Large-scale Problems

- Machine learning: usually minimizing the training loss

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

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

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

- Gradient descent:

\[
\mathbf{w} \leftarrow \mathbf{w} - \eta \quad \nabla f(\mathbf{w})
\]

Main computation
Large-scale Problems

- Machine learning: usually minimizing the training loss

\[
\min_w \left\{ \frac{1}{N} \sum_{n=1}^{N} \ell(w^T x_n, y_n) \right\} := f(w) \quad \text{(linear model)}
\]

\[
\min_w \left\{ \frac{1}{N} \sum_{n=1}^{N} \ell(h_w(x_n), y_n) \right\} := f(w) \quad \text{(general hypothesis)}
\]

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

- Gradient descent:

\[
w \leftarrow w - \eta \quad \nabla f(w)
\]

Main computation

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

  each \(f_n(w)\) only depends on \((x_n, y_n)\)
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 slowly when millions of samples

- Faster way to compute “approximate gradient”?
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 compute “approximate gradient”?

- Use stochastic sampling:
  - Sample a small subset \( B \subseteq \{1, \cdots, N\} \)
  - Estimated gradient

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

\( |B| \): batch size
Stochastic gradient descent

Stochastic Gradient Descent (SGD)

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

\[
\begin{align*}
    w &\leftarrow w - \eta^t \frac{1}{|B|} \sum_{n \in B} \nabla f_n(w)
\end{align*}
\]
Stochastic Gradient Descent (SGD)

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

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

**Extreme case:** \(|B| = 1 \Rightarrow \text{Sample one training data at a time}\)
Logistic Regression by SGD

- Logistic regression:

\[
\min_w \frac{1}{N} \sum_{n=1}^{N} \log(1 + e^{-y_n w^T x_n})
\]

SGD for Logistic Regression

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

\[
w \leftarrow w - \eta^t \frac{1}{|B|} \sum_{i \in B} \frac{-y_n x_n}{1 + e^{y_n w^T x_n}} \nabla f_n(w)
\]
Why SGD works?

- Stochastic gradient is an unbiased estimator of full gradient:

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

\[
= \nabla f(w)
\]
Why SGD works?

- Stochastic gradient is an unbiased estimator of full gradient:

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

- Each iteration updated by

gradient + zero-mean noise
In gradient descent, \( \eta \) (step size) is a fixed constant.

Can we use fixed step size for SGD?

Even if we got a minimizer, SGD will move away from it.
In gradient descent, \( \eta \) (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
In gradient descent, $\eta$ (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 $\mathbf{w}^*$ 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

- In gradient descent, $\eta$ (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
Stochastic gradient descent

- In gradient descent, $\eta$ (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 $\mathbf{w}^*$ 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

(Even if we got minimizer, SGD will move away from it)
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)
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’s the gradient?
Revisit perceptron Learning Algorithm

\[ \ell(w^T x_n, y_n) = \max(0, -y_n w^T x_n) \]

Consider two cases:

- **Case I:** \( y_n w^T x_n > 0 \) (prediction **correct**)
  - \( \ell(w^T x_n, y_n) = 0 \)
  - \( \frac{\partial}{\partial w} \ell(w^T x_n, y_n) = 0 \)

- **Case II:** \( y_n w^T x_n < 0 \) (prediction **wrong**)
  - \( \ell(w^T x_n, y_n) = -y_n w^T x_n \)
  - \( \frac{\partial}{\partial w} \ell(w^T x_n, y_n) = -y_n x_n \)

**SGD update rule:** Sample an index

\[ w_{t+1} \leftarrow \begin{cases} 
  w_t & \text{if } y_n w^T x_n \geq 0 \text{ (predict correct)} \\
  w_t + \eta_t y_n x_n & \text{if } y_n w^T x_n < 0 \text{ (predict wrong)} 
\end{cases} \]

Equivalent to Perceptron Learning Algorithm when \( \eta_t = 1 \)
Revisit perceptron Learning Algorithm

\[ \ell(w^T x_n, y_n) = \max(0, -y_n w^T x_n) \]

Consider two cases:

- **Case I:** \( y_n w^T x_n > 0 \) (prediction **correct**)
  - \( \ell(w^T x_n, y_n) = 0 \)
  - \( \frac{\partial}{\partial w} \ell(w^T x_n, y_n) = 0 \)

