Classification with Machine Learning

Kevin Wang

Colaboratory notebook:

https://colab.research.google.com/drive/1Cein0r-J9N2vX1xh24cRLEHgBkJx3p7w

You should know the following things

- Basic matrix operations
- Derivatives
  - If you don’t know – just think of them as slope of function at any given point
- Basic programming skills
  - We use Python

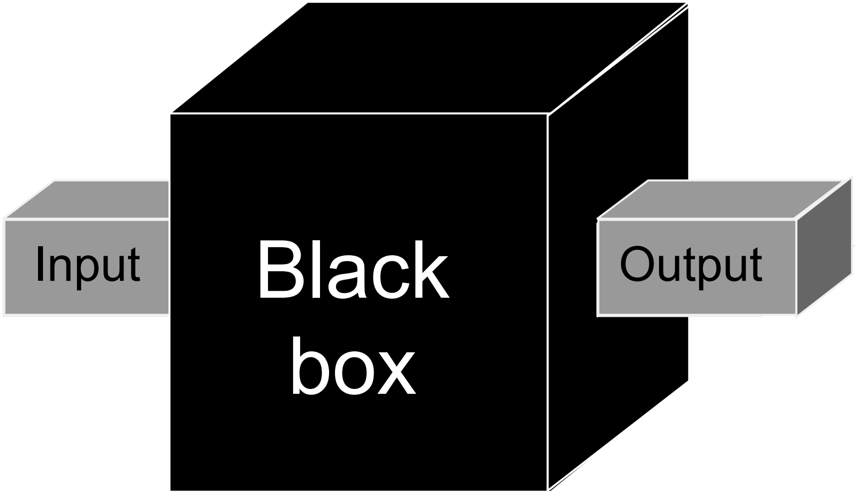
What is machine learning?

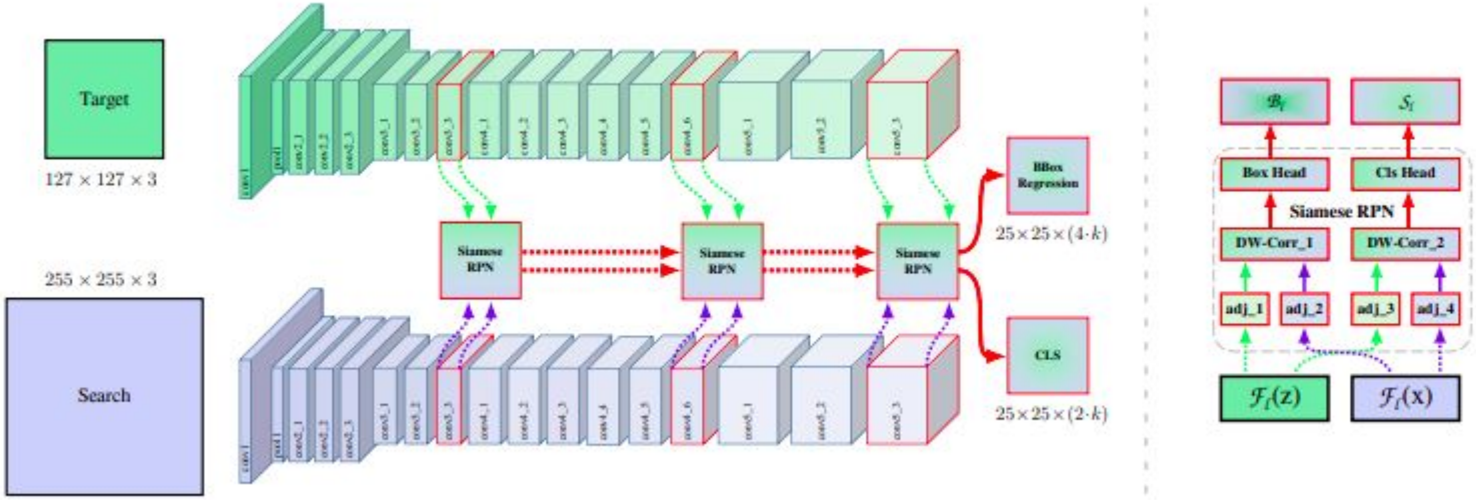
- "Field of study that gives computers the ability to learn without being explicitly programmed"
  - - Andrew Samuel, ML pioneer in the 50s

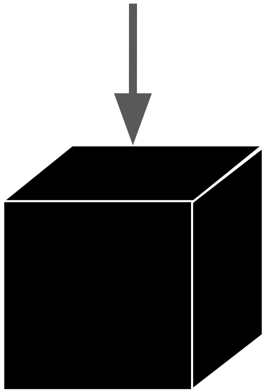
- You create a model that is trained to perform some task
  - Model: an algorithm that can be adjusted by training
  - Training: the stage where the model learns how to perform a task
- Almost all artificial intelligence research today is in the field of machine learning
  - Other AI methods exist, but are not as powerful / promising

Another way to think of it: black box model

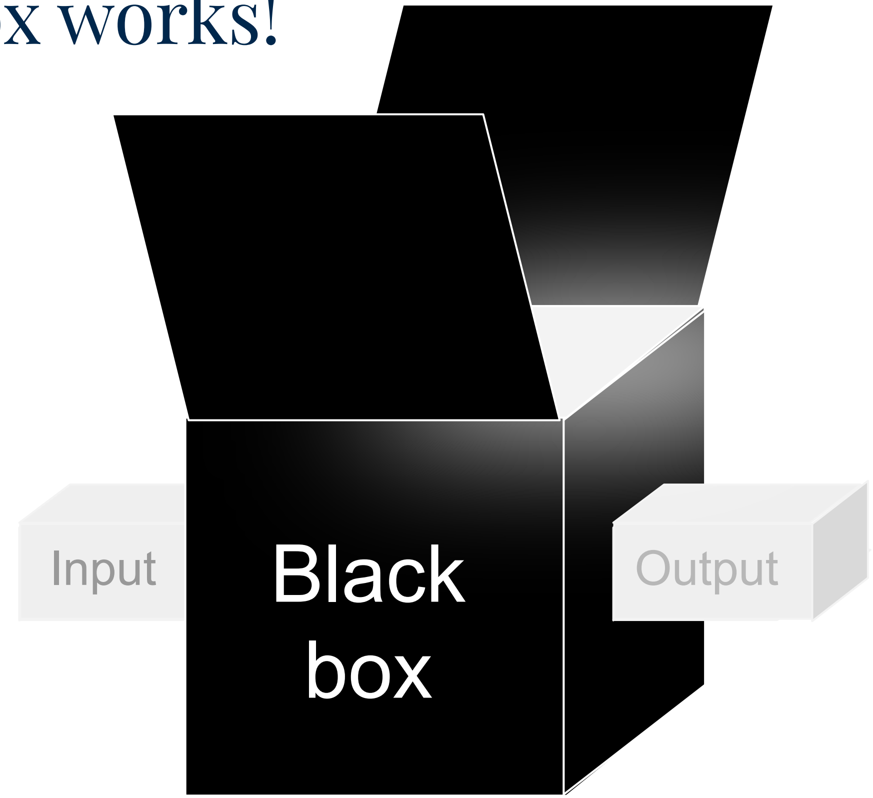
- You can see the input and the output
- A black box takes in the input to produce the output
    - You don't need to configure the inside of the black box to make use of it
- You define the model in your black box using machine learning
    - The power of ML is that we don't need to configure the black box manually!




Target
127 × 127 × 3
255 × 255 × 3
Search
Siamese RPN
Siamese RPN
Siamese RPN
BBox Regression
CLS
25 × 25 × (4·k)
25 × 25 × (2·k)
$\mathcal{B}_l$
$\mathcal{S}_l$
Box Head
Cls Head
Siamese RPN
DW-Corr_1
DW-Corr_2
adj_1
adj_2
adj_3
adj_4
$\mathcal{F}_l(z)$
$\mathcal{F}_l(x)$



The purpose of these tutorials is to learn how the inside of the black box works!



Classification

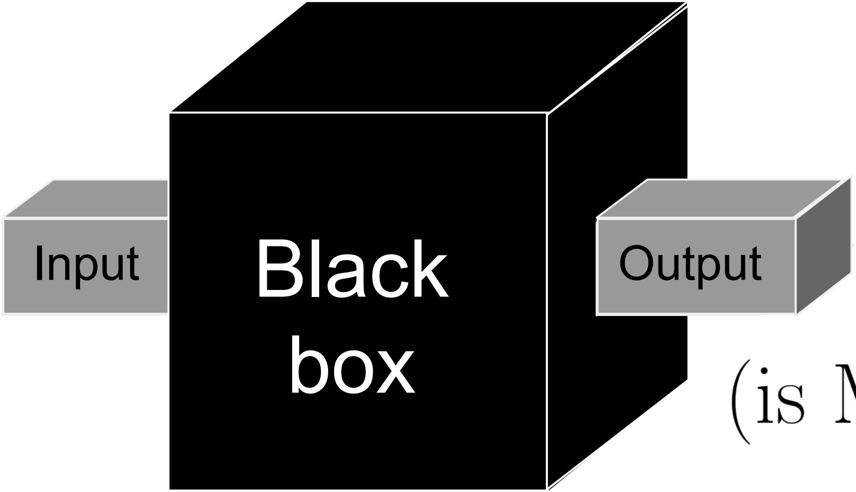
- Input: data
  - Image of UMich logo
  - Information about a tumor
  - Post from Piazza
- Output: what <u>class</u> is the input?
  - Is it a UMich logo or a State logo?
  - Is tumor cancerous or benign?
  - Is this post about an exam or a project?

Classes of input data are their **labels**

Class = 1 (is a Michigan logo)

Class = 0 (is not a Michigan logo)

We’d like the model to do this:



$$1$$

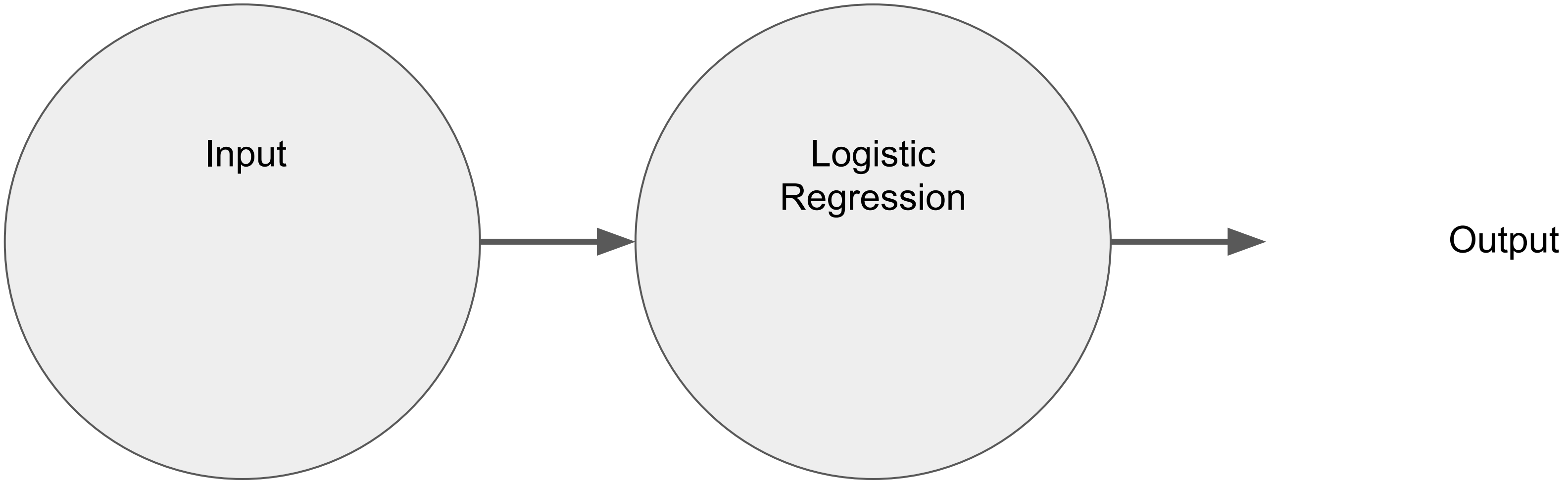
(is Michigan logo)

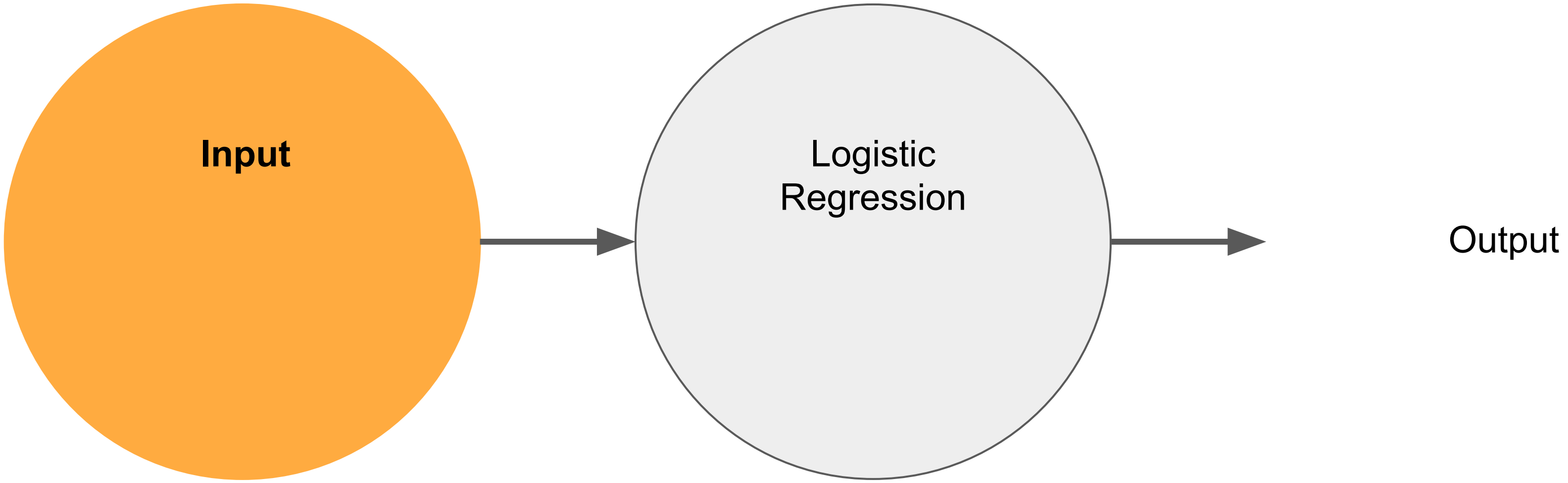
How?

- For today, we're using a model that will perform logistic regression
    - "logistic" indicates the function that makes the decision…more on that later

The flow of logistic regression:

- Training step: requires labeled training data
  - Format the input
    - Break it down into features
  - Several times:
    - Run the model, get your output
    - Based on your output, revise the model
- Test step: input data, run model, and get output
  - Note there is no labeling in this step
- This is similar to how other ML models work


Input
Logistic
Regression
Output


Input
Logistic Regression
Output

The training dataset contains many instances / examples of data

- Use lowercase *x* for a single instance
- Denote which instance using superscript with parentheses
  - *i* is the standard placeholder variable
- We say there are *m* total
  - Not a universal convention!
- Case and superscript are important!
  - Leaving out superscript usually means arbitrary instance

This is the third instance / example in the training dataset

$$x^{(3)}$$

To represent an arbitrary instance, use

$$x^{(i)}$$

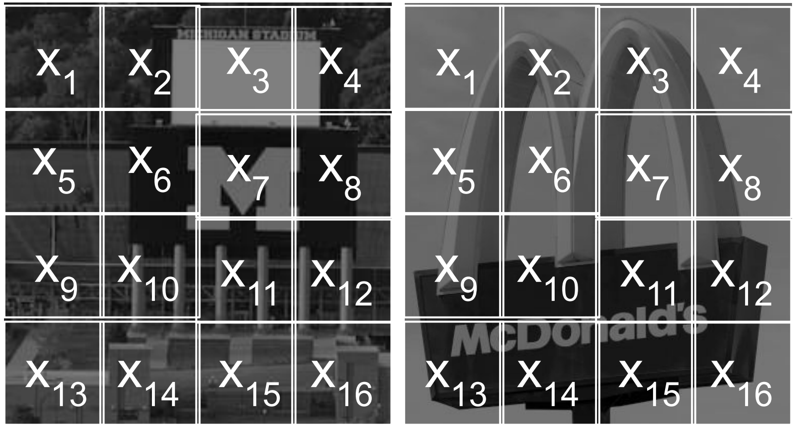
or

$$x$$

Defining features

- Features might be:
  - pixel values (images)
  - size, age, location, growth rate (tumor)
  - words in post (Piazza)
- Features are denoted by subscript:
  - $x_1, x_2, x_3, \ldots$
    - Indices start at 1
- Every instance has the same number of features
  - This number is *n*

Breaking down two images into several features based on pixel value



(pixels not drawn to scale)

Instance with its features is represented by a vector

In general, this is how a single instance
is represented

$$x^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix}$$

- How might we represent the dataset as a whole?

The dataset as a whole is represented by a matrix

- Name is X
  - vector = lowercase, Matrix = Uppercase
- Instances are stored in a row
- One column is one instance (with its features)
  - Rows are less useful for us
- It can often help to think of X as a row vector of column vectors
  - Multiplication

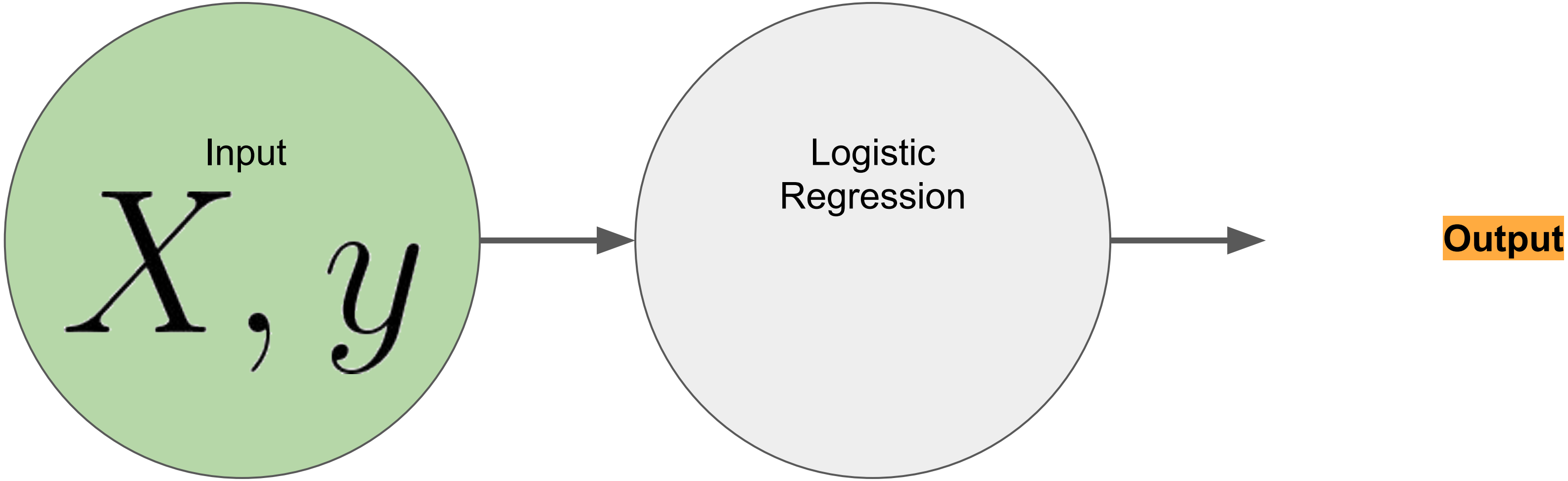
$$X = \begin{bmatrix} | & | & | & & | \\ | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(m)} \\ | & | & | & & | \\ | & | & | & & | \end{bmatrix}$$

Don’t forget the labels:

- Also stored as a vector
  - A row vector this time
- Superscript in parentheses denotes corresponding example $x^{(i)}$ from the dataset

Note superscripts to denote specific examples

$$y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \dots & y^{(m)} \end{bmatrix}$$


Input
X, y
Logistic
Regression
Output

The output is a vector of predicted probabilities

- Output is a vector of predicted classes for corresponding instances in *X*
  - We call this vector "*y_hat*"
  - Also a row vector
- At test time, you'd round the predicted values
  - Training uses unrounded values

The *i*th value in *y_hat* equals the predicted probability (given the *i*th training example) that $y^{(i)} = 1$

$$\hat{y}^{(i)} = \text{predicted } P(y^{(i)} = 1 | x^{(i)})$$

y_*hat* is used to denote the entire vector

$$\hat{y} = \begin{bmatrix} \hat{y}^{(1)} & \hat{y}^{(2)} & \hat{y}^{(3)} & \ldots & \hat{y}^{(m)} \end{bmatrix}$$

And now a brief digression…

…to matrix dimensions

Pop quiz:

- What are the dimensions of *y*?
    -
- What are the dimensions of *y_hat*?
    -
- What are the dimensions of a single training example $x^{(i)}$?
    -
- What are the dimensions of *X*?
    -
- Why should every training instance have the same number of features?
    -

- Let *m* = number of instances and *n* = number of features

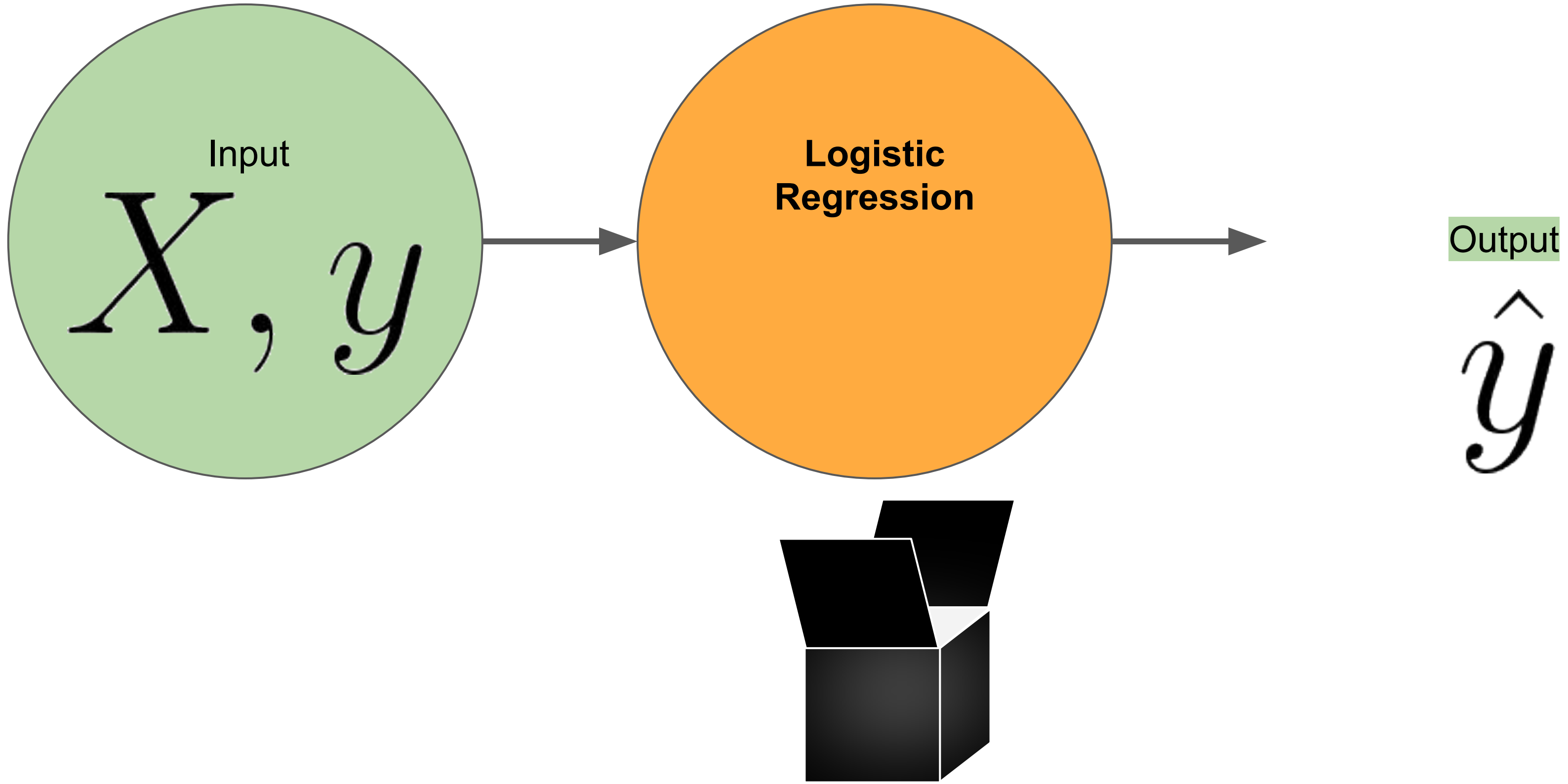
Pop quiz:

- What are the dimensions of y?
  - (1, m)
- What are the dimensions of *y_hat*?
  - (1, m)
- What are the dimensions of a single training example $x^{(i)}$?
  - (n, 1)
- What are the dimensions of X?
  - (n, m)
- Why should every training instance have the same number of features?
  - 1. it wouldn't make sense otherwise!
  - 2. keep dimensions of $x^{(i)}$ consistent for all i

- Take questions

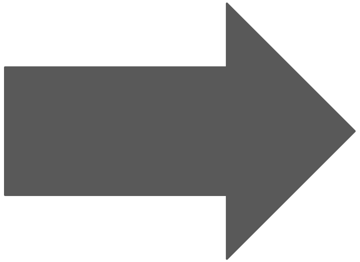
Vectors and matrices simplify our math

- And they will help with the intuition for logistic regression
  - The inner workings of the black box…
  - …and the interesting part of today

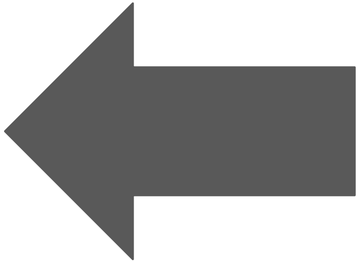

Input
X, y
Logistic
Regression
Output
$\hat{y}$

There are two steps in logistic regression

- First step: forward propagation
  - Given matrix of instances *X*, make predictions on all of them

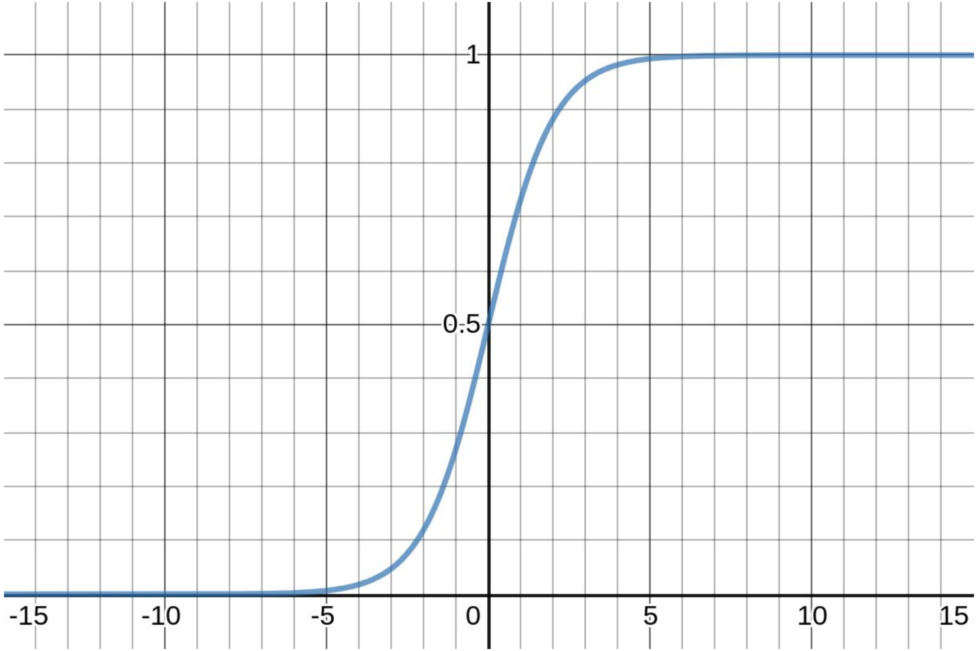


- Second step: backward propagation
  - Calculate derivatives
  - Update values used in the forward propagation step



First step: forward propagation

The sigmoid / logistic function makes the predictions

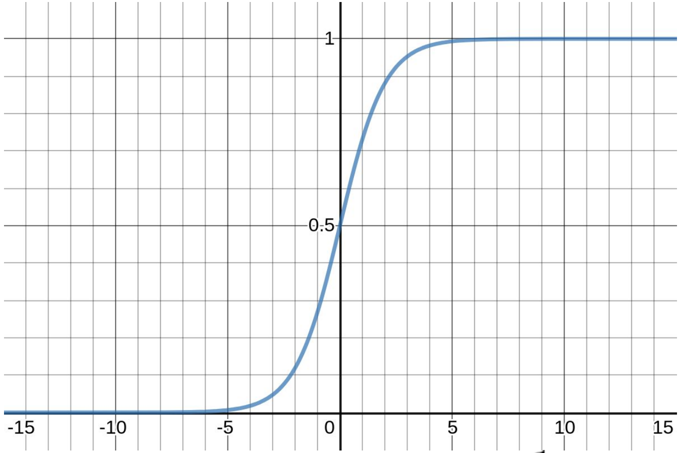


Sigmoid function: $\hat{y}^{(i)} = \sigma(x^{(i)}) = \dfrac{1}{1 + e^{-x^{(i)}}}$

Why is the sigmoid function a good choice?

- For all possible values *x*, outputs values between 0 and 1
  - This will be our predicted probability that *x* is class 1
- The more extreme the value of *x* is, the more certain the output
  - Very positive *x*: *y_hat* ≈ 1
  - Very negative *x*: *y_hat* ≈ 0

$$\hat{y}^{(i)} = \text{predicted } P(y^{(i)} = 1 | x^{(i)})$$



$$\hat{y}^{(i)} = \sigma(x^{(i)}) = \frac{1}{1 + e^{-x^{(i)}}}$$

Why won’t this work?

Why won't this work?

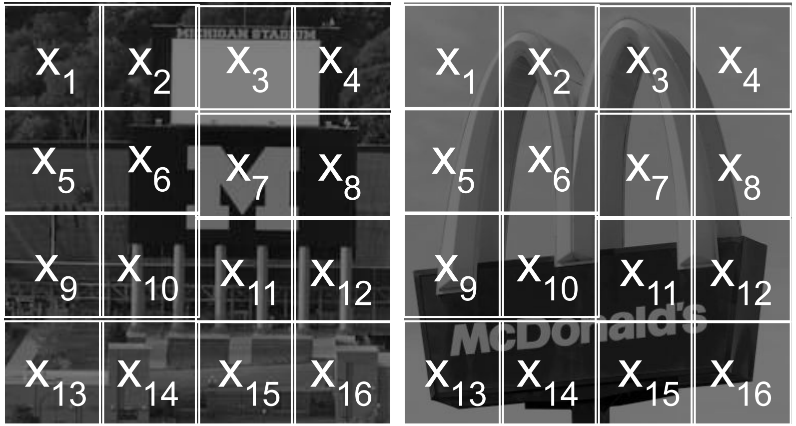
- Setting a scalar equal to a vector doesn't make sense

$$\hat{y}^{(i)} = \sigma(x^{(i)}) = \frac{1}{1 + e^{-x^{(i)}}}$$

- y_hat$^{(i)}$ is a scalar…
  - …but $x^{(i)}$ is a vector

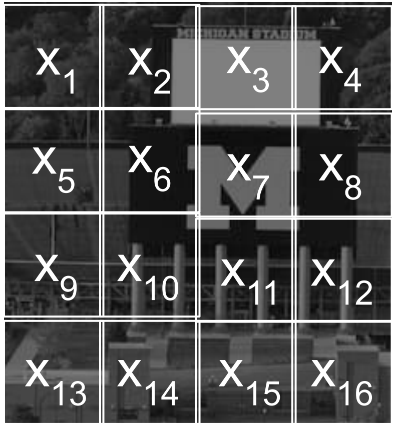
Why won't this work?

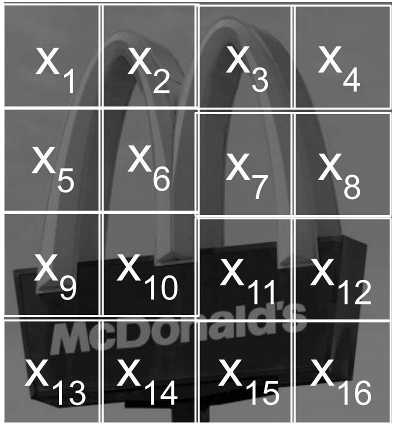
- Do all features have equally important information?
  - Are the border pixels of an image as important for classification as the center pixels?
  - Does the word "the" indicate as much about a Piazza post as the word "midterm"?



Why won't this work?

- Do all features have equally important information?
  - Are the border pixels of an image as important for classification as the center pixels?
  - Does the word "the" indicate as much about a Piazza post as the word "midterm"?
- Other way of wording it: do all features have the same <u>weight</u>?





Our goal is to take a weighted sum of the features

- This will be a scalar *z*
  - So we can properly calculate scalar *y_hat*$^{(i)}$
  - We'll need to calculate a $z^{(i)}$ for every $x^{(i)}$
- We'll need to represent our weights somehow

$$z^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots + w_n x_n^{(i)}$$

The weight vector

- Multiply every feature by a weight
    - Weights are stored in vector *w*
    - Each feature has a corresponding weight

$$x_1 \rightarrow w_1 x_1$$

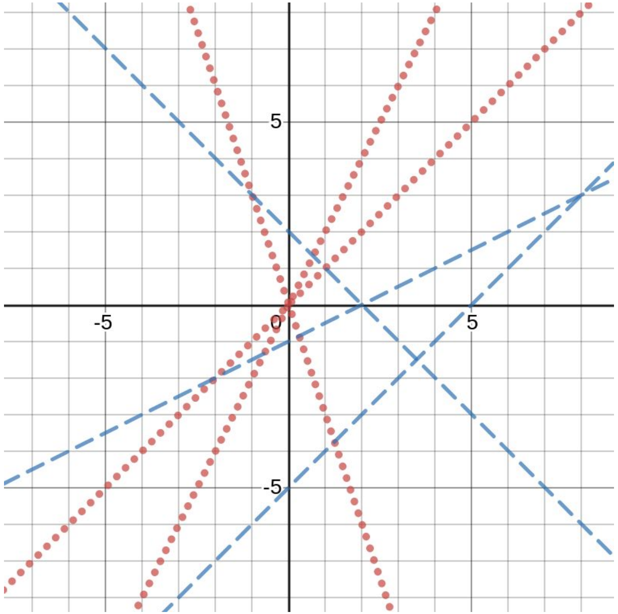
$$x_2 \rightarrow w_2 x_2$$

$$\cdots$$

$$x_n \rightarrow w_n x_n$$

The bias

- Add a bias value to the weighted sum $z^{(i)}$
  - Bias is denoted as *b*
  - Think of it like y-intercept versus no y-intercept – you have more options for your model



Representing weights and bias

- We just need one bias term *b*
  - *b* is not a vector but a scalar

- What are the dimensions of *w*?
  - How many weights do we need?

Representing weights and bias

- You have to have a weight value for every feature, but just add one bias value
  - So there are *n* weights, 1 bias
- The formula, with *z* being the result:
  - Note $w^T x^{(i)}$ is the same as $w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots$

$$z = w^T x + b$$

- We should technically be clearer here – how?

Weight vector: multiply

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \ldots \\ w_n \end{bmatrix}$$

Bias scalar: add

$$b$$

One value of z per instance, and a z vector

- The *i*th z value is denoted with same parenthetical superscript as $x^{(i)}$, $y^{(i)}$

Capital *z* is a vector of size *m*:

$$z^{(i)} = w^T x^{(i)} + b$$

$$z = \left[z^{(1)} \;\; z^{(2)} \;\; z^{(3)} \;\; \ldots \;\; z^{(m)} \right]$$

The beauty of matrices

- The equation for entire vector *z* is easy to compute using matrix multiplication
    - Python has libraries to work with matrices

$$z = w^T X + b$$

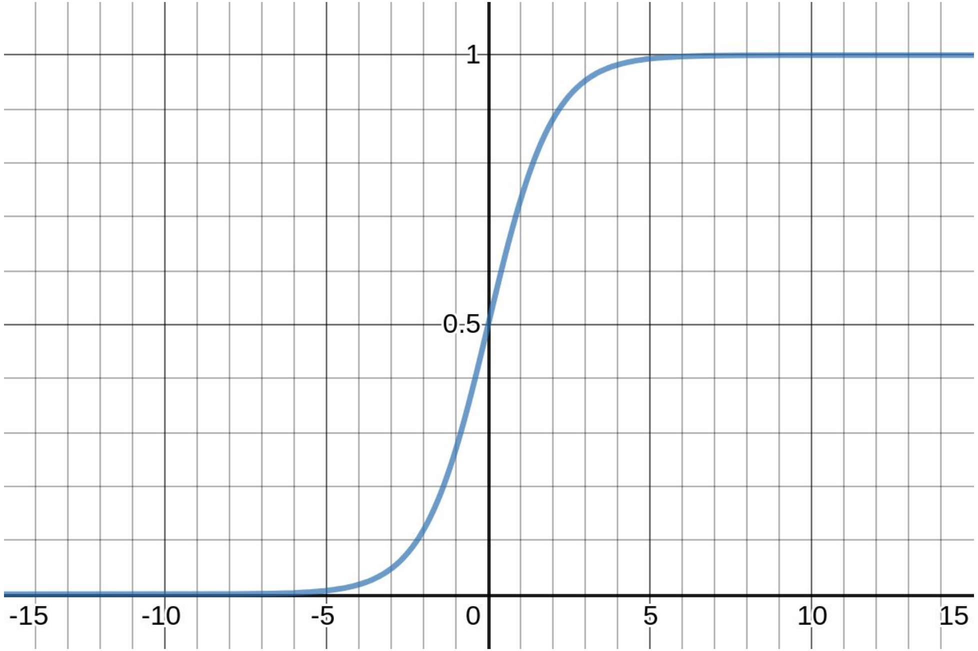
The beauty of matrices

- The equation for entire vector z is easy to compute using matrix multiplication
  - Python has libraries to work with matrices

$$z = w^T X + b$$

$(1, m)$ $\quad$ $(n, 1)^T = (1, n)$ $\quad$ (n, m) $\quad$ $(1, 1)$ added m times

The *i*th value of y_hat is computed on *i*th value in z



Sigmoid function: $$\hat{y}^{(i)} = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$$

Detail: sigmoid function is an activation function

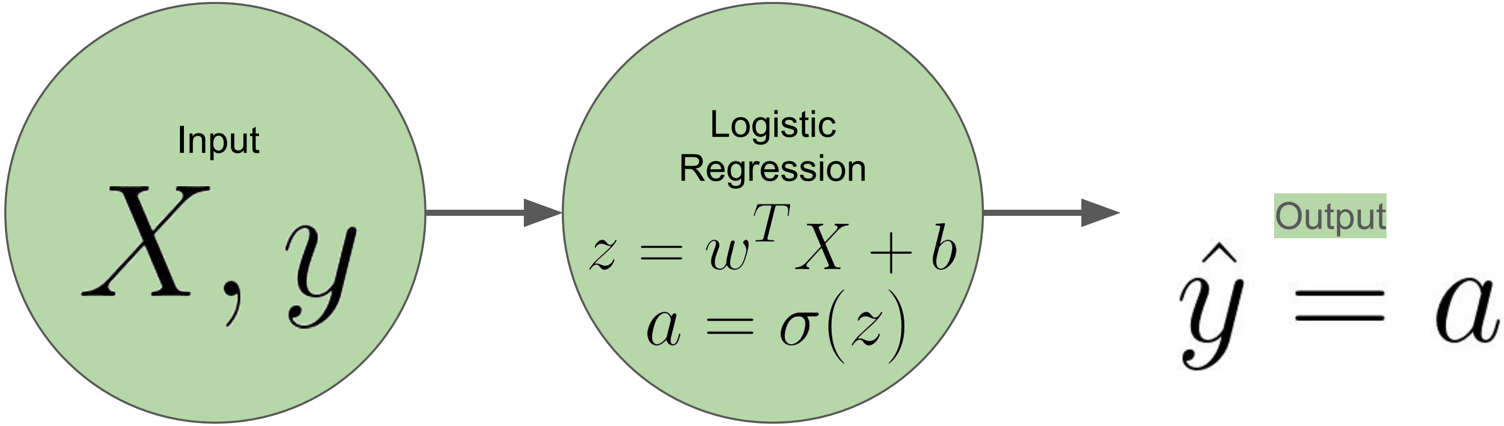
- "How much does *z* 'activate' the function?"
- *y_hat* will also be called a, and *y_hat*$^{(i)}$ will also be called a$^{(i)}$

$$a = \hat{y}$$

$$a^{(i)} = \hat{y}^{(i)} = \frac{1}{1 + e^{-z^{(i)}}}$$

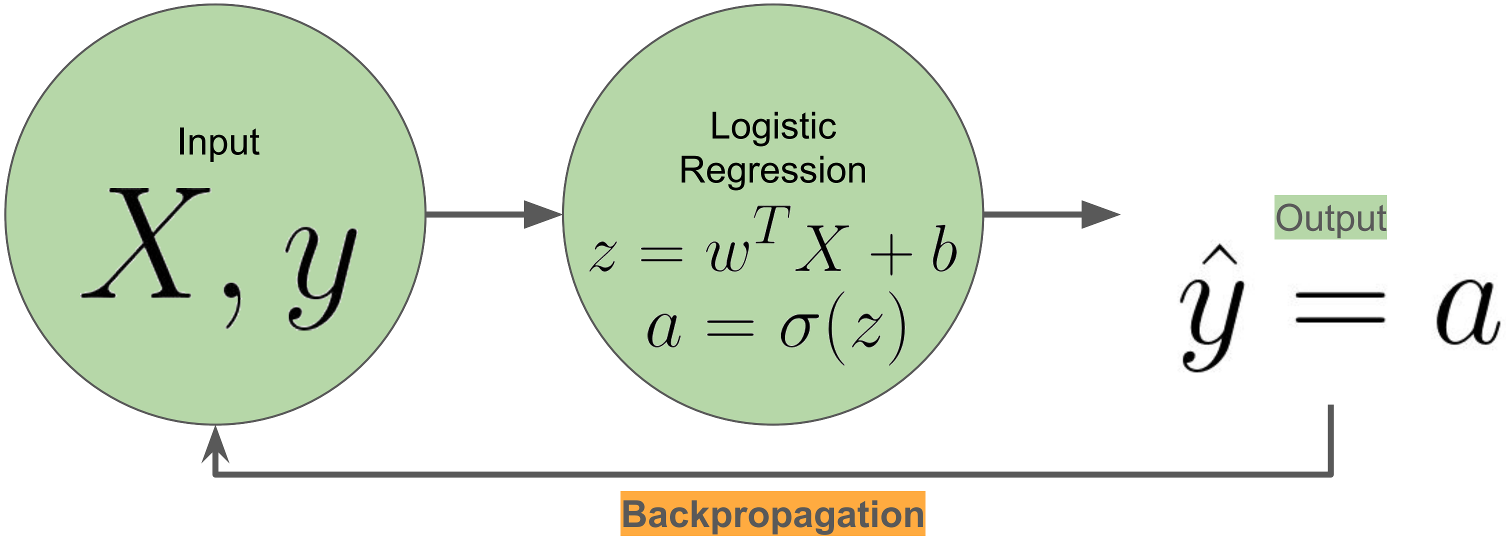
Questions?

But how do we get good predictions?



How to choose good *w*, *b*?

Second step: backward propagation



Two parts to backprop:

- Calculating loss / cost
  - Cost function is higher when model performs poorly, lower when model performs well
- Gradient descent
  - Use derivatives to adjust the values of w and b

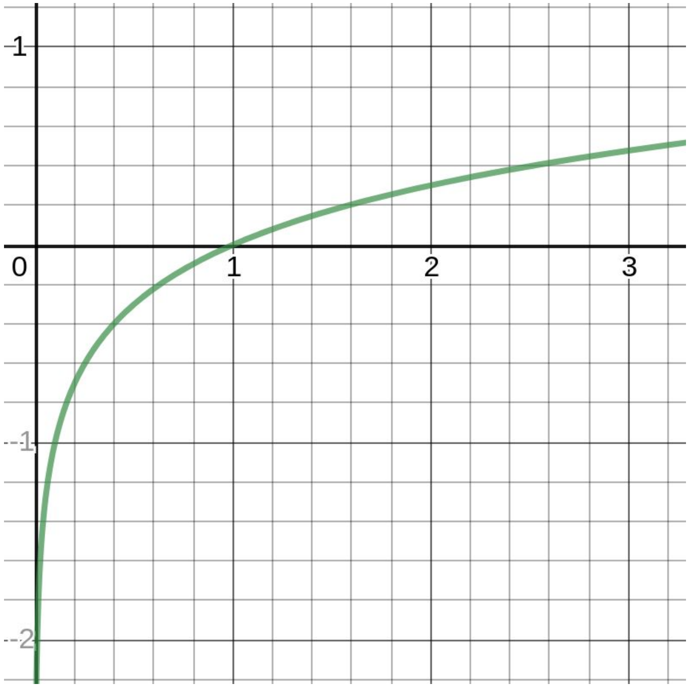
Cost of one example breaks down into two cases:

- If $y^{(i)} = 1$:
  - If $\text{y\_hat}^{(i)} \approx 1$, model did well and should have low cost
  - If $\text{y\_hat}^{(i)} \approx 0$, model did poorly and should have high cost
- If $y^{(i)} = 0$:
  - If $\text{y\_hat}^{(i)} \approx 0$, model did well and should have low cost
  - If $\text{y\_hat}^{(i)} \approx 1$, model did poorly and should have high cost

Cost of one example is a piecewise function

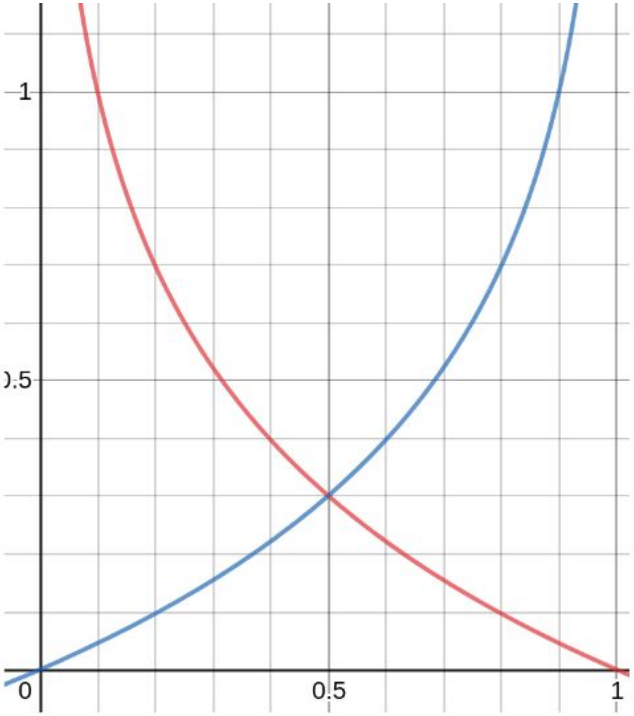
- We can cleverly use logarithms for our costs:

$$L(\hat{y}, y) = \begin{cases} y = 1 & -\log(\hat{y}) \\ y = 0 & -\log(1 - \hat{y}) \end{cases}$$



- What about $y\_hat^{(i)} > 1$ or $y\_hat^{(i)} < 0$?

Cost of one example looks like this when graphed



- Red: y = 1
- Blue: y = 0

$$L(\hat{y}, y) = \begin{cases} y = 1 & -\log(\hat{y}) \\ y = 0 & -\log(1 - \hat{y}) \end{cases}$$

Total cost should take into account all examples

- We simply take the average:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

- J can be represented in terms of *w* and *b*

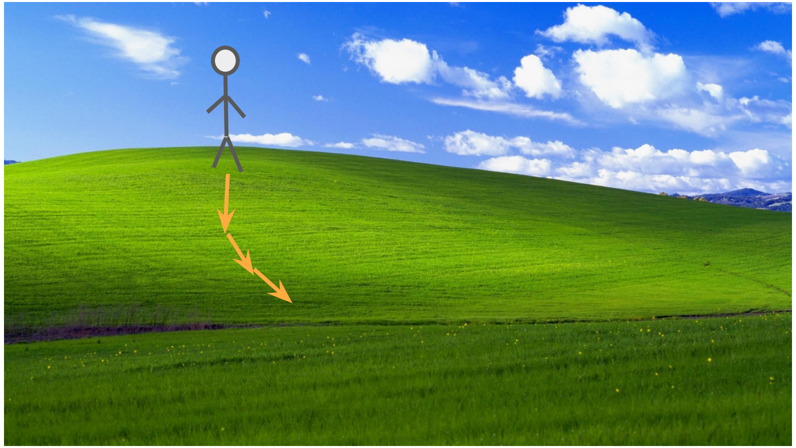
Note: Cost on one example is usually written like so

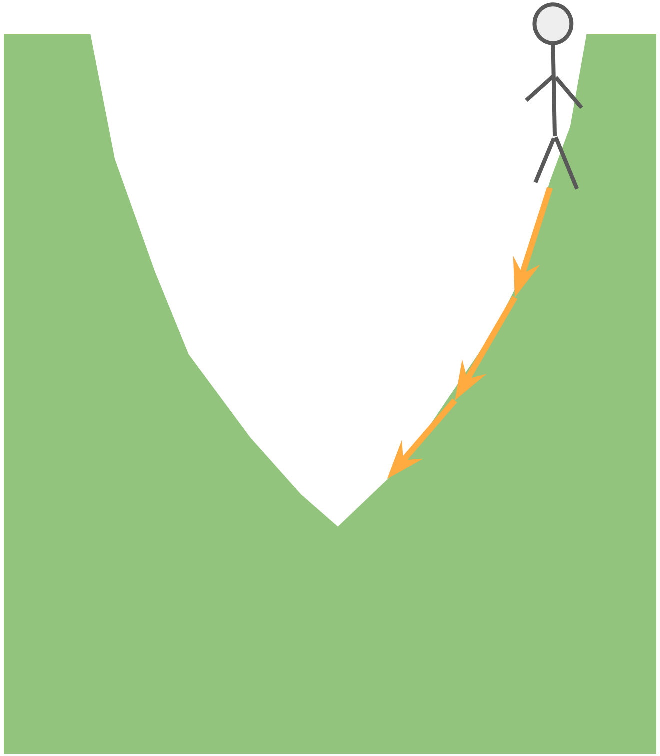
$$L(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

- This is effectively the same as the piecewise function, but is quicker to implement in code and makes computation faster
  - Plug in values for *y* to see why

Gradient descent is like descending a hill

- Objective is to get to the bottom
- Go down in the direction of steepest slope
  - What would this look like in 3 dimensions?
  - What about > 3?

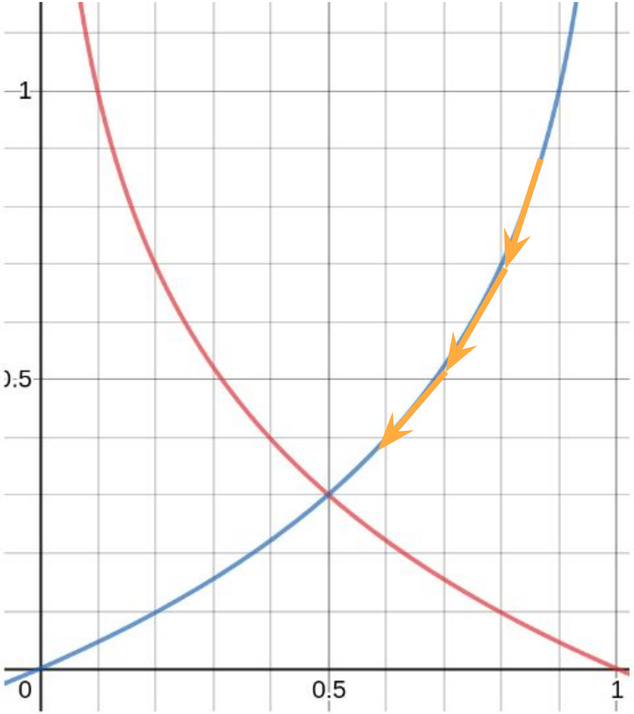




Gradient descent is like descending a hill

- We calculate slope using derivatives
  - “slope at a point, based on some variable”
- Subtract the current value by the derivative

- We will not prove the correctness of the derivatives – it’s more important to know why we’re using them



We calculate 3 important derivatives

- All derivatives are with respect to the total cost function J
    - J depends on multiple variables, which means we're really calculating partial derivatives
- As shorthand, we write only the denominators of the derivatives

- We use:
    - dz, an intermediate value
    - dw, used to update w
    - db, used to update b

dz

- Simply subtract labels *y* from predictions *a*

$$dz = a - y$$

dw

- X is (n, m) while dz is $(1, m)$
  - Transposed to $(m, 1)$

$$dw = \frac{1}{m}(Xdz^T)$$

db

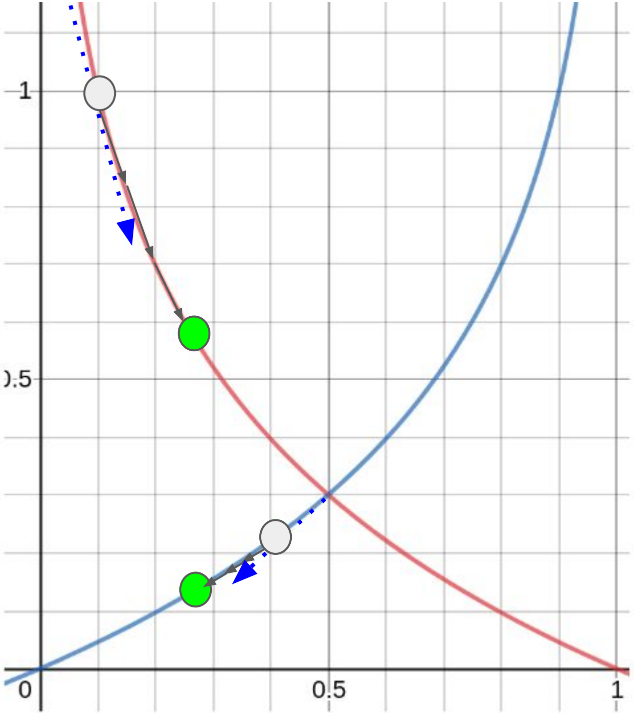
- Average of the elements in *dz*

$$db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)}$$

From variable values, subtract their derivatives

- Moving down the hill
  - The “bottom” of the hill is the minimum of the cost function

- We usually scale down the size of this step
  - Learning rate, α



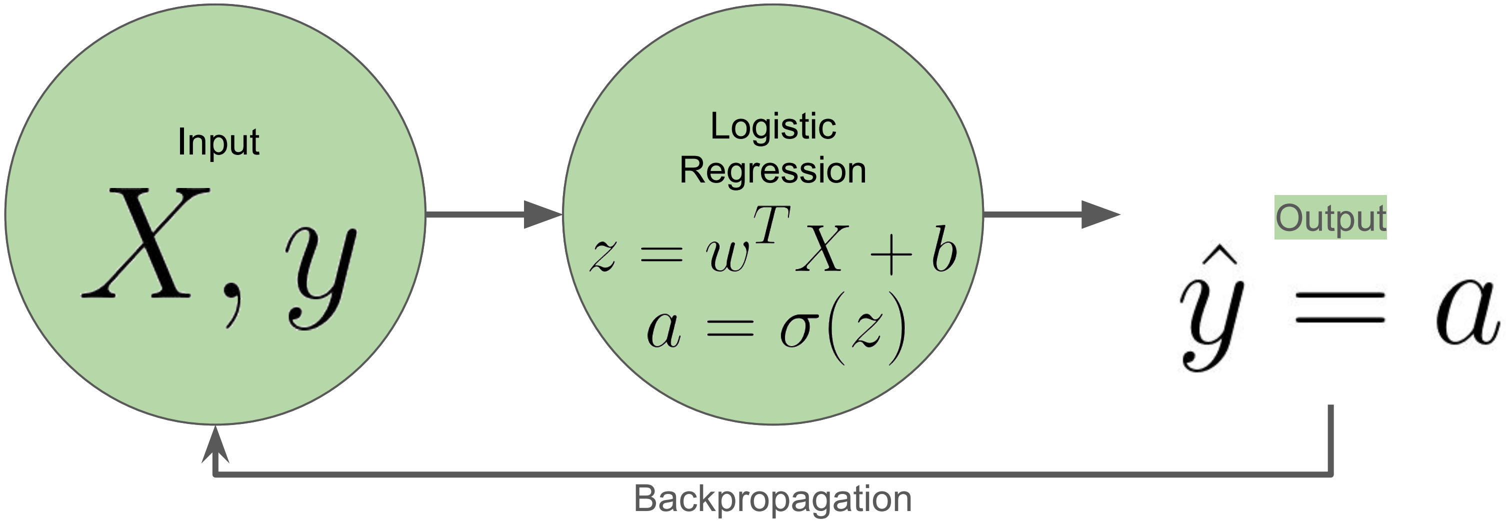
We update *w* and *b* with the derivatives

$$w := w - \alpha dw$$

$$b := b - \alpha db$$

- No reason to update *z* because it depends on *w* and *b*

And that’s the entire process!



- That is one iteration – now do it for thousands of iterations

Review

- Input:
  - Training set X with *m* instances $x^{(i)}$
    - Each instance has *n* features x_1, x_2, ...
  - Labels y with *m* labels $y^{(i)}$
- Output:
  - Predicted probabilities y_hat with *m* predictions $\hat{y}^{(i)}$

Review

- Logistic regression:
  - Weight vector *w* and bias value *b*
    - Use these to calculate *z*
  - Input *z* into the activation function, the logistic / sigmoid function
    - The value we get is *a*
  - Fine-tune the weights by running gradient descent
    - Subtracting slopes (partial derivatives)

Table of important values

Name	Calculated during:	Dimensions	Purpose
X	Input	(n,m)	Input dataset
y	Input	$(1,m)$	Labels
y_hat (=a)	Forward propagation	$(1,m)$	Predictions made by model
w	Backward propagation	$(n,1)$	Weights, used in forward prop
b	Backward propagation	(1,1), scalar	bias, used in forward prop
z	Forward propagation	$(1,m)$	Intermediate value in forward prop
J	Backward propagation	(1,1), scalar	Measures model inaccuracy

Questions?