L12 - Statistical Mechanics 4

Machine learning

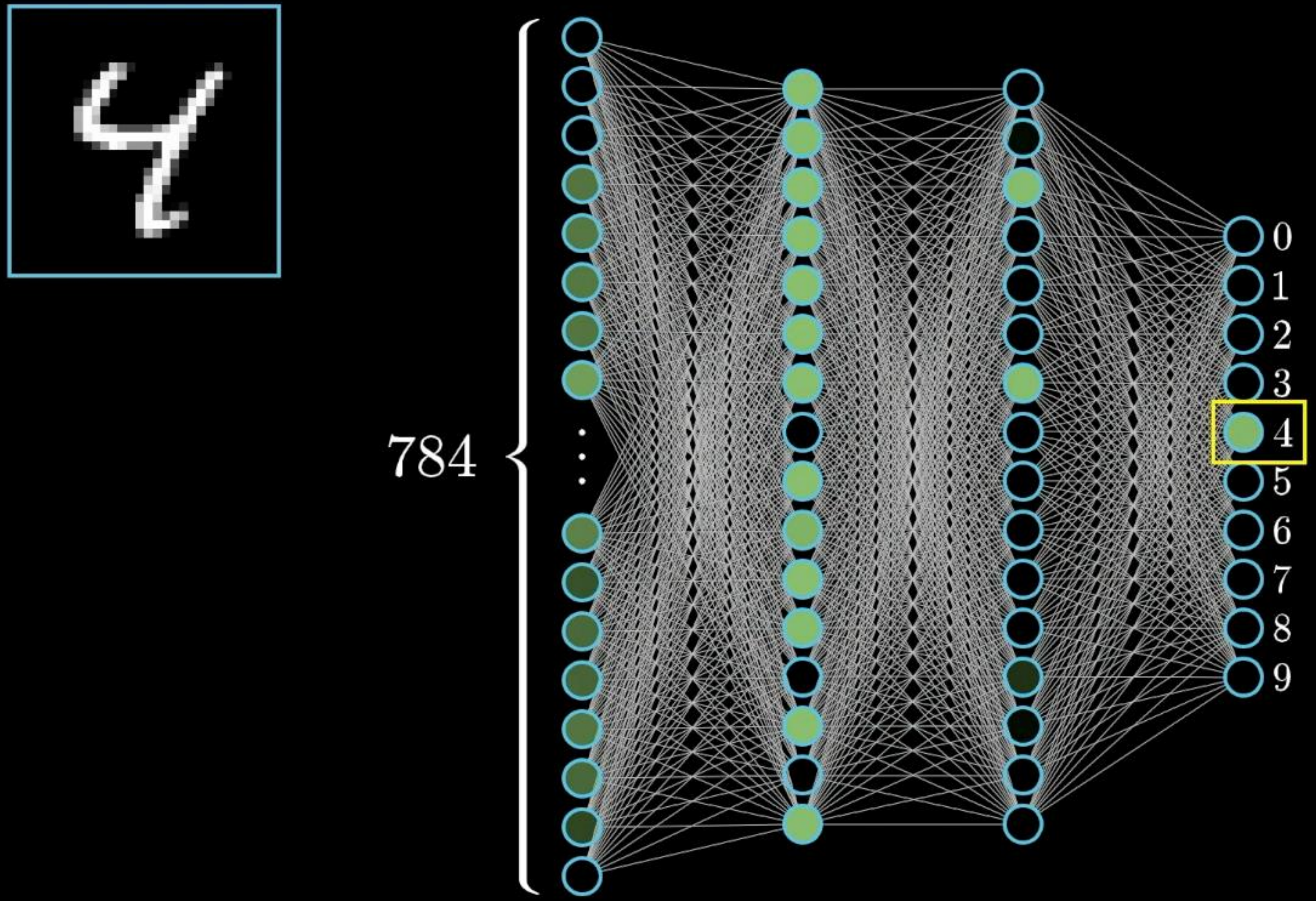
Neural networks

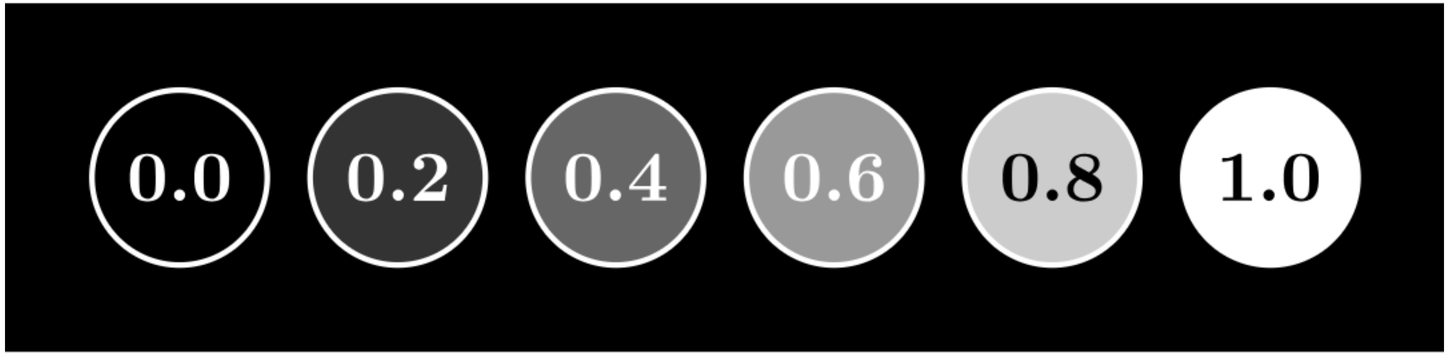
What is a Neural Network?

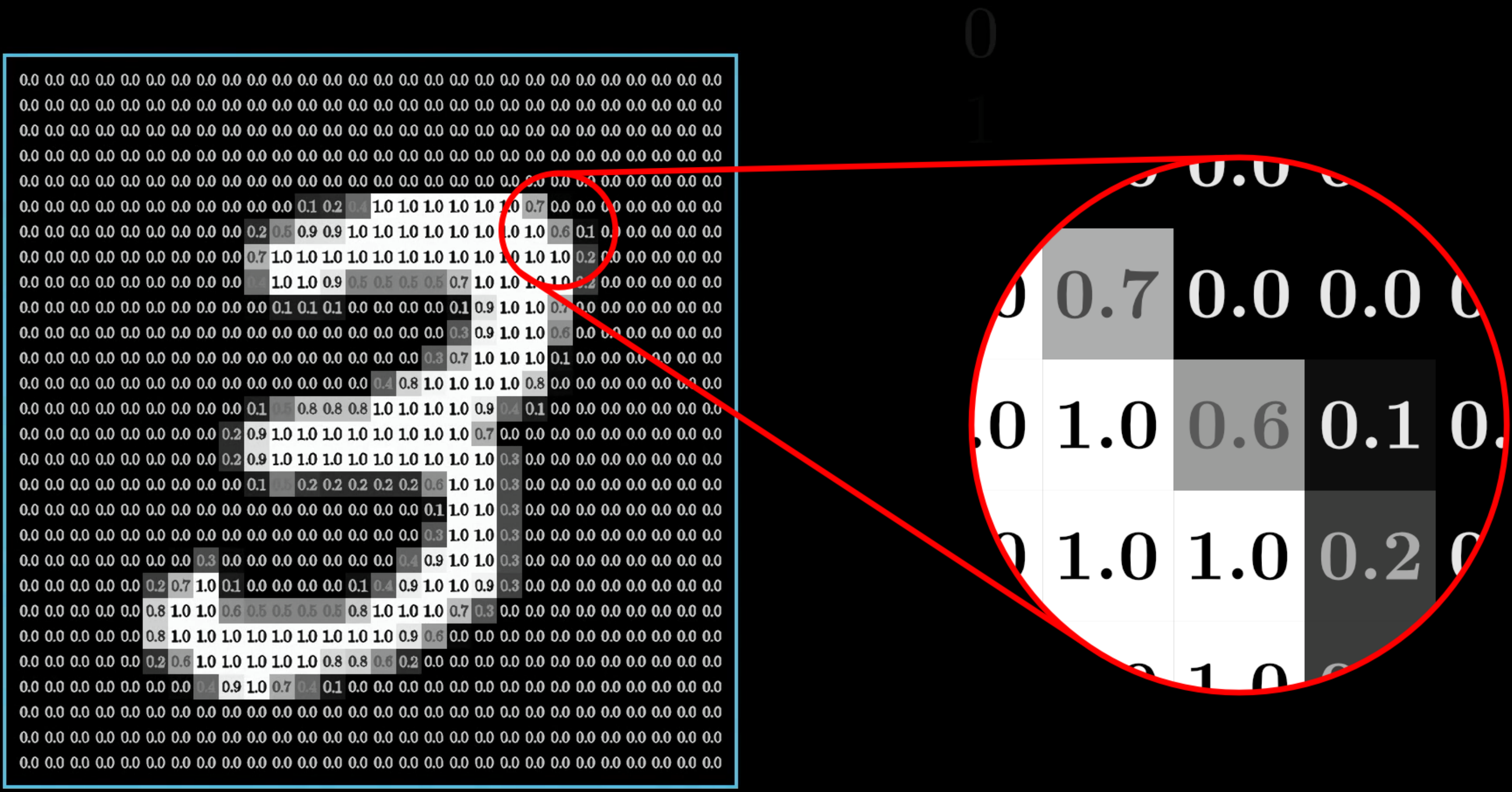
Adapted from:
www.3blue1brown.com/lessons/neural-networks

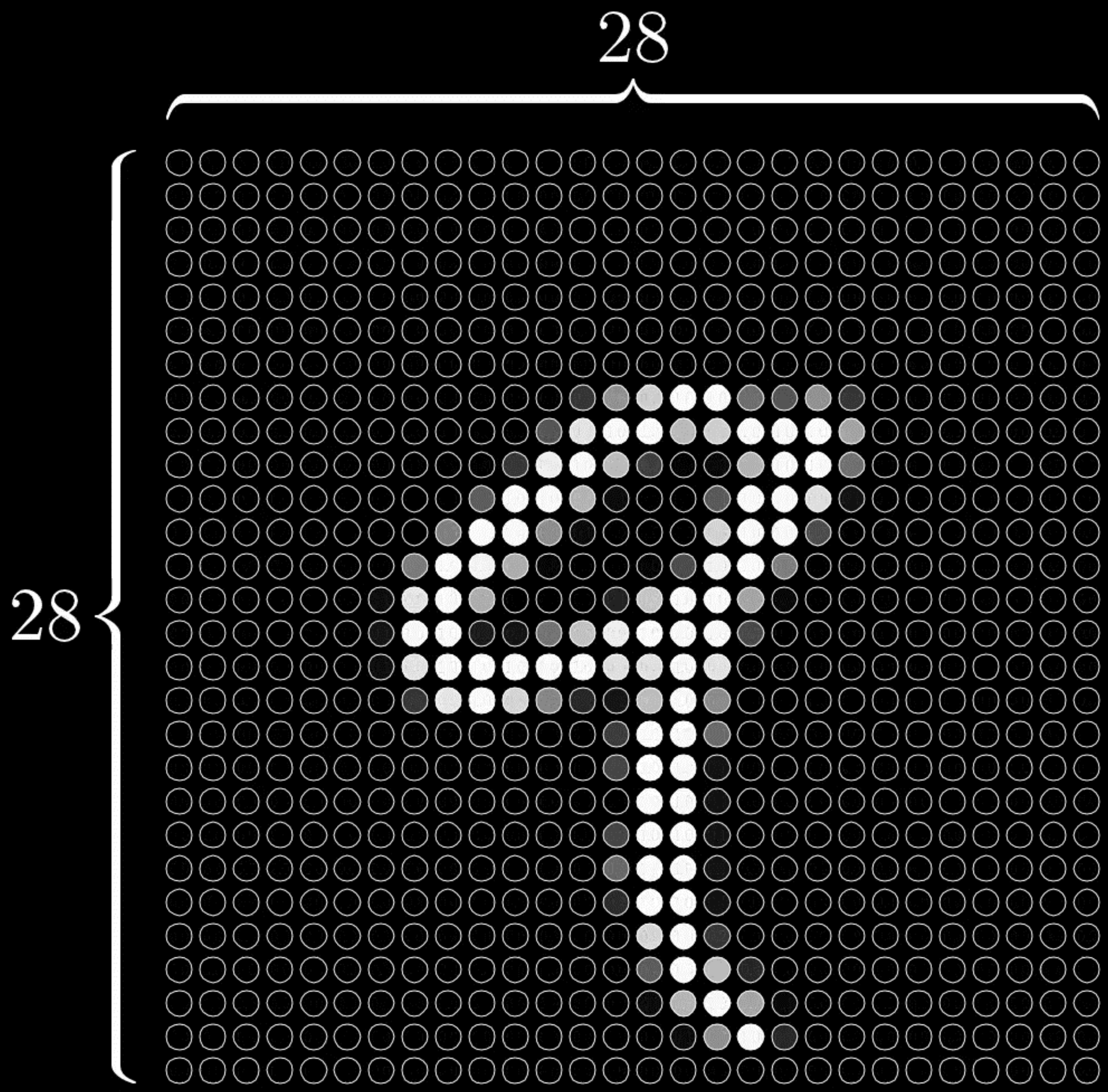
Plain vanilla

(aka "multilayer perceptron")

