Scientific Computing

Wednesday, April 15

Announcements

* Friday, April 17: pre-recorded lecture
  no office hours

* Homework 6 assigned,
  due Monday, April 27, 11:59pm

Office Hours:

Mon, 9:30-10:30

~~Fri, 2:00-3:00~~

Cudahy 307

Coding time:

First: the "numpy" Python library

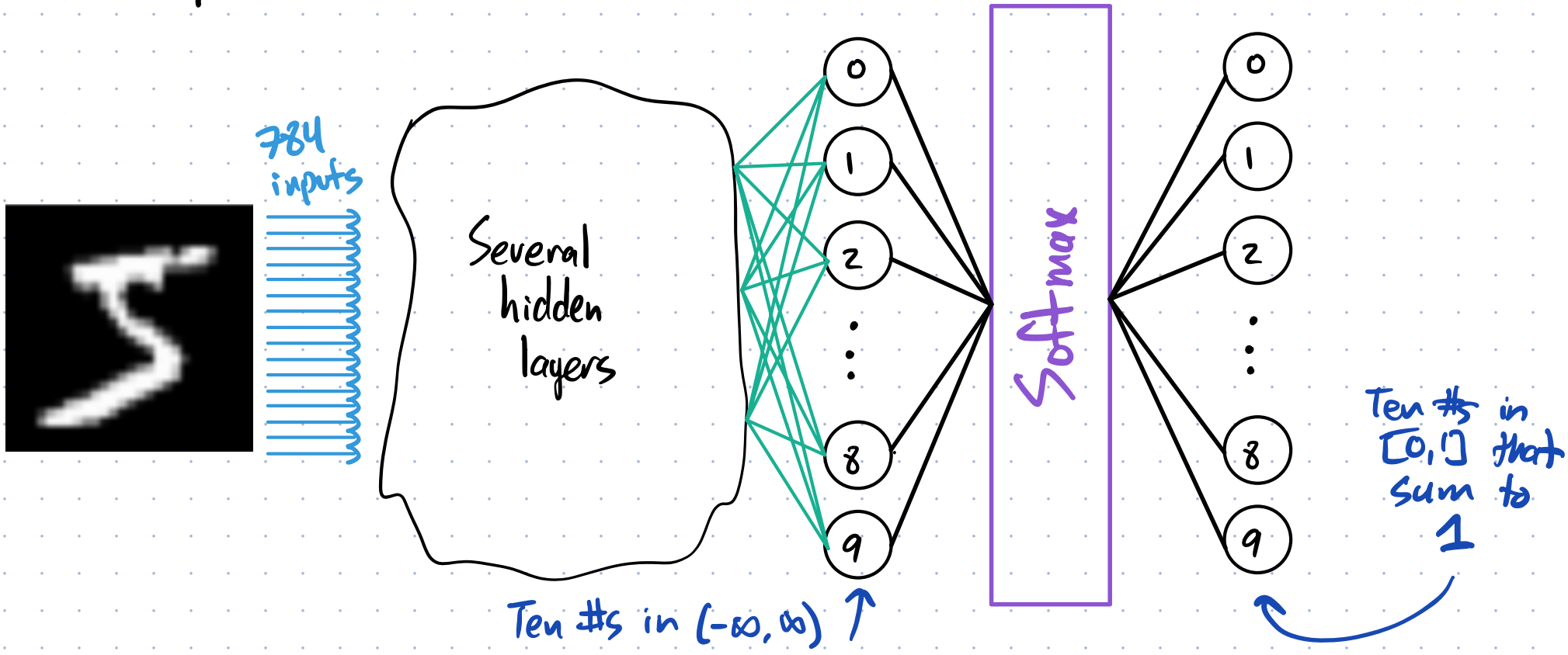
* Jupyter notebook demo

Next: Coding our first layers together from scratch.

