COMP5211: Machine Learning

Lecture 4

Minhao Cheng

Optimization

Gradient descent convergence rate

- Suppose f is convex and differentiable and its gradient is lipshcitz continuous, then if we run gradient for t iterations with a fixed step $\alpha \leq \frac{1}{L}$, it will yield a solution that satisfies:
  - $f(w^t) - f(w^*) \leq \frac{\|w^0 - w^*\|_2^2}{2\alpha t}$
- Proof：
  - $$\begin{aligned} f(w^{t+1}) &\leq f(w^t) + \nabla f(w^t)^T(w^{t+1} - w^t) + \frac{1}{2}\nabla^2 f(w^t)\|w^{t+1} - w^t\|^2 \\ &= f(w^t) + \nabla f(w^t)^T(w^{t+1} - w^t) + \frac{1}{2}L\|w^{t+1} - w^t\|^2 \\ &= f(w^t) + \nabla f(w^t)^T(w^t - \alpha\nabla f(w^t) - w^t) + \frac{1}{2}L\|w^t - \alpha\nabla f(w^t) - w^t\|^2 \\ &= f(x) - (1 - \frac{1}{2}L\alpha)\alpha\|\nabla f(x)\|_2^2 \leq f(w^t) - \frac{1}{2}\alpha\|\nabla f(x)\|_2^2 \end{aligned}$$
  - ...

Optimization
Large-scale problem

- Machine learning: usually minimizing the training loss:
  - $\min_w \{\frac{1}{N}\sum_{n=1}^{N} \ell(w^T x_n, y_n)\} := f(w)$ (linear model)
  - $\min_w \{\frac{1}{N}\sum_{n=1}^{N} \ell(f_W(x_n), y_n)\} := f(w)$ (general hypothesis)
  - ℓ: loss function (e.g., $\ell(a,b) = (a-b)^2$)
- Gradient descent:
  - $w \leftarrow w - \eta \underbrace{\nabla f(w)}_{\text{Main computation}}$

Optimization
Large-scale problem

- Machine learning: usually minimizing the training loss:
  - $\min_w \{\frac{1}{N}\sum_{n=1}^{N} \ell(w^T x_n, y_n)\} := f(w)$ (linear model)
  - $\min_w \{\frac{1}{N}\sum_{n=1}^{N} \ell(f_W(x_n), y_n)\} := f(w)$ (general hypothesis)
  - ℓ: loss function (e.g., $\ell(a,b) = (a-b)^2$)
- Gradient descent:
  - $w \leftarrow w - \eta \underbrace{\nabla f(w)}_{\text{Main computation}}$
- In general, $f(w) = \frac{1}{N}\sum_{n=1}^{N} f_n(w)$,
  - Each $f_n(w)$ only depends on (x_n, y_n)

Optimization

Stochastic gradient

- Gradient: $\nabla f(w) = \frac{1}{N} \sum_{n=1}^{N} \nabla f_n(w),$
- Each gradient computation needs to go through all training samples
  - Slow when millions of samples
- Faster way to compare “approximate gradient”?

Optimization

Stochastic gradient

- Gradient: $\nabla f(w) = \frac{1}{N} \sum_{n=1}^{N} \nabla f_n(w),$
- Each gradient computation needs to go through all training samples
  - Slow when millions of samples
- Faster way to compare "approximate gradient"?
- Use stochastic sampling:
  - Sample a small subset $B \subseteq \{1, \ldots, N\}$
  - Estimated gradient
    - $\nabla f(w) \approx \frac{1}{B} \sum_{n \in B} \nabla f_n(w)$
    - $|B|$: batch size

Optimization

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

Optimization

Logistic Regression by SGD

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

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

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

Optimization

Why SGD works?

- Stochastic gradient is an unbiased estimator of full gradient:
  - $$\mathbb{E}[\frac{1}{|B|}\sum_{n\in B}\nabla f_n(w)] = \frac{1}{N}\sum_{n=1}^{N}\nabla f_n(w) = \nabla f(w)$$

Optimization

Why SGD works?

