Follow the Moving Leader in Deep Learning

Shuai Zheng [1] **James T. Kwok** [1]

Abstract

Deep networks are highly nonlinear and difficult to optimize. During training, the parameter iterate may move from one local basin to another, or the data distribution may even change. Inspired by the close connection between stochastic optimization and online learning, we propose a variant of the *follow the regularized leader* (FTRL) algorithm called *follow the moving leader* (FTML). Unlike the FTRL family of algorithms, the recent samples are weighted more heavily in each iteration and so FTML can adapt more quickly to changes. We show that FTML enjoys the nice properties of RMSprop and Adam, while avoiding their pitfalls. Experimental results on a number of deep learning models and tasks demonstrate that FTML converges quickly, and outperforms other state-of-the-art optimizers.

1. Introduction

Recently, deep learning has emerged as a powerful and popular class of machine learning algorithms. Well-known examples include the convolutional neural network (LeCun et al., 1998), long short term memory (Hochreiter & Schmidhuber, 1997), memory network (Weston et al., 2014), and deep Q-network (Mnih et al., 2015). These models have achieved remarkable performance on various difficult tasks such as image classification (He et al., 2016), speech recognition (Graves et al., 2013), natural language understanding (Bahdanau et al., 2015; Sukhbaatar et al., 2015), and game playing (Silver et al., 2016).

Deep network is a highly nonlinear model with typically millions of parameters (Hinton et al., 2006). Thus, it is imperative to design scalable and effective solvers. However, training deep networks is difficult as the optimization can suffer from pathological curvature and get stuck in local minima (Martens, 2010). Moreover, every critical point that is not a global minimum is a saddle point (Kawaguchi, 2016), which can significantly slow down training. Second-order information is useful in that it reflects local curvature of the error surface. However, a direct computation of the Hessian is computationally infeasible. Martens (2010) introduced Hessian-free optimization, a variant of truncated-Newton methods that relies on using the linear conjugate gradient to avoid computing the Hessian. Dauphin et al. (2014) proposed to use the absolute Hessian to escape from saddle points. However, these methods still require higher computational costs.

Recent advances in deep learning optimization focus mainly on stochastic gradient descent (SGD) (Bottou, 1998) and its variants (Sutskever et al., 2013). However, SGD requires careful stepsize tuning, which is difficult as different weights have vastly different gradients (in terms of both magnitude and direction). On the other hand, online learning (Zinkevich, 2003), which is closely related to stochastic optimization, has been extensively studied in the past decade. Well-known algorithms include follow the regularized leader (FTRL) (Kalai & Vempala, 2005), follow the proximally-regularized leader (FTPRL) (McMahan & Streeter, 2010) and their variants (Duchi & Singer, 2009; Duchi et al., 2011; Shalev-Shwartz, 2012; Xiao, 2010). In particular, adaptive gradient descent (Adagrad) (Duchi et al., 2011) uses an adaptive per-coordinate stepsize. On convex problems, it has been shown both theoretically and empirically that Adagrad is especially efficient on high-dimensional data (Duchi et al., 2011; McMahan et al., 2013). When used on deep networks, Adagrad also demonstrates significantly better performance than SGD (Dean et al., 2012). However, in Adagrad, the variance estimate underlying the adaptive stepsize is based on accumulating all past (squared) gradients. This becomes infinitesimally small as training proceeds. In more recent algorithms, such as RMSprop (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2015), the variance is estimated by an exponentially decaying average of the squared gradients.

Another problem with the FTRL family of algorithms is that in each round, the learner has to solve an optimization problem that considers the sum of all previous gradients.

[1] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong. Correspondence to: Shuai Zheng <szhengac@cse.ust.hk>, James T. Kwok <jamesk@cse.ust.hk>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

For highly nonconvex models such as the deep network, the parameter iterate may move from one local basin to another. Gradients that are due to samples in the distant past are less informative than those from the recent ones. In applications where the data distribution is changing (as in deep reinforcement learning), this may impede parameter adaptation to the environment.

To alleviate this problem, we propose a FTPRL variant that reweighs the learning subproblems in each iteration. The proposed algorithm, which will be called *follow the moving leader* (FTML), shows strong connections with popular deep learning optimizers such as RMSprop and Adam. Experiments on various deep learning models demonstrate that FTML outperforms or at least has comparable convergence performance with state-of-the-art solvers.

The rest of this paper is organized as follows. Section 2 first gives a brief review on FTRL and other solvers for deep learning. Section 3 presents the proposed FTML. Experimental results are shown in Section 4, and the last section gives some concluding remarks.

Notation. For a vector $x \in \mathbb{R}^d$, $\|x\| = \sqrt{\sum_{i=1}^d x_i^2}$, diag$(x)$ is a diagonal matrix with x on its diagonal, $\sqrt{x}$ is the element-wise square root of x, x^2 denotes the Hadamard (elementwise) product $x \odot x$, and $\|x\|_Q^2 = x^T Q x$, where Q is a symmetric matrix. For any two vectors x and y, x/y, and $\langle x, y\rangle$ denote the elementwise division and dot product, respectively. For a matrix X, $X^2 = XX$, and diag(X) is a vector with the diagonal of X as its elements. For t vectors $\{x_1, \dots, x_t\}$, $x_{1:t} = \sum_{i=1}^t x_i$, and $x_{1:t}^2 = \sum_{i=1}^t x_i^2$. For t matrices $\{X_1, \dots, X_t\}$, $X_{1:t} = \sum_{i=1}^t X_i$.

2. Related Work

2.1. Follow the Regularized Leader and its Variants

In online learning, the learner observes a sequence of functions f_i's, which can be deterministic, stochastic, or even adversarially chosen. Let $\Theta \subseteq \mathbb{R}^d$ be a convex compact set. At round t, the learner picks a predictor $\theta_{t-1} \in \Theta$, and the adversary picks a loss f_t. The learner then suffers a loss $f_t(\theta_{t-1})$. The goal of the learner is to minimize the cumulative loss suffered over the course of T rounds. In online convex learning, f_t is assumed to be convex.

