

Natural Language Processing (AI615)

Prepared by Dr. Ashraf Elsayed

Lecture 7





مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY

Logistic Regression

Lecture 7

Prepared by
Dr. Ashraf Elsayed

www.midocean.ae

The objectives of this lecture is to:

- Understand Generative and Discriminative Classifiers
- Define Logistic Regression
- Understand cross-entropy loss and stochastic gradient descent
- Understand and explain Multinomial Logistic Regression
- Define and explain softmax function

Generative and Discriminative Classifiers

- Naive Bayes is a **generative** classifier
- by contrast:
- Logistic regression is a **discriminative** classifier

Finding the correct class c from a document d in Generative vs Discriminative Classifiers

- Naive Bayes

$$\hat{c} = \operatorname{argmax}_{c \in C} \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

- Logistic Regression

$$\hat{c} = \operatorname{argmax}_{c \in C} \overbrace{P(c|d)}^{\text{posterior}}$$

Components of a probabilistic machine learning classifier

Given m input/output pairs $(x^{(i)}, y^{(i)})$:

1. A **feature representation** of the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \dots, x_n]$. Feature j for input $x^{(i)}$ is x_j , more completely $x_j^{(i)}$, or sometimes $f_j(x)$.
2. A **classification function** that computes $\hat{y}$, the estimated class, via $p(y|x)$, like the **sigmoid** or **softmax** functions.
3. An objective function for learning, like **cross-entropy loss**.
4. An algorithm for optimizing the objective function: **stochastic gradient descent**.

The two phases of logistic regression

- **Training:** we learn weights w and b using **stochastic gradient descent** and **cross-entropy loss**.
- **Test:** Given a test example x we compute $p(y|x)$ using learned weights w and b , and return whichever label ($y = 1$ or $y = 0$) is higher probability

Classification Reminder

- Positive/negative sentiment
- Spam/not spam
- Authorship attribution

Text Classification: definition

- *Input:*
 - a document x
 - a fixed set of classes $C = \{c_1, c_2, \dots, c_J\}$
- *Output:* a predicted class $\hat{y} \in C$

Binary Classification in Logistic Regression

- Given a series of input/output pairs:
 - $(x^{(i)}, y^{(i)})$
- For each observation $x^{(i)}$
 - We represent $x^{(i)}$ by a **feature vector** $[x_1, x_2, \dots, x_n]$
 - We compute an output: a predicted class $\hat{y}^{(i)} \in \{0, 1\}$

Features in logistic regression

- For feature x_i , weight w_i tells is how important is x_i
 - x_i = "review contains 'awesome'": $w_i = +10$
 - x_j = "review contains 'abysmal'": $w_j = -10$
 - x_k = "review contains 'mediocre'": $w_k = -2$

Logistic Regression for one observation x

- Input observation: vector $x = [x_1, x_2, \dots, x_n]$
- Weights: one per feature: $W = [w_1, w_2, \dots, w_n]$
 - Sometimes we call the weights $\theta = [\theta_1, \theta_2, \dots, \theta_n]$
- Output: a predicted class $\hat{y} \in \{0, 1\}$

(multinomial logistic regression: $\hat{y} \in \{0, 1, 2, 3, 4\}$)

↓
MaxEnt

How to do classification

- For each feature x_i , weight w_i tells us importance of x_i
 - (Plus we'll have a bias b)
- We'll sum up all the weighted features and the bias

$$z = \sum_{i=1}^n w_i x_i + b$$

- If this sum is high, we say $y=1$; if low, then $y=0$

$$z = w \cdot x + b$$

But we want a probabilistic classifier

- We need to formalize “sum is high”.
- We’d like a principled classifier that gives us a probability, just like Naive Bayes did
- We want a model that can tell us:
 $p(y=1 | x; \theta)$
 $p(y=0 | x; \theta)$

