Logistic regression and gradient descent

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

11/03 - Gustave Cortal

Generative and discriminative classifiers

Naive Bayes is a **generative** classifier

by contrast:

Logistic regression is a **discriminative** classifier

Generative and discriminative classifiers

Suppose we're distinguishing cat from dog images

Generative classifier

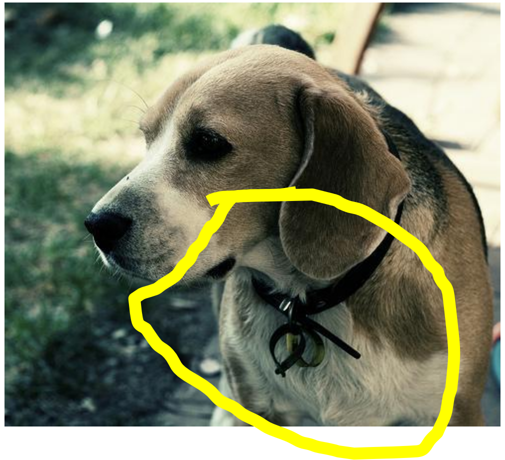
- Build a model of what's in a cat image
  - knows about ears, eyes, nose, etc.
  - assigns a probability to any image:
    - how cat-y is this image?

Also build a model for dog images

Given a new image: **run both models and see which one fits better**

Discriminative classifier

Just try to distinguish dogs from cats



Oh look, dogs have collars!
Let's ignore everything else

Generative vs discriminative classifiers

Naive Bayes

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

Logistic Regression

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \ \overset{\text{posterior}}{P(c|d)}$$

Components of a machine learning classifier

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

1. A **feature representation** of the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \ldots, x_n]$. 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 in logistic regression

Text classification: definition

- Input:
    - a document ***d***
    - a fixed set of classes $\boldsymbol{C = \{c_1, c_2, \ldots, c_J\}}$
- Output: a predicted class $\boldsymbol{c \in C}$

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** $[x_1, x_2, \ldots, x_n]$
- 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 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

Input observation: vector $x = [x_1, x_2, \ldots, x_n]$

Weights: one per feature: $W = [w_1, w_2, \ldots, w_n]$

- Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$

Output: a predicted class $\hat{y} \in \{0,1\}$

We want a probabilistic classifier

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

We need to formalize “sum is high”

We’d like a classifier that gives us a probability (like Naive Bayes)

We want a model that can tell us:

$p(y=1|x; \theta)$

$p(y=0|x; \theta)$

Turn *z* into a probability

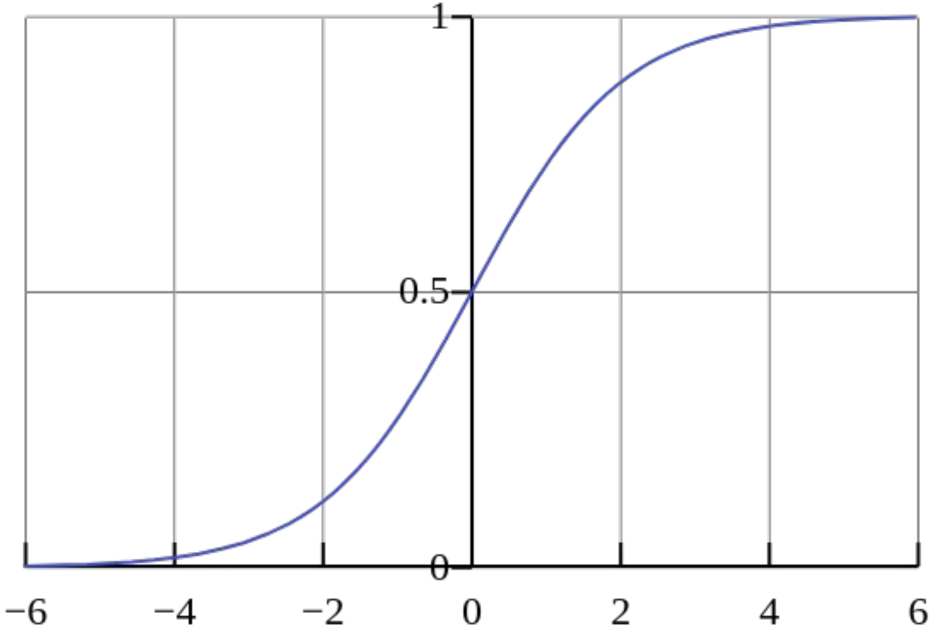
$$z = w \cdot x + b$$

Use a function of z that goes from 0 to 1: the **sigmoid** function

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

The sigmoid function

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



Idea of logistic regression

(1) Compute w·x+b
(2) Pass it through the sigmoid function σ(w·x+b)

