CS 2731
Introduction to Natural Language Processing

Session 5: Logistic regression, part 2

Michael Miller Yoder

September 11, 2024

University of Pittsburgh | **School of Computing and Information**

Course logistics

- [Homework 1](#) **due next Thu Sep 19**

  - Feel free to ask questions in the Canvas discussion forum, email Jayden or Michael

- [Project idea submission form](#) **due Thu Sep 19**

  - All students must submit one idea for credit

  - You can submit one of the example project ideas from the [website](#)

  - Ideas do not need to be fully developed

- Additional readings for Mon Sep 16:

  - Data statements (Bender & Friedman 2018)

  - Model cards (Mitchell et al. 2019)

Lecture overview: logistic regression part 2

- **Learning the weights for features in logistic regression**
  - Cross-entropy loss function
  - Stochastic gradient descent
  - Batch and mini-batch training
  - Regularization
  - Training multinomial logistic regression

Review: classification with logistic regression

1. What is the necessary format for the input to logistic regression? What will the output format be?

2. What is the equation for calculating $\hat{y}$, the predicted class from an input vector x?

Logistic regression: learning the weights

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:

$$y \qquad [= \text{either 0 or 1}]$$

We'll call this difference:

$$L(\hat{y}, y) = \text{how much } \hat{y} \text{ differs from the true } y$$

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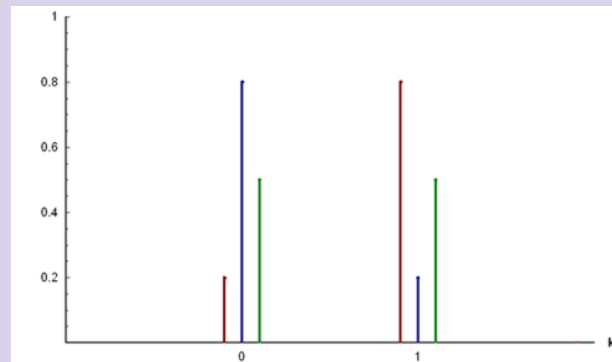
$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

noting:

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

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

$\hat{y}$=0.3

0 ————————————— 1

From the Bernoulli distribution, also expressed as:

$$p(y \mid x) = \begin{cases} \hat{y} & \text{if } y = 1 \\ 1 - \hat{y} & \text{if } y = 0 \end{cases}$$

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

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

Now take the log of both sides (mathematically handy)

Maximize: $\qquad \log p(y|x) \;=\; \log \left[\hat{y}^y \, (1-\hat{y})^{1-y} \right]$
$\qquad\qquad\qquad\quad =\; y \log \hat{y} + (1-y) \log(1-\hat{y})$

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

Deriving cross-entropy loss for a single observation x

Now flip the sign to turn this into a loss: something to minimize

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

Deriving cross-entropy loss for a single observation x

Now flip the sign to turn this into a loss: something to minimize

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

This happens to be the formula for cross-entropy, a measure of difference between distributions from information theory

Claude Shannon

Now flip the sign to turn this into a loss: something to minimize

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

Plugging in the 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 .

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

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

Pretty well! What's the loss?

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

Let's see if this works for our sentiment example

Suppose true value instead was y=0.

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

What's the loss?

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

Slide credit: Jurafsky & Martin

Let's see if this works for our sentiment example

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

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

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

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

Sure enough, loss was bigger when model was wrong!

