Current Topics in Artificial Intelligence: Optimization

Chenxi Liu
Department of Statistics
University of California, Los Angeles
cxliu@ucla.edu

Abstract

This short survey discusses the role of optimization in current deep learning research. Nonlinearity is one of the reasons why deep neural networks are so powerful. Without nonlinearity, deep nets cannot even separate XOR. But at the same time, this nonlinearity and hierarchy usually makes the loss functions nonconvex and highly intractable. This poses at least two questions: 1. What are the appropriate optimization methods for deep neural networks? 2. What are the interesting properties of the loss surface of deep neural networks? In this survey, we start by introducing the general setting of optimization problems (Section 1). Then we focus on popular choices in deep learning, both for parameters (Section 2) and hyper-parameters (Section 3). We end with exciting recent progress in understanding the loss surface of deep neural networks (Section 4). This is the third of the four short surveys.

1 Generic Approach

The purpose of learning is to adjust the weights in the model so as to minimize the loss function. In reality, we usually allow the weights to take on any value in the weight space, so it is an unconstrained minimization problem [1].

$$\text{minimize} \quad l(\theta) \tag{1}$$

To solve the problem, the standard approach is to produce a sequence of points $\theta^{(k)} \in \mathbf{dom}\, l, k = 0, 1, ...$ with

$$l(\theta^{(k)}) \to p^* \tag{2}$$

which is an iterative method for solving optimality condition

$$\nabla l(\theta^*) = 0 \tag{3}$$

If the loss function is convex, then solving the optimality condition is equivalent to finding the global minimum. If the problem is nonconvex, we can apply the same method anyway, but can only converge to local minimum. It is possible to use simulated annealing [2] and find global minimum, but the stochastic nature makes it much slower than other deterministic methods, and thus not commonly used. Additionally, we are also going to see in Section 4 that for deep models, all local minimums lie in an arguably small energy band near global minimum.

1.1 First Order Method

In first order methods, we only consider the first order approximation of the loss surface at current point θ. The most common form is gradient descent, where the direction of update is the same as the steepest descent direction.

$$\Delta\theta = -\nabla l(\theta) \tag{4}$$

According to [1], despite its simplicity, gradient descent is often very slow and rarely used in practice. But we are going to see in Section 3 that first order method is the most popular choice in deep learning.

1.2 Second Order Method

Here we consider the local approximation of the loss surface up to the second order. In Newton's method, the direction of update is the inverse of Hessian times the gradient.

$$\Delta\theta = -\nabla^2 l(\theta)^{-1} \nabla l(\theta) \tag{5}$$

The advantage of Newton's method is that it is affine invariant, i.e. independent of linear changes of coordinates. One potential drawback is that inverting a Hessian is computationally expensive, especially when the dimensionality is high.

2 Tuning Parameters

The most popular choice in deep neural network optimization is stochastic gradient descent (SGD).

In each training iteration of SGD, instead of using the whole training set, we use only one or a small set of random training samples to produce the gradient. The idea may seem unreliable, but consider the case where the dataset is highly redundant. If the first half of the dataset and the second half are very similar, then training on either half gives the same result as training on the whole dataset. So it makes sense to train on a smaller portion of the dataset, which speeds up training at (hopefully) small sacrifice of convergence.

According to [3], stochastic gradient descent means training on *one* random new example, and training on *several* random new examples is called mini-batch gradient descent. But nowadays when people are in fact using mini-batches, they also call it SGD.

Choosing a first order method should not be surprising, because the efficient backpropagation technique [4] only propagates the gradient but not the Hessian. It is possible to extend backpropagation and include Hessian, but the efficient methods for evaluating the Hessian is still $O(n^2)$ [5], where n is the number of parameters in the model. Since n can easily be in the order of millions in a neural network, this is a price very hard to afford.

Several ways to speed up SGD learning are introduced below.

2.1 Momentum

The intuition behind momentum [6] is that we not only consider the gradient at the current position, but also the gradient at the last position:

$$\Delta\theta_t = \alpha\Delta\theta_{t-1} - \epsilon\nabla l(\theta_t) \tag{6}$$

Trusting the previous gradient with weight α is equivalent to a exponential decay smoothing, which helps alleviate the noise introduced by the mini-batches.

