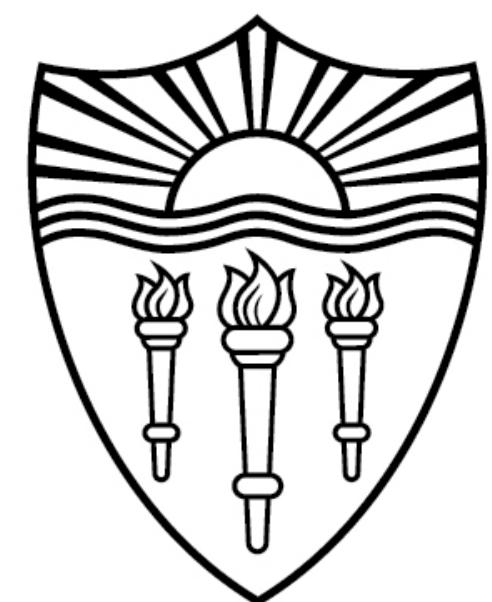


CSCI 544: Applied Natural Language Processing

# **Deep Neural Networks for NLP**

Xuezhe Ma (Max)



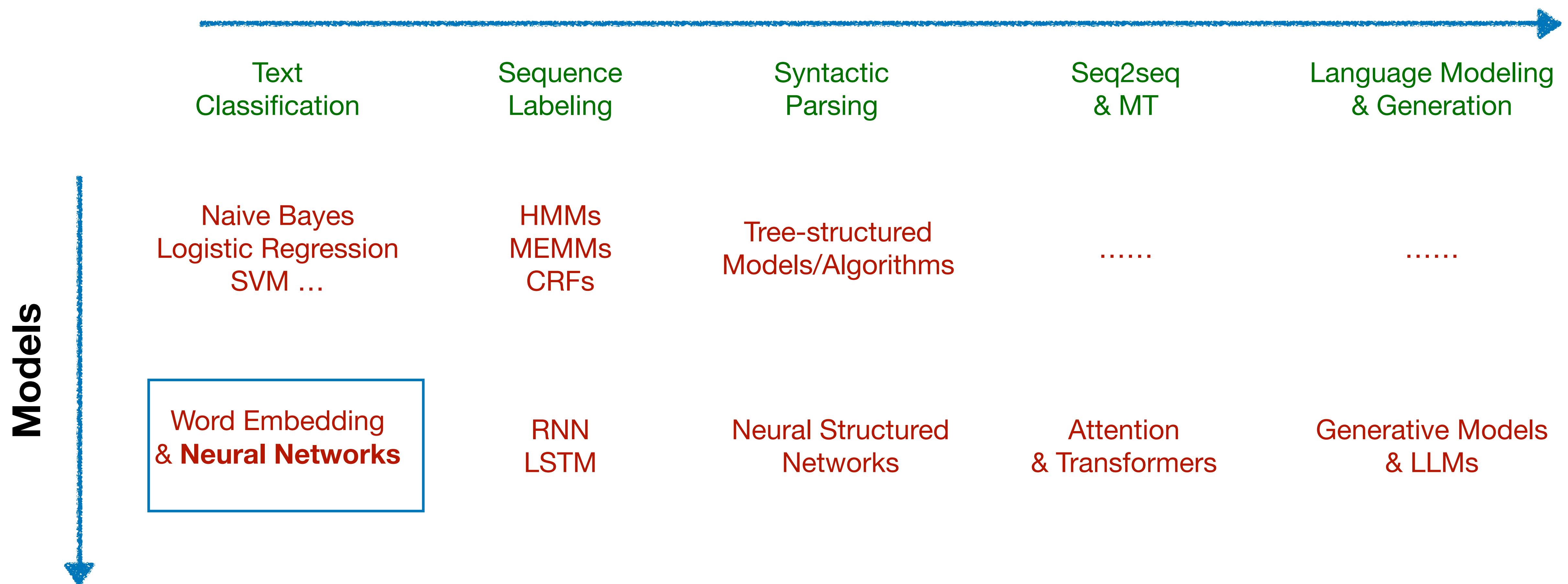
**USC** University of  
Southern California

# Logistical Notes

- **Syllabus Update**
  - We move the lecture for PyTorch to 01/25 (next Thursday)

# Course Organization

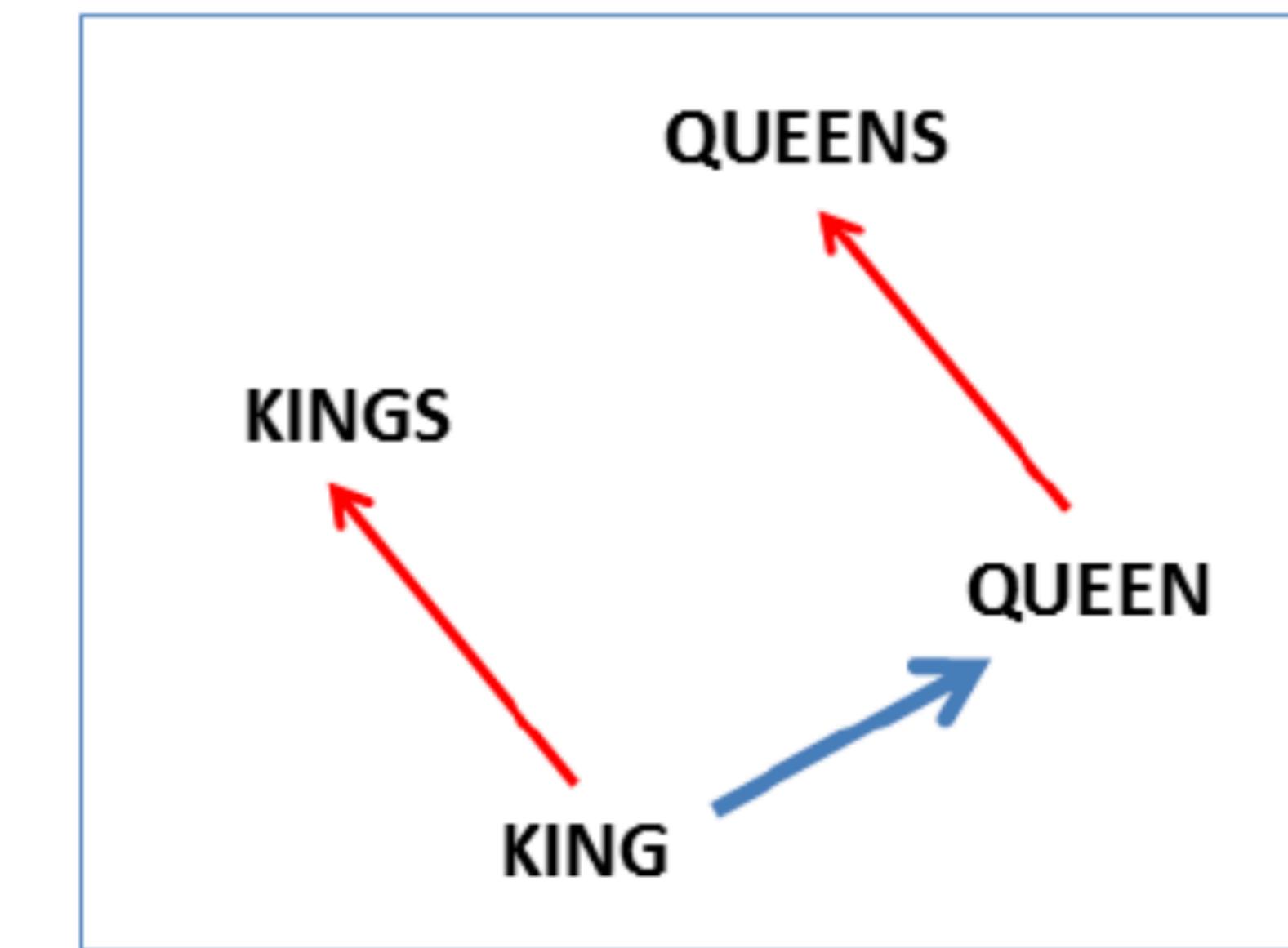
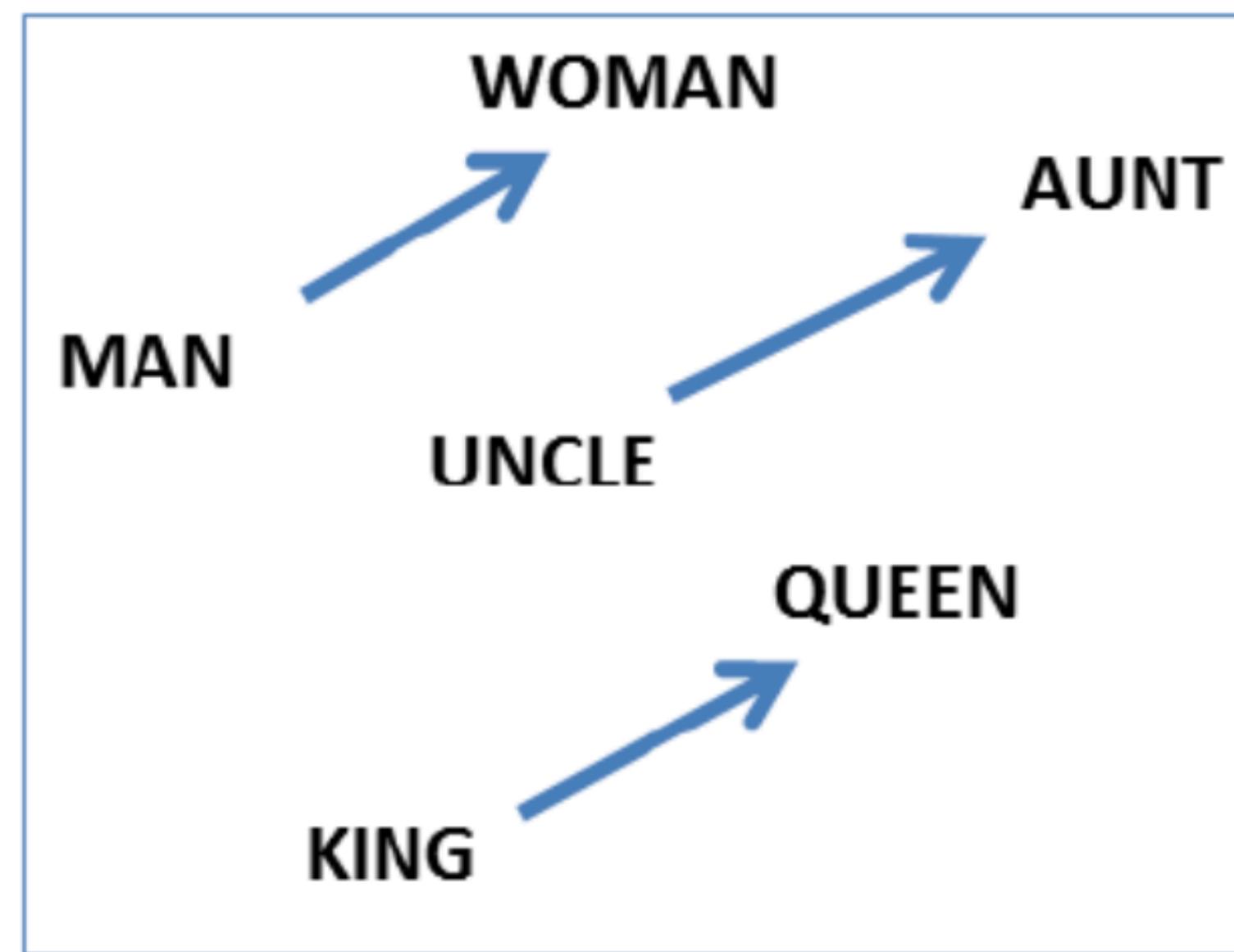
## NLP Tasks



# Recap: Problems of Traditional Text Classification

- Insufficient attention on feature representations

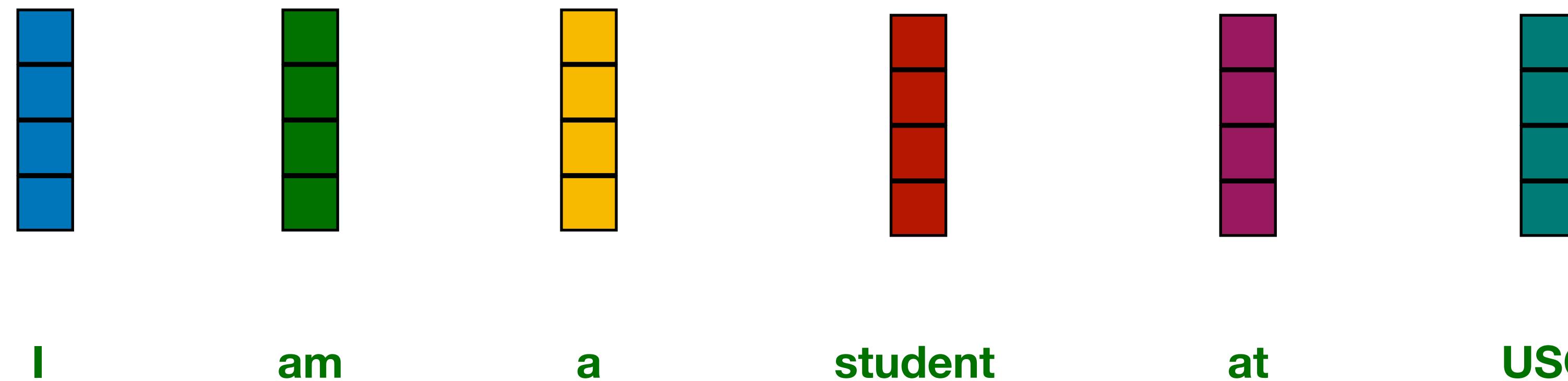
- Bag of words & TF-IDF
- Only frequency information, not semantic meaning of each word (**last lecture**)
- **No contextual information (this lecture)**



# Apply Word Embeddings to Tasks

- Classification

- We need **a single feature vector** to feed into ML classifiers



Element-wise pooling:

$$y = \text{pool}(x_1, \dots, x_n)$$

e.g. AVG, MAX

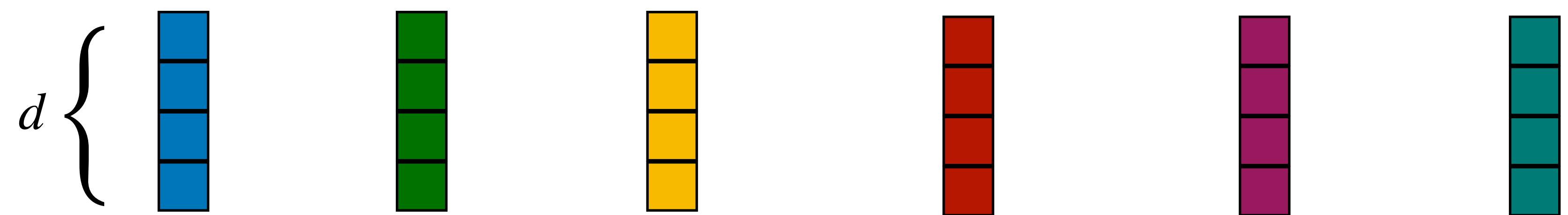
No contextual information!