- Stochastic gradient is an unbiased estimator of full gradient:
  - $$\mathbb{E}[\frac{1}{|B|}\sum_{n\in B}\nabla f_n(w)] = \frac{1}{N}\sum_{n=1}^{N}\nabla f_n(w) = \nabla f(w)$$
- Each iteration updated by
  - Gradient + zero-mean noise

Optimization

Stochastic gradient descent

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

Optimization

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

Optimization

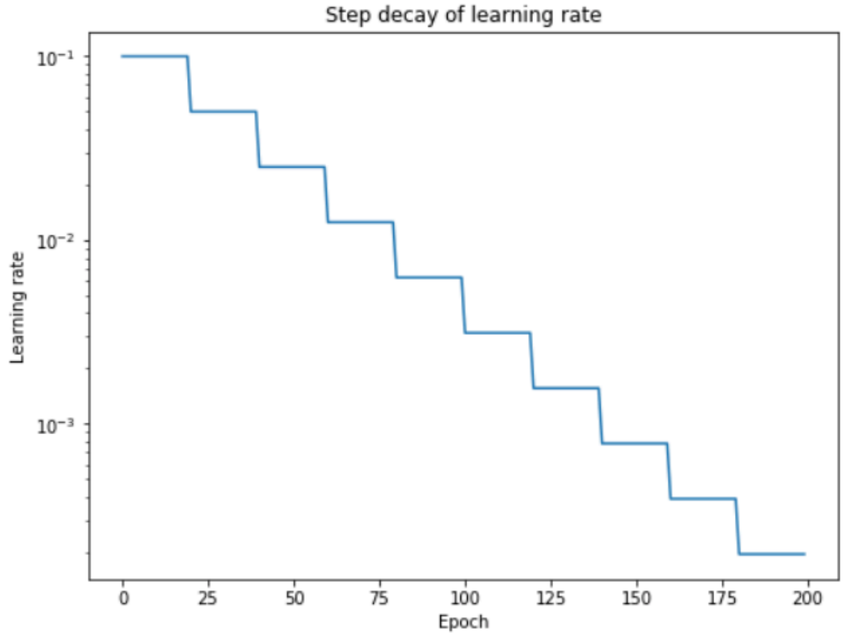
Stochastic gradient descent

- 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 w^* is the minimizer, $\nabla f(w^*) = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(w^*) = 0,$
  - But $\frac{1}{|B|}\sum_{n\in B} \nabla f_n(w) \neq 0$ if B is a subset
  - (Even if we got minimizer, SGD will move away from it)

Optimization

Stochastic gradient descent: step size

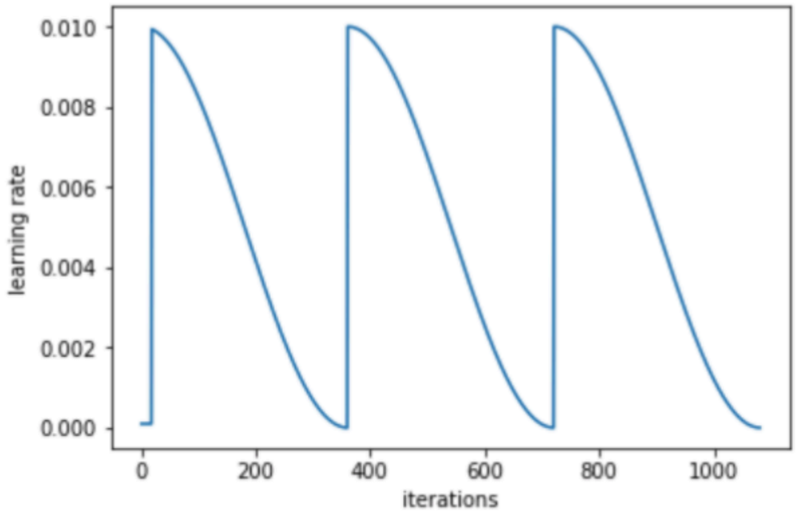
- To make SGD converge:
  - Step size should decrease to 0
  - $\eta^t \to 0$
  - Usually with polynomial rate $\eta^t \approx t^{-a}$ with constant a
- Step decay of learning rate

