Natural Language Processing

CSCI 5832—Lecture 9

Jim Martin

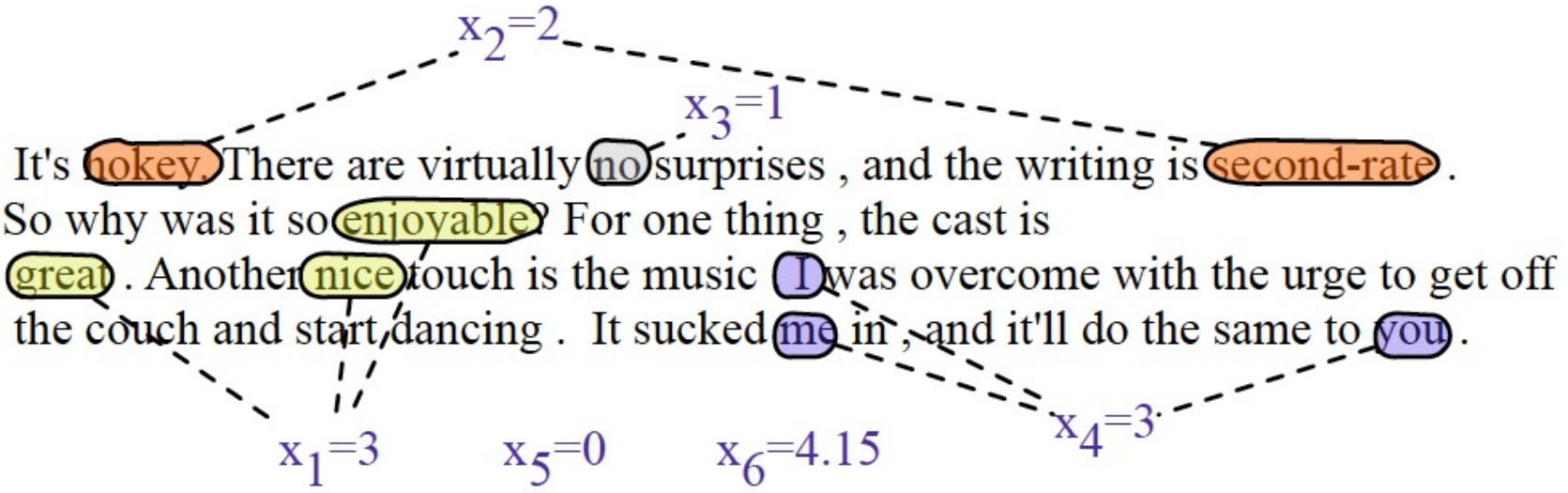
Today

- More Logistic Regression
  - Chapter 5

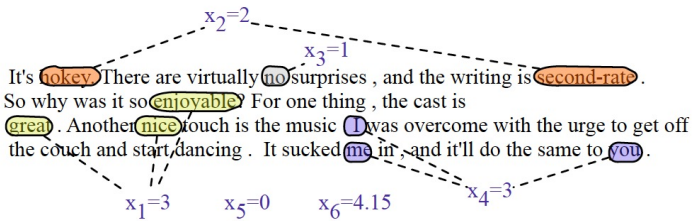
Sentiment Features

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)	$\ln(64) = 4.15$

Sentiment Features



Scoring

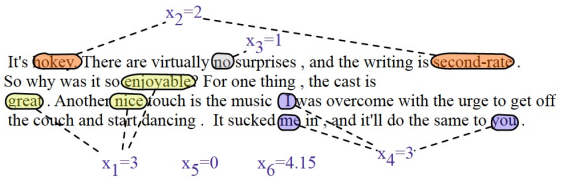


[3, 2, 1, 3, 0, 4.15]

Now assume we have already have weights for each of these features.

