Deep Learning: Session 5

Deep Neural Nets

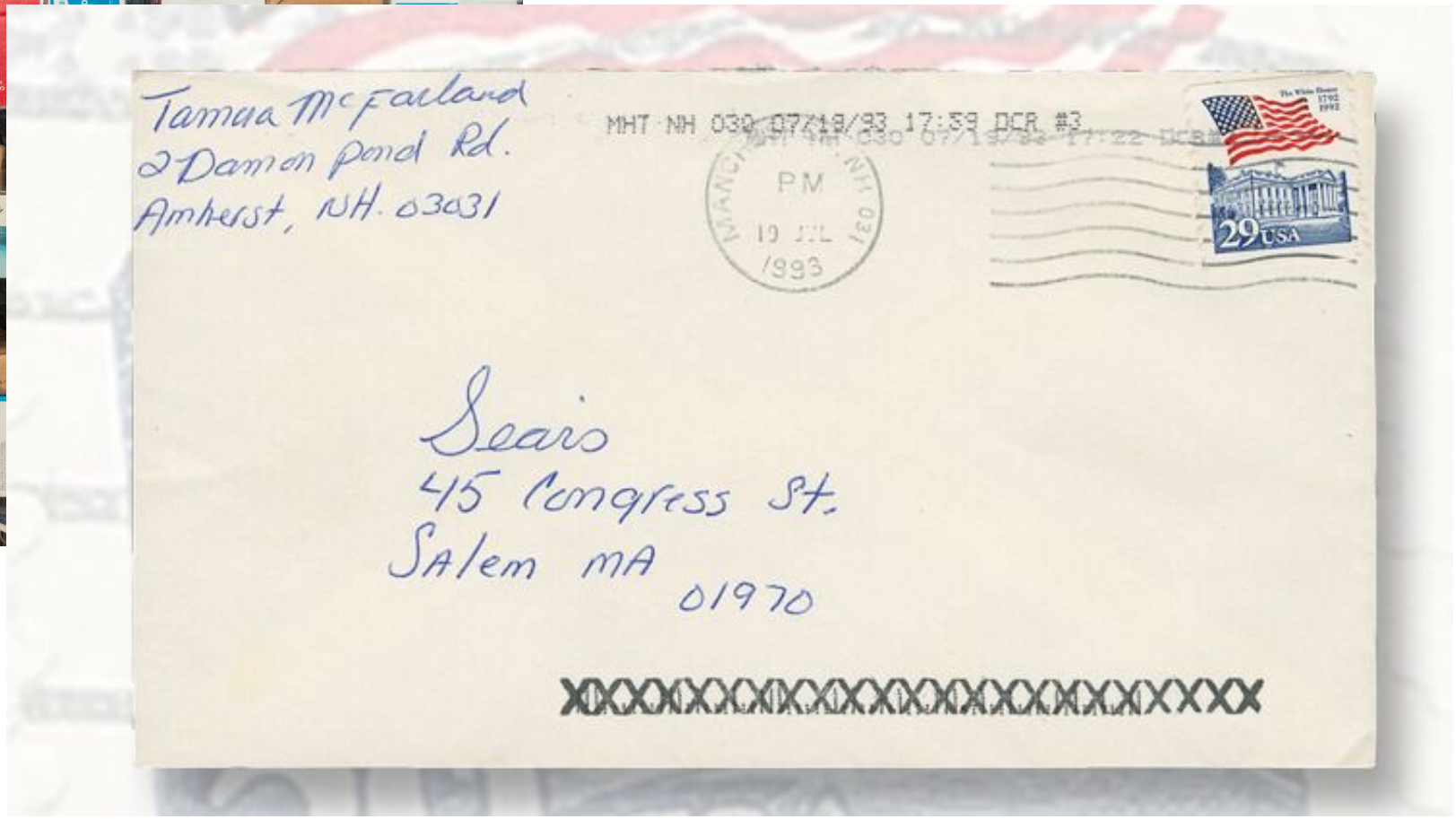
Outline

1. Recap
2. Back Propagation
3. Bias & Variance
4. Regularization

In previous chapters...

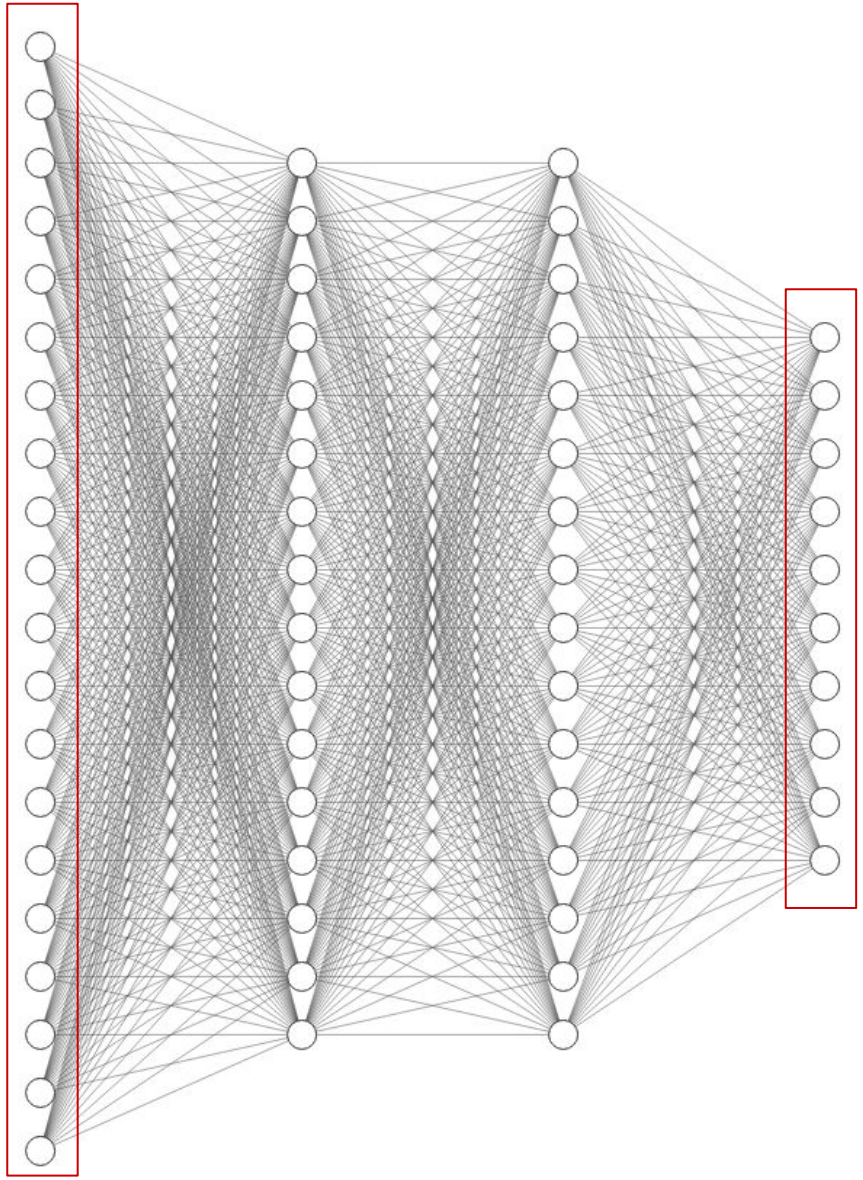
- From circuit theory:
  - We know that for representing some functions we could need many components in shallow architectures.
  - ... Functions that we can easily represent if we include more layers.
- The Divide & Conquer strategy is powerful, and it is efficient.

In previous chapters...



In previous chapters...

784 píxeles

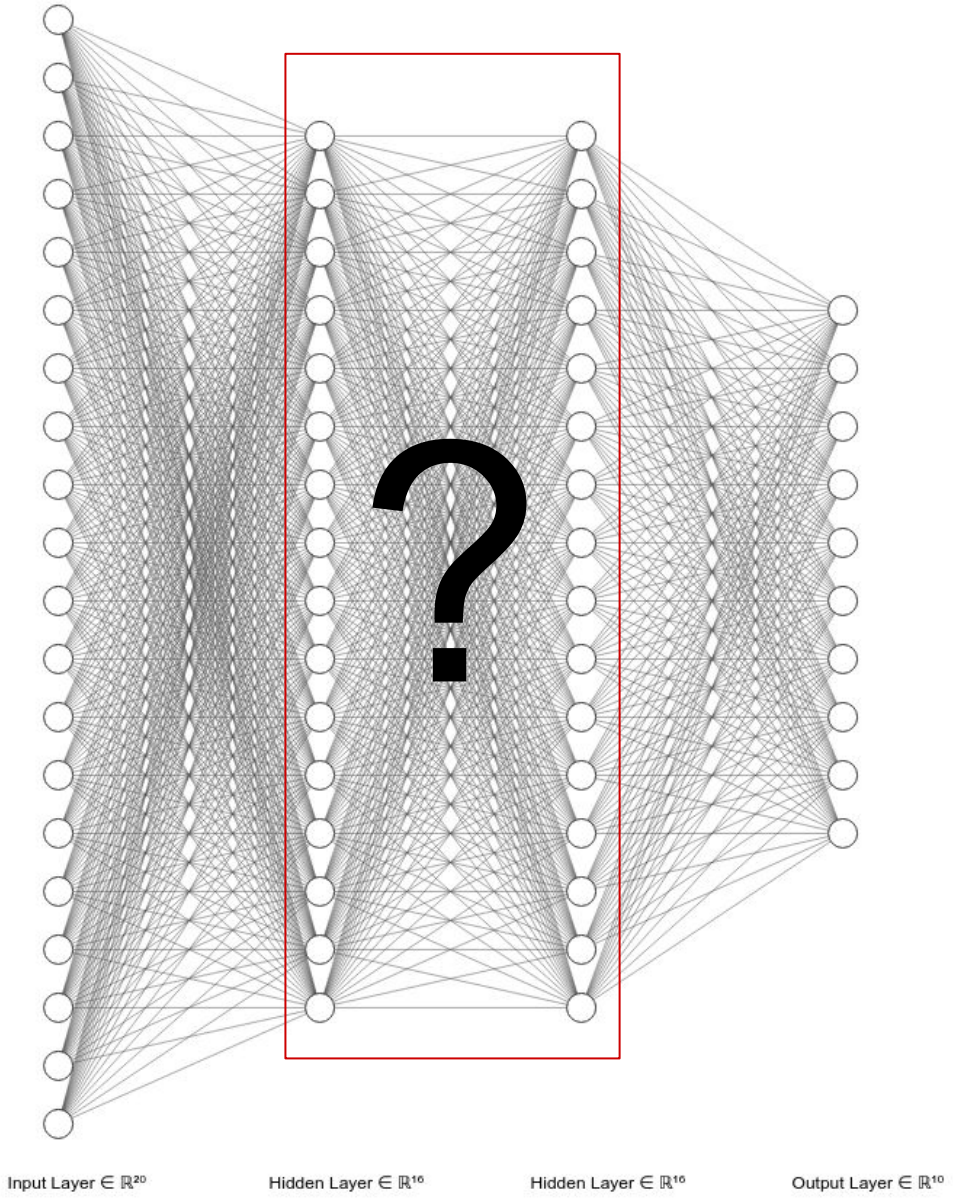


Input layer: 784 neurons

Output layer: 10 neurons.

In previous chapters...

784 píxeles

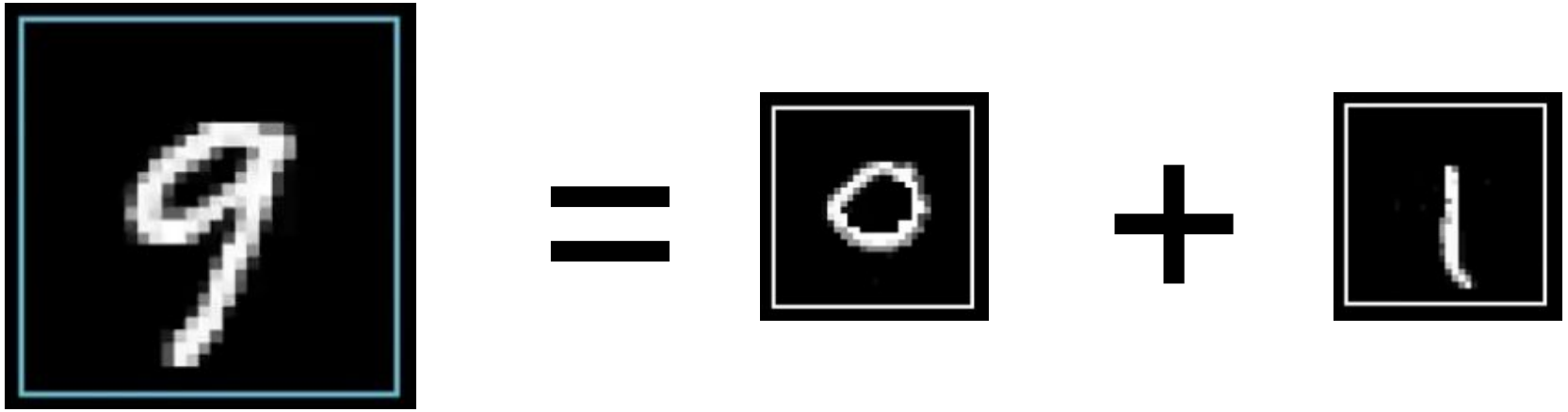


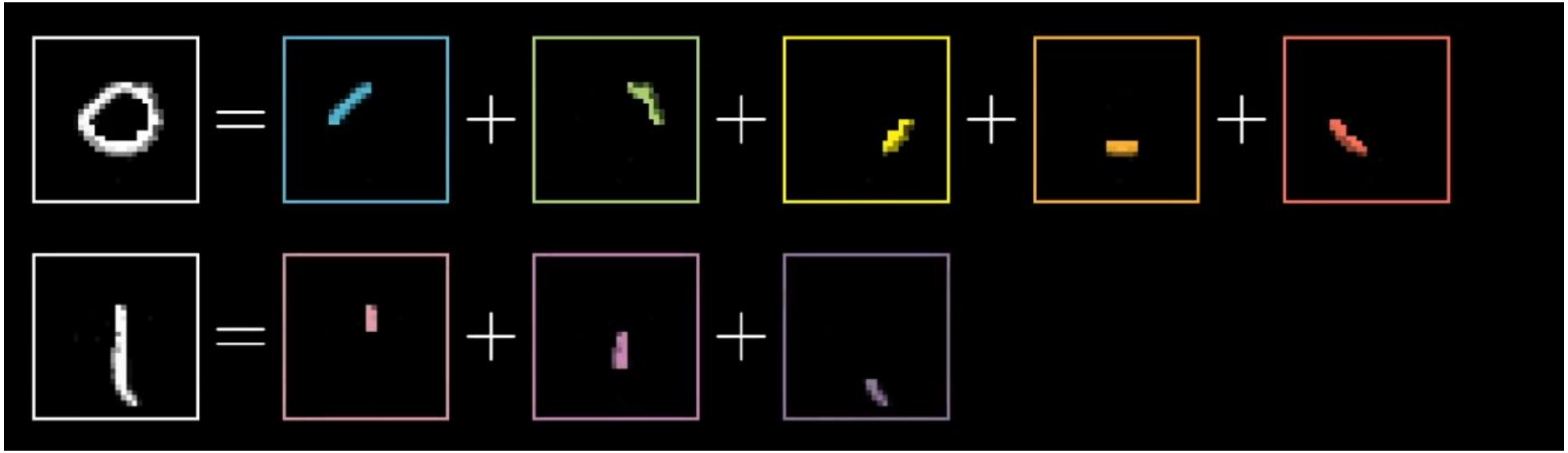
Remember:

