Machine Learning

Lecture 9: Logistic Regression

Logistic Regression

- Important analytic tool in natural and social sciences
- Baseline supervised machine learning tool for classification
- Is also the foundation of neural networks

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

imagenet

imagenet

Generative Classifier:

- Build a model of what's in a cat image
  - Knows about whiskers, ears, eyes
  - Assigns a probability to any image:
    - how cat-y is this image?

Also build a model for dog images

Now 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

Finding the correct class *c* given input *d* in Generative vs Discriminative Classifiers

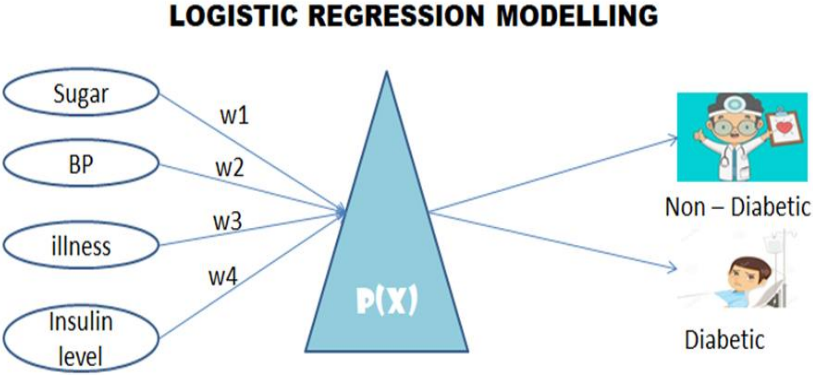
- Naive Bayes

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

- Logistic Regression

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

Classification using Logistic Regression



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 $[x_1, x_2, \ldots, x_n]$. 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

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 $\hat{y}^{(i)} \in \{0,1\}$

Logistic Regression for one observation $\boldsymbol{x}$

- 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 $\hat{y} \in \{0,1\}$

How to do classification

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

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

- 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:

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

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

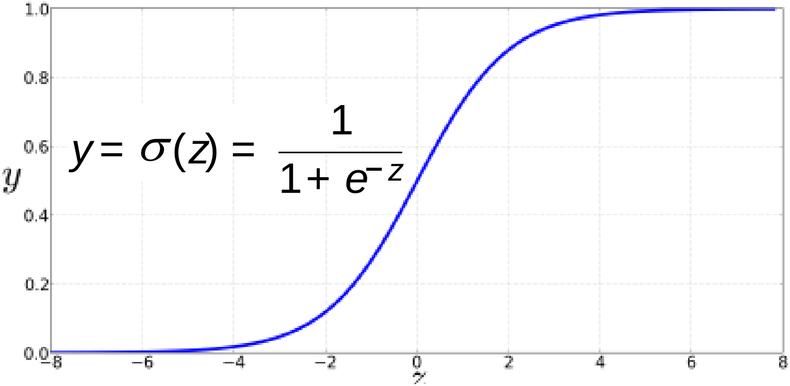
The problem: *z* isn't a probability, it's just a number!

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

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

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

The very useful *sigmoid* or *logistic* function



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

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

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

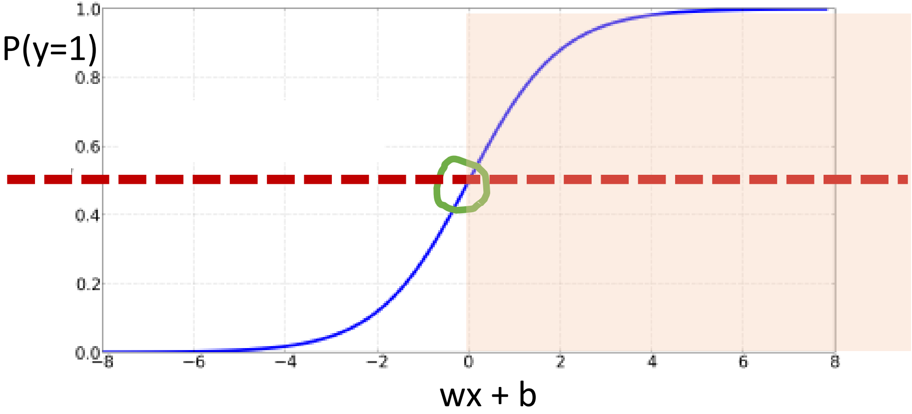
Turning a probability into a classifier

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

0.5 here is called the **decision boundary**

The probabilistic classifier

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



Turning a probability into a classifier

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

if w·x+b > 0

if w·x+b ≤ 0

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:

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

- 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

Bernoulli Distribution
(Coin Distribution)

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

- noting:

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

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

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

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

Deriving cross-entropy loss for a single observation *x*

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

Maximize:

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

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

Minimize:

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

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

Our goal: minimize the loss

- Let's make explicit that the loss function is parameterized by weights

$$\theta=(w,b)$$

- 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:

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

Gradient Descent

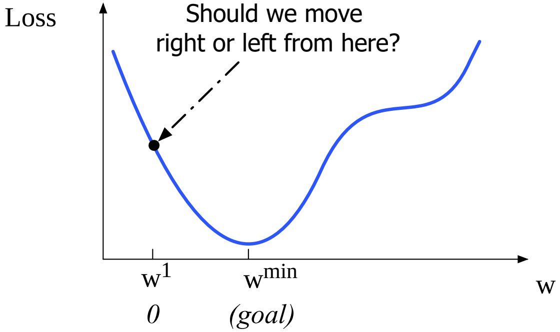
Our goal: minimize the loss

- 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)

Let's first visualize for a single scalar w

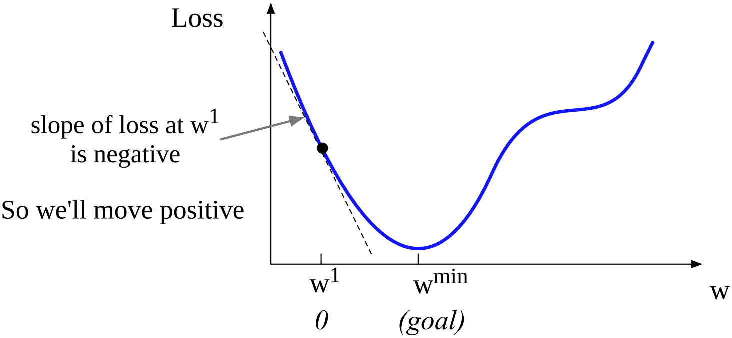
Q: Given current w, should we make it bigger or smaller?

A: Move w in the reverse direction from the slope of the function



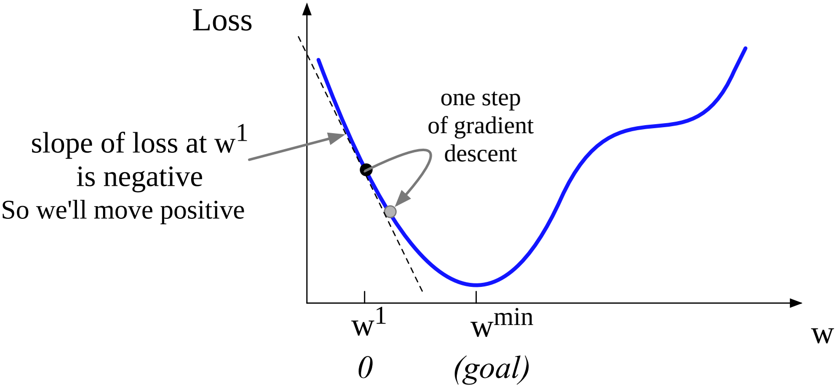
Let's first visualize for a single scalar ***w***

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



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



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

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

Now let's consider N dimensions

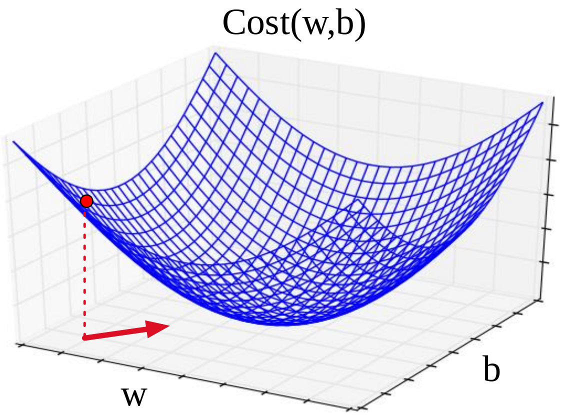
- We want to know where in the N-dimensional space (of the *N* parameters that make up θ) we should move.

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

Imagine 2 dimensions, w and b

- Visualizing the gradient vector at the red point

- It has two dimensions shown in the x-y plane



Real gradients

- Are much longer; lots and lots of weights

- 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 then defined as a vector of these partials.

The Gradient

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

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

The final equation for updating θ based on the gradient is thus

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

What are these partial derivatives for logistic regression?

The loss function

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

The elegant derivative of this function

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

SGD Algorithm

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

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

$\theta \leftarrow 0$

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

The algorithm can terminate when it converges (or when the gradient norm < ε), or when progress halts

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
- An example, where the true y=1 (positive)
- Two features:

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

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

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

- Update step for update θ is:

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

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:

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

Example of gradient descent

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

- Update step for update θ is:

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

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

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

- Update step for update θ is:

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

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:

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

Example of gradient descent

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

- Update step for update θ is:

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

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:

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

Example of gradient descent

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

- Update step for update θ is:

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

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:

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

Example of gradient descent

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

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:

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$$ $\eta = 0.1$;

$$\theta^1 =$$

Example of gradient descent

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

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:

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

$\eta = 0.1$;

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

Example of gradient descent

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

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:

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

$\eta = 0.1$;

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

Mini-batch training

- Stochastic gradient descent chooses a single random example at a time.
- 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)