Final Project Report

Alex Enze Liu

TTIC31230 - Fundamentals of Deep Learning

February 28 2017

Contents

1. Project Proposal

1.1. Project Topic

This project is trying to compare standard SGD to RMSProp and Adam under rigorous hyperparameter tuning based on project 4. Perplexity and Average Loss are mainly used for evaluating the performance of SGD, RMSProp and Adam.

There are a number of hyperparameters in the above algorithms, such as learning rate, decay rate, batch size, depth, etc. To simplify the experiment, I only consider a few among them. The hyperparameters taken into consideration are listed in table 1.

	SGD	RMSProp	Adam
Hyperparameters	Learning Rate	Learning Rate Decay Rate	Learning Rate First Moment Decay Rate Second Moment Decay Rate

Table 1: Hyperparameters To Tune

1.2. Project Overview

The data set will be divided into 3 subsets: train set, test set and validation set (with respectable size). In this project, the learning rate is fixed during training.

The experiment will mainly use random search, which is proposed by [1].

First, as is suggested in [2], the experiment will apply random search based on log scale on learning rate and decay rate. For example, the experiment will sample the hyperparameters on a very large scale first: $\eta = 10^{uniform(-8,3)}, \beta = 1 - 10^{uniform(-4,-1)}$. The search space of decay rate initially is small, as [3] and [4] already gives a few values, such as 0.9, 0.99 and 0.999. Next, based on the results, a few (η, β) pairs with less loss will be selected for further experimentation. The search space will then be narrowed based on selected pairs. Finally, selecting the learning rate repeatedly until the search space is narrowed down to a very small interval. [2] also recommends that the search could also be staged. That is, while the search interval is being narrowed, more epochs are performed during training to get more accurate results.

Other possible approaches reported by [5] for tuning hyperparamters, such as Sequential Model-based Global Optimization and Tree-structured Parzen Estimator, might also be used for tuning hyperparameters.

1.3. Possible Results and Explanations

There are many possible results, and some of them are very hard to explain.

- Normally we expect Adam to converges first, then RMSProp, finally SGD. The reason of this can relate to why RMSProp and SGD are proposed. RMSProp fixes the problem in SGD when the optimal point is on the shallow direction (i.e. the gradient is small on that direction). RMSProp rescales the gradient and accelerates the progress. Adam in some sense combines momentum and RMSProp, and compares favorably to Adam or SGD [6].

- One possible result is that Adam converges first, then SGD, then RMSProp. As is mentioned above, RMSProp rescales the gradient. It is possible that rescaling actually slows down the progress of converging. By comparison, Adam updates are estimated using a running average of first and second moment of the gradient [3]. [3] also points out another reason that might lead to slow convergence in RMSProp; RMSProp lacks a bias-correction term, which could result in very large stepsizes and divergence.

- It is also possible that Adam converges first, then RMSProp, and SGD does not converge. This happens when there is a saddle point. [6] shows that SGD actually stick in the saddle point while Adam and RMSProp will finally get out of the saddle point.

- Another possible result is that RMSProp performs better than Adam. This is reported by [7], when ϵ in Adam is not carefully selected. Since ϵ is not part of the experiment, a bad choice of ϵ could lead to this result.

- [8] shows another possible situation, where RMSProp and SGD perform slightly better than Adam in the beginning, but in the end Adam has the best performance. The reason is unknown. It is probably because of the special dataset.

- Finally, when doing the previous experimentation on momentum and SGD, I found that sometimes momentum and SGD produced almost same results in terms of average loss. So I think it is possible that SGD, Adam and RMSProp have same results, on condition that the hyperparameters, such as learning rate, are not well-selected.

2. Project Design

2.1. Framework of the Project

The framework I am using is edf. Most of the code is the same as edf. But I did modify the code a bit base on my needs. I changed the LSTM function in proj4.ipynb from SGD to Adam and RMSProp. The modification is shown in Code 1.

```
#Some Code

#This function is almost the same as provided LSTM
def LSTM_Adam(eta, beta1, beta2, nunmber_of_epoch=10):
    #Some Code
    for ep in range(nunmber_of_epoch):

        perm = np.random.permutation(len(minbatches)).tolist()
        stime = time()

        for k in range(len(minbatches)):

            minbatch = minbatches[perm[k]]
            x_padded = utils.make_mask(minbatch)
            inp.set(x_padded)
            loss, score = BuildModel()
            edf.Forward()
            edf.Backward(loss)
            edf.GradClip(10)

            ### The only change is here, replace SGD by Adam.
            edf.Adam(eta,beta1,beta2)
            #edf.SGD(eta)
            #RMSProp is the same

        duration = (time() - stime)/60.
    #Some Code
```

