SFU NatLangLab

CMPT 413/825: Natural Language Processing

Logistic Regression

Fall 2020

2020-09-25

Adapted from slides from Anoop Sarkar, Danqi Chen and Karthik Narasimhan

Naïve Bayes and Logistic Regression

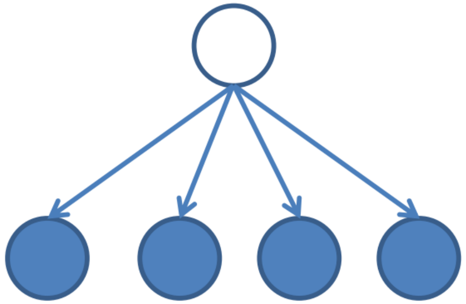
Naïve Bayes

- Generative Model

$$\hat{c} = \text{argmax}_c\, P(c)P(d|c)$$

Can sample words to generate document

- Features assumed to be independent



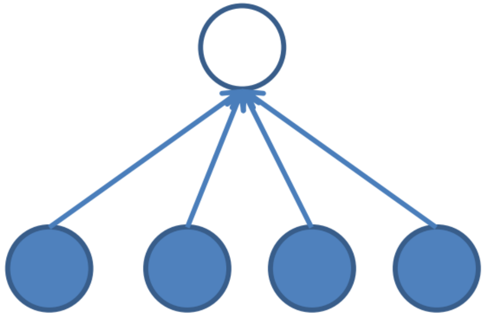
Logistic Regression

- Discriminative Model

$$\hat{c} = \text{argmax}_c\, P(c|d)$$

Optimize for $P(c|d)$ directly

- Features don't have to be independent



Logistic Regression

- Powerful supervised model
- Baseline approach to most NLP tasks
- Connections with neural networks
- Binary (2 class) or multinomial (>2 classes)

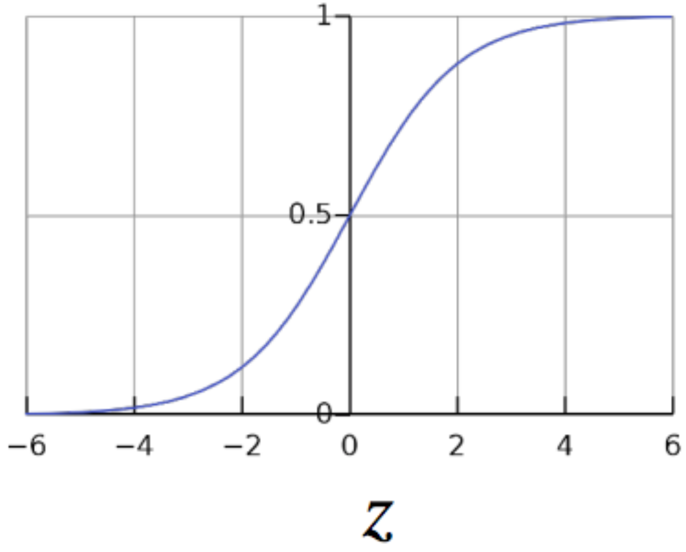
Logistic Regression Overview

- Input features: $f(x) \rightarrow [f_1, f_2, \ldots, f_m]$
  - Need to determine features
- Output: estimate $P(y = c|x)$ for each class c
  - Need to model $P(y = c|x)$ with a family of functions
- Train phase: Learn parameters of model to minimize loss function
  - Need Loss function and Optimization algorithm
- Test phase: Apply parameters to predict class given a new input

Binary Logistic Regression

- Input features: $f(x) \rightarrow [f_1, f_2, \dots, f_m]$
- Output: $P(y = 1|x)$ and $P(y = 0|x)$
- Classification function: $\sigma(z) = \frac{1}{1+e^{-z}}$
- How is z related to our features?