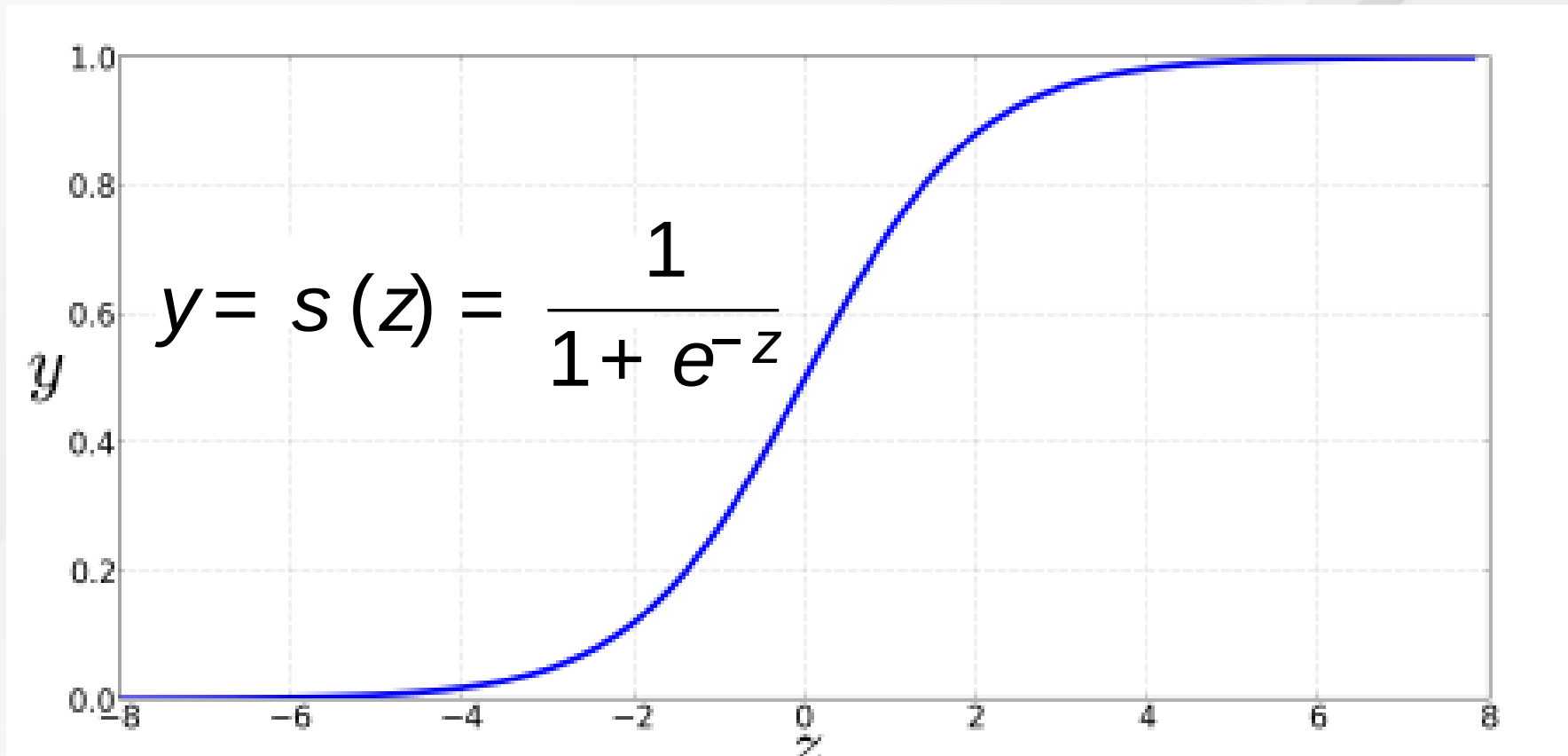
The problem: z isn't a probability, it's just a number!

$$z = w \cdot x + b$$

- Solution: use a function of z that goes from 0 to 1

$$y = s(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-z)}$$

The very useful sigmoid or logistic function



Idea of logistic regression

- We'll compute $w \cdot x + b$
- And then we'll pass it through the sigmoid function:

$$\sigma(w \cdot x + b)$$

- And we'll just treat it as a probability

Making probabilities with sigmoids

$$\begin{aligned} P(y = 1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1 + \exp(-(w \cdot x + b))} \end{aligned}$$

$$\begin{aligned} P(y = 0) &= 1 - \sigma(w \cdot x + b) \\ &= 1 - \frac{1}{1 + \exp(-(w \cdot x + b))} \\ &= \frac{\exp(-(w \cdot x + b))}{1 + \exp(-(w \cdot x + b))} \end{aligned}$$

