A Novel Optimization Method for Recurrent Neural Network by Non-Euclidean Geometry

Chen Liu
Master Student, IN, EPFL
chen.liu@epfl.ch

Ya-Ping Hsieh
Supervisor, PH.D, EL, EPFL
ya-ping.hsieh@epfl.ch

Volkan Cevher
Supervisor, Associate Professor, EL, EPFL
volkan.cevher@epfl.ch

Abstract

Recurrent neural network(RNN) is widely used in temporal applications. Traditionally, RNN is trained using stochastic gradient descent, which is based on Euclidean geometry. In this report, we introduce a novel optimization method based on non-Euclidean geometry to train RNN. We will argue that it better estimates the gradient to the minimum of the object function by theoretical analysis and experiments.

1 Introduction

Compared with other neural network models, recurrent neural network(RNN) is more powerful to process sequential data. It is widely used in fields like natural language processing([1]) and machine translation([2]). RNN's unique recurrent connection correlates consecutive temporal states and this structure can be incorporated into other neural network models. One example is recurrent convolutional neural networks([3]), which combines recurrent connections with convolutional neural network([4]) and outperforms state-of-art models in image classification tasks in 2015.

On the other hand, recurrent connections significantly increase the complexity and sensitively of the neural network models. Generally, it is more difficult to train models with recurrent connection. The most important strategy to train RNN is called backpropagation through time(BPTT, [5]). BPTT backpropagates gradient both spatially and temporally, which can estimate the gradient of each parameter effectively and efficiently. However, training such temporal-dependent neural network by BPTT usually required fined-tuned hyper-parameters like learning rate.([6])

If we regard training a neural network as an unconstrained optimization problem where we need to minimize a function $f(\mathbf{x})$, usually non-convex, in the space $\mathbf{x} \in \mathbb{R}^n$.Our task is to find a position whose value is as close as possible to the global optimum. We can find that most gradient-based optimization methods are based on Euclidean geometry. That is to say, the norm is define as Euclidean norm $\| - \|_2$. For example, in classic stochastic gradient descent(SGD), $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k)$. We have $f(\mathbf{x}_{k+1}) \simeq f(\mathbf{x}_k) - \eta \|\nabla f(\mathbf{x}_k)\|_2$ using first-order Taylor expansion.

Recently, some optimization methods based on non-Euclidean are proposed. They use gradient based on maximum norm instead of Euclidean norm. Mathematically, they are inspired from a nice property of log-sum-exp function $h(\omega, \mathbf{x}) = \log \sum_i^n \omega_i e^{x_i}$, which is usually the form of many models' loss function. This kind of methods is called stochastic spectral descent(SSD) and will be demonstrated in the following chapters. SSD has achieved success over SGD in training restricted Boltzmann machine([7]), multi layer perceptron([8]) and convolutional neural network([8]) on various datasets. RNN is quite different from these models because it is time-dependent. In the following chapters, we will apply SSD's idea to 1-layer RNN, the simplest version, and show its superiority over traditional SGD.

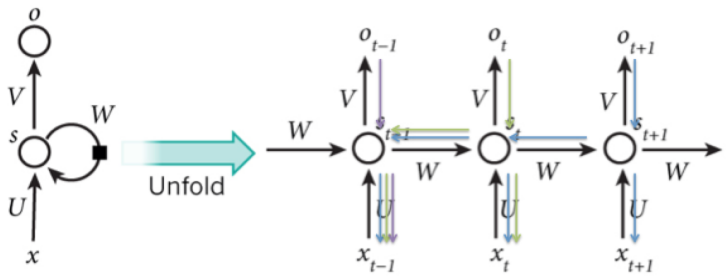


Figure 1: Structure of recurrent neural network and unfolding it via temporal dimension

In chapter 2, we will review some basic knowledge about our research, including recurrent neural network(RNN) and stochastic spectral descent(SSD) for other neural networks models. Our research topic is inspired by the similar form of loss function between RNN and these models. In chapter 3, we will prove the loss function of RNN can be bounded using Schatten-∞ and propose our learning strategy for different parameters. Detailed proof of this part is available in Appendix. Next in chapter 4, we will demonstrate our experiment results both on stimulated sequences and real world data. Finally we draw a short conclusion in the last chapter.

2 Preliminaries and Models

In this chapter, we will review some basic knowledge our research is based on. In this and the followings chapters, we use bold lowercase letters to denote vectors and bold upper case letters to denote matrices.

