Adaptive Learning Rates, Inference, and Algorithms other than SGD

CS6787 Lecture 8 — Fall 2019

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.

Intuition: Polyak's step length

- Approximate the objective with a linear approximation at the current iterate.

$$\hat{f}(w) = f(w_k) + (w - w_k)^T \nabla f(w_k)$$

- Choose the step size that makes the approximation equal to the known optimal value.

$$\begin{aligned} f^* = \hat{f}(w_{k+1}) &= \hat{f}(w_k - \alpha \nabla f(w_k)) \\ &= f(w_k) - \alpha \|\nabla f(w_k)\|^2 \end{aligned} \quad \Rightarrow \alpha = \frac{f(w_k) - f^*}{\|\nabla f(w_k)\|^2}$$

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 on full objective is **expensive relative to SGD update.**

Adaptive methods for SGD

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

- You'll see Adam in one of this Wednesday's papers

AdaGrad

Adaptive gradient descent

Per-parameter adaptive learning rate schemes

- Main idea: set the **learning rate per-parameter** dynamically at each iteration based on observed statistics of the past gradients.

$$(w_{t+1})_j = (w_t)_j - \alpha_{j,t} \left(\nabla f(w_t; x_t)\right)_j$$

  - Where the step size now depends on the parameter index **j**
  - Corresponds to a multiplication of the gradient by a diagonal scaling matrix.

- There are many different schemes in this class

AdaGrad: One of the first adaptive methods

- **AdaGrad:** Adaptive subgradient methods for online learning and stochastic optimization
  - J Duchi, E Hazan, Y Singer
  - Journal of Machine Learning Research, 2011

- High level approach: can use **history of sampled gradients** to choose the step size for the next SGD step to be inversely proportional to the usual magnitude of gradient steps in that direction
  - On a per-parameter basis.

AdaGrad

Algorithm 1 AdaGrad

input: learning rate factor η, initial parameters w_0

initialize $t \leftarrow 0$

loop

sample a stochastic gradient $g_t \leftarrow \nabla f(w_t; x_t)$

update model: for all $j \in \{1, \ldots, d\}$

$$(w_{t+1})_j \leftarrow (w_t)_j - \frac{\eta}{\sqrt{\sum_{k=0}^{t} (g_t)_j^2 + \epsilon}} \cdot g_j$$

$t \leftarrow t + 1$

end loop

Can think of this as like the norm of the gradients in the jth parameter.

Memory-efficient implementation of AdaGrad

Algorithm 1 AdaGrad

input: learning rate factor η, initial parameters $w_0 \in \mathbb{R}^d$, small number ϵ
initialize $t \leftarrow 0$
initialize $r \leftarrow 0 \in \mathbb{R}^d$
loop
sample a stochastic gradient $g_t \leftarrow \nabla f(w_t; x_t)$
accumulate second moment estimate $r_j \leftarrow r_j + (g_t)_j^2$ for all $j \in \{1, \ldots, d\}$
update model: for all $j \in \{1, \ldots, d\}$

$$(w_{t+1})_j \leftarrow (w_t)_j - \frac{\eta}{\sqrt{r_j} + \epsilon} \cdot g_j$$

$t \leftarrow t + 1$
end loop

Important thing to notice: step size is monotonically decreasing!

Demo

AdaGrad for Non-convex Optimization

- **What problems might arise when using AdaGrad for non-convex optimization?**
  - Think about the step size always decreasing. Could this cause a problem?

- **If you do think of a problem that might arise, how could you change AdaGrad to fix it?**

RMSProp

Algorithm 1 RMSProp

input: learning rate factor η, initial parameters $w_0 \in \mathbb{R}^d$,

initialize $t \leftarrow 0$

initialize $r \leftarrow 0 \in \mathbb{R}^d$

loop

sample a stochastic gradient $g_t \leftarrow \nabla f(w_t; x_t)$

accumulate second moment estimate $r_j \leftarrow \rho \cdot r_j + (1-\rho)\,(g_t)_j^2$ for all $j \in \{1, \ldots, d\}$

update model: for all $j \in \{1, \ldots, d\}$

$$(w_{t+1})_j \leftarrow (w_t)_j - \frac{\eta}{\sqrt{r_j} + \epsilon} \cdot g_j$$

$t \leftarrow t + 1$

end loop

Just replaces the gradient accumulation of AdaGrad with an exponential moving average.

A systems perspective

- **What is the computational cost of AdaGrad and RMSProp?**
  - **How much additional memory is required compared to baseline SGD?**
  - **How much additional compute is used?**

Adaptive methods, summed up

- Generally useful when we can expect there to be **different scales for different parameters**
  - But can even work well when that doesn't happen, as we saw in the demo.

- Very commonly used class of methods for training ML models.

- We'll see more of this when we study **Adam** on Wednesday
  - Adam is basically RMSProp + Momentum.

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 Algorithms

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

- Just need to run the forward pass of the network.
  - A bunch of matrix multiplies and non-linear units.

- Unlike backpropagation for learning, here we do not need to keep the activations around for later processing.

- This makes inference a much simpler task than learning.
  - Although it can still be costly — it's a lot of linear algebra to do.

Inference on neural networks (continued)

- 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.

