CS260: Machine Learning Algorithms

Lecture 4: Stochastic Gradient Descent

Cho-Jui Hsieh
UCLA

Jan 16, 2019

Large-scale Problems

- Machine learning: usually minimizing the training loss

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

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

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

- Gradient descent:

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

Large-scale Problems

- Machine learning: usually minimizing the training loss

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

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

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

- Gradient descent:

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

- In general, $f(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} f_n(\boldsymbol{w})$,

  each $f_n(\boldsymbol{w})$ only depends on $(\boldsymbol{x}_n, y_n)$

Stochastic gradient

- Gradient:

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

- Each gradient computation needs to go through all training samples

  slow when millions of samples

- Faster way to compute "approximate gradient"?

Stochastic gradient

- Gradient:

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

- 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 $B \subseteq \{1, \cdots, N\}$
  - Estimated gradient

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

  $|B|$: batch size

Stochastic gradient descent

Stochastic Gradient Descent (SGD)

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

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

Stochastic gradient descent

Stochastic Gradient Descent (SGD)

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

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

Extreme case: $|B| = 1 \Rightarrow$ **Sample one training data at a time**

Logistic Regression by SGD

- Logistic regression:

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

SGD for Logistic Regression

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

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

Why SGD works?

- Stochastic gradient is an unbiased estimator of full gradient:

$$
\begin{aligned}
E[\frac{1}{|B|}\sum_{n\in B}\nabla f_n(\boldsymbol{w})] &= \frac{1}{N}\sum_{n=1}^{N}\nabla f_n(\boldsymbol{w}) \\
&= \nabla f(\boldsymbol{w})
\end{aligned}
$$

Why SGD works?

- Stochastic gradient is an unbiased estimator of full gradient:

$$E[\frac{1}{|B|}\sum_{n\in B}\nabla f_n(\boldsymbol{w})] = \frac{1}{N}\sum_{n=1}^{N}\nabla f_n(\boldsymbol{w})$$
$$= \nabla f(\boldsymbol{w})$$

- Each iteration updated by

gradient + zero-mean noise

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?
- SGD with fixed step size cannot converge to global/local minimizers

Stochastic gradient descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?
- SGD with fixed step size cannot converge to global/local minimizers
- If $\boldsymbol{w}^*$ is the minimizer, $\nabla f(\boldsymbol{w}^*) = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(\boldsymbol{w}^*) = 0$,

Stochastic gradient descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?
- SGD with fixed step size cannot converge to global/local minimizers
- If $\boldsymbol{w}^*$ is the minimizer, $\nabla f(\boldsymbol{w}^*) = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(\boldsymbol{w}^*) = 0$,

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

Stochastic gradient descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?
- SGD with fixed step size cannot converge to global/local minimizers
- If $\boldsymbol{w}^*$ is the minimizer, $\nabla f(\boldsymbol{w}^*) = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(\boldsymbol{w}^*) = 0$,

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

(Even if we got minimizer, SGD will move away from it)

Stochastic gradient descent, step size

- To make SGD converge:

  Step size should decrease to 0

  $\eta^t \to 0$

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

Stochastic gradient descent vs Gradient descent

Stochastic gradient descent:

- pros:

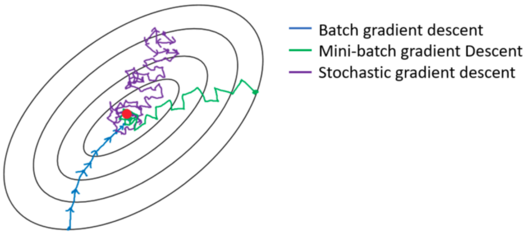
  cheaper computation per iteration

  faster convergence in the beginning

- cons:

  less stable, slower final convergence

  hard to tune step size



(Figure from https://medium.com/@ImadPhd/gradient-descent-algorithm-and-its-variants-10f652806a3)

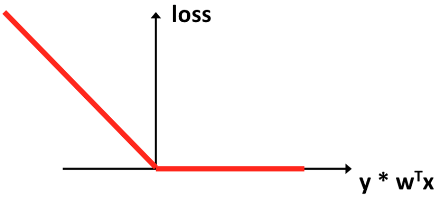
Revisit perceptron Learning Algorithm

- Given a classification data $\{\boldsymbol{x}_n, y_n\}_{n=1}^{N}$
- Learning a linear model:

$$\min_{\boldsymbol{w}} \frac{1}{N} \sum_{n=1}^{N} \ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n)$$

- Consider the loss:

$$\ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = \max(0, -y_n \boldsymbol{w}^T \boldsymbol{x}_n)$$



What's the gradient?