2.1 Recurrent Neural Network

Recurrent neural network(RNN) is a temporal-dependent neural network model whose structure is shown in Figure 1. The parameters of RNN includes initial hidden state $\mathbf{s}_0$, input matrix $\mathbf{U}$, recurrent connection $\mathbf{W}$ and output matrix $\mathbf{V}$. If we unfold RNN in temporal dimension, like the right part of Figure 1, we can get a clearer view of RNN. The hidden state is the addition of the input information and the recurrent information (memory). The output part is the same as other neural network models with softmax as their classifiers.

$$\mathbf{s}_t = \sigma(\mathbf{W}\mathbf{s}_{t-1} + \mathbf{U}\mathbf{x}_t) \tag{1}$$
$$\mathbf{o}_t = softmax(\mathbf{V}\mathbf{s}_t) \tag{2}$$

We denote the input dimension, hidden states dimension, output dimension and length of sequences as N, H, K and M respectively. The loss function of RNN is defined as the average Boltzmann's entropy between predict label $\mathbf{o}_t$ and the real label $\mathbf{y}_t$.

$$E(\mathbf{U}, \mathbf{W}, \mathbf{V}, \mathbf{s}_0) = \frac{1}{M}\sum_{t=1}^{M} \mathbf{y}_t^T \log \mathbf{o}_t \tag{3}$$

The real label $\mathbf{y}_t$ is an one-hot vector, so equation 3 can be simplified as below.

$$\begin{aligned} E &= -\frac{1}{M}\sum_{t=1}^{M}\sum_{i=1}^{K} \mathbf{y}_{t,i} \log \mathbf{o}_{t,i} \\ &= -\frac{1}{M}\sum_{t=1}^{M}\sum_{i=1}^{K} \mathbf{y}_{t,i} \log \frac{e^{\mathbf{V}_{i,:}\mathbf{s}_t}}{\sum_{j=1}^{K} e^{\mathbf{V}_{j,:}\mathbf{s}_t}} \\ &= \frac{1}{M}\sum_{t=1}^{M}(\log(\sum_{j=1}^{K} e^{\mathbf{V}_{j,:}\mathbf{s}_t}) - \mathbf{y}_t^T \mathbf{V}\mathbf{s}_t) \end{aligned} \tag{4}$$

To train a neural network model, we often use backpropagation to calculate each parameter's gradient. For RNN, we use the strategy called backpropagation through time(BPTT, [5]) which backpropagates the gradient via both spatial and temporal dimension (up to down and right to left in Figure 1). Figure 1 shows this process where blue, green and purple denotes the gradient from the (t+1)-th, t-th and (t-1)-th token.

2.2 Stochastic Spectral Descent

Consider the minimization of a function $F(\mathbf{x})$ with Lipschitz gradient. L_p is its Lipschitz constant.

$$\|\triangledown F(\mathbf{x}_1) - \triangledown F(\mathbf{x}_2)\|_q \leq L_p \|\mathbf{x}_1 - \mathbf{x}_2\|_p \tag{5}$$

Here $\|-\|_p$ and $\|-\|_q$ are conjugated norms i.e $p^{-1} + q^{-1} = 1$. It can be proved that $F(\mathbf{x})$ admits a global upper bound.([9])

$$F(\mathbf{y}) \leq F(\mathbf{x}) + \langle \triangledown F(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L_p}{2} \|\mathbf{y} - \mathbf{x}\|_p^2 \tag{6}$$

Consider an iteration rule based on MM method i.e find a $\mathbf{y}$ to minimize the upper bound for a given $\mathbf{x}$ in formula 6

$$\mathbf{x}_{k+1} \in argmin_{\mathbf{y}} F(\mathbf{x}_k) + \langle \triangledown F(\mathbf{x}_k), \mathbf{y} - \mathbf{x}_k \rangle + \frac{L_p}{2} \|\mathbf{y} - \mathbf{x}_k\|_p^2 \tag{7}$$

To solve problem 7, we need to introduce #-operator([10]).

$$\mathbf{s}_p^{\#} = argmax_{\mathbf{x}} \{\langle \mathbf{s}, \mathbf{x} \rangle - \frac{1}{2} \|\mathbf{x}\|_p^2\} \tag{8}$$

Then, the iteration rule can be formulated as:([11])

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{L} [\triangledown F(\mathbf{x}_k)]_p^{\#} \tag{9}$$

if $p = 2$, $x^{\#} = x$. Such generalized gradient descent reduces to stochastic gradient descent(SGD). For generalized gradient descent, we can estimate the converging speed if we have L_p.([9])

$$F(\mathbf{x}_k) - F(\mathbf{x}^*) = O(\frac{L_p R^2}{k}) \tag{10}$$

x^* is the optimum and $R = \|x_0 - x^*\|_p$ means the distance from the initial points to optimum. There is a trade-off when we choose the value of p. When we increase p, the value of R will get smaller but L_p tends to be larger([11]). For log-sum-exp function $h(\omega, \mathbf{x}) = \log \sum_i^n \omega_i e^{x_i}$, we have $L_2 = \frac{1}{4}$ and $L_\infty = \frac{1}{2}$.([7]) Both of them are independent of the dimension of $\mathbf{x}$. However, $\|-\|_2$ roughly increases linearly with the dimension while $\|-\|_\infty$ is less dependent on dimension. So we can say $L_2 R_2^2 >> L_\infty R_\infty^2$ when $\mathbf{x}$ is high-dimensional. This is exactly the case of many neural network models, where the number of parameters is very large and the loss function is often in the form of log-sum-exp.

