CS 232: Artificial Intelligence

Fall 2023

Prof. Carolyn Anderson
Wellesley College

New policy: earn extra late days

You can earn bonus late days by attending a CS research talk. To be eligible:

- The talk must be about CS research or AI research in a related field
- The talk must be in-person (so that you have the ability to ask questions)
- You must write a paragraph about the talk and what you learned and email it to me.

Research

- *Manipulation-robust citizens' assembly selection* studies how to reduce incentives for people to try to increase their chances of being chosen for the assembly by misreporting their features.

- *The distortion of public-spirited participatory budgeting* studies the welfare of participatory budgeting outcomes in a beyond-worst-case model of voter behavior: instead of considering only their own interests, voters also weigh the interests of others. This model is motivated by the potential for this behavior to be cultivated in practice, via democratic deliberation.

Bailey Flanagan

October 25th

Computer Science Colloquium Series | Fall 2023

Supporting Responsible AI Practices in Public Sector Contexts

Anna is a third year PhD student at Carnegie Mellon's Human-Computer Interaction Institute. Her research focuses on improving the design, evaluation, and governance of AI technologies used to inform complex, consequential decisions in real-world organizations. In addition to her research, she will share prior experiences forming collaborations with public sector agencies, doing research internships with industry groups, travelling to conferences, and mentoring undergraduate students. The session will end with an open Q/A discussion on applying to and doing a PhD in Computer Science / Human-Computer Interaction and other topics.

WELLESLEY

Accessibility and Disability Resources:
accessibility@wellesley.edu

Anna Kawakami '21

Nov 2nd, 12:45-2:00 | SCI H401
Lunch will be served

Questions??? eni.mustafara

November 2nd

Recap

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 (binary) logistic regression: summary

Given:

- a set of classes: (+ sentiment, - sentiment)
- a vector **x** of features `[x1, x2, …, xn]`
  - x1= count("awesome")
  - x2 = log(number of words in review)
- A vector **w** of weights `[w1, w2, …, wn]`
  - w_i for each feature f_i

$$P(y=1) = \sigma(w \cdot x + b)$$

bias

$$= \frac{1}{1+\exp(-(w \cdot x + b))}$$

probability that the review is positive

Classifying sentiment for input x

$x = [3, 2, 1]$ $\quad w = [4, -3, -1]$ $\quad b = -0.5$

$$p(+|x) = P(Y=1|x) = \sigma(wx+b)$$

$$= \sigma([4, -3, -1] \cdot [3, 2, 1] + -0.5)$$

$$= \sigma(4.5)$$

$$= 0.989$$

$$p(-|x) = P(Y=0|x) = 1 - \sigma(wx+b)$$

$$= 1 - 0.989 = 0.011$$

The two phases of logistic regression

Training: we learn weights *w* and *b* using **stochastic gradient descent** and **cross-entropy loss**.

optimizer

training objective: minimize 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

How Does Learning Work?

Slides borrowed from Jurafsky & Martin Edition 3

Learning in Supervised Classification

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

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 as:

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

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

If $y=0$: $p(y|x) = 1-\hat{y}^{1-0} = 1-\hat{y}$

Deriving cross-entropy loss for a single observation x

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

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

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

$= y \log \hat{y} + (1-y)\log(1-\hat{y})$

math.log: put in log space

math.exp: take out of log space

Deriving cross-entropy loss for a single observation x

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

Minimize the cross-entropy loss

Minimize: $L_{\text{CE}}(\hat{y}, y) = -\log p(y|x)$

$$= -[y \log \hat{y} + (1-y) \log (1-\hat{y})]$$

$$= -[y \log \sigma(wx+b) + (1-y) \log (1-\sigma(wx+b))]$$

Does this work 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 .

