MLP Coursework 1: Learning Algorithms and Regularization

s1546138

Abstract

This report contains our findings of benchmarking RMSProp and Adam optimizers against Stochastic Gradient Descent without explicit regularization as a baseline. The optimizers are tested on Convolutional Neural Networks (CNNs)

1. Introduction

Our objective is to investigate the effectiveness of different optimizers, namely SGD, RMSprop and Adam on neural network models trained on the balanced EMNIST dataset. We will also look into learning rate scheduling using cosine annealing with warm restarts and the effect of different regularization options on the Adam optimizer (ridge regularization vs weight decay).

We focus on implementation of algorithms proposed in the various papers and applying them to a non-trivial classification task.

We use an early stopping criterion to measure the convergence behavior of the different optimizers. The stopping criterion is defined as 3 sequential epochs where the validation loss does not decrease.

1.1. EMNIST Dataset

The EMNIST (Extended MNIST) Balanced dataset (Cohen et al., 2017) will be used to evaluate the performance of various models, optimizers and hyper-parameters. The dataset was selected as it contains a greater number of examples and classes, which should prove a greater challenge to classifiers and provide a stronger signal for evaluating various trained models, while remaining a simple enough task that can be solved with a reasonable amount of compute resource.

The is an extended version of the MNIST handwritten digits dataset, and contains the addition of handwritten alphabet glyphs in both lower and upper case. The glyphs from both datasets are extracted from the NIST Special Database 19, into 28×28 black and white rasters.

The dataset contains 62 classes (10 digits, 26 of each lower and upper case letters), of which we are only using a subset of 47 of those classes. The reason being 15 of the letters (namely C, I, J, K, L, M, O, P, S, U, V, W, X, Y, Z) do not have glyphs that are easily distinguishable between the lower and upper case variants. These classes will be merged to contain examples with both upper and lower case examples.

There are a total of 100,000 samples in the training set, 15,800 samples in the validation set and 15,800 samples in the test set.

2. Baseline systems

We first establish a baseline model trained using a SGD optimizer without explicit regularization. The baseline experiments were performed using the EMNIST training/validation sets for hyperparameter selection. We used a batch size of 100 for all experiments, and trained for a maximum of 100 epochs per model. For each hidden layer, we have 100 units with ReLU activation. No explicit regularization is used, e.g. weight regularization / dropout.

The hyper-parameters that we are interested in tuning is the number of hidden layers in the NN and learning rate of SGD optimizer. Hyperparameter selection is performed with grid search with holdout validation. The results of the baseline experiments are reported in Table 4, and the error plots are shown in Appendix 1.1.

Finally, we show the impact that number of layers plays in the NN model by comparing the test losses of the best performers by validation loss of each category (bolded rows in Table 4).

From the table and graphs, it is observed that increasing the number of hidden layers does not improve model performance noticeably. We also observe that models with more hidden layers are more sensitive to the learning rate parameter, and training diverges when the learning parameter is too large (this effect is more noticeable with more hidden layers). Judging from the validation results (and graphs in Appendix 1), 100 epochs is too many and the models are be overfitted (to varying extents) and could benefit from early stopping. This however, is not too much of a concern as the 100 epochs looks about right for $\eta = 0.01$, our chosen learning rate parameter from validation results. The parameters of the trained model are saved and the best models tested against the test set. We show both the cross entropy softmax error as well as the overall accuracy (which is requested by the coursework). Due to the large number of class labels however, the cross entropy error should be a more reliable measure of classifier performance.

