ECE594: Mathematical Models of Language

Spring 2022

Lecture 3: Word-level Models–Logistic Regression

Logistics

- Need feedback providers for Thursday

Text Classification

- Generative model–naïve Bayes classifier
  - Naïve Bayes independence assumption–to learn the joint distribution, modeling the probability of the text x
  - In classification problems, always given x, and need to predict y. Instead of modeling the probability of the text x, a difficult task, what if we focus directly on the problem of predicting y?
  - **Discriminative learning** algorithms have this focus

Text Classification: Definition

- *Input*:
  - a document d
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

- *Output*: a predicted class $c \in C$

Supervised Machine Learning

- *Input:*
  - a document d
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
  - A training set of m hand-labeled documents $(d_1,c_1),\ldots,(d_m,c_m)$, *i.i.d*
- *Output:*
  - a learned classifier $\gamma: d \rightarrow c$

Linear Classification

- Naïve Bayes (Generative classifier)
- Logistic regression (Discriminative classifier)
- Classification decision based on weighted sum of individual features

Discriminative Classifier

Just try to distinguish dogs from cats

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

Discriminative Classifiers

- Logistic Regression

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

Logistic Regression

- Vector of features $f(x) = [f_1(x), f_2(x), \ldots, f_n(x)]$
- $f_1$count(positive lexicon words)

$$f_2 = \begin{cases} 1 \text{ if “no”} \in \text{doc} \\ 0 \text{ otherwise} \end{cases}$$

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

  - *Feature importance*

- $Z = W \cdot f(x) + b$ is a scoring function for the compatibility of the base features f and the label y.
- b – bias term to account for the error that is introduced by approximating actual y using a simple model
- Output: a predicted class $\hat{y} \in \{0,1\}$

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 $[f_1(x), f_2(x), \ldots, f_n(x)]$.

2. A **classification function** computes $\hat{y}$, the estimated class, via $p(y|x)$
   - **sigmoid** or **softmax** functions

3. An **objective function** for learning that is optimized using the training data
   - **cross-entropy loss**

4. An algorithm for **optimizing the objective function**
   - stochastic gradient descent

The two phases of logistic regression

- **Training**: Learning W and *b* using to minimize **cross-entropy loss using stochastic gradient descent**.

- **Test**: Given a test example *x* we compute $p(y=1|x)$ and $p(y=0|x)$ using learned weights *w* and *b*, and return the label with higher probability

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 $[f_1(x), f_2(x), \dots, f_n(x)]$.

2. A **classification function** computes $\hat{y}$, the estimated class, via $p(y|x)$
   - **sigmoid** or **softmax** functions

3. An **objective function** for learning that is optimized using the training data
   - **cross-entropy loss**

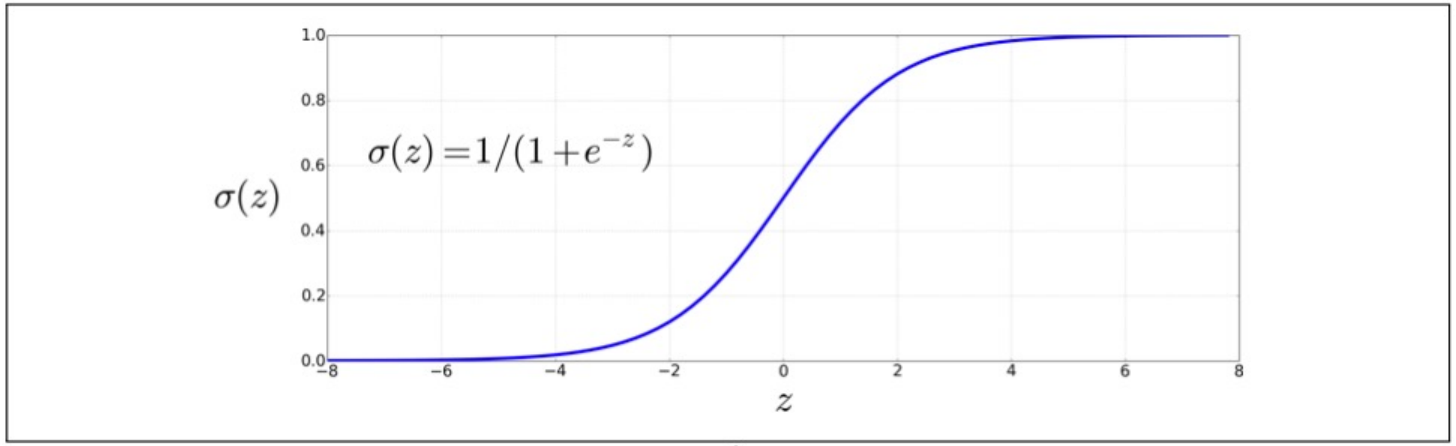
4. An algorithm for **optimizing the objective function**
   - stochastic gradient descent