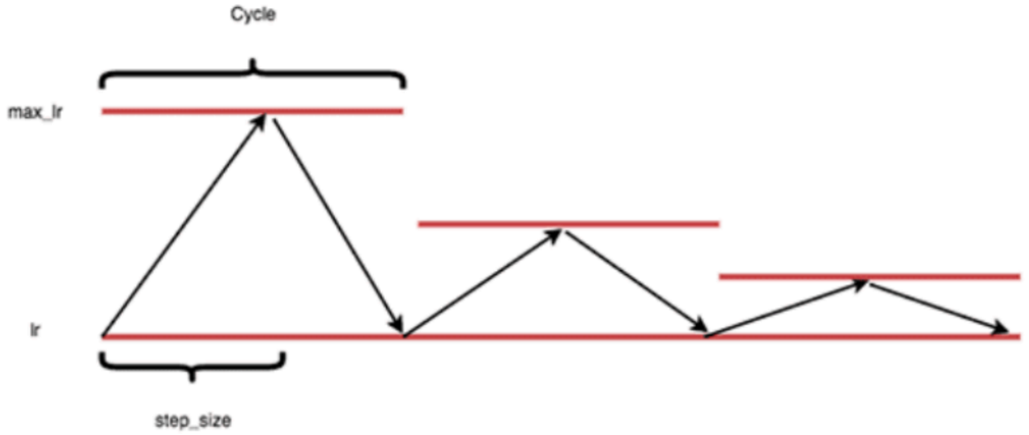


Optimization

learning rate scheduling

- Other cyclic learning rate scheduling

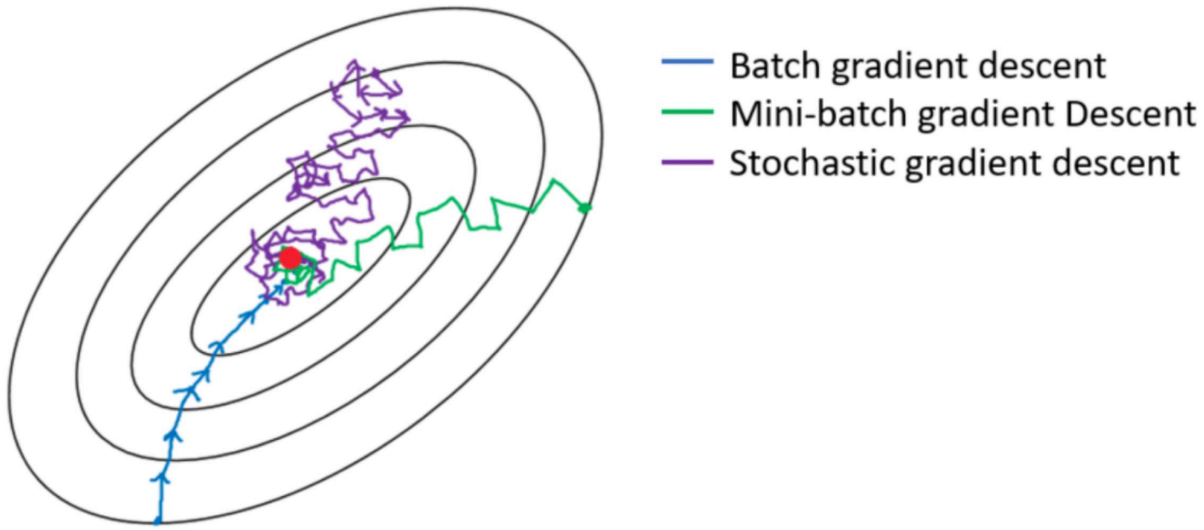




Optimization

Stochastic gradient descent vs Gradient descent

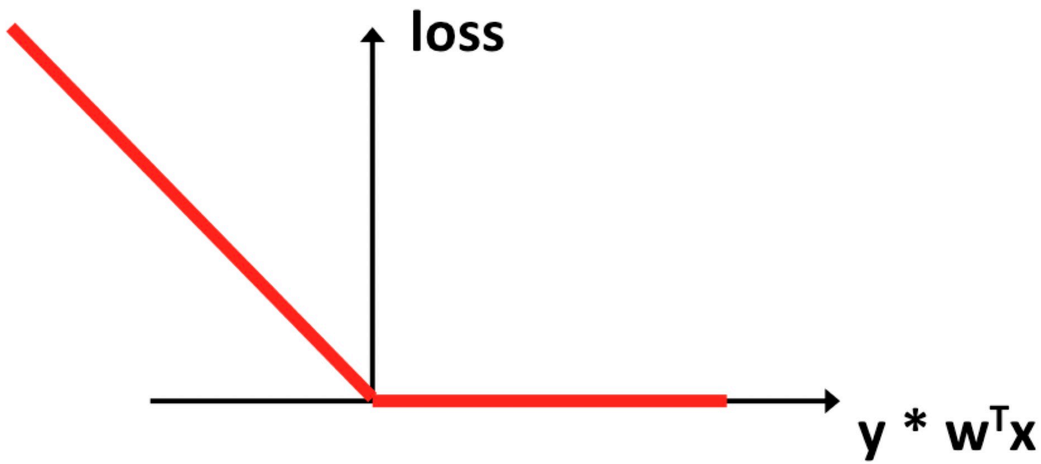
- Stochastic gradient descent:
  - Pros:
    - Cheaper computation per iteration
    - Faster convergence in the beginning
  - Cons:
    - Less stable, slower final convergence
    - Hard to tune step size



Optimization

Ex. Perception learning

- Given a classification data $\{x_n, y_n\}_{n=1}^N$
- Learning a linear model:
  - $\min_w \frac{1}{N} \sum_{n=1}^{N} \ell(w^T x_n, y_n)$
- Consider the loss:
  - $\ell(w^T x_n, y_n) = \max(0, -y_n w^T x_n)$
- What' the gradient?



Optimization

Ex. Perception learning

- $\ell(w^T x_n, y_n) = \max(0, -y_n w^T x_n)$
- Consider two cases:
  - Case I: $y_n w^T x_n > 0$ (predict correctly)
    - $\ell(w^T x_n, y_n) = 0$
    - $\frac{\partial}{\partial w} \ell(w^T x_n, y_n) = 0$

Optimization

