L-BFGS in Reinforcement Learning

Ikeda, Shotaro
ikeda2@illinois.edu

Mishra, Akshay
akmishr2@illinois.edu

Agarwal, Shubhankar
mystery@illinois.edu

December 17, 2017

1 Abstract

In current deep learning literature, first-order methods are much more common due to it's computation speed. However, there is a similarity between Adam and L-BFGS, with computations of different version of scaling. With this project, we show that L-BFGS is not an in-place replacement for Adam and requires a through investigation for the policy to converge.

2 Introduction

In class, we learned the power of second order methods and speed of convergence using those methods. In current deep learning literature, first-order methods are much more common due to it's computation speed. The current state of the art first order method, Adam computes a version of scaling using the first and second moment of the gradients, successfully becoming state-of-the-art in terms of convergence. However, we observe the similarity between Adam and L-BFGS, since L-BFGS computes a different type of scaling by satisfying the secant conditions and using an approximation of the Hessian to accelerate convergence. Knowing these two facts, we conduct an in-depth study using both optimizers on deep reinforcement learning tasks.

2.1 Reinforcement Learning

For our reinforcement learning algorithm, we use Q-Learning with temporal difference update. We also use a target network, which helps stabilize learning. For our test environment, we used OpenAI's CartPole, which is a task to balance a pole on a cart as long as possible.

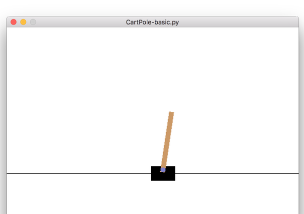

Figure 1: CartPole Environment

2.2 Optimizers

We compare Adam and L-BFGS. Here, we introduce what guided us to state the similarity between Adam and L-BFGS.

2.2.1 Adam Optimizer

The Adam update uses an estimate of the first and second moments of the gradients, to have either a dampening or multiplicative effect on the gradient update.

Adam uses the following hyper-parameters:

- β_1, a mixing coefficient determining the momentum of the first moment estimate
- β_2, a mixing coefficient determining the momentum of the second moment estimate
- ε, a factor to make sure division by zero does not occur.
- η, the learning rate

These four hyper-parameters create the following rule for the tth update. Suppose θ is the current set of parameters, that we wish to train. Then the update rule for θ is

$$\begin{aligned}
\theta &\leftarrow \theta_{t-1} \\
g_t &\leftarrow \nabla_\theta f_\theta(x) \\
m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\
v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \\
\hat{m}_t &\leftarrow \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &\leftarrow \frac{v_t}{1-\beta_2^t} \\
\theta_t &\leftarrow \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{v_t} + \varepsilon}
\end{aligned}$$

Notice here, that the update rule is not explicitly based off of the gradient calculation g_t by itself. The Adam calculates estimates for the first and second moments, using those to obtain a some notion of scaling the gradient.

2.2.2 L-BFGS Optimizer

L-BFGS update on the other hand, uses a rough estimate of the Hessian with some constraints.

Let α be the step size. Typically line search is used, but pytorch's LBFGS operates on batches and uses fixed-width instead of a line search. In particular, the update for BFGS is as of follows

$$
\begin{aligned}
\theta &\leftarrow \theta_t \\
p_k &\leftarrow -B_k^{-1}\nabla_\theta \\
s_k &\leftarrow \alpha \cdot p_k \\
x_{k+1} &\leftarrow x_k + s_k \\
y_k &\leftarrow \nabla_\theta(x_{k+1}) - \nabla_\theta(x_k) \\
B_{k+1} &\leftarrow B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}
\end{aligned}
$$

Notice that the step is chosen with an approximation of the inverse Hessian, B_k. Since the approximation of the Hessian is a guaranteed decent direction, due to the secant conditions, it can also be thought of as another version of scaling the gradient. L-BFGS is another approximation on top of this, without storing the Hessian, and keeping track of the last h steps.

3 Experiments

For each experiment, we run the operation 5 times, and show 1 standard deviation away from each run.

3.1 Replacing Adam

We attempt to replace Adam by finding a set of hyper-parameters that induces stable learning for Adam, and seeing how well replacing by L-BFGS does.

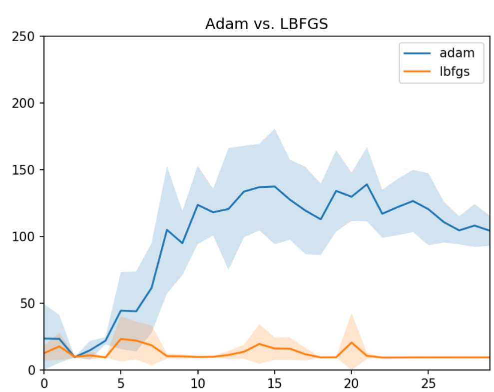

Figure 2: Since the parameters are tuned for Adam, it is not surprising L-BFGS does not perform worse. However, it performs much worse.

To confirm the odd behavior with L-BFGS, we fine tune the parameters by performing a small grid search across modifying α, the weight for prioritizing loss in the replay buffer across batches, η for the learning rate, and h for the history size.

We found out that using LBFGS tended to explode the loss after a certain point, after which the performance of the model would not improve no matter what happened. It was not uncommon to

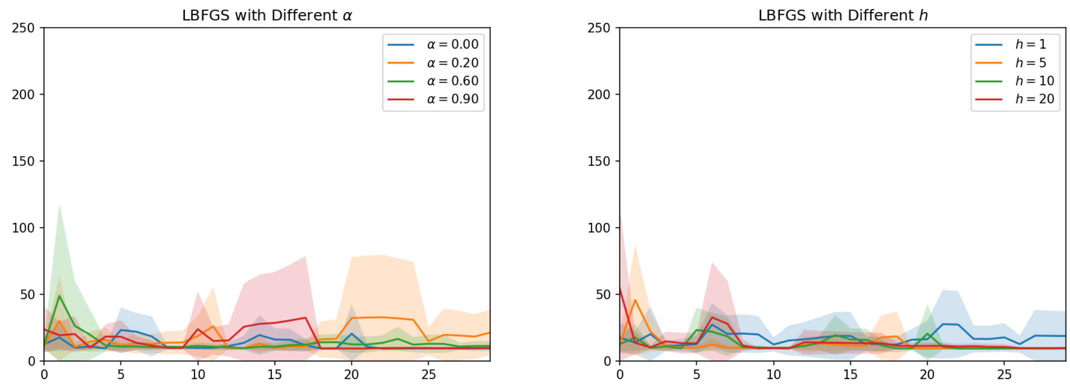

Figure 3: Various hyperparameter searches, α on left, h on right

see a loss of NaN during training. In addition, the greater the history size, the deep net trended towards faster explosion of the loss.