Machine Learning Practical
Assignment 1

Siyuan Zhao

October 26, 2016

1 Learning rate schedules

In this section, the exponential time-dependent learning rate schedule is tested to evaluate its influences on training. The equation is given below.

$$\eta(t) = \eta_0 \exp(-t/r) \tag{1}$$

1.1 Methods and Implementation

In this experiment, a new class called **TimeDependentLearningRateScheduler** is developed in the **mlp.schedulers.py**, in the **__init__** method, two parameters called **init_learning_rate** and **free_parameter** is defined. They represents η_0 and r in equation (1).

In the **update_learning_rule** method, the **learning_rate** attribute of a learning rule object is updated for each epoch.

In the notebook **Coursework_1**, the method **train_model_and_plot_stats_with_dynamic_LR** defines the training processing applying exponential learning rates and gives the results of error and accuracy for both training and error data sets.

1.2 Results

The performances based on different free parameters are given in Figure.1.

The performances based on different initial learning rates are given in Figure.2.

To compare with the constant learning rate, the results of constant learning rates when the learning rate is set to 0.1 is given in Figure 3.

r	error(train)	error(valid)	acc(train)	acc(valid)
10	1.61e-01	1.61e-01	9.54e-01	9.55e-01
100	1.70e-02	8.32e-02	9.98e-01	9.76e-01
1000	6.68e-03	8.65e-02	1.00e+00	9.77e-01
1e+04	5.84e-03	8.76e-02	1.00e+00	9.77e-01
5e+04	5.78e-03	8.77e-02	1.00e+00	9.77e-01

Table 1: Performance of different r with initial learning rate = 0.1

initial learning rate	error(train)	error(valid)	acc(train)	acc(valid)
0.01	1.67e-01	1.66e-01	9.52e-01	9.54e-01
0.1	5.78e-03	8.77e-02	1.00e+00	9.77e-01
0.5	3.27e-04	1.04e-01	1.00e+00	9.79e-01
1	1.34e-04	1.21e-01	1.00e+00	9.78e-01
10	1.89e+00	1.88e+00	1.92e-01	1.94e-01

Table 2: Performance of different initial learning rates with $r = 10$

learning rate type	error(train)	error(valid)	acc(train)	acc(valid)
const LR = 0.1	5.64e-03	8.92e-02	1.00e+00	9.77e-01
TD init_LR =0.1 & r = 5e+04	5.78e-03	8.77e-02	1.00e+00	9.77e-01

Table 3: Comparison Performance between constant and time-dependent learning rates

1.3 Discussion and Conclusion

1.3.1 Effects of free parameter r

Figure 1 indicates that the free parameter r determines how quickly the learning rate decays. larger free parameter converges faster, with higher accuracy. However, smaller free parameter has slower converge speed since the learning rate in each epoch is very small. In summary, larger r results in a better performance and faster convergence.

1.3.2 Effects of initial learning rate

From Table 2, the accuracy is increasing with the increase of initial learning rate except for extremely values such as 10 because the system becomes more noisy suggesting that the the gradient descent step size is too large. However, the validation error starts to increases when learning rates is over 0.5. This suggests that the model is overfitting when the initial learning rate is too large.

1.3.3 Comparison with constant learning rate

From Table 3, we can see that the final validation accuracy is similar. However, the time dependent learning rate converges faster than the baseline. Moreover, it has smaller error in validation sets while larger error in training set which means the gap of error between the training and validation sets is smaller in time-dependent learning rates which indicates that the time-dependent learning rats has better performance.

mom_coeff	error(train)	error(valid)	acc(train)	acc(valid)
0.1	4.47e-03	8.90e-02	1.00	0.9775
0.3	2.56e-03	9.25e-02	1.00	0.9786
0.5	1.34e-03	9.64e-02	1.00	0.9786
0.9	1.28e-04	1.13e-01	1.00	0.9787

Table 4: Performance of different momentum coefficients

2 Momentum learning rule

In this section, a gradient descent learning rule with momentum is investigated.F Firstly, the basic gradient descent learning is compared to the momentum learning rule for several values of the momentum coefficient. Secondly, analogous to the time-dependent learning rate, the time-dependent momentum coefficient is applied and investigated.

2.1 Methods and Implementation

2.1.1 First Task

In the first task, a for loop is used to test different momentum coefficient. since the momentum coefficient should be in the range of [0 1], so the 4 figures is chose to compare.

2.1.2 Second Task

In the second task, a new schedule is developed according to the following equation.

$$\alpha(t) = \alpha_\infty(1 - \frac{\gamma}{t+\tau}) \tag{2}$$

It is realised in **MomentumCoefficientScheduler** in **mlp.schedulers** with the same interface as the learning rate scheduler. It has three attributes **asy_mo_coeff, tau, gamma**, which represent α_∞, τ and γ. Also, one method **update_learning_rule** which implements the above equation. In the notebook, a new method called **train_model_and_plot_stats_with_dynamic_mom_coeff** is developed to train the model based on the momentum coefficient scheduler.

The range of τ to investigate should be limited. If it is much larger than the γ and t, the value of momentum coefficient rarely changes which means it becomes another forms of constant momentum coefficient. the value for both γ and τ should not be chosen too small or too large since t is from 1 to 100.

2.2 Results

2.2.1 First Task

The results is shown in Figure 4. The final error and accuracy rate is shown in the Table 4. It indicates that a model with larger momentum coefficient will converge faster than that with smaller momentum coefficient.

2.2.2 Second Task

In these parts, three attributes α_∞, τ and γ are investigated for different values. The results is given by Figure 5. 6. and 7. The final performance is shown in Table 5. 6. and 7.

asy_mom_coeff	error(train)	error(valid)	acc(train)	acc(valid)
0.1	4.47e-03	8.90e-02	1.00	0.9775
0.3	2.56e-03	9.25e-02	1.00	0.9786
0.5	1.34e-03	9.64e-02	1.00	0.9786
0.7	5.83-04	1.02e-01	1.00	0.9783
0.9	1.28e-04	1.13-e01	1.00	0.9787

Table 5: Performance of different asymptotic momentum coefficients with $\gamma = 1$, $tau = 2$

τ	error(train)	error(valid)	acc(train)	acc(valid)
1.0	1.71e-04	1.14e-01	1.00	0.9787
2.0	1.68e-04	1.10e-01	1.00	0.9804
5.0	1.65e-04	1.08e-01	1.00	0.9801
10.0	1.59e-04	1.10e-01	1.00	0.9791
100.0	1.39e-04	1.13-e01	1.00	0.9795

Table 6: Performance of different τ with $\gamma = 1.0$ and asy_mom_coeff $= 0.9$

γ	error(train)	error(valid)	acc(train)	acc(valid)
0.2	1.37e-04	1.14e-01	1.00	0.9774
0.5	1.50e-04	1.16e-01	1.00	0.9785
0.7	1.59e-04	1.13e-01	1.00	0.9797
0.9	1.67e-04	1.13e-01	1.00	0.9793

Table 7: Performance of different γ with $\tau = 1.0$ and asy_mom_coeff $= 0.9$

2.3 Discussion and Conclusion

2.3.1 First task

Figure 4 indicates that larger momentum coefficient results in a faster convergences and higher speed of learning. Table 4 indicates that a larger momentum coefficient has larger validation accuracy and smaller train error, which means a better performance. However, the validation error increase when the momentum coefficients increases which indicates overfitting problem.

