Redes Neuronales para Lenguaje Natural

2023

Grupo de Procesamiento de Lenguaje Natural
Instituto de Computación

Redes Neuronales Completamente Conectadas

- **Regresión Logística**
- Neurona Artificial, Redes Multicapa, Backpropagation
- Aplicaciones, Clasificación, Modelo de lenguaje
- Implementación

Jurafsky and Martin 3rd edition. Cap 5 y 7. https://web.stanford.edu/~jurafsky/slp3/

(estas slides tienen partes de las de Jurafsky y Martin.)

PyTorch https://pytorch.org

Redes Neuronales Completamente Conectadas (Perceptrón Multicapa)

Regresión Logística

¿Por qué empezamos con regresión logística?

- Introduce conceptos fundamentales que se usan en las redes neuronales
- Define un marco general de clasificación y entrenamiento que es el mismo que vamos a usar con redes neuronales
- Va a facilitar el entendimiento de lo que sigue (o eso esperamos)

Clasificación

- Entrada:
  - Vector de entrada **x**
  - Conjunto de clases $\mathbf{C = \{c_1, \ldots, c_k\}}$

- Salida:
  - Clase predicha $\mathbf{\hat{y} \in C}$

Clasificación

- Entrada:
  - Vector de entrada **x** (Ej. Un comentario)
  - Conjunto de clases $\mathbf{C = \{c_1, \ldots, c_k\}}$ (Ej. [Pos, Neg, None])
- Salida:
  - Clase predicha $\mathbf{\hat{y} \in C}$ (Ej. Polaridad del comentario)

Clasificación

- Entrada:
  - Vector de entrada **x** (Ej. Correo electrónico)
  - Conjunto de clases $\mathbf{C = \{c_1, \ldots, c_k\}}$ (Ej. [Alta_prio, Baja_prio, Spam])
- Salida:
  - Clase predicha $\mathbf{\hat{y} \in C}$ (Ej. Clasificación del correo)

Clasificación

- Entrada:
  - Vector de entrada **x** (Ej. Texto)
  - Conjunto de clases $\mathbf{C = \{c_1, \ldots, c_k\}}$ (Ej. Palabras)
- Salida:
  - Clase predicha $\mathbf{\hat{y} \in C}$ (Ej. Próxima palabra)

Clasificación Binaria

- Entrada:
  - Vector de entrada **x**
- Salida:
  - Clase predicha $\hat{\mathbf{y}} \in \{0,1\}$

Clasificación Binaria

- Entrada:
  - Vector de entrada **x**
- Salida:
  - Clase predicha $\hat{y} \in \{0,1\}$

Ejemplos

- Análisis de sentimiento
- Detección de spam
- Detección de discurso de odio
- Detección de humor
- Detección de relaciones
- Clasificación multiclase como k clasificaciones binarias

Regresión Logística

- **Vector de entrada: $x = \langle x_1, \ldots, x_d \rangle$**
  - "F*eatures* definidas a mano" de la entrada
    - Ej. X_1= count('no'),X_2= ocurre('pero'), X_2= "cant. Ocurr. Lexico positivo", ...
    - Ingeniería de *features*
  - Bag of Words (conteos de palabras),
  - Word Embeddings (más adelante)

- **Conjunto de entrenamiento: $D = \{(x^{(i)}, y^{(i)}) : i = 1 \ldots N\}$**
  - **$x^{(i)} \in R^d$, $y^{(i)} \in \{0,1\}$**

- **Pesos: w_i y b**
  - Se ajustan durante el entrenamiento (aprendizaje)

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

Regresión Logística

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

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

Regresión Logística

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

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

Problema:

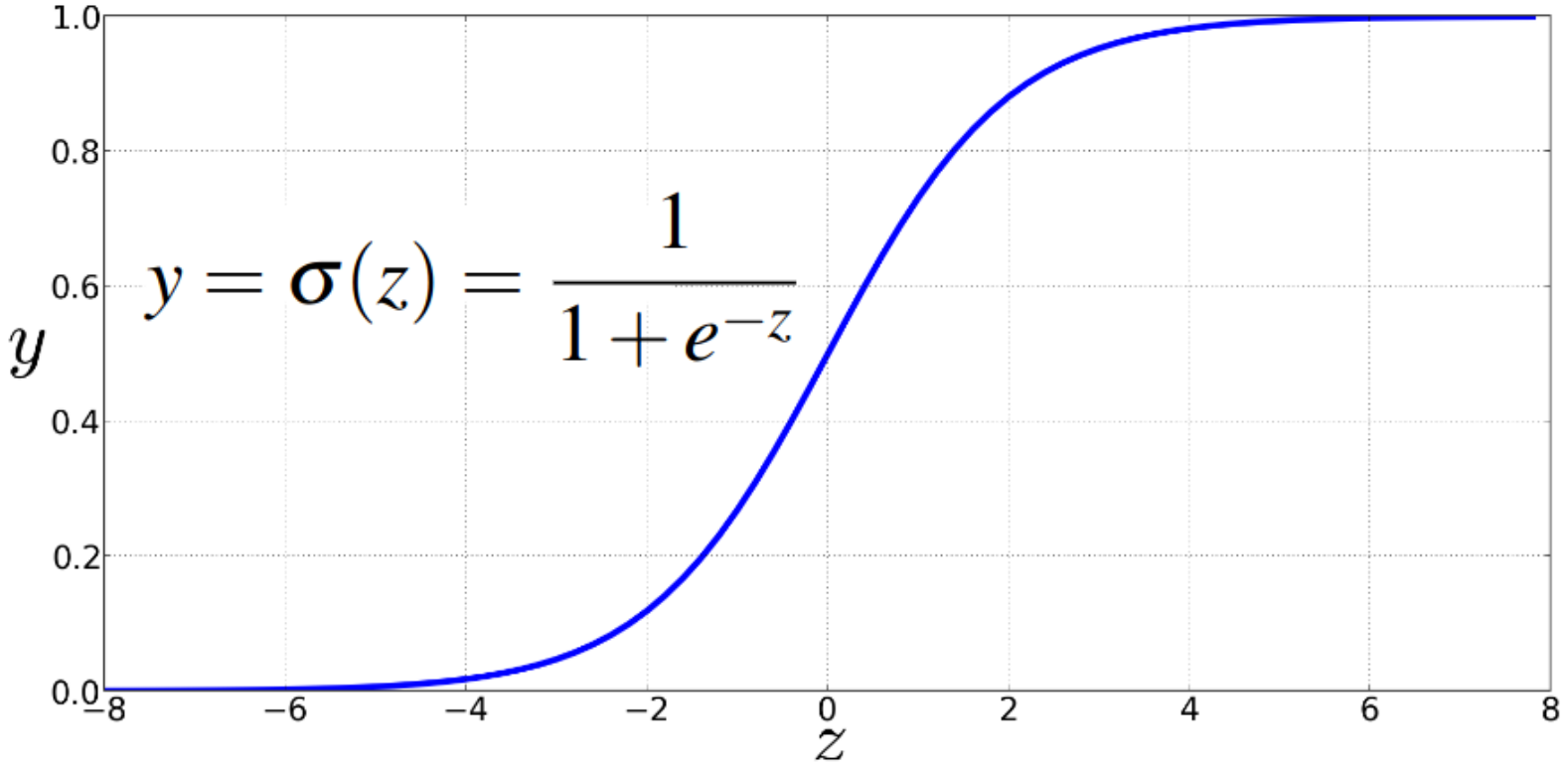
La salida es un valor real,

¿cómo decidimos entre 0 o 1?

Solución:

Que la salida sea una probabilidad

Regresión Logística (función logistica o sigmoide)



Regresión Logística

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

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

- “Vectorizamos” la entrada y se calcula z

- Con la función sigmoid se vuelve la salida a una probabilidad

Regresión Logística

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

Regresión Logística (Interpretación geométrica)

- Se separan las entradas con un hiperplano
  - Las que están de un lado ($w \cdot x + b > 0$) se clasifican como positivas
  - Las que están del otro ($w \cdot x + b \leq 0$) como negativas

- ¿Cómo “encontramos” **w** y **b**?

