AI/ML Bootcamp
Deep Learning

Overview

- Introduction to neural networks

Overview

- Introduction to neural networks
- Feedforward / Fully-connected neural networks or MLPs

Overview

- Introduction to neural networks
- Feedforward / Fully-connected neural networks or MLPs
- Convolutional neural networks (CNNs)

Overview

- Introduction to neural networks
- Feedforward / Fully-connected neural networks or MLPs
- Convolutional neural networks (CNNs)
- Recurrent neural networks (RNNs)

Overview

- Introduction to neural networks
- Feedforward / Fully-connected neural networks or MLPs
- Convolutional neural networks (CNNs)
- Recurrent neural networks (RNNs)
- Optimization in neural network training

Some applications

Robotics

Object detection

Reinforcement learning

Art

Image synthesis

Problem solving in video games

Some applications

Robotics

Object detection

Reinforcement learning

Art

Image synthesis

Problem solving in video games

Some applications

(Robotics usually solve problems in more controlled environments)

Robotics

Object detection

Reinforcement learning

Art

Image synthesis

Problem solving in video games

Some applications

(Self-driving cars: quite impressive by itself)

Robotics

Object detection

Reinforcement learning

Art

Image synthesis

Problem solving in video games

Some applications

(Multi-agent environments and Game AI: extreme number of possible games states, even to get a non-trivial solution)

Robotics

Object detection

Reinforcement learning

Art

Image synthesis

Problem solving in video games

Some applications

Robotics

Object detection

Reinforcement learning

(Image synthesis and generating art is considered a milestone in getting closer to human AI and perception)

Art

Image synthesis

Problem solving in video games

Some applications

All these problems involve gazillion variables to be considered ‘easy solvable’!

Robotics

Object detection

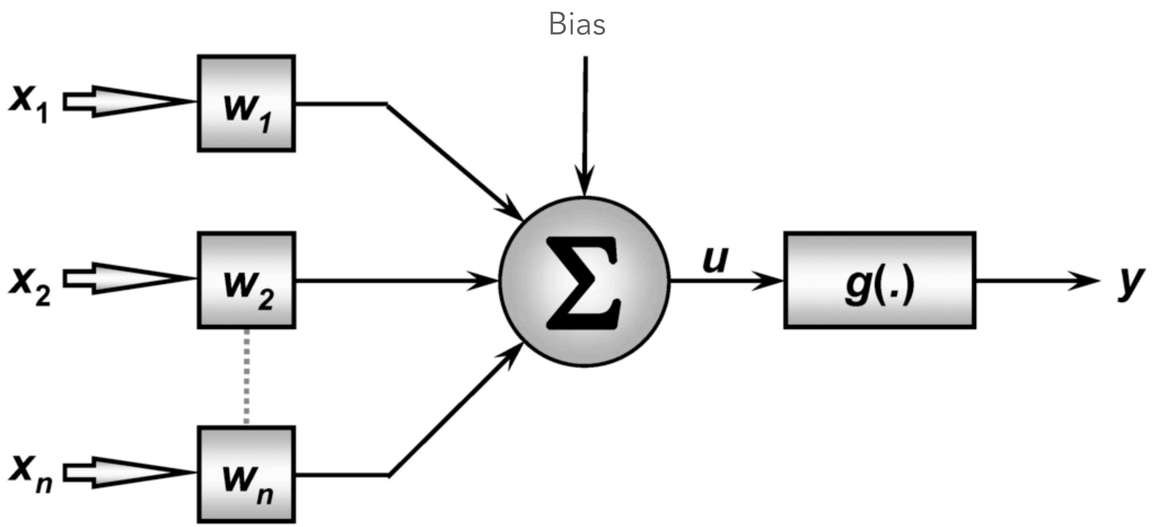
Reinforcement learning

Art

Image synthesis

Problem solving in video games

How did we end up with deep learning: Perceptron

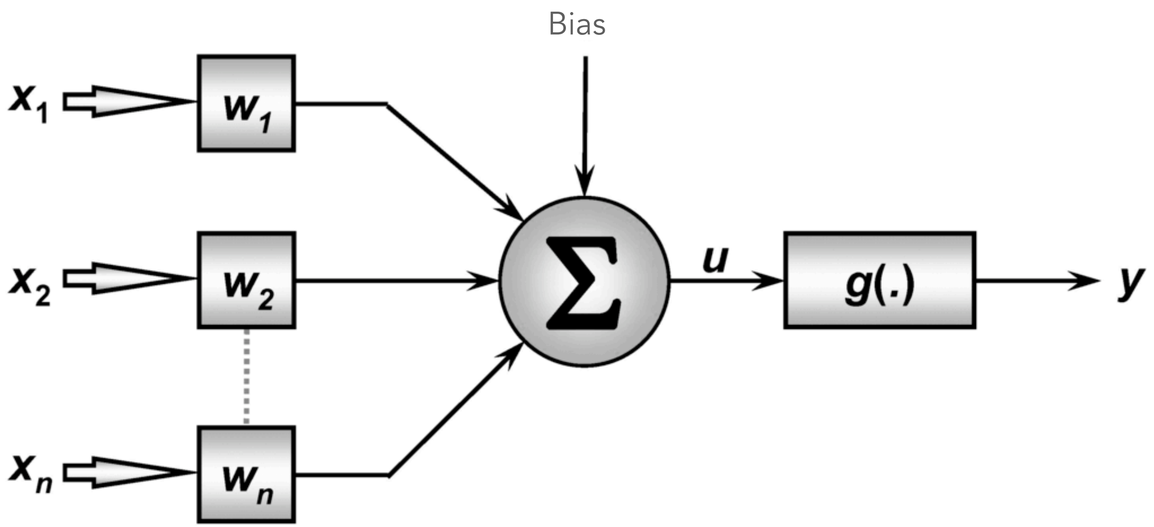


[Rosenblatt, Wightman; 1958]

[Widrow, Hoff; 1960]

[Minsky, Papert; 1969]

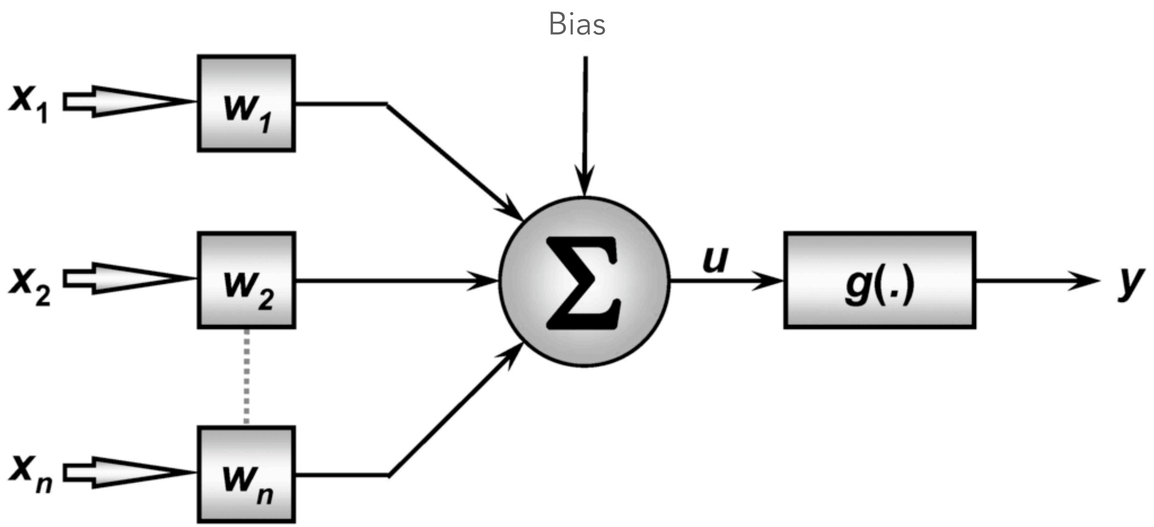
How did we end up with deep learning: Perceptron



[Rosenblatt, Wightman; 1958]
[Widrow, Hoff; 1960]
[Minsky, Papert; 1969]

- Original Perceptron was proposed for binary classification

How did we end up with deep learning: Perceptron

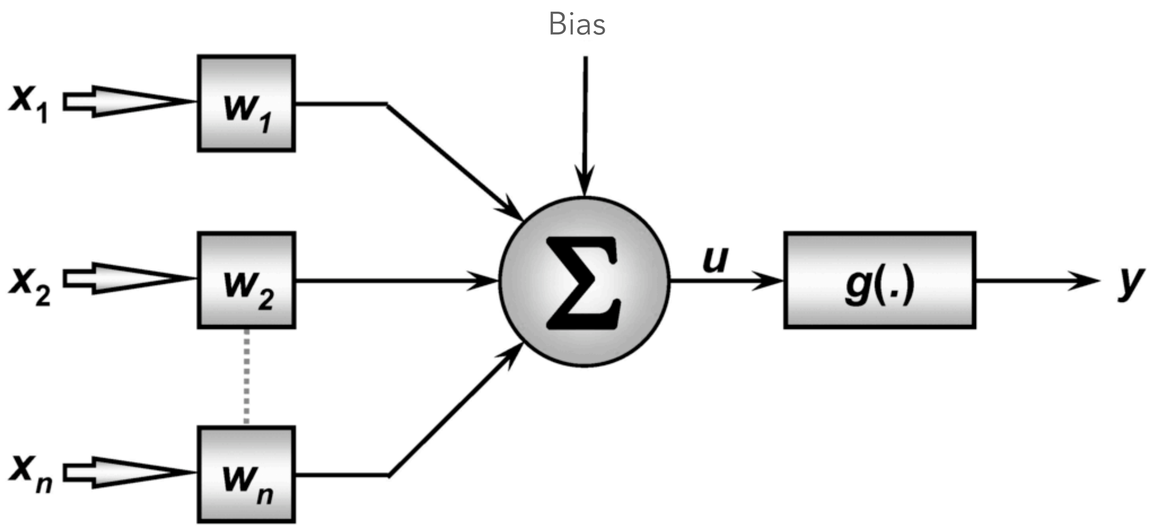


[Rosenblatt, Wightman; 1958]
[Widrow, Hoff; 1960]
[Minsky, Papert; 1969]

- Original Perceptron was proposed for binary classification
- Setting: given a set of input signals $x = (x_1, x_2, \ldots, x_n)^\top$, we update a set of weights $(w_1, w_2, \ldots, w_n)^\top$ to decide between two classes, based on the quantized output:

$$u = \sum_{i=1}^{n} w_i x_i, \quad \text{output} = g(u) = \texttt{quant}(u)$$

How did we end up with deep learning: Perceptron



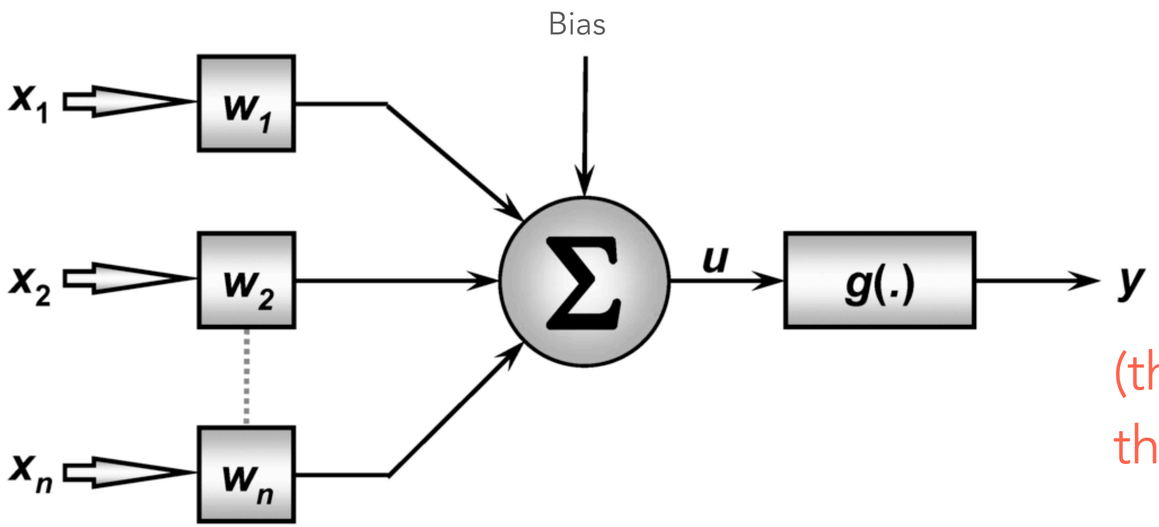
[Rosenblatt, Wightman; 1958]
[Widrow, Hoff; 1960]
[Minsky, Papert; 1969]

- Original Perceptron was proposed for binary classification
- Setting: given a set of input signals $x = (x_1, x_2, \ldots, x_n)^\top$, we update a set of weights $(w_1, w_2, \ldots, w_n)^\top$ to decide between two classes, based on the quantized output:

(If quantized output is greater than threshold: Class +1
Otherwise: Class -1)

$$u = \sum_{i=1}^{n} w_i x_i, \quad \text{output} = g(u) = \texttt{quant}(u)$$

How did we end up with deep learning: Perceptron



[Rosenblatt, Wightman; 1958]
[Widrow, Hoff; 1960]
[Minsky, Papert; 1969]

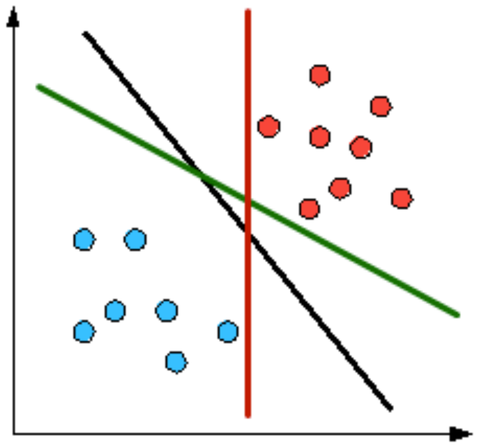
(this is model description; we have not discussed the algorithm to train/solve this problem)

- Original Perceptron was proposed for binary classification
- Setting: given a set of input signals $x = (x_1, x_2, \ldots, x_n)^\top$, we update a set of weights $(w_1, w_2, \ldots, w_n)^\top$ to decide between two classes, based on the quantized output:

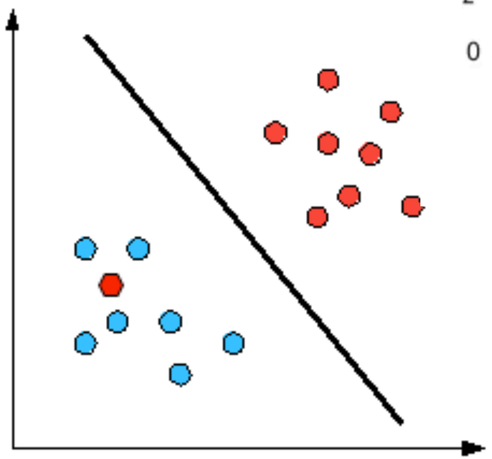
(If quantized output is greater than threshold: Class +1
Otherwise: Class -1)

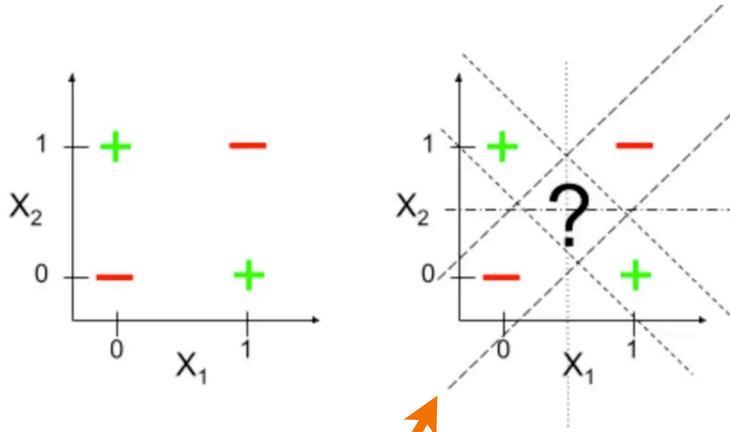
$$u = \sum_{i=1}^{n} w_i x_i, \quad \text{output} = g(u) = \texttt{quant}(u)$$

Potential issues with Perceptron

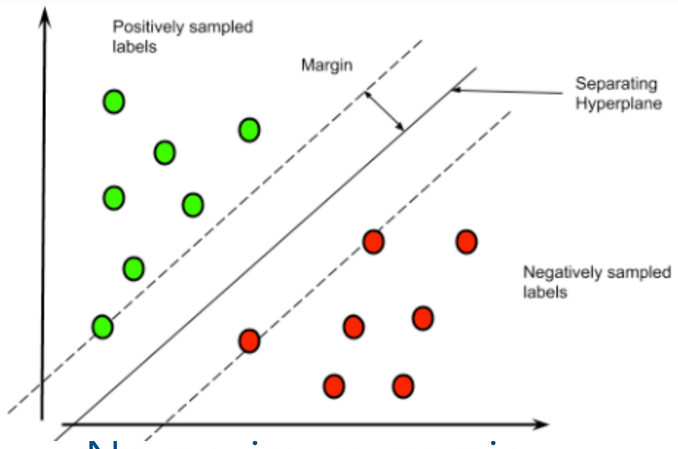


Assumes data is linearly separable





Makes no non-linear 'decisions'



No maximum margin separation

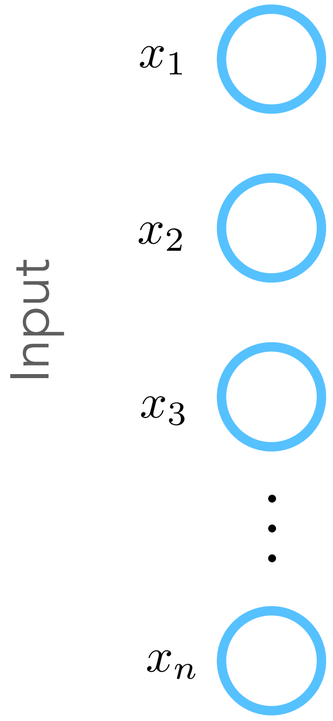
Works for binary classification only

Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network

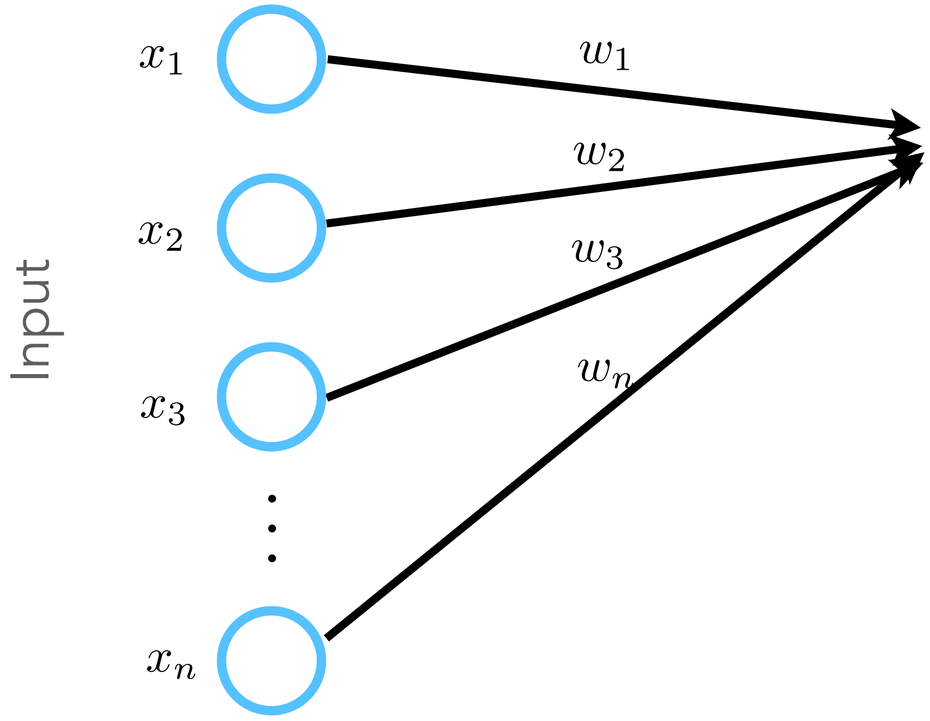
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network



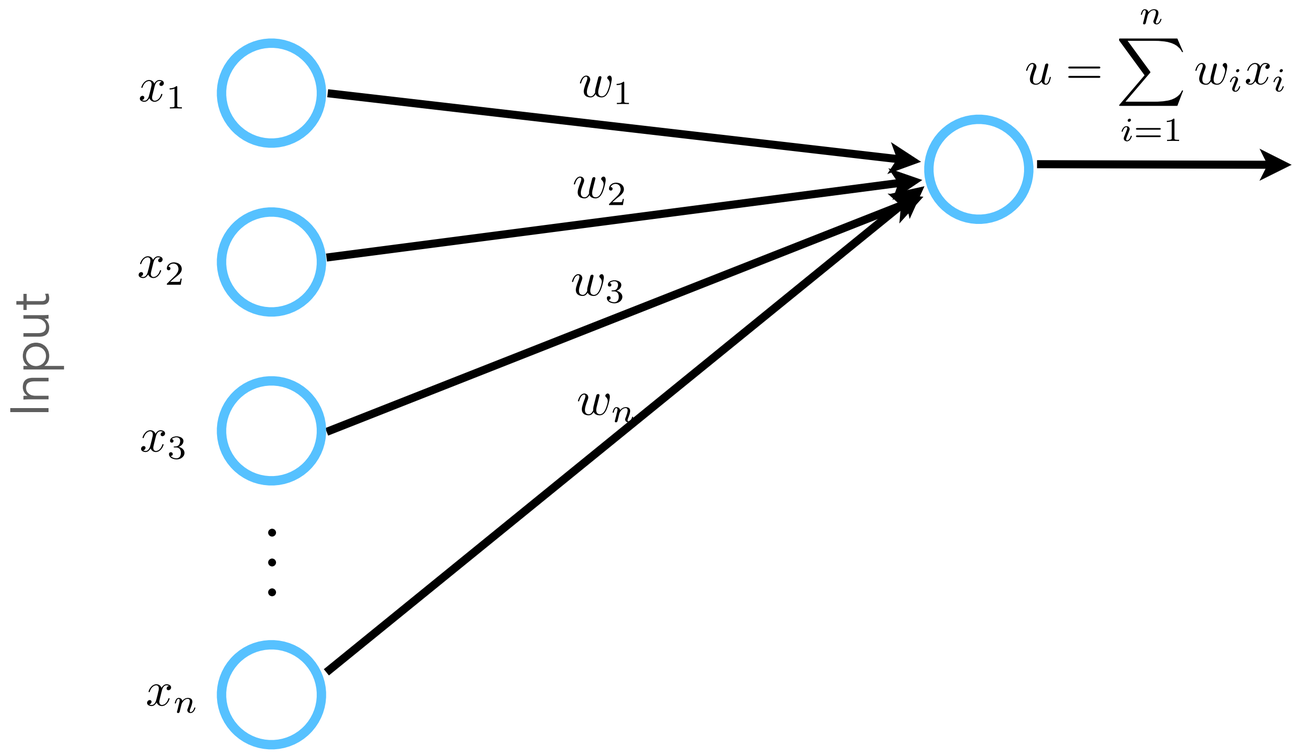
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network



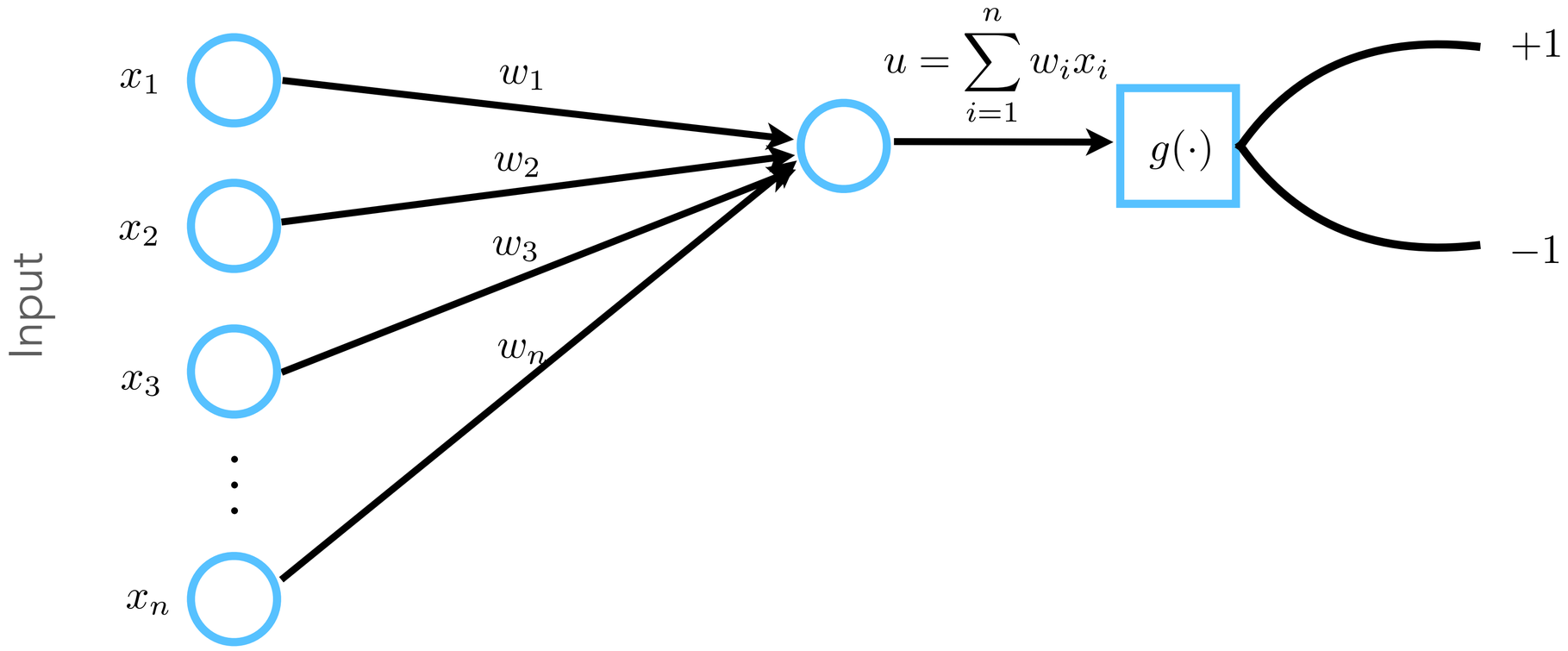
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network



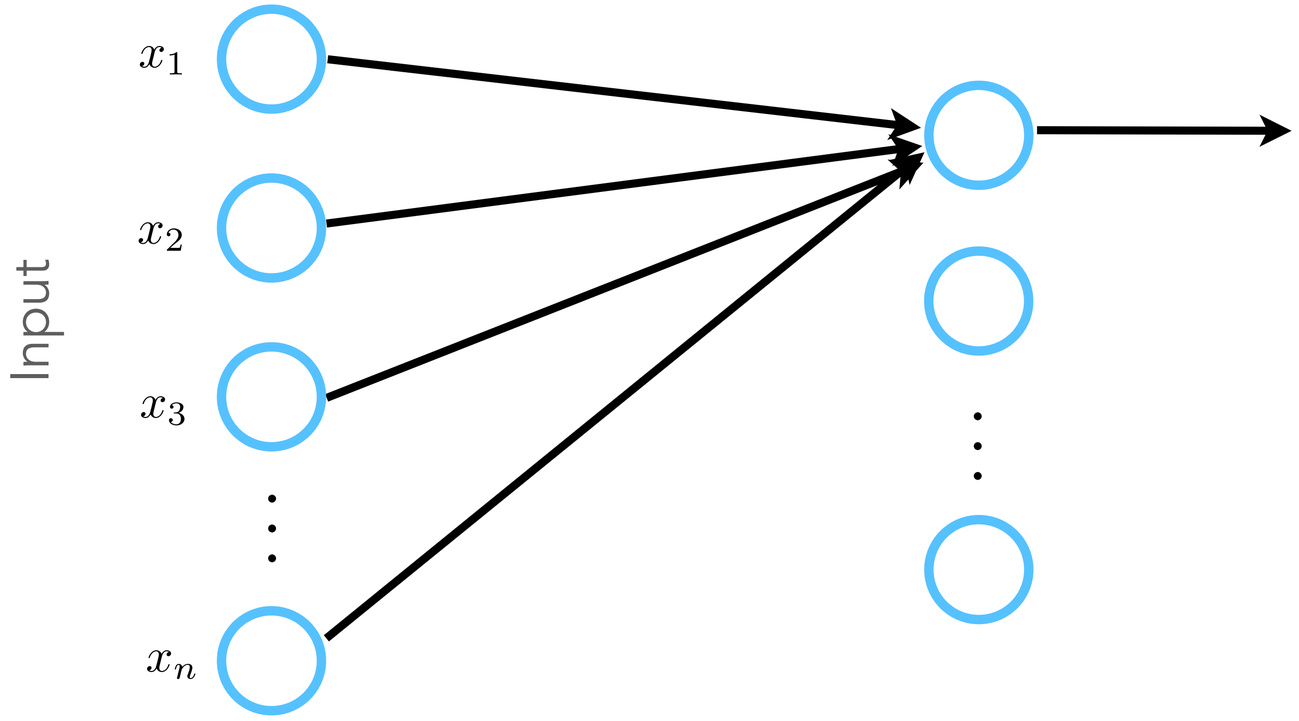
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network



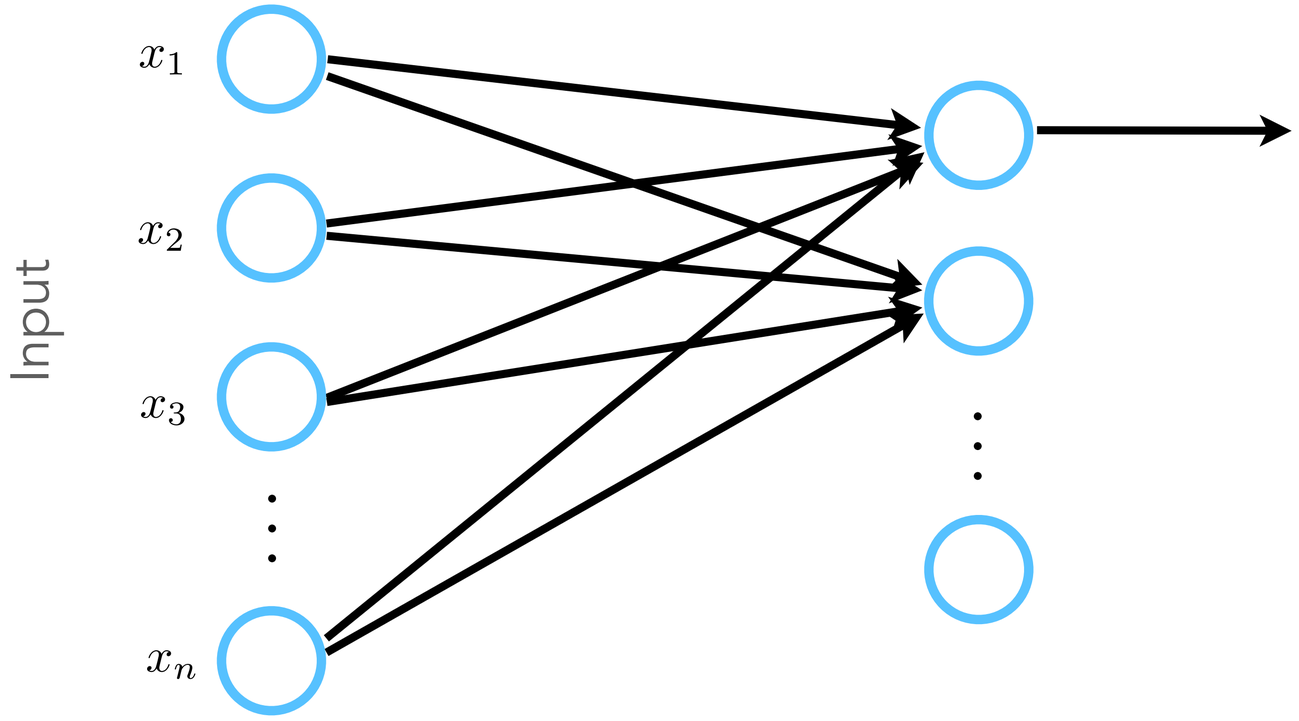
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network



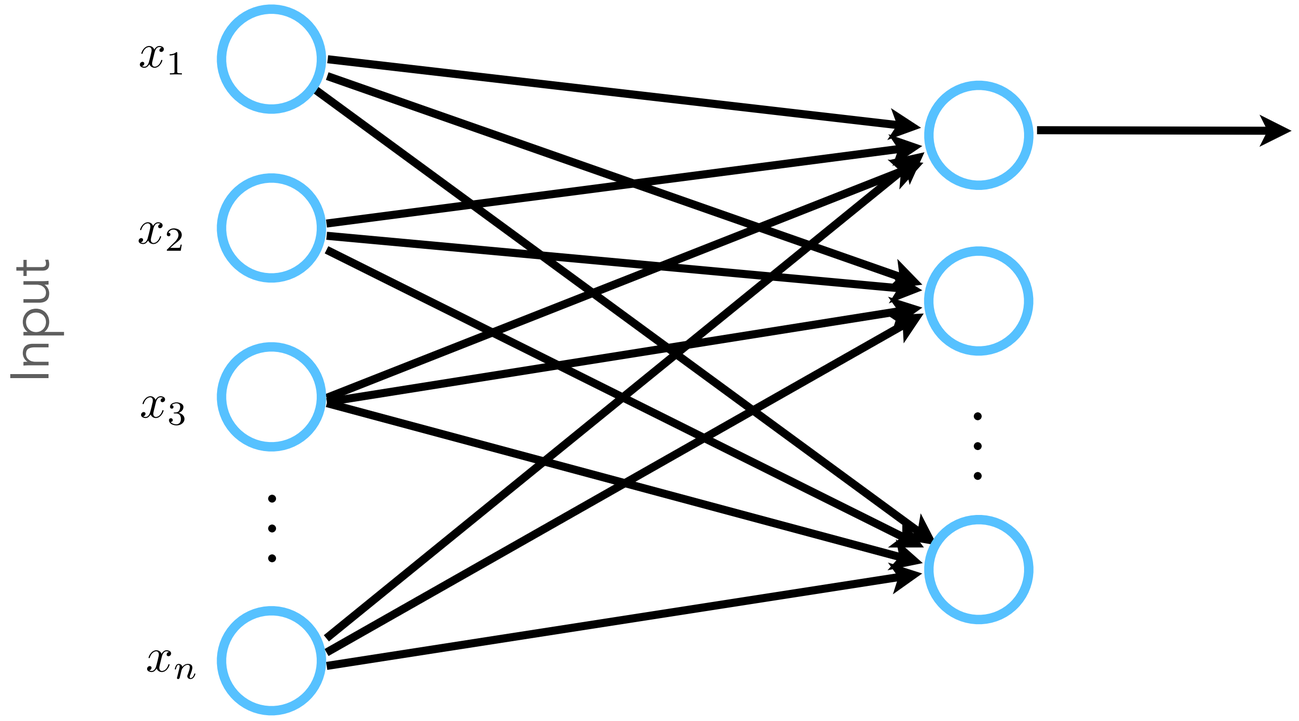
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network



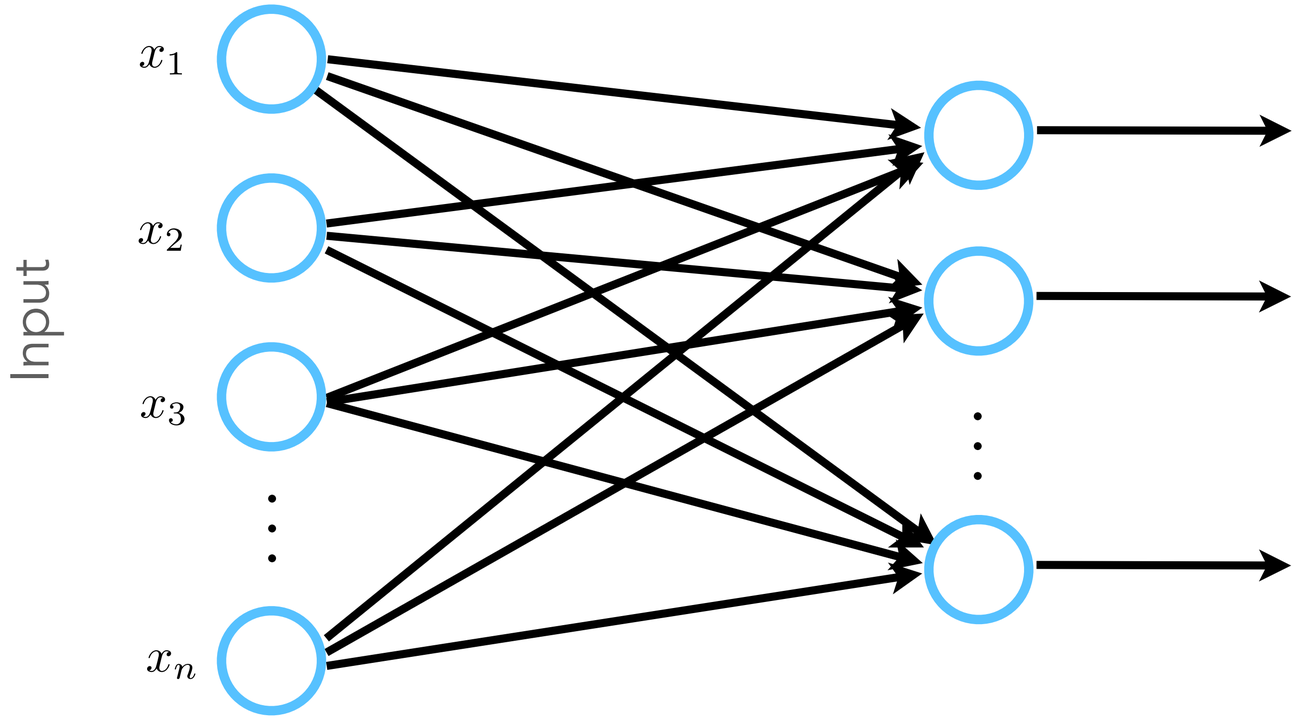
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network



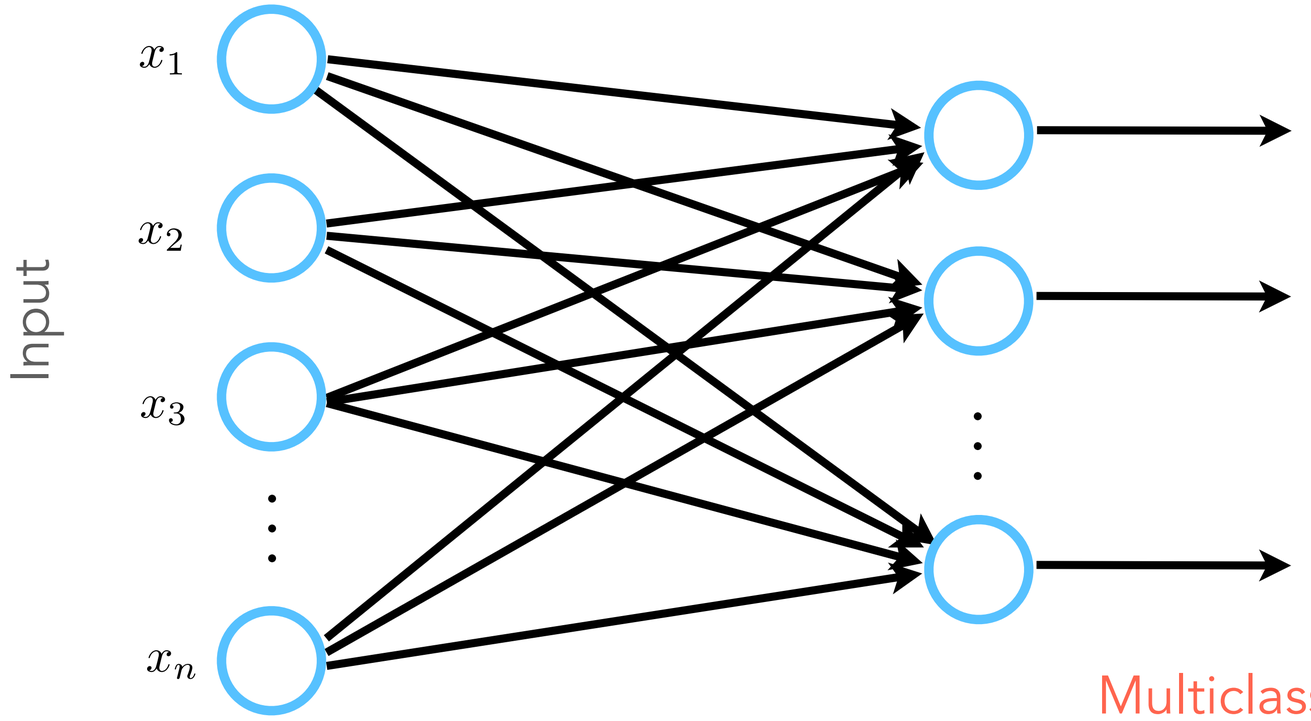
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network



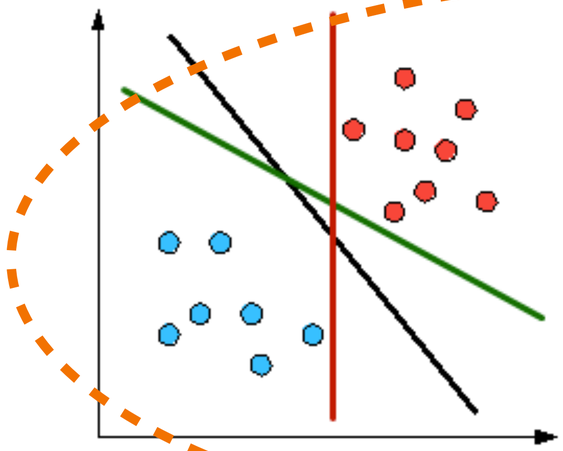
Multiclass classification with Perceptron

- Equivalent to no-hidden layer (one layer) feed-forward neural network

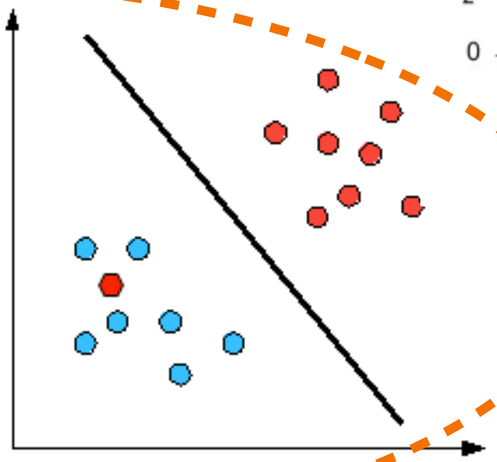


Multiclass classification!

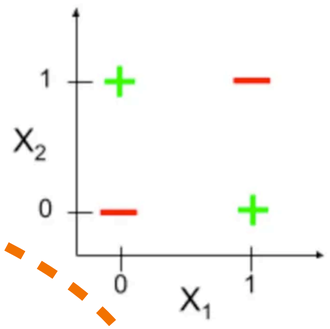
Potential issues with Perceptron

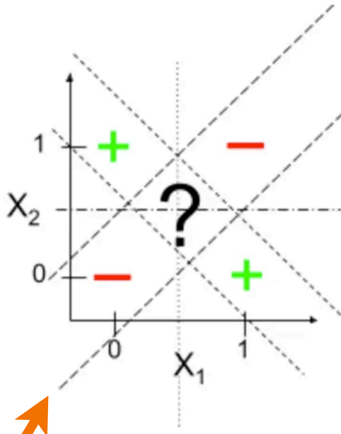


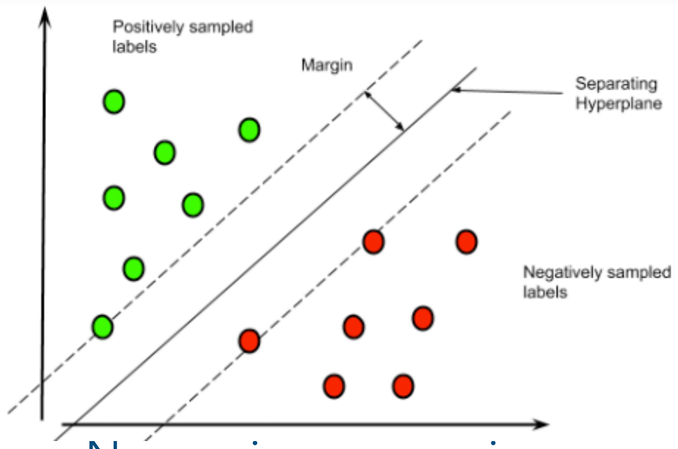
Assumes data is linearly separable



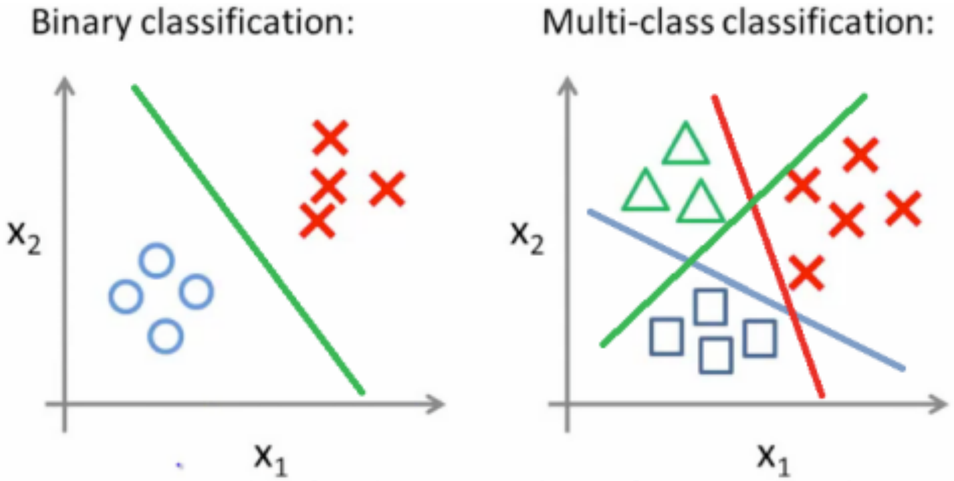
Makes no non-linear 'decisions'







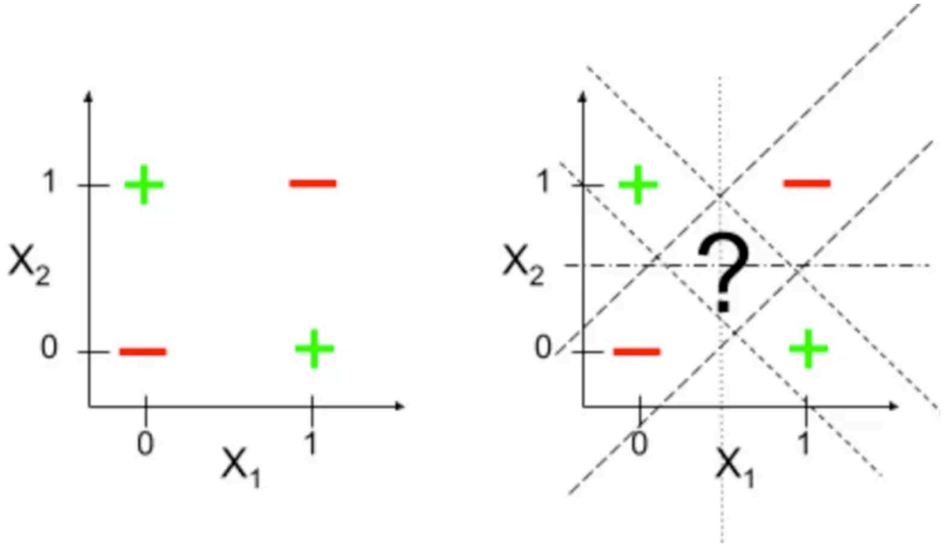
No maximum margin separation



Works for binary classification only

Potential issues with Perceptron

[Minsky, Papert; 1969]



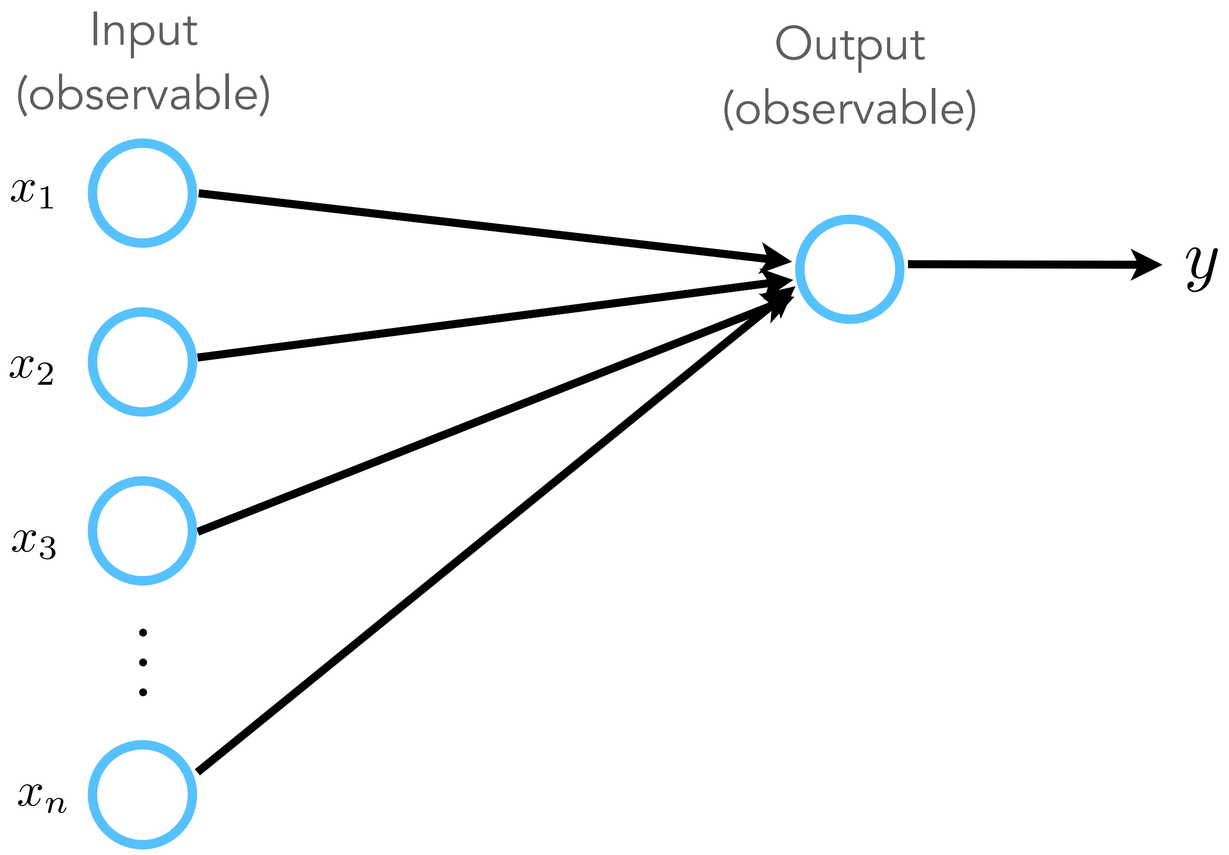
There is no line decision boundary to solve this problem

- 2D function XOR cannot be handled with single-layer perceptrons

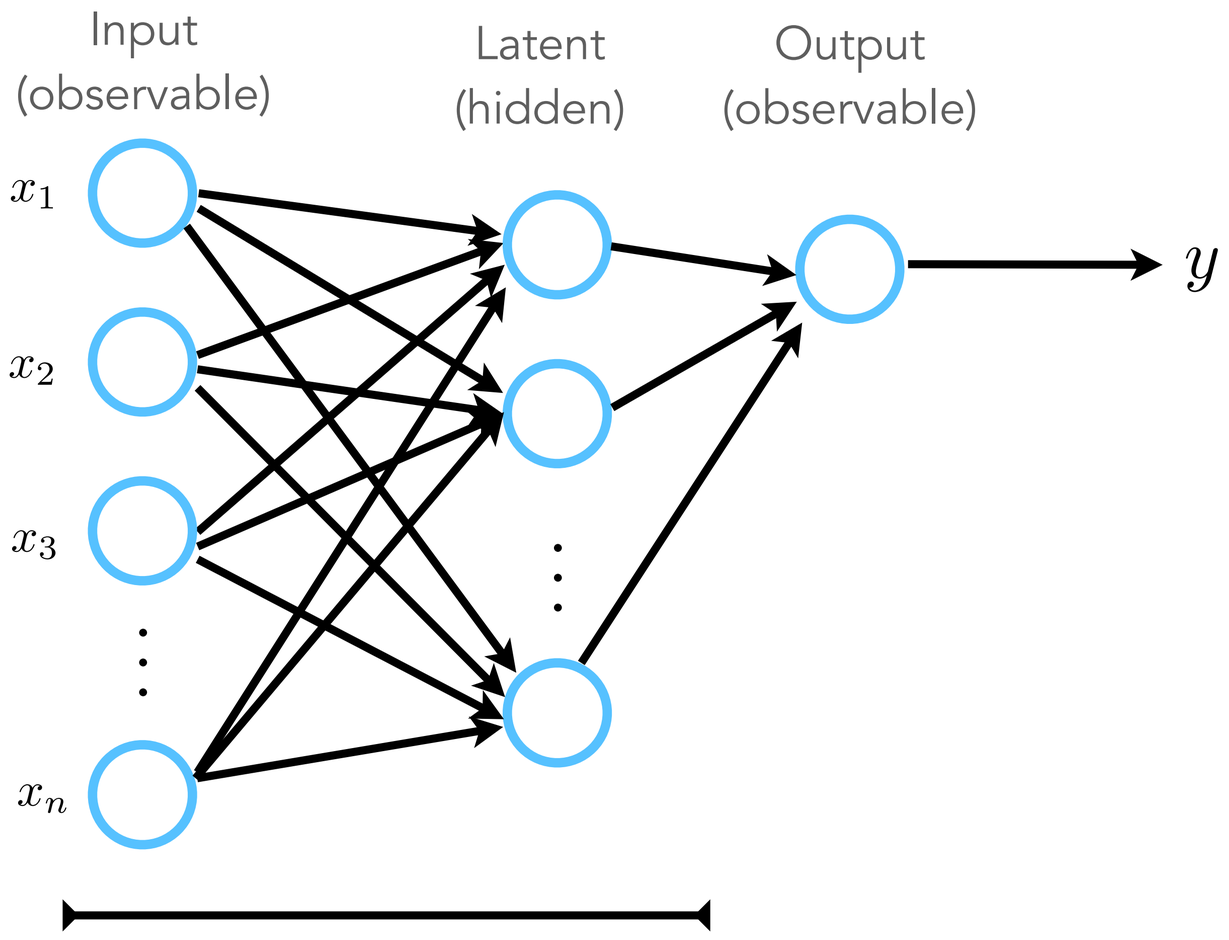
$$f(x_1, x_2) = x_1 \oplus x_2 \longrightarrow \begin{array}{l} x_1 = 0 \ \& \ x_2 = 0 \rightarrow f(x_1, x_2) = 0 \\ x_1 = 0 \ \& \ x_2 = 1 \rightarrow f(x_1, x_2) = 1 \\ x_1 = 1 \ \& \ x_2 = 0 \rightarrow f(x_1, x_2) = 1 \\ x_1 = 1 \ \& \ x_2 = 1 \rightarrow f(x_1, x_2) = 0 \end{array}$$

- Single layer perceptron cannot solve XOR
- Multi-layer perceptron considered in [Minsky; 1960's], however no clear way algorithm to train
- This was enough to "kill the hype" around perceptrons

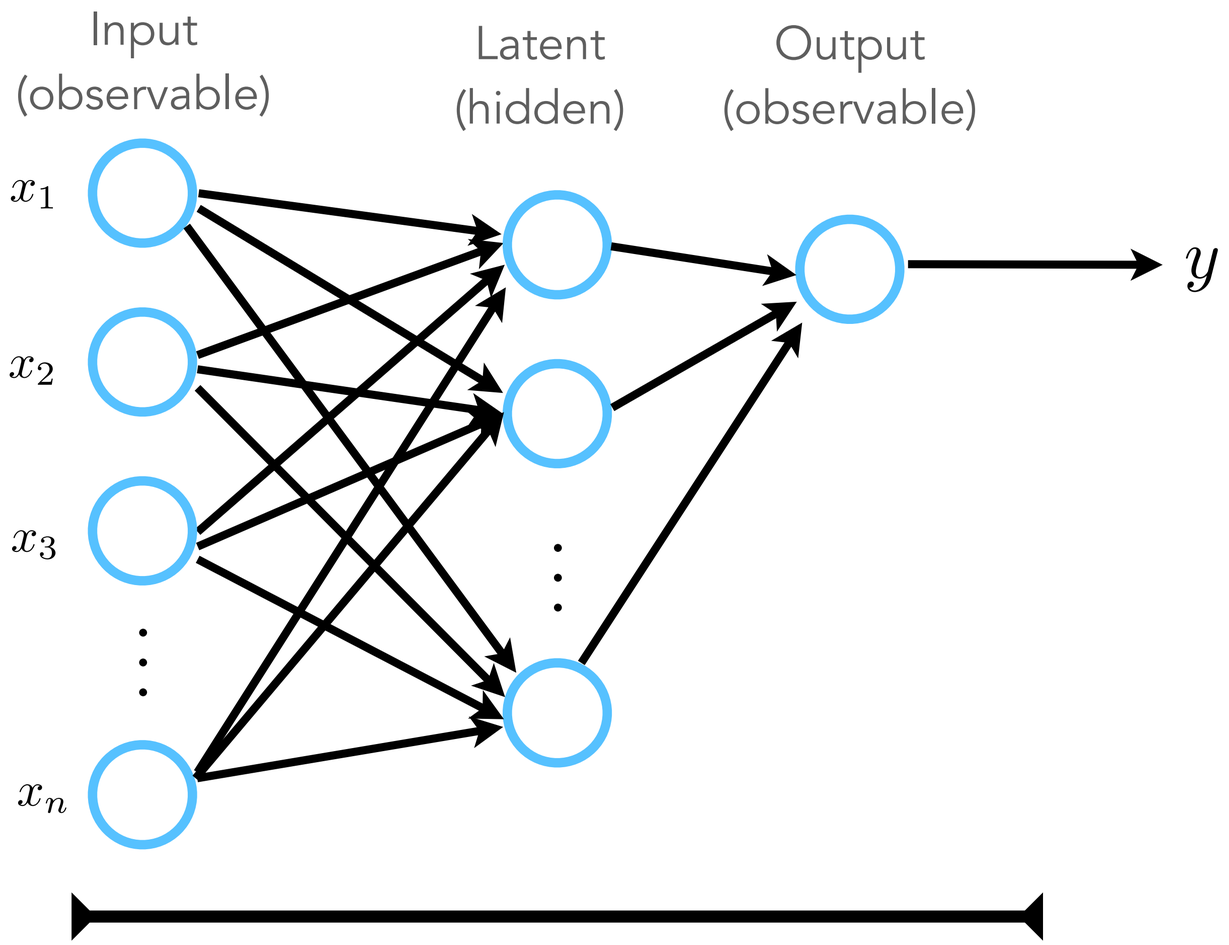
What is a multi-layer perceptron?



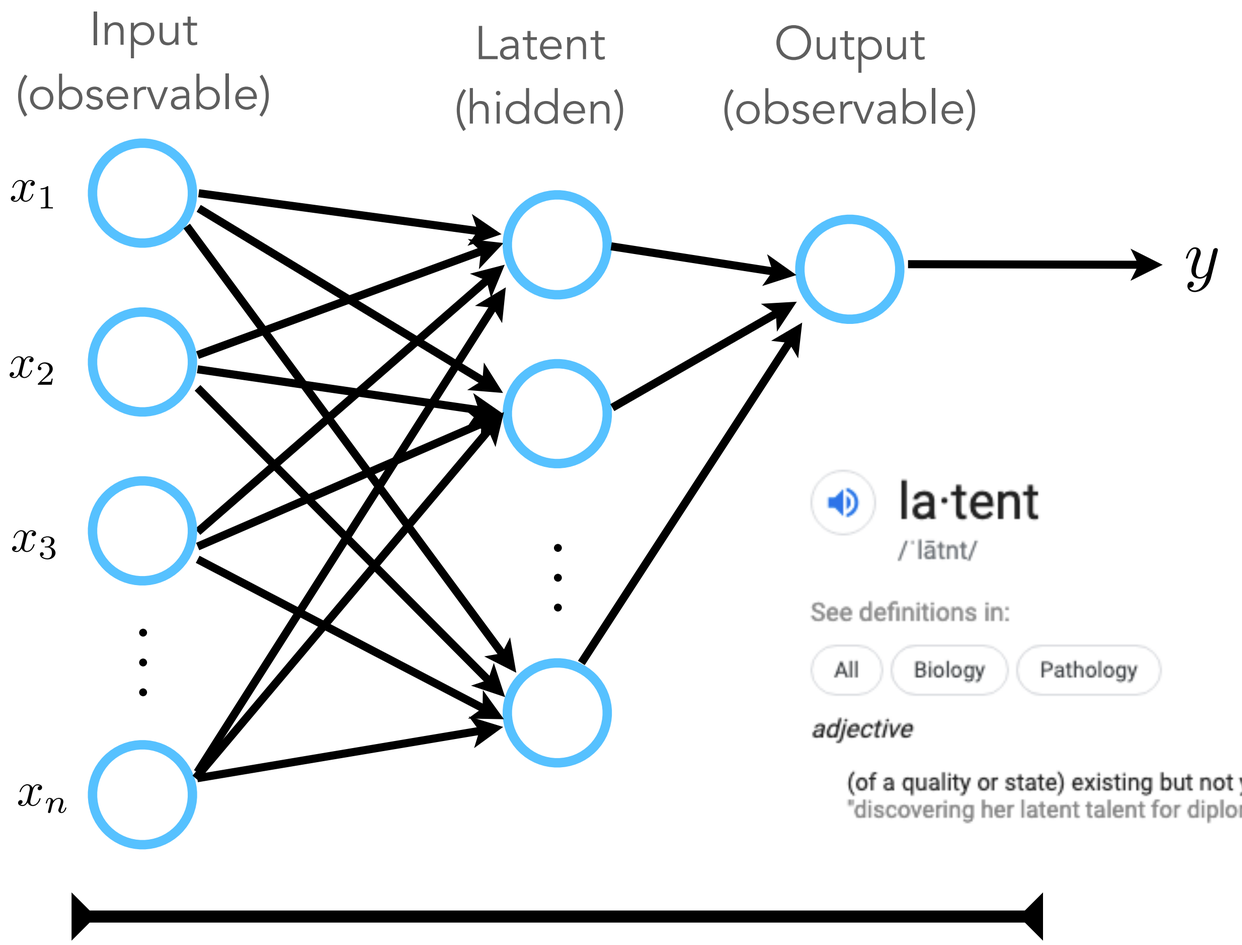
What is a multi-layer perceptron?

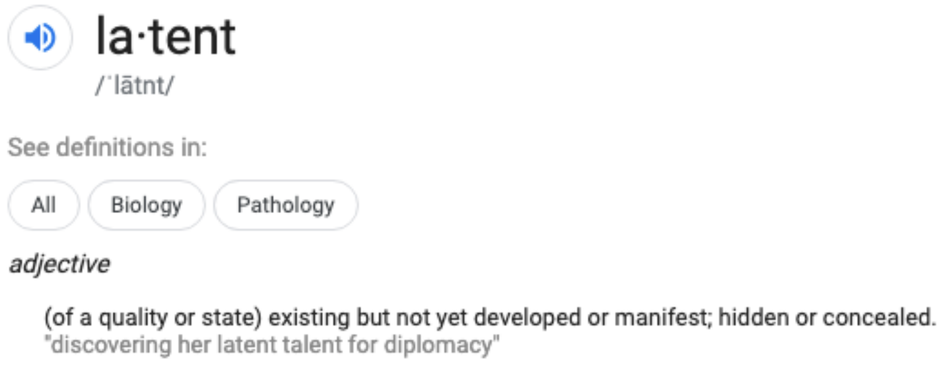


What is a multi-layer perceptron?



What is a multi-layer perceptron?



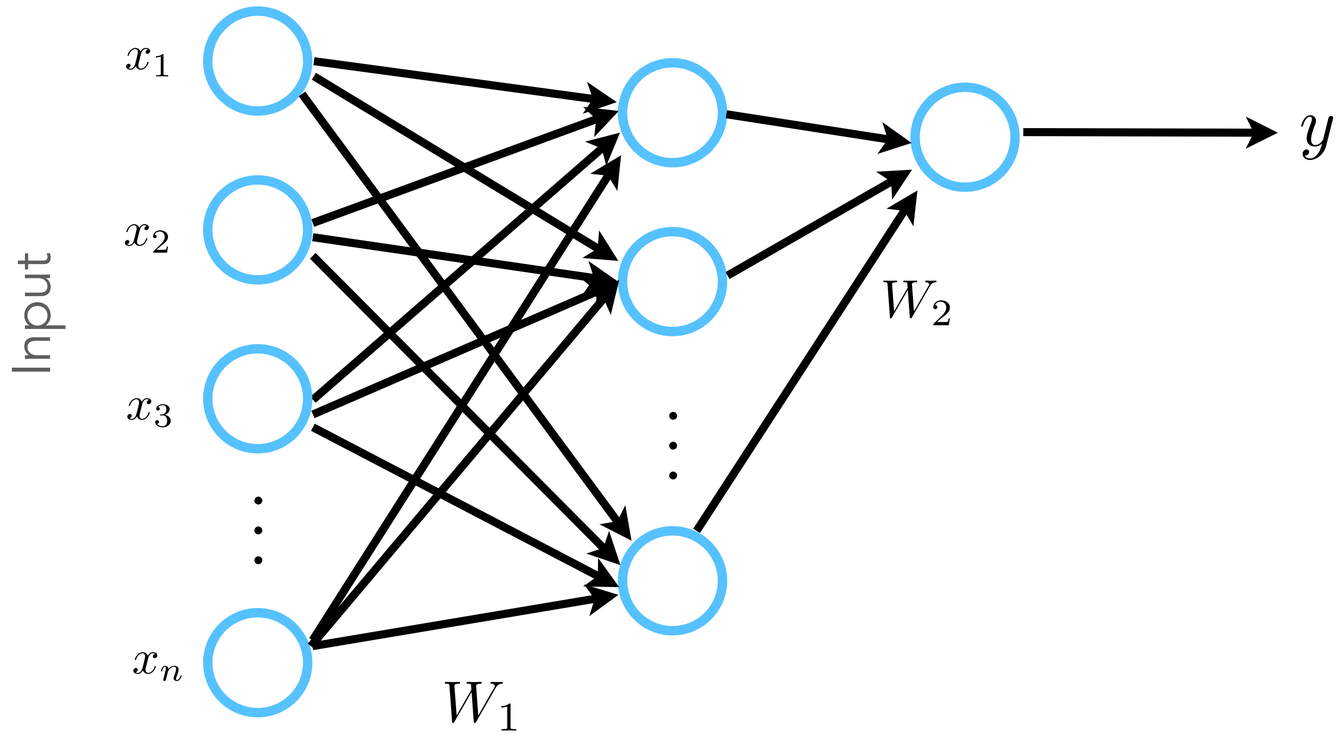


But is this enough?

- Assume we got a way to train multilayer perceptrons

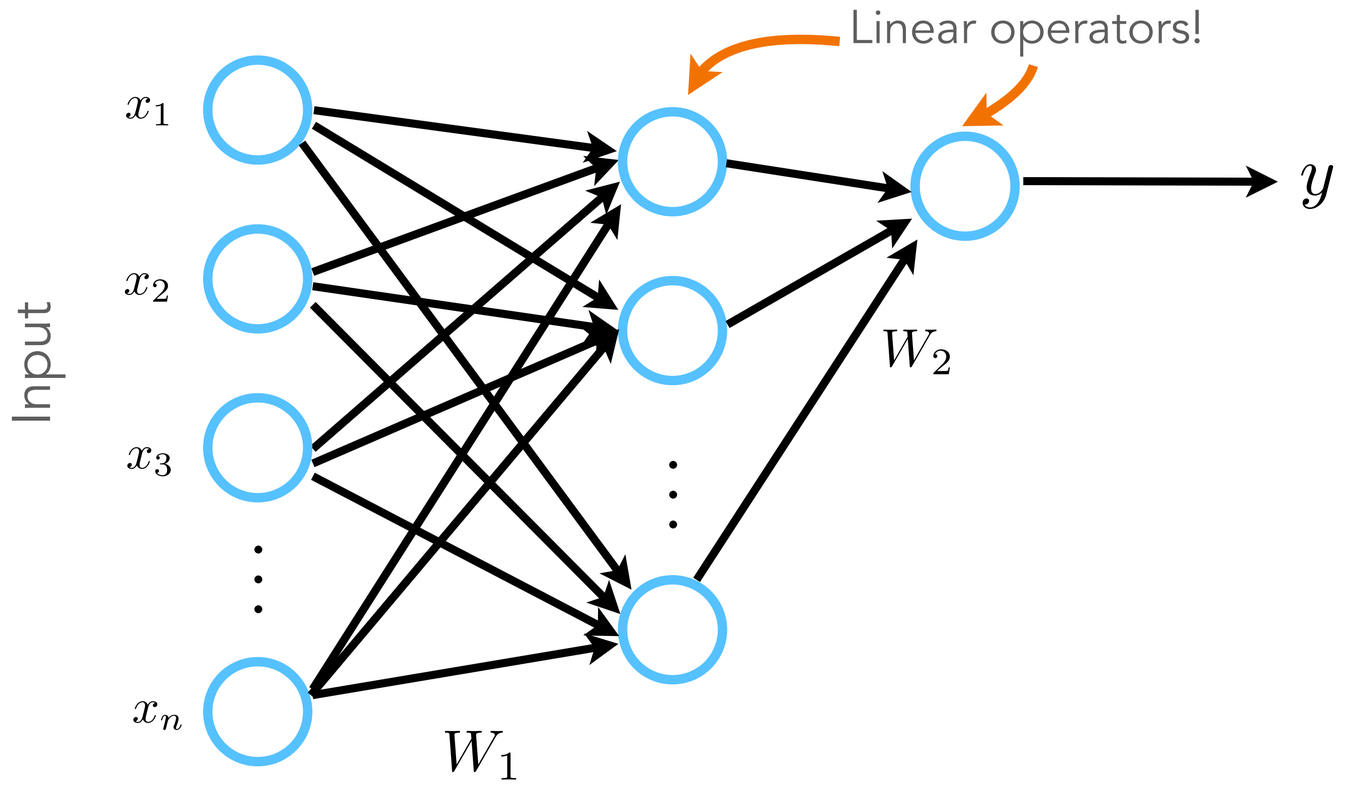
But is this enough?

- Assume we got a way to train multilayer perceptrons



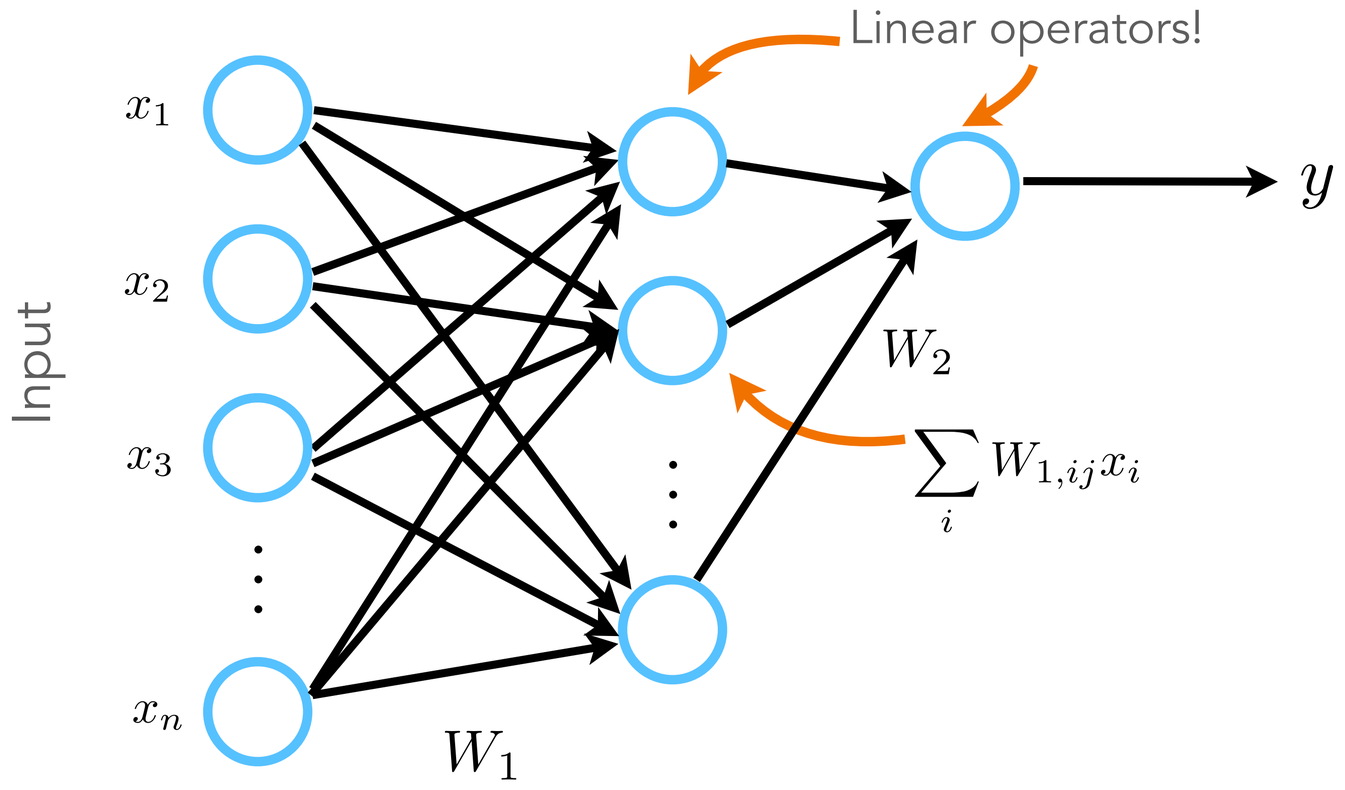
But is this enough?

- Assume we got a way to train multilayer perceptrons



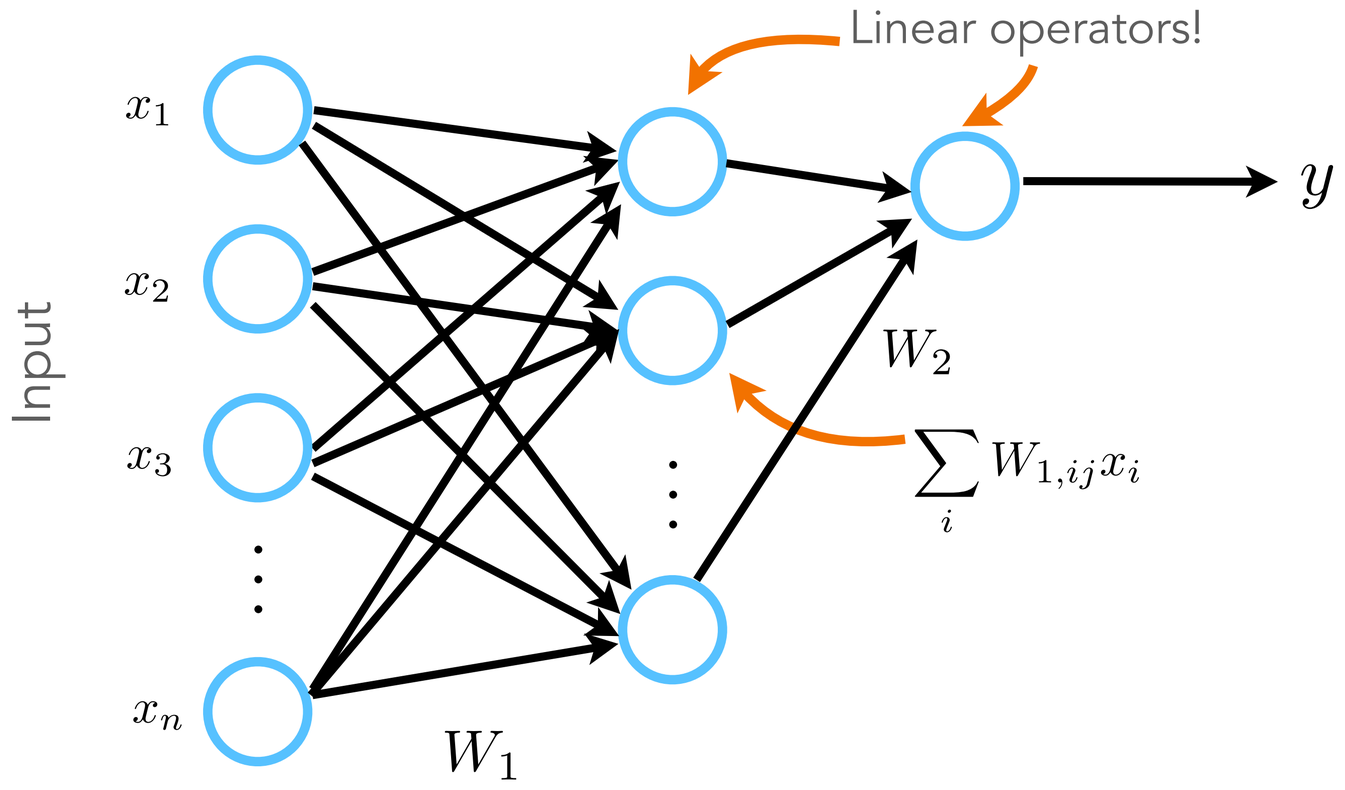
But is this enough?

- Assume we got a way to train multilayer perceptrons



But is this enough?

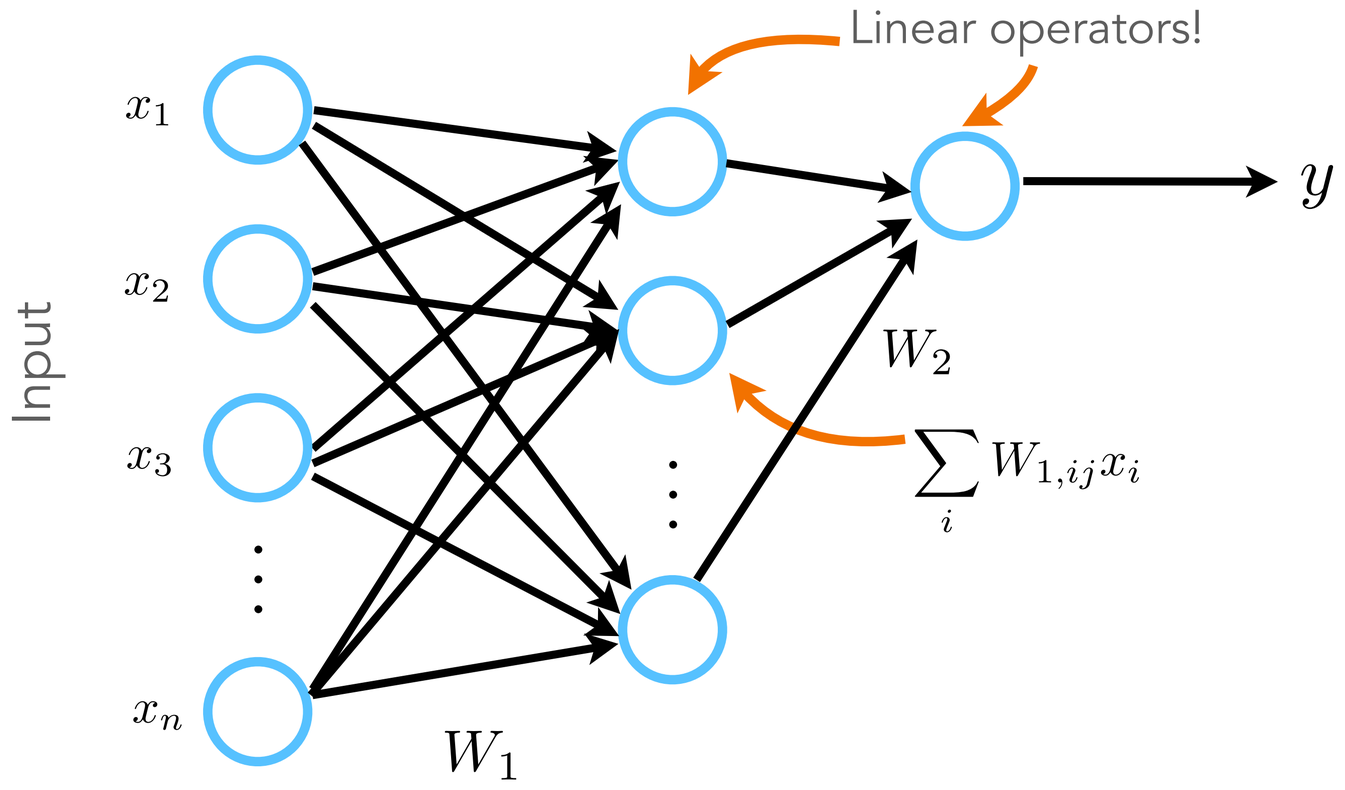
- Assume we got a way to train multilayer perceptrons



$$W_L \cdots W_2 \cdot W_1 x = W x$$

But is this enough?

- Assume we got a way to train multilayer perceptrons



$$W_L \cdots W_2 \cdot W_1 x = W x$$

$$||$$

Equal to having one layer perceptron

(In words, the expressivity power of this perceptron is limited by the linearity of operations)

What is a non-linear function?

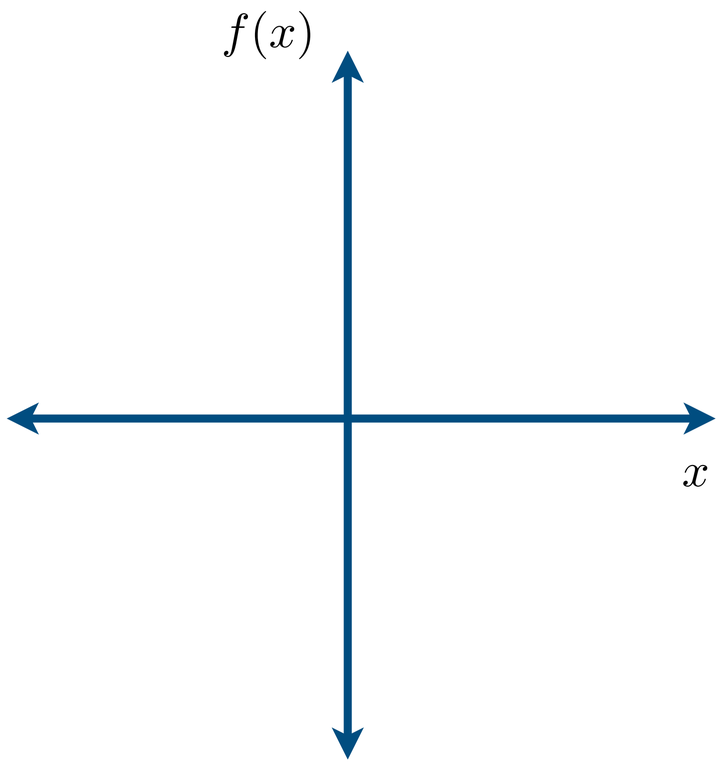
(More in the slides following)

- Non-linear function: a function that affects input in a non-linear way.

What is a non-linear function?

(More in the slides following)

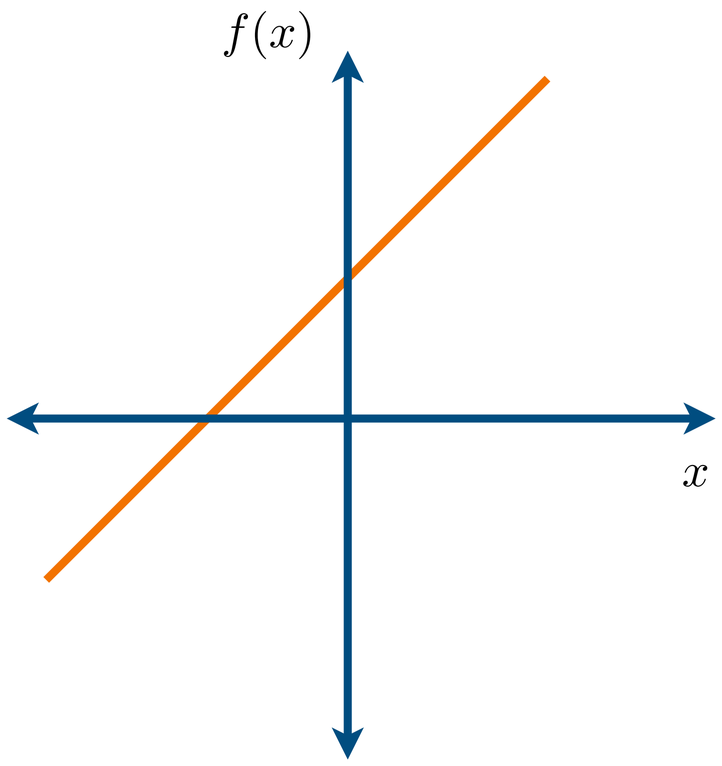
- Non-linear function: a function that affects input in a non-linear way.



What is a non-linear function?

(More in the slides following)

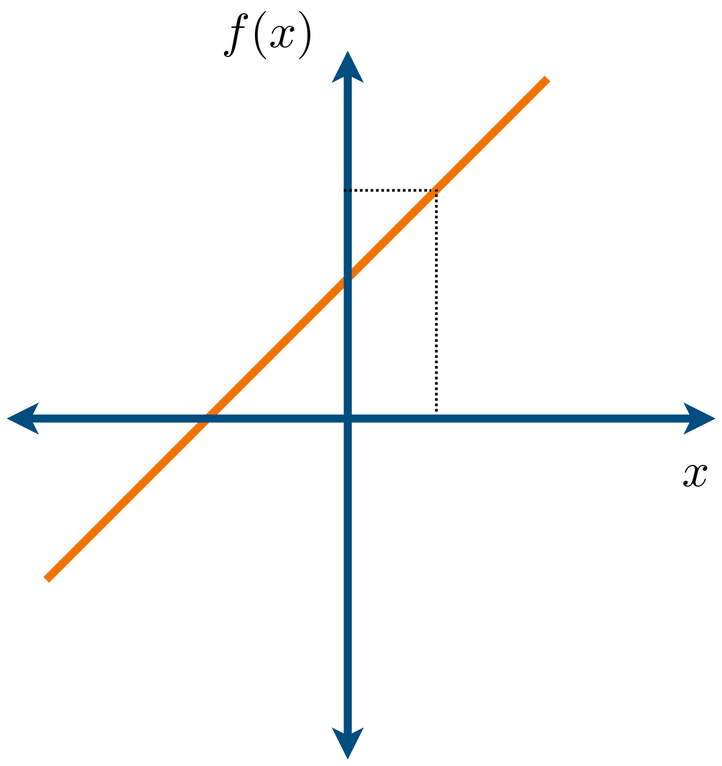
- Non-linear function: a function that affects input in a non-linear way.



What is a non-linear function?

(More in the slides following)

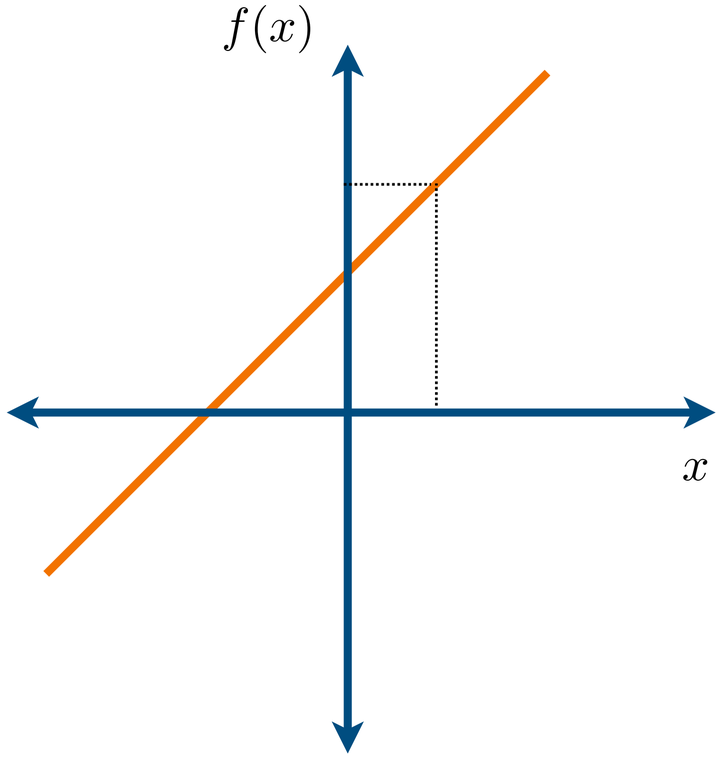
- Non-linear function: a function that affects input in a non-linear way.

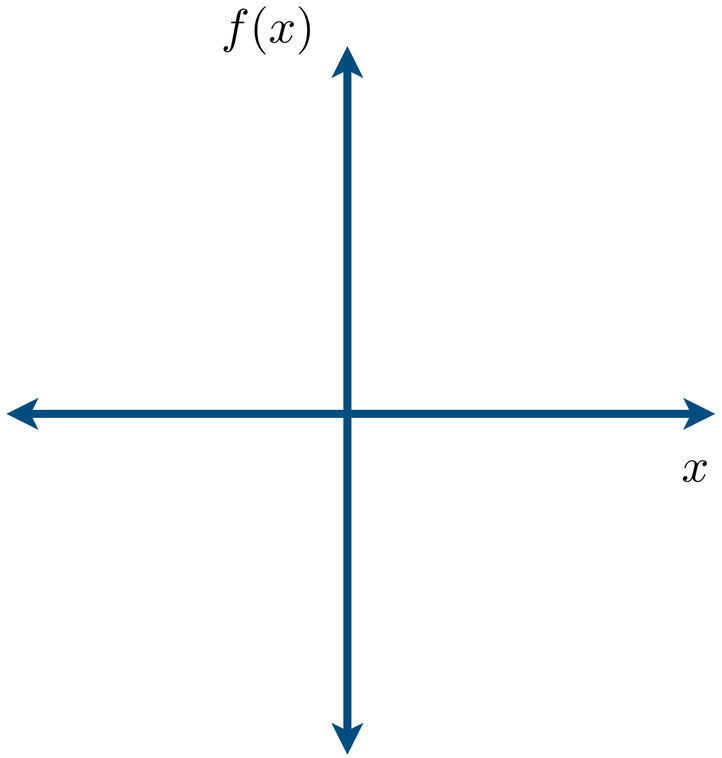


What is a non-linear function?

(More in the slides following)

- Non-linear function: a function that affects input in a non-linear way.

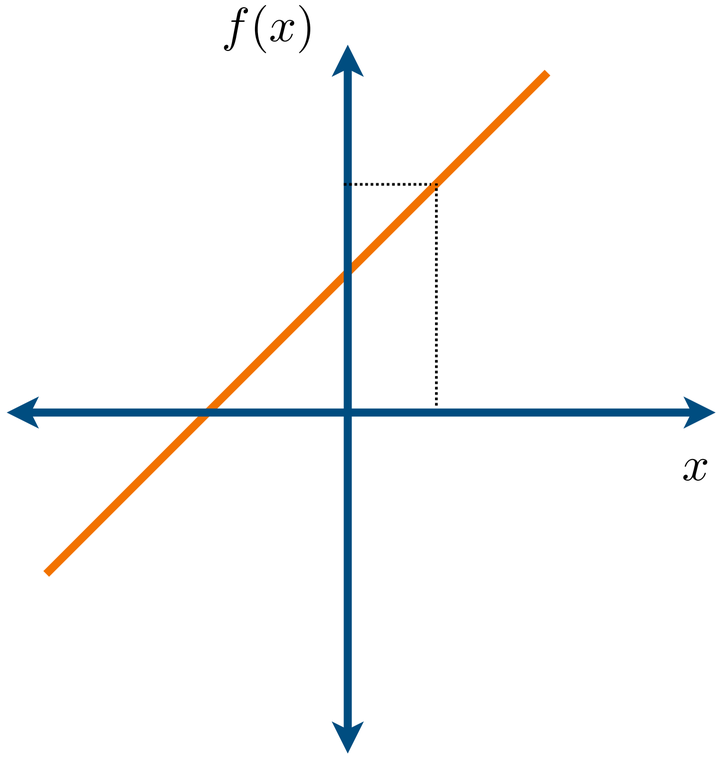


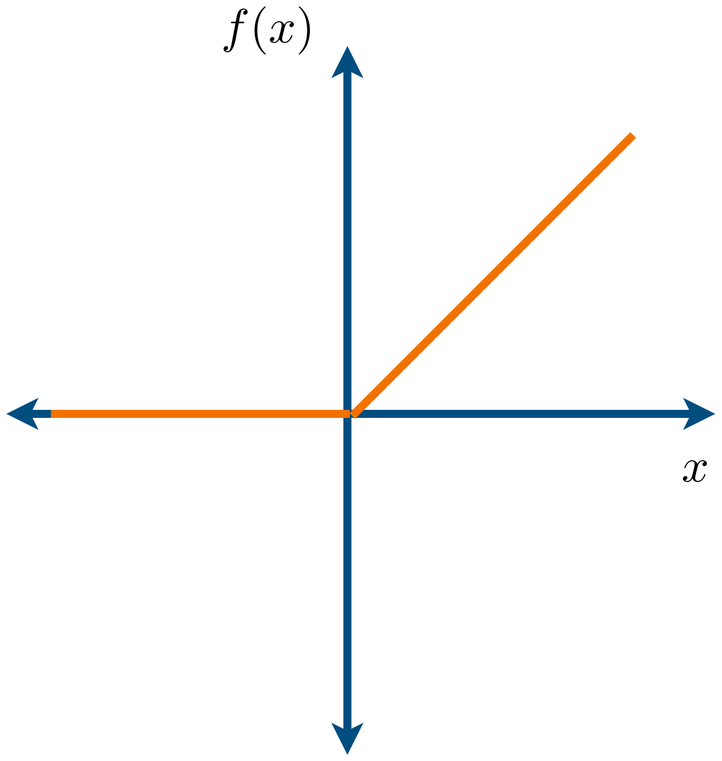


What is a non-linear function?

(More in the slides following)

- Non-linear function: a function that affects input in a non-linear way.

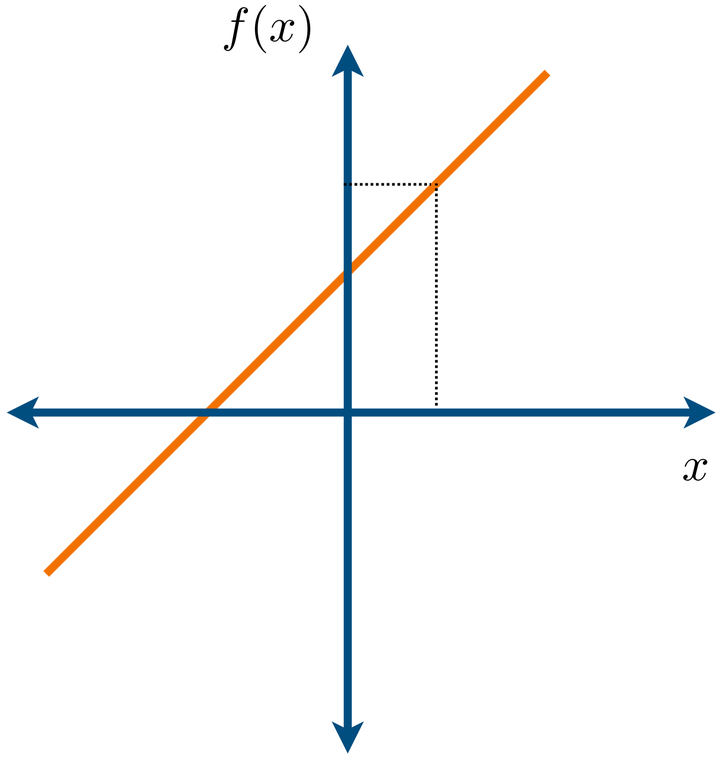


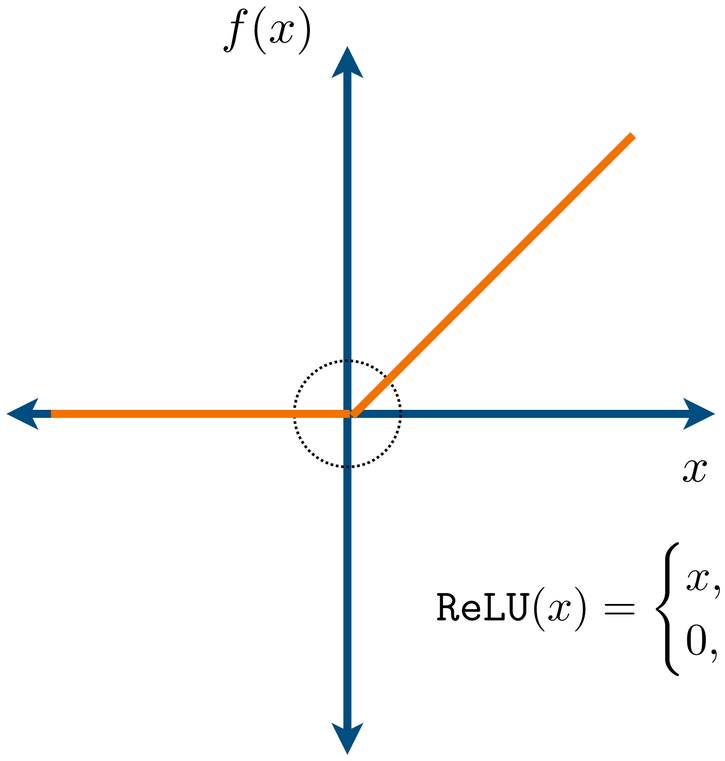


What is a non-linear function?

(More in the slides following)

- Non-linear function: a function that affects input in a non-linear way.

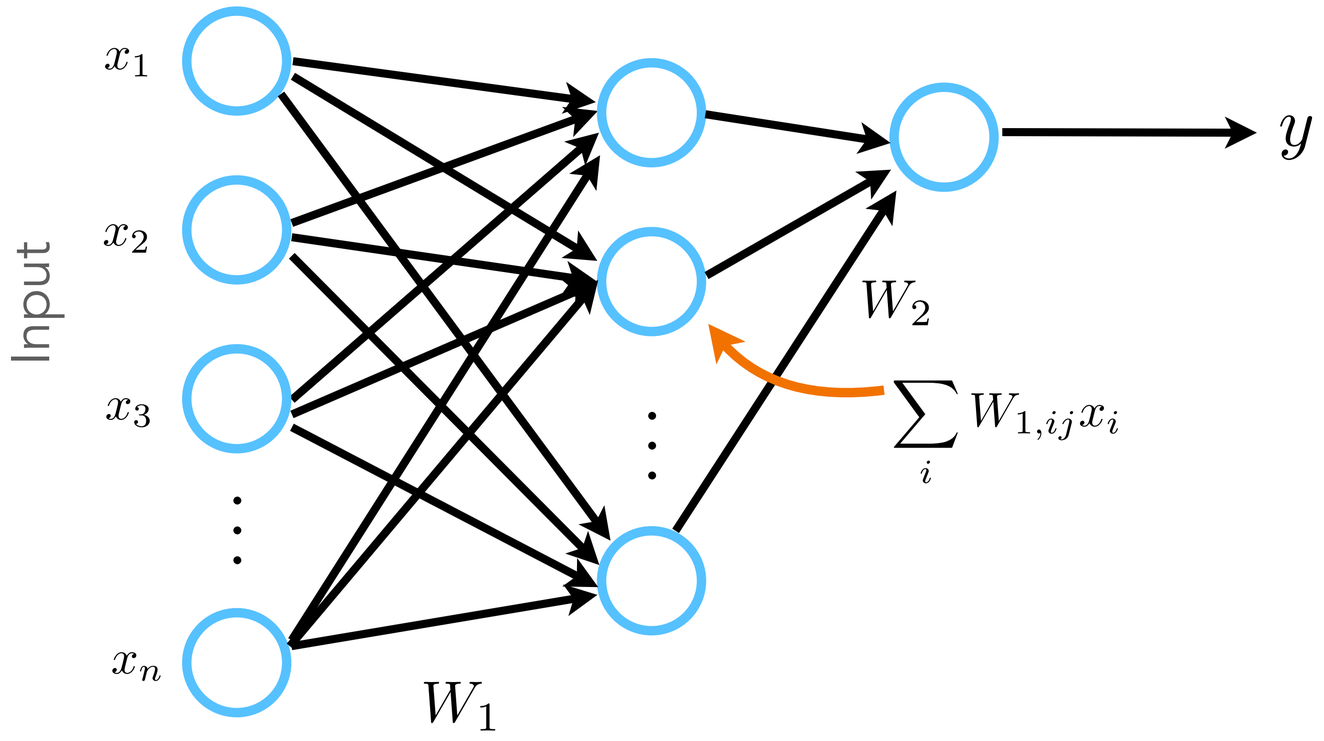




$$\texttt{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

But is this enough?

- Assume we got a way to train multilayer perceptrons



$$W_L \cdots W_2 \cdot W_1 x = W x$$

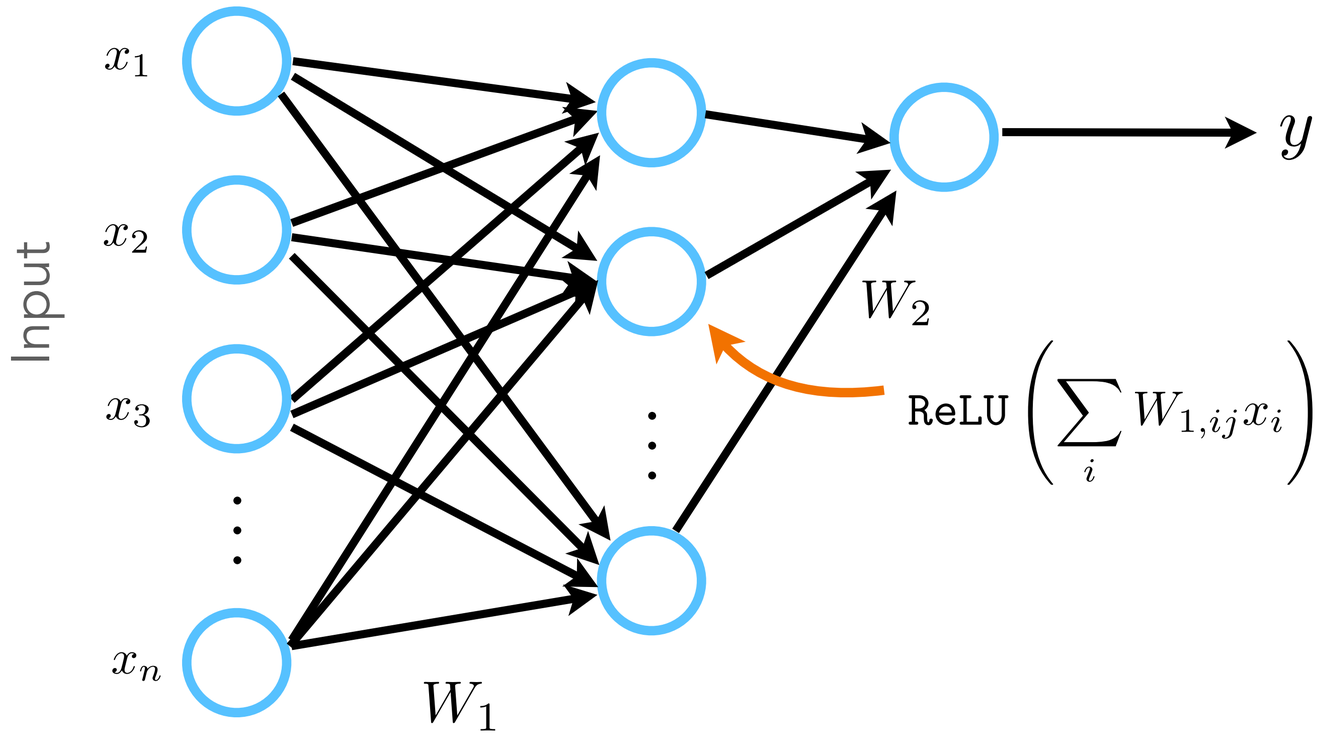
$$||$$

Equal to having one layer perceptron

(Alternatively, the expressivity power of this perceptron is limited by the linearity of operations)

But is this enough?

- Assume we got a way to train multilayer perceptrons



$$W_L \cdots W_2 \cdot W_1 x = W x$$

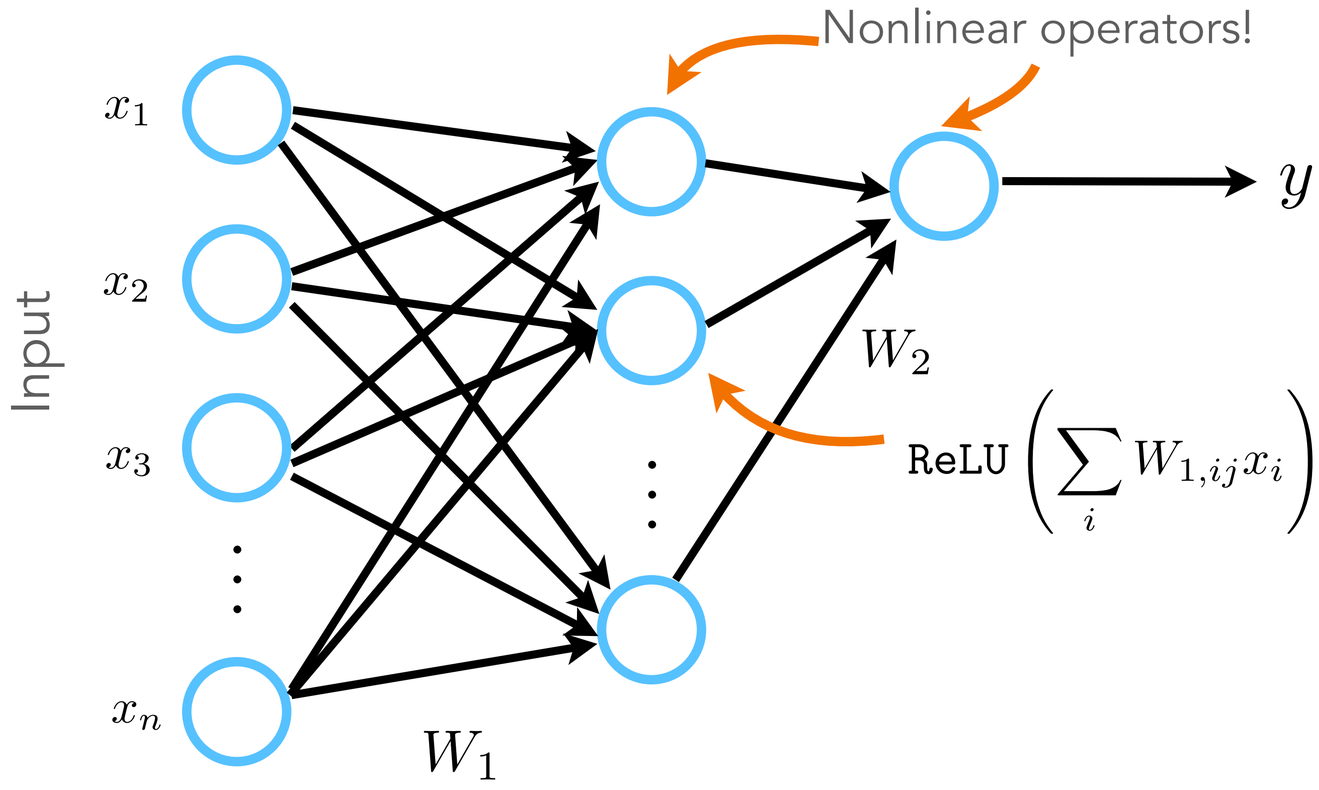
$$||$$

Equal to having one layer perceptron

(Alternatively, the expressivity power of this perceptron is limited by the linearity of operations)

But is this enough?

- Assume we got a way to train multilayer perceptrons



$$W_L \cdots W_2 \cdot W_1 x = Wx$$

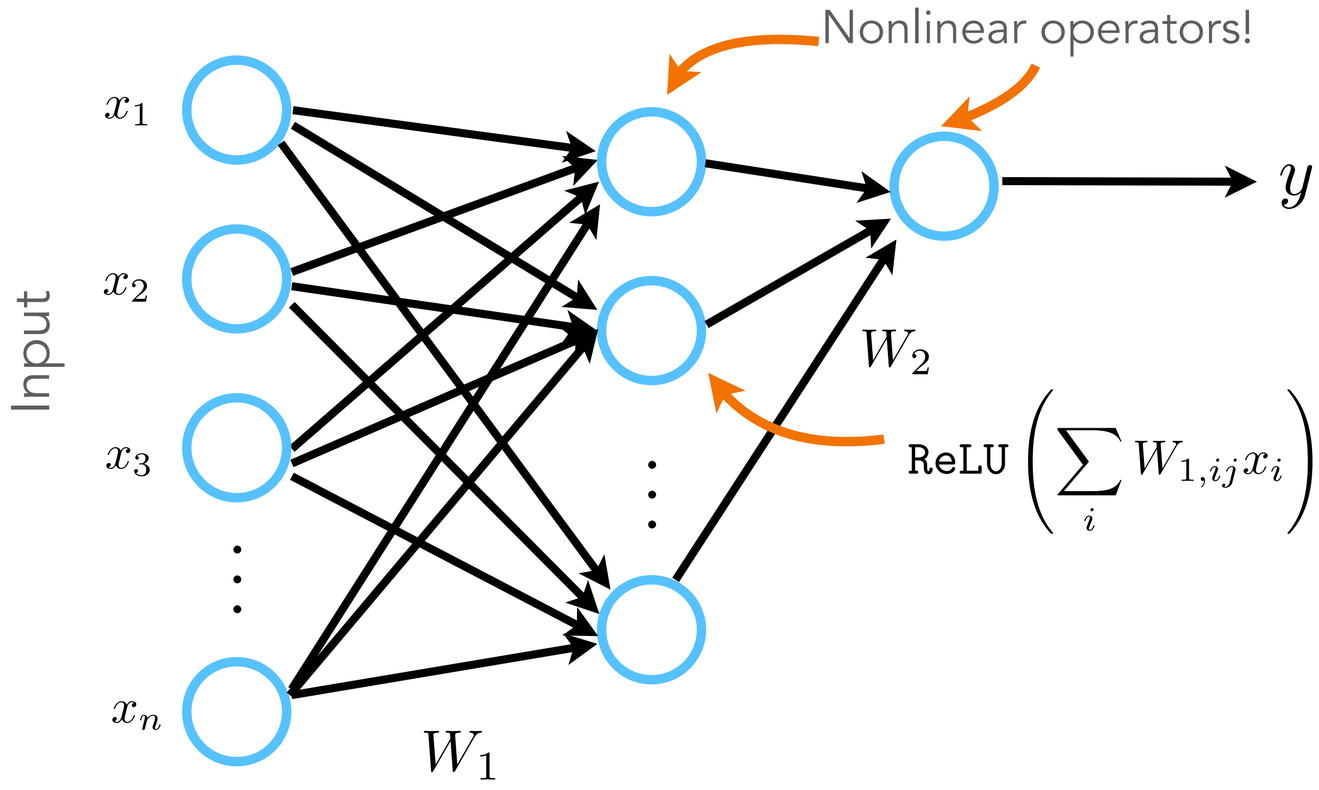
$$||$$

Equal to having one layer perceptron

(Alternatively, the expressivity power of this perceptron is limited by the linearity of operations)

But is this enough?

