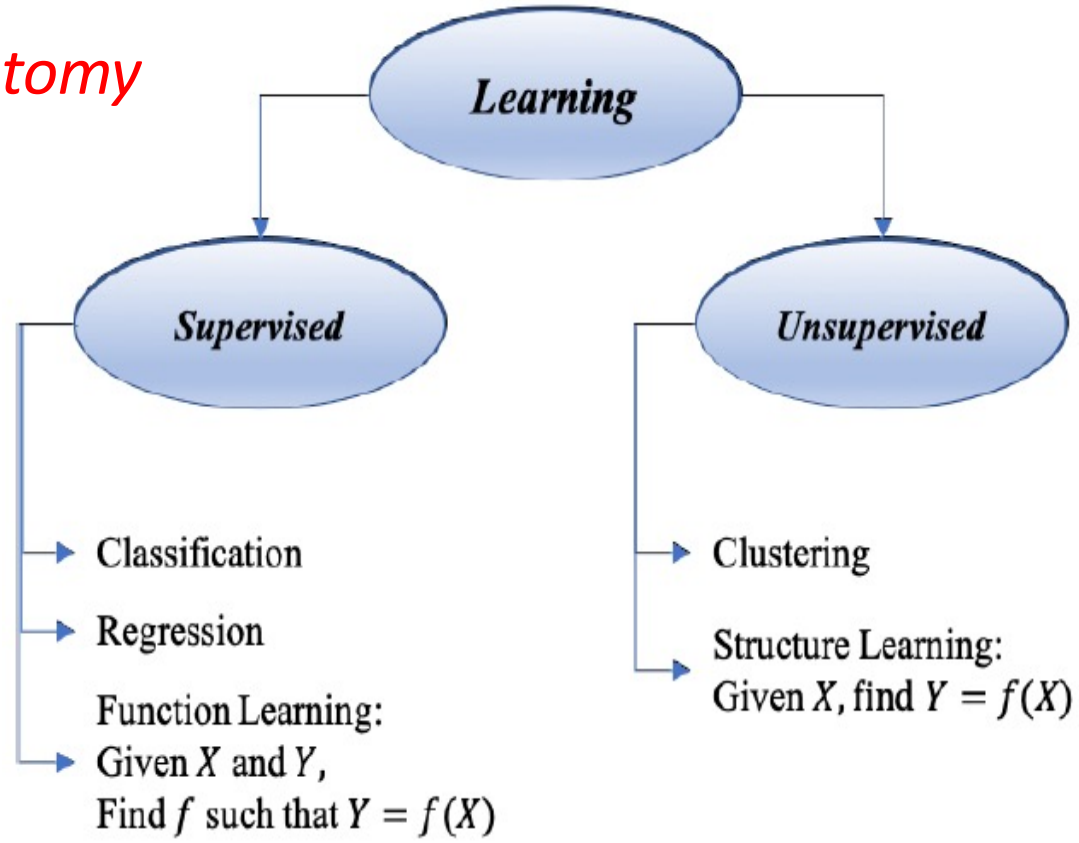
Learning Dichotomy

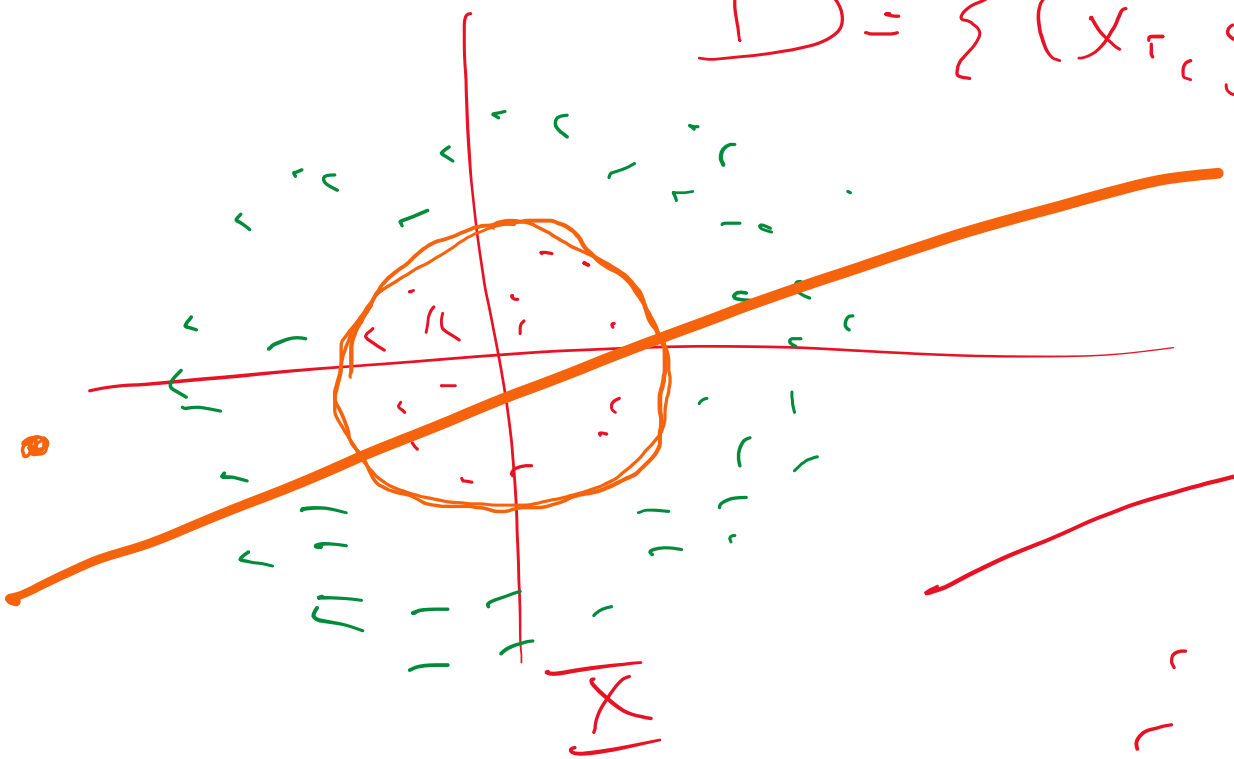


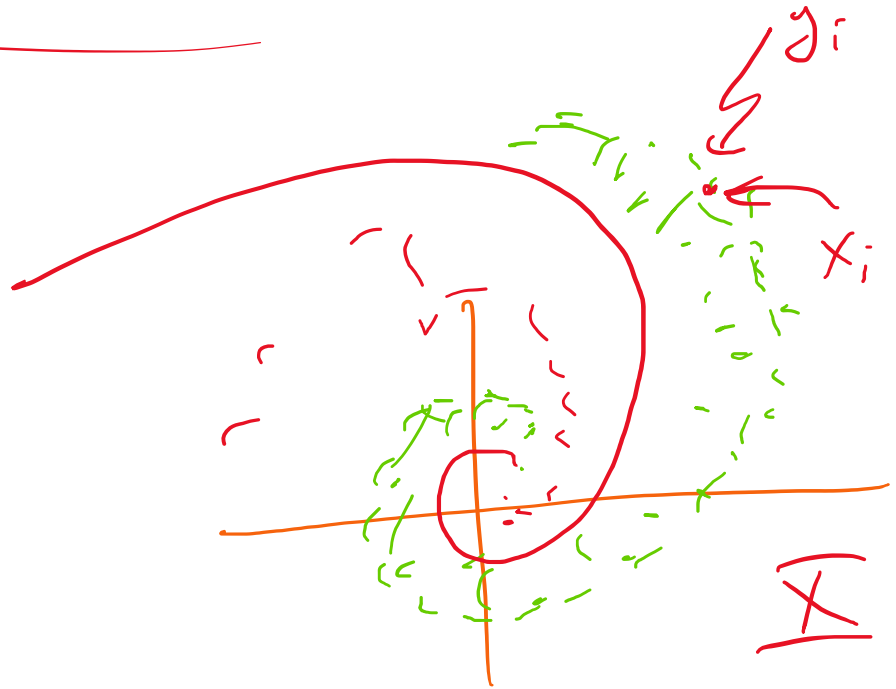
Labelled data
Learn a function

Pattern
Structure

Well…at least one more
- re-enforcement learning

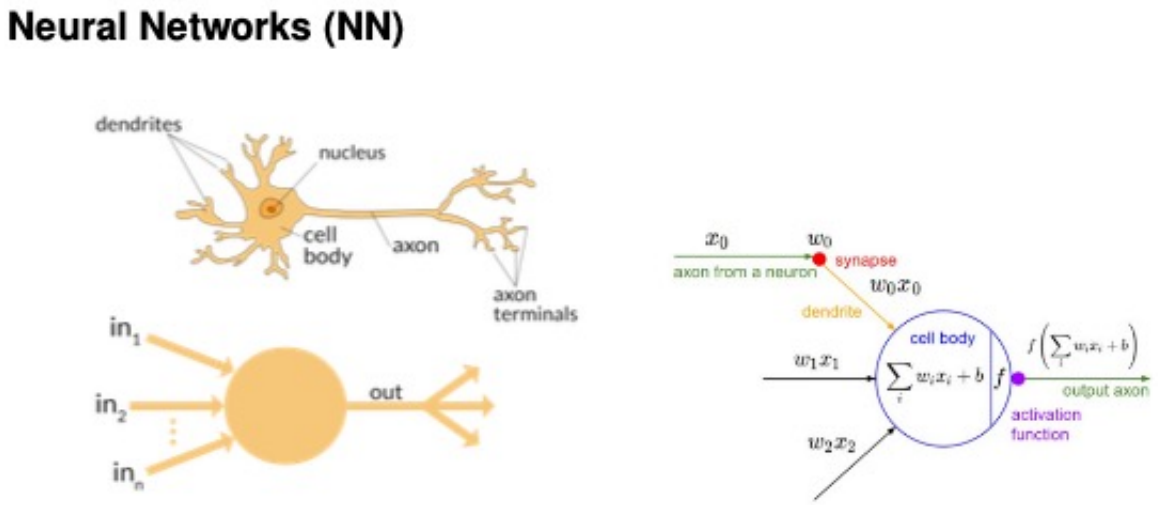
$$D = \{(x_i, y_i)\}$$





And now for something completely different

Artificial Neural Networks



Suddenly ANN is best – why here why now? - I thought it was a crazy idea when I first heard of it.