Sigmoid function

1. $Z = W \cdot f(x) + b$ is a scoring function, z in $(-\infty, +\infty)$

2. Need a $p(y|x)$ a probability

3. Use the sigmoid
(aka the logistic) function

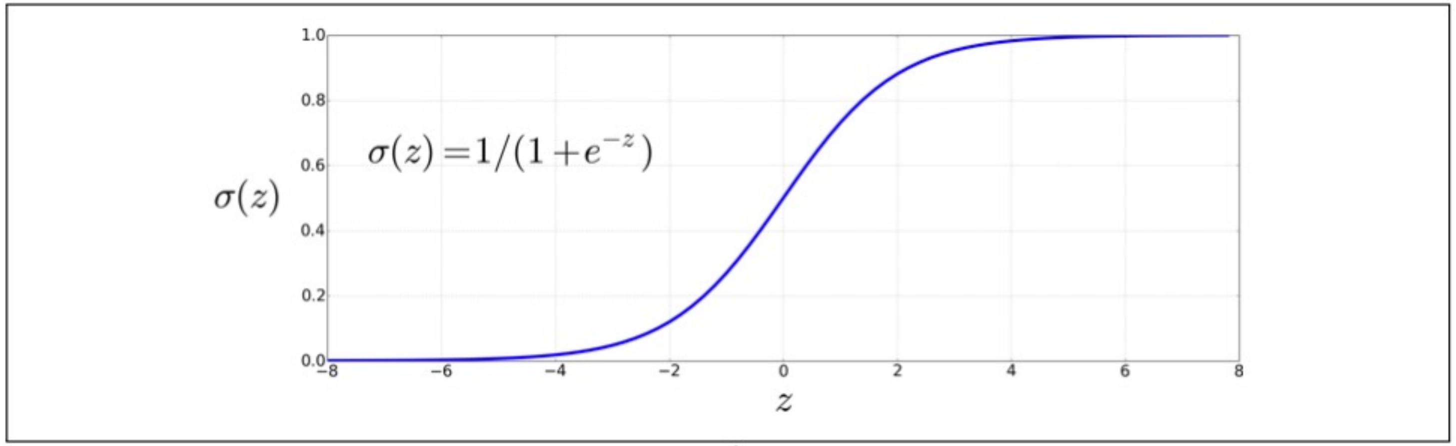


Sigmoid function

1. The property that

$1-\sigma(x) = \sigma(-x)$

permits us to write



- $P(y = 1|\mathrm{x}) = \sigma(\mathrm{z}) = \dfrac{1}{1+\exp(-\mathrm{z})}$

- $P(y = 0|\mathrm{x}) = \sigma(-\mathrm{z}) = \dfrac{\exp(-\mathrm{z})}{1+\exp(-\mathrm{z})}$

