MLP Coursework 1: Learning Algorithms and Regularization

s1786991

Abstract

Although learning algorithms based on *stochastic gradient descent* (SGD) (Bottou, 2010) has been broadly investigated and applied in the literature of deep learning, the review and understanding of them are still of significant importance. In this work, we compare vanilla SGD (Bottou, 2010), RMSProp (Geoffrey & Tijmen, 2012), and Adam (Kingma & Ba, 2014) in terms of their behavior during training and inference, using a multilayer perceptron model with 3 hidden layers trained on EMNIST (Cohen et al., 2017) for classification. Cosine annealing with restarts (Loshchilov & Hutter, 2016; 2017) is also investigated for both SGD and Adam on the improvement of performance. We finally analyze the effects of L2 regularization and weight decay in Adam. This report aims to help us get a better understanding of the learning algorithm, learning rate scheduler, and regularization/weight decay.

1. Introduction

In this work, we first establish a baseline model for the classification of digits and characters in EMNIST (Cohen et al., 2017), using *multi-layer perceptrons*. EMNIST is a dataset extended from MNIST, which has 62 potential classes (10 digits, 26 lowercase letters, and 26 uppercase letters). We use the balanced and merged version of it with 47 labels as 15 confusing upper-case and lower-case letters are merged together, for example, C, I, J, etc. The final dataset we used has 100,000 training entities, 15,800 validation entities, and 15,800 testing entities. Each entity is an image of size 28×28 containing an item belong to one of total 47 classes. The model is trained with mini-batch gradient descent, also called vanilla *stochastic gradient descent* (SGD) throughout this report. The details of hyper-parameters, architecture, the performance of the baseline model are shown in section 2.

Vanilla SGD is an efficient optimization method using mini-batches of size m for each update rather than all training examples in a dataset which known as batch gradient (GD), where the redundancy of computation is considerably reduced. The fluctuations of loss values, resulting from an approximation of gradients from mini-batches, may help find potentially better local minima. However, it suffers the difficulty for fine-tuning a proper learning rate, which is undesirably expensive, as its learning rate is used equally and globally for all the parameters. In order to tackle this challenge, adaptive learning methods, such as *RMSprop* (Geoffrey & Tijmen, 2012), *Adam* (Kingma & Ba, 2014), etc., are proposed, where each parameter has a different learning rate. In section 3, we explore the relationship among these learning algorithms and conduct experiments to investigate their behaviors during training. The results demonstrate that the adaptive methods have faster convergence speed than SGD, and may provide better performance on training data.

The convergence for stochastic optimization depends on the reduction of learning rate through time (Robbins & Monro, 1985) as SGD introduces noise during optimization (random sampling of m training examples). A learning rate schedule is often used to do the task. In section 4, we implement cosine annealing learning rate scheduler with warm restarts (Loshchilov & Hutter, 2016; 2017). We also study the learning behavior of SGD and Adam with this schedule, which yields better anytime performance.

Except for techniques mentioned above, regularization is another technique often used for improving the capability of generalization of a model. While L2 regularization and weight decay are identical for SGD, the same statement does not hold true for Adam (Loshchilov & Hutter, 2017). The effects of L2 regularization and explicit weight decay is studied for Adam in section 5. We also investigate the normalization effect of weight decay due to its dependence on the total number of weight updates.

Overall, in section 2, we build up a baseline model; we then study *SGD*, *RMSProp*, *Adam* in section 3; section 4 provides study of cosine annealing with warm restarts; in section 5, we review the effect of weight decay in *Adam*; finally, we conclude this report in section 6.

2. Baseline systems

In order to obtain a well-performed baseline system while keeping simplicity, we build multi-layer perceptron models with 2-5 hidden layers, where each hidden layer has 100 hidden units. The weights in each model are initialized from a Glorot uniform initializer (Glorot & Bengio, 2010) with a default gain of 1.0 and a random seed so that the optimization of models with same architecture starts from the same point, and all the biases are initialized by 0. We use mini-batch stochastic gradient descent to train all the models, with a batch size of 100. We keep the same batch size for all the experiments in this report. In this section, we first conduct hyper-parameter (learning rate) selection, then compare the best systems with 2-5 hidden layers using test set accuracy.

