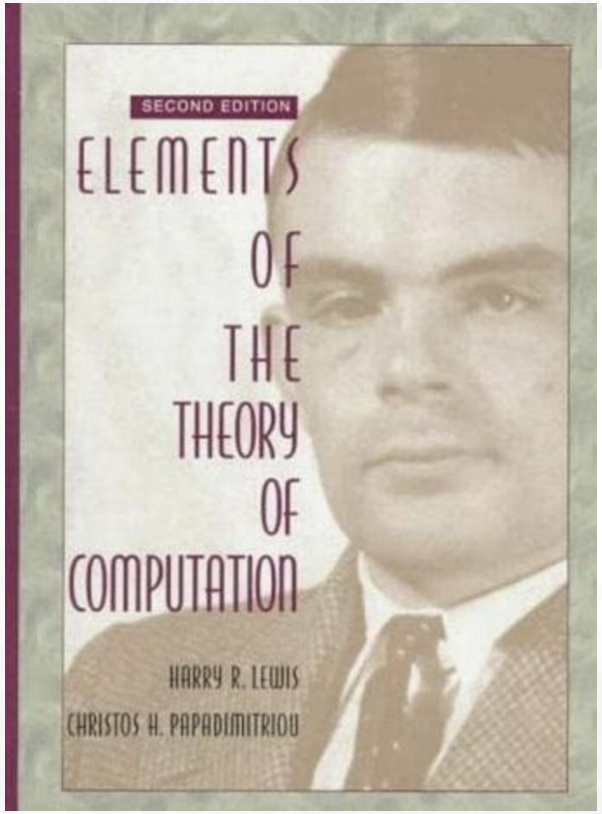
15. Deep Learning

Artificial Intelligence

- Alan Turing
  - Turing test, a method to assess a machine's ability to exhibit human-like intelligent behavior
- Artificial intelligence, or AI
  - technology that enables computers and machines to simulate human intelligence and problem-solving capabilities.

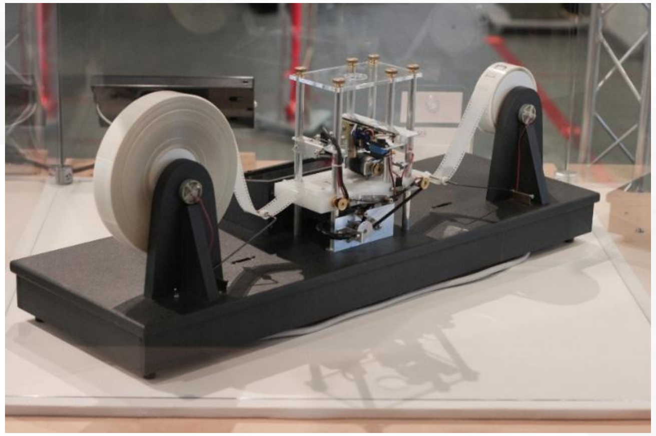


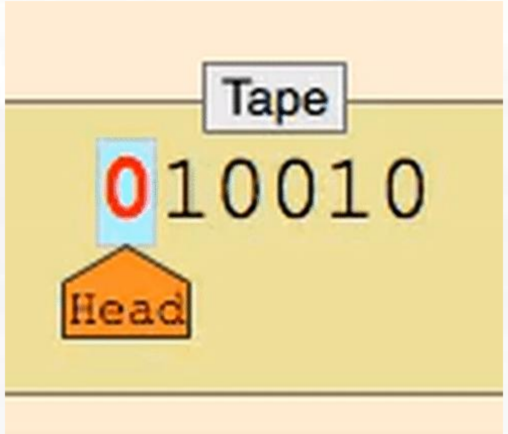
Turing Test

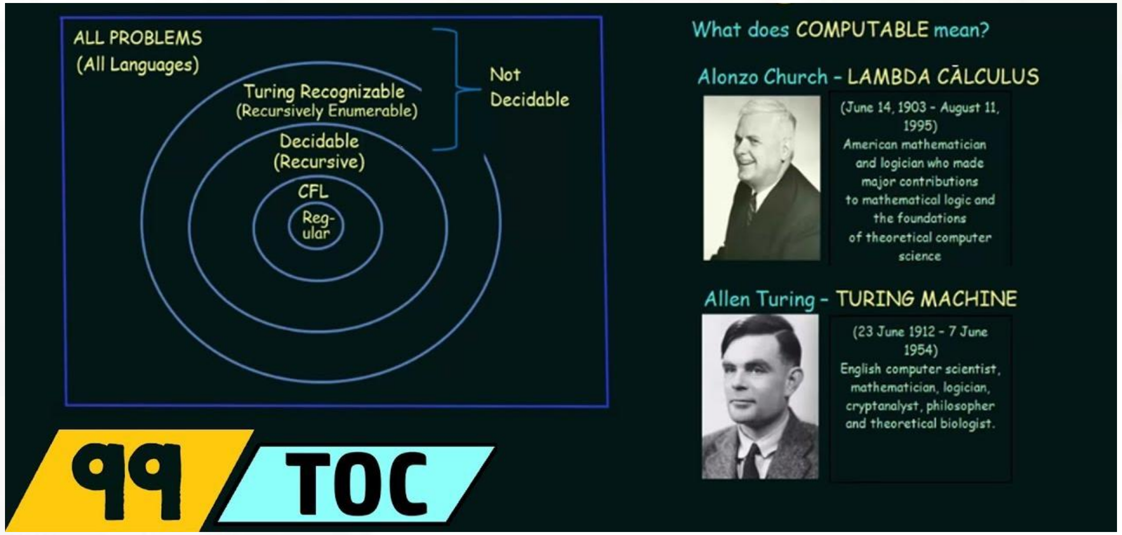
Artificial Intelligence

Turing Machine

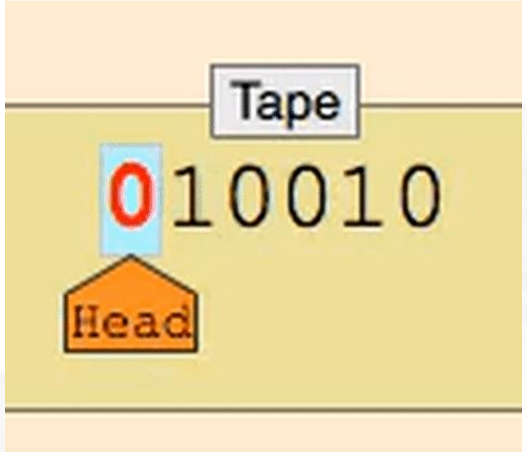
- a mathematical model of computation describing an abstract machine that manipulates symbols on a strip of tape according to a table of rules
- Despite the model's simplicity, it is capable of implementing **any computer algorithm**.

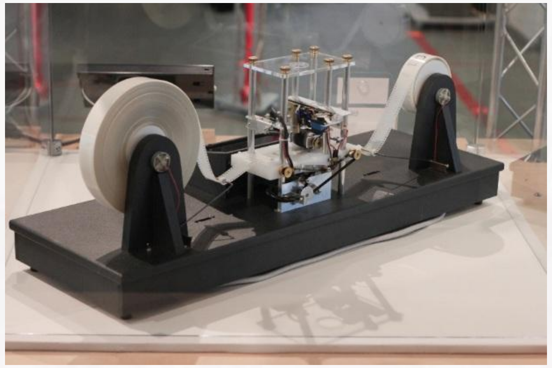


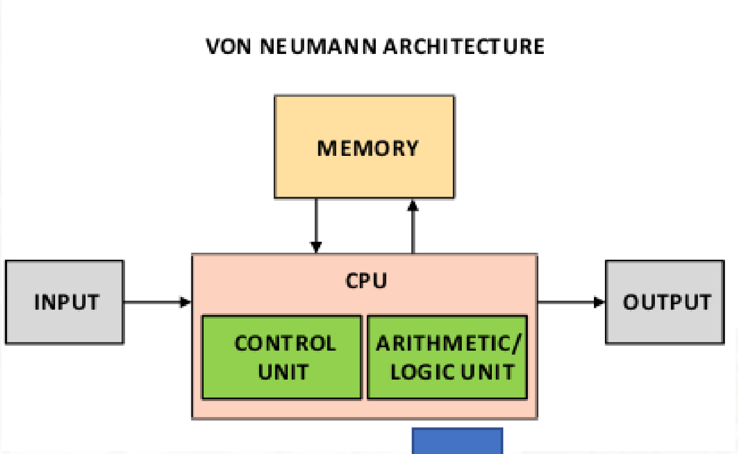


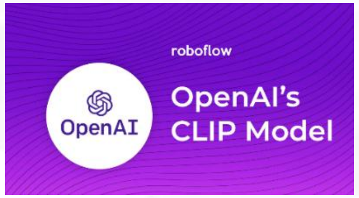


Artificial Intelligence




Turing Machine





Foundation model

Text to Image

ChatGPT

SAM

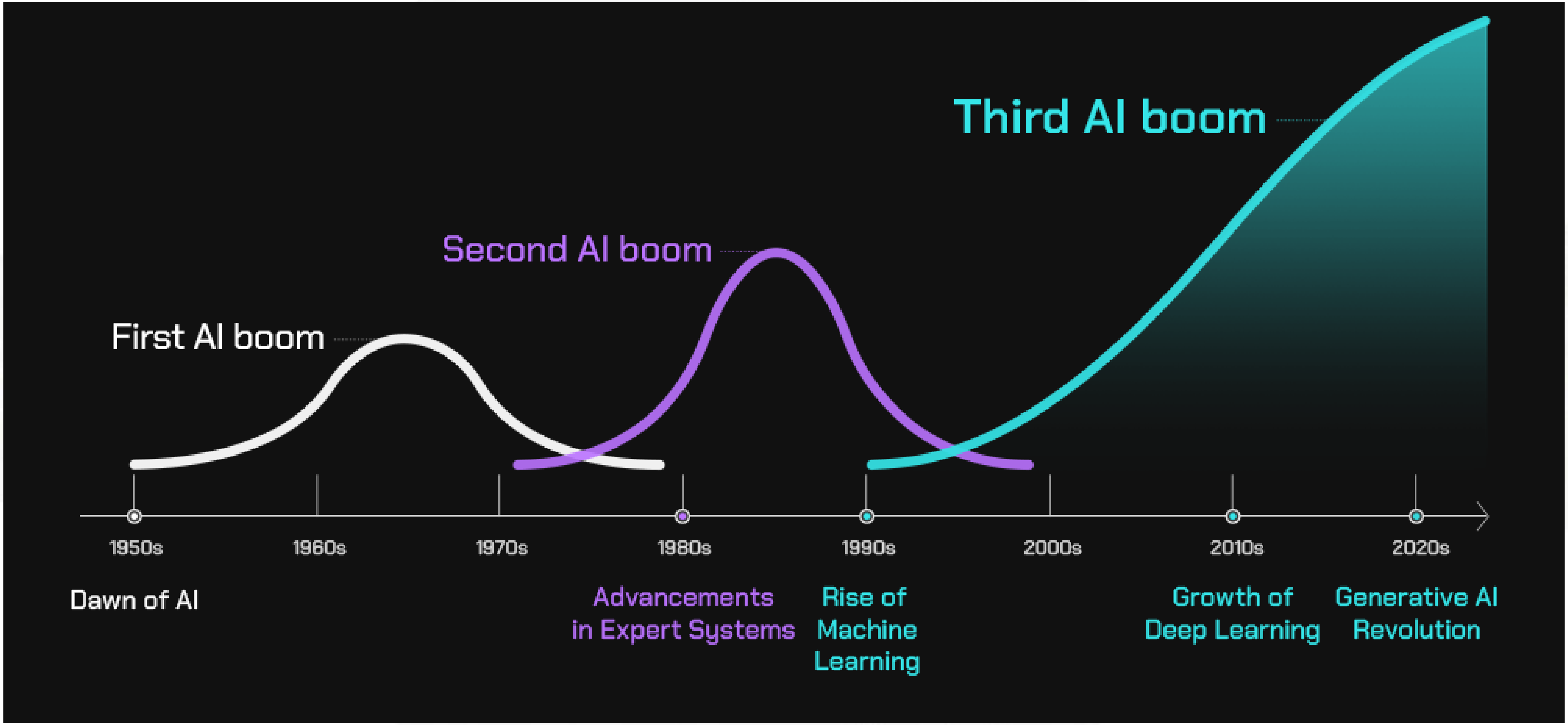

SORA

MLLM

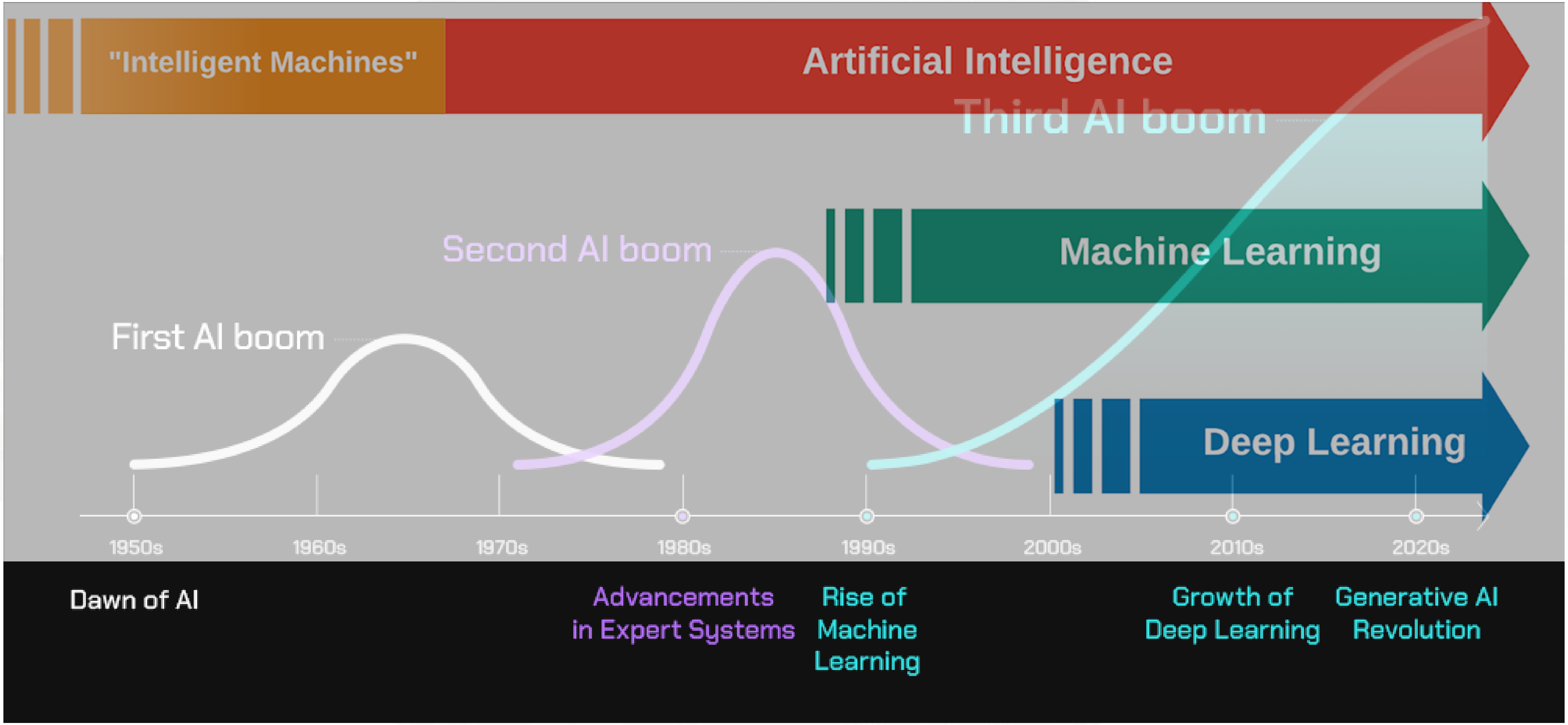
DriveGPT


Genie

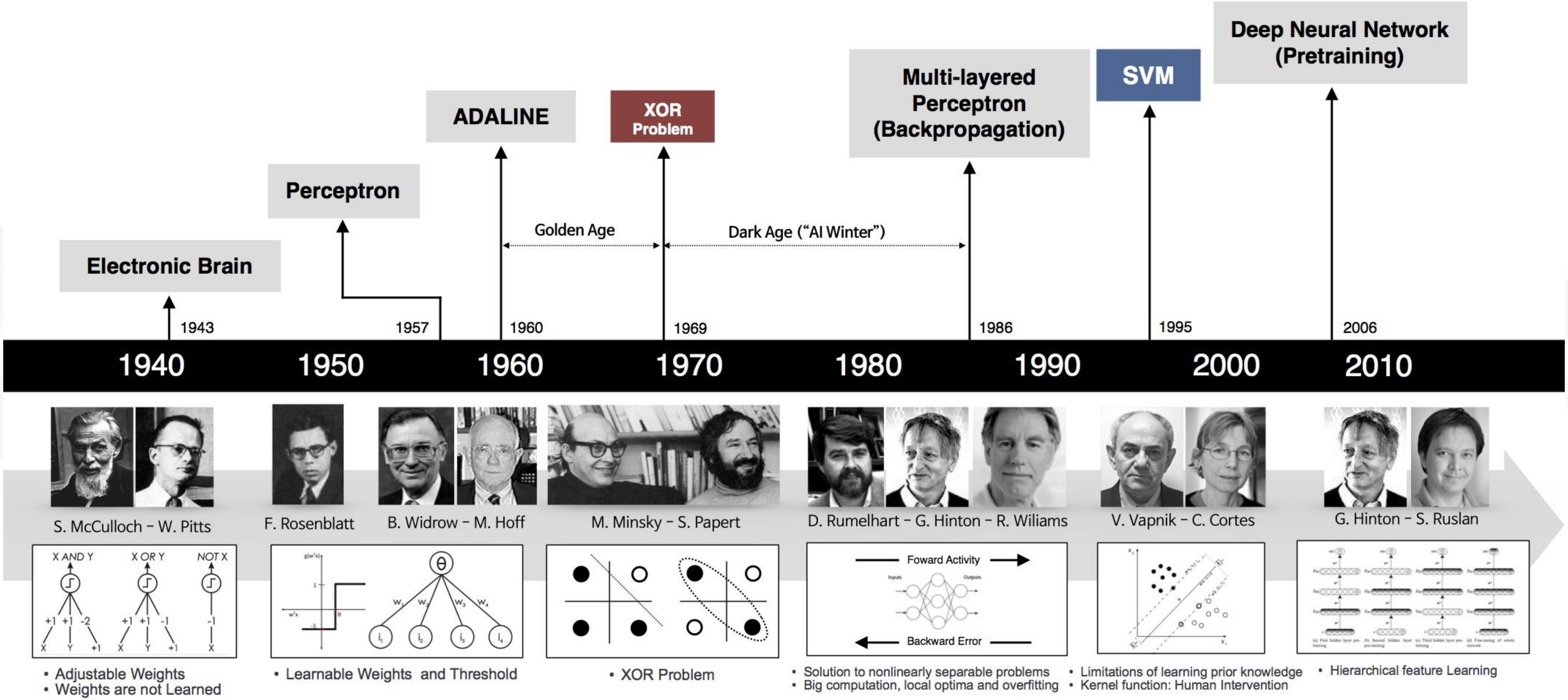
The Development of AI



The Development of AI

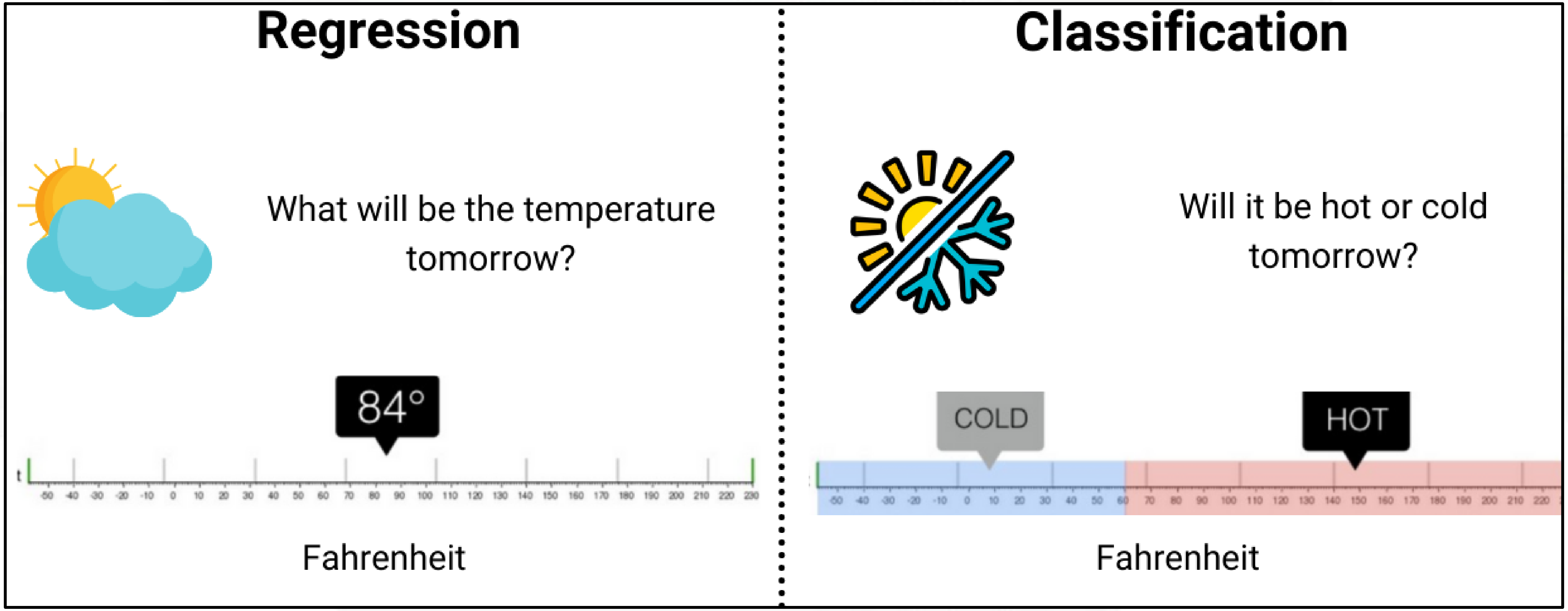


The Development of DL



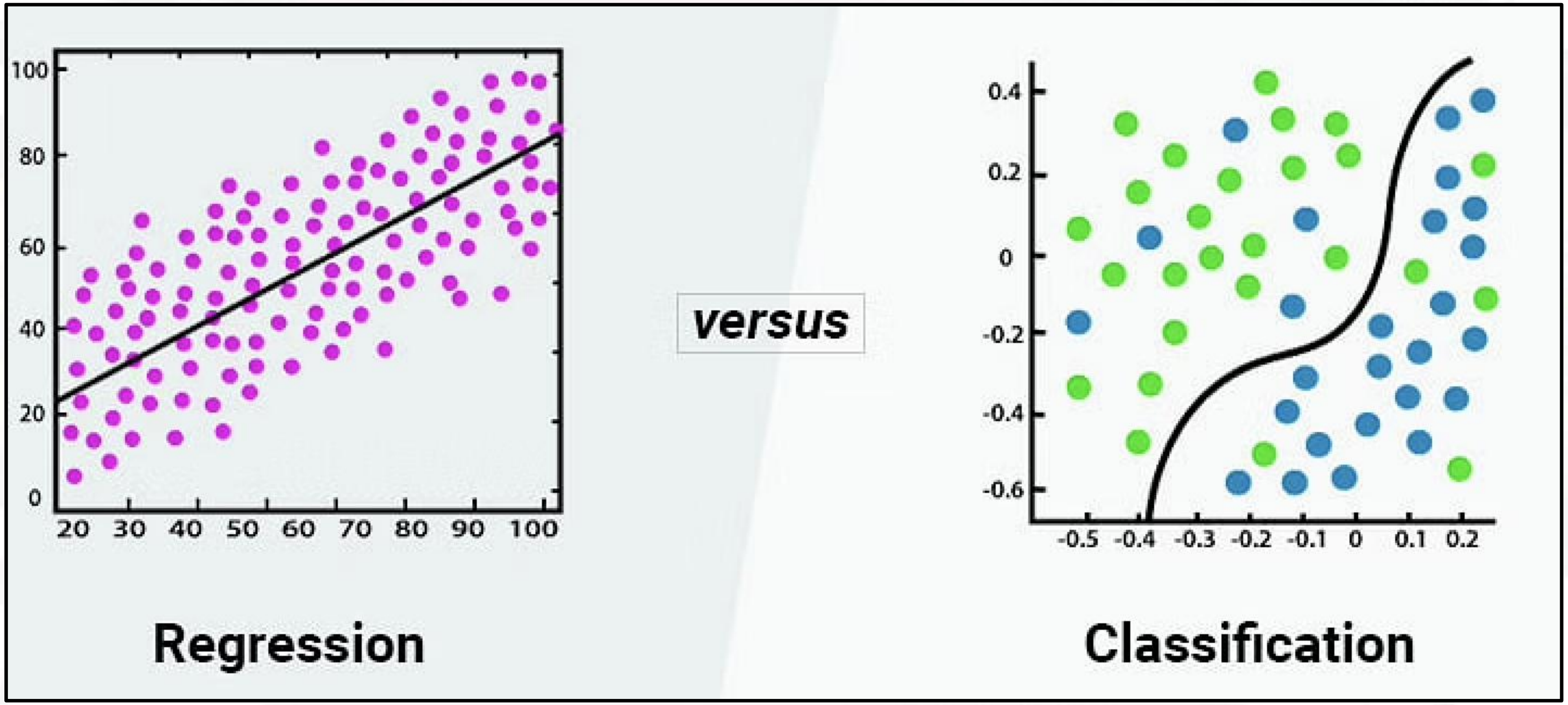
Data View: Fundamental Problems

- Output: Continuous description or discrete state



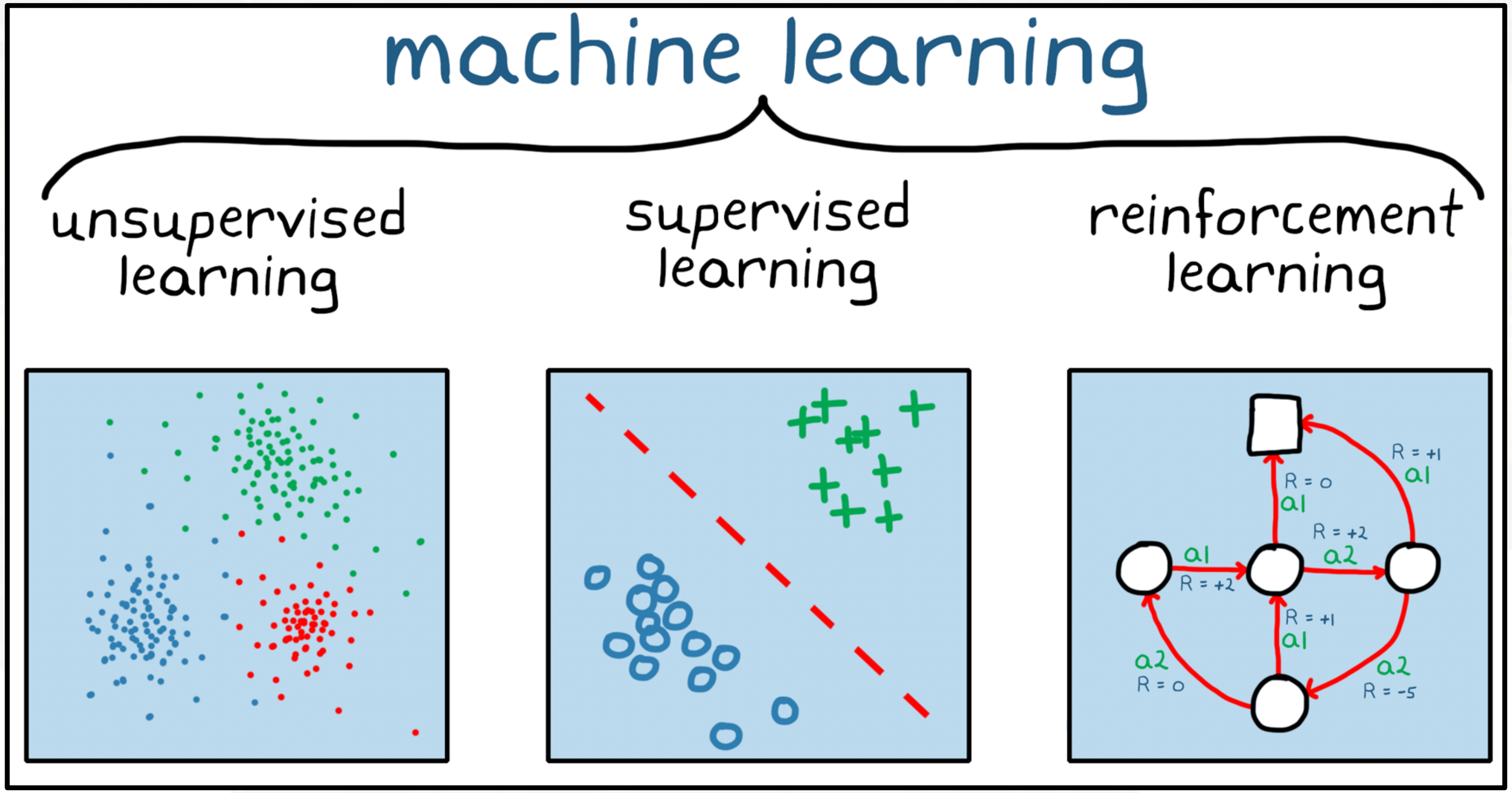
Data View: Fundamental Problems

- Output: Continuous description or discrete state

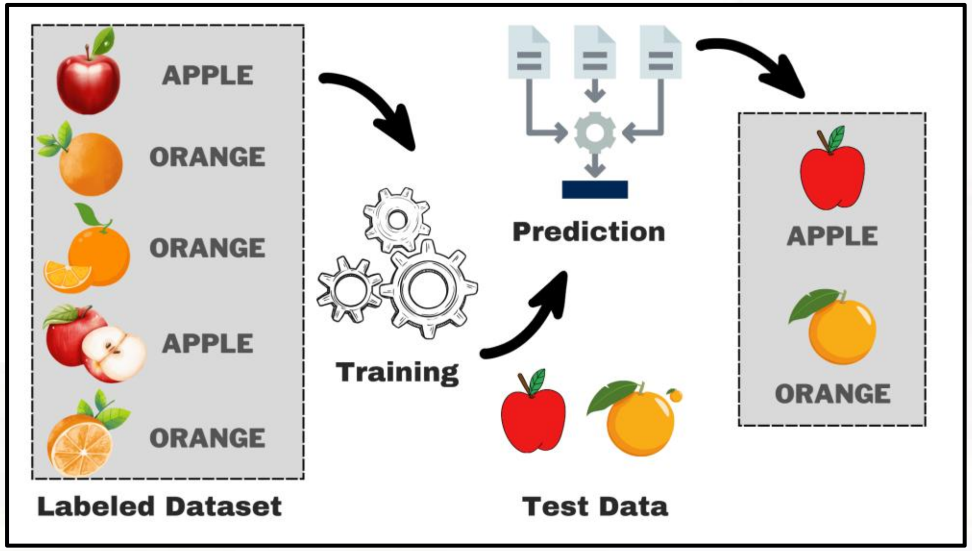


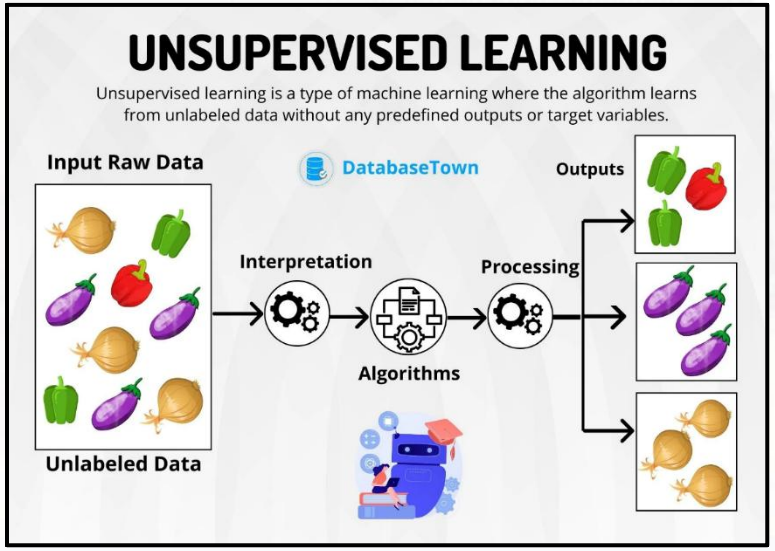
Data view: Learning Paradigm

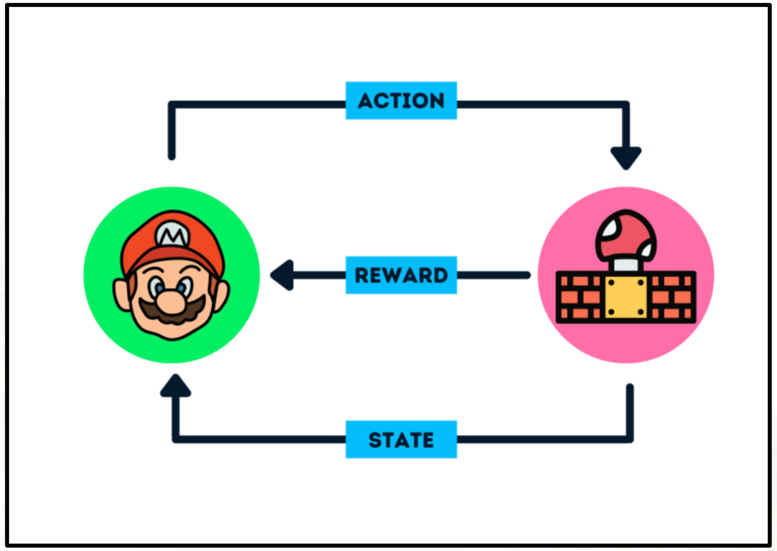
- Availability of outputs: unavailable, available, partially available



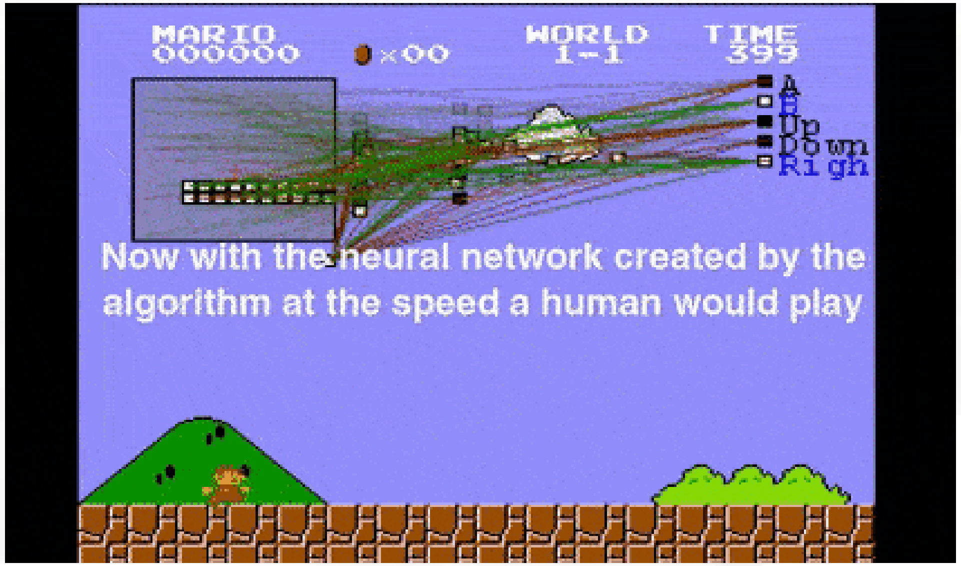
Data view: Learning Paradigm







Weight	Texture	Label
150g	Bumpy	Orange
170g	Bumpy	Orange
140g	Smooth	Apple
130g	Smooth	Apple

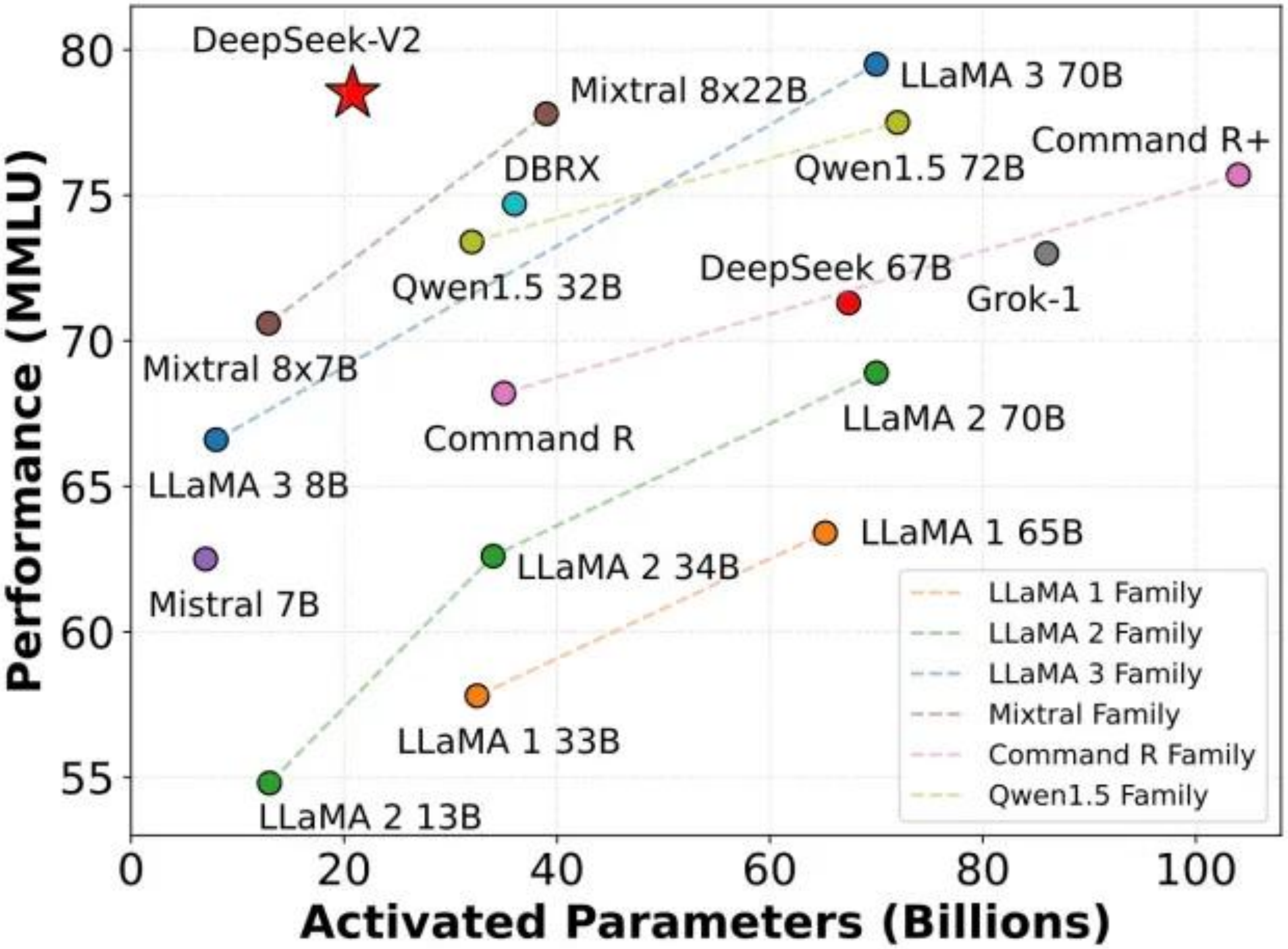


Action	↑	↓	→	←
Start	0	0	1	0
Idle	0	0	0	0
Correct Path	0	50	22	0
Wrong Path	15	0	18	0
End	0	0	1	0

Model view: neural network

Aspect	Neural Networks	Support Vector Machines (SVM)
Model Type	Non-linear, based on layers of neurons.	Linear or non-linear (with kernel tricks), based on maximizing margins.
Data Requirements	Requires large datasets for training.	Works well with small to medium-sized datasets.
Complexity	Can model highly complex, non-linear relationships.	Handles non-linearity through kernel functions.
Training Time	Computationally expensive, especially for deep learning.	Slower training for large datasets, quadratic complexity in training.
Scalability	Scales well with data and features but requires more computational power.	Struggles with very large datasets and high-dimensional data.
Interpretability	Black-box model; difficult to interpret.	Easier to interpret, especially with linear SVM.

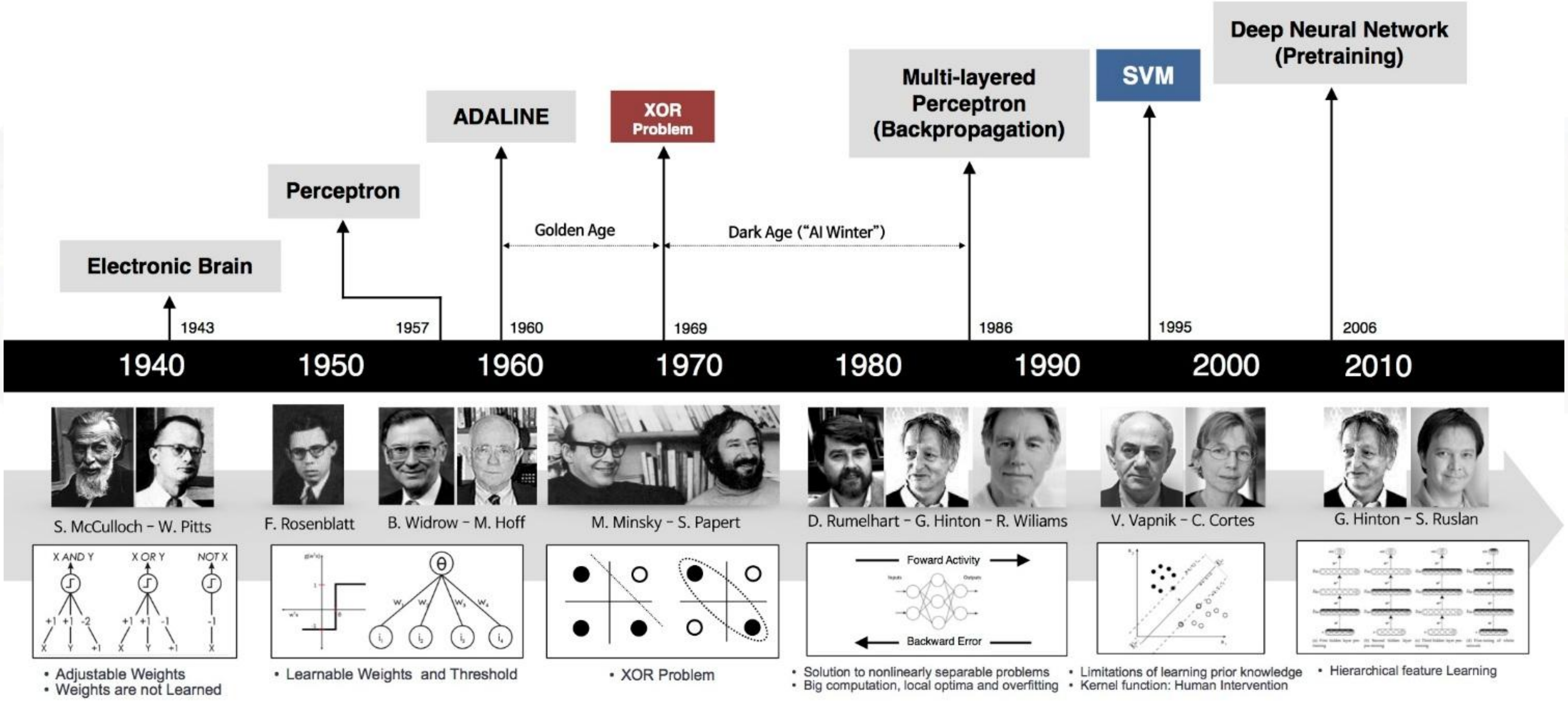
Model view: Neural Network



Model view: neural network

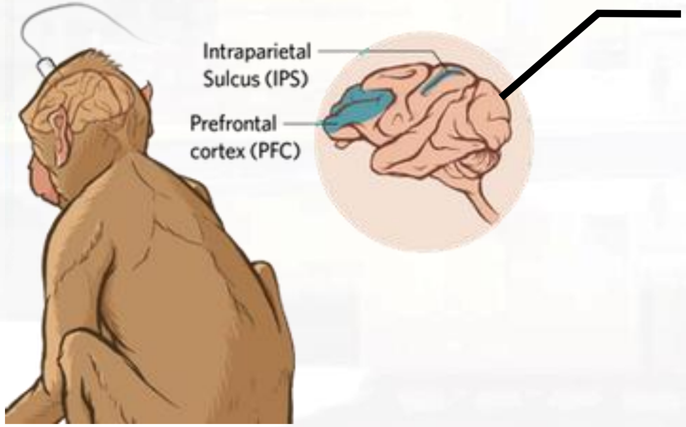
- Large searching space and fitting capacity
- -> Scaling law
- ->Data-driven
- ->Fit everything
- ->Data is all you need and expert is gone

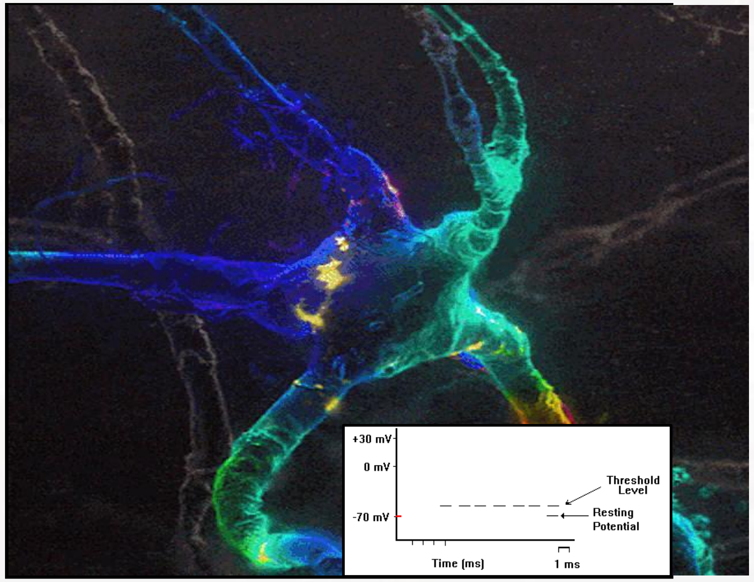
Computational Neuron and Perceptron



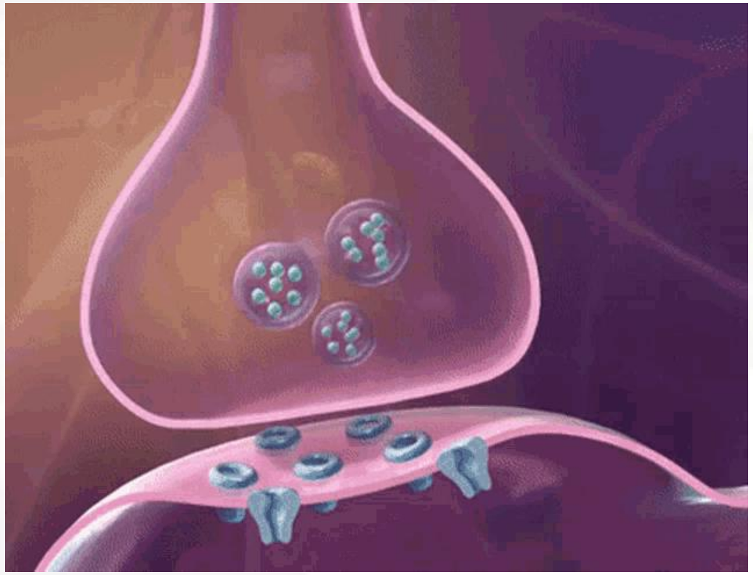
Biological Neuron

Action potentials are the primary means of information transmission in biological neurons. They implement spatiotemporal information processing through an event-driven threshold-triggering mechanism (when the sum of synaptic inputs reaches a critical value), ensuring a balance between signal transmission reliability and energy efficiency.



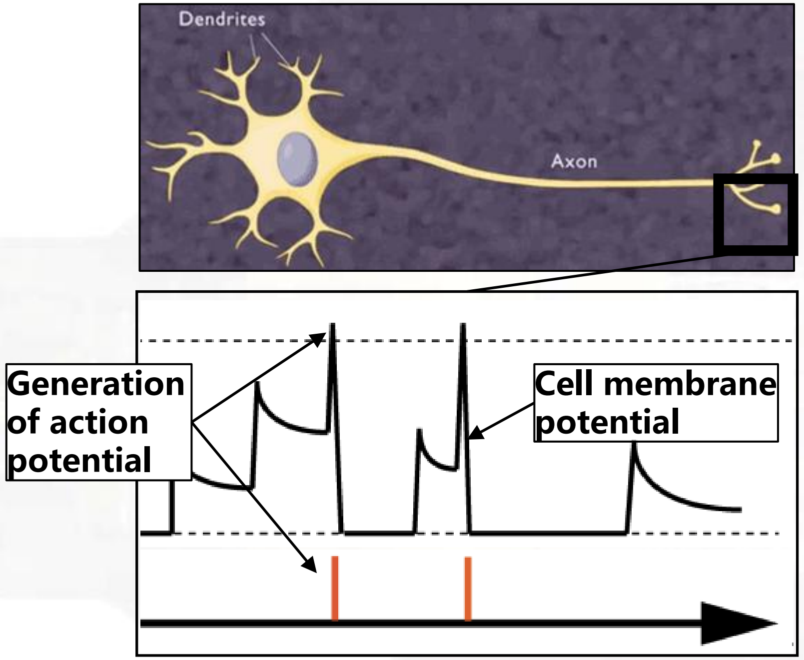


Action potential illustration: A brief and specially shaped transmembrane potential pulse generated when the cell membrane at resting membrane potential is subjected to an appropriate stimulus.

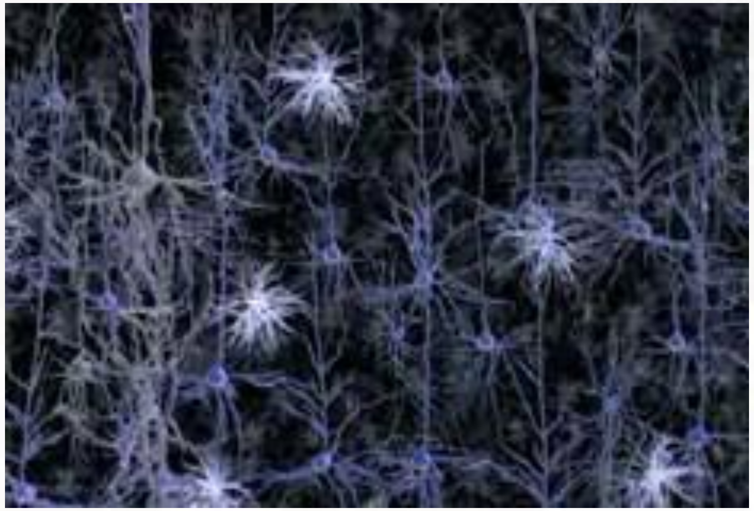


Event-driven schematic: When an electrical signal (action potential) reaches the threshold, it triggers a pulse and transmits information directionally through synaptic release of chemical transmitters.

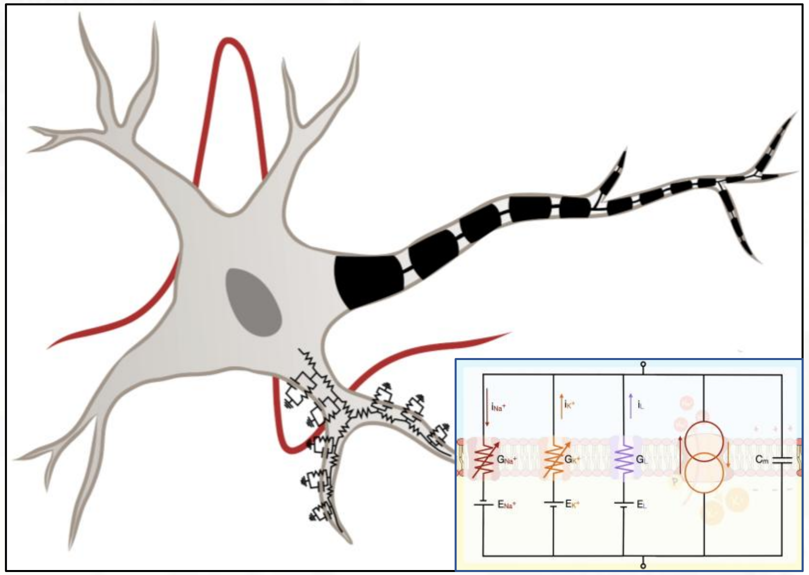
Biological Neuron



Discrete Spiking representation: An action potential is expressed in an **all-or-none** manner.

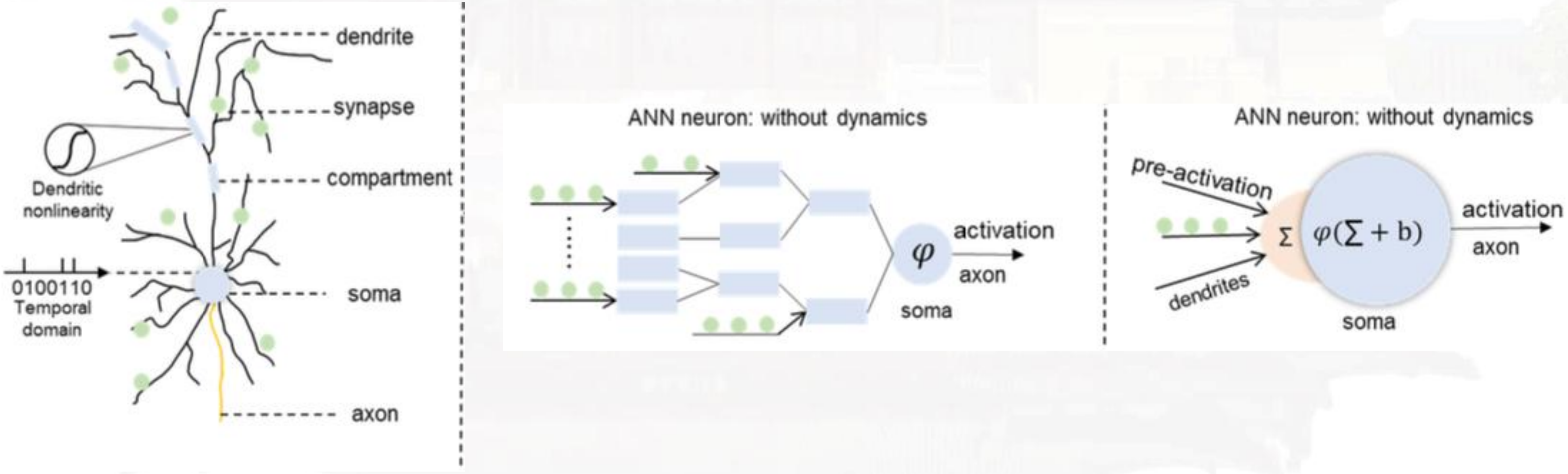


Asynchronous spiking information transmission: Whether the current neuron transmits information is **unrelated to whether other neurons transmit information**; it only depends on whether the threshold of the current neuron is triggered.



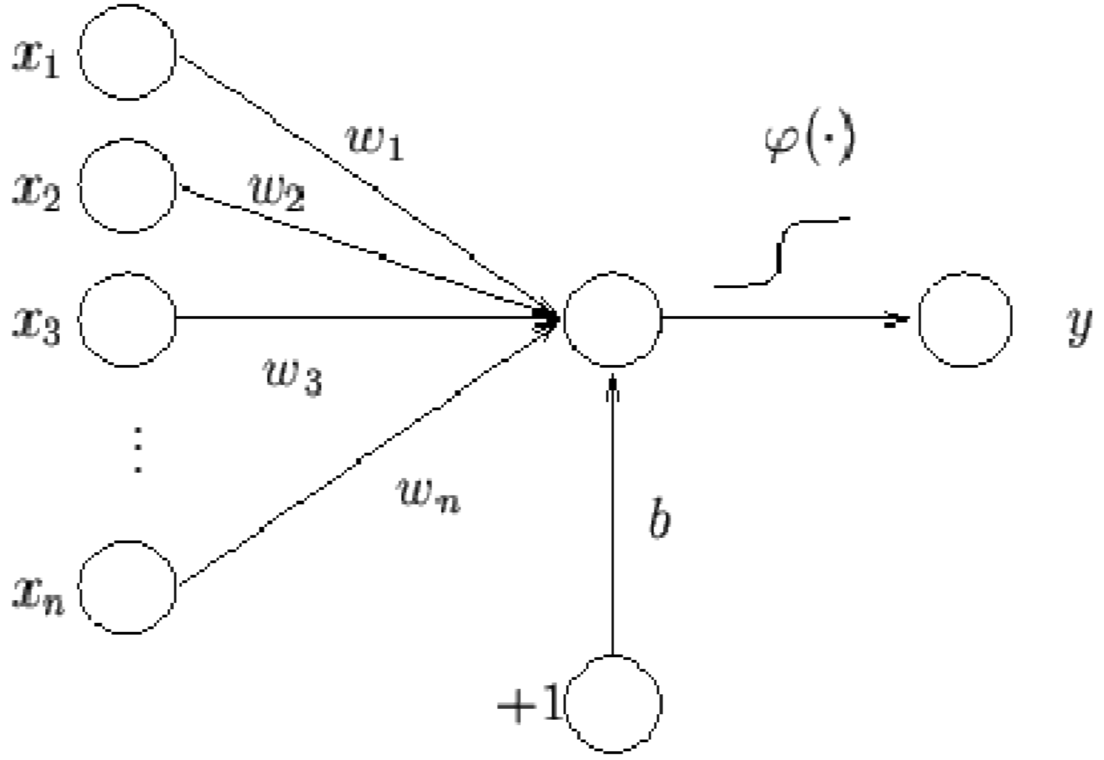
Equivalent circuit

Computational Neuron



Perception

$$\varphi(\vec{x}) = \operatorname{sgn}\left(\sum_{i=1..n} w_i \times x_i + b\right)$$



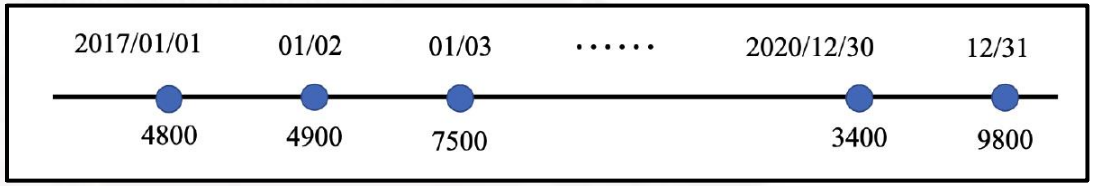
Example: Internet Traffic Prediction

- Suppose someone wants to make money through a video platform; they would care about whether the channel has traffic, so they would know their potential earnings.
- Assume that the backend can see a lot of relevant information, such as the number of people who like posts each day, the number of subscribers, and the number of views.
- Based on all the past information of a channel, it is possible to predict the number of views for tomorrow.
- Find a function whose input is the backend information and whose output is the total number of views the channel will have the next day.