Code 1: Modification in LSTM Function

Also, during the experiment, I found that epsilon in edf.py is so small that overflow happened quite often when learning rate was large (e.g. greater than 1). To expand the search scope, I changed epsilon in Adam and RMSProp from 1e-8 to 1e-1. The modification is shown in Code 2.

```

def RMSProp(lr, g=0.9, ep=1e-1):
    #Modified, epsilon here is set to large value to avoid overflow
    .
    #Adam is the same.

    #soma code
    for p in params:
        p.grad_hist = g*p.grad_hist + (1-g)*p.grad*p.grad
        p.grad = p.grad/np.sqrt(p.grad_hist + DT(ep))
    SGD(lr)
```

Code 2: Modification in RMSProp function

2.2. Searching for Best Hyperparameters

The searching process consists of 2 stages. For stage 1, we search for a large space using random search to get enough samples (in this project we sampled 60 points). For stage 2, we search near the best-performing points with more epochs to get more accurate results.

In stage 1, the search space of learning rate, decay rate in RMSProp, beta1 and beta2 in Adam is set to 10^{-4} - 10, 0.84 - 0.99, 0.84 - 0.999, 0.84 - 0.9999 respectively. The epoch is set to 1 due to the limited amount of time. In my project stage 1 is actually divided into 3 rounds to get enough samples from different intervals (e.g. 0.01-1, 1-10). A snippet of code controlling searching is shown in code 3.

```
#Round 1
pair_numbers = 36 #Number of Sample points
number_of_epoch = 1 #Epoch = 1
np.random.seed(0) #For repeatability
for i in range(pair_numbers):
    eta = 10 ** np.random.uniform(-4,1)
    #Eta: 0.0001 - 10
    g_RMSProp = 1 - 10 ** np.random.uniform(-0.8,-2)
    #Decay_Rate: 0.84 - 0.99
    beta1_Adam = 1 - 10 ** np.random.uniform(-0.8,-3)
    #Beta1: 0.84 - 0.999
    beta2_Adam = 1 - 10 ** np.random.uniform(-0.8,-4)
    #Beta2: 0.84 - 0.9999

    NormalSGD(eta, number_of_epoch)
    MyRMSProp(eta, g_RMSProp, number_of_epoch)
    MyAdam(eta, beta1_Adam, beta2_Adam, number_of_epoch)
```

Code 3: Stage 1 Searching

After getting enough samples in stage 1, we select 3 samples with smallest loss each from SGD, RMSProp and Adam. We then perform the random search near the selected points. Stage 2 is also divided into 3 rounds by narrowing the search space each round. A snippet of code is shown in code 4.

```
#Round3
pair_numbers = 16
number_of_epoch = 2
#Set Epoch to 2
np.random.seed(5)

#Load Best 3 to SGD array, SGD[0] = savedEta
#Load Best 3 to RMSProp array, RMSProp[0] = savedEta, RMPSProp[1]=
    savedDecayRate
#Load Best 3 to Adam array, Adam[0] = savedEt a, Adam[1] =
    savedBeta1, Adam[2] = savedBeta2

for j in range(pair_numbers):
    for i in range(3):
        # Top 3 For Each Method
        #SGD
        eta = SGD[i][0] * np.random.uniform(0.99,1.01)
        #Search around eta*0.99 - eta*1.01
        NormalSGD(eta, number_of_epoch)

        #RMSProp
        eta = RMSProp[i][0] * np.random.uniform(0.99,1.01)
        #Search around eta*0.99 - eta*1.01
        g_RMSProp = 1 - 10 ** np.random.uniform(-0.8,-2)
        #0.84 - 0.99
        MyRMSProp(eta, g_RMSProp, number_of_epoch)

        #Adam
        eta = Adam[i][0] * np.random.uniform(0.99,1.01)
        #Search around eta*0.99 - eta*1.01
        beta1_Adam = 1 - 10 ** np.random.uniform(-0.8,-3)
        #0.84 - 0.999
        beta2_Adam = 1 - 10 ** np.random.uniform(-0.8,-4)
        #0.84 - 0.9999
        MyAdam(eta, beta1_Adam, beta2_Adam, number_of_epoch)
```

Code 4: Stage 2 Searching

3. Results and Analysis

3.1. Stage 1

3.1.1. Learning Rate

The results for stage 1 with η ranging from 0.0001 to 10 are displayed in figure 1.