2.1. Hyper-parameters selection

For each system, we need to select a proper learning rate α and the number of epochs for training. Normally, the search of α is done by randomly sampling from the logarithm scale of a range avoid sampling too many undesirable α. We would firstly obtain a coarse learning rate, then perform the same procedure repetitively with a more and more precise range to fine-tune the best learning rate.

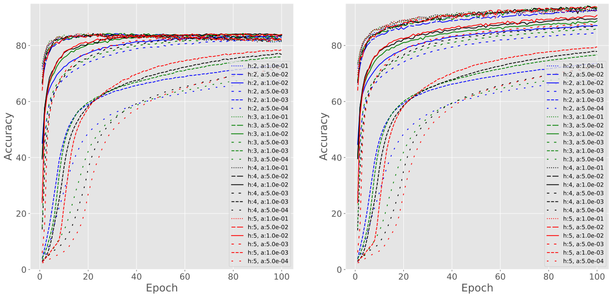
However, due to the limited computation resource, we only do a coarse grid search rather than random search without fine-tuning. Although such a search strategy cannot ensure an optimal learning rate, it would still give us a close approximation of the best learning rate. We sample 6 learning rates from $[5 - e4, 0.1]$, and train systems with 2-5 hidden layers using them for 100 epochs. It is true that many systems may overfit the training data for 100 epochs, but we could compute the maximum validation accuracy as shown in table 1, rather than the final validation accuracy, and its corresponding epoch number for each system to get approximations of the best ones. In order to avoid overfitting when using relatively large αs, we also look into the learning curves (training and validation) to decide which learning rate to use, shown in fig. 1.

Although results in table 1 show that, with an α of 0.1 or 0.05, the model could achieve the best performance on validation set, enormous differences are exhibited between the validation accuracy curves and training accuracy curves of these learning rates (see fig. 1), which means too large a learning rate guides the model to overfit on training data hence resulting in poor generalization. Instead of using the best performed but highly overfitted models, we use models with an α of 0.01 at which the models have comparable performance with the overfitted ones. The final settings of systems with 2-5 hidden layers are shown in table 2

Hidden Layers	2	3	4	5
α = .1000	84.18	84.08	84.20	84.32
α = .0500	84.16	84.30	84.25	84.04
α = .0100	84.03	84.18	84.29	84.03
α = .0050	82.51	83.35	83.56	83.85
α = .0010	73.35	75.63	77.54	78.55
α = .0005	67.39	69.32	70.58	72.08

Table 1: The best validation accuracy in percentage for each setting (α and hidden layers) of the system during training of 100 epochs.

2.2. Comparison results



Validation

Training

Figure 1: Accuracy curves for different learning rates.

We re-train the systems with their best hyper-parameter setting in table 2, and compute their accuracy on the testing set. Re-training the models may result in slightly different performance compared to the results reported in table 1, because optimization variance would result in the different performance of models trained even with same hyper-parameters and same start points in parameter space. Using Early Stopping technique could alleviate this problem by saving the best models only. From the results, the model with 3 hidden layers could achieve the best performance. Although increasing the number of hidden layers could monotonically increase the training set accuracy, the generalization of such models decreases due to overfitting. Therefore, we choose the system with 3 hidden layers as our baseline model.

Hidden Layers	Epochs	Train Acc.	Valid. Acc	Test Acc.
2	100	87.21	83.39	82.72
3	85	88.76	**84.06**	**83.01**
4	90	88.80	83.23	82.13
5	85	90.18	83.44	82.77

Table 2: The hyper-parameters of the best systems (all the models use an α of 0.01) and their performance on training, validation, test set in terms of accuracy percentage.

3. Learning algorithms - RMSProp & Adam

SGD (Robbins & Monro, 1985) mainly has 2 drawbacks. Firstly, it is easy to get trapped in sub-optimal local minima or saddle points due to its trouble to navigate ravines. Secondly, it uses the same learning rate α for all the parameters, whereas we expect larger learning rate for sparse data or unfrequent features and smoother learning rate for dominant data or frequent features. In order to overcome these disadvantages, plenty of advanced version of it has been proposed.

