Step-size Adaptation Using Exponentiated Gradient Updates

Ehsan Amid [1] **Rohan Anil** [1] **Christopher Fifty** [1] **Manfred K. Warmuth** [1]

Abstract

Optimizers like Adam and AdaGrad have been very successful in training large-scale neural networks. Yet, the performance of these methods is heavily dependent on a carefully tuned learning rate schedule. We show that in many large-scale applications, augmenting a given optimizer with an adaptive tuning method of the step-size greatly improves the performance. More precisely, we maintain a global step-size scale for the update as well as a gain factor for each coordinate. We adjust the global scale based on the alignment of the average gradient and the current gradient vectors. A similar approach is used for updating the local gain factors. This type of step-size scale tuning has been done before with gradient descent updates. In this paper, we update the step-size scale and the gain variables with exponentiated gradient updates instead. Experimentally, we show that our approach can achieve compelling accuracy on standard models without using any specially tuned learning rate schedule. We also show the effectiveness of our approach for quickly adapting to distribution shifts in the data during training.

1. Introduction

Successful training of large neural networks heavily depends on the choice of a good optimizer as well as careful tuning of the hyperparameters such as the learning rate, momentum, weight decay, etc. Among the hyperparameters, the learning rate schedule is one of the most important elements for achieving the best accuracy. In many cases, the schedule consists of an initial ramp-up followed by a number of staircase decays throughout the training phase. The schedule is typically tailored manually for the problem and requires re-training the model numerous times.

The most recently proposed optimizers for deep neural networks (AdaGrad (Duchi et al., 2011), RMSProp (Hinton et al., 2014), Adam (Kingma & Ba, 2014), etc.) have been based on adapting the gradient via a (diagonal) preconditioner matrix. However, these techniques still require a carefully tuned learning rate schedule to achieve the optimal performance. Furthermore, little has been done for adapting the per-coordinate or the overall step-size. The common idea among the few available approaches (Jacobs, 1988; Riedmiller & Braun, 1993; Schraudolph, 1999; Baydin et al., 2018) is to move faster along directions that are consistently making progress and punish those that alternate often. However, for the previous methods, 1) the formulation is not general enough to be compatible with different optimizers; and 2), the update equations are based on inferior heuristics or in some cases use the incorrect gradients (Schraudolph, 1999) (as we will discuss later).

In this paper, we aim to unify such approaches in a more rigorous manner. Our formulation is versatile and accepts any pre-conditioned gradient by an adaptive gradient optimizer as input. Thus, it can be coupled with a wide range of commonly used optimizers. Concretely, we propose an abstraction in the form of a momentum update by passing the pre-conditioned gradient, proposed by an arbitrary internal optimizer to the meta algorithm. We then make the step-size adaptive by introducing the following hyperparameters: one overall step-size scale as well as local gain factors for each coordinate. The scale and gains are non-negative and trained using the Unnormalized Exponentiated Gradient (EGU) updates (Kivinen & Warmuth, 1997). As a brief introduction, EGU minimizes a function f by adding a relative entropy (a.k.a. Kullback-Leibler) divergence (Kullback & Leibler, 1951) as an *inertia* term. The goal of adding the inertia term is to keep the updated parameters $\boldsymbol{\theta}^{t+1}$ close to the previous parameters $\boldsymbol{\theta}^t$ at step t:

$$\boldsymbol{\theta}^{t+1} = \operatorname{argmin}_{\tilde{\boldsymbol{\theta}} \succcurlyeq \mathbf{0}} \big\{ {}^1\!/\!_\eta \, D_{\text{RE}}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\theta}^t) + f(\tilde{\boldsymbol{\theta}}) \big\}\,, \quad (1)$$

where $\eta > 0$ is a learning rate parameter and

$$D_{\text{RE}}(\boldsymbol{u}, \boldsymbol{v}) = \sum_i \big(u_i \log \frac{u_i}{v_i} - u_i + v_i \big)\,.$$

The update multiplies each parameter by an exponentiated gradient factor:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t \odot \exp\big(-\,\eta\, \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^t) \big)\,, \qquad \text{(EGU)} \quad (2)$$

[1]Google Research, Brain Team, Mountain View, CA. Correspondence to: Rohan Anil <rohananil@google.com>.