H. Layers	Learning Rate	Training Loss	Training Acc	Validation Loss	Validation Acc	Test Loss	Test Acc
2	**0.01**	**0.382**	**0.873**	**0.499**	**0.837**	**0.546**	**0.826**
	0.05	0.206	0.922	0.643	0.826		
	0.1	0.169	0.932	0.839	0.821		
	0.5	0.314	0.886	1.08	0.789		
	1.0	0.501	0.844	1.02	0.781		
3	**0.01**	**0.315**	**0.891**	**0.488**	**0.842**	**0.534**	**0.831**
	0.05	0.165	0.934	0.800	0.823		
	0.1	0.171	0.931	0.991	0.816		
	0.5	0.394	0.863	0.977	0.796		
	1.0	3.85	0.0212	3.85	0.0215		
4	**0.01**	**0.281**	**0.900**	**0.508**	**0.839**	**0.552**	**0.829**
	0.05	0.143	0.941	0.874	0.827		
	0.1	0.188	0.924	0.993	0.815		
	0.5	0.358	0.876	0.744	0.814		
	1.0	3.85	0.0213	3.85	0.0210		
5	**0.01**	**0.271**	**0.900**	**0.546**	**0.834**	**0.587**	**0.826**
	0.05	0.157	0.938	0.952	0.824		
	0.1	0.200	0.921	0.943	0.813		
	0.5	0.406	0.864	0.687	0.818		
	1.0	3.85	0.0211	3.85	0.0213		

Table 1. Baseline results using SGD optimizer

3. Learning algorithms – RMSProp and Adam

Algorithm 1 and Algorithm 2 show the annotated update rules for RMSProp and Adam optimizers respectively.

The idea and intuition behind RMSProp is to replace the accumulated squared gradients term in the Adagrad (Duchi et al., 2011) update rule with an exponentially decaying moving average. This solves the problem with using accumulated g^2 that Adagrad faces, which is that the accumulated term grows unbounded during training, resulting in the updates getting smaller and smaller over time, reaching a point where the parameters essentially are no longer updated.

The advantage of Adagrad (that Adam and RMSProp shares) is that the learning rates for different parameters can be individually moderated using the gradient/momentum w.r.t. each parameter. This it much easier to choose a sensible learning rate, as the algorithms are less sensitive to the learning rate.

Adam uses moving averages of both the first and second order moments of the gradient, which are also exponentially decayed. The authors noted that since the moment vectors are initialized to 0, they are biased towards 0. The effect is worse when the exponential decay rates (β_1, β_2) are small, and during the initial timesteps. To counteract this effect, we can correct for the initialization bias by dividing by $(1-\beta^t)$, where $(1-\beta^t)$ is derived from the ratio between a stationary true second moment g^2 and the expected value of the exponential moving average at timestep t.

For RMSprop, the beta parameter (rho in the original presentation) is kept at 0.9 (Stanford CS231n), and we tune the learning rate parameter. Experiment results are shown in (Figure 5). Validation points to $\alpha = 0.0005$ as the best learning rate, and we trained a model with early stopping used it in the final evaluations.

Model	Params	Epochs	Test Loss	Test Acc
Baseline	$lr = 0.01$	**100**	**0.534**	**0.831**
RMSprop	$lr = 0.0005$	22	0.572	0.826
Adam	$lr = 0.0005$	20	0.560	0.826
RMSprop	$lr = 0.0001$	26	0.538	0.829
Adam	$lr = 0.0001$	57	0.548	0.827

Table 2. RMSprop and Adam compared to baseline (see Figure 5 for curves)

For Adam, the bias correction parameters are quite robust (Fig 4. in (Kingma & Ba, 2014) shows this in a non-rigorous way), so they can be left as their default suggested values. ϵ is also left at it's default value. It should be noted at this point that Adam takes twice as long to train per epoch as SGD in our framework.

Figure 6 shows the loss curves for the Adam optimizer with various learning rate parameter. $\alpha = 0.0005$ performed the best and we trained a model with early stopping used it in the final evaluations.

We note that with $lr = 0.0005$, both RMSprop and Adam converged much faster compared to baseline, albeit having slightly poorer test performance. With a smaller $lr = 0.0001$, both models converged to a slightly better solution, but Adam took over twice the number of epochs before stopping compared to $lr = 0.0005$.

