Investigation of Stochastic Gradient Descent in Neural Networks

Hugh Chen
Department of Statistics
University of Washington
Seattle, Washington
`{hughchen}@uw.edu`

Abstract

Neural networks are machine learning techniques that are capable of arbitrarily reducing bias by introducing parameters and non-linearities - which often leads to non-convex optimization. Due to the generalizability and widespread appeal of neural networks, many smart techniques have been developed to optimize neural networks. In this paper, I investigate and specify different variants of stochastic gradient descent for neural networks. Then, I explore these methods experimentally. First, I experiment with methods for visualizing these methods for gradient descent. Second, I analyze these methods applied to neural networks in simple settings. Finally, I look at the utility of ensemble methods which can be applied in addition to these variants of SGD.
Keywords: neural networks, stochastic optimization, ensembles.

1 Background

Stochastic gradient descent (SGD) is a stochastic approximation for minimizing an objective function (generally a loss function) [1]. Defining a function $\mathcal{L}$ over a set of weight parameters w, our general form of objective function is:

$$\mathcal{L}(w; y_1, \cdots, y_n, x_1, \cdots, x_n) = \sum_{i=1}^{n} \mathcal{L}_i(w; y_i, x_i)$$

We denote a prediction using weights w on input x_i as $h(w, x_i)$. For regression, mean squared error is a commonly used loss function where

$$\mathcal{L}_i(w; y_i, x_i) = (y_i - h(w, x_i))^2$$

For binary classification problems, it is common to utilize binary cross-entropy loss functions where

$$\mathcal{L}_i(w; y_i, x_i) = y_i \ln(h(w, x_i)) + (1 - y_i) \ln(1 - h(w, x_i))$$

Other loss functions include exponential cost, Hellinger distance, Kullback-Leibler divergence, generalized Kullback-Leibler divergence, and Itakura-Saito distance [2].

The traditional gradient descent method performs updates according to:

$$w \leftarrow w - \eta \nabla \mathcal{L}(w; y_1, \cdots, y_n, x_1, \cdots, x_n) = w - \eta \sum_{i=1}^{n} \nabla \mathcal{L}_i(w; y_i, x_i)$$

In the above iteration, η defines a learning rate that specifies how fast the parameters update according to the gradient of the loss function. In many cases, evaluating the $\sum_{i=1}^{n} \nabla \mathcal{L}_i(w)$ may be too expensive to compute at every iteration. Instead, **stochastic gradient descent** updates by approximating the true gradient with respect to all possible training examples with a single randomly picked example at a time:

$$w \leftarrow w - \eta \nabla \mathcal{L}_i(w; y_i, x_i)$$

Then, utilizing batches instead of single examples serves as tractable yet more accurate representations of the true gradient. Stochastic gradient descent is useful for a number of machine learning algorithms: Perceptron, Adaline and k-Means, SVM, and Lasso may all be implemented according to this formulation [1]. Due to the layered structure and supervised nature of neural networks, stochastic gradient descent applied via backpropagation is the standard method for optimizing NN weights [3]. Backpropagation is essentially applying the chain rule to tractably compute the derivative given that we know our output values under the setting of neural networks (even in the case of dimensionality reduction for autoencoders).

1.1 Extensions to SGD

Momentum for stochastic gradient descent is analogous to momentum in physics, whereby the previous update informs our current iteration [3]. Momentum has been successful with dealing with curvature issues in objective functions for deep and recurrent neural networks [4]. We define w_t to represent the weight parameters at time t. Then, we define the update step for our randomly chosen sample i:

$$\Delta w_t \leftarrow -\eta \nabla \mathcal{L}_i(w_{t-1}; y_i, x_i) + \alpha \Delta w_{t-1}$$
$$w_t \leftarrow w_{t-1} + \Delta w_t$$

Momentum is a useful extension to SGD in that it prevents oscillations and helps accelerate SGD. As intuition, momentum formalizes that idea that if a particular direction (vector in the parameter space) has been been considered important many times in the recent past, exploring that direction at the current time step is a good idea. Directions d with low curvature will have slower local change in their rate of reduction ($d^T \nabla f$), leading to persistence across iterations and accumulated velocity towards d.