Two popular online learning algorithms are the *follow the regularized leader* (FTRL) (Kalai & Vempala, 2005; Shalev-Shwartz, 2012), and its variant *follow the proximally-regularized leader* (FTPRL) (McMahan & Streeter, 2010). Both achieve the optimal $O(\sqrt{T})$ regret, where T is the number of rounds (Shalev-Shwartz, 2012). Other FTRL-like algorithms include *regularized dual averaging* (RDA) (Xiao, 2010) as well as its adaptive variant presented in (Duchi et al., 2011). Gradient descent style algorithms like *online forward and backward splitting* (FOBOS) (Duchi & Singer, 2009) and *adaptive gradient descent* (Adagrad) (Duchi et al., 2011) can also be expressed as special cases of the FTRL family (McMahan, 2011).

At round t, FTRL generates the next iterate θ_t by solving the optimization problem:

$$\theta_t = \arg\min_{\theta \in \Theta} \sum_{i=1}^t \left(\langle g_i, \theta \rangle + \frac{\alpha_t}{2}\|\theta\|^2 \right),$$

where g_t is a subgradient of f_t at θ_{t-1} (usually, $\theta_0 = 0$), and α_t is the regularization parameter at round t. Note that the regularization is centered at the origin. McMahan & Streeter (2010) generalizes this to FTPRL by centering regularization at each iterate θ_{i-1} as in online gradient descent and online mirror descent (Cesa-Bianchi & Lugosi, 2006),

$$\theta_t = \arg\min_{\theta \in \Theta} \sum_{i=1}^t \left(\langle g_i, \theta \rangle + \frac{1}{2}\|\theta - \theta_{i-1}\|_{Q_i}^2 \right), \tag{1}$$

where Q_i is a full or diagonal positive semidefinite matrix, and $\|\theta - \theta_{i-1}\|_{Q_i}$ is the corresponding Mahalanobis distance between θ and θ_{i-1}. When Q_i is diagonal, each of its entries controls the learning rate in the corresponding dimension. When $\Theta = \mathbb{R}^d$, θ_t can be obtained in closed-form (McMahan, 2011):

$$\theta_t = \theta_{t-1} - Q_{1:t}^{-1} g_t. \tag{2}$$

When

$$Q_t = \frac{1}{\eta}\text{diag}\left(\sqrt{g_{1:t}^2} - \sqrt{g_{1:t-1}^2} \right), \tag{3}$$

where $\eta > 0$ is the stepsize, (2) becomes the update rule of Adagrad (Duchi et al., 2011)

$$\theta_t = \theta_{t-1} - \text{diag}\left(\frac{\eta}{\sqrt{g_{1:t}^2} + \epsilon \mathbf{1}} \right) g_t. \tag{4}$$

Here, $\epsilon > 0$ (usually a very small number) is used to avoid division by zero, and $\mathbf{1}$ is the vector of all 1's.

In general, all these algorithms satisfy (McMahan & Streeter, 2010):

$$Q_{1:t} = \text{diag}\left(\frac{1}{\eta} \left(\sqrt{g_{1:t}^2} + \epsilon \mathbf{1} \right) \right). \tag{5}$$

It can be shown that this setting is optimal within a factor of $\sqrt{2}$ of the best possible regret bound for any non-increasing per-coordinate learning rate schedule (McMahan & Streeter, 2010).

2.2. Adaptive Learning Rate in Deep Learning

In training deep networks, different weights may have vastly different gradients (in terms of both magnitude and direction). Hence, using a per-coordinate learning rate as in Adagrad can significantly improve performance over standard SGD (Dean et al., 2012). However, a caveat is that Adagrad suffers from diminishing stepsize. As optimization proceeds, the accumulated squared gradient $g_{1:t}^2$ in (5) becomes larger and larger, making training difficult.

To alleviate this problem, a number of algorithms have been proposed (Zeiler, 2012; Tieleman & Hinton, 2012; Kingma & Ba, 2015). Typically, they employ an average of the past squared gradients (i.e., $v_t = \sum_{i=1}^t \alpha_{i,t} g_i^2$, where $\alpha_{i,t} \in [0,1]$), which is exponentially decaying. For example, RMSprop (Tieleman & Hinton, 2012) uses

$$v_i = \beta v_{i-1} + (1-\beta) g_i^2, \tag{6}$$

where β is close to 1, and the corresponding $\alpha_{i,t}$ is $(1-\beta)\beta^{t-i}$. This v_t can then be used to replace $g_{1:t}^2$, and the update in (4) becomes

$$\theta_t = \theta_{t-1} - \text{diag}\left(\frac{\eta}{\sqrt{v_t} + \epsilon \mathbf{1}}\right) g_t. \tag{7}$$

Zeiler (2012) further argues that the parameter and update should have the same unit, and modifies (7) to the Adadelta update rule:

$$\theta_t = \theta_{t-1} - \text{diag}\left(\frac{\sqrt{u_{t-1}} + \epsilon \mathbf{1}}{\sqrt{v_t} + \epsilon \mathbf{1}}\right) g_t,$$

where $u_{t-1} = \sum_{i=0}^{t-1} \alpha_{i,t-1} (\triangle\theta_i)^2$, and $\triangle\theta_t = \theta_t - \theta_{t-1}$ with $\triangle\theta_0 = 0$.

As v_0 in (6) is often initialized to 0, the bias has to be corrected. Adam (Kingma & Ba, 2015) uses the variance estimate $v_t/(1-\beta^t)$ (which corresponds to $\alpha_{i,t} = (1-\beta)\beta^{t-i}/(1-\beta^t)$).