Slide credit: Jurafsky & Martin
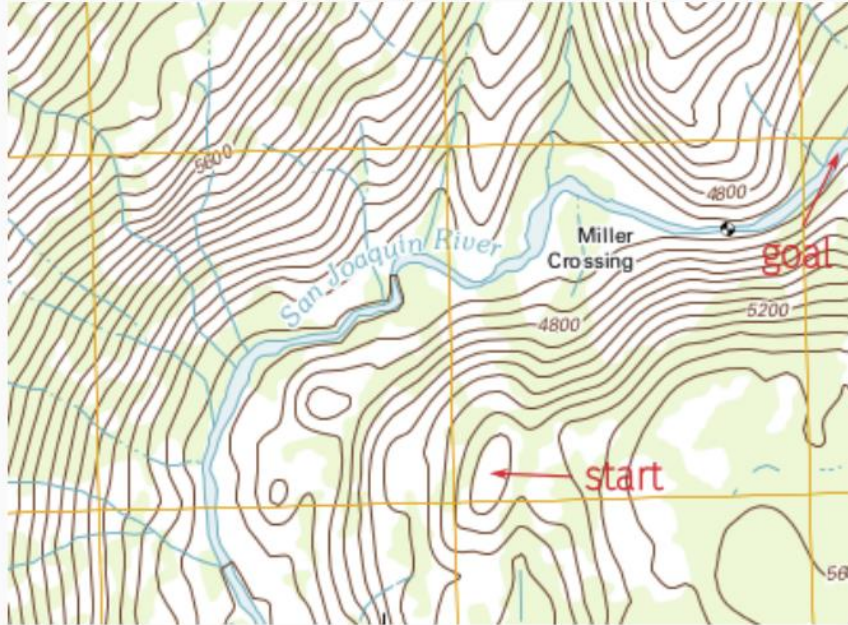
Stochastic gradient descent

Let's make it explicit that the loss function is parameterized by weights $\theta = (w, b)$.

We'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependency on θ more obvious.

We want the weights that minimize the loss (L_{CE}), averaged over all examples:
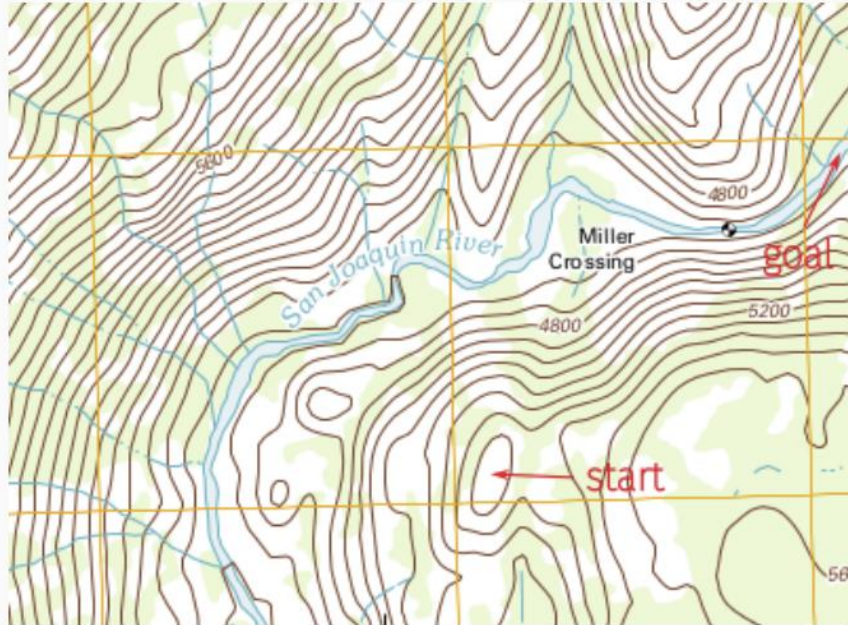
$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(f(x^{(i)}; \theta), y^{(i)})$$

Slide credit: David Mortensen

The Intuition of Gradient Descent

- You are on a hill

- It is your mission to reach the river at the bottom of the canyon (as quickly as possible)