Algorithm 1 RMSProp

Require: α : Stepsize
Require: β : Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ : Initial parameter vector
  Initialize $m \leftarrow 0$ (moving average of squared gradients)
  while θ not converged **do**
    $g \leftarrow \nabla_\theta f(\theta)$ (Get gradients w.r.t. stochastic objective)
    $m \leftarrow \beta \cdot m + (1 - \beta) \cdot g^2$ (Update moving average of g^2 using exponential decay)
    $\theta \leftarrow \theta - \alpha \cdot g / (\sqrt{m} + \epsilon)$ (Update parameters w/ gradient divided by moving avg of g^2)
  end while
  return θ

Algorithm 2 Adam

Require: α : Stepsize
Require: β_1, β_2 : Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ : Initial parameter vector
  Initialize $m_1 \leftarrow 0$ (1st moment vector)
  Initialize $m_2 \leftarrow 0$ (2nd moment vector)
  Initialize $t \leftarrow 0$ (timestep)
  while θ not converged **do**
    $t \leftarrow t + 1$
    $g \leftarrow \nabla_\theta f(\theta)$ (Get gradients w.r.t. stochastic objective)
    $m_1 \leftarrow \beta_1 \cdot m_1 + (1 - \beta_1) \cdot g$ (Update biased first moment estimate)
    $m_2 \leftarrow \beta_2 \cdot m_2 + (1 - \beta_2) \cdot g^2$ (Update biased second raw moment estimate)
    $\hat{m_1} \leftarrow m_1 / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{m_2} \leftarrow m_2 / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta \leftarrow \theta - \alpha \cdot \hat{m_1} / (\sqrt{\hat{m_2}} + \epsilon)$ (Update parameters)
  end while
  return θ

In related literature by Wilson et al., it was also observed that adaptive gradient optimization methods albeit having better training performance than SGD, generalized poorly (often significantly) compared to SGD.

4. Cosine annealing learning rate scheduler

The cosine annealing learning rate scheduler introduced in (Loshchilov & Hutter, 2016), aims to improve the anytime performance when training neural networks. The goal of learning rate being decayed through cosine annealing is to reach a good position quickly, while decreasing the learning rate enough to narrow down to a better solution. Warm restarts resets the learning rate at scheduled intervals, which then continue to decay per the schedule. Restarts helps in cases of poor initialization of NN weights, and modeling multimodal functions. It is shown in (Loshchilov & Hutter, 2016) that annealing with restarts can lead to 2-4x faster convergence (in the CIFAR dataset). The initial learning weight is the most important hyperparameter (Bengio, 2012), and thus we will focus on tuning that owing to time and compute constraints.

Cosine annealing of learning rate shows better performance in test performance in Adam while also converging in far fewer epochs. With the addition of warm restarts, Adam performs better than all previous models yet. However, we note that generalization of the model is not very good, as seen by the big gap between training and validation curves.

It should be noted that the early stopping criteria might need to be tweaked when used in conjunction with warm restarts, as resetting the learning rate leads to a uptick in the loss function as shown in the graphs. A more lax stopping criteria would probably perform better. Given that a important feature of the proposed scheduler is it's anytime performance, our evaluation metric considering only the final performance might not be the best evaluation of the scheduler. Future work should focus more on characterizing the anytime performance of the scheduler versus other annealing schedules for learning rate.

5. Regularization and weight decay with Adam

L2 regularization adds the L2 norm of weights (multiplied by some constant) to the loss function, in order to regularize the weights towards 0. In Loshchilov & Hutter (2017), the authors proposes that weight decay for Adam implemented in popular ML frameworks using L2 regularization is wrong, more generally in the context of adaptive gradient

Model	Params	Epochs	Test Loss	Test Acc
Baseline	$lr = 0.01$	100	0.534	0.831
Adam	$lr = 0.0001$	57	0.548	0.827
SGD Anneal noRestart	$lr = 0.05$	26	0.538	0.828
Adam Anneal noRestart	$lr = 0.0005$	17	0.544	0.829
SGD Anneal Restart	$lr = 0.05, T_i = 10, T_{mult} = 3, decay = 0.9$	30	0.543	0.827
Adam Anneal Restart	$lr = 0.0005, T_i = 10, T_{mult} = 3, decay = 0.9$	**40**	**0.522**	**0.837**

