STA 221: LECTURE 9

KRISHNA BALASUBRAMANIAN

(University of California, Davis)

PART II: SUPERVISED LEARNING

Supervised Learning

- Supervised Learning consists of two problems:
  - Regression: Predicting real-number outputs. For example, predicting stocks, height, weight, etc.
  - Classification: Predicting binary (or categorical) outputs. For example, predicting gender, class grades, spam/not-spam, etc.

Regression

- Input: training data $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and corresponding outputs (also called as labels) $y_1, y_2, \ldots, y_n \in \mathbb{R}$
- Training: compute a function f such that $f(\boldsymbol{x}_i) \approx y_i$ for all i
- Prediction: given a testing sample $\tilde{\boldsymbol{x}}$, predict the output as $f(\tilde{\boldsymbol{x}})$
- Examples:
  - Income, number of children $\Rightarrow$ Consumer spending
  - Processes, memory $\Rightarrow$ Power consumption
  - Financial reports $\Rightarrow$ Risk
  - Atmospheric conditions $\Rightarrow$ Precipitation

Linear Regression

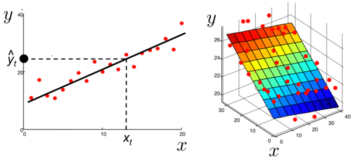
- ▷ Assume $f(\cdot)$ is a linear function parameterized by $\boldsymbol{w} \in \mathbb{R}^d$:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$$

- ▷ Training: compute the model $\boldsymbol{w}$ such that $\boldsymbol{w}^T \boldsymbol{x}_i \approx y_i$ for all i
- ▷ Equivalent to solving

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^d} \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2$$

- ▷ Prediction: given a testing sample $\tilde{\boldsymbol{x}}$, the prediction value is $\boldsymbol{w}^T \tilde{\boldsymbol{x}}$



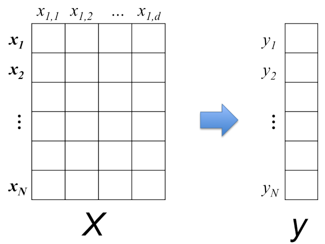
Assume the data is generated from the probability model:

$$y_i \sim \boldsymbol{w}^T \boldsymbol{x}_i + \varepsilon_i, \quad \varepsilon_i \sim N(0,1)$$

Maximum likelihood estimator:

$$\begin{aligned}
\boldsymbol{w}^* &= \arg\max_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{i=1}^{n} \log P(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}} e^{-(\boldsymbol{w}^T\boldsymbol{x}_i - y_i)^2/2}\right) \\
&= \arg\max_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{i=1}^{n} -\frac{1}{2}(\boldsymbol{w}^T\boldsymbol{x}_i - y_i)^2 + \text{constant} \\
&= \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{i=1}^{n} (\boldsymbol{w}^T\boldsymbol{x}_i - y_i)^2
\end{aligned}$$

Linear Regression: written as a matrix form



- Linear regression: $\boldsymbol{w}^* = \operatorname{argmin}_{\boldsymbol{w} \in \mathbb{R}^d} \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2$
- Matrix form: let $X \in \mathbb{R}^{n \times d}$ be the matrix where the i-th row is $\boldsymbol{x}_i$, $\mathbf{y} = [y_1, \ldots, y_n]^T$, then linear regression can be written as

$$\boldsymbol{w}^* = \underset{\boldsymbol{w} \in \mathbb{R}^d}{\operatorname{argmin}} \|X\boldsymbol{w} - \mathbf{y}\|_2^2$$

- Minimize the sum of squared error $J(\boldsymbol{w})$

$$\begin{aligned} J(\boldsymbol{w}) &= \frac{1}{2}\|X\boldsymbol{w} - \mathbf{y}\|^2 \\ &= \frac{1}{2}(X\boldsymbol{w} - \mathbf{y})^T(X\boldsymbol{w} - \mathbf{y}) \\ &= \frac{1}{2}\boldsymbol{w}^T X^T X \boldsymbol{w} - \mathbf{y}^T X \boldsymbol{w} + \frac{1}{2}\mathbf{y}^T\mathbf{y} \end{aligned}$$