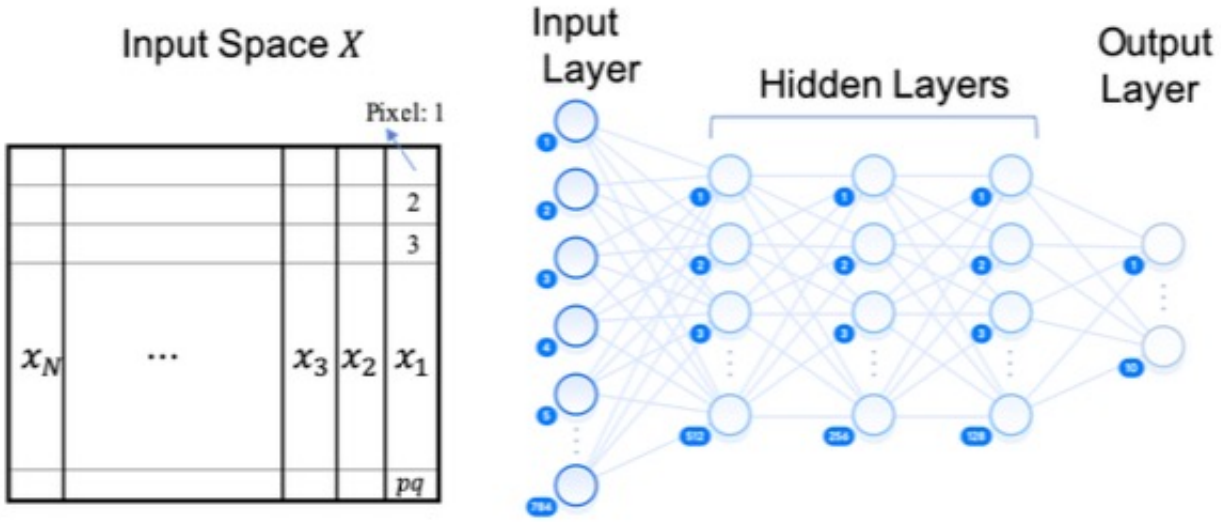
Advancements of technology

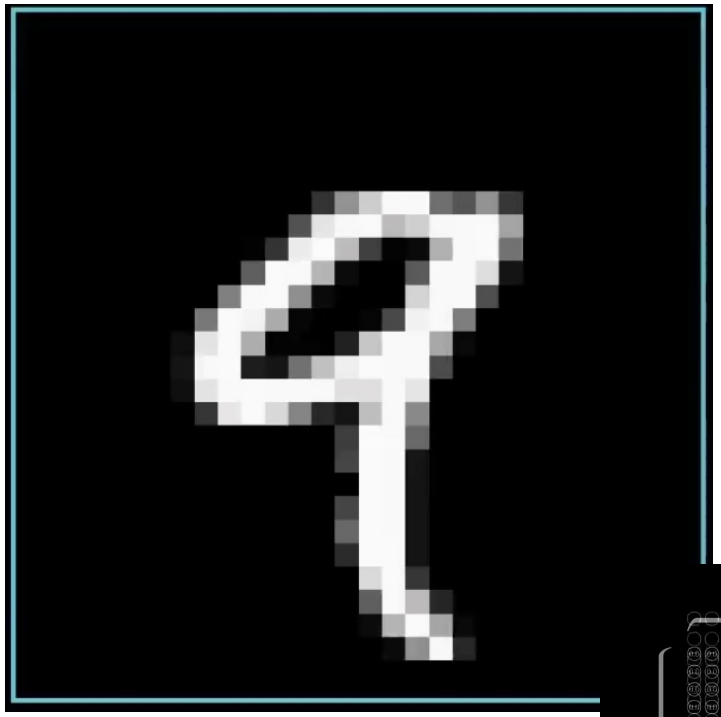
- Stochastic Gradient Descent.
- The computational GPU (Graphic Processing Unites), and the huge evolution of computations speed.

The problem – an example data set USPS Figs –

A supervised learning problem

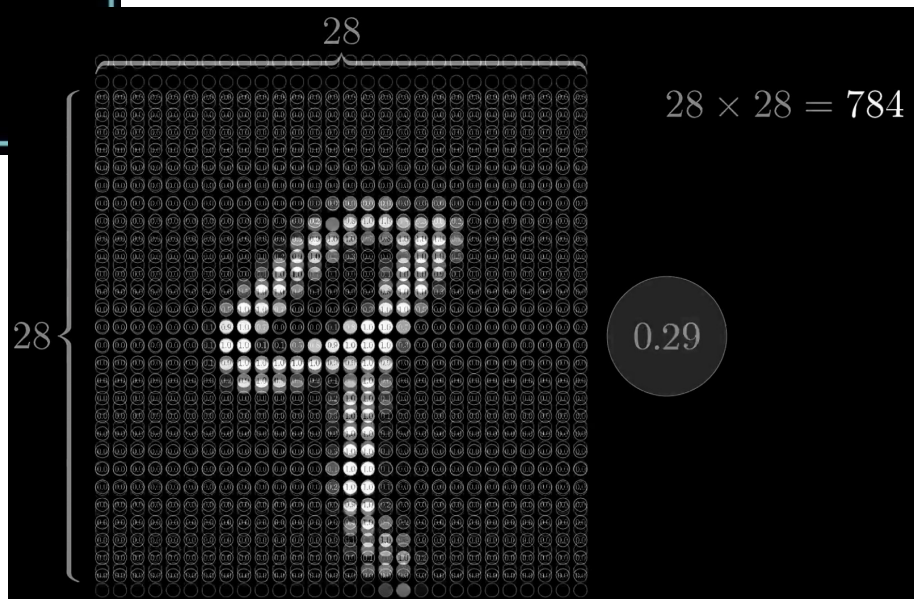
The basis picture of data input into a ANN
-input layer
-hidden inner layers
-output layer



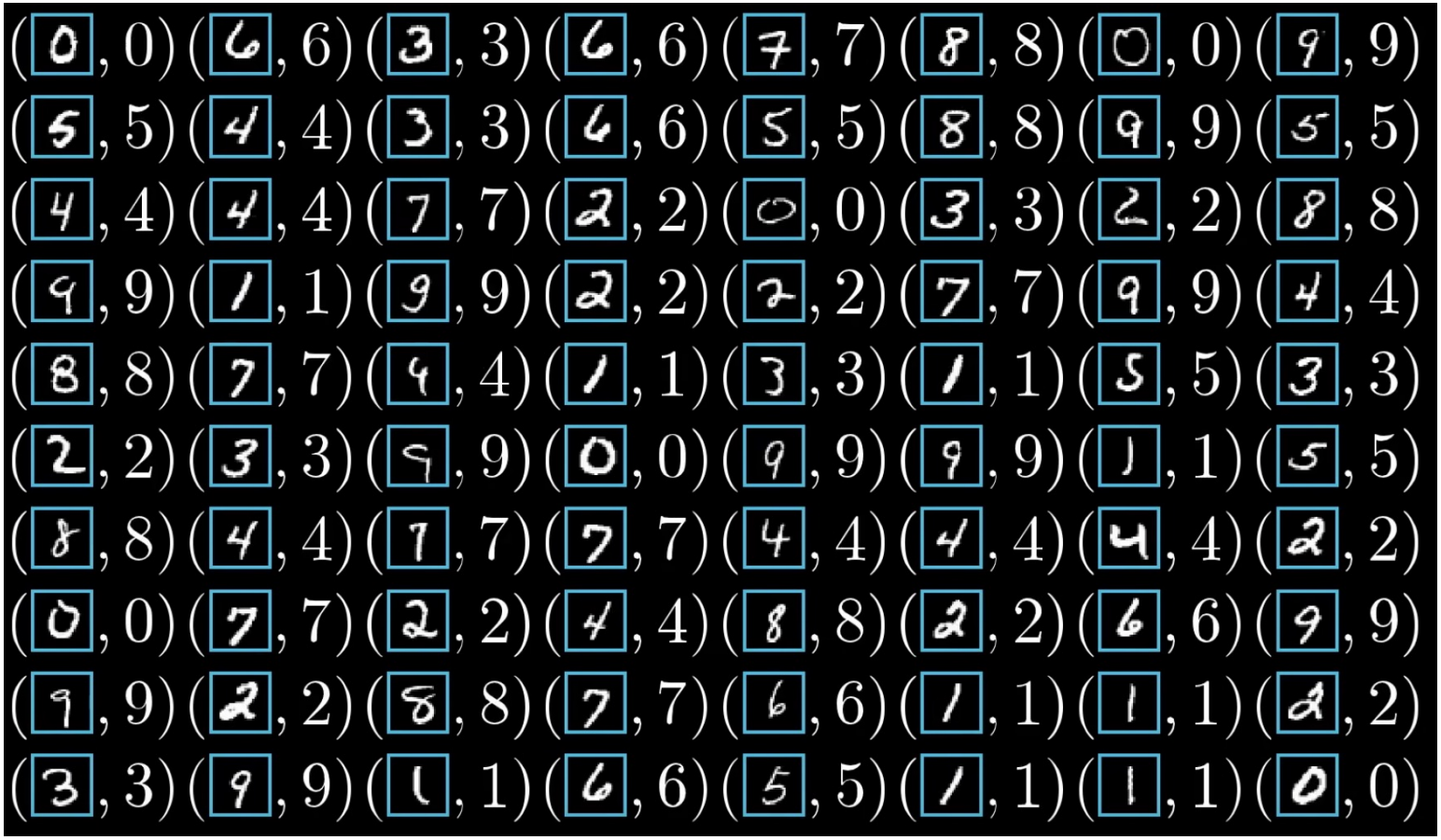


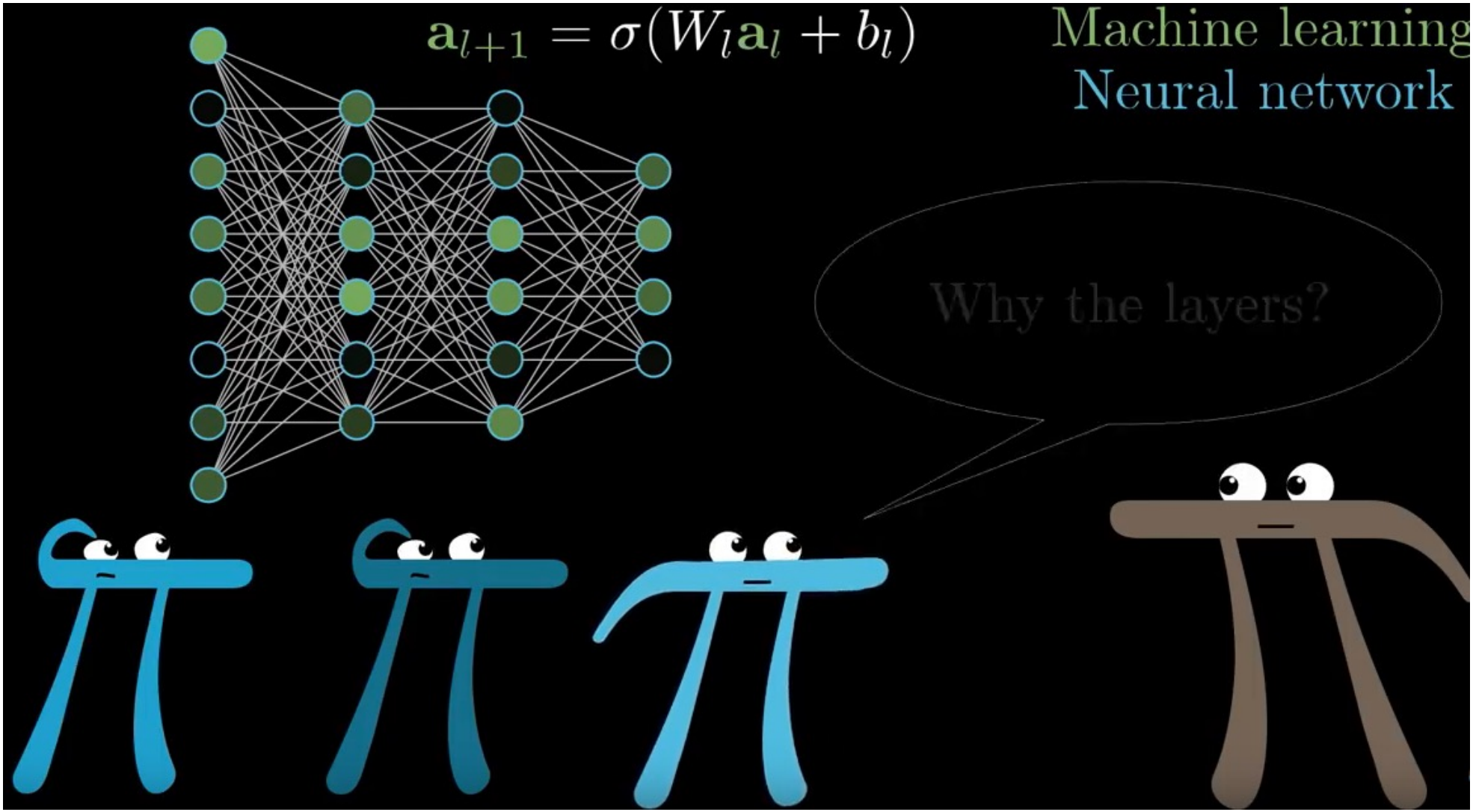
This datum is a 784 numbers between 0 and 1.