Sigmoid



vector notation

$$z = \sum_{i=1}^{m} w_i f_i + b \; = \mathbf{w} \cdot \mathbf{f} + b = \mathbf{w}^T \mathbf{f} + b$$

weight vector

$\mathbf{w} = [w_1, w_2, \dots, w_m]$

bias term

Can roll bias term into feature vector **f**

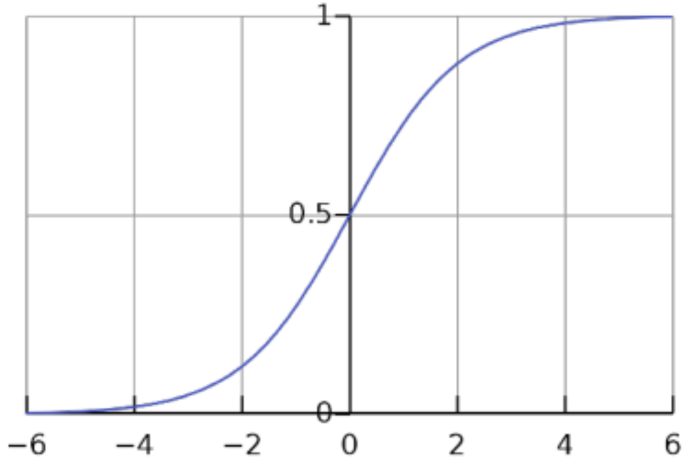
$$\mathbf{f}(\mathbf{x}) = [1, f_1, f_2, \dots, f_m]$$

$$z = \mathbf{v} \cdot \mathbf{f}(\mathbf{x})$$

$$\mathbf{v} = [b, w_1, w_2, \dots, w_m]$$

Binary Logistic Regression

Sigmoid



- Input features: $f(x) \rightarrow [f_1, f_2, \dots, f_m]$
- Output: $P(y = 1|x)$ and $P(y = 0|x)$
- Classification function: $\sigma(z) = \frac{1}{1+e^{-z}}$

  $z = \mathbf{v} \cdot \mathbf{f}(\mathbf{x})$

z

Q: Why do we need a bias term?

Q: What if it is the only term we had?

dummy feature

Example

Features: $[1, \text{count}(\text{“}amazing\text{”}), \text{count}(\text{“}horrible\text{”}), \dots]$

Weights: [-1.0, 0.8, -0.4, …]