2.5, -5.0, -1.2, 0.5, 2.0, 0.7

Scoring



[3,2,1,3,0,4.15]

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Weights

- So… where do those weights come from?
- We'll learn the weights from a training set of documents already labeled with the right answer.
  - We'll use 1 and 0 as labels to stand for the right answer (positive or negative)
  - The system answers will be between 0 and 1

Weights

- To learn the weights, we need some measure of how well (or badly) we're doing with a current set of weights.
- We'll call that measure a *Loss Function*
- The lower the loss the better we're doing
  - We want to minimize the loss

Loss Functions

- There are lots of ways to evaluate how well we're doing on a test/validation set.
  - Accuracy, F1, Precision, recall.
    - All discrete
  - How close are our answers are to the correct answers?
    - What does close even mean?
  - What's the nature of the overall loss over the entire training/test data

Loss Functions

- What we want is a function that tells us how well our model is doing.
- And does it in a way that can be used to guide the training process

Developing CE Loss

- Let's start with the following notion

  the probability assigned by a model to the correct answer

  - In the binary case, the correct answer is always either 1 or 0.
  - The model output is a number between 0 and 1. That represents the probability that the item belongs to the class.
    - By convention, the probability given by the model is the probability that the doc belongs to class 1 (or the positive class)

Cross Entropy Loss

$$p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

y is the correct answer.
y_hat is the system answer.

Logs

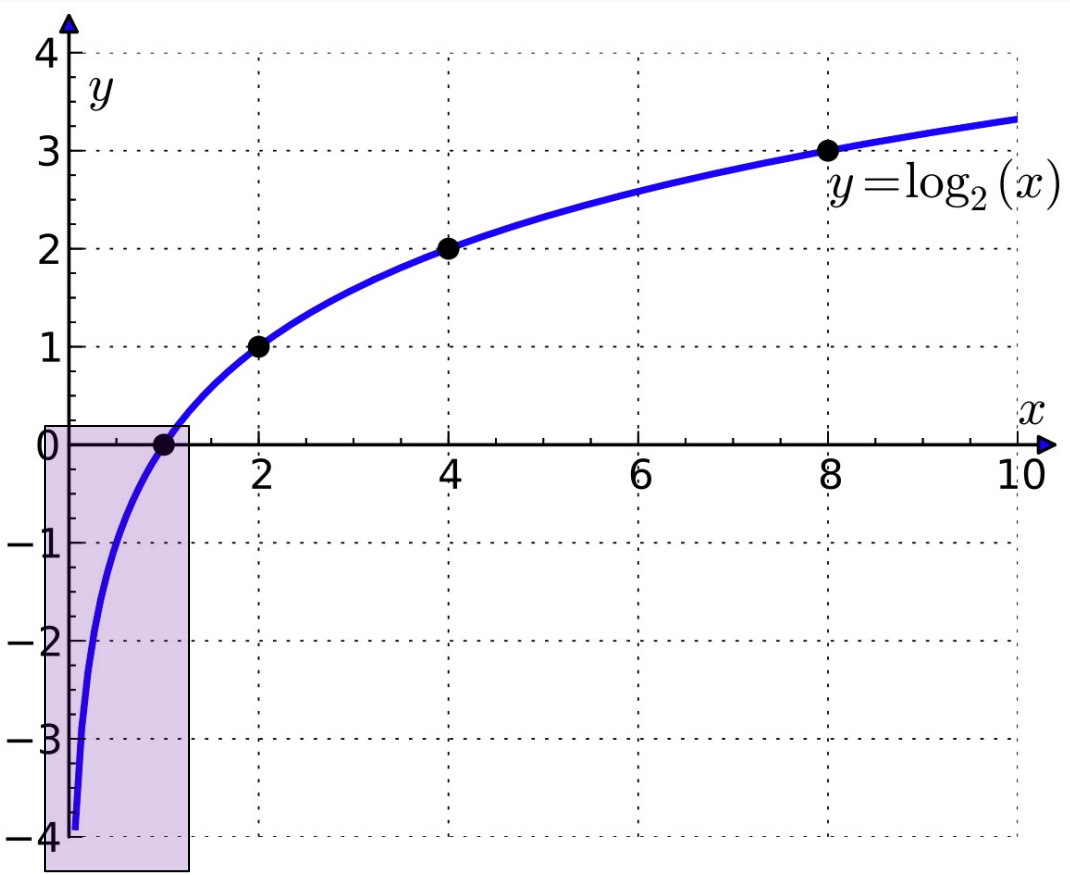
- Probabilities are the opposite of what we want for a loss.
  - We want bad performance to have high loss and good performance to have low loss.
- So, we'll take the negative of the log of the probability assigned to the correct answer as the loss

Logs

Log of a value between 0 and 1 ranges from -infinity to 0.

Taking the negative of that it ranges from infinity to 0.