2.3.2 Second task

From Figure 5, it can be seen that larger α_∞ results in a faster convergence and its final validation accuracy is similar to smaller α_∞. Table 5 indicates that the validation error begins to increase when α_∞ increase while the final training error is decreasing. It indicates overfitting problem.

When γ is fixed, with the increase of τ, the final training error decreases in the condition that τ is close to γ. The higher validation accuracy is achieved when $\tau = 2.0$. Then, with the increase of τ the validation error increases. In general, The model is not quite sensitive to this parameter.

when τ is fixed, the system performance is better as γ approaching to τ.. When γ is very small, the momentum at first epoch is close to asymptotic which is similar to a constant momentum coefficient learning rule. Moreover, the speed of learning rarely changes.

In conclusion, using a variable momentum coefficient improves the performance over a constant momentum coefficient baseline according to Table 4. 5. 6. and 7.

Learing Rate	error(train)	error(valid)	acc(train)	acc(valid)
1e-03	3.05e-01	2.83e-01	0.9163	0.9231
1e-02	5.12e-02	9.26e-02	0.9863	0.9718
5e-02	6.64e-04	9.59e-02	1.00	0.9799
1e-01	1.85e-04	1.18e-01	1.00	0.9780
5e-01	8.47e-02	2.72e-01	0.9834	0.9453

Table 8: Performance of different learning rates using AdaGrad Learning rule

3 Adaptive learning rules

In this section, the adaptive learning rules is developed especially the **AdaGrad** and the **RMSProp** which are implemented in the **mlp/learning_rules.py** module.

3.1 Methods and Implementation

AdaGrad methods is implemented in the method **AdagradLearningRule**. The initial attributes is learning rate and a small constant ϵ 10^{-8}. The update procedure is given by the following equations:

$$S_i(0) = 0 \tag{3}$$

$$S_i(t) = S_i(t-1) + D_i(t)^2 \tag{4}$$

$$\Delta w_i(t) = \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} D_i(t) \tag{5}$$

S represents sum squared gradient and is represented as **caches** in the method **AdagradLearningRule**. $D_i(t)$ is the gradient of error function E with respect to a weight W_i at update time t. $\Delta w_i(t)$ is the weight change for each echo which is represented as **param** in the code.

Similarly, RMSpropLearingRule is implemented in the method **RMSpropLearningRule** but with one more parameter called decay rate and the implementation of updates is based on the following equations. The main difference is that the RMSProp method replaces the sum by a moving average for S.

$$S_i(t) = \beta S_i(t-1) + (1-\beta) D_i(t)^2 \tag{6}$$

$$\Delta w_i(t) = \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} D_i(t) \tag{7}$$

β is represented as *decay_rate* in the method **RMSpropLearningRule**, which is normally 0.9 as suggested.

3.2 Results

The investigation for AdaGrad learning rules is conducted by selecting different values of learning rate. The results is shown in Figure 8. and Table 8.

Similarly, the effects of learning rates for RMSprop learning rule is given in Figure 9 and Table 9. Additionally, an experiment was conducted to investigate how sensitive to system is to the decay rate β. The results for different β is shown in Figure 10 and Table 10.

Learing Rate	error(train)	error(valid)	acc(train)	acc(valid)
1e-04	4.13e-02	9.23e-02	0.9890	0.9734
1e-03	7.41e-08	3.14e-01	1.0000	0.9757
5e-03	1.62e-04	5.04e-01	1.0000	0.9750
1e-02	6.38e-03	4.90e-01	0.9990	0.9760
5e-02	2.31e-01	3.65e-01	0.9609	0.9520
1e-01	4.16e-01	4.72e-01	0.9278	0.9270

Table 9: Performance of different learning rates using RMSprop Learning rule

decay rate	error(train)	error(valid)	acc(train)	acc(valid)
0.1	2.69e-02	7.17-02	0.9969	0.9643
0.2	1.06e-02	6.07e-01	0.9986	0.9683
0.5	5.57e-05	6.41e-01	1.0000	0.9697
0.9	1.62e-04	5.04e-01	1.0000	0.9750
0.99	1.19e-03	2.72e-01	0.9996	0.9767

Table 10: Performance of different decay rates using RMSprop learning rule

3.3 Discussion and Conclusion

3.3.1 AdaGrad

For AdaGrad learning rules, except for large learning rules (over 0.5) or small learning rules (below 0.01) the speed of convergence is the highest with a good performance. Moreover, the validation error starts to increase while the training error is decrease when learning rate increases which suggests the model is overfitting to the data. one possible of drawback could be that the learning stops too early

3.3.2 RMSprop

The RMSprop learning rules is unstable during converges suggesting a large step size. Although providing with good speed of convergence, the model using RMSprop learning rules reaches its convergence very early but is quite unstable during the later training. Also, the model is not overfitting when the learning rate is $1e-4$. For higher learning rate, the model starts to overfit the training data. Interestingly, when leaning rate = 0.05, the trend of validation error follows the training error. However, the system is quite unstable. This is probably because of the large step size prevents the model from overfitting.

Increasing decay rates results in a better final accuracy on validation. However, the training error increases after decay rate increases over 0.5. However, in general, the system is not very sensitive to decay rate. From the plots, the rends are similar for different values of decay rates. In summary, a high decay rate is suggested to achieve better performance.

3.3.3 Comparison

The comparison of performance in terms speed of convergence and the final error / accuracy on both training and validation data sets. The Table 11 shows the results including final error

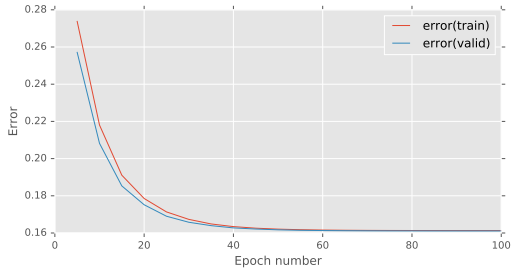
learning rules	free params	error(train)	error(valid)	acc(train)	acc(valid)	run time/epoch
Basic	LR = 0.1	5.64e-03	8.92e-02	1.00	9.77e-01	1.10s
Momentum	LR 0.1, Mom = 0.9	1.28e-04	1.13e-01	1.00	0.9787	1.26s
AdaGrad	LR = 0.1, β= 0.9	1.85e-04	1.18e-01	1.00	0.9780	2.09s
RMSprop	LR = 5e-03	1.62e-04	5.04e-01	1.00	0.9750	2.05s

Table 11: Overall Performance of different learning rules

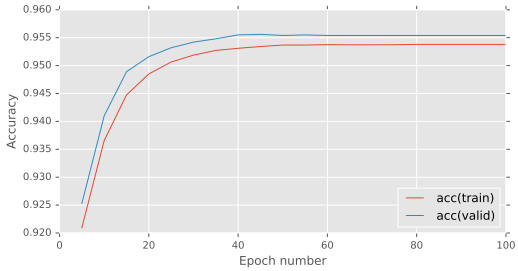
and classification accuracy for both training and validation data sets, run time per epoch, free parameters set for different learning rules, where LR represents learning rates, Mom represents momentum coefficient and For AdaGrad, RMSprop, basic gradient descent and momentum learning rules. The speed of convergence from high to low is: AdaGrad, RMSprop, momentum learning and basic gradient descent.

For the training error, the adaptive learning rules have better results. However, they are not as good as basic gradient descent learning rules in terms of validation error. This suggests a overfitting problem. Early stop may be a solution to overcome overfitting. All of four models have similar results in accuracy. However, the final accuracy of RMSprop is not the best results during the entire epochs probability because of its instability.