Another recent proposal is the equilibrated stochastic gradient descent (Dauphin et al., 2015). It uses the variance estimate $v_t = v_{t-1} + (H_t \zeta_t)^2$, where H_t is the Hessian and $\zeta_t \sim \mathcal{N}(0,1)$. It is shown that $(H_t\zeta_t)^2$ is an unbiased estimator of $\sqrt{\text{diag}(H_t^2)}$, which serves as the Jacobi preconditioner of the absolute Hessian. Computation of the Hessian can be avoided by using the R-operator (Schraudolph, 2002), though it still costs roughly twice that of standard backpropagation.

3. Follow the Moving Leader

Recall that at round t, FTRL generates the next iterate θ_t as

$$\theta_t = \arg\min_{\theta \in \Theta} \sum_{i=1}^t P_i(\theta), \tag{8}$$

where $P_i(\theta) = \langle g_i, \theta \rangle + \frac{1}{2}\|\theta - \theta_{i-1}\|_{Q_i}^2$. Note that all P_i's have the same weight. However, for highly nonconvex models such as the deep network, the parameter iterate may move from one local basin to another. P_i's that are due to samples in the distant past are less informative than those from the recent ones.

3.1. Weighting the Components

To alleviate this problem, one may consider only P_i's in a recent window. However, a large memory is needed for its implementation. A simpler alternative is by using an exponential moving average of the P_i's: $S_i = \beta_1 S_{i-1} + (1-\beta_1) P_i$, where $\beta_1 \in [0,1)$ and $S_0 = 0$. This can be easily rewritten as $S_t = (1-\beta_1)\sum_{i=1}^t \beta_1^{t-i} P_i$. Instead of minimizing (8), we have

$$\theta_t = \arg\min_{\theta \in \Theta} \sum_{i=1}^t w_{i,t} P_i(\theta), \tag{9}$$

where the weights

$$w_{i,t} = \frac{(1-\beta_1)\beta_1^{t-i}}{1-\beta_1^t} \tag{10}$$

are normalized to sum to 1. The denominator $1-\beta_1^t$ plays a similar role as bias correction in Adam. When $\beta_1 = 0$, $w_{i,t} = 0$ for $i < t$, and $w_{t,t} = 1$. Thus, (9) reduces to $\min_{\theta\in\Theta} P_t(\theta)$. When $\beta_1 \to 1$, the following Lemma shows that all P_i's are weighted equally, and (8) is recovered.

Lemma 1. $\lim_{\beta_1 \to 1} w_{i,t} = 1/t$.

Note that the Hessian of the objective in (8) is $Q_{1:t}$. This becomes $\sum_{i=1}^t w_{i,t} Q_i$ in (9). Recall that $Q_{1:t}$ depends on the accumulated past gradients in (5), which is then refined by an exponential moving average in (6). As in Adam, we define $v_i = \beta_2 v_{i-1} + (1-\beta_2) g_i^2$, where $\beta_2 \in [0,1)$ and $v_0 = 0$, and then correct its bias by dividing by $1-\beta_2^t$. Thus, (5) is changed to

$$\sum_{i=1}^t w_{i,t} Q_i = \text{diag}\left(\frac{1}{\eta_t}\left(\sqrt{\frac{v_t}{1-\beta_2^t}} + \epsilon_t \mathbf{1}\right)\right), \tag{11}$$

where η_t and ϵ_t are the stepsize and ϵ value at time t, respectively. When $\beta_2 = 0$, (11) reduces to $\sum_{i=1}^t w_{i,t} Q_i = \text{diag}\left(\frac{1}{\eta_t}(\sqrt{g_t^2} + \epsilon_t \mathbf{1})\right)$. When $\beta_2 \to 1$, all g_i^2's are weighted equally and (11) reduces to $\sum_{i=1}^t w_{i,t} Q_i = \text{diag}\left(\frac{1}{\eta_t}\left(\sqrt{\frac{g_{1:t}^2}{t}} + \epsilon_t \mathbf{1}\right)\right)$. Using $\eta_t = \eta/\sqrt{t}$ and $\epsilon_t = \epsilon/\sqrt{t}$, this is further reduced to (5). The following shows that Q_t in (11) has a closed-form expression.

Proposition 1. *Define* $d_t = \frac{1-\beta_1^t}{\eta_t}\left(\sqrt{\frac{v_t}{1-\beta_2^t}} + \epsilon_t \mathbf{1}\right)$. *Then,*

$$Q_t = diag\left(\frac{d_t - \beta_1 d_{t-1}}{1-\beta_1}\right). \tag{12}$$

Algorithm 1 Follow the Moving Leader (FTML).

```
**Input:** $\eta_t > 0$, $\beta_1, \beta_2 \in [0, 1)$, $\epsilon_t > 0$.
**initialize** $\theta_0 \in \Theta$; $d_0 \leftarrow 0$; $v_0 \leftarrow 0$; $z_0 \leftarrow 0$;
**for** $t = 1, 2, \dots, T$ **do**
    fetch function $f_t$;
    $g_t \leftarrow \partial_\theta f_t(\theta_{t-1})$;
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$;
    $d_t \leftarrow \frac{1-\beta_1^t}{\eta_t}\left(\sqrt{\frac{v_t}{1-\beta_2^t}} + \epsilon_t \mathbf{1}\right)$;
    $\sigma_t \leftarrow d_t - \beta_1 d_{t-1}$;
    $z_t \leftarrow \beta_1 z_{t-1} + (1 - \beta_1) g_t - \sigma_t \theta_{t-1}$;
    $\theta_t \leftarrow \Pi_\Theta^{\text{diag}(d_t/(1-\beta_1^t))}(-z_t/d_t)$;
**end for**
**Output:** $\theta_T$.
```