Solution: Three Steps

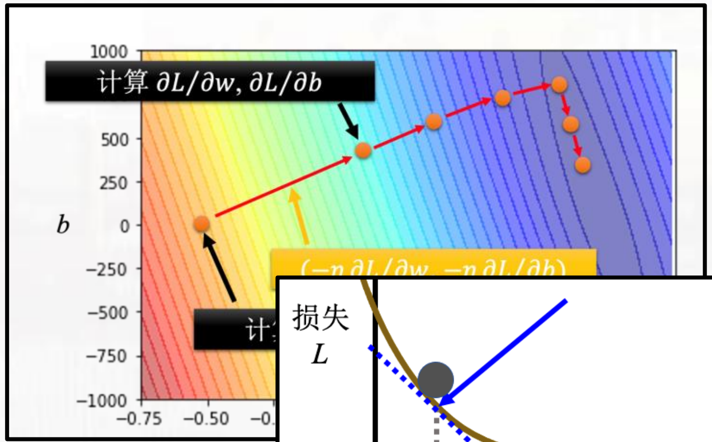
- **Design a function with unknown parameters**
  - model: f、feature: x_1
  - parameter: b , w
  - weight: w 、 bias: b
- **Define loss function: L**
  - Mean Absolute Error, MAE: $e = |\hat{y} - y|$
  - Mean Squared Error, MSE: $e = (\hat{y} - y)^2$
  - cross entropy
- **Solve an optimization problem**
  - global minimum and local minimum

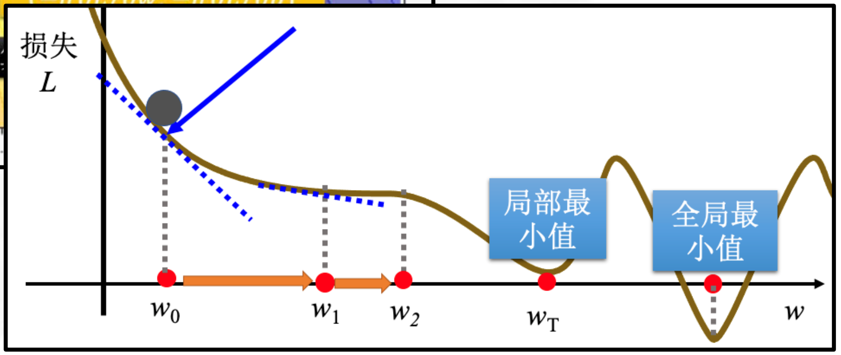
$$y = b + wx_1$$



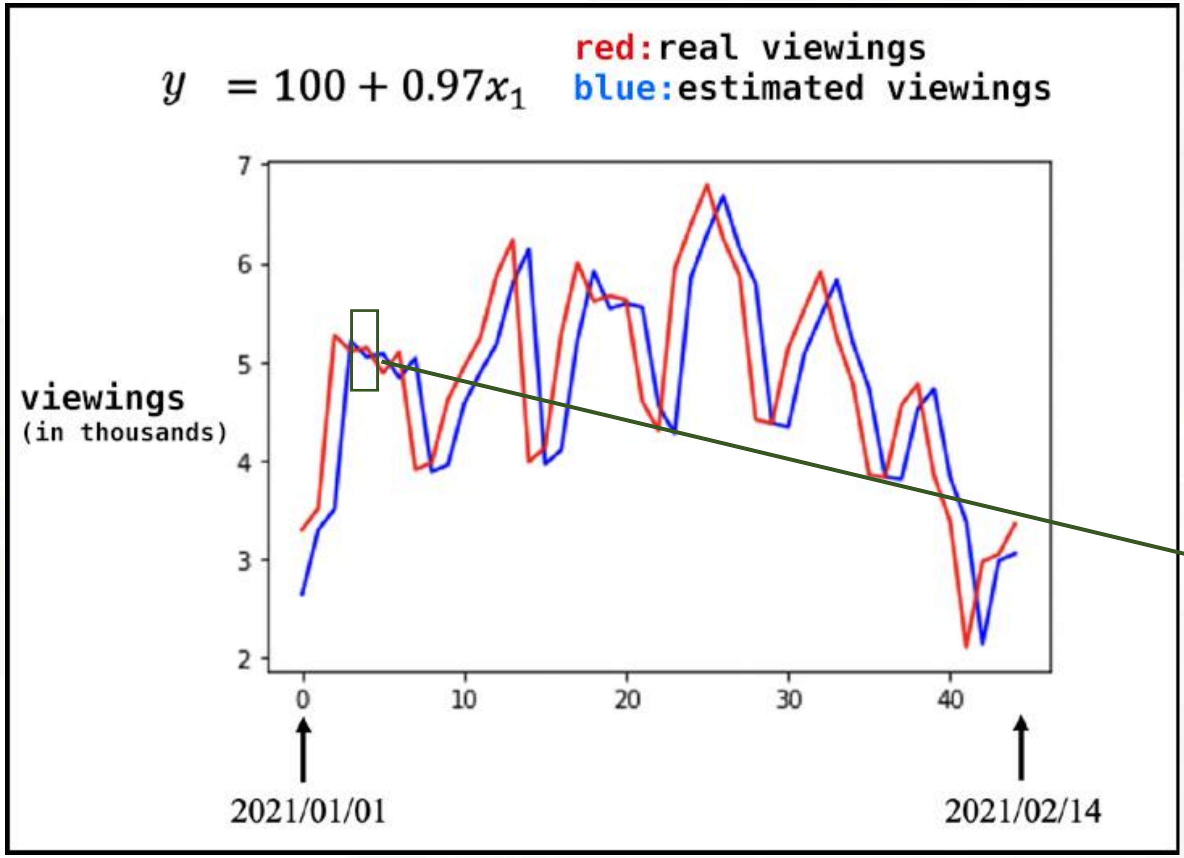
$$\hat{y} = 500 + 1x_1 \qquad e_1 = |y - \hat{y}| = 400$$

$$e_2 = |y - \hat{y}| = 2100 \qquad L = \frac{1}{N}\sum_n e_n$$

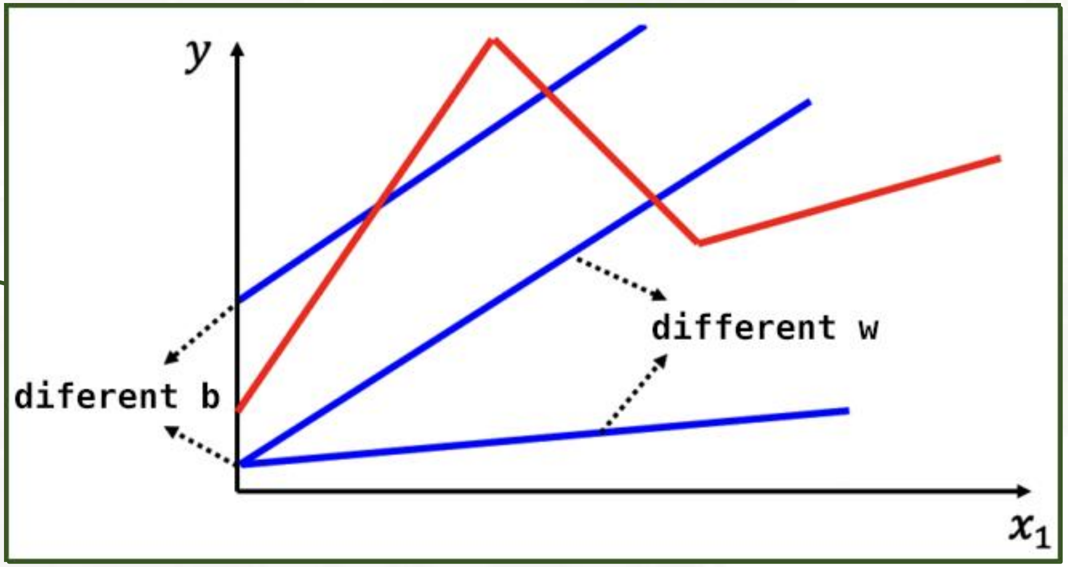




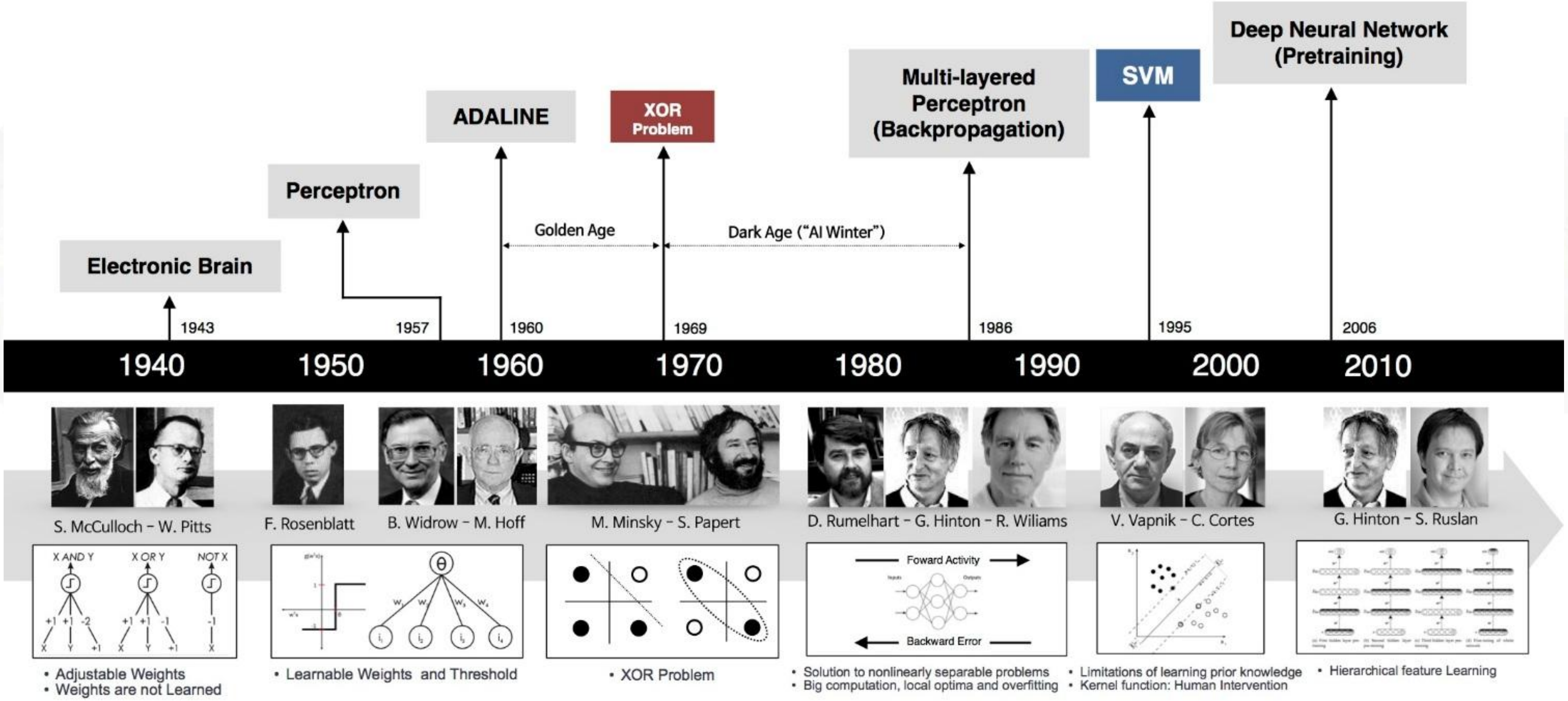
Advantage of Perception



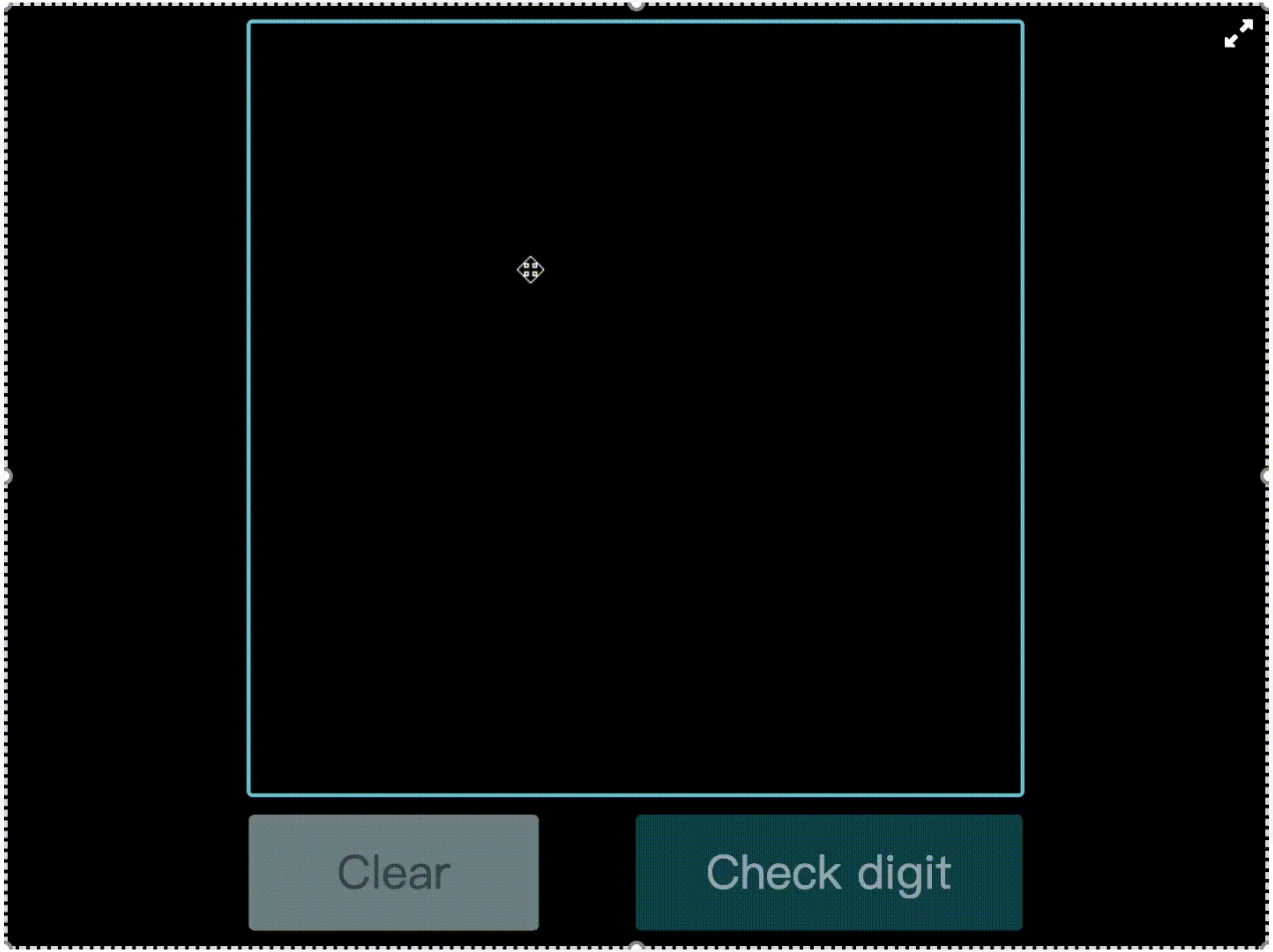
$$y = b + wx_1$$



Multi-Layered Perceptron

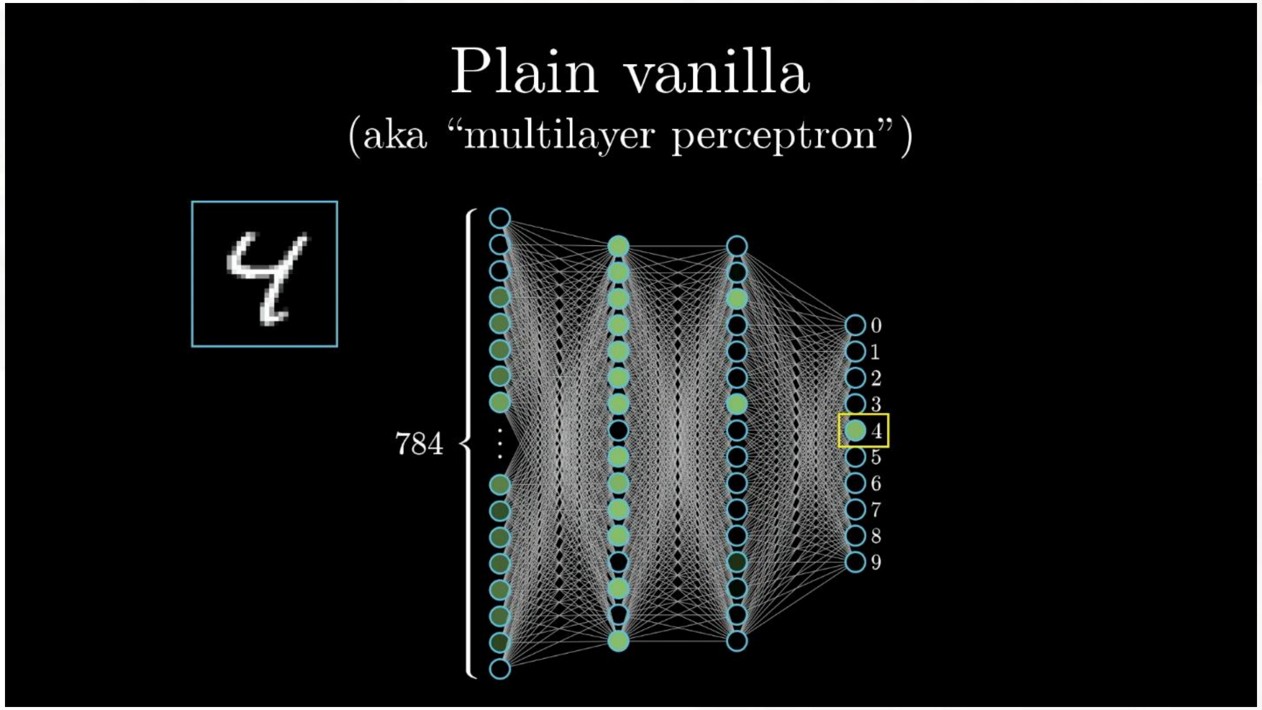


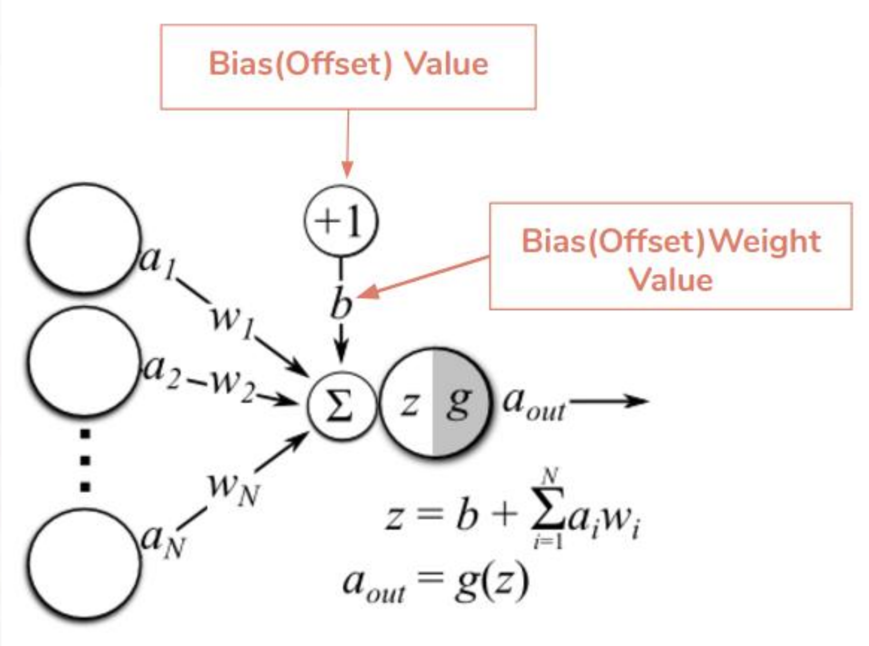
Multi-Layered Perceptron



Multi-Layered Perceptron

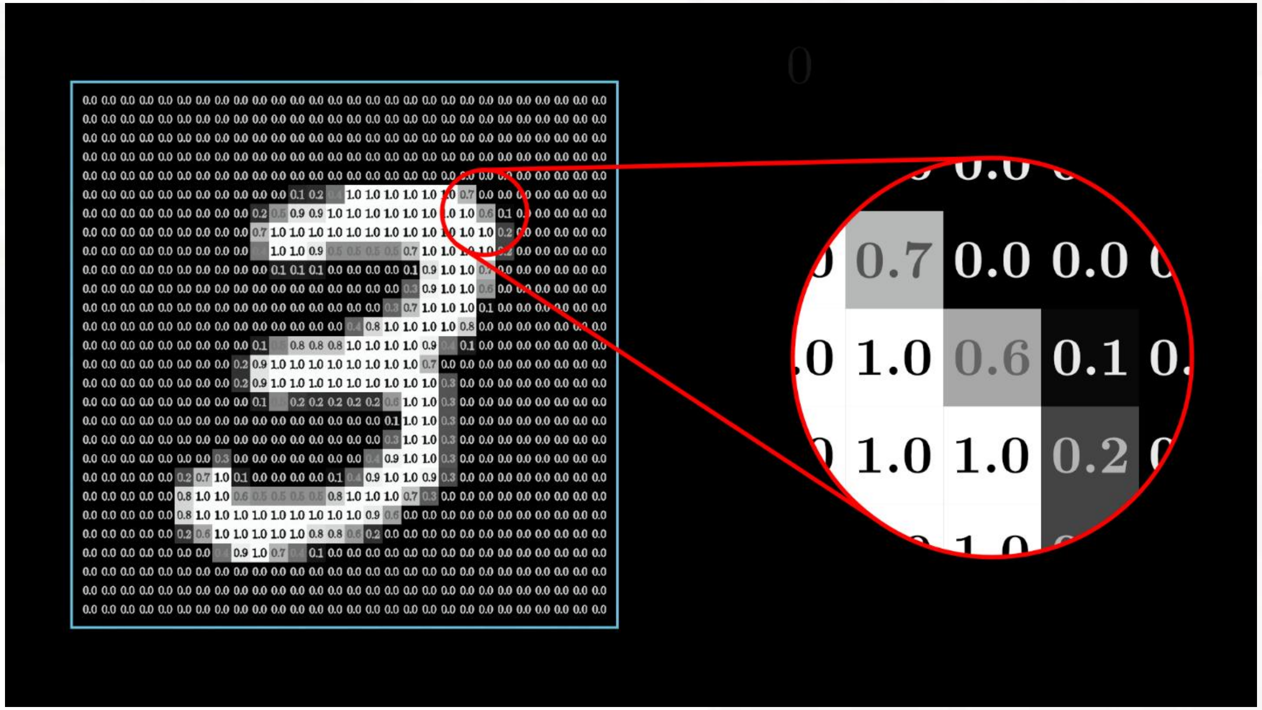
- Understand MLP, understand everything
  - A few layers of neurons linked together.

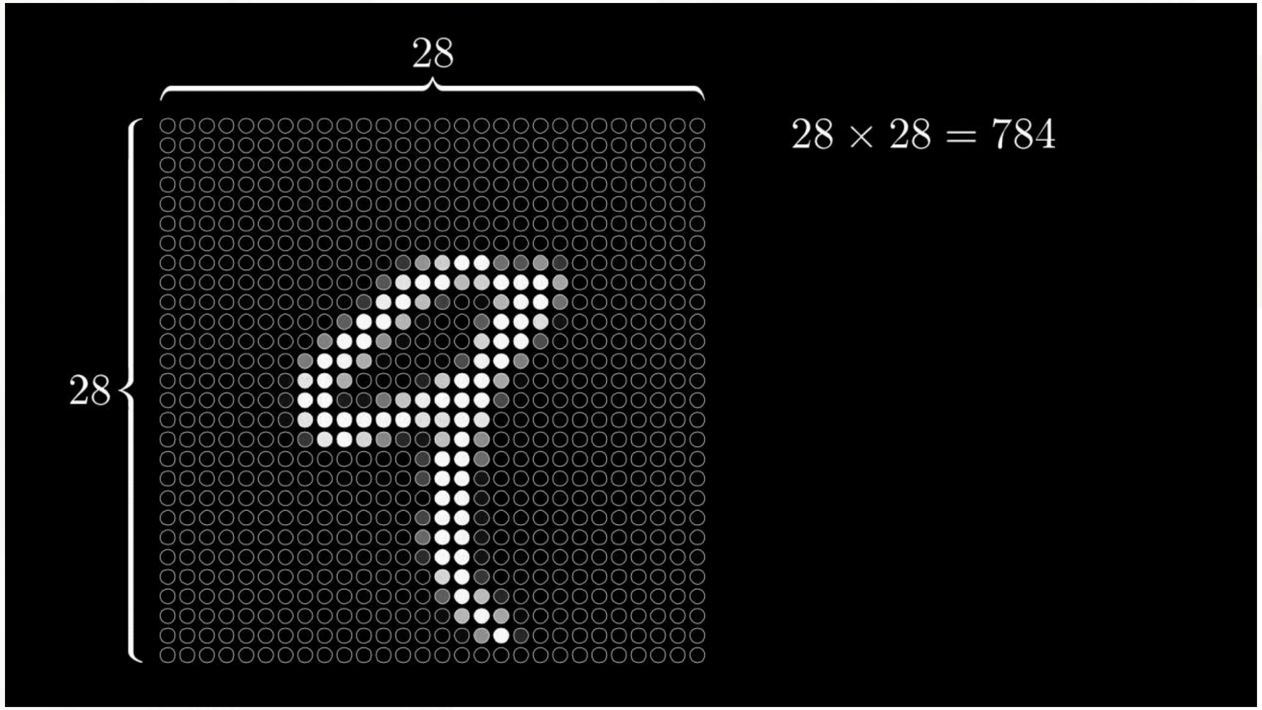




Multi-Layered Perceptron

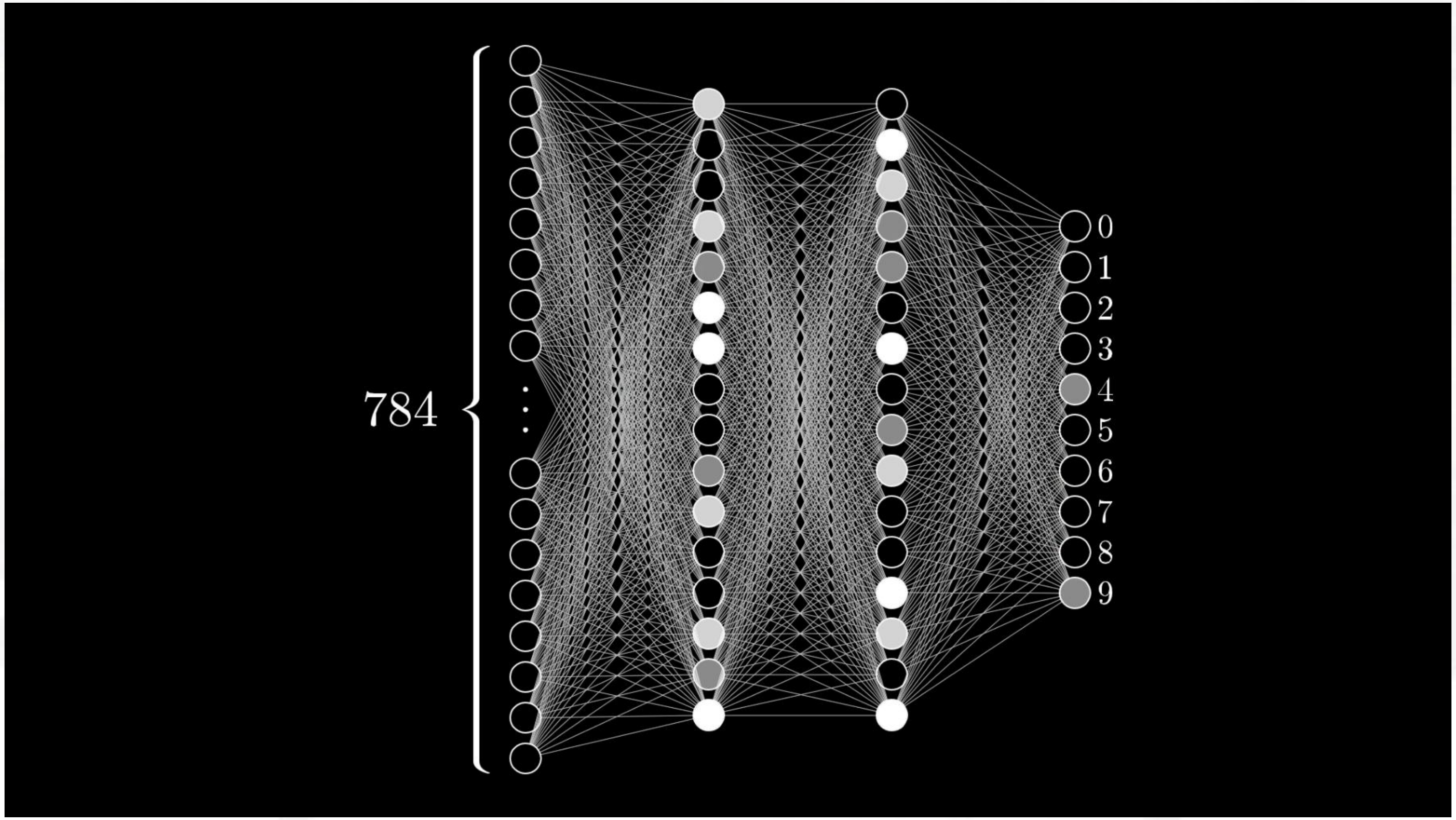
- Neuron: a thing that holds a number
  - represent the inputs and outputs of our network (the images and digit predictions) in terms of these neuron values





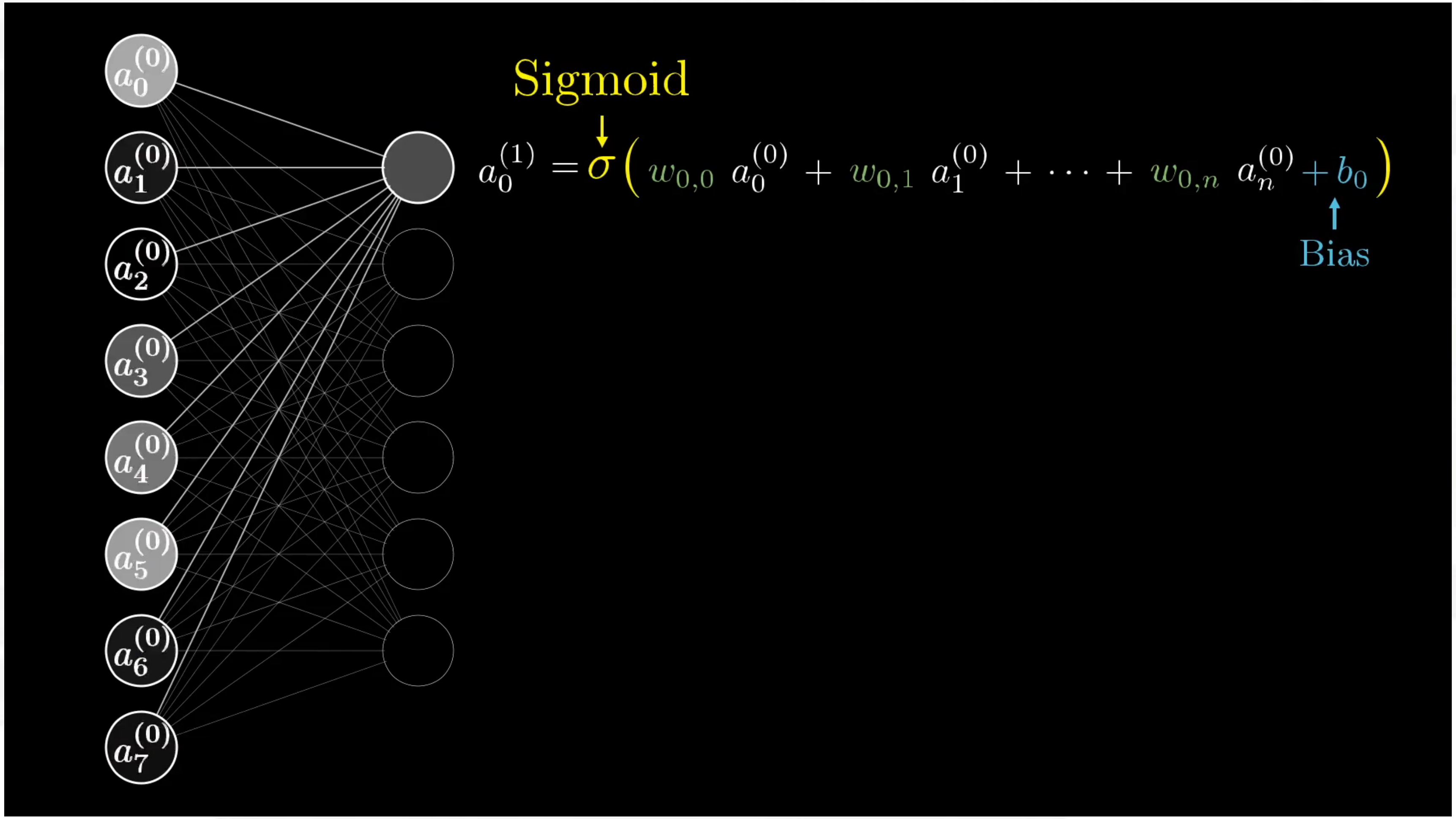
Multi-Layered Perceptron

what kind of digit does this network think it's looking at? How certain does it feel?

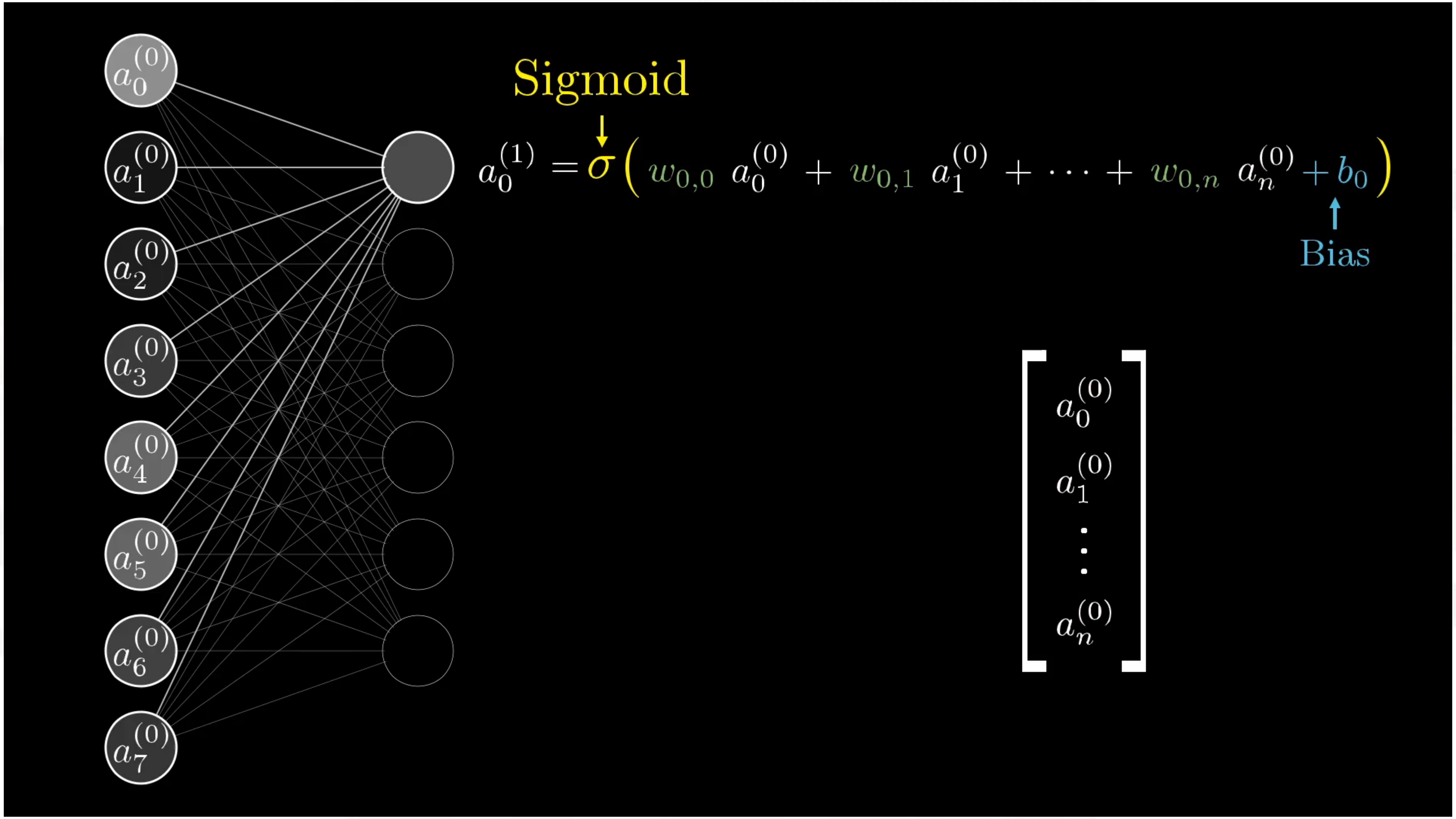


Multi-Layered Perceptron

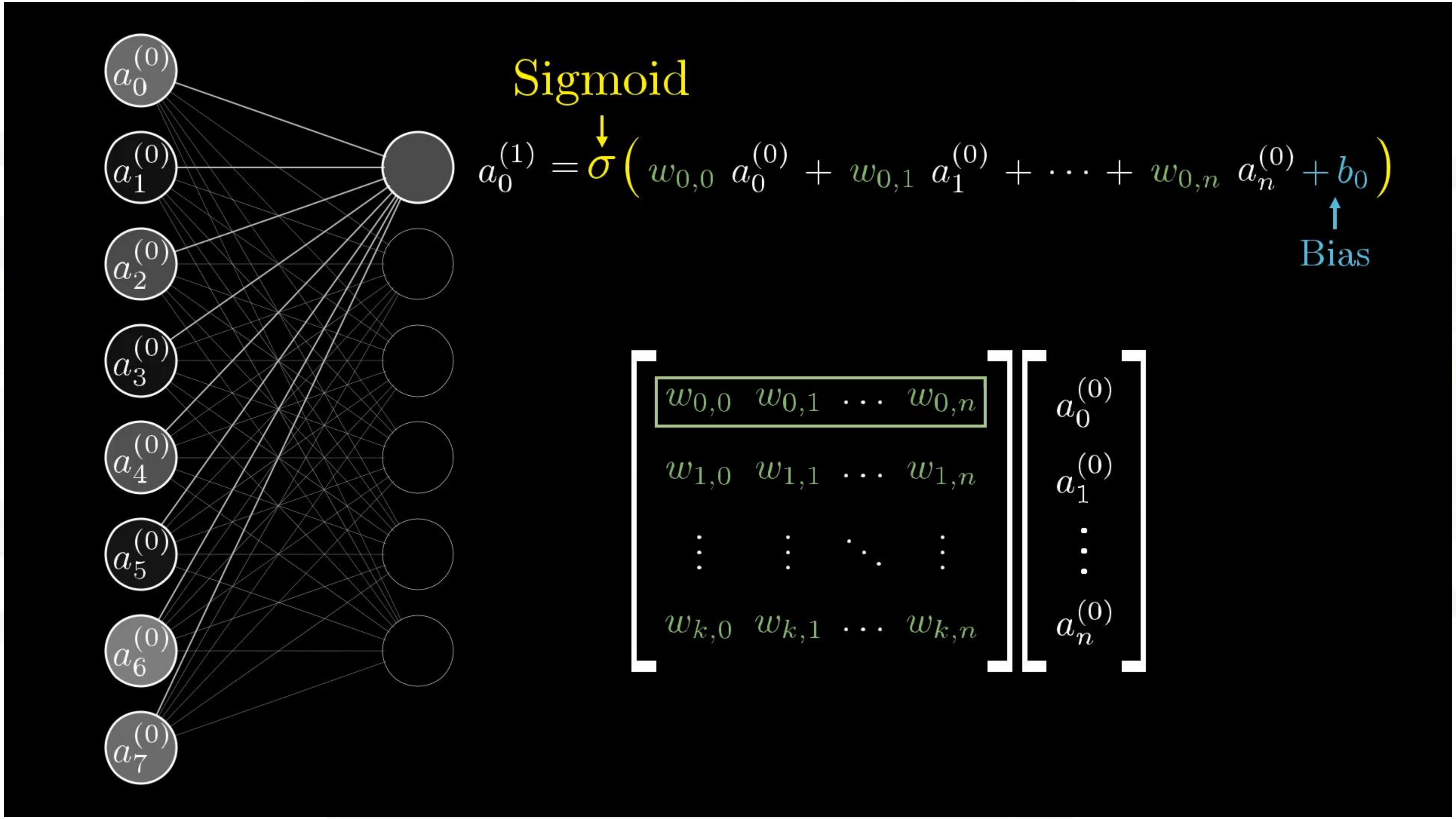
More Compact Notation



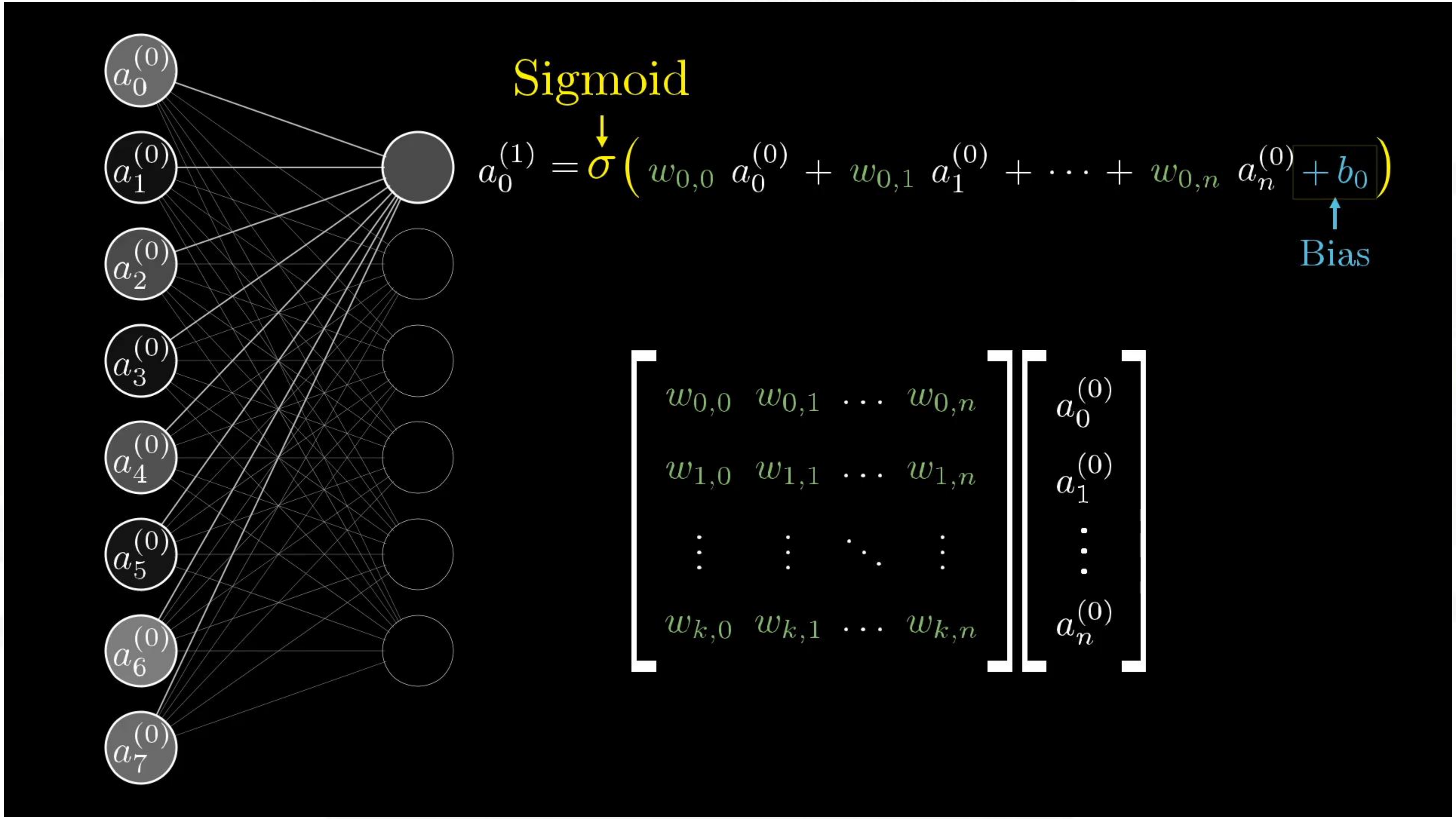
Multi-Layered Perceptron



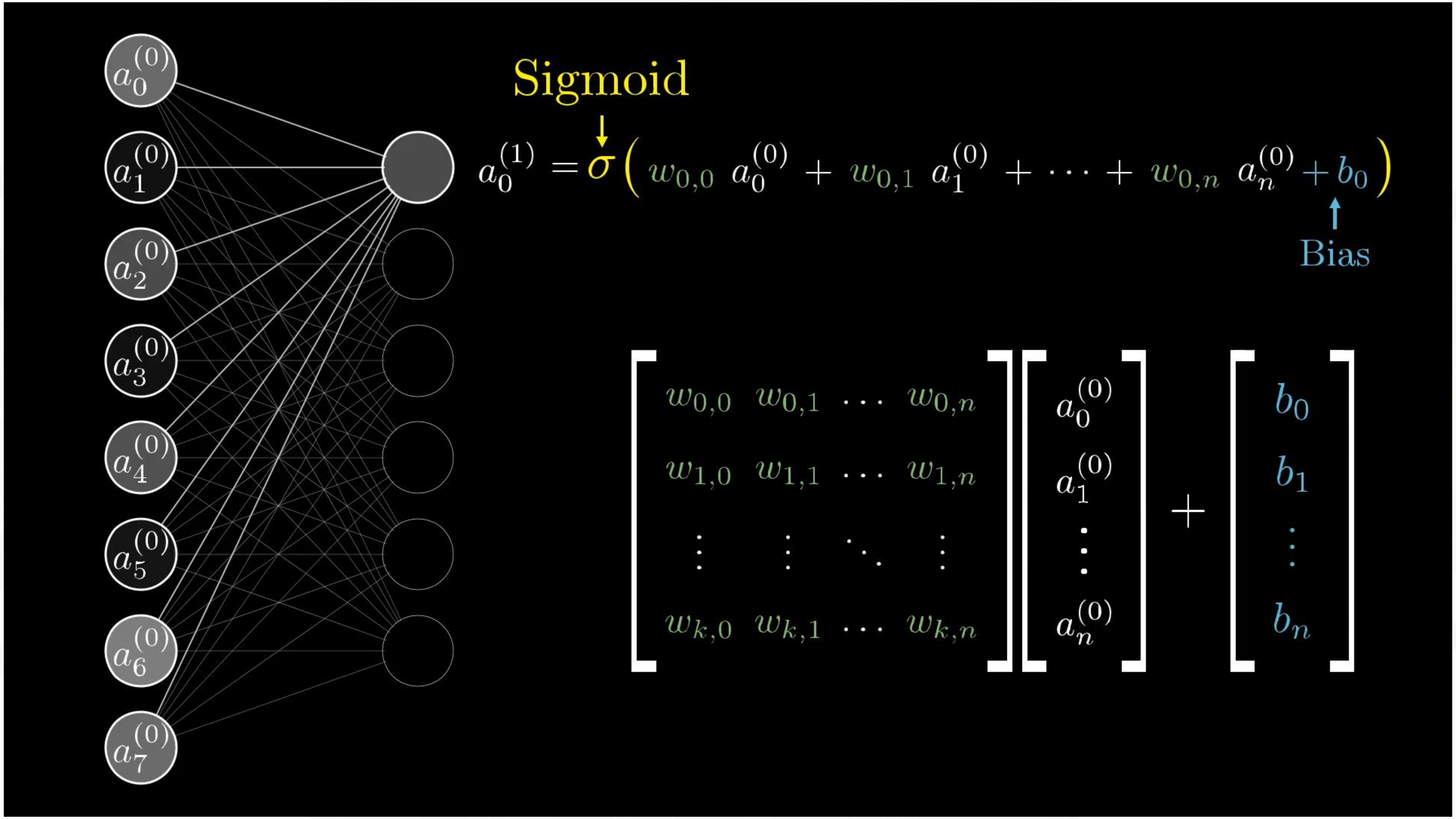
Multi-Layered Perceptron



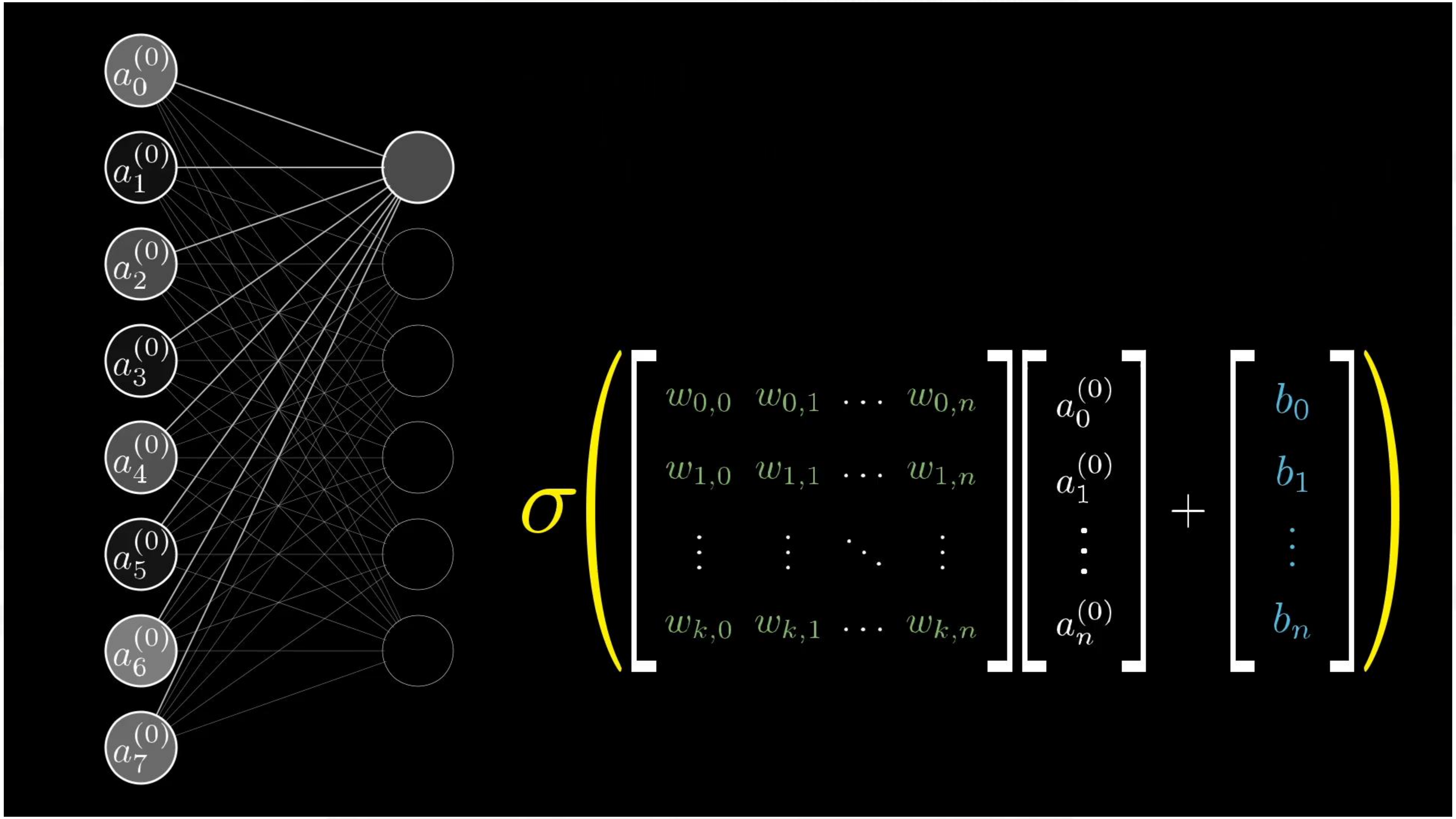
Multi-Layered Perceptron



Multi-Layered Perceptron

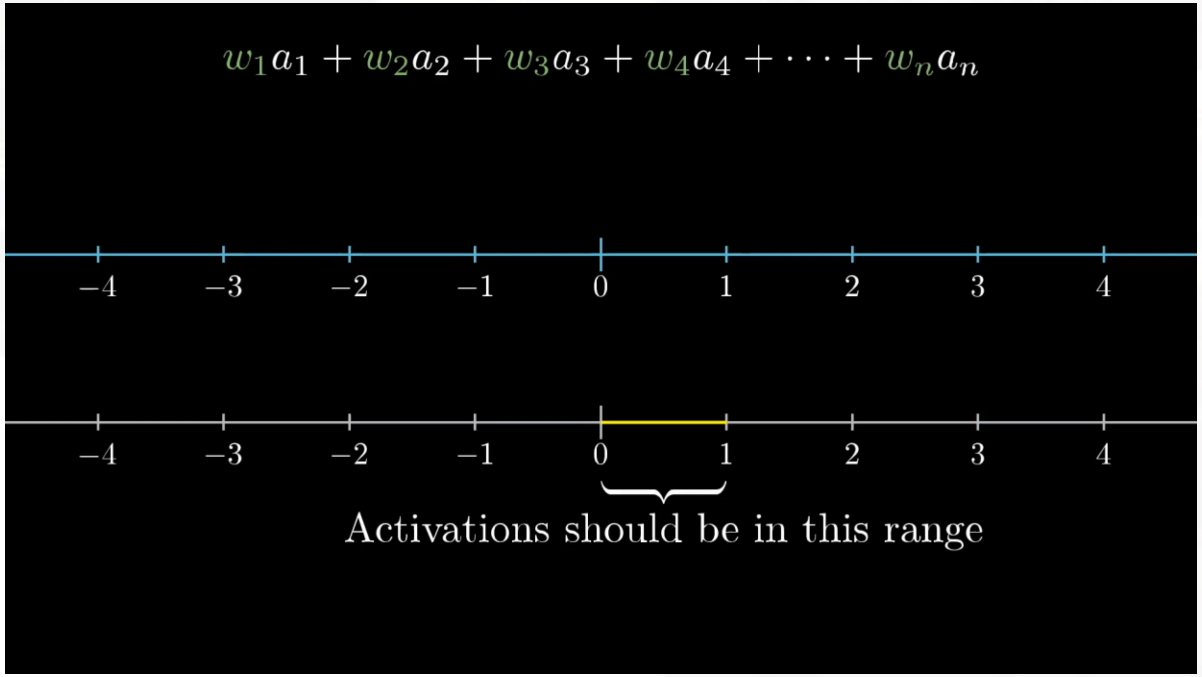


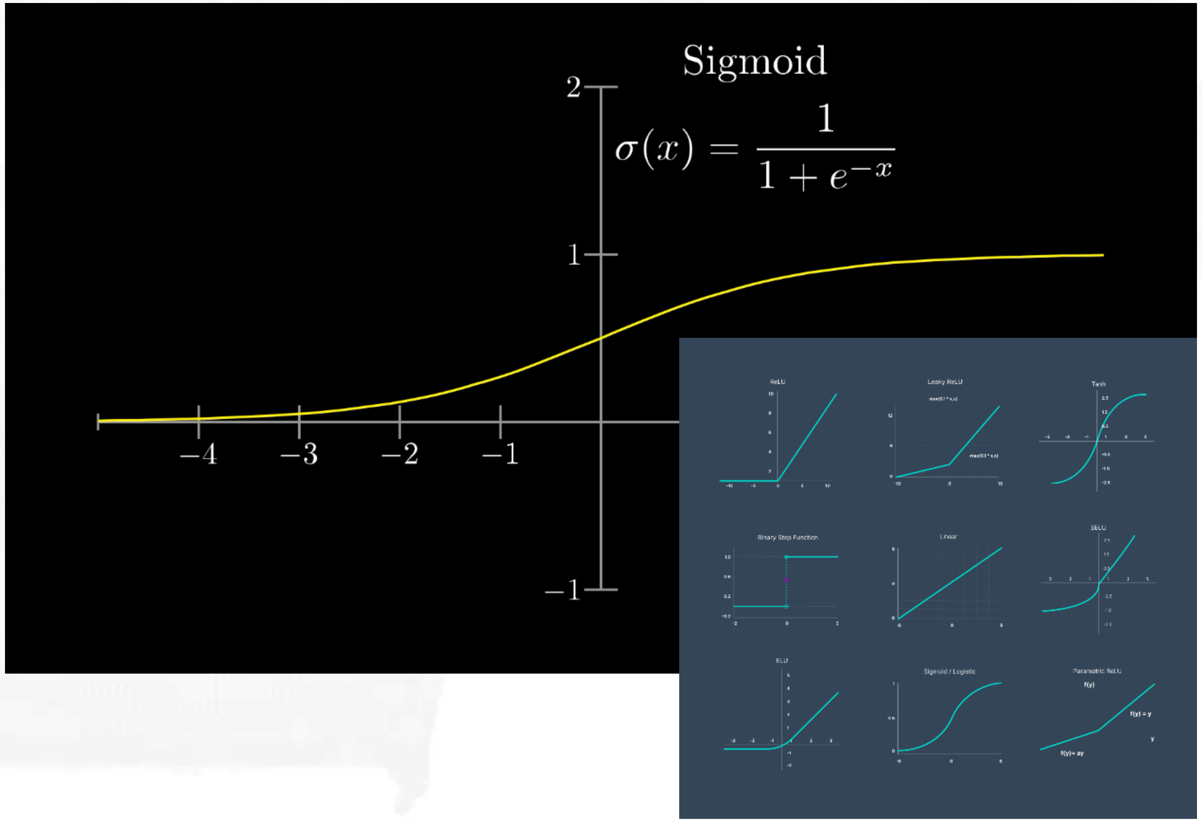
Multi-Layered Perceptron



Multi-Layered Perceptron

- The result of the weighted sum like this can be any number, but for this network we want the activations to be values between 0 and 1.





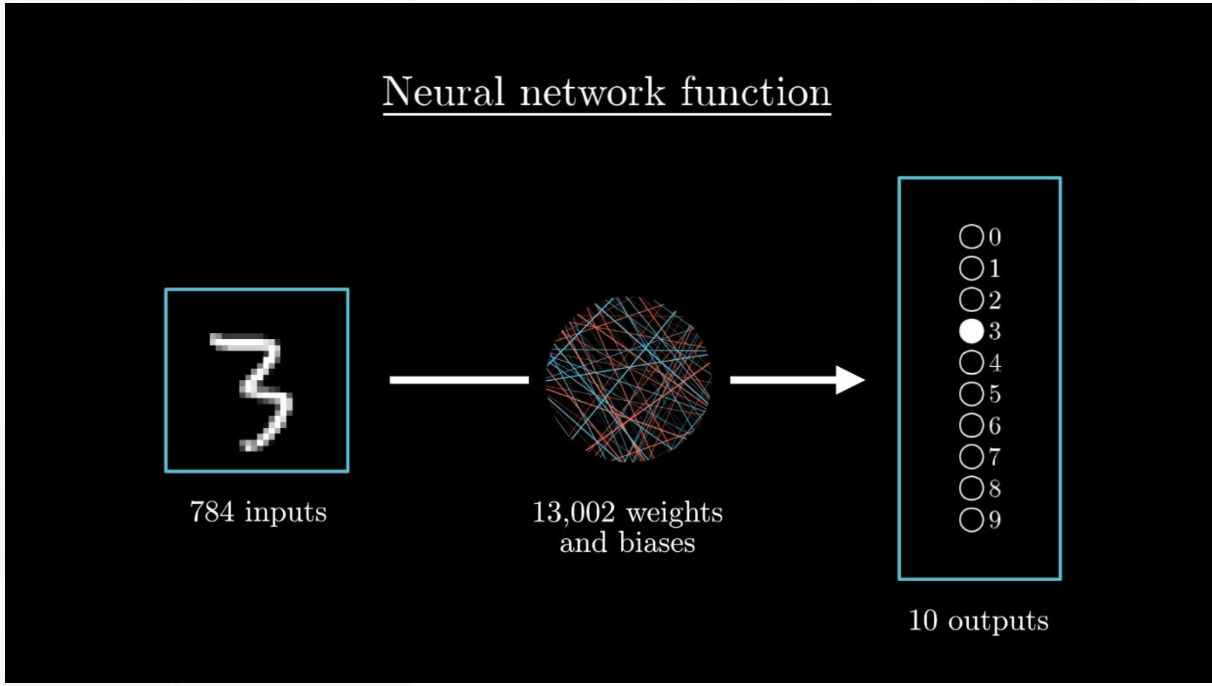
Multi-Layered Perceptron

Formulated function

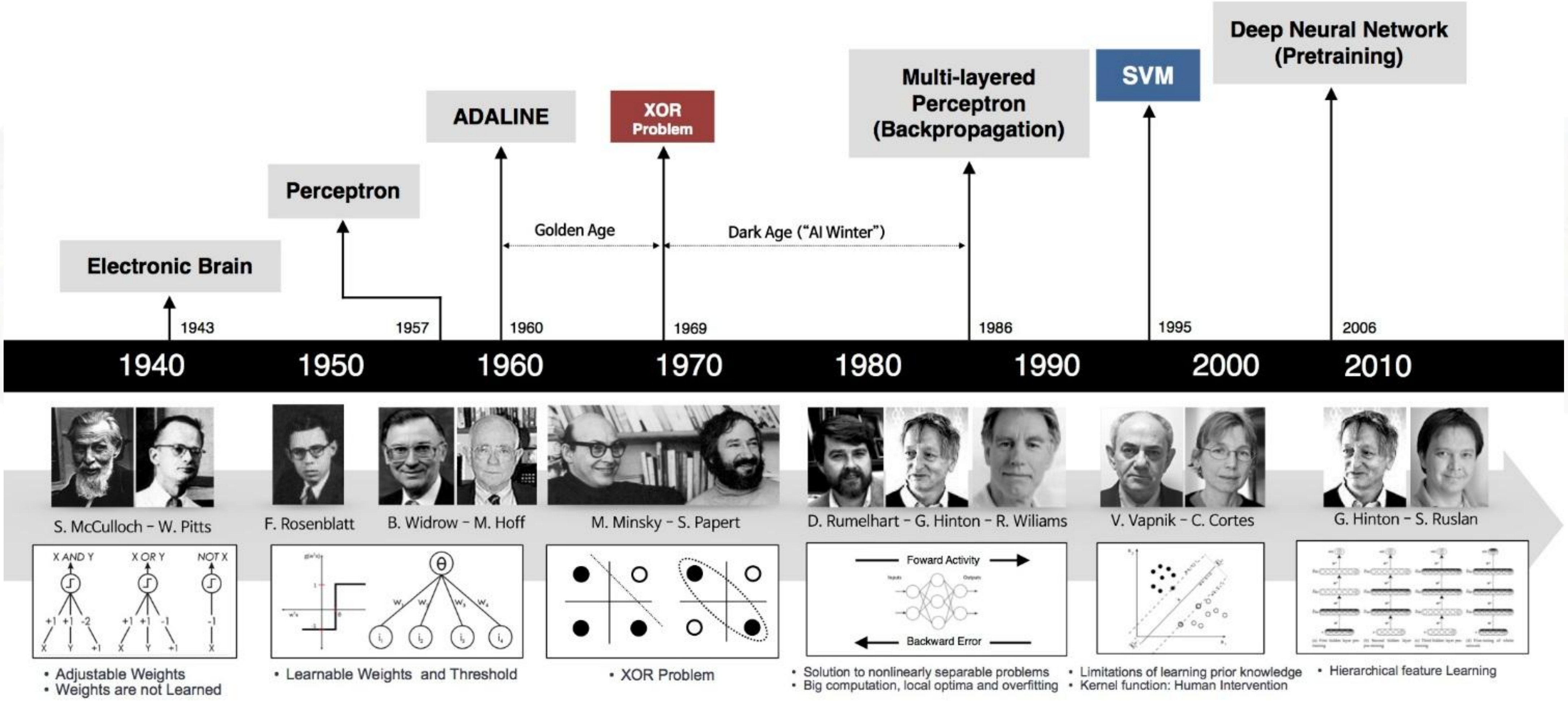
Superscript corresponds to the layer

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

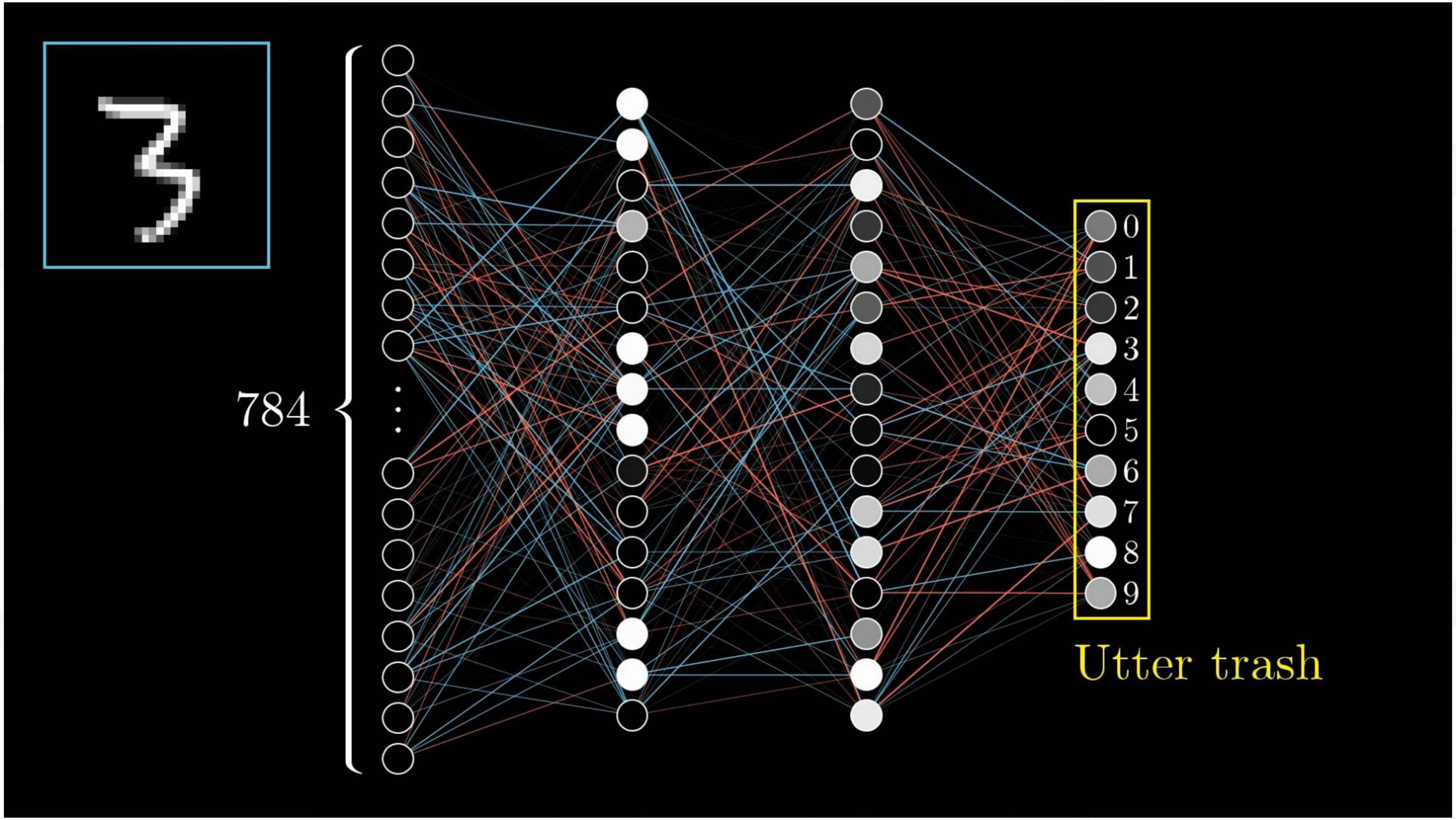
Subscript corresponds to a neuron in the layer



Backpropagation

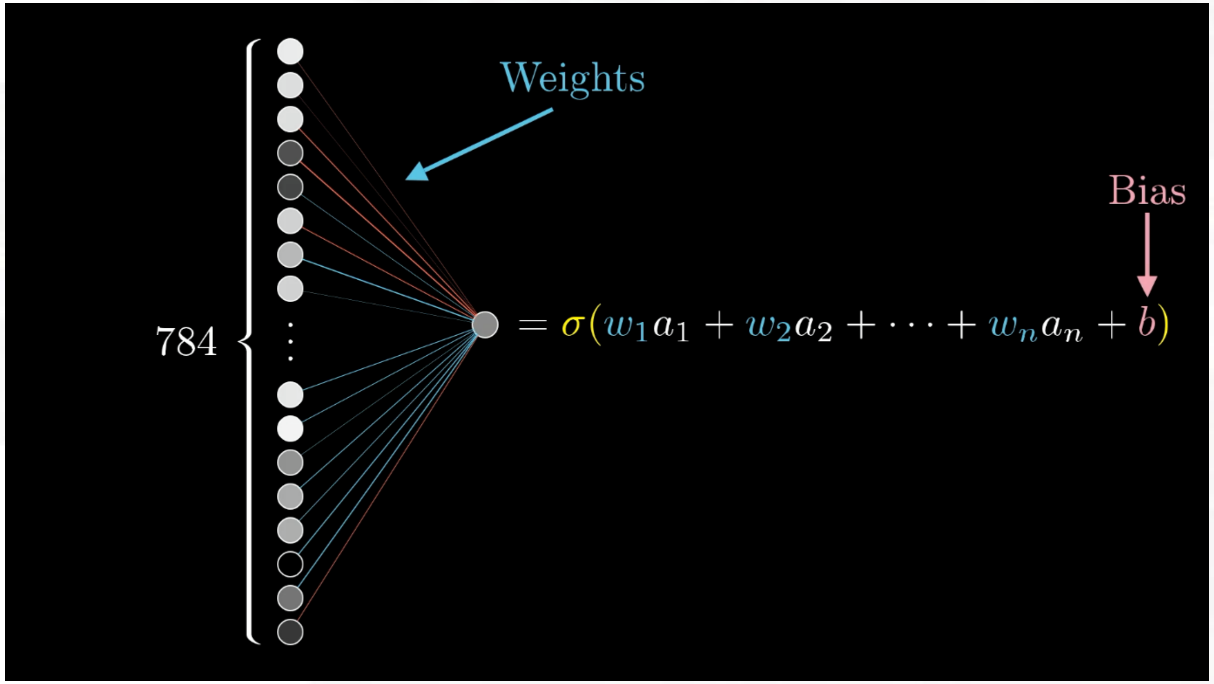


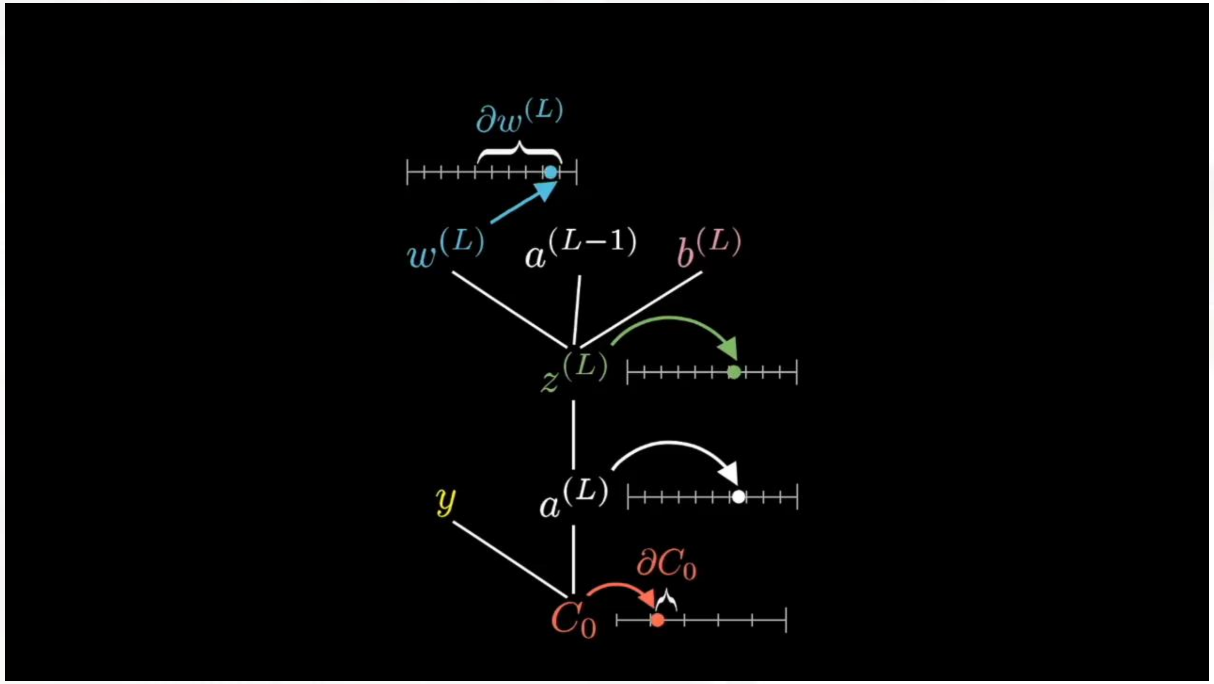
Optimization



Optimization

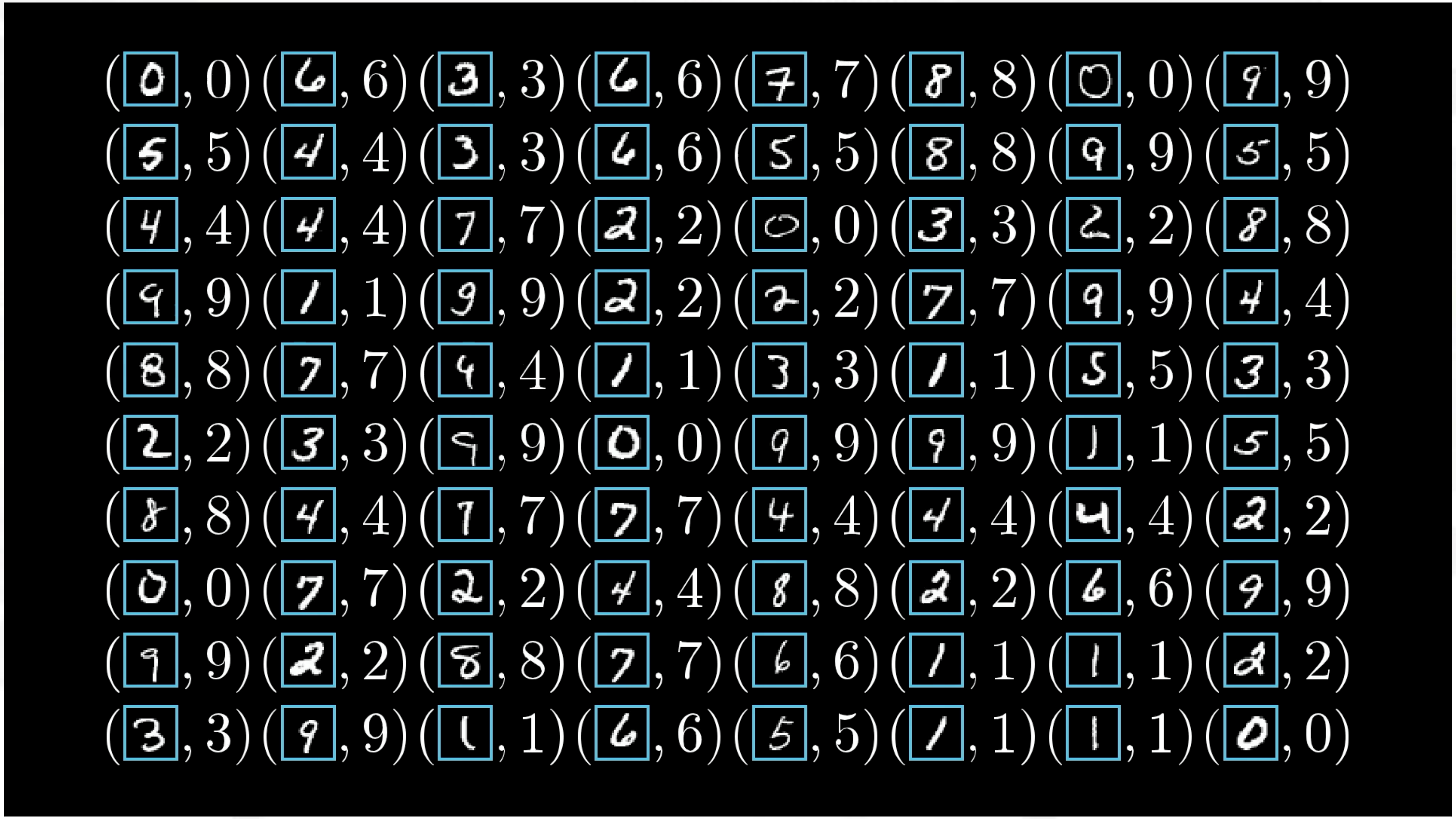
- The behavior of the network depends on its weights and biases.



