MLP Coursework 1: Learning Algorithms and Regularization

s1837379

Abstract

In this coursework, we explore the use of different training techniques for fitting a neural network on the EMNIST dataset of handwritten digits. We experiment with different numbers of hidden layers (two through five); different learning rules (SGD, RMSProp, and Adam); different learning rate schedules (constant and cosine annealing with and without restarts); and regularization (L2 and weight decay). We conclude that the Adam learning rule makes a three-hidden-layer model reach a good fit faster than the baseline and that L2 regularization reaches a better test set accuracy than the baseline after 100 epochs of training.

1. Introduction

In recent years, artificial intelligence systems have become an increasingly important part of software and workflows across many industries. Such systems work by analyzing many example data points to attempt to "learn" fundamental properties of the data, which they can later apply to make predictions about previously unseen data.

A common approach for leaning such properties for high-dimensional inputs is to model them using a deep neural network. This involves taking linear combinations of a vector of input data (e.g. an image), passing it through a non-linear function, and repeating this process for several "layers" that each take the previous layer's output as input. The final layer produces an output (e.g. probabilities for different class labels for the image). Each layer has its own set of weights by which it multiplies its input vector. The learning process involves optimizing these weights to produce the correct outputs for the given inputs.

A basic process to optimize the weights in a deep neural network is to use gradient descent, a technique that repeatedly calculates the gradient of the error of the current weights' output (e.g. the difference between the correct and predicted probabilities for image labels) and then slightly tweaks the weights in the steepest direction that would decrease the error, propagating this change backwards through the model's layers.

During this process, there are many steps we can take to improve our performance in terms of accuracy and speed. First, it may not always be the best idea to follow the steepest descent directly, but rather keep track of our general momentum to avoid oscillations; these ideas are incorporated in the RMS-Prop (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2014) learning algorithms. Second, we can tweak by how much we change the weights on each training pass: we can adjust this learning rate to vary over time using a cosine annealing learning schedule (Loshchilov & Hutter, 2017). Third, to ensure we don't overfit on training data, we can apply L2 regularization, keeping weights relatively small, or decay our weights over time (Loshchilov & Hutter, 2017). In this report, we explore and compare all these options.

The rest of this report is structured as follows: in Section 2, we describe the dataset on which we experiment; in Section 3, we explain the baseline against which we compare the algorithms; in Section 4, we explore and test the RMSProp and Adam learning algorithms; in Section 5, we explore and test cosine annealinig learning schedules; in Section 6, we explore and test L2 regularization and weight decay on the Adam learning rule; finally, in Section 7, we compare all the models we've implemented.

2. The EMNIST Dataset

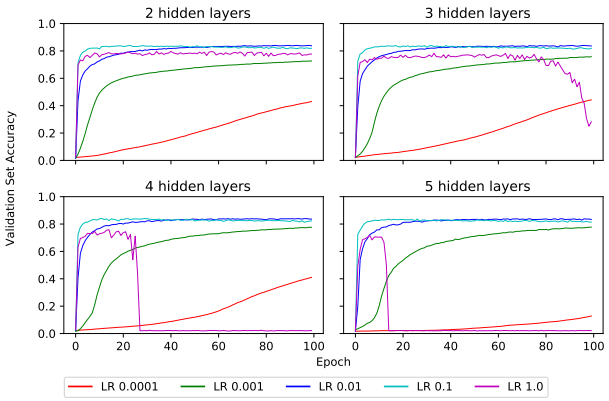
A common machine learning problem with many practical applications is digitizing hand-written text, which is useful for, for example, automating (snail) mail delivery. More formally, (a part of) this problem is one of classification: given a scanned input of a handwritten character, can we classify which letter or digit it is?

Handwritten character classification is such a common problem that a standard dataset has emerged for it: the EMNIST dataset, which is "a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format" (Cohen et al., 2017). With 26 upercase and lowercase letters and 10 digits, EMNIST has 62 classes. We use a version of the dataset that merges letters whose upper- and lowercase versions are hard to distinguish at a normalized scale (such as C, I and M), leaving 47 classes.

