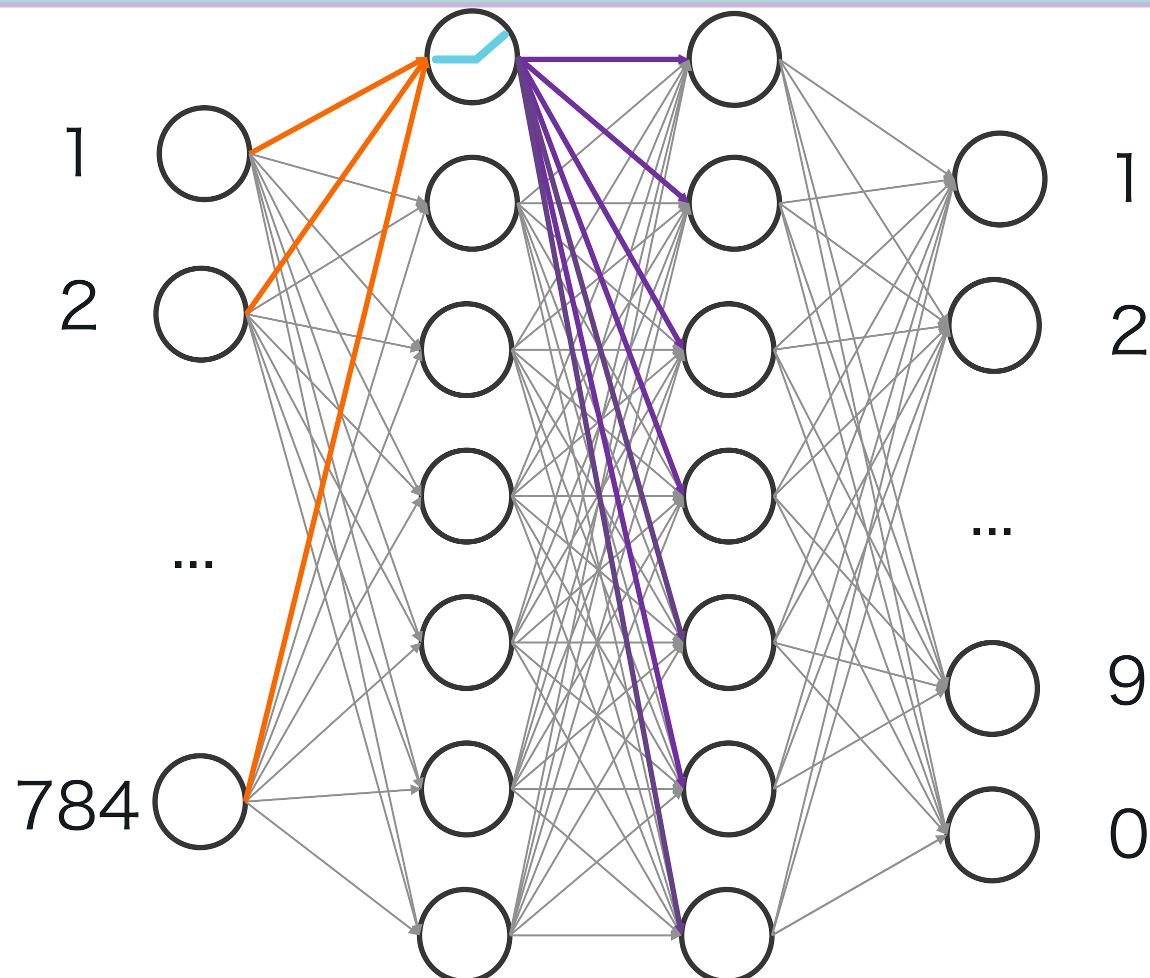


NEURAL NETWORKS

ECS170 Spring 2018
Josh McCoy, @deftjams

Handwritten Digits

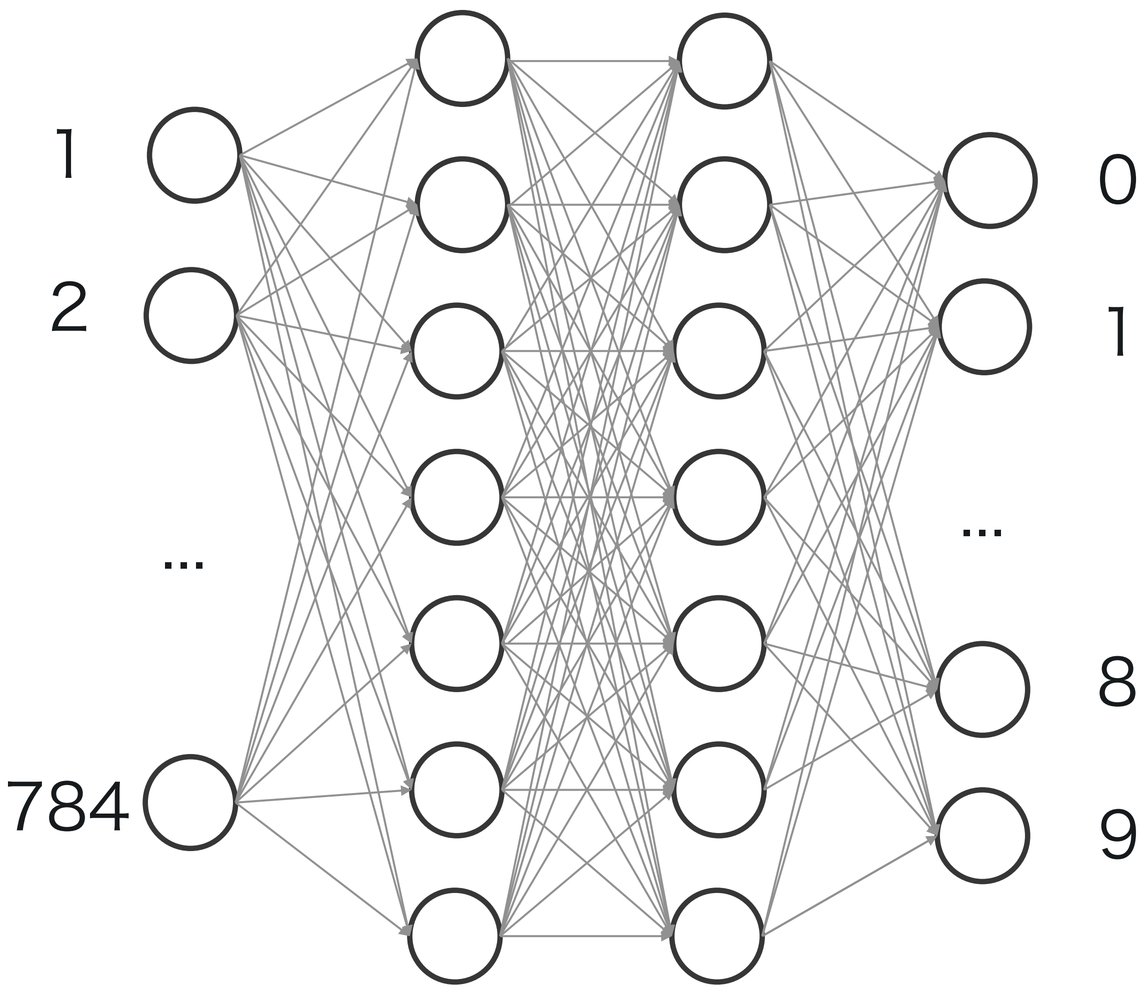
