

STOR566: Introduction to Deep Learning

Lecture 4: Optimization

Yao Li
UNC Chapel Hill

Aug 25, 2022

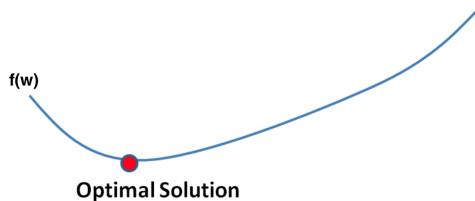
Materials are from *Learning from data* (Caltech) and *Deep Learning* (UCLA)

Optimization

- Goal: find the minimizer of a function

$$\min_{\mathbf{w}} f(\mathbf{w})$$

- Machine learning algorithm: find the hypothesis that **minimizes training error**



Gradient descent

Gradient Descent

- Gradient descent: repeatedly do

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \alpha \nabla f(\mathbf{w}^t)$$

$\alpha > 0$ is the **step size**

Gradient Descent

- Gradient descent: repeatedly do

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \alpha \nabla f(\mathbf{w}^t)$$

$\alpha > 0$ is the **step size**

- Generate the sequence $\mathbf{w}^1, \mathbf{w}^2, \dots$

converge when α is sufficiently small

Gradient Descent

- Gradient descent: repeatedly do

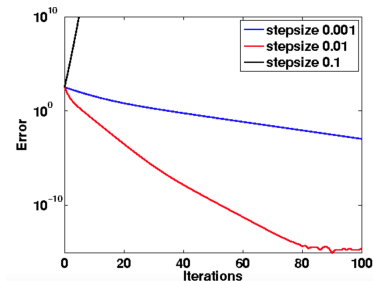
$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \alpha \nabla f(\mathbf{w}^t)$$

$\alpha > 0$ is the **step size**

- Generate the sequence $\mathbf{w}^1, \mathbf{w}^2, \dots$

converge when α is sufficiently small

- Step size **too large** \Rightarrow **diverge**; **too small** \Rightarrow **slow convergence**



Convergence

- f : convex, twice-differentiable, L -Lipschitz continuous gradient
($\nabla^2 f(\mathbf{x}) \preceq LI$ for all \mathbf{x})
- **Theorem:** gradient descent converges if $\alpha < \frac{2}{L}$
- Optimal: $\alpha < \frac{1}{L}$

Line Search

- In practice, we do not know $L \dots$
need to tune step size when running gradient descent

Line Search

- In practice, we do not know $L \dots$
 - need to tune step size when running gradient descent
- Line Search: Select step size automatically (for gradient descent)

Line Search

- The **back-tracking** line search:
 - Start from some **large α_0**
 - Try $\alpha = \alpha_0, \frac{\alpha_0}{2}, \frac{\alpha_0}{4}, \dots$
Stop when α satisfies some **sufficient decrease condition**

Line Search

- The **back-tracking** line search:
 - Start from some **large** α_0
 - Try $\alpha = \alpha_0, \frac{\alpha_0}{2}, \frac{\alpha_0}{4}, \dots$
 - Stop when α satisfies some **sufficient decrease condition**
 - A simple condition: $f(\mathbf{w} + \alpha \mathbf{d}) < f(\mathbf{w})$

Line Search

- The **back-tracking** line search:
 - Start from some **large α_0**
 - Try $\alpha = \alpha_0, \frac{\alpha_0}{2}, \frac{\alpha_0}{4}, \dots$
 - Stop when α satisfies some **sufficient decrease condition**
 - A simple condition: $f(\mathbf{w} + \alpha \mathbf{d}) < f(\mathbf{w})$
 - A (provable) sufficient decrease condition:

$$f(\mathbf{w} + \alpha \mathbf{d}) \leq f(\mathbf{w}) + \sigma \alpha \nabla f(\mathbf{w})^T \mathbf{d}$$

for a constant $\sigma \in (0, 1)$

Line Search

gradient descent with backtracking line search

- Initialize the weights \mathbf{w}_0
- For $t = 1, 2, \dots$
 - Compute the gradient

$$\mathbf{d} = -\nabla f(\mathbf{w})$$

- For $\alpha = \alpha_0, \alpha_0/2, \alpha_0/4, \dots$
Break if $f(\mathbf{w} + \alpha\mathbf{d}) \leq f(\mathbf{w}) + \sigma\alpha\nabla f(\mathbf{w})^T\mathbf{d}$
 - Update $\mathbf{w} \leftarrow \mathbf{w} + \alpha\mathbf{d}$
- Return the final solution \mathbf{w}