Ex. Perception learning

- $\ell(w^T x_n, y_n) = \max(0, -y_n w^T x_n)$
- Consider two cases:
  - Case I: $y_n w^T x_n > 0$ (predict correctly)
    - $\ell(w^T x_n, y_n) = 0$
    - $\frac{\partial}{\partial w} \ell(w^T x_n, y_n) = 0$
  - Case II: $y_n w^T x_n < 0$ (predict wrongly)
    - $\ell(w^T x_n, y_n) = -y_n w^T x_n$
    - $\frac{\partial}{\partial w} \ell(w^T x_n, y_n) = -y_n x_n$

Optimization

Ex. Perception learning

- $\ell(w^T x_n, y_n) = \max(0, -y_n w^T x_n)$
- Consider two cases:
  - Case I: $y_n w^T x_n > 0$ (predict correctly)
    - $\ell(w^T x_n, y_n) = 0 \qquad \frac{\partial}{\partial w}\ell(w^T x_n, y_n) = 0$
  - Case II: $y_n w^T x_n < 0$ (predict wrongly)
    - $\ell(w^T x_n, y_n) = -y_n w^T x_n \qquad \frac{\partial}{\partial w}\ell(w^T x_n, y_n) = -y_n x_n$
  - SGD update rule: Sample an index n
    - $$w^{t+1} \begin{cases} w^t, & if\ y_n w^T x_n \geq 0 \quad \text{(predict correctly)} \\ w^t + \eta^t y_n x_n, & if\ y_n w^T x_n < 0 \quad \text{(predict wrongly)} \end{cases}$$