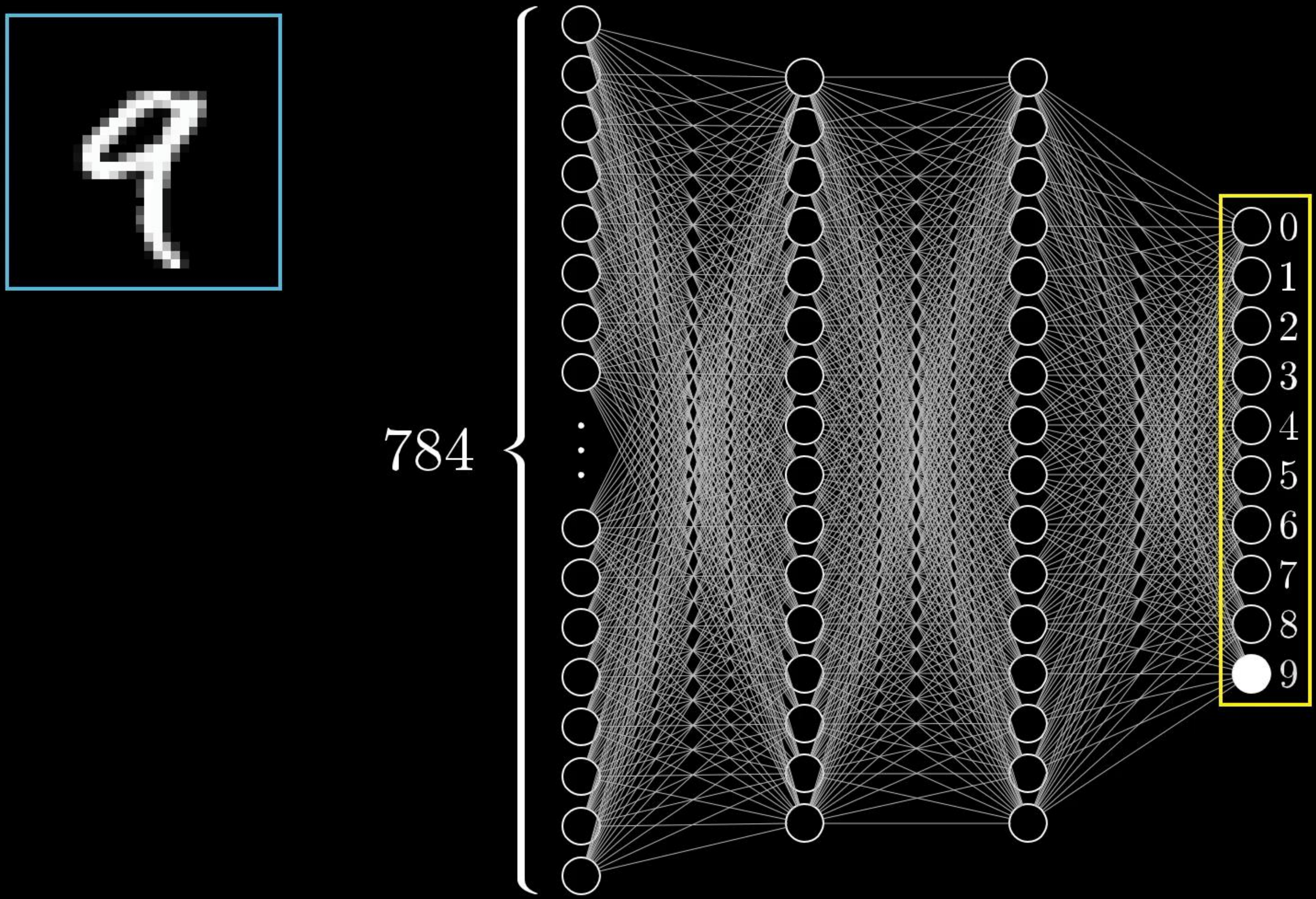


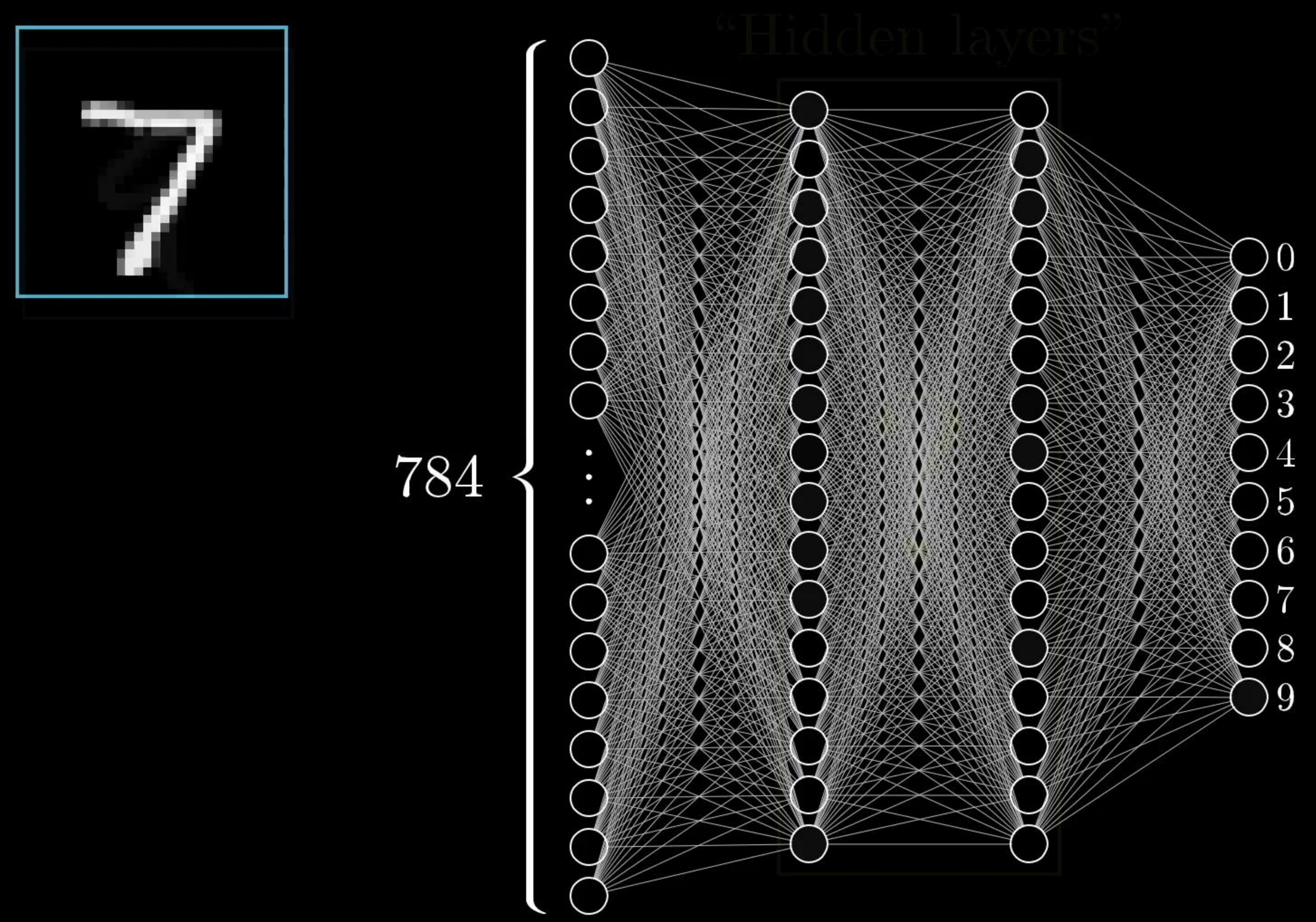

0.0
0.2
0.4
0.6
0.8
1.0


0.7 0.0 0.0
1.0 0.6 0.1
1.0 1.0 0.2



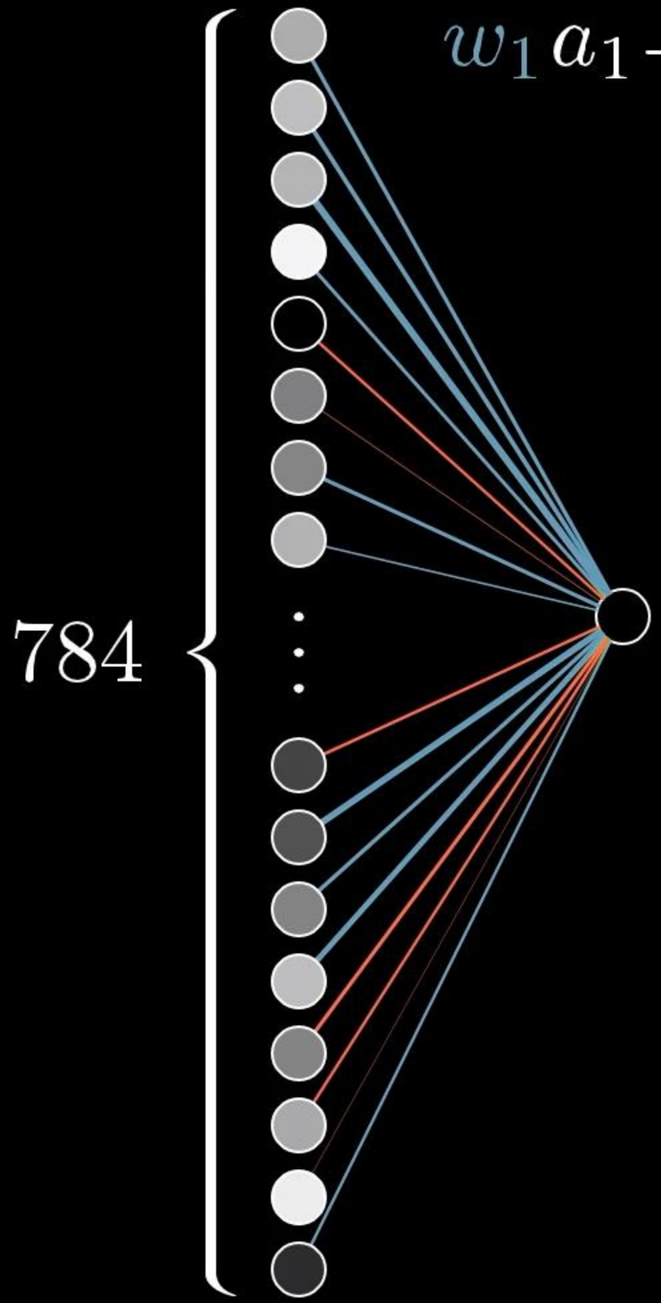
$$28 \times 28 = 784$$

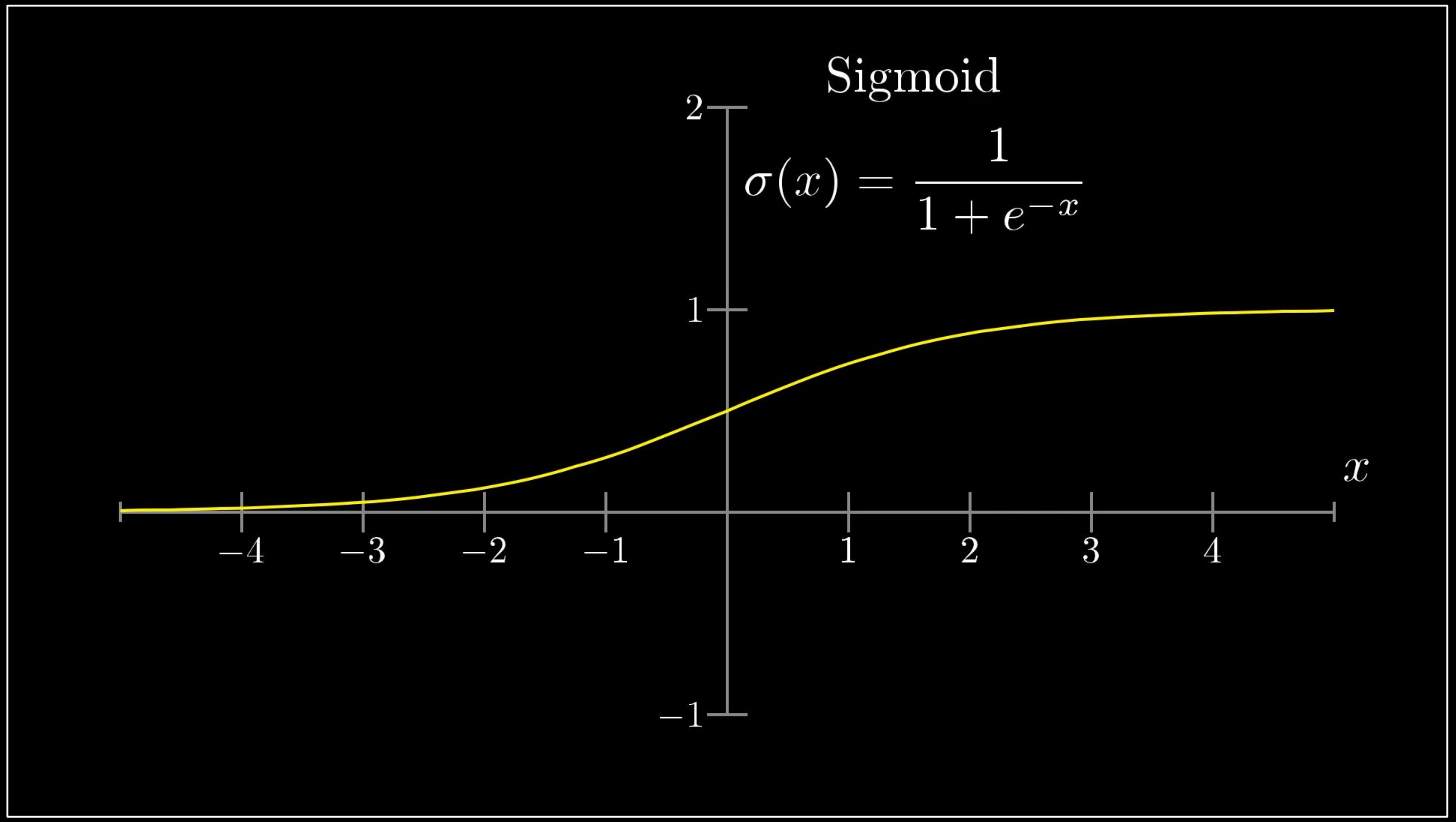

784
0
1
2
3
4
5
6
7
8
9



“Hidden layers”
784
0
1
2
3
4
5
6
7
8
9

$$w_1 a_1 + w_2 a_2 + w_3 a_3 + w_4 a_4 + \cdots + w_n a_n$$

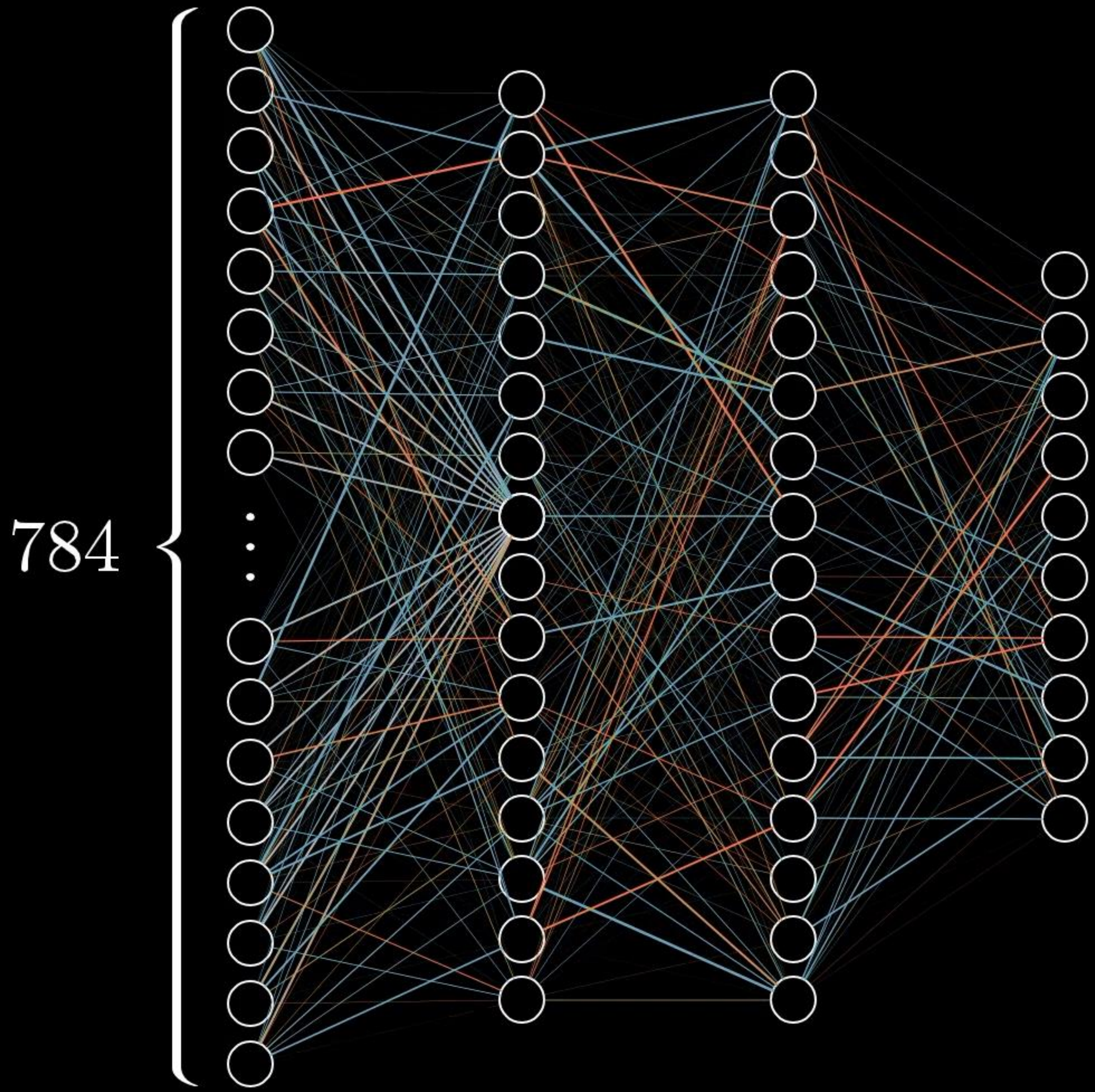




$$\sigma\left(w_1 a_1 + w_2 a_2 + w_3 a_3 + \cdots + w_n a_n \boxed{-10}\right)$$

"bias"

Only activate meaningfully
when weighted sum > 10



$$784\times16+16\times16+16\times10$$

weights

$$16+16+10$$

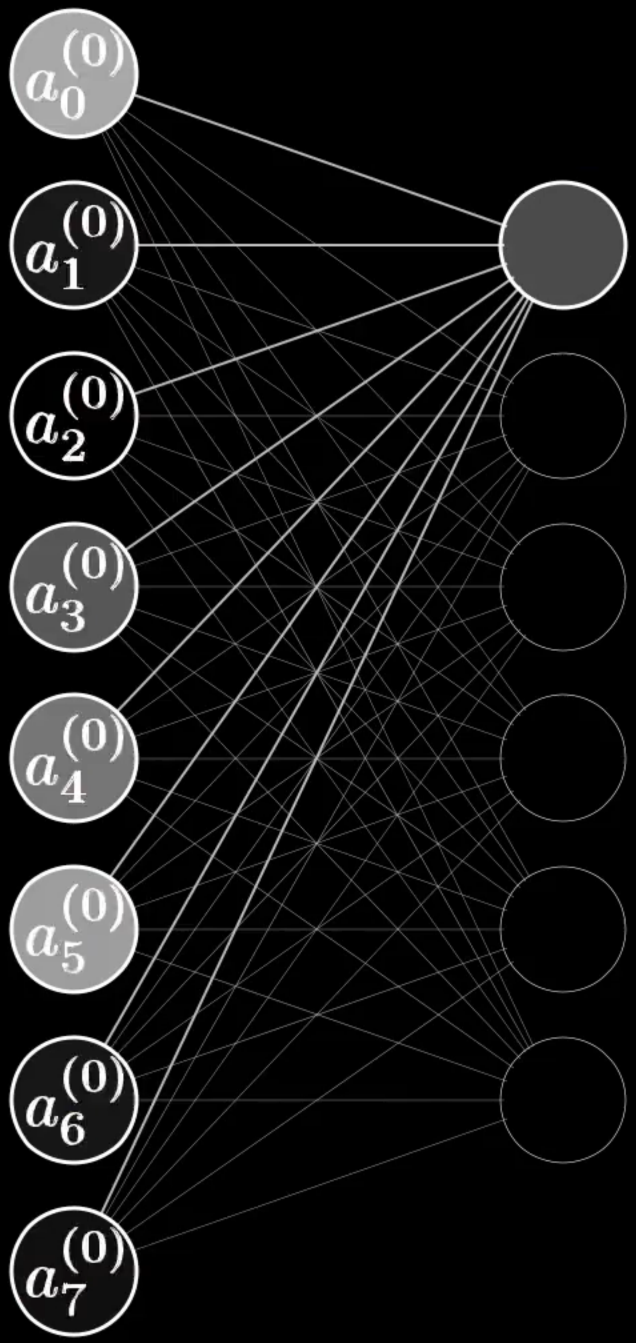
biases

13,002

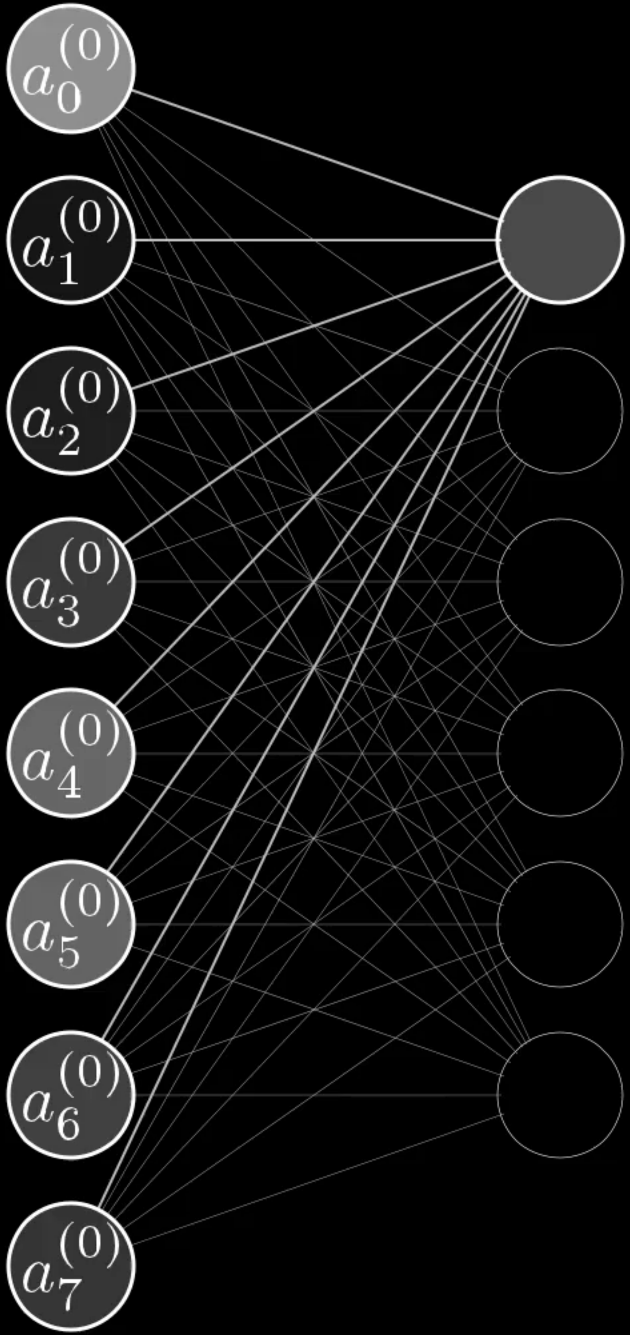
Superscript corresponds to the layer

$$\boldsymbol{a}_0^{(1)} = \sigma\left(w_{0,0}\boldsymbol{a}_0^{(0)} + w_{0,1}\boldsymbol{a}_1^{(0)} + \cdots + w_{0,n}\boldsymbol{a}_n^{(0)} + b_0\right)$$

Subscript corresponds to a neuron in the layer



$$a_0^{(1)} = \overset{\text{Sigmoid}}{\overset{\downarrow}{\sigma}}\left(w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} + \cdots + w_{0,n}\, a_n^{(0)} + \underset{\text{Bias}}{\underset{\uparrow}{b_0}} \right)$$

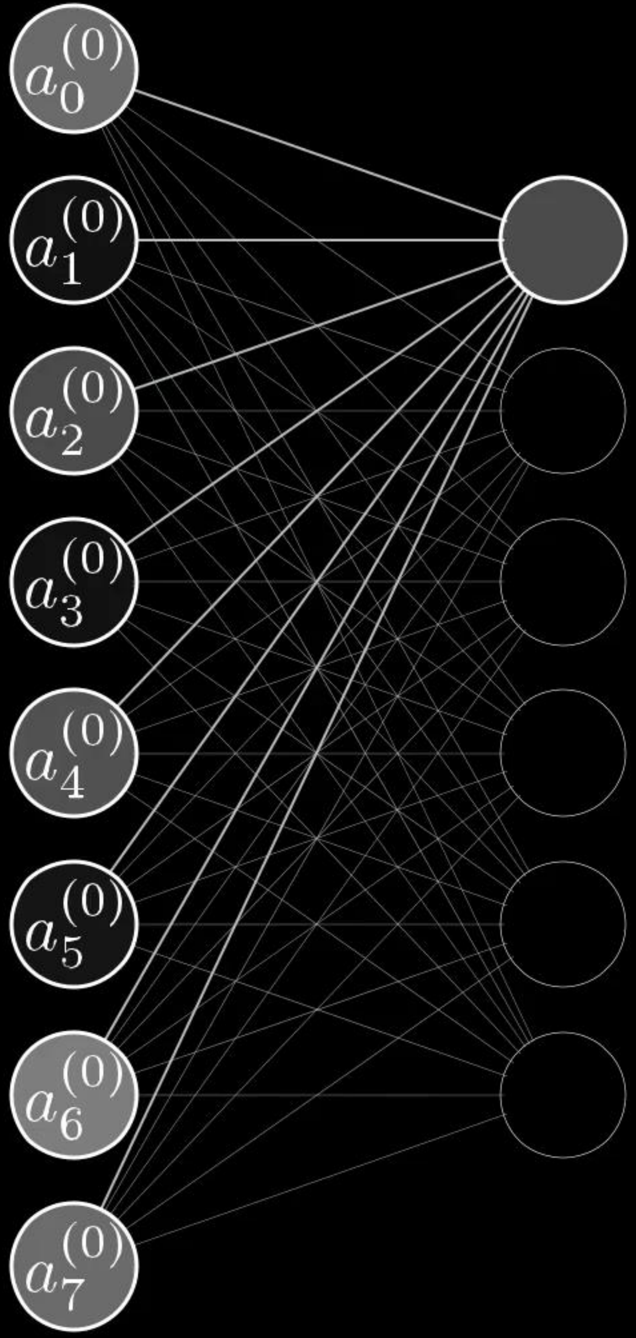


Sigmoid

$$a_0^{(1)} = \sigma\left(w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} + \cdots + w_{0,n}\, a_n^{(0)} + b_0 \right)$$

