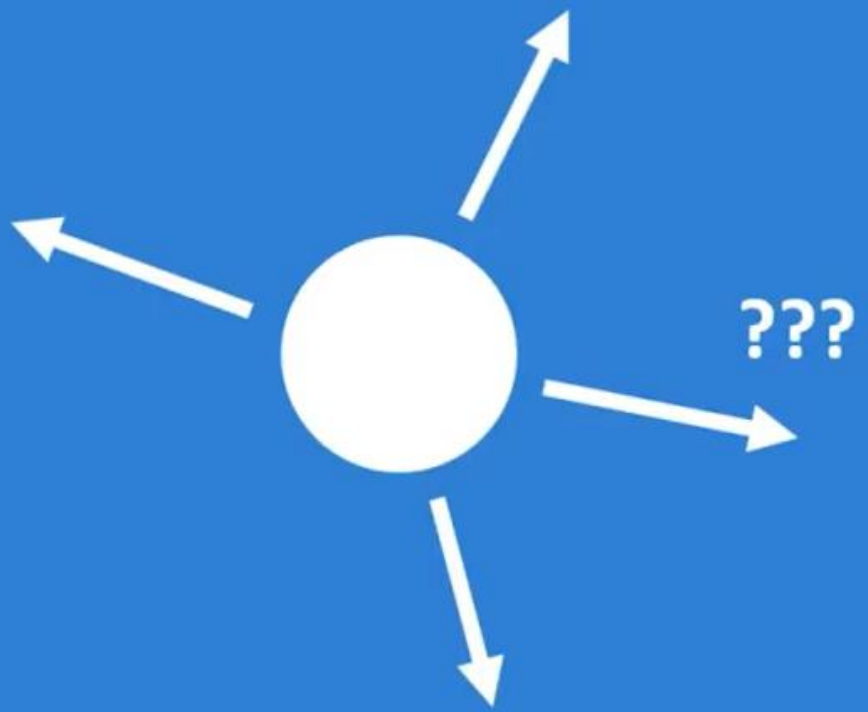
Modeling Sequence Data

1. Neurons with Recurrence (intuitions of RNNs)
2. Vanilla Recurrent Neural Networks (RNNs)
3. Sequence Modeling Problem: Predict the Next Word
4. Backpropagation Through Time (BPTT)
5. Long Short Term Memory (LSTM) Networks
6. RNN Applications

Given an image of a ball,
can you predict where it will go next?

Given an image of a ball,
can you predict where it will go next?



Given an image of a ball,
can you predict where it will go next?

Sequences in the Wild

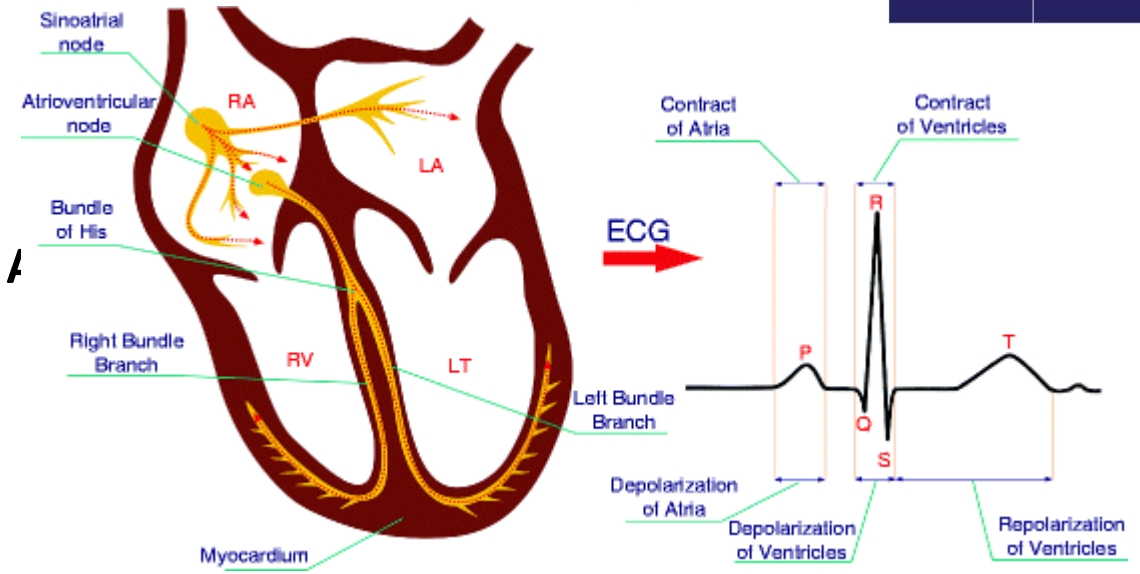
character: 6 . S I 9 I

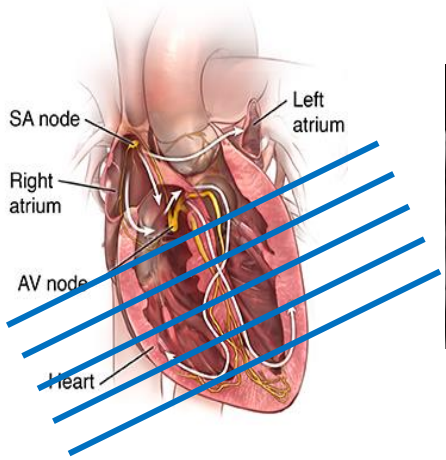
word: Introduction to Deep Learning

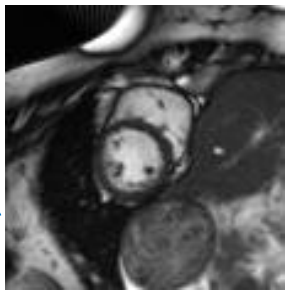
Audio:

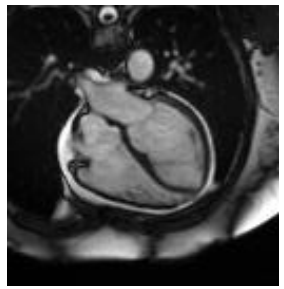
Sequences in the Wild

90% accurate					80% accurate		50% accurate		
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed
76°	74°	70°	70°	71°	76°	75°	?	?	?









Sequence Modeling Applications

One to One
Binary Classification

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

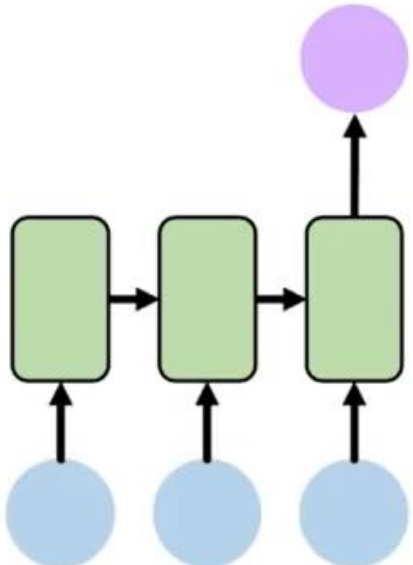
Sequence Modeling Applications

One to One

Binary Classification

"Will I pass this class?"

Student → Pass?



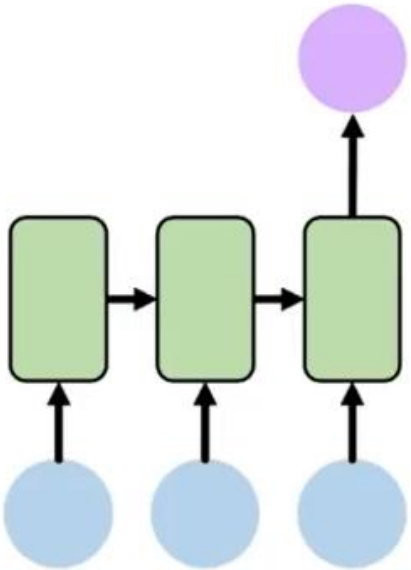
Many to One

Sentiment Classification

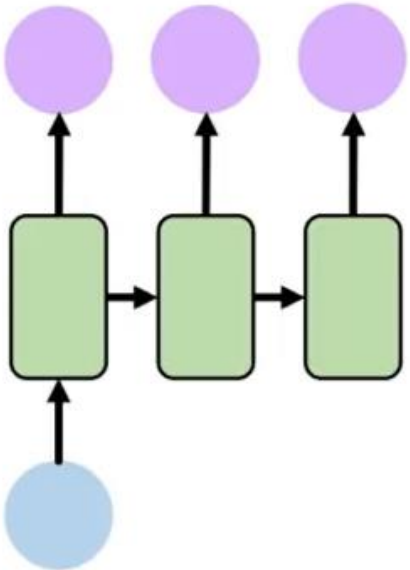
Sequence Modeling Applications

One to One
Binary Classification

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



Many to One
Sentiment Classification



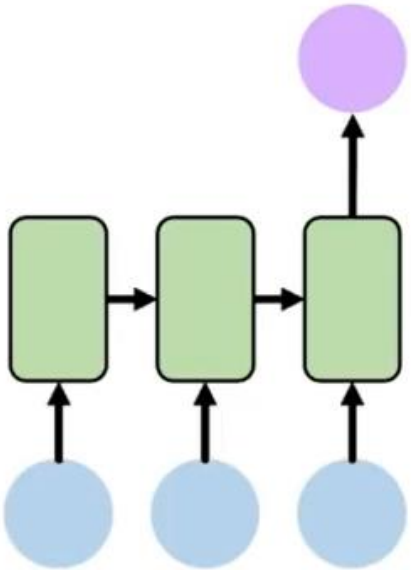
One to Many
Image Captioning

"A baseball player throws a ball."

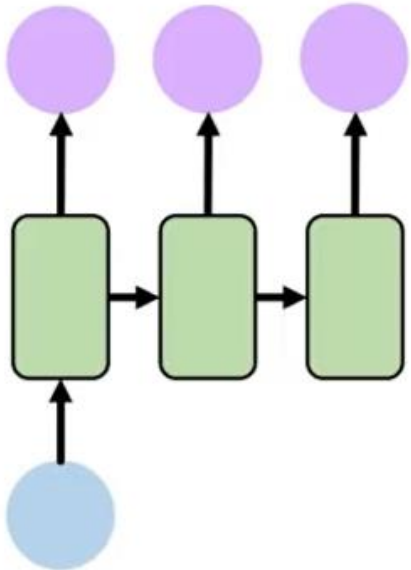
Sequence Modeling Applications

One to One
Binary Classification

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

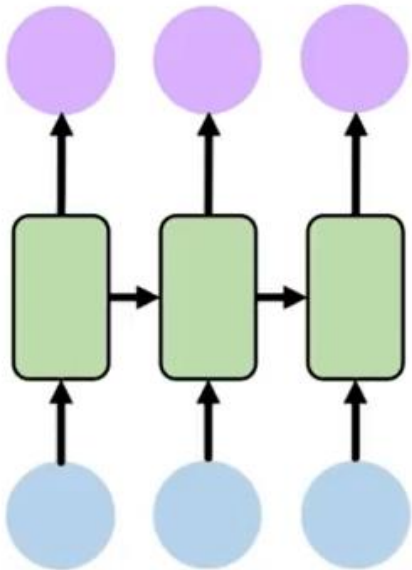


Many to One
Sentiment Classification



One to Many
Image Captioning

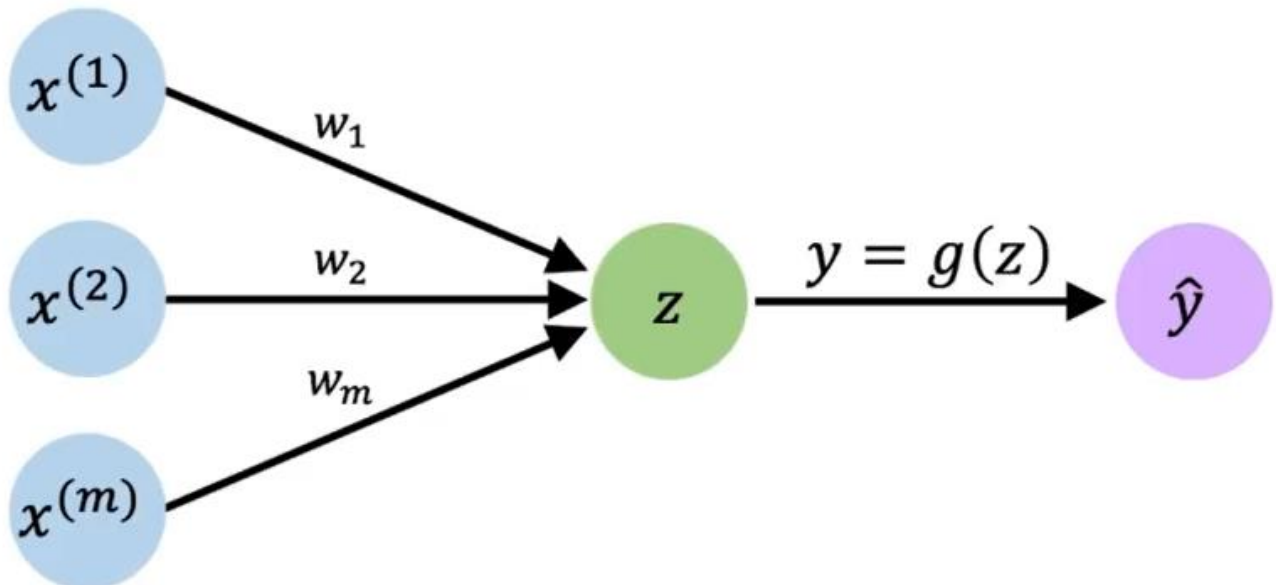
"A baseball player throws a ball."