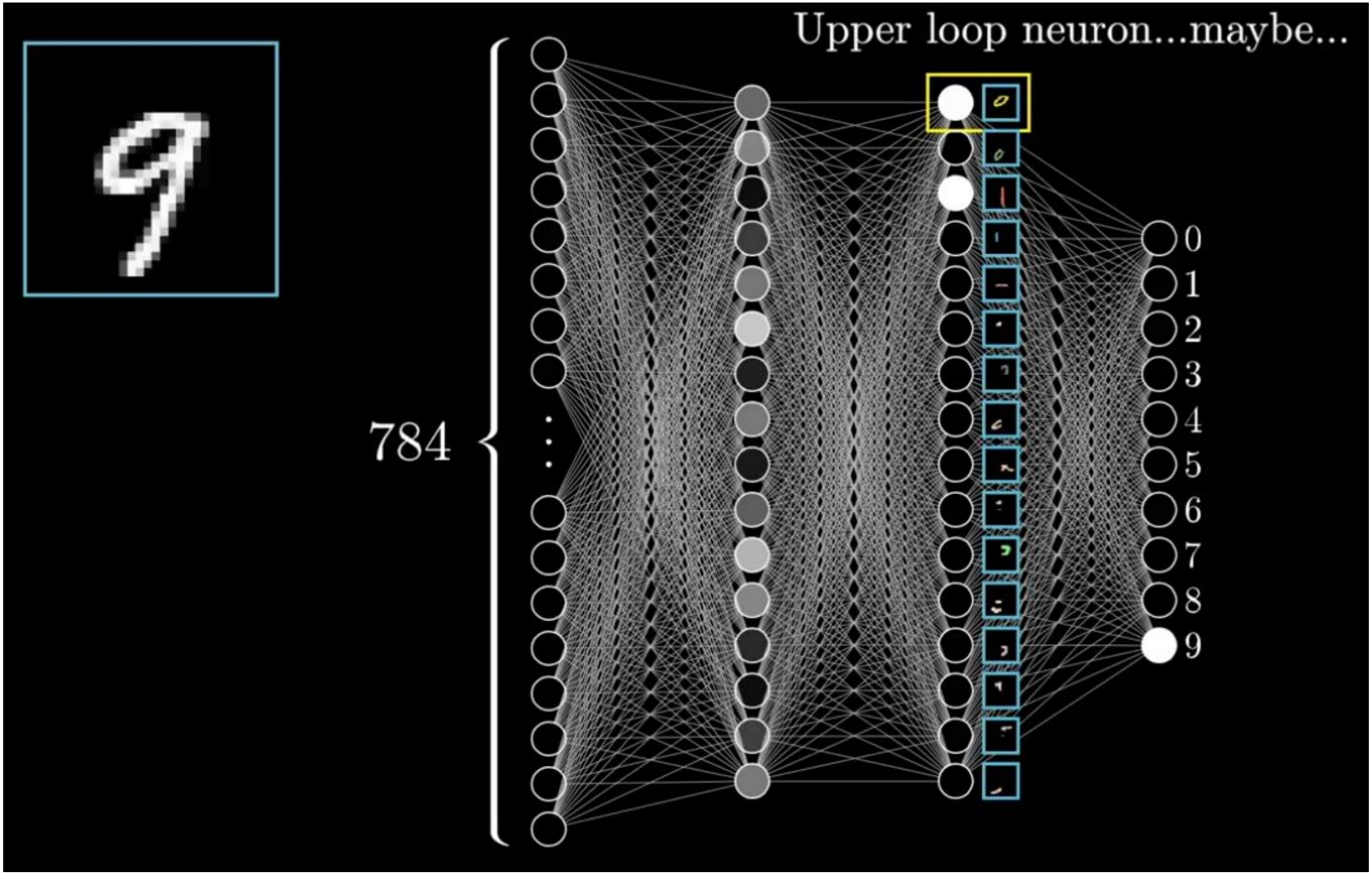
¿Why do we use layers?

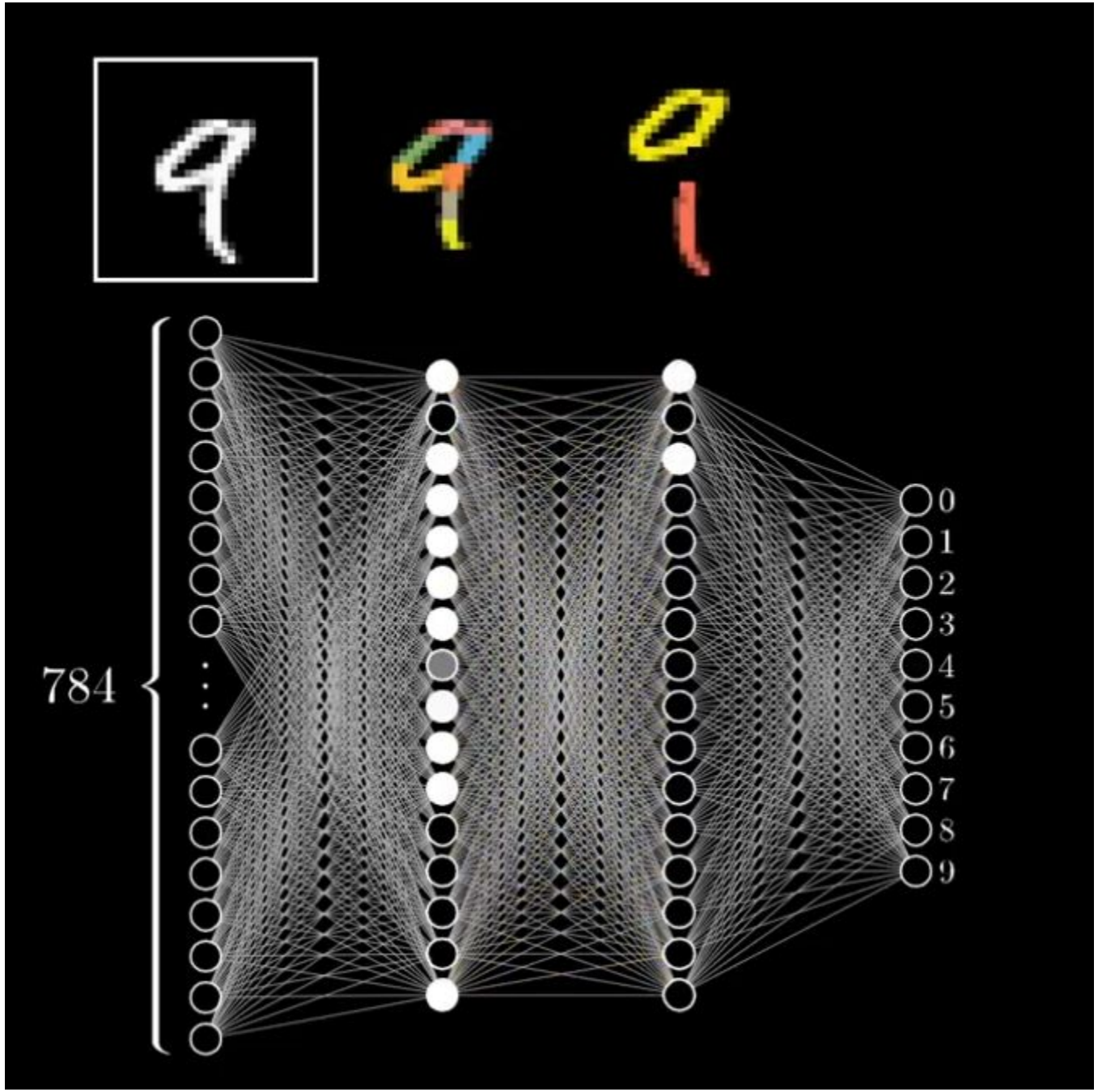
- XOR
- Circuit theory

In previous chapters...

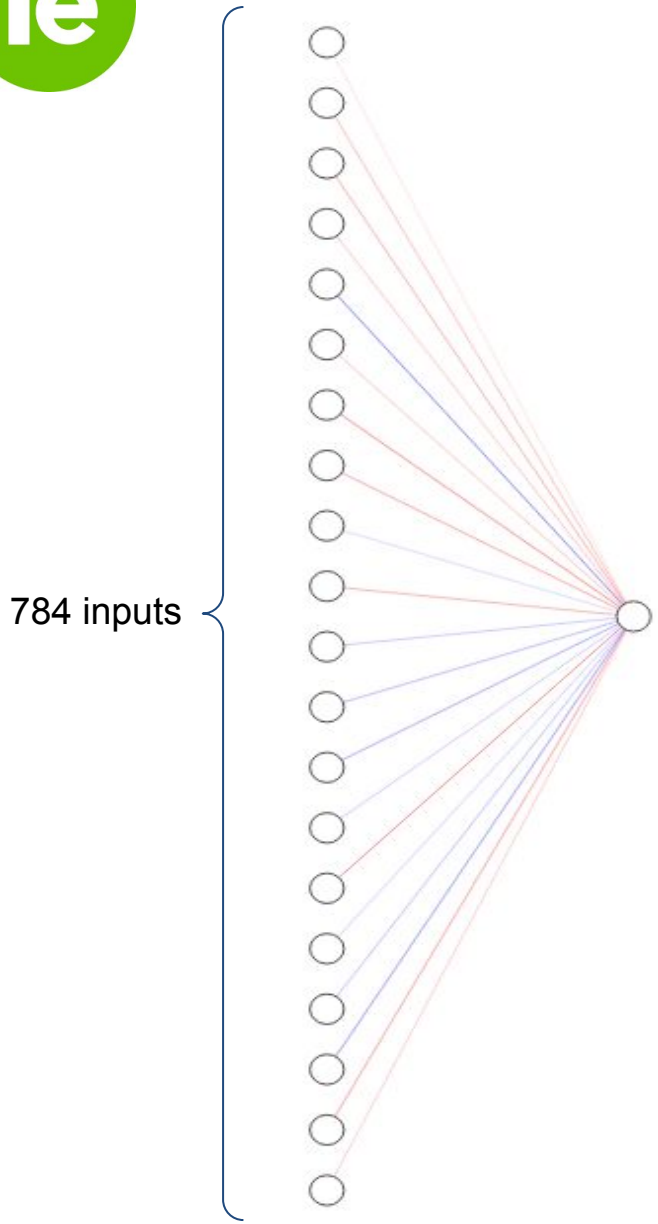


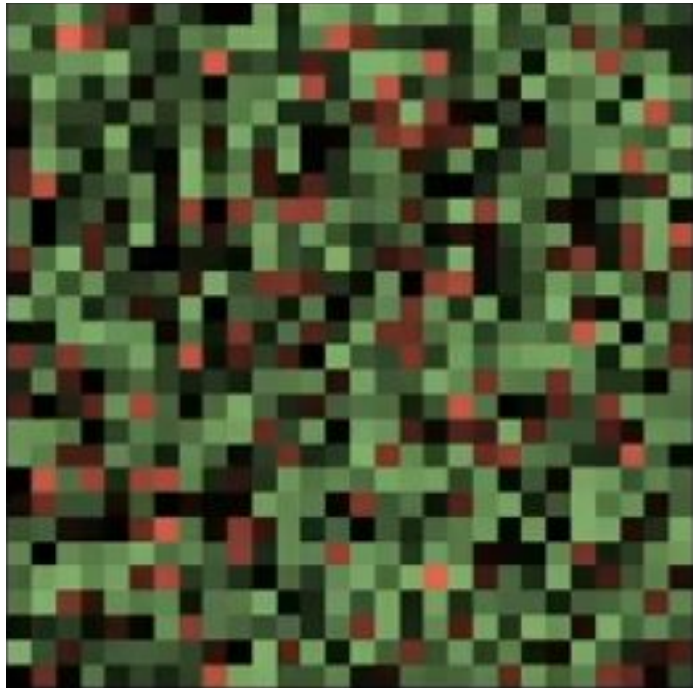


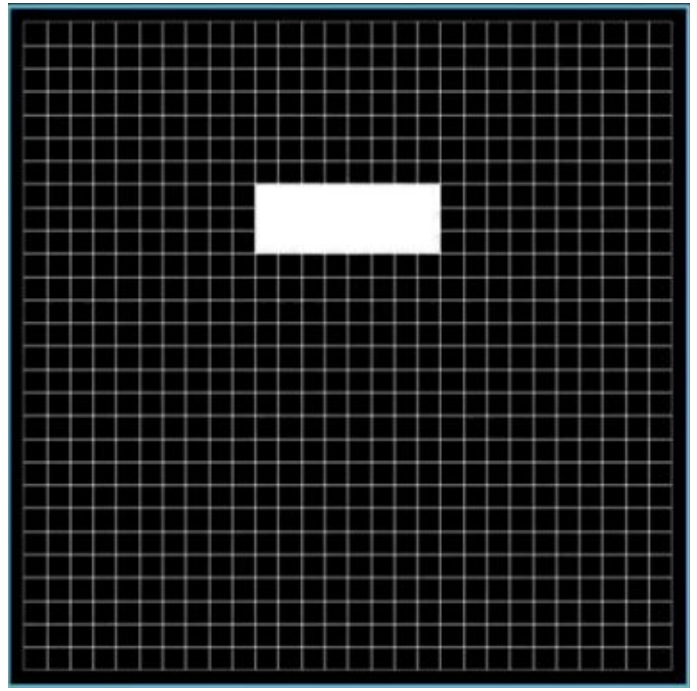

Upper loop neuron...maybe...
784
0
1
2
3
4
5
6
7
8
9


784
0
1
2
3
4
5
6
7
8
9

In previous chapters...



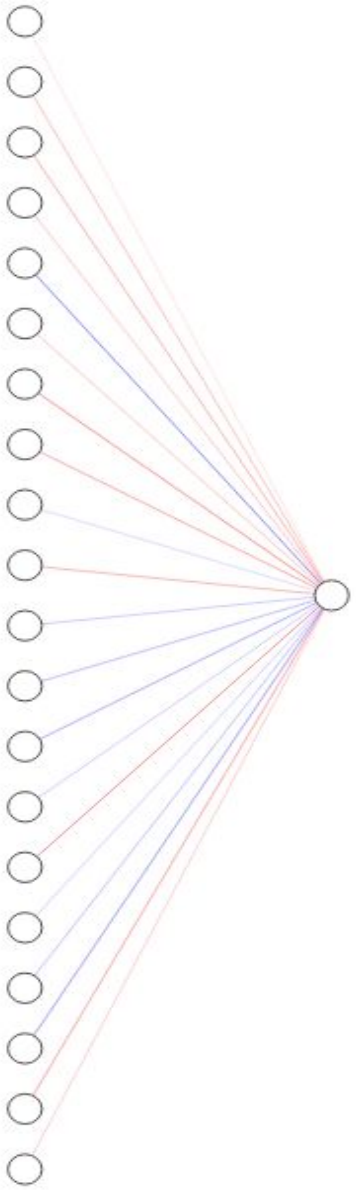




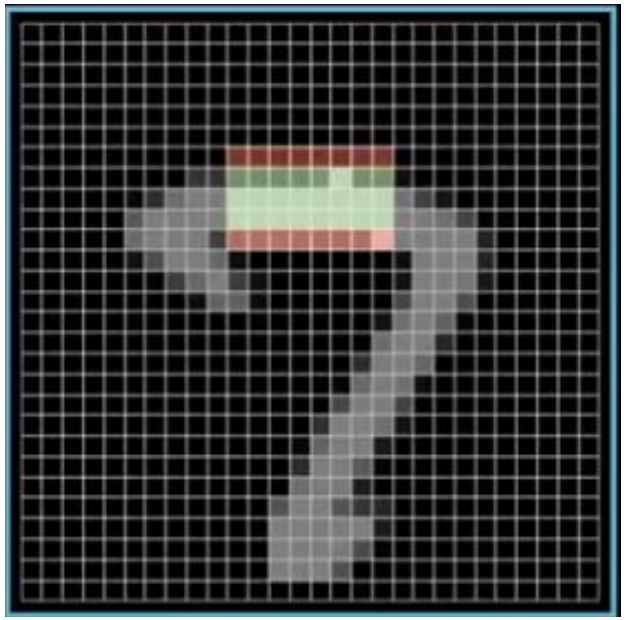
$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

In previous chapters...