Substituting this back into (9), θ_t is then equal to

$$\arg\min_{\theta\in\Theta} \sum_{i=1}^{t} w_{i,t}\left(\langle g_i, \theta\rangle + \frac{1}{2}\|\theta - \theta_{i-1}\|^2_{\text{diag}\left(\frac{\sigma_i}{1-\beta_1}\right)}\right), \quad (13)$$

where $\sigma_i \equiv d_i - \beta_1 d_{i-1}$. Note that some entries of σ_i may be negative, and $\|\theta - \theta_{i-1}\|^2_{\text{diag}(\sigma_i/(1-\beta_1))}$ is then not a regularizer in the usual sense. Instead, the negative entries of σ_i encourage the corresponding entries of θ to move away from those of θ_{i-1}. Nevertheless, from the definitions of d_t, σ_t and (11), we have $\sum_{i=1}^t w_{i,t}\text{diag}(\sigma_i/(1-\beta_1)) = \sum_{i=1}^t w_{i,t} Q_i = \text{diag}(d_t/(1-\beta_1^t))$, and thus the following:

Lemma 2. $\sum_{i=1}^t w_{i,t} \mathit{diag}(\sigma_i/(1-\beta_1)) \succ 0$.

Hence, the objective in (13) is still strongly convex. Moreover, the following Proposition shows that θ_t in (13) has a simple closed-form solution.

Proposition 2. *In (13),*

$$\theta_t = \Pi_\Theta^{\mathit{diag}(d_t/(1-\beta_1^t))}(-z_t/d_t),$$

where $z_t = \beta_1 z_{t-1} + (1-\beta_1) g_t - \sigma_t \theta_{t-1}$, and $\Pi_\Theta^A(x) \equiv \arg\min_{u\in\Theta} \frac{1}{2}\|u - x\|_A^2$ is the projection onto Θ for a given positive semidefinite matrix A.

The proposed procedure, which will be called *follow the moving leader* (FTML), is shown in Algorithm 1. Note that though $\{P_1, \dots, P_t\}$ are considered in each round, the update depends only the current gradient g_t and parameter θ_{t-1}. It can be easily seen that FTML is easy to implement, memory-efficient and has low per-iteration complexity.

3.2. Relationships with Adagrad, RMSprop and Adam

3.2.1. Relationship with Adagrad

The following Propositions show that we can recover Adagrad in two extreme cases: (i) $\beta_1 = 0$ with decreasing stepsize; and (ii) $\beta_1 \to 1$ with increasing stepsize.

Proposition 3. *With $\beta_1 = 0$, $\beta_2 \to 1$, $\eta_t = \eta/\sqrt{t}$, and $\epsilon_t = \epsilon/\sqrt{t}$, θ_t in (13) reduces to:*

$$\Pi_\Theta^{\mathit{diag}((\sqrt{g_{1:t}^2}+\epsilon\mathbf{1})/\eta)}\left(\theta_{t-1} - \mathit{diag}\left(\frac{\eta}{\sqrt{g_{1:t}^2} + \epsilon\mathbf{1}}\right) g_t\right),$$

which recovers Adagrad in (4).

Proposition 4. *With $\beta_1 \to 1$, $\beta_2 \to 1$, $\eta_t = \eta\sqrt{t}$, and $\epsilon_t = \epsilon/\sqrt{t}$, we recover (1) with Q_i in (3). If $\Theta = \mathbb{R}^d$, it generates identical updates as Adagrad in (4).*

3.2.2. Relationship with RMSprop

When $\Theta = \mathbb{R}^d$, McMahan (2011) showed that (1) and (2) generate the same updates. The following Theorem shows that FTML also has a similar gradient descent update.

Theorem 1. *With $\Theta = \mathbb{R}^d$, FTML generates the same updates as:*

$$\theta_t = \theta_{t-1} - \mathit{diag}\left(\frac{1-\beta_1}{1-\beta_1^t}\frac{\eta_t}{\sqrt{v_t/(1-\beta_2^t)} + \epsilon_t\mathbf{1}}\right) g_t. \quad (14)$$

When $\beta_1 = 0$ and bias correction for the variance is not used, (14) reduces to RMSprop in (7). However, recall from Section 3.1 that when $\beta_1 = 0$, we have $w_{i,t} = 0$ for $i < t$, and $w_{t,t} = 1$. Hence, only the current loss component P_t is taken into account, and this may be sensitive to the noise in P_t. Moreover, as demonstrated in Adam, bias correction of the variance can be very important. When $\beta_2 \to 1$, the variance estimate of RMSprop, $\sum_{i=1}^t (1-\beta_2)\beta_2^{t-i} g_i^2$, becomes zero and blows up the stepsize, leading to divergence. In contrast, FTML's Q_i in (12) recovers that of Adagrad in this case (Proposition 4). In practice, a smaller β_2 has to be used for RMSprop. However, a larger β_2 enables the algorithm to be more robust to the gradient noise in general.

3.2.3. Relationship with Adam

At iteration t, instead of centering regularization at each θ_{i-1} in (13), consider centering all the proximal regularization terms at the last iterate θ_{t-1}. θ_t then becomes:

$$\arg\min_{\theta\in\Theta} \sum_{i=1}^{t} w_{i,t}\left(\langle g_i, \theta\rangle + \frac{1}{2}\|\theta - \theta_{t-1}\|^2_{\text{diag}\left(\frac{\sigma_i}{1-\beta_1}\right)}\right). \quad (15)$$

Compared with (13), the regularization in (15) is more aggressive as it encourages θ_t to be close only to the last iterate θ_{t-1}. The following Proposition shows that (15) generates the same updates as Adam.

Proposition 5. *In (15),*

$$\theta_t = \Pi_\Theta^{A_t}\left(\theta_{t-1} - A_t^{-1}\sum_{i=1}^t w_{i,t} g_i\right), \quad (16)$$

where $A_t = \mathit{diag}((\sqrt{v_t/(1-\beta_2^t)} + \epsilon_t\mathbf{1})/\eta_t)$.