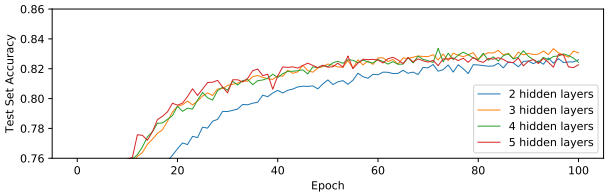
EMNIST has a total of $131,600$ examples, which we split into $100,000$ data points (76%) for training, $15,800$ (12%) for validation, and $15,800$ (12%) for testing.

3. Baseline Systems

As a baseline, we experiment with Stochastic Gradient Descent (SGD) at different learning rates in a network with two to five hidden layers of 100 ReLU units each, trained for 100 epochs.



(a) Validation set comparison of baseline SGD with different learning rates across two to five hidden layers.



(b) Test set comparison of baseline SGD at learning rate 0.01 and different hidden layer counts.

Figure 1: Baseline SGD comparisons, trained for 100 epochs each.

The accuracy on the validation set of each of these 20 models is plotted in Figure 1(a). Clearly, learning rates 0.0001 (red) and 0.001 (green) underfit on each model, never reaching more than approximately 40% and 60% accuracy, respectively. Learning rates 0.1 (cyan) and 1.0 (magenta) find a good fit in just a few epochs, but behave erratically afterwards—especially 1.0, which appears to become unstable on higher hidden layer counts.

For each hidden layer count, the sweet spot appears to be a learning rate of around 0.01. For two through five hidden layers, the accuracy on the validation set at learning rate 0.01 after 100 epochs is 83.7%, 83.2%, 82.7%, and 81.7%, respectively. This performance trends slightly downward as the number of hidden layers increases, which could be explained by the model being increasingly capable of overfitting on the training data.

The accuracy on the test set for each layer count trained at learning rate 0.01 is plotted in Figure 1(b); all models perform quite similarly. Two hidden layers takes slightly longer to find a good fit than the rest; three hidden layers edges out four and five hidden layers, but not significantly. Given the computational advantage of having fewer layers, though, it makes sense to go with a three-hidden-layer SGD model trained at a learning rate of 0.01 as baseline model.

4. Learning Algorithms: RMSProp & Adam

Two improvements over standard SGD are the RMSProp (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2014). In this section, we investigate each of them and compare their performance to the baseline.

SGD, our baseline model, adjusts each weight w_i on each tth mini batch according to Equation 1, where d_i is the error gradient with respect to the weights and η is the learning rate:

$$w_i(t) = w_i(t-1) - \eta d_i(t) \tag{1}$$

4.1. RMSProp

RMSProp differs mostly from gradient descent because it keeps track of its momentum over time, which enables it to keep moving in the general direction down the error function while avoiding oscillations from using only the steepest descent at the current point. To do this, RMSProp adjusts the way weights are updated, incorporating a moving average of the squared gradient S_i defined by Equation 2, where β_2[1] is a hyperparameter for the decay that the authors suggest to set around 0.9.

$$S_i(t) = \beta_2 S_i(t-1) + (1-\beta_2) d_i(t)^2 \tag{2}$$

To update the weights, RMSProp replaces Equation 1 from SGD with Equation 3, where ϵ is a small smoothing factor set to around 10^{-8}.

$$w_i(t) = w_i(t-1) - \frac{\eta}{\sqrt{S_i(t)} + \epsilon} d_i(t) \tag{3}$$

4.2. Adam

Adam is an iteration of RMSProp to address an issue raised by Hilton that momentum is not very effective. Instead of updating the weights using the gradient directly as RMSProp does, Adam updates the weights using a momentum-smoothed gradient M_i, defined in Equation 4, where β_1 is a smoothing hyperparameter.

