Langevin Dynamics-Based Neural Network Optimization on Real-World Data

Unique Divine
Columbia University in the City of New York
{u.divine}@columbia.edu

Abstract

In this semester project, I study the effectiveness of Stochastic Gradient Markov Chain Monte Carlo (SG-MCMC) methods for the optimization of deep neural networks. As there are already strong theoretical justifications for why Bayesian variants of Stochastic Gradient Descent (SGD) can aid in non-convex optimization, this work focuses on empirically comparing standard optimization schemes to Bayesian variants such as Stochastic Gradient Langevin Dynamics (SGLD) and precondition SGLD (pSGLD).

I provide a from-scratch implementation of both SGLD and pSGLD in the GitHub repository that includes the demo. Benchmarks are done with matching learning rates for each optimizer, hyperparameters cited from the papers in which they were proposed, and varied neural network architectures that have between 70,000 and 400,000 learnable parameters. Overall, it is found that the optimization schemes with an adaptive learning rate are the most robust. However, SG-MCMC based methods are shown to make small improvements over vanilla SGD to train more shallow networks that have fewer parameters. All code for this project is available at https://github.com/Unique-Divine/SA-Project.

1 Introduction

Overfitting is problematic for effective training of neural networks. As is evident from techniques like dropout (Srivastava et al., 2014), the injection of extra stochasticity during model learning can act as a form of regularization, helping prevent models from getting stuck in local optima. Bayesian approaches offer a similar benefit by quantifying a measure, or estimate, of uncertainty.

1.1 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods are a class of statistical sampling algorithms that have had much success in the scientific community for solving high-dimensional problems through the use simulations (Andrieu et al., 2003). These techniques can be applied to optimization problems in high dimensional vector spaces, making them a natural candidate for deep artificial neural network training. MCMC methods are inherently Bayesian and therefore facilitate capturing uncertainty in learned parameters (Welling and Teh, 2011). Many claim that his helps avoid overfitting.

However, one downside of many MCMC algorithms is that each iteration may require computations over an entire dataset. With the recent successes of deep learning and big data, large datasets of high-dimensional raw inputs are becoming increasingly common, and this requires greater computational resources, often causing most MCMC-based methods to become computationally intractable.

1.2 Bayesian variants of Stochastic Gradient Descent

Because MCMC algorithms were not designed to process massive sample sizes, (Welling and Teh, 2011) proposed a framework for learning from large scale datasets called Stochastic Gradient Langevin Dynamics (SGLD). This consists of adding Gaussian noise to vanilla SGD optimization, iterating through samples, or batches, of the data with a Bayesian posterior distribution. In the GitHub repository, I implemented this algorithm as a close-to-the-hardware PyTorch optimizer in `optimization.py`.

Langevin dynamics are a class of MCMC techniques that take gradient steps while adding in Gaussian noise to parameter updates (Neal et al., 2011). This differs from the traditional solution in maximum a posterioti (MAP) constraint optimiza-

tion:

$$\Delta\theta_t = \frac{\epsilon}{2}\left(\nabla \ln P(\theta_t) + \sum_i \nabla \ln P(x_i|\theta_t\right) + \eta_t \tag{1}$$

$$\eta_t \sim \mathcal{N}(0, \epsilon)$$

- θ: Parameter vector
- $P(\theta)$: Prior distribution
- $\{x_i\} \in X$: Data. Note that $P(\theta|X) \propto P(\theta)\prod_i P(x_i|\theta)$ by Bayes' Theorem.

Langevin dynamics incorporate a random walk that can be computationally inefficient (Neal et al., 2011). By blending the two paradigms, SGD and Langevin dynamics, a more powerful technique that allows efficient computations over massive datasets was born. SGLD is essentially the update from Eq.1 with a decreasing step size and stochastic gradient given by $G_{\text{true}}(\theta) + G_t(\theta)$, where

$$G_{\text{true}}(\theta) = \nabla \ln P(\theta) + \sum_i \nabla \ln P(x_i|\theta), \text{ and}$$

$$G_t(\theta) = \nabla \ln P(\theta) + \frac{N}{n}\sum_i \nabla \ln P(x_{t,i}|\theta) - G_{\text{true}}(\theta).$$

The authors of SGLD, (Welling and Teh, 2011), found the method to be effective, but they tested mostly on simulated data and found inconsistent hyperparameters.

Preconditioned SGLD (pSGLD): Although the theoretical arguments for the effectiveness of SGLD are strong, the algorithm because the mixing time can become cumbersome and computationally expensive. Preconditioned Stochastic Gradient Langevin Dynamics were proposed in an attempt to make up for the shortcomings of standard SGLD (Li et al., 2016). In this paper, empirical results were provided with both optimizers for Logistic Regression, Feed-Forward Neural Networks, and Convolutional Neural Nets, with pSGLD providing state-of-the-art performance. pSGLD is based on the RMSprop algorithm and seemed to allow neural networks to overcome overfitting problems resulting from pathological curvature.