Classical momentum method (Polyak, 1964) is proposed to help SGD algorithm escape traps and speed up convergence by introducing velocity, the direction and speed at which the parameters move through the parameter space, into parameter updates, where it could use such velocity information and its position to decide updating direction.

The second problem of SGD is solved by the proposal of adaptive learning algorithms. *AdaGrad* (Duchi et al., 2011) is the first mini-batch based adaptive learning algorithm, adapting individual learning rules for each parameter by scaling them inversely according to the square root of the accumulated square values of all previous gradients. However, as it utilizes all the gradients information previous to current step, *AdaGrad* shrinks the learning rates too aggressively, which makes it difficult to optimize the objective function in later steps. *RMSProp* (Geoffrey & Tijmen, 2012) resolves this limitation by discarding the remote history and moving average by exponentially decay the average, introducing a new hyper-parameter, the decay rate β, which controls the length scale of the moving average, as described in algorithm 2. *Adam* (Kingma & Ba, 2014) is another adaptive learning algorithm, which combines the *RMSProp* and momentum methods, and also introduces some modifications to them. *Adam* estimates the first-order moment and second-order moment and exponentially decays them. It also includes the bias corrections of the estimates of moments to reduce variance early in training (see algorithm 3).

In this section, we explore the different learning behaviors result from the using of vanilla SGD, *RMSProp*, and *Adam*. We first choose the hyper-parameters for each algorithm using the validation set in section 3.1, and then compare the learning curves and test accuracy of them in section 3.2.

Algorithm 1 *SGD*. *SGD* with L_2 regularization and *SGD* with weight decay (*SGDW*), both with momentum

```
Require: Learning rate $\alpha$, momentum factor $\beta$, weight decay/$L_2$ regularization factor $w$
Require: Objective function with parameters $\theta$, $f(\theta)$
Require: Initial parameters $\theta_0$
  $t \leftarrow 0$   (Initialize time step)
  $m_0 \leftarrow 0$   (Initialize moments)
  while $\theta_0$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ $+ w \cdot \theta_{t-1}$ (Get gradients at t)
    $\eta_t \leftarrow$ Scheduler(t) (Fixed, decay, or warm restarts)
    $m_t \leftarrow \beta \cdot m_{t-1} - \eta_t \cdot \alpha \cdot g_t$   (Update momentum)
    $\theta_t \leftarrow \theta_{t-1} + \alpha \cdot g_t$ $+ \eta_t \cdot w \cdot \theta_{t-1}$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

3.1. Hyper-parameters selection

We train the baseline model with 3 hidden layers using vanilla SGD, *RMSProp*, and *Adam* algorithms. For SGD, we use the best hyper-parameter setting in section 2 with a learning rate of 0.01 and a training process of 100 epochs.

The hyper-parameters search is more complicated for *RMSProp* and *Adam* due to extra hyper-parameters of decay rate. In order to set proper decay rates for both algorithms, we initialize samples around the default value. The learning rates are also sampled from a range around the suggested value. More specifically, for *RMSProp*, the sample decay rate βs are $[0.5, 0.7, 0.9]$ and the sampled learning rate α are $[0.002, 0.001, 0.0005, 0.00025, 0.0001]$. We train models with each combination of learning rate α and decay rate β for 100 epochs, choose the one with best validation accuracy, and record its corresponding epoch number. For *Adam*, sampled β_1s are $[0.5, 0.75, 0.9, 0.95]$, β_2s are $[0.9, 0.94, 0.98, 0.999]$, and the learning rates are the same as *RMSProp*. There are numerous combinations of hyper-parameters in *Adam*, and training each setting for 100 epochs is extremely inefficient. We instead do an initial experiment with all the settings for 30 epochs and select a well-performed one for each learning rate, then train models with these settings for 100 epochs. Similar to the learning curves in fig. 1, models with relatively high learning rates achieve the best performance yet overfit the training data. Therefore, we choose 0.0001 as our final learning rate for both algorithms. All the final settings we used for comparison on test set are shown in table 3.

Algorithm 2 *RMSProp*