→ Treat the result 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

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

The **decision boundary** is 0.5

Turning a probability into a classifier

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

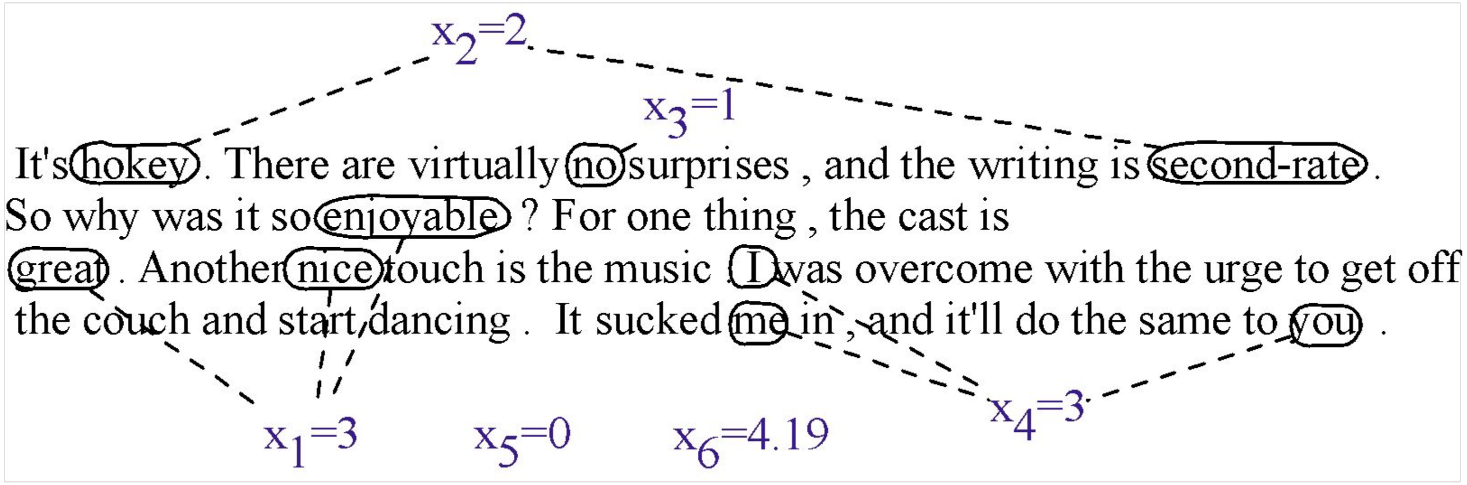
if w·x+b > 0

if w·x+b ≤ 0

Example for sentiment classification

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 .


x2=2
x3=1
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 .
x1=3
x5=0
x6=4.19
x4=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)	$\ln(66) = 4.19$

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)	$\ln(66) = 4.19$

Suppose w = $[2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ b = 0.1

Classifying sentiment for input *x*

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

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

Cross-entropy loss

How to get *w*?

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**

Intuition of negative log likelihood loss = cross-entropy loss

We choose the parameters *w* and *b* that maximize

- the log probability
- of the true *y* labels in the training data
- given the observations *x*

Deriving cross-entropy loss

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

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

noting:

if y=1, this simplifies to $\hat{y}$

if y=0, this simplifies to 1– $\hat{y}$

Deriving cross-entropy loss

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

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

Deriving cross-entropy loss

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

Maximize:

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

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

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

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

Sentiment example

We want loss to be:

- smaller if the model estimate is close to correct
- bigger if model is confused

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 . $\longrightarrow$ **y=1 (positive)**

Sentiment example

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

$$\begin{aligned} p(+|x) = P(Y=1|x) &= s(w\cdot x+b) \\ &= s([2.5,-5.0,-1.2,0.5,2.0,0.7]\cdot[3,2,1,3,0,4.19]+0.1) \\ &= s(.833) \\ &= 0.70 \end{aligned} \tag{5.6}$$

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

Sentiment example

Instead, suppose true value was y=0

$$p(-|x) = P(Y=0|x) = 1 - s(w \cdot x + b)$$
$$= 0.30$$

What's the loss?

