Hyperparameter optimization
Review — We’ve covered many methods

• Stochastic gradient descent
  • Step size/learning rate, how long to run
• Mini-batching
  • Batch size
• Momentum
  • Momentum parameter
• Kernel trick/feature extraction
  • How many features to extract
• Variance reduction
  • Step size, epoch length

How do we set these parameters?
So Far: Theory

• Theoretical analysis of convex problems gives us a recipe for assigning hyperparameters
  • Also gives guarantees that the algorithm will converge with some optimal rate

• Often based on strong-convexity/Lipschitz constants $\mu$, $L$, etc.
  • Parameters that we can bound analytically, regardless of the data

• This is usually enough to get an asymptotically optimal rate
  • Certainly in the worst case
The Worst-Case Perspective

• Essentially, the theory I showed you is doing

\[
\arg \min_{\text{parameters}} \max_{\text{data}} \text{(objective)}
\]

• We’re not using the training data at all to set the parameters
  • Or if we are, we’re only using it to compute constants like \(\mu\) and \(L\)

• Question: can we use the data to improve our choice of parameters over what the theory gives us?
Demo
What happened?

• Theory only \textit{minimizes an upper bound} on the objective.

• But actual algorithm can \textit{do much better than the bound}.
  • As we saw in the demo.

• Problem: in the demo, to find the best parameter setting, we had to first solve the problem exactly, then run the algorithm many times.
  • \textit{Computationally intractable} in practice!

• \textit{Can we use a cheaper heuristic to set the parameters?}
Hyperparameter Optimization
Hyperparameter Optimization

• Also called metaparameter optimization
  • I am more used to this term so you will hear me using it more often

• Also called tuning

• Any system that chooses hyperparameters automatically

• What’s the difference between the model parameters, and the hyperparameters?
Many Settings; Many Strategies

• In some settings, just care about the model accuracy
  • Just want to set things like the learning rate

• In other settings, also want to make the hardware fast
  • Want to choose what hardware to run on, how many cores, etc.

• In all settings, there’s many ways to do hyperparameter optimization
Simplest Strategy:
The Null Hyperparameter Optimizer

- Simplest thing to do is to just set the parameters based on folklore.

- Minibatch size: \( b = \) 

- Momentum: \( \beta = \) 

- SVRG: epoch length = ? \times \text{Training Set Size}
The Effect of Using Folklore

• **Folklore can lead you astray!**
  - Can actually find **simple cases where the folklore settings are wrong**.
  - This is a good way to start a research paper.

• …but **folklore is folklore for a reason!**
  - It exists where people have found empirically that they get good results.
  - So when you try something new, the **first thing to compare to is folklore**.

• To be honest, the results you get from just using the folklore settings are really not that bad for a lot of practical purposes.
From the simplest strategy to…
The Most Complicated Strategy

• Spend twenty-five years training a Strong AI on custom hardware, then have it set your hyperparameters.

• …more explicitly, just get a human to set your hyperparameters.

• Fortunately, we happen to have a lot of humans
  • But human effort, particularly expert human effort, doesn’t scale.
Tuning By Hand

• Just fiddle with the parameters until you get the results you want

• Probably the most common type of hyperparameter optimization

• Upsides: the results are generally pretty good…

• Downsides: lots of effort, and no theoretical guarantees
  • Although there’s nothing fundamental that prevents us from having theory here
Demo
Grid Search

• Define some grid of parameters you want to try

• Try all the parameter values in the grid
  • By running the whole system for each setting of parameters

• Then choose the setting with the best result

• Essentially a brute force method
Downsides of Grid Search

• As the number of parameters increases, the cost of grid search increases exponentially!
  • Why?

• Still need some way to choose the grid properly
  • Something this can be as hard as the original hyperparameter optimization

• Can’t take advantage of any insight you have about the system!
Making Grid Search Fast

• **Early stopping to the rescue**
  • Can run all the grid points for one epoch, then discard the half that performed worse, then run for another epoch, discard half, and continue.

• Can **take advantage of parallelism**
  • Run all the different parameter settings independently on different servers in a cluster.
  • An **embarrassingly parallel task**.
  • Downside: **doesn’t reduce the energy cost**.
One Variant: Random Search

• This is just grid search, but with randomly chosen points instead of points on a grid.