Polyak-Ruppert Averaging is also know as the averaged stochastic gradient descent (ASGD) algorithm was independently invented by Ruppert and Polyak in the 1980s [1, 5]. This variant of SGD performs SGD as normal, but simultaneously records an average of its parameter vector over time. If we assume that we have the sequence of weight vectors obtained by traditional SGD, $w_1, \cdots, w_N$, where N is simply the number of training epochs. Then the recorded parameter average is:

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

AdaGrad stands for adaptive gradient. It is a modified stochastic gradient descent with a per-parameter learning rate [6]. AdaGrad consists of a base learning rate η, multiplied with the elements of a vector $\{G^w_{j,j}\}$ - the diagonal of the outer product matrix:

$$G^w = \sum_{\tau=1}^{t} \nabla \mathcal{L}_i(w_\tau; y_i, x_i)(\nabla \mathcal{L}_i(w_\tau; y_i, x_i))^T$$

Then, the weights are updated as follows:

$$w_t \leftarrow w_{t-1} - \eta \times \text{diag}(G)^{-1/2} \circ \nabla \mathcal{L}_i(w_i; y_i, x_i)$$

Note that $\circ$ denotes an elementwise product. In order for ease of interpretation we may also derive:

$$G^w_{j,j} = \sum_{\tau=1}^{t} ((\nabla \mathcal{L}_i(w_\tau; y_i, x_i))_j)^2$$

$$(w_t)_j \leftarrow (w_{t-1})_j - \frac{\eta}{\sqrt{(G^w_{j,j})}} \times (\nabla \mathcal{L}_i(w_{t-1}; y_i, x_i))_j$$

Here, it is apparent that AdaGrad keeps track of the "history" of the gradient descent by using the ℓ_2 norm of the gradient at each time step for each parameter. Since we utilize the reciprocal of the

ℓ_2 norm, we can see that "sparse" parameters are emphasized - suggesting improved convergence performance in settings where sparse parameters are informative.

Because we simply use the ℓ_2 norm across the gradients at each time step, we can see that one undesirable property of AdaGrad could be that the weights are influenced equally by iteration 1 and iteration $t-1$. Ostensibly, the gradient at time step 1 should have less influence on our current update. To this end, one may use RMSProp instead.

RMSProp stands for Root Mean Square Propagation - similarly to AdaGrad the learning rate is adapted for each of the parameters. At a high level, RMSProp divides the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight parameter. First, the running average is calculated as:

$$\mu(w_t) \leftarrow \gamma\mu(w_{t-1}) + (1-\gamma)(\nabla\mathcal{L}_i(w_{t-1}; y_i, x_i))^2$$

Above, γ represents the forgetfulness factor (a hyperparameter). Then, weights are updated as follows:

$$w_t \leftarrow w_{t-1} - \frac{\eta}{\sqrt{\mu(w_{t-1})}}\nabla\mathcal{L}_i(w_{t-1}; y_i, x_i)$$

RMSProp was developed in response to AdaGrad's radically diminishing learning rates [7]. In comparison to AdaGrad, RMSProp allows us to modify how much the norm of the gradient effects our parameter tuning.

Adam stands for Adaptive Moment Estimation - an update to RMSProp [8]. Adam utilizes averages of both the gradient and the square of the gradient (surrogates for the first and second moments).

$$\mu_t^w \leftarrow \gamma_1\mu_{t-1}^w + (1-\gamma_1)\nabla\mathcal{L}_i(w_{t-1}; y_i, x_i))$$

$$\sigma_t^w \leftarrow \gamma_2\sigma_{t-1}^w + (1-\gamma_2)(\nabla\mathcal{L}_i(w_{t-1}; y_i, x_i)))^2$$

Then, our weight update becomes:

$$w_t \leftarrow w_{t-1} - \eta\frac{\mu_t^w/(1-\gamma_1)}{\sqrt{\sigma_t^w/(1-\gamma_2)} + \epsilon}$$

Where the γ_1, γ_2 are the forgetfulness factors for μ, σ respectively. Finally, ϵ is simply a small offset such that we are never dividing by zero. At the cost of an additional parameter over RMSProp, Adam is capable of incorporating surrogates for the first and second moments of the gradient.