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

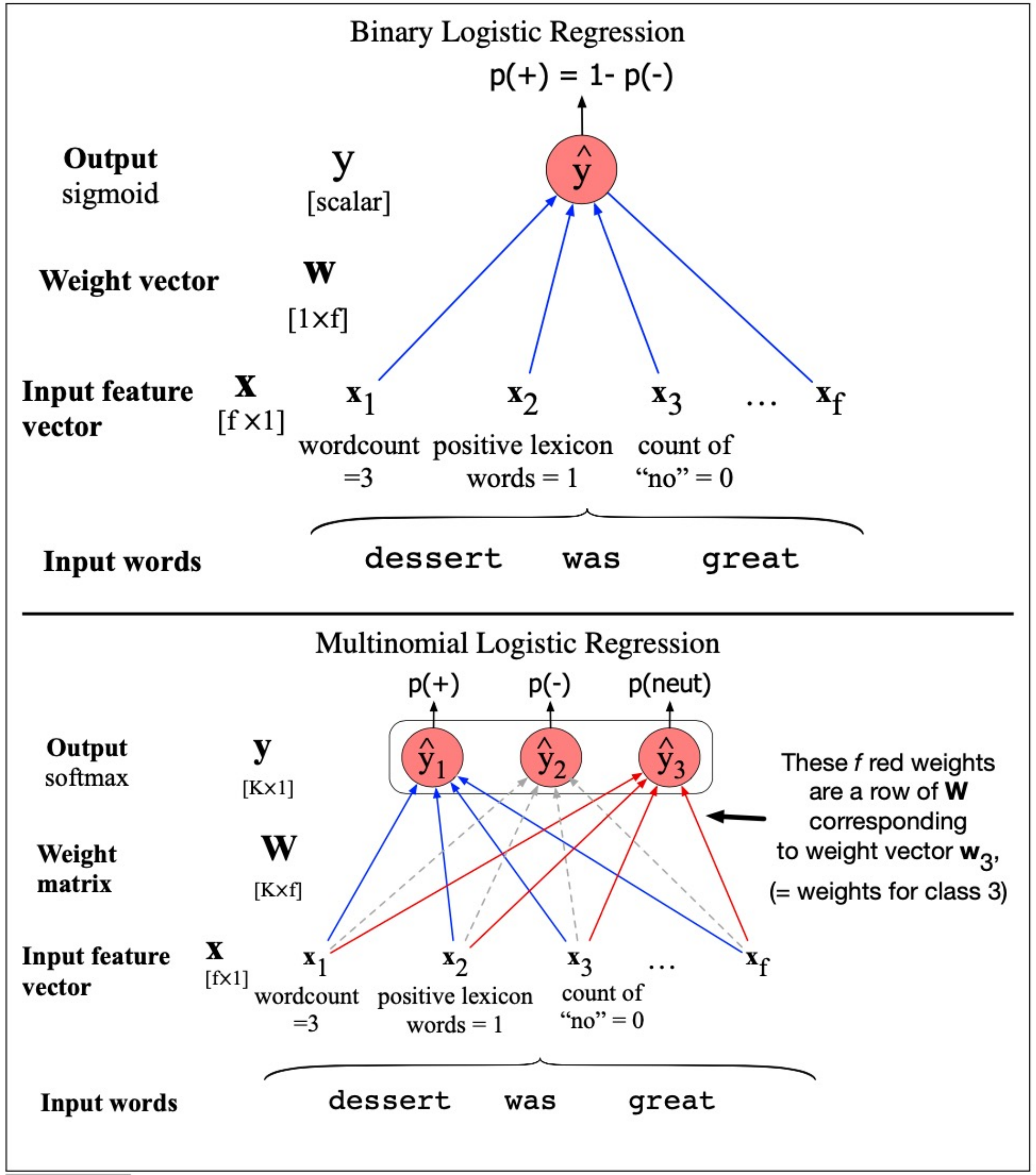
if w·x+b ≤ 0

From two to multiple classes

Multinomial (softmax) regression,
maxent classification)

- Classes 1, 2…K
- Separate weight vectors w_k and bias b_k for each of the K classes
- Given an input vector $z = [z_1, z_2, \ldots, z_K]$, softmax function maps z to a probability distribution

$$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(\mathbf{z}_1)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}, \frac{\exp(\mathbf{z}_2)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}, \ldots, \frac{\exp(\mathbf{z}_K)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}\right]$$



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 $[f_1(x), f_2(x), \ldots, f_n(x)]$.