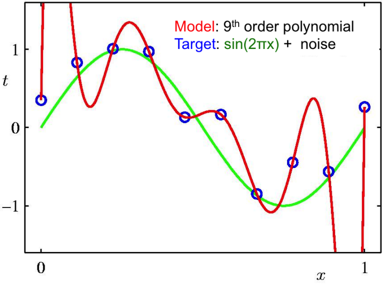
- Derivative: $\frac{\partial}{\partial \boldsymbol{w}} J(\boldsymbol{w}) = X^T X \boldsymbol{w} - X^T \mathbf{y}$
- Setting the derivative equal to zero gives the normal equation

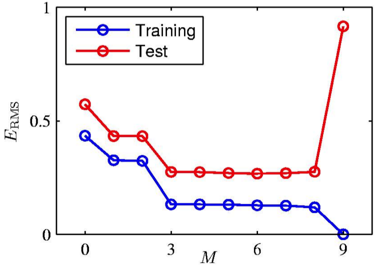
$$X^T X \boldsymbol{w}^* = X^T \mathbf{y}$$

- Therefore, $\boldsymbol{w}^* = (X^T X)^{-1} X^T \mathbf{y}$

Regularized Linear Regression

Overfitting





- Overfitting: the model has low training error but high prediction error.
- Using too many features can lead to overfitting

Regularization to Avoid Overfitting

- Enforce the solution to have low L2-norm:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \|\boldsymbol{w}^T \boldsymbol{x}_i - y_i\|^2 \text{ s.t. } \|\boldsymbol{w}\|^2 \leq K$$

- Equivalent to the following problem with some λ

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \|\boldsymbol{w}^T \boldsymbol{x}_i - yi\|^2 + \lambda \|\boldsymbol{w}\|^2$$

- Regularized Linear Regression:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \|X\boldsymbol{w} - \mathbf{y}\|^2 + R(\boldsymbol{w})$$

$R(\boldsymbol{w})$: regularization

- Ridge Regression (ℓ_2 regularization):

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \|X\boldsymbol{w} - \mathbf{y}\|^2 + \lambda\|\boldsymbol{w}\|^2$$

Ridge Regression

- ▷ Ridge regression: $\operatorname{argmin}_{\boldsymbol{w} \in \mathbb{R}^d} \underbrace{\frac{1}{2}\|X\boldsymbol{w} - \mathbf{y}\|^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2}_{J(\boldsymbol{w})}$
- ▷ Closed form solution: optimal solution $\boldsymbol{w}^*$ satisfies $\nabla J(\boldsymbol{w}^*) = 0$:

$$\begin{aligned} X^T X\boldsymbol{w}^* - X^T\mathbf{y} + \lambda\boldsymbol{w}^* &= 0 \\ (X^T X + \lambda I)\boldsymbol{w}^* &= X^T\mathbf{y} \end{aligned}$$

- ▷ Optimial solution: $\boldsymbol{w}^* = (X^T X + \lambda I)^{-1} X^T\mathbf{y}$
- ▷ Inverse always exists because $X^T X + \lambda I$ is positive definite

- When X is dense:
  - Closed form solution requires $O(nd^2 + d^3)$ if X is dense
  - Efficient if d is very small
  - Runs forever when $d > 100,000$
- Typical case for big data applications:
  - $X \in \mathbb{R}^{n \times d}$ is sparse with large n and large d
  - How can we solve the problem?

    Iterative algorithms for optimization.

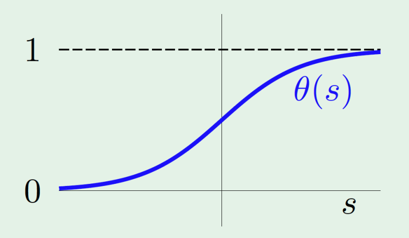
Logistic Regression

- ▷ Input: training data $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and corresponding outputs $y_1, y_2, \ldots, y_n \in \{+1, -1\}$s
- ▷ Training: compute a function f such that $\mathrm{sign}(f(\boldsymbol{x}_i)) \approx y_i$ for all i
- ▷ Prediction: given a testing sample $\tilde{\boldsymbol{x}}$, predict the output as $\mathrm{sign}(f(\tilde{\boldsymbol{x}}))$

- Assume linear scoring function: $s = f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$
- Logistic hypothesis:

$$P(y = 1 \mid \boldsymbol{x}) = \theta(\boldsymbol{w}^T\boldsymbol{x}),$$

where $\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$ is also called as sigmoid function.