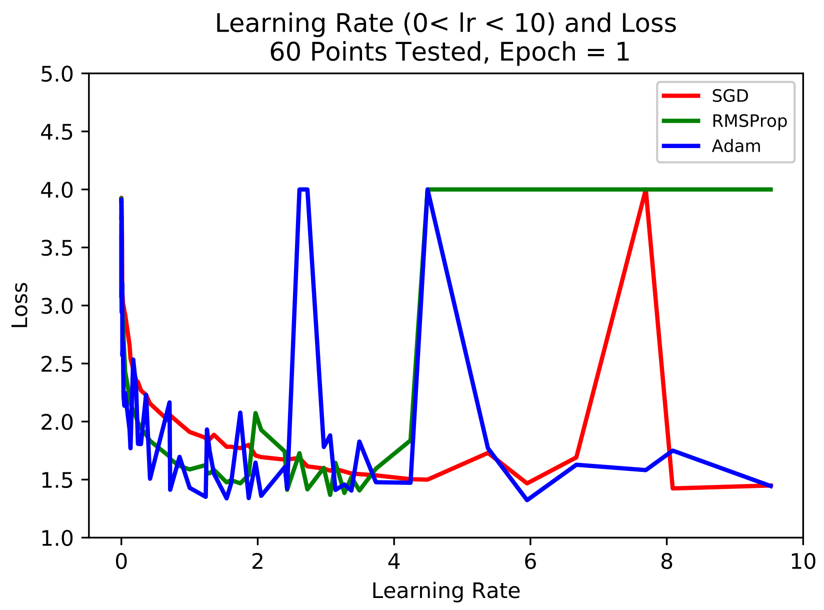

Figure 1: Loss with different learning rate

When you first look at figure 1, you might feel perplexed. However, if we divide this figure into 2 figures, there are some interesting results. In this project I break the results into 2 parts. Figure 2 shows loss with $0 < \eta < 1$; Figure 3 shows the loss with $1 < \eta < 10$.

As can be seen in figure 2 and 3, within the range of $0 < \eta < 1$, all the 30 samples show that RMSProp is guaranteed to have a performance better than SGD, while Adam is not. One possible explanation for the fluctuation in Adam's performance is that, besides eta, Beta1, Beta2 and Epsilon in Adam could have some influence on its performance. Furthermore, we can guess that a decay rate ranging from 0.8 - 0.99 does not have a significant influence of RMSProp's performance.

In figure 3, We can find all the situations predicted previously, and what happens in figure 3 is a bit hard to explain. It seems like that there is no rule to predict the performance between SGD, RMSProp and Adam. But we do have several findings. We notice that RMSProp stops working when learning rate is greater than 4. From this we can have the intuition that learning rate over 4 might be too high. So I did some extra experiments on several samples with $\eta > 4$: I ran 5 more epochs for these samples. The loss results are displayed in table 2. We can see that loss increases rather than decreases within 5 epochs, this further proves our intuition. Combining our previous hypothesis that η is the main factor that influences RMSProp, we infer that what happens to RMSProp in figure 3 might because of the relatively high learning rate.

	Loss		
Epochs	SGD ($\eta = 8.0867$)	SGD ($\eta = 9.5237$)	Adam ($\eta = 5.9508$)
Epoch 0	1.4239	1.4493	1.3228
Epoch 1	1.3831	1.3460	1.3728
Epoch 2	1.2228	1.2331	1.2664
Epoch 3	1.5269	1.2109	1.3191
Epoch 4	1.1834	1.1808	1.2051
Epoch 5	1.1811	1.2069	1.2363