2. A **classification function** computes $\hat{y}$, the estimated class, via $p(y|x)$
   - **sigmoid** or **softmax** functions

3. An **objective function** for learning that is optimized using the training data
   - **cross-entropy loss**

4. An algorithm for **optimizing the objective function**
   - stochastic gradient descent

Learning: Cross-Entropy Loss

- 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*

Cross-entropy loss for a single observation x

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

We 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- $\hat{y}$

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

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

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:

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

- 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 $\hat{y}$:

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

Stochastic Gradient Descent

- The loss function is parameterized by weights θ=(w,b)
- We represent $\hat{y}$ as $f\,(x;\,\theta)$ to make the dependence on θ explicit

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

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

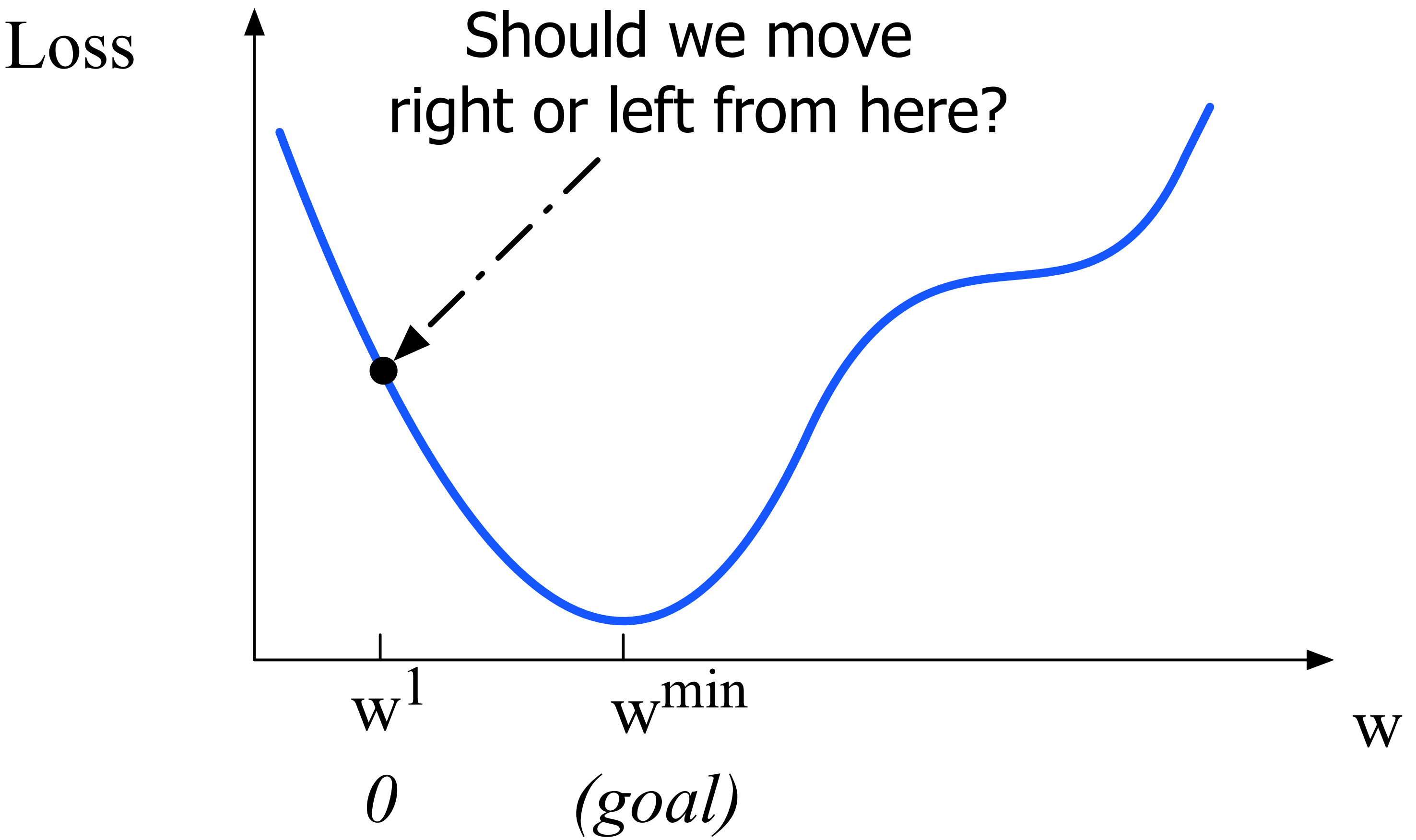
Our goal: minimize the loss

- For logistic regression, loss function is **convex (**can you prove this?**)**
- In particular, if an objective function is differentiable, then gradient-based optimization can be employed; if it is also convex, then gradient-based optimization is guaranteed to find the globally optimal solution.
- Gradient descent starting from any point is guaranteed to find the minimum

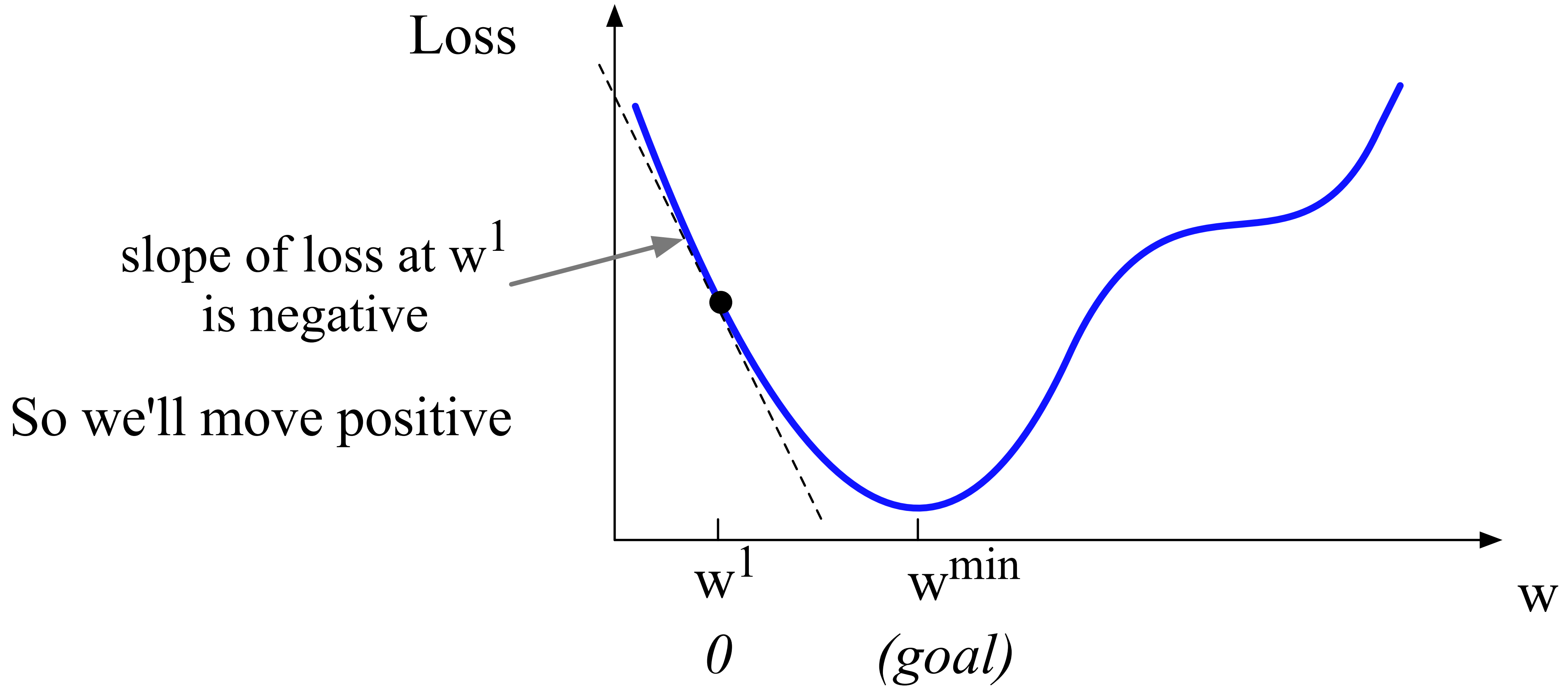
Let's first visualize for a single scalar w