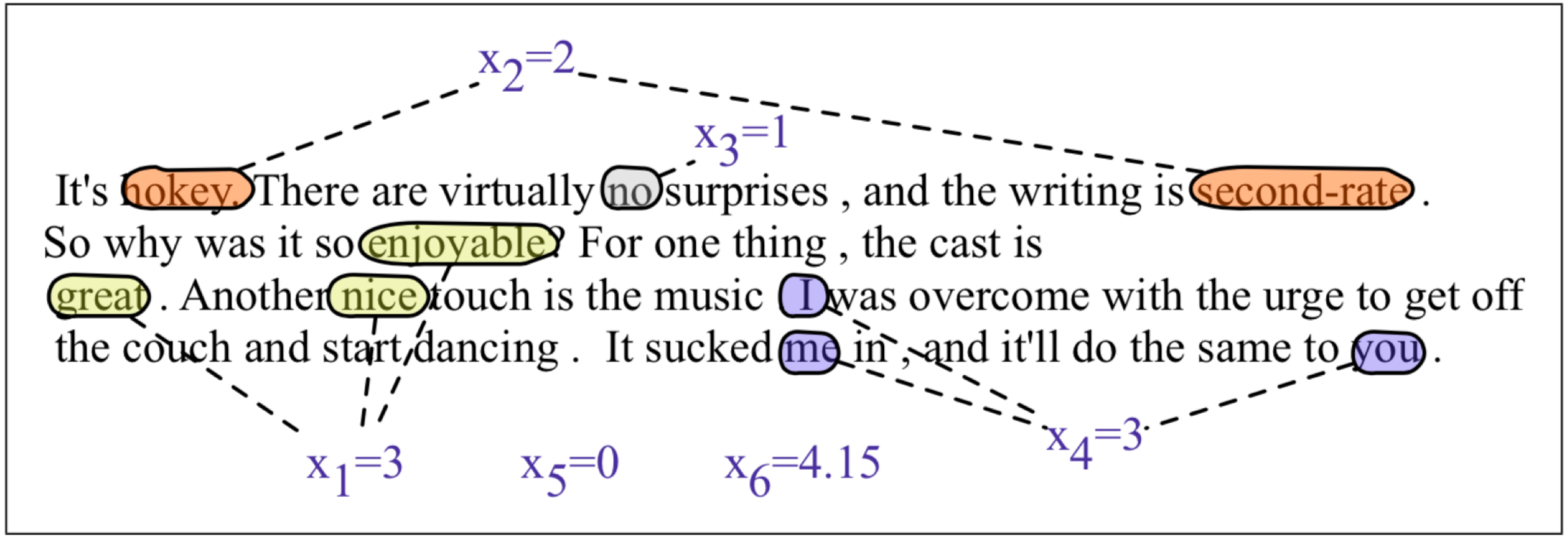
bias

Features can be anything!

No more independence assumptions!

Can also add $\text{count}(\text{“}amazing\text{”}, \text{“}horrible\text{”})$ to the features

Sentiment Analysis Example



Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Example from https://web.stanford.edu/~jurafsky/slp3/5.pdf

Sentiment Analysis Example

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- Assume weights $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ and bias $b = 0.1$

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Example from https://web.stanford.edu/~jurafsky/slp3/5.pdf

Learning the weights

- Goal: predict label $\hat{y}$ as close as possible to actual label y
- Need distance metric/loss function: $L(\hat{y}, y)$
- Maximum likelihood estimate:

Choose parameters so that $\log(\mathrm{y}|\mathrm{x})$ is maximized over the training dataset

$$\text{Maximize } \log \prod_{i=1}^{n} P\left(y^{(i)} \middle| x^{(i)}\right)$$

where $(x^{(i)}, y^{(i)})$ are paired documents and labels

Binary Cross Entropy Loss

- Let $\hat{y} = \sigma(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}))$
- Classifier probability: $P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$

$y = 1$: $P(y|x) = \hat{y}$ $\qquad$ $y = 0$: $P(y|x) = 1 - \hat{y}$

- Log probability: $\log P(y|x) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$

Binary Cross Entropy Loss

- Let $\hat{y} = \sigma(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}))$
- Classifier probability: $P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$
- Log probability: $\log P(y|x) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$
- Loss:

$$L(\hat{y}, y) = -\log \prod_{i=1}^{n} P\left(y^{(i)} \middle| x^{(i)}\right) = -\sum_{i=1}^{n} \log P\left(y^{(i)} \middle| x^{(i)}\right)$$

$$= -\sum_{i=1}^{n} [y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)]$$

Cross-entropy between the true distribution $P(y|x)$ and predicted distribution $P(\hat{y}|x)$

Binary Cross Entropy Loss

- Cross Entropy Loss:

$$\mathrm{L}_{\mathrm{CE}} = -\sum_{i=1}^{n} \log\left[y^{(i)}\log\hat{y}^{(i)} + \left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)\right]$$

- Ranges from 0 (perfect predictions) to $+\infty$
- Lower loss = better classifier
- *Cross-entropy* between the true distribution $P(y|x)$ and predicted distribution $P(\hat{y}|x)$

Multinomial Logistic Regression

- Input features: $f(x) \rightarrow [f_1, f_2, \dots, f_m]$
- Output: $P(y = c|x)$ for each class c
- Classification function Softmax

$$\frac{\exp(\mathbf{v}\cdot\mathbf{f}(\mathbf{x},y))}{\underbrace{\sum_{y'} \exp(\mathbf{v}\cdot\mathbf{f}(\mathbf{x},y'))}_{\text{Normalization}}}$$

Features are a function of both input x and output class c

Values are now between 0 and 1

Multinomial Logistic Regression

Features are a function of both input x and output class c

Var	Definition	Wt
$f_1(0,x)$	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	-4.5
$f_1(+,x)$	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	2.6
$f_1(-,x)$	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1.3

Multinomial Logistic Regression

- Generalize binary loss to multinomial CE loss

$$\mathrm{L_{CE}}(\hat{y}, \mathrm{y}) = -\sum_{c=1}^{k} 1\{\mathrm{y} = \mathrm{c}\} \log P(y = c|x)$$

$$= \sum_{c=1}^{k} 1\{\mathrm{y} = \mathrm{c}\} \frac{\exp(\mathbf{v_c} \cdot \mathbf{f}(\mathbf{x}, c))}{\sum_{y'=1}^{k} \exp(\mathbf{v_{y'}} \cdot \mathbf{f}(\mathbf{x}, y'))}$$