Revisit perceptron Learning Algorithm

$$\ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = \max(0, -y_n \boldsymbol{w}^T \boldsymbol{x}_n)$$

Consider two cases:

- Case I: $y_n \boldsymbol{w}^T \boldsymbol{x}_n > 0$ (prediction correct)
  - $\ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = 0$
  - $\frac{\partial}{\partial \boldsymbol{w}} \ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = 0$

Revisit perceptron Learning Algorithm

$$\ell(\boldsymbol{w}^T\boldsymbol{x}_n, y_n) = \max(0, -y_n\boldsymbol{w}^T\boldsymbol{x}_n)$$

Consider two cases:

- Case I: $y_n\boldsymbol{w}^T\boldsymbol{x}_n > 0$ (prediction correct)
  - $\ell(\boldsymbol{w}^T\boldsymbol{x}_n, y_n) = 0$
  - $\frac{\partial}{\partial \boldsymbol{w}}\ell(\boldsymbol{w}^T\boldsymbol{x}_n, y_n) = 0$
- Case II: $y_n\boldsymbol{w}^T\boldsymbol{x}_n < 0$ (prediction wrong)
  - $\ell(\boldsymbol{w}^T\boldsymbol{x}_n, y_n) = -y_n\boldsymbol{w}^T\boldsymbol{x}_n$
  - $\frac{\partial}{\partial \boldsymbol{w}}\ell(\boldsymbol{w}^T\boldsymbol{x}_n, y_n) = -y_n\boldsymbol{x}_n$

Revisit perceptron Learning Algorithm

$$\ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = \max(0, -y_n \boldsymbol{w}^T \boldsymbol{x}_n)$$

Consider two cases:

- Case I: $y_n \boldsymbol{w}^T \boldsymbol{x}_n > 0$ (prediction correct)
  - $\ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = 0$
  - $\frac{\partial}{\partial \boldsymbol{w}} \ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = 0$
- Case II: $y_n \boldsymbol{w}^T \boldsymbol{x}_n < 0$ (prediction wrong)
  - $\ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = -y_n \boldsymbol{w}^T \boldsymbol{x}_n$
  - $\frac{\partial}{\partial \boldsymbol{w}} \ell(\boldsymbol{w}^T \boldsymbol{x}_n, y_n) = -y_n \boldsymbol{x}_n$

SGD update rule: Sample an index n

$$\boldsymbol{w}^{t+1} \leftarrow \begin{cases} \boldsymbol{w}^t & \text{if } y_n \boldsymbol{w}^T \boldsymbol{x}_n \geq 0 \text{ (predict correct)} \\ \boldsymbol{w}^t + \eta^t y_n \boldsymbol{x}_n & \text{if } y_n \boldsymbol{w}^T \boldsymbol{x}_n < 0 \text{ (predict wrong)} \end{cases}$$

Equivalent to Perceptron Learning Algorithm when $\eta^t = 1$

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:

$$\begin{aligned} \boldsymbol{v}_t &= \beta \boldsymbol{v}_{t-1} + (1-\beta)\nabla f(\boldsymbol{w}_t) \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \alpha \boldsymbol{v}_t \end{aligned}$$

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

Momentum

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

$$\begin{aligned} \boldsymbol{v}_t &= \beta \boldsymbol{v}_{t-1} + (1-\beta)\nabla f(\boldsymbol{w}_t) \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \alpha \boldsymbol{v}_t \end{aligned}$$

  $\beta \in [0, 1)$: discount factors, α: step size
- Equivalent to using moving average of gradient:

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

Momentum

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

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

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

- Equivalent to using moving average of gradient:

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

- Another equivalent form:

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

Momentum gradient descent

Momentum gradient descent

- Initialize $\boldsymbol{w}_0, \boldsymbol{v}_0 = 0$
- For $t = 1, 2, \cdots$
  - Compute $\boldsymbol{v}_t \leftarrow \beta \boldsymbol{v}_{t-1} + (1-\beta)\nabla f(\boldsymbol{w}_t)$
  - Update $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \alpha \boldsymbol{v}_t$

α: learning rate

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

Momentum stochastic gradient descent

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

Momentum stochastic gradient descent

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

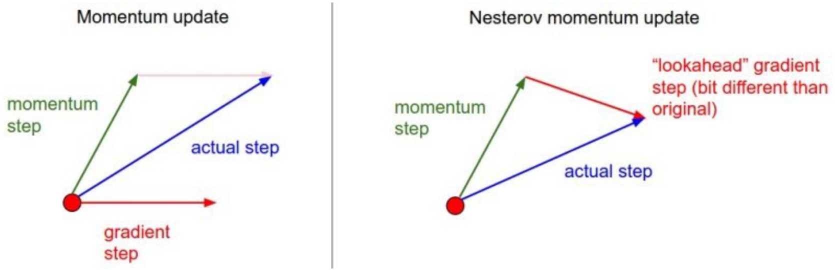
α: learning rate

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

Nesterov accelerated gradient

- Using the "look-ahead" gradient

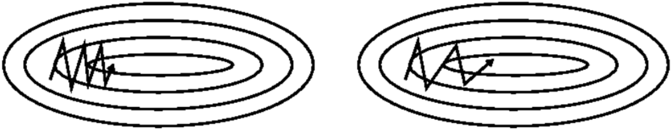
$$\boldsymbol{v}_t = \beta \boldsymbol{v}_{t-1} + \alpha \nabla f(\boldsymbol{w}_t - \beta \boldsymbol{v}_{t-1})$$
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \boldsymbol{v}_t$$



(Figure from `https://towardsdatascience.com`)

Why momentum works?

- Reduce variance of gradient estimator for SGD
- Even for gradient descent, it's able to speed up convergence in some cases:



Left—SGD without momentum, right— SGD with momentum. (Source: Genevieve B. Orr)

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 $\boldsymbol{w}_0$
- For $t = 1, 2, \cdots$
  - Sample an $i \in \{1, \cdots, N\}$
  - Compute $\boldsymbol{g}^t \leftarrow \nabla f_i(\boldsymbol{w}_t)$
  - $G_i^t \leftarrow G_i^{t-1} + (g_i^t)^2$
  - Update $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \frac{\eta}{\sqrt{G_i^t + \epsilon}} g_i^t$

η: step size (constant)

ϵ: small constant to avoid division by 0

Adagrad

- For each dimension i, we have observed T samples $g_i^1, \cdots, g_i^t$
- Standard deviation of g_i:

$$\sqrt{\frac{\sum_{t'}(g_i^{t'})^2}{t}} = \sqrt{\frac{(G_i^t)^2}{t}}$$

- Assume step size is $\eta/\sqrt{t}$, then the update becomes

$$w_i^{t+1} \leftarrow w_i^t - \frac{\eta}{\sqrt{t}} \frac{\sqrt{t}}{\sqrt{(G_i^t)^2}} g_i^t$$

Adam: Momentum + Adaptive updates (2015)

Adam

- Initialize $\boldsymbol{w}_0, \boldsymbol{m}_0 = 0, \boldsymbol{v}_0 = 0,$
- For $t = 1, 2, \cdots$
  - Sample an $i \in \{1, \cdots, N\}$
  - Compute $\boldsymbol{g}_t \leftarrow \nabla f_i(\boldsymbol{w}_t)$
  - $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$
  - $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  - $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$
  - $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$
  - Update $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_t - 1 - \alpha \cdot \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon)$

Conclusions

- Stochastic gradient descent
- Momentum & adaptive updates

Questions?