```
Require: Learning rate $\alpha$
Require: Decay rate $\beta \in [0, 1)$
Require: Objective function with parameters $\theta$, $f(\theta)$
Require: Initial parameters $\theta_0$
  $t \leftarrow 0$   (Initialize time step)
  $s_0 \leftarrow 0$   (Initialize $2^{nd}$ moments)
  while $\theta_0$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $s_t \leftarrow (1 - \beta) \cdot s_{t-1} + \beta \cdot g_t^2$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t / (\sqrt{s_t} + \epsilon)$
  end while
  return $\theta_t$
```

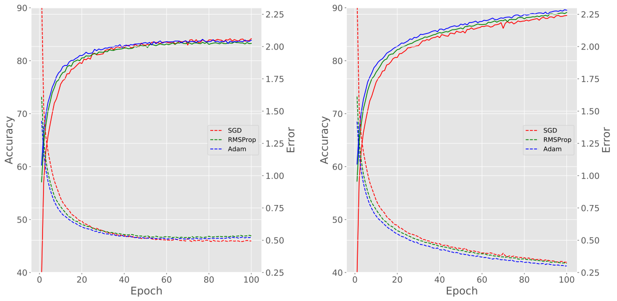
3.2. Learning behaviors

We re-train *Adam* and *RMSProp* with their best hyper-parameter setting for 100 epochs. We then compare the learning curves, for both training and validation, of these three algorithms in fig. 2. The accuracy of them on training, validation and test set are computed in table 3.

From fig. 2, we can see that both *Adam* and *RMSProp* enjoy a faster convergence speed than *SGD*, especially in early epochs, by using the estimates of the moments as described earlier. The usage of decay rate and momentum also facilitates the learning of the models, which results in better training accuracy and lower error. However, the capability of models of generalization to validation and test set are slightly worse with *Adam* and *RMSProp*: higher error and lower accuracy, because adaptive methods always find very different solutions from that of *SGD*, which may lead to poor generalization even though they outperform SGD on training set (Wilson et al., 2017). In section 5, we would study another factor which results in the poor generalization of *Adam*.

4. Cosine annealing learning rate scheduler

It's unreasonable to use the same learning rate throughout the whole training process in *SGD*, where we expect larger steps in former updates and smoother steps in later



Validation Training

Figure 2: Learning curves for different *SGD* (red line), *RMSProp* (green line), and *Adam* (blue line).

Algorithm	Param.	Train acc.	Valid. acc	Test acc.
SGD	α = .01	88.55	**84.18**	**83.18**
RMSProp	α = .0001 β = .90	89.12	83.28	82.27
Adam	α = .0001 $\beta1$ = .50 $\beta2$ = .999	**89.55**	83.87	82.67

Table 3: The accuracy of *SGD*, *RMSProp*, and *Adam*.

updates. A learning rate schedule could be used to anneal the learning rate to decay as increase of training time.

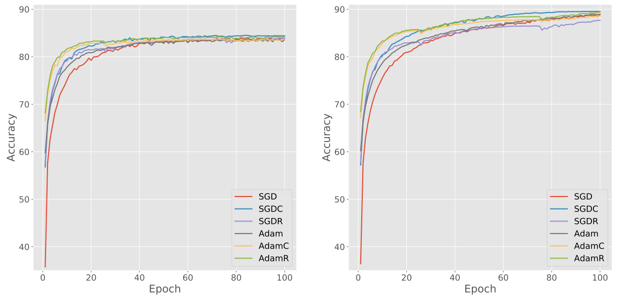
Loshchilov & Hutter (2016) proposed a cosine scheduler, with warm restarts every T_i epochs, which could improve anytime performance. At the n-th epoch, the value of learning rate scaler η_t decays according to eq. (1):

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