# Goal: One Vector to Represent a Document

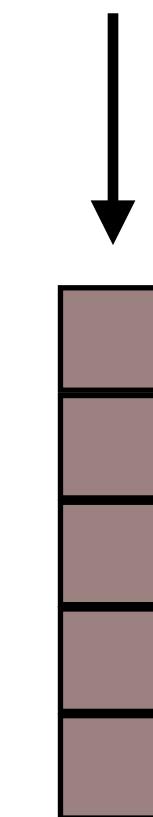
- Classification

- We need **a single feature vector** to feed into ML classifiers

$X \in \mathbb{R}^{n \times d}$  Word Embeddings



$$y = f(x_1, \dots, x_n) \in \mathbb{R}^{d'}$$



Contextual information in  $y$ !

# Compositionality

**Bat**



**hit bat**



**hit with bat**



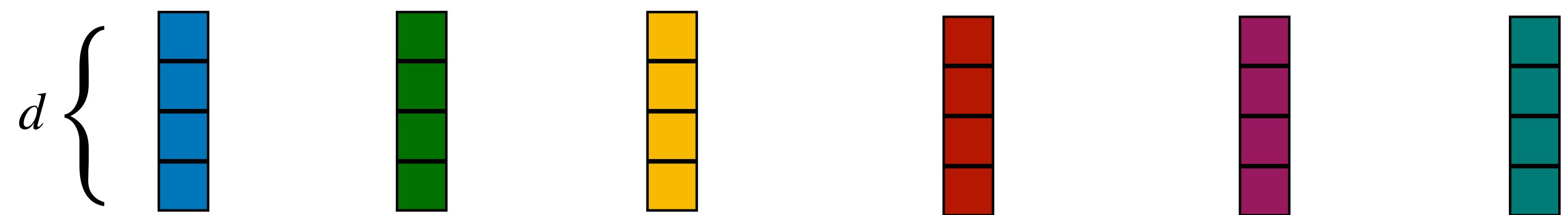
It is **(almost) infeasible** to manually define semantic compositionality

# Goal: One Vector to Represent a Document

- Classification

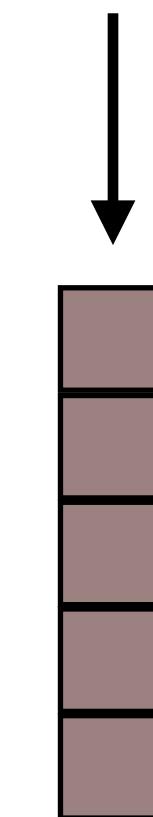
- We need **a single feature vector** to feed into ML classifiers

$X \in \mathbb{R}^{n \times d}$  Word Embeddings



I                  am                  a                  student                  at                  USC

$$y = f(x_1, \dots, x_n) \in \mathbb{R}^{d'}$$



Contextual information in  $y$ !

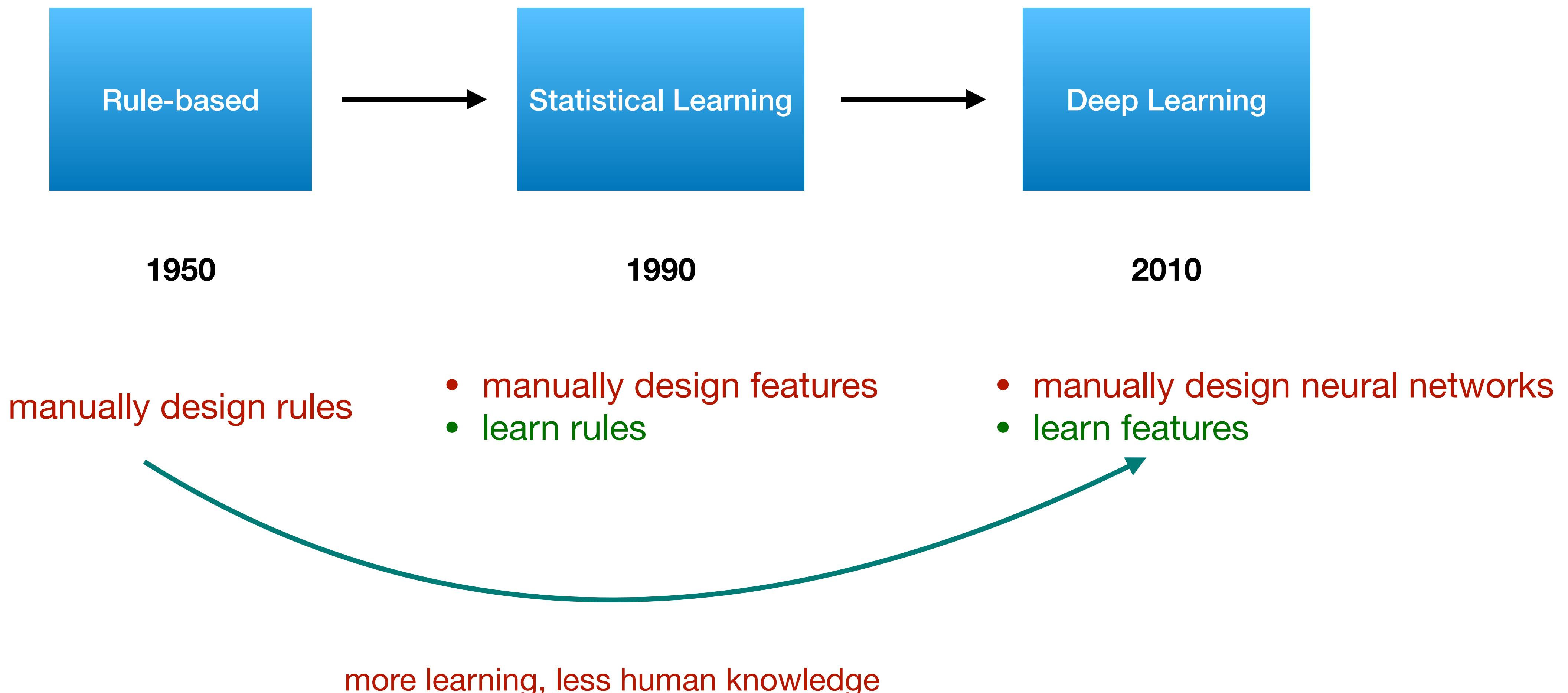
# Feature Representations:

## Engineering vs. Learning

Why deep learning?



# Evolution of NLP



# Outline

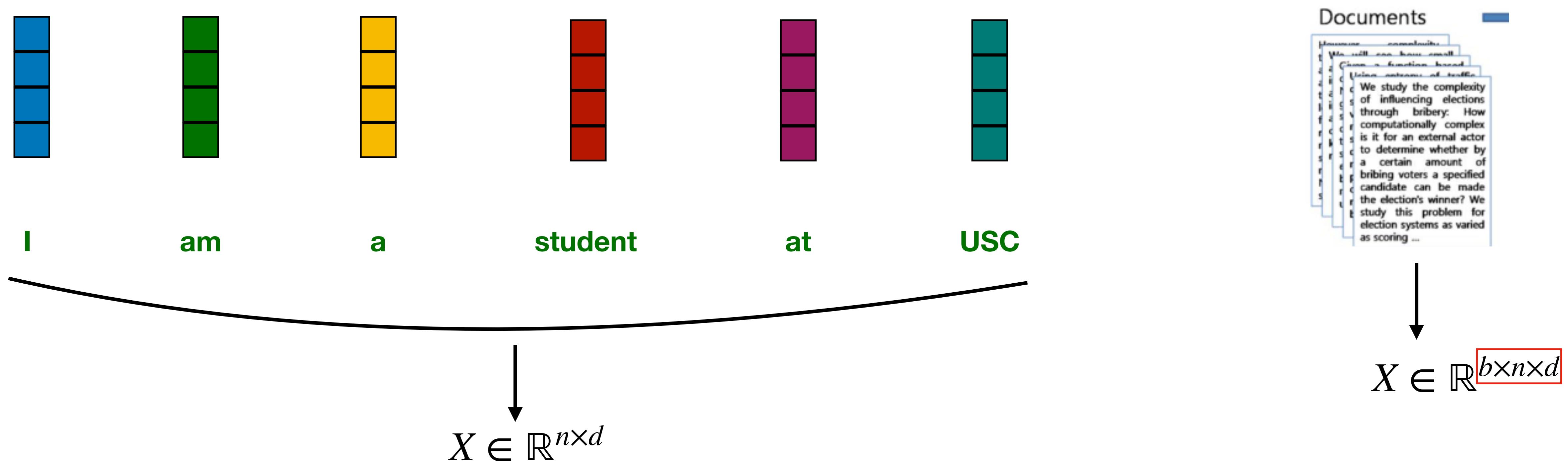
- A Brief Introduction of Some Neural Networks
  - Feedforward Neural Networks
  - Convolutional Neural Networks (CNNs)
  - Recurrent Neural Networks (RNNs) (**next lecture**)

# Deep Neural Networks

# Deep Neural Networks: Basic Concepts

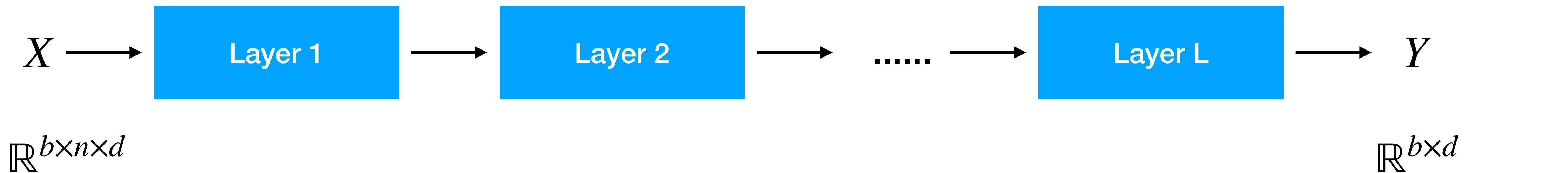
- The dimension of batch

- Each document is a sequence of embedding vectors (one vector per word)
- We once feed **multiple documents** to as input a DNN



# Deep Neural Networks: Basic Concepts

- A deep neural network consists of multiple layers
  - Different layers may have different **architectures** and serve different purposes
  - Each layer maintains its own (learnable) parameters
  - Each layer may require specific shape of its input  $X$