Table 3. Comparison of constant learning rate, cosine annealing and cosine annealing with warm restarts.

algorithms, due to the regularization strength being tied to the magnitude of gradients. This is contrary to views presented in prior literature such as Goodfellow et al. (2016). The paper proposes a 'fix' which decouples the regularization from the loss function, which is how L2 regularization is implemented (as an added term to the loss). Algorithm 2 in Loshchilov & Hutter (2017) illustrates the difference between the new proposed 'weight decay' update compared to L2 regularization. ICLR reviewers noted that while the authors demonstrated empirical results supporting the proposed change using cosine annealing of the learning rate, it does not show evidence of improvement independent of cosine annealing. It was also mentioned that while L2 regularization provides a well reasoned justification for weight decay, the paper does not provide sufficient justification of the modified 'weight decay' algorithm. In the revised ICLR 2019 submission (Anonymous, 2019), the authors elaborated on a justification of decoupling weight decay from optimization steps in adaptive gradient methods from a bayesian filtering perspective, building on Aitchison (2018), where the weight decay term corresponds to a regularizer of the state-transition Gaussian distribution.

We formulate the following experiment (Table 4) to compare L2-regularization against the proposed weight decay algorithm. For each weight regularization method, we train 3 models, one with a constant learning rate, one with cosine annealing and another with cosine annealing and warm restart. This should also show if either methods synergizes well with an annealing schedule for our dataset (the paper claims that AdamW with Cosine Warm Restart improves error by 15% and converges 10x faster on their dataset).

Adam with weight decay performed better than the Adam baseline on all examples, with cosine annealing without restart being the best performing model for AdamW. However, I do suspect that `AdamW WarmRestart` might be underfitted and should be run with a more lax stopping criteria.

Adam with L2 regularization worked really well, beating the performance of Adam Baseline by a significant margin in 2 out of 3 of the experiments. Again, we note that the WarmRestart variant might be underfitted and could benefit from retraining with more epochs.

Regarding the hiccups in WarmRestart runs, although the final performance of the models did not compare well against their counterparts, the loss curves show better generalization characteristics.

It should be pointed out that our experiment results of Adam with restarts do not replicate some of the peculiarities in the (Loshchilov & Hutter, 2017) experiments with the CIFAR and ImageNet datasets. Bringing your attention to Figure 4 in (Loshchilov & Hutter, 2017), the error curves show drastic spikes in error when the learning rate 'reset' happens, which is not observed in our results (Figure 8). I hypothesize that our NN architecture (a simple feedforward network) is much more robust to learning rate changes than the CNNs in the (Loshchilov & Hutter, 2017) paper.

6. Conclusions

Empirical findings (Table 5) show that Adam with L2 regularization and constant learning rate performed the best on the testing dataset. The baseline SGD model generalized surprisingly well, beating both RMSprop and Adam baselines. Adding regularization to model weights also improved Adam performance significantly. The relative poor performance of the more sophisticated methods can be possibly attributed to the limited exploration of hyperparameter space including inefficient hand optimization of hyper-parameters. Algorithmic optimization of hyperparameters can be done as an extension to this work (Feurer et al., 2015).

It is worth mentioning that the performance of classifiers on the test set is often significantly worse than on the validation set. This strongly suggests that the test distribution might be different from the validation/training distributions(Storkey, 2009). This opens up an interesting area for exploration into how we can tackle mismatched distributions(González & Abu-Mostafa, 2014).

The framework that we are working with is not optimized, and the training times for the models limited experimentation heavily. Due to compute and time constraints, we were unable to experiment with longer epochs and smaller learning rates, and could only tune 1 or 2 hyperparameters that were the most important. For reference, 100 epochs of vanilla SGD takes 15 mins on the overloaded `student.compute` cluster, which is twice as slow as on a 7 year-old budget laptop. A more optimized algorithm would enable much more through analysis and characterization of the algorithms covered in the report.