784 inputs



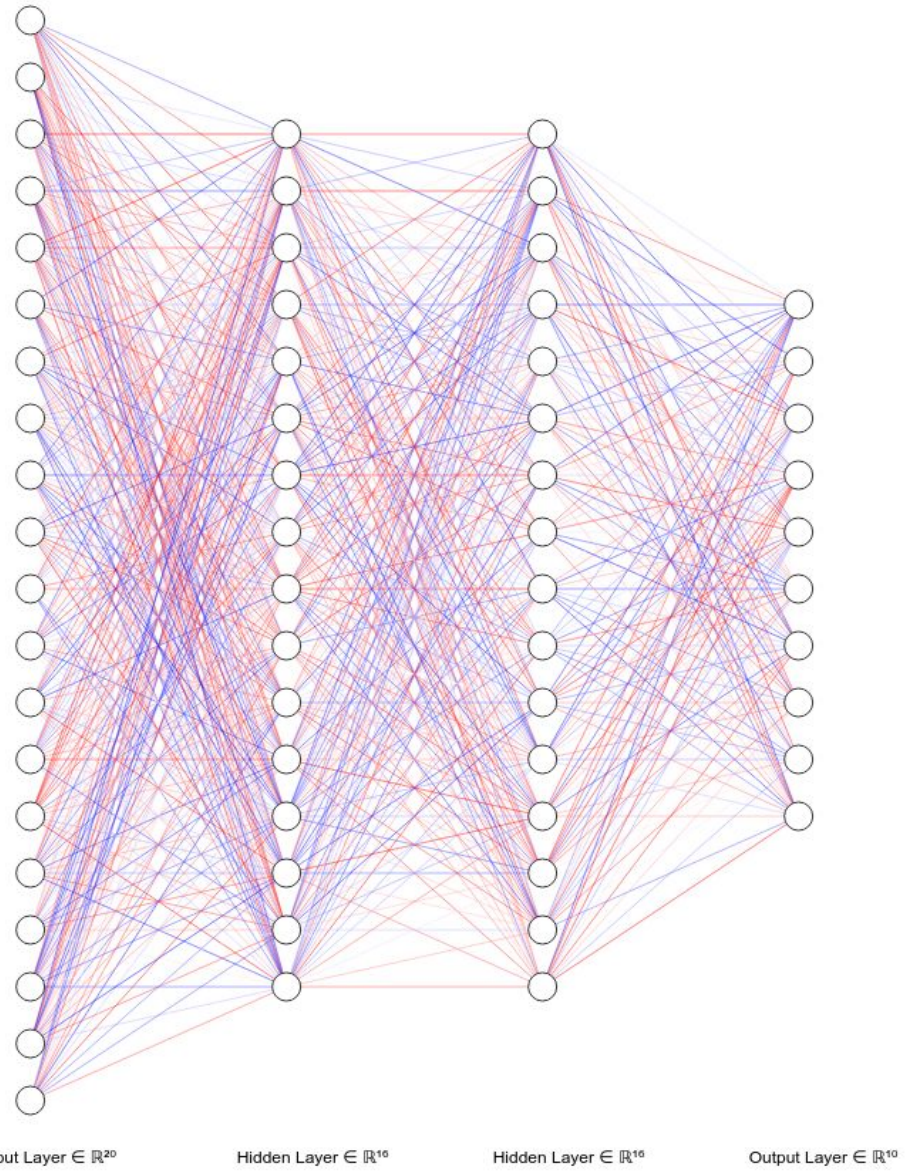
- Watch out! There is a mistake here!



$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

In previous chapters...

784 pixels



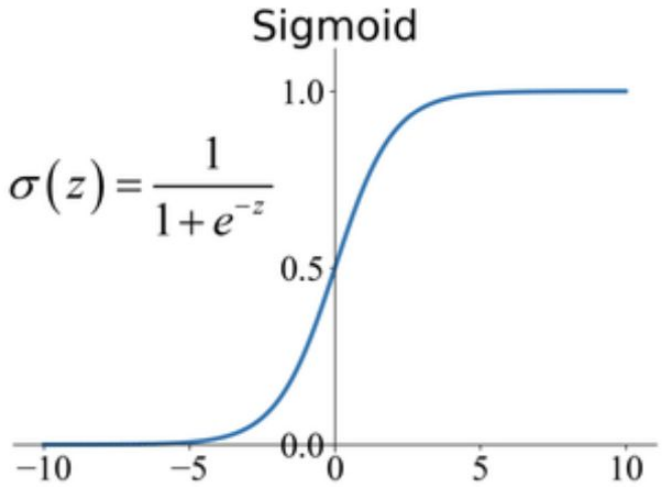
Complete Network:

Weights:
784*16 + 16*16 + 16 * 10

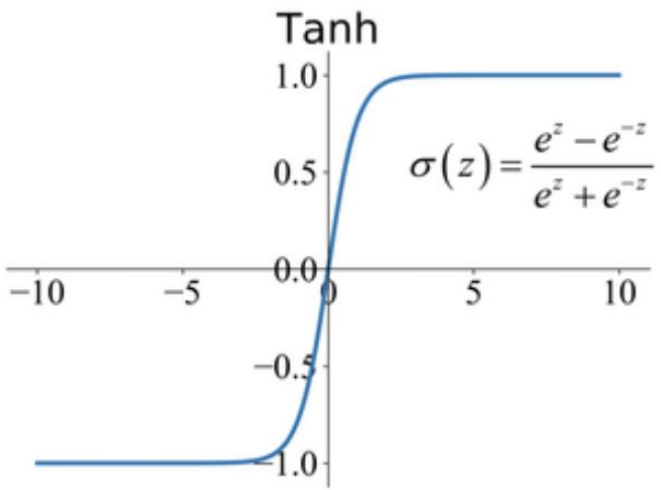
Bias:
16+16+10

Trainable Params:
13002

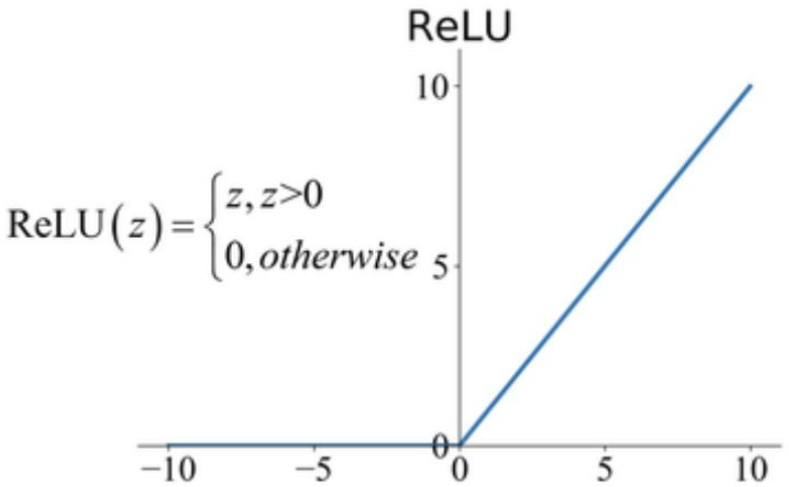
In previous chapters...



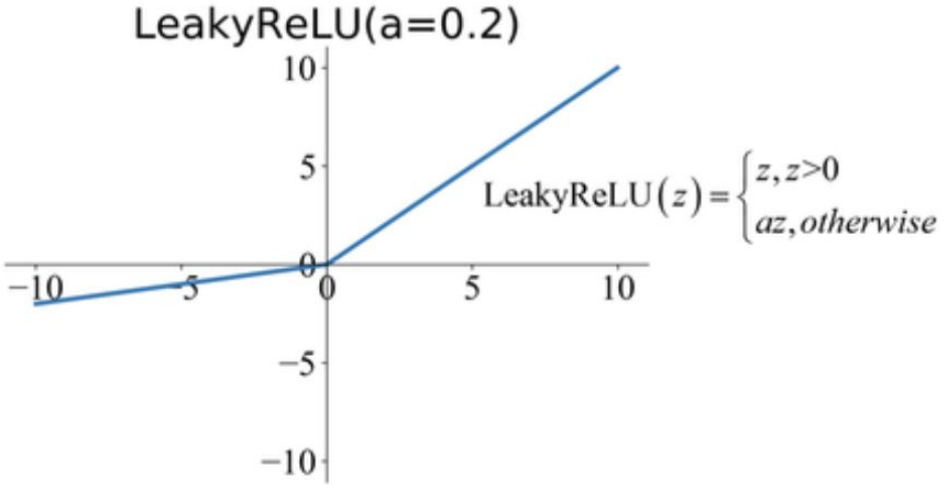
(a)



(b)



(c)



(d)

In previous chapters...

- The combination of linear functions is just another linear function. No matter the number of hidden layers, if we use linear functions as activations, we could just deal with outputs that are a linear combination of our inputs... not interesting.

- Non-linear activations functions are key in neural nets to represent non-linear mapping functions.

- **In the output layer**, at general: use sigmoid for binary classification, softmax for multiclass classification and linear for regression.

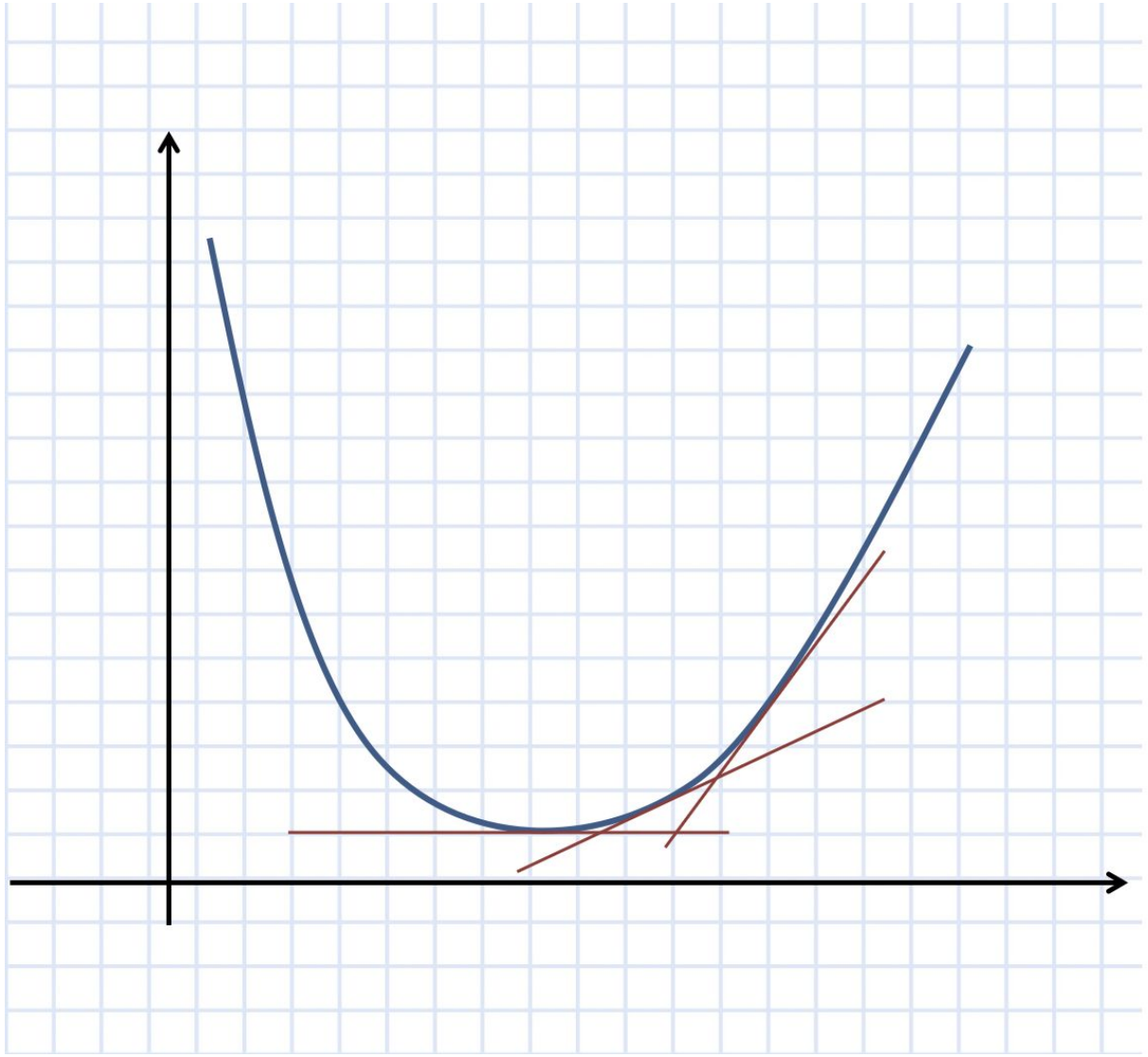
    - You can use softmax for binary classification using two outputs and in the case of a problem where data that can belong to multiple classes, then use sigmoid. [i.e., a picture of Audi belongs to the classes' car and vehicle]

- **In hidden layers:**
  - Tanh use to work better than sigmoid as it centers data.
  - ReLu is faster and it works very well too. It is the most used.

