Lecture 11: Stochastic Gradient Descent (SGD)

Agenda

- Stochastic Gradient Descent (SGD)

Source: Sections VI.5 in Linear Algebra and Learning from Data (2019) by **Gilbert Strang**

Stochastic Gradient Descent (SGD)

All Machine Learning (ML) problems require optimization
Gradient Descent, Newton methods, subgradient and more!

$$\min_x f(x)$$

This is the implementation in MATLAB of gradient descent

And this more ancient method remains "the" method for training large scale ML systems
And in current times these optimization problems are large. So people started liking stuff like gradient descent, which was invented by Cauchy back in the day

GD: Generic Matlab code

```
function [x,f] = gradientDescent(x0)

   fx = @(x)  objective(x); % handle to f(x)
   gfx = @(x) gradient(x);  % handle to nabla f(x)

   x=x0;                    % user supplied starting point
   maxiter = 100;           % tunable parameter

   for k=1:maxiter
      g = gfx(x);           % compute gradient at x
      alpha = stepsize(k);  % implement this
      x = x - alpha*g;      % perform update
      fprintf('Iter: %d\t Obj: %d\n', fx(x));
   end
end
```

You've seen gradient descent. We're only going to change this one line (**marked yellow**). And the change of that one line, surprisingly, is driving all the deep learning toolboxes and all large-scale machine learning, et cetera.

This is an oversimplification, but morally, that's it.

So let's look at what's happening.

Stochastic Gradient Descent (SGD)

Abstractly, these are the kinds of optimization problems we are solving in machine learning

Concrete examples of these optimization problems that they look like that

Find an x over a cost function, where the cost function can be written as a sum: $min_x \frac{1}{x} \sum_{i=1}^{n} f_i(x)$, n→∝

In modern day ML parlance these are also called finite sum problems, in case you run into that term

And they just call it finite because n is finite here

In pure optimization theory parlance, n can go to infinity

And then they're called stochastic optimization problems-- just for terminology

Stochastic Gradient Descent (SGD)

Large Scale ML
Training data:{(x_1,y_1), ...,(x_n,y_n)} $\in \mathcal{R}^d \times \mathcal{Y}$ Large scale ML: both d and n are large, d-> dimension of each input sample and n-> no. of training data/samples

Both n &d are large, i.e., large ML problem
n->1 million or above, no. of training input
d->feature -> 10 million to billion raw vector $\vec{x}$->we're working with

So x_1 through x_n, these could be just raw images, for instance, in ImageNet or some other image data set. They could be text documents

y_1 through y_n, in classical machine learning, think of them as ±1 labels-- cat, not cat-- or in a regression setup as some real number

So **d and n are huge**. And it's because both d and n are huge, we are interested in thinking of optimization methods for large scale ML that can handle such big d and n

And this is driving a lot of research on some theoretical computer science to deal with these two quantities

Stochastic Gradient Descent (SGD)

Example:

Least Squares:

$$\frac{1}{n} || Ax - b ||_2^2 = \frac{1}{n}\sum_{i=1}^{n}(a_i^T x - bi)2 = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

Let's look at the most classic question, least squares regression. A is a matrix of observations--measurements. b are observations. Solve $(Ax-b)^2$. A linear system of equations, the most classical problem in linear algebra, can also be written like that

Lasso / l_1 least Squares:

$$\frac{1}{n} || Ax - b ||_2^2 + \lambda || x ||_1 = \frac{1}{n}\sum_{i=1}^{n}(a_i^T x - bi)2 + \lambda \sum_{i=1}^{l} | x_j |$$

Lasso is essentially least squares, but there's another simple term at the end. That again, looks like f of i.

Support vector machine (SVM):

$$\frac{1}{n} || x ||_2^2 + \frac{C}{n}\sum_{i=1}^{n} \max[0,1 - yi(x^T a_i - b_j)^2]$$

Support vector machines, where people works with small to medium sized data.

Deep learning stuff requires huge amount of data.

If you have small to medium amount of data, logistic regression support, vector machines, trees, etc., this will be your first go to methods.