Optimization

Momentum

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

Optimization
Momentum

- Gradient descent: only using current gradient (local information)
- Momentum: use previous gradient information
- The momentum update rule:
  - $v_t = \beta v_{t-1} + (1-\beta)\nabla f(w_t)$
  - $w_{t+1} = w_t - \alpha v_t$
  - $\beta \in [0,1)$: discount factors, α: step size
- Equivalent to using moving average of gradient
  - $v_t = (1-\beta)\nabla f(w_t) + \beta(1-\beta)\nabla f(w_{t-1}) + \beta^2(1-\beta)\nabla f(w_{t-2}) + \dots$

Optimization
Momentum

- Gradient descent: only using current gradient (local information)
- Momentum: use previous gradient information
- The momentum update rule:
  - $v_t = \beta v_{t-1} + (1-\beta)\nabla f(w_t) \qquad w_{t+1} = w_t - \alpha v_t$
  - $\beta \in [0,1)$: discount factors, α: step size
- Equivalent to using moving average of gradient
  - $v_t = (1-\beta)\nabla f(w_t) + \beta(1-\beta)\nabla f(w_{t-1}) + \beta^2(1-\beta)\nabla f(w_{t-2}) + \dots$
- Another equivalent form:
  - $v_t = \beta v_{t-1} + \alpha \nabla f(w_t)$
  - $w_{t+1} = w_t - v_t$

Optimization

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)

Optimization

Momentum gradient descent

Momentum stochastic gradient descent

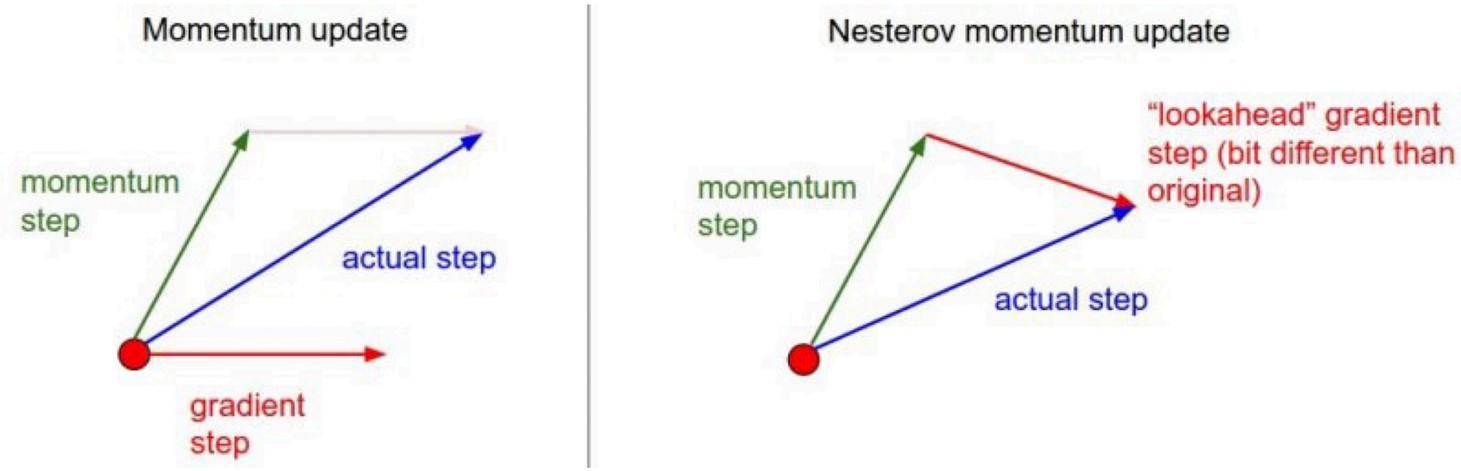
- Initialize $\boldsymbol{w}_0, \boldsymbol{v}_0 = 0$
- For $t = 1, 2, \cdots$
  - Sample an $i \in \{1, \cdots, N\}$
  - Compute $\boldsymbol{v}_t \leftarrow \beta \boldsymbol{v}_{t-1} + (1 - \beta) \nabla f_i(\boldsymbol{w}_t)$
  - Update $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \alpha \boldsymbol{v}_t$