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

Stochastic Gradient Descent

Minimize the loss

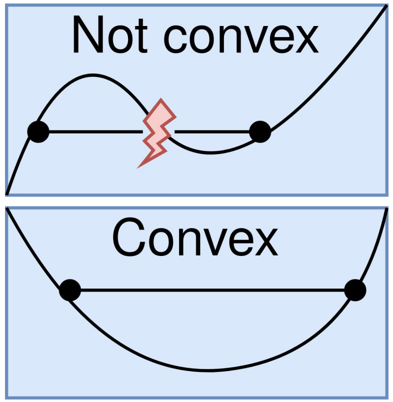
Let's make explicit that the loss function is parameterized by weights θ=(w,b)

- And we'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious

We want the weights that minimize the loss, averaged over all examples:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)}; \theta), y^{(i)})$$

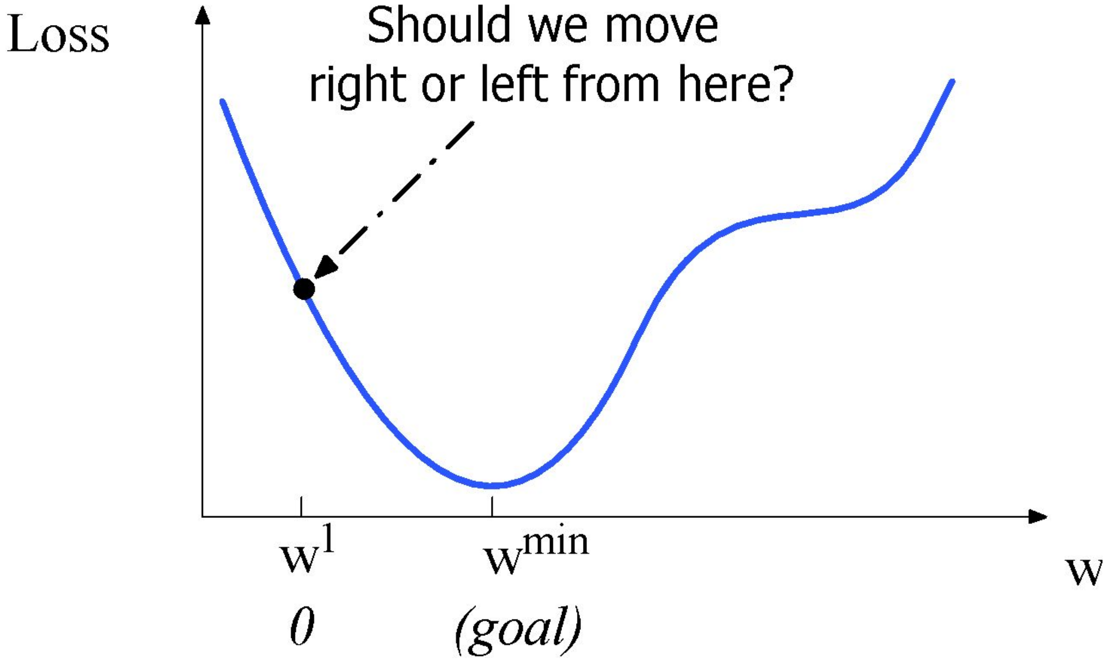
Minimize the loss



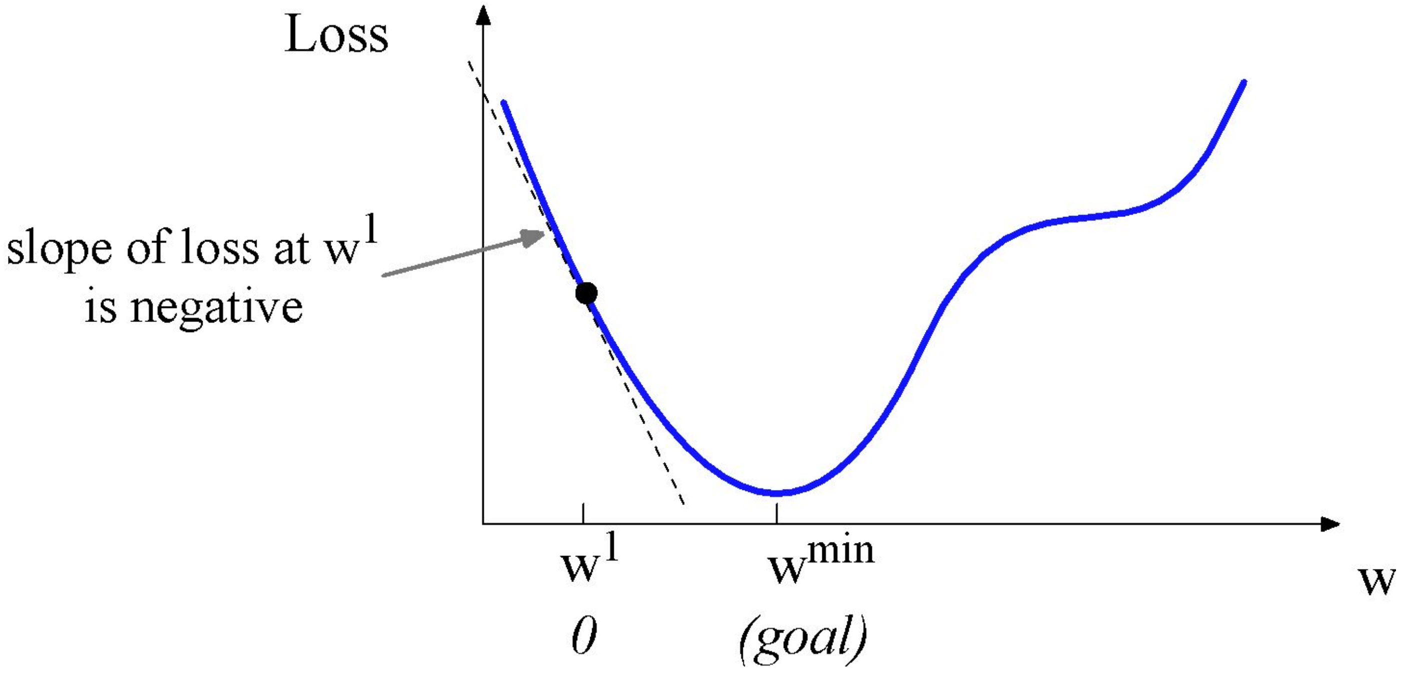
For logistic regression, loss function is **convex**

- A convex function has just one minimum
- Gradient descent starting from any point is guaranteed to find the minimum
    - Loss for neural networks is non-convex