Workshop on "Beyond first-order methods in ML systems" at the 37th International Conference on Machine Learning, Vienna, Austria, 2020. Copyright 2020 by the author(s).

where $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^t)$ denotes the gradient of the objective function $f(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}^t$ and $\odot$ denotes element-wise product.[1] The multiplicative form of the update ensures $\boldsymbol{\theta}^{t+1} \succcurlyeq \mathbf{0}$ at any time. The properties of the Exponentiated Gradient family of updates[2] have been studied extensively in the online learning literature (See e.g. (Kivinen & Warmuth, 1997; Freund & Schapire, 1997; Kivinen et al., 1997; Singer & Warmuth, 1999; Kivinen et al., 2006; Warmuth & Kuzmin, 2008)). Specifically, it has been shown that the EGU update converges significantly faster than gradient descent in cases when only a small subset of the dimensions are relevant (Littlestone, 1988; Vishwanathan & Warmuth, 2005; Warmuth, 2007; Amid & Warmuth, 2020). As a result, EGU is extremely efficient in discovering the relevant dimensions while immediately damping those that are irrelevant. Also, the EGU update naturally maintains the non-negativity of the parameters. Finally, the multiplicative form of the update allows exploring a wider range of values for the parameters more rapidly.

1.1. Related Work

Introducing per-coordinate gains dates back to the Delta-Bar-Delta (DBD) method (Jacobs, 1988) where the set of gains are adaptively updated using the sign agreements of the current gradient and an exponential running average (EMA) of the past gradients. DBD uses a mixture of additive and multiplicative updates where the gains for the coordinates with agreeing gradient signs are increased by a constant amount, otherwise damped by a multiplicative factor. Later, these ideas were extended to different settings (Minai & Williams, 1990; Sutton, 1995). More relevantly, a local gain adaptation method was introduced in (Schraudolph, 1999) where the goal was to update the gains using the EGU updates. However, the wrong gradient term (gradient w.r.t. log-gains instead of gradient w.r.t. gains) was used in the final update.

2. A Meta Algorithm for Adaptive Step-size

Our adaptive learning rate meta algorithm[3] accepts as input a pre-conditioned gradient from an *internal optimizer* $\mathcal{D}$ based on the value of the current weight parameters $\boldsymbol{w}^t$ (and possibly, the cumulative statistic of all the previous steps). The internal optimizer only interacts with the meta algorithm via the values of the parameter and its role is to only generate the pre-conditioned gradients. Let $\tilde{\boldsymbol{g}}^t := \widetilde{\nabla}_{\boldsymbol{w}} L(\boldsymbol{w}^t | \mathcal{X}^t)$ denote the pre-conditioned gradient generated by the internal optimizer at step t using the batch of data $\mathcal{X}^t$. Our **"Funnel"** meta algorithm applies the following update:

$$\boxed{\begin{aligned}\boldsymbol{\nu}^{t+1} &= \mu\, \boldsymbol{\nu}^t + \eta\, (\boldsymbol{p}^{t+1} \odot \tilde{\boldsymbol{g}}^t)\,,\\ \boldsymbol{w}^{t+1} &= \boldsymbol{w}^t - s^{t+1}\, \boldsymbol{\nu}^{t+1}\,.\end{aligned}} \tag{3}$$

The update (3) in fact resembles a heavy-ball momentum update with *base learning rate* η and *momentum* hyper-parameter μ, with the addition of two extra elements: 1) a non-negative per-coordinate gain vector $\boldsymbol{p} \succcurlyeq \mathbf{0}$ which component-wise multiplies the pre-conditioned gradient vector, and 2) a non-negative step-size *scale* $s \geq 0$ which scales the final step. The goal of the gain hyperparameter $\boldsymbol{p}$ is to independently scale each coordinate of the pre-conditioned gradient vector. On the other hand, the step-size scale s adjust the final update that is added to the parameters.

The gain as well as the scale hyperparameters are updated along with the weights at each step. Since both gains and scale are non-negative, a natural choice for the update is the EGU update (2). The multiplicative form of EGU update allows these hyperparameters to effectively adapt to the dynamics of training by changing in a wider range of values more rapidly. The updates are applied by first calculating the gradient of the loss w.r.t. each hyperparameter. That is,

$$\begin{aligned}\nabla_{\boldsymbol{p}} L(\boldsymbol{w}^t | \mathcal{X}) &= -\nabla_{\boldsymbol{w}} L(\boldsymbol{w}^t | \mathcal{X}) \odot s^{t+1}\, \frac{\partial \boldsymbol{\nu}^{t+1}}{\partial \boldsymbol{p}}\\ &\approx -s^{t+1} \nabla_{\boldsymbol{w}} L(\boldsymbol{w}^t | \mathcal{X}) \odot \nabla_{\boldsymbol{w}} L(\boldsymbol{w}^{t-1} | \mathcal{X})\,.\end{aligned}$$