Further work could explore preprocessing of the dataset, for example normalizing inputs, and using batch norms to tackle covariate shifts. An obvious extension would to be

Model	Params	Epochs	Test Loss	Test Acc
SGD Baseline	$lr = 0.01$	100	0.534	0.831
Adam Baseline	$lr = 0.0001$	57	0.548	0.827
AdamW ConstantLR	$lr = 0.0003$,	35	0.515	0.833
AdamW noRestart	$lr = 0.0003$,	31	0.510	0.836
AdamW WarmRestart	$lr = 0.0003, T_i = 10, T_mult = 3$	27	0.548	0.828
AdamL2 ConstantLR	$l2reg = 0.001, lr = 0.0003$,	**42**	**0.496**	**0.836**
AdamL2 noRestart	$l2reg = 0.001, lr = 0.0003$,	41	0.500	0.836
AdamL2 WarmRestart	$l2reg = 0.001, lr = 0.0003, T_i = 10, T_{mult} = 3$	28	0.555	0.812

Table 4. Comparision of Adam with L2 regularization (AdamL2) and Adam with weight decay (AdamW). Each model trained with constant learning rate, cosine annealing without restart and cosine annealing with warm restart. (see Figure 7 for curves)

Model	Params	Epochs	Test Loss	Test Acc
SGD Baseline	$lr = 0.01$	100	0.534	0.831
Adam Anneal Restart	$lr = 0.0005, T_i = 10, T_{mult} = 3, decay = 0.9$	40	0.522	0.837
AdamL2 ConstantLR	$l2reg = 0.001, lr = 0.0003$,	**42**	**0.496**	**0.836**

Table 5. Final comparison of top performers from each Section. Best test performance in bold.

using CNNs, which have been shown to perform well in handwriting classification tasks (Bottou et al., 1994). For cosine annealing with warm restarts, it is worth characterizing the anytime performance of the scheduler, since that is one of the touted pros of the algorithm but Loshchilov & Hutter (2016) does not cover that in much detail.

References

Aitchison, Laurence. A unified theory of adaptive stochastic gradient descent as bayesian filtering. *arXiv preprint arXiv:1807.07540*, 2018.

Anonymous. Decoupled weight decay regularization. In *Submitted to International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7. under review.

Bengio, Yoshua. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pp. 437–478. Springer, 2012.

Bottou, Léon, Cortes, Corinna, Denker, John S, Drucker, Harris, Guyon, Isabelle, Jackel, Lawrence D, LeCun, Yann, Muller, Urs A, Sackinger, Edward, Simard, Patrice, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pp. 77–82. IEEE, 1994.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Feurer, Matthias, Klein, Aaron, Eggensperger, Katharina, Springenberg, Jost, Blum, Manuel, and Hutter, Frank. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pp. 2962–2970, 2015.

González, Carlos R and Abu-Mostafa, Yaser S. Four results in matching data distributions. *Jun*, 21:1–18, 2014.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Loshchilov, Ilya and Hutter, Frank. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Loshchilov, Ilya and Hutter, Frank. Fixing weight decay regularization in Adam. *arXiv preprint arXiv:1711.05101*, 2017. URL https://arxiv.org/abs/1711.05101.

Storkey, Amos. When training and test sets are different: characterizing learning transfer. *Dataset shift in machine learning*, pp. 3–28, 2009.

Wilson, Ashia C, Roelofs, Rebecca, Stern, Mitchell, Srebro, Nati, and Recht, Benjamin. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4148–4158, 2017.