- $\triangleright$ Likelihood of $\mathcal{D} = (\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)$:

$$\Pi_{n=1}^{N} P(y_n \mid \boldsymbol{x}_n)$$

Error measure: likelihood

- Likelihood of $\mathcal{D} = (\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)$:

$$\Pi_{n=1}^{N} P(y_n \mid \boldsymbol{x}_n)$$

- $P(y \mid \boldsymbol{x}) = \begin{cases} \theta(\boldsymbol{w}^T \boldsymbol{x}) & \text{for } y = +1 \\ 1 - \theta(\boldsymbol{w}^T \boldsymbol{x}) = \theta(-\boldsymbol{w}^T \boldsymbol{x}) & \text{for } y = -1 \end{cases}$

  $\Rightarrow P(y \mid \boldsymbol{x}) = \theta(y \boldsymbol{w}^T \boldsymbol{x})$

Likelihood: $\Pi_{n=1}^{N} P(y_n \mid \boldsymbol{x}_n) = \Pi_{n=1}^{N} \theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n)$

Maximizing the likelihood

Find $\boldsymbol{w}$ to maximize the likelihood!

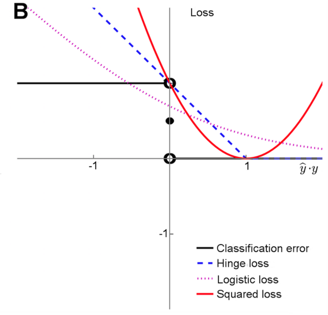
$$
\begin{aligned}
& \max_{\boldsymbol{w}} \Pi_{n=1}^{N} \theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n) \\
\Leftrightarrow & \max_{\boldsymbol{w}} \log(\Pi_{n=1}^{N} \theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n)) \\
\Leftrightarrow & \min_{\boldsymbol{w}} -\log(\Pi_{n=1}^{N} \theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n)) \\
\Leftrightarrow & \min_{\boldsymbol{w}} -\sum_{n=1}^{N} \log(\theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n)) \\
\Leftrightarrow & \min_{\boldsymbol{w}} \sum_{n=1}^{N} \log(\frac{1}{\theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n)}) \\
\Leftrightarrow & \min_{\boldsymbol{w}} \sum_{n=1}^{N} \log(1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n})
\end{aligned}
$$

Empirical Risk Minimization (linear)

- Most linear ML algorithms follow

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

  - Linear regression: $\text{loss}(h(\boldsymbol{x}_n), y_n) = (\boldsymbol{w}^T \boldsymbol{x}_n - y_n)^2$
  - Logistic regression: $\text{loss}(h(\boldsymbol{x}_n), y_n) = \log(1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n})$



Empirical Risk Minimization (general)

- Assume $f_W(\boldsymbol{x})$ is the decision function to be learned (W is the parameters of the function)
- General empirical risk minimization:

$$\min_{W} \frac{1}{N} \sum_{n=1}^{N} \text{loss}(f_W(\boldsymbol{x}_n), y_n)$$

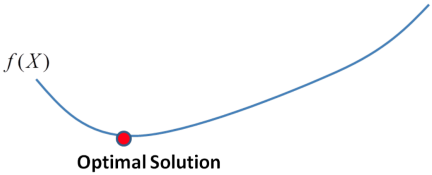
- Example: Neural network ($f_W(\cdot)$ is the network)

Gradient descent

- Goal: find the minimizer of a function

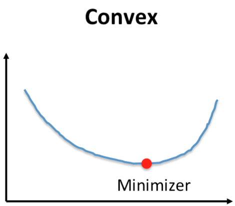
$$\min_{\boldsymbol{w}} f(\boldsymbol{w})$$