Stochastic spectral descent(SSD) is based on these mathematical theorems. Instead of updating the parameters along the Euclidean gradient i.e $\triangledown f(\mathbf{x})$, SSD uses the gradient in non-Euclidean geometry i.e $[\triangledown f(\mathbf{x})]_p^{\#}$ where $p = \infty$. Due to the nice property of log-sum-exp function, we can expect the the non-Euclidean geometry can better exploit the gradient of the loss function i.e a tighter uppper bound of formula 6. For a specific neural network model, we need to estimate the Lipschitz constant of their loss function and different tricks are applied there.

3 Our Proposed Model

In this chapter, we will demonstrate the application of stochastic spectral descent(SSD) in 1-layer RNN. In our derivation, we will use 3-d tensor and generally we use uppercase letters to denote tensors. Let $\mathbf{M}$ be a matrix, $\tilde{\mathbf{M}}$ is defined as a 3-d tensor whose every 2-d slice is $\mathbf{M}$ i.e $\forall i\ \tilde{\mathbf{M}}[i] = \mathbf{M}$. The multiplication rule of 3-d tensor is the same as torch.([12]) That is to say, if $\mathbf{P}$ and $\mathbf{Q}$ are two 3-d tensors of shape $K \times N \times M$ and $K \times M \times L$, then $\mathbf{R} = \mathbf{PQ}$ is a tensor of shape $K \times N \times L$ and $\forall i\ R[i] = P[i]Q[i]$.

3.1 The Gradient of Input and Recurrent Matrices

From equation 1, we calculate its derivatives with respect to $\mathbf{U}$ and $\mathbf{W}$.

$$\frac{\partial[\mathbf{s}_t(\mathbf{W})]_p}{\partial \mathbf{W}_{ij}} = \sigma'(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1})_p(\mathbf{s}_{t-1,j}\delta_{ip} + \sum_{k=1}^{H} \frac{\partial[\mathbf{s}_{t-1}(\mathbf{W})]_k}{\partial \mathbf{W}_{ij}}\mathbf{W}_{pk}) \tag{11}$$

$$\frac{\partial[\mathbf{s}_t(\mathbf{U})]_p}{\partial \mathbf{U}_{ij}} = \sigma'(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1})_p(\mathbf{x}_{t,j}\delta_{ip} + \sum_{k=1}^{H} \frac{\partial[\mathbf{s}_{t-1}(\mathbf{U})]_k}{\partial \mathbf{U}_{ij}}\mathbf{W}_{pk}) \tag{12}$$

Here δ_{ip} is impulse function, it equals to 1 iff $i = p$ and 0 otherwise. If we let $\mathbf{P}_t[i,j,k] = \frac{\partial[\mathbf{s}_t]_k}{\partial \mathbf{W}_{ij}}$, $\mathbf{Q}_t[i,j,k] = \frac{\partial[\mathbf{s}_t]_k}{\partial \mathbf{U}_{ij}}$, $\lambda_t = \sigma'(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1})$, $\mathbf{\Lambda}_t = diag(\lambda_t)$, $\bar{\mathbf{S}}_t[i,j,k] = \mathbf{s}_{t,j}\delta_{ik}$ and $\bar{\mathbf{X}}_t[i,j,k] = \mathbf{x}_{t,j}\delta_{ik}$, we can get the matrix form of the equation above. We have the value of each matrix or tensor except $\mathbf{P}_t$ and $\mathbf{Q}_t$. Since $\mathbf{P}_0$ and $\mathbf{Q}_0$ are all-zero tensor, we can obtain $\mathbf{P}_t$ and $\mathbf{Q}_t$ by iterating the following equations.

$$\mathbf{P}_t = (\bar{\mathbf{S}}_{t-1} + \mathbf{P}_{t-1}\tilde{\mathbf{W}}^T)\tilde{\mathbf{\Lambda}'}_t \tag{13}$$

$$\mathbf{Q}_t = (\bar{\mathbf{X}}_t + \mathbf{Q}_{t-1}\tilde{\mathbf{W}}^T)\tilde{\mathbf{\Lambda}'}_t \tag{14}$$

From equation 4, we can calculate the loss function's gradient with respect to $\mathbf{s}_t$. E_t is t-th token's contribution to the overall loss.

$$\frac{\partial E_t}{\partial \mathbf{s}_t} = \mathbf{V}^T(softmax(\mathbf{V}\mathbf{s}_t) - \mathbf{y}_t) \tag{15}$$

Then we can obtain the Euclidean gradient of $\mathbf{U}$ and $\mathbf{W}$.

$$\frac{\partial E_t}{\partial \mathbf{U}} = \frac{\partial E_t}{\partial \mathbf{s}_t}\frac{\partial \mathbf{s}_t}{\mathbf{U}} = \mathbf{Q}_t\mathbf{V}^T(\tilde{softmax}(\mathbf{V}\mathbf{s}_t) - \mathbf{y}_t) \tag{16}$$

$$\frac{\partial E_t}{\partial \mathbf{W}} = \frac{\partial E_t}{\partial \mathbf{s}_t}\frac{\partial \mathbf{s}_t}{\partial \mathbf{W}} = \mathbf{P}_t\mathbf{V}^T(\tilde{softmax}(\mathbf{V}\mathbf{s}_t) - \mathbf{y}_t) \tag{17}$$