Table 2: Loss for Several Samples with $\eta > 4$

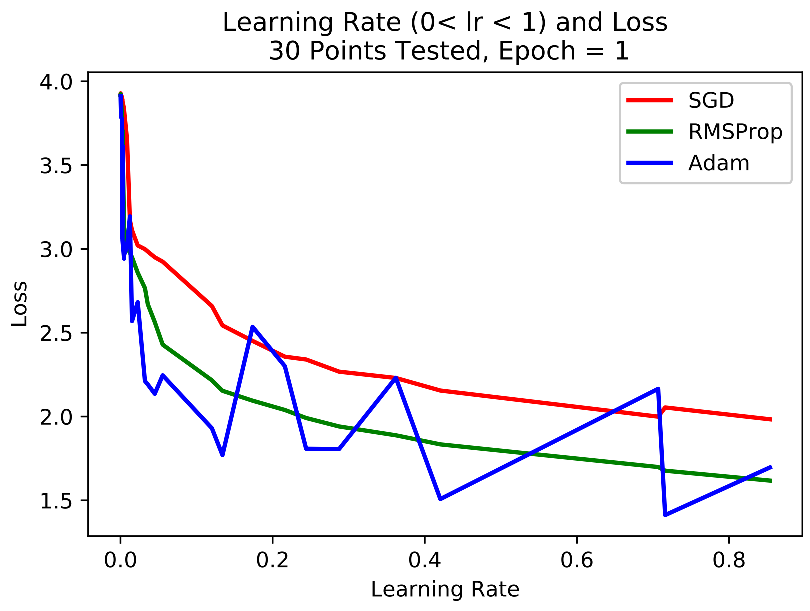

Figure 2: Loss with different learning rate

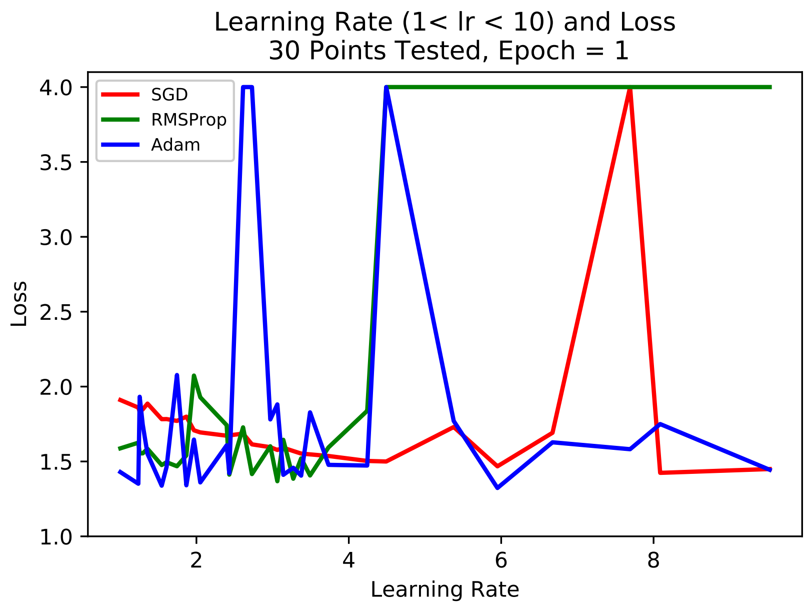

Figure 3: Loss with different learning rate

3.1.2. Other Hyperparameters

In the above section, we notice that besides η, other hyperparameters, like beta1 and beta2, might have some influence on Adam's performance, while decay rate in RMSProp might only affect the performance slightly. So in this section we are trying to use the data we gathered to verify if our observation is right. Figure 4,5,6 show the magnitude of loss with respect to decay rate, beta1, and beta2.

From figure 4, we cannot find any obvious connection between decay rate and loss. From figure 5 and 6, we do see some intervals where loss is relatively small, like 0.87-0.93 (loss under 1.5) in figure 5 and 0.89-0.96 (loss under 1.7) in figure 6. These intervals, to some extend, could indicate that some beta1 and beta2 might be better. However, due to a bug in my code, beta1 and beta2 are not generated uniformly, and the samples in the above intervals are not enough to lead to the conclusion that some beta1 and beta2 are definitely better.