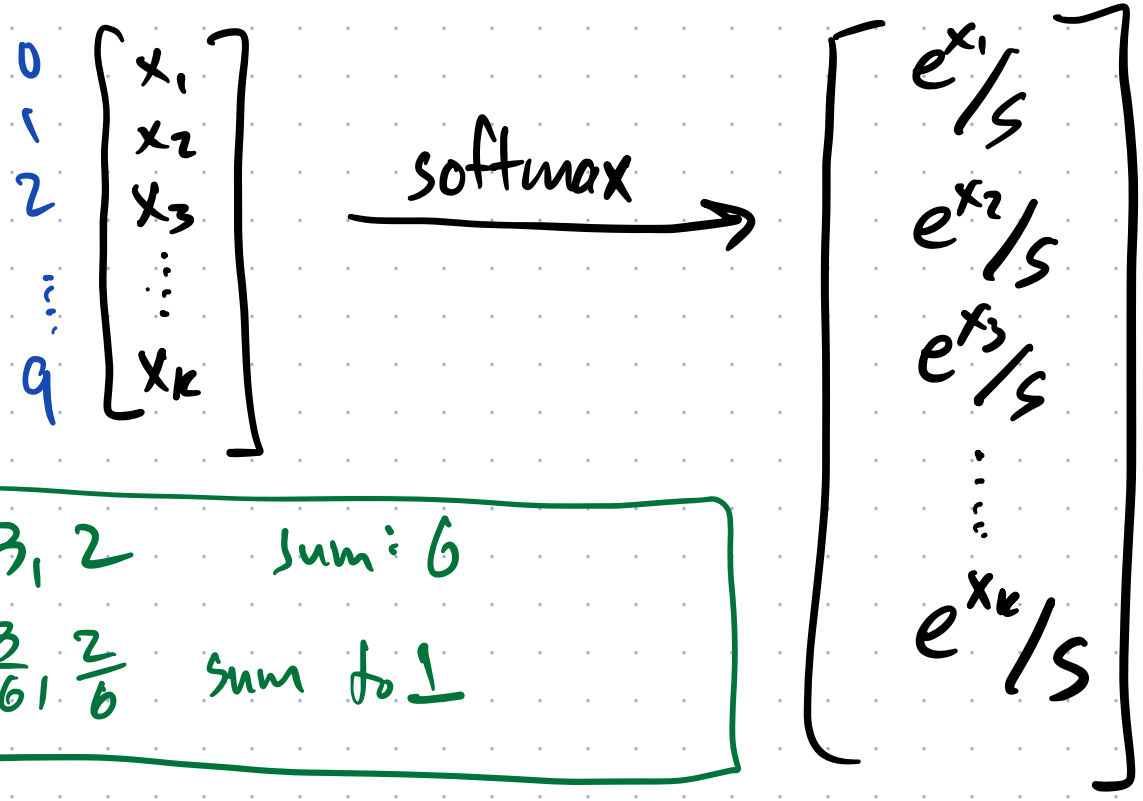
(very heavy inspiration from the "Neural Networks from Scratch" book!)

New Activation Function: Softmax

* Turns a vector of #s into a probability distribution
  (a vector of #s in [0,1] that sums to 1)
* Useful for the output layer in a classification problem.



Unlike our other activation functions, Softmax works on the whole vector at once, not one value at a time individually.