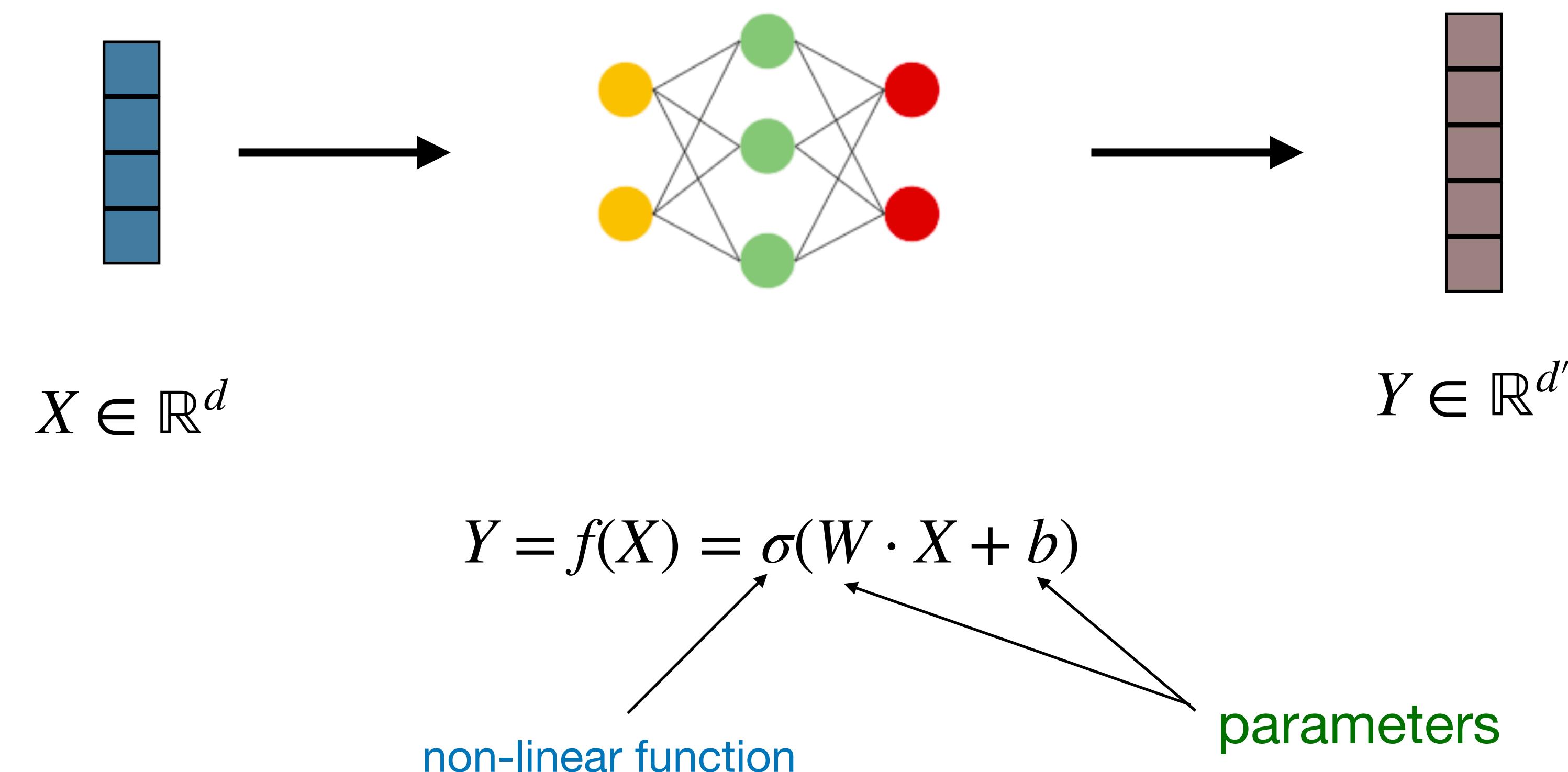


# Feedforward Neural Networks

- One layer of Feedforward neural network

$$f: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$$

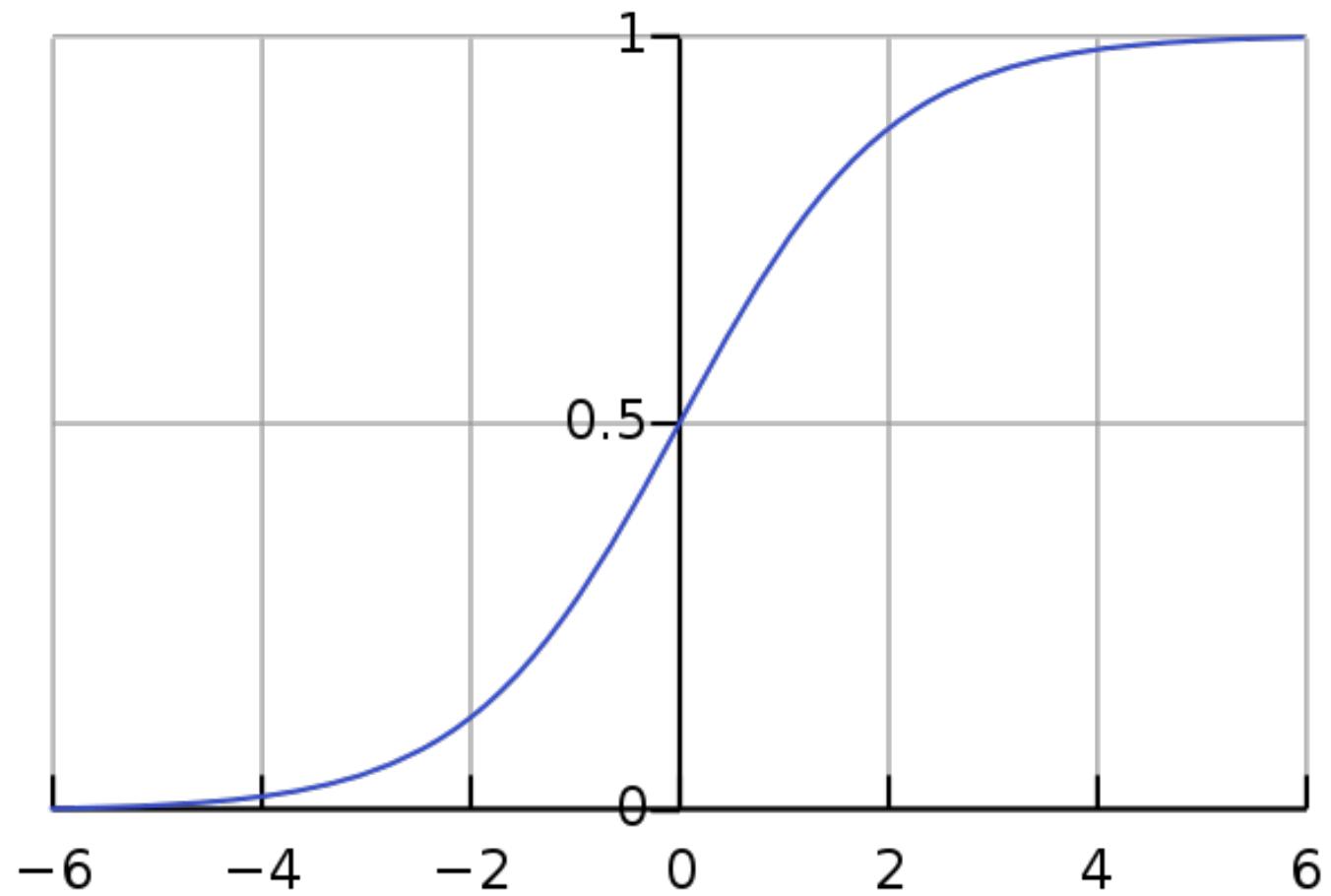
*d and d' are pre-defined hyper-parameters*



# Non-linear Functions

- A.k.a Activation functions

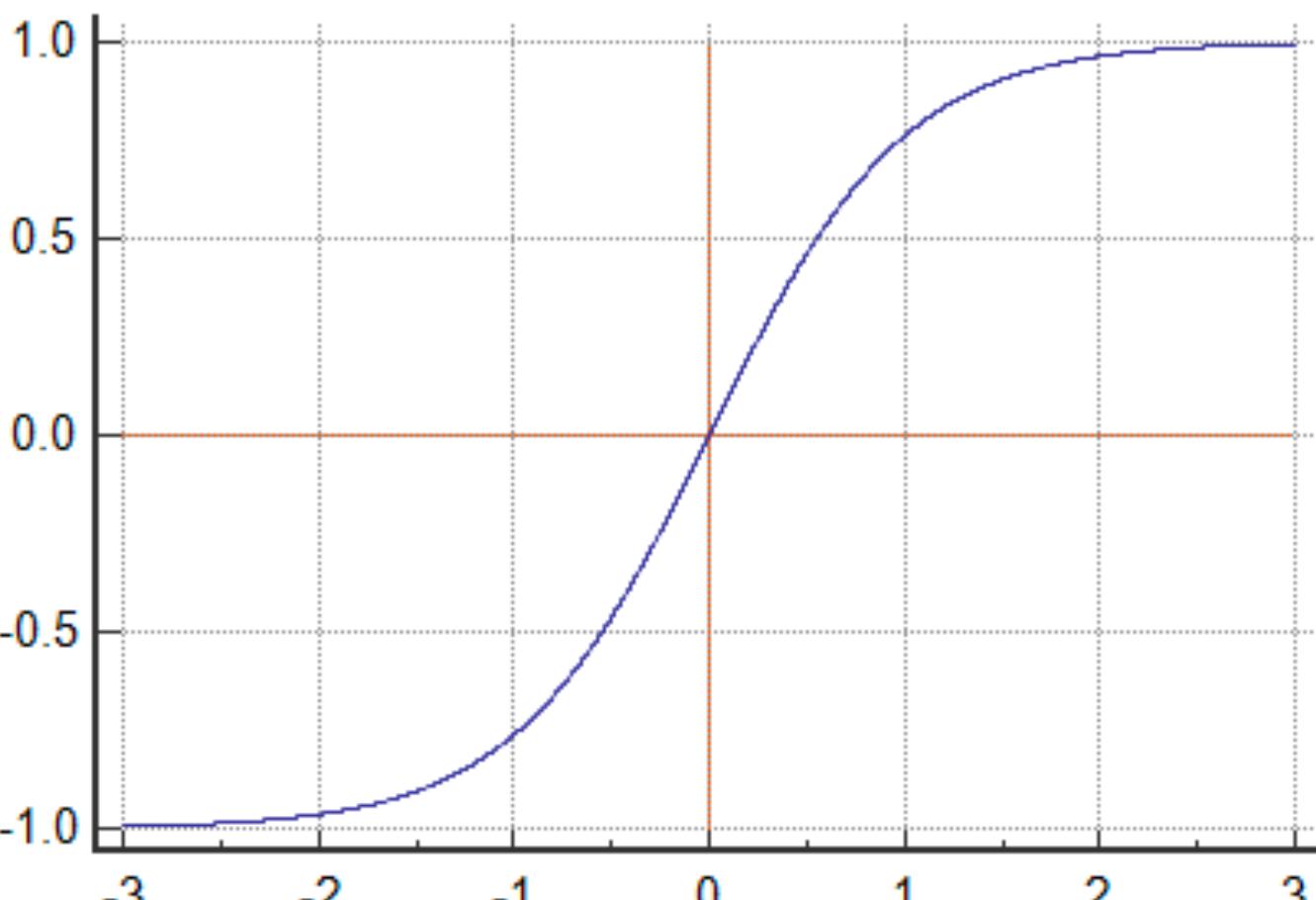
Sigmoid function



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

range (0,1)

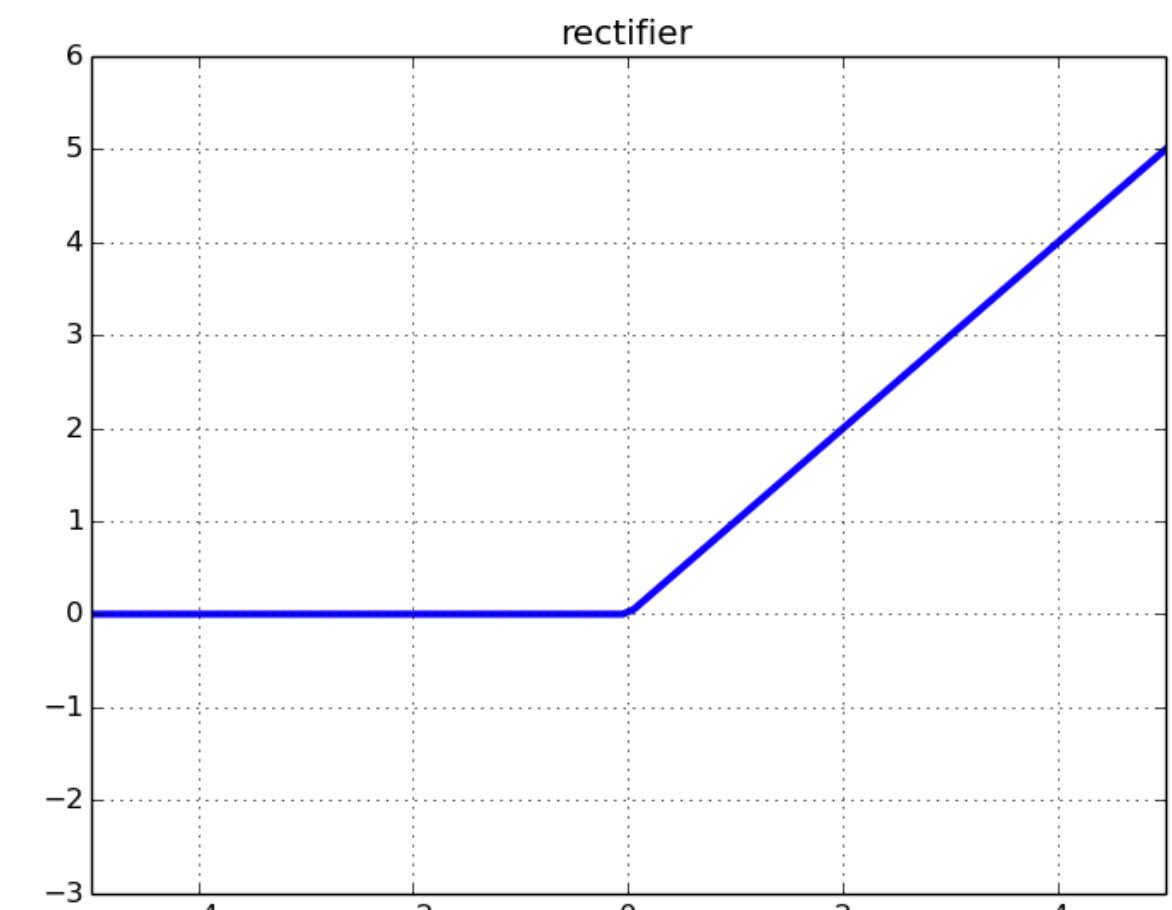
tanh function



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

range (-1,1)

ReLU function



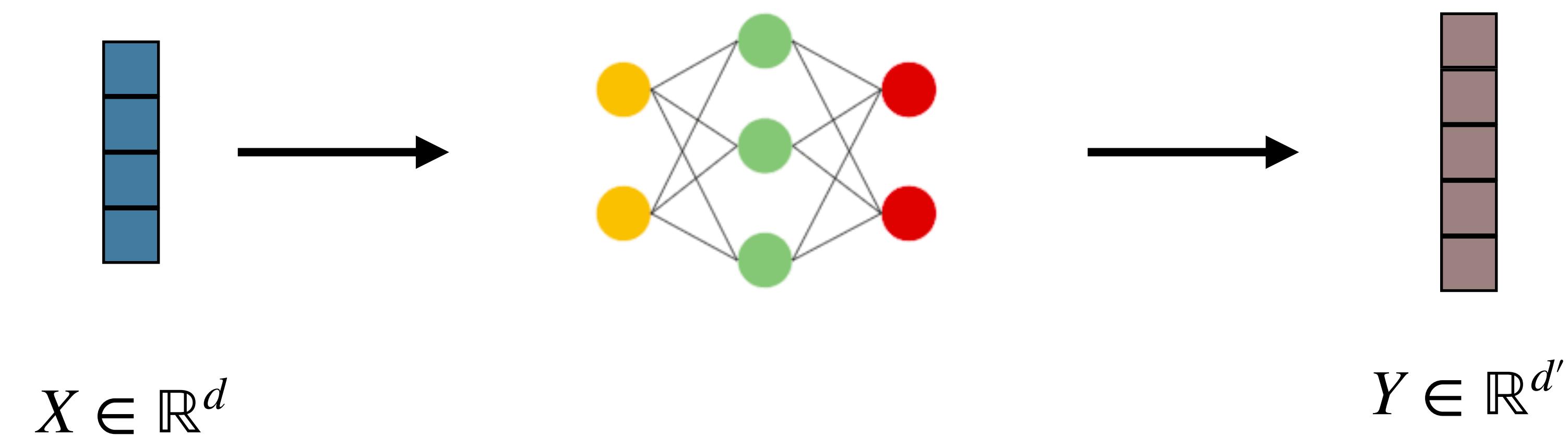
$$x = \max(0, x)$$

range (0,∞)

# Feedforward Neural Networks

- One layer of Feedforward neural network

$$f: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$$



$$Y = f(X) = \sigma(W \cdot X + b)$$

parameters

What is the shape of  $W$  and  $b$  when  $d, d'$  are given?

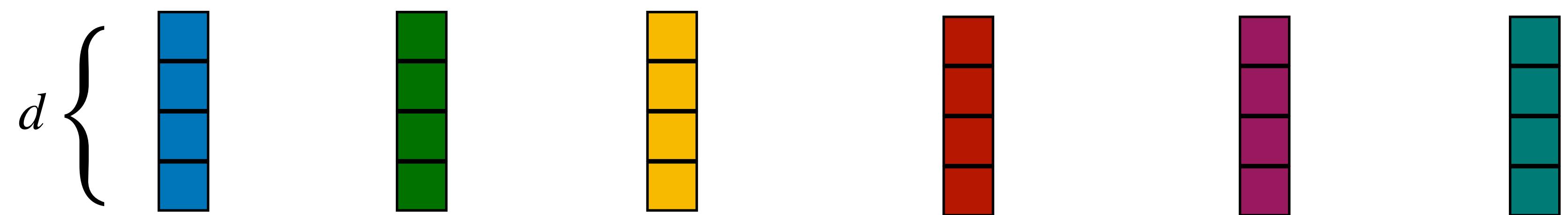
$$W \in \mathbb{R}^{d \times d'}, b \in \mathbb{R}^{d'}$$

# Goal: One Vector to Represent a Document

- Classification

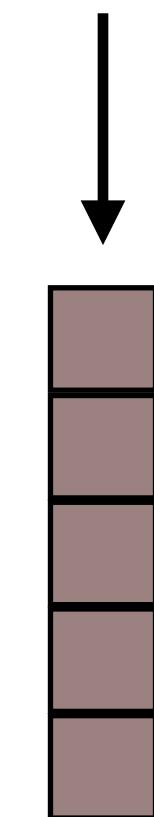
- We need **a single feature vector** to feed into ML classifiers

$X \in \mathbb{R}^{n \times d}$  Word Embeddings



I am a student at USC

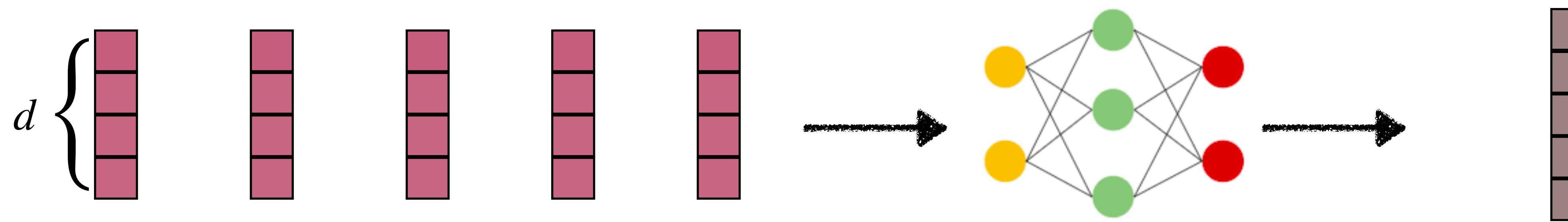
$$y = f(x_1, \dots, x_n) \in \mathbb{R}^{d'}$$



Can we use multi-layer feed forward neural networks to model a sentence/document?

