Gradient Descent

ECE57000: Artificial Intelligence
David I. Inouye
Tuesday, September 15, 2020
Most AI/ML optimizations must be numerically optimized

▶ One common algorithm is **gradient descent**
  ▶ Primary algorithm for deep learning
  ▶ Works in very high dimensions

▶ Other optimization algorithms
  ▶ Expectation Maximization (alternating optimization)
  ▶ Sampling-based optimization (MCMC/Gibb)
  ▶ Greedy optimization (e.g., decision trees)
Gradient descent is like taking steps down the steepest descent into a valley

Vanilla gradient descent (GD) has simple form

1. Start at random parameter, e.g., $\theta^0 \sim \mathcal{N}(0, 1)$

2. Iteratively update parameter via negative gradient of loss function ($\eta_t$ is step size or learning rate)

$$
\theta^{t+1} = \theta^t - \eta_t \nabla_{\theta} \mathcal{L}(\theta^t)
$$

$\eta_t$ is learning rate (or step size)
Stochastic gradient descent (SGD) merely uses one sample in the gradient calculation

- The loss function can usually be split into a summation of losses $\ell(\theta; x_i)$ for each sample $x_i$:
  \[ \mathcal{L}(\theta; \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \ell(\theta; x_i) \]
- SGD approximates the full gradient by the gradient of a single sample
  \[ \nabla_{\theta} \mathcal{L}(\theta^t; \mathcal{D}) \approx \nabla_{\theta} \ell(\theta^t; x_i) \]
  - Theoretically, $\mathbb{E}_i[\nabla_{\theta} \ell(\theta^t; x_i)] = \nabla_{\theta} \mathcal{L}(\theta^t; \mathcal{D})$
- Loop through all $x_i \in \mathcal{D}$
  \[ \theta^{t+1} = \theta^t - \eta_t \nabla_{\theta} \ell(\theta^t; x_i) \]
- One pass through dataset
  - GD: 1 large update with $O(n)$ cost
  - SGD: $n$ smaller updates with $O(1)$ cost each
Stochastic gradient descent (SGD) merely uses one sample in the gradient calculation.

Full gradient is average over single sample gradients. This is why it is “stochastic”.

Mini-batch SGD (or just SGD) uses a small batch of samples in the gradient calculation

- **Mini-batch SGD** approximates the full gradient by the gradient of a batch of samples
  - Sample mini-batch
    
    \[
    \theta^{t+1} = \theta^t - \eta_t \sum_{k=1}^{b} \frac{1}{b} \nabla_{\theta} \ell(\theta^t; x_k)
    \]

- One pass through dataset
  - GD: 1 large update
  - SGD: \(n\) smaller updates
  - Mini-batch SGD: \(\frac{n}{b}\) medium-size updates
Gradient descent demo
for simplified logistic regression
Learning rate / step size is critical for convergence and correctness of algorithm

- If learning rate is **too high**, the algorithm could diverge.
  - Diverge means to get farther away from the solution.
- If learning rate **too low**, the algorithm could take a very long time to converge.

https://www.jeremyjordan.me/nn-learning-rate/
Adaptive learning rates may help *(but not always)*

- Decreasing step size, $\eta_t = \frac{1}{t}$
  - Intuition: Approaches 0 but can cover an infinite distance since $\lim_{a \to \infty} \sum_{t=1}^{a} \frac{1}{t} = \infty$

- ADAM – Adaptive Moment Estimation

- See [https://pytorch.org/docs/stable/optim.html](https://pytorch.org/docs/stable/optim.html) for more options
Parameter initialization can be important if non-convex or step size incorrect

- If convex function, initial parameter $\theta^0$ will not affect final optimization result $\hat{\theta} = \arg\min_\theta \mathcal{L}(\theta)$.
  - Yay! (Assuming appropriate step size.)

- If non-convex, starting position WILL affect final converged $\hat{\theta}$.
  - Sad day. (But sometimes it’s not too bad in practice.)