Let's see if this works for our sentiment example

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

$$p(+|x) = P(y=1|x) = 0.989$$

$$p(-|x) = 0.011$$

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

$$= -[y \log(0.989) + (1-y) \log(1-0.989)]$$

$$= -[1 \cdot \log(0.989) + (1-1) \log(1-0.989)]$$

$$= -\log(0.989) = 0.011$$

What if the true label was 0?

$$p(+|x) = P(y = 1|x) = 0.989$$

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(wx+b) + (1-y) \log(1-\sigma(wx+b))] \\ &= -[y \log 0.989 + (1-y) \log(1-0.989)] \\ &= -[0 \cdot \log(0.989) + (1-0) \log(1-0.989)] \\ &= -\log(1-0.989) \\ &= 4.51 \end{aligned}$$

Let's see if this works for our sentiment example

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

0.011

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

4.51

Sure enough, loss was bigger when model was wrong!

Stochastic Gradient Descent

Slides borrowed from Jurafsky & Martin Edition 3

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:

$$L_{CE}(\hat{y}, y) \longrightarrow L_{CE}(f(x^{(i)}; \theta), y^{(i)})$$

$\hat{y}$

θ: w, b

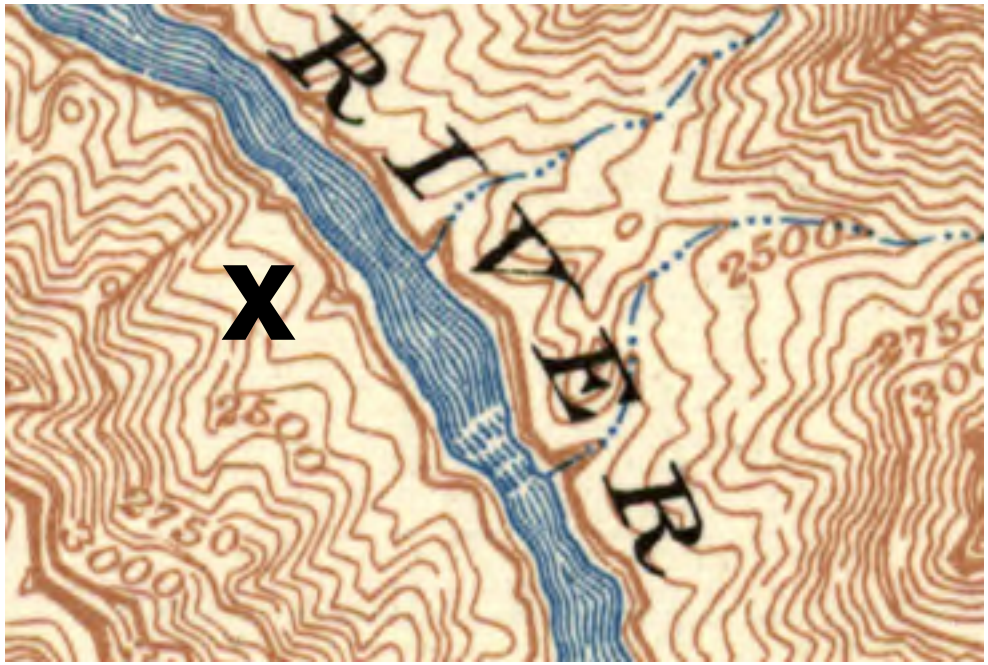
$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(f(x^{(i)}; \theta), y^{(i)})$$