Outline

1. Recap
2. Back Propagation
3. Bias & Variance
4. Regularization

ML Heart: Gradient Descent



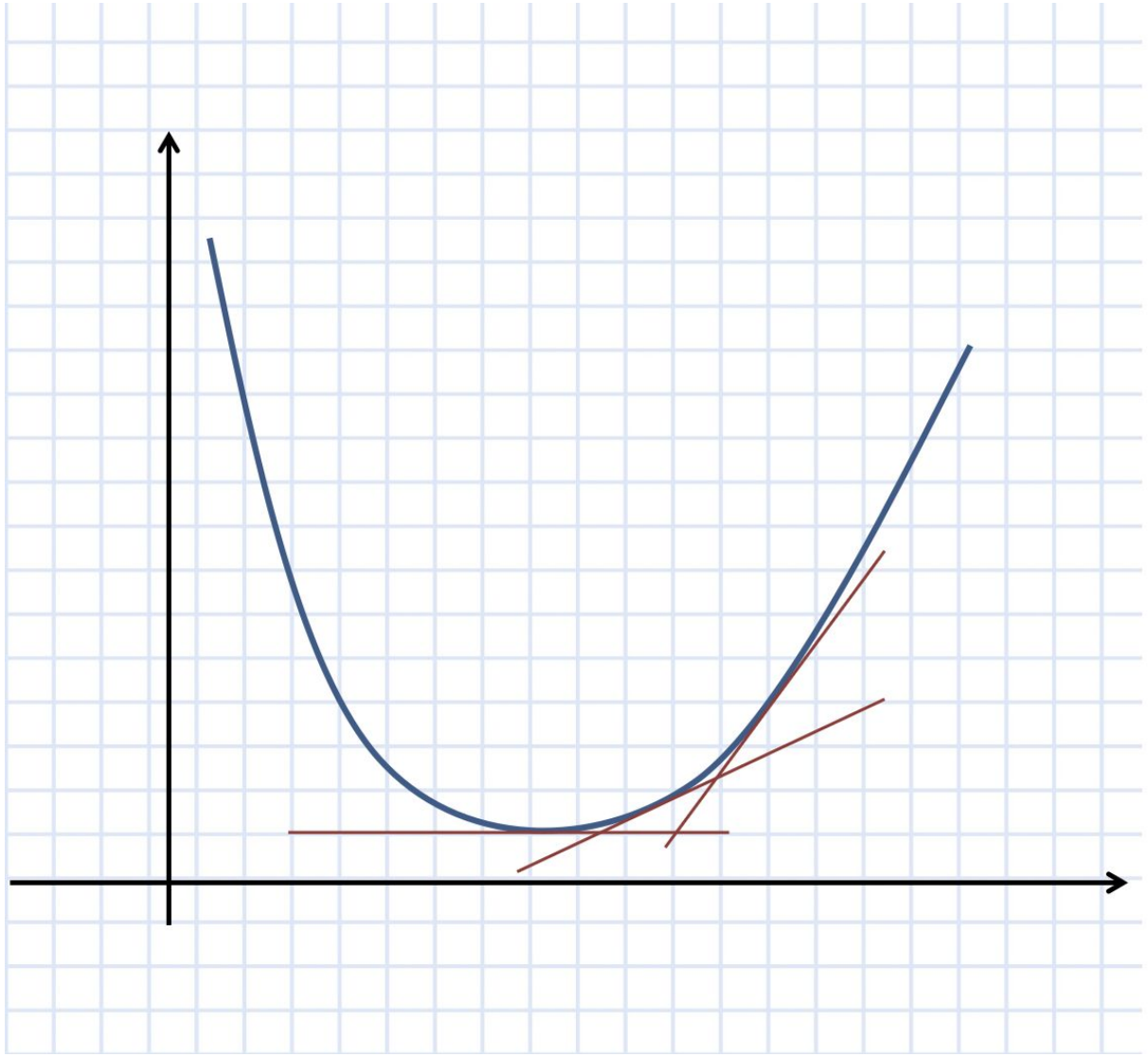
1. Pick a random w (w^0)

2. Repeat until convergence {

$$w^{i+1} = w^i - \alpha dL(w)/d(w)_{[w^i]}$$

}

ML Heart: Gradient Descent



1. Pick a random w (w^0)

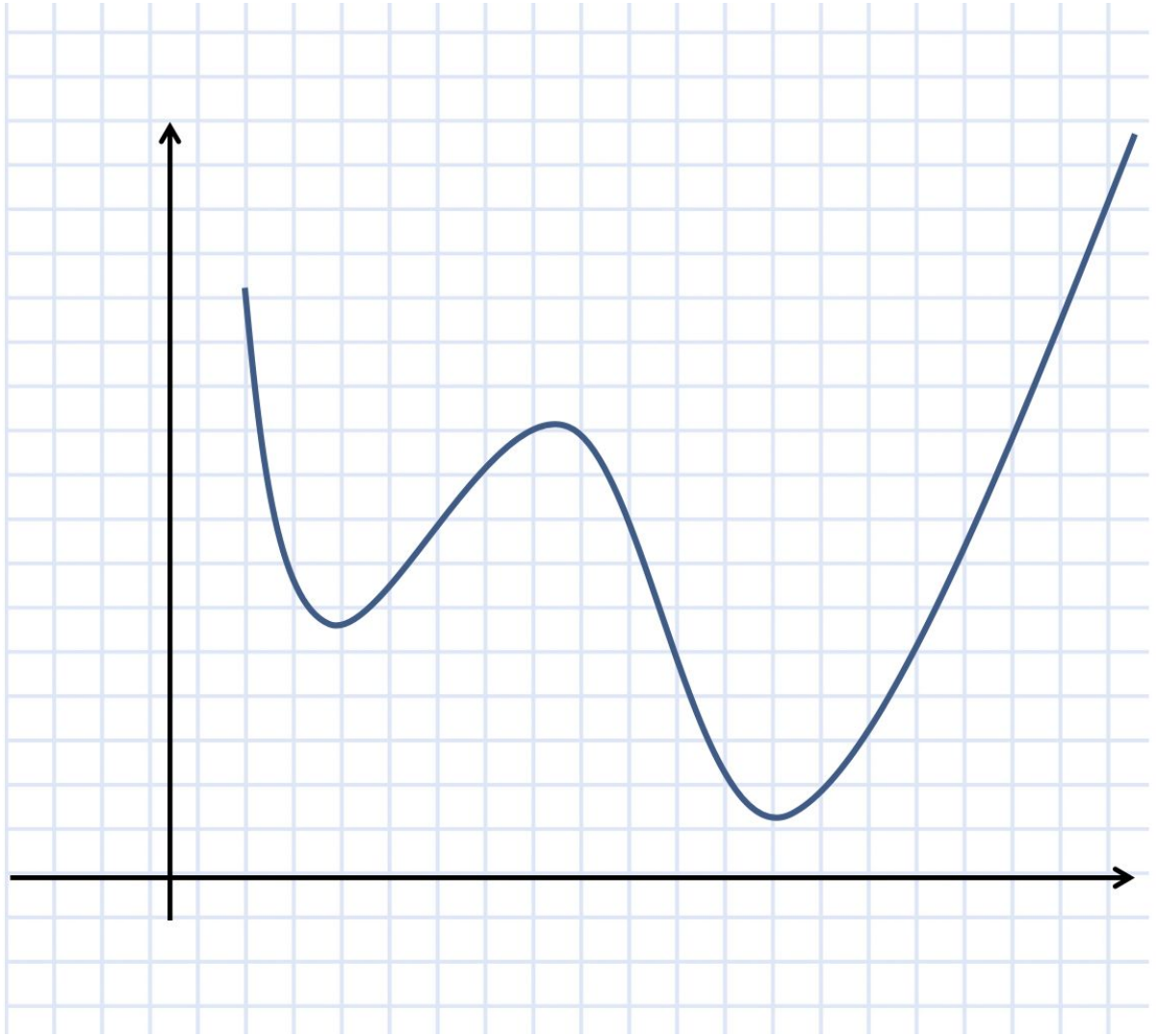
2. Repeat until convergence {

$$w^{i+1} = w^i - \alpha dL(w)/d(w)_{[w^i]}$$

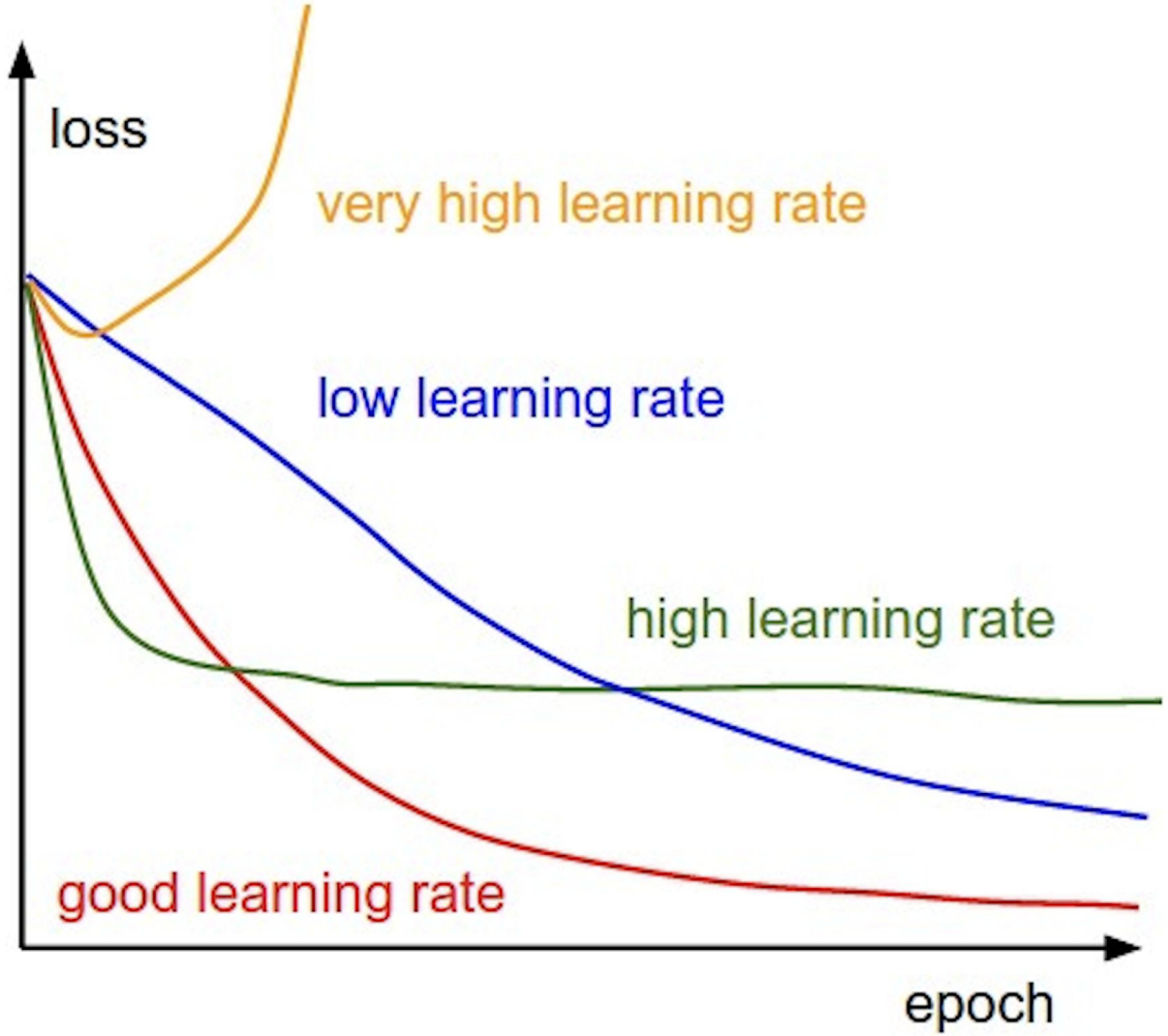
}

Learning Rate role

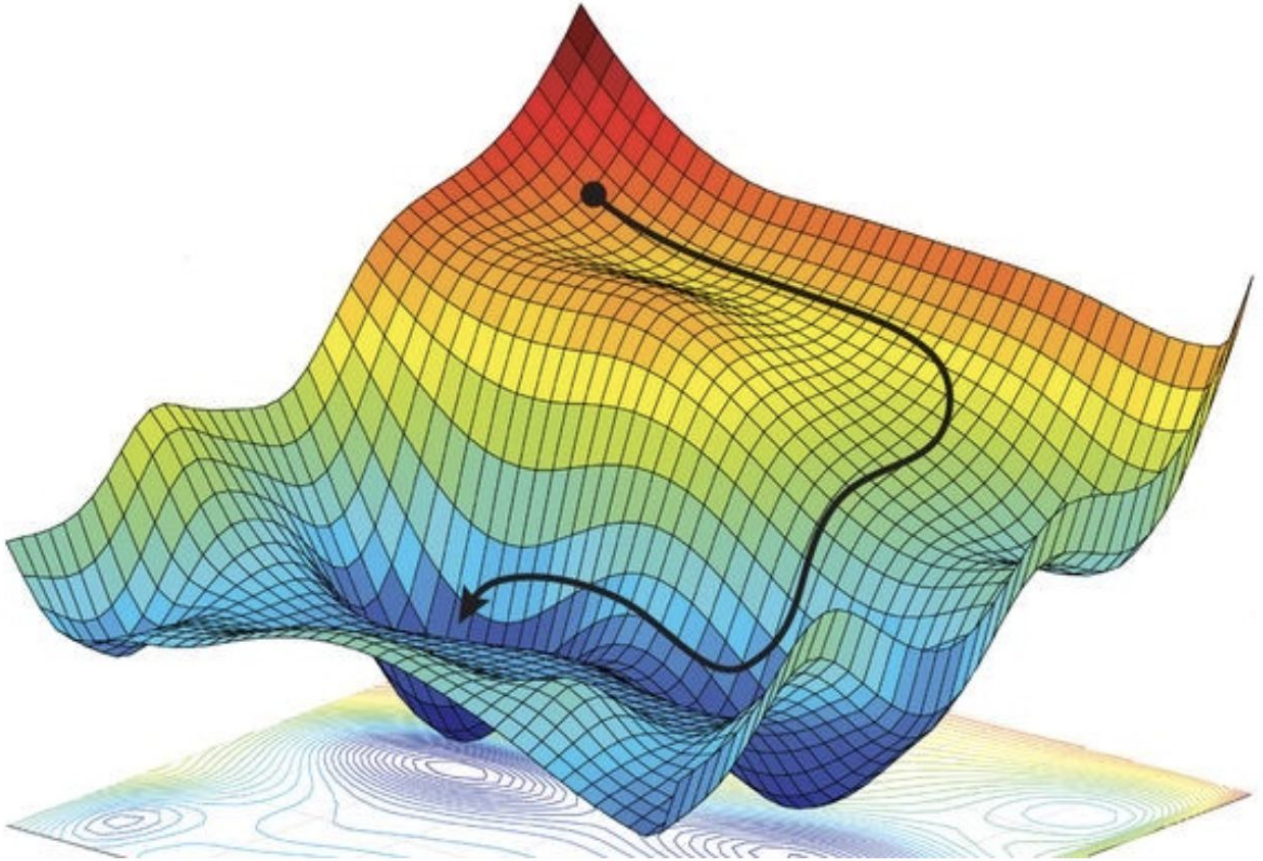
ML Heart: Gradient Descent



Learning Rate role: α



ML Heart: Gradient Descent

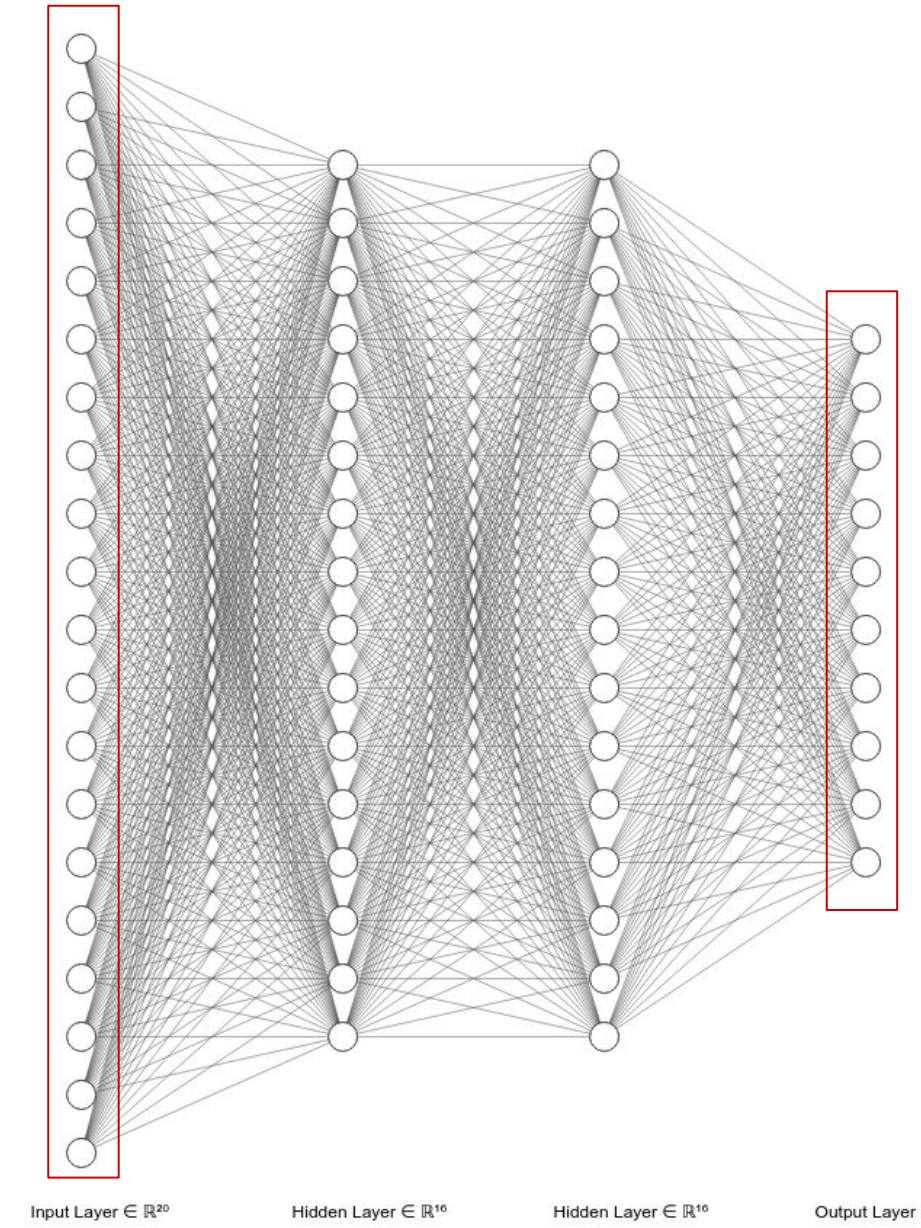


Backpropagation

- Backpropagation (BP) is the smart and elegant way to apply gradient descent in multilayer neural nets.
- It is working by ‘propagating’ the error backwards (from the output layer to the input one) layer by layer.
- All the trainable params, weights and bias are contributing to the error. By using BP, we can assign to all of them their specific contribution to the final error.
- Mathematically, BP exploits the chain rule of the derivatives.