2.2 Separate adaptive learning rate

The intuition is that the appropriate learning rates should vary between individual weights. So we allow the learning rates of individual weights to diverge through training:

$$\Delta\theta_{ij} = -\epsilon g_{ij} \frac{\partial l(\theta)}{\partial \theta_{ij}} \tag{7}$$

If the gradient for that weight does not change sign, $g_{ij}(t) = g_{ij}(t-1) + 0.05$. Otherwise, $g_{ij}(t) = 0.95 g_{ij}(t-1)$. It is important to limit the gains to lie in some reasonable range, e.g. [0.1, 10] or [0.01, 100].

2.3 Rprop and RMSprop

Rprop means in each iteration, we only use the sign of the gradient, but not the magnitude. So it is equivalent to using the gradient but also dividing by the size of the gradient.

In RMSprop [3], we divide the gradient by $\sqrt{MS(t)}$ instead, where

$$MS(t) = 0.9 MS(t-1) + 0.1 \nabla l(\theta_t)^2 \tag{8}$$

2.4 Adagrad

The basic version of Adagrad [7] gives the following update rule:

$$\theta_{t+1} = \theta_t - \alpha \frac{\Delta\theta_t}{\sum_{i=1}^{t} \Delta\theta_i} \tag{9}$$

2.5 Adam

Adam [8] is a recently proposed algorithm that integrates the aforementioned momentum, RMSprop, and Adagrad. The hyper-parameters have intuitive interpretations and typically require little tuning.

Algorithm 1: Adam algorithm. Good default settings for the tested machine learning problems are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$.

while θ_t *not converged* **do**
- $t \leftarrow t + 1$
- $g_t \leftarrow \nabla l(\theta_{t-1})$
- $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
- $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
- $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
- $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
- $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

end

3 Tuning Hyper-parameters

Although the number of *parameters* in a deep model is usually huge, the processing of learning those parameters is at least automatic. The step that is most painful is actually designing the model itself, i.e. choosing *hyper-parameters*.

Choosing good hyper-parameters is regarded essential for optimization method to work, but requires a lot of skill. One approach is naive grid search, but when the number of hyper-parameters increases

this soon becomes impractical. The other approach is sampling random combinations, which works better when some hyper-parameters have no effect [9].

Here we consider whether it is possible to replace the graduate students who spend days tuning hyper-parameters and automate this process.

3.1 Bayesian Optimization

Bayesian optimization [10] selects hyper-parameters by looking at the results so far, and predict regions of the hyper-parameter space that might give better results. Specifically, Gaussian Process models are used. For each input dimension, they learn the appropriate scale for measuring similarity, and output a Gaussian distribution of values instead of just a single value.

During Bayesian optimization, we keep track of the best setting so far, and pick a setting of the hyper-parameters such that the *expected improvement* in this best setting is big. This means we don't need to worry about the downside (because if they perform worse we just keep the current setting) and just look at the upside.

3.2 Best Arm Identification

More recently, [11] proposed an alternative way to do hyper-parameter optimization. It views different hyper-parameter settings as different "arms", and try to find the best "arm" within a fixed budget. More specifically, the algorithm uniformly allocates the budget to a set of arms for a fixed number of iterations, evaluate the performance, discard the worst half, and repeat the process until only one arm remains.

The advantage over Bayesian optimization is that instead of being inherently sequential, best arm identification is embarrassingly parallel. So it suits parallel training of very large models very well.

4 Loss Surface

When training a neural network multiple times, despite the fact that both the initialization and the selection of mini-batch training samples are random, the models usually deliver very similar performance. This phenomenon suggests that the loss surface of deep neural networks has some interesting properties, and indeed in the past few years this has been a central topic in deep learning theory.

4.1 Saddle Points

For a differentiable function of several real variables, a critical point is a value in its domain where all partial derivatives are zero. A critical point may either be a local maximum, a local minimum, or a saddle point, depending on the eigenvalues of the Hessian. [12] [13] discussed the proliferation of saddle points in high dimensions in the context of deep neural networks.

Empirically, they showed that there exist many critical points in a band close to the smallest train error. This is important because both first order and second order methods, as discussed in Section 1, have limitations in this setting. First order methods, like SGD, may make small steps in directions corresponding to eigenvalues of small absolute value, while second order methods, like Newton's method, moves in the opposite direction to the gradient descent if the corresponding eigenvalue is negative. Therefore Saddle-free Newton method (SFN) was proposed, which was simply replacing the Hessian H in Newton's method by $|H|$, which means taking the absolute value of the eigenvalues of H.