loss for each example

parameters

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

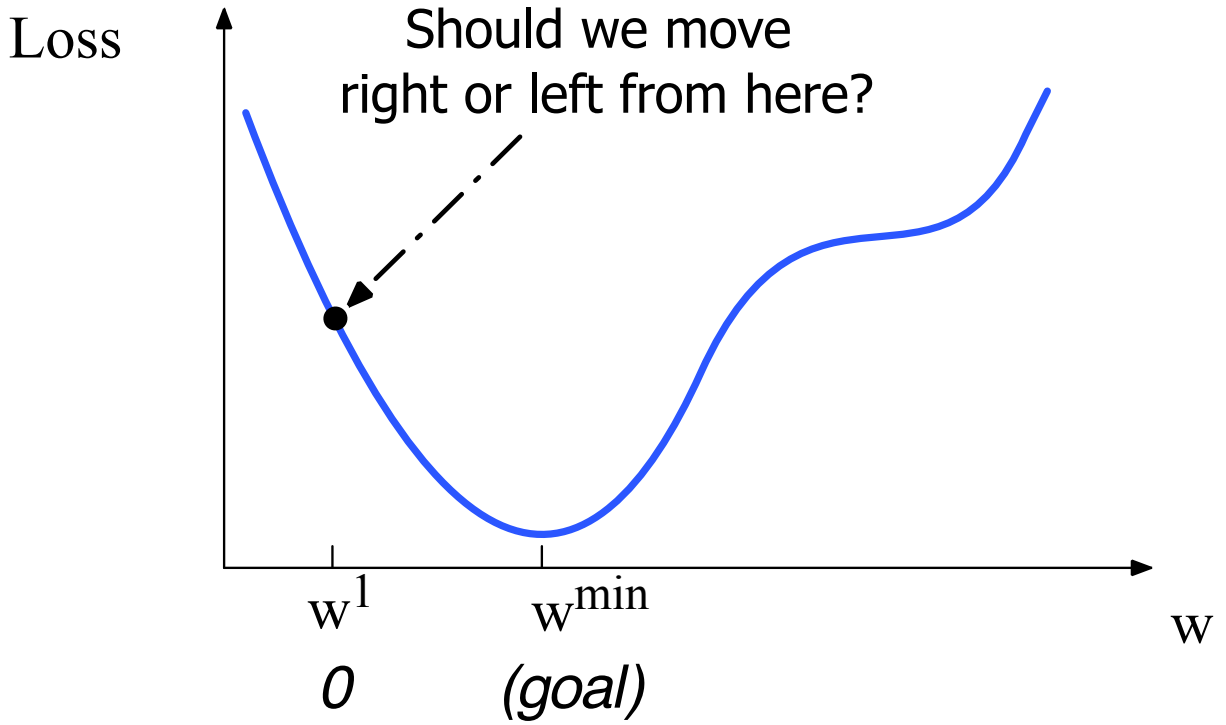
For logistic regression, loss function is **convex**