For now we assume f is twice differentiable



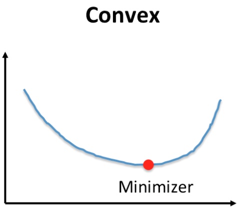
Convex vs Nonconvex

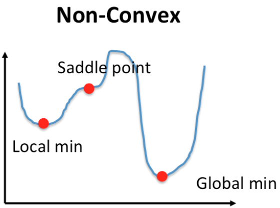
- Convex function:
  - $\nabla f(\boldsymbol{w}^*) = 0 \Leftrightarrow \boldsymbol{w}^*$ is global minimum
  - A function is convex if $\nabla^2 f(\boldsymbol{w})$ is positive definite
  - Example: linear regression, logistic regression, $\cdots$



- Convex function:
  - $\nabla f(\boldsymbol{w}^*) = 0 \Leftrightarrow \boldsymbol{w}^*$ is global minimum
  - A function is convex if $\nabla^2 f(\boldsymbol{w})$ is positive definite
  - Example: linear regression, logistic regression, $\cdots$
- Non-convex function:
  - $\nabla f(\boldsymbol{w}^*) = 0 \Leftrightarrow \boldsymbol{w}^*$ is Global min, local min, or saddle point

    most algorithms only converge to gradient$= 0$
  - Example: neural network, $\cdots$





Gradient Descent

- Gradient descent: repeatedly do

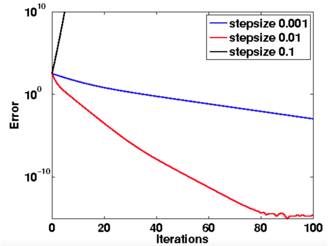
$$\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^{t} - \alpha \nabla f(\boldsymbol{w}^{t})$$

  $\alpha > 0$ is the step size

- Generate the sequence $\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots$

  converge to minimum solution ($\lim_{t \to \infty} \|\nabla f(\boldsymbol{w}^t)\| = 0$)

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



- Reason: successive approximation view

  At each iteration, form an approximation function of $f(\cdot)$:

$$f(\boldsymbol{w}^t + \boldsymbol{d}) \approx g(\boldsymbol{d}) := f(\boldsymbol{w}^t) + \nabla f(\boldsymbol{w}^t)^T \boldsymbol{d} + \frac{1}{2\alpha}\|\boldsymbol{d}\|^2$$

  Update solution by $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \boldsymbol{d}^*$

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} g(\boldsymbol{d})$$

$$\nabla g(\boldsymbol{d}^*) = 0 \Rightarrow \nabla f(\boldsymbol{w}^t) + \tfrac{1}{\alpha}\boldsymbol{d}^* = 0 \Rightarrow \boldsymbol{d}^* = -\alpha \nabla f(\boldsymbol{w}^t)$$

- $\boldsymbol{d}^*$ will decrease $f(\cdot)$ if α (step size) is sufficiently small

Illustration of gradient descent

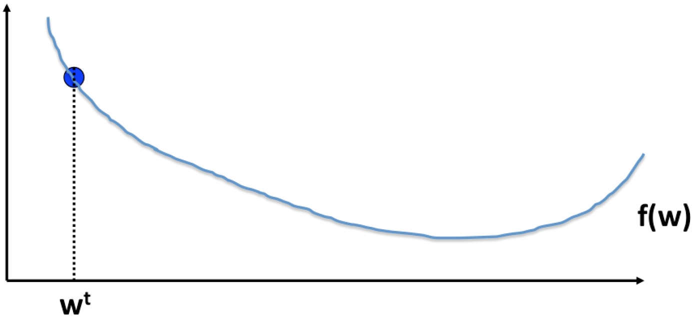
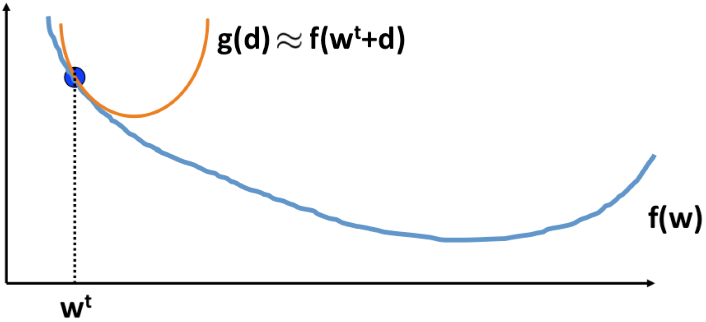