- Optimizing $f(w) = \dfrac{1}{N} \sum_{i=1}^{N} f_i(w)$
- α: learning rate
- β: discount factor ($\beta = 0$ means no momentum)

Optimization

Nesterov accelerated gradient

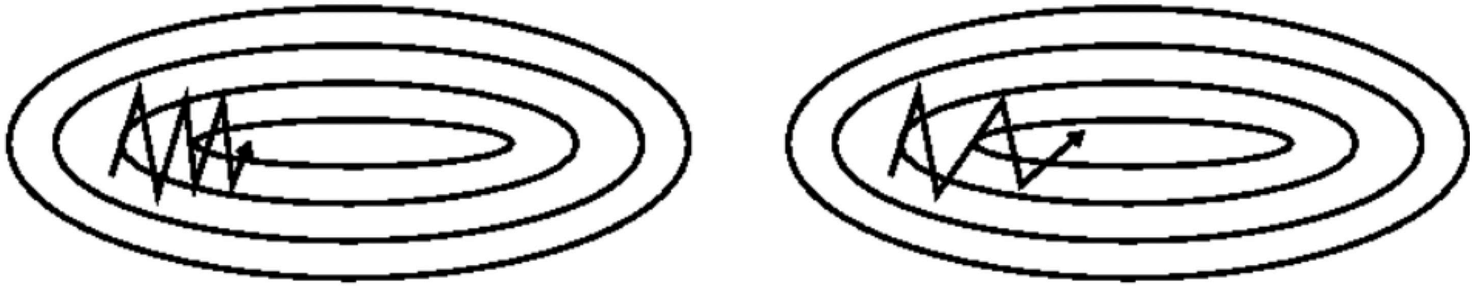
- Using the “look-ahead” gradient
  - $v_t = \beta v_{t-1} + \alpha \nabla f(w_t - \beta v_{t-1})$
  - $w_{t+1} = w_t - v_t$



Optimization

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)

Optimization

Adagrad: Adaptive updates

- 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

Optimization

Adagrad

- For each dimension i, we have observed T samples $g_i^1, \ldots, g_i^t$
- 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
  - $w_i^{t+1} \leftarrow w_i^t - \frac{\eta}{\sqrt{t}} \frac{\sqrt{t}}{\sqrt{(G_i^t)^2}} g_i^t$

Optimization

Adam: Momentum + Adaptive updates

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 $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_t - 1 - \alpha \cdot \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon)$