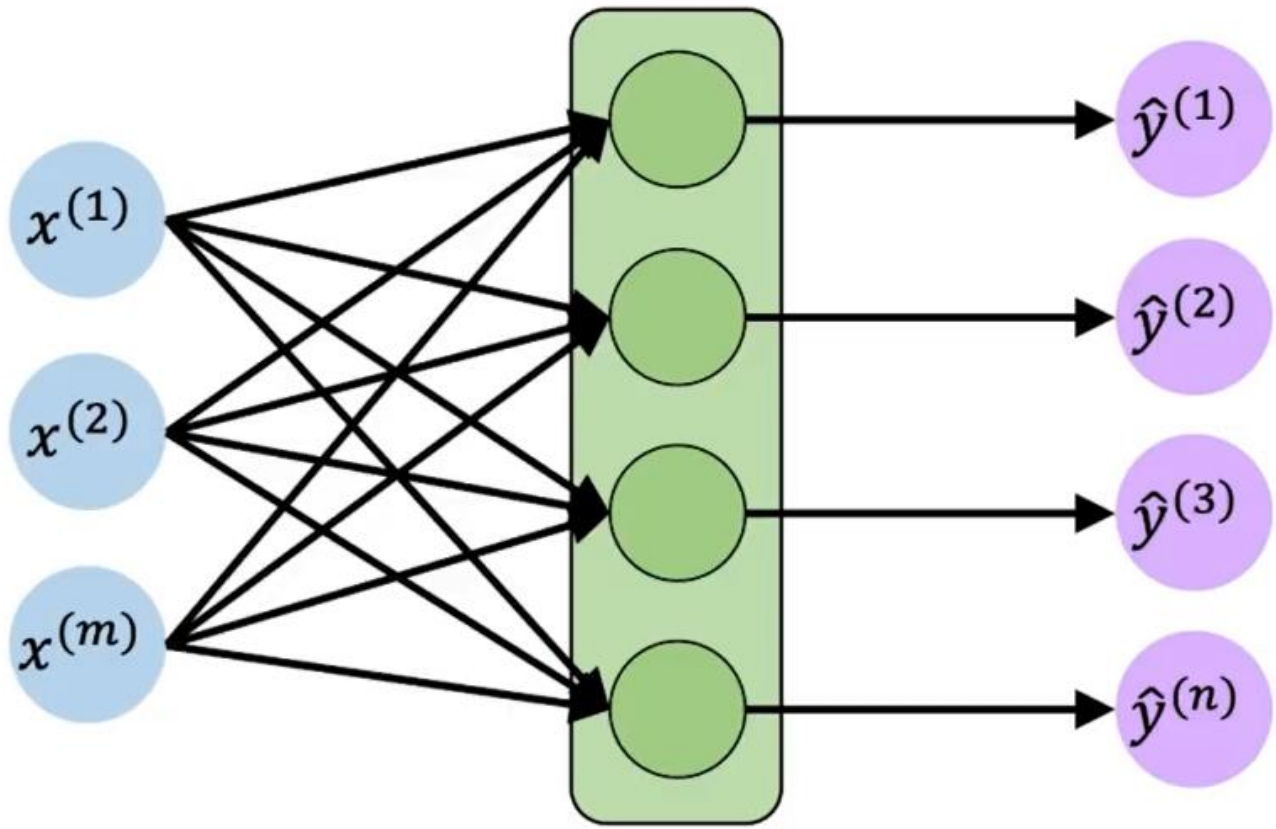
Many to Many
Machine Translation

Neurons with Recurrence

The Perceptron Revisited



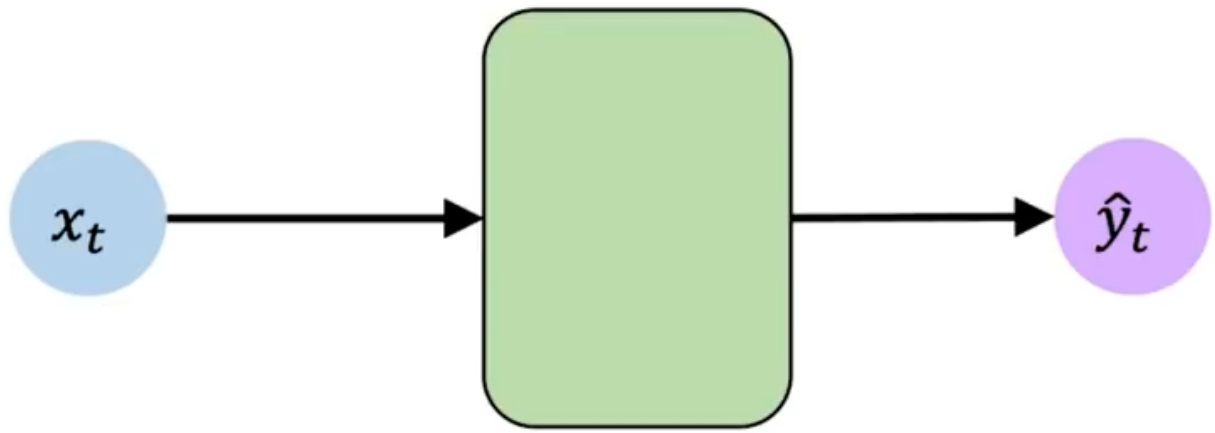
Feed-Forward Networks Revisited



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

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

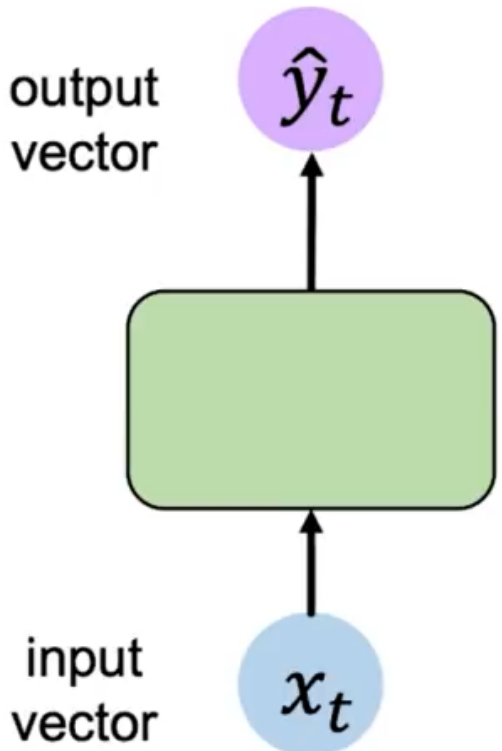
Feed-Forward Networks Revisited



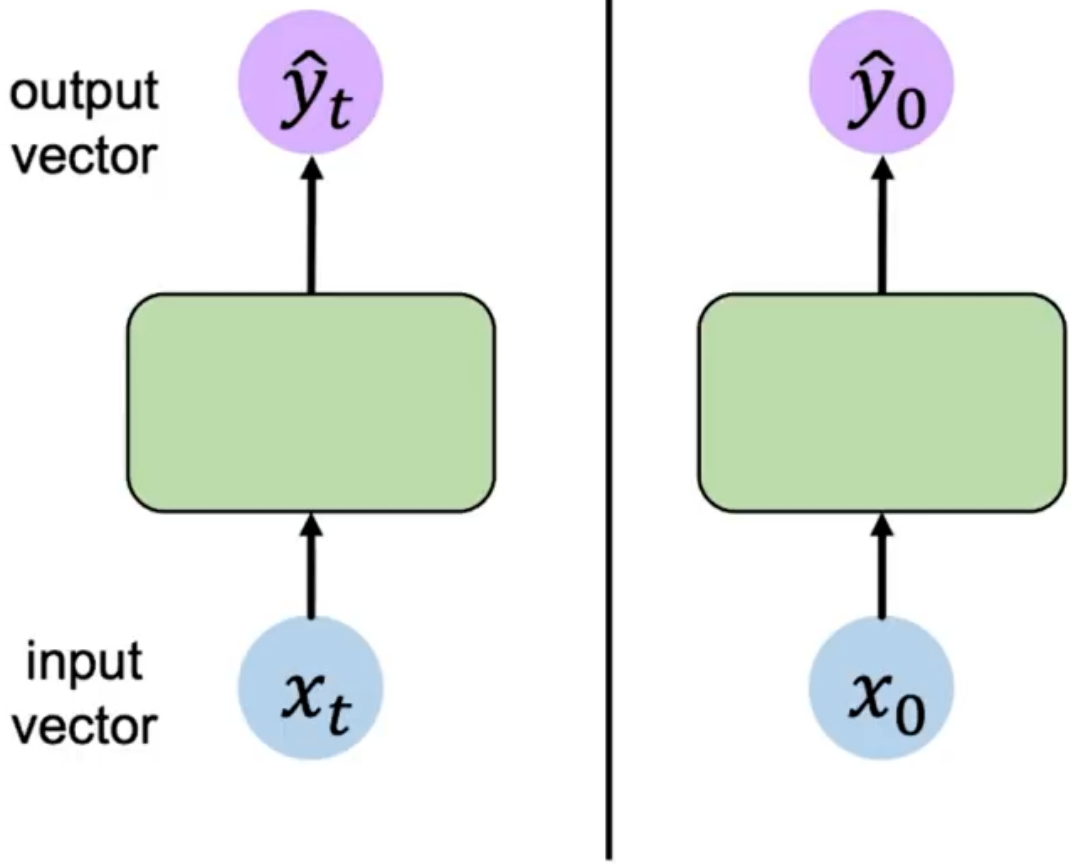
$\boldsymbol{x_t} \in \mathbb{R}^m$

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

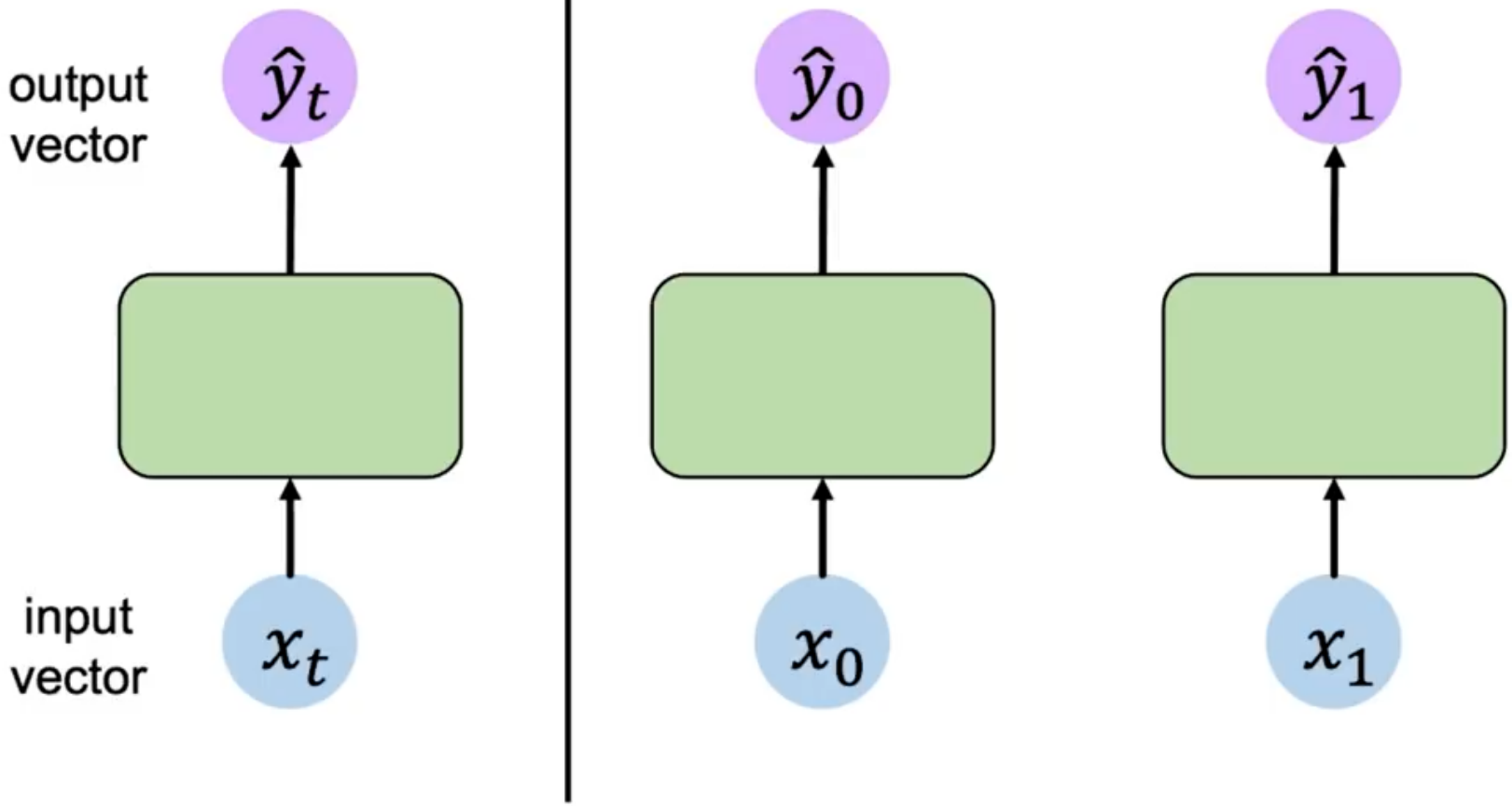
Handling Individual Time Steps



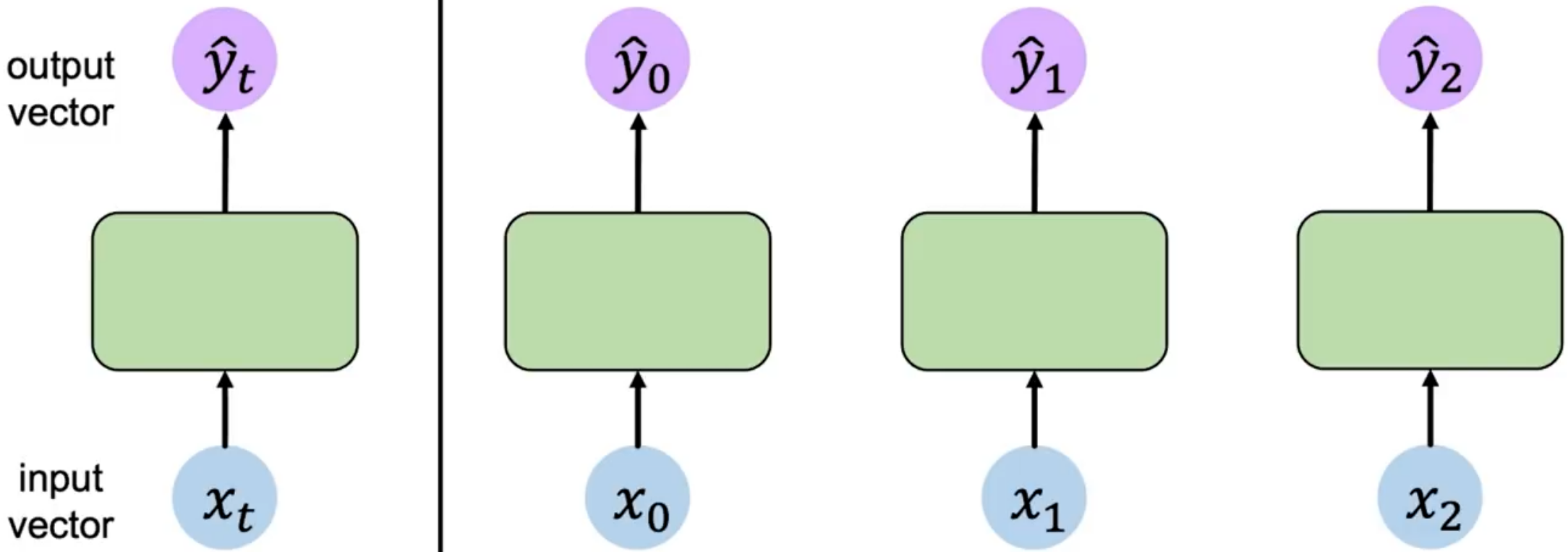
Handling Individual Time Steps



Handling Individual Time Steps

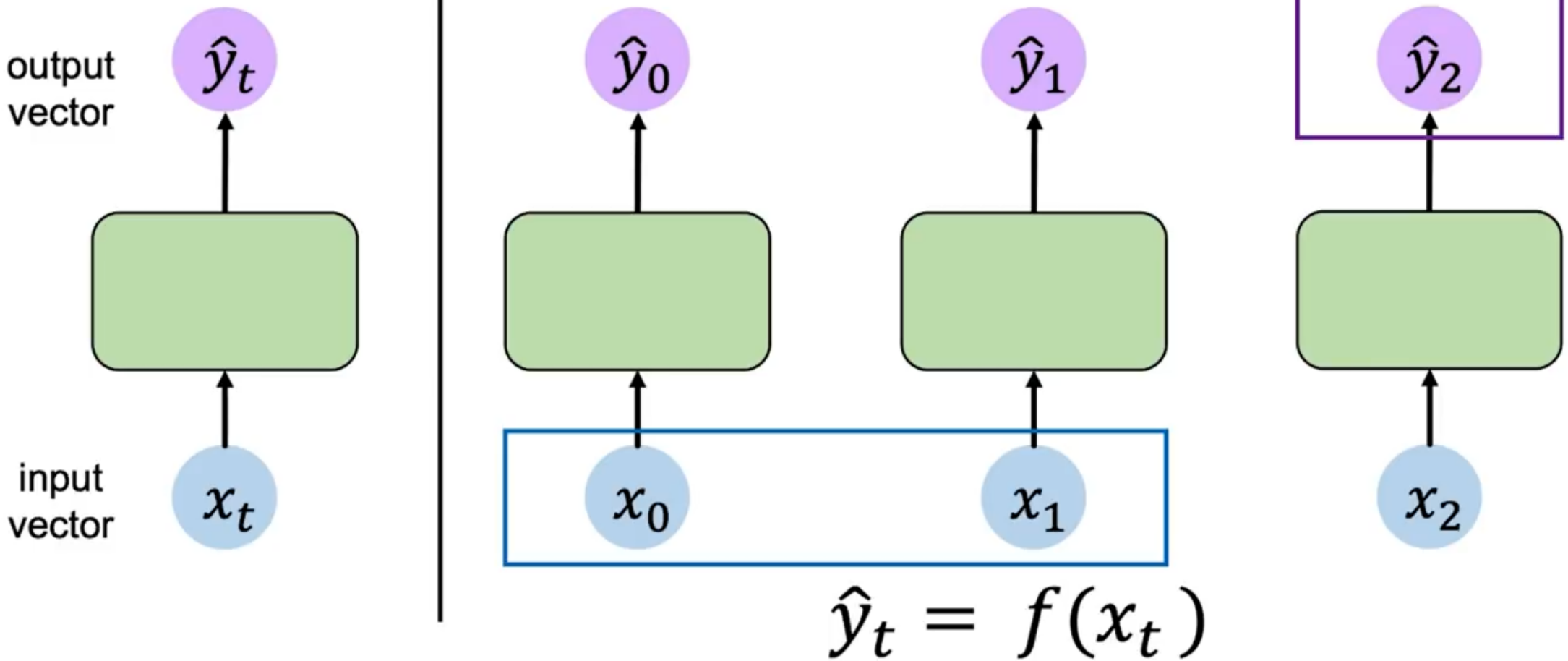


Handling Individual Time Steps



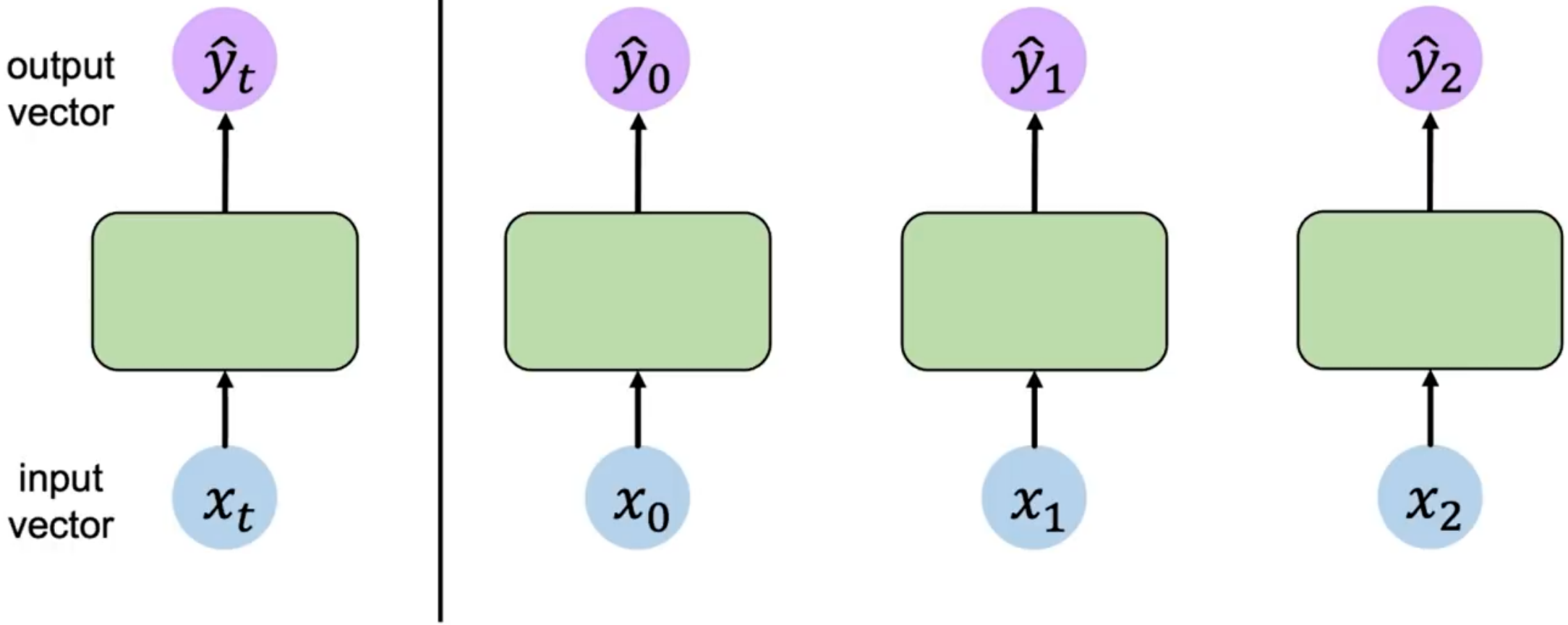