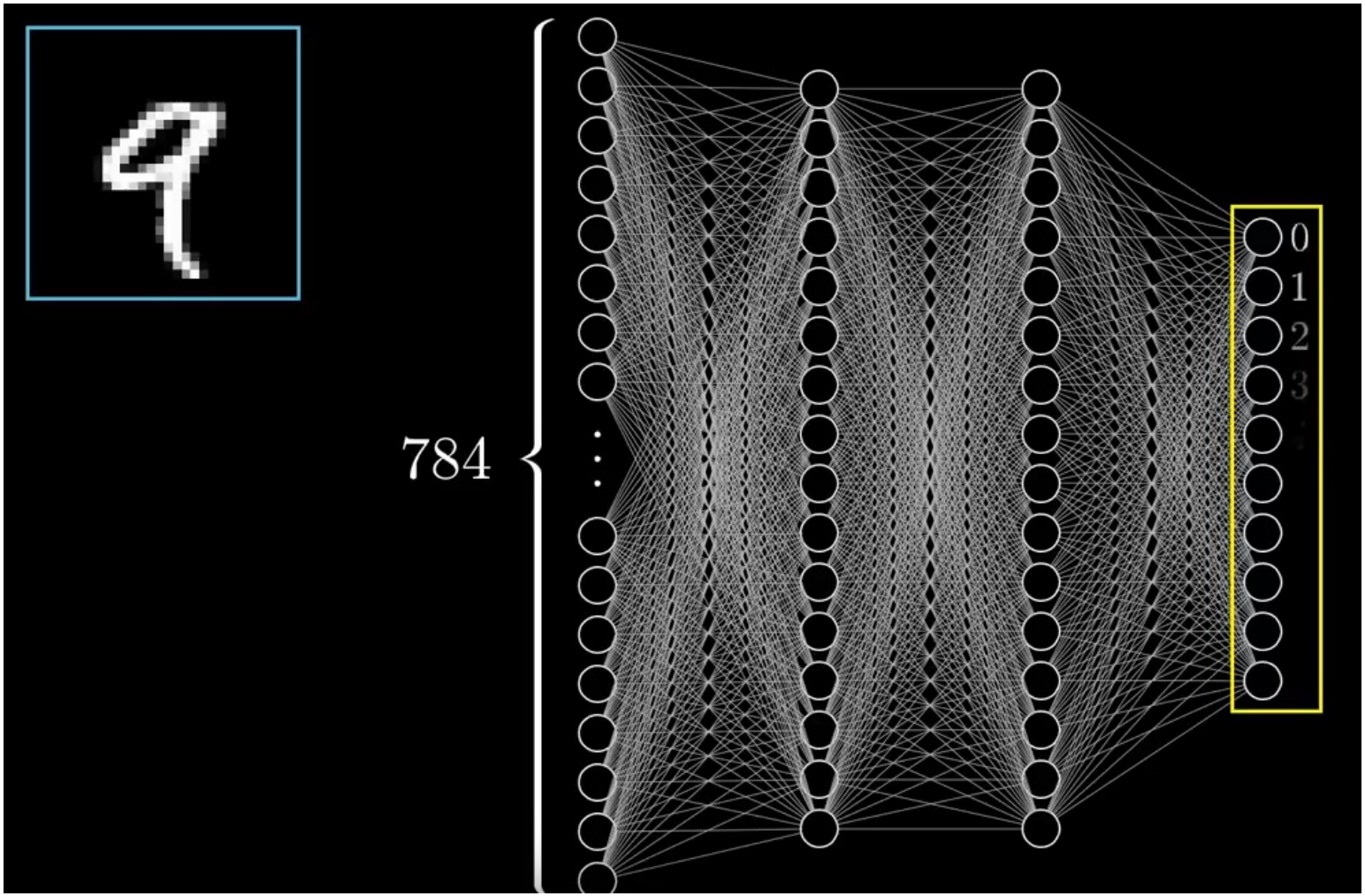
Data {(x_i,y_i)}_i=1..784

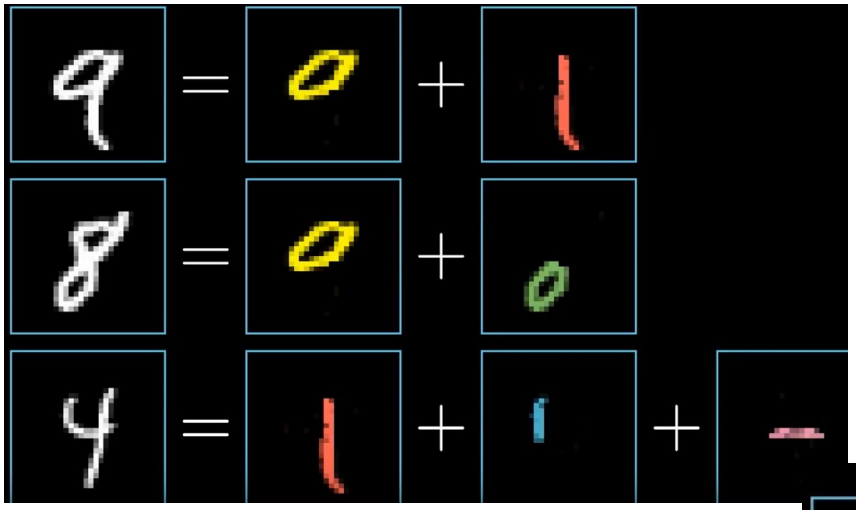


The data is a collection on ordered pairs, vector valued figures together with a symbol

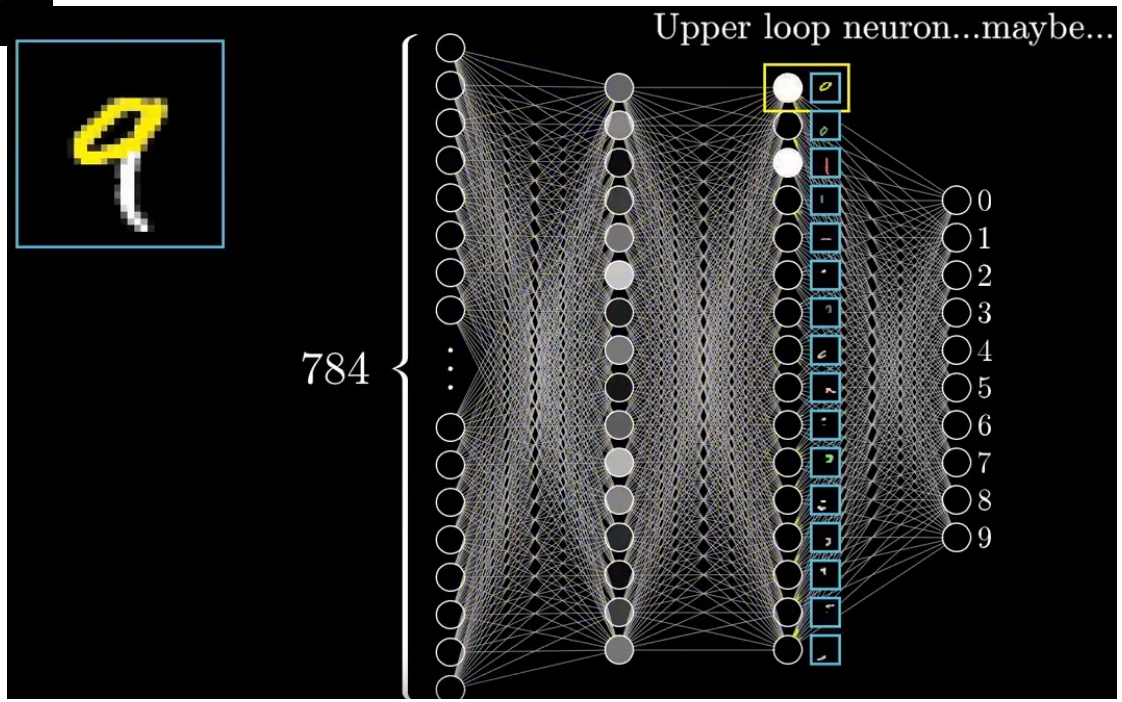


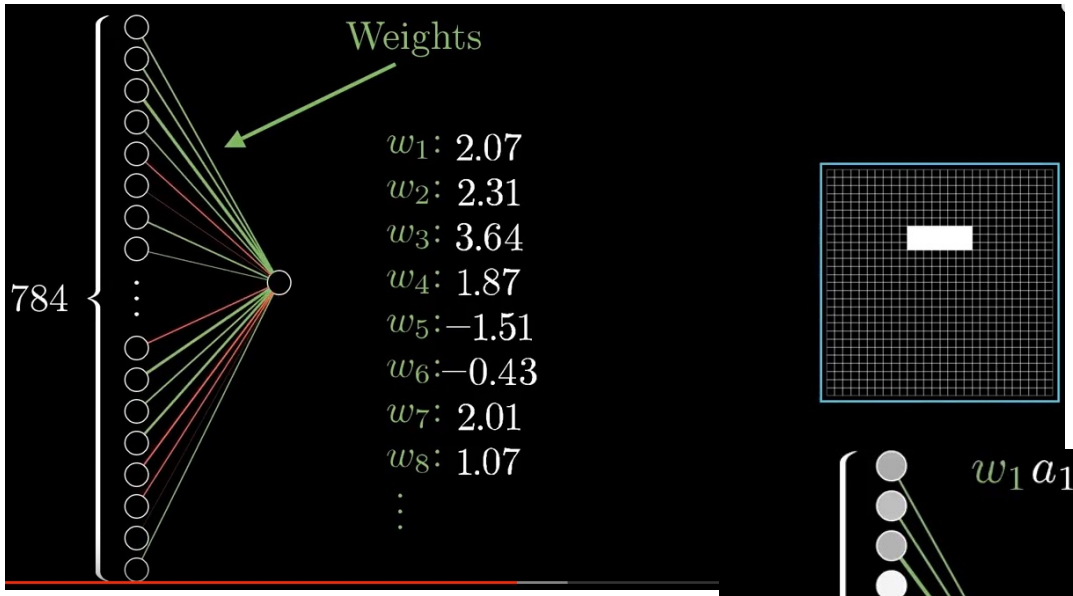

$\mathbf{a}_{l+1} = \sigma(W_l \mathbf{a}_l + b_l)$
Machine learning
Neural network
Why the layers?