# FeedForward Networks for Our Goal?

$$X \in \mathbb{R}^{n \times d} \quad \text{Word Embeddings} \qquad \qquad Y \in \mathbb{R}^{d'}$$

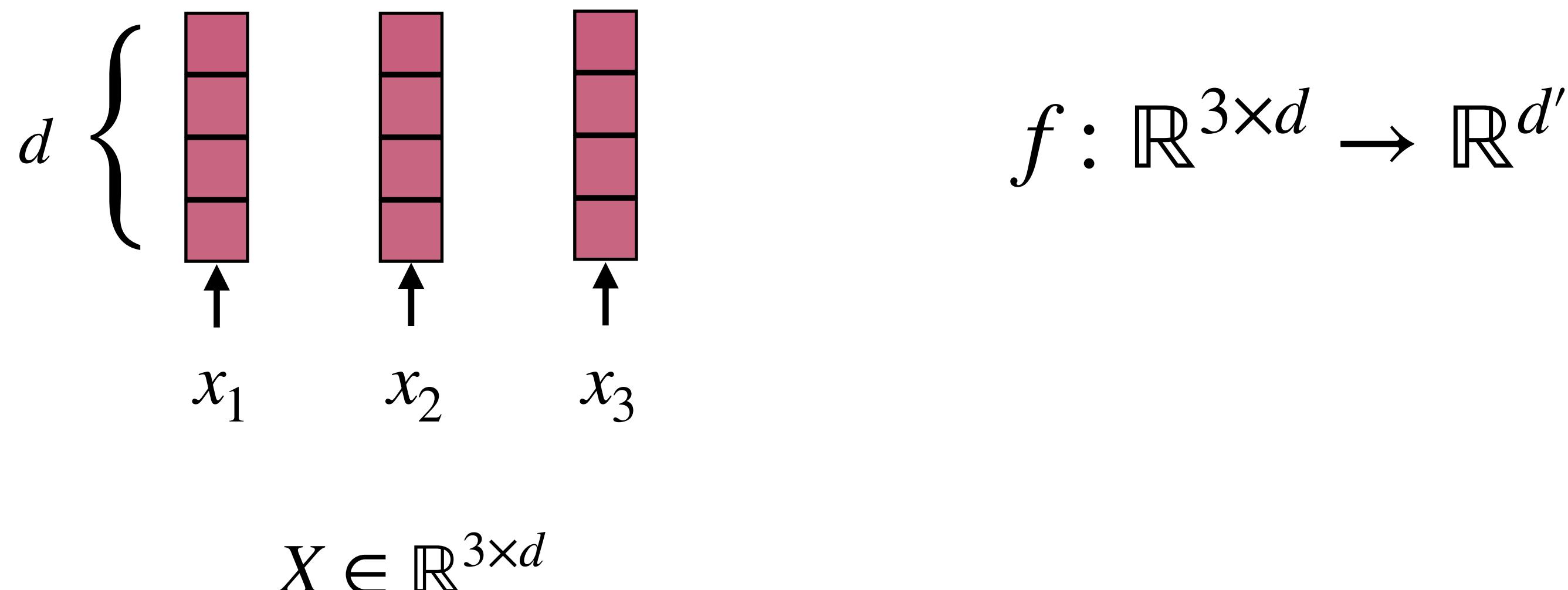


Max is working at USC

**Simple Solution:** flatten  $X$  to a one-dimensional vector, then feed it into a FFN?

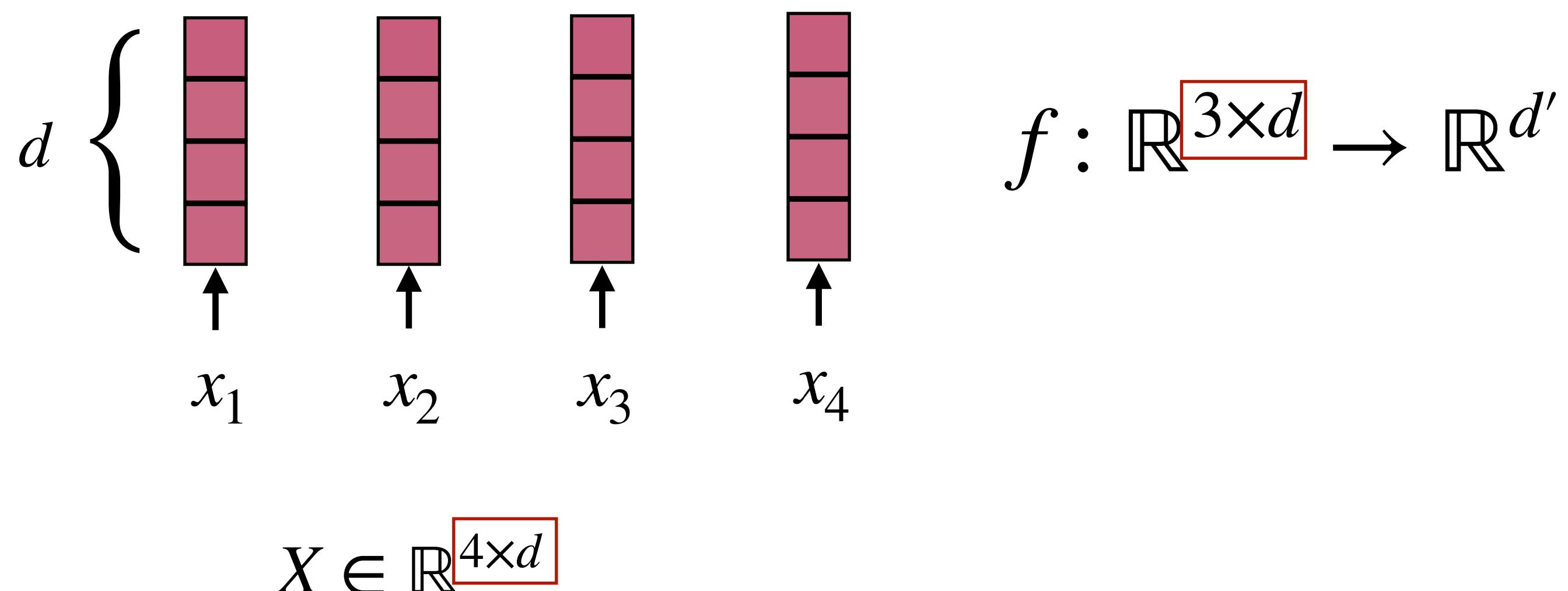
# Modeling Sentences with Various Lengths

- FFNs **cannot** model sentences with various lengths.
  - The input and output dimensions need be pre-defined for each layer



# Modeling Sentences with Various Lengths

- FFNs **cannot** model sentences with various lengths.
    - The input and output dimensions need be pre-defined for each layer

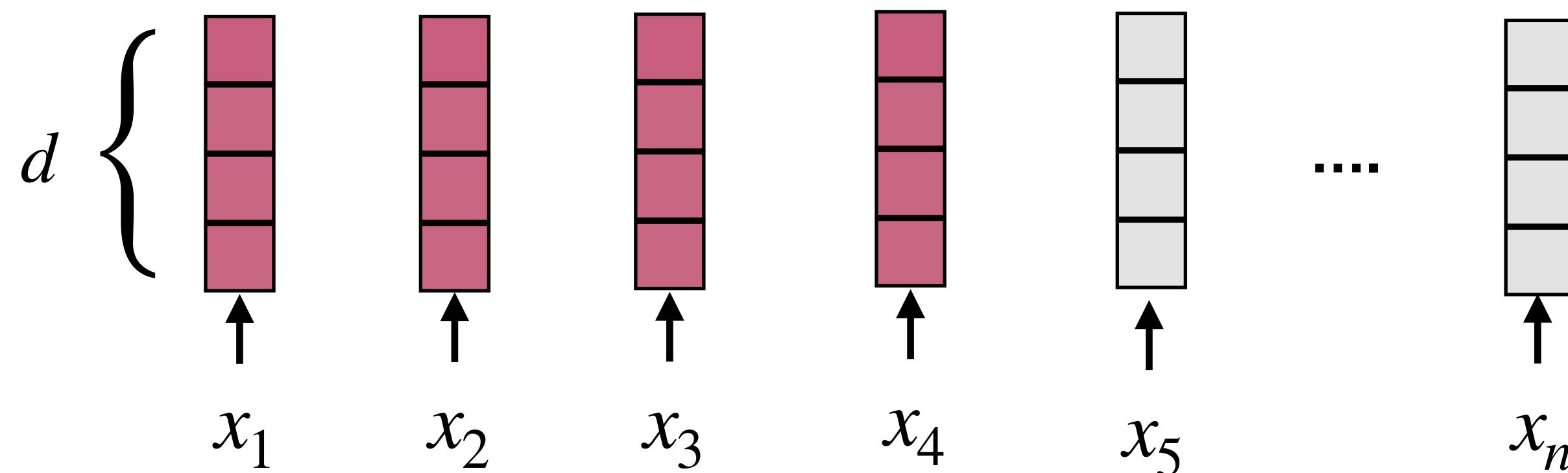


# Why not set a maximum length?

# Modeling Sentences with Various Lengths

- FFNs **cannot** model sentences with various lengths.

- The input and output dimensions need be pre-defined for each layer
- The parameter  $W$  would be too large for long sentences  $W \in \mathbb{R}^{n \times d \times d'}$



$$X \in \mathbb{R}^{n \times d}$$

Why not applying FFN on each vector individually?

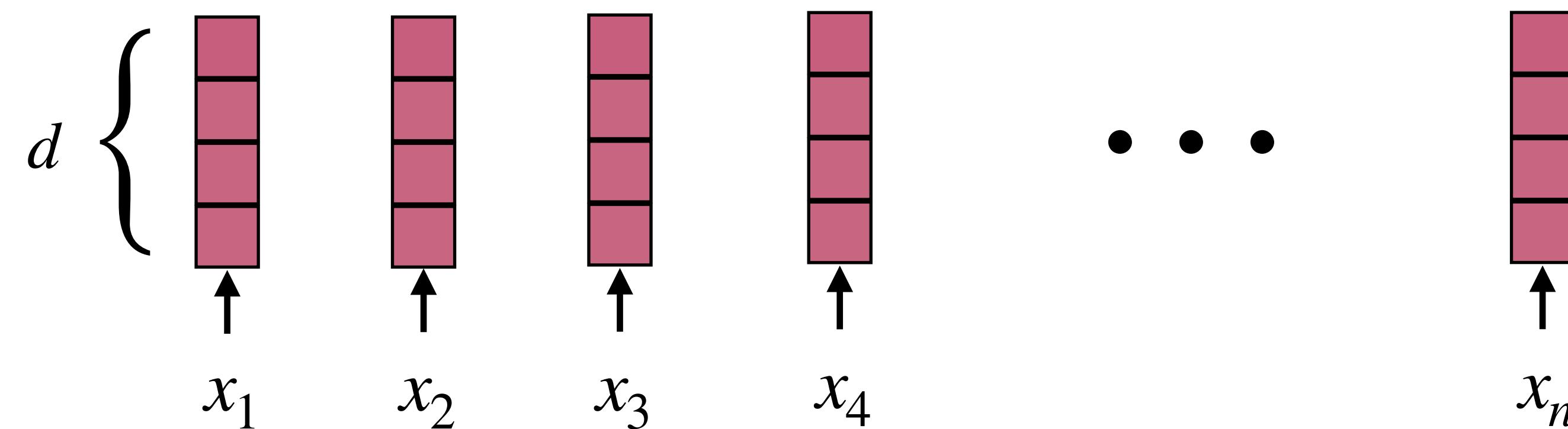
Still no context information!

# Convolutional Neural Networks (CNNs)

- Basic Idea: only model a segment of input with **fixed window-size**

$$f: \mathbb{R}^{k \times d} \rightarrow \mathbb{R}^{d'} \quad k \text{ is the window size}$$