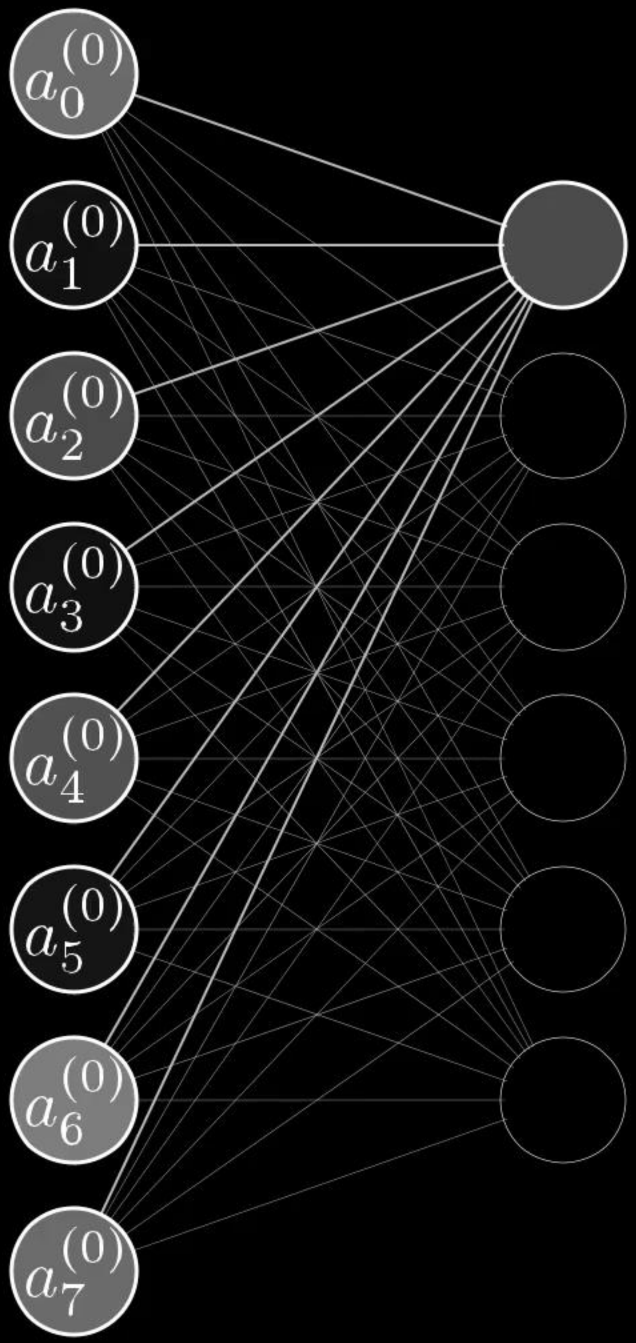
Bias

$$\begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix}$$



$$a_0^{(1)} = \overset{\text{Sigmoid}}{\overset{\downarrow}{\sigma}}\left(w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} + \cdots + w_{0,n}\, a_n^{(0)} + \underset{\text{Bias}}{\underset{\uparrow}{b_0}} \right)$$

$$\begin{bmatrix} \boxed{w_{0,0} \quad w_{0,1} \quad \cdots \quad w_{0,n}} \\ w_{1,0} \quad w_{1,1} \quad \cdots \quad w_{1,n} \\ \vdots \qquad \vdots \qquad \ddots \qquad \vdots \\ w_{k,0} \quad w_{k,1} \quad \cdots \quad w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix}$$

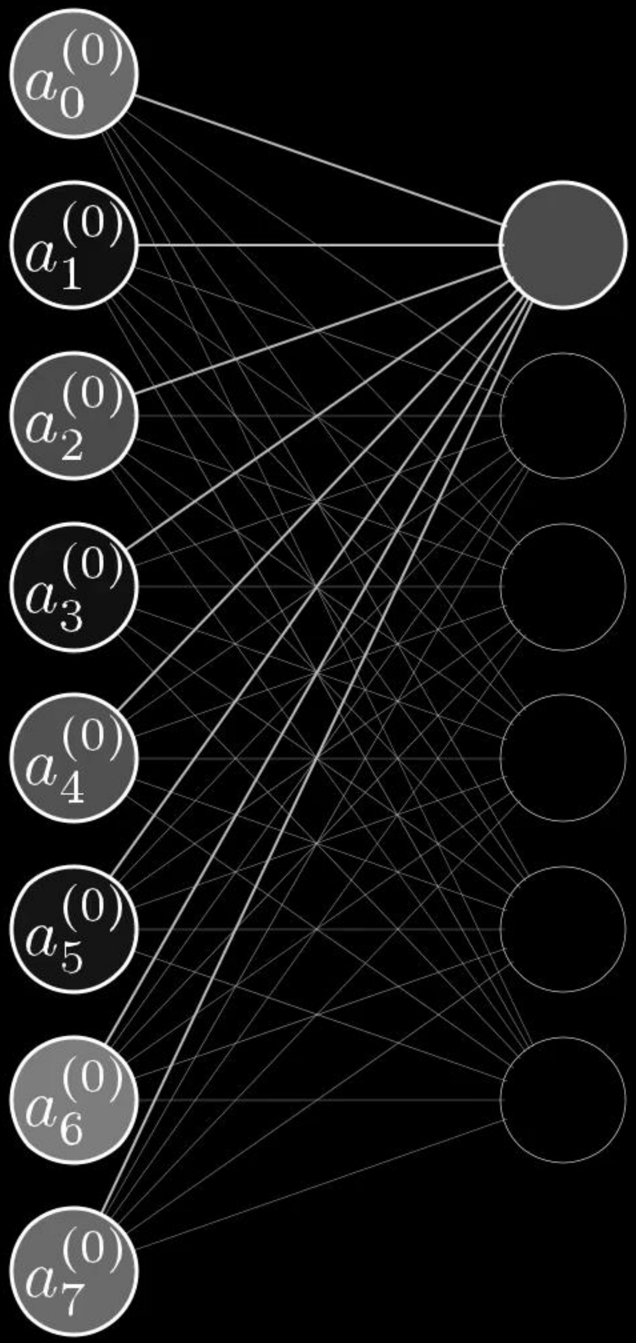


Sigmoid

$$a_0^{(1)} = \sigma\left(w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} + \cdots + w_{0,n}\, a_n^{(0)} \boxed{+ b_0} \right)$$

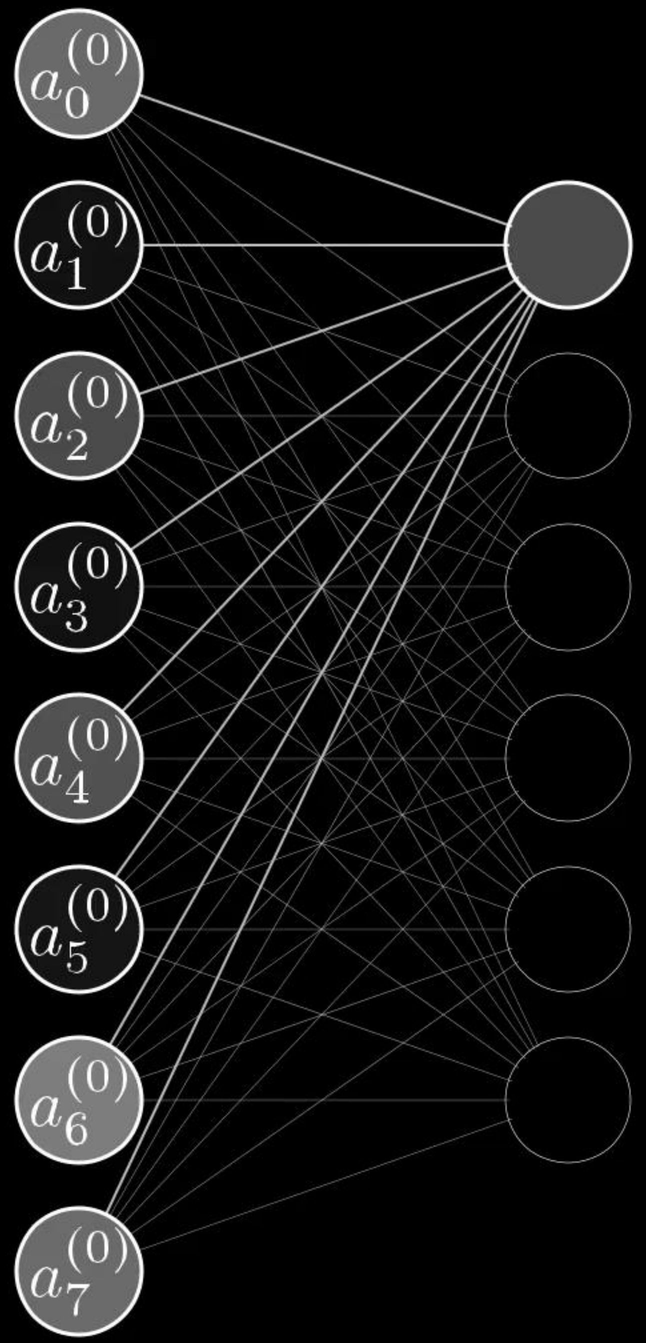
Bias

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix}$$



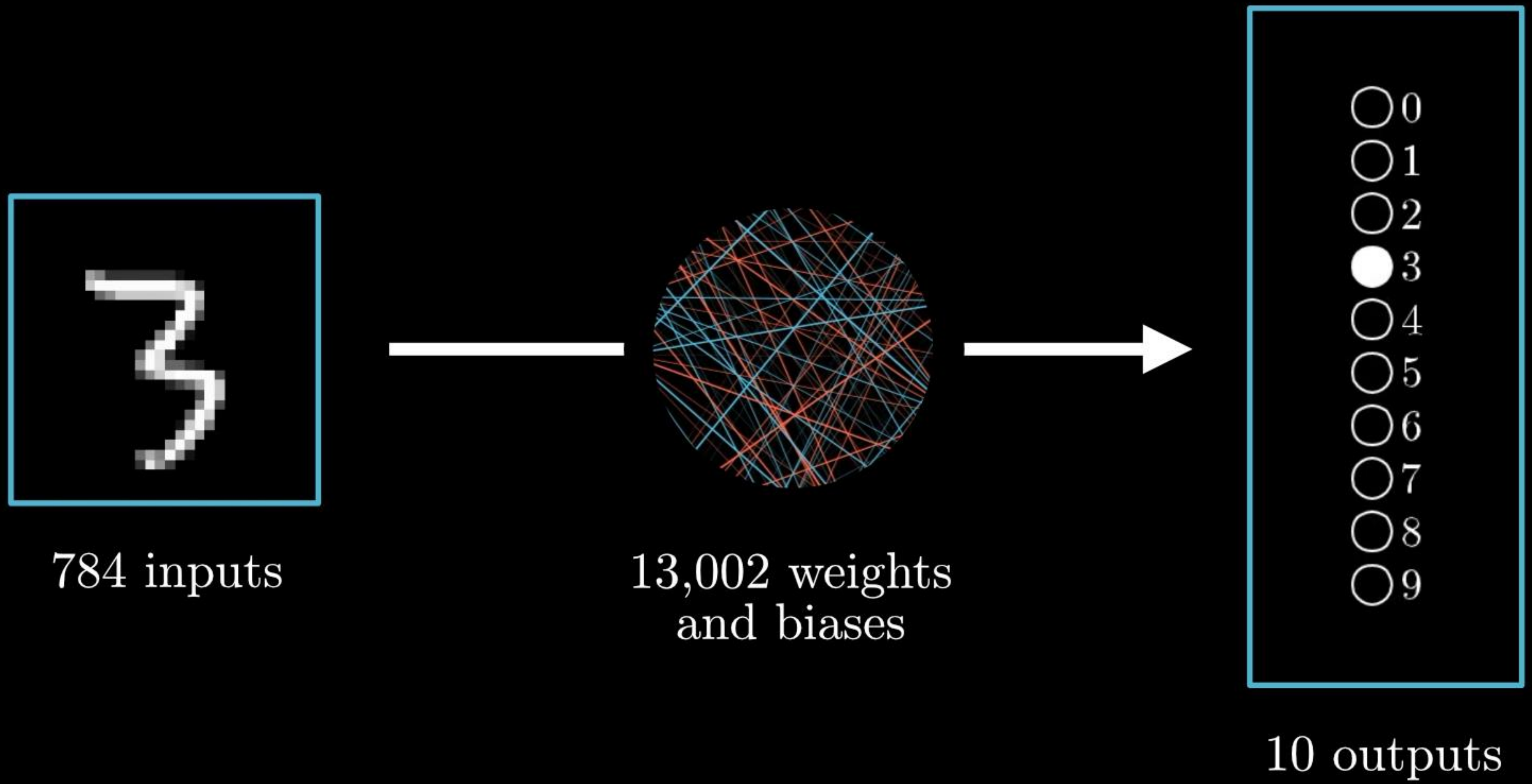
$$a_0^{(1)} = \overset{\text{Sigmoid}}{\sigma}\left(w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)} + \underset{\text{Bias}}{b_0} \right)$$

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$



$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right)$$

Neural network function



A NN with L hidden layers takes an input vector **x** and returns an output vector **y** though the forward equations chain

$$\mathbf{a}^{(1)} = \sigma\left(\mathbf{w}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{a}^{(2)} = \sigma\left(\mathbf{w}^{(1)}\mathbf{a}^{(1)} + \mathbf{b}^{(1)}\right)$$

$$\vdots$$

$$\mathbf{a}^{(L)} = \sigma\left(\mathbf{w}^{(L-1)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L-1)}\right)$$

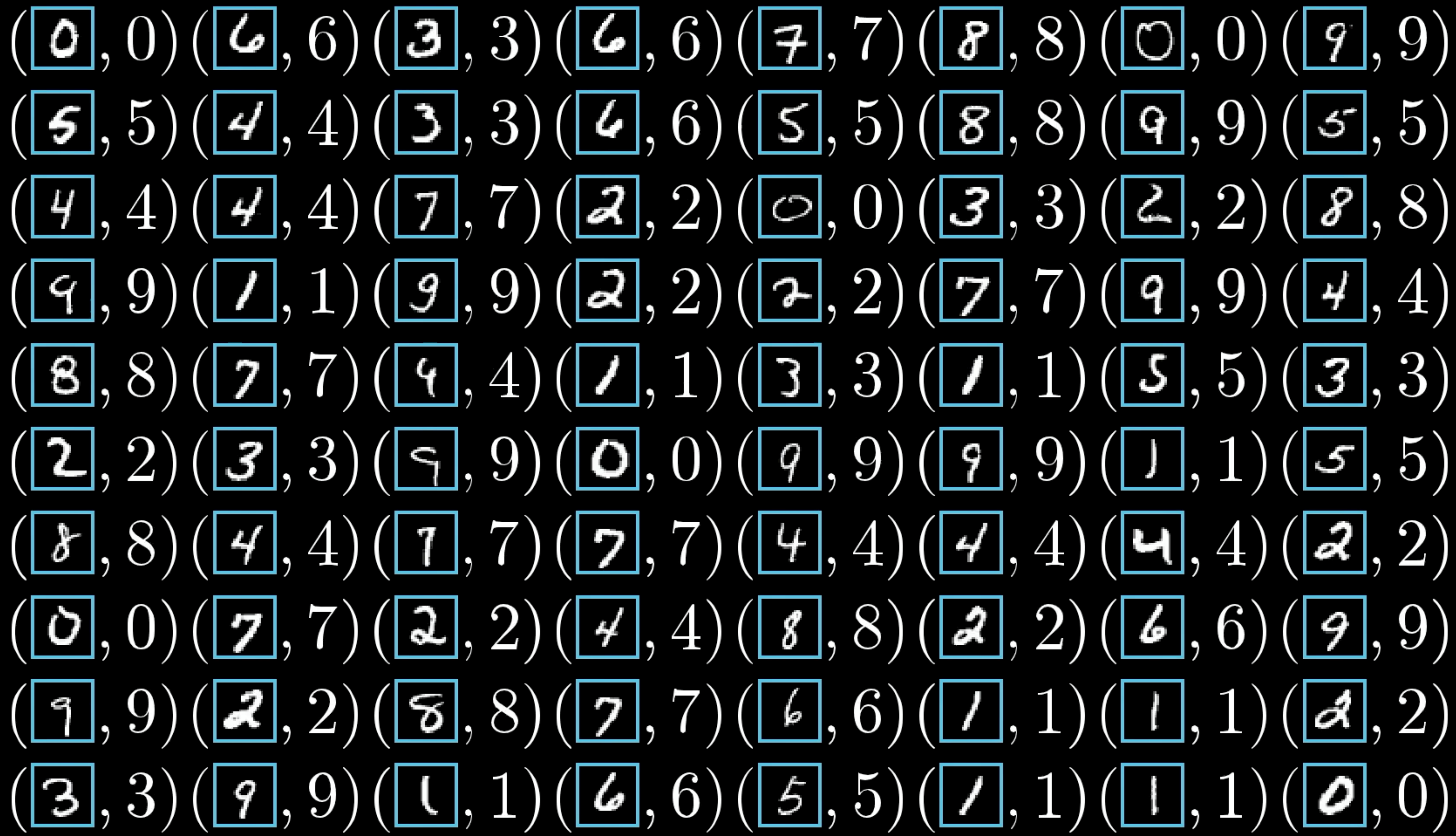
$$\mathbf{y} = \sigma\left(\mathbf{w}^{(L)}\mathbf{a}^{(L)} + \mathbf{b}^{(L)}\right)$$