Loss for a scalar *w*

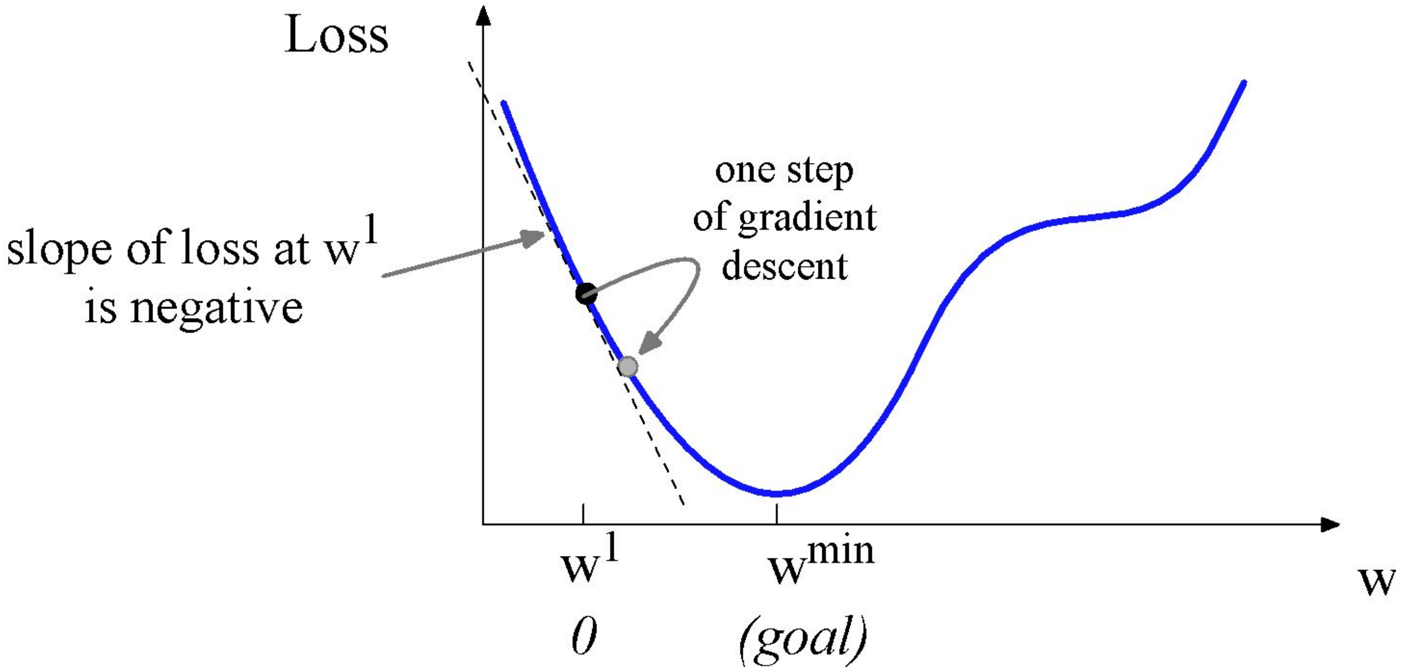


Loss for a scalar ***w***



$\longrightarrow$ move positive

Loss for a scalar *w*


Loss
slope of loss at w^1 is negative
one step of gradient descent
w^1
0
w^{min}
(goal)
w

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 ?

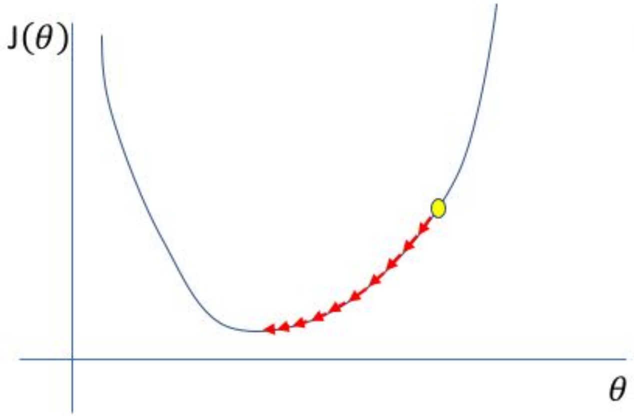
$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x;w), y)$$

h is the **learning rate**

Hyperparameters

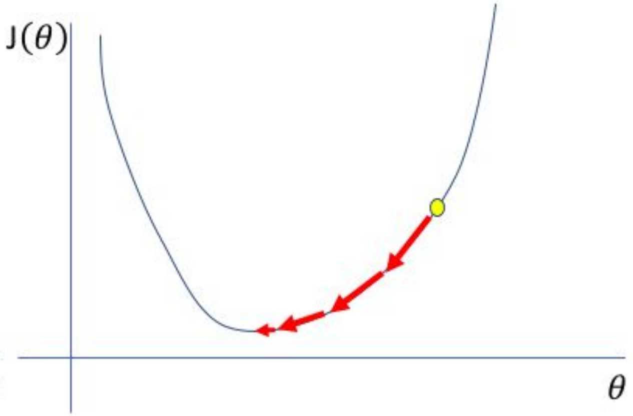
The learning rate *h* is a **hyperparameter**

Too low



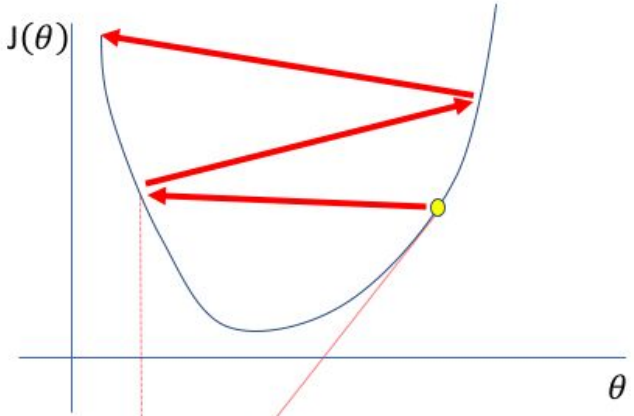
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

Real gradients

For each dimension w_i the gradient component i tells us the slope with respect to that variable

- “How much would a small change in w_i influence the total loss function L?”
- We express the slope as a partial derivative ∂ of the loss ∂w_i

The gradient is the a vector of these partials

What are these partial derivatives for logistic regression?