$$\hat{y}_t = f(x_t)$$

Handling Individual Time Steps

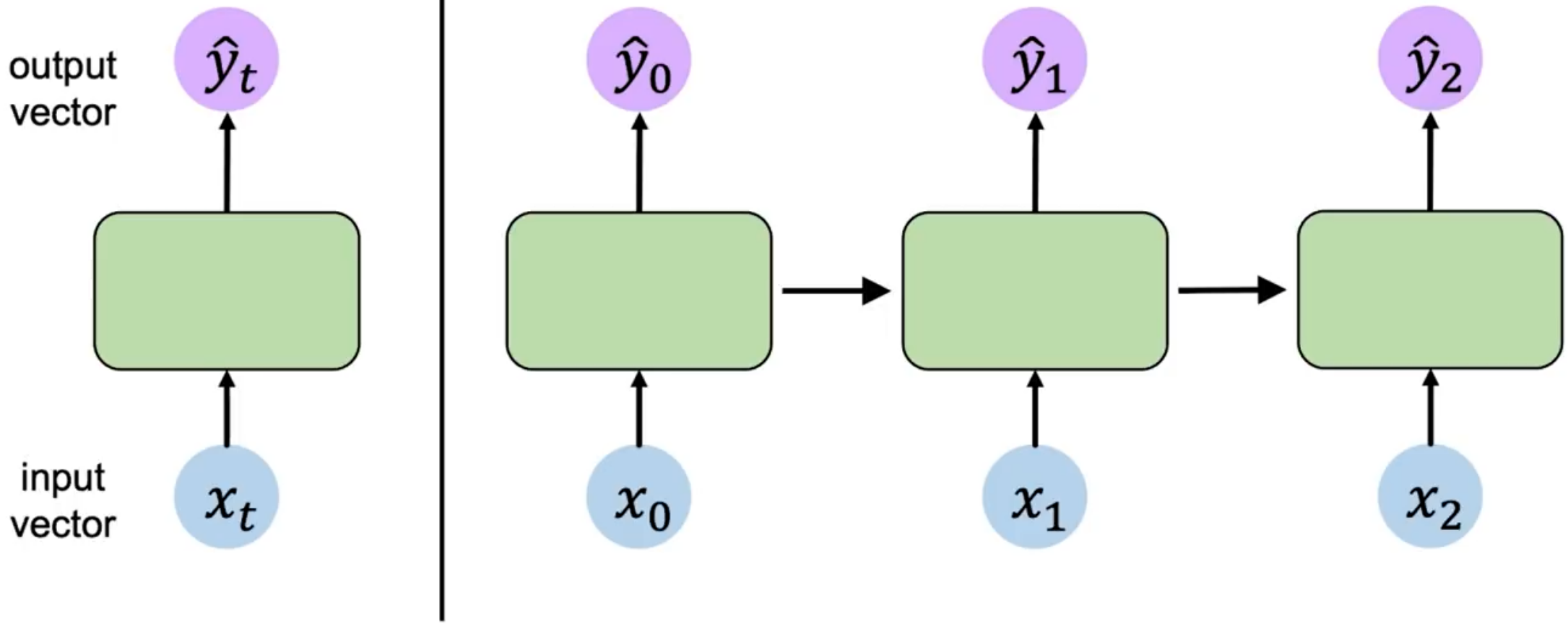


$$\hat{y}_t = f(x_t)$$

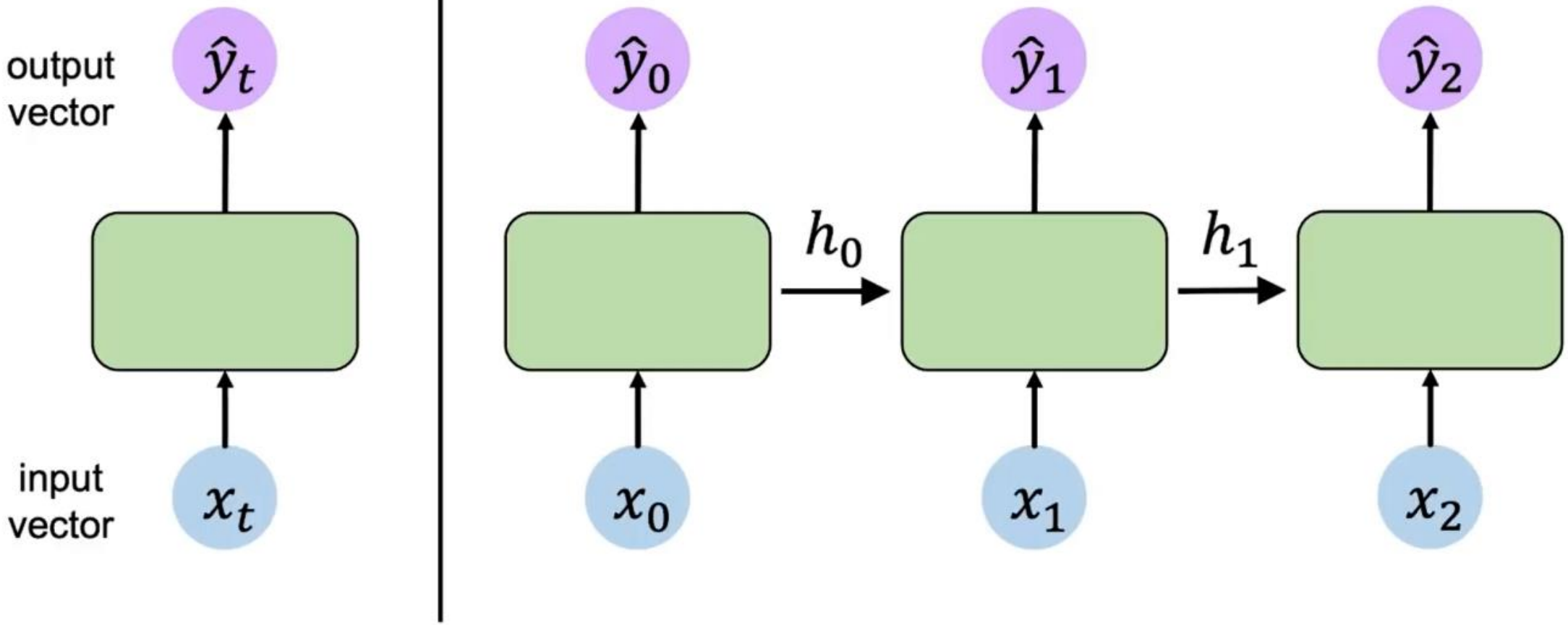
Neurons with Recurrence



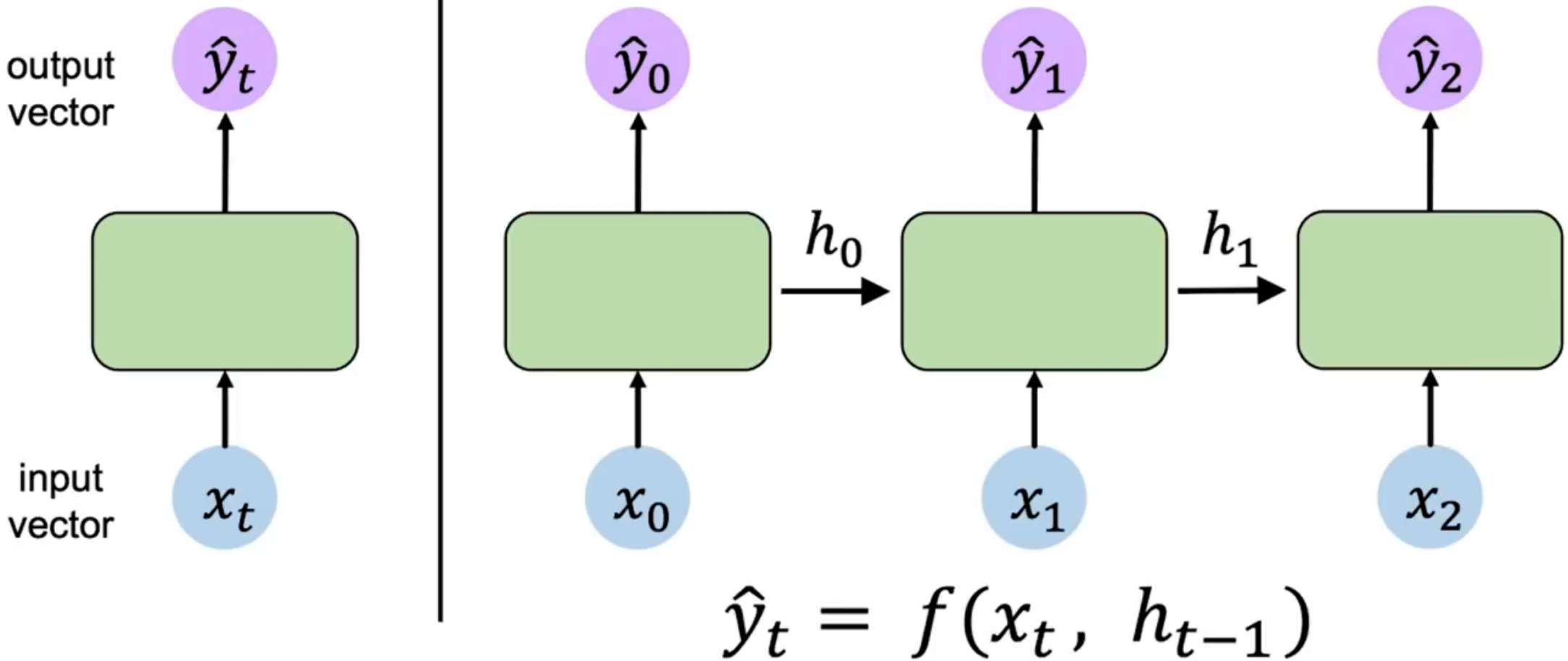
Neurons with Recurrence



Neurons with Recurrence

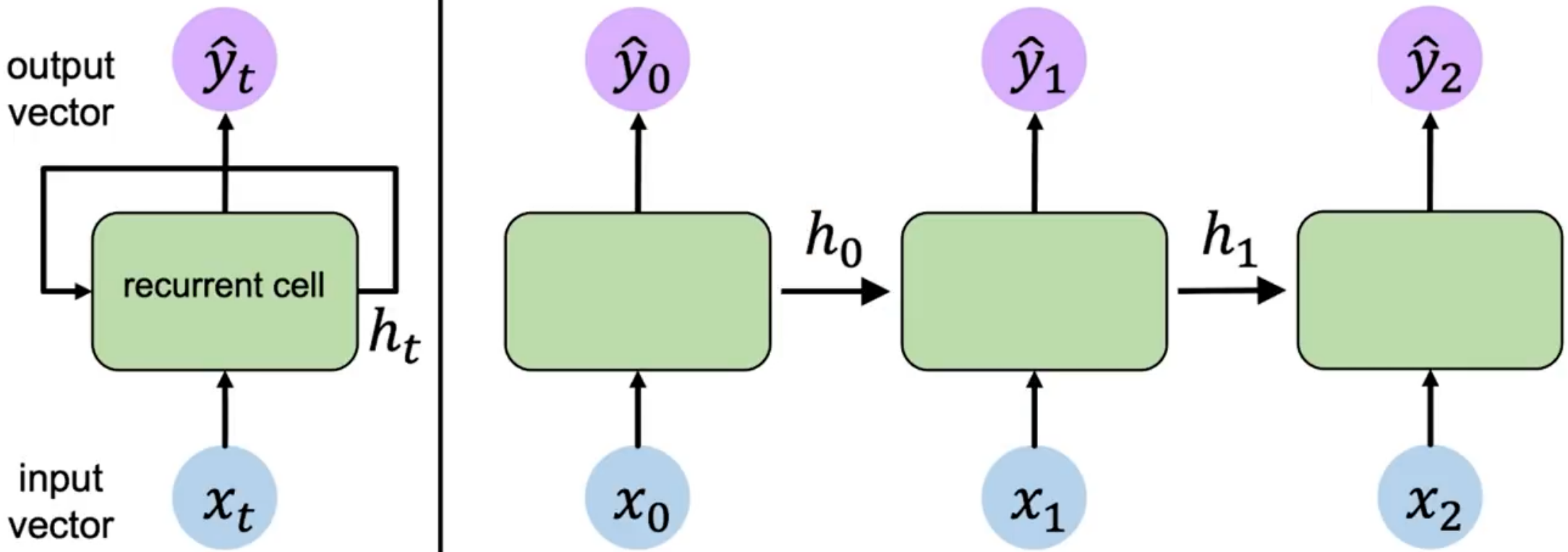


Neurons with Recurrence



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

Neurons with Recurrence

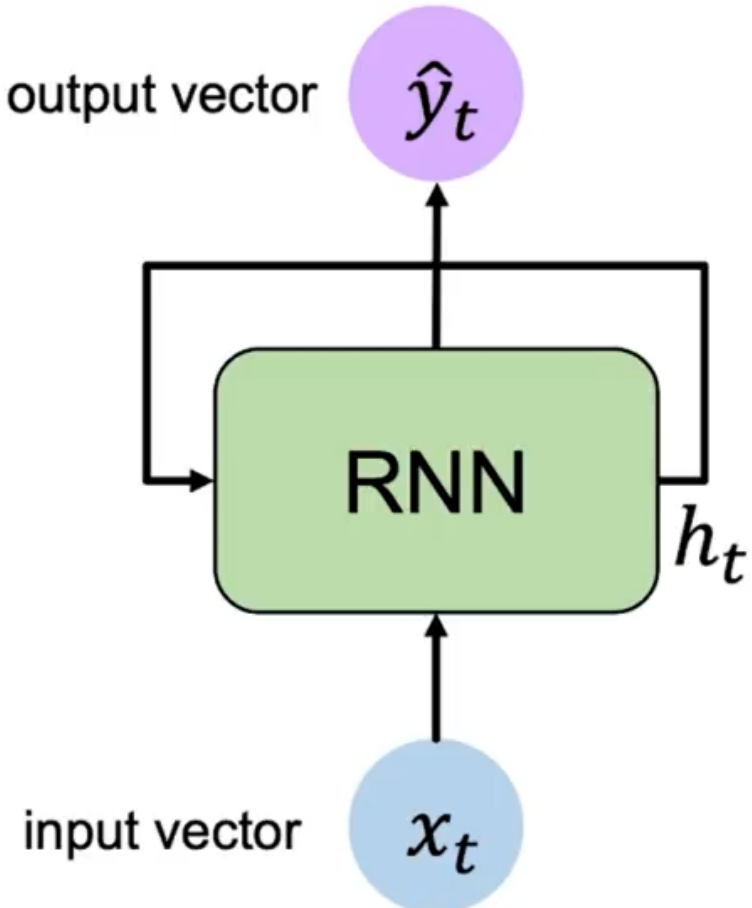


$$\hat{y}_t = f(x_t, h_{t-1})$$

output input past memory

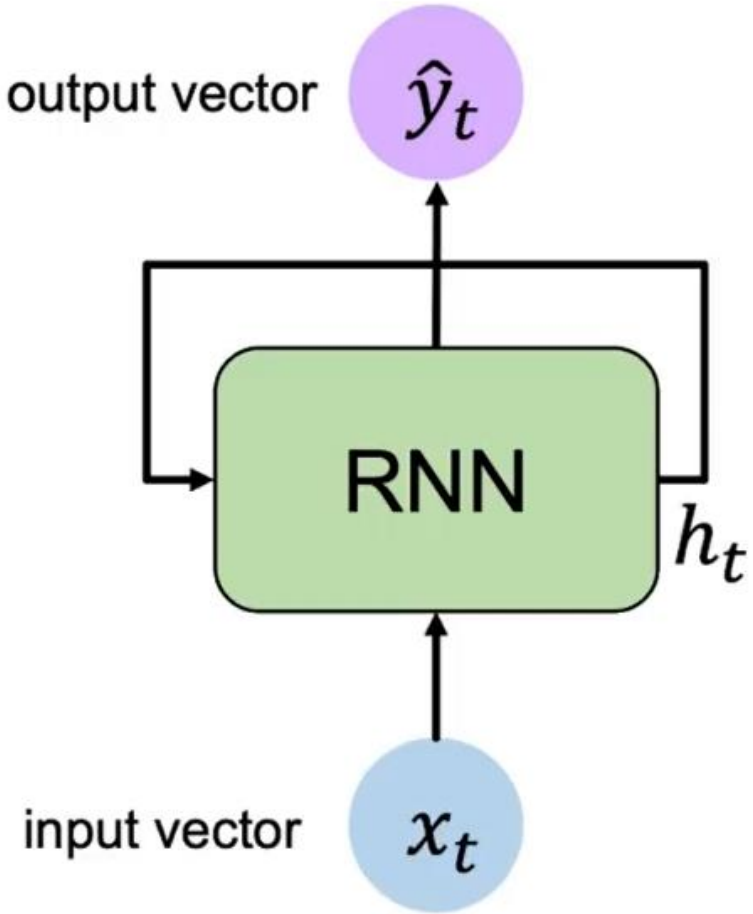
Vanilla Recurrent Neural Networks (RNNs)

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:

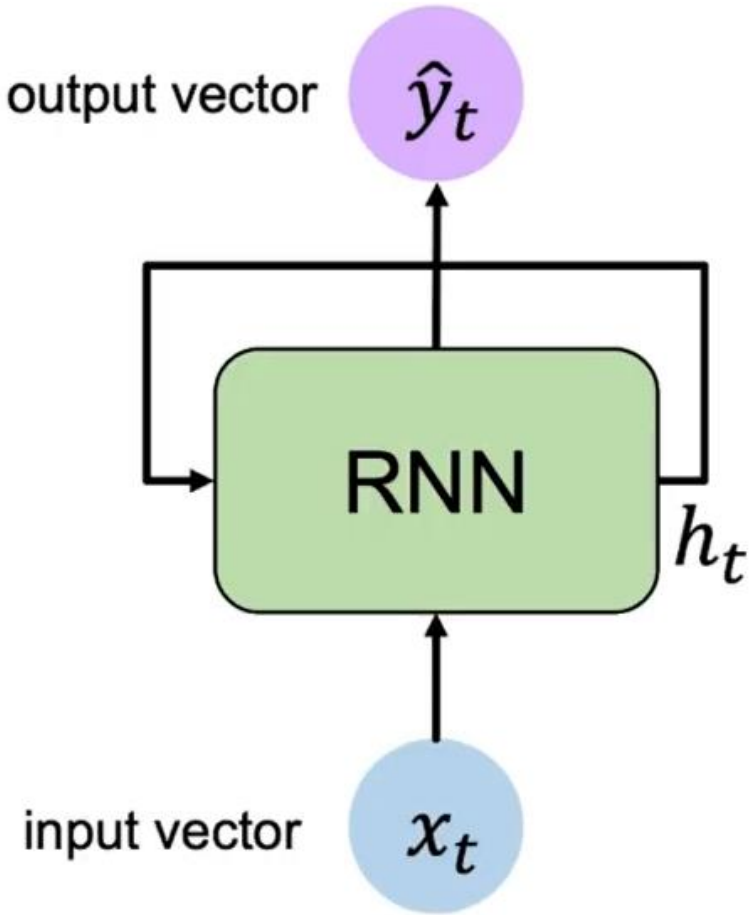
$$h_t = f_W(x_t, h_{t-1})$$

hidden 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:

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

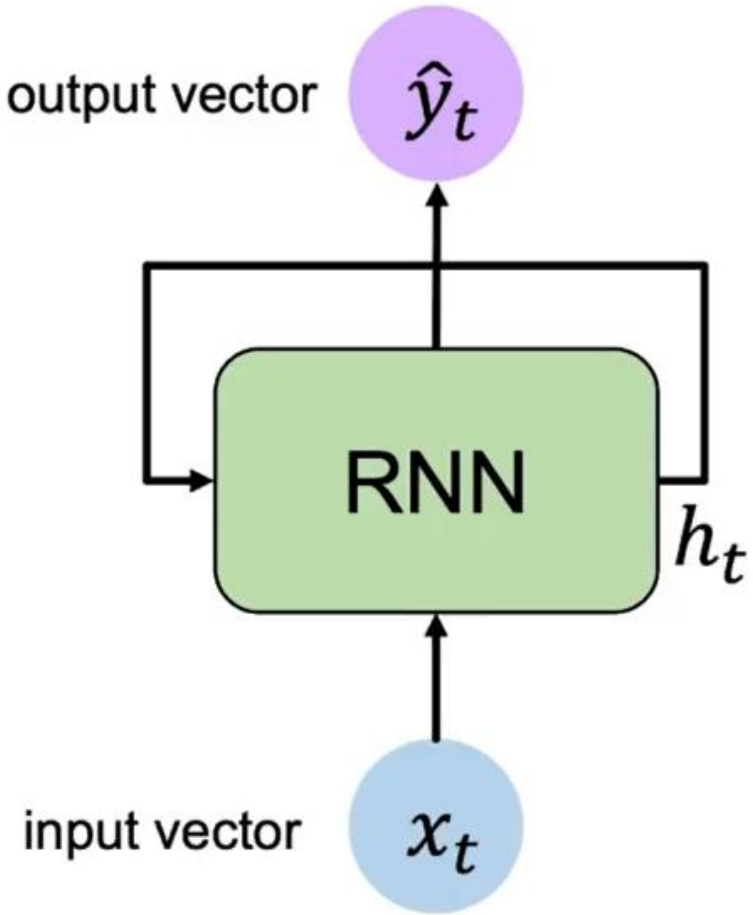
hidden state

function with weights W

input

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

Recurrent Neural Networks (RNNs)



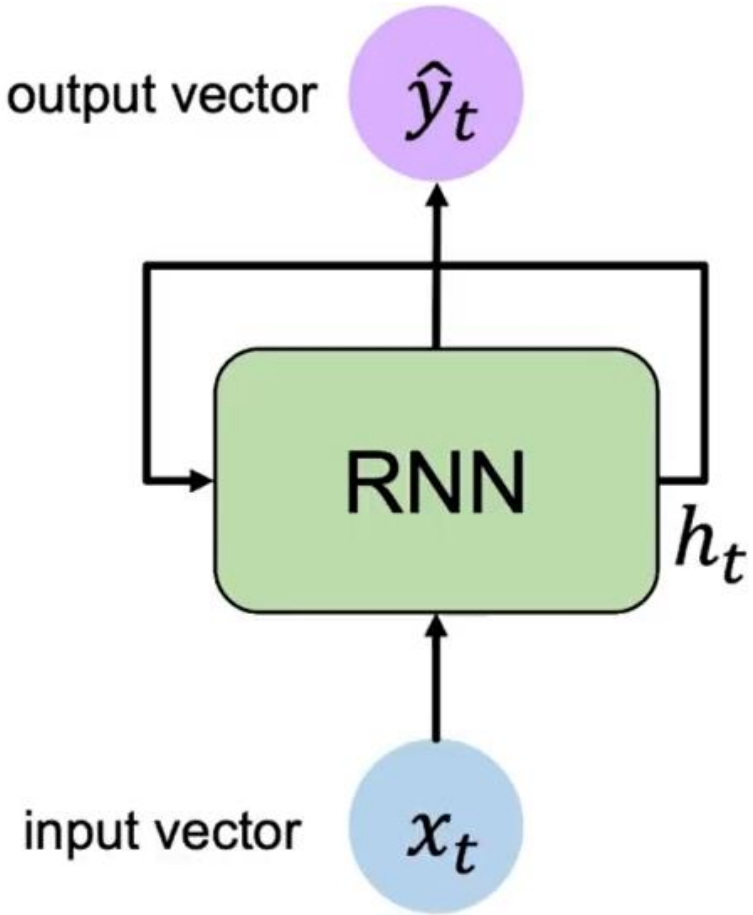
Apply a **recurrence relation** at every time step to process a sequence:

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

hidden 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:

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

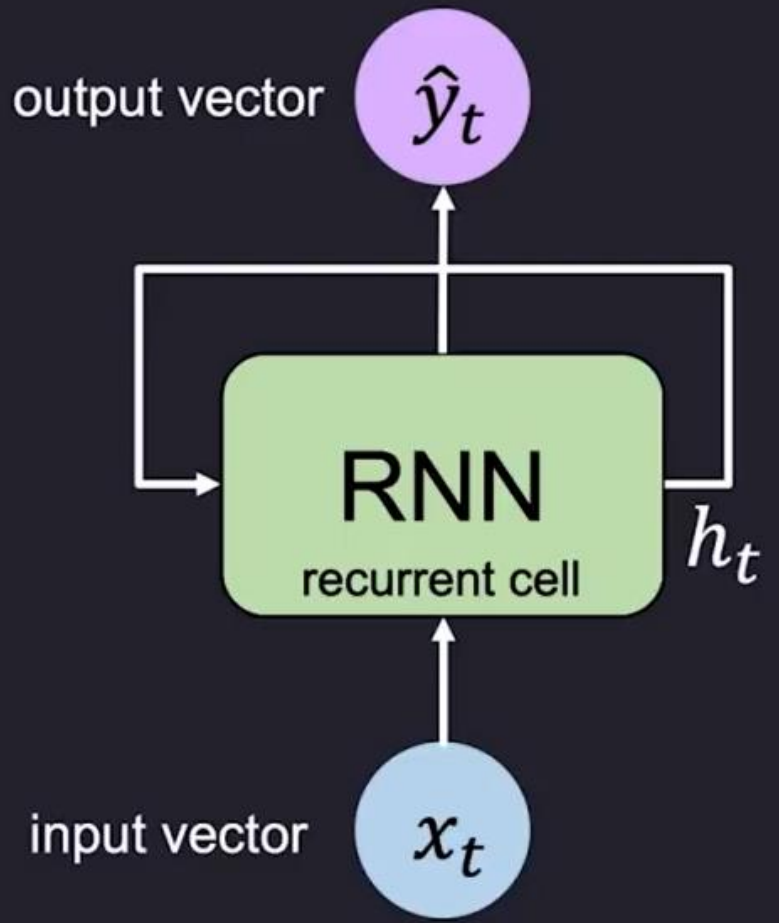
hidden state — function with weights W — input — old state

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

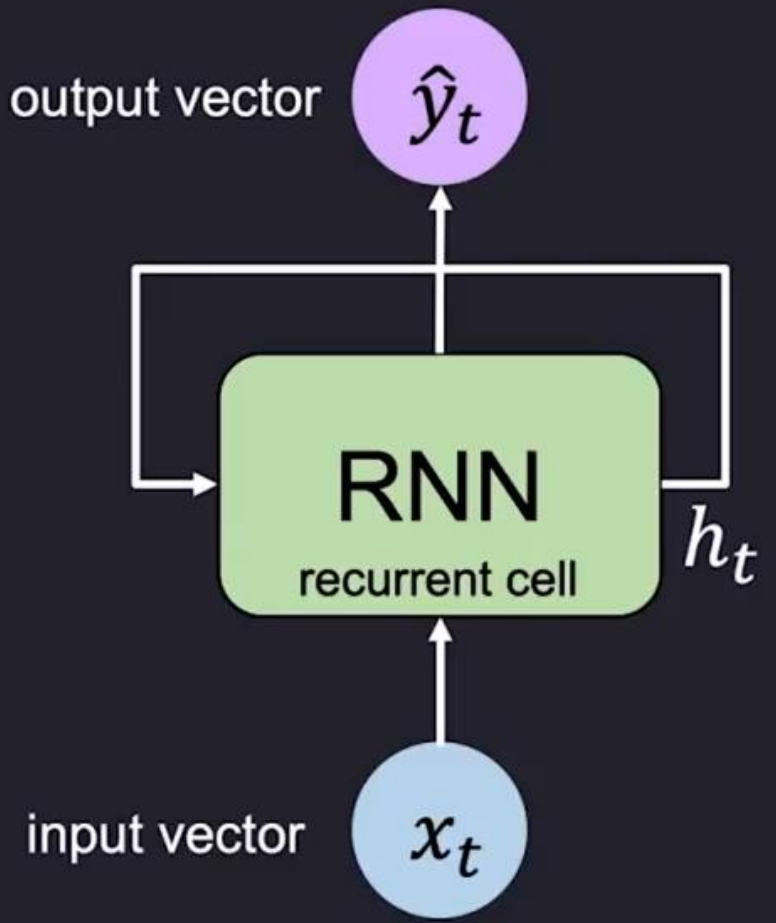
RNN Intuition

```
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!"
```



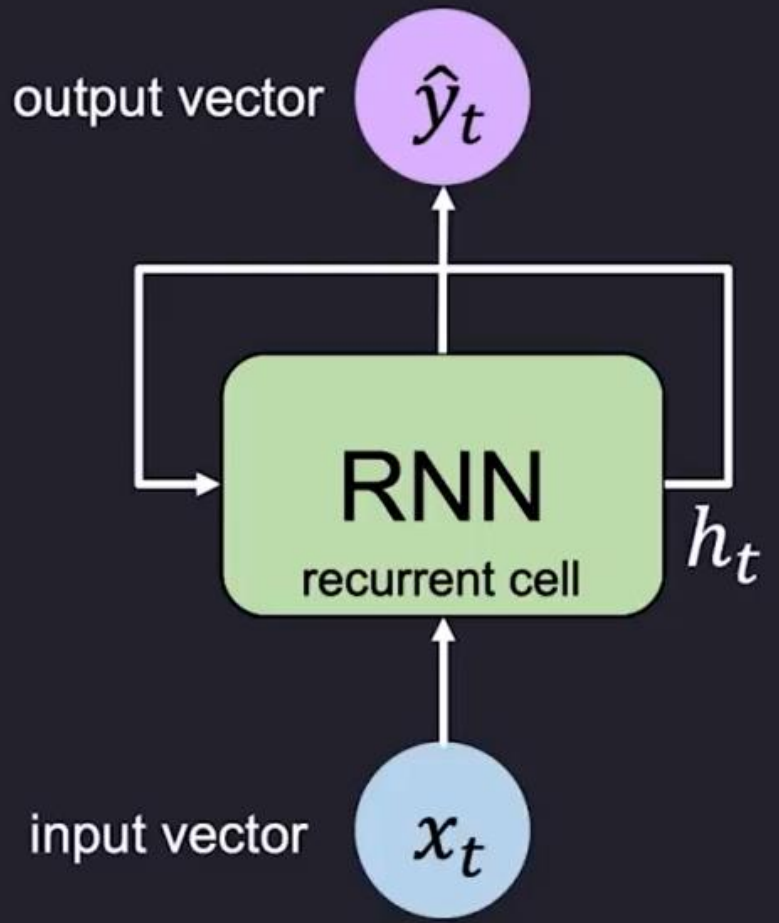
RNN Intuition

```
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!"
```

