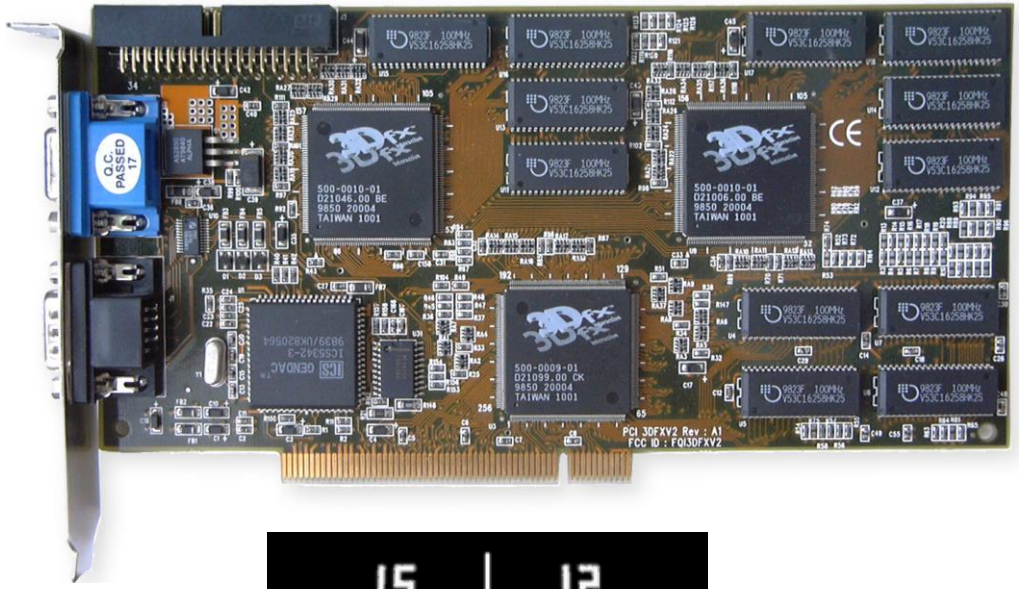
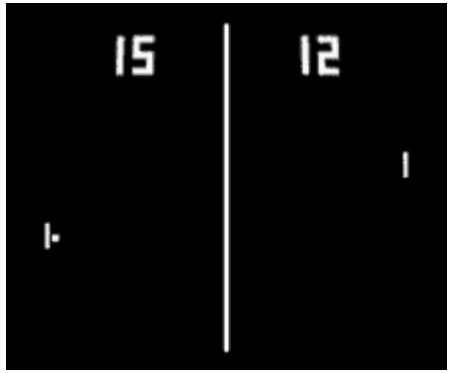
Deep neural networks

Xkdc

Graphical Processing Unit (GPU)

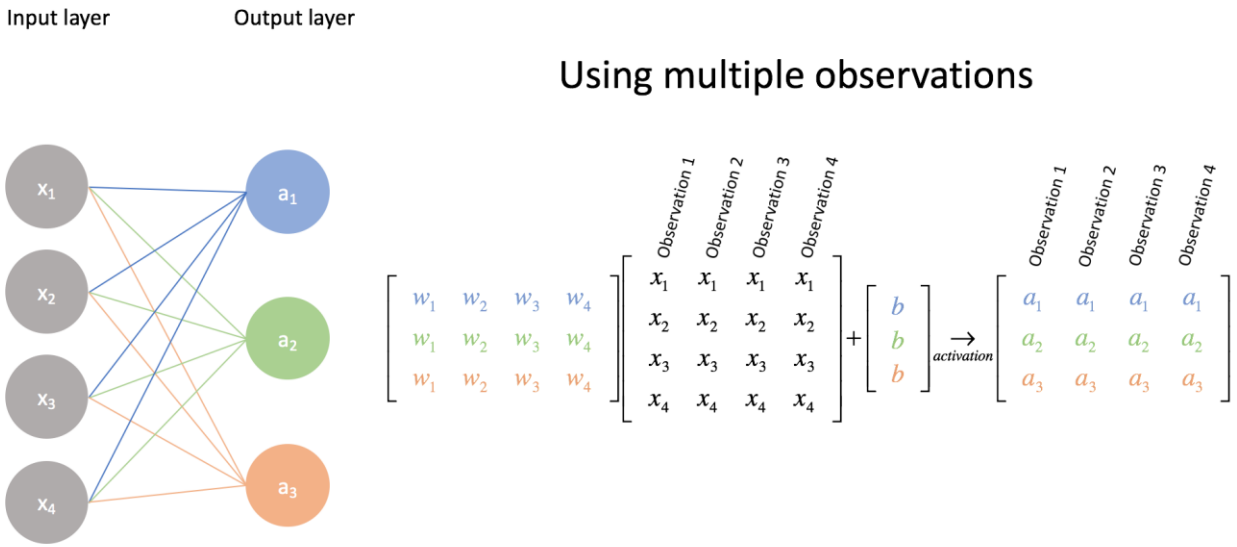


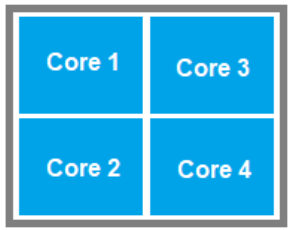


GPU vs CPU

A **CPU** (Central Processing Unit) is optimized for sequential processing and complex tasks, featuring a few powerful cores.

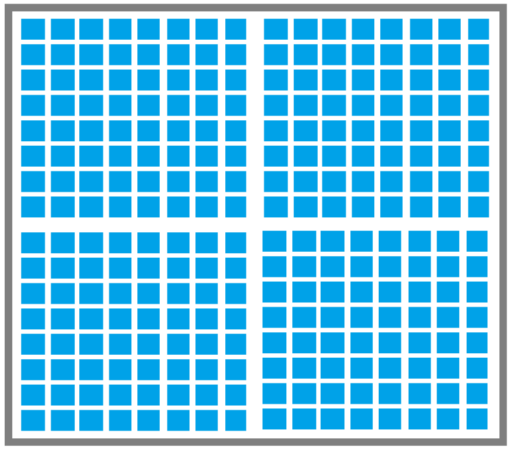
A **GPU** (Graphics Processing Unit) is designed for parallelism, with thousands of smaller cores handling many simple operations simultaneously.





(Fewer strong cores)

CPU



(Thousands weaker cores)

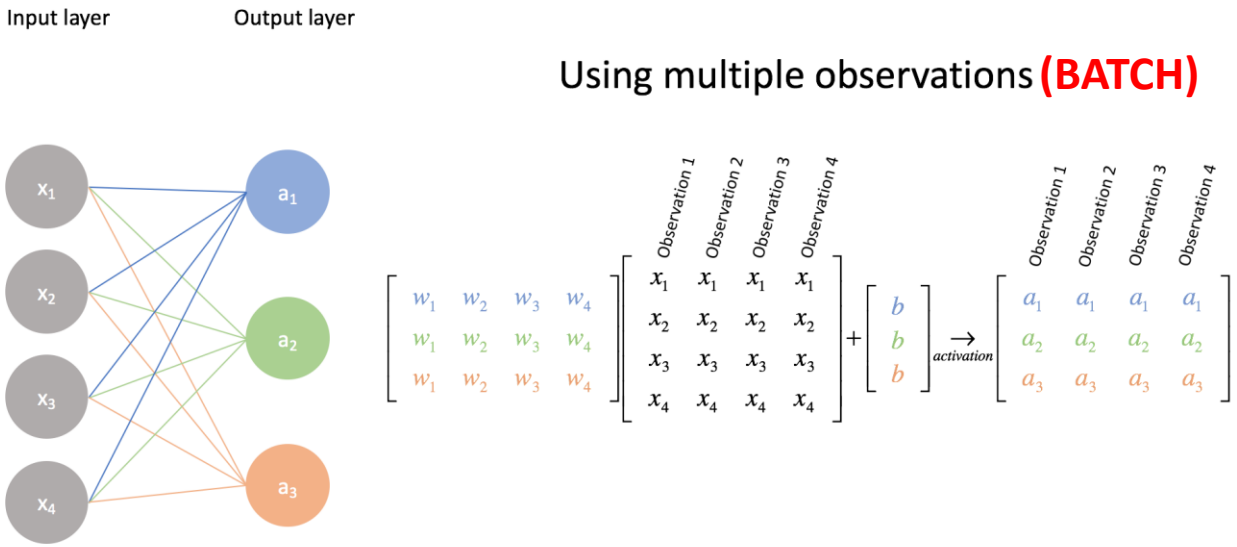
GPU

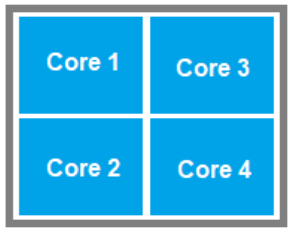
Image by: tecadmin.net

GPU vs CPU

A **CPU** (Central Processing Unit) is optimized for sequential processing and complex tasks, featuring a few powerful cores.

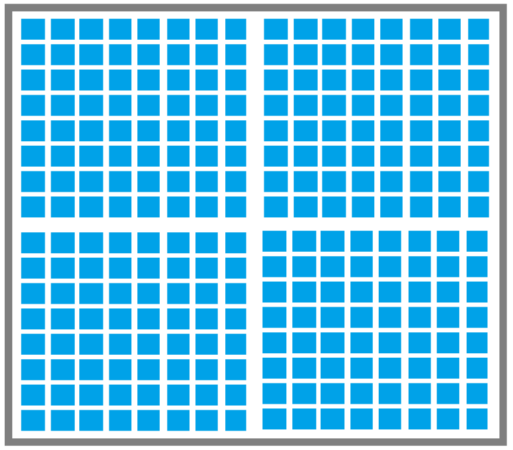
A **GPU** (Graphics Processing Unit) is designed for parallelism, with thousands of smaller cores handling many simple operations simultaneously.





(Fewer strong cores)

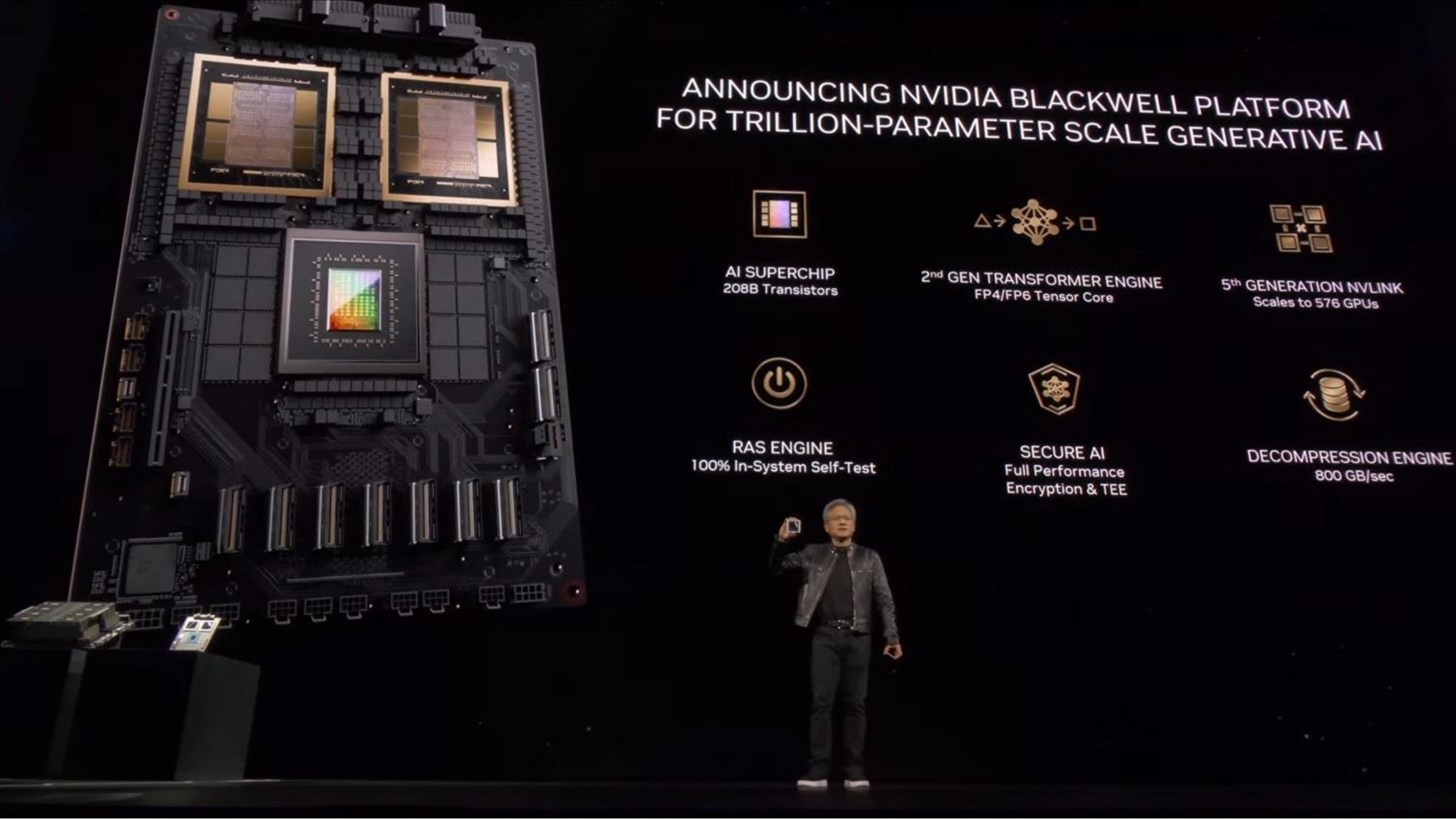
CPU



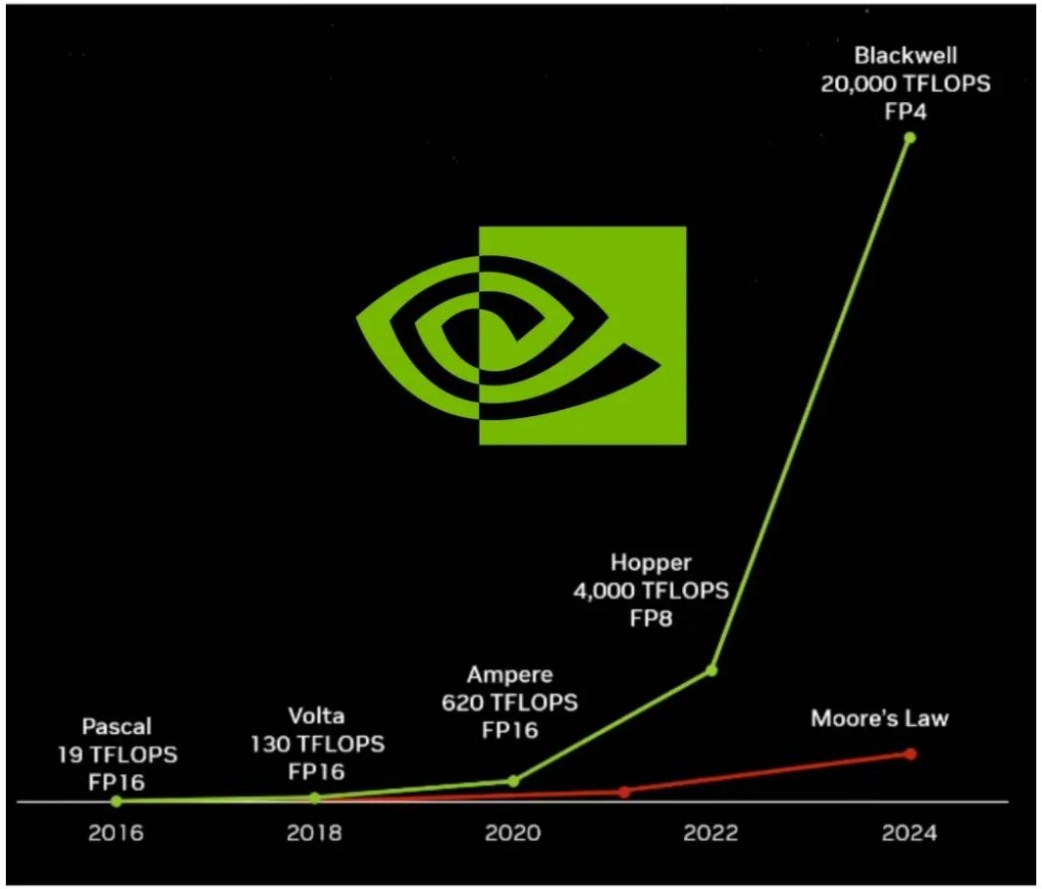
(Thousands weaker cores)

GPU

Image by: tecadmin.net


ANNOUNCING NVIDIA BLACKWELL PLATFORM
FOR TRILLION-PARAMETER SCALE GENERATIVE AI
AI SUPERCHIP
208B Transistors
2nd GEN TRANSFORMER ENGINE
FP4/FP6 Tensor Core
5th GENERATION NVLINK
Scales to 576 GPUs
RAS ENGINE
100% In-System Self-Test
SECURE AI
Full Performance
Encryption & TEE
DECOMPRESSION ENGINE
800 GB/sec

Graphical Processing Unit (GPU)



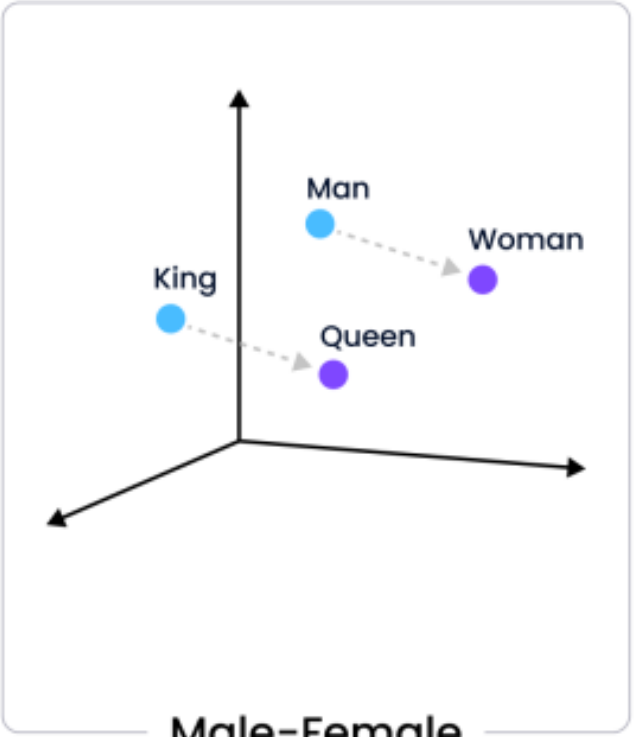
Deep learning: what it really is (to my best understanding)

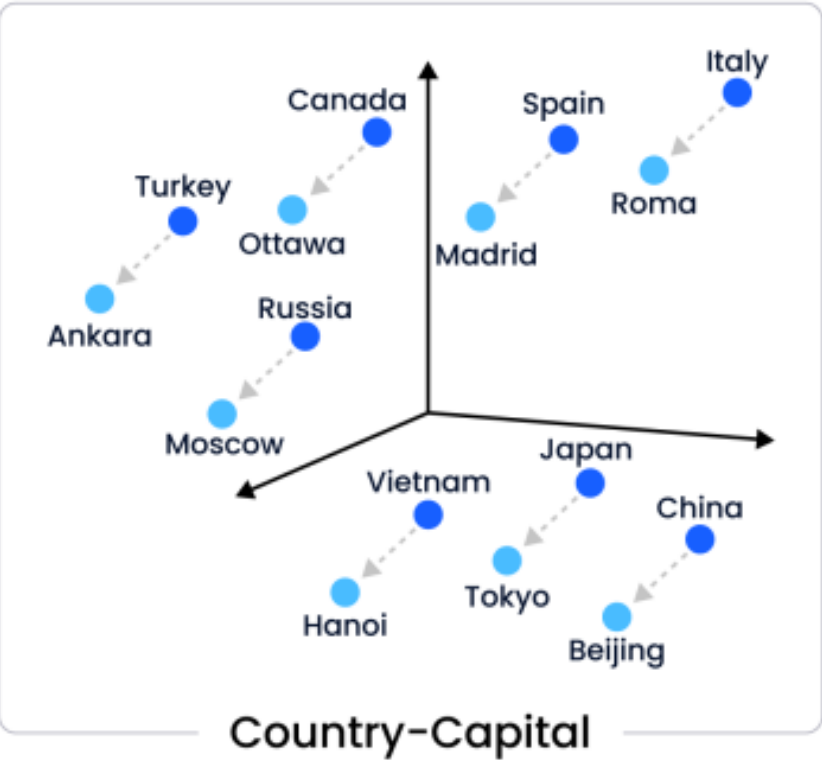
Deep learning models distill vast amounts of information into relatively small, high-dimensional vector **embeddings** that efficiently capture diverse concepts, facts, and nuances.

These embeddings aren't merely compressed data, they encapsulate intricate relationships and **latent** features, enabling models to recognize and generalize patterns across different domains.

By leveraging **superposition**, deep learning allows multiple patterns or concepts to coexist within a single vector, maximizing representational capacity without a proportional increase in size.

Deep learning: what it really is (to my best understanding)





Neural network

Neuron activation as logistic regression: Each neuron in a neural network functions like a logistic regression unit, applying a weighted sum of inputs followed by a nonlinear activation function (often a sigmoid in early networks).

Multi-layer extension: A neural network extends logistic regression by stacking multiple layers of these units, allowing it to model complex, non-linear relationships beyond what a single logistic regression model can achieve.