Low prob → large value
High prob → small value



Cross Entropy Loss

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y})\end{aligned}$$

$$L_{CE}(w,b) = -\left[y\log\sigma(w\cdot x+b) + (1-y)\log\left(1-\sigma(w\cdot x+b)\right)\right]$$

Negative of the log probability the model assigns to the correct answer.

Multiclass Models

- Switch to “Multinomial logistic regression”
  - Softmax
- We’ll still assume we have objects represented as vectors of features, and weights on the features, and a scoring function.

Softmax

$$p(y = c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

Softmax

$$p(y=c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

This gives us a score for a single class *c*.

Result is a normalized probability distribution over the classes.

This sums the scores for all the classes.

Note that the feature weights and bias term are now indexed per class.

Softmax Loss

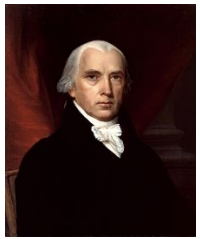
- So what's the loss for the Softmax?
  - We'll use the same Cross-Entropy idea
  - The probability assigned by the model to the correct class
  - Then use the negative log probability of that
- In general, CE is a measure of the distance between two probability distributions.
  - In our case, between the system output distribution and the correct one.
  - But in our applications the correct one, is always just one 1, and a bunch of zeroes. So CE collapses to focus on that one 1.

Federalist Example

$$p(y=c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

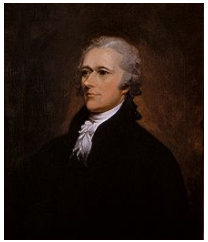
.00039

P(Jay | doc) = .00039/(.00039+.0409+.0000017)
= .00946



.0409

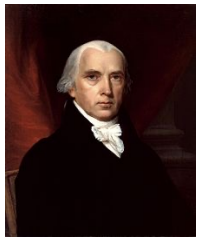
P(Madison | doc) = .0409/(.00039+.0409+.0000017)
= .99

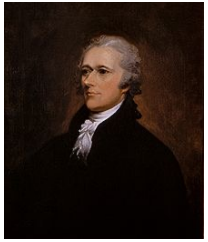


.0000017

P(Hamilton | doc) = .0000017/(.00039+.0409+.0000017)
= .000041

Federalist Example





System output is [0.00946, 0.99, 0.000041]

Correct answer is [0, 1, 0]

CE loss is -log P(0.99) = 0.004 (for log10)

Learning

- We want to find the weights that minimize the average loss across an entire training set.

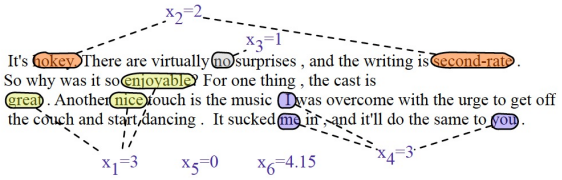
$$
\begin{aligned}
Cost(w,b) &= \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)
\end{aligned}
$$

Learning

- We'll do this by starting with a random set of weights and then iteratively updating those weights to lower this cost.

$$
\begin{aligned}
Cost(w,b) &= \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)
\end{aligned}
$$

Scoring



[3,2,1,3,0,4.15]