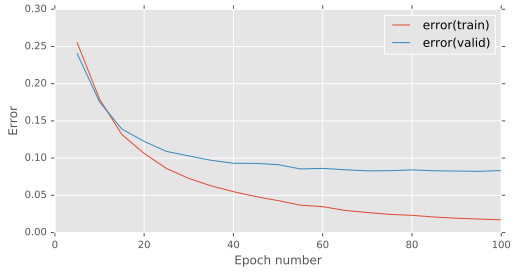
In terms of run time per epoch, applying adaptive learning rules is more time-consuming than the basic gradient descent or momentum learning rules. It should be cared to choose adaptive learning rules if the computational resources is limited.



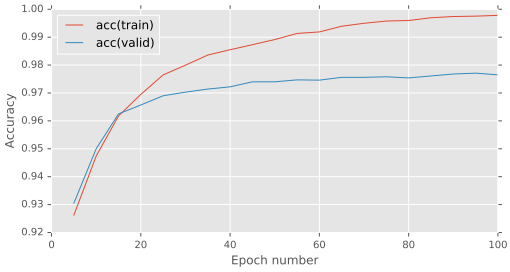
(a) error with $r = 10$



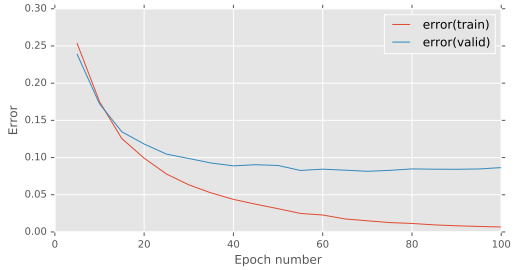
(b) accuracy with $r = 10$



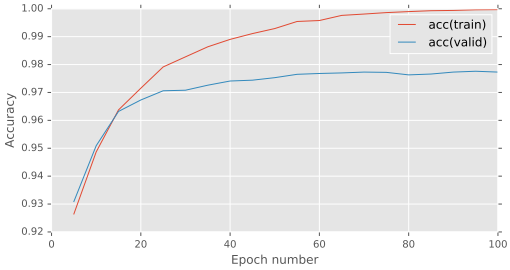
(c) error with $r = 100$



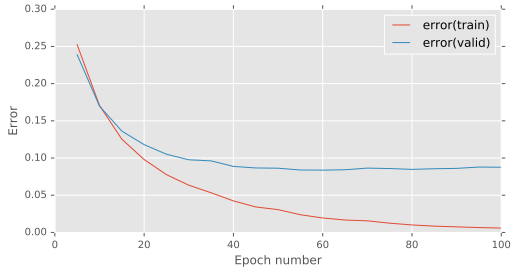
(d) accuracy with $r = 100$



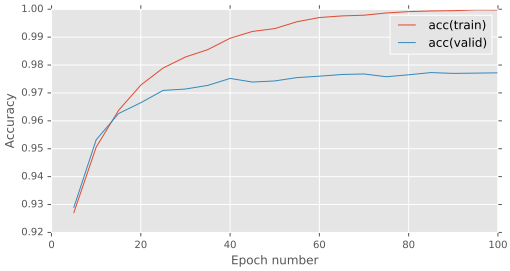
(e) error with $r = 1000$



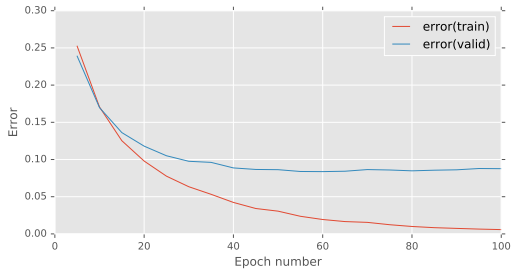
(f) accuracy with $r = 1000$



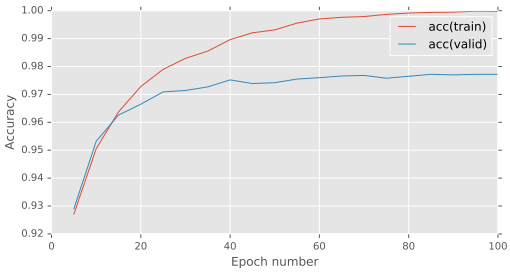
(g) error with $r = 10000$



(h) accuracy with $r = 10000$

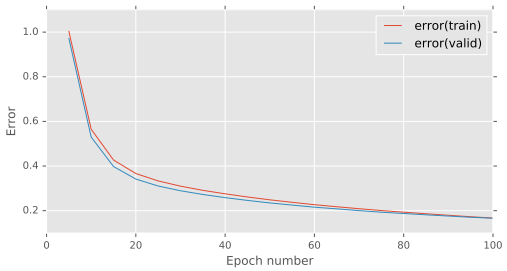


(i) error with $r = 50000$

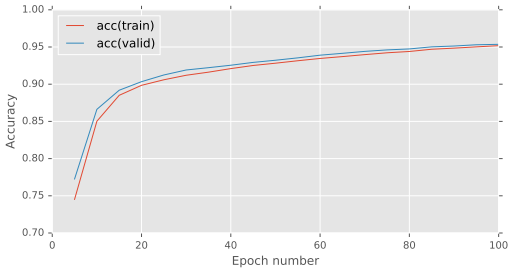


(j) accuracy with $r = 50000$

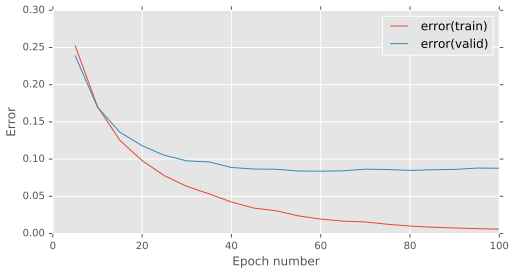
Figure 1: Errors and accuracies with different r when initial learning rate = 0.1



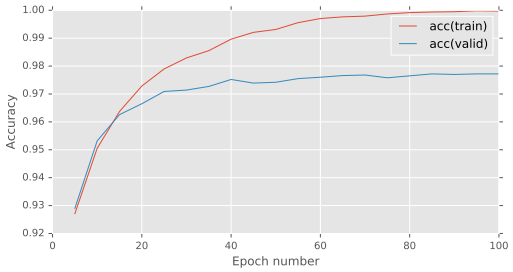
(a) error with initial learning rate = 0.01



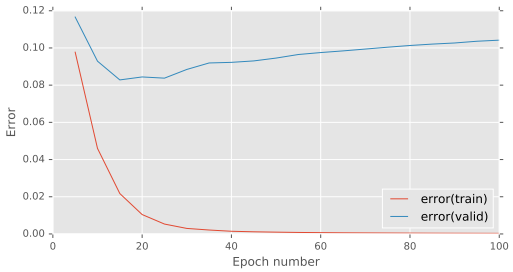
(b) accuracy with initial learning rate = 0.01



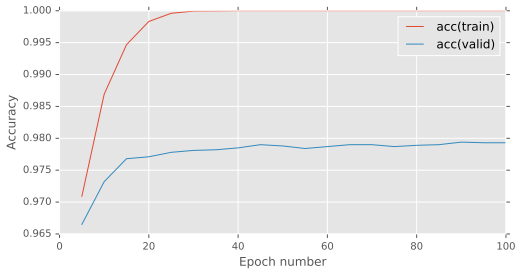
(c) error with initial learning rate = 0.1