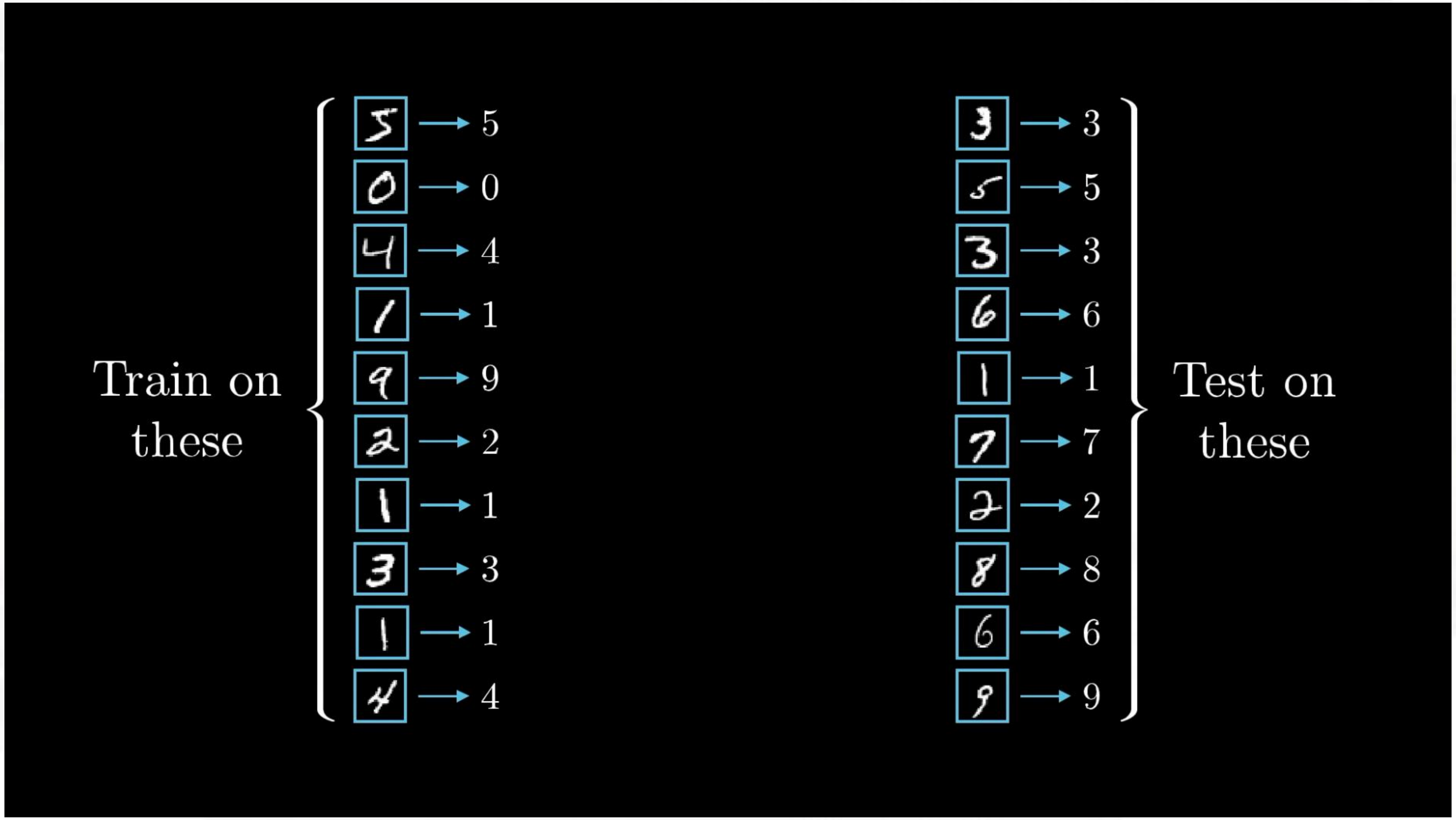


Optimization

Data Preparation



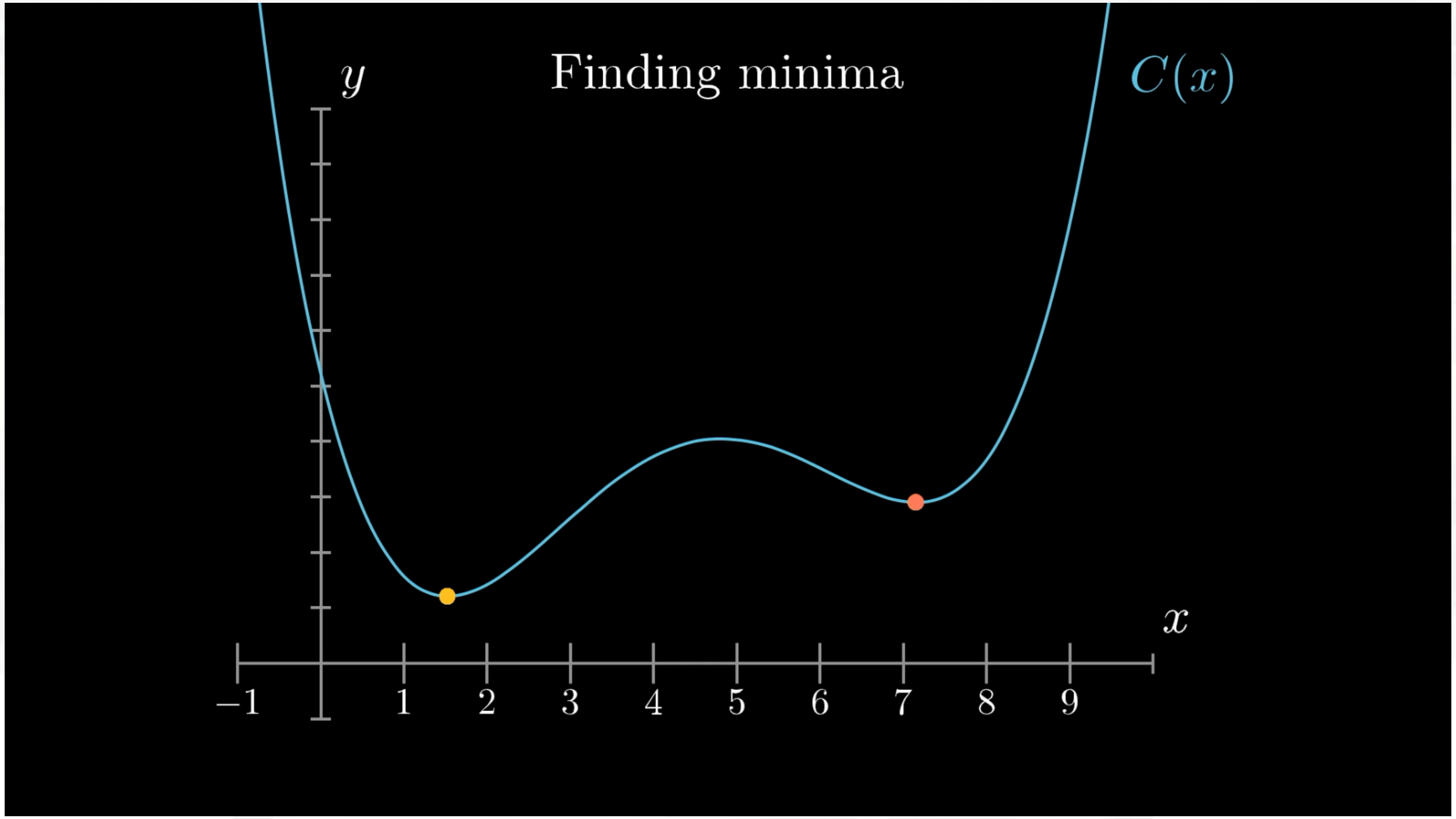
Optimization



Optimization

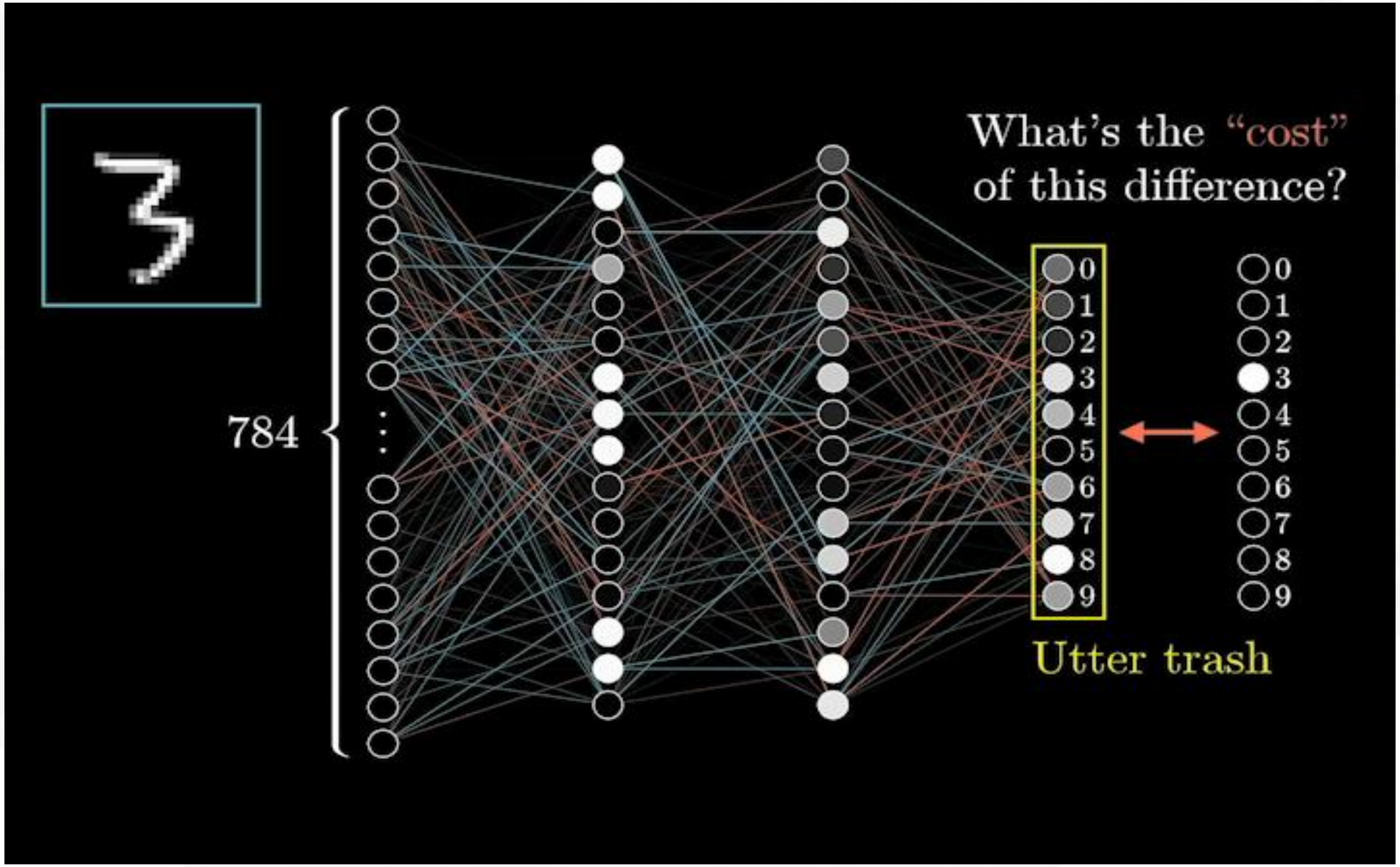
浙江大学 ZHEJIANG UNIVERSITY

- finding the minimum of a specific **function**

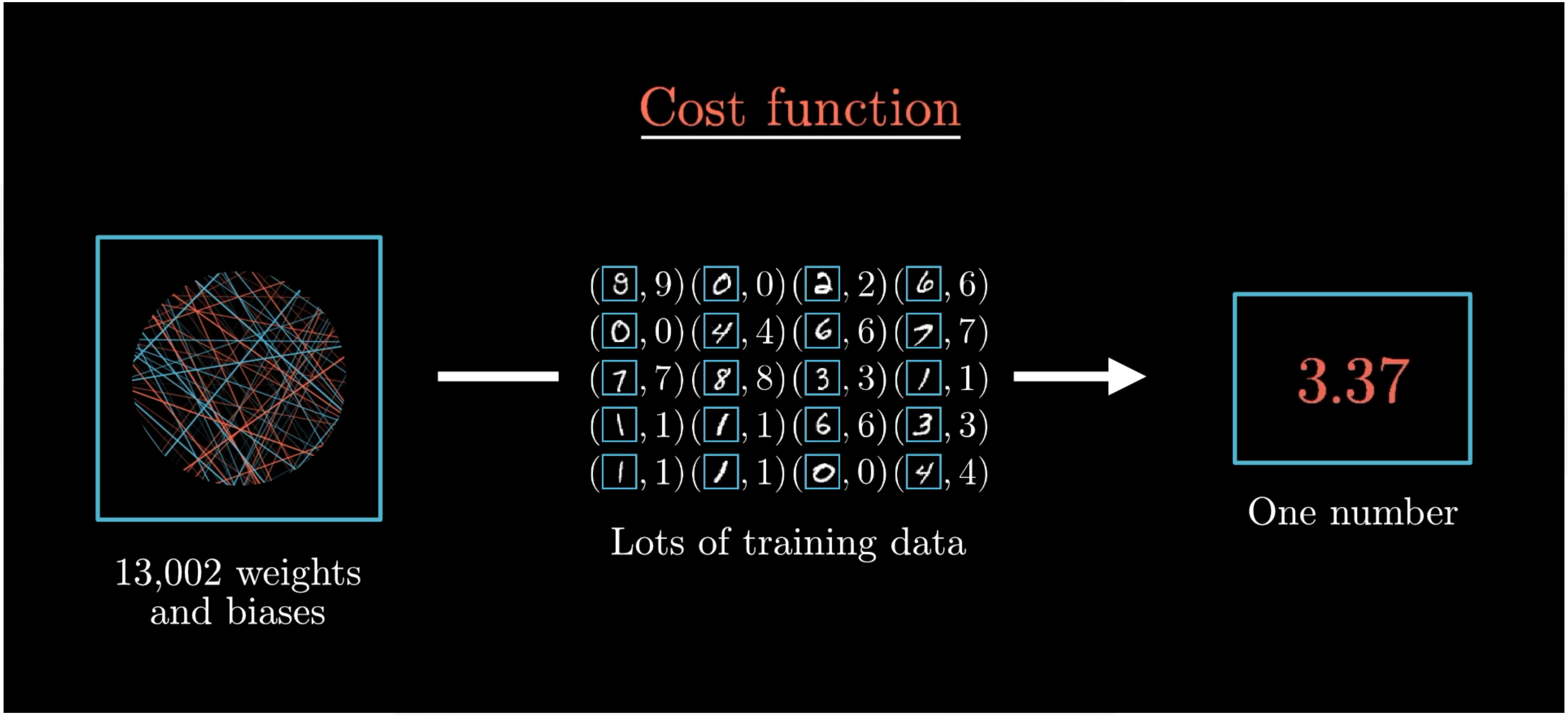


Optimization

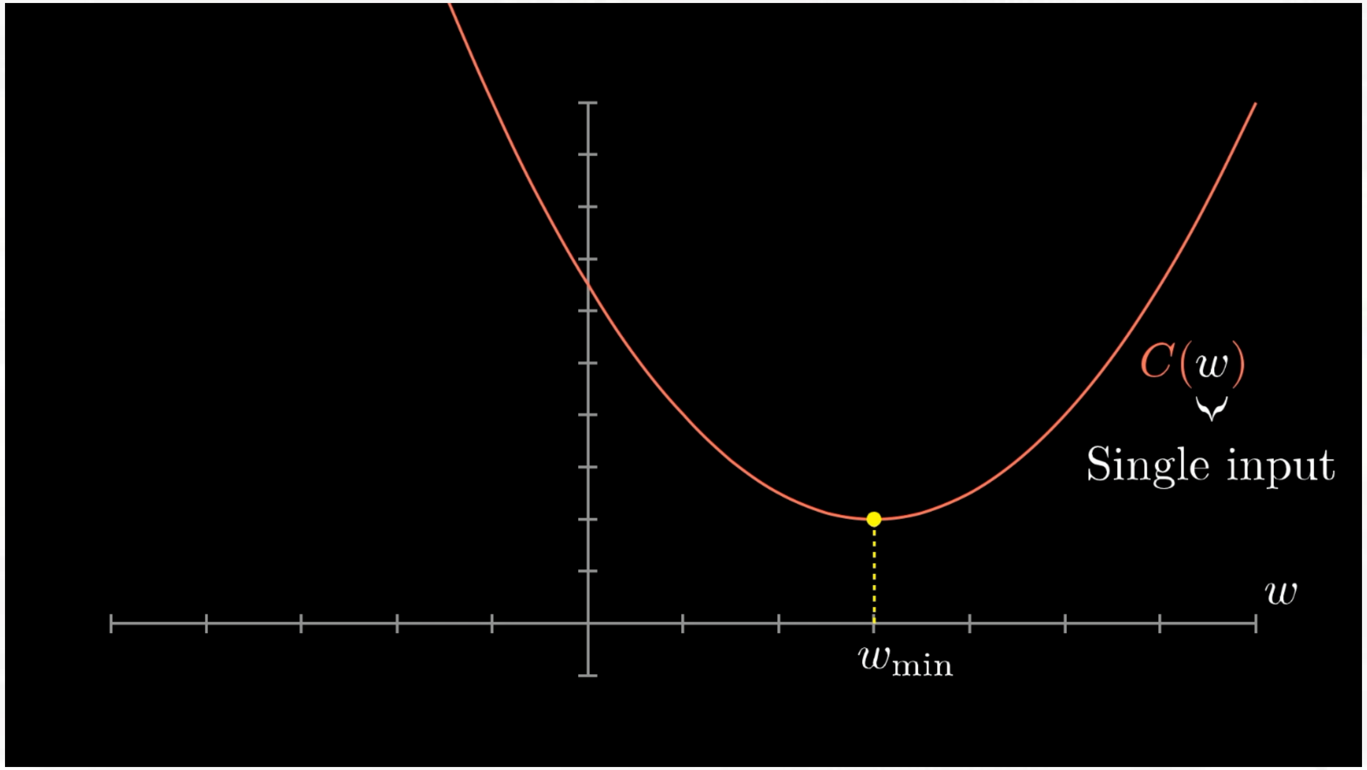
■ The Cost Function



Optimization

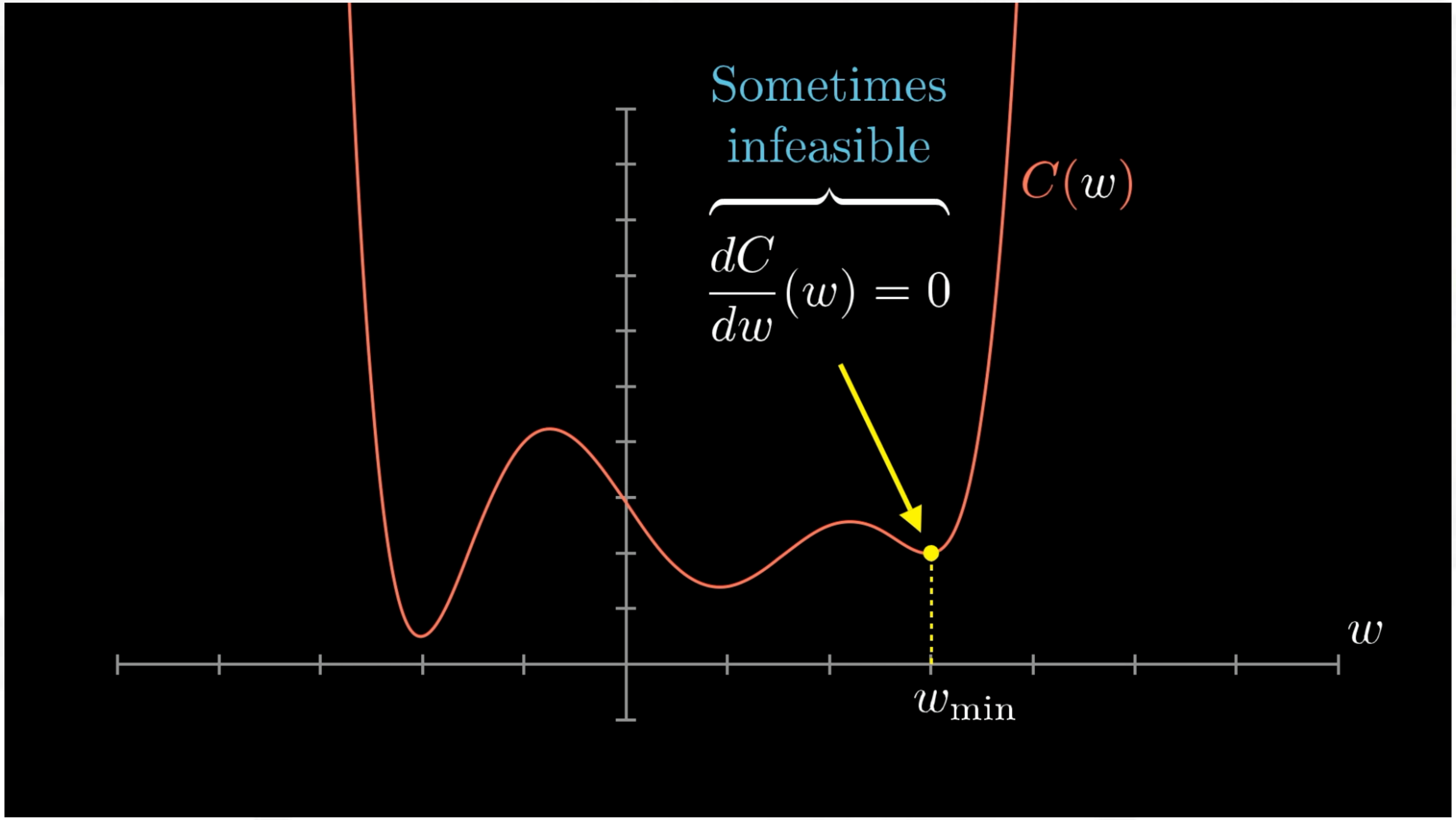


Optimization



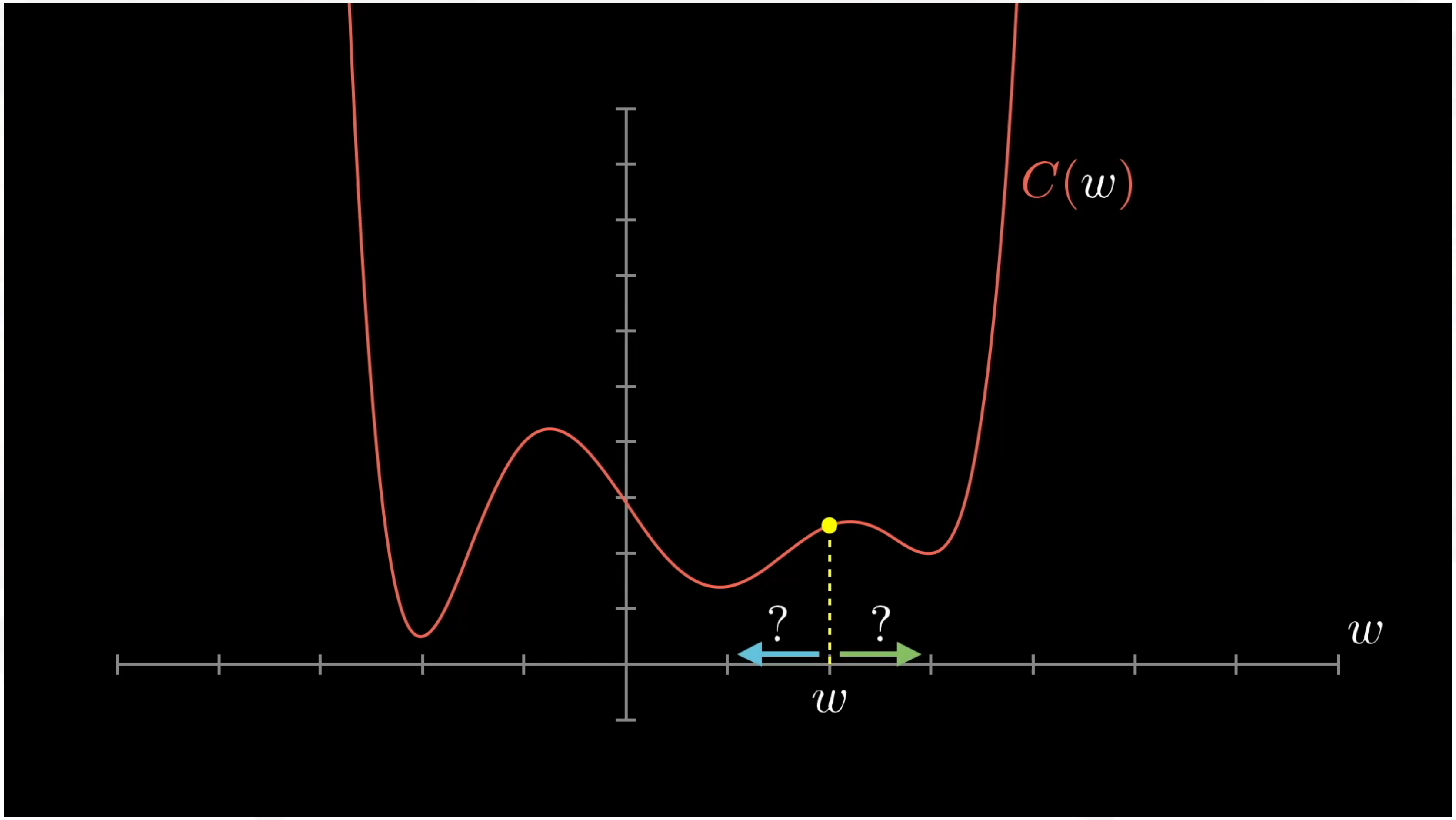
Optimization

- For a complicated cost function, computing the exact minimum directly isn't going to work.



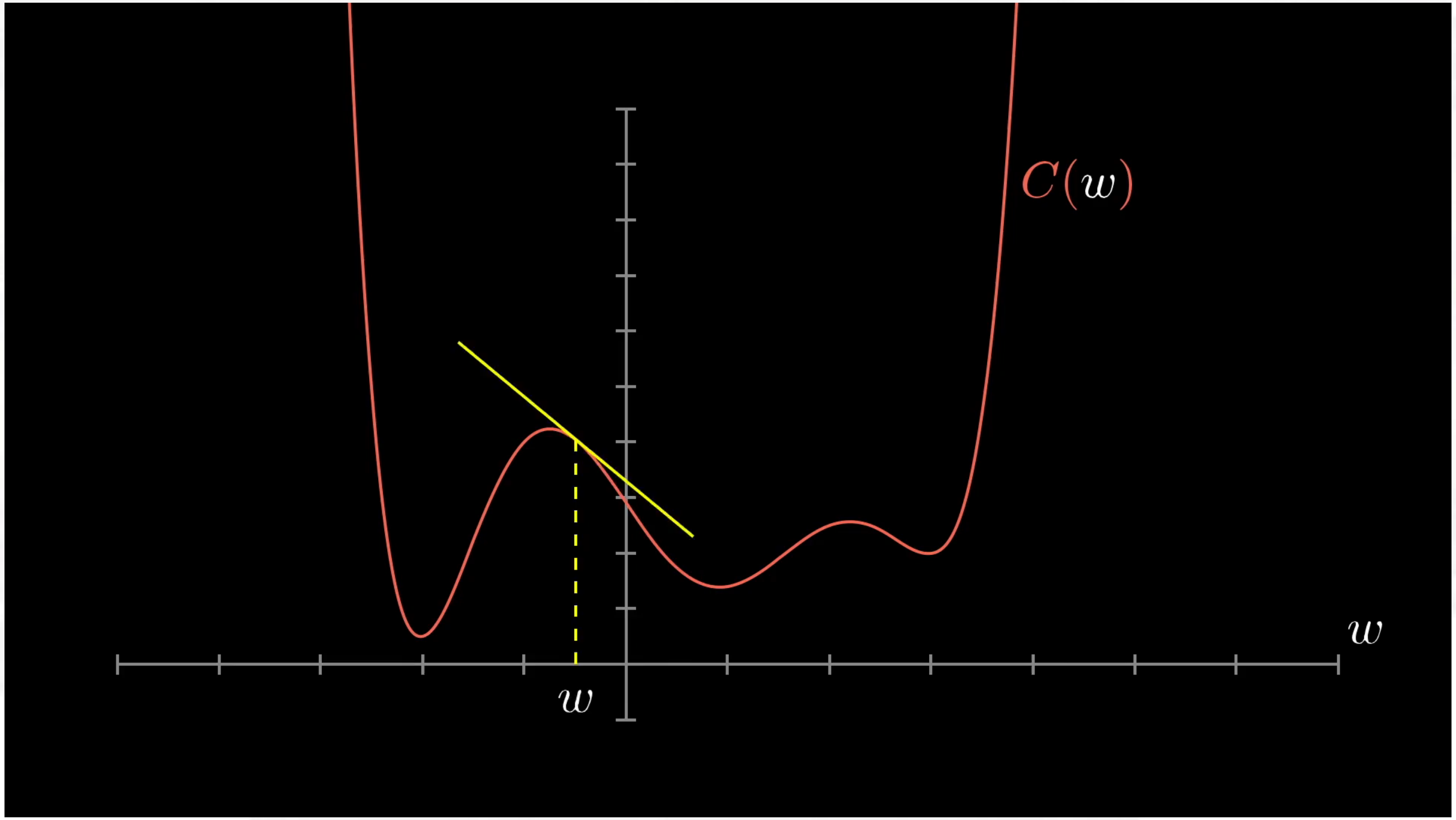
Optimization

- By following the slope (moving in the downhill direction), we approach a local minimum.



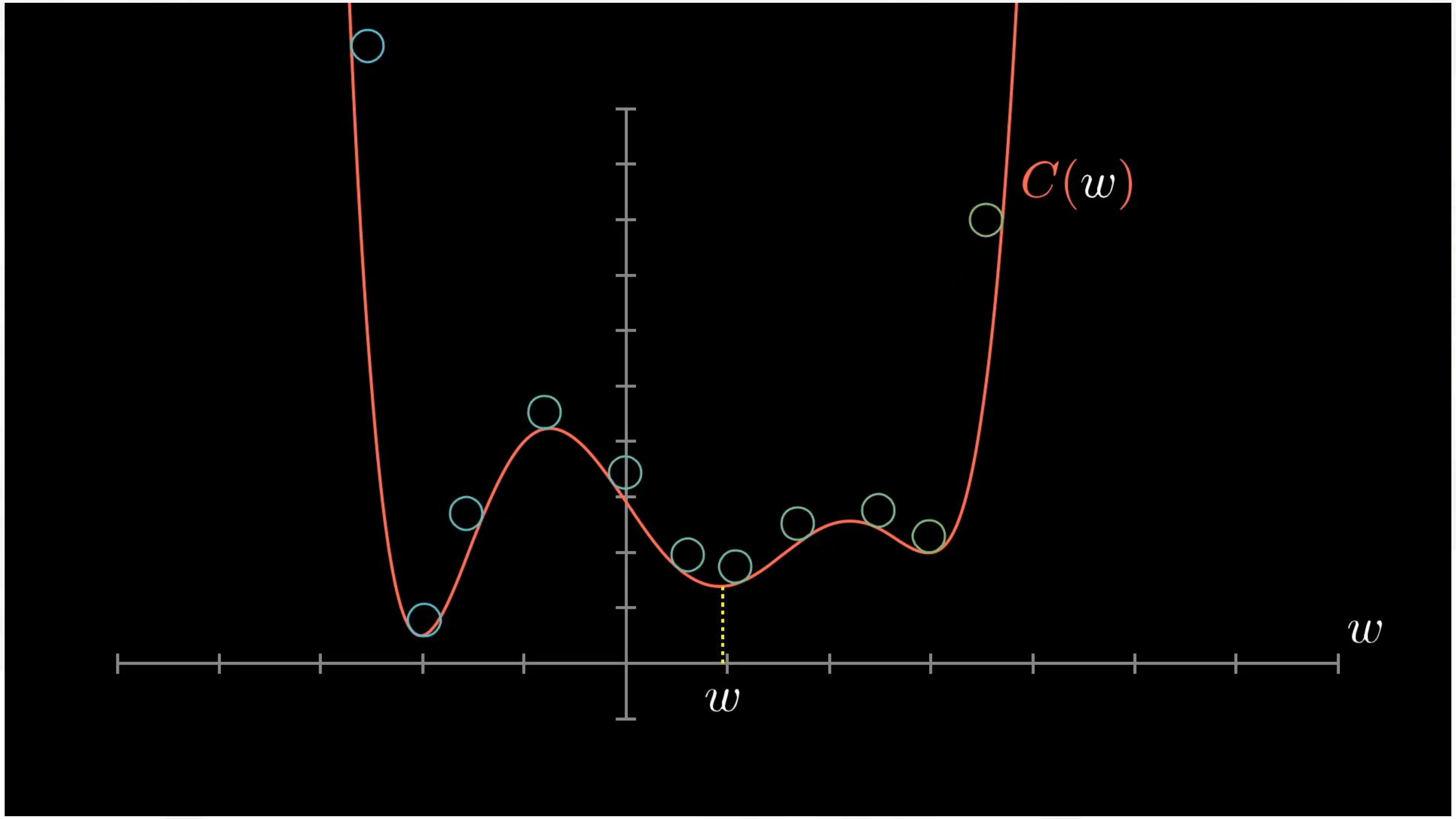
Optimization

- As the slope gets shallower, take smaller steps to avoid overshooting the minimum.



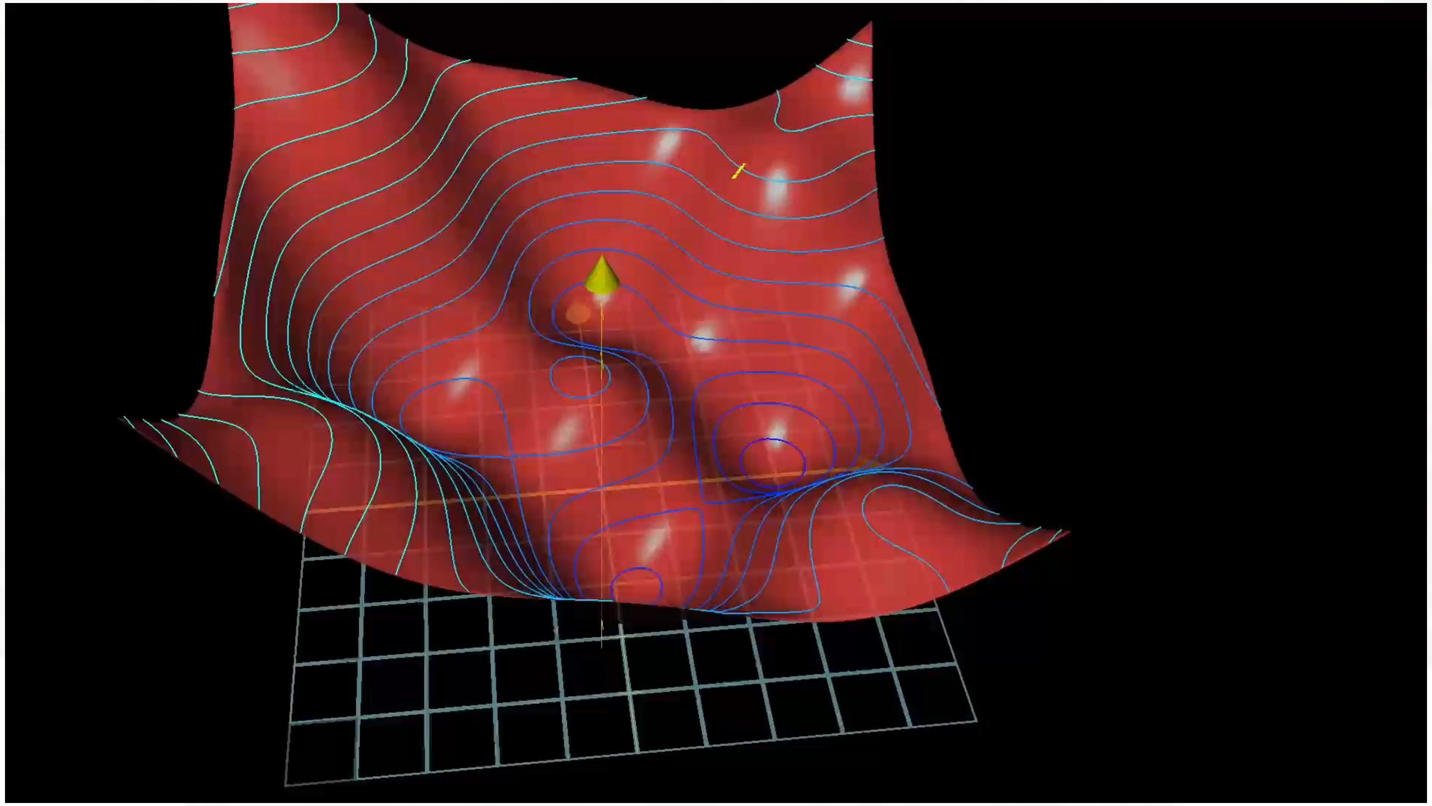
Optimization

- Moving the input position according to the slope is a lot like a ball rolling down a hill.



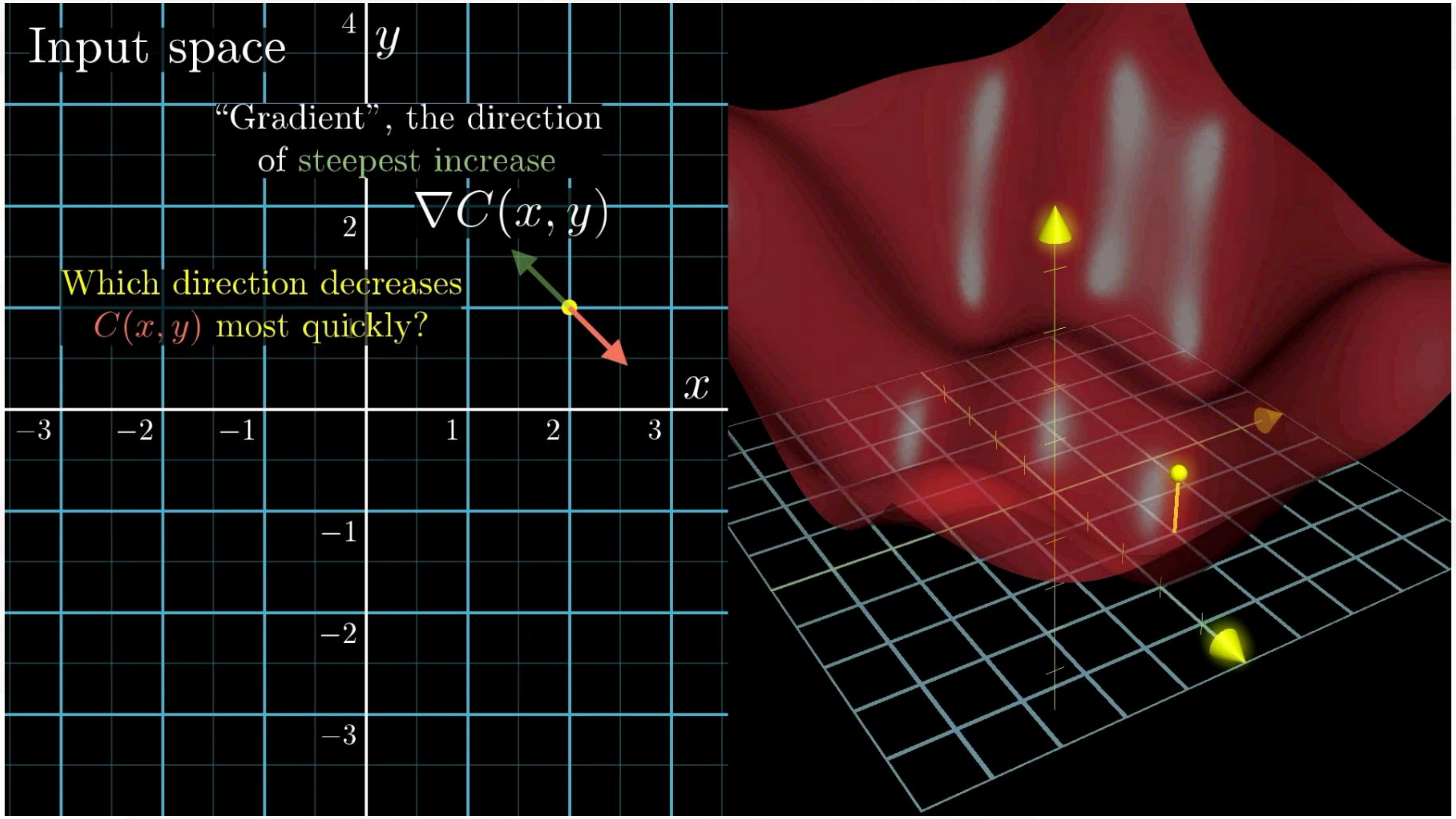
Optimization

- We can imagine minimizing a function that takes two inputs
- Gradient descent just means walking in the downhill direction to minimize the cost function.



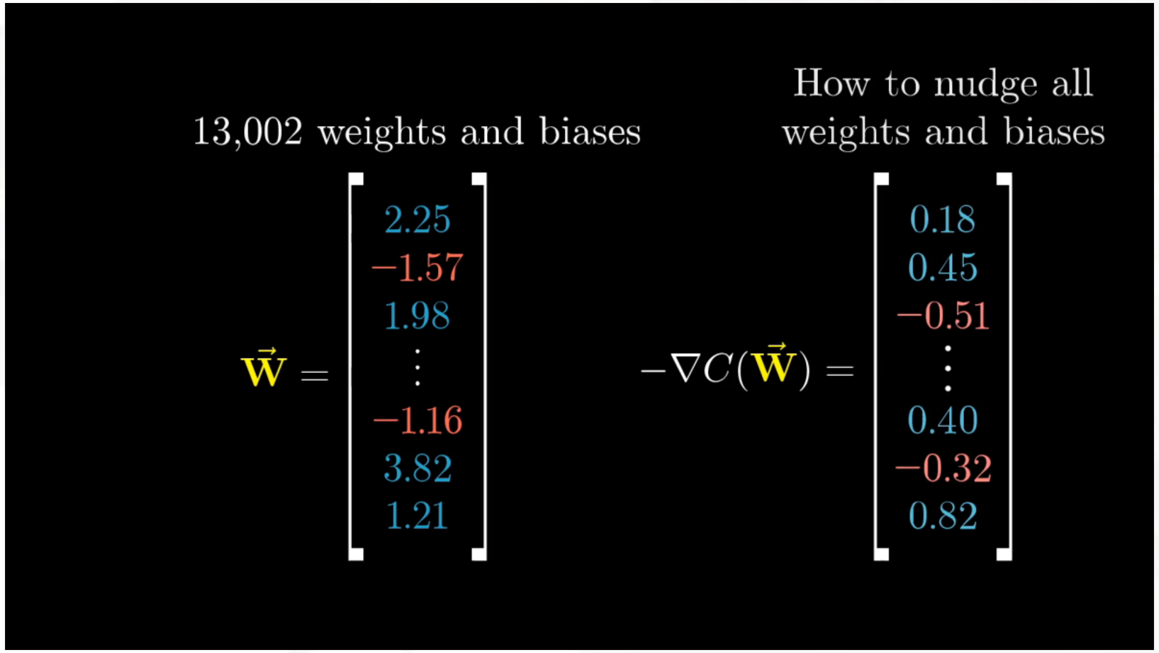
Optimization

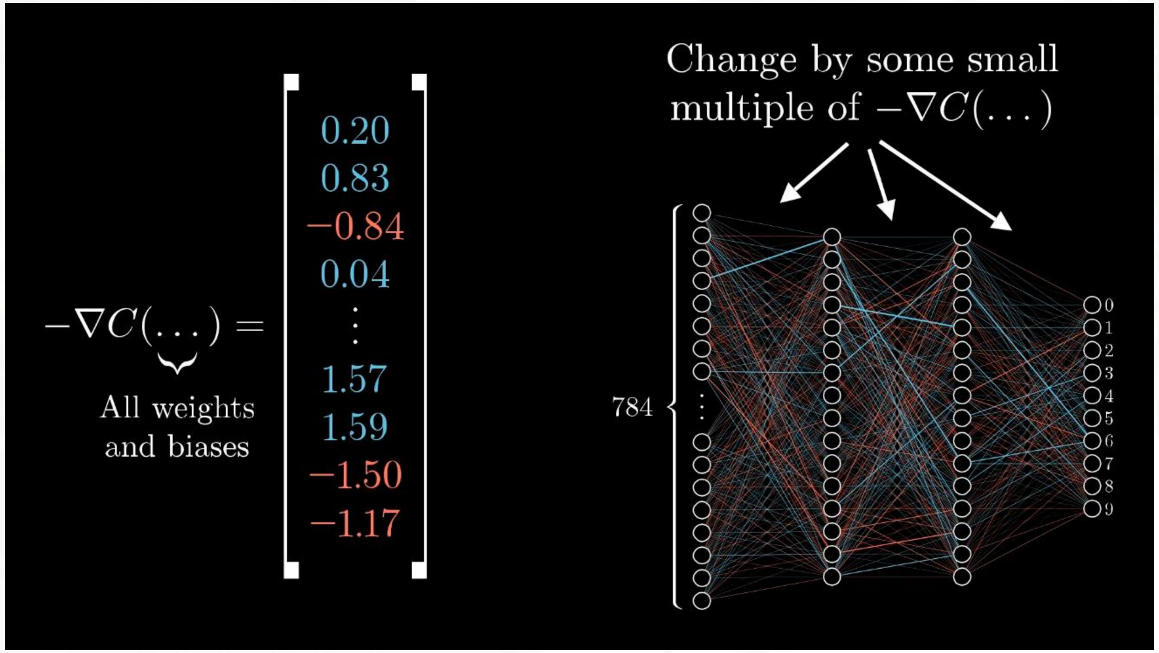
- The gradient, ∇C∇C, gives the uphill direction, so the negative of the gradient, −∇C−∇C, gives the downhill direction.



Optimization

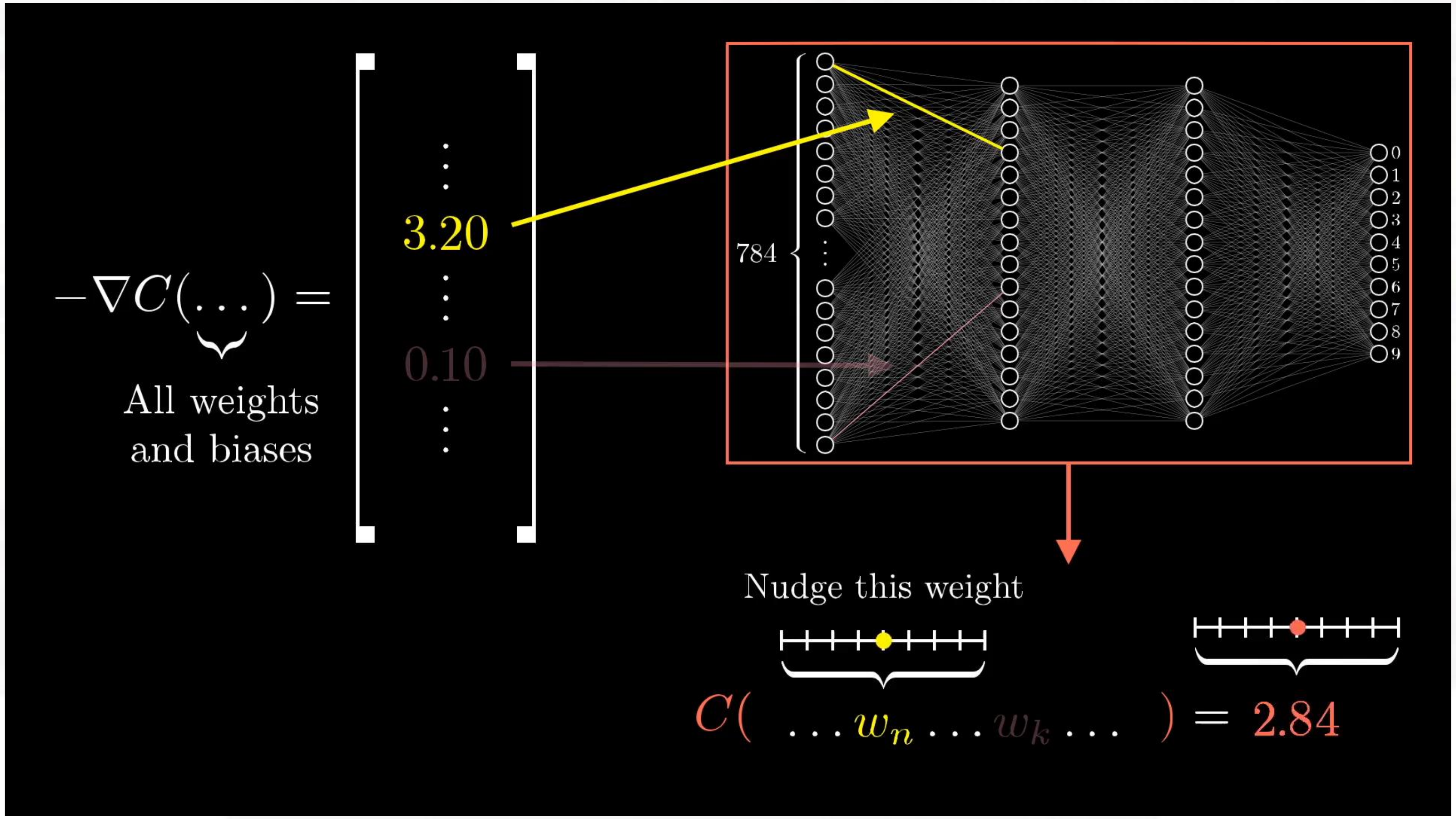
■ Another Way to Think About The Gradient





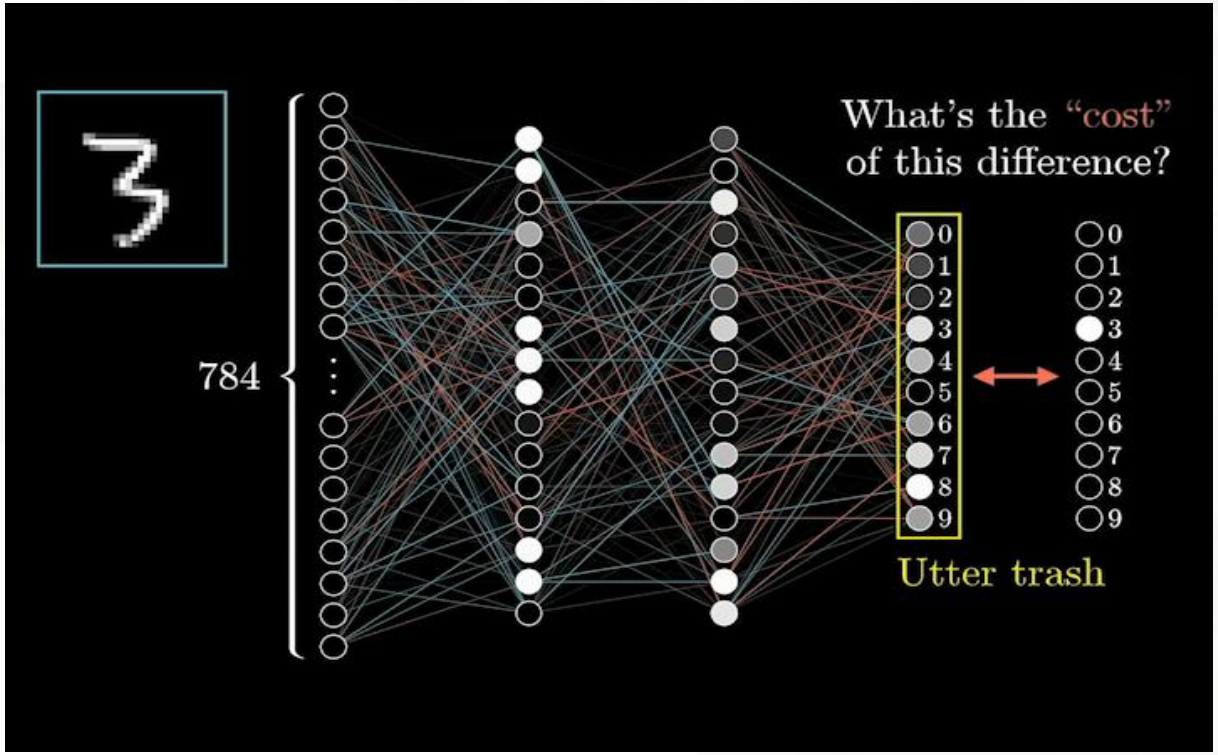
Optimization

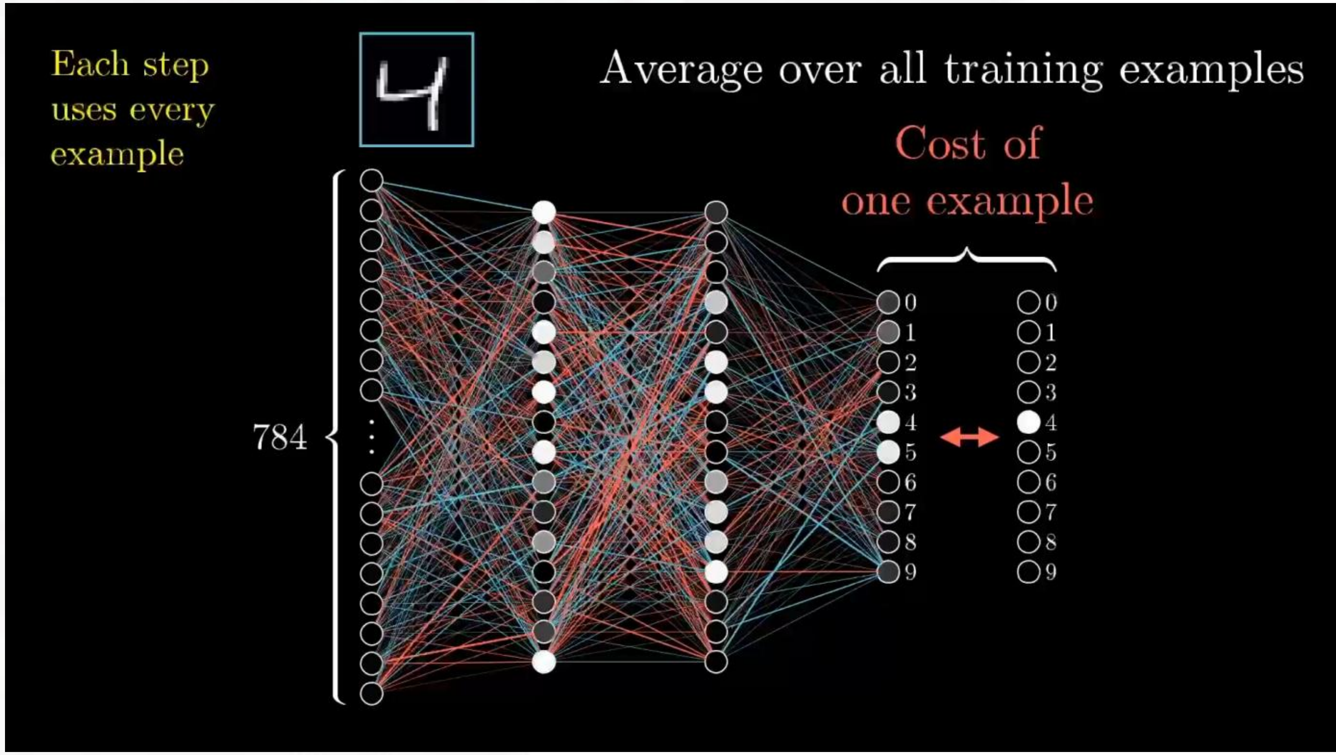
- Changing a weight that has a larger magnitude in the negative gradient vector has a bigger effect on the cost.



Forward and Backward

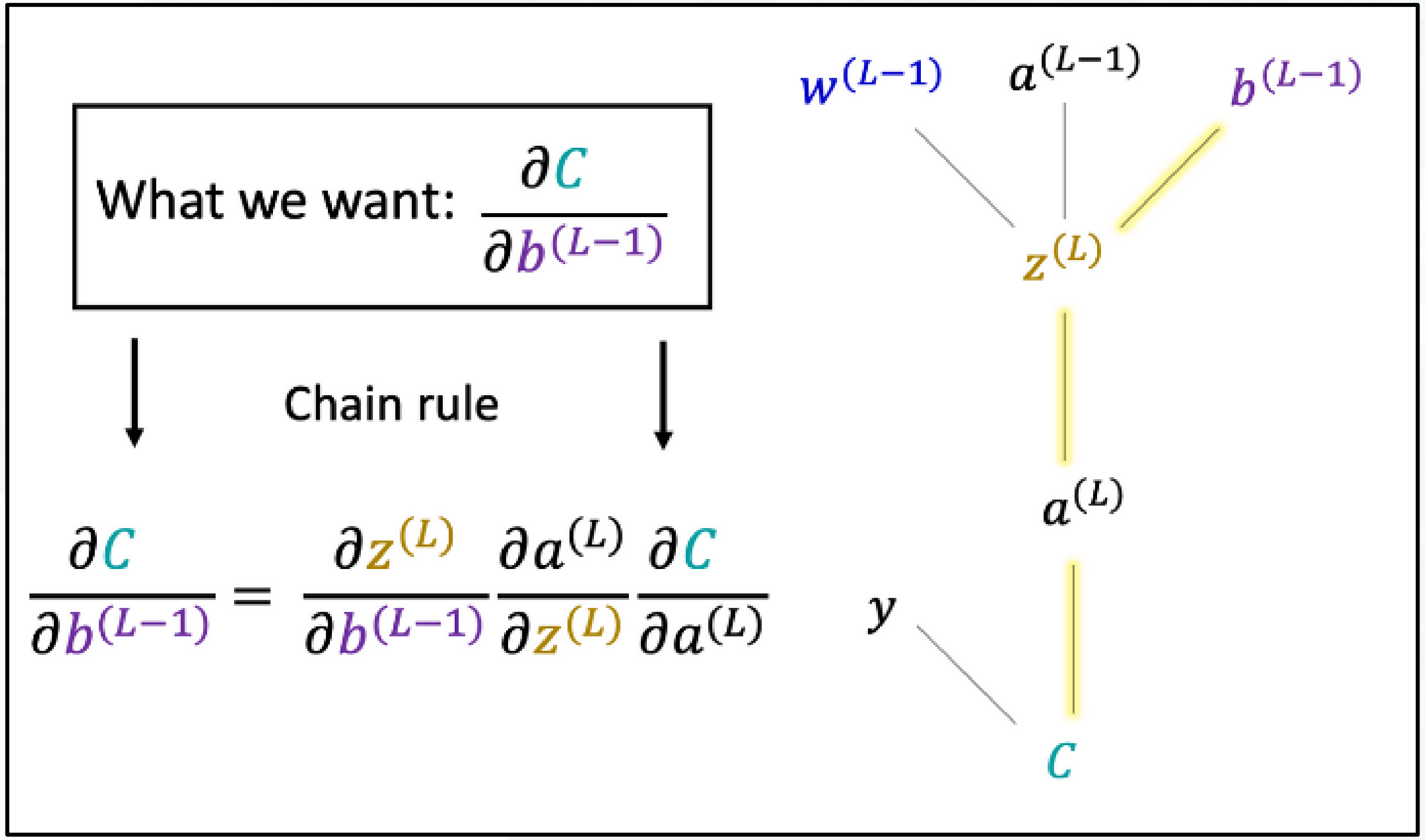
- Calculate the prediction error for training data
- Update model parameters based on the error

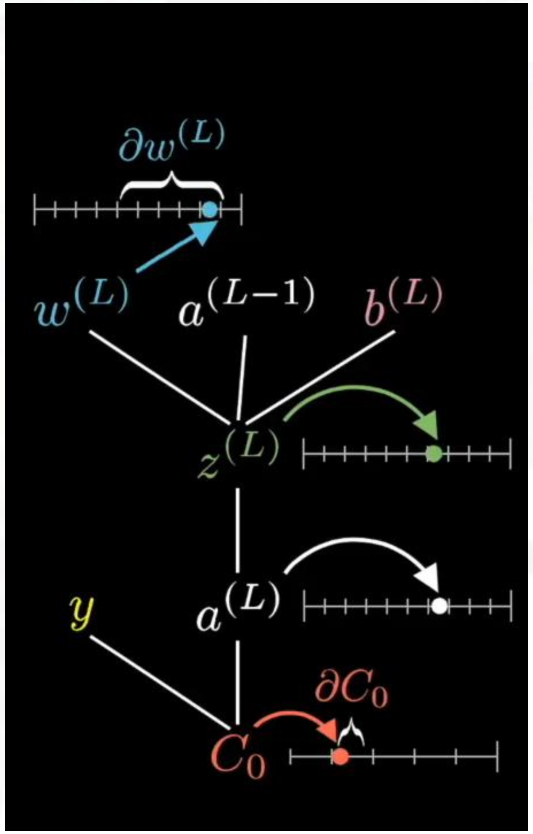




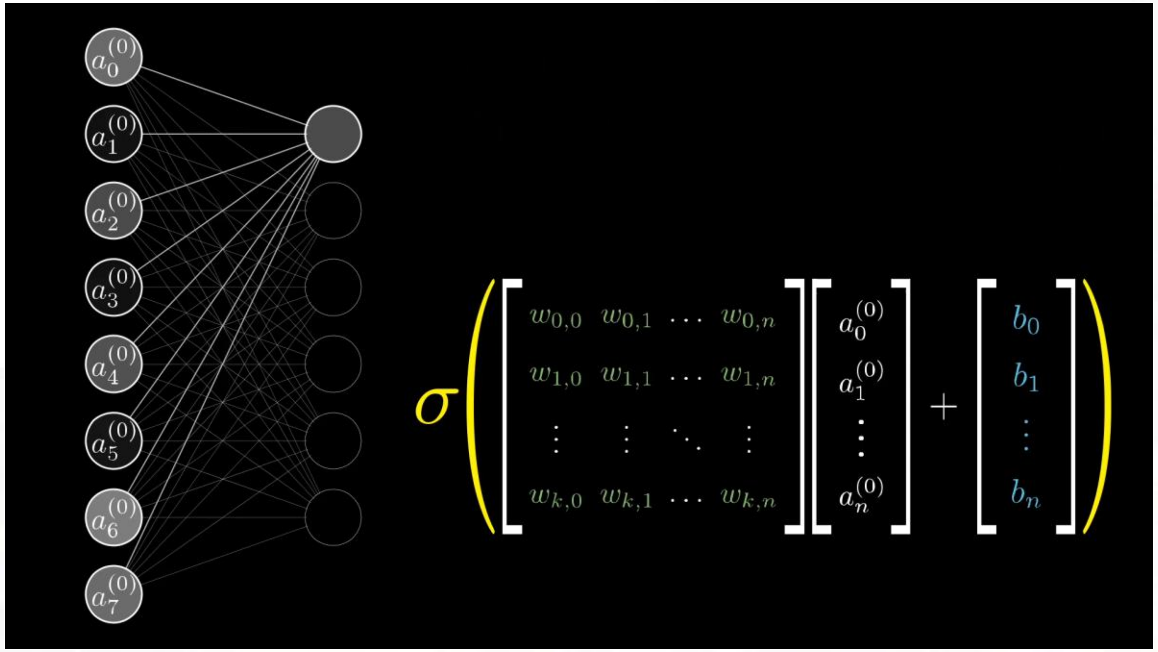
Backpropagation

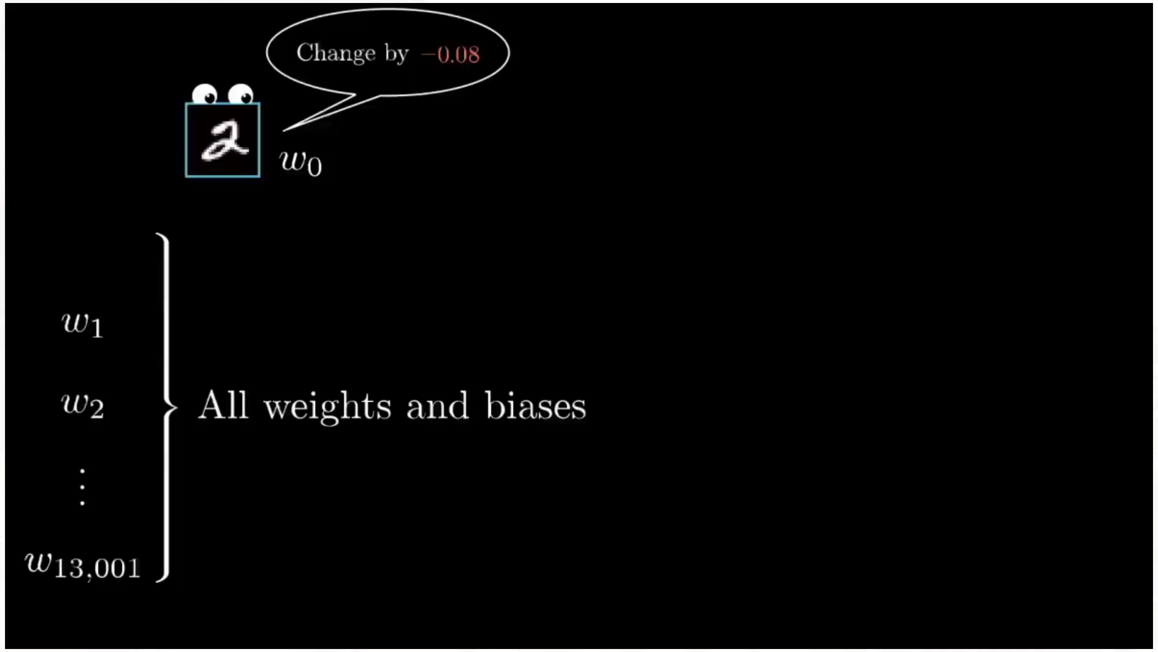
- Approximate partial derivatives based on the chain rule
- Solving neural network parameters based on gradient descent





Chain Rule





- Matrix operations form
- sample-wise and parameter-wise differentiation

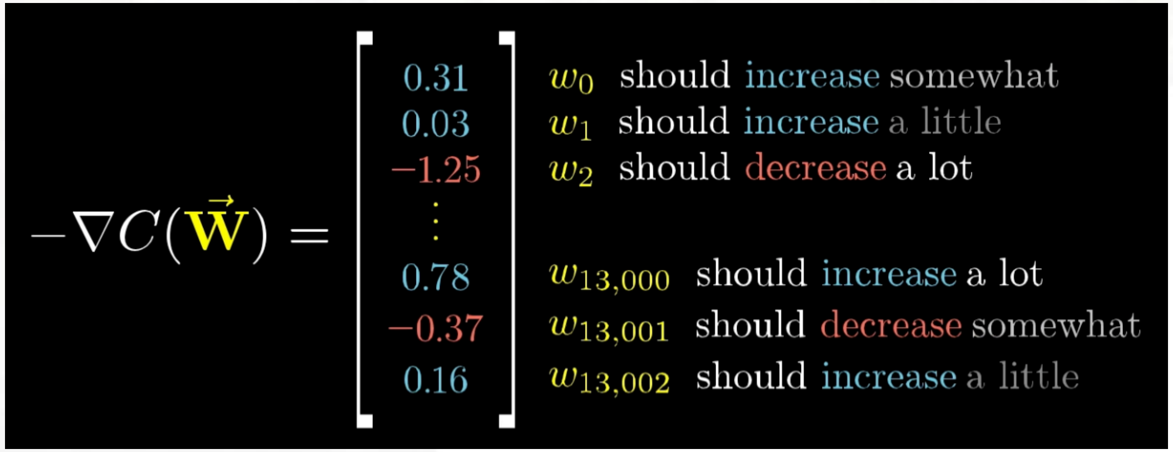
$$\frac{\partial C_0}{\partial w^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

$$C_0 = \left(a^{(L)} - y\right)^2 \longrightarrow \frac{\partial C_0}{\partial a^{(L)}} = 2\left(a^{(L)} - y\right)$$