The NN is a function of **x** with a parametric dependence on **w** and **b**

$$\mathbf{y} = f\left(\mathbf{x};\mathbf{w},\mathbf{b}\right)$$

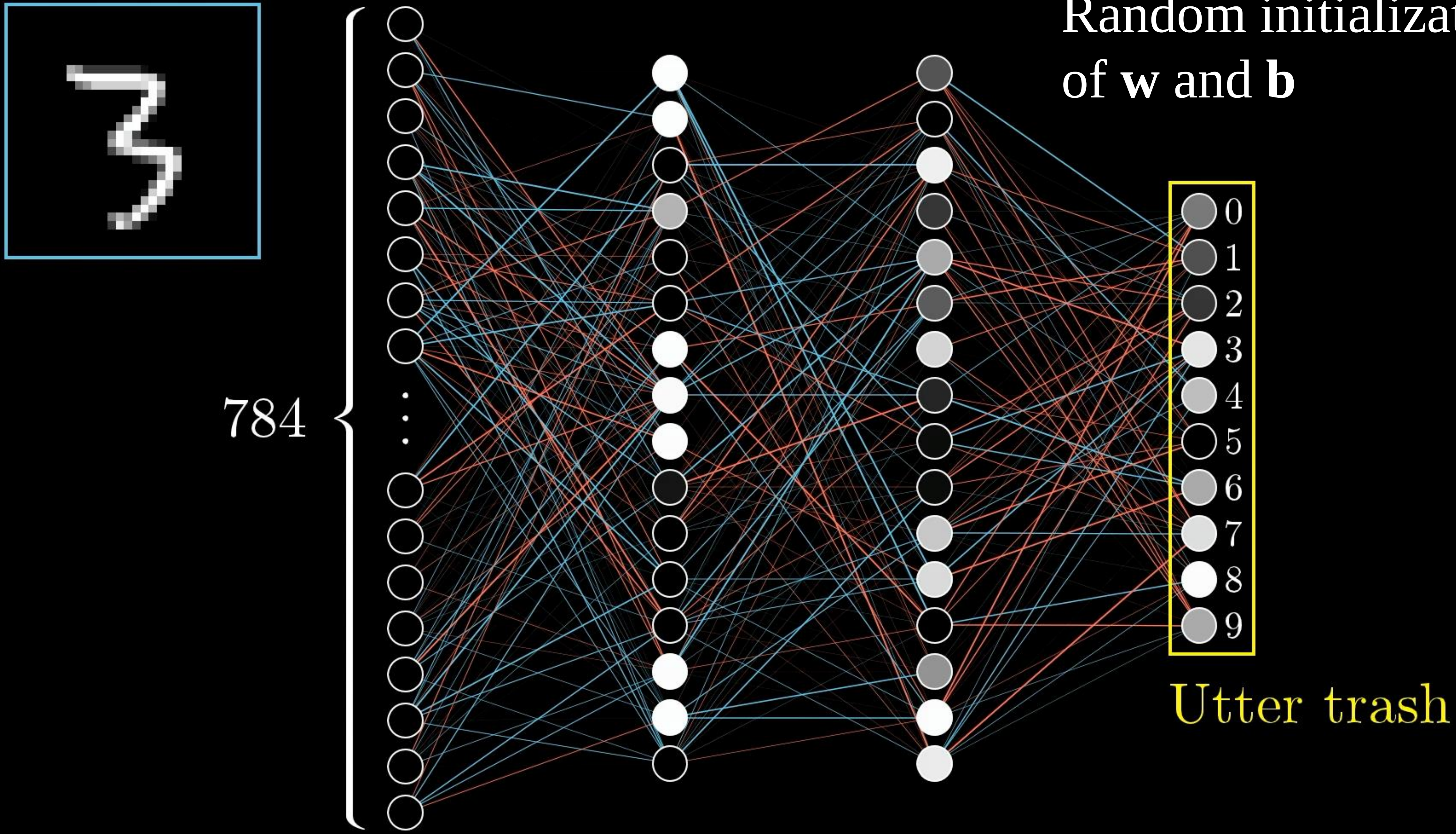
The cost of learning

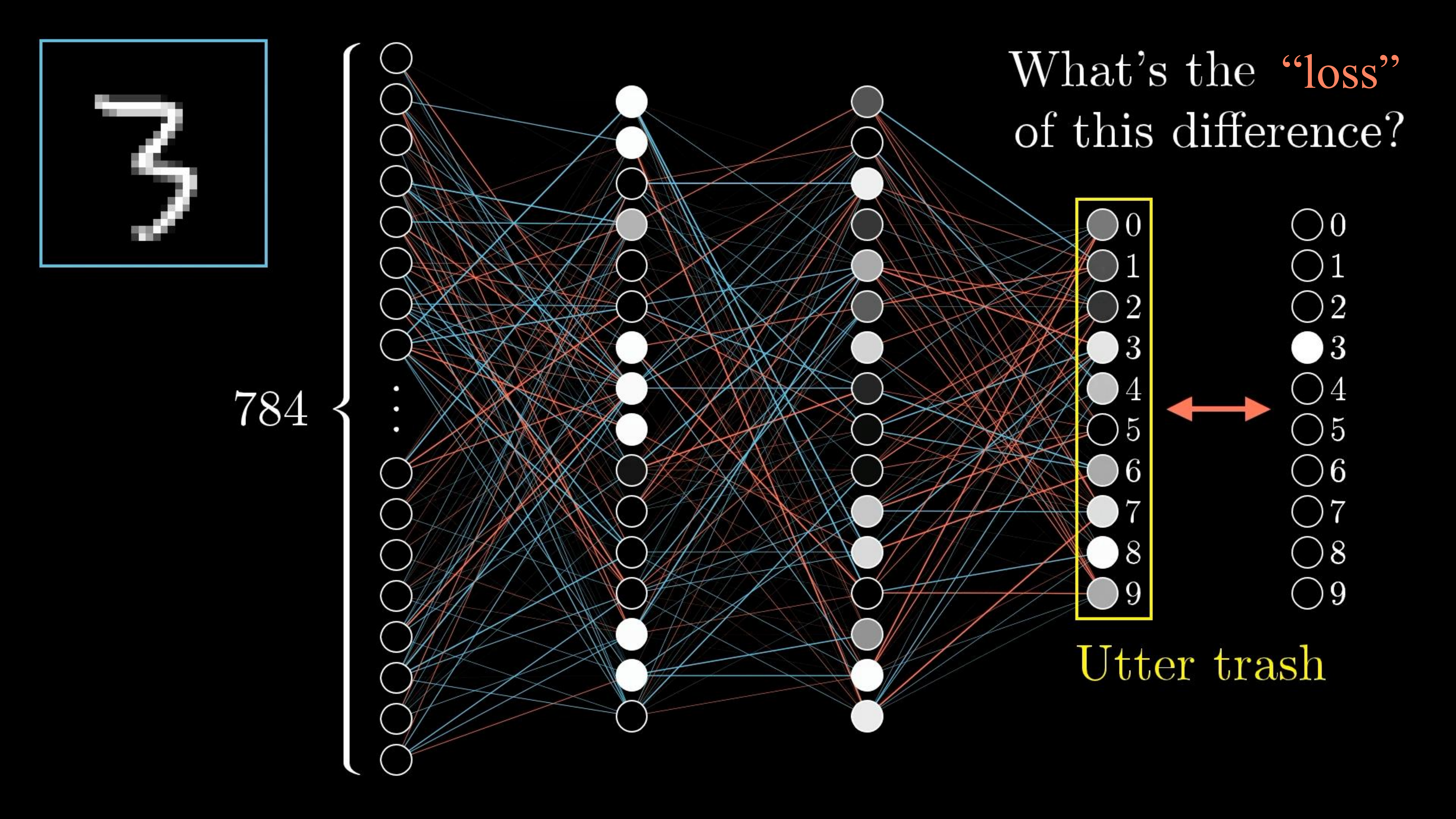
Adapted from:
www.3blue1brown.com/lessons/gradient-descent


(, 0) (, 6) (, 3) (, 6) (, 7) (, 8) (, 0) (, 9)
(, 5) (, 4) (, 3) (, 6) (, 5) (, 8) (, 9) (, 5)
(, 4) (, 4) (, 7) (, 2) (, 0) (, 3) (, 2) (, 8)
(, 9) (, 1) (, 9) (, 2) (, 2) (, 7) (, 9) (, 4)
(, 8) (, 7) (, 4) (, 1) (, 3) (, 1) (, 5) (, 3)
(, 2) (, 3) (, 9) (, 0) (, 9) (, 9) (, 1) (, 5)
(, 8) (, 4) (, 7) (, 7) (, 4) (, 4) (, 4) (, 2)
(, 0) (, 7) (, 2) (, 4) (, 8) (, 2) (, 6) (, 9)
(, 9) (, 2) (, 8) (, 7) (, 6) (, 1) (, 1) (, 2)
(, 3) (, 9) (, 1) (, 6) (, 5) (, 1) (, 1) (, 0)


Train on these
5
0
4
1
9
2
1
3
1
4
3
5
3
6
1
7
2
8
6
9
Test on these

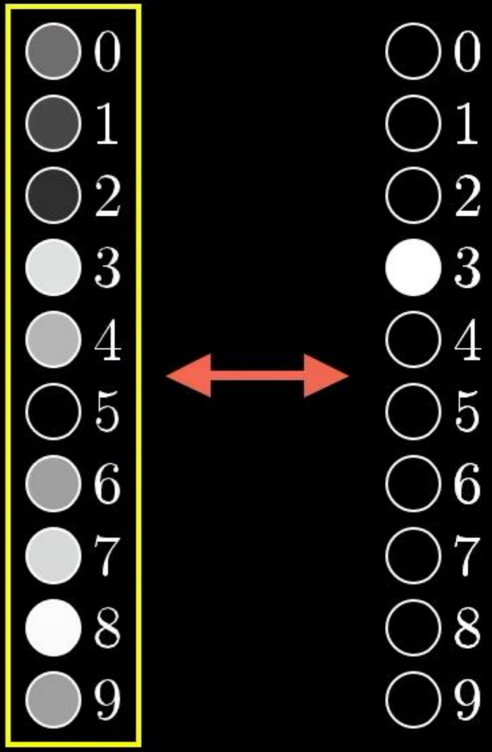
Random initialization of **w** and **b**





What's the "loss"
of this difference?
784
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
Utter trash

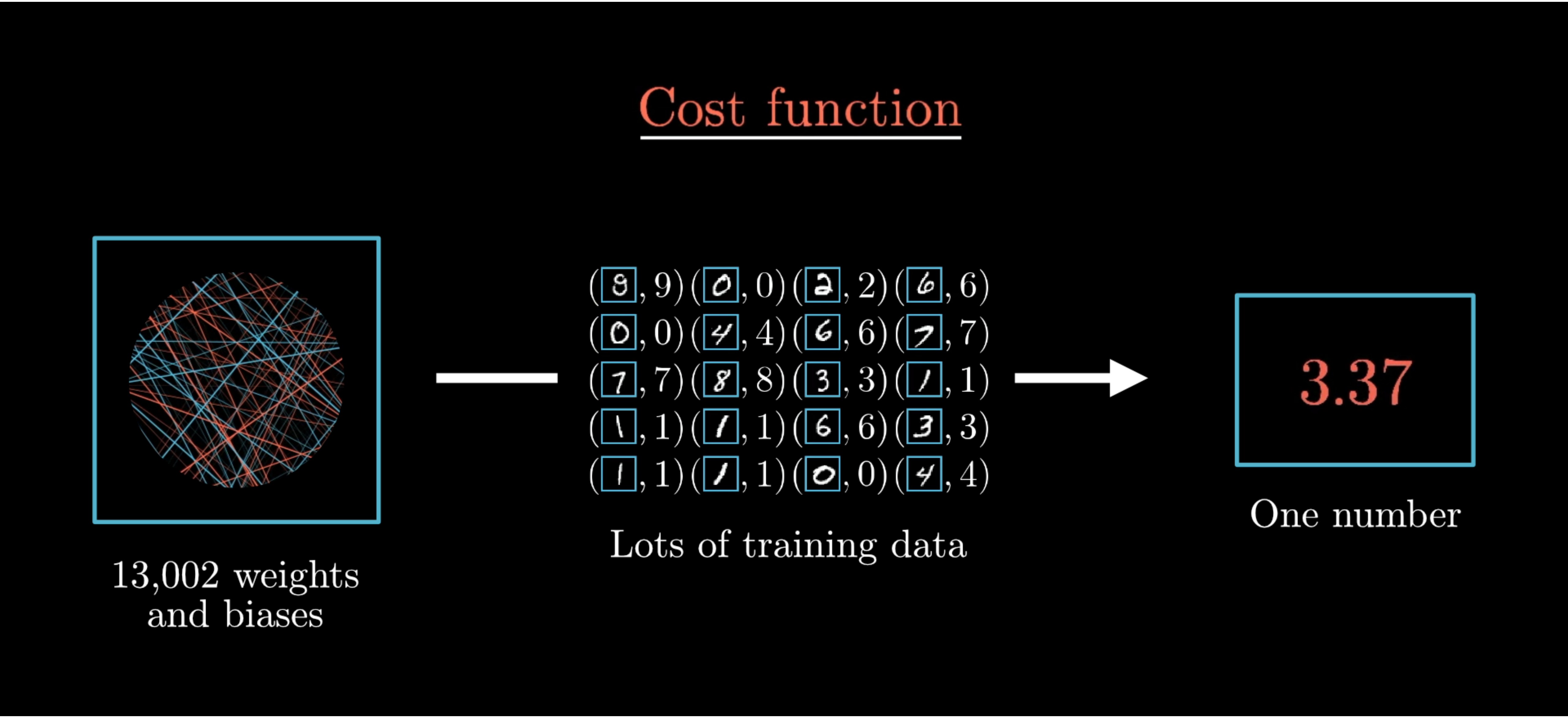
What's the "loss" of this difference?



Utter trash

Loss of 

$$\left\{\begin{array}{l}(0.43-0.00)^2+\\(0.28-0.00)^2+\\(0.19-0.00)^2+\\(0.88-1.00)^2+\\(0.72-0.00)^2+\\(0.01-0.00)^2+\\(0.64-0.00)^2+\\(0.86-0.00)^2+\\(0.99-0.00)^2+\\(0.63-0.00)^2\end{array}\right.$$


Cost function
13,002 weights
and biases
(, 9) (, 0) (, 2) (, 6)
(, 0) (, 4) (, 6) (, 7)
(, 7) (, 8) (, 3) (, 1)
(, 1) (, 1) (, 6) (, 3)
(, 1) (, 1) (, 0) (, 4)
Lots of training data
3.37
One number

For each training data *k*, with true value $\hat{\mathbf{y}}_k$
get the ***loss function*** over the output neurons

$$\mathcal{L}_k(\mathbf{w},\mathbf{b}) = \frac{1}{N_{out}} \sum_{i=1}^{N_{out}} \left(NN\left(x_{k,i};\mathbf{w},\mathbf{b}\right) - \hat{y}_{k,i} \right)^2$$

Considering all training data, get the ***cost function***

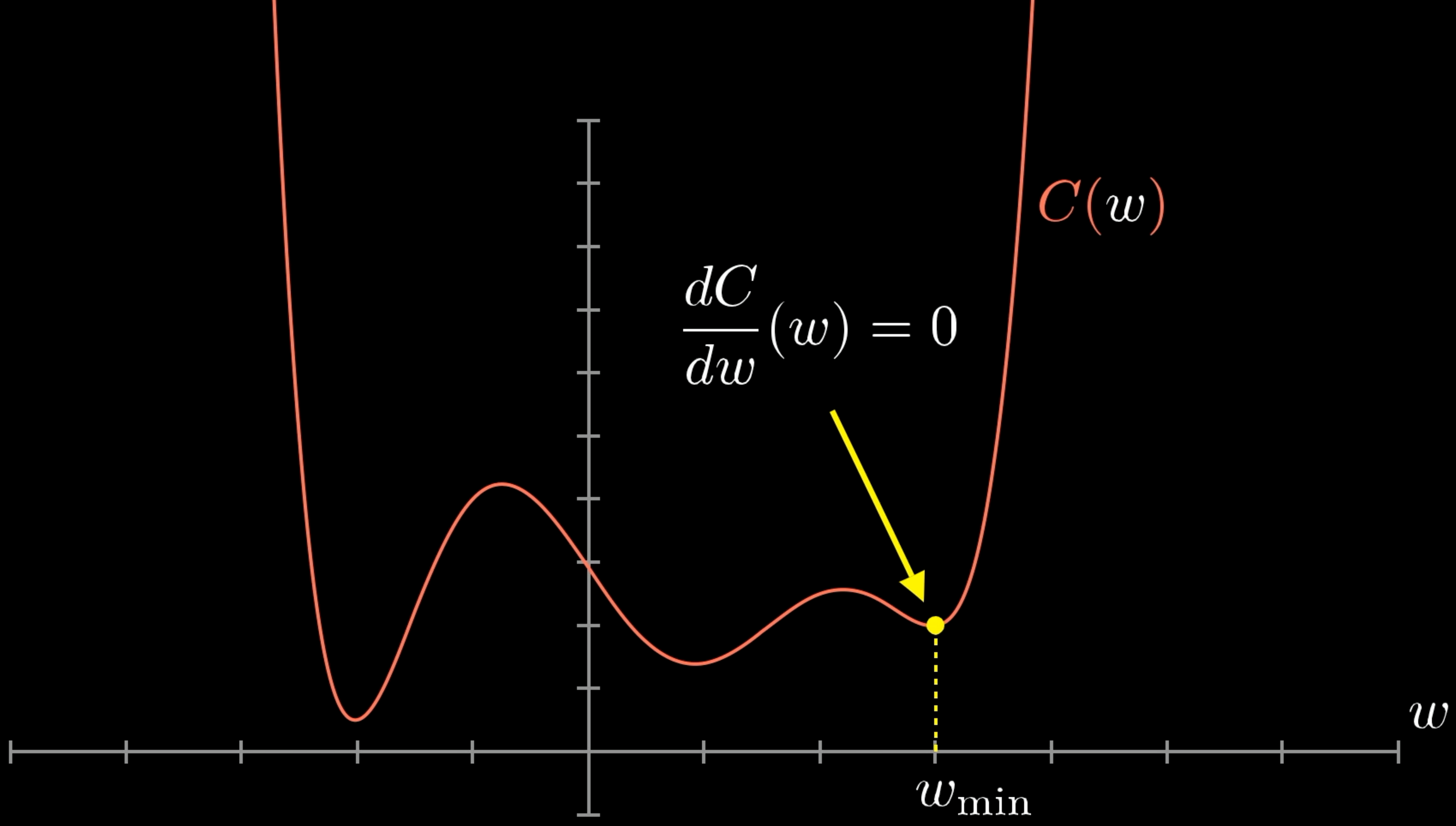
$$C(\mathbf{w},\mathbf{b}) = \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} \mathcal{L}_k(\mathbf{w},\mathbf{b})$$

The optimized NN is the one with **w** and **b** that minimizes $C(\mathbf{w}, \mathbf{b})$