$$M_i(t) = \beta_1 M_i(t-1) + (1-\beta_1) d_i(t) \tag{4}$$

To update the weights, Adam replaces Equation 1 from SGD with Equation 5; the authors recommend setting $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

$$w_i(t) = w_i(t-1) - \frac{\eta}{\sqrt{S_i(t)} + \epsilon} M_i(t) \tag{5}$$

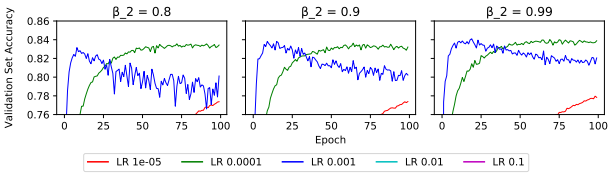
4.3. Experimentation

To determine the best hyperparameter settings for RMSProp and Adam, we performed the following experiments. For all experiments, we kept the hidden both the hidden layers and smoothing factor ϵ constant, respectively at three 100-ReLU-unit hidden layers and at the recommended setting of 10^{-8}.

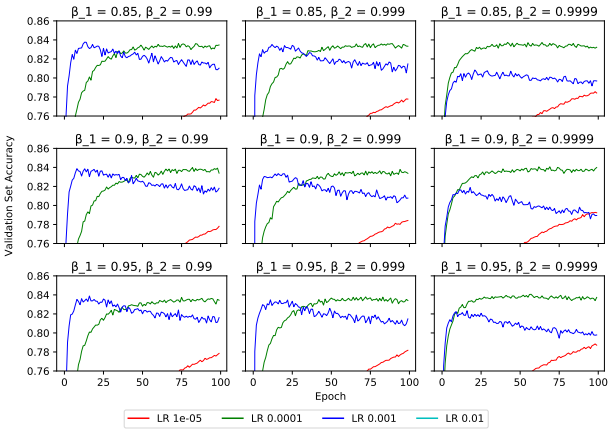
For RMSProp, we varied the learning rate and β_2 hyperparameters as follows:

- β_2: 0.8, 0.9, and 0.99
- Learning rate: 10^{-5}, 0.0001, 0.001, 0.01, and 0.1

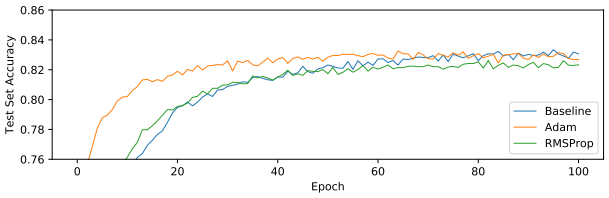
[1] We label this hyperparameter β_2 for consistency with the code, even though we introduce it before Adam's β_1



(a) RMSProp performs best on the validation set at $\beta_2 = 0.99$ and learning rate 0.0001 (right, green). Learning rates 0.01 and 0.1 are too low to appear on the graph.



(b) Adam performs best on the validation set at learning rate 0.0001 (green); $\beta_1 = 0.9$ and $\beta_2 = 0.9999$ slightly beat other βs settings.



(c) At best validation set score settings, Adam reaches its peak performance faster than SGD and the baseline.

Figure 2: Accuracy of RMSProp and Adam on validation set (2(a) and 2(b)) and on test set (2(c)) with hyperparameters set to those with best validation set accuracy; everything trained for 100 epochs.

The accuracy of each of these 15 models on the validation set is plotted in 2(a). From these graphs, it is clear that $\beta_2 = 0.99$ (the recommended setting) at learning rate 0.0001 performs best: it reaches both the highest validation score of all hyperparameter settings (TODO) and produces the smoothest curve.