where $\eta_{min}^{(i)}$ and $\eta_{max}^{(i)}$ are ranges for the scheduler and T_{cur} accounts for how many epochs have been performed since last restart, which is updated at the onset of each epoch, rather than each batch iteration t as described in (Loshchilov & Hutter, 2016; 2017). To achieve better performance, one could initially set T_i to a small value and scale it by T_{mult} at each restart. Also, the maximum learning rate η_{max} could be decayd by multiplying it with a discount factor at each restart to ensure smoother updates. Cosine annealing with warm restarts is also combined with adaptive method *Adam* in (Loshchilov & Hutter, 2017), which outperforms a fixed learning rate. We explore cosine annealing, with and without warm restarts, for both *SGD* and *Adam* in this section.

4.1. Hyper-parameters selection

Same hyper-parameters in section 3.1 are used for *SGD* and *Adam* in this part. We first evaluate the most efficient couples of η_{min} and η_{max} using cosine annealing without restart to find for both algorithms, namely *AdamC* and *SGDC*. We choose 16 learning rate couples from a logarithmically-spaced grid, and find the best performing one. We then experiment on both algorithms with restarts, namely *AdamR* and *SGDR*, where T_i = 25 and $T_(mult)$ = 3, using η_{min} and η_{max} found in previous step. We evaluate decay factor λ of η_{max} by selecting them from a grid scale [0.5, 1] of size 6. All final hyparameters are shown in table 4.



Validation Training

Figure 3: Accuracy curves for different algorithms.

Algorithms	Param.	Train Acc.	Valid. Acc.	Test Acc.
SGDC	η_{max}=.02 η_{min}=1e-5	89.54	84.40	83.36
AdamC	η_{max}=2.5e-4 η_{min}=1e-6	89.94	84.60	82.83
SGDR	λ =0.8	87.78	83.84	82.57
AdamR	λ =0.7	90.03	84.18	83.17

Table 4: The hyper-parameters (inherited from top to bottom) for algorithms and their performance on training, validation, test set in terms of accuracy percentage.

4.2. Effect of cosine annealing with warm restarts

We compare the performance of the models trained with the 6 algorithms in table 4. We also plot their validation curves during training to study their learning behaviors. As shown in table 4, using cosine annealing improves model performance, except in SGDR, which may result from poor hyperparameter selection. Noteworthy is that, shown in fig. 3, there is a generalization gap for all the algorithms. While annealing and restarts continuously improve training accuracy, they contribute little to improve validation accuracy in later updates. This may result from limited training epochs (Hoffer et al., 2017).

5. Regularization/weight decay in Adam

Although weight decay and L_2 regularization are identical for *SGD*, L_2 regularization are less effective than weight decay for *Adam*. As shown in algorithm 3, while weight decay coefficient is only scaled by learning rate scheduler, L_2 regularization coefficient and weights are scaled and normalized, weights decay proportional to their normalized amplitudes, which make the weights decay weaker (Loshchilov & Hutter, 2017). Whereas in algorithm 1, this two coefficient could be made equivalent by scale the learning rate. In this section, we first compare the effect of L_2 regularization and weight decay in *Adam*. Then we study the behavior of *Adam* with weight decay (*AdamW*), using cosine annealing with warm restarts.

Algorithm 3 *Adam*. *Adam* with L_2 regularization and *Adam* with weight decay (*AdamW*)