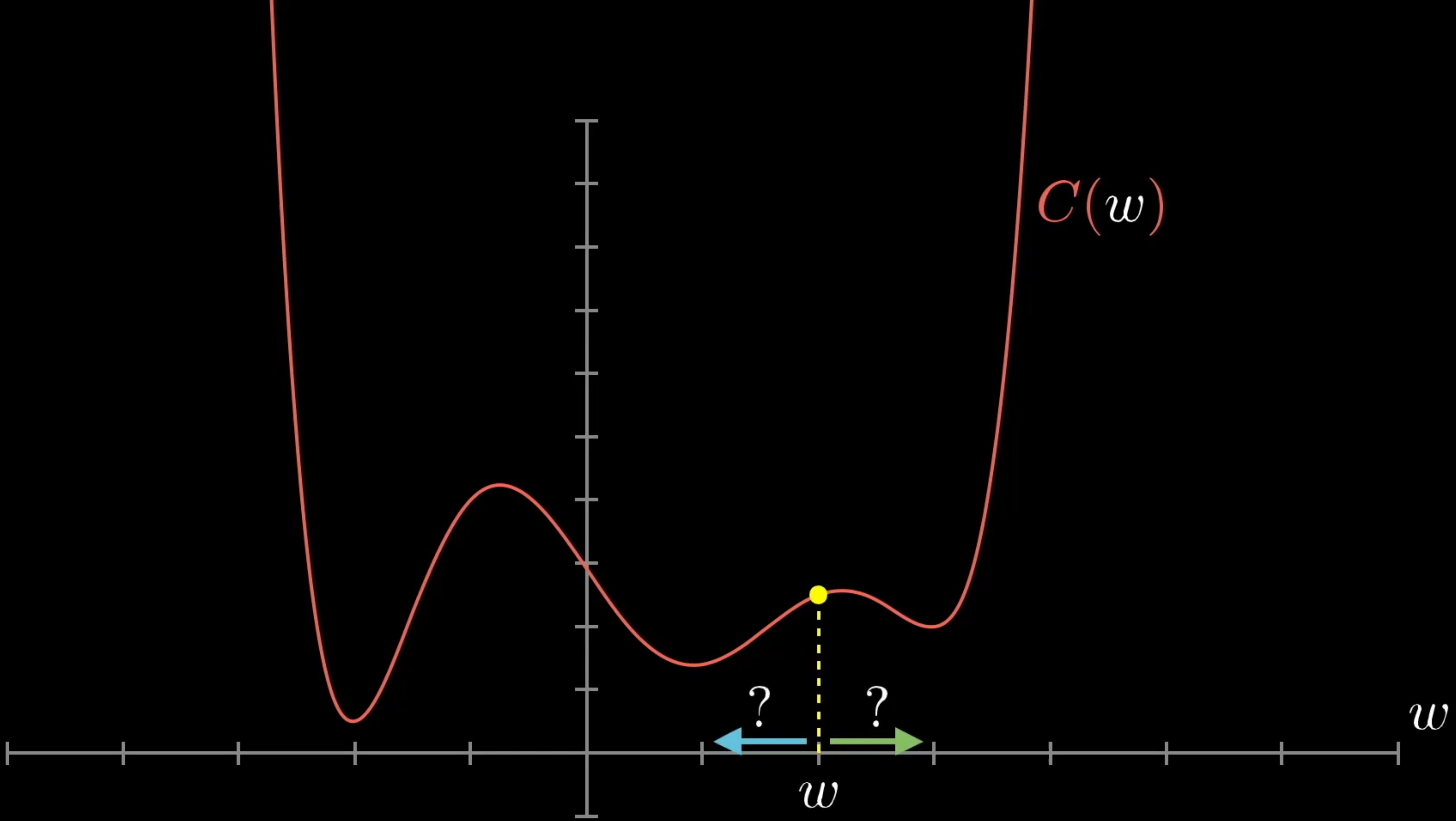
$$\min_{\mathbf{w},\mathbf{b}} C(\mathbf{w},\mathbf{b}) \Rightarrow \nabla_{\mathbf{w},\mathbf{b}} C(\mathbf{w},\mathbf{b}) = 0$$

How to Train Your Neural Network

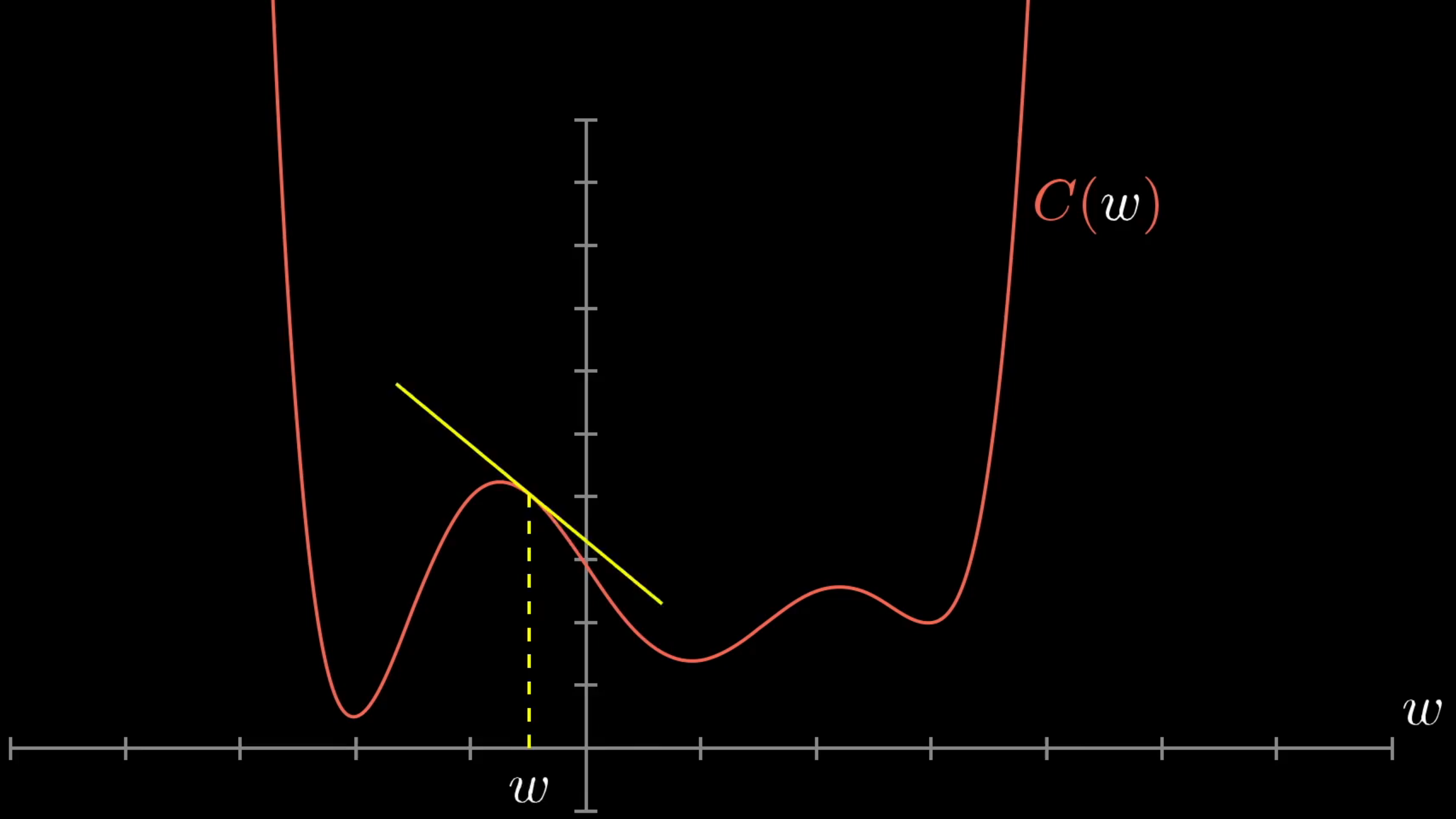
Adapted from:
www.3blue1brown.com/lessons/gradient-descent



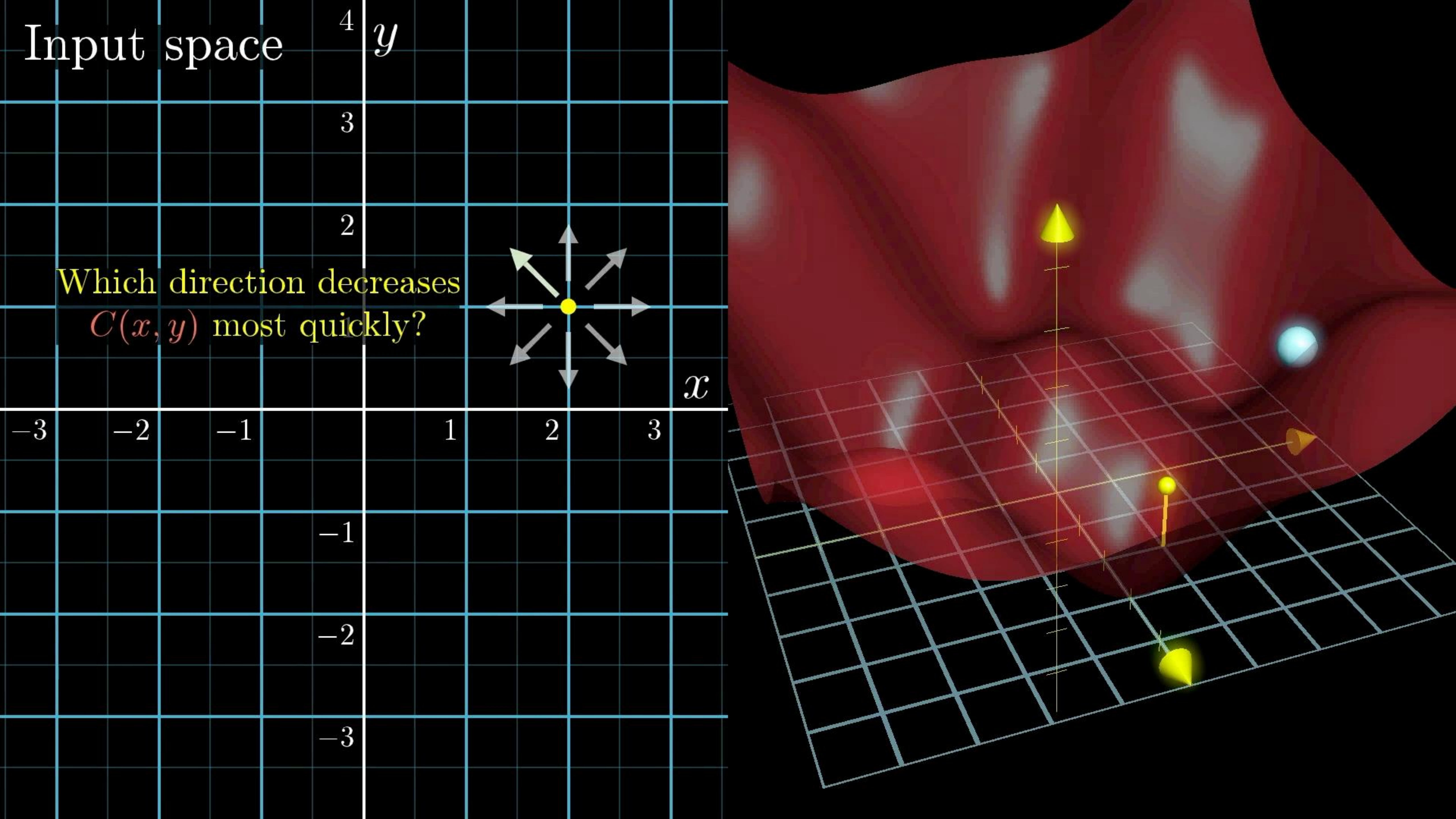
$C(w)$
$\frac{dC}{dw}(w) = 0$
w
$w_{\min}$



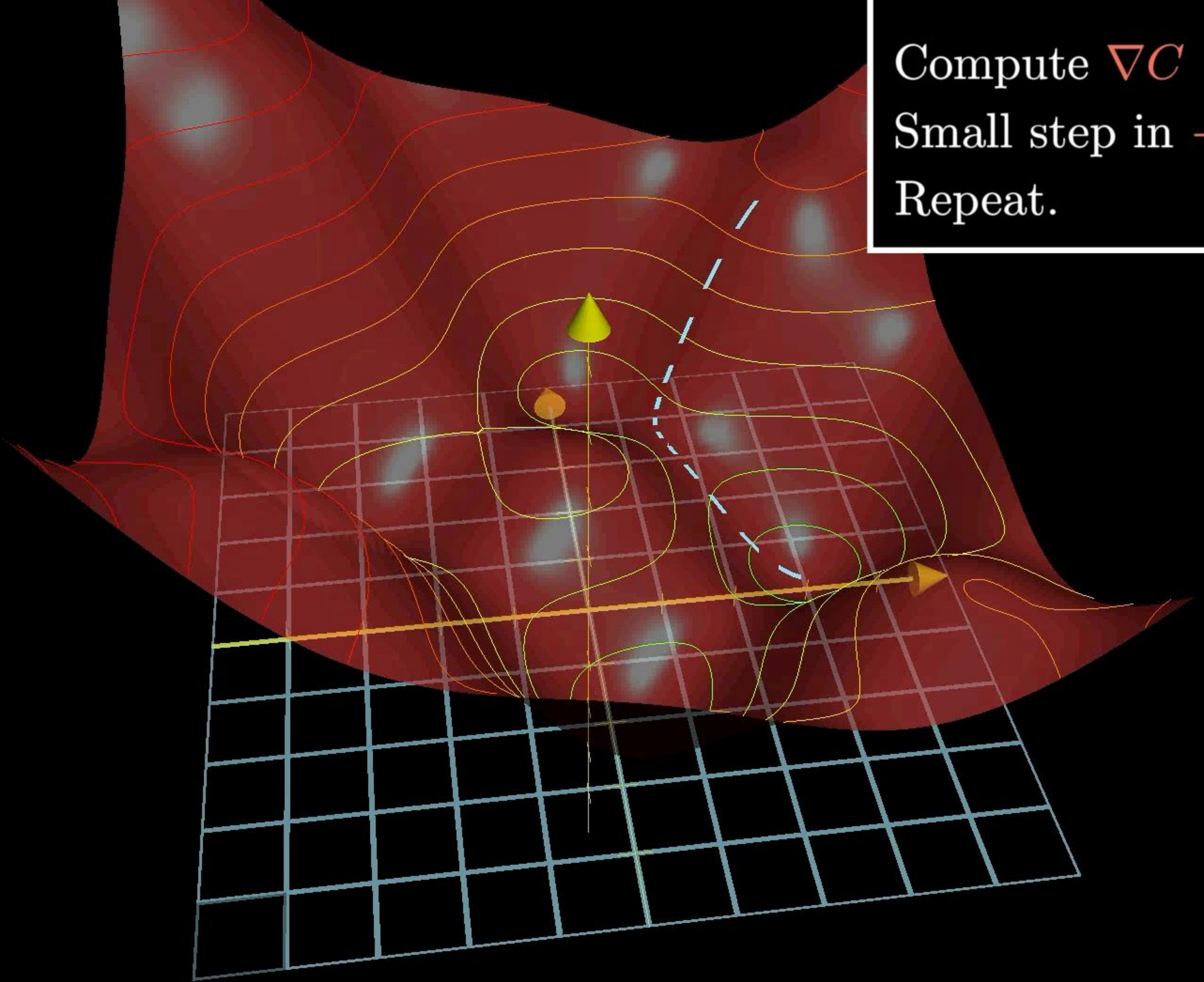
$C(w)$
?
?
w
w



$C(w)$
w
w



Input space
Which direction decreases
$C(x, y)$ most quickly?
x
y
4
3
2
1
−1
−2
−3



Compute ∇C

Small step in $-\nabla C$ direction

Repeat.

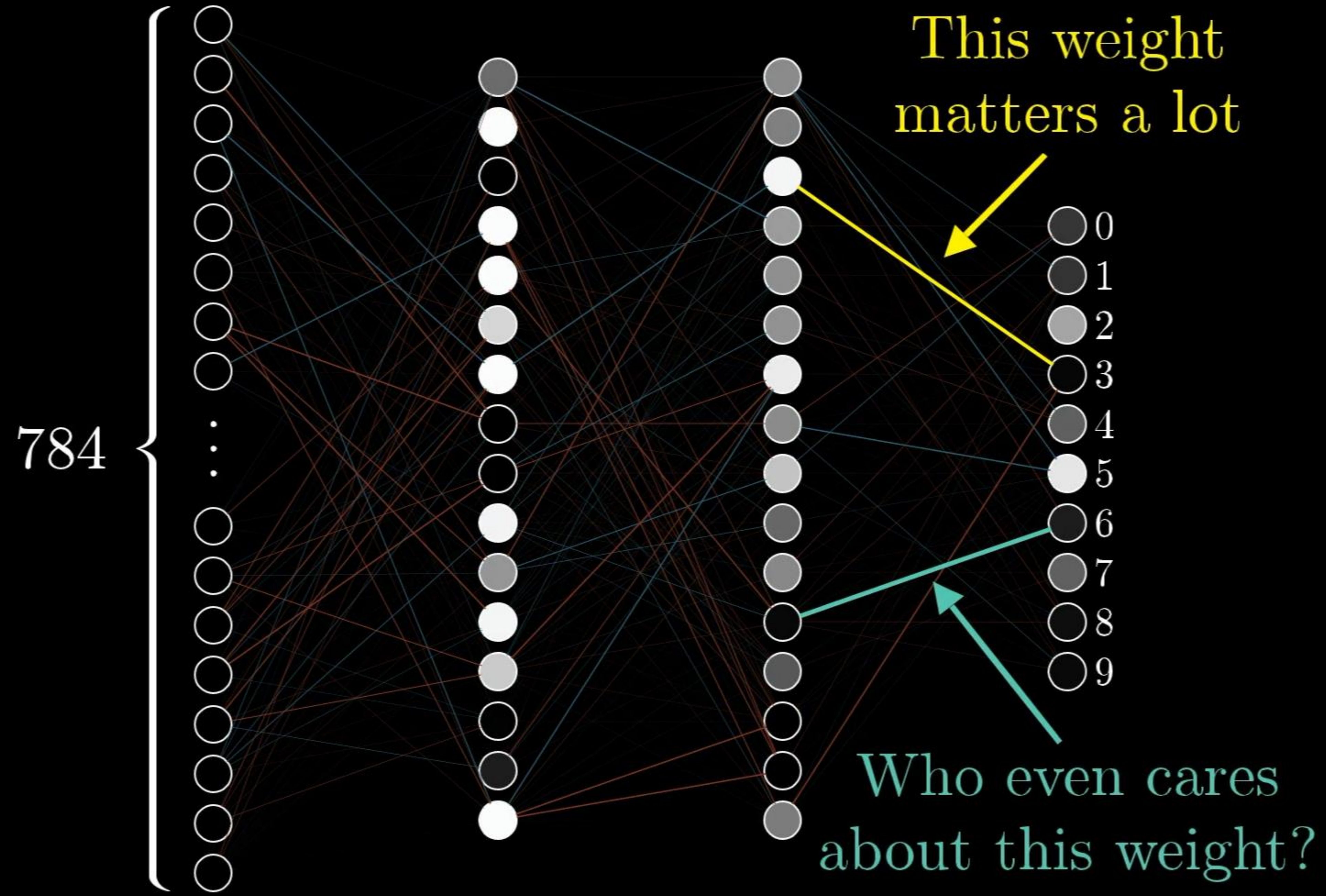
13,002 weights and biases

$$\vec{\mathbf{W}} = \begin{bmatrix} 2.25 \\ -1.57 \\ 1.98 \\ \vdots \\ -1.16 \\ 3.82 \\ 1.21 \end{bmatrix}$$

How to nudge all weights and biases

$$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.18 \\ 0.45 \\ -0.51 \\ \vdots \\ 0.40 \\ -0.32 \\ 0.82 \end{bmatrix}$$

$$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.78 \\ -0.37 \\ 0.16 \end{bmatrix} \begin{array}{l} w_0 \text{ should increase somewhat} \\ w_1 \text{ should increase a little} \\ w_2 \text{ should decrease a lot} \\ \\ w_{13,000} \text{ should increase a lot} \\ w_{13,001} \text{ should decrease somewhat} \\ w_{13,002} \text{ should increase a little} \end{array}$$



This weight matters a lot
784
0
1
2
3
4
5
6
7
8
9
Who even cares about this weight?

The minimization is done with **gradient descent**:

$$\mathbf{w}_{m+1} = \mathbf{w}_m - \gamma_m \frac{\partial C(\mathbf{w}_m, \mathbf{b}_m)}{\partial \mathbf{w}_m} \qquad (\gamma_m - \text{learning rate})$$

$$\mathbf{b}_{m+1} = \mathbf{b}_m - \gamma_m \frac{\partial C(\mathbf{w}_m, \mathbf{b}_m)}{\partial \mathbf{b}_m}$$

Usually, the gradient is computed for a sub-set of **w** and **b** components chosen at random. It is called **stochastic gradient descent**

en.wikipedia.org/wiki/Gradient_descent

Now that we have optimal **w** and **b**, we can use the neural network to identify images that were not in the training set.