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`'": $w_k = -2$

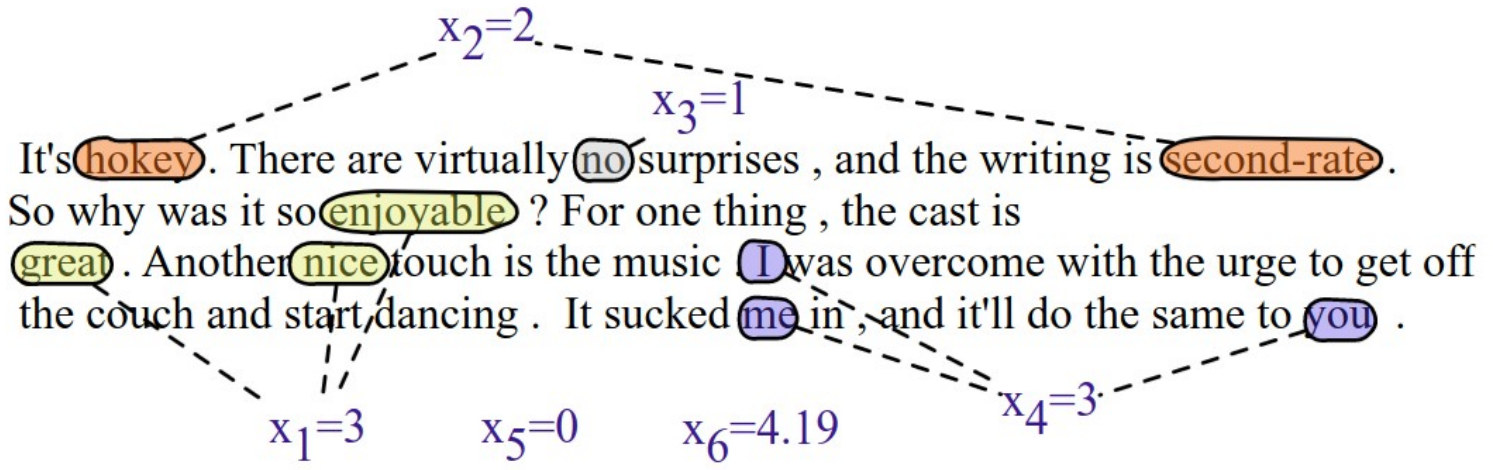
Features in logistic regression

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`'": $w_k = -2$

Esto va en dirección de la interpretabilidad del modelo.
Por ejemplo, responder porque el modelo devuelve la salida que devuelve.
Vamos a ver que no es tan directo con las redes neuronales.

Ejemplo



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	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Var	Definition	Val	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)	$\ln(66) = 4.19$	

$$\text{Suppose w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$$

$$\text{b} = 0.1$$

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

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

¿Cómo encontramos w y b?

Clasificación supervisada

- Tenemos:
  - Clase esperada de un conjunto de entradas (conjunto de entrenamiento)
    - $\mathbf{y^{(i)} \in \{0,1\}}$
  - Salida computada por el sistema para cada entrada
    - $\mathbf{\hat{y}^{(i)} \in [0,1]}$

- Queremos **w** y **b** que minimice la **distancia** entre la salida esperada y la obtenida
  - Función de pérdida (o costo) (Ej. **cross-entropy loss**)
  - Algoritmo de optimización para ajustar **w** y **b** (Ej. **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