where $S = \sum_{i=1}^{k} e^{x_i}$

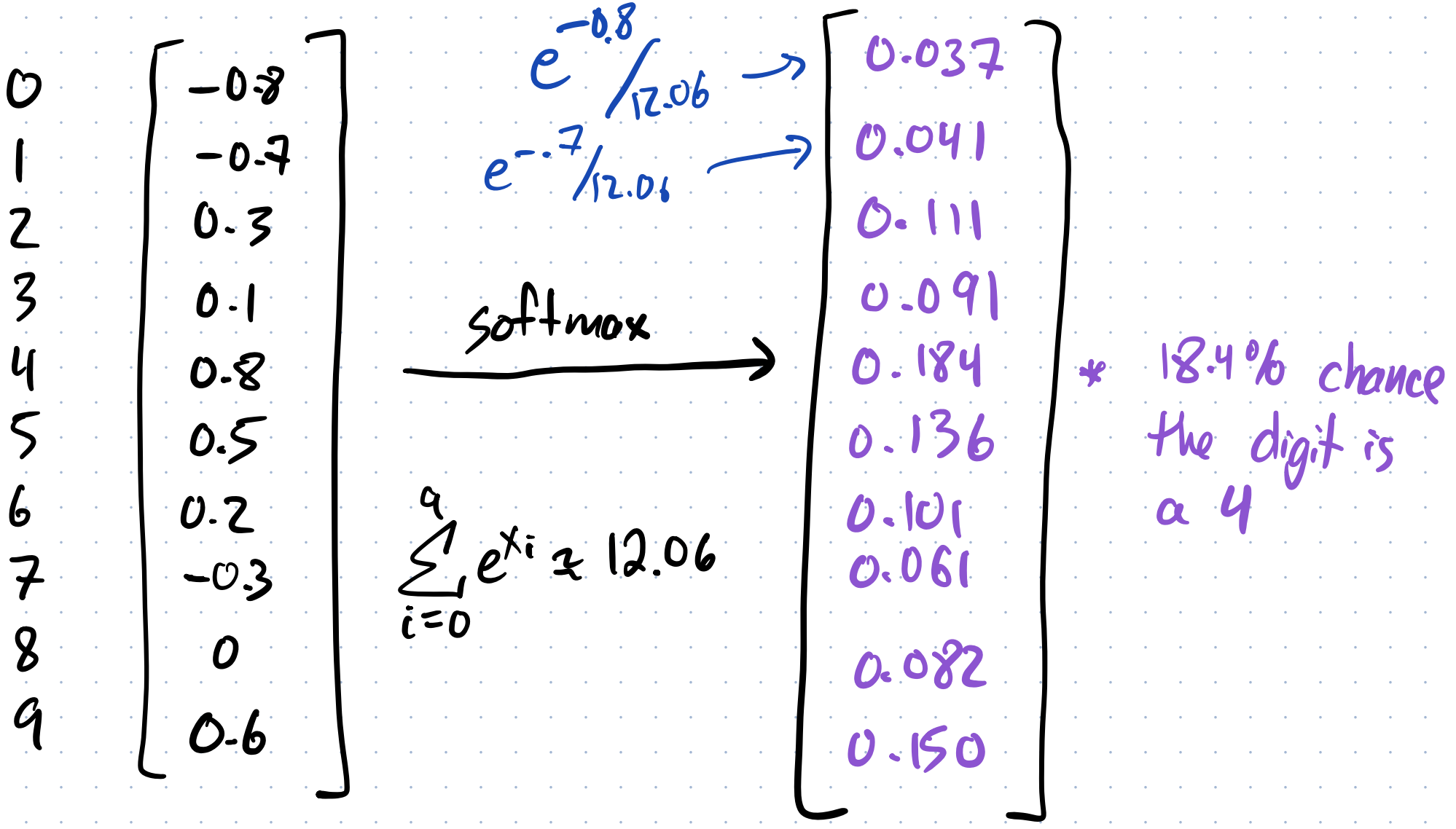
1, 3, 2 sum: 6

$\frac{1}{6}, \frac{3}{6}, \frac{2}{6}$ sum to 1

Obvious these add up to 1 because the denominator is their sum.

Obviously > 0 because $e^x > 0$.

Ex:


0
1
2
3
4
5
6
7
8
9
-0.8
-0.7
0.3
0.1
0.8
0.5
0.2
-0.3
0
0.6
softmax
$\sum_{i=0}^{9} e^{x_i} \approx 12.06$
$e^{-0.8}/12.06$
$e^{-.7}/12.06$
0.037
0.041
0.111
0.091
0.184
0.136
0.101
0.061
0.082
0.150
* 18.4% chance the digit is a 4

Numeric Stability (because e^x gets big!)

