Follow the Moving Leader in Deep Learning (FTML)

Team 8

TA - Sharfuddin Mohammed

Team Members - Yash Verma (201501103)
Harshit Patni (201501107)
Aayush Surana (201531012)
Lakshya Agrawal (201530102)

Introduction

- Deep learning has emerged as a powerful and a popular class of machine learning problems. They are highly non-linear and difficult to optimize. A few examples include Convolutional Neural Network, Long Short Term Memory, Memory Network, etc.
- Training deep networks is difficult as the optimization can suffer from pathological curvature and get stuck in local minima. Here, comes the difficulties of getting stuck in saddle points, which are defined as those critical points which are not the global minima, and also different weights may have different gradients in terms of magnitude and direction.
- To come out of this, second-order information is useful as it reflects the local curvature of the error surface. But, here again we face the problem of computing the Hessian matrix for the function. At a point, to check whether it is a saddle point or not, if the Hessian matrix at that point is Indefinite, meaning it is neither positive-definite, negative-definite, positive semi-definite, negative semi-definite and having both positive and negative Eigen values, is a saddle point.
- To avoid the Hessian computation, many other methods were proposed like Hessian-free optimization, using the absolute Hessian to escape from saddle points, but still they required high computational costs.

Well known highly used Stochastic Gradient Descent algorithms -

Averaging

Averaged stochastic gradient descent, is ordinary stochastic gradient descent that records an average of its parameter vector over time. That is, the update is the same as for ordinary stochastic gradient descent, but the algorithm also keeps track of

$$\bar{w} = \frac{1}{t}\sum_{i=0}^{t-1} w_i$$

When optimization is done, this averaged parameter vector takes the place of w.

AdaGrad

AdaGrad (for adaptive gradient algorithm) is a modified stochastic gradient descent with per-parameter learning rate. Informally, this increases the learning rate for more sparse parameters and decreases the learning rate for less sparse ones. This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative. It still has a base learning rate η, but this is multiplied with the elements of a vector $\{G_{j,j}\}$ which is the diagonal of the outer product matrix.

$$G = \sum_{\tau=1}^{t} g_\tau g_\tau^{\mathsf{T}}$$

where $g_\tau = \nabla Q_i(w)$, the gradient, at iteration τ. The diagonal is given by $G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2$

This vector is updated after every iteration. The formula for an update is now $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$

Adagrad is especially efficient on high-dimensional data.

RMSProp

RMSProp (for Root Mean Square Propagation) is also a method in which the learning rate is adapted for each of the parameters. The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight. So, first the running average is calculated in terms of means square,

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$

where, γ is the forgetting factor.

And the parameters are updated as,

$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

Adam

Adam (short for Adaptive Moment Estimation) is an update to the RMSProp optimizer. In this optimization algorithm, running averages of both the gradients and the second moments of the gradients are used. Given parameters $w^{(t)}$ and a loss function $L^{(t)}$,where t indexes the current training iteration (indexed at 1), Adam's parameter update is given by:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)} \qquad \hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^t}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2 \qquad \hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2^t}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

- 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.
- However, Adagrad suffers from diminishing stepsize. As optimization proceeds, the accumulated squared gradient $g_{1:t}^2$ becomes larger and larger.
- To alleviate this problem, 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 decreasing.
- For example, RMSprop uses

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

where $\boldsymbol{\beta}$ is close to 1 and the update becomes

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

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

This was proposed to ensure that the parameter and update have the same unit.

Follow the leader (FTL)

- The simplest learning rule to try is to select (at the current step) the hypothesis that has the least loss over all past rounds. This algorithm is called Follow the leader, and is simply given by, at round t,

$$w_t = \arg\min_{w \in S} \sum_{i=1}^{t-1} v_i(w)$$

- This method can thus be looked as a greedy algorithm. For the case of online quadratic optimization (where the loss function is $v_t(w) = ||w - x_t||_2^2$, one can show a regret bound that grows as $\log(T)$.
- However, similar bounds cannot be obtained for the FTL algorithm for other important families of models like online linear optimization. To do so, one modifies FTL by adding a regularisation term.
- Here, the difficulty arises as the parameter ω may shift drastically from round to round, and FTL can fail.