- Assume we got a way to train multilayer perceptrons



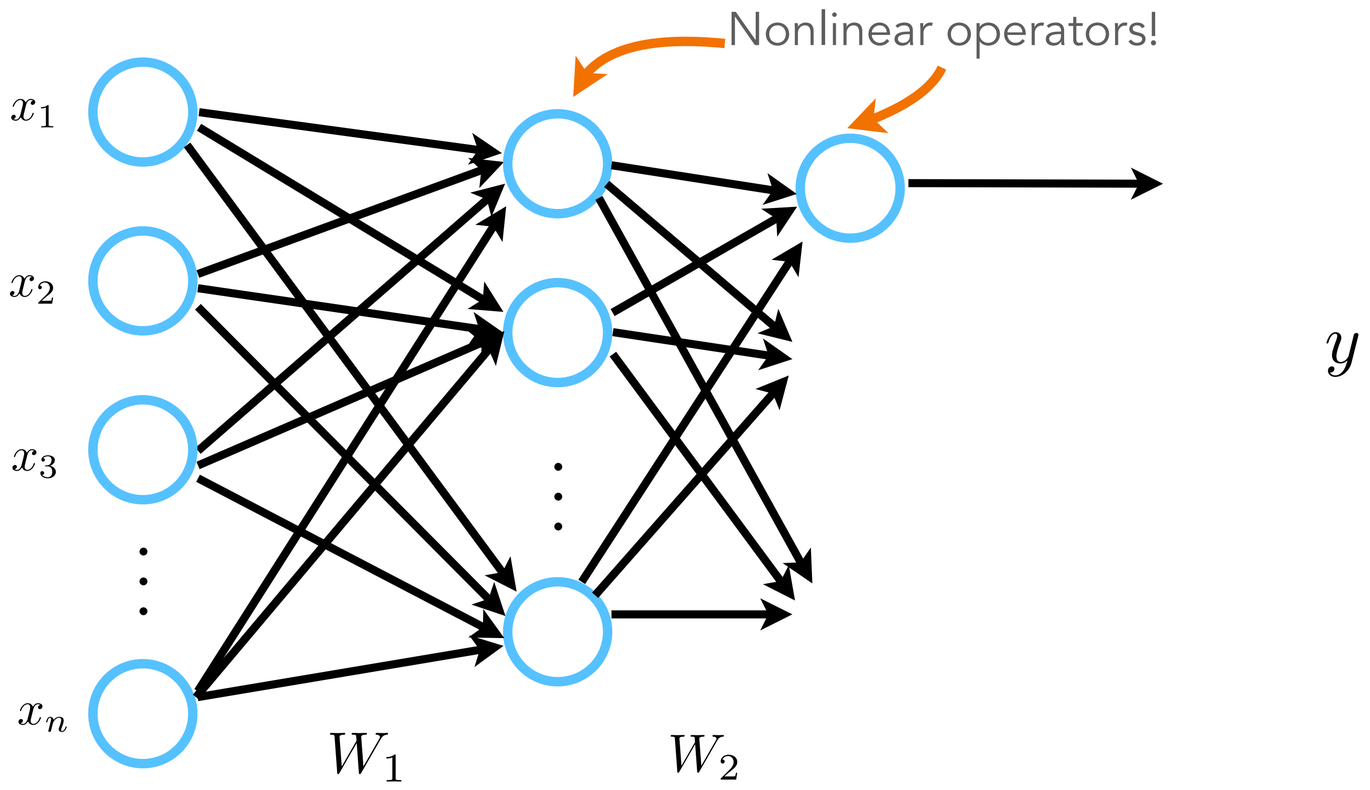
$$W_L \cdots W_2 \cdot W_1 x = W x$$

$$\not\equiv$$

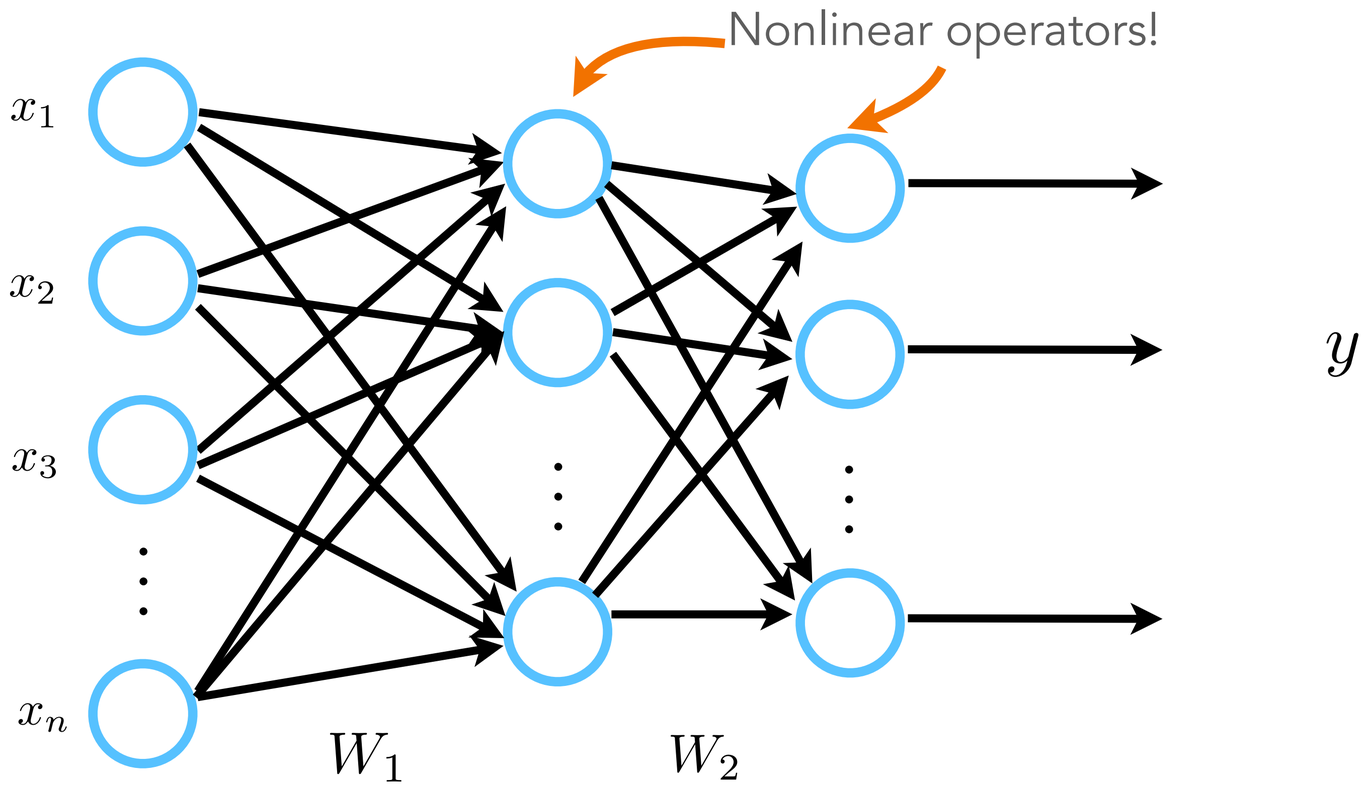
Equal to having one layer perceptron

(Alternatively, the expressivity power of this perceptron is limited by the linearity of operations)

Combining all these things together: MLP

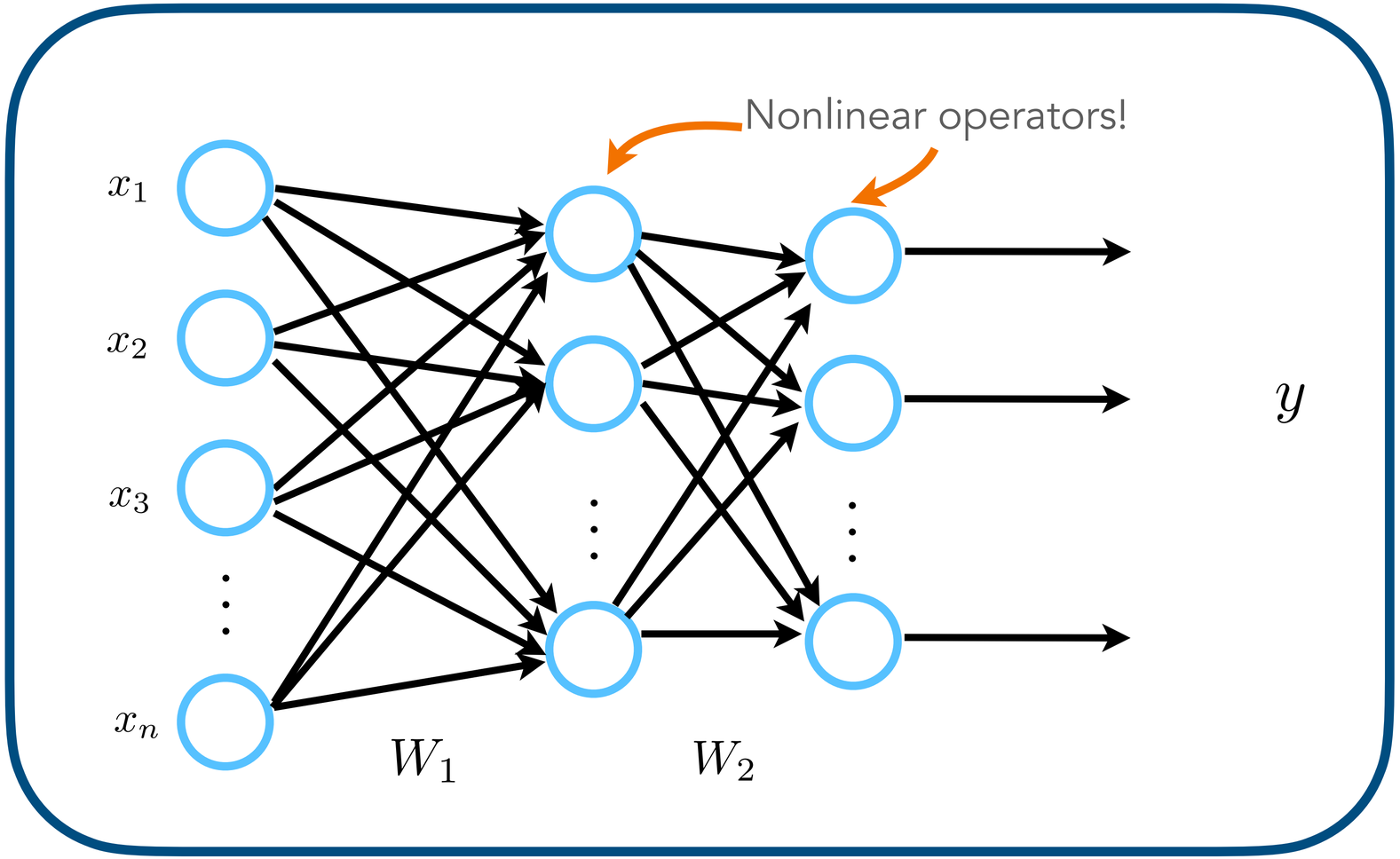


Combining all these things together: MLP

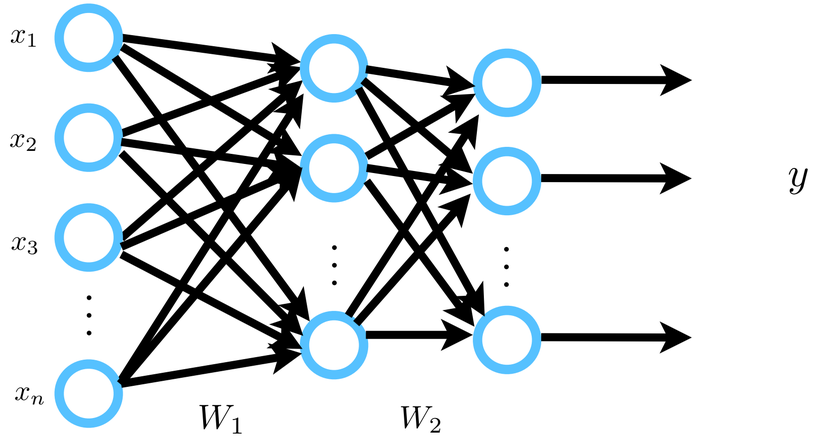


Combining all these things together: MLP

Feedforward/fully connected neural network

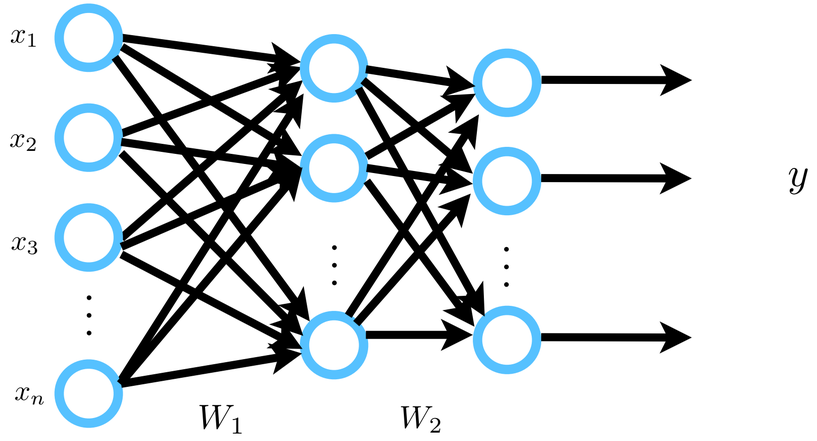


Some issues even with simple MLPs



Some issues even with simple MLPs

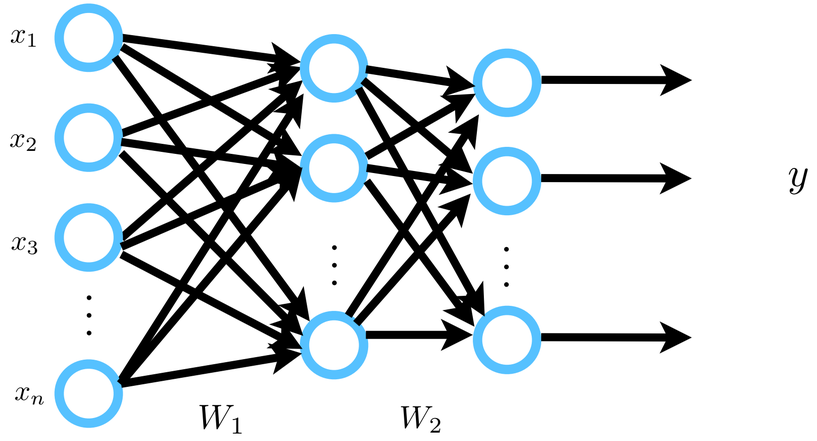
- Going deeper: lack of processing power



Some issues even with simple MLPs

- Going deeper: lack of processing power
- Algorithms for training suffer from many issues

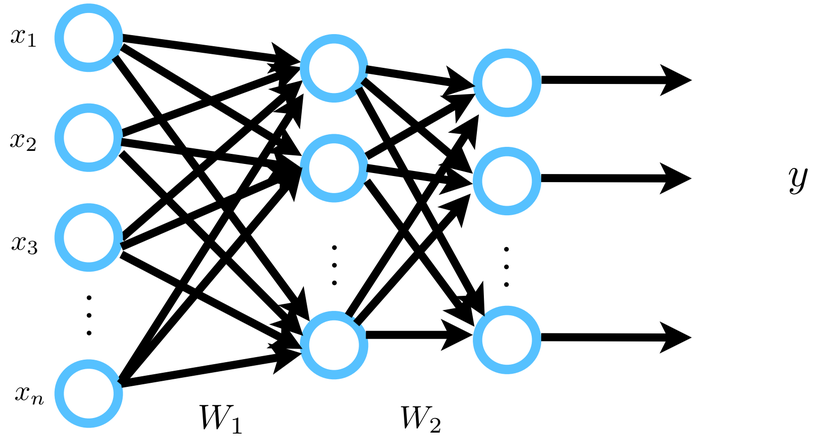
  E.g., exploding/vanishing quantities



Some issues even with simple MLPs

- Going deeper: lack of processing power
- Algorithms for training suffer from many issues

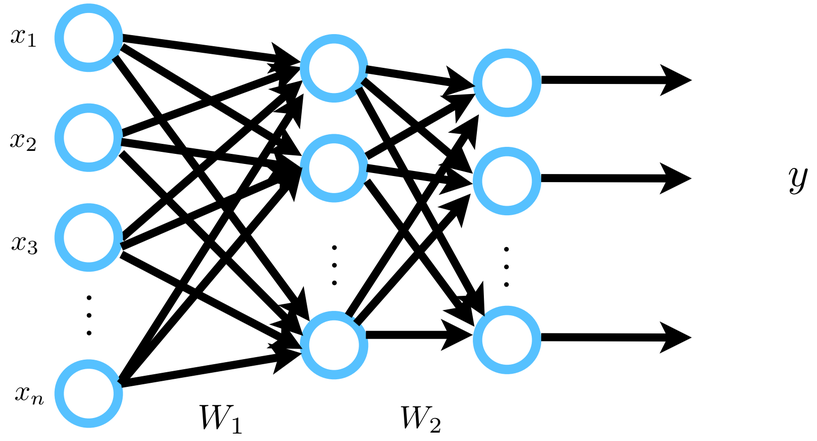
  E.g., exploding/vanishing quantities
- Data hungry: require a lot of input/output



Some issues even with simple MLPs

- Going deeper: lack of processing power
- Algorithms for training suffer from many issues

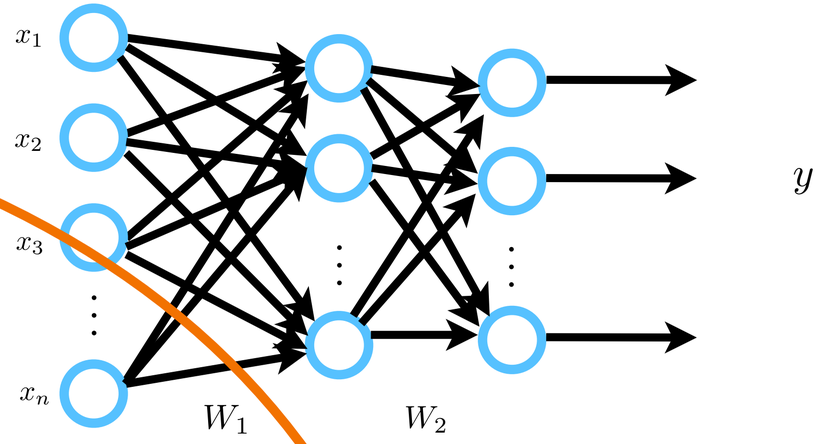
  E.g., exploding/vanishing quantities
- Data hungry: require a lot of input/output
- Overfitting



Some issues even with simple MLPs

- Going deeper: lack of processing power
- Algorithms for training suffer from many issues

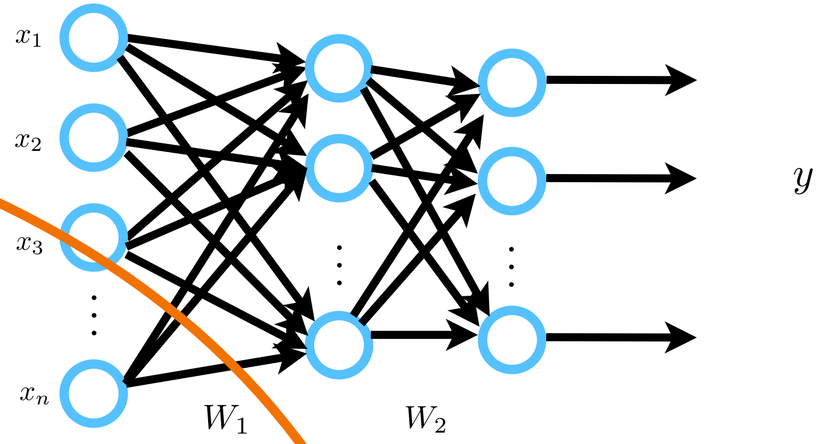
  E.g., exploding/vanishing quantities
- Data hungry: require a lot of input/output
- Overfitting



Some issues even with simple MLPs

- Going deeper: lack of processing power
- Algorithms for training suffer from many issues

  E.g., exploding/vanishing quantities
- Data hungry: require a lot of input/output
- Overfitting

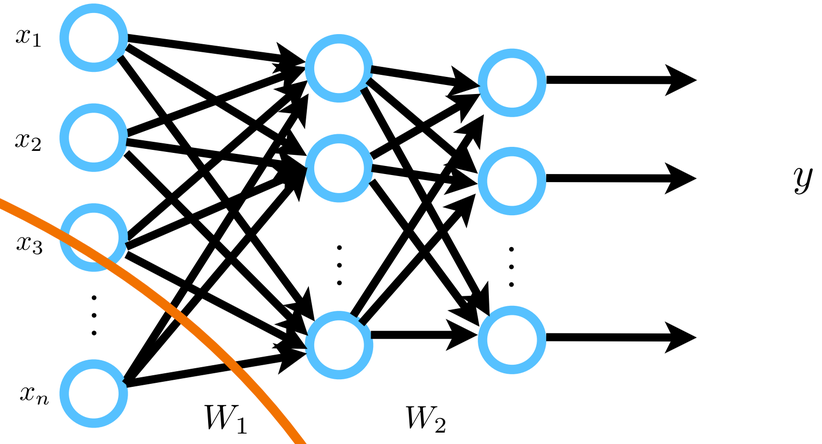


E.g., ImageNet dataset: 14+ millions images, 21K+ categories

Some issues even with simple MLPs

- Going deeper: lack of processing power
- Algorithms for training suffer from many issues

  E.g., exploding/vanishing quantities
- Data hungry: require a lot of input/output
- Overfitting



ML Community
(academics + industry labs)

E.g., ImageNet dataset: 14+ millions images, 21K+ categories

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

$$\widehat{y}_i = f\left(W, x_i\right)$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

Input

$$\widehat{y}_i = f\left(W, x_i\right)$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

Model

Input

$$\widehat{y}_i = f\left(W, x_i\right)$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

Model

Input

$$\widehat{y}_i = f\left(W, x_i\right)$$

Output

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$\widehat{y}_i = f\left(W, x_i\right)$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$\widehat{y}_i = f\left(W, x_i\right)$$

Variables

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$\widehat{y}_i = f\left(W, x_i\right)$$

Variables

$$W = \{W_1, W_2, \ldots, W_L\}$$

$$W_i \in \mathbb{R}^{p_{\text{in}} \times p_{\text{out}}}$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

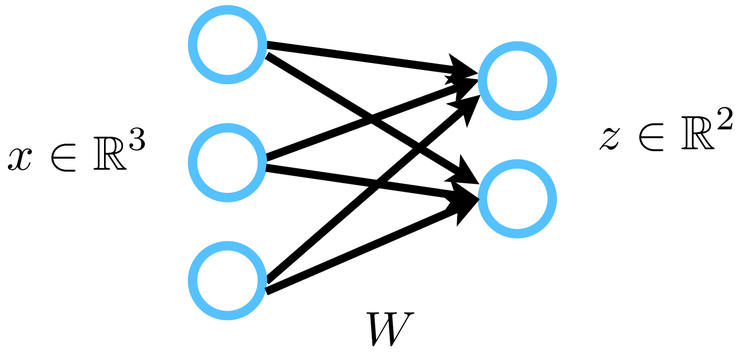
$$\widehat{y}_i = f\left(W, x_i\right)$$

Variables

$$W = \{W_1, W_2, \ldots, W_L\}$$

$$W_i \in \mathbb{R}^{p_{\text{in}} \times p_{\text{out}}}$$

Example:



A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

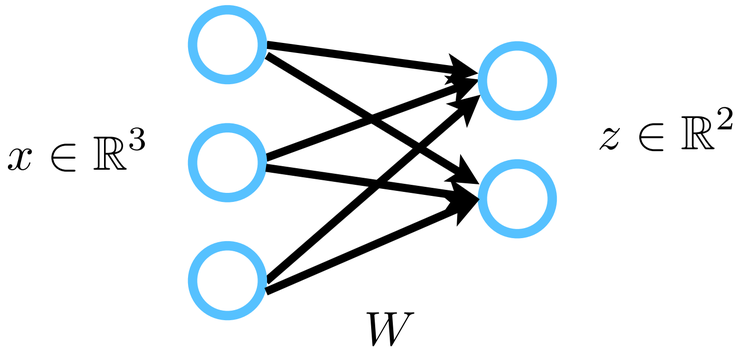
$$\widehat{y}_i = f\left(W, x_i\right)$$

Variables

$$W = \{W_1, W_2, \ldots, W_L\}$$

$$W_i \in \mathbb{R}^{p_{\text{in}} \times p_{\text{out}}}$$

Example:



$$z = Wx = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

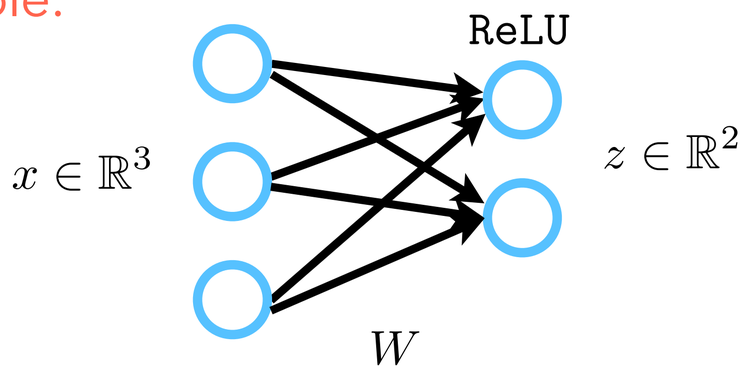
$$\widehat{y}_i = f\left(W, x_i\right)$$

Variables

$$W = \{W_1, W_2, \ldots, W_L\}$$

$$W_i \in \mathbb{R}^{p_{\text{in}} \times p_{\text{out}}}$$

Example:



$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \texttt{ReLU}(W_{1,:}x) \\ \texttt{ReLU}(W_{2,:}x) \end{bmatrix}$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

  The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$\widehat{y}_i = f\left(W, x_i\right)$$

Estimated output

The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$\widehat{y}_i = f\left(W, x_i\right)$$

The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

Estimated output

How does it compare with real labels/output?

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$\widehat{y}_i = f\left(W, x_i\right)$$

The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

Estimated output

How does it compare with real labels/output?

$$\ell(y_i, \widehat{y}_i)$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$\widehat{y}_i = f\left(W, x_i\right)$$

Estimated output

The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

How does it compare with real labels/output?

$$\ell(y_i, \widehat{y}_i)$$

Examples:

$$\ell(y_i, \widehat{y}_i) = \frac{1}{2}\|y_i - \widehat{y}_i\|_2^2$$

$$\ell(y_i, \widehat{y}_i) = \texttt{cross-entropy}(y_i, \widehat{y}_i)$$

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$\widehat{y}_i = f\left(W, x_i\right)$$

Estimated output

The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

How does it compare with real labels/output?

$$\ell(y_i, \widehat{y}_i)$$

Examples:

$$\ell(y_i, \widehat{y}_i) = \frac{1}{2}\|y_i - \widehat{y}_i\|_2^2$$

$$\ell(y_i, \widehat{y}_i) = \texttt{cross-entropy}(y_i, \widehat{y}_i)$$

Goal: make loss as small as possible over the whole dataset ($\{x_i, y_i\}_{i=1}^n$**)**

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

  The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

  The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

  Neural networks are trained with local search optimization algorithms such as stochastic gradient descent, Adam, Adagrad, etc.

A bit of math in the mix: Neural Networks

- Verbose description: A neural network is a **model/black box** that takes input and and provides some answer.

  The model is **parameterized by a number of variables** and includes various operations (matrix/matrix multiplications, convolutions, etc) as well as non-linear transformations.

$$W^\star \in \arg\min_W \sum_{\{x_i, y_i\}} \ell(y_i, \widehat{y}_i)$$

$$\arg\min_W \sum_{\{x_i, y_i\}} \ell(y_i, f(W, x_i))$$

The output of the model is compared to some ground truth in order to minimize a loss function (user-defined).

Neural networks are trained with local search optimization algorithms such as stochastic gradient descent, Adam, Adagrad, etc.

What differentiates one NN from another

- Operations involved: Feedforward layers only
  Convolution layers
  Pooling operations
  Regularization techniques (dropout, batch normalization)
  Residual steps
  Recurrent steps

Modules

- Non-linear functions used: ReLU, leaky ReLUs, tanh, sigmoid, etc.
- Deep vs. shallow, wide vs. narrow: Although shallow wide NNs work well in theory, deep nets are more efficient, and generalize better.
- Objective functions: euclidean norm (regression), cross entropy (classification) only input data (autoencoders), min-max (GANs), etc.

(or type of learning)

Modules

Overview

- Introduction to neural networks
- **Feedforward / Fully-connected neural networks or MLPs**
- Convolutional neural networks (CNNs)
- Recurrent neural networks (RNNs)
- Optimization in neural network training

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)

Neural networks represented as boxes

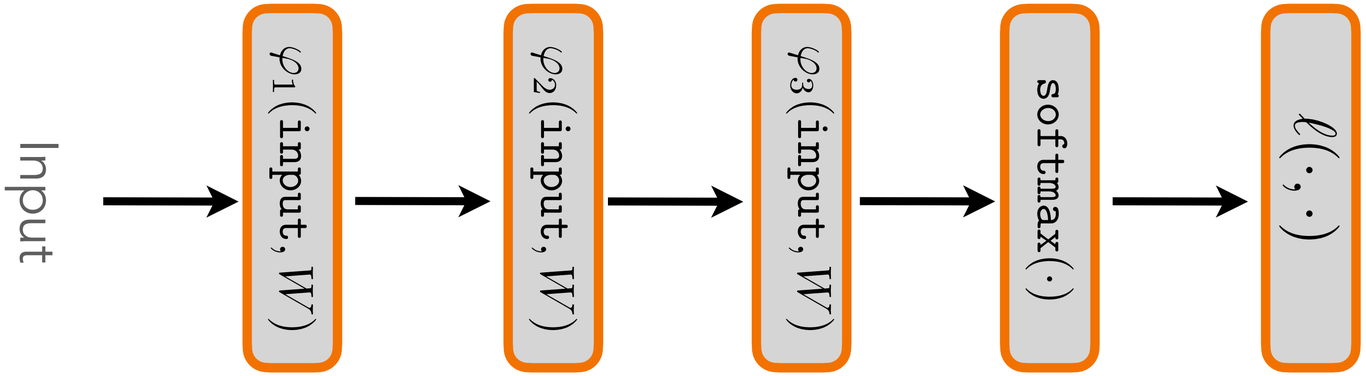
- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

  Softmax before output, loss function $\ell(\cdot, \cdot)$

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

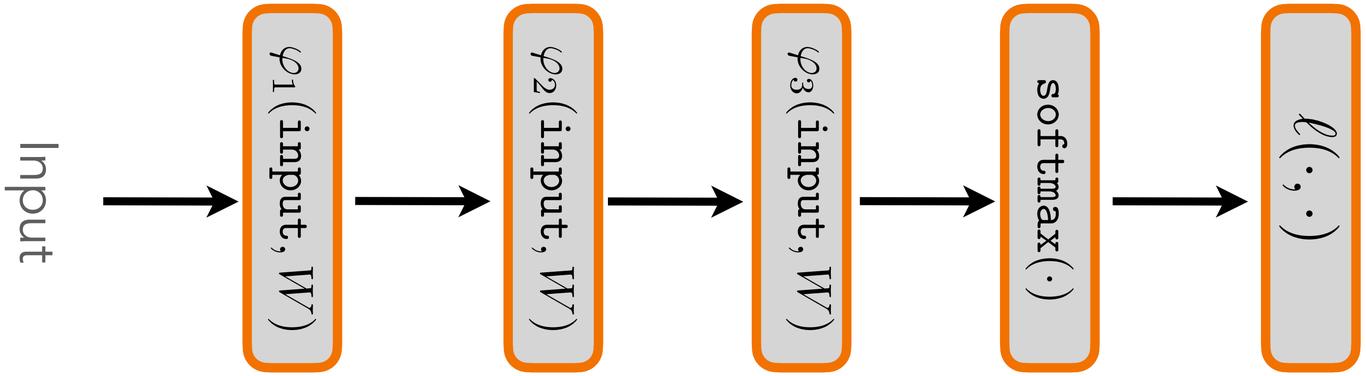
  Softmax before output, loss function $\ell(\cdot, \cdot)$



Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

  Softmax before output, loss function $\ell(\cdot,\cdot)$



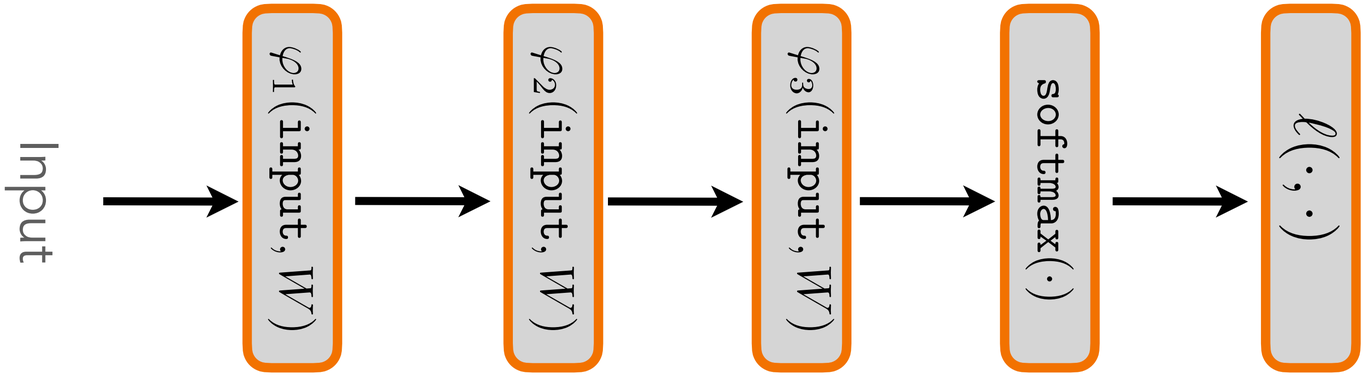
$$\ell\left(\texttt{softmax}\left(\varphi_3(\varphi_2(\varphi_1(x_i, W), W), W\right), \quad y_i\right)$$

Softmax: function that normalizes and outputs a vector of probabilities on the possible outcomes

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

  Softmax before output, loss function $\ell(\cdot,\cdot)$



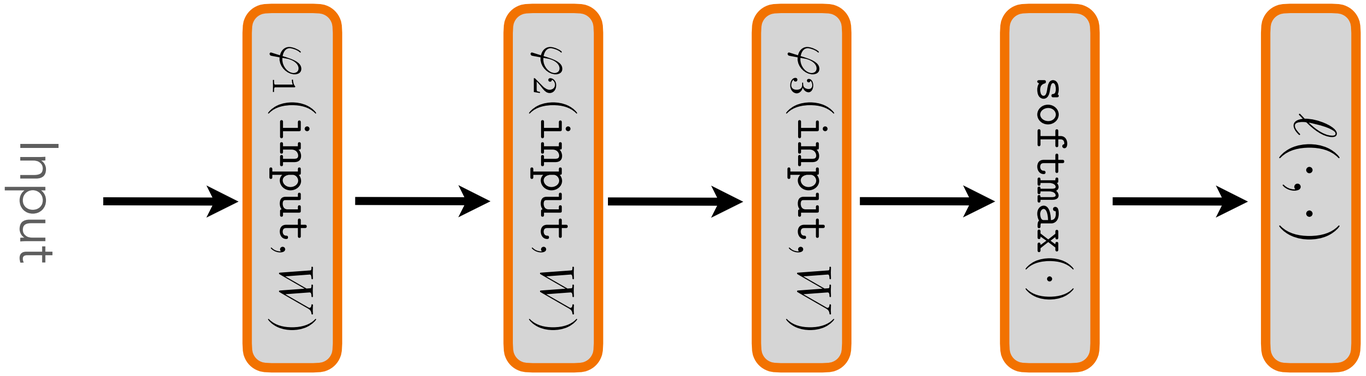
$$\ell\left(\texttt{softmax}\left(\varphi_3(\varphi_2(\boxed{\varphi_1(x_i, W)}, W), W\right), \quad y_i\right)$$

Softmax: function that normalizes and outputs a vector of probabilities on the possible outcomes

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

  Softmax before output, loss function $\ell(\cdot,\cdot)$



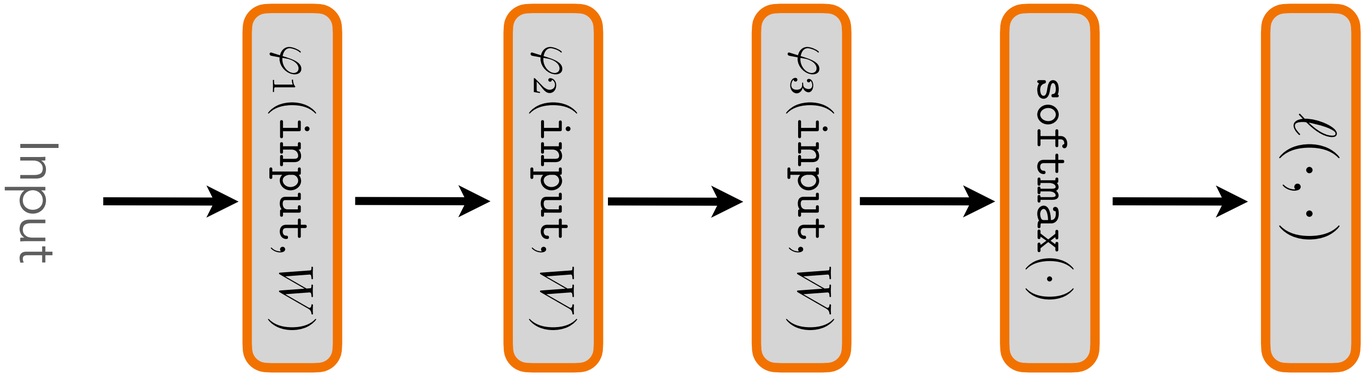
$$\ell\left(\texttt{softmax}\left(\varphi_3\left(\boxed{\varphi_2\left(\varphi_1(x_i, W), W\right)}, W\right), \quad y_i\right)\right.$$

Softmax: function that normalizes and outputs a vector of probabilities on the possible outcomes

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

  Softmax before output, loss function $\ell(\cdot,\cdot)$



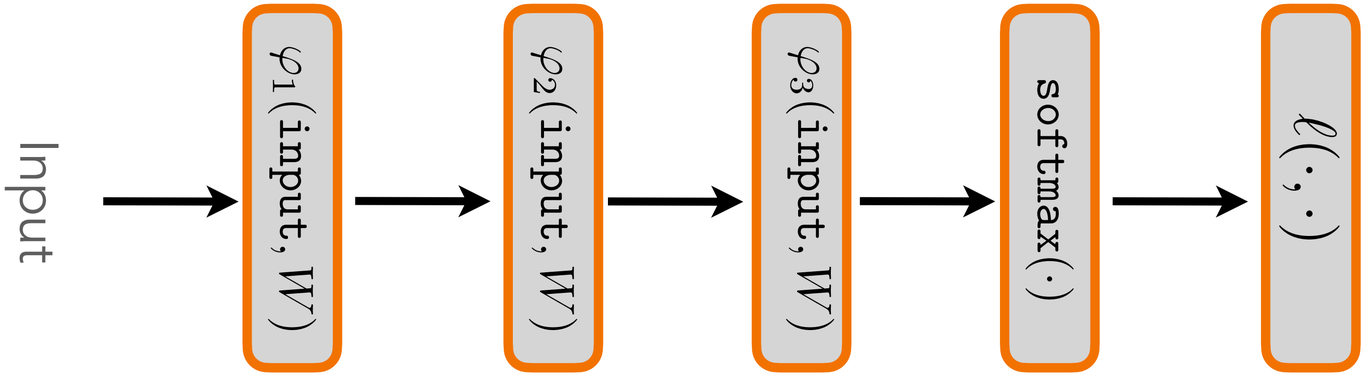
$$\ell\left(\texttt{softmax}\left(\boxed{\varphi_3(\varphi_2(\varphi_1(x_i, W), W), W)}\right), \quad y_i\right)$$

Softmax: function that normalizes and outputs a vector of probabilities on the possible outcomes

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

Softmax before output, loss function $\ell(\cdot,\cdot)$



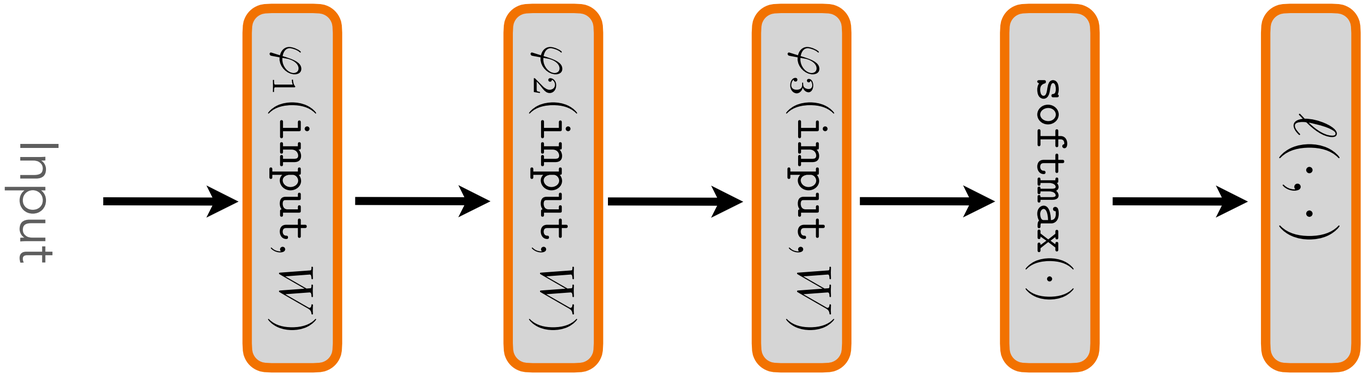
$$\ell\left(\boxed{\texttt{softmax}\left(\varphi_3(\varphi_2(\varphi_1(x_i, W), W), W)\right)}, \quad y_i\right)$$

Softmax: function that normalizes and outputs a vector of probabilities on the possible outcomes

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

  Softmax before output, loss function $\ell(\cdot, \cdot)$

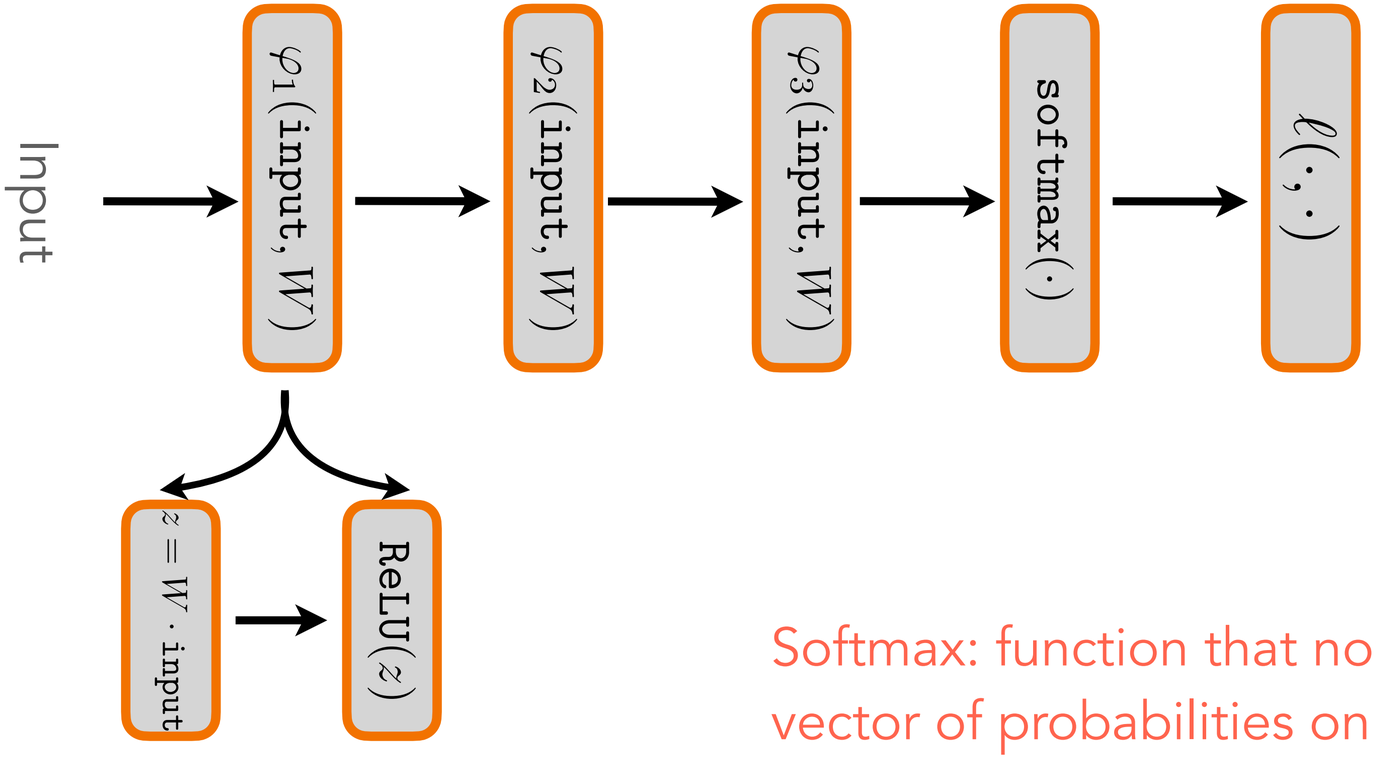


Softmax: function that normalizes and outputs a vector of probabilities on the possible outcomes

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

  Softmax before output, loss function $\ell(\cdot, \cdot)$

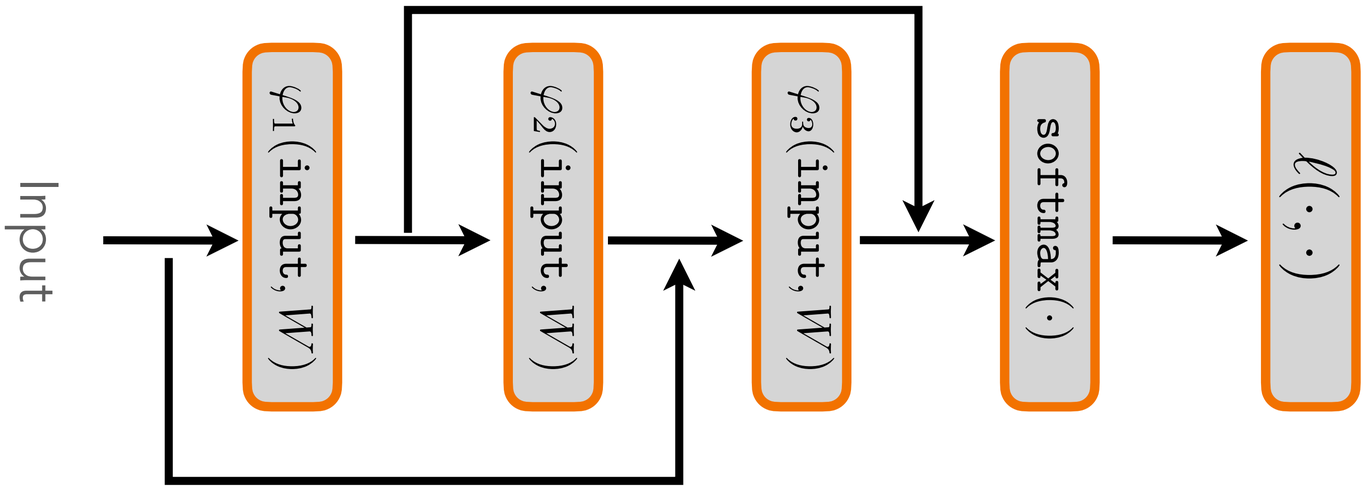


Softmax: function that normalizes and outputs a vector of probabilities on the possible outcomes

Neural networks represented as boxes

- Most used representation of neural networks (but math will get us the details)
- Example: feedforward neural network, with 3 hidden layers, and ReLUs

  Softmax before output, loss function $\ell(\cdot, \cdot)$

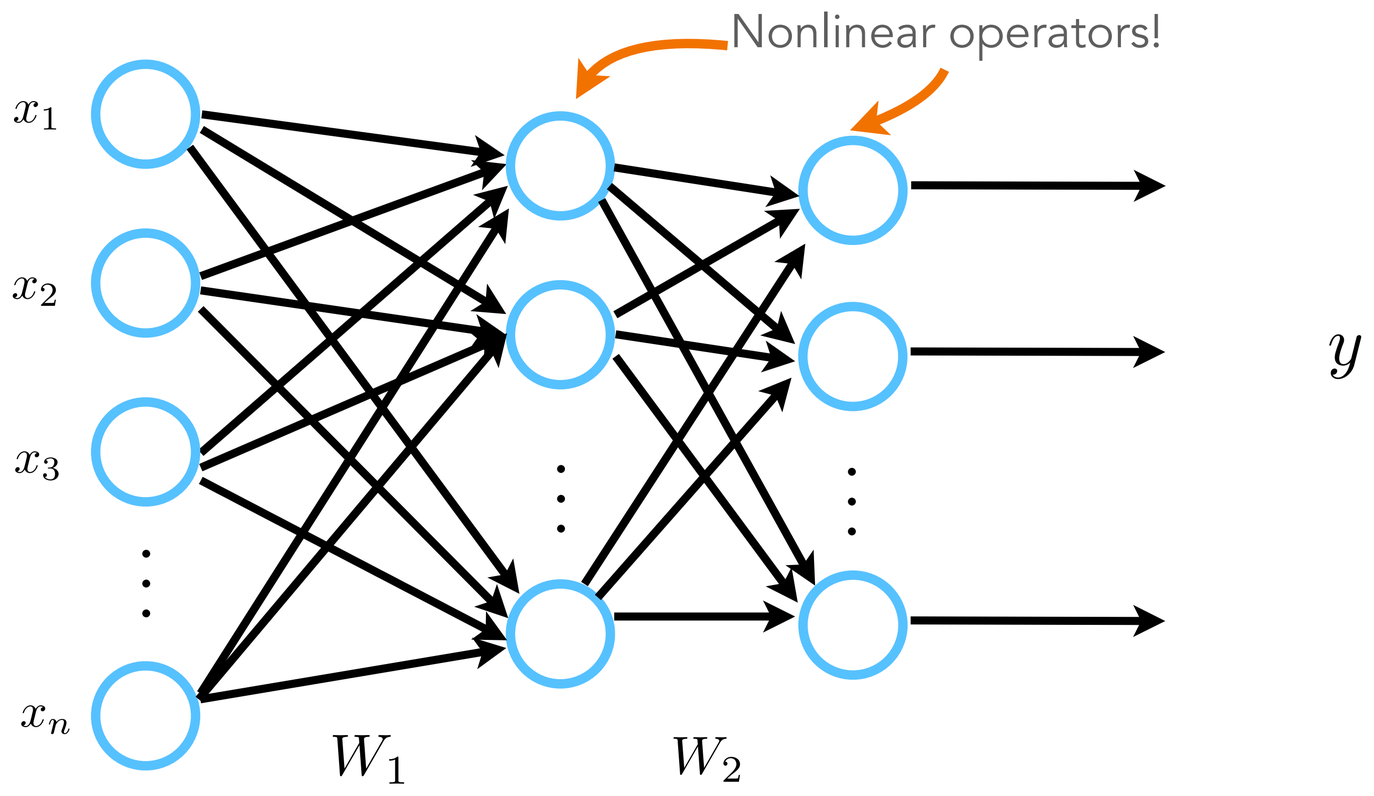


Softmax: function that normalizes and outputs a vector of probabilities on the possible outcomes

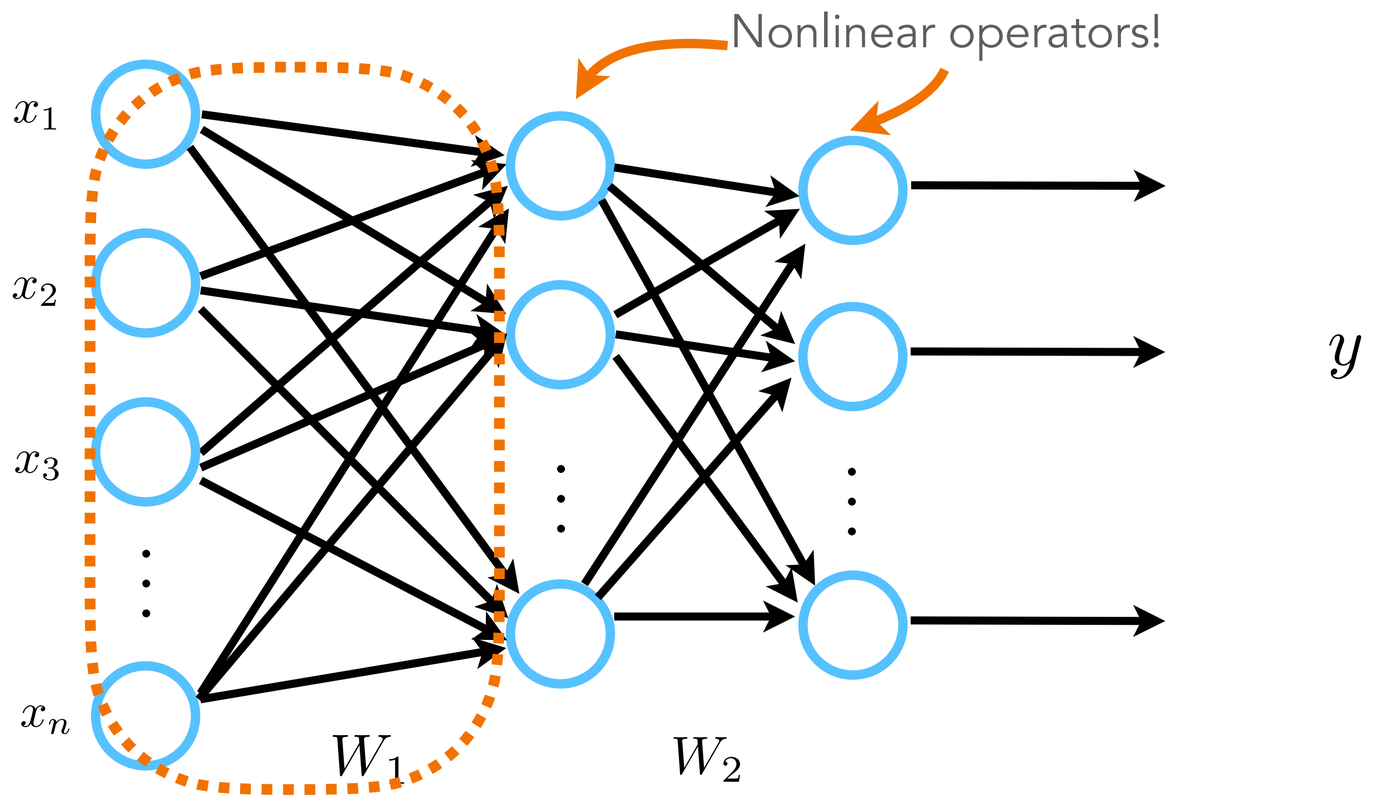
Neural networks represented as boxes

```
channels, width, height = self.dims
self.input_size = channels * width * height

# Define layers
self.layers = nn.Sequential(
    # Layer 1
    nn.Linear(self.input_size, hidden_size),
    nn.ReLU(),
    nn.Dropout(0.2),

    # Layer 2
    nn.Linear(hidden_size, hidden_size),
    nn.ReLU(),
    nn.Dropout(0.2),

    # Layer 3 (output layer)
    nn.Linear(hidden_size, self.num_classes)
)
```

Fully connected neural networks or MLPs

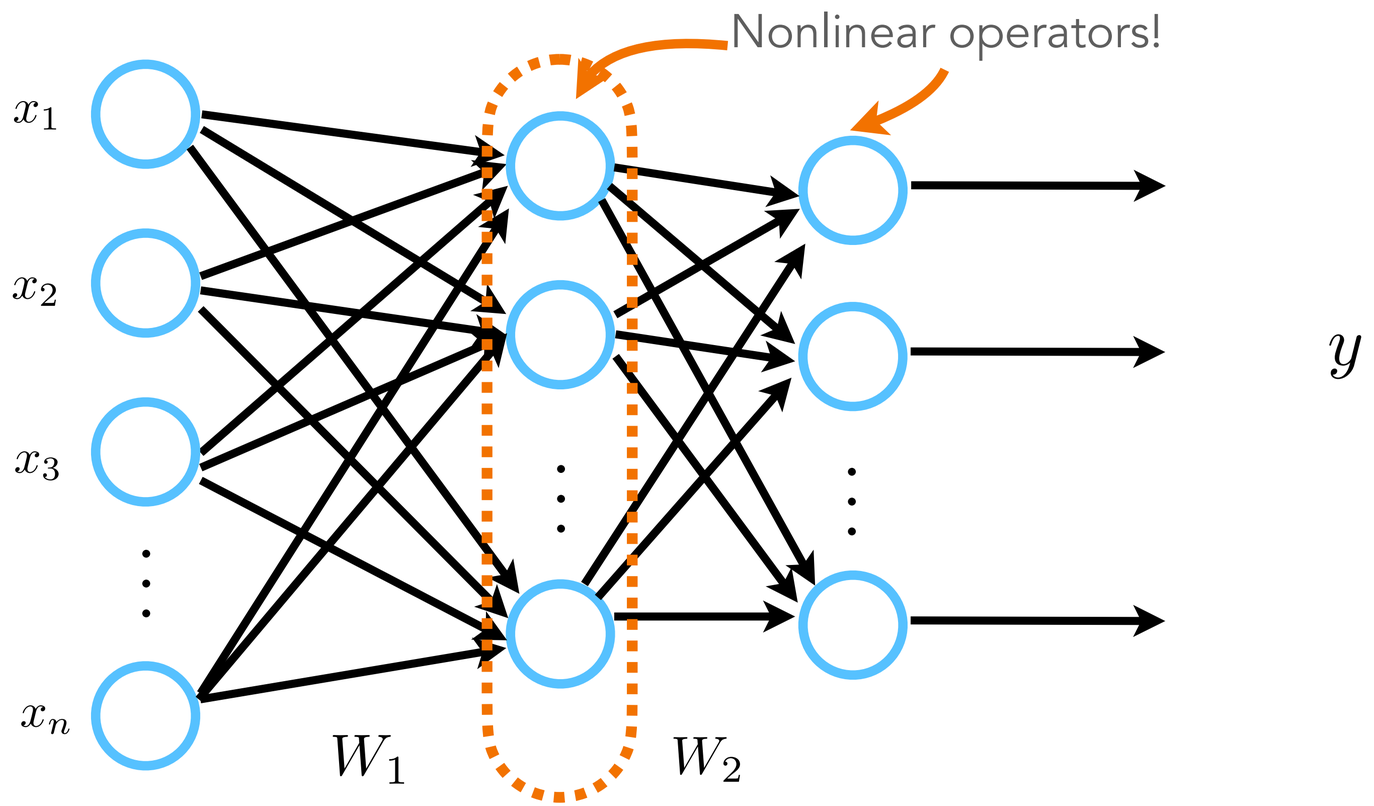


Fully connected neural networks or MLPs



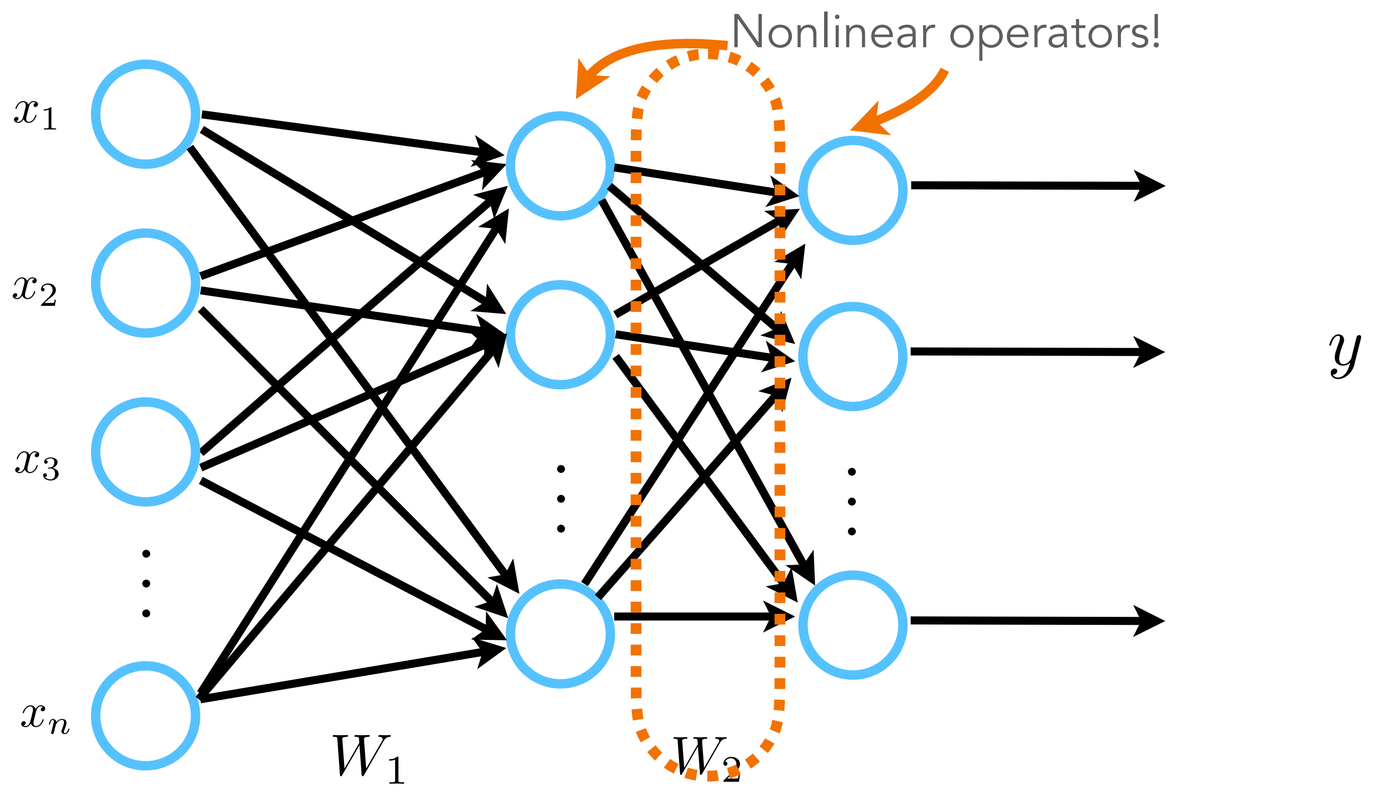
1. $z_1 = W_1 \cdot x$

Fully connected neural networks or MLPs



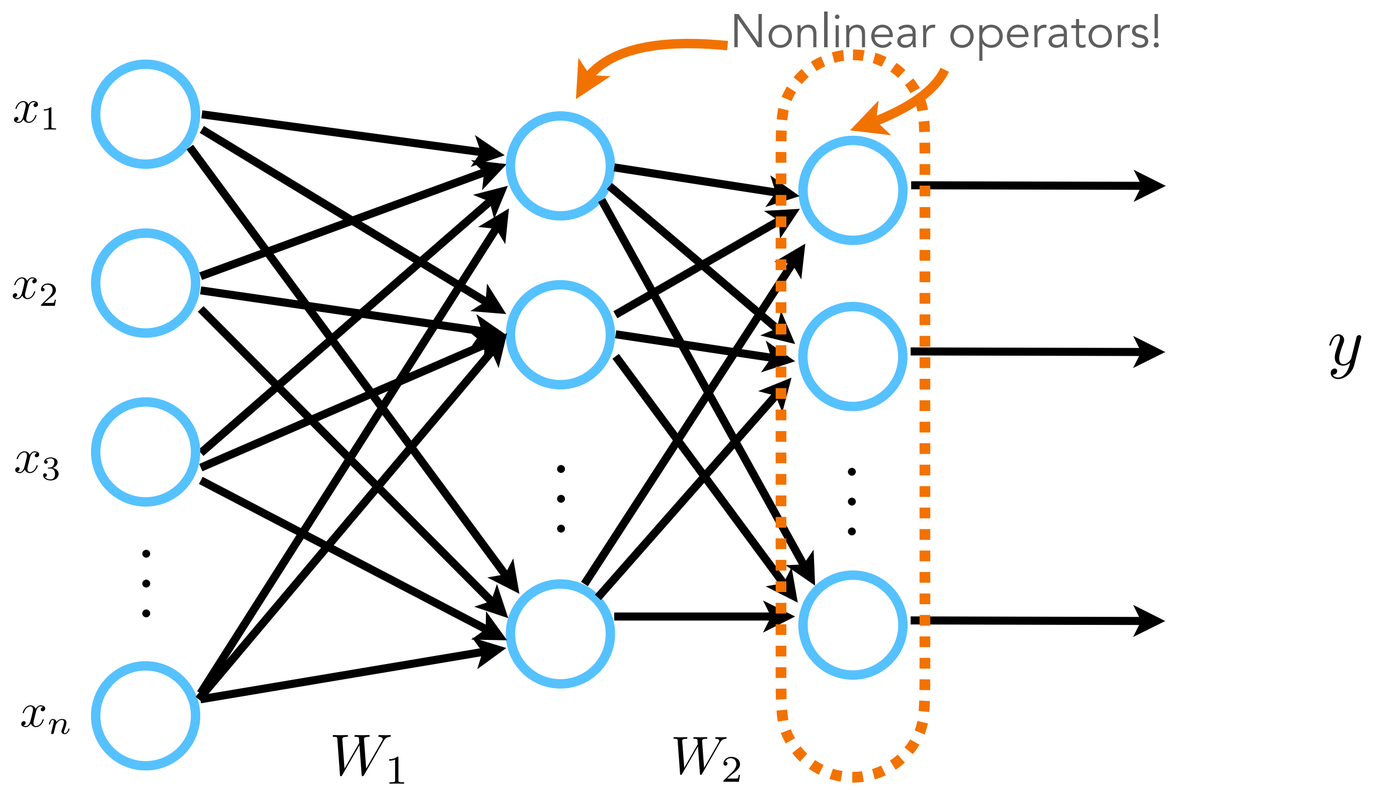
1. $z_1 = W_1 \cdot x$
2. $\sigma(z_1) = \sigma(W_1 \cdot x)$

Fully connected neural networks or MLPs



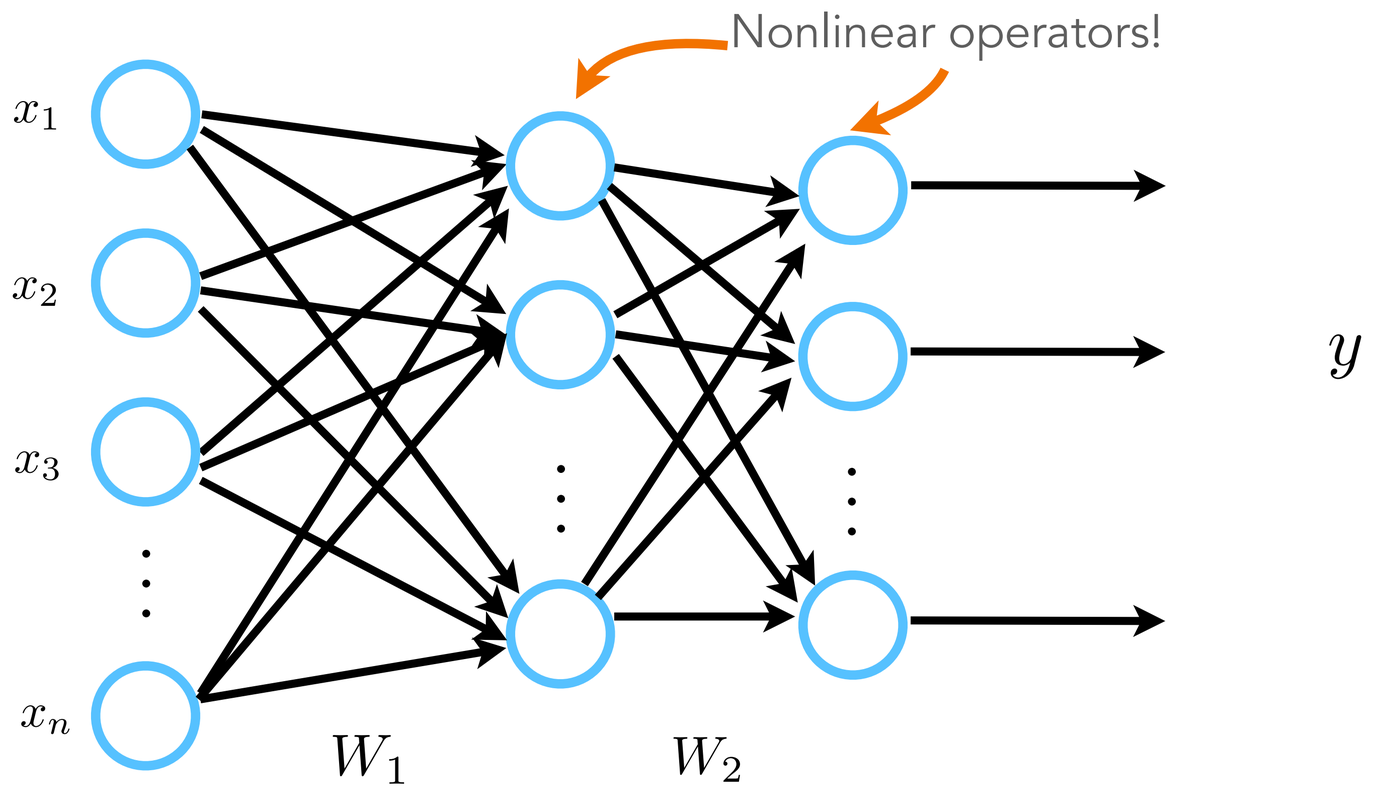
1. $z_1 = W_1 \cdot x$
2. $\sigma(z_1) = \sigma(W_1 \cdot x)$
3. $z_2 = W_2 \cdot \sigma(z_1) = W_2 \cdot \sigma(W_1 \cdot x)$

Fully connected neural networks or MLPs



1. $z_1 = W_1 \cdot x$
2. $\sigma(z_1) = \sigma(W_1 \cdot x)$
3. $z_2 = W_2 \cdot \sigma(z_1) = W_2 \cdot \sigma(W_1 \cdot x)$
4. $\texttt{softmax}(z_2) = \texttt{softmax}(W_2 \cdot \sigma(W_1 \cdot x))$

Fully connected neural networks or MLPs



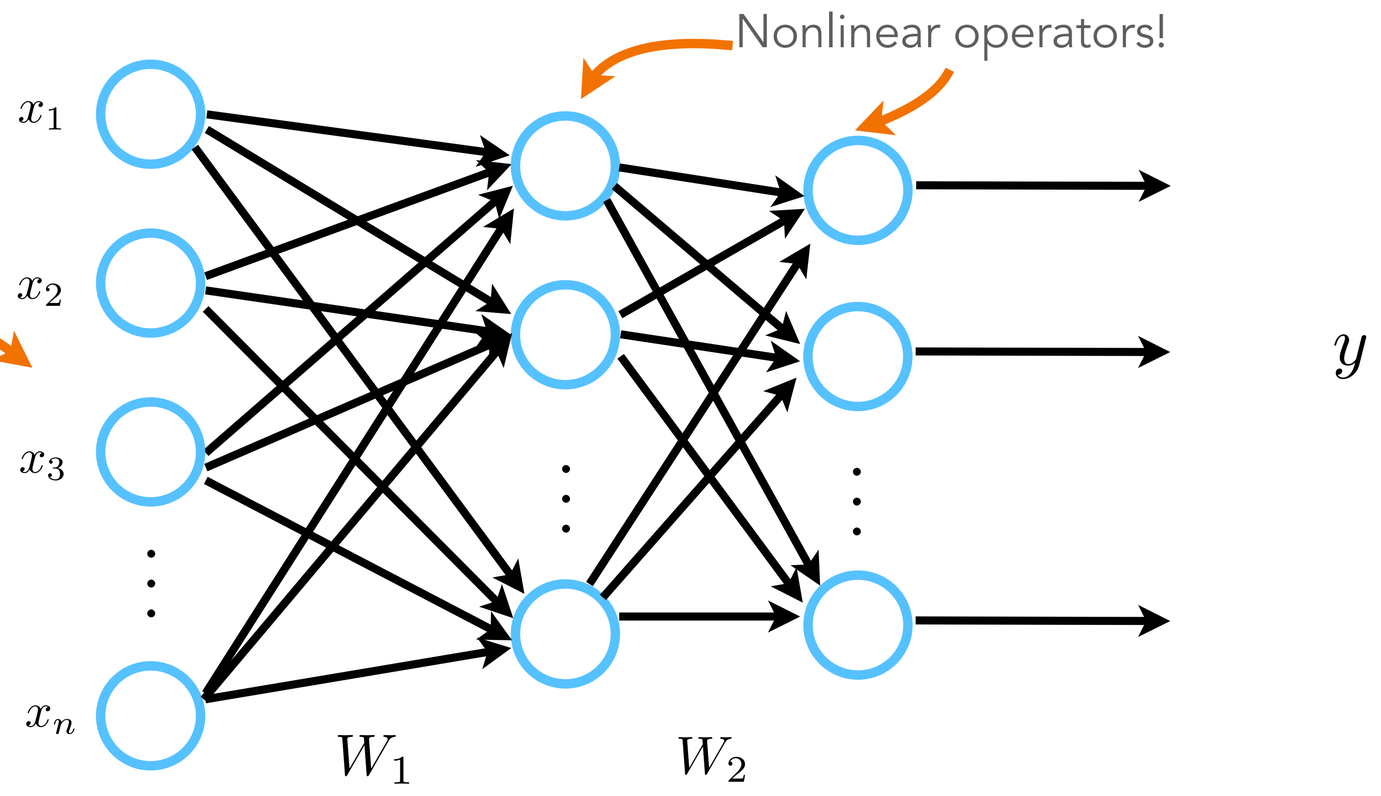
1. $z_1 = W_1 \cdot x$
2. $\sigma(z_1) = \sigma(W_1 \cdot x)$
3. $z_2 = W_2 \cdot \sigma(z_1) = W_2 \cdot \sigma(W_1 \cdot x)$
4. $\texttt{softmax}(z_2) = \texttt{softmax}(W_2 \cdot \sigma(W_1 \cdot x))$

- Nothing more complicated
- Boils down to what modules will be used
- Famous architecture due to simplicity and theoretical guarantees

Fully connected neural networks or MLPs

This sequence of operations is also known as the **forward pass on the neural network**

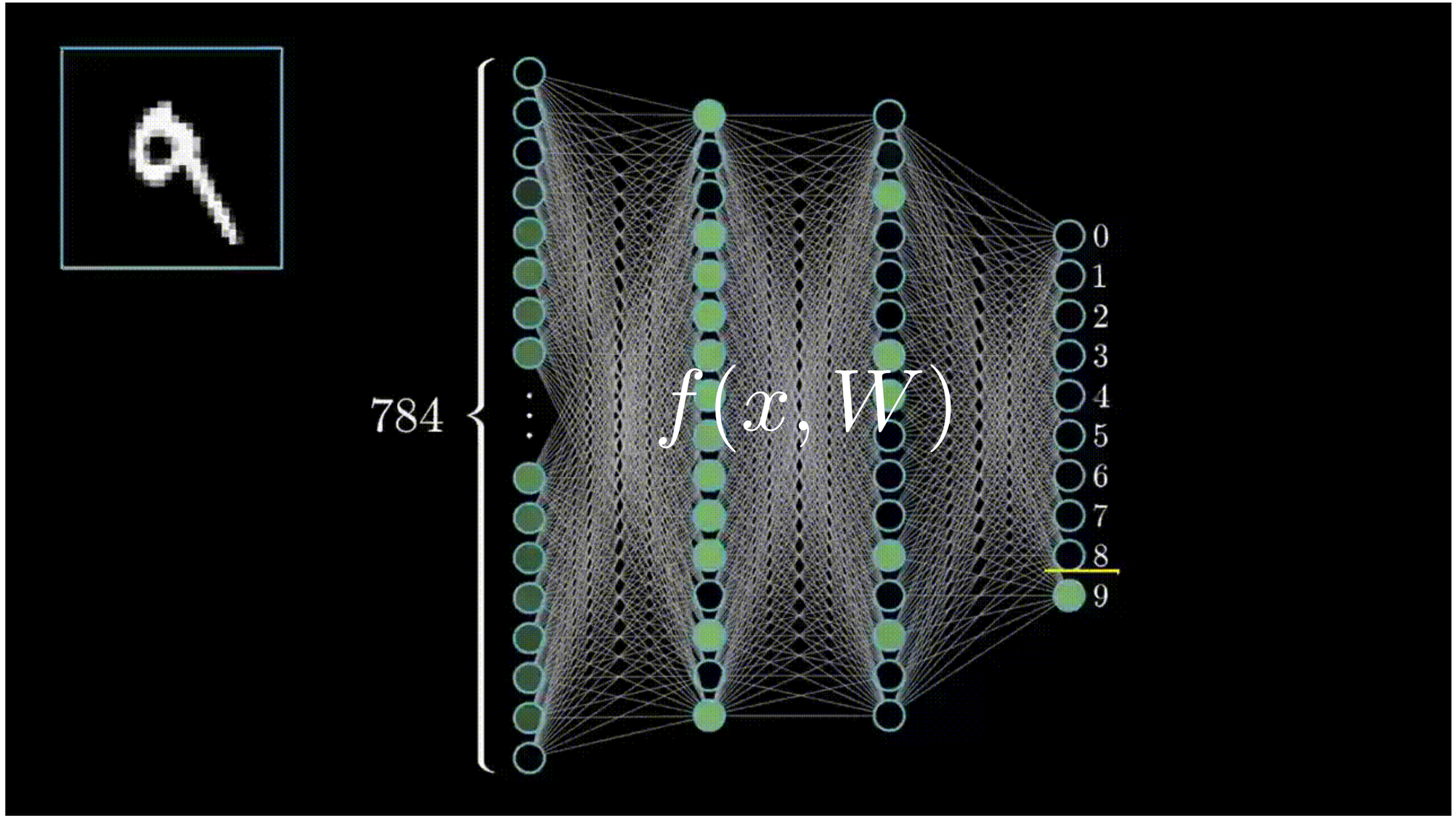
You can think of forward pass as function evaluation



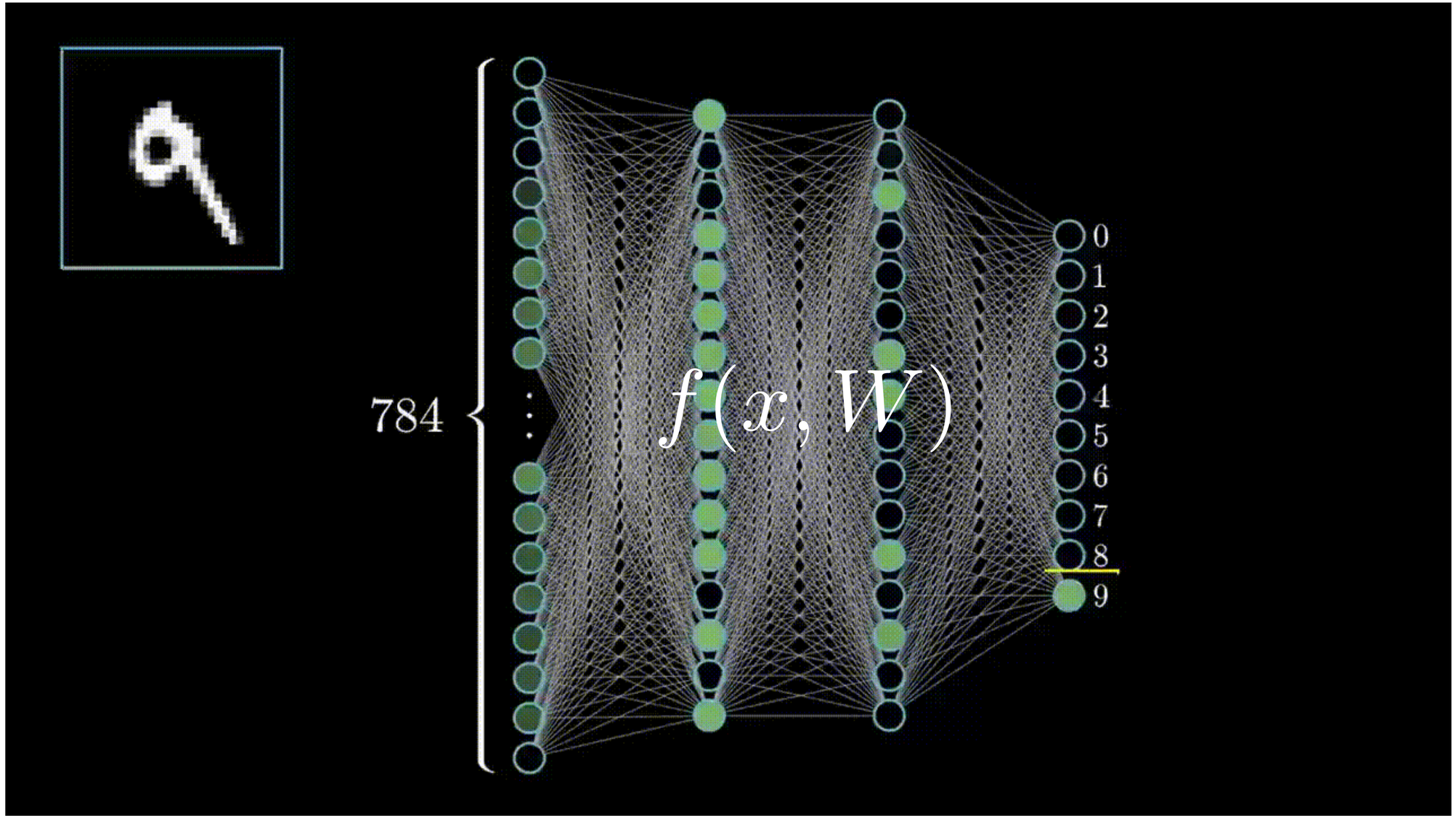
1. $z_1 = W_1 \cdot x$
2. $\sigma(z_1) = \sigma(W_1 \cdot x)$
3. $z_2 = W_2 \cdot \sigma(z_1) = W_2 \cdot \sigma(W_1 \cdot x)$
4. $\texttt{softmax}(z_2) = \texttt{softmax}(W_2 \cdot \sigma(W_1 \cdot x))$

- Nothing more complicated
- Boils down to what modules will be used
- Famous architecture due to simplicity and theoretical guarantees

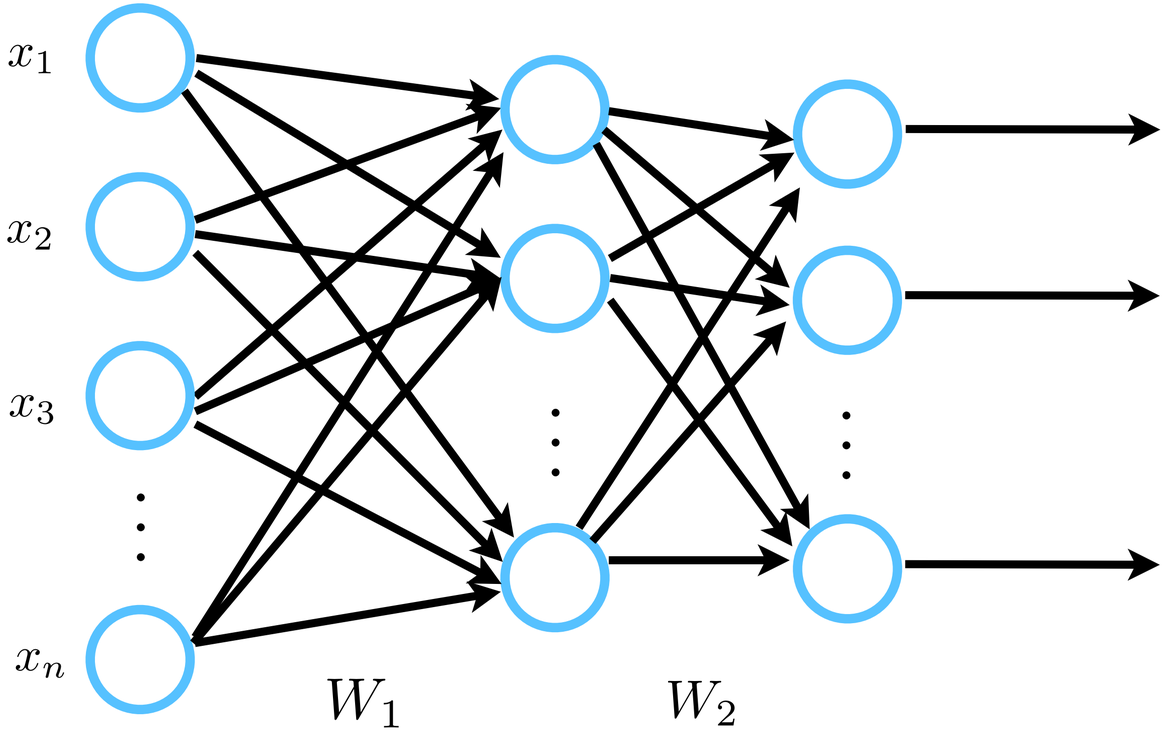
Fully connected neural networks or MLPs



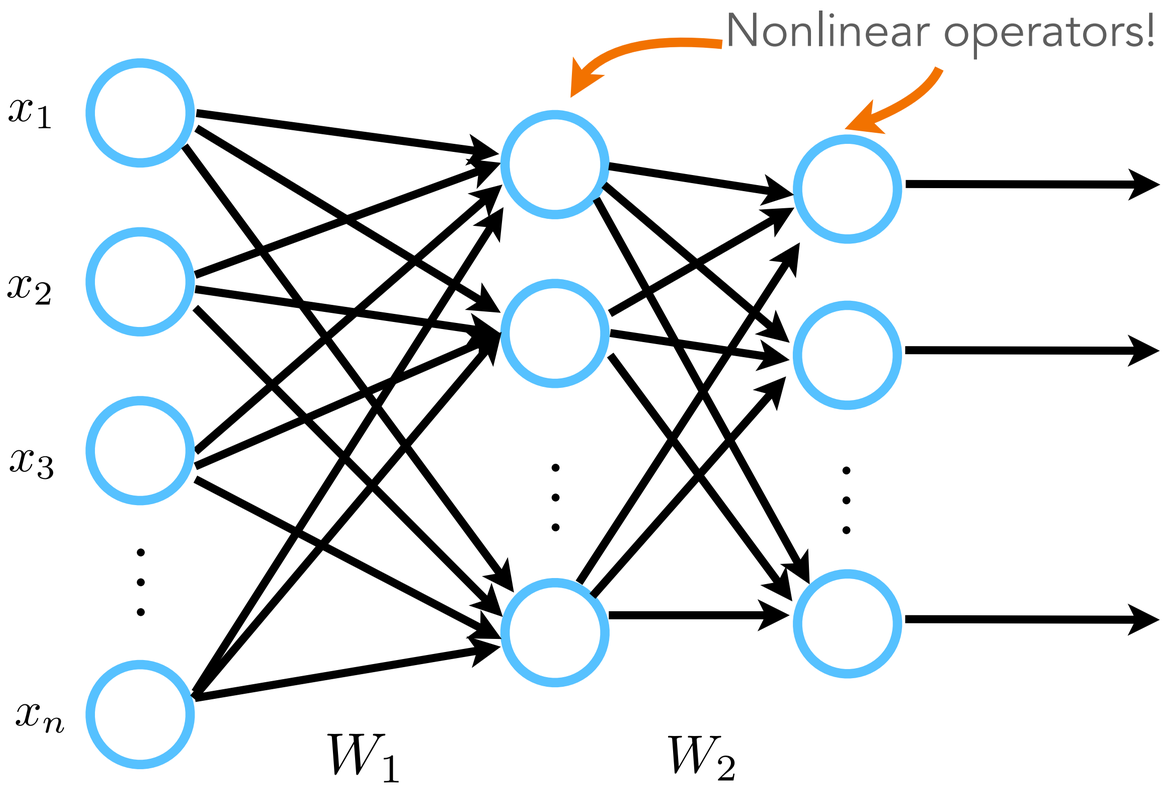
Fully connected neural networks or MLPs



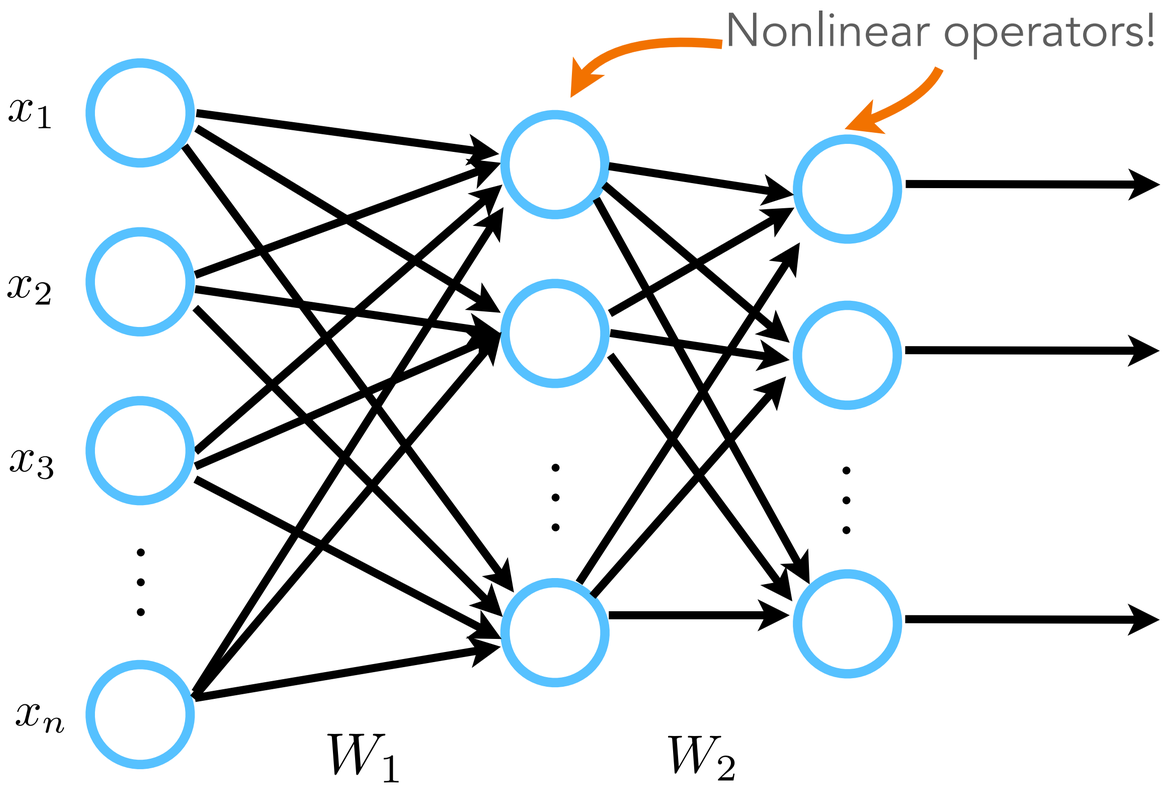
What are our options here wrt modules?



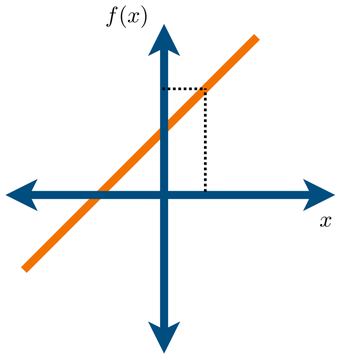
What are our options here wrt modules?

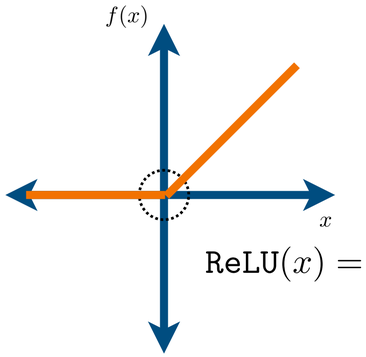


What are our options here wrt modules?



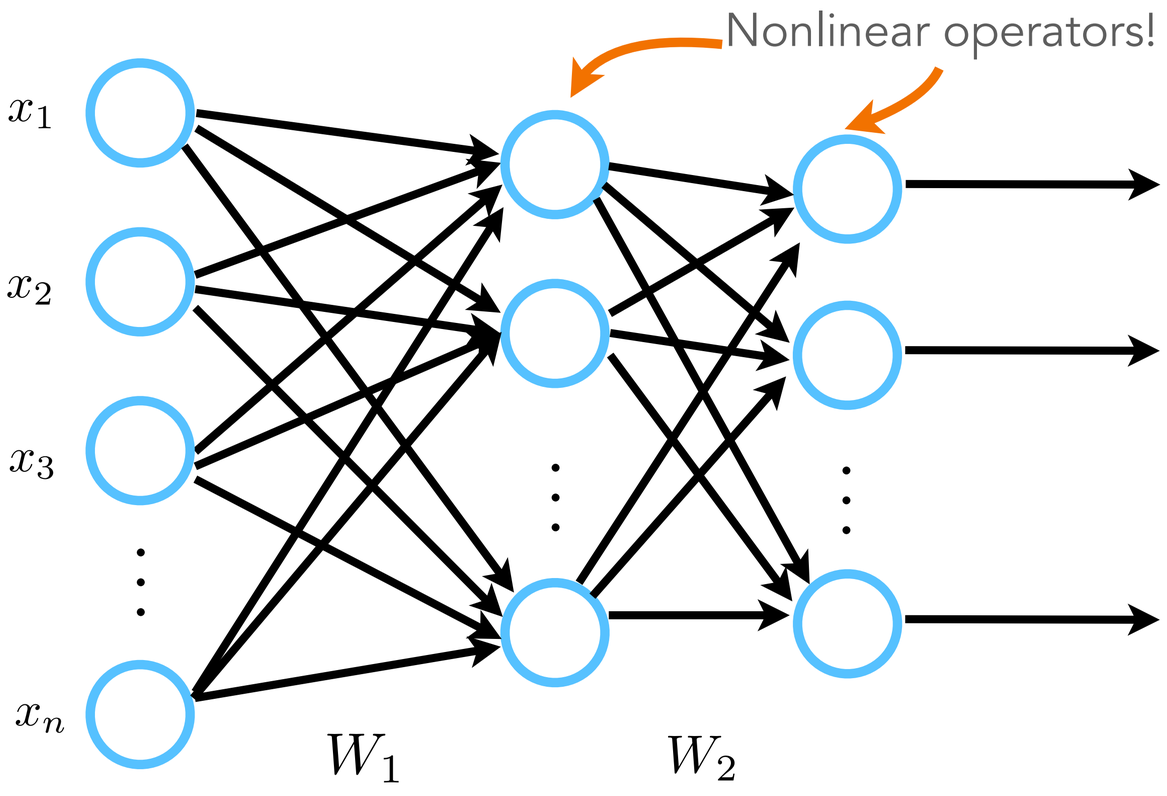
- We saw linear and ReLU



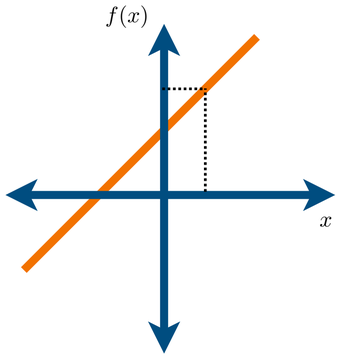


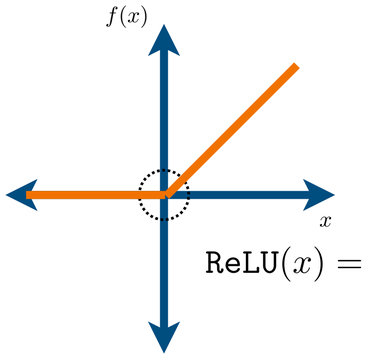
$$\texttt{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

What are our options here wrt modules?



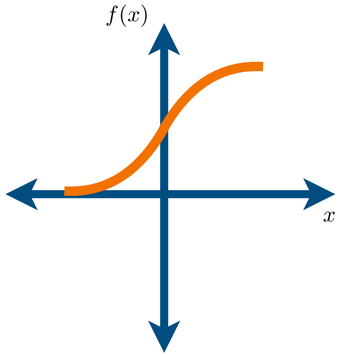
- We saw linear and ReLU



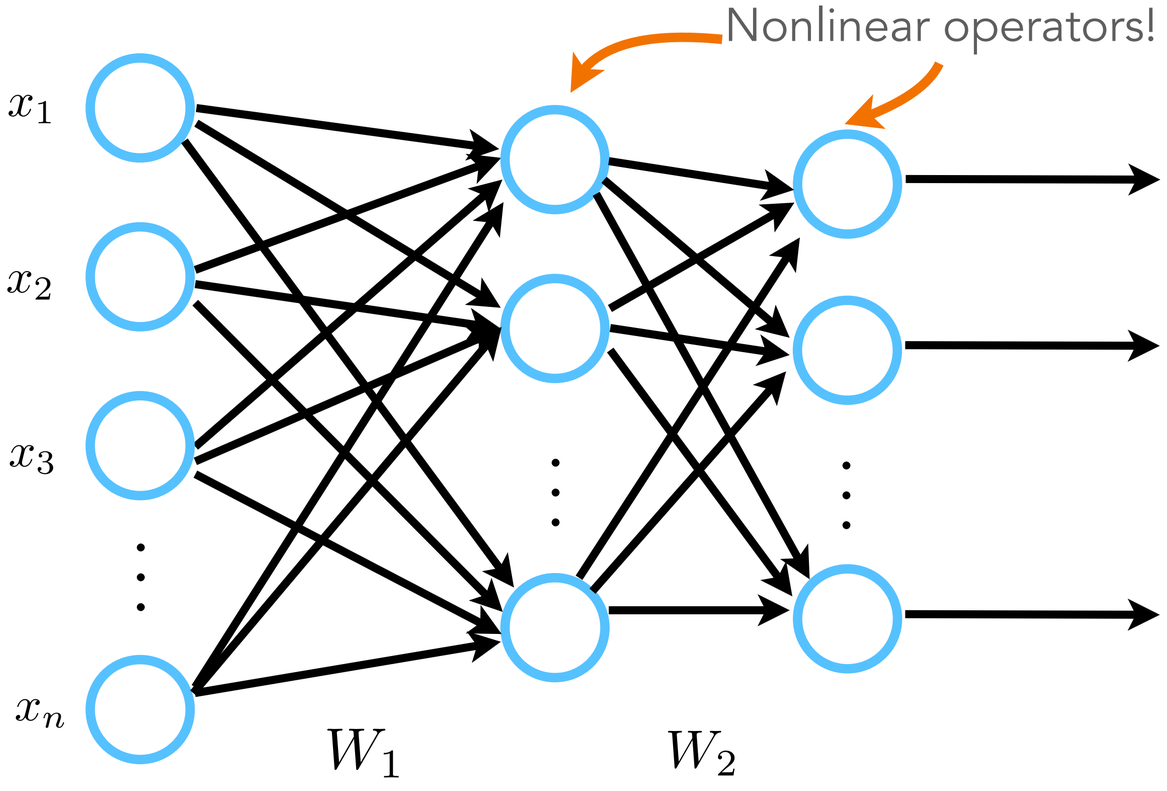


$$\texttt{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

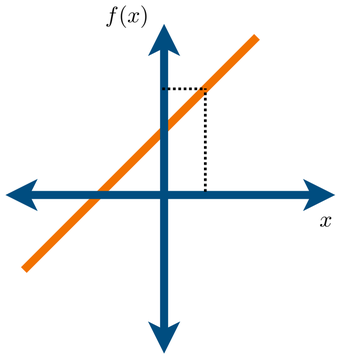
- There is also sigmoid..

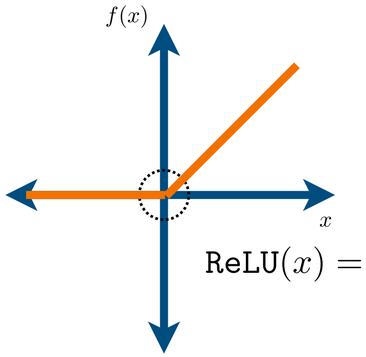


What are our options here wrt modules?



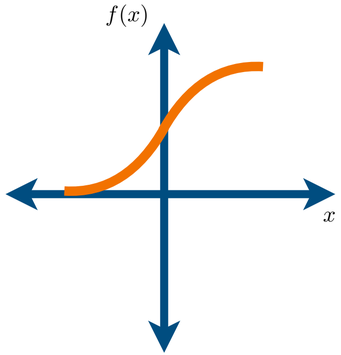
- We saw linear and ReLU

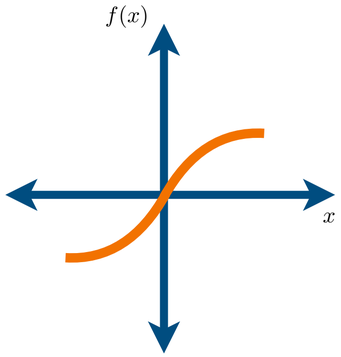




$$\texttt{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

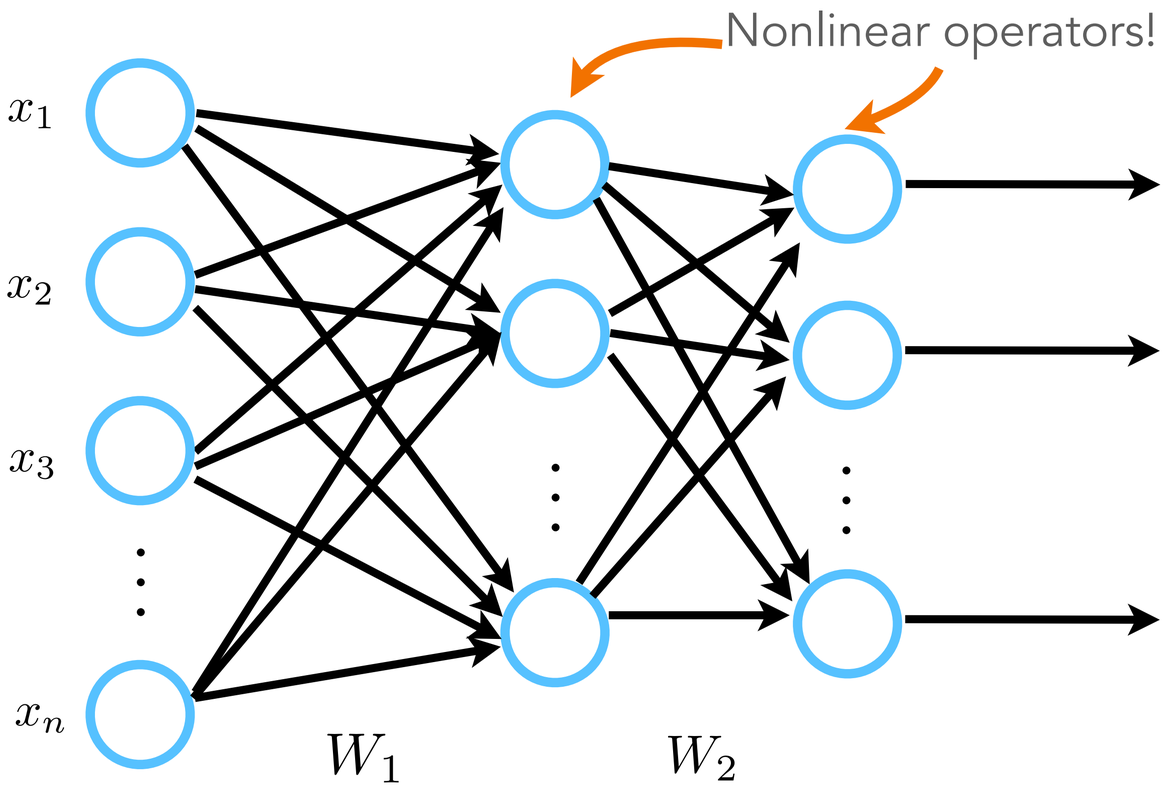
- There is also sigmoid..



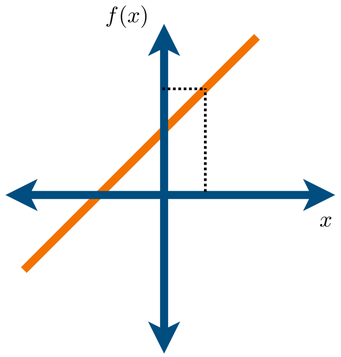


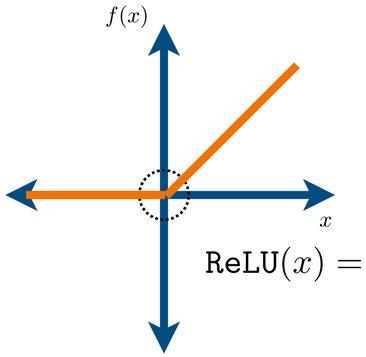
..and tanh functions.

What are our options here wrt modules?



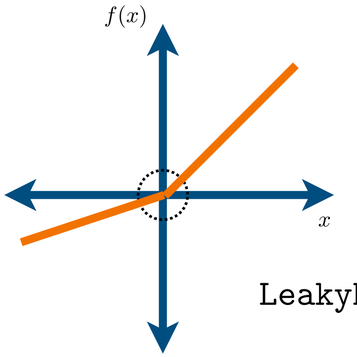
- We saw linear and ReLU

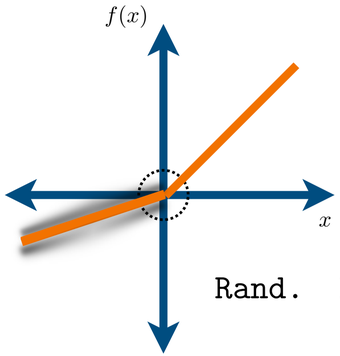




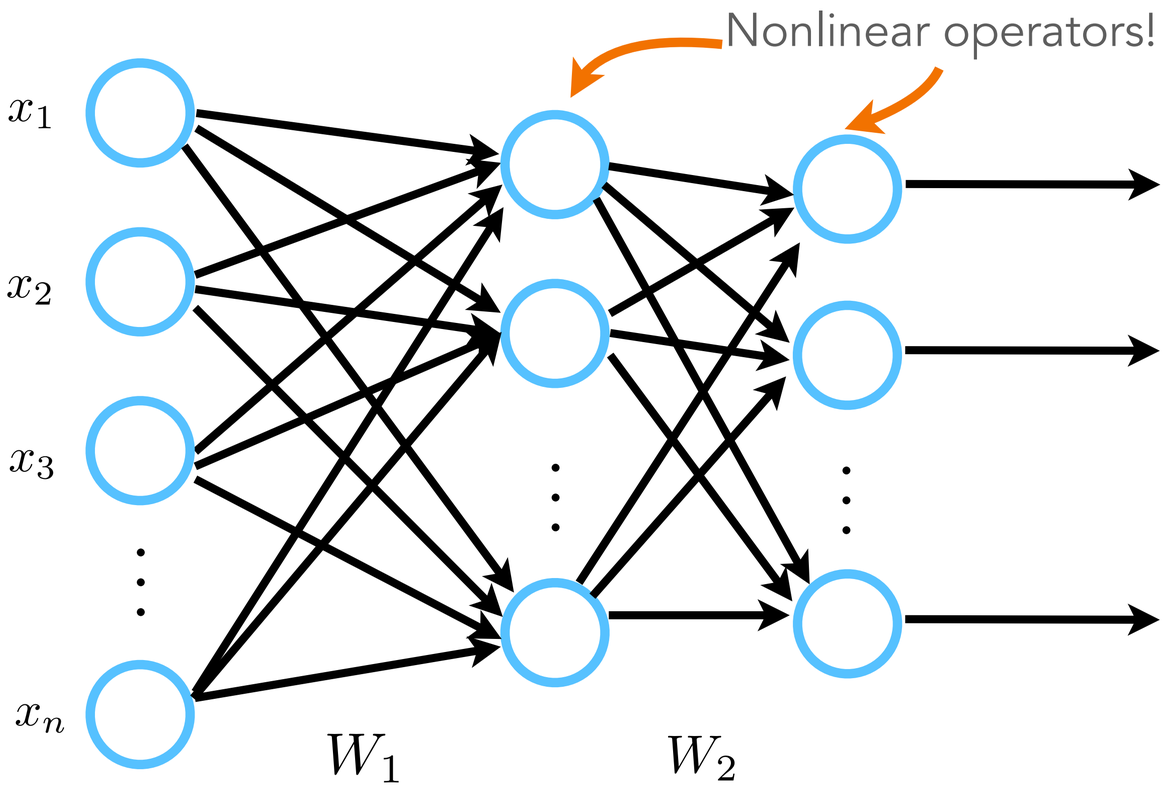
$$\texttt{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

- Zoo of ReLU functions



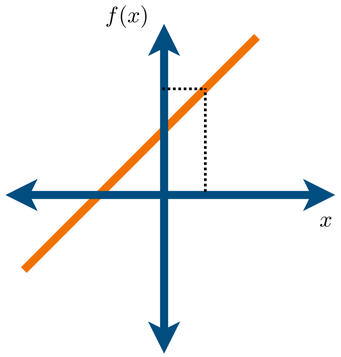


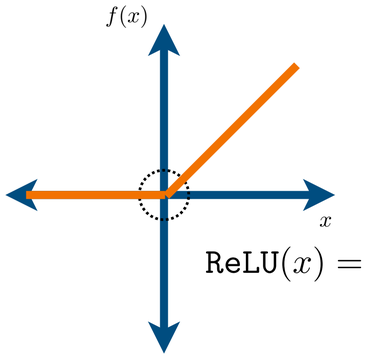
What are our options here wrt modules?



- What are the differences?

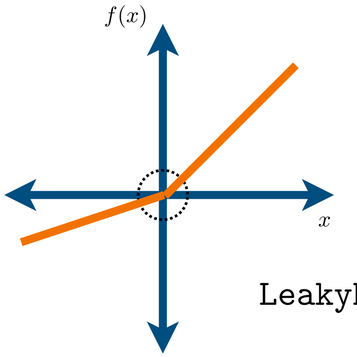
- We saw linear and ReLU

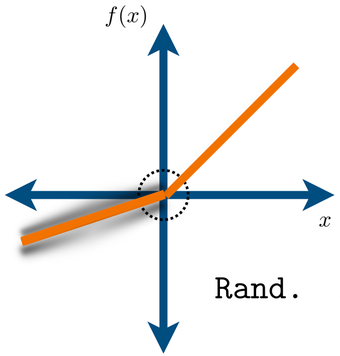




$$\texttt{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

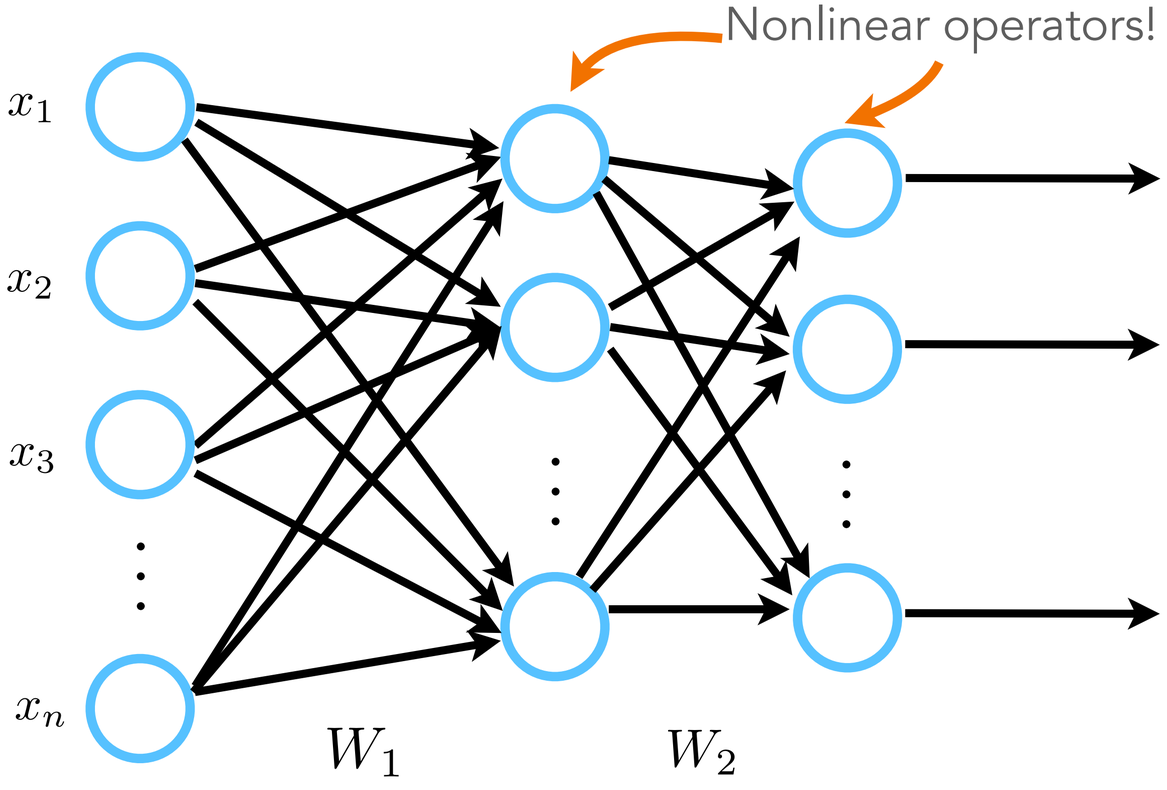
- Zoo of ReLU functions



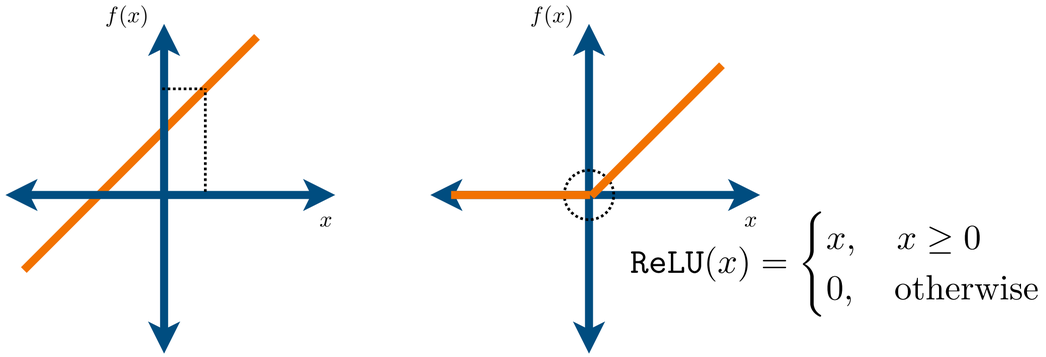


Rand. LeakyReLU

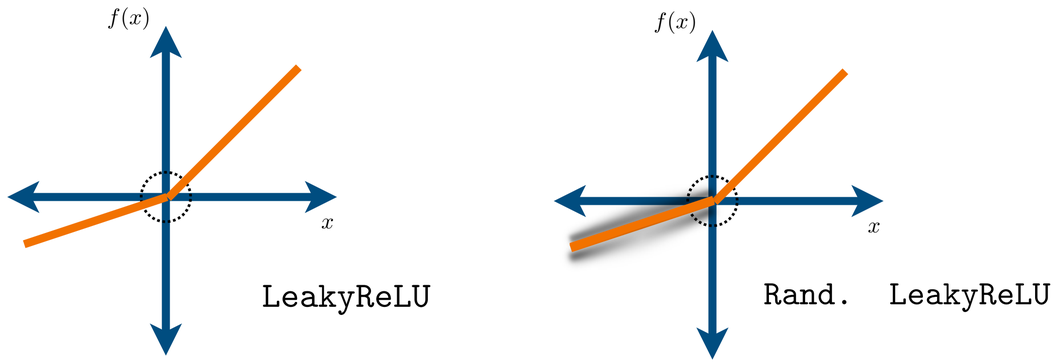
What are our options here wrt modules?



- We saw linear and ReLU

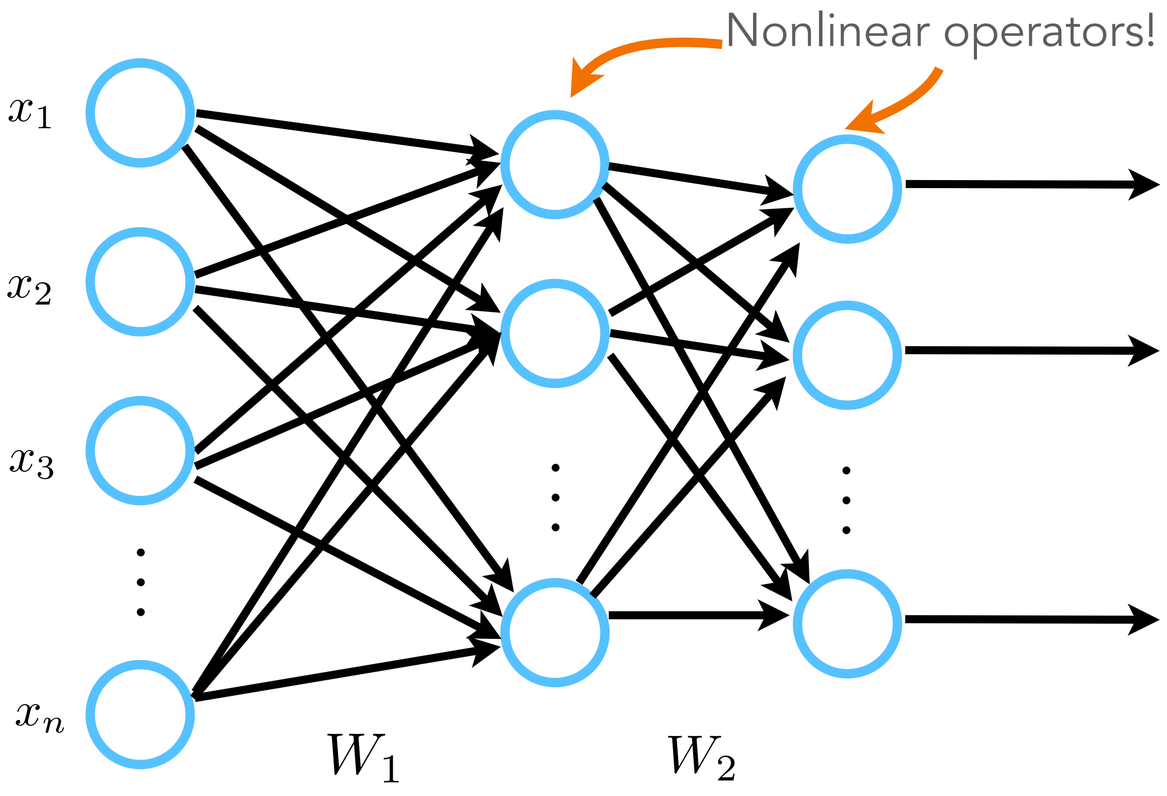


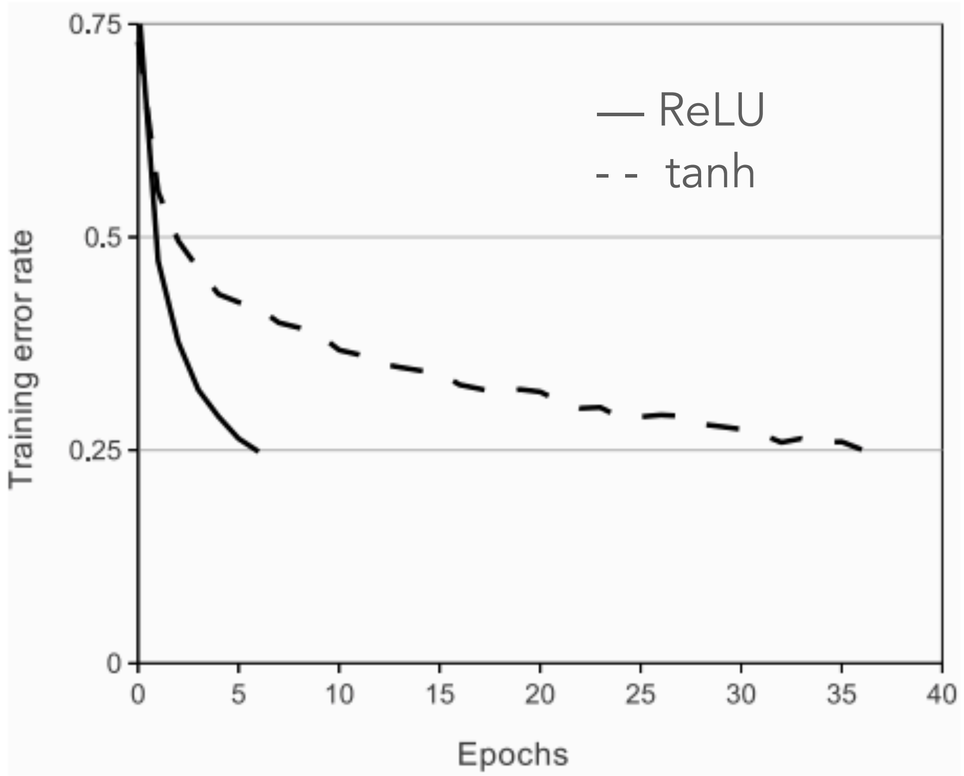
- Zoo of ReLU functions



- What are the differences?
  - Some modules have simpler derivates than others (affects optimization)

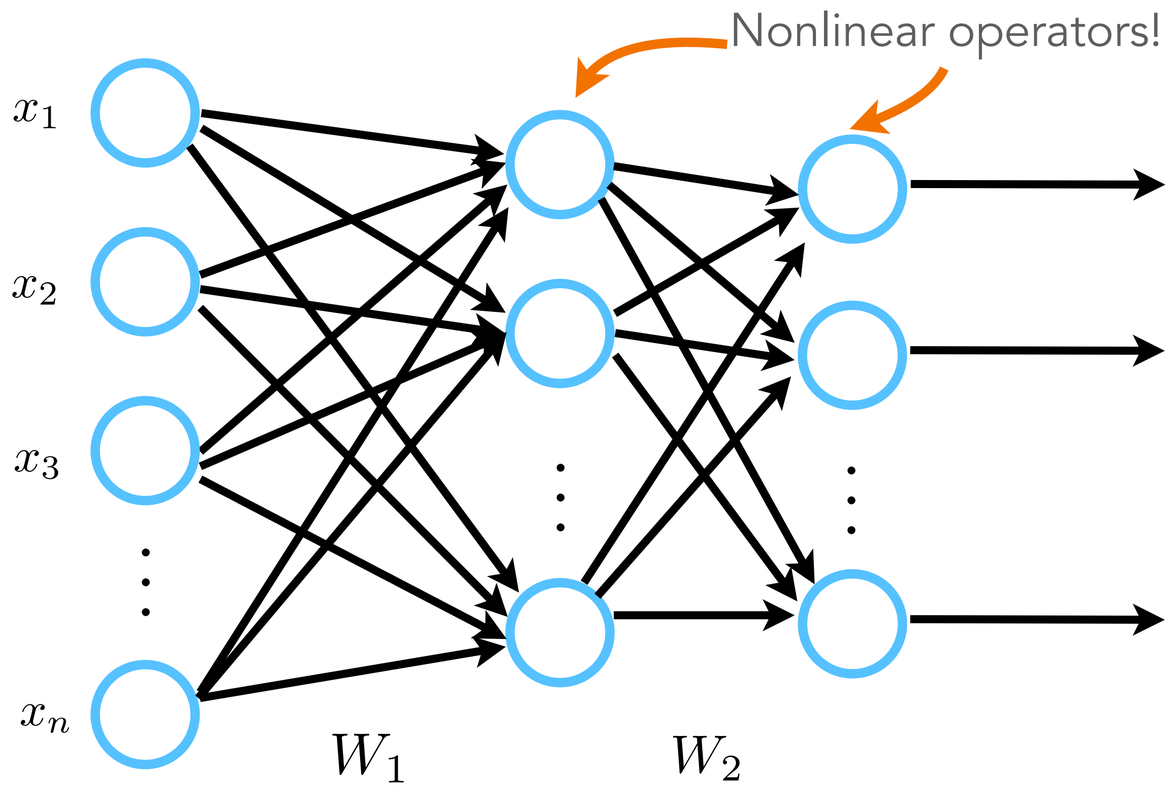
What are our options here wrt modules?