Cross-entropy

$$H(p, q) = -\operatorname{E}_p[\log q]$$

$$H(P, Q) = -\sum_{x \in \mathcal{X}} p(x)\ \log q(x)$$

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

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

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

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

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

Our goal: minimize the loss

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

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

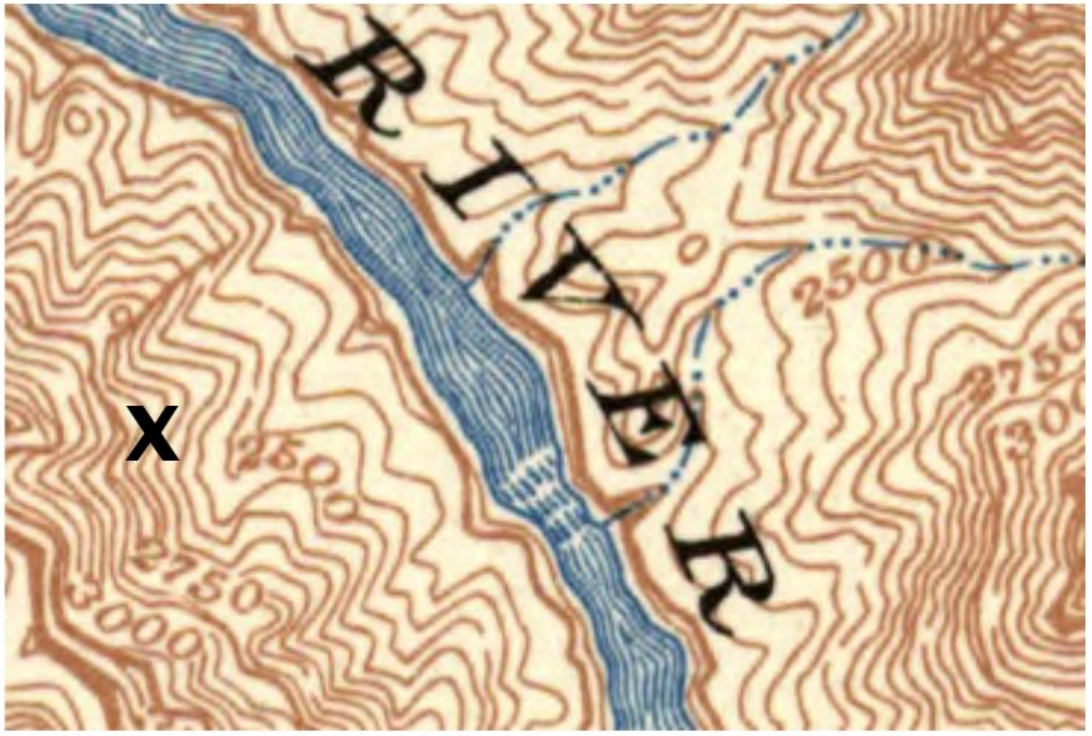
We want the weights that minimize the loss, averaged over all examples:

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

Descenso por gradiente

Intuition of gradient descent

How do I get to the bottom of this river canyon?



Look around me 360°

Find the direction of steepest slope down

Go that way

Our goal: minimize the loss

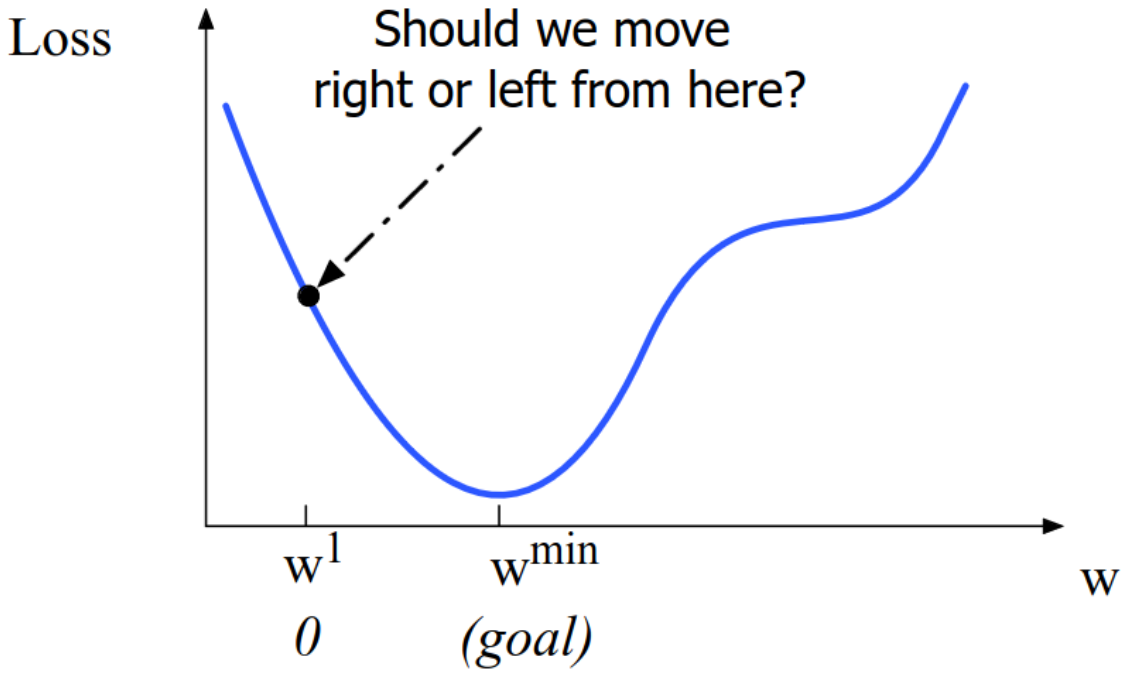
For logistic regression, loss function is **convex**