Follow the Regularized Leader (FTRL)

- The main problem with Follow the Leader is that the parameter can shift dramatically from round to round and FTL can fail to produce low regret for convex problems.
- Thus instead we may consider adding a fixed regularizer to our function as regularizer can help us to stabilize the solution so the low regret can be guaranteed.
- 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

- Intuitively FTRL algorithm tries to balance two things: minimize loss on current vector and ensure that θ_t+1 is close to θ_t
- Issues with FTRL:
  - Uses a fixed regularizer for all t
  - Different feature dimensions may carry different amounts of information
- To take the above information into account another method was proposed which is called Follow the Proximally Regularized Leader.

Follow the Proximally Regularized Leader (FTPRL)

- This method is centering regularization at each iteration :

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

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

- Q_i : full/diagonal positive semidefinite matrix which can incorporate second order information.
- $\|\theta - \theta_{i-1}\|_{Q_i}$ Is the corresponding Mahalanobis distance between θ and θ_{i-1} .
- This method includes Adagrad as special case, this means that FTPRL is equivalent to gradient when the convex constraint set is real space.
- Theoretically for convex problems by setting summation of Q equal to the diagonal matrix we can obtain nearly the optimal regret bound for any non increasing step size. In general all these algorithms satisfy -

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

Follow the Moving Leader (FTML)

FTML is a variant of the FTRL algorithm and is closely related to RMSprop and Adam. It enjoys their nice properties, but avoids their pitfalls. In FTML, the 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.

As in FTRL, we used the next iterate as

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

where $P_i(\theta) = \langle g_i, \theta \rangle + \frac{1}{2}\|\theta - \theta_{i-1}\|_{Q_i}^2$

Here, as we can see all the P_i's have the same weight. 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.

To alleviate this problem, one may consider only P_i's in a recent window. However, a large memory is needed for its implementation. So, here in FTML, we use

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

where the weights $w_{i,t} = \frac{(1-\beta_1)\beta_1^{t-i}}{1-\beta_1^t}$

The weights are normalized to sum to 1. Here, the denominator $1-\beta_1^t$ plays a similar role as bias correction in Adam.

So, the next iterate $\theta_{\square}$ is generated as

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

where $\sigma_i \equiv d_i - \beta_1 d_{i-1}$.

FTML is easy to implement, memory-efficient and has low per-iteration complexity.

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

Hessian of the objective as in FTRL, $\theta_t = \arg\min_{\theta \in \Theta} \sum_{i=1}^{t} P_i(\theta)$, is $Q_{1:t}$

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

$Q_{1:t}$ Depends on the accumulated past gradients in $Q_{1:t} = \text{diag}\left(\frac{1}{\eta} \left(\sqrt{g^2_{1:t}} + \epsilon \mathbf{1} \right) \right)$

We define $v_i = \beta_2 v_{i-1} + (1 - \beta_2) g_i^2$,

where $\beta_2 \in [0, 1)$ and $v_o = 0$ and then correct its bias by dividing by $1 - \beta_2^t$.

Thus, $Q_{1:t} = \text{diag}\left(\frac{1}{\eta} \left(\sqrt{g^2_{1:t}} + \epsilon \mathbf{1} \right) \right)$

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

$\sum_{i=1}^{t} w_{i,t} Q_i$

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

Substituting this back into $\theta_t = \arg\min_{\theta\in\Theta} \sum^{t} P_i(\theta)$,

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

$$\sum_{i=1}^{t} w_{i,t}\text{diag}(\sigma_i/(1-\beta_1)) = \sum_{i=1}^{t} w_{i,t} Q_i = \text{diag}(d_t/(1-\beta_1^t))$$

Proposition 2 : $\theta_t = \Pi_{\Theta}^{diag(d_t/(1-\beta_1^t))}(-z_t/d_t)$

where $z_t = \beta_1 z_{t-1} + (1-\beta_1)g_t - \sigma_t \theta_{t-1}$

$$\Pi_{\Theta}^{A}(x) \equiv \arg\min_{u \in \Theta} \frac{1}{2}\|u-x\|_A^2$$

This is the projection onto **Θ** for a given positive semidefinite matrix A.

Relationship with Adagrad :

Proposition 3 : With $\beta_1 = 0$, β_2 tends to 1, $\eta_t = \eta/\sqrt{t}$, and $\epsilon_t = \epsilon/\sqrt{t}$, θ_t

$$\arg\min_{\theta \in \Theta} \sum_{i=1}^{t} w_{i,t} \left(\langle g_i, \theta \rangle + \frac{1}{2} \|\theta - \theta_{i-1}\|^2_{\text{diag}\left(\frac{\sigma_i}{1-\beta_1}\right)} \right)$$ reduces to $$\Pi_{\Theta}^{diag((\sqrt{g^2_{1:t}}+\epsilon \mathbf{1})/\eta)} \left(\theta_{t-1} - diag\left(\frac{\eta}{\sqrt{g^2_{1:t}} + \epsilon \mathbf{1}} \right) g_t \right)$$

which recovers Adagrad.

Relationship with RMSprop : With $\Theta = \mathbb{R}^d$, FTML generates the same updates as

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

When $\beta_1 = 0$, and bias correction for the variance is not used, the above equation reduces to RMSprop.

Relationship with Adam : At iteration t, instead of considering regularization at each θ_{i-1} in

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

consider centering all the proximal regularization terms at the last iterate θ_{t-1}.

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

Here, the regularization is more aggressive as θ_t is close only to the last iterate θ_{t-1}.

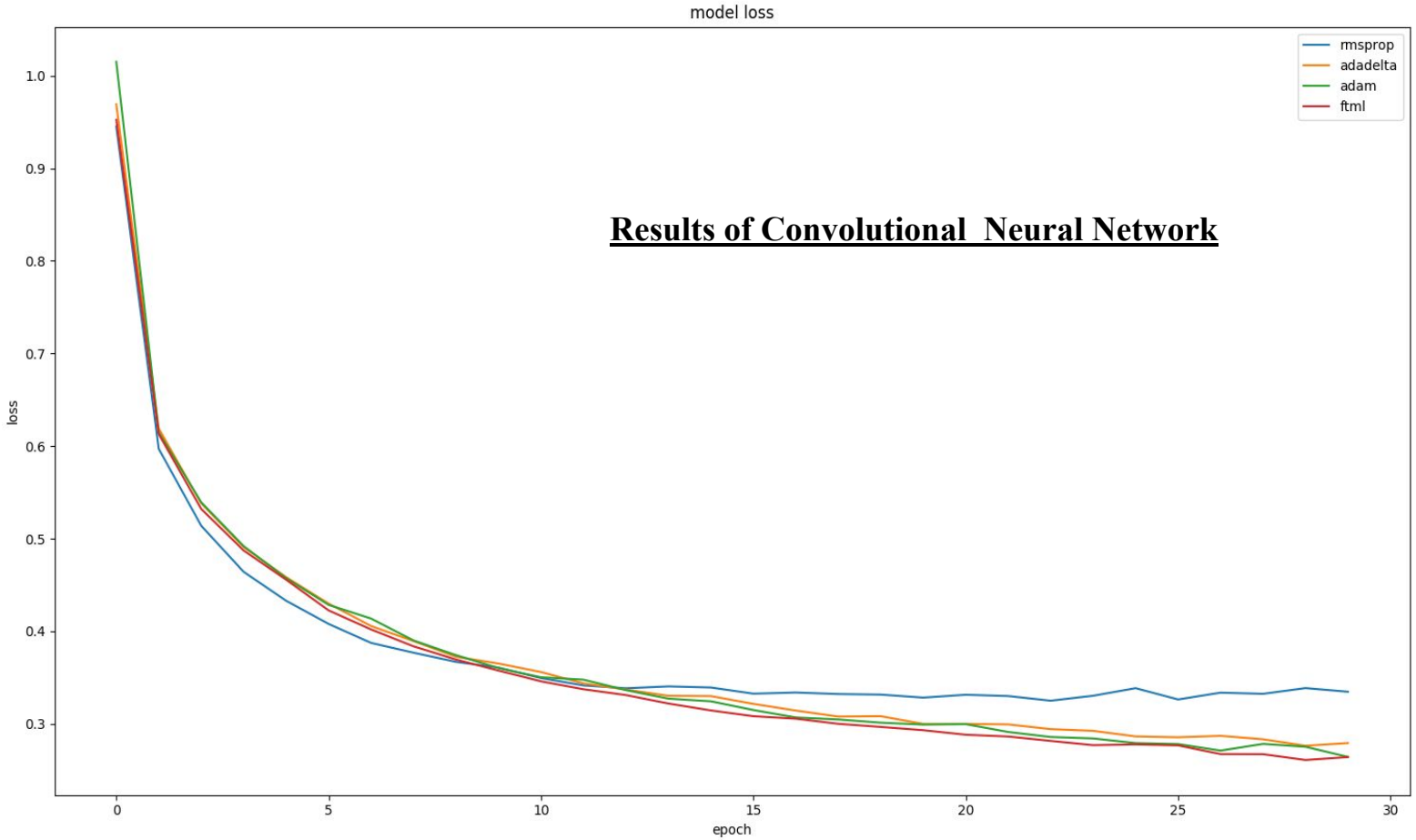
Proposition 5 : $$\theta_t=\Pi_{\Theta}^{A_t}\left(\theta_{t-1}-A_t^{-1}\sum_{i=1}^{t} w_{i,t}g_i\right)$$

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

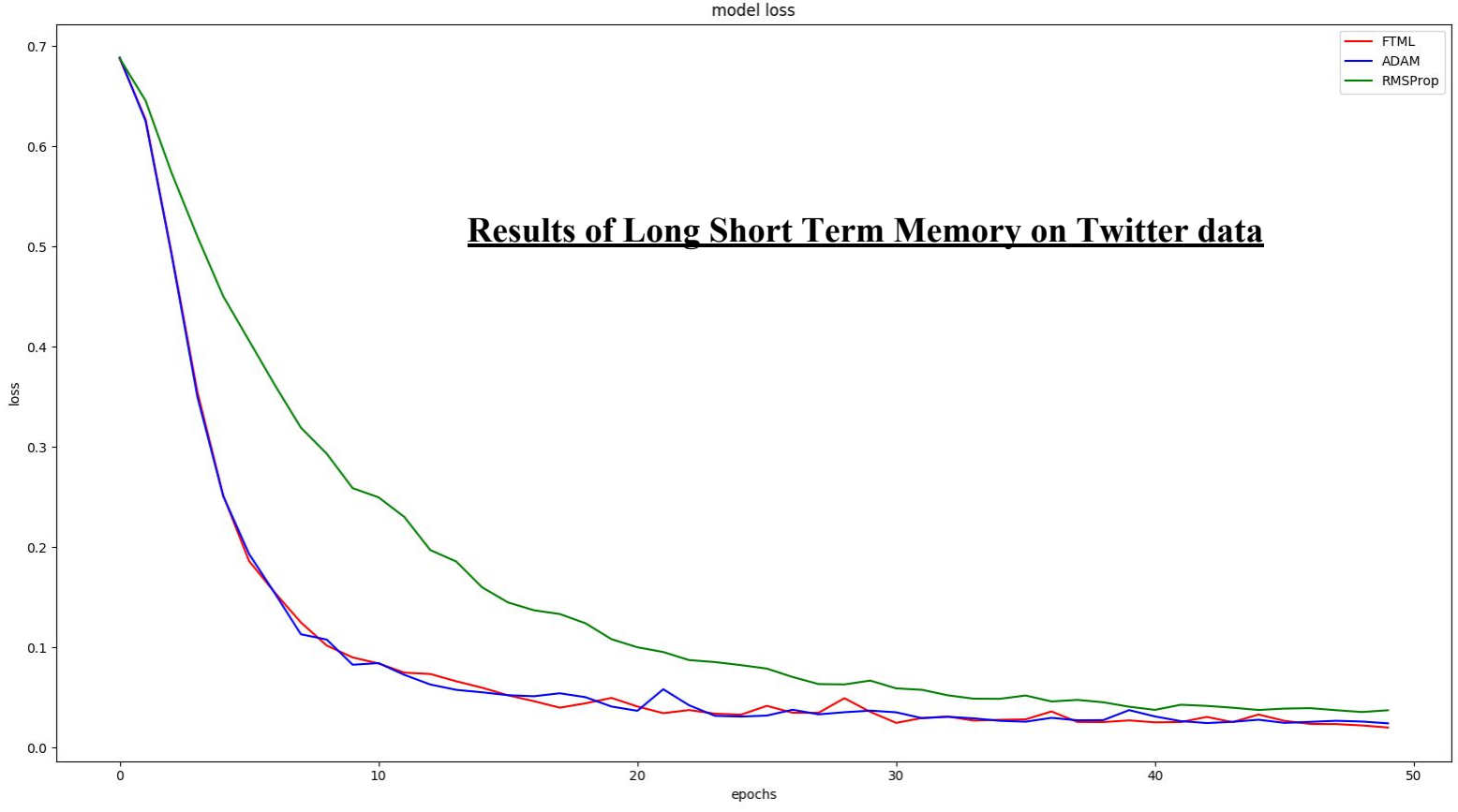
As in Adam, $\sum_{i=1}^{t} w_{i,t}g_i$ above in the equation can be obtained as $m_t/(1-\beta_1^t)$, where m_t is computed as an exponential moving average of g_t's: $m_t=\beta_1 m_{t-1}+(1-\beta_1)g_t$