Let $m = \max(\vec{x})$

Then do $\dfrac{e^{x_i - m}}{\sum_{j=1}^{k} e^{x_j - m}}$ all components ≤ 0

$\dfrac{e^{x_i}}{\sum e^{x_i}}$

$$\frac{e^{x_i - m}}{\sum_{j=1}^{k} e^{x_j - m}} = \frac{\frac{e^{x_i}}{e^m}}{\sum_{j=1}^{k} \frac{e^{x_j}}{e^m}} = \frac{\frac{1}{e^m} \cdot e^{x_i}}{\frac{1}{e^m} \cdot \sum_{j=1}^{k} e^{x_j}}$$

* Let's add this as a new Activation Function class

Batching

So far we've done the linear algebra and coding for feeding forward one input vector at a time.

one input vector

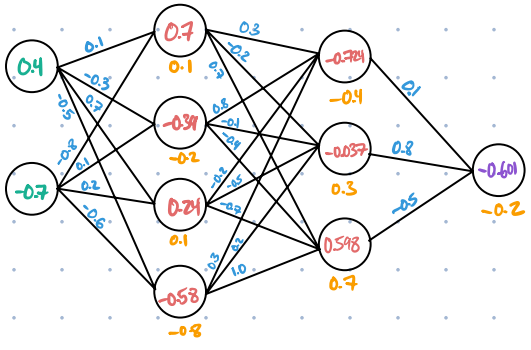
$$\begin{bmatrix} 0.1 & -0.8 \\ -0.3 & 0.1 \\ 0.7 & 0.2 \\ -0.5 & -0.6 \end{bmatrix} \begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \\ 0.1 \\ -0.8 \end{bmatrix} = \begin{bmatrix} 0.7 \\ -0.39 \\ 0.24 \\ -0.58 \end{bmatrix}$$

(ignoring activation functions!)

$$\begin{bmatrix} 0.3 & 0.8 & -0.2 & 0.3 \\ -0.2 & -0.1 & -0.5 & 0.2 \\ 0.7 & -0.4 & -0.7 & 1.0 \end{bmatrix} \begin{bmatrix} 0.7 \\ -0.39 \\ 0.24 \\ -0.58 \end{bmatrix} + \begin{bmatrix} -0.4 \\ 0.3 \\ 0.7 \end{bmatrix} = \begin{bmatrix} -0.724 \\ -0.037 \\ 0.598 \end{bmatrix}$$

$$\begin{bmatrix} 0.1 & 0.8 & -0.5 \end{bmatrix} \begin{bmatrix} -0.724 \\ -0.037 \\ 0.598 \end{bmatrix} + \begin{bmatrix} -0.2 \end{bmatrix}$$

$$= \begin{bmatrix} -0.601 \end{bmatrix}$$



* Numpy does vector calculations faster than individual number calculations

* In the same way, it does matrix calculations faster than one vector at a time!

* We can feed forward many input vectors simultaneously by making one big matrix with many columns.

Let $\begin{bmatrix} \vec{v}_1 & \vec{v}_2 & \vec{v}_3 \end{bmatrix}$ denote the matrix with columns $\vec{v}_1$ $\vec{v}_2$ $\vec{v}_3$.