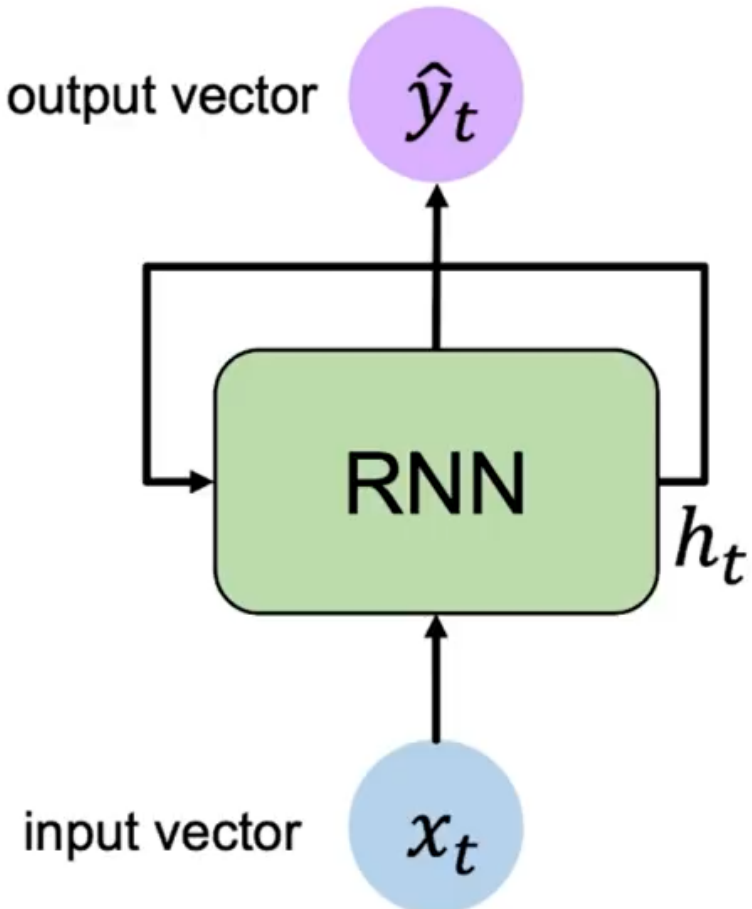


RNN Intuition

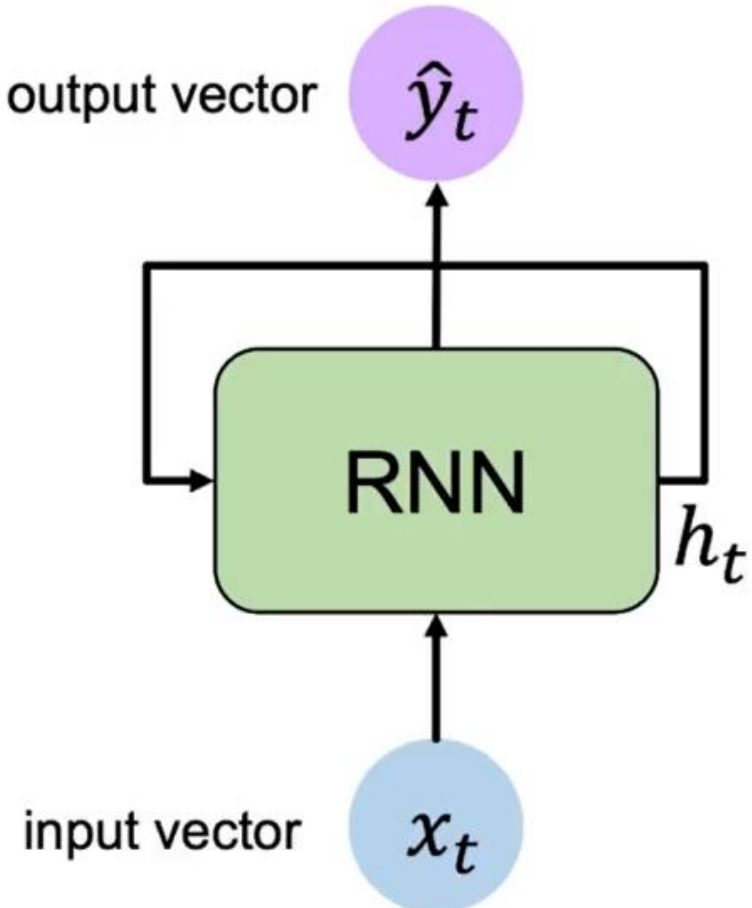
```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

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

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

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



RNN State Update and Output



RNN State Update and Output



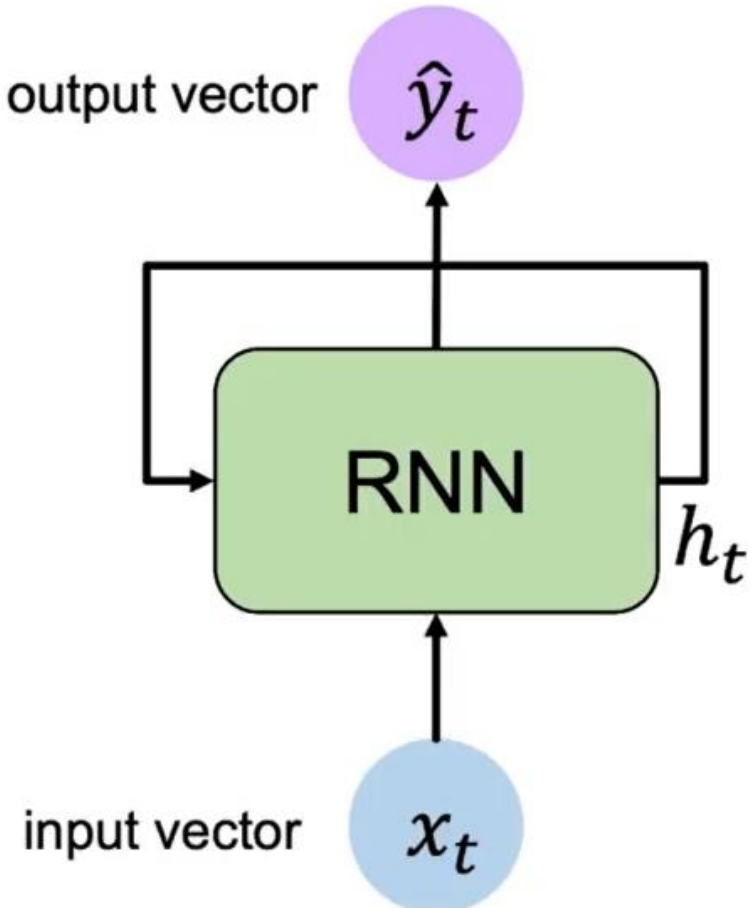
Update Hidden State

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

Input Vector

x_t

RNN State Update and Output



Output Vector

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

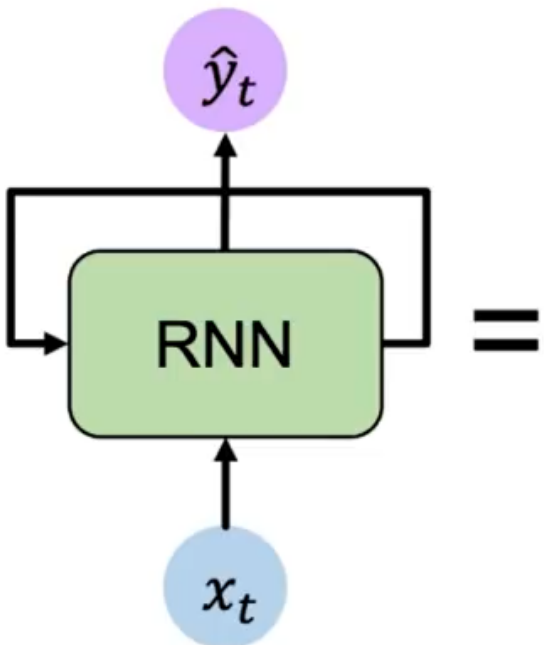
Update Hidden State

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

Input Vector

$$x_t$$

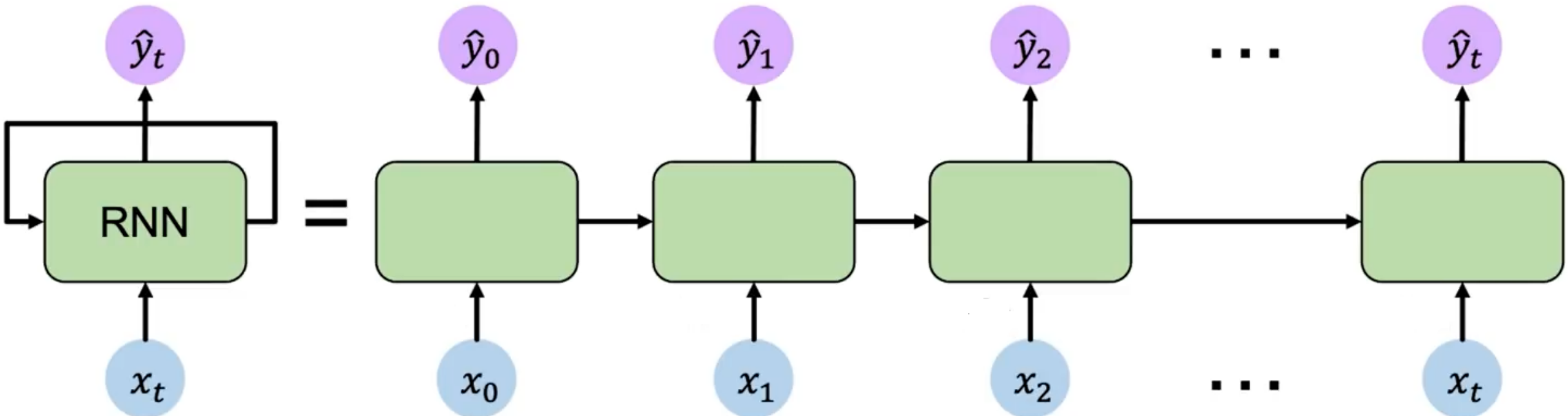
RNNs: Computational Graph Across Time



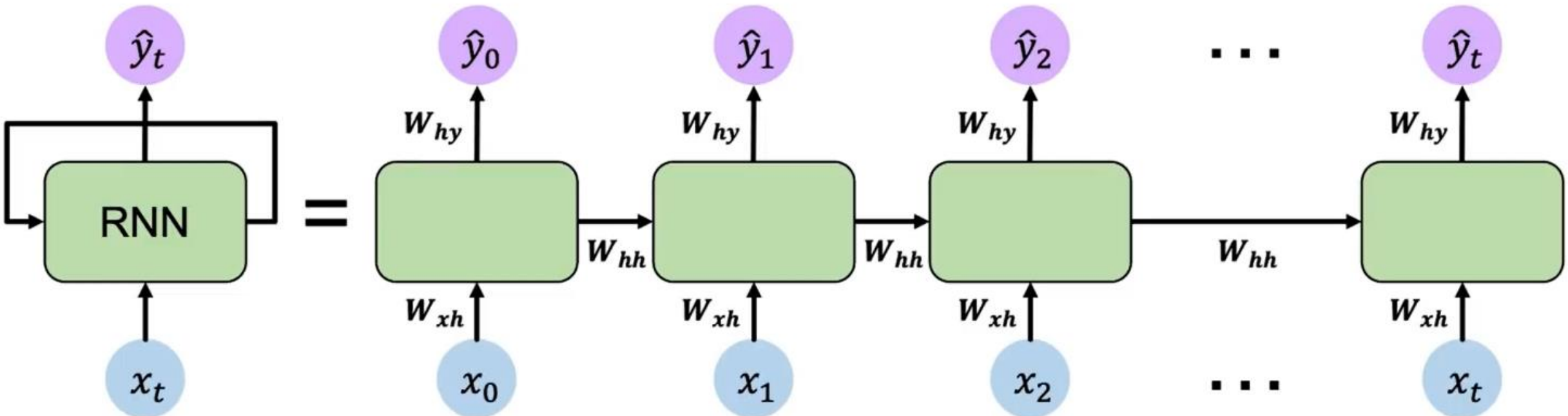
=

Represent as computational graph unrolled across time

RNNs: Computational Graph Across Time

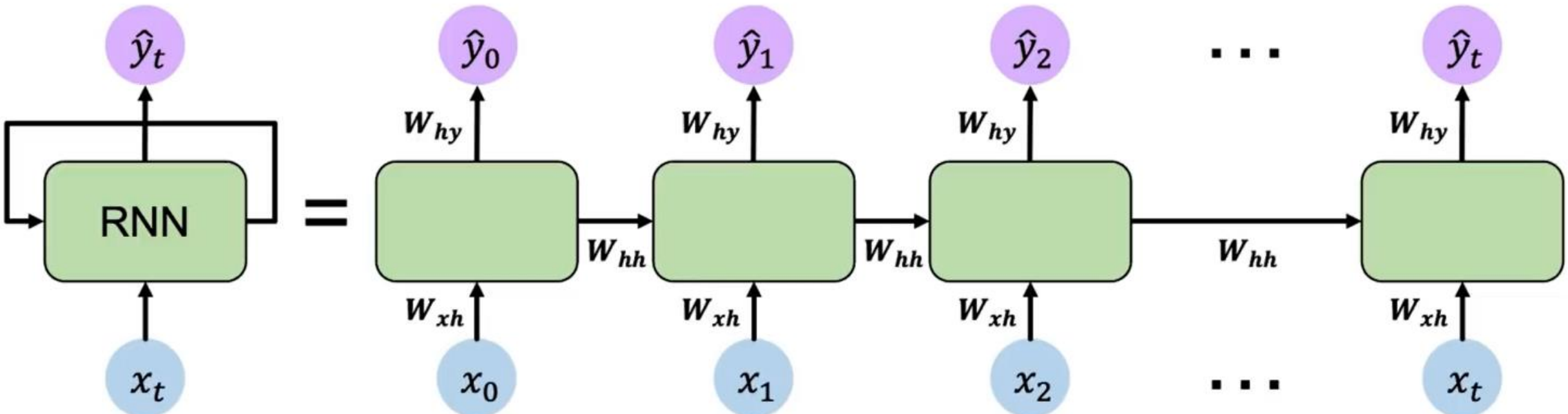


RNNs: Computational Graph Across Time

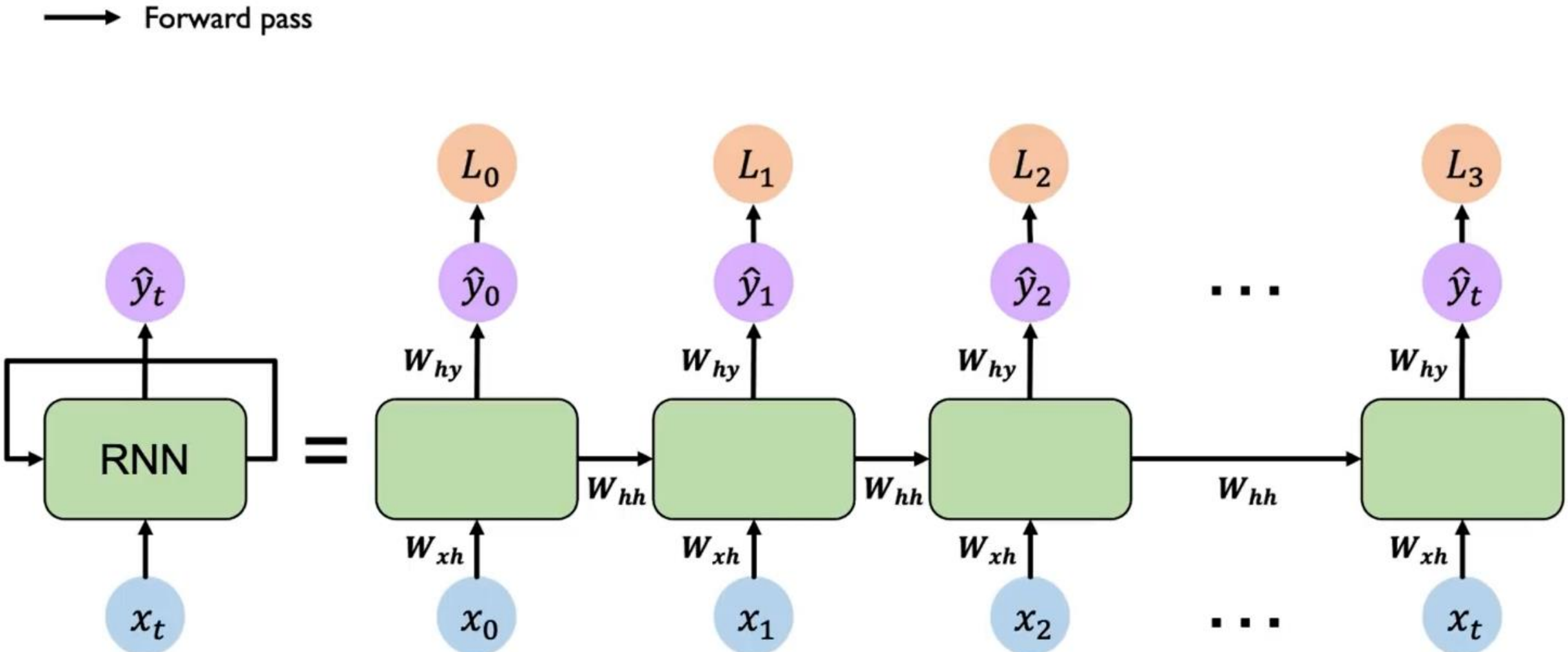


RNNs: Computational Graph Across Time

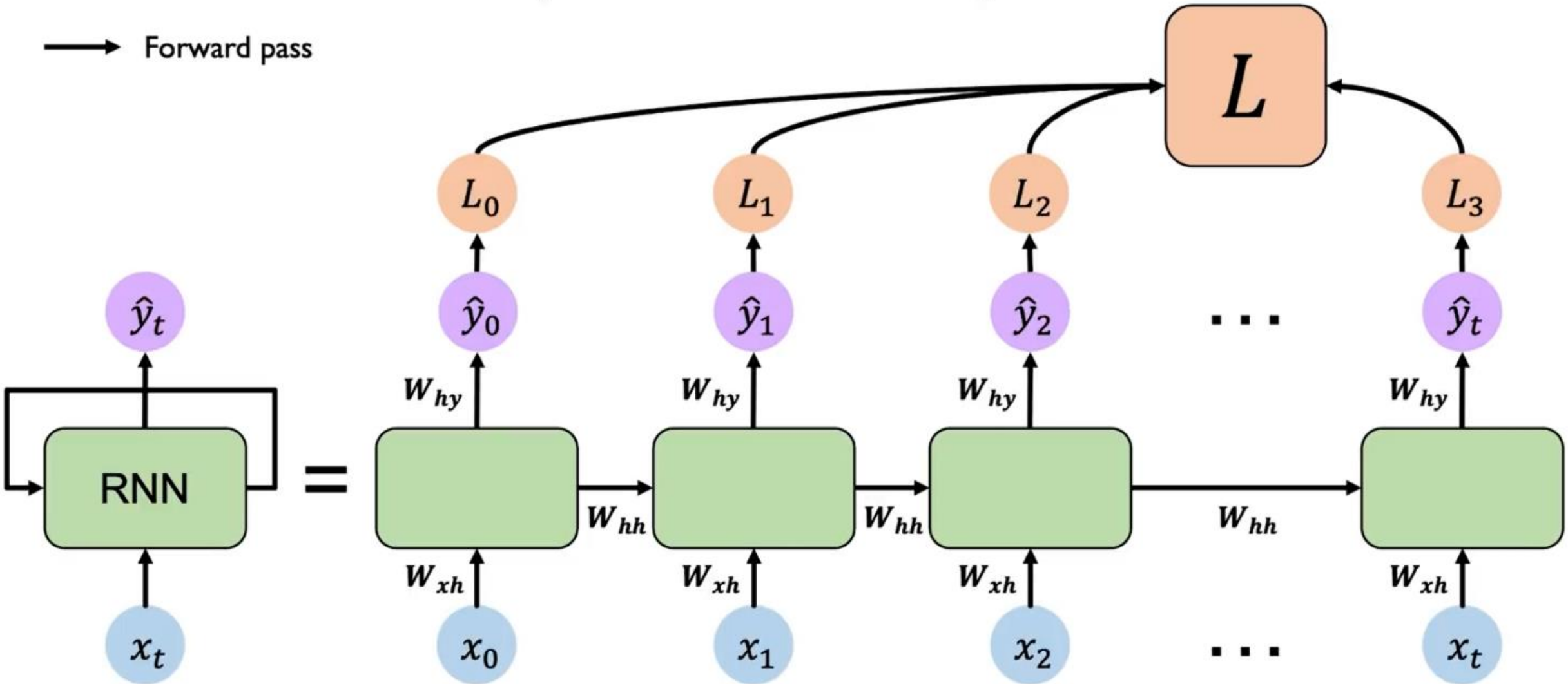
Re-use the **same weight matrices** at every time step



RNNs: Computational Graph Across Time

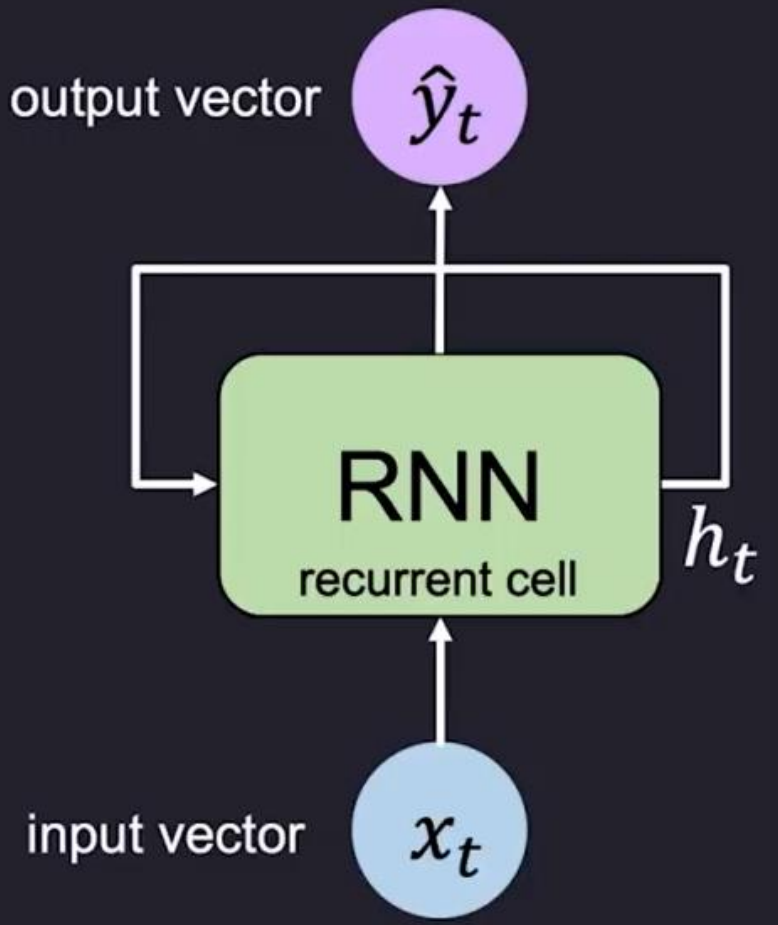


RNNs: Computational Graph Across Time



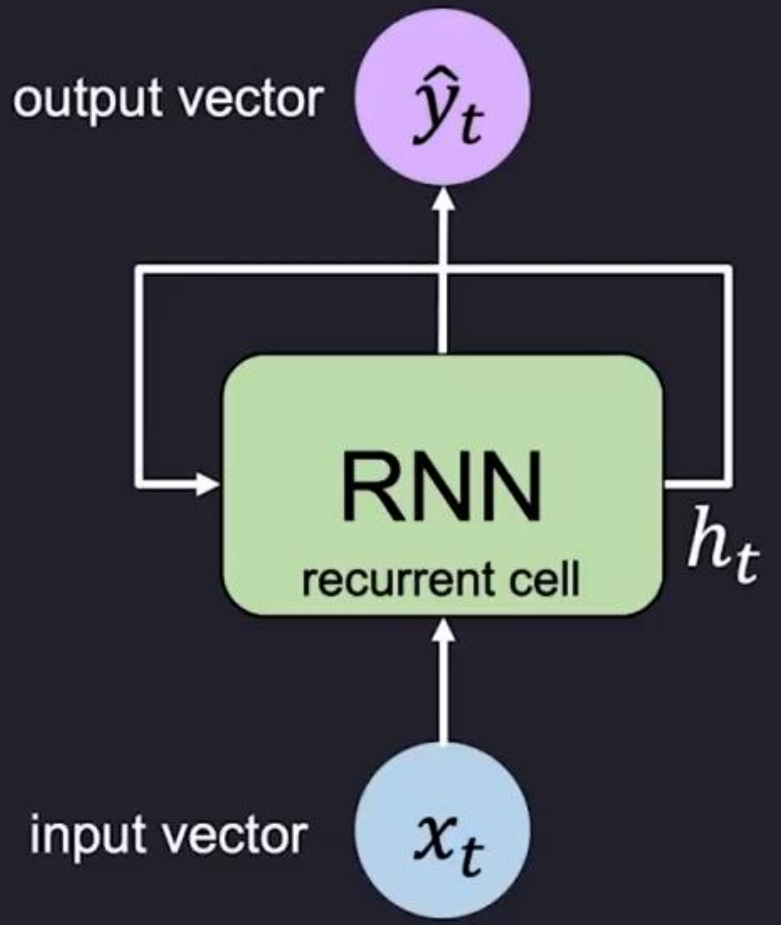
RNNs from Scratch