Snapshot Ensembles are a simple variant on Stochastic Gradient Descent. They behave similarly to Polyak-Ruppert Averaging, but average predictions rather than parameters [9]. If we assume that we have the sequence of weight vectors obtained by SGD, $w_1, \cdots, w_N$, where N is simply the number of training epochs. Then predictions on x_i are:

$$\frac{1}{t}\sum_{i=0}^{t-1} h(w_i, x_i)$$

In Huang et. al. (2017) we see that utilizing a cyclic learning rate schedule this ensemble approach is simple, yet surprisingly effective. It yields lower error rates with no additional training cost and compares with traditional network ensembles [9]. Here, I apply no cycle snapshot ensembles for the sake of exploration.

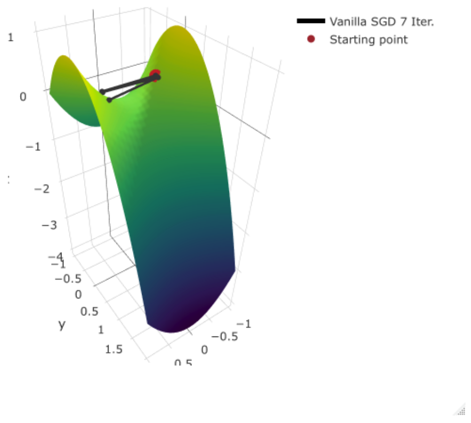
2 Initial Experimental Results

Data We consider the following two data sets:

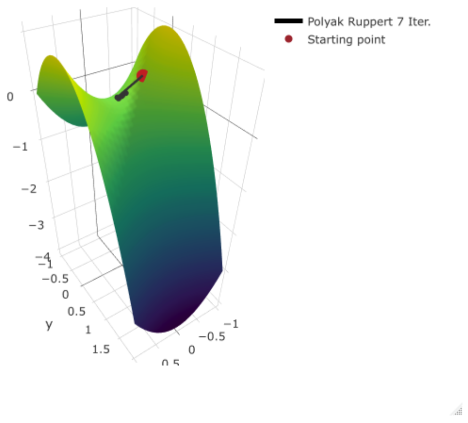
Reuters is a popular data set containing 11,228 newswires from Reuters with 46 topics - text categorization.

CIFAR-10 is a popular data set containing 60,000 32x32 color images with 10 classes - image classification.

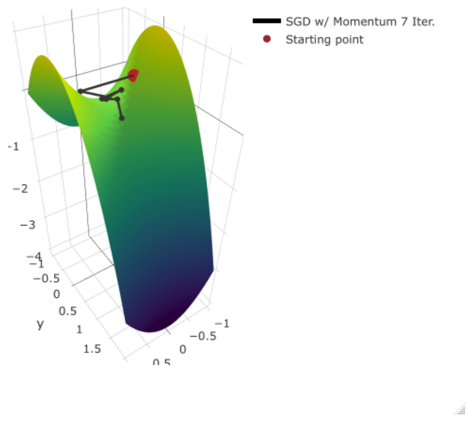
Evaluation Metric *Accuracy* is a useful metric in its straightforward nature and ease of use. In multilabel prediction problems for machine learning, a common approach is to predict the probabilities