Stochastic Gradient descent

Large-scale Problems

- Machine learning: usually minimizing the training loss

$$\min_{\mathbf{w}} \left\{ \frac{1}{N} \sum_{n=1}^N \ell(\mathbf{w}^T \mathbf{x}_n, y_n) \right\} := f(\mathbf{w}) \text{ (linear model)}$$

$$\min_{\mathbf{w}} \left\{ \frac{1}{N} \sum_{n=1}^N \ell(h_{\mathbf{w}}(\mathbf{x}_n), y_n) \right\} := f(\mathbf{w}) \text{ (general hypothesis)}$$

ℓ : loss function (e.g., $\ell(a, b) = (a - b)^2$)

- Gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \underbrace{\nabla f(\mathbf{w})}_{\text{Main computation}}$$

Large-scale Problems

- Machine learning: usually minimizing the training loss

$$\min_{\mathbf{w}} \left\{ \frac{1}{N} \sum_{n=1}^N \ell(\mathbf{w}^T \mathbf{x}_n, y_n) \right\} := f(\mathbf{w}) \text{ (linear model)}$$

$$\min_{\mathbf{w}} \left\{ \frac{1}{N} \sum_{n=1}^N \ell(h_{\mathbf{w}}(\mathbf{x}_n), y_n) \right\} := f(\mathbf{w}) \text{ (general hypothesis)}$$

ℓ : loss function (e.g., $\ell(a, b) = (a - b)^2$)

- Gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \underbrace{\nabla f(\mathbf{w})}_{\text{Main computation}}$$

- In general, $f(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N f_n(\mathbf{w})$,
each $f_n(\mathbf{w})$ only depends on (\mathbf{x}_n, y_n)

Stochastic gradient

- Gradient:

$$\nabla f(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \nabla f_n(\mathbf{w})$$

- Each gradient computation needs to go through **all training samples**
slow when millions of samples
- Faster way to compute “**approximate gradient**”?

Stochastic gradient

- Gradient:

$$\nabla f(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \nabla f_n(\mathbf{w})$$

- Each gradient computation needs to go through **all training samples** slow when millions of samples
- Faster way to compute “**approximate gradient**”?
- Use **stochastic sampling**:
 - Sample a small subset $B \subseteq \{1, \dots, N\}$
 - Estimated gradient

$$\nabla f(\mathbf{w}) \approx \frac{1}{|B|} \sum_{n \in B} \nabla f_n(\mathbf{w})$$

$|B|$: batch size

Stochastic gradient descent

Stochastic Gradient Descent (SGD)

- Input: training data $\{\mathbf{x}_n, y_n\}_{n=1}^N$
- Initialize \mathbf{w} (zero or random)
- For $t = 1, 2, \dots$
 - Sample a **small batch** $B \subseteq \{1, \dots, N\}$
 - Update parameter

$$\mathbf{w} \leftarrow \mathbf{w} - \eta^t \frac{1}{|B|} \sum_{n \in B} \nabla f_n(\mathbf{w})$$

Stochastic gradient descent

Stochastic Gradient Descent (SGD)

- Input: training data $\{\mathbf{x}_n, y_n\}_{n=1}^N$
- Initialize \mathbf{w} (zero or random)
- For $t = 1, 2, \dots$
 - Sample a **small batch** $B \subseteq \{1, \dots, N\}$
 - Update parameter

$$\mathbf{w} \leftarrow \mathbf{w} - \eta^t \frac{1}{|B|} \sum_{n \in B} \nabla f_n(\mathbf{w})$$

Why SGD works?

Logistic Regression by SGD

- Logistic regression:

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^N \underbrace{\log(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n})}_{f_n(\mathbf{w})}$$

SGD for Logistic Regression

- Input: training data $\{\mathbf{x}_n, y_n\}_{n=1}^N$
- Initialize \mathbf{w} (zero or random)
- For $t = 1, 2, \dots$
 - Sample a batch $B \subseteq \{1, \dots, N\}$
 - Update parameter

$$\mathbf{w} \leftarrow \mathbf{w} - \eta^t \frac{1}{|B|} \sum_{i \in B} \underbrace{\frac{-y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}}}_{\nabla f_n(\mathbf{w})}$$

Stochastic gradient descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?