Backpropagation as gradient descent: Training a neural network uses backpropagation, which is essentially gradient descent applied to multiple layers of logistic regression units, adjusting weights to minimize error across the entire network.

Logistic regression

Fit a logistic model

$$f(x,\theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m) = g(\theta' x)$$

such that the cost function

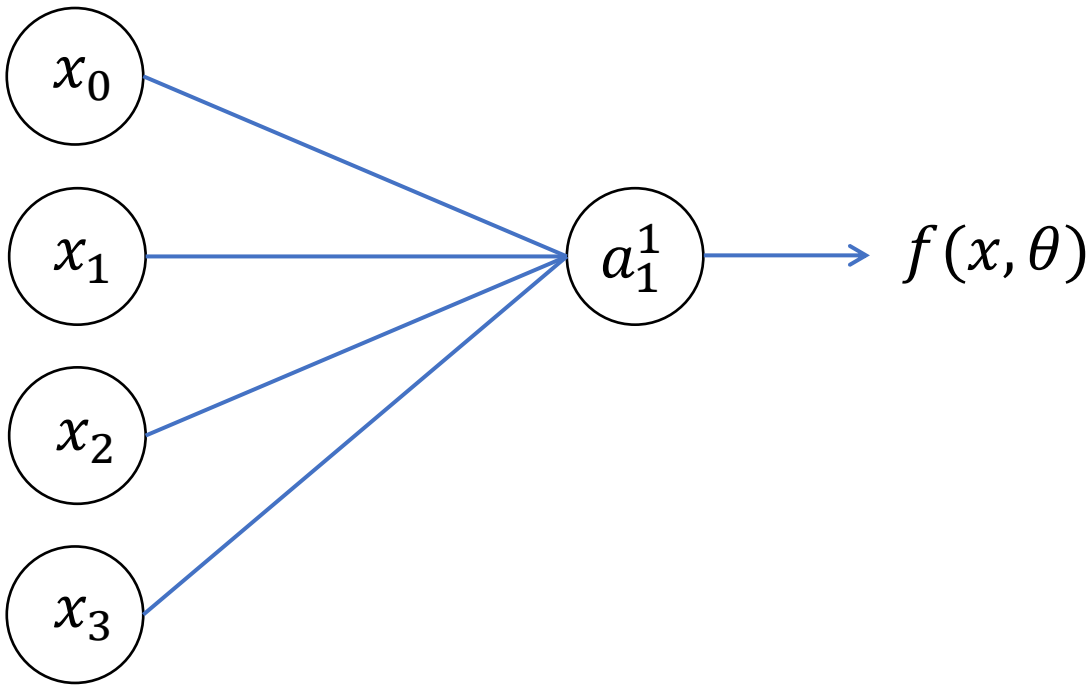
$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)},\theta))]$$

is minimised with the gradient descent algorithm:

$$\theta_j := \theta_j - \alpha \frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_j^{(i)}$$

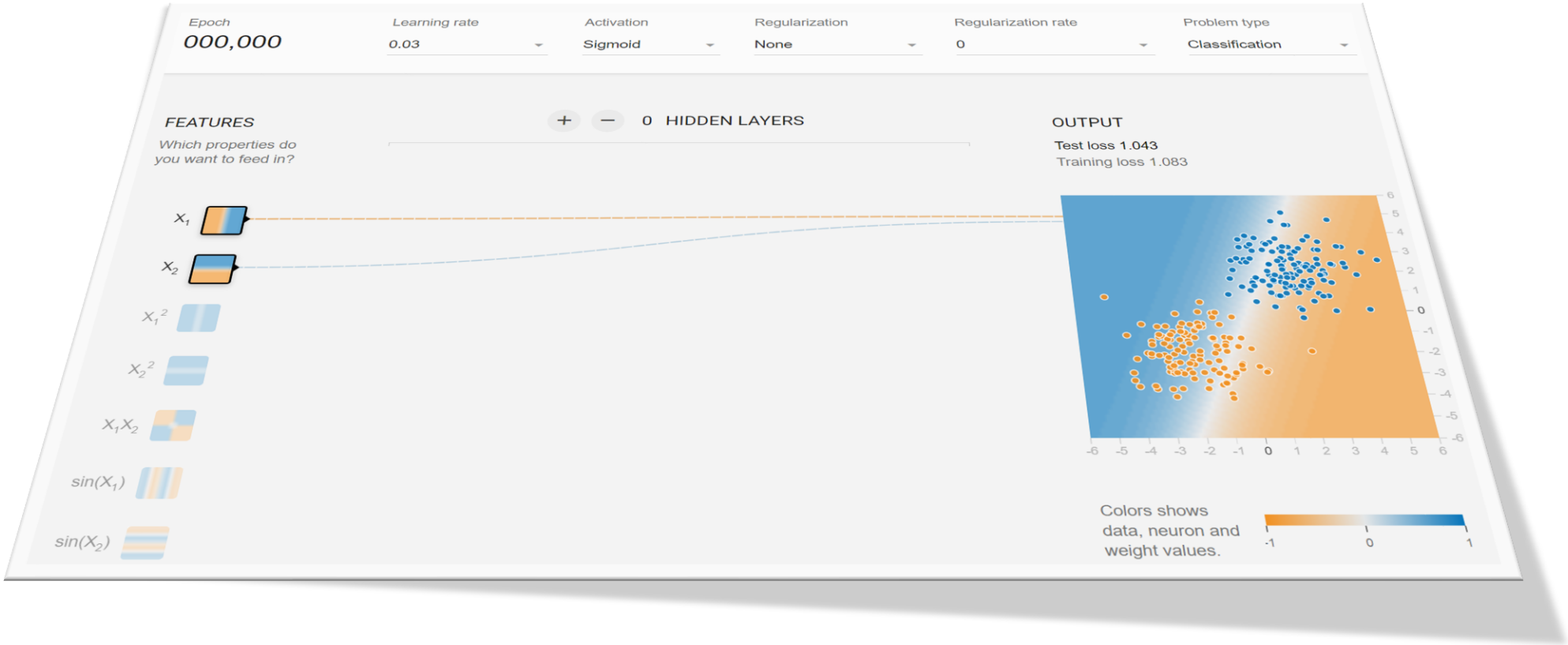
Neural network

We can visualize a logistic regression model as follows:



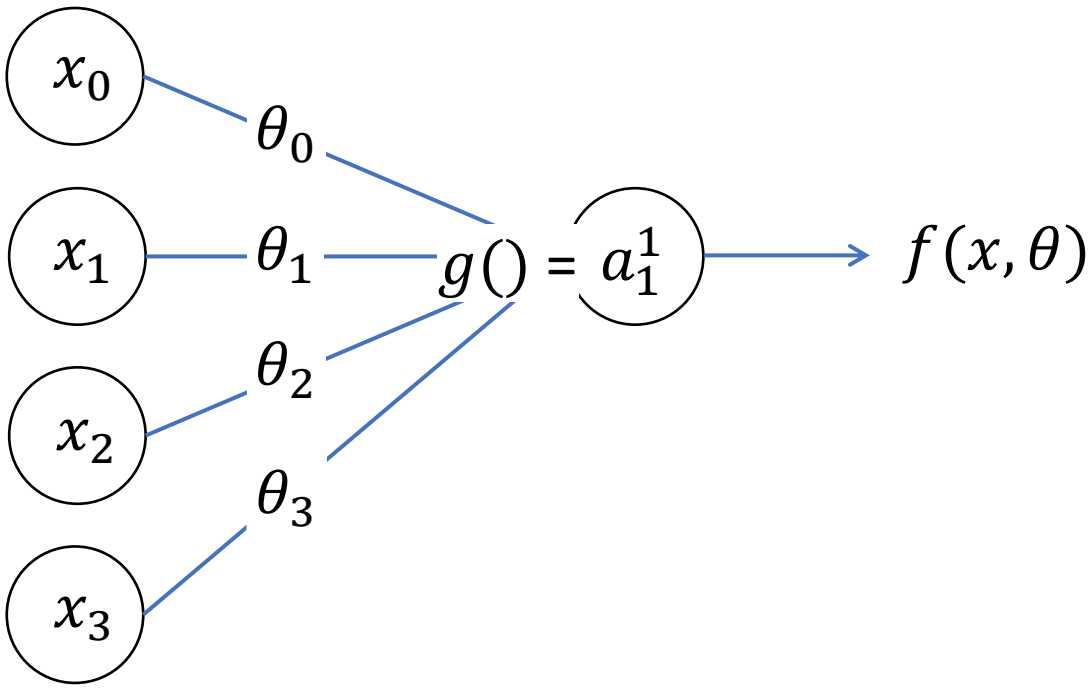
Neural network

We can visualize a logistic regression model as follows (ctrl-click image):



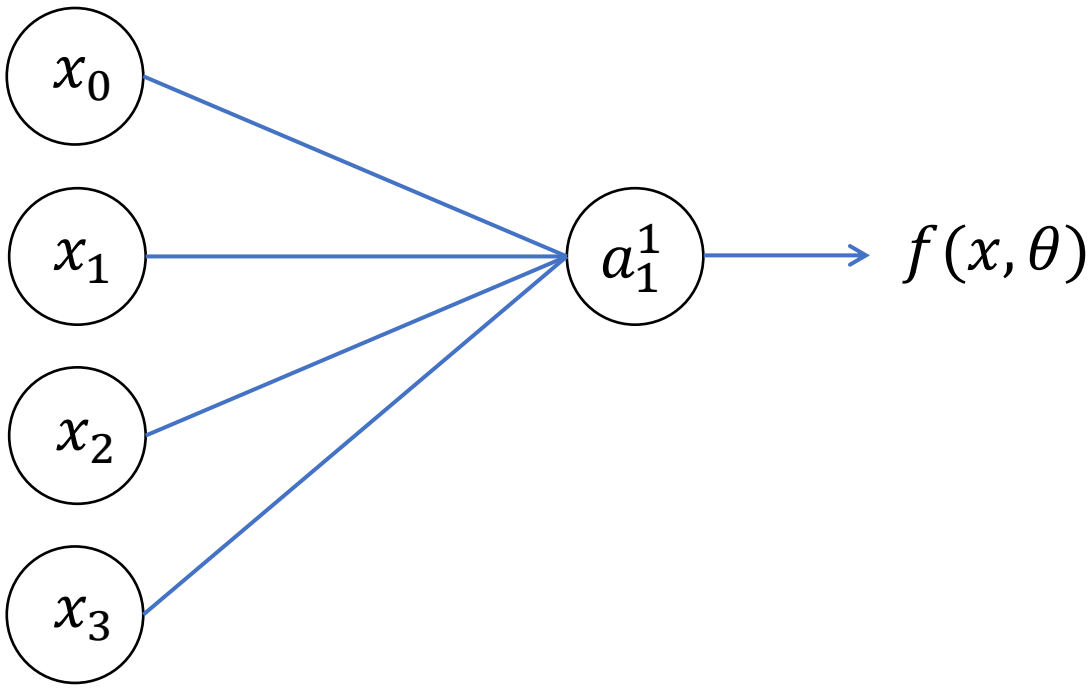
Neural network

We can visualize a logistic regression model as follows:



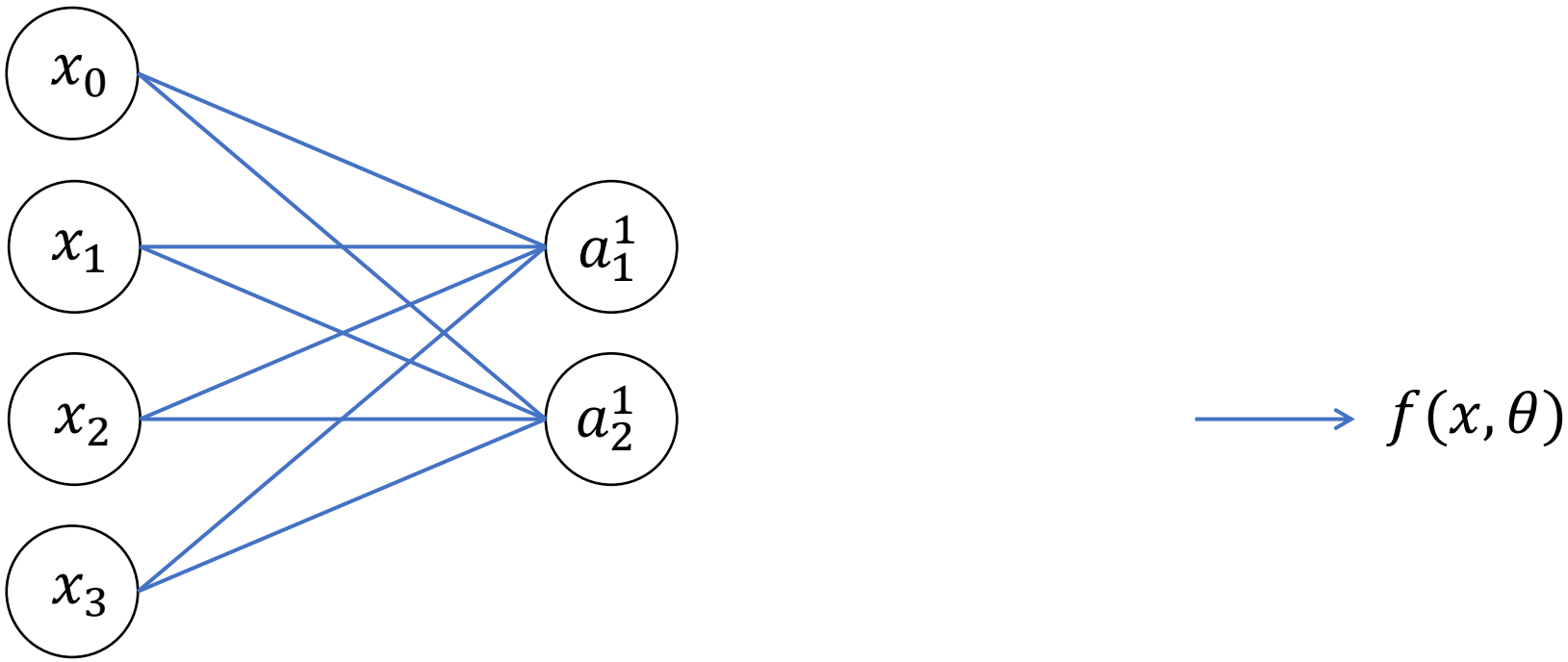
Neural network

We can visualize a logistic regression model as follows:



Neural network

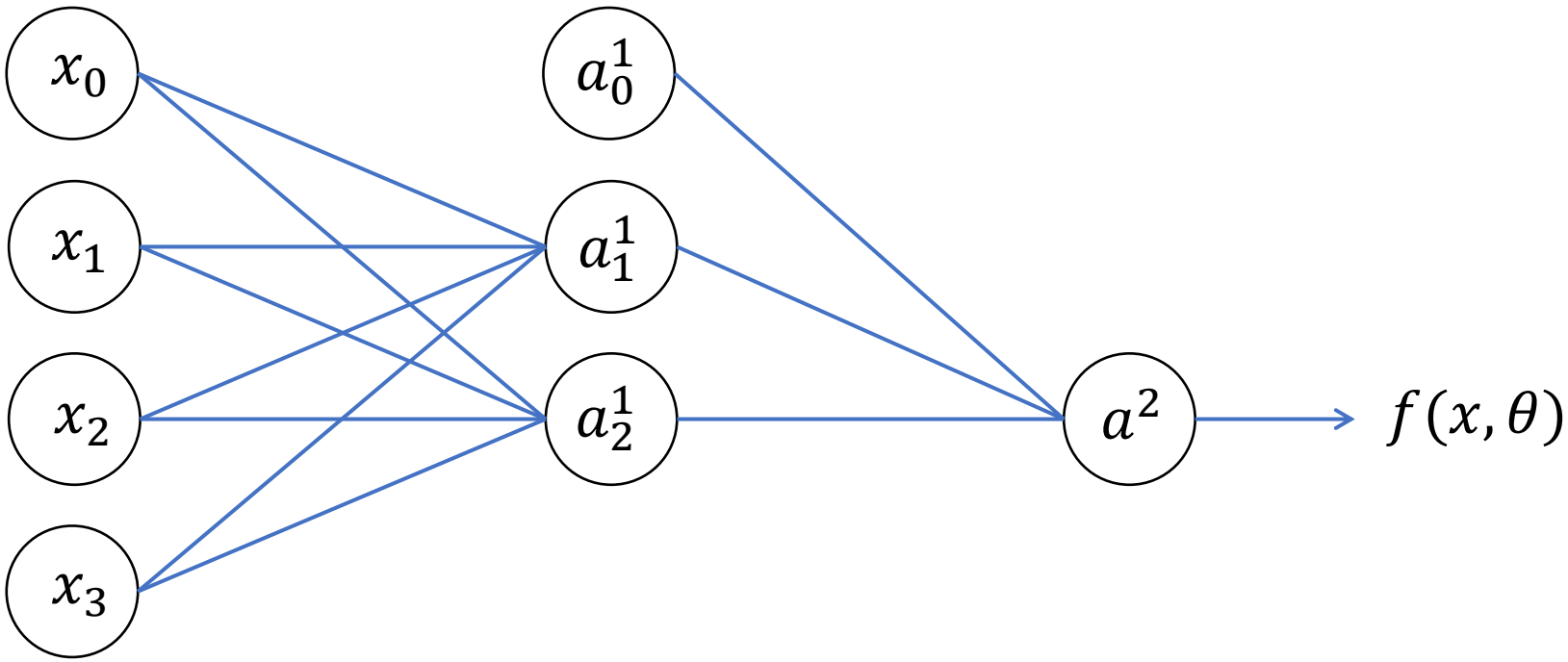
A neural network is an **architecture** that consists of logistic models:



An additional logistic model is added as a **node** in the **model's first (hidden) layer**.

Neural network

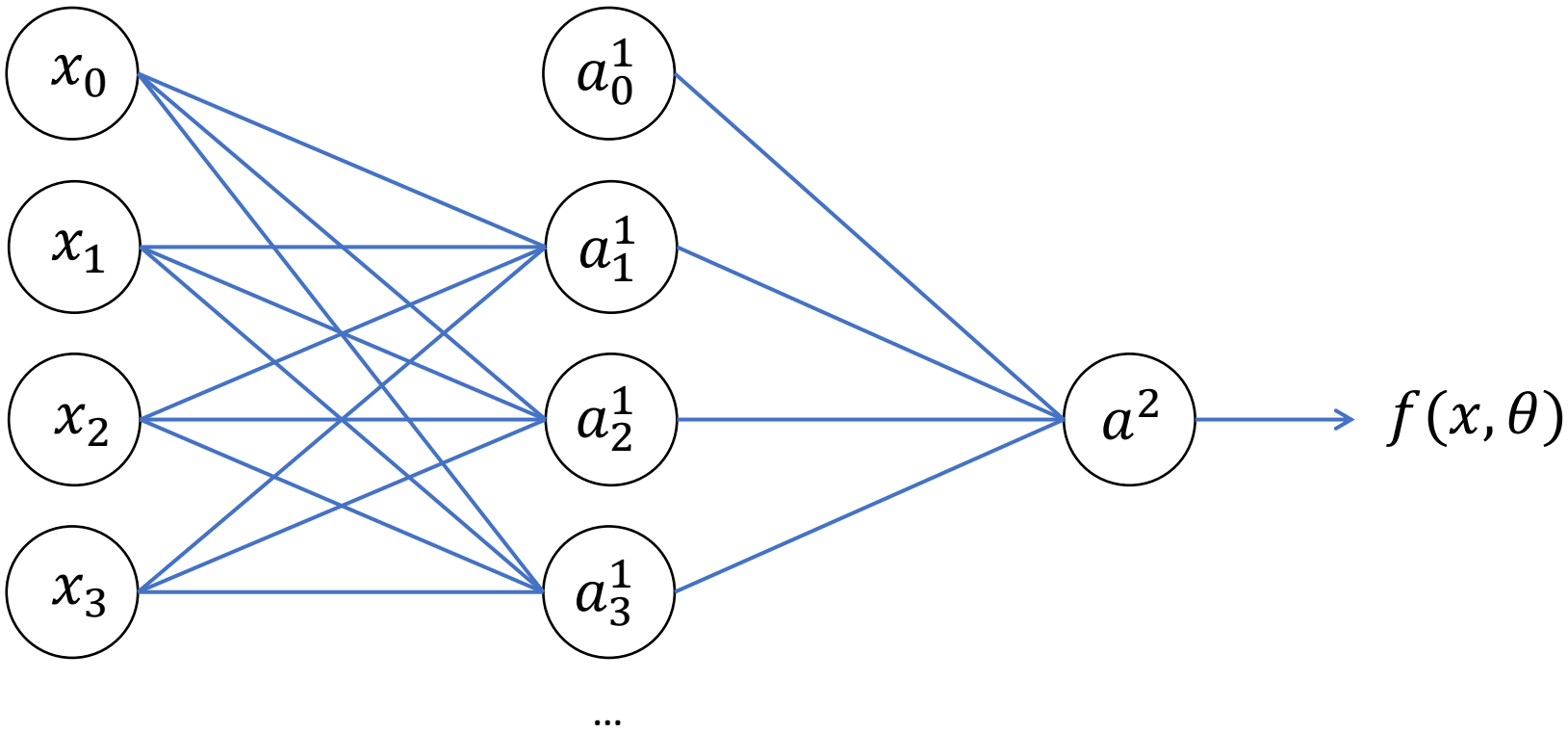
A neural network is an **architecture** that consists of logistic models:



An additional logistic model is added as a **node** in the **model's first (hidden) layer**, followed by a logistic regression model in the **output layer**.