7. Appendix

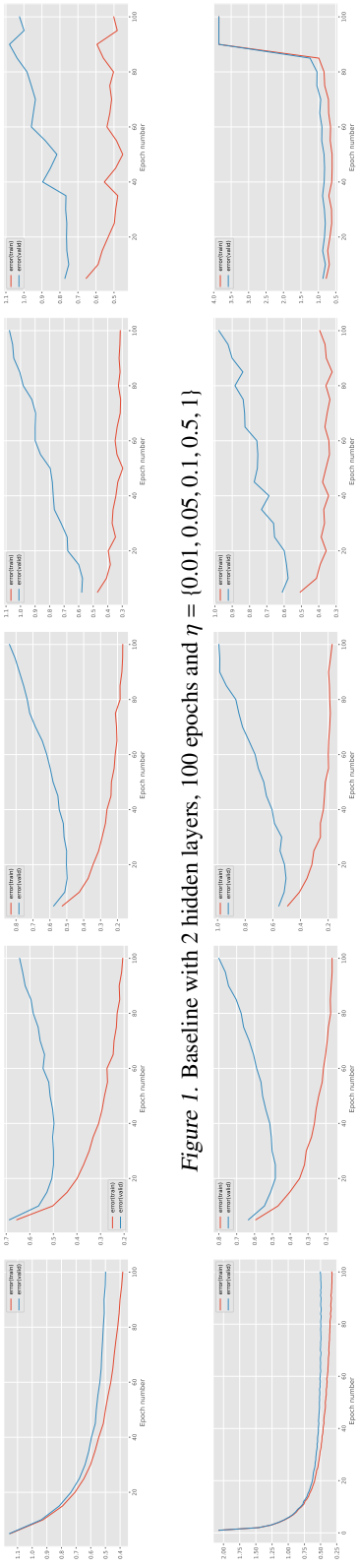


Figure 1. Baseline with 2 hidden layers, 100 epochs and $\eta = \{0.01, 0.05, 0.1, 0.5, 1\}$

Figure 2. Baseline with 3 hidden layers, 100 epochs and $\eta = \{0.01, 0.05, 0.1, 0.5, 1\}$

Figure 3. Baseline with 4 hidden layers, 100 epochs and $\eta = \{0.01, 0.05, 0.1, 0.5, 1\}$

Figure 4. Baseline with 5 hidden layers, 100 epochs and $\eta = \{0.01, 0.05, 0.1, 0.5, 1\}$

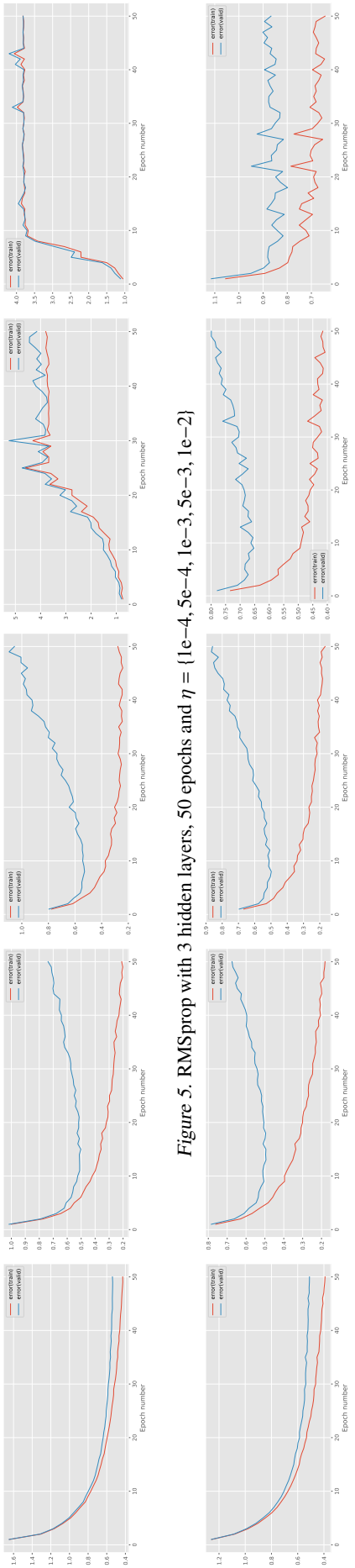
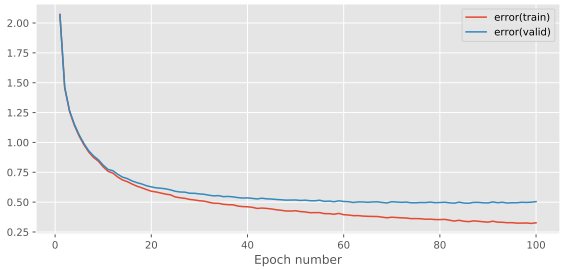
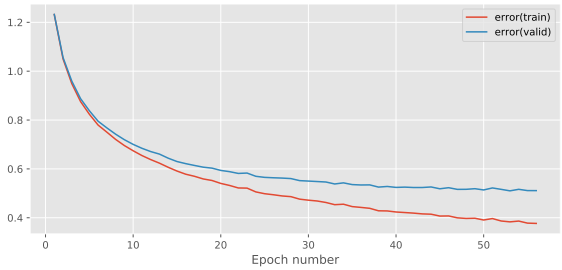


