 more epochs we were able to derive a better final model with higher validation accuracy than before. Note that here we have started from 0.1 initial inclusive probability for the hidden layers but this did not have any slowing-down or other unwanted effect.
This experiment is a good indication that we should better start from lower probabilities rather than higher ones for the hidden layer.

Conclusions

Annealing dropout seems like a simple and yet very powerful technique to optimize the model in hand in terms of classification accuracy by avoiding local minima and going slowly from simpler models, where only a few important features of the problem have been captures and it builds on top of them to end-up with a more complex model that can capture the full complexity of the input, these small details that make a difference in the final classification accuracy.

To summarize it the experiments suggest that Annealed Dropout guarantees the best possible model with the current architecture.

It also works without requiring any pretraining to start-off from a good place rather than what the Glorot initialization of the weights might give you. This is because from where it starts makes little difference how we choose the initial weights. These initial simpler models are going to quickly converge to capture the more basic features of our problem. In other words at the beginning we have clearly underfitted. The only thing to avoid is, as always not set too large weights in the beginning to avoid saturation effects.