As in Adam, $\sum_{i=1}^{t} w_{i,t} g_i$ in (16) can be obtained as $m_t/(1-\beta_1^t)$, where m_t is computed as an exponential moving average of g_t's: $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$.

Note that the θ_t updates of Adagrad (4), RMSprop (7), and FTML (14) depend only on the current gradient g_t. On the other hand, the Adam update in (16) involves $\sum_{i=1}^{t} w_{i,t} g_i$, which contains all the past gradients (evaluated at past parameter estimates θ_{i-1}'s). This is similar to the use of momentum, which is sometimes helpful in escaping from local minimum. However, when the data distribution is changing (as in deep reinforcement learning), the past gradients may not be very informative, and can even impede parameter adaptation to the environment. Recently, it is also reported that the use of momentum can make training unstable when the loss is nonstationary (Arjovsky et al., 2017). Indeed, Theorem 4.1 in (Kingma & Ba, 2015) shows that Adam has low regret only when β_1 is decreasing w.r.t. t. When $\beta_1 \to 0$, $\sum_{i=1}^{t} w_{i,t} g_i \to g_t$ and so only the current gradient is used.

Remark 1. *(Summary) RMSprop and Adam are improvements over Adagrad in training deep networks. However, RMSprop uses $\beta_1 = 0$ (and thus relies only on the current sample), does not correct the bias of the variance estimate, but centers the regularization at the current iterates θ_{i-1}'s. On the other hand, Adam uses $\beta_1 > 0$, bias-corrected variance, but centers all regularization terms at the last iterate θ_{t-1}. The proposed FTML combines the nice properties of the two.*

4. Experiments

In this section, experiments are performed on a number of deep learning models, including convolutional neural networks (Section 4.1), deep residual networks (Section 4.2), memory networks (Section 4.3), neural conversational model (Section 4.4), deep Q-network (Section 4.5), and long short-term memory (LSTM) (Section 4.6). A summary of the empirical performance of the various deep learning optimizers is presented in Section 4.7.

The following state-of-the-art optimizers for deep learning models will be compared: (i) Adam (Kingma & Ba, 2015); (ii) RMSprop (Tieleman & Hinton, 2012); (iii) Adadelta (Zeiler, 2012); and (iv) Nesterov accelerated gradient (NAG) (Sutskever et al., 2013). For FTML, we set $\beta_1 = 0.6$, $\beta_2 = 0.999$, and a constant $\epsilon_t = \epsilon = 10^{-8}$ for all t. For FTML, Adam, RMSprop, and NAG, η is selected by monitoring performance on the training set (note that Adadelta does not need to set η). The learning rate is chosen from $\{0.5, 0.25, 0.1, \dots, 0.00005, 0.000025, 0.00001\}$. Significantly underperforming learning rates are removed after running the model for $5-20$ epochs. We then pick the rate that leads to the smallest final training loss.

4.1. Convolutional Neural Networks

In the section, we perform experiments with the convolutional neural network (CNN) (LeCun et al., 1998). We use the example models on the *MNIST* and *CIFAR-10* data sets from the Keras library[1]. For *MNIST*, the CNN has two alternating stages of 3×3 convolution filters (using ReLU activation), followed by a 2×2 max-pooling layer and a dropout layer (with a dropout rate of 0.25). Finally, there is a fully-connected layer with ReLU activation and a dropout rate of 0.5. For *CIFAR-10*, the CNN has four alternating stages of 3×3 convolution filters (using ReLU activation). Every two convolutional layers is followed by a 2×2 max-pooling layer and a dropout layer (with a dropout rate of 0.25). The last stage has a fully-connected layer with ReLU activation and a dropout rate of 0.5. Features in both data sets are normalized to $[0, 1]$. Minibatches of sizes 128 and 32 are used for *MNIST* and *CIFAR-10*, respectively.

As the iteration complexities of the various algorithms are comparable and the total cost is dominated by backpropagation, we report convergence of the training cross entropy loss versus the number of epochs. This setup is also used in (Zeiler, 2012; Kingma & Ba, 2015).

Figure 1 shows the convergence results. As can be seen, FTML performs best on both data sets. Adam has comparable performance with FTML on *MNIST*, but does not perform as well on *CIFAR-10*. The other methods are much inferior. In particular, RMSprop is slow on both *MNIST* and *CIFAR-10*, and Adadelta tends to diverge on *CIFAR-10*.

4.2. Deep Residual Networks

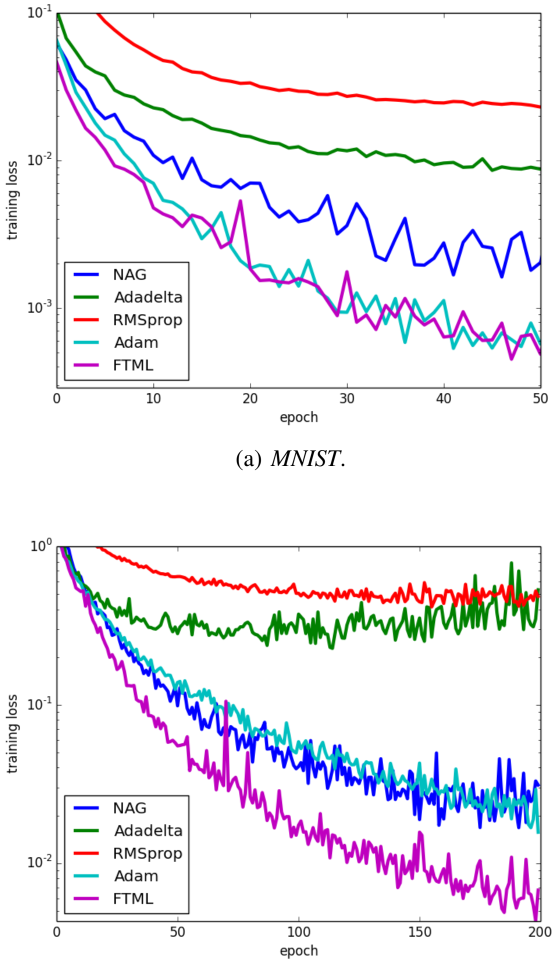
Recently, substantially deeper networks have been popularly used, particularly in computer vision. For example, a 152-layer deep residual network (He et al., 2016) achieves state-of-the-art performance on ImageNet classification, and won the first place on the ILSVRC 2015 classification task.