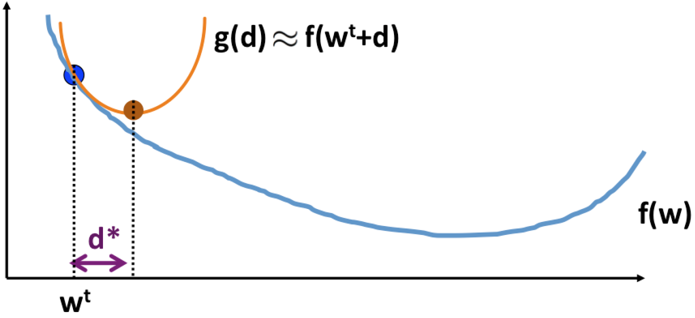
Illustration of gradient descent



Form a quadratic approximation

$$f(\boldsymbol{w}^t + \boldsymbol{d}) \approx g(\boldsymbol{d}) = f(\boldsymbol{w}^t) + \nabla f(\boldsymbol{w}^t)^T \boldsymbol{d} + \frac{1}{2\alpha}\|\boldsymbol{d}\|^2$$

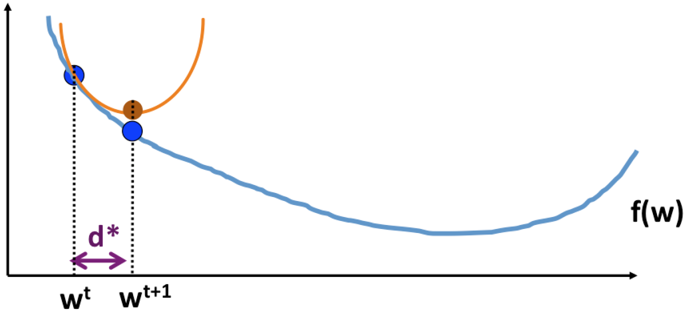
Illustration of gradient descent



Minimize $g(\boldsymbol{d})$:

$$\nabla g(\boldsymbol{d}^*) = 0 \Rightarrow \nabla f(\boldsymbol{w}^t) + \frac{1}{\alpha}\boldsymbol{d}^* = 0 \Rightarrow \boldsymbol{d}^* = -\alpha \nabla f(\boldsymbol{w}^t)$$

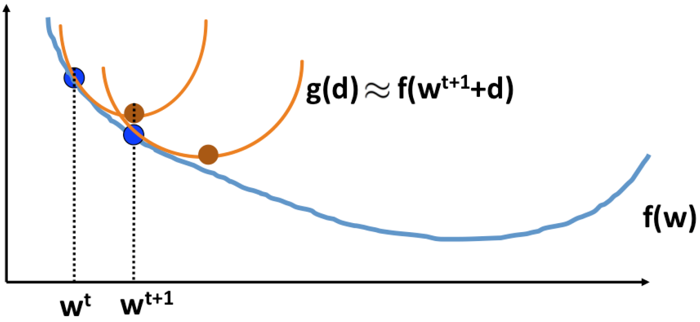
Illustration of gradient descent



Update

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{d}^{*} = \boldsymbol{w}^{t} - \alpha \nabla f(\boldsymbol{w}^{t})$$

Illustration of gradient descent

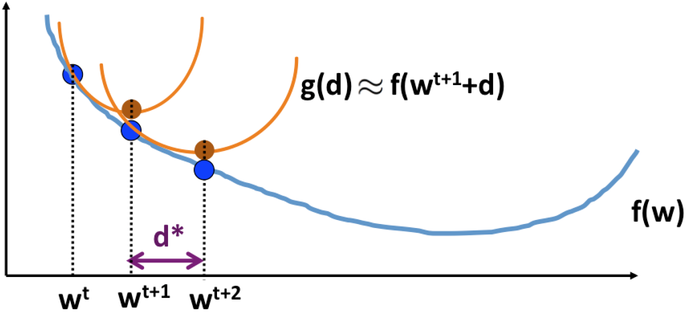


Form another quadratic approximation

$$f(\boldsymbol{w}^{t+1} + \boldsymbol{d}) \approx g(\boldsymbol{d}) = f(\boldsymbol{w}^{t+1}) + \nabla f(\boldsymbol{w}^{t+1})^T \boldsymbol{d} + \frac{1}{2\alpha}\|\boldsymbol{d}\|^2$$