For Adam, we varied the learning rate, β_1, and β_2 hyperparameters as follows:

- β_1: 0.85, 0.9, and 0.95
- β_2: 0.99, 0.999, and 0.999
- Learning rate: 10^{-5}, 0.0001, 0.001, and 0.01

The accuracy of each of these 36 models on the validation set is plotted in 2(b). From these graphs, we can see that $\beta_1 = 0.9$, $\beta_2 = 0.9999$, and learning rate 0.0001 perform best, although many settings of appear similar[2]. Although some other settings produce quite similar results, it seems best to stick with the recommended settings for generalization.

After determining these best settings for RMSProp and Adam on the validation set, we compare them to the baseline on their test set accuracies, as plotted in 2(c). Adam clearly comes out ahead, reaching a high accuracy (in the 82.5 – 83% range) on the test set after just 50 epochs, while the baseline takes 100 epochs to reach a similar performance. RMSProp never quite reaches this level, peaking at 82.2% accuracy.

These results suggest that Adam is a worthwhile improvement over our SGD baseline, especially in terms of training speed: using Adam enables us to reach the same accuracy as SGD at a training budget of half as many epochs.

5. Cosine Annealing Learning Rate Scheduler

So far, we've explored using different learning rules (namely, RMSProp and Adam) to improve the accuracy and training speed of a neural network classifier on EMNIST data. Another approach to improving learning is to adjust the learning rate—how much of a "bump" the gradients can give to the weights in each mini batch iteration—over time. The intuition is that, as we approach a minimum in the cost function, we should slow down to avoid constantly overshooting it.

As presented by (Loshchilov & Hutter, 2017), cosine annealing is such an algorithm that works by periodically quickly decreasing the learning rate and then slowly increasing it again. It calculates the learning rate η according to Equation 6, where i is the index of the current epoch.

$$\eta_t = \eta_{min}^{(i)} + 0.5(\eta_{max}^{(i)} - \eta_{min}^{(i)})(1 + \cos(\pi T_{cur}/T_i)) \quad (6)$$

Cosine annealing using the following hyperparameters, some of which can be found in Equation 6:

- η_{min}: the minimum learning rate
- η_{max}: the maximum learning rate, which is also the starting learning rate
- T_i: the total iterations per period, after which we restart by quickly decreasing the learning rate
- T_{mult}: a factor by which we increase or decrease T_i after each restart
- d: a factor by which we decrease η_{max} after each restart

We apply the final two hyperparameters outside Equation 6, whenever a restart occurs (e.g. after T_i iterations pass after the previous restart).

[2]This could be an indication that we should have tried more diverse settings for β_1 and β_2; due to time constraints, however, we did not attempt these. Training 36 models on a laptop takes quite a bit of time.

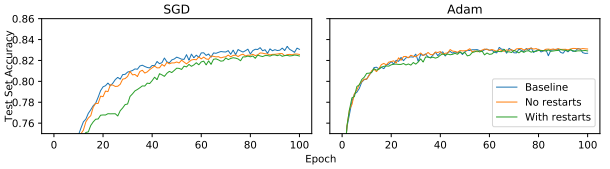
After implementing the cosine annealing learning rate scheduler to pass unit tests, we ran six experiments: the following three learning schedules, applied to both the SGD and Adam learning rules, all trained for 100 epochs.

1. Keeping the learning rate constant
2. Adjusting the learning rate using cosine annealing without restarts
3. Adjusting the learning rate using cosine annealing with restart

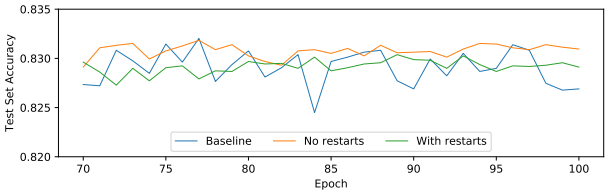
For each of these, we keep the learning rule hyperparameters constant (e.g. $\beta_1 = 0.9, \beta_2 = 0.9999$ and $\epsilon = 10^{-8}$ for Adam). For the each learning rule, we set the maximum (starting) learning rate where they performed best on previous experiments: $\eta_{max} = 0.01$ for SGD; $\eta_{max} = 0.0001$ for Adam. Finally, we keep the cosine annealing scheduler's hyperparameter η_{min} constant at 0.00001. *Experiment 1*, the baseline, requires settings that we have previously run in other tasks. No need to re-run them!