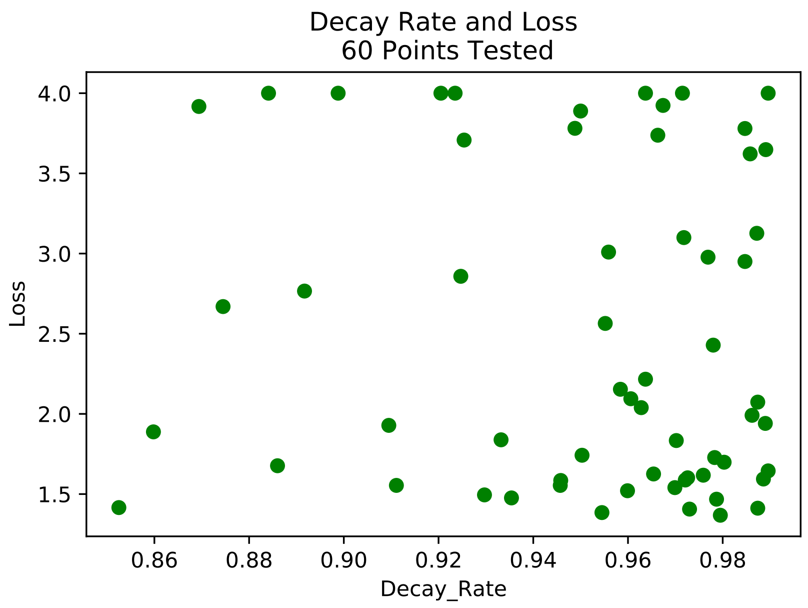

Figure 4: Loss with different learning rate

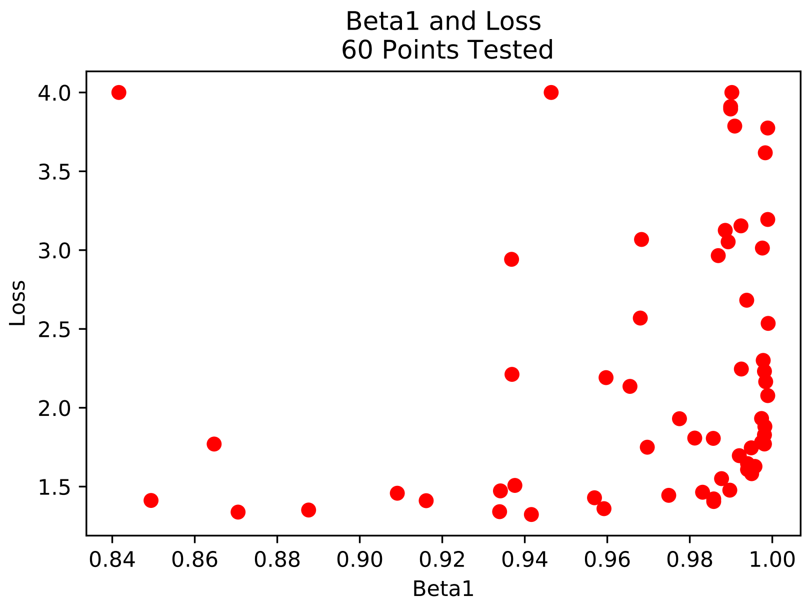

Figure 5: Loss with different learning rate

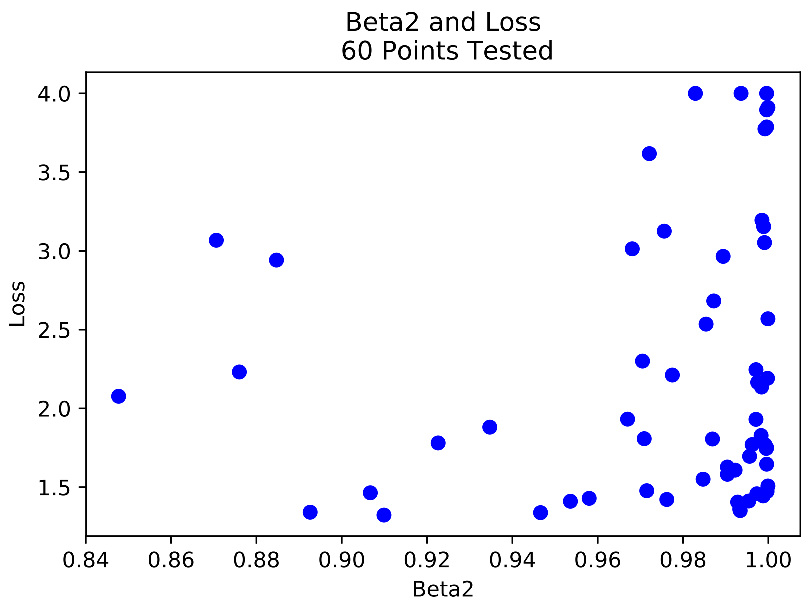

Figure 6: Loss with different learning rate

3.2. Stage 2

In stage 2 we focus on finding the hyperparamters with best performance for each method, so that we can compare the performance between these 3 methods. First, three best samples of each method from stage 1 are selected. Then, a series of random search are performed near these samples.