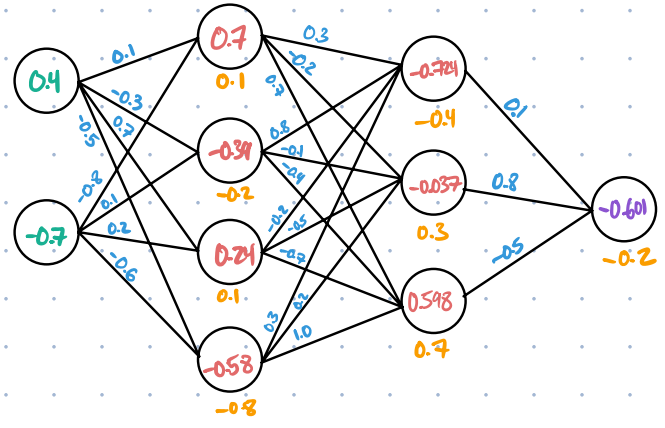
Fact: $M \cdot \begin{bmatrix} \vec{v}_1 & \vec{v}_2 & \vec{v}_3 \end{bmatrix} = \begin{bmatrix} M\vec{v}_1 & M\vec{v}_2 & M\vec{v}_3 \end{bmatrix}$

$$\begin{bmatrix} 0.1 & -0.8 \\ -0.3 & 0.1 \\ 0.7 & 0.2 \\ -0.5 & -0.6 \end{bmatrix} \begin{bmatrix} 0.4 & -0.1 & 0.8 \\ -0.7 & 0.2 & 0.6 \end{bmatrix} + \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ -0.2 & -0.2 & -0.2 \\ 0.1 & 0.1 & 0.1 \\ -0.8 & -0.8 & -0.8 \end{bmatrix} = \begin{bmatrix} 0.7 & -0.7 & -0.3 \\ -0.39 & -0.15 & -0.38 \\ 0.24 & 0.07 & 0.78 \\ -0.58 & -0.87 & -1.56 \end{bmatrix}$$

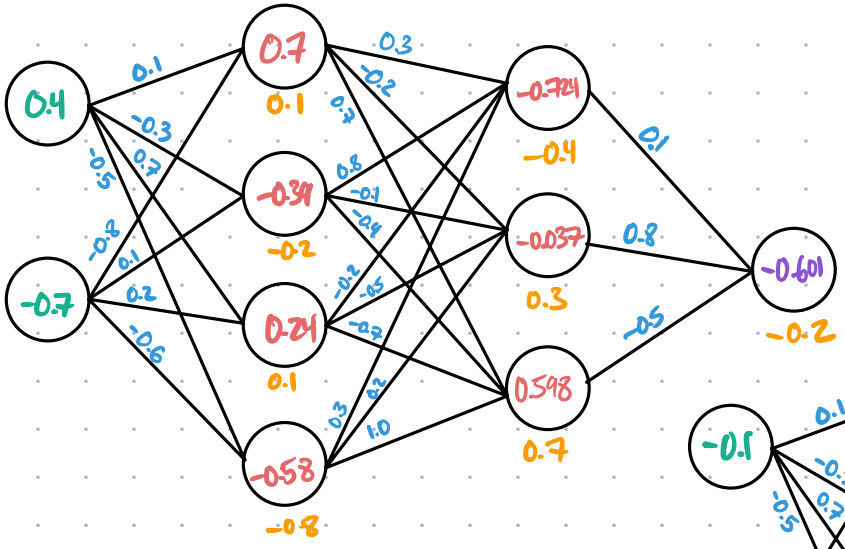
$$\begin{bmatrix} 0.3 & 0.8 & -0.2 & 0.3 \\ -0.2 & -0.1 & -0.5 & 0.2 \\ 0.7 & -0.4 & -0.7 & 1.0 \end{bmatrix} \begin{bmatrix} 0.7 & -0.7 & -0.3 \\ -0.39 & -0.15 & -0.38 \\ 0.24 & 0.07 & 0.78 \\ -0.58 & -0.87 & -1.56 \end{bmatrix} + \begin{bmatrix} -0.4 & -0.4 & -0.4 \\ 0.3 & 0.3 & 0.3 \\ 0.7 & 0.7 & 0.7 \end{bmatrix}$$