Equation 4 shows that E_t has two parts, one is log-sum-exp function and the other is linear. Let $\mathbf{\Phi} = \{\mathbf{U}, \mathbf{W}\}$, we can prove the following relations using the fact Lipschitz constant L_∞ of log-sum-exp is $\frac{1}{2}$. The detailed proof is available in Appendix A.

$$\begin{aligned} E_t(\mathbf{\Phi} + \Delta\mathbf{\Phi}) \leq E_t(\mathbf{\Phi}) + \langle \nabla_{\mathbf{\Phi}} E_t(\mathbf{\Phi}), \Delta\mathbf{\Phi}\rangle + \frac{1}{2}\|\mathbf{V}\mathbf{s}_t(\mathbf{\Phi} + \Delta\mathbf{\Phi}) - \mathbf{V}\mathbf{s}_t(\mathbf{\Phi})\|_\infty^2 + \\ 2\|\mathbf{V}\mathbf{s}_t(\mathbf{\Phi} + \Delta\mathbf{\Phi}) - \mathbf{V}\mathbf{s}_t(\mathbf{\Phi}) - \langle \mathbf{V}\nabla\mathbf{s}_t(\mathbf{\Phi}), \Delta\mathbf{\Phi}\rangle\|_\infty \end{aligned} \tag{18}$$

For input matrix $\mathbf{U}$, we can bound the last two parts of formula 18 and get Lipschitz constant $L_\mathbf{U}$ of E_t as a function of $\mathbf{U}$. The matrix norm we use here is Schatten-∞ norm.([13]) $\|\mathbf{Q}_{t_{:,:,p}}\|_{S^1}$ mean Schatten-1 norm of tensor slice i.e matrix $\mathbf{Q}[:,:,p]$. Likewise, $\|\mathbf{Q}_{t_{:,:,*}}\|_{S^1}$ is a vector whose p-th element is $\|\mathbf{Q}_{t_{:,:,p}}\|_{S^1}$. The proof of the following bounds are in Appendix B.1.

$$\|\mathbf{V}\mathbf{s}_t(\mathbf{U}^k + \Delta\mathbf{U}) - \mathbf{V}\mathbf{s}_t(\mathbf{U}^k)\|_\infty \leq \max_q \|\mathbf{Q}_t\tilde{\mathbf{V}}_q\|_{S^1}\|\Delta\mathbf{U}\|_{S^\infty} \tag{19}$$

$$\begin{aligned} &\|\mathbf{V}\mathbf{s}_t(\mathbf{U}^k + \Delta\mathbf{U}) - \mathbf{V}\mathbf{s}_t(\mathbf{U}^k) - \langle \mathbf{V}\nabla_\mathbf{U}\mathbf{s}_t(\mathbf{U}^k), \Delta\mathbf{U}\rangle\|_\infty \\ &\leq \frac{1}{2}[\max_q \|\mathbf{V}_q\|_2\|\mathbf{x}_t\|_2 max_p(\lambda_p'^{-1}\lambda_p''\|\mathbf{Q}_{t-1_{:,:,p}}\|_{S^1}) \\ &+ \max_q \|\mathbf{V}_q\|_2\|\mathbf{W}\|_{S^\infty}\|\|\mathbf{Q}_{t-1_{:,:,*}}\|_{S^1}\|_2\|\max_p(\lambda_p'^{-1}\lambda_p''\|\mathbf{Q}_{t-1_{:,:,p}}\|_{S^1})]\|\Delta\mathbf{U}\|_{s^\infty}^2 \end{aligned} \tag{20}$$

$$L_{\mathbf{U}} = \frac{1}{2}\max_q \|\mathbf{Q}_t\tilde{\mathbf{V}}_q\|_{S^1}^2 + \max_q \|\mathbf{V}_q\|_2\|\mathbf{x}_t\|_2 max_p(\lambda_p'^{-1}\lambda_p''\|\mathbf{Q}_{t-1_{:,:,p}}\|_{S^1}) \\ + \max_q \|\mathbf{V}_q\|_2\|\mathbf{W}\|_{S^\infty}\|\|\mathbf{Q}_{t-1_{:,:,*}}\|_{S^1}\|_2\|\max_p(\lambda_p'^{-1}\lambda_p''\|\mathbf{Q}_{t-1_{:,:,p}}\|_{S^1}) \tag{21}$$

Similarly, we can obtain Lipschitz constant $L_{\mathbf{W}}$ of E_t as a function of $\mathbf{W}$, which is also based on Schatten-∞ norm. The proof of this part is included in Appendix B.2.