784
0
1
2
3



What do you wish in the “neurons”?

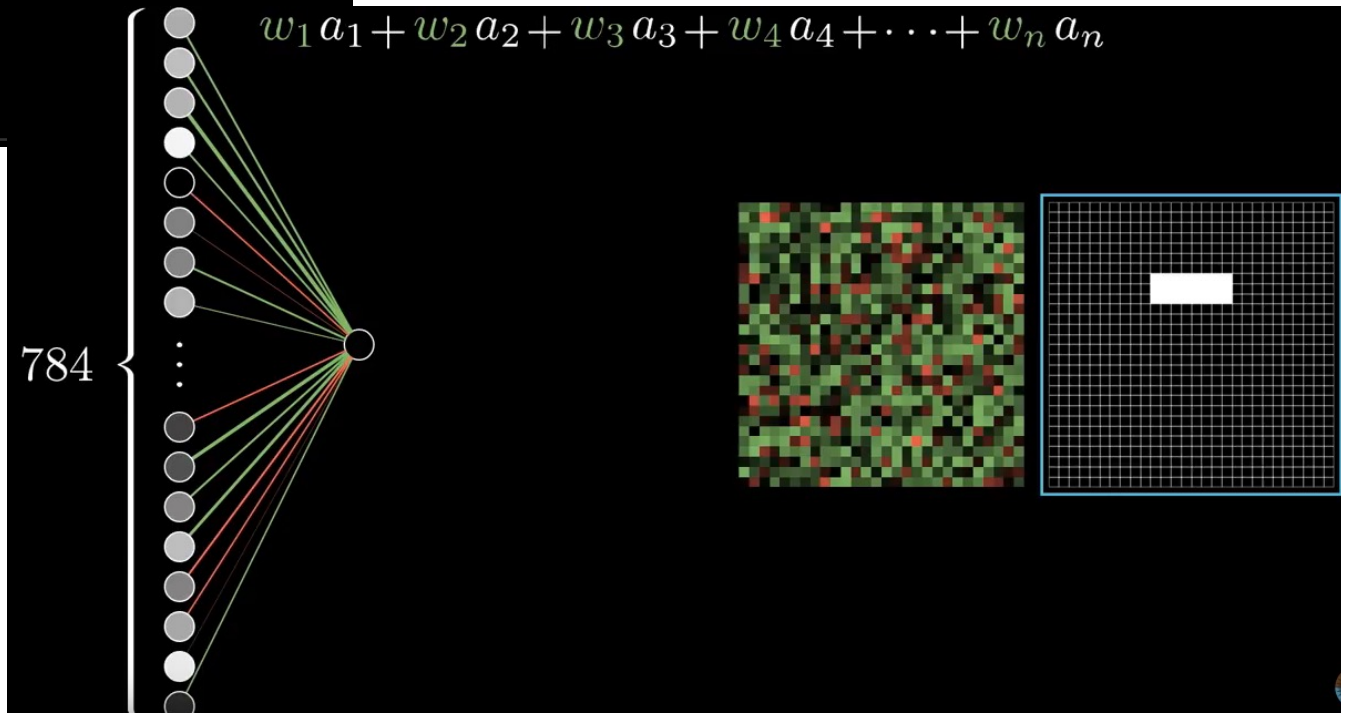




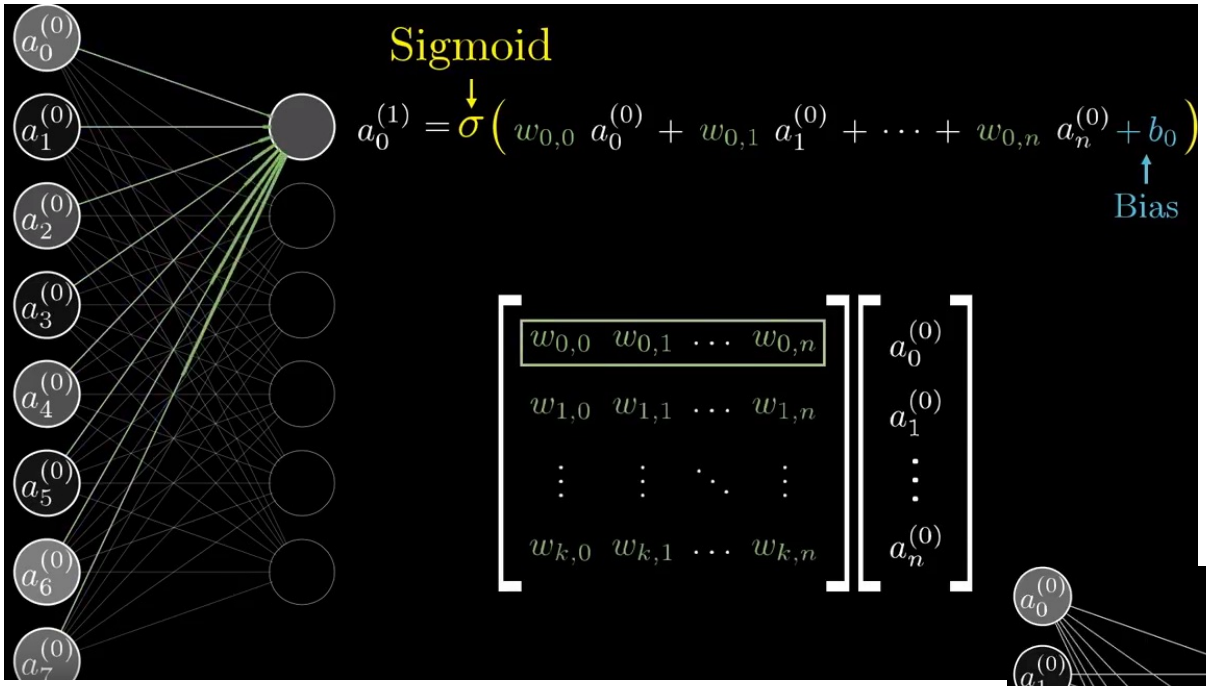
What do you wish in the “neurons”?

Vs

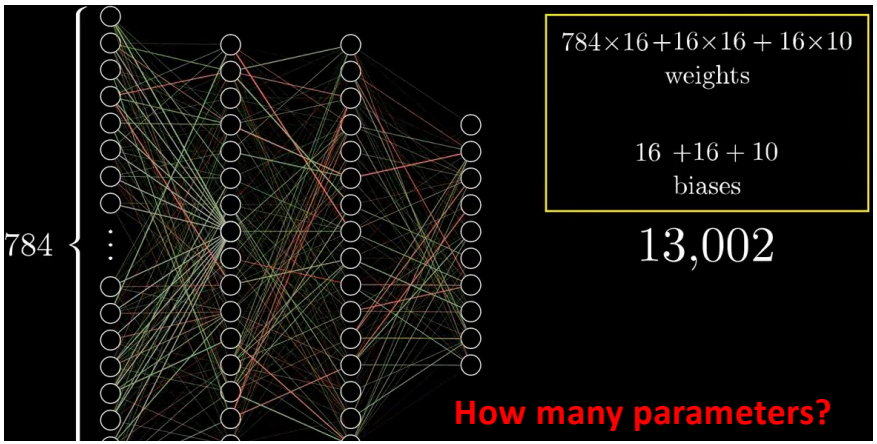
What do we get in the neurons.

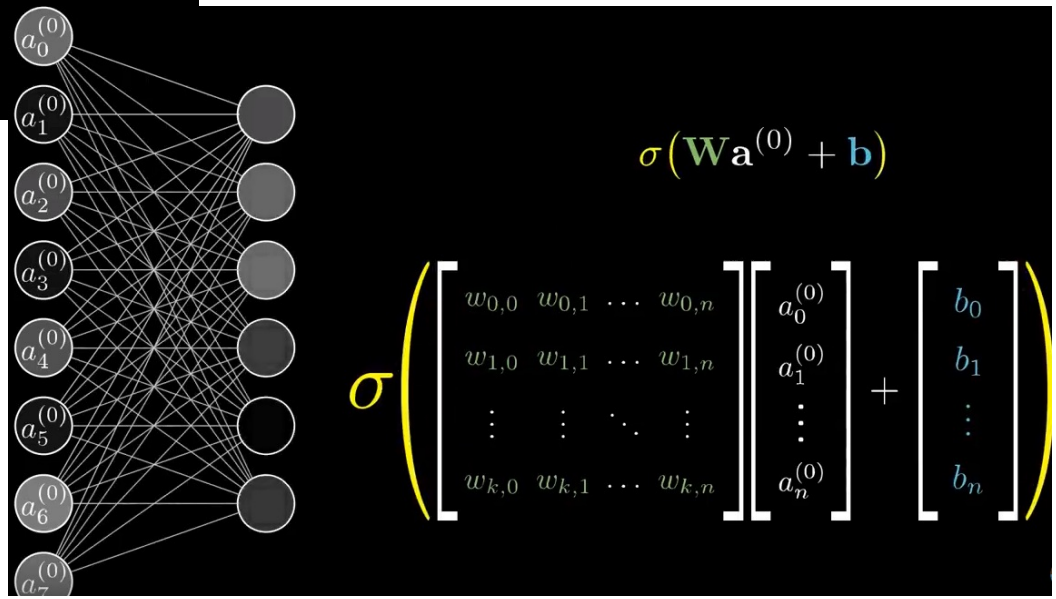


The ANN – Mathematical Process



The matrix form of a layer, and composition





The basic nodal statement.