where we omit the long-term dependencies on the gain hyperparameters. Let $\boldsymbol{g}^t := \nabla_{\boldsymbol{w}} L(\boldsymbol{w}^t | \mathcal{X}^t)$ denote the gradient of the loss using the batch of data $\mathcal{X}^t$. We also remove the s^{t+1} term from the gradient to reduce the interdependency of the gains and the scale. Thus, the exponentiated gradient gain update becomes

$$\boldsymbol{p}^{t+1} = \boldsymbol{p}^t \odot \exp\big(\gamma_p\, \boldsymbol{g}^t \odot \tilde{\boldsymbol{g}}^{t-1}\big)\,, \tag{4}$$

where $\gamma_p \geq 0$ is the gain learning rate hyperparameter. Notice that update (4) depends on the value of the gradient on the current batch $\mathcal{X}^t$ and the value of the pre-conditioned gradient at the previous batch $\mathcal{X}^{t-1}$. To account for the stochasticity of the gradients due to different batches of data, we replace the second term in the gradient by an EMA of all the past pre-conditioned gradients, that is,

$$\boldsymbol{p}^{t+1} = \boldsymbol{p}^t \odot \exp\big(\gamma_p\, \boldsymbol{g}^t \odot \tilde{\boldsymbol{m}}^t\big)\,, \tag{5}$$

where $\boldsymbol{m}^{t+1} = \beta\, \boldsymbol{m}^t + (1-\beta)\, \tilde{\boldsymbol{g}}^t$ and $\tilde{\boldsymbol{m}}^{t+1} = \frac{\boldsymbol{m}^{t+1}}{1-\beta^{t+1}}$. The hyperparameter $0 \leq \beta \leq 1$ is the decay factor for the pre-conditioned gradient EMA and $\tilde{\boldsymbol{m}}^t$ corrects the initialization bias of $\boldsymbol{m}^t$ at zero. Similarly, for the the step-size scale hyperparameter s, we have

$$\nabla_s L(\boldsymbol{w}^t | \mathcal{X}) = -\nabla_{\boldsymbol{w}} L(\boldsymbol{w}^t | \mathcal{X}) \cdot \boldsymbol{\nu}^t\,.$$

[1]The exact minimization of (1) uses the gradient $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^{t+1})$ which is then approximated by $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^t)$ in the EGU update.

[2]Most research has focused on the normalized version of the Exponentiated Gradient update (abbreviated as EG) that divides the weights of update (2) by their total sum.

[3]Code available online at: `https://users.soe.ucsc.edu/~eamid/funnel.html`.

Algorithm 1 Funnel SGD with Momentum

```
Input: Loss function $L$, internal optimizer $\mathcal{D}$, initial parameter $\boldsymbol{w}^0$, base learning rate $\eta$, pre-conditioned gradient EMA decay factor $\beta$, gain and step-scale learning rate hyperparameters $(\gamma_p, \gamma_s)$
$t, \boldsymbol{m}^0, \boldsymbol{p}^0, s^0 \leftarrow 0, \boldsymbol{0}, \boldsymbol{1}, 1$ {Initialization}
repeat
  Obtain $\boldsymbol{g}^t = \nabla_{\boldsymbol{w}} L(\boldsymbol{w}^t | \mathcal{X}^t)$ {Gradient}
  Obtain $\tilde{\boldsymbol{g}}^t = \tilde{\nabla}_{\boldsymbol{w}} L(\boldsymbol{w}^t | \mathcal{X}^t)$ from $\mathcal{D}(L, \boldsymbol{w}^t, \mathcal{X}^t)$ {Pre-conditioned grad.}
  $\boldsymbol{p}^{t+1} \leftarrow \begin{cases} \boldsymbol{p}^t \odot \exp\left(\gamma_p\, \boldsymbol{g}^t \odot \tilde{\boldsymbol{m}}^t\right) & \text{(Unnormalized)} \\ \boldsymbol{p}^t \odot \exp\left(\gamma_p\, \mathrm{sgn}(\boldsymbol{g}^t) \odot \mathrm{sgn}(\boldsymbol{m}^t)\right) & \text{(Normalized)} \end{cases}$
  $s^{t+1} \leftarrow \begin{cases} s^t \exp\left(\gamma_s\, \boldsymbol{g}^t \cdot \boldsymbol{\nu}^t\right) & \text{(Unnormalized)} \\ s^t \exp\left(\gamma_s\, \frac{\boldsymbol{g}^t}{\|\boldsymbol{g}^t\|} \cdot \frac{\boldsymbol{\nu}^t}{\|\boldsymbol{\nu}^t\|}\right) & \text{(Normalized)} \end{cases}$
  $\boldsymbol{m}^{t+1} \leftarrow \beta\, \boldsymbol{m}^t + (1 - \beta)\, \tilde{\boldsymbol{g}}^t$
  $\boldsymbol{\nu}^{t+1} \leftarrow \mu\, \boldsymbol{\nu}^t + \eta\, (\boldsymbol{p}^{t+1} \odot \tilde{\boldsymbol{g}}^t)$
  $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - s^{t+1}\, \boldsymbol{\nu}^{t+1}$ {Parameter update}
  $t \leftarrow t + 1$
until $\boldsymbol{w}^t$ not converged
return $\boldsymbol{w}^t$
```