2 Methods

In this work, I again compare Langevin-Dynamics based optimization algorithms except now with compatison to both vanilla SGD and Adam

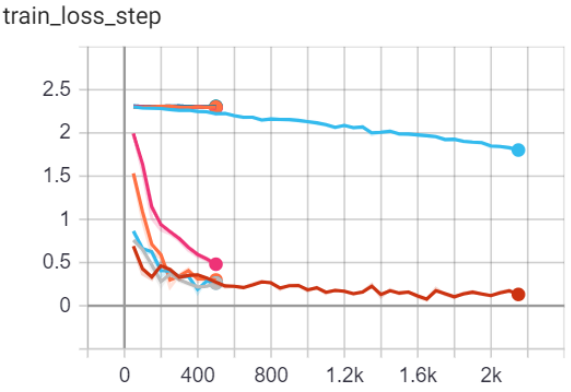

Figure 1: This the normalized per-batch training losses for 20 different neural networks as described in the Methods section. Each of the four algorithms managed to learn when given additional parameters. The few lines at the top of plot show SGD, SGLD, and pSGLD trained networks with 0 hidden blocks. All optimization methods worked with large numbers of hidden layers (over 100,000 learnable) parameters.

(Kingma and Ba, 2014), an algorithm for first-order gradient based optimization with an adaptive learning rate.

Several regularized feed-forward neural network architectures of varying depth are trained on the MNIST dataset for classification of handwritten digits. The training set size consists of 60,000 images and the test set contains 10,000 images. The feed-forward network architecture consisted of stacked linear blocks. Each block consisted of a standard Linear layer with ReLU activation and dropout (Srivastava et al., 2014) of 15 percent:

```
import torch.nn as nn
def RegularizedLinear(in_dim, out_dim):
    return nn.Sequential(
        nn.Flatten(start_dim=1),
        nn.Linear(in_features=in_dim,
                  out_features=out_dim),
        nn.ReLU(),
        nn.Dropout(p=0.15))
```

For ease of training with GPUs and early stopping, I make use of PyTorch Lightning (Falcon, 2019), a framework for scaling models in AI research.

Using consistent architectures across each algorithm is a natural choice here because we're more interested in comparing the optimization method rather than the predictive power of the neural network architectures. Early stopping was also used since we're not interested in performance as much as with figuring out when certain optimizers would converge. All of the code for the