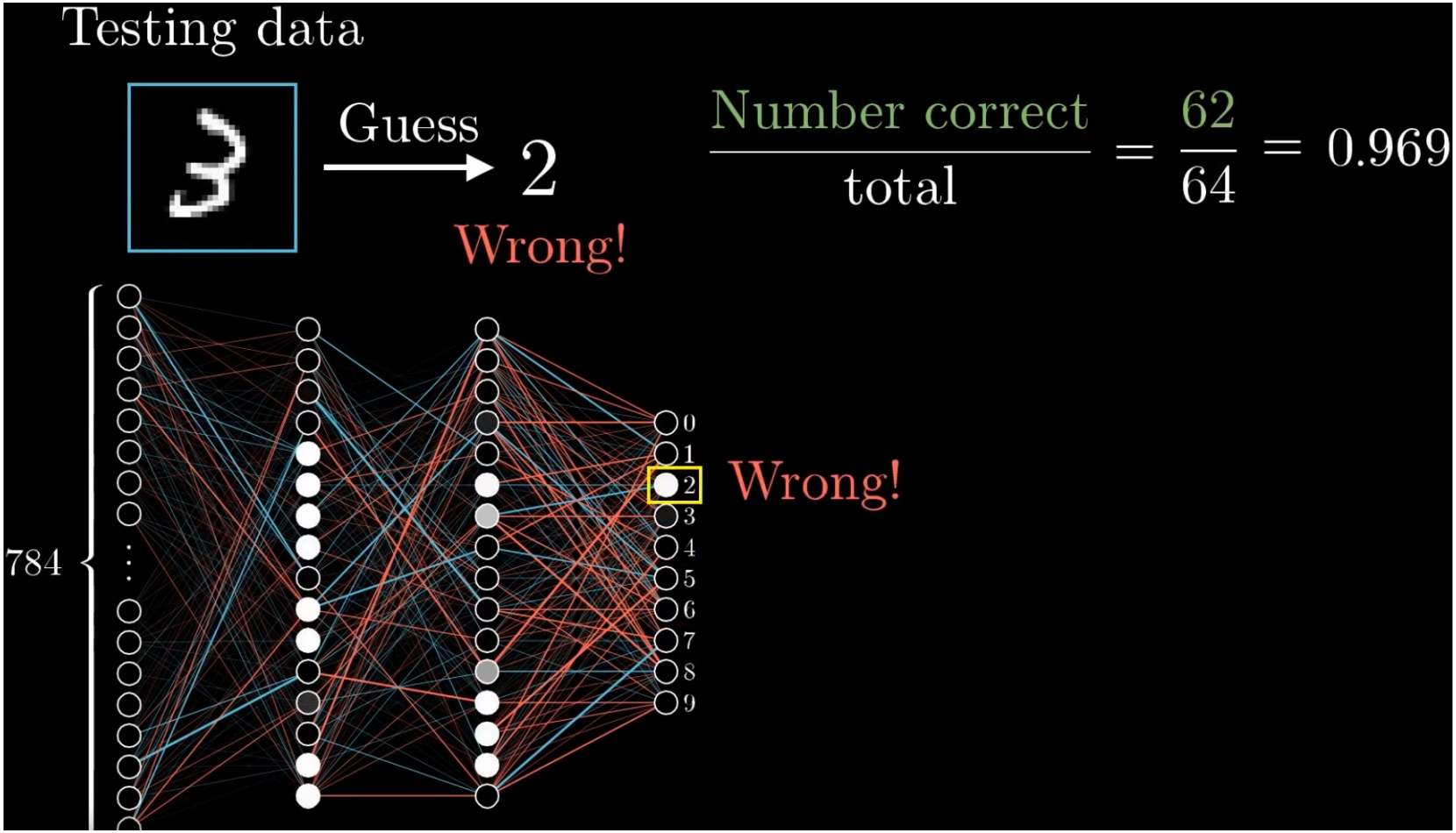
$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

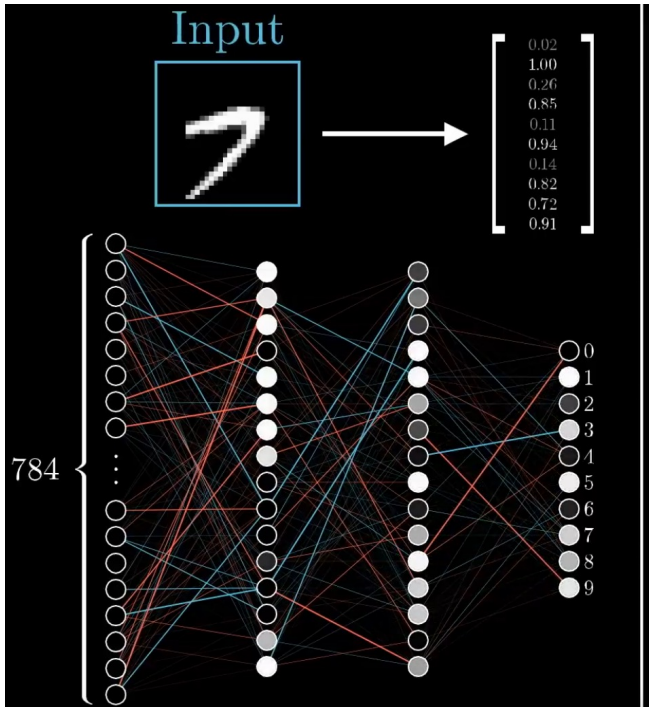
Favorite activation functions

- Linear function: $\sigma(x) = ax + b$, with a and b are constants.
- Sigmoid (Logistic) function: $\sigma(x) = \frac{1}{1+e^{-x}}$.
- Hyperbolic Tangent: $\sigma(x) = \tanh(x)$.
- Rectified Linear Unit (ReLU): $\sigma(x) = \max\{0, x\}$.

What's the cost?
Vs what?

Vs what we want?





Neural network function

Input: 784 numbers (pixels)

Output: 10 numbers

Parameters:

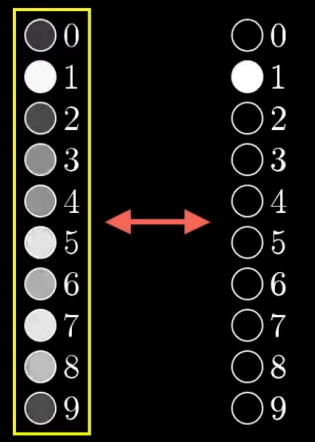
Let's write out the cost function

Average cost of all training data...

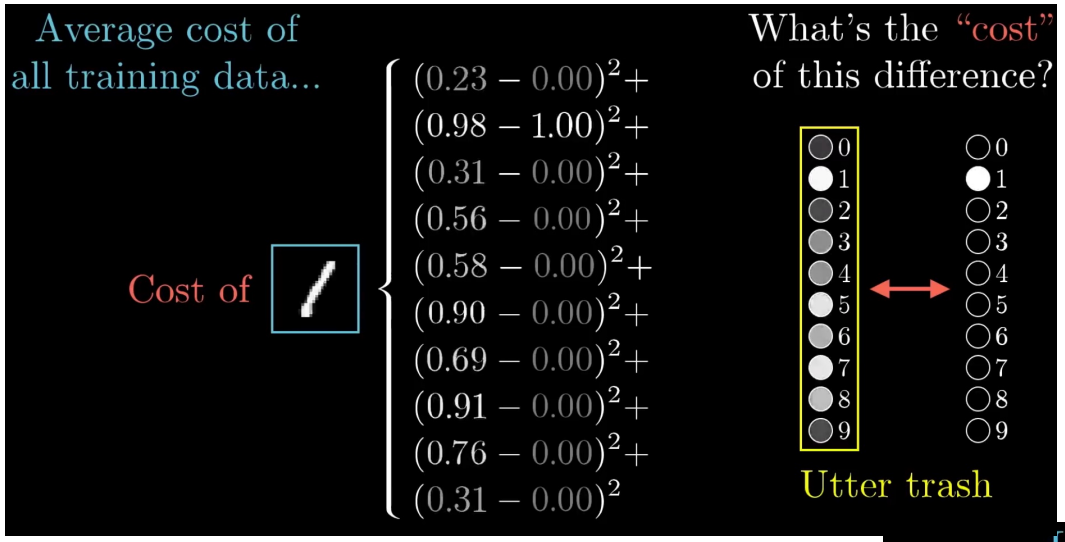
Cost of 

$$
\begin{aligned}
&(0.23 - 0.00)^2 + \\
&(0.98 - 1.00)^2 + \\
&(0.31 - 0.00)^2 + \\
&(0.56 - 0.00)^2 + \\
&(0.58 - 0.00)^2 + \\
&(0.90 - 0.00)^2 + \\
&(0.69 - 0.00)^2 + \\
&(0.91 - 0.00)^2 + \\
&(0.76 - 0.00)^2 + \\
&(0.31 - 0.00)^2
\end{aligned}
$$

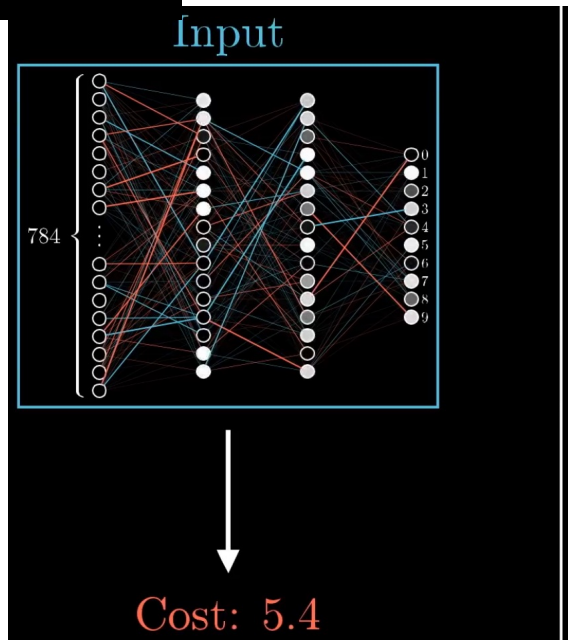
What's the "cost" of this difference?