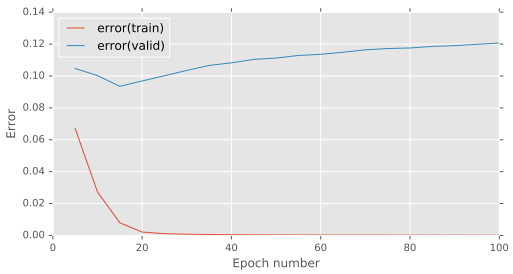
(d) accuracy with initial learning rate = 0.1



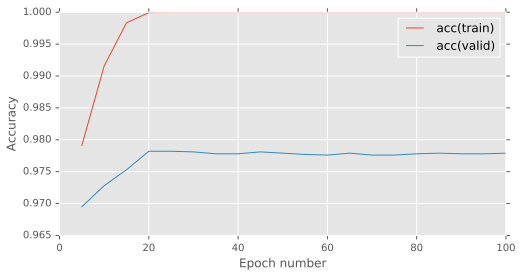
(e) error with initial learning rate = 0.5



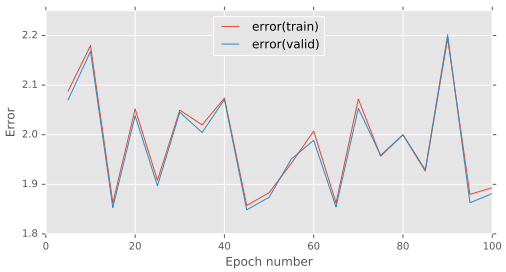
(f) accuracy with initial learning rate = 0.5



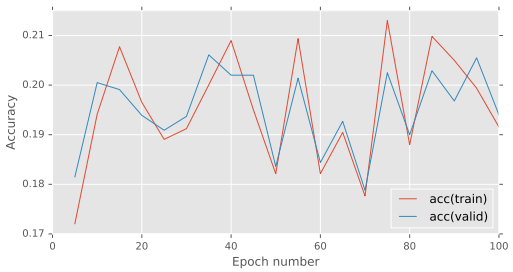
(g) error with initial learning rate = 1



(h) accuracy with initial learning rate = 1

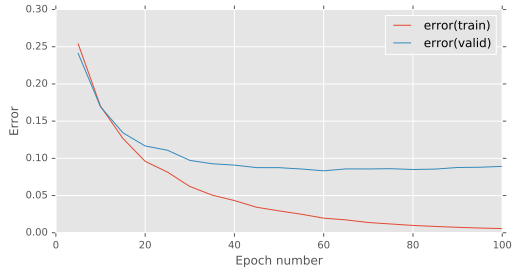


(i) error with initial learning rate = 10

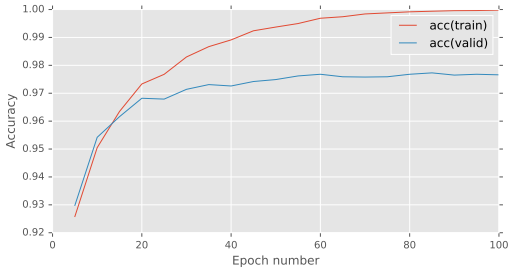


(j) accuracy with initial learning rate = 10

Figure 2: Error and accuracy with different initial learning rates when $r = 5e + 04$

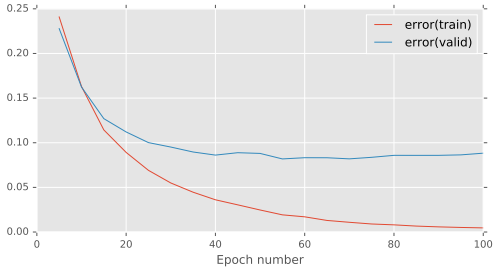


(a) error with constant learning rate = 0.1

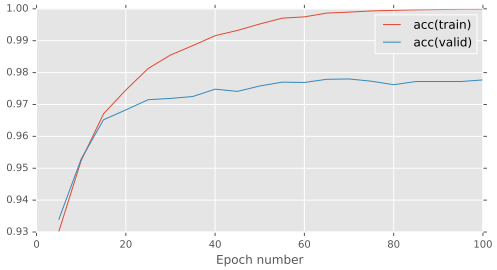


(b) accuracy with initial learning rate = 0.1

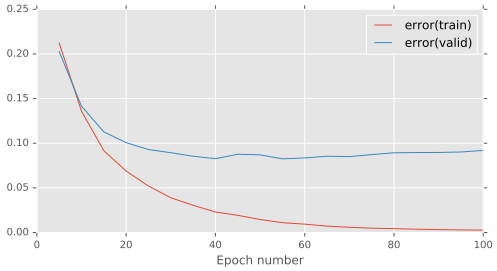
Figure 3: Error and accuracy with learning rates = 0.1 using gradient descent learning rule



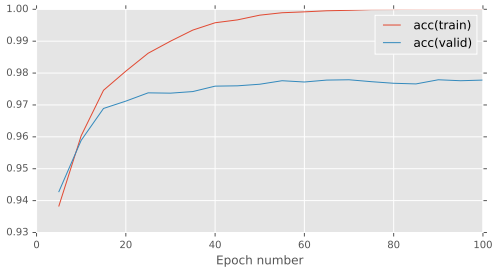
(a) error with momentum coefficient = 0.1



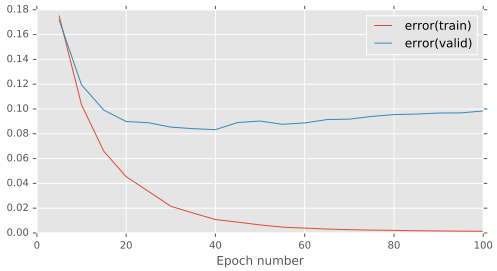
(b) accuracy with momentum coefficient = 0.1



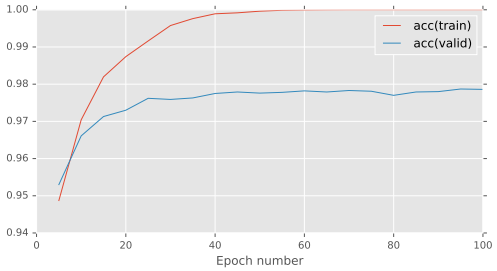
(c) error with momentum coefficient = 0.3



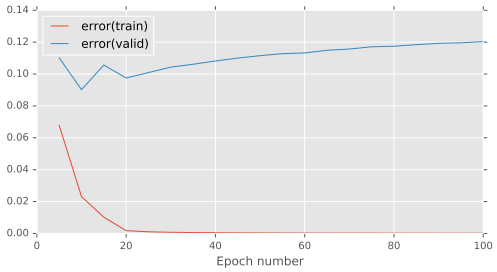
(d) accuracy with momentum coefficient = 0.3



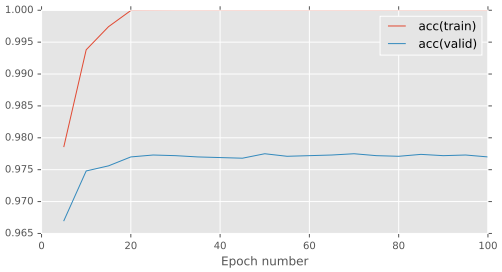
(e) error with momentum coefficient = 0.5