They are still very widely used

Stochastic Gradient Descent (SGD)

Example:

Deep Neural Networks (DNN): $\frac{1}{n}\sum_{i=1}^{n} loss[yi\ , DNN(x; a_i)] = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$

Deep neural networks that are another example of this finite sum problem

So you have n training data points, there's a neural network loss, like cross entropy, or what have you, squared loss, cross entropy, -- any kind of loss, y_i's are labels– cat or not cat, or maybe a multiclass

And then you have a transfer function called a deep neural network which takes raw images as input and generates a prediction whether this is a dog or not

That whole thing we call DNN

So it's a function of a_i's which are the training data. x are matrices of the neural network, so we've just compressed the whole neural network into this notation

Once again, it's an instance of that finite sum

So that f_i in there captures the entire neural network architecture. But mathematically, it's still just one instance of this finite sum problem

Stochastic Gradient Descent (SGD)

Maximum-likelihood Estimation (mle)

$\frac{1}{n}\sum_{i=1}^{n} \log\_likelihood[(x; a_i)] = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$

This maximum likelihood estimation is log likelihood over n observations

You want to maximize log likelihood. Once again, just a finite sum

Stochastic Gradient Descent (SGD)

Why SGD?

Most of the problems that we're interested in machine learning and statistics, when they are written down as an optimization problem, they look like these finite sum problems. And that's the reason to develop specialized optimization procedures to solve such finite some problems.
And that's where SGD comes in

Basic gradient descent iteration: $f(x)= \frac{1}{n}\sum_{i=1}^{n} f_i(x)$

$x_{k+1}= x_k-\eta_k \nabla f(x) = x_k-\eta_k \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(xk)$

Just for notation, f(x) refers to that entire summation. $f_i(x)$ refers to a single component

So if you were to try to solve-- that is, to minimize this cost function, neural network, SVM, that's what one iteration would look like

Because it's a finite sum, gradients are linear operators

Gradient of the sum is the sum of the gradient-- that's gradient descent

Stochastic Gradient Descent (SGD)

- Things to think about
- What might be some drawback of this iteration?
- What else could we do?
- Pretty big sum. Especially if n is very big number, a billion
- That sum is some is huge. So getting a single gradient to do a single step of gradient descent for a large data set could take you hours or days
- So that's a major drawback

How to counter that drawback, at least, say, purely from an engineering perspective

Stochastic Gradient Descent (SGD)

Incremental gradient methods:

- What if, at iteration k, we randomly pick an Integer $i(k) \in \{1,2,\ldots,n\}$

  Randomly pick some integer, i(k) out of the n training data points, and we instead just perform this update

- And instead just perform the update?

  $x_{k+1}=x_k-\eta_k \nabla f_{i(k)}(x_k)$

  Instead of using the full gradient, you just compute the gradient of a single randomly chosen data point.

- The update requires only gradient info for $f_{i(k)}$ n times faster than using $\nabla f(x)$

  One iteration is now n times faster. If n were a million or a billion, that's super fast. But why should this work right

Stochastic Gradient Descent (SGD)

But does this make sense?

- So of course, it's n times faster, and the key question here,
- scientific question-- is does this make sense? It makes great engineering sense.
- Does it make algorithmic or mathematical sense?
- So this idea of doing stuff in the stochastic manner was actually originally proposed by Robbins and Monro, somewhere, around 1951.
- And that's the most advanced method that we are essentially using currently

Stochastic Gradient Descent (SGD)

Example-some insights

Let's look at a simple, one-dimensional optimization problem:

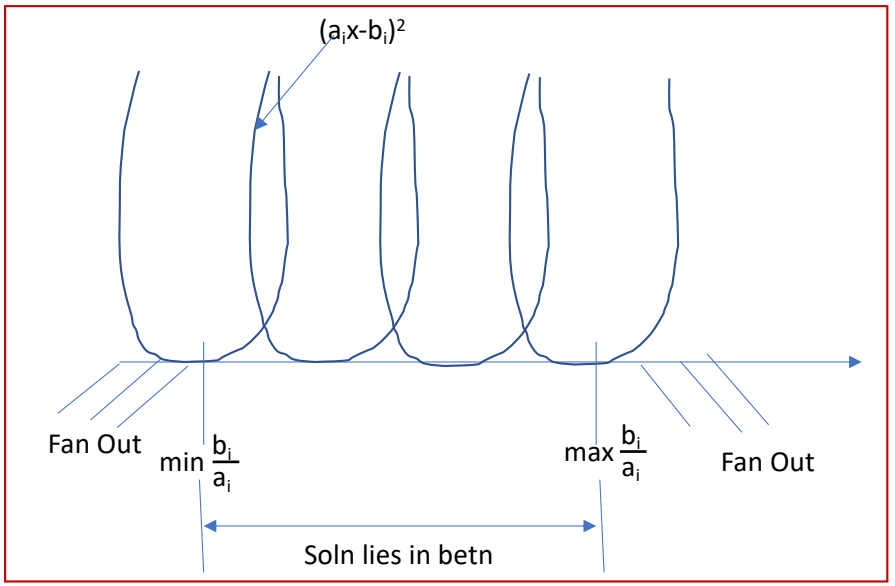
Assume all variables involved are scalars