- What is your strategy?
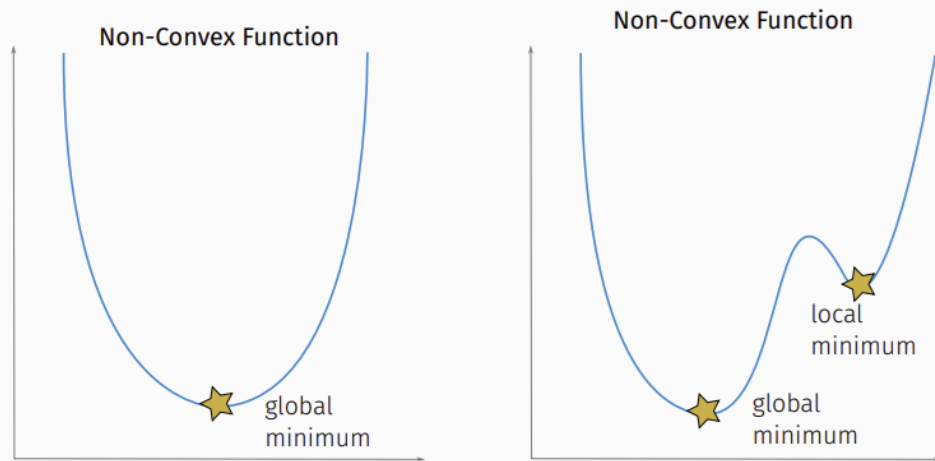
The Intuition of Gradient Descent

- You are on a hill
- It is your mission to reach the river at the bottom of the canyon (as quickly as possible)
- What is your strategy?
  1. Determine in which direction the steepest downhill slope lies
  2. Take a step in that direction
  3. Repeat until a step in any direction will take you up hill

Slide credit: David Mortensen

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

- Just one minimum
- Gradient descent is guaranteed to find the minimum, no matter where you start

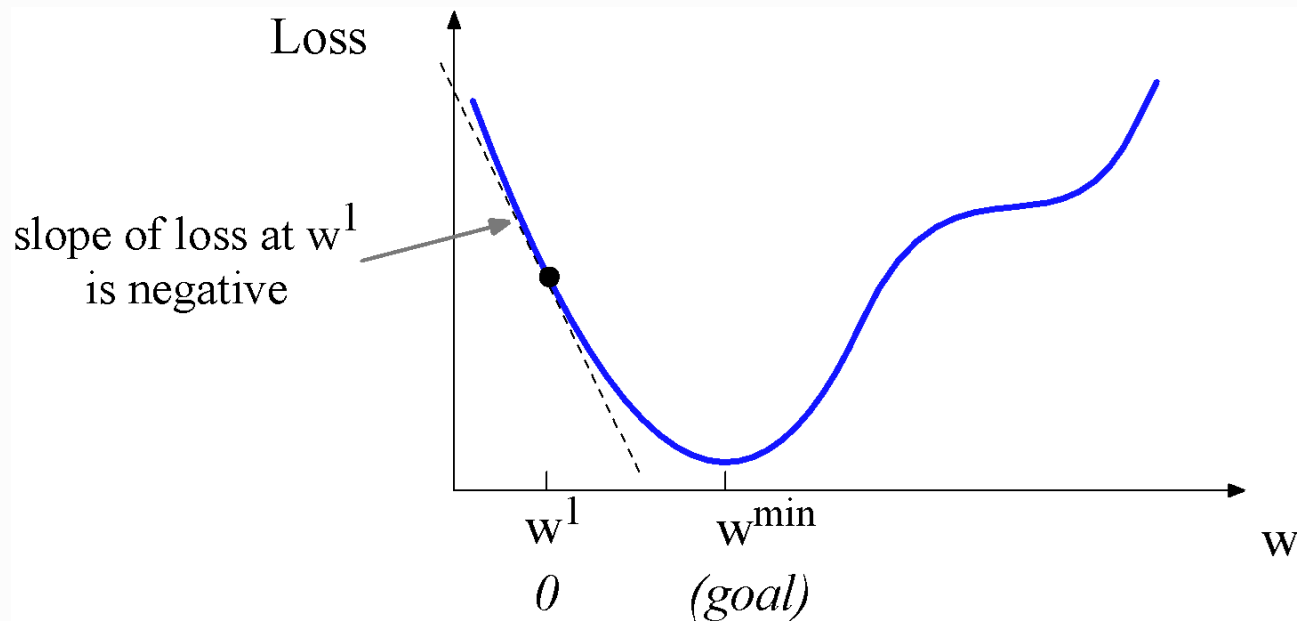Q: Given current w, should we make it bigger or smaller?
A: Move *w* in the reverse direction from the slope of the function

Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?
A: Move *w* in the reverse direction from the slope of the function

slope of loss at w^1 is negative

So we'll move positive (to the right)

Q: Given current w, should we make it bigger or smaller?
A: Move *w* in the reverse direction from the slope of the function

Loss

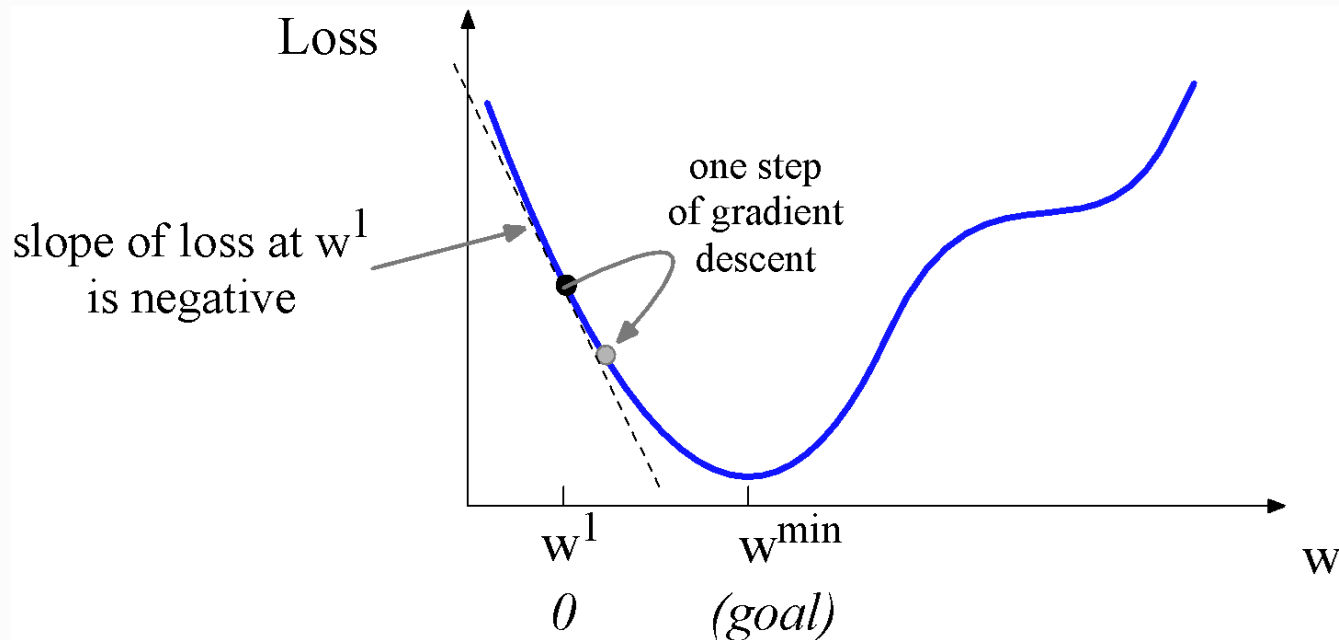slope of loss at w^1
is negative

one step
of gradient
descent

So we'll move
positive (to the right)

w^1
w^{min}

0
(goal)

w

A Gradient is a Vector Pointing in the Direction of Greatest Increase

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.

- We move by the value of the gradient (in our example, the slope)

$$\frac{d}{d\mathbf{w}} L_{CE}(f(\mathbf{x}; \mathbf{w}), y)$$

weighted by the LEARNING RATE η

- The higher the learning rate, the faster **w** changes:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{d}{dw} L_{CE}(f(\mathbf{x}; \mathbf{w}), y)$$

Slide credit: David Mortensen

We want to know where in the N-dimensional space (of the N parameters that make up θ) we should move.

The **gradient is just such a vector**; it expresses the directional components of the sharpest slope along each of the N dimensions.