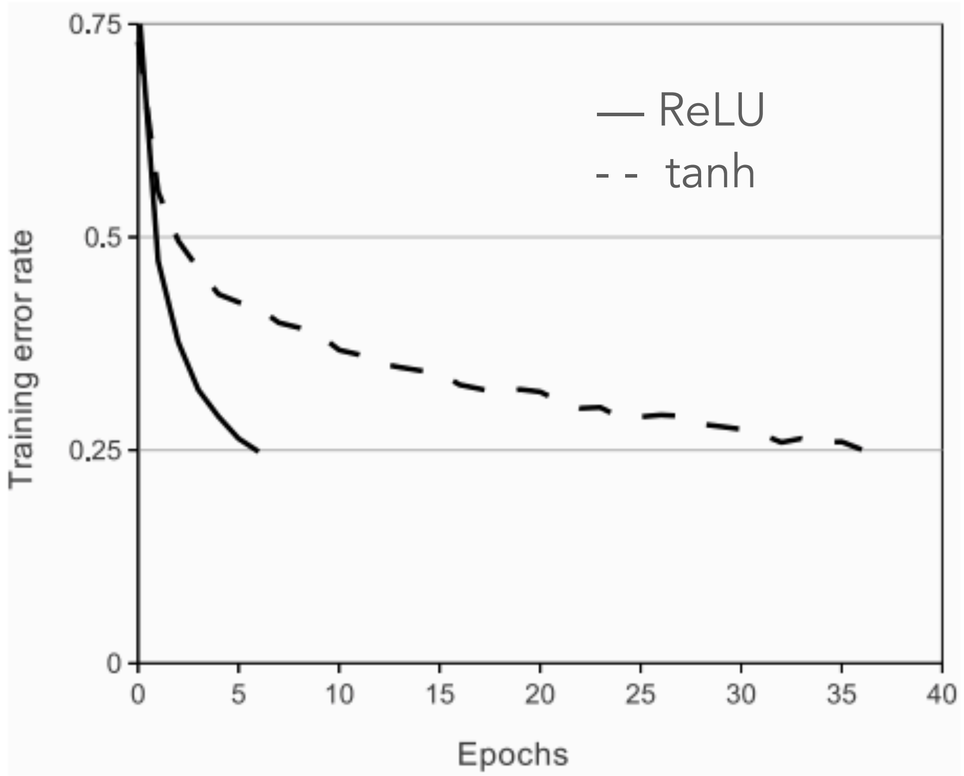




- What are the differences?
  - Some modules have simpler derivates than others (affects optimization)

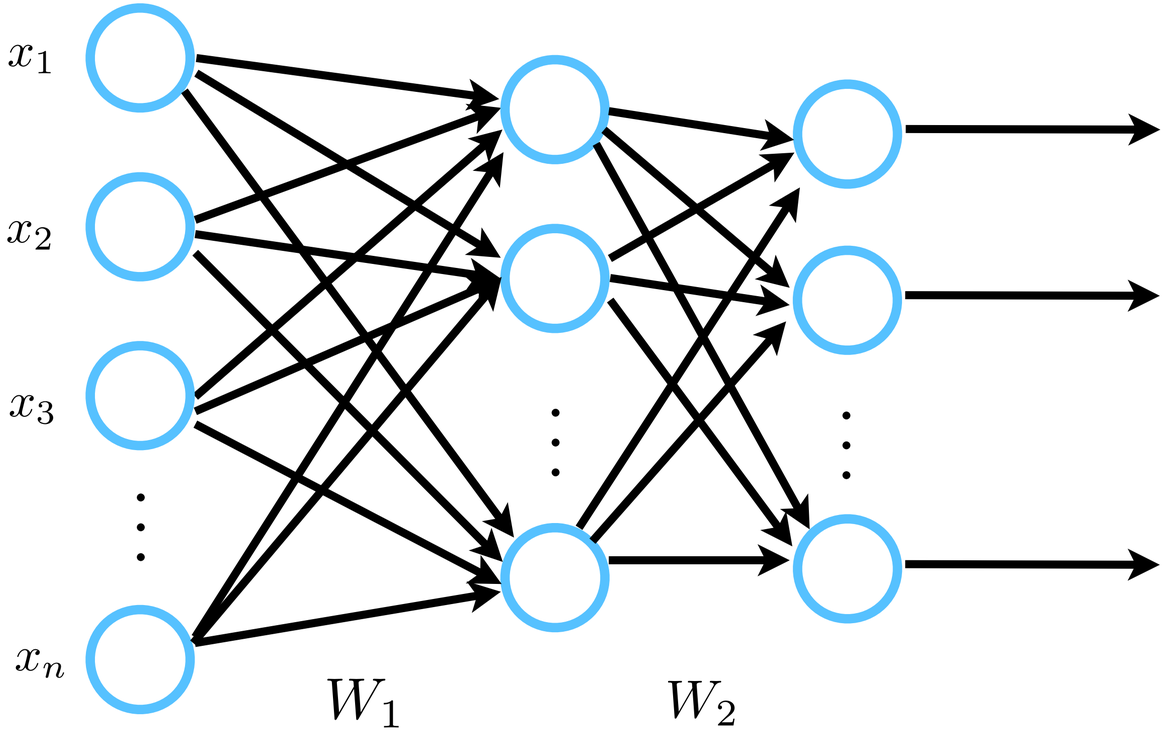
What are our options here wrt modules?



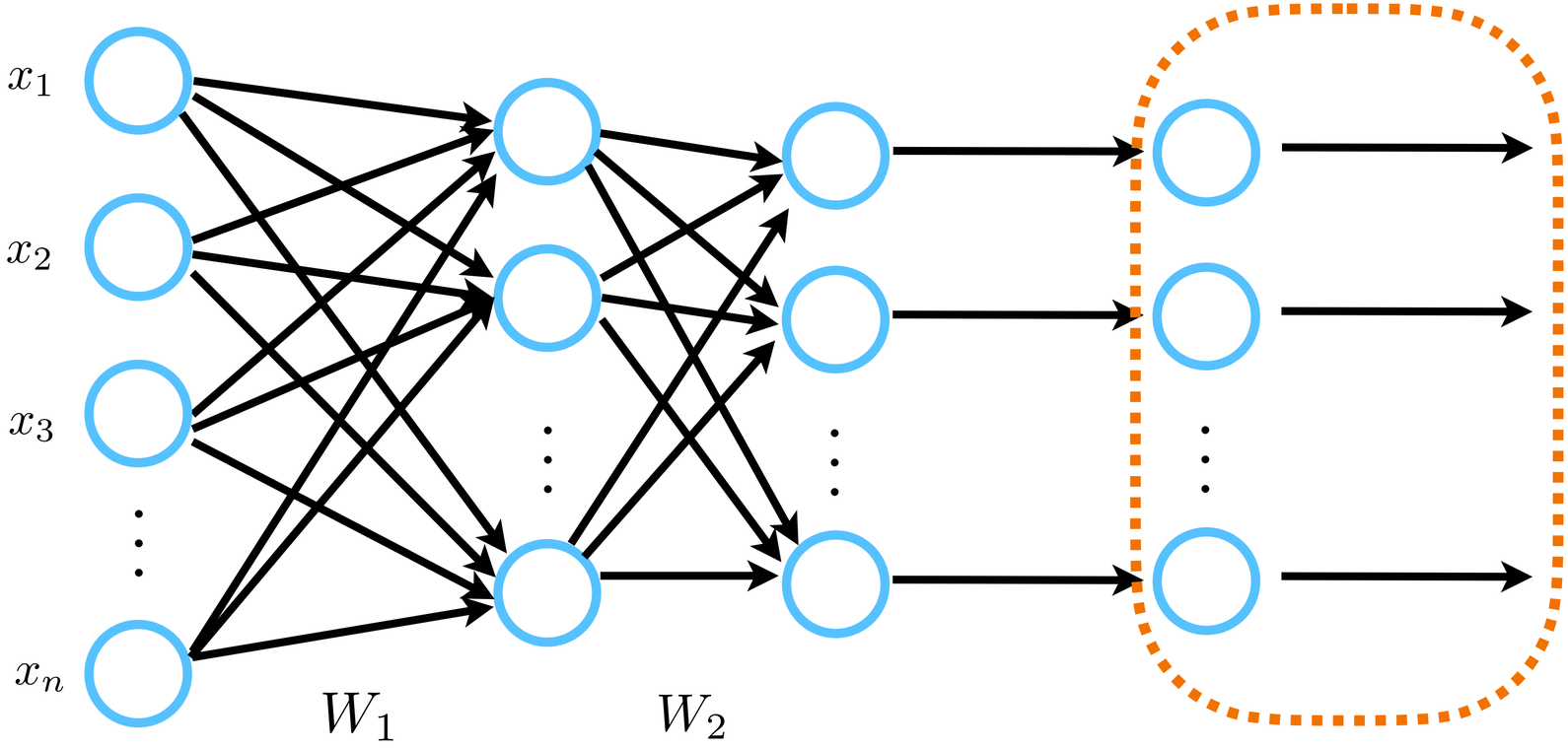


- What are the differences?
  - Some modules have simpler derivates than others (affects optimization)
  - Saturation issues (vanishing/exploding gradients)

What are our options here wrt modules?

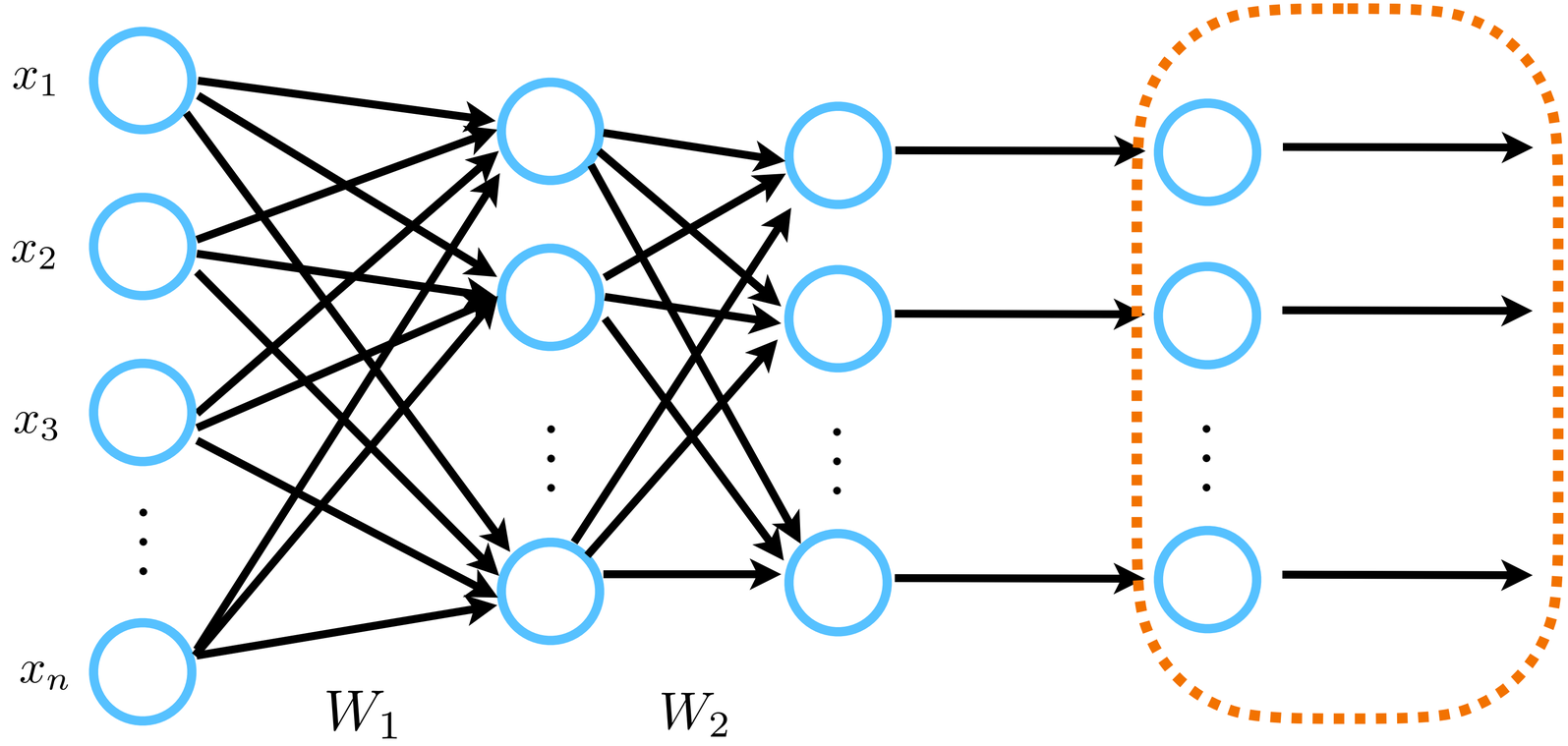


What are our options here wrt modules?



- In many cases, we use an additional module: $\texttt{softmax} : \mathbb{R}^k \to \mathbb{R}^k$

What are our options here wrt modules?

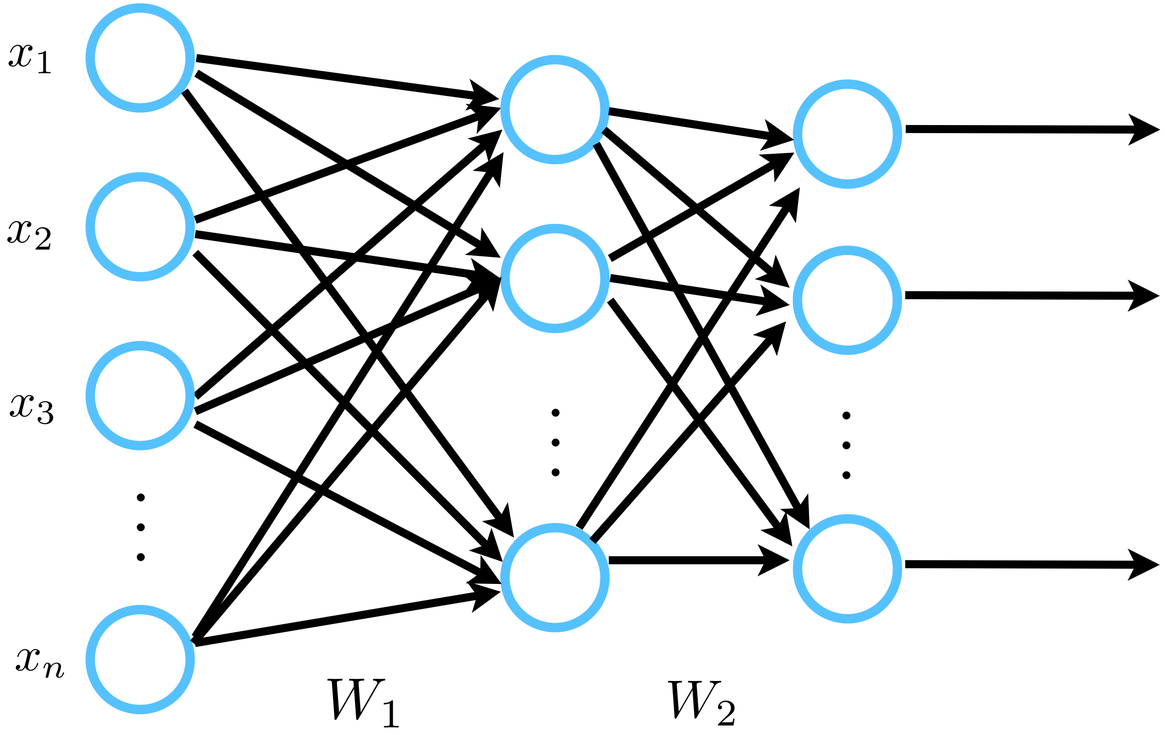


- In many cases, we use an additional module: $\texttt{softmax} : \mathbb{R}^k \to \mathbb{R}^k$

$$\texttt{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

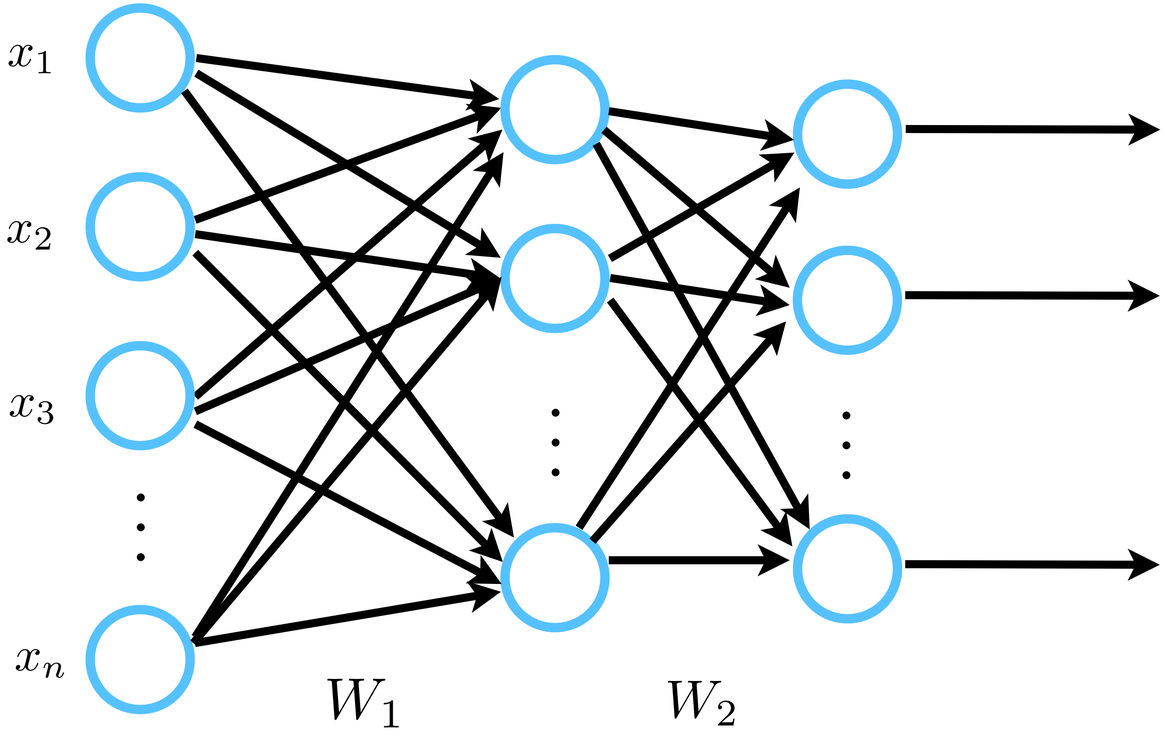
- Even for this module, there are alternatives: sparsemax, sparse projections onto the simplex, ..
- Out of scope

What are our options here wrt modules?



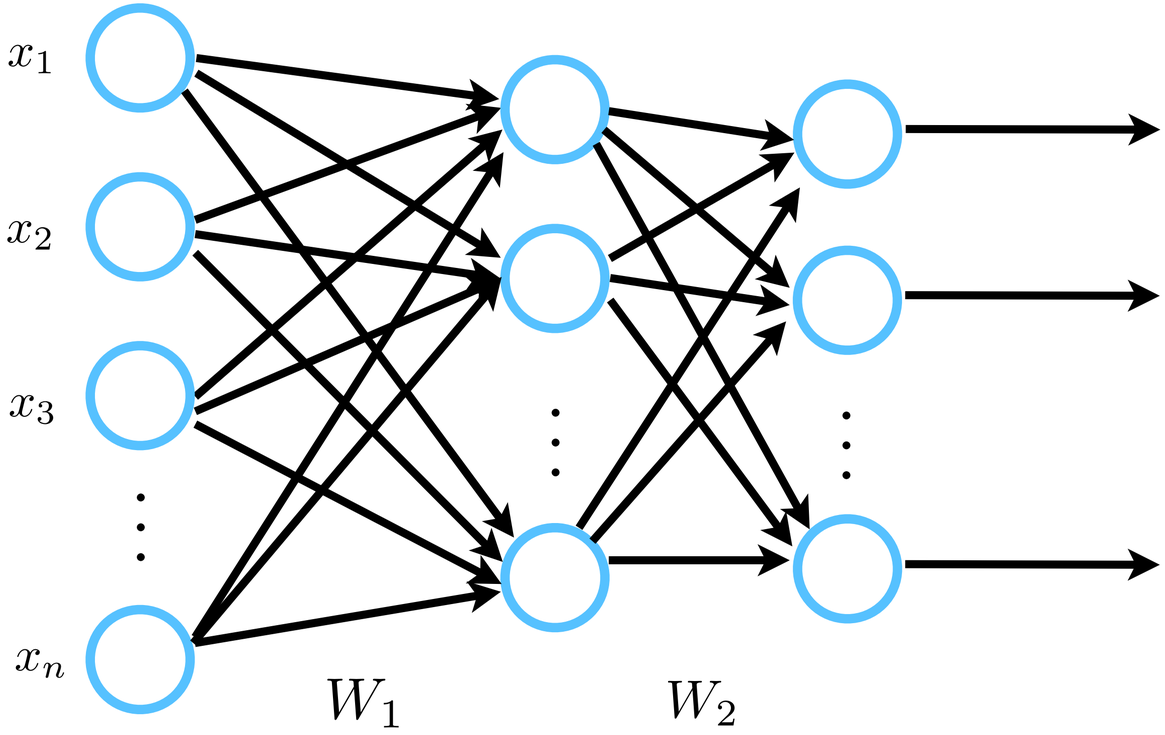
- Regularization modules: dropout and batch normalization
  - Dropout resembles to ensemble training: trains for subset of iterations a subset of the network - this creates diversity in decisions at the end, increasing accuracy

What are our options here wrt modules?



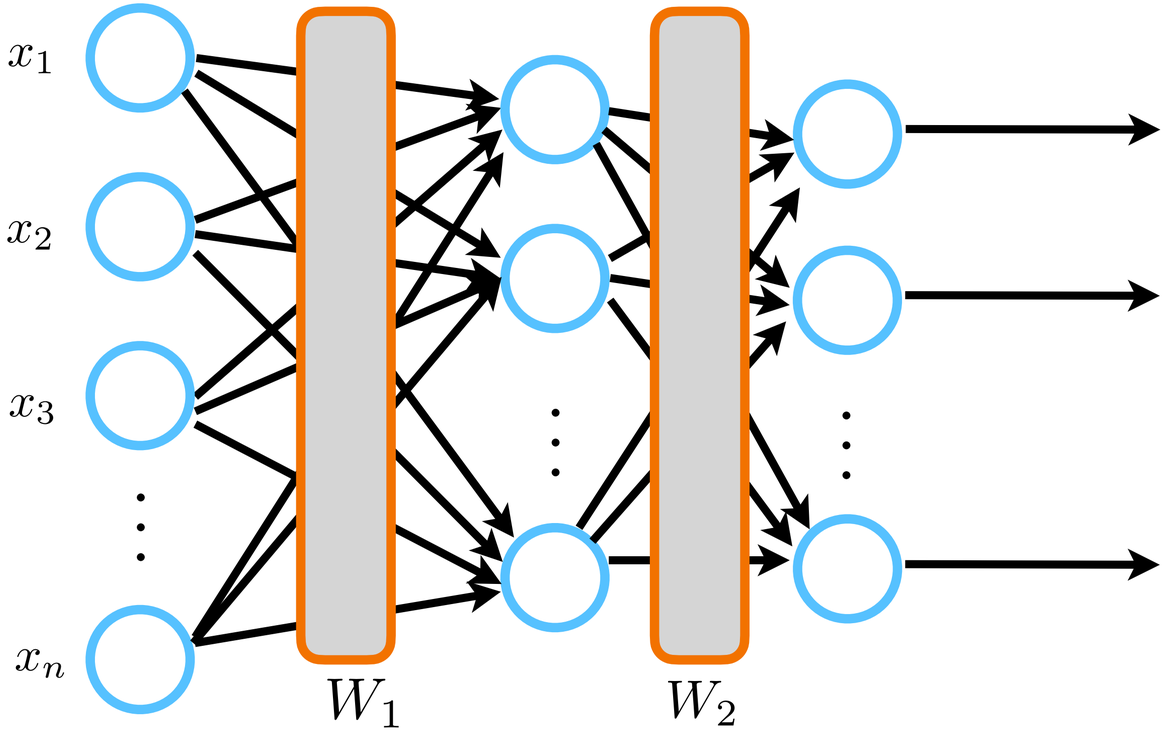
- Regularization modules: dropout and batch normalization
  - Dropout resembles to ensemble training: trains for subset of iterations a subset of the network - this creates diversity in decisions at the end, increasing accuracy

What are our options here wrt modules?



- Regularization modules: dropout and batch normalization
  - Dropout resembles to ensemble training: trains for subset of iterations a subset of the network - this creates diversity in decisions at the end, increasing accuracy

What are our options here wrt modules?



$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} x_i$$

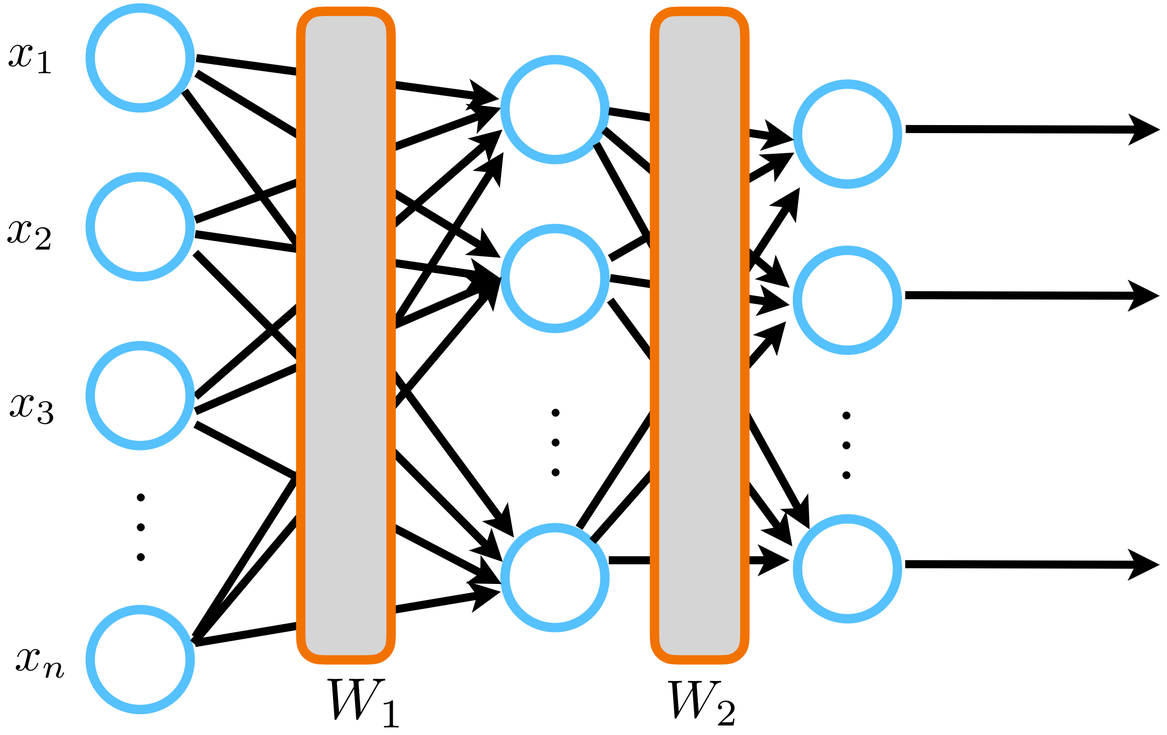
$$\sigma_B = \frac{1}{B}\sum_{i=1} (x_i - \mu_B)^2$$

$$\widehat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$\widehat{y}_i = \gamma \cdot \widehat{x}_i + \beta$$

- Regularization modules: dropout and batch normalization
  - Dropout resembles to ensemble training: trains for subset of iterations a subset of the network - this creates diversity in decisions at the end, increasing accuracy
  - Batch normalization makes sure that inputs to layers behave like “whitened data”.

What are our options here wrt modules?



$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} x_i$$

$$\sigma_B = \frac{1}{B}\sum_{i=1} (x_i - \mu_B)^2$$

$$\widehat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$\widehat{y}_i = \gamma \cdot \widehat{x}_i + \beta$$

Both techniques increase significantly the performance of a neural network (e.g., BN allows larger step sizes)

- Regularization modules: dropout and batch normalization
  - Dropout resembles to ensemble training: trains for subset of iterations a subset of the network - this creates diversity in decisions at the end, increasing accuracy
  - Batch normalization makes sure that inputs to layers behave like “whitened data”.

``But how do we train a neural network?"

Overview

- Introduction to neural networks
- Feedforward / Fully-connected neural networks or MLPs
- Convolutional neural networks (CNNs)
- Recurrent neural networks (RNNs)
- Optimization in neural network training

Overview

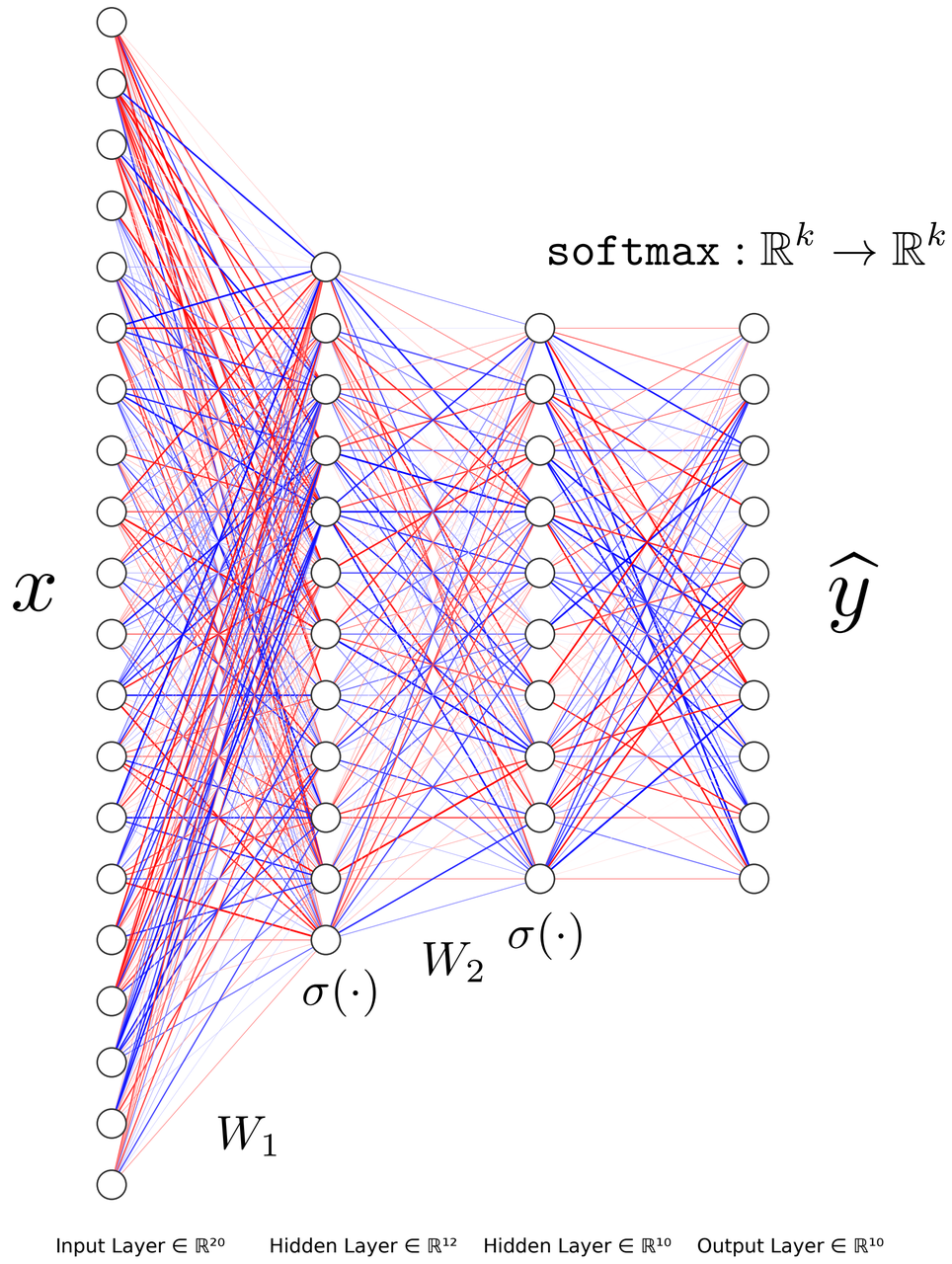
- Optimization in neural network training

$$\min_{x} \quad f(x)$$

$$\text{s.t.} \quad x \in \mathcal{C}$$

Examples

- Neural networks



Examples

- Neural networks

$$\begin{aligned} \min_{X} \quad & f(X) \\ \text{s.t.} \quad & X \in \mathcal{C} \end{aligned} \quad \Rightarrow \quad \min_{W_i} f(W_1, W_2) := \frac{1}{n} \sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i)$$

where

$$\widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

$$\min_{x} \quad f(x)$$

$$\text{s.t.} \quad x \in \mathcal{C}$$

Smooth function
(differentiable)

$$\min_{x} \quad f(x)$$

$$\text{s.t.} \quad x \in \mathcal{C}$$

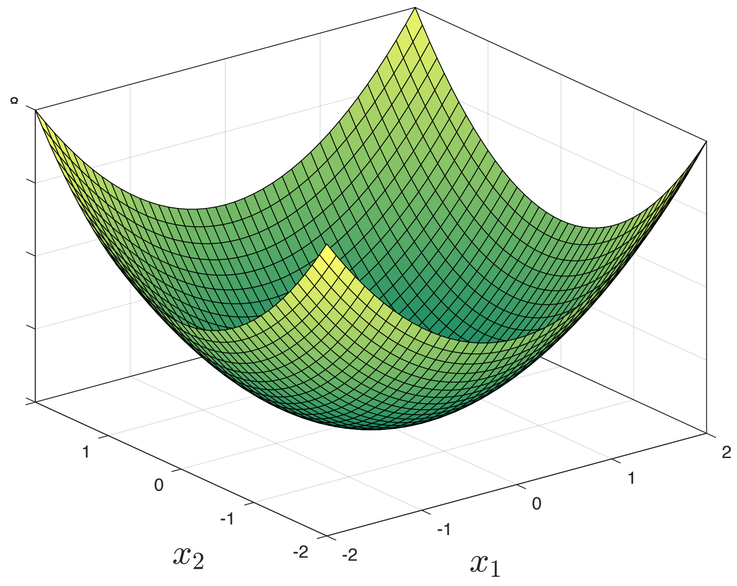
“Nice” functions vs. “nasty” functions

“Nice” functions vs. “nasty” functions

$$\min_x \frac{1}{n} \sum_{i=1}^{n} \left(y_i - a_i^\top x\right)^2$$

“Nice” functions vs. “nasty” functions

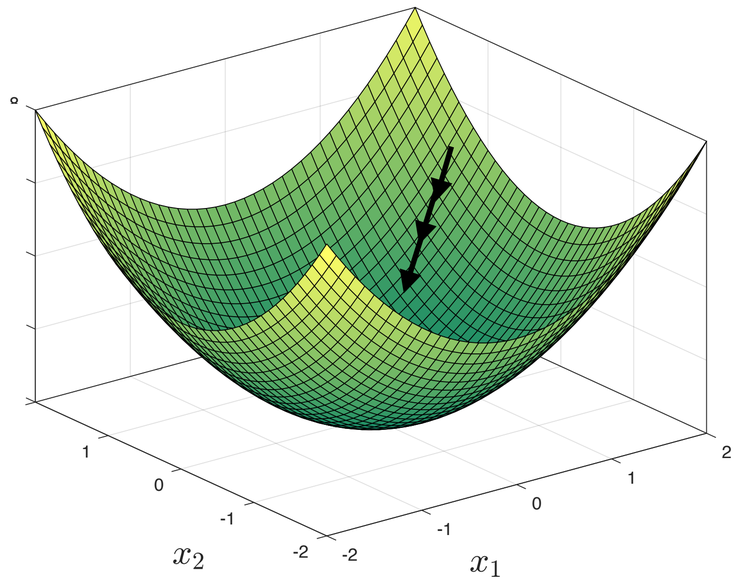
$$\min_x \frac{1}{n} \sum_{i=1}^{n} \left(y_i - a_i^\top x\right)^2$$



Least-squares

"Nice" functions vs. "nasty" functions

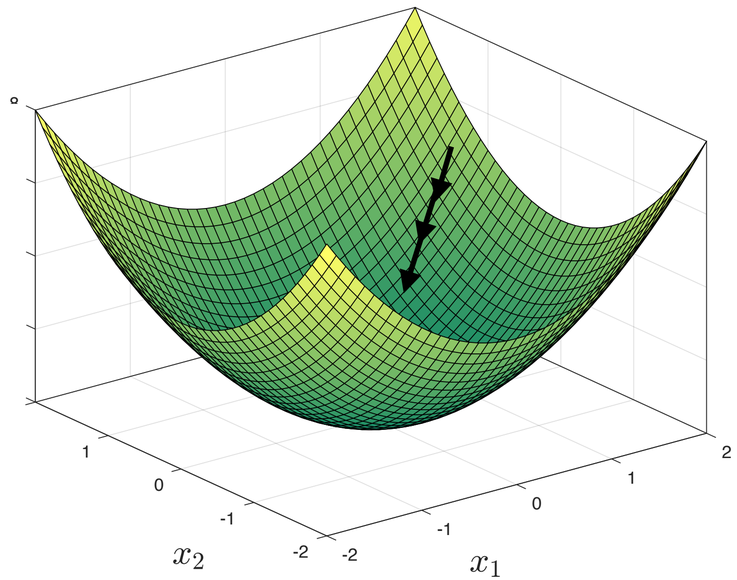
$$\min_x \frac{1}{n} \sum_{i=1}^{n} \left(y_i - a_i^\top x\right)^2$$



Least-squares

"Nice" functions vs. "nasty" functions

$$\min_x \frac{1}{n} \sum_{i=1}^{n} \left(y_i - a_i^\top x\right)^2$$



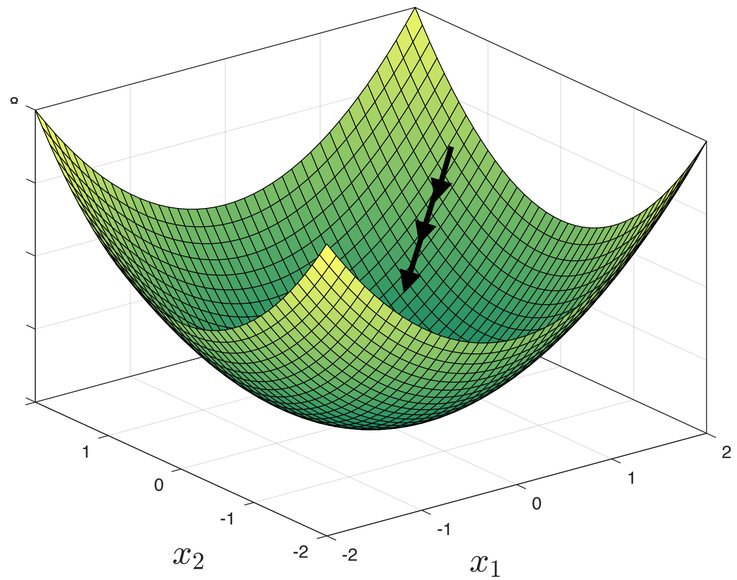
Least-squares

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n} \sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i)$$

$$\widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

"Nice" functions vs. "nasty" functions

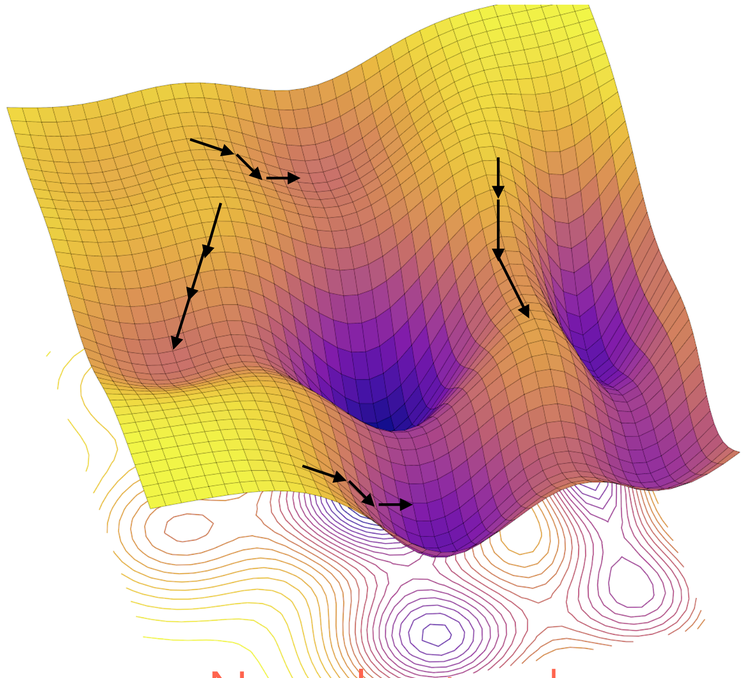
$$\min_x \frac{1}{n} \sum_{i=1}^{n} \left(y_i - a_i^\top x\right)^2$$



Least-squares

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n} \sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i)$$

$$\widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$



Neural networks

Gradient descent

Gradient as local information (short story)

Gradient as local information (short story)

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

Motivation: Gradient descent for MLPs

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n} \sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i) \quad \text{where} \quad \widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

Motivation: Gradient descent for MLPs

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n} \sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i) \quad \text{where} \quad \widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

- Using modules:

Motivation: Gradient descent for MLPs

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n} \sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i) \quad \text{where} \quad \widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

- Using modules:

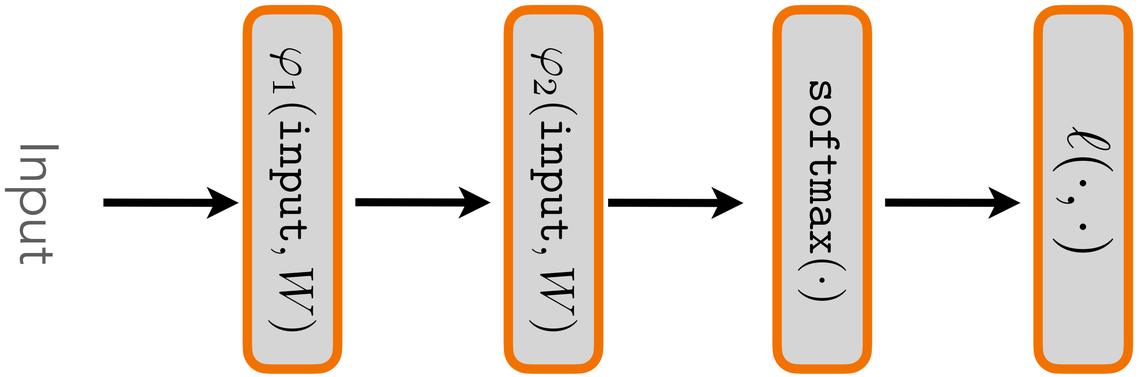
$$\varphi_1(\texttt{input}, W) = \sigma(W_1 \cdot x_i)$$
$$\varphi_2(\texttt{input}, W) = \sigma(W_2 \cdot \varphi_2(\cdot))$$

Motivation: Gradient descent for MLPs

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n}\sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i) \quad \text{where} \quad \widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

- Using modules:

$$\varphi_1(\texttt{input}, W) = \sigma(W_1 \cdot x_i)$$
$$\varphi_2(\texttt{input}, W) = \sigma(W_2 \cdot \varphi_2(\cdot))$$

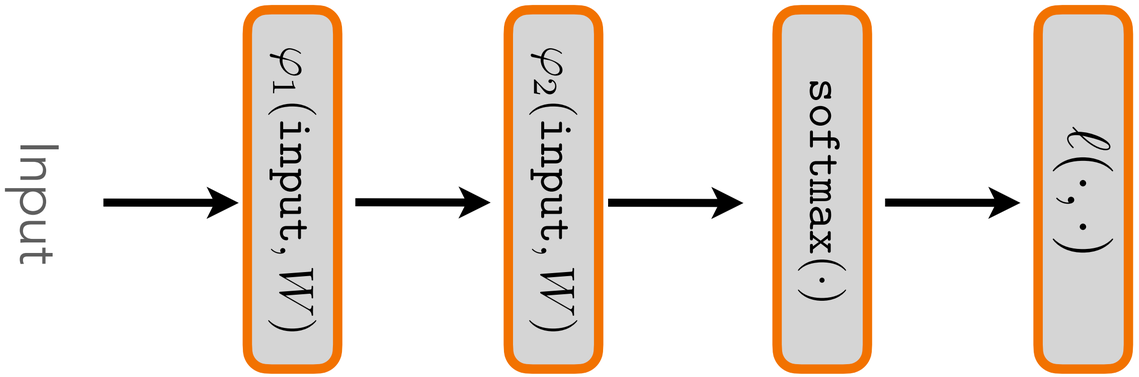


Motivation: Gradient descent for MLPs

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n}\sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i) \quad \text{where} \quad \widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

- Using modules:

$$\varphi_1(\texttt{input}, W) = \sigma(W_1 \cdot x_i)$$
$$\varphi_2(\texttt{input}, W) = \sigma(W_2 \cdot \varphi_2(\cdot))$$



$$\begin{aligned}
\nabla f_i(x) &= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial W} \\
&= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial \texttt{softmax}(\cdot)} \cdot \frac{\partial \texttt{softmax}(\cdot)}{\partial W} \\
&= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial \texttt{softmax}(\cdot)} \cdot \frac{\partial \texttt{softmax}(\cdot)}{\partial \varphi(\cdot, W_2)} \cdot \frac{\partial \varphi(\cdot, W_2)}{\partial W} \\
&= \cdots
\end{aligned}$$

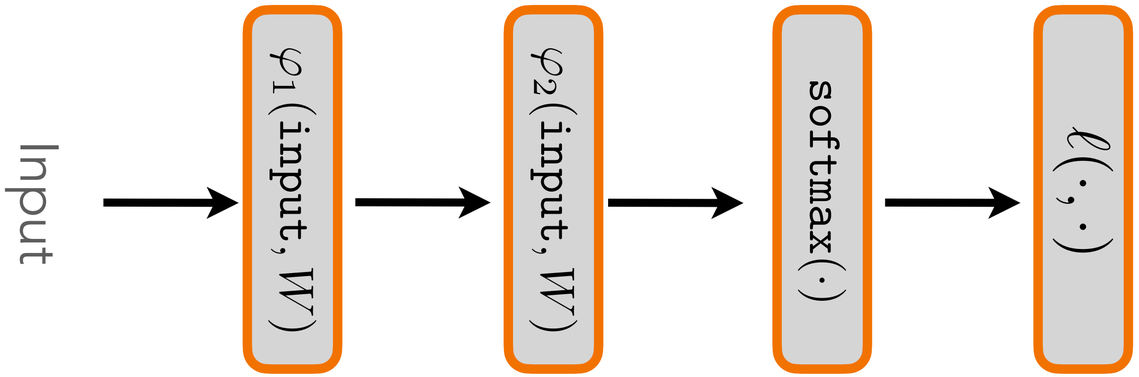
(chain rule of derivatives)

Motivation: Gradient descent for MLPs

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n}\sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i) \quad \text{where} \quad \widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

- Using modules:

$$\varphi_1(\texttt{input}, W) = \sigma(W_1 \cdot x_i)$$
$$\varphi_2(\texttt{input}, W) = \sigma(W_2 \cdot \varphi_2(\cdot))$$



$$\begin{aligned}
\nabla f_i(x) &= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial W} \\
&= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial \texttt{softmax}(\cdot)} \cdot \frac{\partial \texttt{softmax}(\cdot)}{\partial W} \\
&= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial \texttt{softmax}(\cdot)} \cdot \frac{\partial \texttt{softmax}(\cdot)}{\partial \varphi(\cdot, W_2)} \cdot \frac{\partial \varphi(\cdot, W_2)}{\partial W} \\
&= \cdots
\end{aligned}$$

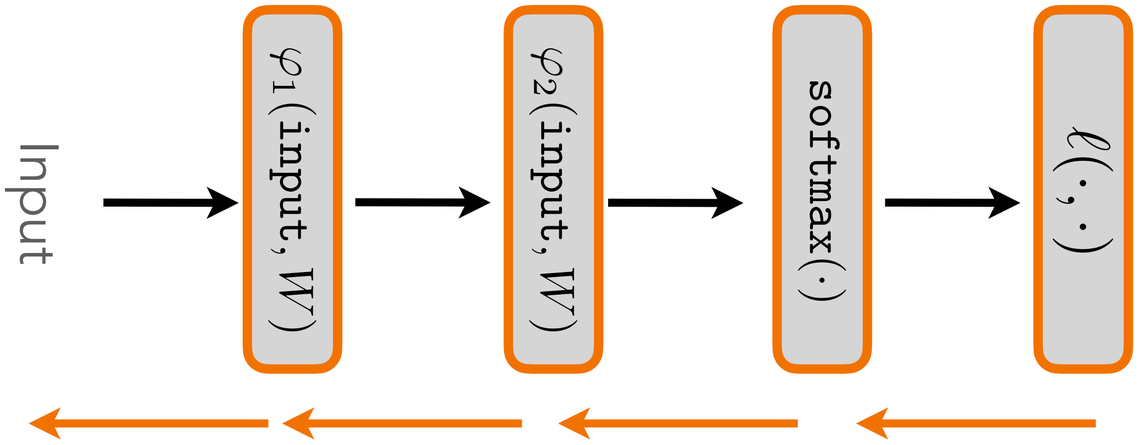
(chain rule of derivatives)

Motivation: Gradient descent for MLPs

$$\min_{W_i} f(W_1, W_2) := \frac{1}{n}\sum_{i=1}^{n} \ell(\widehat{y}_i,\ y_i) \quad \text{where} \quad \widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)$$

- Using modules:

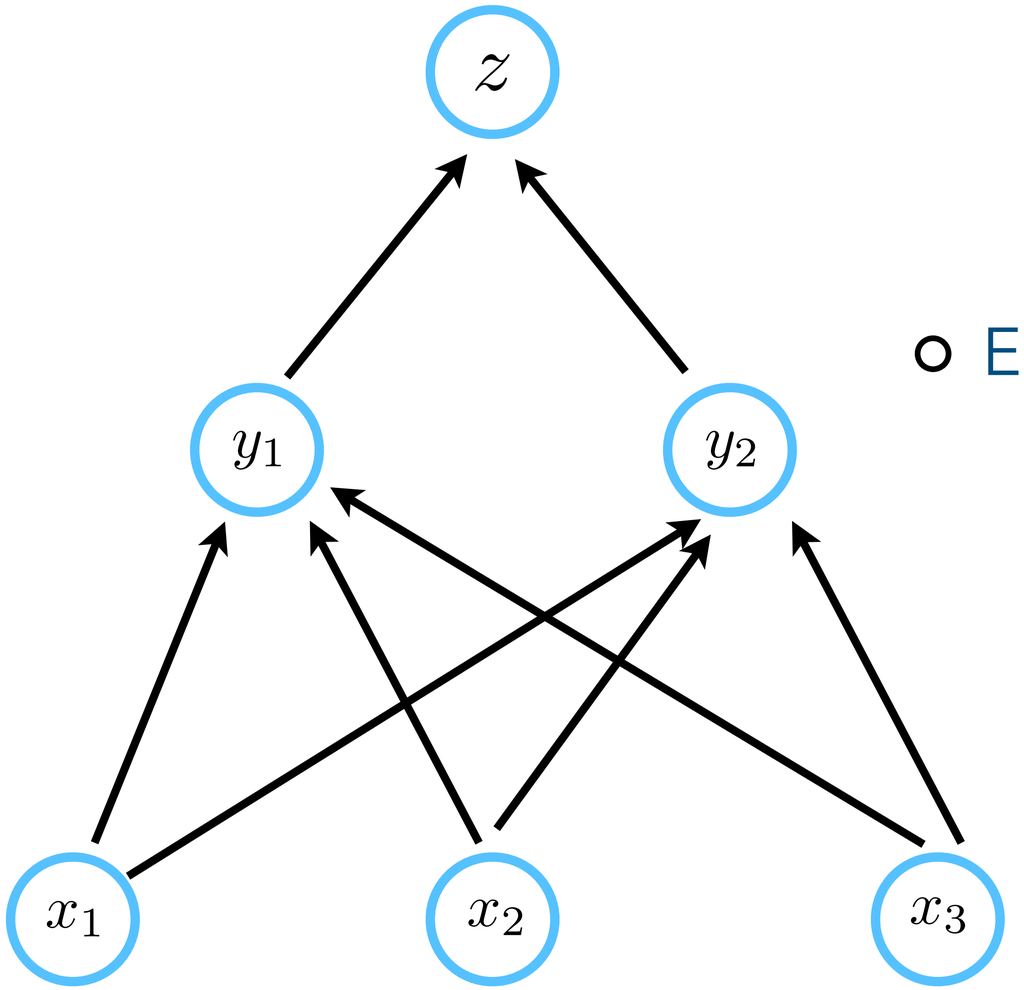
$$\varphi_1(\texttt{input}, W) = \sigma(W_1 \cdot x_i)$$
$$\varphi_2(\texttt{input}, W) = \sigma(W_2 \cdot \varphi_2(\cdot))$$



(backward pass on modules!)

$$\begin{aligned}
\nabla f_i(x) &= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial W} \\
&= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial \texttt{softmax}(\cdot)} \cdot \frac{\partial \texttt{softmax}(\cdot)}{\partial W} \\
&= \frac{\partial \ell(y_i, \widehat{y}_i)}{\partial \texttt{softmax}(\cdot)} \cdot \frac{\partial \texttt{softmax}(\cdot)}{\partial \varphi(\cdot, W_2)} \cdot \frac{\partial \varphi(\cdot, W_2)}{\partial W} \\
&= \cdots
\end{aligned}$$

(chain rule of derivatives)



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

- E.g.,

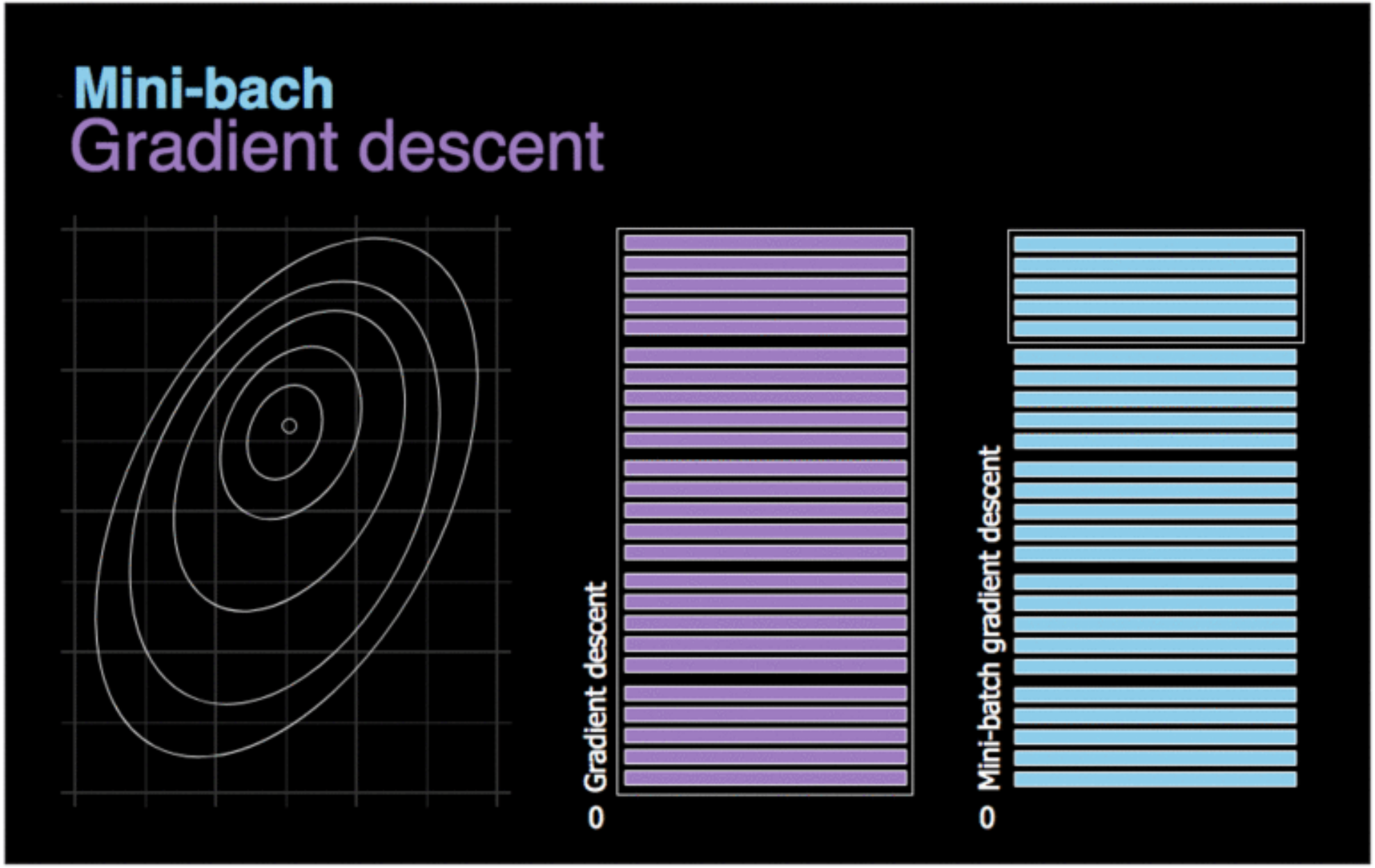
$$\frac{\partial z}{\partial x_3} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_3} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_3}$$

(Follow all relevant paths)

Backpropagation = Gradient descent

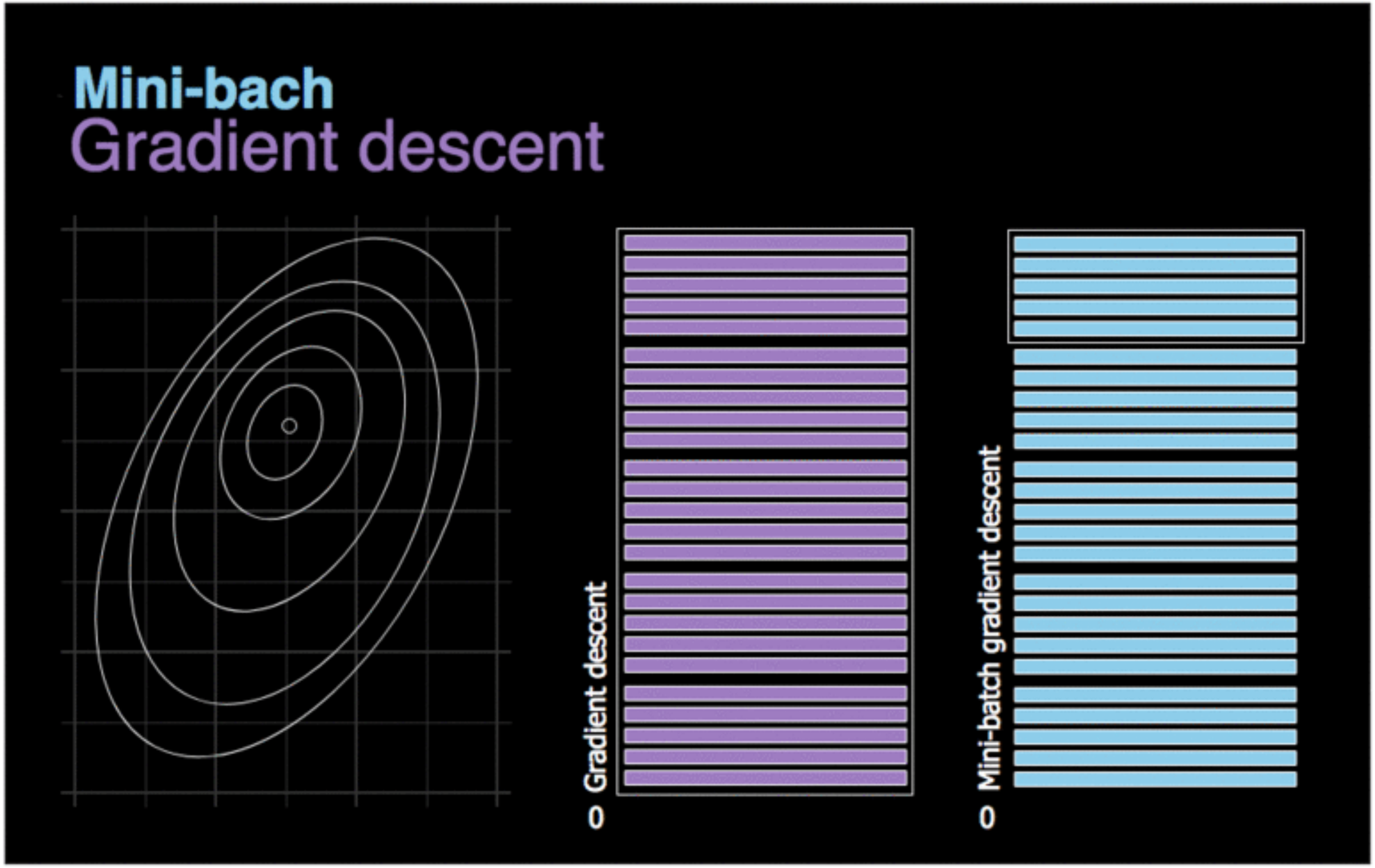
(Just done efficiently on graphs, without redoing calculations)

Stochastic gradient descent



Tribute to
Piotr Skalski

Stochastic gradient descent



Tribute to
Piotr Skalski

Overview

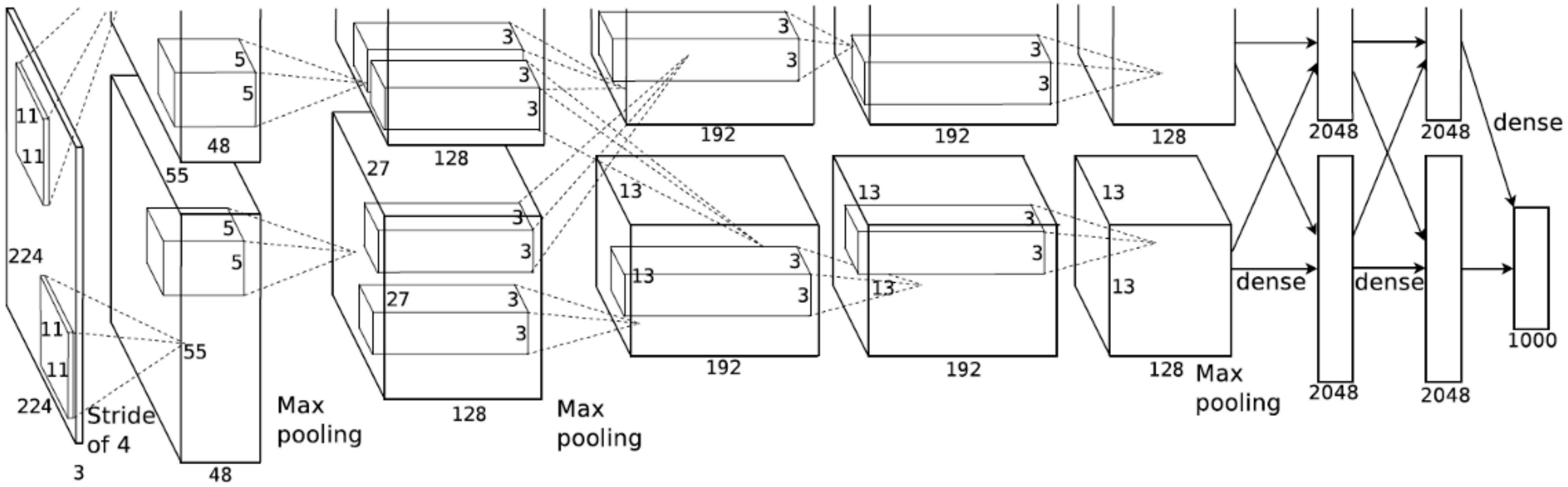
- Introduction to neural networks
- Feedforward / Fully-connected neural networks or MLPs
- Convolutional neural networks (CNNs)
- Recurrent neural networks (RNNs)
- Optimization in neural network training

Overview

- Convolutional neural networks (CNNs)

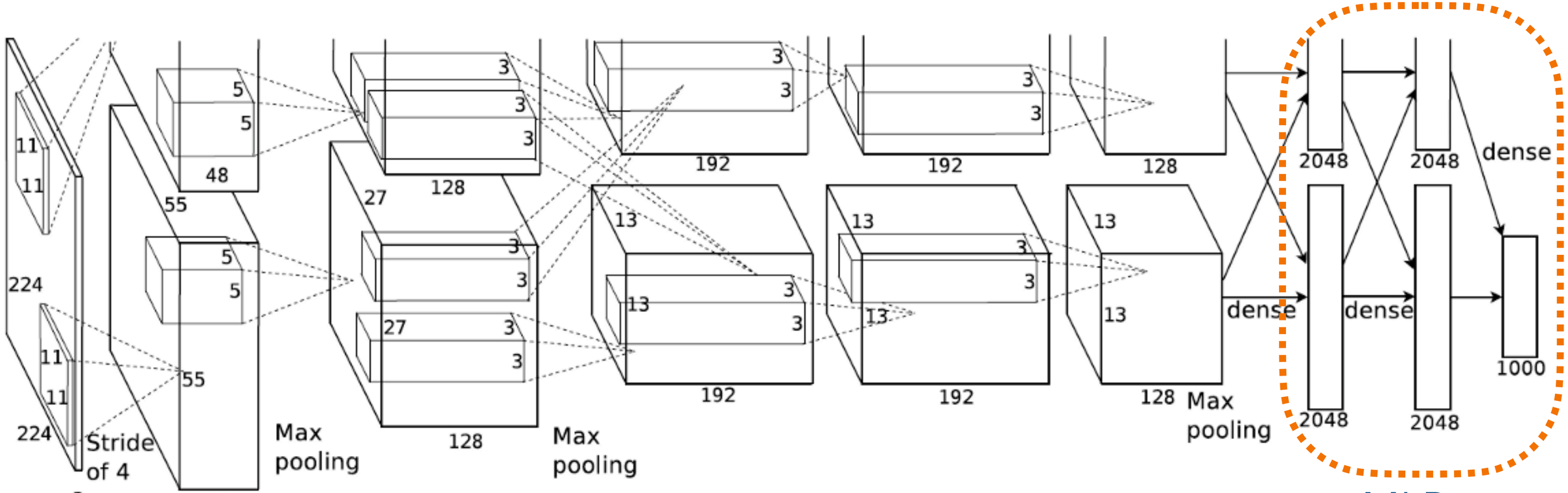
Spoiler alert

AlexNet - Winner of ImageNet 2012 competition



Spoiler alert

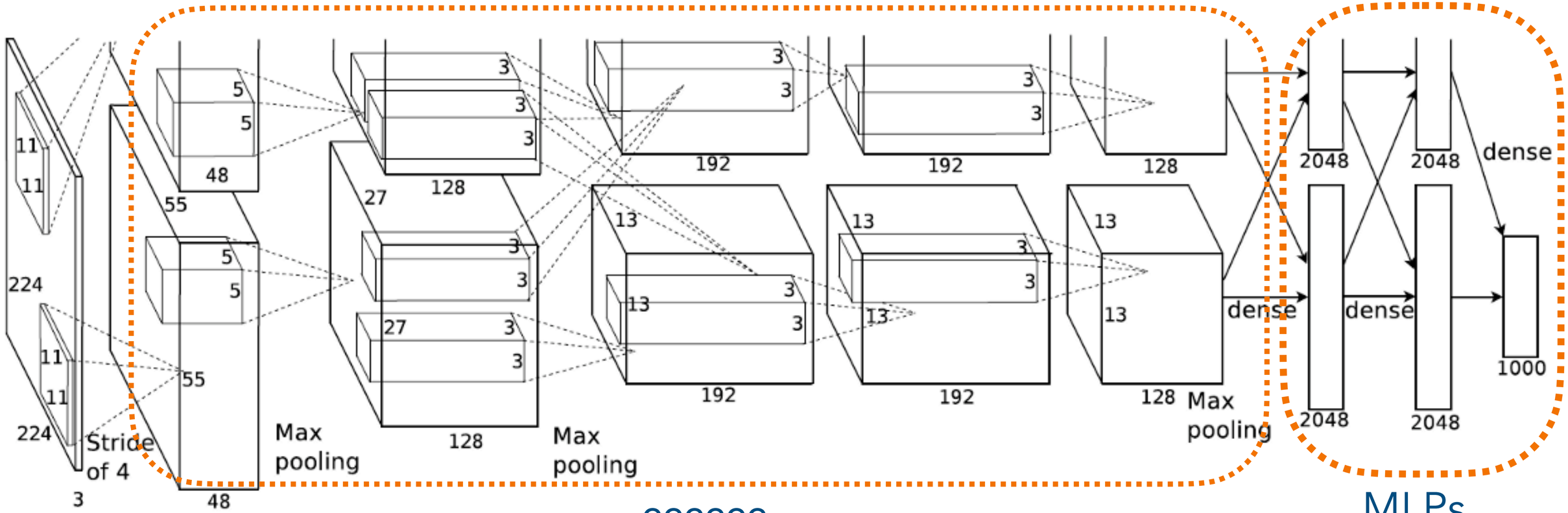
AlexNet - Winner of ImageNet 2012 competition



MLPs

Spoiler alert

AlexNet - Winner of ImageNet 2012 competition

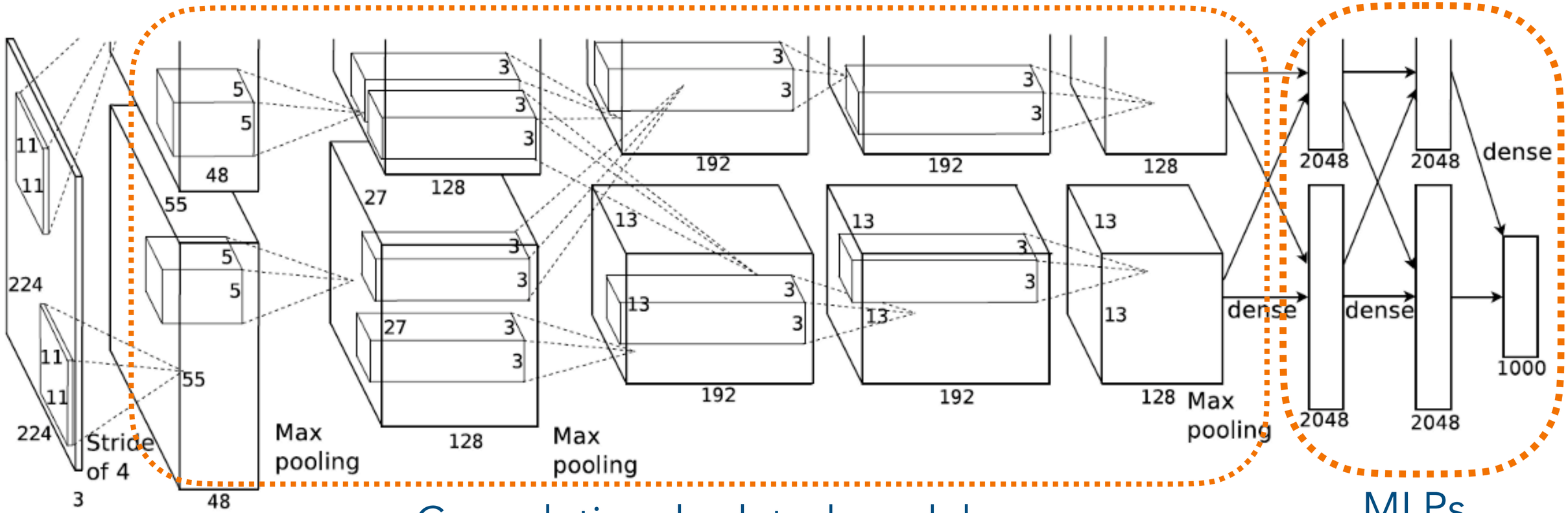


??????

MLPs

Spoiler alert

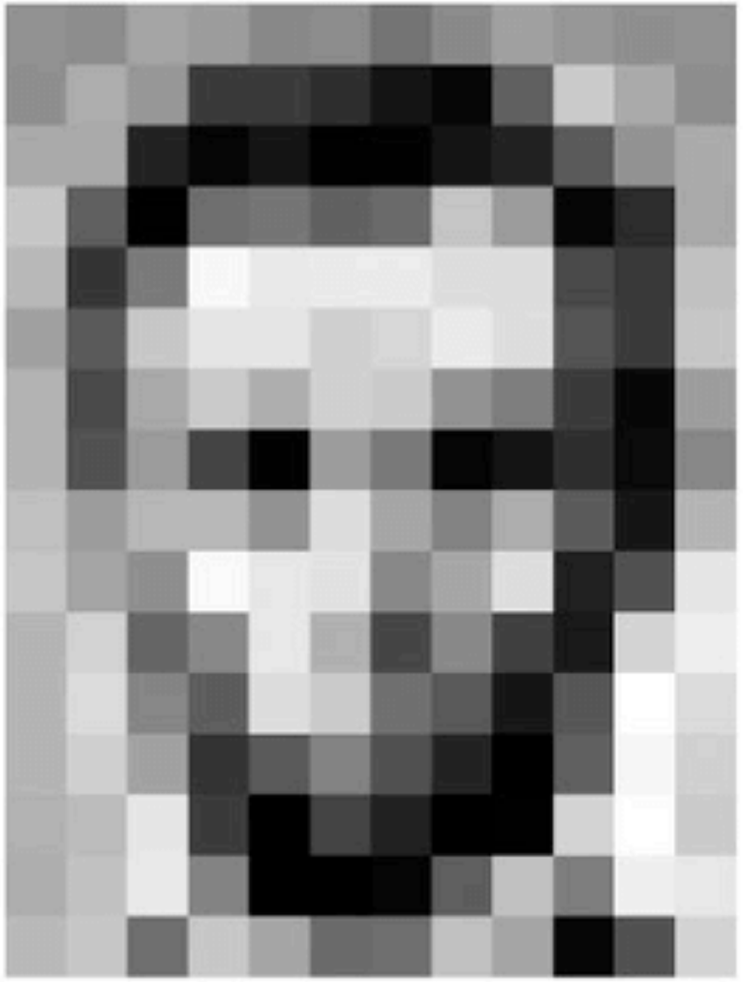
AlexNet - Winner of ImageNet 2012 competition



Convolutional-related modules

MLPs

What is an image for a computer?



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Filters/Kernels

- Fairly old concept of extracting information from images

Filters/Kernels

- Fairly old concept of extracting information from images
- Example: Sobel filters

Filters/Kernels

- Fairly old concept of extracting information from images
- Example: Sobel filters

Horizontal

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Vertical

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Filters/Kernels

- Fairly old concept of extracting information from images
- Example: Sobel filters

Horizontal

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Vertical

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Applying Sobel kernels on original image = edge detection

Filters/Kernels

- Fairly old concept of extracting information from images
- Example: Sobel filters

Horizontal

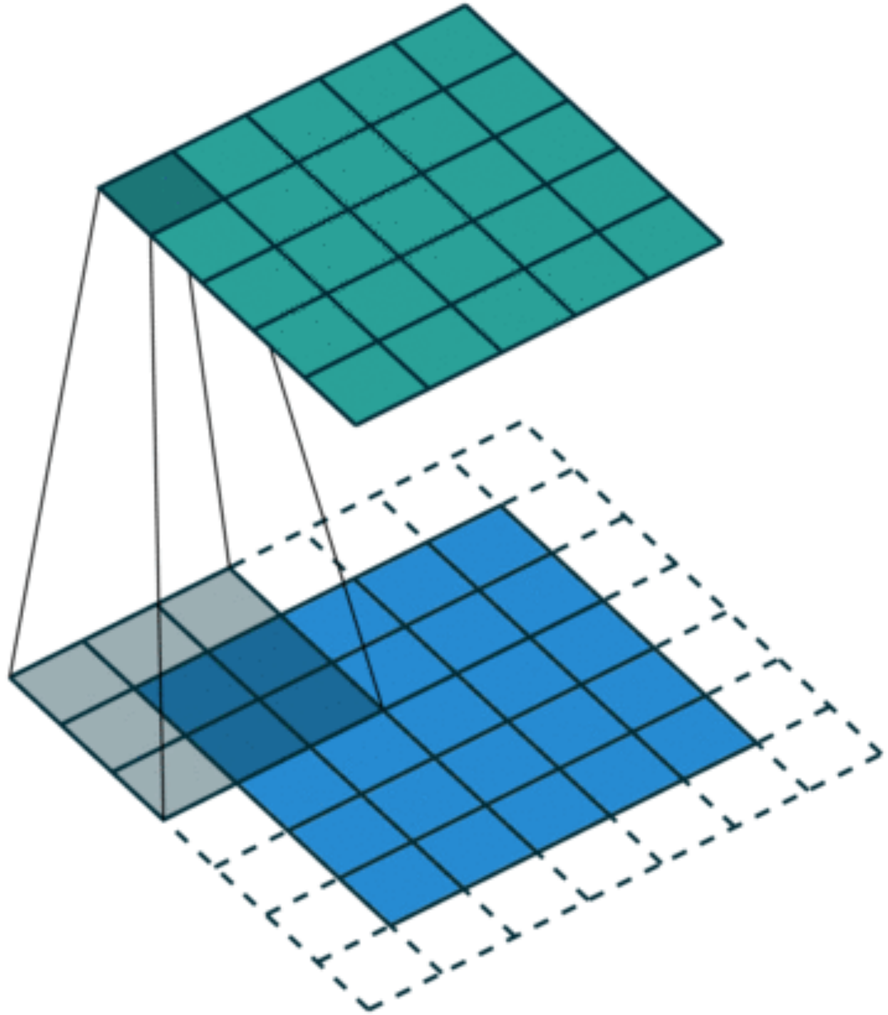
$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Vertical

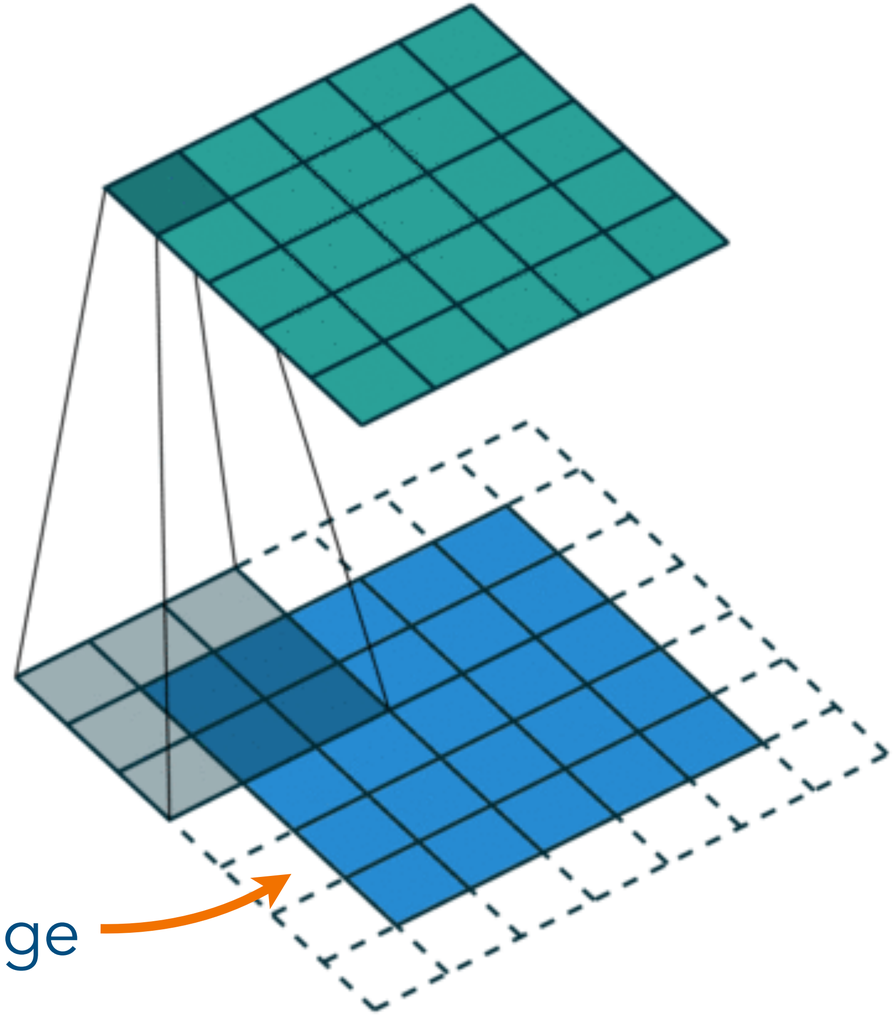
$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Applying Sobel kernels on original image = edge detection

Convolution operation: Basic operation

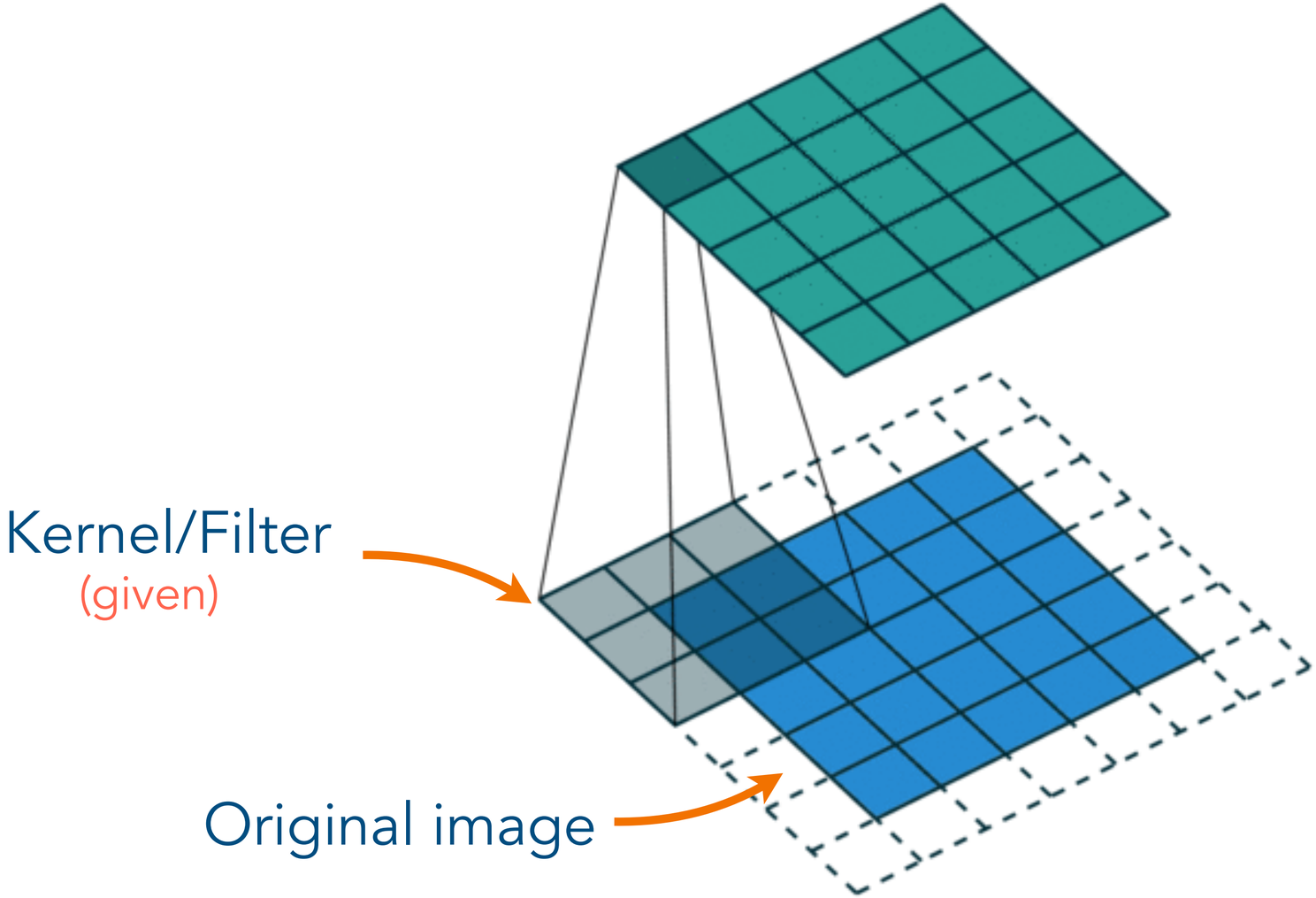


Convolution operation: Basic operation

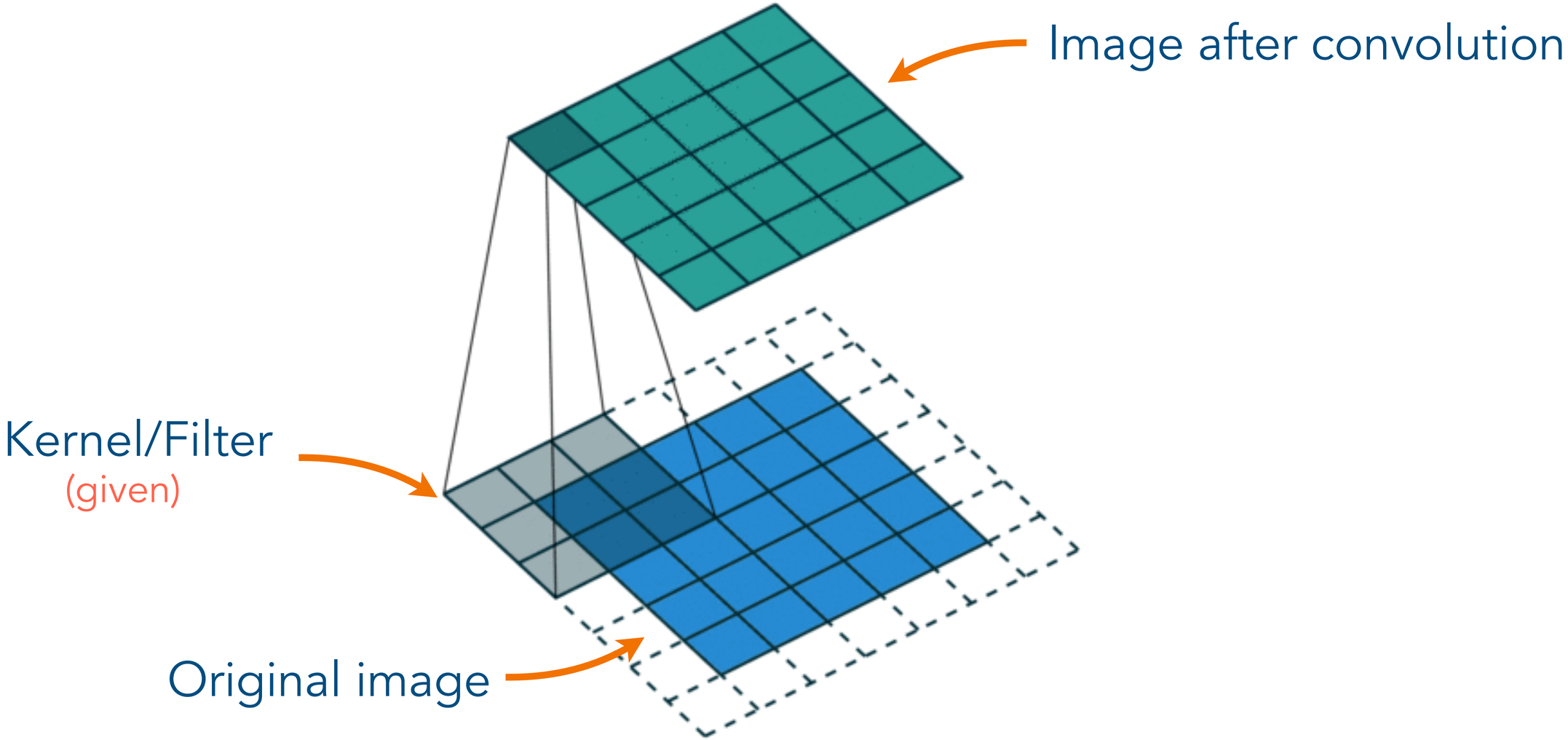


Original image

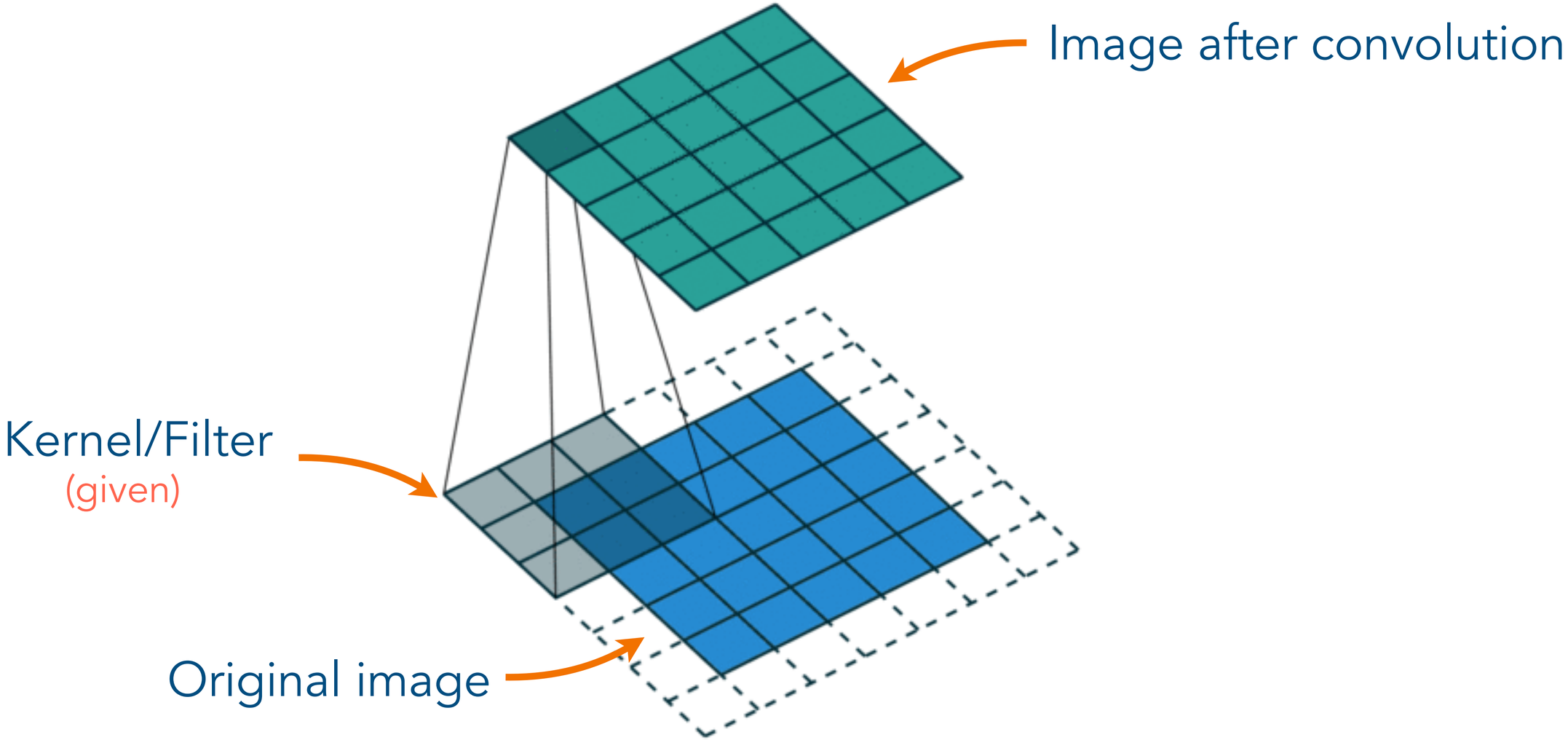
Convolution operation: Basic operation



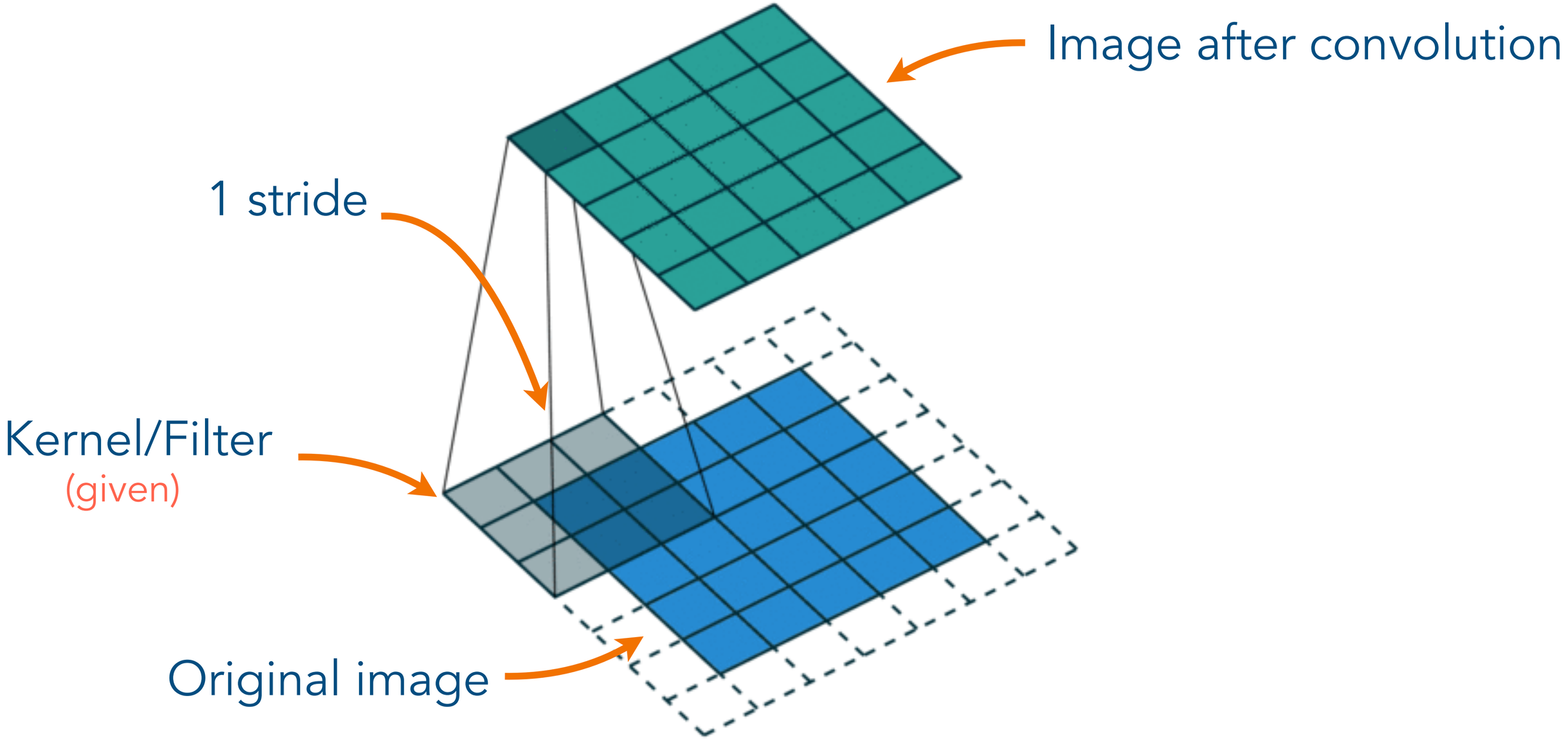
Convolution operation: Basic operation



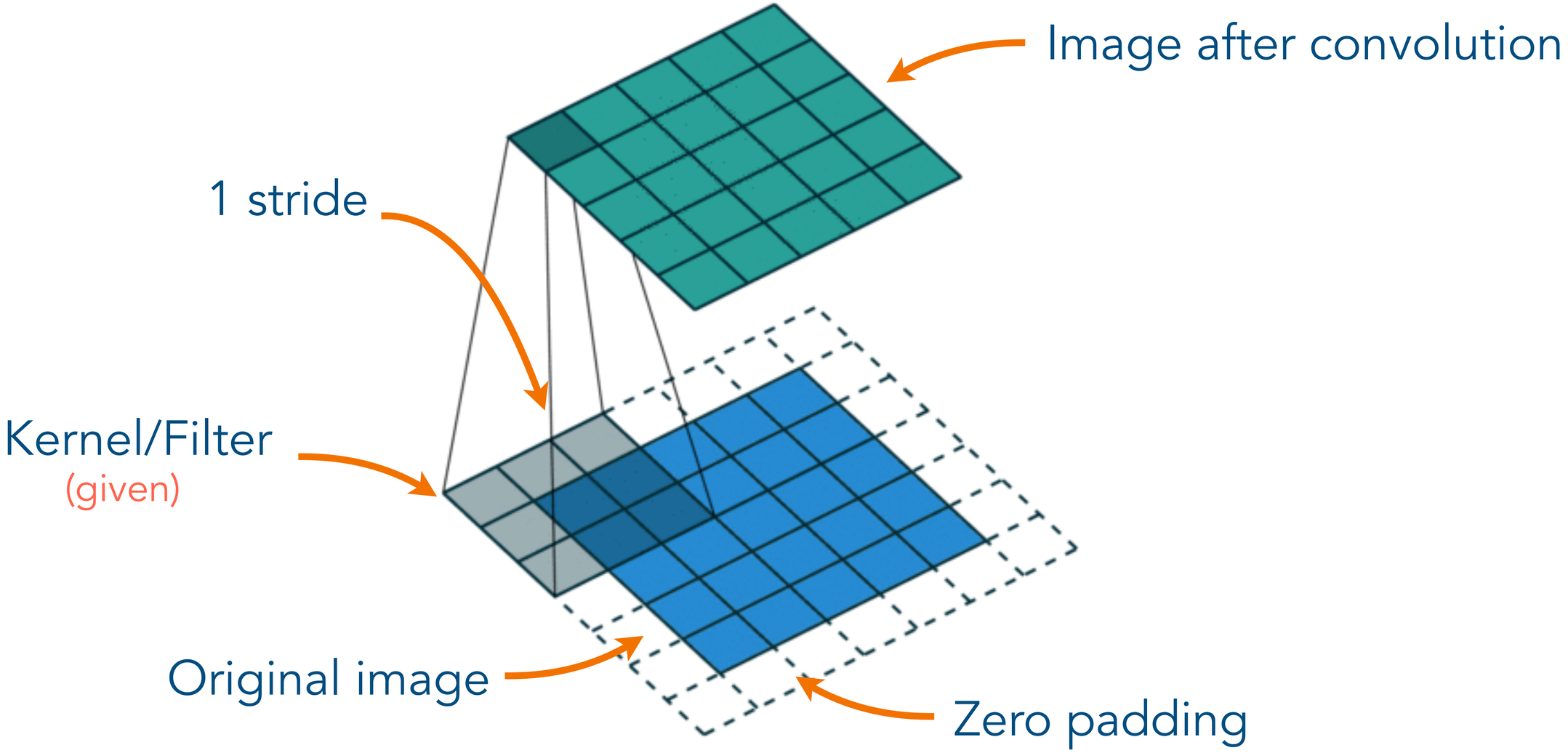
Convolution operation: Basic operation



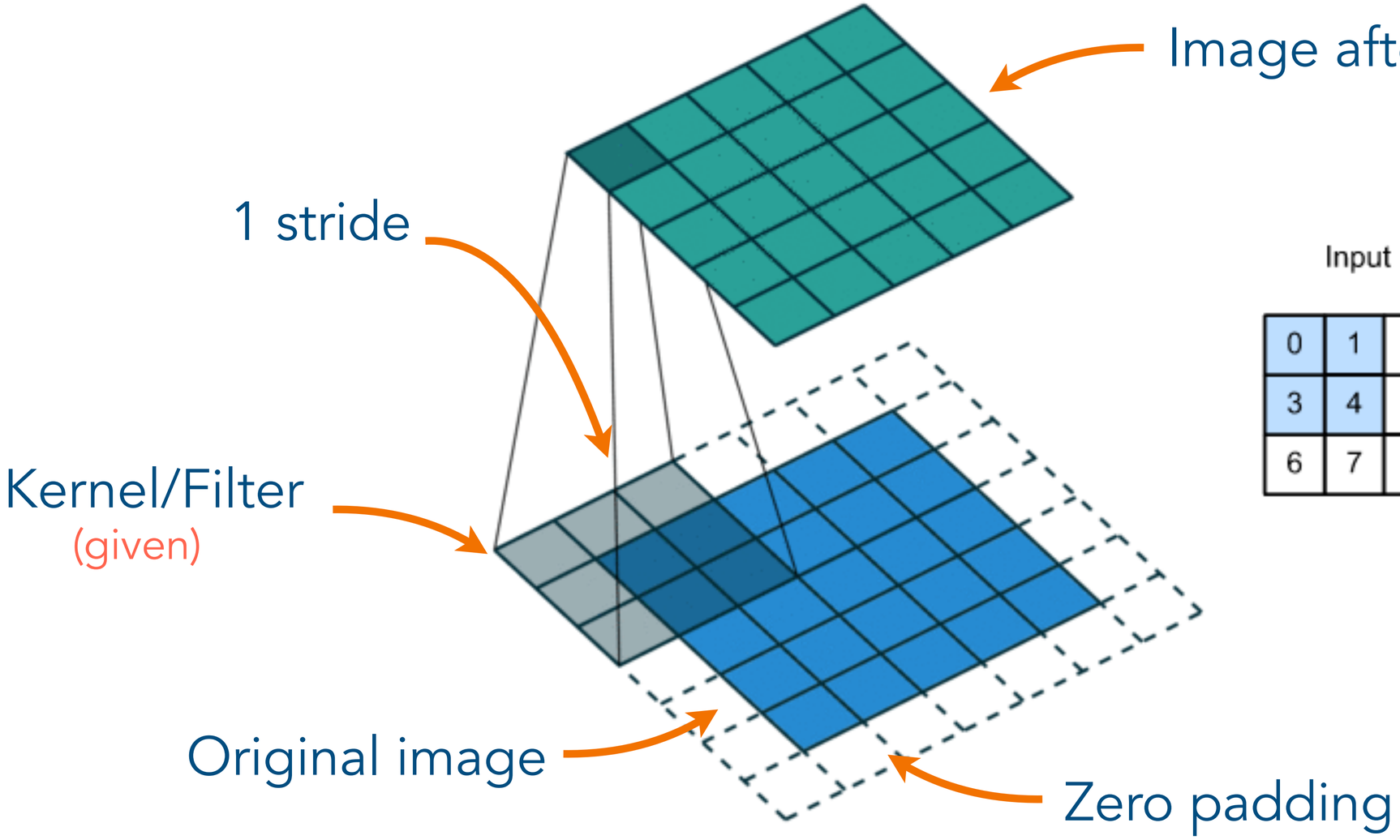
Convolution operation: Basic operation



Convolution operation: Basic operation

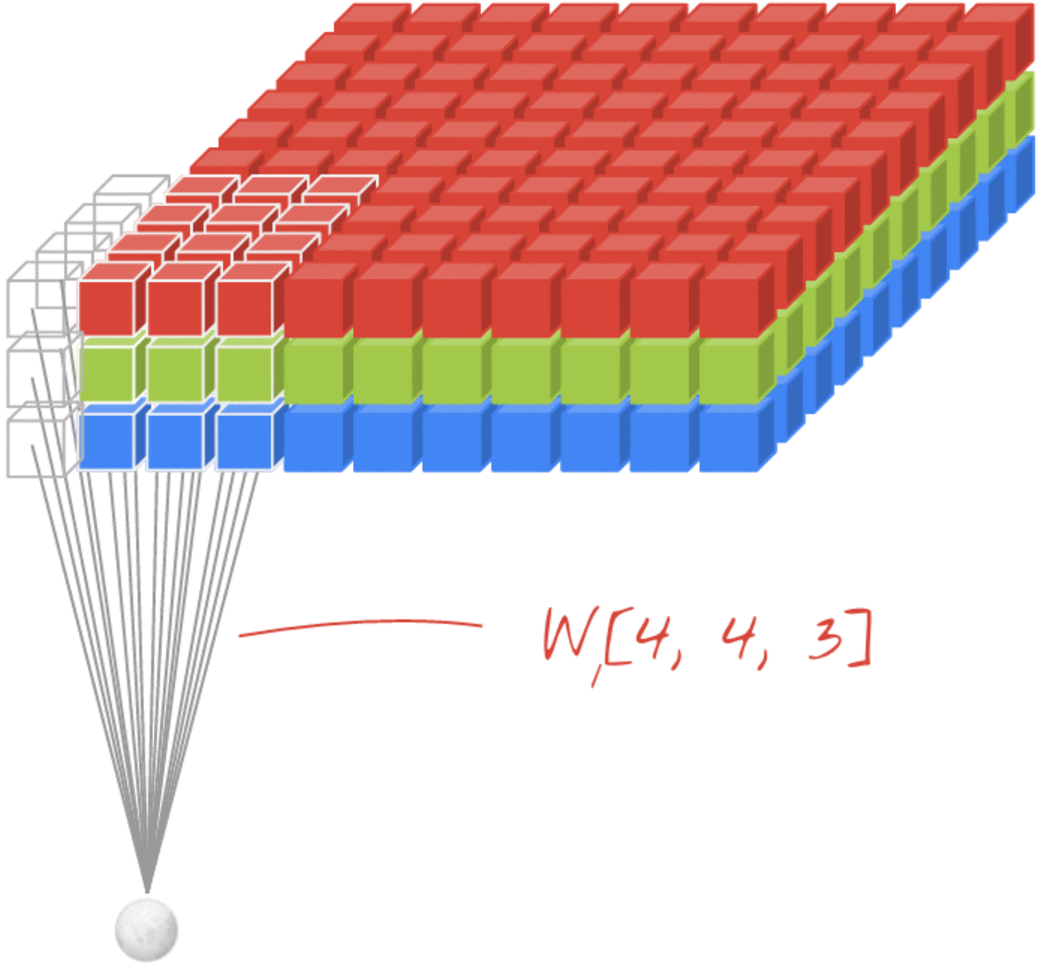


Convolution operation: Basic operation

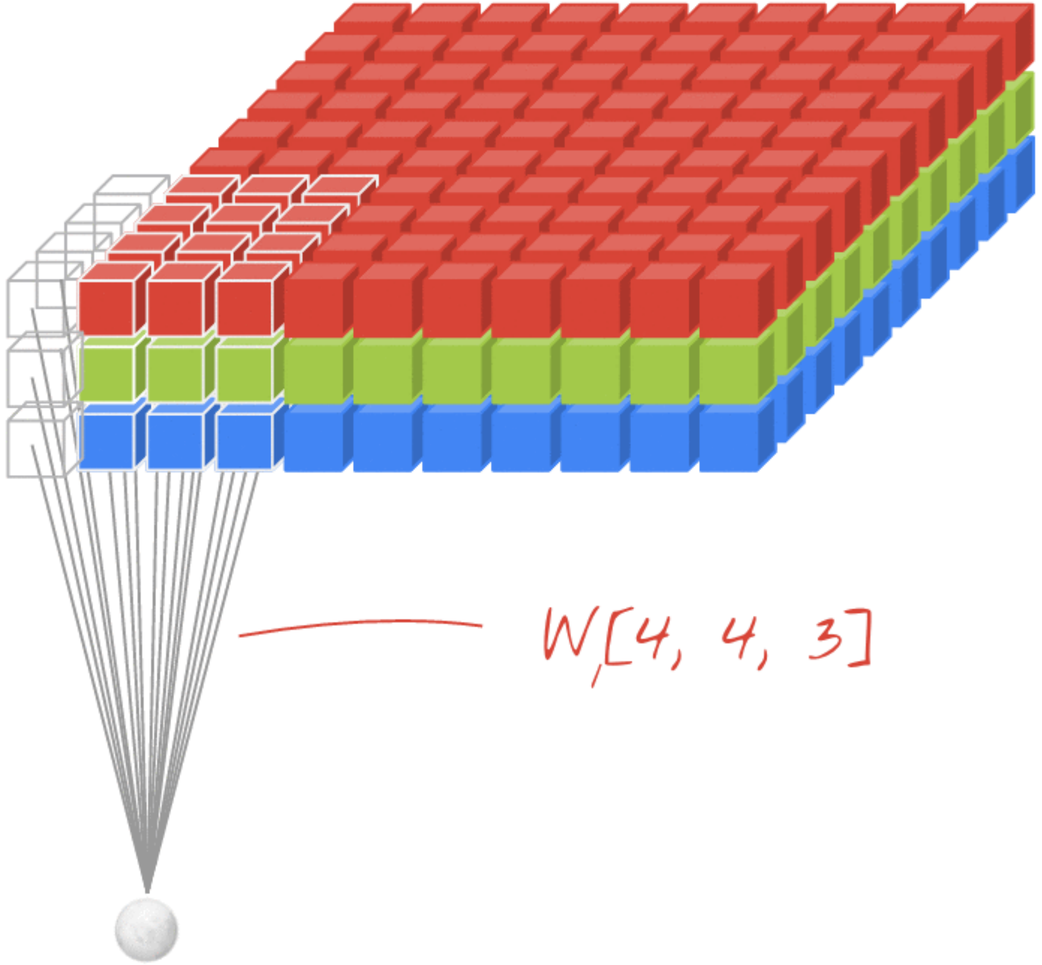


Input				Kernel				Output		
0	1	2	*	0	1	=		19	25	
3	4	5		2	3			37	43	
6	7	8								

Convolution operation: Channels and multiple filters



Convolution operation: Channels and multiple filters



Convolution operation

- Fairly old concept of extracting information from images
- Example: Sobel filters

Horizontal

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Vertical

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Applying Sobel kernels on original image = edge detection

Convolution operation

"Handcrafted filters/kernels"
(Sobel, Gaussian, smoothness, Gabor, etc.)

- Fairly old concept of extracting information from images
- Example: Sobel filters

Horizontal

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Vertical

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Applying Sobel kernels on original image = edge detection

Convolution operation

"Handcrafted filters/kernels"
(Sobel, Gaussian, smoothness, Gabor, etc.)

- Fairly old concept of extracting information from images
- Example: Sobel filters

Are they optimal for all computer vision applications?

Horizontal

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Vertical

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Applying Sobel kernels on original image = edge detection

Learnable kernels

Learnable kernels

- Instead of handcrafted kernels..

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Learnable kernels

- Instead of handcrafted kernels..

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

.. we try to learn one:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

..based on our application.

Learnable kernels

- Instead of handcrafted kernels..

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

.. we try to learn one:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

..based on our application.

- Convolution modules are fully differentiable!

Learnable kernels

- Instead of handcrafted kernels..

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

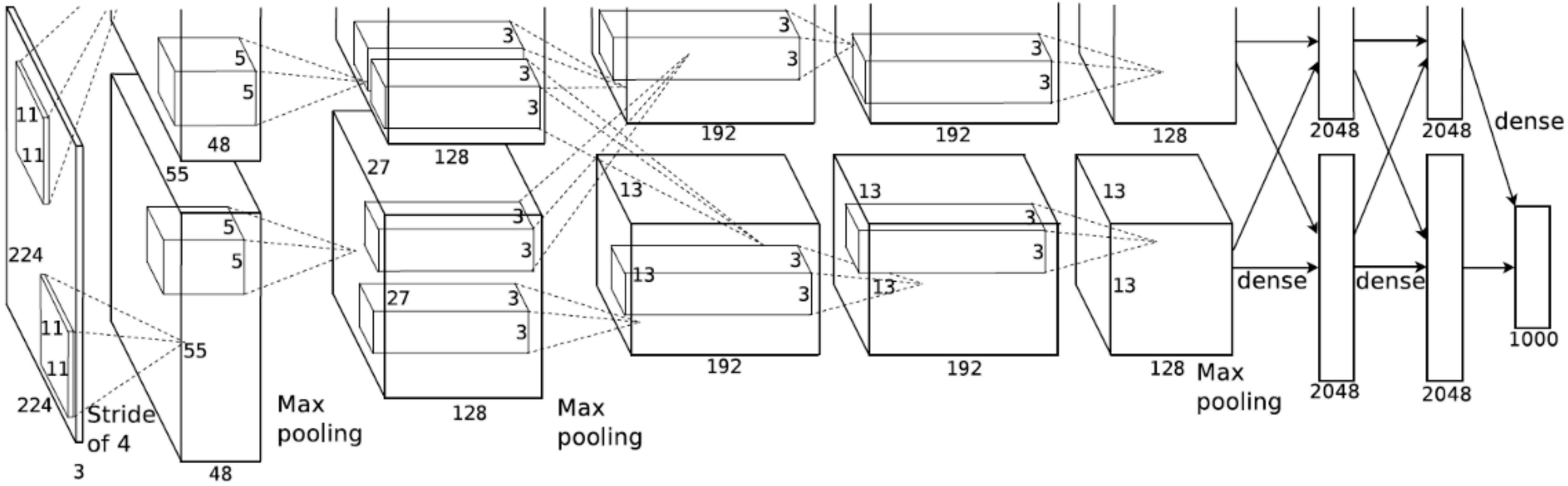
.. we try to learn one:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

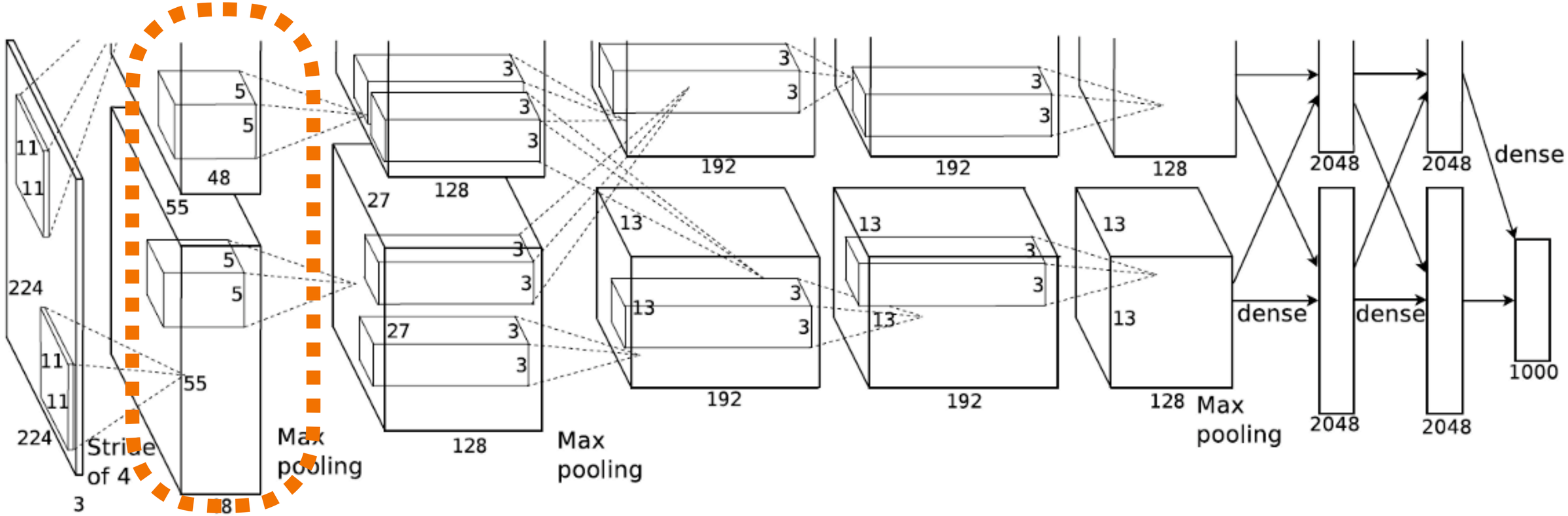
..based on our application.