Optimization

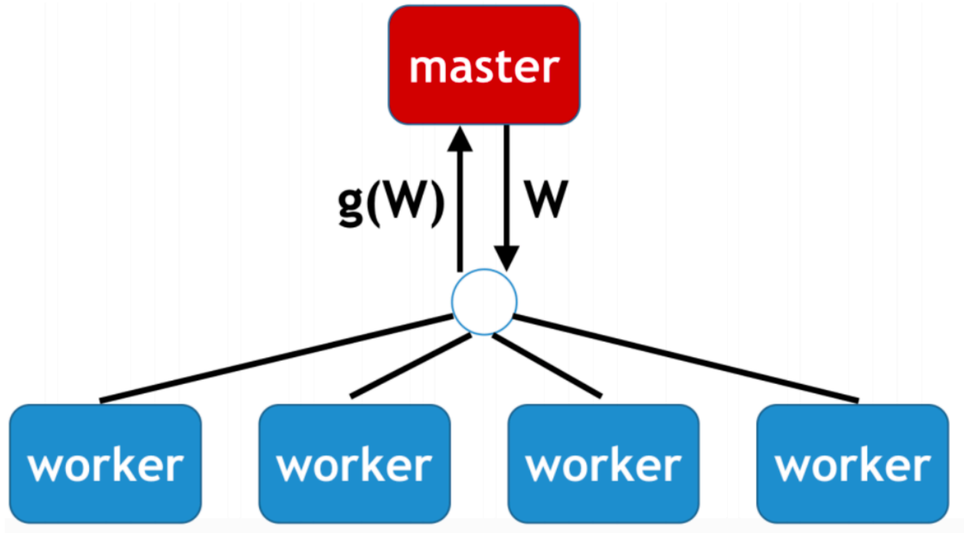
Batch size selection

- Larger batch size ⇒ more computation per update, less noise
- Usually, choose batch size large enough that
  - Fits in (GPU) memory
  - Can fully utilize the computation resource
- For example, 512 batch size for ImageNet training on a standard GPU.

Optimization

Large batch training

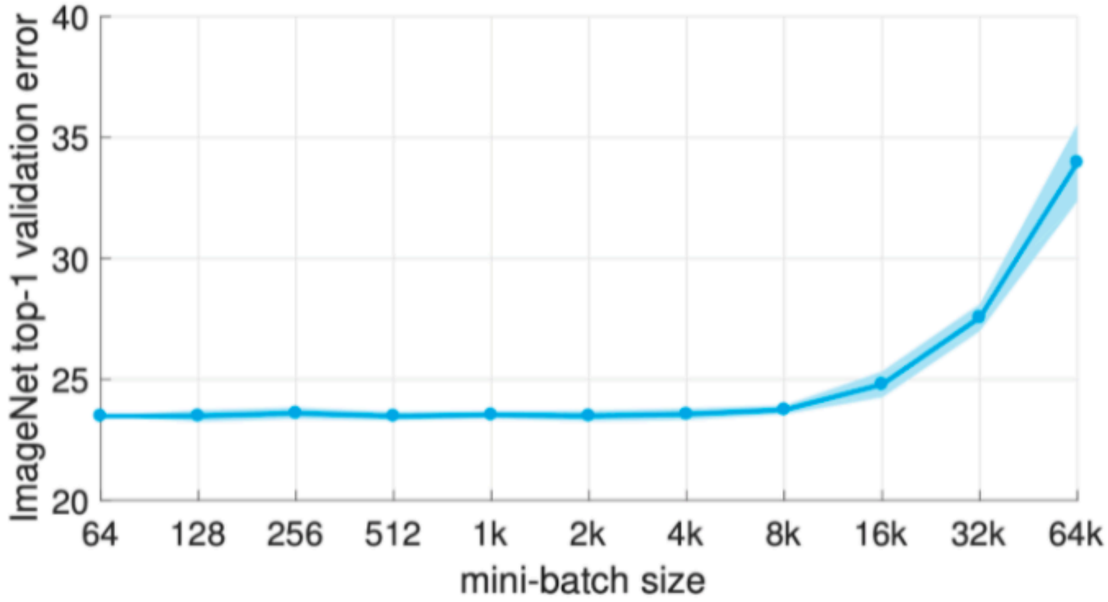
- What if we want to train a model with hundreds or thousands of GPUs
- Data parallel distributed computing:
  - Increasing the batch size linearly with number of devices.
  - ⇒ Large batch training



Optimization

Problem of large batch training

- Tend to converge to models with lower test accuracy when using very large batch size