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

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

```
function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, $x$, $y$) returns $\theta$
    # where: L is the loss function
    #        f is a function parameterized by $\theta$
    #        x is the set of training inputs $x^{(1)}, x^{(2)}, ..., x^{(m)}$
    #        y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, ..., y^{(m)}$

$\theta \leftarrow 0$
repeat til done    # see caption
  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 $\theta$ to maximize loss?
    3. $\theta \leftarrow \theta - \eta\, g$                # Go the other way instead
return $\theta$
```

Stochastic Gradient Descent: an example

An example: one step of gradient descent

A mini-sentiment example, where 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:

$w_1 = w_2 = b = 0$
$h = 0.1$

An example: one step of a gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

Update step for update θ is:

$$q_{t+1} \;=\; q_t - h \,—L(f(x;q),y)$$

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

Gradient vector has 3 dimensions:

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

An example: one step of a gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

Update step for update θ is:

$$q_{t+1} = q_t - h - L(f(x;q), y)$$

where

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

Gradient vector has 3 dimensions:

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

An example: one step of a gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

Update step for update θ is:

$$q_{t+1} = q_t - h - L(f(x;q), y)$$

where

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

Gradient vector has 3 dimensions:

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

An example: one step of a gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

Update step for update θ is:

$$q_{t+1} = q_t - h -L(f(x;q), y)$$

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

Gradient vector has 3 dimensions:

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

An example: one step of a gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

Update step for update θ is:

$$q_{t+1} = q_t - h - L(f(x;q),y)$$

where

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

Gradient vector has 3 dimensions:

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

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

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

$$q_{t+1} = q_t - h -L(f(x;q), y) \qquad \eta = 0.1$$

$$\theta^1 =$$

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

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

$$q_{t+1} = q_t - h - L(f(x;q), y)$$

$\eta = 0.1$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

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

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

$$q_{t+1} = q_t - h -L(f(x;q),y) \qquad \eta = 0.1$$

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

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

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

$$q_{t+1} = q_t - h -L(f(x;q),y)$$

$\eta = 0.1$

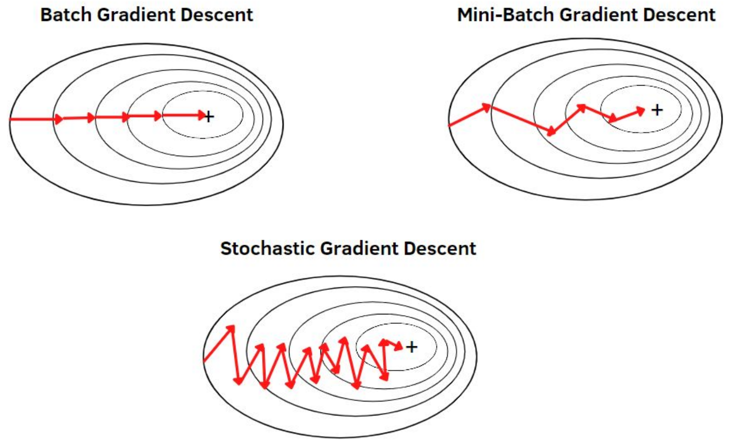
$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Mini-batch training

Stochastic gradient descent chooses a single random example at a time

→ result in choppy movements

More common to compute gradient over batches of training instances



Logistic regression: regularization

Overfitting

A model that perfectly match the training data has a problem

It will also **overfit** to the data, modeling noise

- A random word that perfectly predicts *y* (it happens to only occur in one class) will get a very high weight
- Failing to generalize to a test set without this word.

A good model should be able to **generalize**

Overfitting

+

This movie drew me in, and it'll do the same to you.

X1 = "this"
X2 = "movie
X3 = "hated"
X4 = "drew me in"

-

I can't tell you how much I hated this movie. It sucked.

X5 = "the same to you"
X7 = "tell you how much"

Regularization

Add a regularization term *R*(θ) to the loss function

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) - \alpha R(\theta)$$

Choose an *R*(θ) that penalizes large weights

- Intuition: fitting the data well with lots of big weights not as good as fitting the data a little less well, with small weights

L2 regularization (= ridge regression)

L2 norm $||\theta||_2$ is the **Euclidean distance** of θ to the origin

$$R(\theta) \;=\; ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

L2 regularized objective function:

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} \theta_j^2$$

L1 regularization (= lasso regression)

L1 norm ||θ|| is the sum of the absolute values of the weights

$$R(\theta) \;=\; ||\theta||_1 = \sum_{i=1}^{n} |\theta_i|$$

L1 regularized objective function:

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |\theta_j|$$

Multinomial logistic regression

Multinomial Logistic Regression

The probability of everything must still sum to 1

P(positive|doc) + P(negative|doc) + P(neutral|doc) = 1

Need a generalization of the sigmoid called the **softmax**

- Takes a vector *z* = [*z*1, *z*2, ..., *zk*] of *k* arbitrary values
- Outputs a probability distribution
  - each value in the range [0,1]
  - all the values summing to 1

Softmax function

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

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

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

Softmax function

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

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

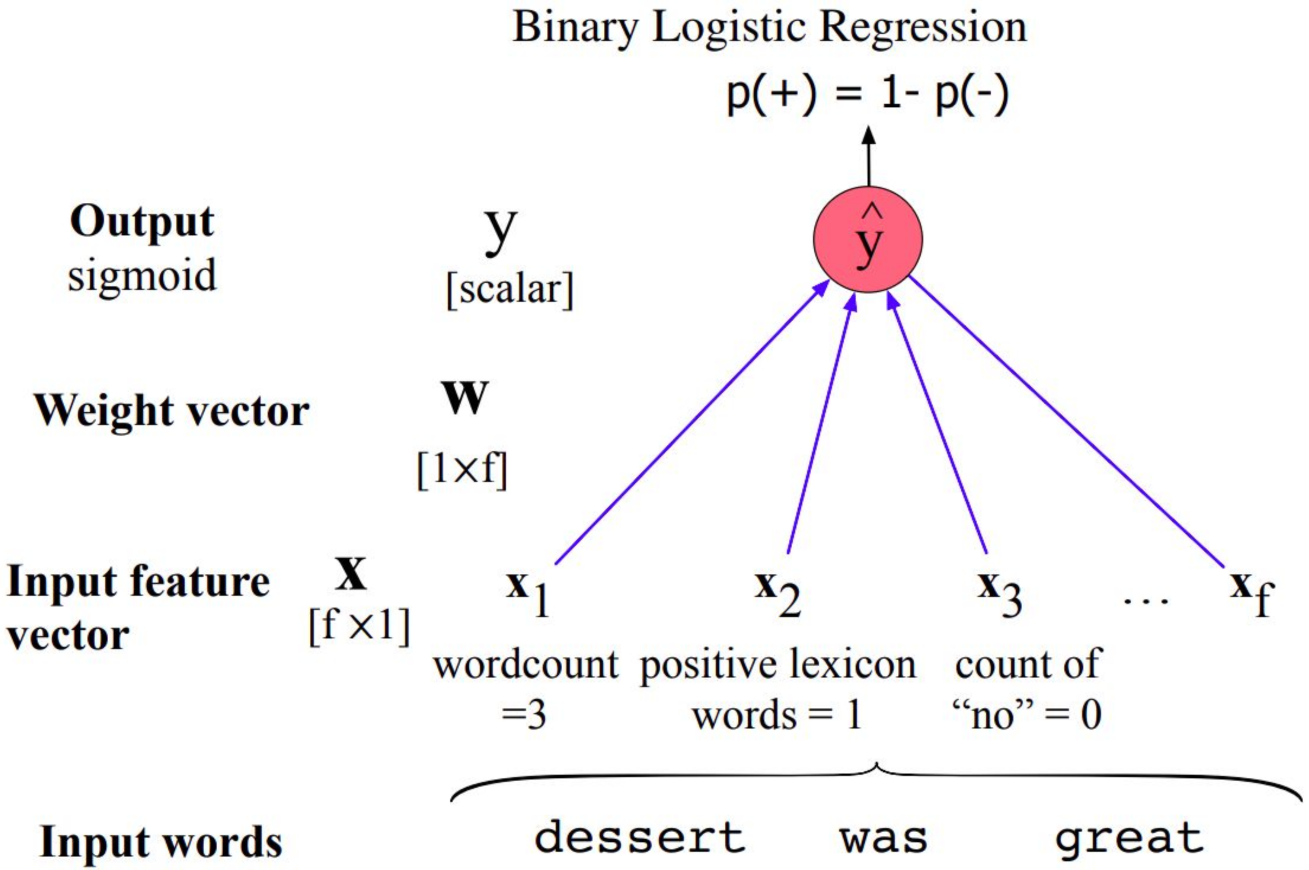
$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