- 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?

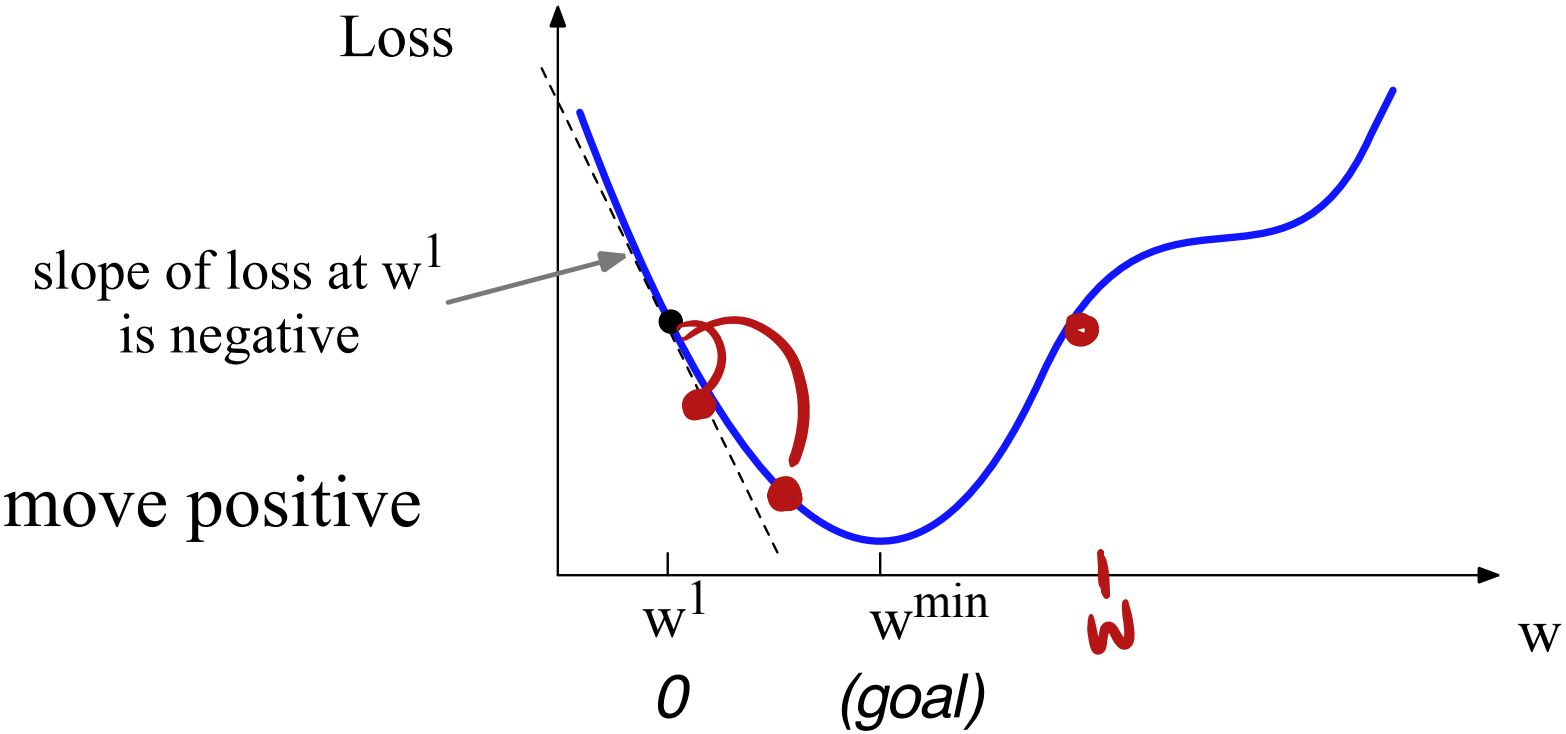
A: Move *w* in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