Back propagation

784 pixels



Input Layer: 784 neurons

Output Layer: 10 neurons.

How can we apply Gradient Descent?

Back propagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

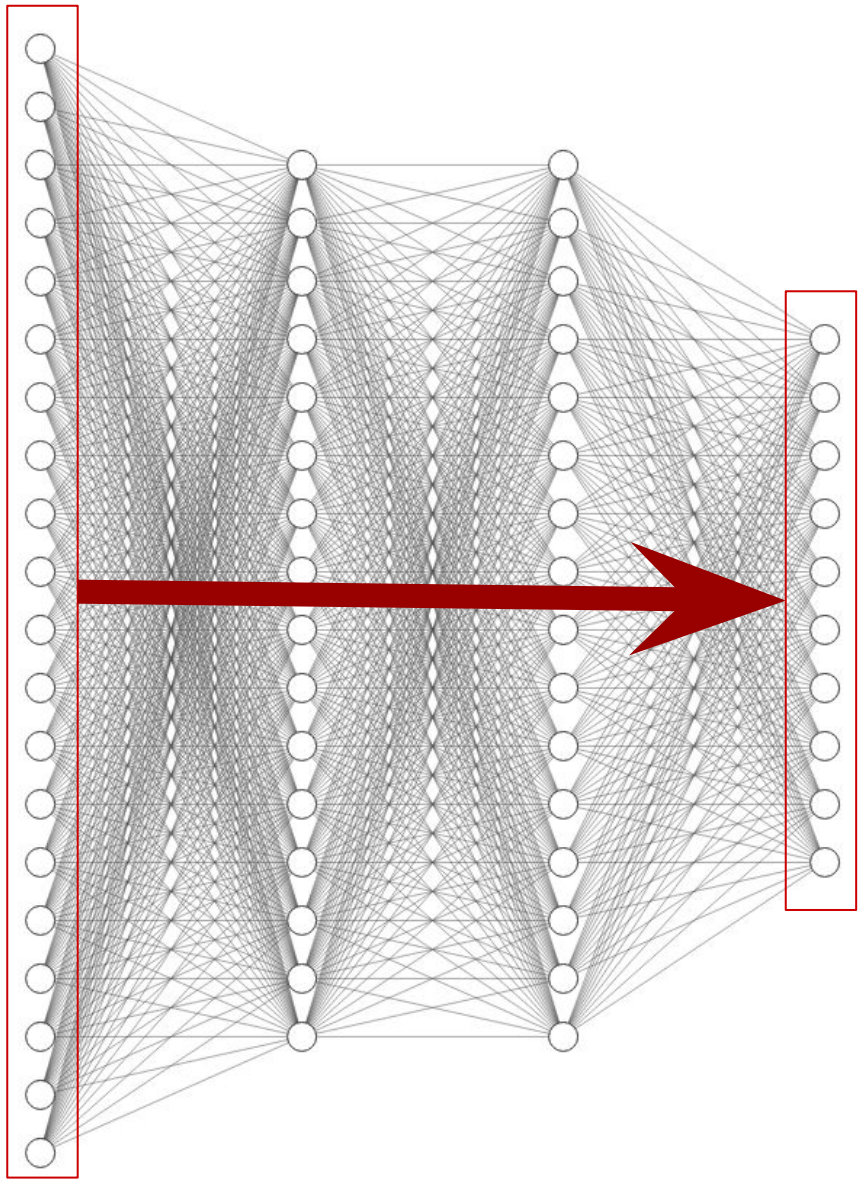
$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

$$\frac{\partial C}{\partial w^l_{j\,k}} = a^{l-1}_k \delta^l_j$$

Back Propagation

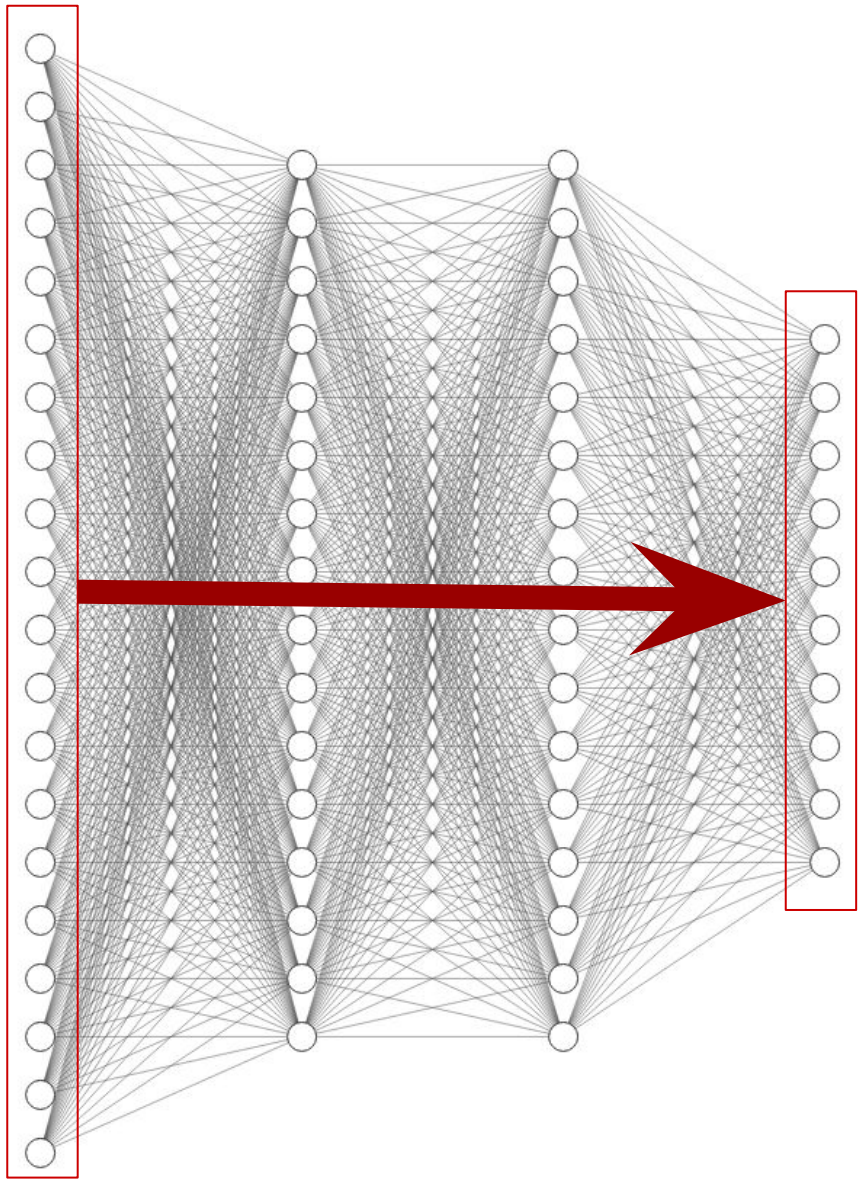
784 pixels



0.1	0
0.12	0
0	0
0.21	1
0.1	0
0.14	0
..	..
..	..
..	..
..	..
0.11	0

Back Propagation

784 pixels



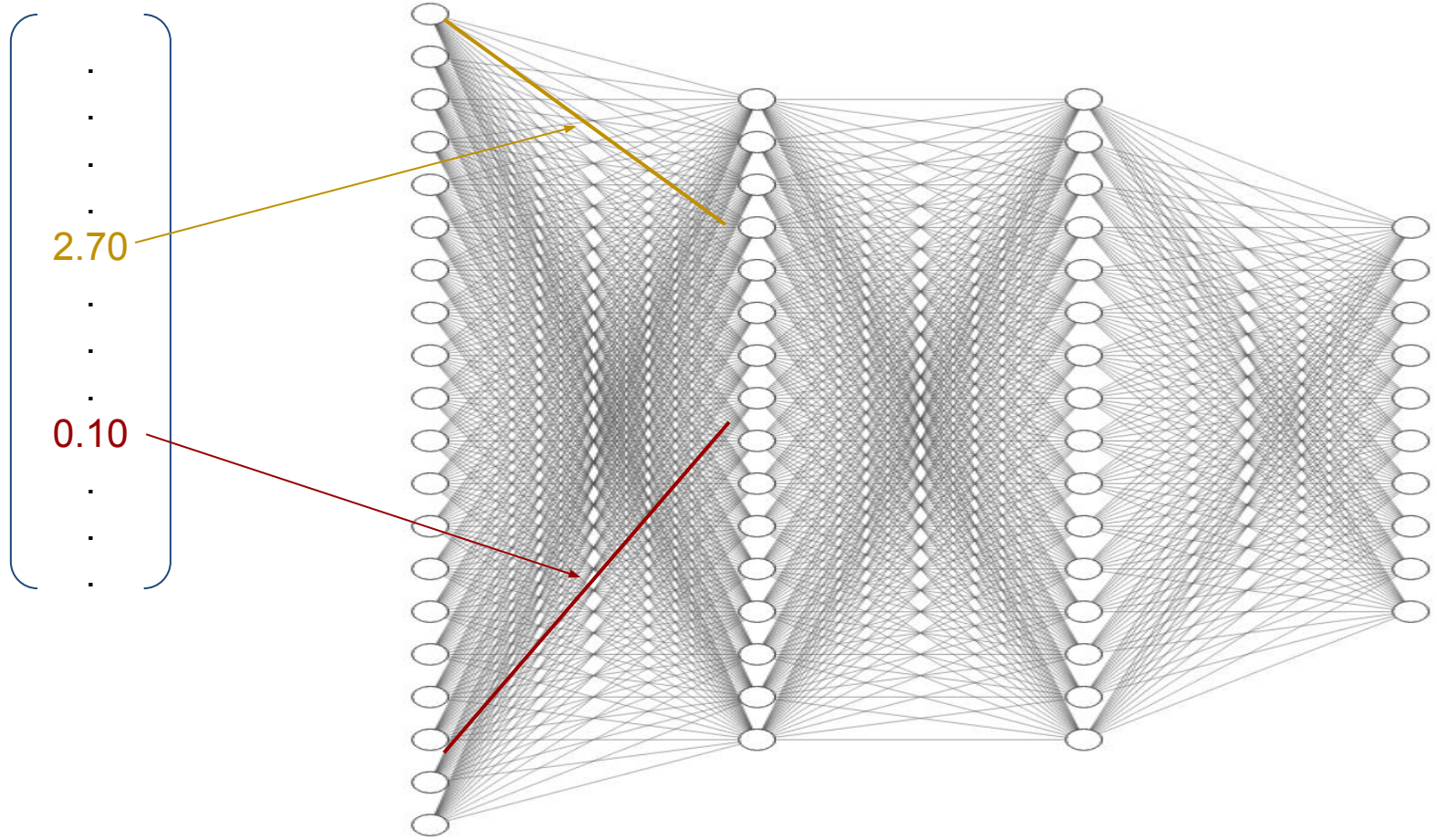
At a high level:

- Compute Error: Cost Function
- Compute the negative of the gradient for every weight of the net on every input-output pair
- Modify Weights
- Repeat

Back Propagation

$-\nabla C(...) =$

All weights and biases

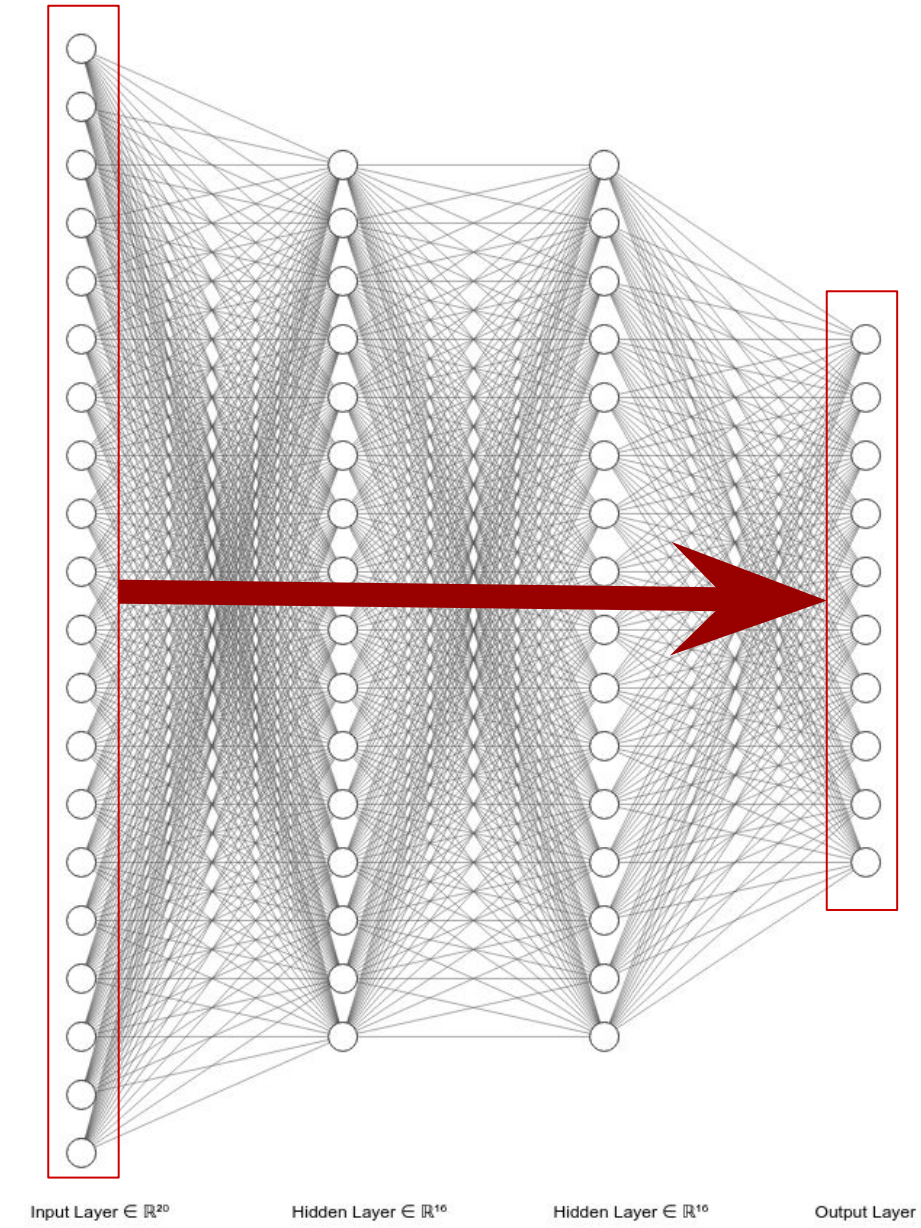


Let us assume we know how to compute gradients:

- Each gradient indicates how sensitive the cost function is to each one of the weights
- Yellow weight has 32 times more impact on const function than red weight

Back Propagation

784 pixels

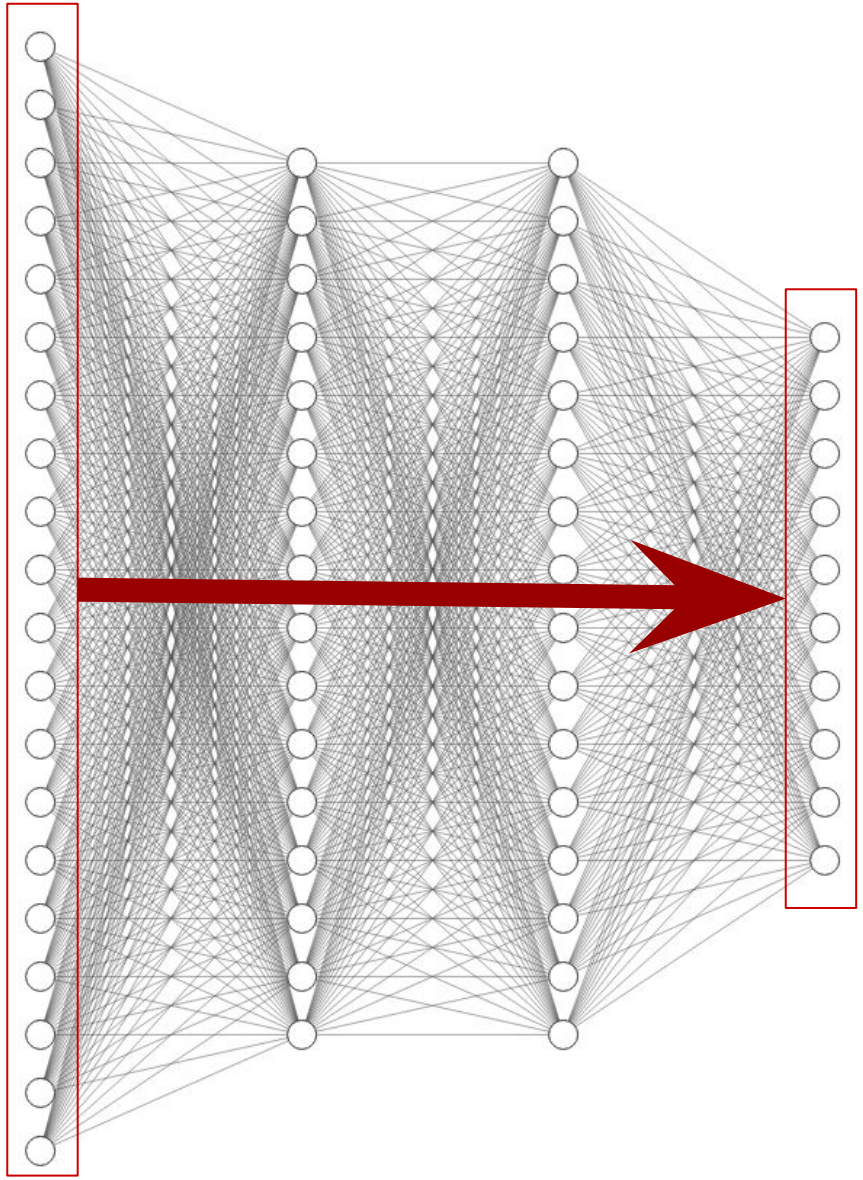


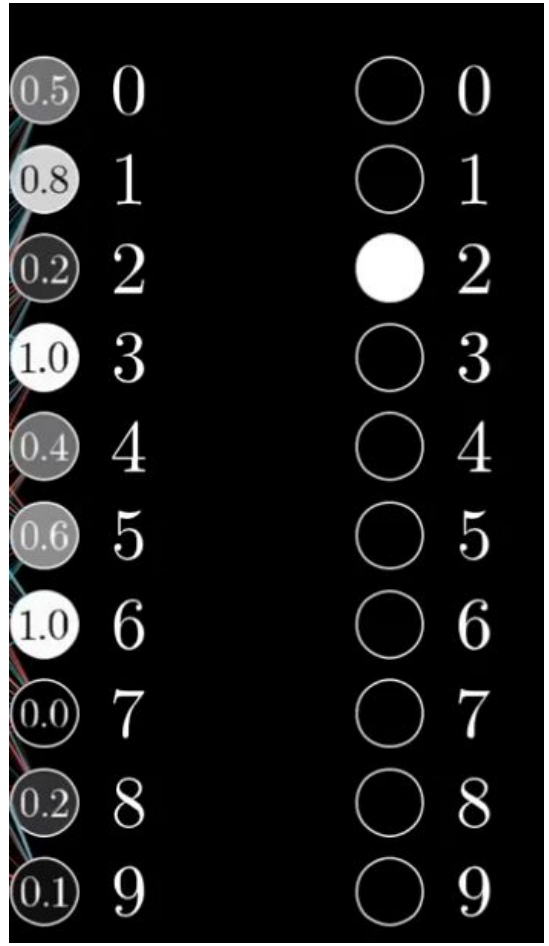
At a high level:

- How can we modify a neuron activation?
  - Weights
  - Previous Activations
- Can we modify Previous Activations?

