Introduction to Optimization - Theory & Practice

Mark Schmidt

Canada CIFAR AI Chair
Fellow | Amii
Associate Professor | University of British Columbia
Introduction

• 1\textsuperscript{st} paragraph of CIFAR e-mail:
  – I am writing ... to invite you to deliver the lecture on “Introduction to Optimization – Theory & Practice”.

• My reply after reading first paragraph:
  – Yup, happy to do it!

• 3\textsuperscript{rd} (unread) paragraph of CIFAR e-mail:
  – This year, we are asking all of our speakers to provide a \textbf{45 minute} lecture.

• Me after looking later reading full invite:
  – “How do I possibly cover optimization theory \textit{and} practice in 45 minutes?”
Real Introduction

- I am assuming **you have already trained** a deep learning model.
  - Probably using **stochastic gradient descent (SGD)** or a variant like **Adam**.
- And it might useful if I could answer questions like these:
  - Question 1: How do I set the step size?
  - Question 2: How I pick the mini-batch size?
  - Question 3: Should I use variance reduction?
  - Question 4: What about random shuffling?
  - Question 5: Should I use importance sampling?
  - Question 6: Are there faster algorithms?
  - Question 7: Should we just use Adam?
Disclaimer

• These questions are complicated, depending on many factors.

• I will just focus on applying SGD to differentiable functions.
  – And I will assume the basic setting of IID data (no reinforcement learning).

• I will overview some important trade-offs to consider in this setting.
  – Which may help improve your understanding of these questions.
  – I will necessarily need to skip over a lot of precise details.

• If you want to see the long/detailed version, see here:
  – https://www.cs.ubc.ca/~schmidtm/Courses/5XX-S22 (in progress, 6 hours in).
Optimization and Gradient Descent

• We usually “train” models by solving an optimization problem.

\[
\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w)
\]

– We have ‘d’ parameters ‘w’ that we can modify the fit the data better.
  • For a deep neural network, this would contain the parameters of the various layers.
– We have a function ‘f’ that measures how well we fit the data.
  • Usually this function is the average over a set of ‘n’ functions f_i.
  • Each f_i measures how well we fit training example ‘i’.
    – Could be squared error or cross entropy of network’s predictions.

• Most popular DL algorithm is stochastic gradient descent (SGD).
Deterministic and Stochastic Gradient Descent

• For minimizing functions of the form:

\[ f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w) \]

• **Deterministic gradient descent** repeats the iteration:

\[ w_{k+1} = w_k - \alpha_k \nabla F(w_k) \]

  – For a “step size” \( \alpha_k \).

• **Stochastic gradient descent** repeats the iteration:

\[ w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k) \]

  – For a random training example ‘\( i_k \)’.  
  – Key advantage is cost per iteration: 1 billion times faster for 1 billion examples.
Deterministic Gradient Descent in 1 Dimension

- **Deterministic gradient descent** is based on a simple observation:
  - Given parameters ‘w’, the **direction of largest decrease** is $-\nabla f(w)$. 
Deterministic Gradient Descent in 1 Dimension

- **Deterministic gradient descent** is based on a simple observation:
  - Give parameters ‘$w$’, the direction of largest decrease is $-\nabla f(w)$.
Deterministic Gradient Descent in 1 Dimension

- **Deterministic gradient descent** is based on a simple observation:
  - Give parameters ‘$w$’, the direction of largest decrease is $-\nabla f(w)$.
Deterministic Gradient Descent in 1 Dimension

- **Deterministic gradient descent** is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.
Deterministic Gradient Descent in 1 Dimension

- **Deterministic gradient descent** is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.

Now the slope $\nabla f(w_4)$ is positive, so we move in the negative direction.
Deterministic gradient descent (DGD) converges with constant $\alpha_k$.

Stochastic gradient descent (SGD) converges with decreasing $\alpha_k$.

Both methods are local optimization methods:

- May not find the global mimimum.
- But with appropriate step sizes, we have that $\nabla f(w_k)$ converges to 0.
Polyak-Lojasiewicz Inequality and Invexity

- For some functions, we can show that DGD/SGD find a global minimum.