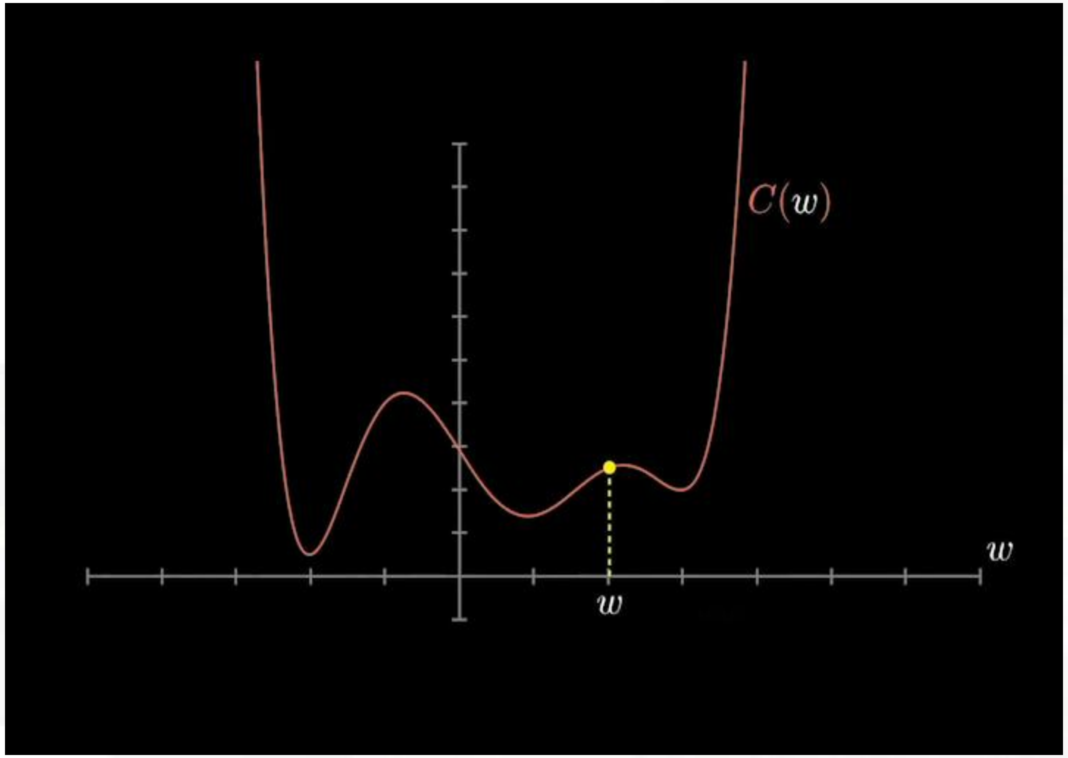
$$a^{(L)} = \sigma\left(z^{(L)}\right) \longrightarrow \frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'\left(z^{(L)}\right)$$

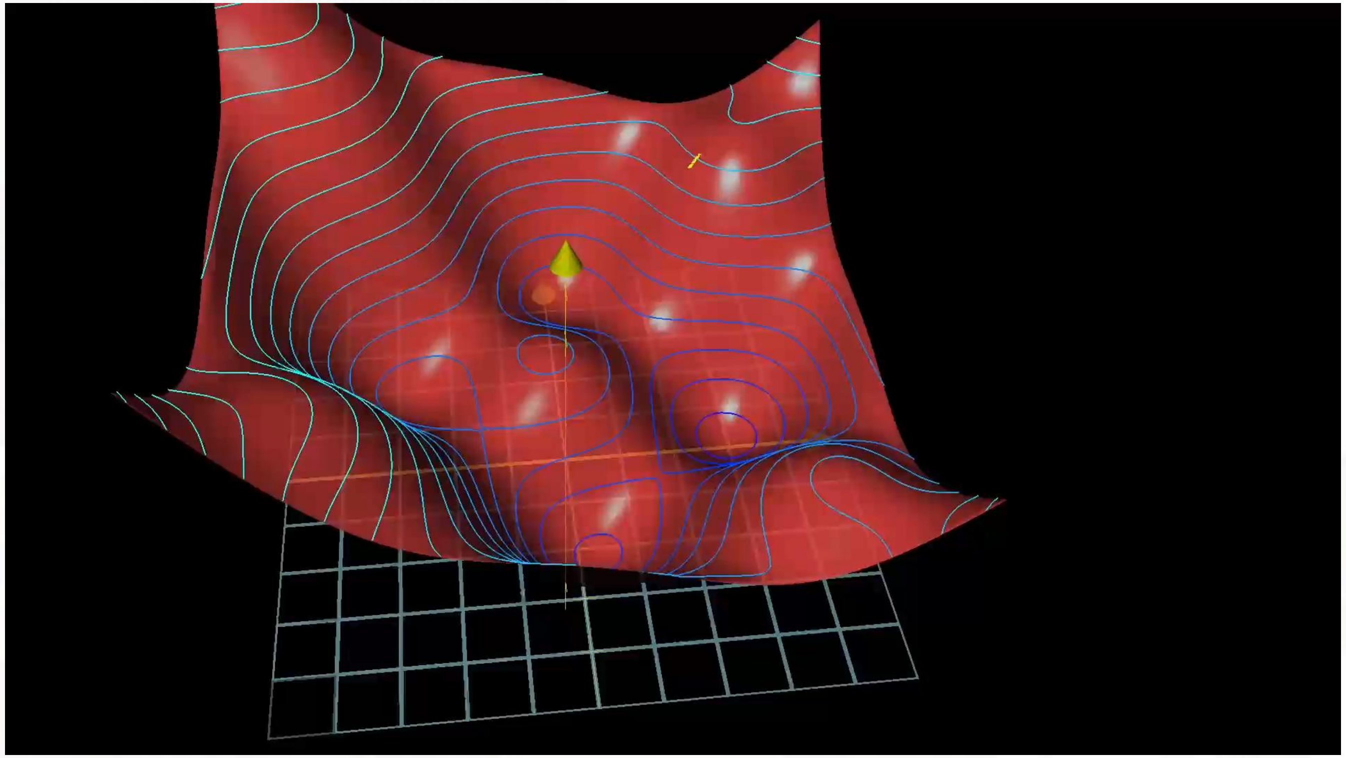
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)} \longrightarrow \frac{\partial z^{(L)}}{\partial a^{(L-1)}} = w^{(L)}$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)} \longrightarrow \frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$



Gradient Descent





Algorithm

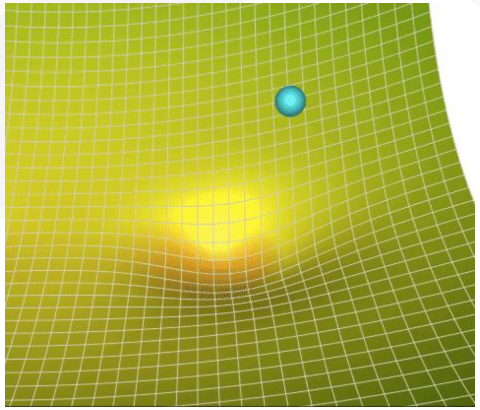
- Calculate ∇C based on chain rules
- Move all parameters toward $-\nabla C$ slightly
- Repeat all the steps above

- (randomly) select θ_0
- get gradient $g_0 = \nabla L_1(\theta_0)$

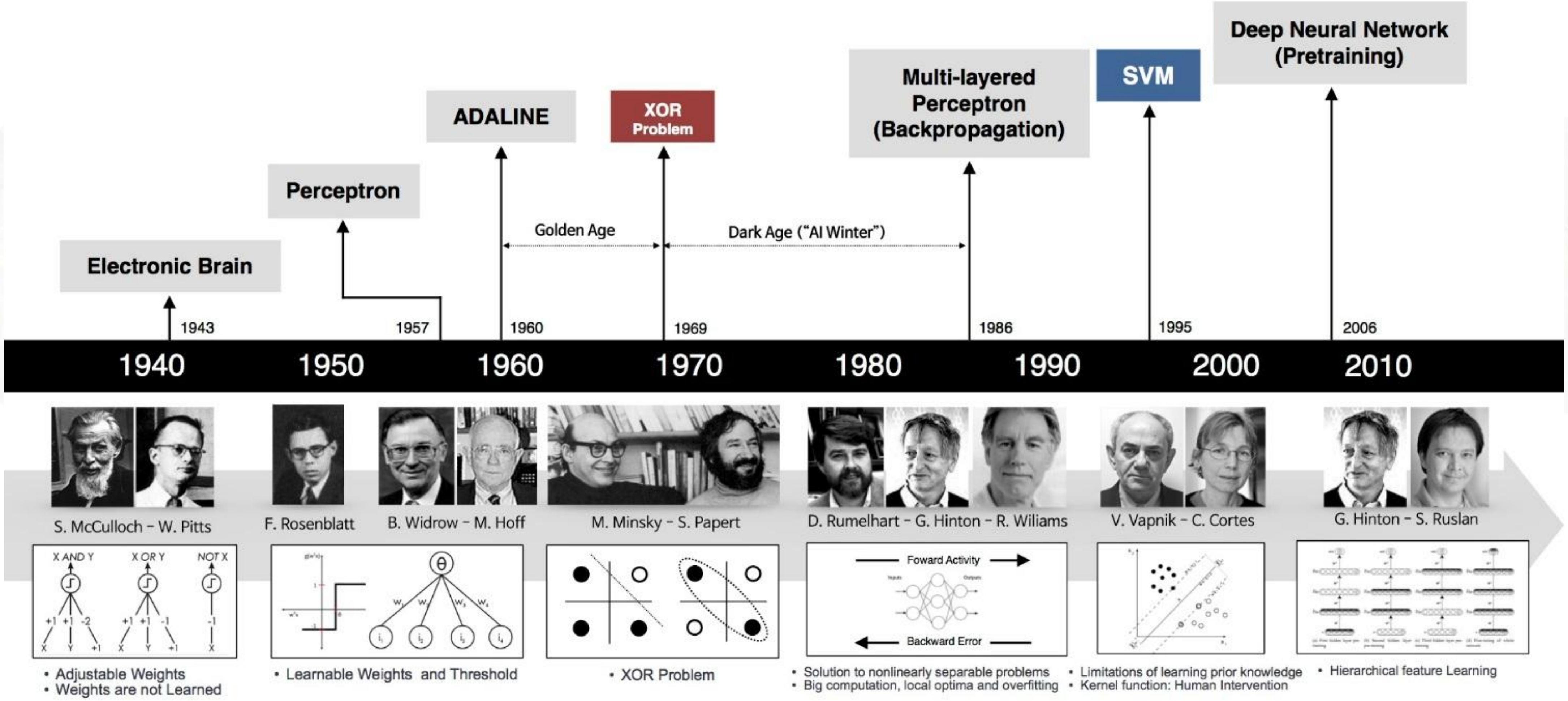
  update $\theta_1 \leftarrow \theta_0 - \eta g_0$
- get gradient $g_1 = \nabla L_2(\theta_1)$

  update $\theta_2 \leftarrow \theta_1 - \eta g_1$
- get gradient $g_2 = \nabla L_3(\theta_2)$

  update $\theta_3 \leftarrow \theta_2 - \eta g_2$



The Problem of Backpropagation



The History of Backpropagation

- The time when the relevant ideas were proposed
  - 1986, the term was widely known
  - 1974, the first time used to train neural network
  - 1960, The basic knowledge of backpropagation has been accepted and widely used.
  - 2006, Introduction of pre-training and fine-tuning mechanisms
- Issues During Training
  - Gradient vanishing
  - Gradient exploding

Published: 09 October 1986

Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536 (1986) | Cite this article

110k Accesses | **14712** Citations | **378** Altmetric | Metrics

Gradient Theory of Optimal Flight Paths

HENRY J. KELLEY[1]

Grumman Aircraft Engineering Corp.
Bethpage, N. Y.

An analytical development of flight performance optimization according to the method of gradients or "method of steepest descent" is presented. Construction of a minimizing sequence of flight paths by a stepwise process of descent along the local gradient direction is described as a computational scheme. Numerical application of the technique is illustrated in a simple example of orbital transfer via solar sail propulsion. Successive approximations to minimum time planar flight paths from Earth's orbit to the orbit of Mars are presented for cases corresponding to free and fixed boundary conditions on terminal velocity components.

Gradient Vanishing and Exploding

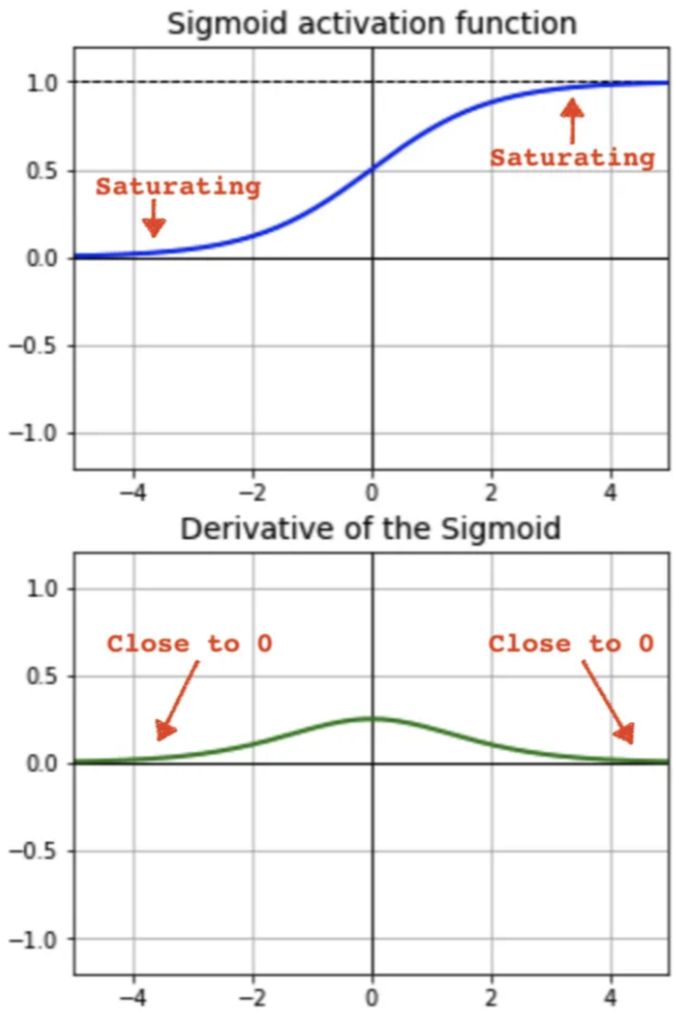
Gradient Vanishing	Gradient Exploding
The weights are almost 0	Weights with NaN value
The weights near the output layer update quickly, while the weights in the input layer hardly update.	Weights increase explosively
Converges very slowly	Instable performance
Stop learning	Performs poorly on the training set

Gradient Vanishing and Exploding

$$\frac{\partial C_0}{\partial w^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

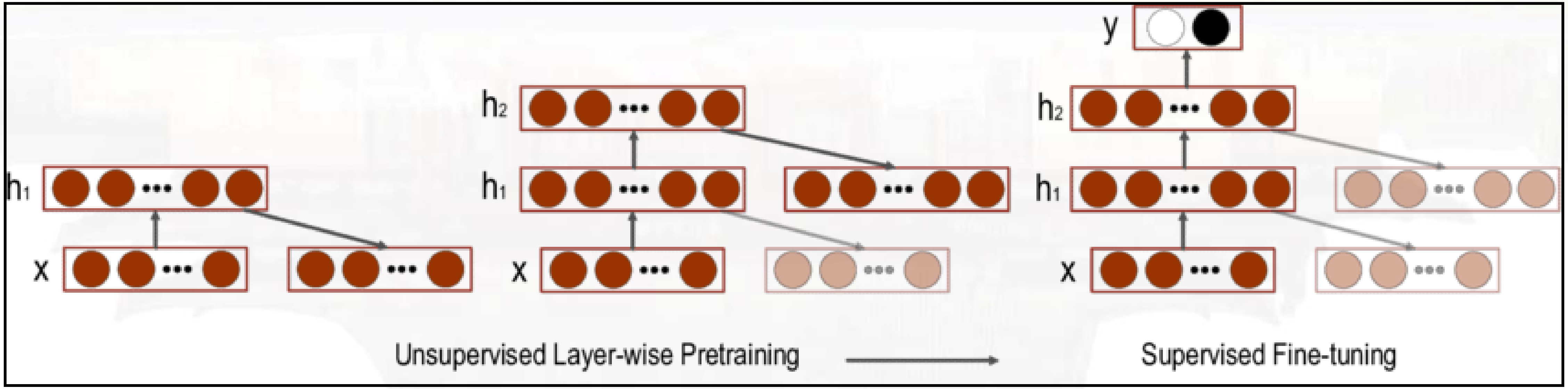
$$a^{(L)} = \sigma\left(z^{(L)}\right) \longrightarrow \frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'\left(z^{(L)}\right)$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)} \longrightarrow \frac{\partial z^{(L)}}{\partial a^{(L-1)}} = w^{(L)}$$



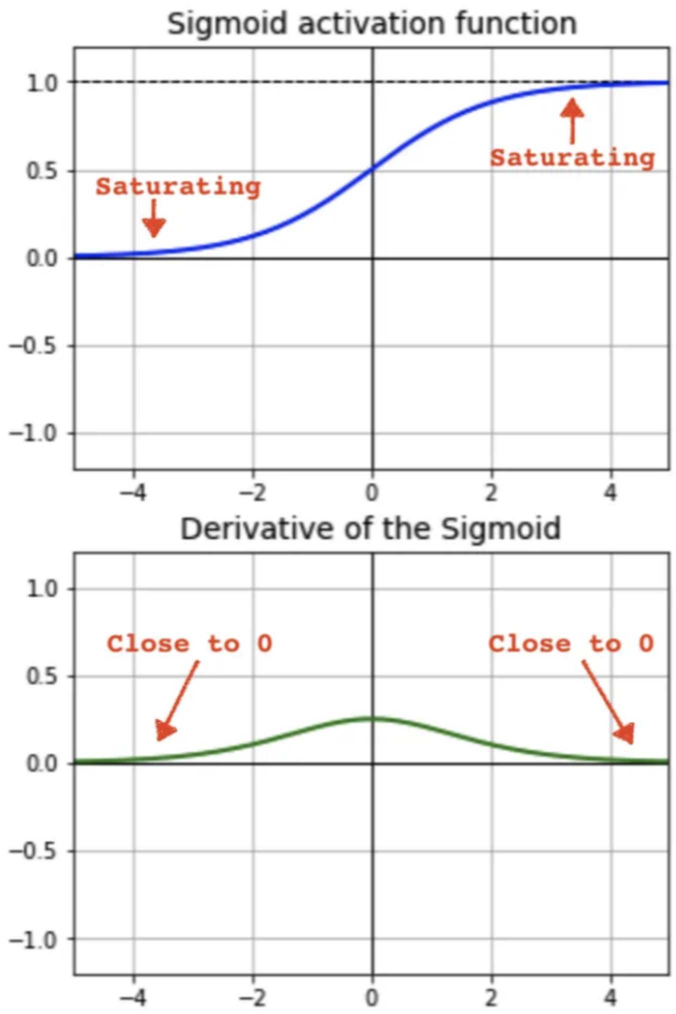
Gradient Vanishing and Exploding

- Use greedy unsupervised learning to find a sensible set of weights one layer at a time. Then fine-tune with backpropagation.
  - The precious information in the labels is only used for the final fine-tuning.
  - We do not start backpropagation until we already have sensible weights that already do well at the task.

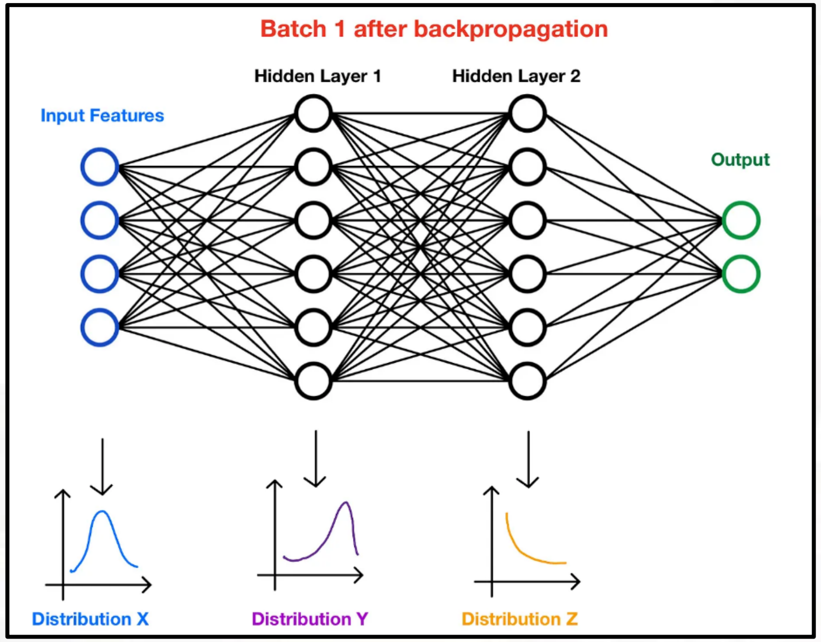


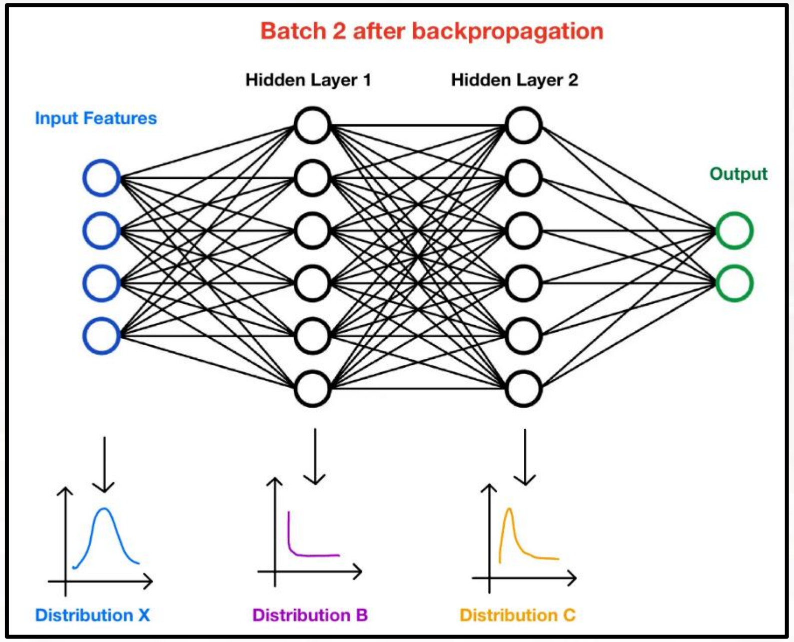
Gradient Vanishing and Exploding

1. *Zero Initialization*: Initialize all the weights and biases to zero. This is not generally used in deep learning as it leads to symmetry in the gradients, resulting in all the neurons learning the same feature.
2. *Random Initialization:* Initialize the weights and biases randomly from a uniform or normal distribution. This is the most common technique used in deep learning.
3. *Xavier Initialization:* Initialize the weights with a normal distribution with mean 0 and variance of sqrt(1/n), where n is the number of neurons in the previous layer. This is used for the sigmoid activation function.
4. *He Initialization:* Initialize the weights with a normal distribution with mean 0 and variance of sqrt(2/n), where n is the number of neurons in the previous layer. This is used for the ReLU activation function.
5. *Orthogonal Initialization:* Initialize the weights with an orthogonal matrix, which preserves the gradient norm during backpropagation.
6. *Uniform Initialization:* Initialize the weights with a uniform distribution. This is less commonly used than random initialization.
7. *Constant Initialization:* Initialize the weights and biases with a constant value. This is rarely used in deep learning.



Gradient Vanishing and Exploding



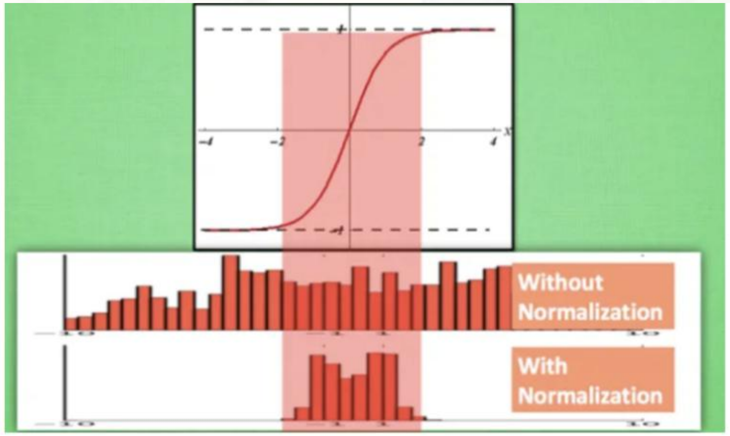


$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

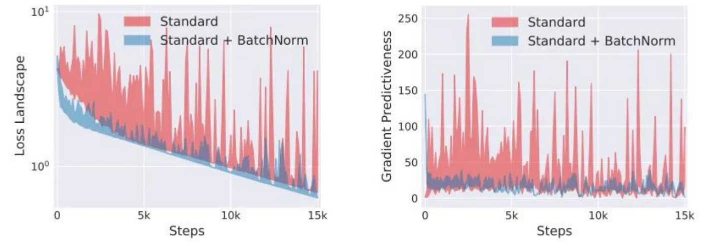
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



Internal Covariate Shift

- Slow Convergence
- Gradient Vanishing

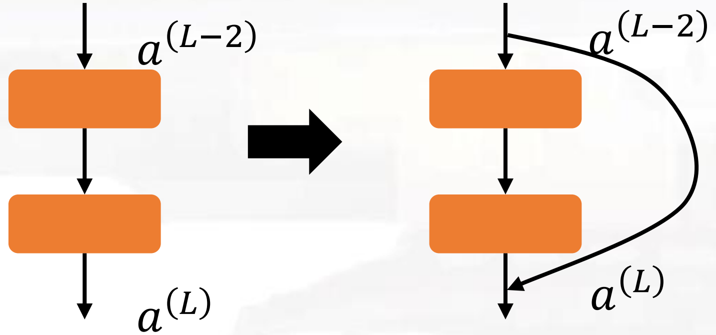


(a) loss landscape (b) gradient predictiveness

Gradient Vanishing and Exploding

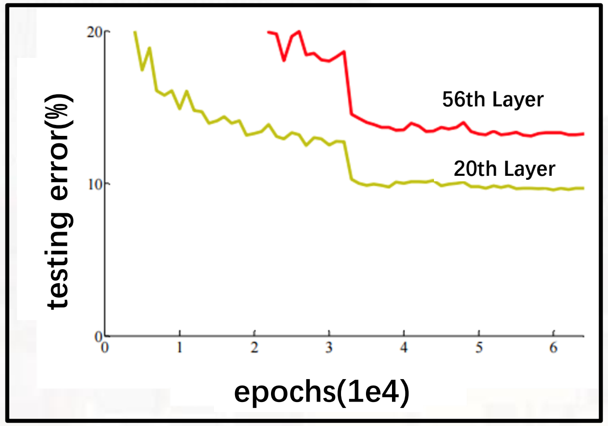
ResNets @ ILSVRC & COCO 2015 Competitions

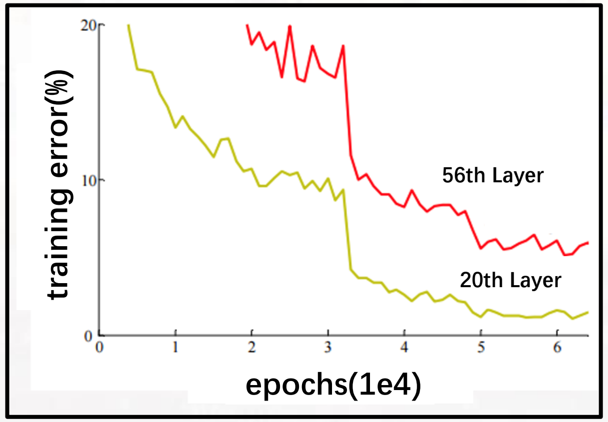
- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

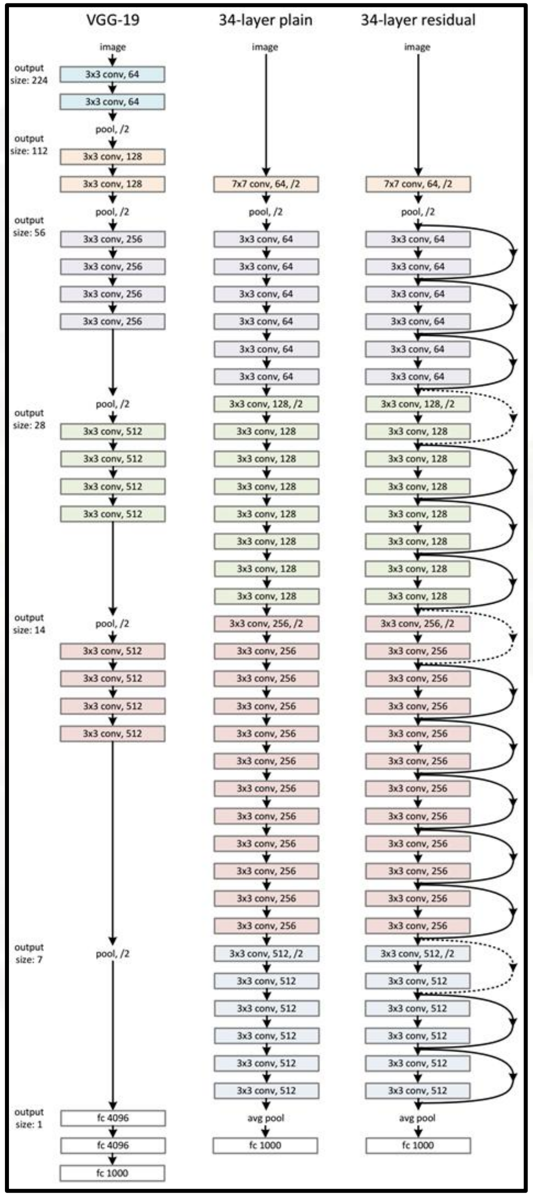


$$a^{(L)} = a^{(L-2)} + H(x)$$

$$\frac{\partial a^{(L)}}{\partial a^{(L-2)}} = 1 + \frac{\partial H(x)}{\partial a^{(L-2)}}$$





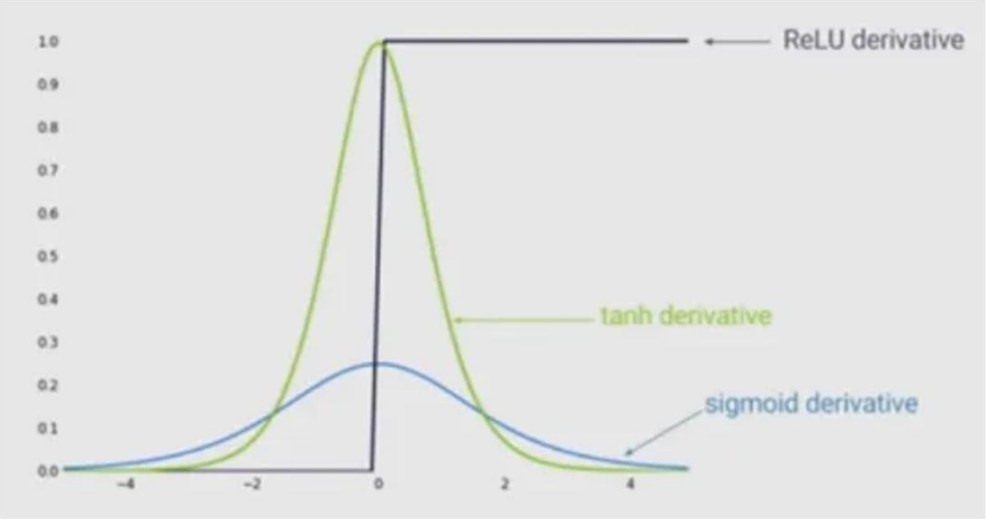


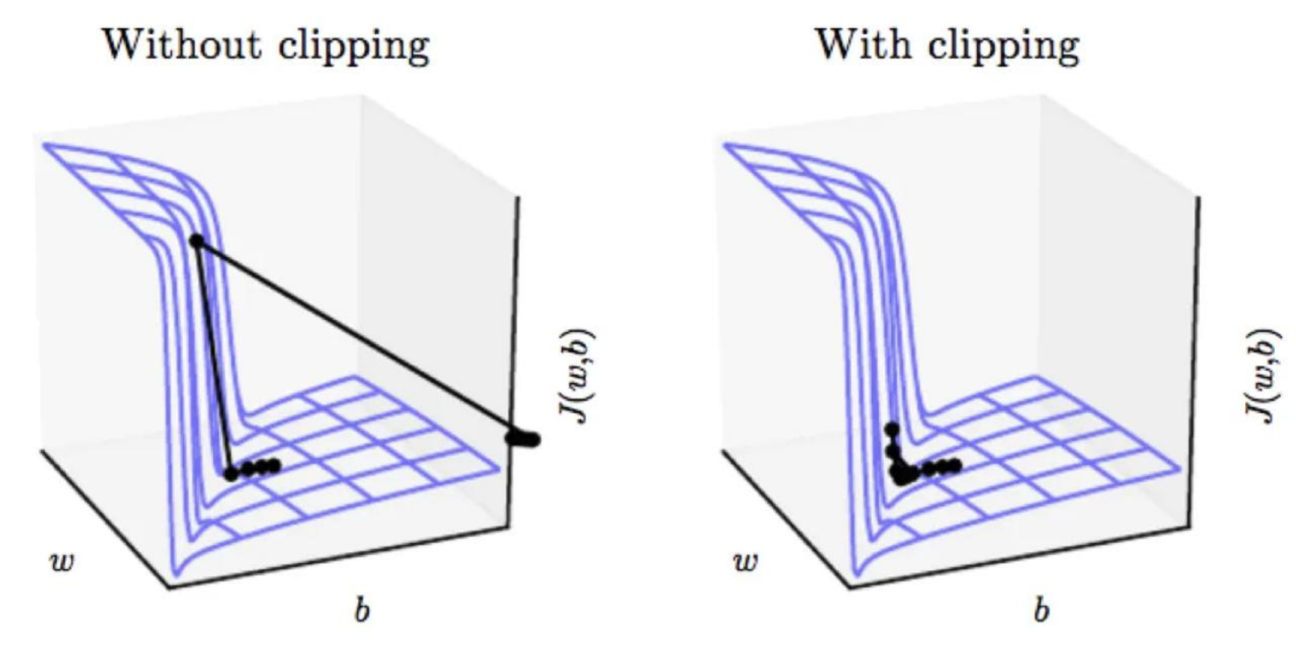
$$\frac{\partial C_0}{\partial w^{(L-2)}} = \frac{\partial z^{(L-2)}}{\partial w^{(L-2)}} \cdot \frac{\partial a^{(L-2)}}{\partial z^{(L-2)}} \cdot \frac{\partial z^{(L-1)}}{\partial a^{(L-2)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}}$$

Gradient Vanishing and Exploding

- ReLU
- Gradient clipping（gradient exploding） $\nabla C = \eta \frac{\nabla C}{|\nabla C|_2}$

$$\frac{\partial C_0}{\partial w^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$





Gradient Vanishing and Exploding

- **Batch gradient descent**

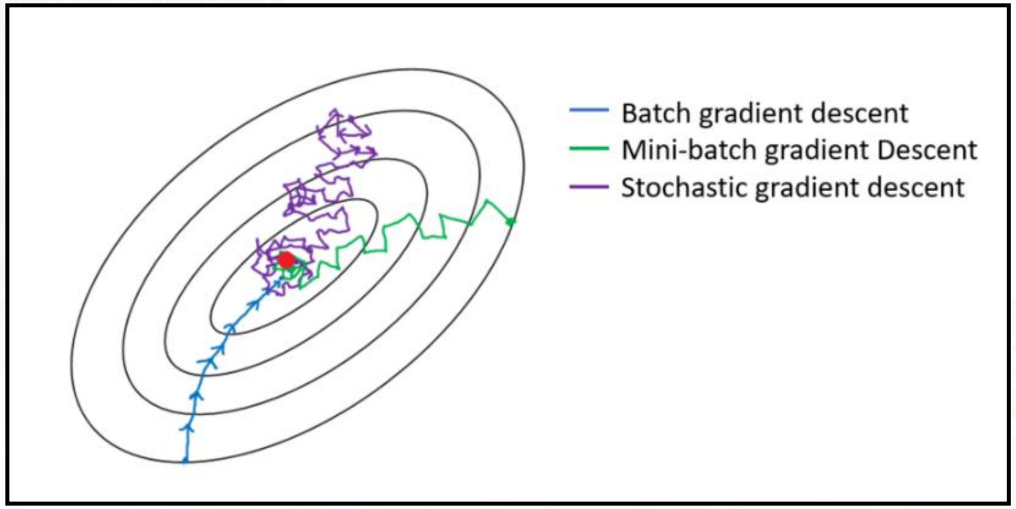
$$W = W - \eta \cdot \nabla_W C(W)$$

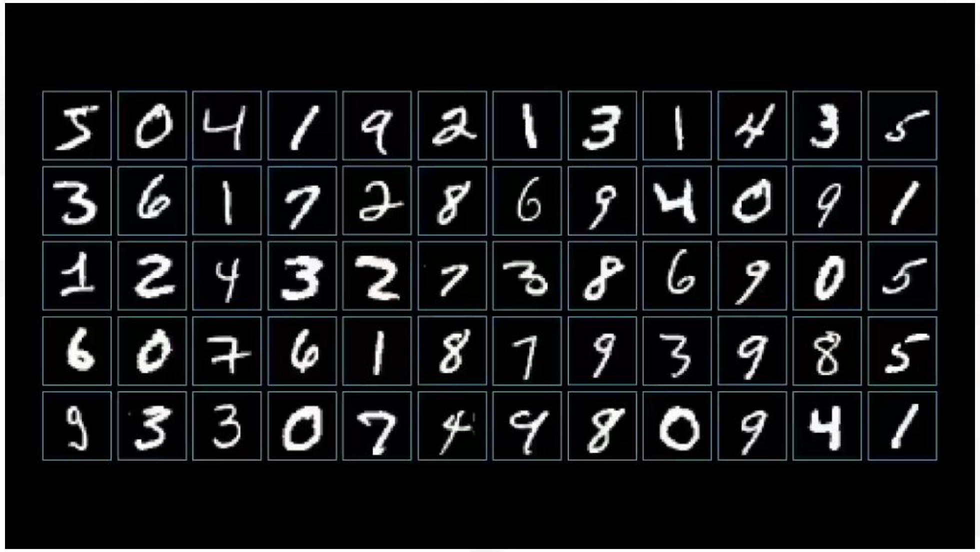
- **Stochastic gradient descent**

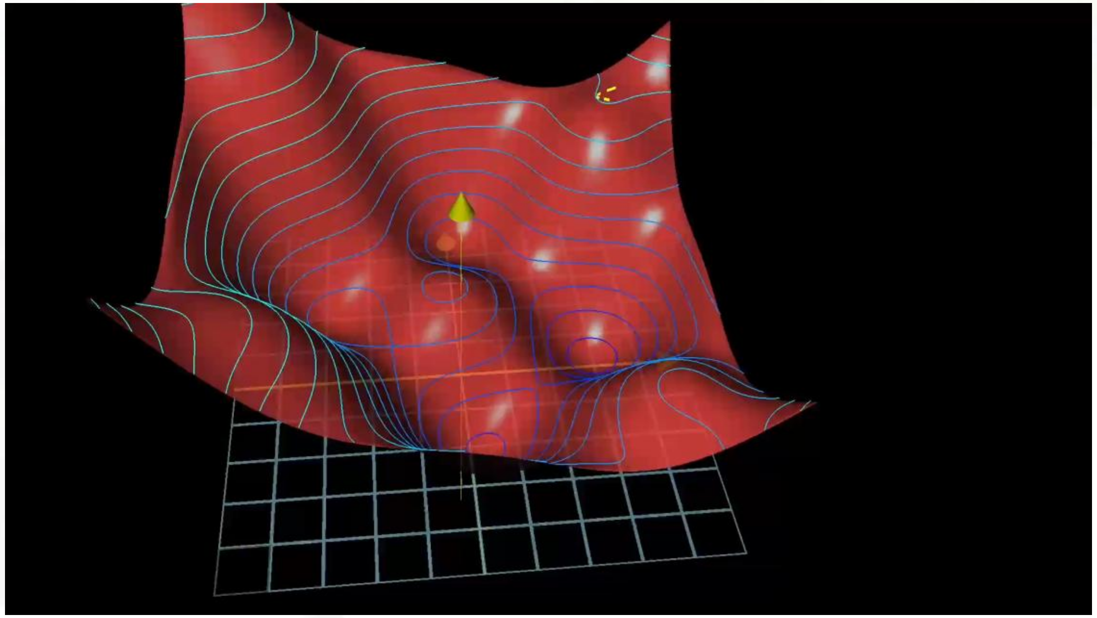
$$W = W - \eta \cdot \nabla_W C\left(W; x^{(i)}; y^{(i)}\right)$$

- **Mini-batch gradient descent**

$$W = W - \eta \cdot \nabla_W C\left(W; x^{(i:i+n)}; y^{(i:i+n)}\right)$$







SGDM

- **Momentum**
  - SGD has the problem of slow convergence in 'valley'-shaped spaces
  - SGDM speeds up SGD by modifying the direction.
- **Add an update direction of a past moment**

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

$$V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + V_t$$

Image 2: SGD without momentum

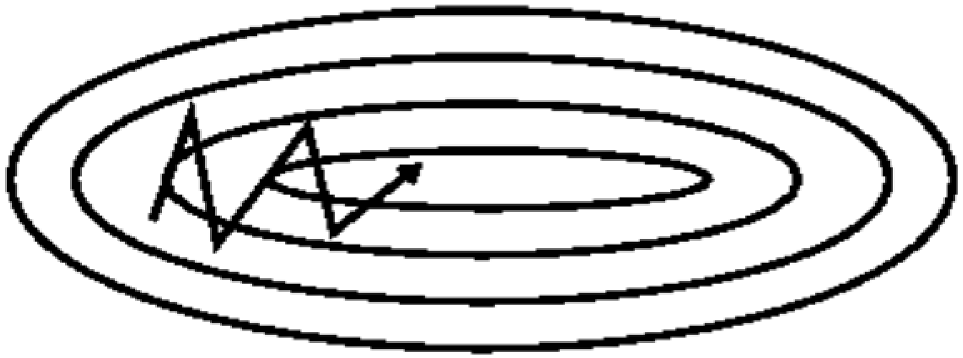


Image 3: SGD with momentum

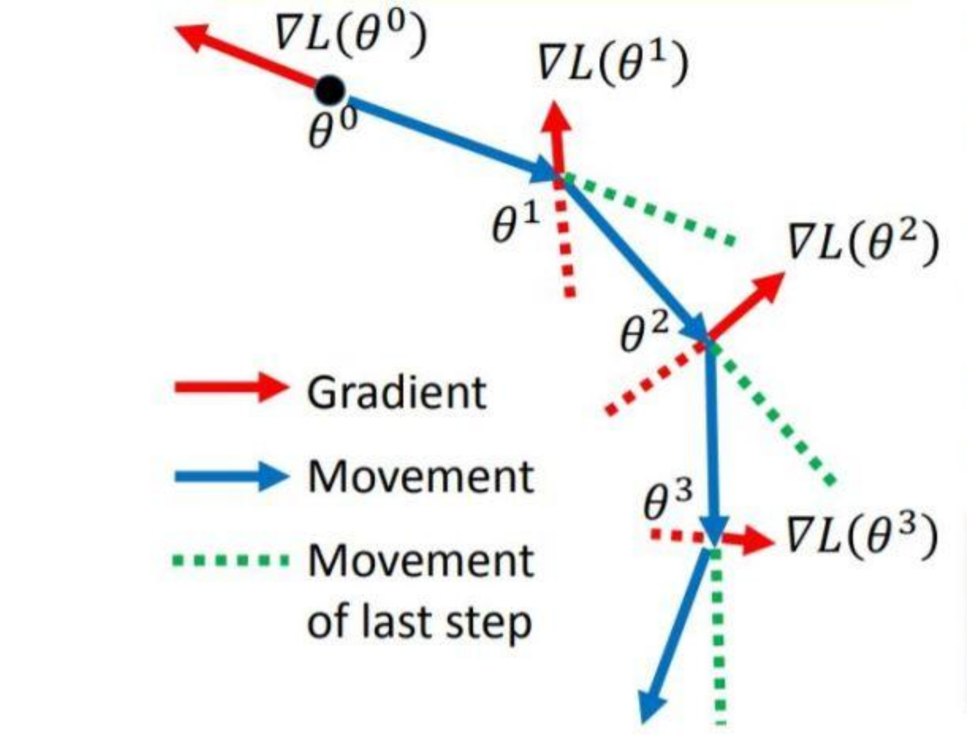
SGDM

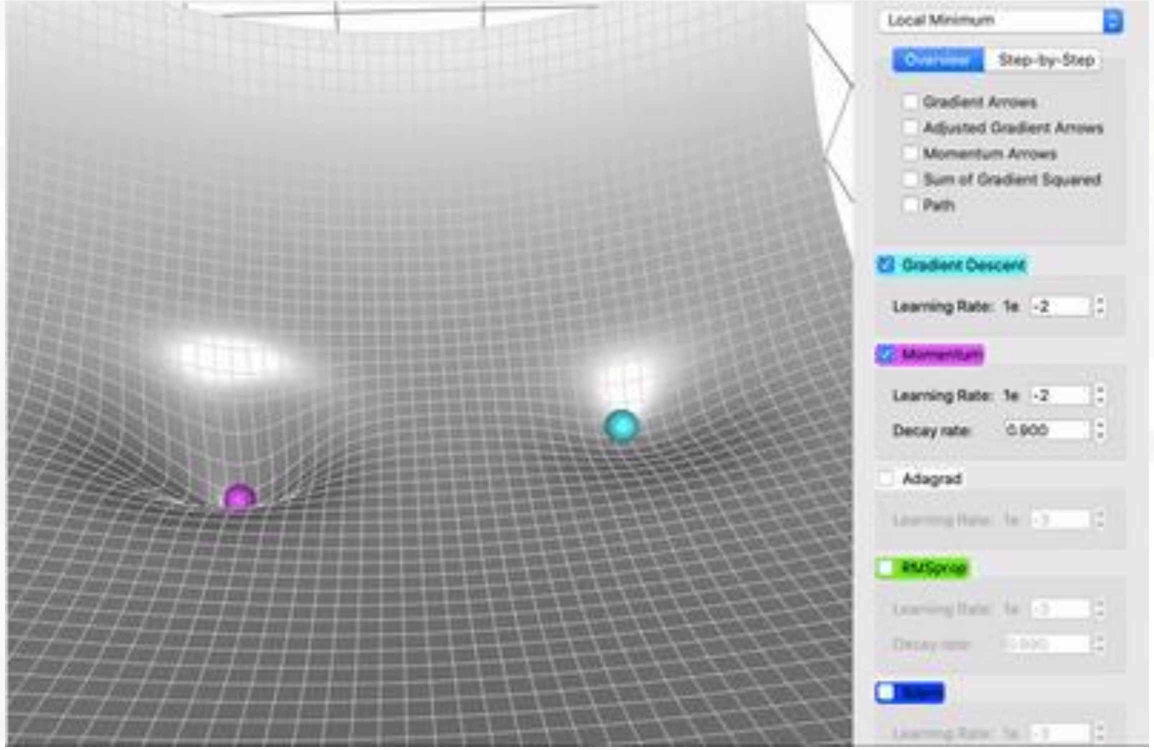
Advantage

- faster convergence
- chance of escaping the local minimum

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}).$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$



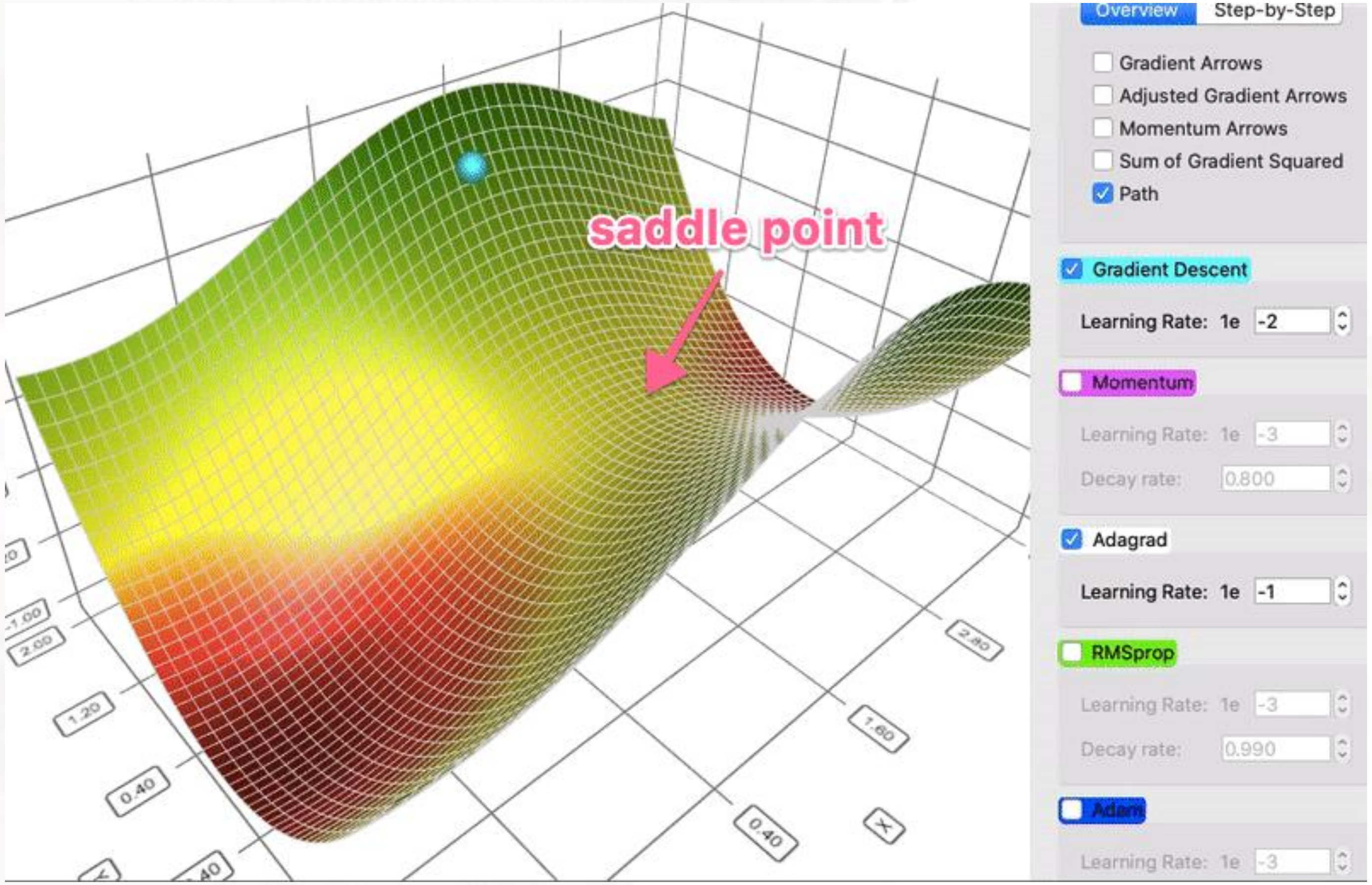


AdaGrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i}$$

$$G_t^{(i,i)} = \sum_\tau^t (g_\tau^{(i)})^2$$

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$



RMSProp

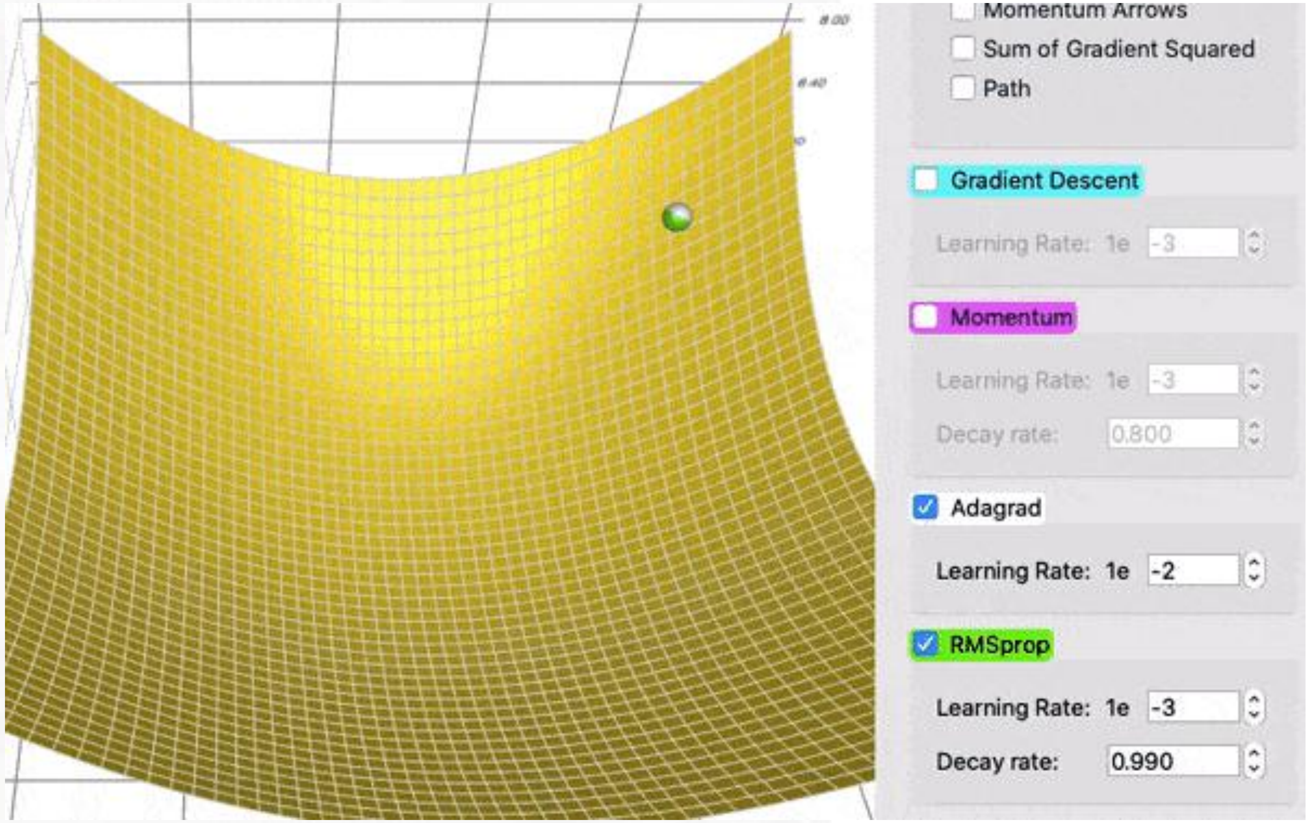
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i}$$

$$G_t^{(i,i)} = \sum_\tau^t (g_\tau^{(i)})^2$$

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$



Adam(Adaptive Moment Estimation)

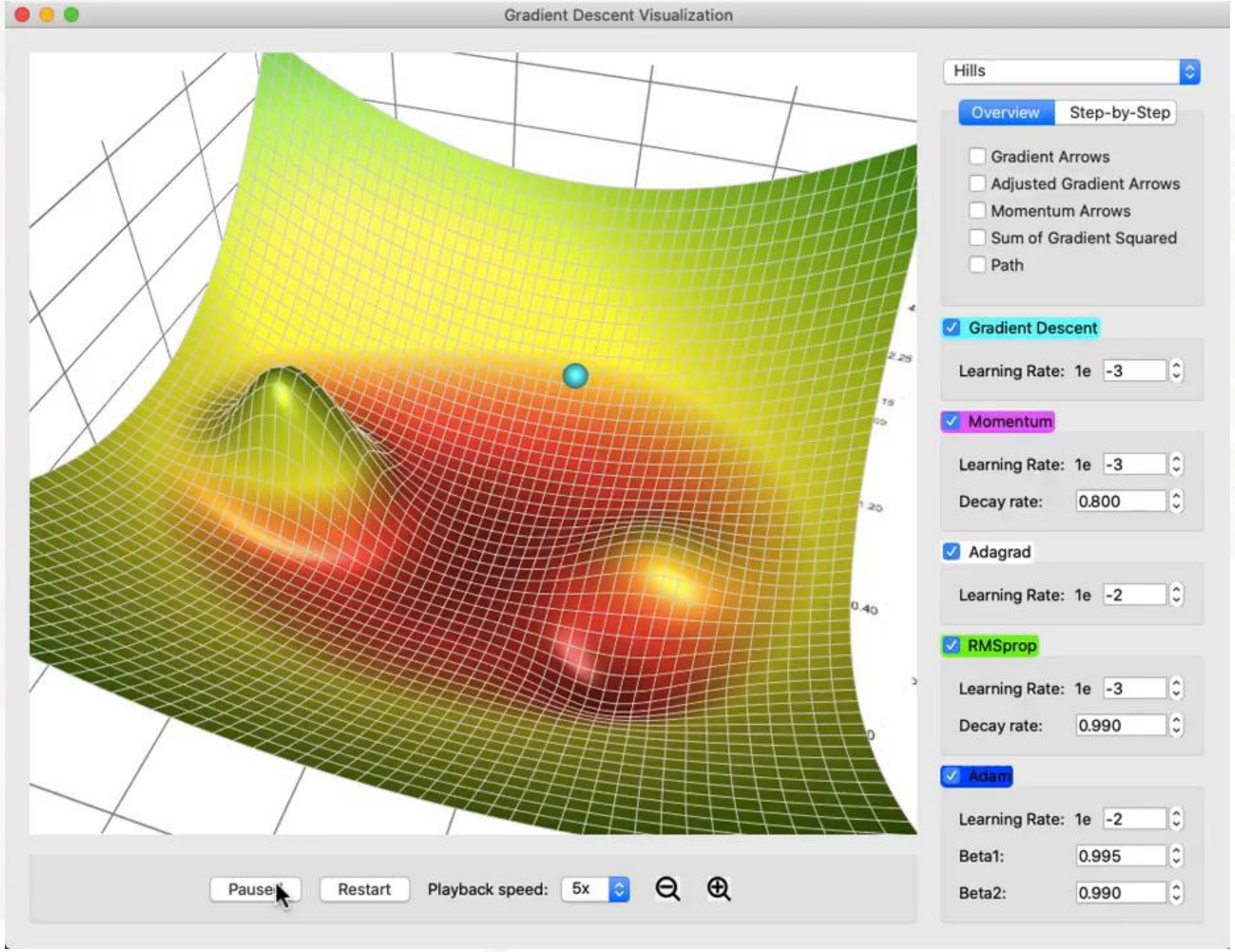
- Combine SGDM and RMSProp

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

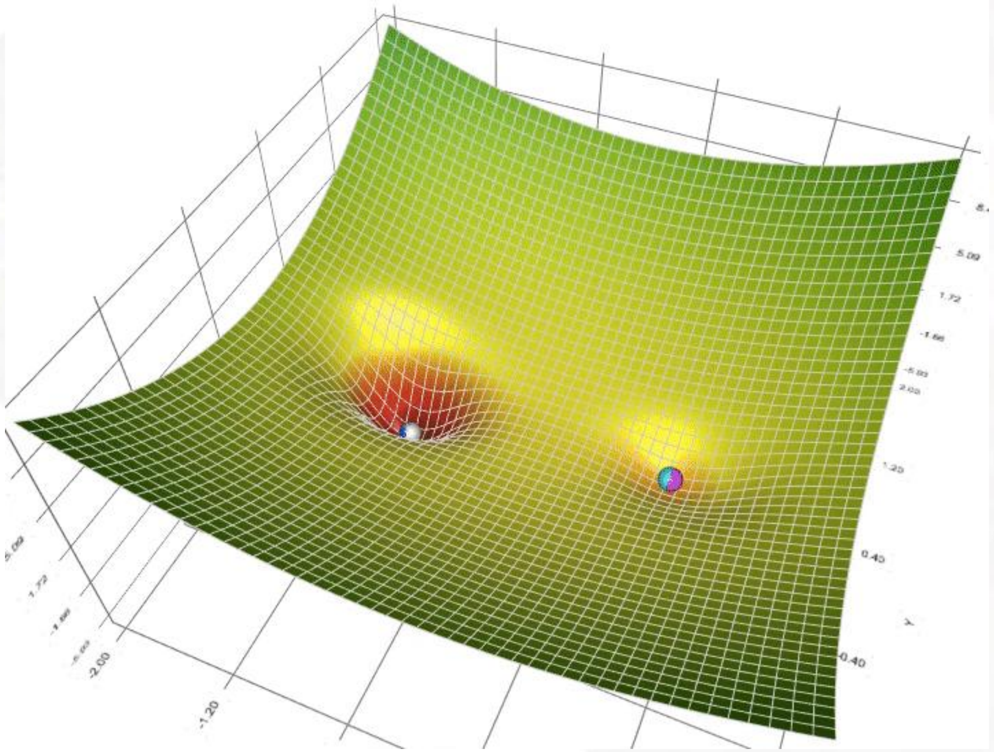
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

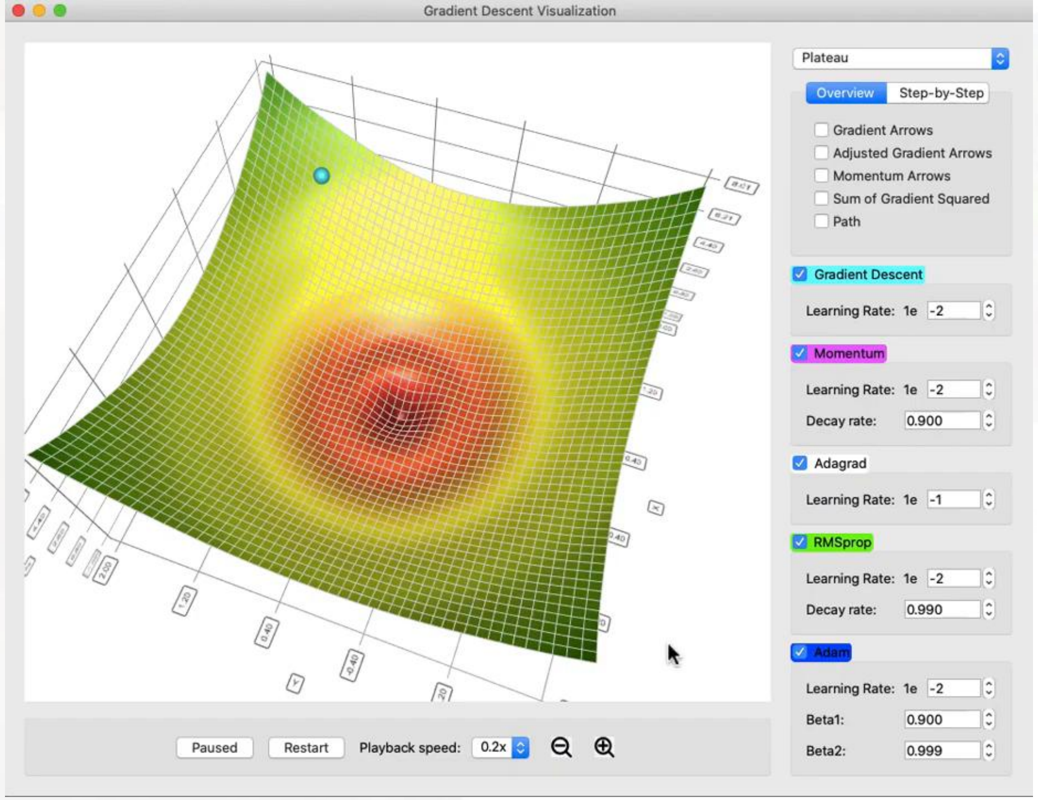
$$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2$$



The Advantage of Adam

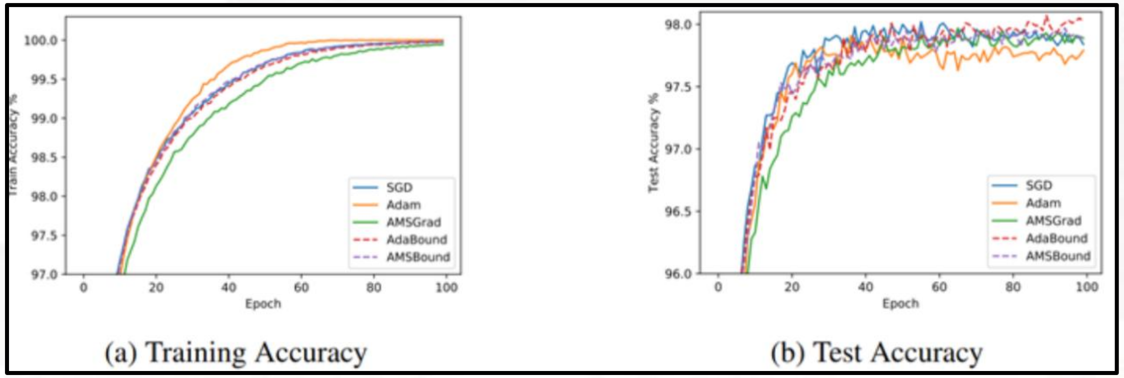
- The advantage of Adaptive
- The advantage of Momentum



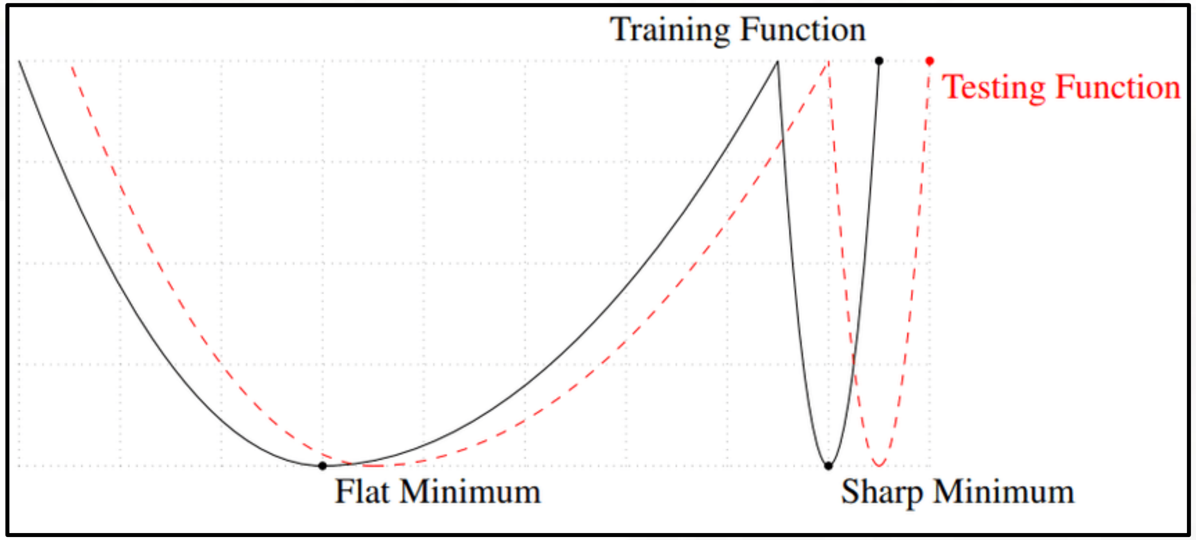


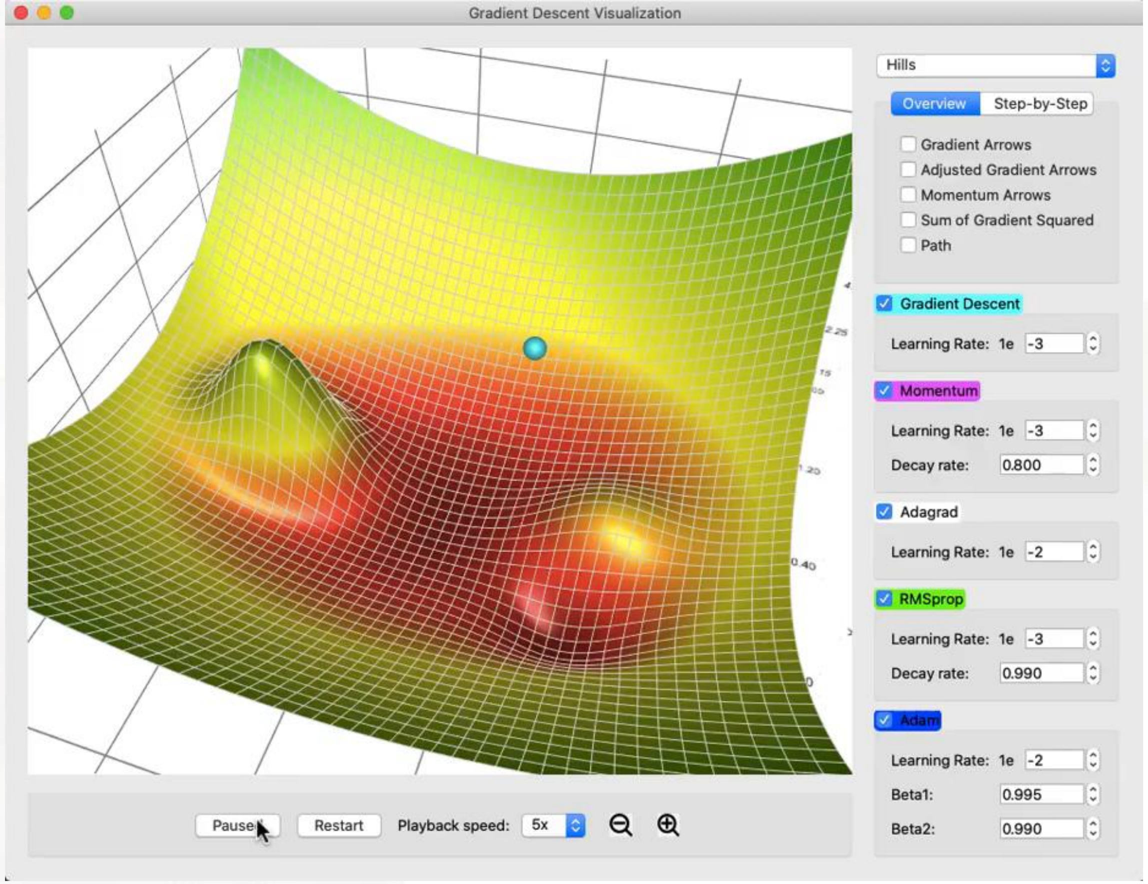
The Problem of Adam

■ Overfitting



(a) Training Accuracy (b) Test Accuracy





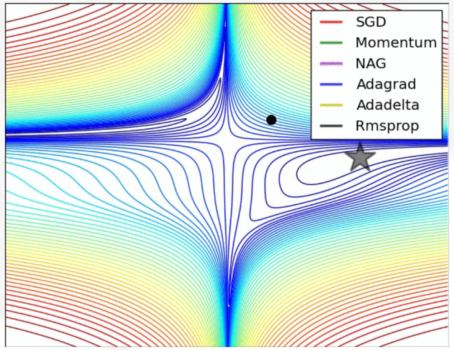
Summary

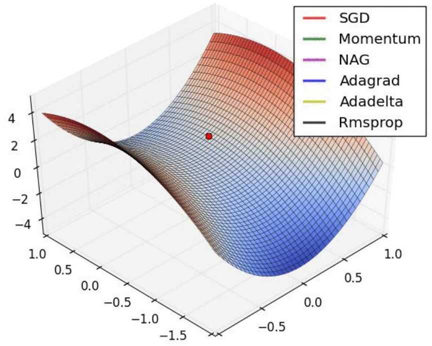
Adaptive vs Non Adaptive

Adaptive Method	Non-Adaptive Method
Adam, AdaGrad, RMSProp	SGDM， SGD
Difficult data, complex networks, hard to converge	Good initialization and learning rate scheduling scheme

	SGDM	Adam
Training Speed	Slow	**Fast**
Convergence	**Good**	Poor
Stability	**Good**	Poor
Generalization	**Good**	Poor

If you are interested in visualizing these or other optimization algorithms, refer to this useful tutorial.