Metrics for Inference

- Important metric: **throughput**
  - **How many examples** can we classify in some amount of time
- Important metric: **latency**
  - **How long** does it take to get a prediction for a single example
- Important metric: **model size**
  - **How much memory** do we need to store/transmit the model for prediction
- Important metric: **energy use**
  - **How much energy** do we use to produce each prediction
- **What are examples where we might care about each metric?**

Improving the performance of inference

We can use many of the methods we've already discussed!

Altering the batch size

- Just like with learning, we can **make predictions in batches**
- Increasing the batch size helps **improve parallelism**
  - Provides more work to parallelize and an additional dimension for parallelization
  - This improves **throughput**
- But increasing the batch size can make us do more work before we can return an answer for any individual example
  - Can negatively affect **latency**

Demo

Compression

- Find an **easier-to-compute network** with similar accuracy
  - Or find a network with **smaller model size**, depending on the goal

- **Many techniques** for doing this
  - We'll talk about this later in the semester when we come back to it

- Usually involve some sort of **fine-tuning**
  - Apply a lossy compression operation, then retrain the model to improve accuracy

Efficient architectures

- Some neural network architectures are **designed to be efficient at inference time**
  - Examples: MobileNet, ShuffleNet, CirCNN
- These networks are often based on sparsely connected neurons
  - This limits the number of weights which makes models smaller and easier to run inference on
- To be efficient, we can just **train one of these networks in the first place** for our application.

Re-use of computation

- For video and time-series data, there is a lot of **redundant information** from one frame to the next.

- We can try to **re-use some of the computation** from previous frames.

- This is less popular than some of the other approaches here, because it is not really general.

The last resort for speeding up DNN inference

- **Train another, faster type of model** that is not a deep neural network
  - For some real-time applications, you can't always use a DNN

- If you can get away with **a linear model**, that's almost always much faster.

- Also, **decision trees** tend to be quite fast for inference.

Where do we run inference?

The hardware that underlies the systems side of inference

Inference in the cloud

- Most inference today is run on **cloud platforms**
- The cloud supports **large amounts of compute**
  - And makes it easy to access it and make it reliable
- This is a good place to put AI that needs to think about data
- For interactive models, **latency** is critical

Inference on edge devices

- Inference can run on your **laptop or smartphone**
    - Here, the size of the model becomes more of an issue
    - Limited smartphone memory
- This is good for **user privacy and security**
    - But not as good for companies that want to keep their models private
- Also can be used to deploy **personalized models**

Inference on sensors

- Sometimes we want **inference right at the source**
  - On the sensor where data is collected

- Example: a surveillance camera taking video
  - Don't want to stream the video to the cloud, especially if most of it is not interesting.

- **Energy use** is very important here.

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)
  - Infer by simulating a Markov chain — a random process — that we can prove will converge to the distribution we want to sample from over time
  - Asymptotically exact, but approximate for any finite execution time
- Variational inference
  - Infer by solving an optimization problem that models the target distribution as a member of a tractable family of distributions.
  - Can use many of the same techniques for speedup we have discussed in class.
  - Approximate method, since the class may not contain the real distribution.

Examples of Markov Chain Monte Carlo Methods

- Gradient-based methods
  - Stochastic gradient Langevin dynamics
  - Hamiltonian Monte Carlo
  - Stochastic gradient Hamiltonian Monte Carlo

- Non-gradient-based methods
  - **Gibbs sampling**
  - Metropolis-Hastings

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.

lo…

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

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

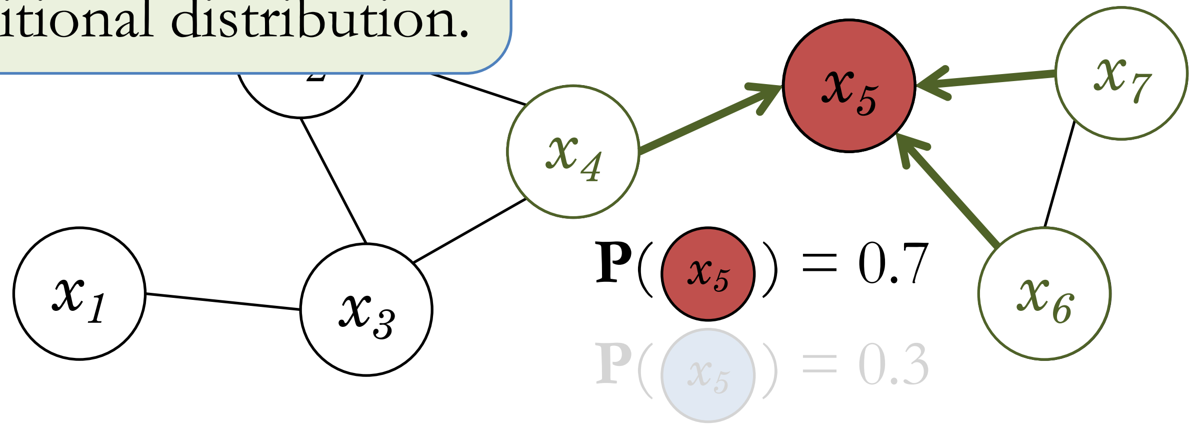
out… x

e…

Compute its conditional distribution given the other variables.

Output the current state as a sample.

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