Thus, applying the EGU updates results in

$$s^{t+1} = s^t \exp\left(\gamma_s\, \boldsymbol{g}^t \cdot \boldsymbol{\nu}^t\right), \tag{6}$$

where $\gamma_s \geq 0$ is the scale learning rate hyperparameter. The pseudo-code for the Funnel Stochastic Gradient Descent with Momentum is shown in Algorithm 1, in which we also include a normalized version of the updates.

3. Experiments

In this section, we show two cases where our adaptive step-size Funnel method proves to be extremely effective. In the first part of the experiments, we consider the problem of distribution shift in the data during training. Next, we consider the setting where we remove the learning rate schedule for training of large-scale models for image classification. In all experiments, we use the normalized version of the scale and gain updates (see Algorithm 1).

3.1. Distribution Shift

Adaptive optimization methods work well on standard datasets where the distribution of the data is fixed. Typical tuning procedure for static datasets involves a decaying learning rate schedule in case of Adam or implicit decay schedule of AdaGrad (due to accumulation of gradient statistics). On real world problems where models are training on a stream of freshly arriving data (e.g. click through rate prediction in online advertisement or recommender systems), there is a natural shift in the distribution in the dataset over time (e.g. user preferences can change). This requires the optimization method to be adaptive w.r.t. a changing distribution.

To illustrate the advantage of our proposed step-size adaptive mechanism in such cases, we simulate distribution shift on the MNIST dataset of handwritten digits (LeCun et al., 2010). We split the train, validation, and test into three disjoint sets. For each set, we replace pixel values less than 10^{-2} with a random uniform from $[0, 1]$ and rotate the images by 0, 45, and 90 degrees each as shown in Figure 1. We train a logistic regression model with AdaGrad (Duchi et al., 2011) as well with Funnel (with AdaGrad as the internal optimizer) at batch size 10k, for 100 epochs for each set sequentially (90 degrees first, followed by 0 degrees, and finally 45 degree). The results, presented in Figure 2, indicate that AdaGrad's performance is deteriorated when we switch the training set where as with Funnel higher top-1 accuracy is obtained. We also show the evolution of the learning rate scale, and the per-coordinate gains throughout the training and find that the optimization method is able to adjust these hyperparameters in a data dependent way (see Figure 2(b) and Figure 2(c)).

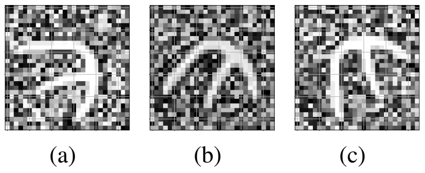


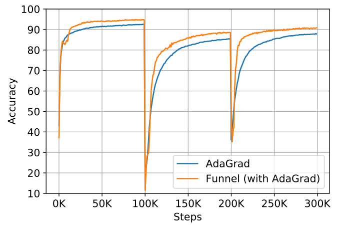
Figure 1. Samples from the rotated MNIST datasets: random background with (a) 0, (b) 45, and (c) 90 degrees rotation.

3.2. Adaptive Learning Rate Schedule

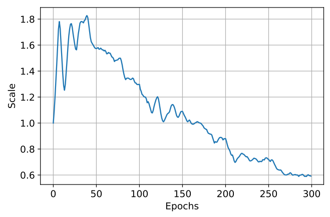
In this section, we conduct experiments on large-scale convolutional neural networks where the baseline model is trained with a highly tuned learning rate schedule. We compare the performance of different optimizer on the same network when the learning rate schedule is removed. For each experiment, we tune the remaining hyperparameters of the optimizers independently. We also repeat each experiment for the best tuning 5 times and average the results. We consider the following models for the experiments: 1) MobileNetV1 model on CIFAR-10 dataset, and 2) ResNet50 model on the ImageNet dataset.