min f(x)=$\frac{1}{2}\sum_{i=1}^{n}(aix\text{-}b_i)^2$

Solving $\nabla f(x) = 0$ we obtain

$x^*=\frac{\sum_i a_i b_i}{\sum_i a_i^2}$ $\quad (\nabla f(x) = \sum_{i=1}^{n}(aix\text{-}b_i)a_i)$

Min of $f_i(x)=\frac{1}{2}\text{-}(a_ix\text{-}b_i)^2 \Longrightarrow x_i^*=\frac{b_i}{a_i}$



Stochastic Gradient Descent (SGD)

- That means when you're outside where the individual solutions
- let's call this the fanout zone. And also, this side is the fanout zone.
- And this region, within which the true minimum can lie, you can say, that's the region of confusion
- Because there, by minimizing an individual f_i, you're not going to be able to tell what is the combined x*
- And gain some mathematical insight into that simulation
- that if you have a scalar x that is outside this region of confusion
- which states that if you're far from the region within which an optimum can lie
- So you're far away

Stochastic Gradient Descent (SGD)

Assume all variables involved are scalars

min $f(x)=\frac{1}{2}\sum_{i=1}^{n}(aix\text{-}b_i)^2$

If we have a scalar x that lies outside R?

$$\nabla f_i(x) = (a_i x\text{-}b_i)a_i$$

$$\nabla f(x) = \sum_{i=1}^{n}(aix\text{-}b_i)a_i$$

$\nabla f_i(x)$ has **same sign** as $\nabla f(x)$
So $\nabla f_i(x)$ **instead** of $\nabla f(x)$ also ensures progress

So suppose that's where you are. What happens when you're in that far out region?

So if you're in the fan out region, you use a stochastic gradient of some i-th component

What does gradient descent do? It says walk in the direction of the negative gradient.

And far away from the optimum, outside the region of confusion, you've stochastic gradient has the same sign as the true gradient

Maybe in more linear algebra terms, it makes an acute angle with your gradient.

So that means if even though a stochastic gradient is not exactly the full gradient, it has some component in the direction of the true gradient

Stochastic Gradient Descent (SGD)

- This is one 1D
- Here it is, exactly the same sign
- In multiple dimensions, this is the idea that it'll have some component in the direction of true gradient when you're far away
- Which means, if you then use that direction to make an update in that style, you will end up making solid progress
- And the beauty is, in the time it would have taken you to do one single iteration of batch gradient descent
- far away you can do millions stochastic steps, and, each step will make some progress

Stochastic Gradient Descent (SGD)

Mathematical ideas behind SGD

$\min_x f(x)$

Suppose instead of subroutines to compute f(x) and $\nabla f(x)$, we are only given access to **noisy estimates**

SGD uses "Stochastic gradients" g(x) such that $\mathbb{E}[g(x)] = \nabla f(x)$

The true gradient was expensive to compute, so we create a randomized estimate of the true gradient. And the randomized estimate is much faster to compute. SGD is one example

And mathematically, what will start happening is, depending on how good randomized estimate is, our method may or may not convert to the right answer. If we have a difficult quantity, produce a randomized estimate and save on computation. And this is the key property
So stochastic gradient descent, it uses stochastic gradients. Stochastic is, here, used very loosely. And it just means that some randomization. That's all it means

Key property that we have is in expectation. The expectation is over whatever randomness you used. So if you picked some random training data point out of the million, then the expectation is over the probability distribution over what kind of randomness you used. If you picked uniformly at random from a million points, then this expectation is over that uniform probability

Stochastic Gradient Descent (SGD)

In Expectation to “true gradients”

Unbiased property⟶smaller variance⟹ truly gradient

Key property for SGD is that that over that randomness. The thing that instead of the true gradient in expectation actually it is the true gradient. So in statistics language, this is called the stochastic gradient that we use is an unbiased estimate of the true gradient

Using stochastic gradients by encapsulating everything within expectations over the randomness. In particular, the unbiasedness is great. But there's another very important aspect to why it works, beyond this unbiasedness, that the amount of noise, or the amount of stochasticity is controlled. So just because it is an unbiased estimate, doesn't mean that it's going to work that well.