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Learning

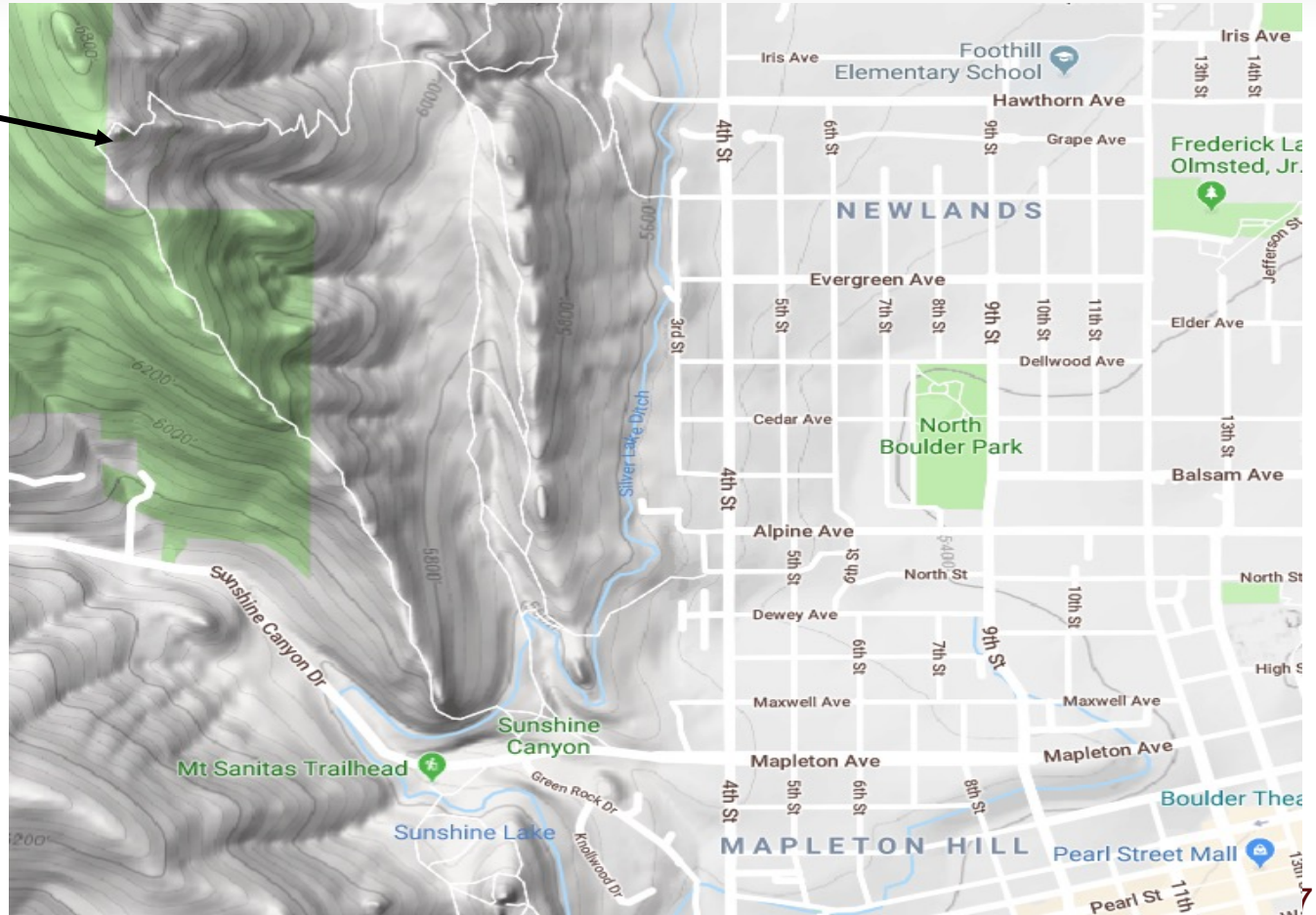
$$w_{t+1} = w_t - \mu\Delta$$

Derivatives

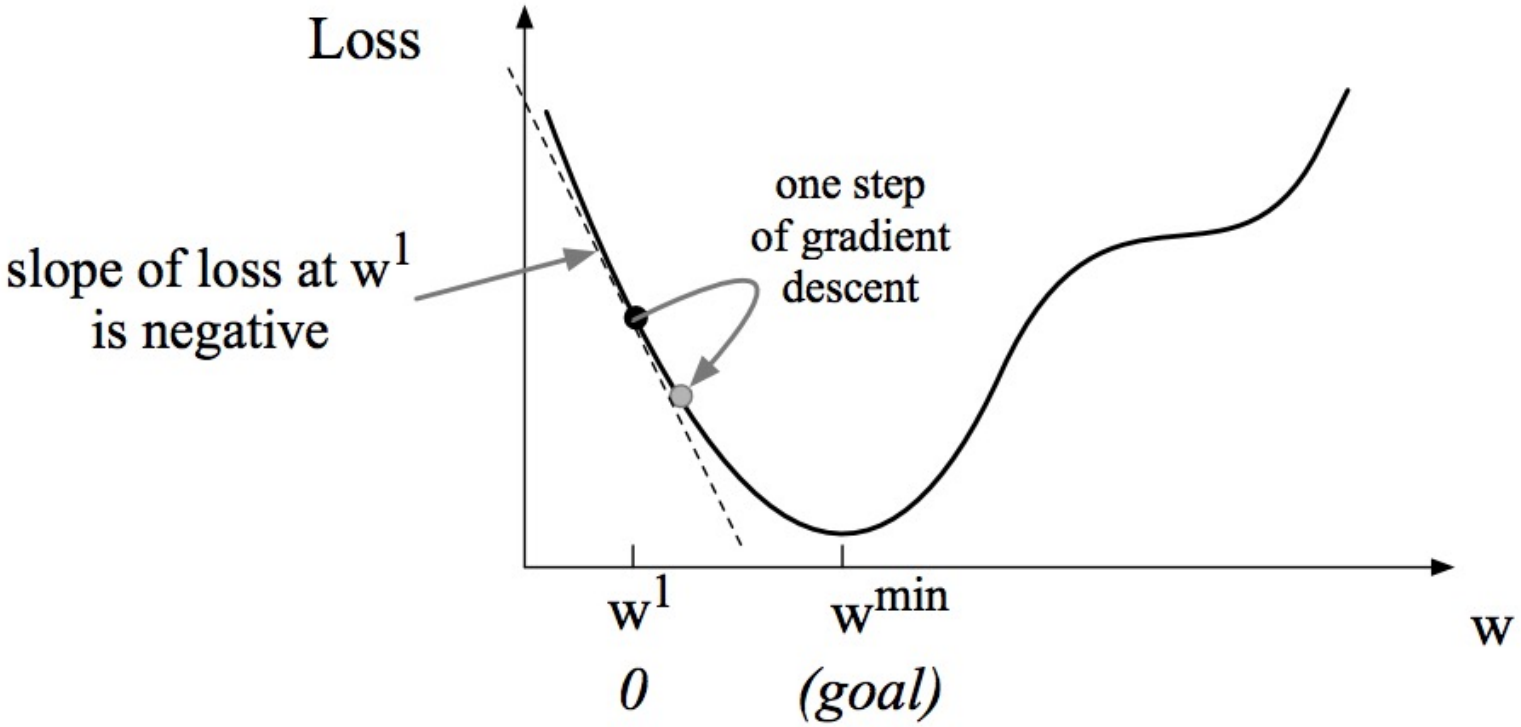
- Fortunately, basic calculus tells us how to do that.
- The derivative of the loss function with respect to the weights tells us the direction and magnitude of change we should make to each weight.
  - I.e. The “slope” of the loss at a particular point.

Motivation

This is you.
Find the fastest
way down.



For a Single Weight/Feature



Partial Derivative

- Of course, in real applications we have many features/weights not just one.
- So we need the vector of the partial derivatives of the loss wrt the weights
  - Call that vector the gradient

Partial Derivative

$$L_{CE}(w,b) = -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))]$$

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

If we have *n* weights, we end up with *n* partial derivatives. The vector consisting of those derivatives is called the gradient. We'll use the gradient to update all the weights.

CE Loss Partial Derivative

- Looks like gobbledygook, but it matches our earlier intuitions about loss functions and what need to learn
  - Is a weight too high or too low?
  - How far off is it?
  - How much should it change?

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Computed answer Right answer