```
Require: Learning rate $\alpha$, weight decay/$L_2$ regularization factor $w$
Require: Exponential decay rates for the moment estimates $\beta_1, \beta_2 \in [0, 1)$
Require: Objective function with parameters $\theta$, $f(\theta)$
Require: Initial parameters $\theta_0$
  $t \leftarrow 0$   (Initialize time step)
  $m_0 \leftarrow 0$   (Initialize $1^{st}$ moments)
  $v_0 \leftarrow 0$   (Initialize $2^{nd}$ moments)
  while $\theta_0$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ $+w \cdot \theta_{t-1}$
    $m_t \leftarrow (1 - \beta_1) \cdot m_{t-1} + \beta_1 \cdot g_t$
    $v_t \leftarrow (1 - \beta) \cdot v_{t-1} + \beta \cdot g_t^2$
    $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$
    $\eta_t \leftarrow$ Scheduler(t) (Fixed, decay, or warm restarts)
    $\theta_t \leftarrow \theta_{t-1} - \eta_t \cdot (\alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ $+w \cdot \theta_{t-1}$ )
  end while
  return $\theta_t$ (Resulting parameters)
```

5.1. Regularization vs. weight decay

We choose the decay factor for both technique using *Adam*, with hyper-parameters described in table 4. The best setting of L_2 regularization and weight decay coefficients and their performance are shown in table 5. In (Loshchilov & Hutter, 2017), the authors also propose normalized weight decay, which relieves the dependence of weight decay on training budgets. We compare weight decay with its normalized version using the same values. In fig. 5, we plot the validation accuracy curves of weight decay and its normalized version, which shows that the normalized weight decay is more insensitive and robust to other settings of hyper-parameters and could improve the model performance. Indeed, from the results in table 5 and fig. 4, normalized weight decay achieves the best performance on test data due to its stronger regularization effect and its robustness gained from normalization.

5.2. AdamW with consine annealing & warm restarts

We use the hyper-parameters of best AdamW model in section 5.1, and combine it with cosine annealing and warm restarts, using the hyper-parameter setting suggested in table 4. We compare the learning behaviors of *AdamW* with a fixed learning rate, *AdamW* with cosine annealing, namely *AdamWC*, and *AdamW* with cosine annealing and warm restarts, namely *AdamWR*, as shown in fig. 6. It's interesting to see that *AdamWR* significantly outperform *AdamW* in term of training accuracy. Also, in early steps, the validation curves follow closely with training curves, then slightly stagnate once reaching a threshold. *AdamWR* achieve the best performance on test set.

METHOD	λ	TRAIN ACC.	VALID. ACC	TEST ACC.
L_2	1E-3	88.78	83.37	83.10
WEIGHT DECAY	1E-5	87.32	84.00	83.14
NORM. WD.	1E-4	88.89	84.23	83.41

Table 5: The accuracy of different regularization techniques in terms of accuracy percentage.

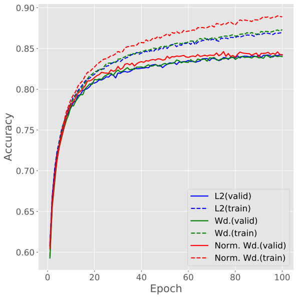


Figure 4: Accuracy curves (train and valid.) for regularization techniques

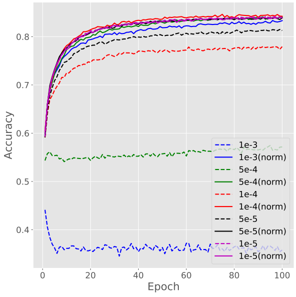


Figure 5: Difference between validatiaon accuracy curves for same weight decay rates

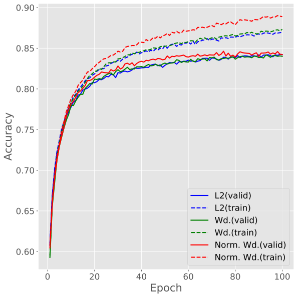


Figure 6: Accuracy curves

METHOD	TEST ACC.
ADAM	82.67
ADAMC	82.83
ADAMR	83.17
ADAMW	83.10
ADAMWC	83.05
ADAMWR	**83.58**

Table 6: Test accuracy

6. Conclusions

From the experiments conducted, we could conclude that, although *Adam* may fit the training set better than *SGD* and speed up the convergence, it exhibits poor generalization as it tends to have a very different solution from *SGD* (Wilson et al., 2017). One could use normalized weight decay and cosine annealer with warm restarts to improve its generalization and training performance. As demonstrated in section 5, the combination of these two techniques achieves the best performance on test set, using the relevant hyper-parameters reported in table 4 and table 5. Future work would focus on the exploration of *Convolutional Neural Networks* and *Recurrent Neural Networks* using adaptive learning methods.

References

Bottou, Léon. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT*, pp. 177–186. Springer, 2010.

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

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.

Geoffrey, E. Hinton and Tijmen, Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Hoffer, Elad, Hubara, Itay, and Soudry, Daniel. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pp. 1731–1741, 2017.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL https://arxiv.org/abs/1412.6980.

Loshchilov, Ilya and Hutter, Frank. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. URL https://arxiv.org/abs/1608.03983.

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

Polyak, Boris T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Robbins, Herbert and Monro, Sutton. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pp. 102–109. Springer, 1985.

Wilson, Ashia C, Roelofs, Rebecca, Stern, Mitchell, Srebro, Nati, and Recht, Benjamin. The marginal value of adaptive gradient methods in machine learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4148–4158. Curran Associates, Inc., 2017.