- 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

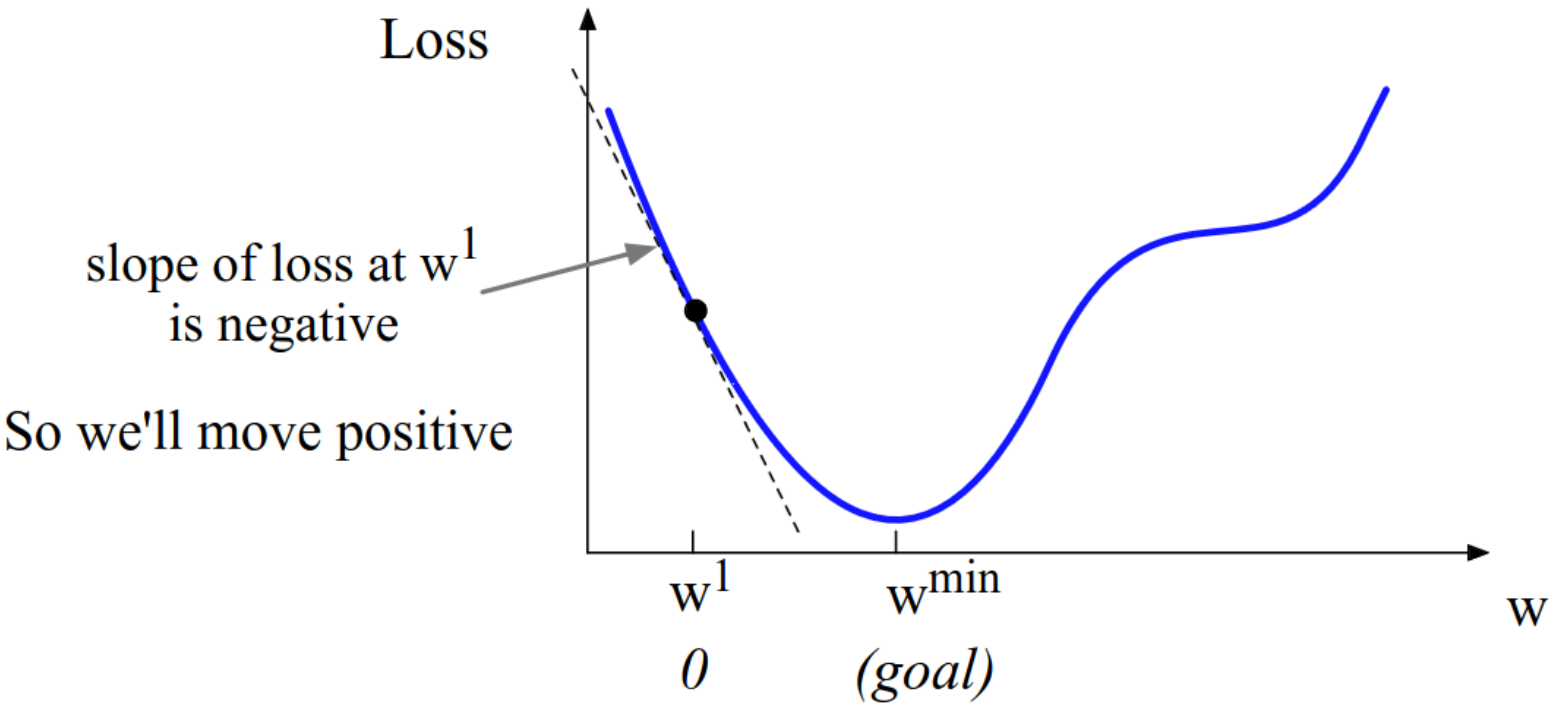
Q: Given current w, should we make it bigger or smaller?