Back Propagation - Single Input

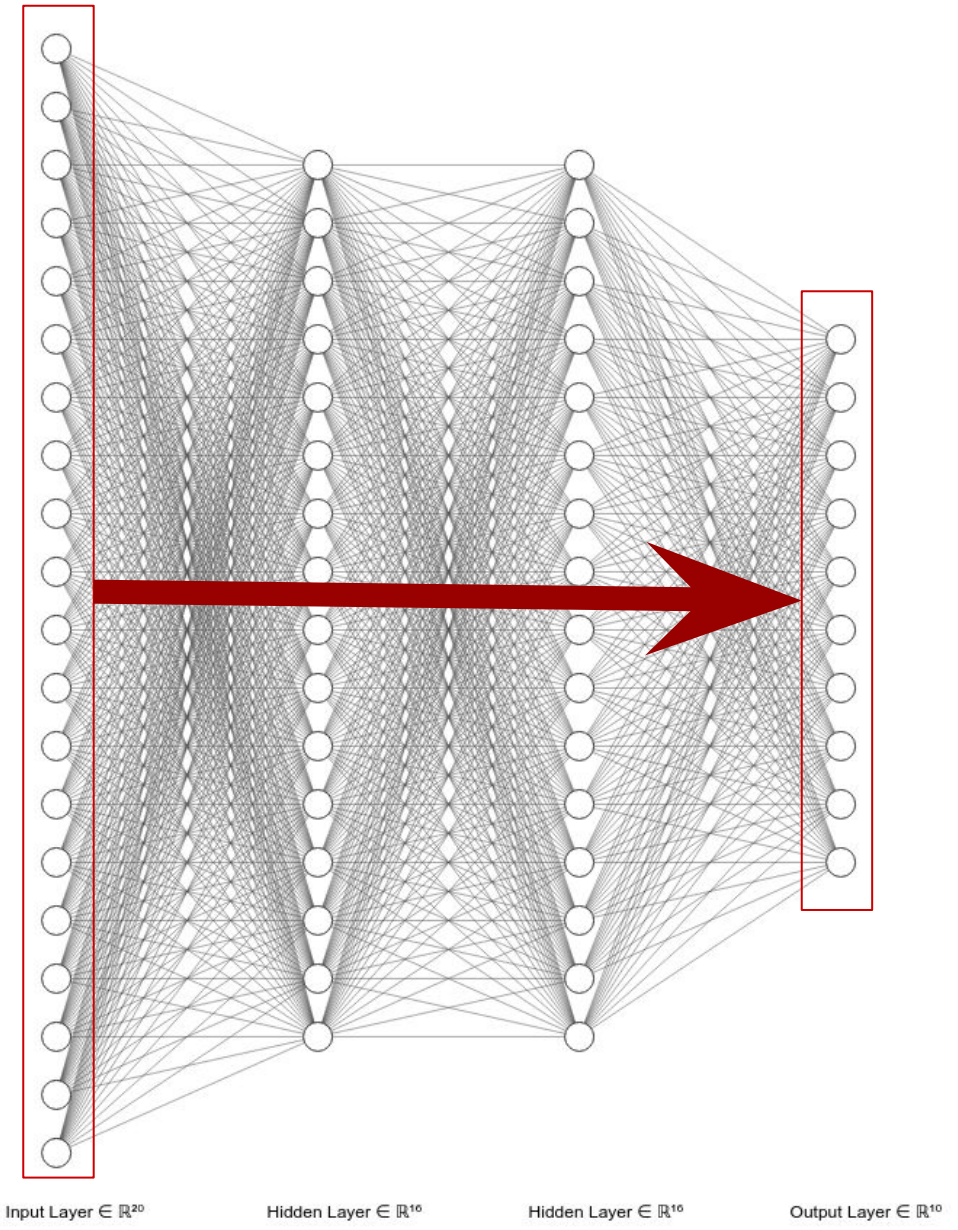
784 pixels

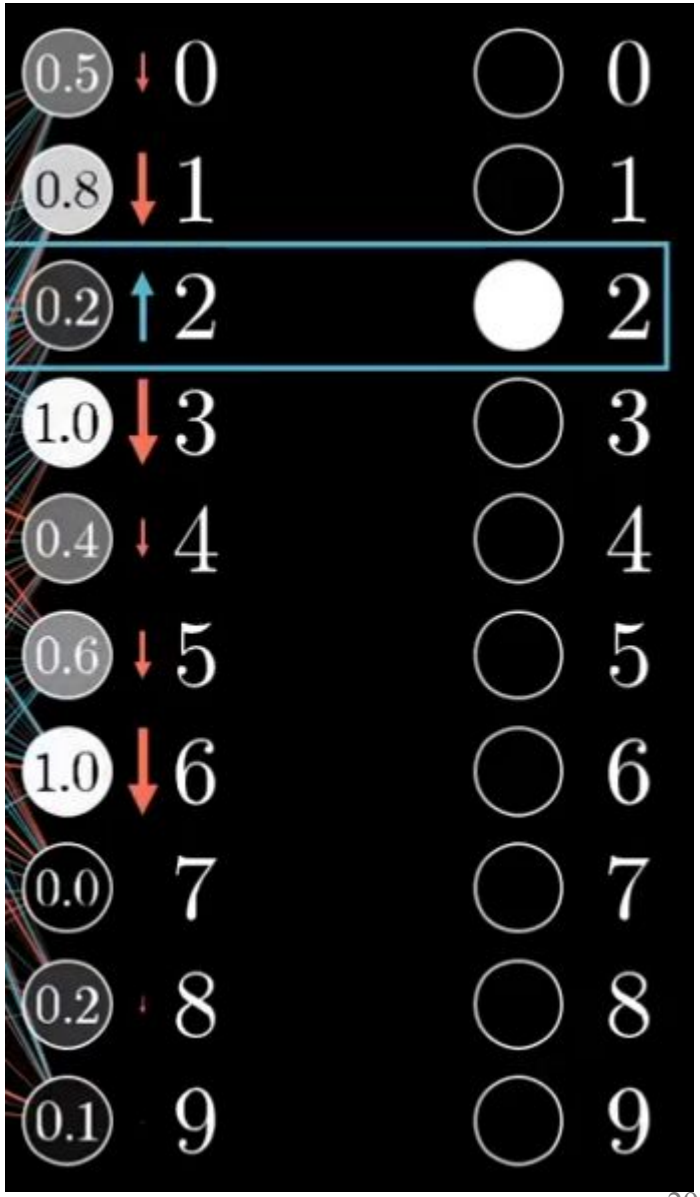




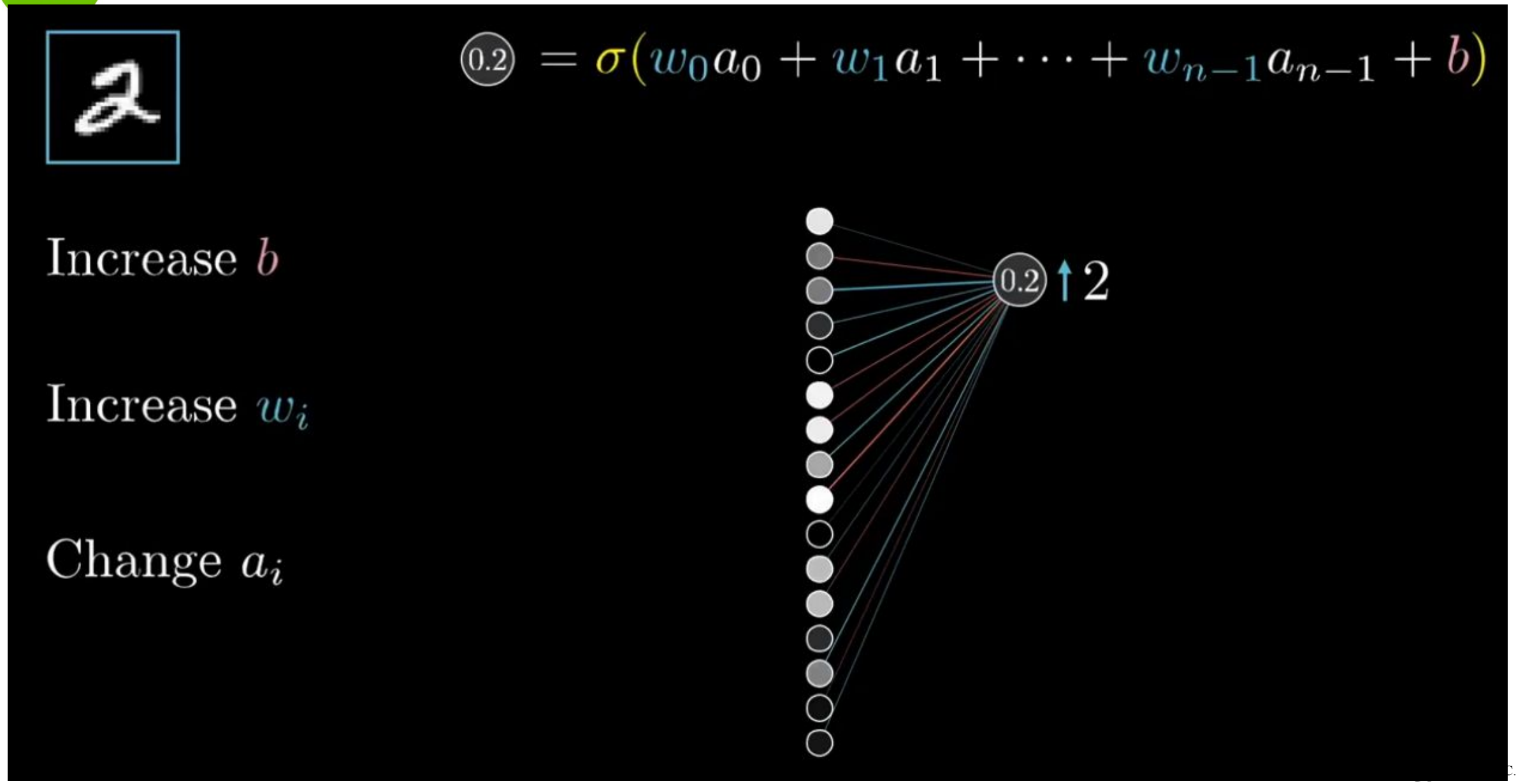
Back Propagation - Single Input

784 pixels

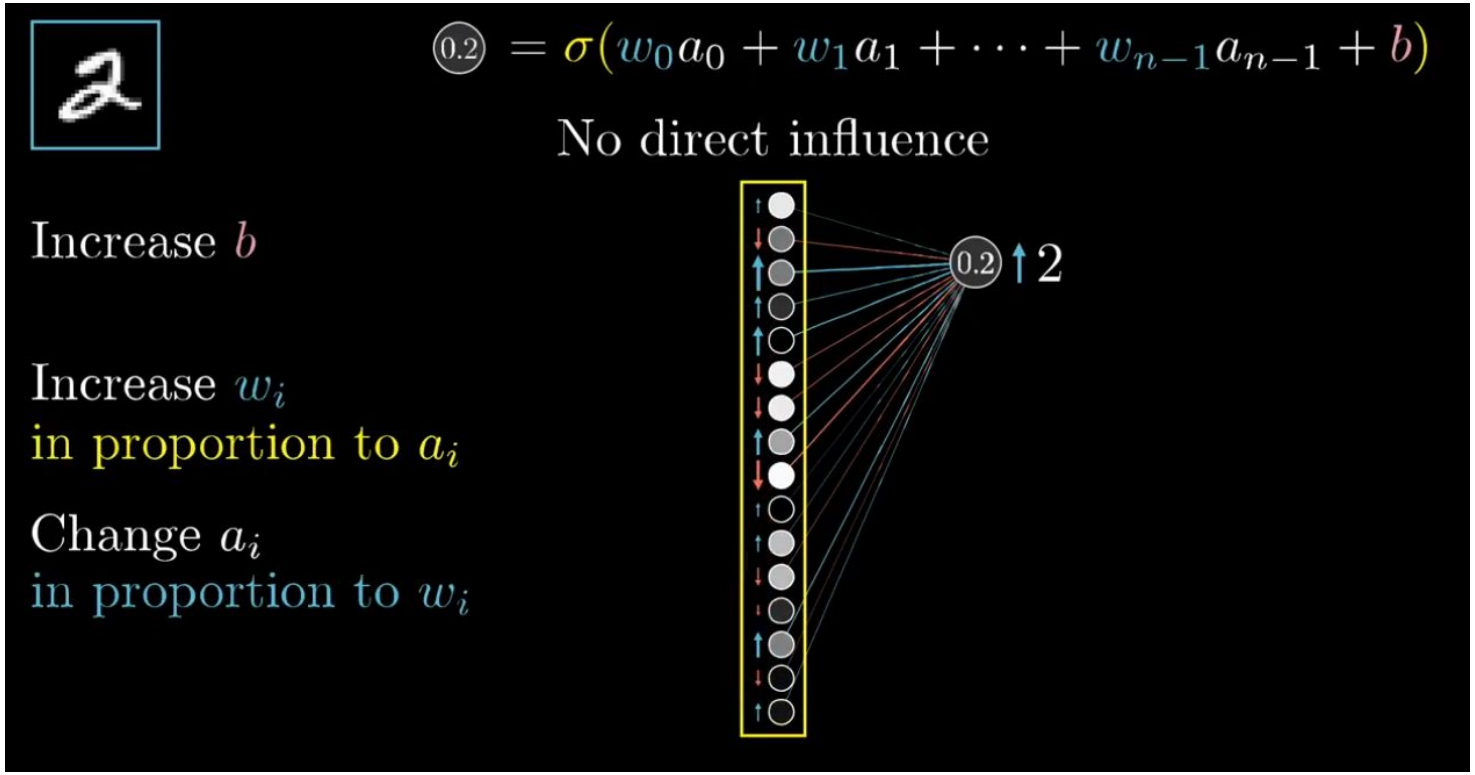




Back Propagation - Single Input



Back Propagation - Single Input



Changing weights:

- Weights connecting neurons with higher values have more impact

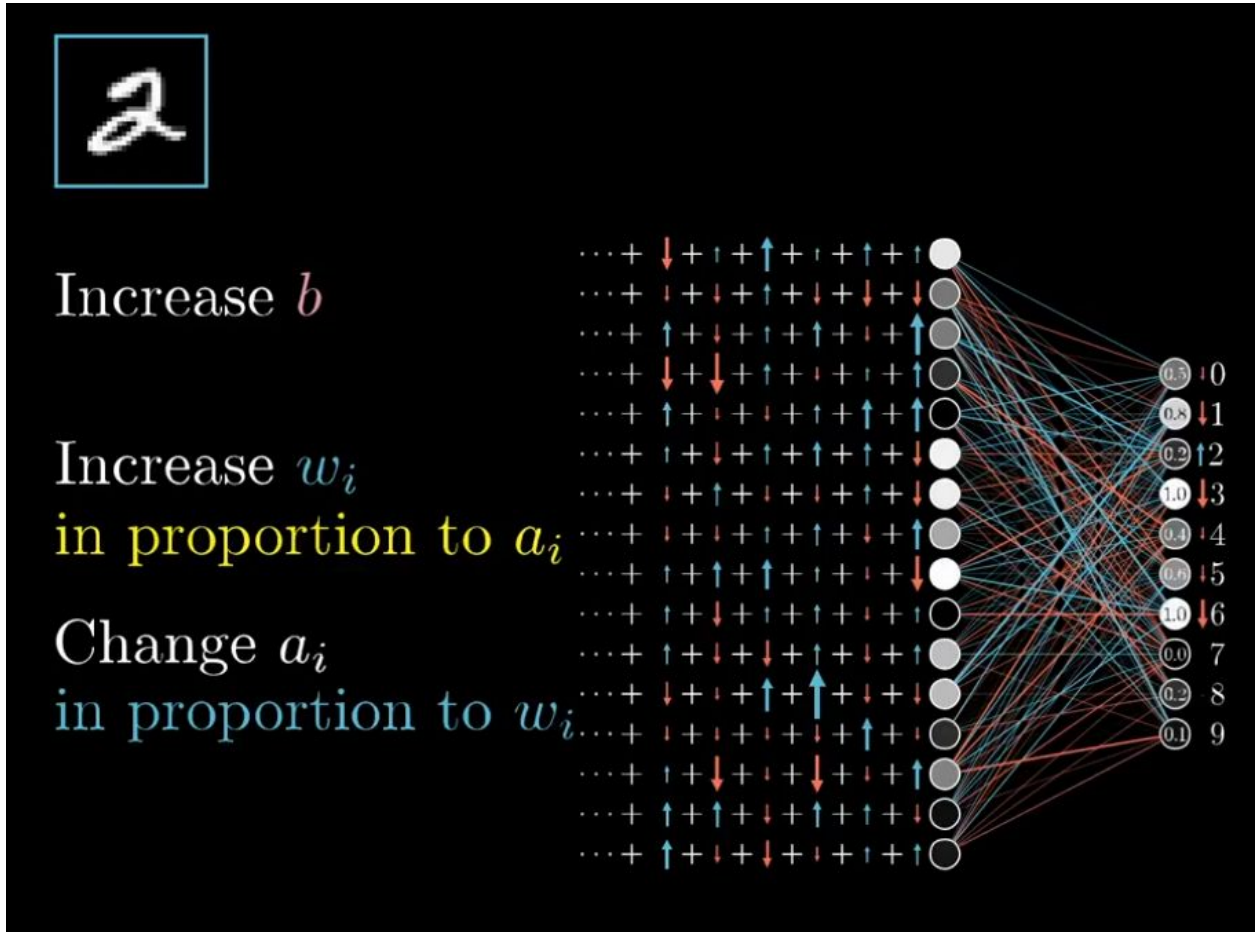
Changing bias:

- Affects all data, cannot be adjusted for a single input

Changing Activations:

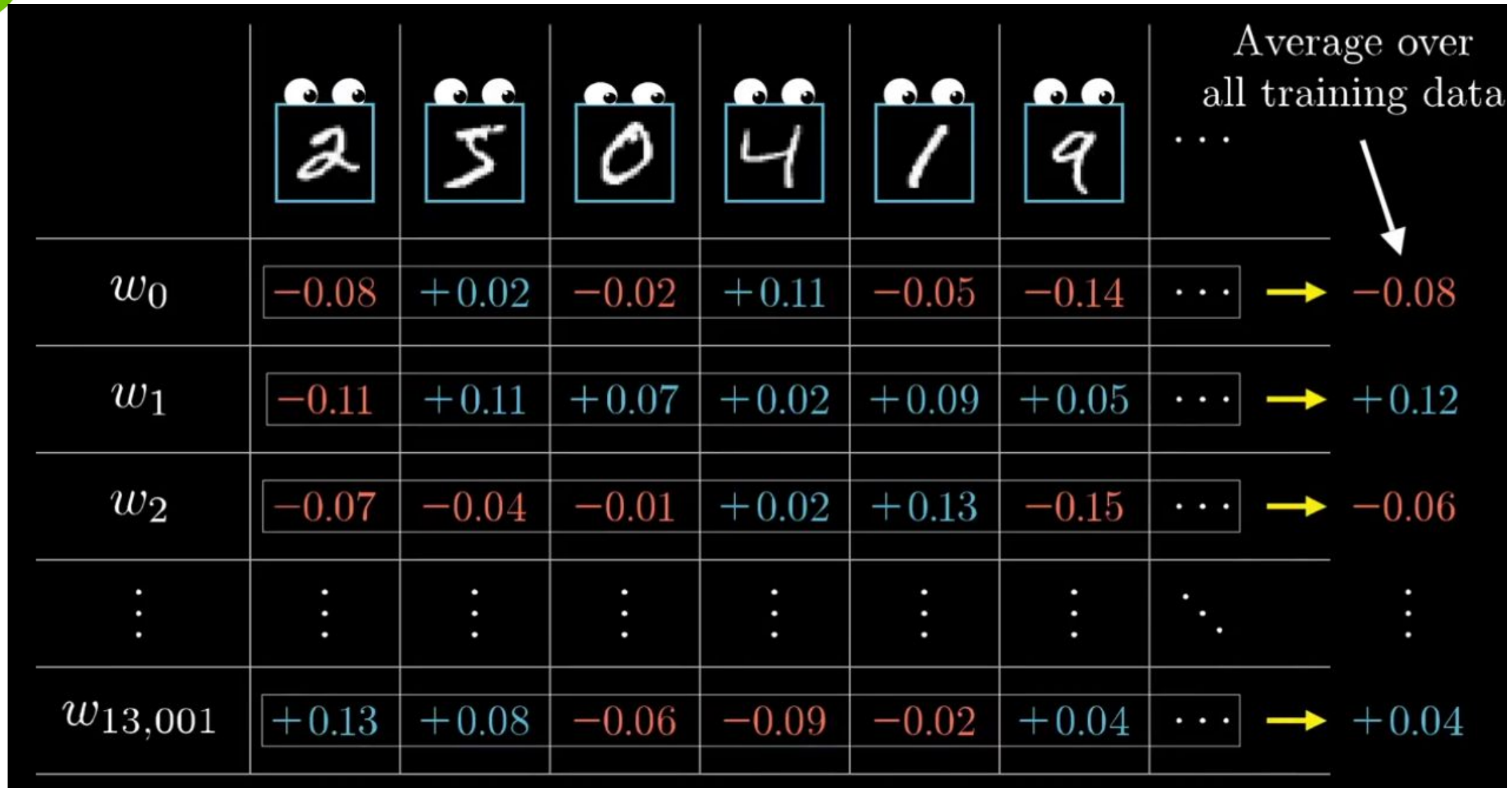
- We do not have direct influence but we note down how we would like to change them

Back Propagation - Single Input

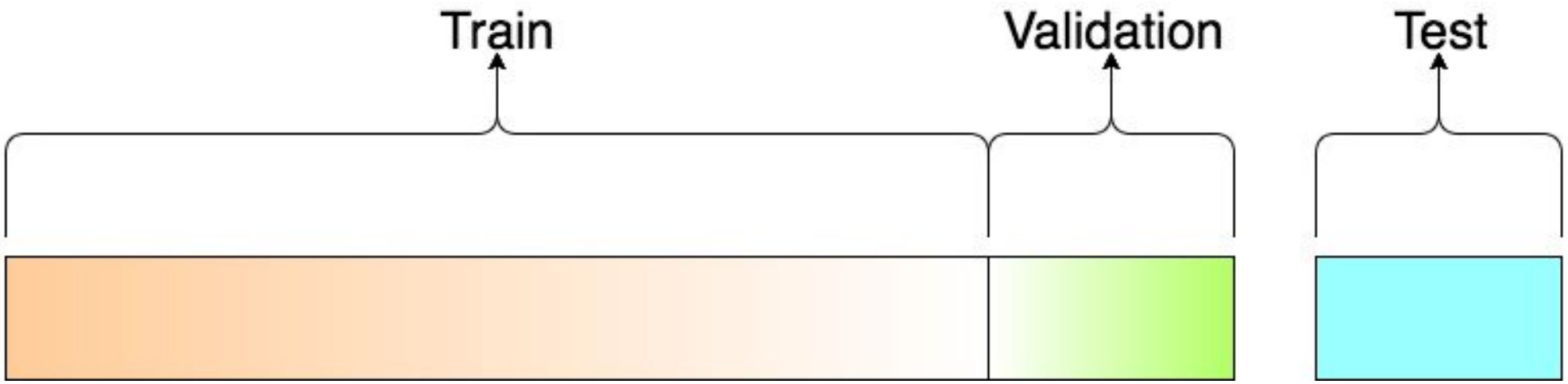


Watch out! Every neuron has its own idea of how the rest of the neurons in previous layer should be changed… Backward Pass!
But… of course… this is only one input-output… we do not want all numbers to be twos!