Logistic Regression Model Summary

- Classification function family:

$$\Pr(y|x;v) = \frac{\exp(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}, y))}{\sum_{y'} \exp(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}, y'))}$$

- Maximize log probability = Minimize cross-entropy

- Log probability can be decomposed into

$$\log \Pr(y \mid \mathbf{x}; \mathbf{v}) = \underbrace{\mathbf{v} \cdot \mathbf{f}(\mathbf{x}, y)}_{\text{linear term}} - \underbrace{\log \sum_{y'} exp\left(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}, y')\right)}_{\text{normalization term}}$$

- Log-linear model
- Maxent
- Shallow neural network

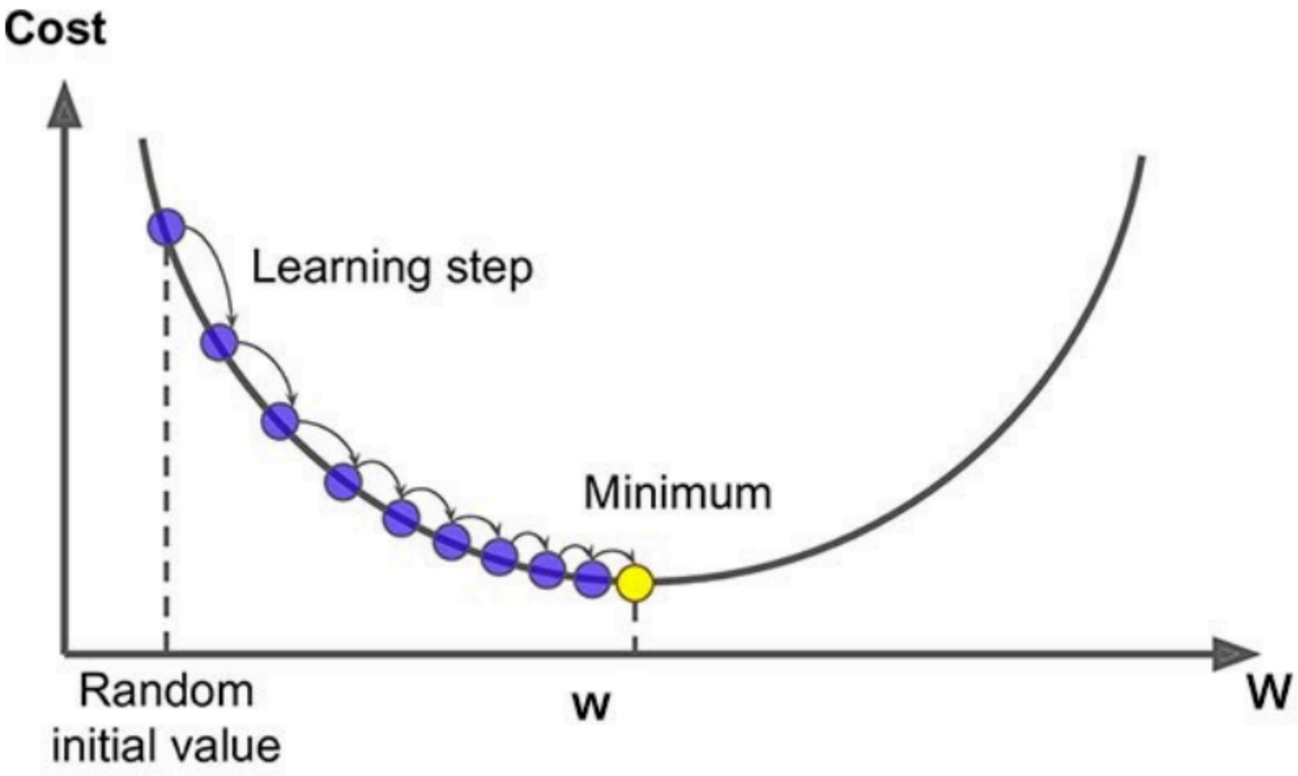
Optimization

- We have our loss function and our estimator $\hat{y}$ = $\sigma(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}))$
- How do we find the best set of parameters/weights: $\mathbf{v}$

$$\hat{\mathbf{v}} = \hat{\theta} = \arg\min \frac{1}{n} \sum_{i=1}^{n} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