For the MobileNetV1 trained on the CIFAR10 model, we consider the SGD Momentum optimizer as the baseline optimizer. The baseline learning rate schedule consists of two staircase decays without any initial ramp-ups. We train the baseline model for 150k steps with a batch size of 50.

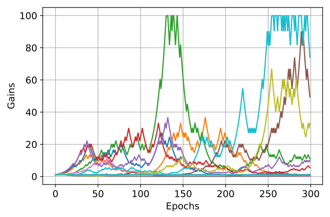
Next, we remove the learning rate schedule and train the model using the SGD Momentum optimizer as well as the Funnel SGD Momentum. For the vanilla Momentum, we use the same learning rate and momentum values as the baseline. We also use the same learning rate and momentum values for Funnel and tune the gain and scale learning rate values in the range $[10^{-5}, 10^{-3}]$. We set $\beta = 0.9$ for the gradient EMA. We also allow the gains and the scale to vary in the range $[0, 10^3]$. We repeat each experiment 5 times for 150k iterations and using the same batch size of 50. The best performance for Funnel is achieved with $(\gamma_p, \gamma_s) = (10^{-4}, 10^{-3})$. The results are shown in Table 1. As can



(a)

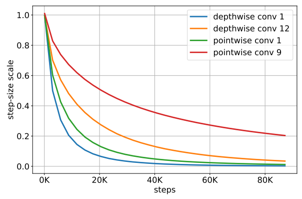


(b)

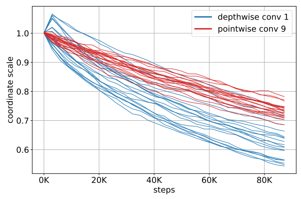


(c)

Figure 2. (a) top-1 accuracy on shifting MNIST dataset. Every 100k steps (100 epochs) we shift the datasets. (b) Evolution of learning rate scales, and (c) gain values for some of the parameters throughout training.



(a)



(b)

Figure 3. Scale and gain hyperparameters of the MobileNetV1 model: (a) step-size scales for different layers, (b) gains for a subset of coordinates in two different layers. Funnel successfully discovers a decay schedule for all the layers, with a different decay rate. Also, notice that using Funnel the gains vary at a different rate for each layer and each coordinate. Also, some of the gains ramp up initially for the first ∼5k steps before starting to decay.

be seen from the table, the top-1 accuracy of the baseline Momentum model drops by around 5% by removing the learning rate schedule. However, our Funnel method can match the baseline performance without using a learning rate schedule. We plot the scale and gain values for a subset of the layers in Figure 3. In Figure 3(a), we plot the scales for 2 depthwise and 2 pointwise convolutional layers. As can be seen from the figure, Funnel is able to recover a decay schedule for all the layers, with a different decay rate. In Figure 3(b), we show a gains for a subset of coordinates for a depthwise as well as a pointwise convolutional layer. Notice that the gains vary at different rates for each layer and each coordinate. Also, some of the gains ramp up initially for the first ∼5k steps before starting to decay.

We also consider the ResNet50 model trained on the ImageNet dataset. The baseline optimizer corresponds to an SGD Momentum with an initial ramp-up followed by a stair-case decay learning rate schedule. We train the model for 100 epochs using a batch size of 4096. Next, we retrain the model using SGD Momentum, AdaGrad-EMA, and Funneled SGD Momentum optimizers while removing the learning rate schedule. The reason for choosing AdaGrad is the fact the the effective learning rate is naturally decayed for this optimizer, thus mimicking a decaying schedule. Note that AdaGrad-EMA is a slight variant of AdaGrad where we apply an EMA on the pre-conditioned gradients. The original formulation of AdaGrad performs poorly in this setting. We similarly do a hyperparameter search for all the models. The best performing learning rate for AdaGrad-EMA is achieved at 10^{-3}. For Funnel, we use the original learning rate as the baseline ($\eta = 0.1$) and set $(\gamma_p, \gamma_s) = (10^{-4}, 5 \times 10^{-3})$. The results are shown in Table 2. As can be seen from the table, Funnel achieves the best performance among the other optimizers when the learning rate schedule is removed.