This is the basics. But there is so much more…

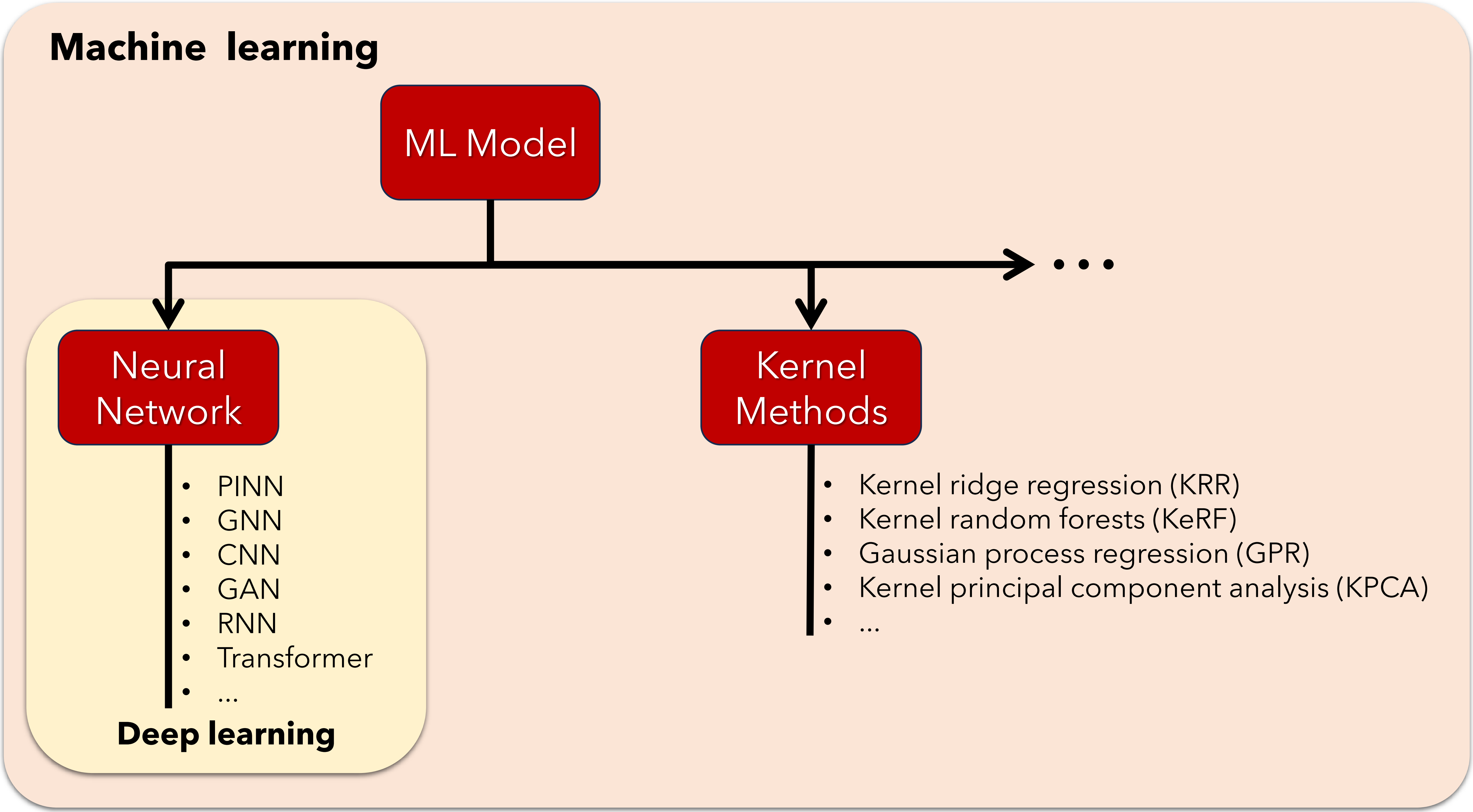
- **Physics-informed neural networks (PINN)**
  Integrates partial differential equations expressing physical laws into the NN cost function
  youtu.be/G_hIppUWcsc

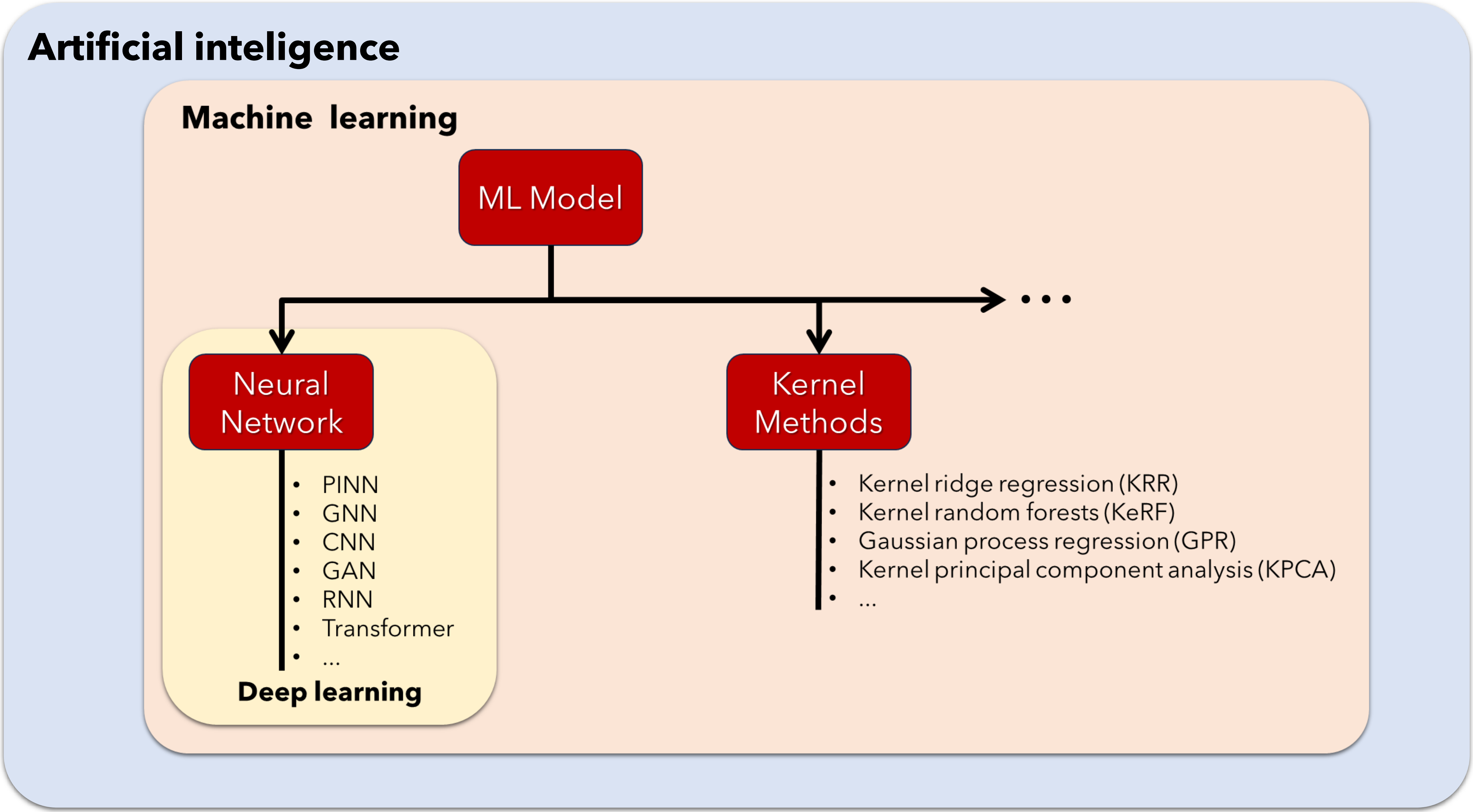
- **Graph neural networks (GNN)**
  NN for processing data that can be represented as vertices connected by edges
  distill.pub/2021/gnn-intro/

- **Convolutional neural networks (CNN)**
  NN with convolutional layers that capture local patterns and global features in the input data
  tinyurl.com/convnnet

- **Generative adversarial networks (GAN)**
  Two NN – a generator and a discriminator – compete to create realistic-looking outputs
  tinyurl.com/ganetintro

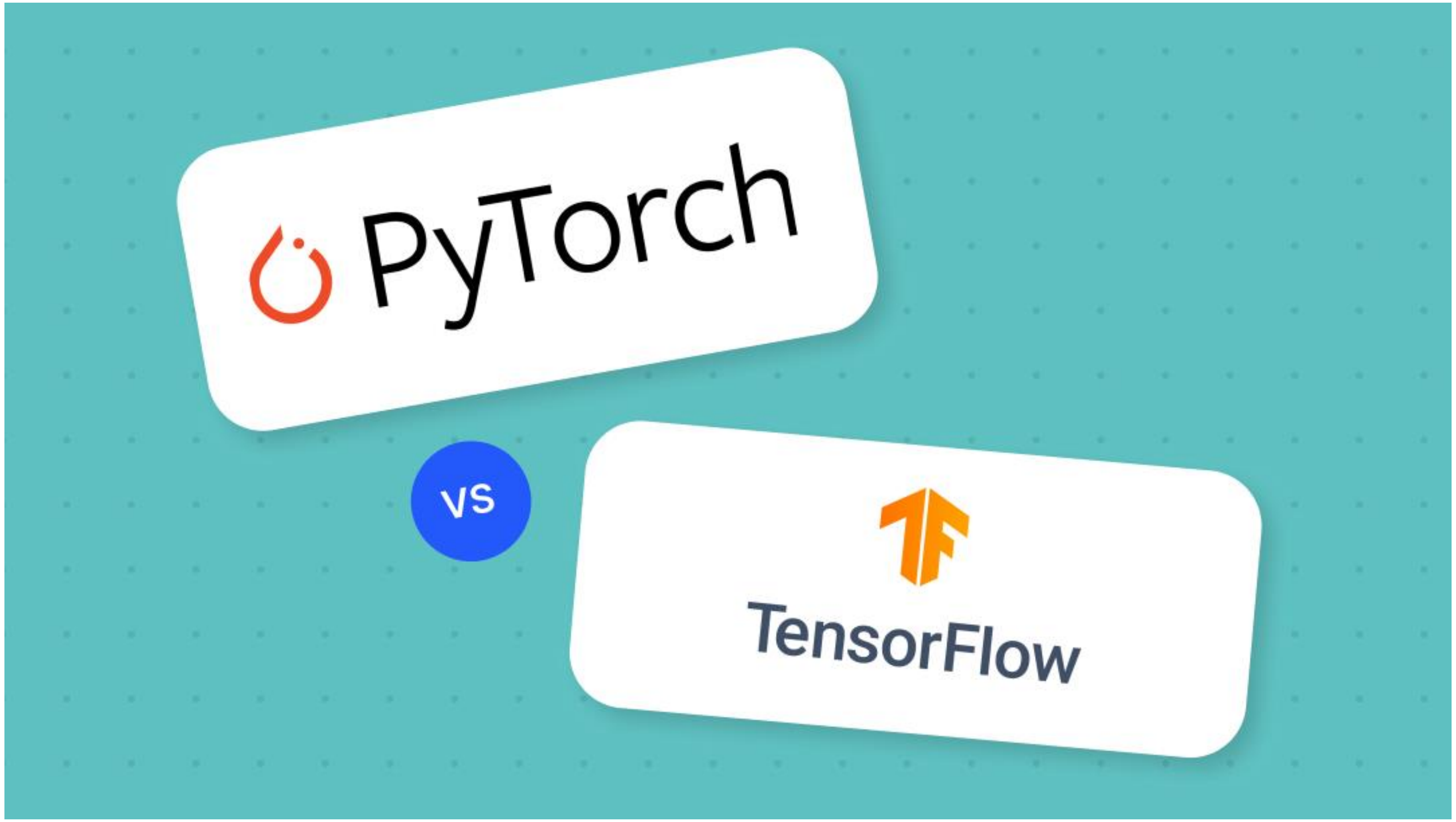
- **Transformer neural networks**
  NN architecture for encoding words, word position, and word's contextual relation with others in the sentence (self-attention).
  https://youtu.be/zxQyTK8quyY


Machine learning
ML Model
...
Neural Network
PINN
GNN
CNN
GAN
RNN
Transformer
...
Deep learning
Kernel Methods
Kernel ridge regression (KRR)
Kernel random forests (KeRF)
Gaussian process regression (GPR)
Kernel principal component analysis (KPCA)
...

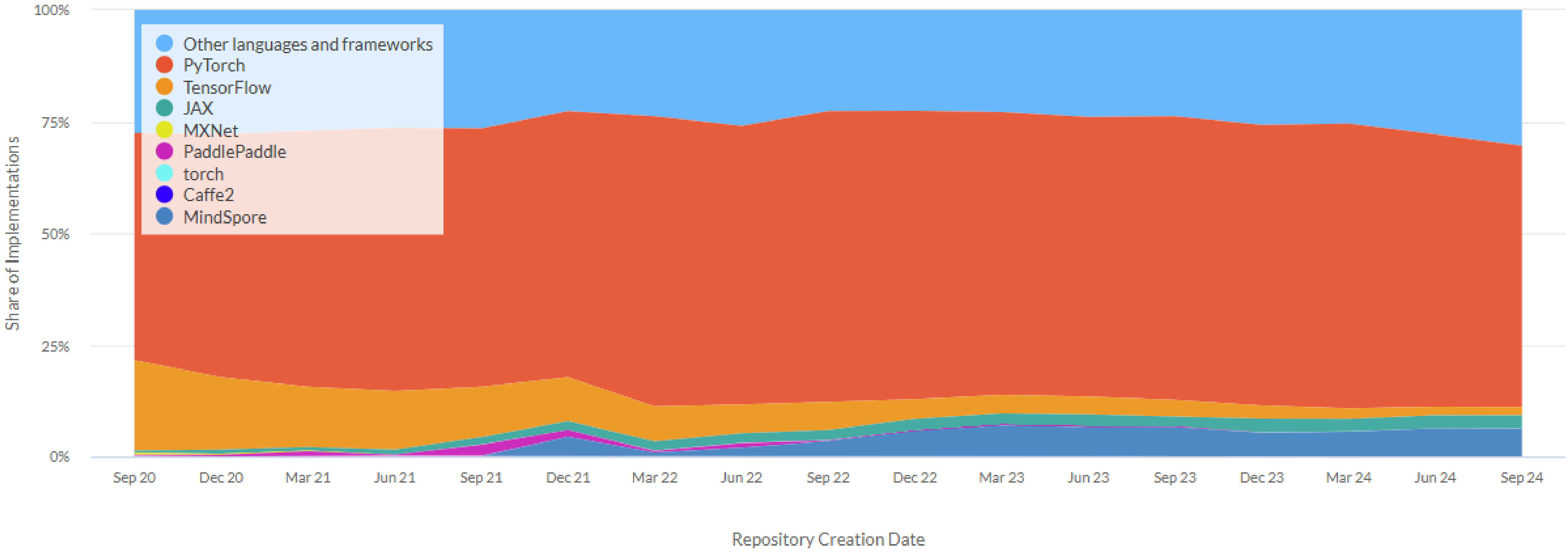

Artificial inteligence
Machine learning
ML Model
...
Neural Network
• PINN
• GNN
• CNN
• GAN
• RNN
• Transformer
• ...
Deep learning
Kernel Methods
• Kernel ridge regression (KRR)
• Kernel random forests (KeRF)
• Gaussian process regression (GPR)
• Kernel principal component analysis (KPCA)
• ...

Machine learning in practice

Machine learning development network



www.v7labs.com/blog/pytorch-vs-tensorflow



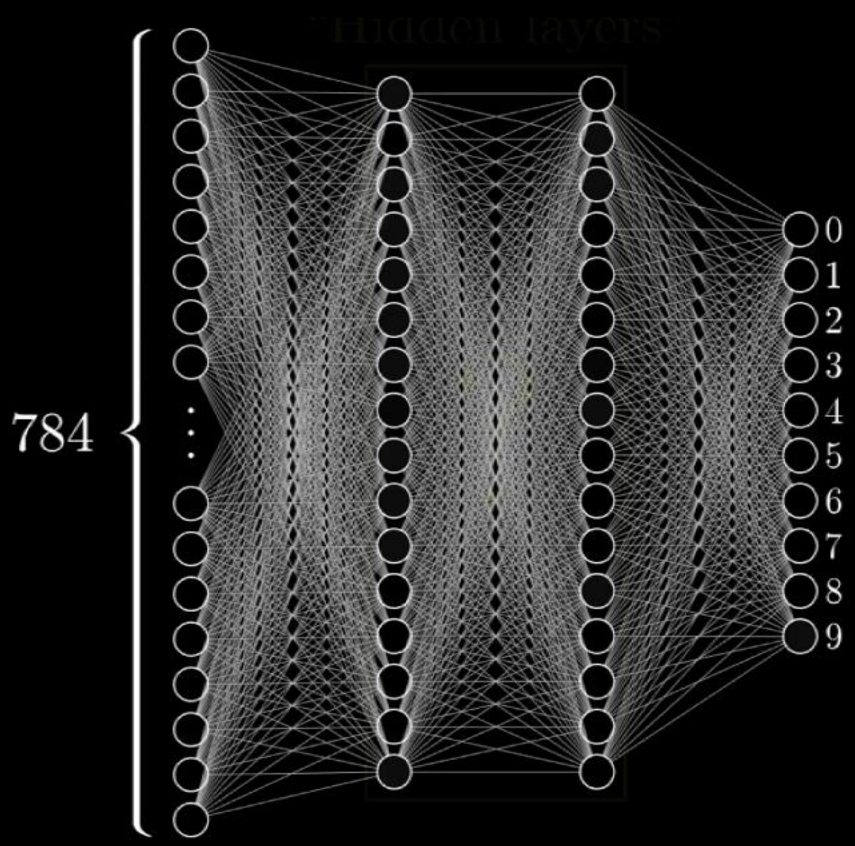
www.v7labs.com/blog/pytorch-vs-tensorflow