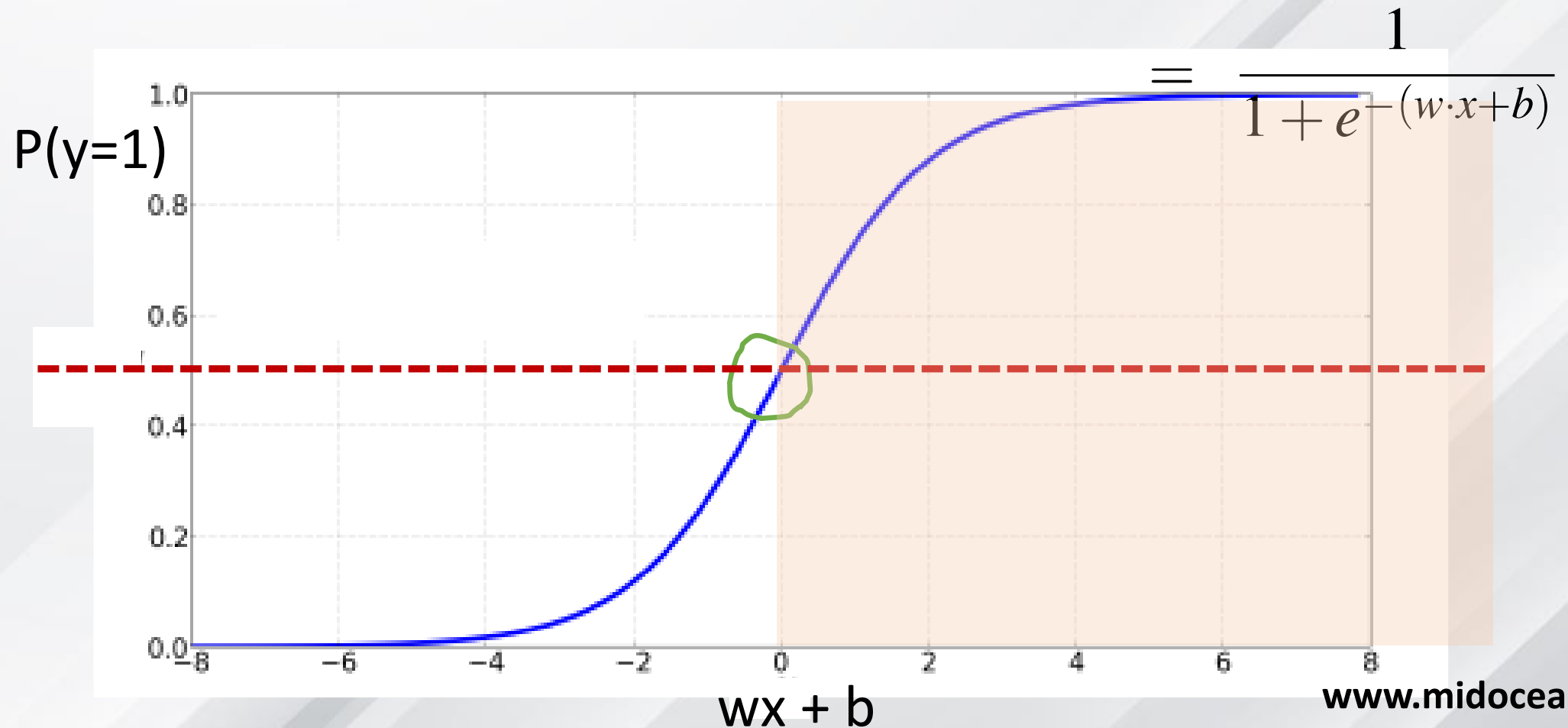
Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

0.5 here is called the **decision boundary**

The probabilistic classifier

$$P(y = 1) = \sigma(w \cdot x + b)$$



Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases} \quad \begin{array}{l} \text{if } w \cdot x + b > 0 \\ \text{if } w \cdot x + b \leq 0 \end{array}$$

Sentiment example: does $y=1$ or $y=0$?

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable? For one thing, the cast is great. Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

$x_2=2$
 $x_3=1$
 $x_1=3$ $x_5=0$ $x_6=4.19$ $x_4=3$

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)	$\ln(66) = 4.19$

Classifying sentiment for input x

Var	Definition	Value in Fig. 5.2
x_1	$\text{count}(\text{positive lexicon}) \in \text{doc}$	3
x_2	$\text{count}(\text{negative lexicon}) \in \text{doc}$	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	$\text{count}(\text{1st and 2nd pronouns}) \in \text{doc}$	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	$\log(\text{word count of doc})$	$\ln(66) = 4.19$

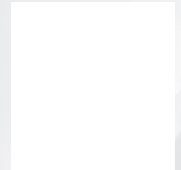
Suppose $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

$b = 0.1$

Classifying sentiment for input x

$$\begin{aligned} p(+ | x) &= P(Y = 1 | x) = s(w \cdot x + b) \\ &= s([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= s(.833) \\ &= 0.70 \end{aligned}$$

$$\begin{aligned} p(- | x) &= P(Y = 0 | x) = 1 - s(w \cdot x + b) \\ &= 0.30 \end{aligned}$$