Table 1. MobileNetV1 top-1 and top-5 accuracy on the CIFAR-10 dataset: the top baseline result is obtained using a learning rate schedule. We compare the performance of different optimizers when the learning rate schedule is removed.

Method	Top-1 Test Accuracy	Top-5 Test Accuracy
Momentum (w/ schedule)	89.51 ± 0.18	99.35 ± 0.05
Momentum (w/o schedule)	84.14 ± 0.49	98.93 ± 0.35
Funnel-SGD (w/o schedule)	$\mathbf{89.61 \pm 0.28}$	$\mathbf{98.94 \pm 0.26}$

Table 2. ResNet50 top-1 and top-5 accuracy on the ImageNet dataset: the top baseline result is obtained using a learning rate schedule. We compare the performance of different optimizers when the learning rate schedule is removed.

Method	Top-1 Test Accuracy	Top-5 Test Accuracy
Momentum (w/ schedule)	76.57 ± 0.16	93.21 ± 0.04
Momentum (w/o schedule)	53.80 ± 0.86	78.67 ± 0.74
AdaGrad-EMA (w/o schedule)	70.70 ± 0.24	89.74 ± 0.17
Funnel-SGD (w/o schedule)	$\mathbf{72.39 \pm 0.09}$	$\mathbf{90.97 \pm 0.06}$

4. Conclusion and Future Work

We provided an adaptive method for unifying the existing ideas in the domain of learning rate adaptation in a more rigorous manner. This is done by introducing a per-coordinate gain as well as an overall step-size scale and updating these hyperparameter using the well-known unnormalized exponentiated gradient updates. Our meta algorithm can easily adapt to many widely used optimizers, without special modification. We present very promising experimental results for our new adaptive method, namely, for adapting to distribution shift and finding effective learning rate schedules.

A long-term goal is to also replace the common gradient descent based optimizers by updates from the exponentiated gradient family. For this, a more rigorous study with a focus on large-scale applications is needed.

References

Amid, E. and Warmuth, M. K. Winnowing with gradient descent. *Conference on Learning Theory (COLT)*, 2020.

Baydin, A. G., Cornish, R., Martínez-Rubio, D., Schmidt, M., and Wood, F. Online learning rate adaptation with hypergradient descent. In *6th International Conference on Learning Representations, ICLR*, 2018.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

Freund, Y. and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Hinton, G., Srivastava, N., and Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2014.

Jacobs, R. A. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

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

Kivinen, J. and Warmuth, M. K. Exponentiated gradient versus gradient descent for linear predictors. *Inf. Comput.*, 132(1):1–63, 1997.

Kivinen, J., Warmuth, M. K., and Auer, P. The perceptron algorithm versus winnow: linear versus logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, 97(1-2):325–343, 1997.

Kivinen, J., Warmuth, M. K., and Hassibi, B. The p-norm generalization of the LMS algorithm for adaptive filtering. *IEEE Transactions on Signal Processing*, 54(5):1782–1793, 2006.

Kullback, S. and Leibler, R. A. On information and sufficiency. *The annals of mathematical statistics*, 22(1): 79–86, 1951.

LeCun, Y., Cortes, C., and Burges, C. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Littlestone, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning*, 2(4):285–318, 1988.

Minai, A. A. and Williams, R. D. Back-propagation heuristics: a study of the extended Delta-Bar-Delta algorithm. In *1990 IJCNN International Joint Conference on Neural Networks*, pp. 595–600 vol.1, 1990.

Riedmiller, M. and Braun, H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE international conference on neural networks*, pp. 586–591. IEEE, 1993.

Schraudolph, N. N. Local gain adaptation in stochastic gradient descent. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pp. 569–574 vol.2, 1999.

Singer, Y. and Warmuth, M. K. Batch and on-line parameter estimation of Gaussian mixtures based on the joint entropy. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 578–584, 1999.

Sutton, R. Adapting bias by gradient descent: An incremental version of Delta-Bar-Delta. *Proceedings of the Tenth National Conference on Artificial Intelligence*, 06 1995.

Vishwanathan, S. and Warmuth, M. Leaving the span. In *Proceedings of the 18th Annual Conference on Learning Theory (COLT 05)*, Bertinoro, Italy, June 2005. Springer.

Warmuth, M. K. Winnowing subspaces. In *Proceedings of the 24th International Conference on Machine Learning*, ICML'07, pp. 999–1006, New York, NY, USA, 2007. ACM.

Warmuth, M. K. and Kuzmin, D. Randomized online PCA algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9 (Oct):2287–2320, 2008.