(f) accuracy with momentum coefficient = 0.5

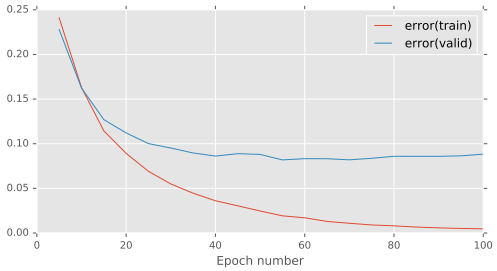


(g) error with momentum coefficient = 0.9

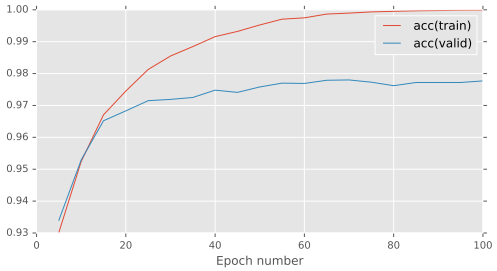


(h) accuracy with momentum coefficient = 0.9

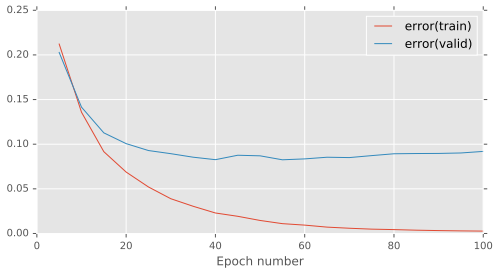
Figure 4: Error and accuracy with different momentum coefficients using momentum learning rule



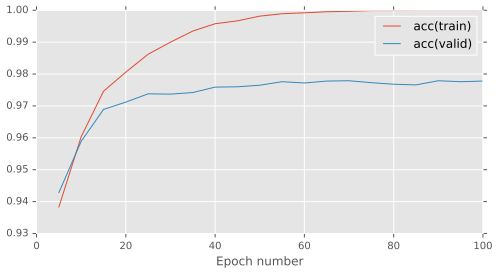
(a) error with asym_mom_coeff = 0.1



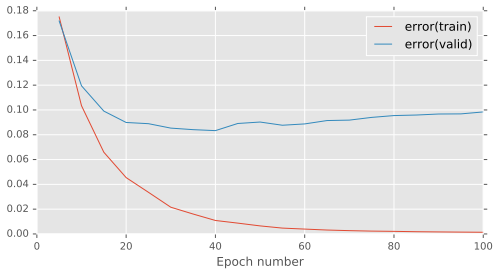
(b) accuracy with asym_mom_coeff = 0.1



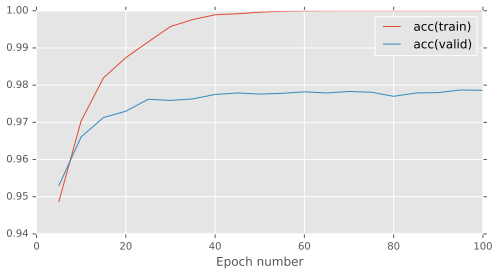
(c) error with asym_mom_coeff = 0.3



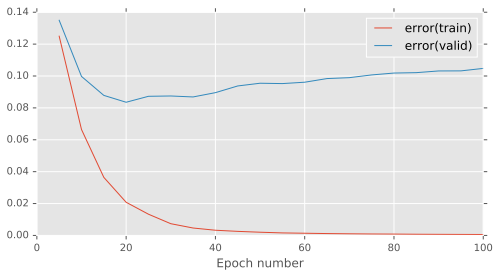
(d) accuracy with asym_mom_coeff = 0.3



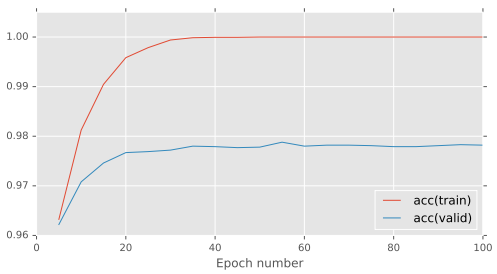
(e) error with asym_mom_coeff = 0.5



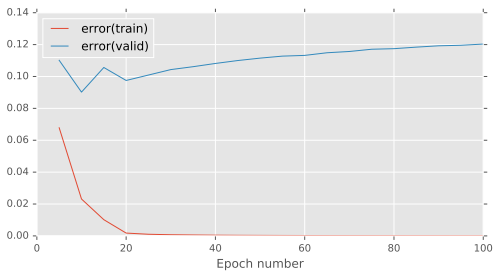
(f) accuracy with asym_mom_coeff = 0.5



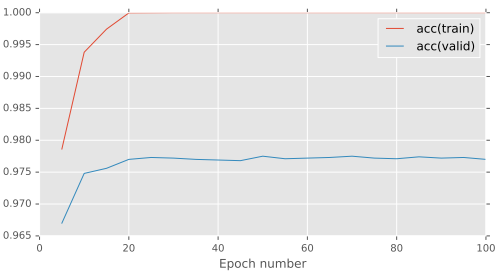
(g) error with asym_mom_coeff = 0.7



(h) accuracy with asym_mom_coeff = 0.7

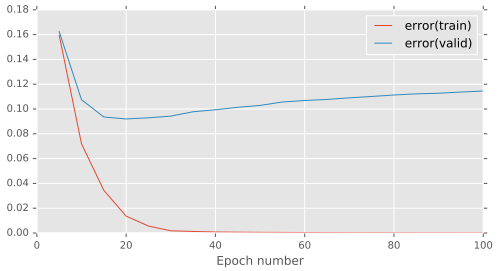


(i) error with asym_mom_coeff = 0.9

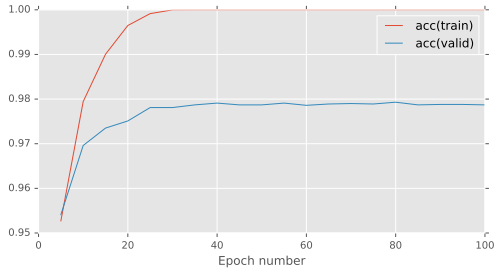


(j) accuracy with asym_mom_coeff = 0.9

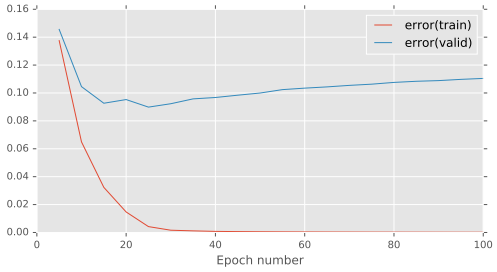
Figure 5: Error and accuracy with several asymptotic momentum coefficients using momentum coefficient scheduler