Can you write a program that takes a 28x28 pixel image that recognizes 2s?

0 is white, 1 is black.

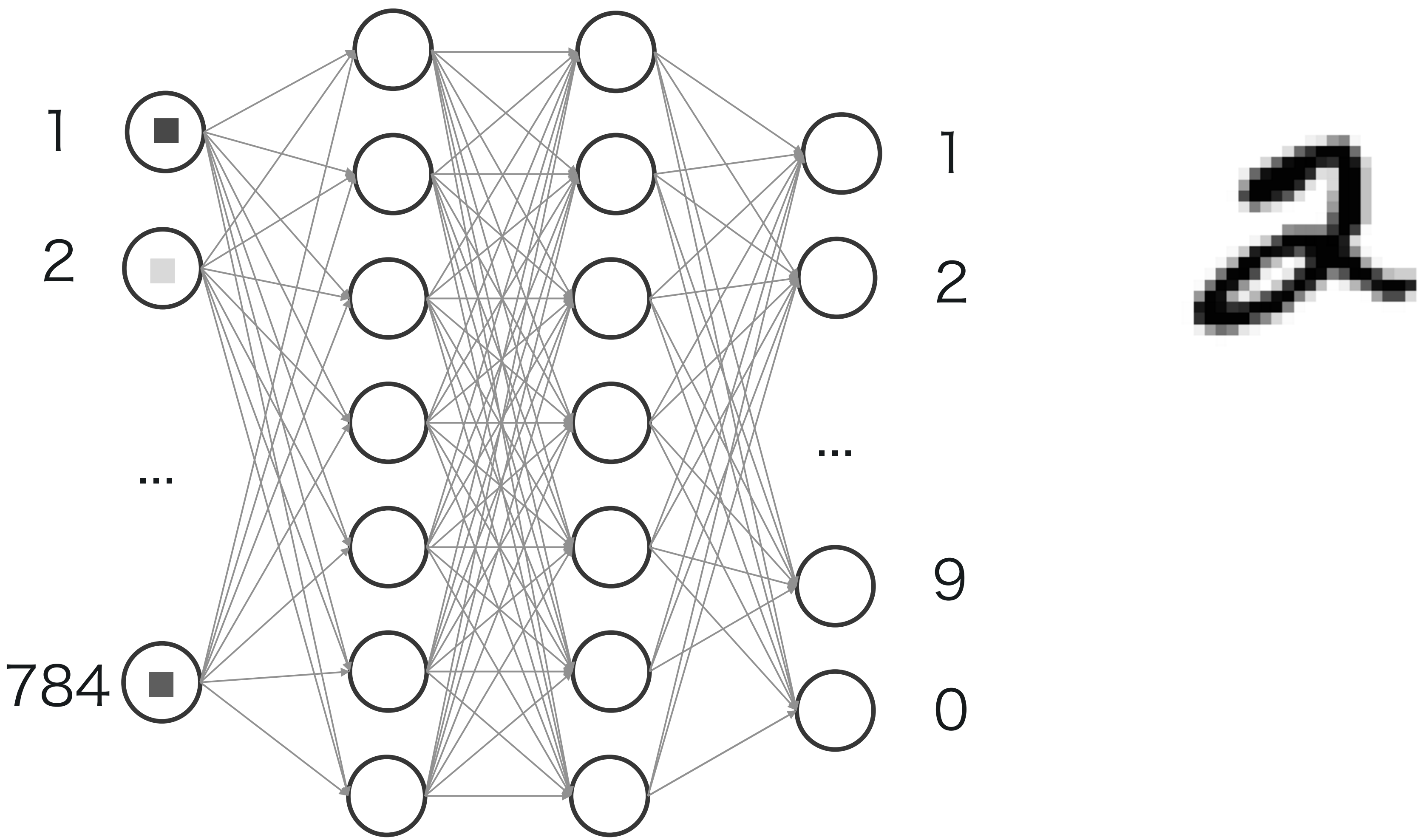
How about a program that recognizes all of these 2s?