$$\|\mathbf{V}\mathbf{s}_t(\mathbf{W}^k + \Delta\mathbf{W}) - \mathbf{V}\mathbf{s}_t(\mathbf{W}^k)\|_\infty \leq \max_q \|\mathbf{P}_t\tilde{\mathbf{V}}_q\|_{S^1}\|\Delta\mathbf{W}\|_{S^\infty} \tag{22}$$

$$\|\mathbf{V}\mathbf{s}_t(\mathbf{W}^k + \Delta\mathbf{W}) - \mathbf{V}\mathbf{s}_t(\mathbf{W}^k) - \langle\mathbf{V}\nabla_{\mathbf{W}}\mathbf{s}_t(\mathbf{W}^k), \Delta\mathbf{W}\rangle\|_\infty \\ \leq \frac{1}{2}[\max_q \|\mathbf{V}_q\|_2\|\mathbf{s}_{t-1}\|_2 \max_p(\lambda_p'^{-1}\lambda_p''\|\mathbf{P}_{t-1_{:,:,p}}\|_{S^1}) + 2\max_q \|\mathbf{V}_q\|_2\|\lambda_t'\|_\infty\|\|\mathbf{P}_{t-1_{:,:,*}}\|_{S^1}\|_2 \\ + \max_q \|\mathbf{W}\|_{S^\infty}\|\mathbf{V}_q\|_2 \max_p(\lambda_p'^{-1}\lambda_p''\|\mathbf{P}_{t-1_{:,:,p}}\|)\|\|\mathbf{P}_{t-1_{:,:,*}}\|_{S^1}\|_2]\|\Delta\mathbf{W}\|_{S^\infty}^2 \tag{23}$$

$$L_{\mathbf{W}} = \frac{1}{2}(\max_q \|\mathbf{P}_t\tilde{\mathbf{V}}_q\|_{S^1})^2 + \max_q \|\mathbf{V}_q\|_2\|\mathbf{s}_{t-1}\|_2 \max_p(\lambda_p'^{-1}\lambda_p''\|\mathbf{P}_{t-1_{:,:,p}}\|_{S^1}) \\ +2\max_q \|\mathbf{V}_q\|_2\|\lambda_t'\|_\infty\|\|\mathbf{P}_{t-1_{:,:,*}}\|_{S^1}\|_2 + \max_q \|\mathbf{V}_q\|_2\|\mathbf{W}\|_{S^\infty}\|\|\mathbf{P}_{t-1_{:,:,*}}\|_{S^1}\|_2 \max_p(\lambda_p'^{-1}\lambda_p''\|\mathbf{P}_{t-1_{:,:,p}}\|) \tag{24}$$

From formulas above, we can conclude that the loss function can be bounded like formula 6 and p is Schatten-∞ norm. Because of the nice property of log-sum-exp function, we can expect this is a tighter bounded than $p = 2$ scenario. In the implementation part, we do not calculate the exact value of $L_{\mathbf{U}}$ and $L_{\mathbf{W}}$ and use a fine-tuned constant redefined instead, which is like learning rate in SGD. This is because computing precise values of expressions like $\|\mathbf{P}_{t_{:,:,*}}\|_{S^1}$ are very expensive.

3.2 The Gradient of Output Matrix

The gradient of output matrix $\mathbf{V}$ is much easier to calculate. From equation 4, we can get the gradient of E_t with respect to $\mathbf{V}$ directly.

$$\frac{\partial E_t}{\partial \mathbf{V}} = softmax(\mathbf{V}\mathbf{s}_t)\mathbf{s}_t^T - \mathbf{y}_t\mathbf{s}_t^T = (softmax(\mathbf{V}\mathbf{s}_t) - \mathbf{y}_t)\mathbf{s}_t^T \tag{25}$$

According to equation 4, E_t has two parts: log-sum-exp part with respect to $\mathbf{V}\mathbf{s}_t$ and the linear part. Thus, we can draw an upper bound of E_t with respect to $\mathbf{V}$ using the property of log-sum-exp function.

$$E_t(\mathbf{V} + \Delta\mathbf{V}) \leq E_t(\mathbf{V}) + \langle\nabla_{\mathbf{V}\mathbf{s}_t}E_t(\mathbf{V}), \Delta\mathbf{V}\mathbf{s}_t\rangle + \frac{1}{2}\|\Delta\mathbf{V}\mathbf{s}_t\|_\infty^2 \\ \leq E_t(\mathbf{V}) + \langle\nabla_{\mathbf{V}}E_t(\mathbf{V}), \Delta\mathbf{V}\rangle + \frac{1}{2}\|\mathbf{s}_t\|_2^2\|\Delta\mathbf{V}\|_{S^\infty}^2 \tag{26}$$

Like matrix $\mathbf{U}$ and $\mathbf{W}$, we can get Lipschitz constant for matrix $\mathbf{V}$ in Schatten-∞ norm, which is much simpler than those of $\mathbf{U}$ and $\mathbf{W}$.

$$L_{\mathbf{V}} = \frac{1}{2}\|\mathbf{s}_t\|_2^2 \tag{27}$$

However, this is only a theoretical upper bound of the loss function. In implementation, we can not use SSD to update matrix $\mathbf{V}$, otherwise we will get very poor performance. From equation 25, $\nabla_{\mathbf{V}}E_t$ is the outer product of two vectors and it is always rank-deficient(of rank 1). The #-operator that minimize the expression 8 in Schatten-∞ norm is given by the singular value decomposition(SVD) of the original matrix.([11]) However, the SVD of a seriously rank-deficient is very sensitive and imprecise.([14]) For $[\nabla_{\mathbf{V}}E_t]^\#$ determines the direction of the gradient, a imprecise calculation of the gradient will catastrophically hurt the performance. As a result, to update matrix $\mathbf{V}$, we still use traditional Euclidean gradient.