- Convolution modules are fully differentiable!
- Good practices: Stride 1, 3x3 or 5x5 kernels work well

AlexNet

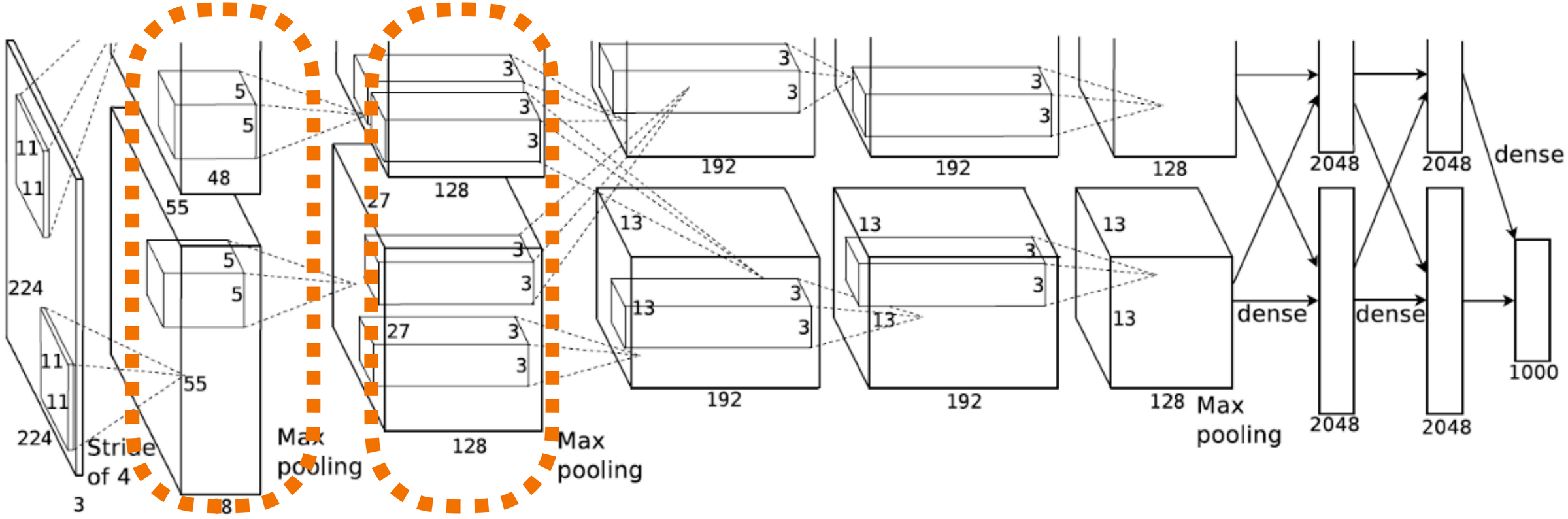

224
224
3
11
11
11
11
Stride
of 4
55
55
48
48
5
5
5
5
Max
pooling
27
27
128
128
3
3
3
3
3
3
3
3
3
3
Max
pooling
13
13
13
13
192
192
192
192
128
128
13
13
13
13
Max
pooling
dense
dense
dense
2048
2048
2048
2048
1000

AlexNet

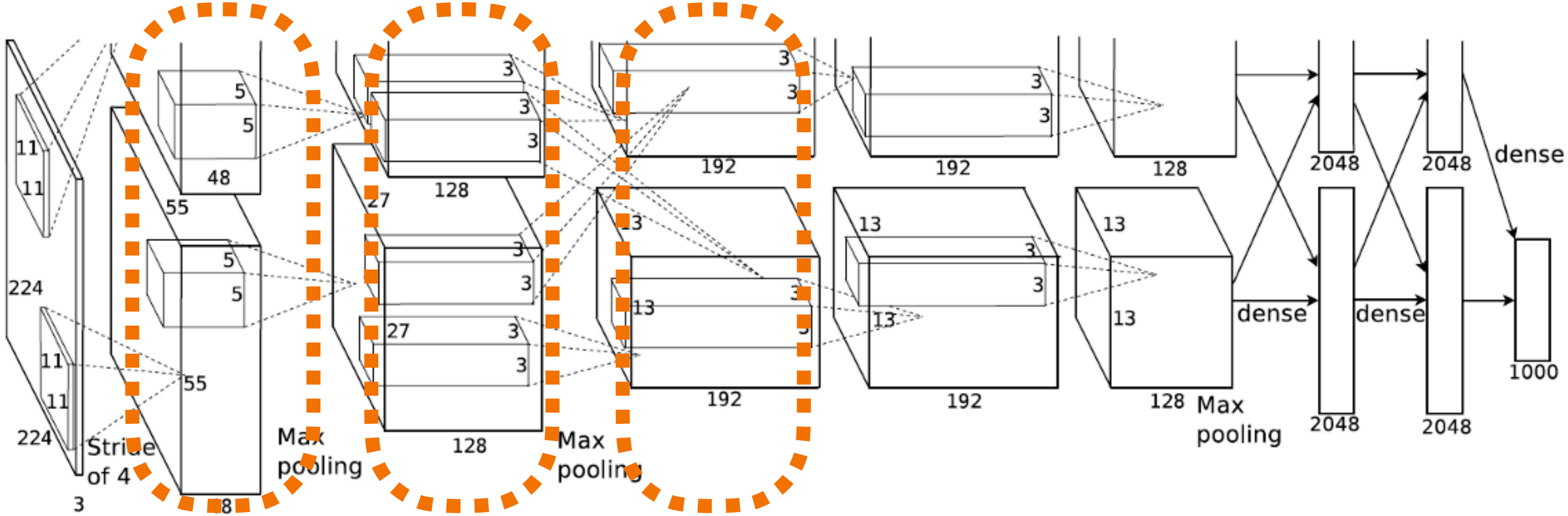


conv.
module 1

AlexNet


224
224
3
11
11
11
11
Stride
of 4
55
55
48
48
5
5
5
5
Max
pooling
27
27
128
128
3
3
3
3
3
3
3
3
Max
pooling
192
192
128
13
13
13
13
13
13
192
192
128
Max
pooling
dense
dense
dense
2048
2048
2048
2048
1000
conv.
module 1
conv.
module 2

AlexNet

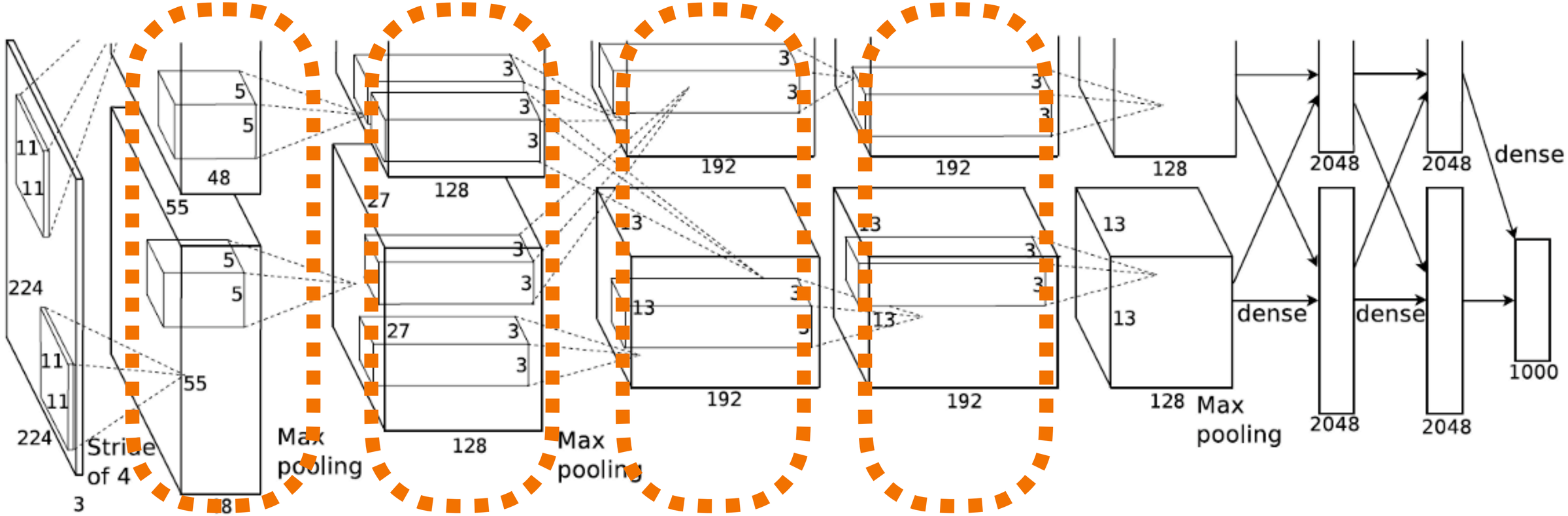


conv. module 1

conv. module 2

conv. module 3

AlexNet



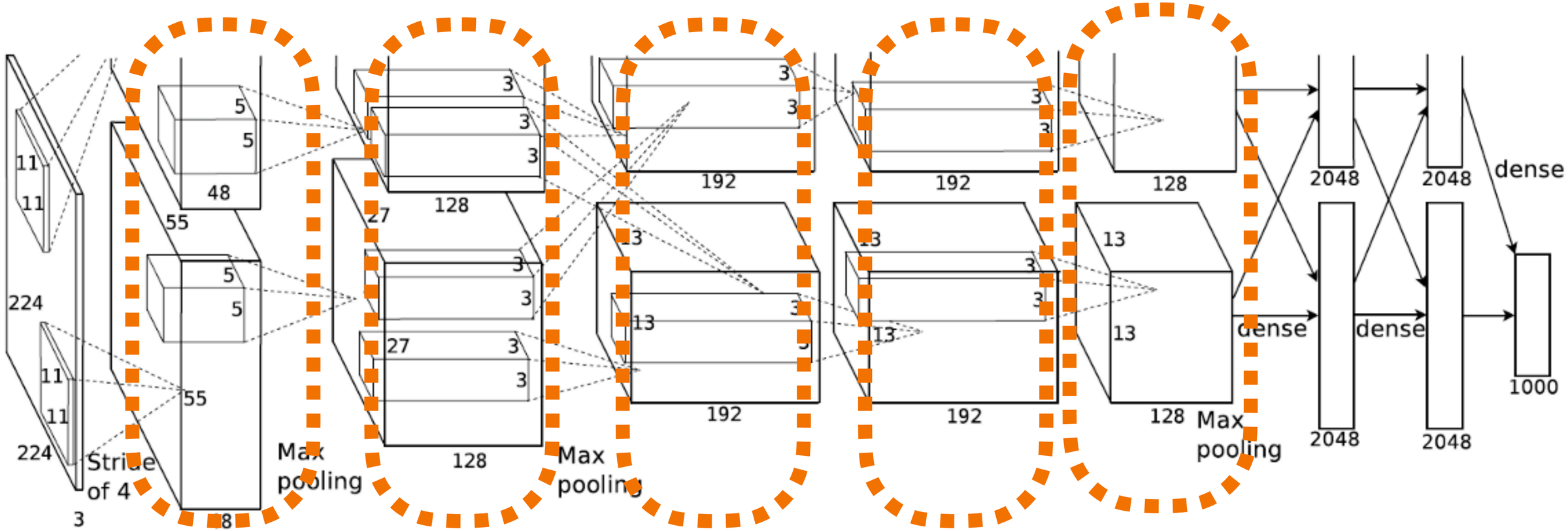
conv. module 1

conv. module 2

conv. module 3

conv. module 4

AlexNet



conv. module 1

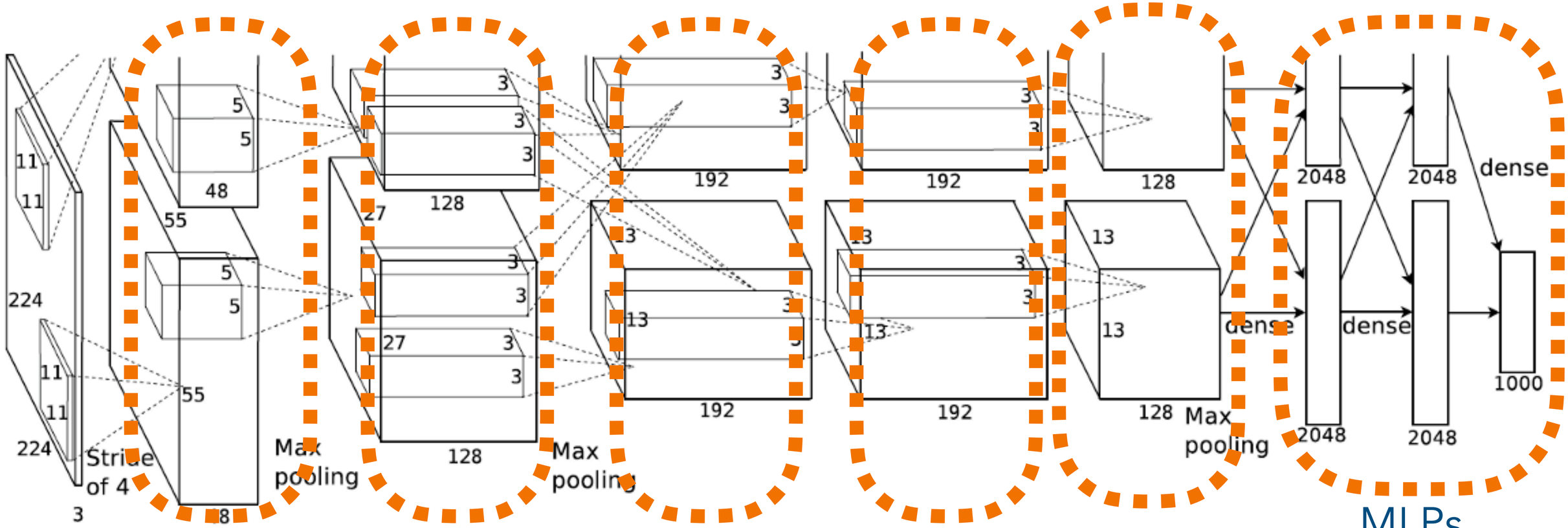
conv. module 2

conv. module 3

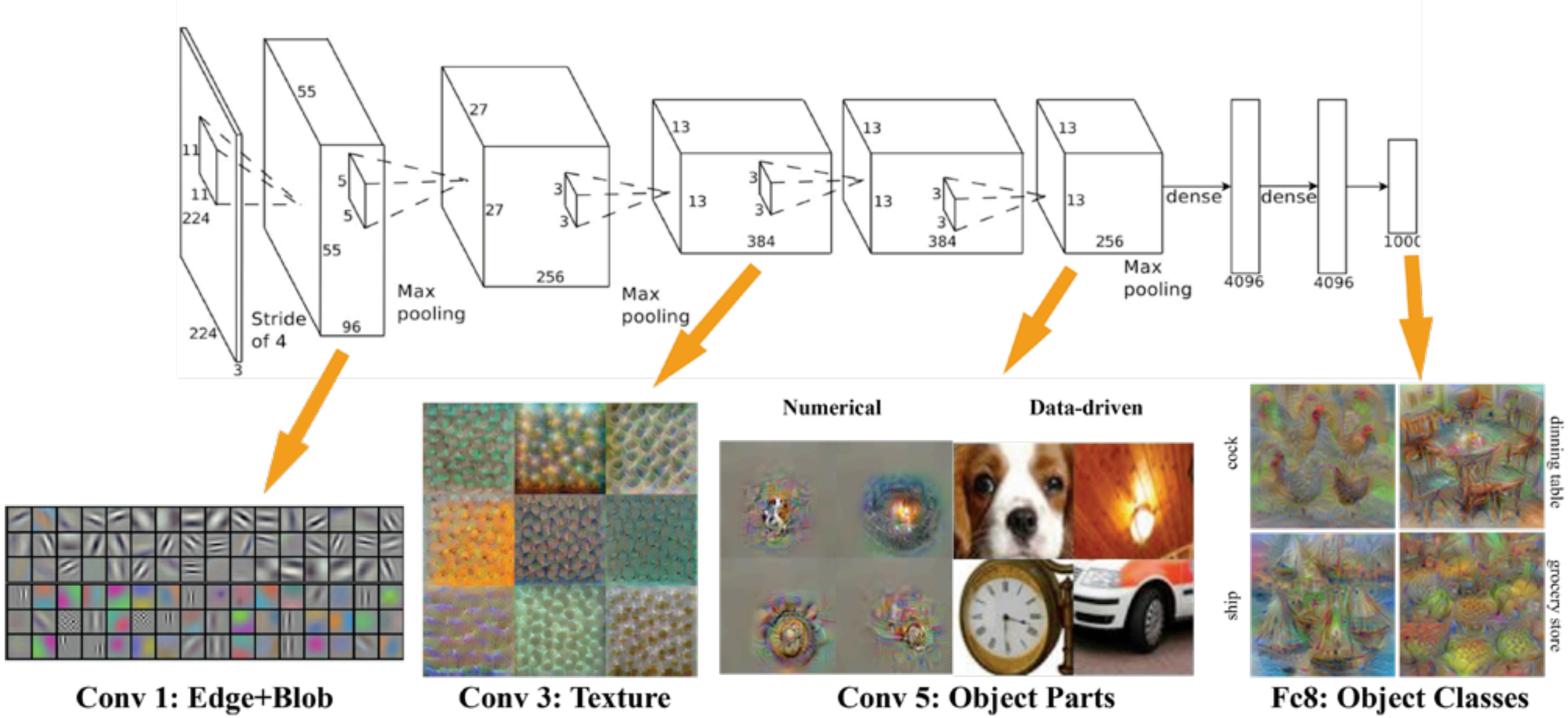
conv. module 4

conv. module 5

AlexNet


224
224
3
11
11
11
11
Stride
of 4
55
55
48
5
5
5
5
Max
pooling
27
27
128
128
3
3
3
3
3
3
3
3
Max
pooling
13
13
13
13
192
192
192
192
3
3
3
3
3
3
3
3
128
128
13
13
13
13
Max
pooling
dense
dense
dense
2048
2048
2048
2048
1000
conv.
module 1
conv.
module 2
conv.
module 3
conv.
module 4
conv.
module 5
MLPs

AlexNet



Hierarchical learning, feature maps

Tribute to mNeuron

Residual Networks (ResNets)

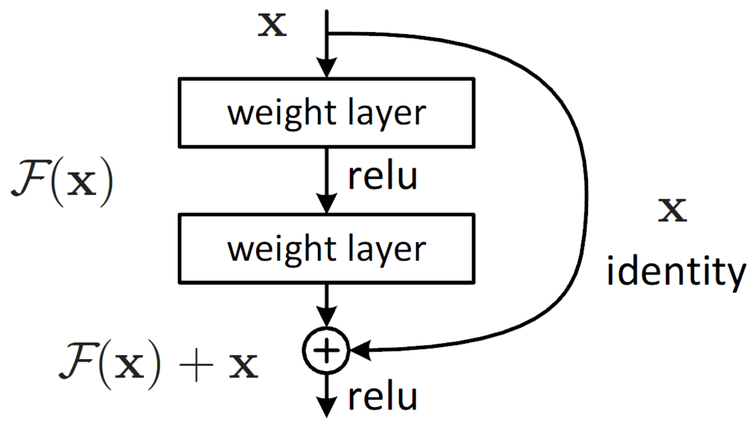
- Architecture that allows truly deep networks via skip connections

Residual Networks (ResNets)

- Architecture that allows truly deep networks via skip connections
- Skip connections provide the option to learn the identity map from layers to layers (could tackle vanishing gradients issue)

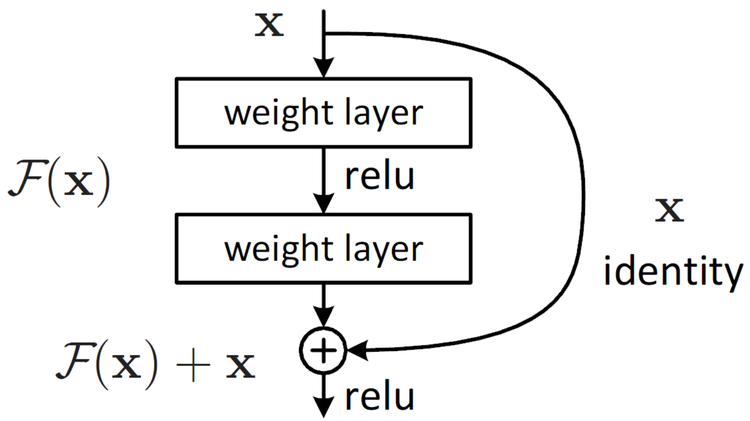
Residual Networks (ResNets)

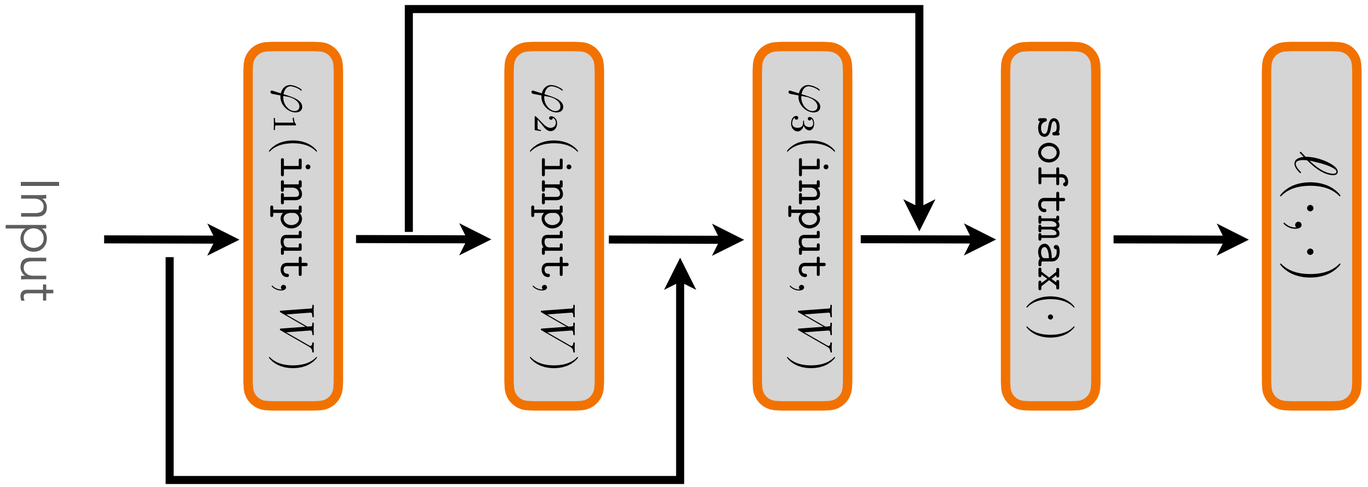
- Architecture that allows truly deep networks via skip connections
- Skip connections provide the option to learn the identity map from layers to layers (could tackle vanishing gradients issue)



Residual Networks (ResNets)

- Architecture that allows truly deep networks via skip connections
- Skip connections provide the option to learn the identity map from layers to layers (could tackle vanishing gradients issue)





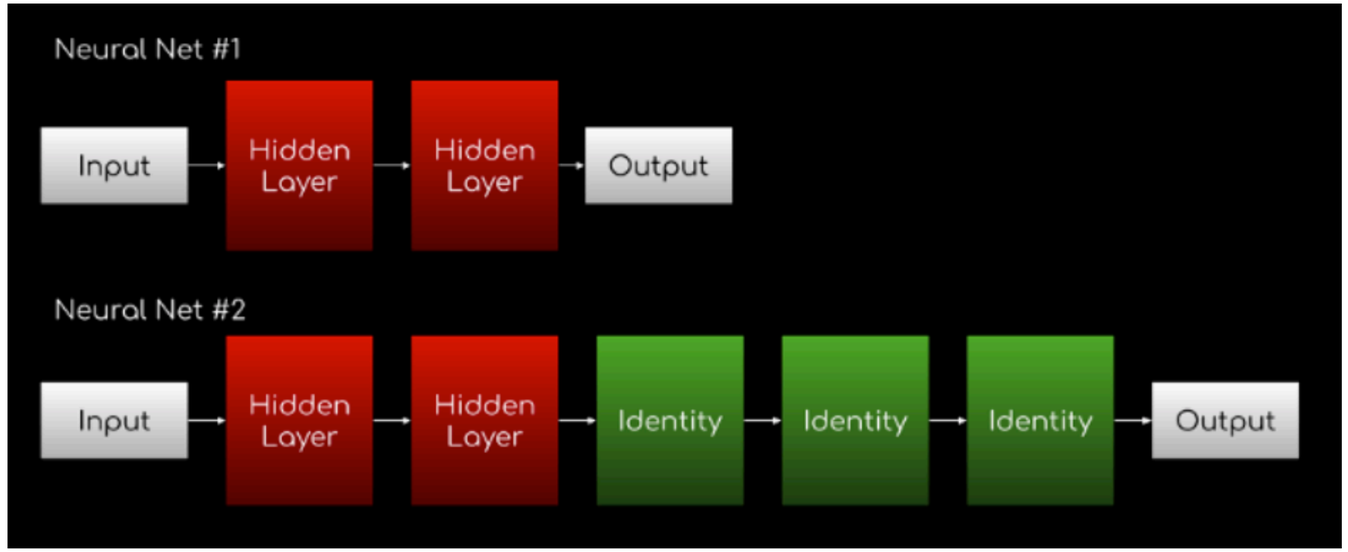
Residual Networks (ResNets)

- Basic idea: A deeper layer should be as good as a shallow layer, as it can learn the identity map: $\mathcal{F}(x) = x$

Tribute to
Michael Ye

Residual Networks (ResNets)

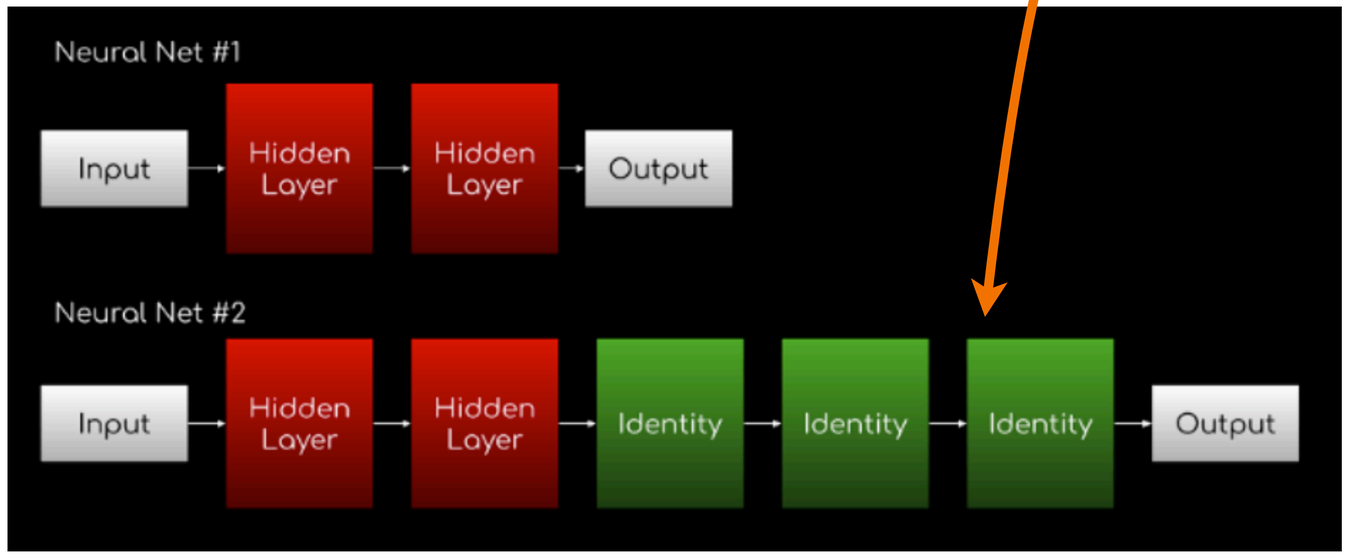
- Basic idea: A deeper layer should be as good as a shallow layer, as it can learn the identity map: $\mathcal{F}(x) = x$
- Example:



Tribute to
Michael Ye

Residual Networks (ResNets)

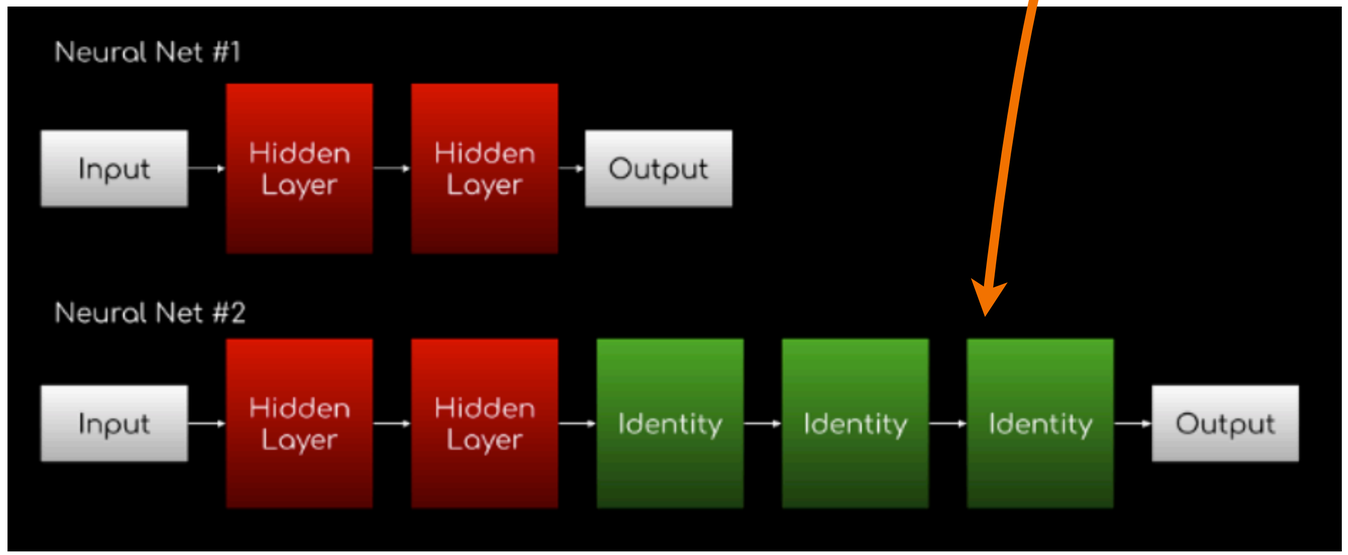
- Basic idea: A deeper layer should be as good as a shallow layer, as it can learn the identity map: $\mathcal{F}(x) = x$
- Example:



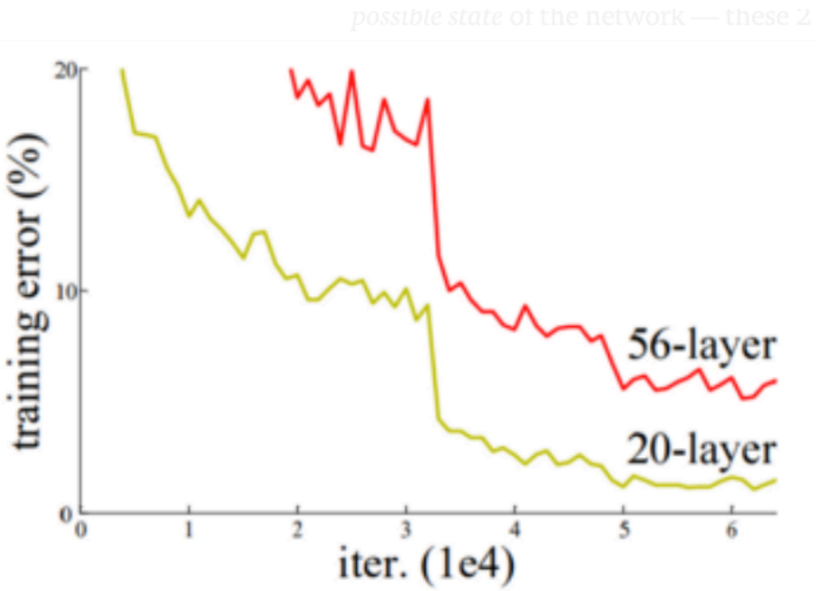
Tribute to
Michael Ye

Residual Networks (ResNets)

- Basic idea: A deeper layer should be as good as a shallow layer, as it can learn the identity map: $\mathcal{F}(x) = x$
- Example:



- Turns out that learning (enforcing) $\mathcal{F}(x) = x$ is hard:



Tribute to Michael Ye

Overview

- Introduction to neural networks
- Feedforward / Fully-connected neural networks or MLPs
- Convolutional neural networks (CNNs)
- Recurrent neural networks (RNNs)
- Optimization in neural network training

Overview

- Recurrent neural networks (RNNs)

The case of data sequences

- Examples:
  - Speech
  - Text
  - Videos
  - Stock market
  - Sequence of clicks on the web

$\vdots$

The case of data sequences

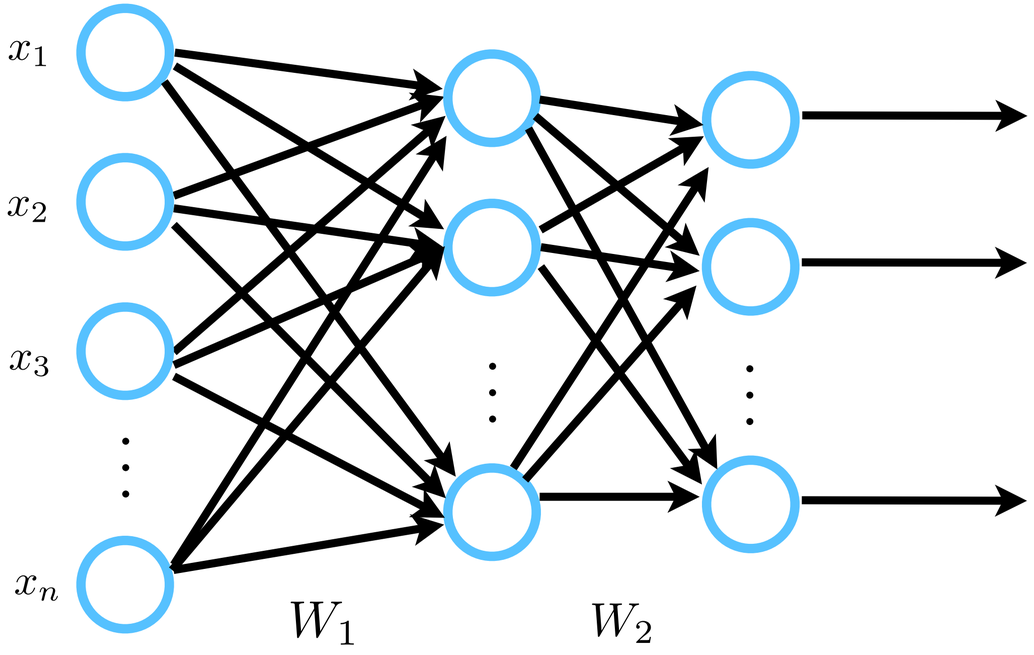
- Examples:
  - Speech
  - Text
  - Videos
  - Stock market
  - Sequence of clicks on the web

  ⋮

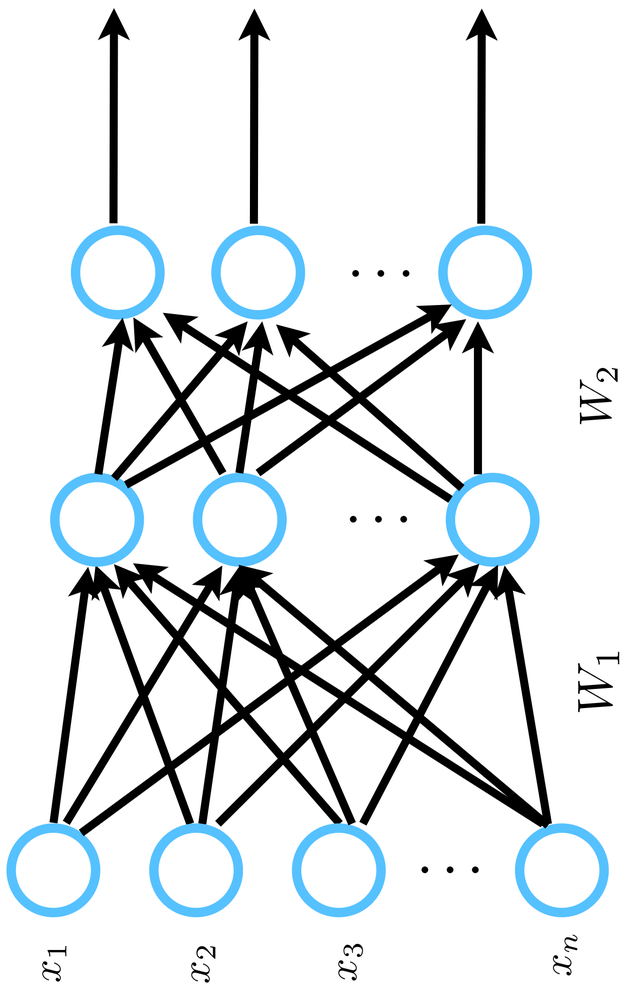
Can we exploit the fact that data "arrive" as a sequence of information?

- Issue: What is the dimension of the data?
- Issue: Is input data iid?

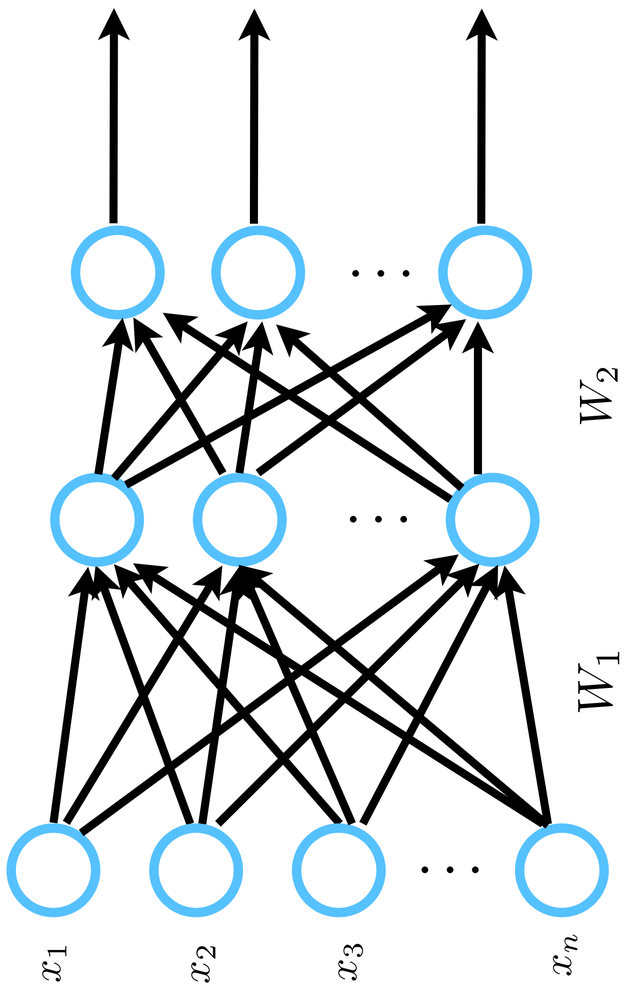
Thus far..

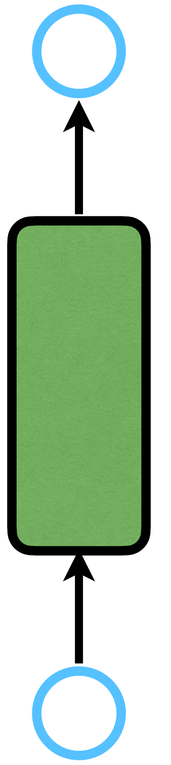


Thus far..

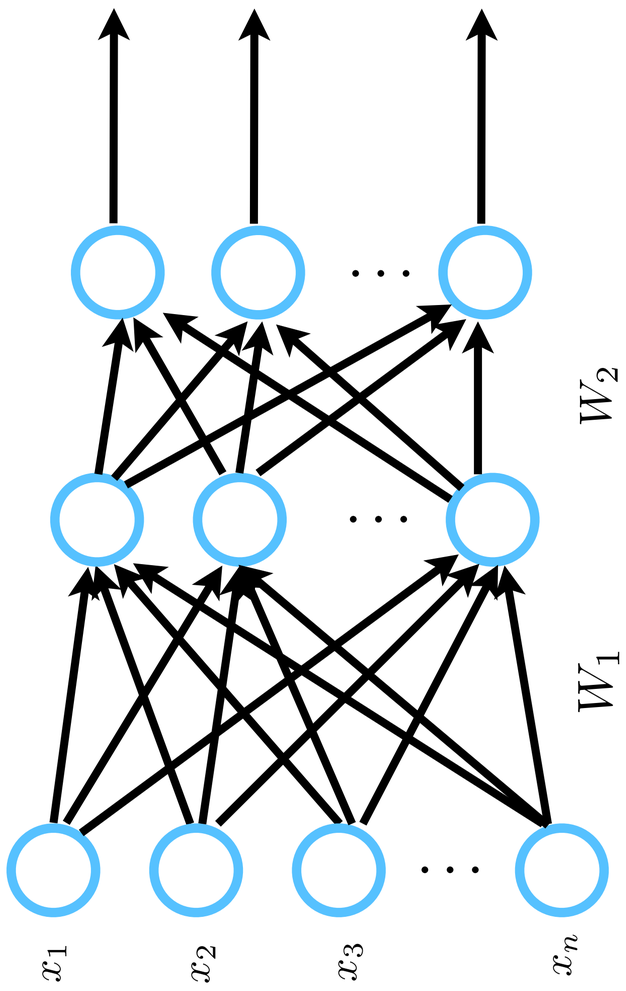


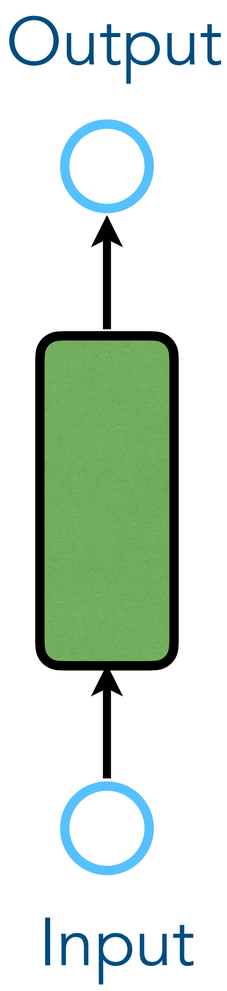
Thus far..



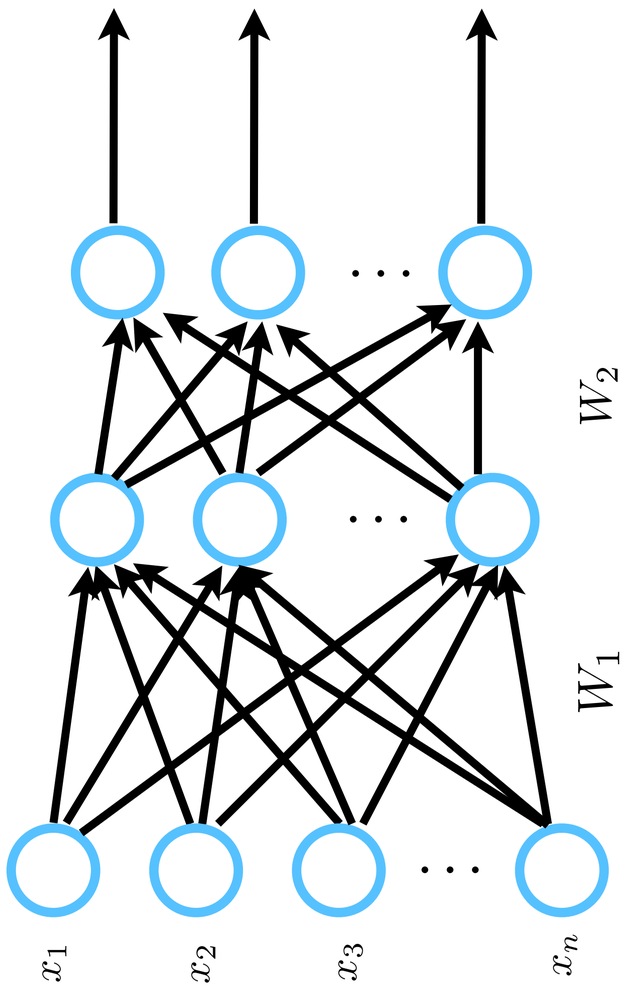


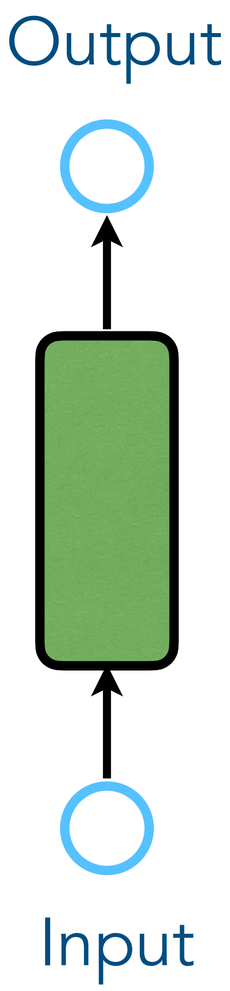
Thus far..





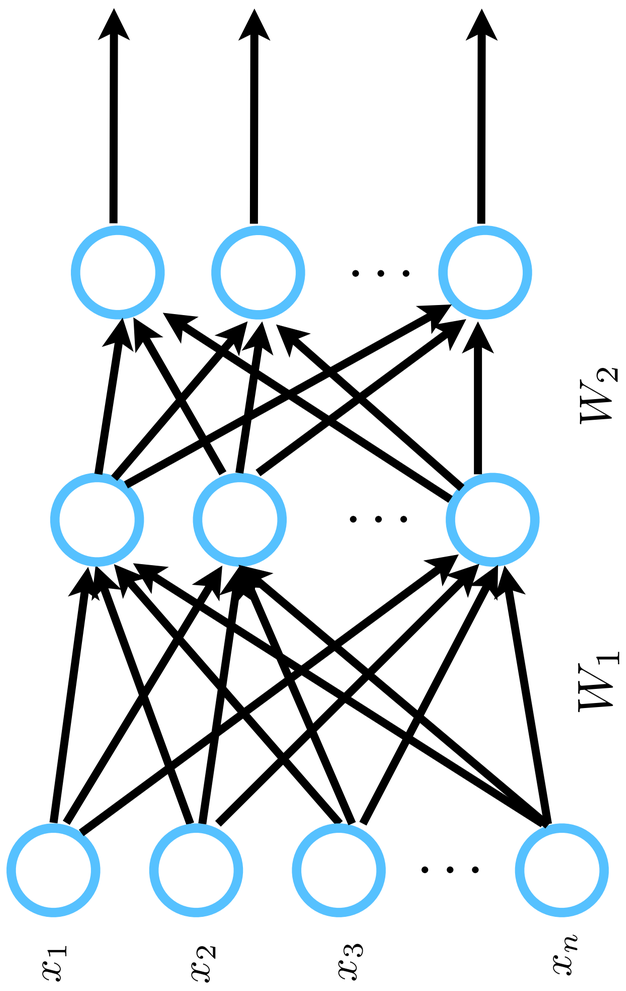
Thus far..

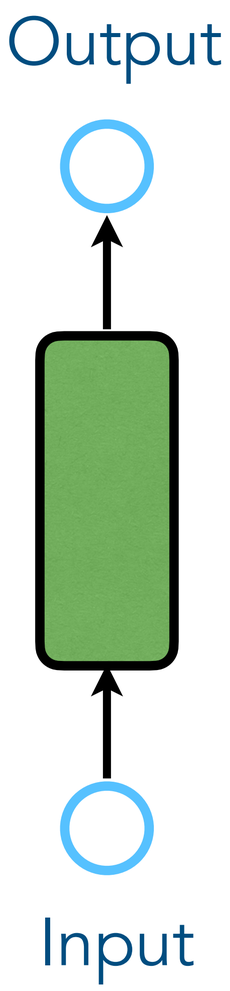




- One to one classification/ regression

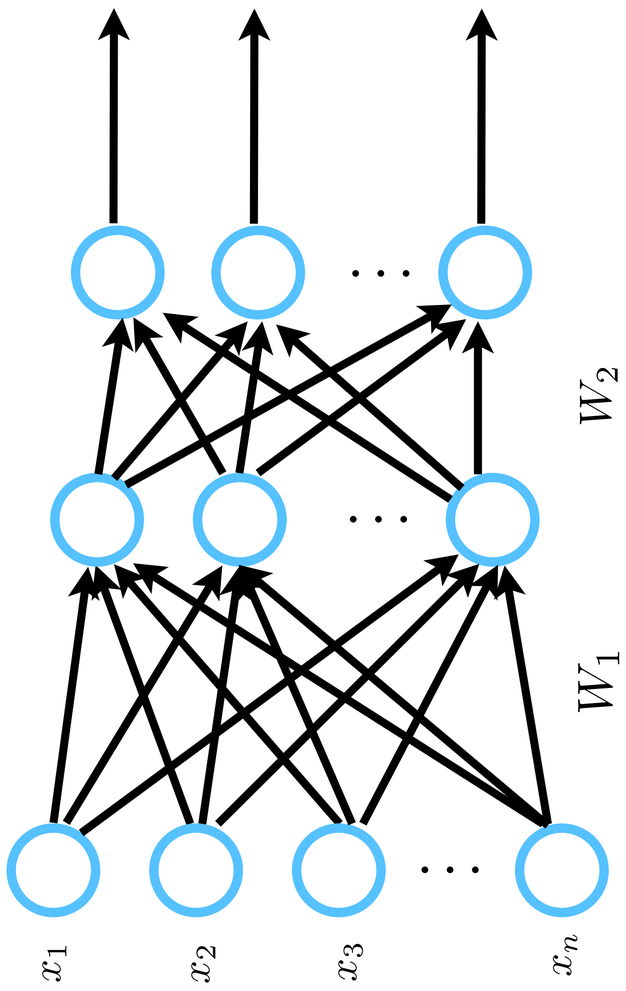
Thus far..

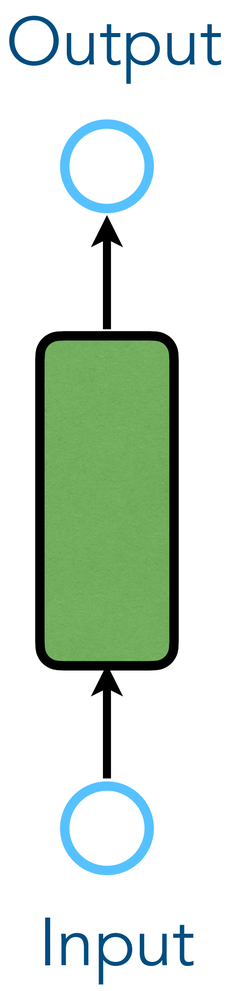




- One to one classification/ regression
- There is no notion of sequence

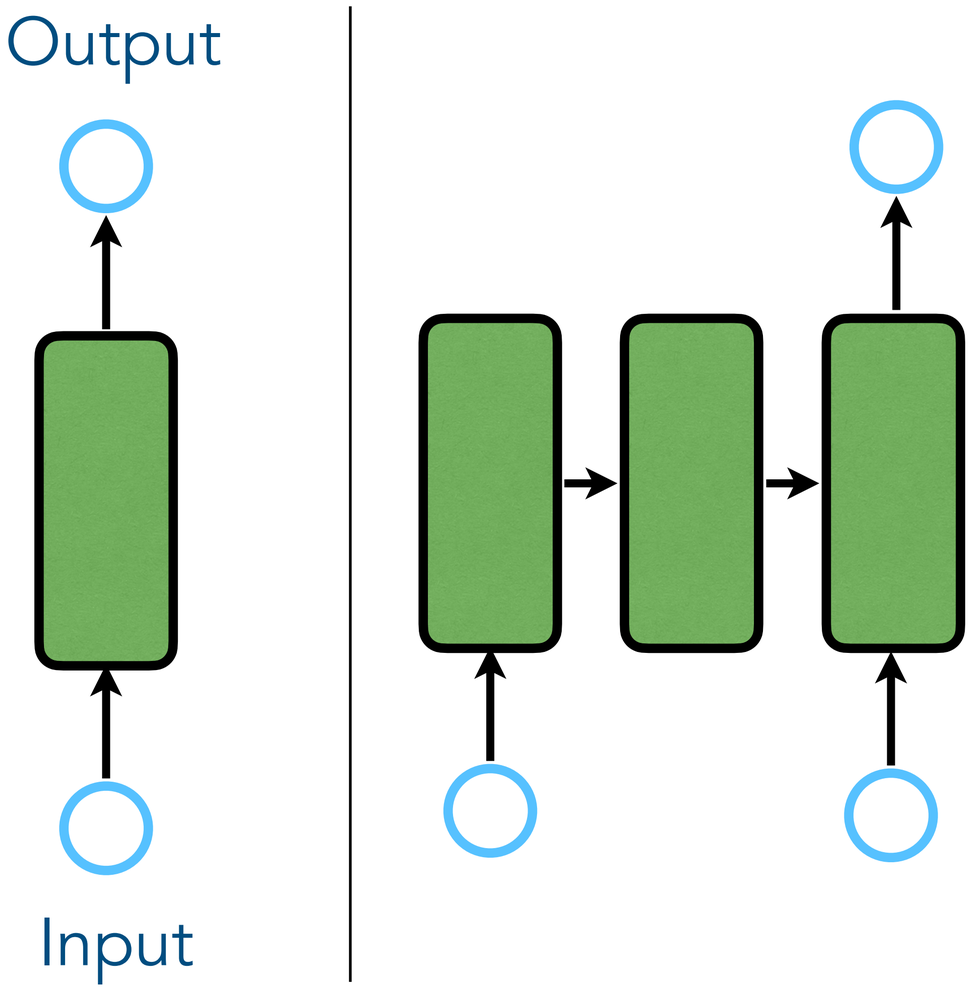
Thus far..



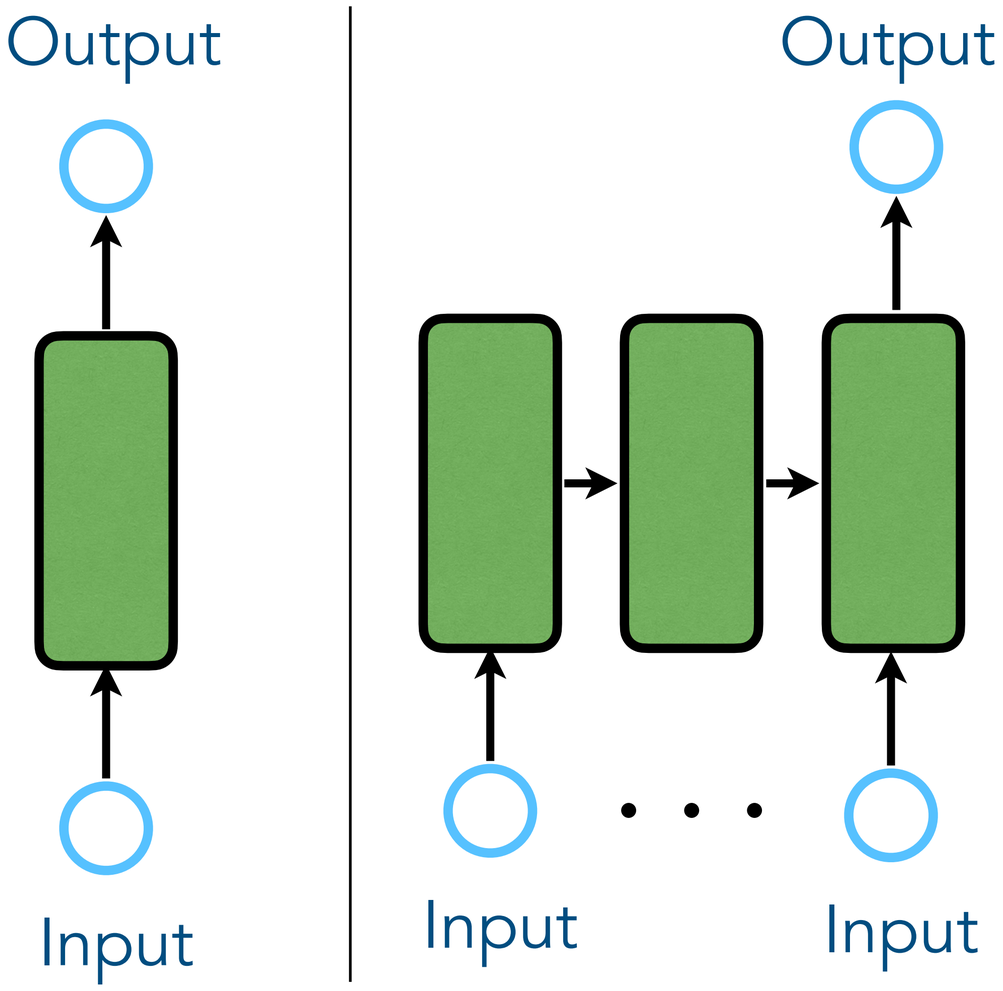


- One to one classification/ regression
- There is no notion of sequence
- E.g., given single image, classify it

Sequence modeling problems

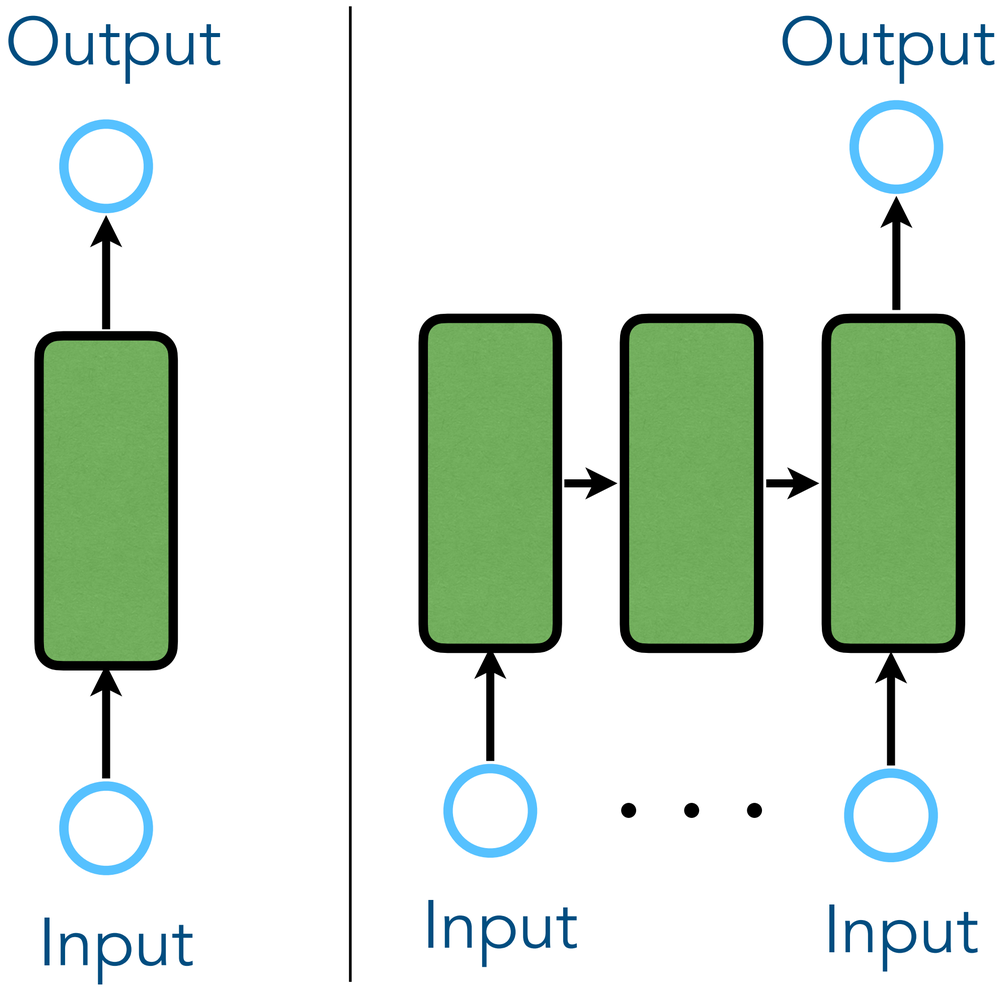


Sequence modeling problems



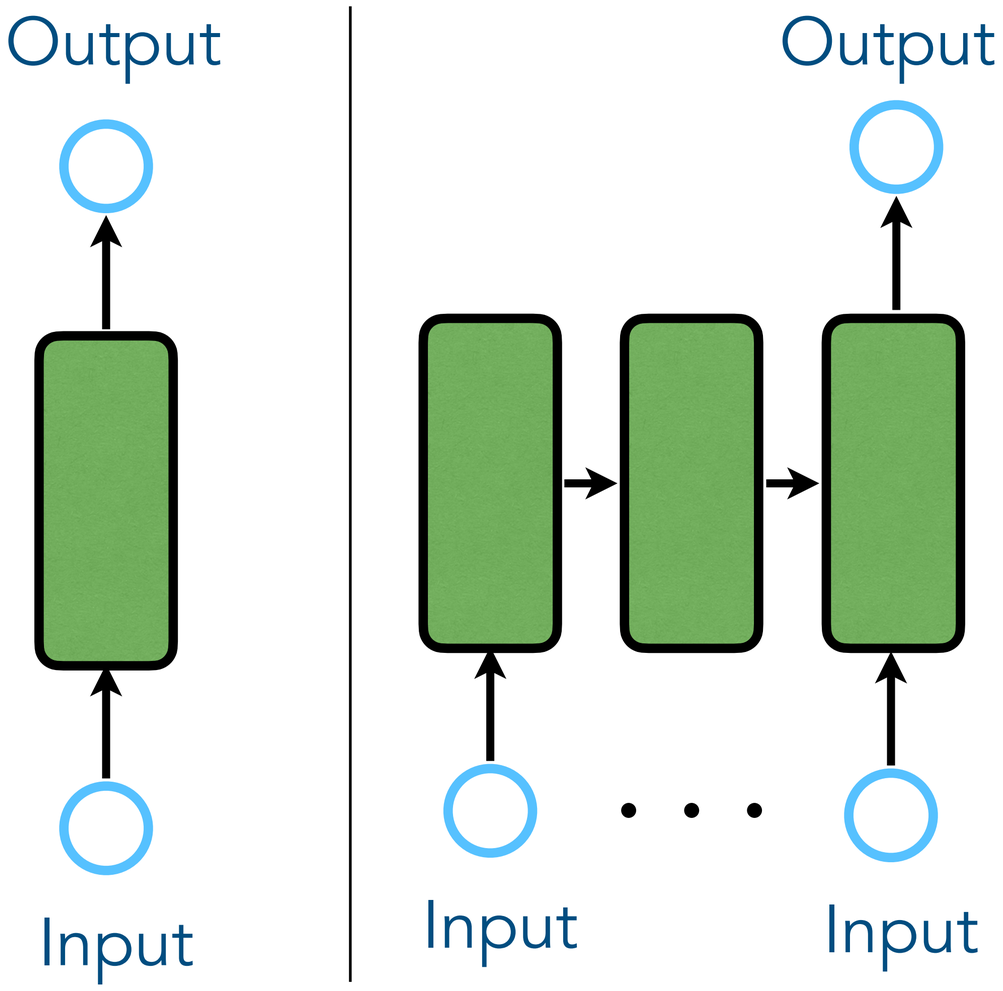
- **Many to one** classification/ regression

Sequence modeling problems



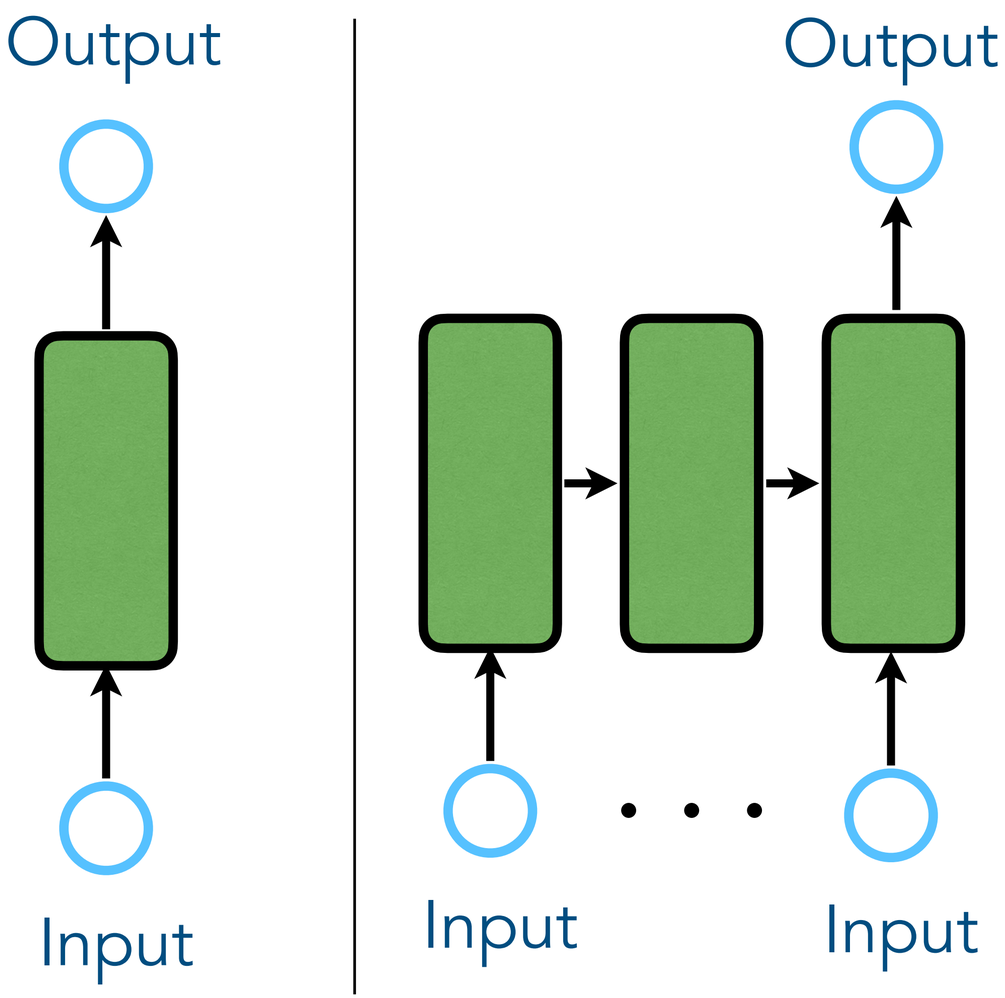
- **Many to one** classification/ regression
- There is notion of sequence in input

Sequence modeling problems

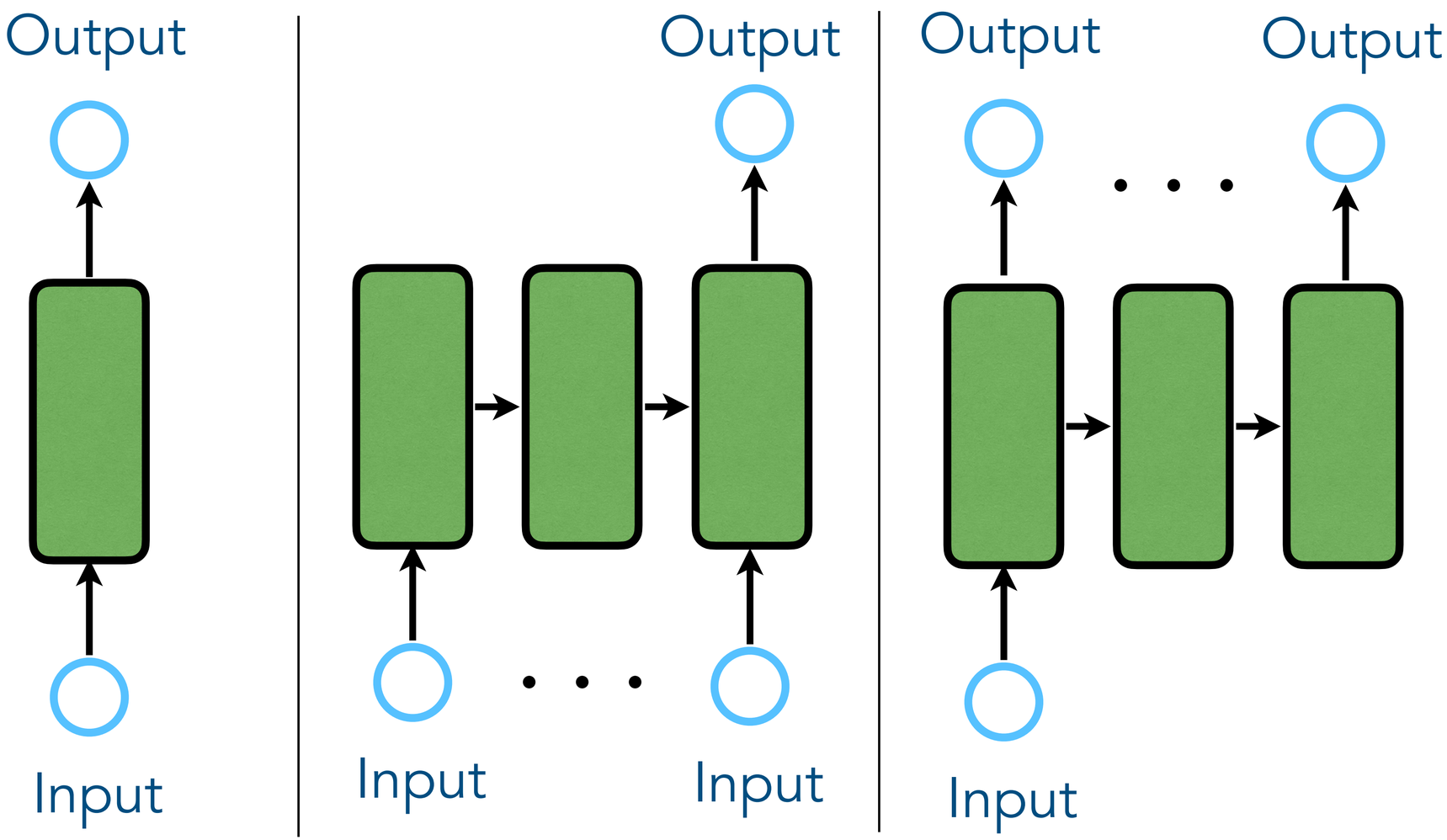


- **Many to one** classification/ regression
- There is notion of sequence in input
- E.g., sequence of data, make a decision at the end (sentiment classification)

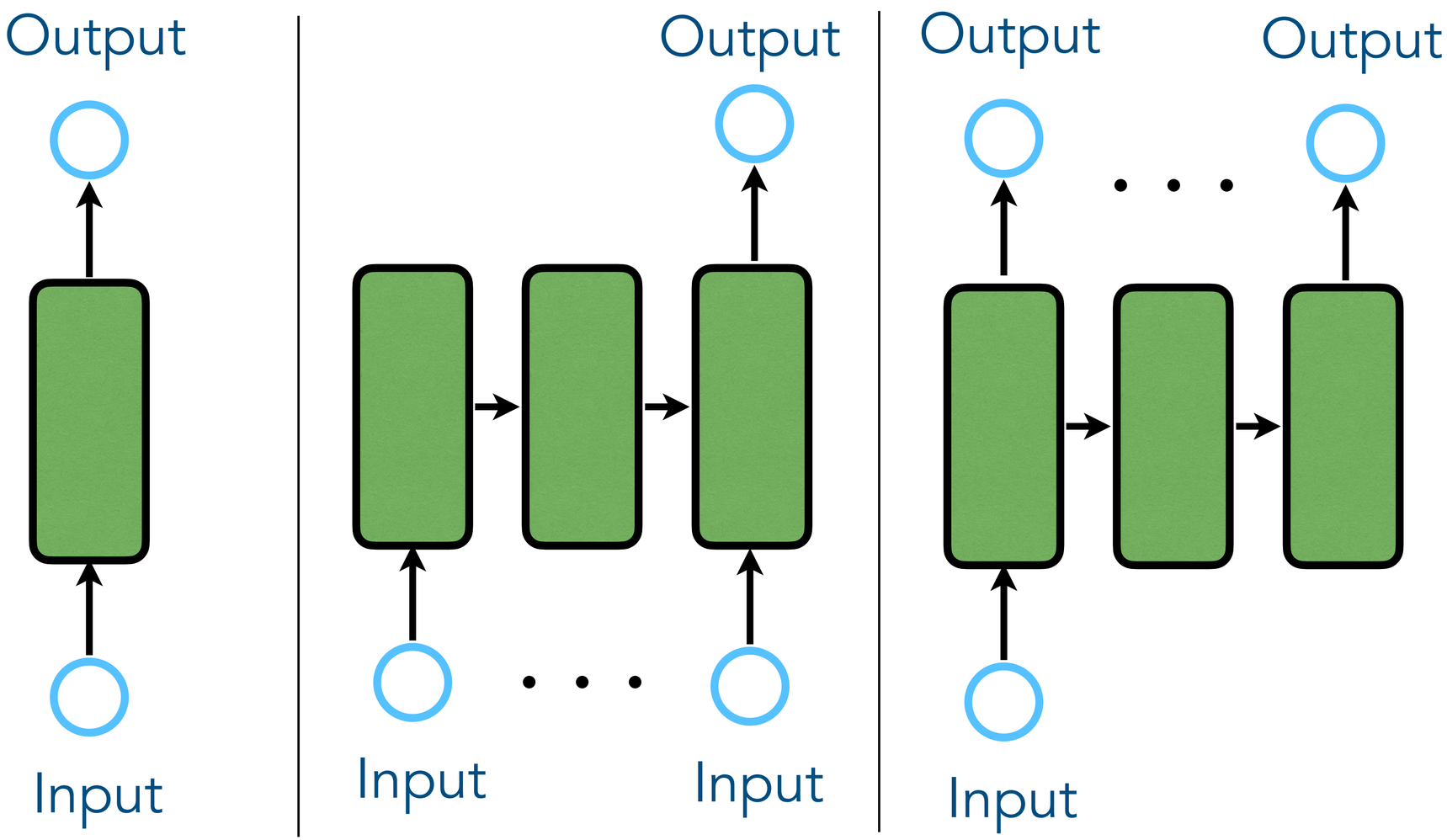
Sequence modeling problems



Sequence modeling problems

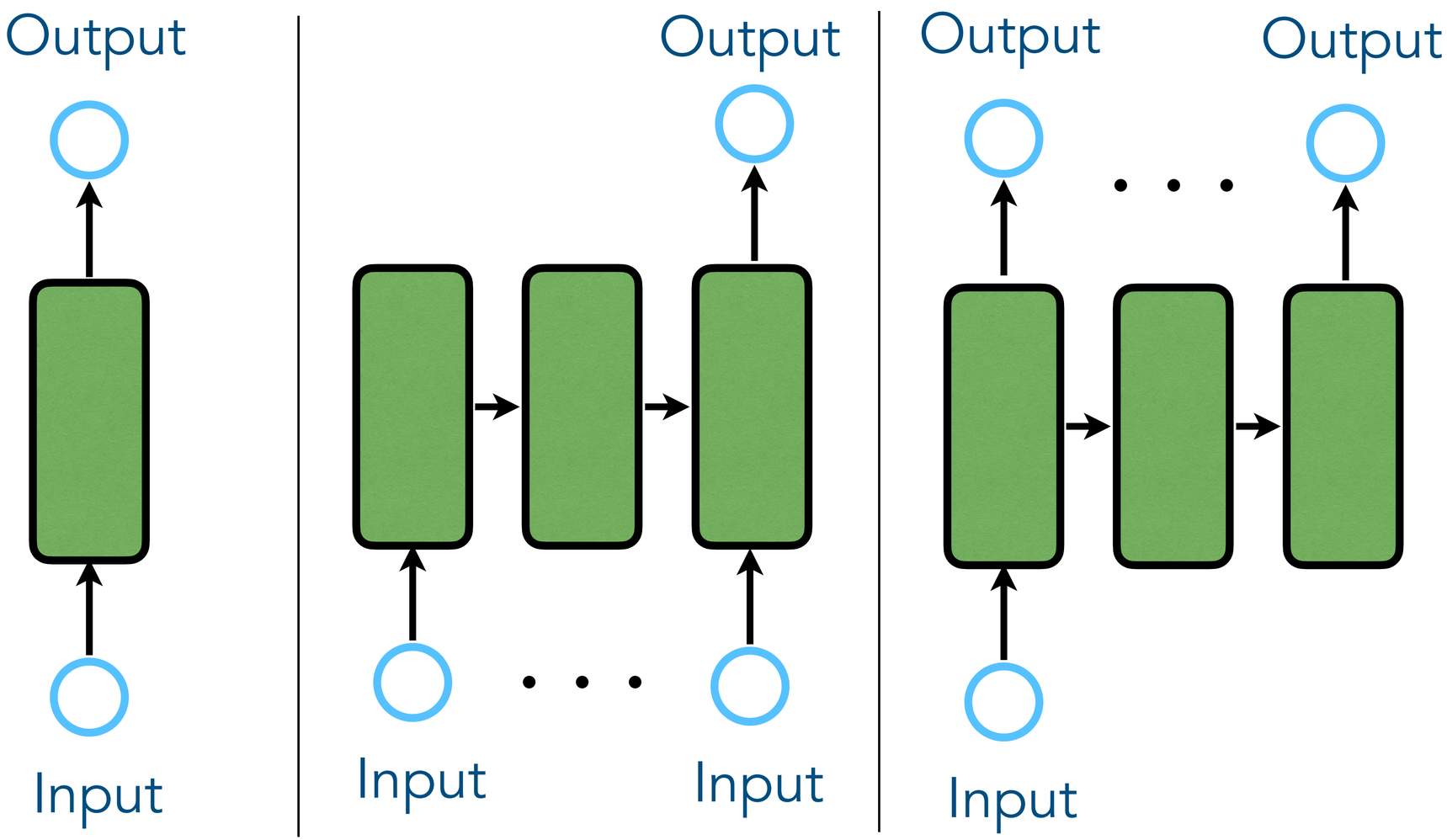


Sequence modeling problems



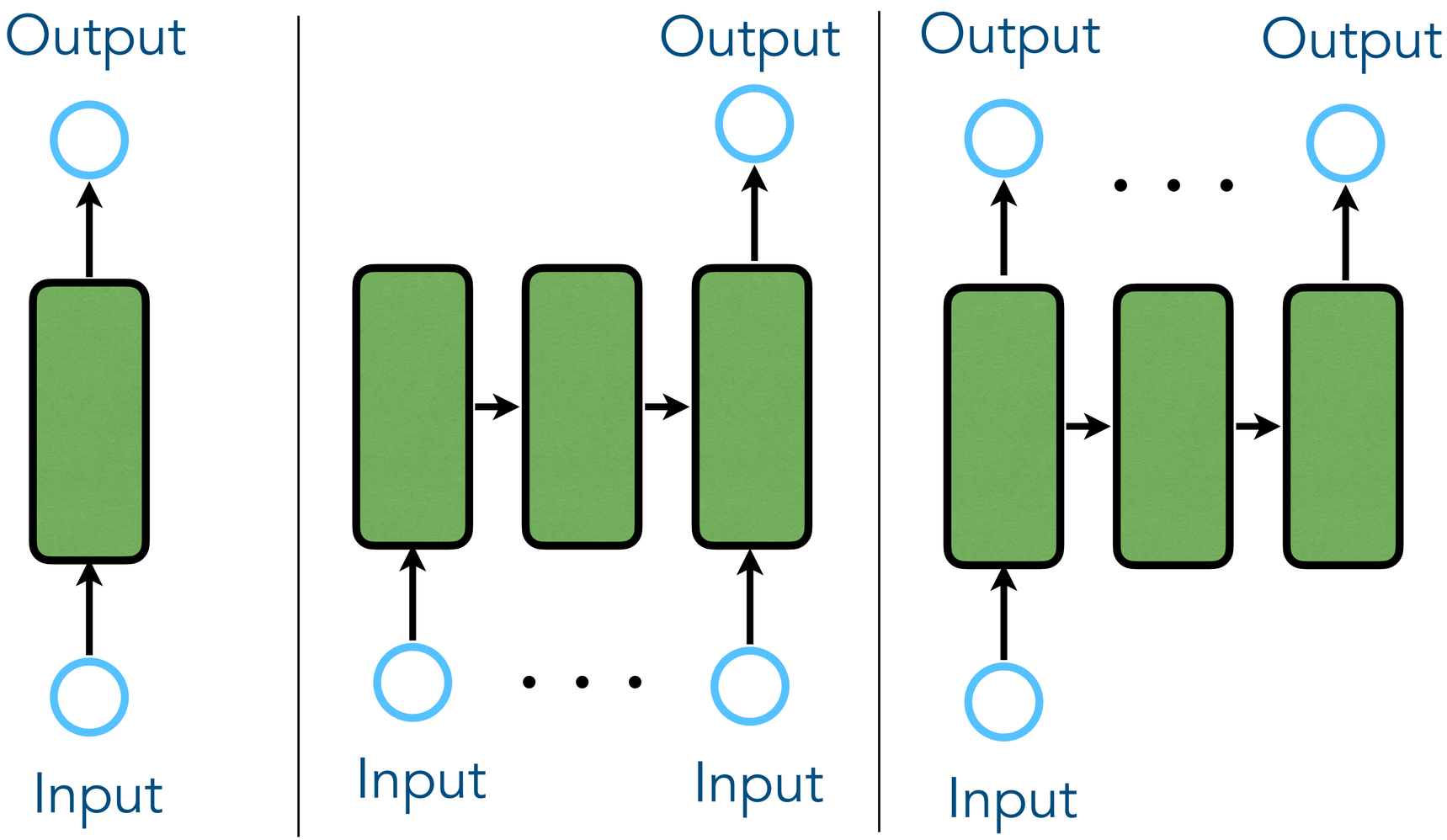
- **One to many**
  classification/
  regression

Sequence modeling problems



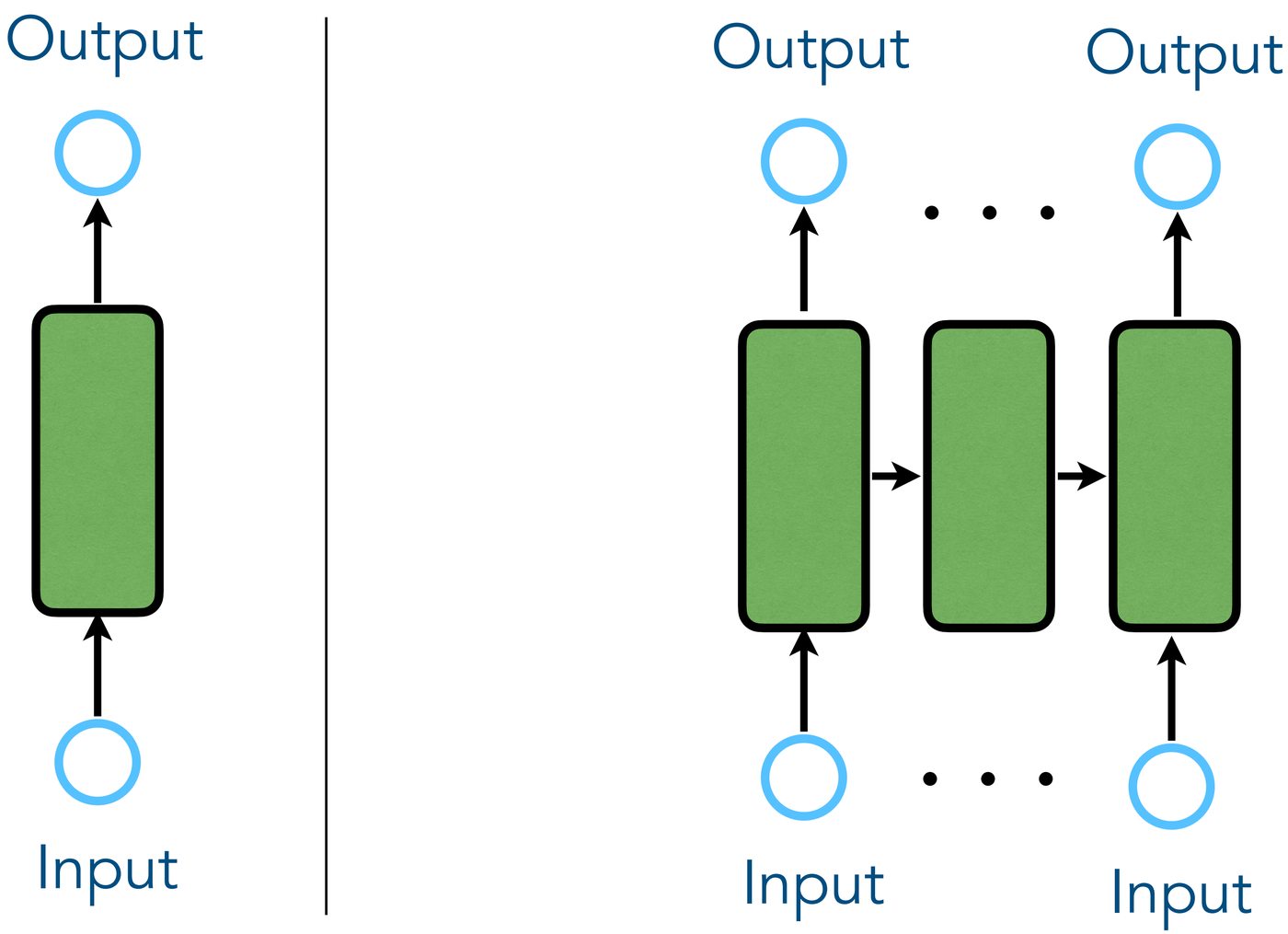
- **One to many** classification/ regression
- There is notion of sequence in the output space

Sequence modeling problems

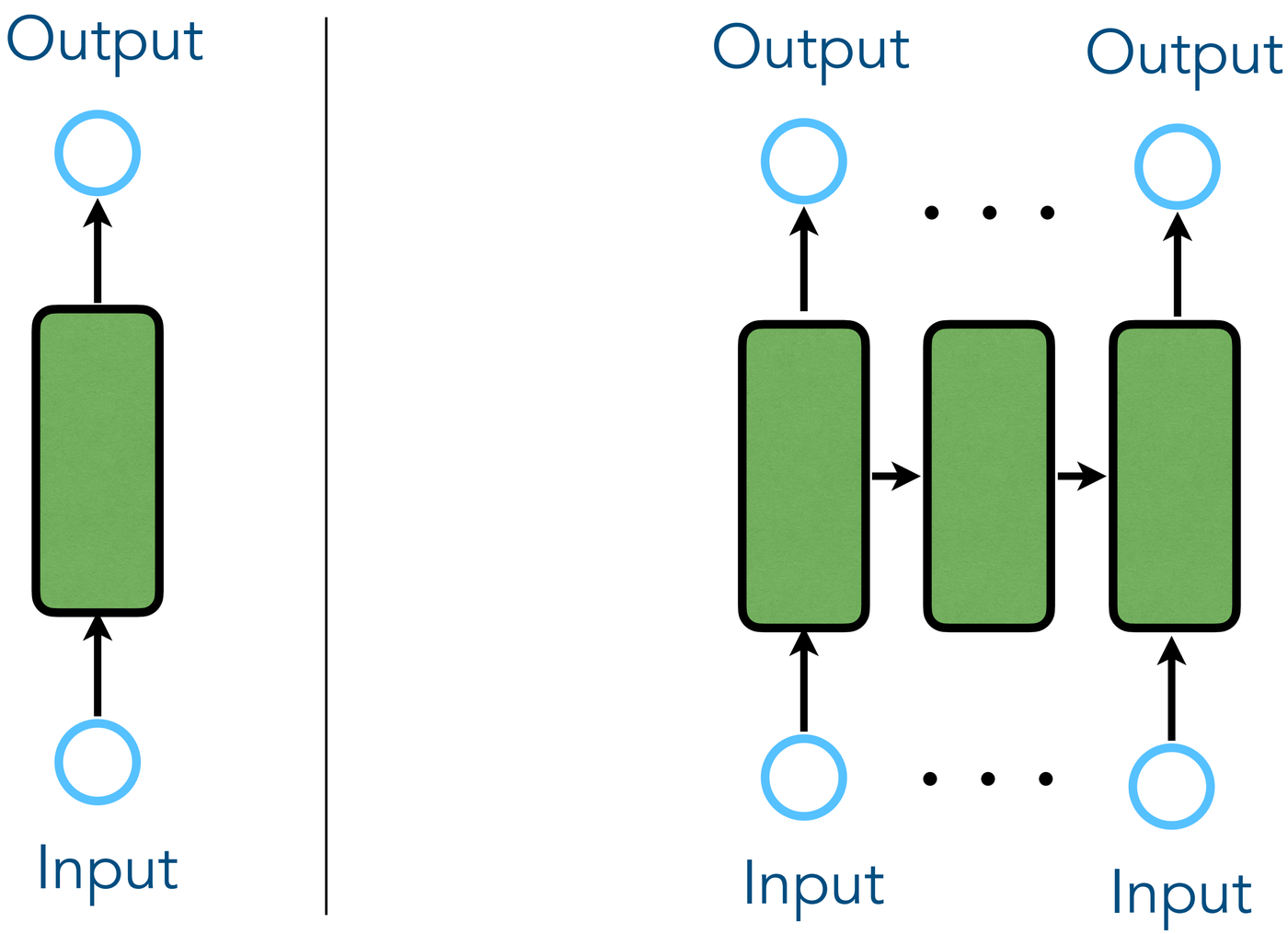


- **One to many** classification/ regression
- There is notion of sequence in the output space
- E.g., given an image, create a caption

Sequence modeling problems

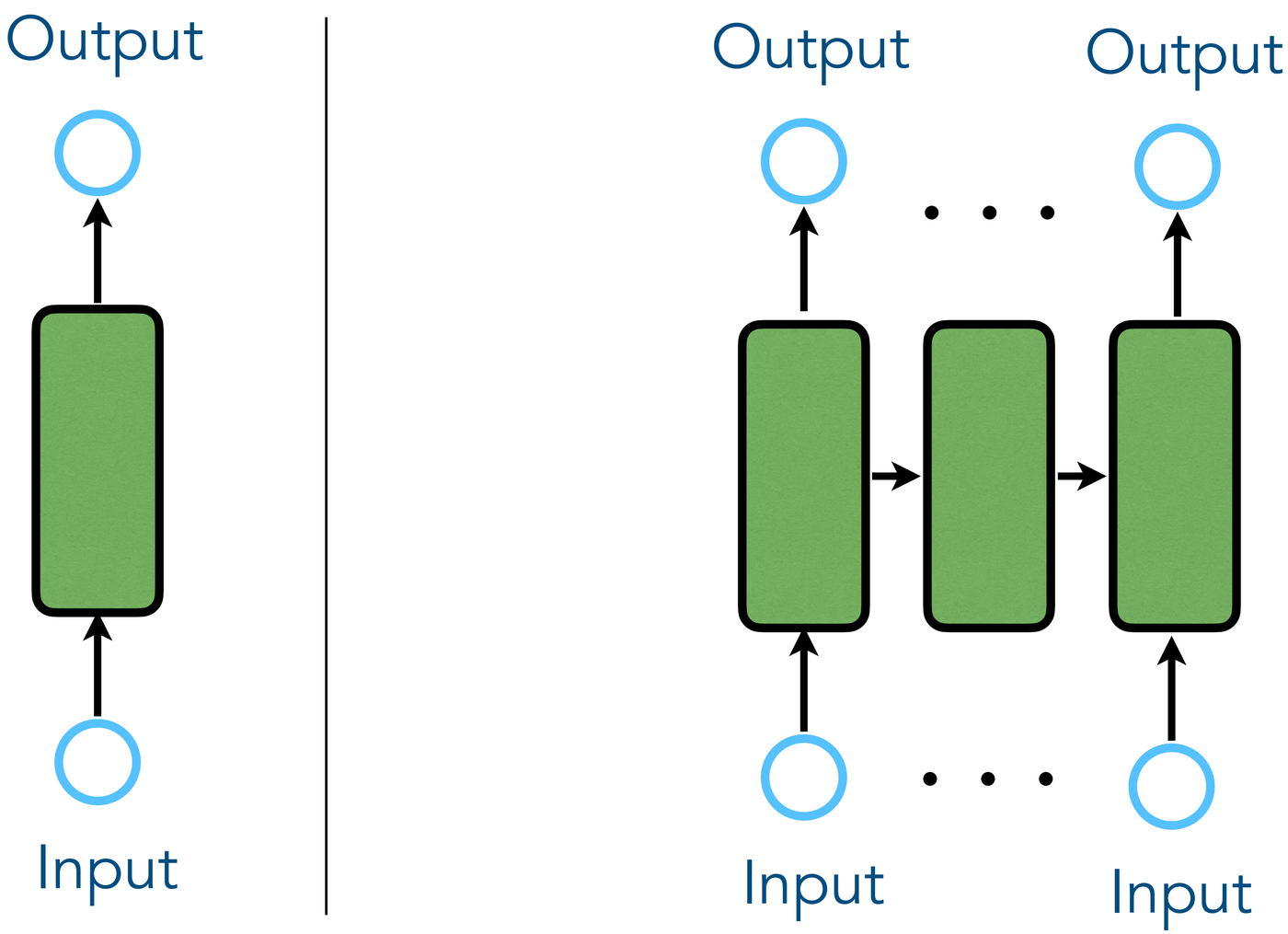


Sequence modeling problems



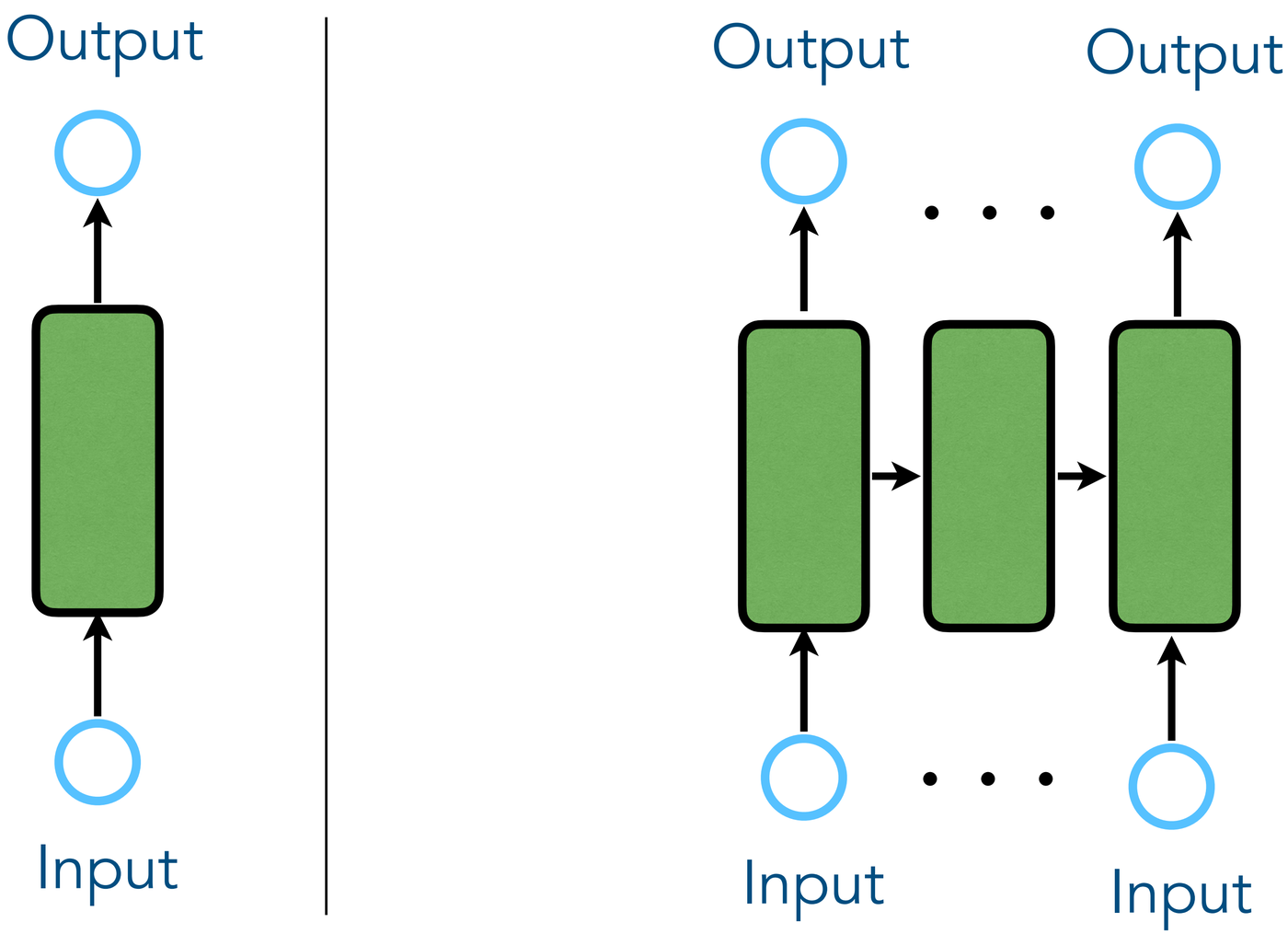
- **Many to many** classification/ regression

Sequence modeling problems



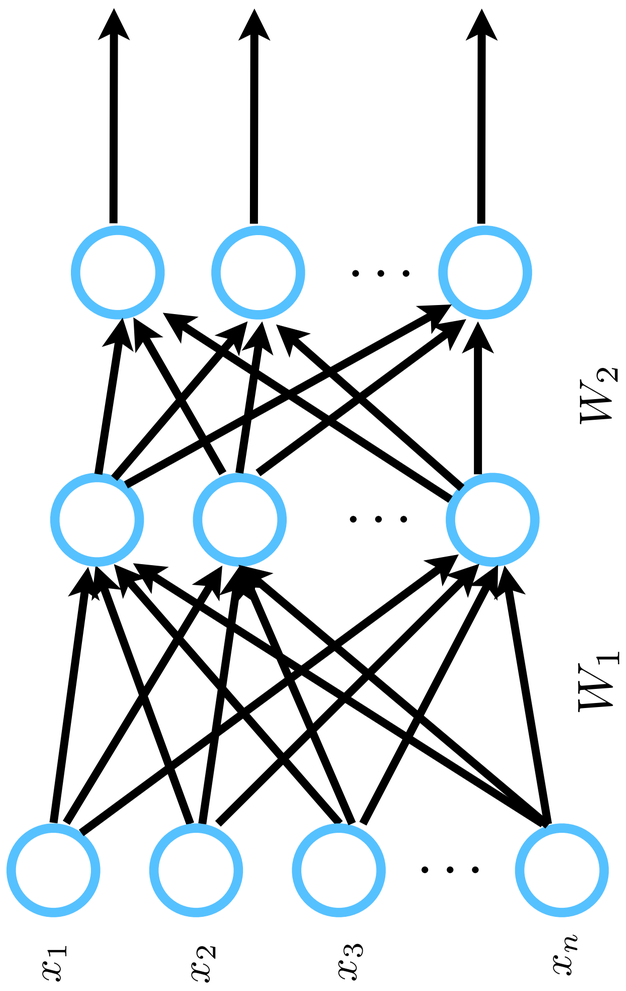
- **Many to many** classification/ regression
- There is notion of sequence both in input and output space

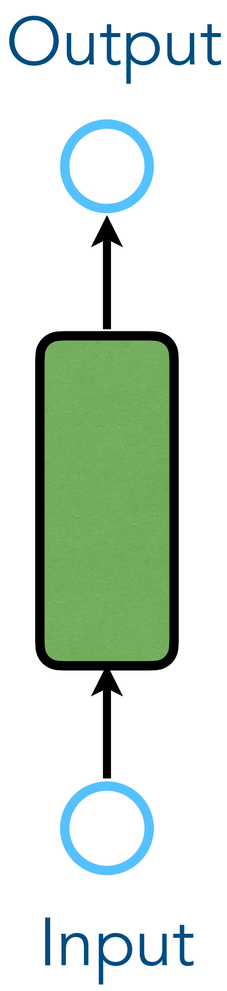
Sequence modeling problems



- **Many to many** classification/ regression
- There is notion of sequence both in input and output space
- E.g., machine translation, forecasting, music generation

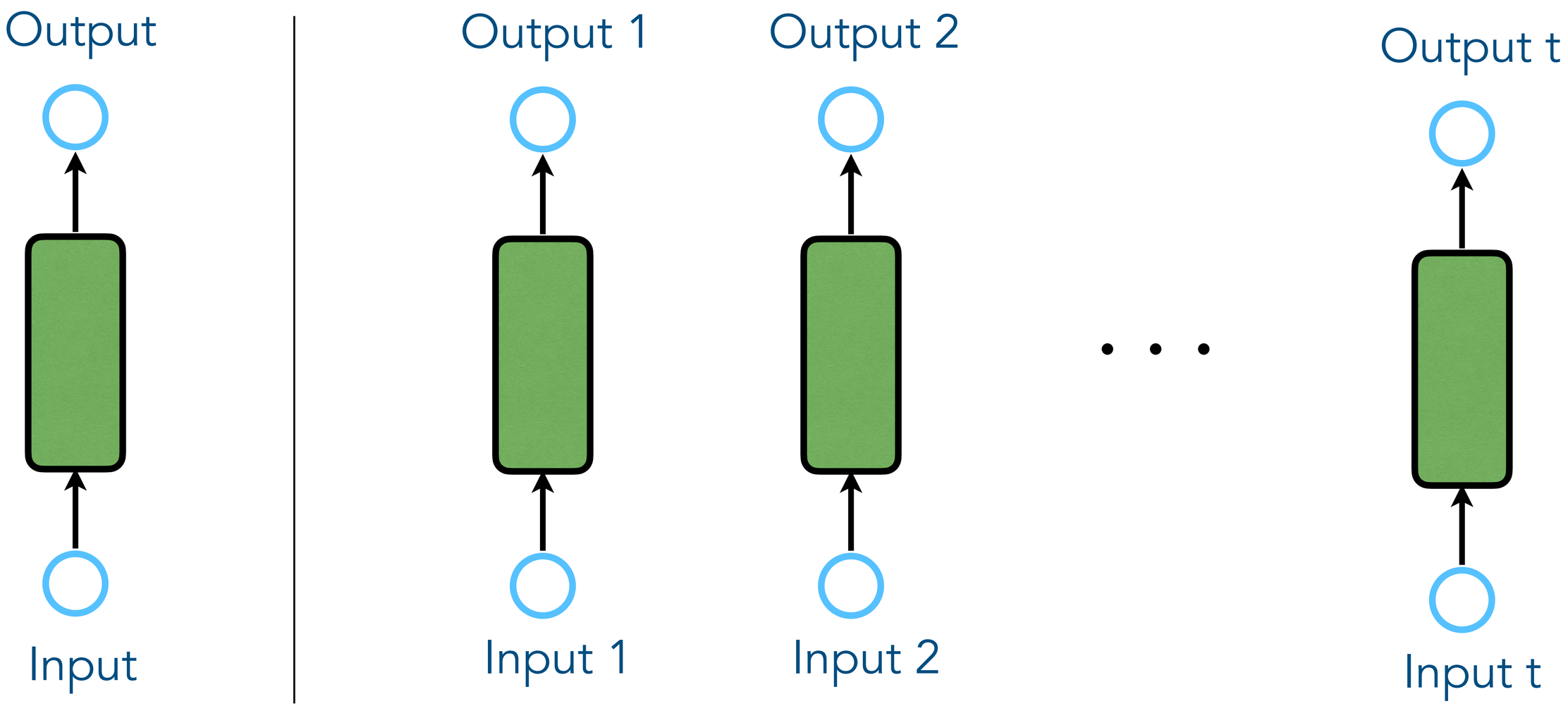
How does this relate to the models so far?





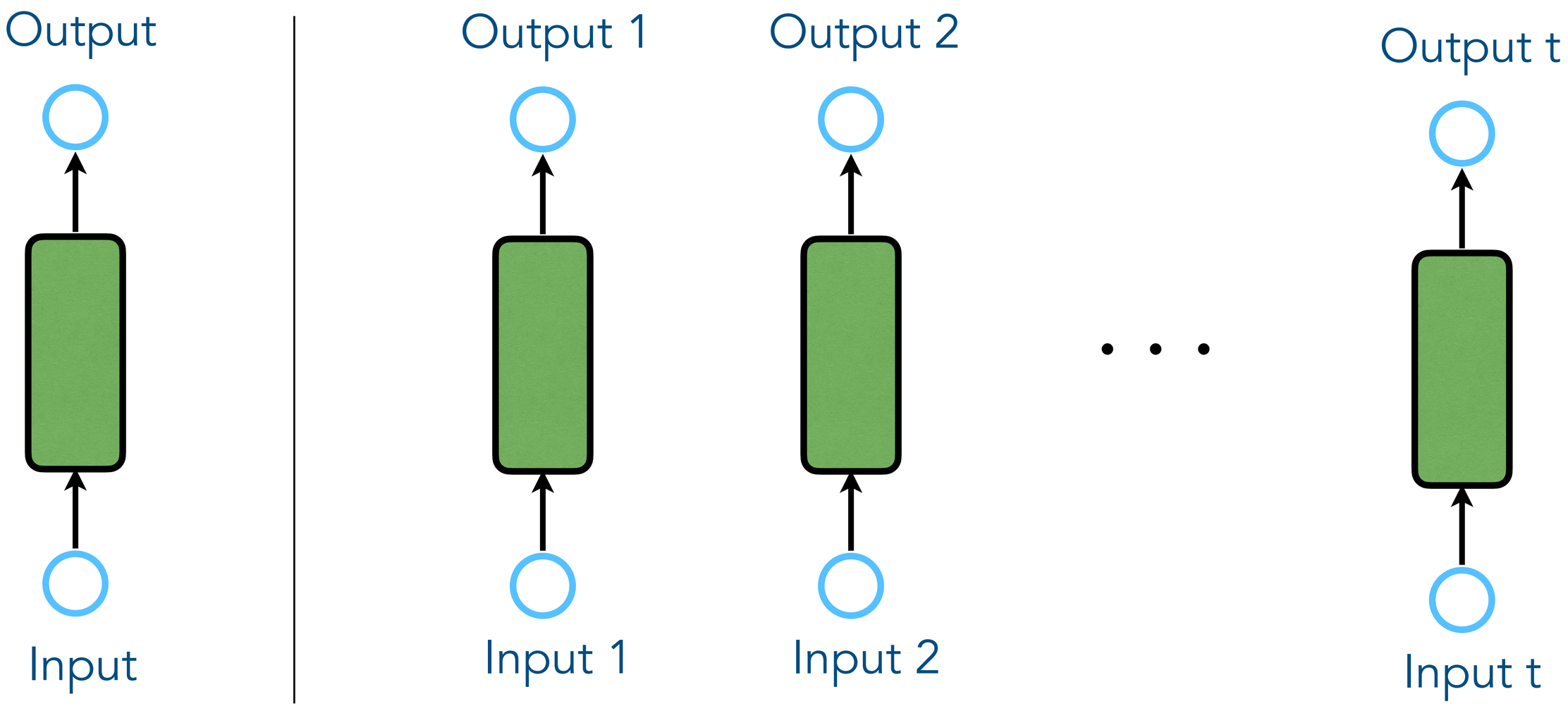
How does this relate to the models so far?

$$\hat{y}_t = f(x_t)$$



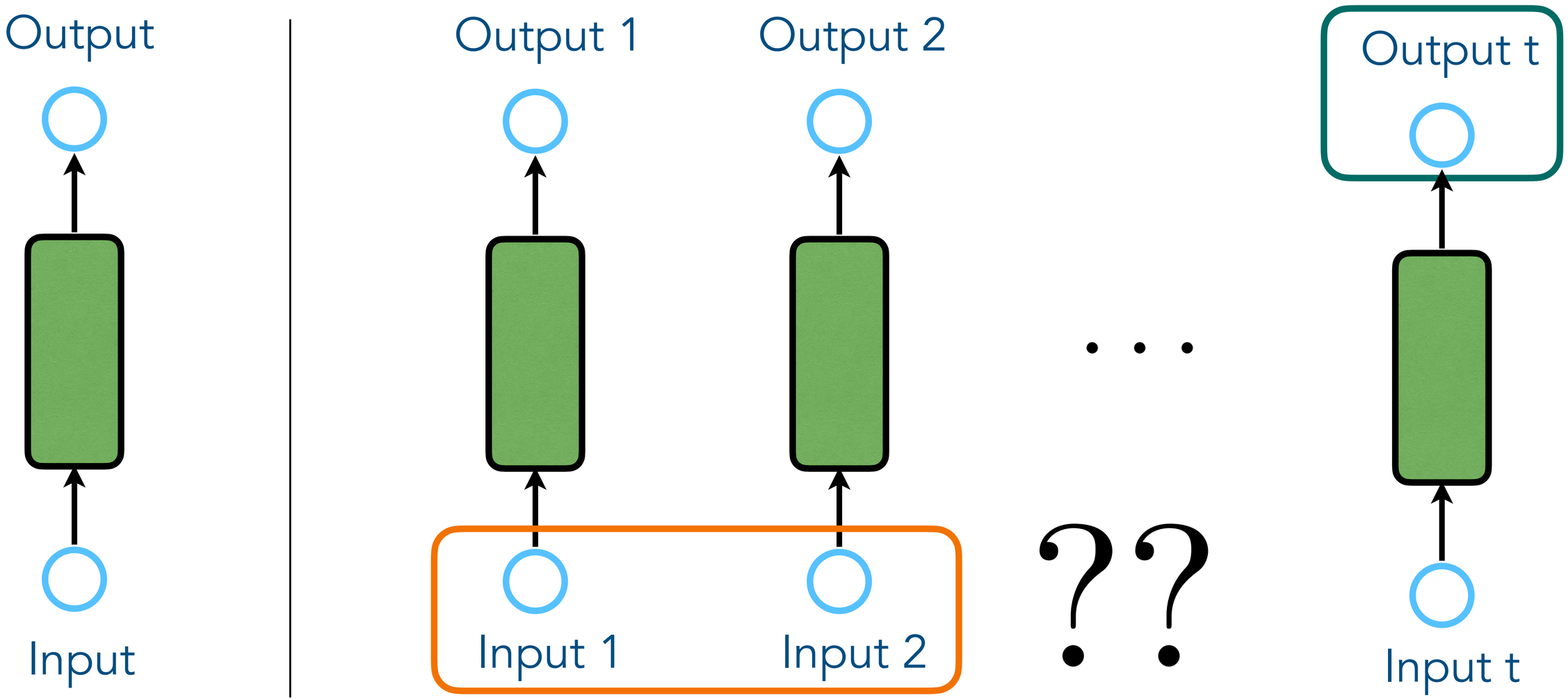
How does this relate to the models so far?

$$\hat{y}_t = f(x_t)$$

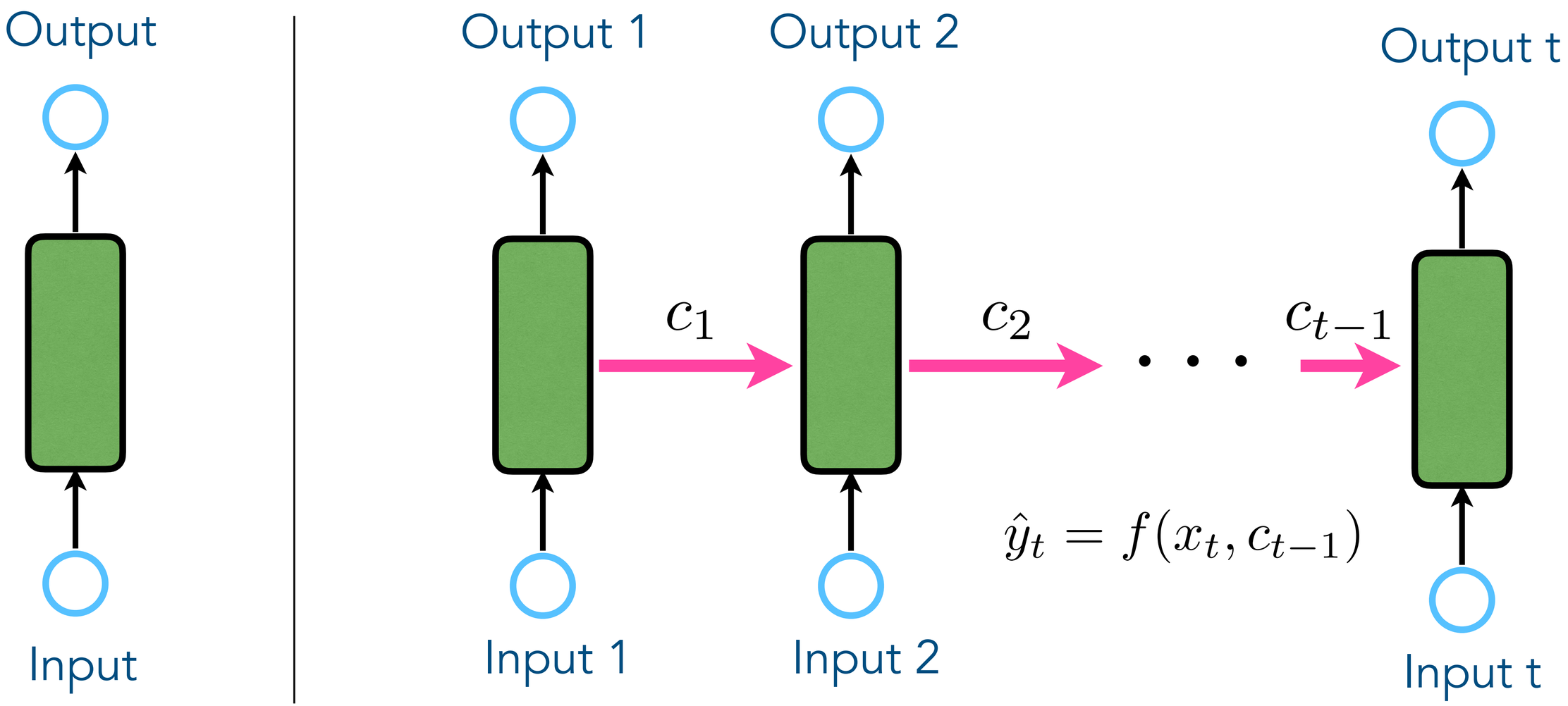


The way we use so far NNs is independent of each "run"

How does this relate to the models so far?



What if we insert ``context'' from previous runs?