Why is this task so hard?

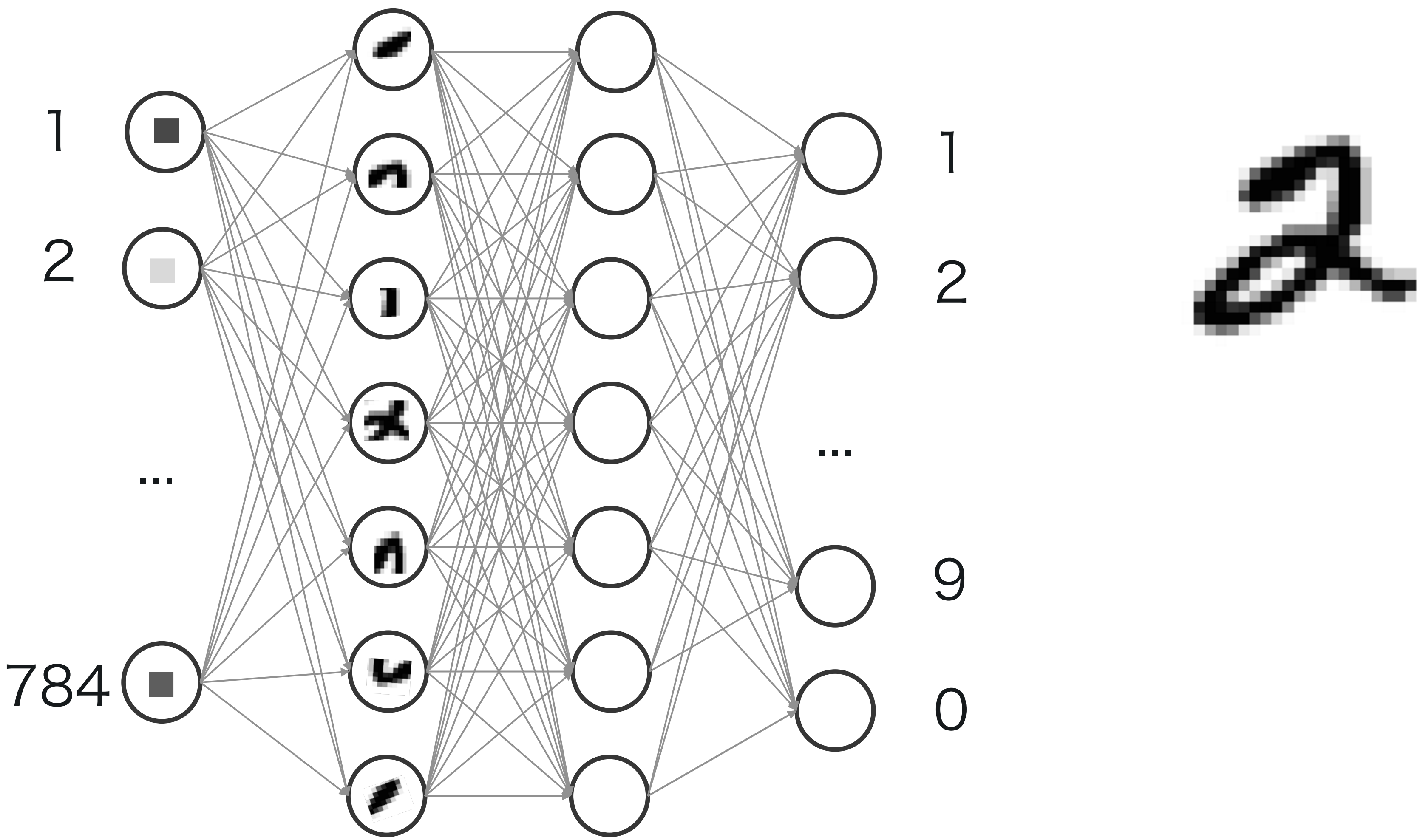
Multi-Layer Perceptrons



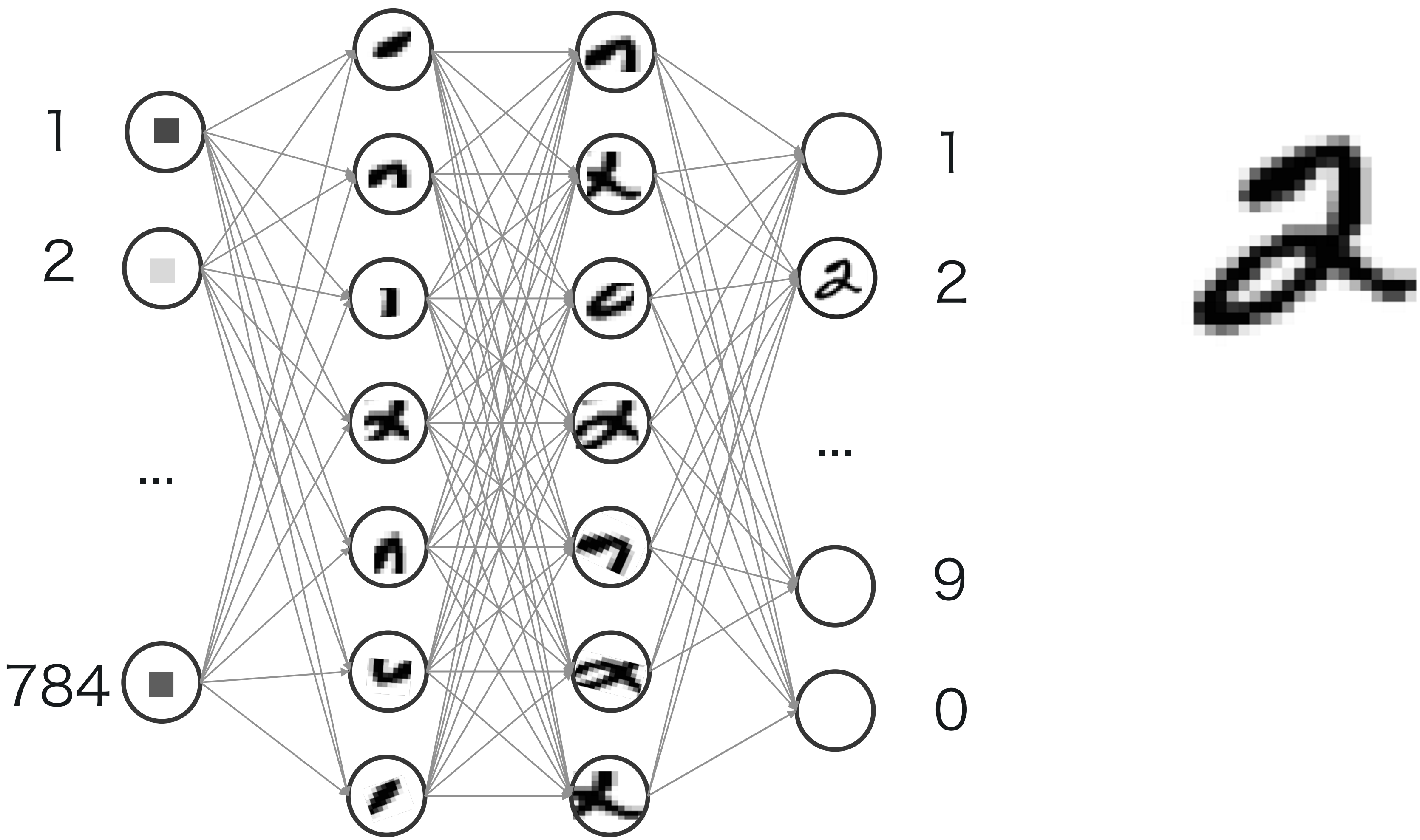
Multi-Layer Perceptrons



Multi-Layer Perceptrons

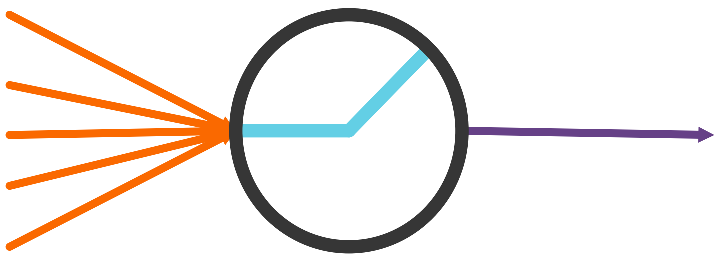


Multi-Layer Perceptrons

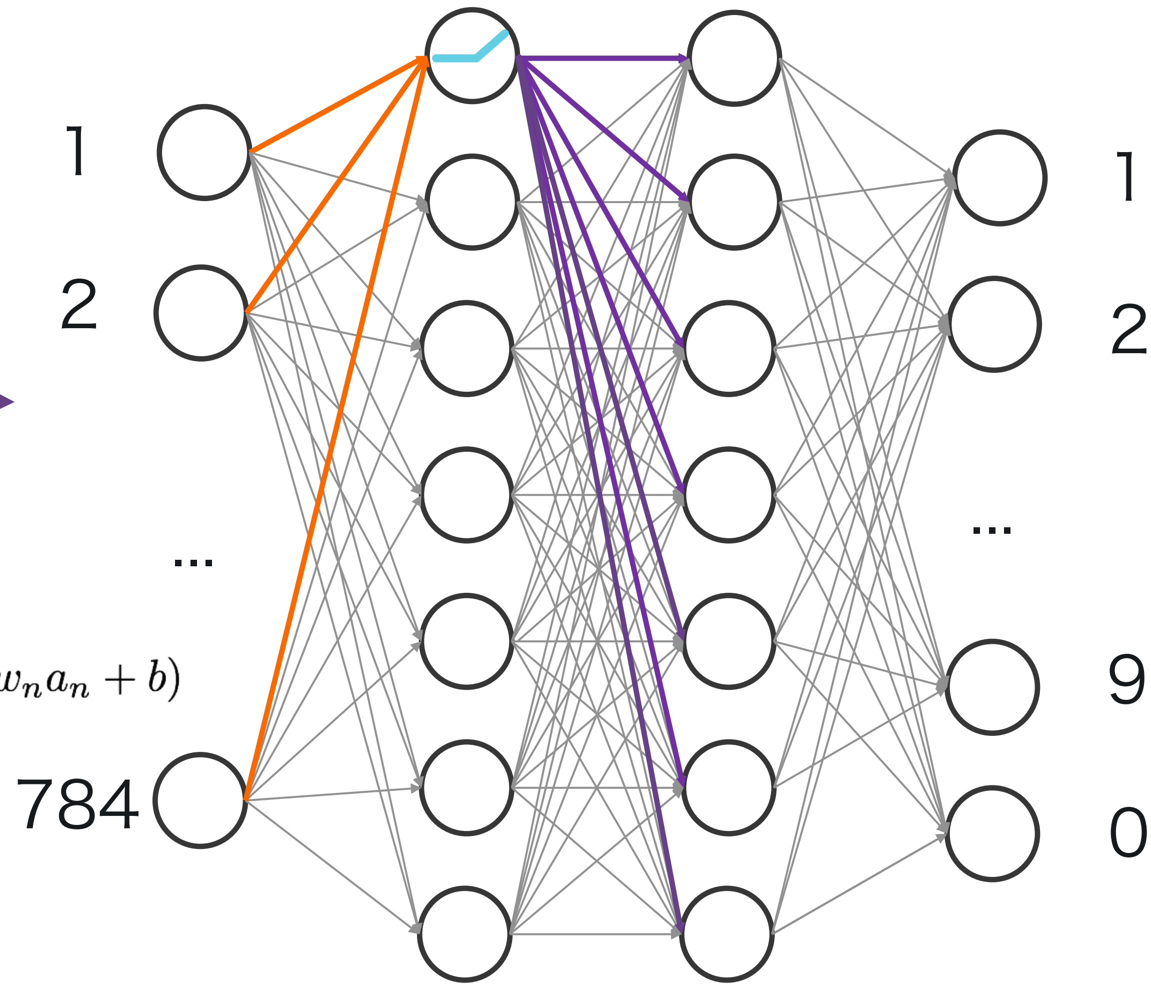


Nodes are Perceptrons

activation = g(Sum of weights + bias)