- Use gradient descent!
  - Find direction of steepest slope
  - Move in opposite direction

Gradient descent (1-D)

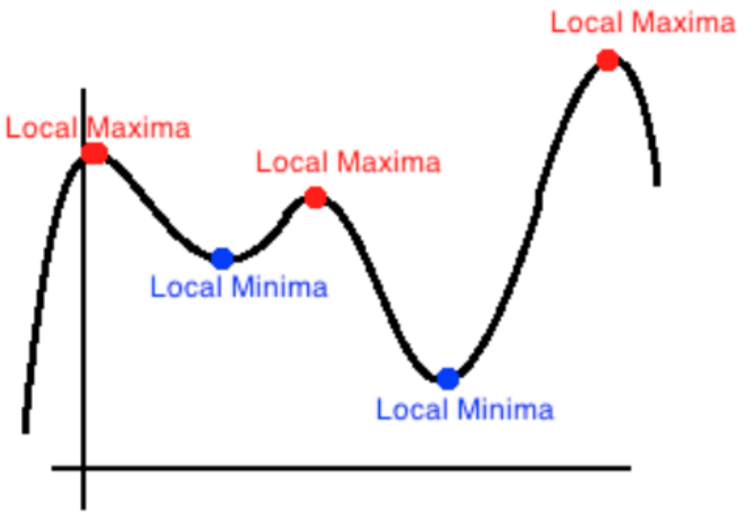


$$\theta^{t+1} = \theta^{t} - \eta \frac{d}{d\theta} f(x;\theta)$$

Gradient descent for LR

- Cross entropy loss for logistic regression is convex (i.e. has only one global minimum)
  - No local minima to get stuck in

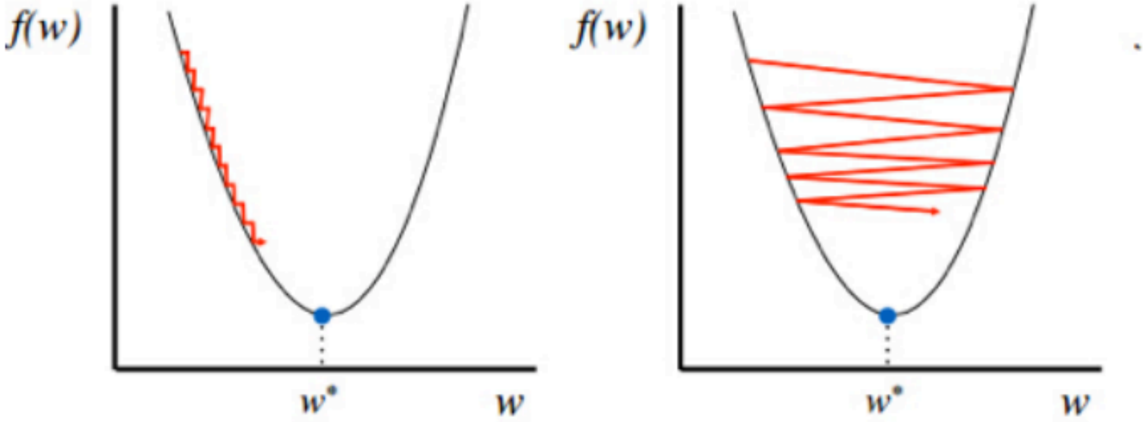
- Deep neural networks are not so easy
  - Non-convex



Learning Rate

- Updates: $\theta^{t+1} = \theta^{t} - \eta \frac{d}{d\theta} f(x;\theta)$ Magnitude of movement