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

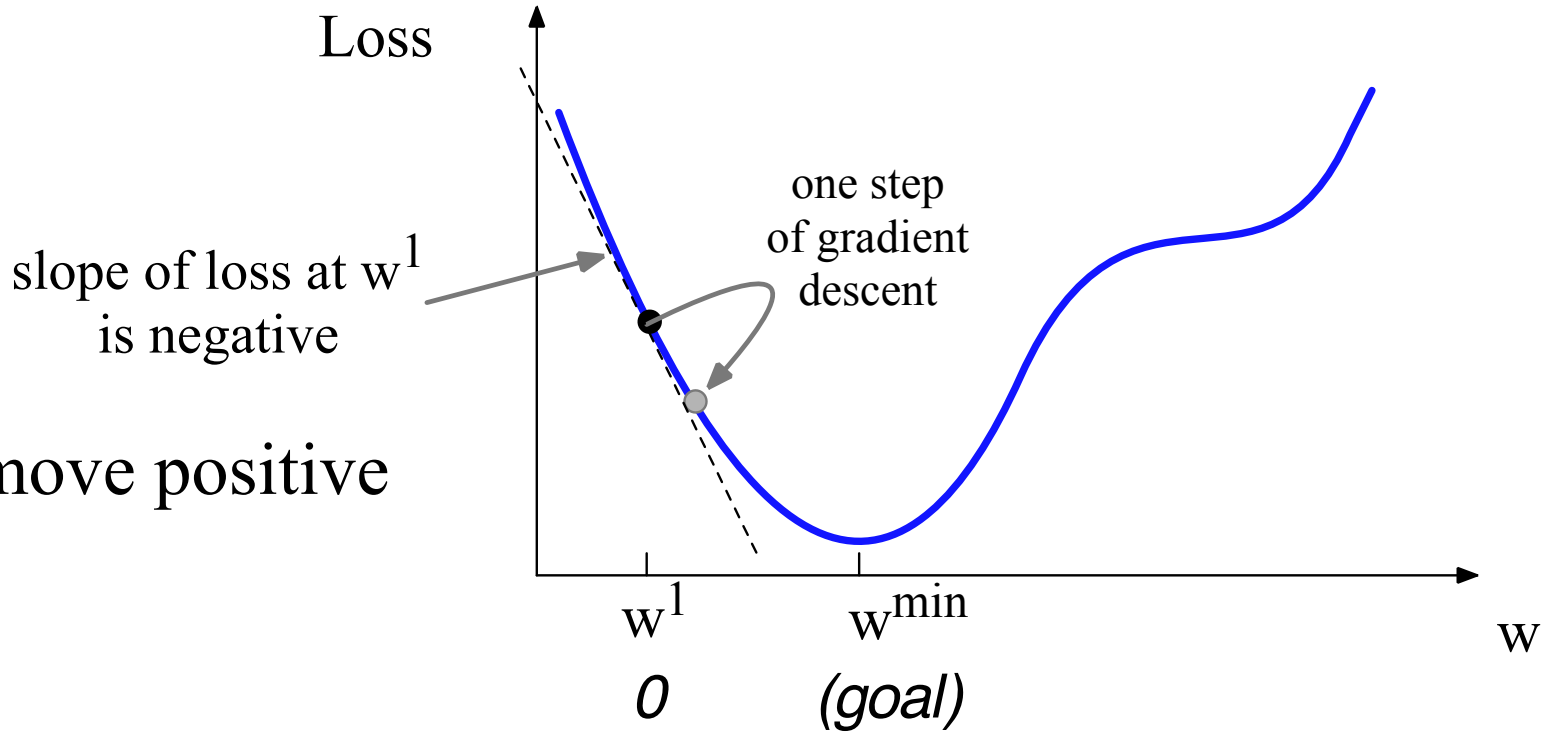
A: Move *w* in the reverse direction from the slope of the function