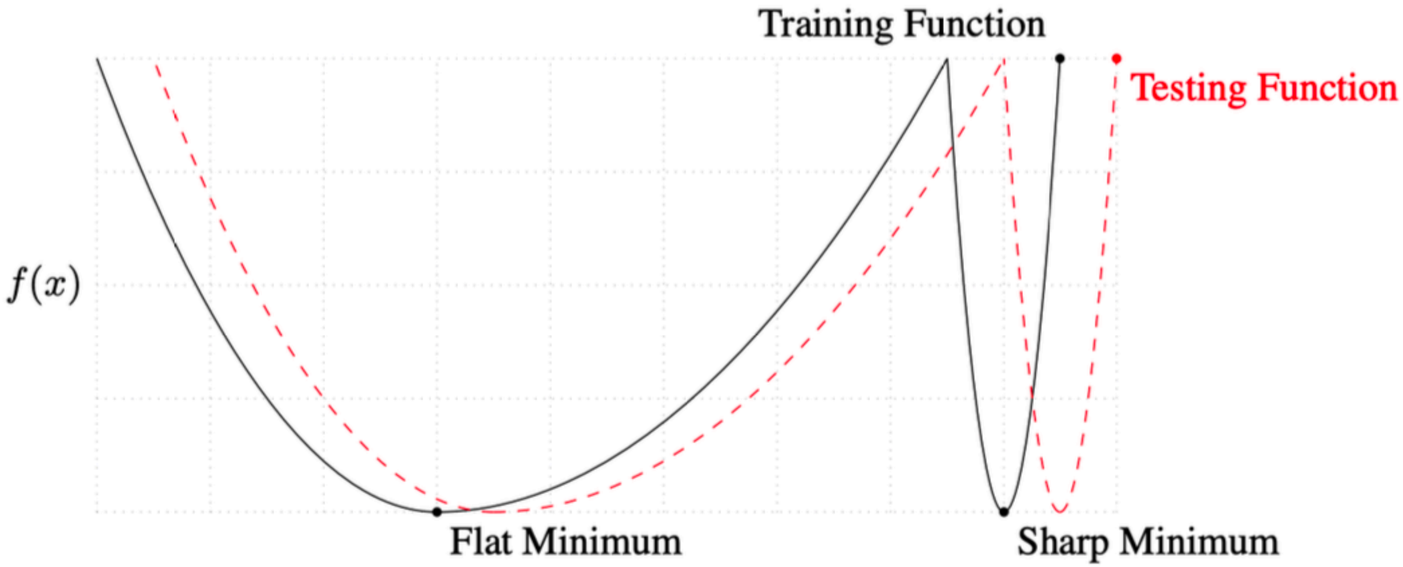


(Goyal et al., 2017) Training Imagenet in 1 hour

Optimization

Sharp vs wide local minimum

- Large-batch SGD/Adam:
  - Usually converge to a sharp local minimum (not enough inherent noise in SGD)
  - Harder to generalize to test data



Optimization

A simple but practical solution: learning rate scaling

- Var$[\frac{1}{|B|}\sum_{n\in B} g_n] \approx O(\frac{1}{|B|})$
- Batch size ↑, learning rate ↑
  - LR scaling: LR as $O(\sqrt{|B|})$ or $O(|B|)$
- However, LR has to be bounded for convergence (as for GD)
  - $LR \leq O(1/L)$ $\quad\quad$ L : Lipchitz constant
- Can't unlimitedly increase LR

Optimization

Non-uniform updates between different layers

- Some layer becomes the bottleneck of LR

Layers	$\|\|w\|\|_2$	$\|\|\nabla w\|\|_2$	$\|\|w\|\|_2/\|\|\nabla w\|\|_2$
fc8.0	20.24	0.078445	258
fc8.1	0.316	0.006147	51
fc7.0	20.48	0.110949	184
fc7.1	6.400	0.004939	1296
fc6.0	30.72	0.097996	314
fc6.1	6.400	0.001734	**3690**
conv5.0	6.644	0.034447	193
conv5.1	0.160	0.000961	166
conv4.0	8.149	0.039939	204
conv4.1	0.196	0.000486	403
conv3.0	9.404	0.049182	191
conv3.1	0.196	0.000511	384
conv2.0	5.545	0.057997	96
conv2.1	0.160	0.000649	247
conv1.0	1.866	0.071503	26
conv1.1	0.098	0.004909	**20**

Optimization

Another solution

- Use **Layer-wise Adaptive LR Scaling**
  - $$w^{(i)} \leftarrow w^{(i)} - \eta \frac{\|w^{(i)}\|}{\|g^{(i)}\|} g^{(i)},$$
  - where $(\cdot)^{(i)}$ means the i-th layer of neural network
- LARS: g is the stochastic gradient
- LAMB: g is the Adam update
- Google and Nvidia both use LAMB to train BERT within 1 minute (4096 TPU / 2048 GPU)