In this section, we perform experiments with a 110-layer deep residual network on the *CIFAR-10* and *CIFAR-100* data sets. The code is based on its Torch implementation[2]. We leave the architecture and related settings intact, and use the same learning rate schedule. The default optimizer in the Torch code is NAG. Here, we also experiment with Adadelta, RMSprop, Adam and the proposed FTML. A minibatch size of 32 is used.

Convergence of the training cross entropy loss is shown in Figure 2. As can been seen, all optimizers, except Adadelta, are very competitive and have comparable per-

[1] https://github.com/fchollet/keras.

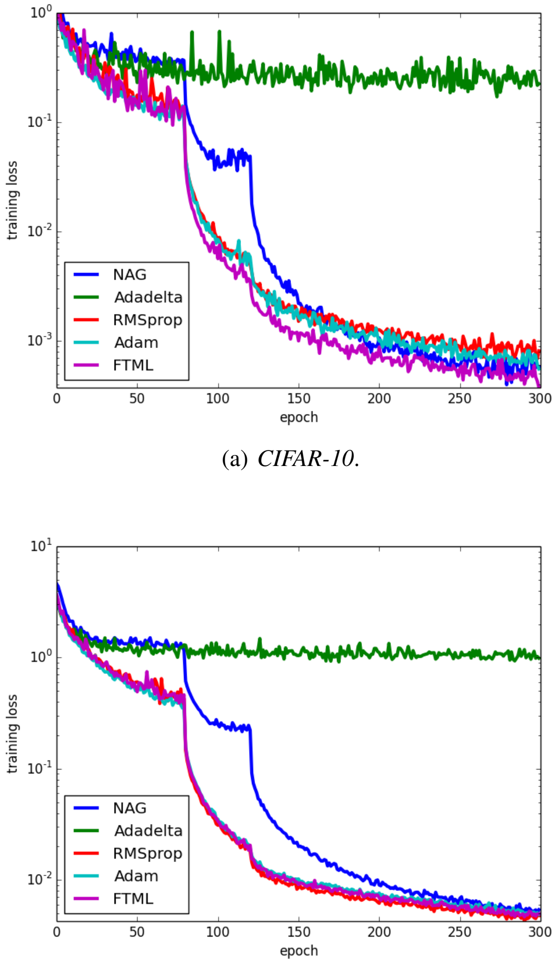
[2] https://github.com/facebook/fb.resnet.torch.



(a) *MNIST*.

(b) *CIFAR-10*.

Figure 1. Results on convolutional neural network.



(a) *CIFAR-10*.

(b) *CIFAR-100*.

Figure 2. Results on deep residual network.

formance on these two data sets. NAG shows slower initial convergence, while FTML converges slightly faster than the others on the *CIFAR-10* data set.

4.3. Memory Networks

Recently, there has been a lot of attention on combining inference, attention and memory for various machine learning tasks. In particular, the memory network (Weston et al., 2014; Sukhbaatar et al., 2015) has been popularly used for natural language understanding.

In this section, we use the example model of the end-to-end memory network (with LSTM) from the Keras library. We consider the question answering task (Sukhbaatar et al., 2015; Weston et al., 2016), and perform experiments on the "single supporting fact" task in the *bAbI* data set (Weston et al., 2016). This task consists of questions in which a previously given single sentence provides the answer. An example is shown below. We use a single supporting memory, and a minibatch size of 32.

Single Supporting Fact:
Mary moved to the bathroom.
John went to the hallway.
Where is Mary? A: bathroom

Convergence of the training cross entropy loss is shown in Figure 3. As can be seen, FTML and RMSprop perform best on this data set. Adam is slower, while NAG and Adadelta perform poorly.

4.4. Neural Conversational Model

The neural conversational model (Vinyals & Le, 2015) is a sequence-to-sequence model (Sutskever et al., 2014) that is capable of predicting the next sequence given the last or previous sequences in a conversation. A LSTM layer en-

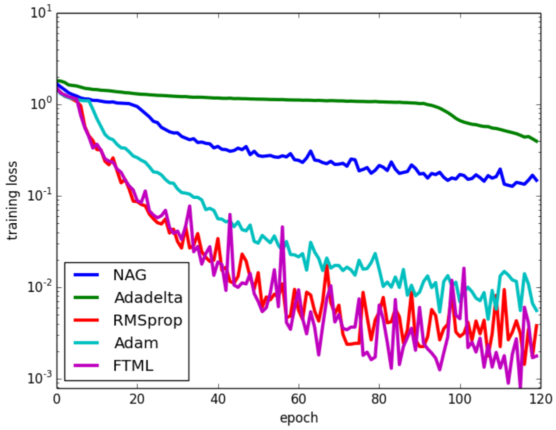


Figure 3. Results on memory network.

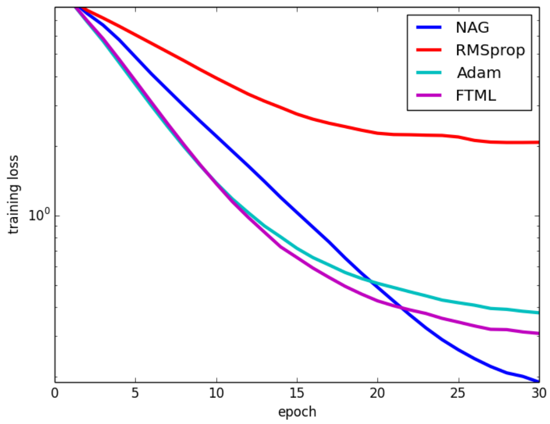


Figure 4. Results on neural conversational model.

codes the input sentence to a thought vector, and a second LSTM layer decodes the thought vector to the response. It has been shown that this model can often produce fluent and natural conversations.