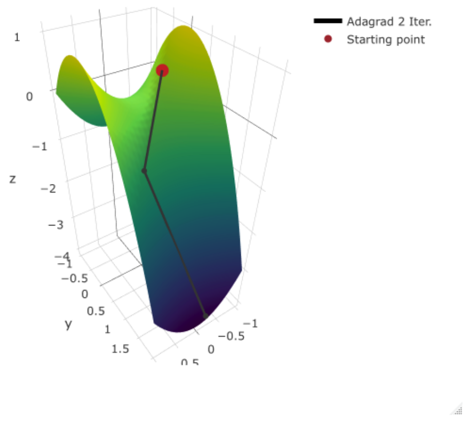
(a) Vanilla SGD



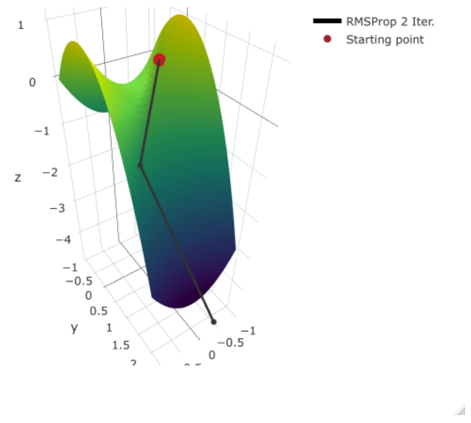
(b) Polyak Averaging



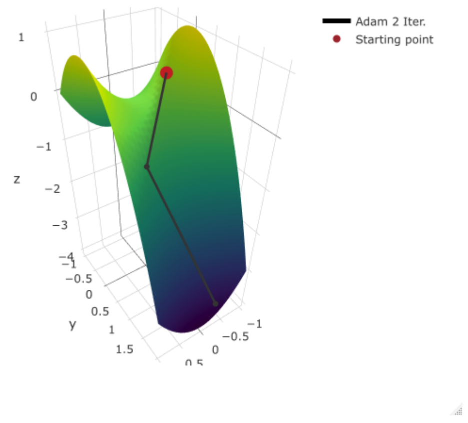
(c) Vanilla SGD with Momentum



(d) Adagrad



(e) RMSProp



(f) Adam

Figure 1: SGD variants applied to $\mathcal{L}(x, y) = x^2 - y^2$ utilizing $\eta = 1$.

from the network with a given sample point and label the point based off the maximally probable class. Then, accuracy is the percentage of labels that match the true labels for a given data set.

Performance In Figure 1, we can see an application of different methods of optimization with stochastic gradient descent. Here we have an example that illustrates how vanilla SGD can run into trouble. Although in this case, if we have sufficient iterations, all methods will travel in the right direction eventually, we can see that with unlucky initializations we can end up bouncing back and forth in this ridge. We can see that Polyak Averaging avoids this bouncing, but does not improve the direction that the gradient descent takes. Then, with momentum we can see that after seven iterations, gradient descent is heading in the correct direction. In (d), (e), and (f) we can see that Adagrad, RMSProp, and Adam all immediately avoid this problem. Note that I don't tune the hyperparameters in any of these settings - only adjusting for the number of iterations to make the graphs fit.

Table 1: LSTM network Performance on Hypoxemia

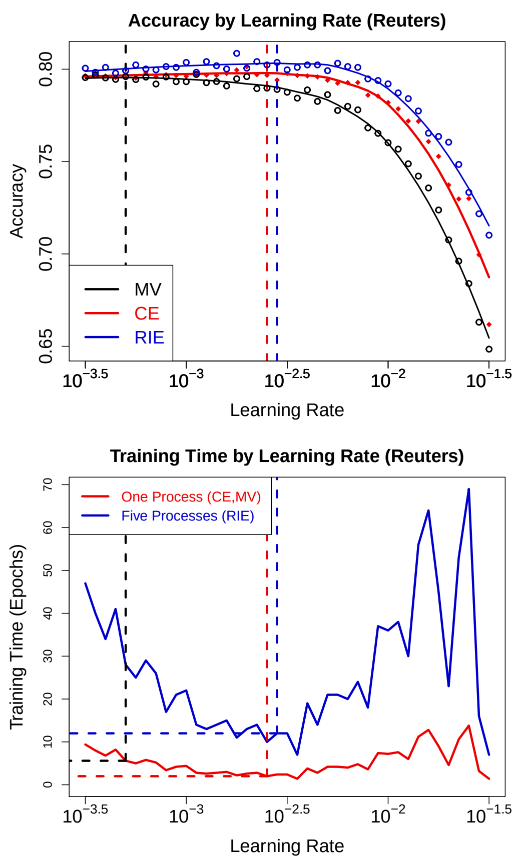
Model	Max Accuracy	Epochs at Max	Max Learning Rate
Vanilla SGD	0.78807	60	10^{-2}
Vanilla SGD with Momentum	0.79074	35	$10^{-1.5}$
Polyak Averaging on Vanilla SGD	0.76714	28	$10^{-1.5}$
Snapshot Ensemble on Vanilla SGD	0.78050	28	$10^{-1.5}$
Adagrad	0.80142	4	$10^{-1.8}$
RMSProp	0.79653	2	$10^{-2.6}$
Adam	0.80009	2	$10^{-2.7}$

Applying a variety of Stochastic Gradient Descent variants on the Reuters data set with a non-convex neural network.