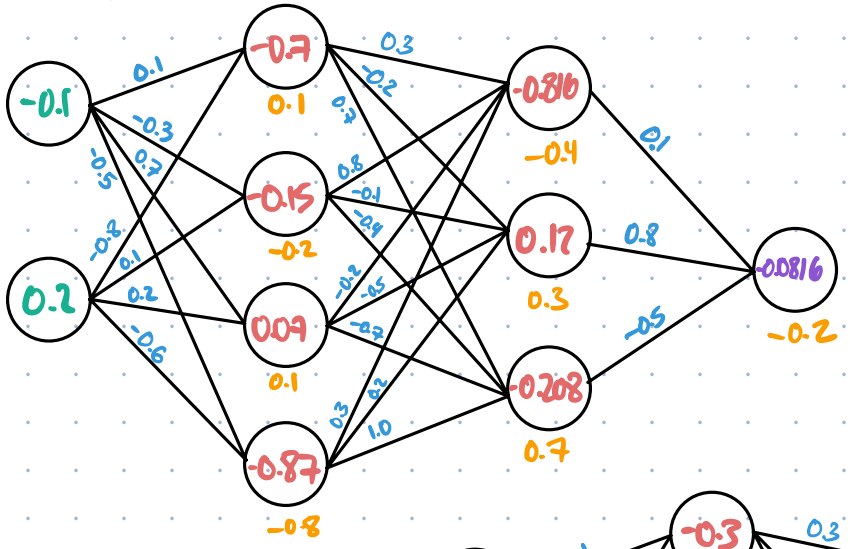
$$= \begin{bmatrix} -0.724 & -0.816 & -1.418 \\ -0.037 & 0.12 & -0.304 \\ 0.598 & -0.208 & -1.464 \end{bmatrix}$$

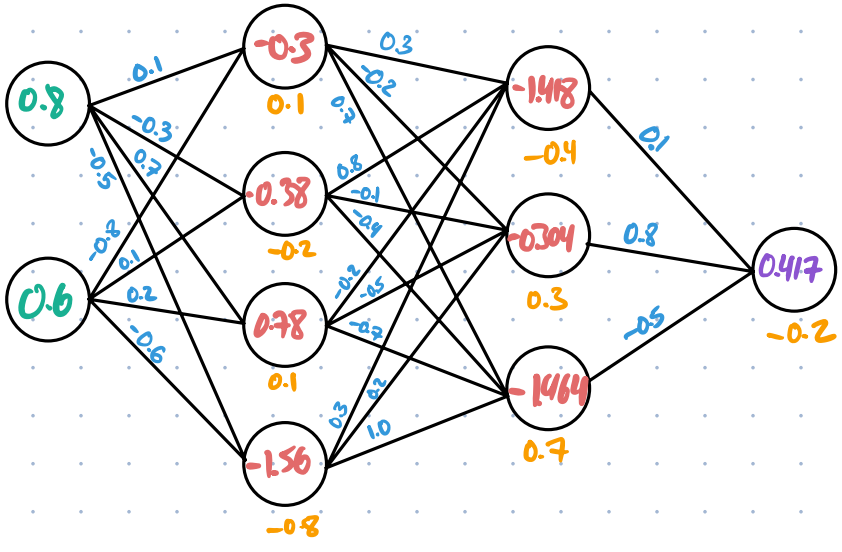
$$\begin{bmatrix} 0.1 & 0.8 & -0.5 \end{bmatrix} \begin{bmatrix} -0.724 & -0.816 & -1.418 \\ -0.037 & 0.12 & -0.304 \\ 0.598 & -0.208 & -1.464 \end{bmatrix} + \begin{bmatrix} -0.2 & -0.2 & -0.2 \end{bmatrix} = \begin{bmatrix} -0.601 & -0.0816 & 0.147 \end{bmatrix}$$



(ignoring activation functions!)







Three inputs fed forward all in one calculation!

Could pass all 60,000 MNIST digits through in one calculation!

* Let's update our code to support batching.

* Let's update our code to support batching.

* Big speed update!

* Demo: - Training on MNIST in browser

- Use in our new classes to predict digits