Utter trash



Let's write out the cost function



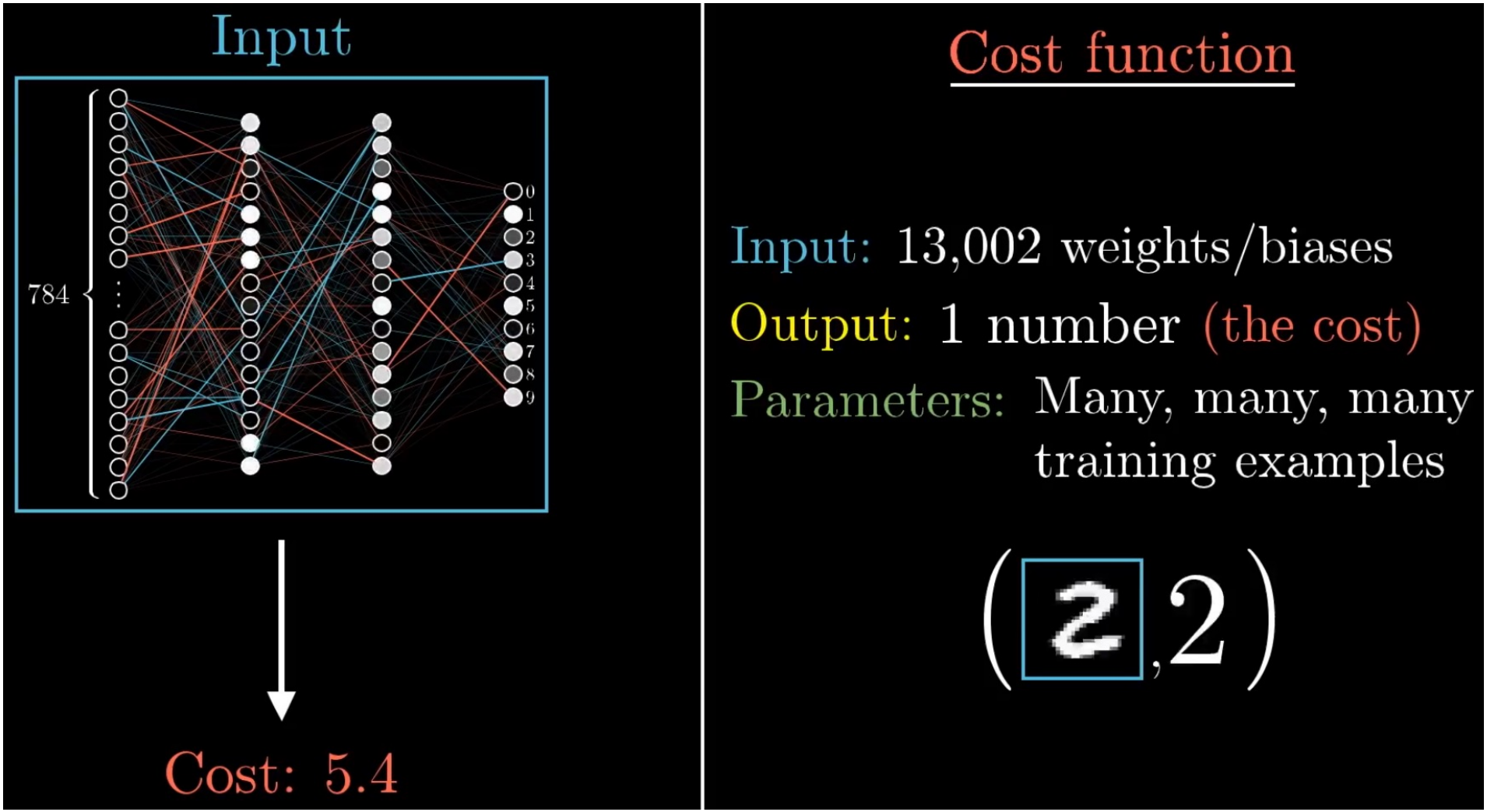
Cost function

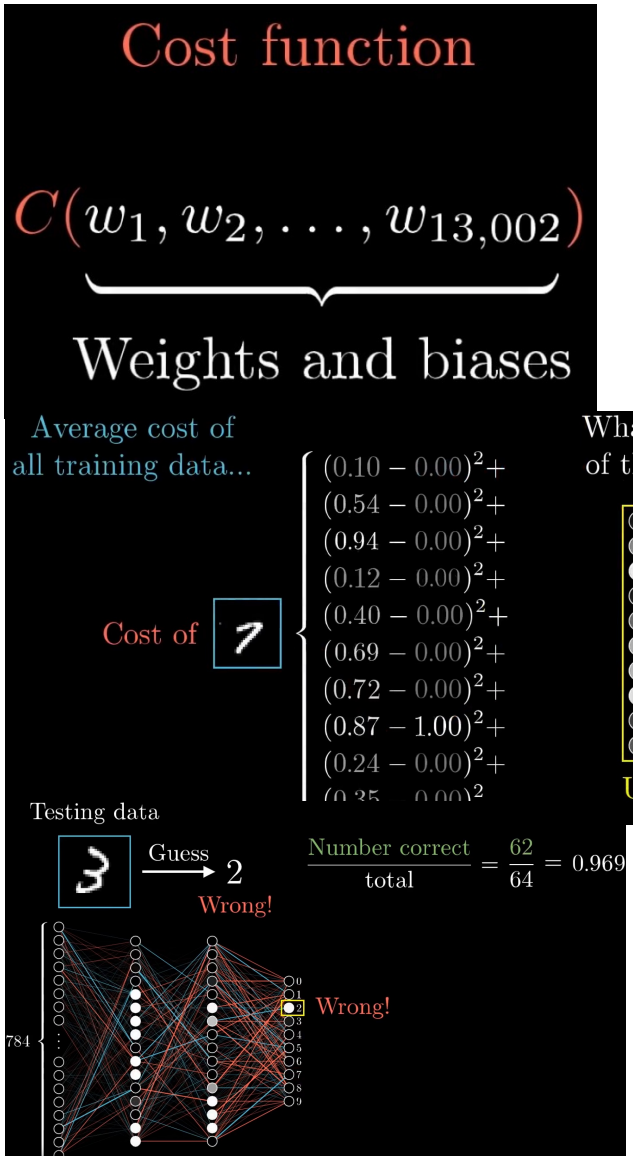
Input: 13,002 weights/biases

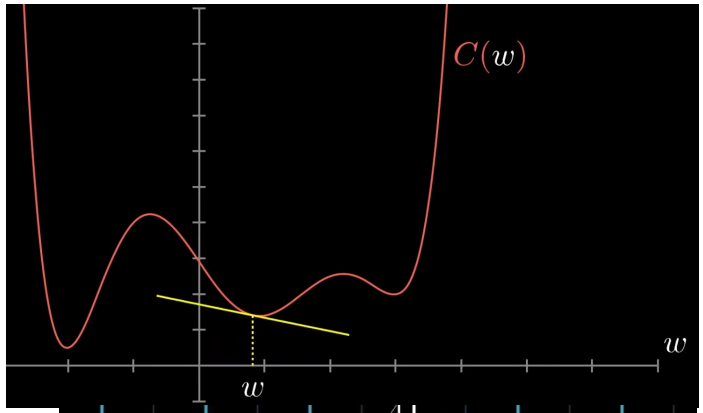
Output: 1 number (the cost)

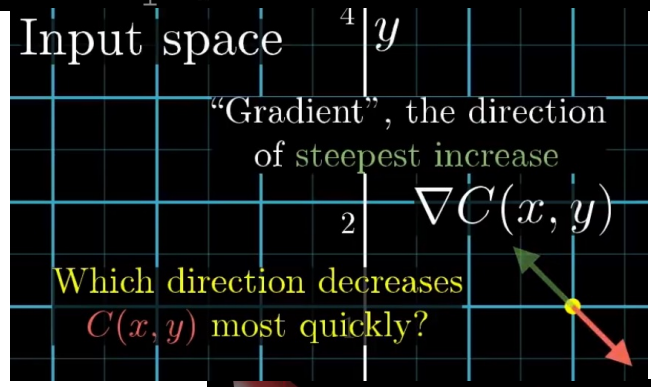
Parameters: Many, many, many training examples