```
class MyRNNCell(torch.nn.Module):
  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])
```



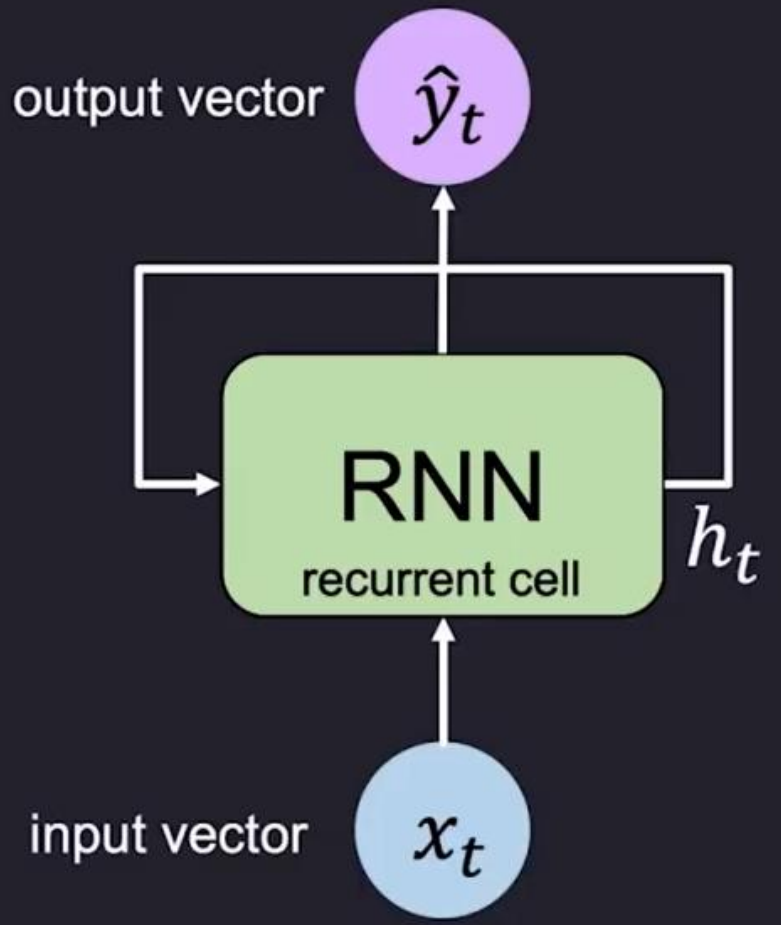
RNNs from Scratch

```
class MyRNNCell (torch.nn.Module):
  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
```



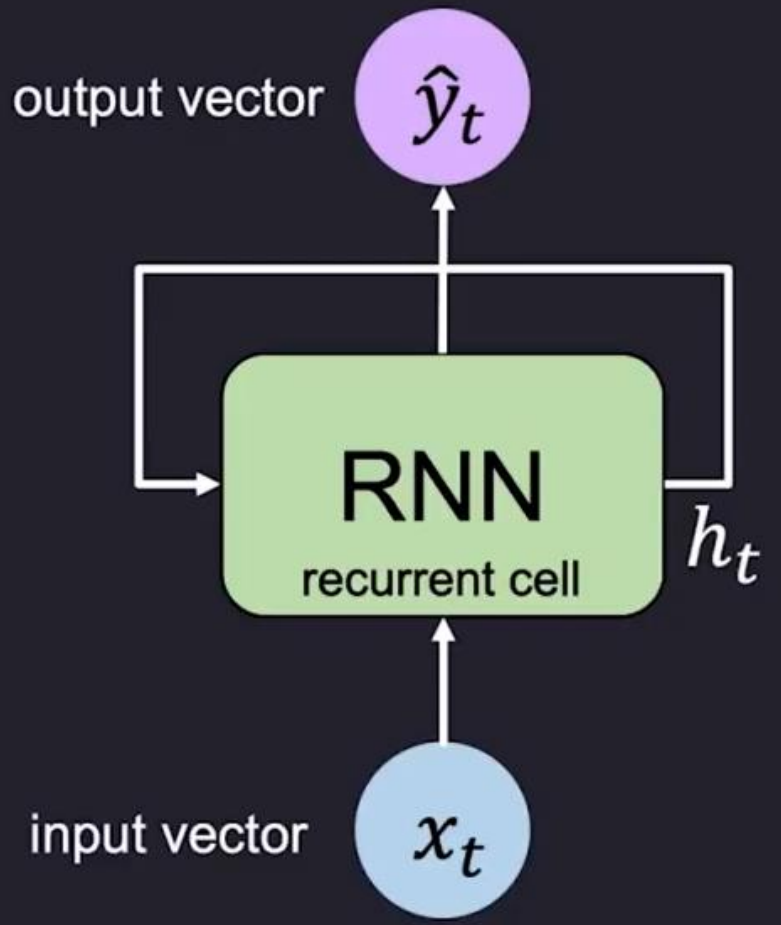
RNNs from Scratch

```
class MyRNNCell(torch.nn.Module):
  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
```

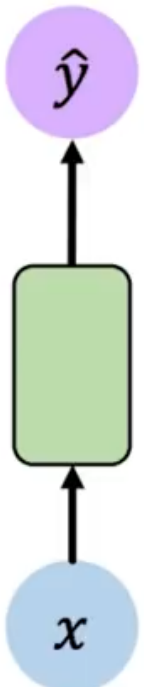


RNNs from Scratch