Slide credit: David Mortensen

Imagine 2 dimensions, *w* and *b*

Visualizing the gradient vector at the red point

It has two dimensions shown in the x-y plane

$$\mathrm{Cost(w,b)}$$

b

w
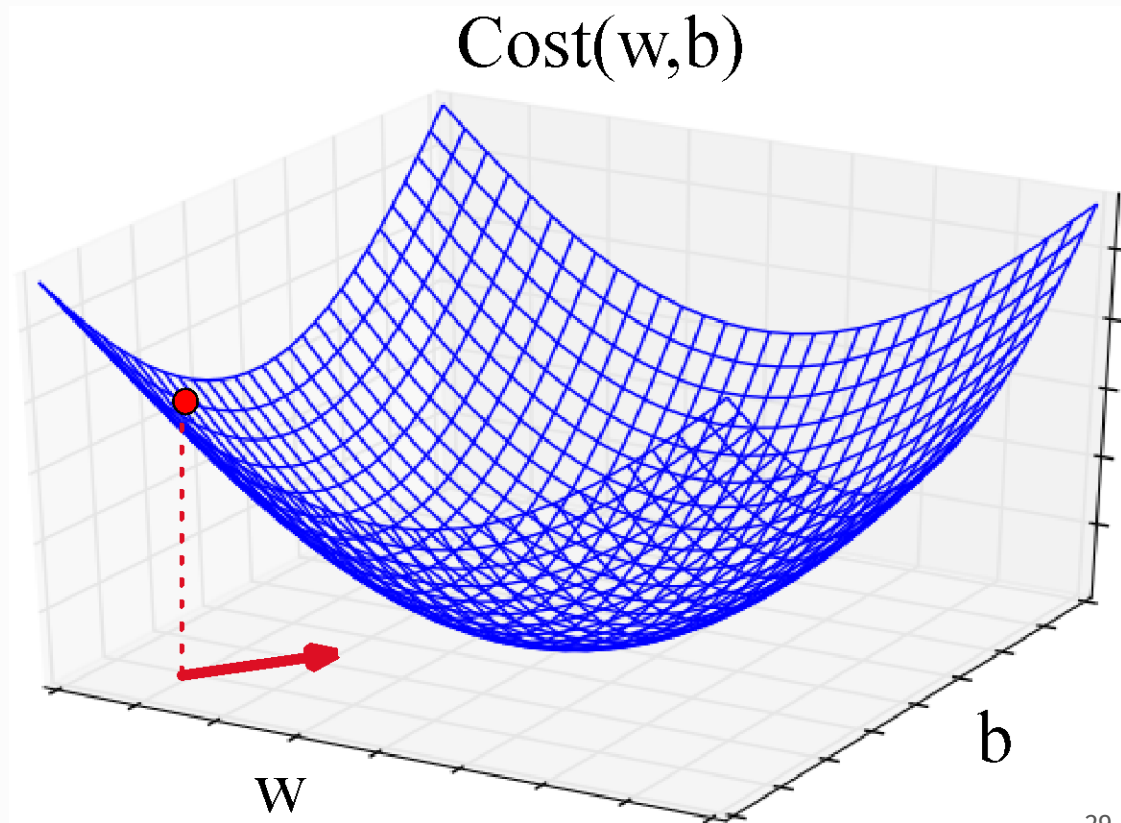
But Real Gradients Have More than Two Dimensions

- They are much longer

- They have lots of weights

- For each dimension w_i, the gradient component i tells us the slope w.r.t. that variable
  - "How much would a small change in w_i influence the total loss function L?"
  - The slope is expressed as the partial derivative ∂ of the loss ∂w_i

- We can then define the gradient as **a vector of these partials**

Let's represent $\hat{y}$ as $f(x; \theta)$ to make things clearer:

$$\nabla_\theta L(f(\mathbf{x}; \theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_0} L(f(\mathbf{x}; \theta), y) \\ \frac{\partial}{\partial w_1} L(f(\mathbf{x}; \theta), y) \\ \frac{\partial}{\partial w_2} L(f(\mathbf{x}; \theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(\mathbf{x}; \theta), y) \end{bmatrix}$$

Note that, since we are representing the bias b as w_0, θ is more-or-less equivalent to $\mathbf{w}$.