3.3 The Gradient of Initial States

For initial states, we still use traditional Euclidean gradient to update it. Like matrix $\mathbf{U}$ and $\mathbf{W}$, the gradient of $\mathbf{s}_0$ is calculated iteratively.

$$\frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_0} = \mathbf{\Lambda}_t' \mathbf{W} \frac{\partial \mathbf{s}_{t-1}}{\partial \mathbf{s}_0} \tag{28}$$

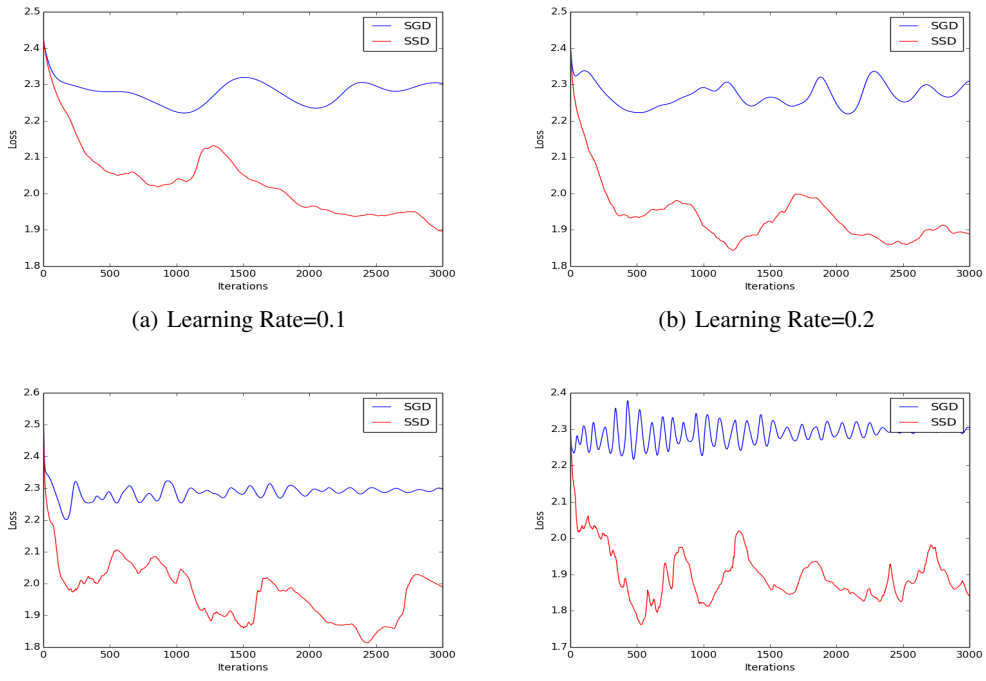
$$\frac{\partial E_t}{\partial \mathbf{s}_0} = \frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_0}^T \frac{\partial E_t}{\partial \mathbf{s}_t} \tag{29}$$

4 Experiments

In this chapter, we will compare stochastic spectral descent(SSD) with traditional stochastic gradient descent(SGD). In the first experiment We test both optimization methods on a sequence of data generated randomly. In the next one, we will test them on a real and bigger dataset called ATIS([15]). In both experiments, we will clearly see SSD's superiority over SGD.

4.1 Fitting Random Sequences

In this part, we generate a sequences of data randomly to compare the performance between SGD and SSD. The length of the sequence is S and the dimension of the input and label of each token is N and K respective, aligned with RNN's input and output dimension. The data is generated from a Gaussian distribution $N(\mu, \sigma^2)$ i.i.d.



(a) Learning Rate=0.1

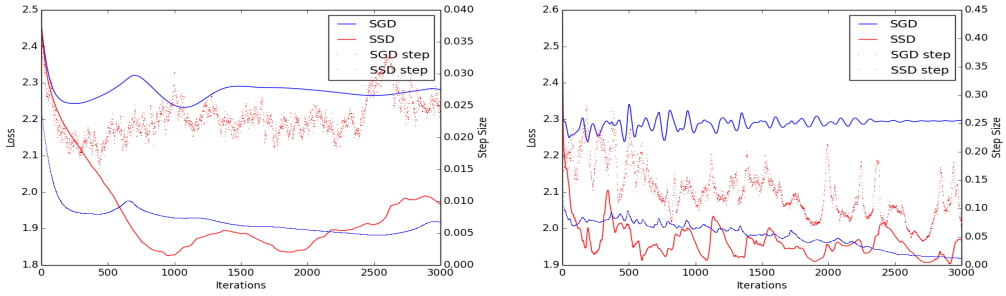
(b) Learning Rate=0.2

(c) Learning Rate=0.5

(d) Learning Rate=1.0

Figure 2: Learning curve on the random generated data in different learning rate settings.