- Recent works argue that some deep learning problems satisfy PL inequality.
  - Polyak-Lojasiewicz inequality.
    - “Gradient increases at least quadratically as we increase $f(w)$ above global minima”.
  - PL condition implies invexity.
    - Invexity is a generalization of convexity, where $\nabla f(w_k) = 0$ implies that $w_k$ is a global min.
Question 1: How do I set the step size?

- For gradient descent, there is a **trade-off** between:
  \[ \alpha_k \text{ "step size"} \quad \text{vs.} \quad \frac{2}{L} \text{ "Lipschitz constant of gradient"} \]

- The number ‘L’ is **how fast the gradient can change**:
  - The “Lipschitz constant”: \(|\nabla f(w) - \nabla f(v)| \leq L||w - v||

- Need smaller steps for bigger ‘L’.

---

- Function with a small ‘L’
- Function with a big ‘L’
Question 1: How do I set the step size?

- For gradient descent, there is a trade-off between:
  \[ \alpha_k \text{ vs. } \frac{2}{L} \]
  - "Step size"
  - "Lipschitz constant of gradient"

- If \( \alpha_k < \frac{2}{L} \), DGD is guaranteed to decrease the function ‘f’.
- If \( \alpha_k \) is much smaller than \( \frac{2}{L} \), DGD will converge more slowly.
  - Step size is too small.
- If \( \alpha_k \) is bigger than \( \frac{2}{L} \), you might increase the function.
  - Step could be too big and “overshoot” region where function decreases.
  - But you might get lucky and decrease the function a lot.
  - In practice, people use clever step sizes or line searches to make DGD work well.
Comparison of Fixed Step Sizes for DGD

• 25 steps of deterministic gradient descent with different $\alpha_k$.

• Progress is very slow if $\alpha_k$ is too small.

- 1/L
- 1/2L
- 1/10L
Comparison of Fixed Step Sizes for DGD

• Increasing the step size:

  - Progress can be faster if $\alpha_k \geq 2/L$ or it may increase function.
Comparison of **Adaptive Step Sizes** for DGD

- For DGD you can also use **clever step sizes** or **line-searches**.

- Better than fixed steps in practice, do not work for SGD in general.
Question 1b: How do I set the SGD step size?

- For stochastic gradient descent, there is a second trade-off:
  
  \[ \|\nabla f(w_k)\|^2 \text{ vs. } \alpha_k \sigma_k^2 \]

  - "Size of gradient of overall function" vs. "Variation in the stochastic gradients".

- The number \( \sigma_k^2 \) measures the variation in the gradients.
  - We have \( \sigma_k^2 = \frac{1}{n} \sum_{i=1}^{n} \|\nabla f_i(w^k) - \nabla f(w^k)\|^2 \).

- Need smaller steps for bigger \( \sigma_k^2 \) (but also depends on \( \|\nabla f(w_k)\| \)).
Question 1b: How do I set the SGD step size?

• For stochastic gradient descent, there is a second trade-off:
  
  \[
  \frac{\|\nabla f(w_k)\|^2}{\alpha_k \sigma_k^2} \quad \text{vs.} \quad \alpha_k \sigma_k^2
  \]

  "Size of gradient of overall function" vs. "Step size" vs. "Stochastic gradients"

• If \( \|\nabla f(w_k)\|^2 \gg \alpha_k \sigma_k^2 \), SGD is guaranteed to decrease expected ‘f’.
  – As long as we still have \( \alpha_k < 2/L \).
  – For DGD, we do not need to worry about this trade-off because \( \sigma_k = 0 \).

• If \( \alpha_k \sigma_k^2 \gg \|\nabla f(w_k)\|^2 \), noise in SGD makes us increase expected ‘f’.
  – Step size if too big, causing the noise to interfere with progress.
  – Need to decrease step size to re-gain progress, but this slows down convergence.
Question 1b: How do I set the SGD step size?

• For stochastic gradient descent, there is a second trade-off:
  
  \[ \| \nabla f(w_k) \| \quad \text{vs.} \quad \alpha_k \]

  "Size of gradient of overall function" \quad \text{"Variation in the stochastic gradients"}

• Optimization: we want \( \nabla f(w_k) \) to converge to 0 as ‘k’ grows.
  – So we need to use decreasing step sizes to guarantee continued progress.
    • But as we decrease the step size SGD will converge slower.