What is the final equation for updating θ based on the gradient?

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

(For us, L is the cross-entropy loss L_{CE}).

The textbook lays out the derivation in §5.10 but here's the basic idea:

Here is the cross-entropy loss function (for binary classification):

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

The derivative of this function is:

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

which is very manageable!

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ
  # where: L is the loss function
  #   f is a function parameterized by $\boldsymbol{\theta}$
  #   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
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
    1. Optional (for reporting):   # How are we doing on this tuple?
       Compute $\hat{y}^{(i)} = f(x^{(i)}; \boldsymbol{\theta})$ # 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. $g \leftarrow \nabla_{\boldsymbol{\theta}} L(f(x^{(i)}; \boldsymbol{\theta}), y^{(i)})$ # How should we move θ to maximize loss?
    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead
  return $\boldsymbol{\theta}$

A Sidenote: Hyperparameters

The learning rate (our η) is a **hyperparameter**, a term you will keep hearing

- **Set it too high?** The learner will catapult itself across the minimum and may not converge
- **Set it too low?** The learner will take a long time to get to the minimum, and may not converge in our lifetime

But what are hyperparameters again?

- Hyperparameters are parameters in a machine learning model that are not learned empirically
- They have to be set by the human who is designing the algorithm

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

η = 0.1

Example of gradient descent

Update step for update θ is:

$$\theta_{t+1} = \theta_t - \eta \frac{d}{d\theta} L(f(x;\theta), y)$$

$$w_1 = w_2 = b = 0;$$
$$x_1 = 3; \quad x_2 = 2;$$
$$y = 1$$

where

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

Gradient vector has 3 dimensions:

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

Example of gradient descent

Update step for update θ is:

$$\theta_{t+1} = \theta_t - \eta \frac{d}{d\theta} L(f(x;\theta),\, y)$$

$$w_1 = w_2 = b = 0;$$
$$x_1 = 3;\quad x_2 = 2;$$
$$y = 1$$

where

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

Gradient vector has 3 dimensions:

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

Slide adapted from Jurafksy & Martin

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:

$$\theta_{t+1} = \theta_t - \eta \frac{d}{d\theta} L(f(x; \theta), y)$$

$\eta = 0.1;$

$$\theta^1 =$$

Example of gradient descent

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:

$$\theta_{t+1} = \theta_t - \eta \frac{d}{d\theta} L(f(x;\theta),\, y)$$

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

$\eta = 0.1;$

Batch and mini-batch training

Mini-batching

- In stochastic gradient descent, the algorithm chooses one random example at each iteration
- The result? Sometimes movements are choppy and abrupt
- In practice, instead, we usually compute the gradient over **batches** of training instances
- Entire dataset: BATCH TRAINING
- *m* examples (e.g., 512 or 1024): MINI-BATCH TRAINING

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.
- Fails to generalize to a test set without this word

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

Regularization Is One Solution to Overfitting

Add a regularization term $R(\theta)$ to the loss function (which we will write, for now, as maximizing log probability rather than minimizing loss):

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

The insight: we choose, for $R(\theta)$, a function that penalizes large weights because fitting the data well with **big weights** is not as good as fitting the data a bit less well with **small weights**.

We can define $R(\theta)$ as the (square of the) **L2 norm**, that is, the Euclidean distance from θ to the origin.

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

If we L2 regularize the objective function, we get:

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

The larger the weights are, the farther the vector will be from the origin, and thus the more will be deducted from the log probability.

