More Non-Convexity, Adaptive Learning Rates, and Algorithms other than SGD

CS6787 Lecture 8 — Fall 2018

Adaptive learning rates

- So far, we've looked at update steps that look like

$$w_{t+1} = w_t - \alpha_t \nabla f_t(w_t)$$

- Here, the learning rate/step size is **fixed a priori** for each iteration.

- What if we use a **step size that varies depending on the model**?

- This is the idea of an **adaptive learning rate**.

Example: Polyak's step length

- This is an simple step size scheme for gradient descent that works when the optimal value is known.

$$\alpha_k = \frac{f(w_k) - f(w^*)}{\|\nabla f(w_k)\|^2}$$

- Can also use this with an estimated optimal value.

Example: Line search

- Idea: just choose the step size that minimizes the objective.

$$\alpha_k = \arg\min_{\alpha > 0} f(w_k - \alpha \nabla f(w_k))$$

- Only works well for gradient descent, not SGD.

- Why?
  - SGD moves in random directions that don't always improve the objective
  - Doing line search is expensive relative to SGD update.

Adaptive methods for SGD

- Several methods exist
  - AdaGrad
  - AdaDelta
  - RMSProp
  - Adam

- You'll see two of these in this Wednesday's papers

One Non-Convex Case Where We Can Show Global Convergence Using Adaptive Learning Rates: PCA

Principal Component Analysis

- Setting: find the dominant **eigenvalue-eigenvector pair** of a positive semidefinite symmetric matrix **A**.

$$u_1 = \arg\max_x \frac{x^T A x}{x^T x} \qquad \lambda_1 = \frac{u_1^T A u_1}{u_1^T u_1}$$

- Many ways to write this problem, e.g.

$\|B\|_F$ is *Frobenius norm*

$$\|B\|_F^2 = \sum_i \sum_j B_{i,j}^2$$

$$\sqrt{\lambda_1} u_1 = \arg\min_x \|x x^T - A\|_F^2$$

Recall: PCA is Non-Convex

- PCA is **not convex** in any of its formulations

- **Why?** Think about the solutions to the problem: **u** and **–u**
  - Two distinct solutions → can't be convex

- But it turns out that we can still show that with appropriately chosen step sizes, **gradient descent converges globally**!
  - This is one of the easiest non-convex problems, and a good place to start to understand how a method works on non-convex problems.

Gradient Descent for PCA

- Gradient of the objective is

$$f(x) = \frac{1}{4}\|xx^T - A\|_F^2, \nabla f(x) = (xx^T - A)x$$

- Gradient descent update step

$$x_{t+1} = x_t - \alpha_t \left(x_t x_t^T x_t - A x_t \right)$$

Gradient Descent for PCA (continued)

- Choose **adaptive** step size for parameter **η**: $\alpha_t = \dfrac{\eta}{1+\eta x_t^T x_t}$

- Then we get:

$$\begin{aligned}
x_{t+1} &= \left(1 - \frac{\eta}{1+\eta x_t^T x_t} x_t^T x_t\right) x_t + \frac{\eta}{1+\eta x_t^T x_t} A x_t \\
&= \frac{1}{1+\eta x_t^T x_t} \left(\left((1+\eta x_t^T x_t) - \eta x_t^T x_t\right) x_t + \eta A x_t\right) \\
&= \frac{1}{1+\eta x_t^T x_t} \left(x_t + \eta A x_t\right)
\end{aligned}$$

Gradient Descent for PCA (continued)

- So we're left with

$$x_{t+1} = \frac{1}{1 + \eta x_t^T x_t}(I + \eta A)x_t$$

- And applying this inductively gives us

$$x_T = (I + \eta A)^T x_0 \prod_{t=0}^{T-1} \frac{1}{1 + \eta \|x_t\|^2}$$

Convergence in Direction

- It should be clear that the **direction of the iterates converges**
  - This is the same expression as we get for power iteration!

$$
\begin{aligned}
\frac{x_K}{\|x_K\|} &= \frac{(I+\eta A)^K x_0}{\|(I+\eta A)^K x_0\|} \\
&= \frac{1}{\|(I+\eta A)^K x_0\|} \cdot \left(I + \eta \sum_{i=1}^{n} \lambda_i u_i u_i^T \right)^K x_0 \\
&= \frac{1}{\|(I+\eta A)^K x_0\|} \cdot \sum_{i=1}^{n} (1+\eta\lambda_i)^K u_i u_i^T x_0 \\
&= \frac{\sum_{i=1}^{n} (1+\eta\lambda_i)^K u_i u_i^T x_0}{\| \sum_{i=1}^{n} (1+\eta\lambda_i)^K u_i u_i^T x_0 \|}
\end{aligned}
$$