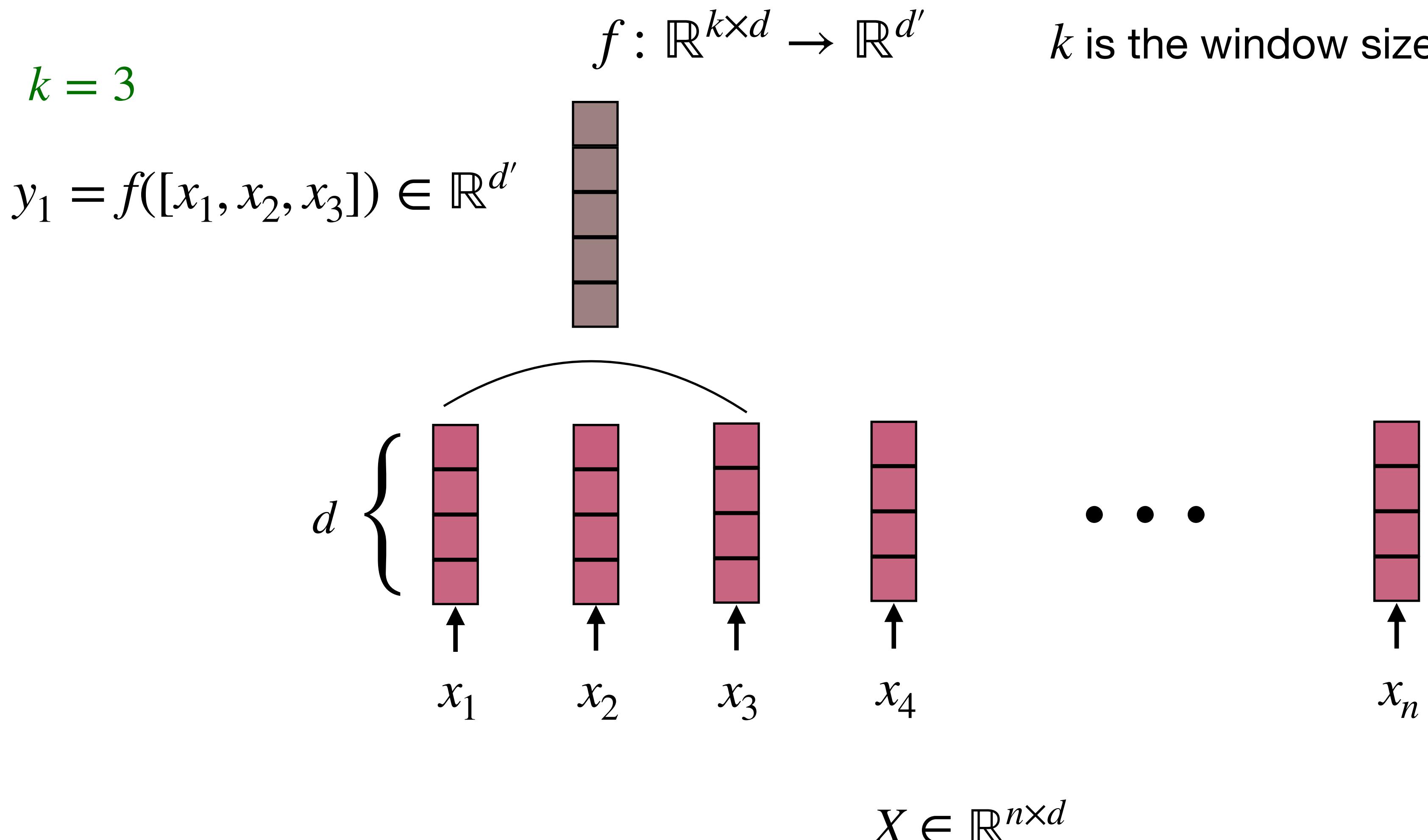
shape of the parameter  $W$  is  $\mathbb{R}^{k \times d \times d'}$



$$X \in \mathbb{R}^{n \times d}$$

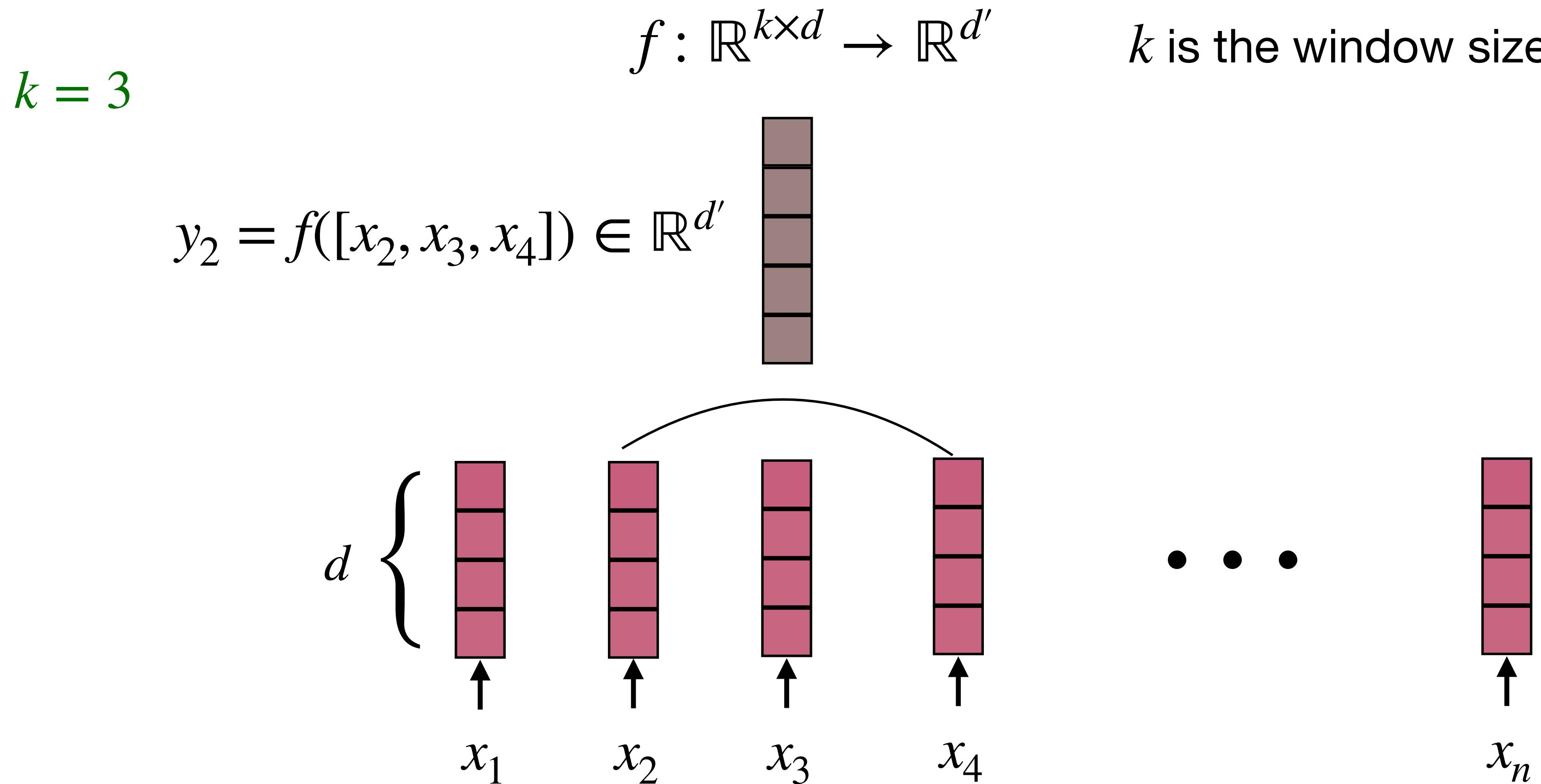
# Convolutional Neural Networks (CNNs)

- Basic Idea: only model a segment of input with **fixed window-size**



# Convolutional Neural Networks (CNNs)

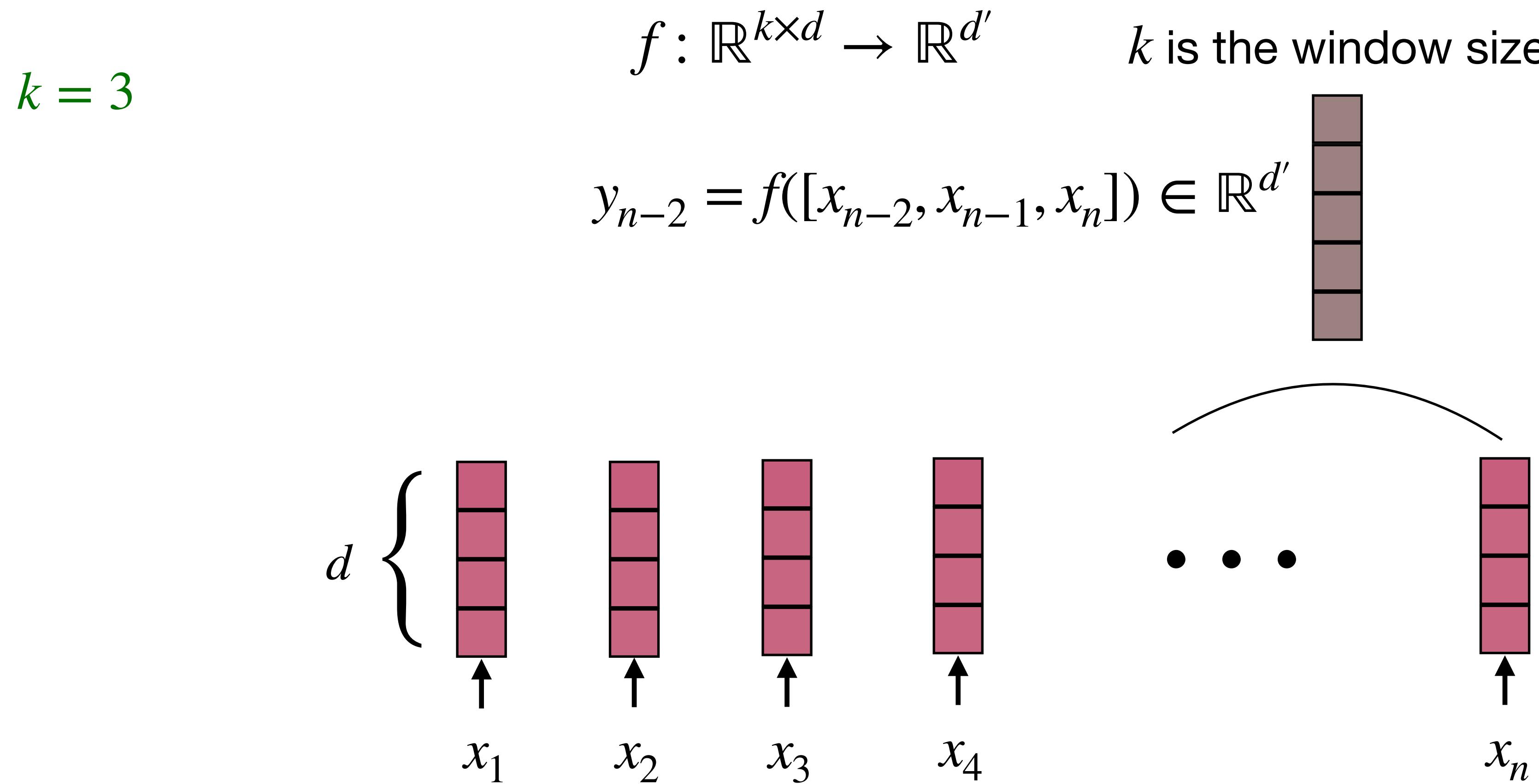
- Basic Idea: only model a segment of input with **fixed window-size**



$$X \in \mathbb{R}^{n \times d}$$

# Convolutional Neural Networks (CNNs)

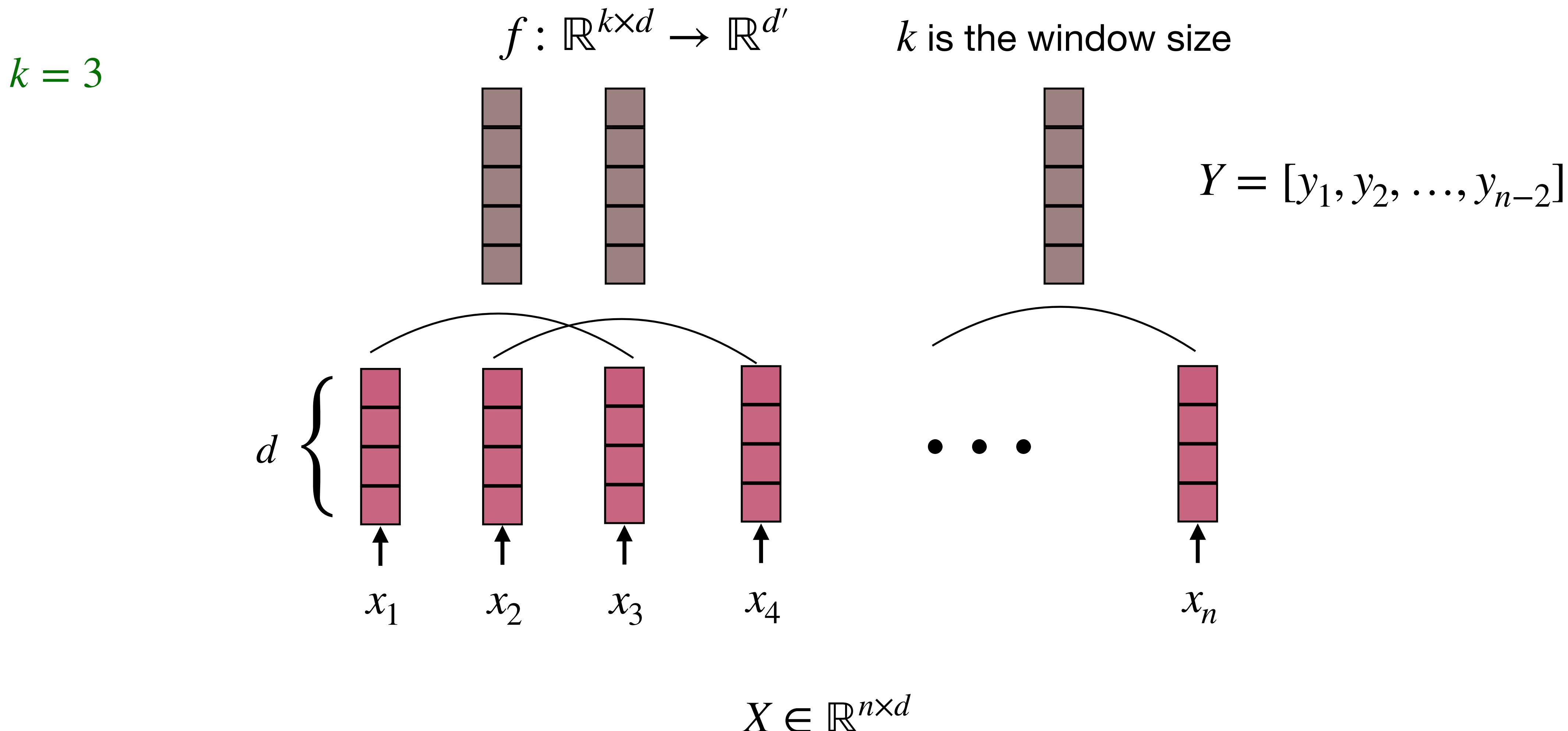
- Basic Idea: only model a segment of input with **fixed window-size**



$$X \in \mathbb{R}^{n \times d}$$

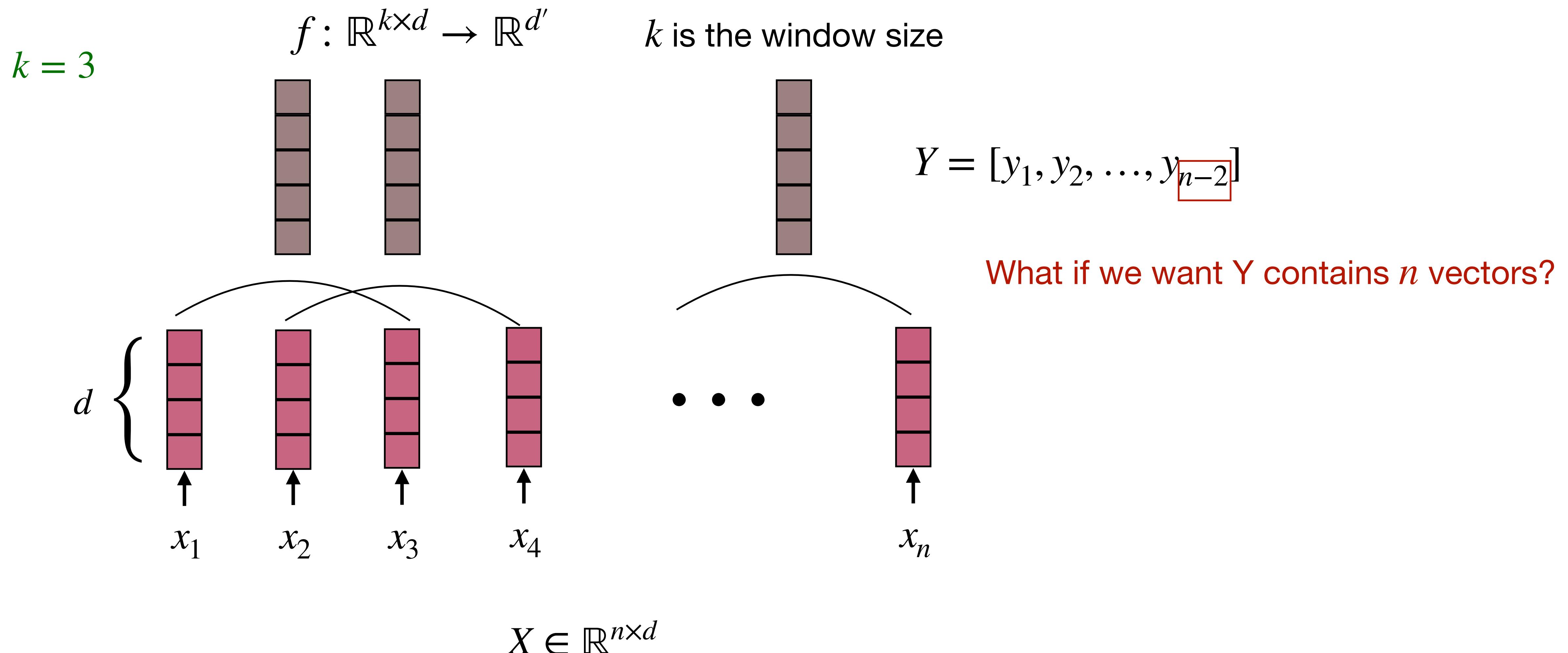
# Convolutional Neural Networks (CNNs)