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

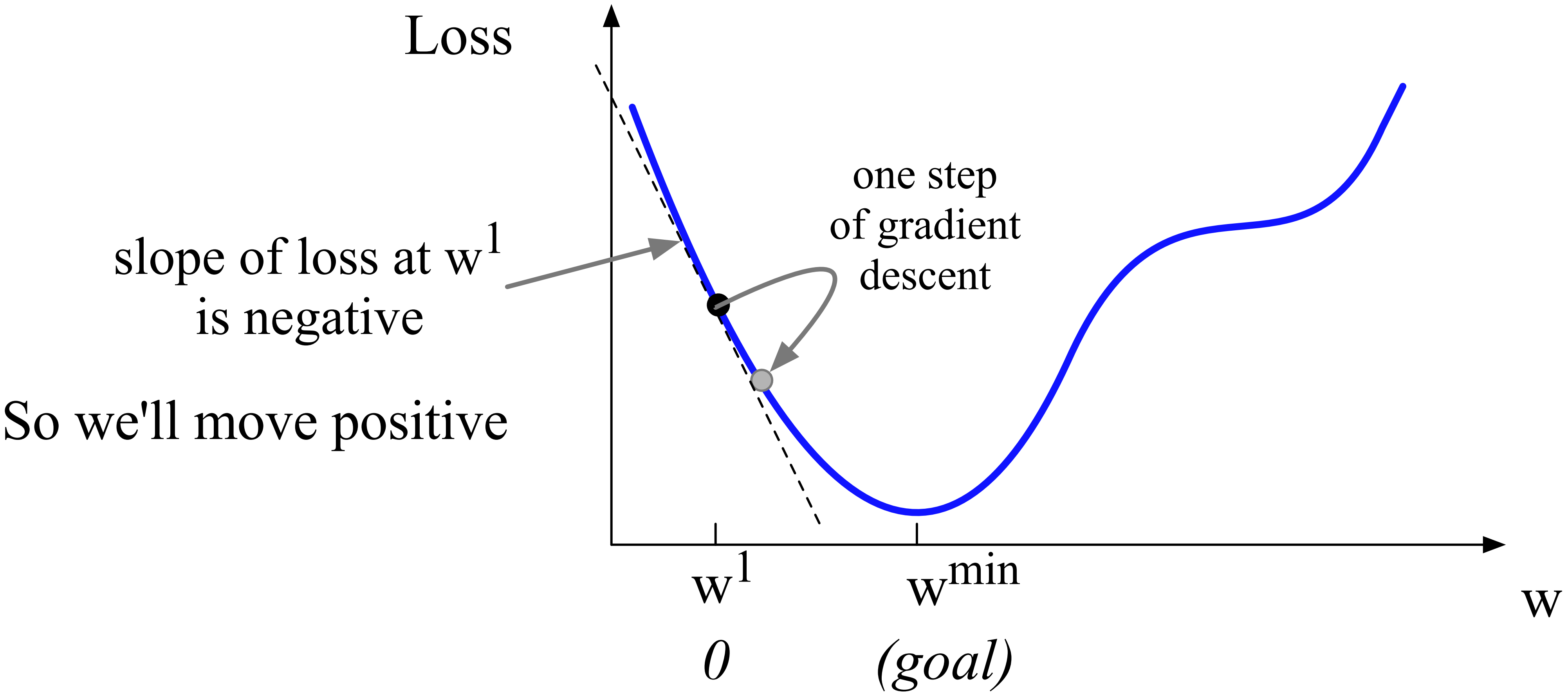
A: Move *w* in the reverse direction from the slope of the function