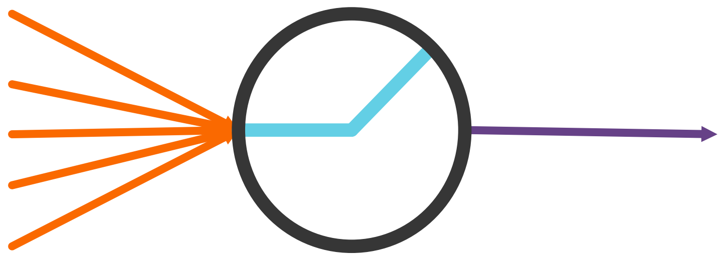


$$a^L = g(w_1 a_1 + w_2 a_2 + w_3 a_3 + \cdots + w_n a_n + b)$$



Nodes are Perceptrons

activation = g(Sum of weights + bias)

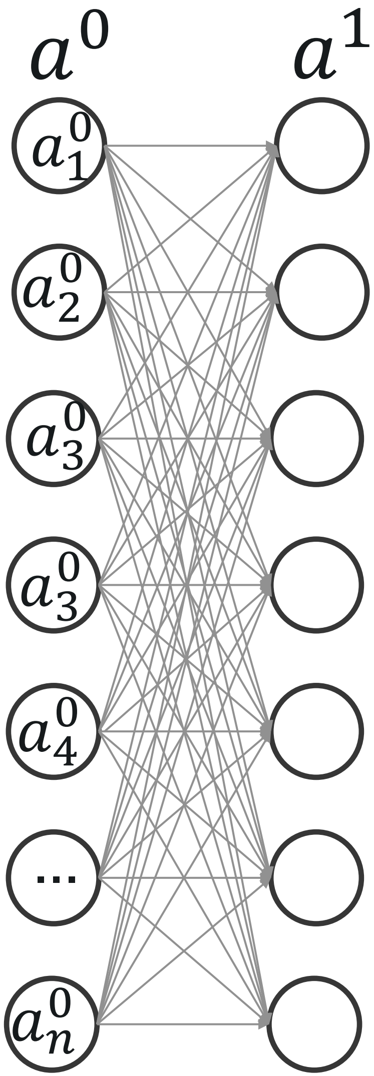


$$a^L = g(w_1 a_1 + w_2 a_2 + w_3 a_3 + \cdots + w_n a_n + b)$$

$$a^L = g(w_1^{L-1} a_1^{L-1} + w_2^{L-1} a_2^{L-1} + w_3^{L-1} a_3^{L-1} + \cdots + w_n^{L-1} a_n^{L-1} + b)$$

$$a^L = g\left(\sum_{j=1}^{n_{L-1}} w_j^{L-1} a_j^{L-1} + b\right)$$

Maxtrix Representation



$$a^1 = g\left(Wa^0 + b\right)$$

$$a^1 = g\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} \begin{bmatrix} a_1^0 \\ a_1^0 \\ \vdots \\ a_n^0 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right)$$