- Basic Idea: only model a segment of input with **fixed window-size**



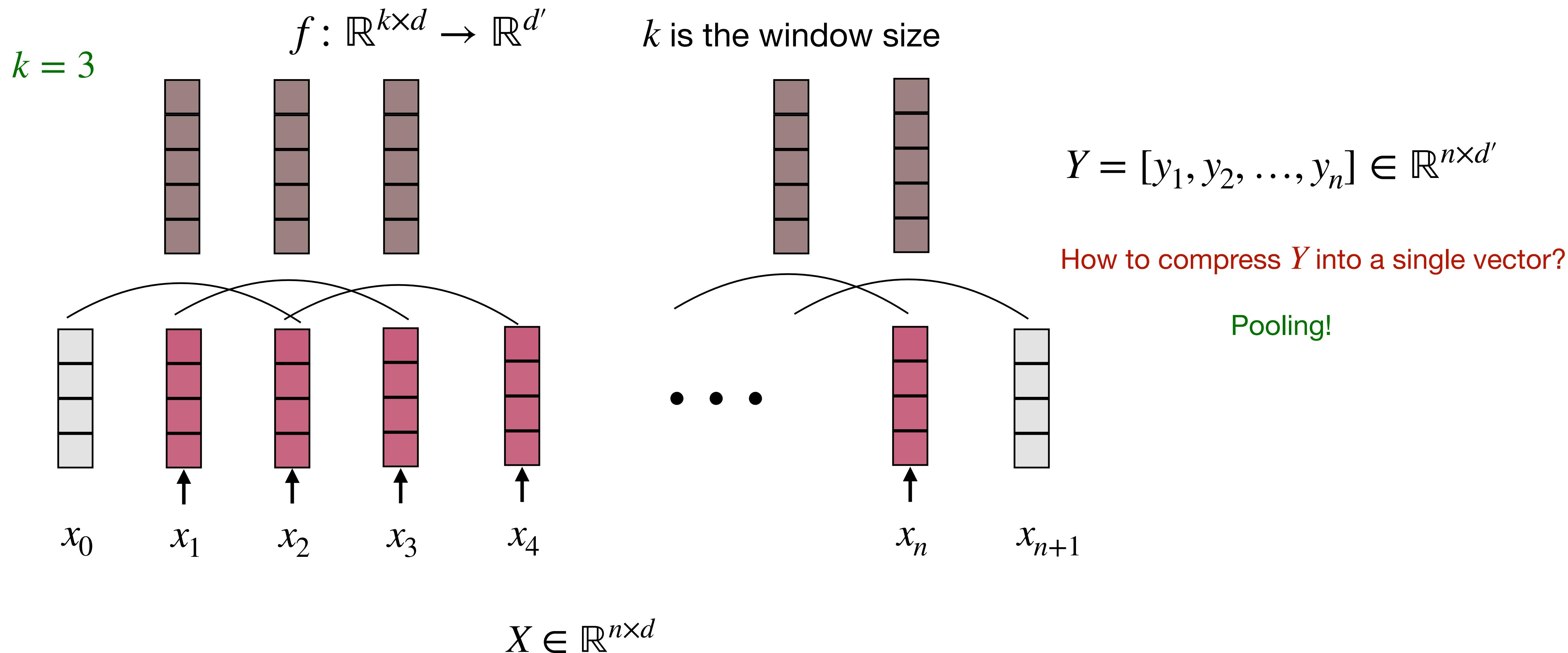
# Convolutional Neural Networks (CNNs)

- Basic Idea: only model a segment of input with **fixed window-size**



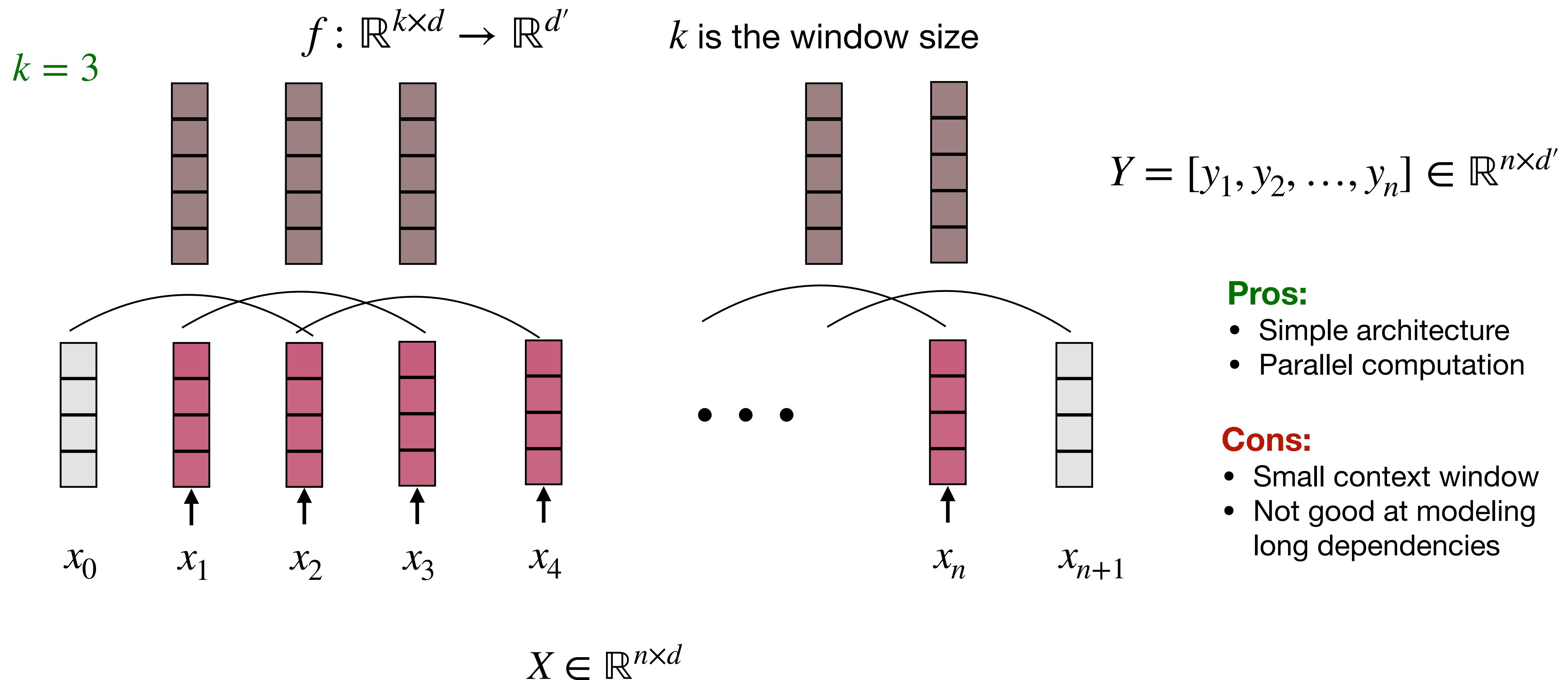
# Convolutional Neural Networks (CNNs)

- Basic Idea: only model a segment of input with **fixed window-size**



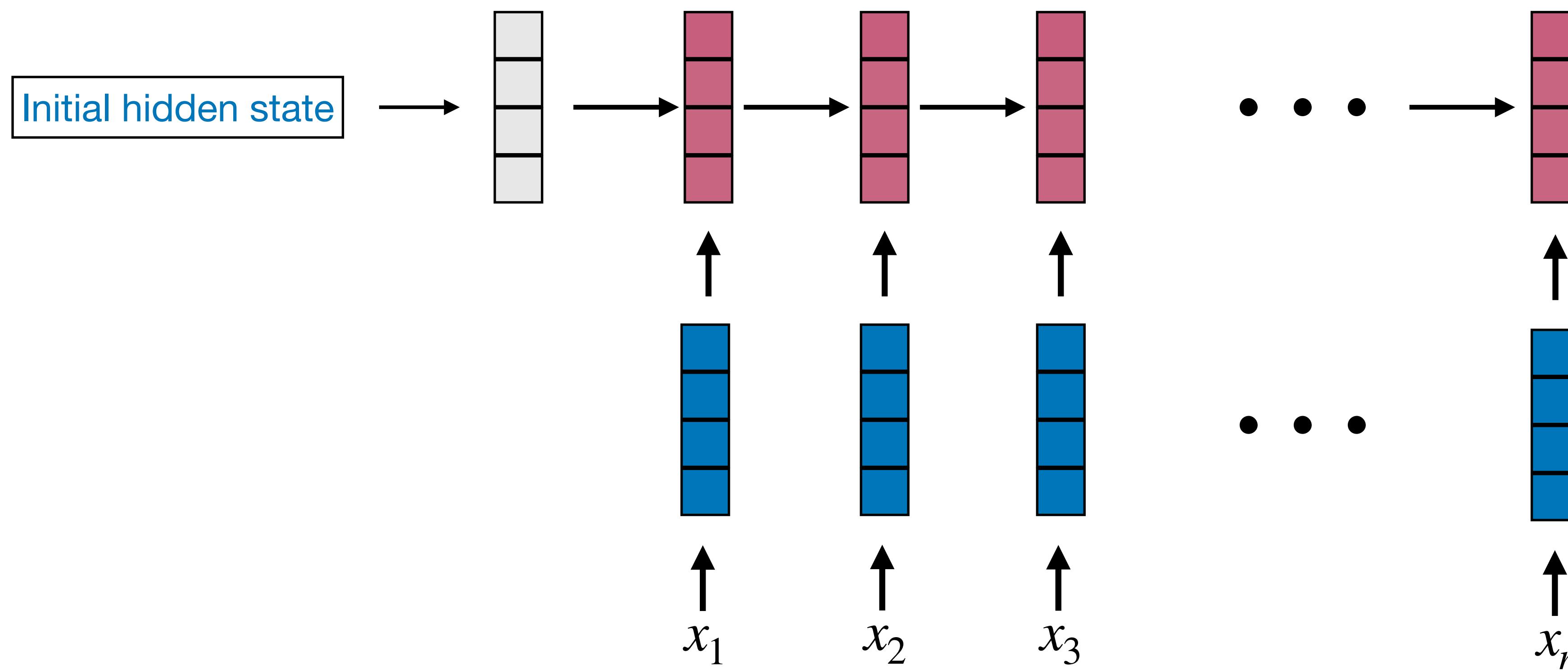
# Convolutional Neural Networks (CNNs)

- Basic Idea: only model a segment of input with **fixed window-size**



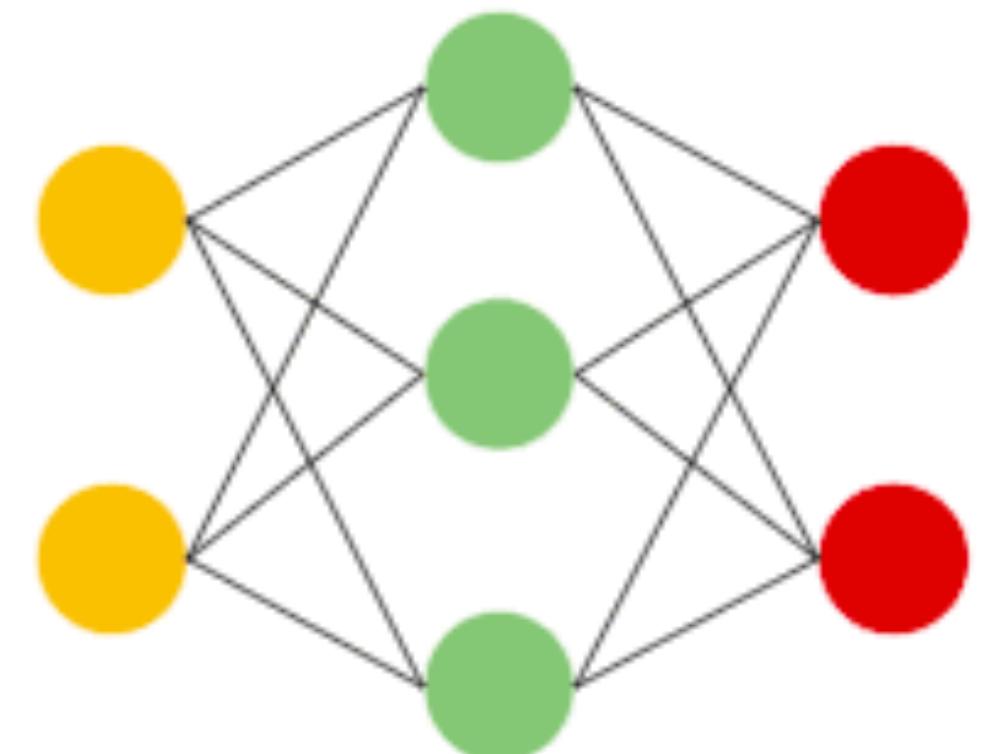
# Recurrent Neural Networks

- Recurrent Neural Networks (RNNs)
  - Re-using one feed forward network in a recurrent way