In this experiment, we use the publicly available Torch implementation[3] with a constant stepsize, and its default data set *Cornell Movie-Dialogs Corpus* (with $50,000$ samples) (Danescu-Niculescu-Mizil & Lee, 2011). The number of hidden units is set to 1000, and the minibatch size is 10.

Convergence of the training cross entropy loss is shown in Figure 4. Adadelta is not reported here, since it performs poorly (as in previous experiments). As can be seen, FTML outperforms Adam and RMSprop. In particular, RMSprop is much inferior. NAG is slower than FTML and Adam in the first 21 epochs, but becomes faster towards the end of training.

4.5. Deep Q-Network

In this section, we use the Deep Q-network (DQN) (Mnih et al., 2015) for deep reinforcement learning. Experiments are performed on two computer games on the Atari 2600 platform: *Breakout* and *Asterix*. We use the publicly available Torch implementation with the default network setup[4], and a minibatch size of 32. We only compare FTML with RMSprop and Adam for optimization, as NAG and Adadelta are rarely used in training the DQN. As in (Mnih et al., 2015), we use $\epsilon = 10^{-2}$ for all methods, and performance evaluation is based on the average score per episode. The higher the score, the better the performance.

Convergence is shown in Figure 5. On *Breakout*, RMSprop and FTML are comparable and yield higher scores than Adam. On *Asterix*, FTML outperforms all the others. In particular, the DQN trained with RMSprop fails to learn the task, and its score begins to drop after about 100 epochs. A similar problem has also been observed in (Hasselt et al., 2016). Experience replay (Mnih et al., 2015) has been commonly used in deep reinforcement learning to smooth over changes in the data distribution, and avoid oscillations or divergence of the parameters. However, results here show that Adam still has inferior performance because of its use of all past gradients, many of these are not informative when the data distribution has changed.

4.6. Long Short-Term Memory (LSTM)

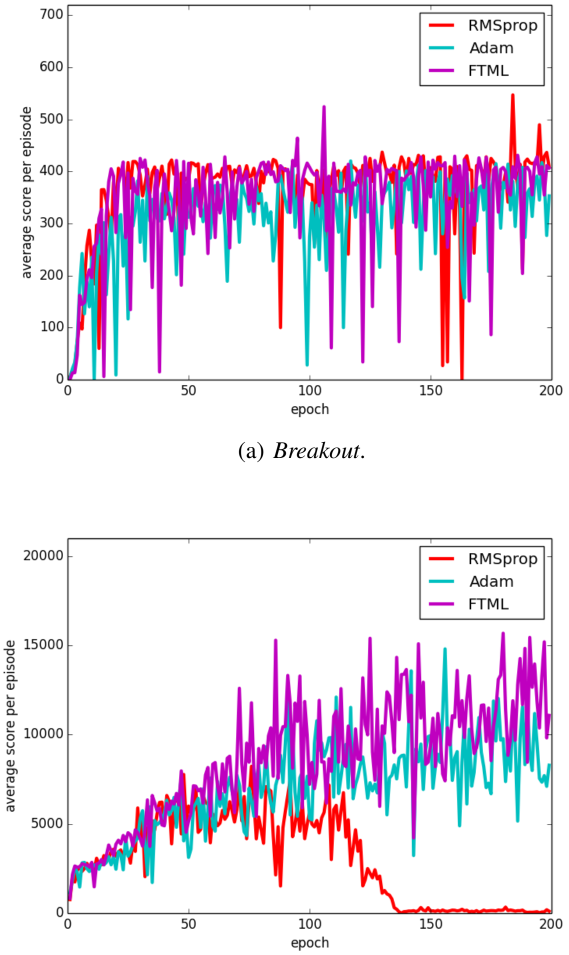
To illustrate the problem of Adam in Section 4.5 more clearly, we perform the following timeseries prediction experiment with the LSTM. We construct a synthetic timeseries of length 1000. This is divided into 20 segments, each of length 50. At each time point, the sample is 10-dimensional. In segment i, samples are generated from a normal distribution with mean $([i, i, \dots, i] + \zeta_i) \in \mathbb{R}^{10}$ and identity covariance matrix, where the components of ζ_i are independent standard normal random variables. Noise from the standard normal distribution is added to corrupt the data. The task is to predict the data sample at the next time point t.

We use a one-layer LSTM implemented in (Léonard et al., 2015). 100 hidden units are used. We truncate backpropagation through time (BPTT) to 5 timesteps, and input 5 samples to the LSTM in each iteration. Thus, the data distribution changes every 10 iterations, as a different normal distribution is then used for data generation. Performance evaluation is based on the squared loss $f_t(\theta_{t-1})$ at time t.

Convergence of the loss is shown in Figure 6(a). As can be

[3] `https://github.com/macournoyer/neuralconvo`.