Convergence in Direction (continued)

- If we look at just one eigendirection:

$$\frac{(u_j^T x_K)^2}{\|x_K\|^2} = \frac{\left(u_j^T \sum_{i=1}^n (1+\eta\lambda_i)^K u_i u_i^T x_0\right)^2}{\|\sum_{i=1}^n (1+\eta\lambda_i)^K u_i u_i^T x_0\|^2}$$

$$= \frac{(1+\eta\lambda_j)^{2K}\left(u_j^T x_0\right)^2}{\sum_{i=1}^n (1+\eta\lambda_i)^{2K}\left(u_i^T x_0\right)^2}$$

$$\leq \frac{(1+\eta\lambda_j)^{2K}\left(u_j^T x_0\right)^2}{(1+\eta\lambda_1)^{2K}\left(u_1^T x_0\right)^2}$$

$$= \left(\frac{1+\eta\lambda_j}{1+\eta\lambda_1}\right)^{2K} \cdot \left(\frac{u_j^T x_0}{u_1^T x_0}\right)^2$$

This is going to zero at a linear rate unless j = 1.

Convergence in Magnitude

- Imagine we've already converged in direction. Then our update becomes

$$x_{t+1} = \frac{1}{1 + \eta\|x_t\|^2}(1 + \eta\lambda_1)x_t$$

- **Why does this converge?**
  - If **x** is large, it will become small in a single step
  - If **x** is small, it will increase slowly by a factor of about $(1 + \eta\lambda_1)$ until it converges to the optimal value.

Why did this work?

- The PCA objective has **no non-optimal local minima**
  - This means **finding a local optimum is as good as solving the problem**

$$f(x) = \frac{1}{4}\|xx^T - A\|_F^2, \; \nabla f(x) = (xx^T - A)x$$

- We **took advantage of the algebraic properties**
  - And the fact that we already knew about power iteration
  - We used this to choose an adaptive step size seemingly out of nowhere

Stochastic Gradient Descent for PCA

- We can use the same logic to show that a variant of SGD with the same adaptive step sizes works for PCA
  - And we can give an **explicit convergence rate**

- The proof is **long and involved**

- With more work, can even show that **variants with momentum and variance reduction also work**
  - Means we can use the same techniques we are used to for this problem too

Can we generalize these results?

- **Difficult to generalize!**
  - Especially to problems like neural nets that are hard to analyze algebraically
- This PCA objective is one of **the simplest non-convex problems**
  - It's just a degree-4 polynomial
- But these results can **give us intuition** about how our methods apply to the non-convex setting
  - To understand a method, PCA is a good place to start

Deep Learning as Non-Convex Optimization

Or, "what could go wrong with my non-convex learning algorithm?"

Lots of Interesting Problems are Non-Convex

- Including deep neural networks

- Because of this, we almost always **can't prove convergence** or anything like that when we run backpropagation (SGD) on a deep net

- But can we use intuition from PCA and convex optimization to understand **what could go wrong when we run non-convex optimization** on these complicated problems?

What could go wrong?
We could converge to a bad local minimum

- Problem: we converge to a local minimum which is bad for our task
  - Often in a **very steep potential well**

- One way to debug: re-run the system with **different initialization**
  - Hopefully it will converge to some other local minimum which might be better

- Another way to debug: **add extra noise to gradient updates**
  - Sometimes called "stochastic gradient Langevin dynamics"
  - Intuition: extra noise pushes us out of the steep potential well

What could go wrong?

We could converge to a saddle point

- Problem: we converge to a saddle point, which is not locally optimal

- Upside: **usually doesn't happen with plain SGD**
  - Because noisy gradients push us away from the saddle point
  - But can happen with more sophisticated SGD-like algorithms

- One way to debug: find the **Hessian** and compute a descent direction

What could go wrong?