Neural network

A neural network is an **architecture** that consists of logistic models:



More additional logistic models can be added as **nodes** in the **model's first (hidden) layer**, followed by a logistic regression model in the **output layer**.

Non-linearity: activation function

An **activation function** determines whether a neuron should be activated by introducing non-linearity into the network, allowing it to learn complex patterns.

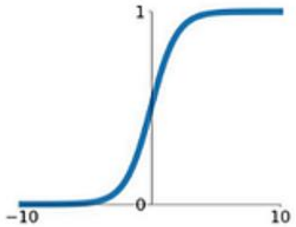
Common activation functions include **ReLU** (Rectified Linear Unit) for efficiency, Sigmoid for probabilities, and Tanh for zero-centered outputs.

Choosing the right activation function impacts model performance, affecting convergence speed, vanishing/exploding gradients, and overall network accuracy.

Non-linearity: activation function

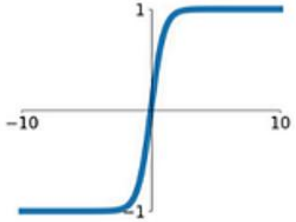
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



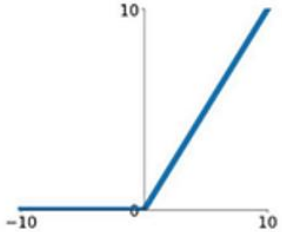
tanh

$$\tanh(x)$$



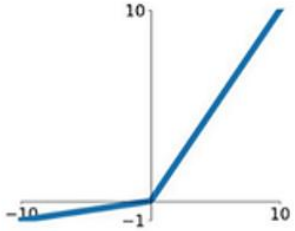
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

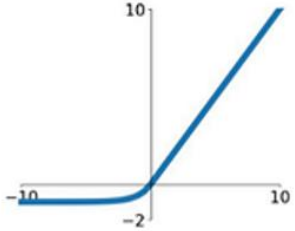


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

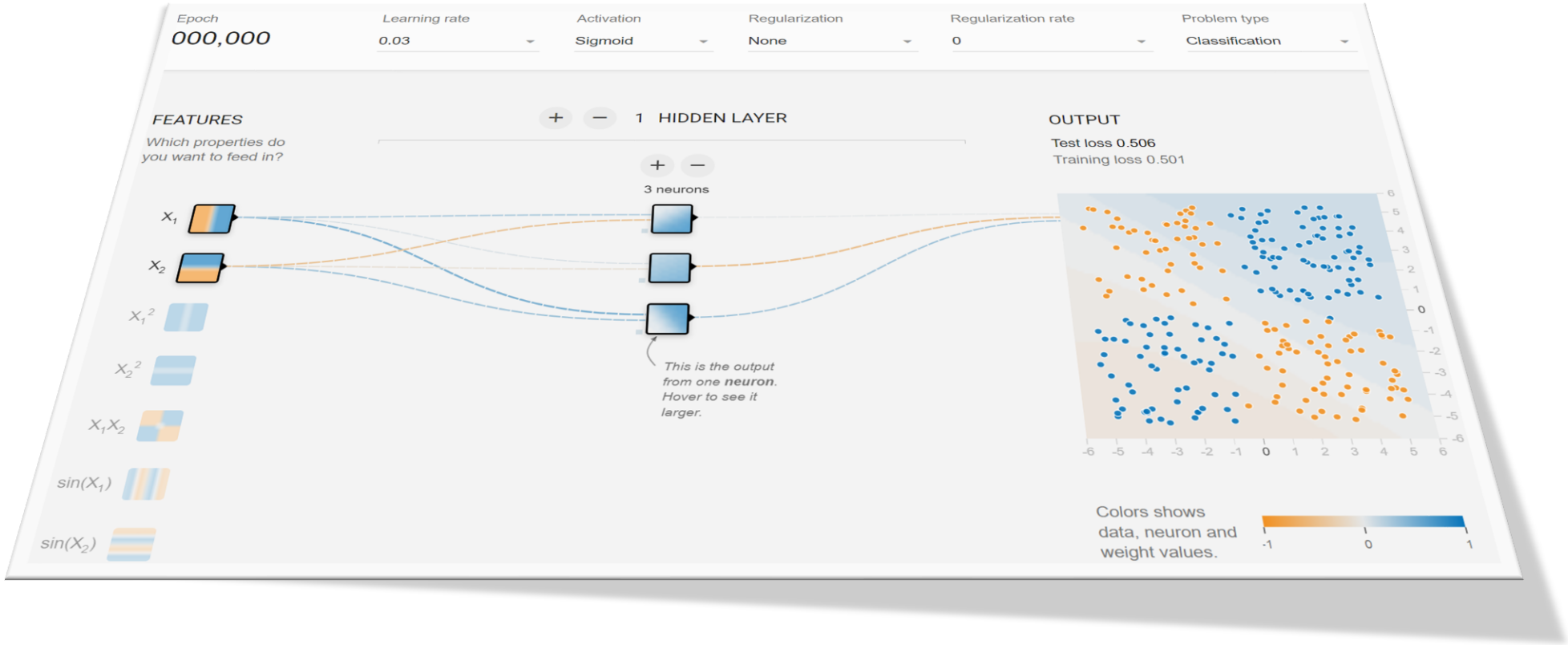
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural network

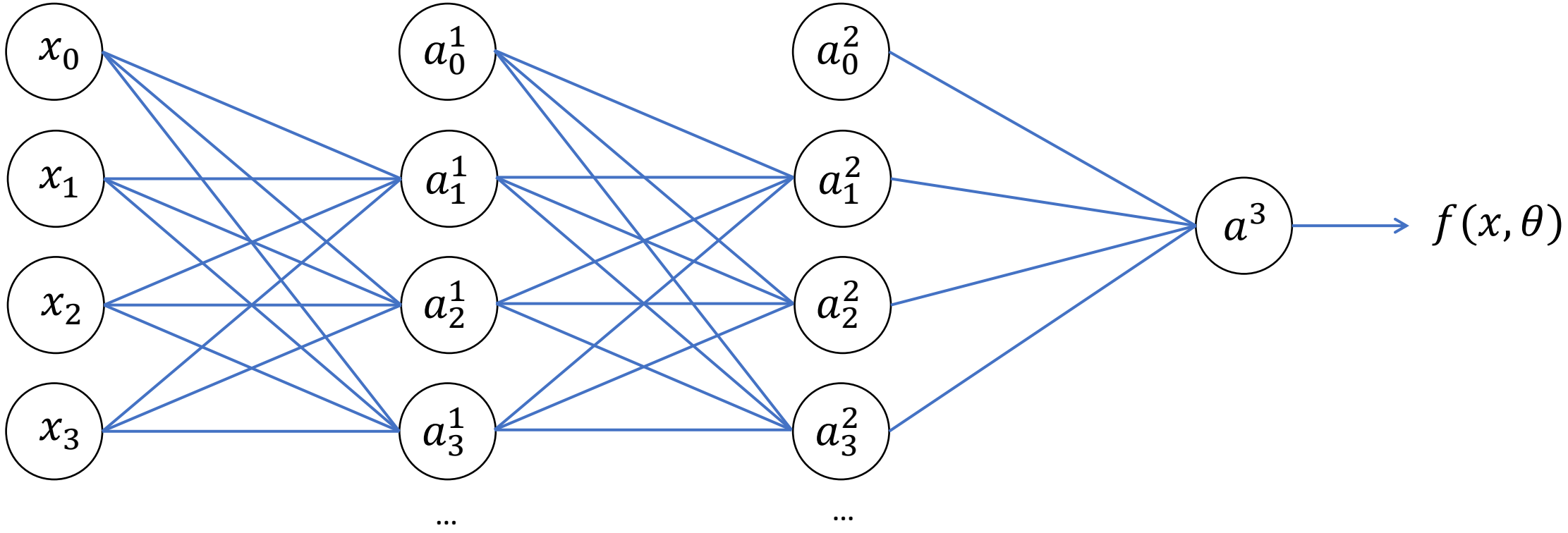
A neural network is an **architecture** that consists of logistic models (ctrl-click image):



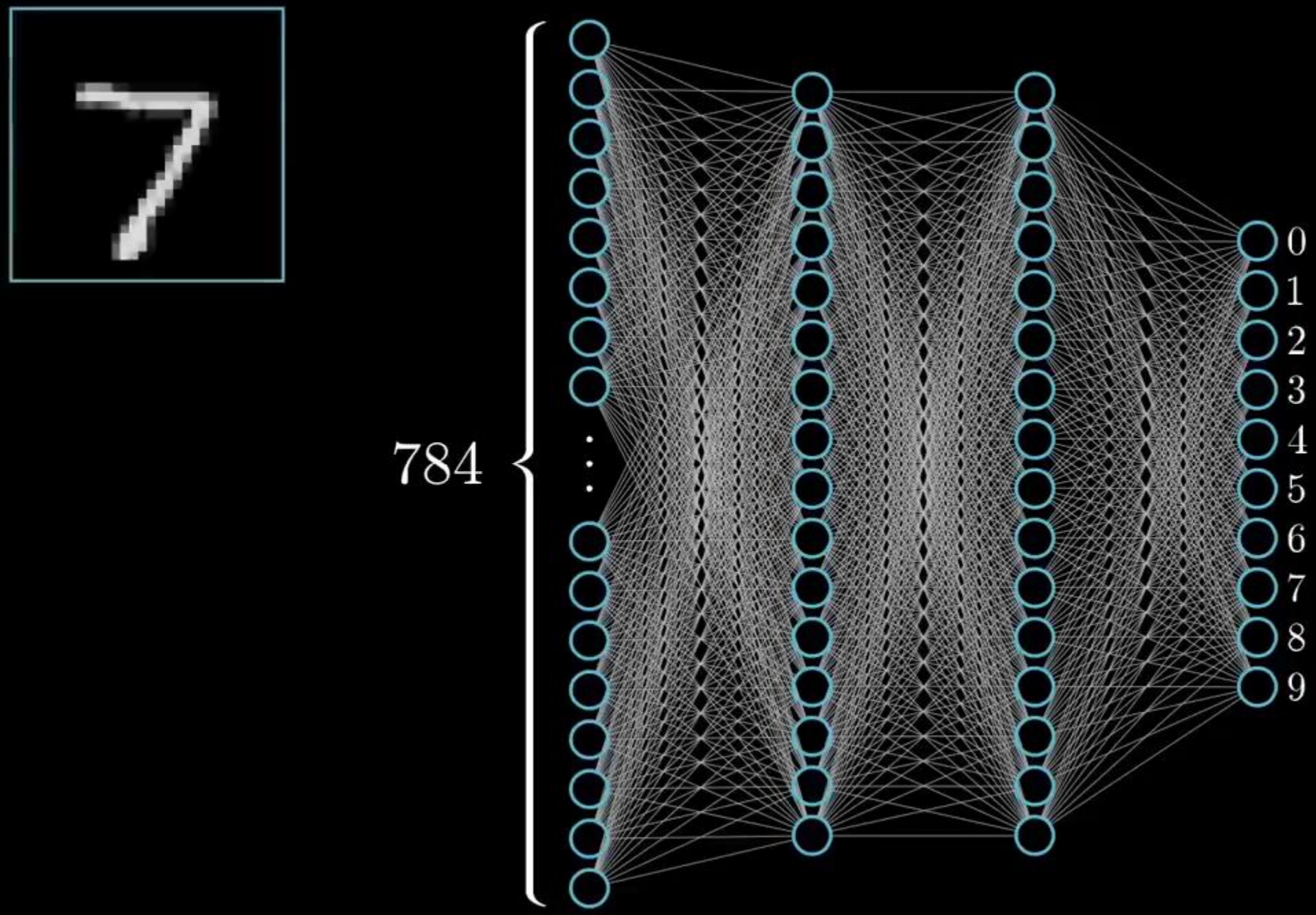
More additional logistic models can be added as **nodes** in the **model's first (hidden) layer**, followed by a logistic regression model in the **output layer**.

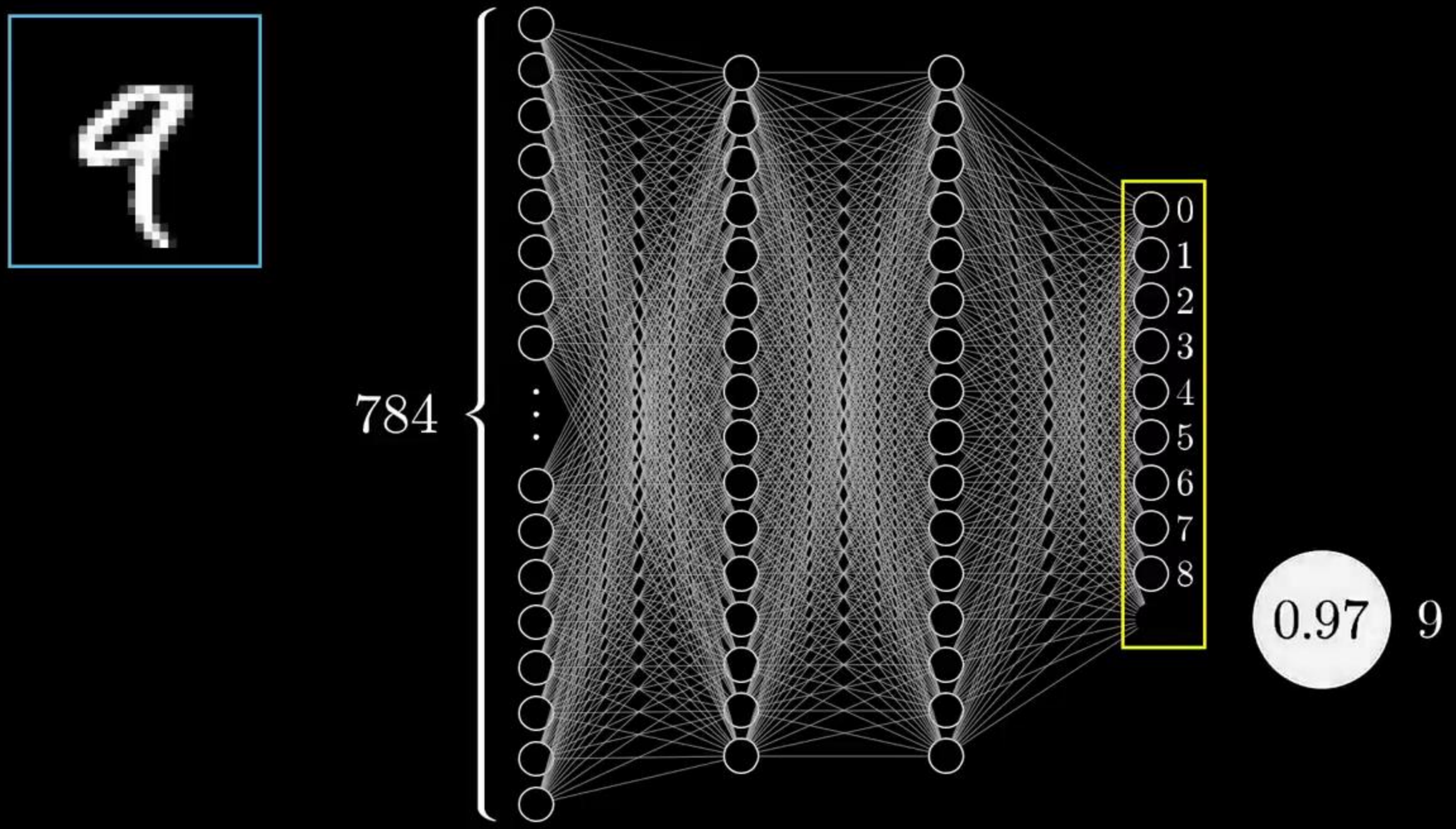
Deep neural network

A neural network is an **architecture** that consists of logistic models:

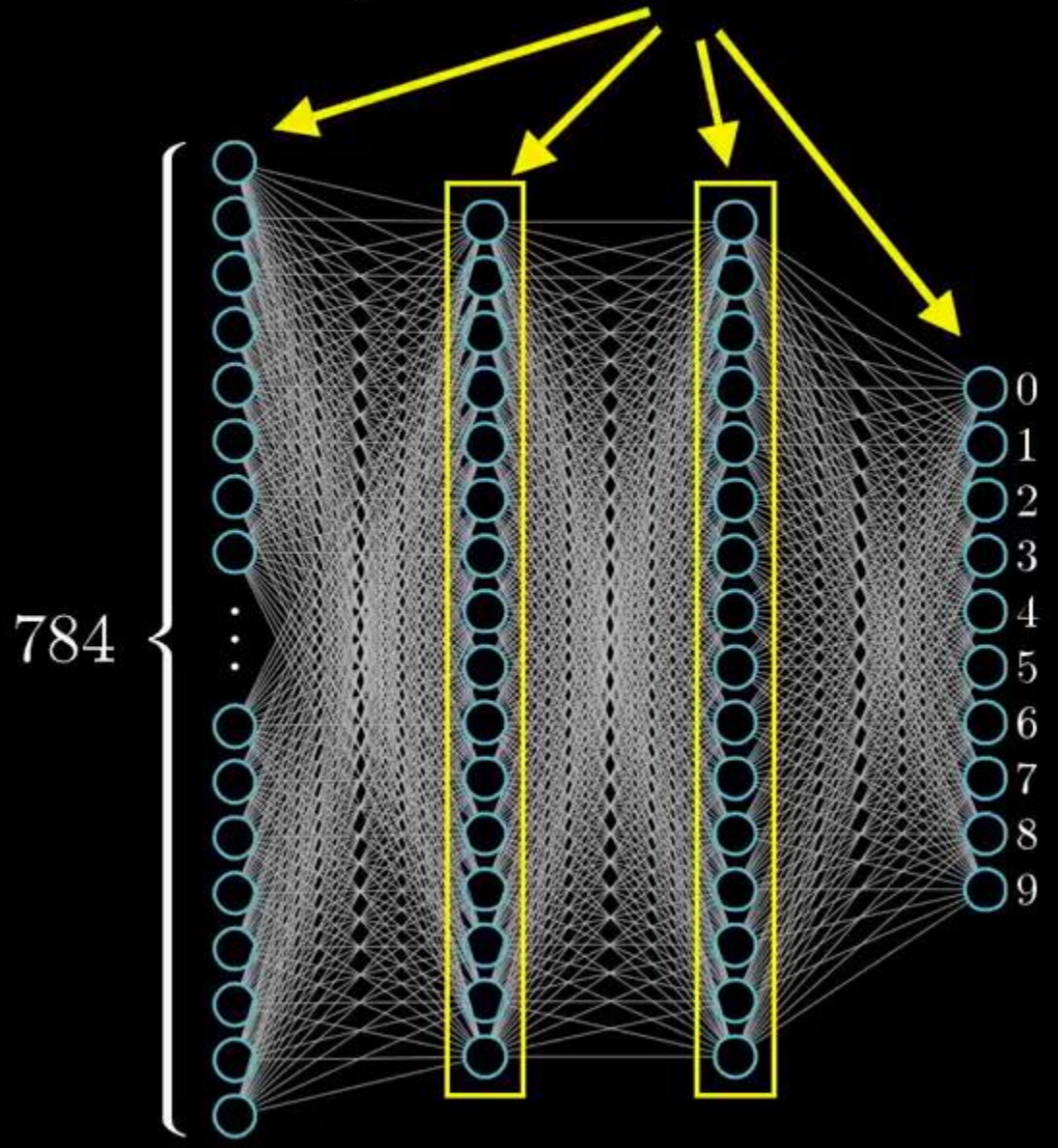


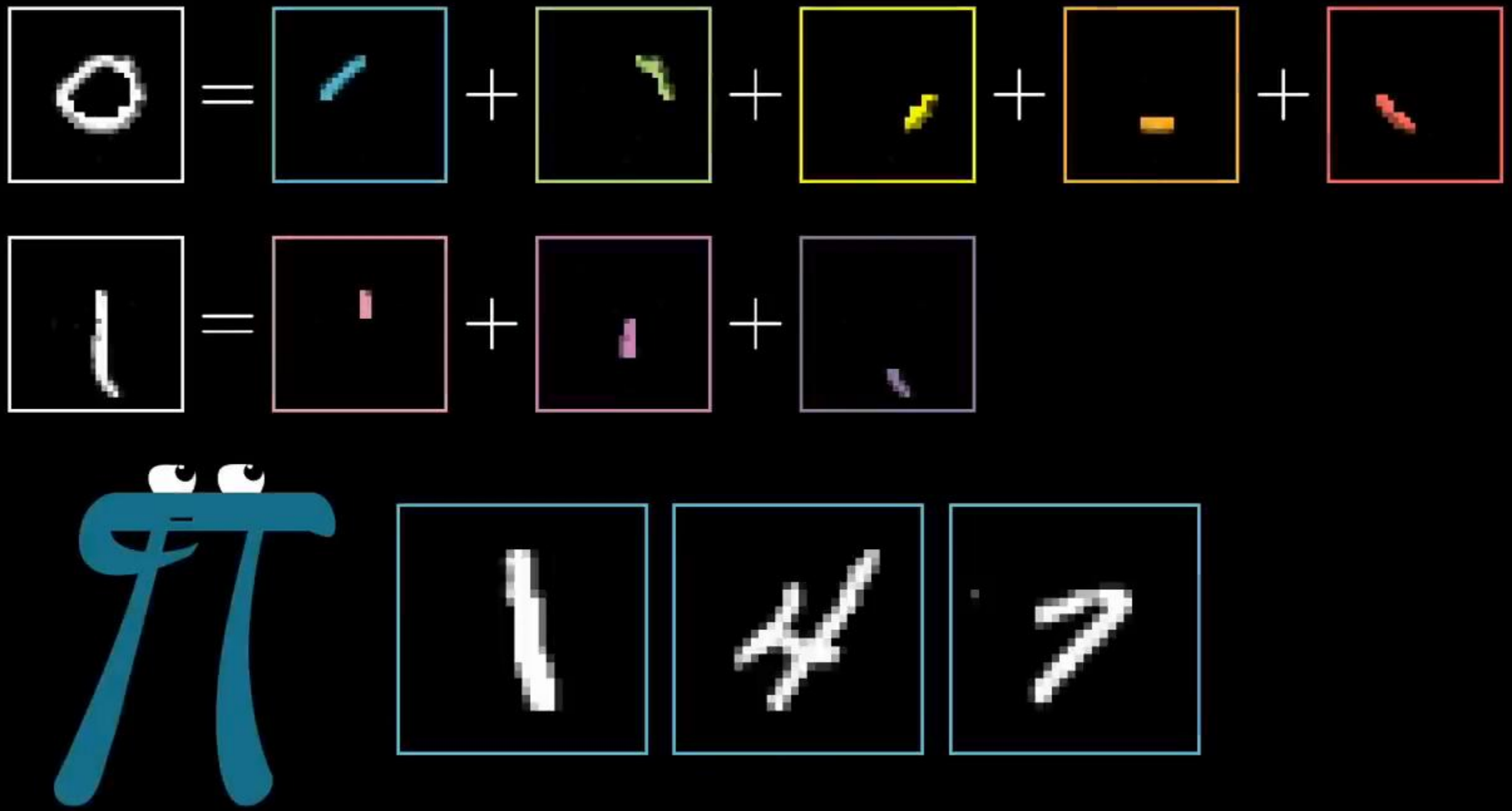
We can also add a **second (or more) layer** of nodes to the model, followed by an **output layer**.