Recurrent Neural Networks

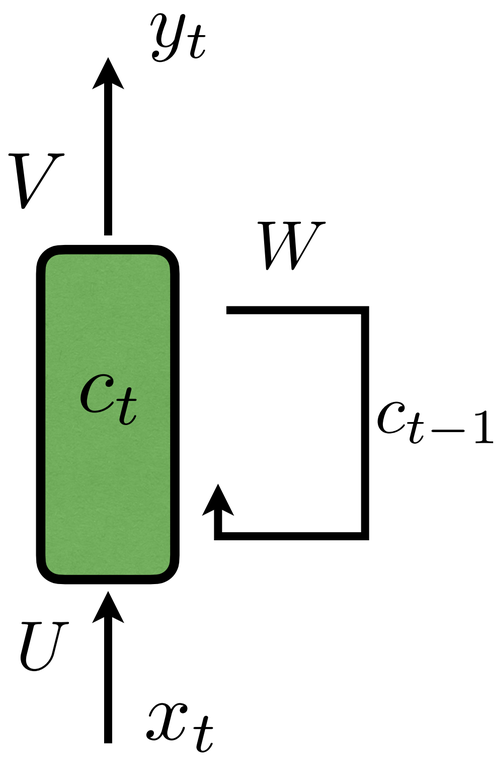
- A notion of **memory**: what we will decide in time t, could/should depend on what we decided in previous time steps

Recurrent Neural Networks

- A notion of **memory**: what we will decide in time t, could/should depend on what we decided in previous time steps
- This can go way back into the past: we need a mechanism that can accumulate this past information

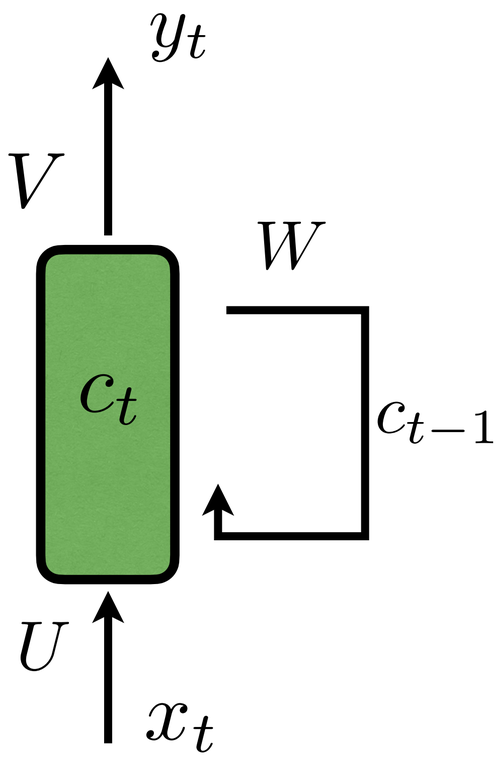
Recurrent Neural Networks

- A notion of **memory**: what we will decide in time t, could/should depend on what we decided in previous time steps
- This can go way back into the past: we need a mechanism that can accumulate this past information



Recurrent Neural Networks

- A notion of **memory**: what we will decide in time t, could/should depend on what we decided in previous time steps
- This can go way back into the past: we need a mechanism that can accumulate this past information



Assume expressions:

$$c_t = \tanh\left(Ux_t + Wc_{t-1}\right)$$

$$y_t = \texttt{softmax}(Vc_t)$$

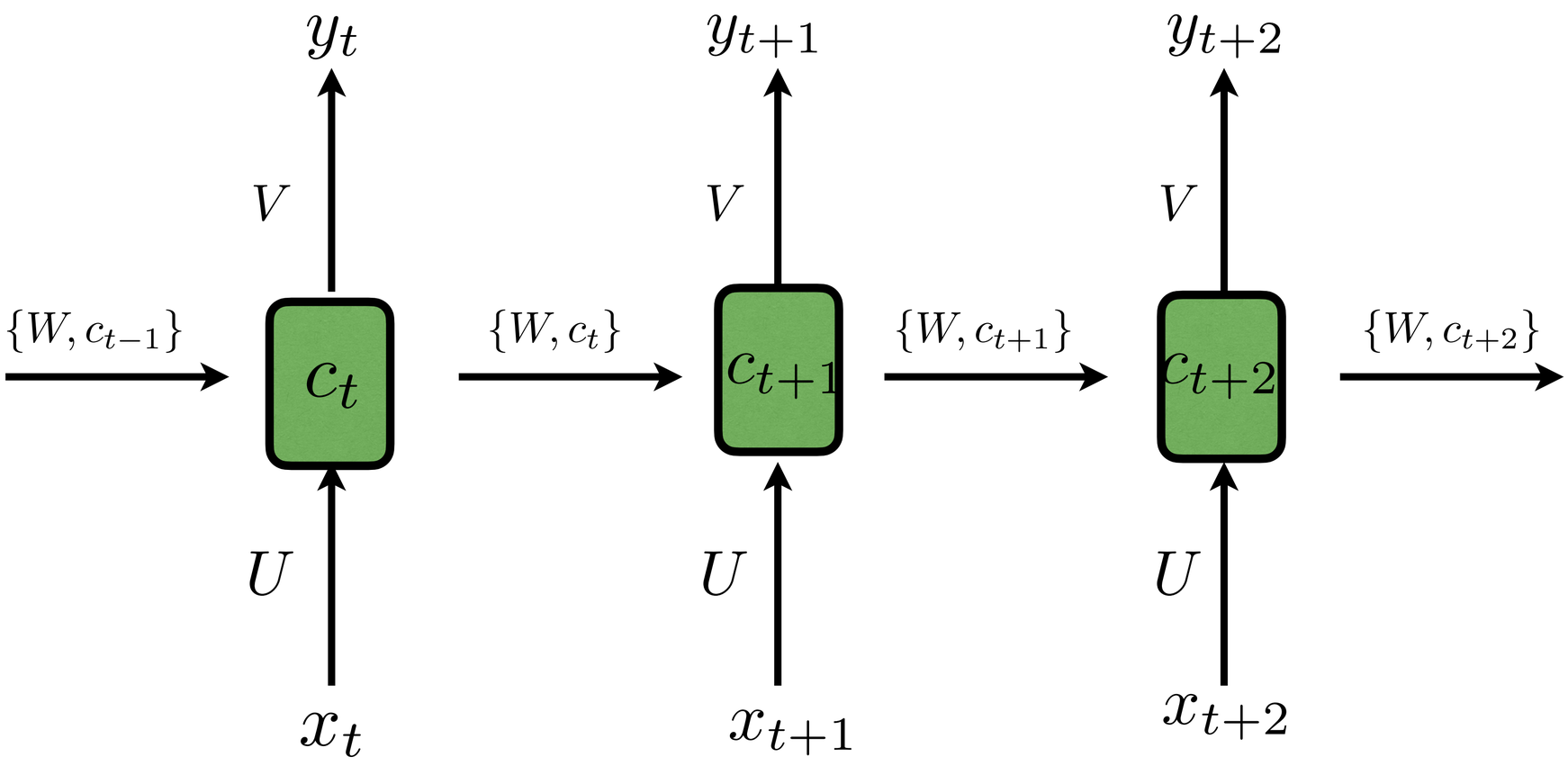
- x_t is input sequence at time t.
- $U,\ V,\ W$ are learnable parameters (as matrices here)

Loss function:

$$\sum_t \ell(y_t, y_t^\star)$$

Unfolding RNNs

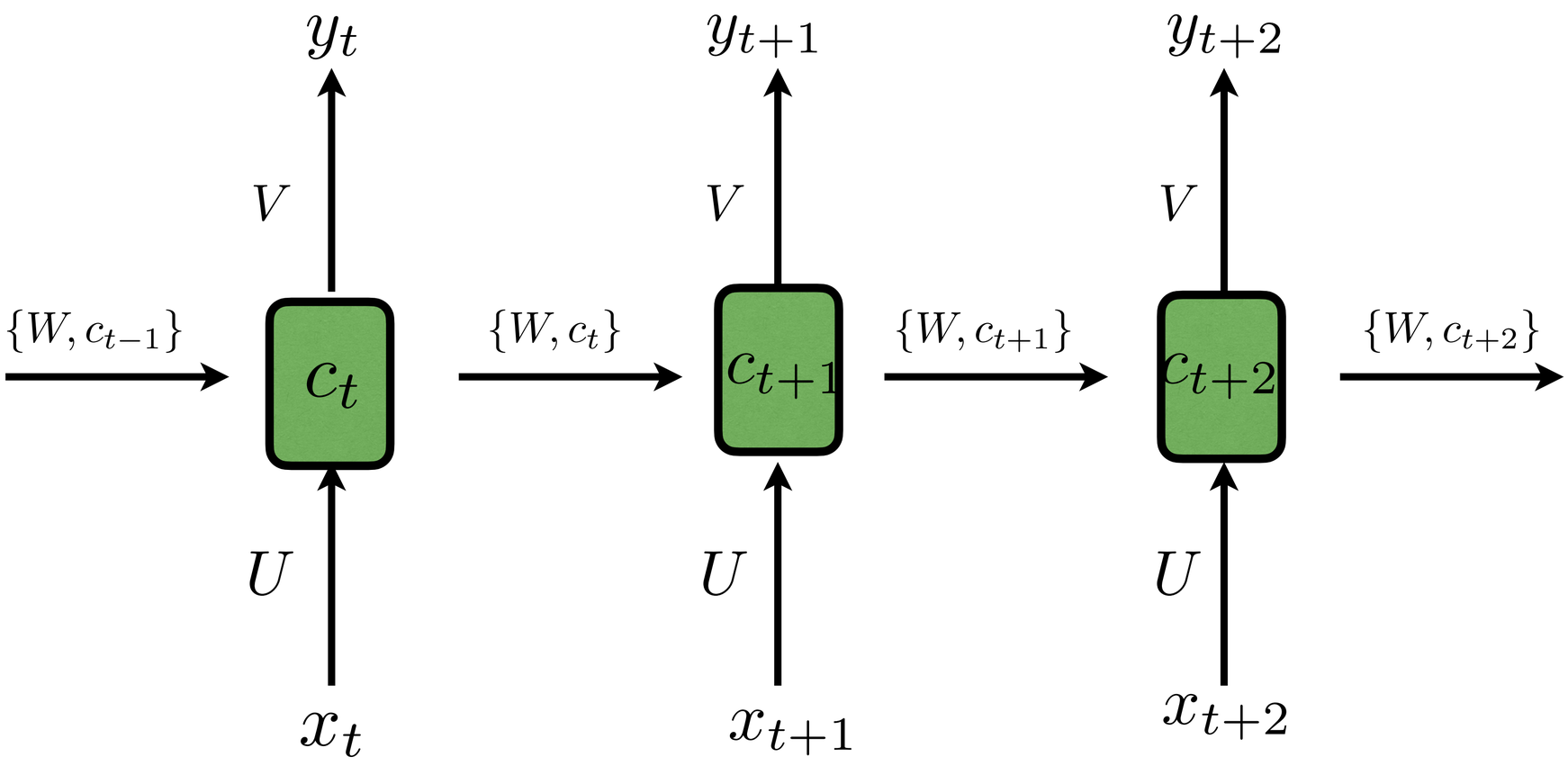
Unfolding RNNs



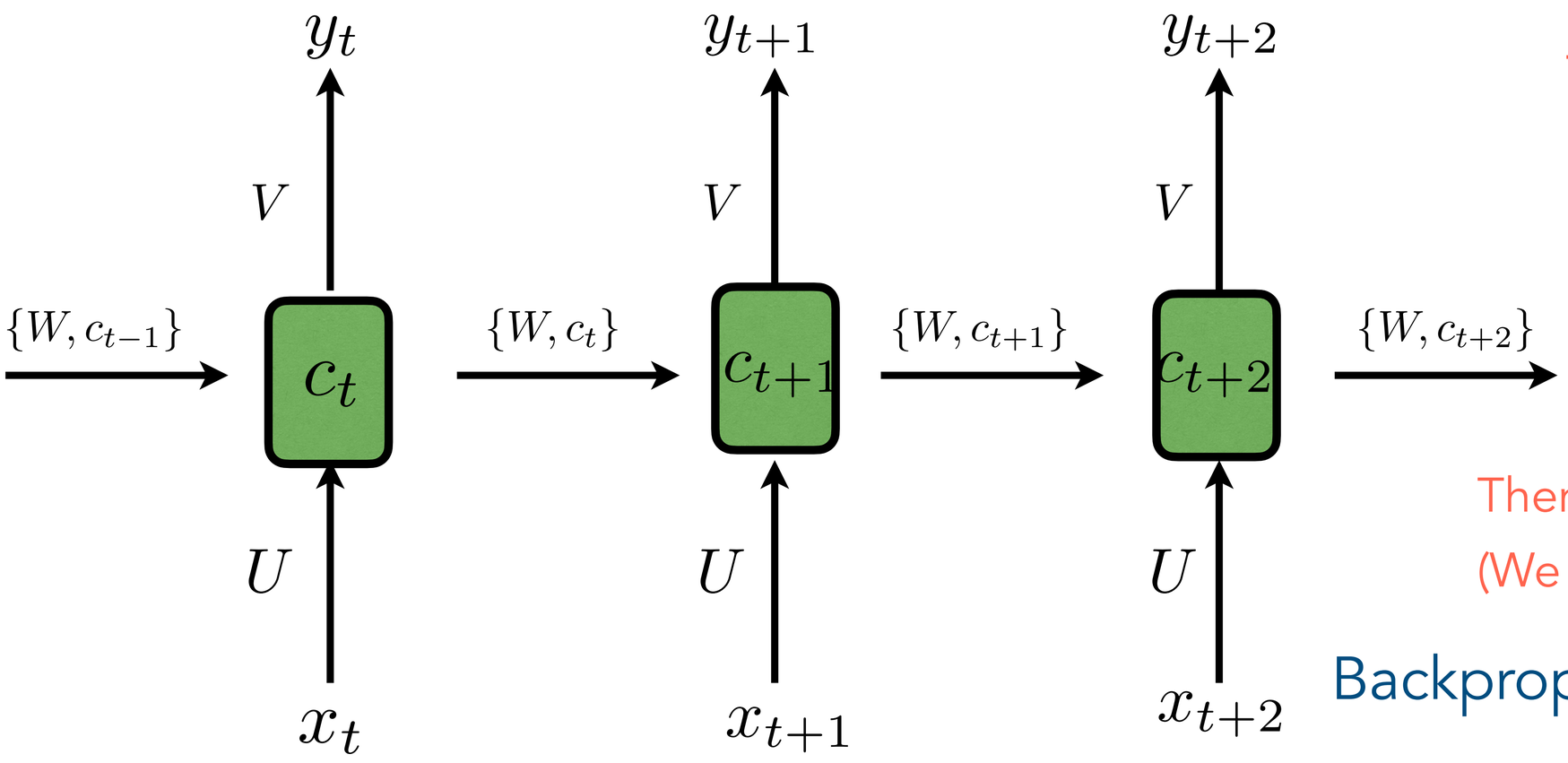
Unfolding RNNs

Main differences with MLPs:

- Output at every time step
- Layer-shared parameters



Unfolding RNNs



Main differences with MLPs:

- Output at every time step
- Layer-shared parameters

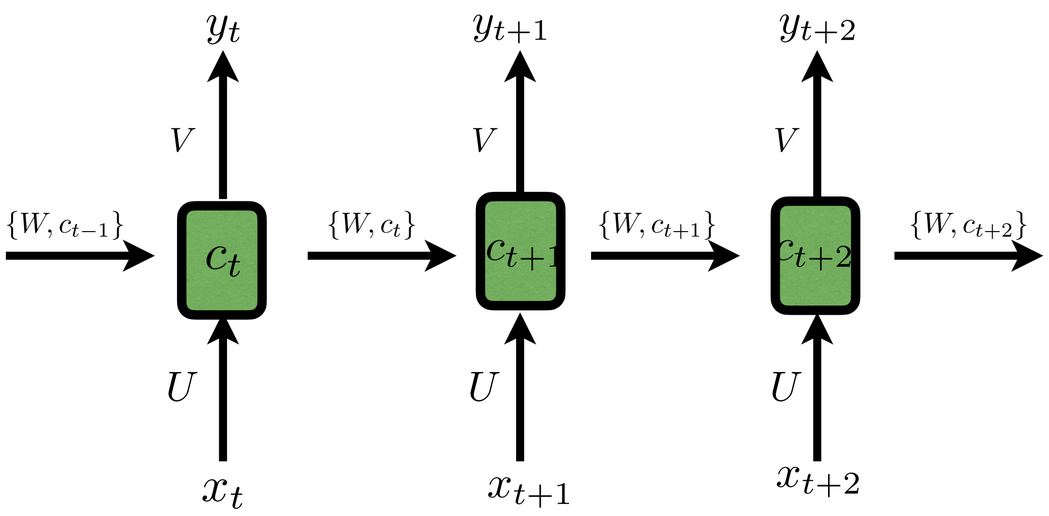
Then, how do we do backprop?

(We have dependences over time!)

Backpropagation through time!

Unfolding RNNs

Unfolding RNNs

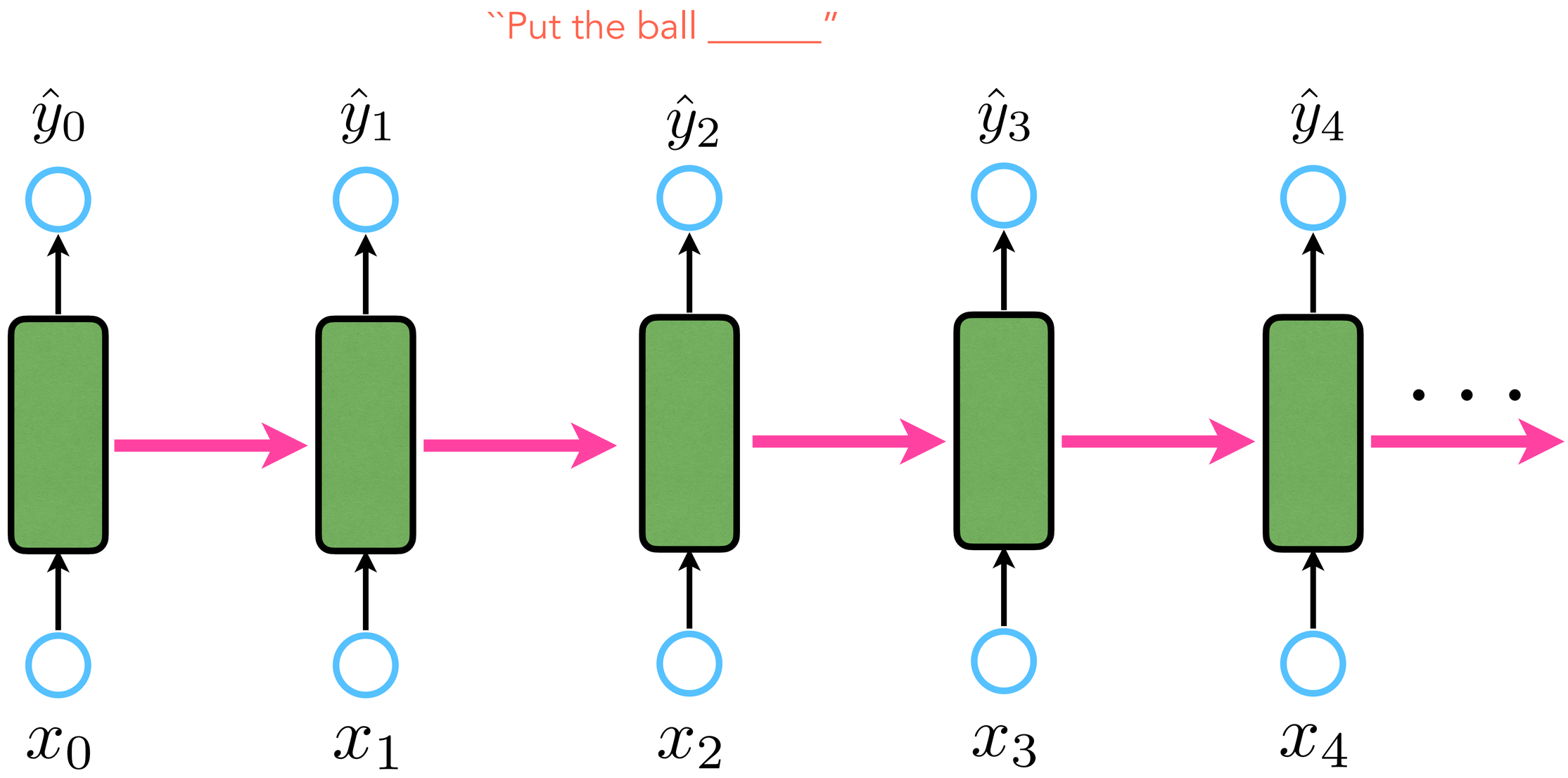


- Backpropagation through time!

  Unfolding back propagation leads to long memories which further leads to vanishing/exploding gradients

  There are tricks that partially solve the problem (rescaling gradients, do fewer time steps)

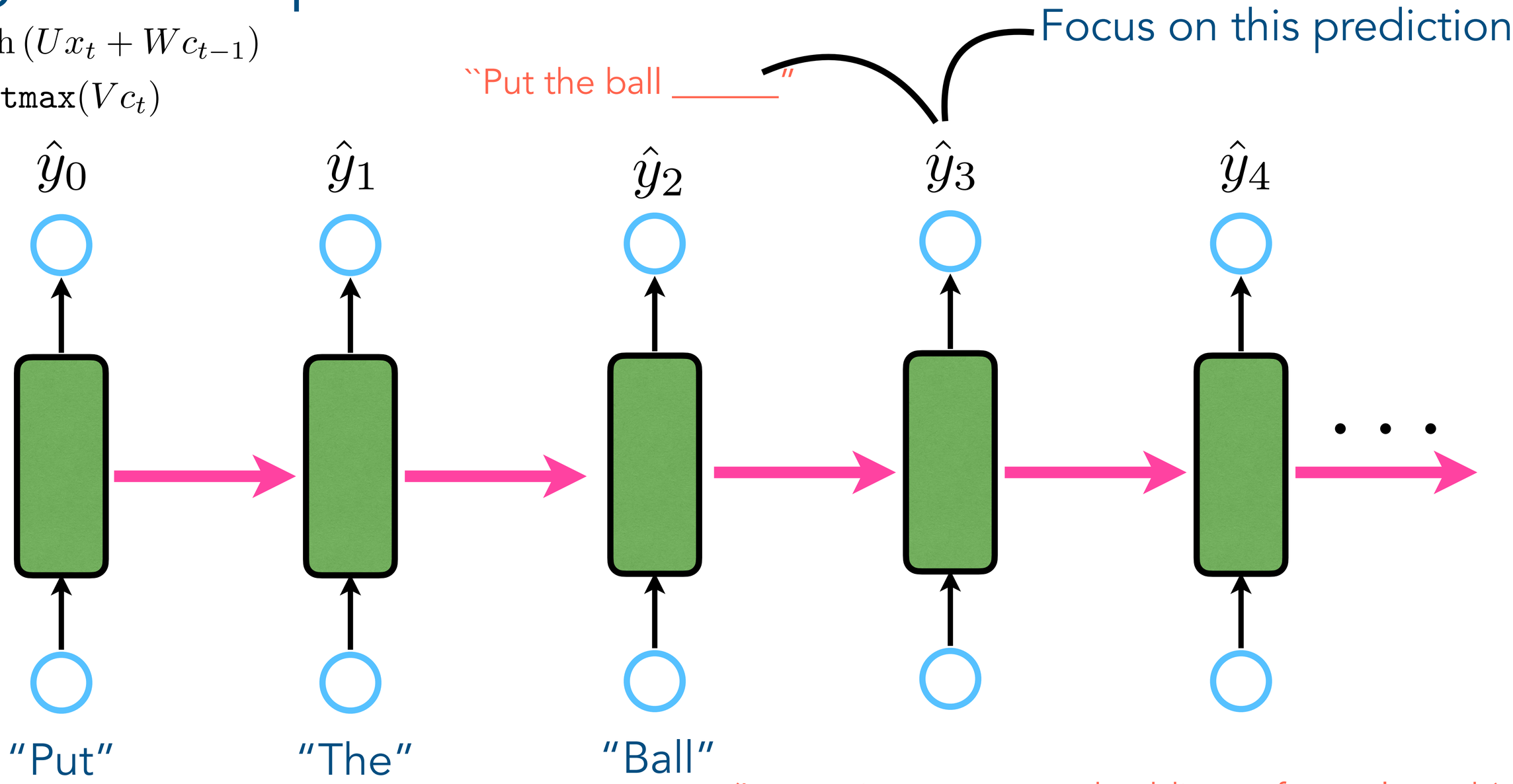
Long-term dependencies



Long-term dependencies

$$c_t = \tanh\left(Ux_t + Wc_{t-1}\right)$$

$$y_t = \texttt{softmax}(Vc_t)$$



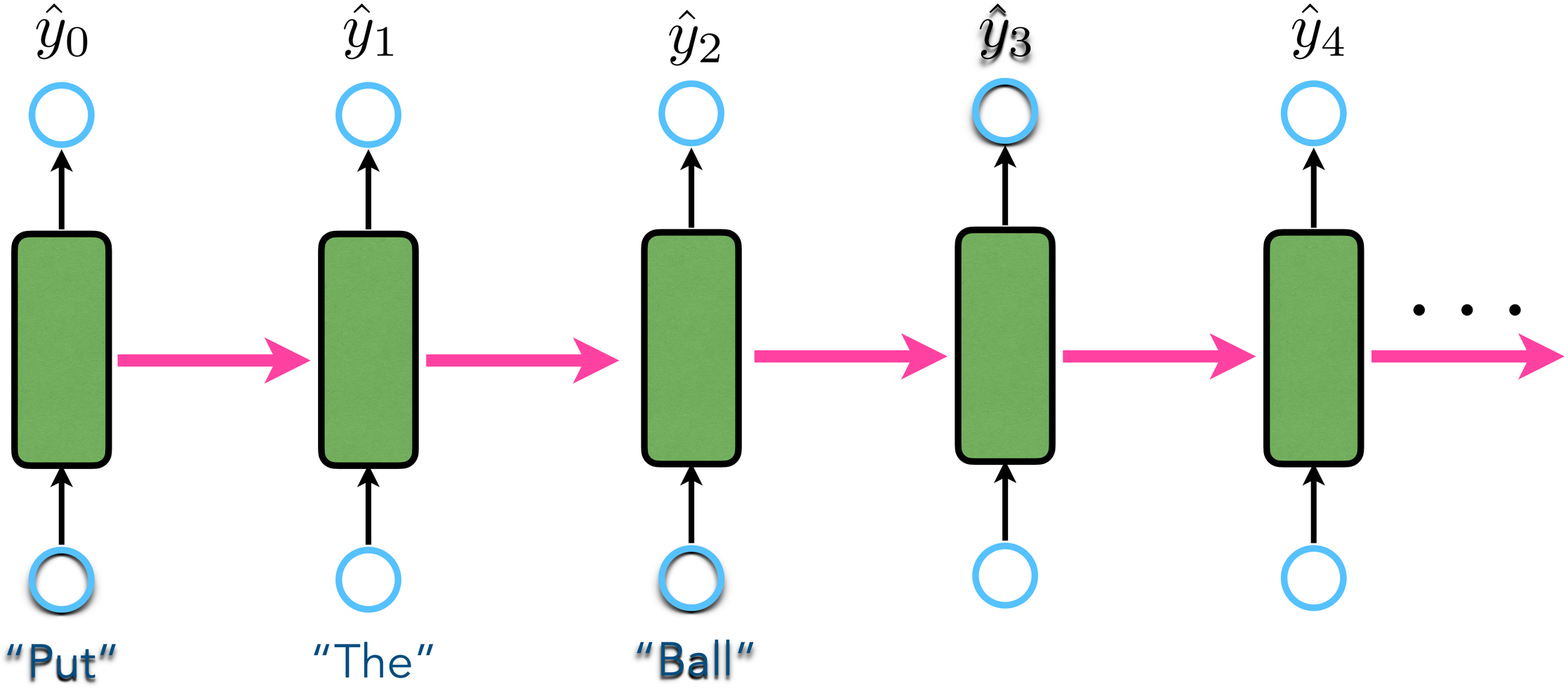
(In practice, we use embeddings of actual words)

Long-term dependencies

$$c_t = \tanh\left(Ux_t + Wc_{t-1}\right)$$

$$y_t = \texttt{softmax}(Vc_t)$$

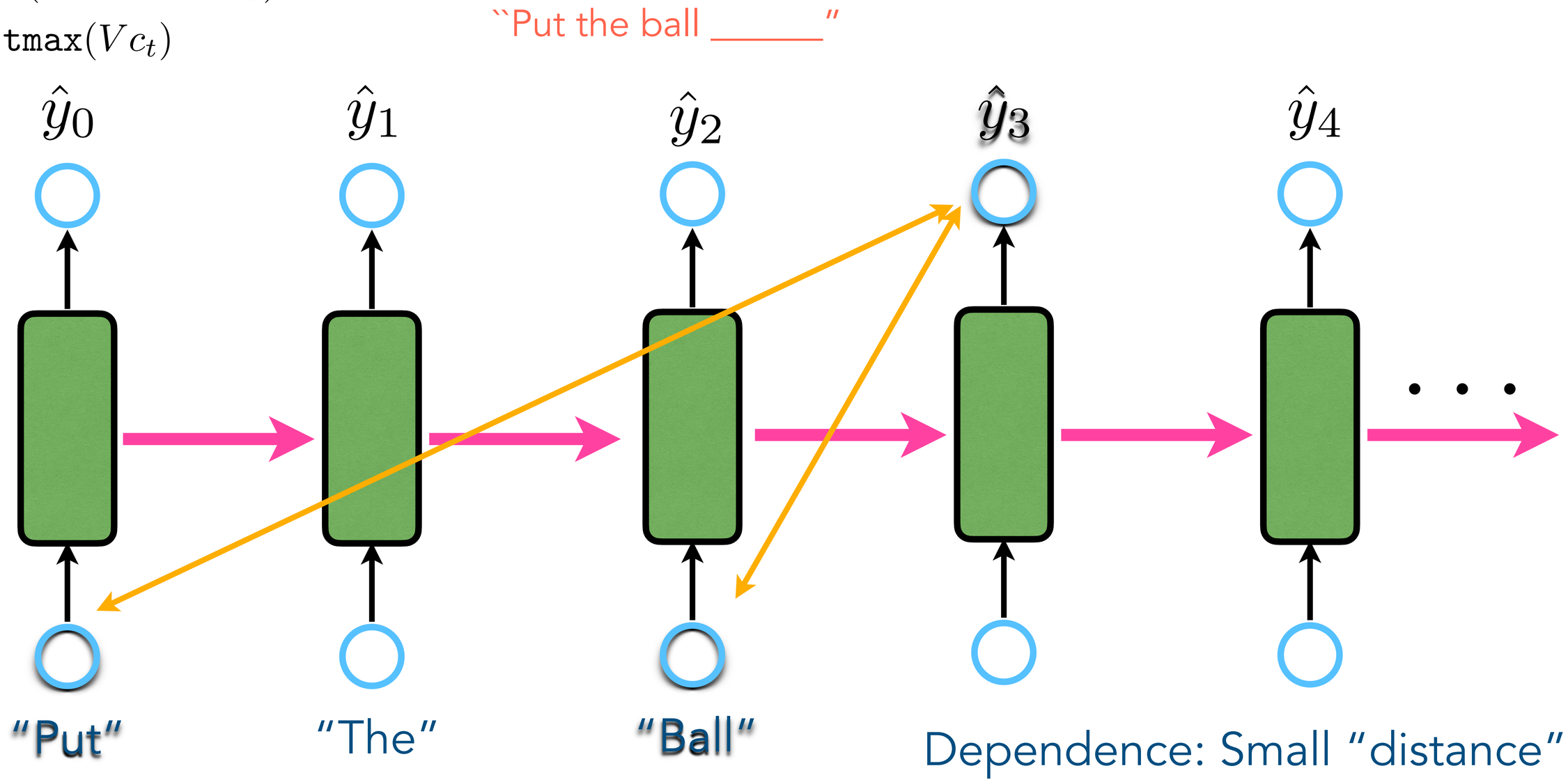
``Put the ball _____"



Long-term dependencies

$$c_t = \tanh\left(Ux_t + Wc_{t-1}\right)$$

$$y_t = \texttt{softmax}(Vc_t)$$

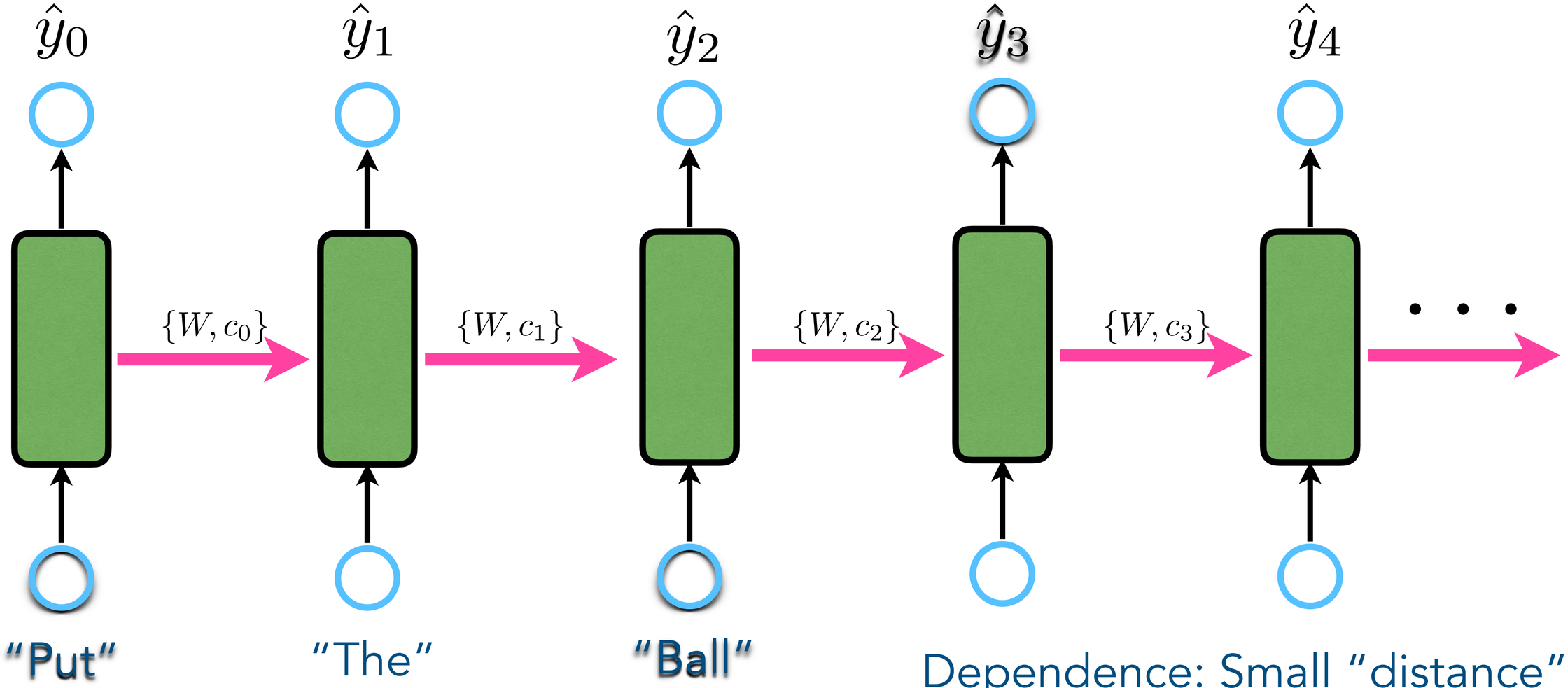


Long-term dependencies

$$c_t = \tanh\left(Ux_t + Wc_{t-1}\right)$$

$$y_t = \texttt{softmax}(Vc_t)$$

``Put the ball _____"

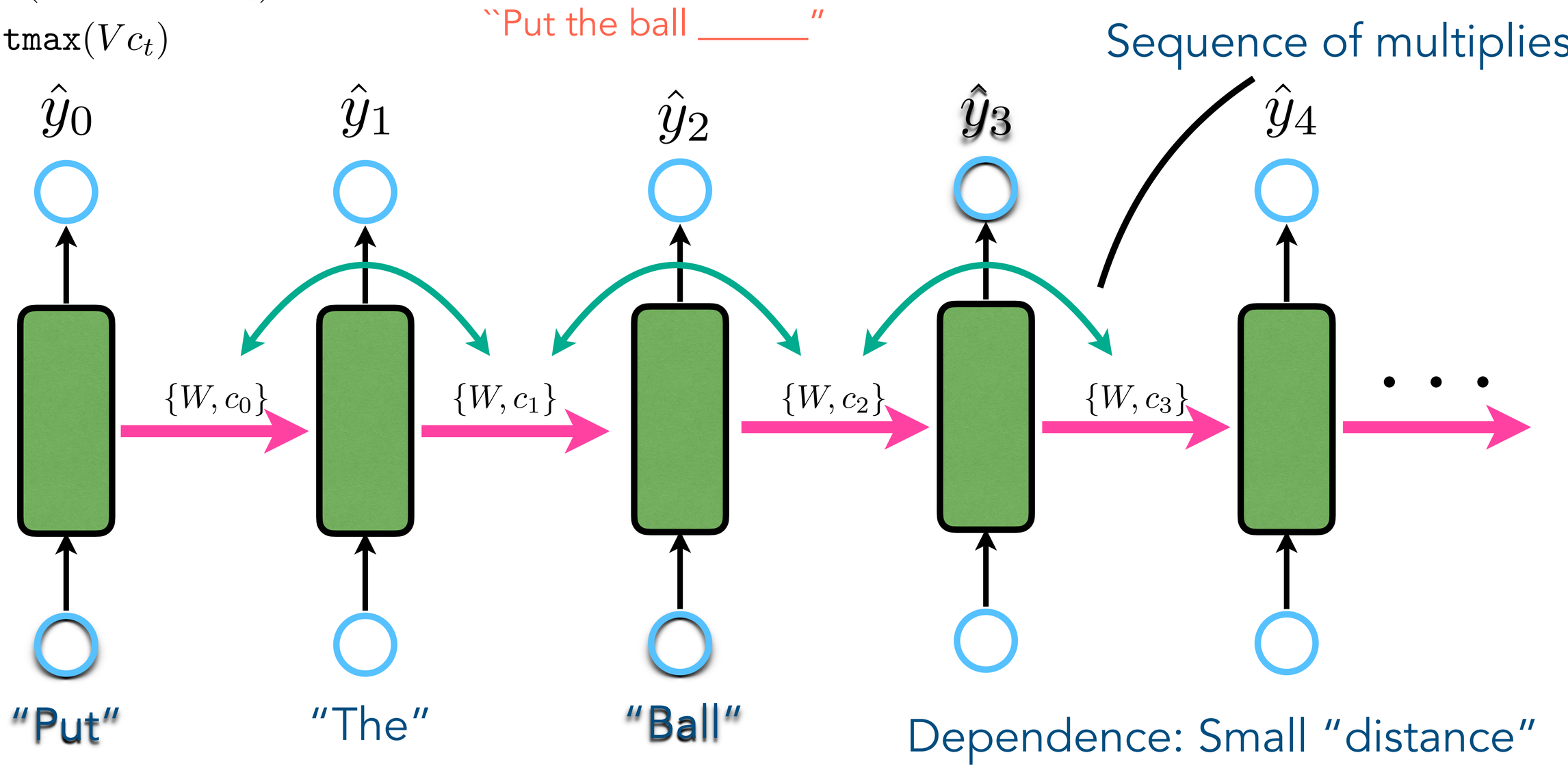


Dependence: Small "distance"

Long-term dependencies

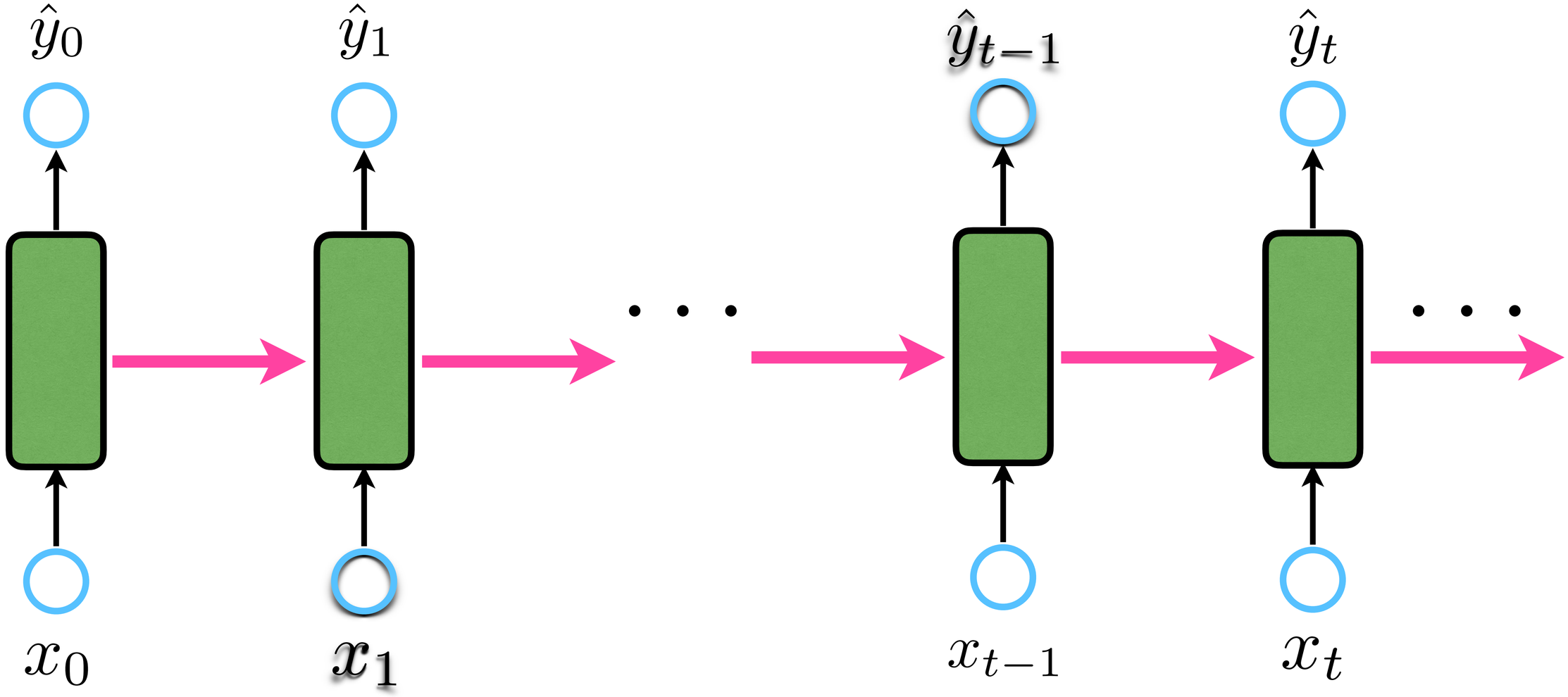
$$c_t = \tanh\left(Ux_t + Wc_{t-1}\right)$$

$$y_t = \texttt{softmax}(Vc_t)$$



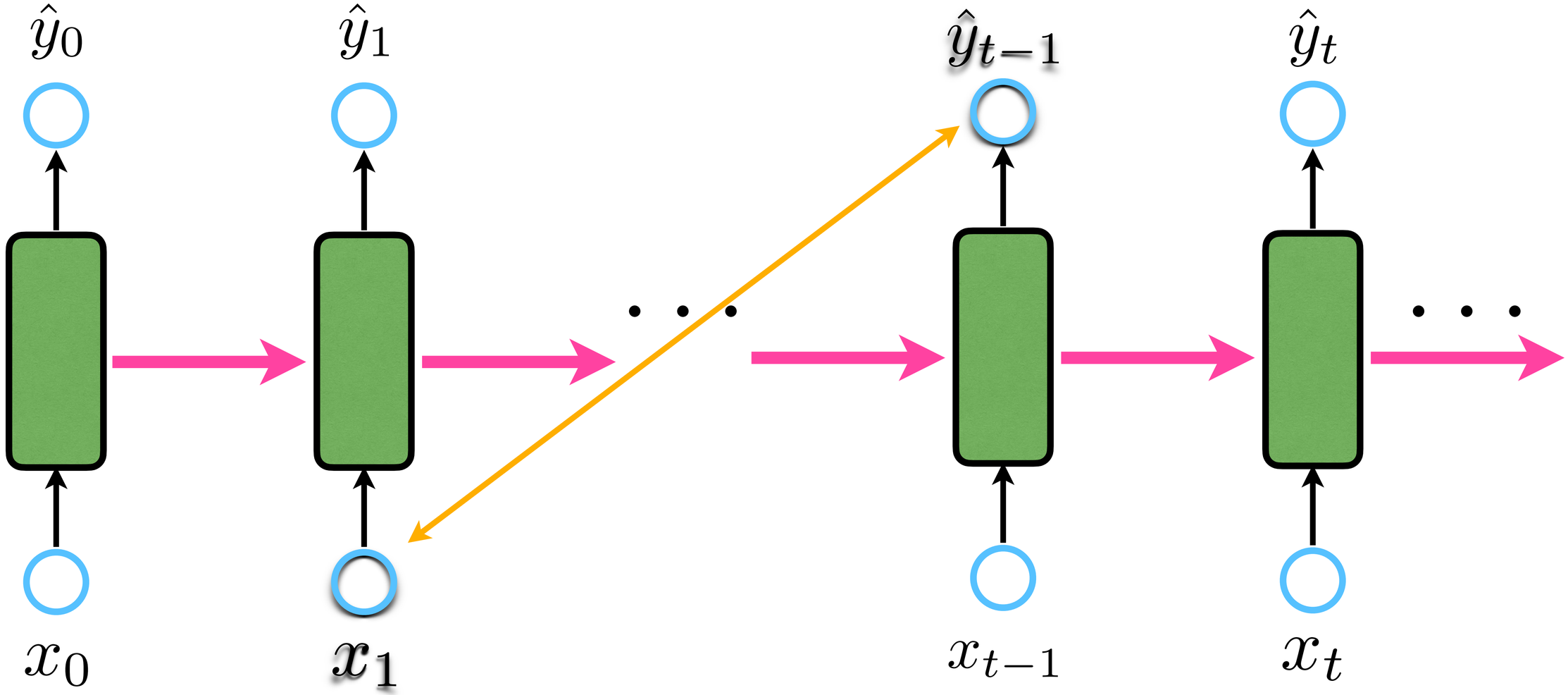
Long-term dependencies

``My name is Tasos and I'm from **Greece**. I have been studying there since elementary school up to university level. I speak _____ fluently"



Long-term dependencies

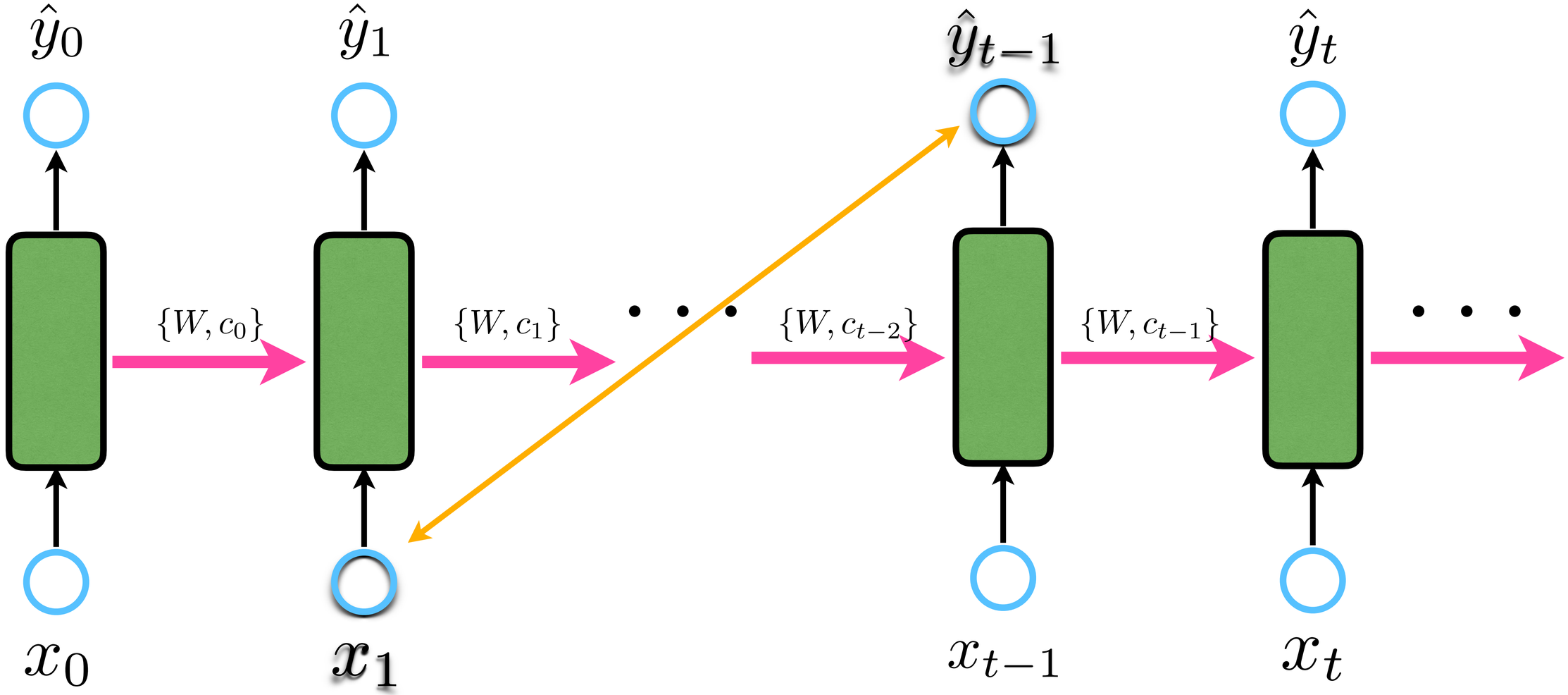
``My name is Tasos and I'm from **Greece**. I have been studying there since elementary school up to university level. I speak _____ fluently"



Dependence: Long "distance"

Long-term dependencies

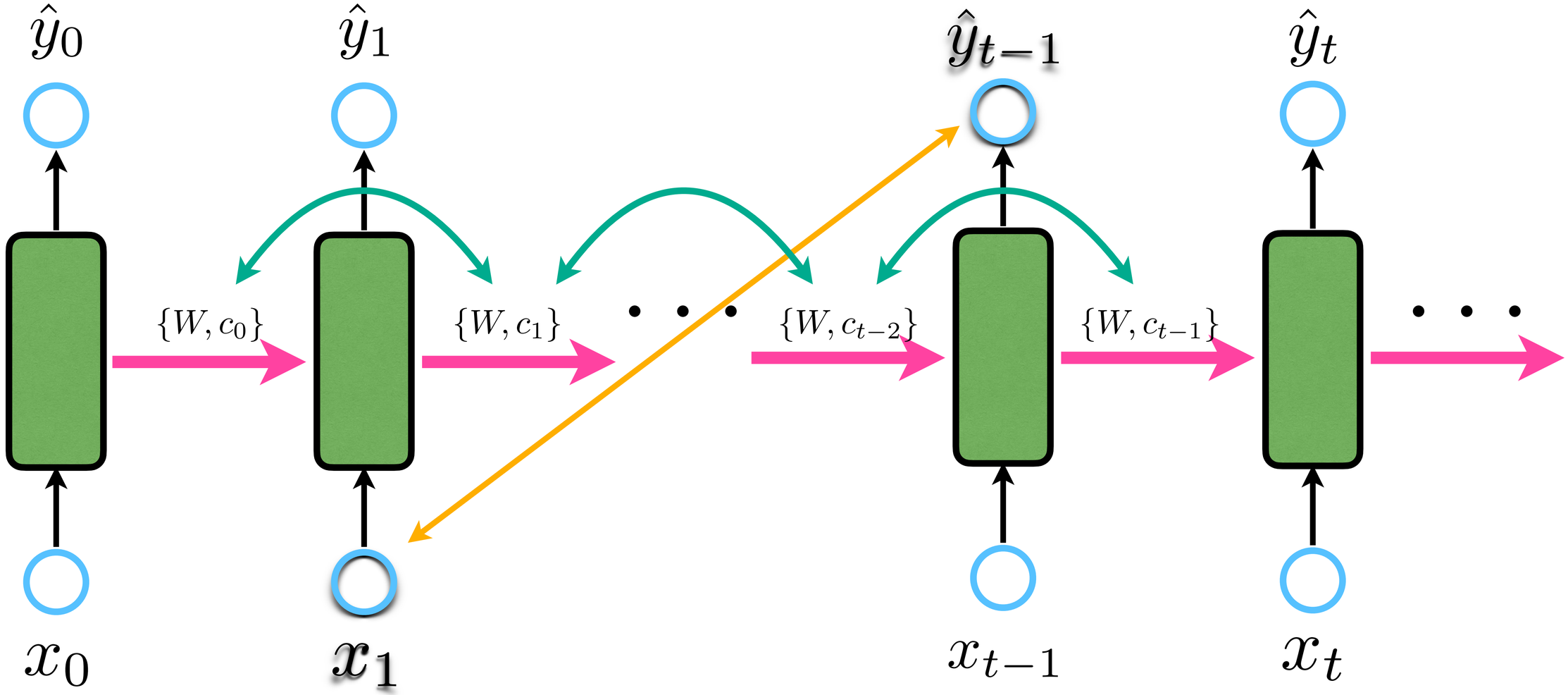
``My name is Tasos and I'm from **Greece**. I have been studying there since elementary school up to university level. I speak _____ fluently"



Dependence: Long "distance"

Long-term dependencies

``My name is Tasos and I'm from **Greece**. I have been studying there since elementary school up to university level. I speak _____ fluently"

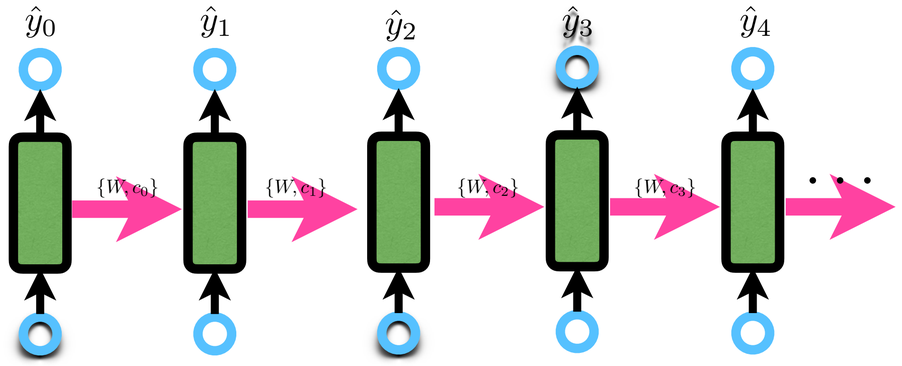


Dependence: Long "distance"

Long-term dependencies

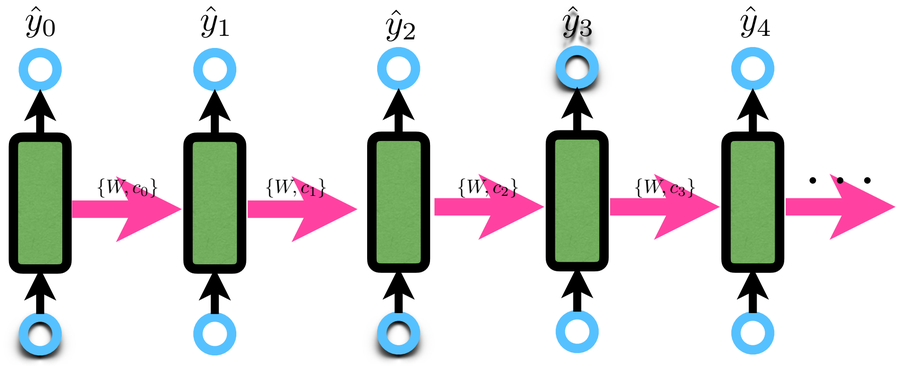
``My name is Tasos and I'm from Greece. I have been studying there since elementary school up to university level. I speak _____ fluently"

- This leads to the problem of vanishing /exploding gradients



Long-term dependencies

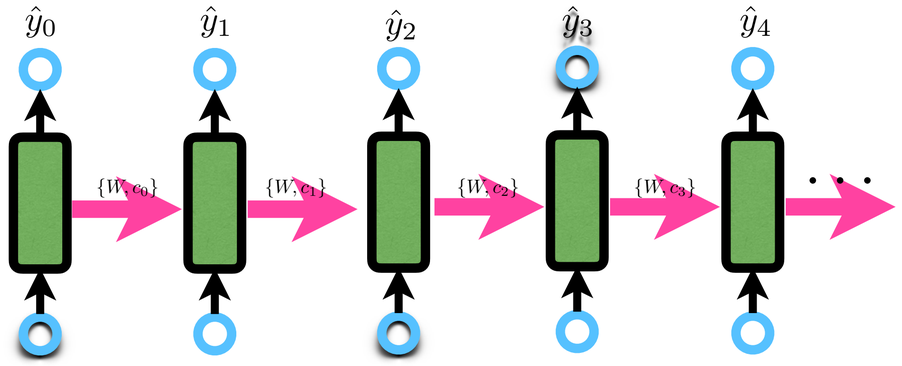
``My name is Tasos and I'm from Greece. I have been studying there since elementary school up to university level. I speak _____ fluently"



- This leads to the problem of vanishing /exploding gradients
- Multiplying many numbers in sequence (many Recurrence steps), introduces errors

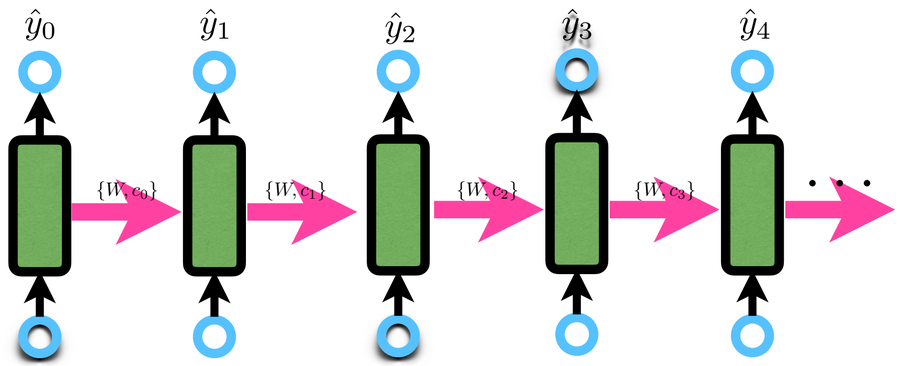
Long-term dependencies

``My name is Tasos and I'm from Greece. I have been studying there since elementary school up to university level. I speak _____ fluently"



- This leads to the problem of vanishing /exploding gradients
- Multiplying many numbers in sequence (many Recurrence steps), introduces errors
  - Either numbers get larger and larger (out of control)
  - Or small numbers at any stage makes the whole process vanish

Unfolding RNNs

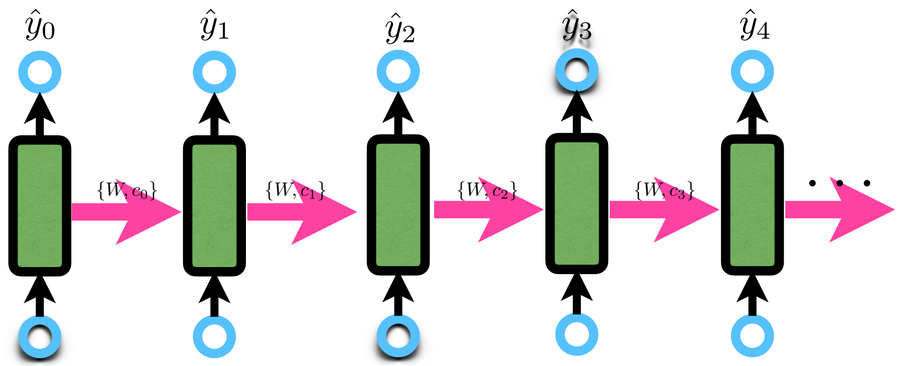


- Backpropagation through time!

  Unfolding back propagation leads to long memories which further leads to vanishing/exploding gradients

  There are tricks that partially solve the problem (rescaling gradients, do fewer time steps)

Unfolding RNNs

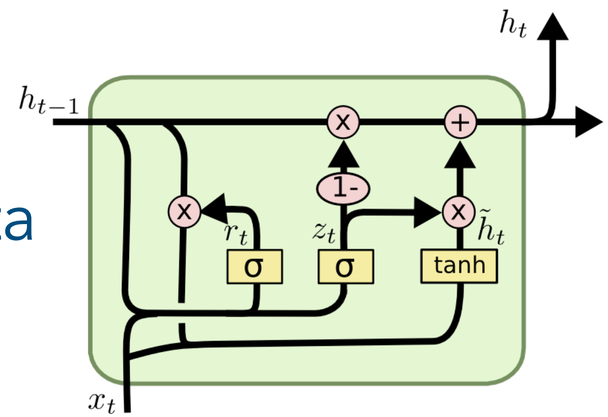


- Backpropagation through time!

  Unfolding back propagation leads to long memories which further leads to vanishing/exploding gradients

  There are tricks that partially solve the problem (rescaling gradients, do fewer time steps)

- Long Short Term Memory (LSTM)
  - Modified version of RNNs
  - Easier to remember/forget past data
  - Vanishing gradients resolved



Control gates

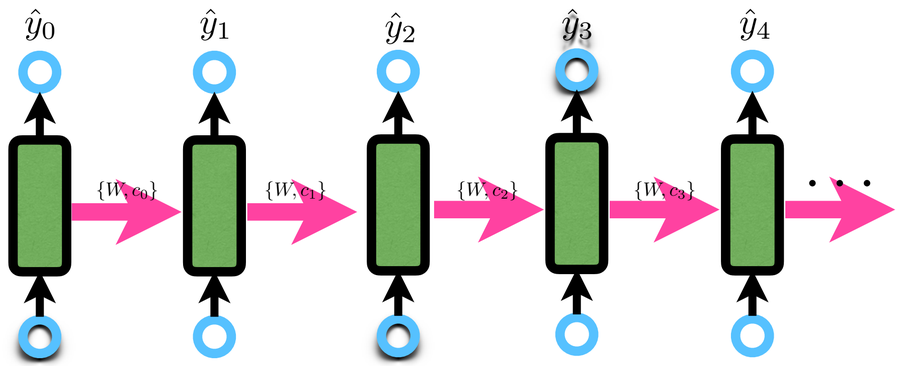
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Unfolding RNNs

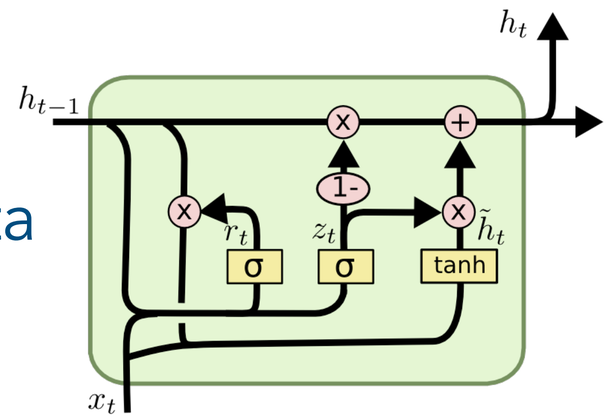


- Backpropagation through time!

  Unfolding back propagation leads to long memories which further leads to vanishing/exploding gradients

  There are tricks that partially solve the problem (rescaling gradients, do fewer time steps)

- Long Short Term Memory (LSTM)
  - Modified version of RNNs
  - Easier to remember/forget past data
  - Vanishing gradients resolved



Control gates

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- LSTMs use gated cells to track information as it flows through many steps

Transformers everywhere

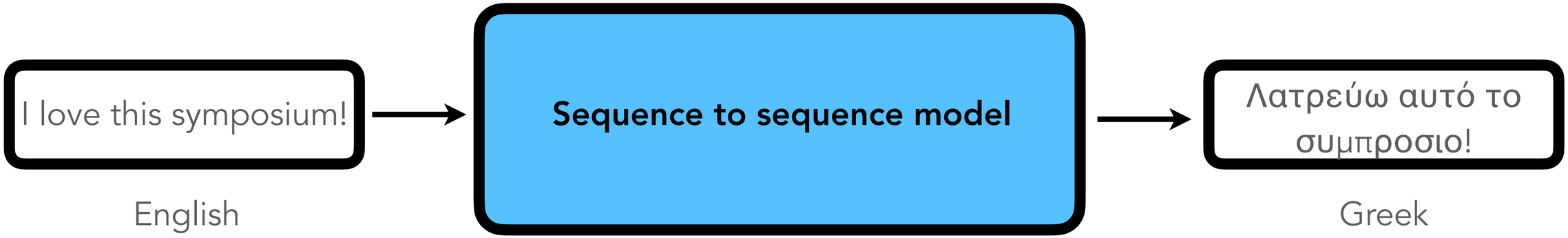
The notion of self-attention

- Consider Sequence-to-Sequence (seq2seq) models

  (Machine translation, text summarization, image captioning..)

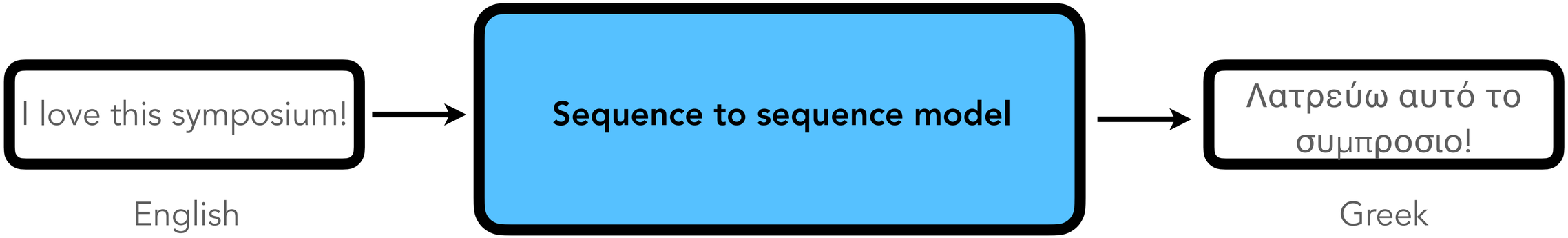
The notion of self-attention

- Consider Sequence-to-Sequence (seq2seq) models
  (Machine translation, text summarization, image captioning..)



The notion of self-attention

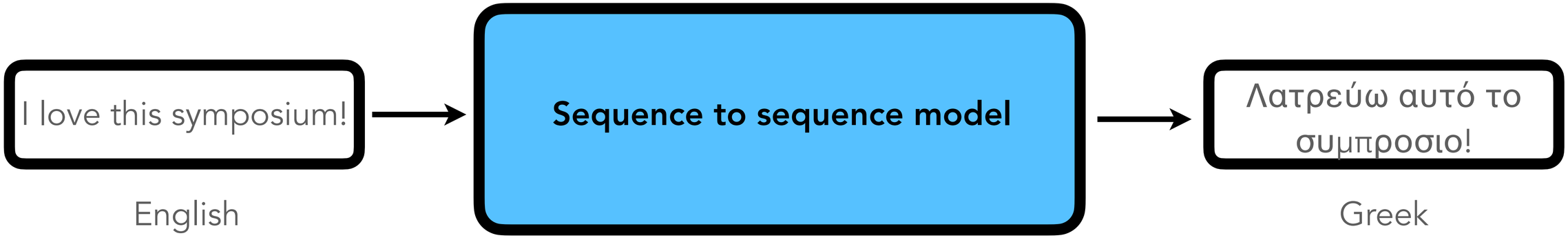
- Consider Sequence-to-Sequence (seq2seq) models
  (Machine translation, text summarization, image captioning..)



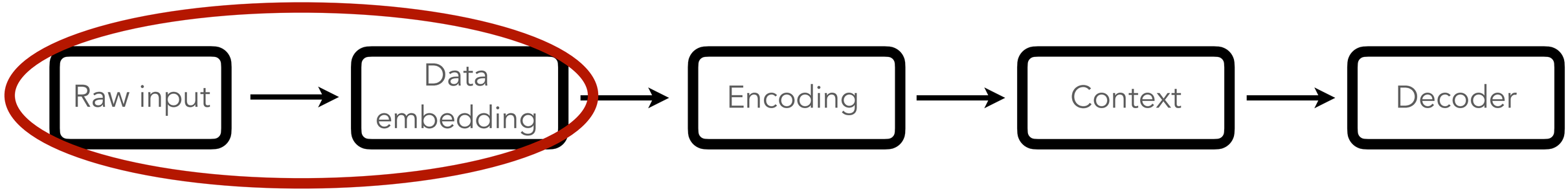
- What happens under the ``hood'':

The notion of self-attention

- Consider Sequence-to-Sequence (seq2seq) models
  (Machine translation, text summarization, image captioning..)



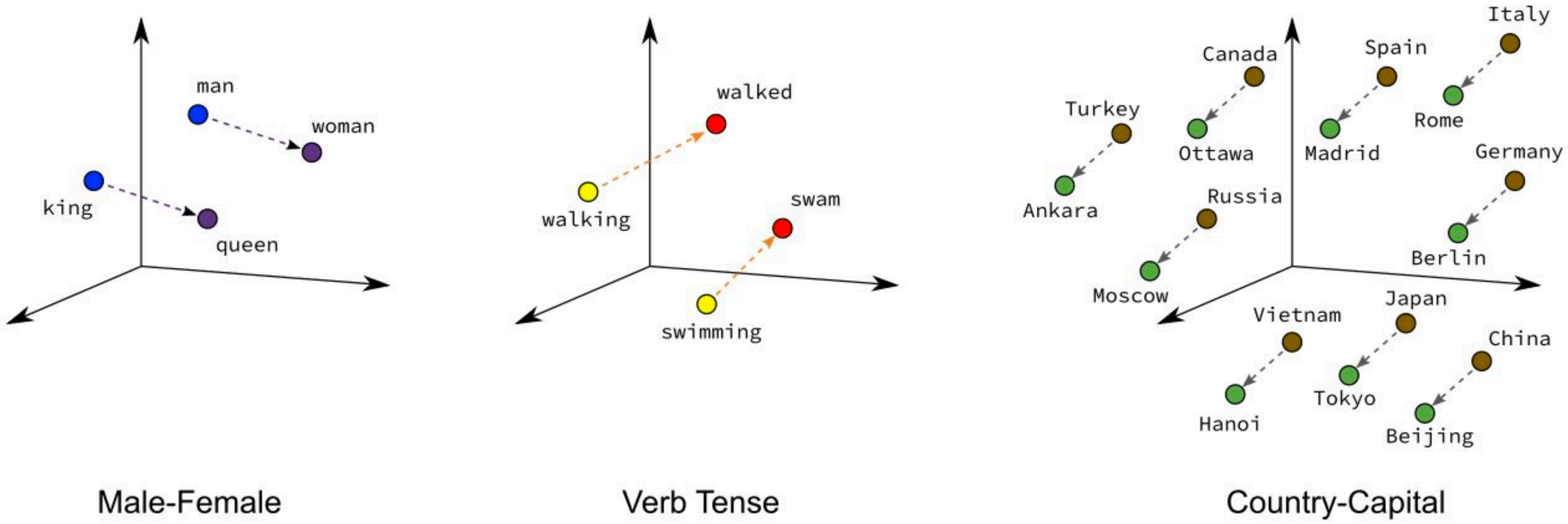
- What happens under the ``hood'':



Data representation and word embeddings

Data representation and word embeddings

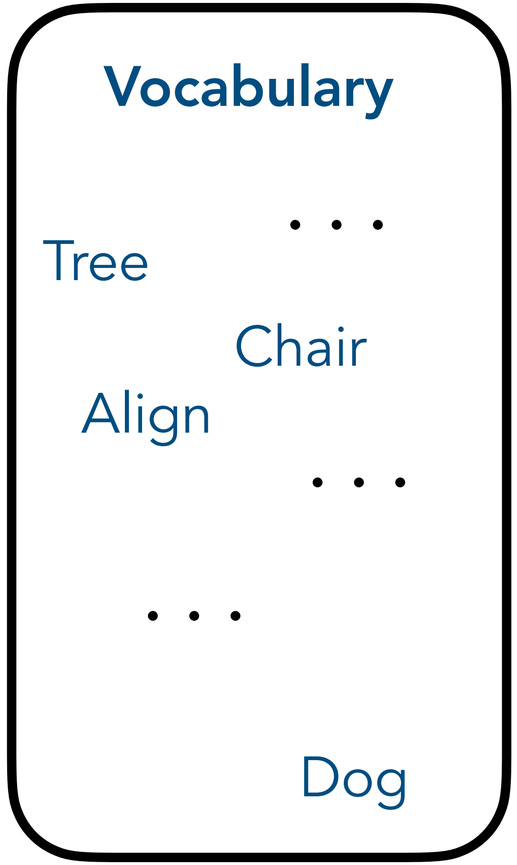
- Mechanisms that turn raw text pieces (words, sentences, etc) into vectors



Tribute to
Google Developers

Data representation and word embeddings

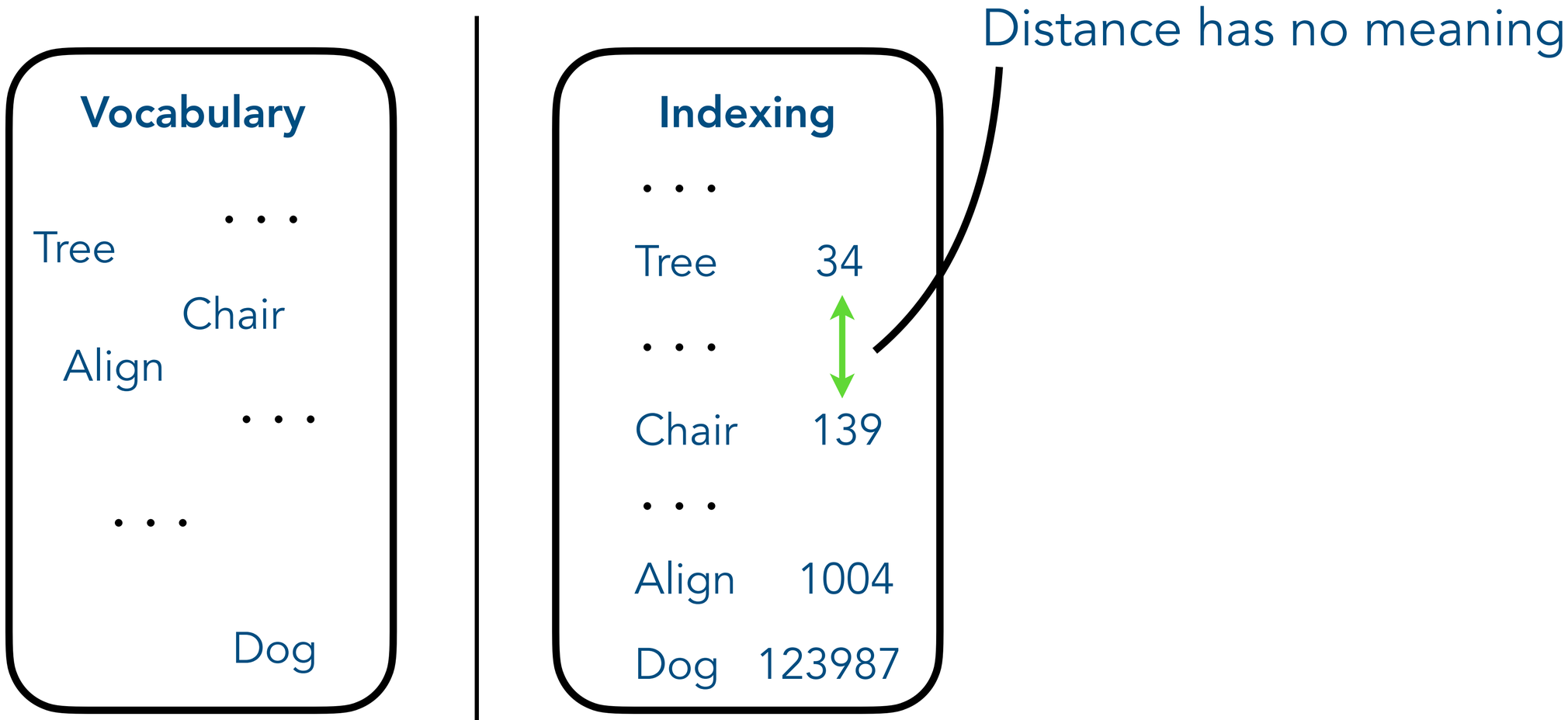
- Mechanisms that turn raw text pieces (words, sentences, etc) into vectors



(NNs cannot interpret words)

Data representation and word embeddings

- Mechanisms that turn raw text pieces (words, sentences, etc) into vectors



(NNs cannot interpret words)

Data representation and word embeddings

- Mechanisms that turn raw text pieces (words, sentences, etc) into vectors

Vocabulary

. . .

Tree

Chair

Align

. . .

. . .

Dog

Indexing

. . .

Tree	34
. . .	
Chair	139
. . .	
Align	1004
Dog	123987

Classical 1-hot embedding

Tree = [0, 0, 0, …, 1, 0, … 0]

Dog = [0, 0, 1, …, 0, 0, … 0]

(Extremely large vector)

Data representation and word embeddings

- Mechanisms that turn raw text pieces (words, sentences, etc) into vectors

Vocabulary

. . .

Tree

Chair

Align

. . .

. . .

Dog

Indexing

. . .

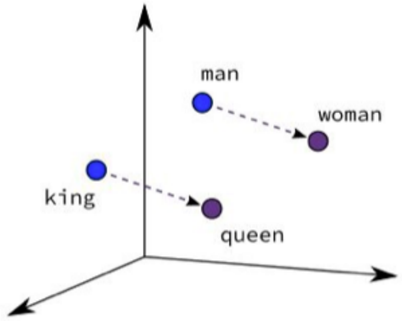
Tree	34
. . .	
Chair	139
. . .	
Align	1004
Dog	123987

Classical 1-hot embedding

Tree = [0, 0, 0, …, 1, 0, … 0]

Dog = [0, 0, 1, …, 0, 0, … 0]

(Extremely large vector)



$$x_{\text{emb}} = Vx$$

Data representation and word embeddings

- Mechanisms that turn raw text pieces (words, sentences, etc) into vectors

Vocabulary

. . .

Tree

Chair

Align

. . .

. . .

Dog

Indexing

. . .

Tree	34
. . .	
Chair	139
. . .	
Align	1004
Dog	123987

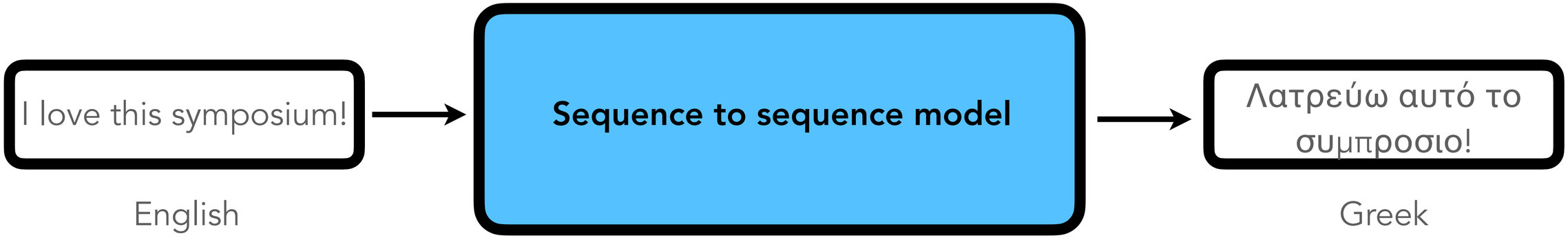
features

Number of vocabulary words

0.5	0.27	0.36	0.9	
0.81	0.12	0.03	0.81	
0.72	-0.5	0.52	0.65	← I
				← Love
0.74	0.31	0.47	0.03	← This
0.19	1.0	-0.7	0.37	← Workshop
0.28	0.4	0.99	0.55	

The notion of self-attention

- Consider Sequence-to-Sequence (seq2seq) models
  (Machine translation, text summarization, image captioning..)

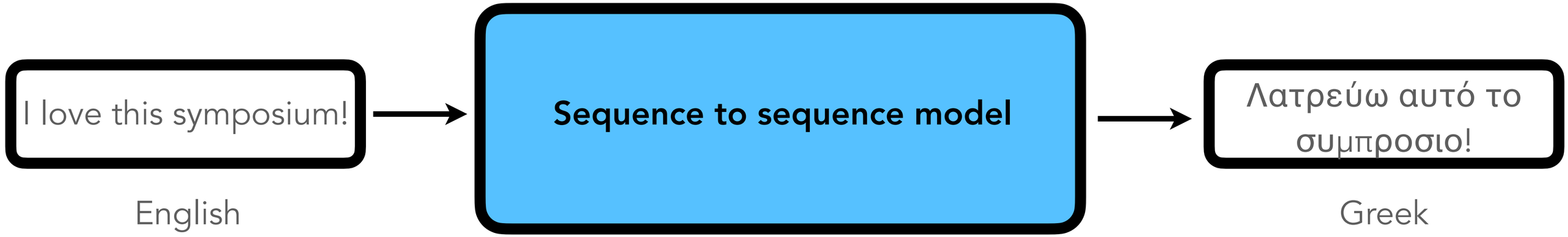


- What happens under the ``hood'':

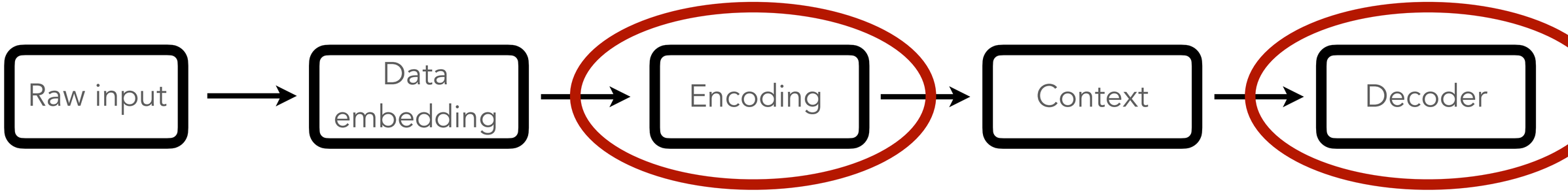
(Often data embedding is part of encoding)

The notion of self-attention

- Consider Sequence-to-Sequence (seq2seq) models
  (Machine translation, text summarization, image captioning..)



- What happens under the ``hood'':



What is encoding?

What is decoding?

(Often data embedding is part of encoding)

Interlude

Encoder-Decoders

(And the ``mother'' of these ideas: autoencoders)

Autoencoders

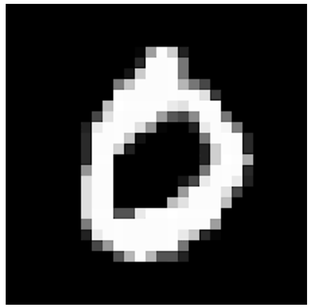
- An old idea that dates back to simple principal component analysis (PCA)

Autoencoders

- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''

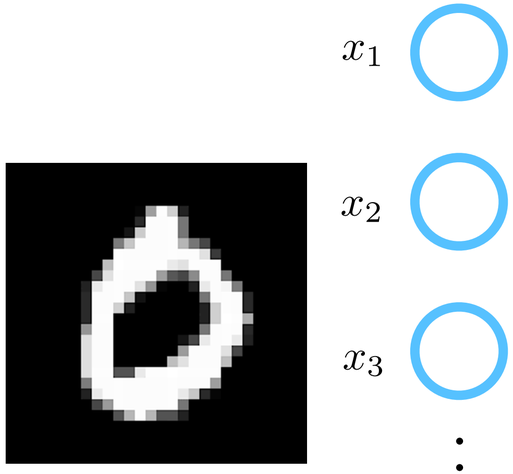
Autoencoders

- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''



Autoencoders

- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''



x_1

x_2

x_3

$\vdots$

x_n

Autoencoders

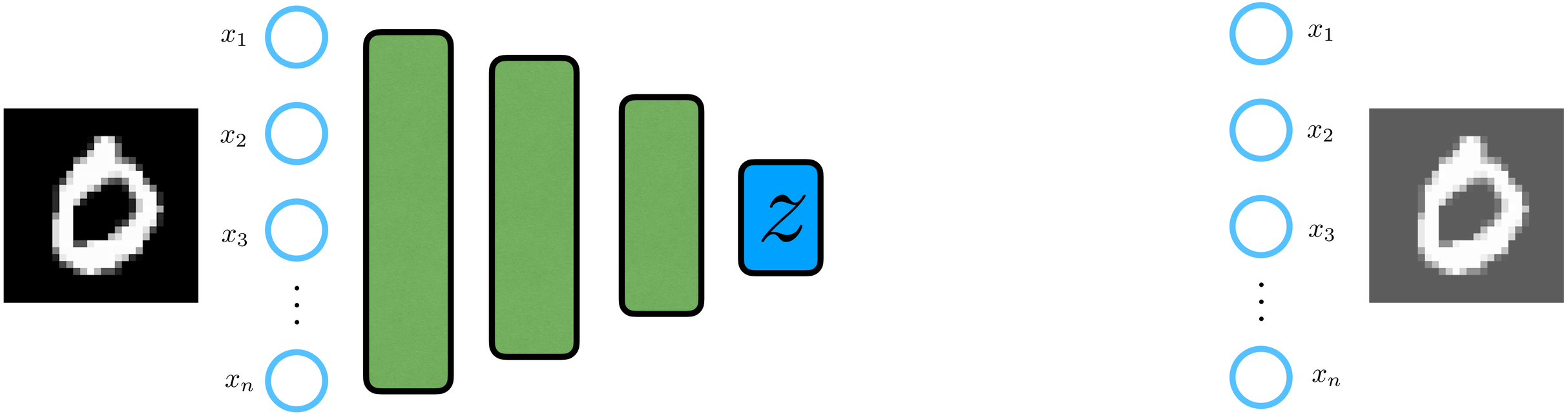
- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''

Autoencoders

- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''

Autoencoders

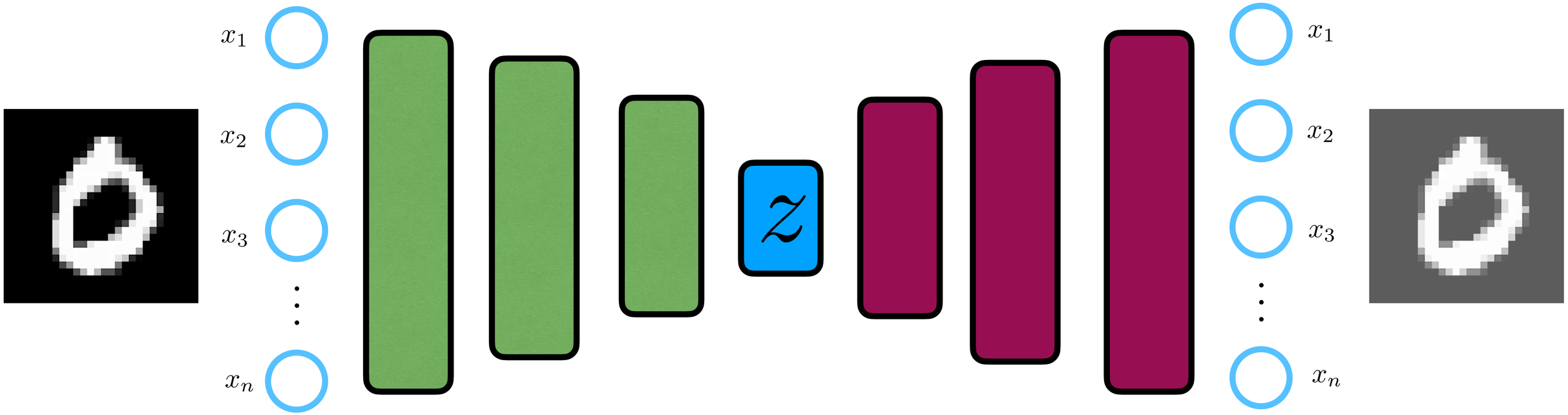
- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''



Learn a low-dimensional feature representation

Autoencoders

- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''

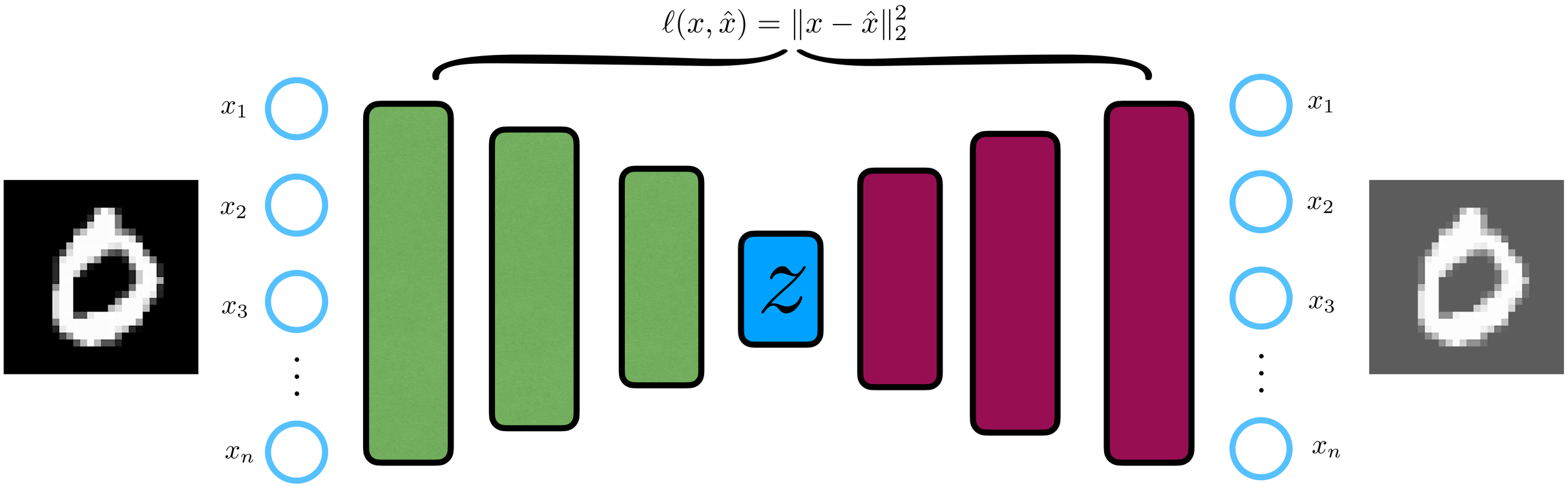


Learn a low-dimensional feature representation

Map back from latent space to a reconstructed image

Autoencoders

- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''

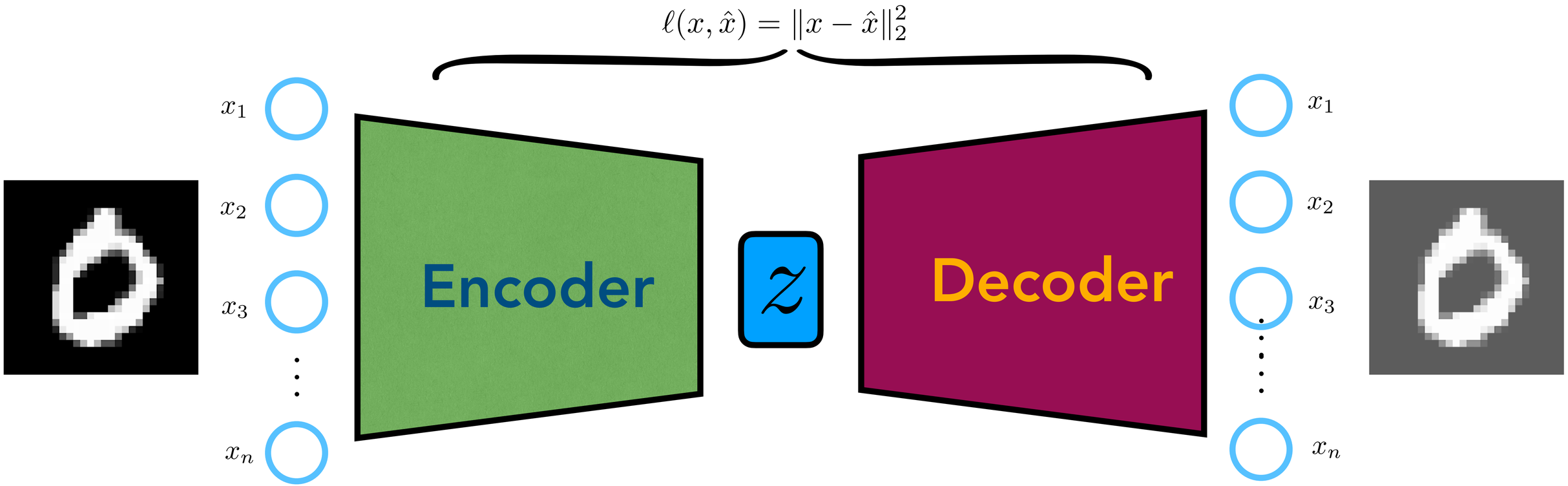


Learn a low-dimensional feature representation

Map back from latent space to a reconstructed image

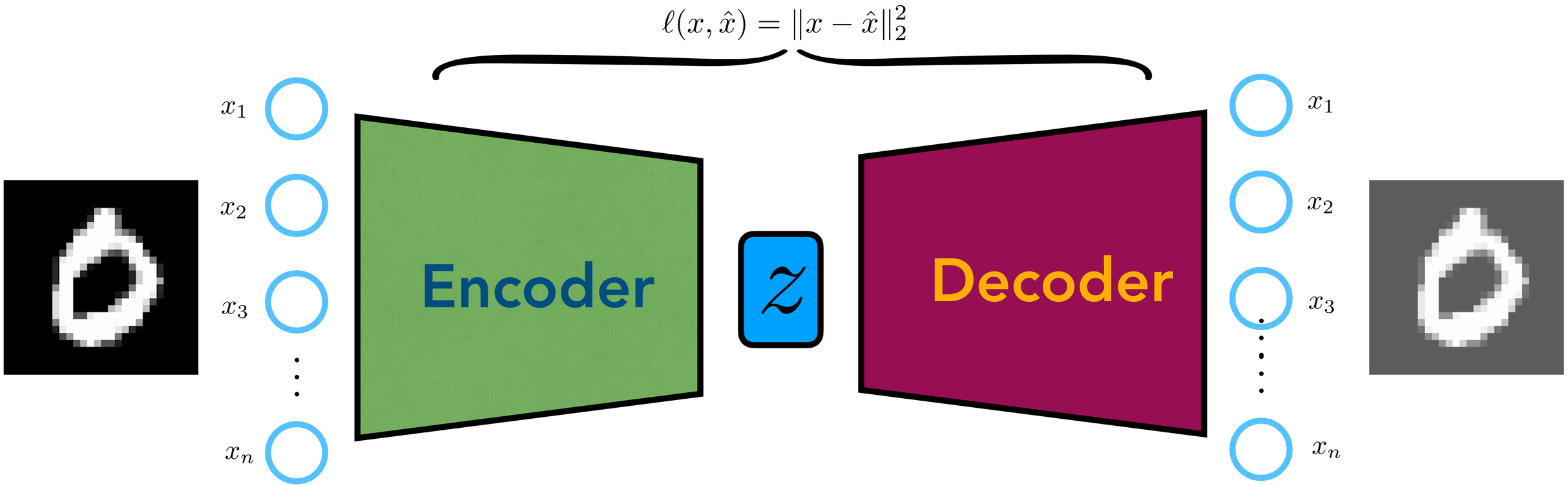
Autoencoders

- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''



Autoencoders

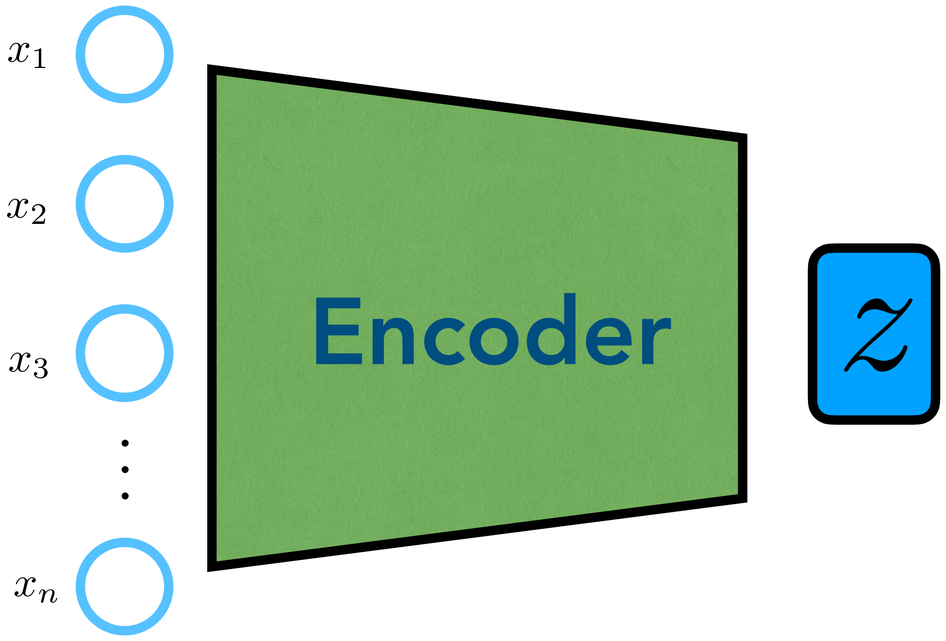
- An old idea that dates back to simple principal component analysis (PCA)
- Aim: extract the most useful info from data via a ``restricted representation''



- Bottleneck effect: Network is ``forced'' to learn a compressed latent rep.

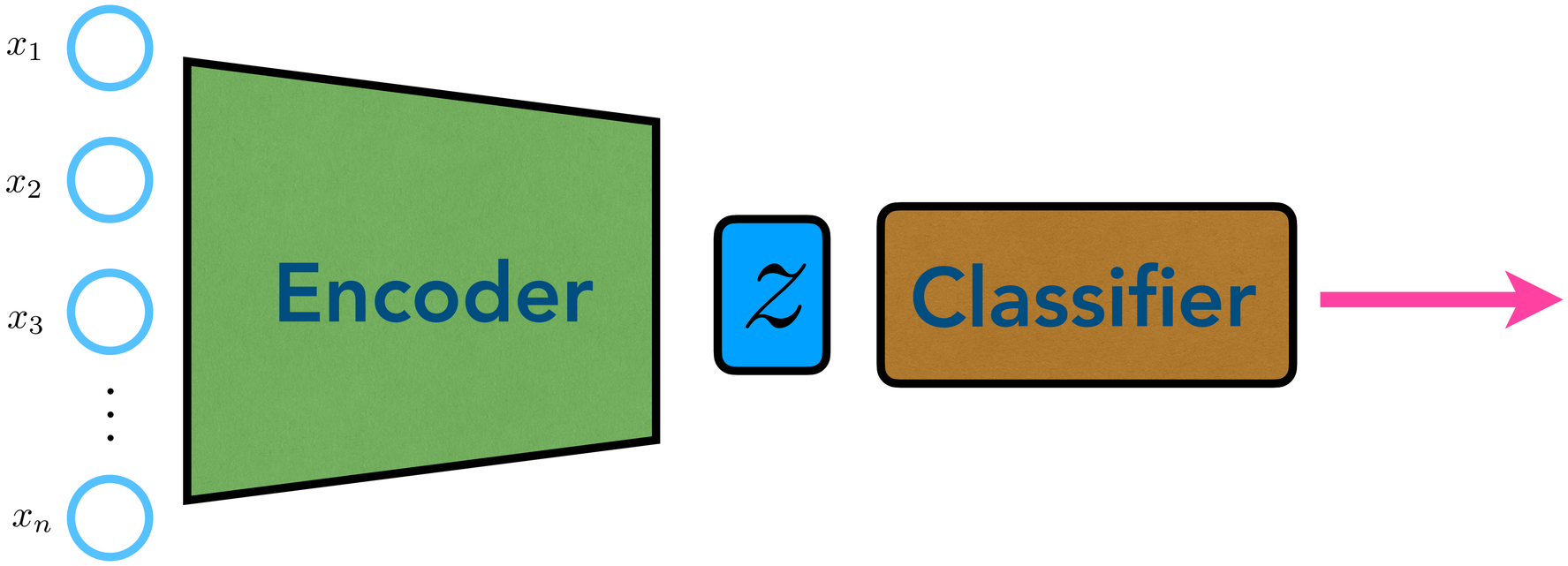
Encoder

- Can be used for downstream tasks



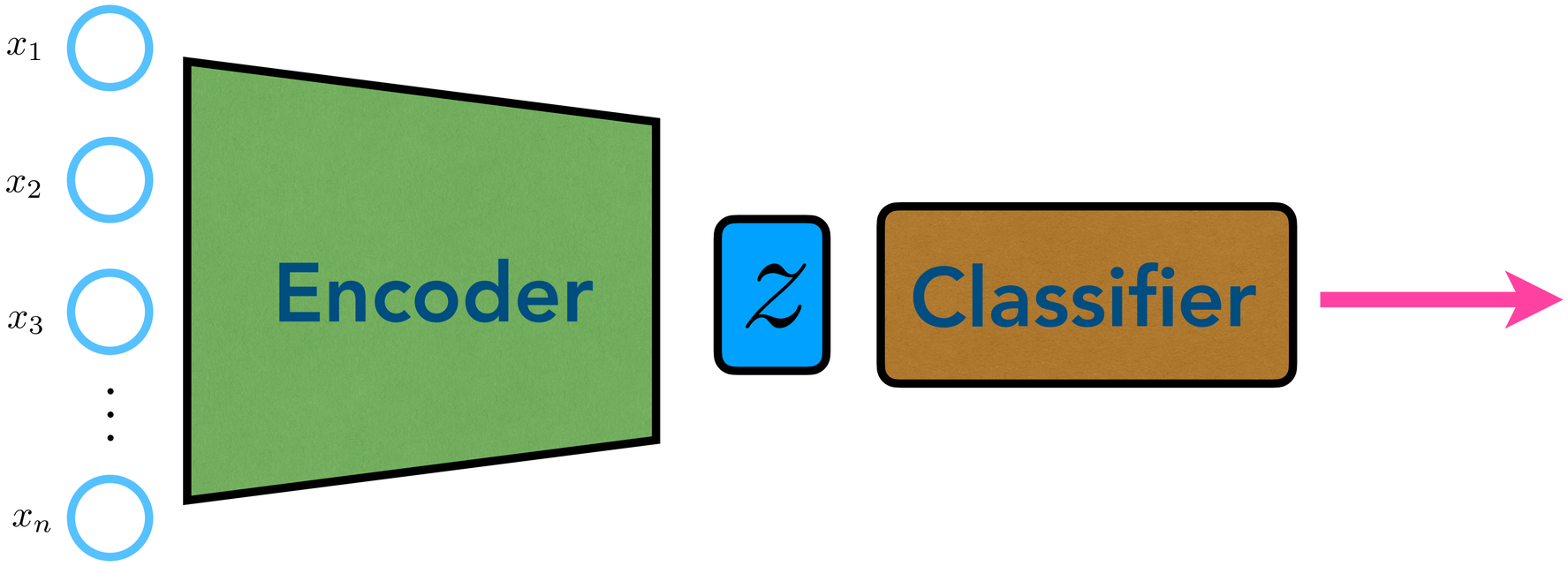
Encoder

- Can be used for downstream tasks



Encoder

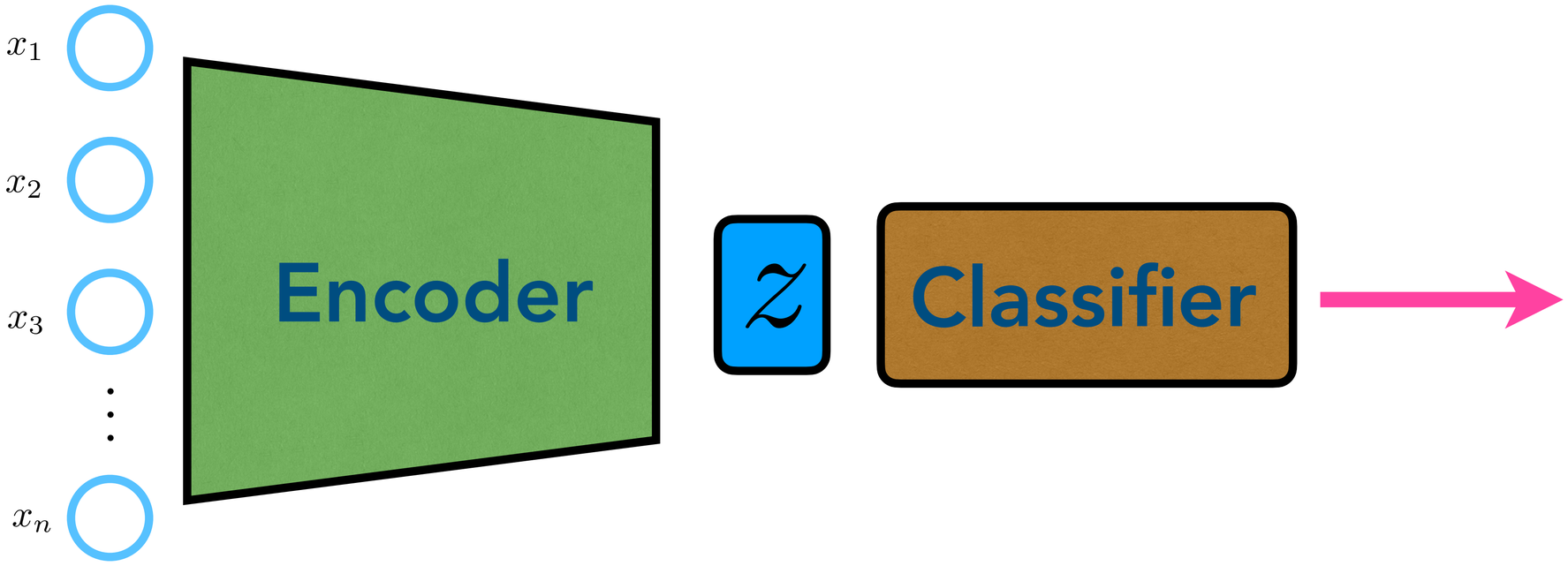
- Can be used for downstream tasks



- Encoder does a lot of work if it is trained already on a lot of unlabeled data

Encoder

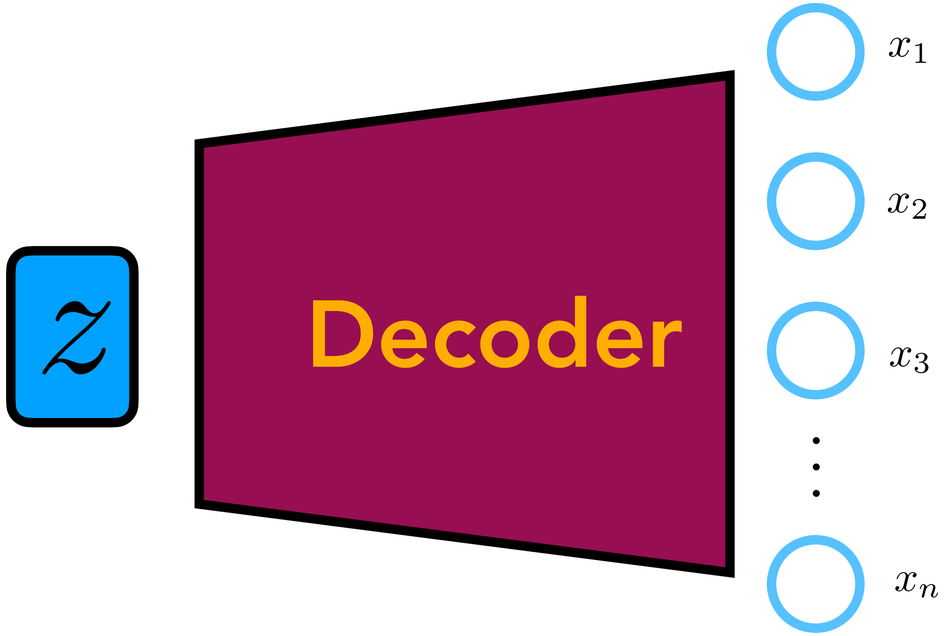
- Can be used for downstream tasks



- Encoder does a lot of work if it is trained already on a lot of unlabeled data
- This is the idea of fine tuning: encoder has good rough features and the classifier does fine-grained job to classify

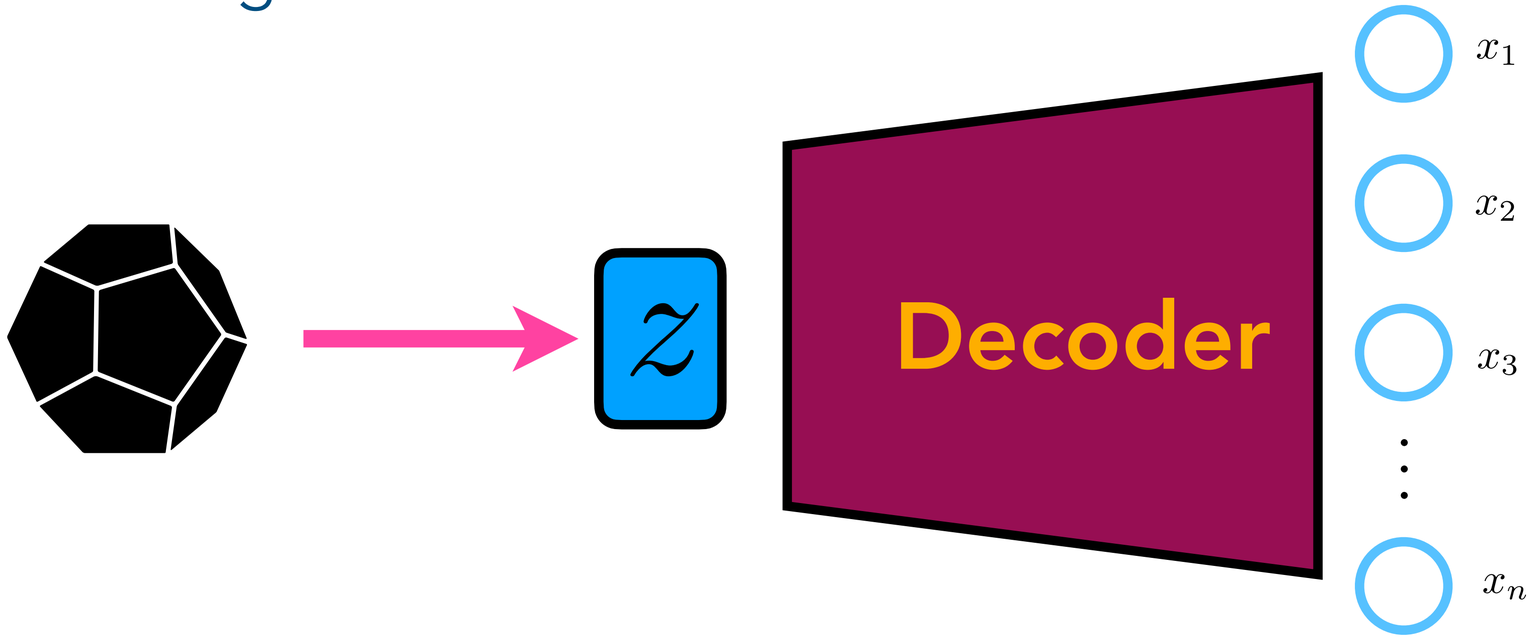
Decoder

- Can be used to generate new data



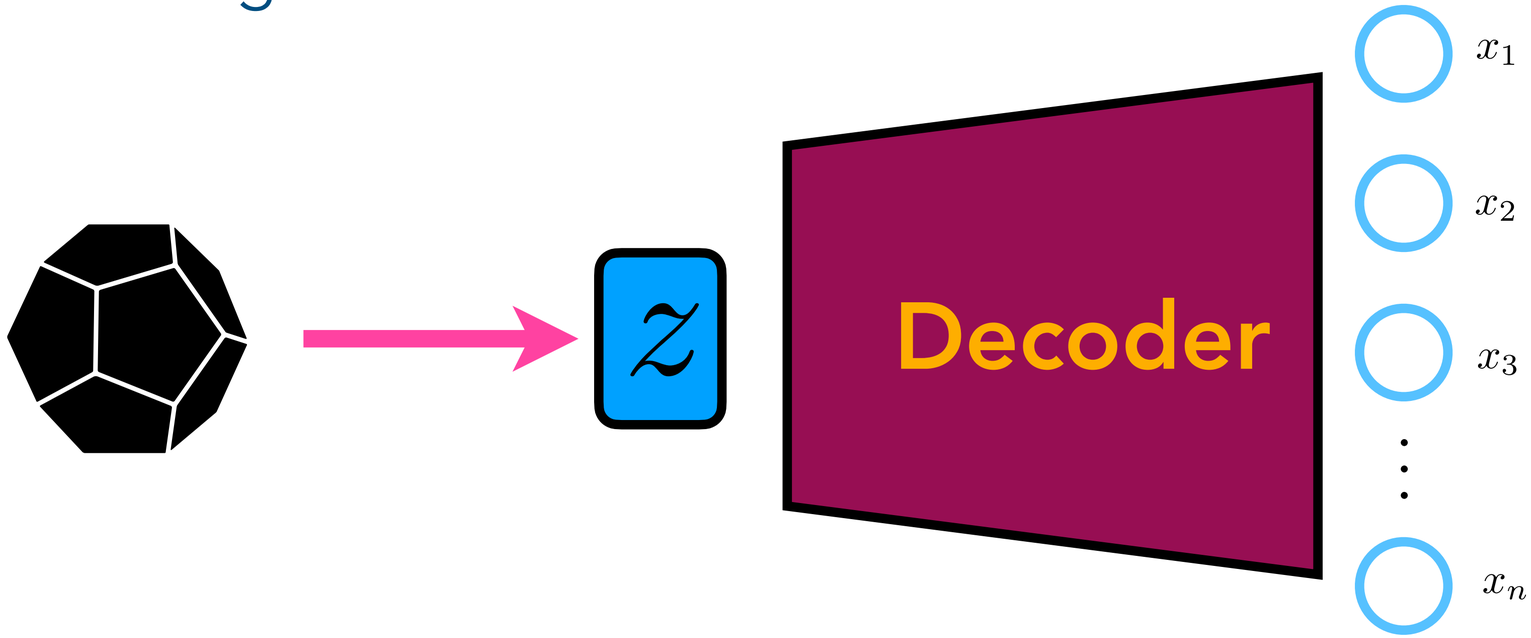
Decoder

- Can be used to generate new data



Decoder

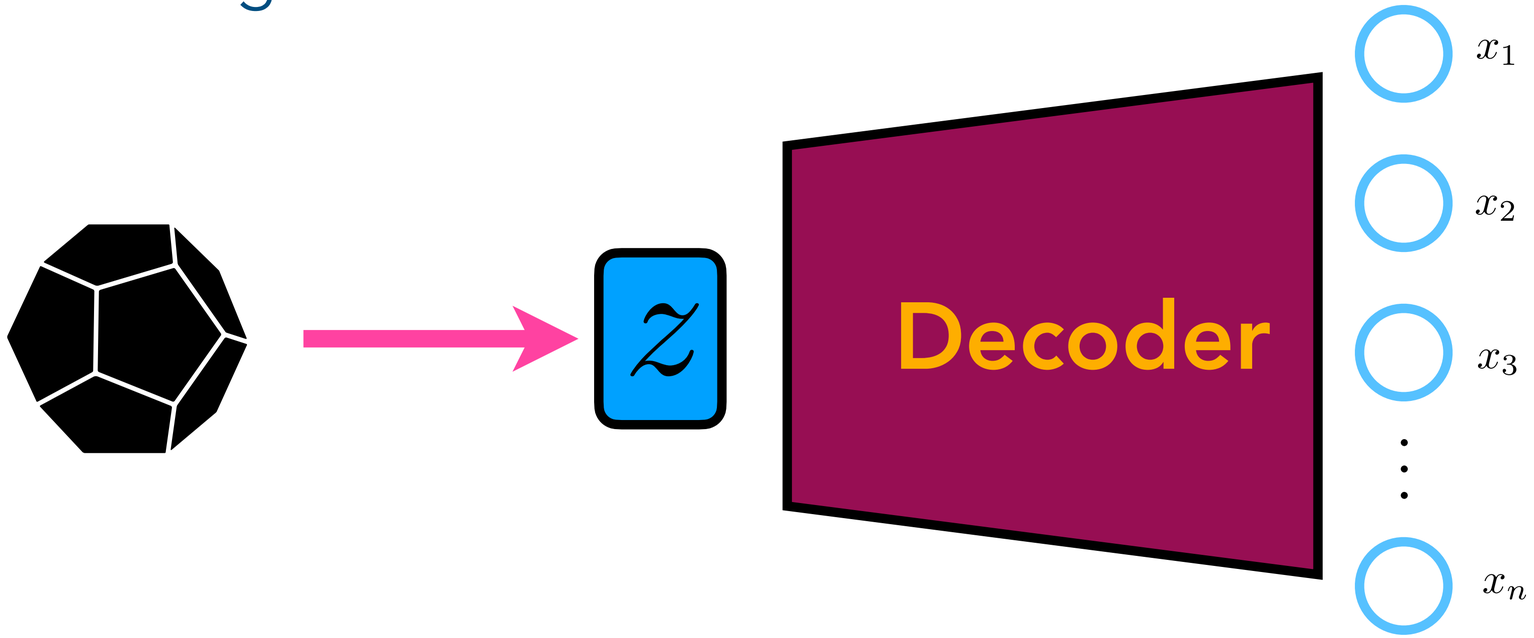
- Can be used to generate new data



- If we can find a good way to ``poke'' the z component, we could have data at the output