Stochastic gradient descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?
- If \mathbf{w}^* is the minimizer, $\nabla f(\mathbf{w}^*) = \frac{1}{N} \sum_{n=1}^N \nabla f_n(\mathbf{w}^*) = 0$,

but $\frac{1}{|B|} \sum_{n \in B} \nabla f_n(\mathbf{w}^*) \neq 0$ if B is a subset

Stochastic gradient descent, step size

- To make SGD converge:

Step size should decrease to 0

$$\eta^t \rightarrow 0$$

Usually with polynomial rate: $\eta^t \approx t^{-a}$ with constant a

- pros:

cheaper computation per iteration

- cons:

less stable, slower final convergence

Momentum

- Gradient descent: only using current gradient (local information)
- Momentum: use **previous gradient information**

Momentum

- Gradient descent: only using current gradient (local information)
- Momentum: use **previous gradient information**
- The momentum update rule:

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1 - \beta) \nabla f(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{v}_t$$

$\beta \in [0, 1)$: discount factors, α : step size

Momentum

- Gradient descent: only using current gradient (local information)
- Momentum: use **previous gradient information**
- The momentum update rule:

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1 - \beta) \nabla f(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{v}_t$$

$\beta \in [0, 1)$: discount factors, α : step size

- Equivalent to using average of gradients:

$$\mathbf{v}_t = (1 - \beta) \nabla f(\mathbf{w}_t) + \beta(1 - \beta) \nabla f(\mathbf{w}_{t-1}) + \beta^2(1 - \beta) \nabla f(\mathbf{w}_{t-2}) + \dots$$

Momentum stochastic gradient descent

Optimizing $f(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{w})$

Momentum stochastic gradient descent

- Initialize $\mathbf{w}_0, \mathbf{v}_0 = 0$
- For $t = 1, 2, \dots$
 - Sample an $i \in \{1, \dots, N\}$
 - Compute $\mathbf{v}_t \leftarrow \beta \mathbf{v}_{t-1} + (1 - \beta) \nabla f_i(\mathbf{w}_t)$
 - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \mathbf{v}_t$

α : learning rate

β : discount factor ($\beta = 0$ means no momentum)

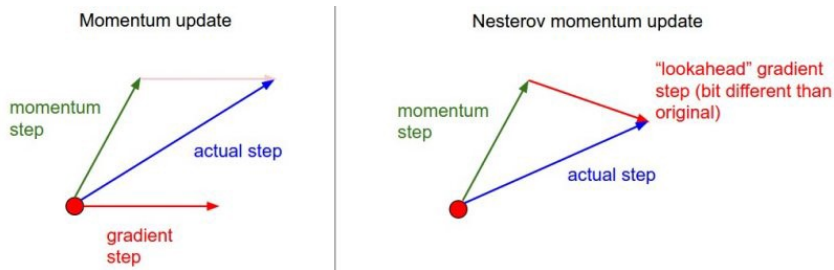
Why it works?

Nesterov accelerated gradient

- Using the “look-ahead” gradient

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1 - \beta) \nabla f(\mathbf{w}_t - \beta \mathbf{v}_{t-1})$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{v}_t$$



(Figure from <https://towardsdatascience.com>)

Adagrad: Adaptive updates (2010)

- SGD update: same step size for all variables
- Adaptive algorithms: each dimension can have a different step size

Adagrad: Adaptive updates (2010)

- SGD update: same step size for all variables
- Adaptive algorithms: each dimension can have a different step size

Adagrad

- Initialize \mathbf{w}^0
- For $t = 1, 2, \dots$
 - Sample an $i \in \{1, \dots, N\}$
 - Compute $\mathbf{g}^t \leftarrow \nabla f_i(\mathbf{w}^t)$
 - $G_j^t \leftarrow G_j^{t-1} + (g_j^t)^2$ for all $j = 1, \dots, d$
 - Update $w_j^{t+1} \leftarrow w_j^t - \frac{\eta}{\sqrt{G_j^t + \epsilon}} g_j^t$

η : step size (constant)

ϵ : small constant to avoid division by 0

- Adam: Momentum + Adaptive updates (2015)

Conclusions

- Gradient descent
- Stochastic gradient descent
- Variants

Questions?