There are some obvious questions to this approach:

1. After the replacement the optimization problem is no longer the same. How close is the new problem to the original problem is open to question.
2. SFN is still a second order method which does not appear very often in DNNs because of computational cost. In fact this may be the reason the empricial experiments were done on down-sampled datasets and small MLP.

This motivates a combination, meaning to use SGD first, and after it converges we switch to SFN and dive further in the loss surface.

4.2 Spin Glass

The recent progress in Physics on spin glasses [14] motivates an alternative explanation of neural networks in terms of weight connections [15] [16].

A p-spin glass can be written as

$$-H_p(\sigma) = \sum_{1 \leq i_1, \ldots, i_p \leq n} J_{i_1, \ldots, i_p} \sigma_{i_1} \sigma_{i_2} \ldots \sigma_{i_p} \tag{10}$$

In a graphical model setting, this is an undirected graph with n cliques of size p.

Note that a deep network (with p layers and n neurons in each layer) normally expressed as

$$Y = W_p^T \sigma(W_{p-1} \sigma(W_{p-2} \ldots \sigma(W_1 X)) \ldots)) \tag{11}$$

can be re-written as

$$Y = \sum_{i=1}^{n} \sum_{\gamma \in \Gamma_i} X_i (\prod_{k=1}^{p} w_\gamma^k) \tag{12}$$

where the first summation is over all dimensions of X, second summation is over all "active" paths in the network, and the term in the parentheses is the product along path γ. With some further assumptions, we can show that the loss function of this network is in exactly the same form as the p-spin glass.

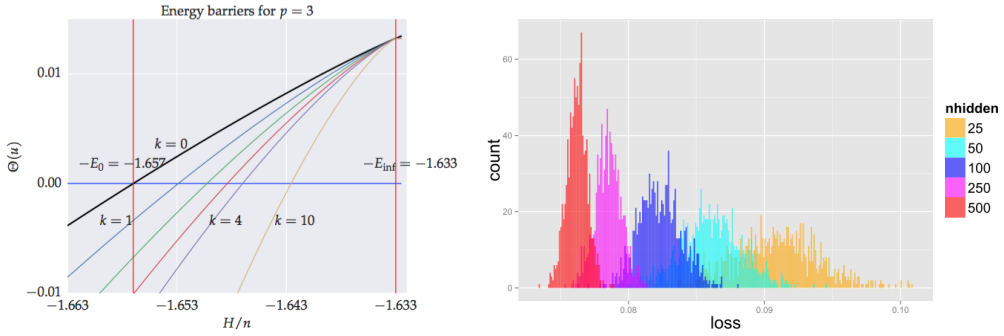

Figure 1: Left: Energy barrier for spin glass model; Right: Experiments on MNIST

Due to space constraints we cannot go into too much detail, but the most interesting conclusion may be: the value H/n forms a layered structure with respect to the index of the critical point. More specifically, local minimums live in the lowest energy band, critical points of index 1 live in an energy band above that, and the same goes for critical points of index 2, 3, ...

However we are still far away understanding deep neural networks fully. Firstly, the assumptions that we made along the way do not generally hold in practice (e.g. fixed weight range, sparse connection). Secondly, the conclusion was drawn for H/n but not H. So when n is large, $[-E_0, -E_\infty]$ may be small, but $[-nE_0, -nE_\infty]$ may not.

References

[1] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[2] S. Kirkpatrick, M. P. Vecchi, *et al.*, "Optimization by simmulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[3] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning," *Coursera, video lectures*, vol. 264, 2012.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[5] C. M. Bishop, "Pattern recognition," *Machine Learning*, 2006.

[6] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 1139–1147, 2013.

[7] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

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

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

[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[11] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," *arXiv preprint arXiv:1502.07943*, 2015.

[12] R. Pascanu, Y. N. Dauphin, S. Ganguli, and Y. Bengio, "On the saddle point problem for non-convex optimization," *arXiv preprint arXiv:1405.4604*, 2014.

[13] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in neural information processing systems*, pp. 2933–2941, 2014.

[14] A. Auffinger, G. B. Arous, and J. Černỳ, "Random matrices and complexity of spin glasses," *Communications on Pure and Applied Mathematics*, vol. 66, no. 2, pp. 165–201, 2013.

[15] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," *arXiv preprint arXiv:1412.0233*, 2014.

[16] P. Chaudhari and S. Soatto, "Trivializing the energy landscape of deep networks," *arXiv preprint arXiv:1511.06485*, 2015.