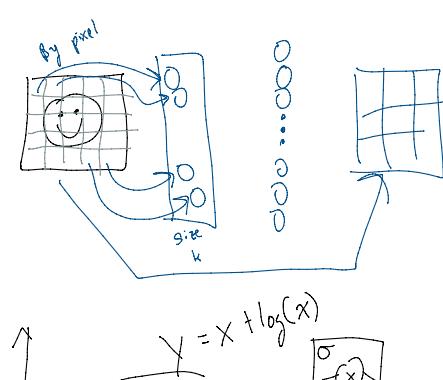
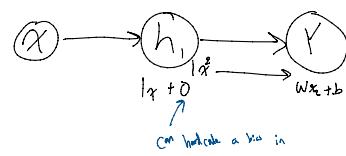
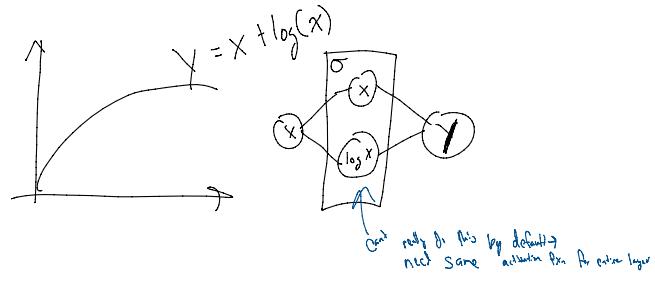


- Critical Algorithm studies recording
- Most confusing was SVM and neural networks
- Neural network notes he's posted on medium.com
  - o There are more links available now
  - o Introduced Neural Networks from logistic regression view
    - Gradient descent
      - Stochastic gradient descent
      - Backpropagating
    - Sigmoid, some kind of probability associated with gradient descent
    - XOR
      - Could learn  $h_1$  and  $h_2$  automatically
        - ◆ Specific function of  $x_1$  and  $x_2$
    - This process in the neural network
      - Goal is to learn features automatically while learning a particular model
      - Capture variations in  $Y$  using those features
      - In practice, you may not learn meaningful features
  - o We can also ask NN to learn features to do a regression function
    - We can ask it to learn something like  $X^2$  on the fly
      - As the activation function
      - Next layer will just be a linear combination
      - Can encode this as a neural network
      - Cheated by knowing something about the data and picking  $X^2$  based on that
  - o If we don't have any intuition about the data...?
    - What if  $Y = X^3$ ?
      - $X^2$  wouldn't model it well
      - To some extent we are defeating the purpose of neural networks
  - o ReLU
    - Approximating  $Y$  with a piecewise linear function
    - The more pieces you have, the better approximation you have
    - At best you get 4 separate pieces from 4? neurons
    - Approximating  $Y=X^2$  with a bunch of linear functions
      - Don't need intuition about the data
      - Does it automatically
      - Still get a good result
  - o Both classification and regression can be done with NN
  - o Rank K approximation
    - Minimized Frobenius? norm/distance
    - Autoencoder
  - Read articles on tuning Neural networks
    - o How do you initialize your weight, how to initialize the bias, ...
  - Activation is for the entire layer



- How do you initialize your weights, how to initialize the bias, ...
- Activation is for the entire layer
- There are things that are relatively unlearnable
- We care about:
  - o Are you learning the right thing
  - o Is it doing it well
  - o Is it useful
- Medical fields:
  - o ID things really quickly using NN
    - But if it's a black box, it doesn't really add knowledge
    - Working on how to extract the knowledge out of the black box
- There is a lot of code available on his git



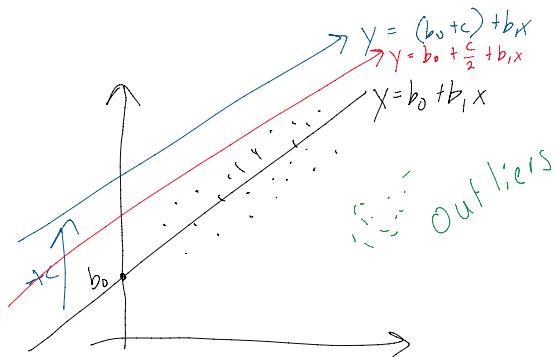
- Python 3 linear\_regression.py vs neural\_network
  - NN
    - o Custom activation function
    - o Play around with this
      - Number of neurons
      - Different activation functions
      - Try different things, challenge yourself, see what works and doesn't work

Coming up:

- HW3
  - Slides are posted
- Early insight presentation
- If you want to attend panelists for both the earlier session and our session
  - Ask to make connections
- A lot of the needed code for HW2 question ... is in the github repo
  - Look at SVD on the image
  - Look at single principal component
  - Approximate the image using a single Principal component
    - o But if you did 1, then 1 and 2, then 1, 2, 3 that is fine

## - HW2

- For the first question, there are some points you are supposed to plot
- Use logistic and linear regression to separate points



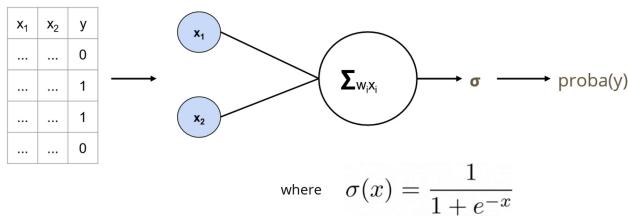
## HW2 Question 1:

- If you don't generate the points this way, then you won't get a logical answer that will give us a line
  - So just making one or two blobs is not sufficient
    - They won't take away "that many points"
- Fit the line on all the points, ignoring the label
- He does have office hours tonight, email with a few suggested times if needed
- Next Monday: no lecture during regular time. Can either attend live or watch the recording Monday
  - A1 class: 12:20pm???? on Monday

## Neural Networks

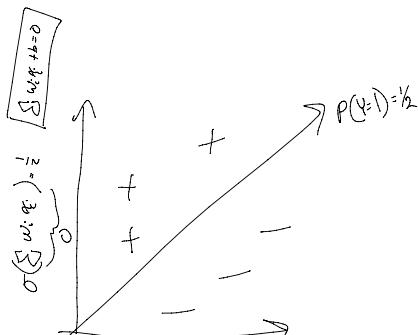
Boston University CS 506 - Lance Galletti

### Logistic Regression Re-Revisited



- With logistic regression, we can split this up into a number of steps
- Sigmoid/logistic function

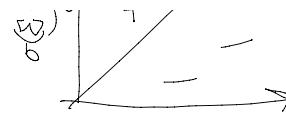
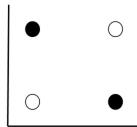
### Logistic Regression Re-Revisited



## Logistic Regression Re-Revisited

XOR function

$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	0

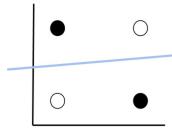


- Why is decision boundary in logistic regression a line?
- Can't split up XOR function with single line

## Logistic Regression Re-Revisited

XOR function

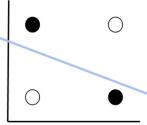
$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	0



## Logistic Regression Re-Revisited

XOR function

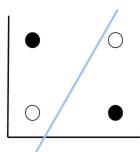
$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	0



## Logistic Regression Re-Revisited

XOR function

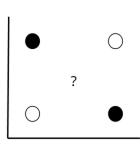
$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	0



## Logistic Regression Re-Revisited

XOR function

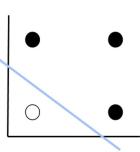
$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	0



## Logistic Regression Re-Revisited

Recall, the OR function is linearly separable:

$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	1



## Logistic Regression Re-Revisited

$$\text{XOR}(x_1, x_2) = \text{OR}(\text{AND}(x_1 = 0, x_2 = 1), \text{AND}(x_1 = 1, x_2 = 0))$$

$$= (x_1 = 0 \wedge x_2 = 1) \vee (x_1 = 1 \wedge x_2 = 0)$$

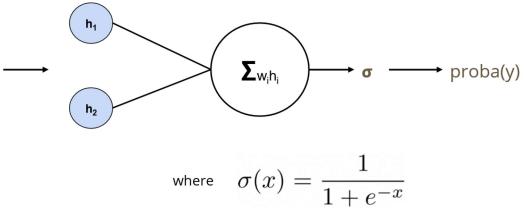
$$= h_1 \vee h_2$$

$x_1$	$x_2$	$h_1$	$h_2$	$y$
0	0	0	0	0
1	0	0	1	1
0	1	1	0	1
1	1	0	0	0

- Want to automatically learn our features to separate our classes

## Logistic Regression Re-Revisited

$h_1$	$h_2$	$y$
0	0	0
0	1	1
1	0	1
0	0	0

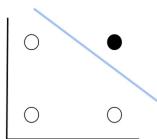


$$\text{where } \sigma(x) = \frac{1}{1 + e^{-x}}$$

## Logistic Regression Re-Revisited

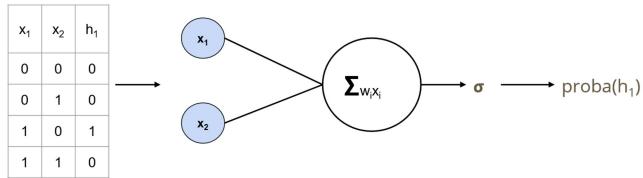
But, the **AND** function is also linearly separable:

