20211101 Lecture

Monday, November 1, 2021 4:25 PM

- No prof lecture hours this week
  - Can email and request schedule time
- Next Monday:
  - Originally was supposed to be AI ethics guest lecturer
  - They will be able to attend the other lecture (Germany time zone)
  - Just watch the recording or attend other section remotely
-
- Midterm Presentations

==========

-
- Nathan Ho
  - First place
  - Senior BA/MS CS
  - Getting started
    - Determine size of problem
    - ID main challenges
    - Determine if there is potential for the task to be possible
  - Approaching the problem
    - Starter code
      - Try to understand data through code
    - Code and data are equivalent

        - Turing machine
        - U(<M, x>) = M(x)
        - Treat M as the data instead
        - Treat M as a program and run M on x
        - or
        - Treat data …
    - Want to understand the code through the data
- Scarce resources
    - Computer time
    - Programmer time
    - Scarcity of resources addressed with
        - General workflow of Program
            - Read data -> train -> validate
                - Read: 15 sec
            - Make small change to code and rest gets reused
                - Train: At most 1 hour: train on 1k, 10L, 100K
            - The programmer doesn't have to interact with the problem again until the end
                - Validate
            - Python pickle library
                - Save pickle object
                    - So you don't have to keep re-fit model, redo TFDIF, etc.
            - Training model:
                - You hit a limit pretty fast
            - Test a lot of ideas on small datasets to

compare between the models
        - Extrapolate how well it will perform on full dataset and how long it would take to train
    - General workflow of Programmer
        - Do workflow of program, re-evaluate and reconsider trajectory of program
        - Write the code of the next program while one is running
            - To avoid wasting computer time in between
            - Research method
            - Refactor code
            - Create visualizers
            - Evaluate workflow
                - If there are annoying steps, try to automate
            - Which experiments are you going to run?
            - Anticipate outcomes of these experiments
                - That way you are ready for the next step
            - Plan out next work session
                - When & what is first task
- Computer screen management
    - Terminal
        - Model (printing)
        - Upper left

- Terminal
  - Viewer/General commands
  - Lower left
- Text editor
  - On the right
  - Notepad++/Code
    - His favorite is notepad++
- If terminal doesn't need to be monitored
  - Use left side of the screen to research documentations & research papers
  - If want to compare with own results, have research upper left and Terminal viewer on lower left

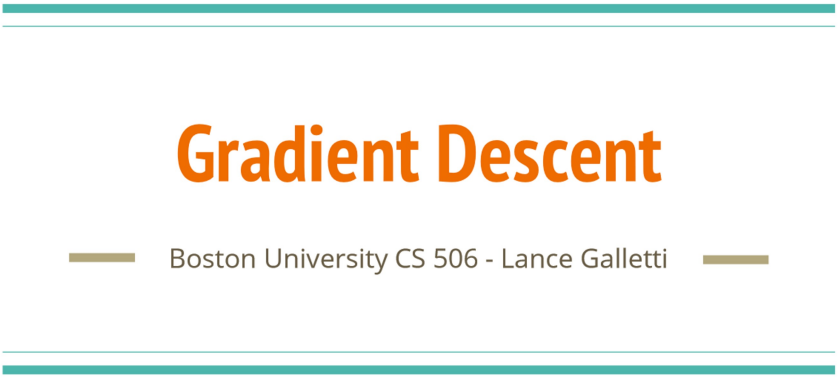
==========

- Prof again:
- Productivity and workflow matters a lot
- sklearn LogisticRegression (slow on full dataset)

20211101Gradient Descent

Monday, November 1, 2021 3:08 PM

- This will transition us into Neural Networks



Logistic Regression Revisited

Given a 2 x 2 grid where each cell a_{ij} can take on one of two colors c_1 and c_2, find a function that can identify the following diagonal pattern:

- Has a diagonal pattern appeared in our image or not?

Logistic Regression Revisited

That is, find **f** such that

$$f\left(\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}\right) = \begin{cases} \checkmark & \text{if } \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \\ c_2 & c_1 \end{bmatrix} = \blacksquare\square \\ \times & \text{otherwise} \end{cases}$$

We can define: ✔ = 1 and ✗ = 0

- Yes it's a diagonal OR No it's not a diagonal

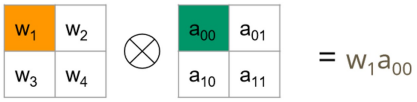
- Assign weights to each cell

Logistic Regression Revisited

We can assign weights to each cell

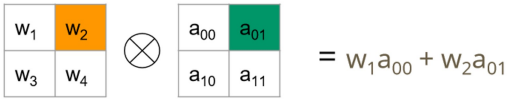
w_1	w_2
w_3	w_4

Logistic Regression Revisited

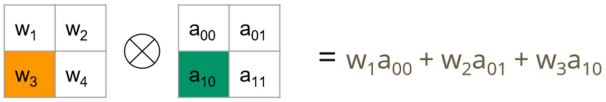


- Multiple by the weight to get linear combination

Logistic Regression Revisited

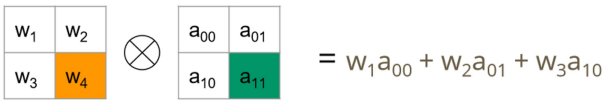


Logistic Regression Revisited



$= w_1a_{00} + w_2a_{01} + w_3a_{10}$

Logistic Regression Revisited



$= w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11}$

Logistic Regression Revisited

We can assign weights to each cell

w_1	w_2
w_3	w_4

such that:

$$\underbrace{w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11}} = b \quad \text{if diagonal pattern found}$$

w_1	w_2
w_3	w_4

$\otimes$

a_{00}	a_{01}
a_{10}	a_{11}

- Get a particular value that lets us know that we have a diagonal pattern

Logistic Regression Revisited

For example:

w_1	w_2
w_3	w_4

=

-2	2
2	-2

What value b do we get when applied to the diagonal pattern?

-2	2
2	-2

$\otimes$

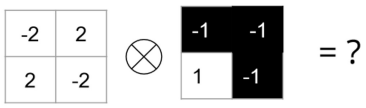
-1	1
1	-1

= ?

- = 2 + 2 +2 + 2 = 8

Logistic Regression Revisited

Any other pattern will have a value lower:



- 2 -2 + 2 +2 = 4
- Any other pattern than this diagonal will have less than 8
- With these weights, we will only get 8 if diagonal

Logistic Regression Revisited

Equivalently we can decide to move the value b to the left of the equation in order for the weighted sum to reveal a diagonal pattern at 0:

$$w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b = 0 \quad \text{if diagonal pattern found}$$

We could then find a function σ to apply to the result of this sum in order to make predictions {0, 1}:

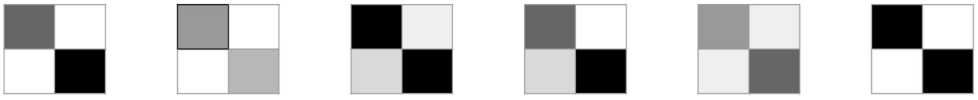
$$\sigma(w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b) = 1 \text{ if } w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b = 0 \text{ else } 0$$

- +b vs -b doesn't really matter when coming up with this initial function
  - (Subtracted both sides with b, but we will still find our answer if we call it +b)

Logistic Regression Revisited

Suppose we relax our definition of diagonal by having a continuum of colors $[c_1, c_2]$. This means there will be a continuum of values for our weighted sum to take when a diagonal pattern is found:

$$w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b > 0 \text{ if diagonal}$$



Logistic Regression Revisited

Suppose we relax our definition of diagonal by having a continuum of colors $[c_1, c_2]$. This means there will be a continuum of values for our weighted sum to take when a diagonal pattern is found:

$$w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b > 0 \text{ if diagonal}$$

We would like our function to adapt to this vagueness of specification / definition by reflecting an uncertainty in prediction (i.e. predicting probabilities of being diagonal)

$$\sigma(w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b) > 0.5 \text{ then diagonal}$$

- If this weighted sum is >0 its diagonal …

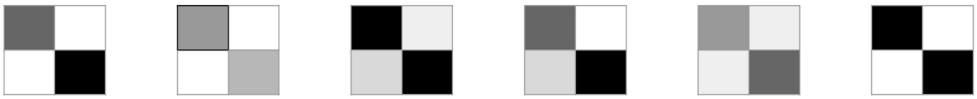
- Some examples will be close to 0 because we aren't sure if it's a diagonal
- We want the vagueness of specification to be/is reflective of the uncertainty of prediction

Logistic Regression Revisited

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

When **σ** is the logistic (also called sigmoid) function, this is Logistic Regression.

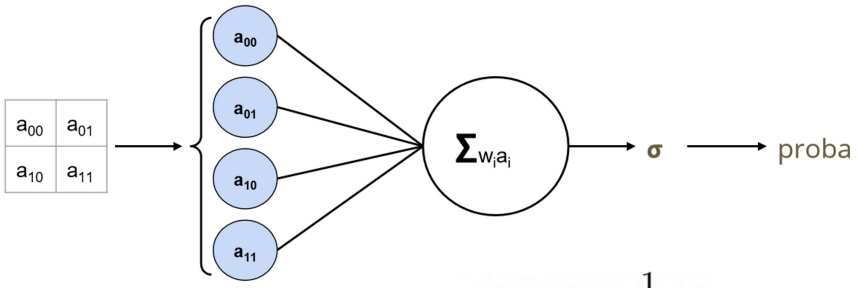
So for each cell we're looking to learn a weight w_i that makes **σ** larger for diagonal patterns. The bias term b lets us account for systemic dimming or brightening of cells (i.e. when the data is not normalized).



.92 .81 .95 .73 .68 .99

- Resembles logistic regression
- Logistic regression is a good option for detecting diagonals in two by two grid

Logistic Regression Revisited



where $\sigma(x) = \frac{1}{1+e^{-x}}$

- Input: two by two image
- Break down into components
- Take weighted sum of the values
- Run through sigma (our logistic function)
- Then output probabilities that something is in a particular class

Logistic Regression Revisited

Recall that logistic regression is looking for weights and a bias that maximizes the probability of having seen the data we saw:

$$\max \prod_{i=1}^{n} P(y_i = 1|x_i)$$

$$= \min -\frac{1}{n} \sum_{i=1}^{n} \left[yi \log(\sigma(-w^T x_i + b)) + (1 - y_i) \log(1 - \sigma(-w^T x_i + b)) \right]$$

$$= \min \text{Cost}(w, b)$$

- How do you learn w & b?
- We want to maximize the probability of seeing the data that we saw
- If yi is 0 or 1 one term drops out
- We want to improve our function in an iterative way
- Don't worry too much of how it's derived
  - You can try to do the math
  - The maximization function for linear regression is in those slides
- Binary cross entropy loss
  - Cost function

Gradient Descent (intuition)

There is no closed form solution to finding the extrema of this cost function. We can however use an iterative process by which we increment w and b gradually toward some minimum (most likely local).

Goal: find a sequence of w_i's (and b's) that converge toward a minimum.

- Ignore b for now, just focus on the bias term
- We want to improve the cost
  - At some point we hope to reach an extreme of the function (min/max)

Gradient Descent (intuition)

Consider a random weight w_0. What happens to Cost(w_0) as you nudge w_0 slightly?

- Start with random weight w
- Nudge w in either directory
  - Will either improve or decrease the cost
  - How much will depend in what direction you nudge w in

Gradient Descent (intuition)

As such we can define the following sequence:

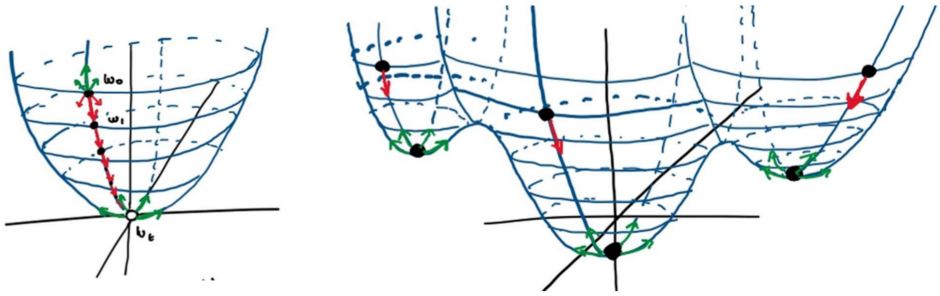
w_1 = best nudge to w_0
w_2 = best nudge to w_1
...

Until we reach w_t that looks like this:

At this point we can stop updating w. Why?

- Eventually we want to reach this point where we can't decrease cost further
  - No further improvements possible

Gradient Descent (intuition)



- If we go up, we are increasing the cost
- Then we find the local minimum
  - There will be multiple local minimums

Gradient Descent (intuition)

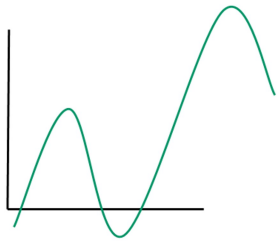
How can we know how much to nudge and in what direction?

Gradients

Intuitively the best nudge should be in the direction of the largest rate of change (steepness) of the function.

Rate of change -> think derivatives

$$\nabla f(x) = f'(x)$$



- Nudge in direction of largest increase/steepest

- Rate of change/derivatives

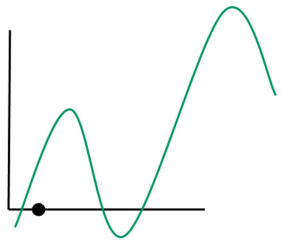
-

Gradients

Intuitively the best nudge should be in the direction of the largest rate of change (steepness) of the function.

Rate of change -> think derivatives

$$\nabla f(x) = f'(x)$$

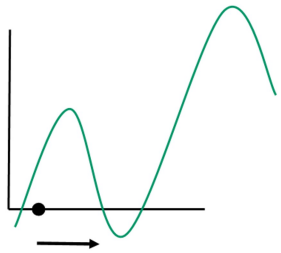


Gradients

Intuitively the best nudge should be in the direction of the largest rate of change (steepness) of the function.

Rate of change -> think derivatives

$$\nabla f(x) = f'(x)$$

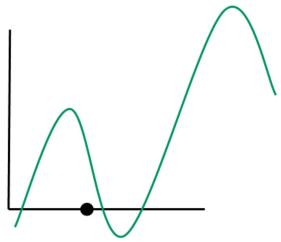


Gradients

Intuitively the best nudge should be in the direction of the largest rate of change (steepness) of the function.

Rate of change -> think derivatives

$$\nabla f(x) = f'(x)$$

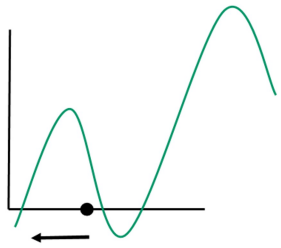


Gradients

Intuitively the best nudge should be in the direction of the largest rate of change (steepness) of the function.

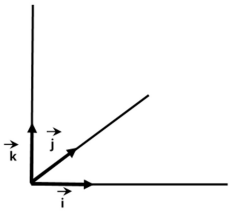
Rate of change -> think derivatives

$$\nabla f(x) = f'(x)$$



Gradients

Intuitively, the rate of change of a multi-dimensional function should be a combination of the rate change in each dimension. For a 3-dimensional function, the rate of change would be:



$$\nabla f(x,y,z) = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j} + \frac{\partial f}{\partial z}\vec{k}$$

- Function of multiple variables will be a function of the

maximum rate of changes for each parameter
  - (Maybe…)
- Want a combination of all of this

Gradients

Example:

$$f(x) = 3x^2\text{-}2y$$

Without even computing derivatives we can see that changes in x create more positive change in f than changes in y.

$$\nabla f = 6xi - 2j$$

This is the gradient of f and can be evaluated at any point (x, y) in the space.

- No matter what we do to x, we cant decrease f(x)
- Evaluate this gradient at any point x, y in the space
- Evaluate new point

Gradients

$$f(x) = 3x^2\text{-}2y \quad , \quad \nabla f = 6xi - 2j$$

Evaluating ∇f at p=(0, 0):

$$\nabla f_p = 6 \cdot 0 \cdot i - 2j = -2j$$

What happens to f as we move 1 unit away from p in the direction of the gradient?

$$p_{new} = 1 \cdot \nabla f_p + p = (0, -2)$$

$$f(p_{new}) = 3 \cdot 0^2 - 2 \cdot (-2) = 4 > f(p) = 0$$

Gradients

$$f(x) = 3x^2\text{-}2y \quad , \quad \nabla f = 6xi - 2j$$

What happens to f if we move 1 unit away from p in a random direction (not following ∇f)? Say (1,0) = 1i + 0j:

$$p_{new} = 1 \cdot (1,0) + p = (1, 0)$$

$$f(p_{new}) = 3 < 4$$

Moving p along the gradient will result in the fastest increase in f from p.

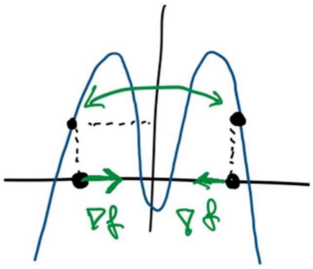
- Sanity check by going in the wrong direction?

- Moving along the gradient will get you the steepest improvement
- BUT if you take too big a step, you may make it worse

Gradients

However, the gradient expresses the **instantaneous** rate of change. At p, ∇f_p is the steepest but the highest value of f will depend on how many units we step in that direction. If we step too many units away, the instantaneous change in f is no longer representative of what values f will take.

Example:



- Worst case: keep bouncing back and forth
  - Too big a step size and you may overshoot the local max/min
- Tuning parameter we might have to deal with

Gradient Descent

Given a "smooth" function f for which there exists no closed form solution for finding its **maximum**, we can find a local maximum through the following steps:

1. Define a step size α (tuning parameter)
2. Initialize p to be random
3. $p_{new} = \alpha \nabla f_p + p$
4. $p \leftarrow p_{new}$
5. Repeat 3 & 4 until $p \sim p_{new}$

To find a local **minimum**, just use $-\nabla f_p$

- Update in direction of the gradient
- Until p is approximately p_new
- Steepest increase is always the opposite direction of the steepest descent
  - ?
- What happens if you want to update b and w?
  - Take partial with respect to both

Gradient Descent

Notes about α:

- If α is too large, GD may overshoot the maximum, take a long time to or never be able to converge
- If α is too small, GD may take too long to converge

-

Gradient Descent

Let's apply this to our diagonal problem to find the weights and bias for logistic regression.

Assume we have the following dataset:

$[0\ 1\ 1\ 0]^T$

- We will update both w and b in this example

- Example:
  - Dataset of one image

Gradient Descent

Recall:

$$\text{Cost}(w, b)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[yi\log(\sigma(-w^T x_i + b)) + (1 - y_i)\log(1 - \sigma(-w^T x_i + b))\right]$$

Gradient Descent

We need to compute ∇Cost(w, b):

$$\nabla \text{Cost}(w, b) = \left[\frac{\partial}{\partial w}\text{Cost}, \frac{\partial}{\partial b}\text{Cost}\right]$$

$$\frac{\partial}{\partial w}\text{Cost} = \frac{1}{n}\sum_{i=1}^{n} x_i(y_i - \sigma(-w^T x_i + b))$$

$$\frac{\partial}{\partial b}\text{Cost} = \frac{1}{n}\sum_{i=1}^{n} \sigma(-w^T x_i + b) - y_i$$

- Take partial derivates with respect to w and b
- Sigma is our sigmoid/logistic function
- Took the log-transform because the computation is easier than the early sum function

Gradient Descent

 = $[0\ 1\ 1\ 0]^T$

1. Start with random w and b:

$w = [0\ 0\ 0\ 0]^T$, $b = 0$

Note: $\sigma(0) = 0.5$

- We are initializing with no information
- Compute the cost (for illustrative purposes?)

Gradient Descent

2. Compute the Cost(w, b)

$Cost([0\ 0\ 0\ 0]^T, 0) = -1 \log(\sigma(0)) = -\log(0.5)$

- Re-using earlier partial derivatives

Gradient Descent

 = $[0\ 1\ 1\ 0]^T$

3. Compute the gradient ∇Cost at (w, b)

$$\frac{\partial}{\partial w}\text{Cost} = \frac{1}{1}\sum_{i=1}^{1}\begin{bmatrix}0\\1\\1\\0\end{bmatrix}(1-\sigma(0)) = \begin{bmatrix}0\\1/2\\1/2\\0\end{bmatrix}$$

Recall we only have one data point

- Apply formula

Gradient Descent

 = $[0\ 1\ 1\ 0]^T$

3. Compute the gradient ∇Cost at (w, b)

$$\frac{\partial}{\partial b}\text{Cost} = \frac{1}{1}\sum_{i=1}^{1}(\sigma(0)-1) = -\frac{1}{2}$$

Recall we only have one data point

Gradient Descent

4. Adjust w & b by taking **α** steps in the direction of -∇Cost$_{(w, b)}$

$$w_{\text{new}} = -\alpha \begin{bmatrix} 0 \\ 1/2 \\ 1/2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -\alpha/2 \\ -\alpha/2 \\ 0 \end{bmatrix} \qquad b_{\text{new}} = \alpha\frac{1}{2} + 0 = \frac{\alpha}{2}$$

- Steps in the direction of the gradient

$-\log(b)$

$\sigma\left(\alpha + \frac{1}{2}\right) > \frac{1}{2}$

$\frac{1}{1 + e^{-\alpha - 1/2}} > \frac{1}{2}$

Gradient Descent

5. Compute the updated Cost

$$\text{Cost}\left(\begin{bmatrix} 0 \\ -\alpha/2 \\ -\alpha/2 \\ 0 \end{bmatrix}, \frac{\alpha}{2}\right) = -\log(\sigma(\alpha + \frac{1}{2}))$$

For what values of **α** is the Cost reduced?

- For what values of alpha increase cost?
- Alpha values that decrease cost?

Stochastic Gradient Descent

Recall the Cost is computed for the entire dataset. This has some limitations:

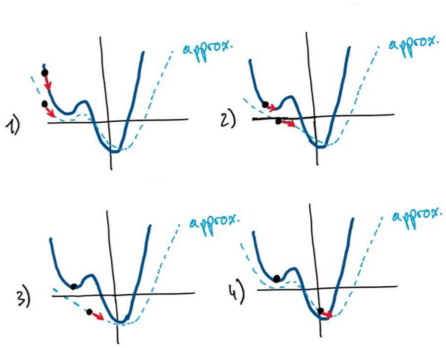
1. It's expensive to run
2. The result we get depends only on the initial starting point

---

- Not great because if we have really complex functions, will have lots of local minimum
- Really expensive to run this
  - Every time you update your weights, you would run a sum over entire dataset to computer cost

Stochastic Gradient Descent

Goal: Approximate the gradient of the Cost using a sample of the data (batch)



- Let's just take a little bit of our data and approximate the cost
- We will bypass some of the smaller local optimum
- Allows us to relax sensitivity to starting condition
    - Batch size & content of batch
- Allows to optimize by computing on a batch rather than the whole dataset

Note

The magnitude of ∇f_p depends on p. A p gets closer to the min / max, the size of ∇f_p decreases.

This also means that points p that contain more “information” have larger gradients. So the order with which this process is exposed to examples matters.

- The information contained in a point of your dataset,
- Size of change is proportional to the amount of information contained in the point?
- Essentially it for gradient descent
- This approximation may vary by batch
    - We are hoping that this will happen
- Introducing randomness by using this approximation
- Batch size will also be a tuning parameteerr

20211101 part 3 Neural Networks

Monday, November 1, 2021 5:31 PM

- Looking for contributor for neural network learning materials
  - kviz folder in the galletilance in git
  - No documentation so the learning curve will be steep?
- Will mention other opportunities for students to get involved in his research (paid)

==========

NEURAL NETWORKS

- Function sigma is our sigmoid function
- XOR function
  - Logistic regression was bad at performing this task
    - Can't separate by a line
- OR function IS linearly separable
- XOR can be written as a function of OR and AND functions
  - Through feature engineering we can solve this problem
- AND function is also linearly separable
- We may be able to learn h1 and h2 automatically through logistic regression
- Putting it together:
  - We get a network of models
  - Lines connecting nodes are the weights associated

with the linear sum

- Layers
  - Input layer
  - Hidden layer
    - Makes features
  - Output layer
- Now we have a neural network
- We need to define:
  - How does input flow through this network
  - How do the weights and biases get updated?
- This network is the model of 3 models working in conjunction (taking in inputs, making outputs to the next node/result)
- Forward propagation
  - The superscript just specifies the layer the weight is coming from
- Matrix notation:
  - Matrix of weights * X vector + bias
  - Make sure you understand the matrix notation
    - Keeping track of the dimensions matters
  - Second layer only has the bias?
    - There are some error prone slides in this presentation
- If all weights were 0 and all biases were 0:
  - Output probability would be: sigma of 0 => 0.5
  - 1 /(1+e^-1/2)?