• Machine learning: we only need \( \nabla f(w_k) \) to close to 0.
  – We expect test error to similar for all \( w_k \) “close enough” to stationary point.
    • If you measure the test error to 2 decimal places, may not need 10 decimal places of accuracy.
  – For any “closeness”, we could use a small-enough constant step size \( \alpha_k = \alpha \).
    • Guarantees progress until \( \nabla f(w_k) \) is “close enough” to zero.
    • But in areas where gradient is small, SGD can behave erratically.
Question 1b: How do I set the SGD step size?

- In practice, SGD algorithm does not have access to $\nabla f(w_k)$ or $\sigma_k$.
  - But we can use the trade-offs to guide practical choice of step size.
Question 1b: How do I set the SGD step size?

- In practice, SGD algorithm does not have access to $\nabla f(w_k)$ or $\sigma_k$.
  - But we can use the trade-offs to guide practical choice of step size.
Question 2: How do I pick the mini-batch size?

- Instead 1 random example, we often use a mini-batch in SGD:
  \[ w_{k+1} = w_k - \alpha_k \left( \frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(w_k) \right) \]
  - We can often compute gradient of many examples in parallel.
  - The mini-batch gives a better approximation of true gradient \( \nabla f(w_k) \).

- With random mini-batches, second part of trade-off changes to:
  \[ \| \nabla f(w_k) \|^2 \quad \text{vs.} \quad \frac{\alpha_k}{|B_k|} \]
  - “With a mini-batch size of 100, effect of noise is divided by 100.”
  - “With a mini-batch size of 100, you can use a step size that 100-times larger.”
  - As long as the step size is still less than \( 2/L \).
Question 2b: How do I grow the mini-batch size?

• If $||\nabla f(w_k)||^2 \gg \alpha_k \sigma_k^2 / |B_k|$, mini-batch SGD guarantees progress.
  – Again, you can decrease $\alpha_k$ to continue progress as $\nabla f(w_k)$ goes to 0.
  – Or, you can increase the batch size $|B_k|$ as $\nabla f(w_k)$ goes to 0 (“batching”).
    • Growing batch sizes gives convergence with a constant step size.

• The situation is better if you have a finite data set.
  – If you sample mini-batch without replacement from ‘n’ examples:
    $$\frac{\alpha_k \sigma_k^2}{|B_k|} \left( n - |B_k| \right) \rightarrow \frac{\alpha_k \sigma_k^2}{n} \text{ “finite sample correction”}$$
    – Drives the effect of $\sigma$ to 0 as $|B_k|$ approaches ‘n’ (instead of $\infty$).

• Growing batches allow using clever step sizes and line-searches.
Comparison of Deterministic, Stochastic, and Hybrid

- Comparing DGD to SGD and a growing batch ("hybrid") method:
Question 3: Should I use variance reduction?

• SGD with a growing batch size ("batching"):
  – Use an estimate of $\nabla f(w_k)$ where $\sigma_k$ converges to 0 as $w_k$ goes to $w_\ast$.
  – But batching may lose low iteration cost of SGD.

• **Variance-reduced** stochastic gradient methods (like SAG and SVRG):
  – Design an estimate of $\nabla f(w_k)$ where $\sigma_k$ converges to 0 as $w_k$ goes to $w_\ast$.
  – But only need 1 random example per iteration.
    • So these methods have the low iteration cost of SGD.

• Variance-reduced methods remove second part of trade-off.
  – This improves convergence speed a lot for under-parameterized models.
  – But variance reduction is not needed for over-parameterized models.
Over-Parameterized Gradient Descent (OGD)

- A model is over-parameterized if it can fit data with training error 0.

- How does this affect optimization?
  - In this setting, when we minimize \( f \) we also minimize each \( f_i \).
  - So at a global minimum \( w_* \) we have \( \nabla f_i(w_*) = 0 \) for all \( i \).
  - And this means that \( \sigma_*^2 = 0 \) (there is no noise at the minimum).
  - So \( \sigma_k \) goes to 0 as \( \nabla f(w_k) \) goes to 0.