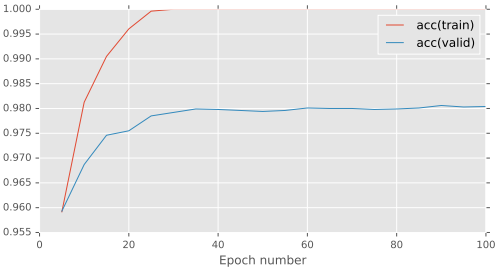
(a) error with $\tau = 1.0$



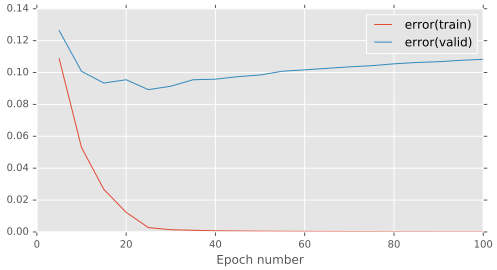
(b) accuracy with $\tau = 1.0$



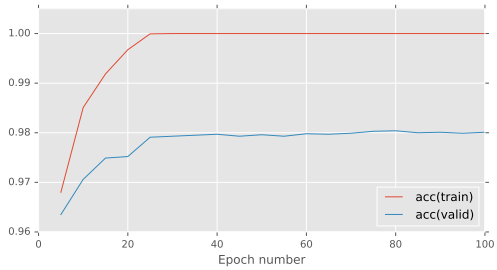
(c) error with $\tau = 2.0$



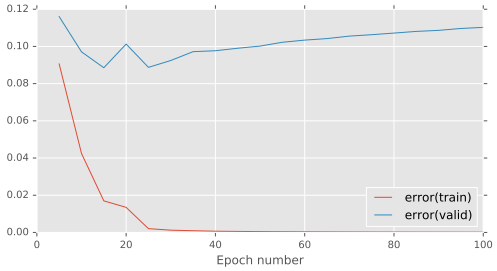
(d) accuracy with $\tau = 2.0$



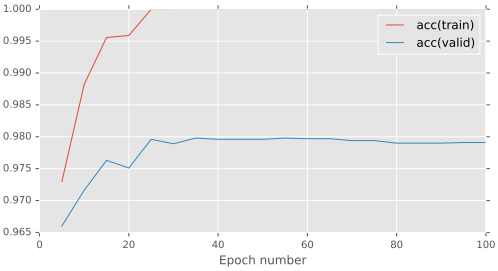
(e) error with $\tau = 5.0$



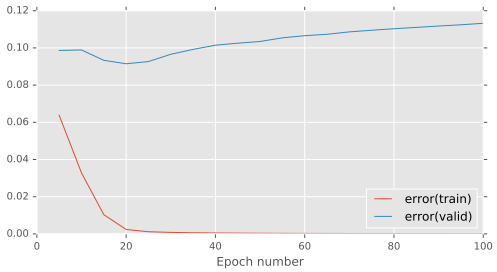
(f) accuracy with $\tau = 5.0$



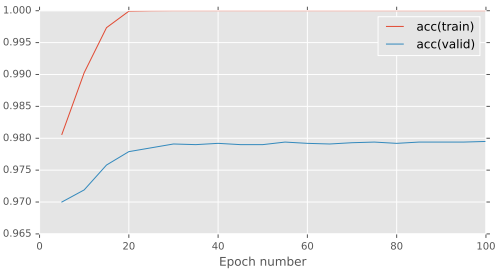
(g) error with $\tau = 10.0$



(h) accuracy with $\tau = 10.0$

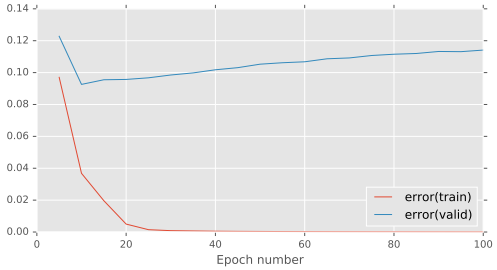


(i) error with $\tau = 100.0$

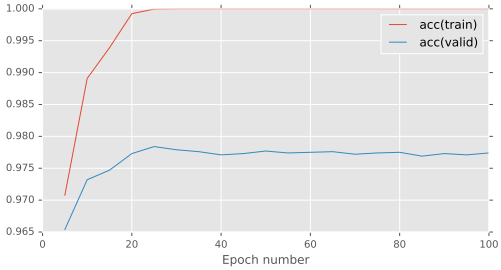


(j) accuracy with $\tau = 100.0$

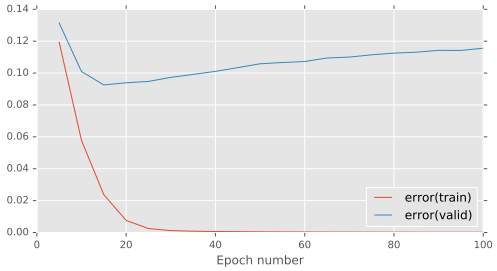
Figure 6: Error and accuracy with several τ when $\gamma = 1$ using variant momentum coefficient



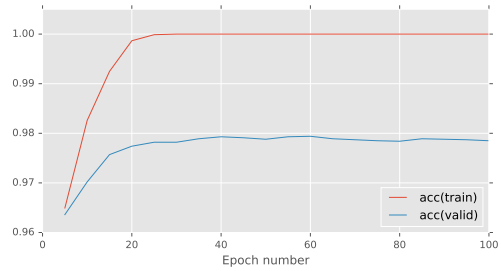
(a) error with $\gamma = 0.2$



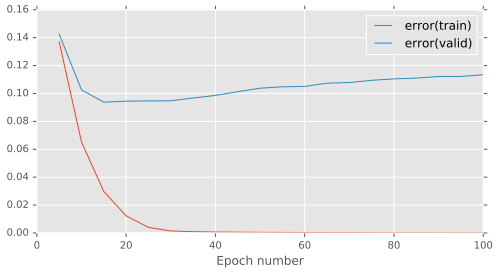
(b) accuracy with $\gamma = 0.2$



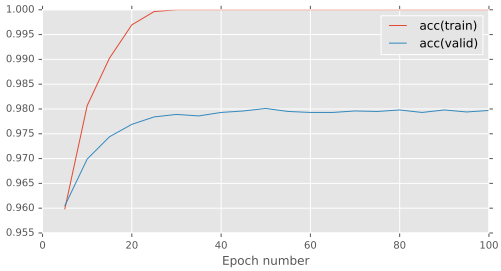
(c) error with $\gamma = 0.5$



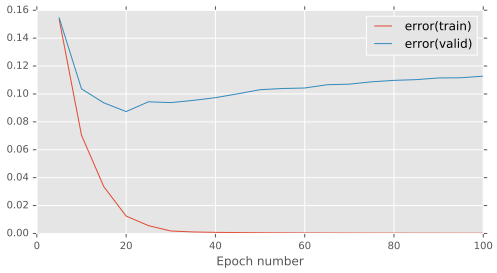
(d) accuracy with $\gamma = 0.5$



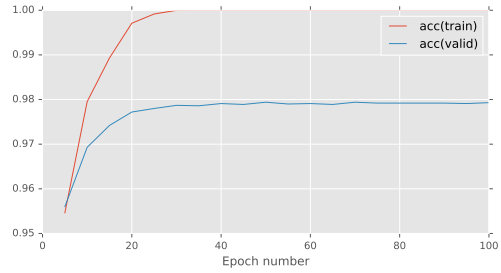
(e) error with $\gamma = 0.7$



(f) accuracy with $\gamma = 0.7$

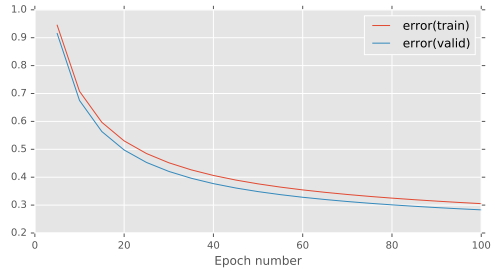


(g) error with $\gamma = 0.9$

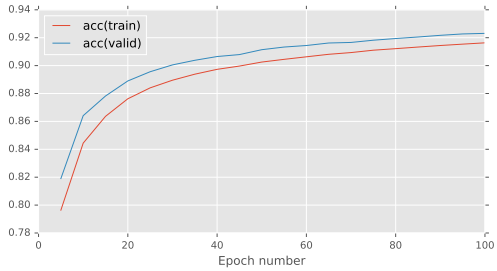


(h) accuracy with $\gamma = 0.9$

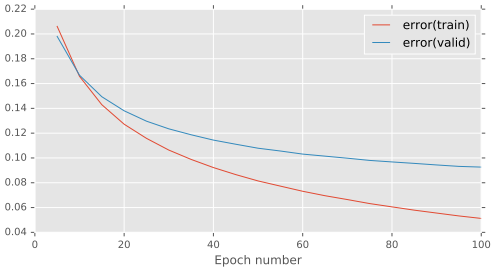
Figure 7: Error and accuracy with several γ when $\tau = 1$ using variant momentum coefficient