- Higher/faster learning rate = larger update



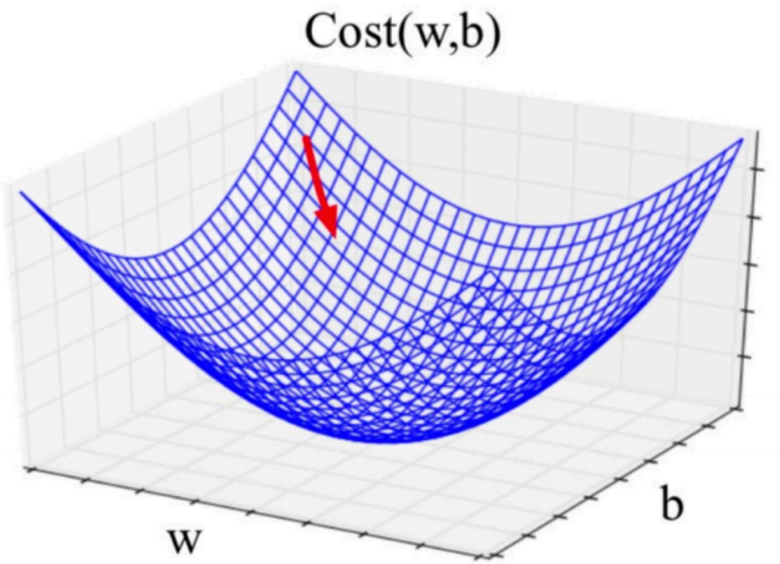
Too small: converge very slowly

Too big: overshoot and even diverge

Gradient descent with vector weights

Express slope as a partial derivative of loss w.r.t each weight:

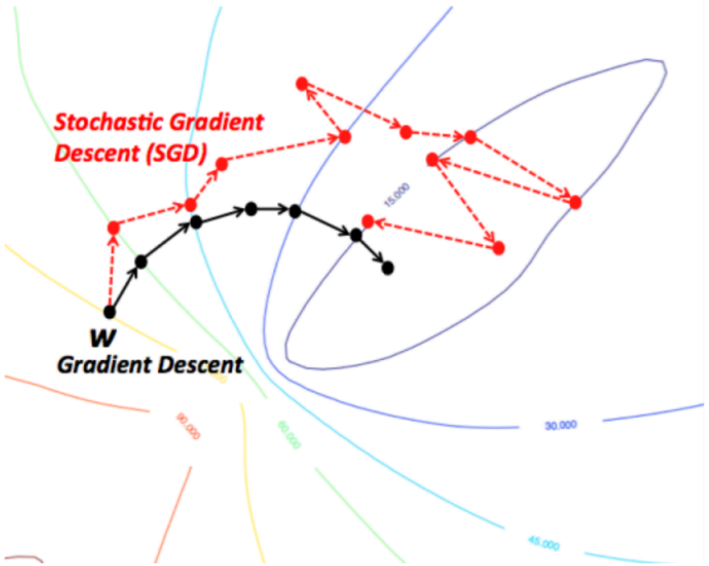
$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$



Updates: $\theta^{(t+1)} = \theta^t - \eta \nabla L(f(x;\theta), y)$

Stochastic Gradient Descent

- Online optimization
- Compute loss and minimize after each training examples
  (or mini-batch)



Regularization

- May overfit on the training data!
- Use regularization to prevent overfitting!
- Objective function:

Penalize large weights

$$\hat{\theta} = \arg\max \sum_{i=1}^{n} \log P\left(y^{(i)} \middle| x^{(i)}\right) - \alpha\, R(\theta)$$

L2 Regularization

$$R(\theta) = ||\theta||^2 = \sum_{j=1}^{d} \theta_j^2$$

Euclidean distance of weight vector θ from origin

L2 regularized objective:

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^{d} \theta_j^2$$

L1 Regularization

$$R(\theta) = ||\theta||_1 = \sum_{j=1}^{d} |\theta_j|$$

Manhattan distance of weight vector θ from origin

L1 regularized objective:

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^{d} |\theta_j|$$

L2 vs L1 regularization

- L2 is easier to optimize
  - L1 is complex since the derivative of |θ| is not continuous at 0

- L2 leads to many small weights
  - L1 prefers sparse weight vectors with many weights set to 0