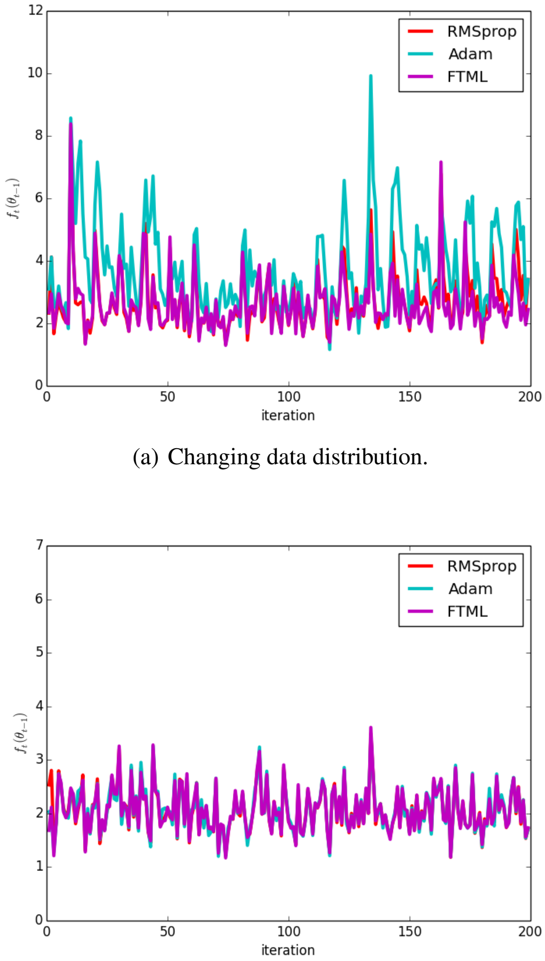
[4] `https://github.com/Kaixhin/Atari`.



(a) *Breakout*.

(b) *Asterix*.

Figure 5. Results on deep Q-network.



(a) Changing data distribution.

(b) Data distribution is stationary.

Figure 6. Results on timeseries data using LSTM.

seen, Adam has difficulty in adapting to the data. In contrast, FTML and RMSprop can adapt more quickly, yielding better and more stable performance.

As a baseline, we consider the case where the data distribution does not change (the means of all the segments are fixed to the vector of ones) Figure 6(b) shows the results. As can be seen, Adam now performs comparably to FTML and RMSprop.

4.7. Summary of Results

The main problem with RMSprop is that its performance is not stable. Sometimes, it performs well, but sometimes it can have significantly inferior performance (e.g., as can be seen from Figures 1, 4 and 5(b)). The performance of Adam is more stable, though it often lags behind the best optimizer (e.g., Figures 1(b), 3, and 4). It is particularly problematic when learning in a changing environment (Figures 5 and 6(a)). In contrast, the proposed FTML shows stable performance on various models and tasks. It converges quickly, and is always the best (or at least among the best) in all our experiments.

5. Conclusion

In this paper, we proposed a FTPRL variant called FTML, in which the recent samples are weighted more heavily in each iteration. Hence, it is able to adapt more quickly when the parameter moves to another local basin, or when the data distribution changes. FTML is closely related to RMSprop and Adam. In particular, it enjoys their nice properties, but avoids their pitfalls. Experimental results on a number of deep learning models and tasks demonstrate that FTML converges quickly, and is always the best (or among the best) of the various optimizers.

Acknowledgments

This research was supported in part by ITF/391/15FX.

References

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein GAN. Preprint arXiv:1701.07875, 2017.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference for Learning Representations*, 2015.

Bottou, L. Online learning and stochastic approximations. In *On-line Learning in Neural Networks*, pp. 9–142. Cambridge University Press, 1998.

Cesa-Bianchi, N. and Lugosi, G. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

Danescu-Niculescu-Mizil, C. and Lee, L. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, pp. 76–87, 2011.

Dauphin, Y., de Vries, H., and Bengio, Y. Equilibrated adaptive learning rates for non-convex optimization. In *Advances in Neural Information Processing Systems*, pp. 1504–1512, 2015.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pp. 2933–2941, 2014.

Dean, J., Corrado, G.S., Monga, R., Chen, K., Devin, M., Le, Q.V., and Ng, A. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

Duchi, J. and Singer, Y. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10(12):2899–2934, 2009.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 (7):2121–2159, 2011.

Graves, A., Mohamed, A., and Hinton, G. Speech recognition with deep recurrent neural networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, 2013.

Hasselt, H. V., Guez, A., and Silver, D. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2094–2100, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hinton, G. E., Osindero, S., and Teh, Y. A fast learning algorithm for deep belief nets. *Neural Computation*, 18 (7):1527–1554, 2006.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kalai, A. and Vempala, S. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

Kawaguchi, K. Deep learning without poor local minima. In *Advances In Neural Information Processing Systems*, pp. 586–594, 2016.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the International Conference for Learning Representations*, 2015.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Léonard, N., Waghmare, S., and Wang, Y. RNN: Recurrent library for Torch. Preprint arXiv:1511.07889, 2015.

Martens, J. Deep learning via Hessian-free optimization. In *Proceedings of the International Conference on Machine Learning*, pp. 735–742, 2010.

McMahan, H. Follow-the-regularized-leader and mirror descent: Equivalence theorems and L1 regularization. In *International Conference on Artificial Intelligence and Statistics*, pp. 525–533, 2011.

McMahan, H. B. and Streeter, M. Adaptive bound optimization for online convex optimization. In *Proceedings of the Annual Conference on Computational Learning Theory*, pp. 244, 2010.

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., Chikkerur, S., Liu, D., Wattenberg, M., Hrafnkelsson, A. M., Boulos, T., and Kubica, J. Ad click prediction: A view from the trenches. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pp. 1222–1230, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

Schraudolph, N. N. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

Shalev-Shwartz, S. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2012.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van D. D., George, Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pp. 2440–2448, 2015.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning*, pp. 1139–1147, 2013.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, COURSERA: Neural networks for machine learning, 2012.

Vinyals, O. and Le, Q. A neural conversational model. Preprint arXiv:1506.05869, 2015.

Weston, J., Chopra, S., and Bordes, A. Memory networks. Preprint arXiv:1410.3916, 2014.

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., and Mikolov, T. Towards AI-complete question answering: A set of prerequisite toy tasks. In *Proceedings of the International Conference for Learning Representations*, 2016.

Xiao, L. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11(10):2543–2596, 2010.

Zeiler, M. D. ADADELTA: An adaptive learning rate method. Preprint arXiv:1212.5701, 2012.

Zinkevich, M. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the International Conference on Machine Learning*, pp. 928–936, 2003.