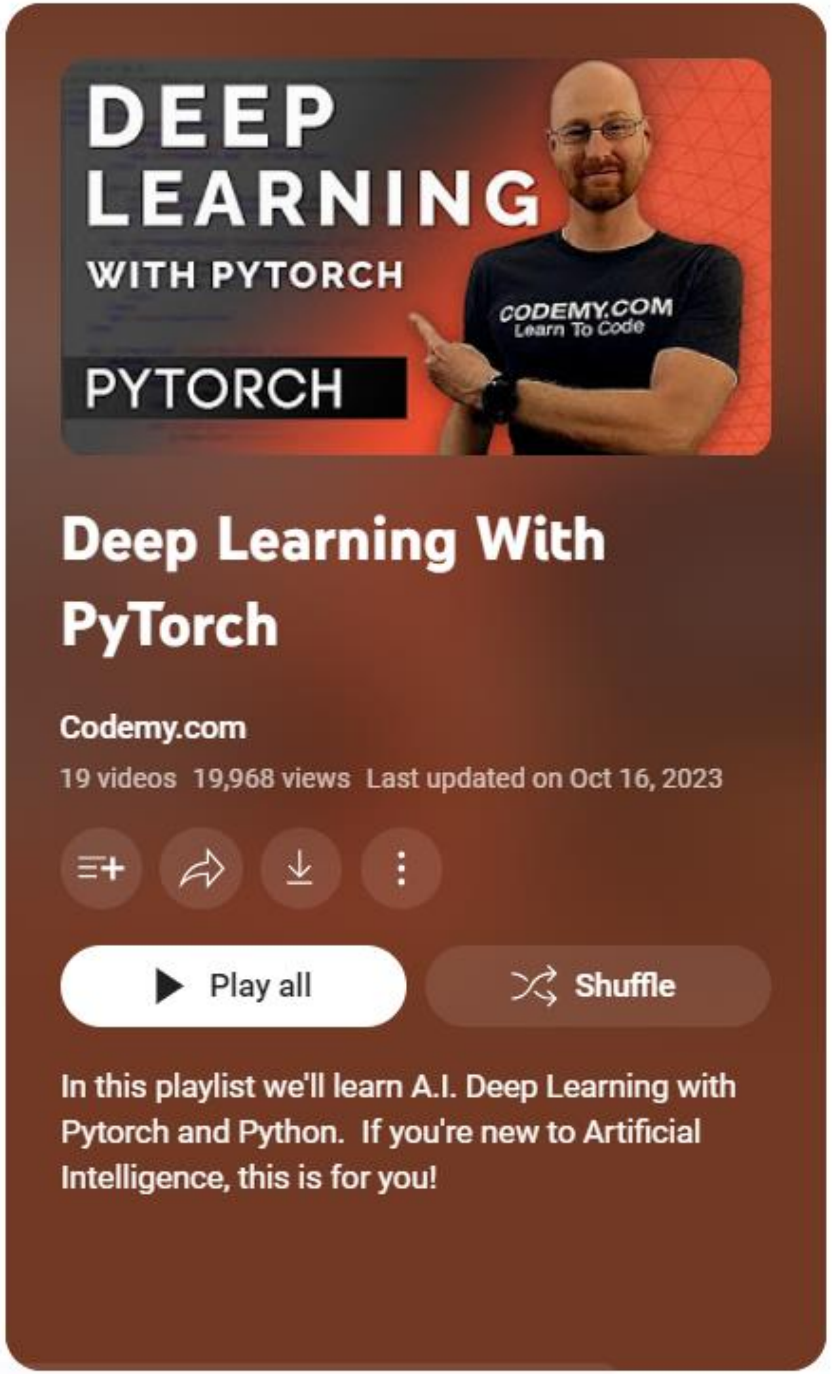
```python
import torch
import torch.nn as nn

class SimpleNeuralNetwork(nn.Module):
    def __init__(self):
        super(SimpleNeuralNetwork, self).__init__()
        self.layer1 = nn.Linear(784, 16)
        self.layer2 = nn.Linear(16, 16)
        self.output_layer = nn.Linear(16, 10)

    def forward(self, x):
        x = torch.sigmoid(self.layer1(x))
        x = torch.sigmoid(self.layer2(x))
        x = self.output_layer(x)
        return x

 # Instantiate the model
model = SimpleNeuralNetwork()
```





tinyurl.com/pytorchlearn

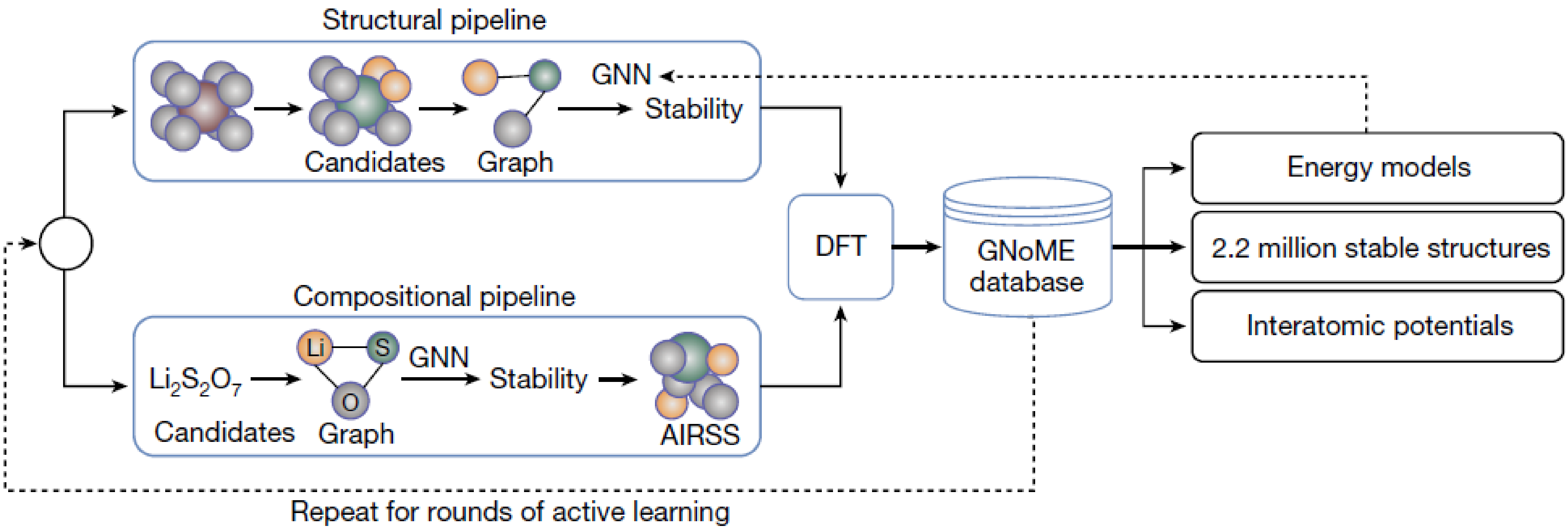
ML for Modeling Materials

AI for theoretical chemistry has been used to

- Search the chemical space of compounds
- Perform dimensionality reduction, clustering, and pattern recognition
- Improve or accelerate quantum chemical methods
- Predict properties as a surrogate approach

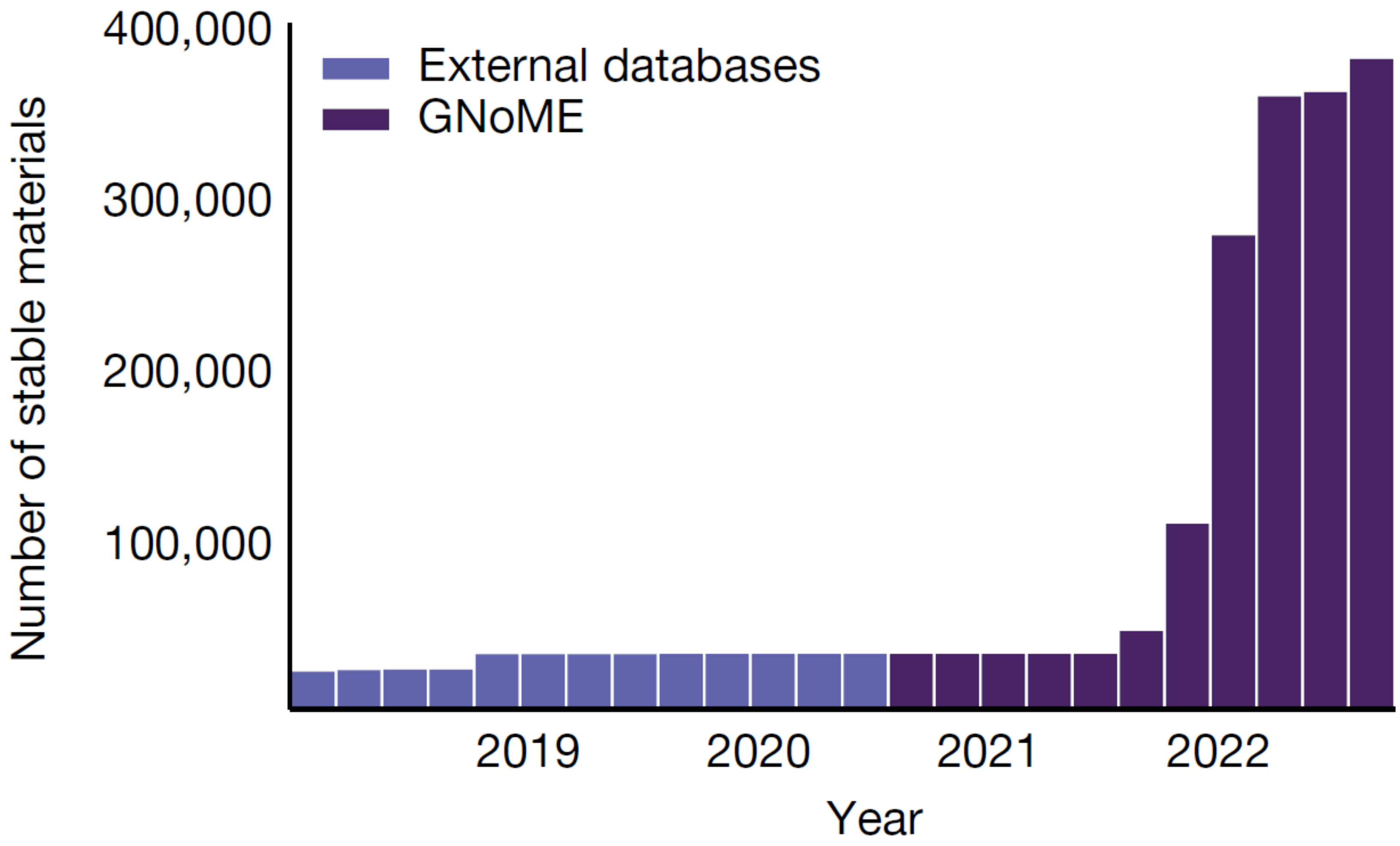
Search the chemical space of compounds

GNN-based discovery of new materials



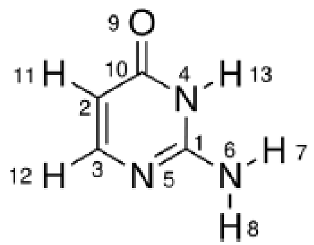
Merchant *et al.* *Nature* **2023**. [10.1038/s41586-023-06735-9](https://doi.org/10.1038/s41586-023-06735-9)

GNN-based discovery added 381,000 new stable materials to the database

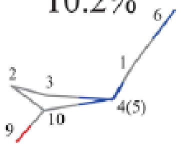
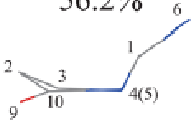
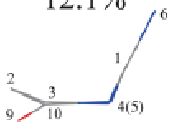
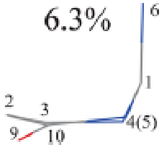
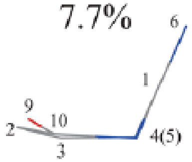


Merchant *et al. Nature* **2023**. [10.1038/s41586-023-06735-9](https://doi.org/10.1038/s41586-023-06735-9)

Perform dimensionality reduction, clustering, and pattern recognition



Cytosine

Channel	Important motion
Cluster A1B1 10.2% 	C1-puckering C10-puckering C=O out-of-plane motion[–][a]
Cluster A1B2C1 56.2% 	C1-puckering NH_2 out-of-plane motion[–][a]
Cluster A1B2C2 12.1% 	C1-puckering
Cluster A1B2C3 6.3% 	C1-puckering NH_2 out-of-plane motion[+][a] C1–N6 bond stretching
Cluster A1B3 7.7% 	C1-puckering C=O out-of-plane motion[+][a]
Cluster A2 7.5% 	C=O stretching motion

Hierarchical protocol for the automatic analysis of the ring deformation in surface hopping

Based on
- dimensionality reduction (PCA)
- clustering (DBSCAN + agglomerative clustering)

Zhu *et al.* *PCCP* **2022**, *24*, 24362

Improve or accelerate quantum chemical methods

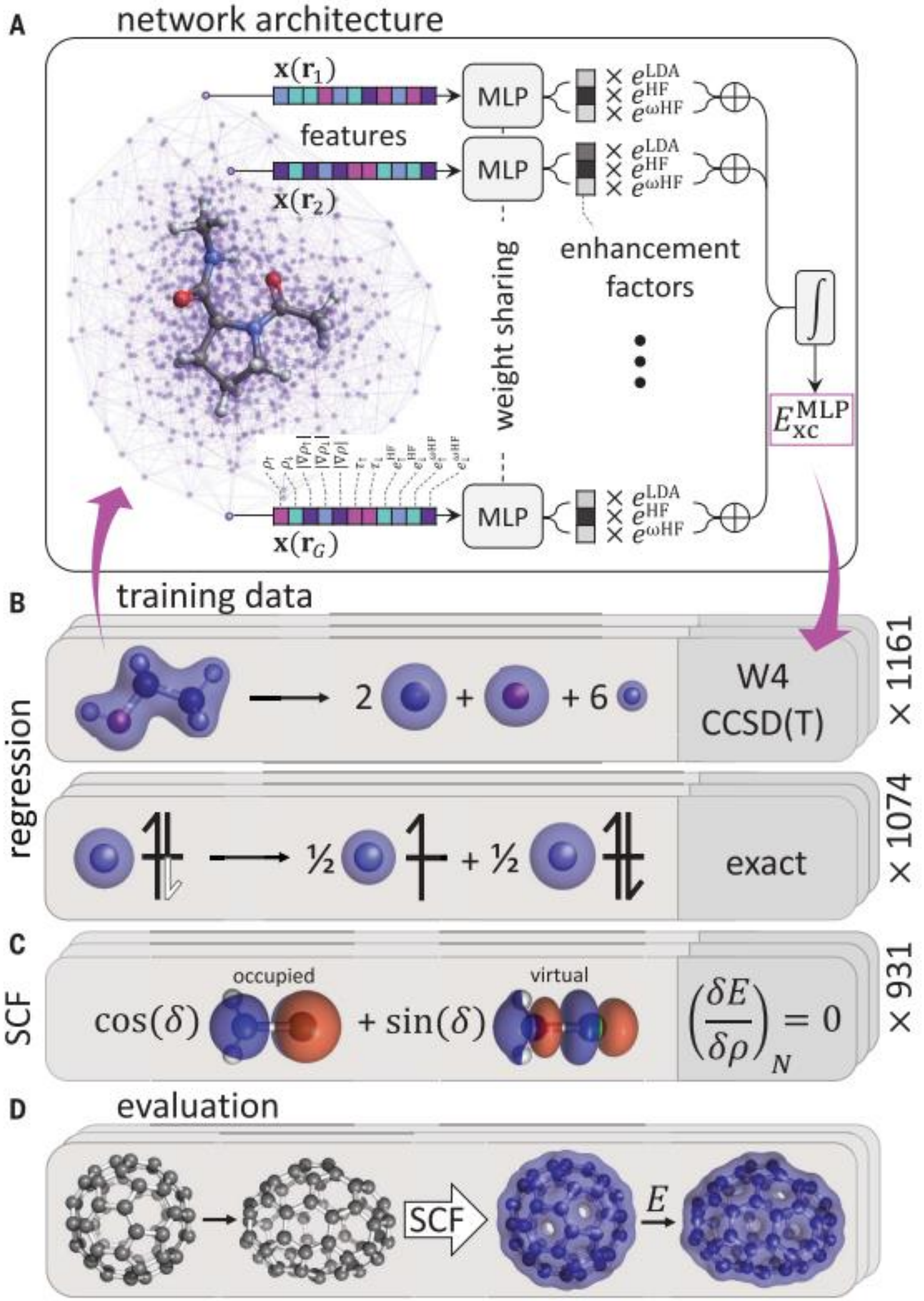
Density functional from an NN

Input features:

- charge density r,
- norm of charge density
- electron kinetic energy density
- local HF exchange energy densities

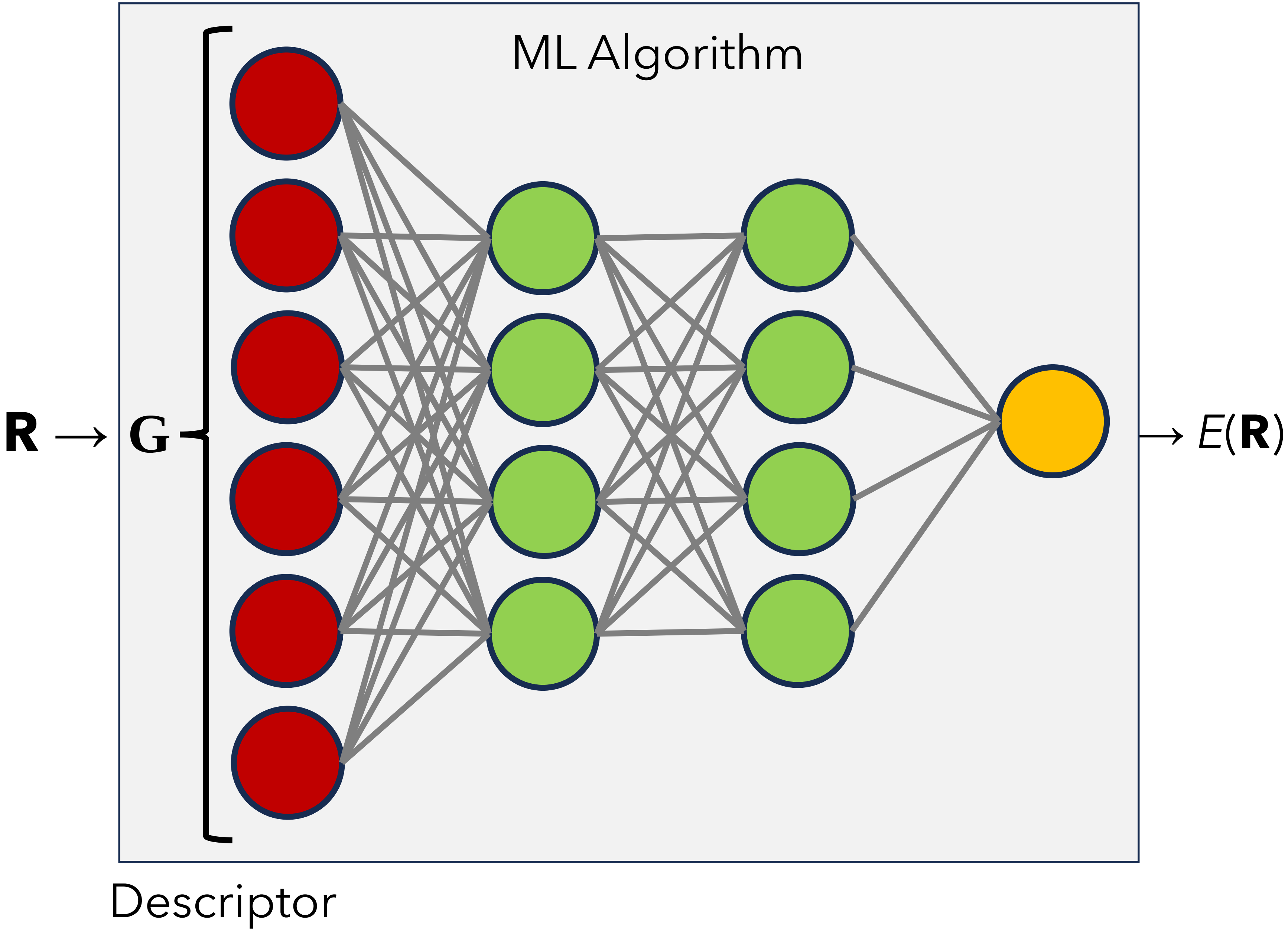
"The resulting functional, DM21 (DeepMind 21), correctly describes typical examples of artificial charge delocalization and strong correlation and performs better than traditional functionals on thorough benchmarks for main-group atoms and molecules. DM21 accurately models complex systems such as hydrogen chains, charged DNA base pairs, and diradical transition states."