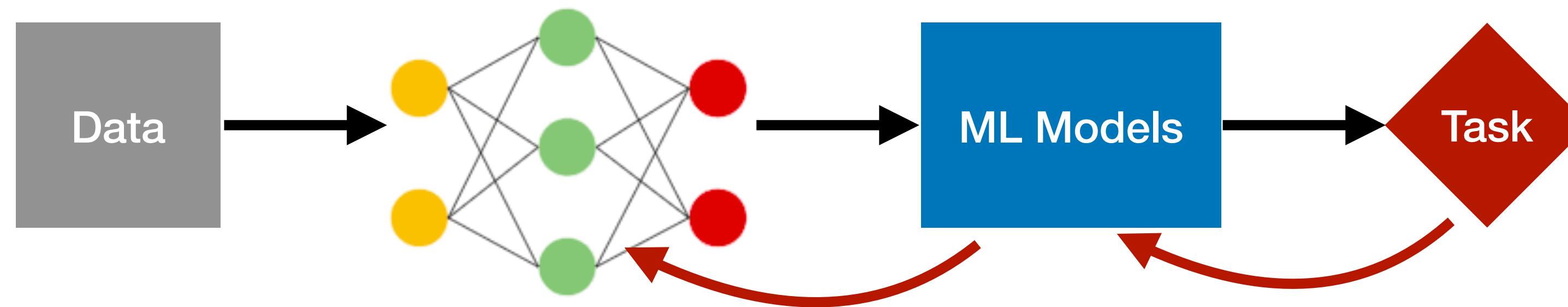


We leave RNN to the next lecture

# How to learn a neural network



# Deep Learning System



Automatically learning feature representations for the end task

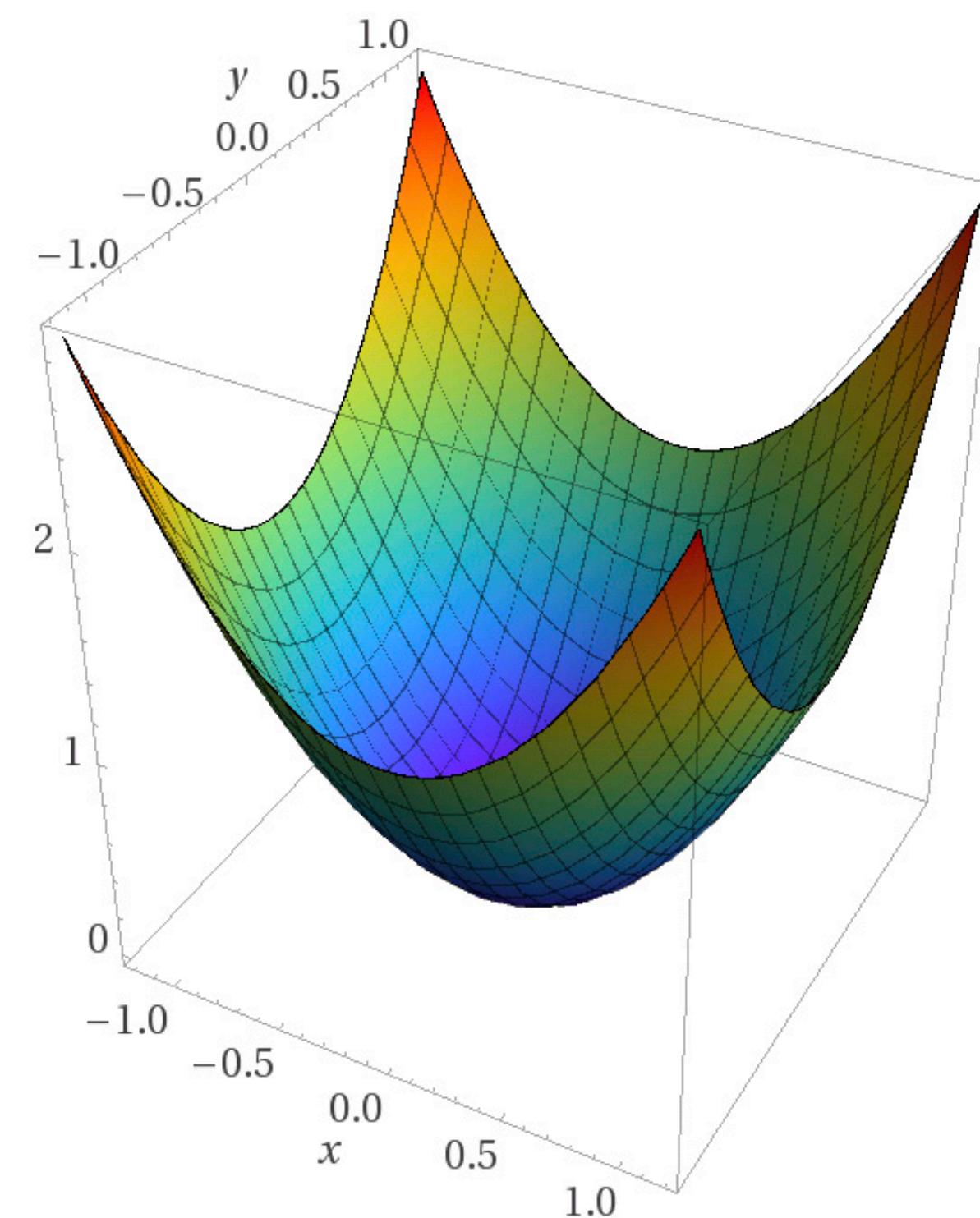
Deep Learning a.k.a Representation Learning

# Training a DNN

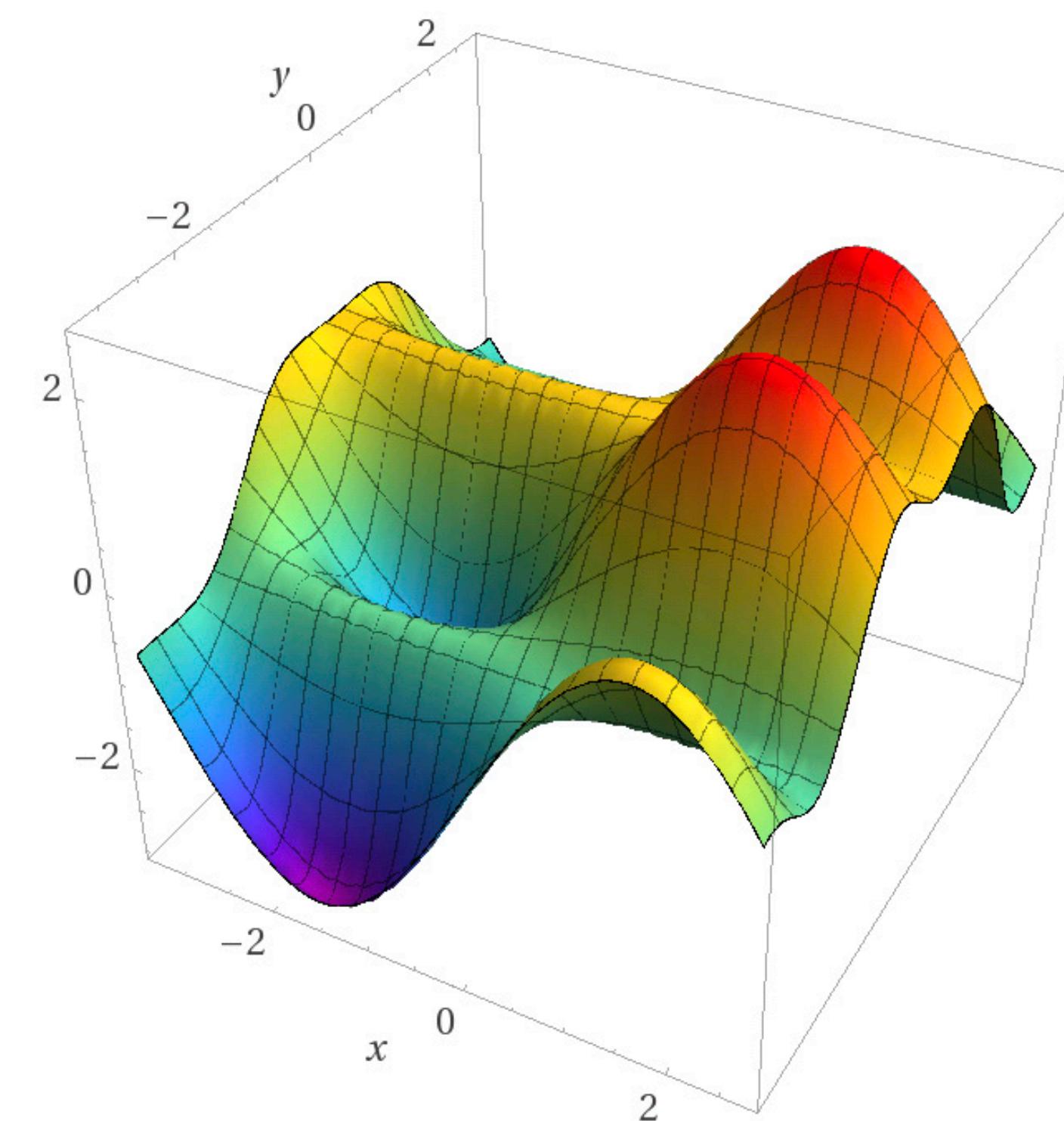
- **Algorithm:** gradient descent in a mini-batch
  - Batch size is a hyper-parameter
- **Gradient calculation:** back-propagation
  - $W \leftarrow W - \gamma \nabla_W L(X)$
- **Initialization of parameters**
  - **Very important!**
  - Different layers may have different initialization strategy
    - A commonly used one for FFN:  $W \in \mathbb{R}^{d \times d'}, W \sim N(0, \frac{1}{\sqrt{d}})$

# Training a Deep Neural Network is Challenging

- DNNs are non-convex



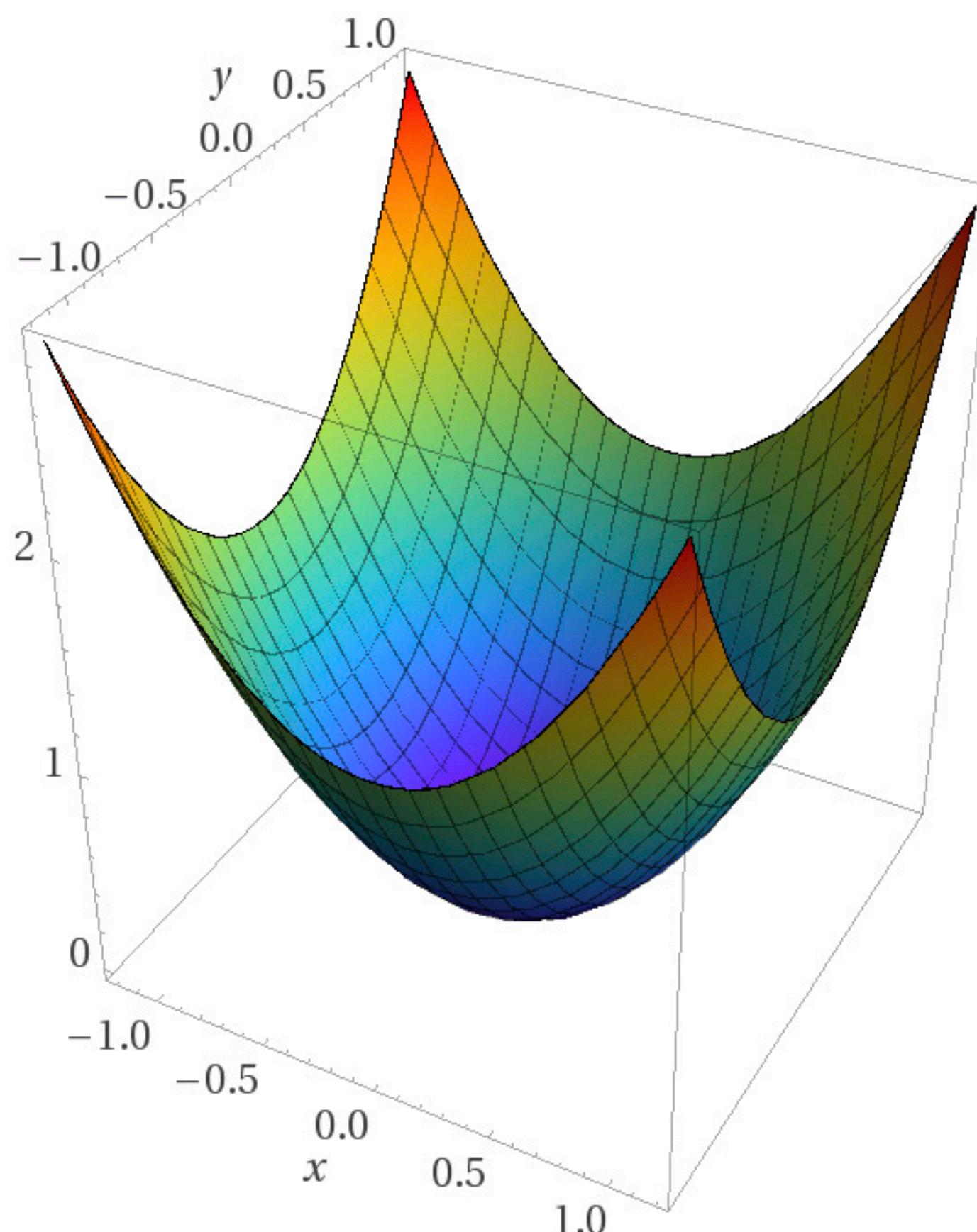
Computed by Wolfram|Alpha



Computed by Wolfram|Alpha

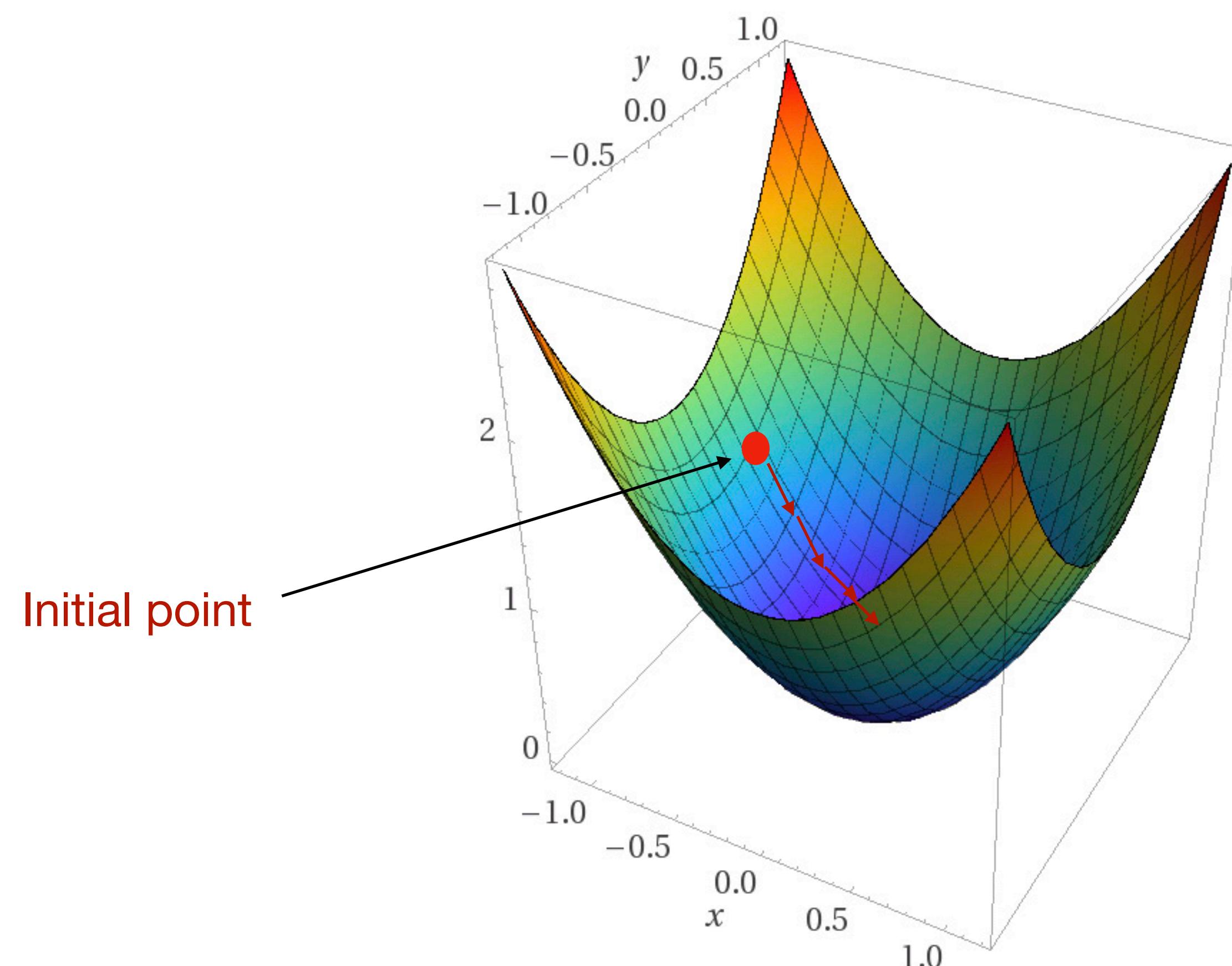
# Training a Deep Neural Network is Challenging

