Biologically-inspired Computing II: Neural Networks

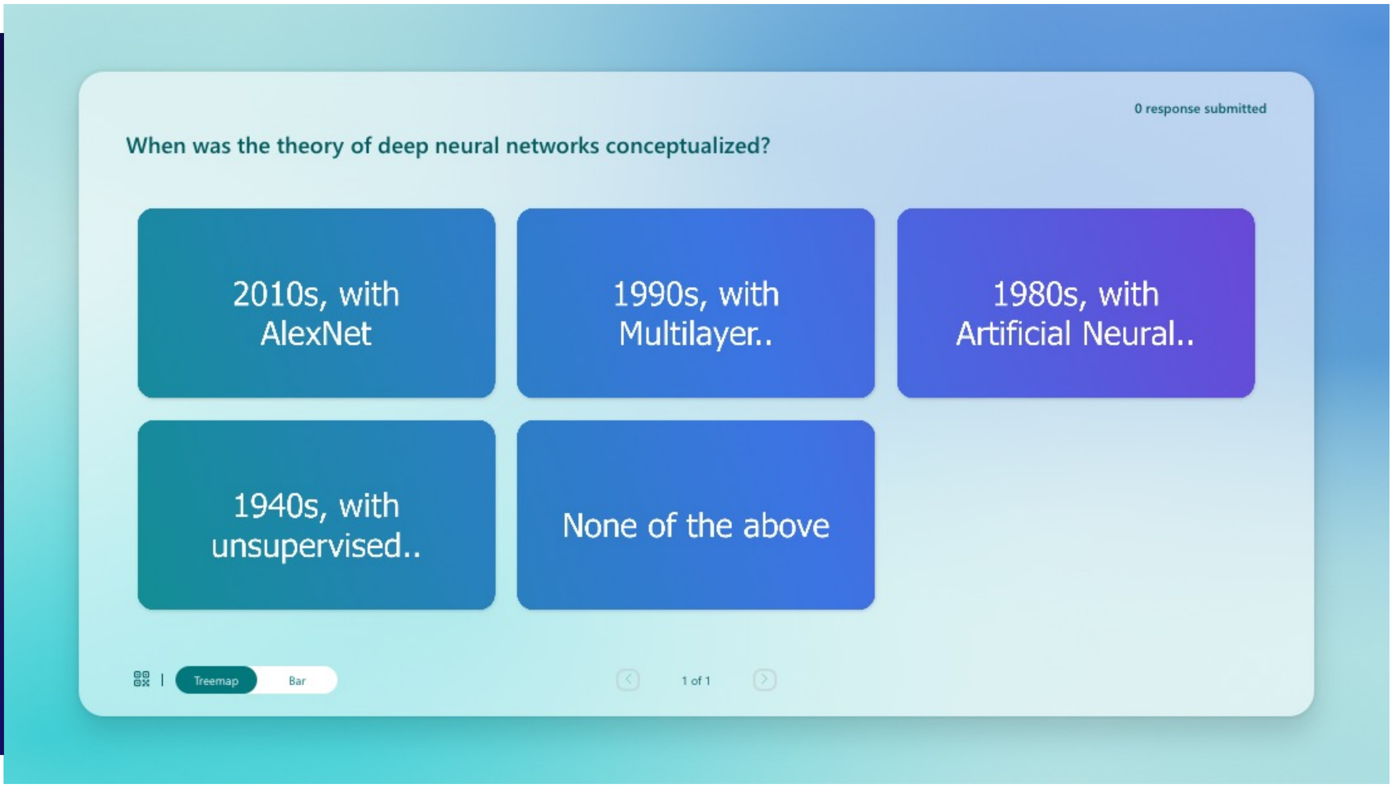
CSCI 4360/6360 Data Science II

Survey

- Who’s heard of artificial neural networks?
- Who knows what a neuron or synapse is?
- Who knows what an activation function is? Different types?
- Input layers, hidden layers, output layers?

- Who’s heard of deep learning?

History of Neural Networks

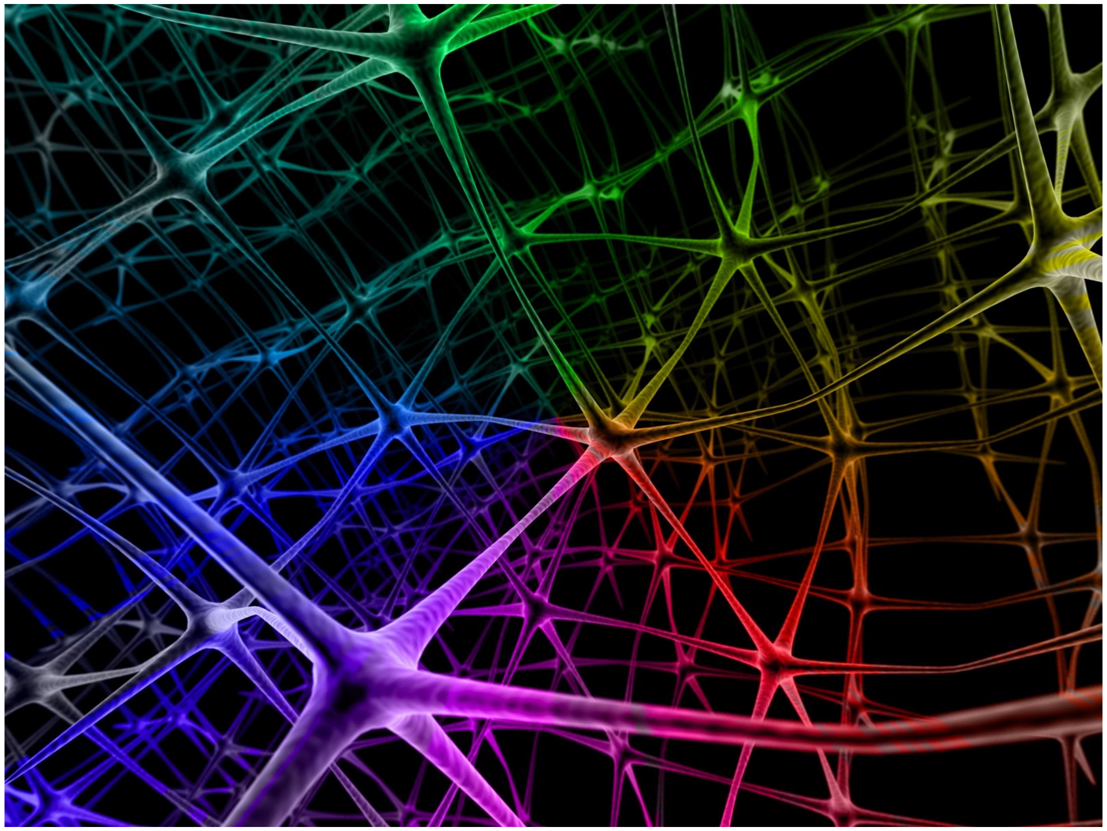

0 response submitted

When was the theory of deep neural networks conceptualized?

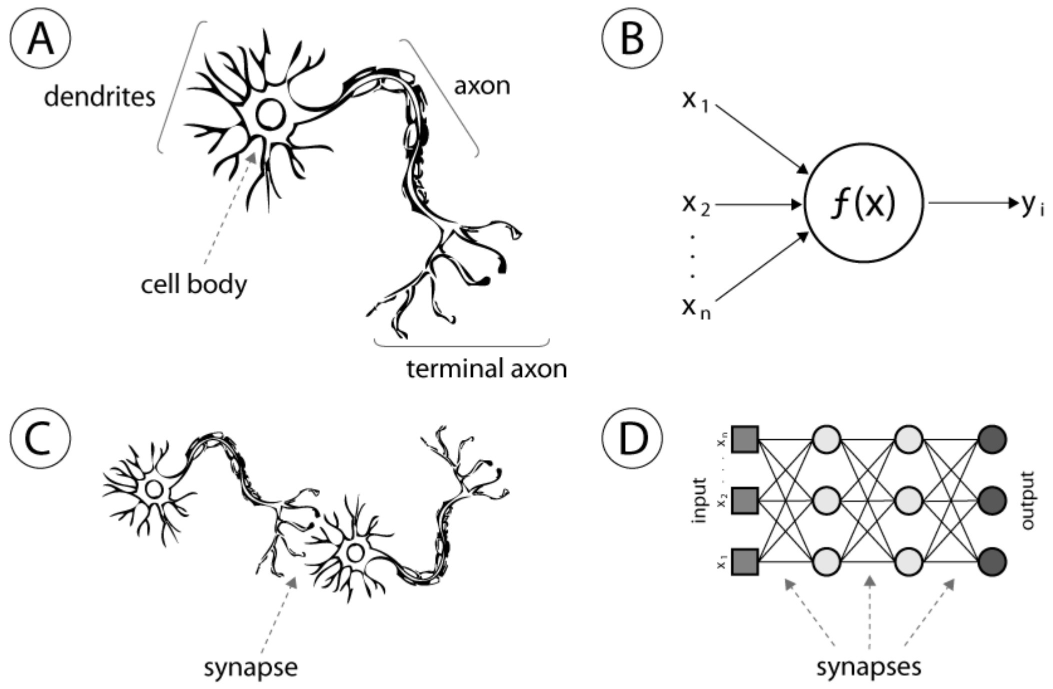
- 2010s, with AlexNet
- 1990s, with Multilayer..
- 1980s, with Artificial Neural..
- 1940s, with unsupervised..
- None of the above

Artificial Neural Networks

- **Not a new concept!**
  - Roots as far back as 1940s work in unsuperivsed learning
  - Took off in 1980s and 1990s
  - Waned in 2000s
- "Biologically-inspired" computing
  - May or may not be true
- **Shift from rule-based to emergent learning**



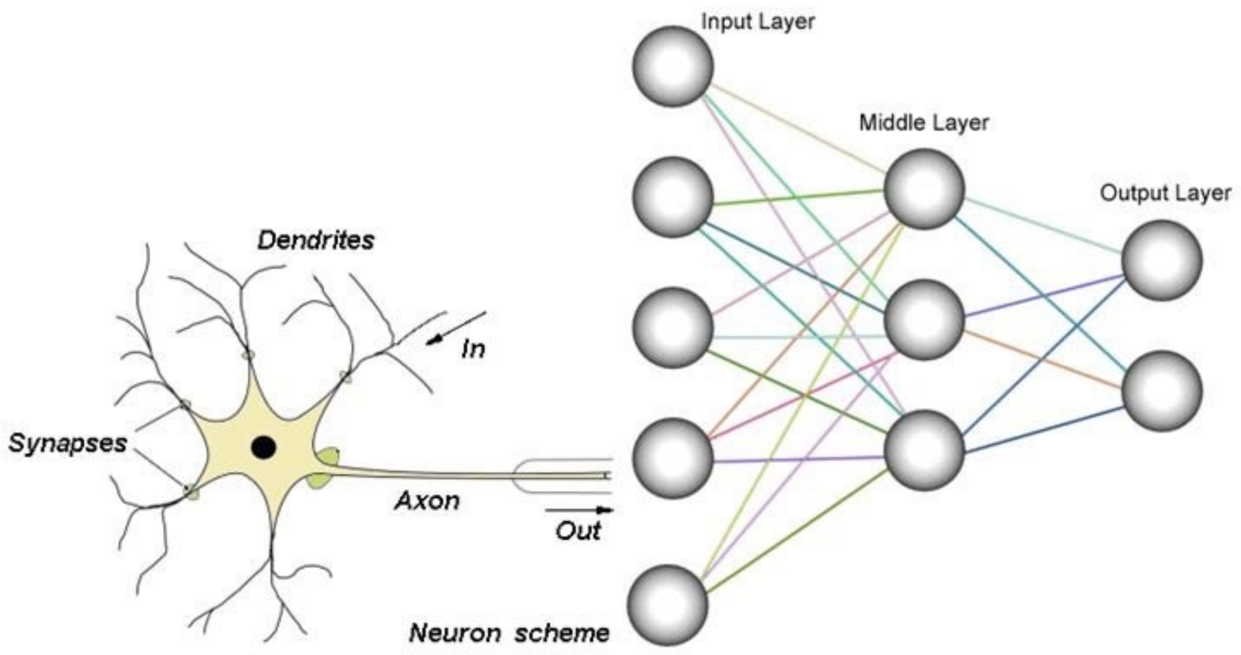
Artificial Neural Networks



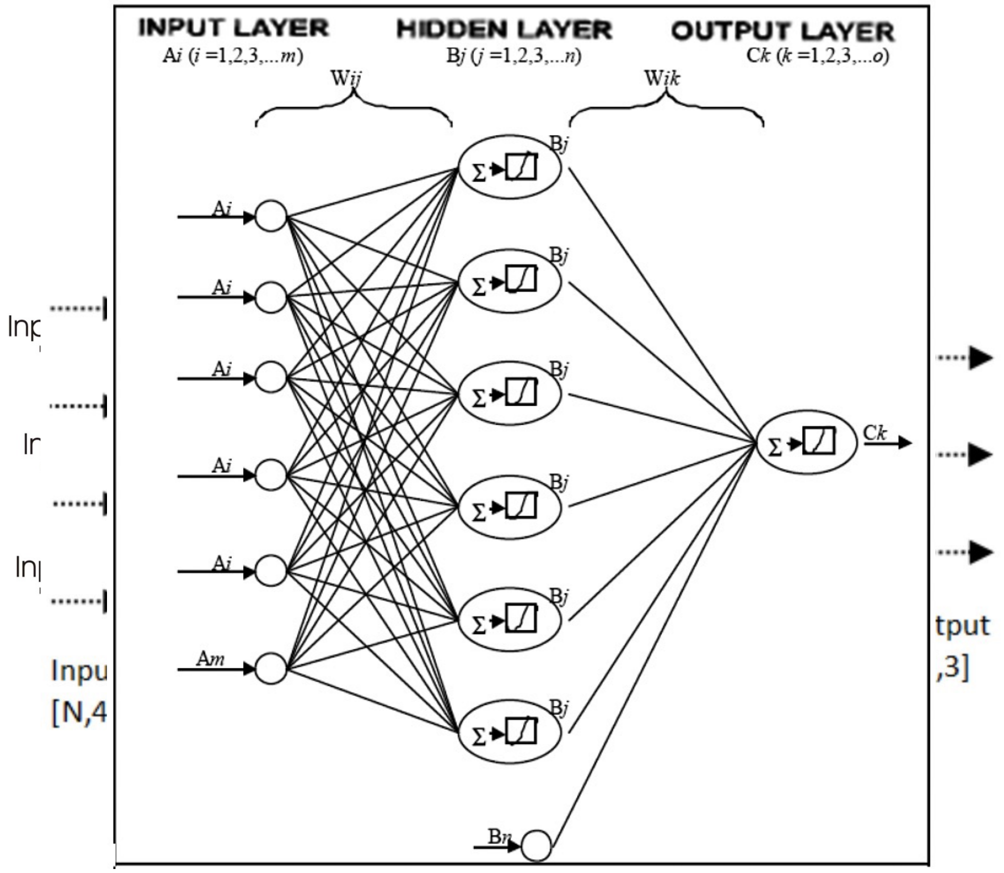
- *Kind-of* modeled after biological brains
  - Hence: “artificial”
- **Neurons:** basic unit of thought and computation
- **Synapses:** connections between neurons
- **Activation functions:** determine whether or not a neuron “fires”, given firings (or not) of previous connected neurons

Artificial Neural Networks

- ANNs organized into *layers*
- Each layer is a collection of neurons
- Each neuron has an activation function that determines whether to ”fire”
- Signal is propagated to the next layer



Artificial Neural Networks



- Types of layers
  - Input
  - Output
  - Hidden
- Types of activation functions
  - Identity
  - Step (threshold)
  - Linear
  - tanh
  - sigmoid
  - Rectified Linear (ReLU)
  - https://en.wikipedia.org/wiki/Activation_function#Comparison_of_activation_functions

Activation Functions

- Among the most important architectural decisions to be made

- **Nonlinear:** two-layer ANN can be proven to be a universal function approximator
- **Continuously differentiable:** essential to gradient-based training of the ANN (which we use in backpropagation)
- **Range:** gradient-based methods are more stable when range of activation function is finite (i.e., tanh is [-1, 1])

Single Neuron

- Input neurons
- Each incoming value from previous layer has a *weight*
- Weighted sum in neuron
- Activation function with a threshold
- Output from neuron sent to next layer

Single Neuron

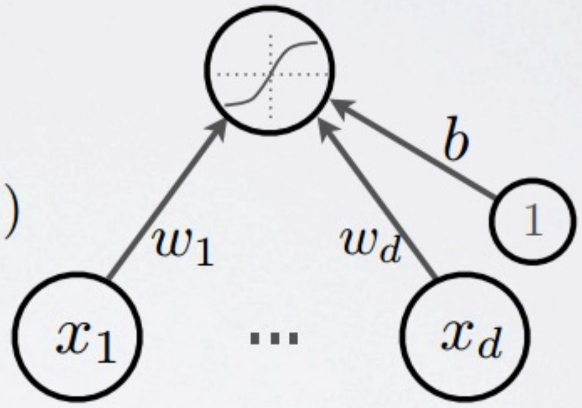
- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

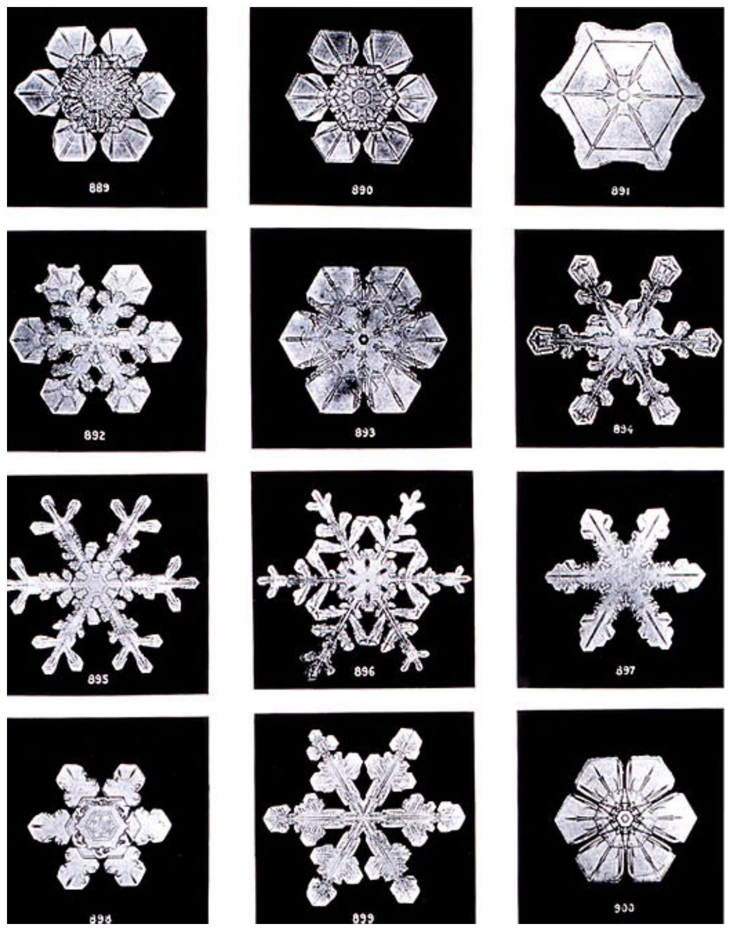
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

- $\mathbf{w}$ are the connection weights
- b is the neuron bias
- $g(\cdot)$ is called the activation function



“Emergent behavior” perspective

- ANNs embody the principle of “emergent behavior”: from relatively simple structure and rules comes remarkably complex phenomena
- Intelligence and intelligent life
- Relationship to ANNs
  - No central network processor
  - “Knowledge” is stored in the network itself (weights)
  - “Hierarchies” of concepts in deep networks



“Emergent behavior” perspective

- Also called “connectionist” models
- **Humans**
  - Neuron switching time: ~0.001s
  - Number of neurons: ~10^{10}
  - Connections per neuron: ~10^4-10^5
  - Scene recognition time: ~0.1s
  - Significant parallel computation
- **ANNs**
  - Neuron-*like* switching units (usually ~10^4; GPT-4 has 10^{12} parameters)
  - Weighted interconnections among units (usually 10^2-10^3)
  - Some parallel computation (limited by hardware, networks, etc)

Upshot: ANN-based artificial intelligence isn’t going to emerge anytime soon

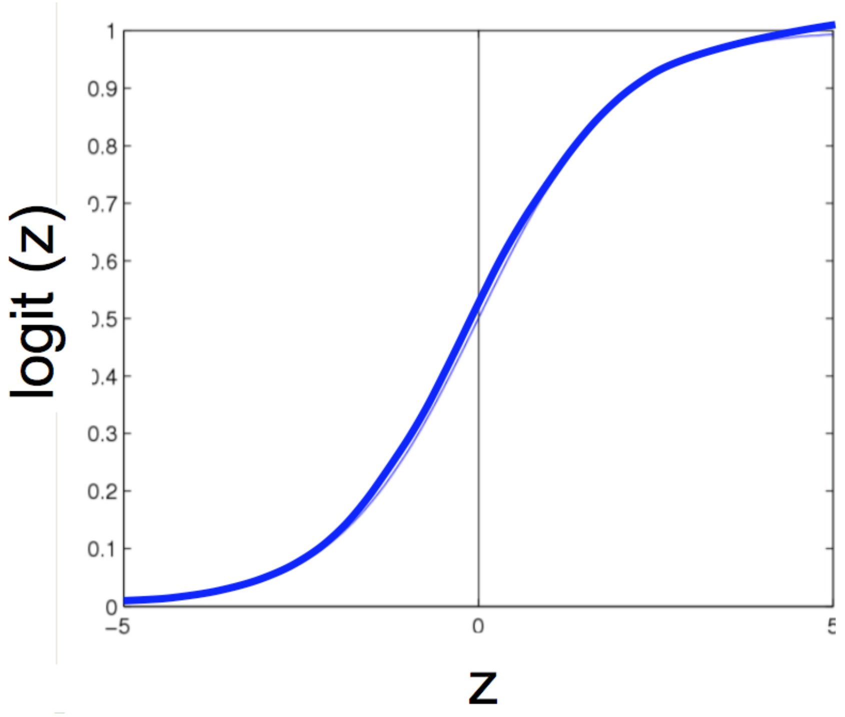
Logistic Regression

- Remember logistic regression?
- Functional form of classifier

$$P(Y=1|X) = \frac{1}{1+\exp(-(w_0+\sum_i w_i X_i))}$$

- Logit function applied to a weighted linear combination of the data

$$\frac{1}{1+\exp(-z)}$$



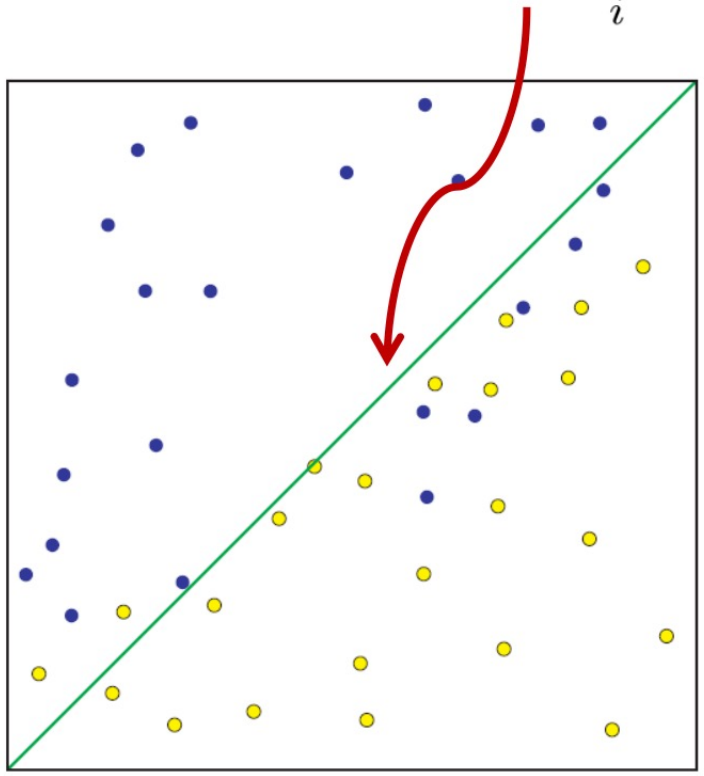
Logistic Regression

- LR is a linear classifier

$$P(Y=0|X) \underset{1}{\overset{0}{\gtrless}} P(Y=1|X)$$

$$0 \underset{1}{\overset{0}{\gtrless}} w_0 + \sum_i w_i X_i$$

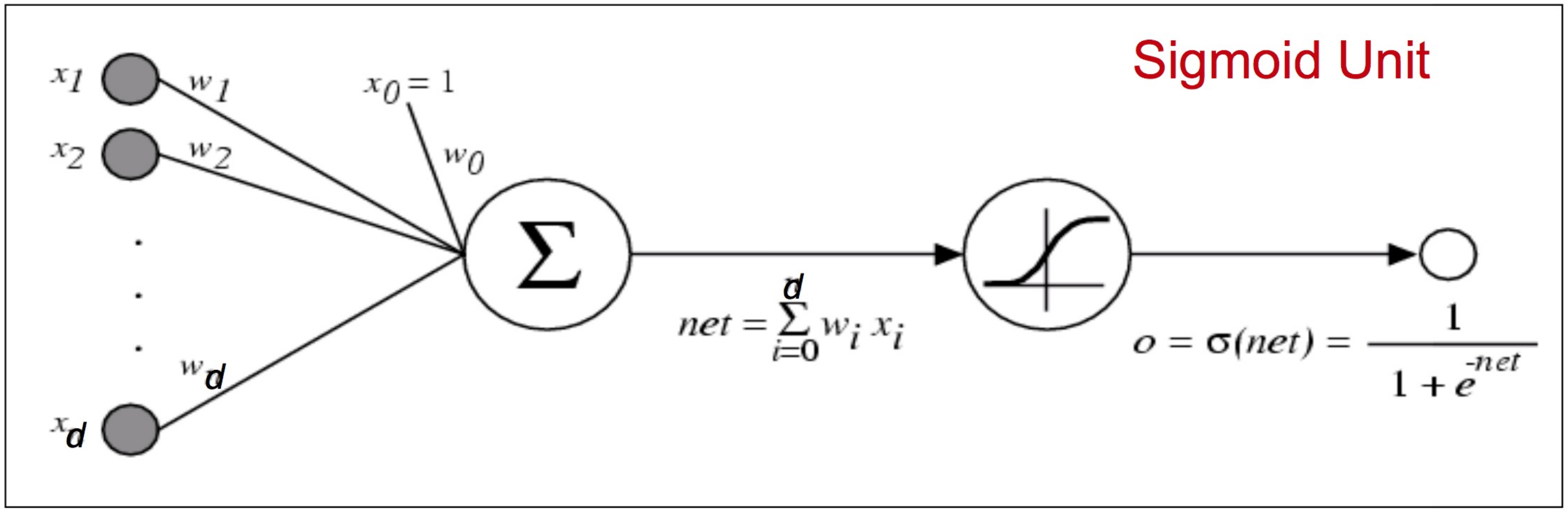
$$w_0 + \sum_i w_i X_i = 0$$



LR as a Graph

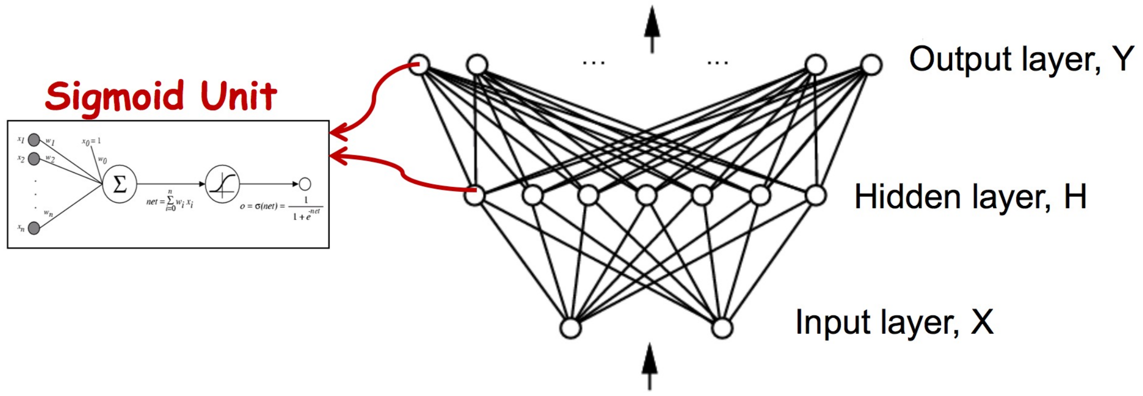
- Define output *o(x)* =

$$\sigma(w_0 + \sum_i w_i X_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$


Sigmoid Unit
x_1
x_2
x_d
w_1
w_2
w_d
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{d} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

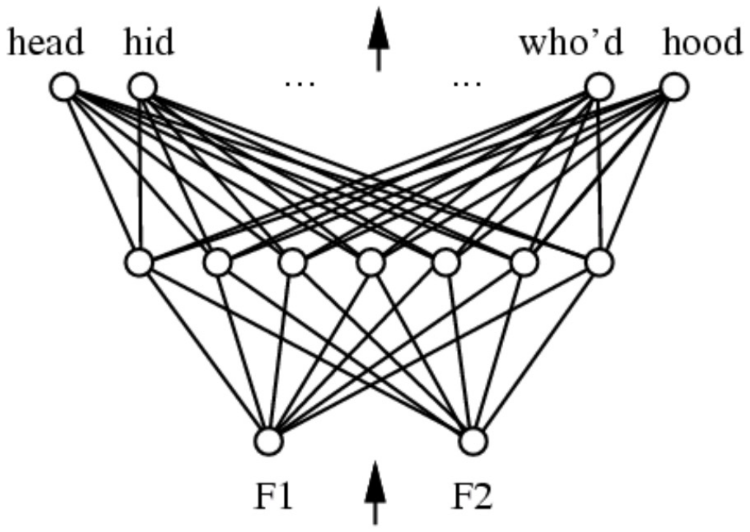
Properties of ANNs

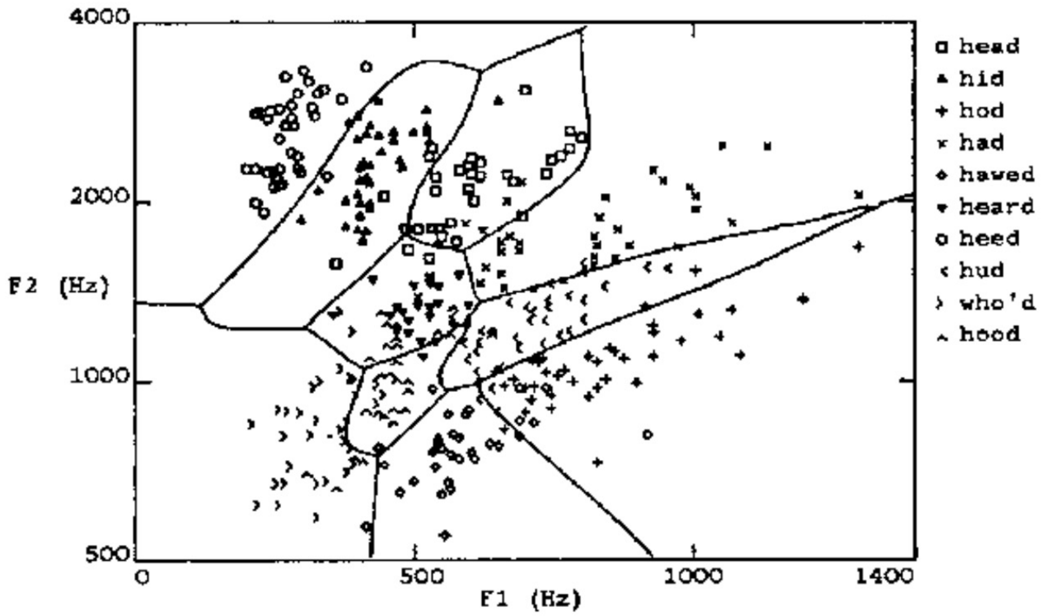
- ANNs learn some $f : X \rightarrow Y$
- X and Y can be continuous / discrete variables
- Focus on *feed-forward neural networks*
  - Form *directed acyclic graphs*, or DAGs
  - **Will break this focus when we reach recurrent neural networks!**


Sigmoid Unit
$x_0 = 1$
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$
Output layer, Y
Hidden layer, H
Input layer, X

ANN in practice

- Learn to differentiate homonyms using frequencies in audio



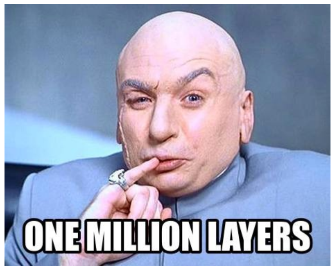


Deep Learning

- Really a reformulation of neural networks to be “deep”
- Original conception called for multilayer neural networks (“multilayer perceptrons”)
- Ran into numerous problems:
  - Theoretical (how to optimize over parameters of deep networks?)
  - Empirical (gradients vanish / explode over deep networks)
  - Engineering (hardware isn’t capable of training deep networks)

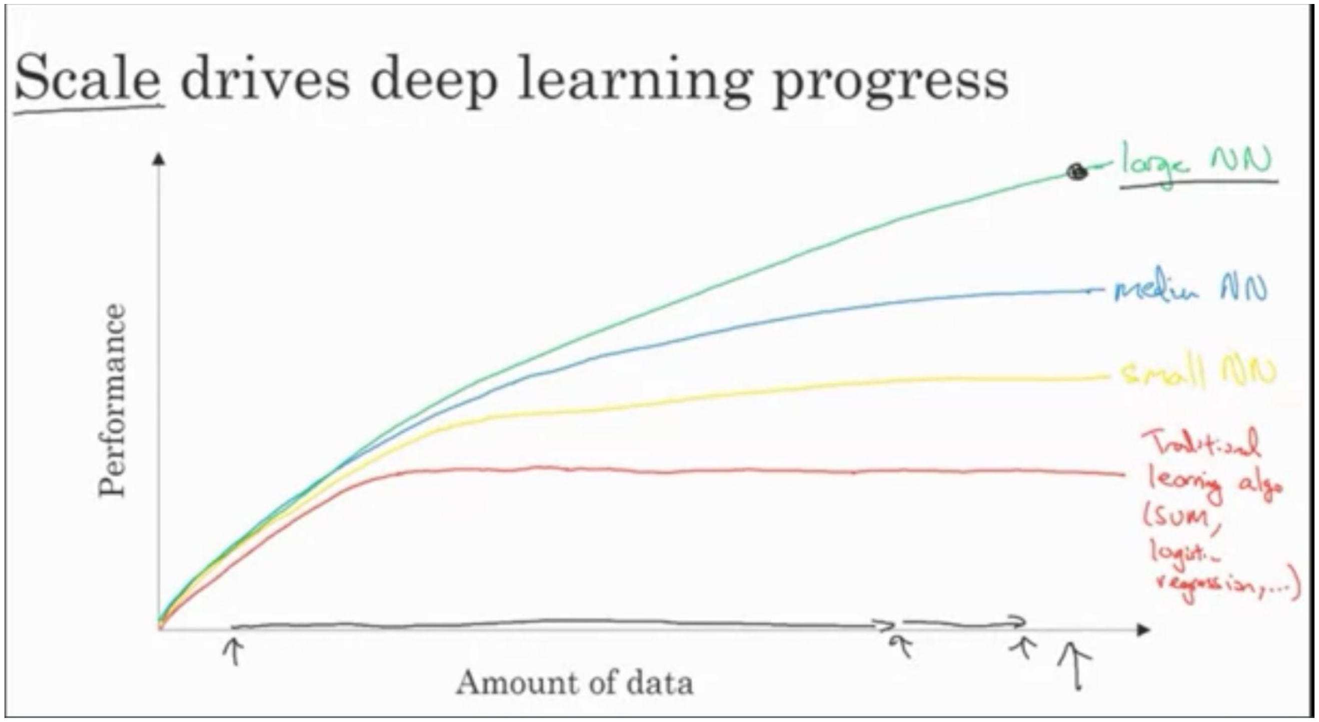
Why is ”deep learning” a thing?

- Concepts have been around for decades
- 1950s: didn’t have backpropagation theory to efficiently train perceptrons of more than 1 layer
- 1980s: didn’t have hardware to efficiently compute gradients for more than 2-3 hidden layers
- 1990s: didn’t have enough data to make deep learning feasible
- 2000s: too depressed with previous failures to look into neural networks; pursued e.g. SVMs instead
- 2010s: Deep Learning^{TM}
- 2020s: Large Language Models (Transformers)



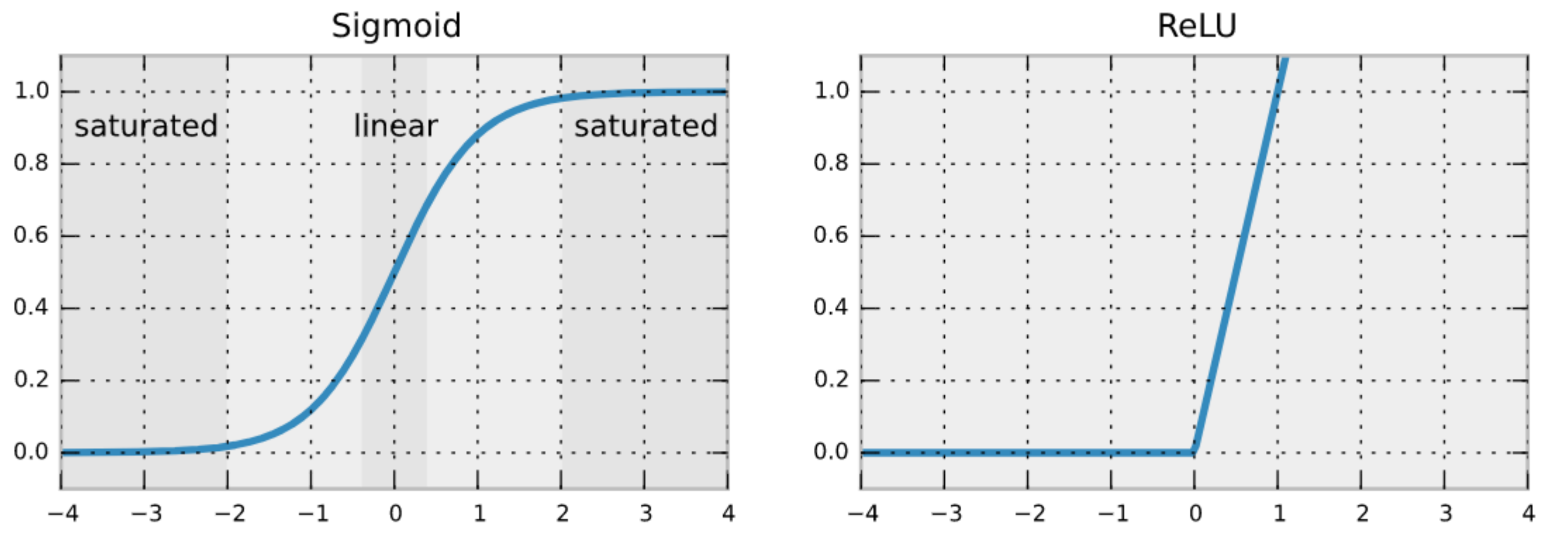
Deep Learning Catalysts

- Scaling of **data** and **computation**
- **Data**
  - ”Big data”
- **Computation**
  - Specialized hardware
  - Open source frameworks
- **Algorithms**
  - Efficient implementations
  - New paradigms



Deep Learning Catalysts

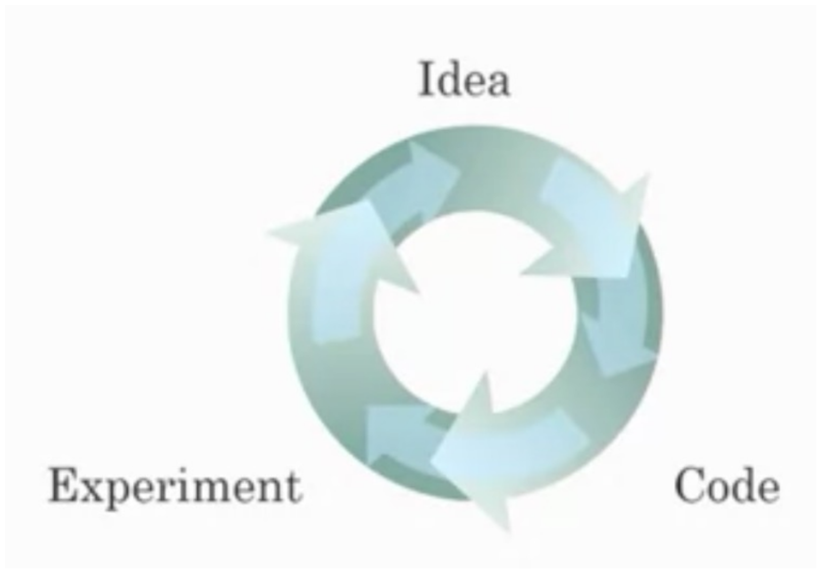
- Switching from sigmoid to ReLU activation functions



- Sigmoid becomes "saturated" at tails, resulting in very slow learning progress

Deep Learning Catalysts

- Hardware efficiency (i.e. Moore’s Law)
- Faster prototyping of
  - New ANN architectures
  - New datasets
  - New activation functions
  - …
- Practitioners and researchers benefit



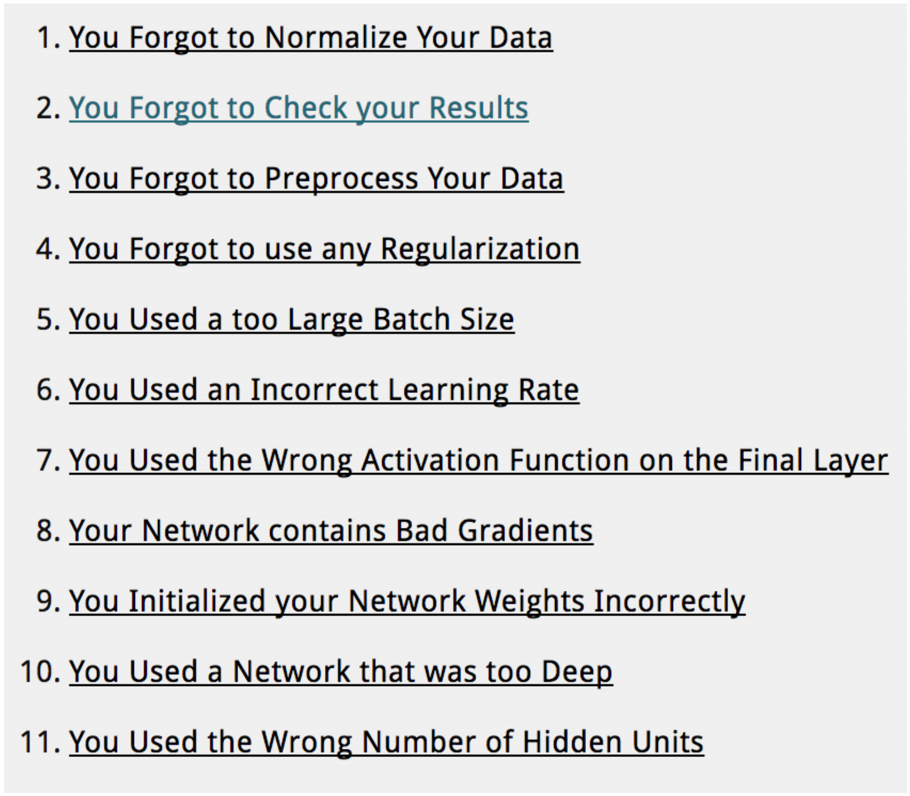
TensorFlow Playground

- Observe the process ANN training (concentric circles dataset)

- 2 inputs, 1 hidden layer (4 neurons), Sigmoid activation, L1 regularization
- 2 inputs, 1 hidden layer (4 neurons), ReLU activation, L1 regularization
- 4 inputs, 0 hidden layers, ReLU activation, L1 regularization

- **How many training epochs are needed?**
- **What are the weights?**

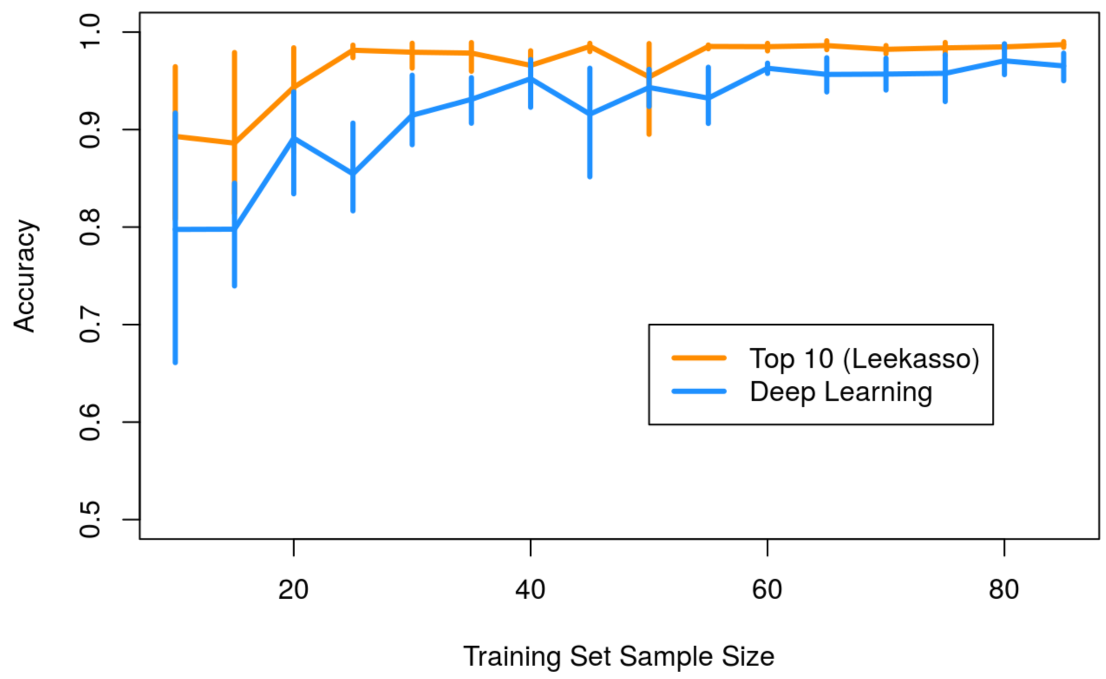
Deep Learning: Caveats



1. You Forgot to Normalize Your Data
2. You Forgot to Check your Results
3. You Forgot to Preprocess Your Data
4. You Forgot to use any Regularization
5. You Used a too Large Batch Size
6. You Used an Incorrect Learning Rate
7. You Used the Wrong Activation Function on the Final Layer
8. Your Network contains Bad Gradients
9. You Initialized your Network Weights Incorrectly
10. You Used a Network that was too Deep
11. You Used the Wrong Number of Hidden Units

Deep Learning: More caveats

- (along the lines of "don't use Hadoop if your data isn't that big")
- At smaller data sizes, no discernible performance bump from deep learning versus "traditional" methods
- "Traditional" methods likely more interpretable and simpler to use



Deep Learning: Even more caveats

- Fizz Buzz (the classic interview question)

“Print numbers 1 through 100, except: if the number is divisible by 3 print ‘fizz’; if it’s divisible by 5 print ‘buzz’; if it’s divisible by 15 print ‘fizzbuzz’.”

- …in Tensorflow!

interviewer: OK, so I need you to print the numbers from 1 to 100, except that if the number is divisible by 3 print "fizz", if it's divisible by 5 print "buzz", and if it's divisible by 15 print "fizzbuzz".

me: I'm familiar with it.

interviewer: Great, we find that candidates who can't get this right don't do well here.

me: ...

interviewer: Here's a marker and an eraser.

me: [thinks for a couple of minutes]

interviewer: Do you need help getting started?

me: No, no, I'm good. So let's start with some standard imports:

```
import numpy as np
import tensorflow as tf
```

interviewer: Um, you understand the problem is *fizzbuzz*, right?

me: Now we need to set up our model in tensorflow. Off the top of my head I'm not sure how many hidden units to use, maybe 10?

interviewer: ...

me: Yeah, possibly 100 is better. We can always change it later.

```
NUM_HIDDEN = 100
```

We'll need an input variable with width NUM_DIGITS, and an output variable with width 4:

```
X = tf.placeholder("float", [None, NUM_DIGITS])
Y = tf.placeholder("float", [None, 4])
```

interviewer: How far are you intending to take this?

me: Oh, just two layers deep -- one hidden layer and one output layer. Let's use randomly-initialized weights for our neurons:

```
def init_weights(shape):
    return tf.Variable(tf.random_normal(shape, stddev=0.01))

w_h = init_weights([NUM_DIGITS, NUM_HIDDEN])
w_o = init_weights([NUM_HIDDEN, 4])
```

So each training pass looks like

```
        for start in range(0, len(trX), BATCH_SIZE):
            end = start + BATCH_SIZE
            sess.run(train_op, feed_dict={X: trX[start:end], Y: trY[start:end]})
```

and then we can print the accuracy on the training data, since why not?

```
        print(epoch, np.mean(np.argmax(trY, axis=1) ==
                             sess.run(predict_op, feed_dict={X: trX, Y: trY})))
```

interviewer: Are you serious?

me: Yeah, I find it helpful to see how the training accuracy evolves.

interviewer: ...

And then our output is just our `fizz_buzz` function applied to the model output:

```
    teY = sess.run(predict_op, feed_dict={X: teX})
    output = np.vectorize(fizz_buzz)(numbers, teY)

    print(output)
```

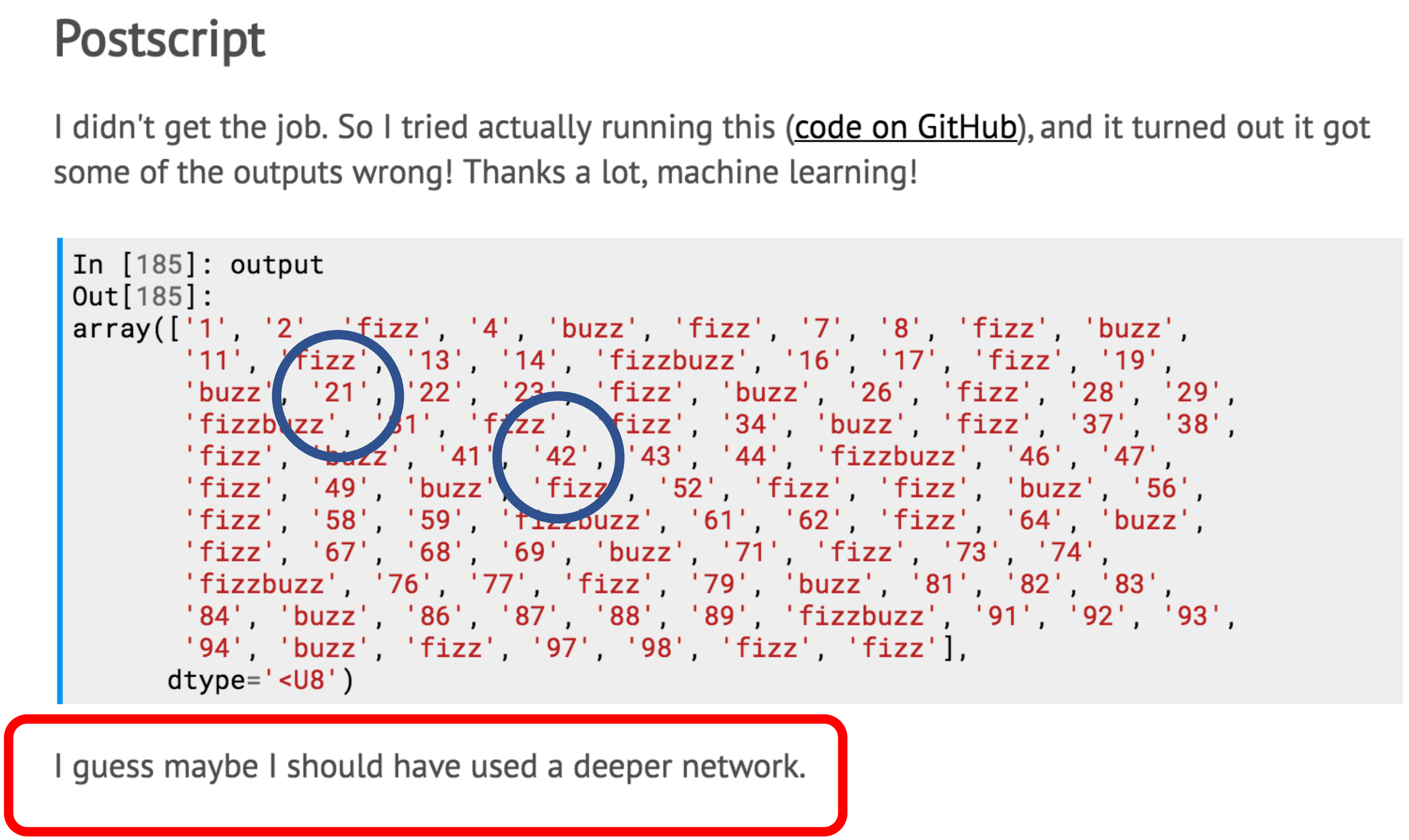
interviewer: ...

me: And that should be your fizz buzz!

interviewer: Really, that's enough. We'll be in touch.

me: In touch, that sounds promising.

interviewer: ...



Postscript

I didn't get the job. So I tried actually running this (code on GitHub), and it turned out it got some of the outputs wrong! Thanks a lot, machine learning!

```
In [185]: output
Out[185]:
array(['1', '2', 'fizz', '4', 'buzz', 'fizz', '7', '8', 'fizz', 'buzz',
       '11', 'fizz', '13', '14', 'fizzbuzz', '16', '17', 'fizz', '19',
       'buzz', '21', '22', '23', 'fizz', 'buzz', '26', 'fizz', '28', '29',
       'fizzbuzz', '31', 'fizz', 'fizz', '34', 'buzz', 'fizz', '37', '38',
       'fizz', 'buzz', '41', '42', '43', '44', 'fizzbuzz', '46', '47',
       'fizz', '49', 'buzz', 'fizz', '52', 'fizz', 'fizz', 'buzz', '56',
       'fizz', '58', '59', 'fizzbuzz', '61', '62', 'fizz', '64', 'buzz',
       'fizz', '67', '68', '69', 'buzz', '71', 'fizz', '73', '74',
       'fizzbuzz', '76', '77', 'fizz', '79', 'buzz', '81', '82', '83',
       '84', 'buzz', '86', '87', '88', '89', 'fizzbuzz', '91', '92', '93',
       '94', 'buzz', 'fizz', '97', '98', 'fizz', 'fizz'],
      dtype='<U8')
```

I guess maybe I should have used a deeper network.

Next:

- A walkthrough of various architectures (CNNs, RNNs, autoencoders, GANs, Transformers, etc)
  - Swappable architectural units
  - Theory
  - Application strengths and weaknesses
- Final project updates
  - #1 is due a week from today! (March 27)
  - #2 is due two weeks later (April 10)
  - Final project presentations are the last week of class (April 22-24)
- Homeworks
  - HW4 due March 31
  - HW5 comes out March 31, due April 15

Questions?

References

- "Why is Deep Learning Taking Off?" https://www.coursera.org/learn/neural-networks-deep-learning/lecture/praGm/why-is-deep-learning-taking-off
- TensorFlow Playground http://playground.tensorflow.org/
- "My neural network isn't working! What should I do?" http://theorangeduck.com/page/neural-network-not-working
- "Don't use deep learning, your data isn't that big" https://simplystatistics.org/2017/05/31/deeplearning-vs-leekasso/
- "Fizz Buzz in Tensorflow" http://joelgrus.com/2016/05/23/fizz-buzz-in-tensorflow/