We get stuck in a region of low gradient magnitude

- Problem: we converge to a region where the gradient's magnitude is small, and then stay there for a very long time
  - Might not affect asymptotic convergence, but very bad for real systems
- One way to debug: use specialized techniques like batchnorm
  - There are many **methods for preventing this problem for neural nets**
- Another way to debug: design your network so that it doesn't happen
  - Networks using a **RELU activation** tend to avoid this problem

What could go wrong?

Due to high curvature, we do huge steps and diverge

- Problem: we go to a region where the gradient's magnitude is very large, and then we make a series of very large steps and diverge
  - Especially bad for real systems using floating point arithmetic

- One way to debug: use adaptive step size
  - Like we did for PCA
  - **Adam** (which we'll discuss on Wednesday) does this sort of thing

- A simple way to debug: just limit the size of the gradient step
  - Often called **gradient clipping**
  - But this can lead to the low-gradient-magnitude issue

What could go wrong?
I don't know how to set my hyperparameters

- Problem: without theory, **how on earth am I supposed to set my hyperparameters**?

- We already have discussed the solution: **hyperparameter optimization**
  - All the techniques we discussed apply to the non-convex case.

- To avoid this: just use hyperparameters from folklore

Takeaway

- Non-convex optimization is **hard to write theory about**
- But it's **just as easy to compute SGD on**
  - This is why we're seeing a renaissance of empirical computing
- We can use the techniques we have discussed to get speedup here too
  - Including adaptive
- We can **apply intuition from the convex case and from simple problems like PCA** to learn how these techniques work

Algorithms other than SGD

Machine learning is not just SGD

- Once a model is trained, we need to use it to classify new examples
  - This **inference task** is not computed with SGD

- There are other algorithms for optimizing objectives besides SGD
  - **Stochastic coordinate descent**
  - **Derivative-free optimization**

- There are other common tasks, such as sampling from a distribution
  - **Gibbs sampling** and other Markov chain Monte Carlo methods
  - And we sometimes use this together with SGD → called **contrastive divergence**

Why understand these algorithms?

- They represent a significant fraction of machine learning computations
  - **Inference** in particular is huge
- You may want to use them **instead of SGD**
  - But you don't want to suddenly pay a computational penalty for doing so because you don't know how to make them fast
- **Intuition from SGD** can be used to make these algorithms faster too
  - And vice-versa

Inference

Inference

- Suppose that our training loss function looks like

$$f(w) = \frac{1}{N} \sum_{i=1}^{n} l(\hat{y}(w; x_i), y_i)$$

- Inference is the problem of computing the prediction

$$\hat{y}(w; x_i)$$

How important is inference?

- **Train once, infer many times**
  - Many production machine learning systems just do inference
- Image recognition, voice recognition, translation
  - All are just applications of inference once they're trained
- Need to get **responses to users quickly**
  - On the web, users won't wait more than a second

Inference on linear models

- Computational cost: relatively **low**
  - Just a matrix-vector multiply

- But still can be more costly in some settings
  - For example, if we need to compute a random kernel feature map
  - **What is the cost of this?**

- **Which methods can we use to speed up inference in this setting?**

Inference on neural networks

- Computational cost: **relatively high**
  - Several matrix-vector multiplies and non-linear elements

- **Which methods can we use to speed up inference in this setting?**

- **Compression**
  - Find an easier-to-compute network with similar accuracy by fine-tuning
  - We'll see this in more detail later in the course.

Other techniques for speeding up inference

- Train a fast model, and run it most of the time
  - If it's **uncertain**, then run a more accurate, slower model

- For video and time-series data, **re-use some of the computation** from previous frames
  - For example, only update some of the activations in the network at each frame
  - Or have a more-heavyweight network run less frequently
  - Rests on the notion that the **objects in the scene do not change frequently** in most video streams

Other Techniques for Training, Besides SGD

Coordinate Descent

- Start with objective

$$\text{minimize:} f(x_1, x_2, \ldots, x_n)$$

- Choose a random index **i**, and update

$$x_i = \arg\min_{\hat{x}_i} f(x_1, x_2, \ldots, \hat{x}_i, \ldots, x_n)$$

- And repeat in a loop

Variants

- Coordinate descent with derivative and step size
  - Sometimes called “stochastic coordinate descent”

$$x_{t+1,i} = x_{t,i} - \alpha_t \cdot \frac{\partial f}{\partial x_i}(x_{t,1}, x_{t,2}, \ldots, x_{t,n})$$

- The same thing, but with a gradient estimate rather than the full gradient.