- Convex optimization is relatively easy
  - Only one optimal point, which is the global optimum
  - Initialization only impacts converging speed but not convergence point



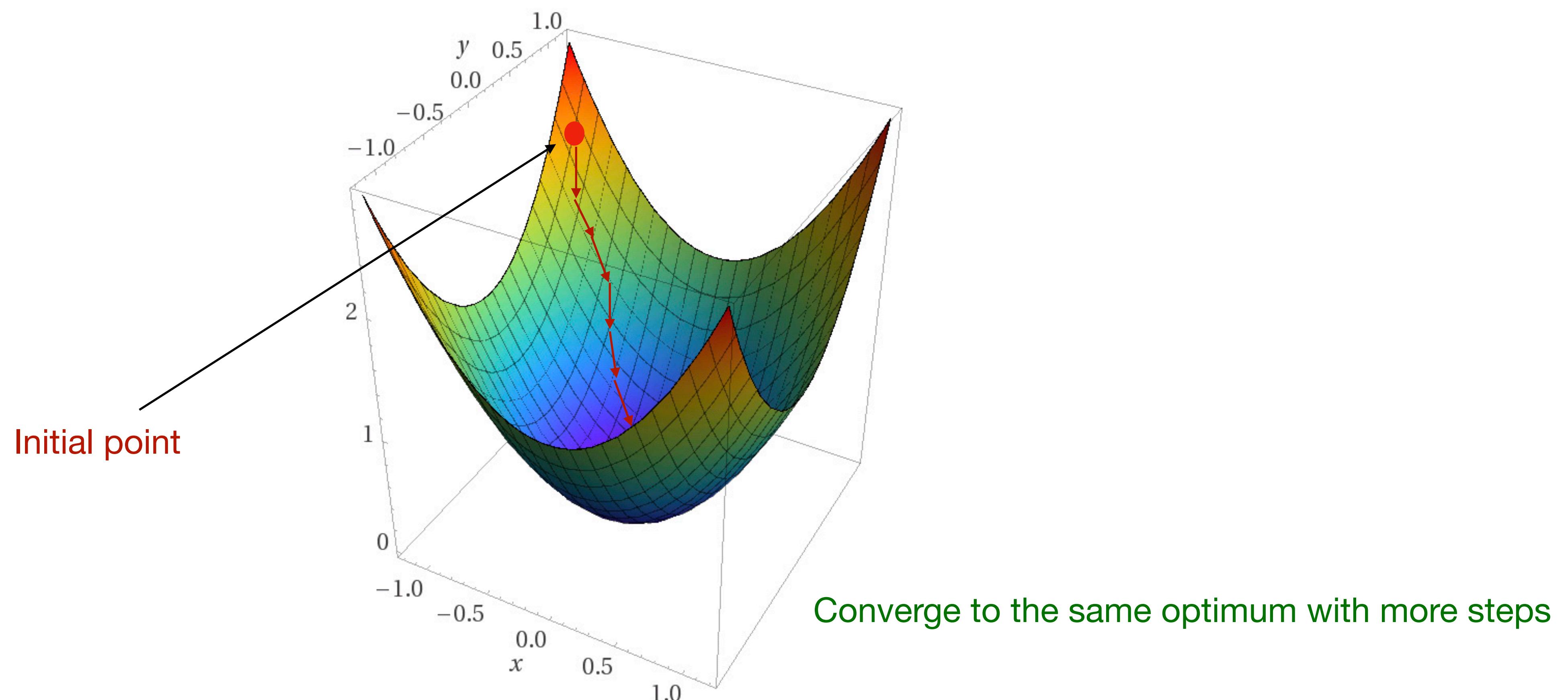
# Training a Deep Neural Network is Challenging

- Convex optimization is relatively easy
  - Only one optimal point, which is the global optimum
  - Initialization only impacts converging speed but not convergence point



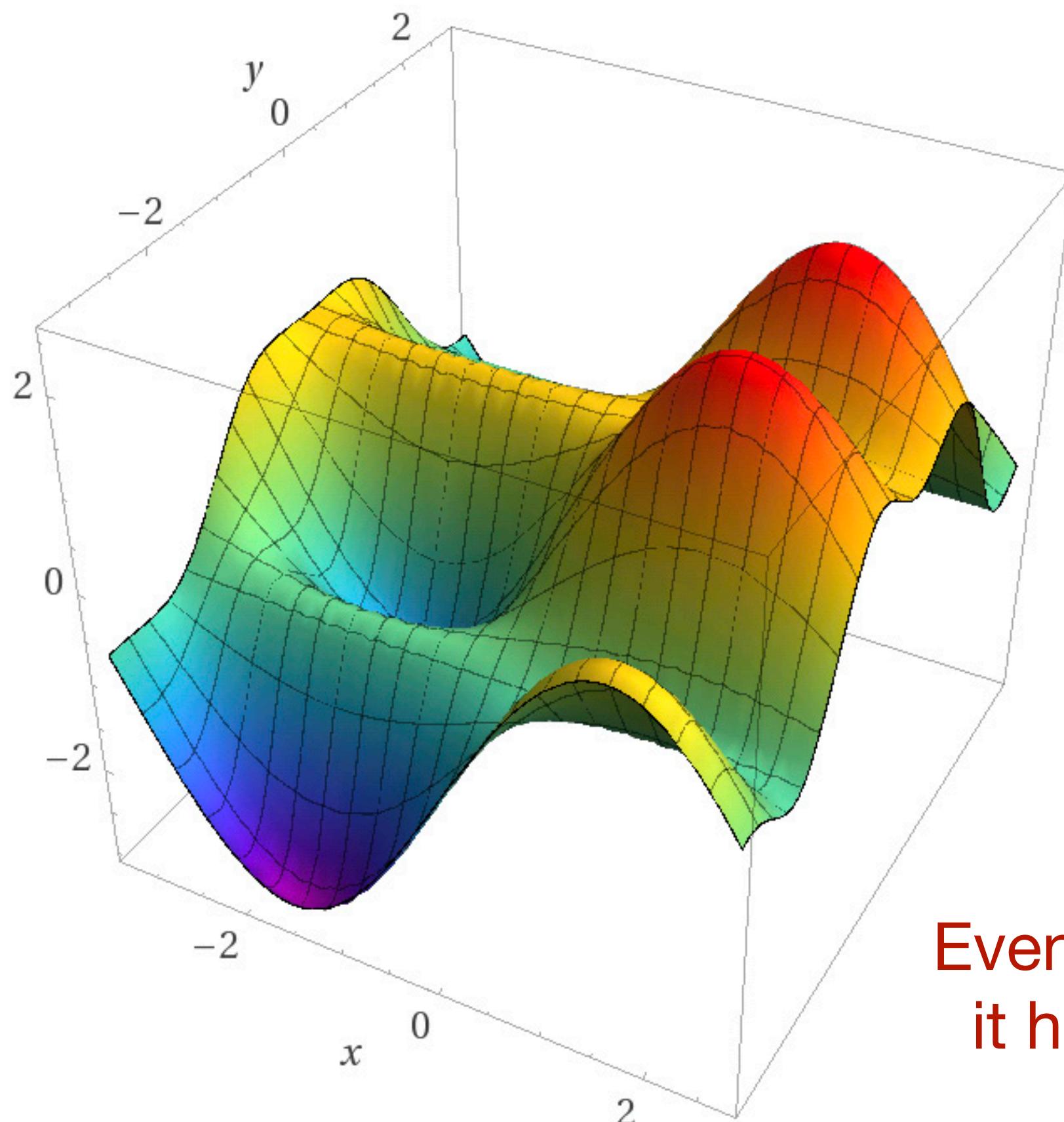
# Training a Deep Neural Network is Challenging

- Convex optimization is relatively easy
  - Only one optimal point, which is the global optimum
  - Initialization only impacts converging speed but not convergence point



# Training a Deep Neural Network is Challenging

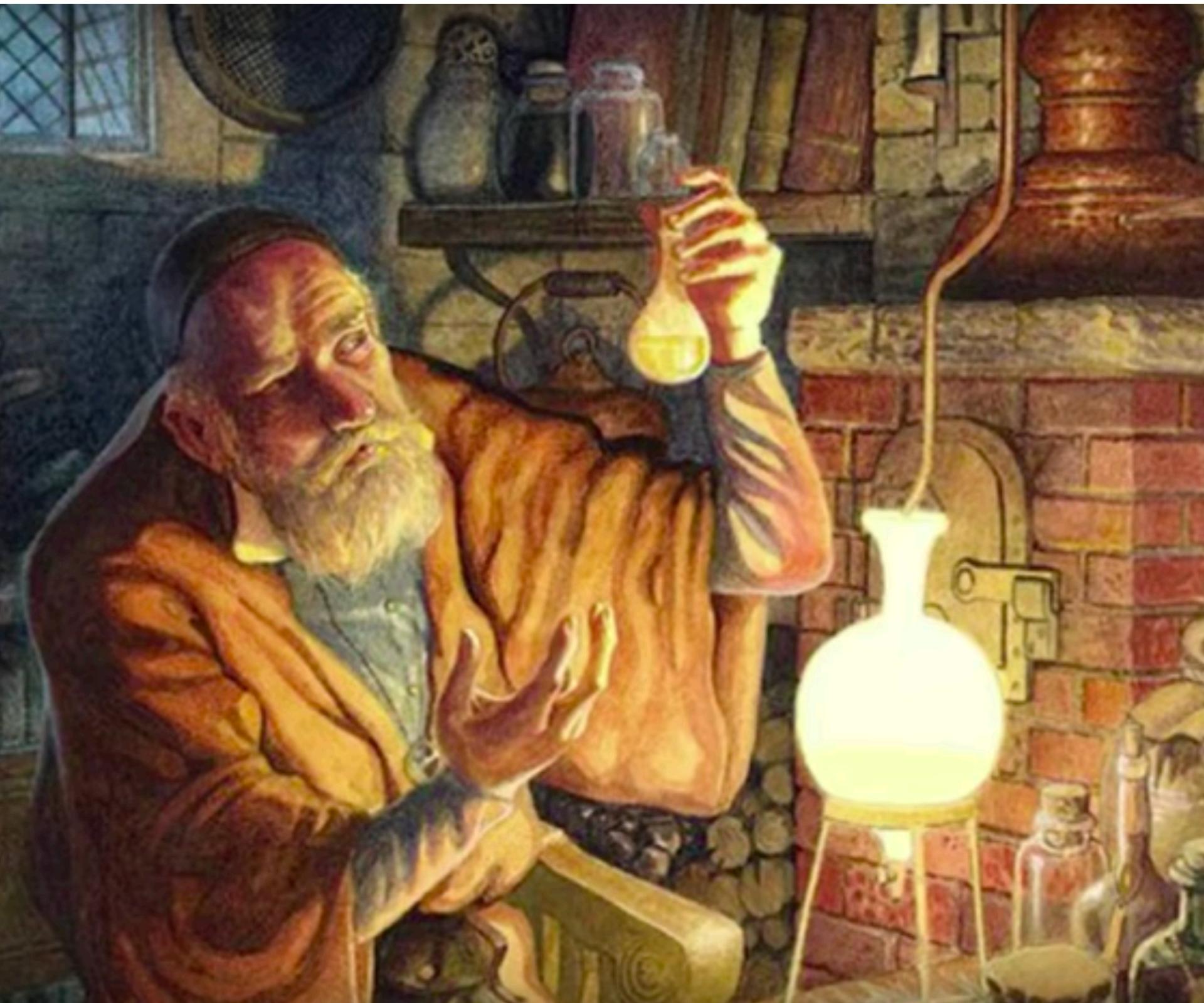
- Non-Convex optimization is extremely hard
  - Multiple local optimal points
  - Initialization impacts the final convergence point



Even the most powerful model, we are not sure if it has converged to the global optimum or not!

# Before Building Your DNNs

- DNNs are non-convex
- No rigorous theoretical supports, but only empirical evidences



Rahimi@NIPS2017:  
Has Deep Learning Become Alchemy?

炼丹

# Back-Propagation

- One sentence to describe back-propagation
  - The chain rule of differential/gradient

$$y = f(x) = f(g_1(x), g_2(x), \dots, g_K(x))$$

$$\nabla_x y = \sum_{j=1}^K \nabla_{g_j} y \cdot \nabla_x g_j(x)$$

**Back-Propagation:** first derive the gradient of each layer (or even each operation), then apply chain rule to compute the gradient of the whole network

# Back-Propagation

- A simple example:

$$y = (x + 1) \times 2$$

A computational graph showing the function  $y = (x + 1) \times 2$ . The output  $y$  is at the top. Two arrows point to it from below: one labeled '+ operation' pointing to the addition node  $x + 1$ , and another labeled 'x operation' pointing to the multiplication node  $\times 2$ .

PyTorch implements gradient calculation for the **whole set** of commonly used operations

# Results

## Movie Review

## Sentiment Analysis

SVM

77.3

82.9

Tree-CRF

77.7

85.4

CNNs (2-layers)

**81.5**

**87.2**

# **Q&A**