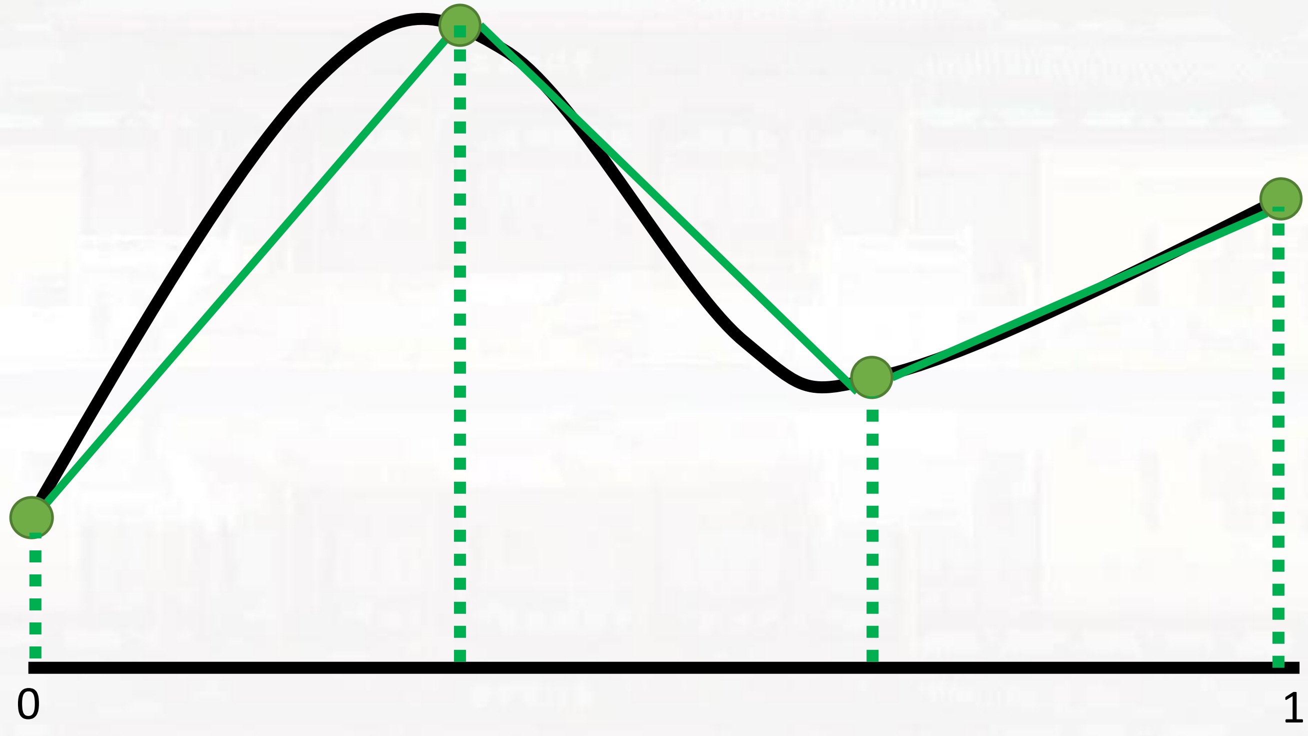




Fit Everything?

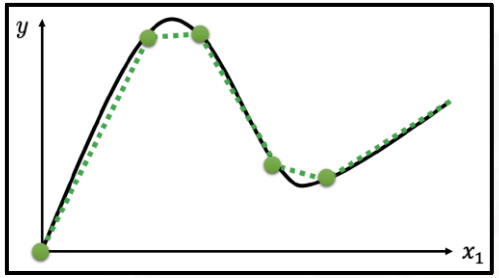
Neural Network

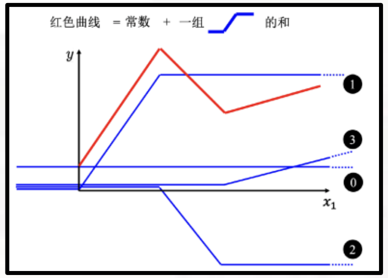
- We can have good approximation with sufficient pieces.

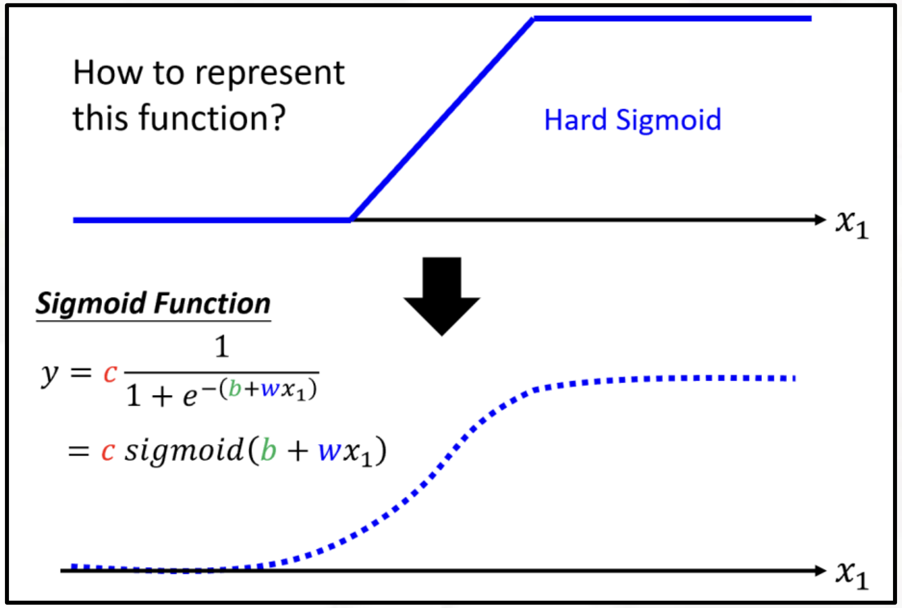


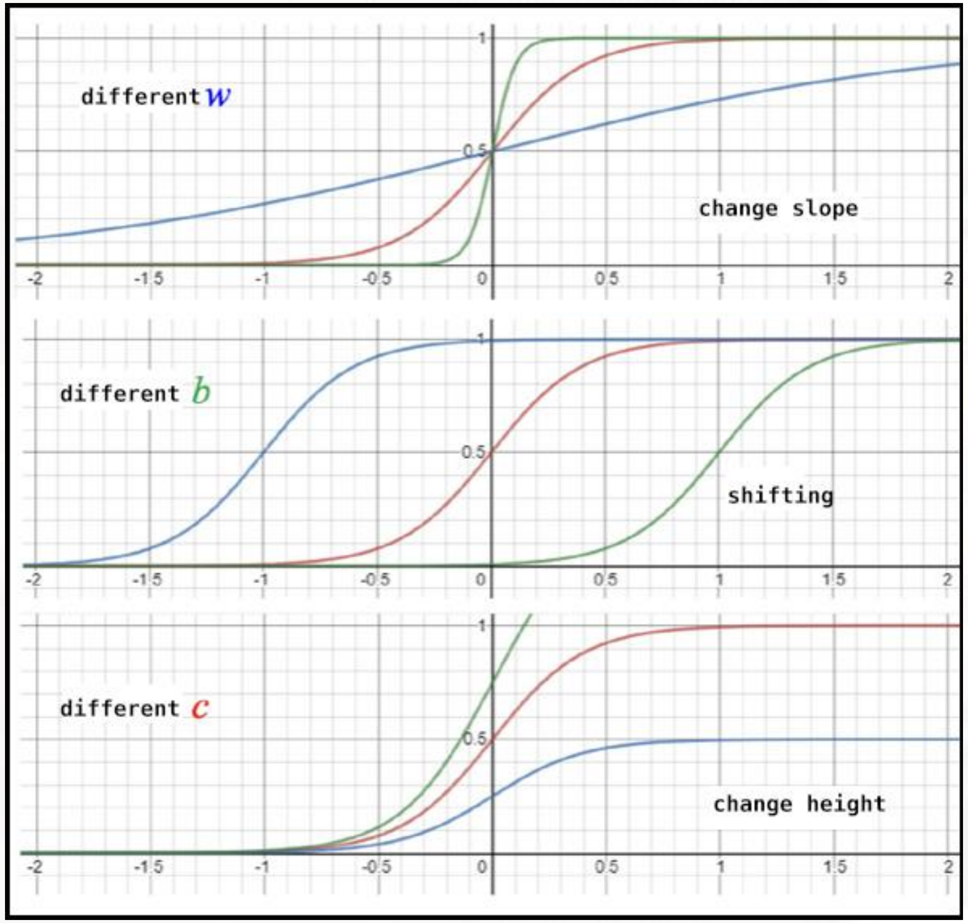
The universal approximation brought by nonlinearity

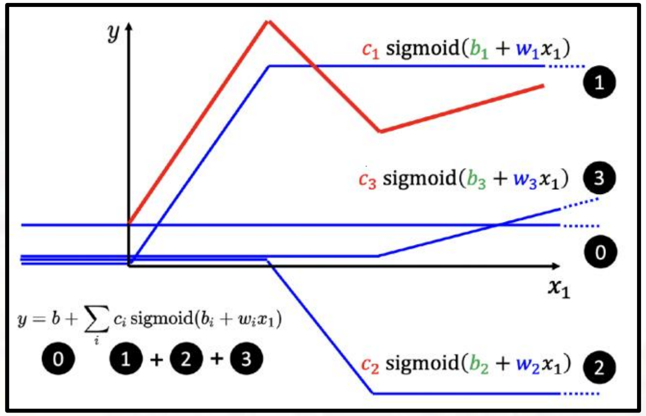
Neural Network











$$y = b + wx_1$$

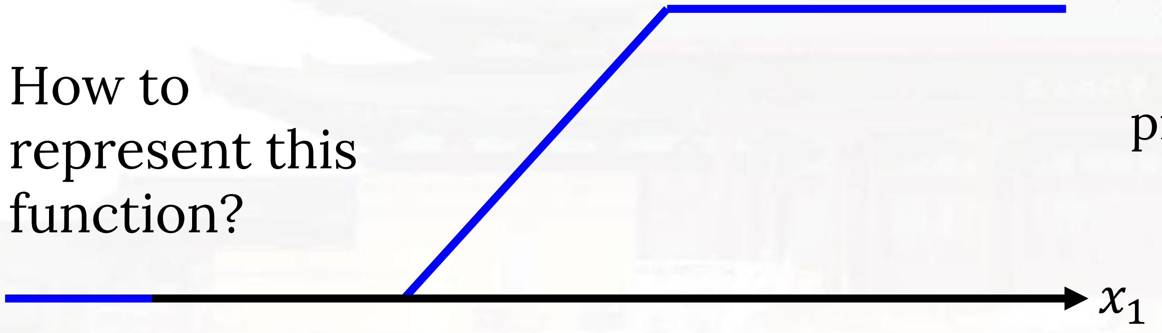
$$y = b + \sum_i c_i \text{ sigmoid}(b_i + w_i x_1)$$

$$y = b + \sum_j w_j x_j$$

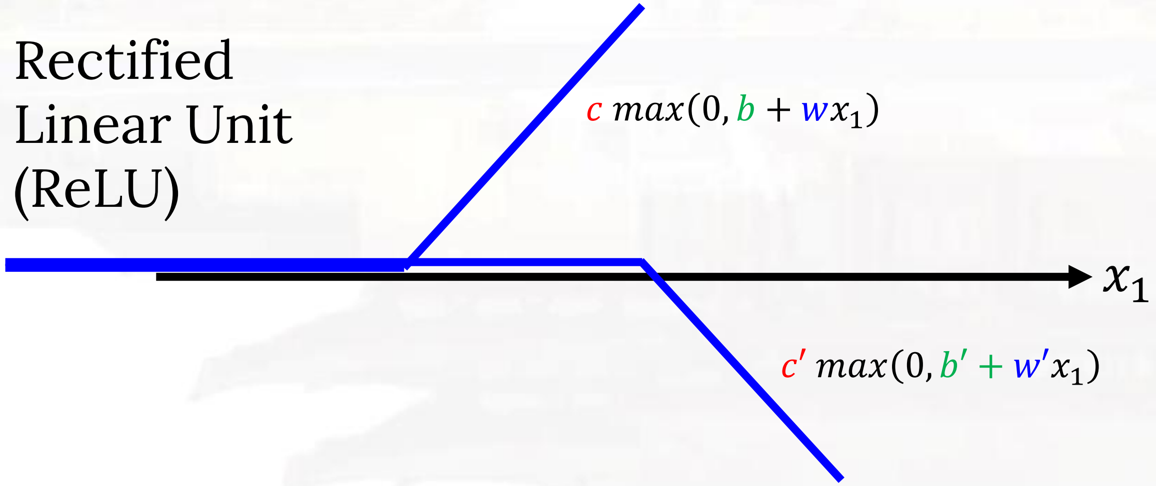
$$y = b + \sum_i c_i \text{ sigmoid}\left(b_i + \sum_j w_{ij} x_j\right)$$

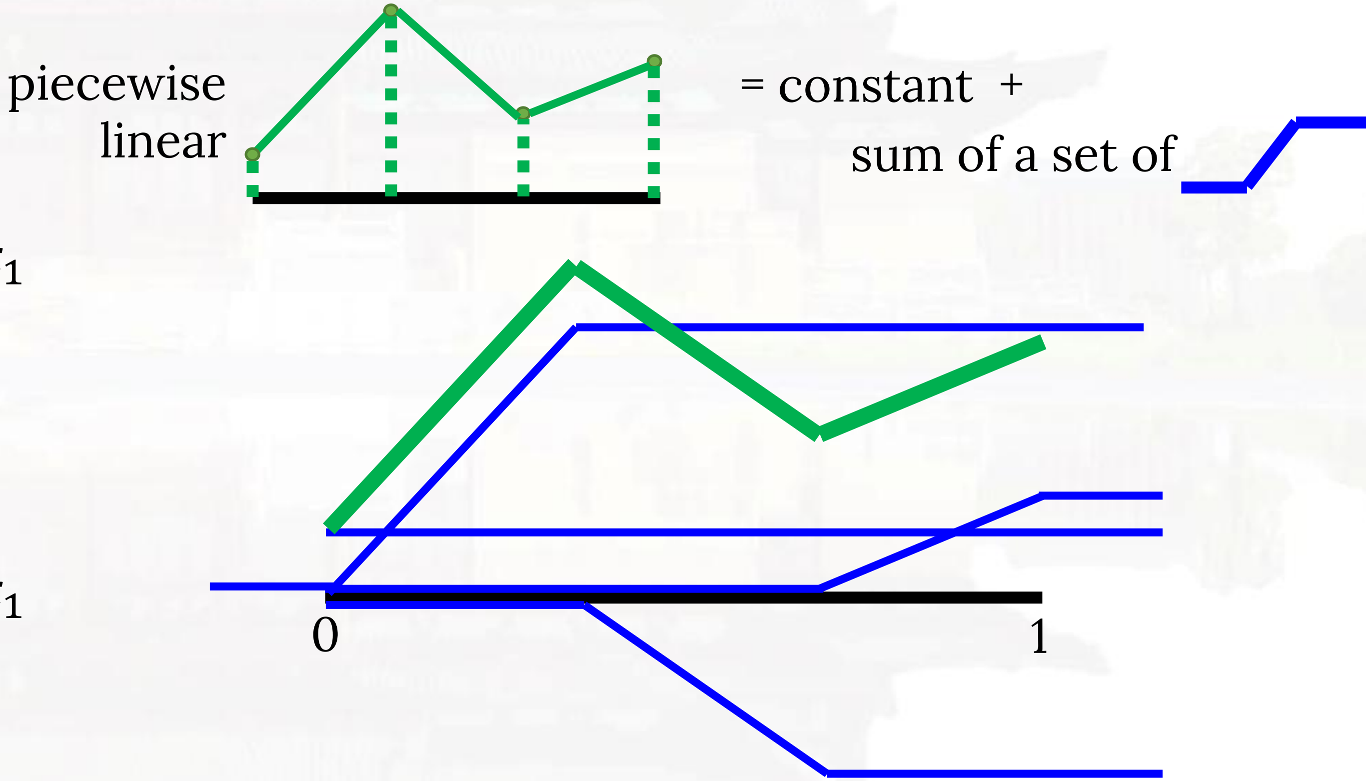
Neural Network

How to represent this function?



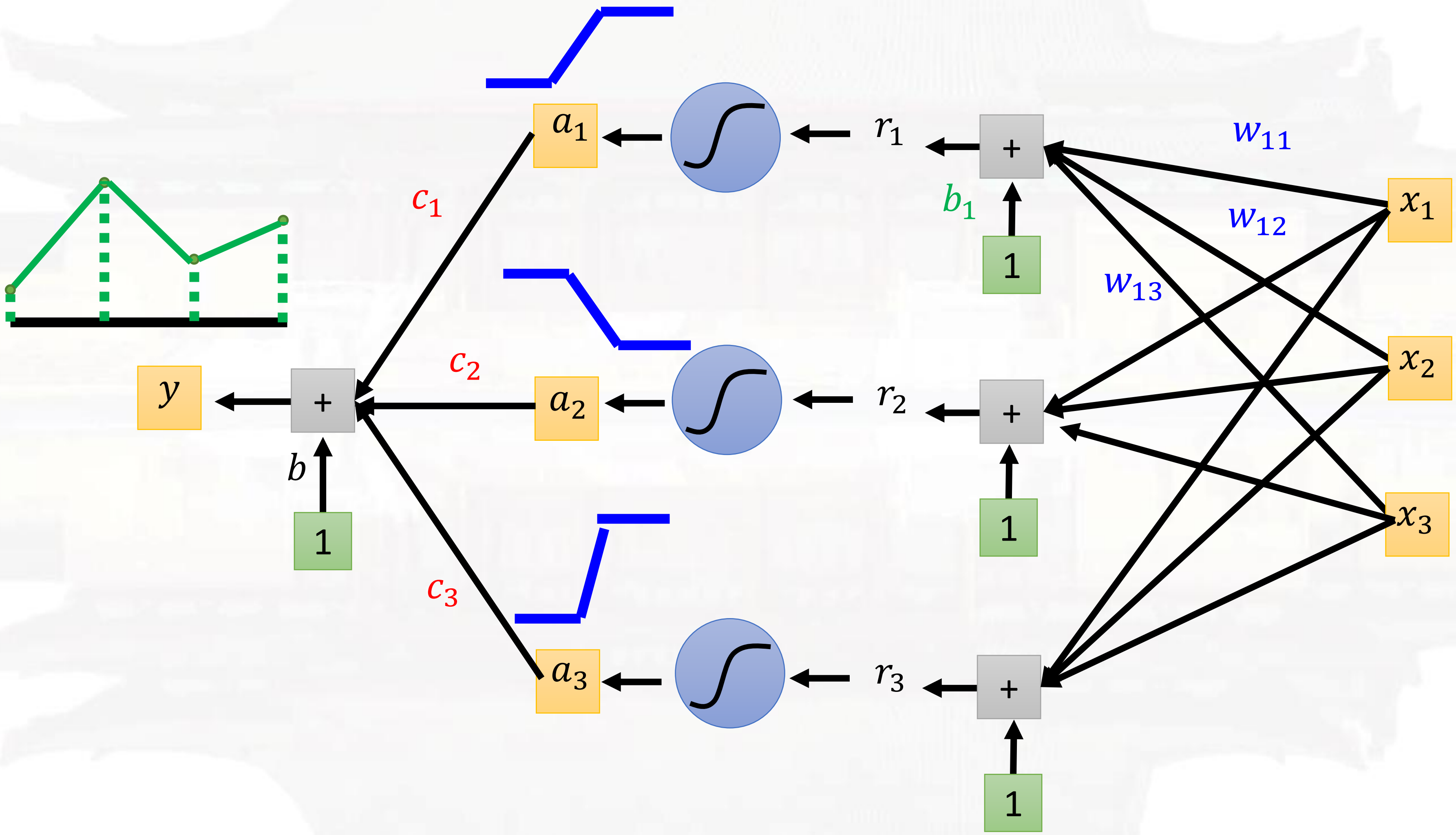
Rectified Linear Unit (ReLU)





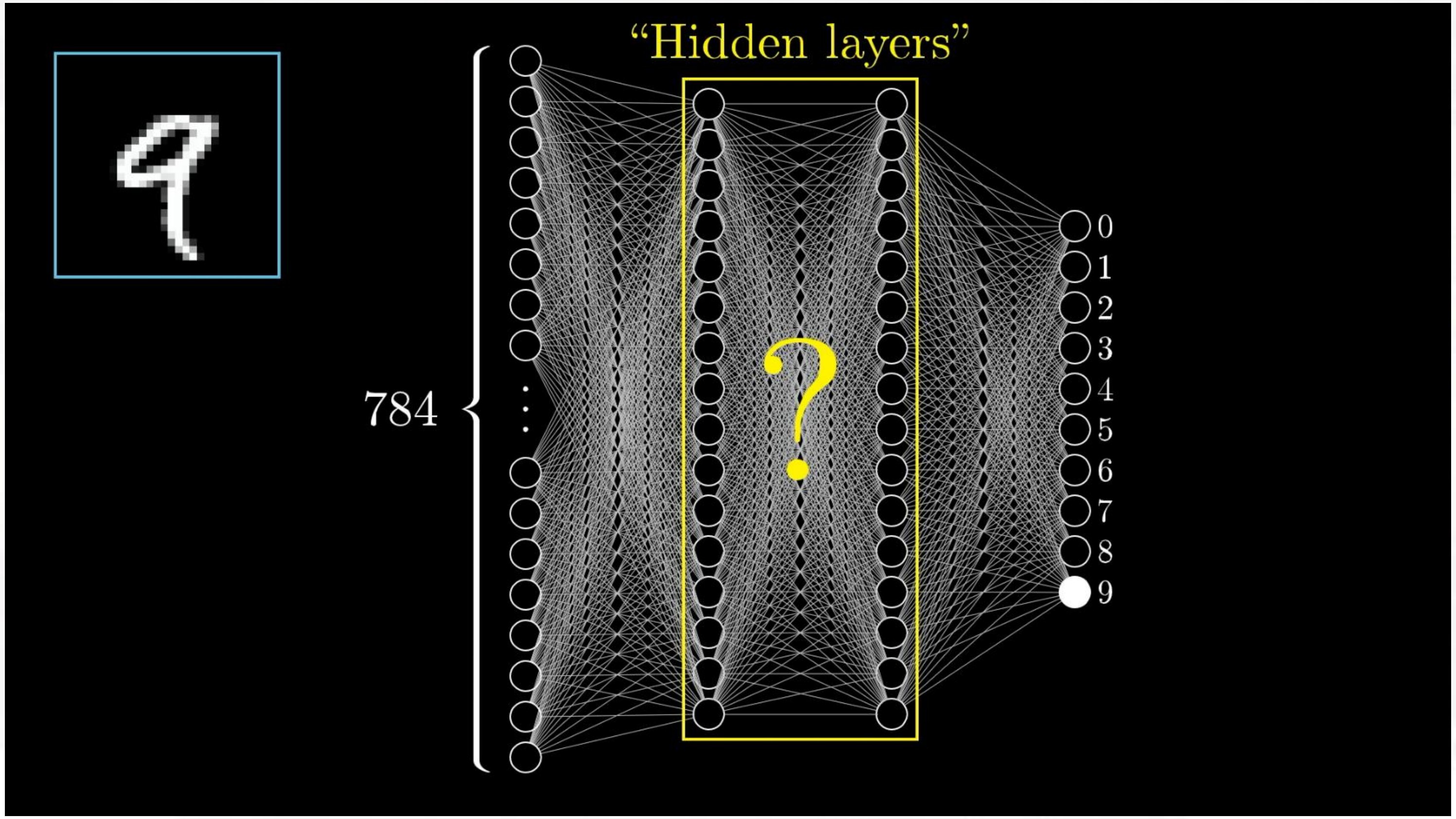
The universal approximation brought by nonlinearity

Neural Network



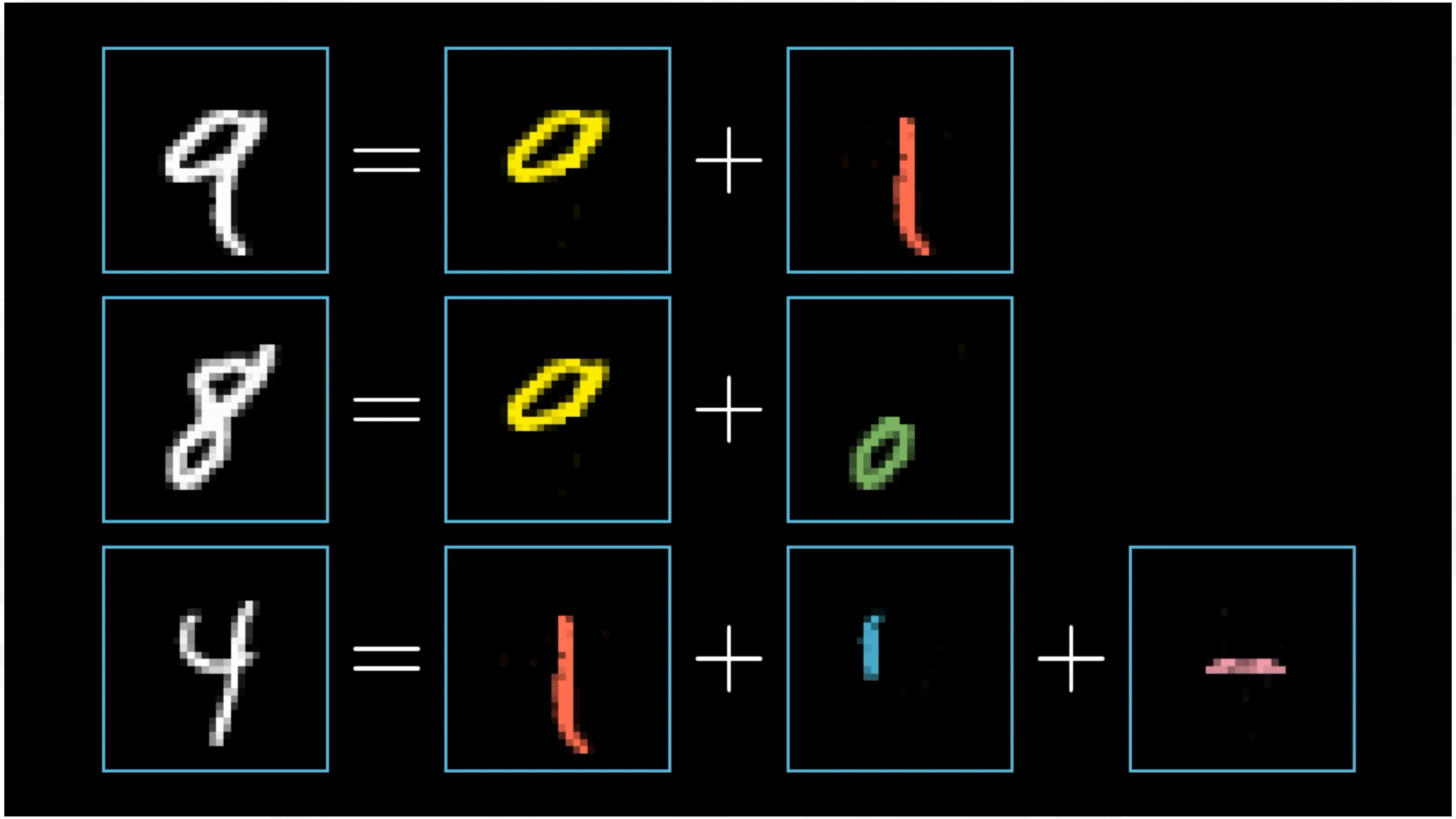
Neural Network

Why Use Layers?



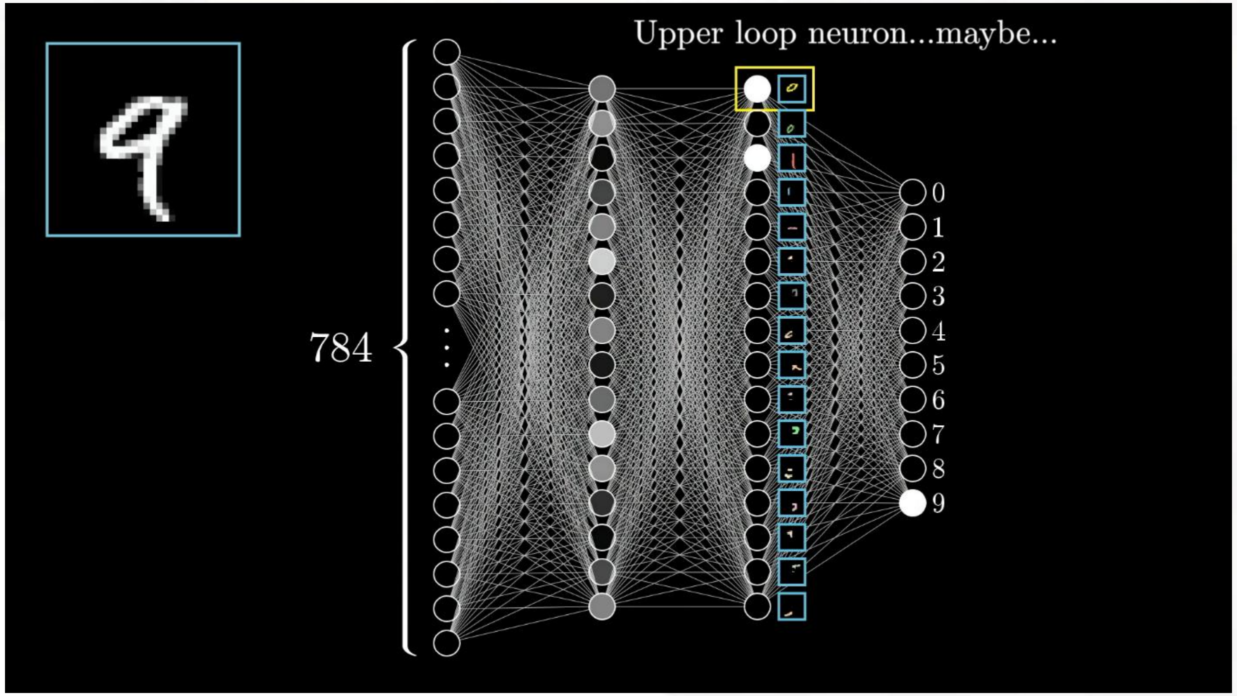
Neural Network

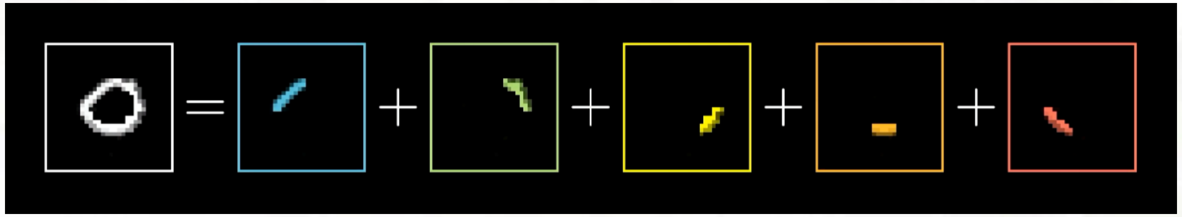
- we piece together various components like loops and lines

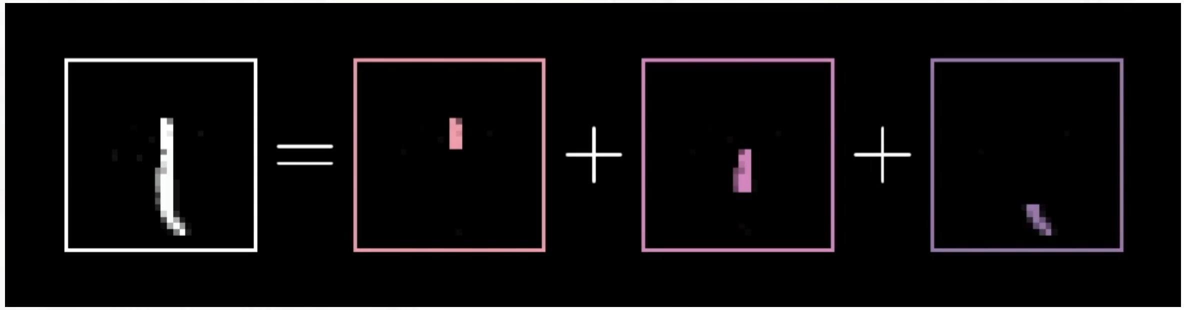


Neural Network

- In a perfect world, we might hope that each neuron in the second-to-last layer corresponds to one of these subcomponents.

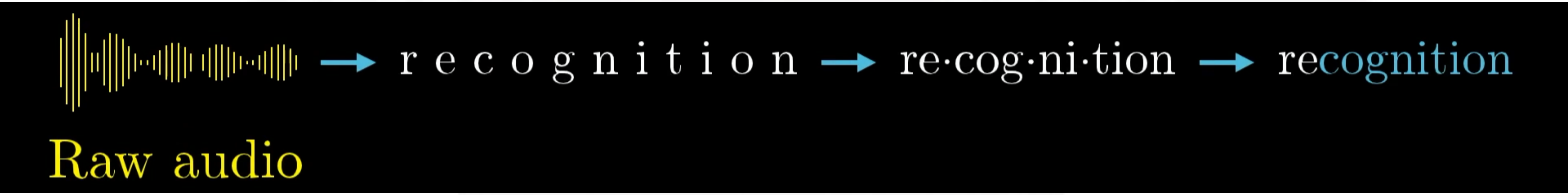






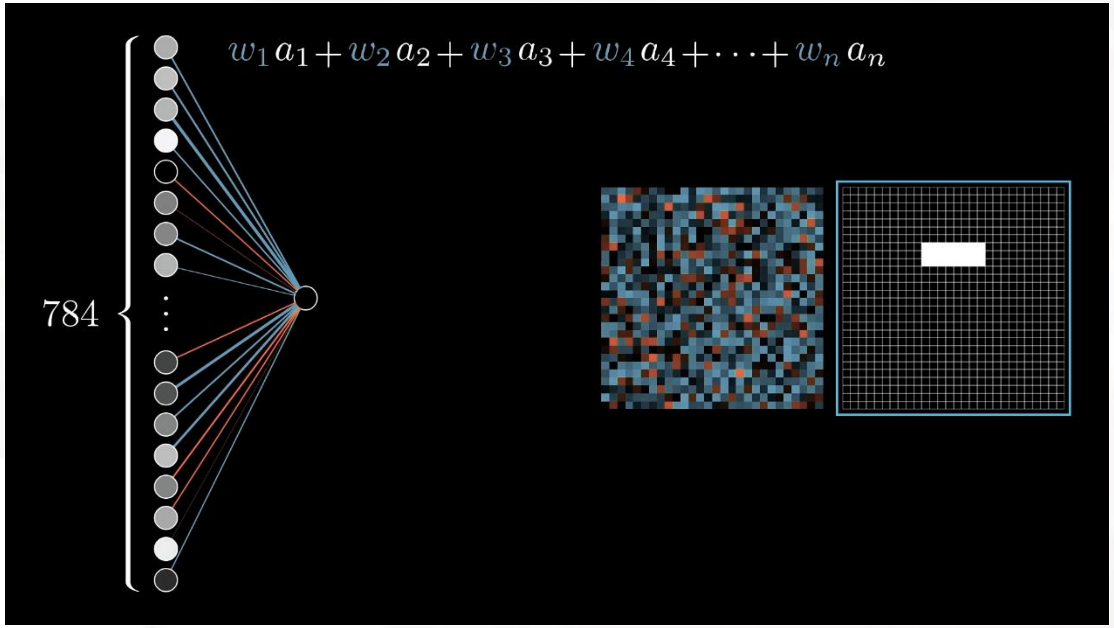
Neural Network

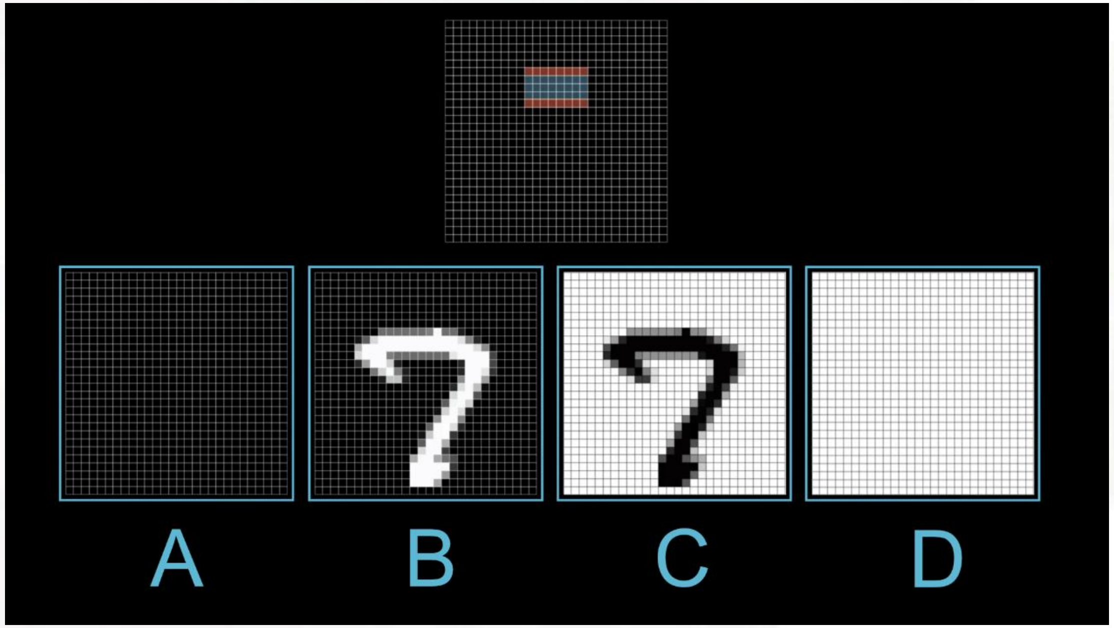
- Layers Break Problems Into Bite-Sized Pieces
  - Edge detection is a useful step for all kinds of image-recognition problems.
  - beyond image recognition



Neural Network

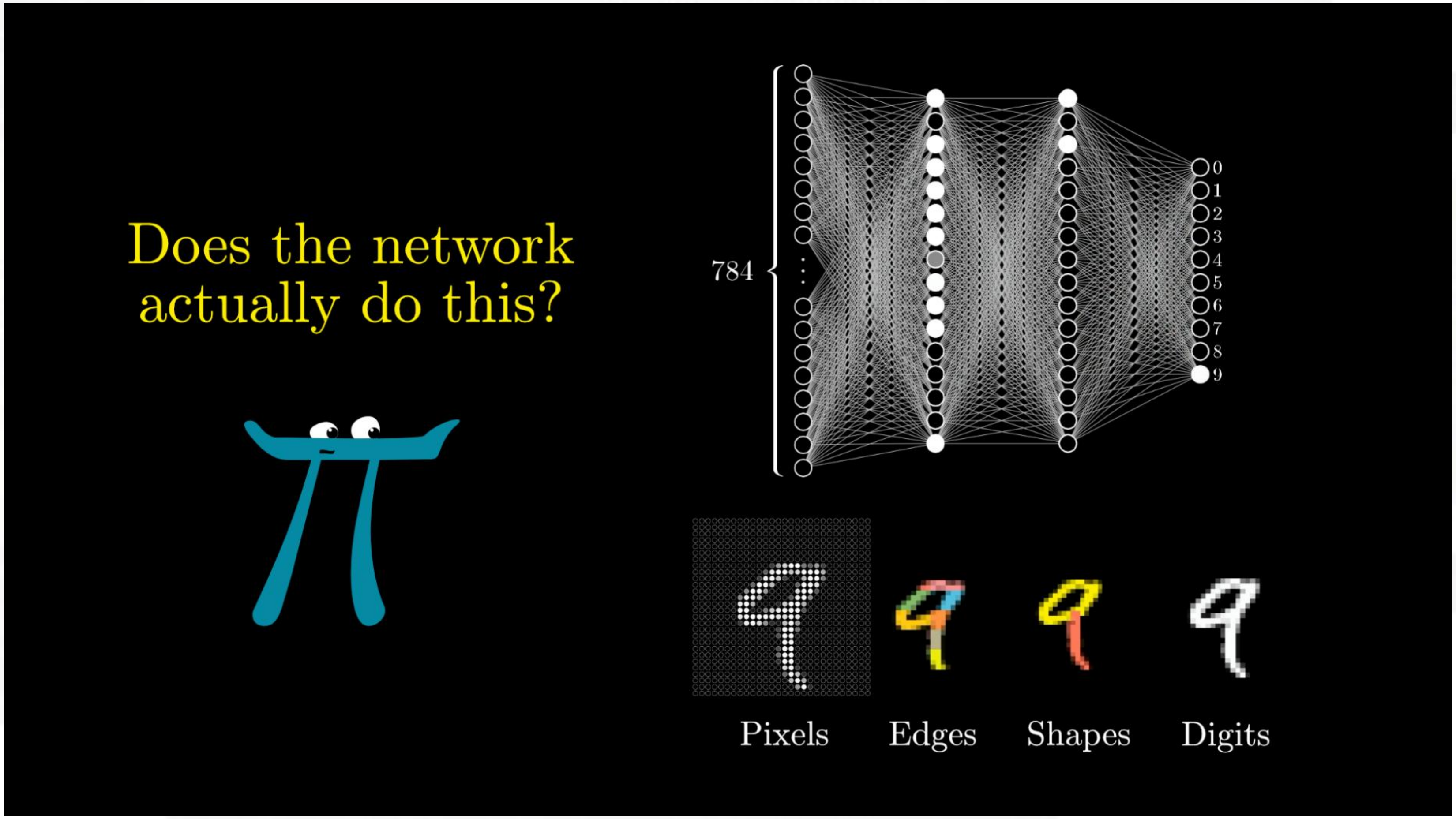
- Rank the four images (A, B, C, and D) based on how much they would activate that neuron:





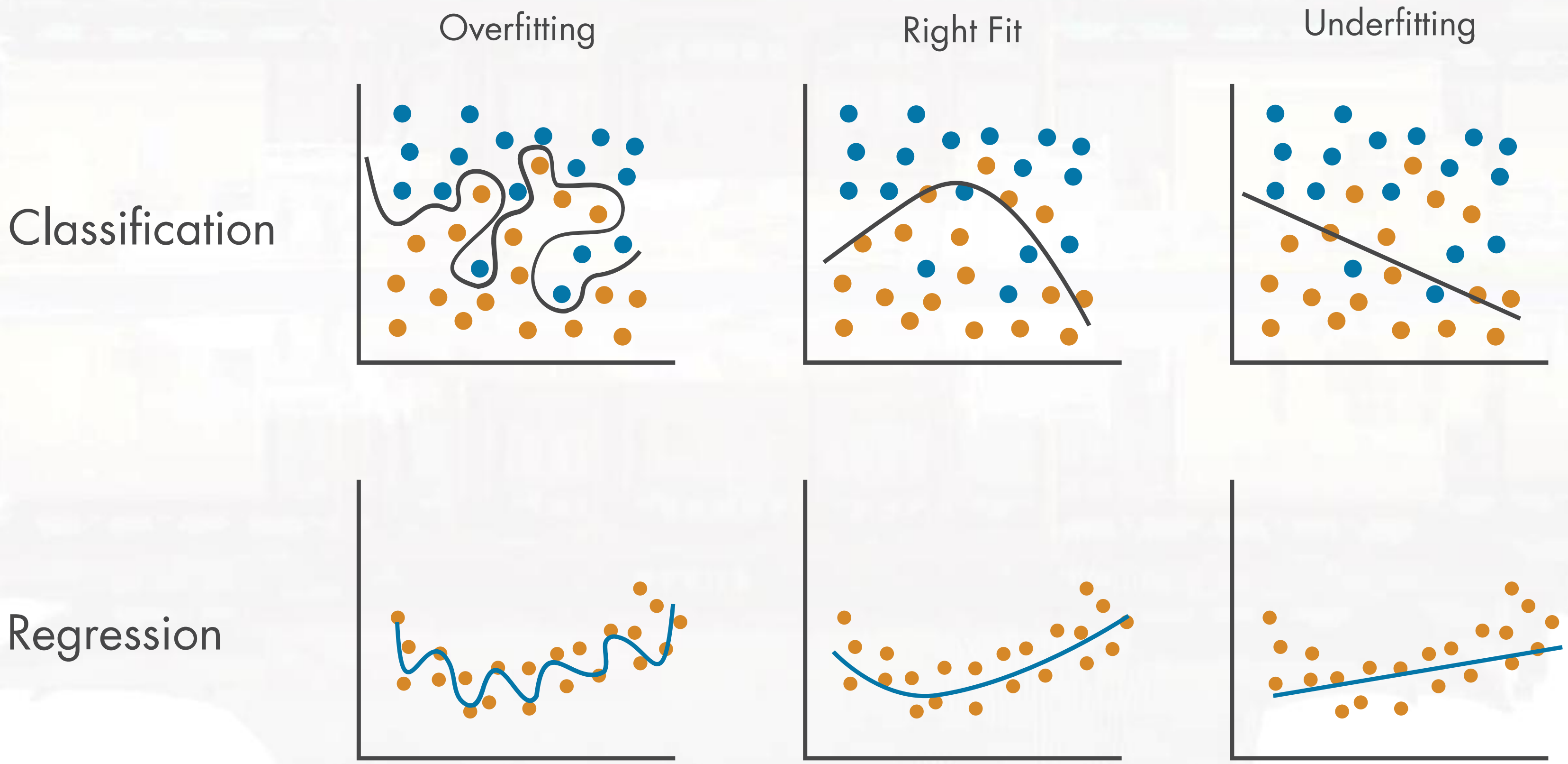
Neural Network

- We hope … but

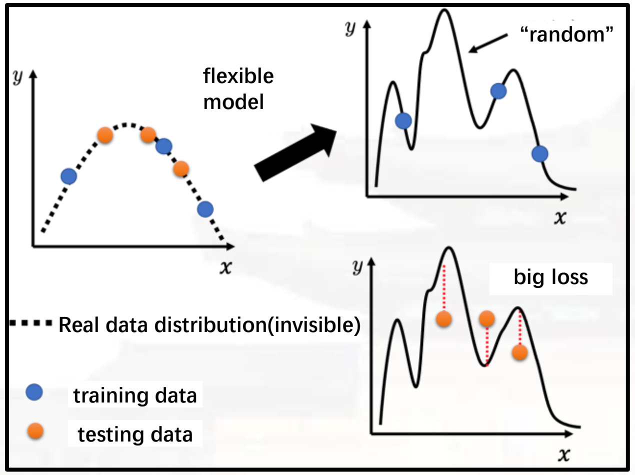


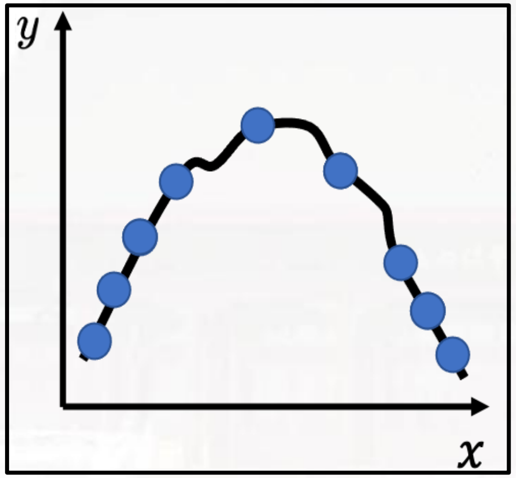
Overfitting and Underfitting

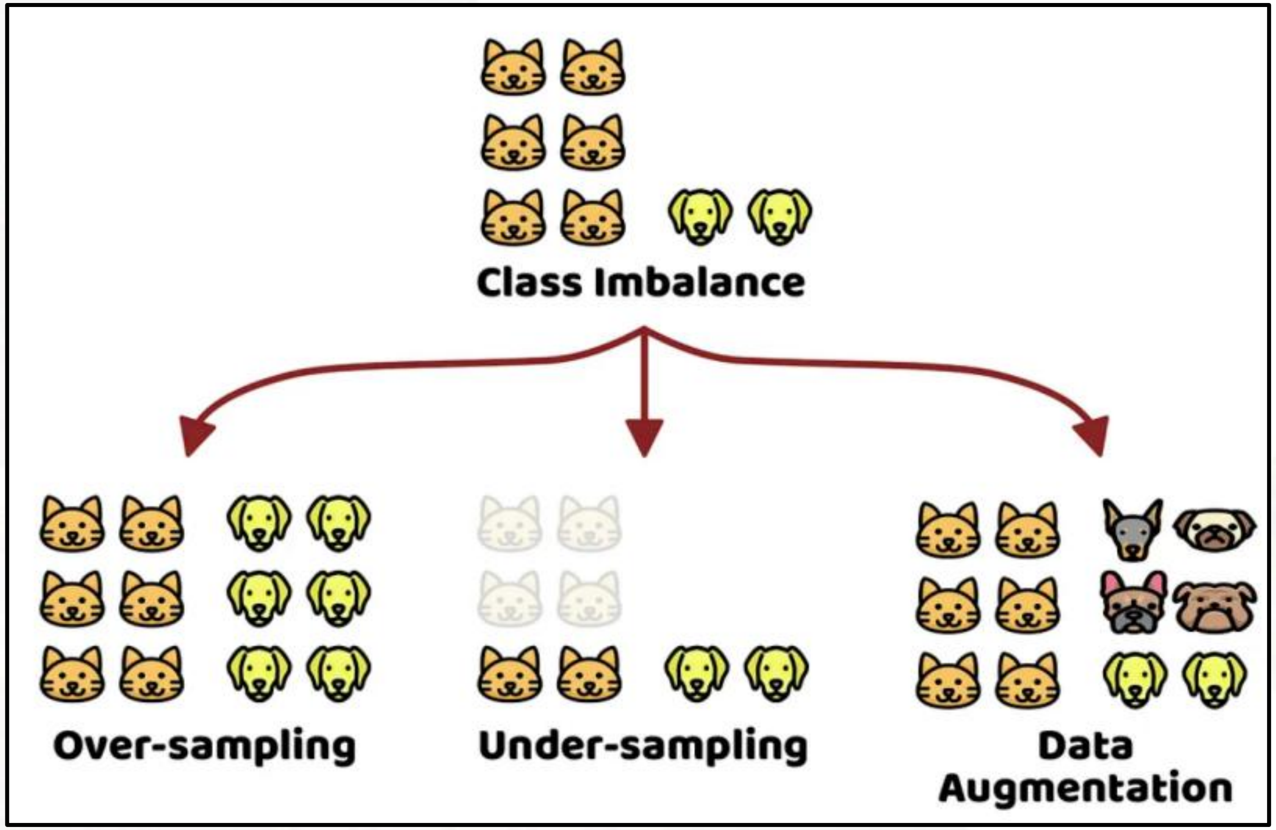
- The balance between data (knowns or constraints) and parameters (unknowns)

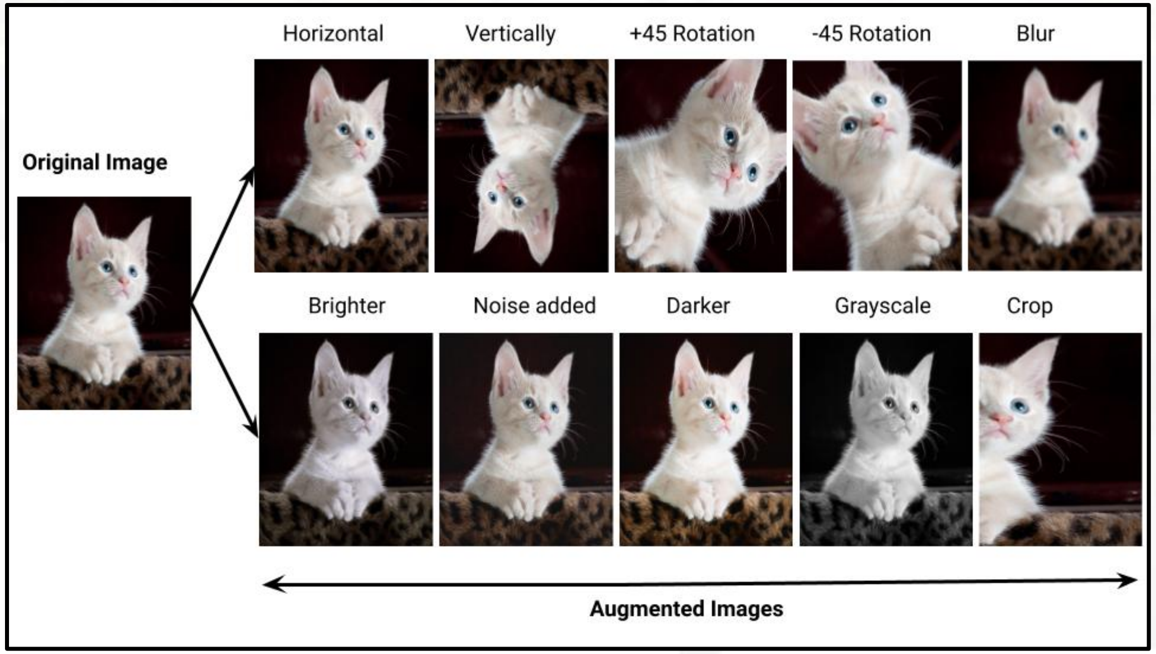


Overfitting: Data Augmentation

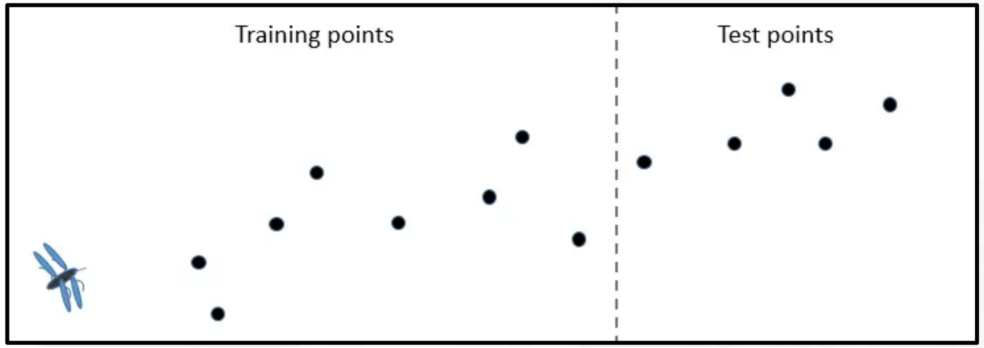


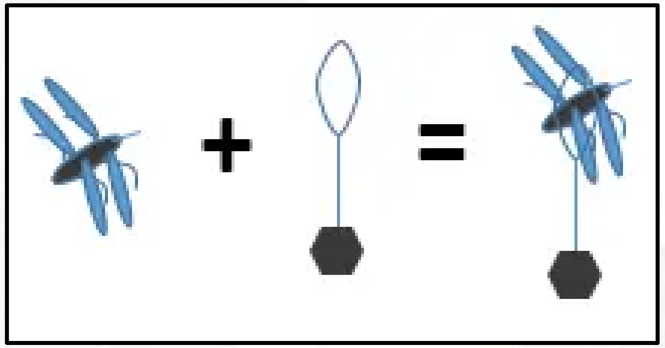


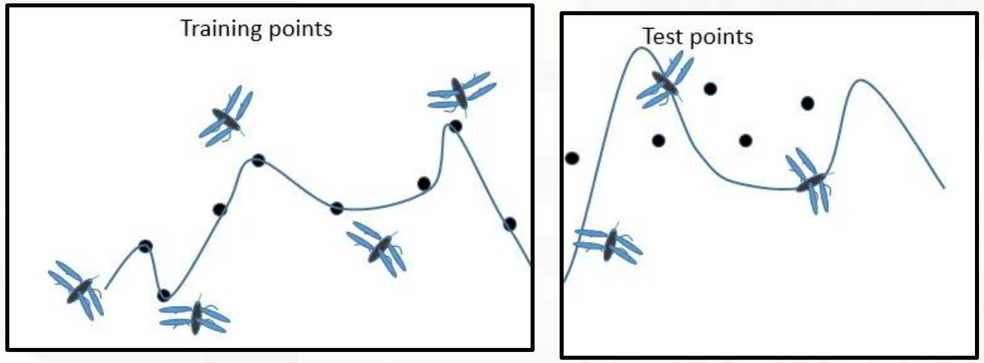


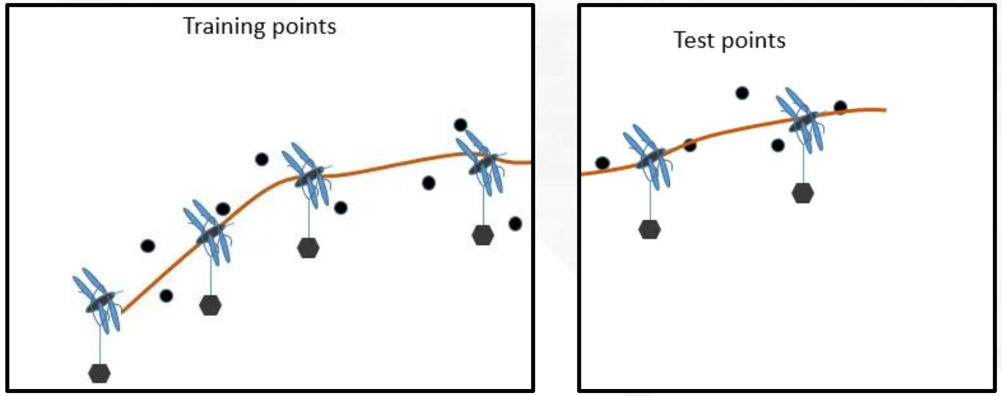


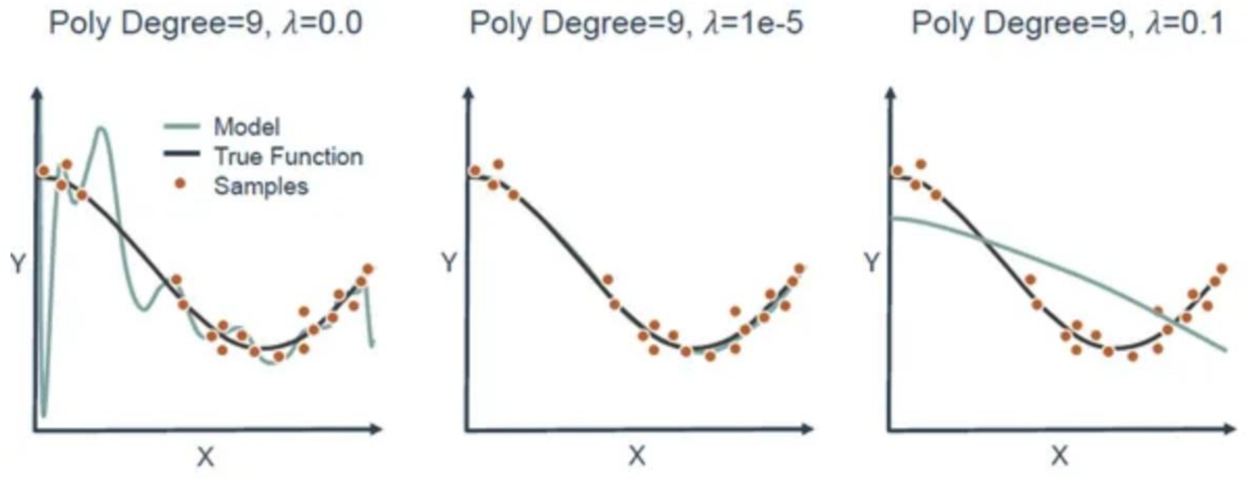
Overfitting: Regularization







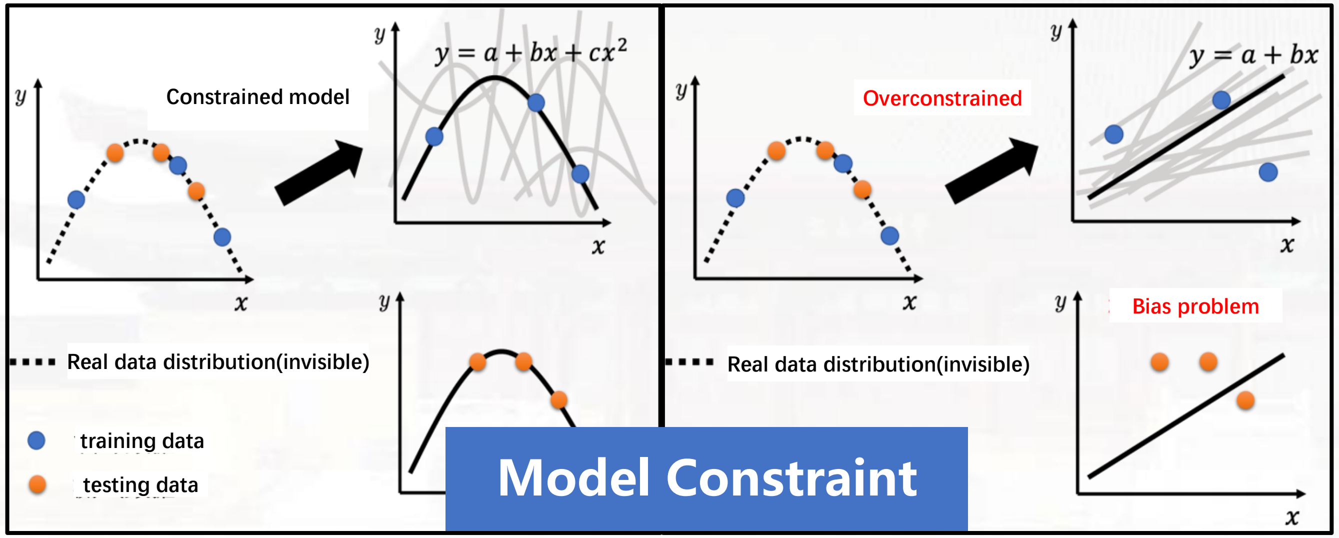


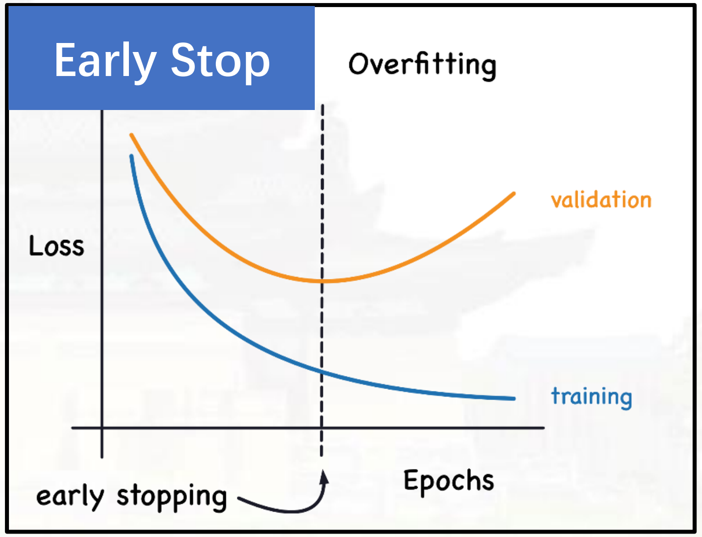


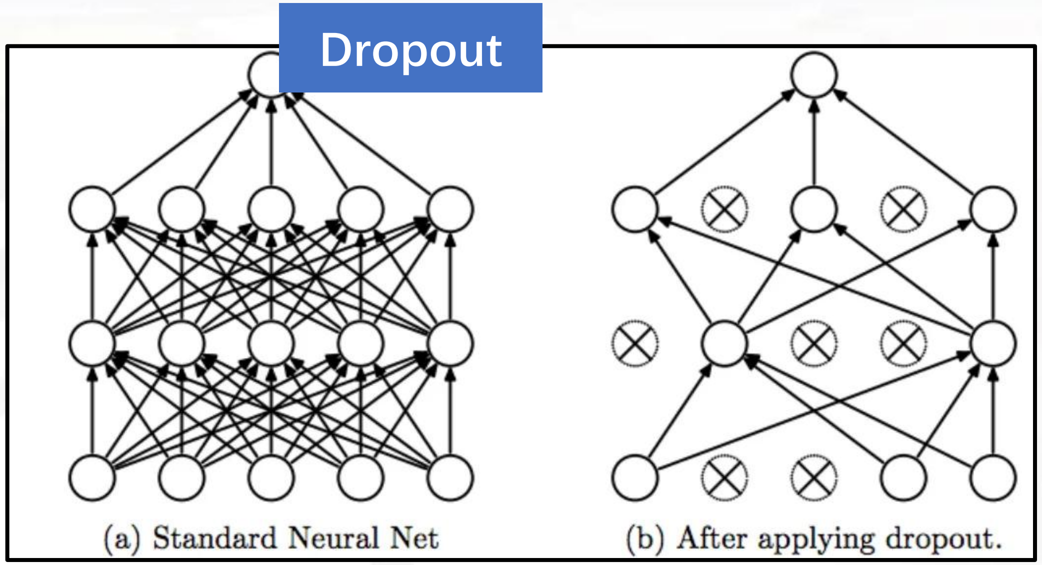
$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda \sum_{j=1}^{k} {\beta_j}^2$$