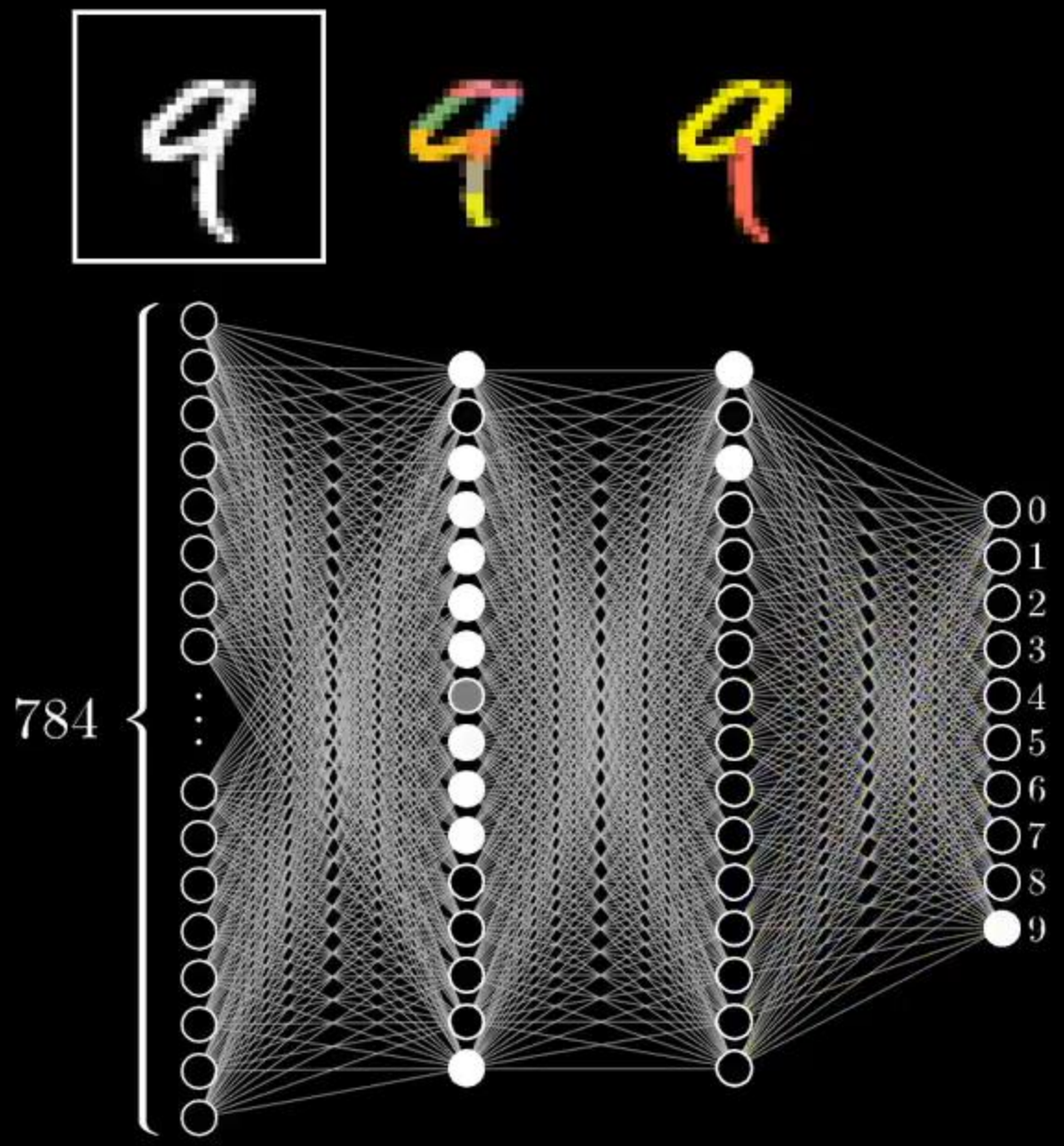

784
0
1
2
3
4
5
6
7
8
9


784
0
1
2
3
4
5
6
7
8
0.97
9

Why the layers?

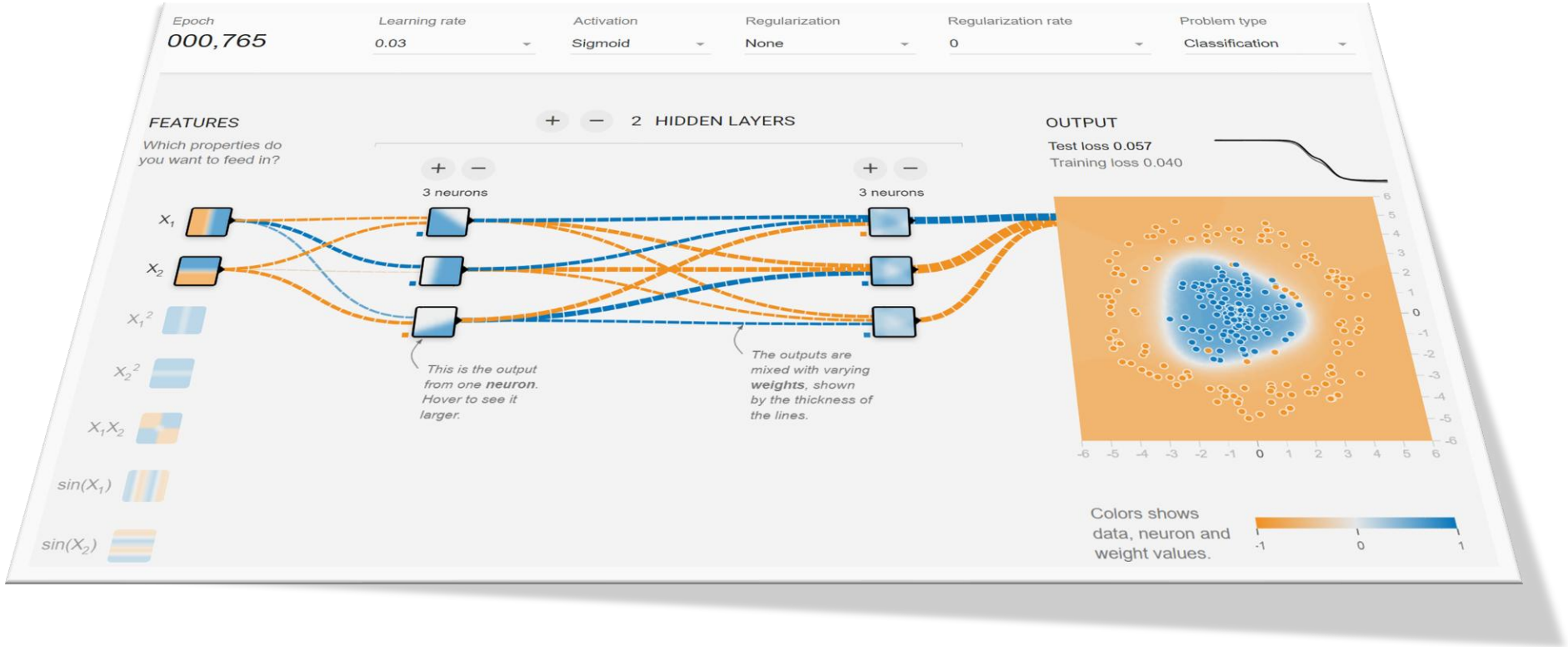





784
0
1
2
3
4
5
6
7
8
9

Neural network

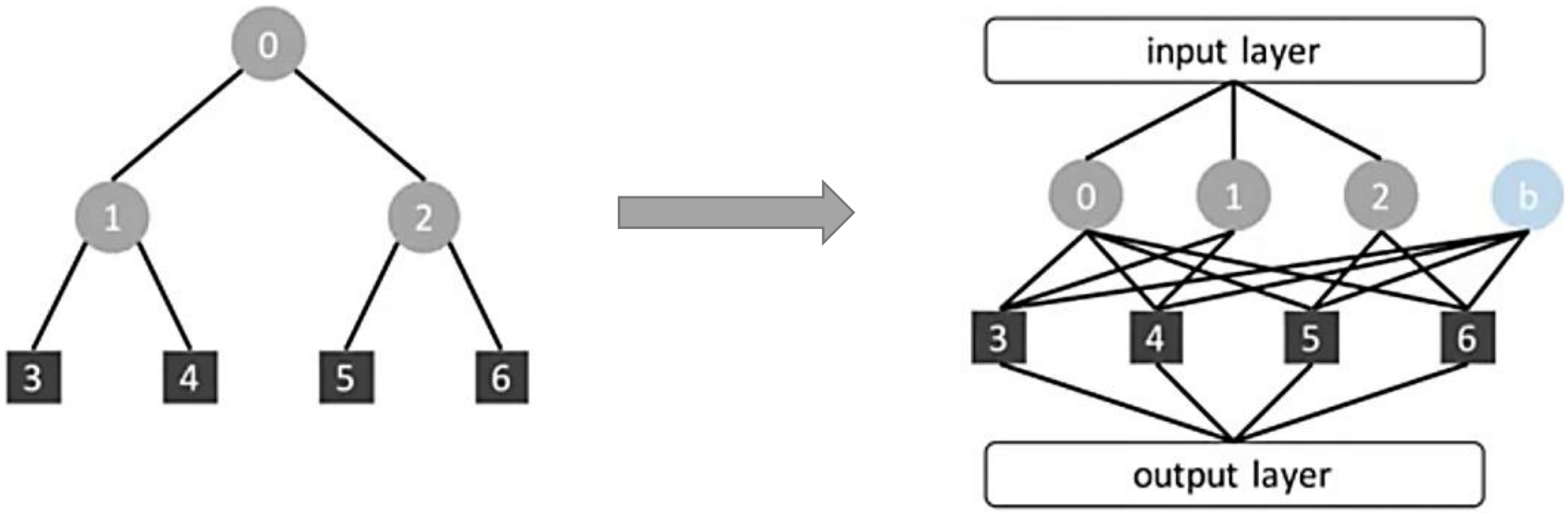
A neural network is an **architecture** that consists of logistic models (ctrl-click image):



We can also add a **second (or more) layer** of nodes to the model, followed by an **output layer**.

Deep neural network

A neural network is somewhat similar to a decision tree.



hidden layer 1

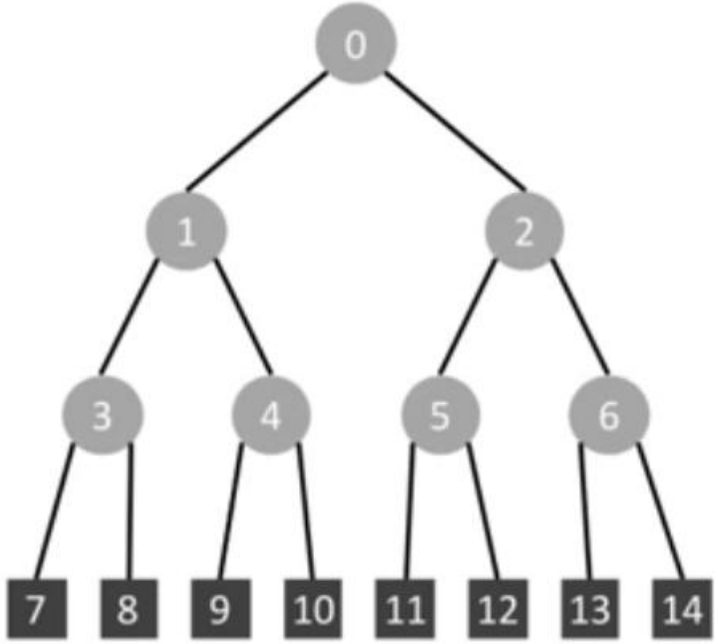
hidden layer 2

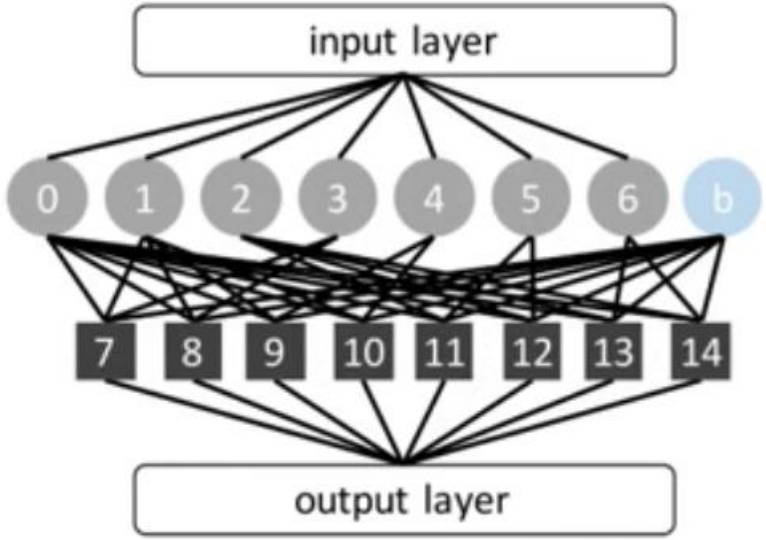
Sethi I.K., Proceedings of the IEEE (1990)

Deep neural network

A neural network is somewhat similar to a decision tree.

Except that adding a level in a decision tree is the same as expanding the first hidden layer in a neural network. This means that neural networks can learn much more complex functions by adding hidden layers.



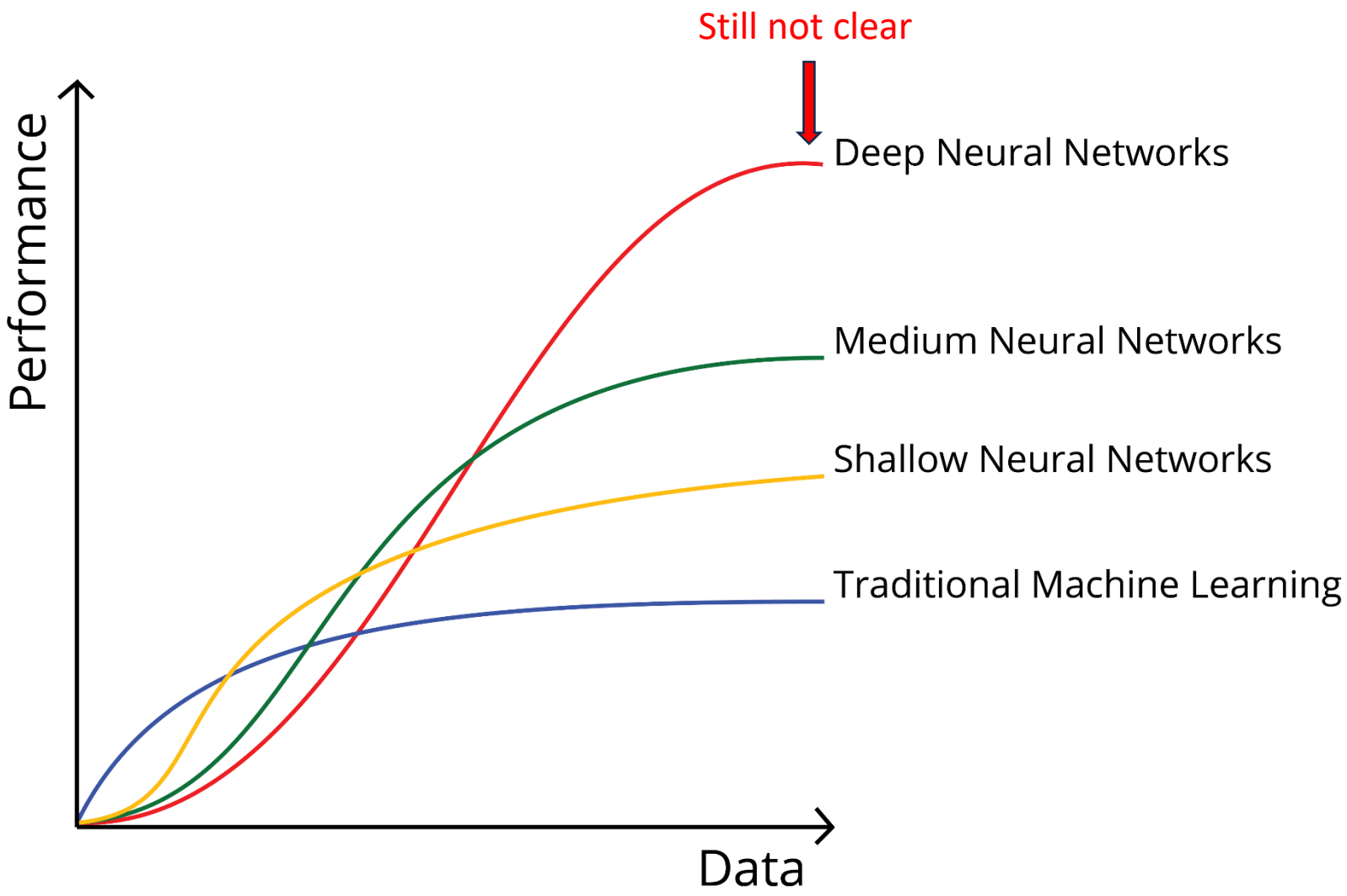


hidden layer 1

hidden layer 2

Sethi I.K., Proceedings of the IEEE (1990)

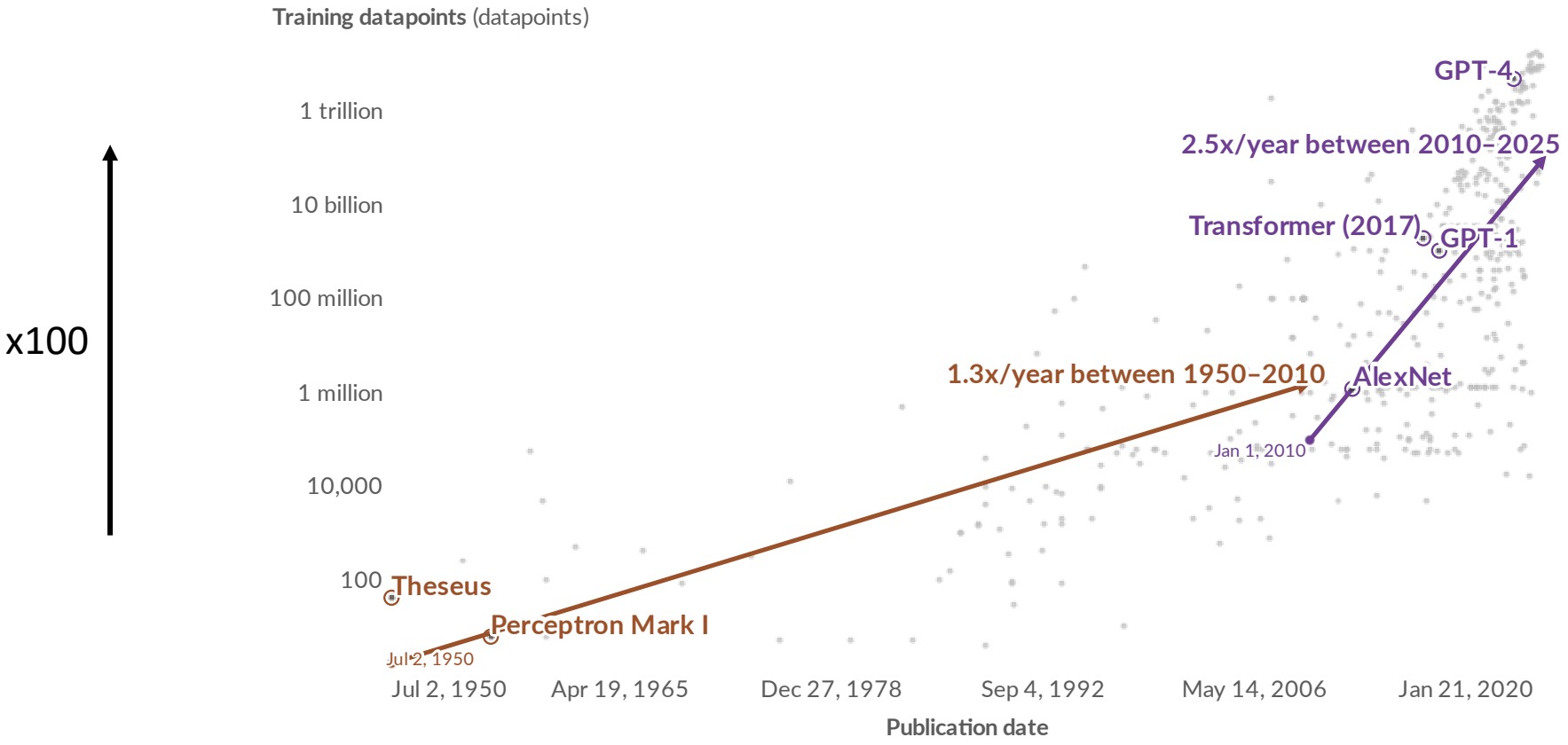
Deep neural network



Exponential growth of datapoints used to train notable AI systems

Our World in Data

Each domain has a specific data point unit; for example, for vision it is images, for language it is words, and for games it is timesteps. This means systems can only be compared directly within the same domain.