$$\boldsymbol{d}^* = -\alpha \nabla f(\boldsymbol{w}^{t+1})$$

Illustration of gradient descent

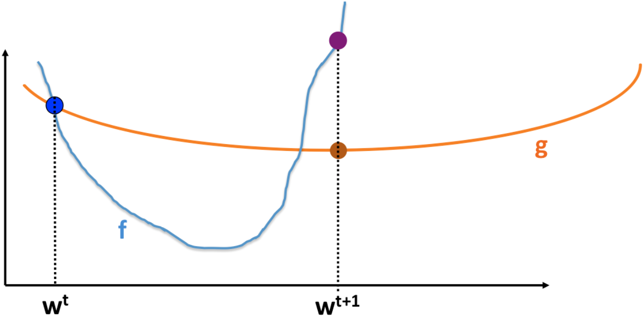


Update

$$\boldsymbol{w}^{t+2} = \boldsymbol{w}^{t+1} + \boldsymbol{d}^* = \boldsymbol{w}^{t+1} - \alpha \nabla f(\boldsymbol{w}^{t+1})$$

When will it diverge?

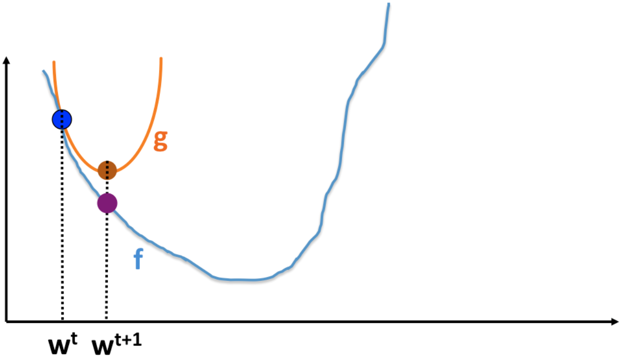
Can diverge ($f(\boldsymbol{w}^t) < f(\boldsymbol{w}^{t+1})$) if g is not an upperbound of f



f(w^t) < f(w^{t+1}), diverge because g's curvature is too small

When will it converge?

Always converge ($f(\boldsymbol{w}^t) > f(\boldsymbol{w}^{t+1})$) when g is an upperbound of f



f(w^t) > f(w^{t+1}), converge when g's curvature is large enough

- Let L be the Lipchitz constant

  $(\nabla^2 f(\boldsymbol{x}) \preceq LI$ for all $\boldsymbol{x})$
- **Theorem:** gradient descent converges if $\alpha < \frac{1}{L}$
- In practice, we do not know $L \cdots$

  need to tune step size when running gradient descent

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

$$\nabla f(\boldsymbol{w}) = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n \boldsymbol{x}_n}{1 + e^{y_n \boldsymbol{w}^T \boldsymbol{x}_n}}$$

  - Update the weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f(\boldsymbol{w})$
- Return the final weights $\boldsymbol{w}$

When to stop?

- Fixed number of iterations, or
- Stop when $\|\nabla f(\boldsymbol{w})\| < \varepsilon$

Stochastic Gradient Descent

▷ 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 $(\boldsymbol{x}_n, y_n)$

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

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

  $|B|$: batch size

Stochastic gradient descent

- 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:

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

- Each iteration updated by

$$\text{gradient} + \text{zero-mean noise}$$

- 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)

- 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:

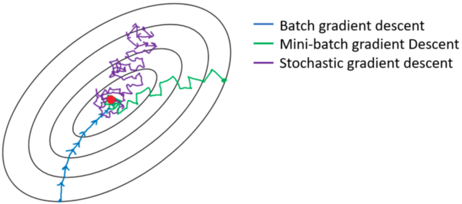
  cheaper computation per iteration

  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`)