Let’s write out the cost function

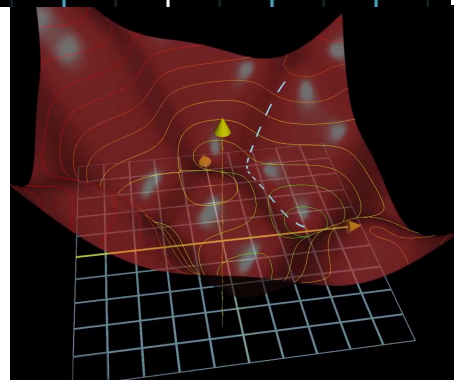








Optimization methods

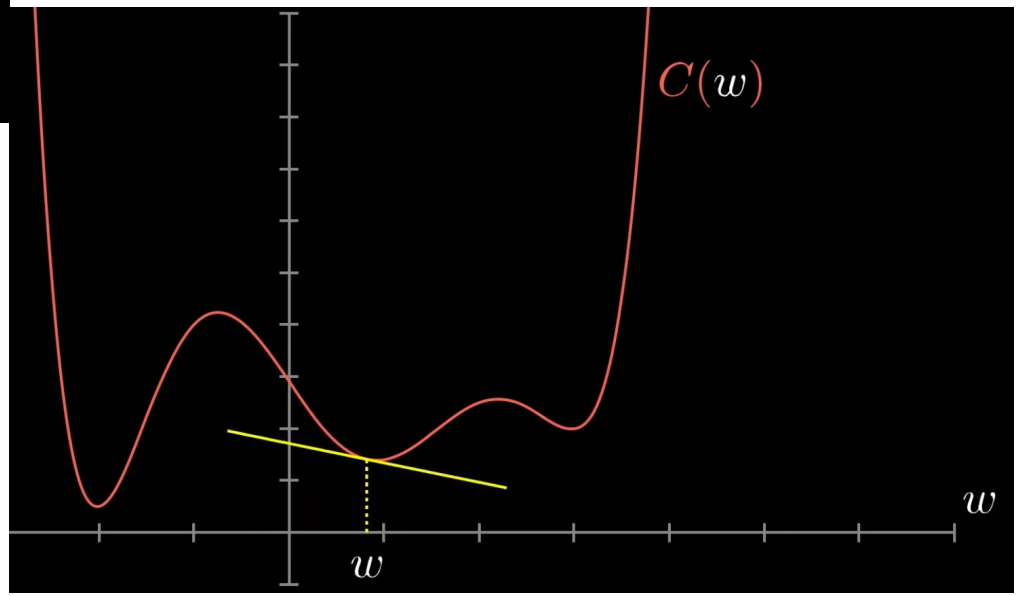


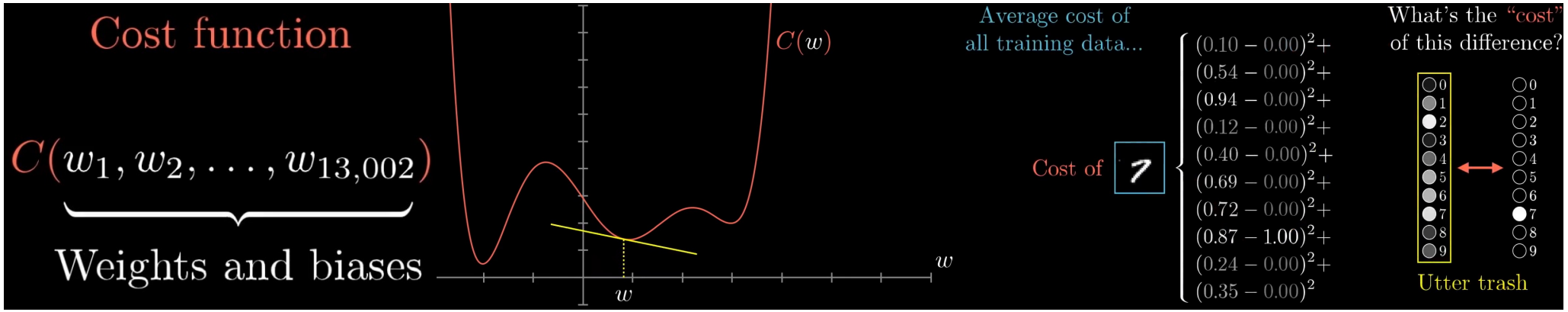
Cost function

$$C(w_1, w_2, \ldots, w_{13,002})$$

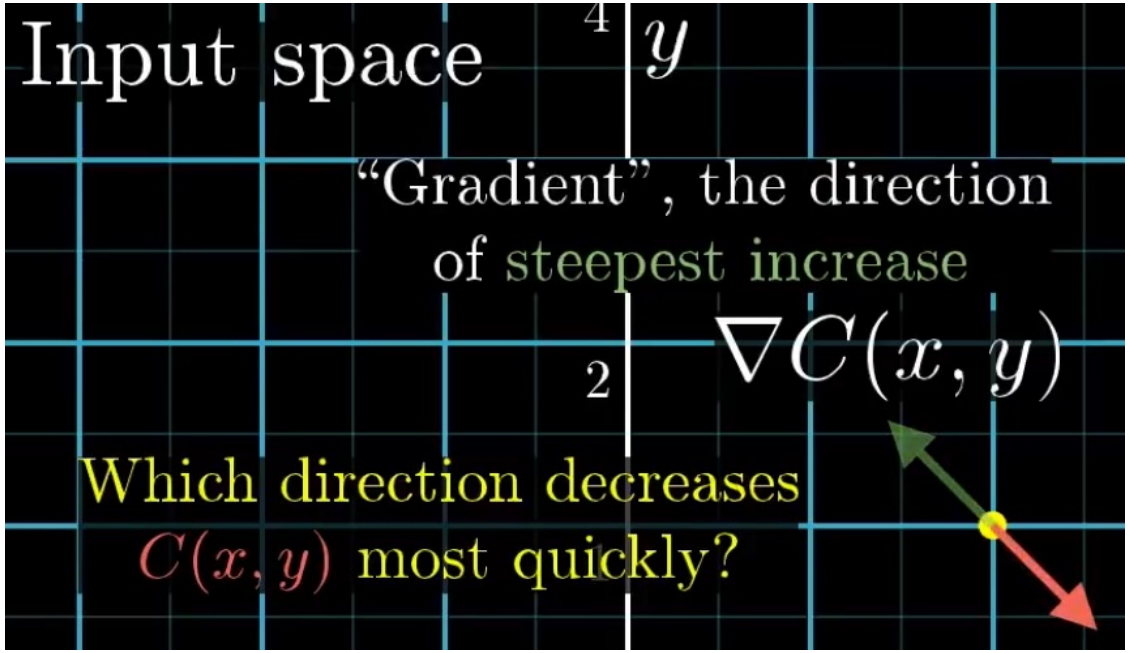
Weights and biases

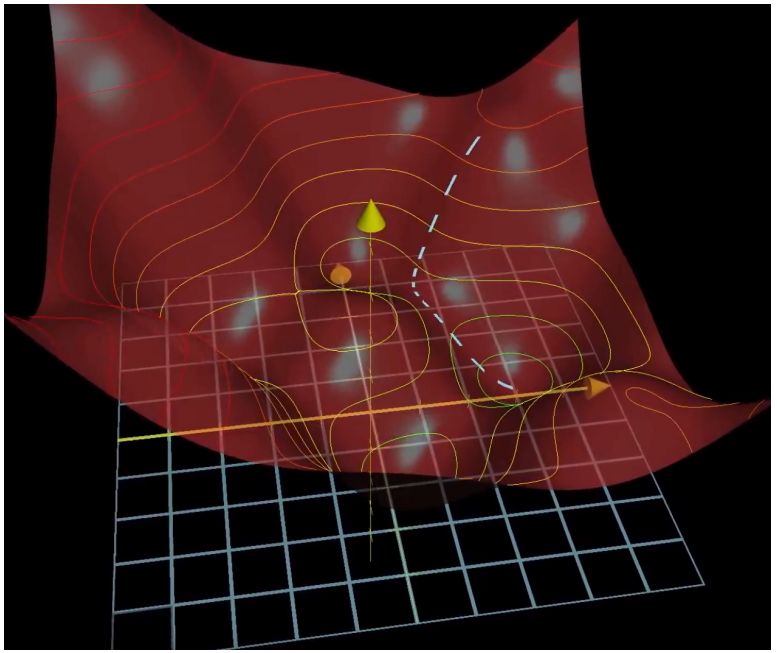
An optimization problem



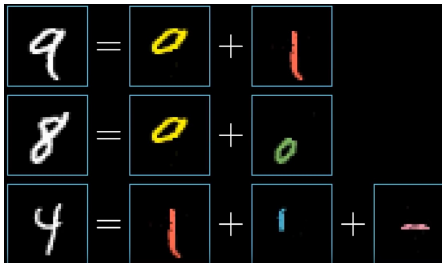


Optimization methods

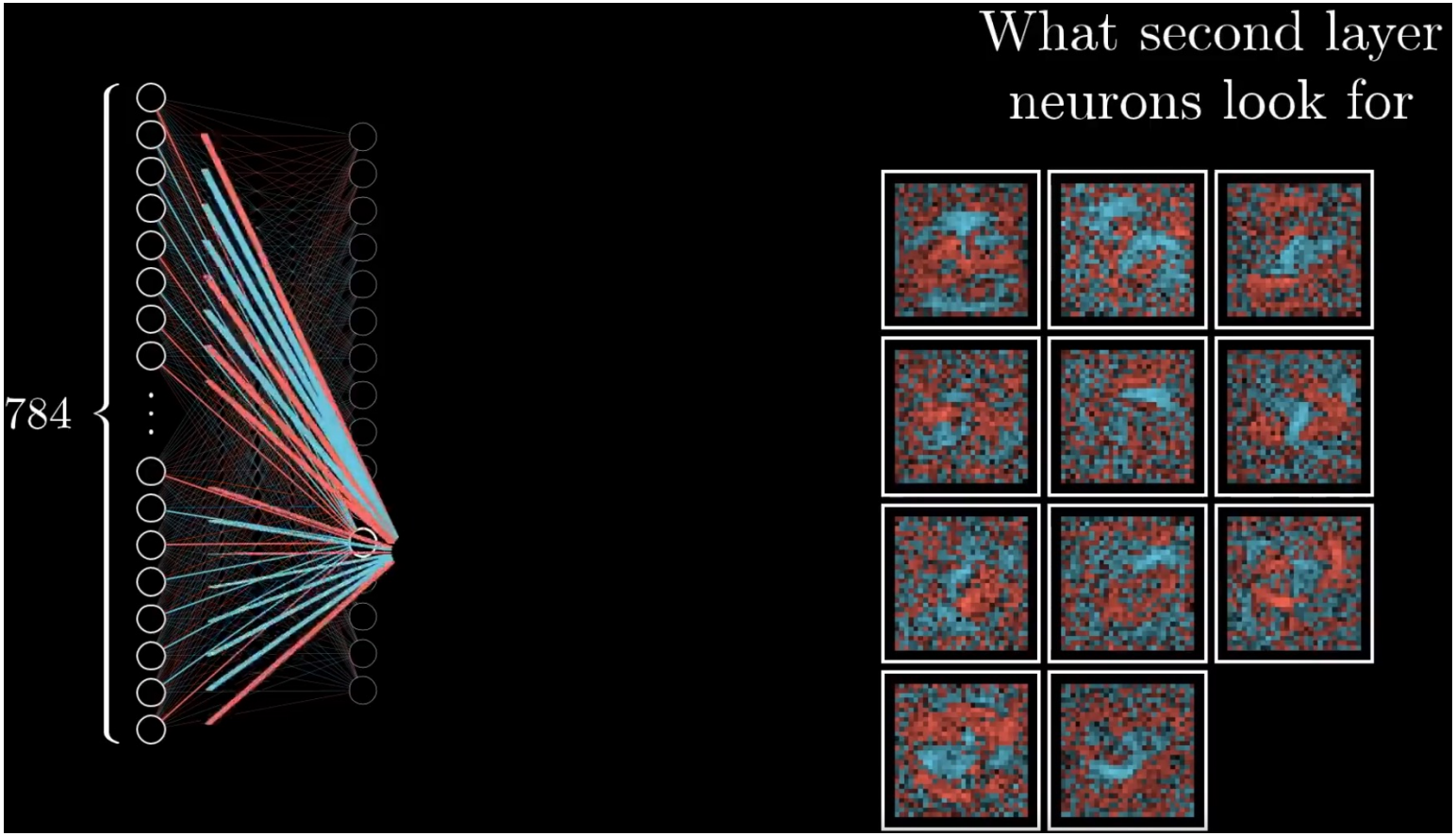




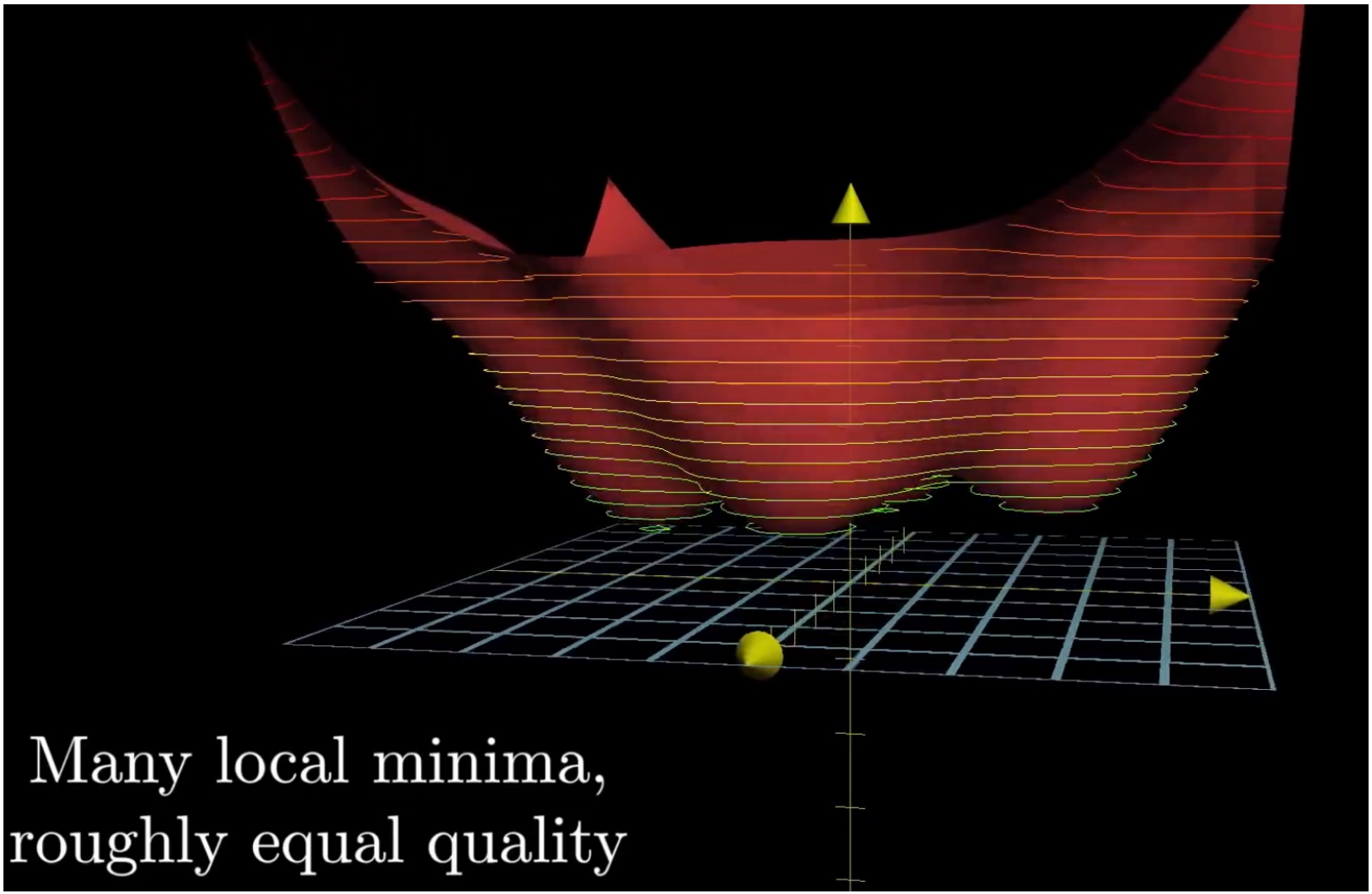
What do you wish in the “neurons”?