- I will use term OGD for over-parameterized gradient descent.
  - When we apply SGD with a constant step size to an over-parameterized problem.
Over-Parameterized Gradient Descent (OGD)

• Since $\sigma_k$ goes to 0, OGD is less affected by the second trade-off.
  – It converges like DGD with the iteration cost of SGD.
  – No need to decrease step sizes or increase batch sizes.
  – Still expect good performance if you are close to being over-parameterized.
    • Will not converge but will get close to solution with large steps, since $\sigma_*^2$ will be small.
Question 3: Should I use variance reduction?

• For over-parameterized models:
  – No need for variance reduction, just use OGD.
    • Variance-reduce methods might be slower than SGD with constant step size.

• For under-parameterized models:
  – Variance reduction may lead to faster convergence.

• Example: generative adversarial networks (GANs).
  – GAN loss cannot be driven to zero.
  – So variance-reduced SGD can lead to faster convergence.
Question 1c: How do I adjust the step size?

• Could we somehow adjust the SGD step size as we go?
  – Accounting for both trade-offs and hoping to get “lucky”?

• Lots of methods have been proposed to do this.
  – “Update step size based on some simple statistics”.
  – “Do a line-search based on the mini-batch”.
  – “Do gradient descent on the step size”.
  – Mark being provocative: almost all of these methods are bad.

• Most of these methods have one at least of these problems:
  – Introduce a new hyper-parameter that is just as hard to tune.
  – Do not converge theoretically (can catastrophically fail).
  – Converges theoretically but works badly in practice.
  – Need to assume that $\sigma_k$ goes to 0.
Question 1c: How do I adjust the step size?

• For under-parameterized problems with fixed batch sizes:
  – I have not had good luck getting anything working well across problems.
    • And I have tried a lot of things over the years.
   – My students say that the recent “coin betting” method works well.
    • And is justified theoretically.

• For over-parameterized problems or batching (where $\sigma_k$ goes to 0):
  – Can often adapt clever step sizes or line searches designed by DGD.

• Example: for OGD you can use an Armijo backtracking line search.
  – “Decrease step size if it makes less progress than a step size like 1/L would.”
  – Theory: “Performs as well or better as best fixed step size” (without knowing ‘L’).
    • And with careful implementation, cost is less in practice than trying out 2 fixed step sizes.
OGD with Armijo Line-Search

- **SGD + Armijo** outperforms other methods on many benchmarks.
OGD with Armijo Line-Search

- **SGD + Armijo** outperforms other methods on many benchmarks.
  - Uses different step sizes on different datasets at different times:

![Graph showing SGD+Armijo Step-Sizes](image)
Question 4: What about random shuffling?

• How should we choose the random example?
  – Classic SGD theory: must sample $i_k$ uniformly from \{1,2,...,n\}.
    • Or with importance sampling from a fixed distribution.

• In practice, people often use random shuffling:
  – Compute a random permutation of the training examples.
  – Go through this random permutation in order.
    • Guarantees each example is chosen at least once every ‘n’ iterations.

• Empirical evidence that random shuffling converges faster.
  – Theory is catching up, showing that it likely is always faster.
Question 5: Should I use importance sampling?

• What about sampling $i_k$ from a non-uniform distribution?

• For OGD, importance sampling leads to a faster method:
  – Suppose you know the Lipschitz constant $L_i$ of each training example.
  – OGD converges faster if you bias sampling by the $L_i$ values.
    • “If the gradient of example ‘i’ can change quickly, sample it more often.”
      – For classification problems, the “local” $L_i$ is small if the examples are correctly classified.
    • Normally OGD needs a step size less $2/\max(L_i)$, this allows step sizes less $2/\mean(L_i)$.

• Importance sampling may also help in “low noise” scenarios.
  – But if $\sigma_*$ is large, we do not expect important sampling to help.
Question 6: Are there faster algorithms?