Overfitting

ZHEJIANG UNIVERSITY





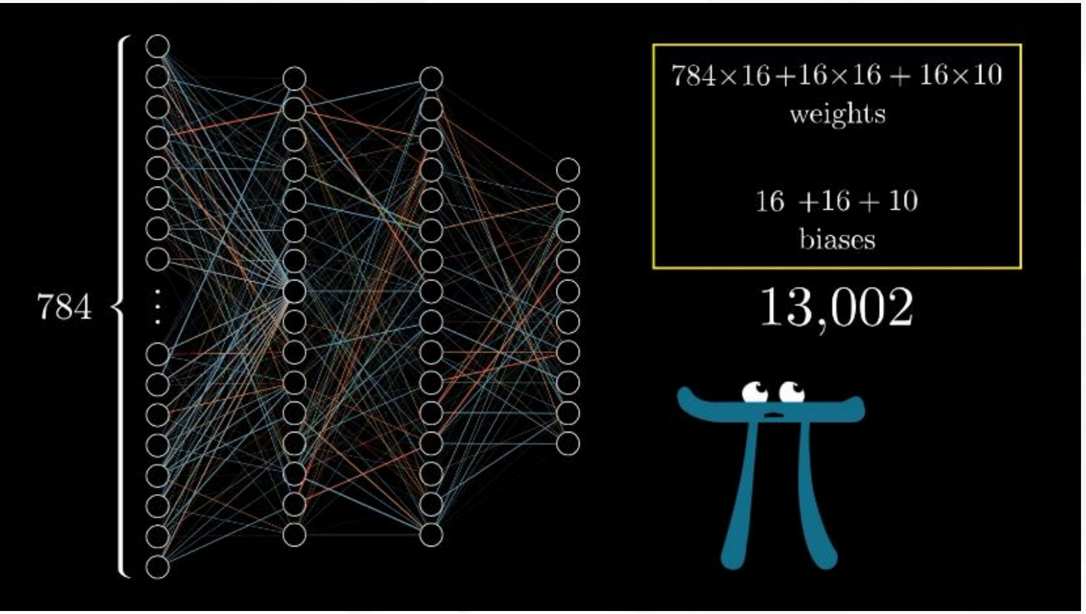


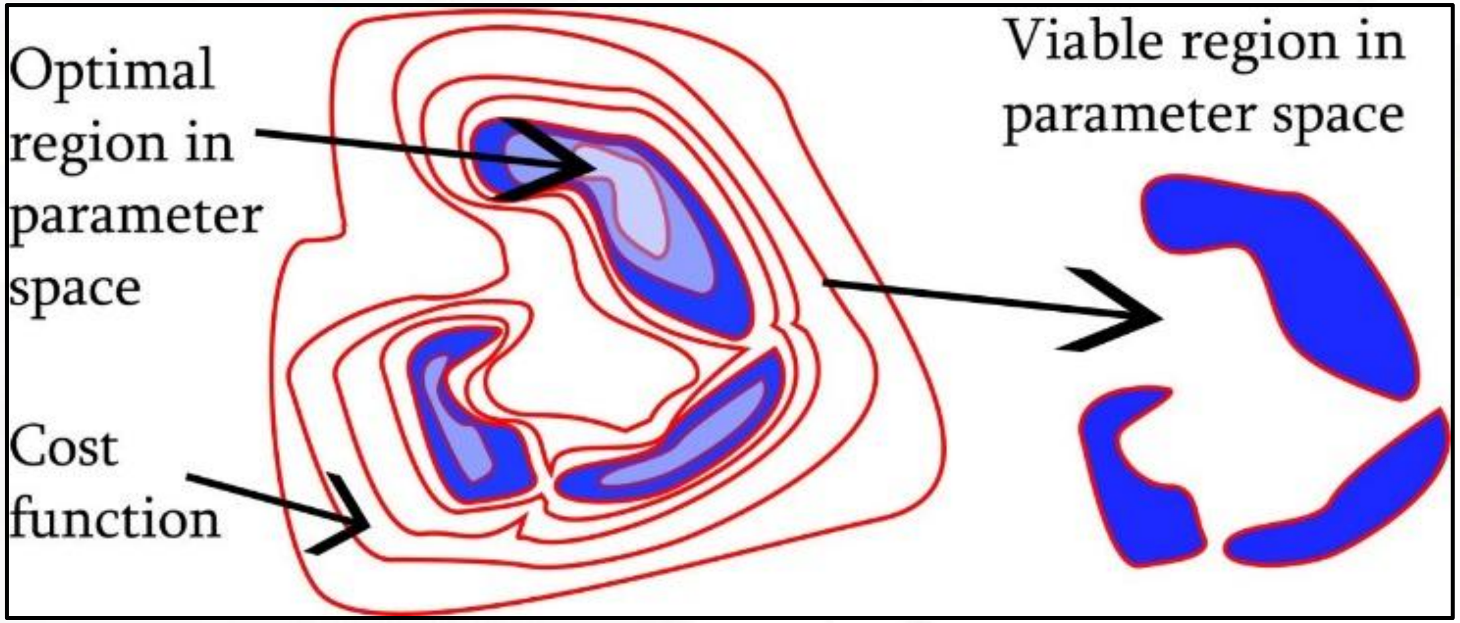
- **Solutions to overfitting**
  - Data augmentation
  - Model constraints: regularization, architecture, dropout
  - Early stop

Architectures

The Problem of MLP

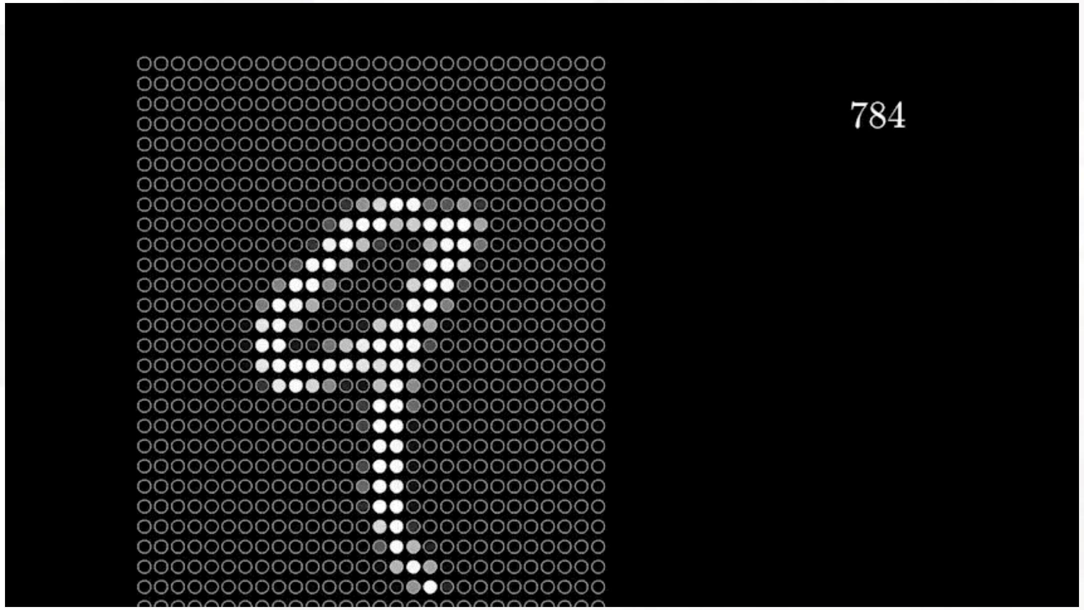
- **Numerous Parameters**
  - Poor Explanation due to Large Scale Minimum
  - Large Scale Minimum due to Large Scale Parameters

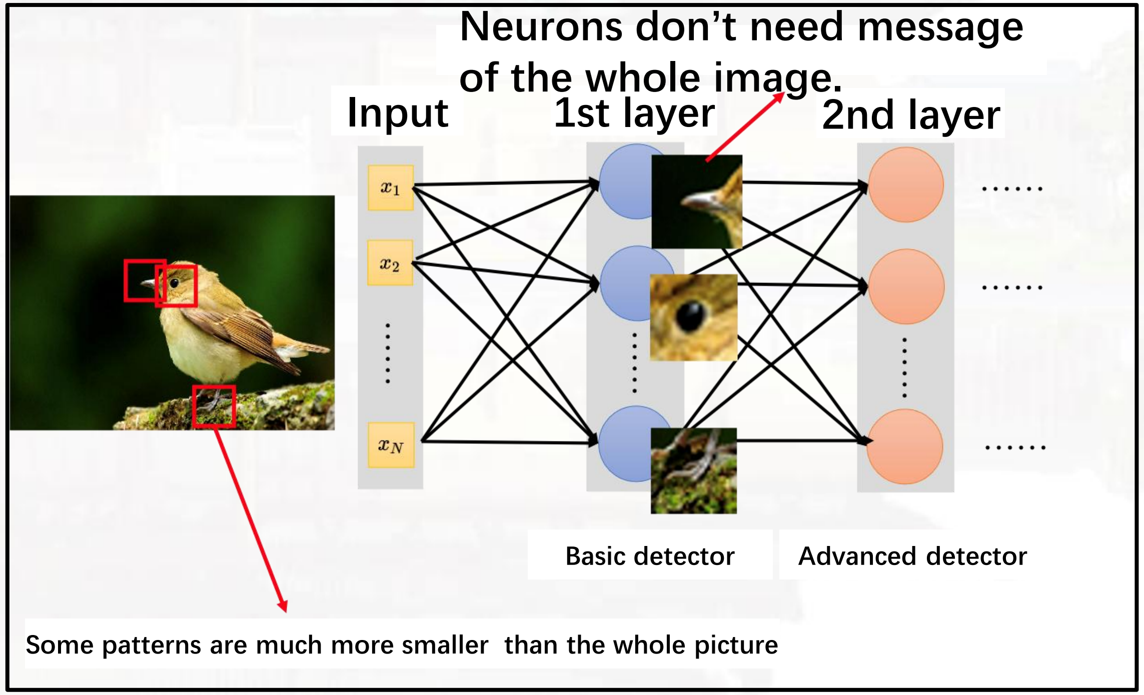




The Problem of MLP

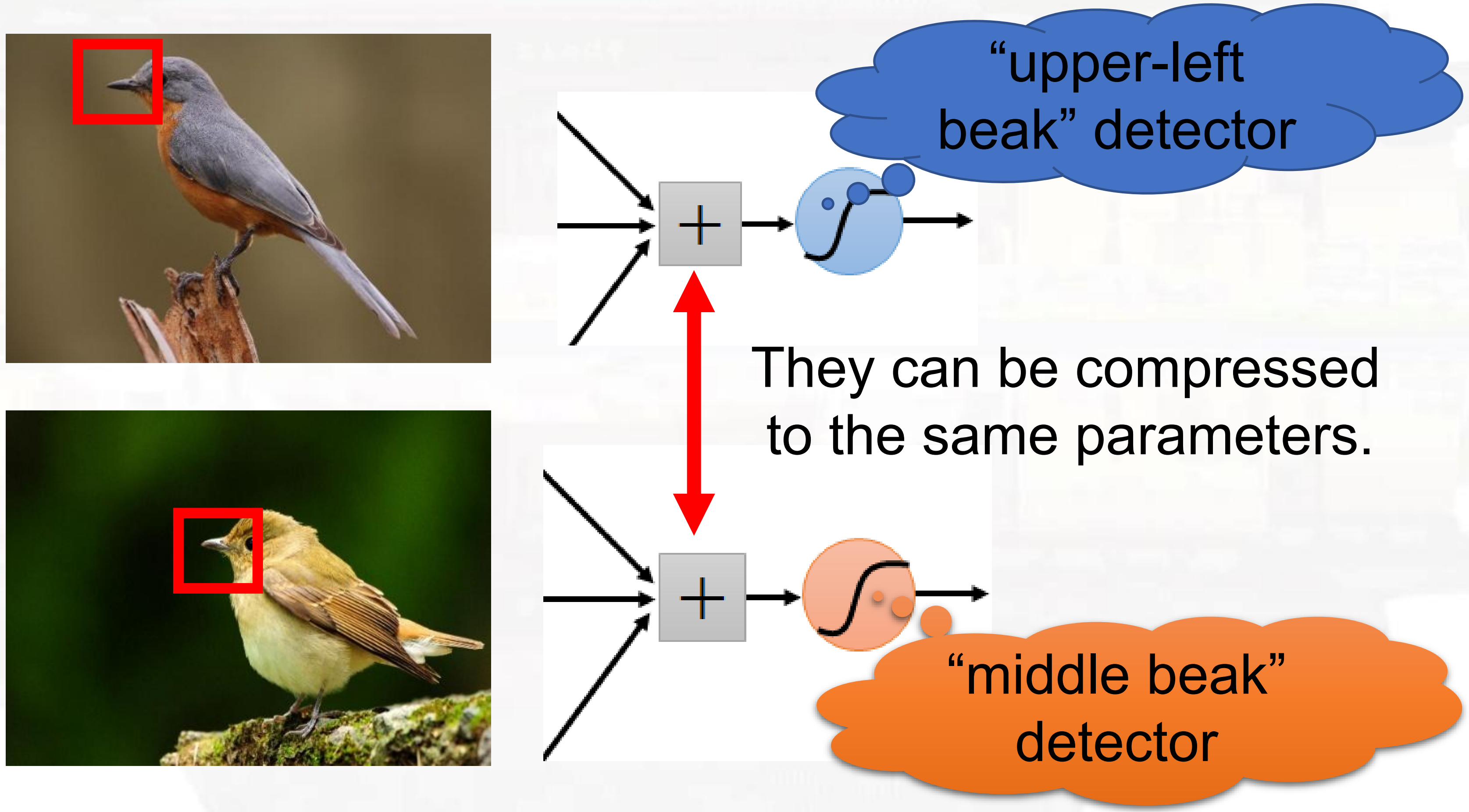
■ Poor Flexibility



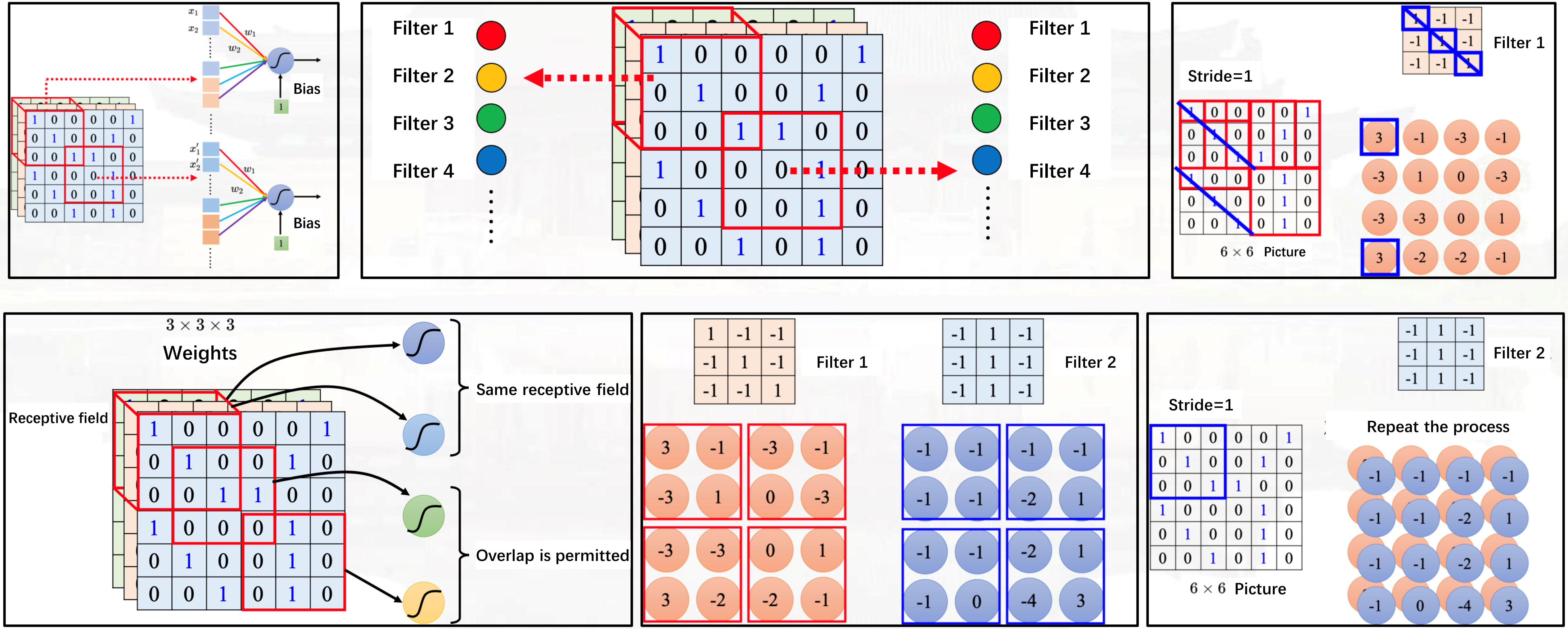


The Problem of MLP

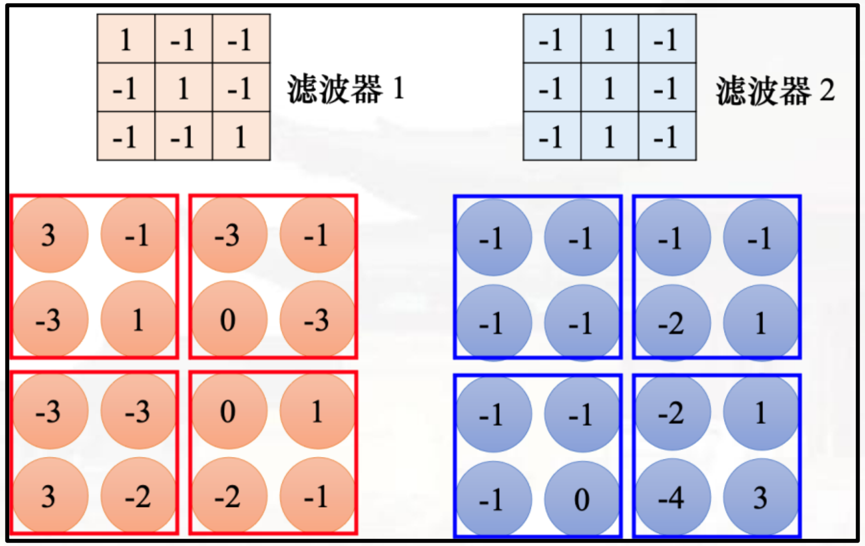
- **Poor Flexibility**

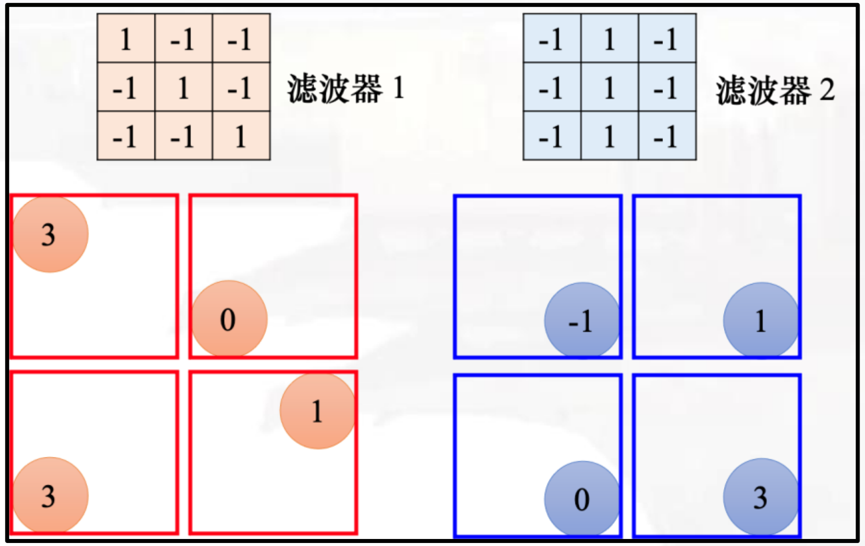


From MLP to CNN



Pooling and Stride



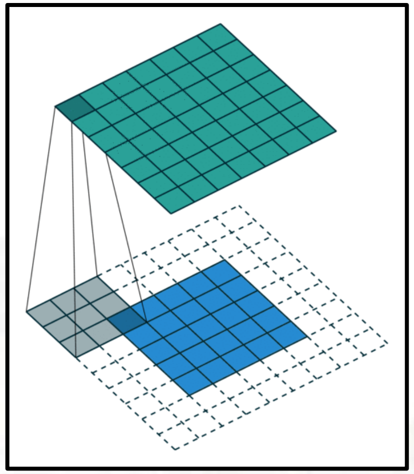


stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product

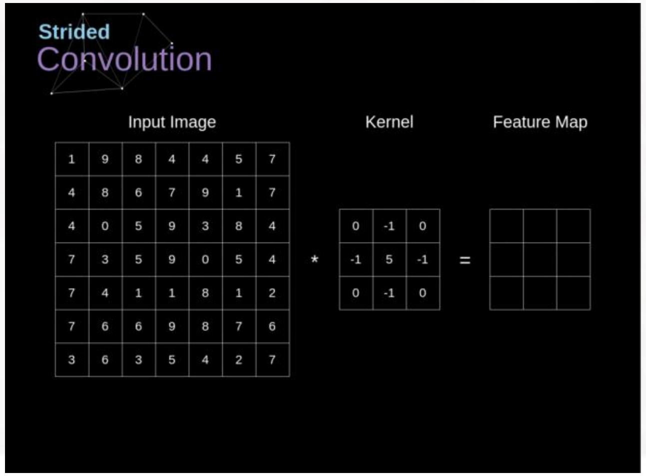
3 -1



If stride=2

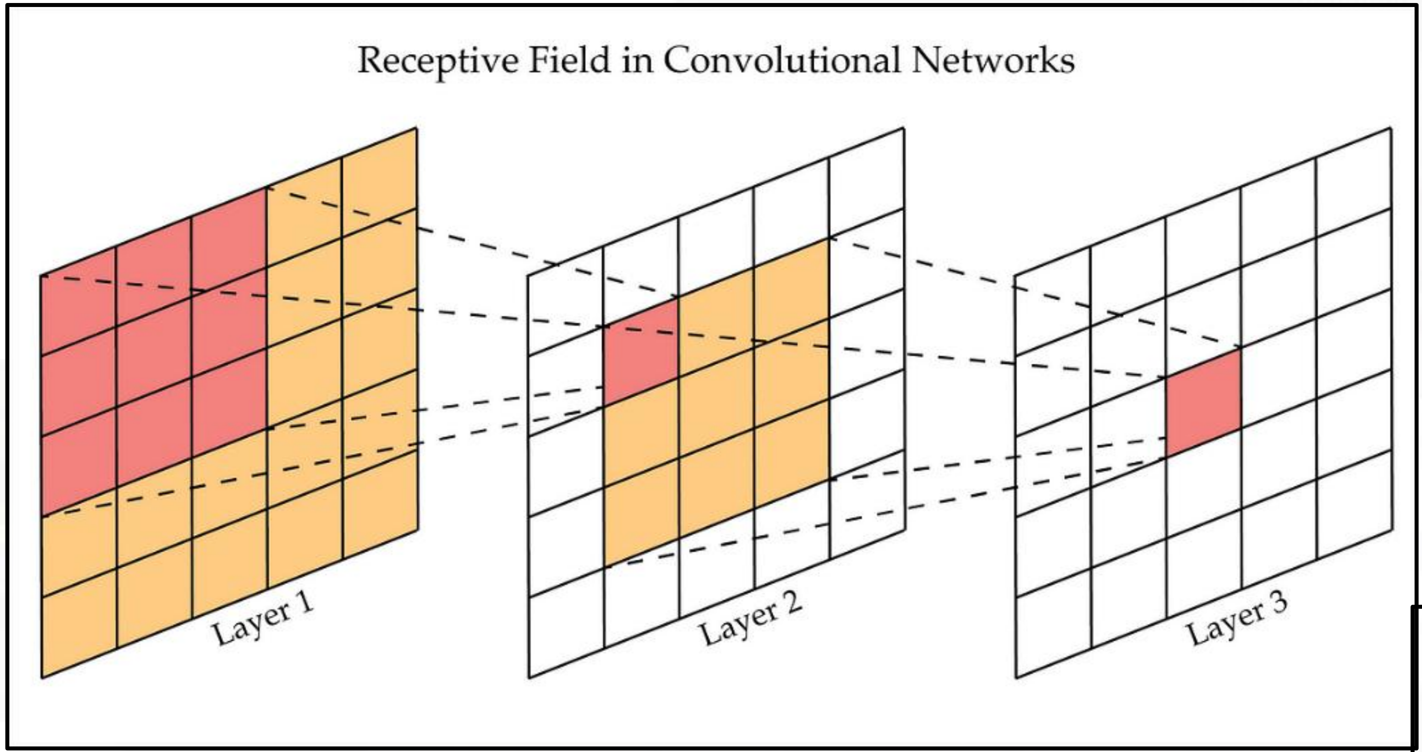
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

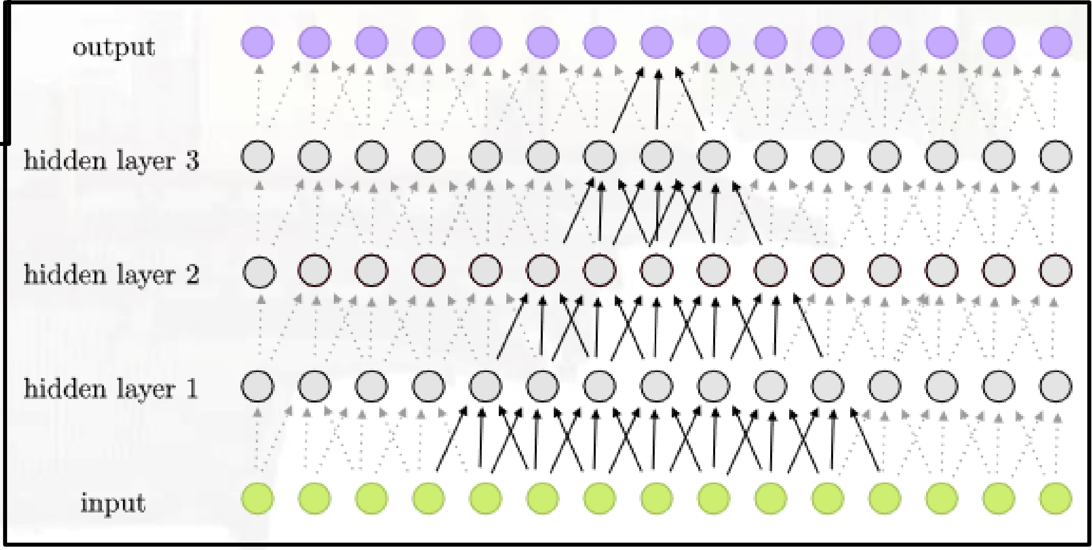
3 -3



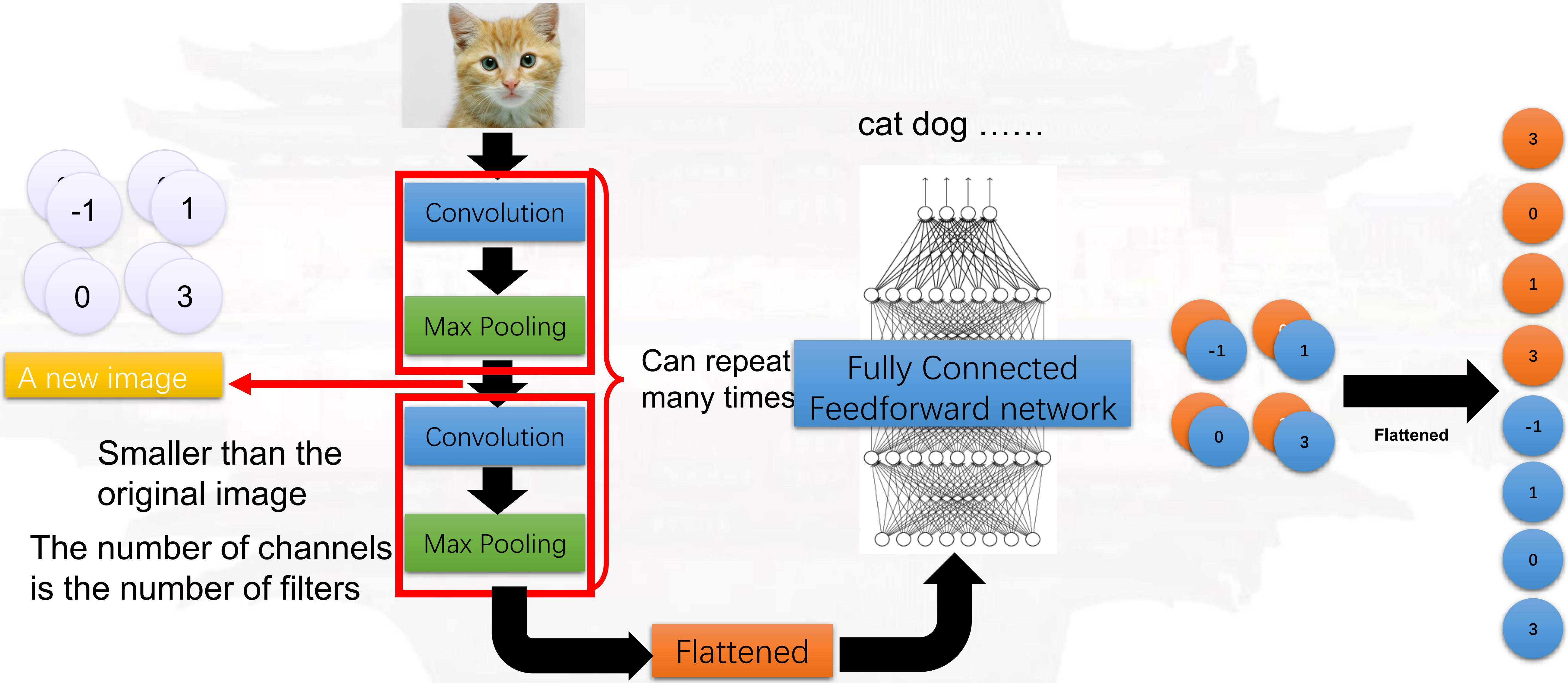
6 x 6 image

Increase Receptive Field



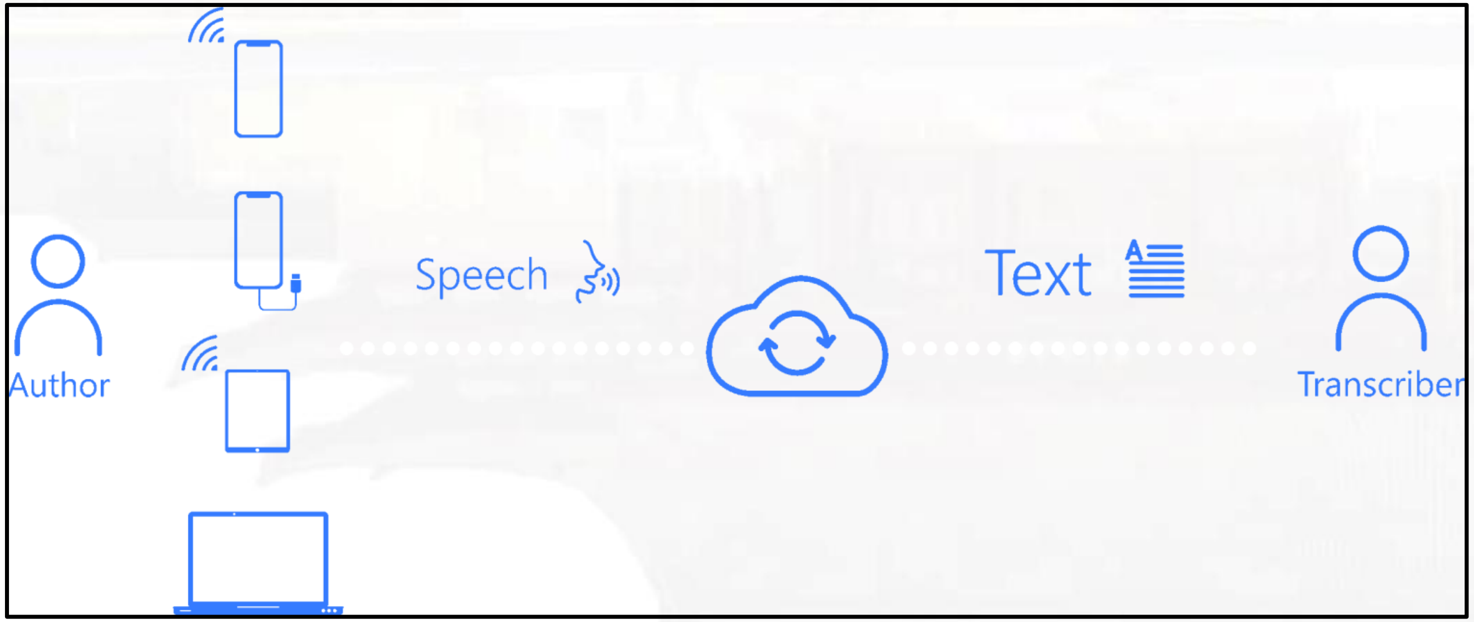


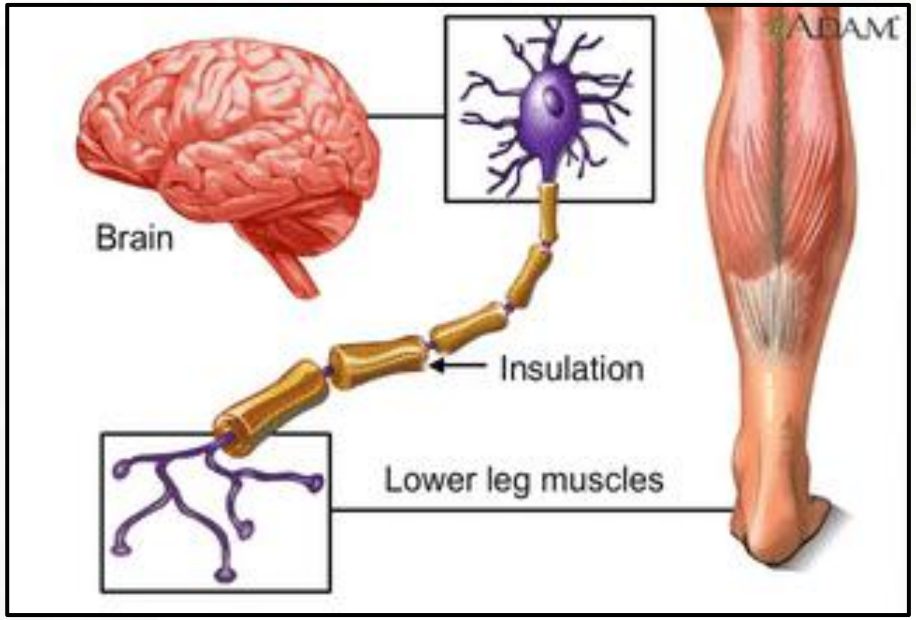
Convolutional Neural Networks



Feedforward Neural Networks

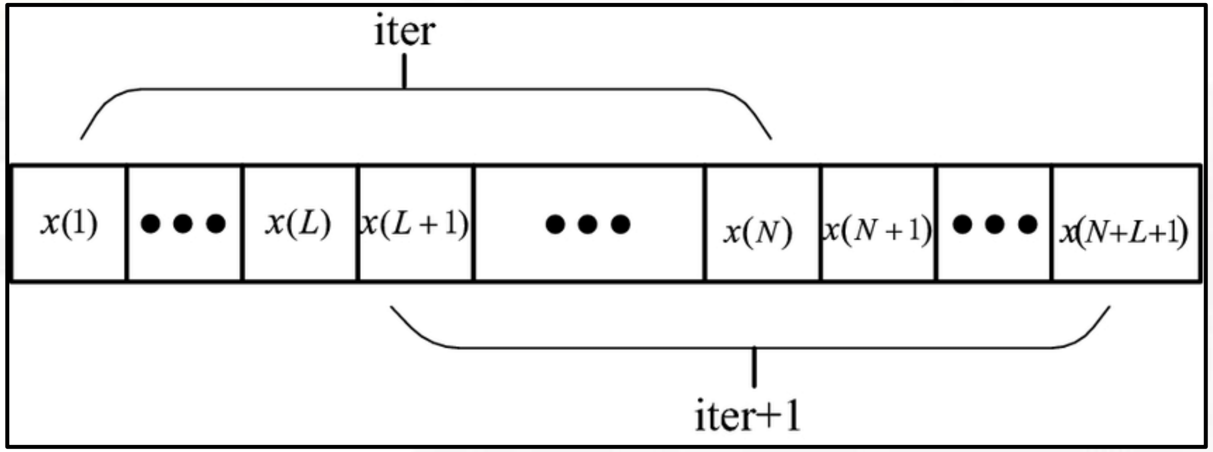
- A feedforward neural network (FNN) is one of the two broad types of artificial neural network, characterized by direction of the flow of information between its layers
- Human brain deals with information streams. Most data is obtained, processed, and generated sequentially.

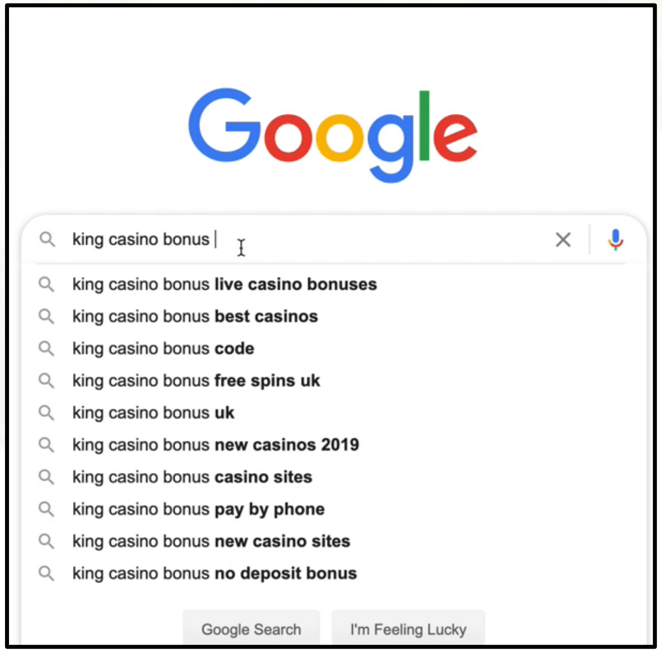




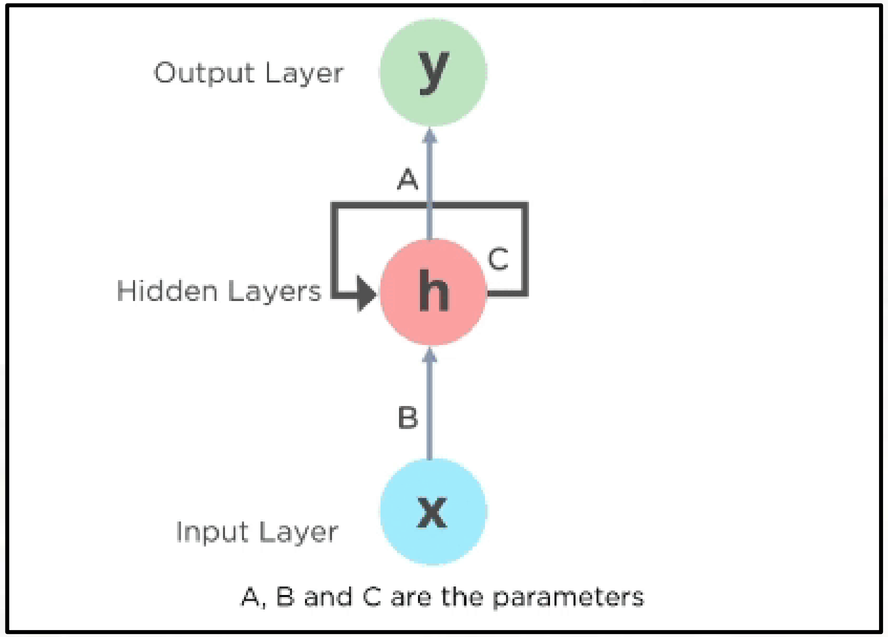
The Problem of FNNs

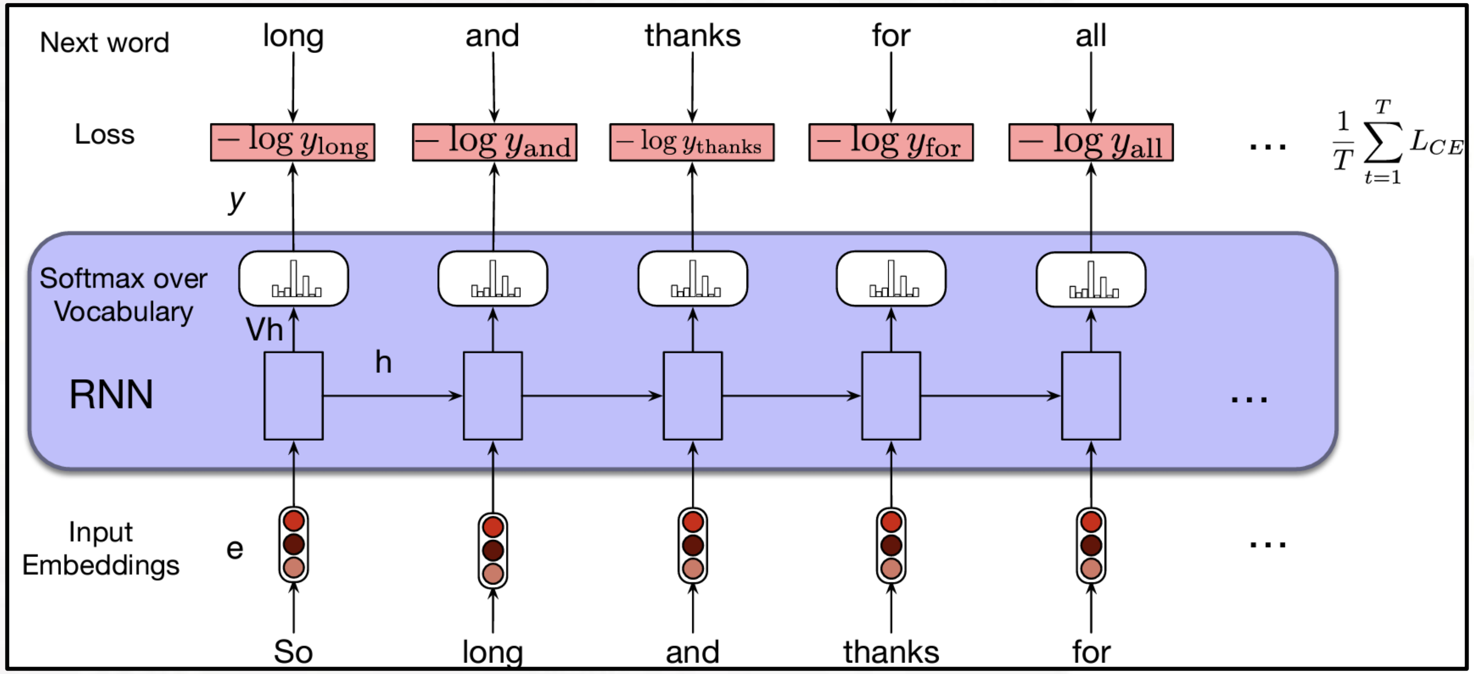
- Human thoughts have persistence; humans don't start their thinking from scratch every second.
- The applications of standard FNNs are limited due to:
  - They only accepted a fixed-size vector as input (e.g., an image) and produce a fixed-size vector as output
  - These models use a fixed amount of computational steps



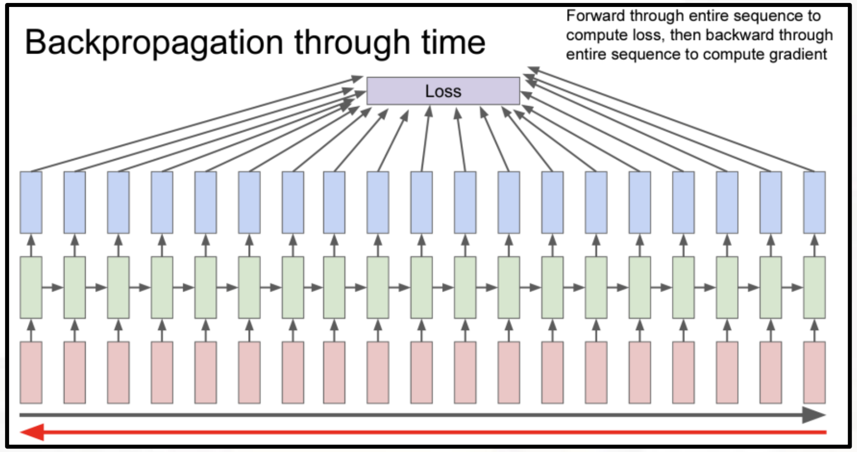


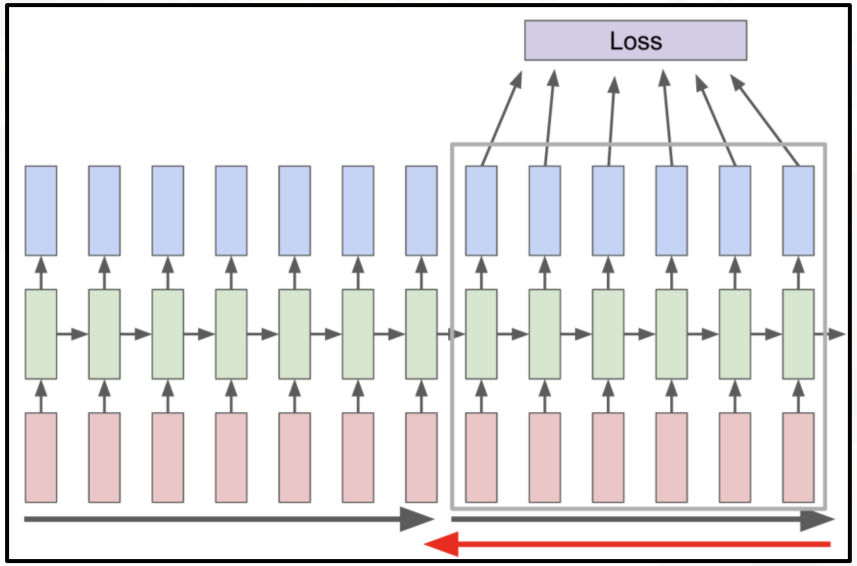
Recurrent Neural Networks

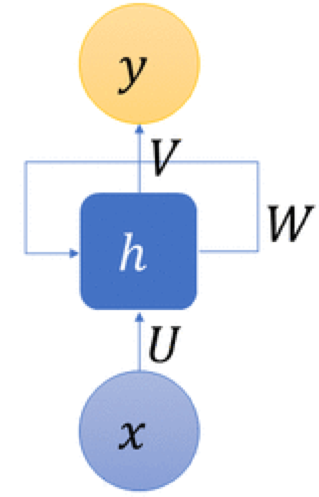




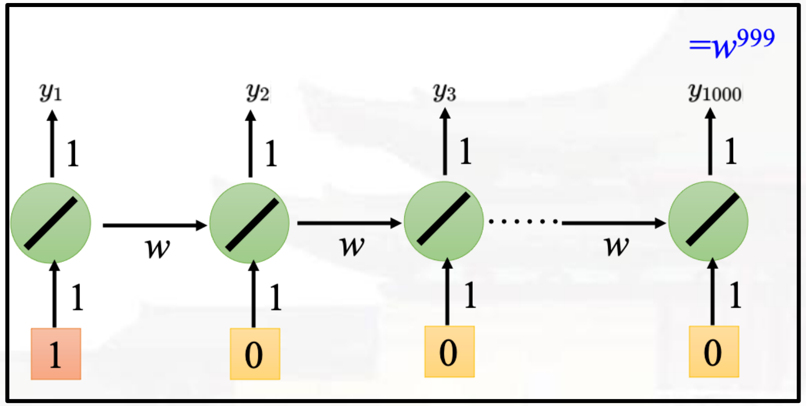
Training RNN







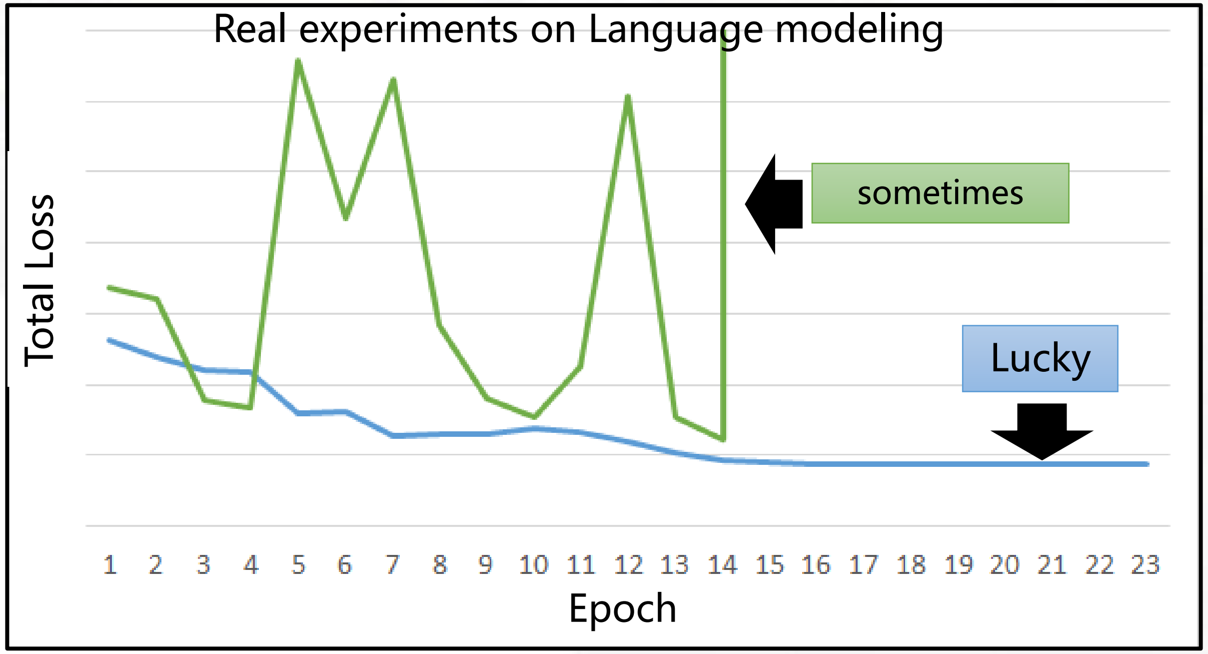
Gradient Vanishing and Exploding

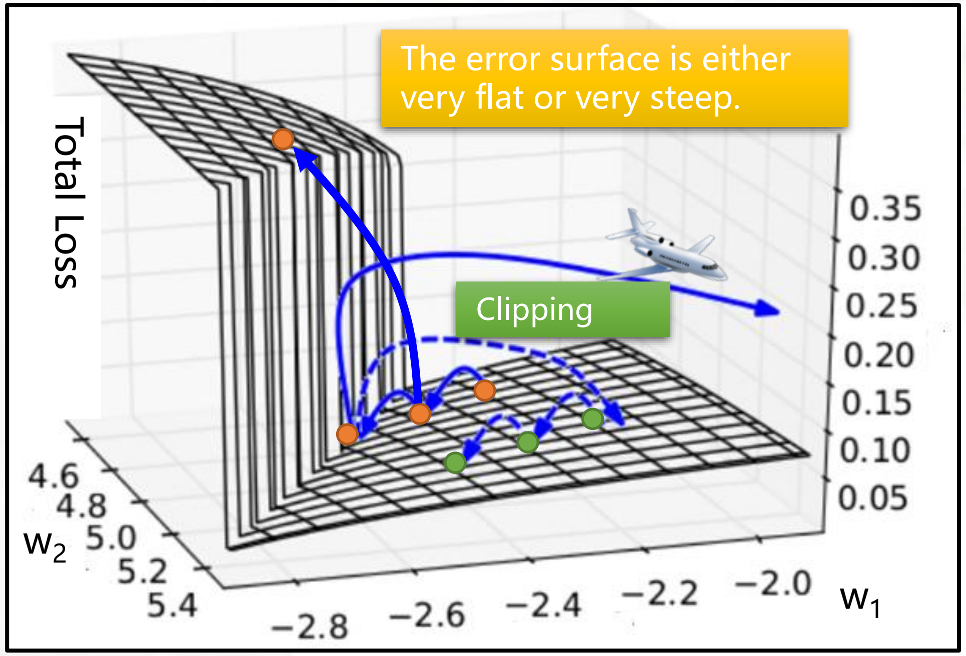


$w = 1$	→	$y^{1000} = 1$
$w = 1.01$	→	$y^{1000} \approx 20000$
$w = 0.99$	→	$y^{1000} \approx 0$
$w = 0.01$	→	$y^{1000} \approx 0$

Large $\partial L / \partial w$ → Small Learning rate?

small $\partial L / \partial w$ → Large Learning rate?





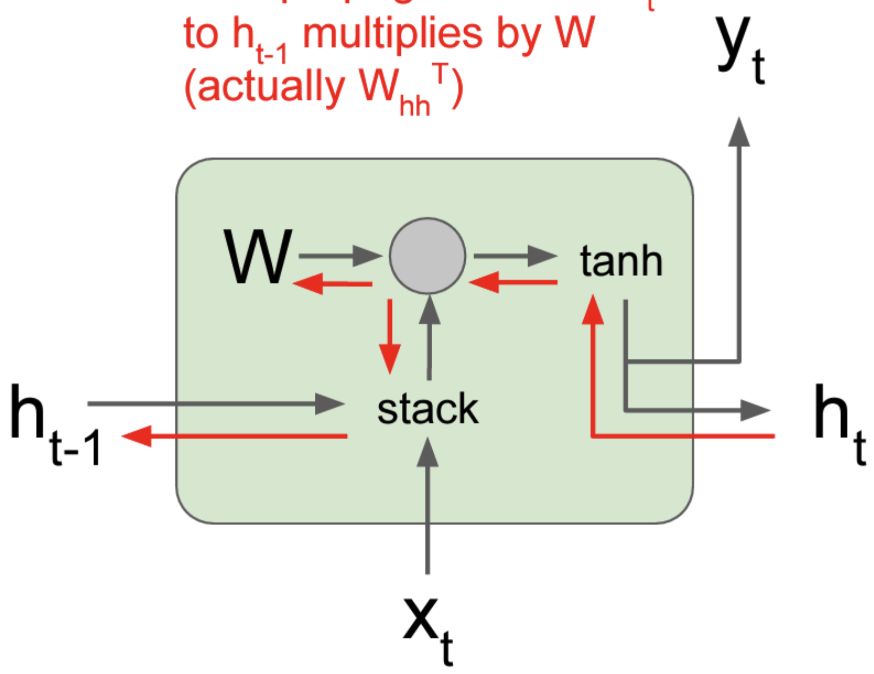
[Razvan Pascanu, ICML' 13]

Gradient Vanishing

Vanilla RNN Gradient Flow

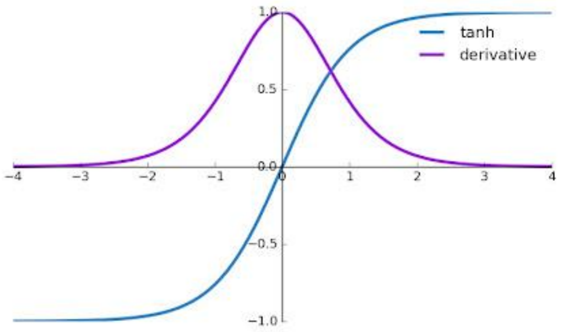
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

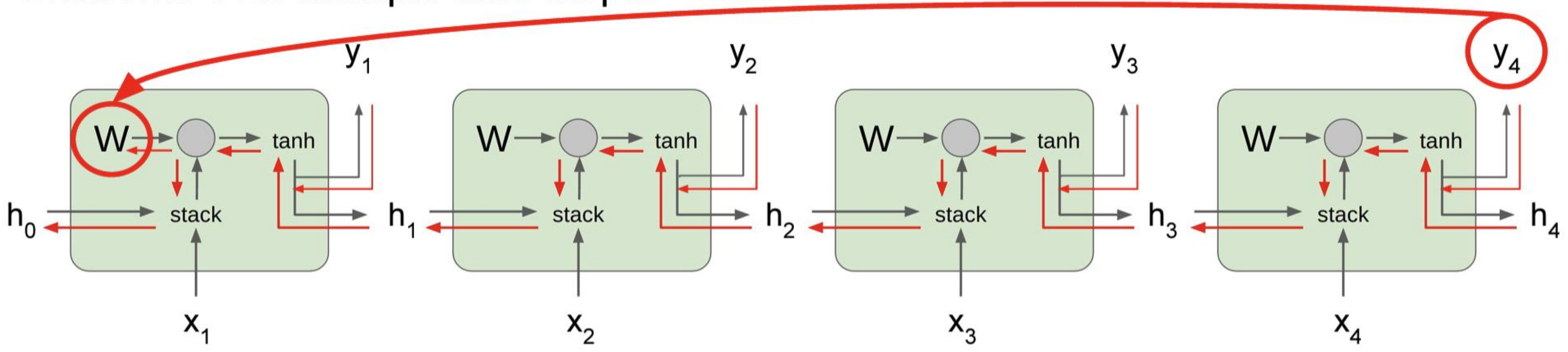


Gradient Vanishing

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

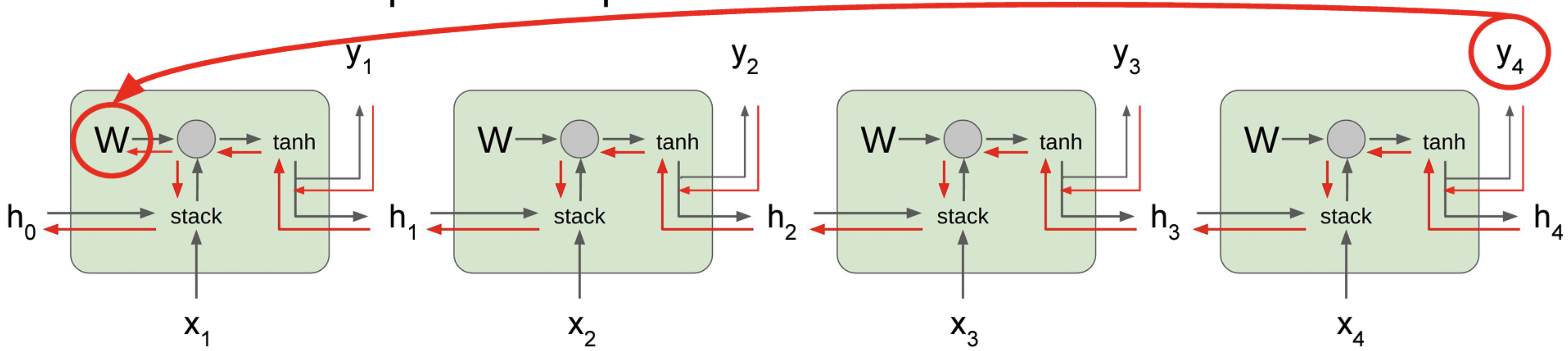
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\frac{\partial h_t}{\partial h_{t-1}}\cdots\frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T}\frac{\partial h_t}{\partial h_{t-1}}\right)\frac{\partial h_1}{\partial W}$$

Gradient Vanishing

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

RNN and LSTM

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

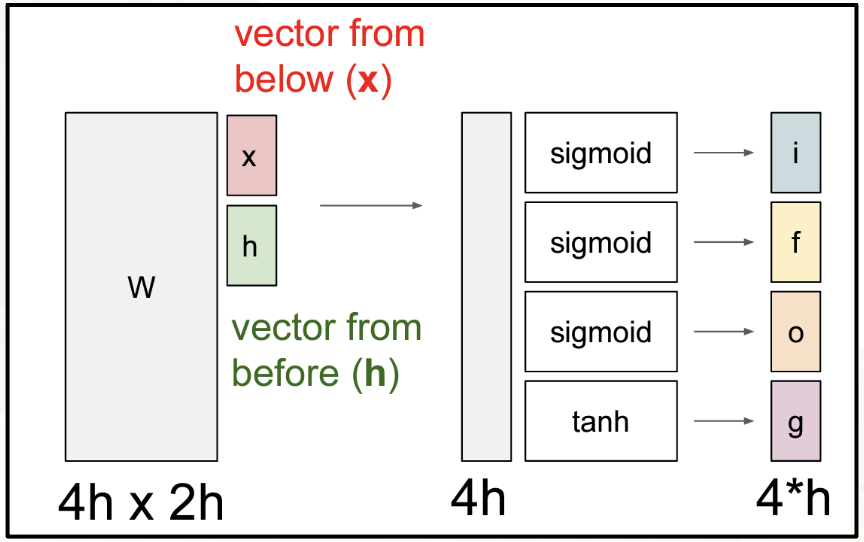
LSTM

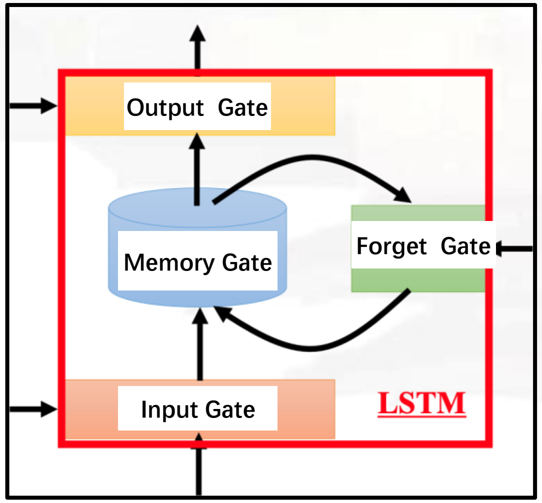
Four gates

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

Cell state → $c_t = f \odot c_{t-1} + i \odot g$

Hidden state → $h_t = o \odot \tanh(c_t)$





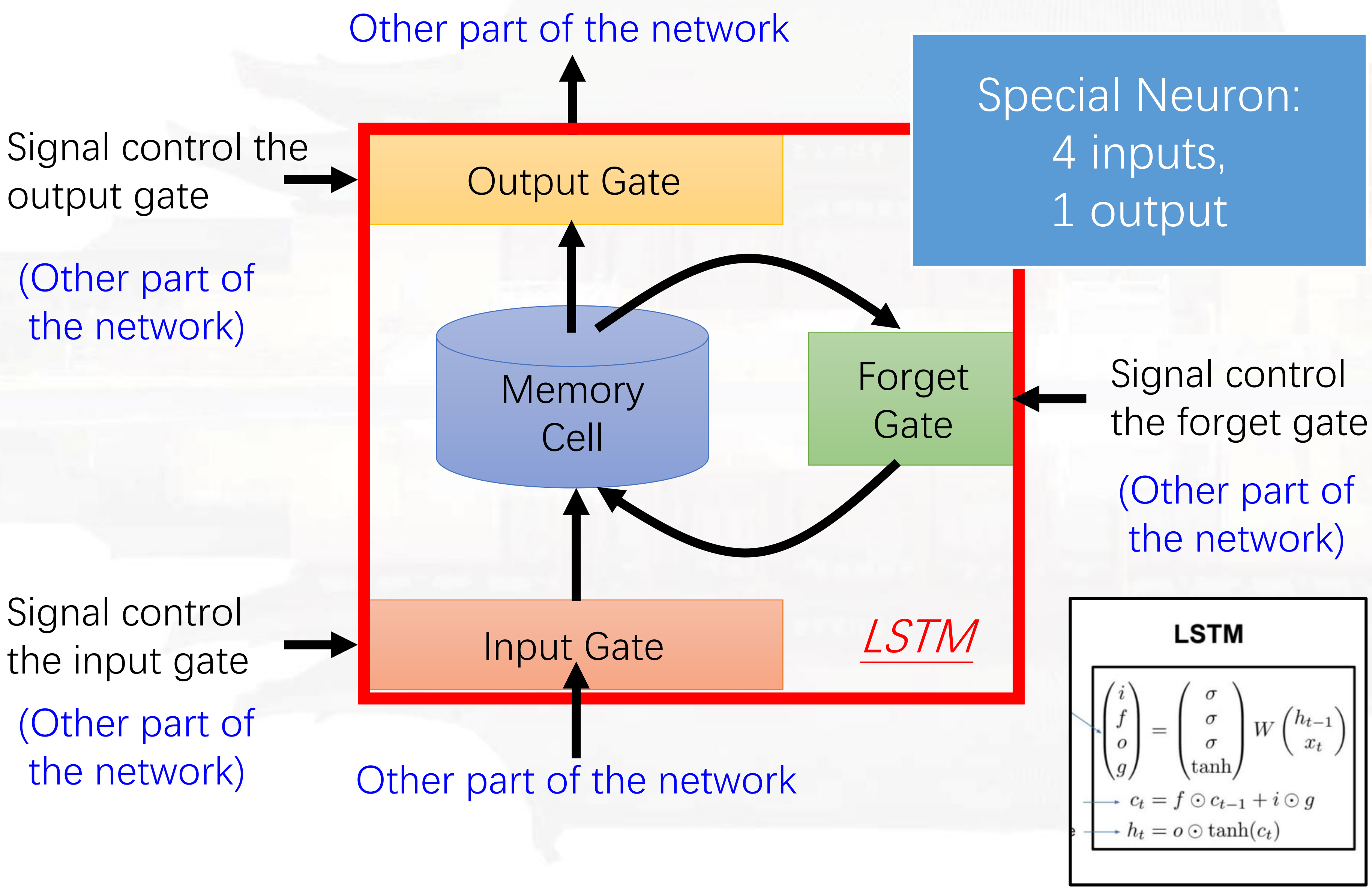
i: Input gate, whether to write to cell

f: Forget gate, Whether to erase cell

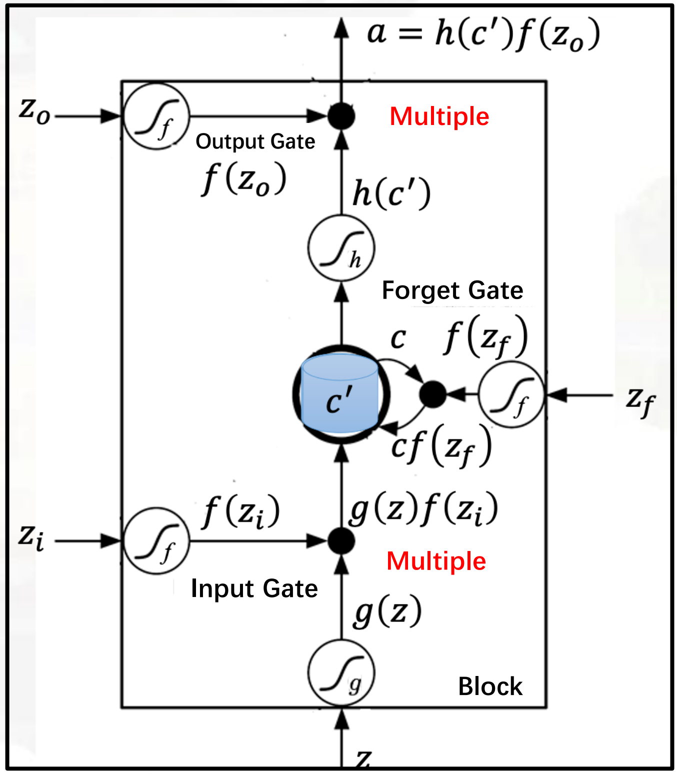
o: Output gate, How much to reveal cell

g: Gate gate (?), How much to write to cell

RNN and LSTM

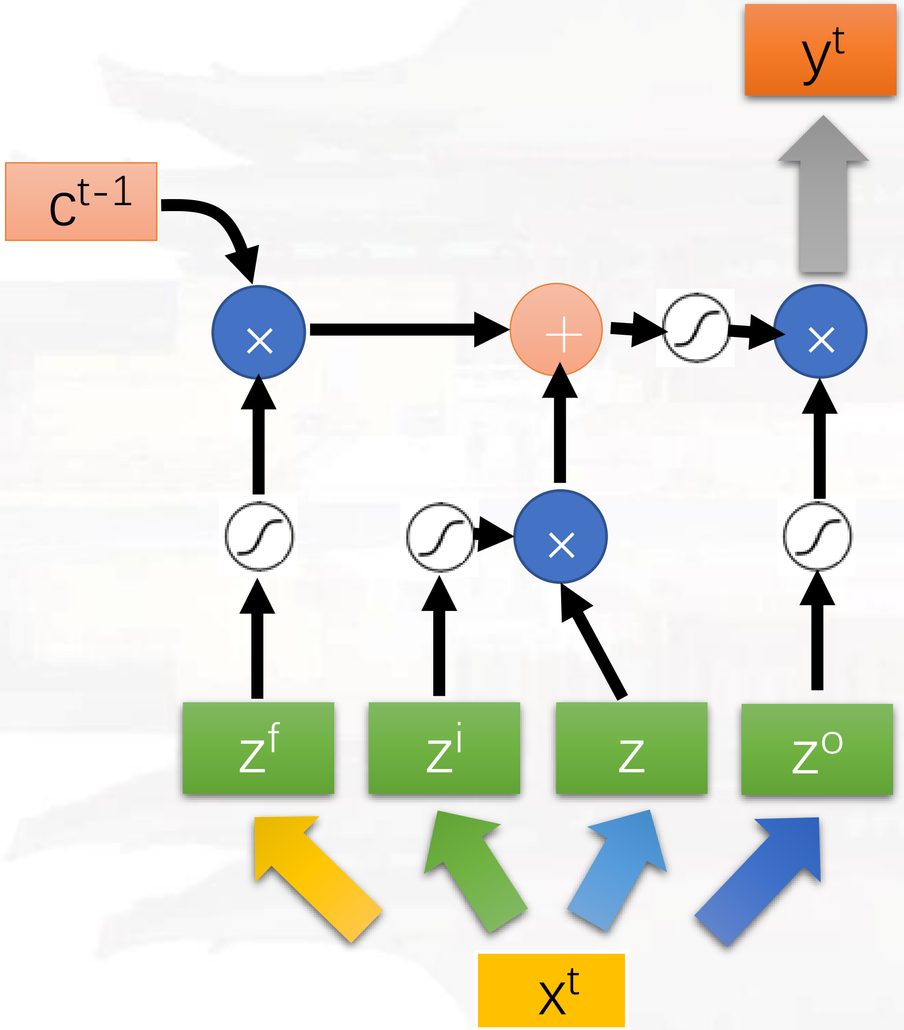


Other part of the network
Signal control the output gate
(Other part of the network)
Output Gate
Memory Cell
Forget Gate
Input Gate
LSTM
Special Neuron: 4 inputs, 1 output
Signal control the forget gate
(Other part of the network)
Signal control the input gate
(Other part of the network)
Other part of the network
LSTM
$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$
$c_t = f \odot c_{t-1} + i \odot g$
$h_t = o \odot \tanh(c_t)$