We train RNN to fit this generated 50-token sequence from standard Gaussian for 3000 iterations using SGD and SSD respectively from the same initial position. The input dimension and output dimension of RNN are both 10. The dimension of the hidden states is 5. Figure 2 shows the trend of loss under different settings. It is obvious that SSD outperforms SGD on such random generated data. We can see that SGD always get stuck in a sub-optimal position while SSD can explore more



(a) Learning Rate=0.1 (b) Learning Rate=1.0

Figure 3: Learning curve(solid line) and step size(dotted line) on the random generated data in different learning rate settings. Due to random initial position, the learning curve is not exactly the same as the corresponding subfigures in Figure 2

and can reach an optimized position. This is confirmed in Figure 3, where we also plot the step size of each iteration in dotted line. Let $\mathbf{P}_k$ be the vector containing all parameters at iteration k, the step size is defined as Euclidean norm $\|\mathbf{P}_{k+1} - \mathbf{P}_k\|_2$. In Figure 3, the quickly vanishing step size of SGD indicates it get stuck in a local optimum. On the contrary, the fluctuating step size SSD shows it is able to explore more to find more optimized position.

Compare the learning curve under different learning rate settings, we can find in Figure 2(d) SGD diverges while SSD can still find a relatively optimal position. This indicates SSD is more robust under different learning rate settings. This property brings convenience when we have to tune learning rates in high dimensional hyper-parameter space.

4.2 Fitting Real Dataset

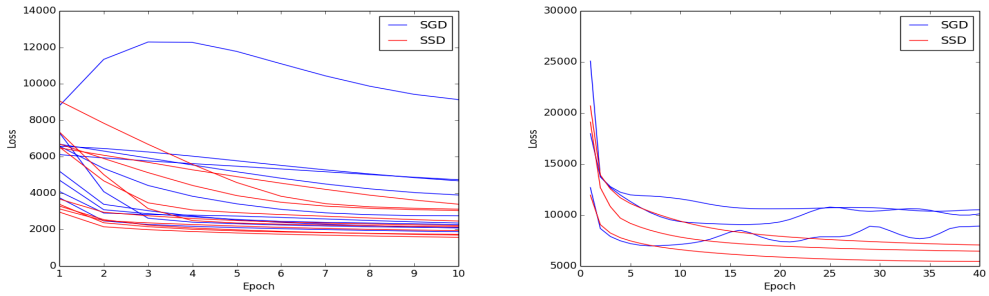
The Airline Travel Information System(ATIS) collected by DARPA ([15]) is a popular benchmark that is suitable for testing recurrent neural network. It contains 4978 sentences in training set and 893 sentences in testing set. The aim of our experiment is to compare the ability of SGD and SSD to find the optimal value of a function, so we only use the training set and concentrate on the trend of the loss through the training process.

All sentences in ATIS are about flight like *Show flights from Boston to New York today*. Each word of each sentence in ATIS is assigned a label from 1 to 128, meaning *departure place*, *destination time* etc. Our task is to predict the label of each word in a given sentence. It is necessary to point out that this dataset is highly unbalanced and most of words are assigned *label 126*, meaning *nothing special*.

We use 'sliding window' of size 3 to predict the label of each word. That is to say, we use the consecutive word before and after a word plus itself to predict its label. Before throwing words into RNN, we should use word embedding algorithms to turn words into vectors. In this experiment, we use the word representation *GoogleNews-vectors-negative300.bin.gz* available at code.google.com/archive/p/word2vec/. It is trained from Google news by word2vec algorithm([16]) and each word has 300 dimensions. To reduce our model complexity, we use principle component analysis(PCA, [17]) to project the word representations into a 10-dimensional space. For words that do not appear in the Google news corpus, we assigned them a default word vector generated randomly.

Now, we can specify the structure of our 1-layer RNN. The dimensions of its input, hidden states and output are 30, 50 and 128 respectively. The learning rate search space of this problem is very large, because we need to set a reasonable learning rate to update $\mathbf{U}$, $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{s}_0$ separately. To explore all possibilities, we use random search to optimize hyper-parameters([18]). That is for each individual run, we set all kinds of learning rates randomly and independently from a distribution. In this experiment, we let the logarithm of all learning rate obeys uniform distribution $U(-3, -1)$.

We generate 10 sets of learning rate configuration for both SGD and SSD. Figure 4.2 shows their learning curves. We run SGD or SSD for 10 epochs and the top three smallest values of final epoch's loss are all using SSD. Using various learning rates and initial positions, it is clear that the variance of SSD's performance is much smaller than SGD's. SGD even diverges on one learning rate configuration out of ten. This phenomenon reinforce the argument in Section 4.1: SSD is more robust than SGD on various values of learning rate.



(a) Learning Curve of SGD and SSD Using Random Generated Learning Rate.