Back Propagation - Single Input



Data Workflow

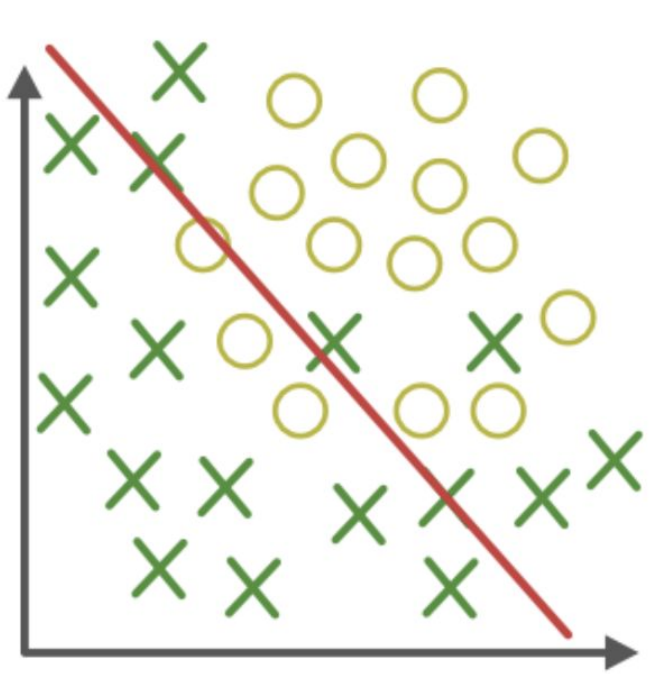


Train
- Adjust parameters

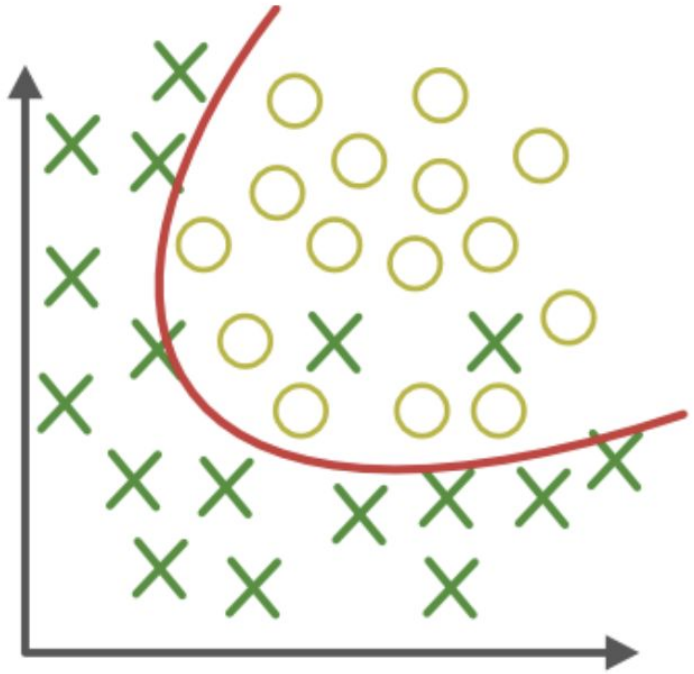
Validation
- Adjust hyper parameters

Test
- Kept as if it was gold!

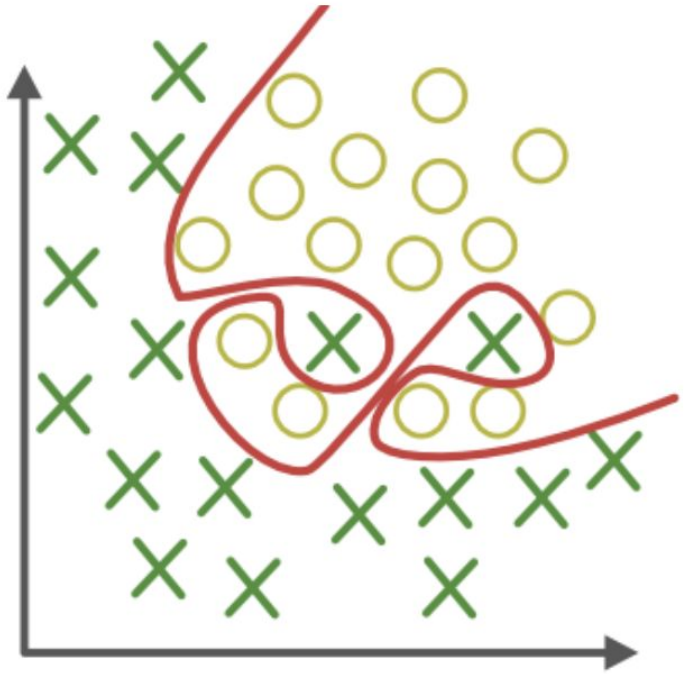
Bias & Variance



High Bias

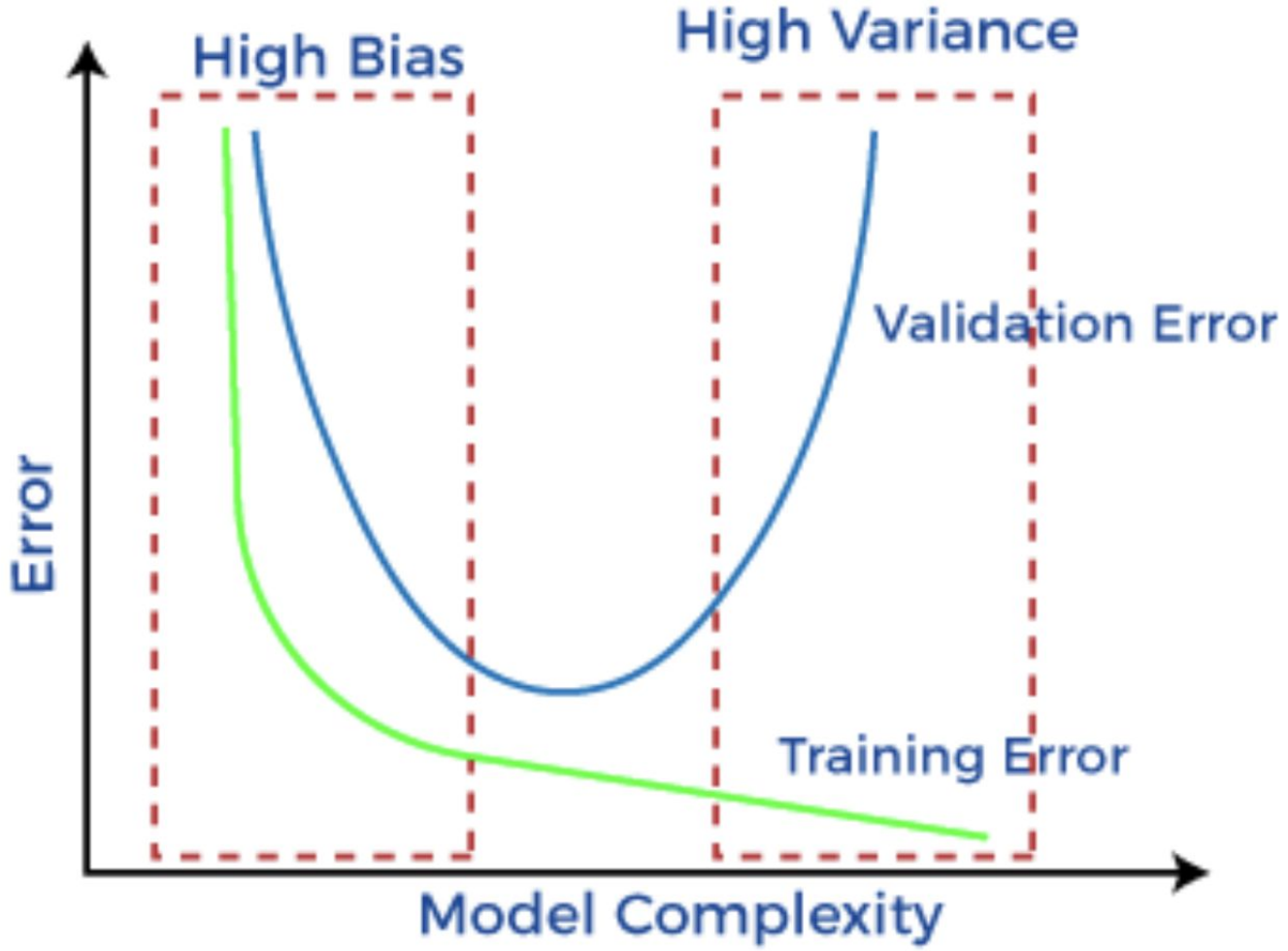


OK



High Variance

Bias & Variance



Bias & Variance

Training error	Dev/Test error	Scenario
High	High	High Bias
High	Low	What did you do? ☺
Low	High	High Variance
Low	Low	Ok

- **High Bias (looking at the train set performance you have a high error)**
  - Is your model too simple? Try a more complex model. Increase hidden units and hidden layers.
  - Train longer or try some advance optimization algorithm.
  - Are you using the appropriate DNN architecture? Are DNN your best bet?

Bias & Variance

Training error	Dev/Test error	Scenario
High	High	High Bias
High	Low	What did you do? ☺
Low	High	High Variance
Low	Low	Ok

- **High Variance (looking at the dev set performance you have a high error)**
  - Do you need more data? Try more data
  - Reduce the complexity of your model
  - Regularization

Bias & Variance

- **High Bias (looking at the train set performance you have a high error)**
  - Is your model too simple? Try a more complex model. Increase hidden units and hidden layers.
  - Train longer or try some advance optimization algorithm.
  - Are you using the appropriate DNN architecture? Are DNN your best bet?

- **High Variance (looking at the dev set performance you have a high error)**
  - Do you need more data? Try more data
  - Reduce the complexity of your model
  - Regularization

Regularization

- **L2 and L1 regularization**

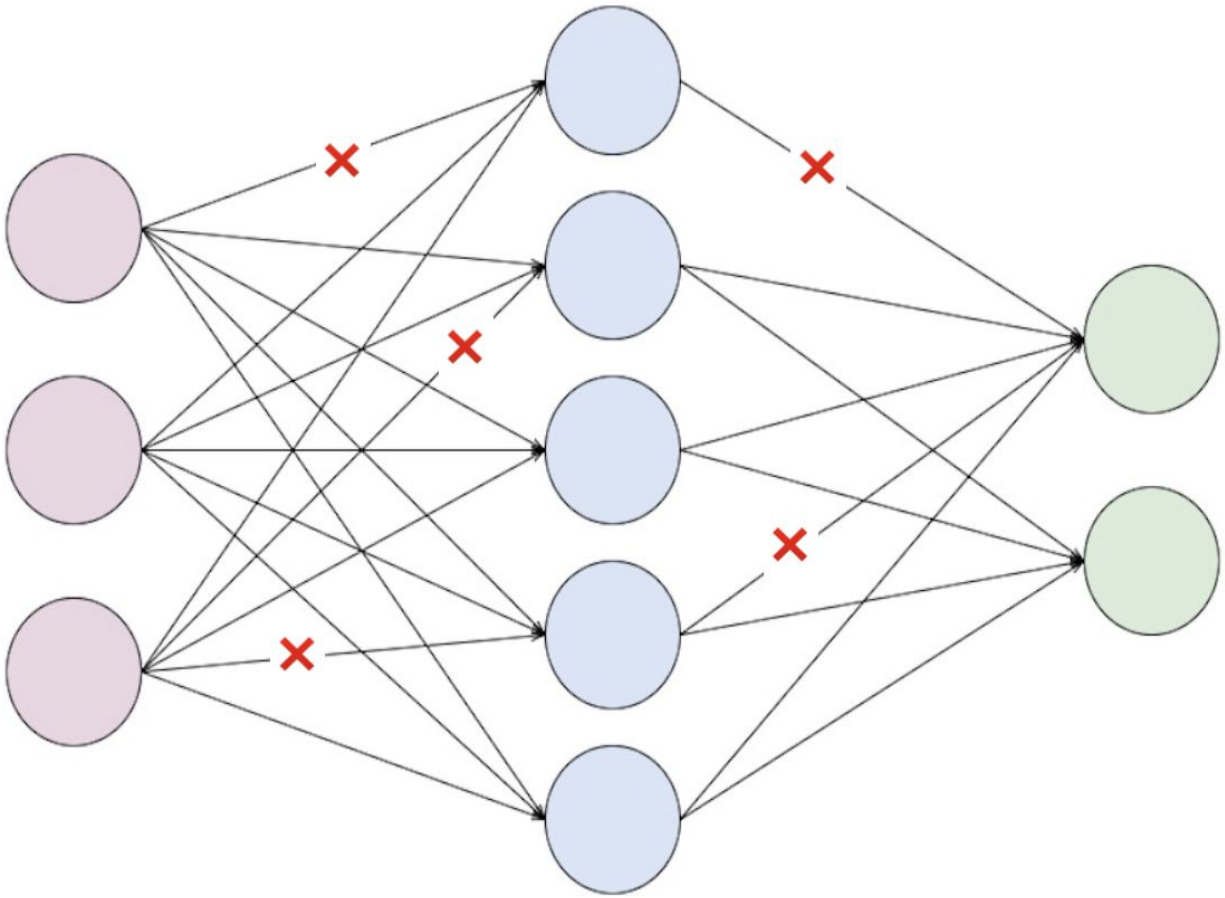
$$L2 \rightarrow \quad L(w) = 1/2M \sum (h_w(x)^{(i)} - y^{(i)})^2 + \lambda/2M \sum w^2$$

$$L1 \rightarrow \quad L(w) = 1/2M \sum (h_w(x)^{(i)} - y^{(i)})^2 + \lambda/2M \sum |w|$$

  - The main effect of both is to reduce the value of the weights, as the larger the weights the larger the loss.
  - You can also regularize the bias, but it is much more significant to regularize the weights as we have much more weights than biases.

Regularization

- **Why L1/L2 works?**



Regularization

- **L2 and L1 regularization in Keras/Tensorflow**

https://keras.io/api/layers/regularizers/

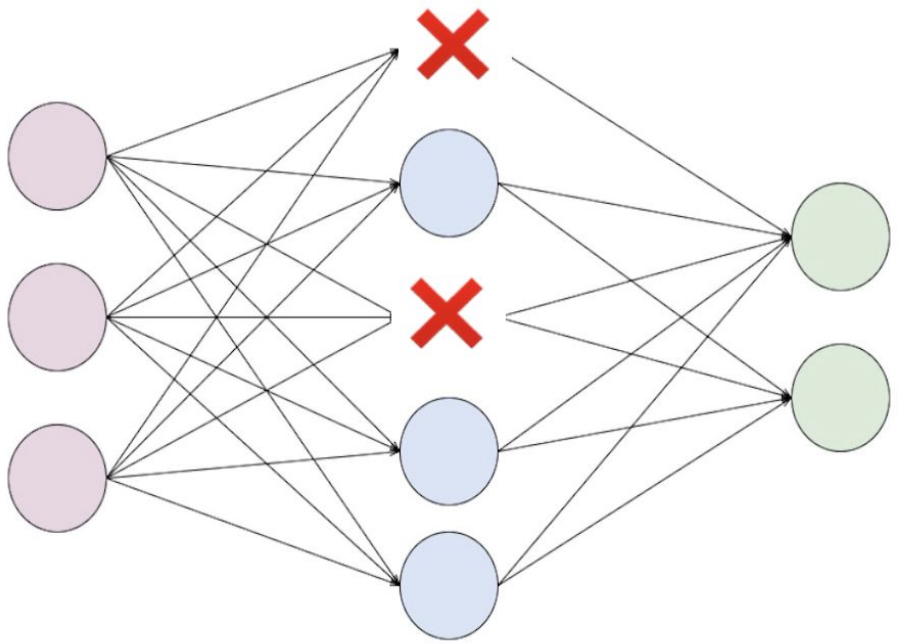
```
from tensorflow.keras import layers
from tensorflow.keras import regularizers

layer = layers.Dense(
    units=64,
    kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
    bias_regularizer=regularizers.L2(1e-4),
    activity_regularizer=regularizers.L2(1e-5)
)
```

Regularization

- **Dropout**

In each iteration of gradient descent, we randomly remove some neurons with a given probability.



Prob to dropout = 0.5

Regularization

- **Why dropout works?**
  - By removing randomly some inputs, the neurons can't rely just in some of them. We are forcing the neuron to exploit all the input knowledge.
  - Somehow, we are better distributing the information across the whole network. All the neurons count, as some of them are forced to live without others.
- **Some notes**
  - Rarely is applied in the input layer or the output layer for obvious reasons.
  - Dropout is applied on training time. In testing time, we DO NOT drop out neurons.

Regularization

- **Dropout in Keras/Tensorflow**

https://keras.io/api/layers/regularization_layers/dropout/

```
tf.keras.layers.Dropout(rate, noise_shape=None, seed=None, **kwargs)
```