• We have faster deterministic algorithms than DGD in various settings:
  – For quadratic functions, we can use the heavy-ball method:
    \[ u_{k+1} = w_k - \alpha_k \nabla f(w_k) + \beta_k (w_k - w_{k-1}) \]
    • Which adds a momentum term to DGD (for some momentum parameter \( \beta_k < 1 \)).
  – For convex functions, we have Nesterov’s accelerated gradient method(s):
    \[ u_{k+1} = w_k - \alpha_k \nabla f(w_k) + \beta_k (w_k - w_{k-1}) + \alpha_k \beta_k (\nabla f(w_k) - \nabla f(w_{k-1})) \]
    • Which can add a “gradient difference” to the heavy-ball method.
  – Nearby a strict local minimum, we can use variations on Newton’s method:
    \[ w_{k+1} = w_k - \alpha_k \left[ \nabla^2 f(w_k) \right]^{-1} \nabla f(w_k) \]
    • Which uses or approximates the second-derivative matrix to converge faster locally.
Gradient Descent vs. Heavy-Ball Method

Gradient Method

Heavy-ball Method

Approaches from left

Bounce around
Gradient Descent vs. Heavy-Ball Method

• Adding **momentum** to DGD with fixed step size and momentum:
Gradient Descent vs. Heavy-Ball Method

• For deterministic quadratics, exist clever ways to set $\alpha_k$ and $\beta_k$. 
Gradient Descent vs. Heavy-Ball Method

- For deterministic quadratics, Newton converges in 1 step.
Question 6: Are there faster algorithms?

• For over-parameterized or growing batch settings:
  – Ideas like momentum/Nesterov/Newton can make SGD converge faster.

• These faster methods should still work well in low-noise settings.

• But these methods do not improve the dependence on $\sigma_k$.
  – We do not expect them to converge faster if noise is large.
  – These methods can even amplify the effect of the noise.
    • SGD may need smaller step sizes if you are using momentum.
    • You might need momentum parameter $\beta_k$ to converge to 0.
Question 7: Should we just use Adam?

• An extremely popular SGD variant is Adam:
  – Combination of several ideas with good properties in specific settings:
    • AdaGrad which has appealing properties for online learning.
    • Momentum which has appealing properties for deterministic quadratic functions.
    • Step sizes that are constant which has appealing properties for low-noise problems.
  • The properties of Adam are not well understood.
    – Analysis in original Adam paper was wrong.
    – There are simple functions where Adam performs terribly.
    – But it is difficult to beat Adam on some architectures like transformers.
    – ?????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????
Gradient Descent vs. Adam

- DGD vs. Adam:

**Gradient Descent (alpha=1/L)**

**Adam (default)**
Gradient Descent vs. Adam

- DGD vs. Adam:
Gradient Descent vs. Adam

• DGD vs. Adam:
Question 7: Should we just use Adam?

- Adam is successful because it better handles heavy-tailed noise?
  - Nope, Adam still outperforms DGD/SGD on neural nets if you remove the noise.

- Adam is successful because we usually apply it to low-noise problems?
  - Nope, Adam can still perform badly on over-parameterized problems.
    • And it works great on some problems that are under-parameterized.

- Adam has more hyper-parameters to search over?
  - Nope, many people just use the default parameters.

- We co-evolved networks architectures with Adam?
  - We have “converged” to architectures that happen to be well-suited for Adam?
    • Would explain abnormally good performance of default parameters.
  - If so, what are the properties that make it well-suited?

- Batch normalization is another common method with missing theory.
  - Though there are many recent papers on this topic.
Summary: The Questions vs. Over-Parameterization

• Answers to our questions depend on if you are over-parameterized.
  • Question 1: How do I set the step size?
    – Bigger is better, but make sure $\alpha_k < 2/L$ and $||\nabla f(w_k)||^2 < \alpha_k \sigma_k^2$.
    – Constant or line-search for over-parameterized, decreasing and slow convergence for under-parameterized.
  • Question 2: How I pick the mini-batch size?
    – Bigger is better, if it does not increasing the iteration cost.
    – Growing batch sizes makes under-parameterized look like over-parameterized.
  • Question 3: Should I use variance reduction?
    – No for over-parameterized, yes for under-parameterized.
  • Question 4: What about random shuffling?
    – Very likely a good idea.
  • Question 5: Should I use importance sampling?
    – Yes for over-parameterized, no for under-parameterized.
  • Question 6: Are there faster algorithms?
    – Yes for over-parameterized, no for under-parameterized.
  • Question 7: Should we just use Adam?
    – Maybe, I have more questions than answers here.