(b) Learning Curve of SGD and SSD for 40 Epochs.

Figure 4: Learning Curves on ATIS

Learning Rate				SGD		SSD	
U	**W**	**V**	$\mathbf{s}_0$	Loss	Time(s)	Loss	Time(s)
0.02	0.02	0.05	0.02	9429.26	40936	6425.21	42299
0.01	0.025	0.024	0.02	9027.36	41801	6973.68	42950
0.01	0.01	0.05	0.01	6850.28	41247	6189.67	42467
0.01	0.003	0.05	0.005	6301.56	41800	6104.17	43021

Table 1: Loss after 10 Epochs and Run Time of Some Fine-tuned Learning Rates. All Configurations are Run on the Same Machine.

From the random search, we can have a rough view of 'reasonable' learning rate choice that can drive the loss to an optimal value quickly and smoothly. Table 1 shows some results of fine-tuned learning rates. It is clear that SSD outperforms SGD under each settings and SSD's performance is more stable. The running time of SSD is a bit longer than SGD, this is because #-operator in SSD needs SVD and contributes to the extra time cost.

If we train the model for more epochs, we can find that SSD's learning curve is more smooth than SGD. In Figure 4.2, we train the model for 40 epochs using 3 sets of learning rates by SGD and SSD respectively. The learning curves of SSD have fewer fluctuations and always reach a more optimal position than SGD.

5 Conclusion and Future Work

In the chapters above, we have demonstrated a novel optimization method called stochastic spectral descent(SSD) to train 1-layer recurrent neural network(RNN) in non-Euclidean geometry. By experimenting on simulated sequences and real dataset, we show its superiority over traditional stochastic gradient descent(SGD). SSD can reach the optimal position more efficiently, more smoothly and more robustly using different learning rates.

While we are now training SGD or SSD using constant learning rate, there are many tricks to use adaptive learning rate to improve the classic optimization methods, including momentum, adagrad, adadelta([19]). On the other hand, we only experiment on a natural language labeling problem(ATIS). This task is relatively simple since most tokens in the dataset are assigned one label. To compare SGD and SSD comprehensively, we need to try other tasks and dataset of different properties.

6 Reference

[1]. Hermans, Michiel, and Benjamin Schrauwen. *"Training and analysing deep recurrent neural networks."* Advances in Neural Information Processing Systems. 2013.

[2]. Cho, Kyunghyun, et al. *"Learning phrase representations using RNN encoder-decoder for statistical machine translation."* arXiv preprint arXiv:1406.1078 (2014).

[3]. Liang Ming, and Xiaolin Hu. *"Recurrent convolutional neural network for object recognition."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[4]. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. *"Imagenet classification with deep convolutional neural networks."* Advances in neural information processing systems. 2012.

[5]. Werbos, Paul J. *"Backpropagation through time: what it does and how to do it."* Proceedings of the IEEE 78.10 (1990): 1550-1560.

[6]. Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. *"Learning long-term dependencies with gradient descent is difficult."* Neural Networks, IEEE Transactions on 5.2 (1994): 157-166.

[7]. Carlson, David E., Volkan Cevher, and Lawrence Carin. *"Stochastic Spectral Descent for Restricted Boltzmann Machines."* AISTATS. 2015.

[8]. Carlson, David E., et al. *"Preconditioned spectral descent for deep learning."* Advances in Neural Information Processing Systems. 2015.

[9]. Kelner, Jonathan A., et al. *"An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations."* Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, 2014.

[10]. Nesterov, Yu. *"Efficiency of coordinate descent methods on huge-scale optimization problems."* SIAM Journal on Optimization 22.2 (2012): 341-362.

[11]. Carlson, David, et al. *"Stochastic Spectral Descent for Discrete Graphical Models."* (2016).

[12]. Collobert, Ronan, Samy Bengio, and Johnny Marithoz. *Torch: a modular machine learning software library.* No. EPFL-REPORT-82802. IDIAP, 2002.

[13]. Bernstein, Dennis S. *Matrix mathematics: theory, facts, and formulas.* Princeton University Press, 2009.

[14]. Hansen, Per Christian. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion.* Vol. 4. Siam, 1998.

[15]. Dahl, Deborah A., et al. *"Expanding the scope of the ATIS task: The ATIS-3 corpus."* Proceedings of the workshop on Human Language Technology. Association for Computational Linguistics, 1994.

[16]. Mikolov, Tomas, et al. *"Distributed representations of words and phrases and their compositionality."* Advances in neural information processing systems. 2013.

[17]. Wold, Svante, Kim Esbensen, and Paul Geladi. *"Principal component analysis."* Chemometrics and intelligent laboratory systems 2.1-3 (1987): 37-52.

[18]. Bergstra, James, and Yoshua Bengio. *"Random search for hyper-parameter optimization."* The Journal of Machine Learning Research 13.1 (2012): 281-305.

[19]. Zeiler, Matthew D. *"ADADELTA: an adaptive learning rate method."* arXiv preprint arXiv:1212.5701 (2012).