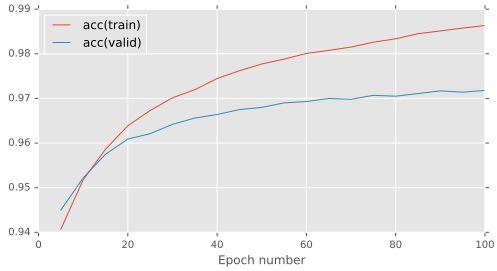
(a) error for Adgrad with learning rate = 0.001



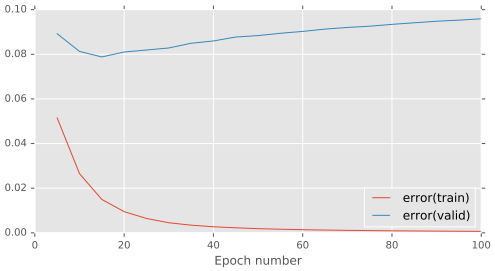
(b) accuracy for Adgrad with learning rate = 0.001



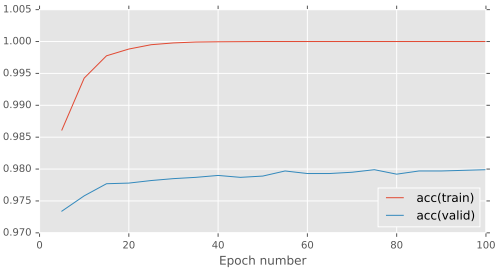
(c) error for Adgrad with learning rate = 0.01



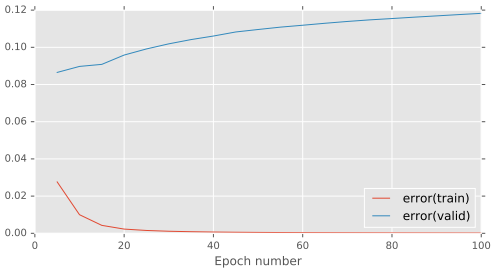
(d) accuracy for Adgrad with learning rate = 0.01



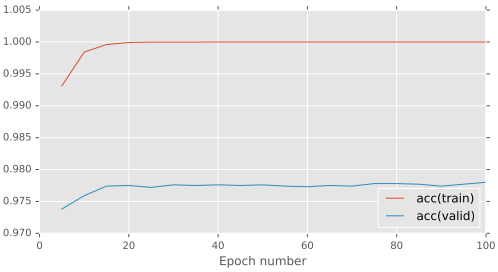
(e) error for Adgrad with learning rate = 0.05



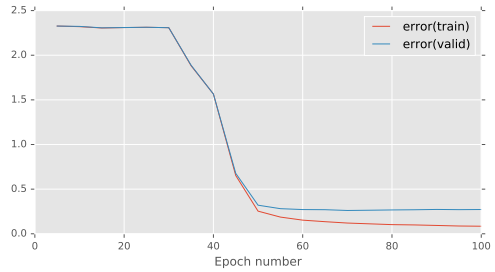
(f) accuracy for Adgrad with learning rate = 0.05



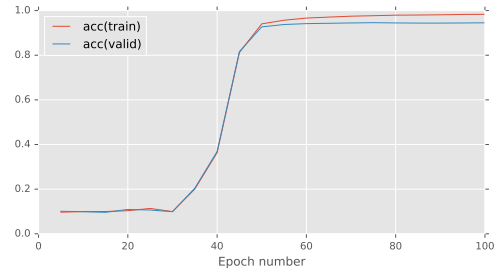
(g) error for Adgrad with learning rate = 0.1



(h) accuracy for Adgrad with learning rate = 0.1

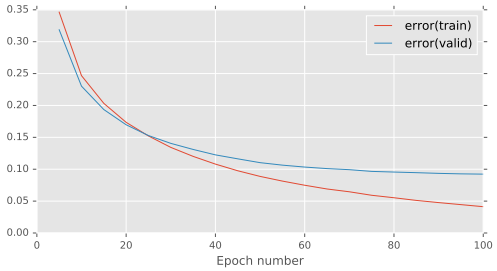


(i) error for Adgrad with learning rate = 0.5

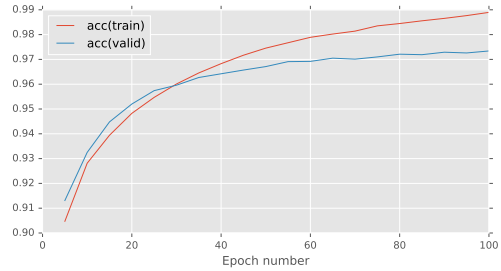


(j) accuracy for Adgrad with learning rate = 0.5

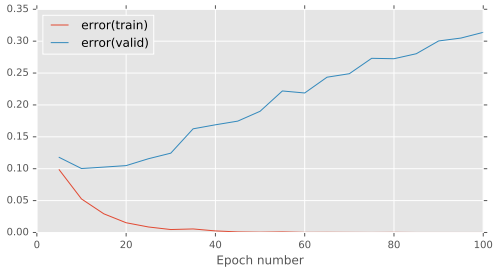
Figure 8: Error and accuracy using AdaGrad learning rules with several learning rates



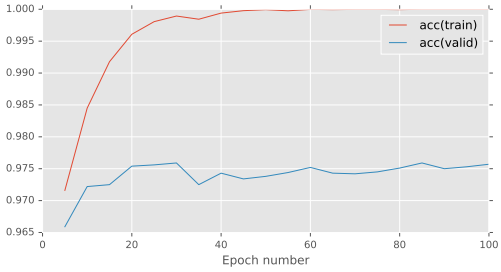
(a) error for RMSprop with LR = 0.0001



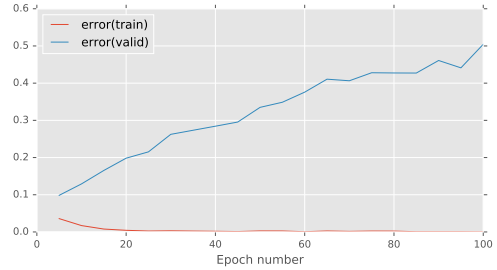
(b) accuracy for RMSprop with LR = 0.0001