FTML - Algorithm

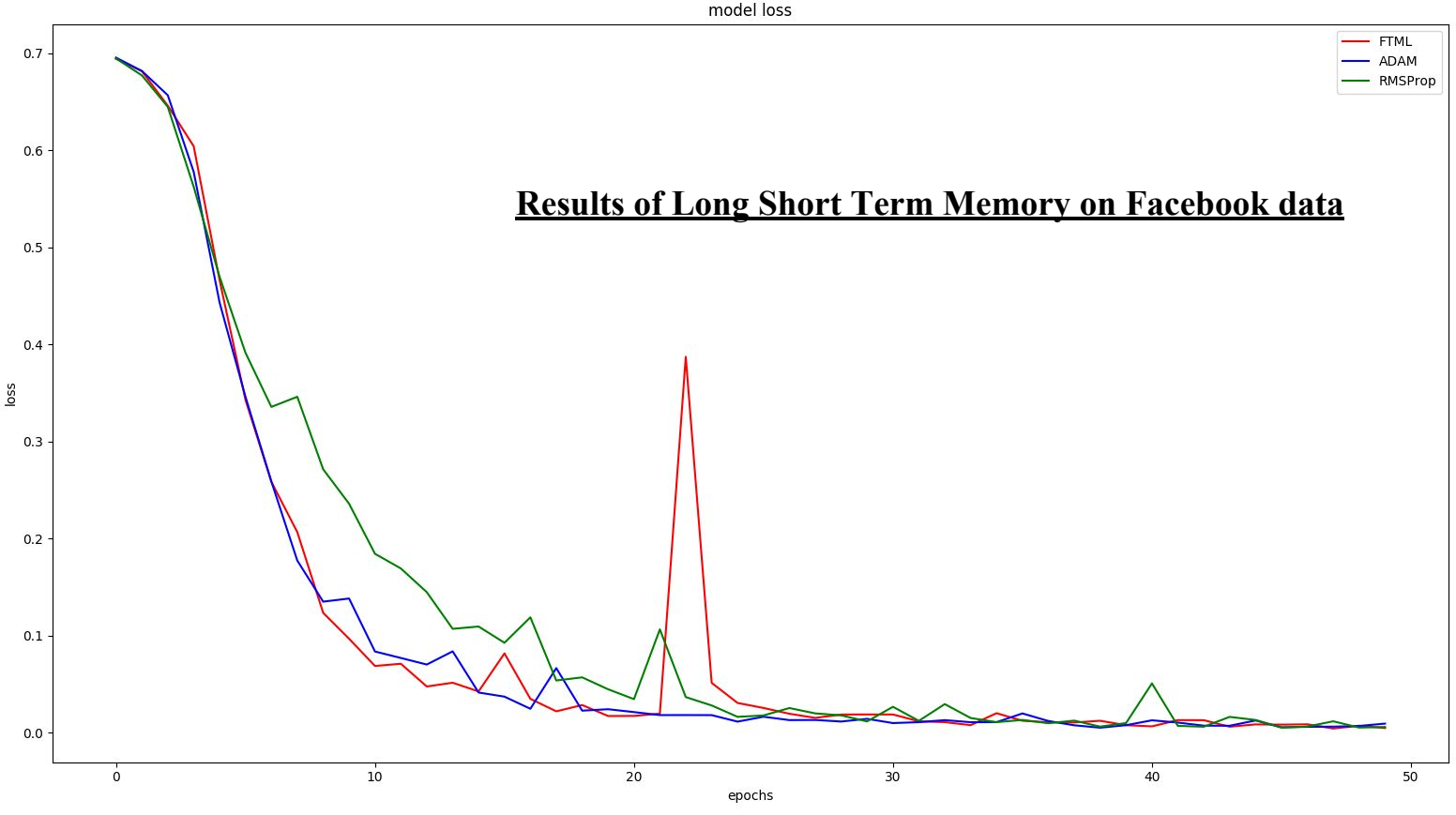
Input: $\eta_t > 0, \beta_1, \beta_2 \in [0, 1), \epsilon_t > 0.$
initialize $\theta_0 \in \Theta; d_0 \leftarrow 0; v_0 \leftarrow 0; z_0 \leftarrow 0;$
for $t = 1, 2, \ldots, T$ **do**
  fetch function f_t;
  $g_t \leftarrow \partial_\theta f_t(\theta_{t-1});$
  $v_t \leftarrow \beta_2 v_{t-1} + (1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



Results of Convolutional Neural Network



Results of Long Short Term Memory on Twitter data



Conclusions

- Therefore, here we proposed a variant of the FTPRL algorithm called the FTML algorithm.
- Motivated by connections between stochastic optimization and online learning
  - ➢ Deep network is highly non-convex
  - ➢ Uniform weight→exponential moving average
  - ➢ Low time and space complexities as SGD
- Here in FTML algorithm, the recent samples are weighted more heavily in each iteration. Hence, it is able to adapt more quickly when the parameter distribution moves to another local basin, or the data distribution changes.
- Excellent empirical performance on different deep learning models. It outperforms or is at least comparable with the state-of-the-art optimizers.
- FTML is closely related to RMSprop and Adam, 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.

Work Done Till Phase One Evaluation

- Literature review of the all the associated algorithms like:
  - ➢ Follow the Leader
  - ➢ Follow the Regularized Leader
  - ➢ Follow the Proximally Regularized Leader
- Complete reading of the paper “Follow the Moving Leader in Deep Learning”
- Started coding - Follow the Moving Leader Algorithm

Work Done Till Now

- Completing the bug free implementation of Follow the Moving Leader Algorithm
- Comparing the performance of algorithm on different types of neural networks and other algorithms
- Understanding the math behind the paper with all the proofs