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

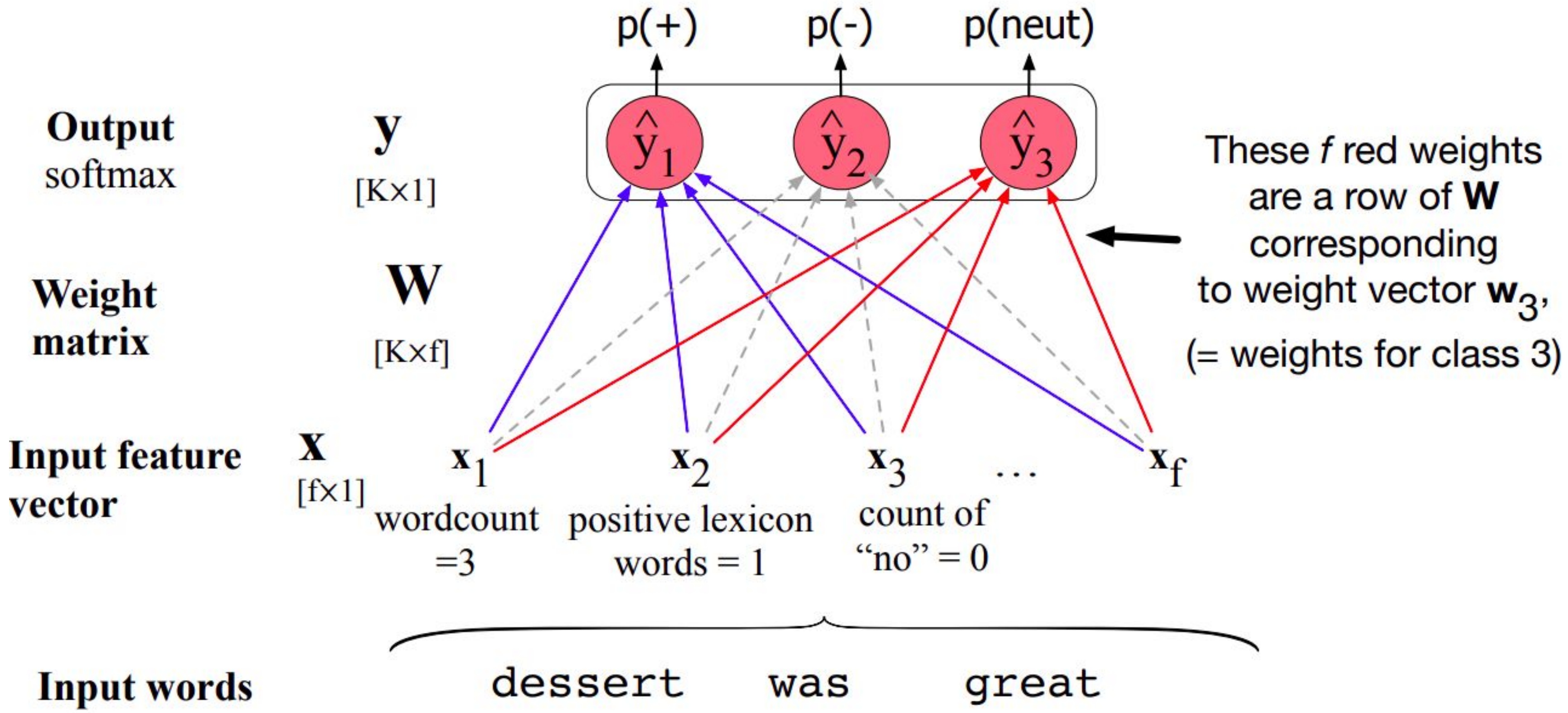
Softmax in multinomial logistic regression

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

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


Binary Logistic Regression
p(+) = 1- p(-)
Output
sigmoid
y
[scalar]
$\hat{y}$
Weight vector
w
[1×f]
Input feature vector
x
[f ×1]
x_1
x_2
x_3
…
x_f
wordcount =3
positive lexicon words = 1
count of "no" = 0
Input words
dessert was great

Multinomial Logistic Regression



p(+)
p(-)
p(neut)
Output
softmax
$\mathbf{y}$
[K×1]
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
These *f* red weights are a row of **W** corresponding to weight vector $\mathbf{w}_3$, (= weights for class 3)
Weight matrix
$\mathbf{W}$
[K×f]
Input feature vector
$\mathbf{x}$
[f×1]
x_1
x_2
x_3
…
x_f
wordcount =3
positive lexicon words = 1
count of "no" = 0
Input words
dessert was great