- **Case II:** \( y_n w^T x_n < 0 \) (prediction **wrong**)
  - \( \ell(w^T x_n, y_n) = -y_n w^T x_n \)
  - \( \frac{\partial}{\partial w} \ell(w^T x_n, y_n) = -y_n x_n \)
Revisit perceptron Learning Algorithm

\[ \ell(w^T x_n, y_n) = \max(0, -y_n w^T x_n) \]

Consider two cases:

- **Case I:** \( y_n w^T x_n > 0 \) (prediction correct)
  - \( \ell(w^T x_n, y_n) = 0 \)
  - \( \frac{\partial}{\partial w} \ell(w^T x_n, y_n) = 0 \)

- **Case II:** \( y_n w^T x_n < 0 \) (prediction wrong)
  - \( \ell(w^T x_n, y_n) = -y_n w^T x_n \)
  - \( \frac{\partial}{\partial w} \ell(w^T x_n, y_n) = -y_n x_n \)

**SGD update rule:** Sample an index \( n \)

\[ w^{t+1} = \begin{cases} w^t & \text{if } y_n w^T x_n \geq 0 \text{ (predict correct)} \\ w^t + \eta^t y_n x_n & \text{if } y_n w^T x_n < 0 \text{ (predict wrong)} \end{cases} \]

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:

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

\( \beta \in [0, 1) \): discount factors, \( \alpha \): 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, \(\alpha\): step size

- Equivalent to using moving average of gradient:

\[
\mathbf{v}_t = (1 - \beta) \nabla f(\mathbf{w}_t) + \beta(1 - \beta) \nabla f(\mathbf{w}_{t-1}) + \beta^2(1 - \beta) \nabla f(\mathbf{w}_{t-2}) + \cdots
\]
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, \( \alpha \): step size
- Equivalent to using moving average of gradient:

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

- Another equivalent form:

\[ \mathbf{v}_t = \beta \mathbf{v}_{t-1} + \alpha \nabla f(\mathbf{w}_t) \]
\[ \mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{v}_t \]
Momentum gradient descent

Initialize \( w_0, v_0 = 0 \)

For \( t = 1, 2, \cdots \)

- Compute \( v_t \leftarrow \beta v_{t-1} + (1 - \beta) \nabla f(w_t) \)
- Update \( w_{t+1} \leftarrow w_t - \alpha v_t \)

\( \alpha \): learning rate

\( \beta \): discount factor (\( \beta = 0 \) means no momentum)
Momentum stochastic gradient descent

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

Momentum stochastic gradient descent

- Initialize $w_0$, $v_0 = 0$
- For $t = 1, 2, \cdots$
  - Sample an $i \in \{1, \cdots, N\}$
  - Compute $v_t \leftarrow \beta v_{t-1} + (1 - \beta) \nabla f_i(w_t)$
  - Update $w_{t+1} \leftarrow w_t - \alpha v_t$

$\alpha$: learning rate

$\beta$: discount factor ($\beta = 0$ means no momentum)
Nesterov accelerated gradient

- Using the “look-ahead” gradient

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

\[ \mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{v}_t \]

(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)

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

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

**Adagrad**

- Initialize $w_0$
- For $t = 1, 2, \ldots$
  - Sample an $i \in \{1, \ldots, N\}$
  - Compute $g^t \leftarrow \nabla f_i(w_t)$
  - $G^t_i \leftarrow G^{t-1}_i + (g^t_i)^2$
  - Update $w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{G^t_i + \epsilon}} g^t_i$

$\eta$: step size (constant)
$\epsilon$: 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 $g_{i}$:
  \[ \sqrt{\frac{\sum_{t'}(g_{i}^{t'})^2}{t}} = \frac{(G_{i}^{t})^2}{t} \]
- Assume step size is $\eta/\sqrt{t}$, then the update becomes
  \[ w_{i}^{t+1} \leftarrow w_{i}^{t} - \frac{\eta \sqrt{t}}{\sqrt{(G_{i}^{t})^2}} g_{i}^{t} \]
Adam: Momentum + Adaptive updates (2015)

Adam

- Initialize $w_0, m_0 = 0, v_0 = 0$,
- For $t = 1, 2, \cdots$
  - Sample an $i \in \{1, \cdots, N\}$
  - Compute $g_t \leftarrow \nabla f_i(w_t)$
  - $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  - $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  - $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
  - $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
  - Update $w_t \leftarrow w_t - 1 - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
Conclusions

- Stochastic gradient descent
- Momentum & adaptive updates

Questions?