```
class MyRNNCell (torch.nn.Module):
  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
```



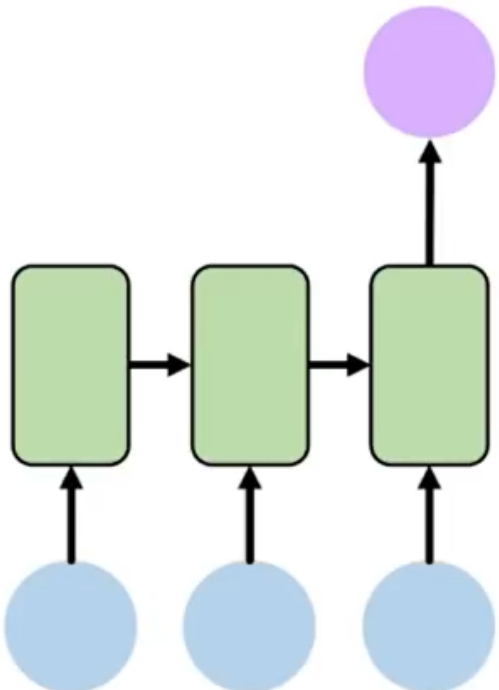
RNNs for Sequence Modeling



One to One
"Vanilla" NN
Binary classification

RNNs for Sequence Modeling

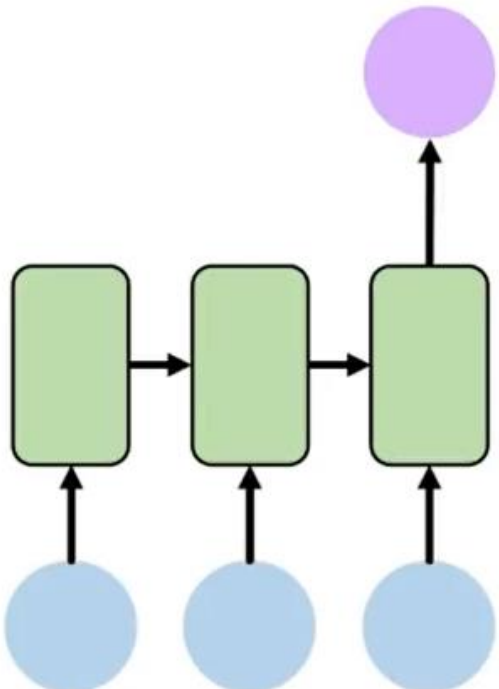
One to One
"Vanilla" NN
Binary classification



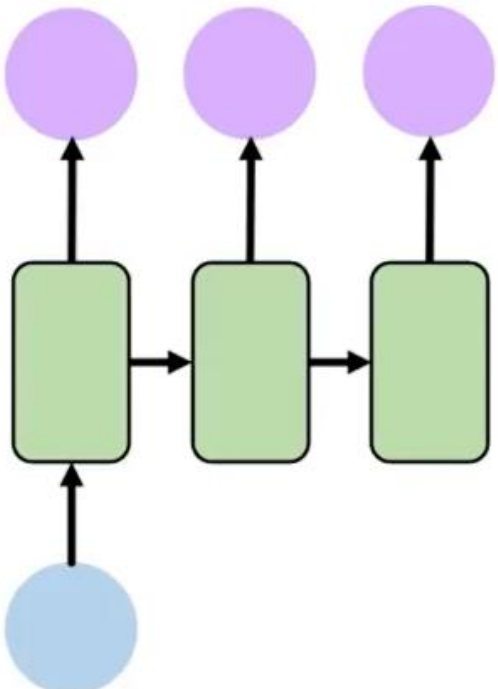
Many to One
Sentiment Classification

RNNs for Sequence Modeling

One to One
"Vanilla" NN
Binary classification



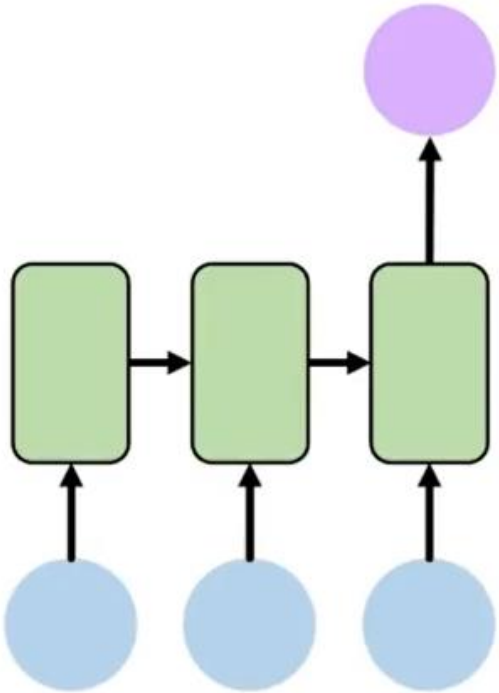
Many to One
Sentiment Classification



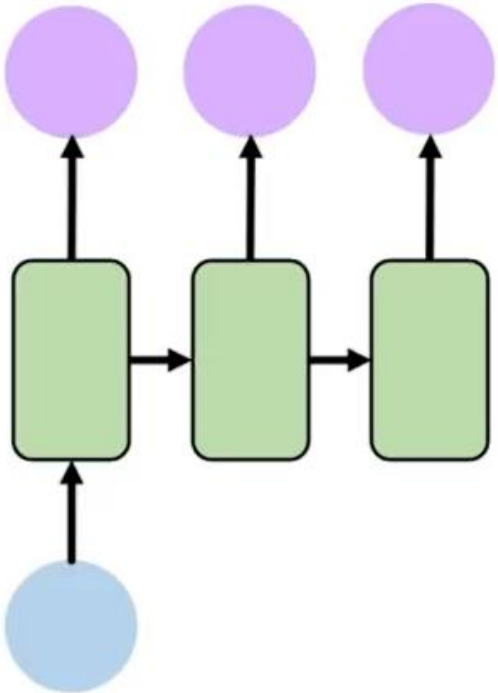
One to Many
Text Generation
Image Captioning

RNNs for Sequence Modeling

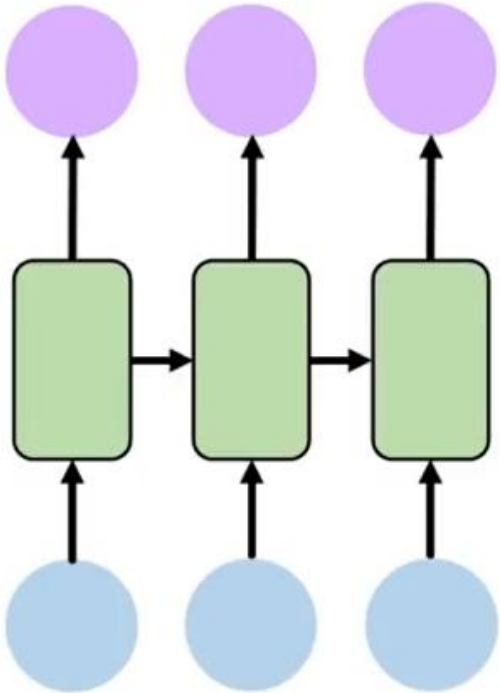
One to One
"Vanilla" NN
Binary classification



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:

Sequence Modeling: Design Criteria

To model sequences, we need to:

1. Handle **variable-length** sequences

Sequence Modeling: Design Criteria

To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies

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**

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

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

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

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

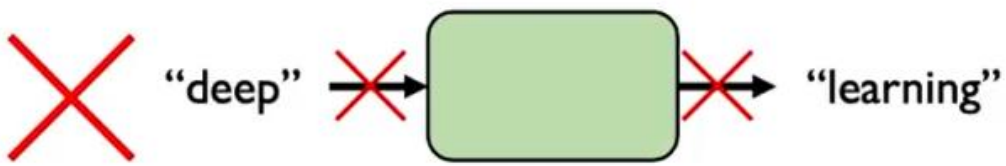
A Sequence Modeling Problem: Predict the Next Word

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

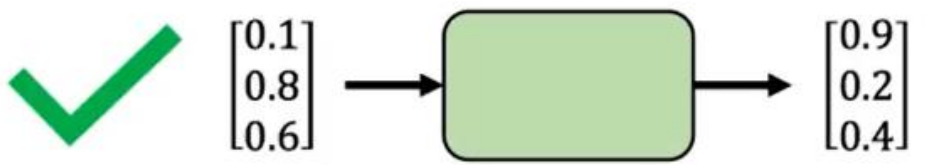
given these words

predict the next word

Representing Language to a Neural Network

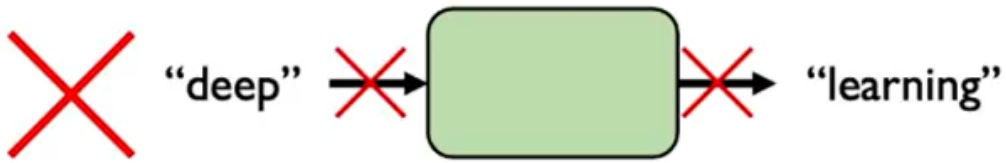


Neural networks cannot interpret words

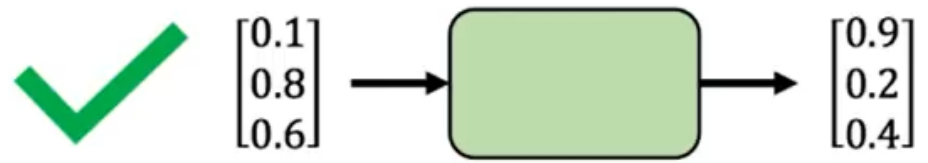


Neural networks require numerical inputs

Encoding Language for a Neural Network



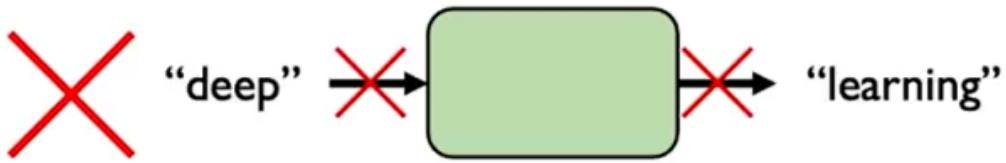
Neural networks cannot interpret words



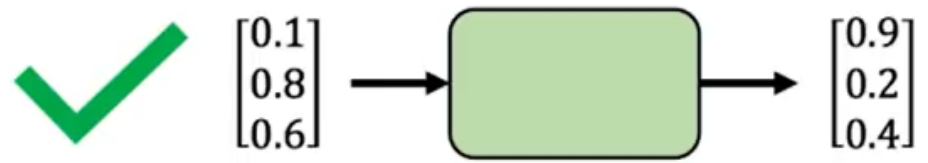
Neural networks require numerical inputs

Embedding: transform indexes into a vector of fixed size.

Encoding Language for a Neural Network

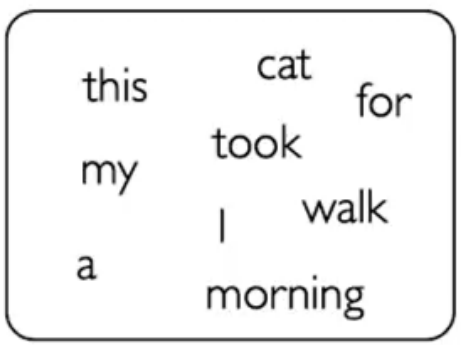


Neural networks cannot interpret words



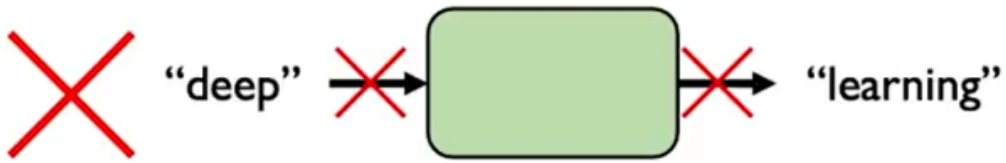
Neural networks require numerical inputs

Embedding: transform indexes into a vector of fixed size.

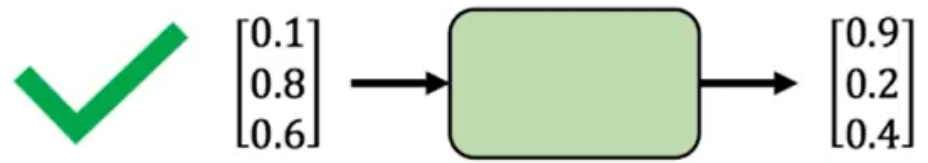


1. Vocabulary:
Corpus of words

Encoding Language for a Neural Network

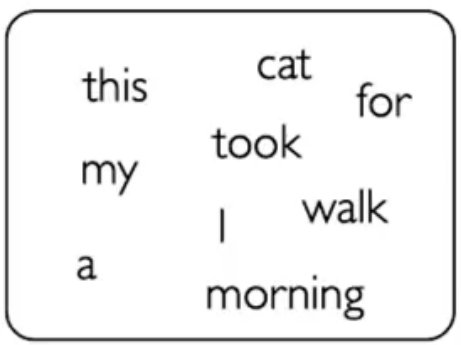


Neural networks cannot interpret words

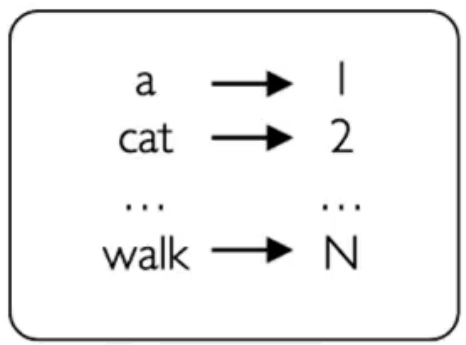


Neural networks require numerical inputs

Embedding: transform indexes into a vector of fixed size.

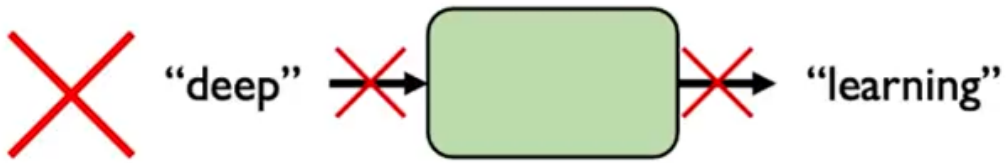


1. Vocabulary:
Corpus of words

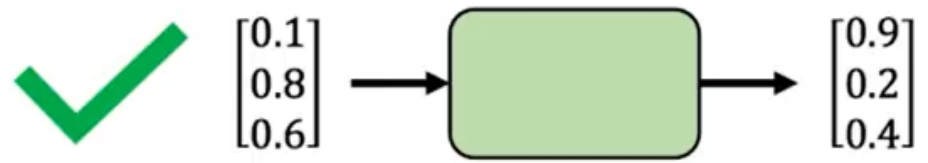


2. Indexing:
Word to index

Encoding Language for a Neural Network

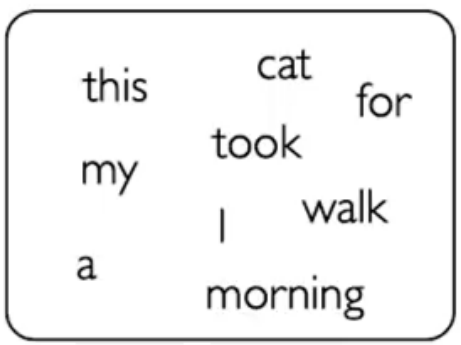


Neural networks cannot interpret words

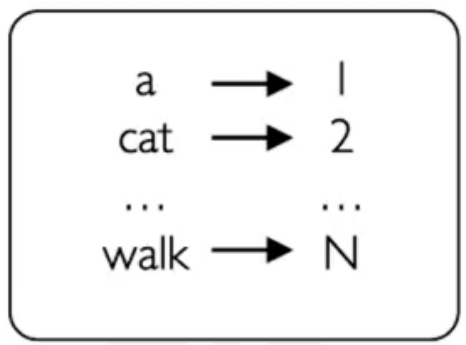


Neural networks require numerical inputs

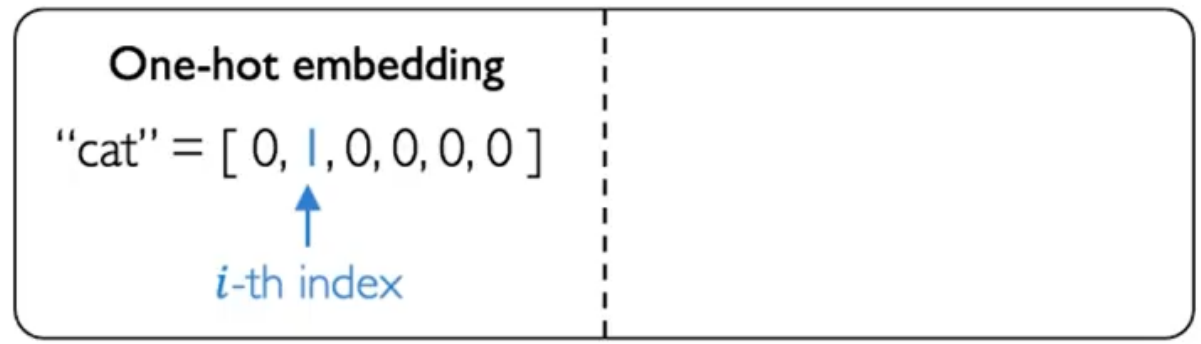
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words

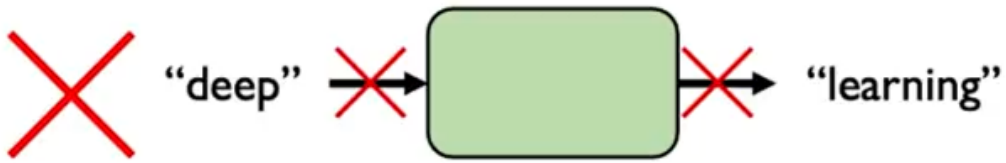


2. Indexing:
Word to index

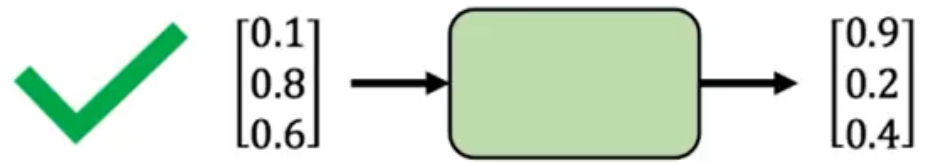


3. Embedding:
Index to fixed-sized vector

Encoding Language for a Neural Network

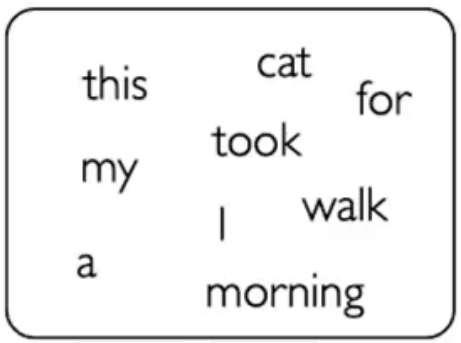


Neural networks cannot interpret words

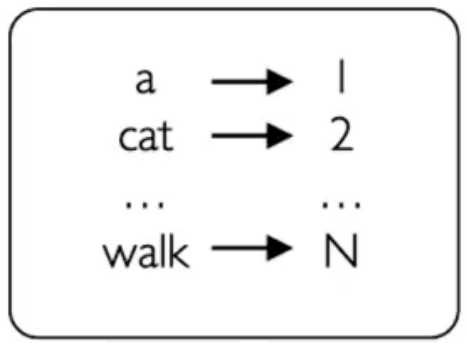


Neural networks require numerical inputs

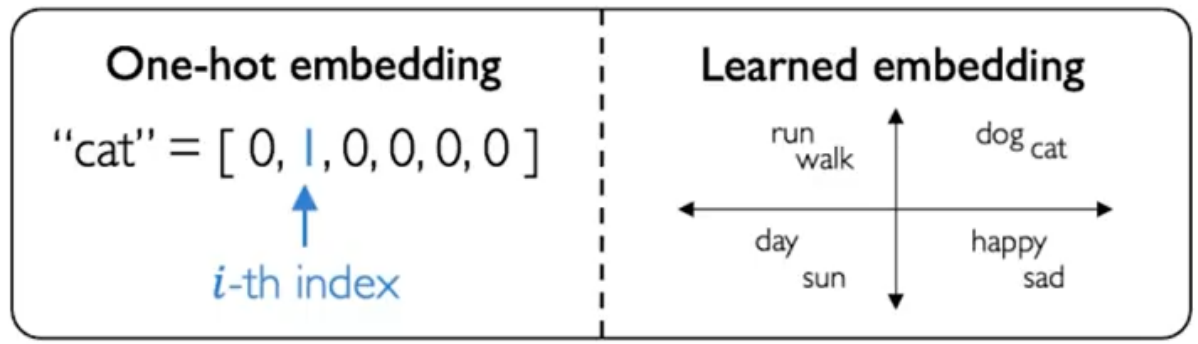
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words

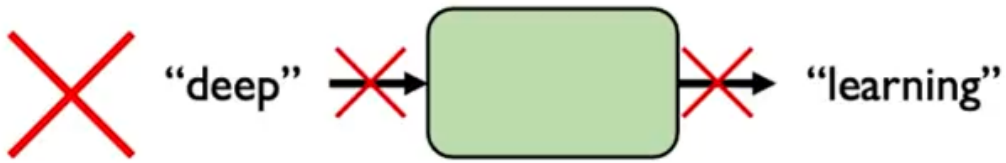


2. Indexing:
Word to index

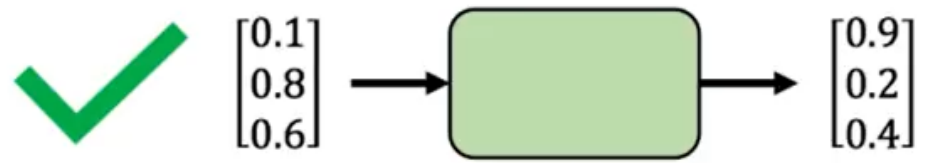


3. Embedding:
Index to fixed-sized vector

Encoding Language for a Neural Network

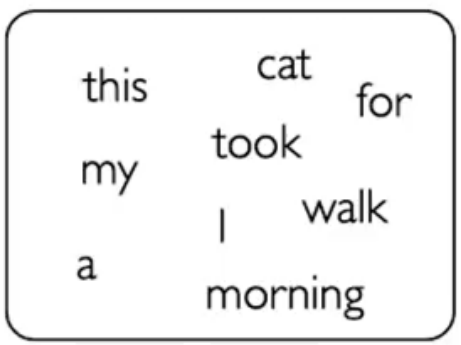


Neural networks cannot interpret words

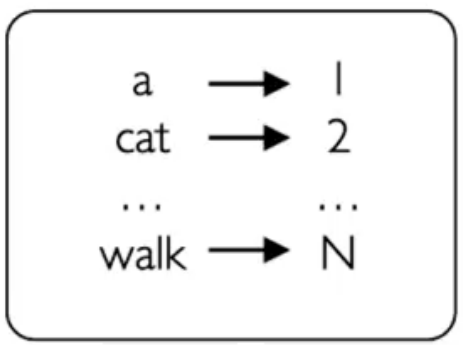


Neural networks require numerical inputs

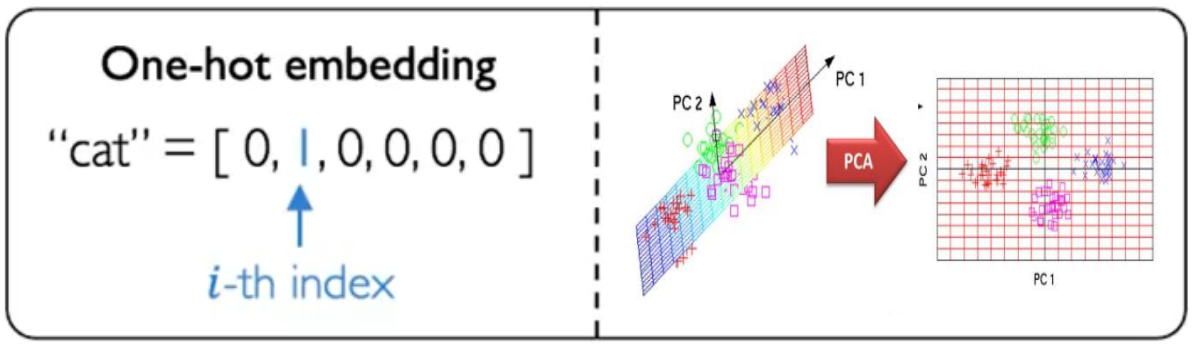
Embedding: transform indexes into a vector of fixed size.



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.

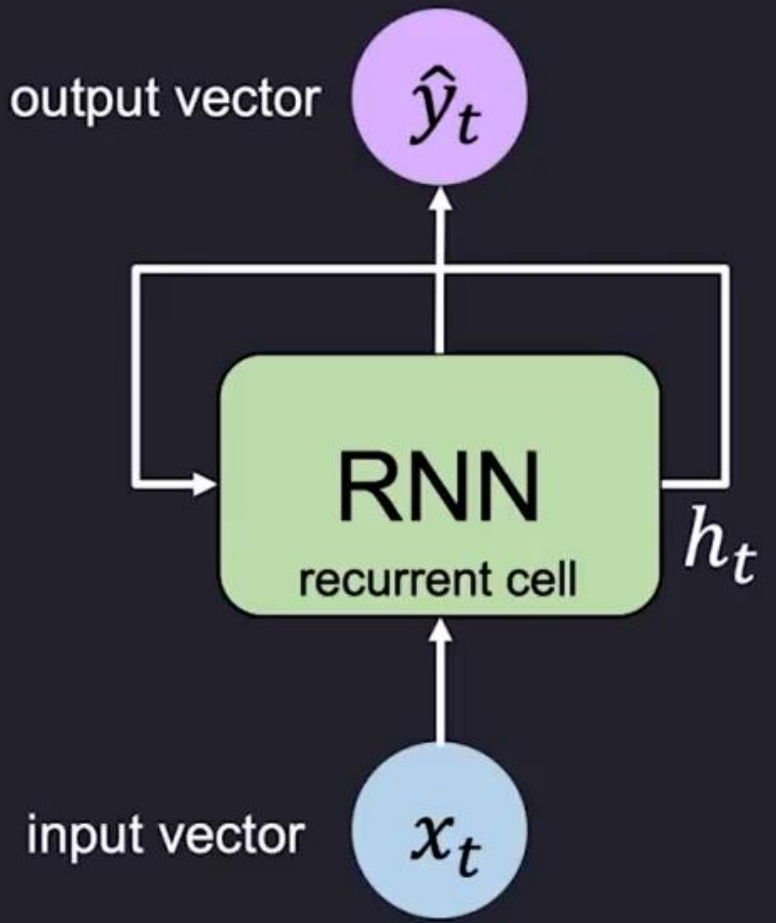
We visited a restaurant for lunch

vs.

We were hungry but cleaned the house before eating

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!"
```



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.

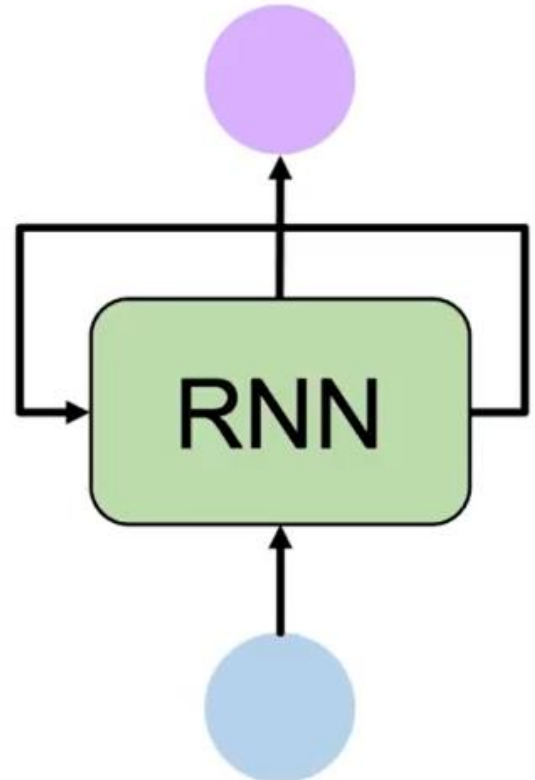
vs.

The food was bad, not good at all.