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



So we'll move positive

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

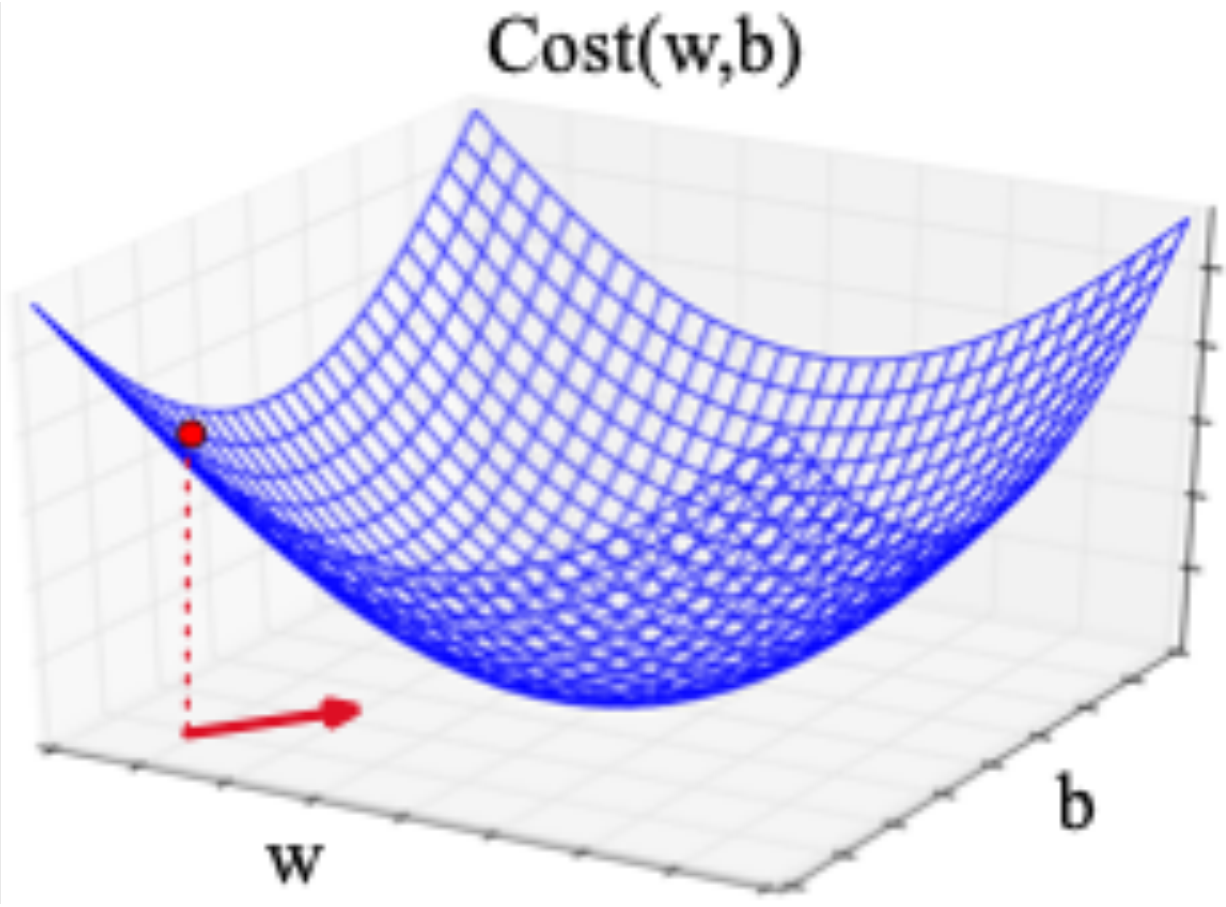
We want to know where in the N-dimensional space (of the *N* parameters that make up θ) we should move.