- **How do these compare to SGD?**

Derivative Free Optimization (DFO)

- Optimization methods that don't require differentiation
- Basic coordinate descent is actually an example of this
- Another example: for normally distributed ε

$$x_{t+1} = x_t - \alpha \frac{f(x_t + \sigma\epsilon) - f(x_t - \sigma\epsilon)}{2\sigma}\epsilon$$

- **Applications?**

Another Task: Sampling

Focus problem for this setting:

Statistical Inference

- Major class of machine learning applications
  - Goal: **draw conclusions from data** using a statistical model
  - Formally: find marginal distribution of unobserved variables given observations

- Example: decide whether a coin is biased from a series of flips

- Applications: LDA, recommender systems, text extraction, data cleaning, data integration etc.

Popular algorithms used for statistical inference at scale

- Markov-chain Monte Carlo methods (MCMC)
  - **Gibbs sampling**
  - Metropolis-Hastings
  - Stochastic gradient Langevin dynamics
  - Hamiltonian Monte Carlo

- Variational inference
  - Infer by solving an optimization problem — can use many of the same techniques we have discussed in class

Graphical models

- A graphical way to describe a probability distribution

- Common in machine learning applications
  - Especially for applications that deal with uncertainty

- Useful for doing statistical inference at scale
  - Because we can leverage techniques for computing on large graphs

What types of inference exist here?

- Maximum-a-posteriori (MAP) inference
  - Find the state with the highest probability
  - Often reduces to an optimization problem
  - **What is the most likely state of the world?**

- Marginal inference
  - Compute the marginal distributions of some variables
  - **What does our model of the world tell us about this object or event?**

What is Gibbs Sampling?

Algorithm 1 Gibbs sampling

Require: Variables x_i for $1 \leq$ … π.

l…

…pling uniformly … n $\{1, \ldots, n\}$.

Re…mple x_s uniformly from $\mathbf{P}_\pi(x_s | x_{\{1,\ldots,n\} \setminus \{s\}})$.

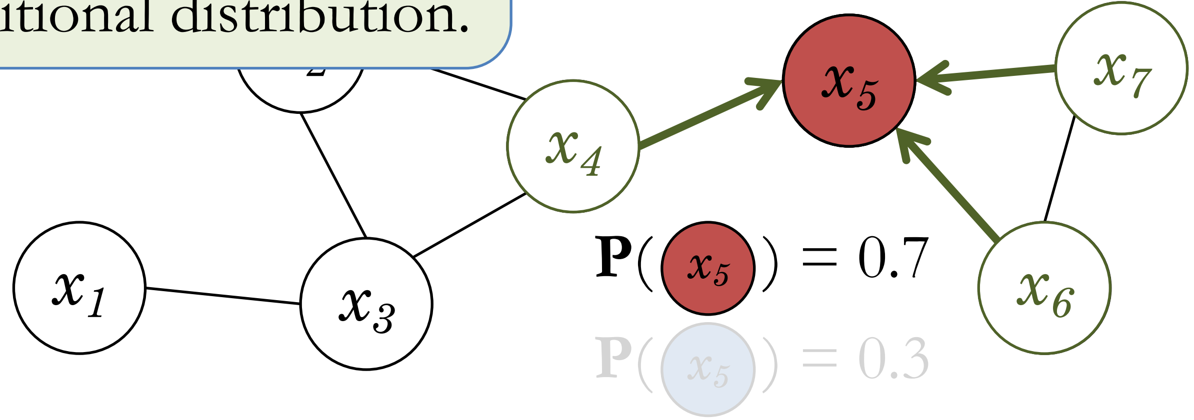
out… x

e…

Output the current state as a sample.

Compute its conditional distribution given the other variables.

Update the variable by sampling from its conditional distribution.



Learning on graphical models

- Contrastive divergence
  - SGD on top of Gibbs sampling

- The de facto way of training
  - Restricted boltzmann machines (RBM)
  - Deep belief networks (DBN)
  - Knowledge-base construction (KBC) applications

What do all these algorithms look like?

Stochastic Iterative Algorithms

Given an immutable input dataset and a model we want to output.

Repeat:

1. **Pick a data point at random**
2. **Update the model**
3. **Iterate**

same structure

same systems properties

same techniques

Questions?

- Upcoming things
  - **Project proposals due today**
  - Paper Presentation #6a and #6b **on Wednesday**
    - On adaptive learning rate methods