Experiment 2, cosine annealing without restarts, requires setting the remaining hyperparameters as follows: $T_i = 100$ (preventing a restart before the end of training) and $T_{mult} = 1$ (irrelevant since there are no restarts).

Experiment 3, cosine annealing with restarts, requires setting the remaining hyperparameters as follows: $T_i = 25$ (causing a restart after 25 epochs) and $T_{mult} = 3$ (causing the second restart to be 75 epochs later, at epoch 100, which is the end of training).



(a) Baseline (constant learning rate schedule), cosine annealing, and cosine annealing test set accuracies, grouped by learning rule; for the latter, a restart can clearly be observed at the 25th epoch.



(b) In this zoomed-in view of training set accuracy during the last 30 training epochs on the Adam learning rule, it is clear that the baseline (without cooled-down learning rate) is relatively unstable.

Figure 3: Comparison of a constant learning rate schedule, a cosine annealing schedule, and a cosine annealing schedule with restarts.

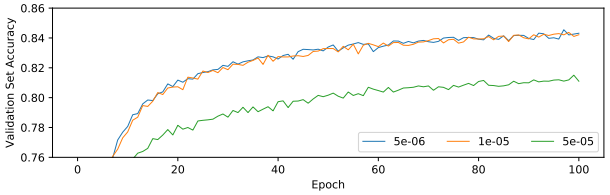
The results of these experiments are plotted in Figure 3(a), grouped by learning rule, which clearly shows a restart at 25 epochs for cosine annealing with restarts on both SGD and Adam.

Accuracy-wise, Figure 3(b) is more interesting: it shows that, on Adam, both versions of cosine annealing are more stable towards the end of training than the baseline. Furthermore, it shows that cosine annealing without restarts on Adam has the best stable accuracy so far, stabilizing between 83.1% and 83.2%[3].

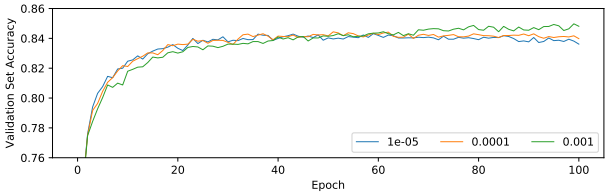
From these experiments, we can see that cosine annealing without restarts is the best-performing model tested so far.

6. Regularization and weight decay with Adam

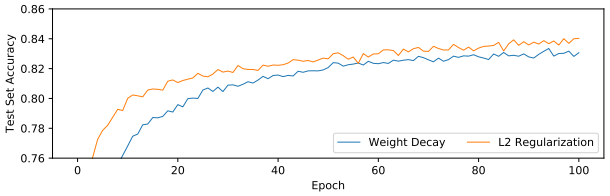
In (Loshchilov & Hutter, 2017), the authors argue that L2 regularization is not particularly effective for algorithms other than SGD because of the interaction between the learning rate parameter and the regularization constant. Instead, they argue, we should explicitly make weight decay part of the learning rule.



(a) The Adam learning rule with weight decay has the best accuracy at the default weight decay setting of 0.00001 (orange line).



(b) L2 Regularization has the best accuracy with L2 coefficient 0.001 (green line).



(c) On best validation set performance hyperparameter settings, L2 regularization outperforms weight decay on the test set.

Figure 4: Experiments with and comparison of the Adam learning rule with weight decay and L2 regularization.

To test these claims, we implement the Adam learning rule with weight decay to pass unit tests, and experiment on weight decay and L2 regularization. Across these experiments, the following are constant: we use the Adam learning rule with $\beta_1 = 0.9, \beta_2 = 0.9999$ and $\epsilon = 10^{-8}$; learning rate 0.0001; three hidden layers with 100 ReLU

[3]Even though the baseline does achieve this accuracy, it does so much more erratically.