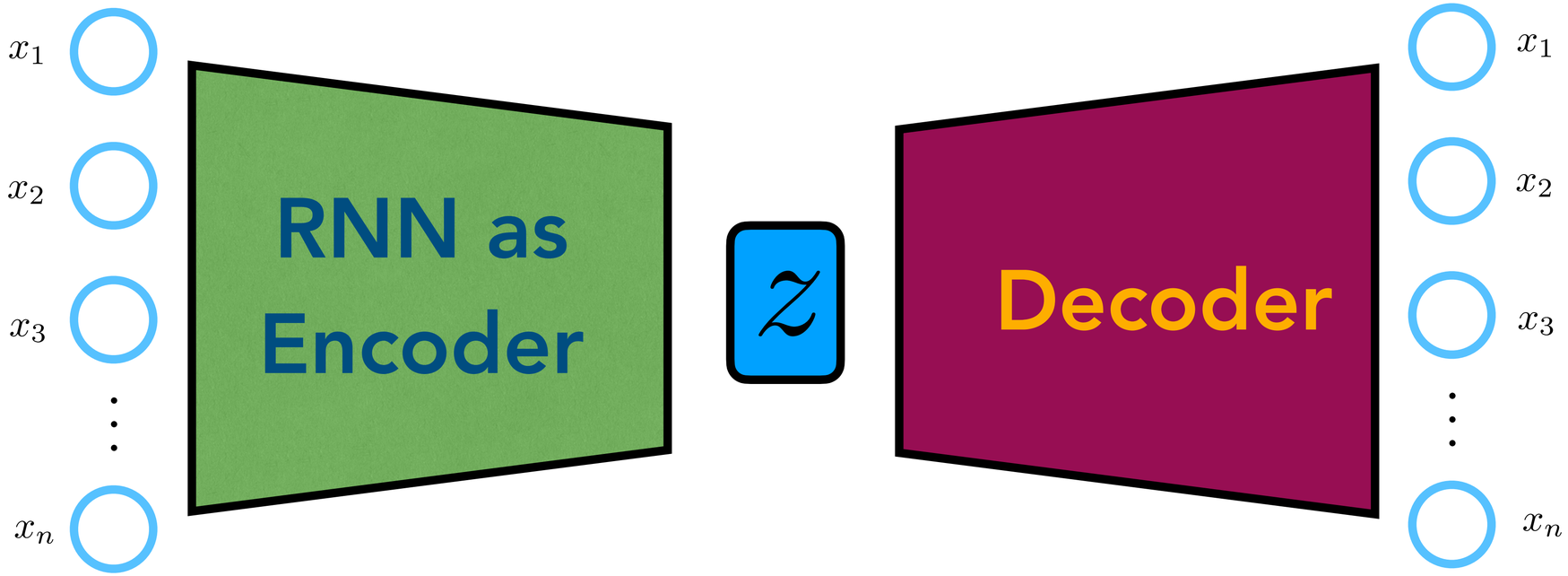
Decoder

- Can be used to generate new data



- If we can find a good way to ``poke'' the z component, we could have data at the output
- This issue motivates the research of Generative AI that we will cover later on

Encoder-decoder in sequence modeling



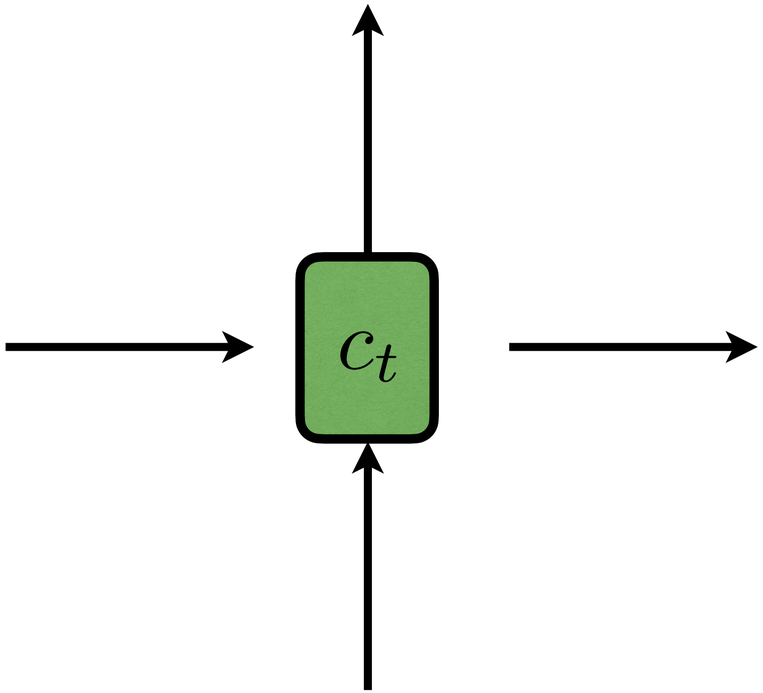
An encoder for seq2seq applications

An encoder for seq2seq applications

- Usually, a NN-based mechanism that takes an input a sequence and returns a context via a vector representation

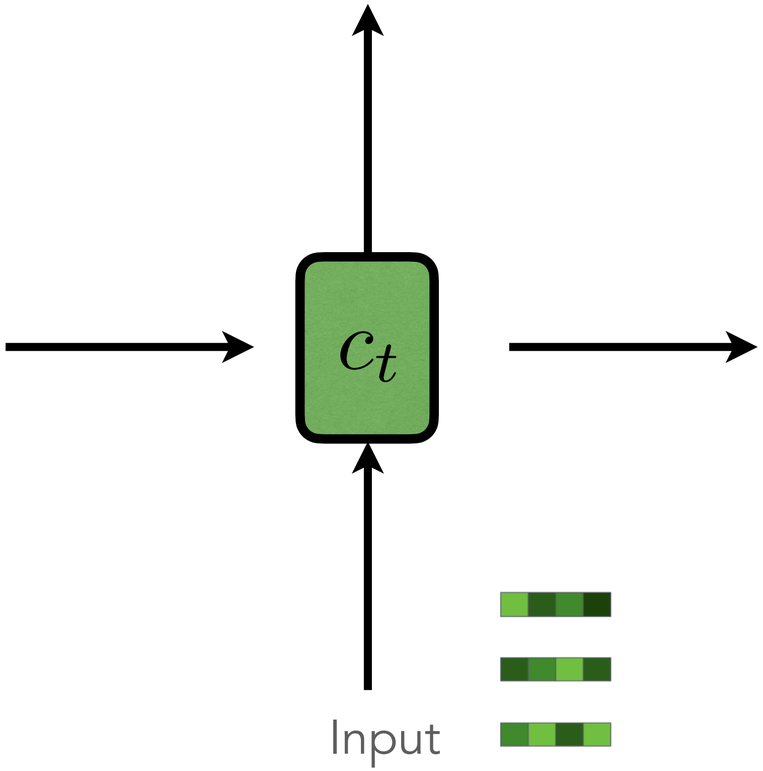
An encoder for seq2seq applications

- Usually, a NN-based mechanism that takes an input a sequence and returns a context via a vector representation
- A RNN architecture is perfect to simulate such environments



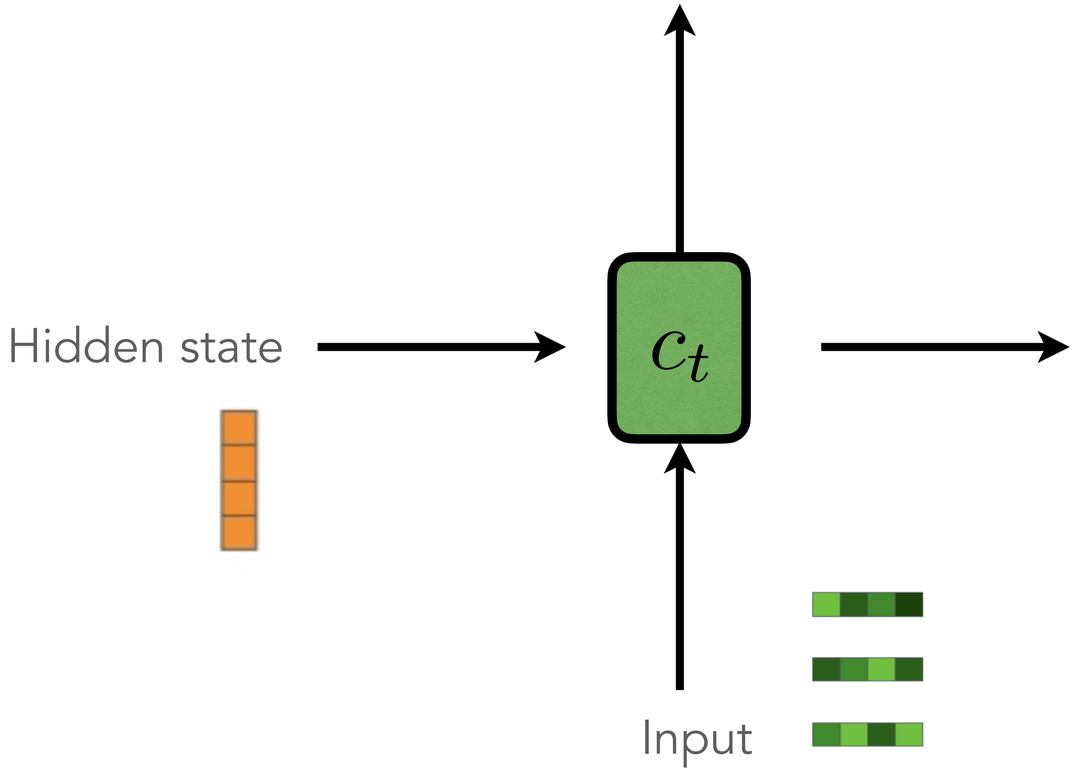
An encoder for seq2seq applications

- Usually, a NN-based mechanism that takes an input a sequence and returns a context via a vector representation
- A RNN architecture is perfect to simulate such environments



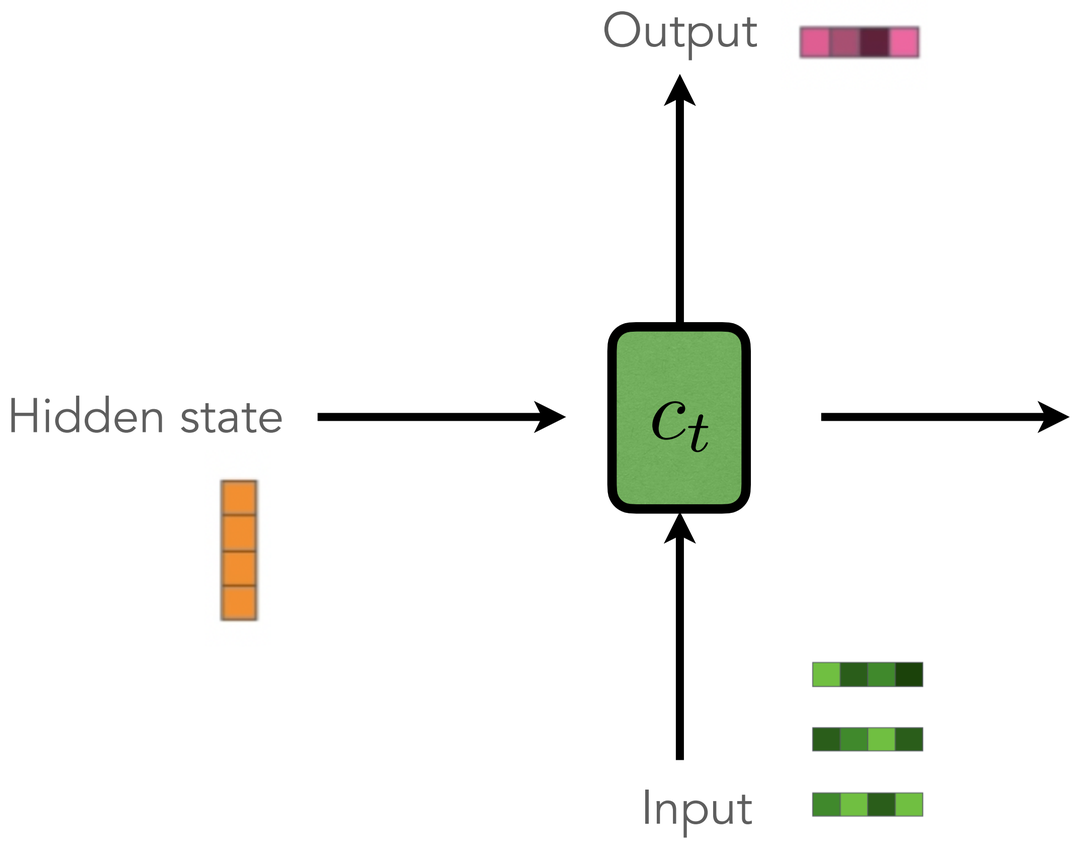
An encoder for seq2seq applications

- Usually, a NN-based mechanism that takes an input a sequence and returns a context via a vector representation
- A RNN architecture is perfect to simulate such environments



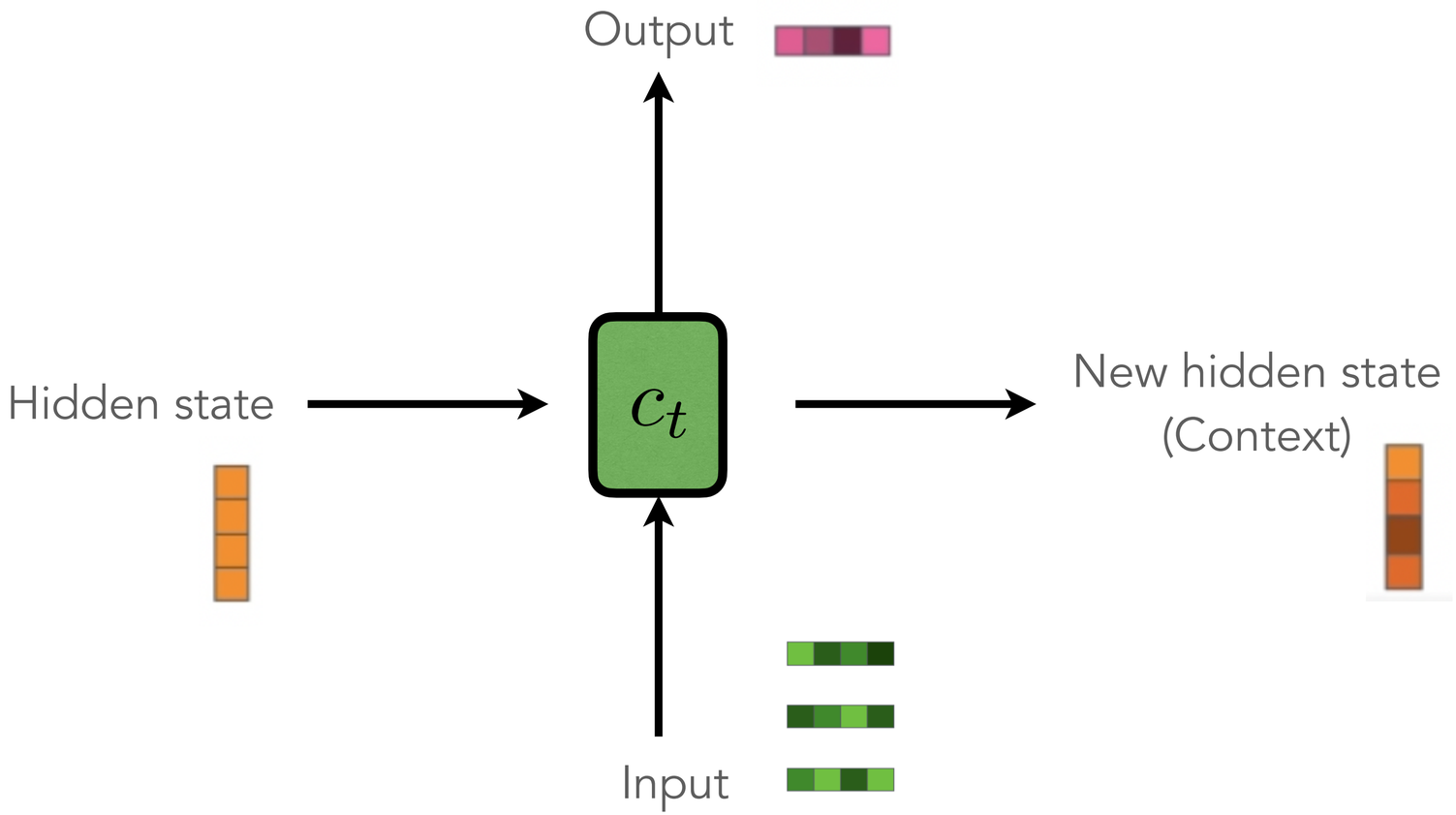
An encoder for seq2seq applications

- Usually, a NN-based mechanism that takes an input a sequence and returns a context via a vector representation
- A RNN architecture is perfect to simulate such environments

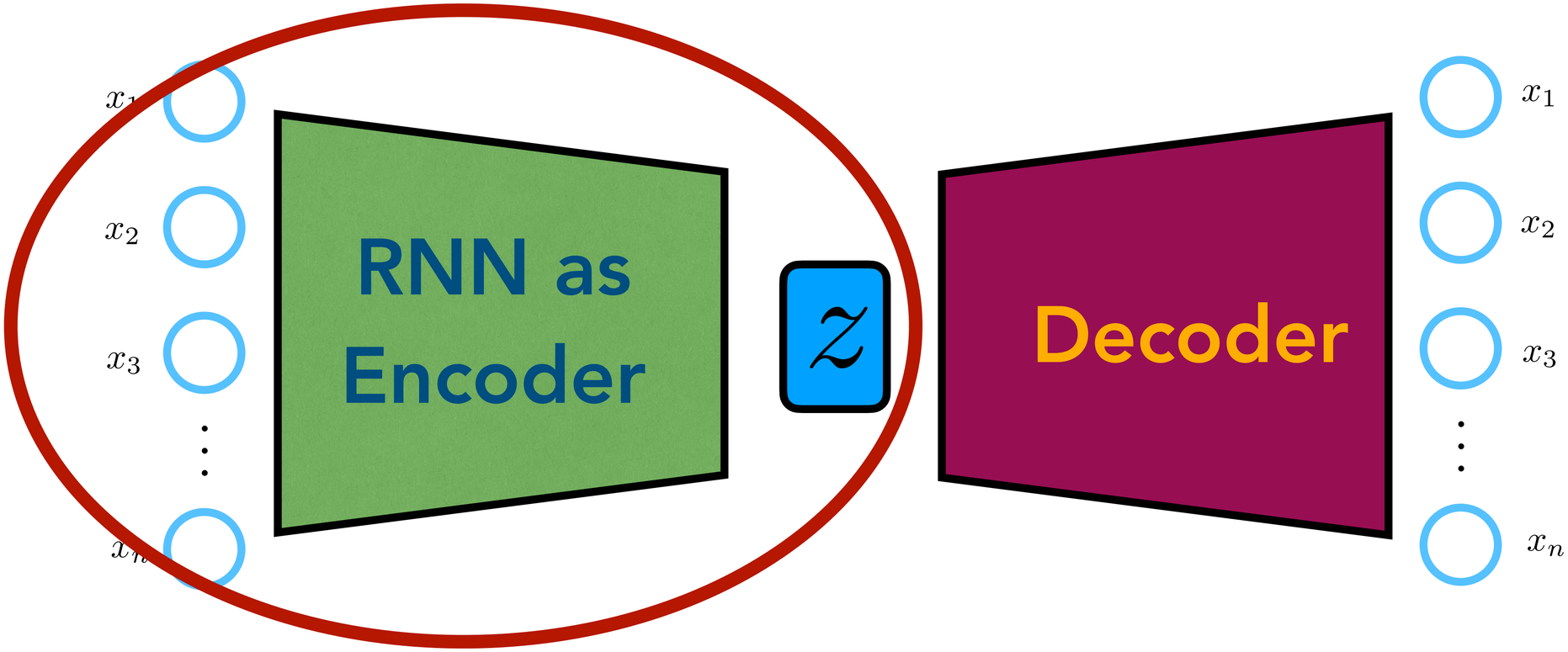


An encoder for seq2seq applications

- Usually, a NN-based mechanism that takes an input a sequence and returns a context via a vector representation
- A RNN architecture is perfect to simulate such environments

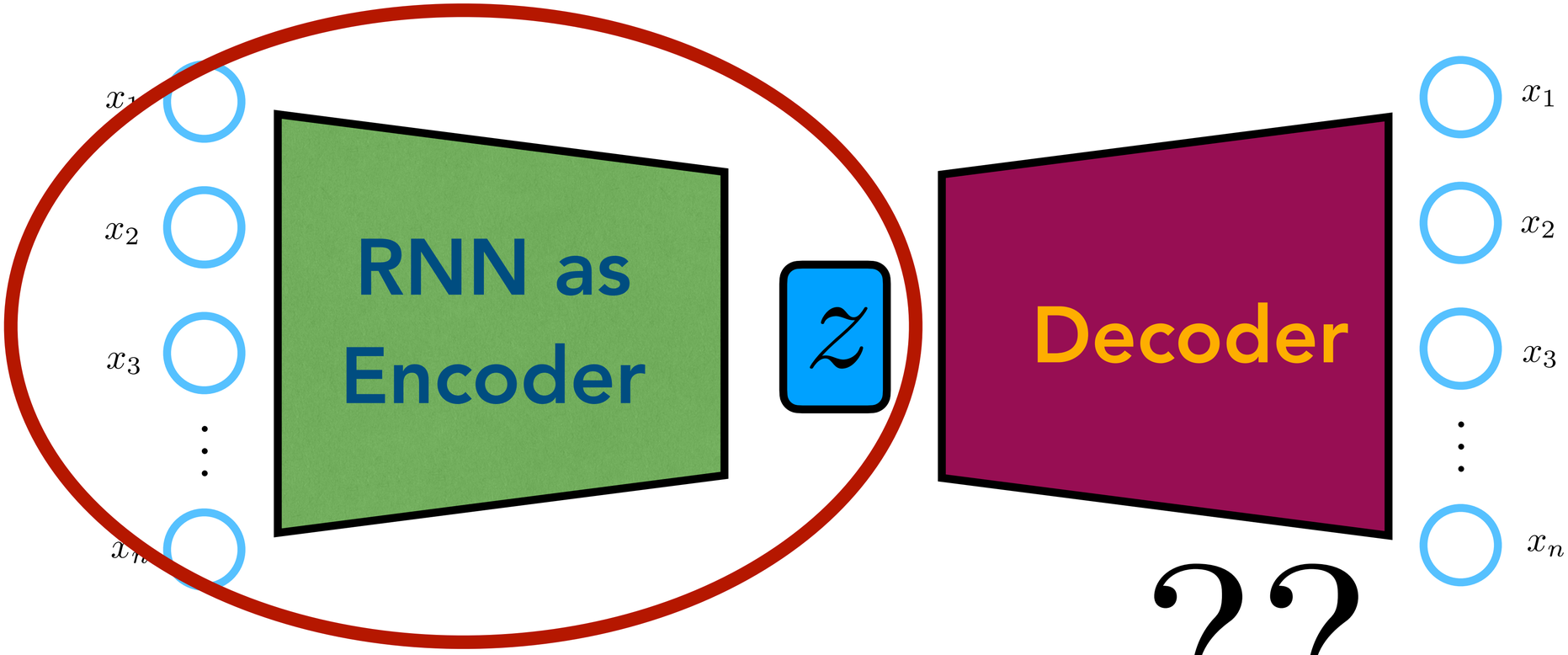


Encoder-decoder in sequence modeling



- This part is covered

Encoder-decoder in sequence modeling



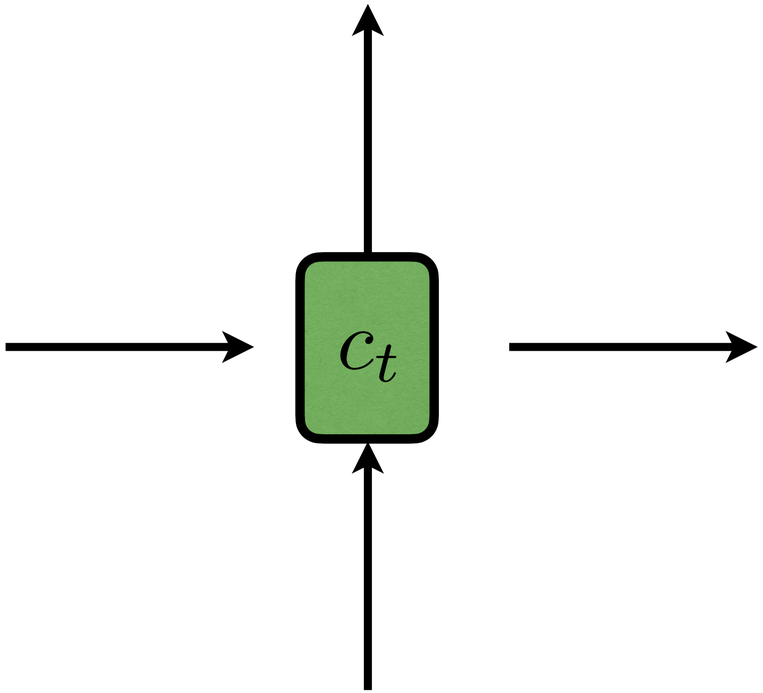
- This part is covered

The decoder

- Usually, a NN-based mechanism that takes an input a context encoded and returns an output based on the task trained

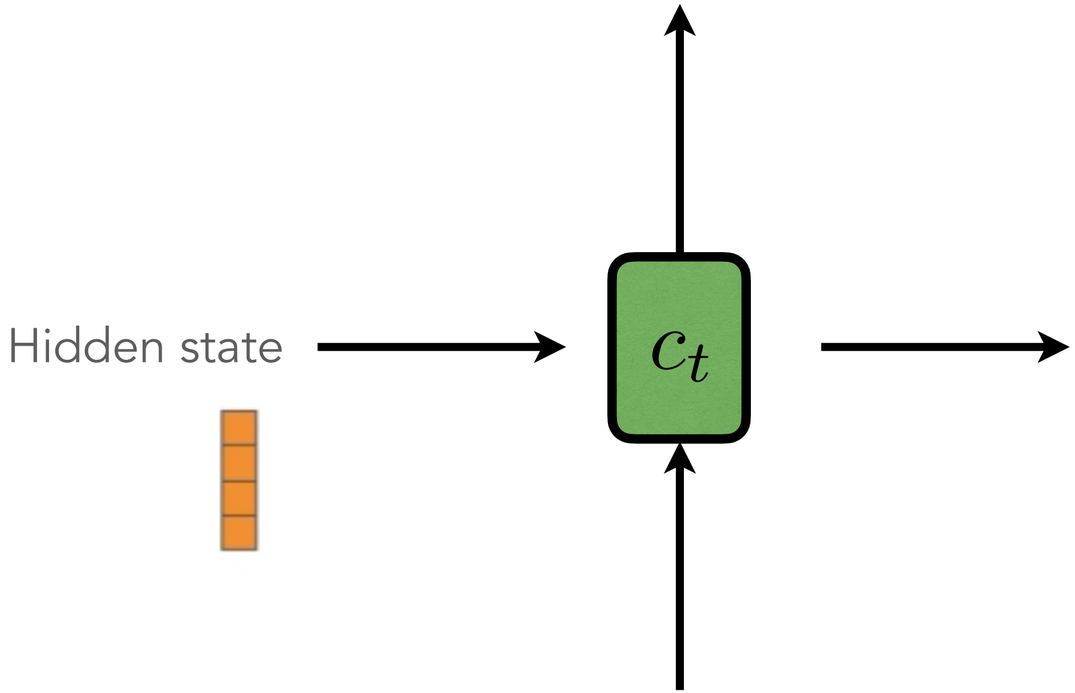
The decoder

- Usually, a NN-based mechanism that takes an input a context encoded and returns an output based on the task trained
- A RNN architecture still applies to simulate the decoding step:



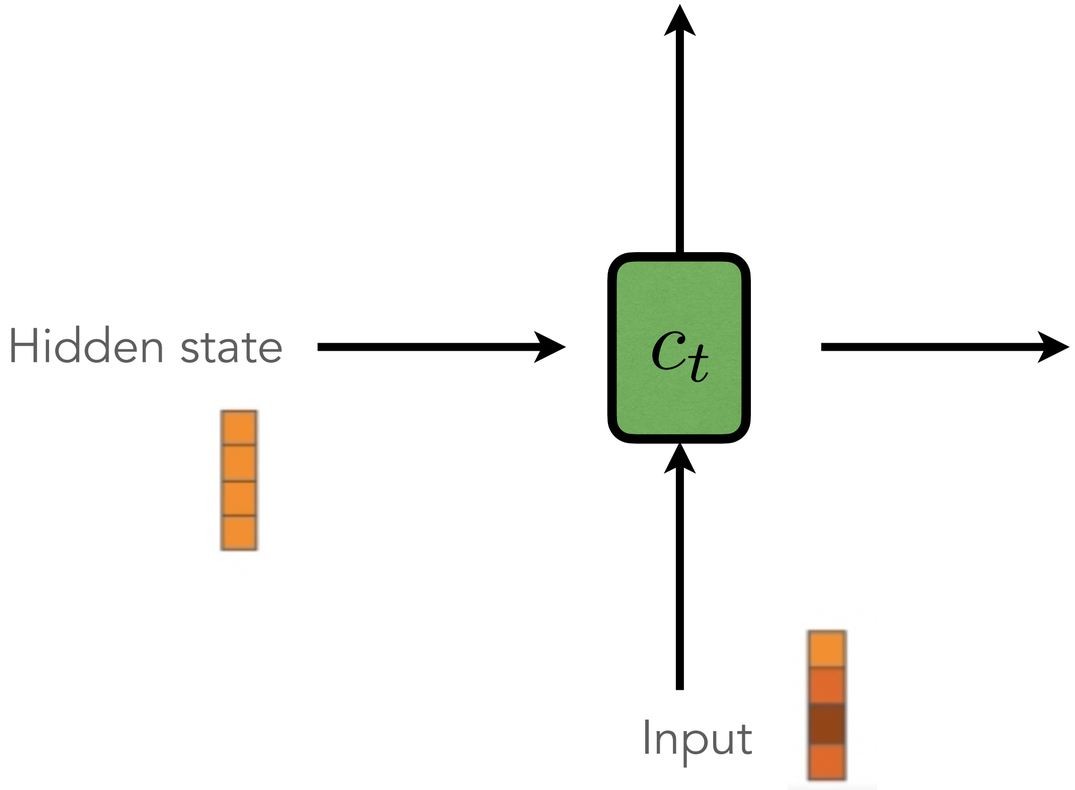
The decoder

- Usually, a NN-based mechanism that takes an input a context encoded and returns an output based on the task trained
- A RNN architecture still applies to simulate the decoding step:



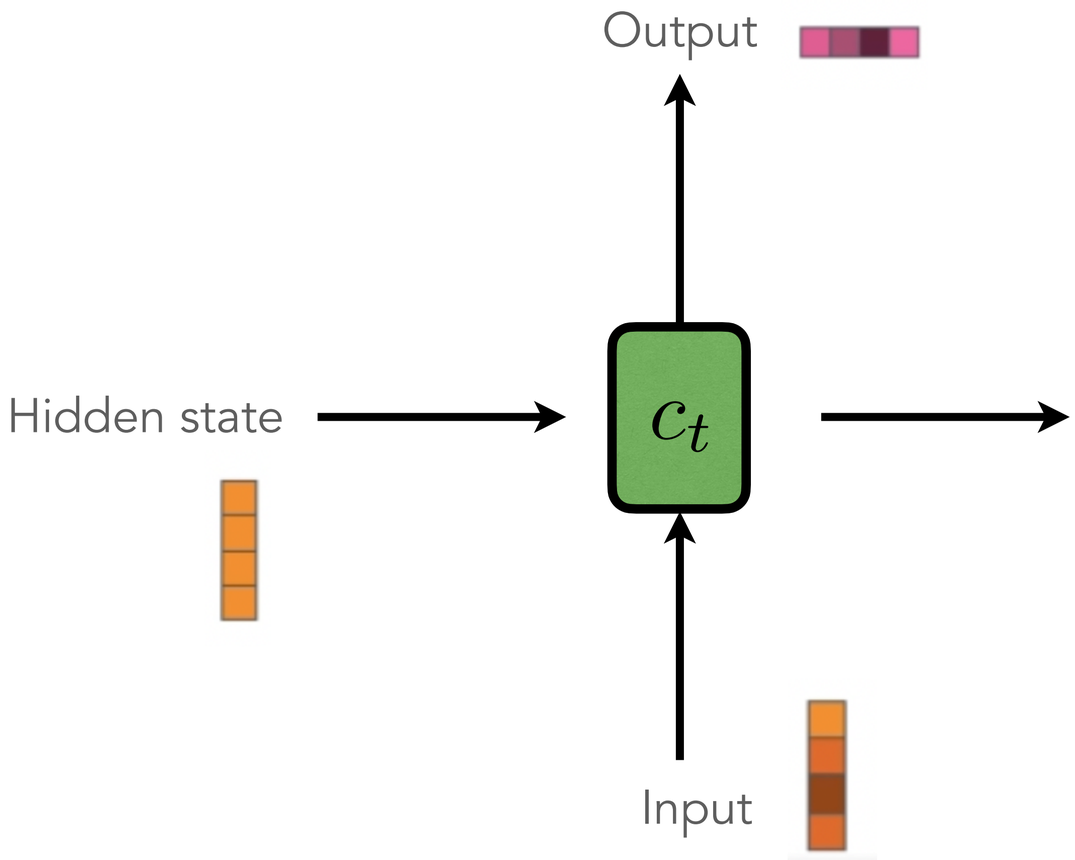
The decoder

- Usually, a NN-based mechanism that takes an input a context encoded and returns an output based on the task trained
- A RNN architecture still applies to simulate the decoding step:



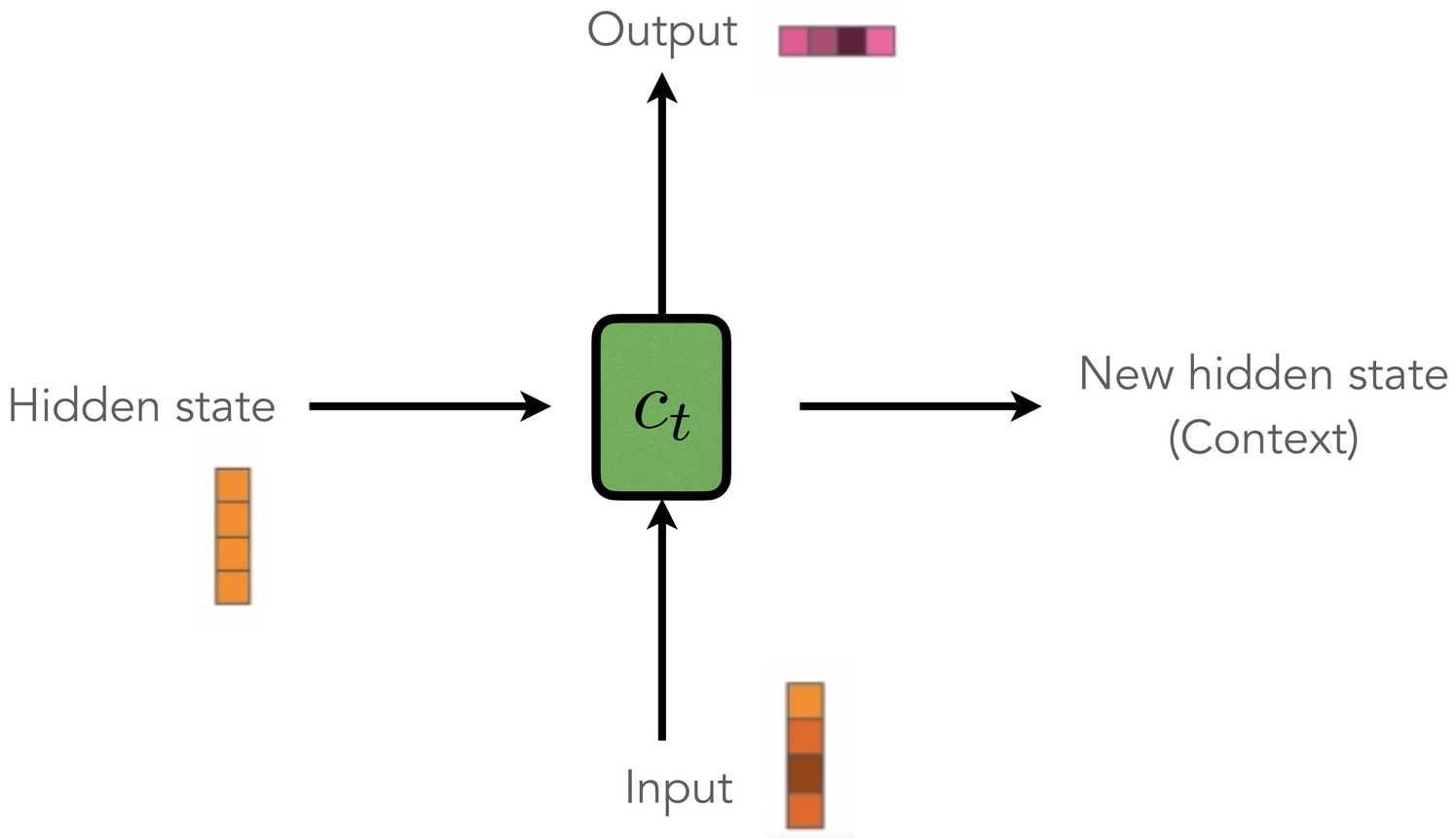
The decoder

- Usually, a NN-based mechanism that takes an input a context encoded and returns an output based on the task trained
- A RNN architecture still applies to simulate the decoding step:



The decoder

- Usually, a NN-based mechanism that takes an input a context encoded and returns an output based on the task trained
- A RNN architecture still applies to simulate the decoding step:



Where does attention come into the picture?

- Based on a long input sequence, classical approaches usually “exploit” only the last context vector - not all of them.

Where does attention come into the picture?

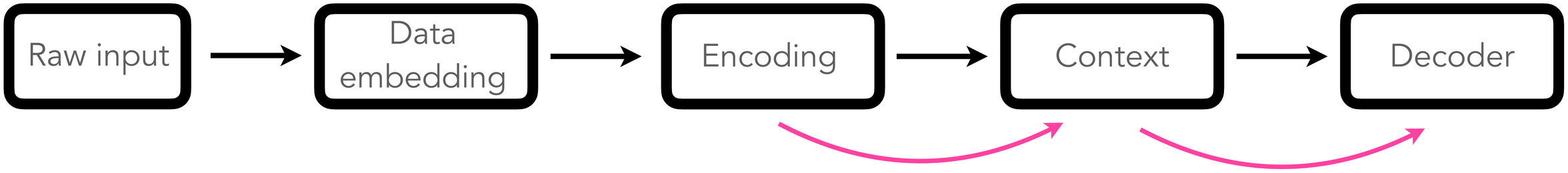
- Based on a long input sequence, classical approaches usually “exploit” only the last context vector - not all of them.
- Attention mechanisms allow the model to focus on different parts of the input input sequence (e.g., potentially, all context vectors from all steps are useful)

Where does attention come into the picture?

- Based on a long input sequence, classical approaches usually "exploit" only the last context vector - not all of them.
- Attention mechanisms allow the model to focus on different parts of the input input sequence (e.g., potentially, all context vectors from all steps are useful)

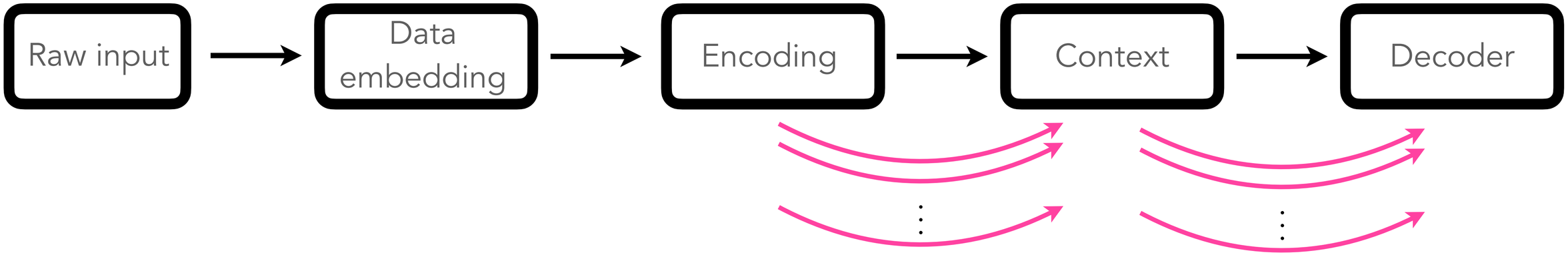
Where does attention come into the picture?

- Based on a long input sequence, classical approaches usually "exploit" only the last context vector - not all of them.
- Attention mechanisms allow the model to focus on different parts of the input input sequence (e.g., potentially, all context vectors from all steps are useful)



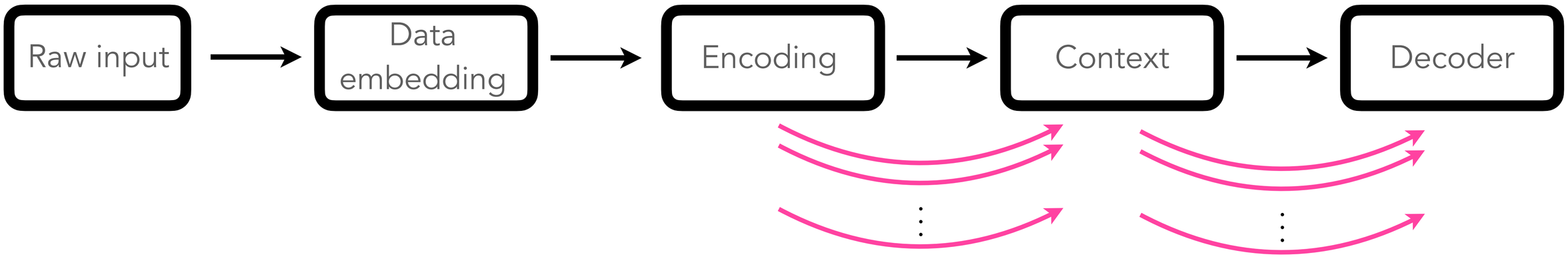
Where does attention come into the picture?

- Based on a long input sequence, classical approaches usually "exploit" only the last context vector - not all of them.
- Attention mechanisms allow the model to focus on different parts of the input input sequence (e.g., potentially, all context vectors from all steps are useful)

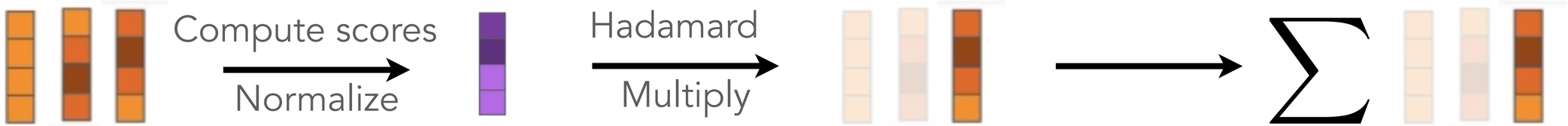


Where does attention come into the picture?

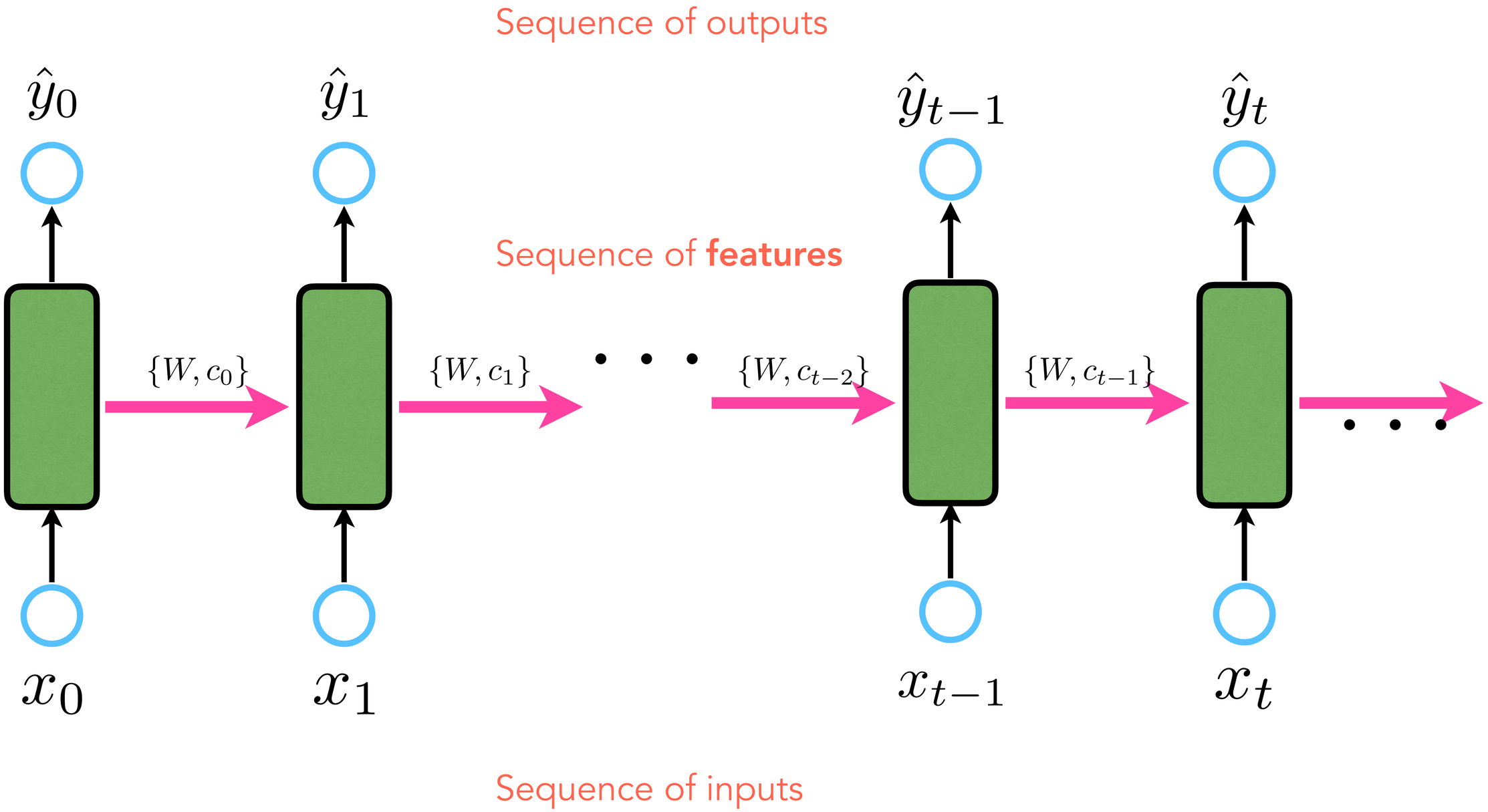
- Based on a long input sequence, classical approaches usually "exploit" only the last context vector - not all of them.
- Attention mechanisms allow the model to focus on different parts of the input input sequence (e.g., potentially, all context vectors from all steps are useful)



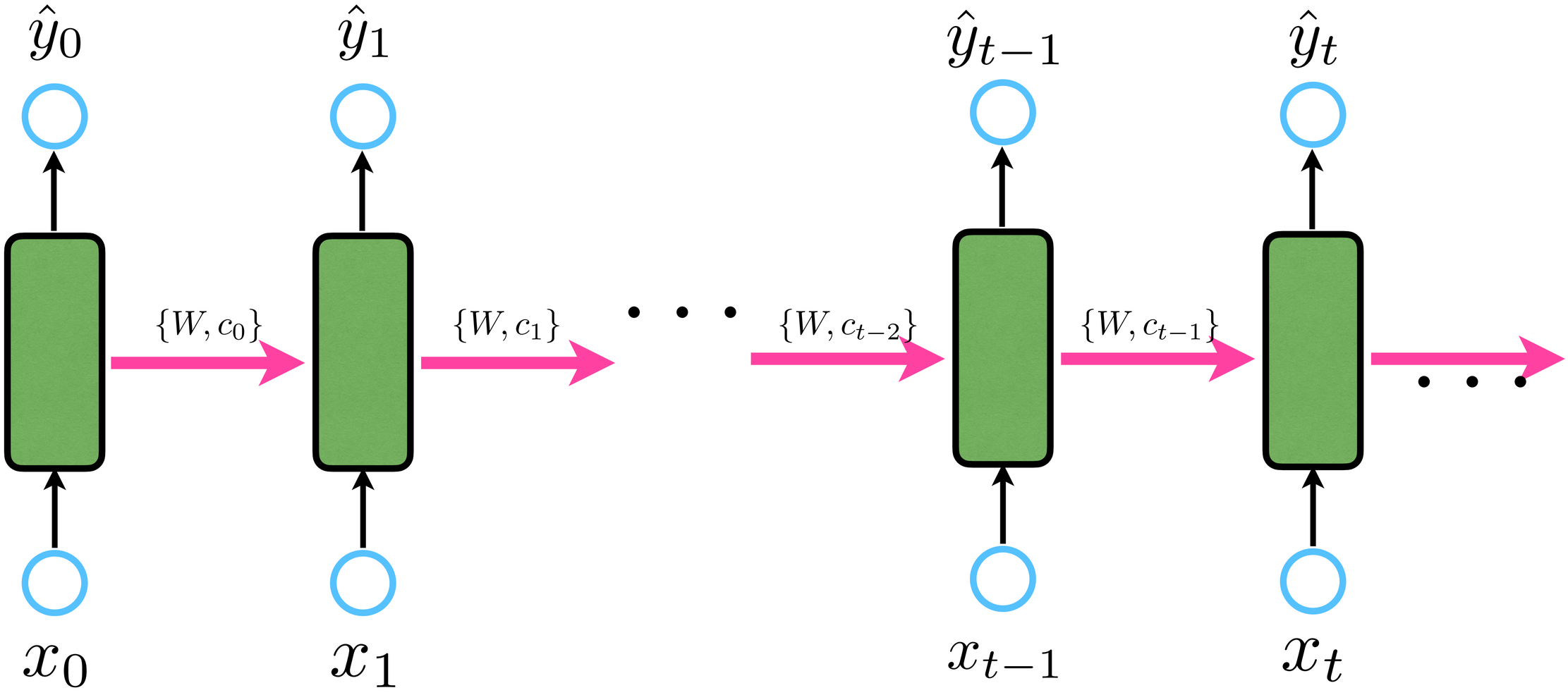
- How would a decoder use that many context vectors?



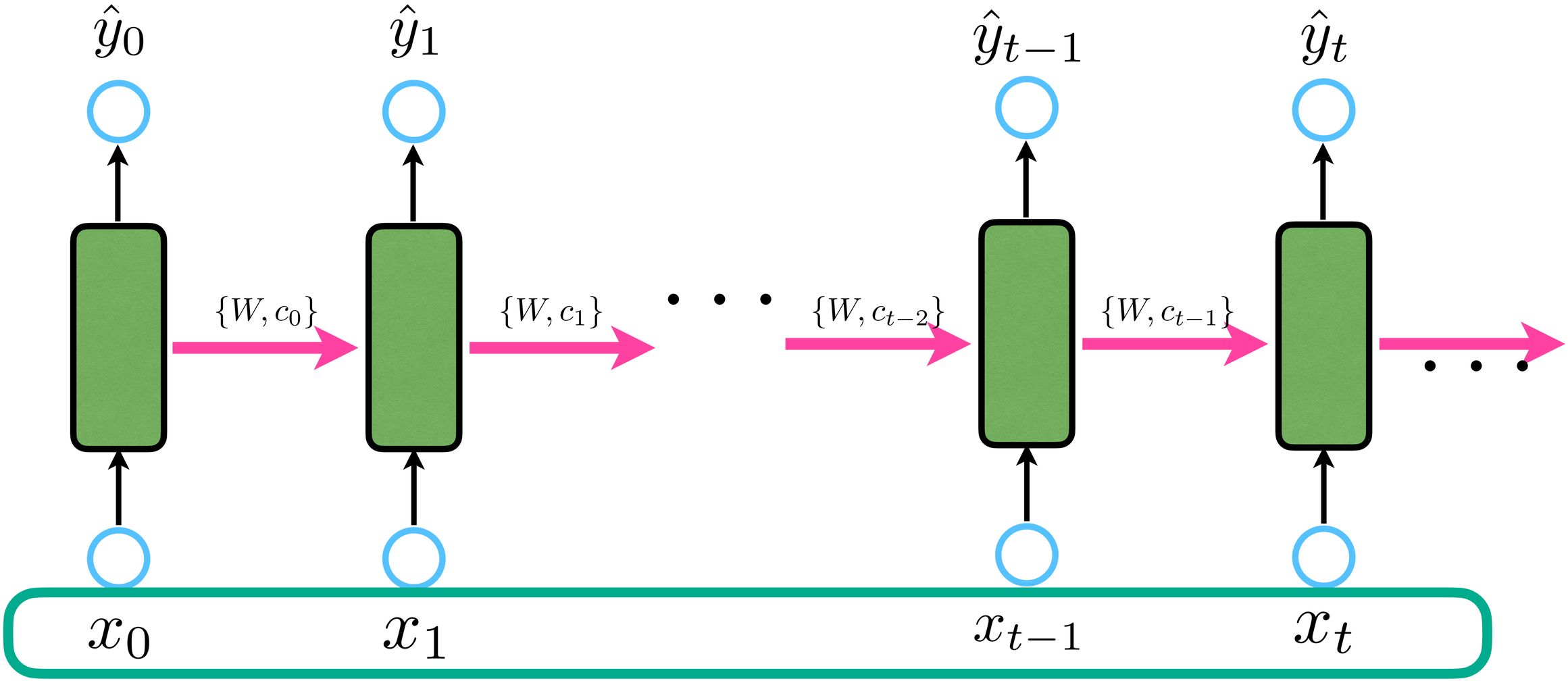
How to overcome the issue of ``last context vector''?



How to overcome the issue of ``last context vector''?

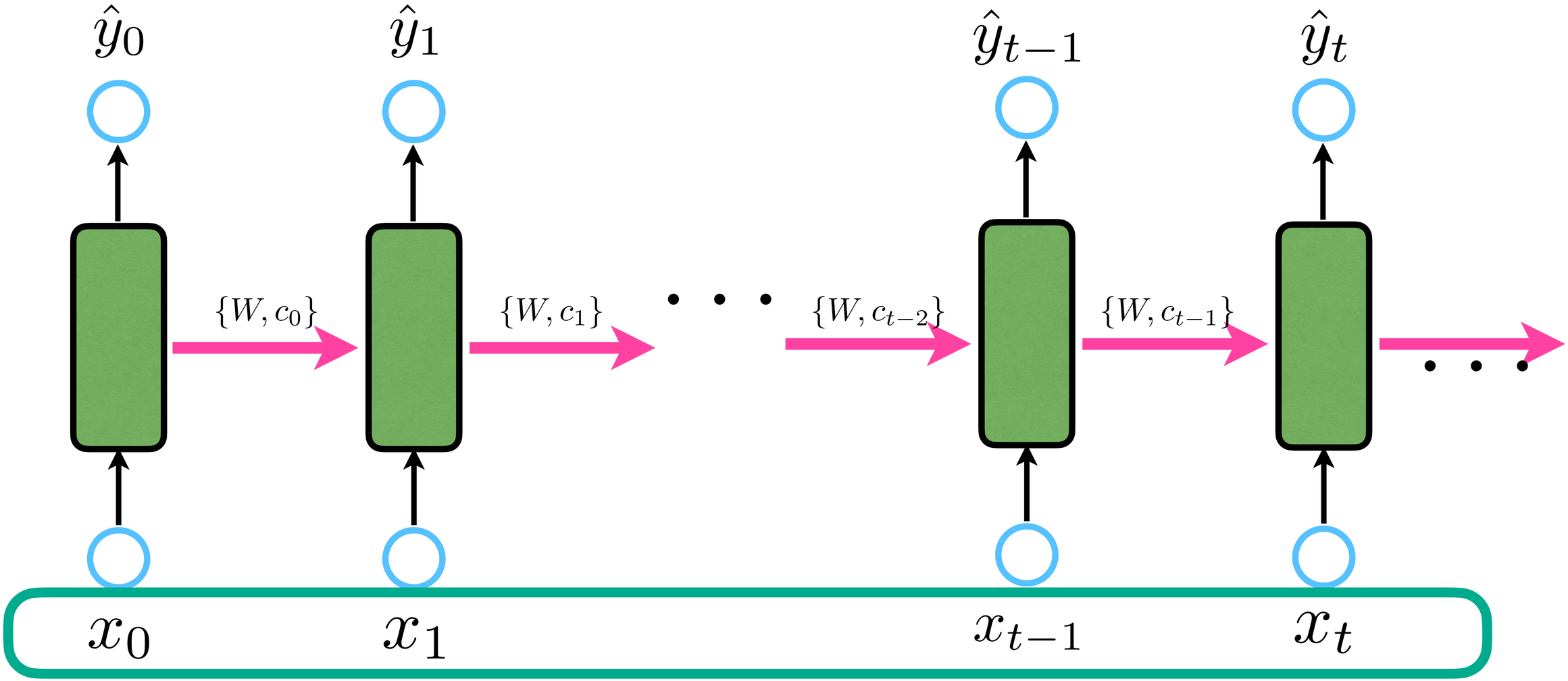


How to overcome the issue of ``last context vector''?

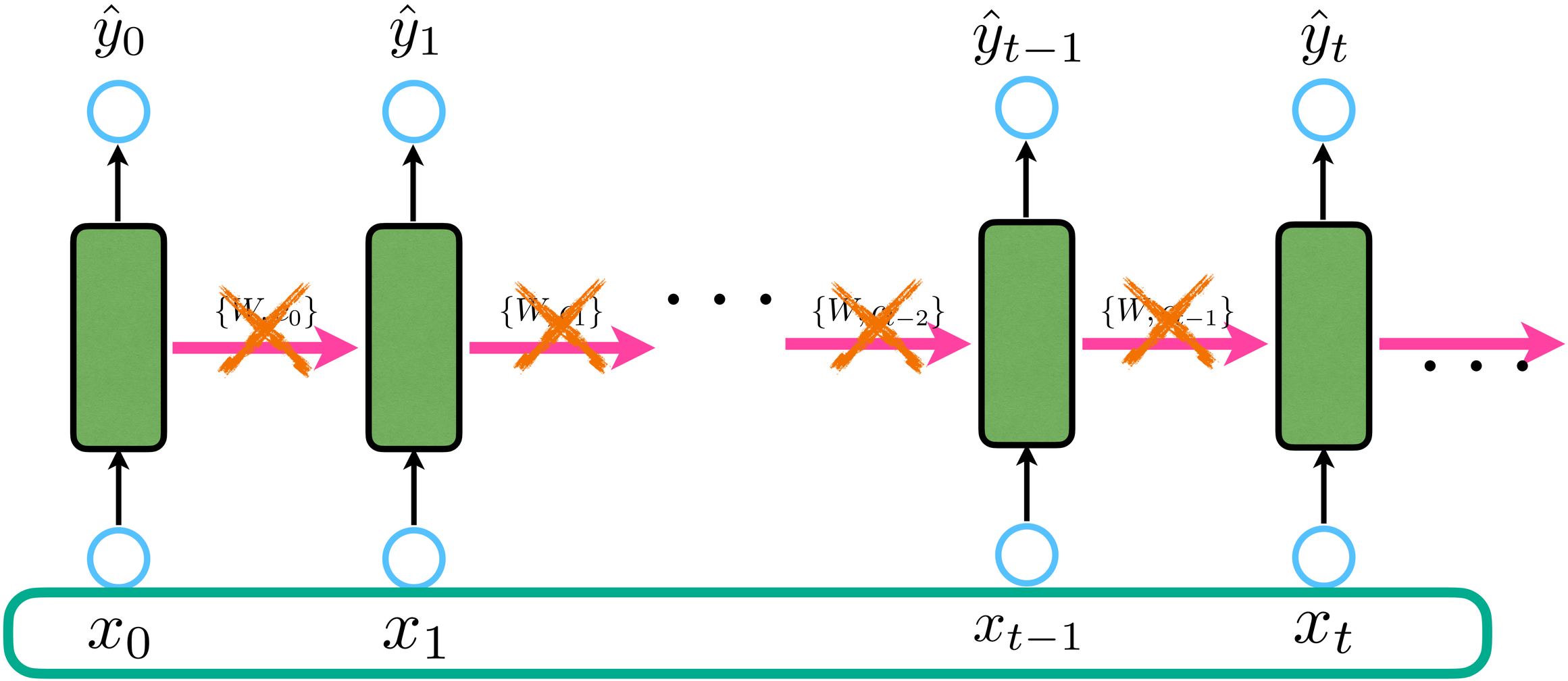


Fix a window (spanning) size

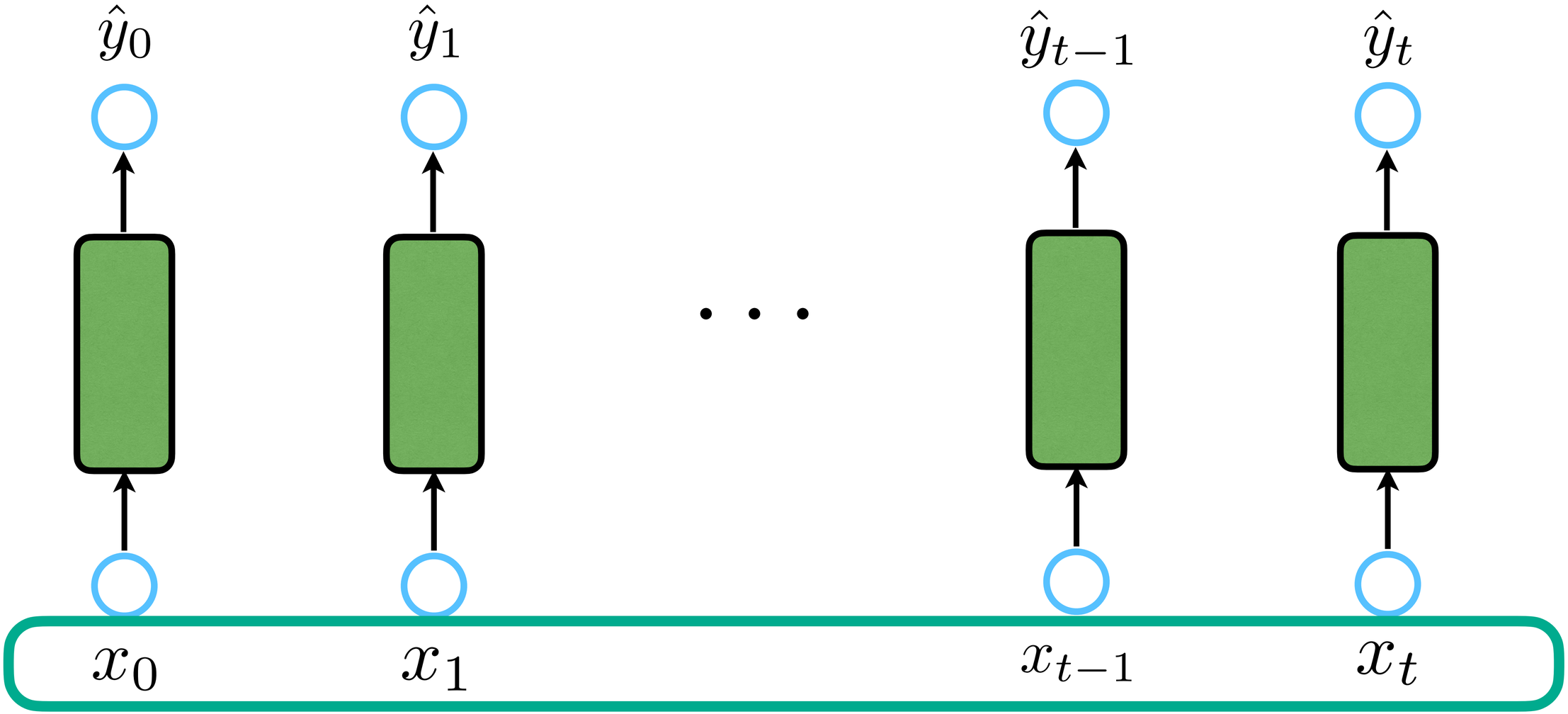
How to overcome the issue of ``last context vector''?



How to overcome the issue of ``last context vector''?

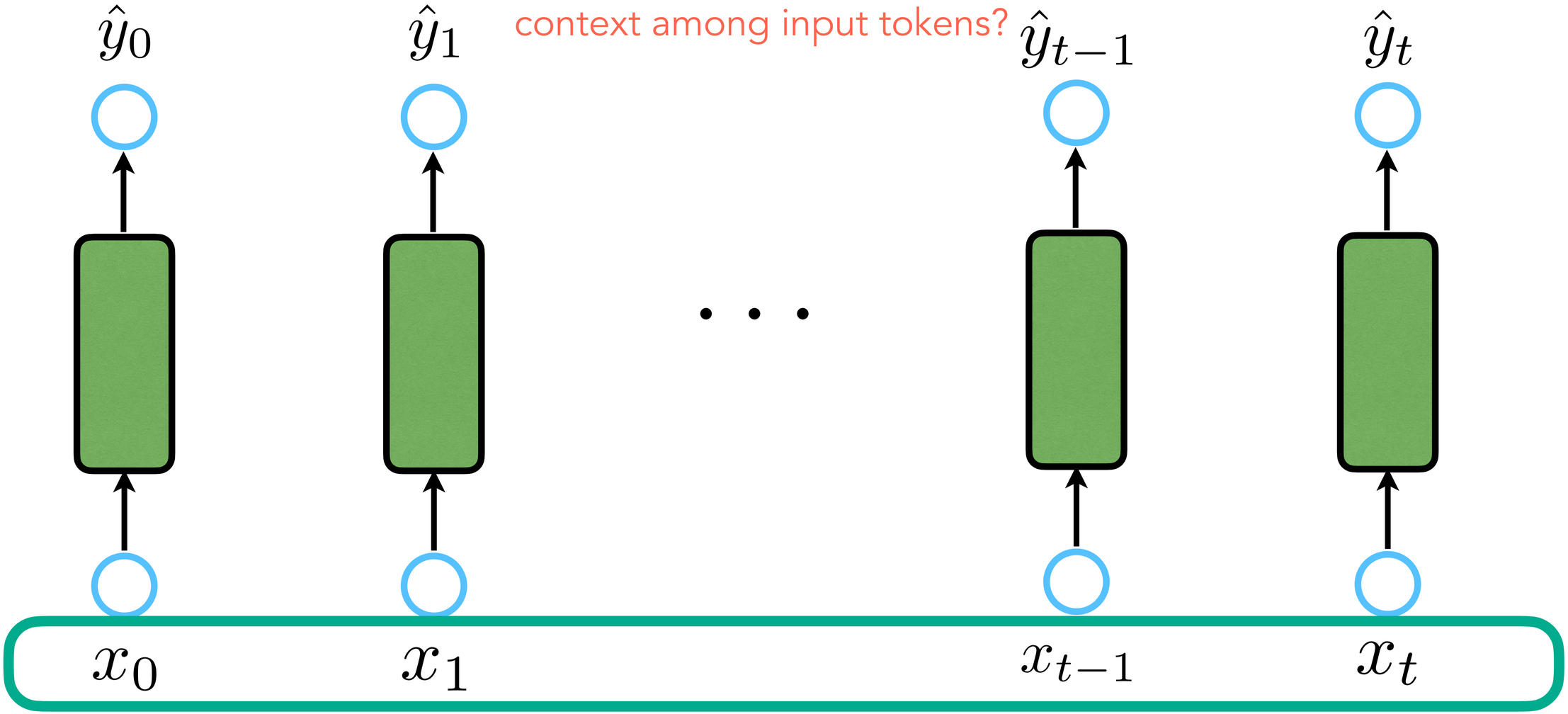


How to overcome the issue of ``last context vector''?



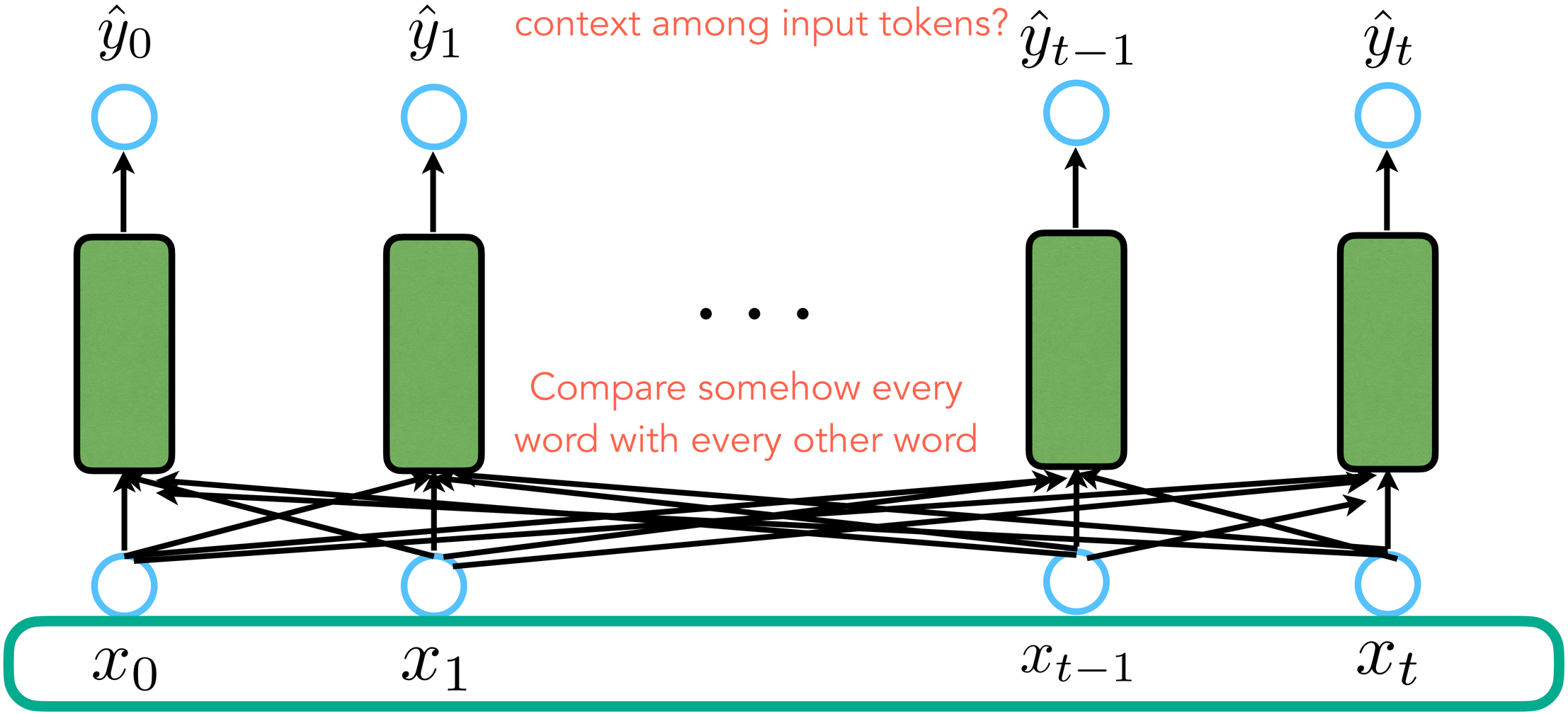
How to overcome the issue of ``last context vector''?

But how are we going to keep context among input tokens?



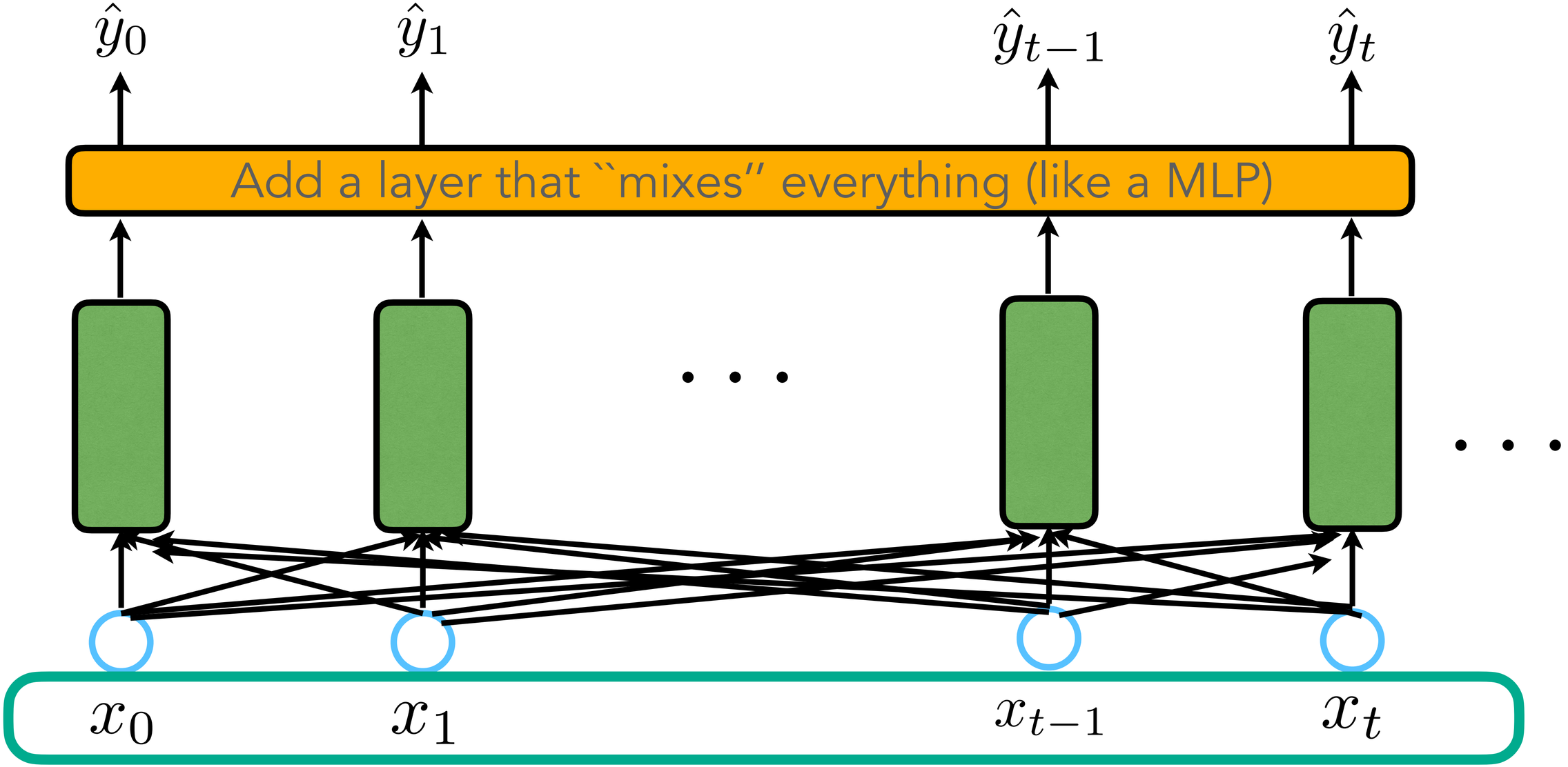
How to overcome the issue of ``last context vector''?

But how are we going to keep context among input tokens?



Compare somehow every word with every other word

How to overcome the issue of ``last context vector''?



That was attention; what is self-attention?

That was attention; what is self-attention?

That was attention; what is self-attention?

- Self-attention breaks the sequential nature of RNNs:
  i.e., the input is a long streak of words, and we compute correlations

That was attention; what is self-attention?

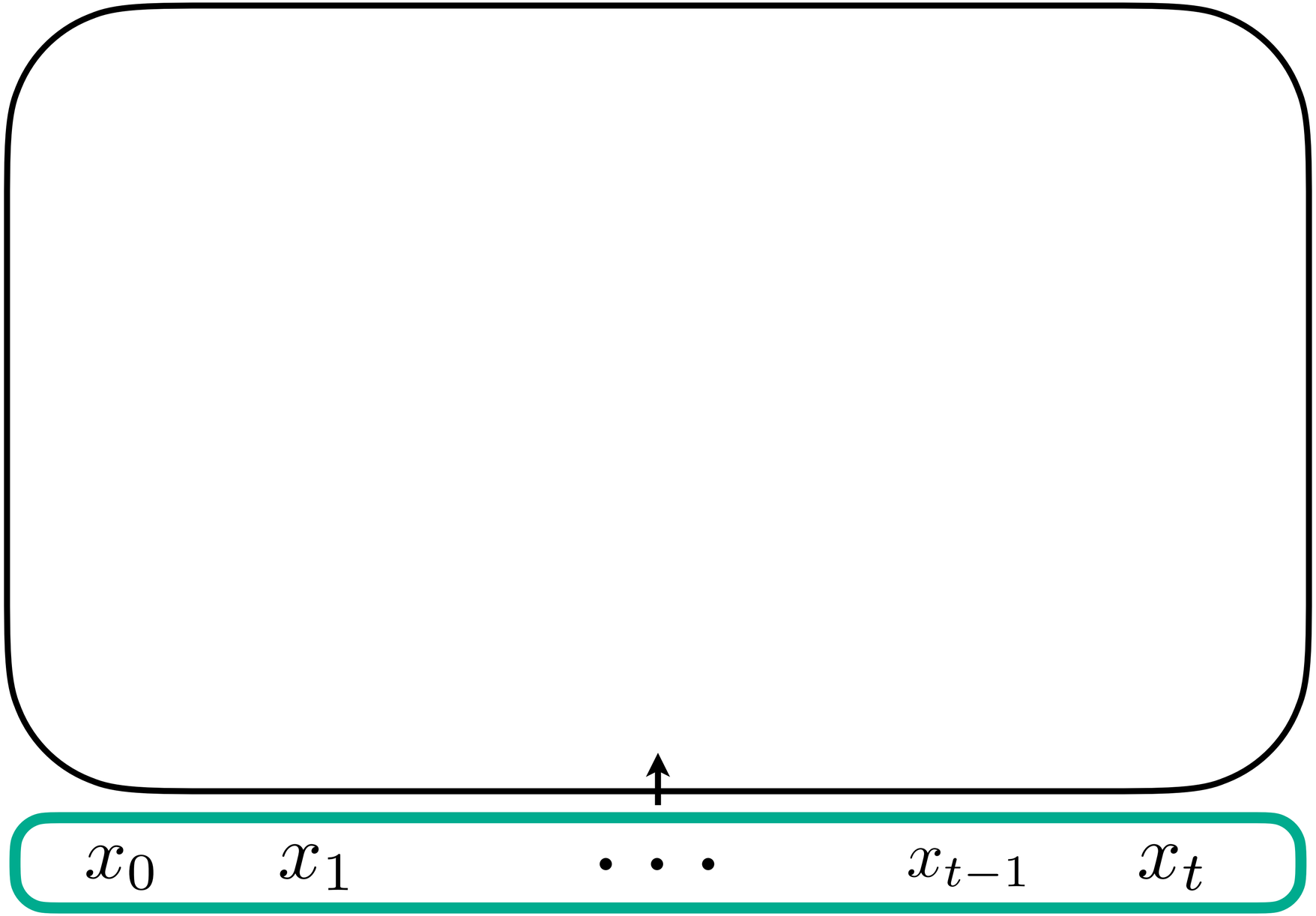
- Self-attention breaks the sequential nature of RNNs:
  i.e., the input is a long streak of words, and we compute correlations
- Self-attention simultaneously compare each part of the input
  with other parts of the input

Transformer module

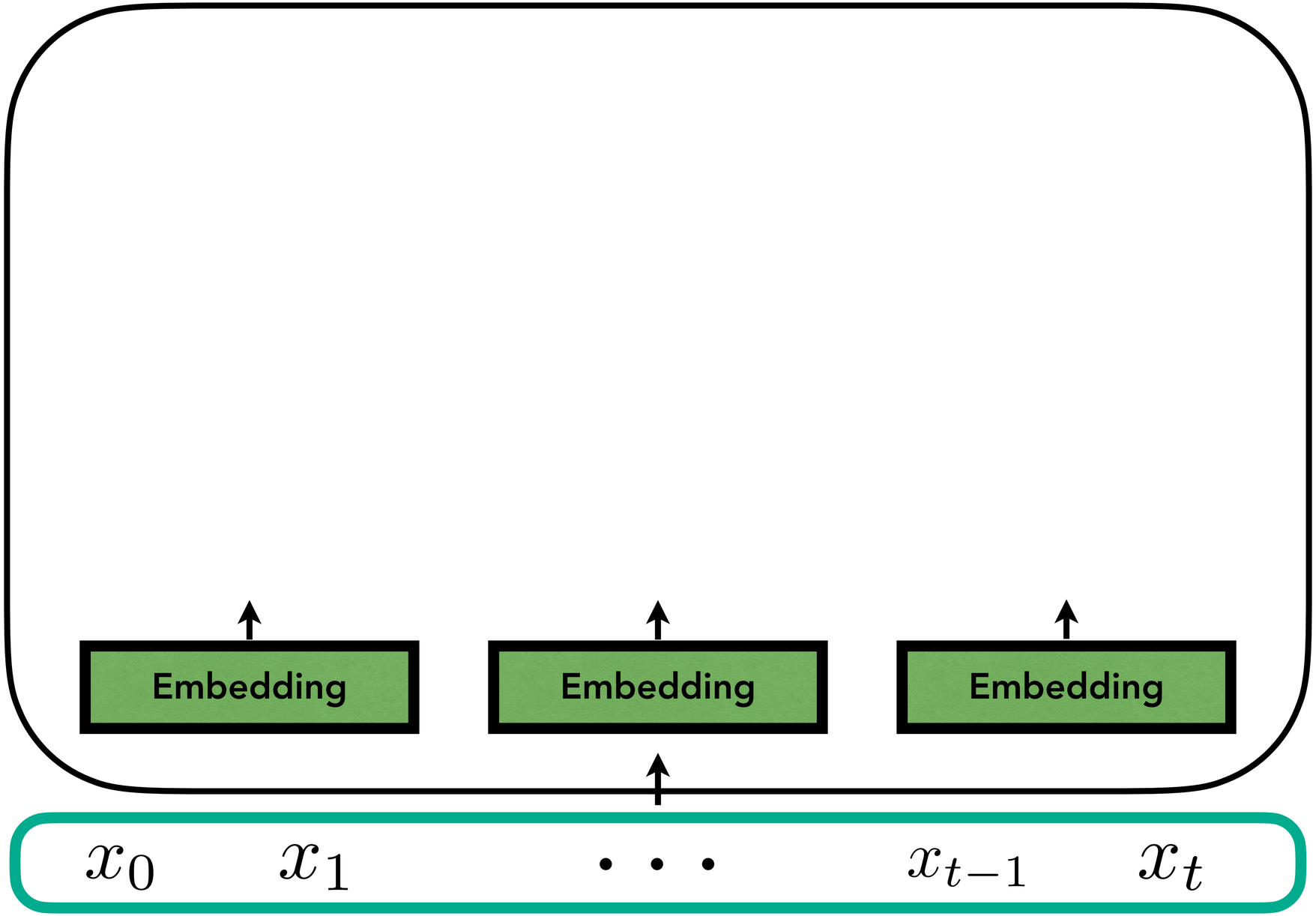
Transformer module

$$x_0 \quad x_1 \quad \cdots \quad x_{t-1} \quad x_t$$

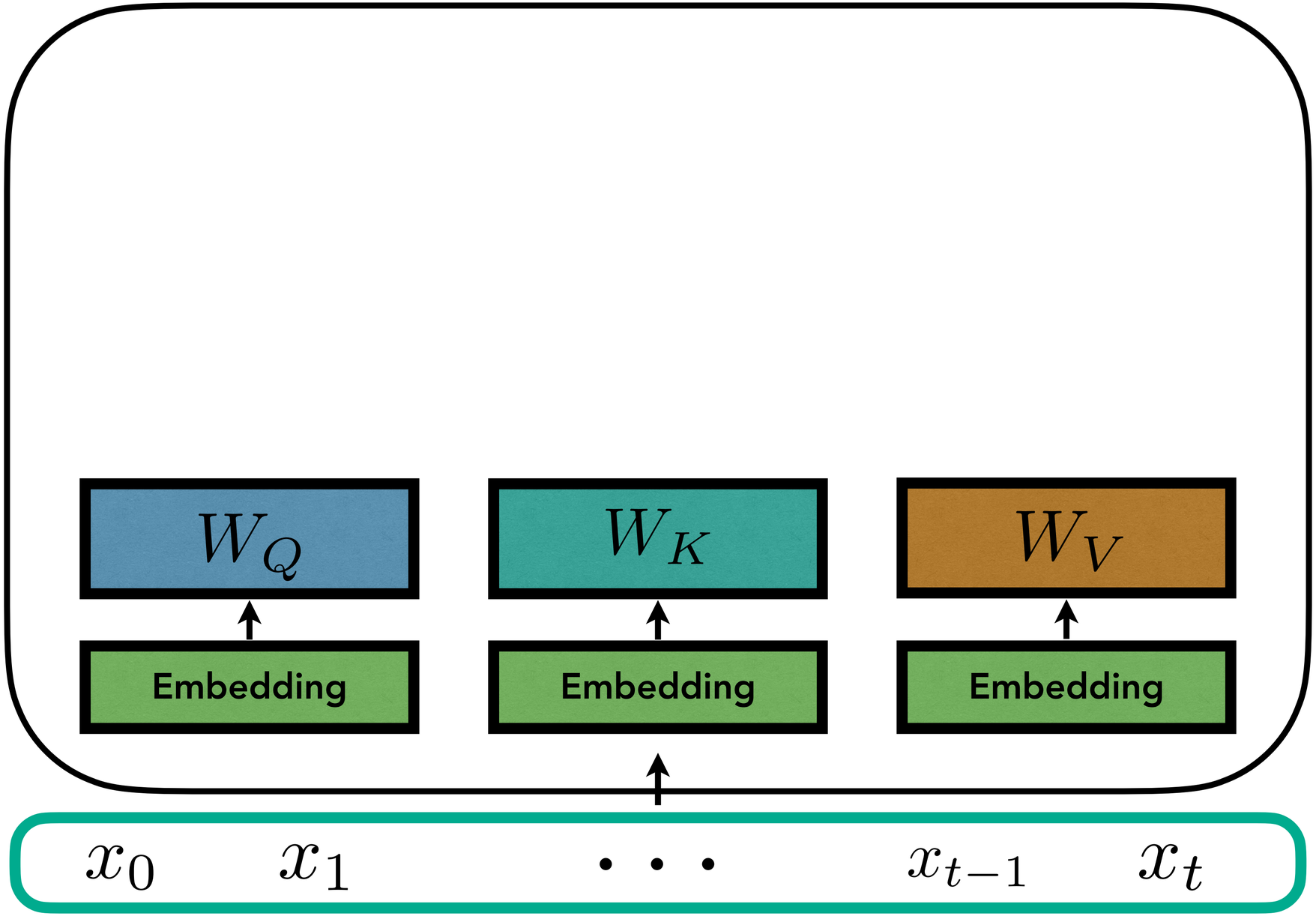
Transformer module



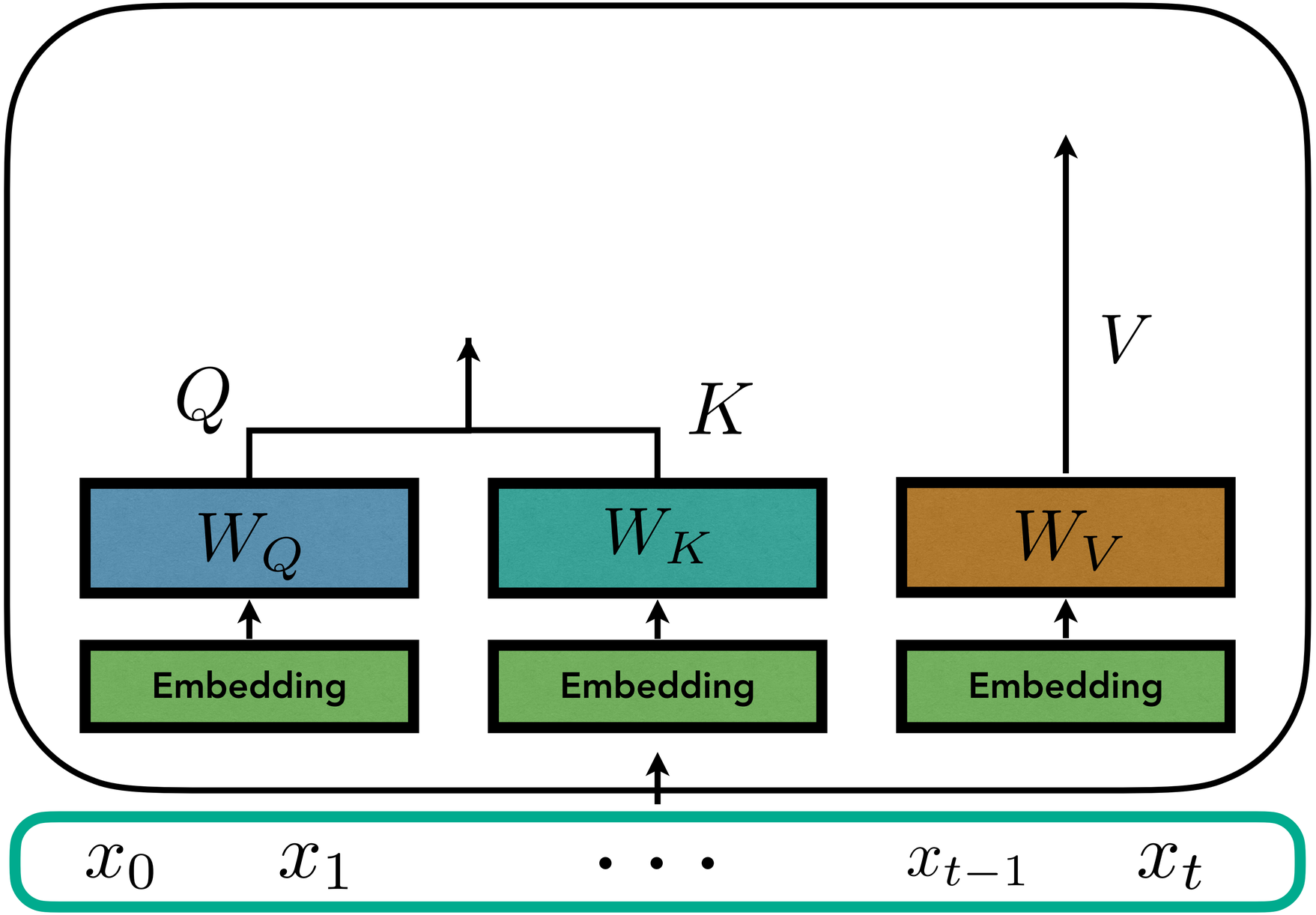
Transformer module



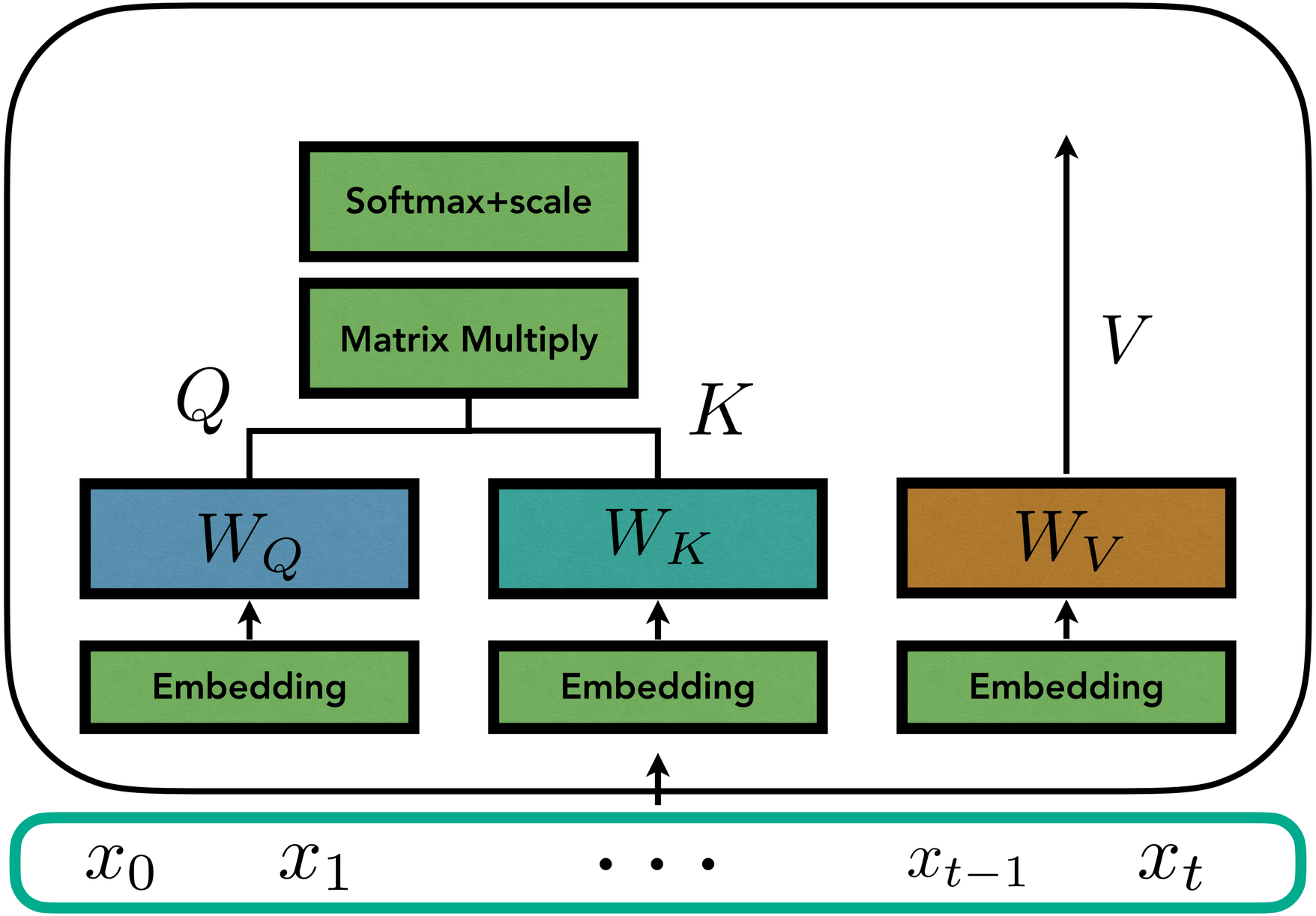
Transformer module



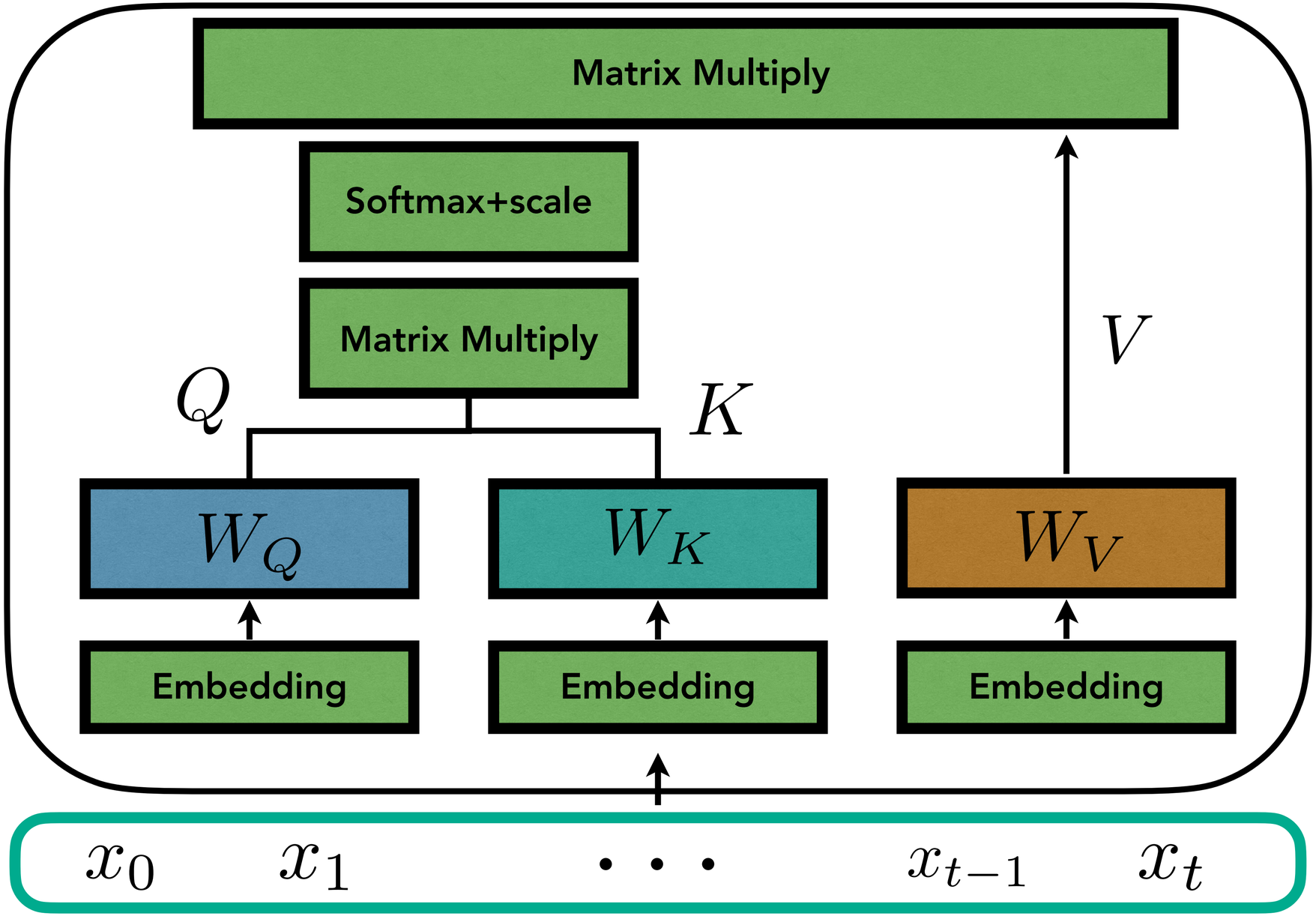
Transformer module



Transformer module

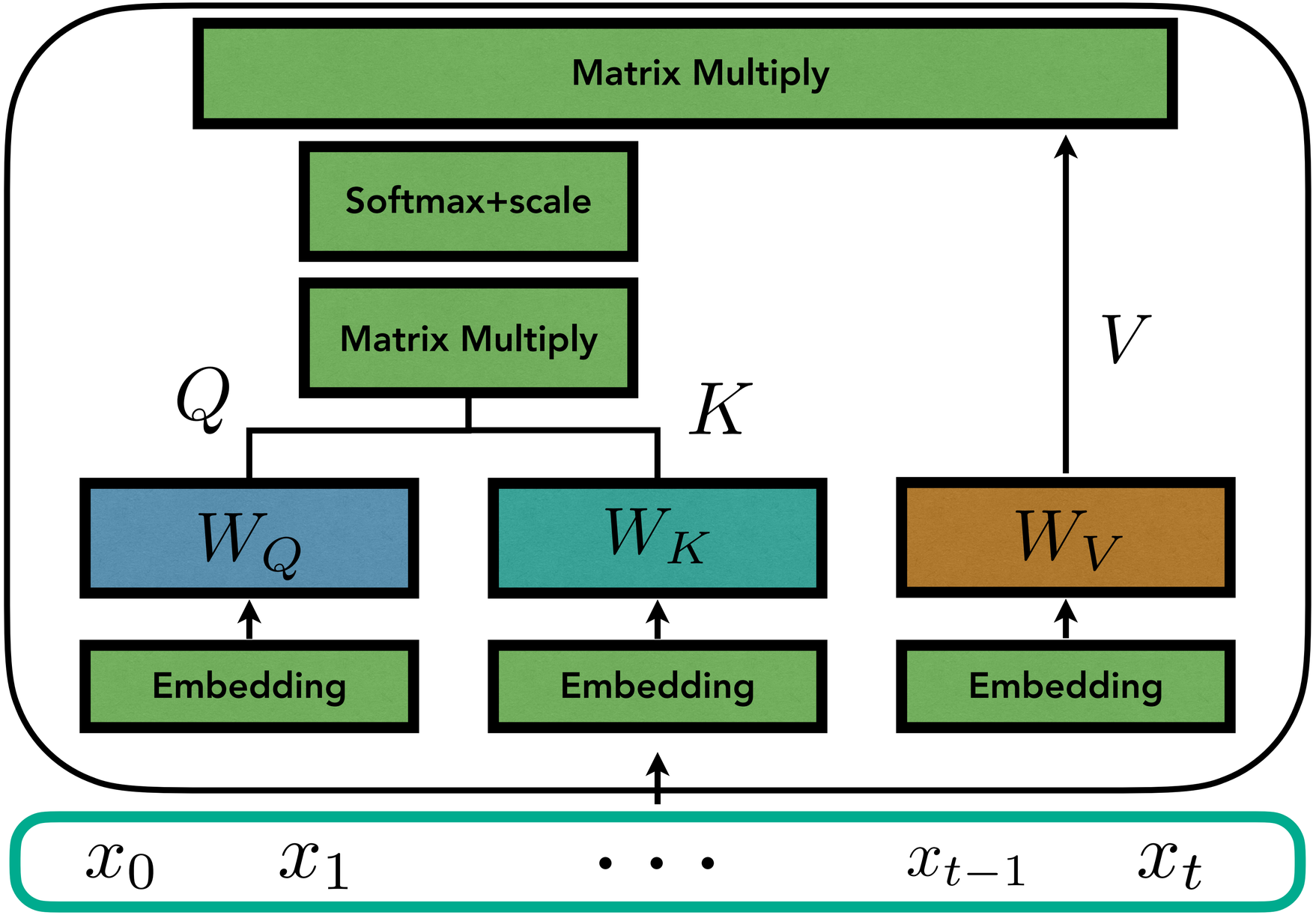


Transformer module



Transformer module

$$\texttt{Att}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^\top}{\text{scale}}\right) \cdot V$$



Transformers = multiple self-attention + FC layers

- Work by Google on 2017

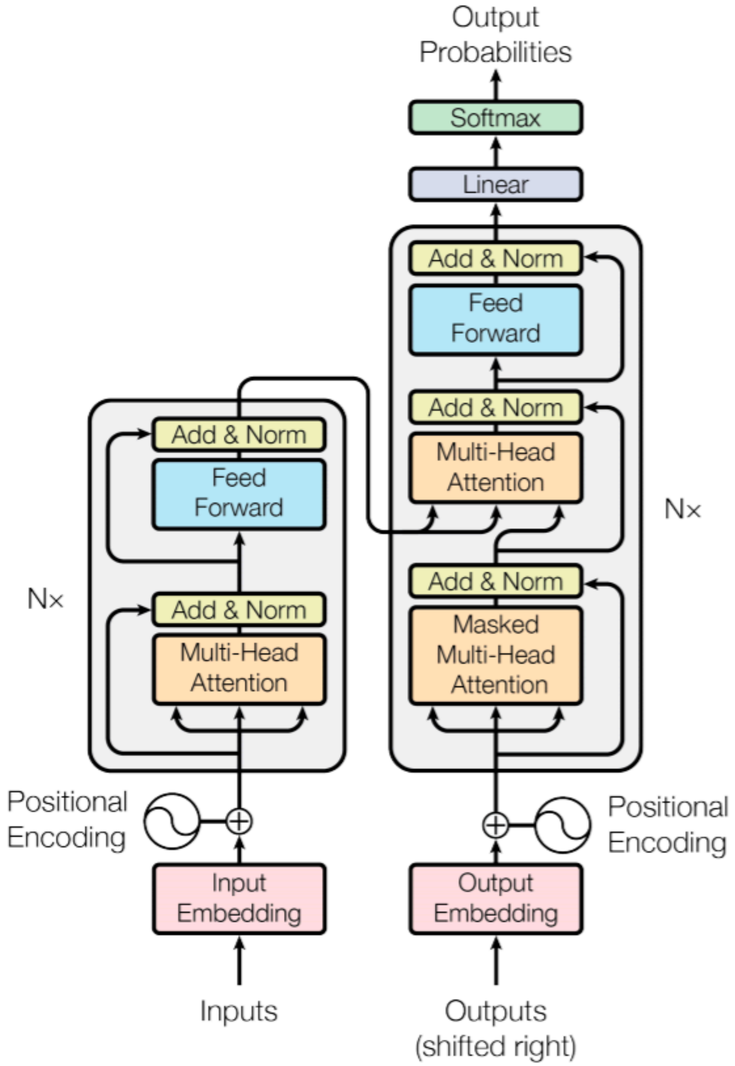


Figure 1: The Transformer - model architecture.

Transformers = multiple self-attention + FC layers

- Work by Google on 2017

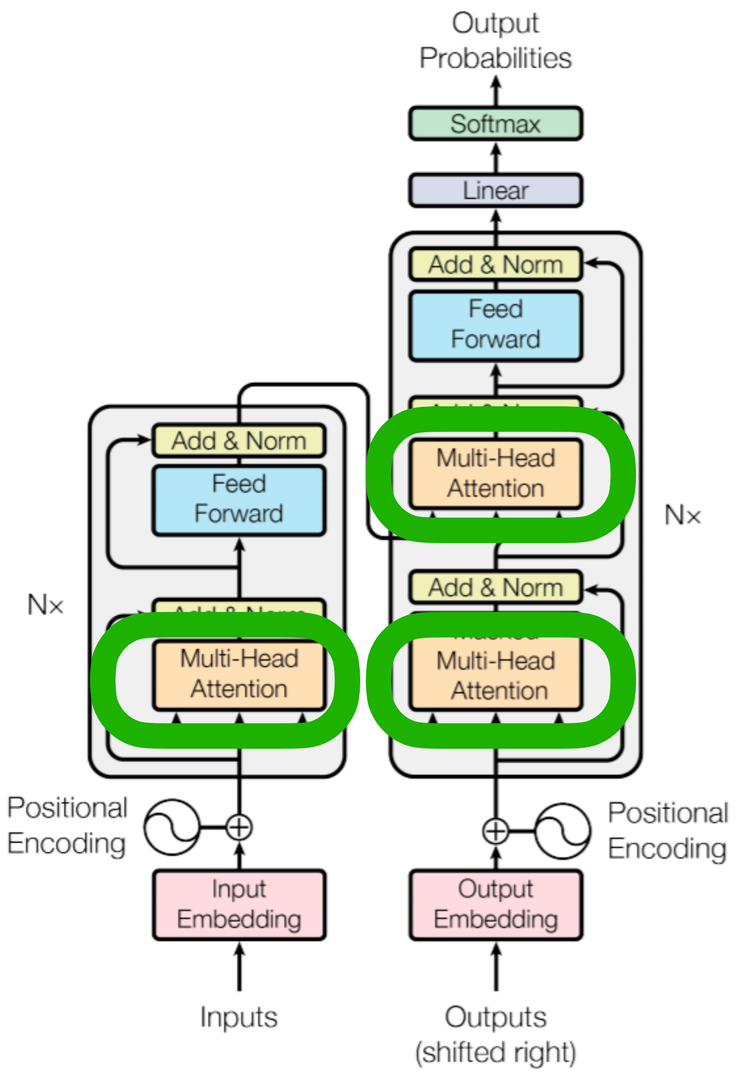


Figure 1: The Transformer - model architecture.

Transformers = multiple self-attention + FC layers

- Work by Google on 2017

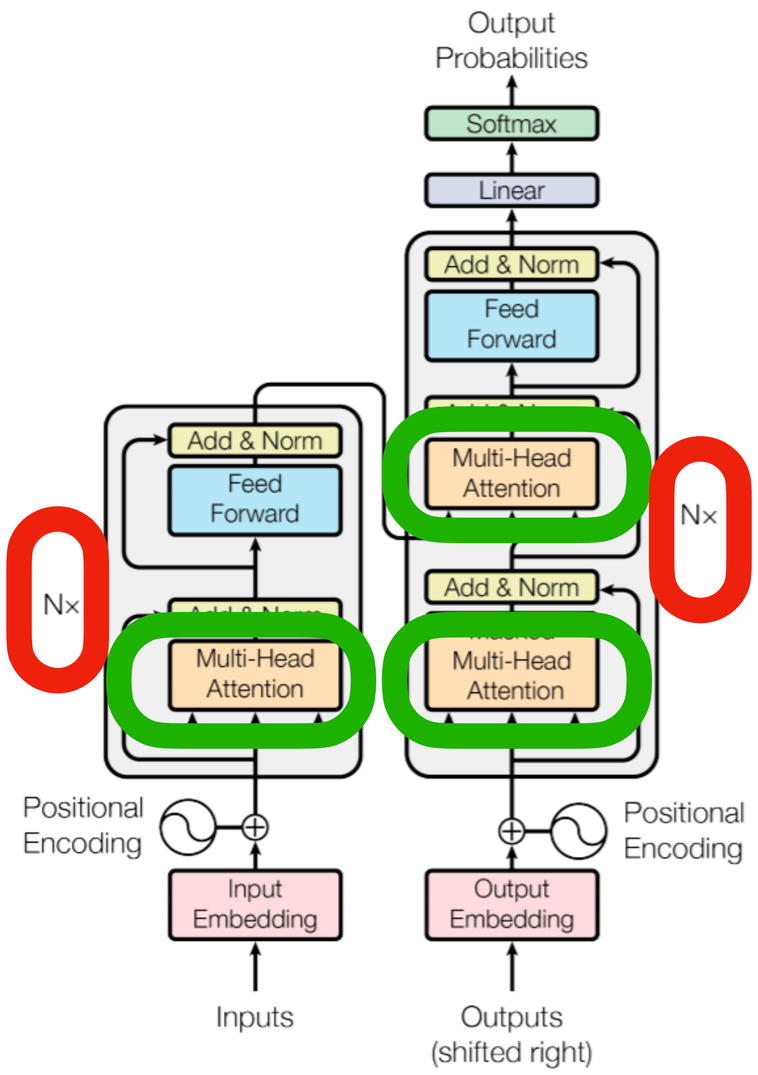


Figure 1: The Transformer - model architecture.

Transformers = multiple self-attention + FC layers

- Work by Google on 2017
- Applications: machine translation
  document summarization
  document generation
  bio sequence analysis
  video understanding
  playing games
  …

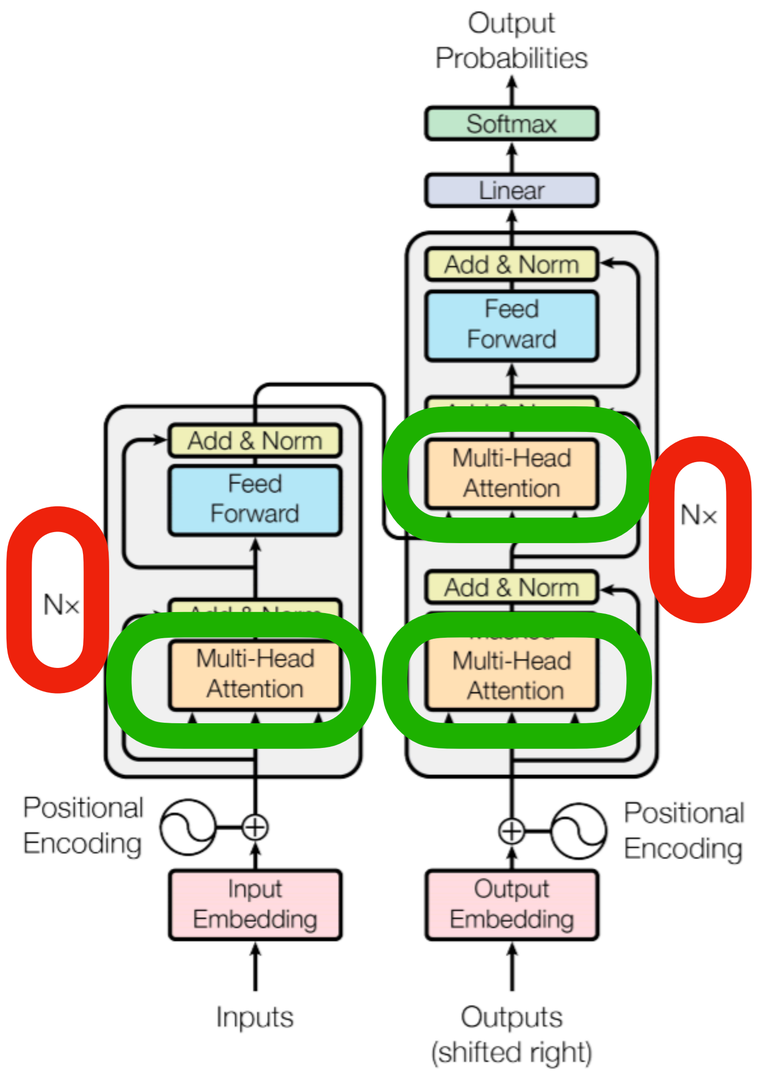


Figure 1: The Transformer - model architecture.

Transformers = multiple self-attention + FC layers

- Work by Google on 2017
- Applications: machine translation
  document summarization
  document generation
  bio sequence analysis
  video understanding
  playing games
  …
- Known large architectures: GPT-2 (1.5b)
  GPT-3 (175b)
  Wu Dao (1.75tr)
- Large models have raised many environmental questions

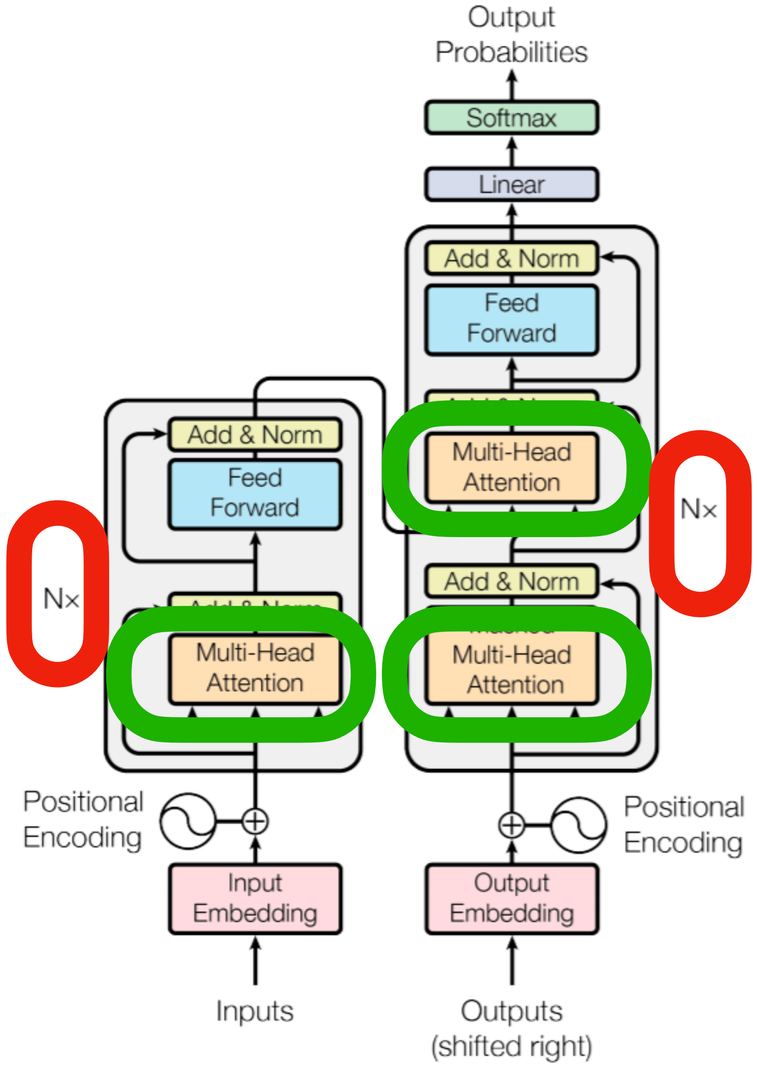


Figure 1: The Transformer - model architecture.

What is needed to train such models?

A useful list

- Selection + size of dataset

A useful list

- Selection + size of dataset

	BLOOM	/ours	NeoX	/ours	OPT	/ours	BloombergGPT
FinPile (old)	451	110%	460	112%	456	111%	412
C4	166	121%	170	123%	170	123%	138
The Pile	203	110%	214	116%	239	130%	184
Wikipedia	21	88%	23	99%	24	103%	24
Total	*390*	*113%*	*408*	*118%*	*434*	*126%*	*345*

Large Language Models

Source: BloombergGPT

Dataset	Docs 1e4	C/D	Chars 1e8	C/T	Toks 1e8	T%
FinPile	*175,886*	*1,017*	*17,883*	*4.92*	*3,635*	*51.27%*
Web	158,250	933	14,768	4.96	2,978	42.01%
News	10,040	1,665	1,672	4.44	376	5.31%
Filings	3,335	2,340	780	5.39	145	2.04%
Press	1,265	3,443	435	5.06	86	1.21%
Bloomberg	2,996	758	227	4.60	49	0.70%
PUBLIC	*50,744*	*3,314*	*16,818*	*4.87*	*3,454*	*48.73%*
C4	34,832	2,206	7,683	5.56	1,381	19.48%
Pile-CC	5,255	4,401	2,312	5.42	427	6.02%
GitHub	1,428	5,364	766	3.38	227	3.20%
Books3	19	552,398	1,064	4.97	214	3.02%
PubMed Central	294	32,181	947	4.51	210	2.96%
ArXiv	124	47,819	591	3.56	166	2.35%
OpenWebText2	1,684	3,850	648	5.07	128	1.80%
FreeLaw	349	15,381	537	4.99	108	1.52%
StackExchange	1,538	2,201	339	4.17	81	1.15%
DM Mathematics	100	8,193	82	1.92	43	0.60%
Wikipedia (en)	590	2,988	176	4.65	38	0.53%
USPTO Backgrounds	517	4,339	224	6.18	36	0.51%
PubMed Abstracts	1,527	1,333	204	5.77	35	0.50%
OpenSubtitles	38	31,055	119	4.90	24	0.34%
Gutenberg (PG-19)	3	399,351	112	4.89	23	0.32%
Ubuntu IRC	1	539,222	56	3.16	18	0.25%
EuroParl	7	65,053	45	2.93	15	0.21%
YouTubeSubtitles	17	19,831	33	2.54	13	0.19%
BookCorpus2	2	370,384	65	5.36	12	0.17%
HackerNews	82	5,009	41	4.87	8	0.12%
PhilPapers	3	74,827	23	4.21	6	0.08%
NIH ExPorter	92	2,165	20	6.65	3	0.04%
Enron Emails	24	1,882	5	3.90	1	0.02%
Wikipedia (7/1/22)	2,218	3,271	726	3.06	237	3.35%
TOTAL	*226,631*	*1,531*	*34,701*	*4.89*	*7,089*	*100.00%*

A useful list

- Selection + size of dataset

	BLOOM	/ours	NeoX	/ours	OPT	/ours	BloombergGPT
FinPile (old)	451	110%	460	112%	456	111%	412
C4	166	121%	170	123%	170	123%	138
The Pile	203	110%	214	116%	239	130%	184
Wikipedia	21	88%	23	99%	24	103%	24
Total	*390*	*113%*	*408*	*118%*	*434*	*126%*	*345*

Large Language Models

JFT-300M

Introduced by Sun et al. in Revisiting Unreasonable Effectiveness of Data in Deep Learning Era

JFT-300M is an internal Google dataset used for training image classification models. Images are labeled using an algorithm that uses complex mixture of raw web signals, connections between web-pages and user feedback. This results in over one billion labels for the 300M images (a single image can have multiple labels). Of the billion image labels, approximately 375M are selected via an algorithm that aims to maximize label precision of selected images.

Computer Vision Models

JFT-3B

Introduced by Zhai et al. in Scaling Vision Transformers

JFT-3B is an internal Google dataset and a larger version of the JFT-300M dataset. It consists of nearly 3 billion images, annotated with a class-hierarchy of around 30k labels via a semi-automatic pipeline. In other words, the data and associated labels are noisy.

Source: BloombergGPT

Dataset	Docs 1e4	C/D	Chars 1e8	C/T	Toks 1e8	T%
FinPile	*175,886*	*1,017*	*17,883*	*4.92*	*3,635*	*51.27%*
Web	158,250	933	14,768	4.96	2,978	42.01%
News	10,040	1,665	1,672	4.44	376	5.31%
Filings	3,335	2,340	780	5.39	145	2.04%
Press	1,265	3,443	435	5.06	86	1.21%
Bloomberg	2,996	758	227	4.60	49	0.70%
PUBLIC	*50,744*	*3,314*	*16,818*	*4.87*	*3,454*	*48.73%*
C4	34,832	2,206	7,683	5.56	1,381	19.48%
Pile-CC	5,255	4,401	2,312	5.42	427	6.02%
GitHub	1,428	5,364	766	3.38	227	3.20%
Books3	19	552,398	1,064	4.97	214	3.02%
PubMed Central	294	32,181	947	4.51	210	2.96%
ArXiv	124	47,819	591	3.56	166	2.35%
OpenWebText2	1,684	3,850	648	5.07	128	1.80%
FreeLaw	349	15,381	537	4.99	108	1.52%
StackExchange	1,538	2,201	339	4.17	81	1.15%
DM Mathematics	100	8,193	82	1.92	43	0.60%
Wikipedia (en)	590	2,988	176	4.65	38	0.53%
USPTO Backgrounds	517	4,339	224	6.18	36	0.51%
PubMed Abstracts	1,527	1,333	204	5.77	35	0.50%
OpenSubtitles	38	31,055	119	4.90	24	0.34%
Gutenberg (PG-19)	3	399,351	112	4.89	23	0.32%
Ubuntu IRC	1	539,222	56	3.16	18	0.25%
EuroParl	7	65,053	45	2.93	15	0.21%
YouTubeSubtitles	17	19,831	33	2.54	13	0.19%
BookCorpus2	2	370,384	65	5.36	12	0.17%
HackerNews	82	5,009	41	4.87	8	0.12%
PhilPapers	3	74,827	23	4.21	6	0.08%
NIH ExPorter	92	2,165	20	6.65	3	0.04%
Enron Emails	24	1,882	5	3.90	1	0.02%
Wikipedia (7/1/22)	2,218	3,271	726	3.06	237	3.35%
TOTAL	*226,631*	*1,531*	*34,701*	*4.89*	*7,089*	*100.00%*

A useful list

- Selection + size of dataset

Open source

	BLOOM	/ours	NeoX	/ours	OPT	/ours	BloombergGPT
FinPile (old)	451	110%	460	112%	456	111%	412
C4	166	121%	170	123%	170	123%	138
The Pile	203	110%	214	116%	239	130%	184
Wikipedia	21	88%	23	99%	24	103%	24
Total	*390*	*113%*	*408*	*118%*	*434*	*126%*	*345*

Large Language Models

JFT-300M

Introduced by Sun et al. in Revisiting Unreasonable Effectiveness of Data in Deep Learning Era

JFT-300M is an internal Google dataset used for training image classification models. Images are labeled using an algorithm that uses complex mixture of raw web signals, connections between web-pages and user feedback. This results in over one billion labels for the 300M images (a single image can have multiple labels). Of the billion image labels, approximately 375M are selected via an algorithm that aims to maximize label precision of selected images.

Computer Vision Models

JFT-3B

Introduced by Zhai et al. in Scaling Vision Transformers

JFT-3B is an internal Google dataset and a larger version of the JFT-300M dataset. It consists of nearly 3 billion images, annotated with a class-hierarchy of around 30k labels via a semi-automatic pipeline. In other words, the data and associated labels are noisy.

Source: BloombergGPT

Dataset	Docs 1e4	C/D	Chars 1e8	C/T	Toks 1e8	T%
FinPile	*175,886*	*1,017*	*17,883*	*4.92*	*3,635*	*51.27%*
Web	158,250	933	14,768	4.96	2,978	42.01%
News	10,040	1,665	1,672	4.44	376	5.31%
Filings	3,335	2,340	780	5.39	145	2.04%
Press	1,265	3,443	435	5.06	86	1.21%
Bloomberg	2,996	758	227	4.60	49	0.70%
PUBLIC	*50,744*	*3,314*	*16,818*	*4.87*	*3,454*	*48.73%*
C4	34,832	2,206	7,683	5.56	1,381	19.48%
Pile-CC	5,255	4,401	2,312	5.42	427	6.02%
GitHub	1,428	5,364	766	3.38	227	3.20%
Books3	19	552,398	1,064	4.97	214	3.02%
PubMed Central	294	32,181	947	4.51	210	2.96%
ArXiv	124	47,819	591	3.56	166	2.35%
OpenWebText2	1,684	3,850	648	5.07	128	1.80%
FreeLaw	349	15,381	537	4.99	108	1.52%
StackExchange	1,538	2,201	339	4.17	81	1.15%
DM Mathematics	100	8,193	82	1.92	43	0.60%
Wikipedia (en)	590	2,988	176	4.65	38	0.53%
USPTO Backgrounds					36	0.51%
PubMed Abstracts					35	0.50%
OpenSubtitles					24	0.34%
Gutenberg (PG-19)	3	399,551	112	4.89	23	0.32%
Ubuntu IRC	1	539,222	56	3.16	18	0.25%
EuroParl	7	65,053	45	2.93	15	0.21%
YouTubeSubtitles	17	19,831	33	2.54	13	0.19%
BookCorpus2	2	370,384	65	5.36	12	0.17%
HackerNews	82	5,009	41	4.87	8	0.12%
PhilPapers	3	74,827	23	4.21	6	0.08%
NIH ExPorter	92	2,165	20	6.65	3	0.04%
Enron Emails	24	1,882	5	3.90	1	0.02%
Wikipedia (7/1/22)	2,218	3,271	726	3.06	237	3.35%
TOTAL	*226,631*	*1,531*	*34,701*	*4.89*	*7,089*	*100.00%*

Open source

A useful list

- Selection + size of dataset

Open source

	BLOOM	/ours	NeoX	/ours	OPT	/ours	BloombergGPT
FinPile (old)	451	110%	460	112%	456	111%	412
C4	166	121%	170	123%	170	123%	138
The Pile	203	110%	214	116%	239	130%	184
Wikipedia	21	88%	23	99%	24	103%	24
Total	*390*	*113%*	*408*	*118%*	*434*	*126%*	*345*

Large Language Models

JFT-300M

Introduced by Sun et al. in Revisiting Unreasonable Effectiveness of Data in Deep Learning Era

Proprietary

JFT-300M is an internal Google dataset us dels. Images are labeled using an algorithm that uses complex mixture of raw eb-pages and user feedback. This results in over one billion labels for the 300M images (a single image can have multiple labels). Of the billion image labels, approximately 375M are selected via an algorithm that aims to maximize label precision of selected images.

Computer Vision Models

JFT-3B

Introduced by Zhai et al. in Scaling Vision Transformers

Proprietary

JFT-3B is an internal Google dataset and a lar consists of nearly 3 billion images, annotated with a class-hierarchy of around ne. In other words, the data and associated labels are noisy.

Source: BloombergGPT

Dataset	Docs 1e4	C/D	Chars 1e8	C/T	Toks 1e8	T%
FinPile	*175,886*	*1,017*	*17,883*	*4.92*	*3,635*	*51.27%*
Web	[illegible]	[illegible]	[illegible]	96	2,978	42.01%
News	[illegible]	[illegible]	[illegible]	44	376	5.31%
Filings	[illegible]	[illegible]	[illegible]	39	145	2.04%
Press	[illegible]	[illegible]	[illegible]	06	86	1.21%
Bloomberg	2,996	758	227	4.60	49	0.70%
PUBLIC	*50,744*	*3,314*	*16,818*	*4.87*	*3,454*	*48.73%*
C4	34,832	2,206	7,683	5.56	1,381	19.48%
Pile-CC	5,255	4,401	2,312	5.42	427	6.02%
GitHub	1,428	5,364	766	3.38	227	3.20%
Books3	19	552,398	1,064	4.97	214	3.02%
PubMed Central	294	32,181	947	4.51	210	2.96%
ArXiv	124	47,819	591	3.56	166	2.35%
OpenWebText2	1,684	3,850	648	5.07	128	1.80%
FreeLaw	349	15,381	537	4.99	108	1.52%
StackExchange	1,538	2,201	339	4.17	81	1.15%
DM Mathematics	100	8,193	82	1.92	43	0.60%
Wikipedia (en)	590	2,988	176	4.65	38	0.53%
USPTO Backgrounds	[illegible]	[illegible]	[illegible]	[illegible]	36	0.51%
PubMed Abstracts	[illegible]	[illegible]	[illegible]	[illegible]	35	0.50%
OpenSubtitles	[illegible]	[illegible]	[illegible]	[illegible]	24	0.34%
Gutenberg (PG-19)	3	399,551	112	4.89	23	0.32%
Ubuntu IRC	1	539,222	56	3.16	18	0.25%
EuroParl	7	65,053	45	2.93	15	0.21%
YouTubeSubtitles	17	19,831	33	2.54	13	0.19%
BookCorpus2	2	370,384	65	5.36	12	0.17%
HackerNews	82	5,009	41	4.87	8	0.12%
PhilPapers	3	74,827	23	4.21	6	0.08%
NIH ExPorter	92	2,165	20	6.65	3	0.04%
Enron Emails	24	1,882	5	3.90	1	0.02%
Wikipedia (7/1/22)	2,218	3,271	726	3.06	237	3.35%
TOTAL	*226,631*	*1,531*	*34,701*	*4.89*	*7,089*	*100.00%*

Proprietary

Open source

A useful list

- Selection + size of dataset
- How to preprocess/clean data

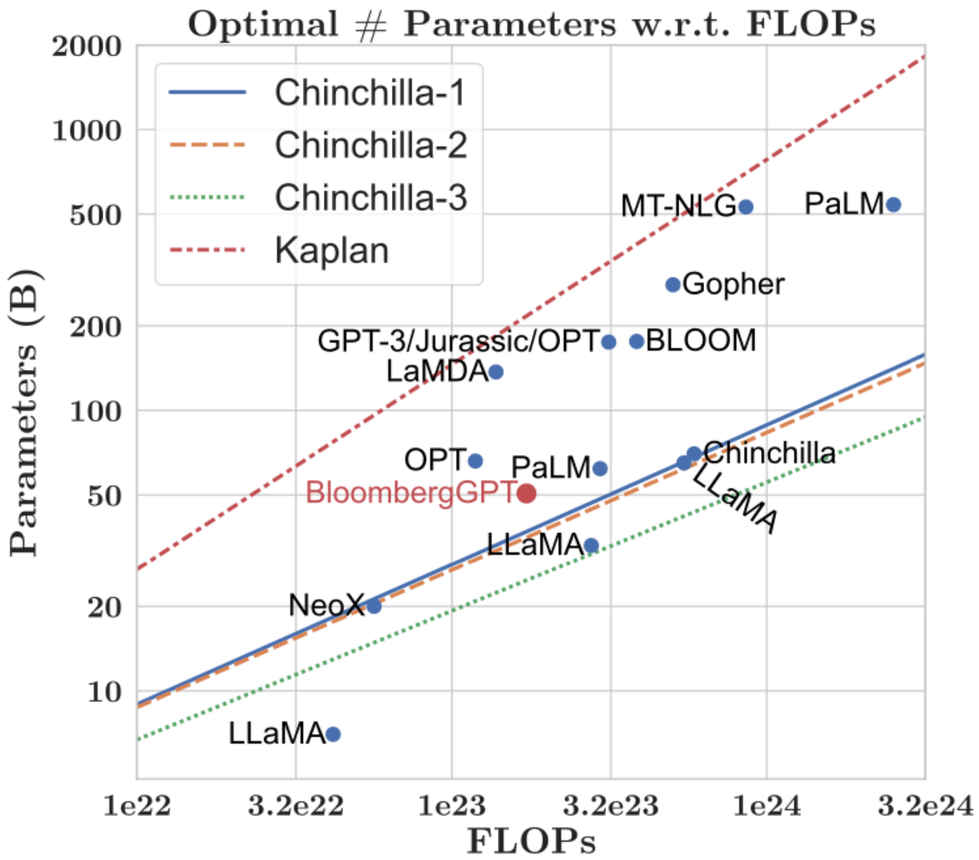
Out of scope: tokenization, deduplication, …

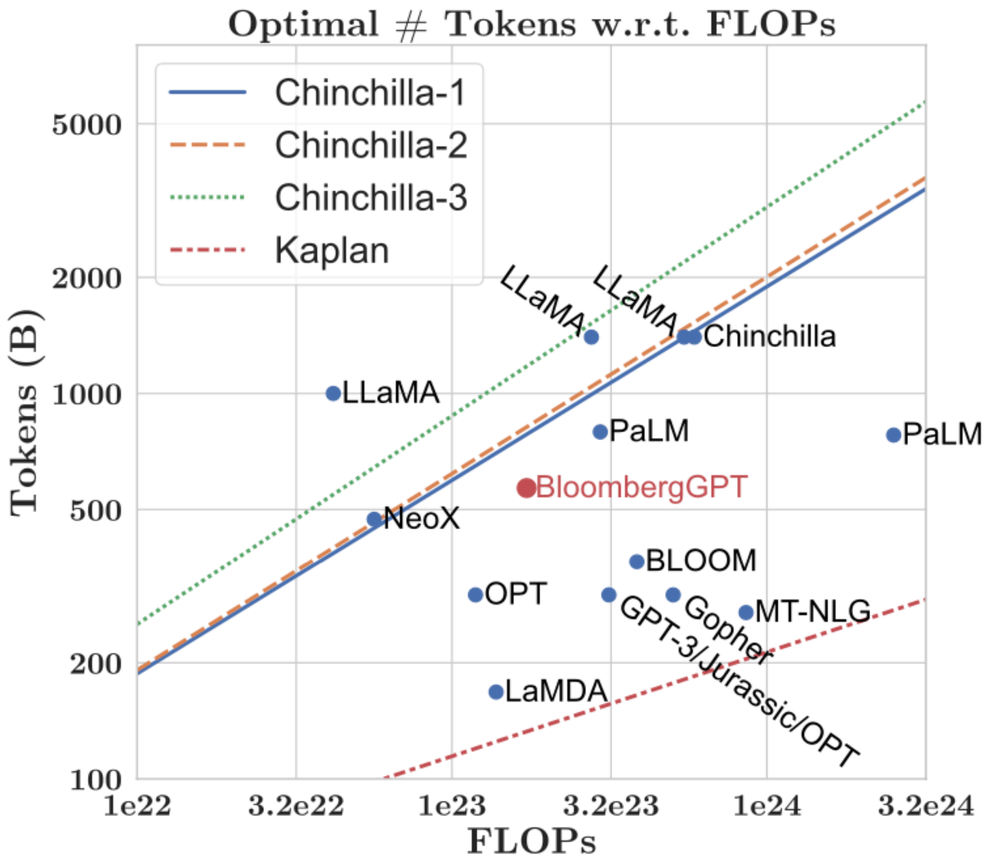
A useful list

Source: BloombergGPT

- Selection + size of dataset
- How to preprocess/clean data
- Choose the size of the model

Idea: it is not always good to have the biggest model if your dates small, and vice versa





A useful list

- Selection + size of dataset
- How to preprocess/clean data
- Choose the size of the model
- Select hyperparameters
- Use of efficient distributed computing protocols

+

A useful list

Source: On Efficient Training of Large-Scale Deep Learning

- Use of efficient distributed computing protocols

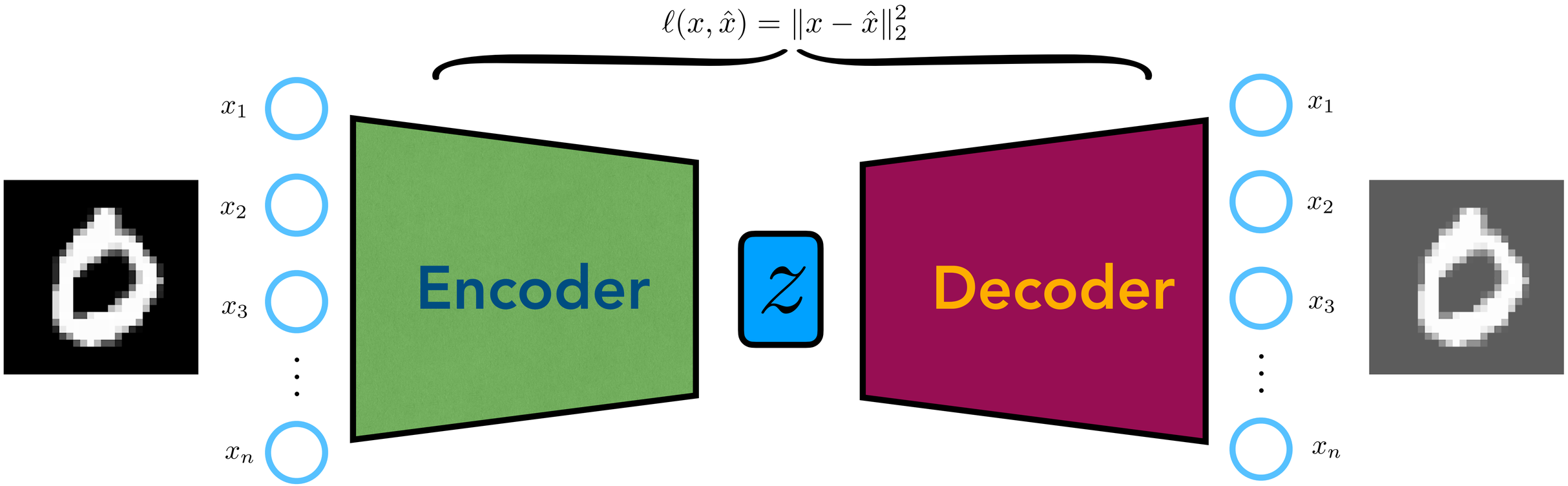
Table 5: Frameworks for efficient training.

Hosters	Libraries	Benefits
PyTorch	PyTorch 2.0	*torch.compile*, mixed precision, DDP, and FSDP
Baidu	PaddlePaddle	3D parallelism, mixed precision, GroupSharded (ZeRO-like), model implementations
ByteDance	LightSeq	mixed precision, fused operators, GEMM optimization
Google	JAX	Autograd, XLA
MicroSoft	DeepSpeed	3D parallelism, ZeRO, mixed precision, data efficiency library
	TorchScale	scaling up Transformers
MetaResearch	FairScale	FSDP, pipeline parallel, general techniques, Adascale
	Fairseq	FSDP, general techniques, model implementations
	xFormers	Transformer model implementations
MosaicML	Composer	multi-GPUs training, implementations of speedup methods
Huggingface	Transformers	model implementations
	Accelerate	distributed training, mixed precision, support of FSDP, DeepSpeed, and Megatron-LM
HPC-AI Tech	Colossal-AI	3D(especially for tensor) parallelism, sequence parallelism, ZeRO
OneFlow	LiBai	3D parallelism, mixed precision, ZeRO, general techniques
NVIDIA	Megatron-LM	distributed pretraining, model implementations
	Apex	AMP, distributed training, fused optimizers, fused layers, and layer norm

``Datasets can be difficult to find - what then?''

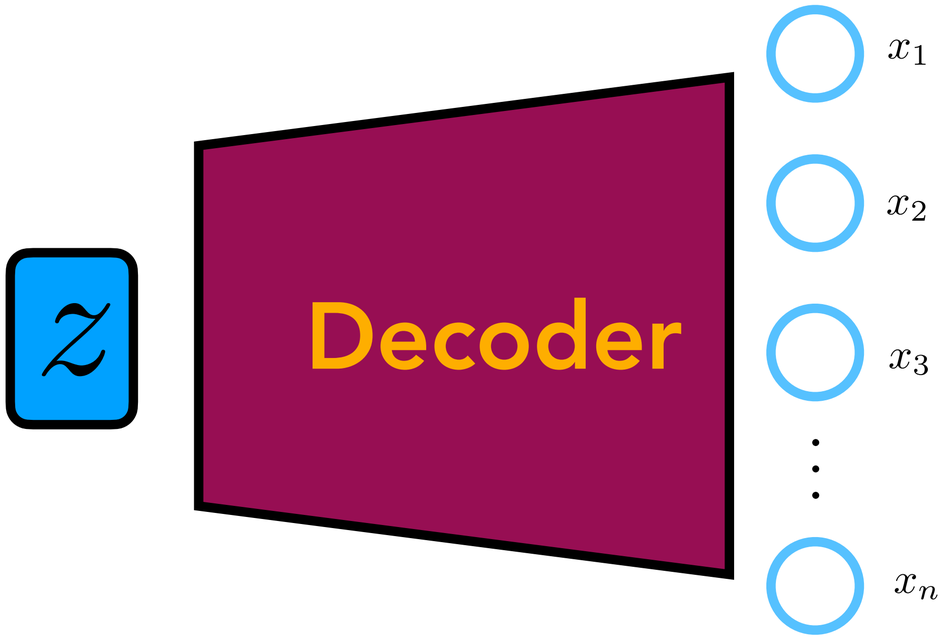
Remember Autoencoders

- Autoencoders rely on the fact that there exists a dataset on which you train



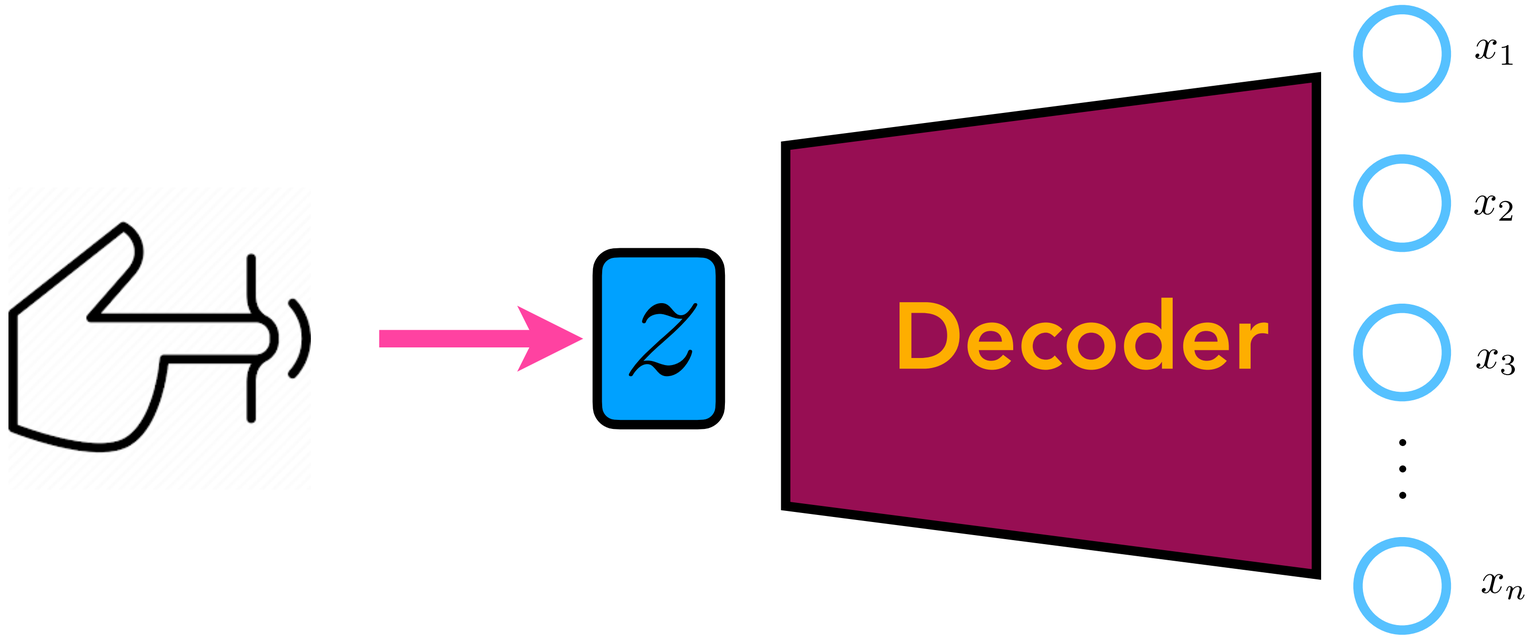
Remember Autoencoders

- Desired goal: find a decoder such that



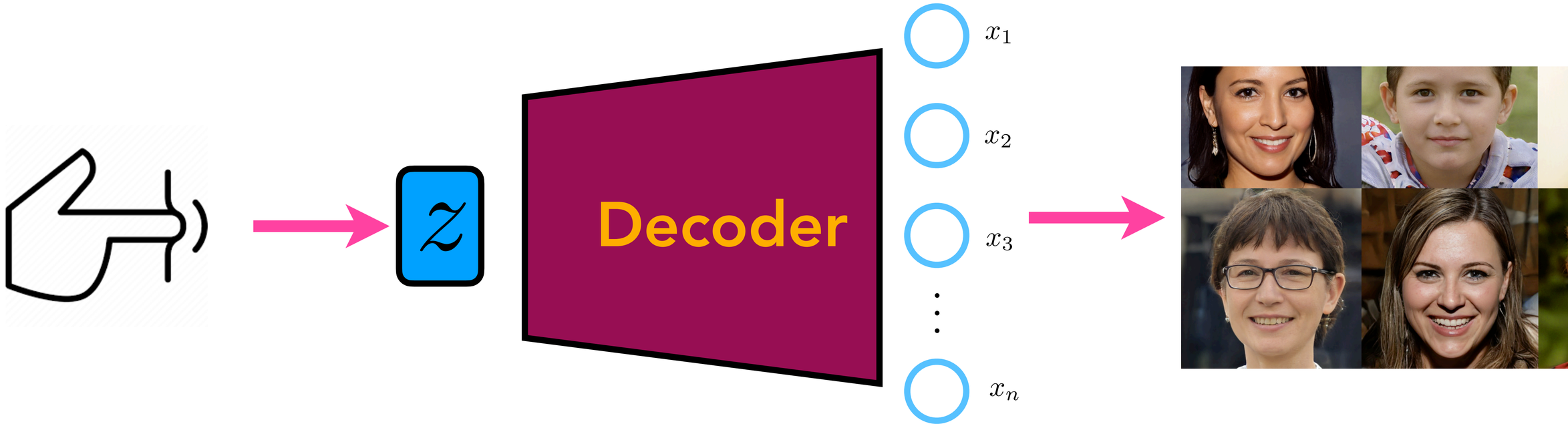
Remember Autoencoders

- Desired goal: find a decoder such that



Remember Autoencoders

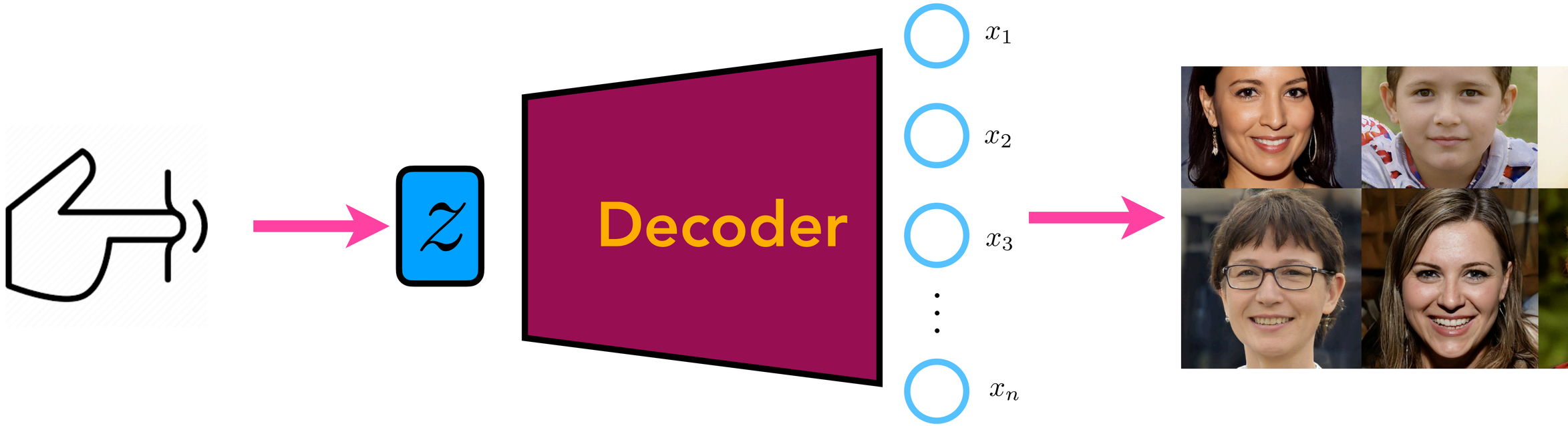
- Desired goal: find a decoder such that



- ``Ideal'' scenario: we insert a random low-dimensional vector z and we get data

Remember Autoencoders

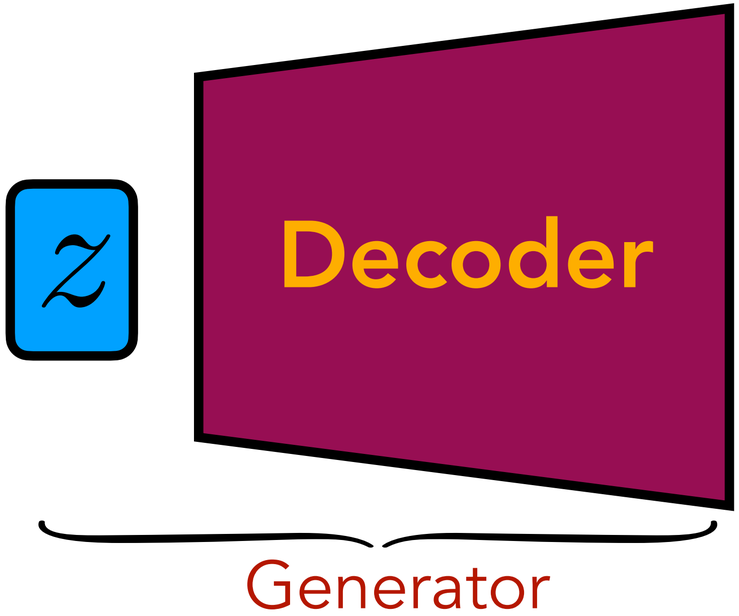
- Desired goal: find a decoder such that



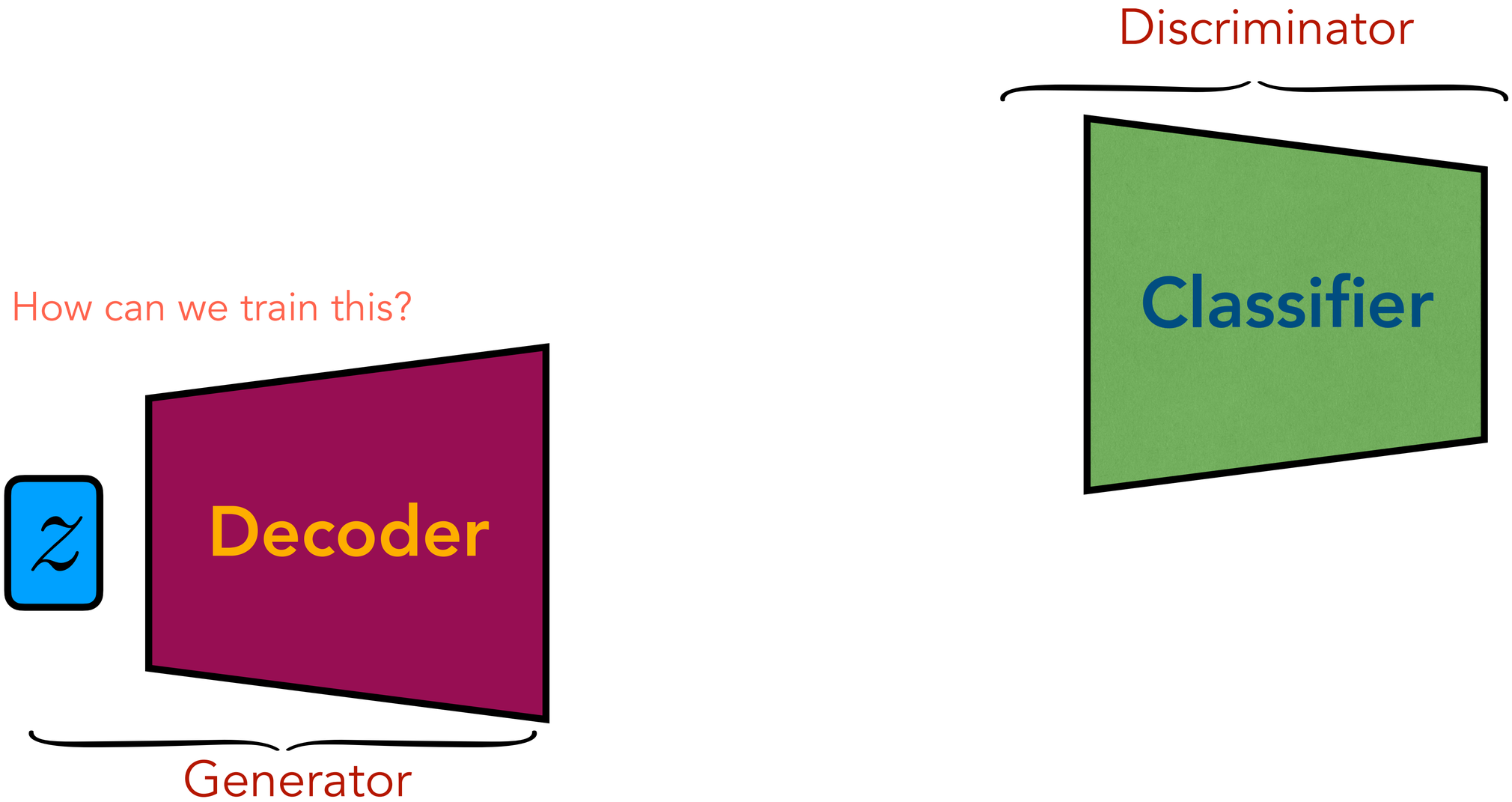
- ``Ideal'' scenario: we insert a random low-dimensional vector z and we get data
- Why ``random''? Easy to sample from and we just need to create data

Generative Adversarial Networks (GANs)

How can we train this?

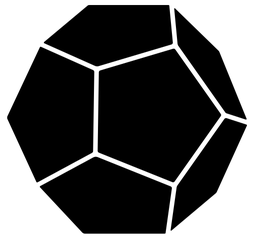


Generative Adversarial Networks (GANs)

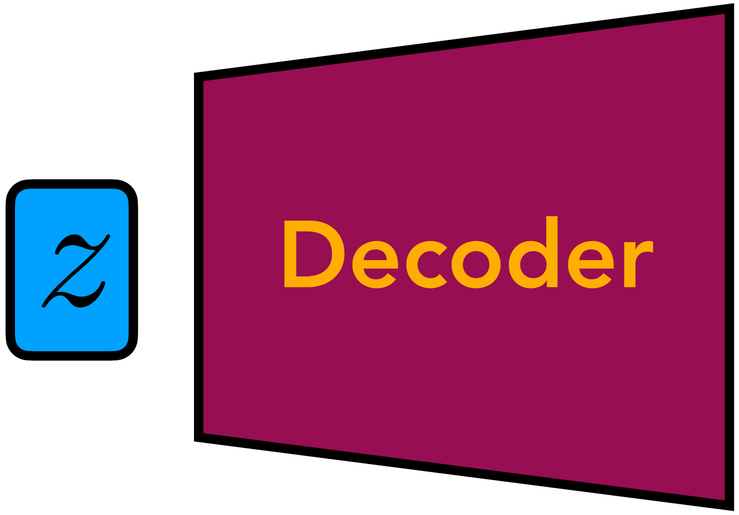


Generative Adversarial Networks (GANs)

Tank of Real Data

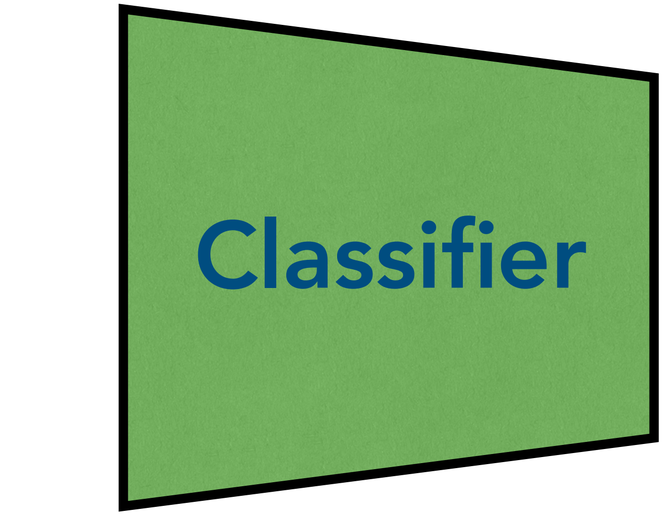


How can we train this?

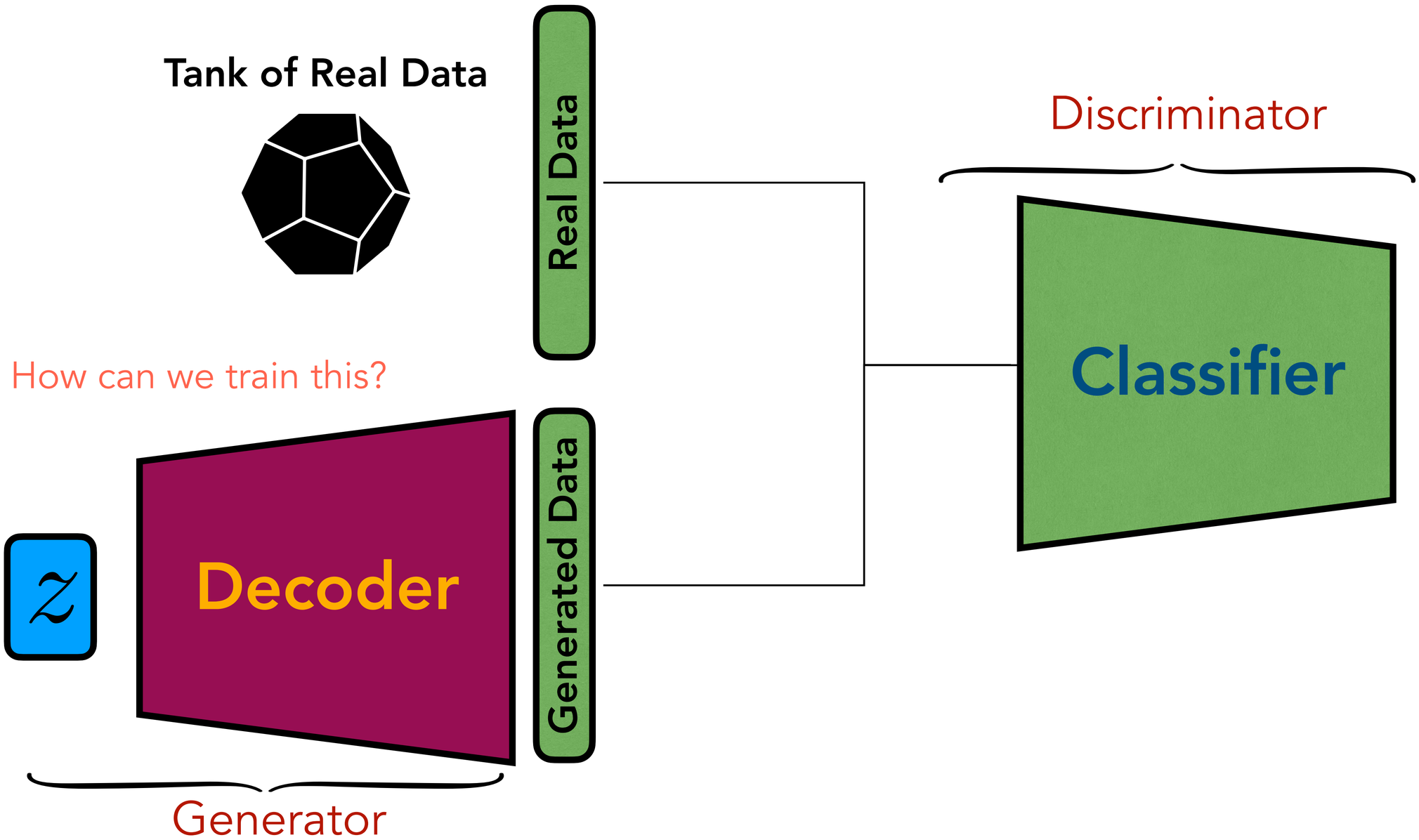


Generator

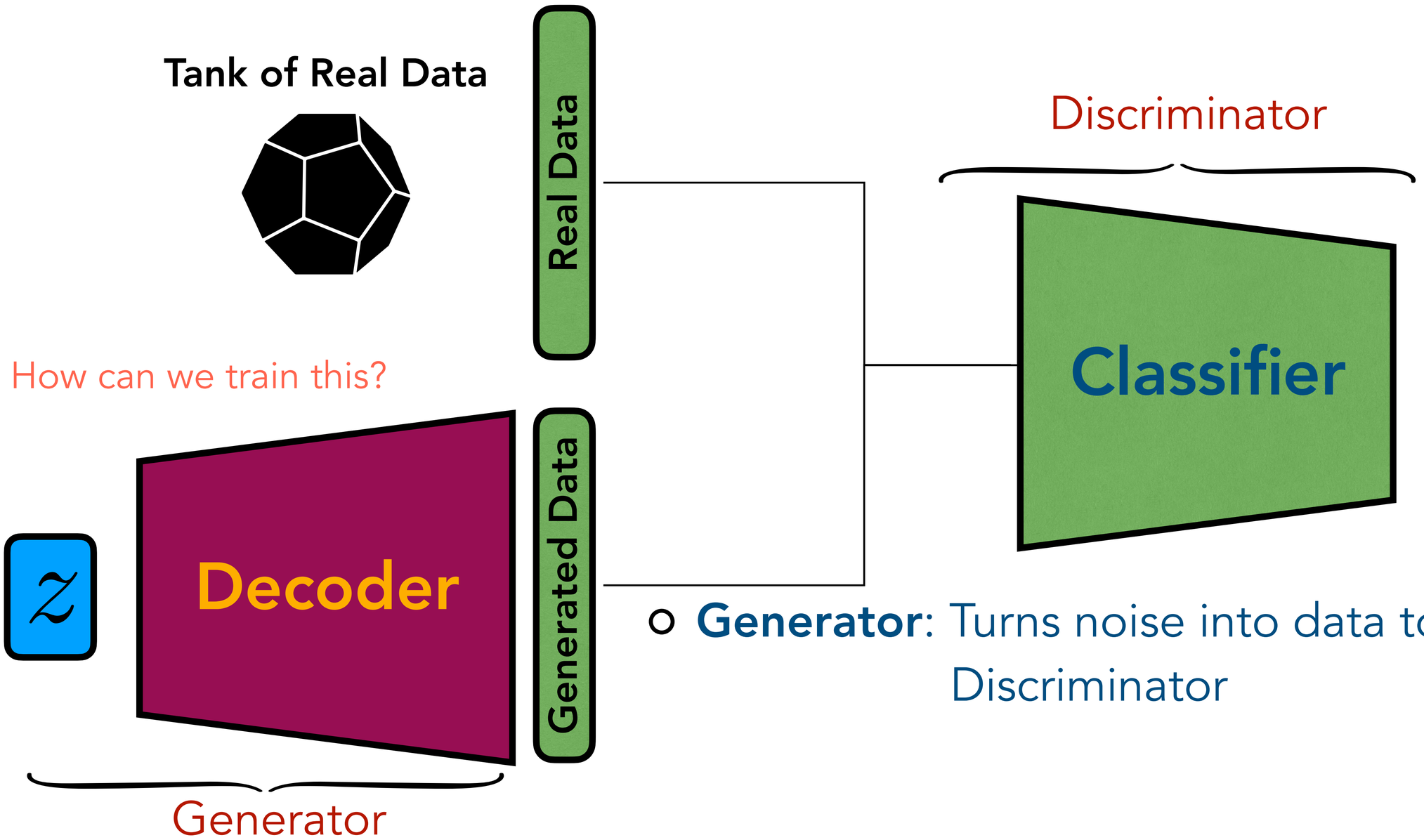
Discriminator



Generative Adversarial Networks (GANs)

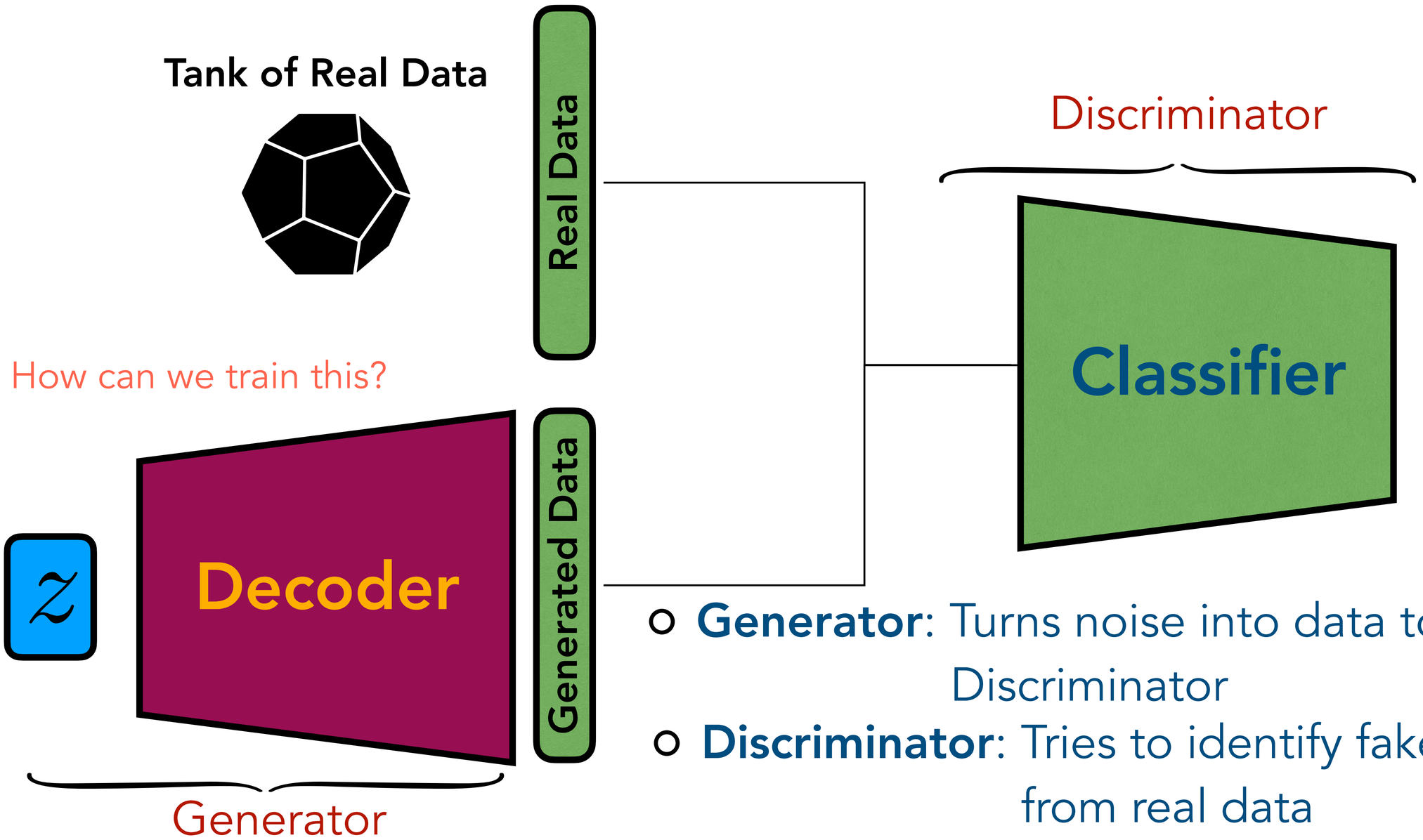


Generative Adversarial Networks (GANs)



- **Generator**: Turns noise into data to ``trick'' Discriminator

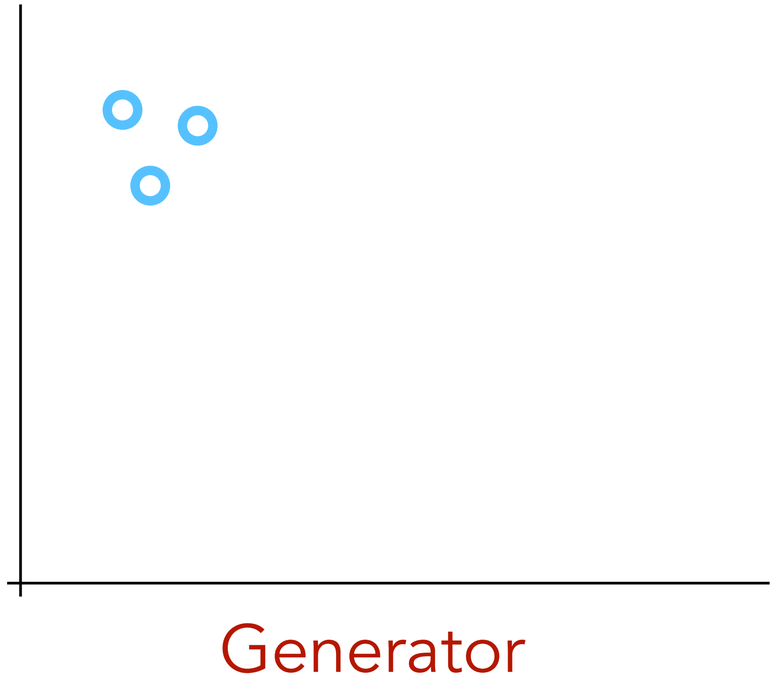
Generative Adversarial Networks (GANs)



- **Generator**: Turns noise into data to ``trick'' Discriminator
- **Discriminator**: Tries to identify fake data from real data

Generative Adversarial Networks (GANs)

- Simple example



Generative Adversarial Networks (GANs)

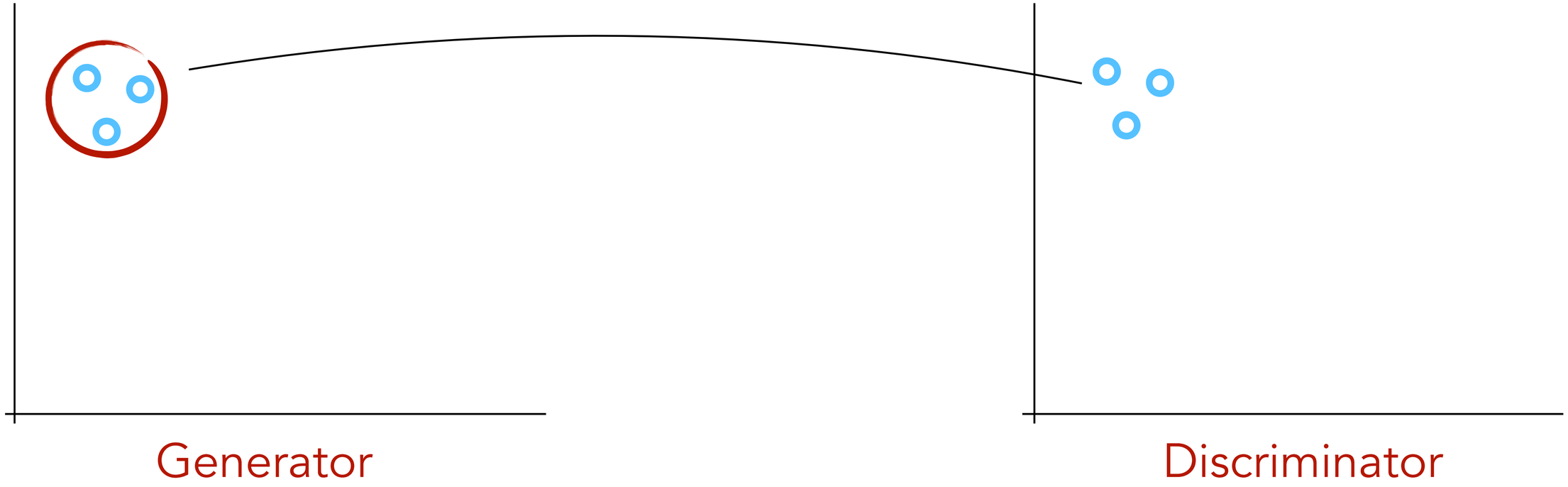
- Simple example

Generator

Discriminator

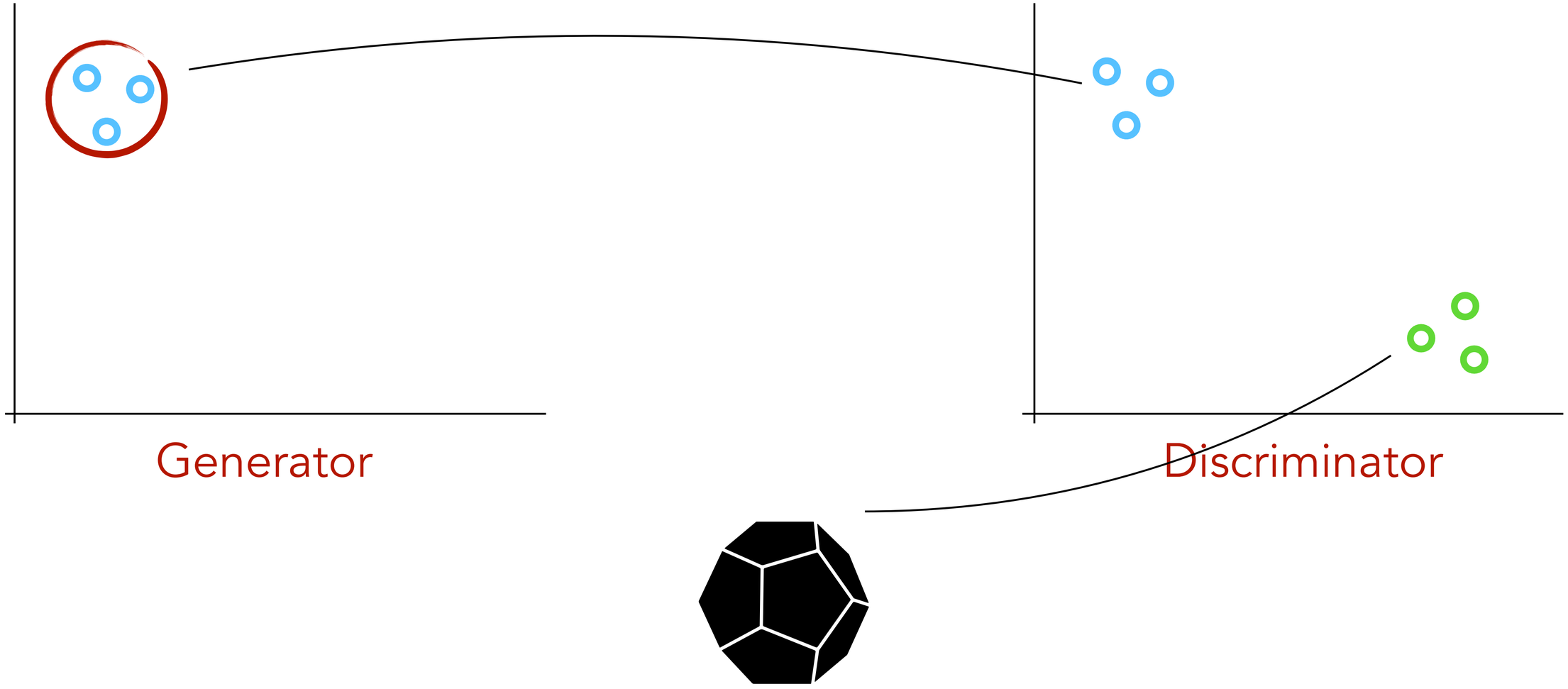
Generative Adversarial Networks (GANs)

- Simple example



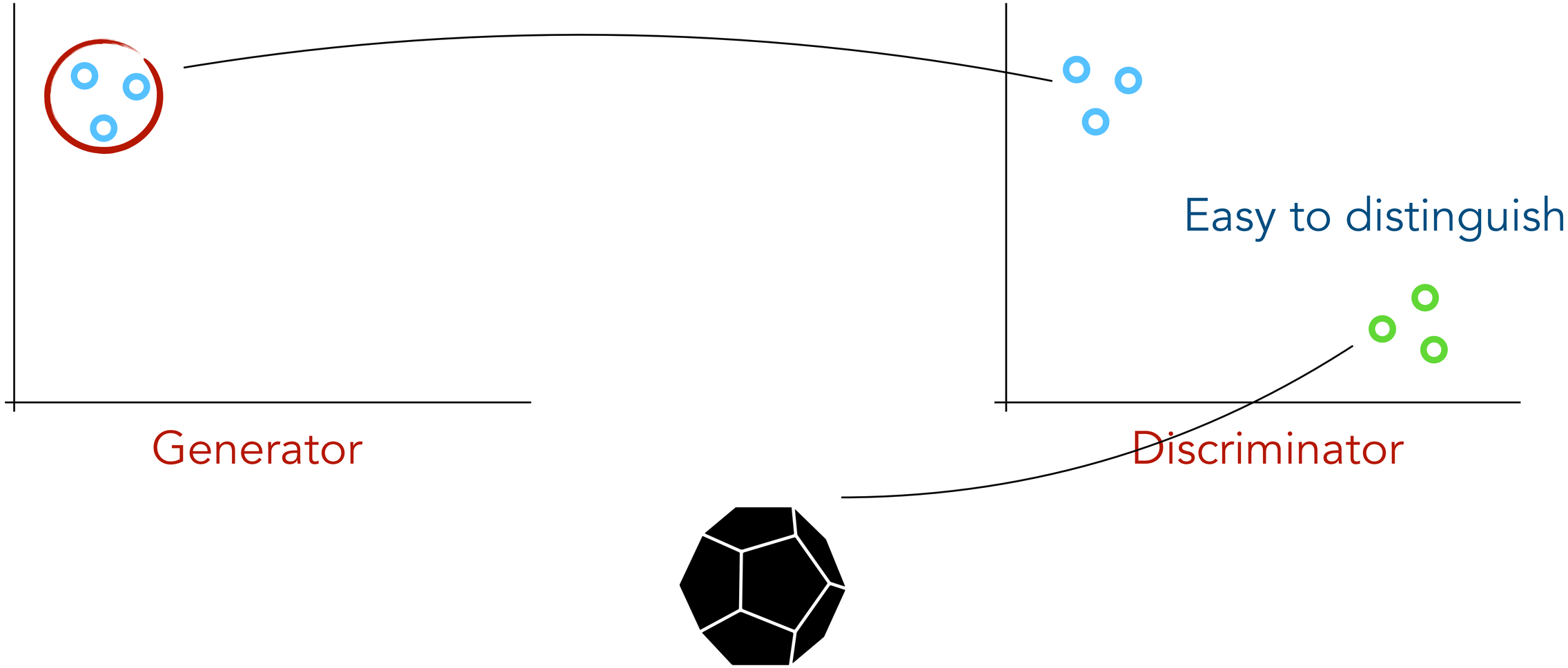
Generative Adversarial Networks (GANs)

- Simple example



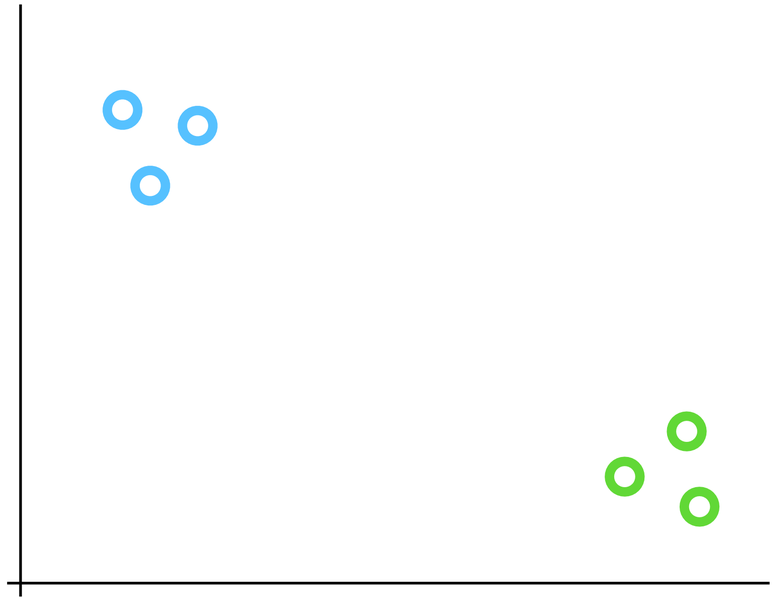
Generative Adversarial Networks (GANs)

- Simple example



Generative Adversarial Networks (GANs)

- Simple example

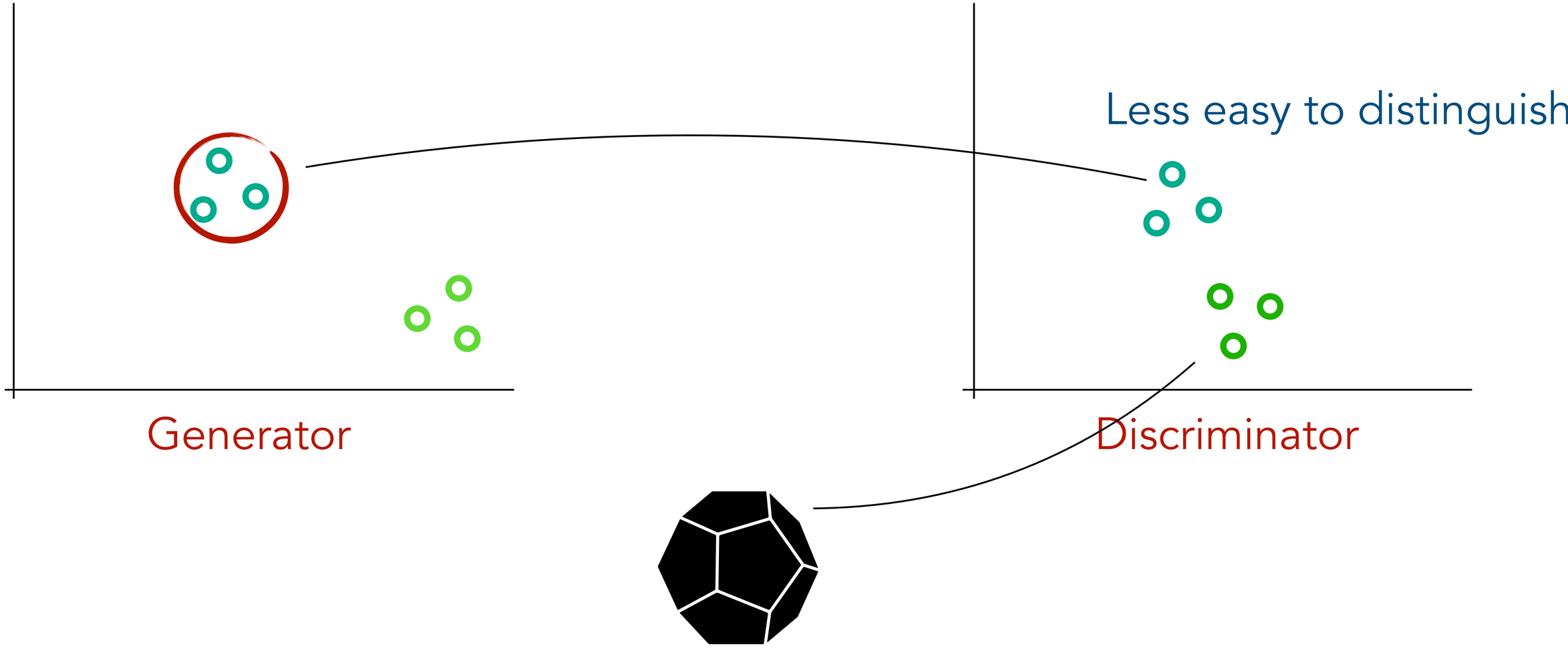


Generator

Discriminator

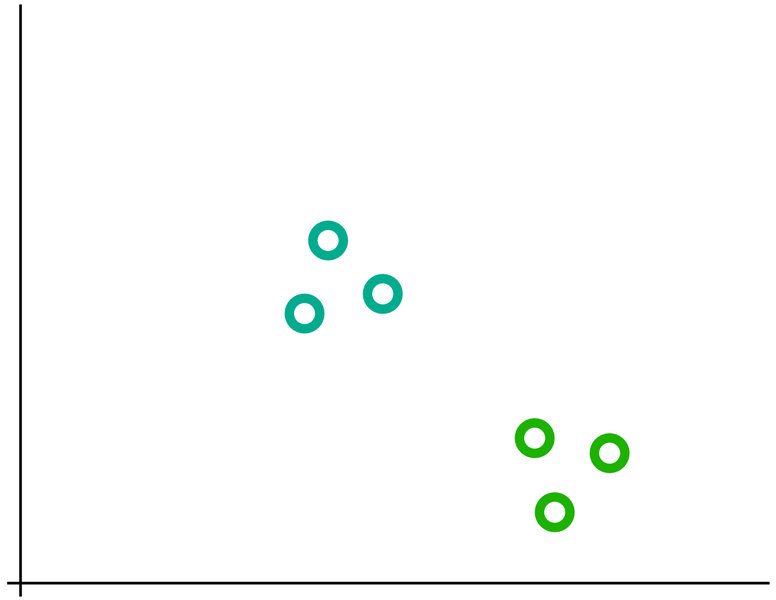
Generative Adversarial Networks (GANs)

- Simple example



Generative Adversarial Networks (GANs)

- Simple example

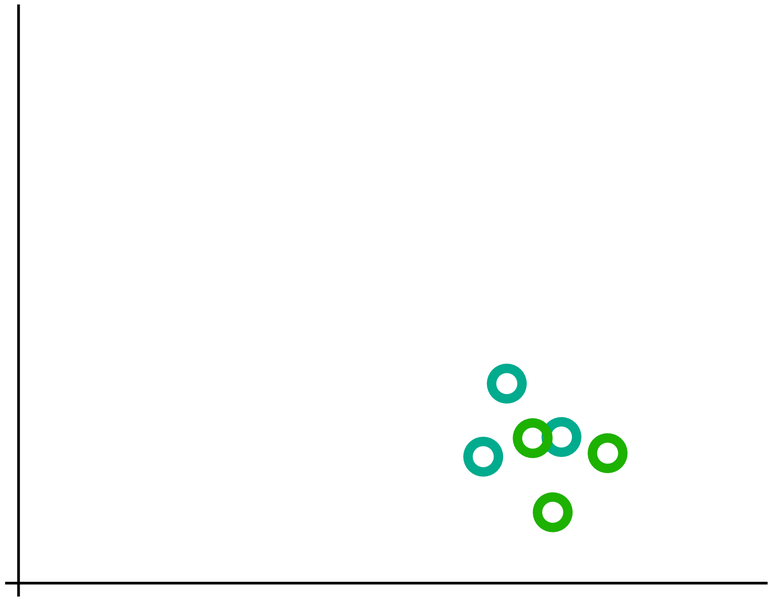


Generator

Discriminator

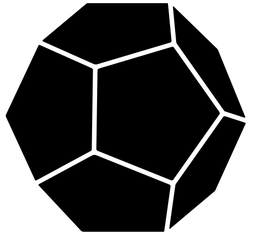
Generative Adversarial Networks (GANs)

- Simple example



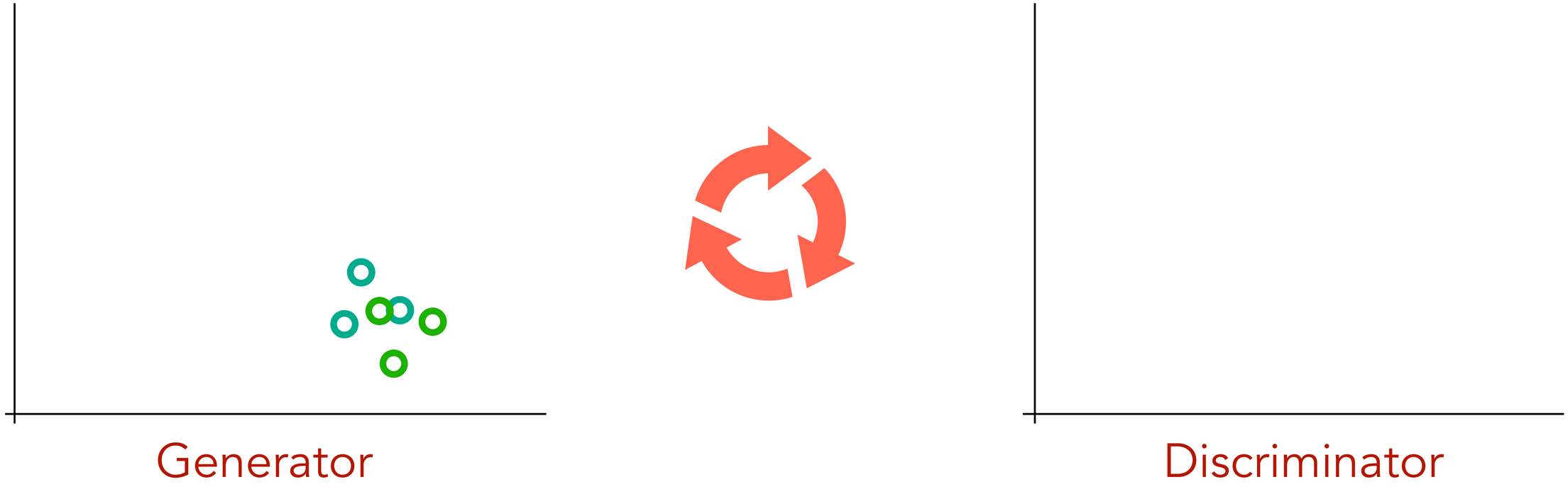
Generator

Discriminator



Generative Adversarial Networks (GANs)

- Simple example

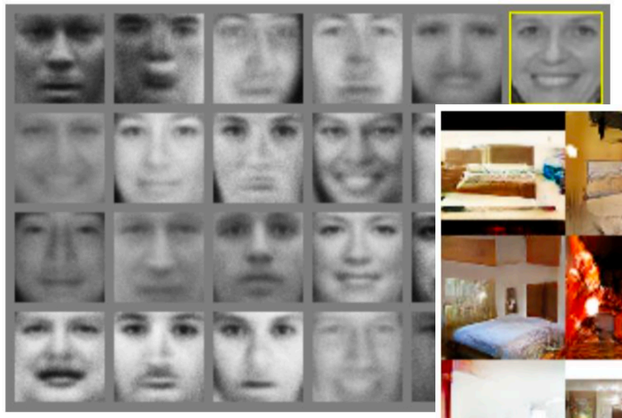


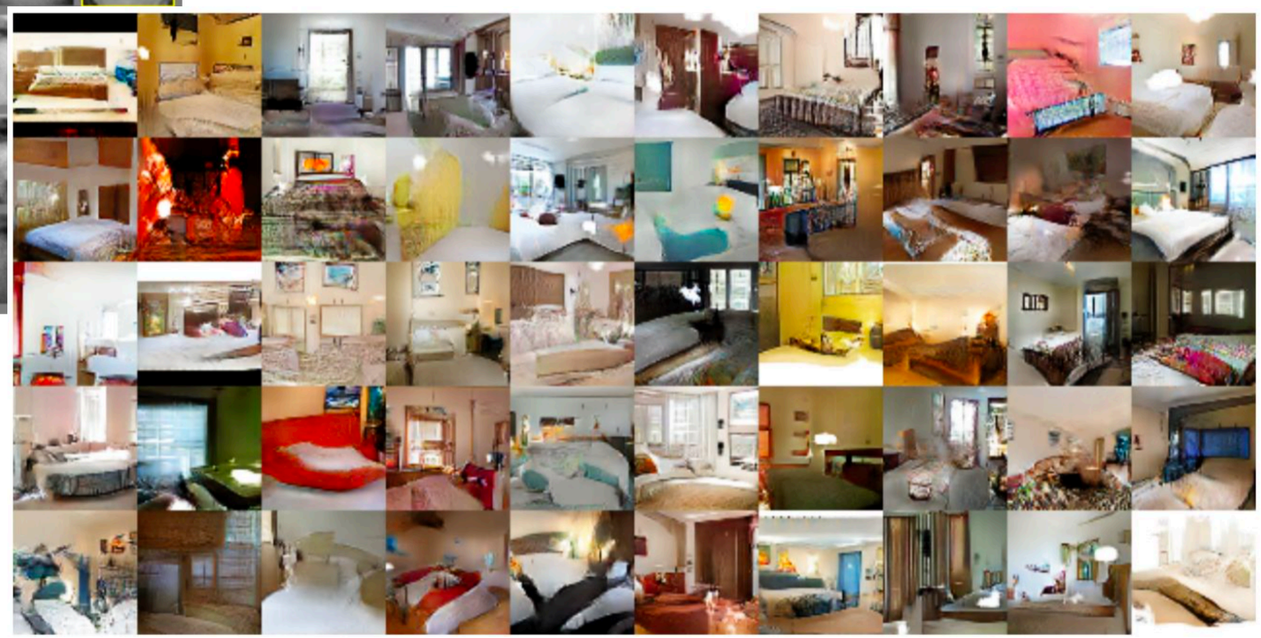
- Twist: GANs introduced game theory ideas in training neural networks

$$\arg\min_{G}\max_{D}\mathbb{E}_{z,x}\left[\log D(G(z)) + \log(1 - D(x))\right]$$

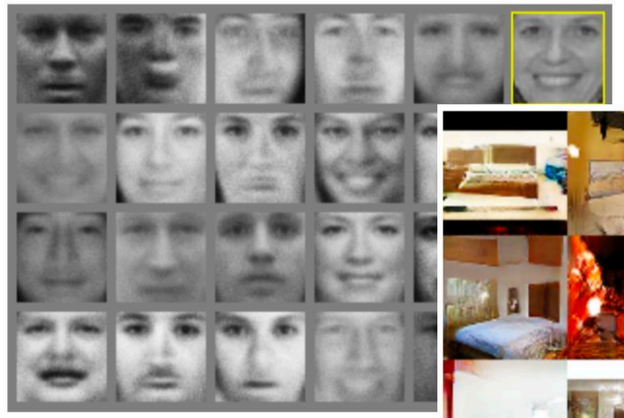
Historical reference: GANs examples

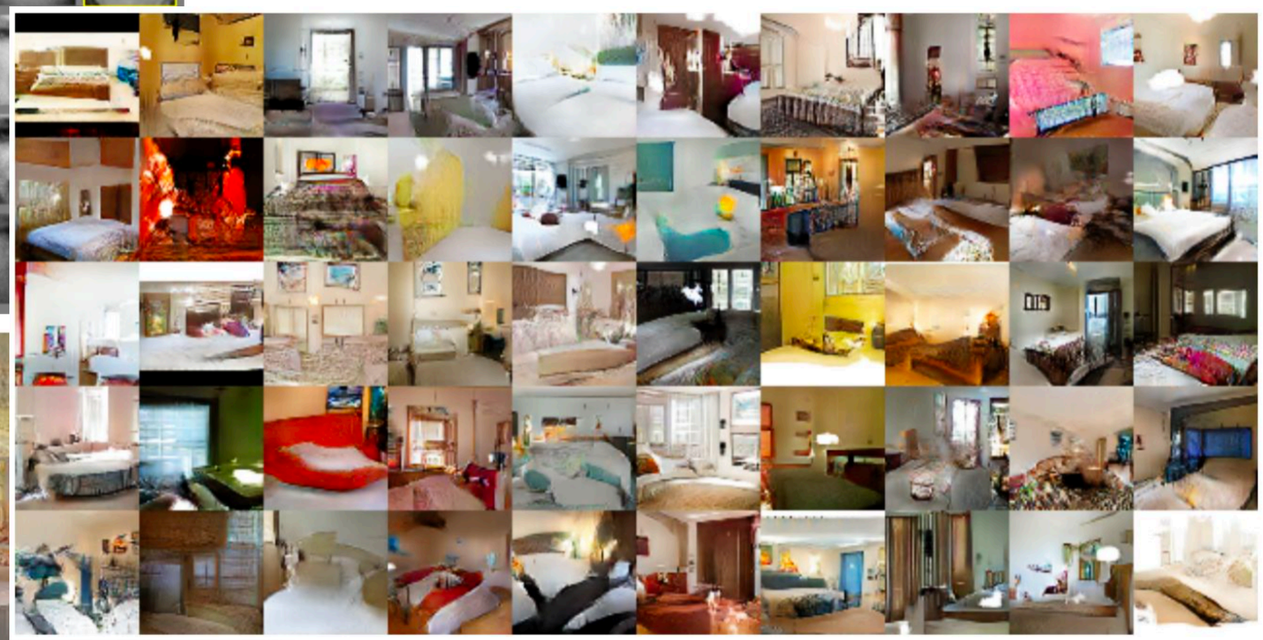
Historical reference: GANs examples

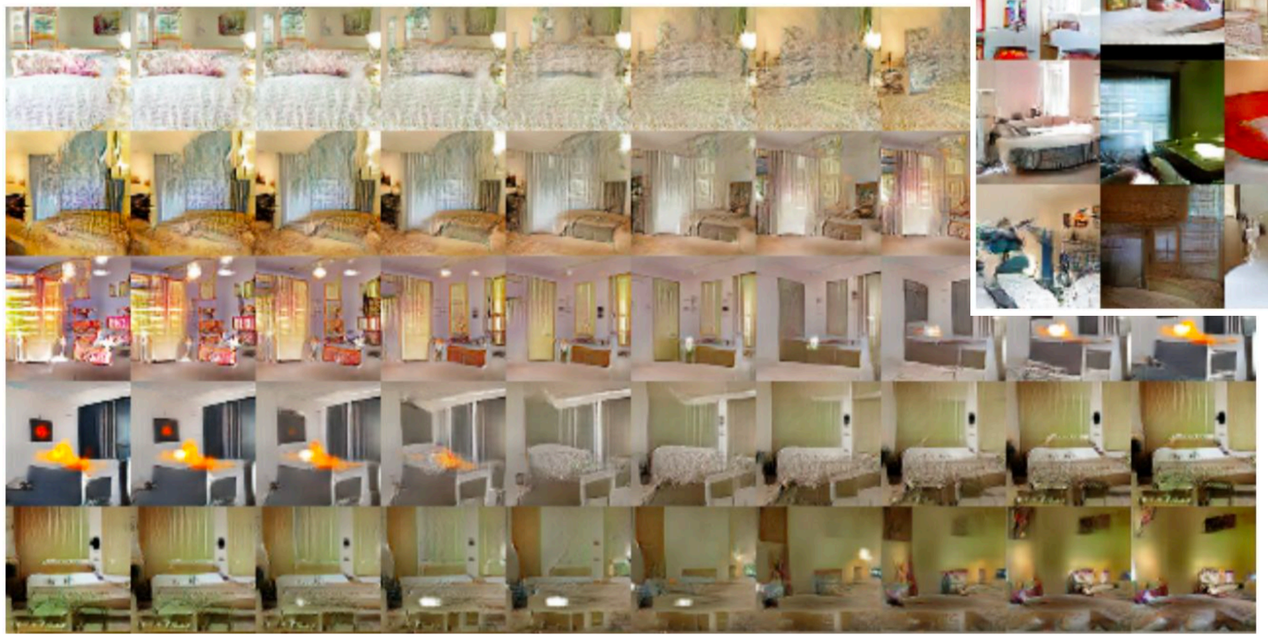




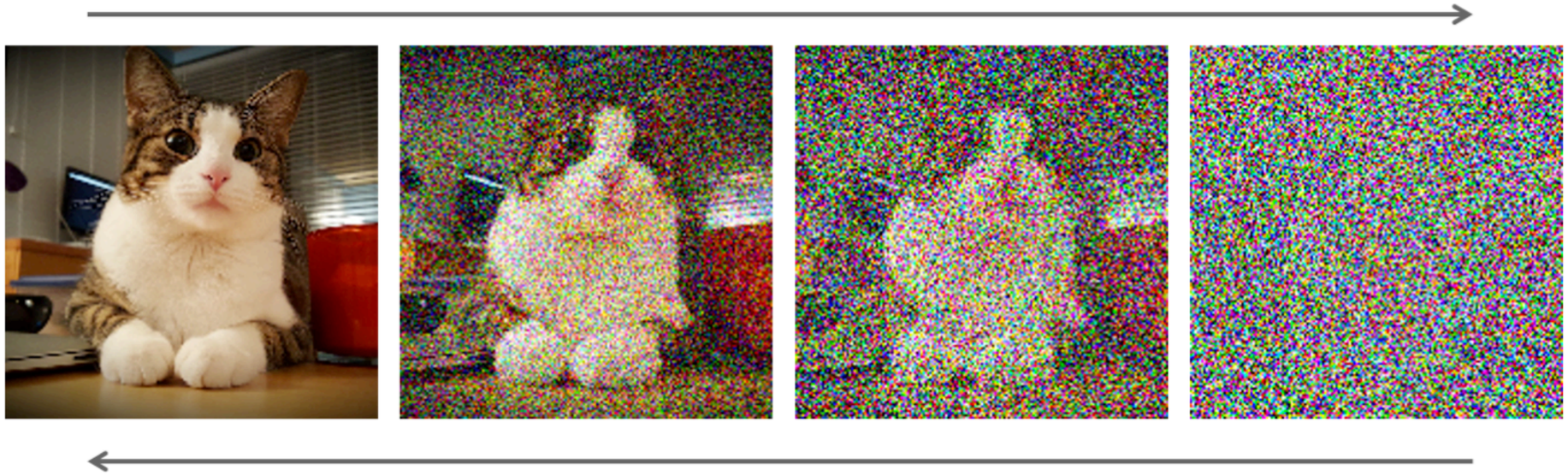
Historical reference: GANs examples







State of the art in GenAI: Diffusion models



Models in 2025

Source: LifeArchitect.ai

- Large Language Models
- The emergence of large datasets

All this is great but…

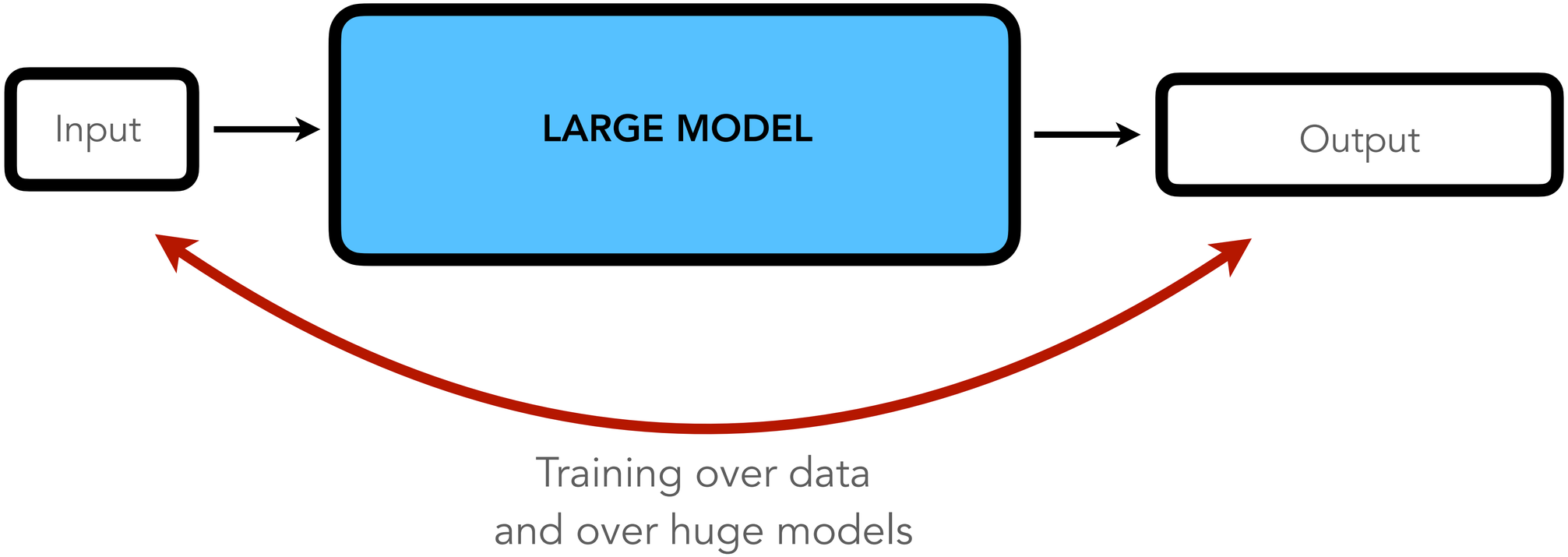
(Democratization backslide, green AI..)

Why can’t we just reuse some of these models?

(Pretrained models, fine-tuning, adaptation)

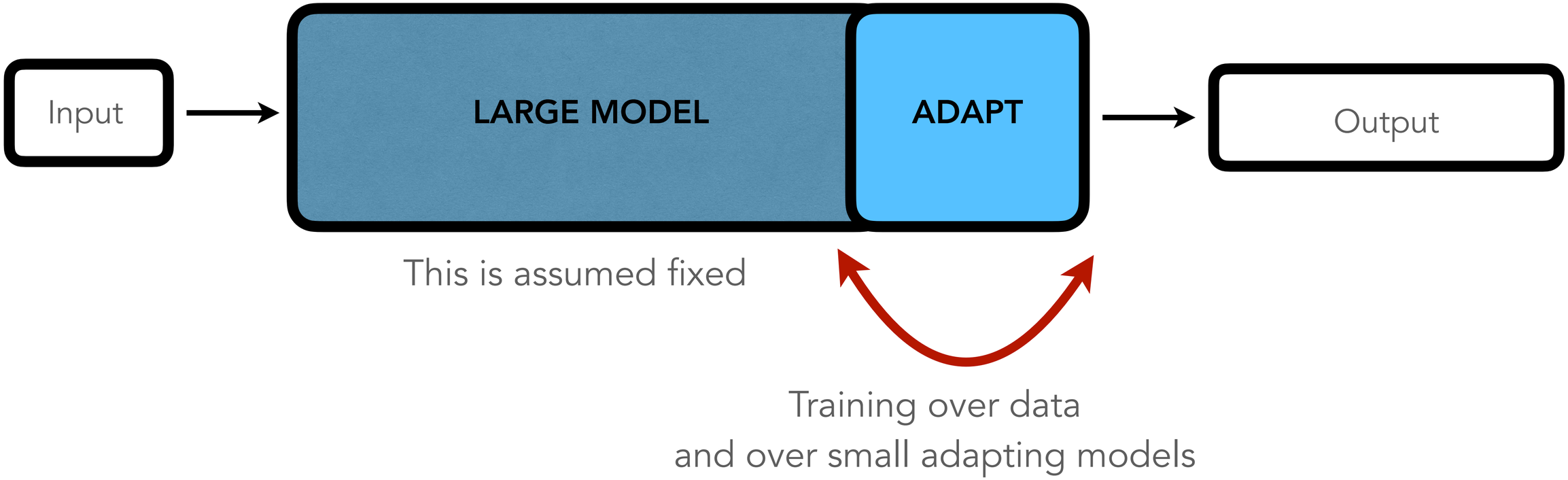
Reusing acquired knowledge + adapting

- A lot of "action" in this field. The idea is simple:



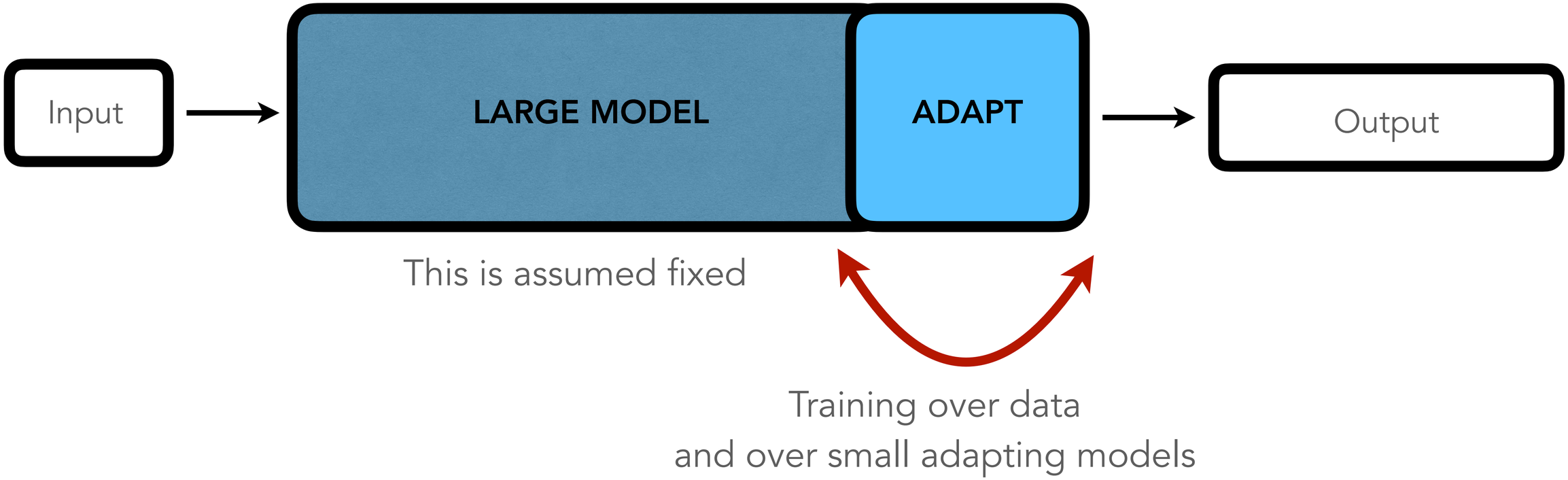
Reusing acquired knowledge + adapting

- A lot of "action" in this field. The idea is simple:



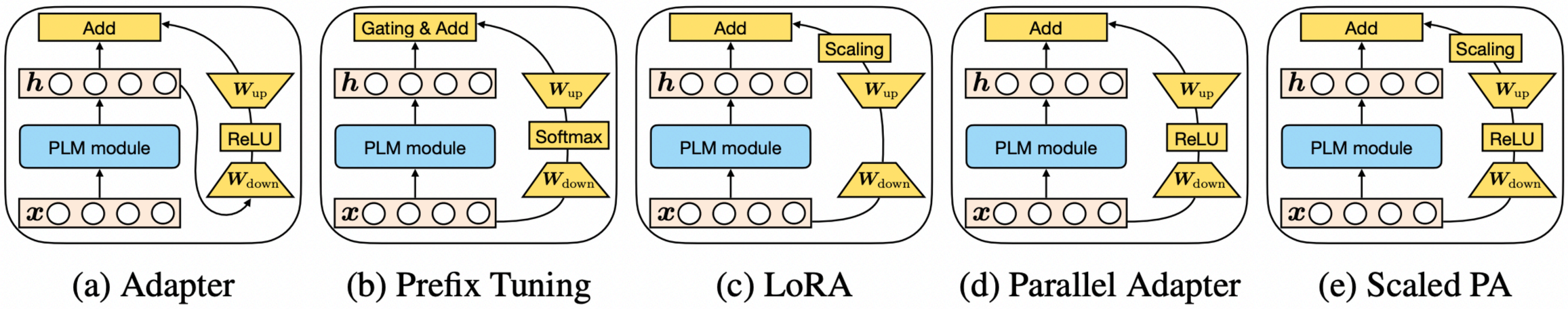
Reusing acquired knowledge + adapting

- A lot of "action" in this field. The idea is simple:



- This idea is super handy in scenarios like transfer learning, continual learning, new task learning, etc..

How does it look like in practice?



(a) Adapter (b) Prefix Tuning (c) LoRA (d) Parallel Adapter (e) Scaled PA

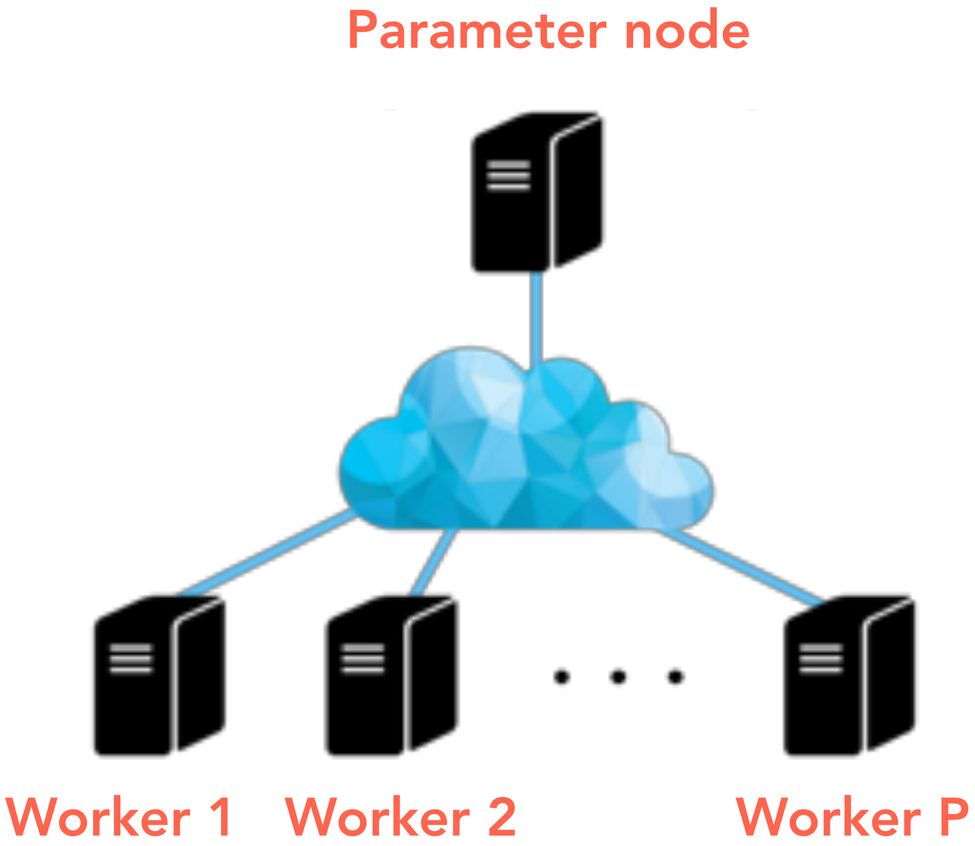
Figure 3: Graphical illustration of existing methods and the proposed variants. "PLM module" represents a certain sublayer of the PLM (e.g. attention or FFN) that is frozen. "Scaled PA" denotes scaled parallel adapter. We do not include multi-head parallel adapter here to save space.

- Examples include LoRA [Hu et al., 2021], Adapters [Houlsby et al., 2019], Prefix tuning [Li & Liang, 2021].
- Lead companies in AI (Google, Microsoft, etc) put some efforts recently (this idea helps also towards green AI)

Federated Learning and ML via edge computing

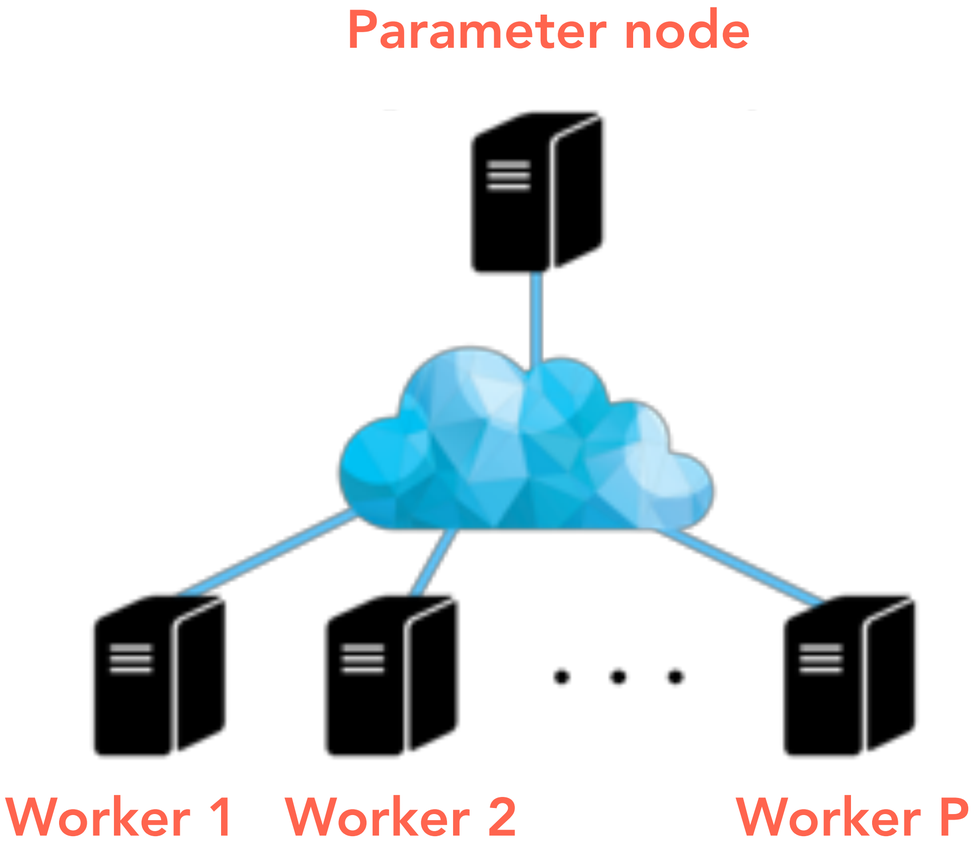
(when data cannot be easily shared due to privacy concerns or inefficiency)

Federated Learning: when data stay close to a client

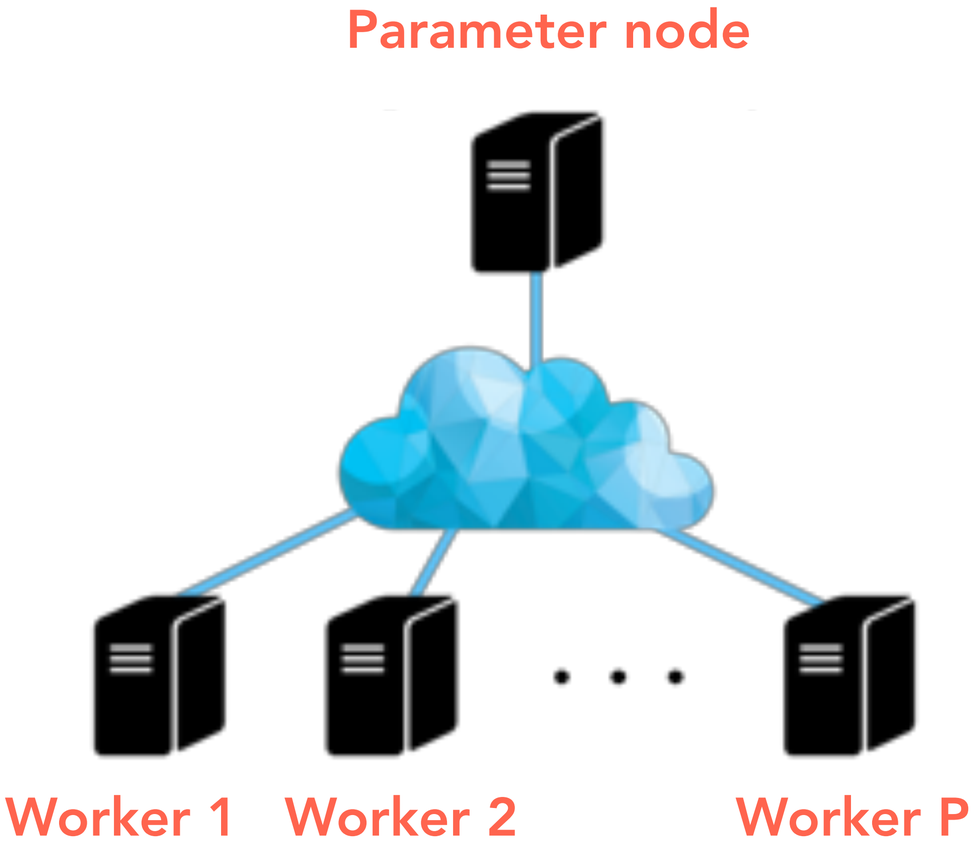


Federated Learning: when data stay close to a client

- Data reside at workers/users' sites

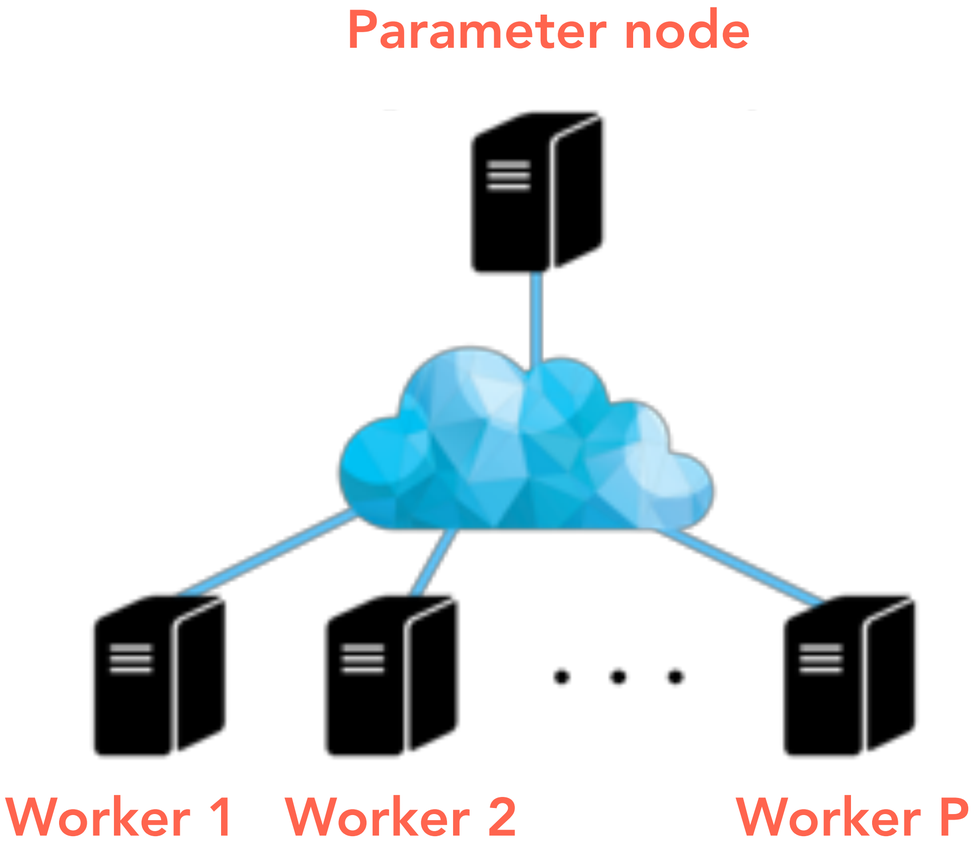


Federated Learning: when data stay close to a client



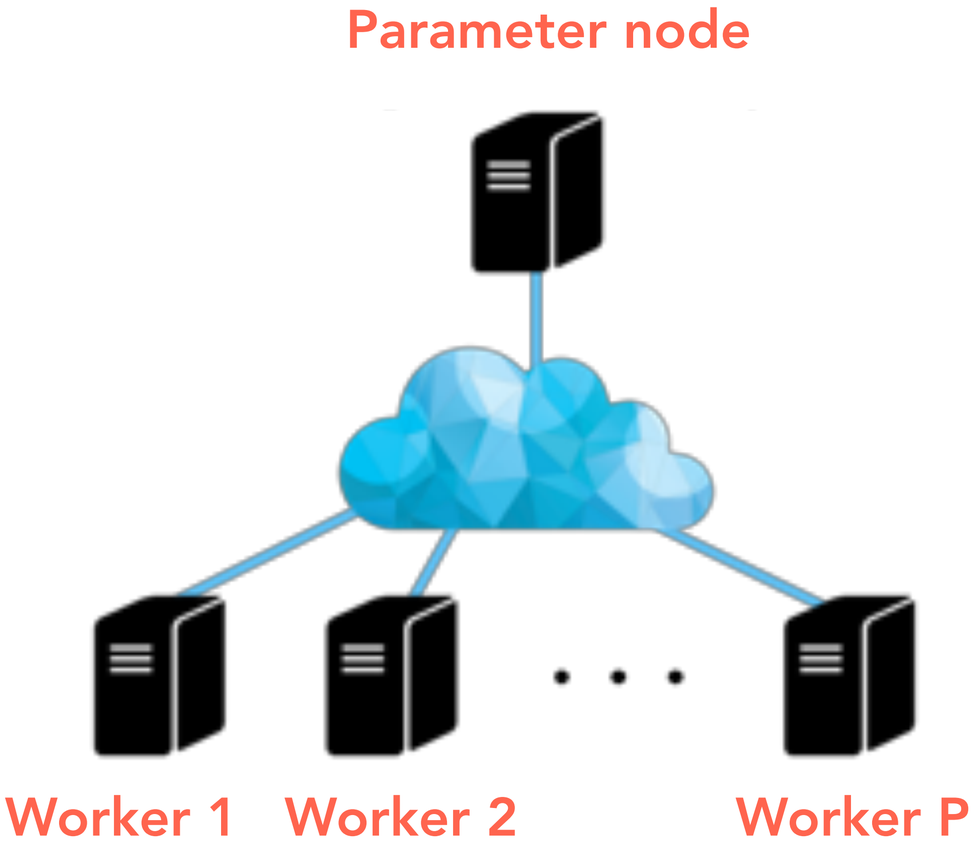
- Data reside at workers/users' sites
- Workers are not necessarily powerful devices (e.g., mobile devices, sensors, ..)

Federated Learning: when data stay close to a client



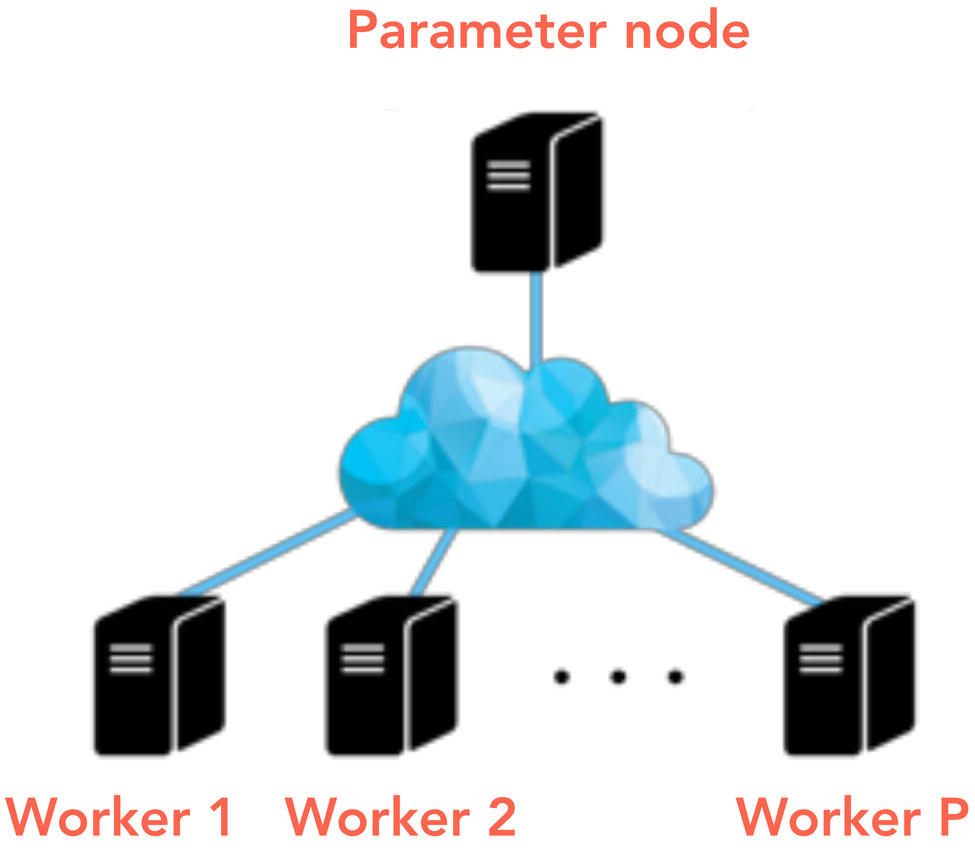
- Data reside at workers/users' sites
- Workers are not necessarily powerful devices (e.g., mobile devices, sensors, ..)
- Workers are not willing to share data with other workers or the server

Federated Learning: when data stay close to a client



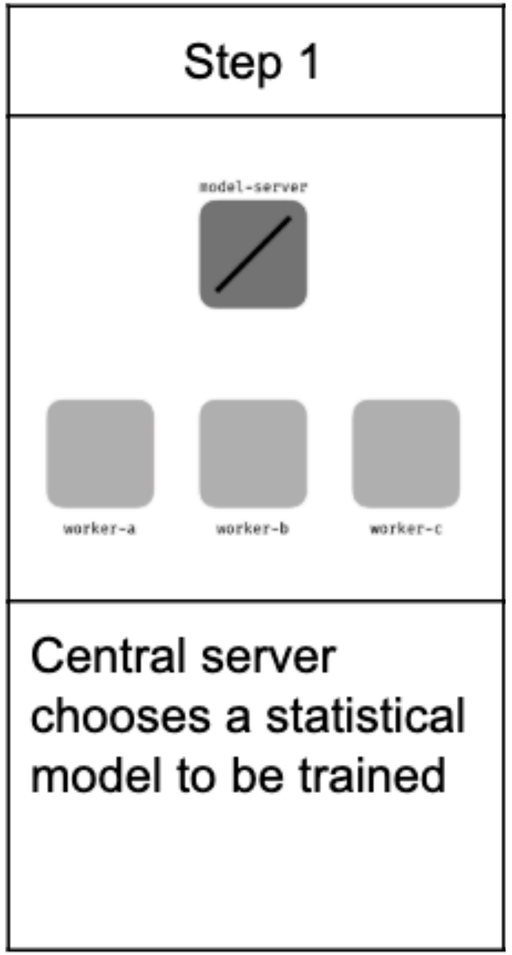
- Data reside at workers/users' sites
- Workers are not necessarily powerful devices (e.g., mobile devices, sensors, ..)
- Workers are not willing to share data with other workers or the server
- **Goal**: how can we train a global model (e.g., a neural network) in such a scenario?

Federated Learning: when data stay close to a client

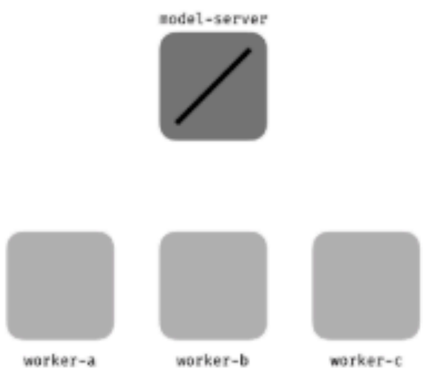
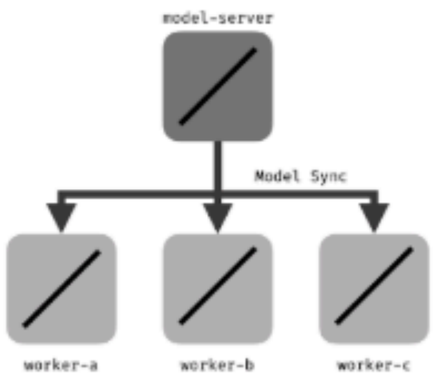


- Data reside at workers/users' sites
- Workers are not necessarily powerful devices (e.g., mobile devices, sensors, ..)
- Workers are not willing to share data with other workers or the server
- **Goal**: how can we train a global model (e.g., a neural network) in such a scenario?
- **Challenges**: Data could be heavily skewed; workers cannot do heavy computations; communication links are slow or not guaranteed; personalization, ..