$x_1$	$x_2$	$y$
0	0	0
1	0	0
0	1	0
1	1	1



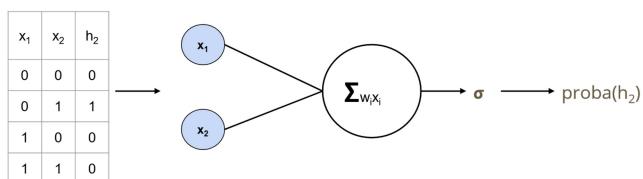
## Logistic Regression Re-Revisited

Since we can learn  $h_1$  and  $h_2$  automatically through logistic regression



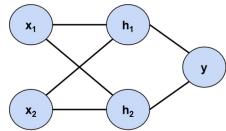
## Logistic Regression Re-Revisited

Since we can learn  $h_1$  and  $h_2$  automatically through logistic regression



## Neural Networks

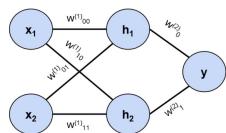
Putting it all together:



---

## Neural Networks

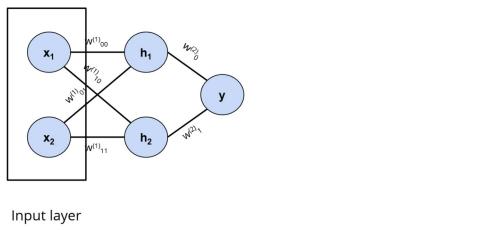
Putting it all together:



---

## Neural Networks

Putting it all together:

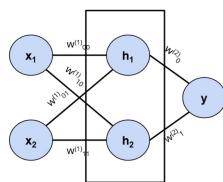


Input layer

---

## Neural Networks

Putting it all together:

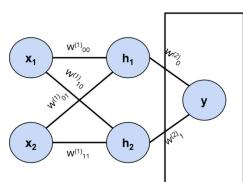


Hidden layer

---

## Neural Networks

Putting it all together:



Output layer

---

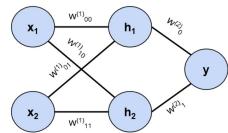
## Neural Networks

We need to define:

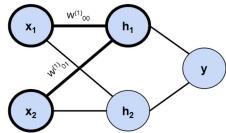
1. How input flows through the network to get the output (forward propagation)
  2. How the weights and biases gets updated (Backpropagation)
- 

- How do we feed information to the model?
- How does information flow through the model?
- How do you update the weights?

## Neural Networks - Forward Propagation



## Neural Networks - Forward Propagation

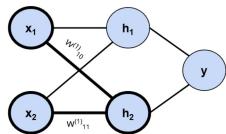


$$h_1 = \sigma(w^{(1)}_{00} x_1 + w^{(1)}_{01} x_2 + b^{(1)}_1)$$

---

- Weighted sum
- Activate using the sigmoid function
- 6 weights, 3 biases, a lot to keep track of

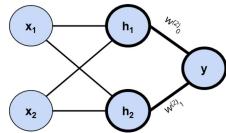
## Neural Networks - Forward Propagation



$$h_2 = \sigma(w^{(1)}_{10} x_1 + w^{(1)}_{11} x_2 + b^{(1)}_2)$$

---

## Neural Networks - Forward Propagation



$$y = \sigma(w^{(2)}_0 h_1 + w^{(2)}_1 h_2 + b^{(2)}_1)$$

---

## Neural Networks - Forward Propagation

Using matrix notation:

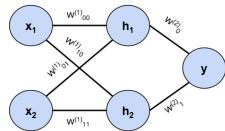
$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \sigma \left( \begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} \\ w_{10}^{(1)} & w_{11}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \right)$$
$$y = \sigma \left( \begin{bmatrix} w_{00}^{(2)} \\ w_{01}^{(2)} \end{bmatrix}^T \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + b^{(2)} \right)$$

---

- It gets very cumbersome to track all the weights, so just having a weight matrix for that layer is easier

## Neural Networks - Forward Propagation

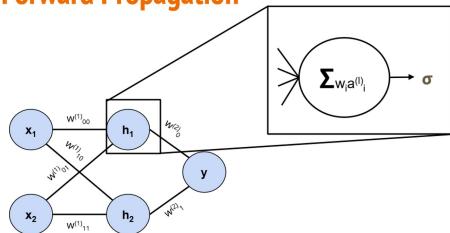
Q: if all the weights and biases are initialized to 0, what will be the output of the network?



- 
- What happens all weights are 0?
    - o Then we get 1/2 as the prediction for the model

## Neural Networks - Forward Propagation

Q: why have  $\sigma$  at all?



- 
- Why do we have sigma at all?
    - o Then we are basically just doing logistic regression on X1 and X2 again
      - With different weights
    - o It is usually important to introduce some sort of function after this weighted sum, in order to make sure you aren't just doing logistic regression again
    - o Sigma: activation function

## Neural Networks - Forward Propagation

If we don't, we just end up with normal logistic regression on  $x_1$  and  $x_2$ .

$$h_1 = w^{(1)}_{00} x_1 + w^{(1)}_{01} x_2 + b^{(1)}_1$$

$$h_2 = w^{(1)}_{10} x_1 + w^{(1)}_{11} x_2 + b^{(1)}_2$$

Then

$$y = \sigma(w^{(2)}_0 h_1 + w^{(2)}_1 h_2 + b^{(2)}_1)$$

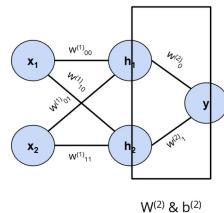
$$= \sigma(w^{(2)}_0(w^{(1)}_{00} x_1 + w^{(1)}_{01} x_2 + b^{(1)}_1) + w^{(2)}_1(w^{(1)}_{10} x_1 + w^{(1)}_{11} x_2 + b^{(1)}_2) + b^{(2)}_1)$$

$$= \sigma(w_1 x_1 + w_2 x_2 + b_2)$$

---

## Neural Networks - BackPropagation

How do weights and biases get updated?



$W^{(2)} & b^{(2)}$

---

## Neural Networks - BackPropagation

Using the chain rule:

$$\begin{aligned}\frac{\partial C}{\partial W^{(2)}} &= \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial W^{(2)}} \quad \text{where } u^{(2)} = W^{(2)}h + b^{(2)} \\ &= \frac{\partial C}{\partial u^{(2)}} \cdot h = \frac{1}{n} \sum_{i=1}^n h(y_i - \sigma(u^{(2)}))\end{aligned}$$

$\uparrow$   
 $h = \sigma(W^{(1)}X + b^{(1)})$

---

- Cost function C
  - o Look at gradient descent slides
- The math isn't as important as why this algorithm works
- Weighted sum of the h's
- Partial derivatives of cost with respect to u, and the partial derivative of u with respect to W

## Neural Networks - BackPropagation

Similarly:

$$\frac{\partial C}{\partial b^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial b^{(2)}} = \frac{1}{n} \sum_{i=1}^n y_i - \sigma(u^{(2)})$$

---

## Neural Networks - BackPropagation

So we can update  $W^{(2)}$  and  $b^{(2)}$  as follows:

$$\begin{bmatrix} W_{new}^{(2)} \\ b_{new}^{(2)} \end{bmatrix} = -\alpha \begin{bmatrix} \frac{\partial C}{\partial W^{(2)}} \\ \frac{\partial C}{\partial b^{(2)}} \end{bmatrix} + \begin{bmatrix} W^{(2)} \\ b^{(2)} \end{bmatrix}$$

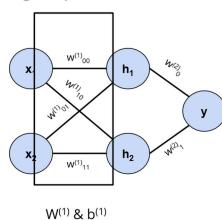
But how do we update  $W^{(1)}$  and  $b^{(1)}$

---

- Similarly we can do the same for  $b$
- So far this is just gradient descent, logistic regression model...
  - o with different variables

## Neural Networks - BackPropagation

How do weights and biases get updated?



---

$W^{(1)} & b^{(1)}$

## Neural Networks - BackPropagation

Using the chain rule:

$$\begin{aligned}\frac{\partial C}{\partial W^{(1)}} &= \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} \quad \text{where } u^{(1)} = w^{(1)}x + b^{(1)} \\ &= \frac{\partial C}{\partial u^{(2)}} \cdot \frac{\partial u^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)}) \cdot x \\ &\quad \uparrow \\ &\quad \text{Already computed}\end{aligned}$$

---

- Need to apply the chain rule a couple of times
- Can update these using gradient descent

## Neural Networks - BackPropagation

Similarly:

$$\frac{\partial C}{\partial b^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)})$$

$\uparrow$   
Already computed

---

## Neural Networks - BackPropagation

Backpropagation: update  $W^{(1)}$  and  $b^{(1)}$  without recomputing values that are computed when getting the gradients of the previously updated layer.

<http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf>

---

- We could technically do gradient descent the way we know how to do it
- But there is a better way to do this so we aren't recomputing the terms
- Backpropagation
  - o Efficiently tracking values so you aren't recalculating them
  - o Backpropagation avoids recomputing values (unlike gradient descent)
- Recommended paper on efficient back propagation

## Neural Networks - BackPropagation

Important Note:

$$\begin{aligned}\frac{\partial C}{\partial W^{(1)}} &= \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} \quad \text{where } u^{(1)} = w^{(1)}x + b^{(1)} \\ &= \frac{\partial C}{\partial u^{(2)}} \cdot \frac{\partial u^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)}) \cdot x\end{aligned}$$

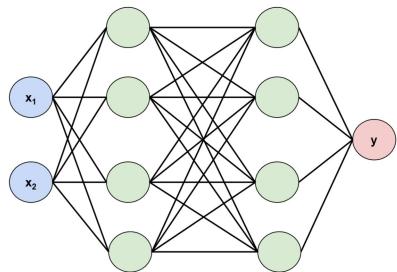
↑  
Depends on both data and weights  
Initializing all weights to zero then is not a good idea

---

- Depends on both the weights AND the data
  - o The weights you learn effects the ability to learn

## Feedforward Neural Networks

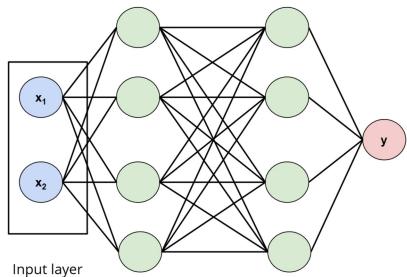
In general:



- 
- Can have any number of neurons within a layer
  - Can have any number of hidden layers
  - Can do multiclass outcome

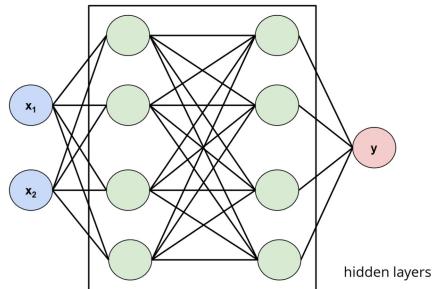
## Feedforward Neural Networks

In general:



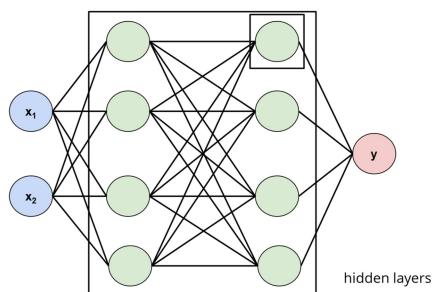
## Feedforward Neural Networks

In general:



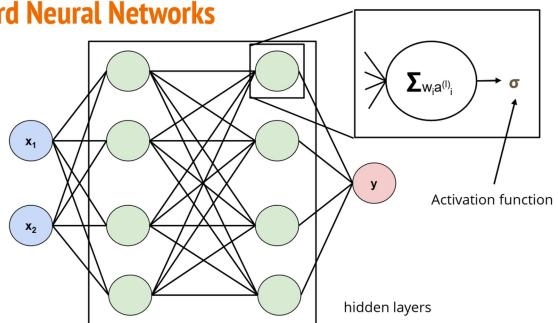
## Feedforward Neural Networks

In general:



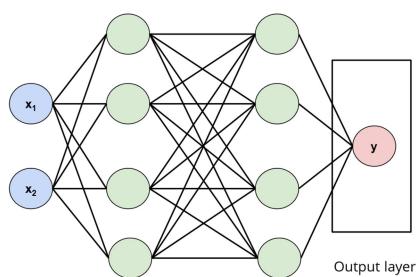
## Feedforward Neural Networks

In general:



## Feedforward Neural Networks

In general:



## Feedforward Neural Networks

The hope:

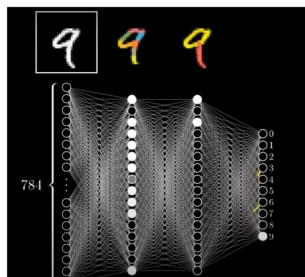


Image from 3b1b

- Hope you learn more abstract features as you go through the model

## Feedforward Neural Networks

The reality:



Figure 11: Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.

Image from "Why Should I Trust You?": Explaining the Predictions of Any Classifier (2016) Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin

	Before	After
Trusted the bad model	10 out of 27	3 out of 27
Snow as a potential feature	12 out of 27	25 out of 27

Table 2: "Husky vs Wolf" experiment results.

- Every wolf had a background of snow, and it learned that
- Has ability to learn, but may learn the wrong thing

## Feedforward Neural Networks

The scary reality:

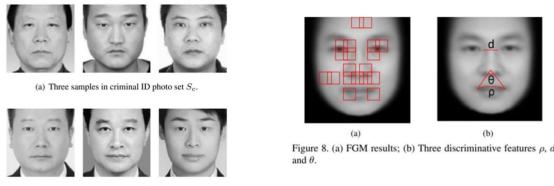


Figure 8. (a) FGM results; (b) Three discriminative features  $\rho$ ,  $d$  and  $\theta$ .

from "Automated Inference on  
Criminality using Face Images",  
Xiaolin Wu, Xi Zhang

According to this model, if you don't smile, you're a criminal

- Learned a smile detector
- Learning features automatically means a non-technical person can try to apply it
  - o But you want someone to have spent some time thinking about a model that's effecting your life

## Neural Networks

Can do both **Classification** and **Regression**

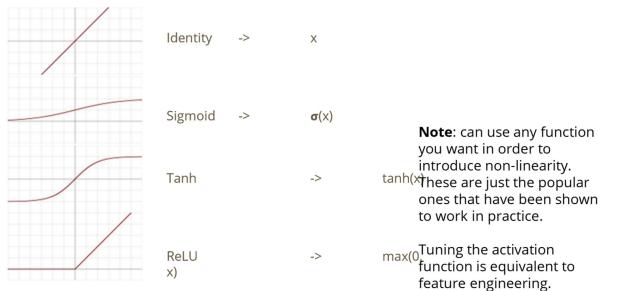
---

## Neural Networks - Tuning Parameters

1. Step size  $\alpha$
  2. Number of BackPropagation iterations
  3. Batch Size
  4. Number of hidden layers
  5. Size of each hidden layer
  6. Activation function used in each layer
  7. Cost function
  8. Regularization (to avoid overfitting)
- 

- Step size like in gradient descent
- Batch size
  - o Stochastic gradient descent
  - o ....
- Activation function
  - o You could use a variety of things
- Regularization to avoid overfitting

## Activation Functions



- Some people have done it so much they know what works but there isn't rules
  - o Black boxes we don't really understand
    - Why a model is predicting something wrong, especially in medicine, matters
    - Start has been made in trying to crack these open and understand
- The more you tune these activation functions, the more you are actually doing feature engineering
- It was complicated with 2 inputs, 2 hidden, and an output
  - o But you could have a large number of neurons and even more weights to keep track of

## Demo

- Check out the jupyter notebook in the Neural Network folder in git
- Can have any number of hidden layers
- Epoch is a term used in a lot of deep learning libraries
- May need to pip3 install pypt5?
- Would like to learn a circle decision boundary
- Learned new features, then linearly separate those new features
  - o If you project this back into the original space, it's no longer

- a line anymore
  - 10H vs sigmoid
  - May learn more with more epochs
  - Just because decision boundary is cosine, doesn't mean decision boundary activation should be cosine
    - o How can you use cosine with only one neuron?
      - You have to choose an activation function for the entire layer
      - That's kind of a limitation
  - What activations functions you should or shouldn't use there is a lot of rigidity in the field
    - o But you can do whatever you want and it may improve things
  - Singular value decomposition on an image
    - o You can do the same thing with a neural network
    - o Create a neural network on principal component extraction?
    - o Will use a cost function to compare the output to the input, and adjust weights
    - o Pretty similar to SVD
    - o Boat images again
    - o Can compare the Frobenius Distance between the rank 10 approximation in SVD to the neural network one
      - Was pretty close
    - o You can generate data like this, neural networks are good for more than classification
  - Code all posted on the repo, try this out on your own
  - Try a different number of layers, different number of neurons, etc.
  - It's bad both if we learn something wrong or it takes too long
    - o So optimize
  - There is some guidelines, but it's hard to know how to do this wrong
- 

- Normalizing your data is important
- What proportion of the data is responsible for creating that feature?
  - o Here you are learning features on parts of the dataset, unlike traditional analysis methods
  - o As data moves away from 0, you really quickly move to all or none of the data activate a neuron
    - This is really bad
    - This just turns this into a linear combination of the data
    - This isn't the case with just RLU for activation function
    - If you don't have normalized data, neurons can be frozen
    - If you don't normalize your data, your gradient descent algorithm often will not work

- <https://medium.com/mlearning-ai/tuning-neural-networks-part-i-normalize-your-data-6821a28b2cd8>
- If

## Neural Networks

First: Normalize your data

<https://medium.com/mlearning-ai/tuning-neural-networks-part-i-normalize-your-data-6821a28b2cd8>

---

## Neural Networks - Initialization Gotchas

<https://medium.com/mlearning-ai/tuning-neural-networks-part-ii-considerations-for-initialization-4f82e525da69>

---

## Neural Networks - Challenges

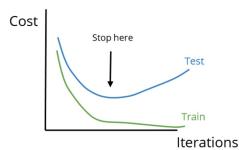
1. High risk of overfitting as you're optimizing on the training set.
  2. As the dimensionality of the input increases:
    - a. So does the number of weights
    - b. The gradients typically get smaller: Vanishing gradient problem
  3. Doesn't do well for computer vision where the object of detection can be anywhere in the image
  4. Doesn't handle sequences of inputs (i.e. providing context for data)
-

- We Can learn a lot, but high risk of overfitting
  - Don't want to memorize dataset
  - Some ways to check if this is happening
- This doesn't scale well
  - Needs much more compute power
- How do you handle sequences of input?
  - e.g. What is the next word that is about to be said, based on the words already spoken

## Neural Networks - Regularization

Two main ways:

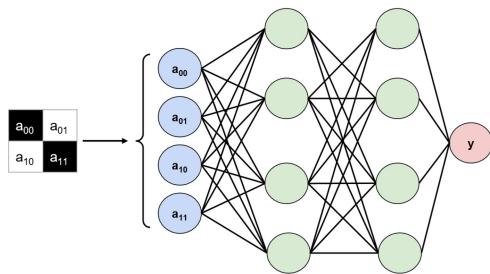
1. Early termination of weight / bias updates



1. Dropout - kill neurons (by setting them to 0) randomly
- 

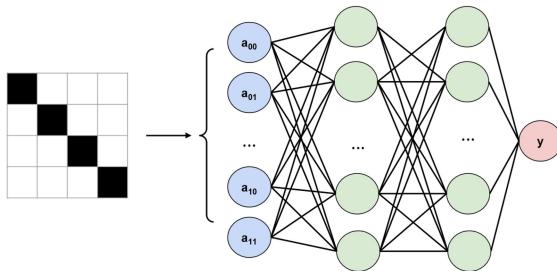
- Get better and better at learning your training set, but you would like to stop learning at the minimum for testing
  - Could do this by early termination
  - Or randomly set a neuron to 0
  - There are ways to do this with libraries

## Neural Networks - Convolutional Neural Networks



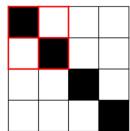
- 
- Specialized for things like computer vision
  - Trying to see the diagonal in the image

## Neural Networks - Convolutional Neural Networks



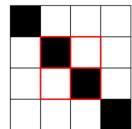
- But if we grow the diagonal, what happens?
  - Can we use what we learned before to investigate the larger diagonal (potentially by identifying multiple smaller diagonals)

## Neural Networks - Convolutional Neural Networks



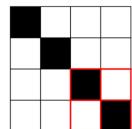
- We can find the 2x2 diagonal in the 4x4 diagonal

## Neural Networks - Convolutional Neural Networks



---

## Neural Networks - Convolutional Neural Networks



---

## Neural Networks - Convolutional Neural Networks

Recall: Our network learns weights for each cell

$w_1$	$w_2$
$w_3$	$w_4$

---

## Neural Networks - Convolutional Neural Networks

$$\begin{array}{c}
 \begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline \end{array} \otimes \begin{array}{|c|c|} \hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline \end{array} = w_1 a_{00} \\
 \end{array}$$

$$\begin{array}{c}
 \begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline \end{array} \otimes \begin{array}{|c|c|} \hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline \end{array} = w_1 a_{00} + w_2 a_{01} \\
 \end{array}$$

$$\begin{array}{c}
 \begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline \end{array} \otimes \begin{array}{|c|c|} \hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline \end{array} = w_1 a_{00} + w_2 a_{01} + w_3 a_{10} \\
 \end{array}$$

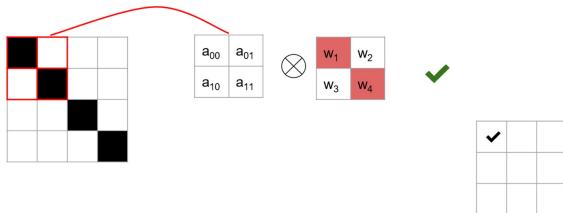
$$\begin{array}{c}
 \begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline \end{array} \otimes \begin{array}{|c|c|} \hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline \end{array} = w_1 a_{00} + w_2 a_{01} + w_3 a_{10} + w_4 a_{11}
 \end{array}$$


---

- We perform this operation to learn

## Neural Networks - Convolutional Neural Networks

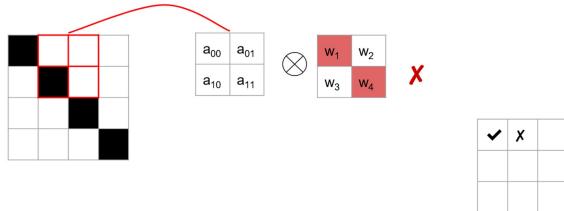
Knowing this, what happens if we slide this filter across the larger diagonal?



- 
- Keep track of where diagonals are found
  - Smaller version of the image is what we are filling out

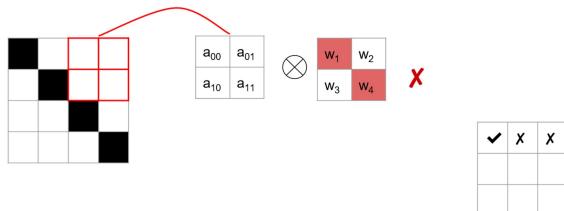
## Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



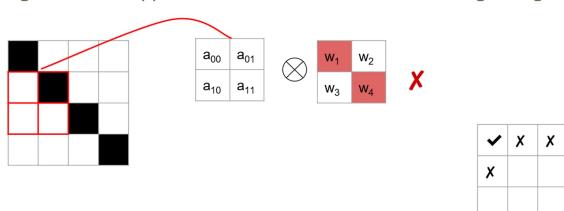
## Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



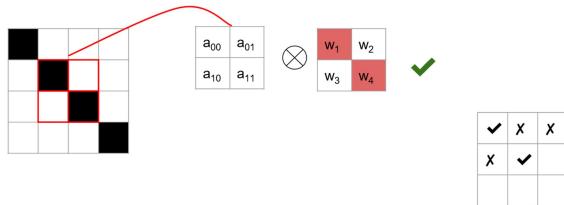
## Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



## Neural Networks - Convolutional Neural Networks

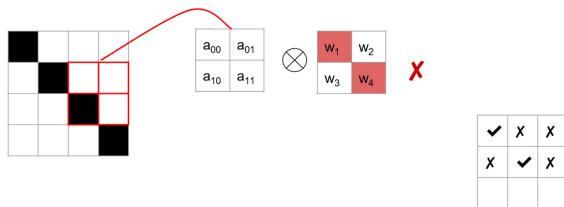
Knowing this, what happens if we slide this filter across the larger diagonal?



---

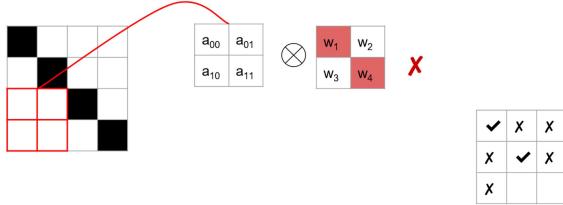
## Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



## Neural Networks - Convolutional Neural Networks

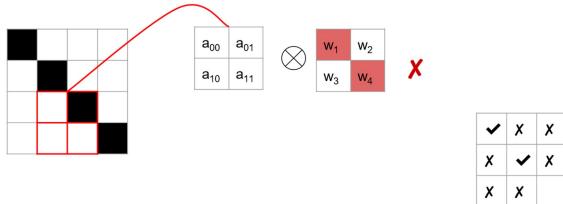
Knowing this, what happens if we slide this filter across the larger diagonal?



✓	X	X
X	✓	X
X		

## Neural Networks - Convolutional Neural Networks

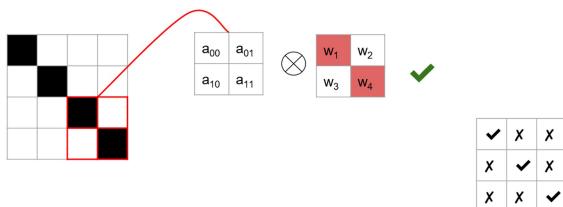
Knowing this, what happens if we slide this filter across the larger diagonal?



✓	X	X
X	✓	X
X	X	

## Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



✓	X	X
X	✓	X
X	X	✓

- Same weights as we were using before

## Neural Networks - Convolutional Neural Networks

Creating such a filter allows us to:

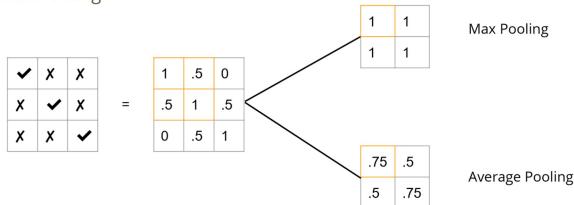
1. Reduce the number of weights
2. Capture features all over the image

The process of applying a filter (or kernel) is called a convolution

- 
- After convolution, there is a pooling step
    - So you have even less weights to keep track of

## Neural Networks - Convolutional Neural Networks

To reduce the weights even further, another phase is done after convolution called Pooling:



- 
- Many ways to pool
  - Max Pooling
    - Max value in a particular square
  -

## Neural Networks - Convolutional Neural Networks

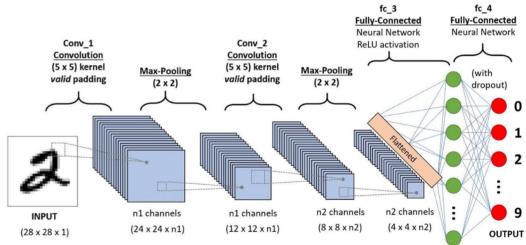


Image from <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

### - High level:

- We are going to learn smaller features, and as you get closer to the output the idea is that hopefully you are learning bigger more complicated features
- How do you do backpropagation in this?
  - It's complicated, and we won't talk about it
    - o Because you are losing some information

## Neural Networks - Convolutional Neural Networks

Main application: Computer vision

## Recurrent Neural Networks

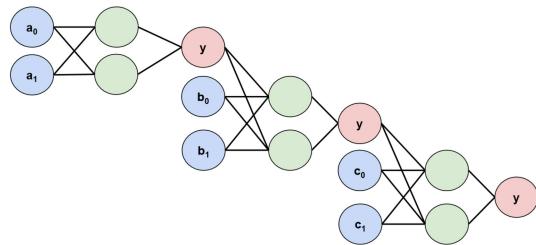
Handling sequences of input.

Intuition: What a word is / might be in a sentence is easier to figure out if you know the words around it.

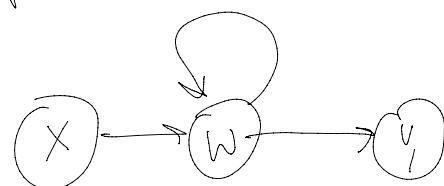
Applications:

1. Predicting the next word
  2. Translation
  3. Speech Recognition
  4. Video Tagging
- 

## Recurrent Neural Networks



This is how it's  
more typically drawn.



- Prefer this drawing to how this is typically drawn
- Back propagation through ...?
- The more words you take into account, the more things that might get lost through the layers
  - Like articles in Spanish?

```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\visualizing_neural_networks.ipynb
1 #%%
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from IPython.display import Image
5 import sklearn.datasets as datasets
6
7 from kviz.dense import DenseGraph
8 import tensorflow as tf
9 from tensorflow import keras, math, random, stack
10 from tensorflow.keras import layers, initializers
11 from tensorflow.keras.activations import relu
12
13 tf.compat.v1.logging.set_verbosity(tf.compat.v1.
    logging.ERROR)
14
15 ACTIVATION = 'sigmoid'
16 model = keras.models.Sequential()
17 model.add(layers.Dense(2, input_dim=2, activation=
    ACTIVATION))
18 model.add(layers.Dense(1, activation=ACTIVATION))
19 model.compile(loss="binary_crossentropy")
20
21 dg = DenseGraph(model)
22 dg.render('graph.png')
23 Image(filename='graph.png')
24 #%%
25 centers = [[0, 0]]
26 t, _ = datasets.make_blobs(n_samples=200, centers=
    centers, cluster_std=1,
27                             random_state=1)
28
29 # CIRCLE
30 def generate_circle_data(t, centers):
31     # create some space between the classes
32     X = np.array(list(filter(
33         lambda x : (x[0] - centers[0][0])**2 + (x[1]
34             - centers[0][1])**2 < 1
35         or (x[0] - centers[0][0])**2 + (x[1] - centers
36             [0][1])**2 > 1.5, t
37     )))
38     Y = np.array([1 if (x[0] - centers[0][0])**2 + (x[1] - centers[0][1])**2 >= 1 else 0 for x in X])
39
40 def generate_line_data(t):
41     # create some space between the classes
42     X = np.array(list(filter(lambda x : x[0] - x[1]
43         < -.5 or x[0] - x[1] > .5, t)))
44     Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x
45         in X])
46
47 def generate_curve_data(t):

```

Page 1 of 4

```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\visualizing_neural_networks.ipynb
36 1] - centers[0][1])**2 >= 1 else 0 for x in X])
37
38
39 # LINE
40 def generate_line_data(t):
41     # create some space between the classes
42     X = np.array(list(filter(lambda x : x[0] - x[1]
43         < -.5 or x[0] - x[1] > .5, t)))
44     Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x
45         in X])
46
47 def generate_curve_data(t):

```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\base\_model.py

```

1 import math as m
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import sklearn.datasets as datasets
5 from tensorflow import keras, math, random, stack
6 from tensorflow.keras import layers, initializers
7 from tensorflow.keras.activations import relu
8
9
10 #
11 #      x[0] --- h1
12 #              \ /   \
13 #               X       output
14 #              / \   /
15 #      v[1] --- h2

```

```

11 #      x1 --- n1
12 #          \ /     \
13 #              X      output
14 #              / \   /
15 #      x[1] --- h2
16 #
17 # This is the base model - nothing fancy here
18
19 # modify this line
20 ACTIVATION = "sigmoid"
21
22 # DONT MODIFY
23 # this is just for declaring
24 LABEL = None
25 THRESH = None
26 # Set random seed for reproducibility
27 np.random.seed(1)
28 random.set_seed(1)
29
30 if ACTIVATION=="tanh":
31     LABEL = -1
32     THRESH = 0
33 if ACTIVATION=="sigmoid":
34     LABEL = 0
35     THRESH = 0.5
36
37
38 # Can use a custom activation function
39 def custom_activation(x):
40     x_0 = math.cos(x[...,0])
41

```

Page 1 of 5

```

81 dg = DenseGraph(model)
82 dg.animate_learning(X, Y, filename='animate').

```

```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\base_model.py
42     return relu(x, threshold=0)
43
44 # 10 H seeming learned the decision boundary better
# than 10H
45
46 # Basically started with two neurons, which wasn't
# enough to approximate decision boundary
47 # Move into higher dimension and separate using plane
48
49 model = keras.models.Sequential()
50 # Can try different activation functions
51 # reduce number of epochs (iterations) to get this to
# run faster
52 model.add(layers.Dense(2, input_dim=2, activation=
    custom_activation))
53 model.add(layers.Dense(1, activation=ACTIVATION))
54 #opt = keras.optimizers.Adam(learning_rate=.01)
55 model.compile(loss="binary_crossentropy") #, optimizer
# =opt)
56
57 centers = [[0, 0]]
58 t, _ = datasets.make_blobs(n_samples=200, centers=
    centers, cluster_std=1,
    random_state=1)
59
60
61 # CIRCLE
62 def generate_circle_data(t):
63     # create some space between the classes
64     X = np.array(list(filter(lambda x : (x[0] -
        centers[0][0])**2 + (x[1] - centers[0][1])**2 < 1 or (
        x[0] - centers[0][0])**2 + (x[1] - centers[0][1])**2
        > 1.5, t)))
65     Y = np.array([1 if (x[0] - centers[0][0])**2 + (x[
        1] - centers[0][1])**2 >= 1 else LABEL for x in X])
66     return X, Y
67
68 # LINE
69 def generate_line_data(t):
70     # create some space between the classes
71     X = np.array(list(filter(lambda x : x[0] - x[1]
        ] < -.5 or x[0] - x[1] > .5, t)))

```

Page 2 of 5

```
 ] < - .5 or x[0] - x[1] > .5, t)))
```

Page 2 of 5

```
File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\base_model.py
72     Y = np.array([1 if x[0] - x[1] >= 0 else LABEL
73         for x in X])
74     return X, Y
75
75 # CURVE
76 def generate_curve_data(t):
77     # create some space between the classes
78     X = np.array(list(filter(lambda x : m.cos(4*x[0]
79         ) - x[1] < -.5 or m.cos(4*x[0]) - x[1] > .5, t)))
80     Y = np.array([1 if m.cos(4*x[0]) - x[1] >= 0 else
81         LABEL for x in X])
82     return X, Y
83
82 # XOR
83 def generate_xor_data():
84     X = np.array([
85         [0,0],
86         [0,1],
87         [1,0],
88         [1,1]])
89     Y = np.array([x[0]^x[1] for x in X])
90     return X, Y
91
92 X, Y = generate_circle_data(t)
93 # X, Y = generate_line_data(t)
94 # X, Y = generate_curve_data(t)
95 # X, Y = generate_xor_data()
96
97 colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
98 colors = np.hstack([colors] * 20)
99
100 plt.scatter(X[:,0],X[:,1],color=colors[Y].tolist(), s
101 =100, alpha=.9)
101 plt.show()
102
103 history = model.fit(X, Y, batch_size=50, epochs=1000)
104
105 # Show the transformation of the input at the first
105 hidden layer
106 layer = model.layers[0]
```

Page 3 of 5

```
File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\base_model.py
107 print(layer.get_config(), layer.get_weights())
108 keras_function = keras.backend.function([model.input
109 ], [layer.output])
110 layerVals = np.array(keras_function(X))[:,0]
111 plt.scatter(layerVals[:,0], layerVals[:, 1], color=
111 colors[Y].tolist(), s=100, alpha=.9)
111 plt.show()
112
113 # create a mesh to plot in
114 h = .02 # step size in the mesh
115 x_min, x_max = layerVals[:, 0].min() - .5, layerVals
115[:, 0].max() + 1
116 y_min, y_max = layerVals[:, 1].min() - .5, layerVals
116[:, 1].max() + 1
117 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
118                         np.arange(y_min, y_max, h))
119 meshData = np.c_[xx.ravel(), yy.ravel()]
120
121 # Plot the decision boundary. For that, we will
121 assign a color to each
122 # point in the mesh
123 fig, ax = plt.subplots()
124 layer = model.layers[-1]
125
```

```

    assign a color to each
122 # point in the mesh
123 fig, ax = plt.subplots()
124 layer = model.layers[-1]
125
126 intermediateModel = keras.models.Sequential()
127 intermediateModel.add(layers.Dense(1, input_dim=2,
activation=ACTIVATION))
128 intermediateModel.compile(loss="binary_crossentropy")
129 intermediateModel.layers[0].set_weights(layer.
get_weights())
130
131 Z = intermediateModel.predict(meshData)
132 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
133 Z = Z.reshape(xx.shape)
134 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
135
136 T = intermediateModel.predict(layerVals)
137 T = np.array([LABEL if x < THRESH else 1 for x in T])
138 T = T.reshape(layerVals[:, 0].shape)
139 ax.scatter(layerVals[:, 0], layerVals[:, 1], color=
colors[T].tolist(), s=100, alpha=.9)

```

Page 4 of 5

```

File : D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\base_model.py
140 ax.set_xlabel("h0")
141 ax.set_ylabel("h1")
142 plt.show()
143
144 # create a mesh to plot in
145 h = .02 # step size in the mesh
146 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
147 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
148 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
149                      np.arange(y_min, y_max, h))
150 meshData = np.c_[xx.ravel(), yy.ravel()]
151
152 fig, ax = plt.subplots()
153 Z = model.predict(meshData)
154 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
155 Z = Z.reshape(xx.shape)
156 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
157 ax.axis('off')
158
159 # Plot also the training points
160 T = model.predict(X)
161 T = np.array([LABEL if x < THRESH else 1 for x in T])
162 T = T.reshape(X[:,0].shape)
163 ax.scatter(X[:, 0], X[:, 1], color=colors[T].tolist
(), s=100, alpha=.9)
164 plt.show()
165

```

Page 5 of 5

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import sklearn.datasets as datasets
4 from tensorflow import keras
5 from tensorflow.keras import layers
6
7 # The model we want to learn is
8 #
9 #   if x[0]**2 + x[1]**2 >= 1 then 1 else 0
10 #
11 #           x[0] --- h11 --- h11
12 #                   \ /     \ /   \
13 #                   X       X      output
14 #                   / \     / \   /
15 #           x[1] --- h21 --- h21
16 #
17 # Try adding multiple layers or the same size - which
18 # number works best?
18
19 # modify this line
20 ACTIVATION = "sigmoid"
21
22 # DONT MODIFY
23 # this is just for declaring
24 LABEL = None
25 THRESH = None
26
27 if ACTIVATION=="tanh":
28     LABEL = -1
29     THRESH = 0
30 if ACTIVATION=="sigmoid":
31     LABEL = 0
32     THRESH = 0.5
33
34 model = keras.models.Sequential()
35 model.add(layers.Dense(2, input_dim=2))
36 model.add(layers.Dense(2))
37 model.add(layers.Dense(1, activation=ACTIVATION))
38 model.compile(loss="binary_crossentropy")
39
40 centers = [[0, 0]]
```

```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_layer.py
41 t, _ = datasets.make_blobs(n_samples=750, centers=
    centers, cluster_std=1,
42                                     random_state=0)
43
44 # create some space between the classes
45 X = np.array(list(filter(lambda x : x[0]**2 + x[1]**2
    < 1 or x[0]**2 + x[1]**2 > 1.5, t)))
46 Y = np.array([1 if x[0]**2 + x[1]**2 >= 1 else LABEL
    for x in X])
47
48 colors = np.array([x for x in '
    bgrcmykbgrcmykbgrcmykbgrcmyk'])
49 colors = np.hstack([colors] * 20)
50
51 plt.scatter(X[:,0],X[:,1],color=colors[Y].tolist(), s=
    10, alpha=0.8)
52 plt.show()
53
54 history = model.fit(X, Y, batch_size=50, epochs=100)
55
56 # Show the transformation of the input at the last
    hidden layer
57 layer = model.layers[9]
58 print(layer.get_config(), layer.get_weights())
59 keras_function = keras.backend.function([model.input
    ], [layer.output])
60 layerVals = np.array(keras_function(X))[0]
61 plt.scatter(layerVals[:, 0], layerVals[:, 1], color=
    colors[Y].tolist(), s=10, alpha=.8)
62 plt.show()
63
64 # create a mesh to plot in
65 h = .02 # step size in the mesh
66 x_min, x_max = layerVals[:, 0].min() - .5, layerVals
    [:, 0].max() + 1
67 y_min, y_max = layerVals[:, 1].min() - .5, layerVals
    [:, 1].max() + 1
68 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
69                      np.arange(y_min, y_max, h))
70 meshData = np.c_[xx.ravel(), yy.ravel()]
71

```

Page 2 of 4

```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_layer.py
72 # Plot the decision boundary. For that, we will
    assign a color to each
73 # point in the mesh
74 fig, ax = plt.subplots()
75 layer = model.layers[10]
76
77 intermediateModel = keras.models.Sequential()
78 intermediateModel.add(layers.Dense(1, input_dim=2,
    activation=ACTIVATION))
79 intermediateModel.compile(loss="binary_crossentropy")
80 intermediateModel.layers[0].set_weights(layer.
    get_weights())
81
82 Z = intermediateModel.predict(meshData)
83 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
84 Z = Z.reshape(xx.shape)
85 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
86 ax.axis('off')
87
88 T = intermediateModel.predict(layerVals)
89 T = np.array([LABEL if x < THRESH else 1 for x in T])
90 T = T.reshape(layerVals[:, 0].shape)
91 ax.scatter(layerVals[:, 0], layerVals[:, 1], color=
    colors[T].tolist(), s=1, alpha=0.9)
92 plt.show()
93
94 # create a mesh to plot in
95 h = .02 # step size in the mesh
96 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
97 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
98 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),

```

```
95 h = .02 # step size in the mesh
96 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
97 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
98 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
99                      np.arange(y_min, y_max, h))
100 meshData = np.c_[xx.ravel(), yy.ravel()]
101
102 fig, ax = plt.subplots()
103 Z = model.predict(meshData)
104 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
105 Z = Z.reshape(xx.shape)
106 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
107 ax.axis('off')
108
```

Page 3 of 4

```
File - D:\Users\lWolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_layer.py
109 # Plot also the training points
110 T = model.predict(X)
111 T = np.array([LABEL if x < THRESH else 1 for x in T])
112 T = T.reshape(X[:,0].shape)
113 ax.scatter(X[:, 0], X[:, 1], color=colors[T].tolist(),
114             (), s=1, alpha=0.9)
114 plt.show()
115
```

Page 4 of 4

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import sklearn.datasets as datasets
4 from tensorflow import keras
5 from tensorflow.keras import layers
6
7 # The model we want to learn is
8 #
9 #   if x[0]**2 + x[1]**2 >= 1 then 1 else 0
10 #
11 #           x[0] --- h1
12 #           |   \   \
13 #           \   X   \
14 #             \ /   \
15 #               X  h2 --- output
16 #               / \   /
17 #               /   X   /
18 #               |   \   /
19 #               x[1] --- h3
20 #
21 # Try adding more neurons for the given hidden layer
22
23 # modify this line
24 ACTIVATION = "sigmoid"
25
26 # DONT MODIFY
27 # this is just for declaring
28 LABEL = None
29 THRESH = None
30
31 if ACTIVATION=="tanh":
32     LABEL = -1
33     THRESH = 0
34 if ACTIVATION=="sigmoid":
35     LABEL = 0
36     THRESH = 0.5
37
38 model = keras.models.Sequential()
39 model.add(layers.Dense(3, input_dim=2, activation='relu'))
40 model.add(layers.Dense(1, activation=ACTIVATION))
```

```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_neuron.py
41 model.compile(loss="binary_crossentropy")
42
43 centers = [[0, 0]]
44 t, _ = datasets.make_blobs(n_samples=200, centers=
    centers, cluster_std=1,
45                               random_state=1)
46
47 # create some space between the classes
48 X = np.array(list(filter(lambda x : x[0]**2 + x[1]**2
    < 1 or x[0]**2 + x[1]**2 > 1.5, t)))
49 Y = np.array([1 if x[0]**2 + x[1]**2 >= 1 else LABEL
    for x in X])
50
51 colors = np.array([x for x in
    'bgrcmykbgrcmykbgrcmykbgrcmyk'])
52 colors = np.hstack([colors] * 20)
53
54 plt.scatter(X[:,0],X[:,1],color=colors[Y].tolist(), s=
    100, alpha=.9)
55 plt.show()
56
57 history = model.fit(X, Y, batch_size=50, epochs=250)
58
59 # Show the transformation of the input at the first
    hidden layer
60 layer = model.layers[0]
61 print(layer.get_config(), layer.get_weights())
62 keras_function = keras.backend.function([model.input
    ], [layer.output])
63 layerVals = np.array(keras_function(X))[0]
64 fig = plt.figure()
65 ax = fig.add_subplot(111, projection='3d')
66 ax.scatter(layerVals[:,0], layerVals[:, 1], layerVals
    [:, 2], color=colors[Y].tolist(), s=100, alpha=.9)
67 plt.show()
68
69 # create a mesh to plot in
70 h = .1 # step size in the mesh
71 x_min, x_max = layerVals[:, 0].min() - .5, layerVals
    [:, 0].max() + 1
72 y_min, y_max = layerVals[:, 1].min() - .5, layerVals

```

Page 2 of 4

```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_neuron.py
72[:, 1].max() + 1
73 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
    np.arange(y_min, y_max, h))
74 meshData = np.c_[xx.ravel(), yy.ravel(), np.zeros(len
    (xx.ravel()))]
75
76
77 # Plot the decision boundary. For that, we will
    assign a color to each
78 # point in the mesh
79 fig, ax = plt.subplots()
80 layer = model.layers[-1]
81
82 intermediateModel = keras.models.Sequential()
83 intermediateModel.add(layers.Dense(1, input_dim=3,
    activation=ACTIVATION))
84 intermediateModel.compile(loss="binary_crossentropy")
85 intermediateModel.layers[0].set_weights(layer.
    get_weights())
86
87 Z = intermediateModel.predict(meshData)
88 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
89 Z = Z.reshape(xx.shape)
90 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
    # plot in 2D
91 ax.axis('off')
92
93 T = intermediateModel.predict(layerVals)
94 T = np.array([LABEL if x < THRESH else 1 for x in T])
95 T = T.reshape(layerVals[:, 0].shape)
96 ax.scatter(layerVals[:, 0], layerVals[:, 1], color=
    colors[T].tolist(), s=100, alpha=.9) # plot in 2D

```

```

94 T = np.array([LABEL if x < THRESH else 1 for x in T])
95 T = T.reshape(layerVals[:, 0].shape)
96 ax.scatter(layerVals[:, 0], layerVals[:, 1], color=
    colors[T].tolist(), s=100, alpha=.9) # plot in 2D
97 plt.show()
98
99
100 # create a mesh to plot in
101 h = .1 # step size in the mesh
102 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
103 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
104 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
105                      np.arange(y_min, y_max, h))
106 meshData = np.c_[xx.ravel(), yy.ravel()]

```

---

Page 3 of 4

```
File - D:\Users\lWolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_neuron.py
107
108 # Plot the decision boundary. For that, we will
109 # assign a color to each
110 fig, ax = plt.subplots()
111 Zp = model.predict(meshData)
112 Z = np.array([np.array([LABEL]) if x < THRESH else np
113 .array([1]) for x in Zp])
114 Z = Z.reshape(xx.shape)
115 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
116 ax.axis('off')
117 # Plot also the training points
118 T = model.predict(X)
119 T = np.array([LABEL if x < THRESH else 1 for x in T])
120 T = T.reshape(X[:,0].shape)
121 ax.scatter(X[:, 0], X[:, 1], color=colors[T].tolist
122 (), s=100, alpha=.9)
123 plt.show()
```

---

Page 4 of 4

```

1 import numpy as np
2 import matplotlib.cm as cm
3 import matplotlib.pyplot as plt
4 from tensorflow import keras, norm
5 from tensorflow.keras import layers
6
7 # Principal Component Extraction using Neural Net
8 #
9 #
10 #      x[0]          x[0]'
11 #      \           /
12 #      \           /
13 #      \           /
14 #      x[1] \   / x[1]'
15 #      \ \ // .
16 #      . --- z --- .
17 #      . // \| .
18 #      / / \| .
19 #      x[n-1] / \ x[n-1]'
20 #      / \   \
21 #      /       \
22 #      x[n]     x[n]'

23 #
24 #
25 # where x' is the approximation of x based on the z
# components extracted
26
27 # MODIFY THIS LINE
28 RANK = 10
29
30 def custom_loss(y_true, y_pred):
31     return norm(y_true - y_pred, ord='euclidean')
32
33 def frobenius(X, Y):
34     return np.linalg.norm(X - Y, ord='fro')
35
36 boat = np.loadtxt('data/boat.dat')
37 plt.figure()
38 _ = plt.imshow(boat, cmap = cm.Greys_r)
39 plt.show()
40

```

```
File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\svd-on-image.py
41 u,s,vt=np.linalg.svd(boat,full_matrices=False)
42 _ = plt.plot(s)
43 plt.title('Singular values of boat image')
44 plt.show()
45
46 # construct a rank-RANK version of the boat
47 s_copy = s.copy()
48 s_copy[RANK:] = 0
49 boatApprox = u.dot(np.diag(s_copy)).dot(vt)
50
51 model = keras.models.Sequential()
52 model.add(layers.Dense(RANK, use_bias=False, input_dim
= len(boat)))
53 model.add(layers.Dense(len(boat), use_bias=False))
54 model.compile(loss=custom_loss)
55
56 history = model.fit(boat, boat, batch_size=50, epochs=
500)
57
58 boatNNApprox = model.predict(boat)
59
60 print("Frobenius Distance between boat and rank-"+str(
RANK)+" approximation: ", frobenius(boat, boatApprox))
61 print("Frobenius Distance between boat and NN output
with 1 hidden layer of "+str(RANK)+" neurons: ",
frobenius(boat, boatNNApprox))
62
63 plt.figure(figsize=(12,9))
64 plt.subplot(1,3,1)
65 plt.imshow(boatApprox,cmap = cm.Greys_r)
66 plt.title('Rank ' + str(RANK) + ' SVD Boat')
67 plt.subplot(1,3,2)
68 plt.imshow(boatNNApprox,cmap = cm.Greys_r)
69 plt.title('Rank ' + str(RANK) + ' NN Boat')
70 plt.subplot(1,3,3)
71 plt.imshow(boat,cmap = cm.Greys_r)
72 plt.title('Original Boat')
73
74 _ = plt.subplots_adjust(wspace=0.5)
75 plt.show()
76
```

---

# Network Analysis

— Boston University CS 506 - Lance Galletti —

---

- Not really supervised or unsupervised

## Example Networks

### Internet:

- What will internet traffic through Belgium look like today?
- Anomalous traffic patterns
- Model of the internet

### Biology:

- Are certain patterns of interactions among genes more common than expected?
- Which regions of the brain communicate for a given task?

### Social:

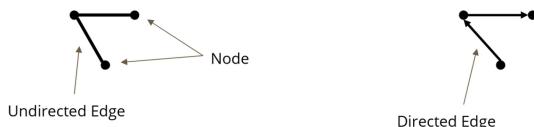
- Who is friends with whom?
- Who are the influencers?
- What social groups are present?
- How does information flow through the network?

- 
- Information flow
    - o Are there particular bottlenecks

## Graphs

We use Graphs and Graph Theory to model / represent and analyse Networks.

A Graph is comprised of Nodes / Vertices connected by Edges. These Edges can be undirected (where edges are symmetrical connections between Nodes - A is connected to B then B is connected to A) or directed.



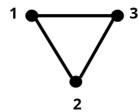
- Composed of nodes and edges
- Undirected vs directed graph
  - o There is only one edge per 2 given nodes
    - Sometimes for certain graphs multiple edges between a node pair is allowed
- Sometimes weights are associated with edges

## Graphs

Formally, a graph  $G$  is an ordered pair of sets  $(V, E)$  where:

- $V$  is the set of all Nodes / Vertices
- $E$  is the set of all Edges

Let  $G = (V, E)$  be undirected where  $V = \{1, 2, 3\}$  and  $E = \{(1,2), (1,3), (2,3)\}$ . What does  $G$  look like?



- $V$ : set of all vertices/nodes
- $E$ : set of all edges
- Triangle

$$\begin{pmatrix} & 1 & 2 & 3 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 \\ 3 & 1 & 1 & 0 \end{pmatrix}$$

by convention  
we say no

## Graphs

How can we efficiently store a graph?

**Adjacency Matrix**

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \longrightarrow \begin{array}{c} \bullet \\ \diagdown \quad \diagup \\ \bullet - \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array}$$

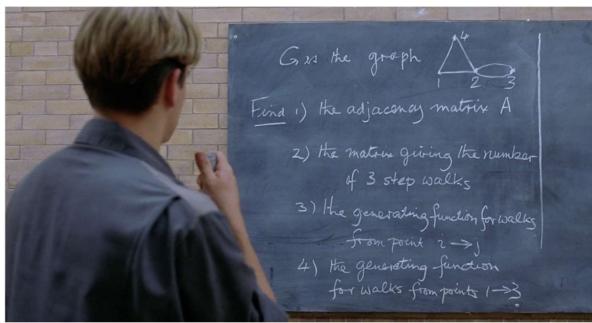
**Adjacency List**

$$\begin{array}{l} 1 : \{2, 3\} \\ 2 : \{1, 3\} \\ 3 : \{1, 2\} \end{array} \longrightarrow \begin{array}{c} \bullet \\ \diagdown \quad \diagup \\ \bullet - \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array}$$

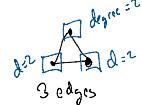

---

- Can capture it as the two sets in previous slide
- But certain queries will be better optimized using a particular structure

## Graphs



- Good Will Hunting
- Now you can solve half of this problem
- This has two edges between 2 and 3
- A node can have an edge onto itself
  - o Usually, by convention, we say no
  - o But depends on setting



## Graphs Characteristics

The **degree** of a Node is the number of edges connected to it.

A **path** between two Nodes is a sequence of edges that joins these two Nodes.

A graph is called **complete** if there is an edge between every pair of Nodes.

Questions:

- What is the sum of the degrees of all Nodes in a Graph as a function of  $N_E$  (the number of Edges)?

$$2 * N_E$$

- How many Edges are in a complete Graph as a function of  $N_V$  (the number of Nodes)

$$N_V(N_V - 1) / 2$$


---

- Degree of node
  - o Number of connections node has
  - o Directed: in degree & out degree
    - We're mainly working with undirected
- Complete: all nodes connected
  - o e.g. triangle
- Some of the degrees of all nodes in a graph as a function of the number of edges:
- $2 * N_E$ 
  - o Every edge is counted twice
- If you shake hands with everyone in the room, you are only going to shake hands with  $n-1$  people because you don't shake hands with yourself

## Graph Problems

- Clique Problem (find largest complete subgraph)
- Coloring Problem (Color a graph with a given a number of colors s.t no two adjacent vertices share a color - or other conditions)
- Travelling Salesman Problem (Given a list of cities and distances between cities, find the shortest path that goes through all cities and returns to its origin)
- Shortest Path (given weights on edges, find the shortest path between two nodes)
- Vertex Cover (When you pick a node, all its adjacent edges get removed. Find the min number of nodes needed to remove all edges from the graph)

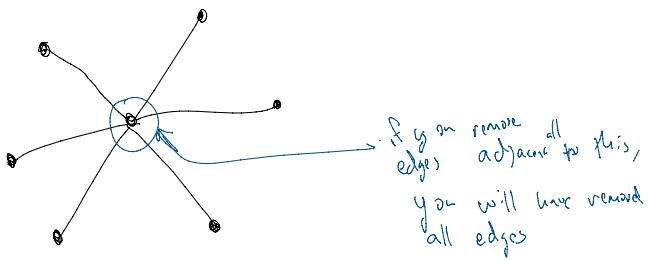
Which one do you think is the easiest to solve?