Figure 5. RMSprop with 3 hidden layers, 50 epochs and $\eta = \{1e{-}4, 5e{-}4, 1e{-}3, 5e{-}3, 1e{-}2\}$

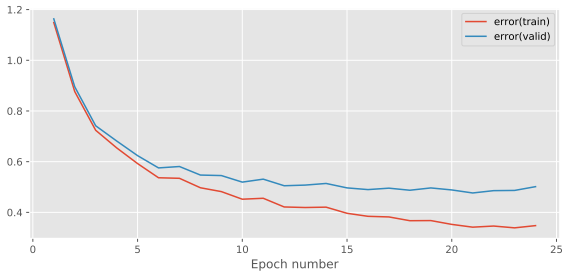
Figure 6. Adam with 3 hidden layers, 50 epochs and $\eta = \{1e{-}4, 5e{-}4, 1e{-}3, 5e{-}3, 1e{-}2\}$



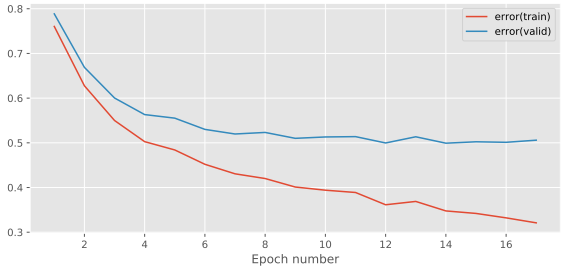
(a) SGD baseline, lr_{max}=0.01



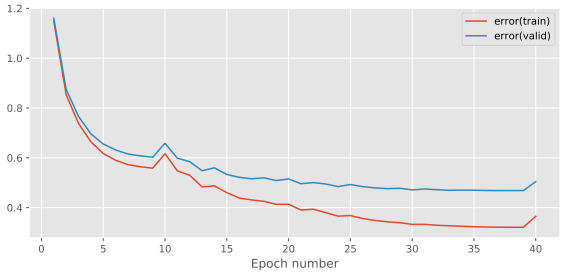
(b) Adam baseline, lr_{max}=0.0001



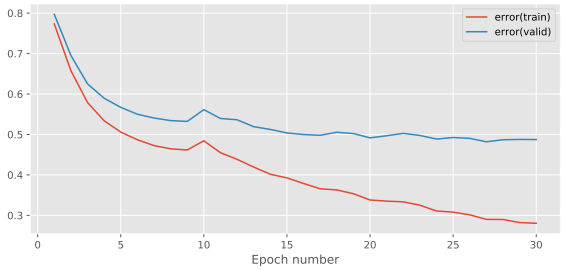
(c) SGD annealed with no restart, lr_{max}=0.01