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



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 # 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 θ to maximize loss?
3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead

return θ

Hyperparameters

hyperparameter: we set
parameter: learned by the model

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.

Game

Logistic Regression Example: Pet Picture Classification

Goal: Classify Pet Pictures

- Dataset: cat + dog pictures
- Goal: classify a picture as either a cat or a dog
- Input: grayscale images

Building a Model

We'll build our model using a machine learning library called **Tensorflow**.

Tensorflow is a Python library, but most functions are implemented in C (so they are fast!).

Tensorflow provides useful abstractions for models:

- **tensor**: n-dimensional container for data
- **layer**: apply functions to an input tensor of n dimensions to produce an output tensor of m dimensions.
- **model**: consist of layers connected together

Example Data

Creating Our Model Architecture

```
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    x = layers.Flatten()(inputs)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


model = make_model(input_shape=image_size, num_classes=2)
keras.utils.plot_model(model, show_shapes=True)
```

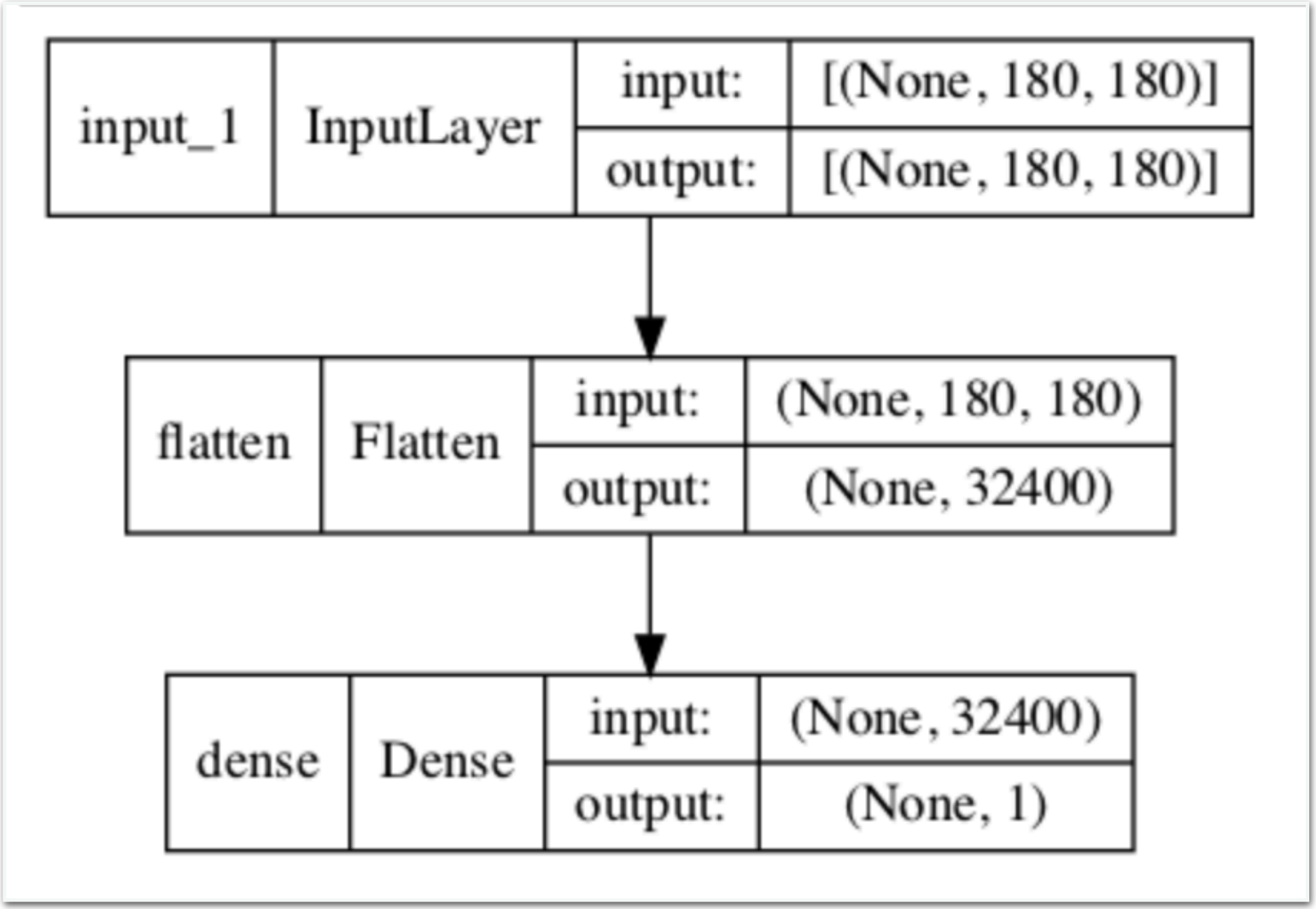
input layer

flatten to a single dimension

select sigmoid or softmax based on number of classes

weights + bias layer - this is the regression bit!

Creating Our Model Architecture



Training

Train the model

```
epochs = 50

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)
```

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**

Evaluating Classifiers

Slides borrowed from Jurafsky & Martin Edition 3

Evaluation

Consider a binary text classification task:

Is this passage from a book a "smell experience" or not?

Towards Olfactory Information Extraction from Text: A Case Study on Detecting Smell Experiences in Novels

Ryan Brate and **Paul Groth**
University of Amsterdam
Amsterdam, the Netherlands
r.brate@gmail.com
p.t.groth@uva.nl

Marieke van Erp
KNAW Humanities Cluster
Digital Humanities Lab
Amsterdam, the Netherlands
marieke.van.erp@dh.huc.knaw.nl

Abstract

Environmental factors determine the smells we perceive, but societal factors factors shape the importance, sentiment and biases we give to them. Descriptions of smells in text, or as we call them 'smell experiences', offer a window into these factors, but they must first be identified. To the best of our knowledge, no tool exists to extract references to smell experiences from text. In

