Two Approaches to Non-convex Machine Learning Scribe: Kiran Vodrahalli
Yuchen Zhang, Stanford Columbia Theory Talk: 09/22/17

1 Introduction

Non-convexity is becoming a important point in machine learning, lots of nonconvex loss functions minimizing wise. Applied in lots of problems. Guarantees for nonconvex is only locally optimal, we claim that this is not good enough. In this talk, we will explore two ideas for attacking non-convexity and local minima.

(a) Idea 1: Inject large random noise into SGD.

(b) Idea 2: Convex relaxation. Challenge: perserve most important features in non-convex model.

2 Injecting Noise into SGD

Let's consider minimizing a function $\min_{x \in K} f(x)$. Gradient descent converges to different solutions with different initialization. Then you can end up stuck at a local optima. How do you escape/avoid bad local minima? Let's consider noisy gradient descent. Every iteration of the update, sample a Gaussian noise vector. Random perturbation allows us to escape certain local minima. However, you have to pay special attention to choice of step size. If it's too small, then noise will cancel itself out quickly before escaping the local min. If you use too large a step size, then the algorithm may overshoot and become unable to converge to anything. So how do you achieve stability with the power to escape?

One answer is to use Langevin Monte Carlo algorithm (LMC). The update:

$$x_t = x_{t-1} - \eta(\nabla f(x) - \sqrt{1/\eta}\sqrt{2T}w)$$

where η is step size, hyperparameter T (temperature). w is Gaussian. This relates to physical diffusion process, discretized. The scaling on the noise connects to the step size. Noise dominates gradient when $\eta \to 0$. LMC escapes local minima with small step sizes. If you replace exact gradient with stochastic gradient, you get SGLD (stochastic gradient Langevin dynamics). Can be more efficient than LMC. SGLD is more general and potentially more computationally efficient.

Why use $\sqrt{1/\eta}$ to scale noise? Only by choosing this specific function is the algorithm able to converge to meaningful stationary distribution: $\mu(x) \sim e^{-f(x)/T}$, Gibbs distribution. If temperature is low, then x minimizes f (with high concentration on distribution of x). Thus as long as you run algorithm for sufficiently long time, you will get a solution almost as good as global minimum.

There's a tradeoff: if you set T close to 0, you'll get global minimum. But the time you need to converge might be very long, exponential dependence on $1/T$.

There are many successful application stories for this algorithm: ICA and logistic regression, the method helps improve generalization (learn distribution instead of single point). For learning neural programmers, neural GPU, deep bidirectional LSTM – it also helps. The algorithm is more robust in the bad stationary points.

2.1 How fast does it converge?

Only recently have people started to study behavior of Langevin algorithms non-asymptotically. The LMC mixing time is polynomial (time to converge to stationary distribution) on convex functions. But we already know convex functions are easy to optimize. We'd like to know about general nonconvex functions. Generally exponential dependence on d and $1/T$. This is average case exponential dependence.

Is mixing time too specific? We only care if we get a good solution in a short amount of time. In practice, SGLD hits a good enough solution far before it converges to the stationary distribution. Consider a W-function: gets the first good solution quickly, mixing time is super long because it takes a while to escape the first minimum to hit the second one (needs to put equal probability on that solution too).

Our analysis focuses on SGLD's hitting time to an arbitrary target set. There's a polynomial upper bound on hitting time, we can apply this to non-convex optimization.

2.2 Preliminaries

For any $f : K \to \mathbb{R}$, define measure

$$\mu_f(x) := \frac{e^{-f(x)}}{\int_K e^{-f(x)}dx} \sim e^{-f(x)}$$

Secondly, define **boundary measure** w.r.t probability measure μ

$$\mu_f(\partial A) := \lim_{\epsilon \to 0} \frac{\mu_f(\text{shell of } A)}{\epsilon}$$

This is a generalization of surface area. If f is constant, this is standard surface area.