Data source: Epoch (2024)

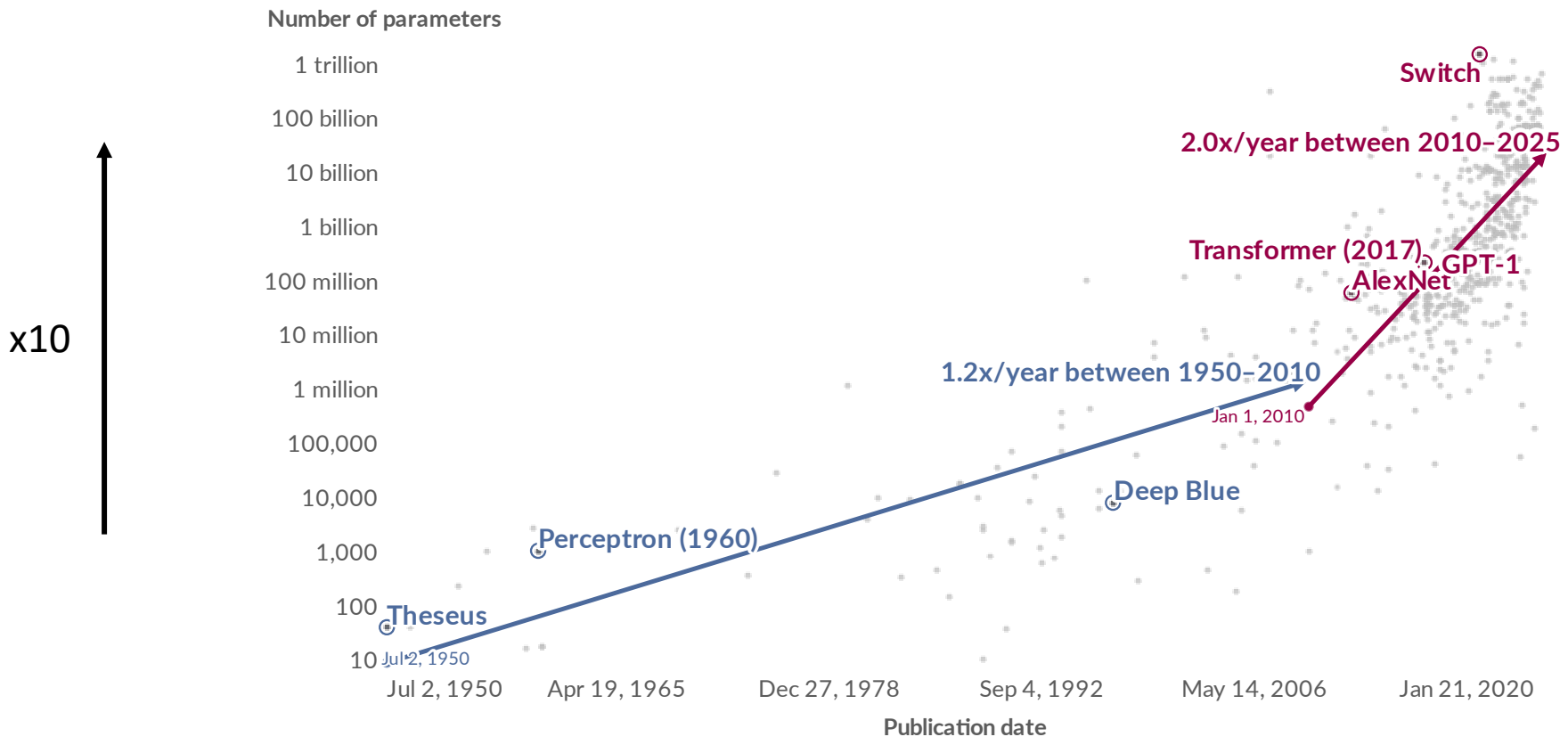
OurWorldinData.org/artificial-intelligence | CC BY

Note: The regression lines show a sharp rise in data used to train AI systems since 2010, driven by the success of deep learning methods that leverage neural networks and massive datasets.

Exponential growth of parameters in notable AI systems

Our World in Data

Parameters are variables in an AI system whose values are adjusted during training to establish how input data gets transformed into the desired output; for example, the connection weights in an artificial neural network.



Data source: Epoch (2024)

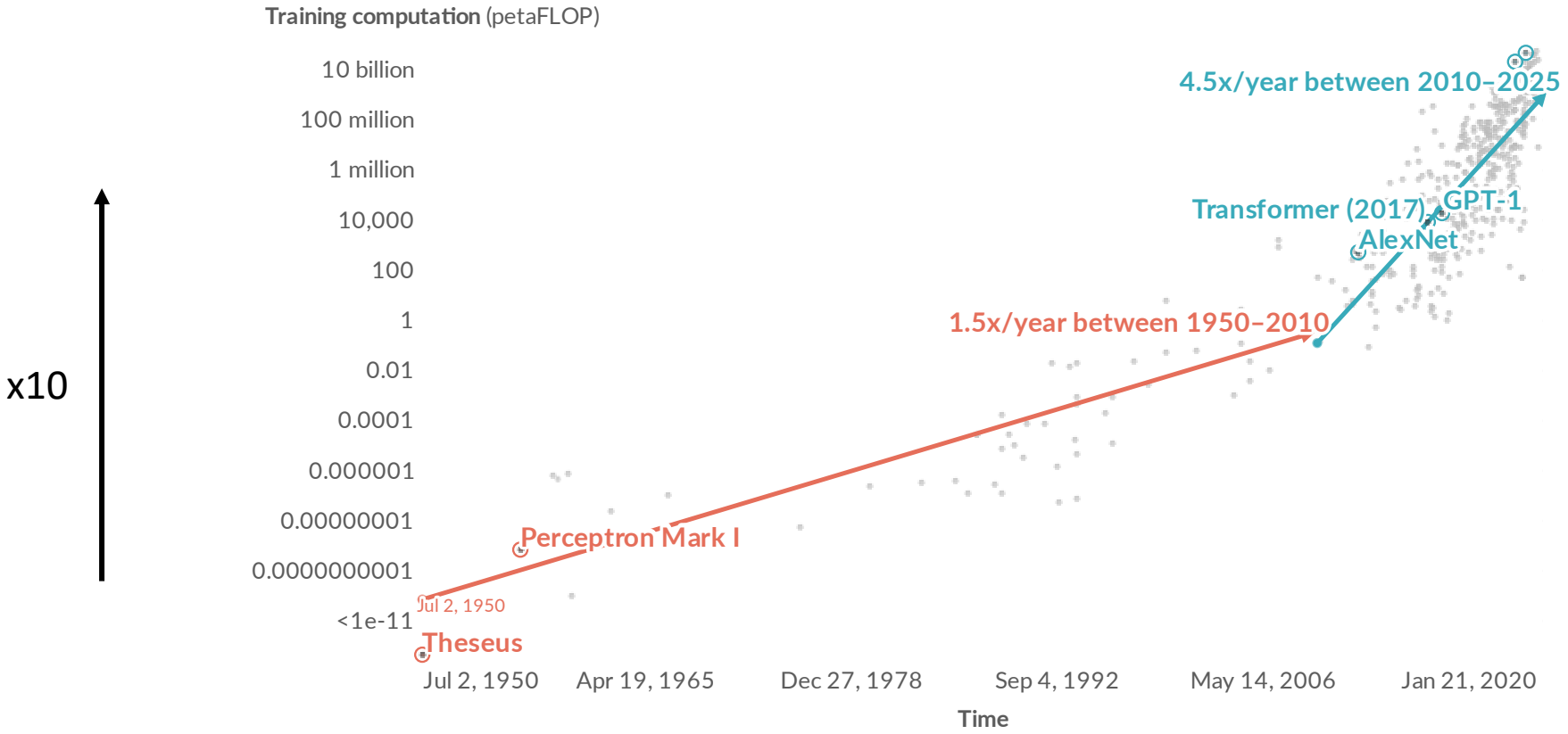
OurWorldinData.org/artificial-intelligence | CC BY

Note: Estimates are based on AI literature with uncertainty up to a factor of 10. The regression lines show a sharp rise in parameters since 2010, driven by the success of deep learning methods that leverage neural networks and massive datasets.

Exponential growth of computation in the training of notable AI systems

Our World in Data

Computation is measured in total petaFLOP, which is 10^{15} floating-point operations[1].



Data source: Epoch (2024)

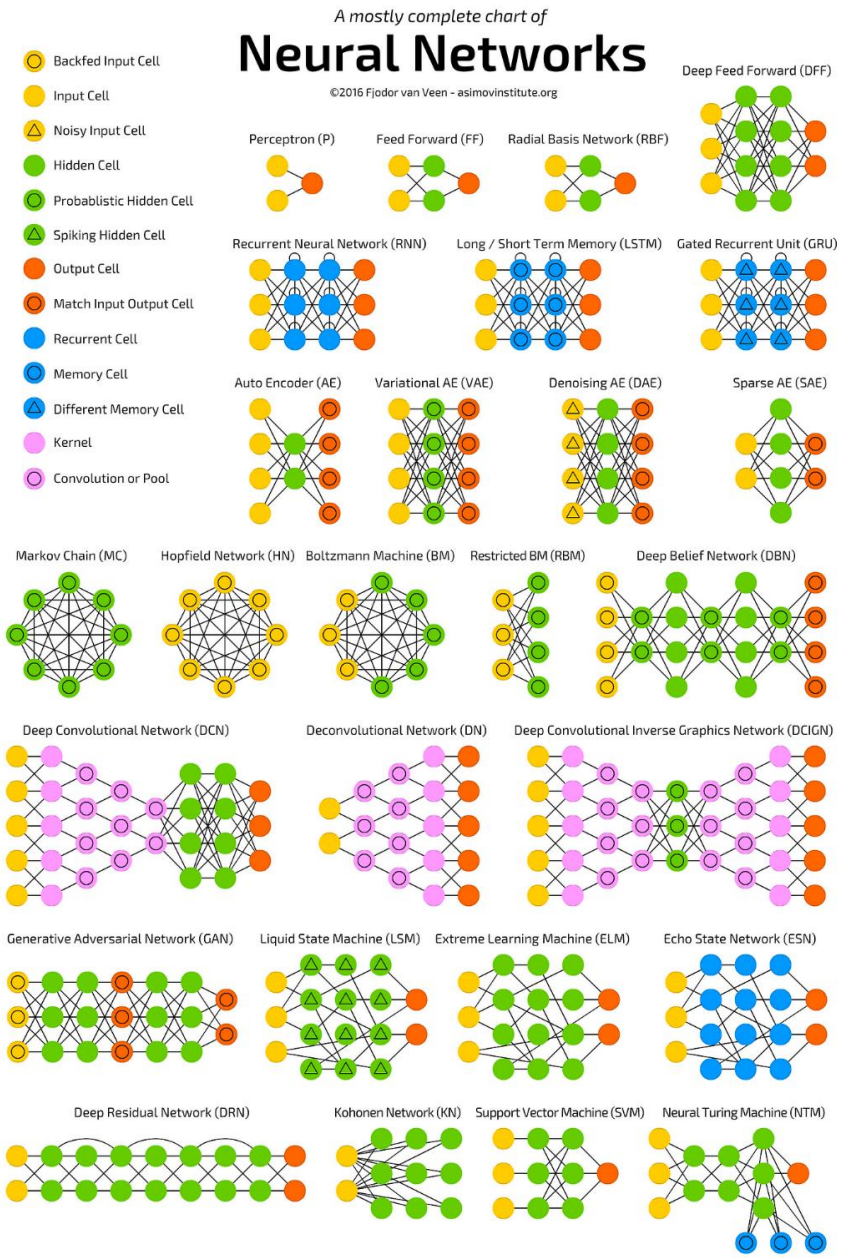
OurWorldinData.org/artificial-intelligence | CC BY

Note: Estimated from AI literature, accurate within a factor of 2, or 5 for recent models like GPT-4. The regression lines show a sharp rise in computation since 2010, driven by the success of deep learning methods that leverage neural networks and massive datasets.

1. Floating-point operation: A floating-point operation (FLOP) is a type of computer operation. One FLOP represents a single arithmetic operation involving floating-point numbers, such as addition, subtraction, multiplication, or division.

Network architecture

A neural network architecture is the structural design of an artificial neural network, defining the arrangement of layers, the number of neurons per layer, and the connections between them. It specifies how data flows through the network.



Fjodor van Veen from Asimov institute

Neural network training

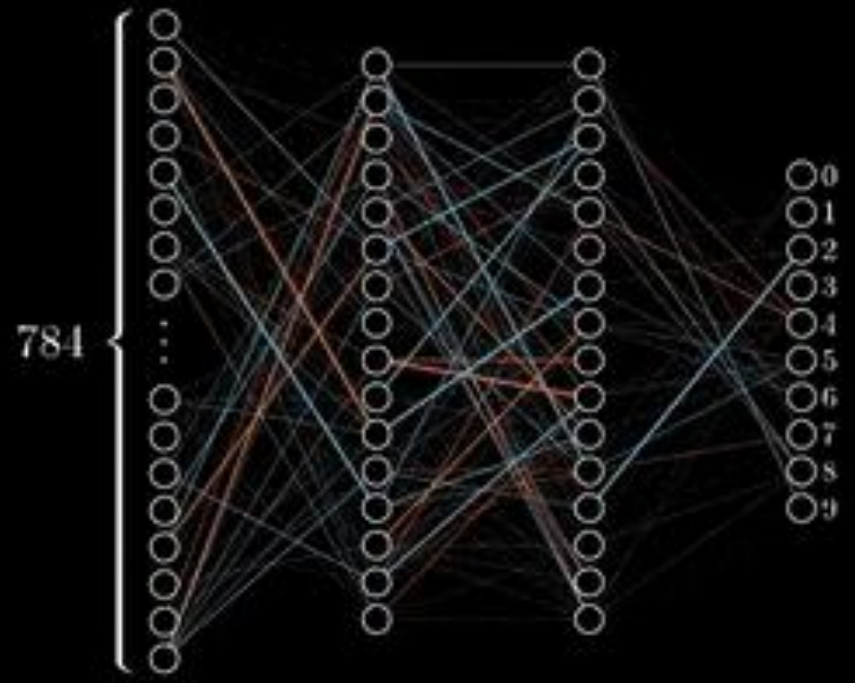
Forward propagation: Input data passes through the network layer by layer, where each neuron applies weights, biases, and an activation function to compute the output.

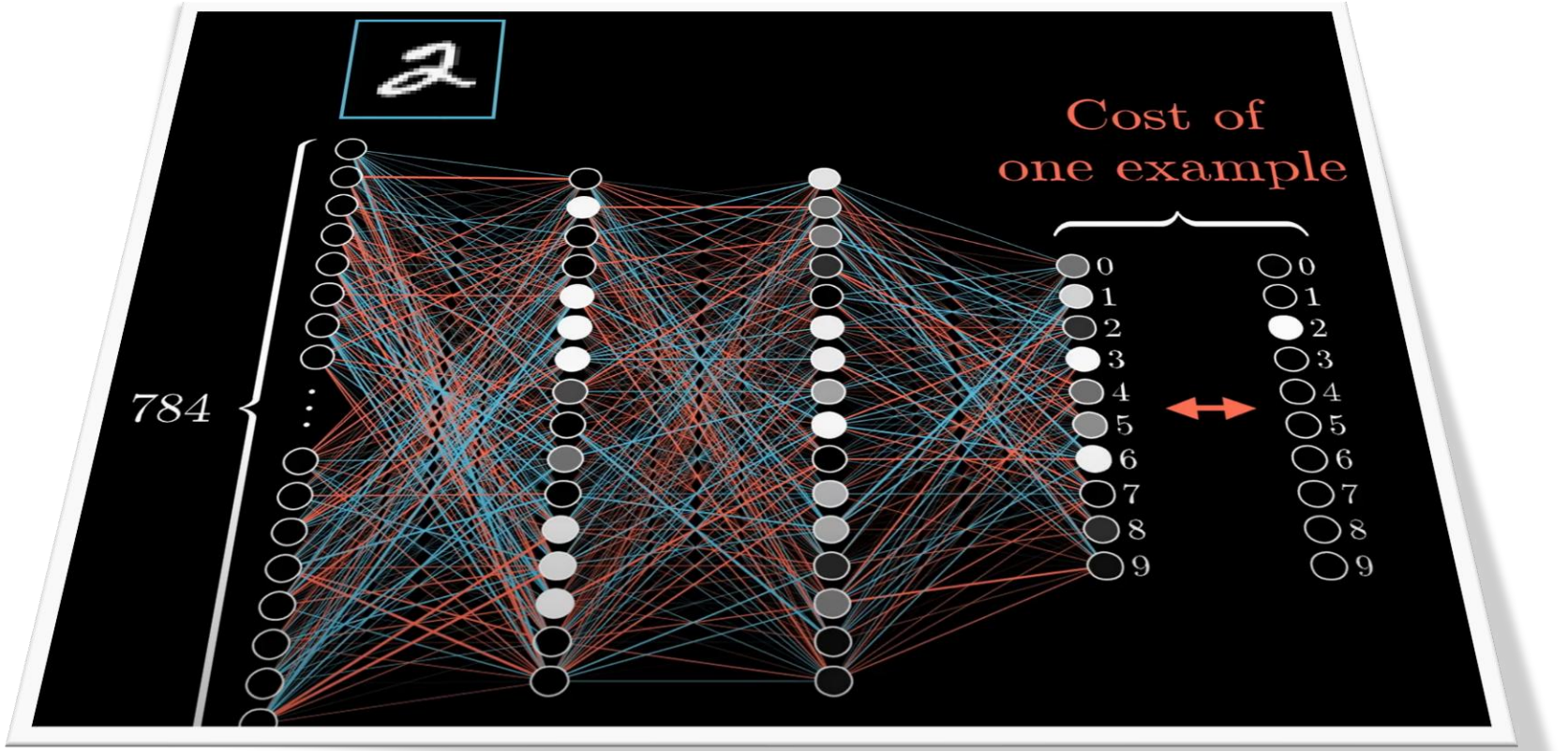
Loss calculation: The network compares the predicted output with the actual target using a loss function to measure error.

Backpropagation: The network computes gradients of the loss with respect to weights using the chain rule of differentiation, propagating the error backward through the layers.

Weight updates: The optimizer adjusts the weights and biases using the gradients to minimize the loss, gradually improving the network's accuracy over multiple iterations (**epochs**).

Training in progress. . .