Take the difference

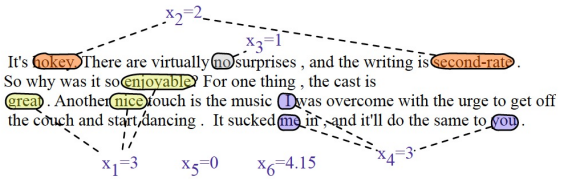
Multiply by the value of the input feature for the weight being adjusted

CE Loss Partial Derivative

- Need to do that for each of the features/weights for a single training example.
- Result is a vector of partial derivatives (the gradient), one for each weight.

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Scoring



[3,2,1,3,0,4.15]

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Updates

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

$$w_{t+1} = w_t - \mu\Delta$$

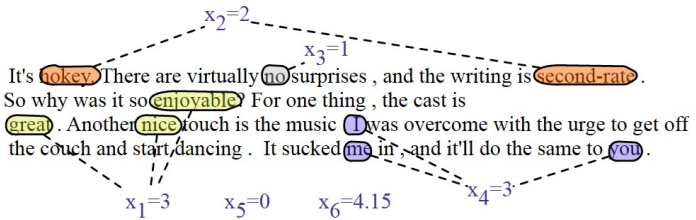
SGD

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(n)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(n)}$

$\theta \leftarrow 0$
repeat T times
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
  Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
  Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
  $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss ?
  $\theta \leftarrow \theta - \eta\, g$ # go the other way instead
return θ

Scoring



[3,2,1,3,0,4.15]

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Example

- Current weight vector
  - [2.5, -5, -1.2, 0.5, 2.0, 0.7]
- Input vector (x)
  - [3, 2, 1, 3, 0, 4.15]
- Computed answer is: 0.69
- Correct answer is 1.