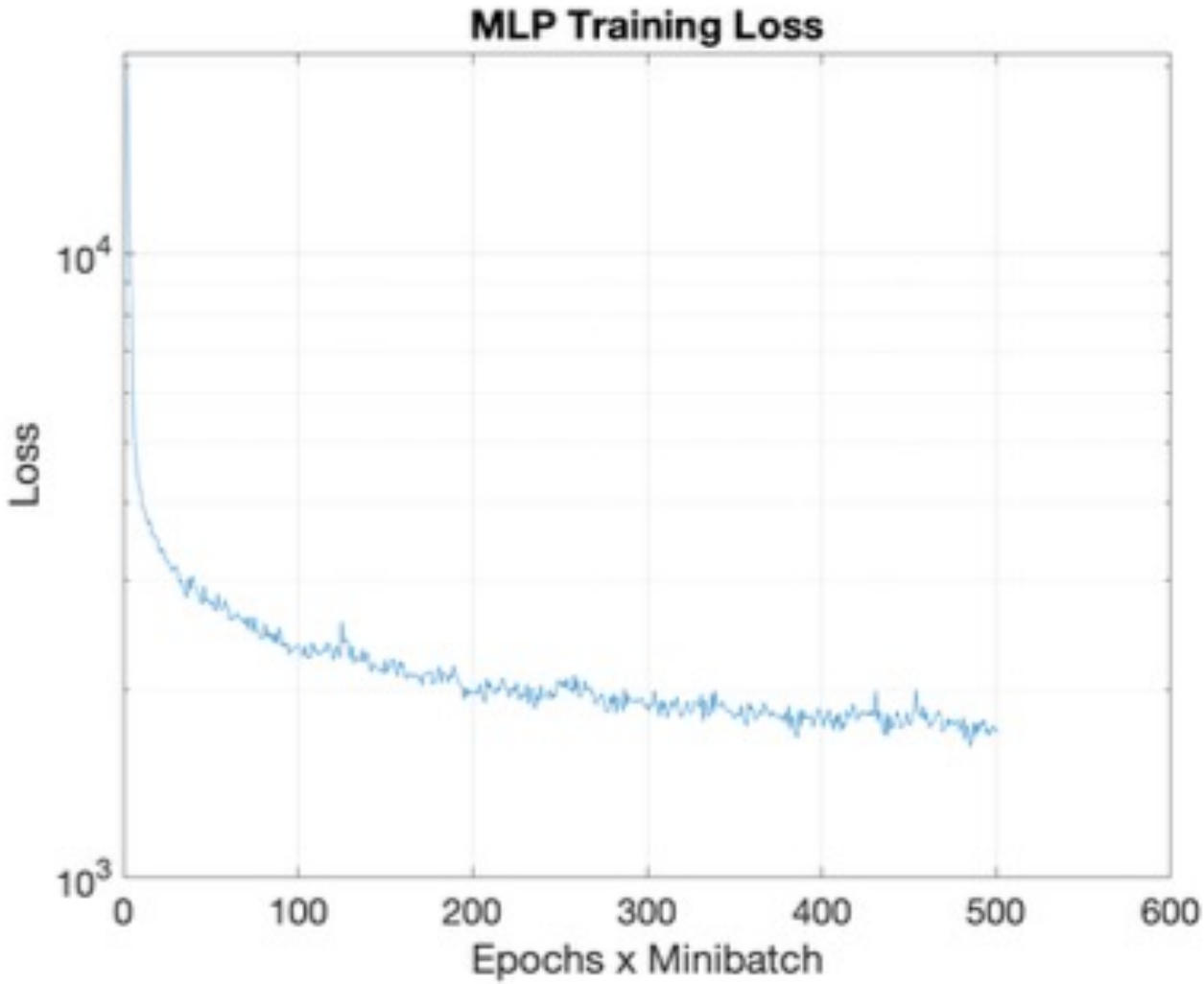
What do we actually get?



MANY local minimima of the VERY high dimensional so how important is it to find the best One?

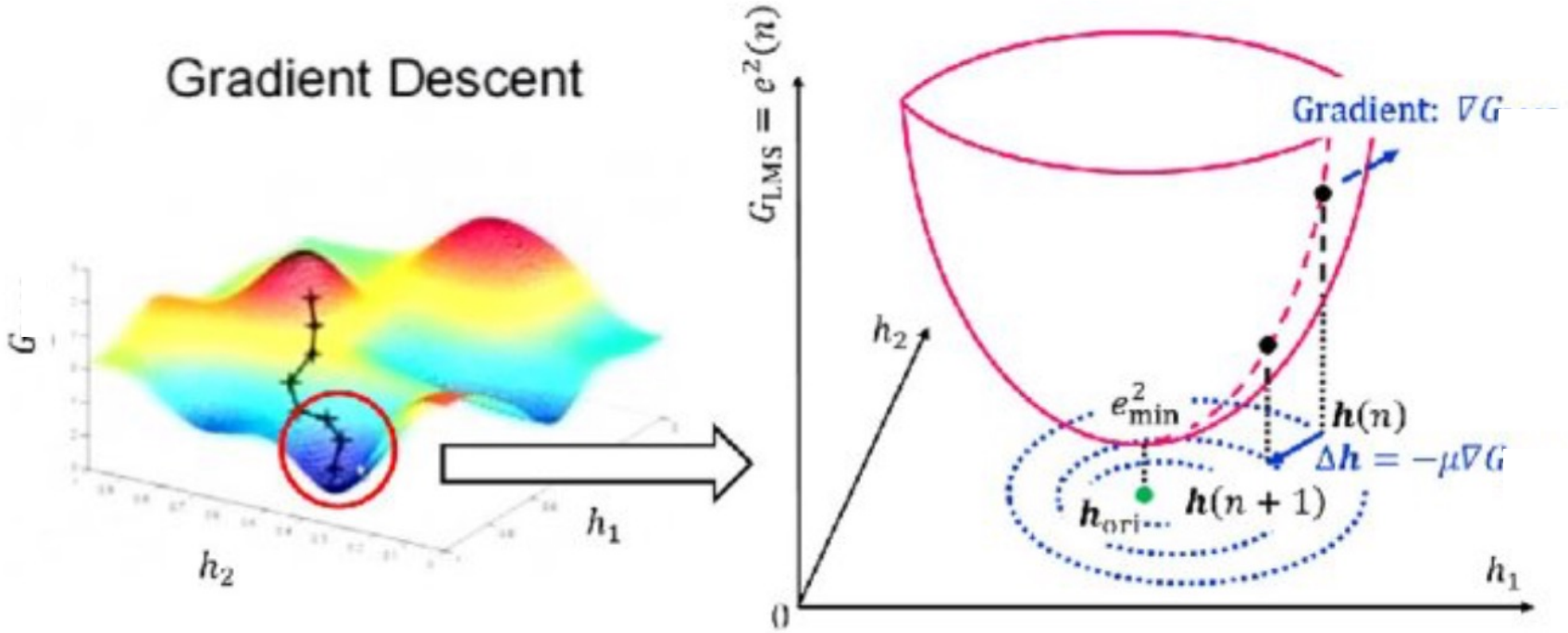


On training

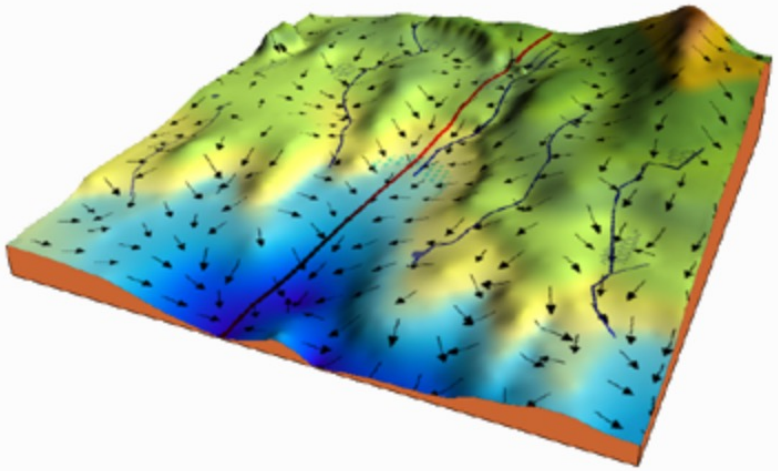

MLP Training Loss
Loss
10^4
10^3
0
100
200
300
400
500
600
Epochs x Minibatch

General smooth optimization, G(h):R^d->R

By general gradient descent



More on gradient descent



Given the cost function:

$$f(m, b) = \frac{1}{N} \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

The gradient can be calculated as:

$$f'(m, b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N} \sum -2(y_i - (mx_i + b)) \end{bmatrix}$$

$$\mathbf{w}_{j+1}(\delta) = \mathbf{w}_j - \delta \nabla f_k(\mathbf{w}_j)$$

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \cdot \nabla_\theta L(\theta^{(t)}, \mathbf{y})$$

In this equation:

- $\theta^{(t)}$ is our current estimate of θ^* at the tth iteration
- α is the learning rate
- $\nabla_\theta L(\theta^{(t)}, \mathbf{y})$ is the gradient of the loss function
- We compute the next estimate $\theta^{(t+1)}$ by subtracting the product of α and $\nabla_\theta L(\theta, \mathbf{y})$ computed at $\theta^{(t)}$

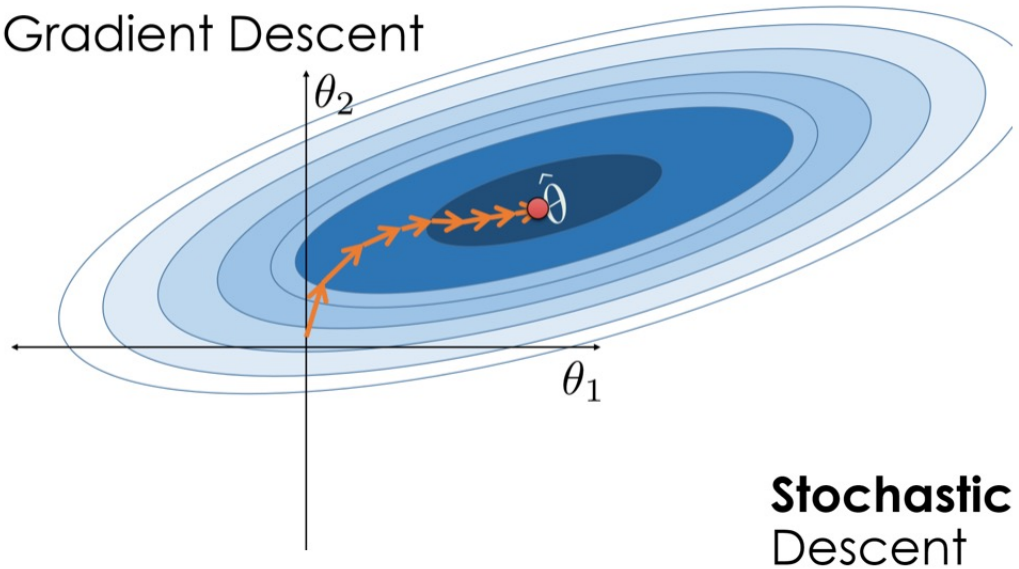
Stochastic Gradient Descent

$$\mathbf{w}_{j+1}(\delta) = \mathbf{w}_j - \delta \nabla f_K(\mathbf{w}_j)$$

Batch (several data points)

$$K \in [k_1, k_2, \cdots k_p]$$

Gradient Descent



Stochastic Gradient Descent