48

Slide credit: David Mortensen

L1 Regularization (or the lasso regression) is named after the **L1 norm** $||W||_1$:

- The sum of the absolute value of the weights

- The MANHATTAN DISTANCE

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

When added (or rather, subtracted) from an objective function, it looks like this:
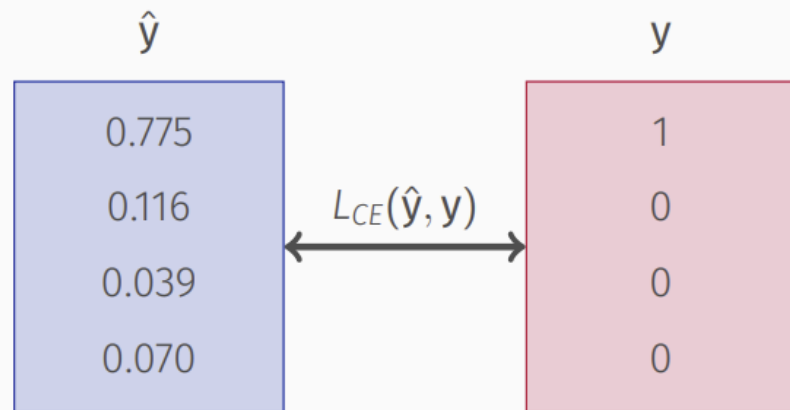
$$\hat{\theta} = \underset{\theta}{\mathrm{argmax}} \left[\sum_{i=1}^{m} \log P(y^{(i)} \mid x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |\theta_j|$$

While the function is different, the insight is similar to that for the L2 norm.

49

Slide credit: David Mortensen

Training multinomial logistic regression

Compare **y**, a ONE-HOT VECTOR (one one, all other elements zero) and

$\hat{y}$ y

| 0.775 |
0.116	$L_{CE}(\hat{y}, y)$	1
0.039		0
0.070		0
		0

How "distant" is $\hat{y}$ from **y**? One measure is categorical cross-entropy loss:

$$
\begin{aligned}
L_{CE} \;=\; & -\sum_{i=1} T_i \log S_i \\
=\; & -[1 \log_2 0.775 + 0 \log_2 0.126 + \\
& \; 0 \log_2 0.039 + 0 \log_2 0.070] \\
=\; & -\log_2 0.775 \\
=\; & 0.3677
\end{aligned}
$$

The elements of $\hat{y}$ that correspond to 0-elements in **y** are effectively ignored.

Reminder: the loss function for binary logistic regression (LR with two classes) is

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

Note that we have two terms—one for when $y = 1$ and one for when $y = 0$—corresponding to the two classes. What if we have K classes?

$$
\begin{aligned}
L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) \;&=\; -\sum_{k=1}^{K} \mathbf{y}_k \log \hat{\mathbf{y}}_k \\
&=\; -\log \hat{\mathbf{y}}_c \quad \text{(where } c \text{ is the correct class)} \\
&=\; -\log \hat{p}(\mathbf{y}_c = 1 \mid \mathbf{x}) \quad \text{(where } c \text{ is the correct class)} \\
&=\; -\log \frac{\exp(\mathbf{w_c} \cdot \mathbf{x} + b_c)}{\sum_{j=1}^{K} \exp(\mathbf{w_j} \cdot \mathbf{x} + b_j)} \quad (c \text{ is the correct class)}
\end{aligned}
$$

Slide credit: David Mortensen

Consider one piece of the gradient—the derivative with respect to one weight.

- For each class k the weight of the ith element of $\mathbf{x}$ (the input features) is $\mathbf{w}_{k,i}$.

- What is the partial derivative of $L_{CE}(\hat{\mathbf{y}}, \mathbf{y})$ wrt $\mathbf{w}_{k,i}$?

- It turns out, after some math, that the difference between the true value for the class k (either 1 or 0) and the probability that the class outputs class k (weighted by the value of the input $\mathbf{x}_i$ corresponding to the ith element of the weight vector for class k.

$$\frac{\partial L_{CE}}{\partial \mathbf{w}_{k,i}} = -(\mathbf{y}_k - \hat{\mathbf{y}}_k)\mathbf{x}_i$$

- The rest of the procedure for training multinomial LR is the same as for binary LR.

Slide credit: David Mortensen

Questions?

Homework 1 due next Thu Sep 19