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



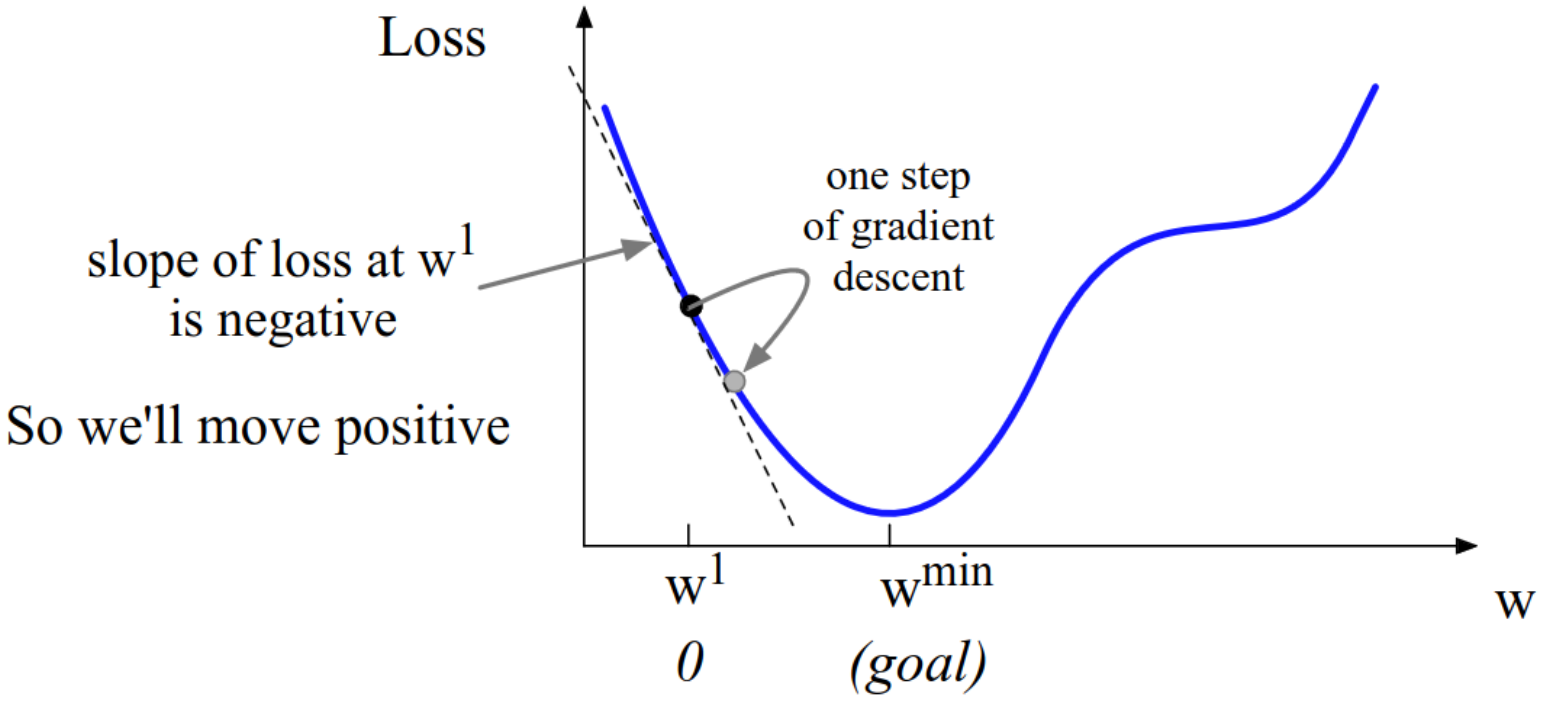
Let's first visualize for a single scalar w

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 ?