Finally, we discuss the restricted Cheeger constant for set $V \subset K$:

$$C_f(V) := \inf_{A \subset V} \frac{\mu_f(\partial A)}{\mu_f(A)}$$

This is a ratio between surface area and volume. Intuitively, $C_f(V)$ is small iff we can find some $A \subset V$ which is isolated from the rest of the parameter space.

We claim that $C_{f/T}(V)$ measures the efficiency of SGLD defined on f, T to escape the set V.

It is easy to understand intuitively: If it's small, it'll take a long time to escape the subset since the subset is not "well-connected" (think random walks on graphs). If the constant is large, no matter where you initialize, it'll be able to escape since the space is "well-connected", and achieve local minimum.

Definition 2.1. Stability property.
If two functions are pointwise close, then their restricted Cheeger constants are close.

If $\sup_{x \in K} |f(x) - F(x)| \leq T$, then $\max\{\frac{C_{f/T}(V)}{C_{F/T}(V)}, \frac{C_{f/T}(V)}{C_{F/T}(V)}\} = O(1)$.

Theorem 2.2. *For arbitrary f and target set $U \subset K$, SGLD's hitting time to U satisfies w.h.p.*

$$\text{hitting time} \leq \frac{poly(d)}{(C_{f/T}(K \setminus U))^2}$$

Thus, we must lower bound restricted Cheeger constant to upper bound hitting time. Thus we only need study geometric properties of f, U. This is much easier than studying SGLD trajectory. The lower bound says that if you choose U as an ϵ-approx local min, and $T \leq O(\epsilon^2/\text{poly(smoothness of } f))$, it's lower bounded. Then if you run SGLD on f, for proxy $F \approx f$, if F is smooth, then $\|f - F\|_\infty = O(\epsilon^2/\text{poly(smoothness of } f))$. We can think of F is a perturbation of f which lets us escape local mins. For ERM, F is population risk and f is empirical risk. Under mild conditions, we get $\|f - F\|_\infty$ as $n \to \infty$. For large enough n, SGLD finds local min of population risk. This doesn't need $\|\nabla f - \nabla F\|_\infty \to 0$ and $\|\nabla^2 f - \nabla^2 F\|_\infty \to 0$, like SGD does. For instance for neural nets, if output is bounded, things are fine - don't need to worry about exploding and vanishing gradients which might be an issue for bounding gradient or Hessian norm.

2.3 Summary

So, SGLD is asymptotically optimal for non-convex optimization. However, we don't understand mixing time behavior. This leads us to consider hitting time, which depends on restricted Cheeger constant. Hitting time is polynomial by lower bounding the restricted Cheeger constant. SGLD more robust than SGD for ERM.

3 Convexified CNN

CNN might be interesting anchor point to understand interplay between convex and non-convex optimization. Maybe we can design better model as well. CNN has interesting features: uses convolutional filters to extract small patches from image; then stack these. CNN uses parameter sharing scheme which results in generalization performance than fully connected neural net. CNN requires nonconvex optimization.

SGLD doesn't guarantee globally optimal solution. What if we want to explicitly compute globally optimal solution? There are other variants, but none are as good as classical CNN in terms of classification accuracy. So how do we design an easy-to-compute, efficient algorithm to achieve same or better performance than classical CNN? A convolutional layer applies non-linear filters to a sliding window of patches. Every filter is a linear transformation followed by nonlinear activation function. You stack the layers to make a CNN. Ontop of the last layer we have a linear mapping to define an output. I.e. for a two-layer net,

$$f_k(x) := \sum_{j=1}^{r} \sum_{p=1}^{q} \alpha_{k,j,p} \sigma(w_j^T z_p(x))$$

where w_j are filter parameters and α is output parameters, and x is the image. The CNN loss is non-convex because of the nonlinear activation function, and parameter sharing across the w_j. Can we re-design the CNN class to be convex while retaining non-linear filters and parameter sharing?