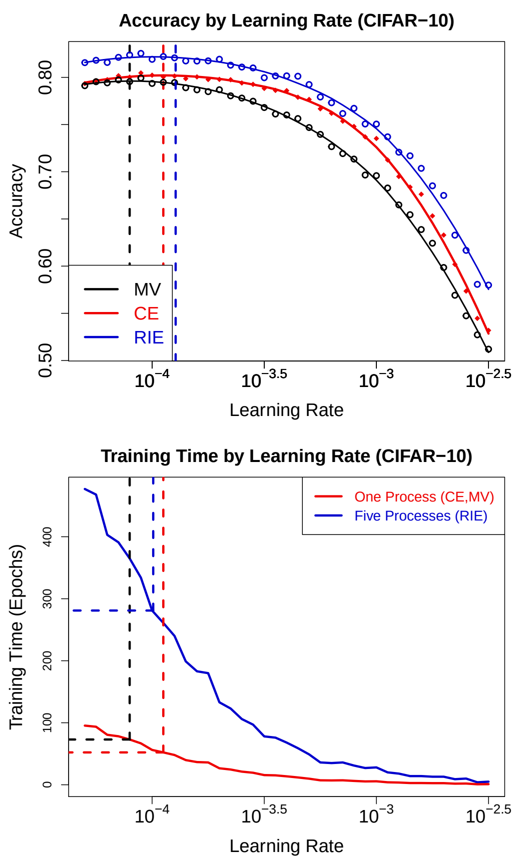
Now, if we take a look at table 1 we can see similar results. For this table, I train a feedforward network structured as follows: 1,000 input nodes, 512 hidden node layer, ReLU activation, 0.5 dropout, 46 node layer, and Softmax. I train until there was no improvement in validation score after ten epochs. I generate five models that vary only across each learning rate in the set $10^{-1.5}, 10^{-1.55}, 10^{-1.60}, \cdots, 10^{-2.8}$ and evaluate the performances. Then, we can see that utilizing Adagrad, RMSProp, and Adam all work very well. Additionally, snapshot ensembles don't perform very well. However, observe that snapshot ensembles can be applied regardless of the methodology for gradient descent.

In the final section, utilizing snapshot ensembles with Adam for gradient descent shows interesting results. I averaged the accuracy and number of epochs for the five runs for two prediction methods. 1.) choosing a final prediction model by evaluating accuracy in a validation set (which I denote as MV for minimum validation) and 3.) snapshot ensembles (which I denote CE for Figure 2a). Finally, RIE denotes traditional ensembles (random initialization ensembles). First, notice that CE consistently outperforms selecting by the MV in terms of accuracy, and appears to capture some portion of the benefit RIE affords. Secondly, we note that CE appears to converge at a higher learning rate than MV does.

Now, looking at the epochs in Figure 2a, it is immediately obvious there is high variance in the epochs to the right of learning rate $10^{-2.5}$, because excessively high learning rates result in unreliable convergence. At lower learning rates ($[10^{-3.5}, 10^{-2.5}]$), we see that the number of epochs to convergence grows as the learning rate decreases since the network moves through the space of parameters slowly. Additionally, we observe that the maximum point for CE translates to approximately two epochs of training time whereas the maximum point for MV translates to five epochs of training time. In this case, one should generally prefer CE because it requires less running time and achieves better performance than MV. Comparing to RIE, we see that RIE takes about twelve epochs to converge under it's optimal learning rate compared to CE's two epochs. This gain in convergence speed is less significant for the Reuters data set because it is small and straightforward; however for larger training data sets, a single epoch can easily take hours or days. In these settings, CE might be a better choice of ensemble. Finally, we observe that the optimal learning rate is very similar between CE and RIE. Since the optimal learning rate between MV and RIE is quite different, another practical use for CE could be to tune the optimal learning rate for RIE. As a final investigation, figure 2b shows similar results utilizing convolutional neural networks.



(a) Reuters



(b) CIFAR-10

Figure 2: Utilizing Adam and different methods for prediction. CE refers to selecting multiple models for snapshot ensembles and MV refers to selecting a model by minimum validation.

References

[1] Léon Bottou. *Large-Scale Machine Learning with Stochastic Gradient Descent*, pages 177–186. Physica-Verlag HD, Heidelberg, 2010.

[2] A list of cost functions used in neural networks, alongside applications. `goo.gl/mRhSBp`.

[3] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*, pages 533–536. Nature, 1986.

[4] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[5] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855, July 1992.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.

[7] Sebastian Ruder. An overview of gradient descent optimization algorithms. `http://ruder.io/optimizing-gradient-descent/index.html#rmsprop`.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[9] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get M for free. *CoRR*, abs/1704.00109, 2017.