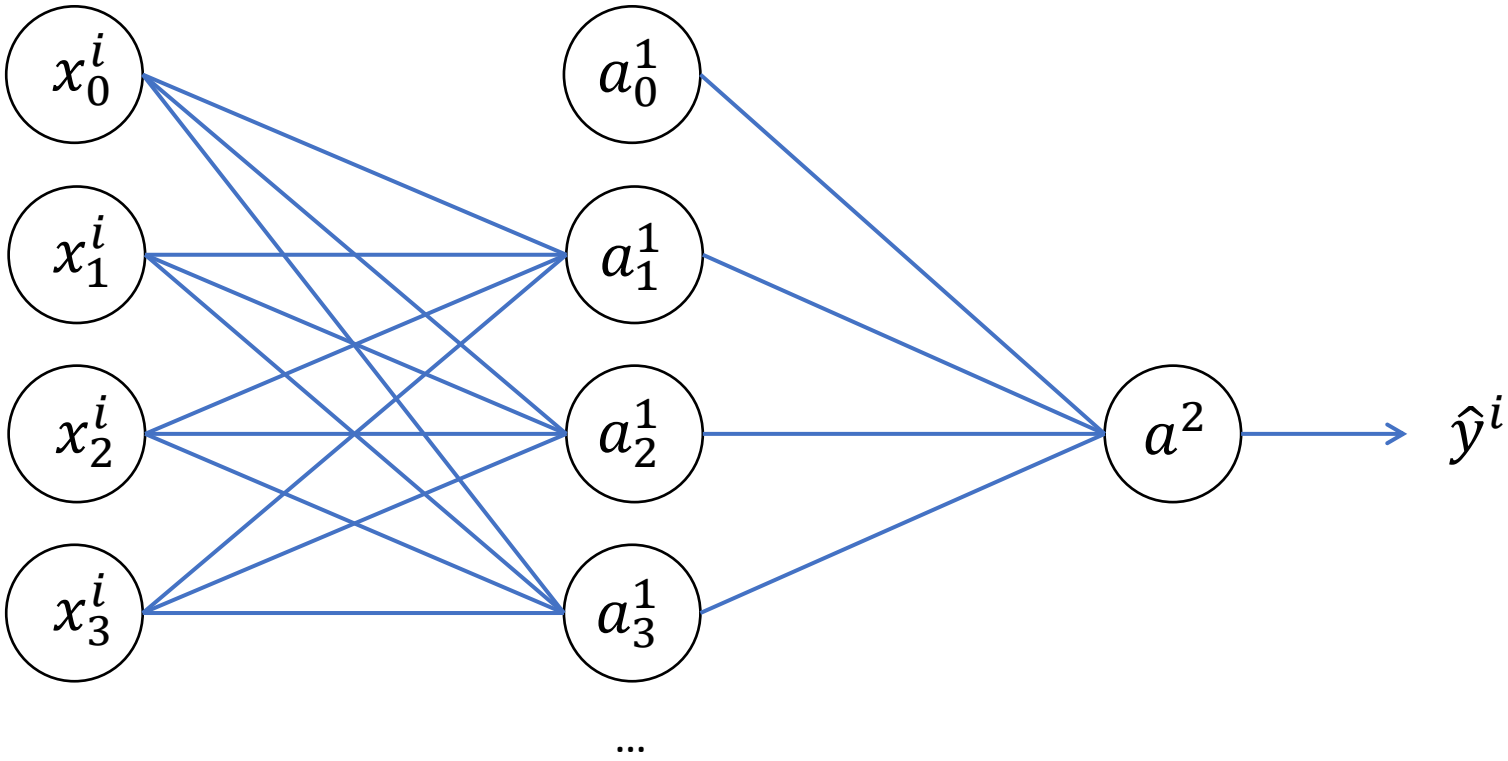


https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=548s&ab_channel=3Blue1Brown

Loss function

This function computes the “distance” between the output of the model and the expected output.

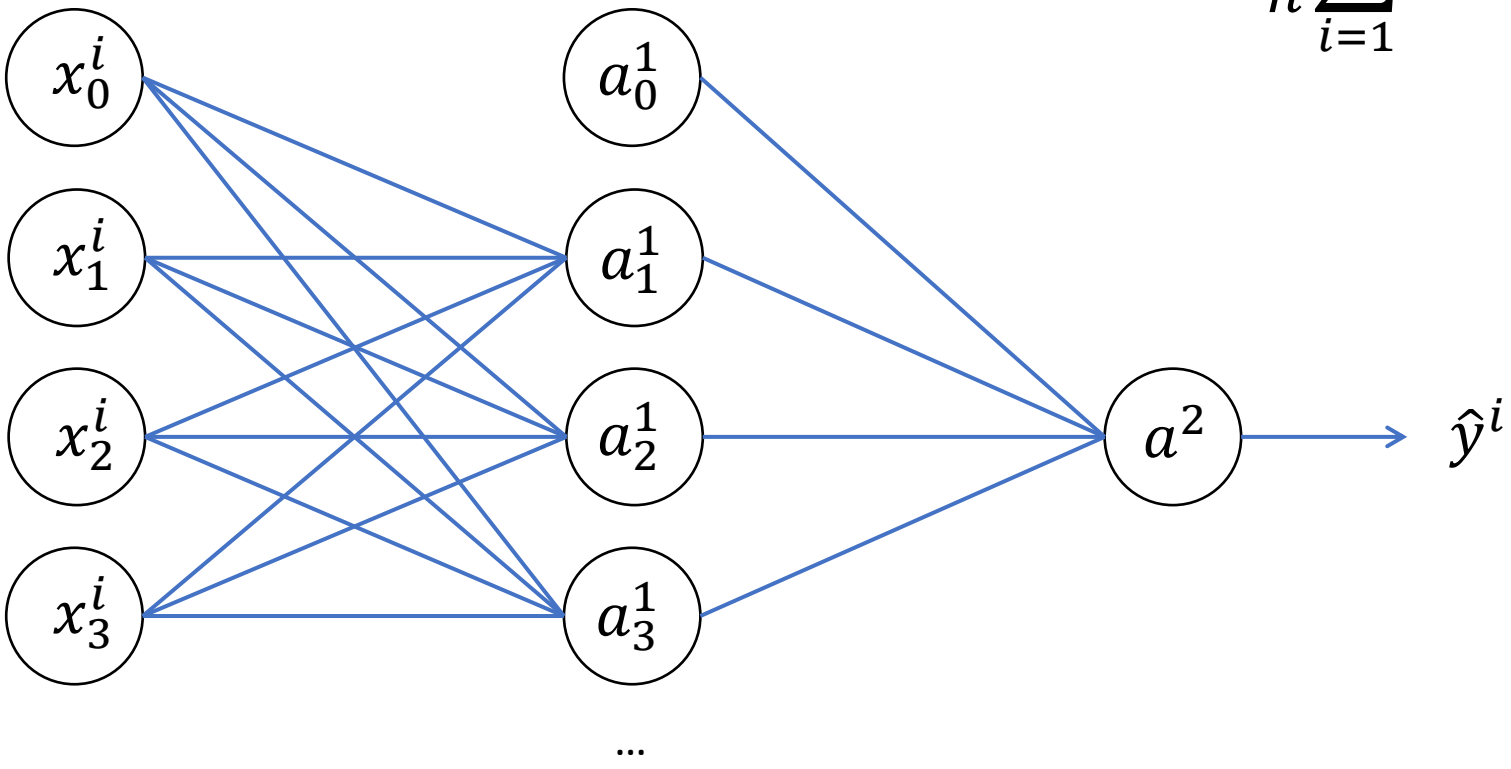
It defines the error the model makes.



Loss function

For a regression task we use the **Mean Squared Error (MSE)** also known as L2 loss.

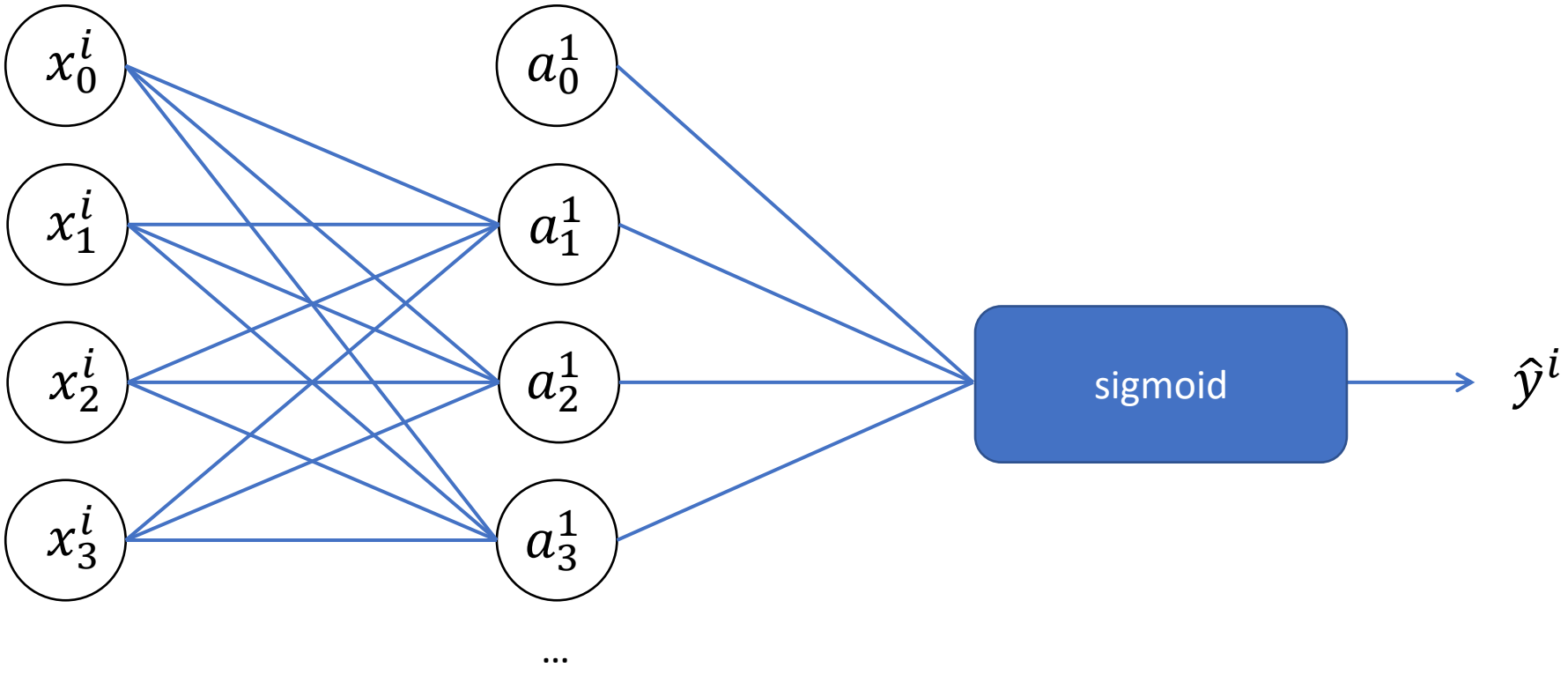
$$\frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$



Loss function

For a two-class (or binary) classification task we use the **Binary Cross Entropy (BCE)** loss.

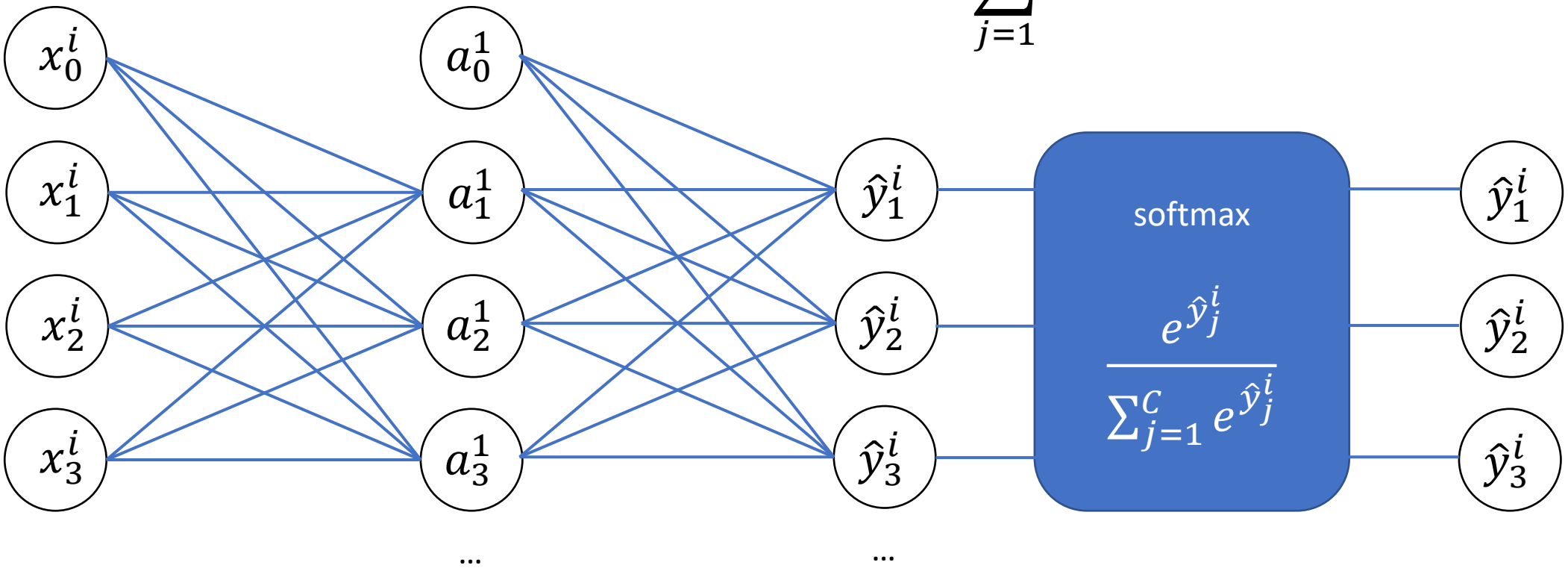
$$-\sum_{i=1}^{n}\left(y^i \log\left(\hat{y}^i\right) + \left(1 - y^i\right)\log\left(1 - \hat{y}^i\right)\right)$$



Loss function

For a multi-class classification task we use the **Categorical Cross Entropy (CE)** loss.

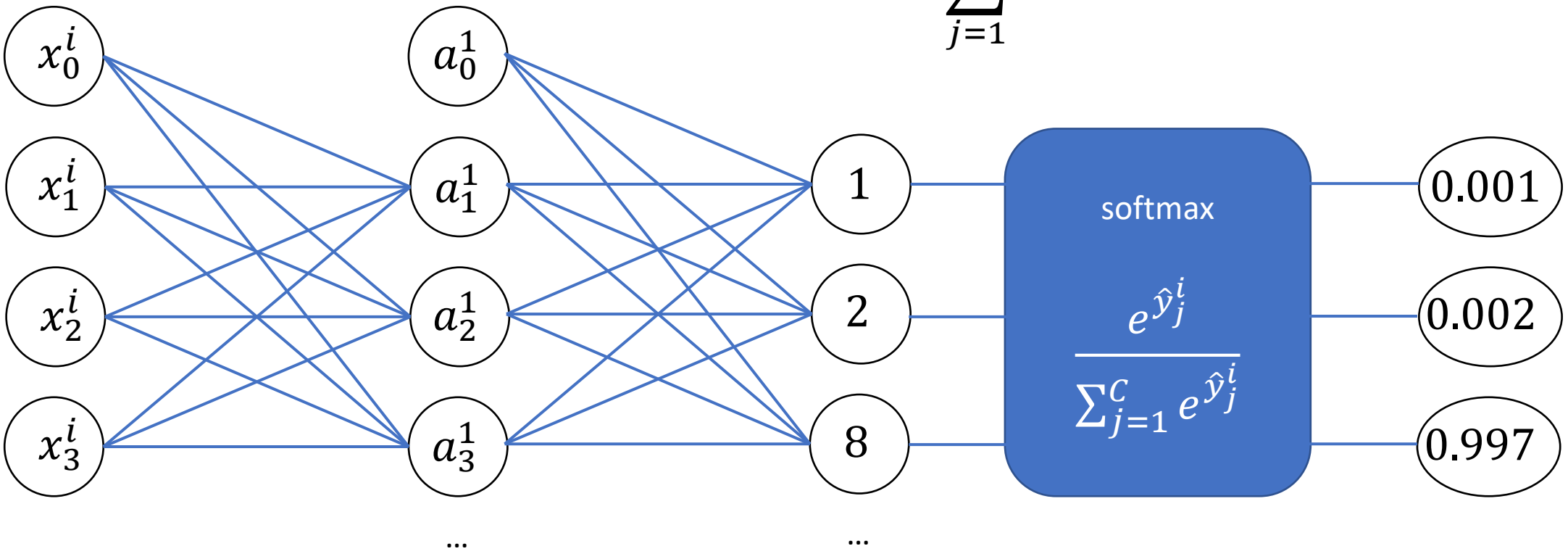
$$-\sum_{j=1}^{C}(y^i \log(\hat{y}^i))$$

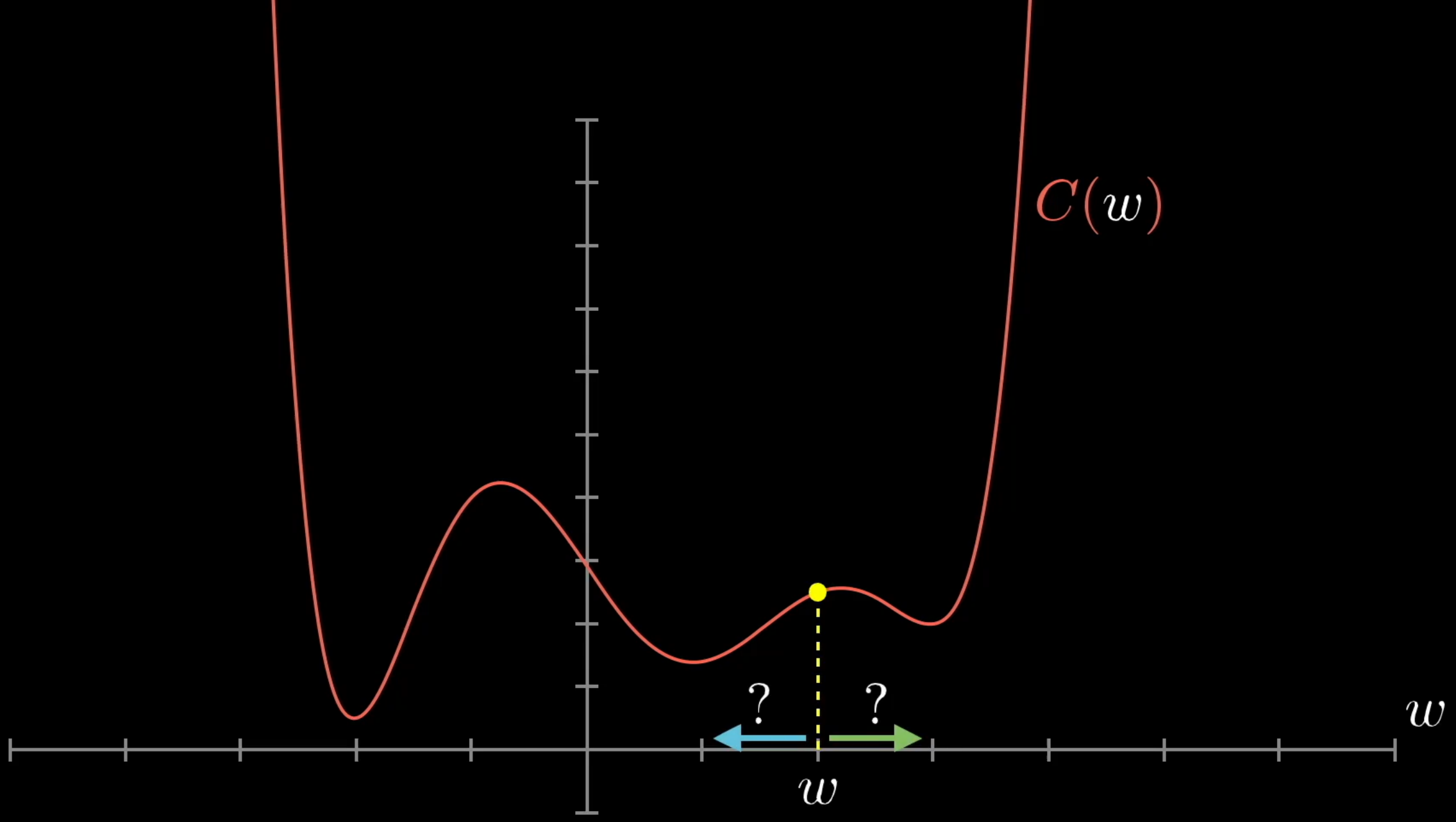


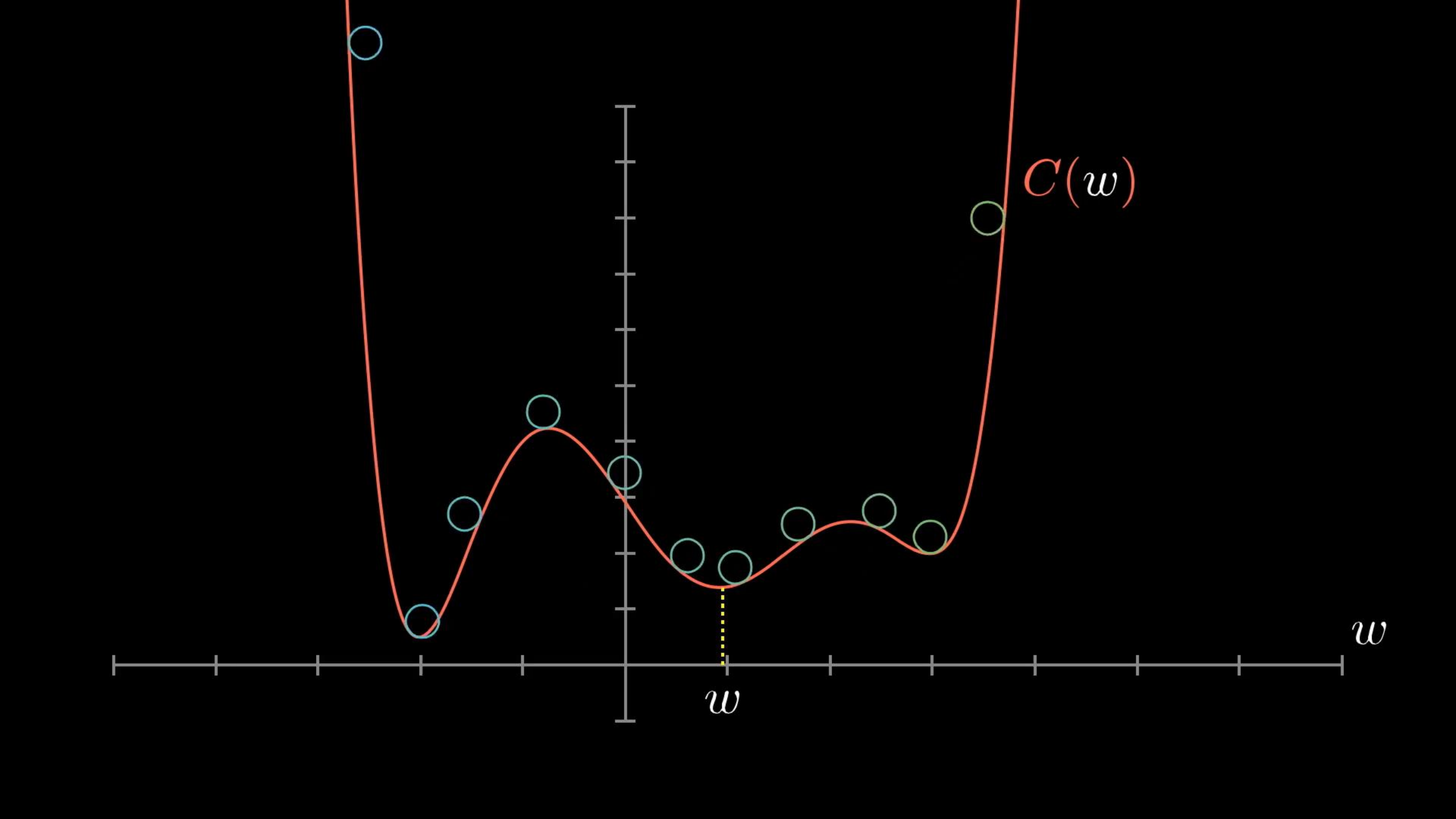
Loss function

For a multi-class classification task we use the **Categorical Cross Entropy (CE)** loss.