Learning

- Adjusting weights and biases
- Back propagation guided by gradient descent
  - Error propagation.
- Sum small changes to weights and bias for all examples

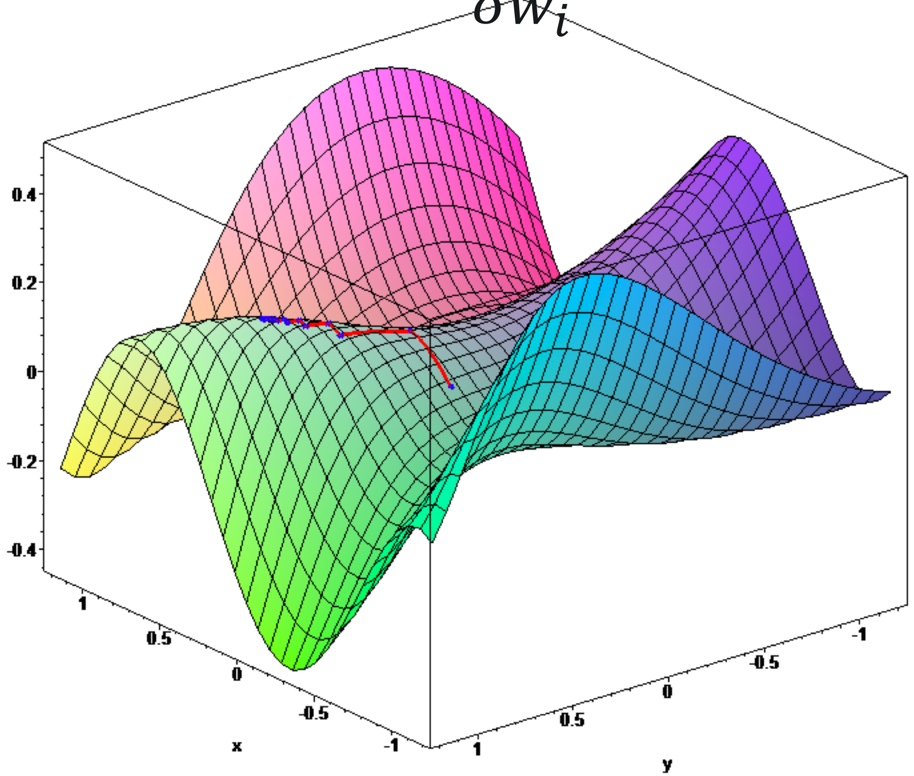
Gradient Descent

w ← any point in parameter space

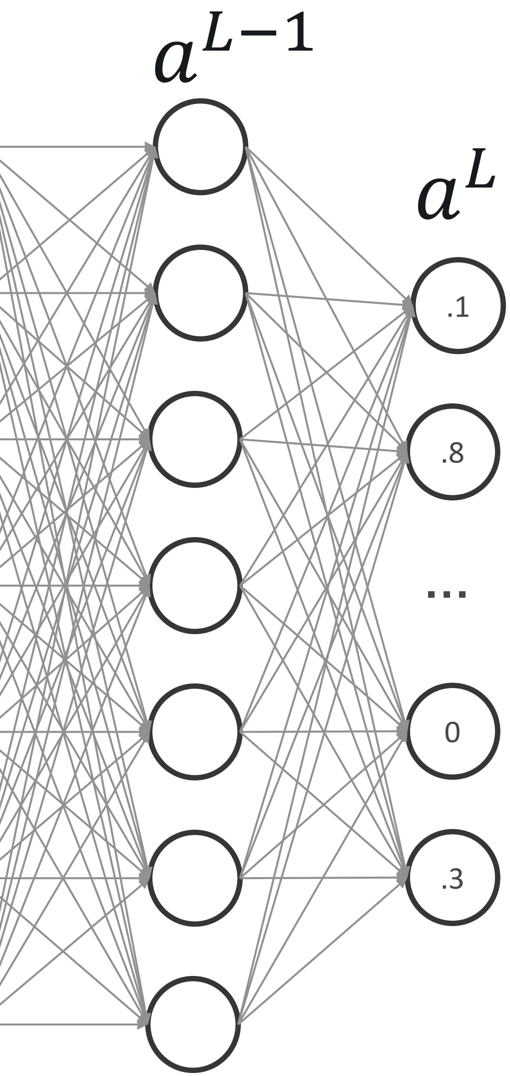
loop until convergence do

for each w_i in w do

$$w_i \leftarrow w_i - \alpha \frac{\delta}{\delta w_i} Loss(w)$$



Loss



$$f(x) = y$$

$$\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$Loss = \sum_{j=0}^{n_L} \left(a_j^L - y_j\right)^2$$

One-Hot Vectors

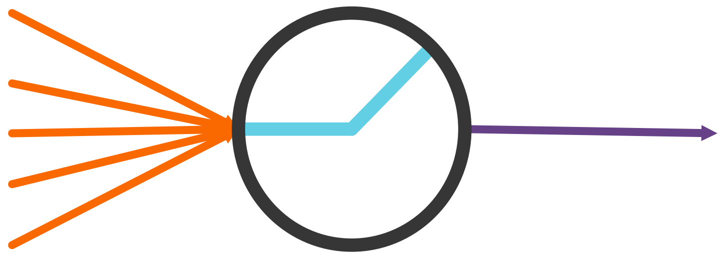
- One valid is hot and the rest are cold.
- Output can only produce a single class.