Example

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\underbrace{\sigma(w \cdot x + b) - y}]x_j$$

- 0.31

Input: [3, 2, 1, 3, 0, 4.15]

Gradient:

[-0.31 * 3, -0.31 * 2, -0.31 * 1, -0.31 * 3, -0.31*0, -0.31*4.15]

=[-0.93, -0.62, -0.31, -0.93, 0 , -1.28]

Update the Weights

- Weights_t:
  - [2.5, -5, -1.2, 0.5, 2.0, 0.7]

$$w_{t+1} = w_t - \mu\Delta$$

- Gradient:

  =[-0.93, -0.62, -0.31, -0.93, 0 , -1.28]

- Assume learning rate of 1 for simplicity
- Weights_t+1:
- [3.42687096, -4.38208602, -0.89104301, 1.42687096, 2. , 1.9821715]

Do it Again

- Input vector (same)
  - [3, 2, 1, 3, 0, 4.15]
- New weights
  - [2.92, -4.72, -1.06, 0.92, 2.0, 1.281]
- Score
  - σ([2.92, -4.72, -1.06, 0.92, 2.0, 1.281] $\cdot$ [3,2,1,3,0,4.15] +.1)
  - = σ(6.43)
  - = .998

What About at the Start?

Example

- Start with zero weight vector
  - [0, 0, 0, 0, 0, 0]
- Input vector (x)
  - [3, 2, 1, 3, 0, 4.15]
- Computed answer is:…

$$P(y=1) = \sigma(w \cdot x + b) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

Example

- Current weight vector
  - [0, 0, 0, 0, 0, 0]
- Input vector (x)
  - [3, 2, 1, 3, 0, 4.15]
- Computed answer is… $\sigma(0 + .1) = 0.52$

Example

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

$$.52 - 1 = -.48$$

Gradient vector is

[-0.48*3, -0.48*2, -0.48*1, -0.48*3, -0.48*0, -0.48*4.15]

=[-1.44, -0.96, -0.48, -1.44, 0, -1.992]

Update the Weights

- Weights_t:
  - [0,0,0,0,0,0]
- Gradient:

$$w_{t+1} = w_t - \mu\Delta$$

=[-1.44, -0.96, -0.48, -1.44, 0, -1.992]

- Assume learning rate of 1 for simplicity
- Weights_t+1:
- =[1.44, 0.96,0.48,1.44, 0,1.992]

Redo the Example

- Input vector (same)
  - [3,2,1,3,0,4.15]
- Weights (new)

  = [1.44, 0.96,0.48,1.44, 0,1.992]
- Score
  - Weights . input + .1 = 19.4
  - $\sigma(19.4) = .999$

But…

- There's that pesky bias term. That's also a parameter and we forgot to update it as well.
- Standard approach is to treat the bias term as an additional feature and extend the representation of the input with an additional dimension with a value of 1.
- Update the bias term along with the weights.

Example

- Current weight vector
  - [2.5, -5, -1.2, 0.5, 2.0, 0.7]
- Input vector (x)
  - [3,2,1,3,0,4.15]

Example

- Current weight vector
  - [2.5, -5, -1.2, 0.5, 2.0, 0.7, 0.1]
- Input vector (x)
  - [3,2,1,3,0,4.15, 1]

SGD

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(n)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(n)}$

$\theta \leftarrow 0$
repeat T times
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
  Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
  Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
  $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss ?
  $\theta \leftarrow \theta - \eta\, g$ # go the other way instead
return θ

Better Training

- Batch training
  - Process each example in the training set and accumulate the gradients
  - Do a single update
  - Repeat
- Minibatch training
  - Select N examples and proceed as with batch training
  - Update after each mini-batch
  - N is chosen to maximize parallelism
- Adjust the learning rate during training.