Kirkpatrick *et al. Science* **2021**, *374*, 1385
Quanta magazine: tinyurl.com/qmdm21

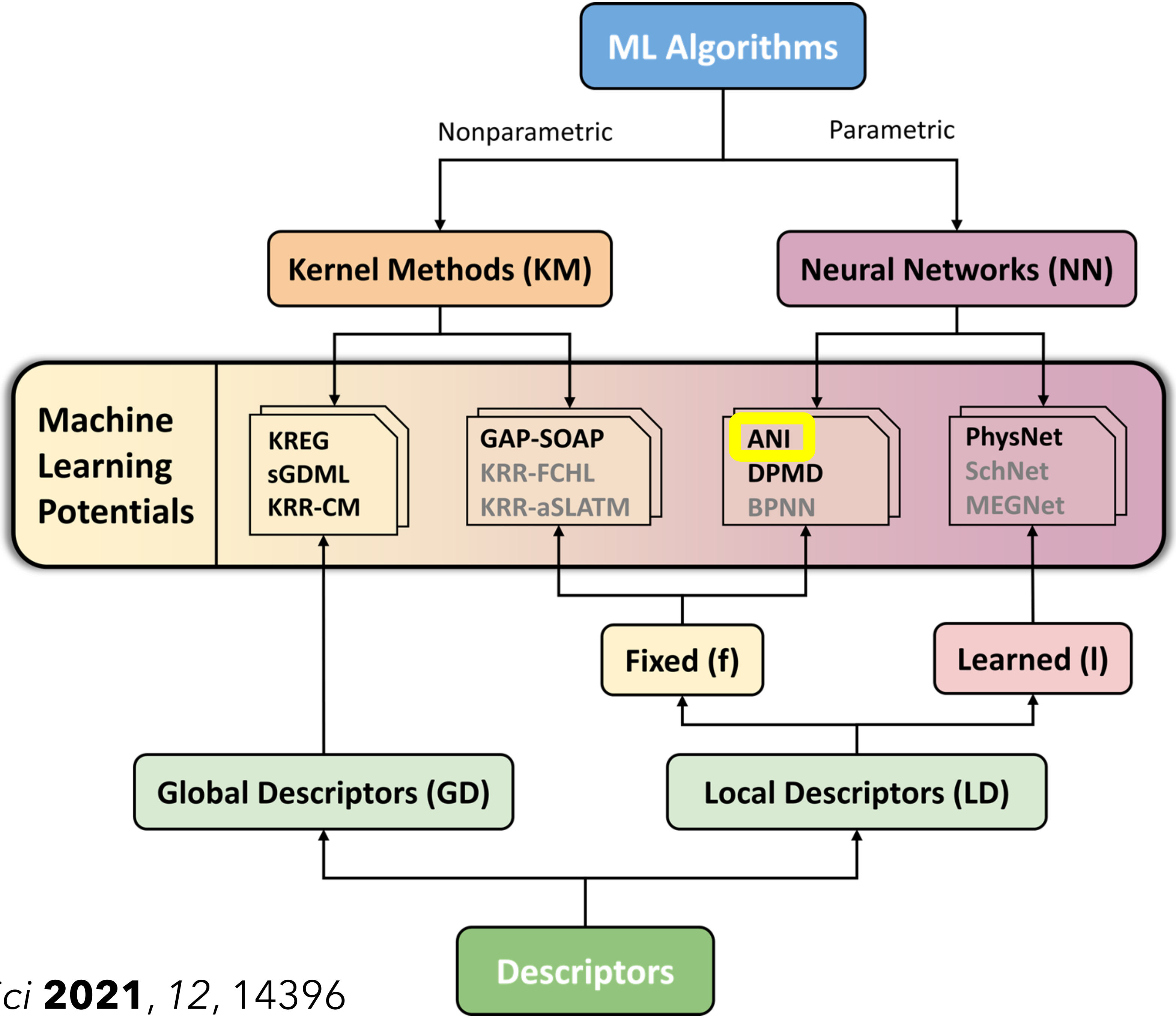


Predict properties as a surrogate approach: ML Potentials

$$\mathbf{R} \longrightarrow \boxed{\left(T_{elec}(\mathbf{r}) + V(\mathbf{r},\mathbf{R})\right)\varphi(\mathbf{r};\mathbf{R}) = E(\mathbf{R})\varphi(\mathbf{r};\mathbf{R})} \longrightarrow E(\mathbf{R})$$

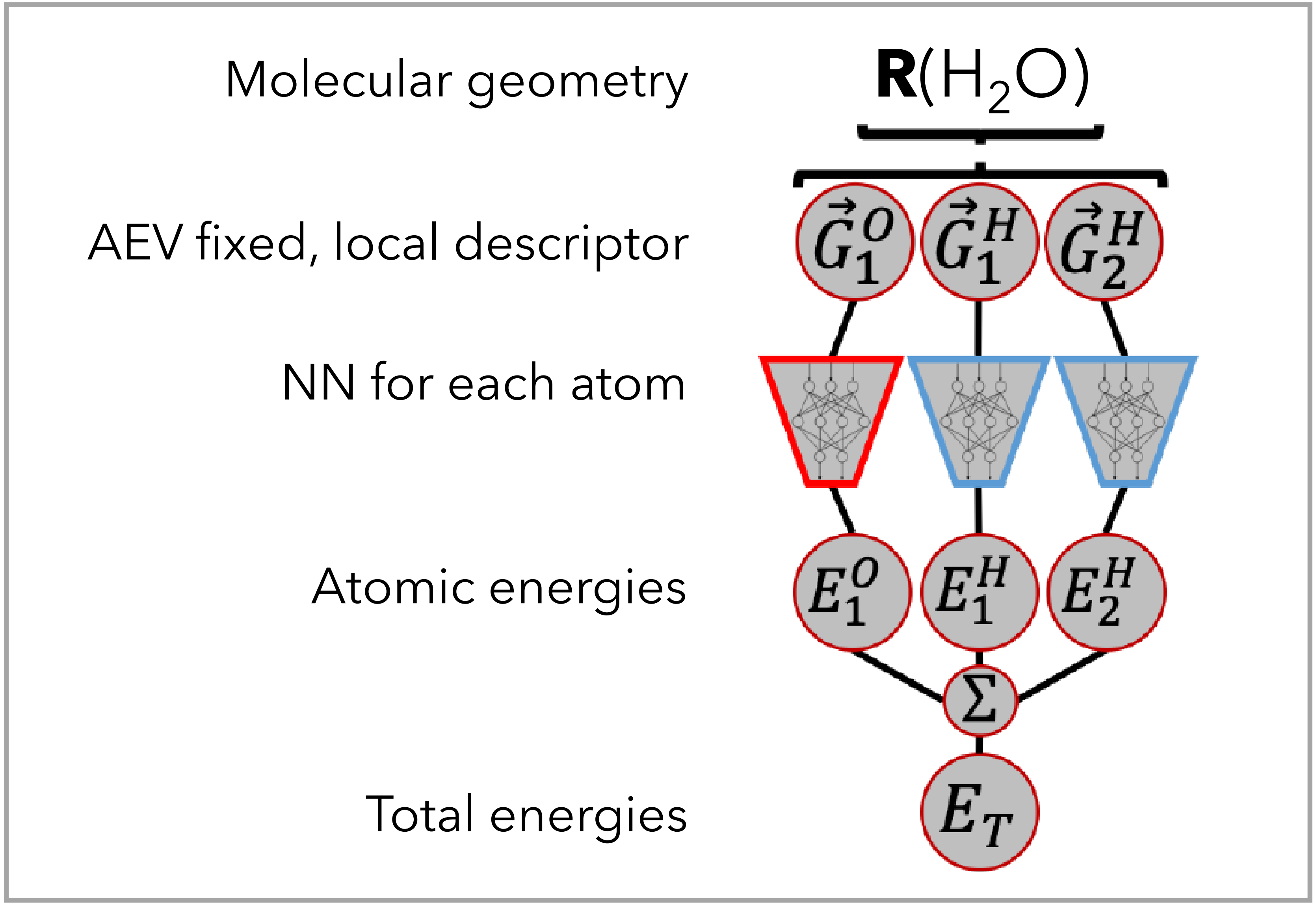


ML Algorithm
$\mathbf{R} \rightarrow \mathbf{G}$
$\rightarrow E(\mathbf{R})$
Descriptor

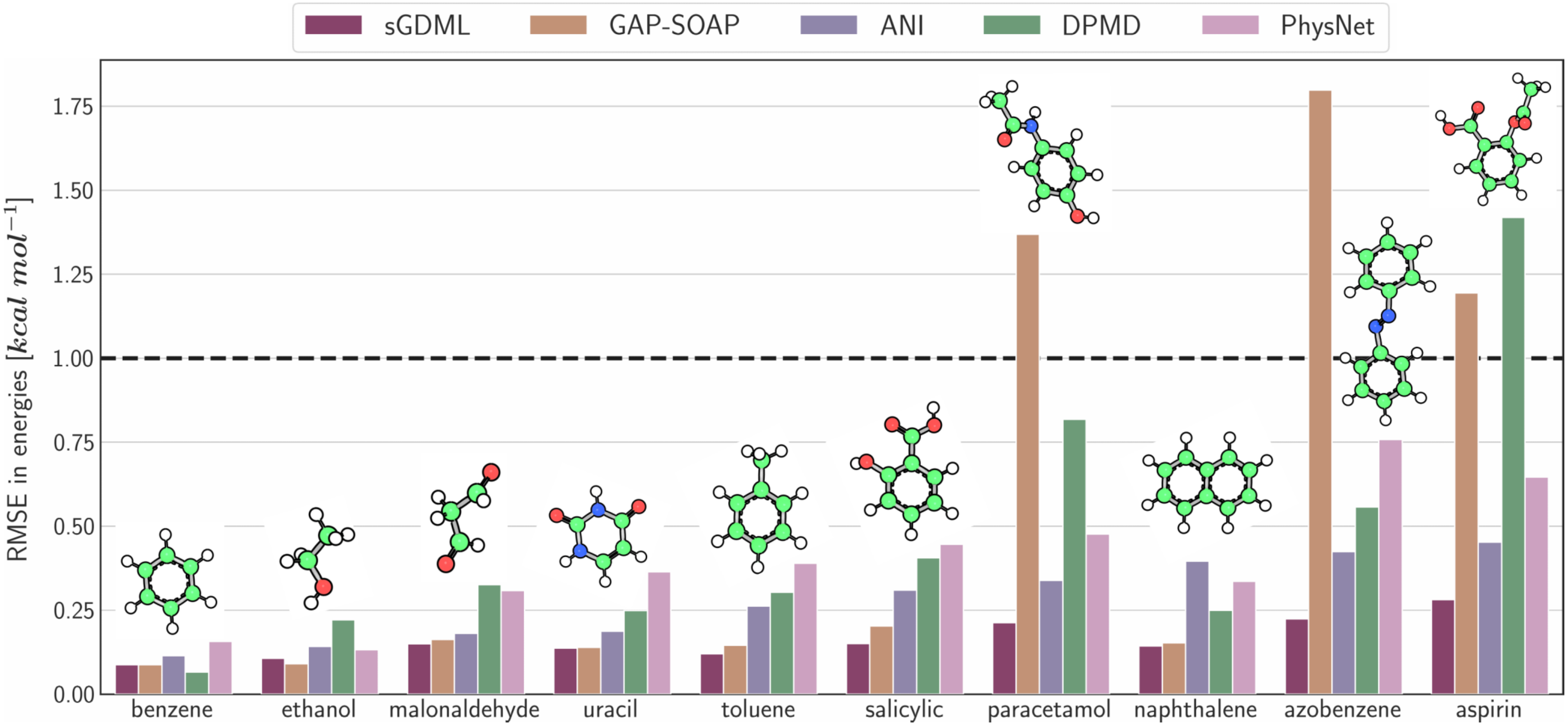


Pinheiro Jr *et al*. *Chem Sci* **2021**, *12*, 14396

Example: ANI ML Potential

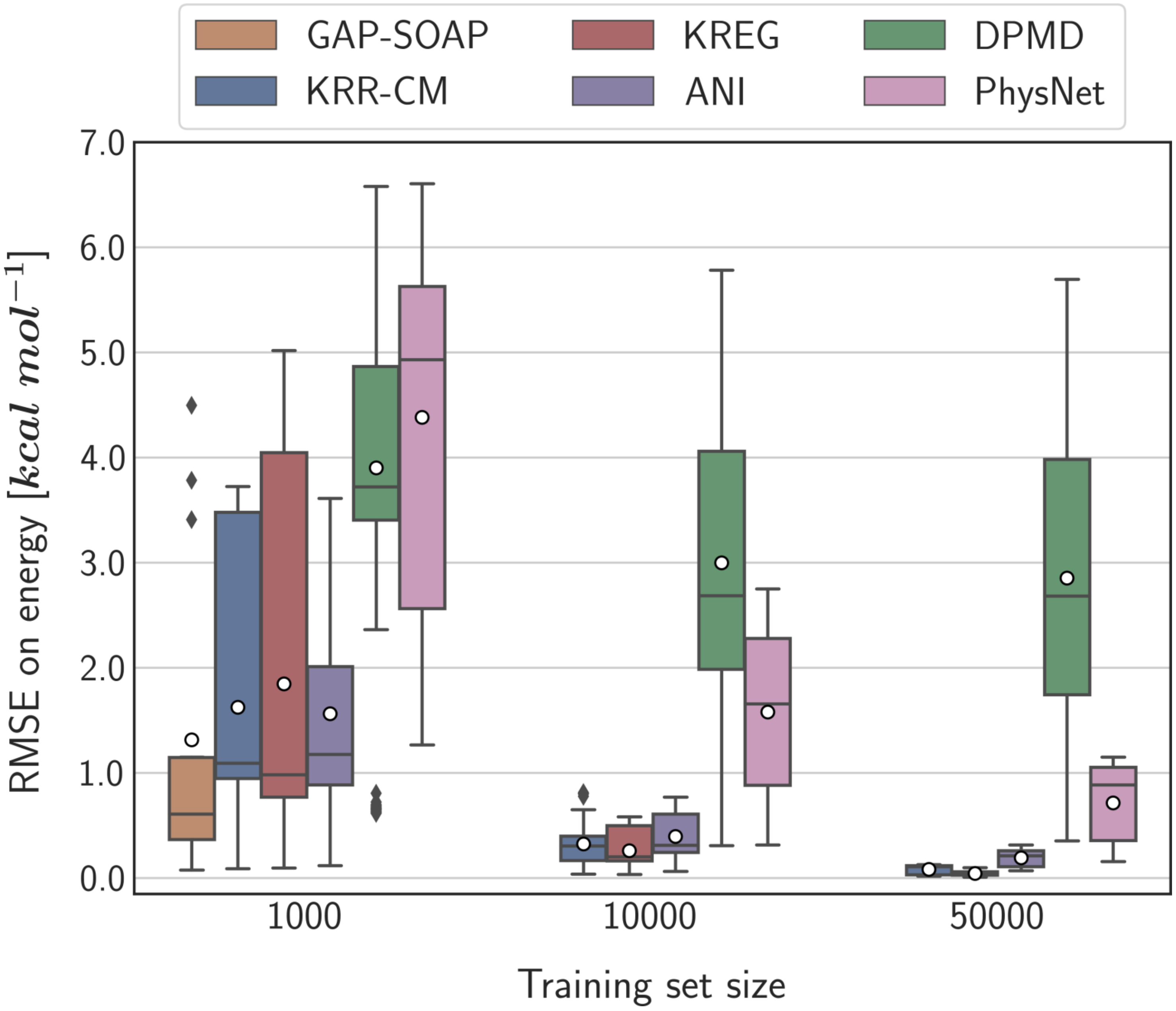


Gao *et al. J Chem Inf Model* **2020**, *60*, 3408



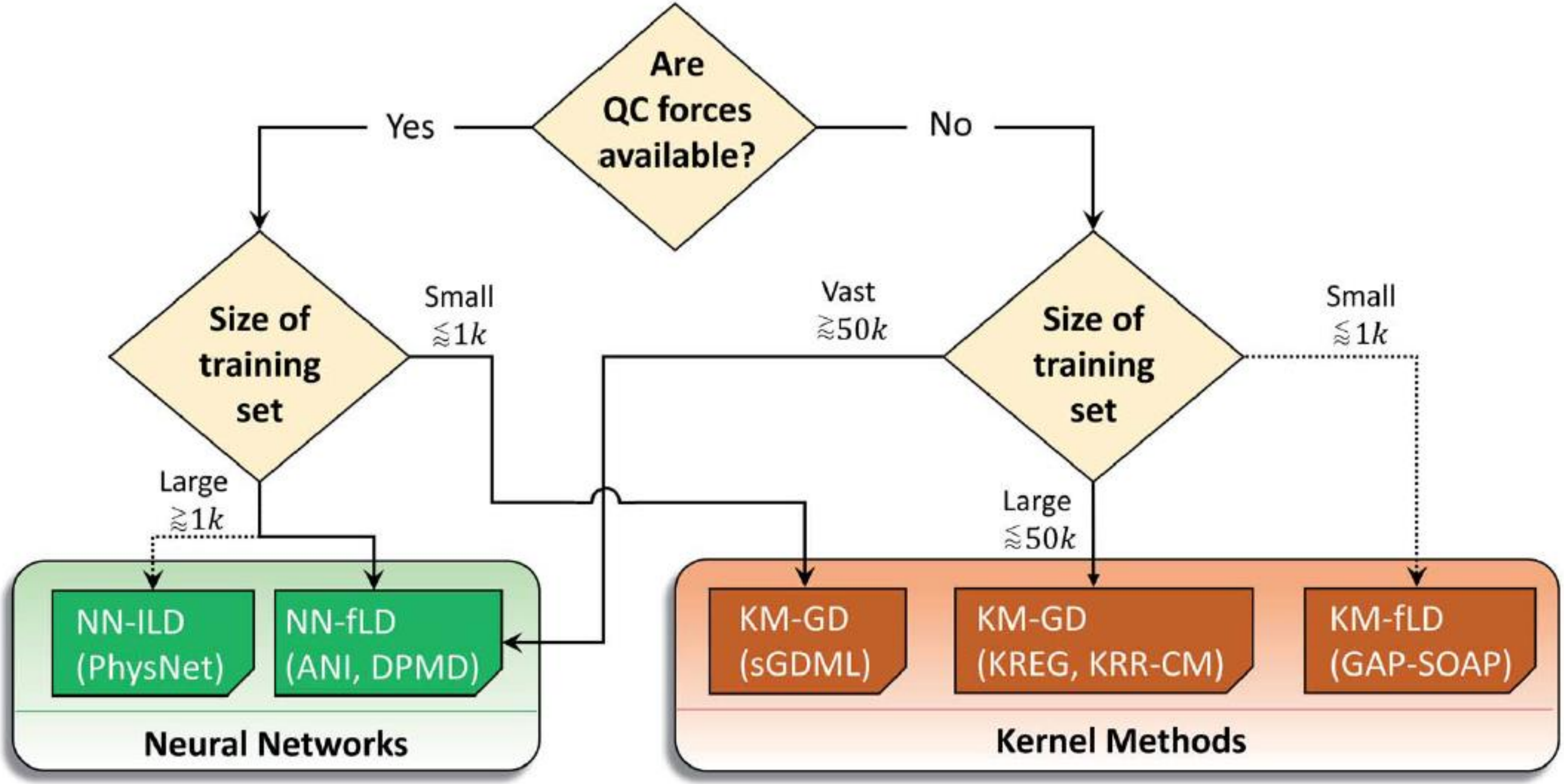
- MD17 Database
- Energy + Force
- N_{train} = 1k; N_{model} = 20; N_{test} = 20k

Pinheiro Jr *et al*. *Chem Sci* **2021**, *12*, 14396



- MD17 Database
- Energy only
- N_{test} = 20k

Pinheiro Jr *et al*. *Chem Sci* **2021**, *12*, 14396



Pinheiro Jr *et al*. *Chem Sci* **2021**, *12*, 14396

To know more:

3Blue1Brown Course on NN

- www.3blue1brown.com/topics/neural-networks

Kernel Methods

- Pinheiro Jr; Dral, In *Quantum chemistry in the age of machine learning,* **2023**; *pp 205*

ML Potentials

- Pinheiro Jr *et al*. *Chem Sci* **2021**, *12*, 14396

Deep Learning Applied to Computational Mechanics

- Vu-Quoc; Humer. *Comput Model Eng Sci* **2023**, *137*, 1069

Papers available for download at:
amubox.univ-amu.fr/s/xXAiMZrDPb9RMRX (Ask me for the password)

Computational modeling of nanosystems

Epiloge

many

Quantum Statistical Mechanics

Classical Statistical Mechanics

Computational Modeling of Nanosystems

Scientific skills	Math skills	Operational skills
• Quantum chemistry • Molecular dynamics • Physical chemistry	• Linear algebra • Statistics • Machine learning	• Modeling • Programming • Data processing

few

Quantum Mechanics

Classical Mechanics

Quantum field theory
Standard model

small

large

General relativity
Dark matter/energy