Let's first visualize for a single scalar w



Let's first visualize for a single scalar w



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

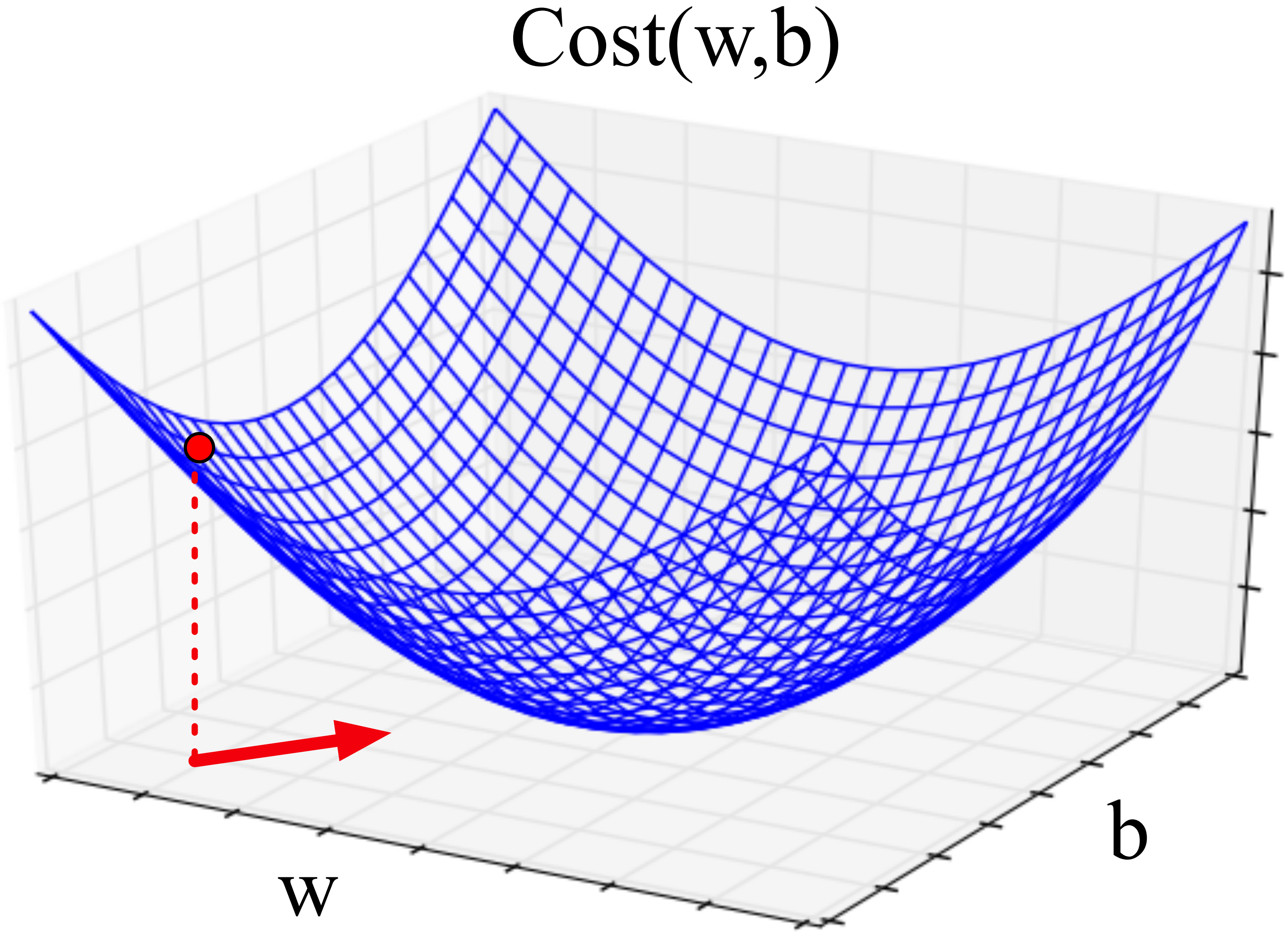
$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w),y)$$

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



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.