• **This solves the curse of dimensionality**
  • Don’t need to increase the number of grid points exponentially as the number of dimensions increases.

• Problem: with random search, *not necessarily going to get anywhere near the optimal parameters* in a finite sample.
One Variant: Best Ball

• Works with epochs.

• At each epoch, do a small grid search around the current hyperparameter settings

• Then evaluate the objective and choose the “best ball”
  • The choice of parameters that gave the best objective for that epoch

• And repeat until a solution of desired quality is achieved.
An Alternative: Bayesian Optimization

• Statistical approach for minimizing noisy black-box functions.

• Idea: learn a statistical model of the function from hyperparameter values to the loss function
  • Then choose parameters to minimize the loss under this model

• Main benefit: choose the hyperparameters to test not at random, but in a way that gives the most information about the model
  • This lets it learn faster than grid search
Effect of Bayesian Optimization

• Downside: it’s a pretty **heavyweight method**
  • The updates are not as simple-to-implement as grid search

• Upside: empirically it has been demonstrated to **get better results in fewer experiments**
  • Compared with grid search and random search

• Pretty widely used method
  • Lots of research opportunities here.
A related method: DFO

• **Derivative-free optimization**

• Also called **zeroth-order optimization**

• These methods optimize a function using only evaluations, no derivatives

• Ideal for use with metaparameter optimization
  • Also ideal for **reinforcement learning**
The opposite of DFO
Gradient-based optimization

• These strategies say: “I’m doing SGD to learn, I may as well use it to optimize my hyperparameters.”

• When we can efficiently differentiate with respect to the hyperparameters, this strategy actually works pretty well.

• But generally, we can’t do it.
Methods that Look at the Data

• Many methods look at curvature/variance info to decide how to set hyperparameters, and update their settings throughout the algorithm.

• Example: ADAGRAD

• Example: Adam
  • Which you will be reading in a few weeks.
Evaluating the Hyperparameter Optimization
How to evaluate the hyperparameters?

• Unlike the model parameters, we’re not given a loss function

• Can’t we just use the training loss?

• **Not always**: we don’t want to overfit the hyperparameters
  • Especially not when they are things that affect the model
Cross-Validation

• Partition part of the available data to create an validation dataset that we don’t use for training.

• Then use that set to evaluate the hyperparameters.

• Typically, multiple rounds of cross-validation are performed using different partitions
  • Can get a very good sense of how good the hyperparameters are
  • But at a significant computational cost!
Evaluating the System Cost

• In practice we **don’t just care about the statistics**
  • Not just about the accuracy after a fixed number of iterations

• We care about wall-clock **time**, and we care about **energy**
  • How much did solving this problem **actually cost**?

• The parameters we chose can affect these systems properties
  • As we saw with our SVRG demo!

• Need to **include systems cost as part of the metric**!
Hardware efficiency

• How long does an iteration take, on average?

• Hardware efficiency measures the systems cost of doing a single update.

• Key point: many hyperparameters do not affect hardware efficiency
  • Which ones?

• Which hyperparameters do affect hardware efficiency?
Statistical Efficiency

• How many iterations do we need to get to a specified level of accuracy?

• Statistical efficiency measures how many updates we need to get an answer of the quality that we want.

• Which hyperparameters affect statistical efficiency?
  • And which ones don’t?
Total performance

• Total cost of running the algorithm is:

  HARDWARE EFFICIENCY x STATISTICAL EFFICIENCY

• We can estimate these quantities separately, then use their product to evaluate our hyperparameters.

• For example, we can use theory to evaluate statistical efficiency and a hardware model to evaluate hardware efficiency.
Benefits of Looking at Both

• Looking at both statistical and hardware efficiency together has some important benefits!

• Many times the optimal parameter settings are different than if you set the parameters to optimize hardware efficiency or statistical efficiency individually.

• There’s a lot of open research opportunities here!
Recent example: YellowFin Tuner

- System that among other things tunes the **momentum**
  - As well as using **asynchronous parallelism**, which we’ll talk about later.

Figure 1 from “**YellowFin and the Art of Momentum Tuning**”
Questions?

• Upcoming things
  • Fall break next Monday — no lecture
  • Paper review #4 due Today
  • Paper Presentation #5 on Wednesday — read paper before class
  • Paper Presentation #6 the following Wednesday