We can build features for logistic regression for any classification task: *period disambiguation*

This ends in a period.  End of sentence
The house at 465 Main St. is new.  Not end

$$\begin{aligned}
 x_1 &= \begin{cases} 1 & \text{if "Case}(w_i) = \text{Lower"} \\ 0 & \text{otherwise} \end{cases} \\
 x_2 &= \begin{cases} 1 & \text{if " } w_i \in \text{AcronymDict"} \\ 0 & \text{otherwise} \end{cases} \\
 x_3 &= \begin{cases} 1 & \text{if " } w_i = \text{St. \& Case}(w_{i-1}) = \text{Cap"} \\ 0 & \text{otherwise} \end{cases}
 \end{aligned}$$

Classification in (**binary**) logistic regression: summary

- Given:
 - a set of classes: (+ sentiment, - sentiment)
 - a vector $\mathbf{x}$ of features $[x_1, x_2, \dots, x_n]$
 - $x_1 = \text{count}(\text{"awesome"})$
 - $x_2 = \log(\text{number of words in review})$
 - A vector $\mathbf{w}$ of weights $[w_1, w_2, \dots, w_n]$
 - w_i for each feature f_i

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

Wait, where did the W 's come from?

- Supervised classification:
- We know the correct label y (either 0 or 1) for each x .
- But what the system produces is an estimate, $\hat{y}$
- We want to set w and b to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.
- We need a distance estimator: a **loss function** or a **cost function**
- We need an optimization algorithm to update w and b to minimize the loss.

Learning components

- A loss function:
 - **cross-entropy loss**
- An optimization algorithm:
 - **stochastic gradient descent**

The distance between $\hat{y}$ and y

- We want to know how far is the classifier output:
- $\hat{y} = \sigma(w \cdot x + b)$
- from the true output:
- y [= either 0 or 1]
- We'll call this difference:
- $L(\hat{y}, y)$ = how much $\hat{y}$ differs from the true y

Intuition of negative log likelihood loss = cross-entropy loss

- A case of conditional maximum likelihood estimation
- We choose the parameters w, b that maximize the log probability of the true y labels in the training data given the observations x

Deriving cross-entropy loss for a single observation x

- **Goal:** maximize probability of the correct label $p(y|x)$
- Since there are only 2 discrete outcomes (0 or 1) we can express the probability $p(y|x)$ from our classifier (the thing we want to maximize) as

- noting:
$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$
- if $y=1$, this simplifies to $\hat{y}$
- if $y=0$, this simplifies to $1 - \hat{y}$

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

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

- Now take the log of both sides (mathematically handy)

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

- Whatever values maximize $\log p(y|x)$ will also maximize $p(y|x)$

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Maximize:

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

- Now flip sign to turn this into a loss: something to minimize
- **Cross-entropy loss** (because is formula for cross-entropy($y, \hat{y}$))

Minimize:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

- Or, plugging in definition of $\hat{y}$:

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

Let's see if this works for our sentiment example

- We want loss to be:
 - smaller if the model estimate is close to correct
 - bigger if model is confused
- Let's first suppose the true label of this is $y=1$ (positive)

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Let's see if this works for our sentiment example

- True value is $y=1$. How well is our model doing?

$$\begin{aligned} p(+|x) = P(Y=1|x) &= s(w \cdot x + b) \\ &= s([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= s(.833) \\ &= 0.70 \end{aligned}$$

- Pretty well! What's the loss?

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log(.70) \\ &= .36 \end{aligned}$$

Let's see if this works for our sentiment example

- Suppose true value instead was $y=0$.

$$\begin{aligned} p(-|x) &= P(Y = 0|x) = 1 - s(w \cdot x + b) \\ &= 0.30 \end{aligned}$$

- What's the loss?

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log (1 - \sigma(w \cdot x + b))] \\ &= -\log (.30) \\ &= 1.2 \end{aligned}$$

Let's see if this works for our sentiment example

- The loss when model was right (if true $y=1$)

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log(.70) \\ &= .36 \end{aligned}$$

- Is lower than the loss when model was wrong (if true $y=0$):

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log (1 - \sigma(w \cdot x + b))] \\ &= -\log (.30) \\ &= 1.2 \end{aligned}$$

- Sure enough, loss was bigger when model was wrong!

Stochastic Gradient Descent

- Stochastic Gradient Descent is used to find the weights (W, b) that minimize the loss.

