11210IPT553000
Deep Learning in Biomedical Optical Imaging

Week 9
Sequence Model
RNN, LSTM, Self-Attention, and Transformer

Instructor: Hung-Wen Chen
2023/11/06 @NTHU, Fall 2023

SPONSORED BY

coursera deeplearning.ai NVIDIA DEEP LEARNING INSTITUTE

NVIDIA DEEP LEARNING INSTITUTE CERTIFIED INSTRUCTOR

Calendar

9	11/06	*Transformer*	*Self-attention*	*Report*
10	11/13	*AI for Medical Diagnosis + Nvidia Workshop*		*HW5*
11	11/20	*Mid-term Review + Introduction to Final Presentation*		
12	11/27	*Mid-term Exam*		
13	12/04	*Guest Lecture*		
14	12/11	*Guest Lecture*		
15	12/18	*Student Projects Presentation*		
16	12/25	*Student Projects Presentation*		
17	01/01	*Holiday - New Year's Day (no class)*		
18	01/08	*Student Projects Presentation*		

Outline of Today's Lecture

- Sequence Models (Course 5)
  - Recurrent Neural Network
  - Natural Language Processing & Word Embeddings
  - Sequence Models & Attention Mechanism
  - Transformer Network

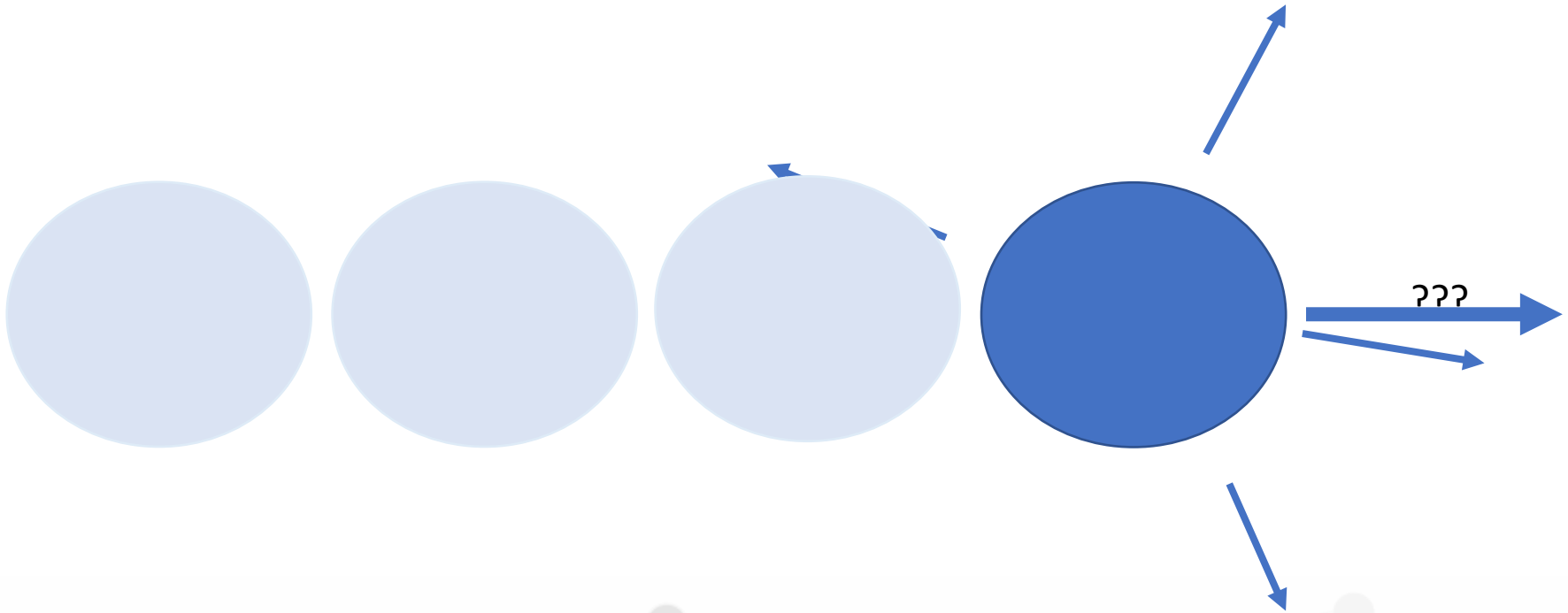
- Lab Practice: Self-Attention

Administrative

- Midterm Review on 11/20 and Midterm on 11/27
- Research Presentation Topic
  - Presentation Week (12/18, 12/25, 1/8)
  - Volunteers of the first week and randomly pick the presentation order on 11/20
  - Pick a research paper published after 2021
  - Send the title and the paper file to TA ASAP
  - Get the confirmation from the lecturer **due on 12/4**

Give an image of a ball,

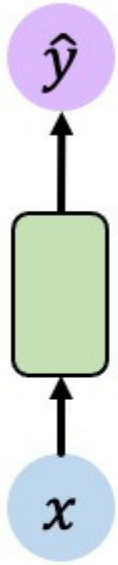
can you predict where it will go next?



Sequences in the Wild

aging

Sequence Modeling Applications

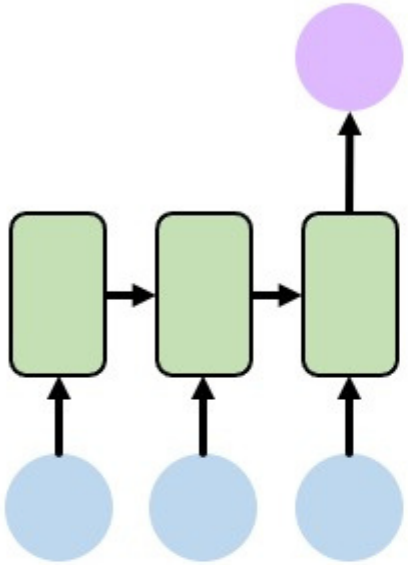


One to One

Binary Classification

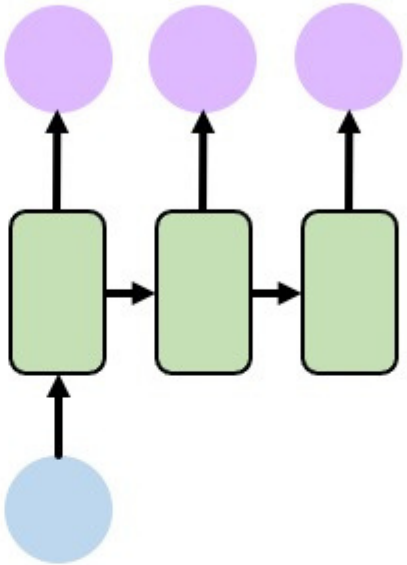
"Will I pass this class?"

Student → Pass?



Many to One

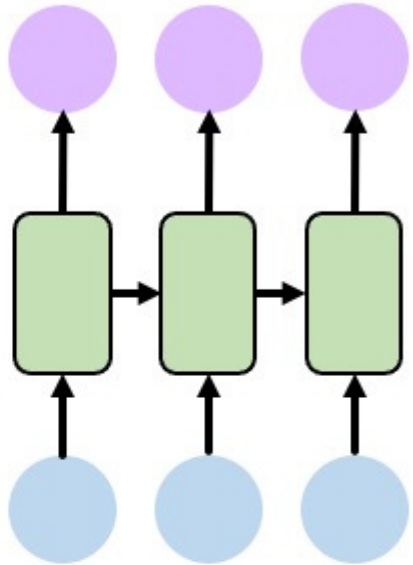
Sentiment Classification



One to Many

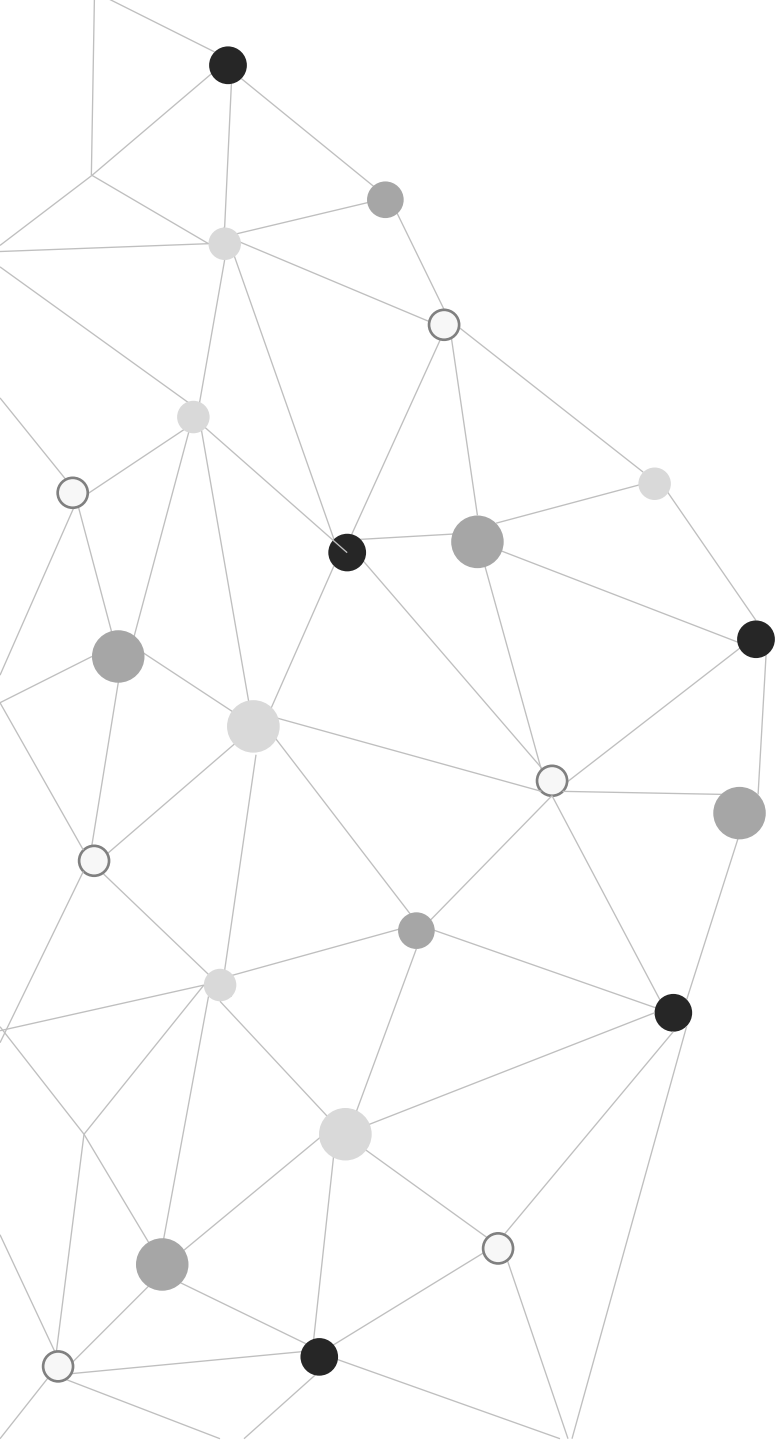
Image Captioning

"A baseball player throws a ball."

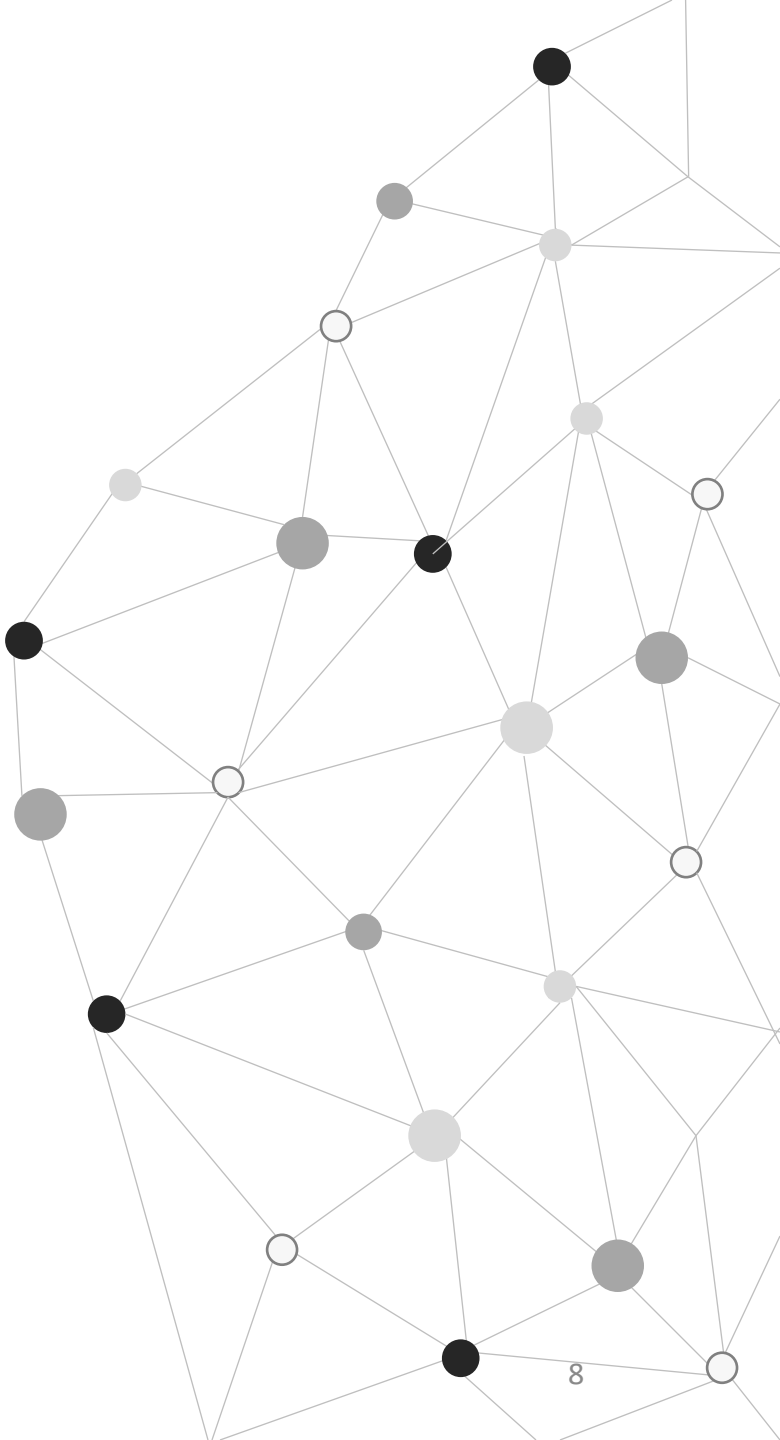


Many to Many

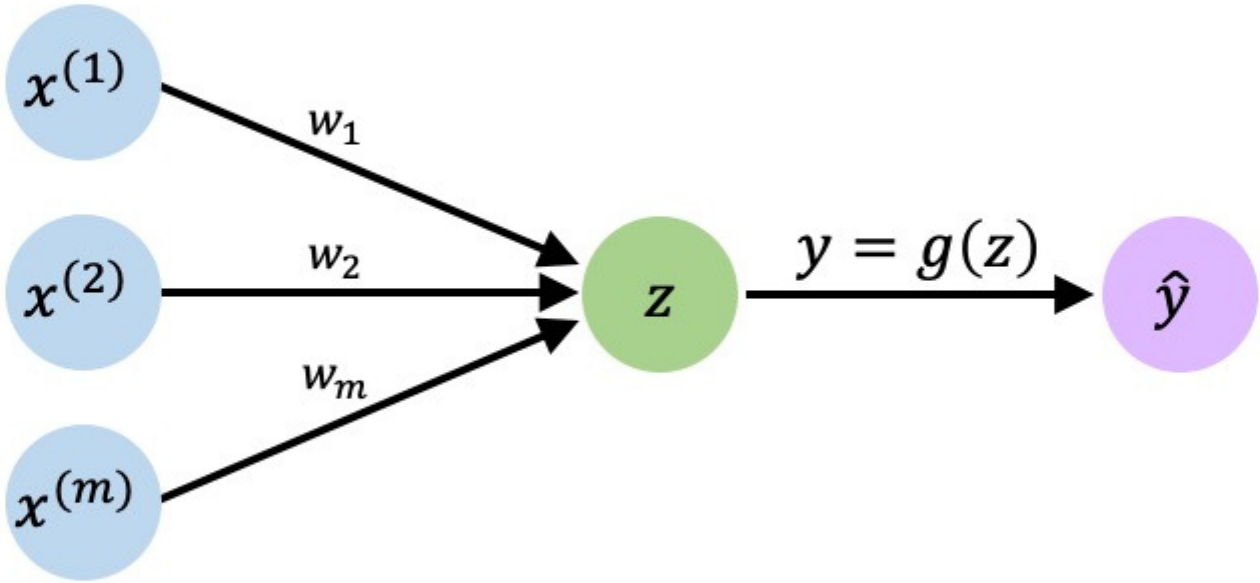
Machine Translation



Neurons with Recurrence



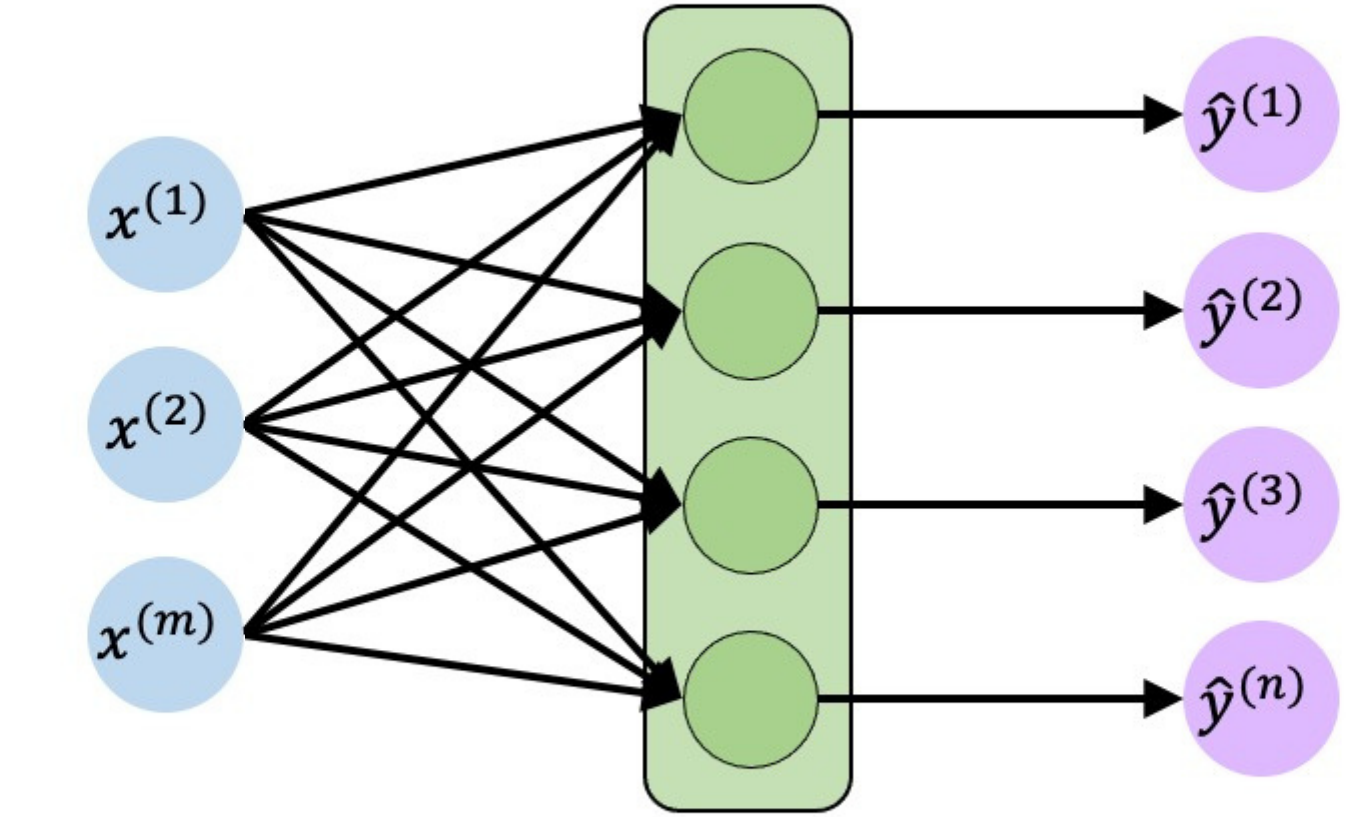
The Perception Revisited



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

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

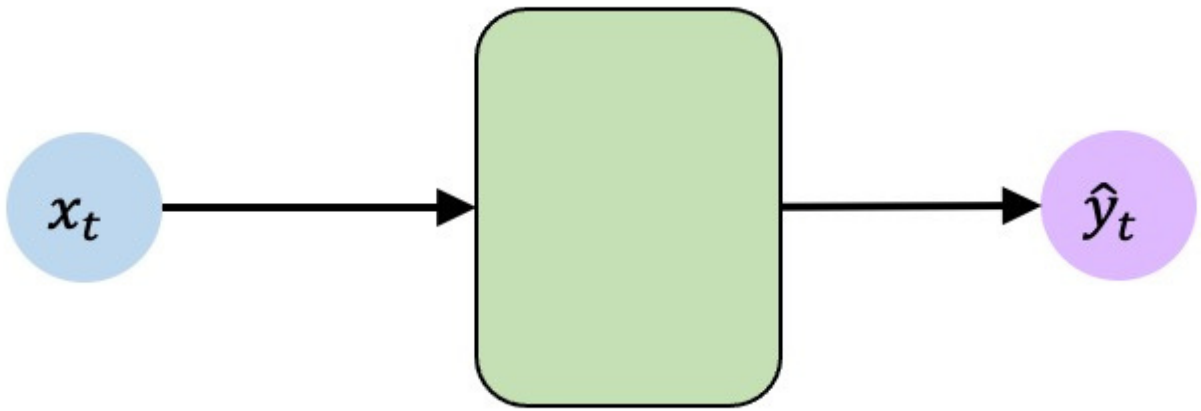
Feed-Forward Networks Revisited



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

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

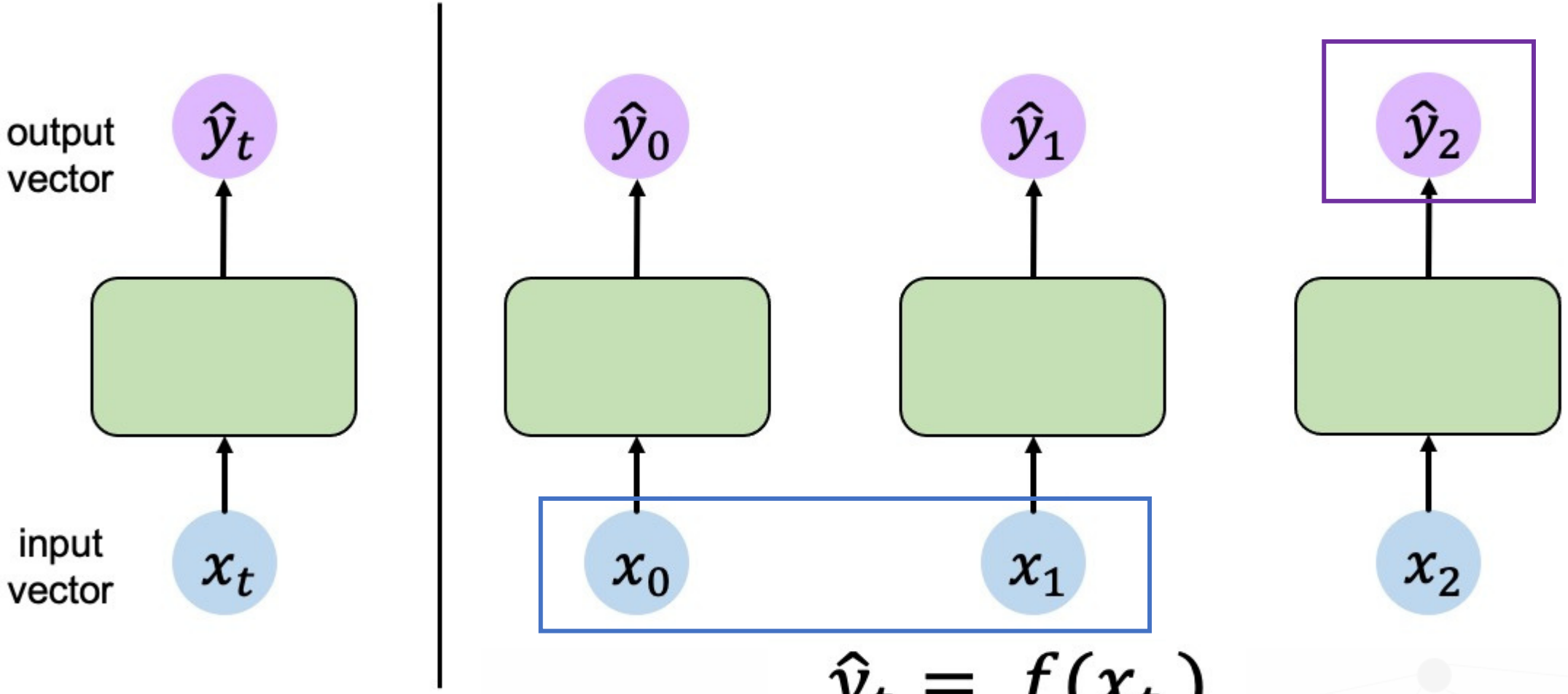
Feed-Forward Networks Revisited



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

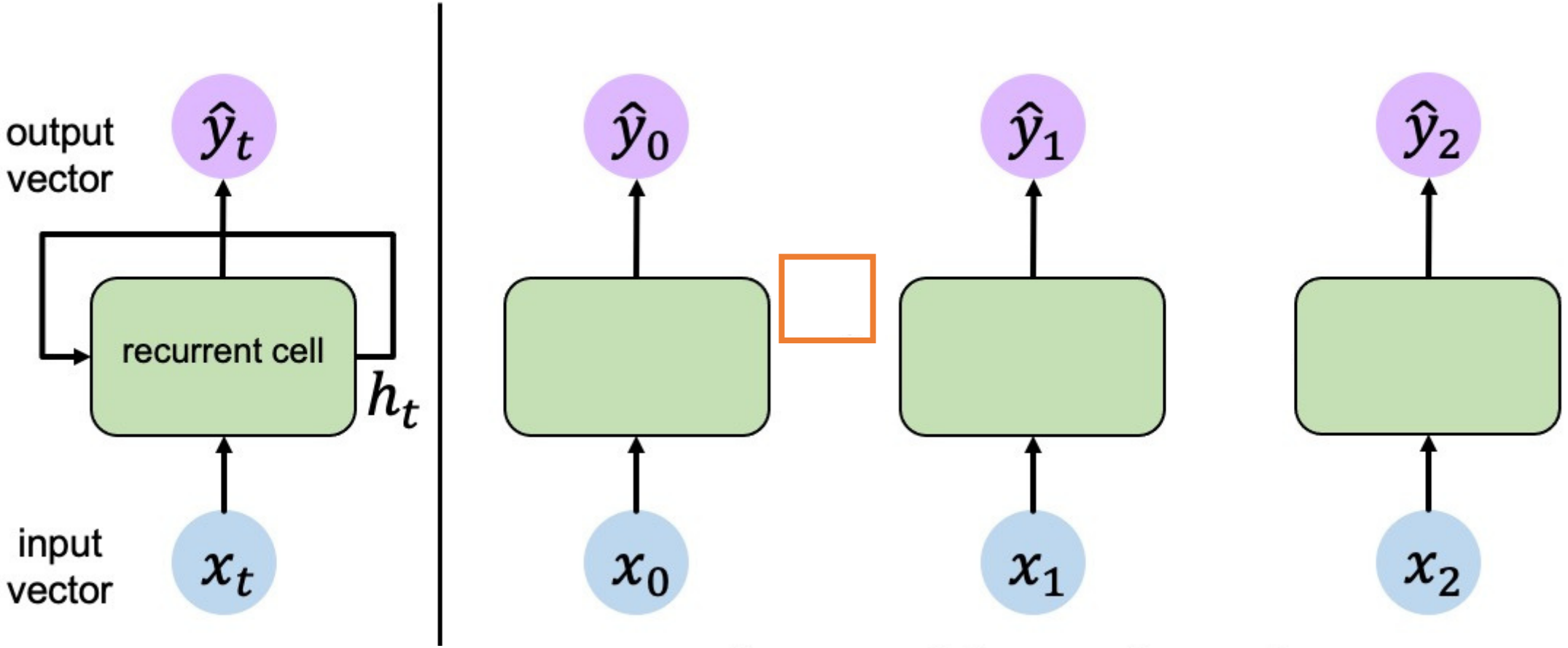
$$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$$

Handling Individual Time Steps

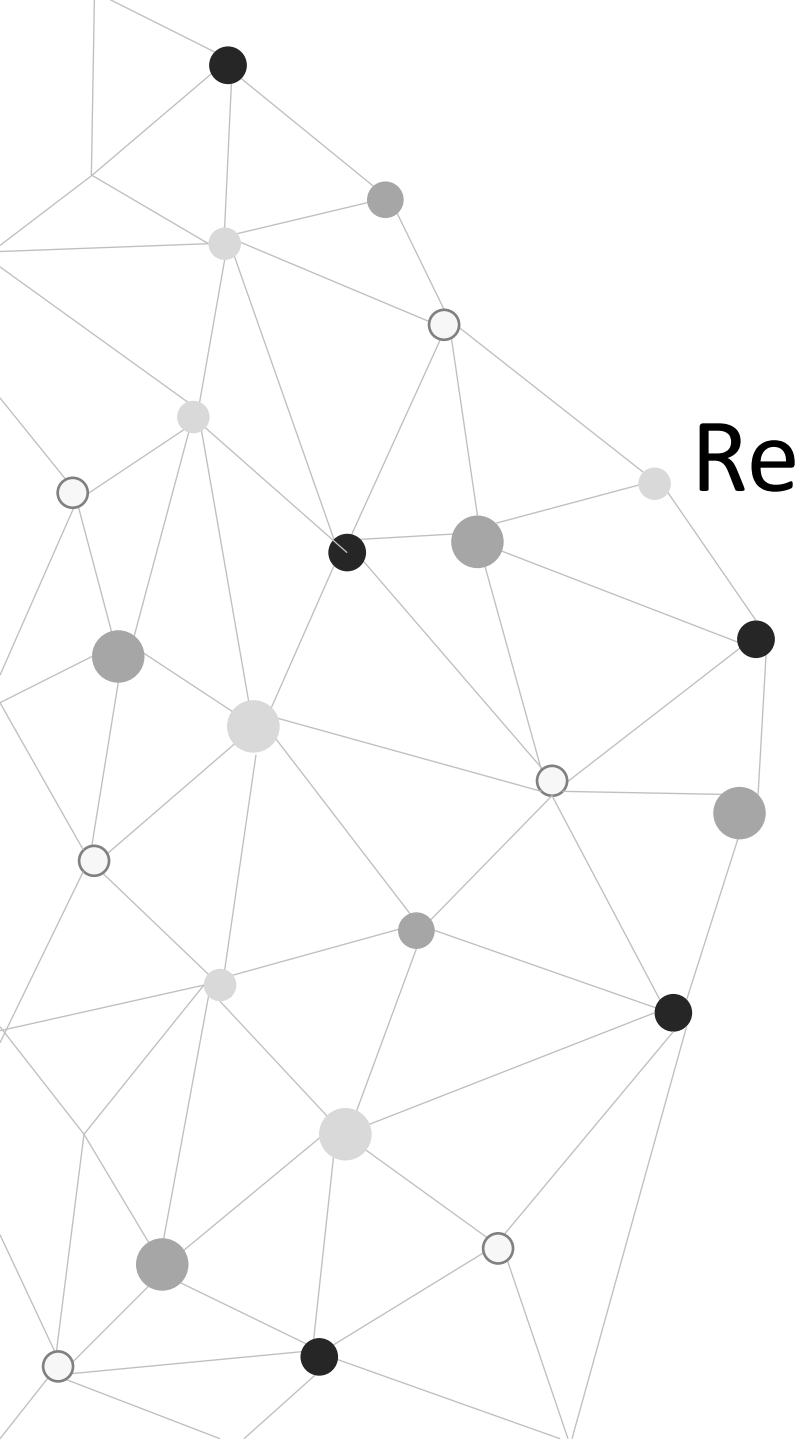


$$\hat{y}_t = f(x_t)$$

Neurons with Recurrence

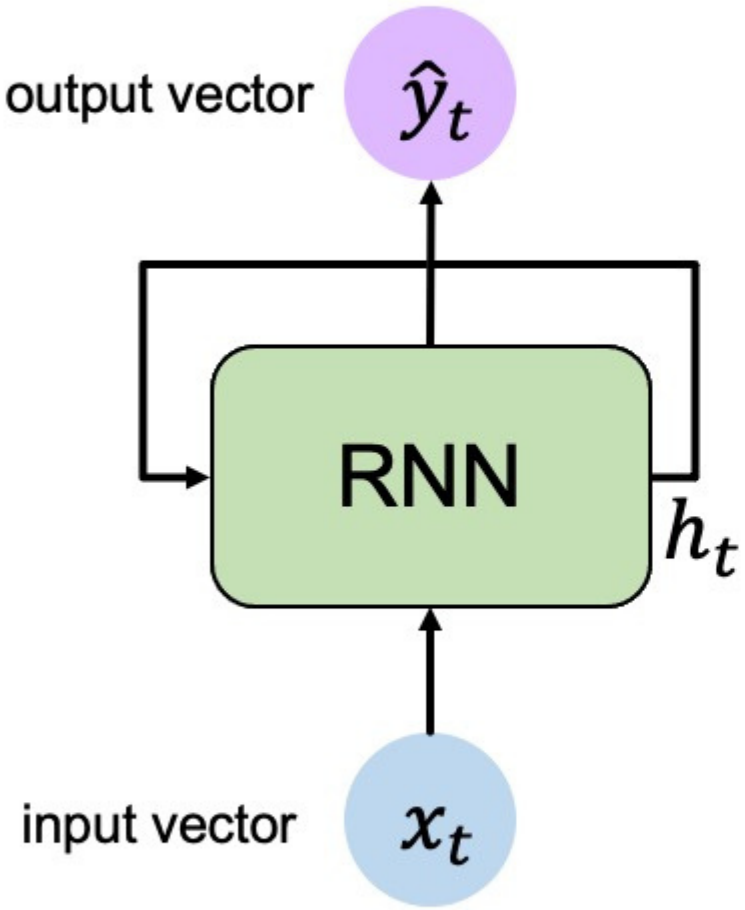


$$\hat{y}_t = f(x_t, h_{t-1})$$



Recurrent Neural Networks (RNNs)

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

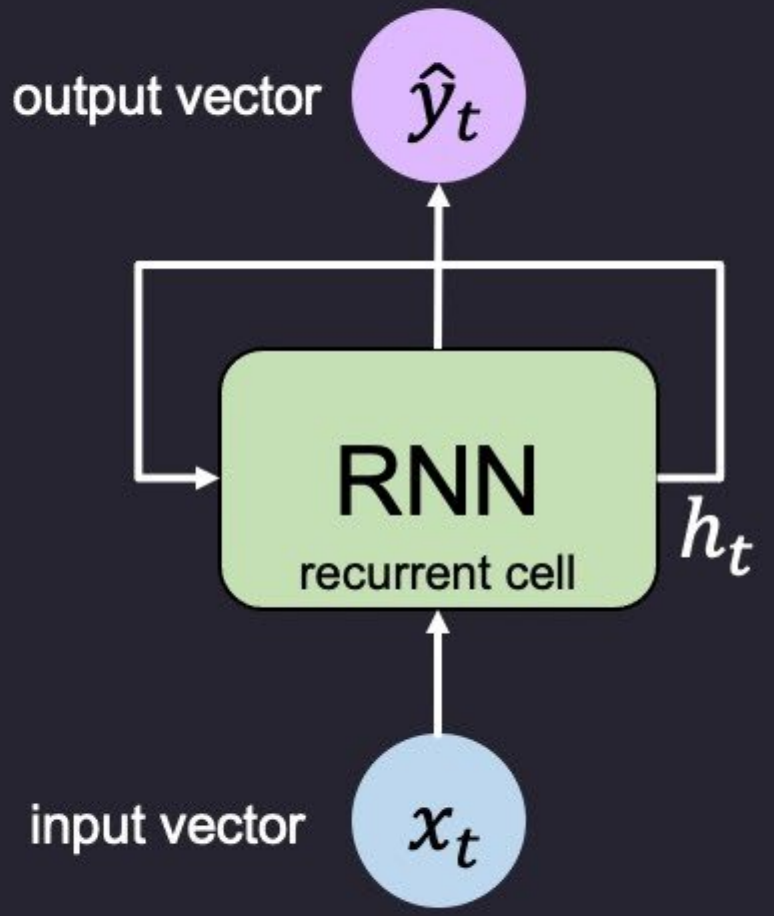
old state

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

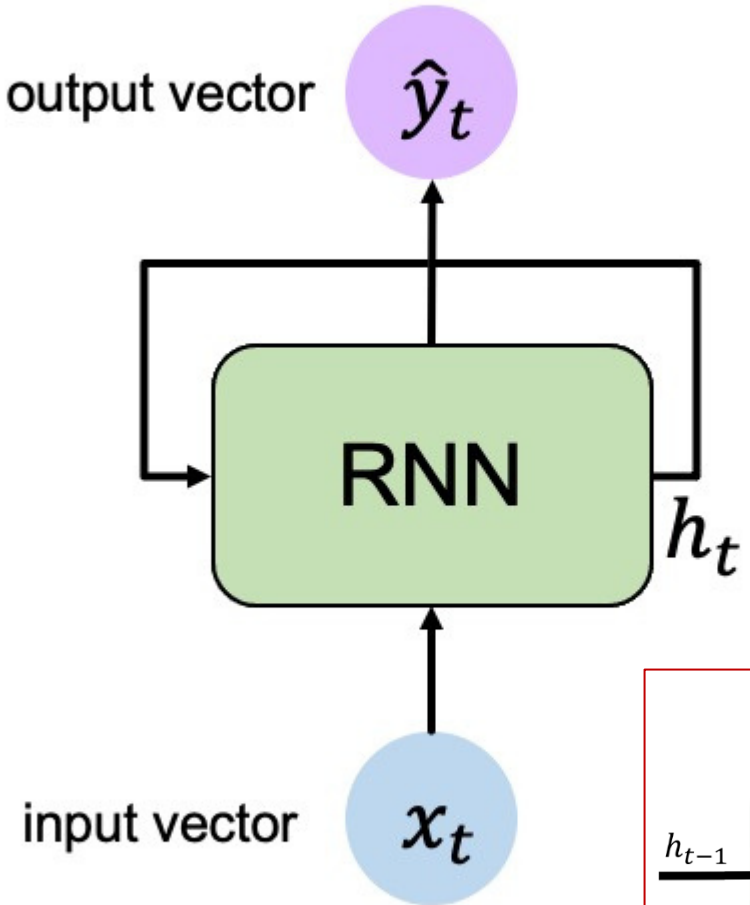
RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

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 State Update and Output

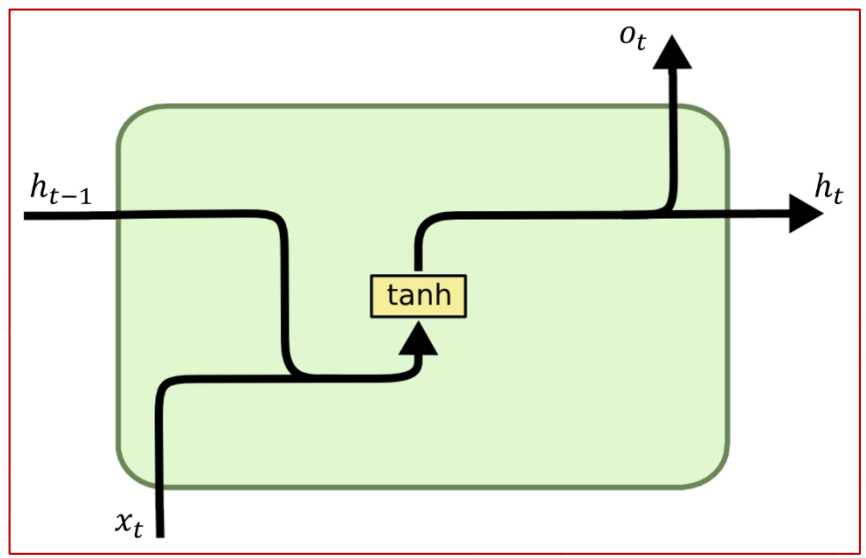


Output Vector

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

Update Hidden State

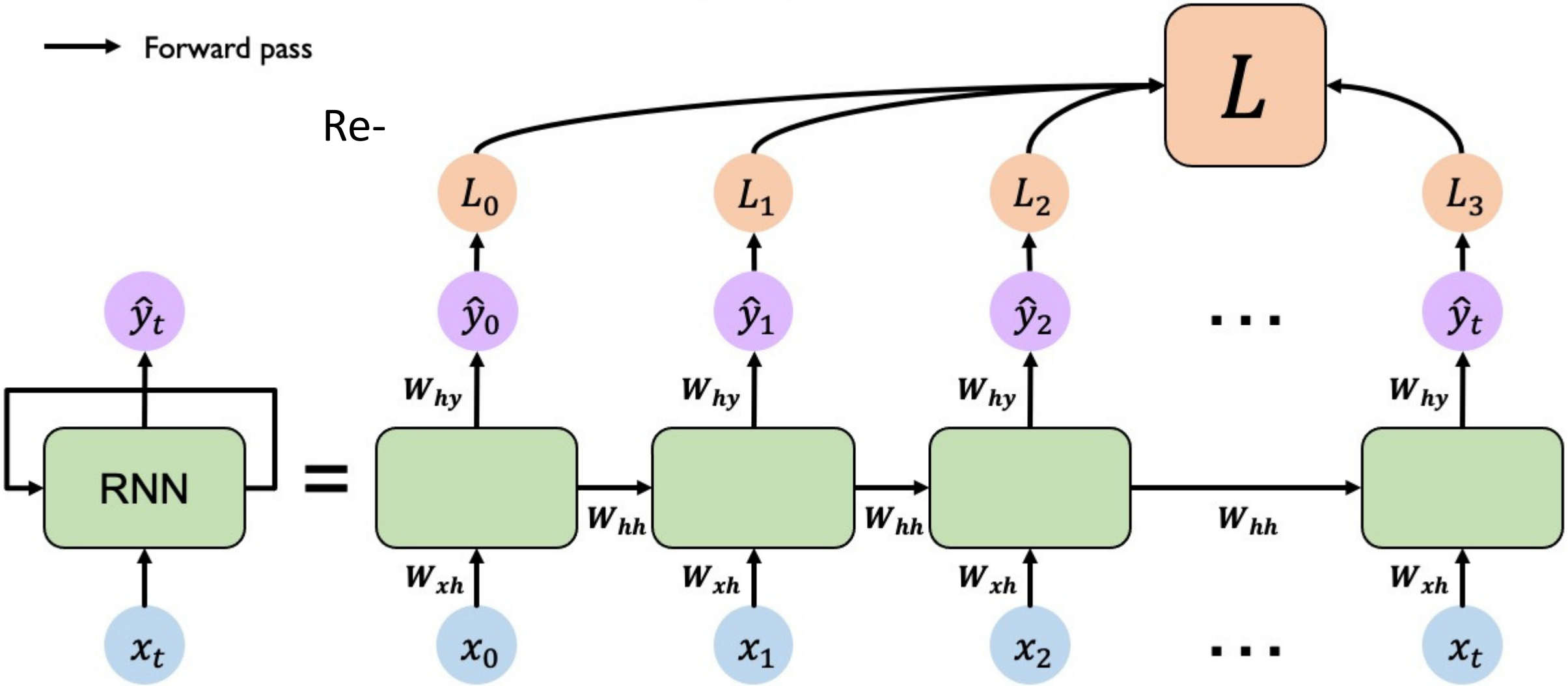
$$h_t = \tanh(\boldsymbol{W}_{hh}^{T} h_{t-1} + \boldsymbol{W}_{xh}^{T} x_t)$$



Input Vector

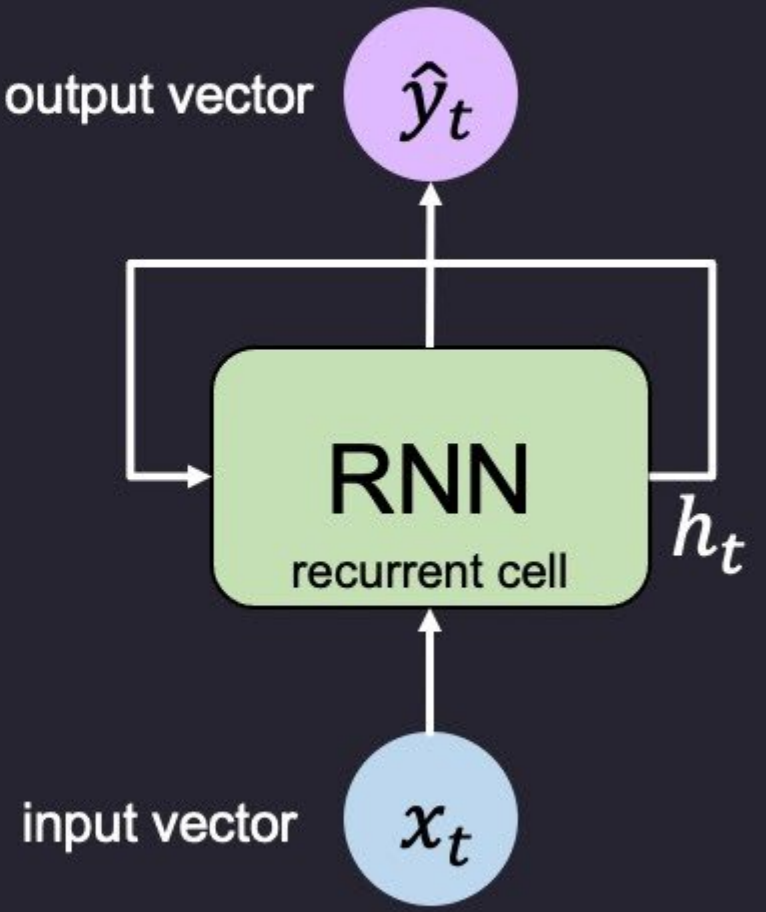
$$x_t$$

RNNs: Computational Graph Across Time

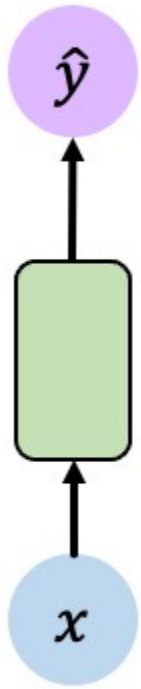


RNNs from Scratch

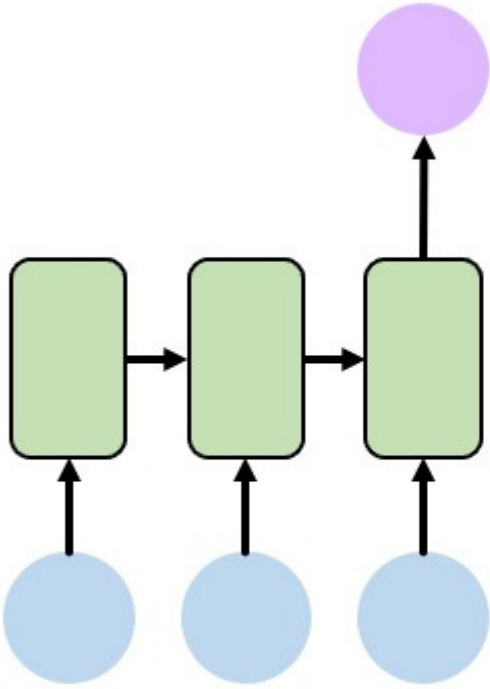
```
tf.keras.layers.SimpleRNN(rnn_units)
```



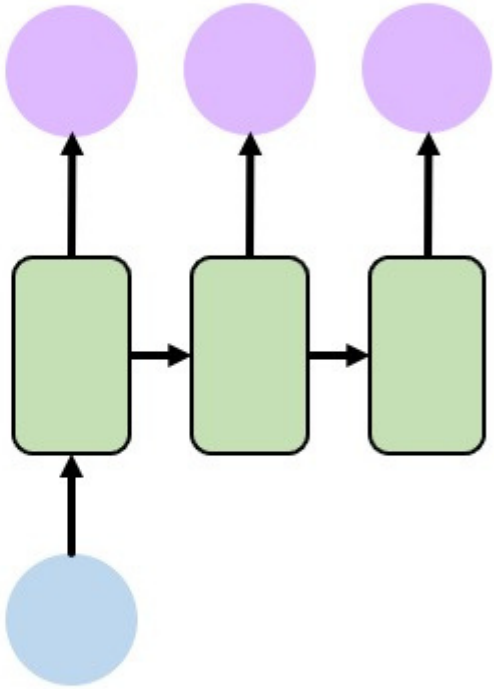
RNNs for Sequence Modeling



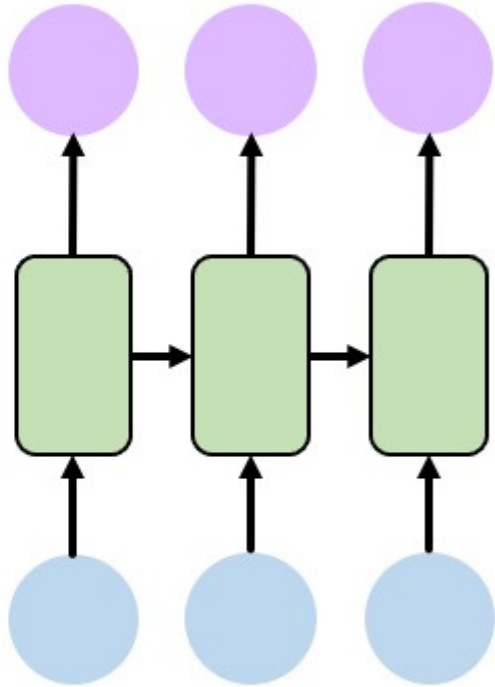
One to One
"Vanilla" NN
Binary classification



Many to One
Sentiment Classification



One to Many
Text Generation
Image Captioning



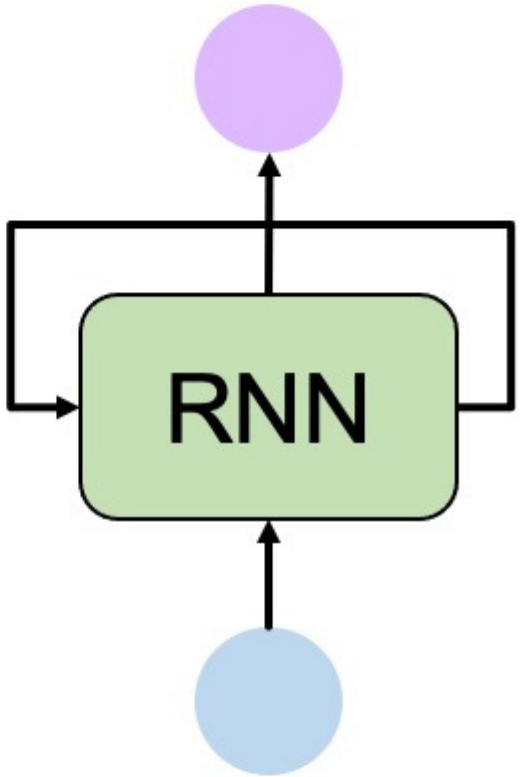
Many to Many
Translation & Forecasting
Music Generation

… and many other architectures and applications

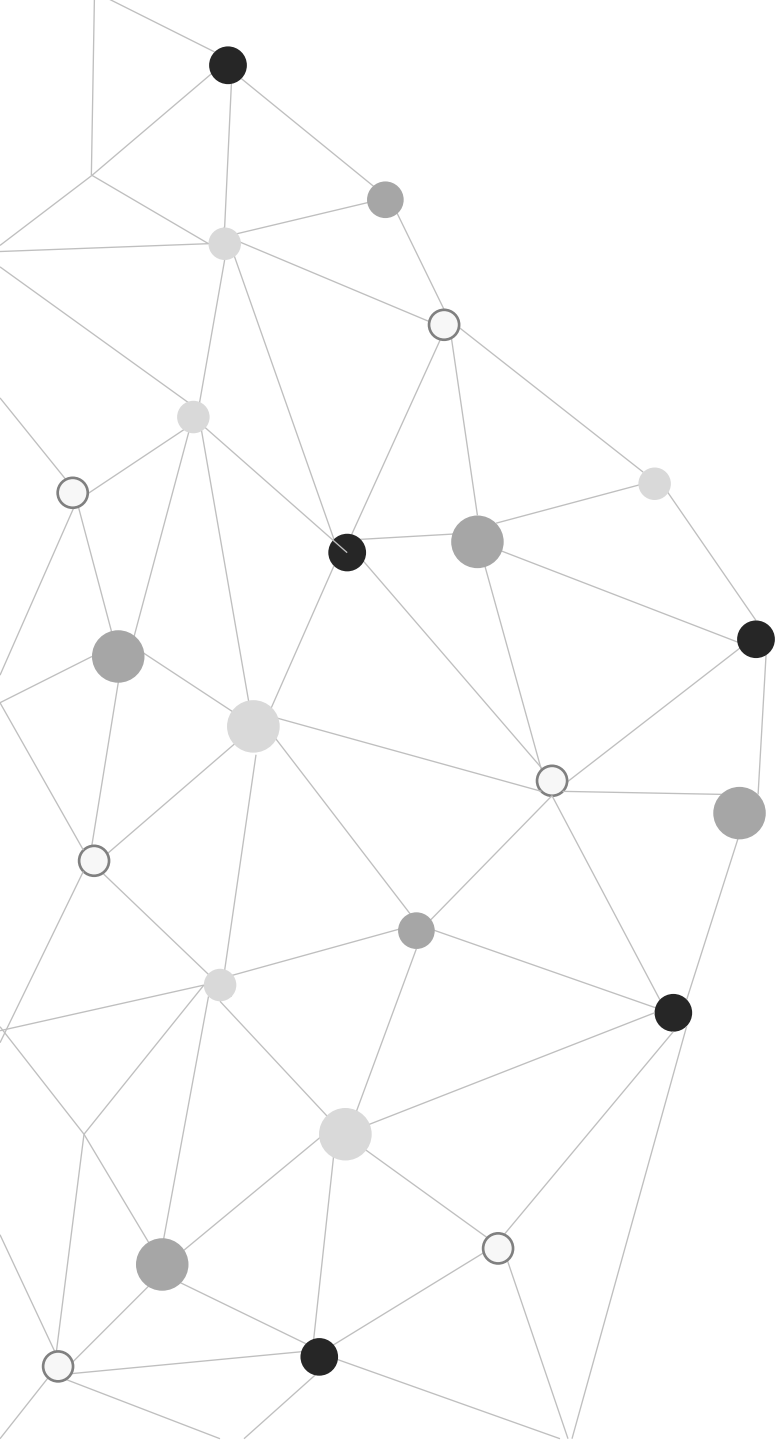
Sequence Modeling: Design Criteria

To model sequences, we need to:

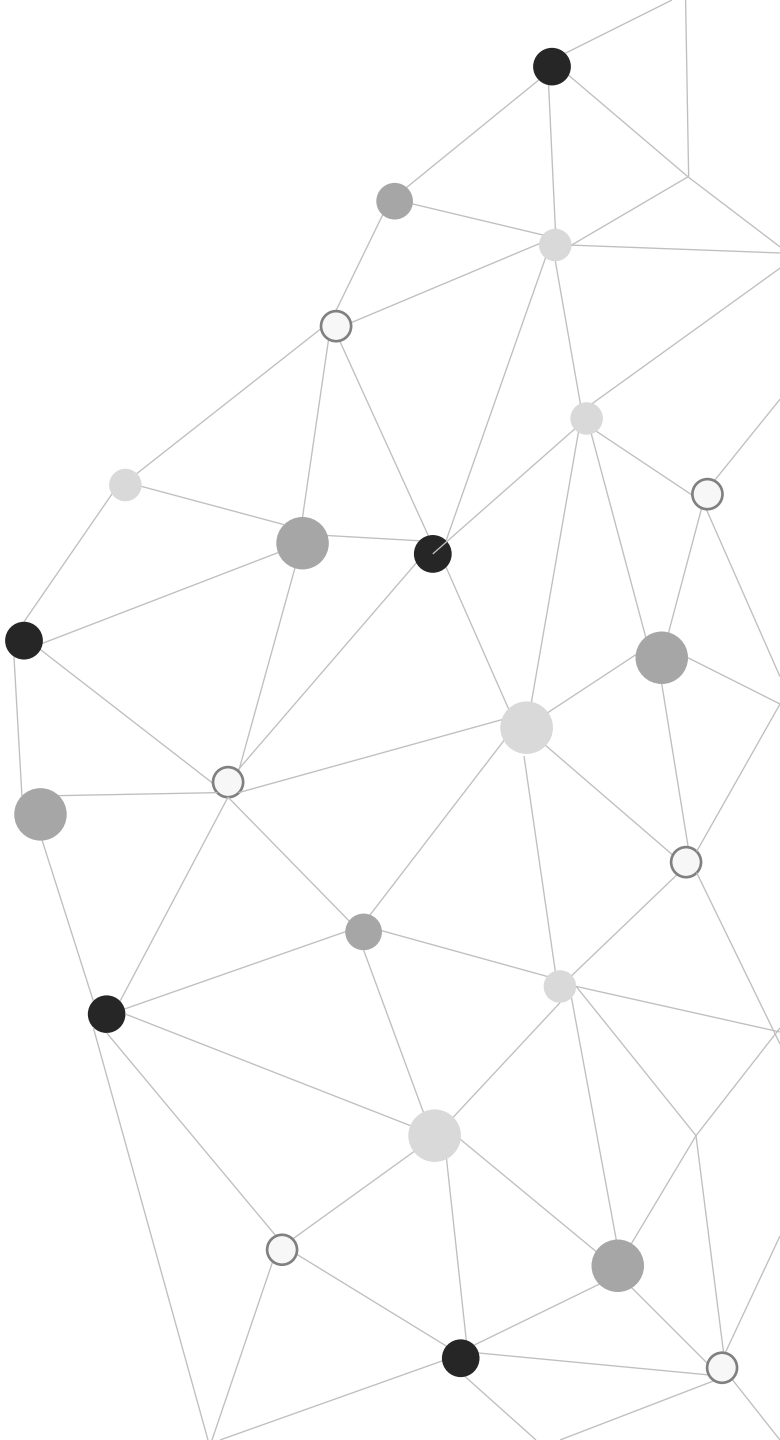
1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



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



A Sequence Modeling Problem: Predict the Next Word



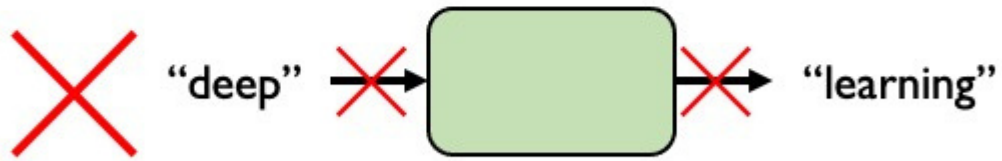
A Sequence Modeling Problem: Predict the Next Word

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

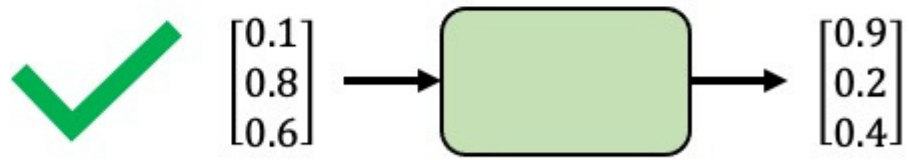
given these words

predict the next word

Representing Language to a Neural Network

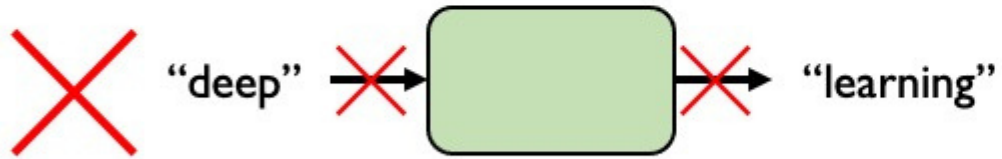


Neural networks cannot interpret words

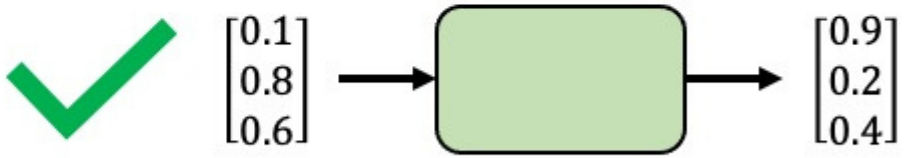


Neural networks require numerical inputs

Encoding Language for a Neural Network

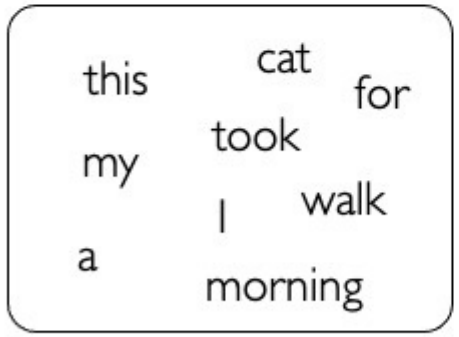


Neural networks cannot interpret words

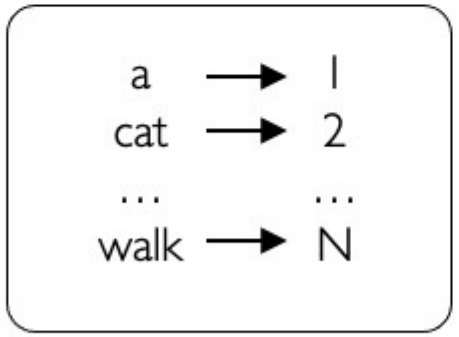


Neural networks require numerical inputs

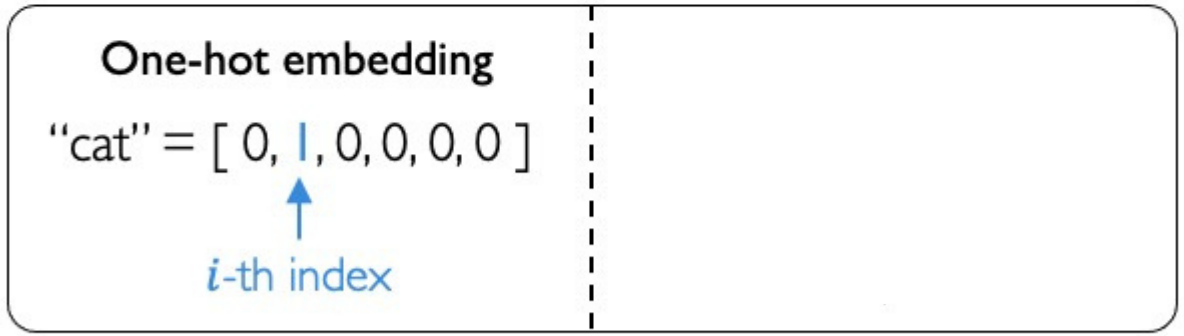
Embedding: transform indexes into a vector of fixed size.



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.

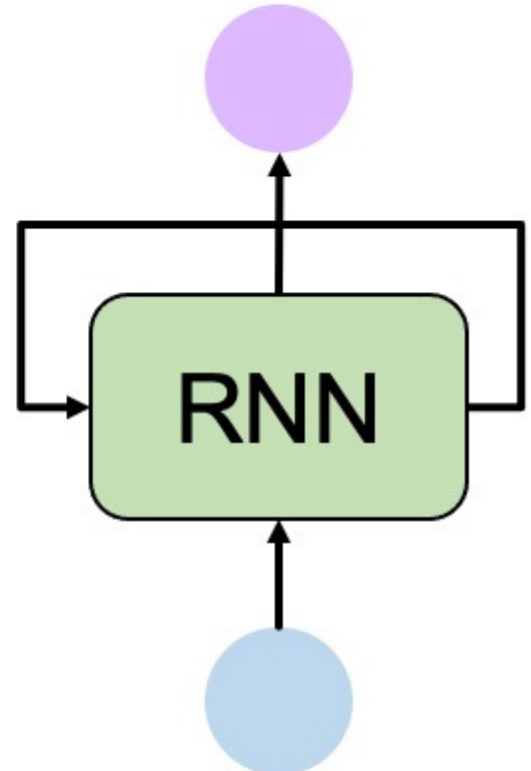
vs.

The food was bad, not good at all.

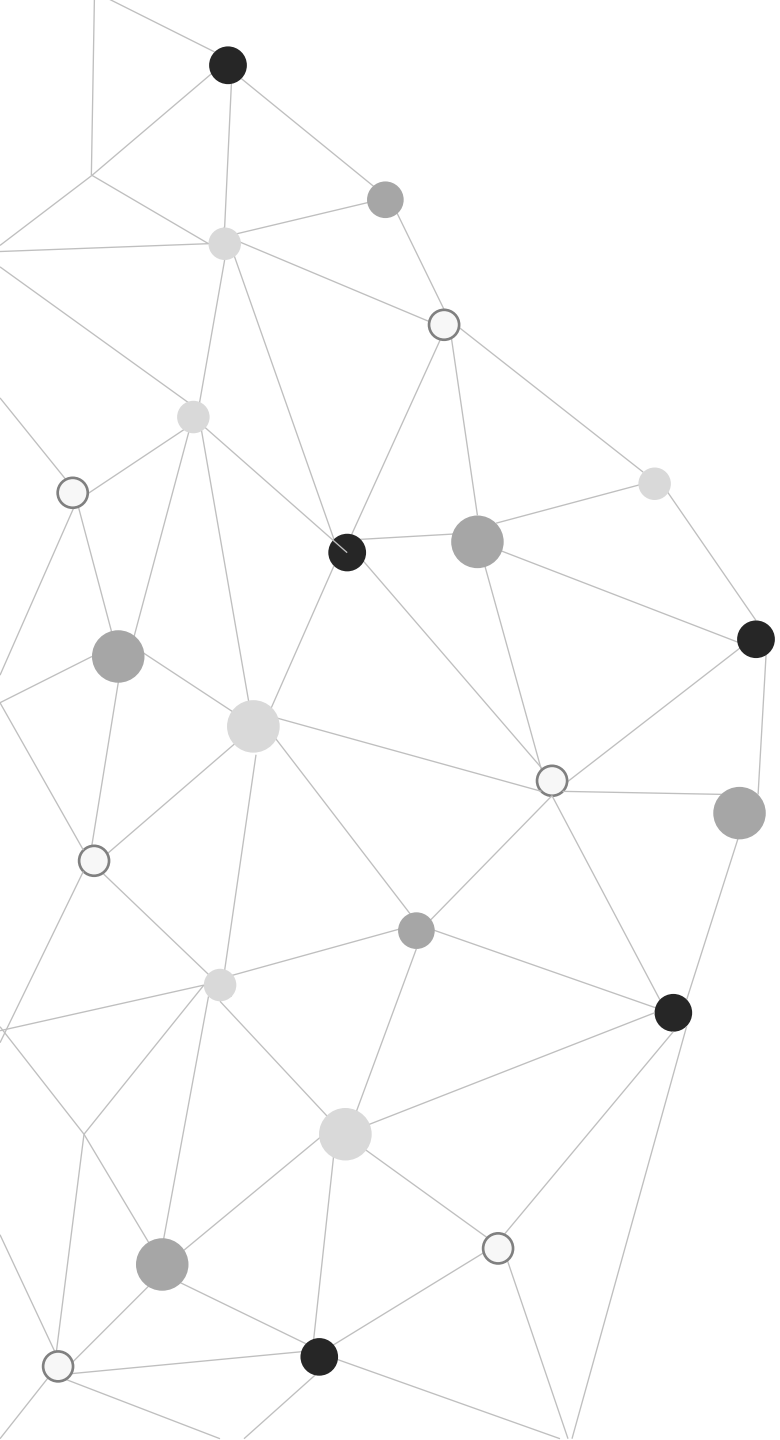
Sequence Modeling: Design Criteria

To model sequences, we need to:

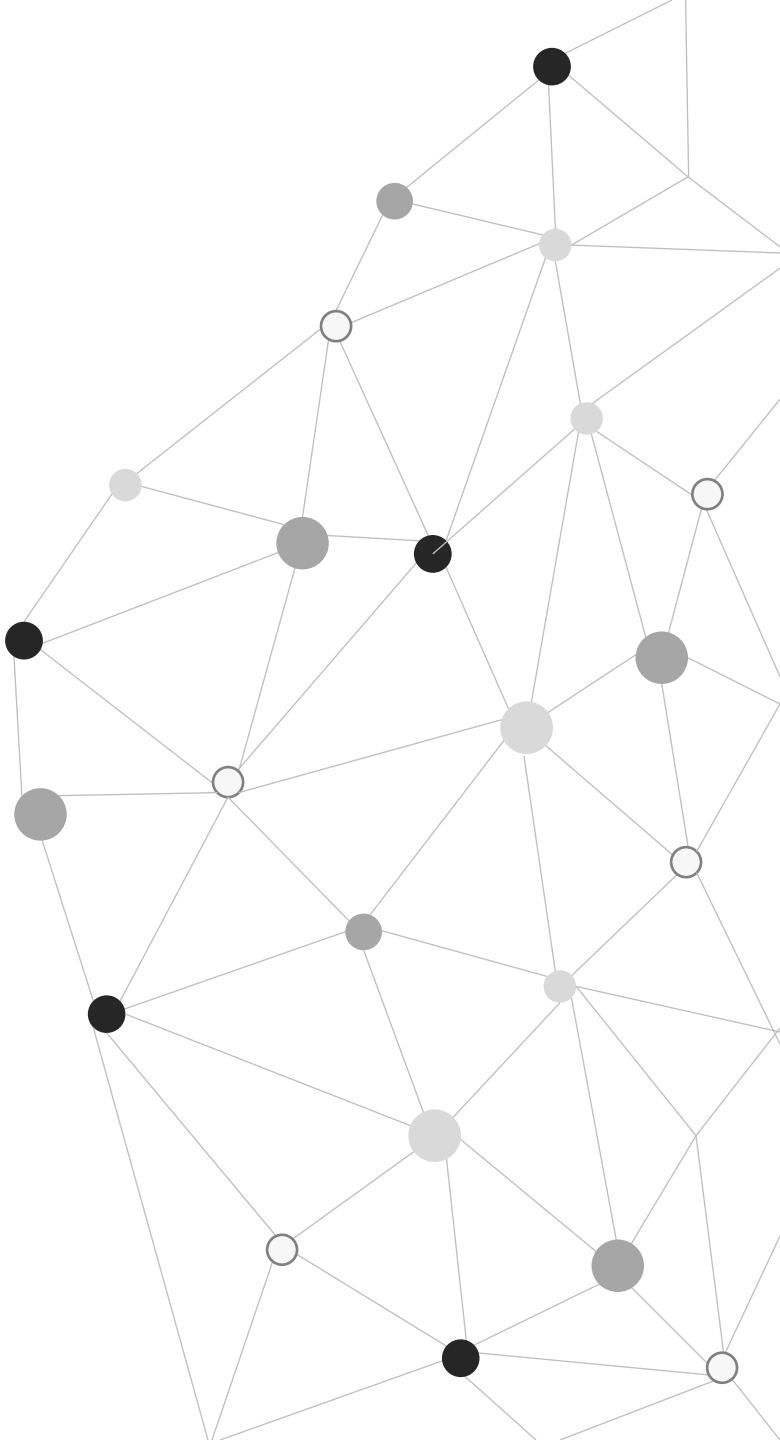
1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



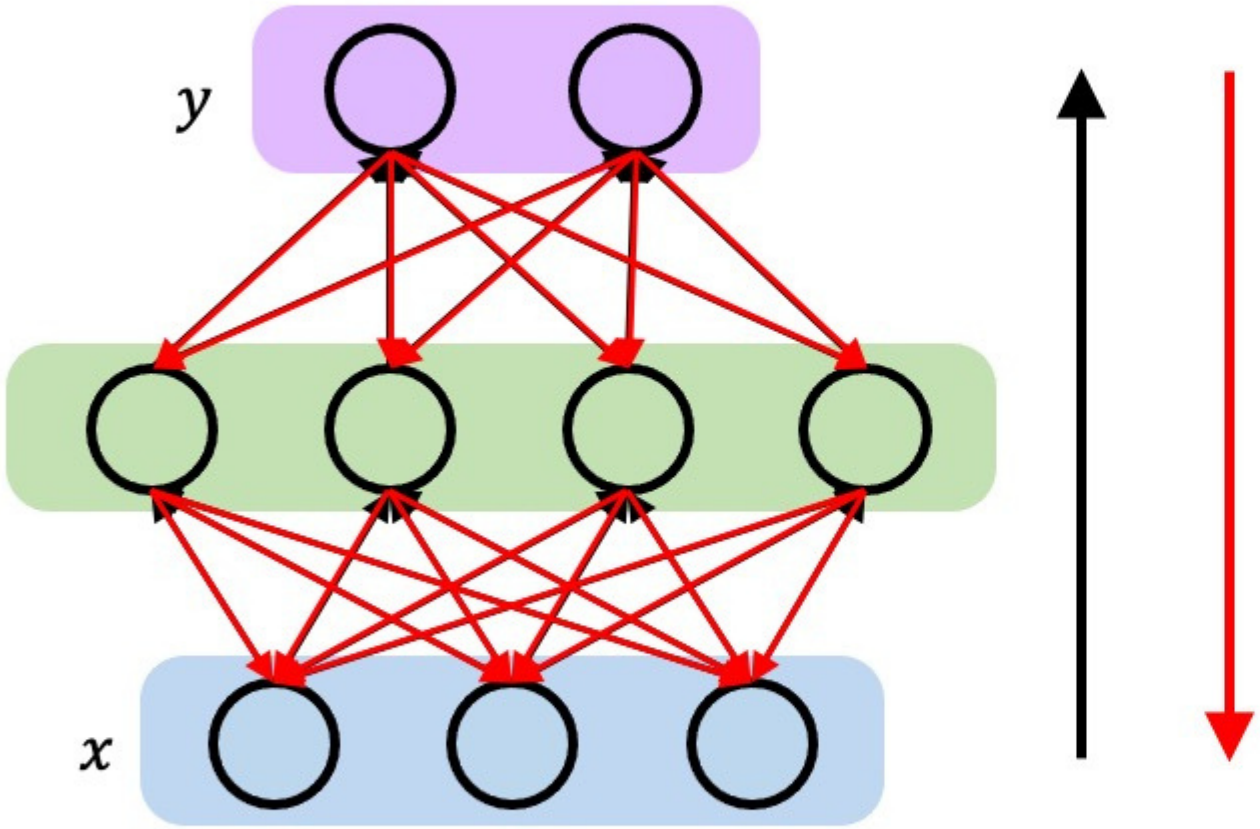
Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria



Backpropagation Through Time (BPTT)



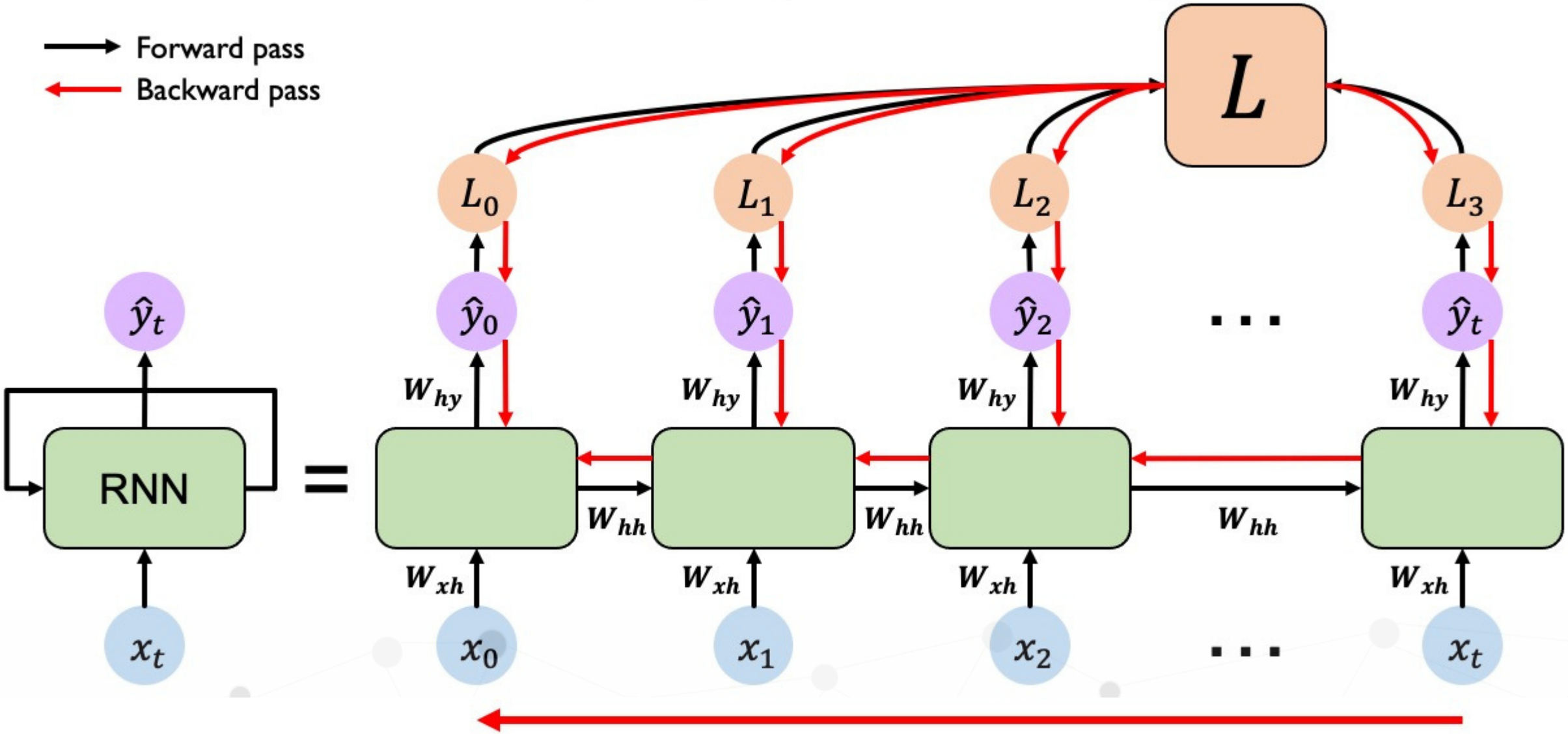
Recall: Backpropagation in Feed Forward Models



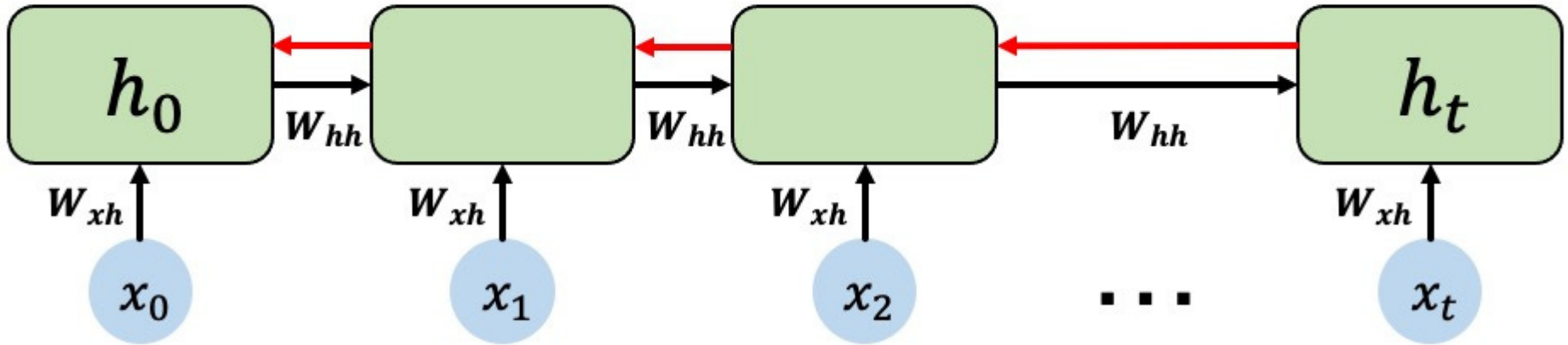
Backpropagation algorithm:

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



Standard RNN Gradient Flow



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

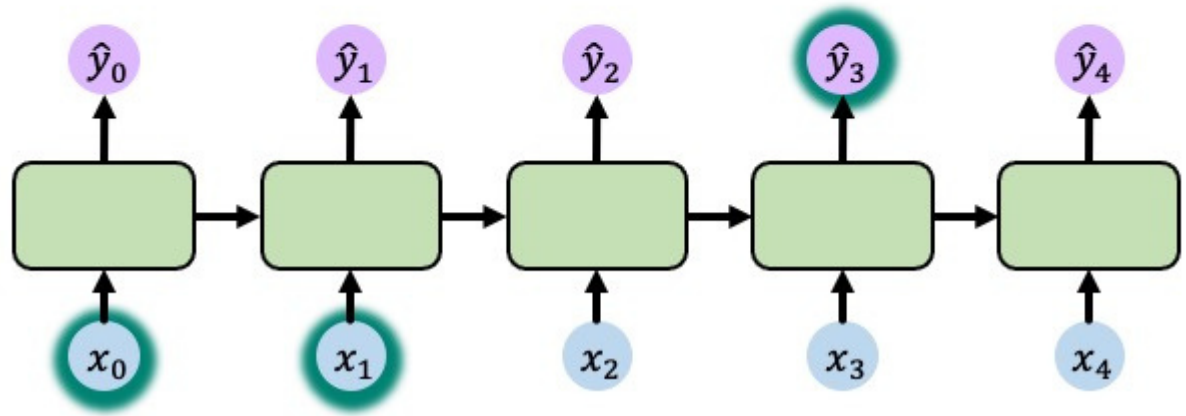
↓

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

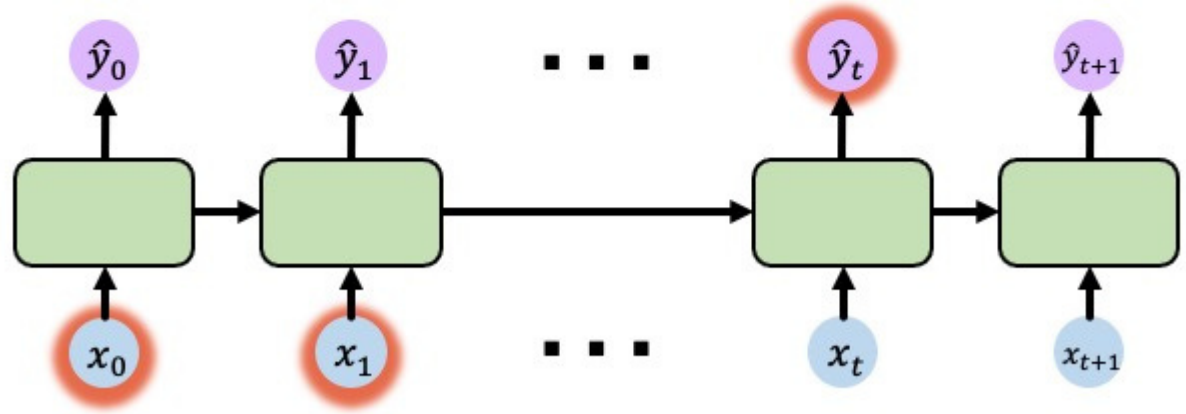
↓

Bias parameters to capture short-term dependencies

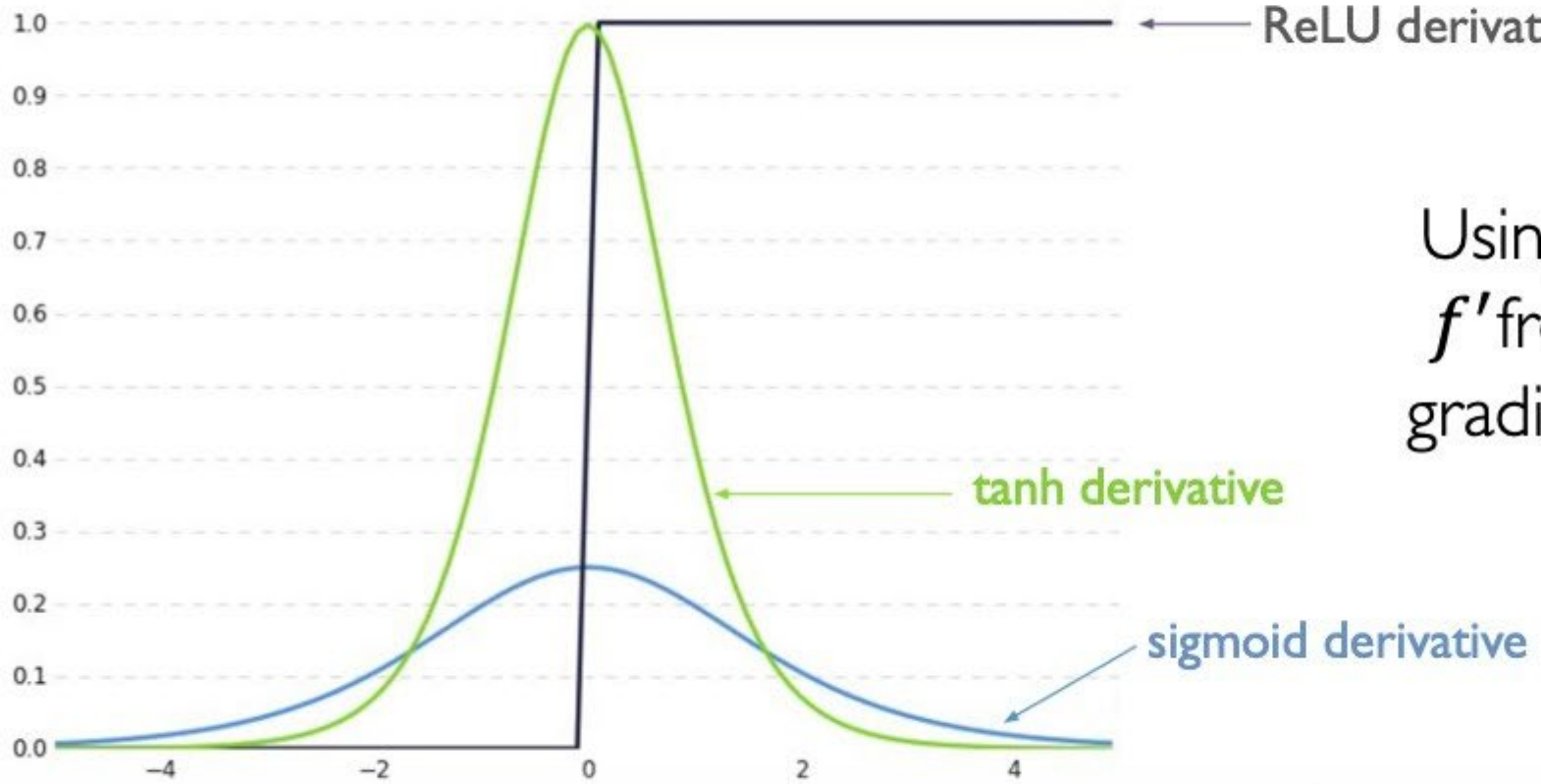
"The clouds are in the ___"



"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

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

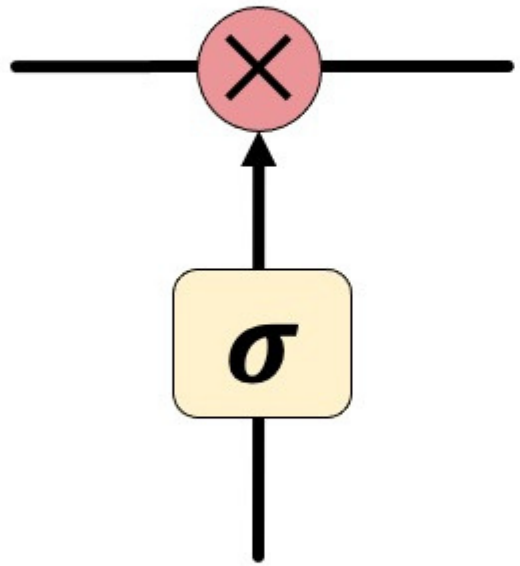
This helps prevent the weights from shrinking to zero.

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.

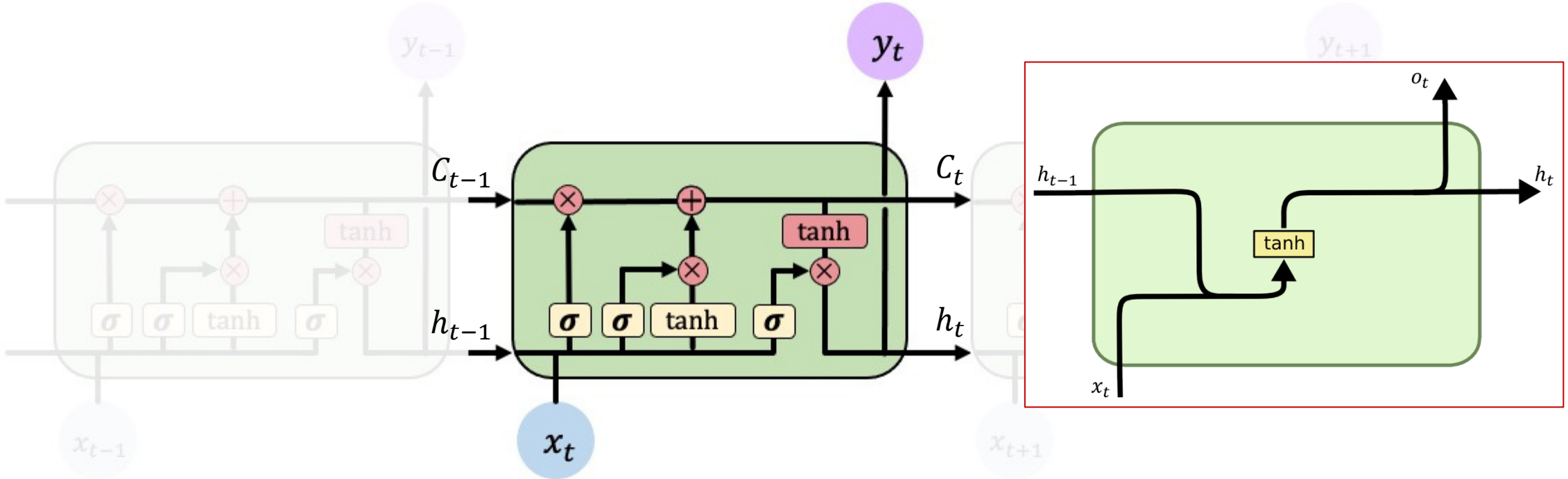
Gates optionally let information through the cell

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

Long Short Term Memory (LSTMs)

Gated LSTM cells **control information flow:**

1) Forget 2) Store 3) Update 4) Output

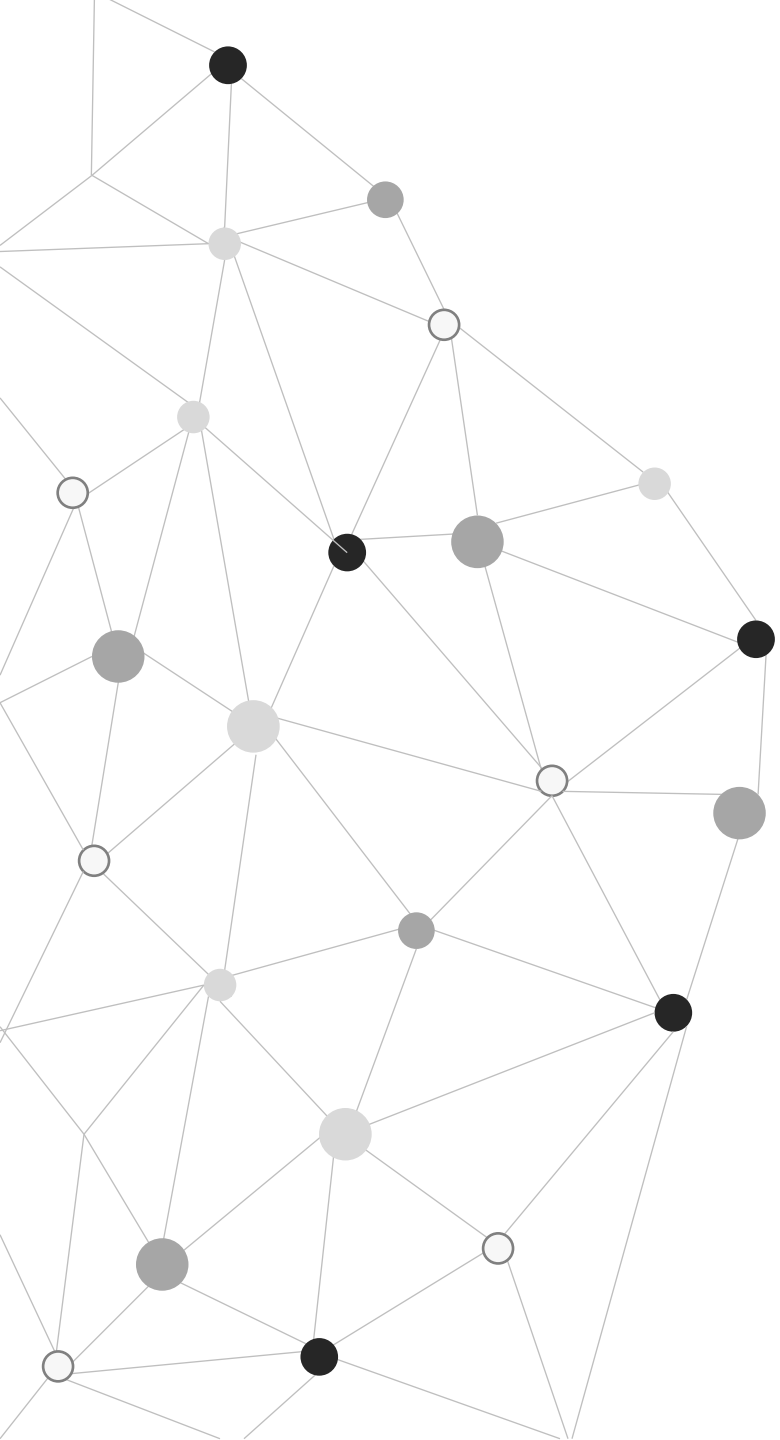


LSTM cells are able to track information throughout many timesteps

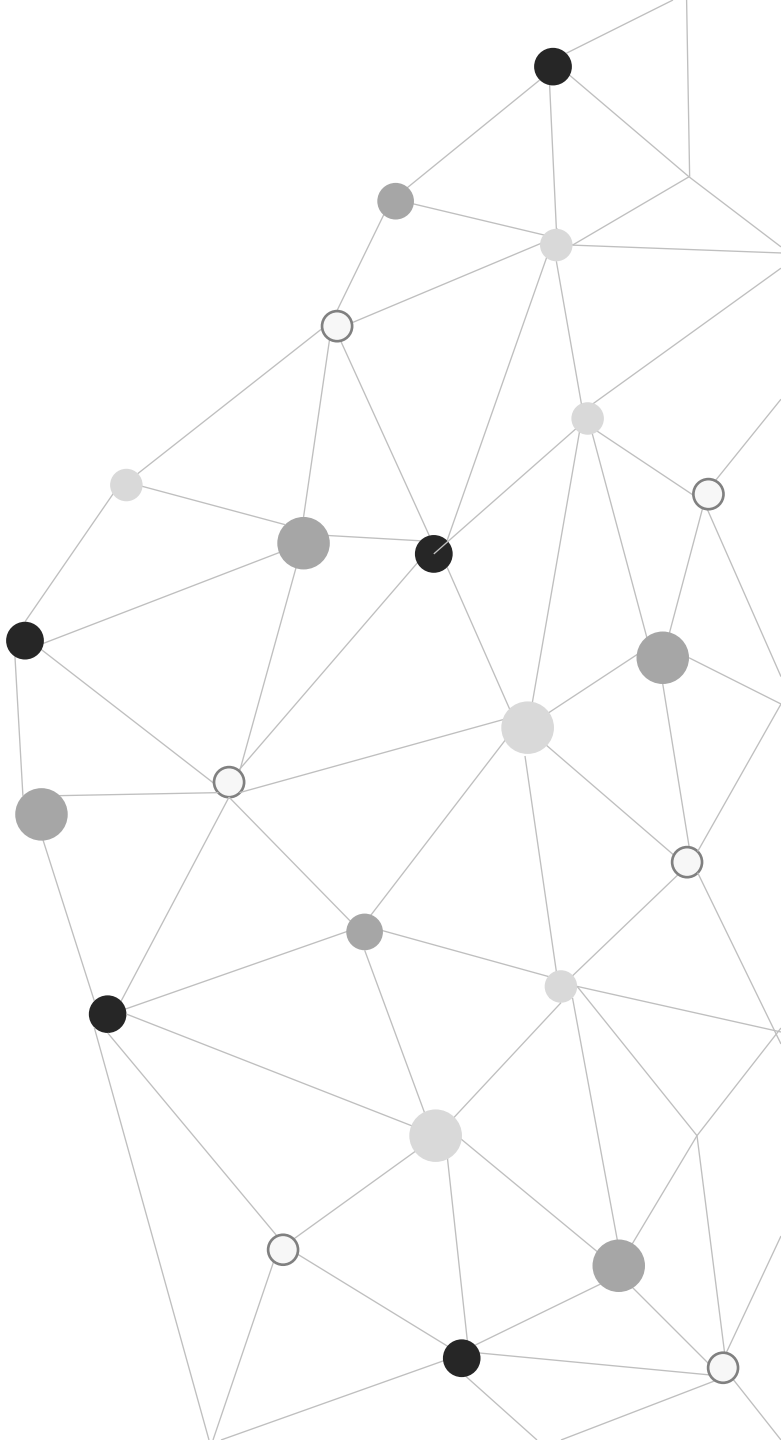
```
tf.keras.layers.LSTM(num_units)
```

LSTMs: Key Concepts

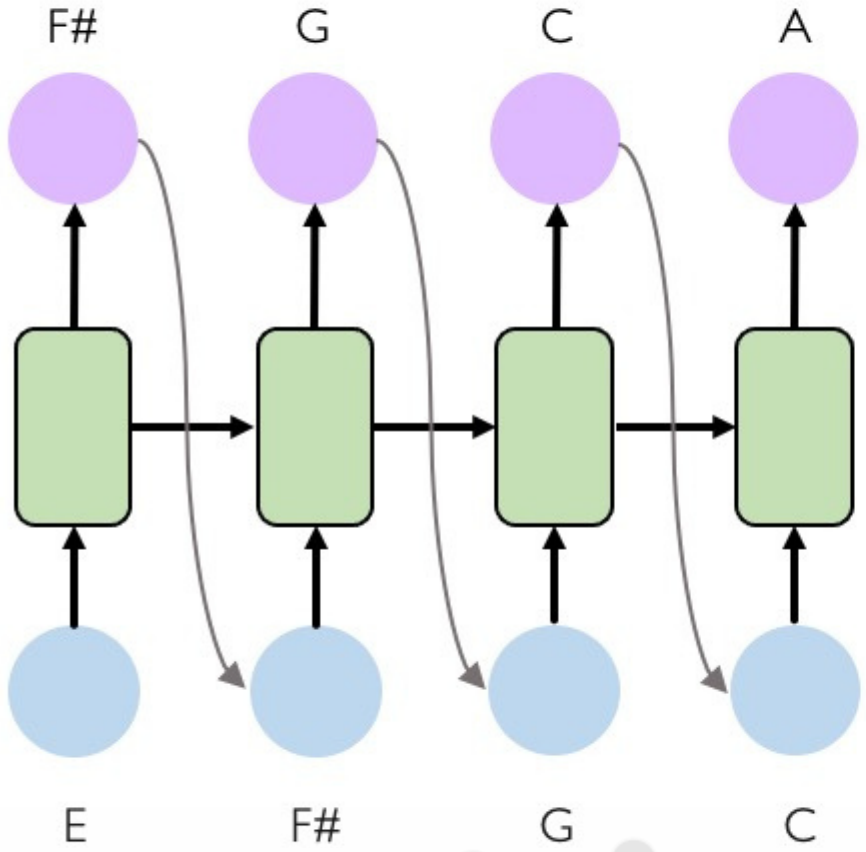
1. Maintain a **cell state**
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state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



RNN Application & Limitations



Example Task: Music Generation



Input: sheet music

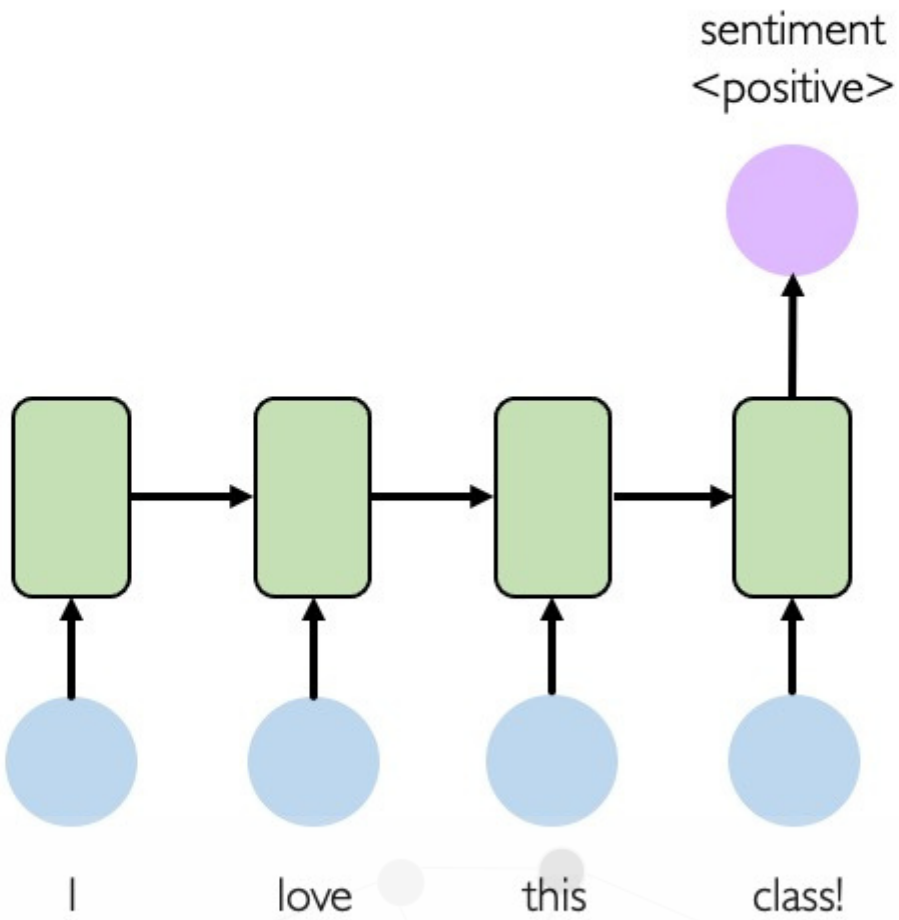
Output: next character in sheet music

The Symphony No. 8 from Schubert ("The Unfinished") actually finished by an A.I.

First movement : 0:06
Second movement : 13:55
Third movement (A.I.) : 24:35
Fourth movement (A.I.) : 35:52

https://youtu.be/RCo8Feho1RI?si=vVGULF9pvTVFoSup&t=2151

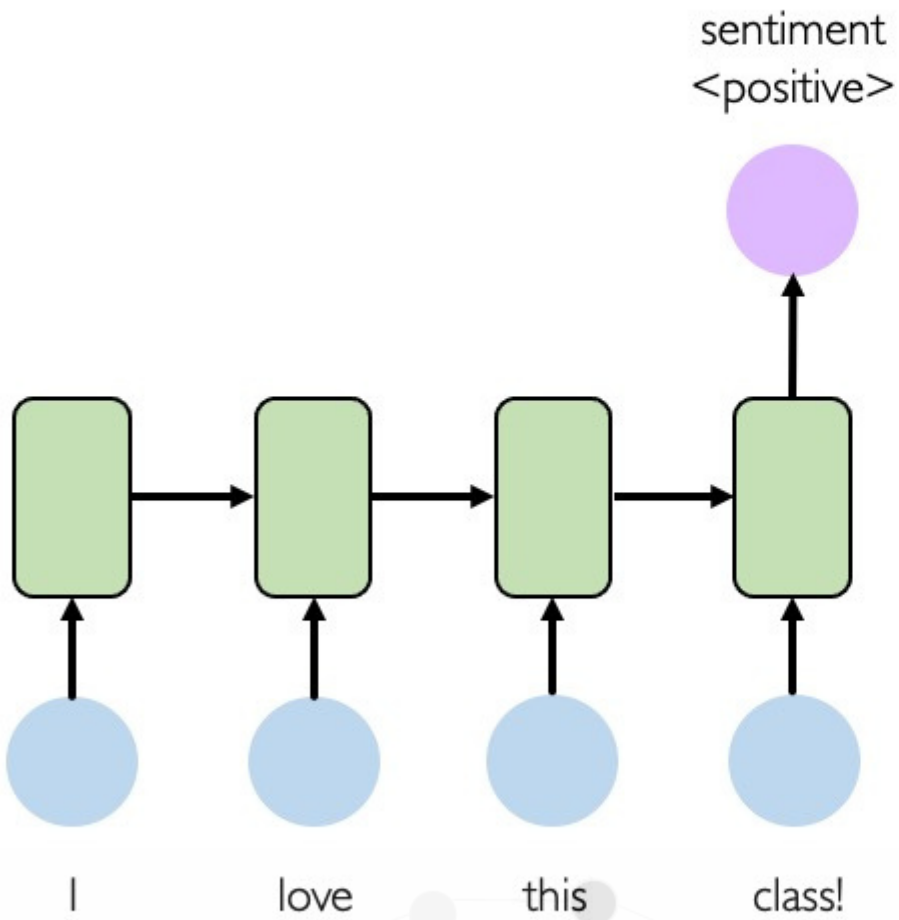
Example Task: Sentiment Classification



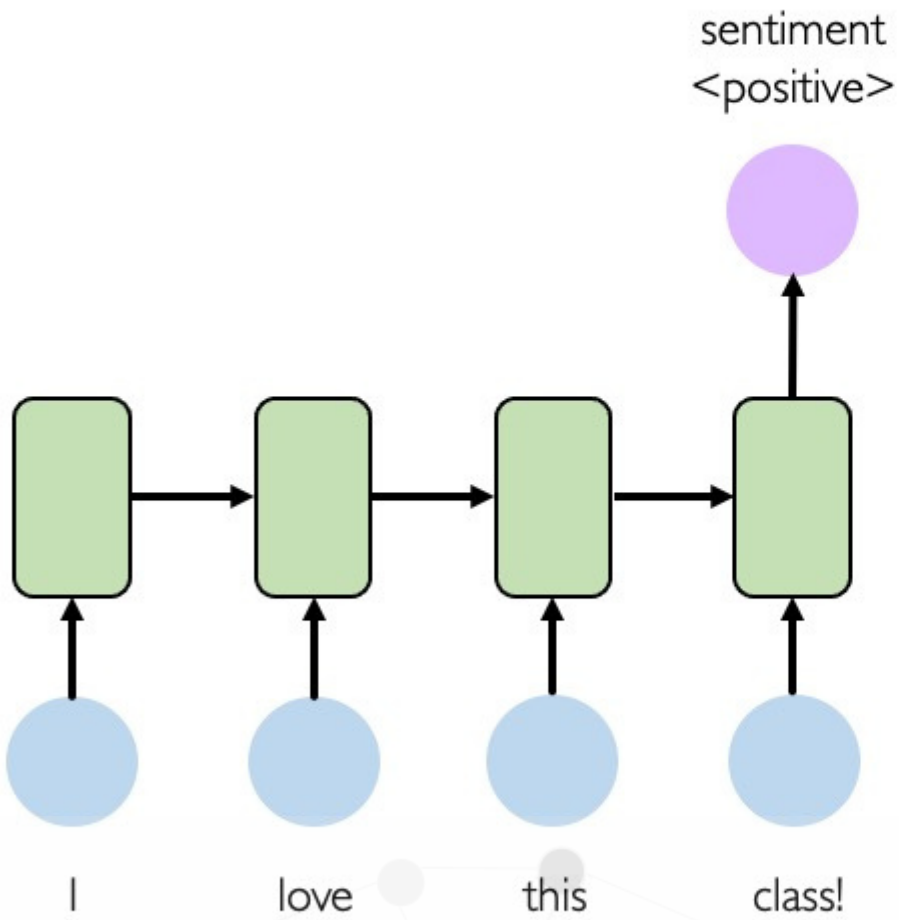
Input: sequence of words

Output: probability of having positive sentiment

Example Task: Sentiment Classification



Limitations of Recurrent Models



Limitations of RNNs

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

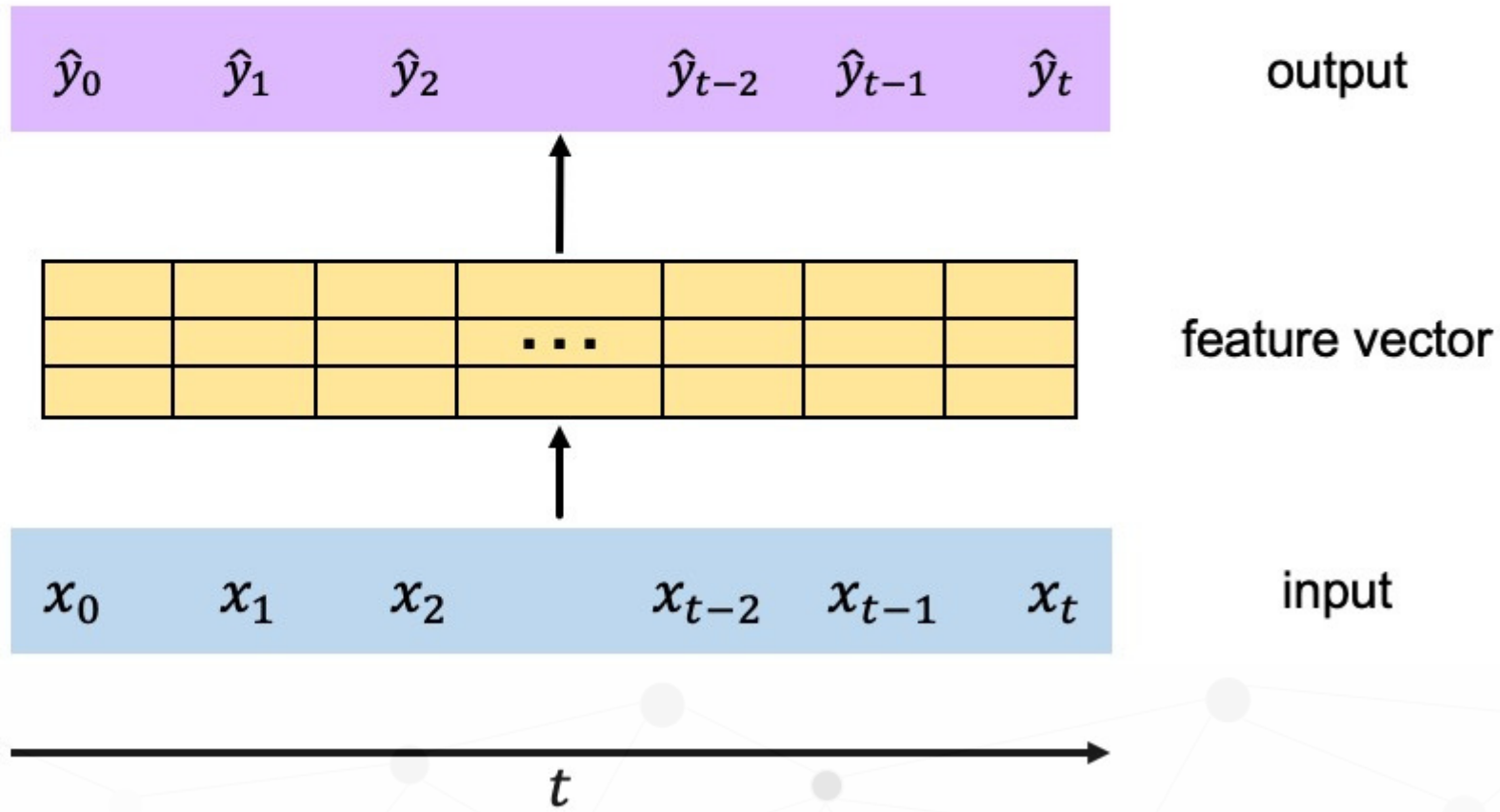
Goal of Sequence Modeling

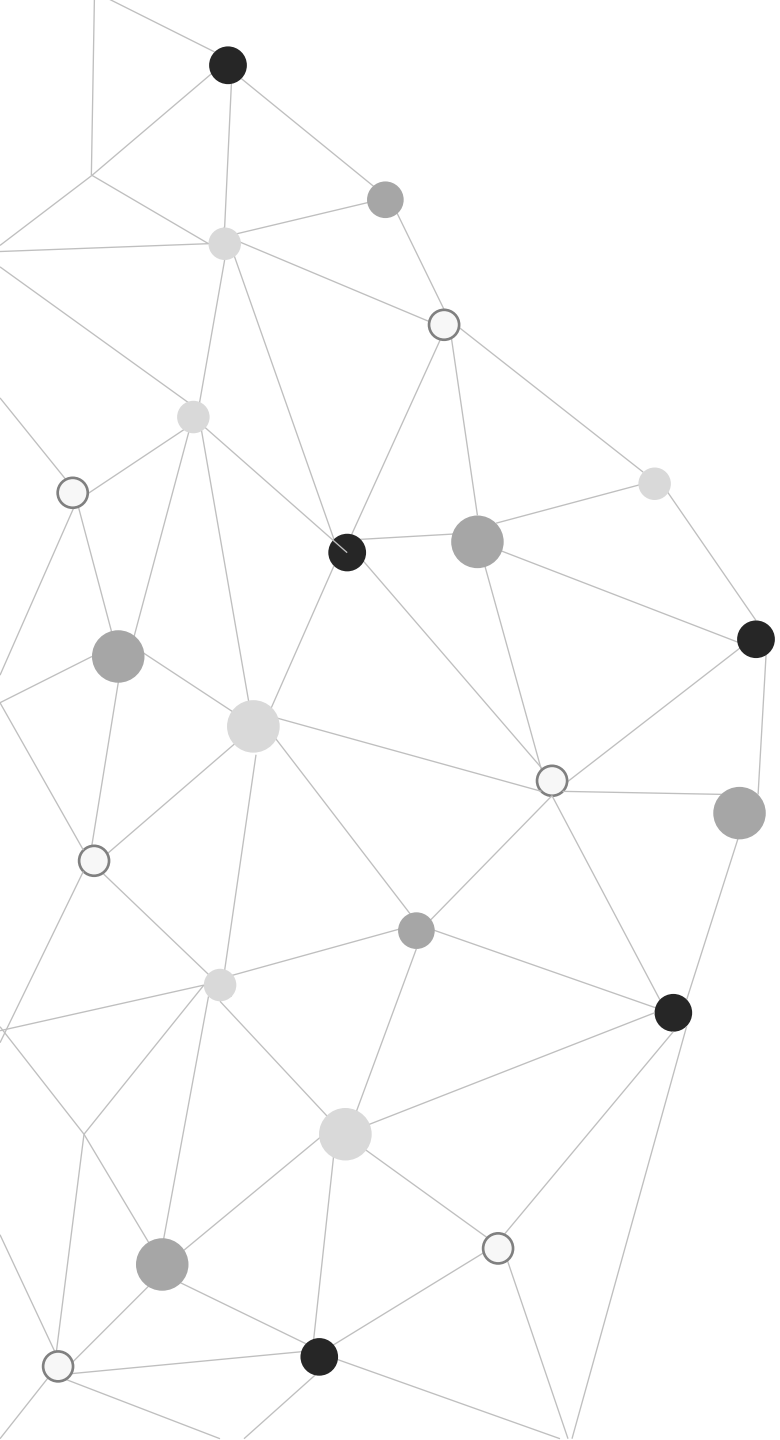
Idea 1: Feed everything into dense network

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

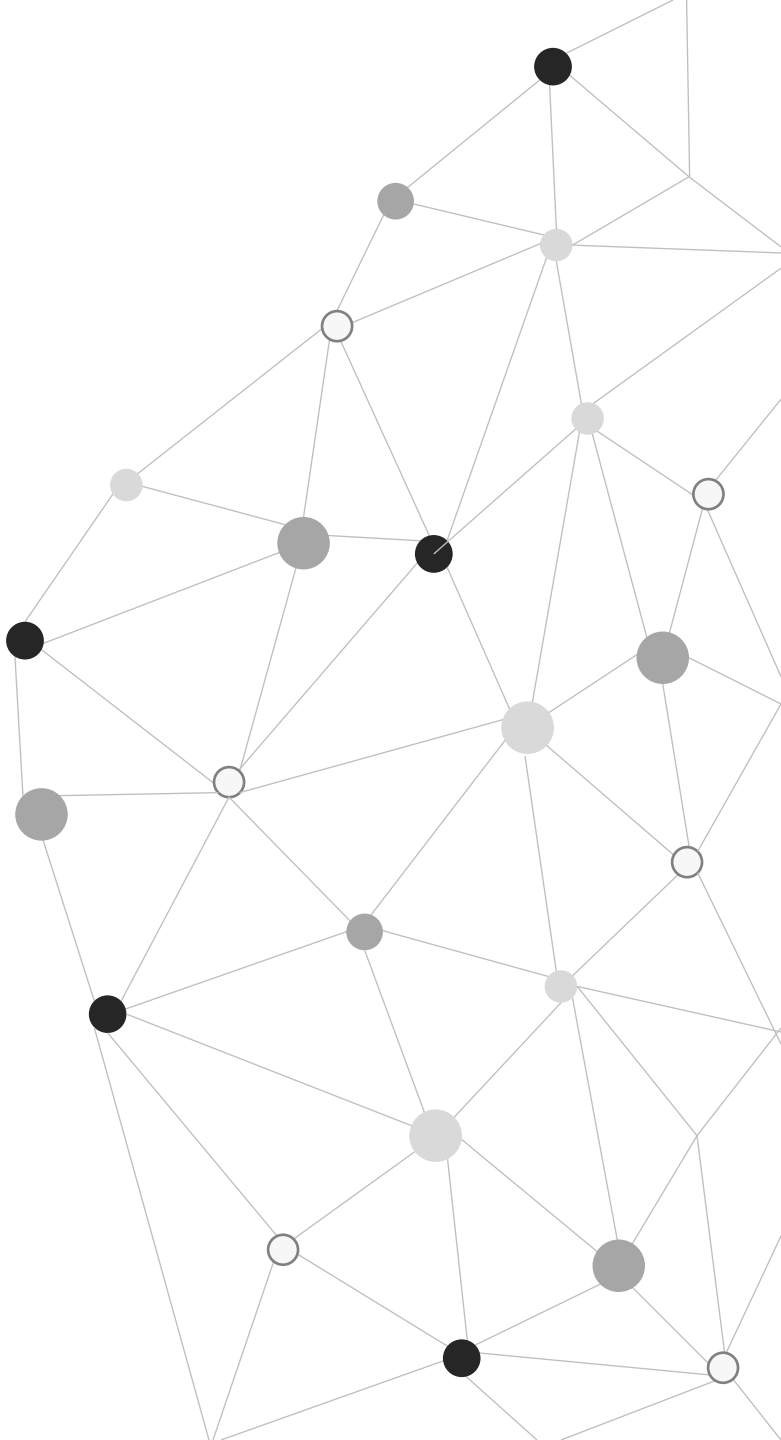
Idea: Identify and attend to what's important

Can we eliminate the need for recurrence entirely?



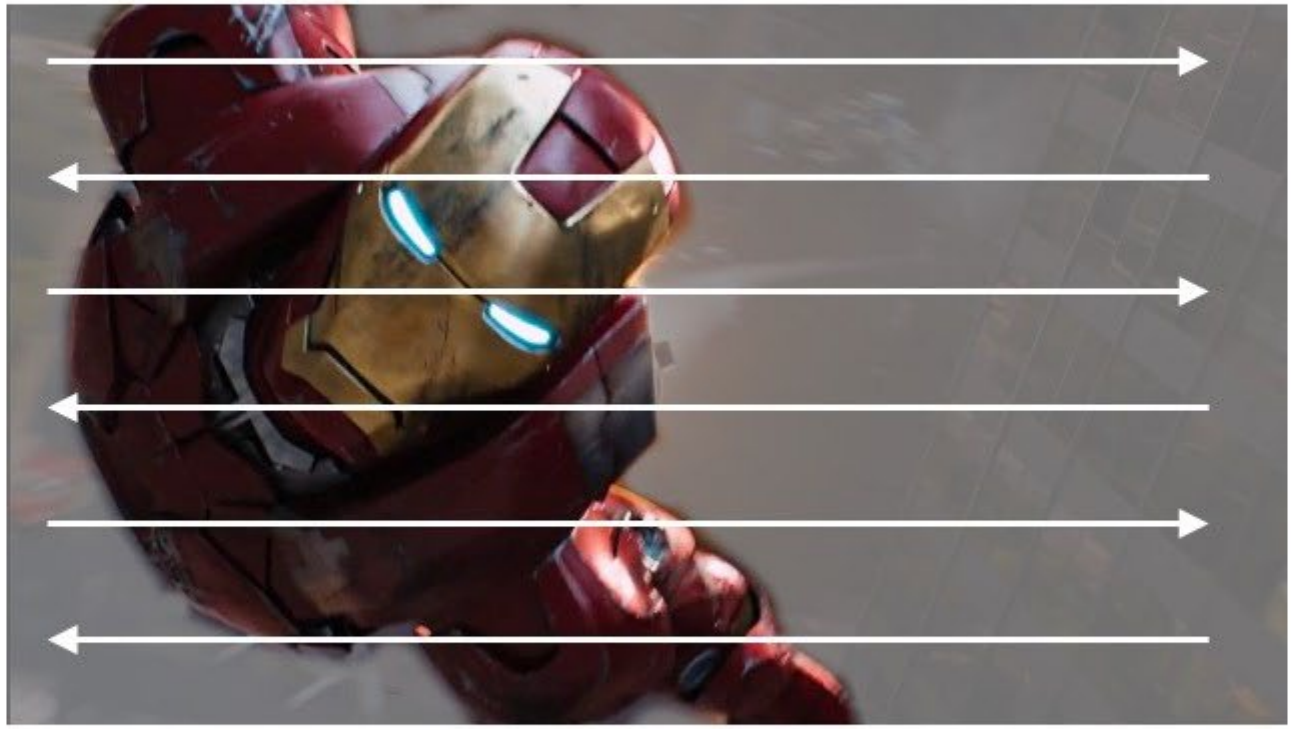


Attention is All You Need



Intuition Behind Self-Attention

Attending to the most important parts of an input.

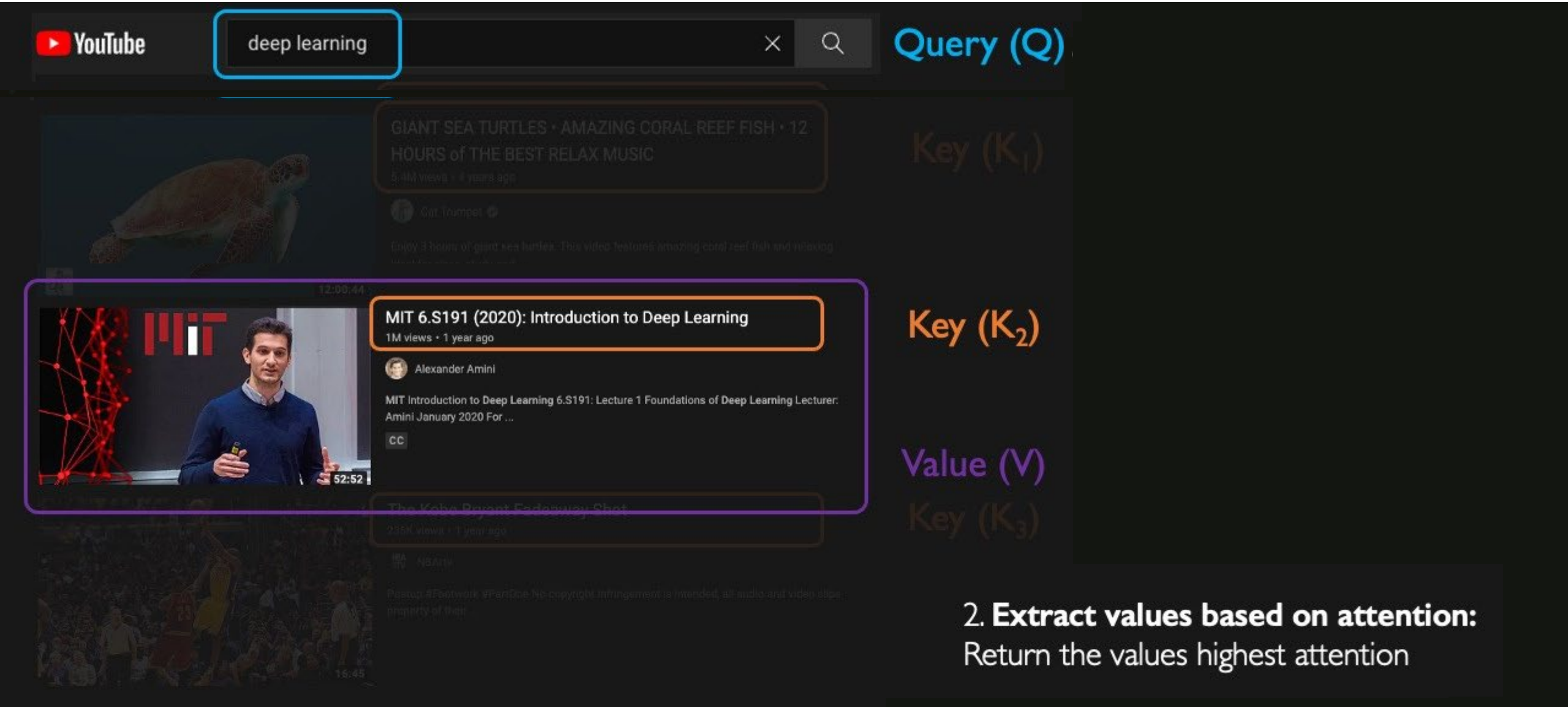


1. Identify which parts to attend to
2. Extract the features with high attention

Similar to a search problem!

A Simple Example: Search

Understanding attention with Search

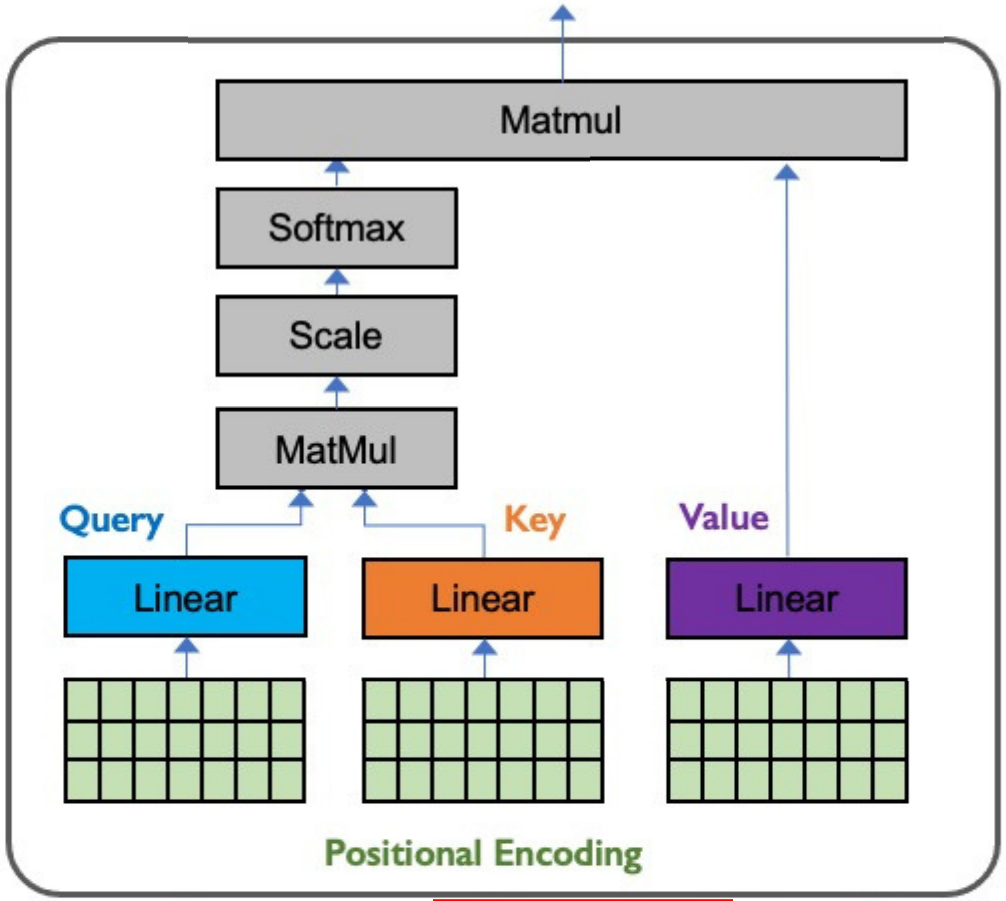


Learning Self-Attention with Neural Networks

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.



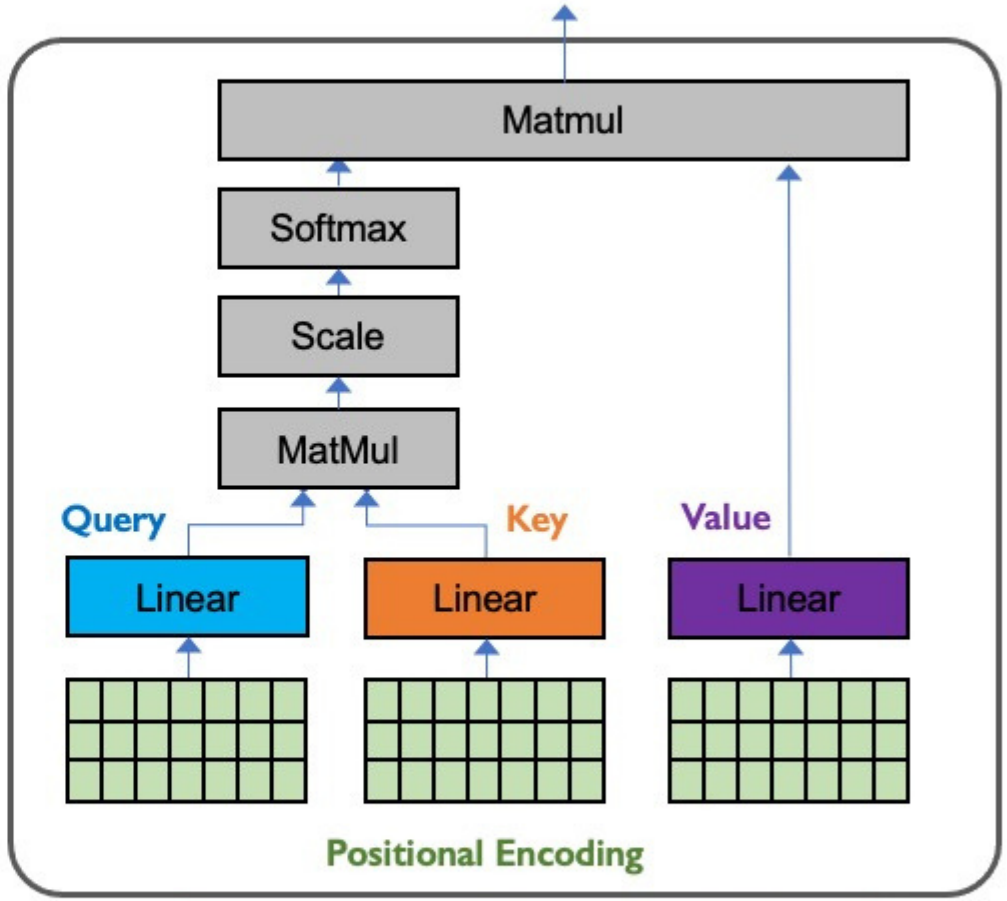
$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V$$

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

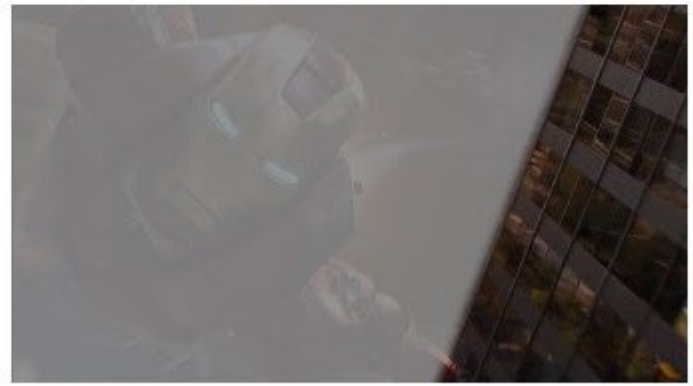
×

Value

=

Output

Output of attention head 1



Output of attention head 2

Output of attention head 3

Self-Attention Applied

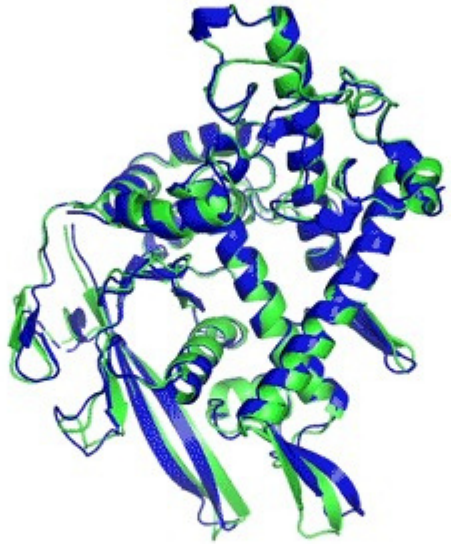
Language Processing

An armchair in the shape of an avocado

BERT, GPT-3

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

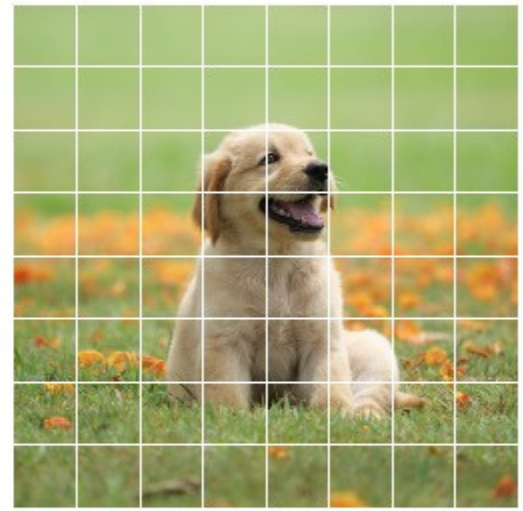
Biological Sequences



AlphaFold2

Jumper et al., *Nature* 2021

Computer Vision



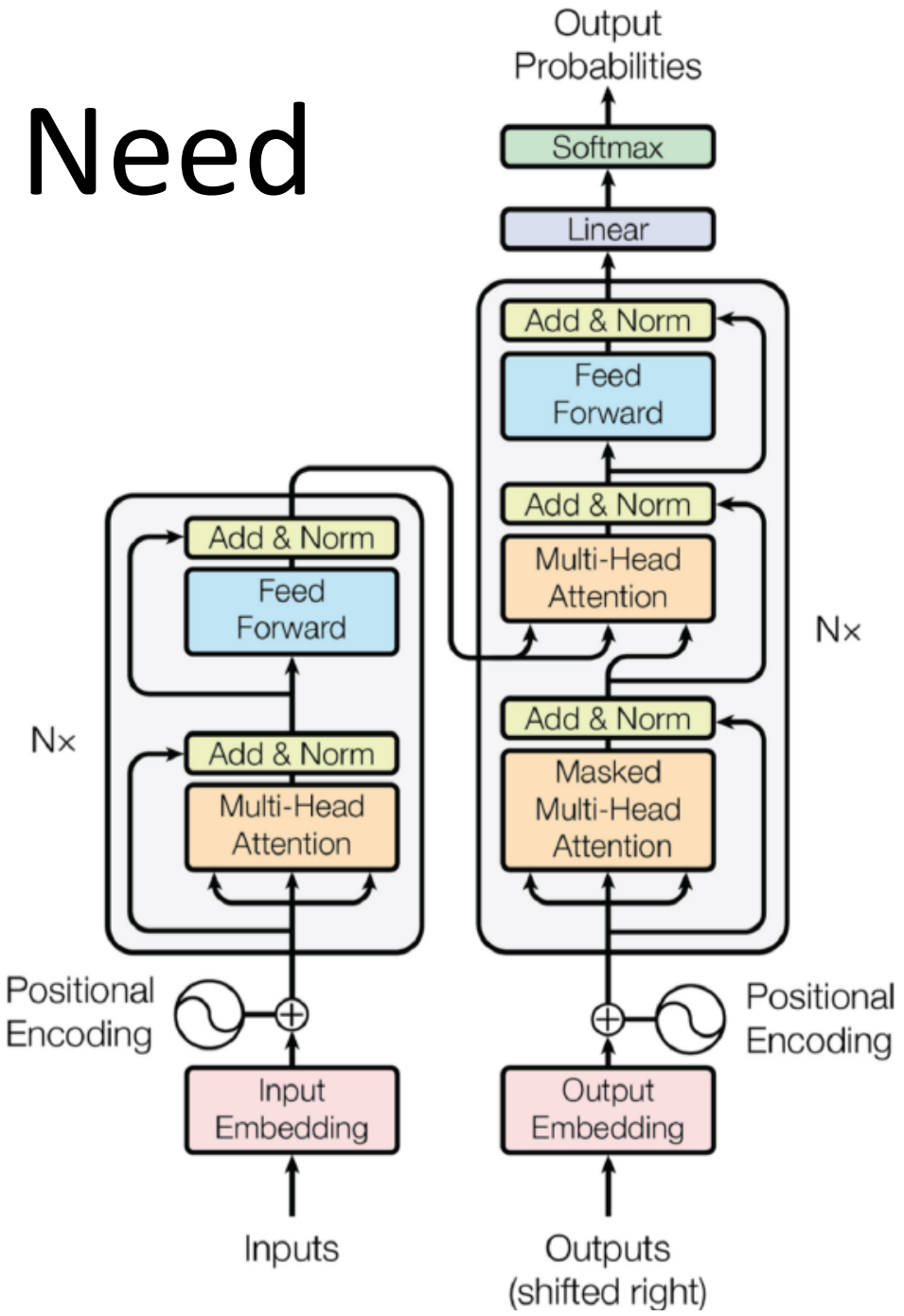
Vision Transformers

Dosovitskiy et al., *ICLR* 2020

and there’s lots more to do!

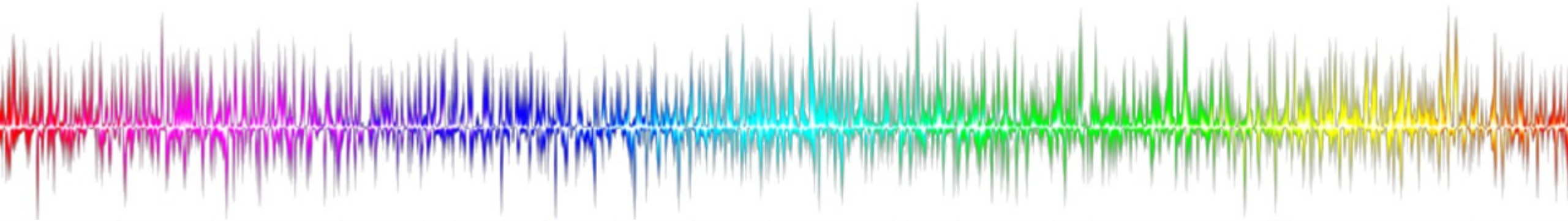
- extending our models to timeseries + waveforms
- complex language models to generate long text or books
- language models to generate code
- controlling cars + robots
- predicting stock market trends
- summarizing books + articles
- handwriting generation
- multilingual translation models
- … many more!

Attention is All You Need



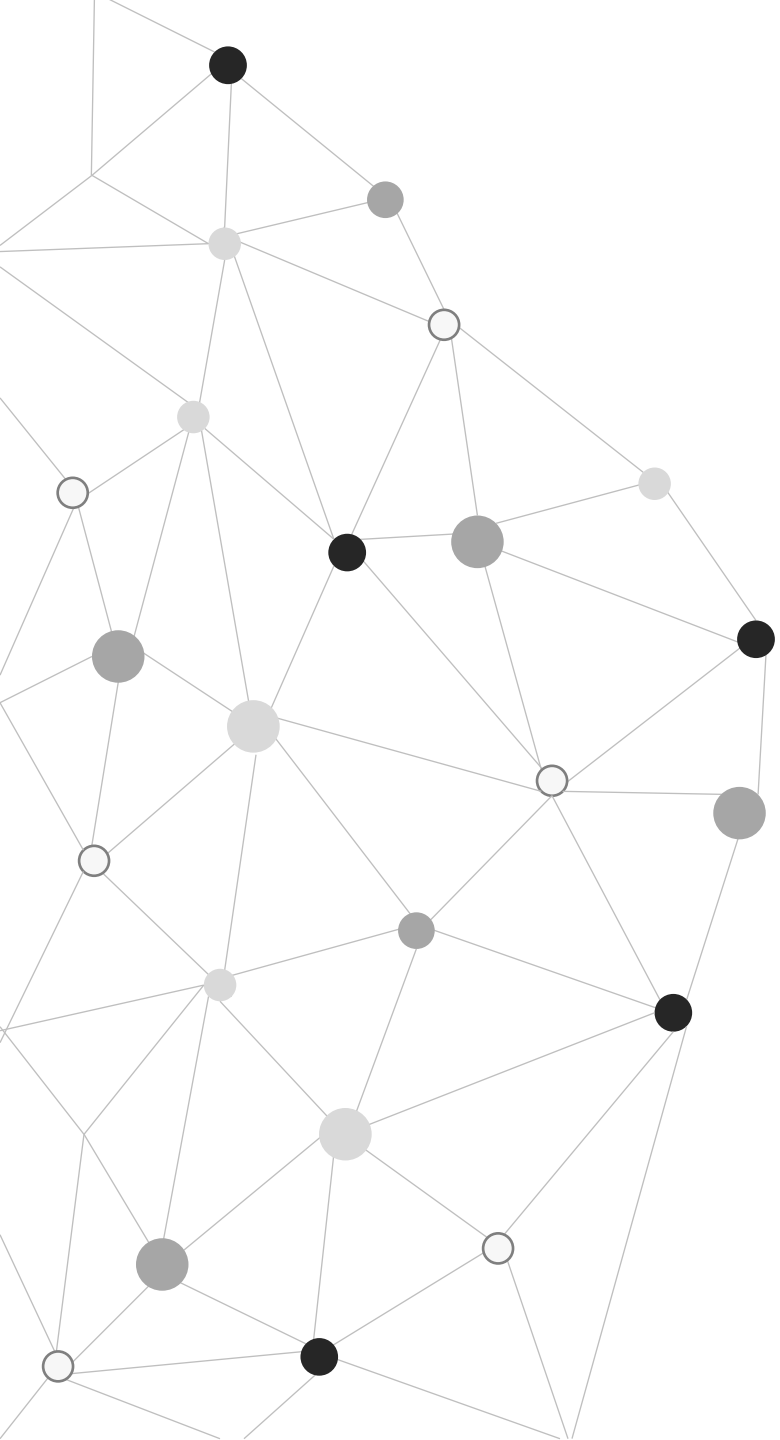
Deep Learning for Sequence Modeling: Summary

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**



More Materials

- Prof. Hung-Yi Lee
  - 【機器學習2021】自注意力機制 (Self-attention) (上) (下)
  - 【機器學習2021】Transformer (上) (下)



Next:

Self-Attention