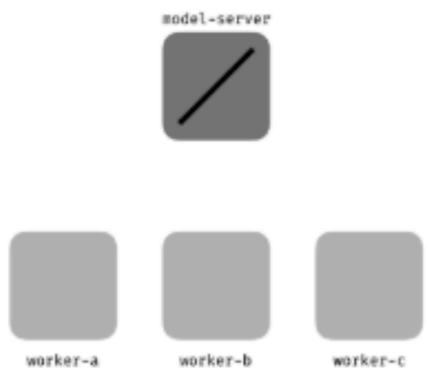
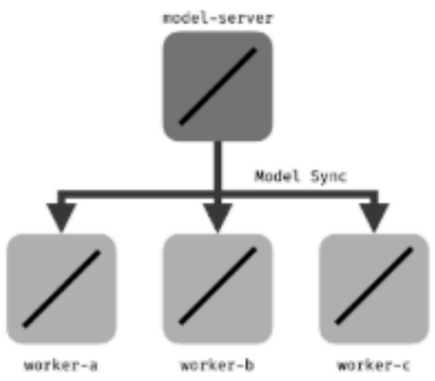
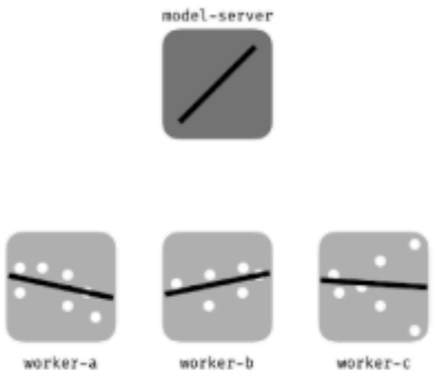
Federated Learning: when data stay close to a client



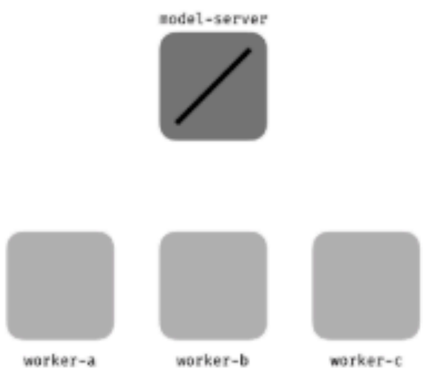
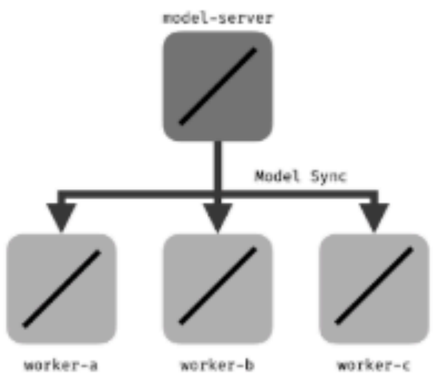
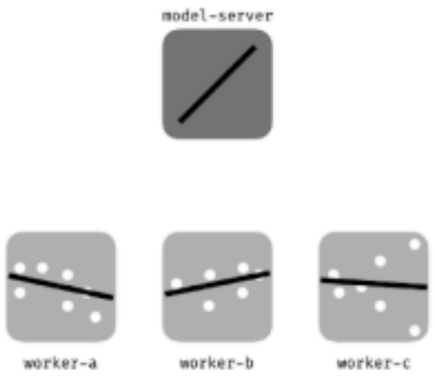
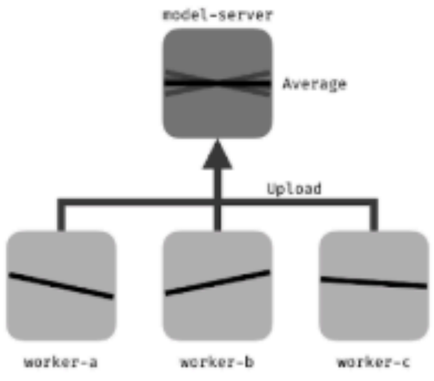
Federated Learning: when data stay close to a client

Step 1	Step 2
	
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes

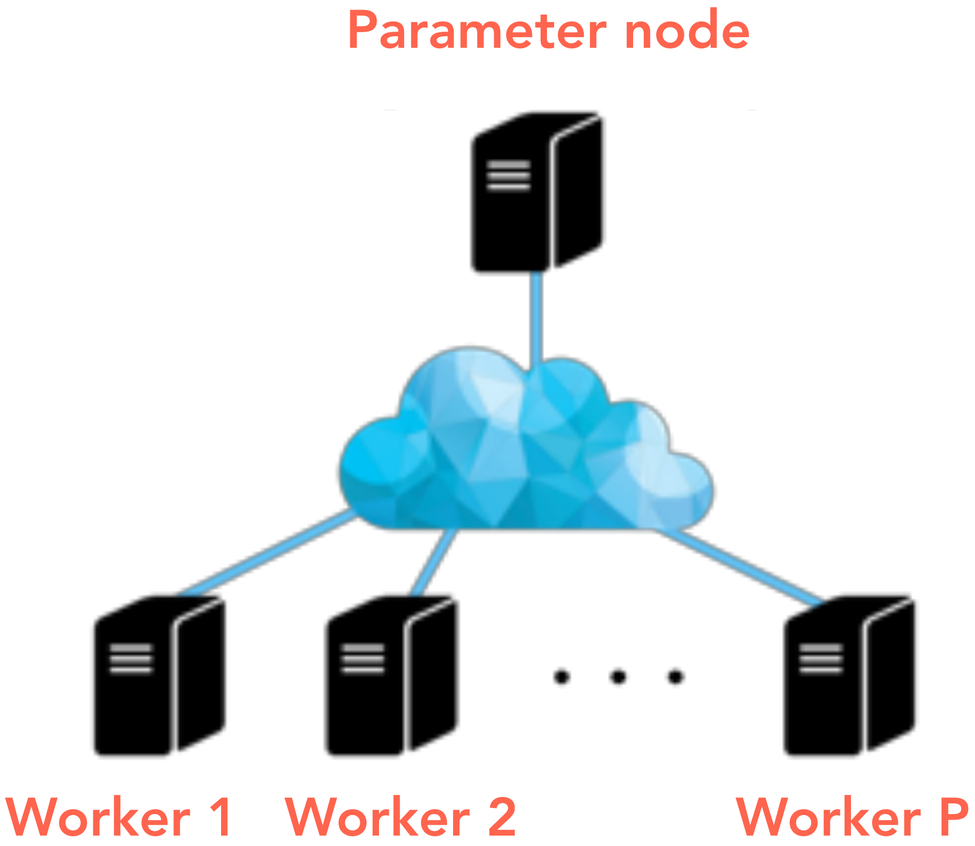
Federated Learning: when data stay close to a client

Step 1	Step 2	Step 3
		
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data

Federated Learning: when data stay close to a client

Step 1	Step 2	Step 3	Step 4
			
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data	Central server pools model results and generate one global mode without accessing any data

Federated Learning: when data stay close to a client

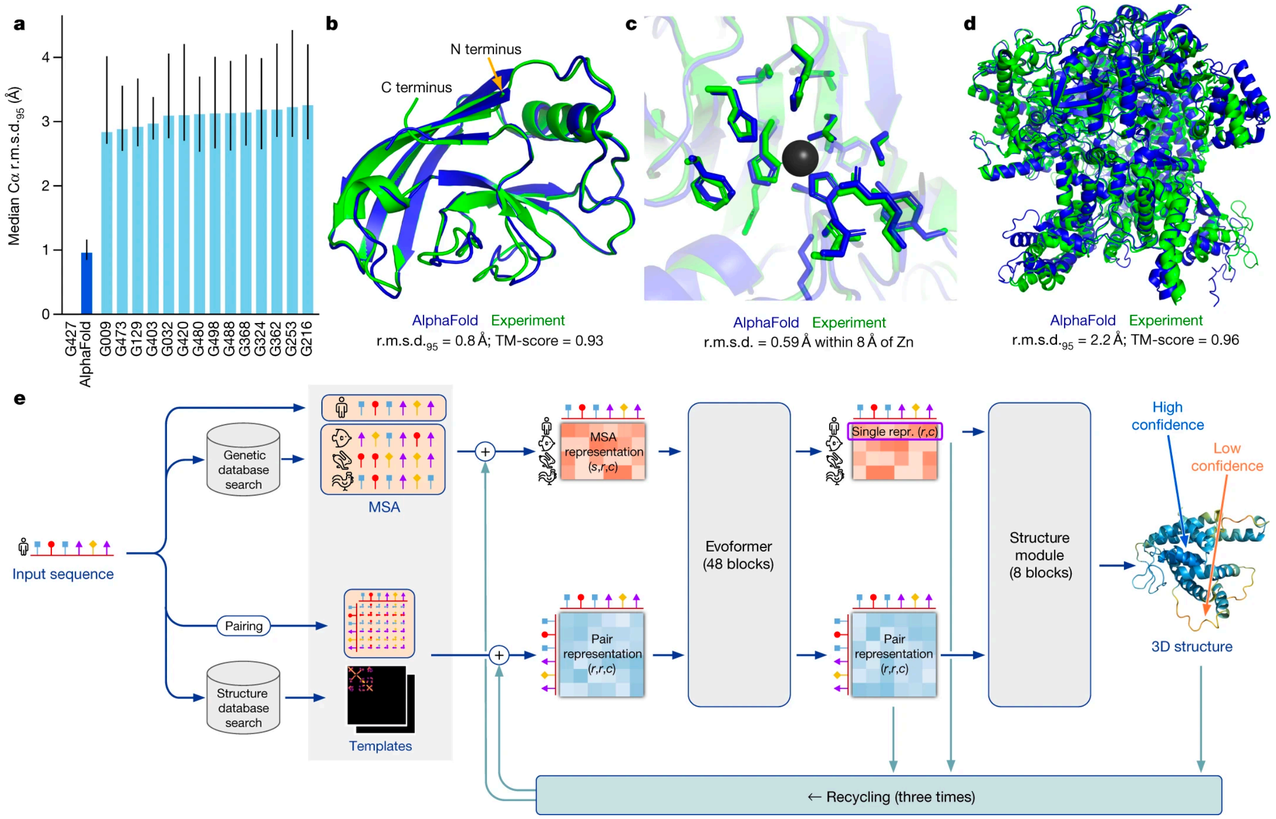


- Types of federated learning: cross-silo vs cross-device cases

- Horizontal FL vs Vertical FL

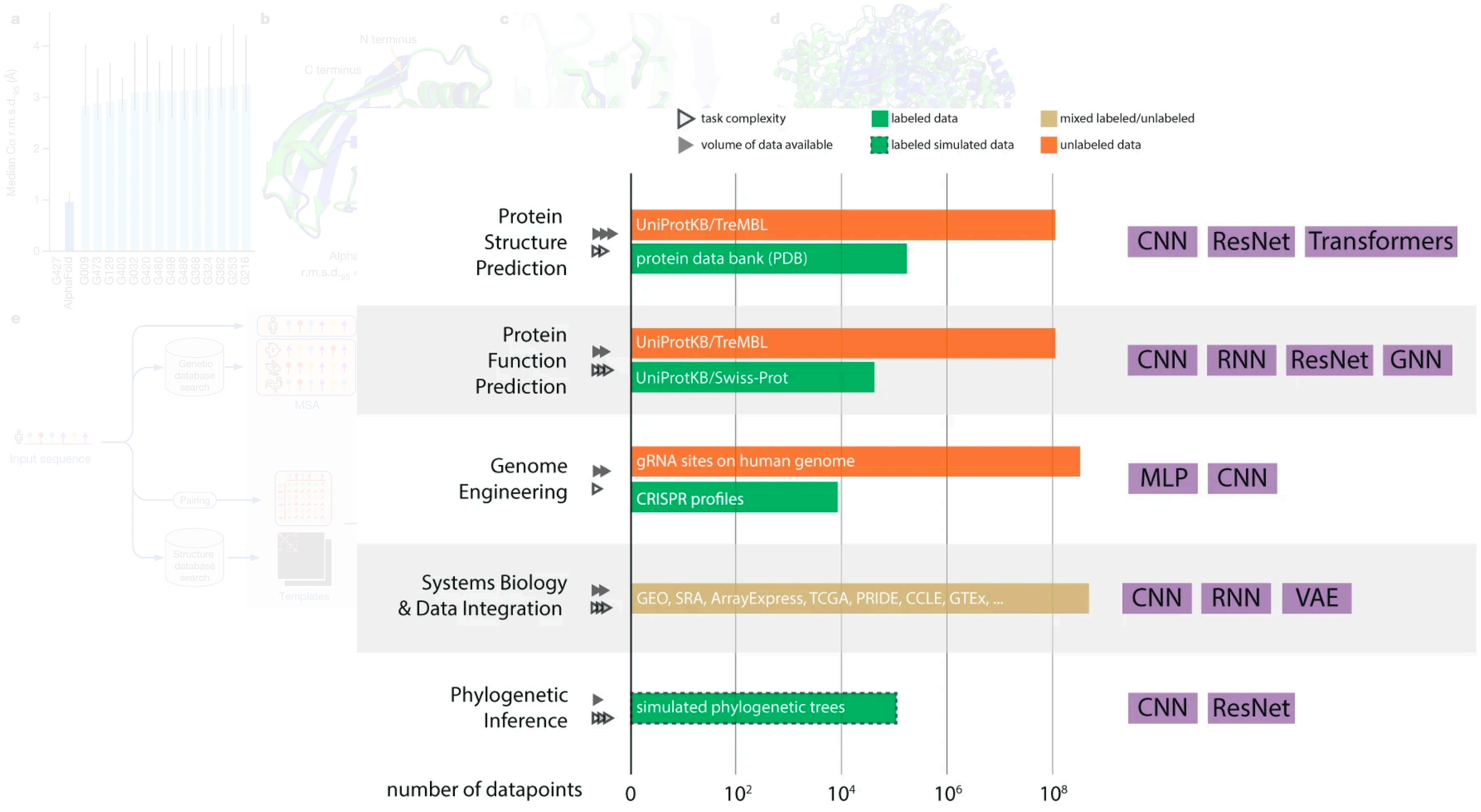
Recent popular successes of deep learning

Other recent successes of deep learning



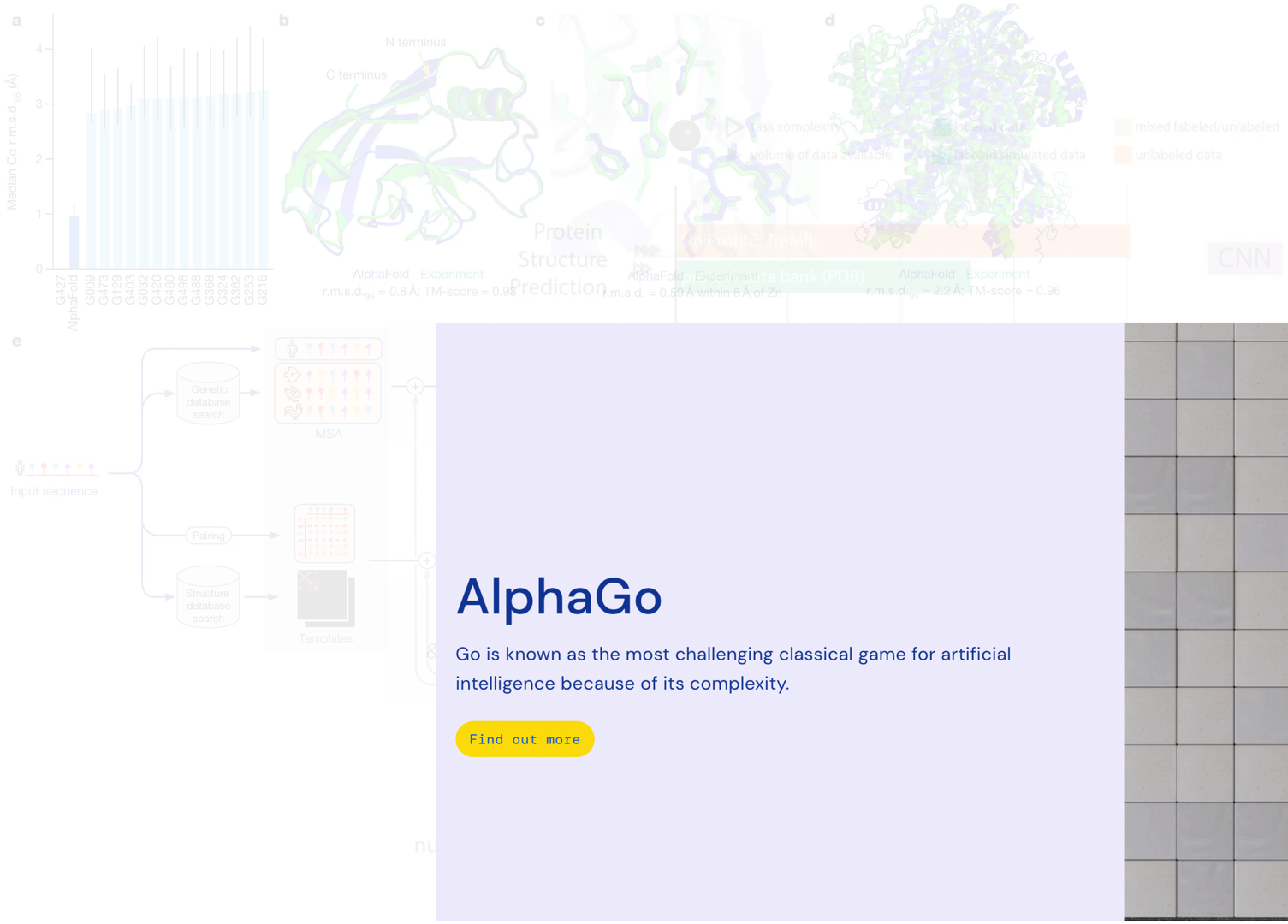
Other recent successes of deep learning

Other recent successes of deep learning



Other recent successes of deep learning

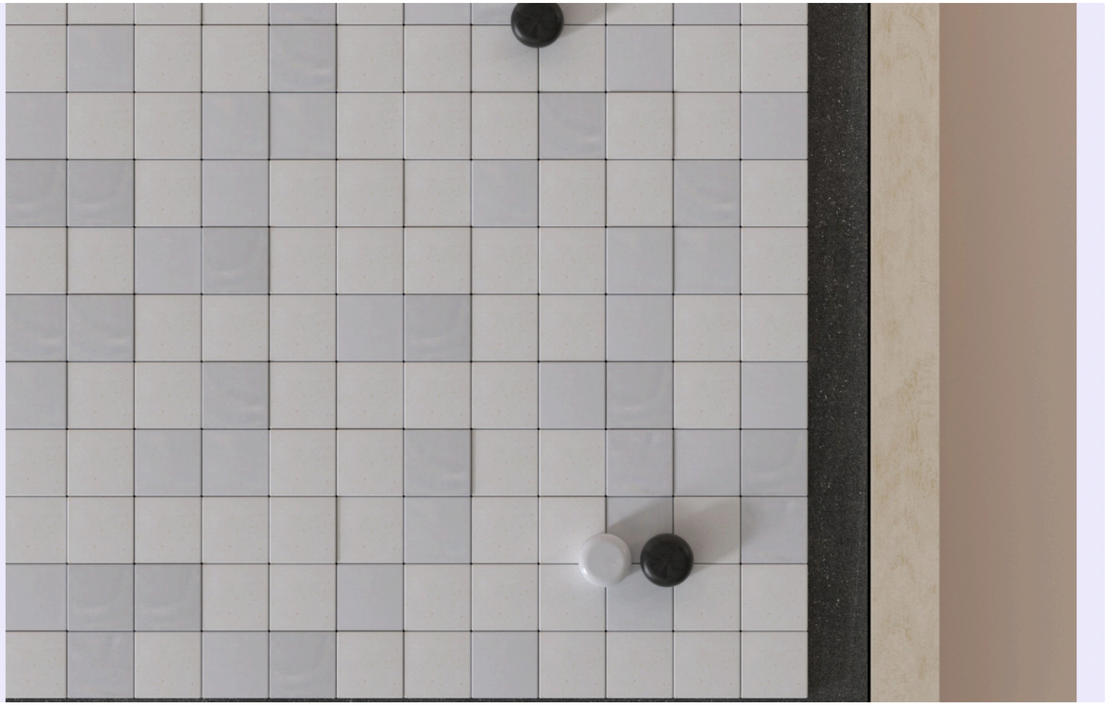
Other recent successes of deep learning



AlphaGo

Go is known as the most challenging classical game for artificial intelligence because of its complexity.

Find out more



Other recent successes of deep learning

Deep learning theory

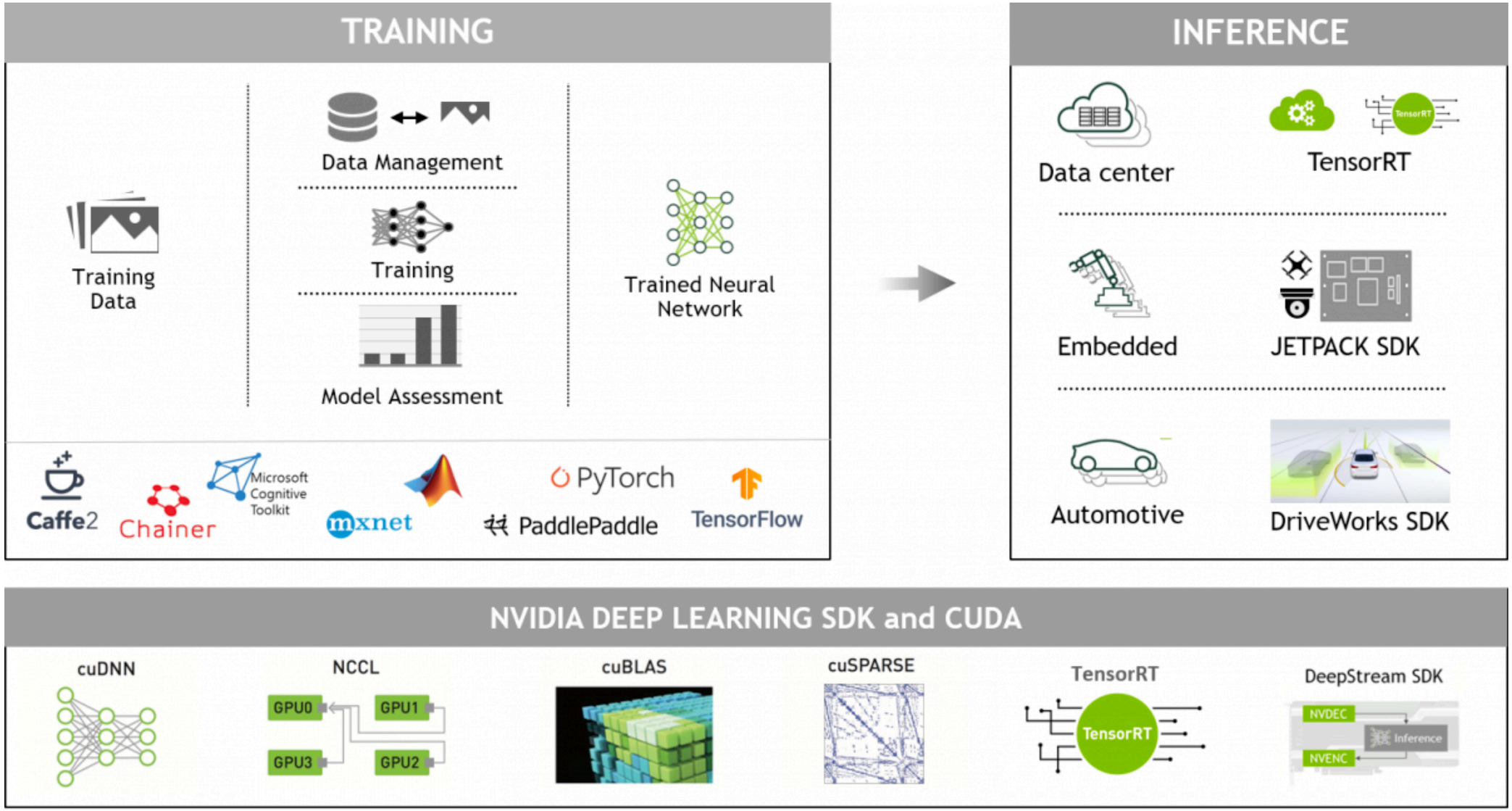
(In order to understand why, when and how such complex models work)

Summary

- Introduction to neural networks
- Feedforward / Fully-connected neural networks or MLPs
- Convolutional neural networks (CNNs)
- Recurrent neural networks (RNNs)
- Optimization in neural network training
- Some trends in deep learning
- What about software?

Software

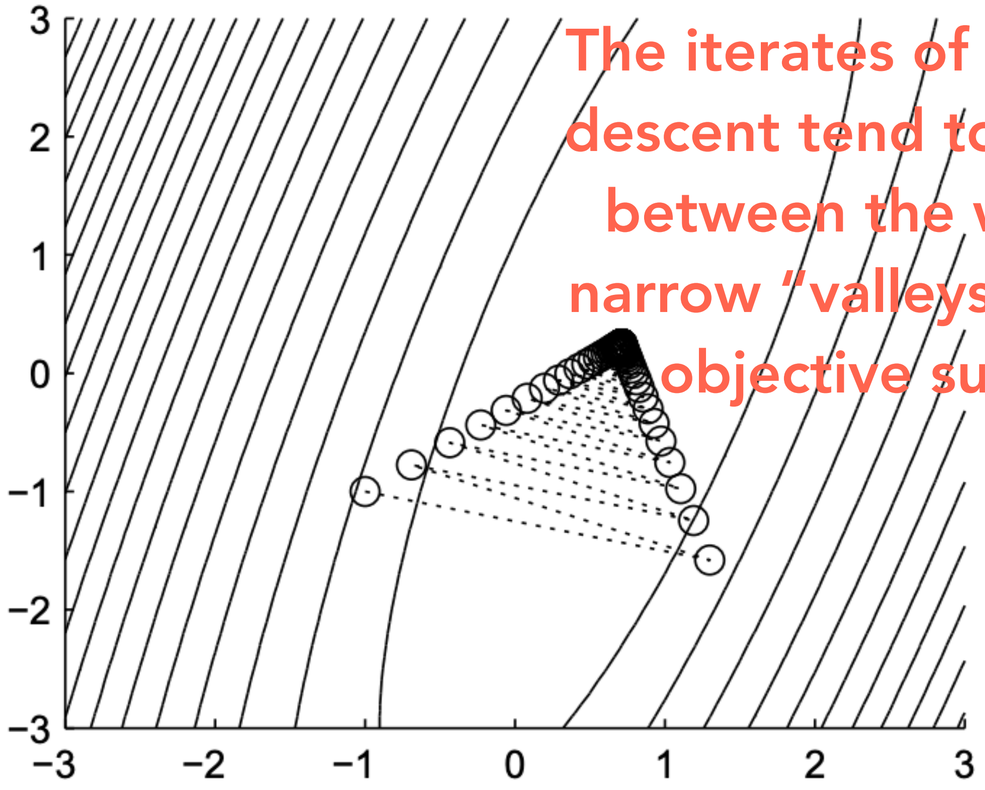
Source: NVidia



Variants of SGD

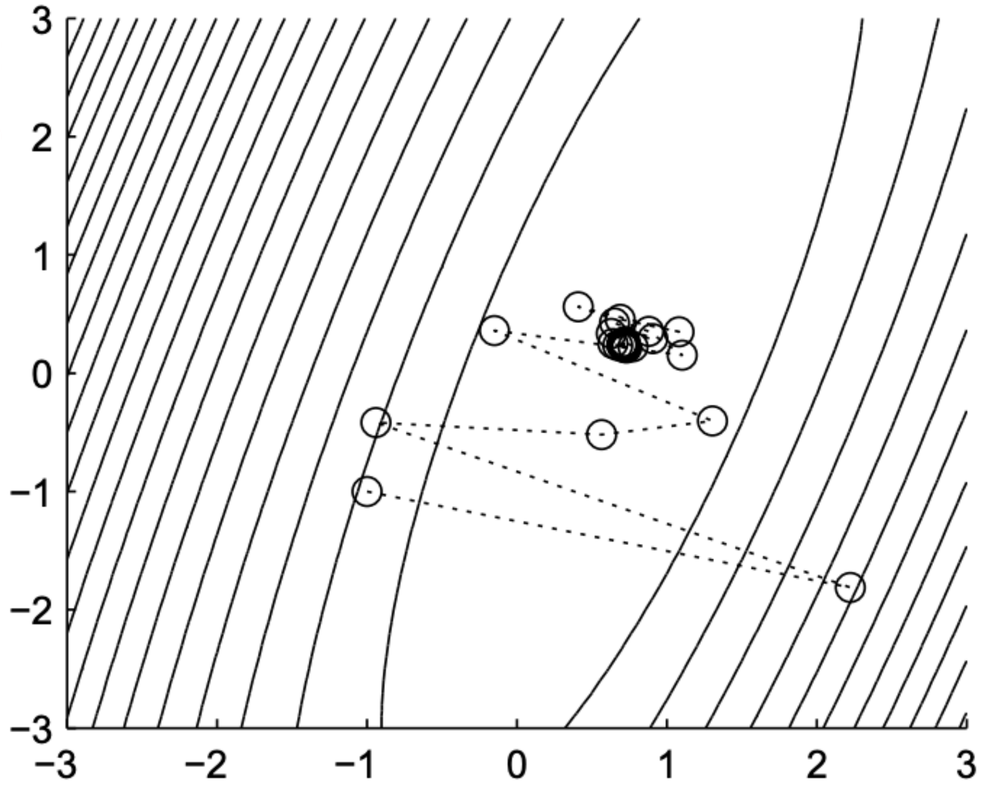
Momentum acceleration

- Heavy ball method



The iterates of gradient descent tend to bounce between the walls of narrow "valleys" on the objective surface

Gradient descent



Extrapolating previous directions

Momentum acceleration

Tribute to
Piotr Skalski

Momentum acceleration

Tribute to
Piotr Skalski

Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step

Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step

Momentum step

Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step

Momentum step

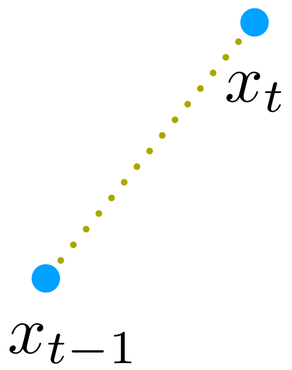
x_{t-1}

Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step Momentum step

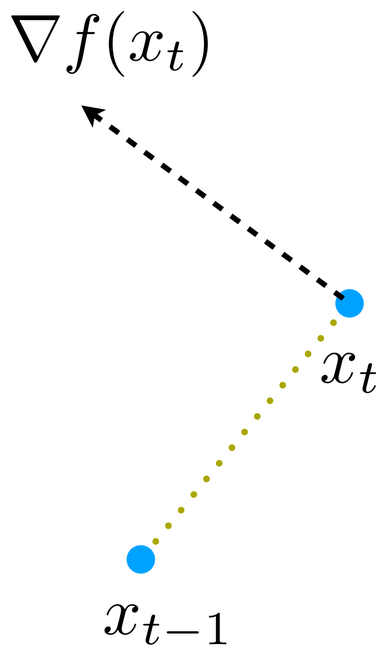


Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step Momentum step

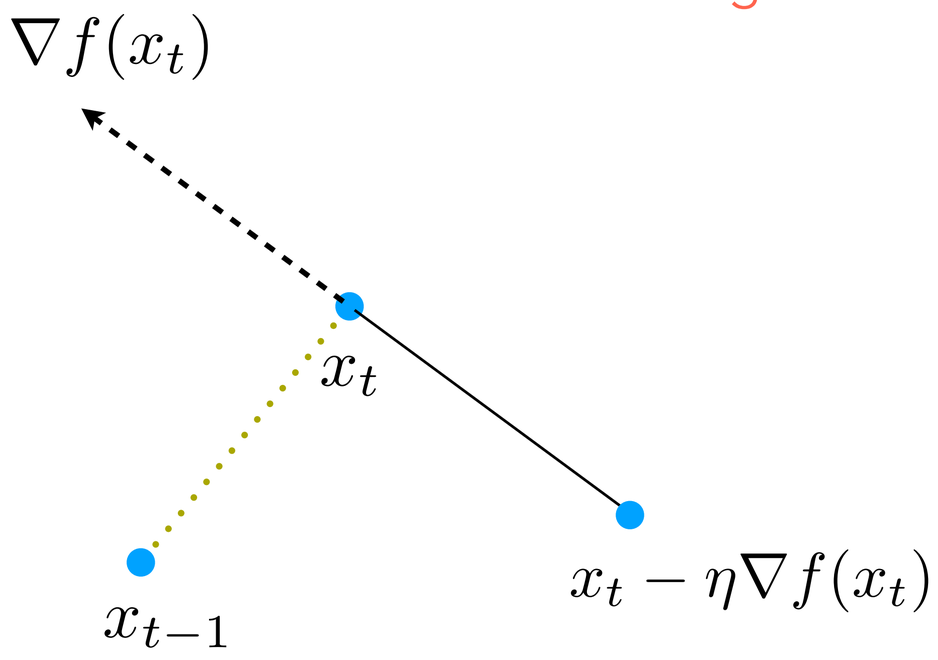


Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step Momentum step



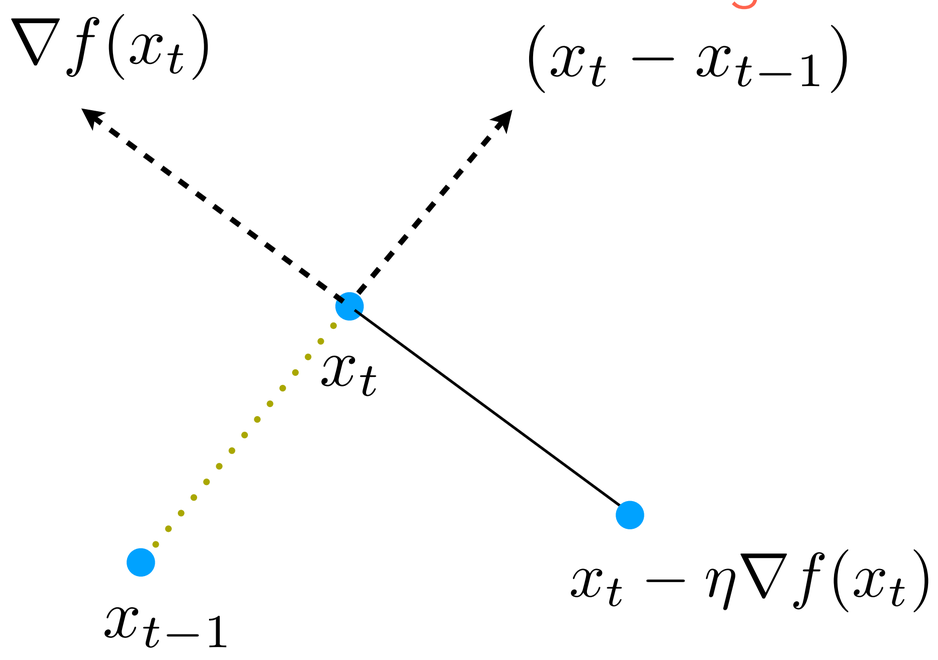
Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step

Momentum step

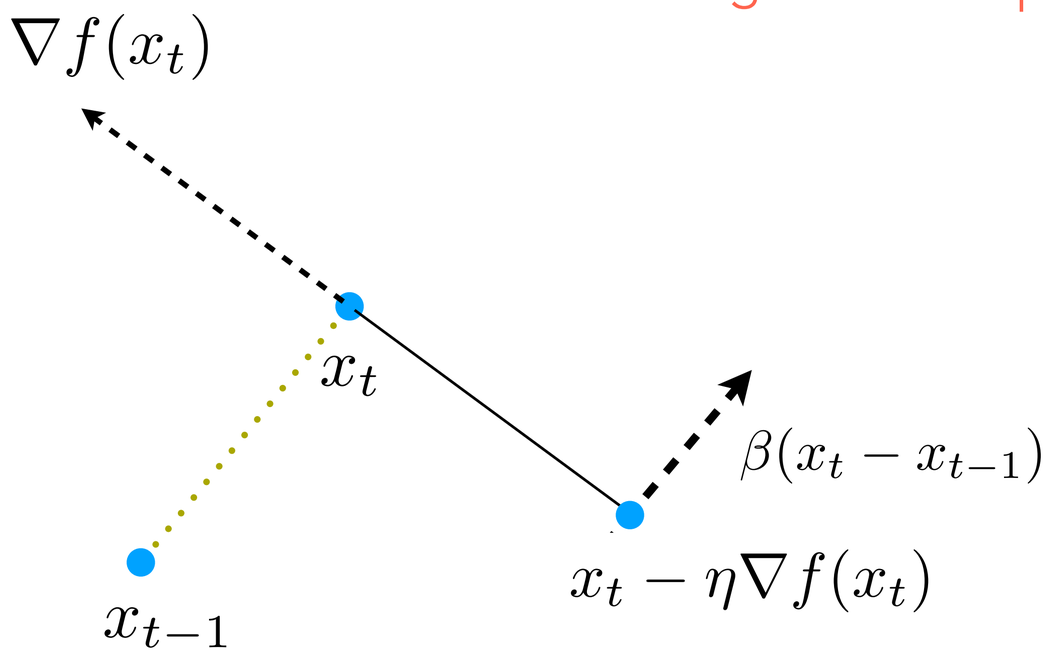


Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step Momentum step

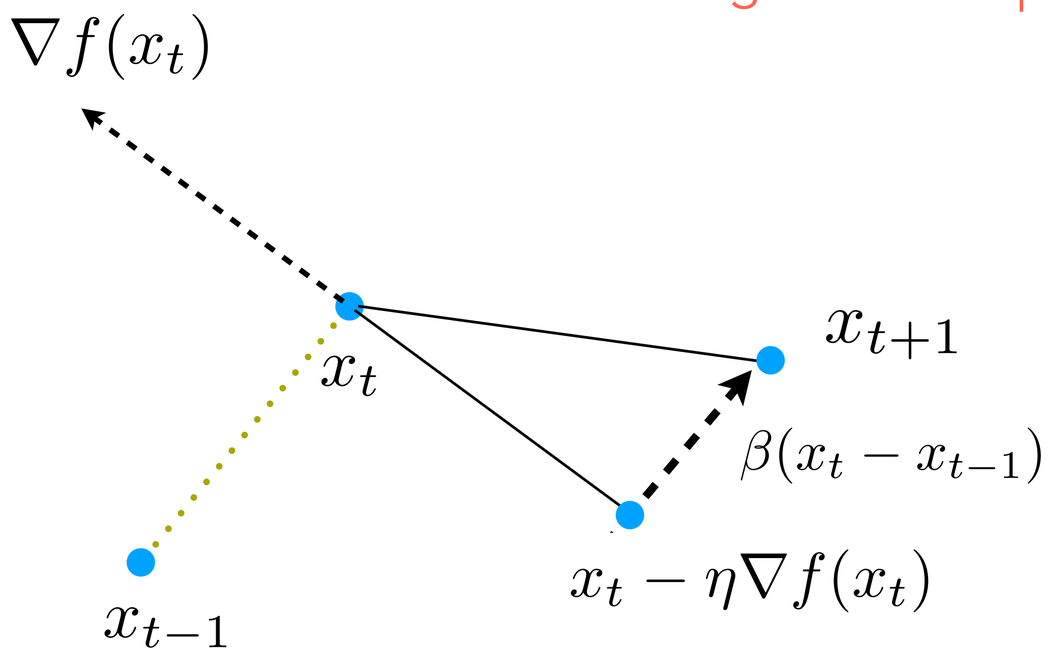


Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step Momentum step

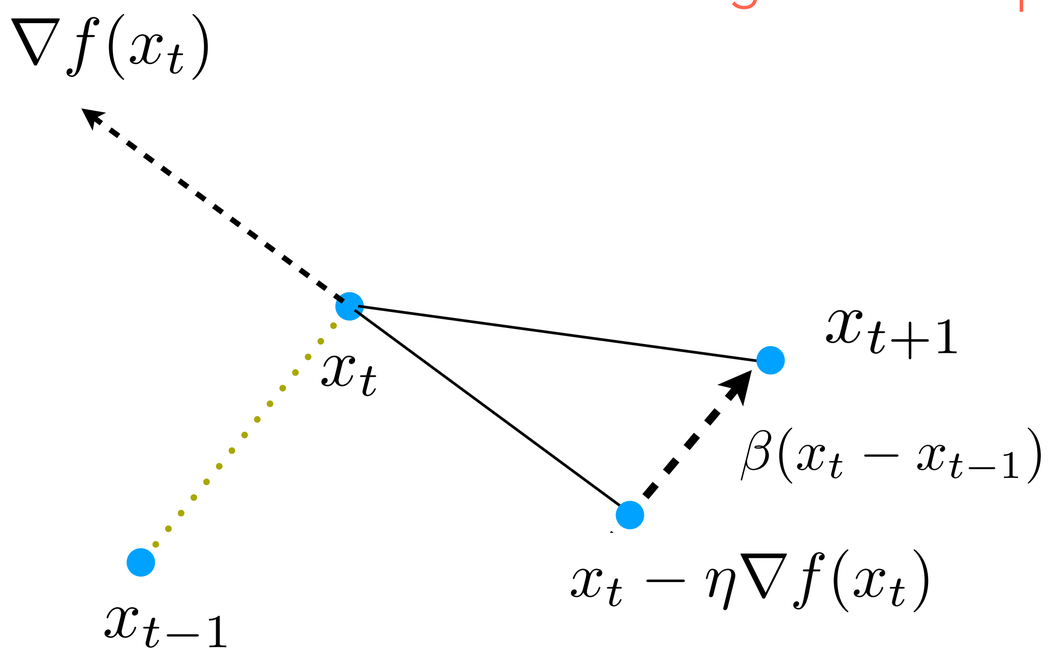


Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Standard gradient step Momentum step



- If current gradient step is in same direction as previous step, then move a little further in that direction

Momentum acceleration

- Nesterov's work

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

Momentum acceleration

- Nesterov's work

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

1. $\beta = \dfrac{\theta_t - 1}{\theta_{t+1}}$ where $\theta_0 = 1, \quad \theta_{t+1} = \dfrac{1 + \sqrt{1 + 4\theta_t^2}}{2}$

Momentum acceleration

- Nesterov's work

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

1. $\beta = \dfrac{\theta_t - 1}{\theta_{t+1}}$ where $\theta_0 = 1, \quad \theta_{t+1} = \dfrac{1 + \sqrt{1 + 4\theta_t^2}}{2}$

2. $\beta = \dfrac{t}{t+3}$

Momentum acceleration

- Nesterov's work

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

1. $\beta = \dfrac{\theta_t - 1}{\theta_{t+1}}$ where $\theta_0 = 1, \quad \theta_{t+1} = \dfrac{1 + \sqrt{1 + 4\theta_t^2}}{2}$

2. $\beta = \dfrac{t}{t+3}$

3. $\beta = 0.9$

Between gradient descent and Newton's method

- **Quasi-Newton methods**

$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$

Between gradient descent and Newton's method

- **Quasi-Newton methods**

$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$

Approximation of the inverse Hessian

Between gradient descent and Newton's method

- **Quasi-Newton methods**

$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$

Approximation of the inverse Hessian

-"Quasi-Newton" reveals that we want to avoid second-order calculations

Between gradient descent and Newton's method

- **Quasi-Newton methods**

$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$

Approximation of the inverse Hessian

-"Quasi-Newton" reveals that we want to avoid second-order calculations

- There are various ways to construct this approximation
  - **(L)-BFGS approximation**
  - **SR1 approximation**

AdaGrad algorithm

(A Google algorithm that found application to ``Large-scale distributed deep networks`` paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

AdaGrad algorithm

(A Google algorithm that found application to ``Large-scale distributed deep networks`` paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

- AdaGrad adapts the initial step size for each of the components:
    - Associates small step sizes to frequently occurring features
    - Associates large step sizes to rate occurring features

AdaGrad algorithm

(A Google algorithm that found application to ``Large-scale distributed deep networks`` paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

- AdaGrad adapts the initial step size for each of the components:
    - Associates small step sizes to frequently occurring features
    - Associates large step sizes to rate occurring features

- What is the main idea? Consider $x_{t+1,i} = x_{t,i} - \eta \nabla f(x_t)_i$

Entrywise representation of GD

AdaGrad algorithm

(A Google algorithm that found application to ``Large-scale distributed deep networks`` paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

- AdaGrad adapts the initial step size for each of the components:
  - Associates small step sizes to frequently occurring features
  - Associates large step sizes to rate occurring features

- What is the main idea? Consider $x_{t+1,i} = x_{t,i} - \eta \nabla f(x_t)_i$

Entrywise representation of GD

Then, practical version of AdaGrad does: $x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{B_{t,ii} + \epsilon}} \cdot \nabla f_{i_t}(x_t)_i$

AdaGrad algorithm

(A Google algorithm that found application to ``Large-scale distributed deep networks`` paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

- AdaGrad adapts the initial step size for each of the components:
  - Associates small step sizes to frequently occurring features
  - Associates large step sizes to rate occurring features

- What is the main idea? Consider $x_{t+1,i} = x_{t,i} - \eta \nabla f(x_t)_i$

Entrywise representation of GD

Then, practical version of AdaGrad does: $x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{\boxed{B_{t,ii}} + \epsilon}} \cdot \nabla f_{i_t}(x_t)_i$

What is this quantity?

AdaGrad algorithm

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

$$x_{t+1} = x_t - \eta B_t^{-1} \nabla f(x_t)$$

Recall: Preconditioning algorithms (BFGS, SR1)

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

$$x_{t+1} = x_t - \eta B_t^{-1} \nabla f(x_t)$$

Recall: Preconditioning algorithms (BFGS, SR1)

where

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

``Square root of the sum of gradient outer products, till current iteration``

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

$$x_{t+1} = x_t - \eta B_t^{-1} \nabla f(x_t)$$

Recall: Preconditioning algorithms (BFGS, SR1)

where

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

``Square root of the sum of gradient outer products, till current iteration``

Full matrix AdaGrad

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

$$x_{t+1} = x_t - \eta B_t^{-1} \nabla f(x_t)$$

Recall: Preconditioning algorithms (BFGS, SR1)

where

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

``Square root of the sum of gradient outer products, till current iteration``

Full matrix AdaGrad

- Compare this to the simpler (and practical version)

$$x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{B_{t,ii} + \epsilon}} \cdot \nabla f_{i_t}(x_t)_i$$

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

$$x_{t+1} = x_t - \eta B_t^{-1} \nabla f(x_t)$$

Recall: Preconditioning algorithms (BFGS, SR1)

where

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

``Square root of the sum of gradient outer products, till current iteration``

Full matrix AdaGrad

- Compare this to the simpler (and practical version)

$$x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{B_{t,ii} + \epsilon}} \cdot \nabla f_{i_t}(x_t)_i$$

Avoids division with zero

AdaGrad algorithm

- ``What is the intuition behind the form of $B_t$?``

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

Relates to the **Fisher Information matrix** (which is related to the expected Hessian) - outside our scope

AdaGrad algorithm

- ``What is the intuition behind the form of $B_t$?``

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

Relates to the **Fisher Information matrix** (which is related to the expected Hessian) - outside our scope

- ``What are some properties of AdaGrad?``
  1. Step size is automatically set - default values for initial step size is $\eta = 0.01$
  2. The original version keeps accumulating squared gradients, which makes resulting step sizes really small.

AdaGrad algorithm

- ``What is the intuition behind the form of $B_t$?``

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

Relates to the **Fisher Information matrix** (which is related to the expected Hessian) - outside our scope

- ``What are some properties of AdaGrad?``
  1. Step size is automatically set - default values for initial step size is $\eta = 0.01$
  2. The original version keeps accumulating squared gradients, which makes resulting step sizes really small.

- ``Are there guarantees for AdaGrad?``
- Yes, in the convex case, using regret bounds - see Literature section

AdaGrad pseudocode

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

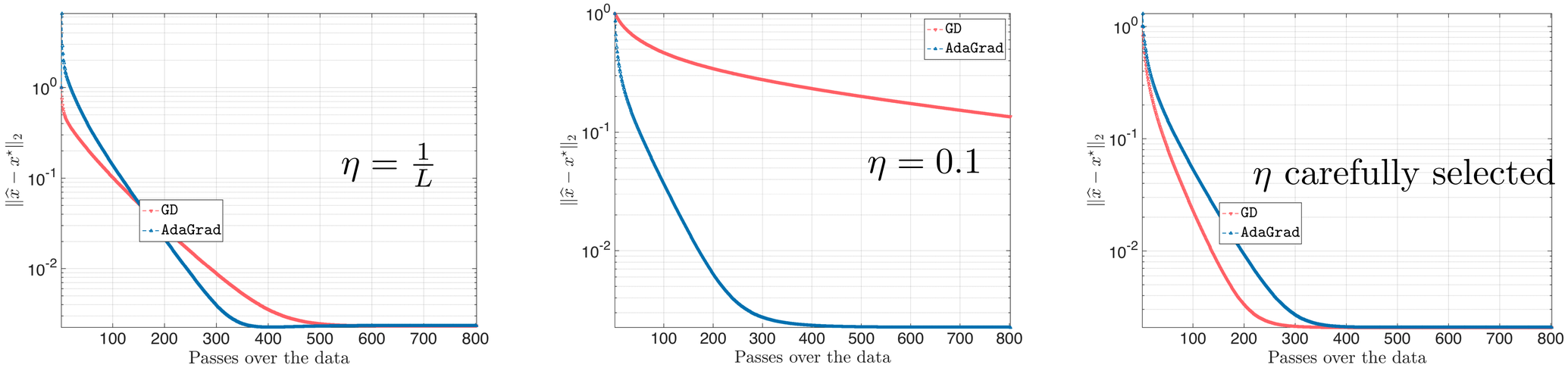
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

AdaGrad in practice

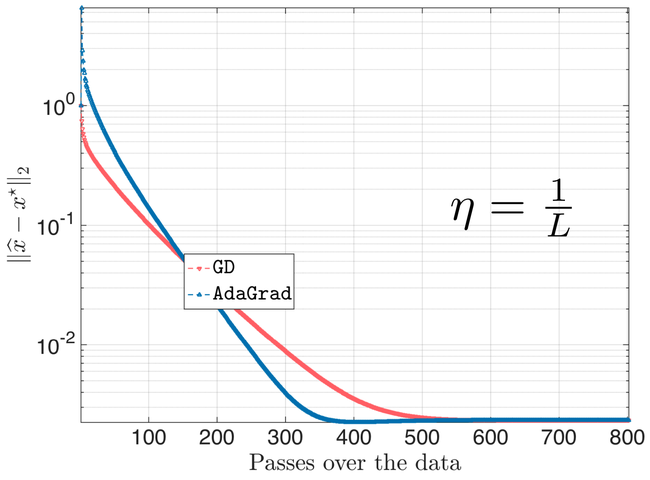
AdaGrad in practice

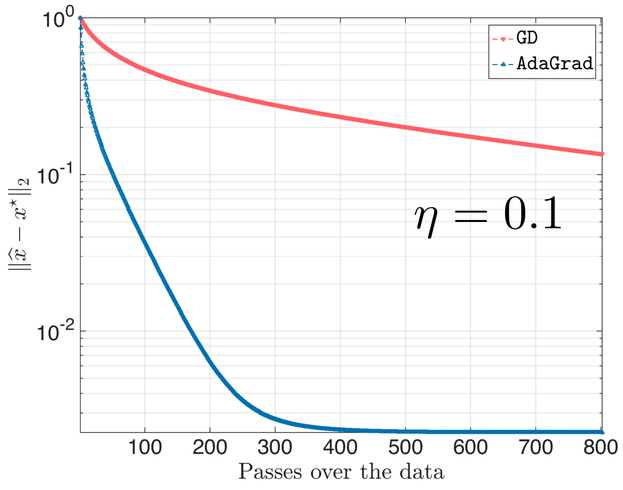
Well-conditioned linear regression

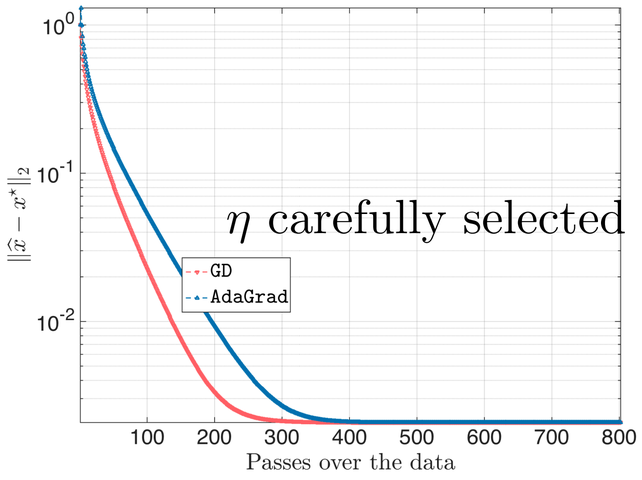


AdaGrad in practice

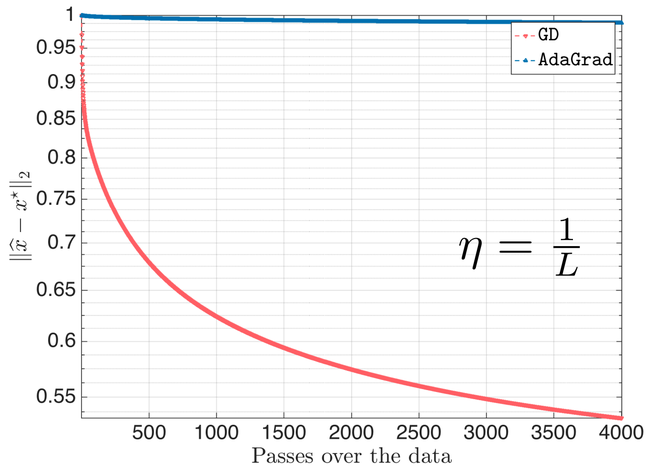
Well-conditioned linear regression

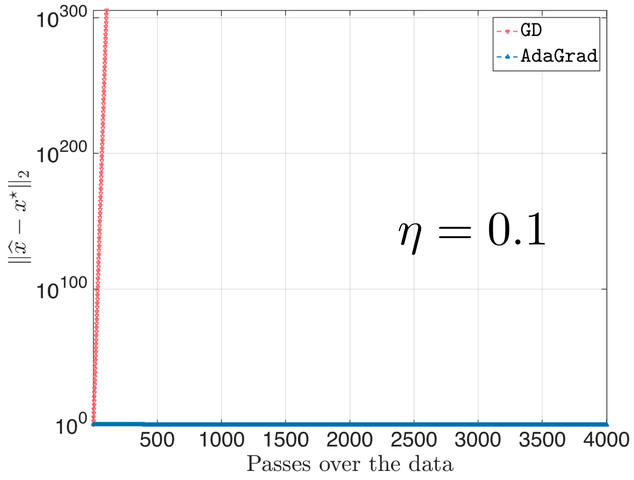


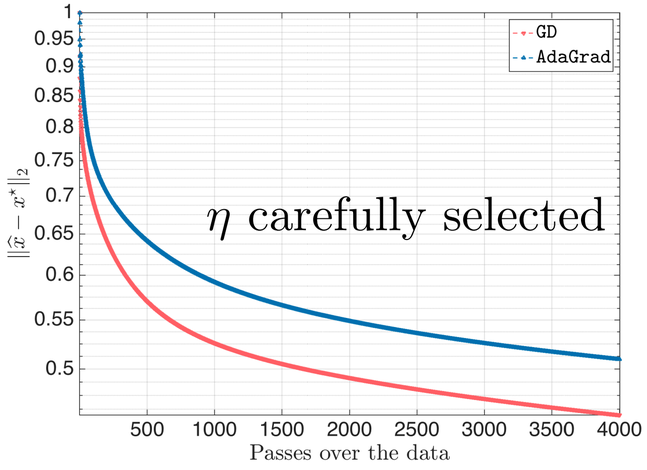




Ill-conditioned linear regression



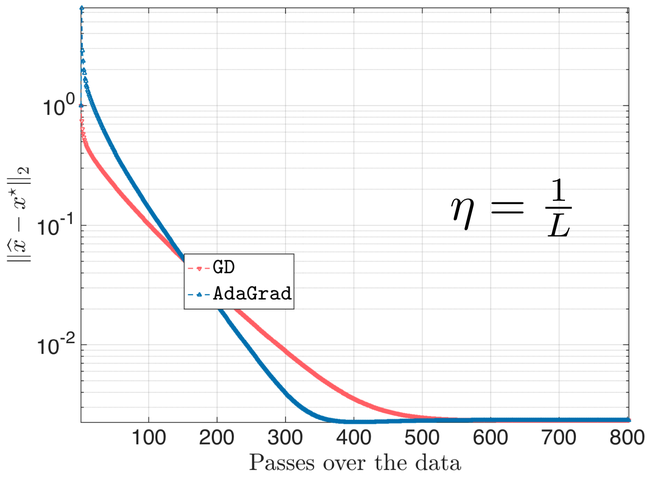


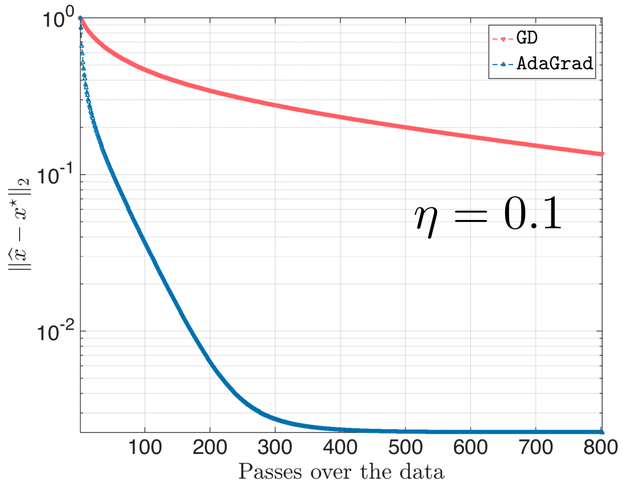


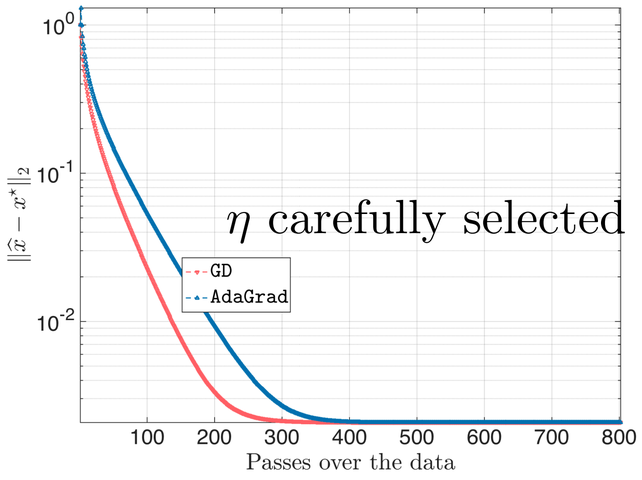
AdaGrad in practice

(Similar performance in logistic regression)

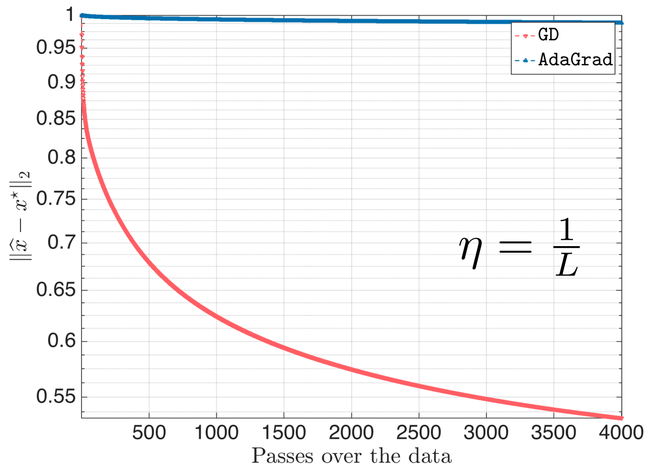
Well-conditioned linear regression

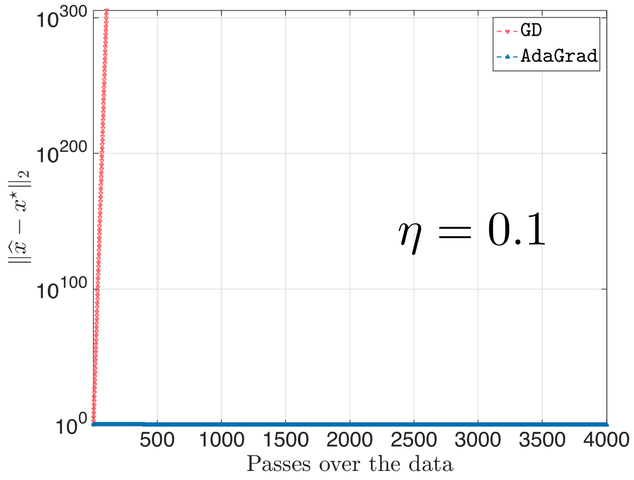


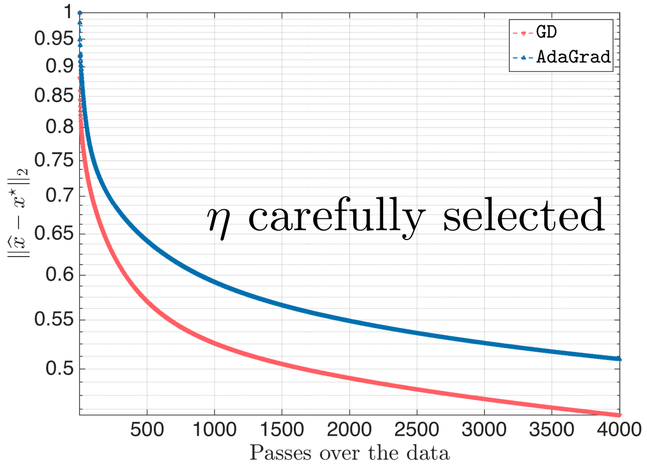




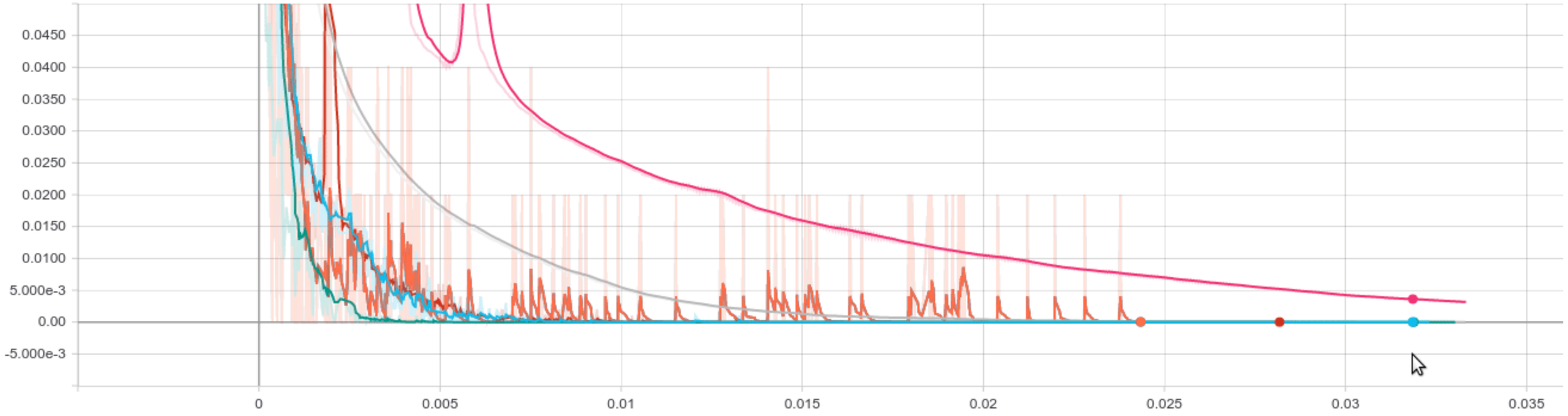
Ill-conditioned linear regression





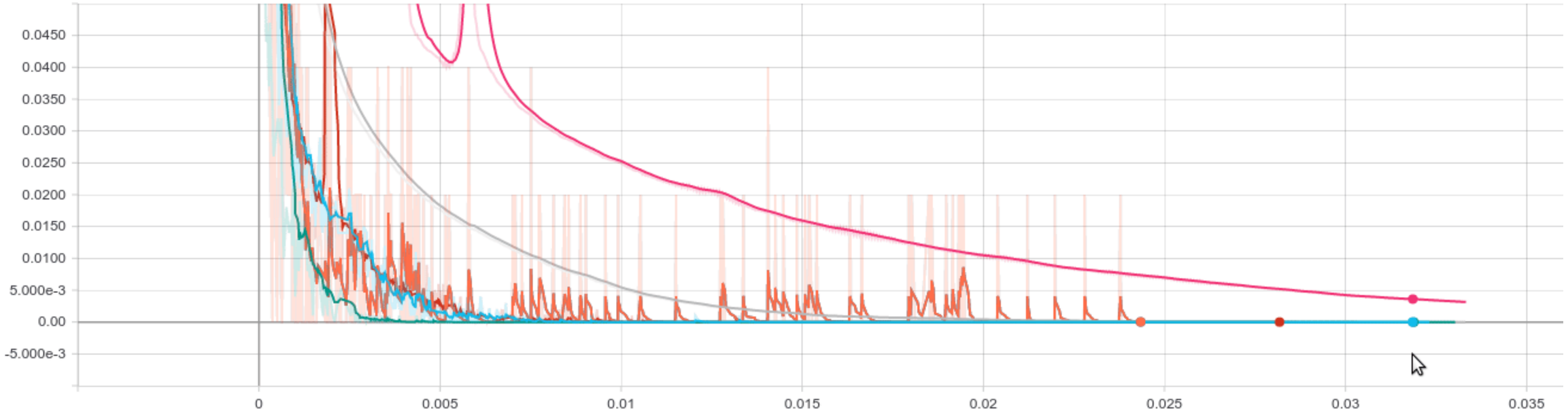


AdaGrad in practice



Name	Smoothed	Value	Step	Time	Relative
AdaGrad2	-1.1921e-7	-1.1921e-7	5.381k	Sun Dec 10, 15:17:05	1m 54s
AdaGrad3	3.6024e-3	3.5634e-3	764.0	Sun Dec 10, 15:19:02	1m 54s
Nesterov2	-1.1921e-7	-1.1921e-7	5.303k	Sun Dec 10, 15:28:30	1m 54s
Nesterov3	-2.3665e-7	-2.3842e-7	764.0	Sun Dec 10, 15:30:29	1m 54s
SGD1	2.1986e-5	0.000	5.486k	Sun Dec 10, 15:01:31	1m 27s
SGD1/.	2.1986e-5	0.000	5.486k	Sun Dec 10, 15:01:31	1m 27s
SGD2	-1.1921e-7	-1.1921e-7	5.486k	Sun Dec 10, 15:03:13	1m 41s

AdaGrad in practice



Name	Smoothed	Value	Step	Time	Relative
AdaGrad2	-1.1921e-7	-1.1921e-7	5.381k	Sun Dec 10, 15:17:05	1m 54s
AdaGrad3	3.6024e-3	3.5634e-3	764.0	Sun Dec 10, 15:19:02	1m 54s
Nesterov2	-1.1921e-7	-1.1921e-7	5.303k	Sun Dec 10, 15:28:30	1m 54s
Nesterov3	-2.3665e-7	-2.3842e-7	764.0	Sun Dec 10, 15:30:29	1m 54s
SGD1	2.1986e-5	0.000	5.486k	Sun Dec 10, 15:01:31	1m 27s
SGD1/.	2.1986e-5	0.000	5.486k	Sun Dec 10, 15:01:31	1m 27s
SGD2	-1.1921e-7	-1.1921e-7	5.486k	Sun Dec 10, 15:03:13	1m 41s

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

  + Diagonal AdaGrad rule: $\mathrm{diag}(B_t) = \mathrm{diag}(B_{t-1}) + \mathrm{diag}\left(\nabla f_{i_t}(x_t) \circ \nabla f_{i_t}(x_t)\right)$

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

+ Diagonal AdaGrad rule: $\mathrm{diag}(B_t) = \mathrm{diag}(B_{t-1}) + \mathrm{diag}\left(\nabla f_{i_t}(x_t) \circ \nabla f_{i_t}(x_t)\right)$

$E[g^2]_t \qquad E[g^2]_{t-1} \qquad g_t^2$

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

+ Diagonal AdaGrad rule: $\mathrm{diag}(B_t) = \mathrm{diag}(B_{t-1}) + \mathrm{diag}\left(\nabla f_{i_t}(x_t) \circ \nabla f_{i_t}(x_t)\right)$

$E[g^2]_t$ $E[g^2]_{t-1}$ g_t^2

+ RMSprop rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

``We always give weight 0.1 to the new information``

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

  + Diagonal AdaGrad rule: $\mathrm{diag}(B_t) = \mathrm{diag}(B_{t-1}) + \mathrm{diag}\left(\nabla f_{i_t}(x_t) \circ \nabla f_{i_t}(x_t)\right)$

$E[g^2]_t$ $E[g^2]_{t-1}$ g_t^2

  + RMSprop rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

``We always give weight 0.1 to the new information``

- Algorithm:

$$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

$$x_{t+1} = x_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla f_{i_t}(x_t)$$

Introducing exponentially weighted averages

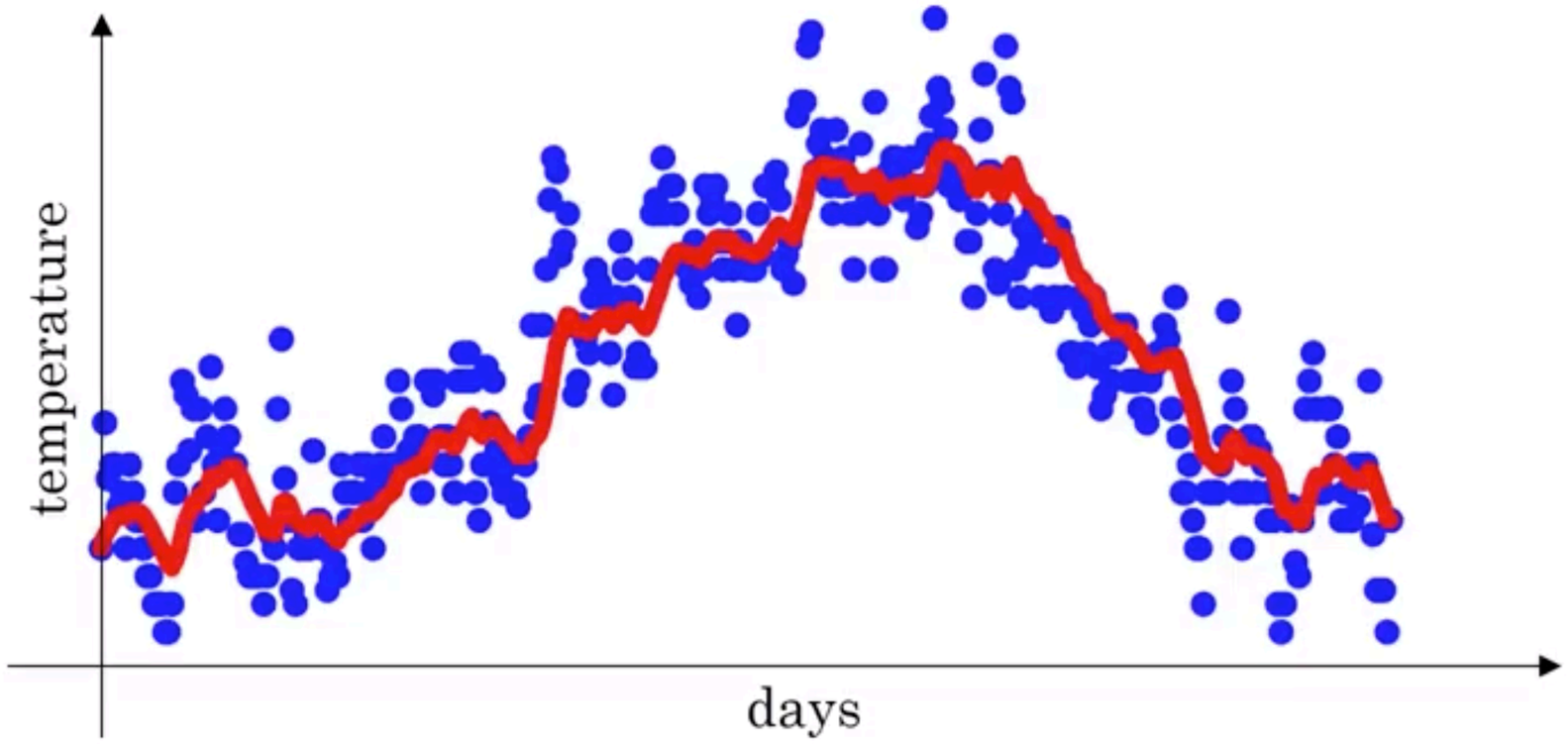
(Adapted from Ng's lectures)

- Toy example: temperature values over a year

Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year

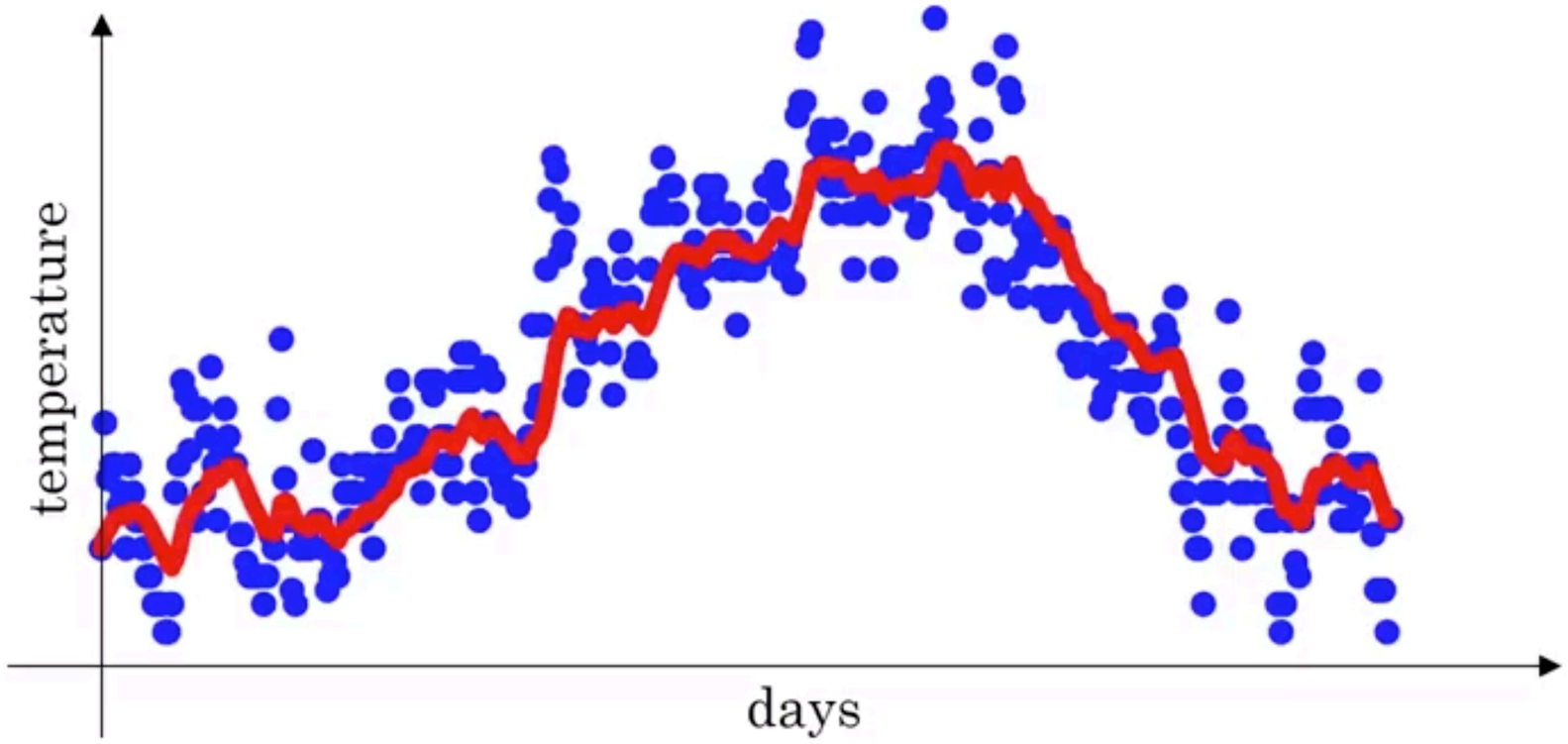


Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year

- Computing trends: local averages and how they evolve



$$V_0 = 0$$

$$V_1 = 0.9V_0 + 0.1\theta_1$$

$$V_2 = 0.9V_1 + 0.1\theta_2$$

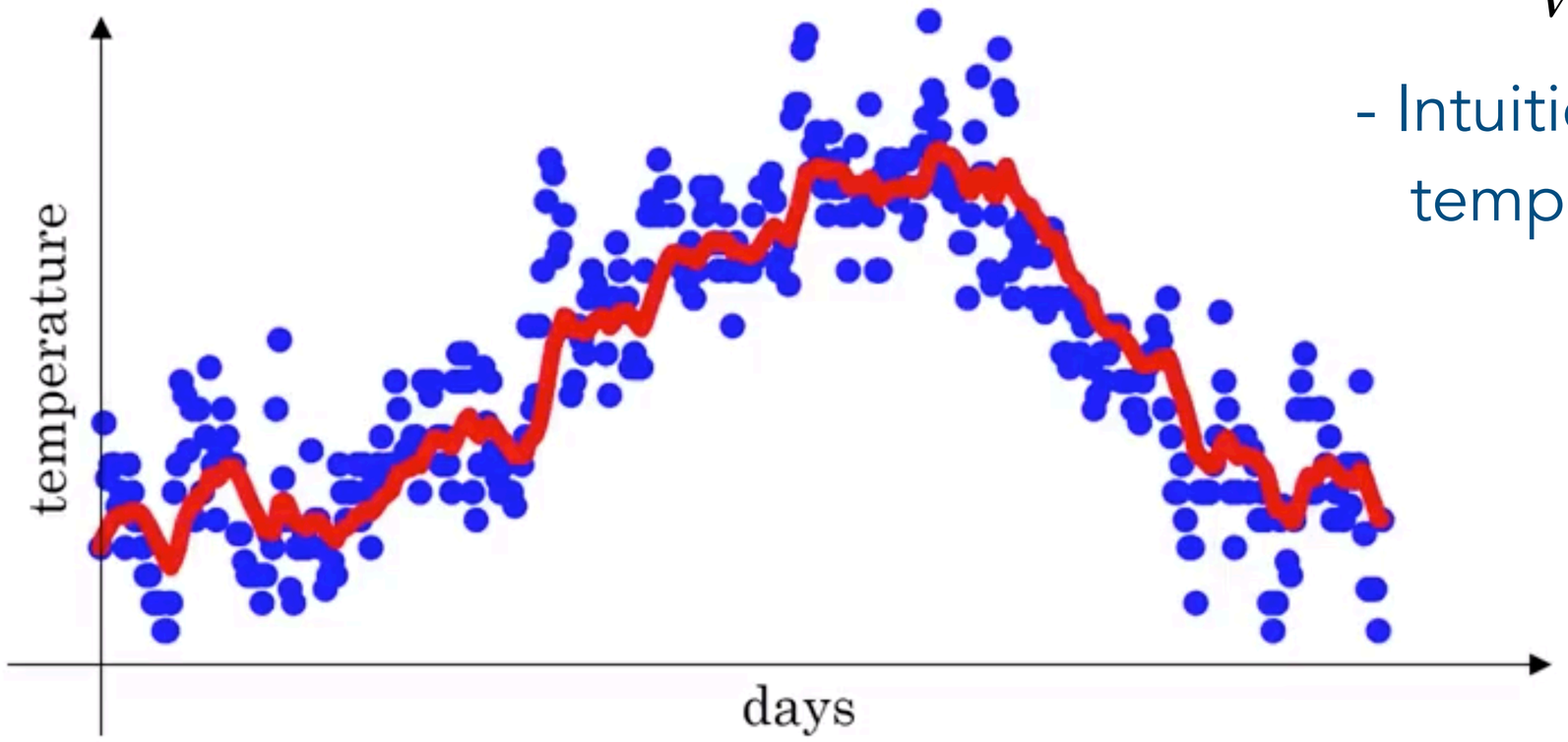
$$\vdots$$

$$V_t = 0.9V_{t-1} + 0.1\theta_t$$

Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year



- General formula:

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

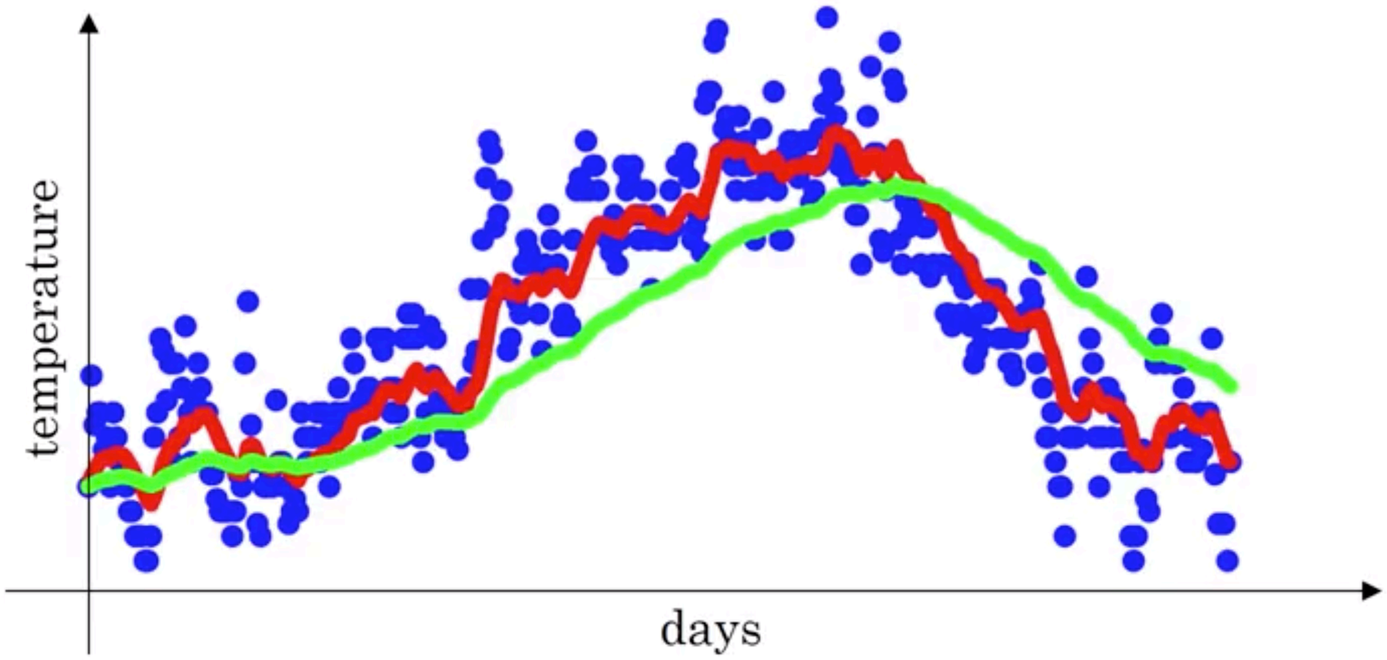
- Intuition: V_t approximates temperature over

$$\approx \frac{1}{1-\beta} \text{ days}$$

Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year



- Examples:

$$\beta = 0.9 \rightarrow \approx 10 \text{ days}$$

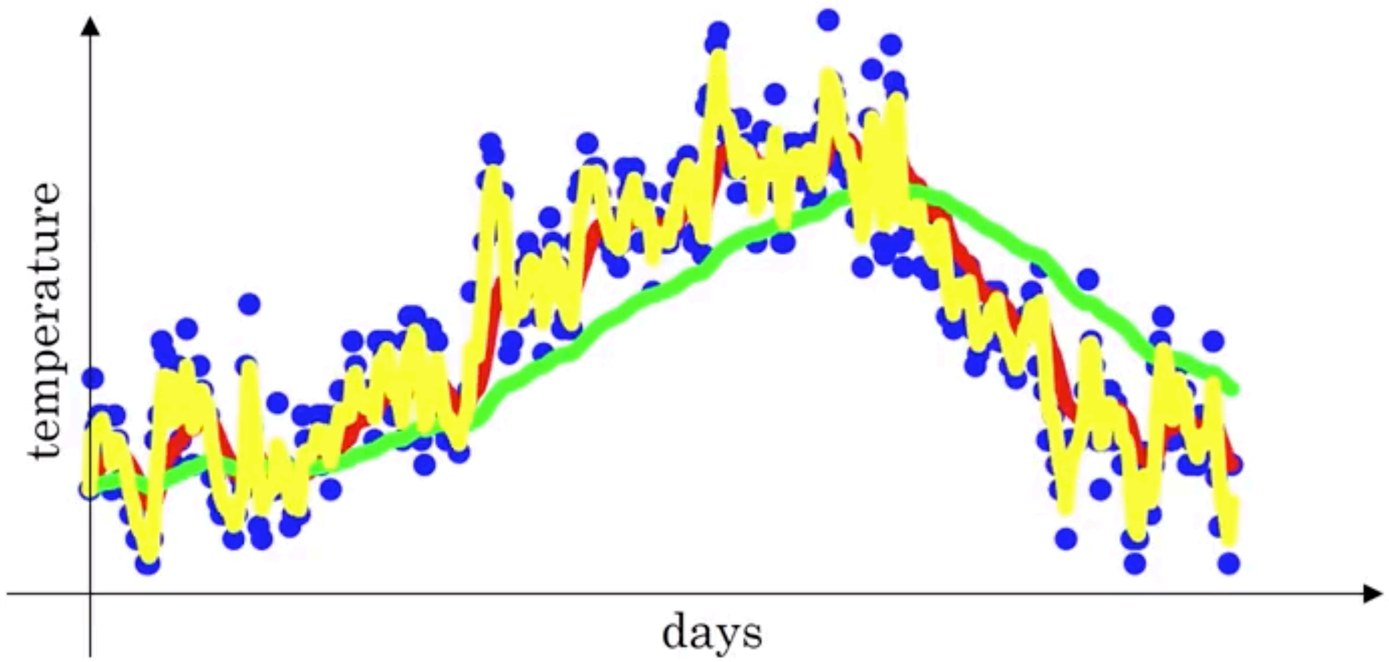
$$\beta = 0.98 \rightarrow \approx 50 \text{ days}$$

$$\beta = 0.5 \rightarrow \approx 2 \text{ days}$$

Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year



- Examples:

$$\beta = 0.9 \rightarrow \quad \approx 10 \text{ days}$$

$$\beta = 0.98 \rightarrow \quad \approx 50 \text{ days}$$

$$\beta = 0.5 \rightarrow \quad \approx 2 \text{ days}$$

Going beyond RMSprop: **Adam** algorithm

- Idea: Use weighted moving average in gradient also:

Going beyond RMSprop: **Adam** algorithm

- Idea: Use weighted moving average in gradient also:

  + **RMSprop** rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

Going beyond RMSprop: **Adam** algorithm

- Idea: Use weighted moving average in gradient also:

+ **RMSprop** rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

+ **Adam** rule: $$E[g^2]_t = \beta_2 \cdot E[g^2]_{t-1} + (1 - \beta_2) \cdot g_t^2$$

and

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla f_{i_t}(x_t)$$

``Moving averages are essentially about averaging many previous values in order to become independent of local fluctuations and focus on the overall trend``

Going beyond RMSprop: **Adam** algorithm

- Idea: Use weighted moving average in gradient also:

+ **RMSprop** rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

+ **Adam** rule: $$E[g^2]_t = \beta_2 \cdot E[g^2]_{t-1} + (1 - \beta_2) \cdot g_t^2$$

and

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla f_{i_t}(x_t)$$

``Moving averages are essentially about averaging many previous values in order to become independent of local fluctuations and focus on the overall trend``

Further: $$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \widehat{v}_t = \frac{E[g^2]_t}{1 - \beta_2^t}$$

Going beyond RMSprop: **Adam** algorithm

- Idea: Use weighted moving average in gradient also:

+ **RMSprop** rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

+ **Adam** rule: $$E[g^2]_t = \beta_2 \cdot E[g^2]_{t-1} + (1-\beta_2) \cdot g_t^2$$

and

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \nabla f_{i_t}(x_t)$$

``Moving averages are essentially about averaging many previous values in order to become independent of local fluctuations and focus on the overall trend``

Further: $$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \widehat{v}_t = \frac{E[g^2]_t}{1-\beta_2^t}$$

- Algorithm: $$x_{t+1} = x_t - \frac{\eta}{\sqrt{\widehat{v}_t} + \epsilon} \cdot \widehat{m}_t$$

$$\beta_1 = 0.9, \ \beta_2 = 0.999$$

Bias correction in weighted averages

(Adapted from Ng's lectures)

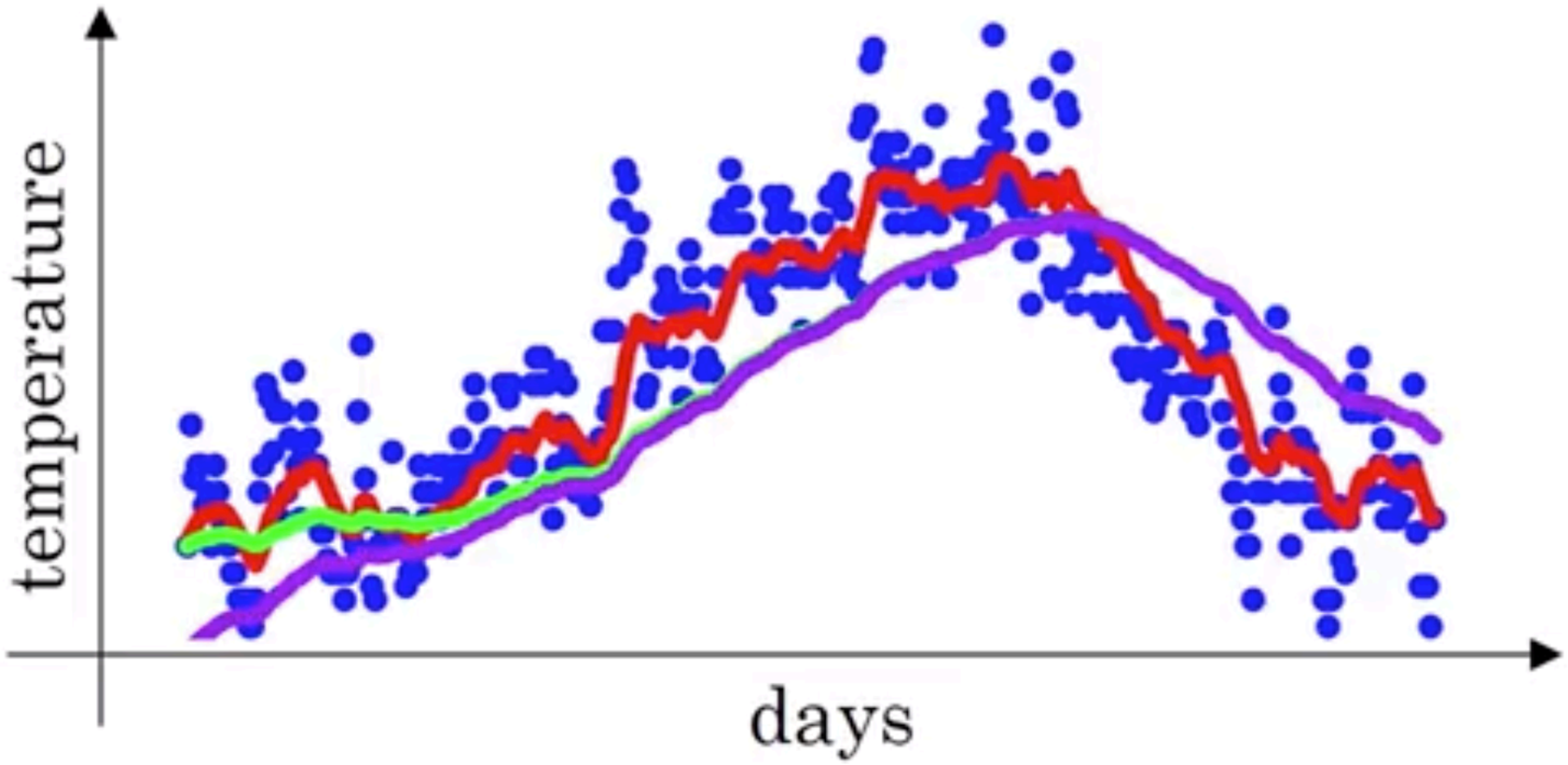
- How to explain these ``weird`` denominators?

Bias correction in weighted averages

(Adapted from Ng's lectures)

- How to explain these ``weird`` denominators?

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

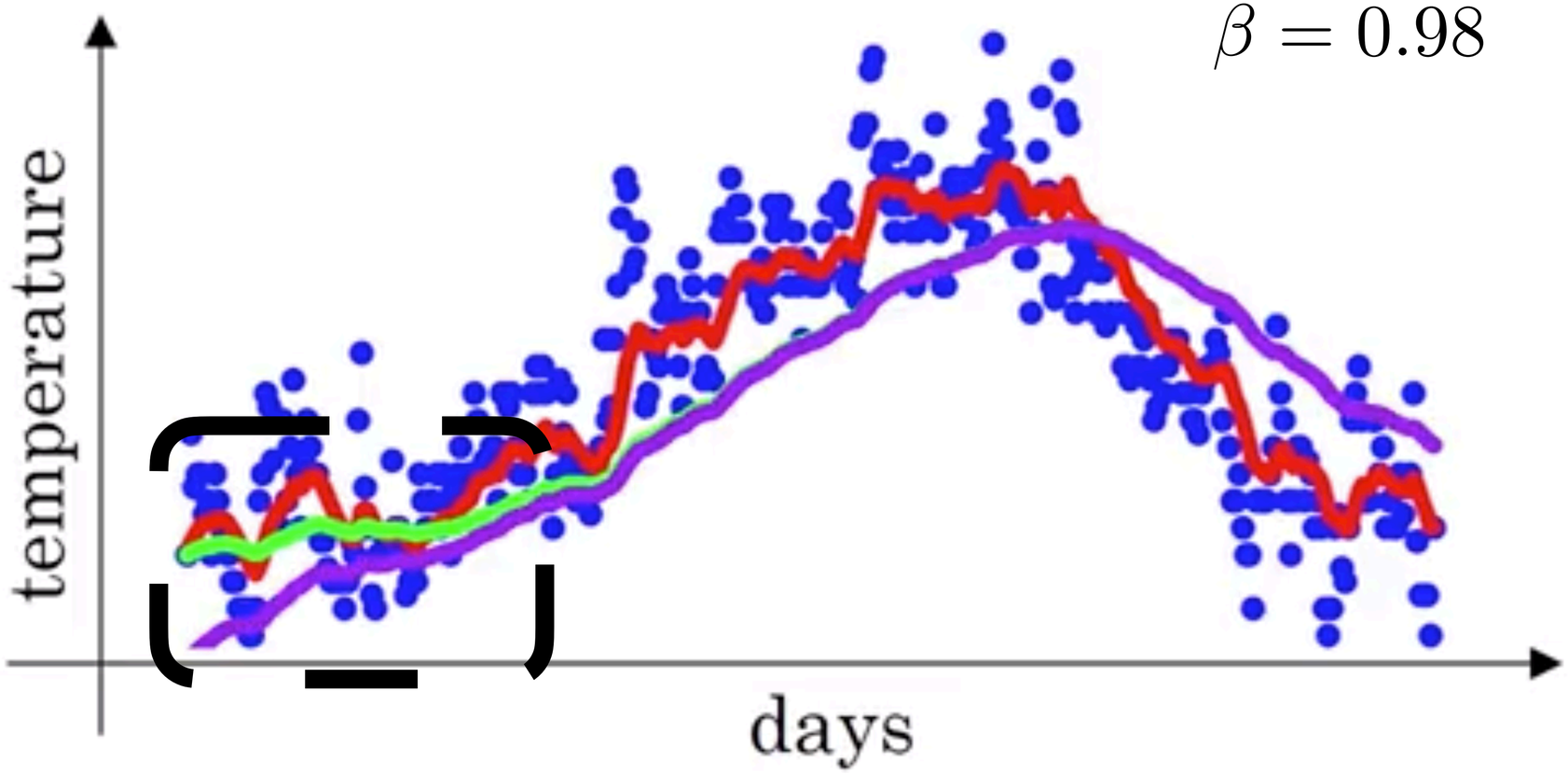


Bias correction in weighted averages

(Adapted from Ng's lectures)

- How to explain these ``weird`` denominators?

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$



Other algorithms and sources

- Not a complete list: **AdaMax, Nadam, AMSGrad**, ..

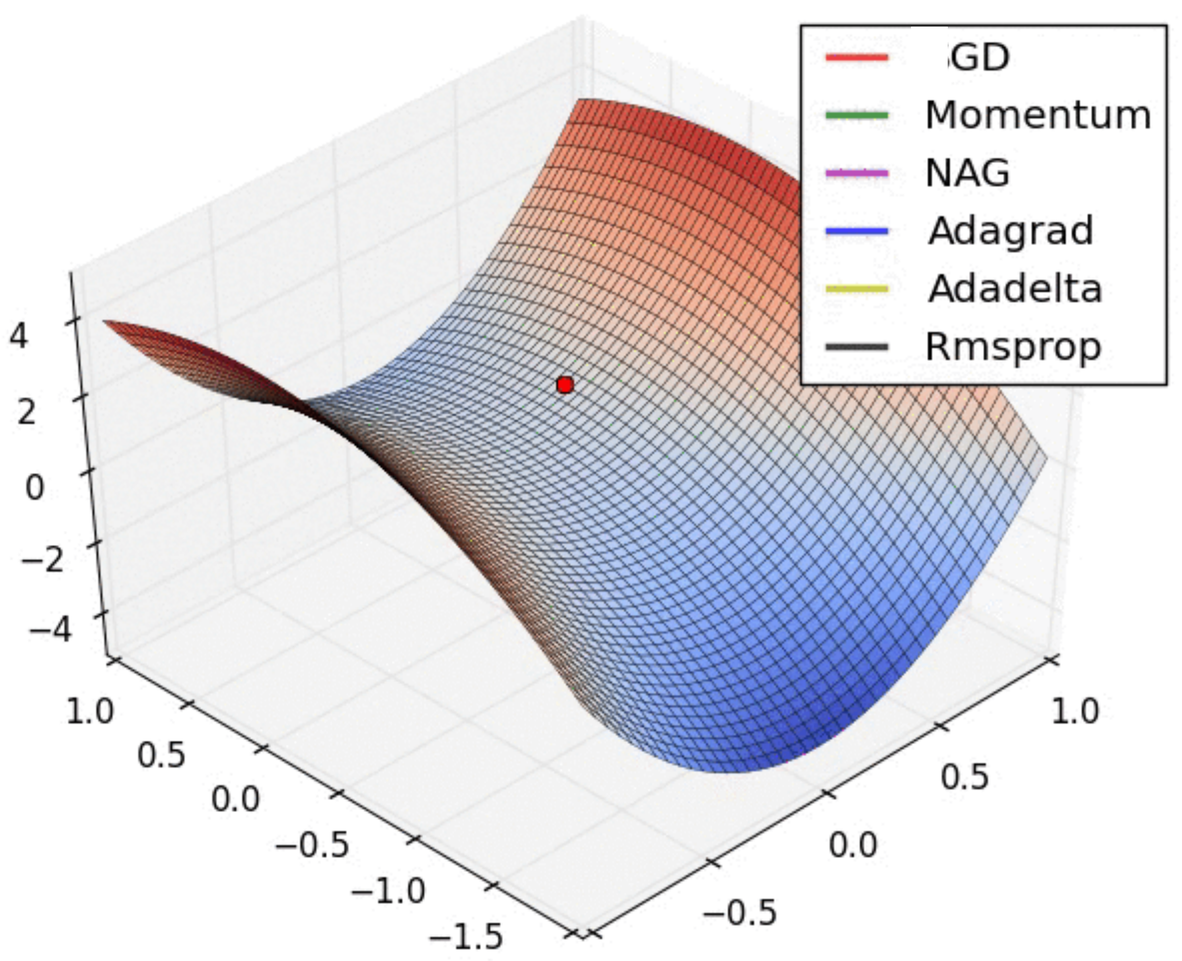
- A nice blog post on the matter:

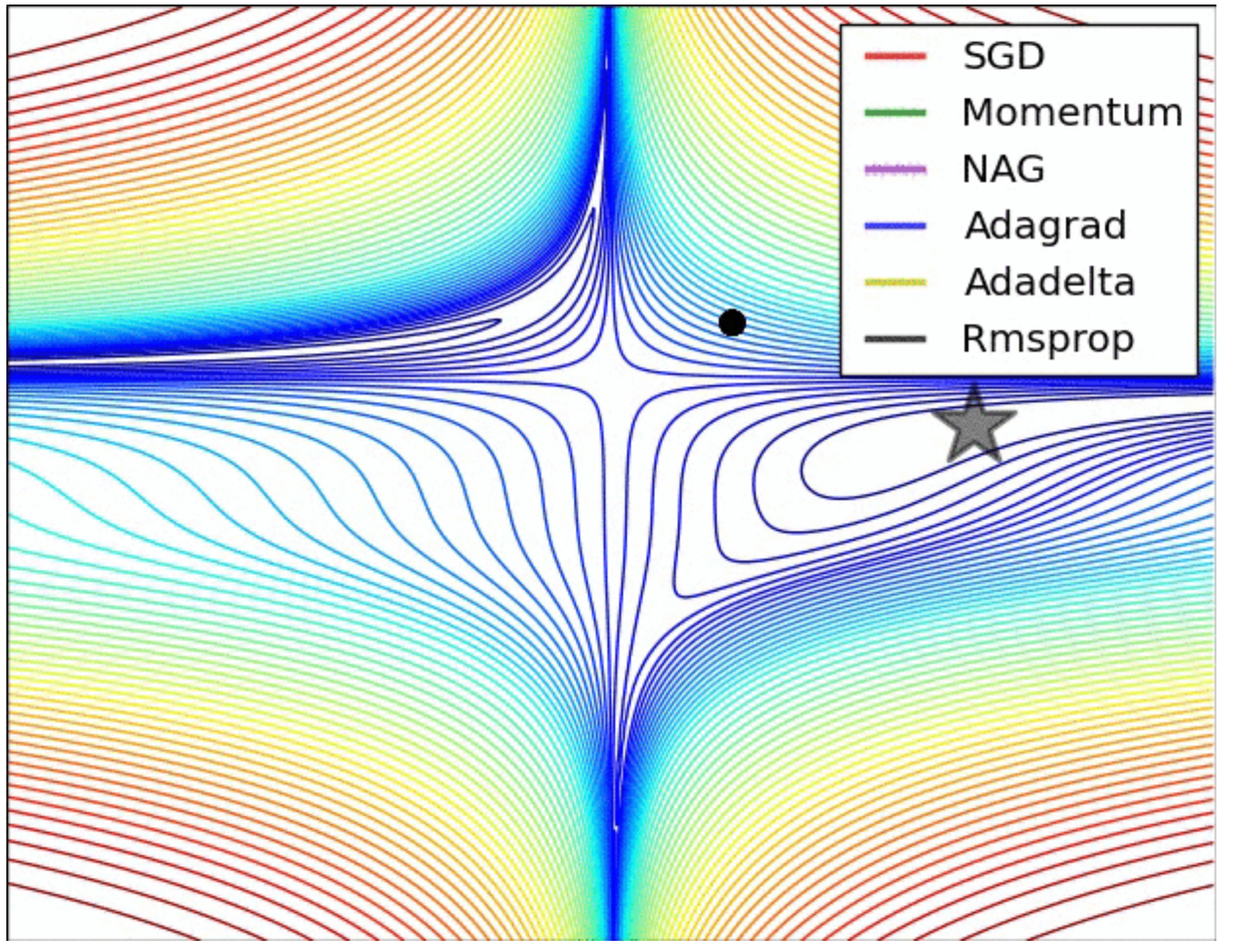
http://ruder.io/optimizing-gradient-descent/

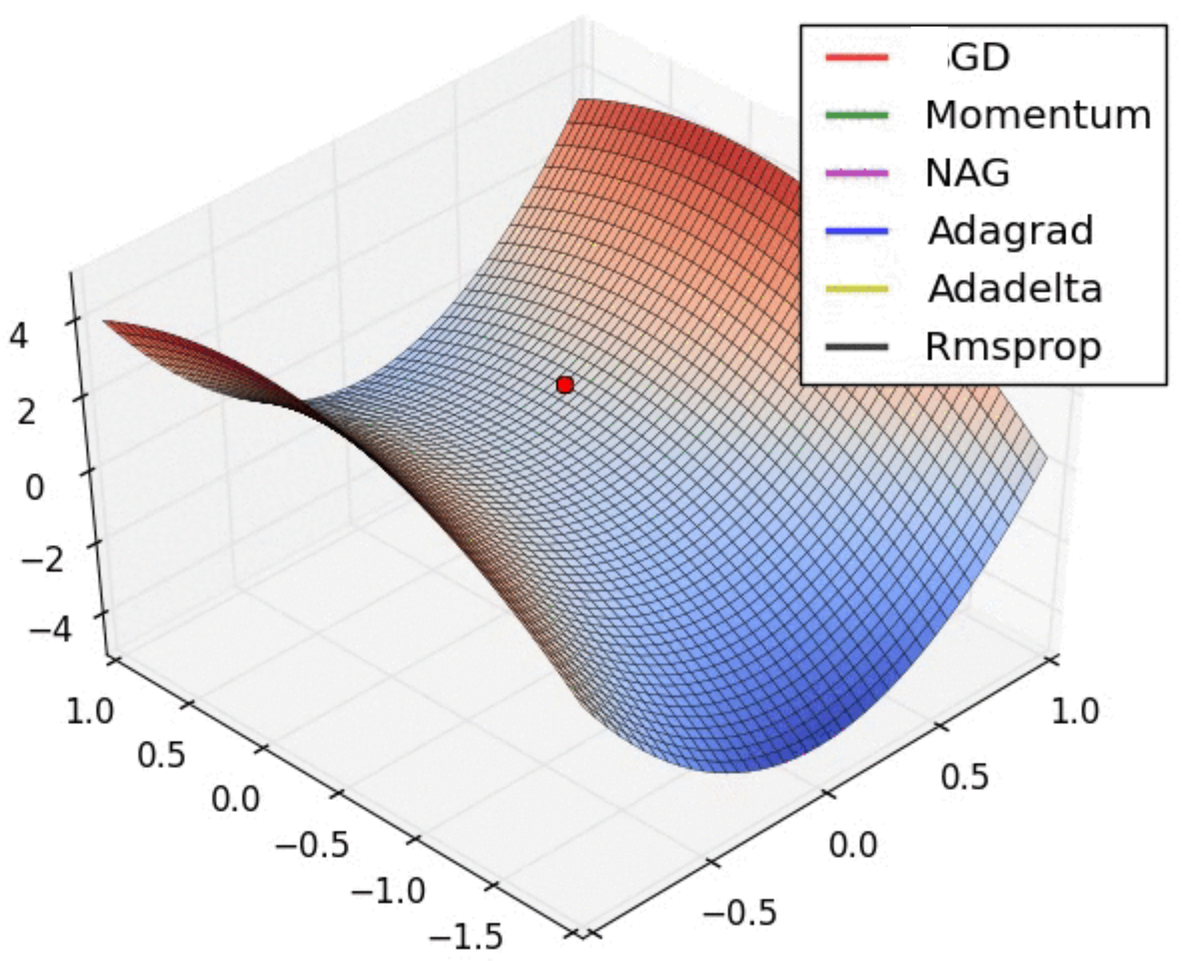
https://johnchenresearch.github.io/demon/

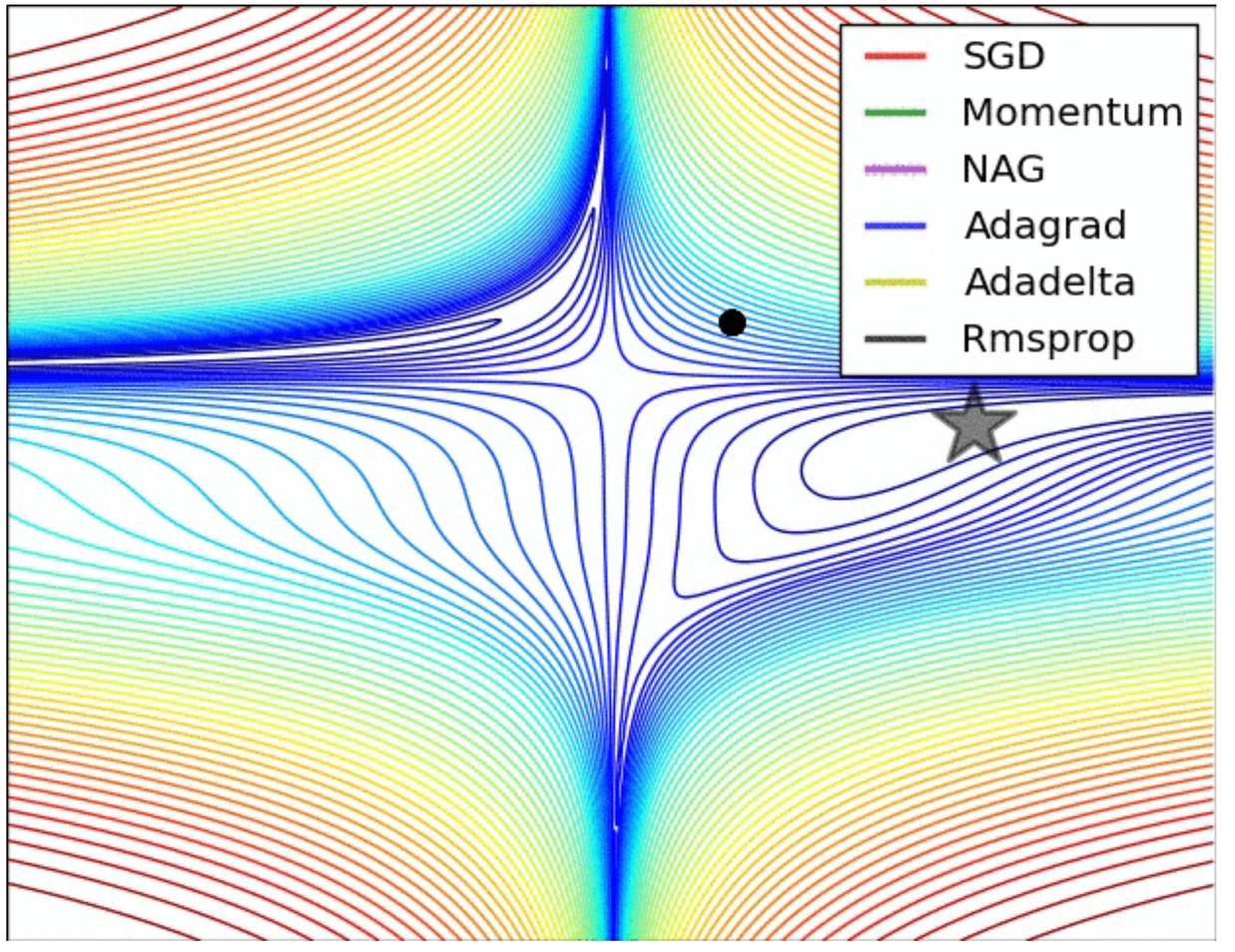
- Choosing the right algorithm: there is no consensus about it (see next slides)

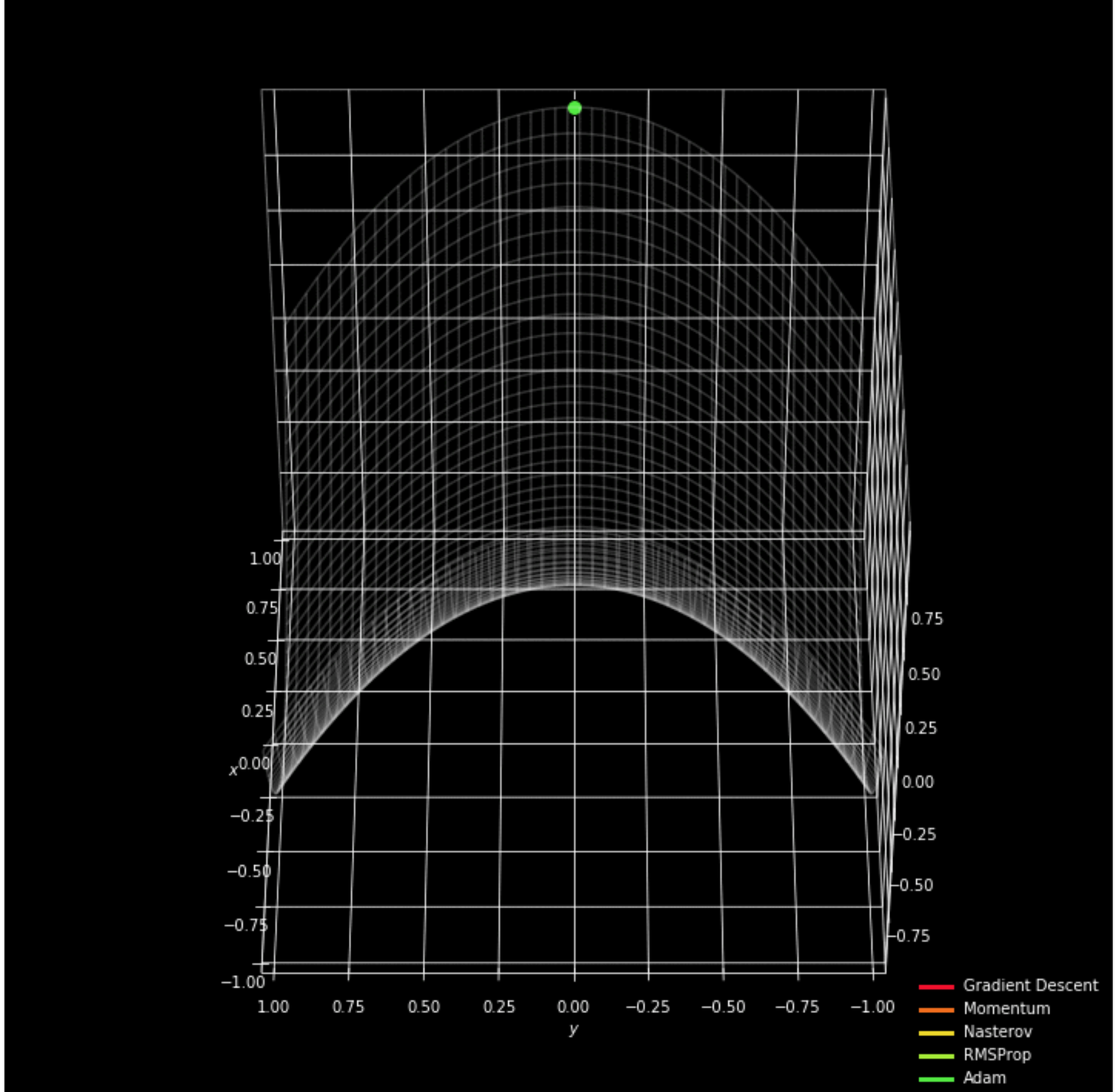
- A visualization of their performance in toy examples:

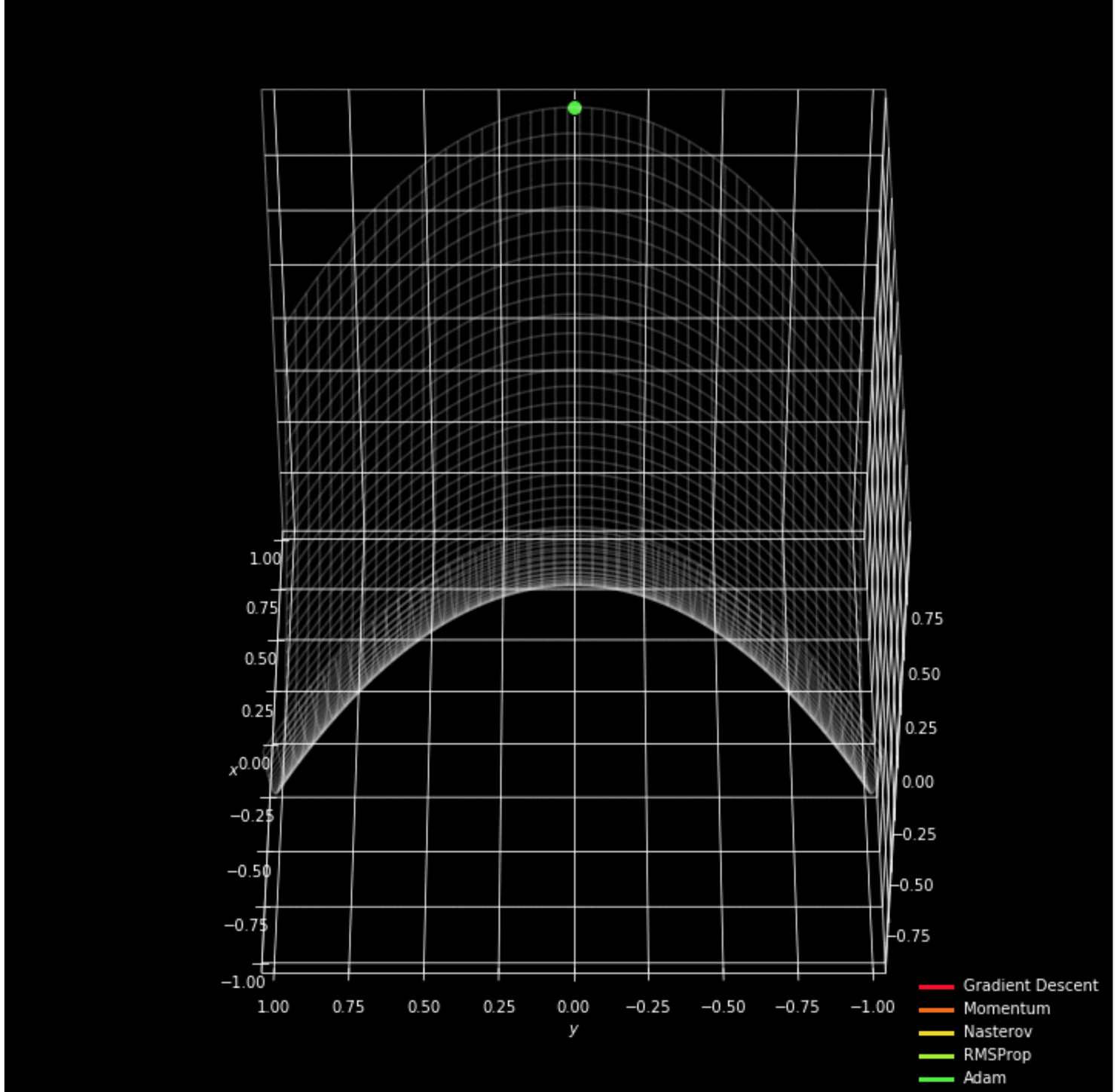

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


1.00
0.75
0.50
0.25
x 0.00
−0.25
−0.50
−0.75
−1.00
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
y
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Gradient Descent
Momentum
Nasterov
RMSProp
Adam


1.00
0.75
0.50
0.25
x 0.00
−0.25
−0.50
−0.75
−1.00
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
y
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Gradient Descent
Momentum
Nasterov
RMSProp
Adam