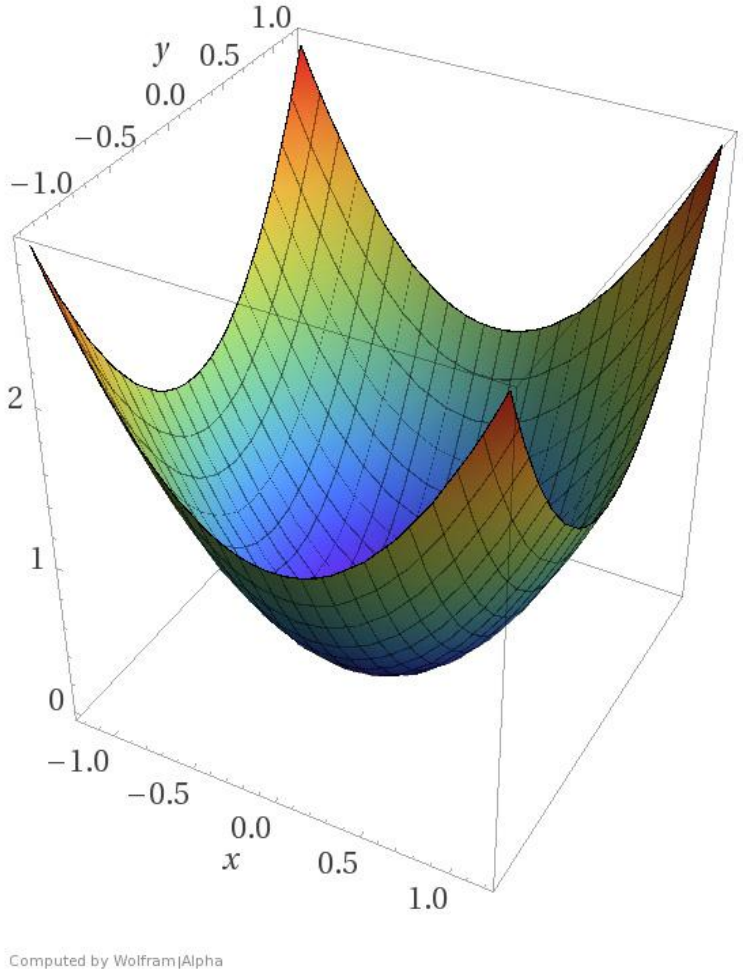
$$-\sum_{j=1}^{C}(y^i \log(\hat{y}^i))$$



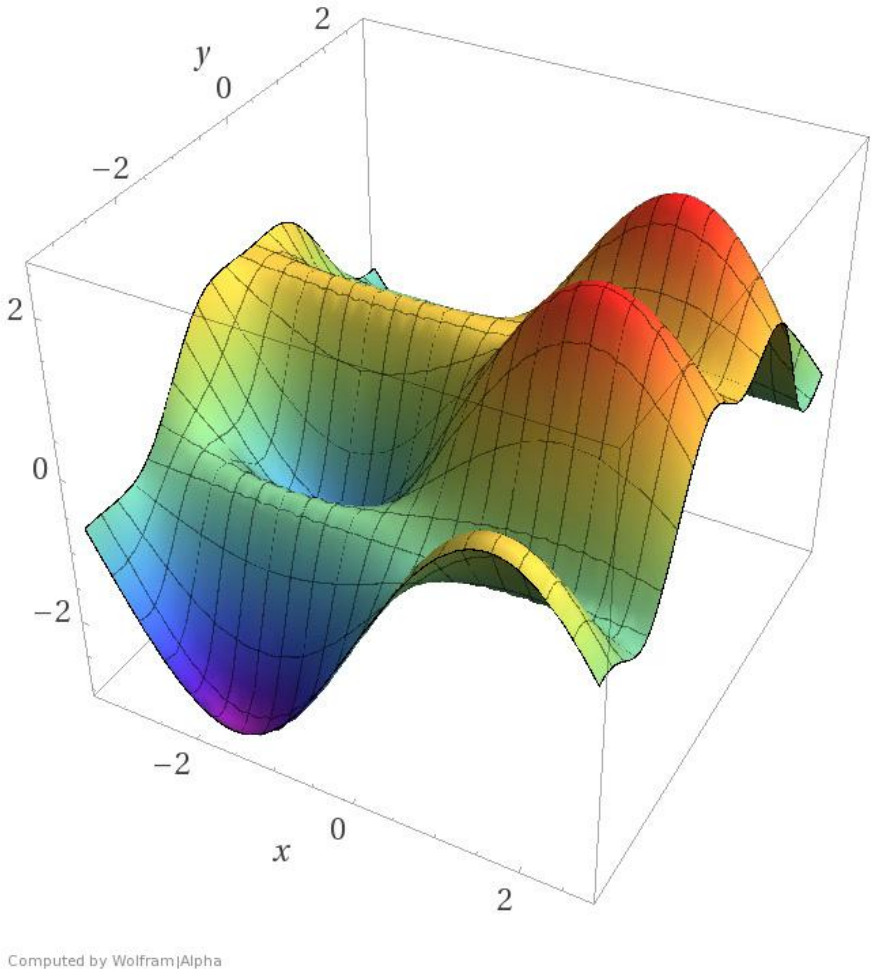

$C(w)$
?
?
w
w


$C(w)$
w
w

Loss function

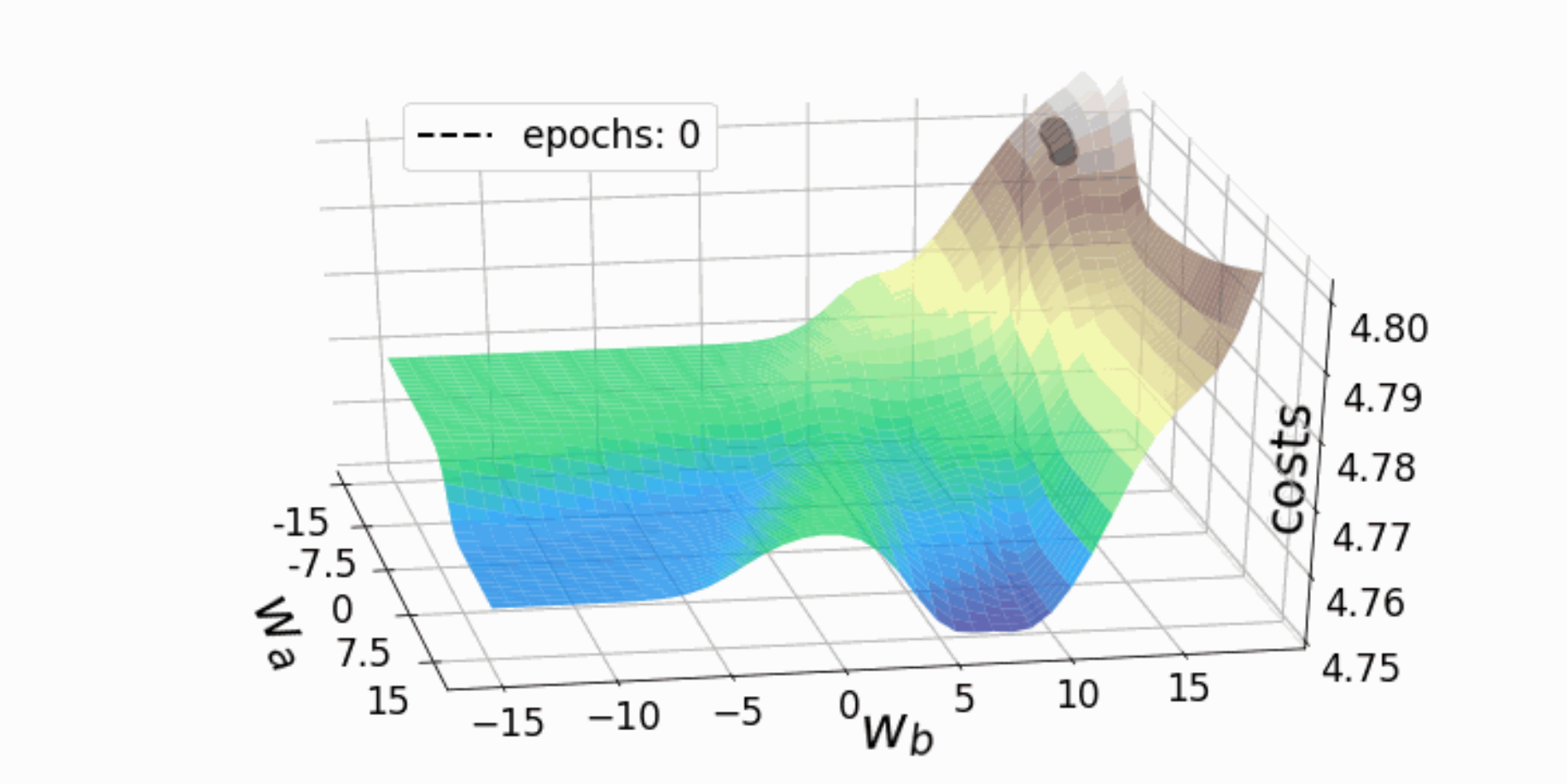


logistic regression



(small) neural network

Loss function

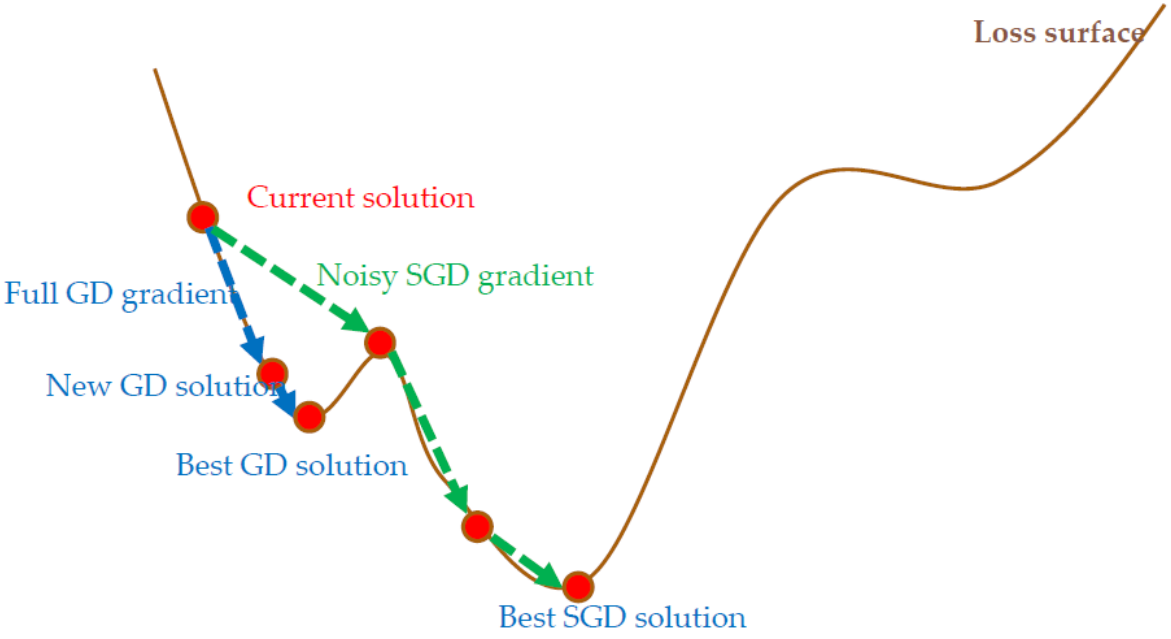


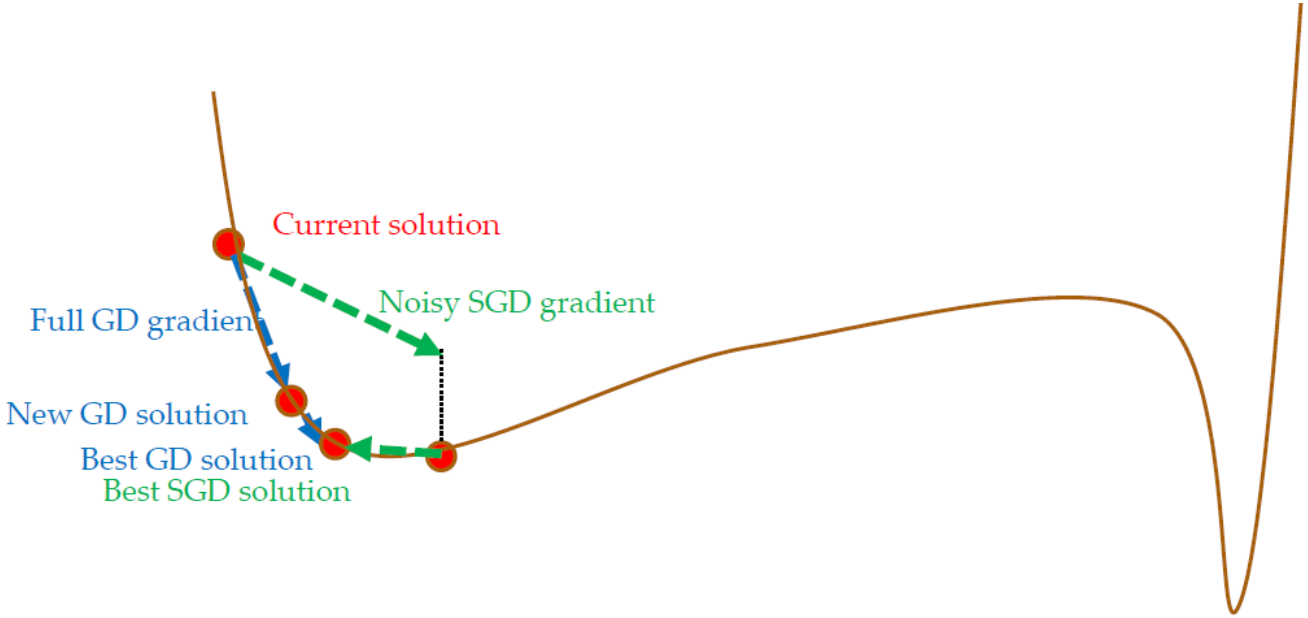
Stochastic gradient descent (SGD)

Gradient descent (GD) takes into consideration the complete training dataset through every iteration, which may become time-consuming and computationally expensive.

GD may converge to a local minimum and may not be able to get away from it.

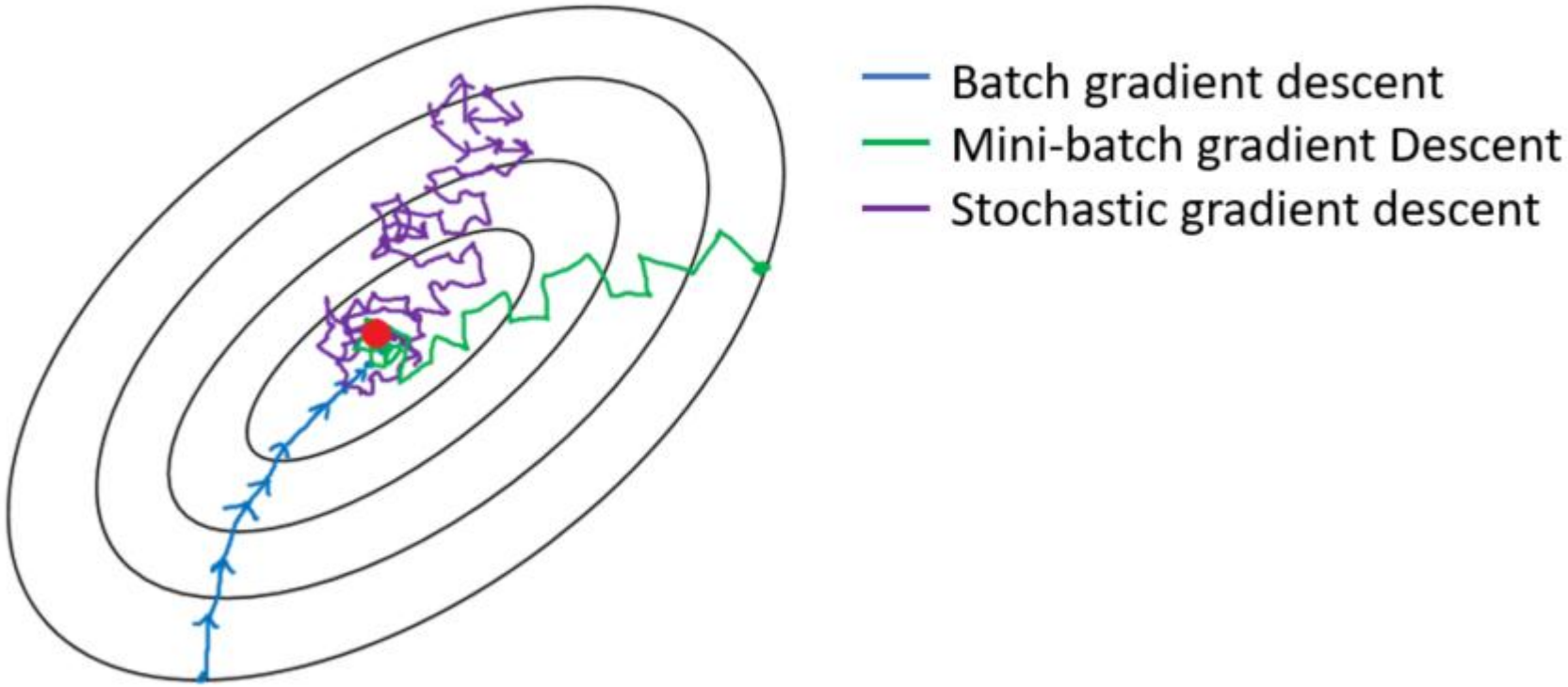
SGD takes into consideration one random data point through every iteration, which may lead to noisier results than the GD and may allow escaping local minima.





Mini batch gradient descent

Mini batch gradient descent is a cross-over between GD and SGD. Instead of iterating through the entire dataset (GD) or one observation (SGD), we split the dataset into small subsets (**mini batches**) and compute the gradients for each batch.