---

- Color problem



- Color problem
  - o Don't want the same color is never adjacent to itself
- Traveling salesman: shortest path that hits all nodes
- Vertex cover:
  - o How many nodes does it take to remove all the nodes from a graph
- 4: shortest path
  - o Easiest one
  - o Dijkstra's algorithm
    - Algorithm studies
- All the rest are equally hard and equivalent?



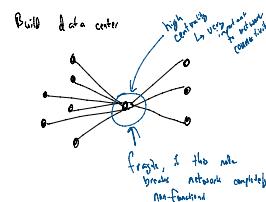
## Network Characteristics

Distribution of edges / node degrees:

- Anomaly detection
- Ranking / Recommendation
- Describe flow through the network

Centrality of a node:

- Identify influencers
- Discover groups / clusterings
- How nodes affect connectivity / flow



- e.g.
  - o Fluid dynamics

## Network Analysis

Networks / Graphs are generated by processes or functions on its nodes / edges. For example: creating a new account (adding a node), making friends / following / connecting (adding an edge), etc.

The state of a Network / Graph at a given point in time is the **stochastic** result of these processes.

One way we can model the characteristics from the previous slide is by modeling the state of the Graph (i.e. finding the random process that generated the given Graph)

- As a random/stochastic result
  - o Completely or at particular frequency
- If the number of connections are exponential, you need to prepare your network ahead of time

## Random Graph Model

1. Let  $G(N, M) = \{G = (V, E) \mid |V| = N, |E| = M\}$  = the set of all graphs with  $N$  nodes and  $M$  edges. Pick uniformly from  $G(N, M)$ .  
 Ex:  $G(3,2) = \{\Delta, V, \nabla\}$  pick each with probability  $1 / 3$

In general, what is the probability with which you pick a graph from  $G(N, M)$ ?

$$p = \binom{\binom{N}{2}}{M}^{-1}$$

2. Let  $G(N, p)$  be generated by randomly connecting nodes with probability  $p$ , independently. What is the probability distribution of  $G(N, p)$  as a function of a number of edges  $M$ ?

**Hint:** we are performing  $N(N-1) / 2$  Bernoulli trials inserting edges independently with probability  $p$ .

$$f_{G(N,M)} = p^M (1-p)^{\binom{N}{2}-M}$$


---

- What is a random graph model?
  - o Pick one of these configurations randomly
- What is the probability with which you pick a graph with  $N$  nodes and  $M$  edges?
  - o How many ways to distribute  $M$  edges over  $N$  nodes
  - o  $M$  edges among  $N$  nodes
  - o Probability
- Or, have a particular probability in mind
  - o Geometric distribution
    - Not quite geometric

## Random Graph Model

Both methods are related in that:  $G(N,p)$  conditioned on the event that it has  $M$  edges, is equal in distribution to  $G(N, M)$ .

Proof:

$$\begin{aligned} P(G(N, p) \mid |E_{G(N,p)}| = M) &= \frac{P(G(N, p), |E_{G(N,p)}| = M)}{P(|E_{G(N,p)}| = M)} \\ &= \frac{p^M (1-p)^{\binom{N}{2}-M}}{\binom{\binom{N}{2}}{M} p^M (1-p)^{\binom{N}{2}-M}} \\ &= \binom{\binom{N}{2}}{M}^{-1} \end{aligned}$$


---

- Proof for why both methods are equivalent

## Random Graph Model

What is the distribution of the **degree** of Nodes?

$$P(\deg(v) = k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}$$

Note: As N goes to infinity while Np remains constant (i.e. p goes to zero at a comparable rate), the above Binomial distribution converges to a Poisson Distribution

Q: What is the expected number of connections / degree for nodes in this Graph?

$$\sim Np$$

Q: Is this realistic for say social networks?

This means on average we all have similar number of connections and that the probability of a high degree node is exponentially small. Probably not realistic.

---

- Binomial distribution
- On average, are most people connected with the same number of people
  - o There should be low variance
  - o BUT this is not accurate/realistic for social networks; some people are extremely connected and some barely

## Power Law

Most real-life social networks follow have a degree distribution following a power law of the form

$$P(k) = Ck^{-\alpha} \text{ for some constants } C \& \alpha$$

What does this mean?



- Try to model the process that generates this particular graph
- To be continued

## Describing / Comparing Graphs

In order to compare graphs, we can define metrics that represent characteristics of the graphs and compare these.

We can talk about metrics that characterize the graph as a whole or characterize a specific node, edge, or set of nodes or edges.

---

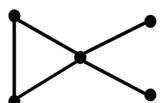
## Metrics on Graphs

### Diameter

Let  $d_{ij}$  be the shortest path between node i and node j. The diameter of G is defined as

$$\text{Diam}(G) = \max_{ij} d_{ij}$$

This captures what we refer to as the small world phenomenon.  
Q: What is the Diameter of



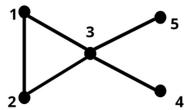
## Metrics on Graphs

### Clustering Coefficient

$$C = \# \text{ triangles} / \# \text{ triplets}$$

A triangle is a closed triplet. A triplet consists of 3 nodes connected by 2 edges. Triangles and triplets are defined as being centered on a node.

Ex: What is the clustering coefficient of



$$C = (1 + 1 + 1 + 0 + 0) / (1 + 1 + 6 + 0 + 0)$$
$$= 3 / 8$$

## Metrics on Graphs

### Density

Let  $N = \# \text{ Nodes}$ ,  $M = \# \text{ Edges}$

$$\text{Density} = 2M / N(N-1)$$

Q: what is the density of a complete graph?  
A: 1

## Metrics on Nodes

### Degree Centrality

The more central a node is, the higher its number of connections

$$C_{\text{deg}}(v) = \text{Deg}(v)$$

### Closeness Centrality

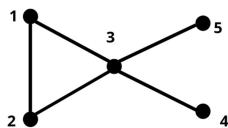
The more central a node is, the closer it is to all other nodes

$$C_{\text{close}}(v) = \frac{1}{\sum_u d(u, v)}$$

---

## Metrics on Nodes

For the following Graph



Q: Which node has the highest degree centrality?

A: 3

Q: What is the closeness centrality (where  $d(u,v) = \#$  edges on the shortest path between  $u & v$ ) of Node 3?

A: 1/4

Q: What is the closeness centrality of Node 5?

A: 1/7

---

## Metrics on Nodes

### Harmonic Centrality

$$C_h(v) = \sum_{u \neq v} \frac{1}{d(u, v)}$$

Where  $d^{-1}(u, v) = 0$  if there is no path between u & v

### Betweenness Centrality

Quantifies the number of times the node acts like a bridge along the shortest path between 2 other nodes

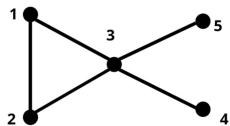
$$C_b(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Where  $\sigma_{st}$  is the total number of shortest paths from s to t and  $\sigma_{st}(v)$  is the number of those shortest paths that go through v

---

## Metrics on Nodes

For the following Graph



Q: What is the harmonic centrality of Node 3?

A: 4

Q: What is the betweenness centrality of Node 3?

A: 10

---

## Recommendation

In homework 3 you will encounter the following problem:

Given a graph and a node  $v$ , how can we recommend nodes that  $v$  is not connected to?

We can rank all other nodes that  $v$  is not connected to from most recommended to least recommended. What scoring function can we use to produce such an ordering?

- Common Neighbors
  - Jaccard's Index
  - Adamic / Adar Index
- 

## Ranking Aggregation

Each of these scoring functions / metrics can generate an ordered list of nodes. We'll refer to this ordered list as a ranking.  
Q: How can we compare rankings?

Suppose we have two rankings  $w_1, w_2$ . Can we use a distance function to compare these?

If we have access to the score, we could take the sum of the square differences in scores.  
Problem is that different scores are not directly comparable.

---

## Ranking Aggregation

We need a distance function on the ordering itself. Does this remind you of anything?

Disagreement distance?

Let's count the number of inversions / disagreements between the rankings for all pairs of Nodes!

---

## Ranking Aggregation

**Kendall  $\tau$  distance**

$$d_\tau(w_1, w_2) = \# \text{ pairs ranked in different order} / \# \text{ pairs}$$

Node	A	B	C	D
w <sub>1</sub>	1	2	3	4
w <sub>2</sub>	3	4	1	2

Pair	Agree
AB	✓
AC	✗
AD	✗
BC	✗
BD	✗
CD	✓

$$d_\tau(w_1, w_2) = 4 / 6$$

---

## Ranking Aggregation

Given  $m$  Rankings  $w_1, \dots, w_m$ , we can generate an aggregate ranking  $w^*$

$$w^* = \arg \max_w \sum_{i=1}^m d_\tau(w, w_i)$$

---