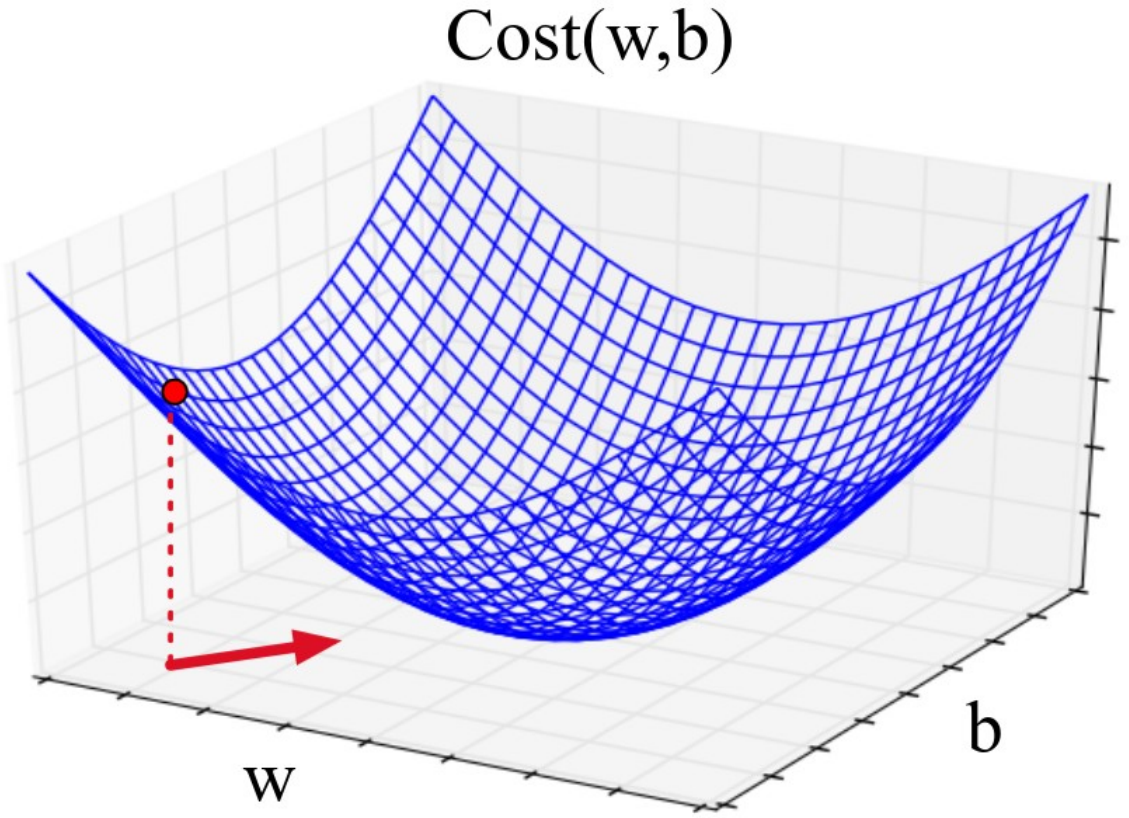
- The value of the gradient (slope in our example) $\frac{d}{dw} L(f(x;w),y)$ weighted by a **learning rate** η
- Higher learning rate means move *w* faster

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

Imagine 2 dimensions, w and b

Visualizing the gradient vector at the red point

It has two dimensions shown in the x-y plane



The gradient

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

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

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

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

The elegant derivative of this function (see textbook 5.8 for derivation)

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

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 $x^{(1)}$, $x^{(2)}$,..., $x^{(m)}$
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 $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # 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

Hiperparámetros, mini-batching y overfitting

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.

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)

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**

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
    - Dropout in neural networks

      Early stopping

      …

Regularization

A solution for overfitting

Add a regularization term $R(\theta)$ to the loss function
(for now written as maximizing logprob rather than minimizing loss)

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

Idea: choose an $R(\theta)$ 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:

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

L1 Regularization (= lasso regression)

The sum of the (absolute value of the) weights

Named after the **L1 norm** $||W||_1$, = sum of the absolute values of the weights, = **Manhattan distance**

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

L1 regularized objective function:

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

Regresión logística multinomial

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 $z = [z1, z2, ..., zk]$ 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 $z = [z_1, z_2, \ldots, z_k]$ of k arbitrary values into probabilities

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

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

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

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 → y=1 neg weight → y=0

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

Multinominal: separate weights for each class:

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