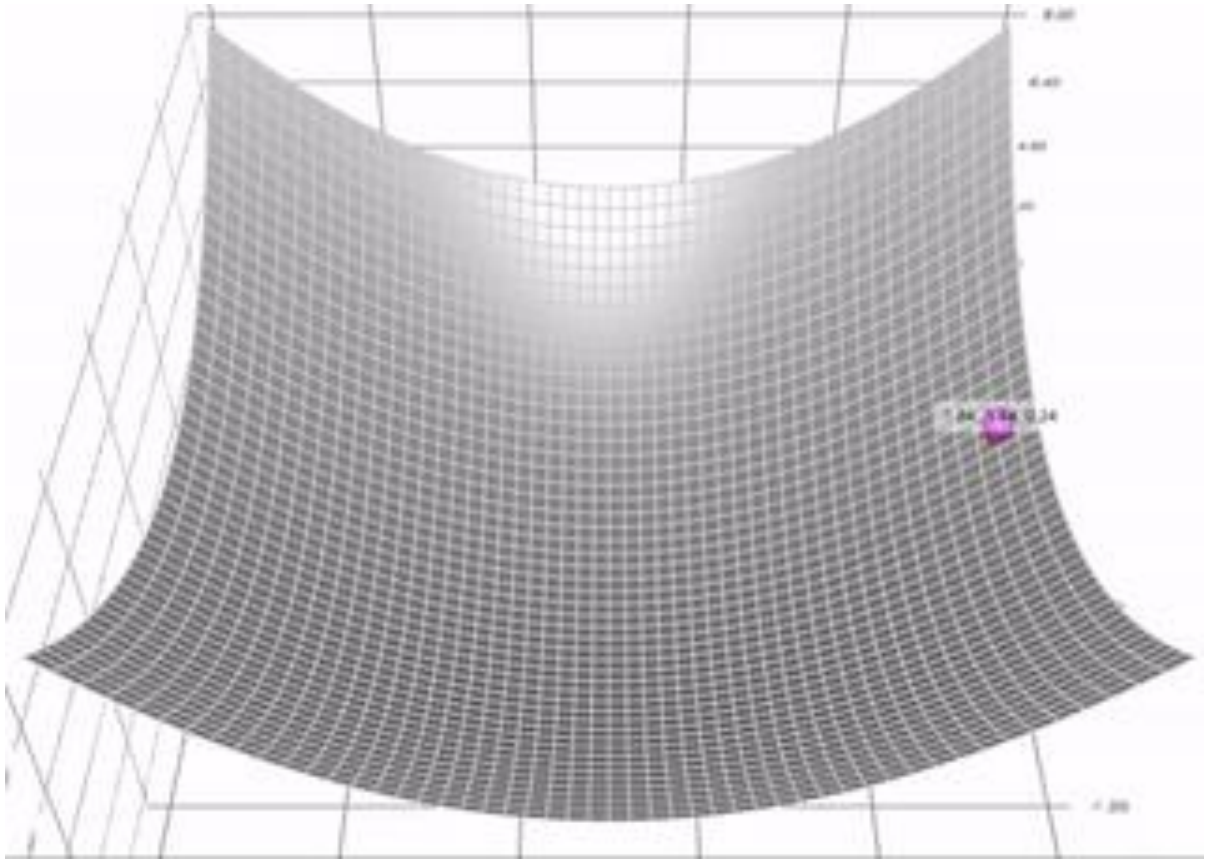


Momentum in SGD

Instead of updating parameters solely based on the current gradient, **momentum** accumulates past gradients, allowing the optimizer to maintain direction and overcome local minima more efficiently.

Momentum introduces a **velocity** vector, which is an **exponentially weighted moving average** of past gradients, helping to smooth updates and accelerate training in consistent directions.

Imagine rolling down a ball inside of a frictionless bowl. Instead of stopping at the bottom, the momentum it has accumulated pushes it forward, and the ball keeps rolling back and forth.

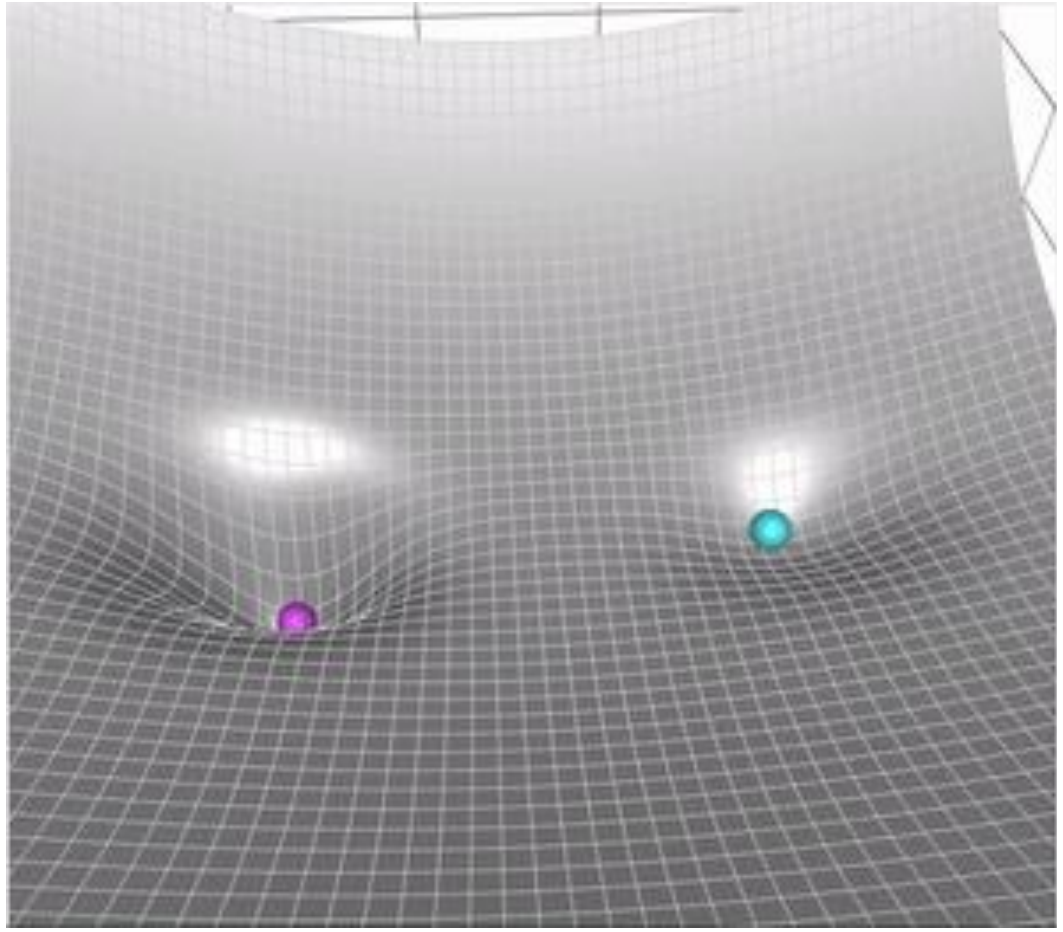


Momentum in SGD

Instead of updating parameters solely based on the current gradient, **momentum** accumulates past gradients, allowing the optimizer to maintain direction and overcome local minima more efficiently.

Momentum introduces a **velocity** vector, which is an **exponentially weighted moving average** of past gradients, helping to smooth updates and accelerate training in consistent directions.

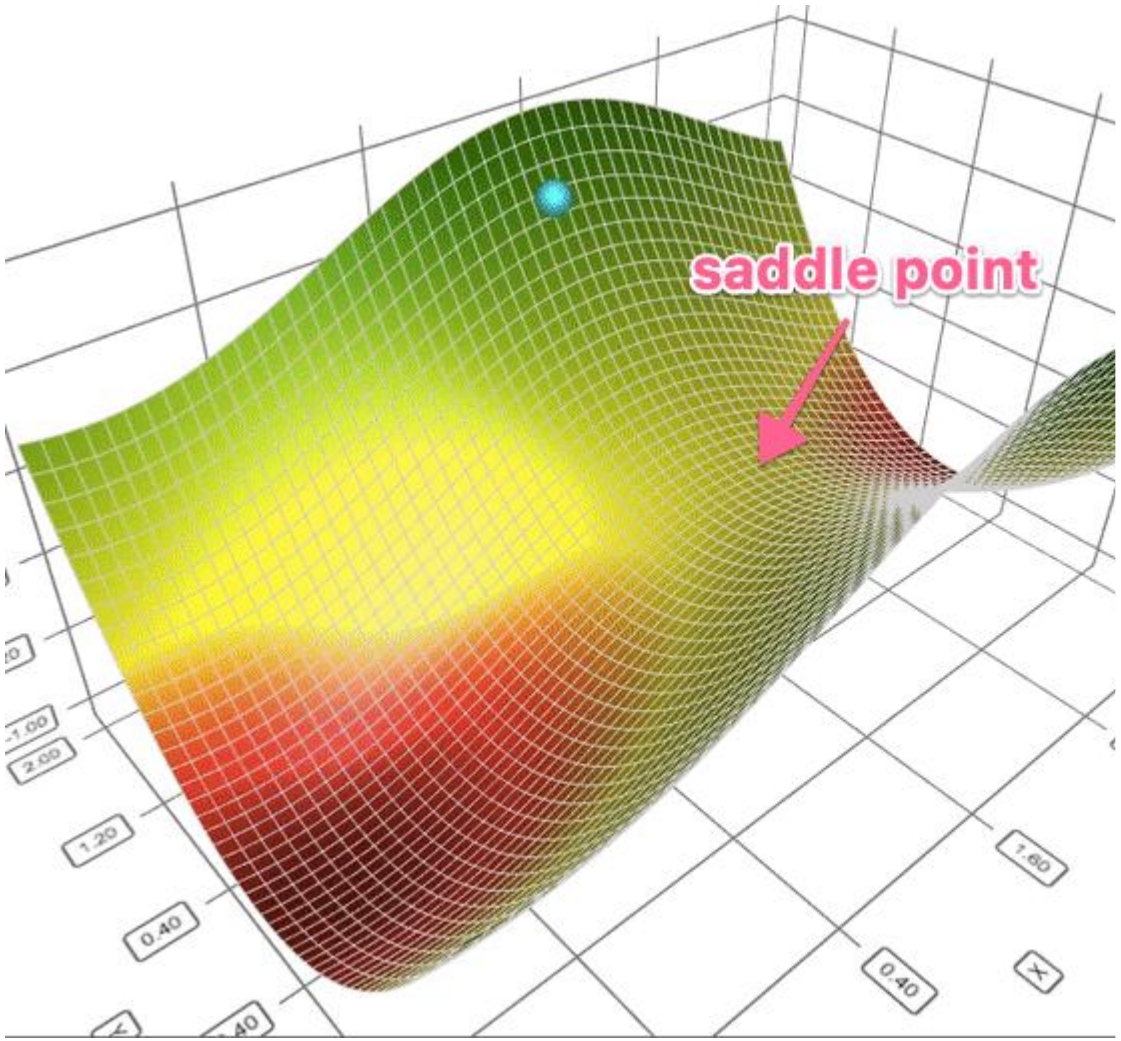
Imagine rolling down a ball inside of a frictionless bowl. Instead of stopping at the bottom, the momentum it has accumulated pushes it forward, and the ball keeps rolling back and forth.



Adaptive Learning Rates

AdaGrad adjusts the learning rate **individually for each feature**, reducing it over time based on the cumulative sum of past squared gradients. This helps features with smaller gradients (e.g., sparse features) receive relatively larger updates.

Adam optimizer blends **momentum (to smooth updates)** and **adaptive learning rates (to scale updates per parameter)**, making it well-suited for noisy gradients and sparse features while maintaining efficient convergence.

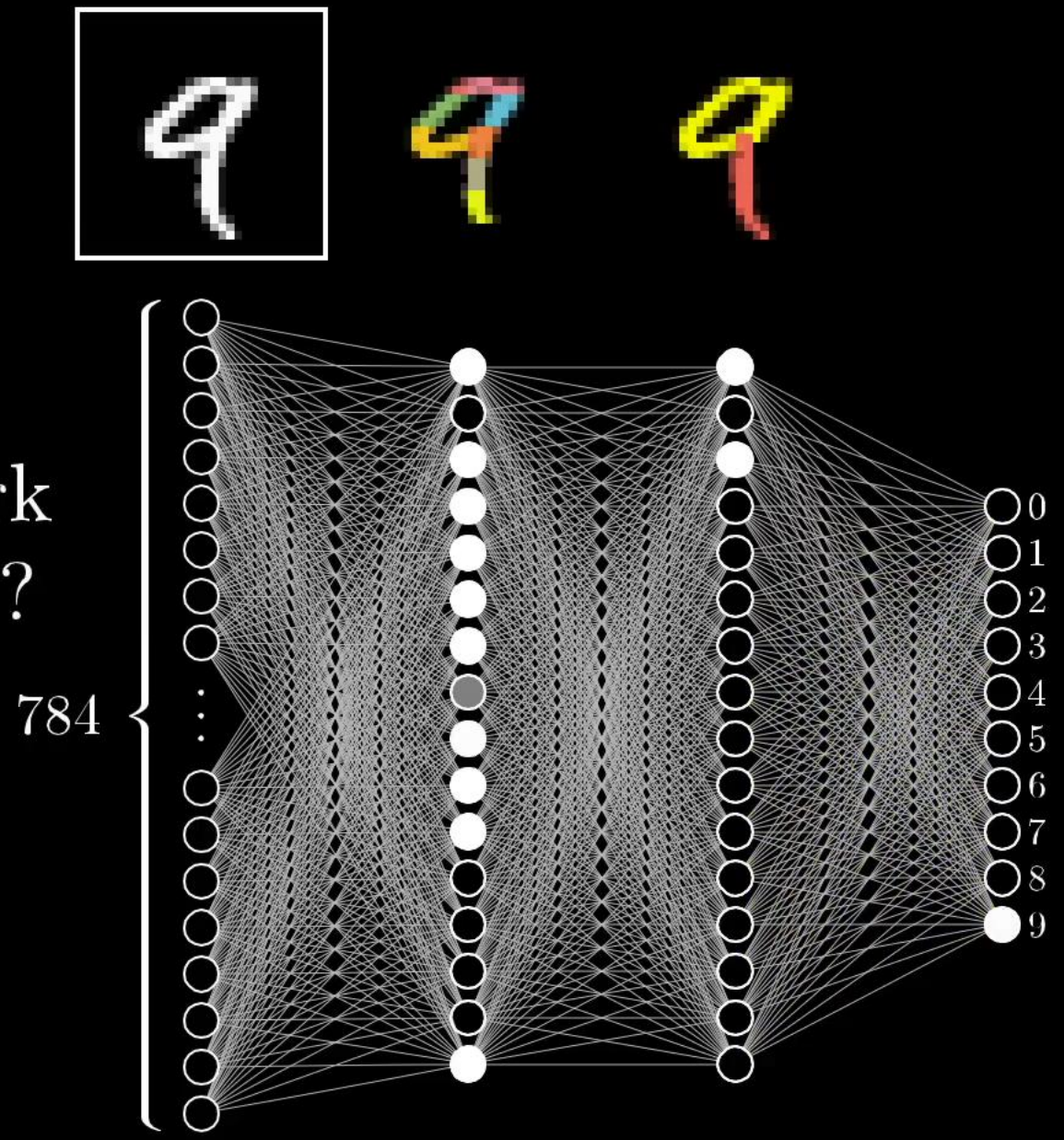


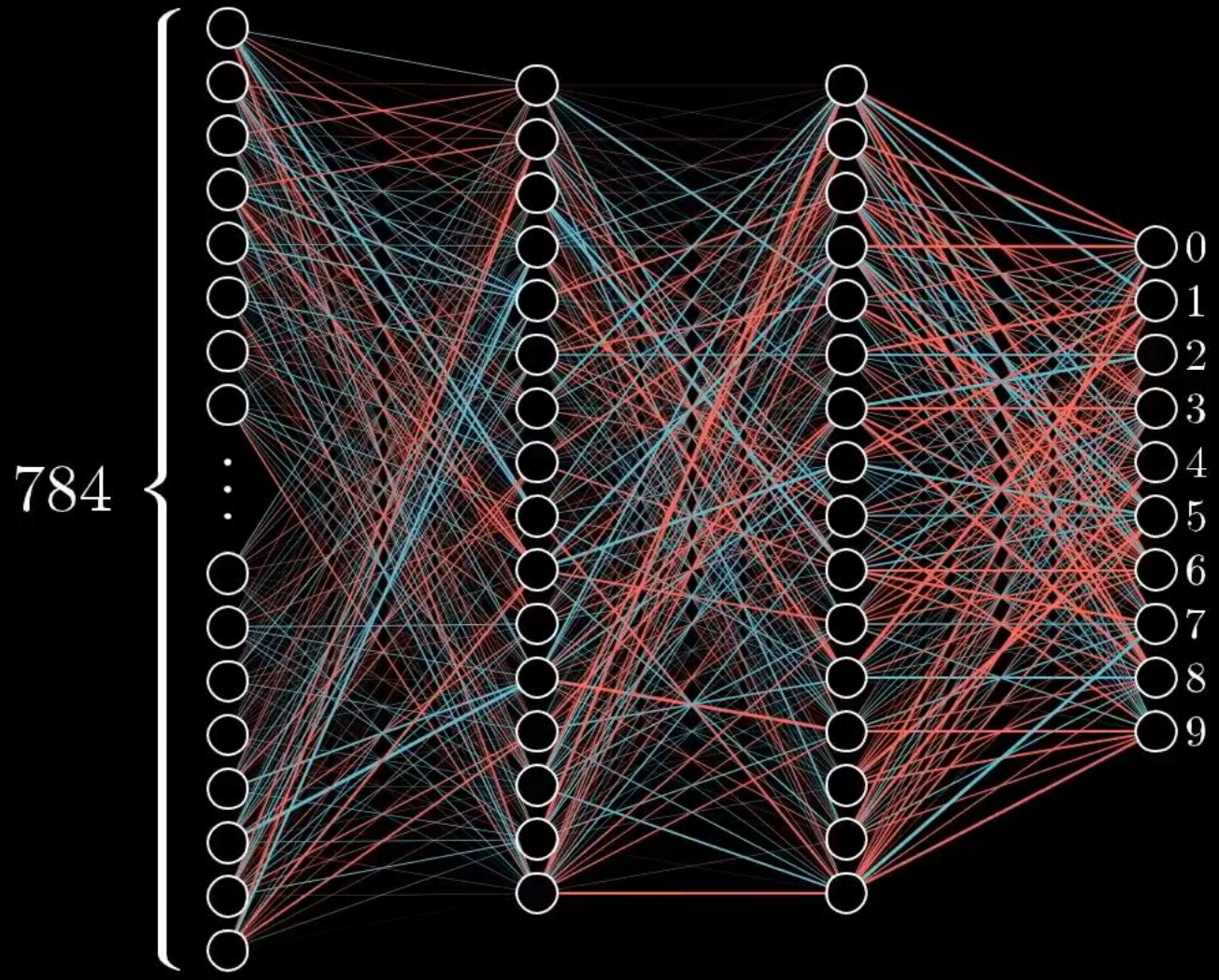
Neural network

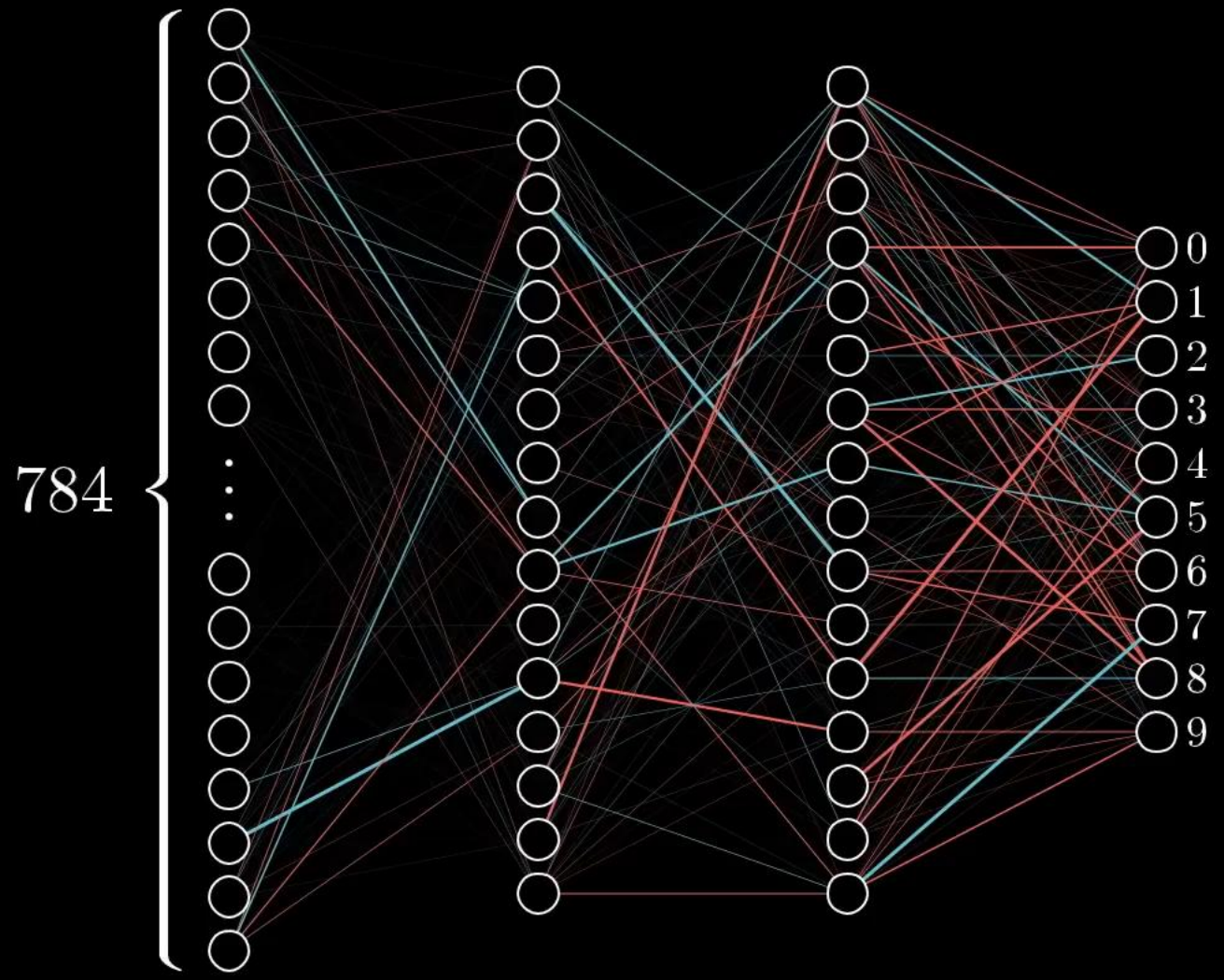
GitHub

mnist_pytorch.ipynb

Does the network actually do this?




784
0
1
2
3
4
5
6
7
8
9


784
0
1
2
3
4
5
6
7
8
9