Gradients

- The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.
- **Gradient Descent:** Find the gradient of the loss function at the current point and move in the **opposite** direction.

How much do we move in that direction ?

- The value of the gradient (slope in our example) $\frac{d}{dw} L(f(x; w), y)$ weighted by a **learning rate** η
- Higher learning rate means move w faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

What are these partial derivatives for logistic regression?

The loss function

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

The elegant derivative of this function

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

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 $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \dots, y^{(m)}$

$\theta \leftarrow 0$

repeat til done

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

1. Optional (for reporting): # How are we doing on this tuple?

 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 $y^{(i)}$?

2. $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?

3. $\theta \leftarrow \theta - \eta g$ # Go the other way instead

return θ

Hyperparameters

- The learning rate η is a **hyperparameter**
 - too high: the learner will take big steps and overshoot
 - too low: the learner will take too long
- Hyperparameters:
- Briefly, a special kind of parameter for an ML model
- Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.

Working through an example

- One step of gradient descent
- A mini-sentiment example, where the true $y=1$ (positive)
- Two features:

$x_1 = 3$ (count of positive lexicon words)

$x_2 = 2$ (count of negative lexicon words)

Assume 3 parameters (2 weights and 1 bias) in Θ^0 are zero:

$$w_1 = w_2 = b = 0$$

$$\eta = 0.1$$

Example of gradient descent

- Update step for update W is:

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

- where $\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$
- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial b} \end{bmatrix}$$

$$w_1 = w_2 = b = 0;$$

$$x_1 = 3; \quad x_2 = 2$$

Example of gradient descent

$$w_1 = w_2 = b = 0;$$

$$x_1 = 3; \quad x_2 = 2$$

- Update step for update W is:

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

- where $\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$
- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix}$$

Example of gradient descent

$$w_1 = w_2 = b = 0;$$

$$x_1 = 3; \quad x_2 = 2$$

- Update step for update W is:

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

- where $\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$
- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}$$

Example of gradient descent

- Update step for update W is:

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

- where $\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$
- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} =$$

$$w_1 = w_2 = b = 0;$$

$$x_1 = 3; \quad x_2 = 2$$

Example of gradient descent

- Update step for update W is:

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

- where $\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$
- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

$$w_1 = w_2 = b = 0;$$

$$x_1 = 3; \quad x_2 = 2$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector W^1 by moving W^0 in the opposite direction from the gradient:

$$W^{t+1} = W^t - h \frac{d}{dw} L(f(x; w), y) \quad \eta = 0.1;$$

$$W_1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Note that enough negative examples would eventually make w_2 negative

Mini-batch training

- Stochastic gradient descent chooses a single random example at a time.
- That can result in choppy movements
- More common to compute gradient over batches of training instances.
- **Batch training:** entire dataset
- **Mini-batch training:** m examples (512, or 1024)

Multinomial Logistic Regression

- Often we need more than 2 classes
 - Positive/negative/neutral
 - Parts of speech (noun, verb, adjective, adverb, preposition, etc.)
 - Classify emergency SMSs into different actionable classes
- If >2 classes we use **multinomial logistic regression**
 - = Softmax regression
 - = Multinomial logit
 - = (defunct names : Maximum entropy modeling or MaxEnt)
- So "logistic regression" will just mean binary (2 output classes)

Multinomial Logistic Regression

- The probability of everything must still sum to 1

$$P(\text{positive}|\text{doc}) + P(\text{negative}|\text{doc}) + P(\text{neutral}|\text{doc}) = 1$$

- Need a generalization of the sigmoid called the **softmax**
 - Takes a vector $z = [z_1, z_2, \dots, z_k]$ of k arbitrary values
 - Outputs a probability distribution
 - each value in the range $[0,1]$
 - all the values summing to 1

The softmax function

Turns a vector $z = [z_1, z_2, \dots, z_k]$ of k arbitrary values into probabilities

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k$$

The denominator $\sum_{i=1}^k e^{z_i}$ is used to normalize all the values into probabilities.

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

The softmax function

- Turns a vector $z = [z_1, z_2, \dots, z_k]$ of k arbitrary values into probabilities

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

$$[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

Softmax in multinomial logistic regression

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

Input is still the dot product between weight vector w and input vector x
But now we'll need separate weight vectors for each of the K classes.

Features in binary versus multinomial logistic regression

- Binary: positive weight $\rightarrow y=1$ neg weight $\rightarrow y=0$

$$x_5 = \begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \quad w_5 = 3.0$$

- Multinomial: separate weights for each class:

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	-5.3



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY



Thank You