(d) Adam annealed with no restart, lr_{max}=0.0001

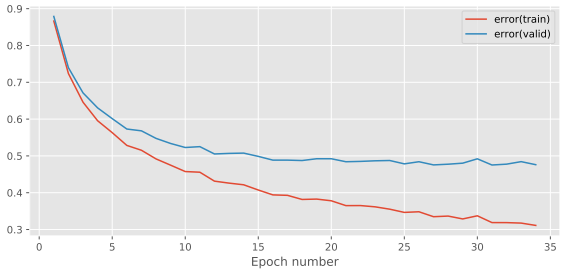


(e) SGD annealed with warm restart, lr_{max}=0.01

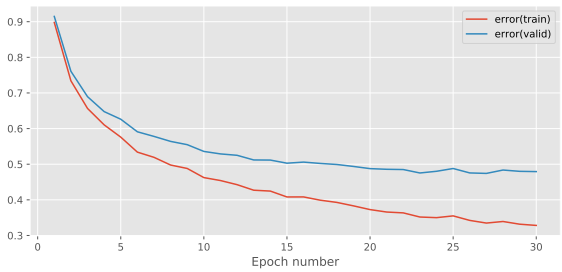


(f) Adam annealed with warm restart, lr_{max}=0.0001

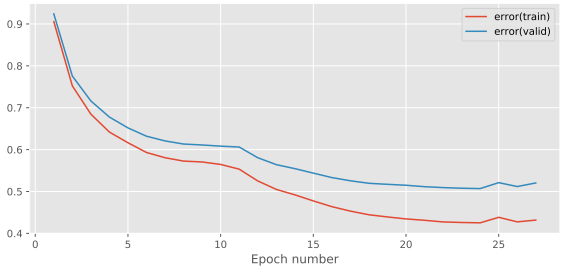
Figure 7. A comparison of no learning rate annealing, annealing with no restart and annealing with restarts. SGD & Adam optimizer on 3 hidden layer ReLU networks.



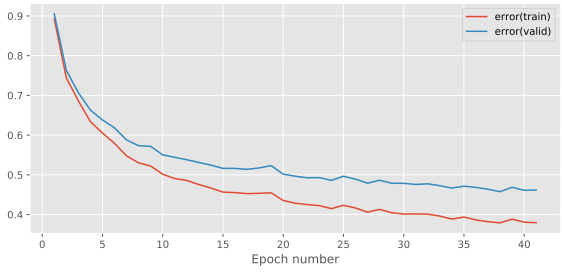
(a) AdamW ConstantLR, lr_{max}=0.0003



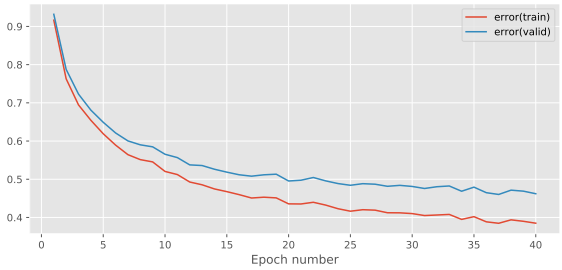
(b) AdamW noRestart, lr_{max}=0.0003



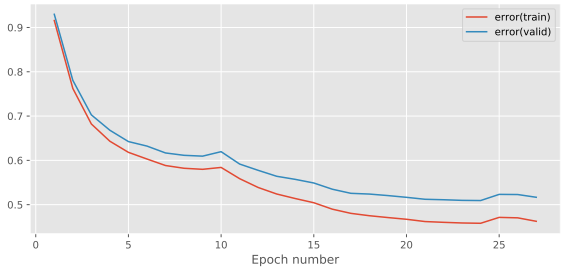
(c) AdamW WarmRestarts, lr_{max}=0.0003



(d) AdamL2 ConstantLR, lr_{max}=0.0003



(e) AdamL2 noRestart, lr_{max}=0.0003



(f) AdamL2 WarmRestarts, lr_{max}=0.0003

Figure 8. A comparison of no learning rate annealing, annealing with no restart and annealing with restarts. AdamL2 & AdamW on 3 hidden layer ReLU networks.