Regularization to Prevent Overfitting

- A model that perfectly match the training data has a problem
- It will **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 **generalize**

Overfitting

+

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

–

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

Useful or harmless features

X1 = "this"
X2 = "movie
X3 = "hated"

X4 = "drew me in"

4gram features that just "memorize" training set and might cause problems

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

Overfitting

- 4-gram model on tiny data will just memorize the data
    - 100% accuracy on the training set
- But it will be surprised by the novel 4-grams in the test data
- Models that are too powerful can **overfit** the data
    - Fitting the details of the training data so exactly that the model doesn't generalize well to the test set
        - How to avoid overfitting?
            - Regularization in logistic regression

Regularization

- A solution for overfitting
- Add a regularization term *R*(θ) to the loss function

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

- Idea: choose an *R*(θ) that penalizes large weights
  - 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)

- The sum of the squares of the weights
- The name is because this is the (square of the) **L2 norm** $||\theta||_2$, = **Euclidean distance** of θ to the origin.

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

- L2 regularized objective function:

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

L1 Regularization (= Lasso regression)

- The sum of the (absolute value of the) weights, **L1 norm or Manhattan distance**

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

- L1 regularized objective function:

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

Optimization

- **Gradient descent**–batch optimization– each update to the weights is based on a computation involving the entire dataset
  - Inefficient–at early stages of training, a small number of training examples could point the learner in the correct direction

- **Stochastic gradient descent**, the approximate gradient is computed by randomly sampling a single instance, and an update is made immediately

- In **mini- batch** stochastic gradient descent, the gradient is computed over a small set of instances

Naïve Bayes vs Logistic Regression

- Twitter sentiment classification

	Naïve Bayes	Logistic Regression
Accuracy (%)	67	77
Precision (%)	69	74

Word-Level Models

- Words as units of text
- Syntactic properties, semantic properties
- How do we derive meaning?
- Language described from 3 perspectives
    - Relations between words
    - Compositionality of how words are formed
    - Distributional properties of co-occurrence

Relation Between Words

- Do they have the same conjugation? (morphology)
- Are they the same part of speech? (syntactic)
- Are they related in meaning? (semantic)

WordNet

- Manually created ontology
  - Word relations–synonymy, hypernymy
  - Notion of word similarity
  - Task-dependent ontology construction

- Multilingual

Compositionality

- Creating meaning from constituent parts
    - Putting together words (compounding)
    - Adding suffixes
    - Putting together words to form phrases and sentences

Word sense ambiguity

- Iraqi head seeks arms
- Drunk gets nine years in violin case

Why disambiguate word sense?

information retrieval

– query: *bat care*

machine translation

– *bat*: *murciélago* (animal) or *bate* (for baseball)

text-to-speech

– *bass* (stringed instrument) vs. *bass* (fish)

Distributional Property

- Creating meaning from context
  - Prominent paradigm for computational models of meaning

Distributional Semantics

- What is *tesgüino*?

(a) A bottle of tesgüino is on the table

(b) People like tesgüino.

(c) Don't have tesgüino before you drive.

(d) Tesgüino is made out of corn

Distributional Hypothesis

(C1) *A bottle of ________ is on the table*

(C2) *People like ________.*

(C3) *Don't have ________ before you drive.*

(C4) ________ *is made out of corn*

	C1	C2	C3	C4
tesgüino	**1**	**1**	**1**	**1**
loud	0	0	0	0
Motor oil	1	0	0	1
tortillas	0	1	0	1
choices	0	1	0	0
wine	1	1	1	1

Distributional Hypothesis

Distributional hypothesis, stated by linguist John R. Firth (1957) as:

“You shall know a word by the company it keeps.”

≈ “words that occur in similar contexts tend to have similar meanings”

One of the most successful ideas of modern statistical NLP

TESGÜINO, UNA BEBIDA RITUAL DE MAÍZ DE LOS RARÁMURIS

¿QUÉ COMER?

COCINA MEXICANA

28 JUL 2015

1

Next Lecture: Distributed Representation of Words