Why? Because it could still fluctuate hugely. Essentially, plus infinity here, minus infinity here. You take an average, you get 0. So that is essentially unbiased, but the fluctuation is gigantic. So whenever talking about estimates, what's the other key quantity we need to care about beyond expectation?

Variance. And really, the key thing that governs the speed at which stochastic gradient descent does the job that we want it to do is, how much variance do the stochastic gradients have?

Stochastic Gradient Descent (SGD)

Two Variants

$$\frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

Start with feasible x_0
For k=0,1,..

- Option I: Randomly pick an index i with replacement
- Option II: Pick index I without without replacement
- Use $g_k = \nabla f_i(x)$ as the stochastic gradient
- Update $x_{k+1} = x_k - \eta_k g_k$

Option 1 says, randomly pick some training data point, use its stochastic gradient. What do we mean by randomly pick? The moment you use the word random, you have to define what's the randomness.
So one randomness is uniform probability over n training data points. That is one randomness.

The other version is you pick a training data point without replacement. So with replacement means uniformly at random.
Each time you draw a number from 1 through n, use their stochastic gradient, move on.
Which means the same point can easily be picked twice, also

And without replacement means, if you've picked a point number three, you're not going to pick it again until you've gone through the entire training data set. Those are two types of randomness

Actually, every person in the real world is using version 2. Are you really going to randomly go through your RAM each time to pick random points? That'll kill your GPU performance like anything

Stochastic Gradient Descent (SGD)

Mini-batch Data

$$Min\ x$$
$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

Idea: Use a mini-batch of stochastic gradients

$x_{k+1}=x_k-\frac{\eta_k}{|I_k|}\sum_{j\in I_k}\nabla f_j(x_k)$

- Each iteration uses $|I_k|$ stochastic gradient
- Useful in parallel settings (eg. GPU based implementation
- Increases parallelism, reduces communication (in distributed settings)

Remark: very large mini-batches not that "favorable" for DNN training

Suppose we had a million points-- each time, instead of picking one, maybe pick 10, or. 100, or 1,000, or what have you. So this averages things. Averaging things reduces the variance. So this is actually a good thing, because the more quantities you average, the less noise you have

A mini-batch of size 1 is the pure vanilla SGD. Mini-batch of size n is nothing other than pure gradient descent. Something in between is what people actually use

Mini-batches are really crucial, especially in the deep learning, GPU-style training, because they allow you to do things in parallel. So mini-batches, the larger the mini batch the more things you can do in parallel

In machine learning you want some region of uncertainty. And what it means actually is, a lot of people have been working on this, including at big companies, that if you reduce that region of uncertainty too much, you end up over-fitting your neural network.
And then it starts sucking in its test data, unseen data performance.
So even though for parallelism, programming, optimization theory, big mini-batch is awesome, unfortunately there's price to be paid, that it hurts your test error performance

Stochastic Gradient Descent (SGD)

Practical Challenges

$$x_{k+1}=x_k-\frac{\eta_k}{|I_k|}\sum_{j\in I_k}\nabla f_j(x_k)$$

- How to pick up step sizes
- Which mini-batch to use
- How to compute Stochastic gradients
- Gradient clipping
- Adding momentum

People have various heuristics for solving these challenges. You can cook up your own, but it's not that one idea always works. So if you look at SGD, what are the moving parts? The moving parts in SGD-- the gradients, stochastic gradient, the step size, the mini batch. So how should I pick step sizes-- very non-trivial problem

Different deep learning toolkits may have different ways of automating that tuning, but it's one of the painful things. Which mini batch to use? With replacement, without replacement. But which mini batch should I use, how large that should be? Again, not an easy question to answer

How to compute stochastic gradients. Does anybody know how stochastic gradients are computed for deep network training? Anybody know? There is a very famous algorithm called back propagation. That back propagation algorithm is used to compute a single stochastic gradient

Some people use the word back prop to mean SGD. But what back prop really means is some kind of algorithm which computes for you a single stochastic gradient.
And hence this TensorFlow, et cetera-- these toolkits-- they come up with all sorts of ways to automate the computation of a gradient. Because, really, that's the main thing.
And then other ideas like gradient clipping, and momentum, et cetera.
There's a bunch of other ideas.