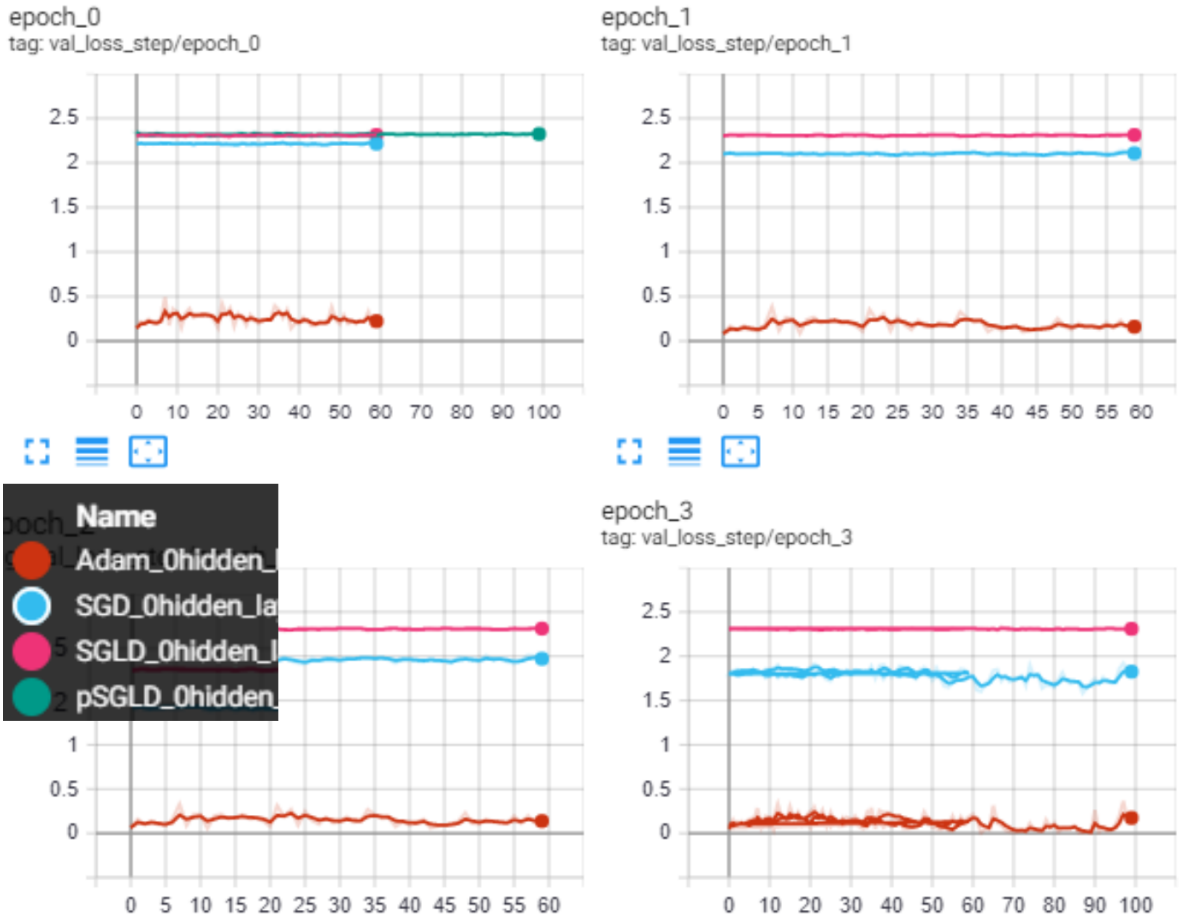

Figure 2: Regularized feed-forward neural nets trained until early stopping with no hidden blocks. Both SGLD and pSGLD improve with time but seem to be stuck in local optima. SGD continually improves and Adam manages to master the dataset quickly while continuing to improve.

neural networks are in `lit_modules.py`, while the stochastic gradient descent algorithms are in `optimization.py`.

The SGD, SGLD, pSGLD, and Adam optimizers are trained and tested until and meeting an early stopping criterion on neural networks composed of 0, 1, 2, 4, and 8 hidden blocks. The input and output layers are the same each time. The number of nodes in the hidden blocks is a function of the number of classes n_{cls} being predicted and the dimension of the inputs d_{in}:

$$n_{\text{hidden}} = \sqrt{d_{\text{in}} + n_{\text{cls}}}.$$

3 Results & Discussion

The SGD optimizer does improve with additional training. In Fig. 1, plot it's clear that most of the population of neural networks stops before batch 70 of this first epoch. This is because the main focus of this report was to compare the relative advantages in the speed of and flexibility of optimization granted by the respective gradient update algorithms.

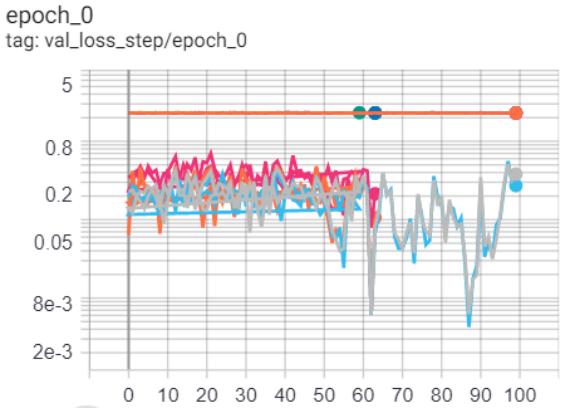

Figure 3: The best optimizer by far was Adam. Each model used a consistent starting learning rate of $1 \cdot 10^{-3}$, which may be the cause of this advantage. Adam 2h (2 hidden layers, sky blue) and Adam 1h (grey) managed to reach the best accuracy on MNIST. The group of losses on the bottom left are for pSGLD 8h, SGLD 8h, the other Adam optimizers, and SGD 1h-8h.

A common practice for this to use early stopping, where metrics are tracked at each step of the training process and the neural network training is halted as soon as metrics start to worsen for a specified number of iterations. I left a lot of leeway and let the networks train until they went 10 batches in a row without improvements.

Please note that the graphs here are for a specific run. The results were fairly inconsistent for all of the non-adaptive optimizers, however it's obvious from my experiment logs that SGLD is outperformed by both vanilla SGD and pSGLD. This may be due to the fact that vanilla SGD has undergone rigorous empirical study and, as a result, may have default parameters in PyTorch that are well-tuned to various problems. On the other hand, SGLD and pSGLD performed best with largely different hyperparameters in each task according to the simulated data experiments in the pSGLD paper (Li et al., 2016).

4 Conclusions

Backpropagation algorithms that adaptively update optimization hyperparameters seem to pose a clear advantage when approaching a new dataset. This work seems to support the findings in 2016 and 2018 (Li et al., 2016) (Brosse et al., 2018). Further study needs to be done with hyperparamter optimization to see if Langevin-based stochastic optimizers can produce more competitive results. The hyperparameters from the originally proposed papers were use for these algorithms, but these optimal parameters were either inconsitent between datasets or they produced on simulated datasets, so more work needs to be done in hyperparameter optimization here in order to objectively compare different stochastic optimizers.

References

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. 2003. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43.

Nicolas Brosse, Alain Durmus, and Eric Moulines. 2018. The promises and pitfalls of stochastic gradient langevin dynamics. *arXiv preprint arXiv:1811.10072*.

WA Falcon. 2019. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Chunyuan Li, Changyou Chen, David Carlson, and Lawrence Carin. 2016. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Radford M Neal et al. 2011. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Max Welling and Yee W Teh. 2011. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer.