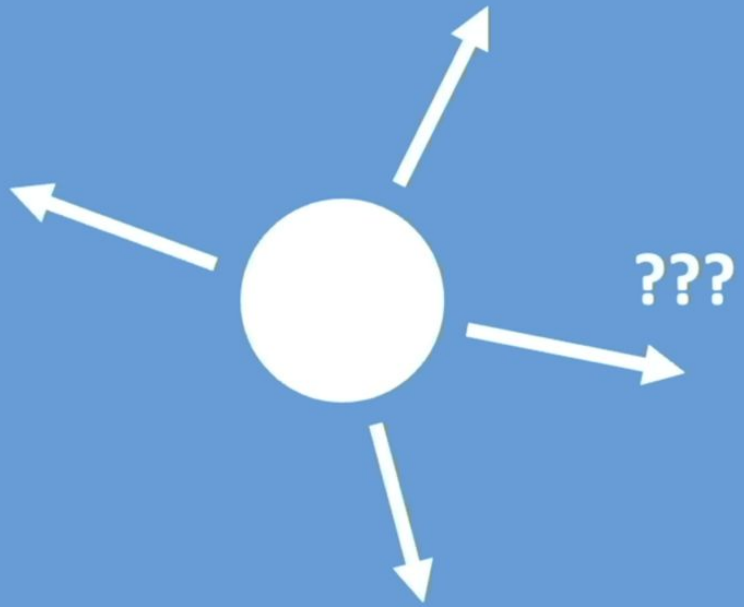
Моделювання послідовностей в глибокому навчанні

План

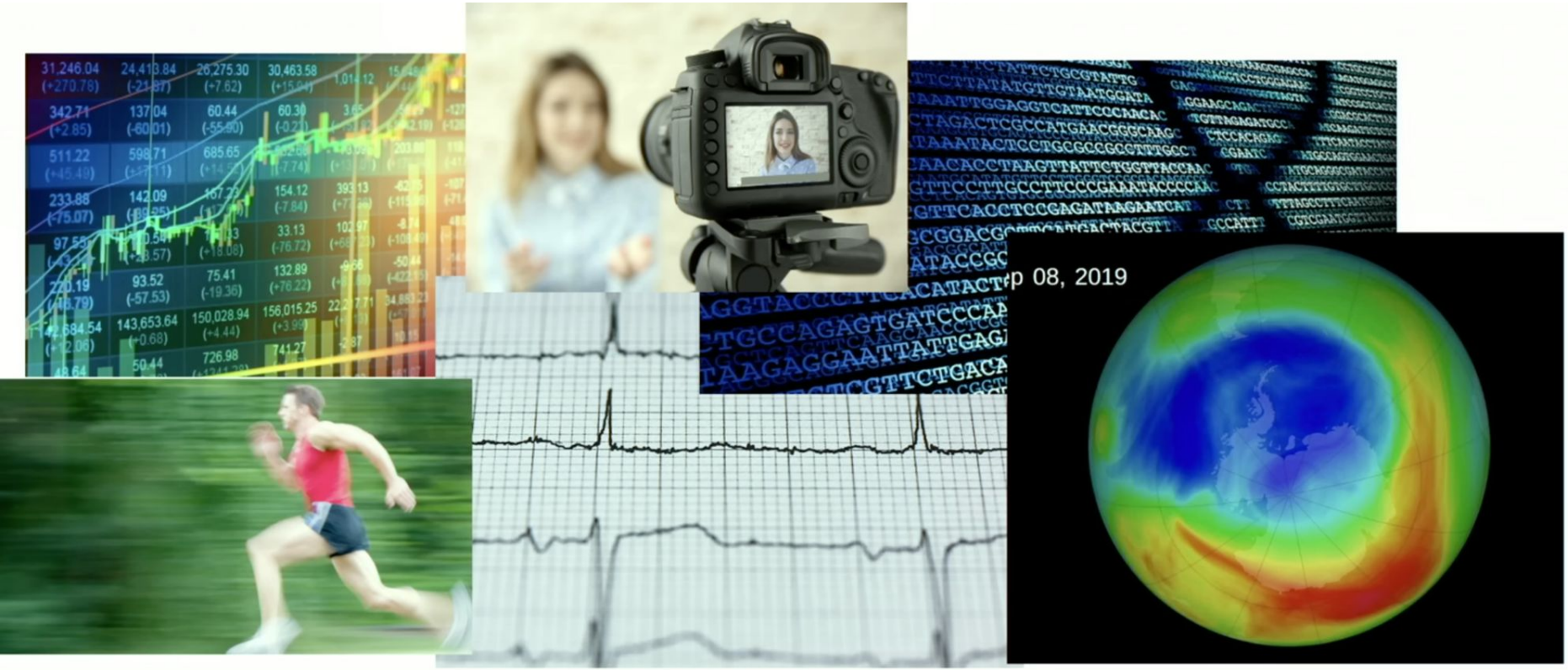
- Вступ до моделювання послідовностей. Принцип роботи рекурентних нейронних мереж
- Математика для побудови RNN
- Демо розв’язку NLP-задачі
- Attention механізм і трансформери

Маючи картинку кульки, чи можемо ми передбачити, куди вона піде далі?

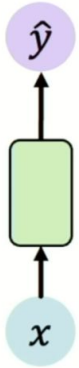


Маючи картинку кульки, чи можемо ми передбачити, куди вона піде далі?

Різноманітність послідовностей в світі

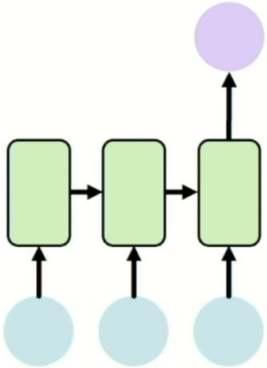


Прикладне використання sequence modeling

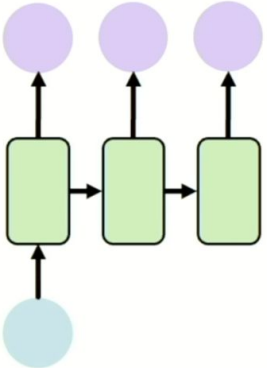


One to One
Binary Classification

"Will I pass this class?"
Student → Pass?

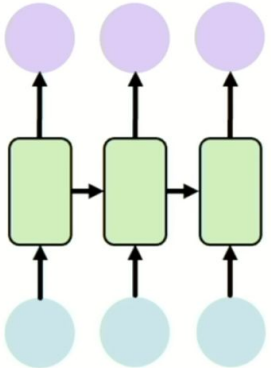


Many to One
Sentiment Classification



One to Many
Image Captioning

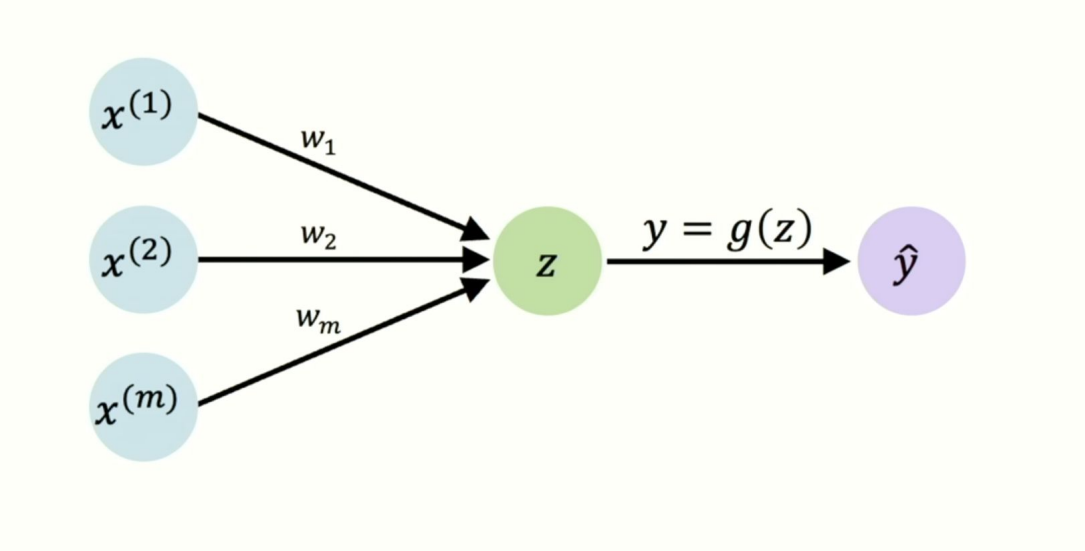
"A baseball player throws a ball."



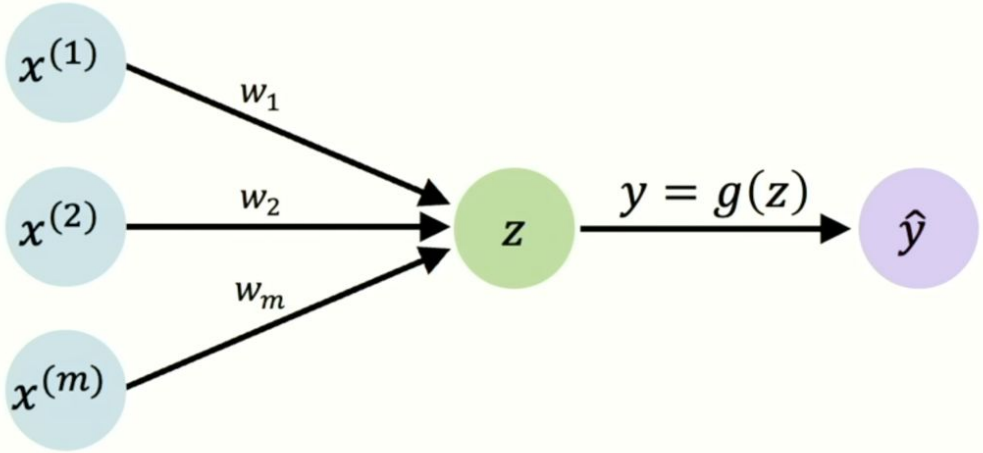
Many to Many
Machine Translation

Рекурентна робота з нейронами

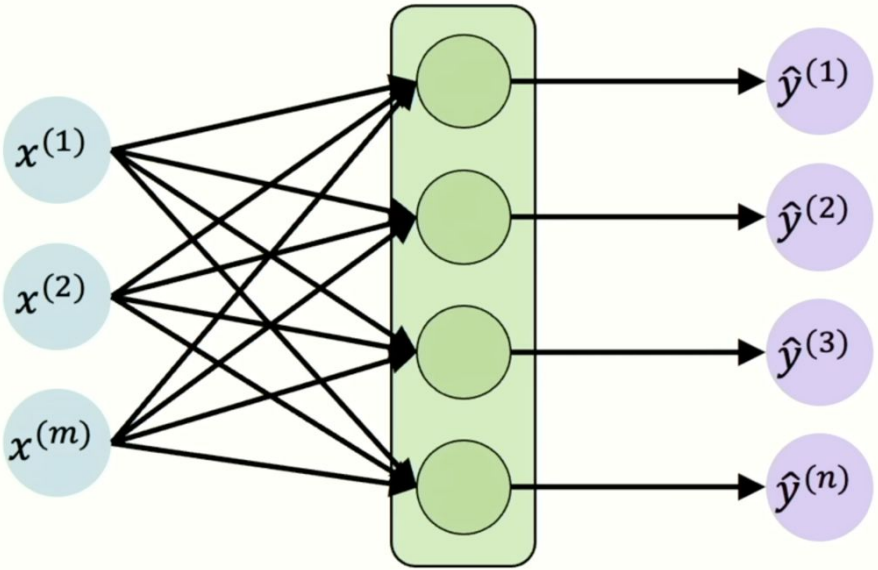
Нагадаємо, як виглядає звичайний перцептрон



The Perceptron Revisited



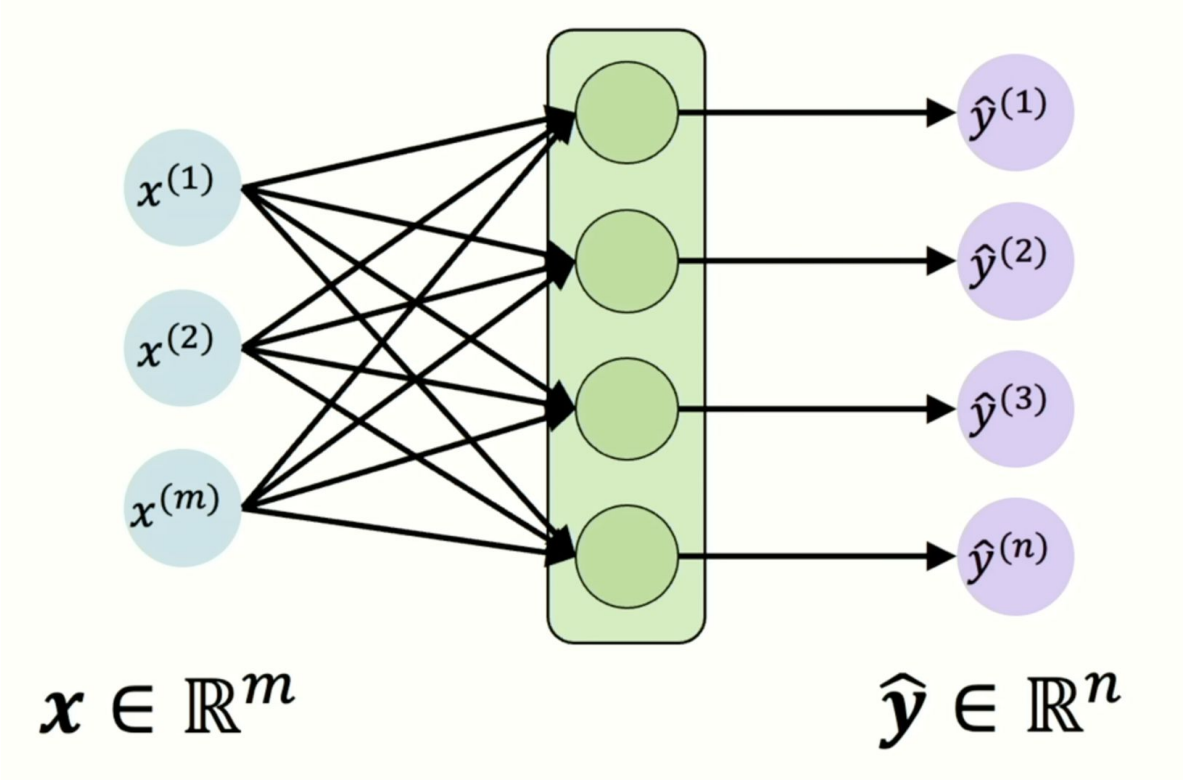
Feed-Forward Networks Revisited



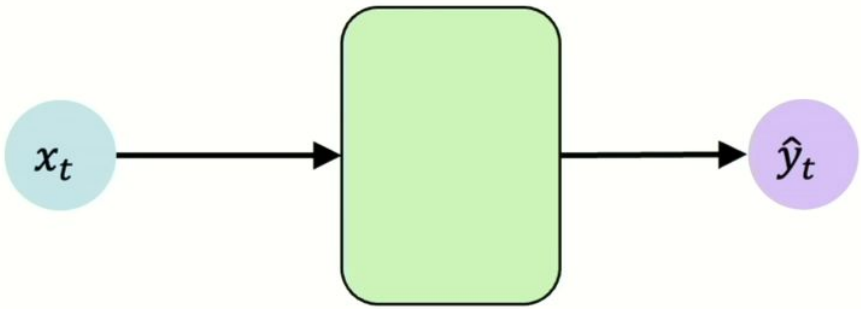
$\boldsymbol{x} \in \mathbb{R}^m$

$\hat{\boldsymbol{y}} \in \mathbb{R}^n$

Feed-Forward Neural Network



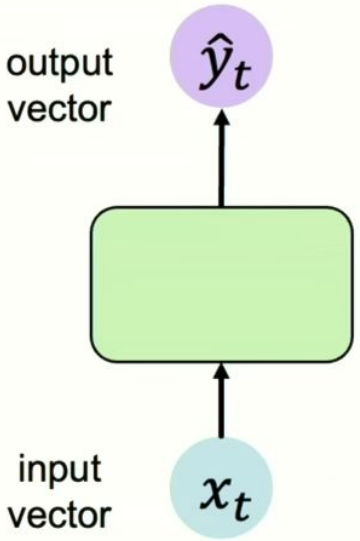
Feed-Forward Networks Revisited



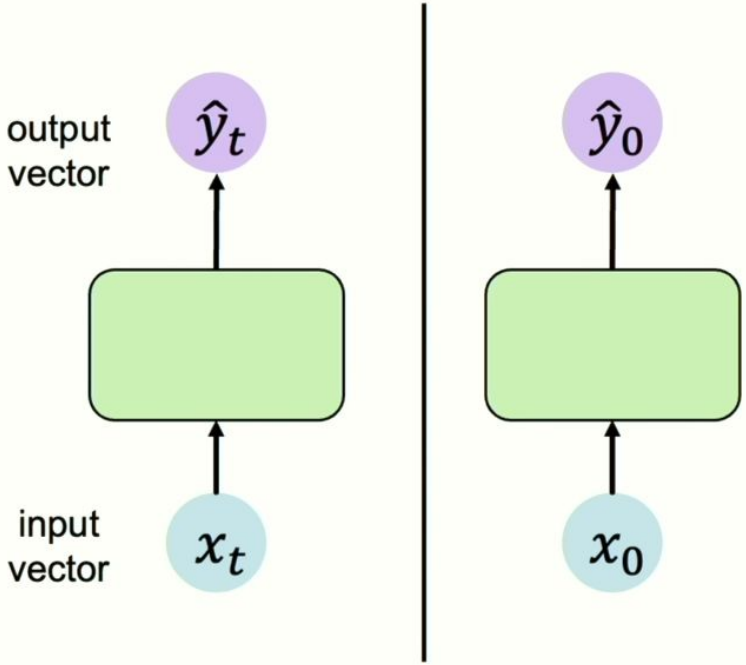
$\boldsymbol{x_t} \in \mathbb{R}^m$

$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$

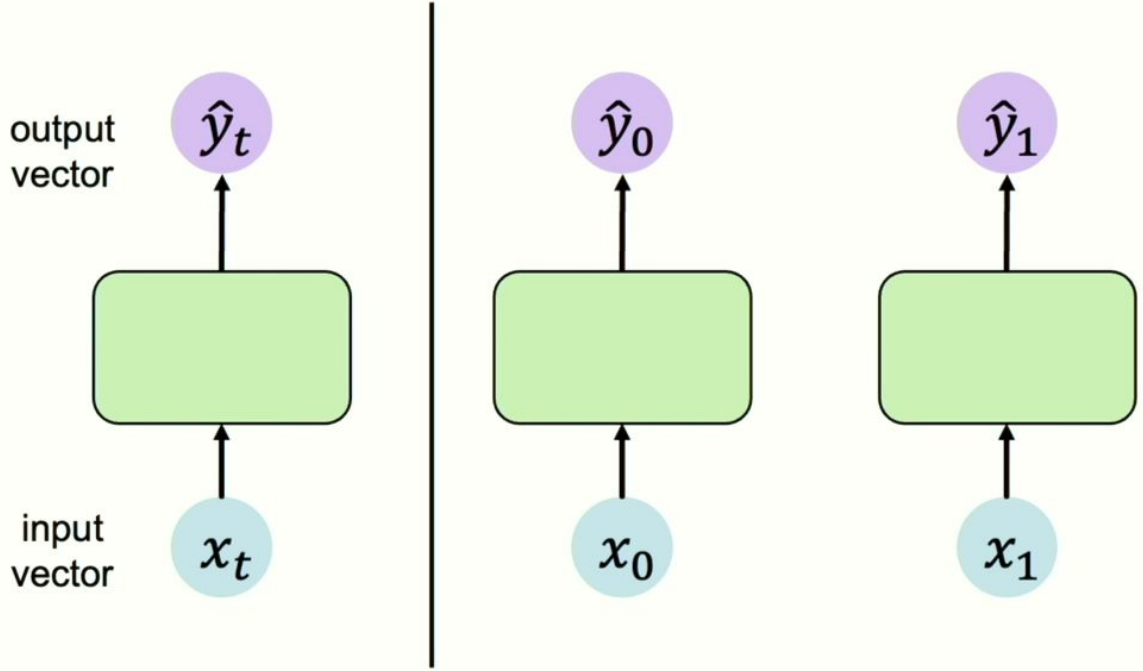
Handling Individual Time Steps



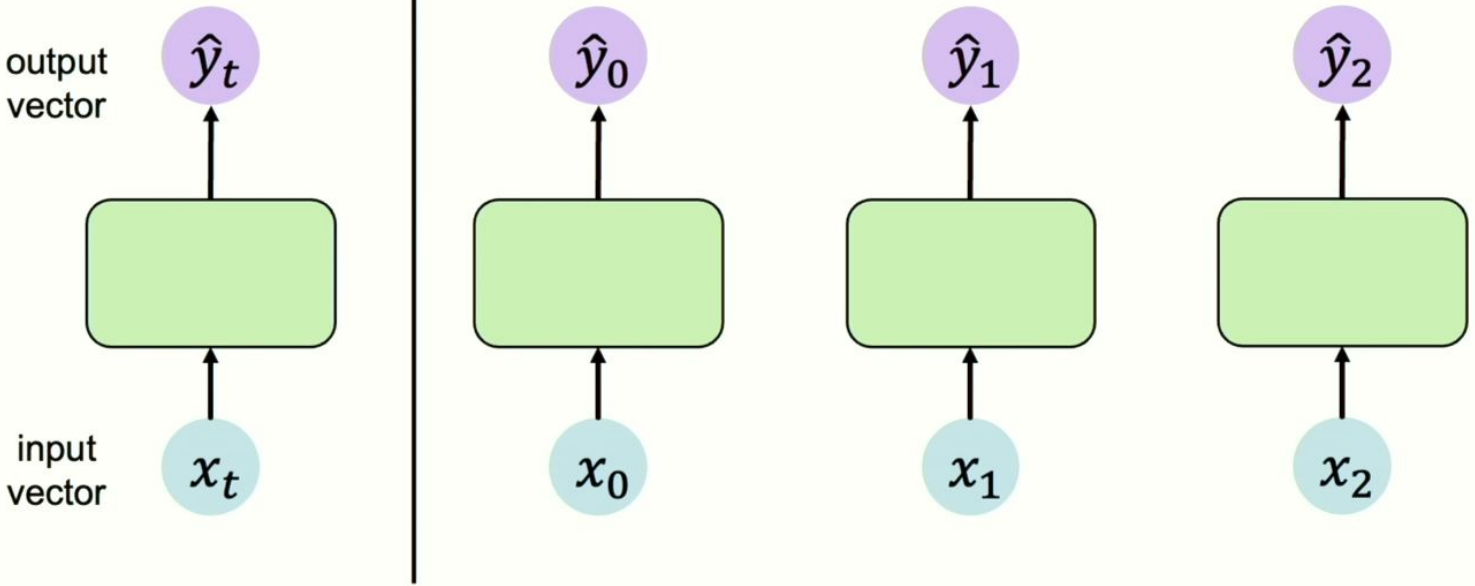
Handling Individual Time Steps



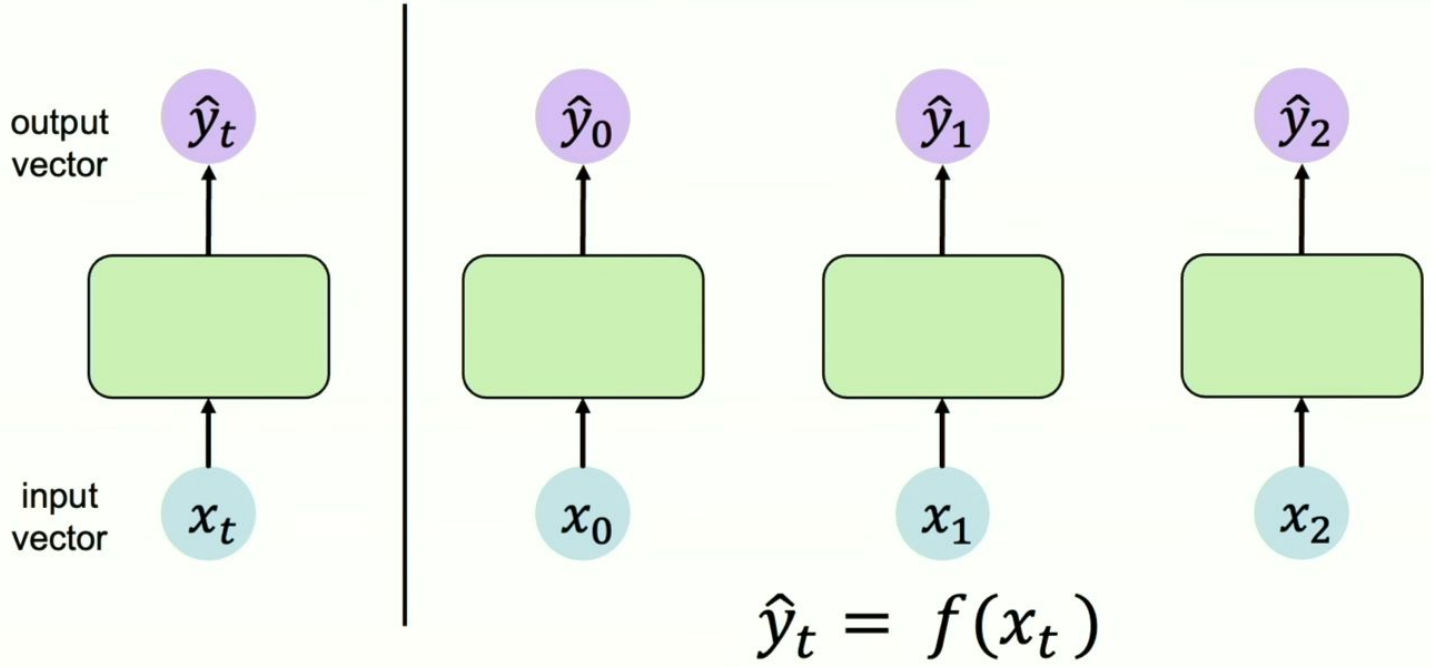
Handling Individual Time Steps



Handling Individual Time Steps

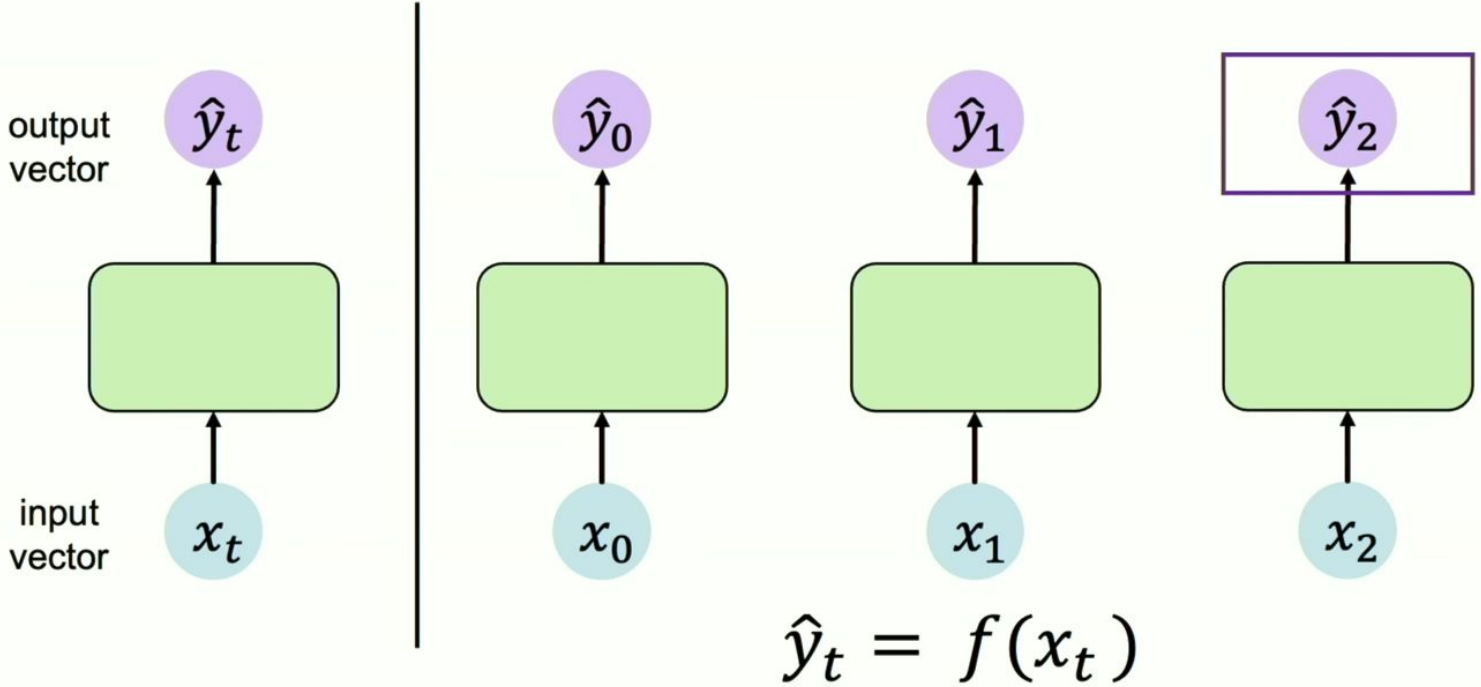


Handling Individual Time Steps



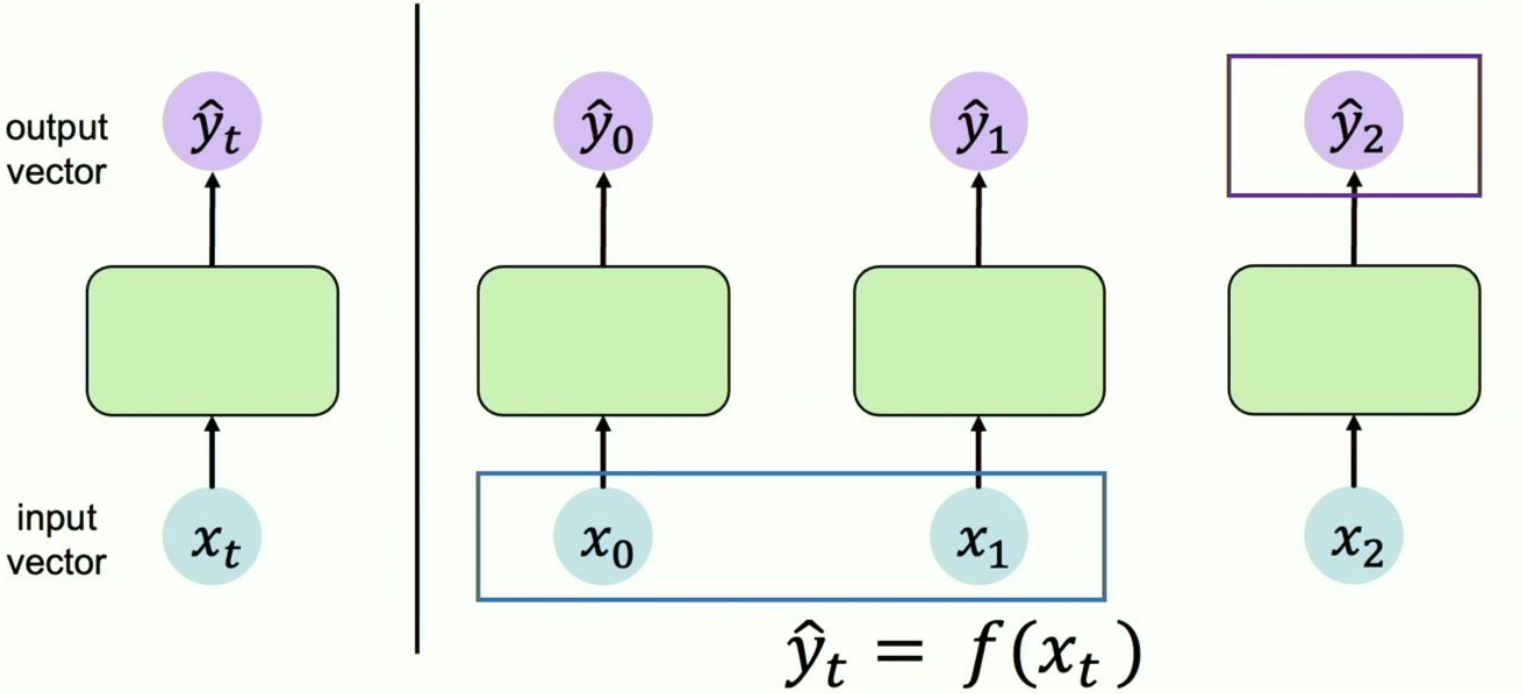
$$\hat{y}_t = f(x_t)$$

Handling Individual Time Steps

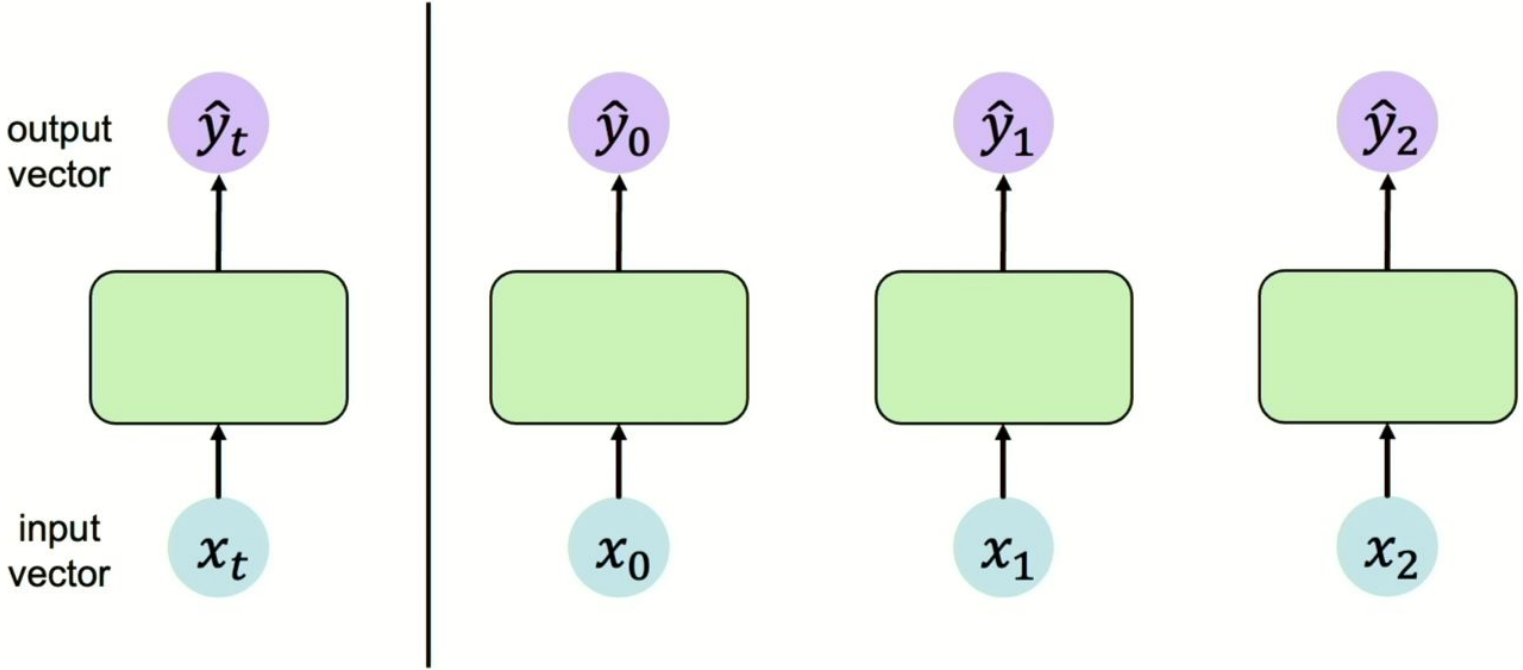


$$\hat{y}_t = f(x_t)$$

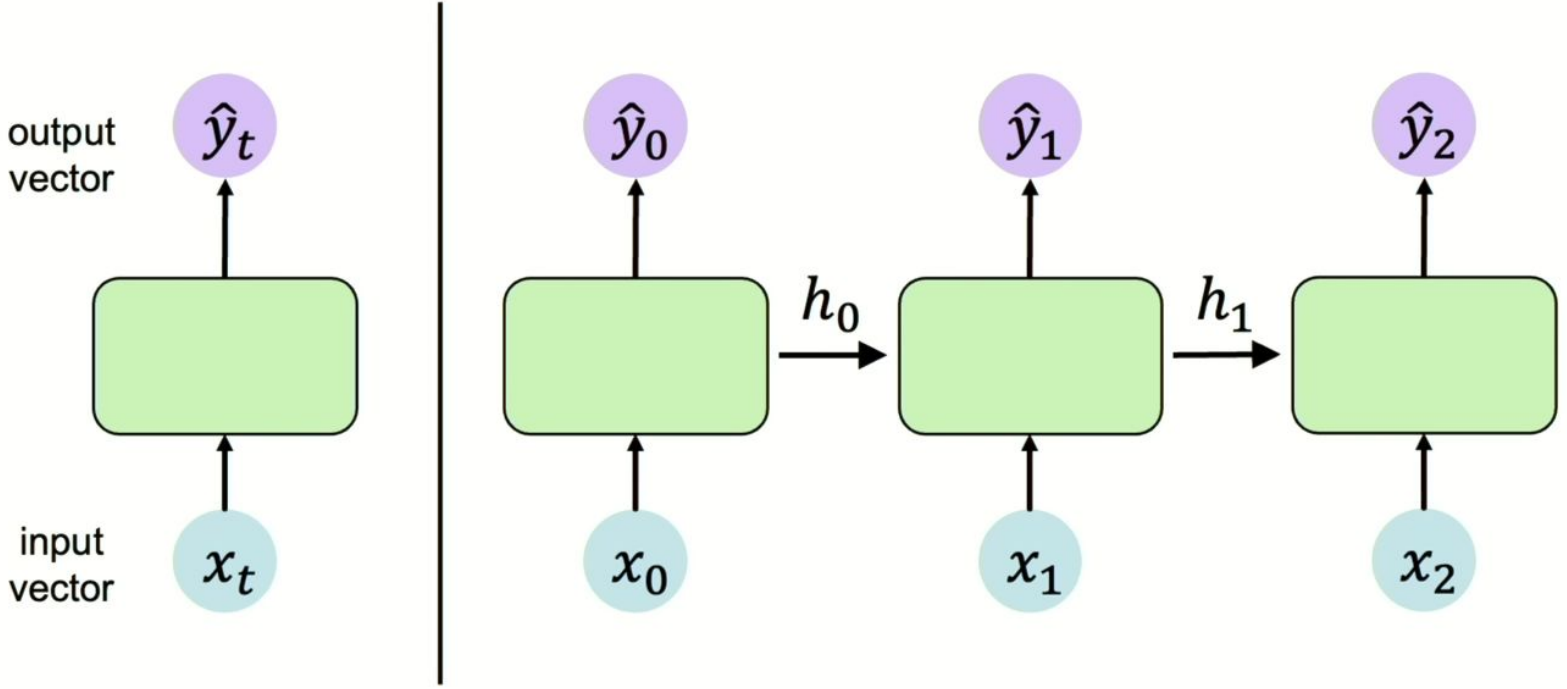
Handling Individual Time Steps



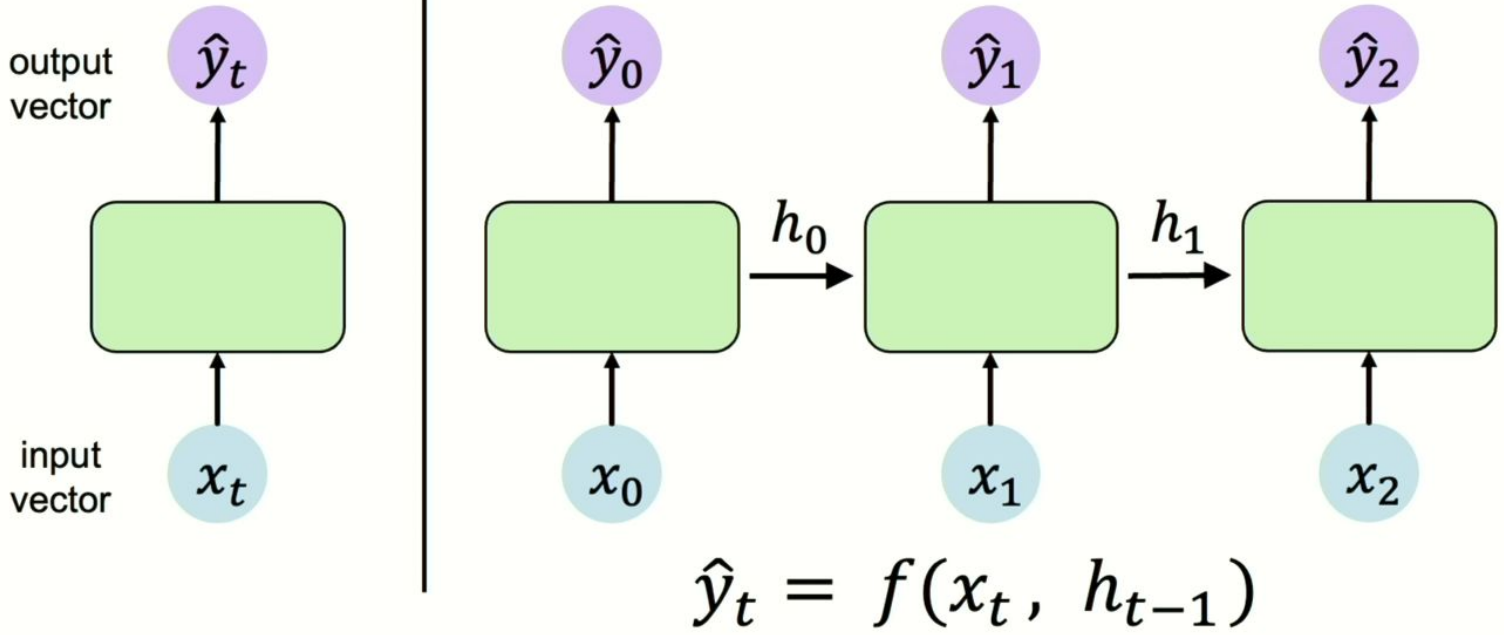
Neurons with Recurrence



Neurons with Recurrence

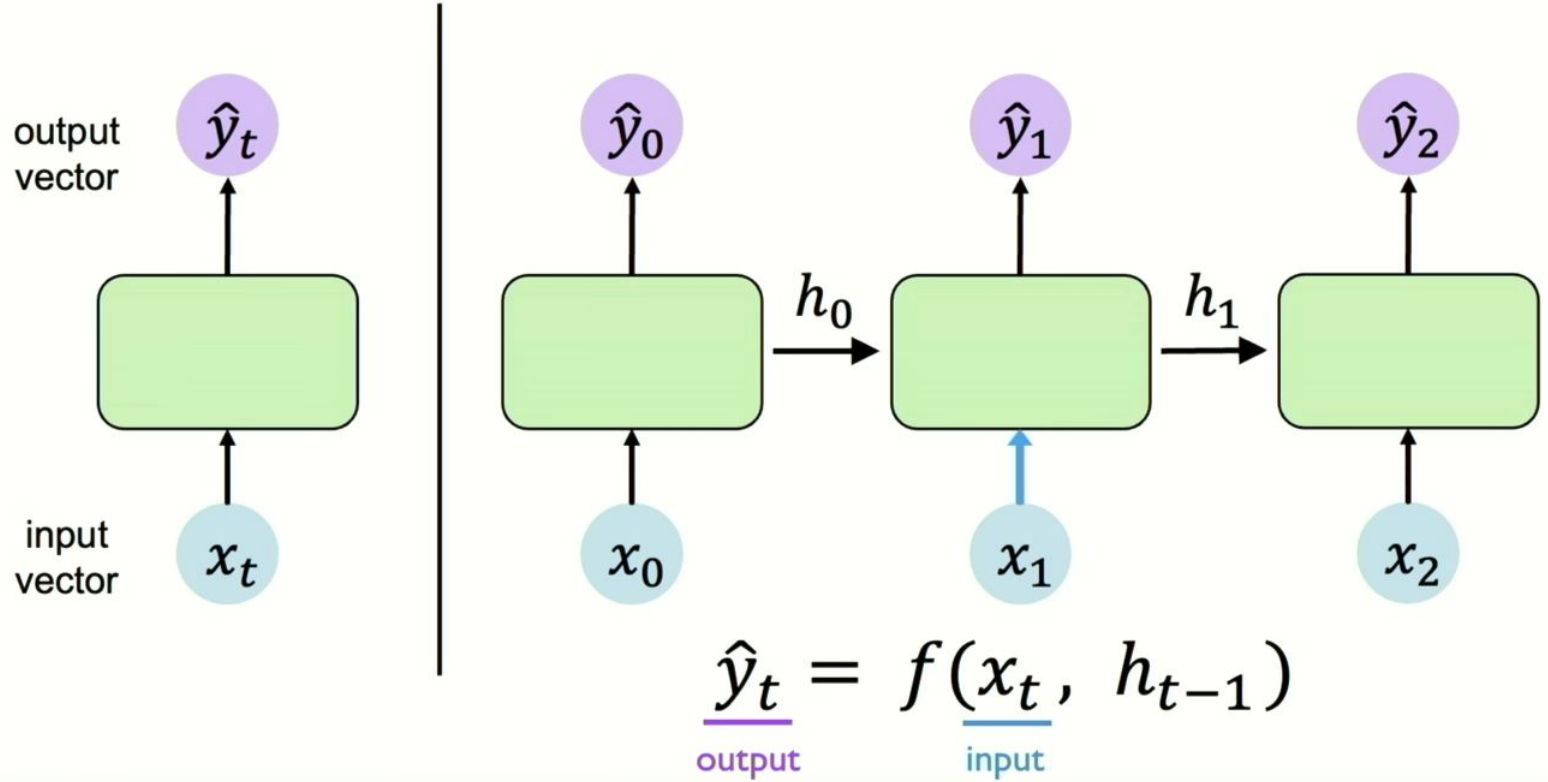


Neurons with Recurrence

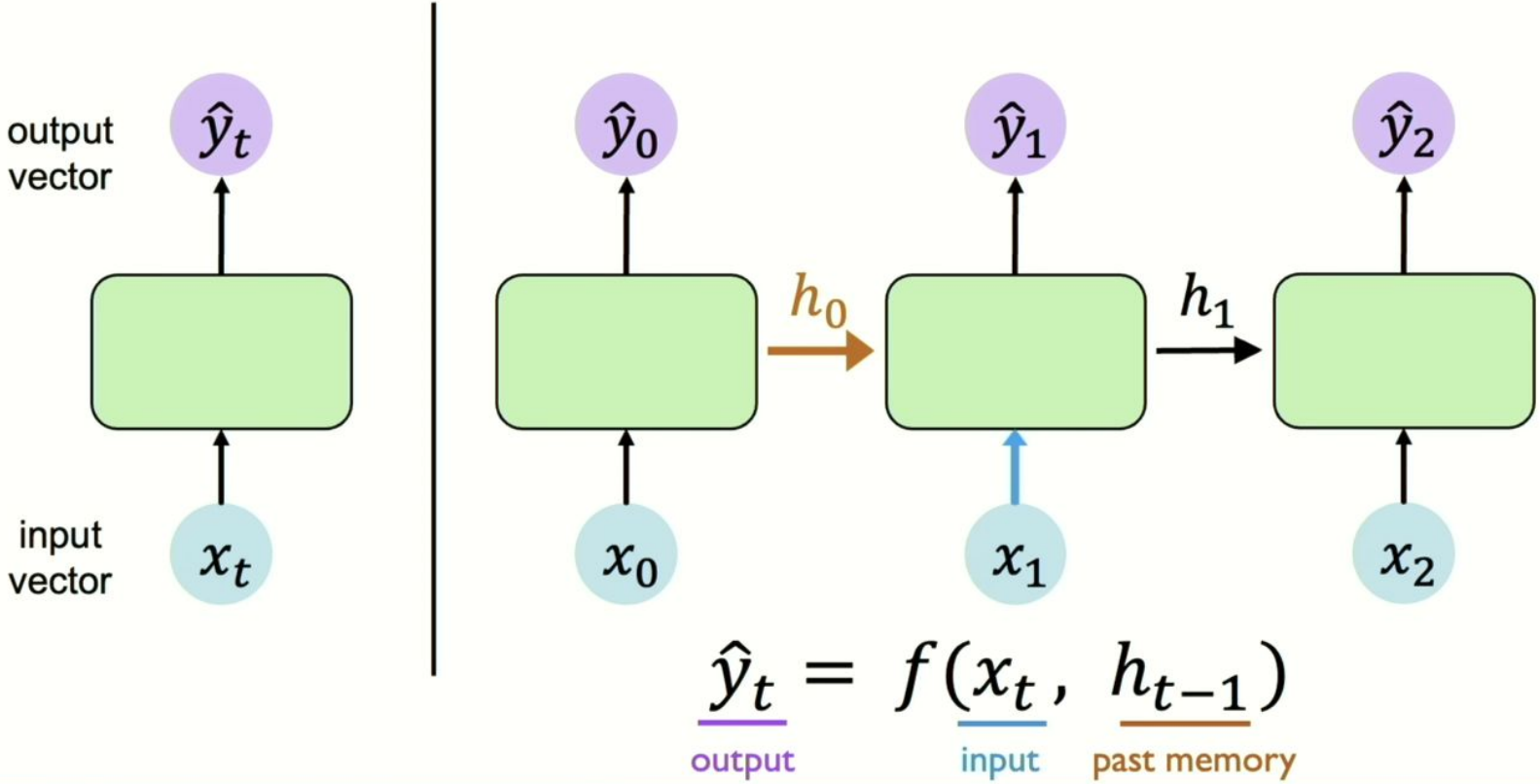


$$\hat{y}_t = f(x_t, h_{t-1})$$

Neurons with Recurrence

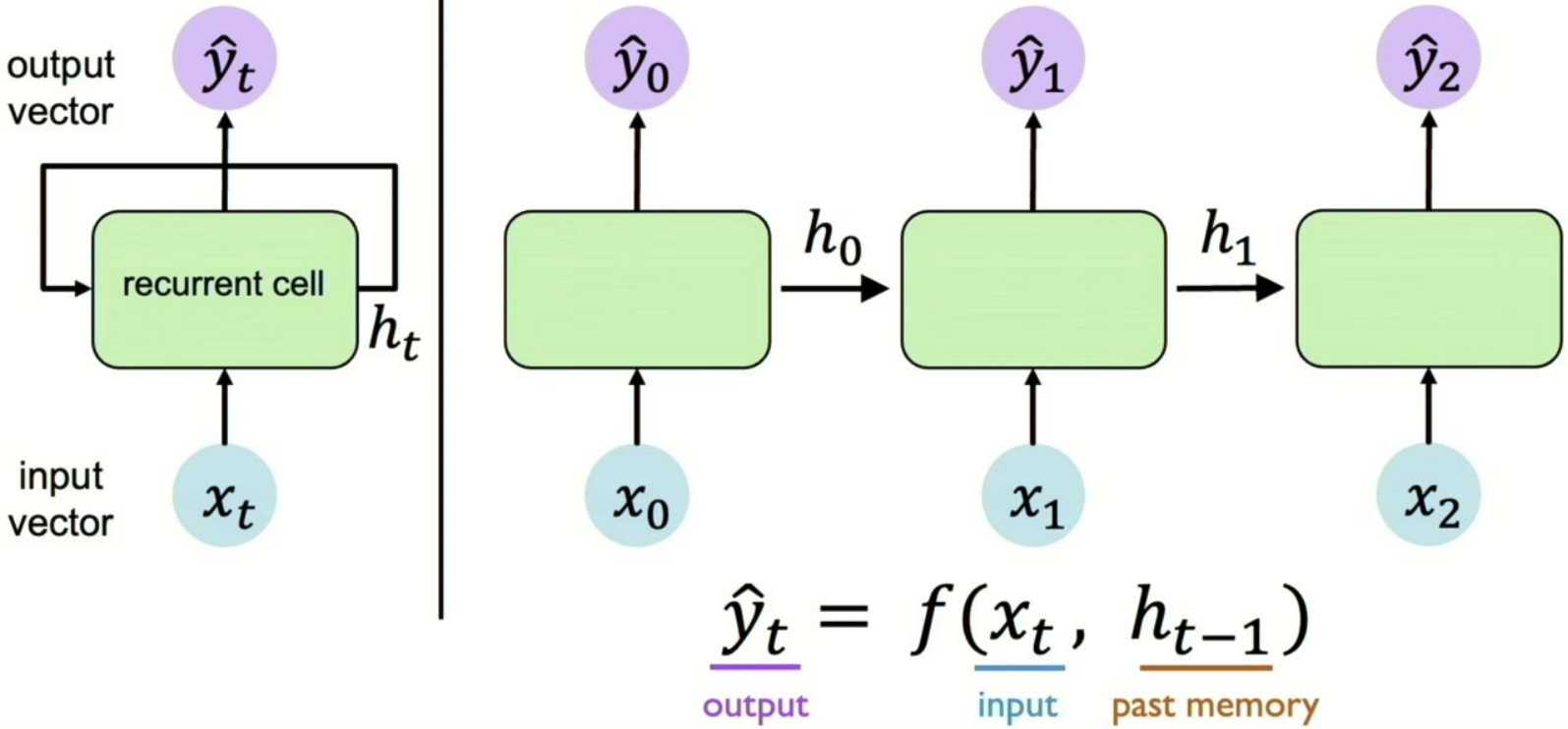


Neurons with Recurrence



$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

Neurons with Recurrence

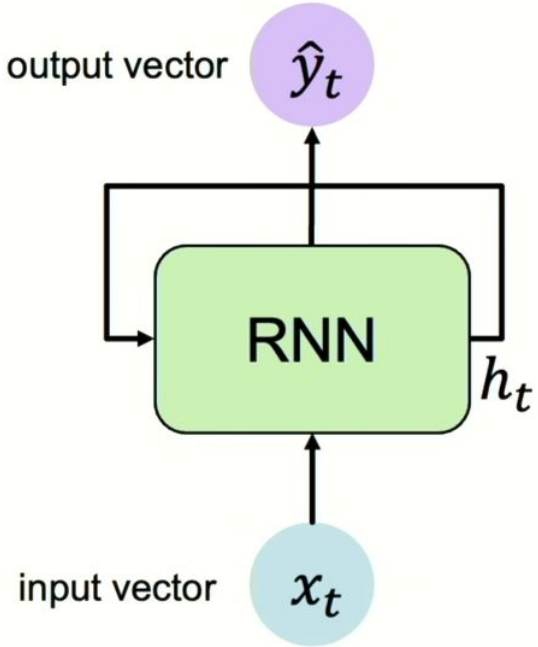


$$\underline{\hat{y}_t} = f(\underline{x_t}, \underline{h_{t-1}})$$

output input past memory

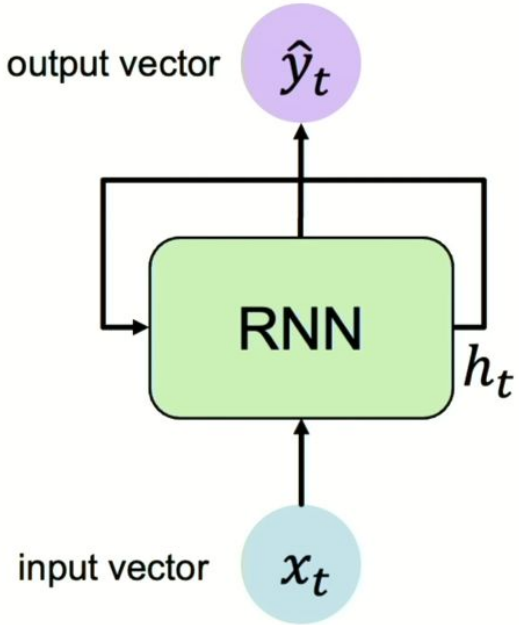
Рекурентні нейронні мережі

Recurrent Neural Networks (RNNs)



RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

Recurrent Neural Networks (RNNs)

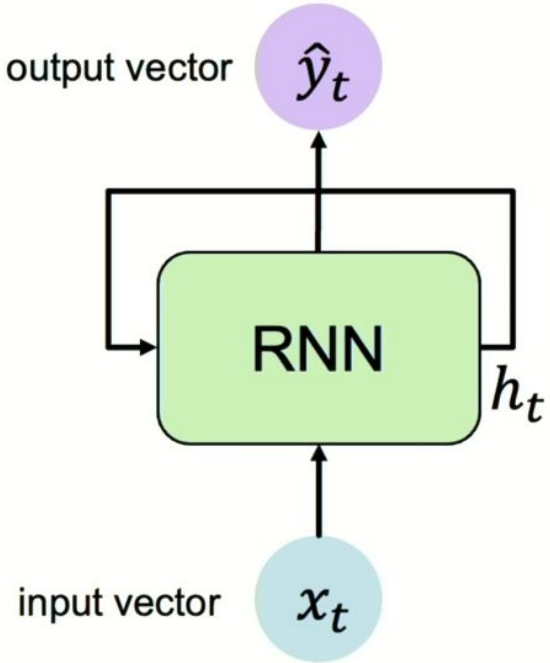


Apply a **recurrence relation** at every time step to process a sequence:

$$h_t = f_W(x_t, h_{t-1})$$

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

Recurrent Neural Networks (RNNs)



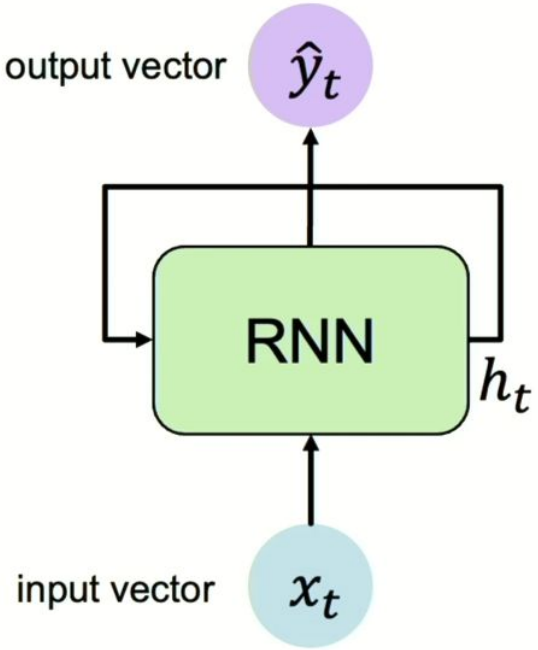
Apply a **recurrence relation** at every time step to process a sequence:

$$h_t = f_W(x_t, h_{t-1})$$

cell state

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

Recurrent Neural Networks (RNNs)



Apply a **recurrence relation** at every time step to process a sequence:

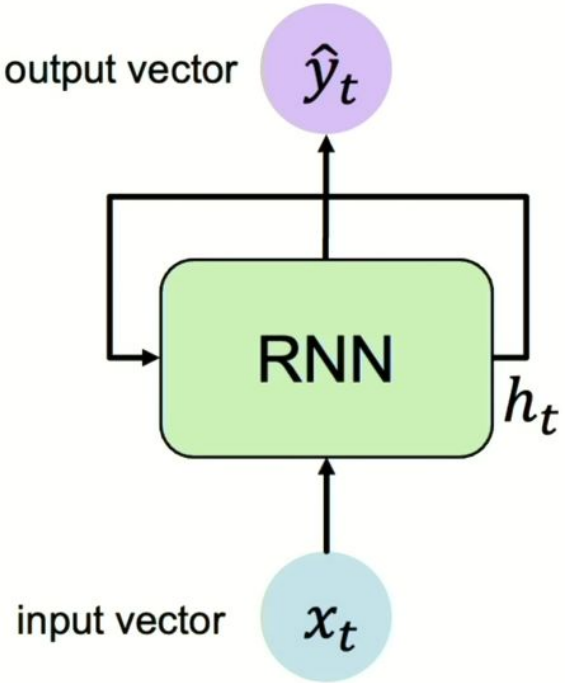
$$h_t = f_W(x_t, h_{t-1})$$

cell state

function with weights W

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

Recurrent Neural Networks (RNNs)



Apply a **recurrence relation** at every time step to process a sequence:

$$h_t = f_W(x_t, h_{t-1})$$

cell state

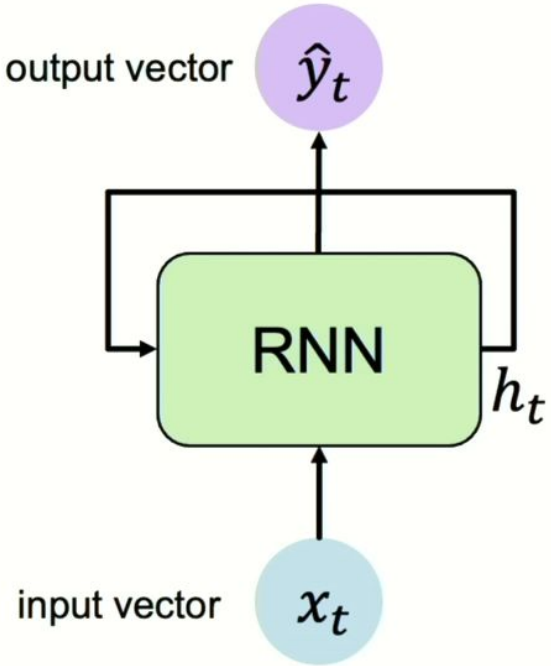
function with weights W

input

old state

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

Recurrent Neural Networks (RNNs)



Apply a **recurrence relation** at every time step to process a sequence:

$$h_t = f_W(x_t, h_{t-1})$$

cell state

function with weights W

input

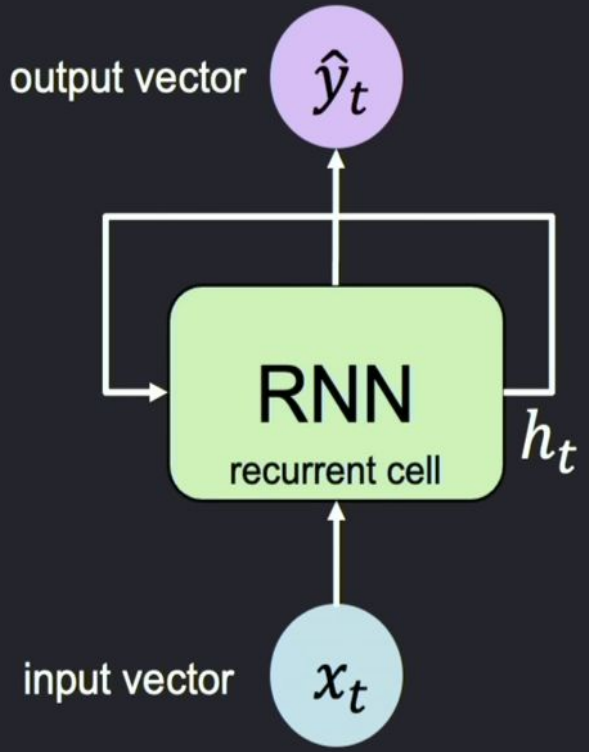
old state

Note: the same function and set of parameters are used at every time step

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

RNN Intuition

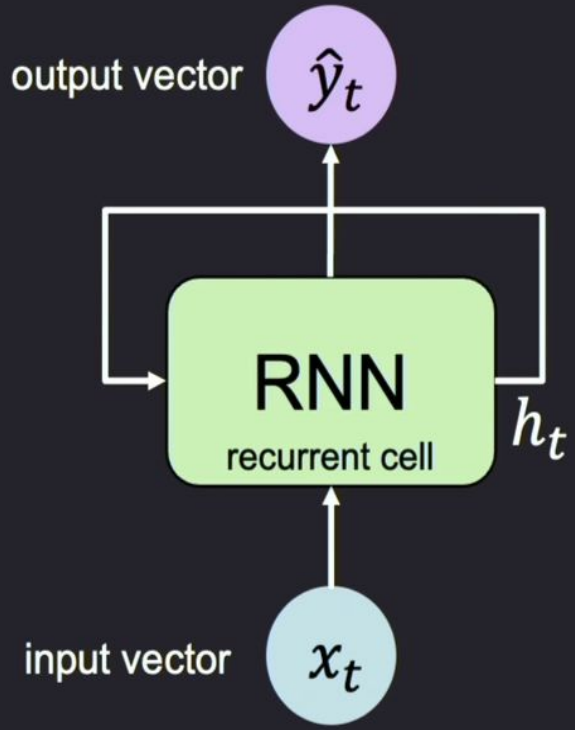
```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```



RNN Intuition

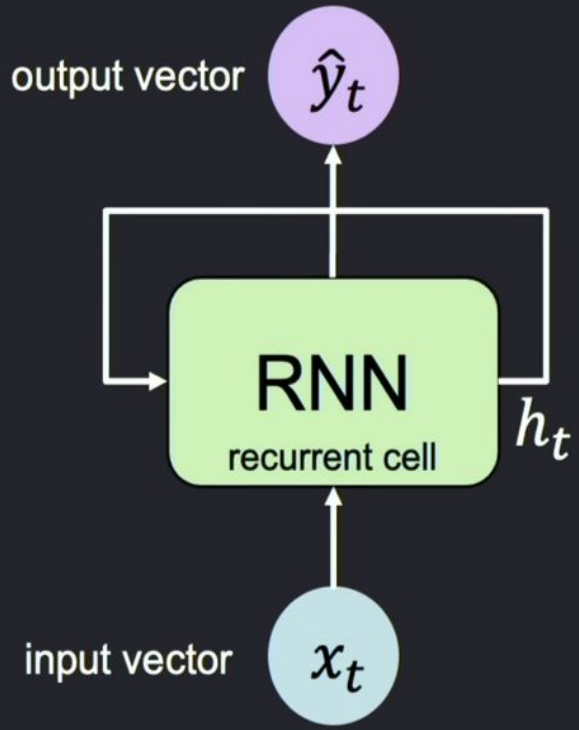
```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]
```

```
for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```



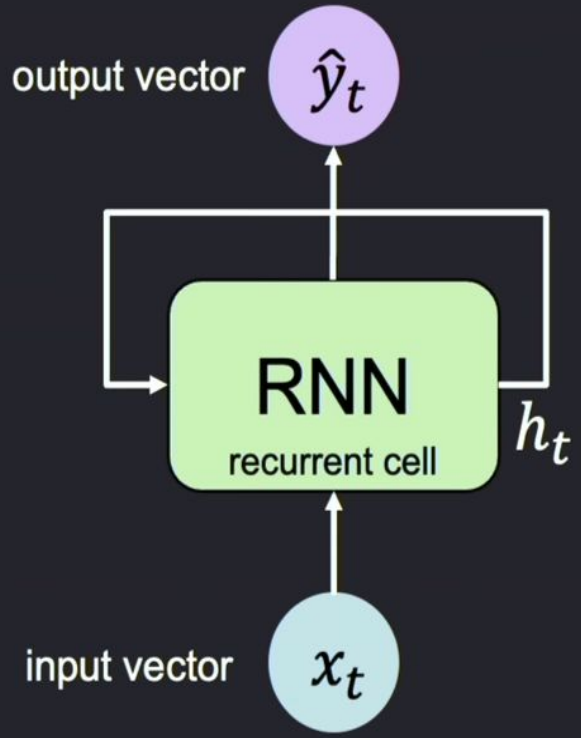
RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```

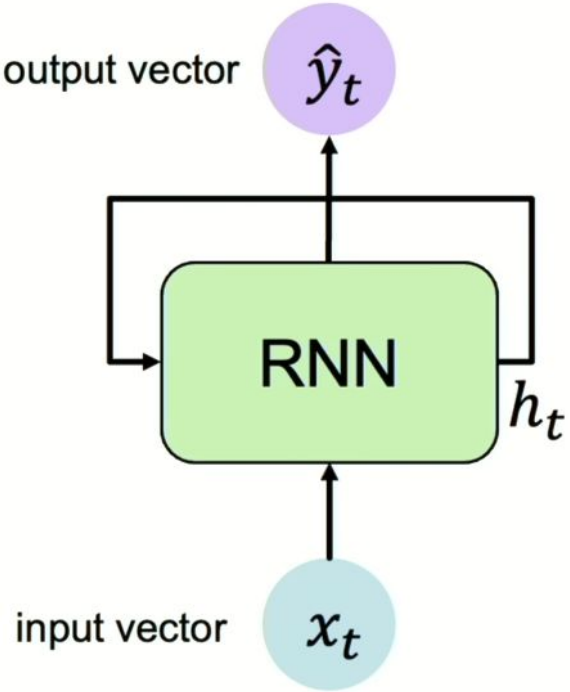


RNN Intuition

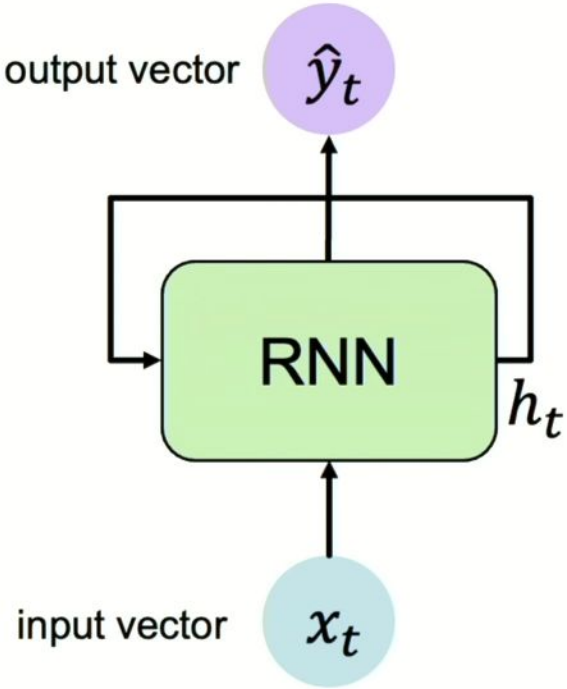
```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```



RNN State Update and Output



RNN State Update and Output



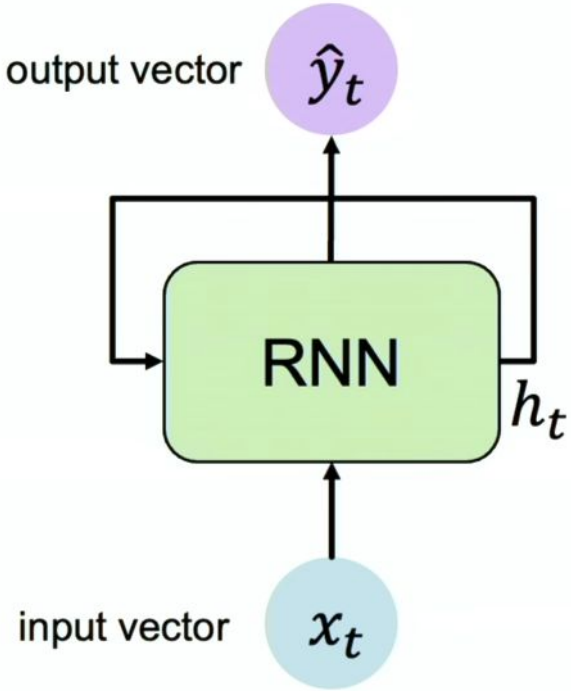
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

Input Vector

x_t

RNN State Update and Output



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{\boldsymbol{hy}}^{\boldsymbol{T}} h_t$$

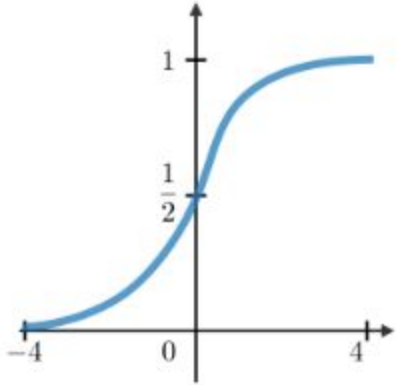
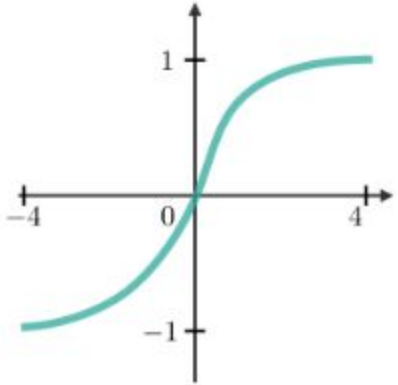
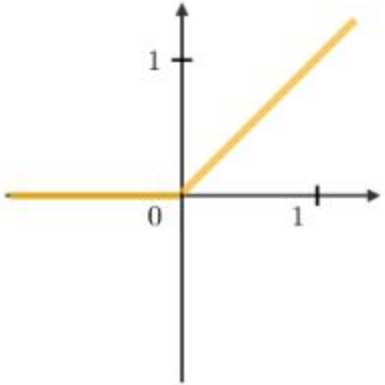
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

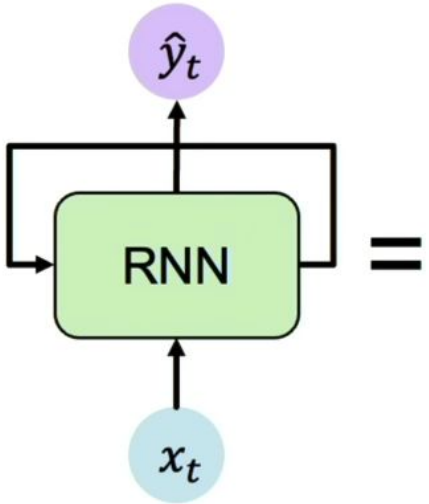
Input Vector

$$x_t$$

Що таке tanh і звідки він тут

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

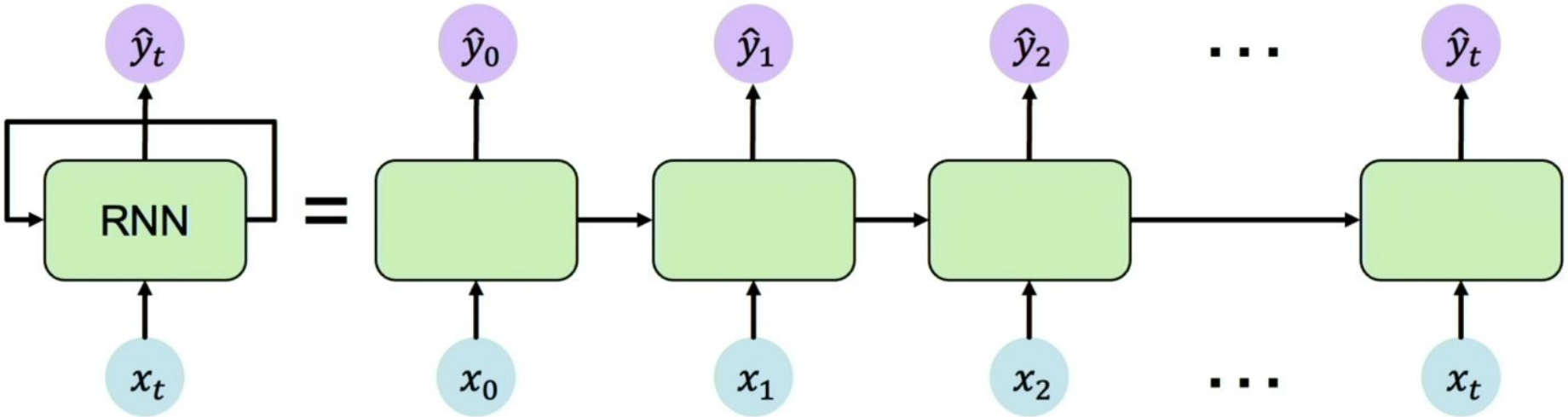
RNNs: Computational Graph Across Time



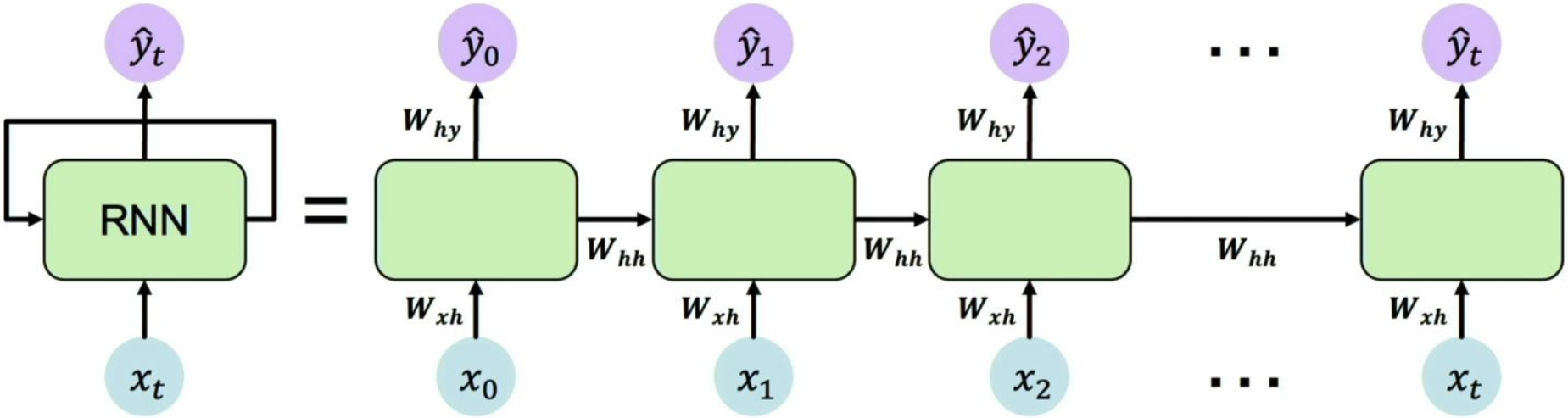
=

Represent as computational graph unrolled across time

RNNs: Computational Graph Across Time

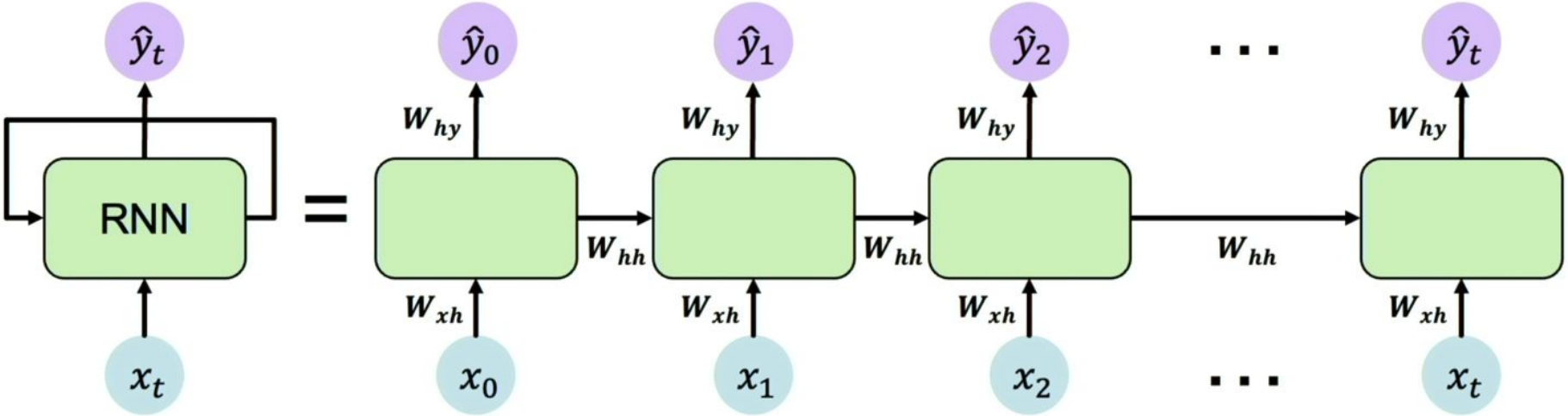


RNNs: Computational Graph Across Time

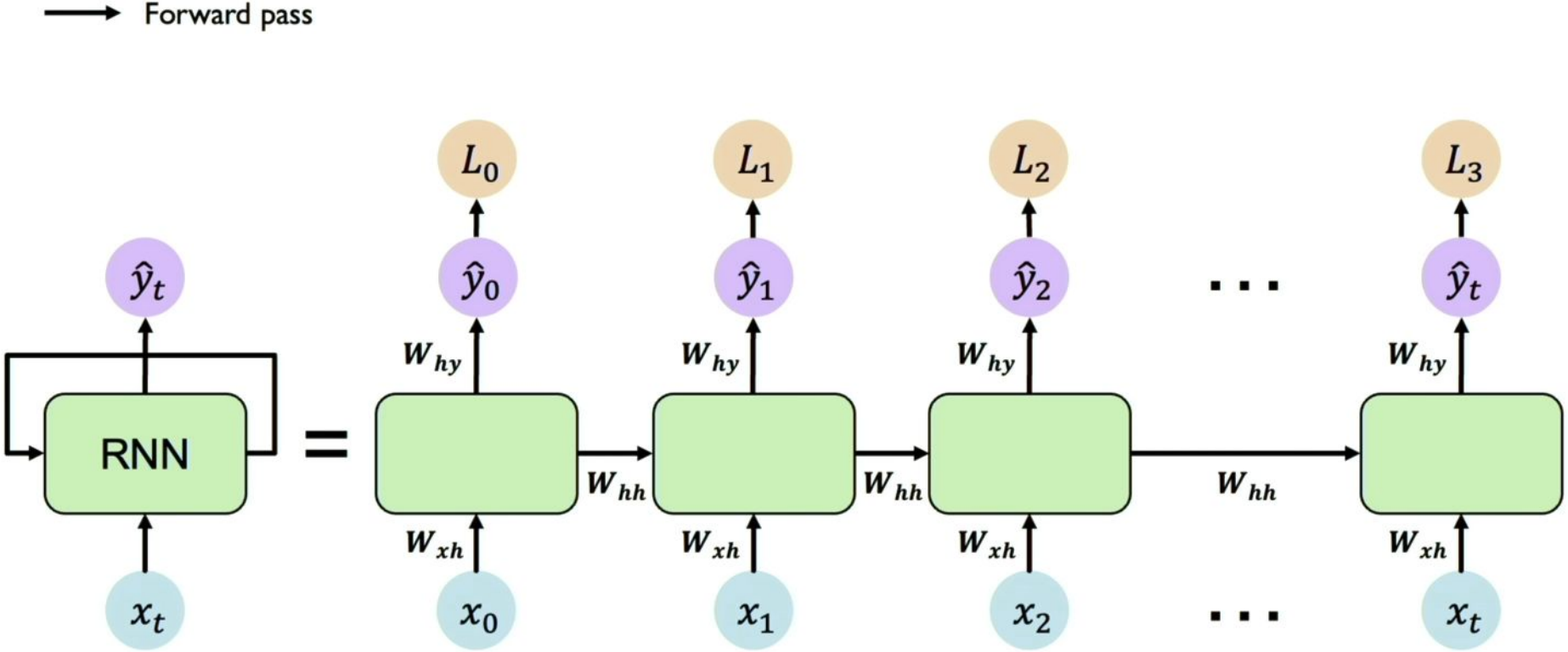


RNNs: Computational Graph Across Time

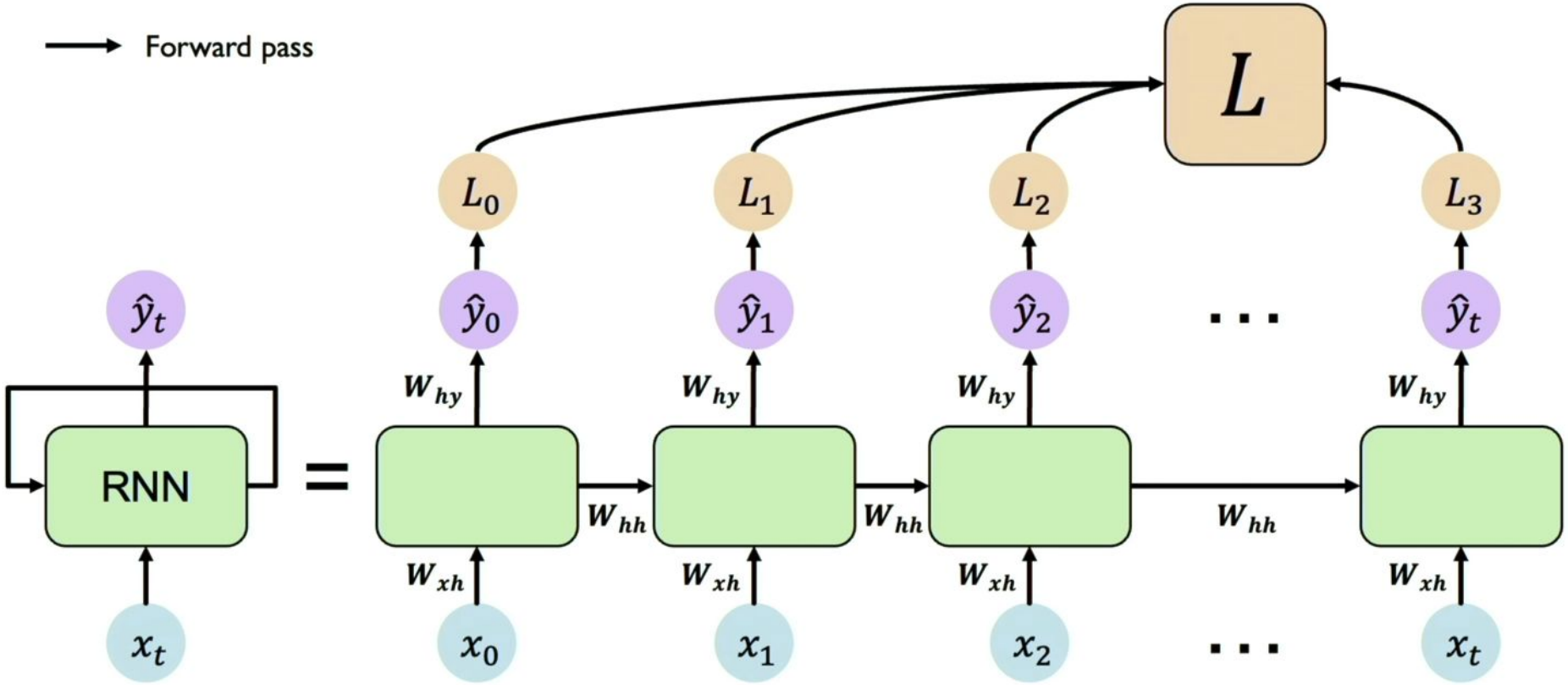
Re-use the **same weight matrices** at every time step



RNNs: Computational Graph Across Time

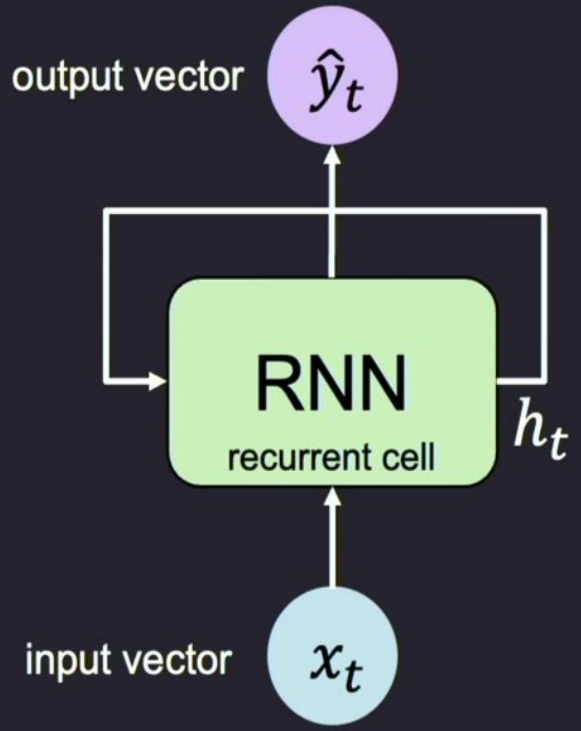


RNNs: Computational Graph Across Time



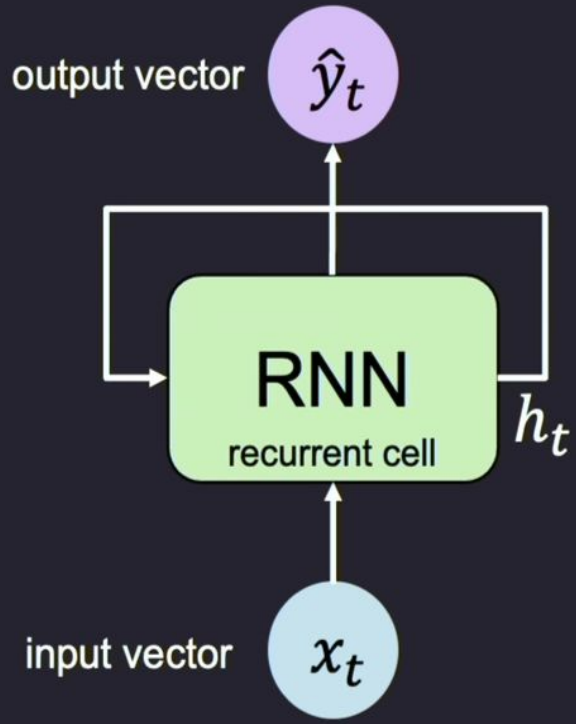
RNNs from Scratch

```
import torch.nn as nn
import torch.nn.functional as F

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.W_xh = nn.Linear(input_size, hidden_size)
        self.W_hh = nn.Linear(hidden_size, hidden_size)
        self.W_hy = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        hidden = F.tanh(self.W_xh(input) + self.W_hh(hidden))
        output = self.W_hy(hidden)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```



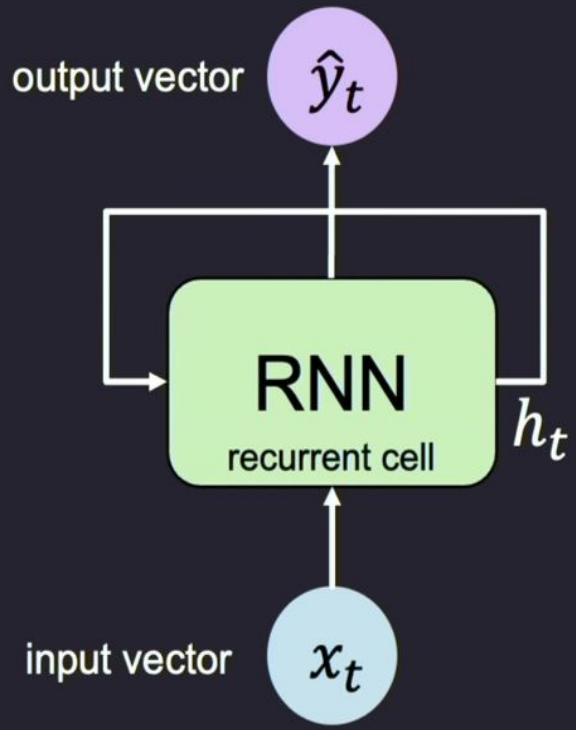
RNNs from Scratch

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



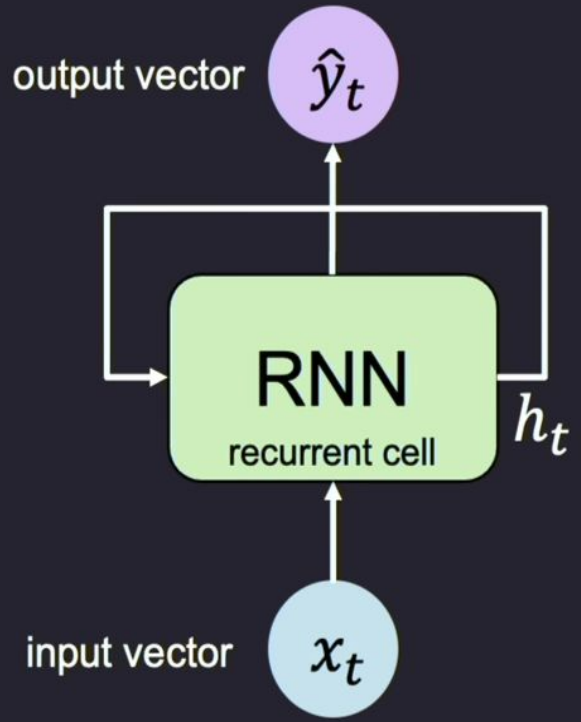
RNNs from Scratch

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



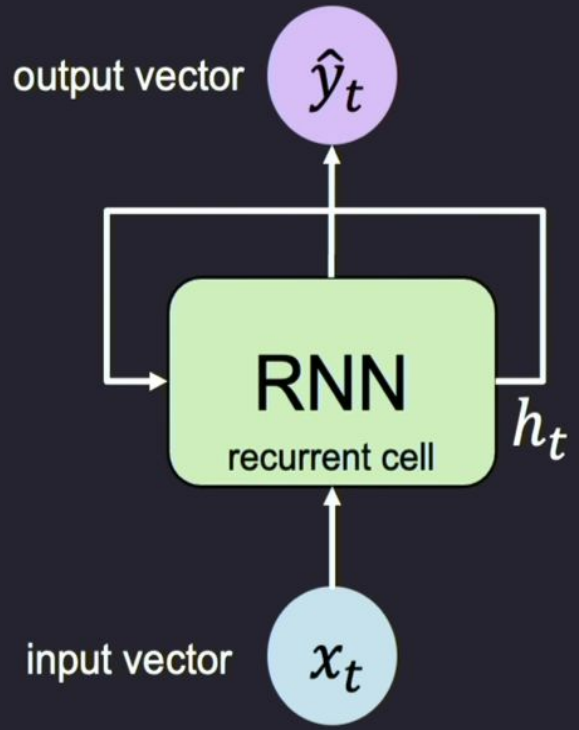
RNNs from Scratch

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



RNNs from Scratch

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



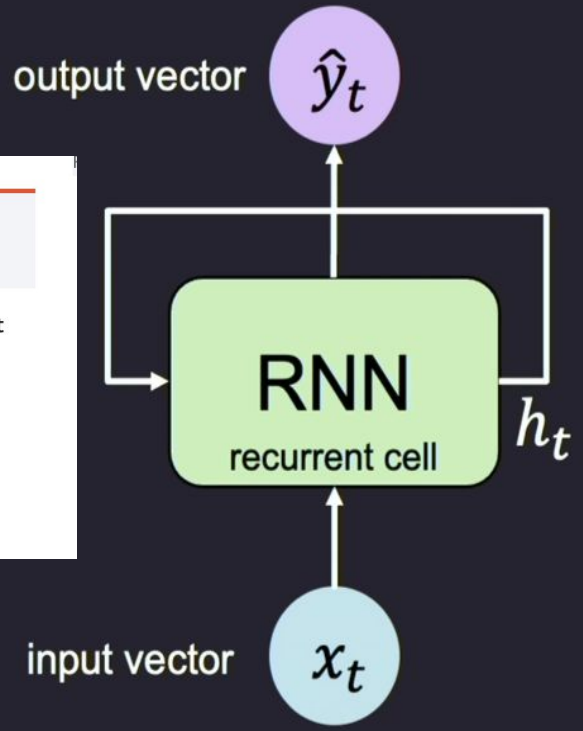
RNN Implementation in PyTorch

CLASS torch.nn.RNN(*input_size*, *hidden_size*, *num_layers=1*, *nonlinearity='tanh'*, *bias=True*, *batch_first=False*, *dropout=0.0*, *bidirectional=False*, *device=None*, *dtype=None*) [SOURCE]

Apply a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence. For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

where h_t is the hidden state at time *t*, x_t is the input at time *t*, and $h_{(t-1)}$ is the hidden state of the previous layer at time *t-1* or the initial hidden state at time *0*. If `nonlinearity` is `'relu'`, then ReLU is used instead of $\tanh$.

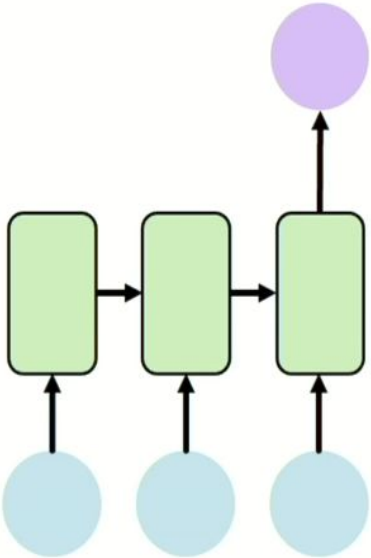


RNNs for Sequence Modeling

One to One
"Vanilla" NN
Binary classification

RNNs for Sequence Modeling

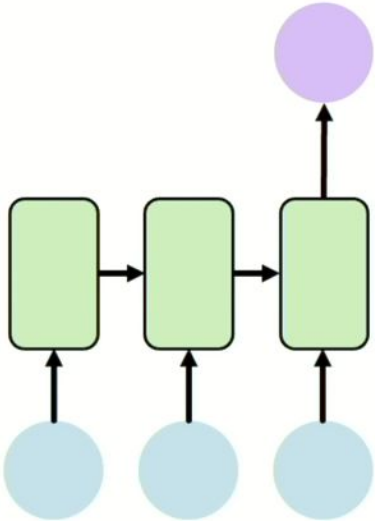
One to One
"Vanilla" NN
Binary classification



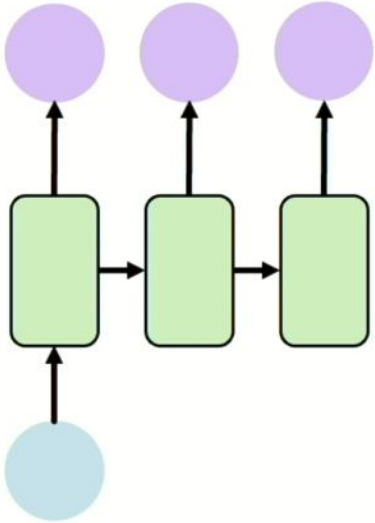
Many to One
Sentiment Classification

RNNs for Sequence Modeling

One to One
"Vanilla" NN
Binary classification

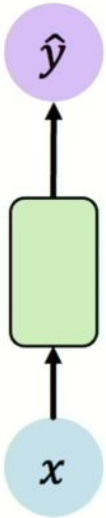


Many to One
Sentiment Classification

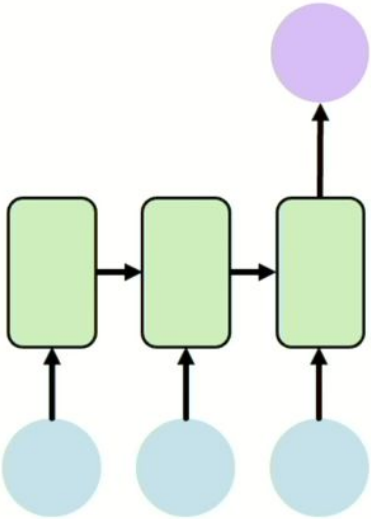


One to Many
Text Generation
Image Captioning

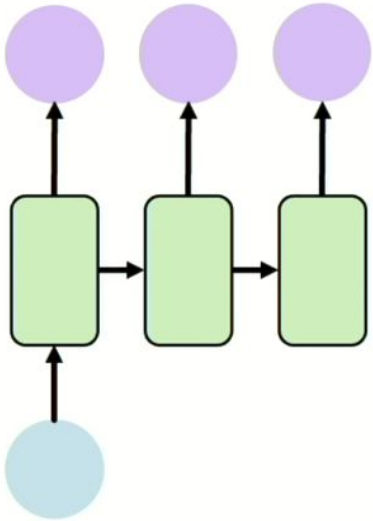
RNNs for Sequence Modeling



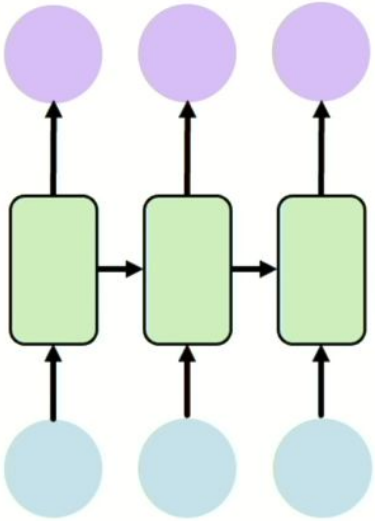
One to One
"Vanilla" NN
Binary classification



Many to One
Sentiment Classification



One to Many
Text Generation
Image Captioning

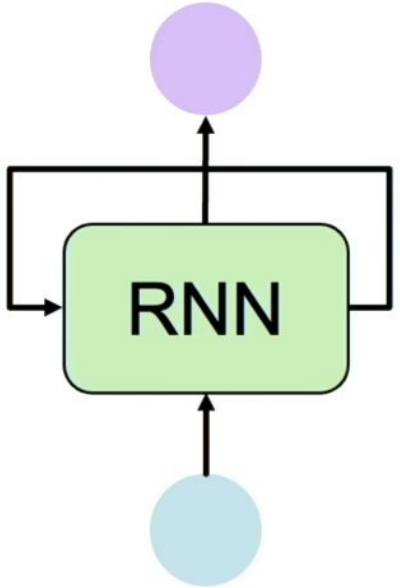


Many to Many
Translation & Forecasting
Music Generation

Sequence Modeling: Design Criteria

To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



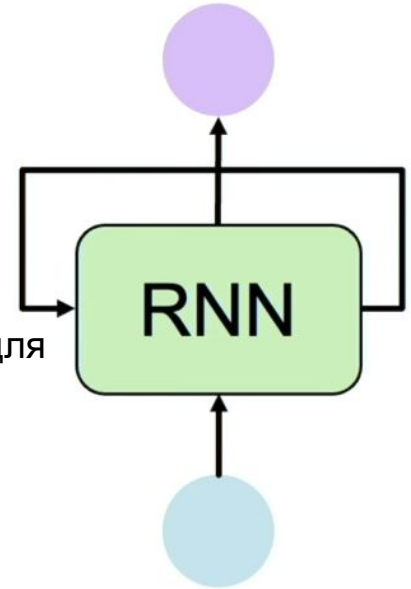
Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria

Моделювання послідовностей: Критерії дизайну

Щоб моделювати послідовності, нам потрібно:

1. Обробляти послідовності **змінної** довжини
2. Відслідковувати **довготривалі** залежності
3. Зберігати **інформацію про порядок**
4. **Спільно використовувати параметри** по всій послідовності

Рекурентні нейронні мережі (RNN) відповідають цим критеріям дизайну для моделювання послідовностей.



Задача моделювання послідовностей: передбачення наступного слова

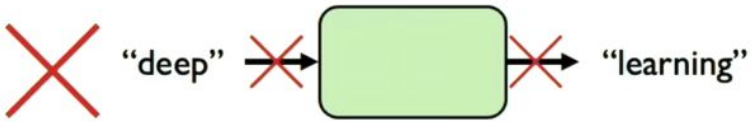
A Sequence Modeling Problem: Predict the Next Word

"This morning I took my cat for a walk."

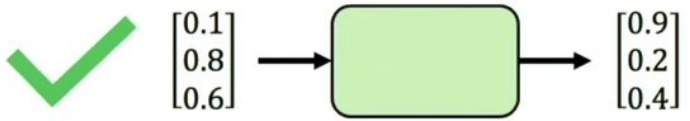
given these words

predict the next word

Representing Language to a Neural Network



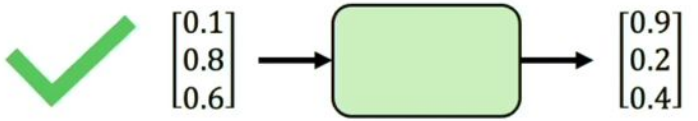
Neural networks cannot interpret words



Neural networks require numerical inputs

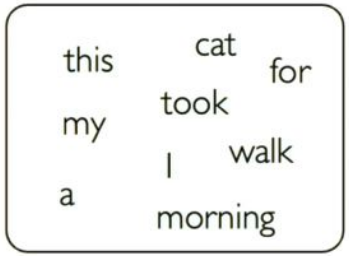
Encoding Language for a Neural Network

Neural networks cannot interpret words

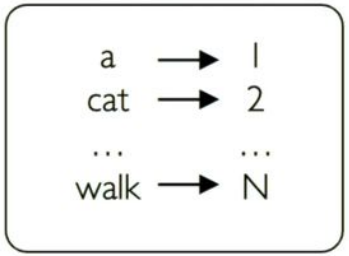


Neural networks require numerical inputs

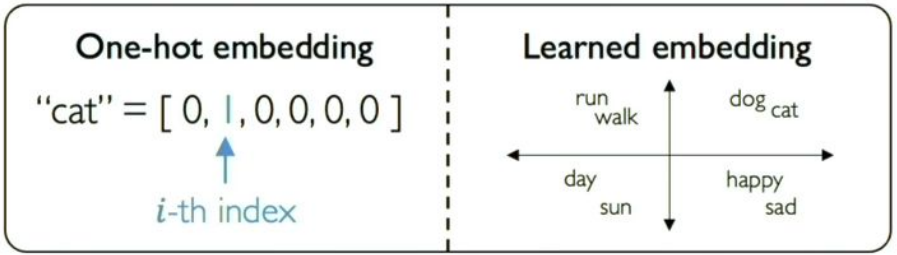
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words



2. Indexing:
Word to index



3. Embedding:
Index to fixed-sized vector

Handle Variable Sequence Lengths

The food was great

vs.

We visited a restaurant for lunch

vs.

We were hungry but cleaned the house before eating

Model Long-Term Dependencies

"**France** is where I grew up, but I now live in Boston. I speak fluent ___."

We need information from **the distant past** to accurately predict the correct word.

Capture Differences in Sequence Order

The food was good, not bad at all.

vs.

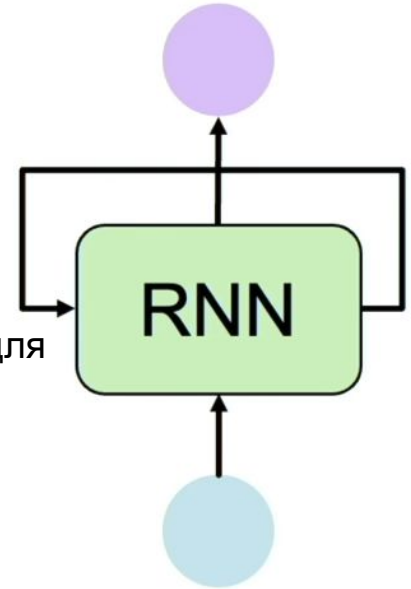
The food was bad, not good at all.

Моделювання послідовностей: Критерії дизайну

Щоб моделювати послідовності, нам потрібно:

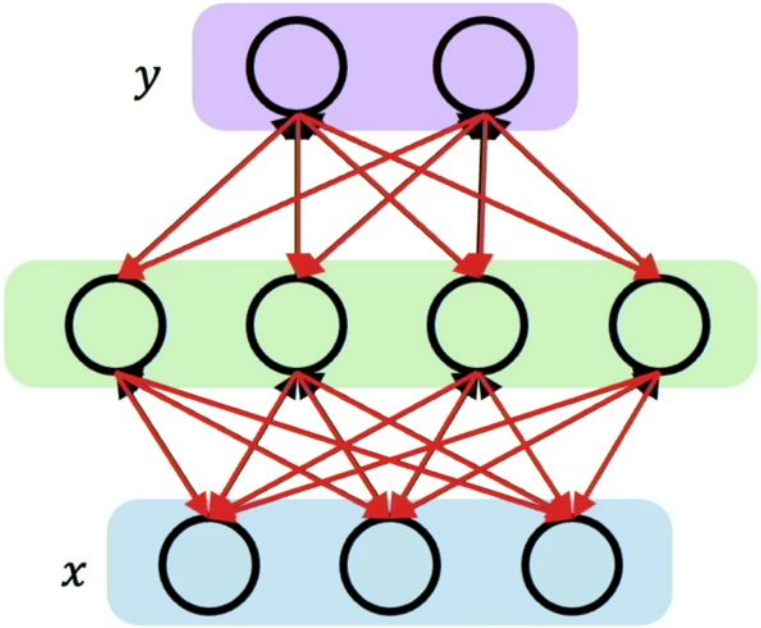
1. Обробляти послідовності **змінної** довжини
2. Відслідковувати **довготривалі** залежності
3. Зберігати **інформацію про порядок**
4. **Спільно використовувати параметри** по всій послідовності

Рекурентні нейронні мережі (RNN) відповідають цим критеріям дизайну для моделювання послідовностей.



Backpropagation Through Time (BPTT)

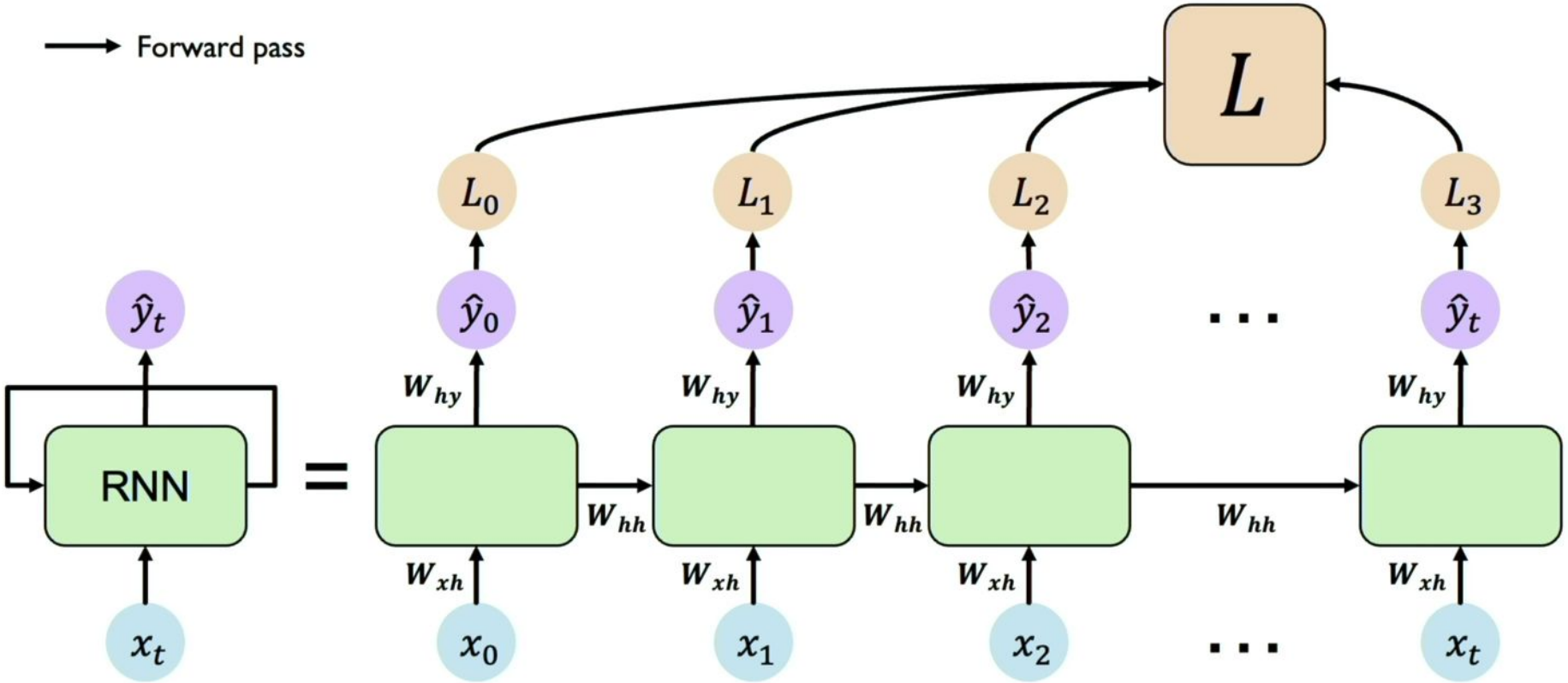
Recall: Backpropagation in Feed Forward Models



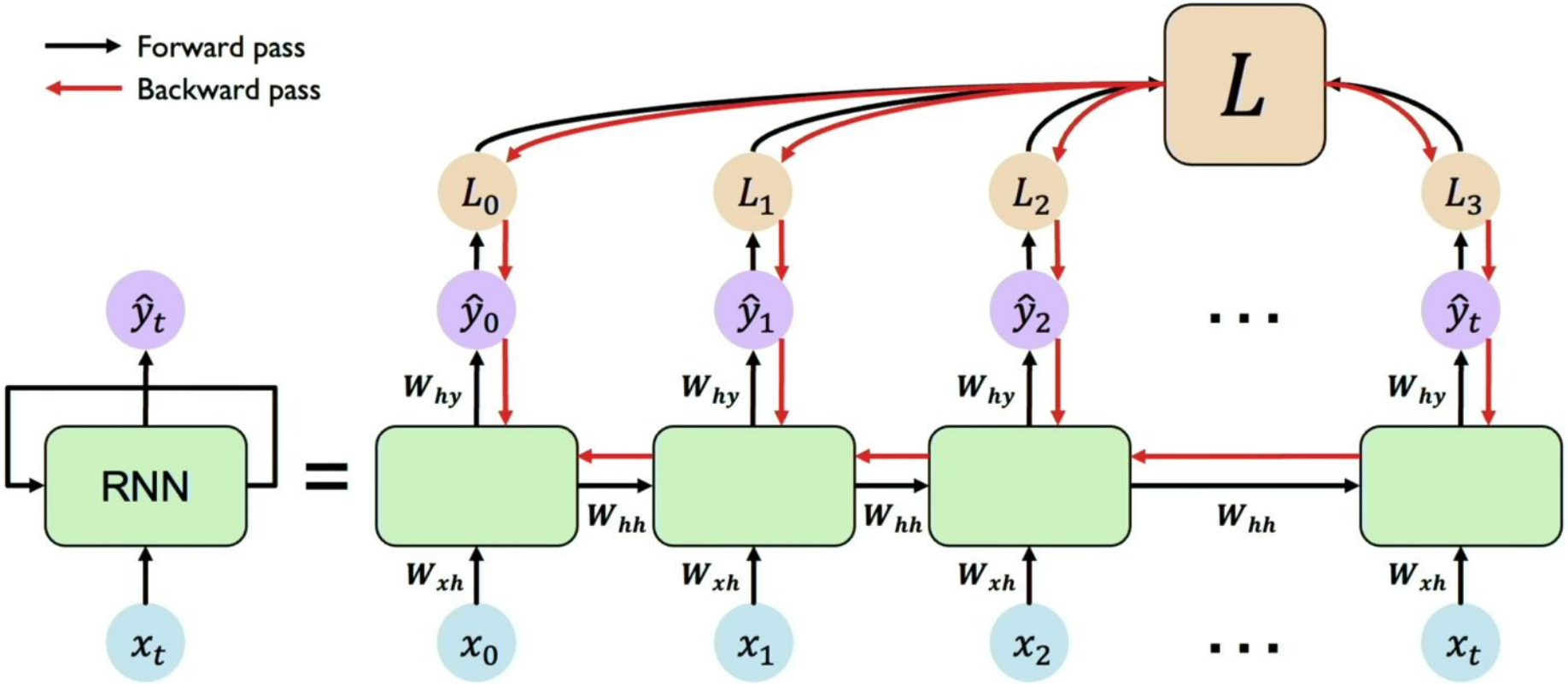
Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

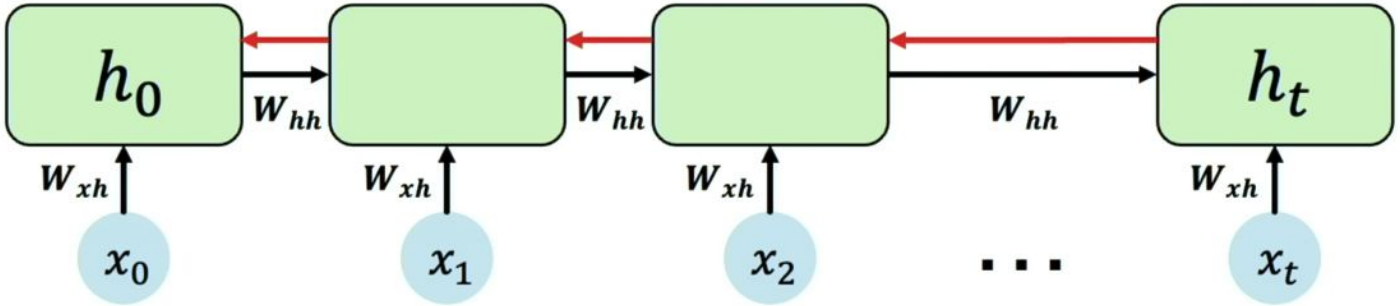
RNNs: Backpropagation Through Time



RNNs: Backpropagation Through Time



Standard RNN Gradient Flow: Exploding Gradients

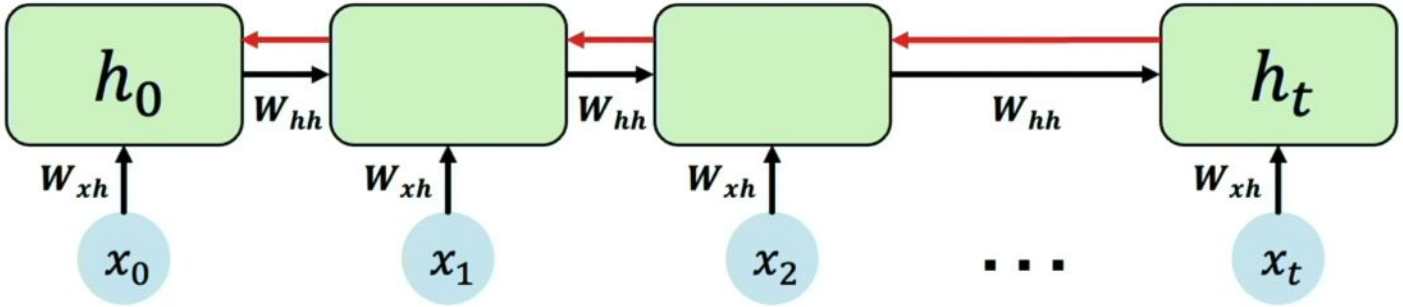


Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

Many values > 1:
exploding gradients

Gradient clipping to
scale big gradients

Standard RNN Gradient Flow: Vanishing Gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

Many values < 1:
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

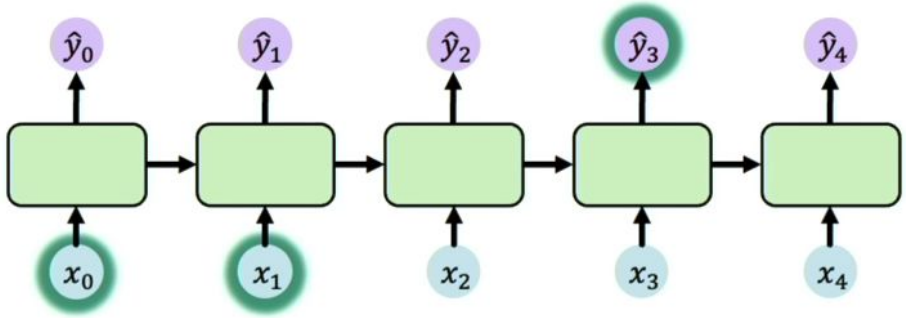
↓

Errors due to further back time steps have smaller and smaller gradients

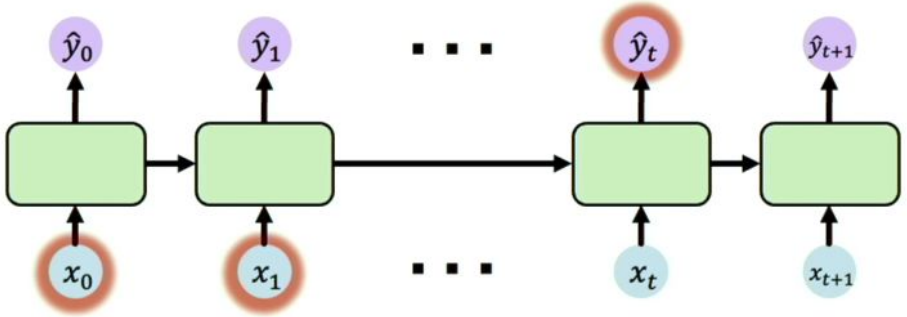
↓

Bias parameters to capture short-term dependencies

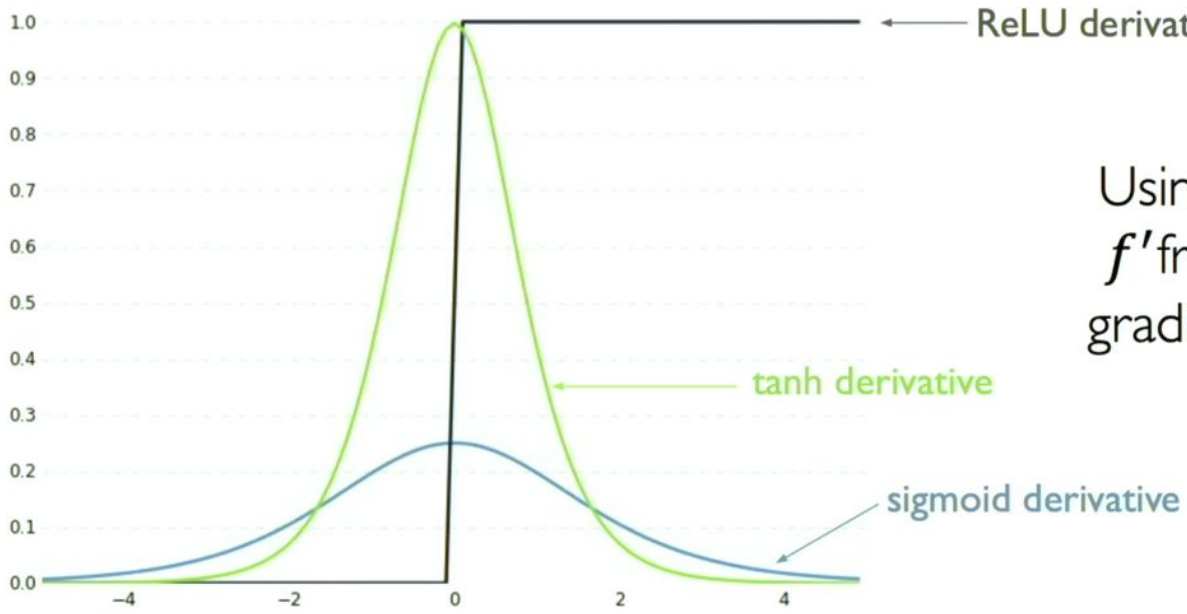
"The clouds are in the ___"



"I grew up in France, … and I speak fluent___ "



Trick #1: Activation Functions



Using ReLU prevents f' from shrinking the gradients when $x > 0$

Trick #2: Parameter Initialization

Initialize **weights** to identity matrix

Initialize **biases** to zero

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

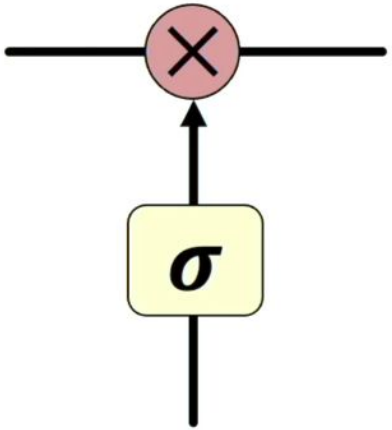
This helps prevent the weights from shrinking to zero.

Trick #3: Gated Cells

Idea: use **gates** to selectively **add** or **remove** information within **each recurrent unit with**

Pointwise multiplication

Sigmoid neural net layer



gated cell

LSTM, GRU, etc.

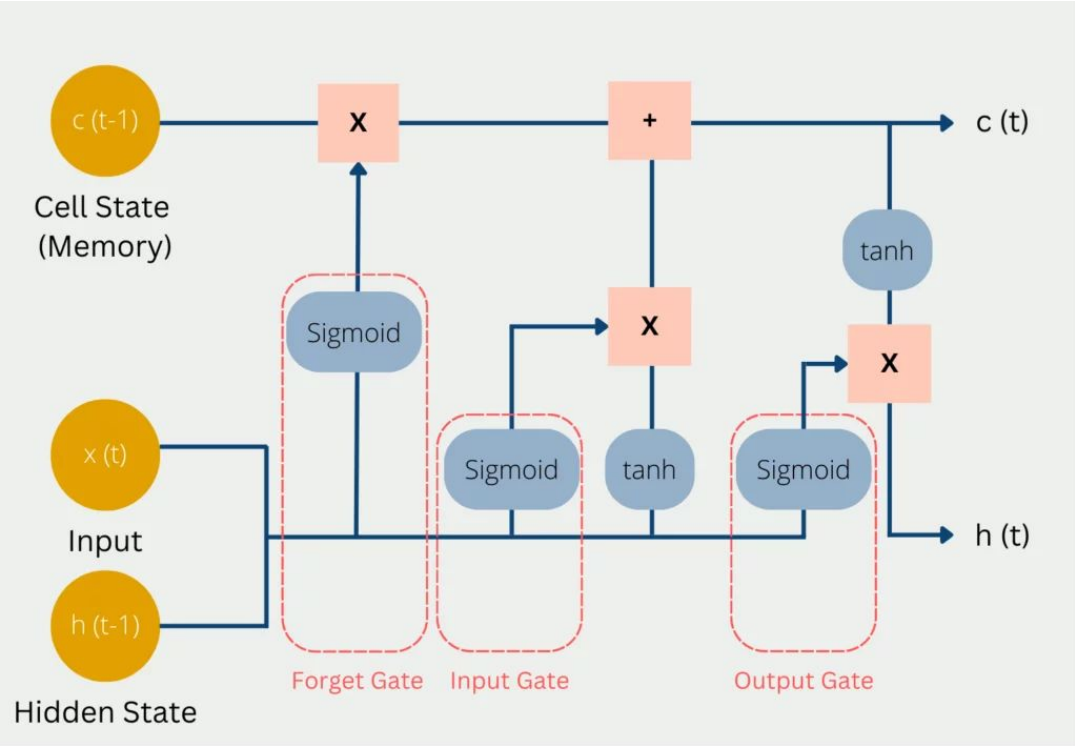
Gates optionally let information through the cell

Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

LSTM основна ідея

LSTM (Long Short-Term Memory) — це вид рекурентної нейронної мережі (RNN), що здатна запам'ятовувати важливу інформацію на тривалий час і забувати нерелевантні дані, завдяки використанню спеціальних воріт: **забування**, **зберігання** та **виходу**. Це дозволяє їй краще працювати з послідовними даними, зберігаючи довготривалі залежності.

Структура LSTM-cell



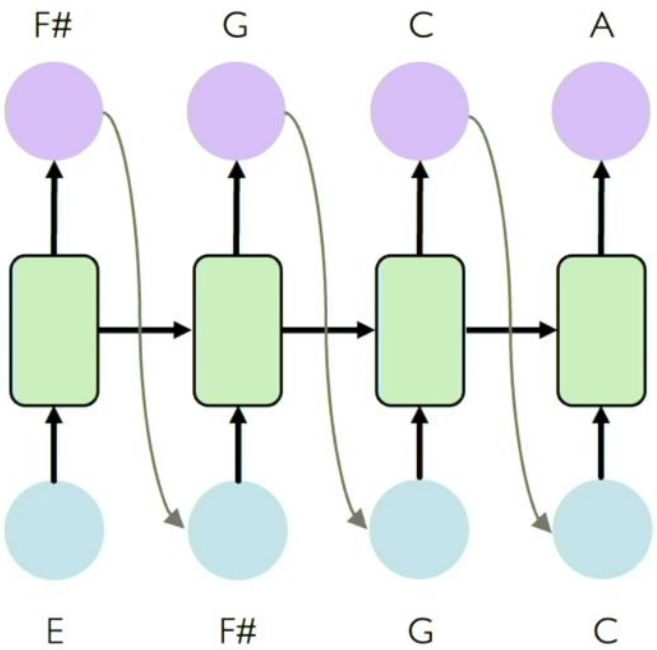
Детально описаний принцип роботи тут:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs: Key Concepts

1. Maintain a **cell state**
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with partially **uninterrupted gradient flow**

Обмеження RNN

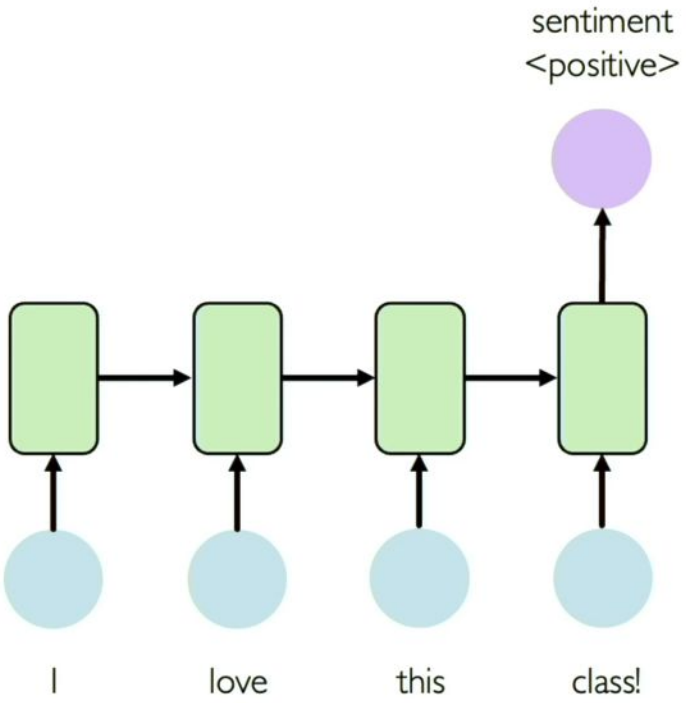
Example Task: Music Generation



Input: sheet music

Output: next character in sheet music

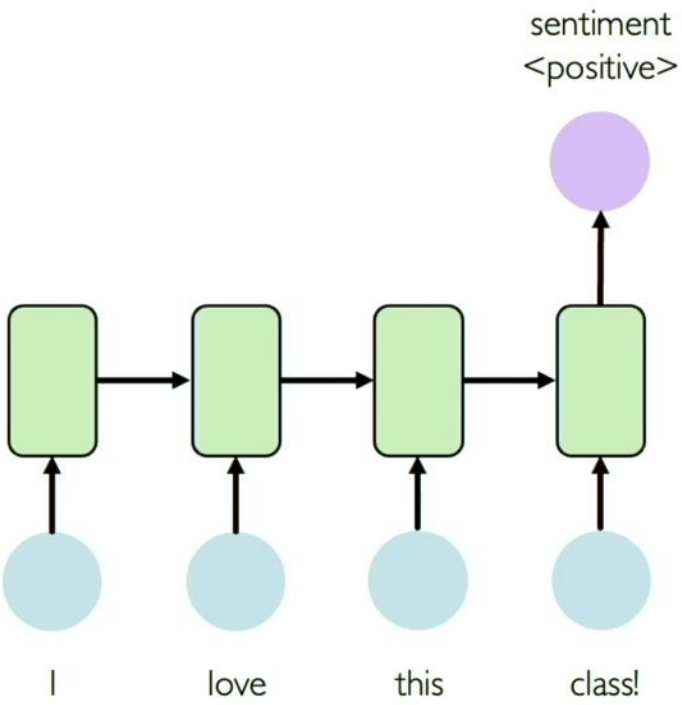
Example Task: Sentiment Classification



Input: sequence of words

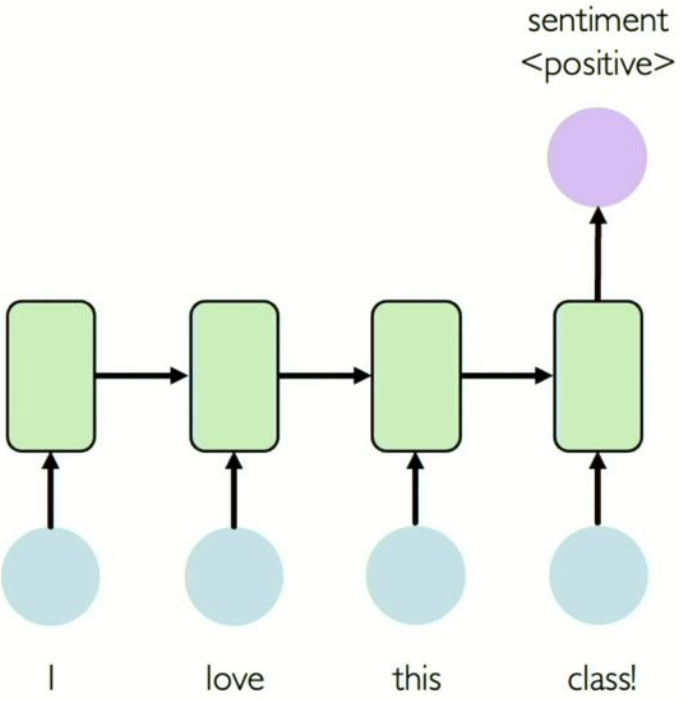
Output: probability of having positive sentiment

Example Task: Sentiment Classification



Tweet sentiment classification

Limitations of Recurrent Models



Limitations of RNNs

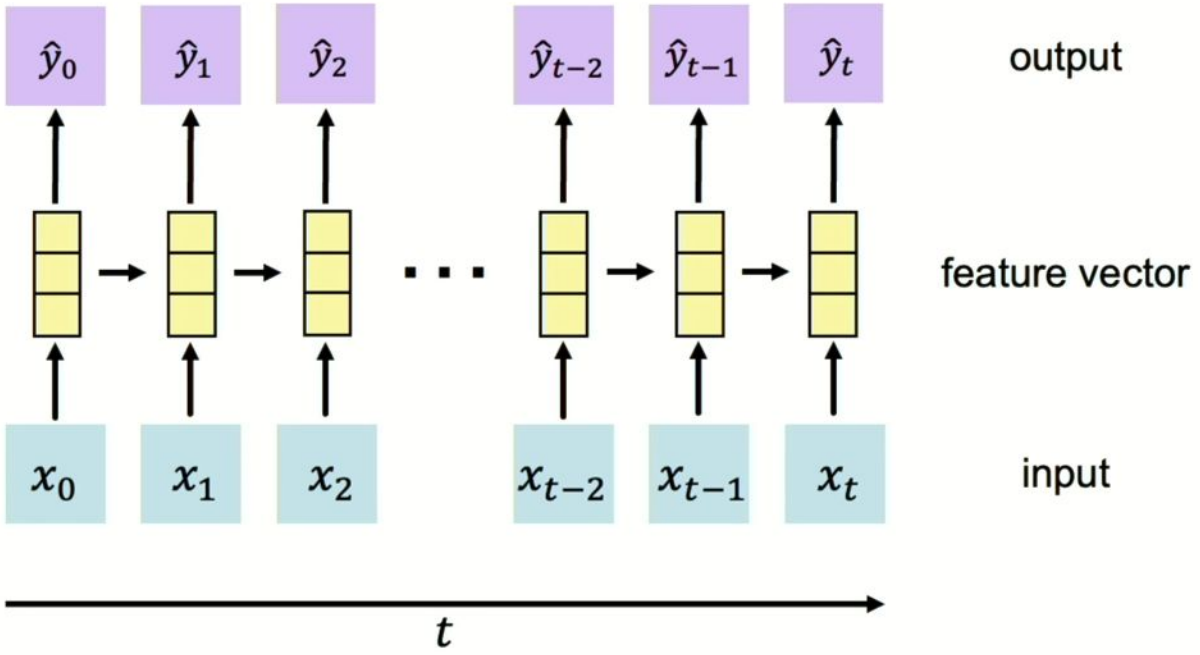
- Encoding bottleneck
- Slow, no parallelization
- Not long memory

Goal of Sequence Modeling

RNNs: recurrence to model sequence dependencies

Limitations of RNNs

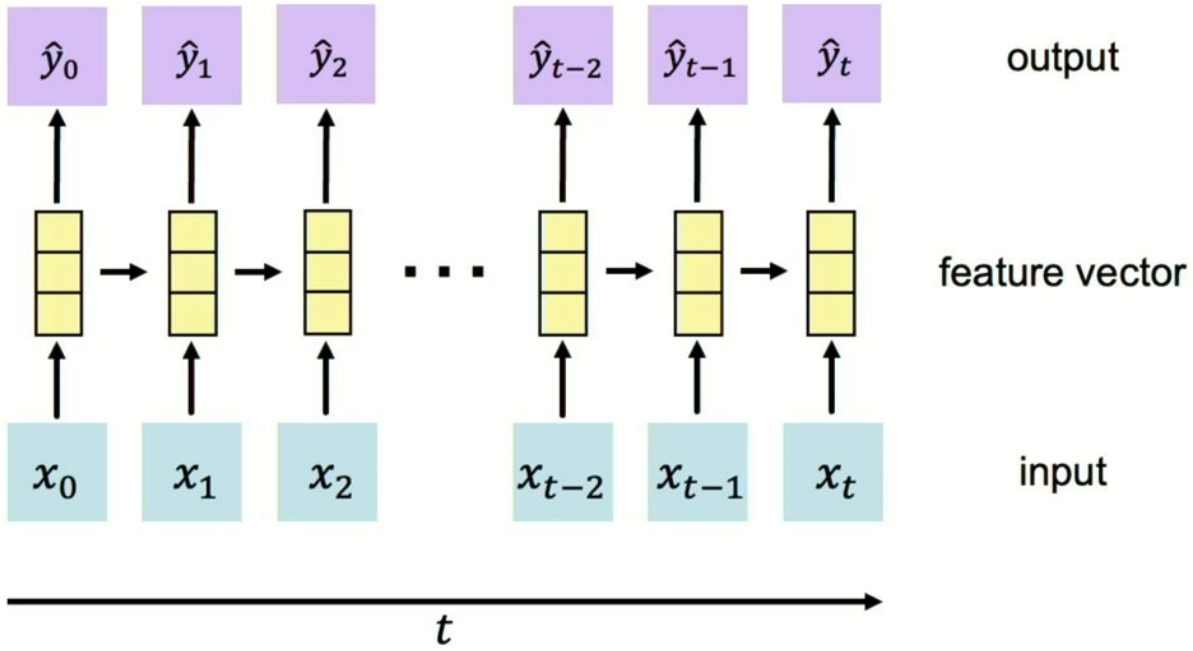
- Encoding bottleneck
- Slow, no parallelization
- Not long memory



Goal of Sequence Modeling

Desired Capabilities

- Continuous stream
- Parallelization
- Long memory

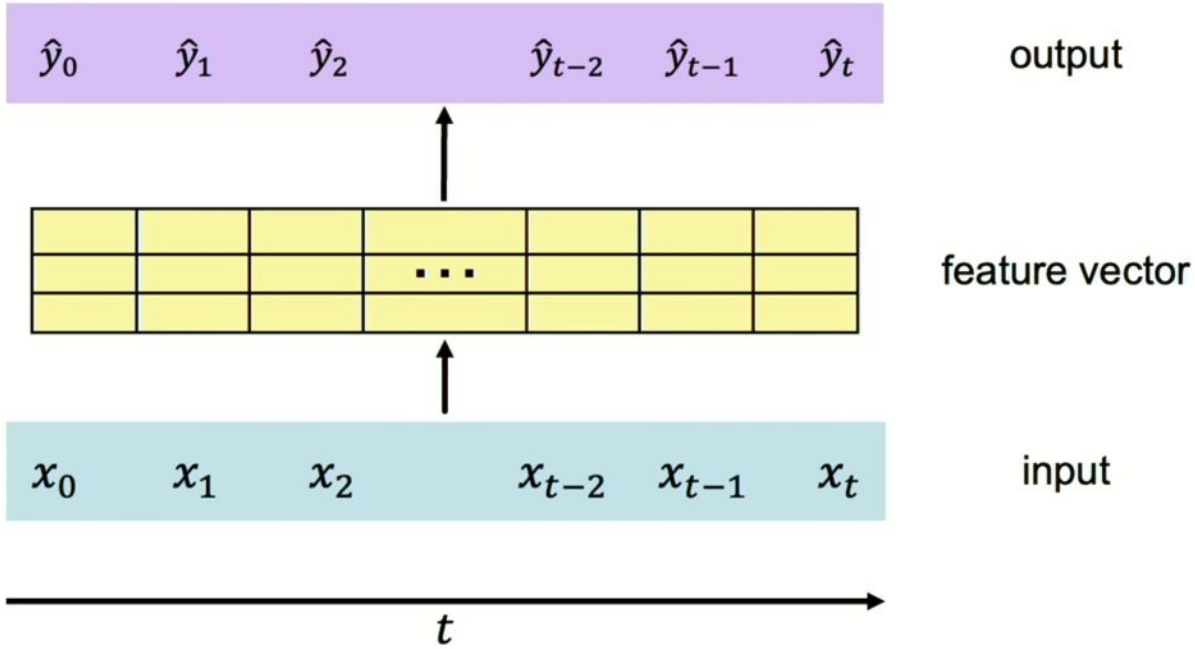


Goal of Sequence Modeling

Can we eliminate the need for recurrence entirely?

Desired Capabilities

- Continuous stream
- Parallelization
- Long memory



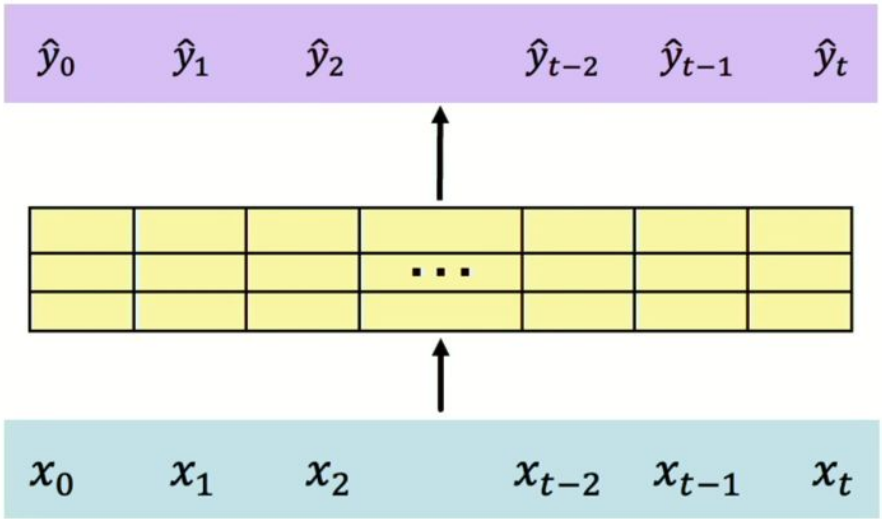
Goal of Sequence Modeling

Idea 1: Feed everything into dense network

- ✓ No recurrence
- ✗ Not scalable
- ✗ No order
- ✗ No long memory

Idea: Identify and attend to what's important

Can we eliminate the need for recurrence entirely?



Attention Is All You Need

https://arxiv.org/abs/1706.03762

Intuition Behind Self-Attention

Attending to the most important parts of an input.

Intuition Behind Self-Attention

Attending to the most important parts of an input.

Intuition Behind Self-Attention

Attending to the most important parts of an input.

1. Identify which parts to attend to
2. Extract the features with high attention

Intuition Behind Self-Attention

Attending to the most important parts of an input.

1. Identify which parts to attend to
2. Extract the features with high attention

Similar to a search problem!

A Simple Example: Search

Understanding Attention with Search

YouTube

GIANT SEA TURTLES • AMAZING CORAL REEF FISH • 12 HOURS of THE BEST RELAX MUSIC

5.4M views • 4 years ago

Cat Trumpet

Enjoy 3 hours of giant sea **turtles**. This video features amazing coral reef fish and relaxing ideal for sleep, study and ...

MIT 6.S191 (2020): Introduction to Deep Learning

1M views • 1 year ago

Alexander Amini

MIT Introduction to **Deep Learning** 6.S191: Lecture 1 Foundations of **Deep Learning** Lecturer: Amini January 2020 For ...

CC

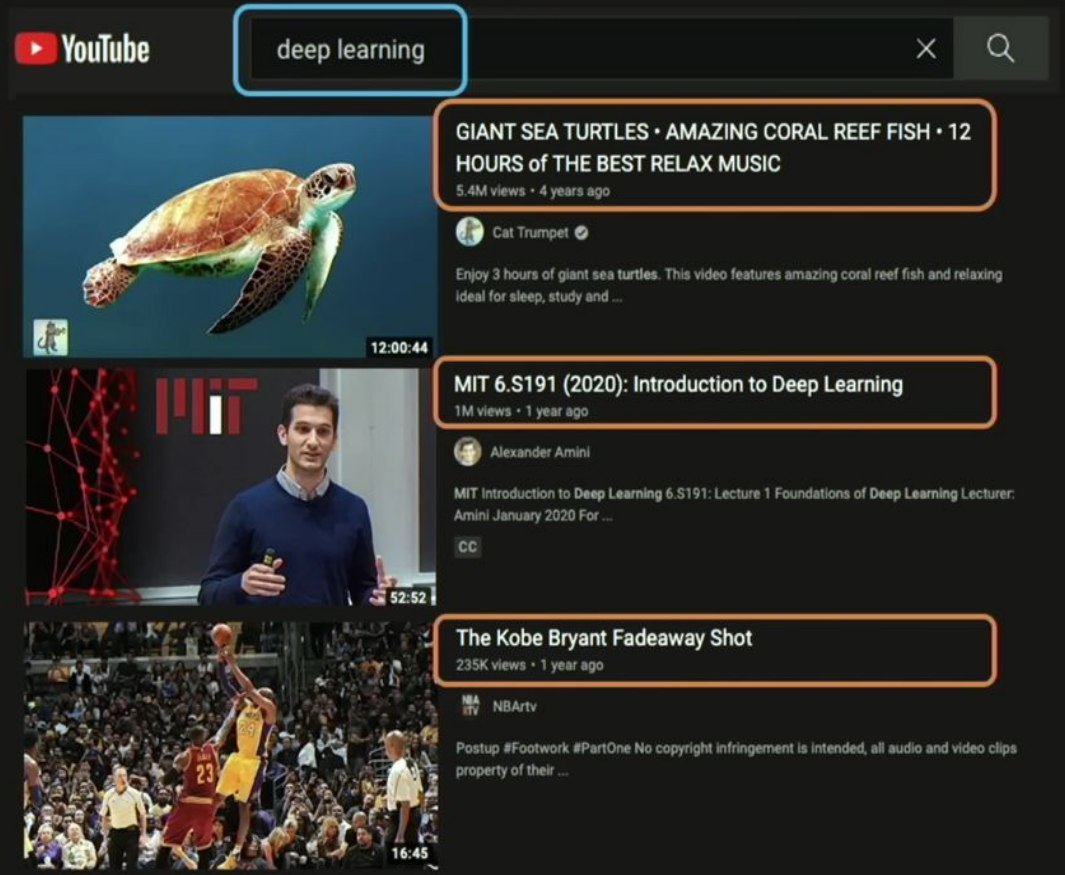
The Kobe Bryant Fadeaway Shot

235K views • 1 year ago

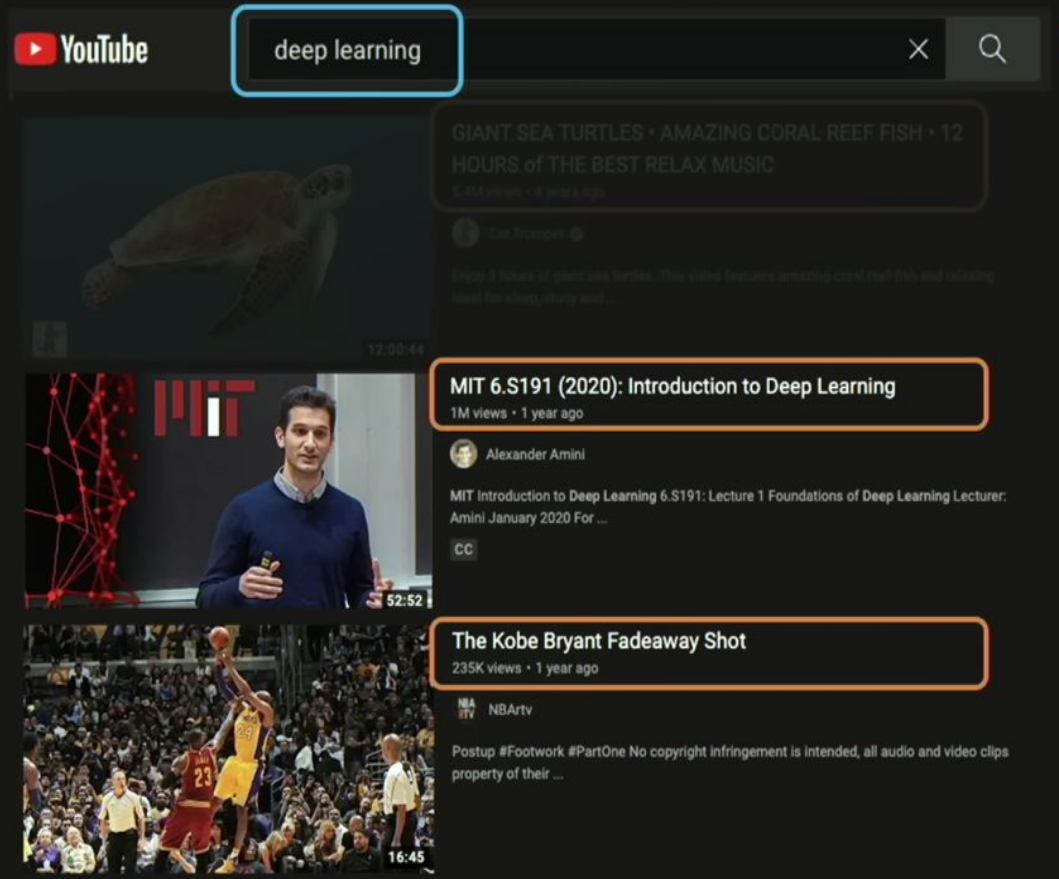
NBArtv

Postup #Footwork #PartOne No copyright infringement is intended, all audio and video clips property of their ...

Understanding Attention with Search



Understanding Attention with Search



Query (Q)

Key (K_1)

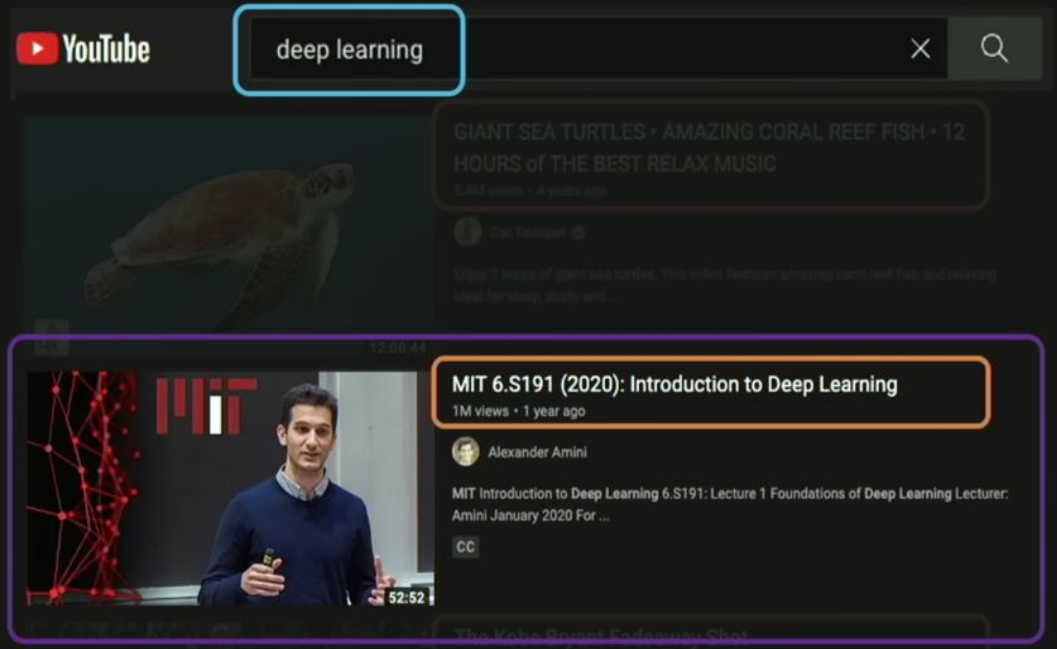
Key (K_2)

Key (K_3)

How similar is the key to the query?

1. **Compute attention mask:** how similar is each key to the desired query?

Understanding Attention with Search



Query (Q)

Key (K_1)

Key (K_2)

Value (V)

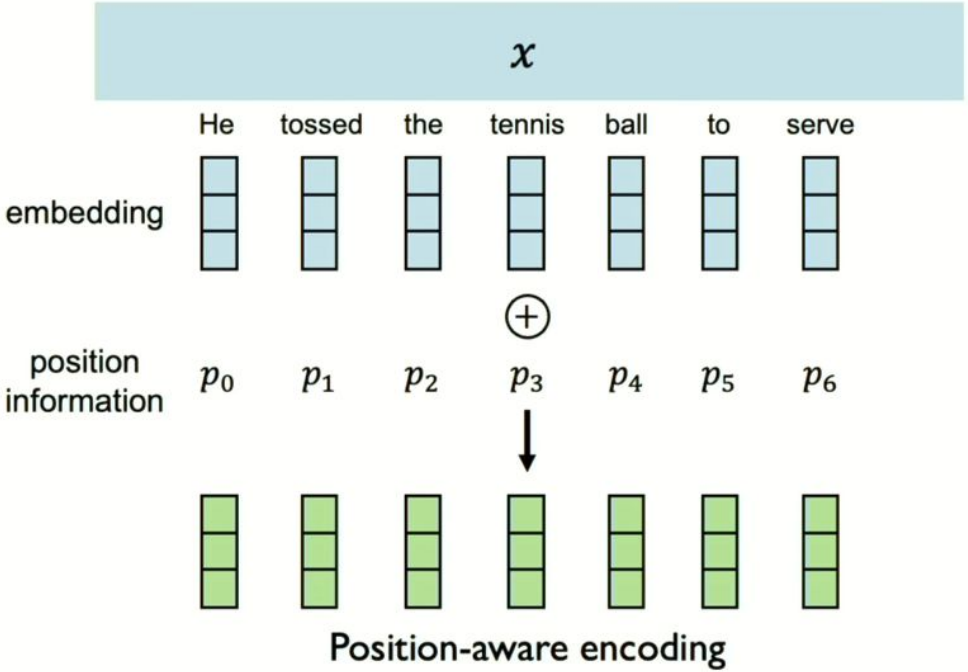
Key (K_3)

2. **Extract values based on attention:**
Return the values highest attention

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

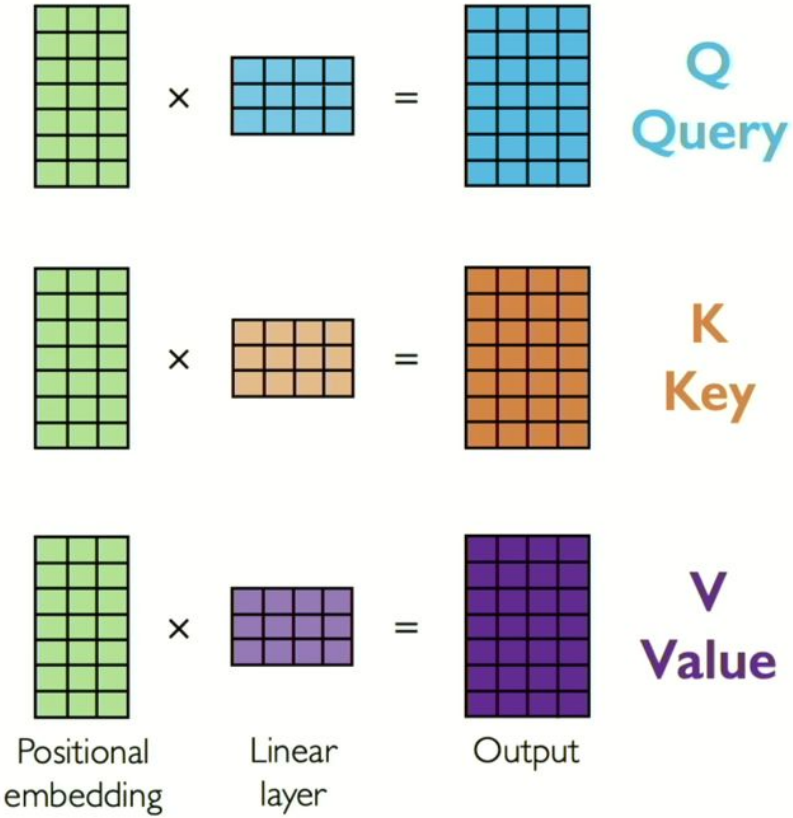


Data is fed in all at once! Need to encode position information to understand order.

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute attention weighting
4. Extract features with high attention



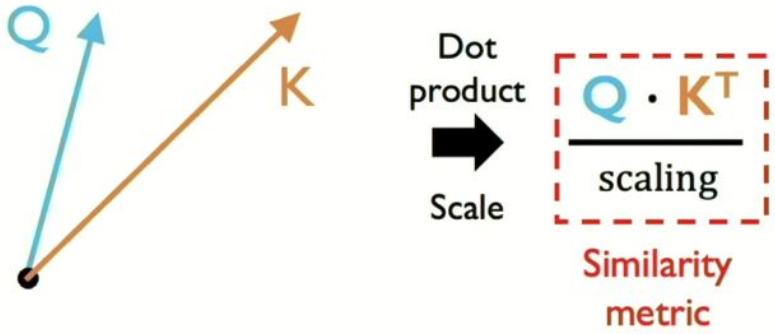
Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention score: compute pairwise similarity between each query and key

How to compute similarity between two sets of features?



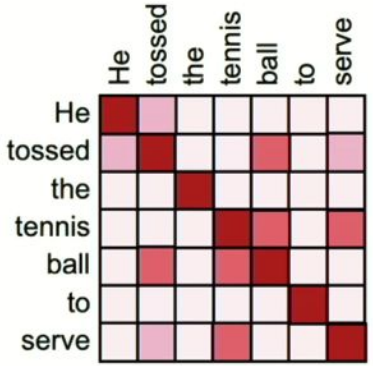
Also known as the "cosine similarity"

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention weighting: where to attend to! How similar is the key to the query?



$$softmax\left(\frac{Q \cdot K^T}{scaling}\right)$$

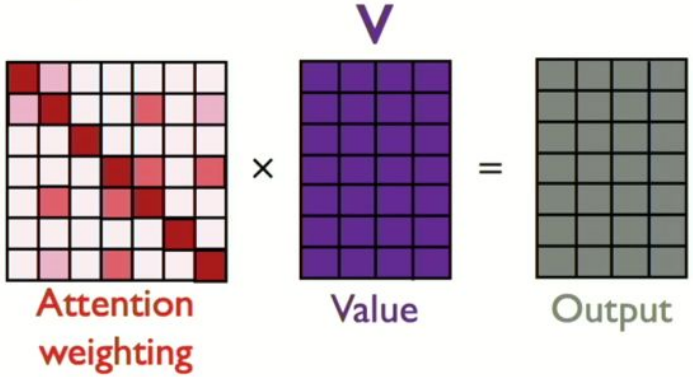
Attention weighting

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Last step: self-attend to extract features



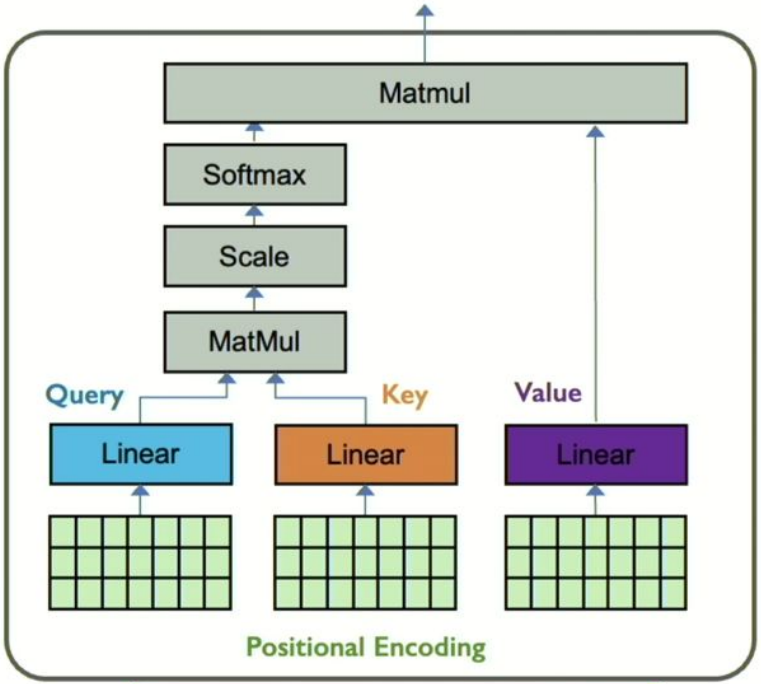
$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V = A(Q, K, V)$$

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

These operations form a self-attention head that can plug into a larger network. Each head attends to a different part of input.



$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V$$

Applying Multiple Self-Attention Heads

Attention weighting

×

Value

=

Output

Self-Attention Applied

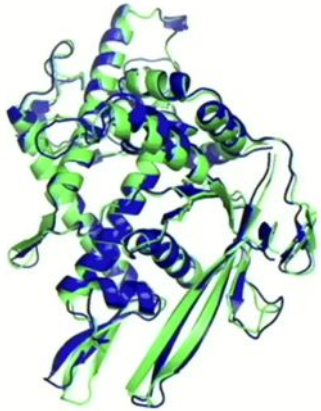
Language Processing

An armchair in the shape of an avocado

Transformers: BERT, GPT

Devlin et al., *NAACL* 2019
Brown et al., *NeurIPS* 2020

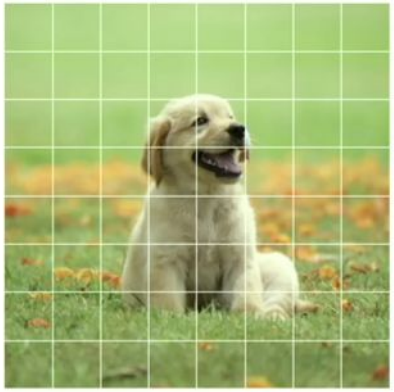
Biological Sequences



Protein Structure Models

Jumper et al., *Nature* 2021
Lin et al., *Science* 2023

Computer Vision



Vision Transformers

Dosovitskiy et al., *ICLR* 2020

Підсумуємо, що ми вивчили з використання DL для моделювання послідовностей

1. **RNN** добре підходять для задач **моделювання послідовностей**

2. Ми моделюємо послідовності через **рекурентні** відношення

3. Навчання RNN відбувається за допомогою **зворотного поширення помилки у часі** (backpropagation through time)

4. Моделі обробки послідовності використовуються для генерації музики, класифікації текстів, машинного перекладу та іншого.

5. **Self-attention** - це підхід до роботи з послідовностями для моделювання послідовностей без рекурентності

6. Self-attention є основою багатьох великих мовних моделей як **GPT, BERT** та інших.

Deep Learning for Sequence Modeling: Summary

1. RNNs are well suited for **sequence modeling** tasks
2. Model sequences via a **recurrence relation**
3. Training RNNs with **backpropagation through time**
4. Models for **music generation**, classification, machine translation, and more
5. Self-attention to model **sequences without recurrence**
6. Self-attention is the basis for many **large language models** – stay tuned!

To exit full screen, press Esc

6.S191: Introduction to Deep Learning

Lab 1: Introduction to TensorFlow and Music Generation with RNNs

Link to download labs:
http://introtodeeplearning.com#schedule

1. Open the lab in Google Colab
2. Start executing code blocks and filling in the #TODOs
3. Need help? Find a TA/instructor!

Deep Learning

IntroToDeepLearning.com

SUBSCRIBE