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Z

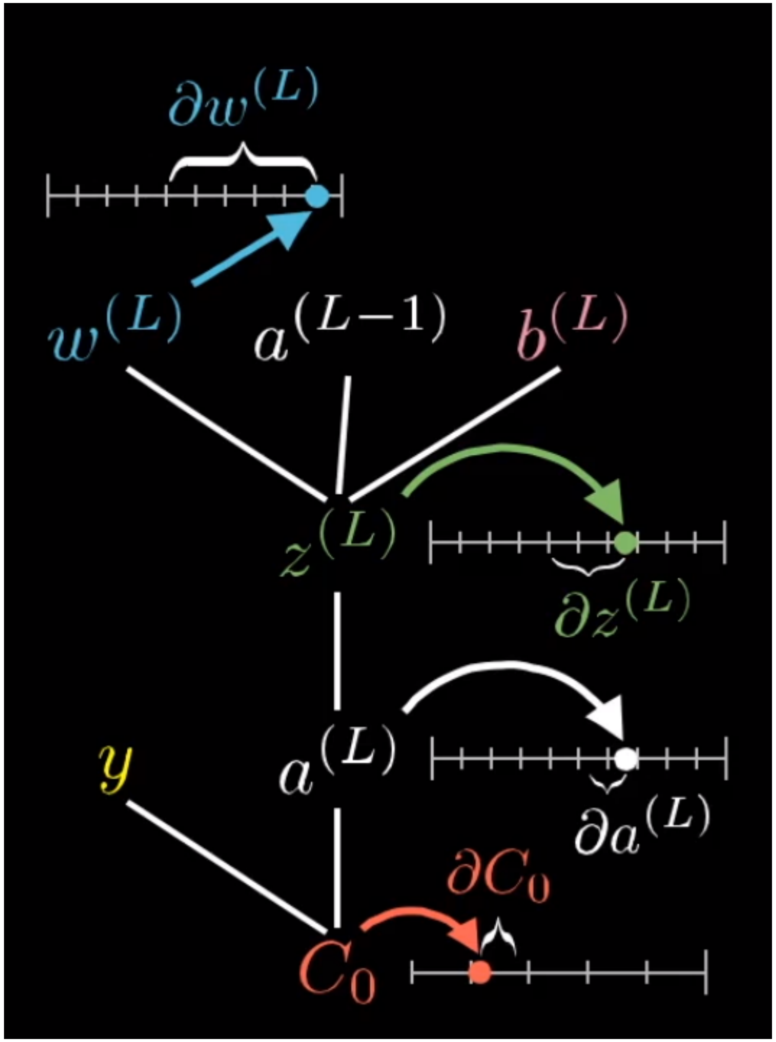
- Useful intermediate representation

$$z_j^L = w_{j0}^L a_0^{L-1} + w_{j1}^L a_1^{L-1} + \cdots + w_{jn}^L a_n^{L-1} + b_j^L$$



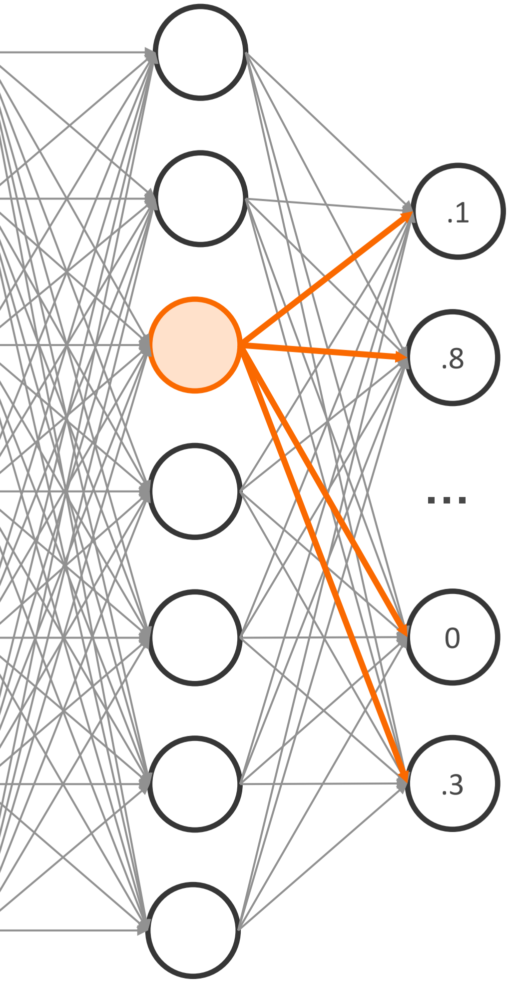
The Chain Rule

$$\frac{\partial Loss}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial Loss}{\partial a^L}$$



https://www.youtube.com/watch?v=tIeHLnjs5U8

Derivative



- The sum of all connected loss:

$$\frac{\partial Loss}{\partial w^L} = \frac{1}{N}\sum_{k=1}^{n} \frac{\partial Loss_k}{\partial w^L}$$

- With chain rule:

$$\frac{\partial Loss}{\partial w^L} = \sum_{j=1}^{n_L} \frac{\partial z_j^L}{\partial a_k^{L-1}} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial Loss}{\partial a_j^L}$$

Updating Bias

- Change is bias is linear the change in Loss

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = 1 \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

All Examples: Avg. Changes to Weights

- To approximate the function with steepest descent, we average the changes to the weights/biases for all examples.

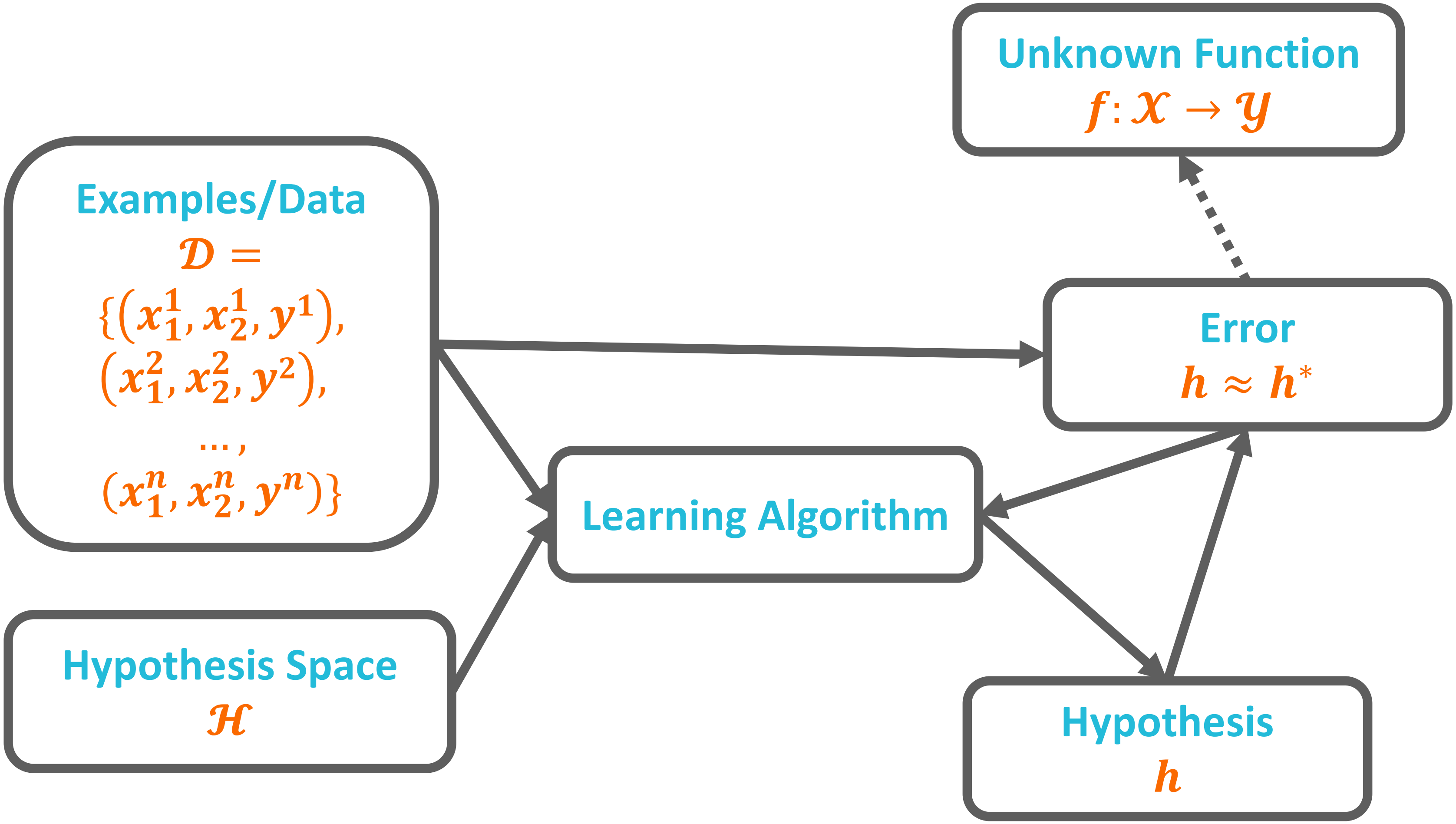
- This uses a lot of resources before a single value in the network is changed.

- How do we fix this?

Mini-Batches

- Big and impractical
- Mini-batches constructed from random examples
  - Ok approximations of the full update function
  - Drunken downhill exploration.

Process


Unknown Function
$f: \mathcal{X} \rightarrow \mathcal{Y}$
Examples/Data
$\mathcal{D} = \{(x_1^1, x_2^1, y^1), (x_1^2, x_2^2, y^2), \ldots, (x_1^n, x_2^n, y^n)\}$
Error
$h \approx h^*$
Learning Algorithm
Hypothesis Space
$\mathcal{H}$
Hypothesis
h

Useful Links

- https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
- http://neuralnetworksanddeeplearning.com/index.html
  - https://github.com/mnielsen/neural-networks-and-deep-learning
- https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/neural_network_raw.ipynb