Consider a linear CNN: $\sum_{j=1}^{r}\sum_{p=1}^{q}\alpha_{k,j,p}w_j^T z_p(x)$. A more compact form is as follows. Consider design matrix $Z(x) \in \mathbb{R}^{q\times d}$ encodes the patches. The filter matrix $W \in \mathbb{R}^{d\times r}$ is the weights. Then, there is the output matrix $A_k \in \mathbb{R}^{r\times p}$, where k means the k^{th} output. Thus, $f_k(x) = \mathrm{Tr}(Z(x)WA_k)$. We can write $WA_k = \Theta_k$. Thus defining $\Theta = [\Theta_1, \cdots, \Theta_k]$. We also know $\mathrm{rank}(\Theta) \leq r$ since $\mathrm{rank}(W) \leq r$. This is equivalent to model class defined by 2-layer CNN. Thus, we have $f(x) = [\mathrm{Tr}(Z(x)\Theta_1), \cdots, \mathrm{Tr}(Z(x)\Theta_k)]$. Learning Θ is non-convex. But we can do nuclear-norm minimization: Relax $\mathrm{rank}(\Theta) \leq r$ to $\|\Theta\|_* \leq Cr$.

Now after we know how to convexify linear CNN, how do we convexify nonlinear two layer CNN? We can parametrize the $h_j(z) = \sigma(w_i^T z)$ using an RKHS. We have $\phi(z)$ is our non-linear mapping; $k(z', z'') = \langle\phi(z'), \phi(z'')\rangle$. So now we have a linear filter in RKHS. Now, how do we define feature map? To make the model generalize well, we only need to use approximate feature map so that it holds for the training set. Then we re-define the patches according to this map, and we have reduced it to linear CNN case.

Why can we always re-parametrize the filters in the RKHS? What additional assumption do we need so the filters belong to RKHS? Well, if σ is smooth, then the filters h_j will also be smooth. It is well known that if we properly choose k, the RKHS will contain all possible sufficiently smooth functions. One sufficient assumption for re-parametrizing is assume smoothness on activation function. But for actual training, we never need to choose the activation function, only need to choose the kernel. Assumption of smoothness on σ is only for theoretical comparison to standard CNN, optimality guarantee. We can thus get an optimality guarantee for learning 2-layer CNNs. So it's tractable, optimal, and converges in $O(1/\sqrt{n})$ rate. By parameter sharing, the CNN filters share the same set of filters. For fully connected neural net, every patch uses individual filter. So the number of parameters which need to be learned is q times more than CNN. Thus number of samples should also be p times more. Thus, the benefit of parameter sharing is preserved in convexified class.

What about multi-layer CNN? We propose a greedy layer-wise training approach: First use algorithm for two-layer neural net. Then we factorize this matrix into filter and output matrix. This is reasonable based on our assumption that $\Theta = WA_k$. We use SVD to do this. Then we extract RKHS filters. Then we repeat to learn parameters for the next layer, and the next layer, and so on. There are no theoretical optimality guarantees comparing to the multi-layer CNN.

3.1 Summary

We have convexified non-linear activations and parameter sharing. This results from a combination of two ideas: 1) CNN filters $\rightarrow$ RKHS filters, 2) parameter sharing $\rightarrow$ nuclear norm constraint. For deeper CNN, convexification technique applies: learn a sequence of convex optimization problems. Empirically it does better (for two, three-layer CNN on

MNIST).

However, they can't get as good performance on CIFAR-10: two issues, max-pooling, deeper layers. They need to be convexified. Greedy layerwise training doesn't appear to be enough for much deeper layers. Speed isn't the issue, random features etc. are enough to have the nuclear norm minimization be fast enough. Speed is comparable to training CNN in Tensorflow.

4 Conclusion

We don't always need to solve non-convex optimization: optimization $\rightarrow$ diffusion process (SGD $\rightarrow$ SGLD), second, proper learning $\rightarrow$ improper learning (CNN $\rightarrow$ CCNN (convex CNN)). High-level open question: Is there a better abstraction for machine learning beyond convex and non-convex optimization? Can we find efficient to compute, analyzable, and powerful enough?

5 Takeaways

6 My Questions