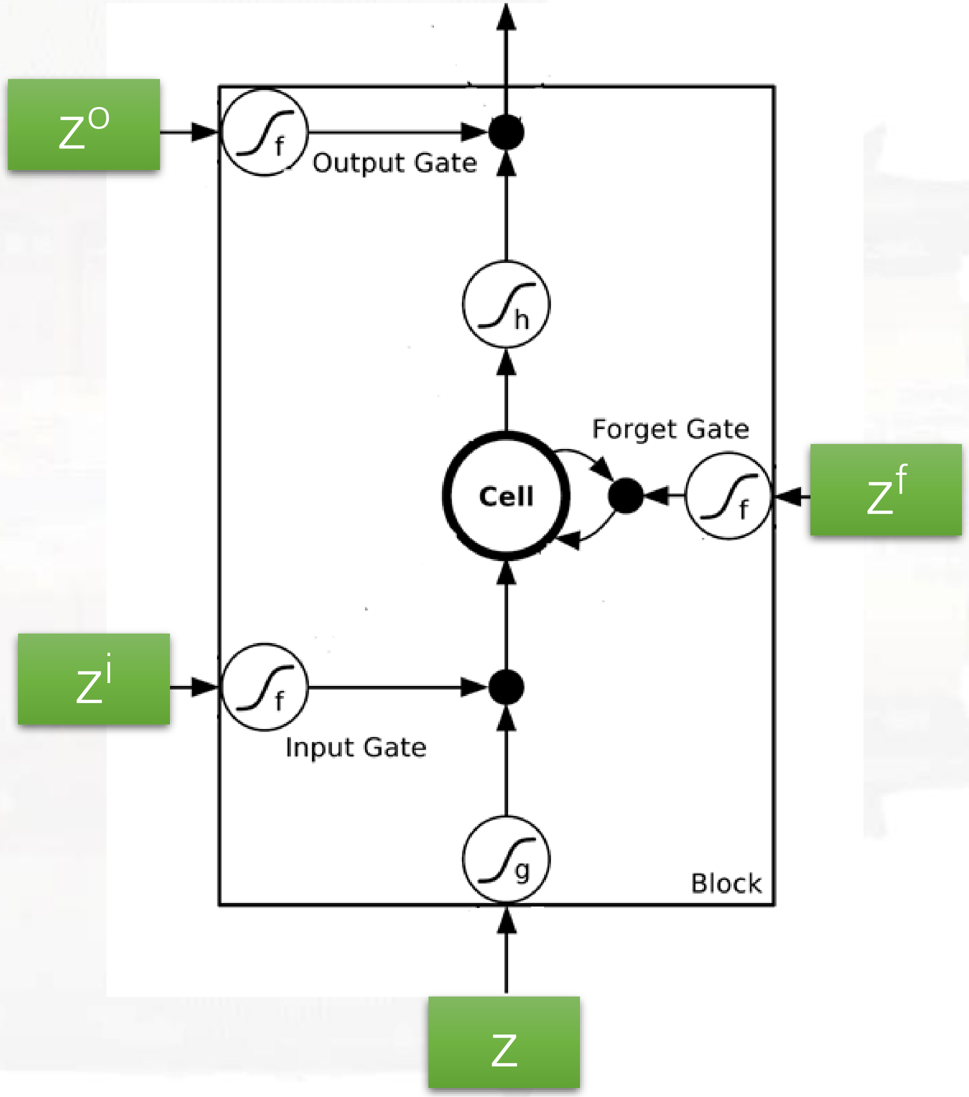


$a = h(c')f(z_o)$
z_o
Multiple
Output Gate
$f(z_o)$
$h(c')$
Forget Gate
c
$f(z_f)$
c'
z_f
$cf(z_f)$
$f(z_i)$
$g(z)f(z_i)$
z_i
Multiple
Input Gate
$g(z)$
Block
z

RNN and LSTM

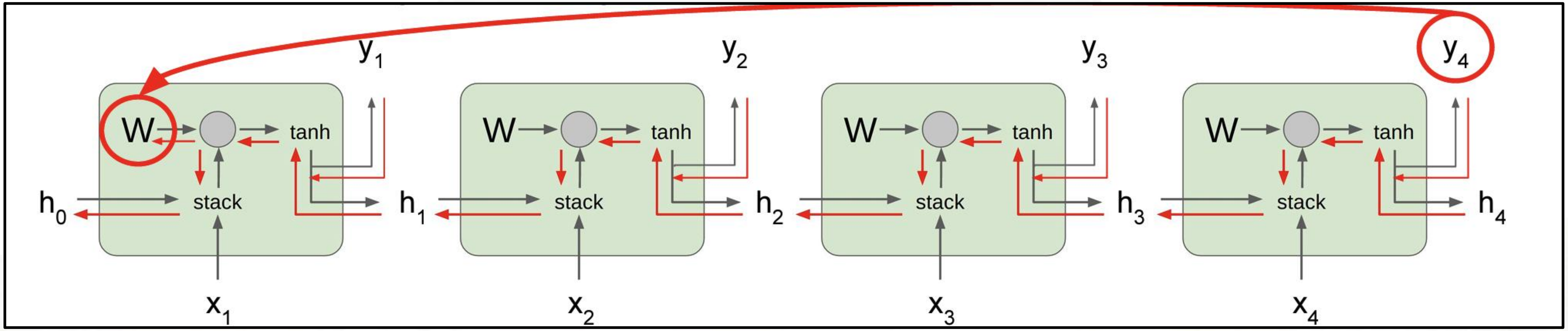


y^t
c^{t-1}
z^f
z^i
z
z^o
x^t

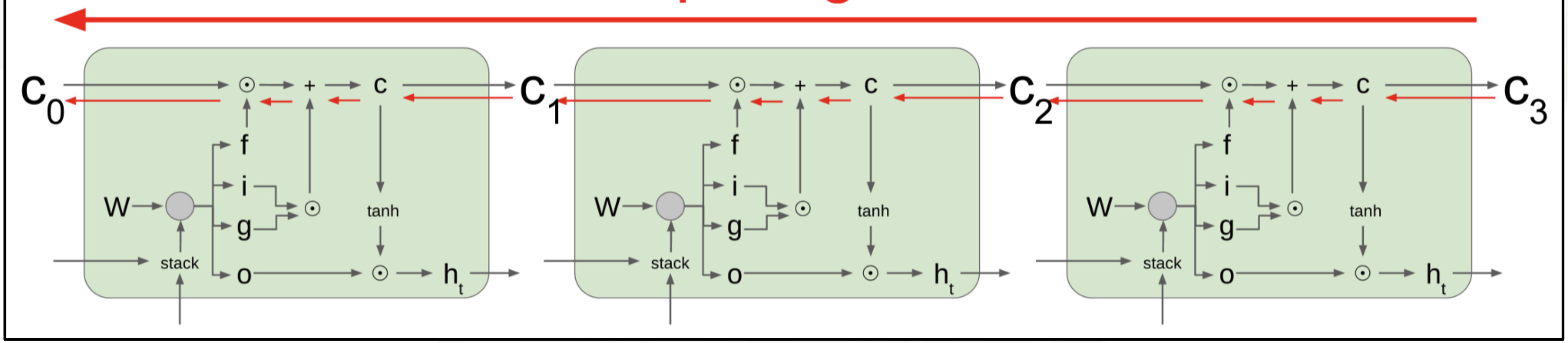


z^o
Output Gate
Forget Gate
Cell
z^f
z^i
Input Gate
Block
z

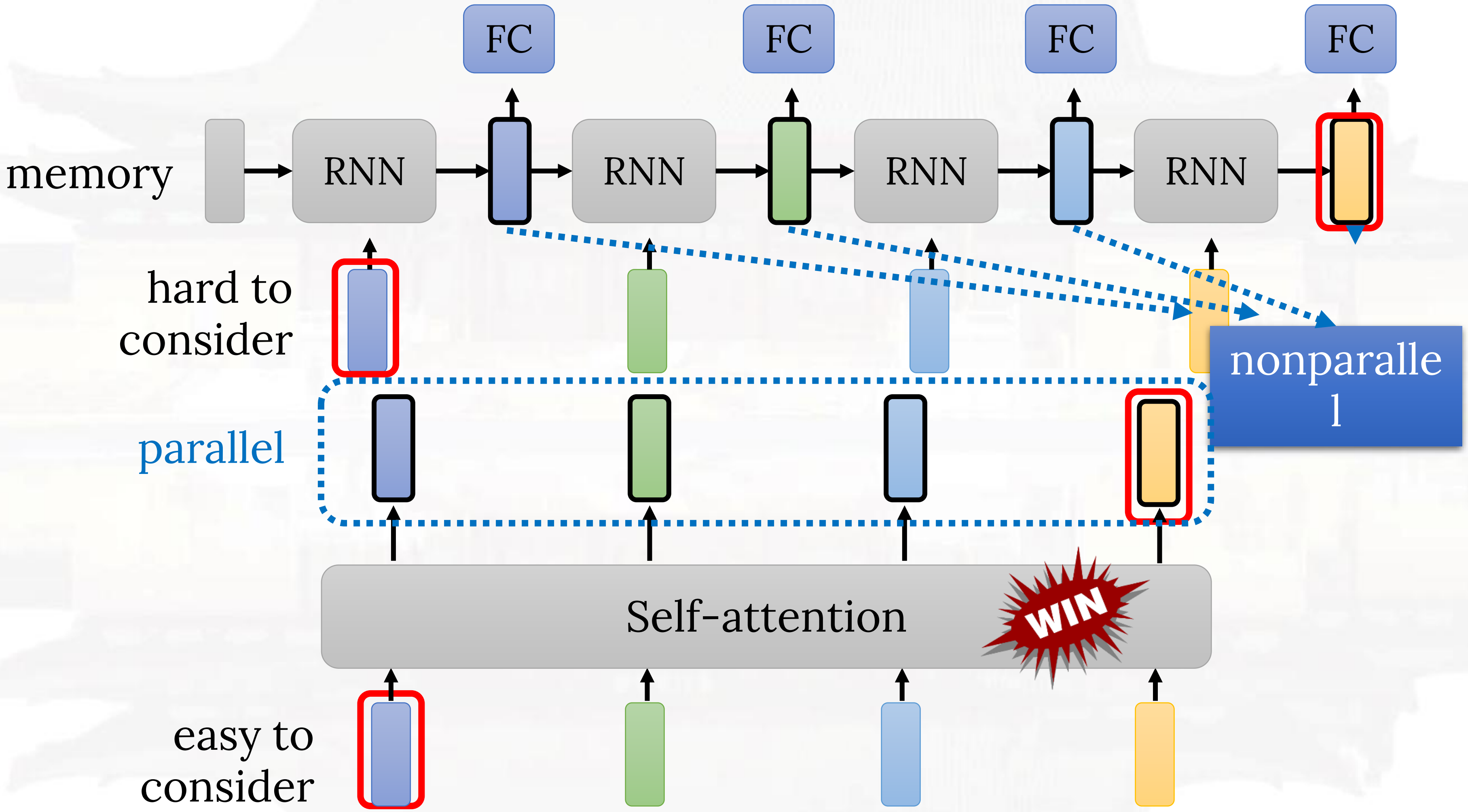
RNN and LSTM



Uninterrupted gradient flow!



From RNN to Attention



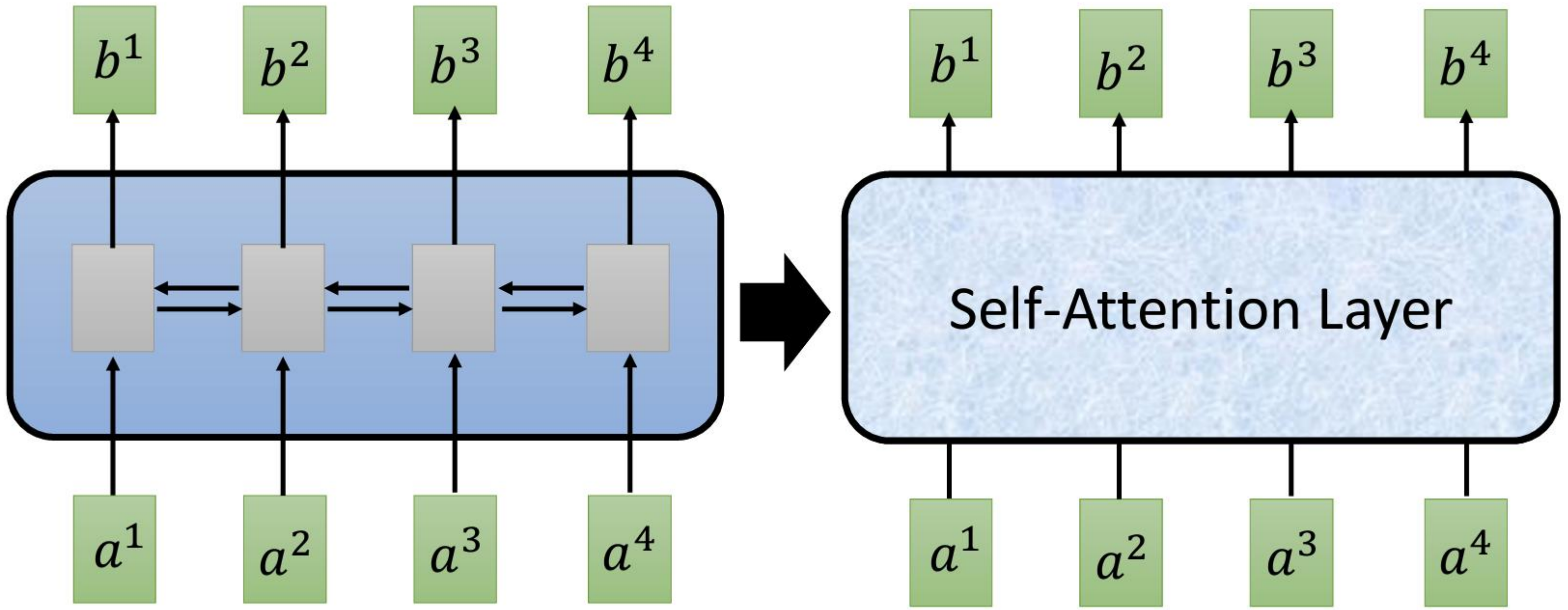
https://arxiv.org/abs/2006.16236

Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention

From RNN to Attention

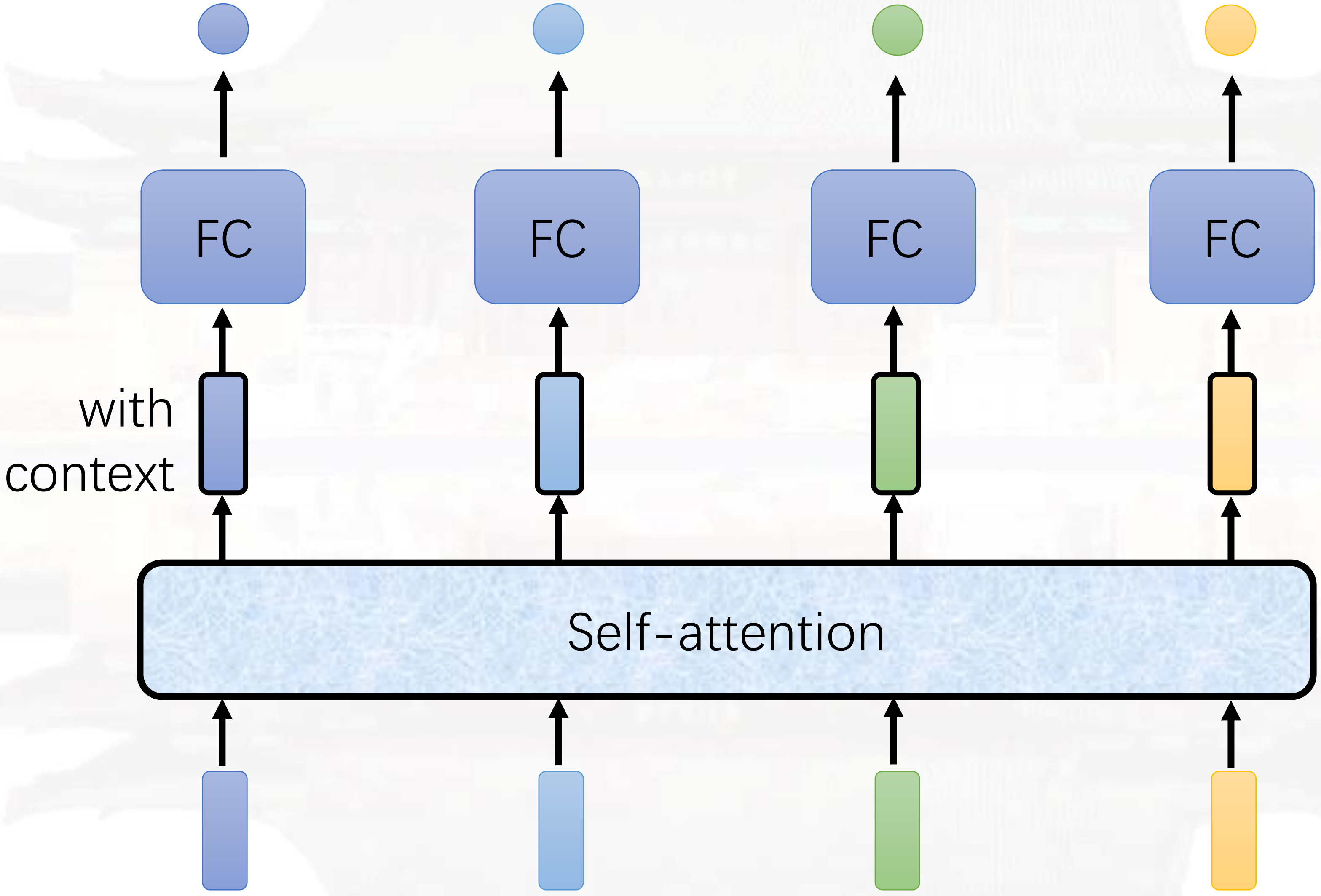
b^i is obtained based on the whole input sequence.

b^1, b^2, b^3, b^4 can be parallelly computed.

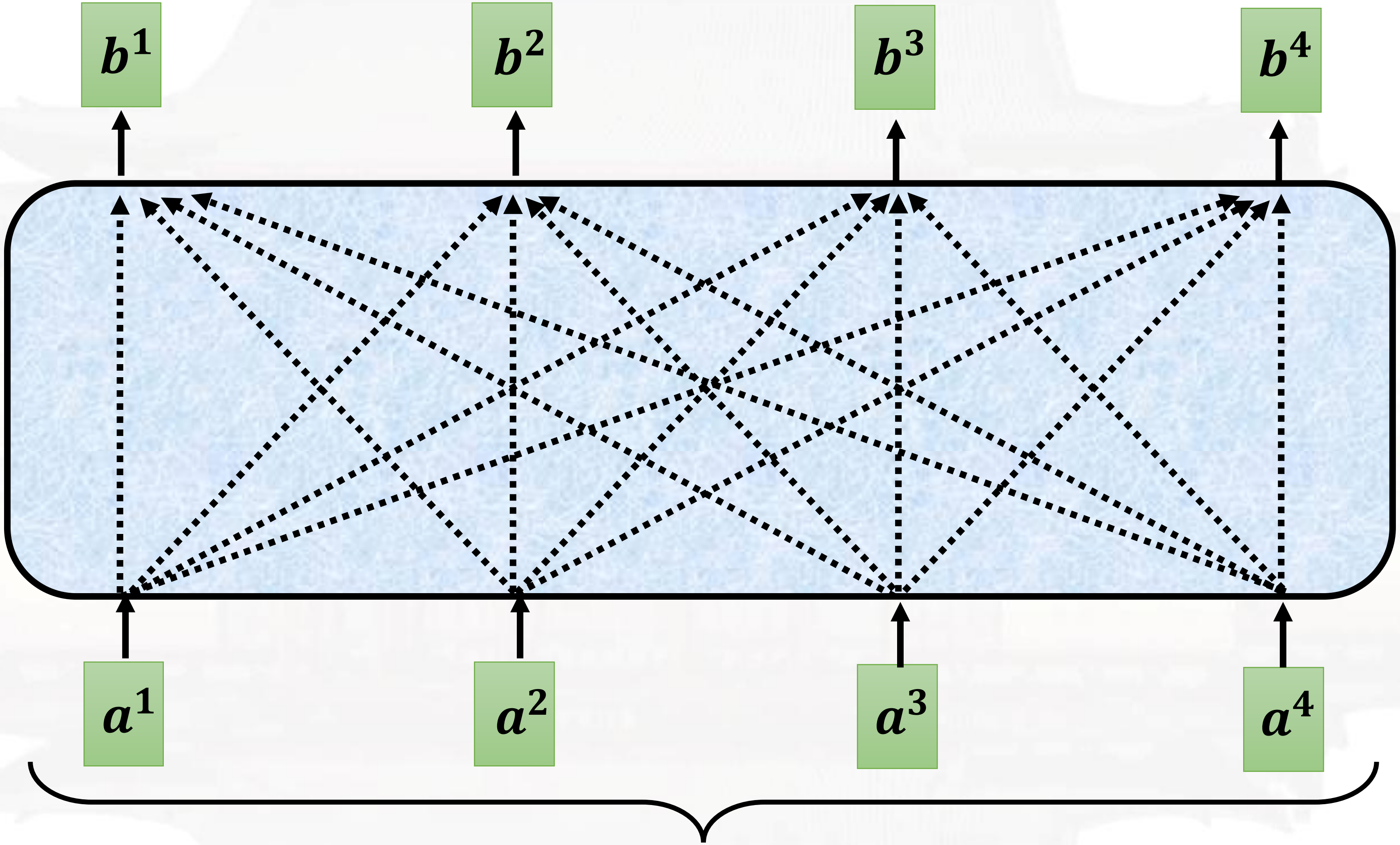


You can try to replace any thing that has been done by RNN with self-attention.

Self-Attention

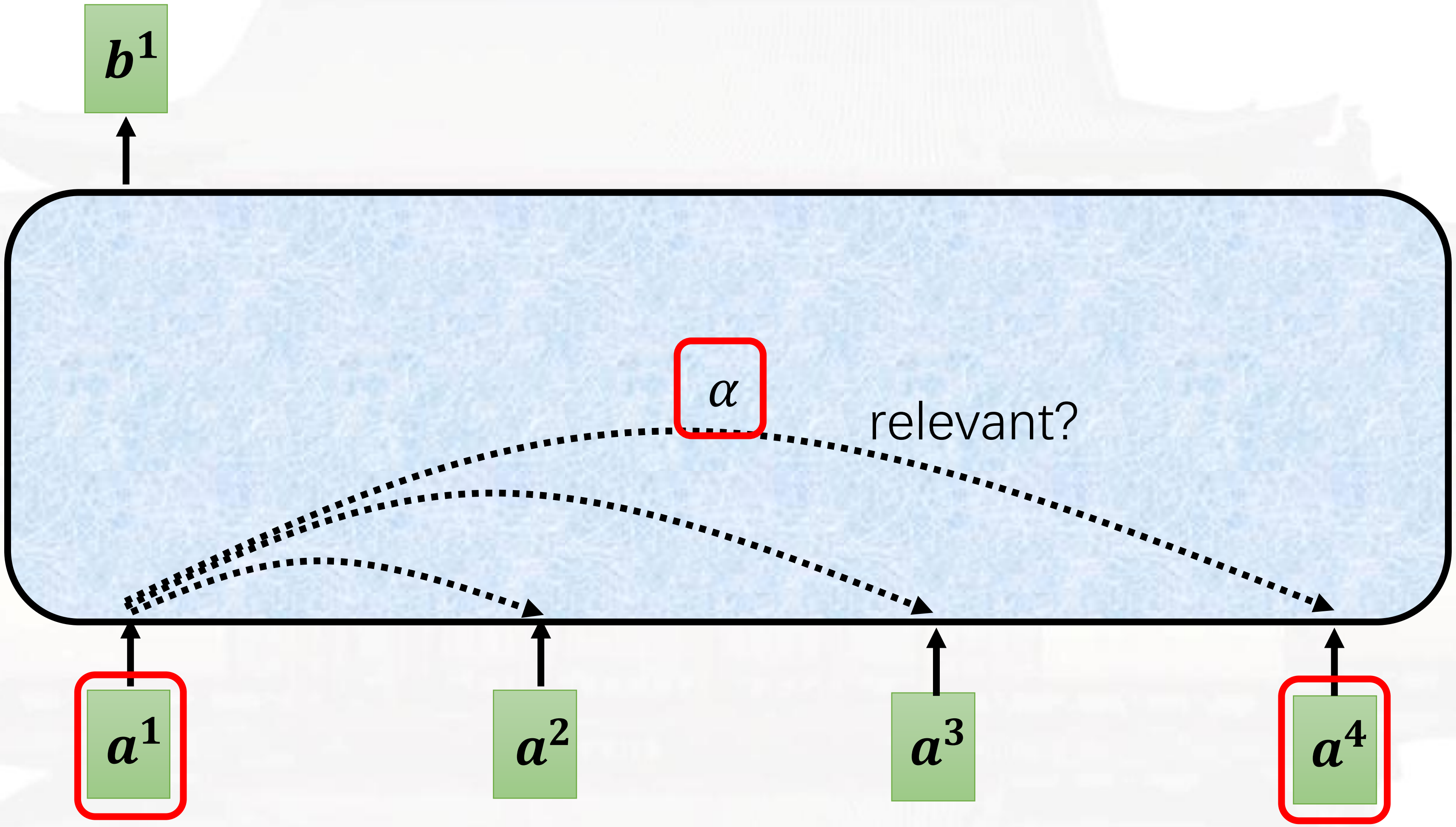


Self-Attention



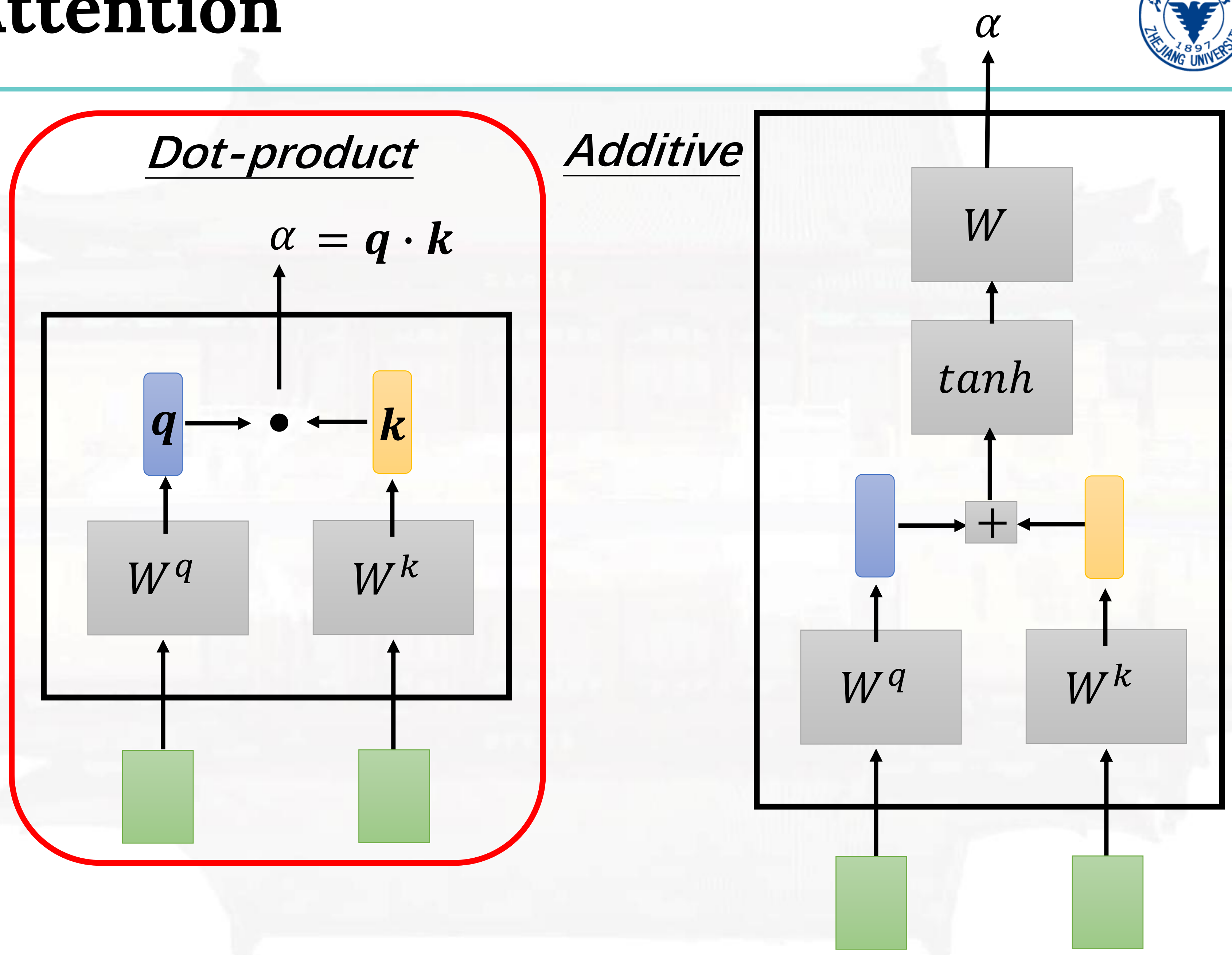
Can be either **input** or **a hidden layer**

Self-Attention



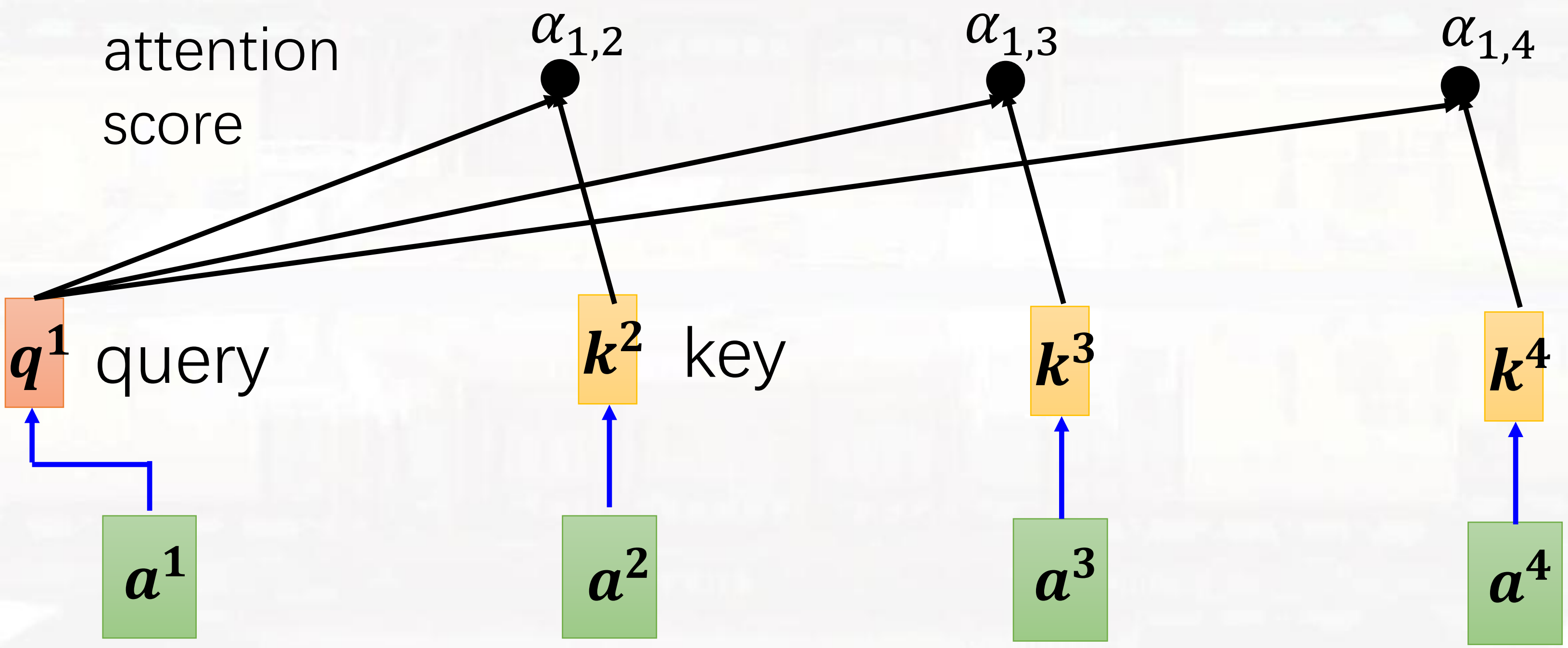
Find the relevant vectors in a sequence

Self-Attention



Self-Attention

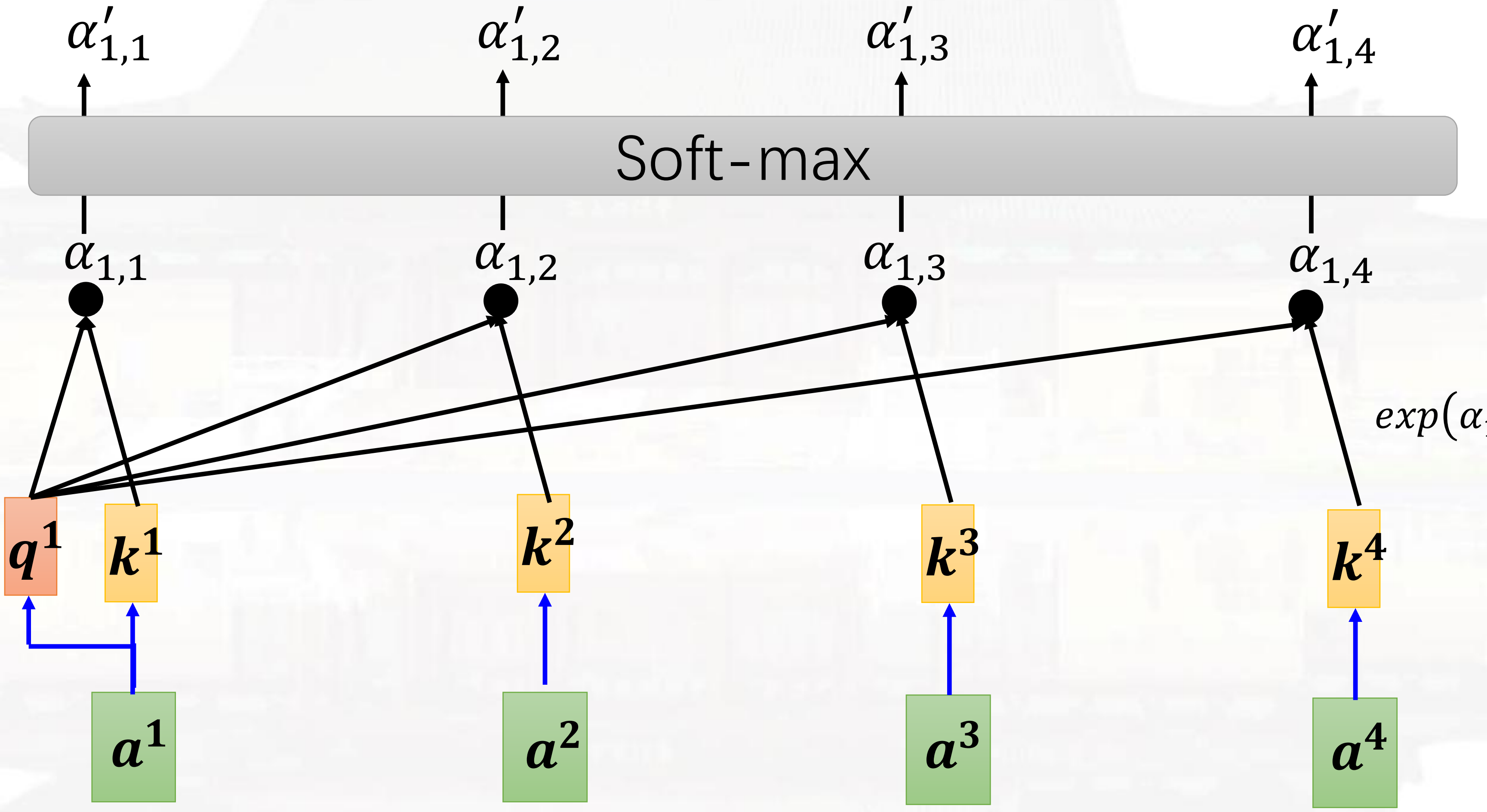
$\alpha_{1,2} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^2$ $\alpha_{1,3} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^3$ $\alpha_{1,4} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^4$



$\boldsymbol{q}^1 = W^q \boldsymbol{a}^1$ $\boldsymbol{k}^2 = W^k \boldsymbol{a}^2$ $\boldsymbol{k}^3 = W^k \boldsymbol{a}^3$ $\boldsymbol{k}^4 = W^k \boldsymbol{a}^4$

Self-Attention

浙江大学 ZHEJIANG UNIVERSITY



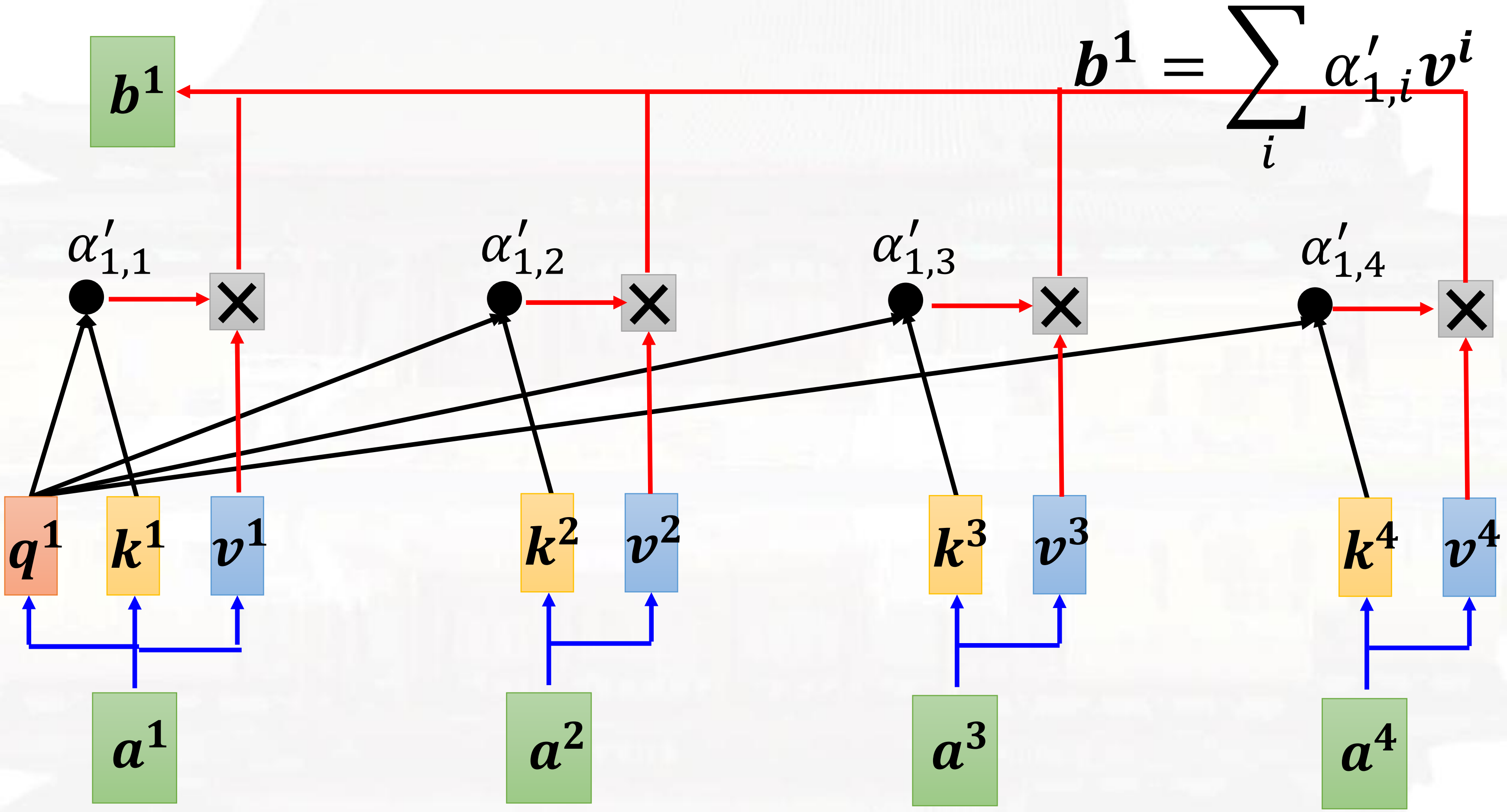
$\alpha'_{1,i} = exp(\alpha_{1,i})/\sum_j exp(\alpha_{1,j})$

$q^1 = W^q a^1$ $k^2 = W^k a^2$ $k^3 = W^k a^3$ $k^4 = W^k a^4$

$k^1 = W^k a^1$

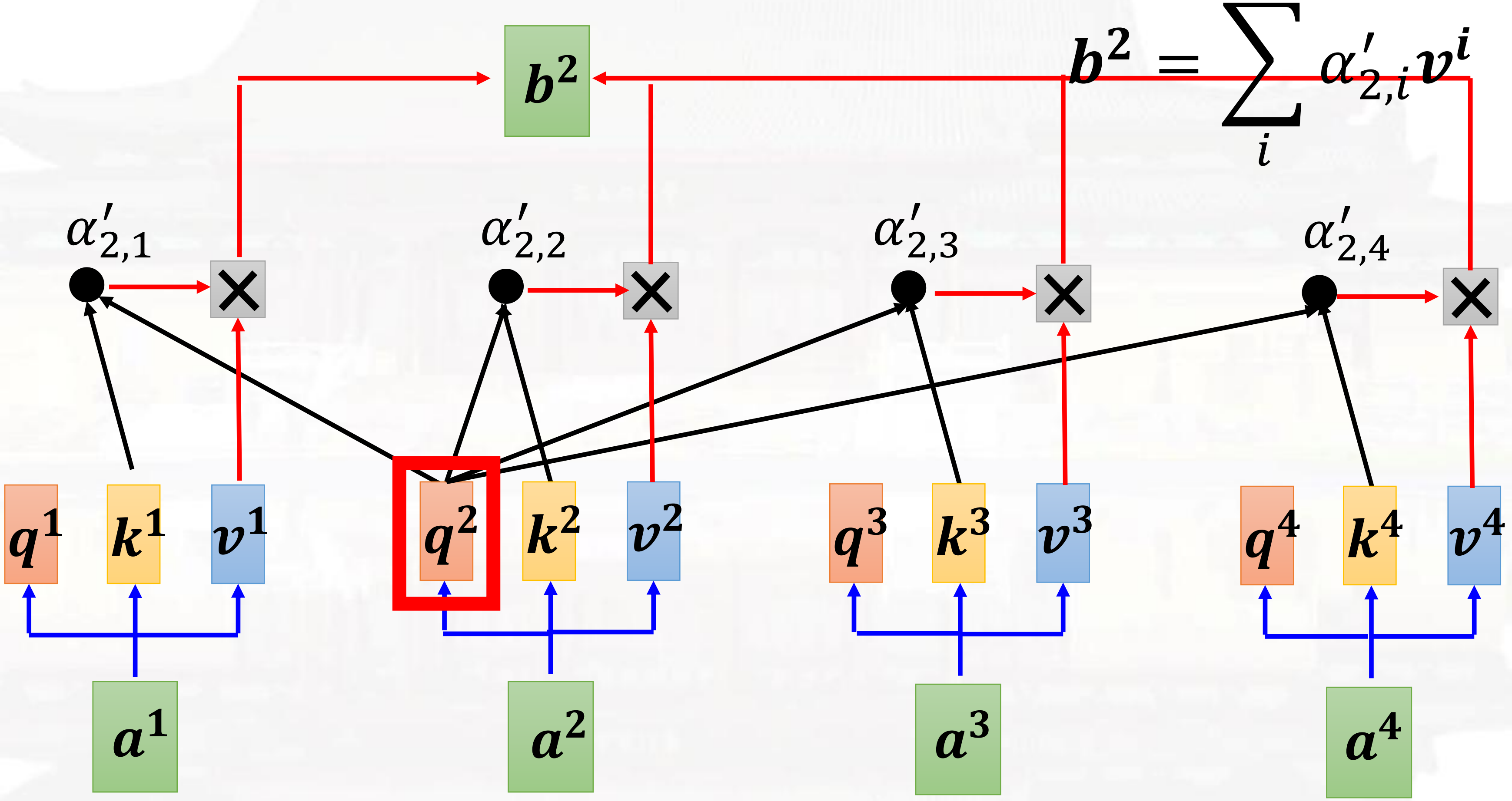
Self-Attention

浙江大学 ZHEJIANG UNIVERSITY

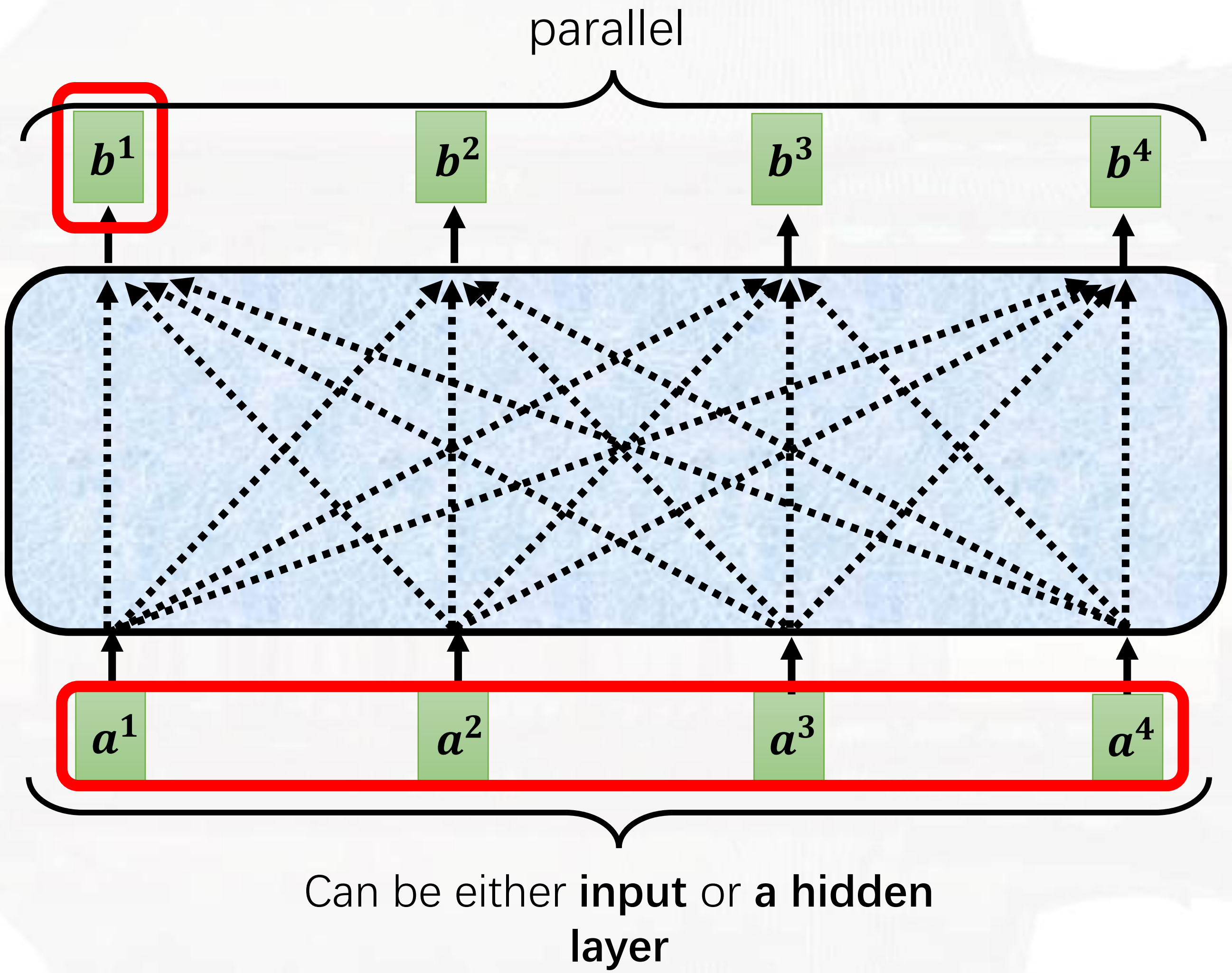


$v^1 = W^v a^1$ $v^2 = W^v a^2$ $v^3 = W^v a^3$ $v^4 = W^v a^4$

Self-Attention



Self-Attention

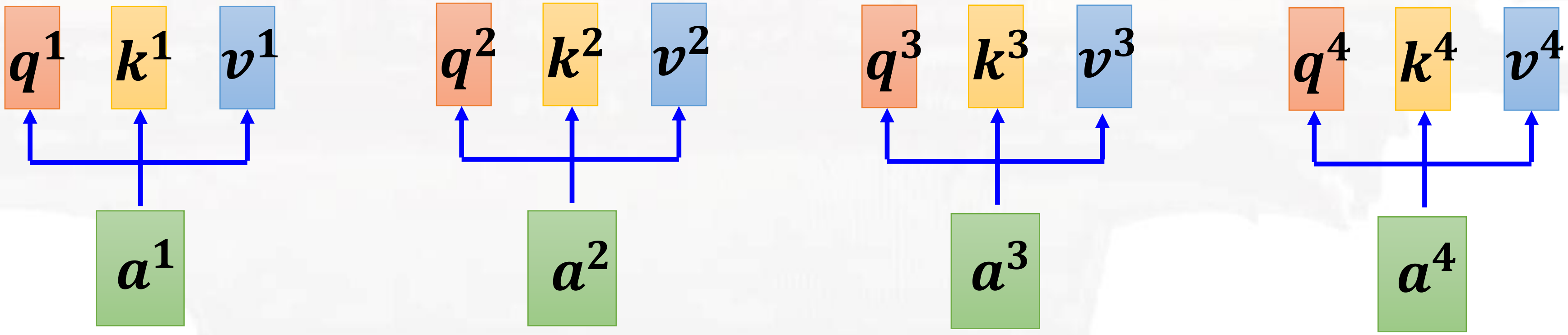


Self-Attention

$$q^i = W^q a^i \qquad \underset{Q}{\begin{bmatrix} q^1 & q^2 & q^3 & q^4 \end{bmatrix}} = W^q \underset{I}{\begin{bmatrix} a^1 & a^2 & a^3 & a^4 \end{bmatrix}}$$

$$k^i = W^k a^i \qquad \underset{K}{\begin{bmatrix} k^1 & k^2 & k^3 & k^4 \end{bmatrix}} = W^k \underset{I}{\begin{bmatrix} a^1 & a^2 & a^3 & a^4 \end{bmatrix}}$$

$$v^i = W^v a^i \qquad \underset{V}{\begin{bmatrix} v^1 & v^2 & v^3 & v^4 \end{bmatrix}} = W^v \underset{I}{\begin{bmatrix} a^1 & a^2 & a^3 & a^4 \end{bmatrix}}$$

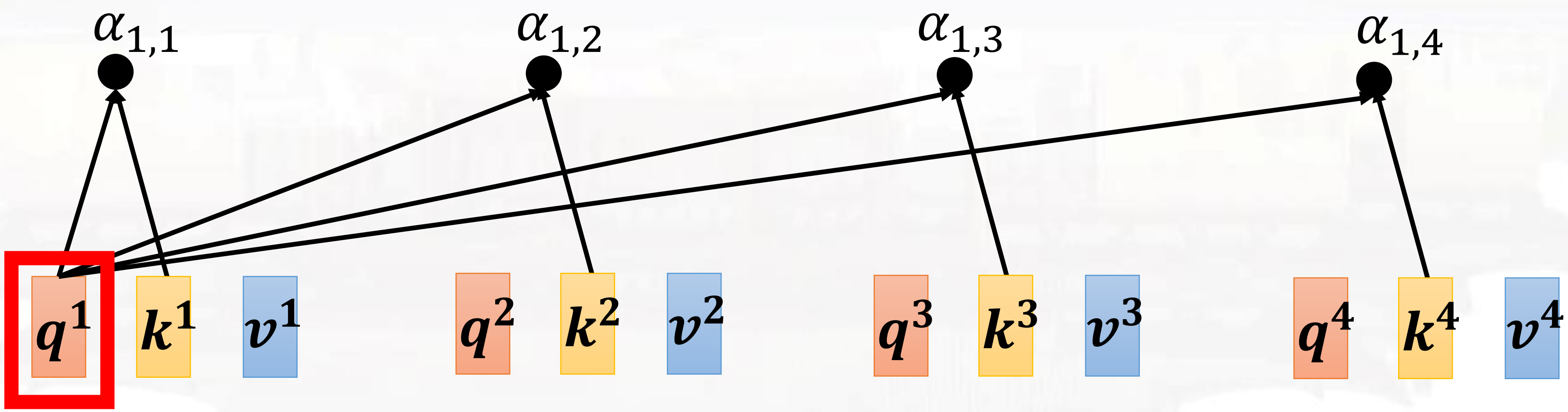


Self-Attention

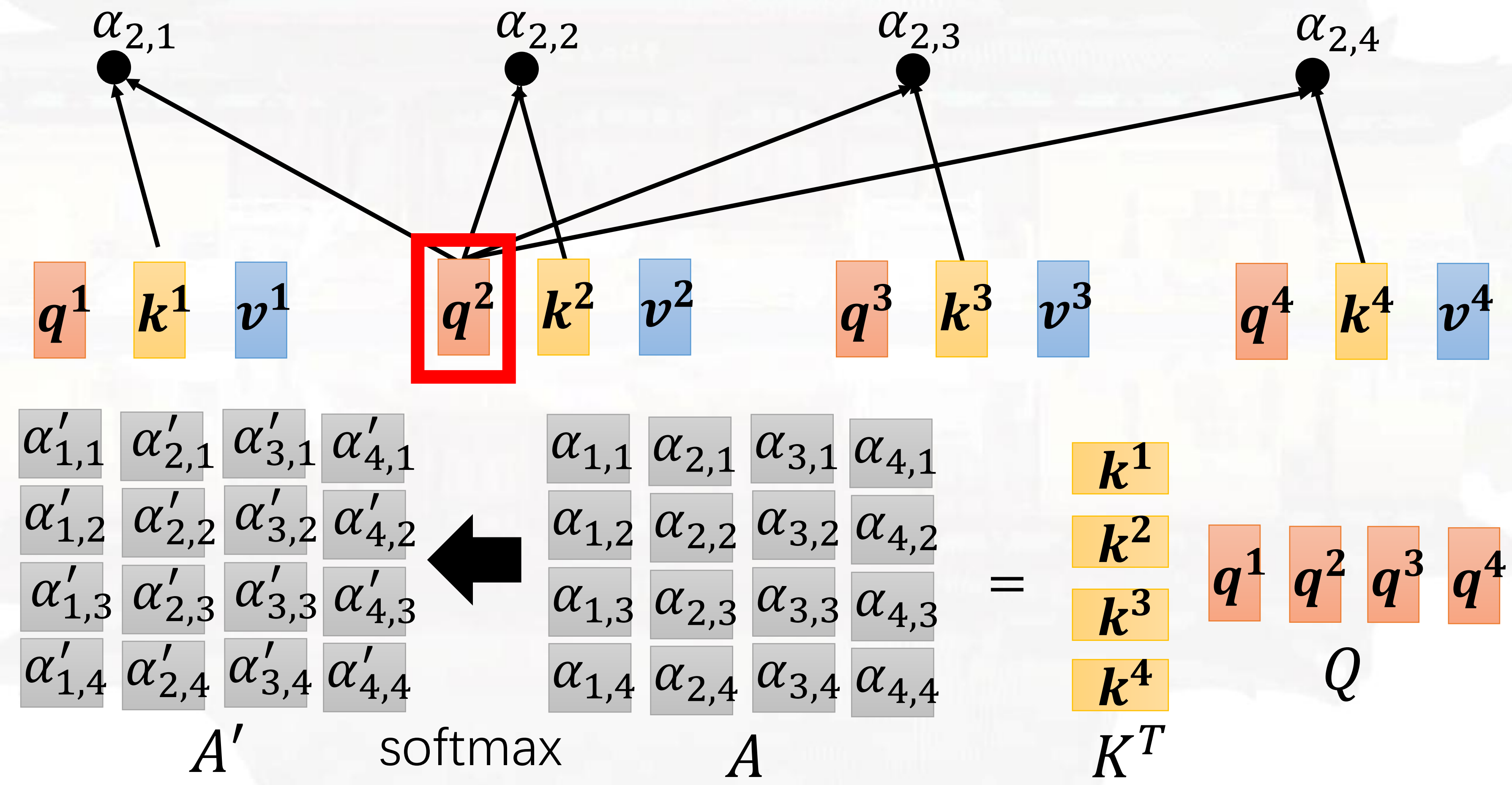
$$\alpha_{1,1} = \boldsymbol{k}^1 \boldsymbol{q}^1 \qquad \alpha_{1,2} = \boldsymbol{k}^2 \boldsymbol{q}^1$$

$$\alpha_{1,3} = \boldsymbol{k}^3 \boldsymbol{q}^1 \qquad \alpha_{1,4} = \boldsymbol{k}^4 \boldsymbol{q}^1$$

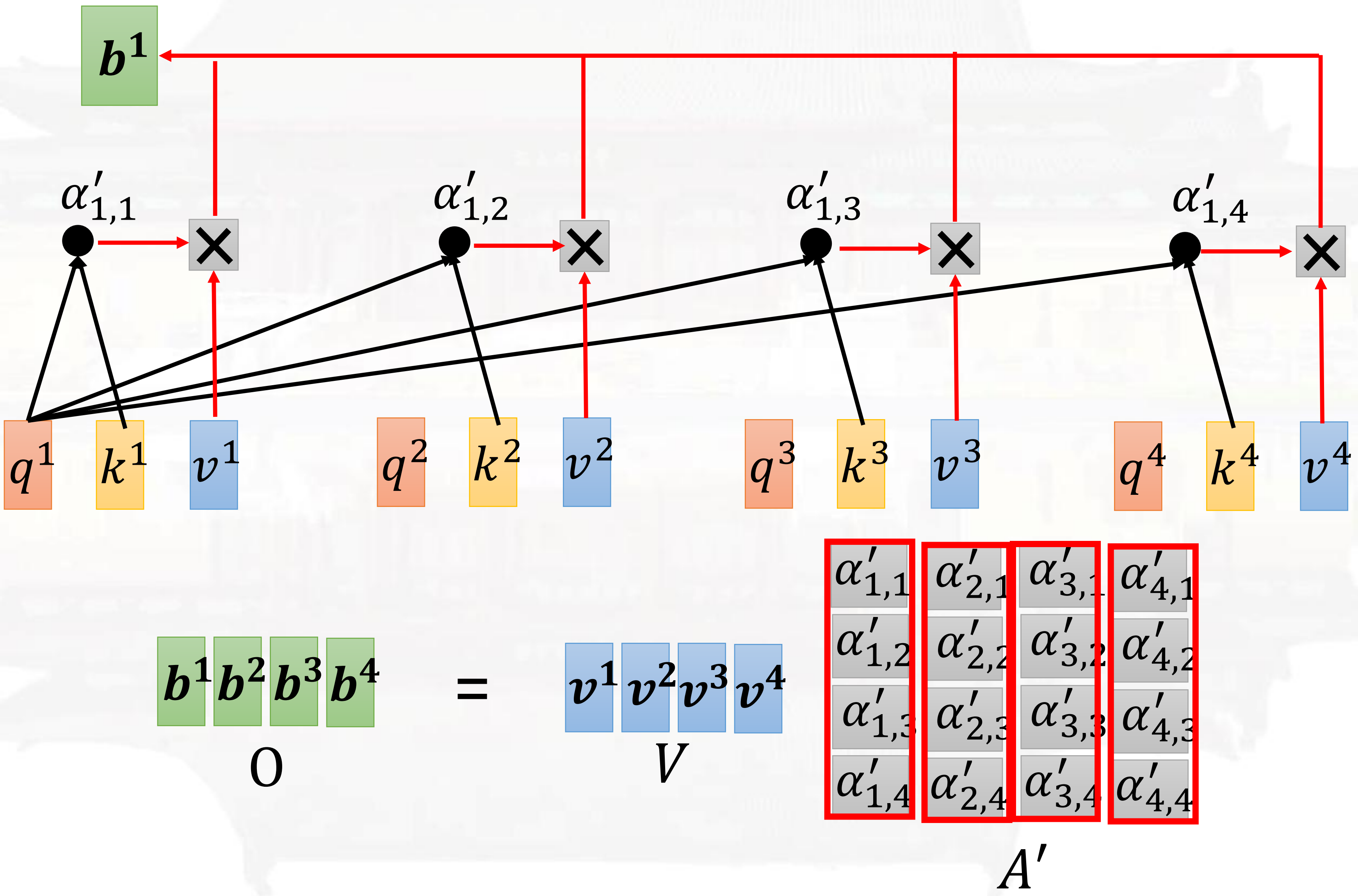
$$\begin{bmatrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{bmatrix} = \begin{bmatrix} \boldsymbol{k}^1 \\ \boldsymbol{k}^2 \\ \boldsymbol{k}^3 \\ \boldsymbol{k}^4 \end{bmatrix} \boldsymbol{q}^1$$