In the beginning my search space is large (i.e. $\eta * 0.5 - \eta * 2$), and it turns out to be inefficient. Then I narrow the space down to $\eta * 0.99 - \eta * 1.01$. The results of stage 2 searching are shown in figure 7,8, and 9. The red dots refer to the best loss samples in stage 1. As can be seen from figure 7,8, and 9, searching near the best samples does improve the performance.

Finally, it is time to compare the performance of these three methods. Table 3 lists the best 5 samples each from the three methods.

From table 3, we have several findings:

- In this project, we do not observe an obvious performance improvement by using Adam and RMSProp. I think the reason is this: As we have mentioned above, the learning rate in SGD is very high and we only run the samples for 2 epochs. A high learning rate initially could cause the loss to decrease sharply, but later the loss will increases rather than decreases.

We have proved this in the above experiments. By comparison, RMSProp and Adam are using relatively smaller learning rates, so the loss is more likely to decrease smoothly. To sum up, I think it is just an illusion that SGD is doing as good as Adam and RMSProp, as a result of high learning rate and a small number of training epochs.

- It is also worth noticing that, in general, Adam is doing good. The variance of loss in Adam is the smallest (only from 1.26 to 1.28), indicating its stability.

	Loss		
Number	SGD(η)	RMSProp(η)	Adam(η)
0	1.2604(9.57)	1.2598(3.06)	1.2628(5.97)
1	1.2705(9.59)	1.2786(3.29)	1.2635(1.53)
2	1.2777(9.46)	1.2868(3.08)	1.2667(1.86)
3	1.2787(9.56)	1.2938(3.06)	1.2710(1.87)
4	1.30254(5.95)	1.2982(3.45)	1.2830(1.54)