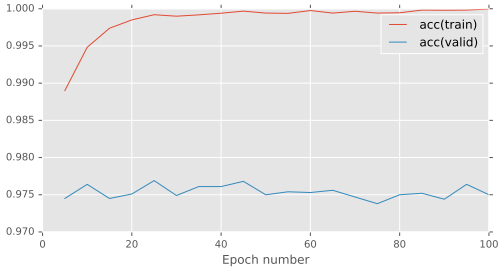
(c) error for RMSprop with LR = 0.001



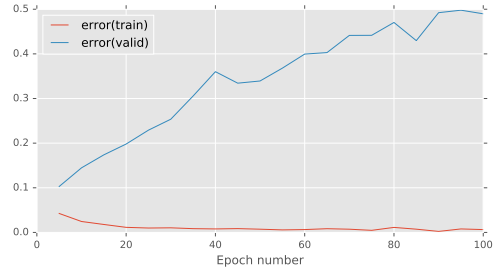
(d) accuracy for RMSprop with LR = 0.001



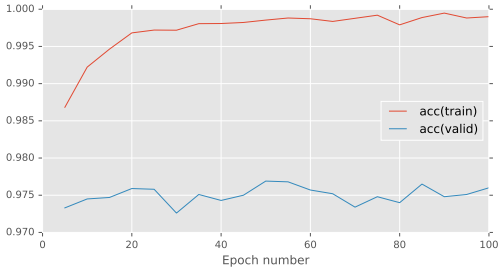
(e) error for RMSprop with LR = 0.005



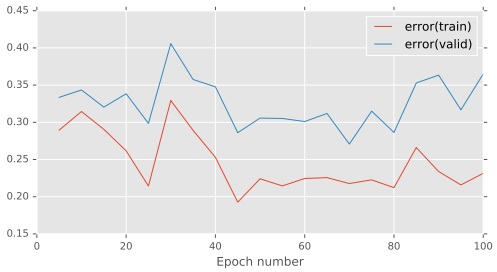
(f) accuracy for RMSprop with LR = 0.005



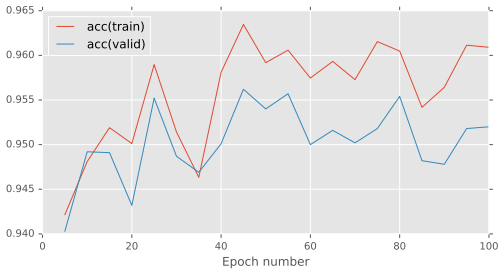
(g) error for RMSprop with LR = 0.01



(h) accuracy for RMSprop with LR = 0.01

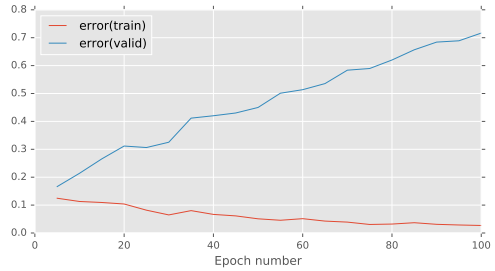


(i) error for RMSprop with LR = 0.05

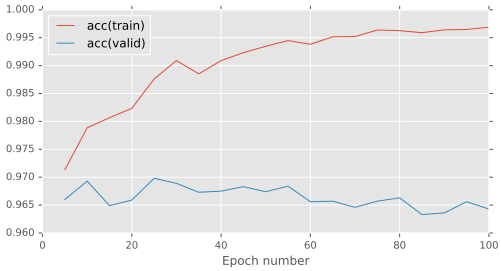


(j) accuracy for RMSprop with LR = 0.05

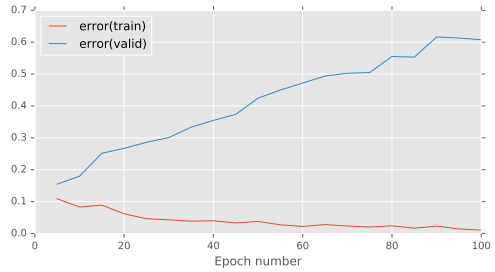
Figure 9: Error and accuracy using RMSprop learning rules with several learning rates



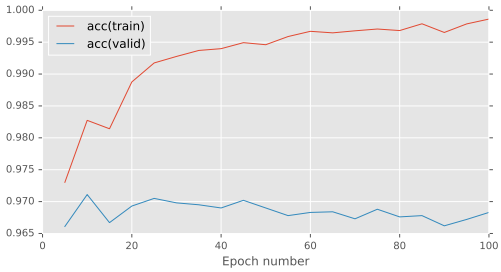
(a) error for RMSprop with decay rate = 0.1



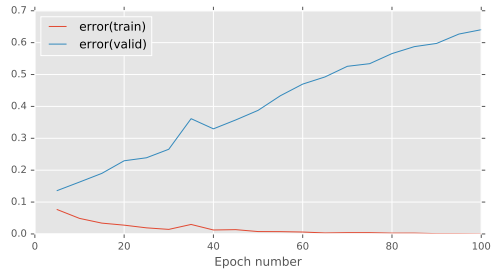
(b) accuracy for RMSprop with decay rate = 0.1



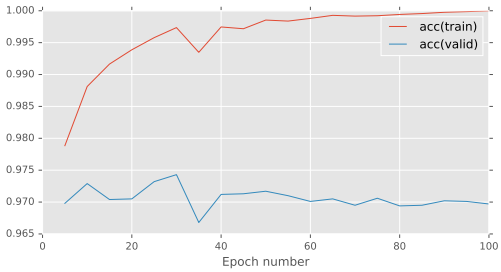
(c) error for RMSprop with decay rate = 0.2



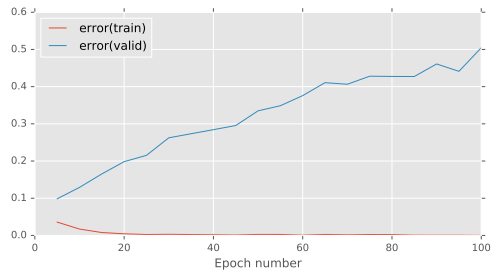
(d) accuracy for RMSprop with decay rate = 0.2



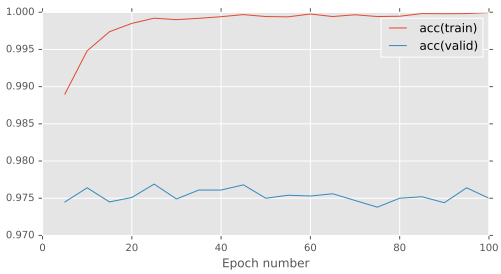
(e) error for RMSprop with decay rate = 0.5



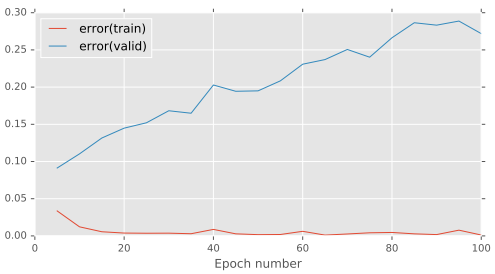
(f) accuracy for RMSprop with decay rate = 0.5



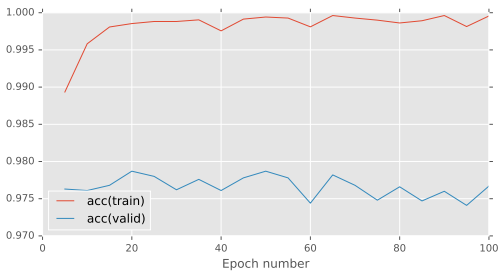
(g) error for RMSprop with decay rate = 0.9



(h) accuracy for RMSprop with decay rate = 0.9



(i) error for RMSprop with decay rate = 0.99



(j) accuracy for RMSprop with decay rate = 0.99

Figure 10: Error and accuracy using RMSprop learning rules with several decay rates