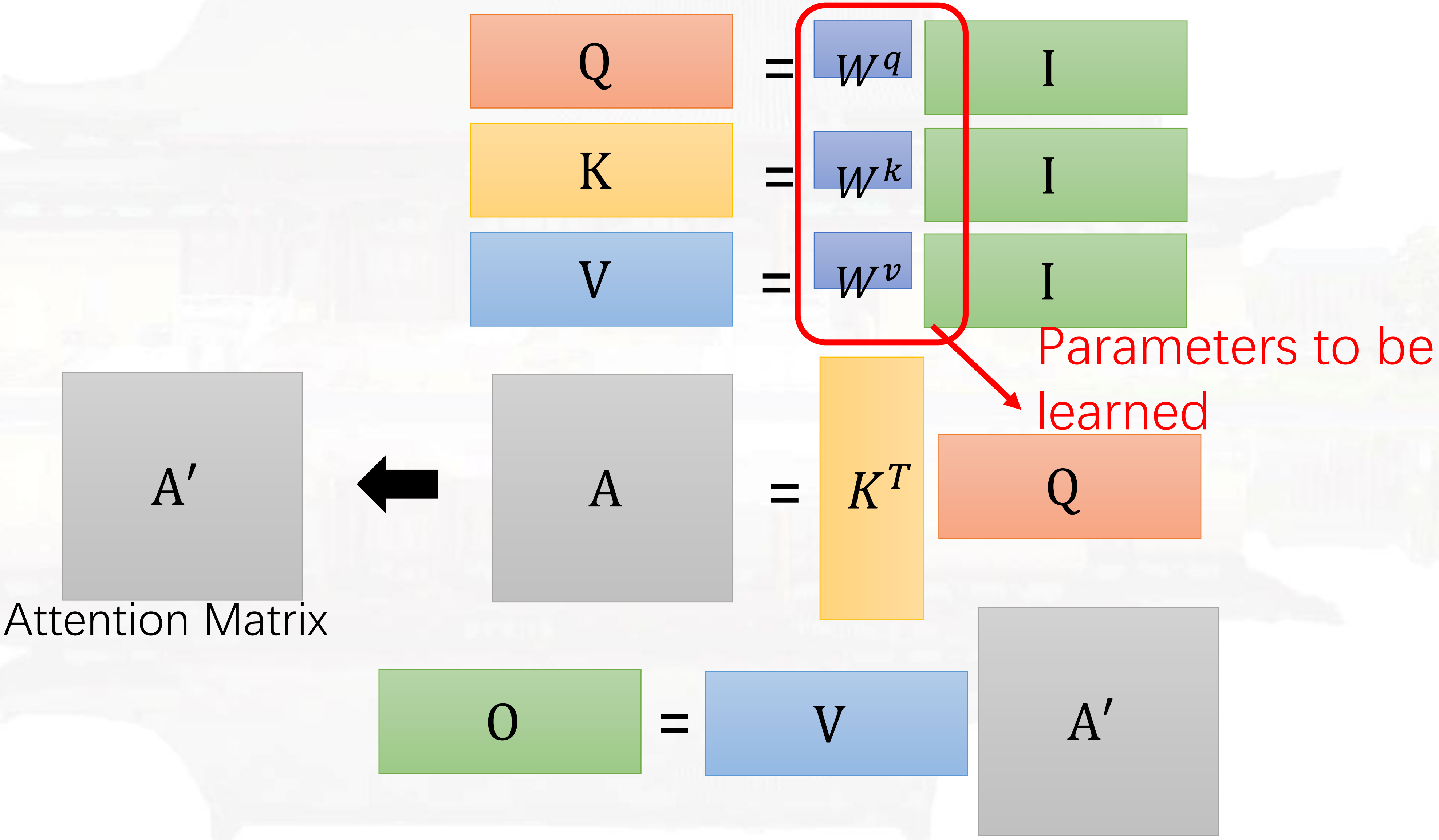
Self-Attention



Self-Attention



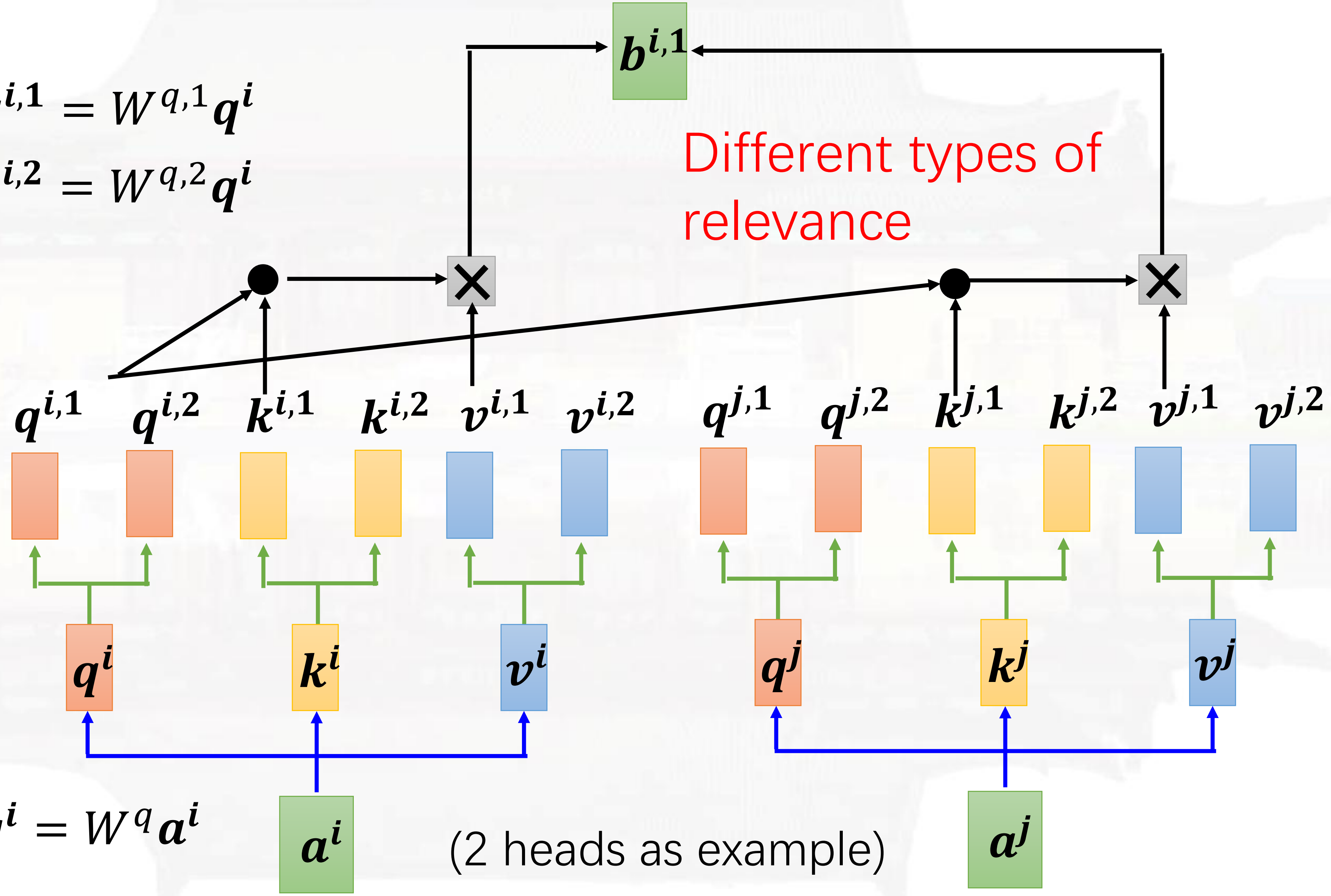
Self-Attention



Multi-head Self-attention

$$q^{i,1} = W^{q,1} q^i$$

$$q^{i,2} = W^{q,2} q^i$$

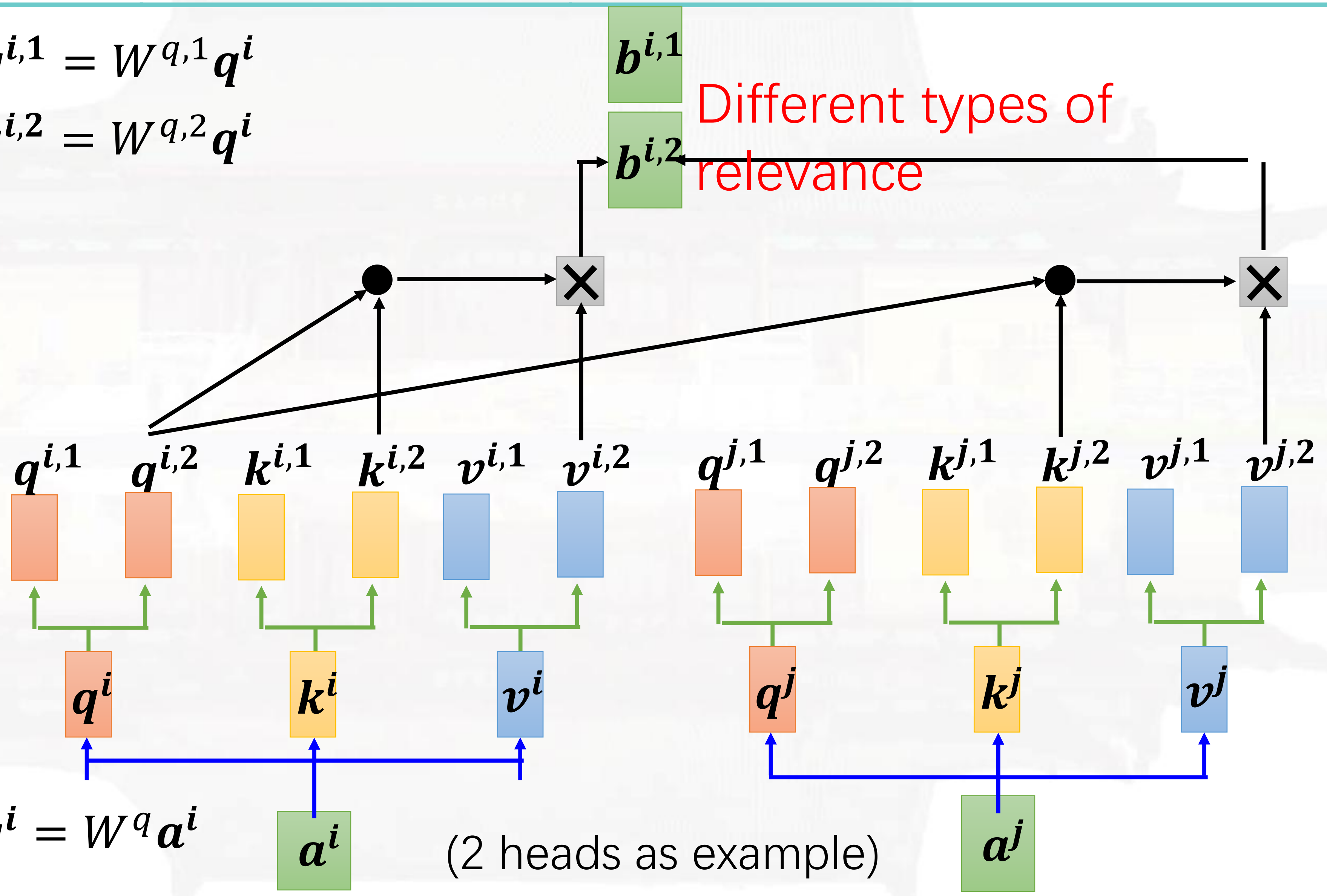


$$q^i = W^q a^i$$

Multi-head Self-attention

$q^{i,1} = W^{q,1} q^i$

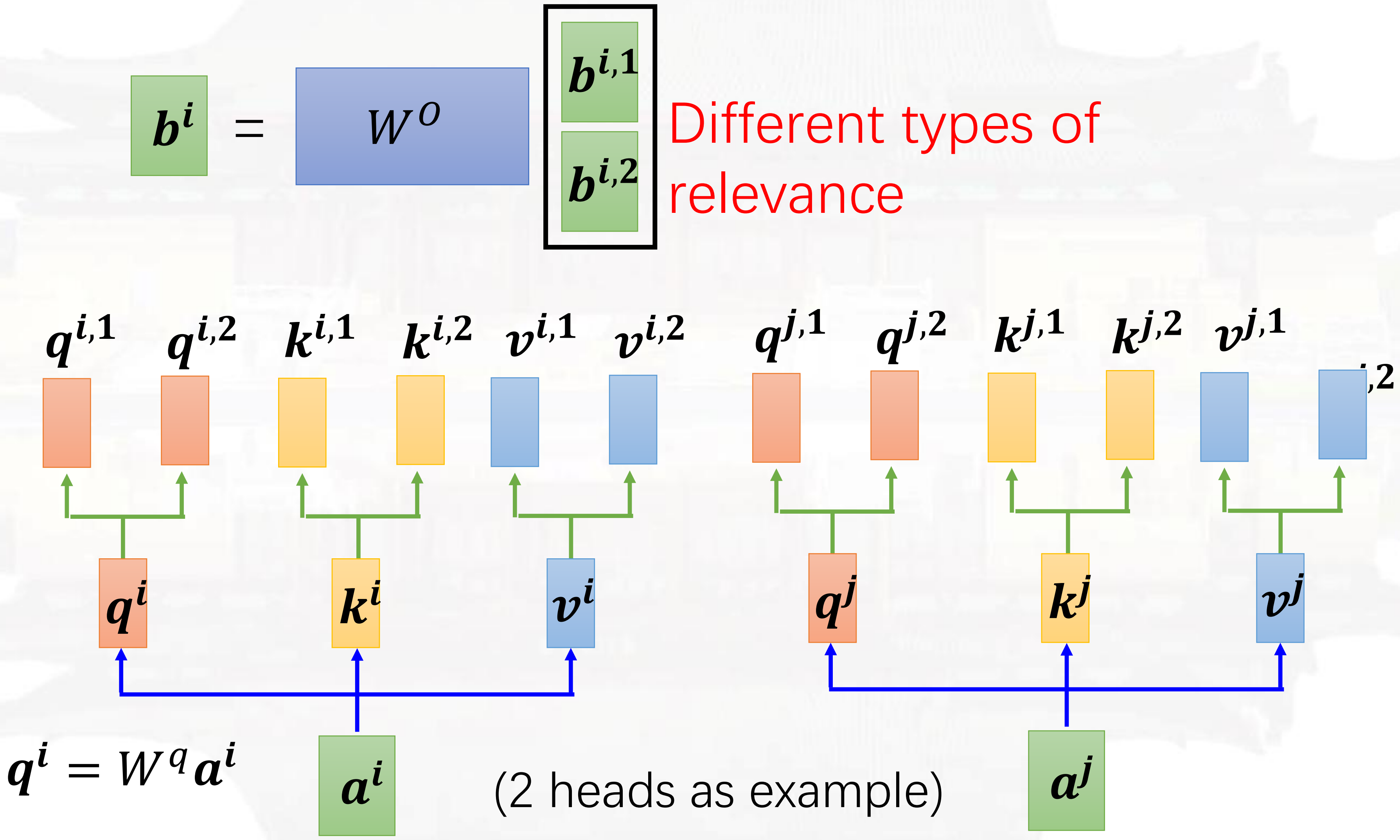
$q^{i,2} = W^{q,2} q^i$



$q^i = W^q a^i$

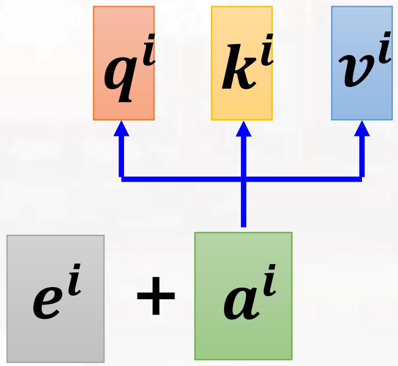
(2 heads as example)

Multi-head Self-attention

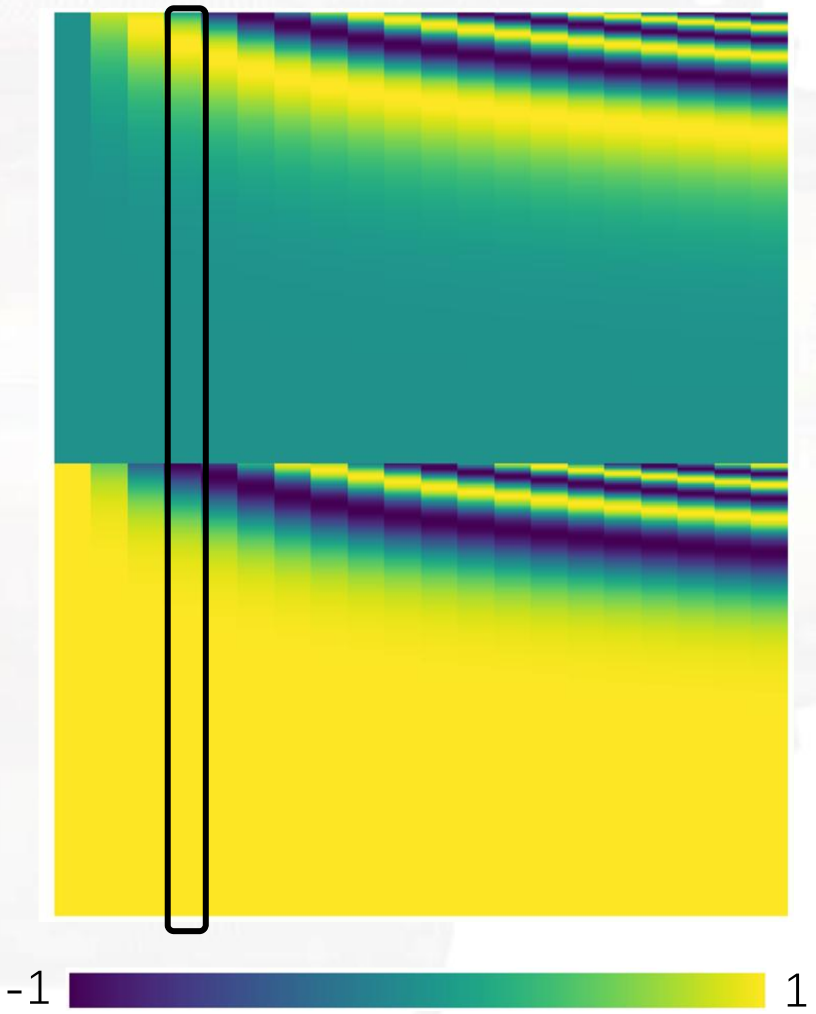


Positional Encoding

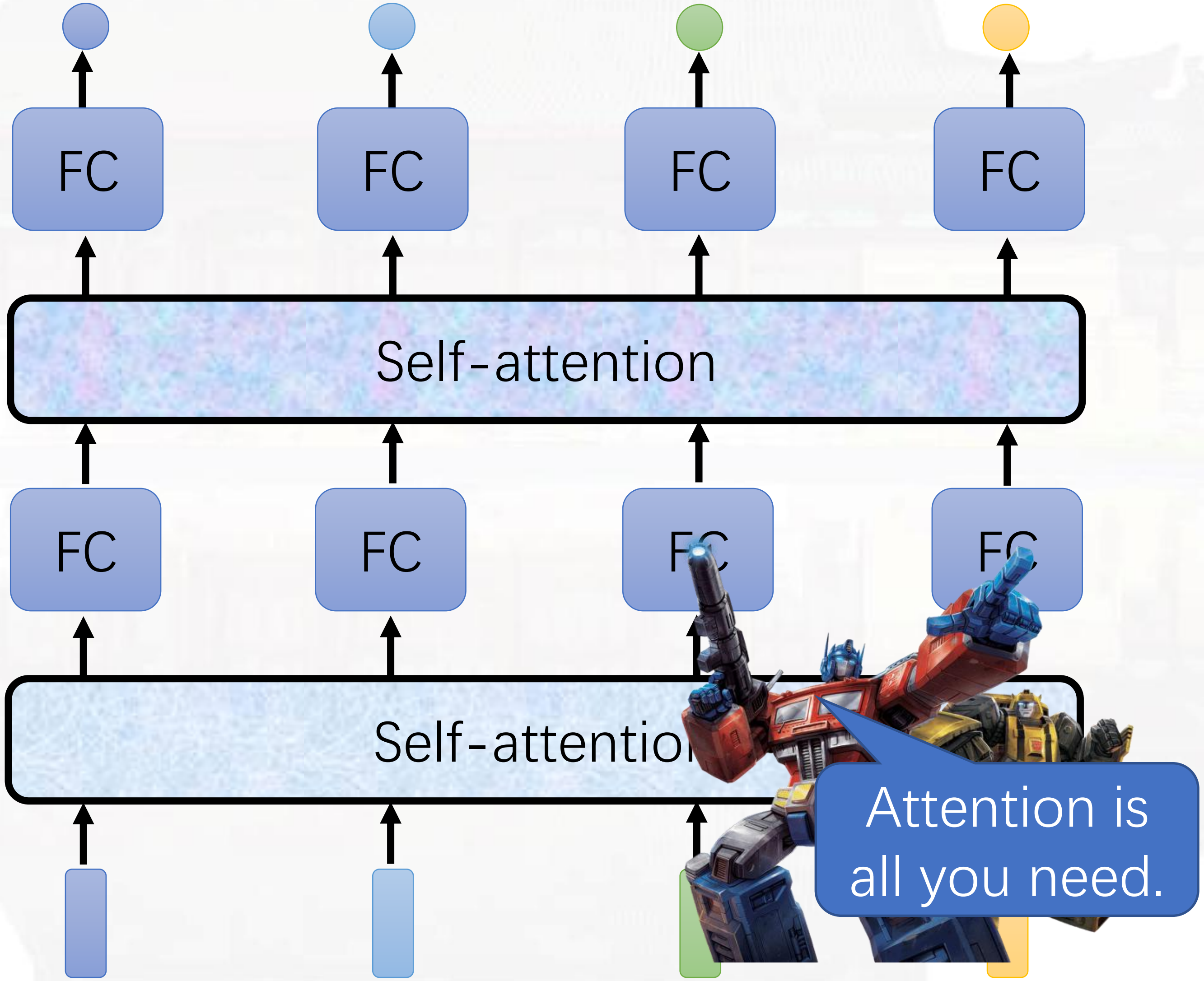
- No position information in self-attention.
- Each position has a unique positional vector e^i
- **hand-crafted**
- **learned from data**



Each column represents a positional vector e^i

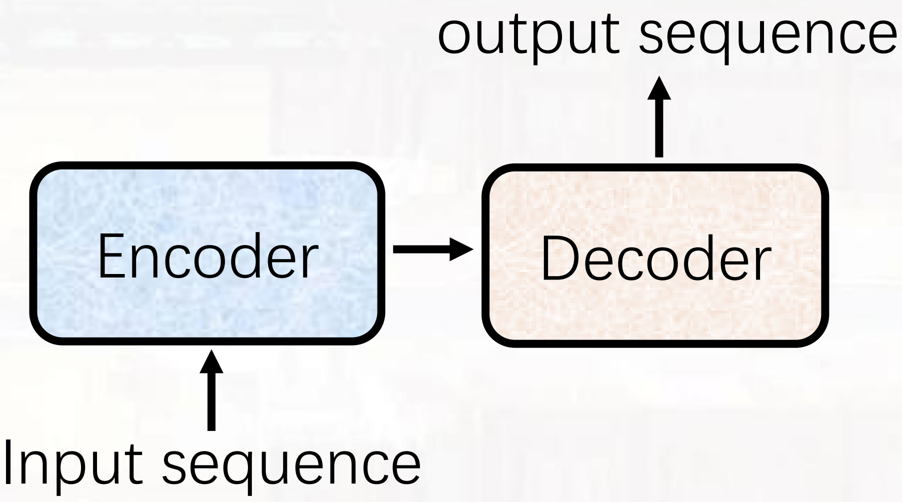


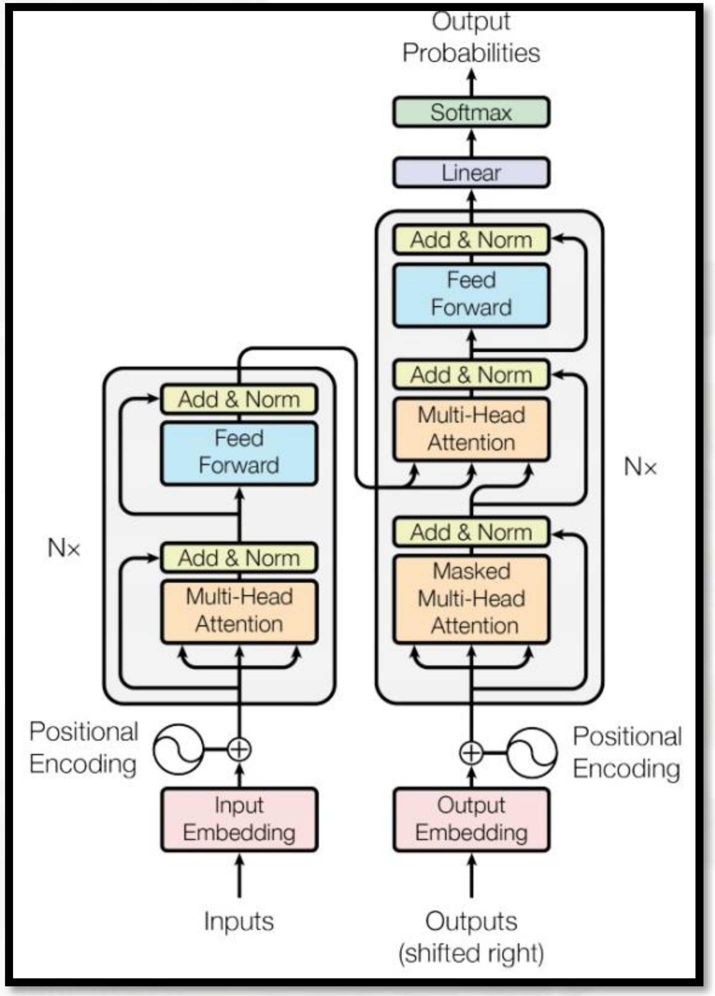
Transformer



https://arxiv.org/abs/1706.03762

Transformer



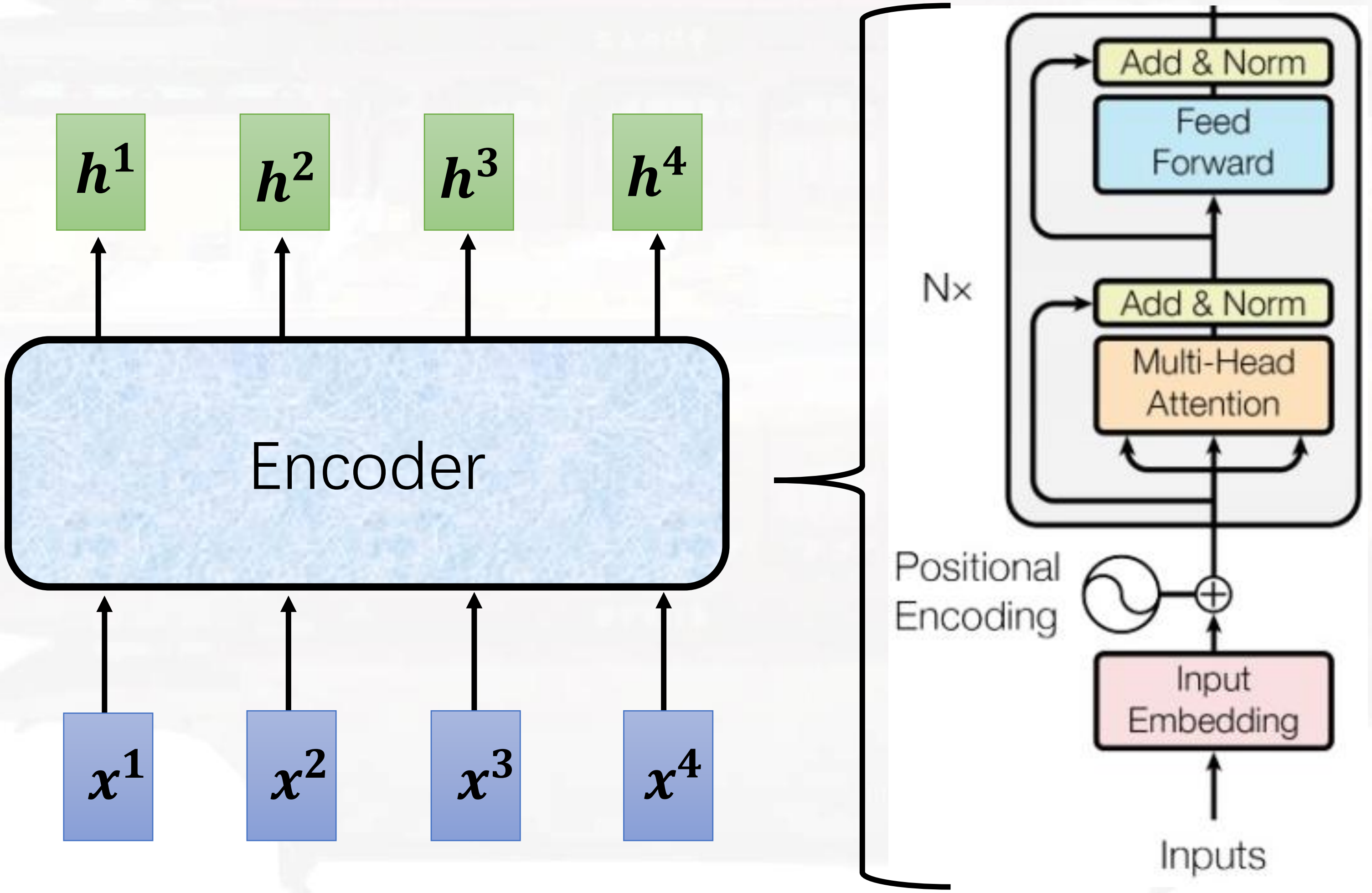


Transformer

https://arxiv.org/abs/1706.03762

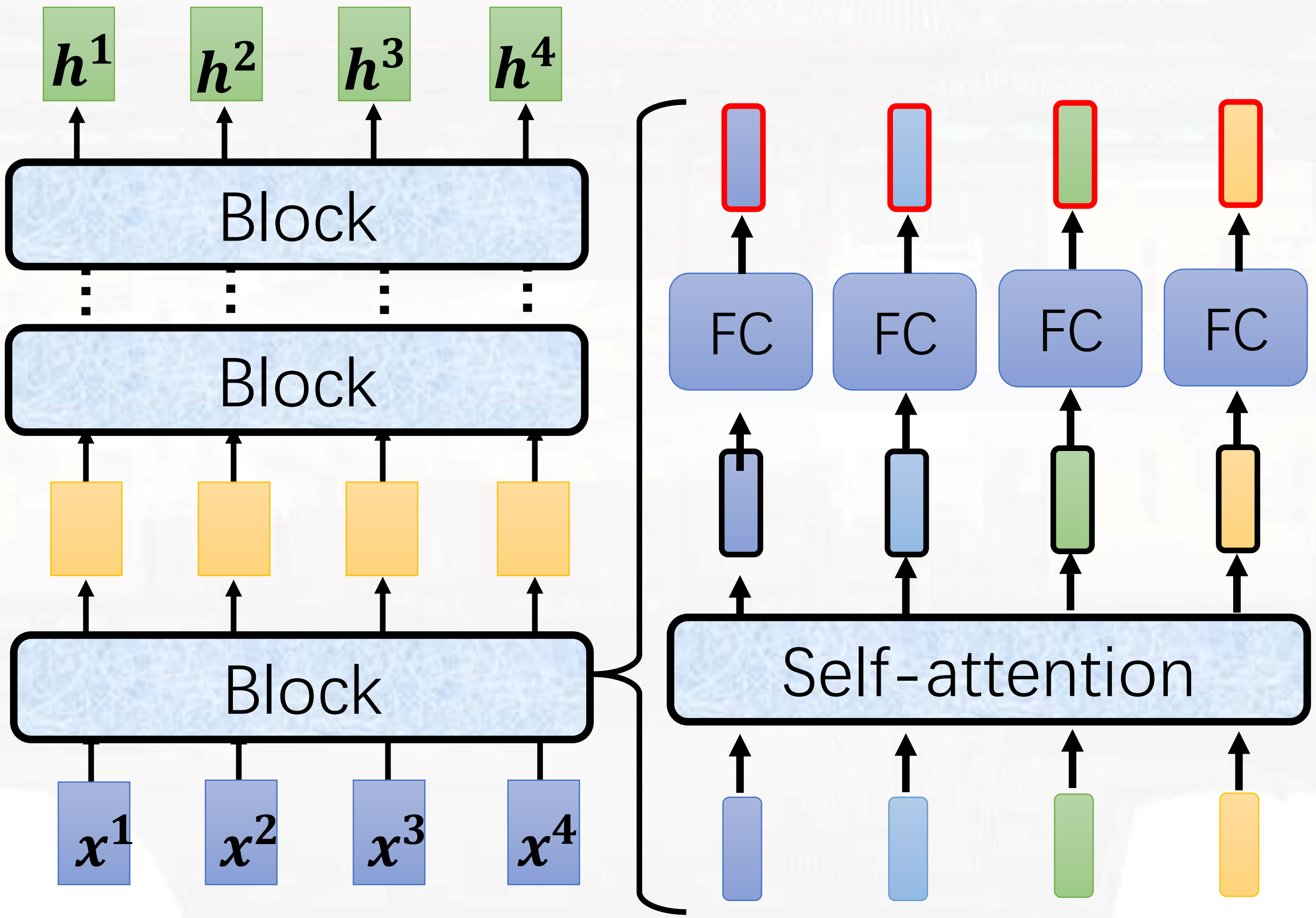
Transformer

- Transformer's Encoder



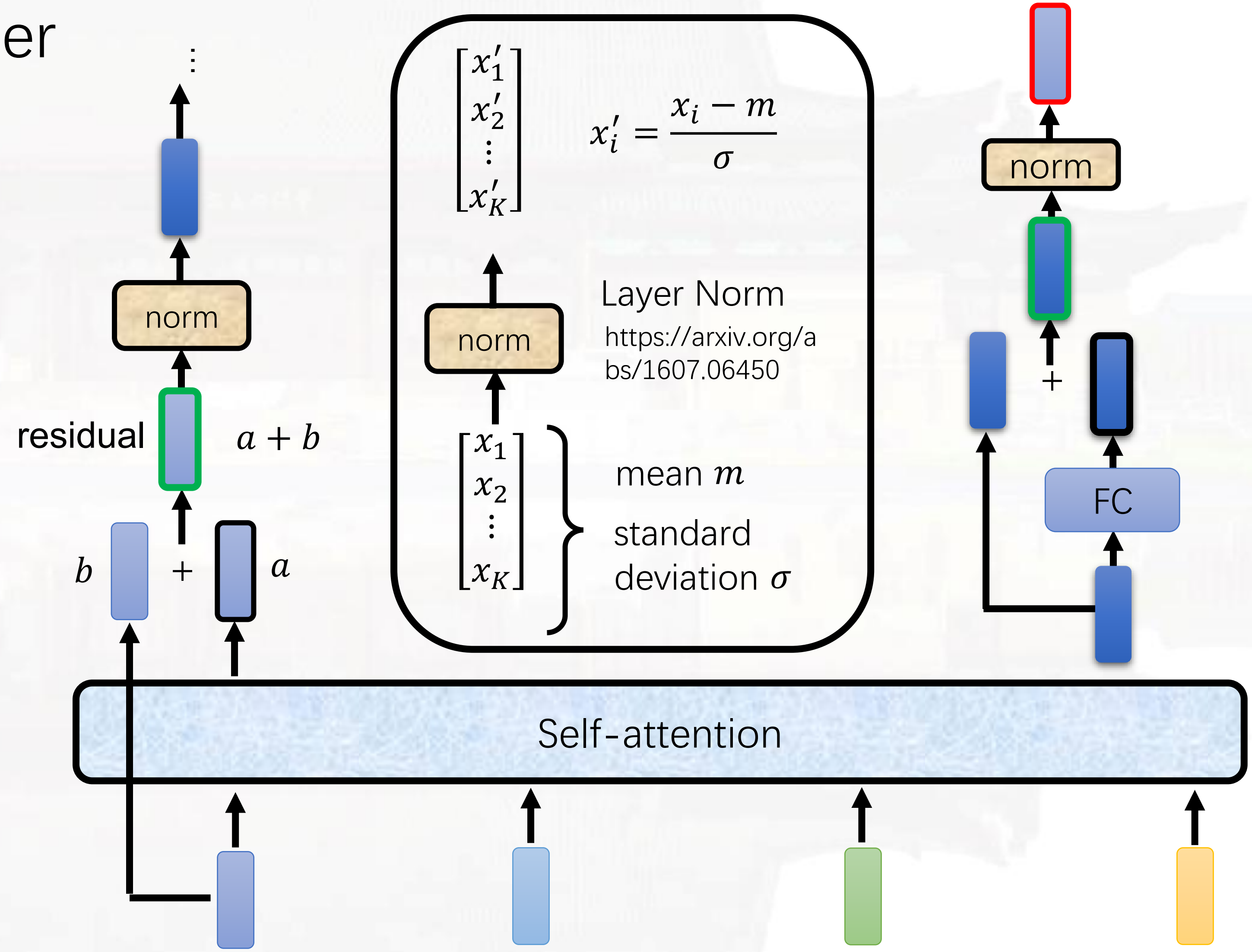
Transformer

- Transformer's Encoder



Transformer

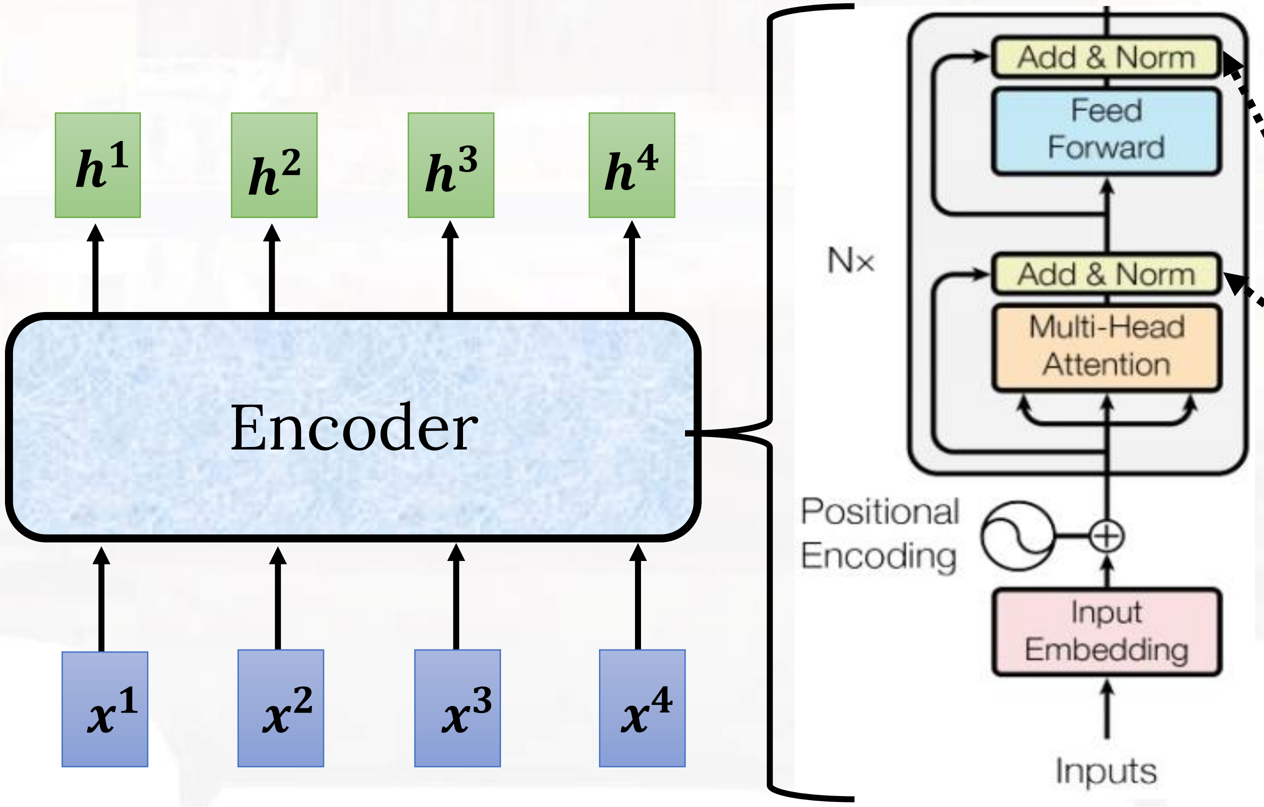
- Transformer's Encoder



Transformer

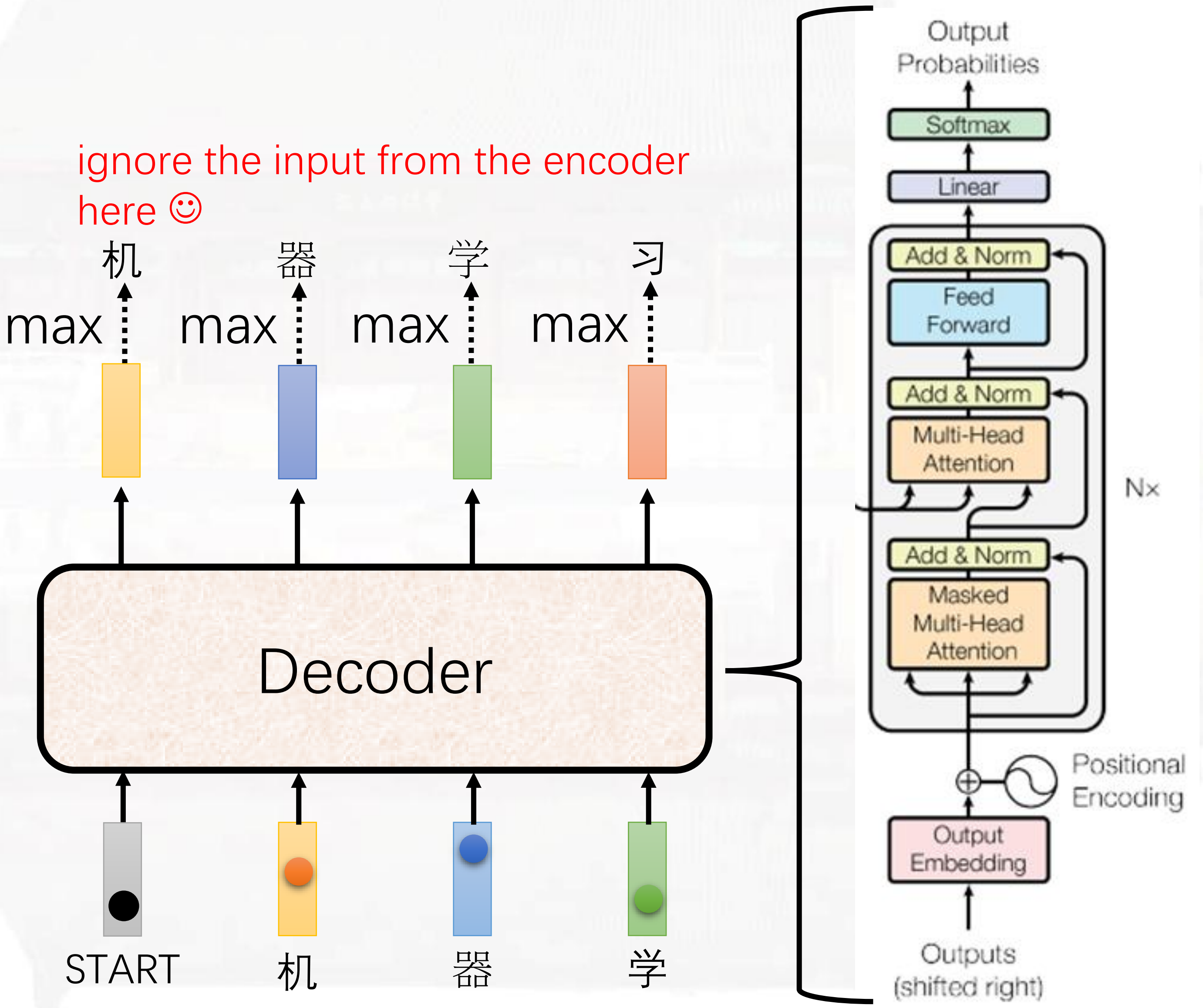
- Transformer's Encoder

I use the same network architecture as transformer encoder.

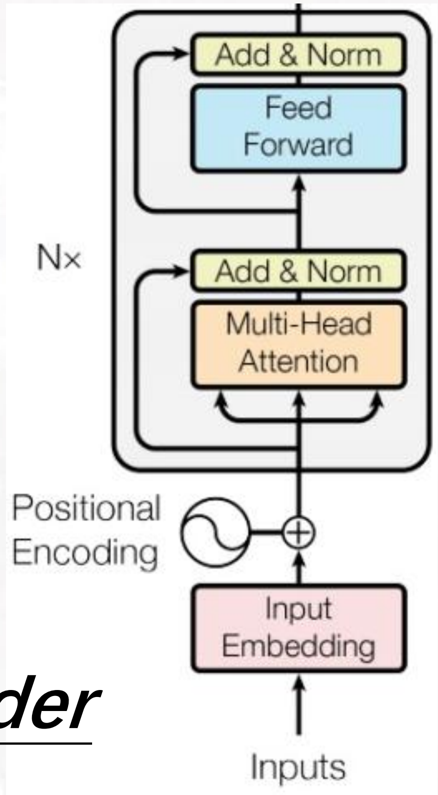


Residual + Layer norm

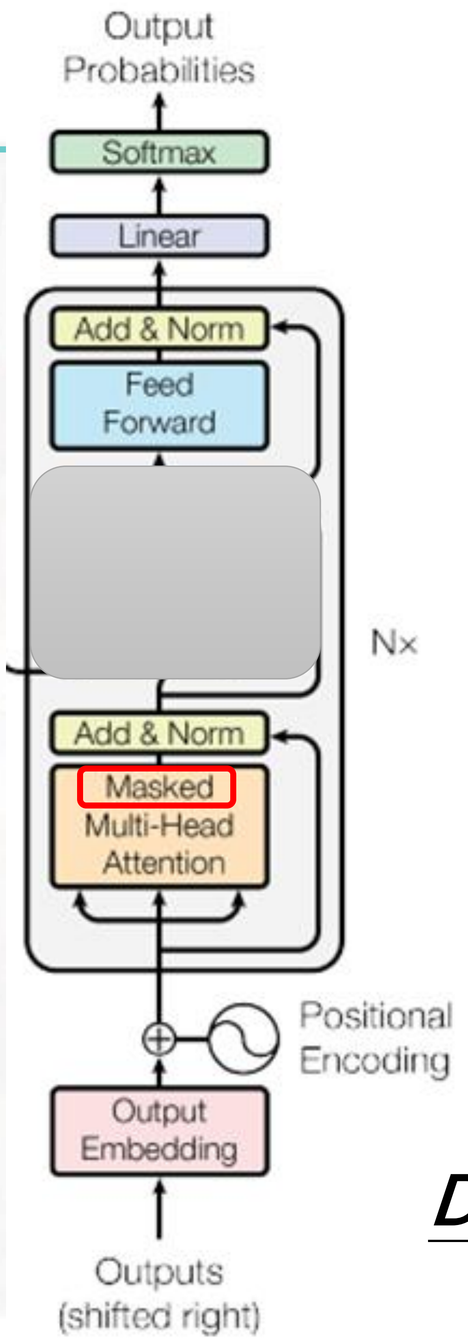
Transformer



Transformer



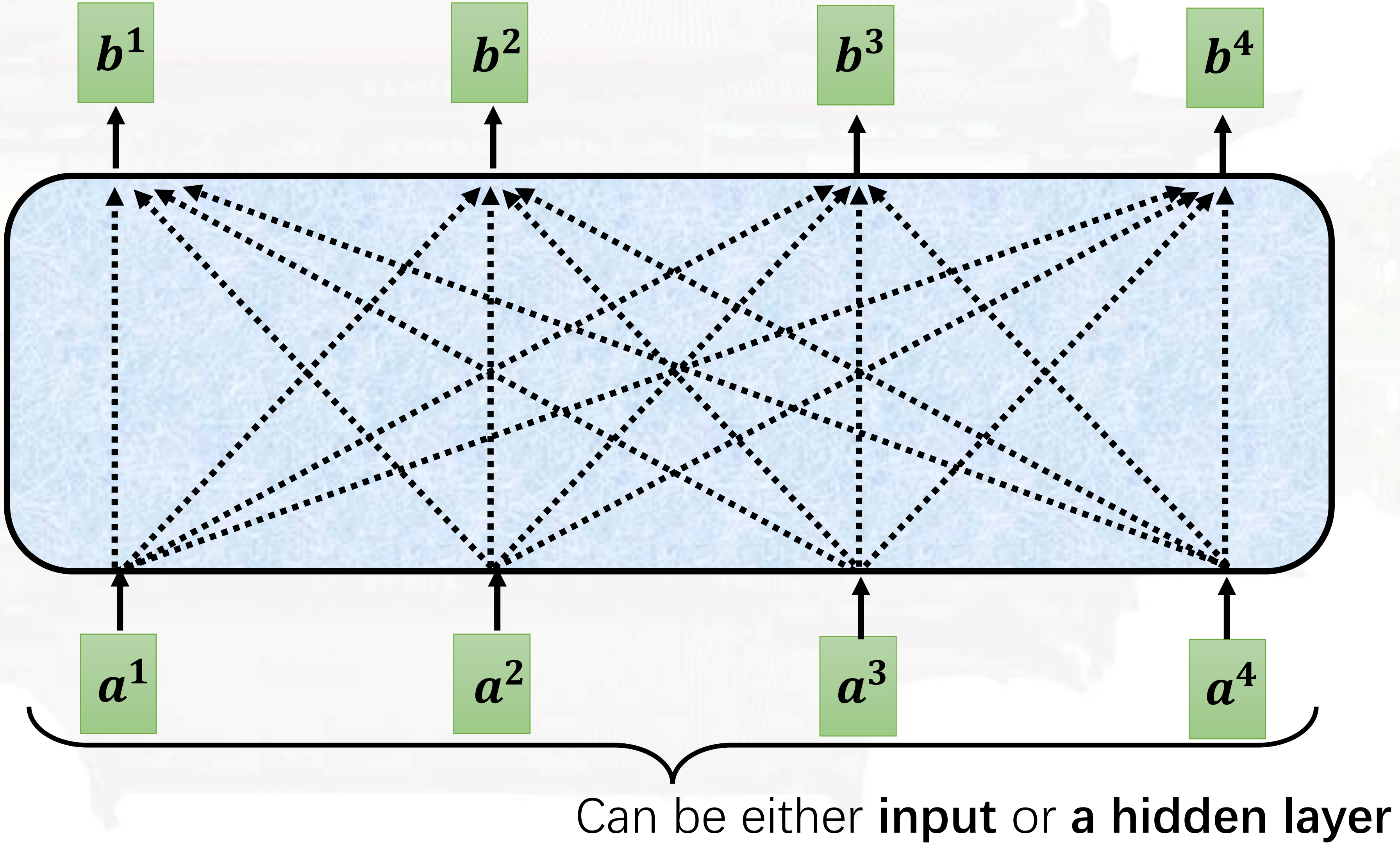
Encoder



Decoder

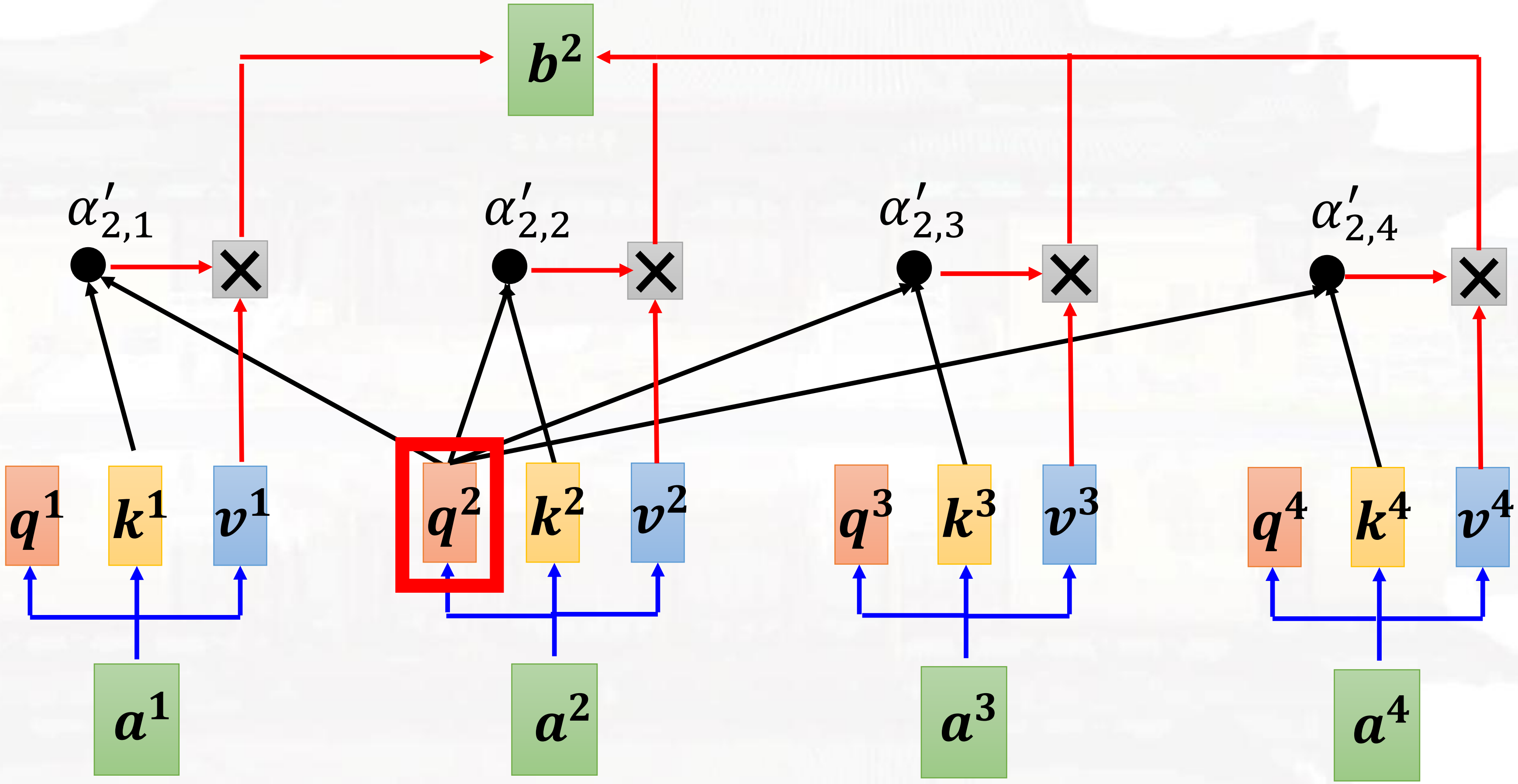
Transformer

- Masked Self-attention



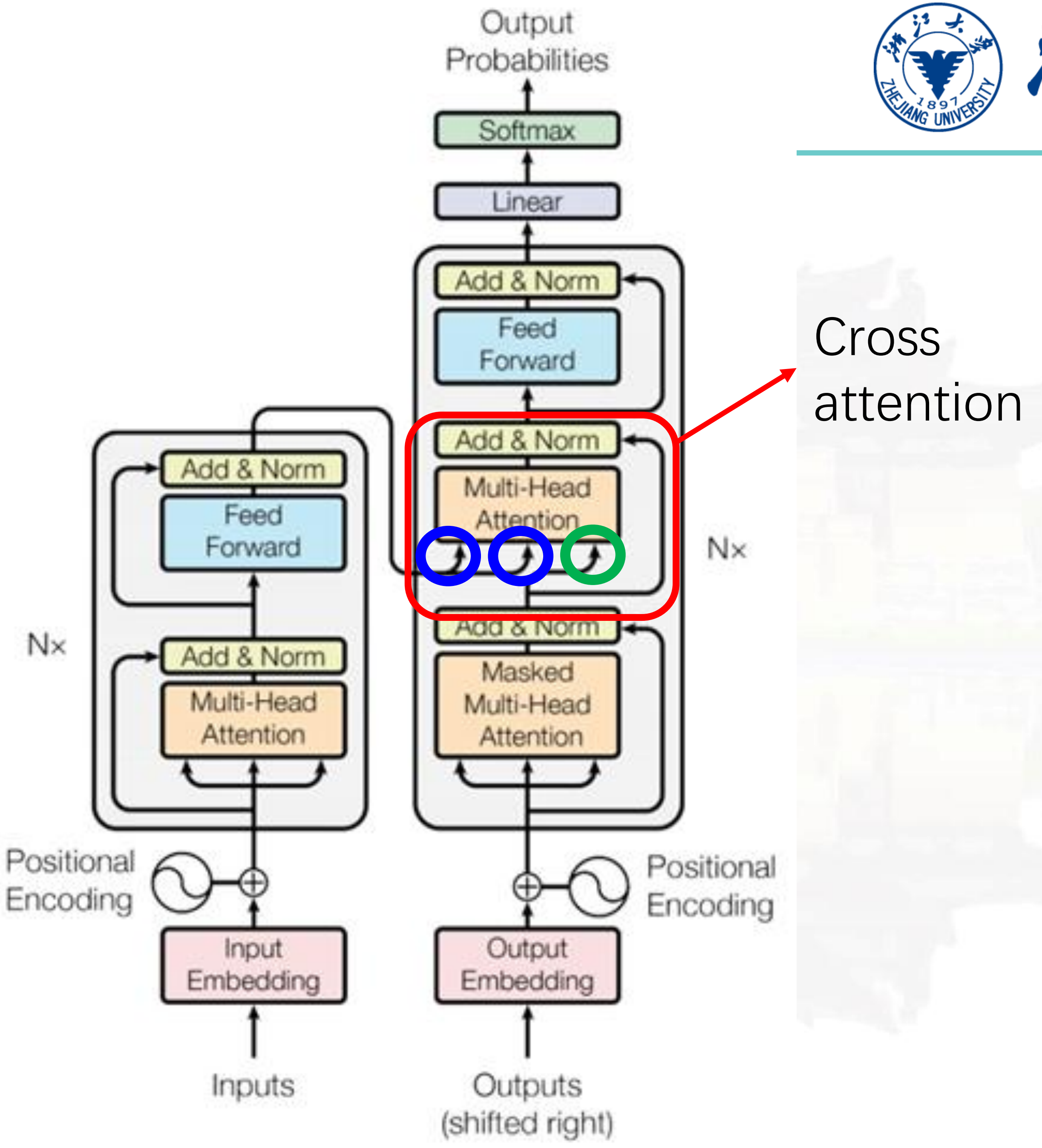
Transformer

- Masked Self-attention

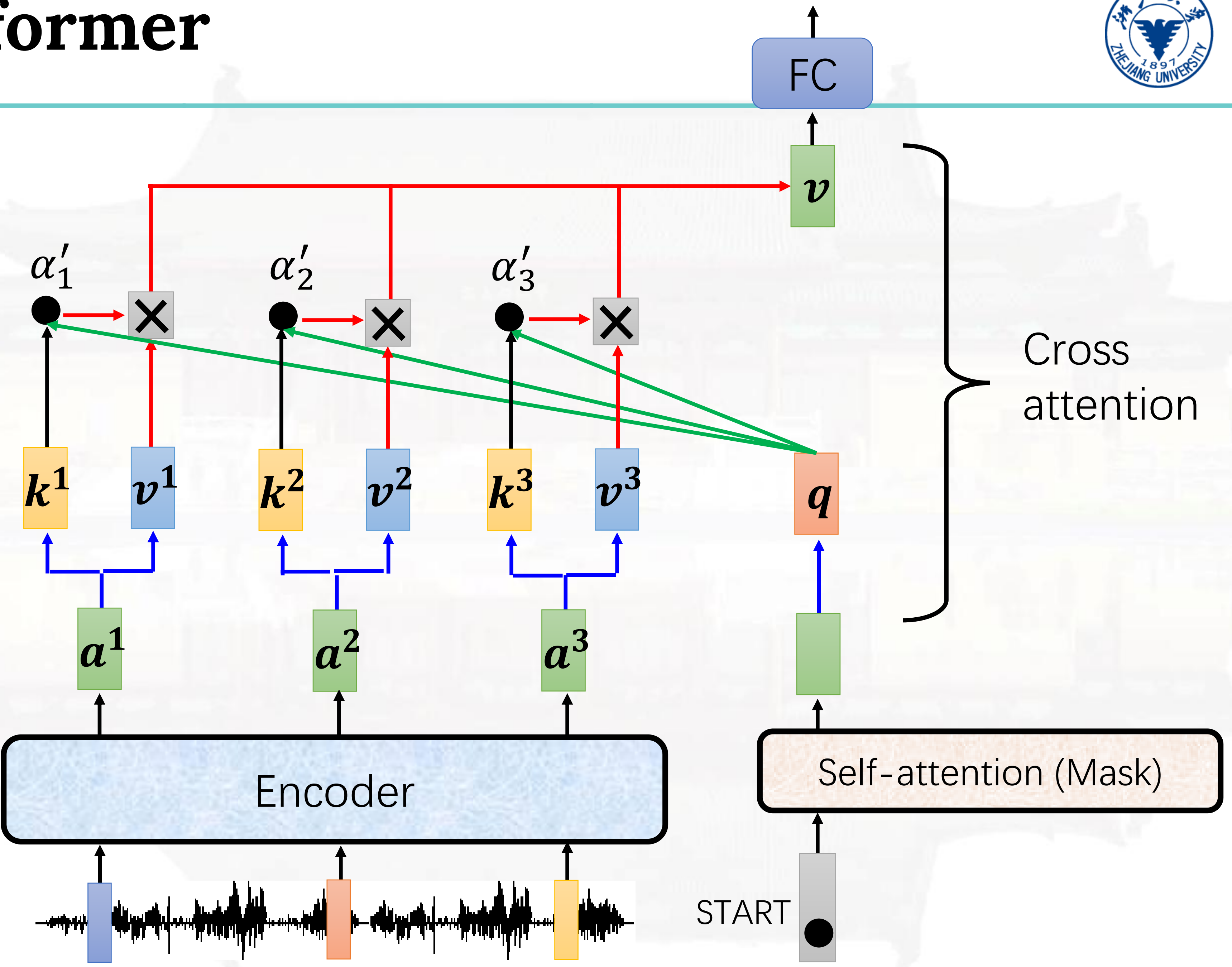


Why masked? Consider how does decoder work

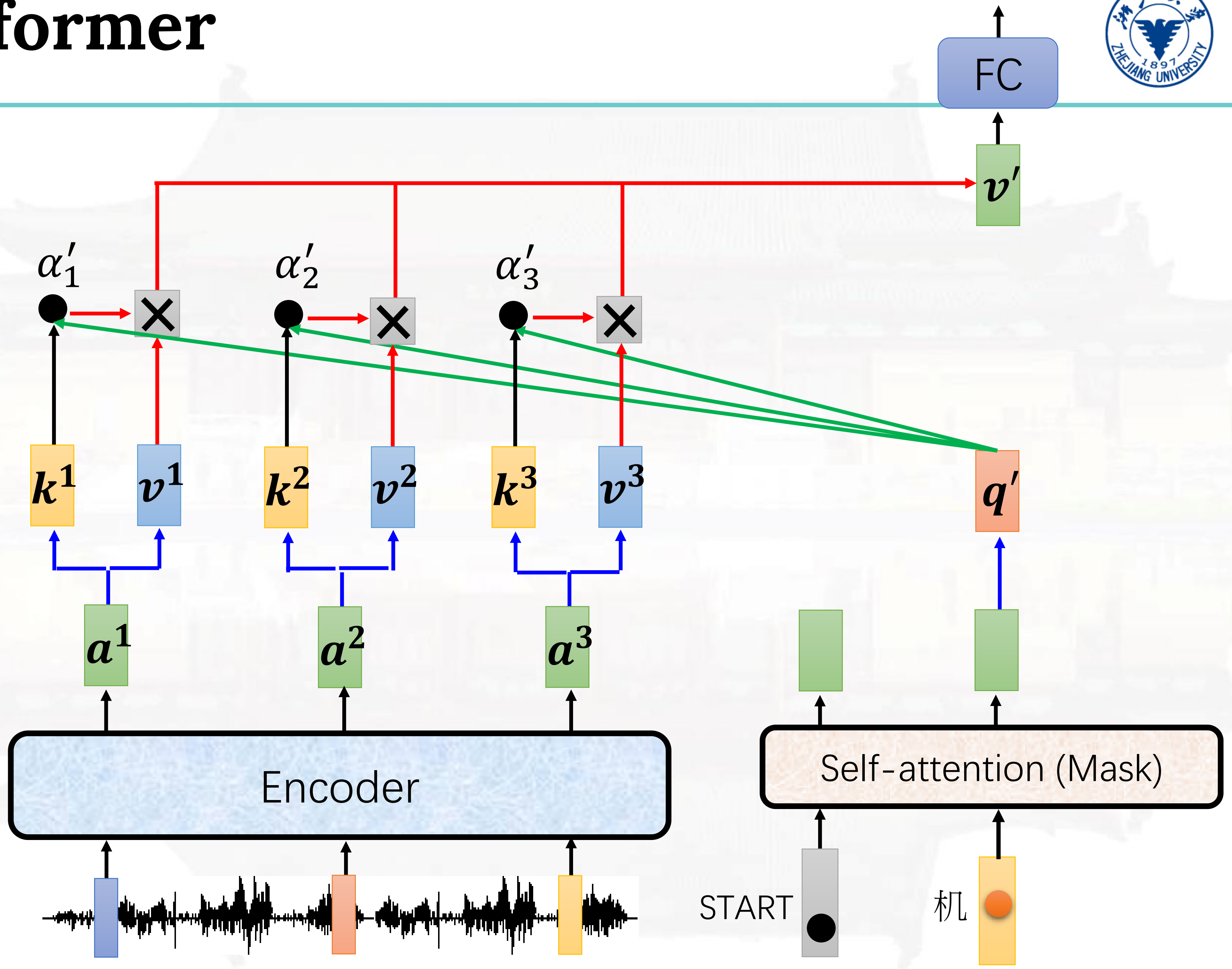
Transformer



Transformer

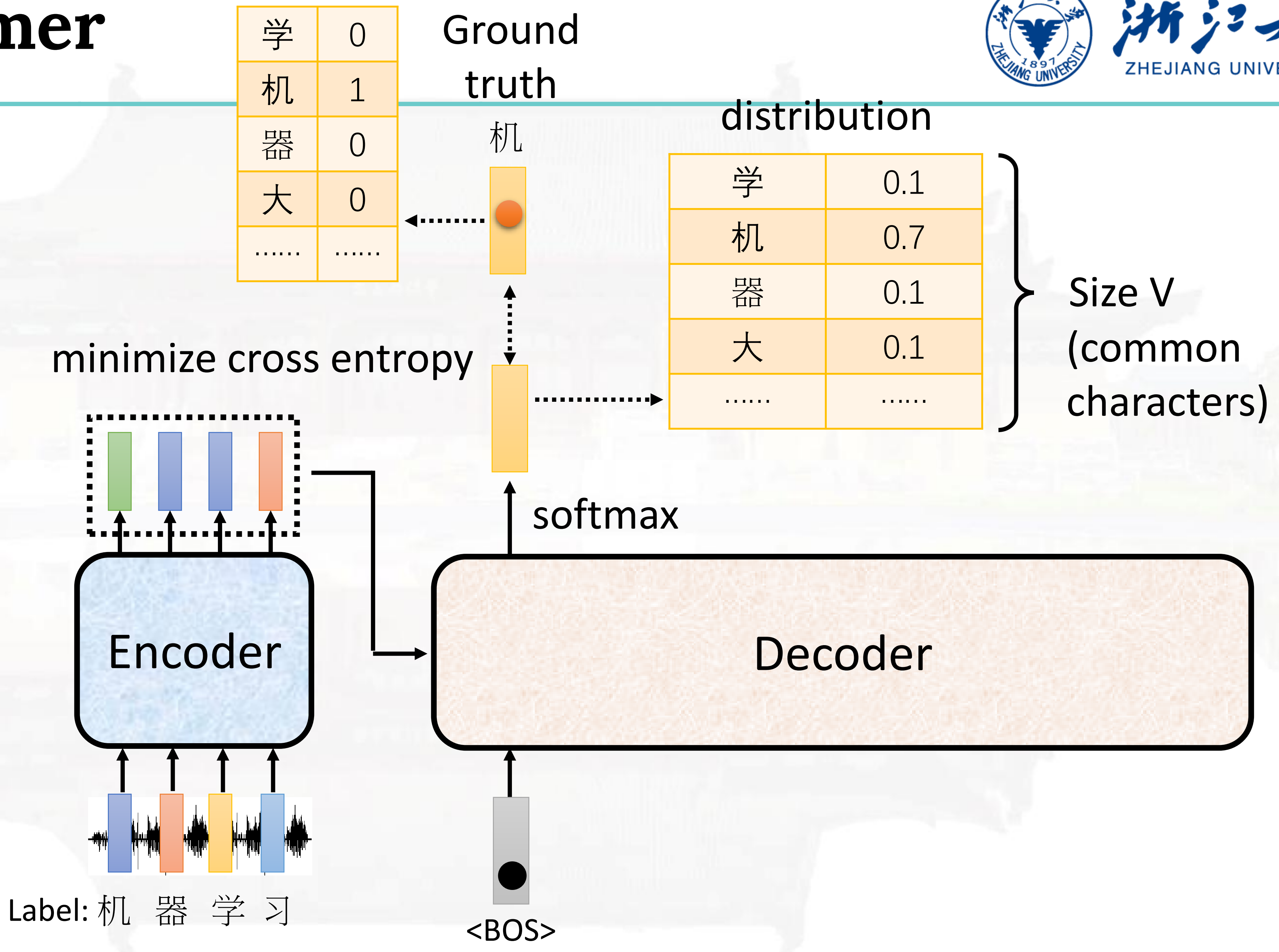


Transformer



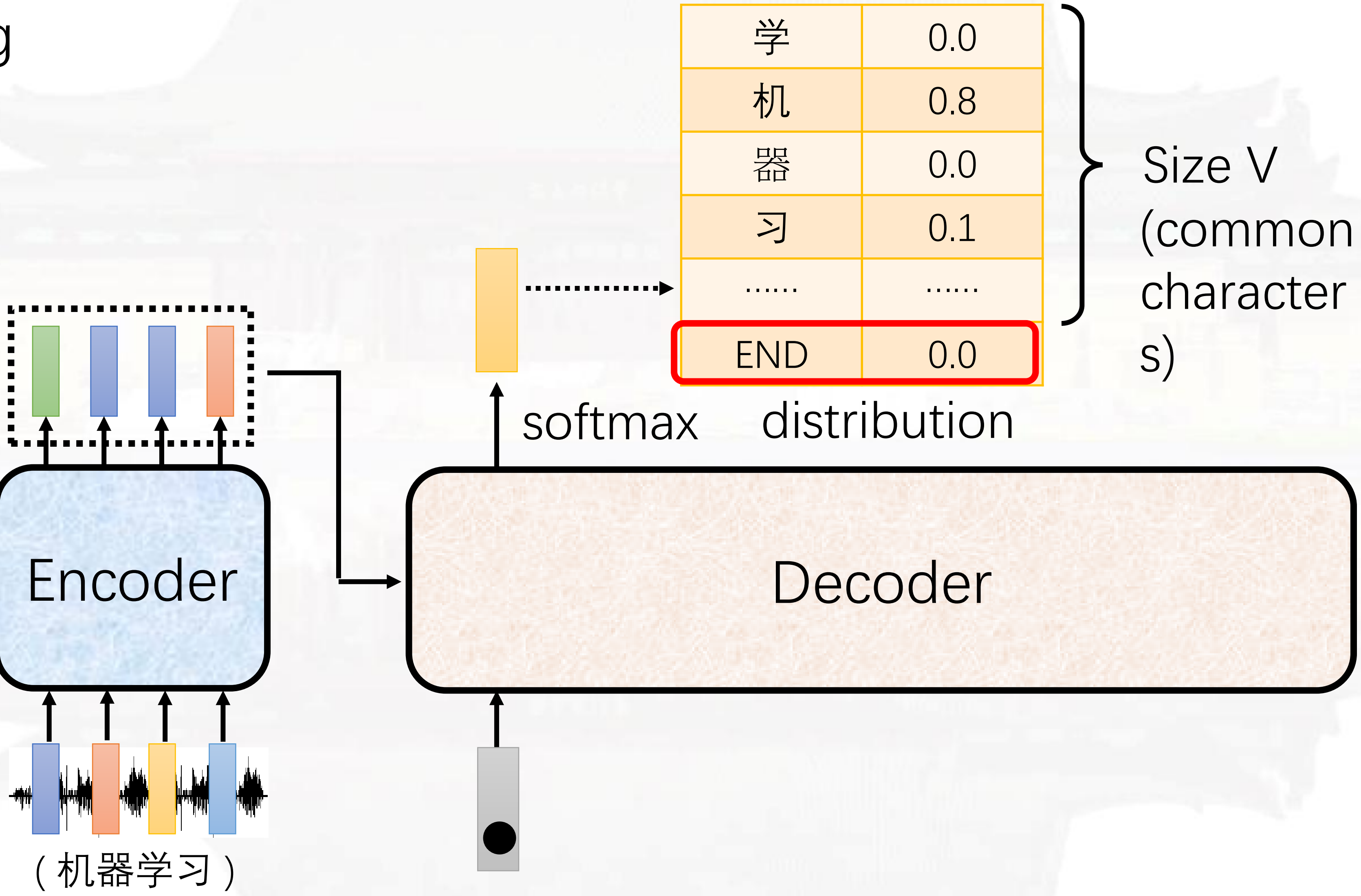
Transformer

- Training



Transformer

- Training



Transformer

- Training