Table 3: Loss for 5 Best-Performing Samples with 2 epochs

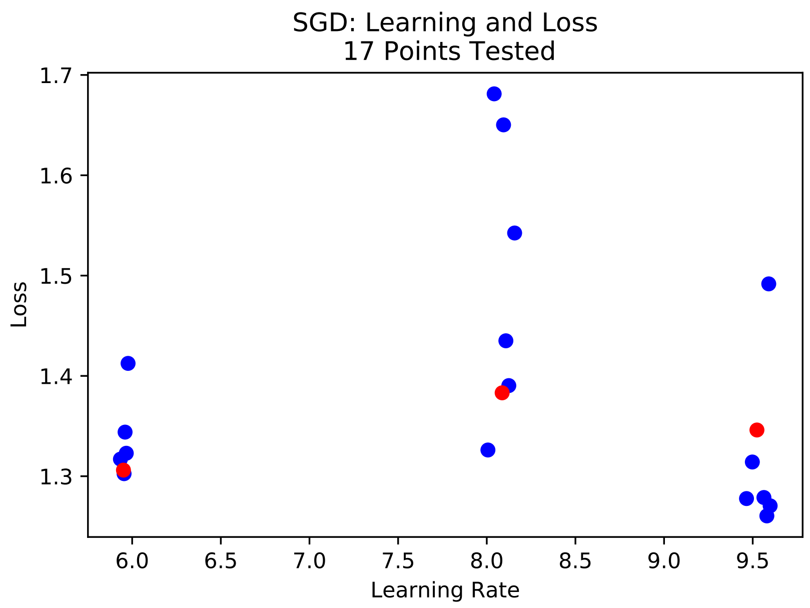

Figure 7: SGD Loss with different learning rate

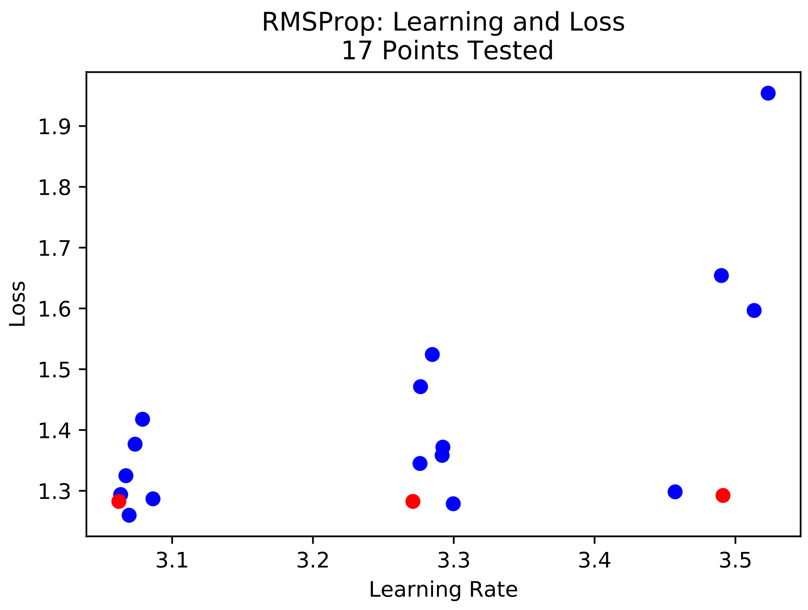

Figure 8: RMS Loss with different learning rate

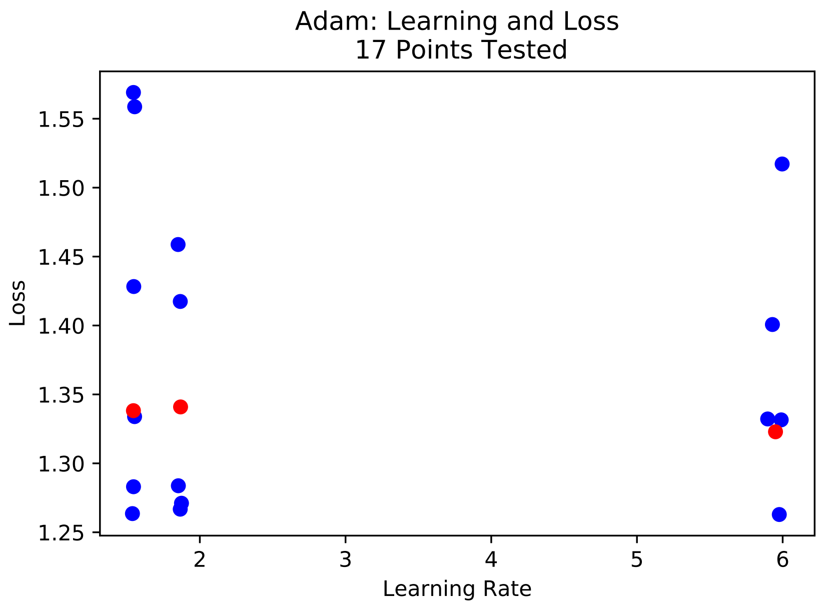

Figure 9: Adam Loss with different learning rate

4. Summary

In this project we were trying to compare the performance of SGD, RMSProp and Adam. We mainly used random search as the approach to find best hyper-parameters. We staged the searching process so that our searching was more efficient and accurate. In the end, base on the data we collected, we compared the results and came to a conclusion. Even though we did not get the most common result that we had expected (i.e. Adam better than RMSProp, RMSProp better than SGD), we introduced possible reasons behind it.

However, there are indeed a few things to improve. First, due to the limited amount of time, we only performed 1 epoch for stage 1. This had a significant weakness that a very large learning rate in SGD could become our best choices. But if we had time to run a few more epochs, we could avoid this from happening. Furthermore, in very beginning of stage 2, I was too optimistic about the search space so that the search did not find any useful sample. Later I had to narrow the search scope down to a really small interval to get some better samples. I should have tuned the search space more carefully to get the best results.

In summary, this project did follow the ideas in project proposal and completed the expected tasks. Nevertheless, given more time, this project could do a better job.

References

[1] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[2] A. Karpathy, "Cs231n convolutional neural networks for visual recognition," *CS231 Course Websites http://cs231n.github.io/neural-networks-3/,[Online]*, 2016.

[3] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[4] G. Hinton, N. Srivastava, and K. Swersky, "Lecture 6a overview of mini–batch gradient descent," *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online]*, 2012.

[5] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2011.

[6] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[7] S. Funk, "Rmsprop loses to smorms3 - beware the epsilon!," *Blog http://sifter.org/ simon/journal/20150420.html,[Online]*, 2015.

[8] M. E. Khan, R. Babanezhad, W. Lin, M. Schmidt, and M. Sugiyama, "Faster stochastic variational inference using proximal-gradient methods with general divergence functions," *arXiv preprint arXiv:1511.00146*, 2015.