	Model	Hyperparameters	Test Set Accuracy
1	Stochastic Gradient Descent	$LR = 10^{-2}$	0.831
2	RMSProp	$LR = 10^{-4}$; $\epsilon = 10^{-8}$; $\beta_2 = 0.99$	0.823
3	Adam	$LR = 10^{-4}$; $\epsilon = 10^{-8}$; $\beta_1 = 0.9$; $\beta_2 = 0.9999$	0.827
4	Cosine annealing (no restarts)	$\eta_{min} = 10^{-5}$; $\eta_{max} = 10^{-4}$; $T_i = 100$; $T_{mult} = 1$	0.831
5	Cosine annealing (with restarts)	$\eta_{min} = 10^{-5}$; $\eta_{max} = 10^{-4}$; $T_i = 25$; $T_{mult} = 3$	0.829
7	Weight decay	$LR = 10^{-4}$; weight decay = 10^{-5}	0.831
6	L2 regularization	$LR = 10^{-4}$; L2 coefficient = $10^{-}3$	**0.840**

Table 1: Comparison of final epoch accuracy on the test set of all models at the best settings found during validation set experimentation. All models have three hidden layers with 100 ReLU units each. Models 4 through 8 all use the Adam learning rule with ϵ, β_1 and β_2 are set to model 3 settings.

units each; and 100 training epochs.

Experiment 1 involves varying the weight decay hyperparameter for the Adam learning rule with weight decay. We vary this parameter slightly around the default, recommended value, setting it at 0.000005, 0.00001 (default setting), and 0.00005.

Experiment 2 involves varying the L2 coefficient hyperparameter for L2 regularization. We vary this parameter slightly around the settings from lab 5, setting it at 0.00001, 0.0001, and 0.001.

The results of experiment 1 are plotted in Figure 4(a), which shows that the Adam learnig rule with weight decay performs best on the validation set with the default weight decay setting of 0.00001. The results of experiment 2 are plotted in Figure 4(b), which shows that L2 regularization has the best accuracy with L2 coefficient 0.001.

Comparing these best settings for each model on the test set, as plotted in Figure 4(c), we can see that L2 regularization outperforms Adam with weight decay. This conclusion contradicts with the literature, since (Loshchilov & Hutter, 2017) claim Adam with weight decay fixes the aspects of Adam with L2 regularization that are ineffective; this suggests that there are better hyperparameter settings for the Adam learning rule with weight decay that we did not find in experimentation[4].

7. Conclusions

Through on the baseline described in Section 3 and the experiments described in Sections 4, 5 and 6, we found the best hyperparameter settings for six different models: stochastic gradient descent; RMSProp; Adam; cosine annealing on Adam (without restarts); cosine annealing on Adam (with restarts); weight decay on Adam; and L2 regularization on adam. These best hyperparameters, along with the corresponding model's final-epoch test set accuracy, are reported in Table 1.

The results in Table 1 show that, at the hyperparameters we found during experimentation, only the L2 regularization model improved over the baseline significantly in terms of final test set accuracy. This is surprising because the literature suggests that the weight decay model should be best; this indicates that we probably did not find the best hyperparameters for weight decay during experimentation.

The table does not, however, give us insight into learning curves. As Figure 2(c) shows, models that use the Adam learning rule reach good accuracy faster than models with RMSProp or SGD do. Overall, this is an interesting conclusion and a good motivation to use the Adam learning rule when training models.

For future work, it would be good to further explore the hyperparameter space for the weight decay model: since the literature (Loshchilov & Hutter, 2017) suggests it should perform better than L2 regularization and it performs about 1% worse on our experiments, there is room for improvement here.

[4]Because tweaking hyperparameters after seeing a model's performance on the test set is bad practice, we did not go back and try other hyperparameters for the Adam learning rule with weight decay.

References

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

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

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

Tieleman, Tijmen and Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.