Evaluation

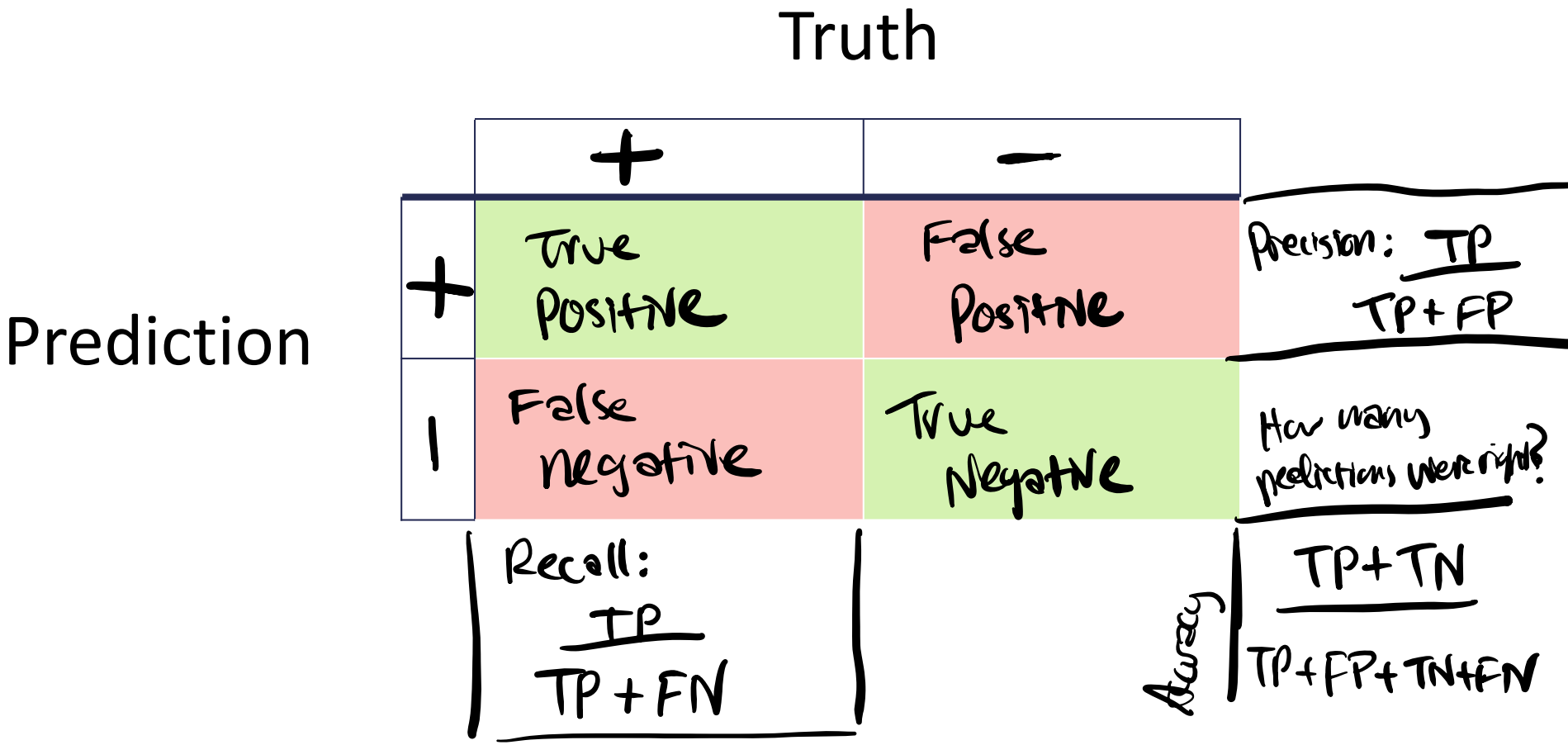
Consider a binary text classification task:

Is this passage from a book a "smell experience" or not?

You build a "smell" detector

- Positive class: paragraph that involves a smell experience
- Negative class: all other paragraphs

The 2-by-2 confusion matrix



Evaluation: Accuracy

Why don't we use **accuracy** as our metric?

Imagine we saw 1 million paragraphs

- 100 of them mention smells
- 999,900 talk about something else

We could build a classifier that labels every paragraph "not about smell"

Evaluation: Accuracy

Why don't we use **accuracy** as our metric?

Imagine we saw 1 million paragraphs

- 100 of them mention smells
- 999,900 talk about something else

We could build a classifier that labels every paragraph "not about smell"

- It would get 99.99% accuracy!
- But the whole point of the classifier is to help literary scholars find passages about smell to study--- so this is useless!
- That's why we use **precision** and **recall** instead

Evaluation: Precision

% of items the system detected (i.e., items the system labeled as positive) that are in fact positive (according to the human gold labels)

$$\text{PRECISION} = \frac{TP}{TP + FP}$$

Evaluation: Recall

% of items actually present in the input that were correctly identified by the system.

$$\text{RECALL} = \frac{TP}{TP + FN}$$

Why Precision and recall

Our no-smells classifier

- Labels nothing as "about smell"

Accuracy = 99.99%

Recall = $\frac{0}{100}$ = 0

Precision = $\frac{0}{0+0}$ error!