Sequence Modeling: Design Criteria

To model sequences, we need to:

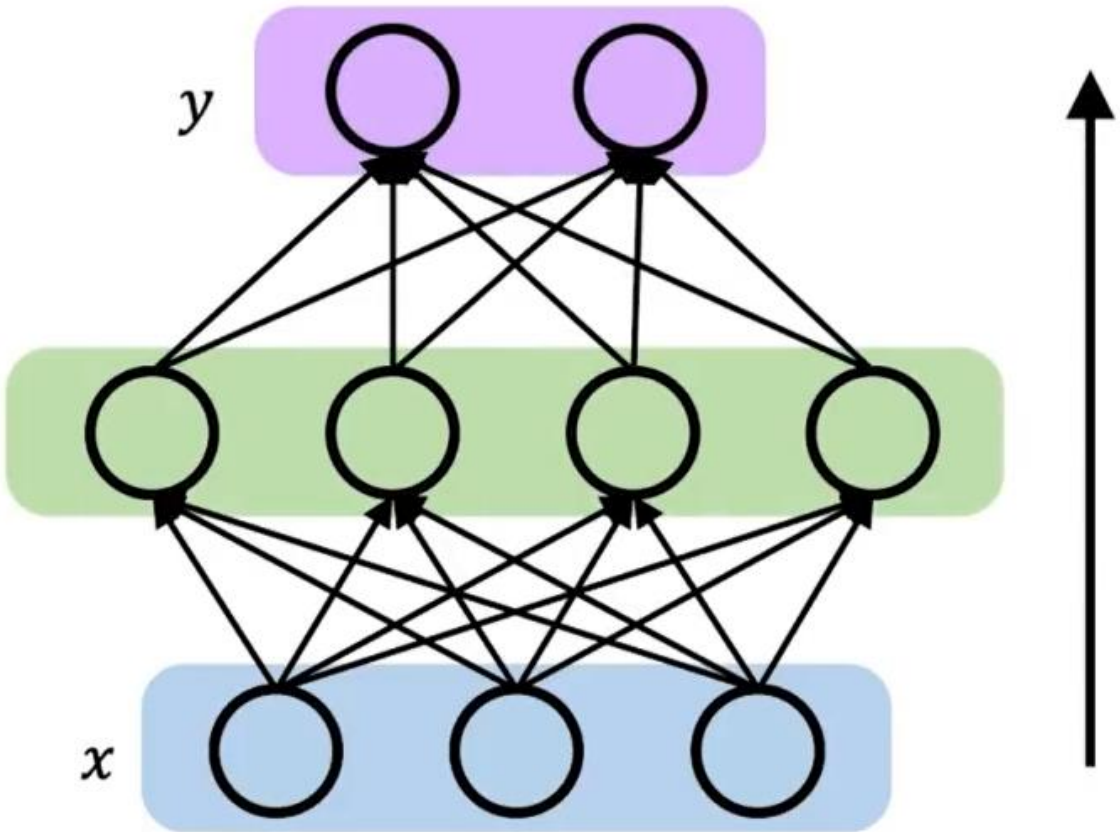
1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



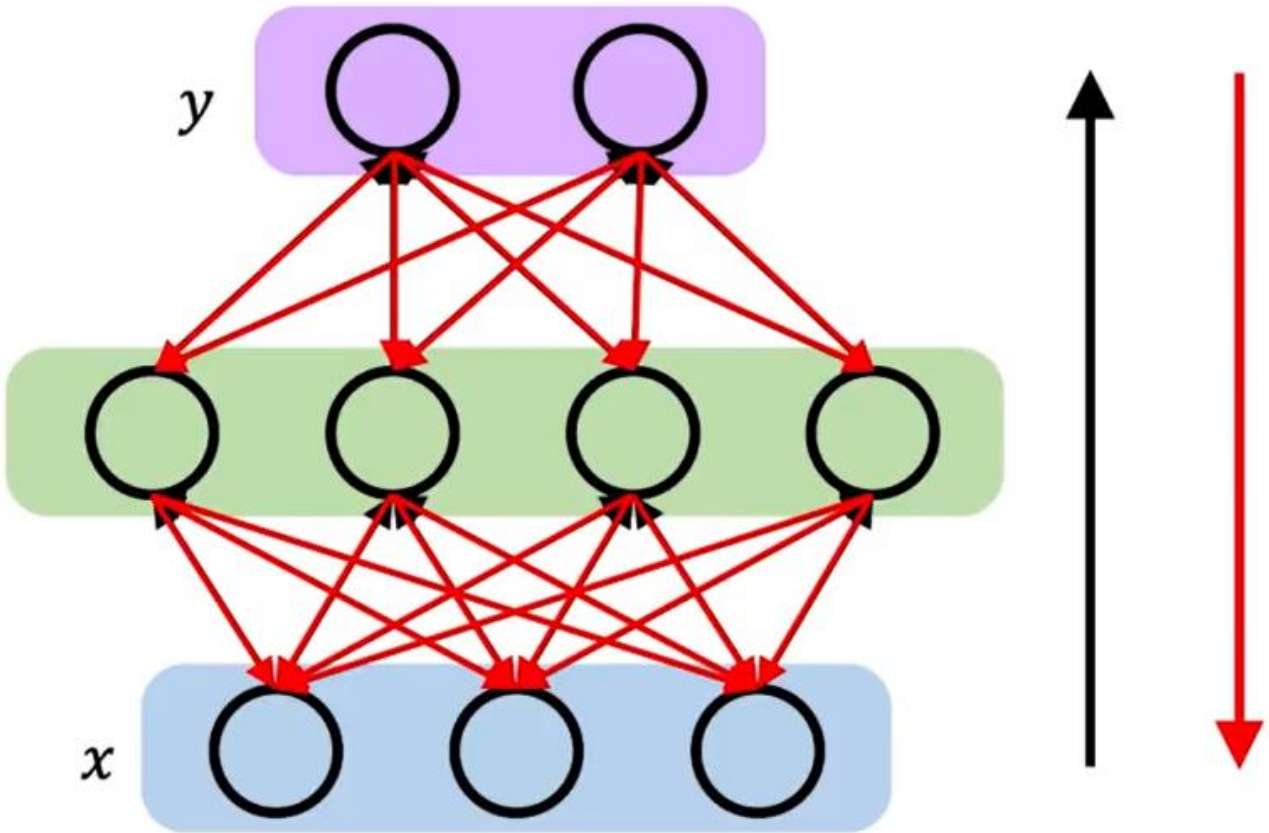
Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria

Backpropagation Through Time (BPTT)

Recall: Backpropagation in Feed Forward Models



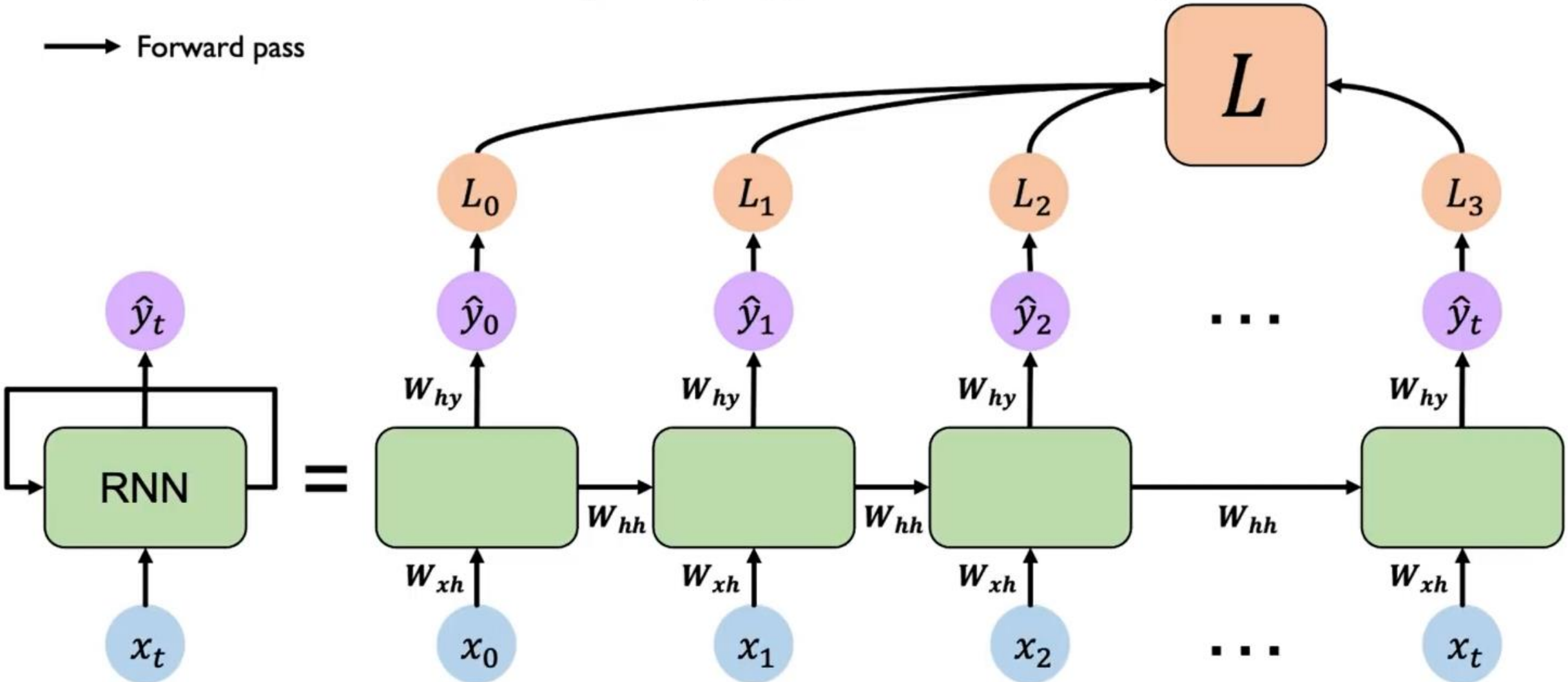
Recall: Backpropagation in Feed Forward Models



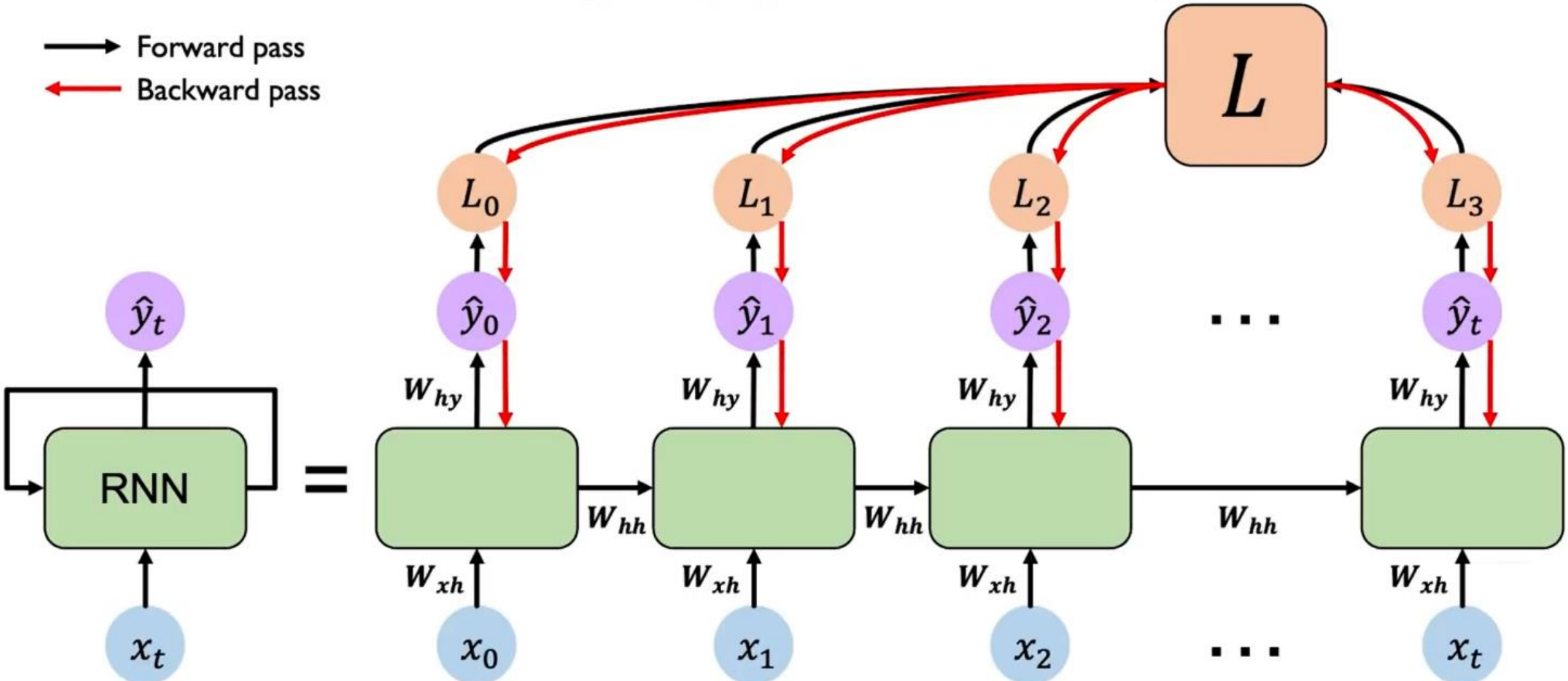
Backpropagation algorithm:

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

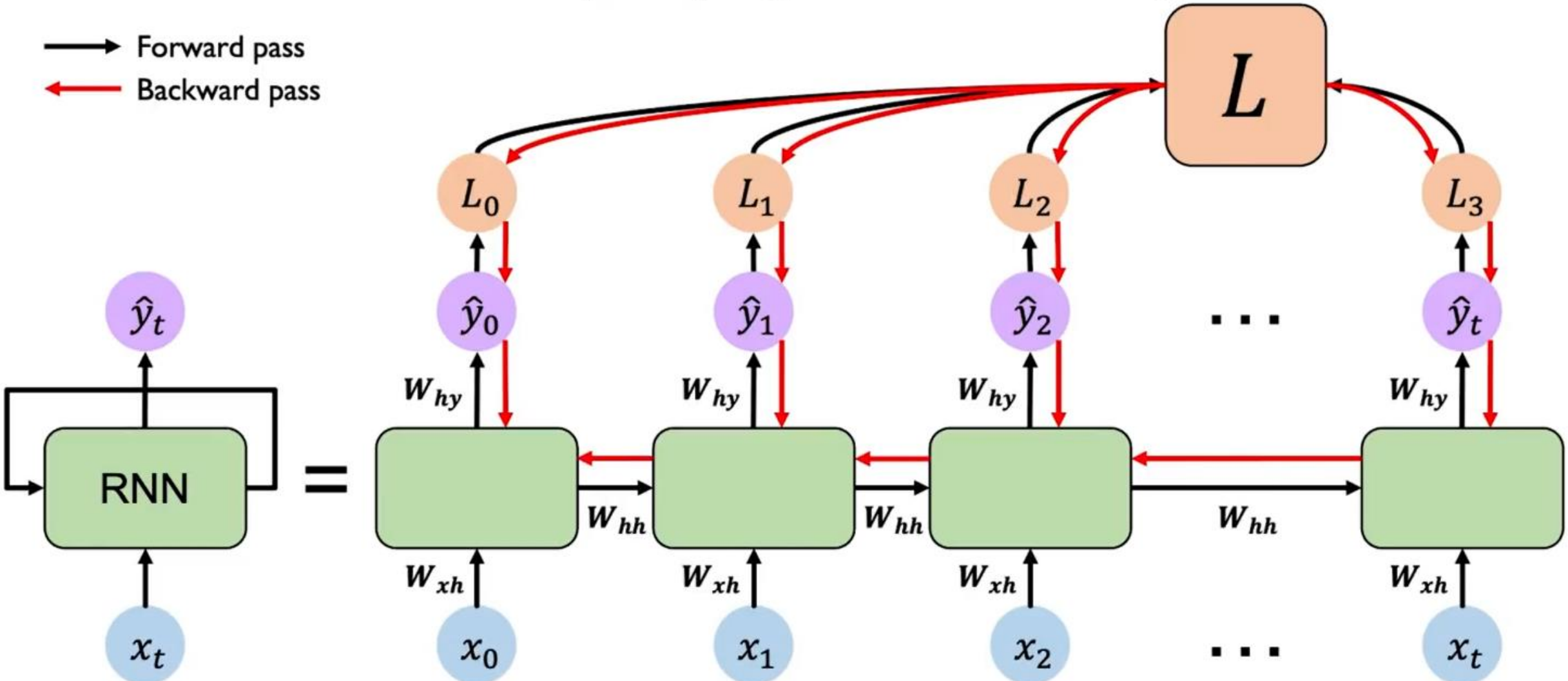
RNNs: Backpropagation Through Time



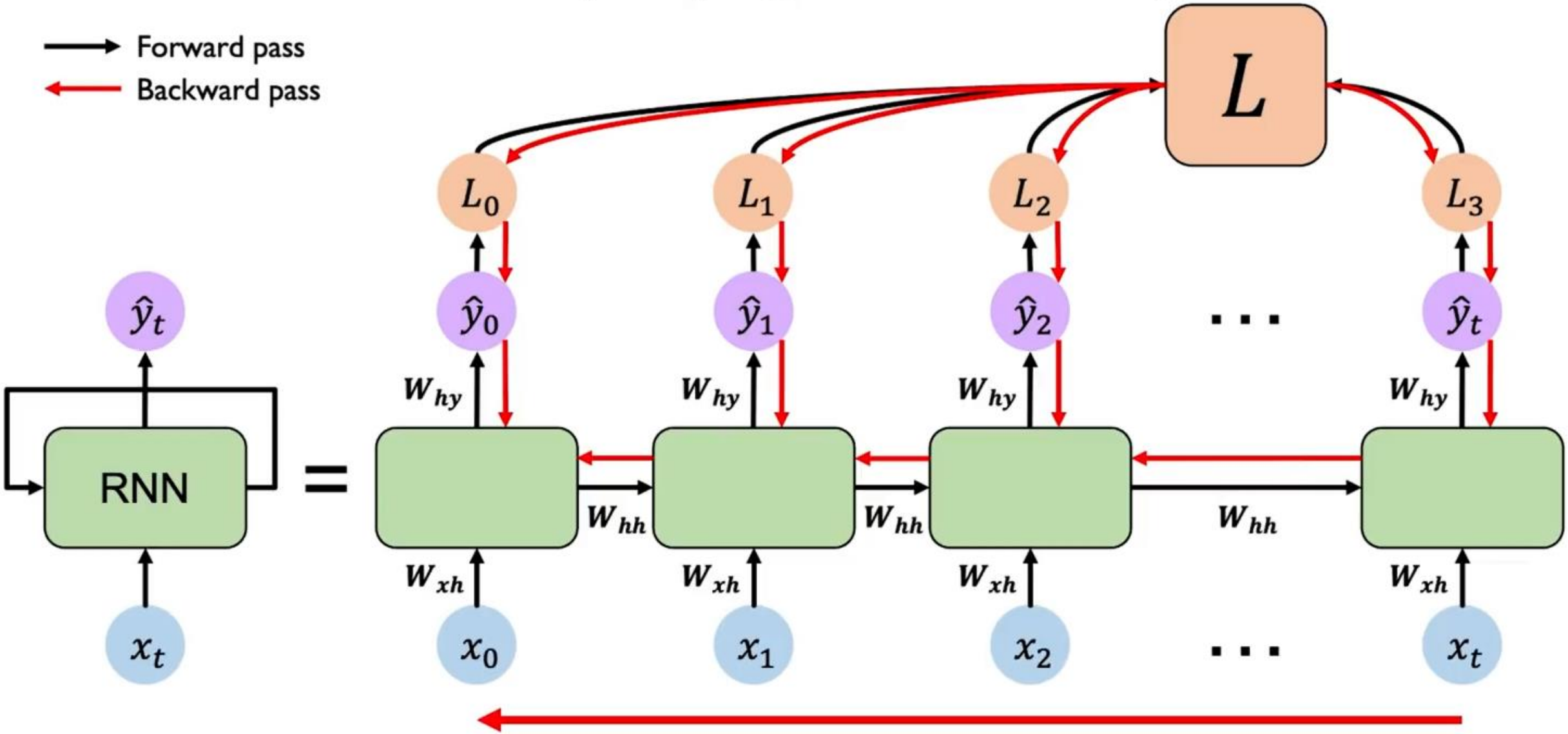
RNNs: Backpropagation Through Time



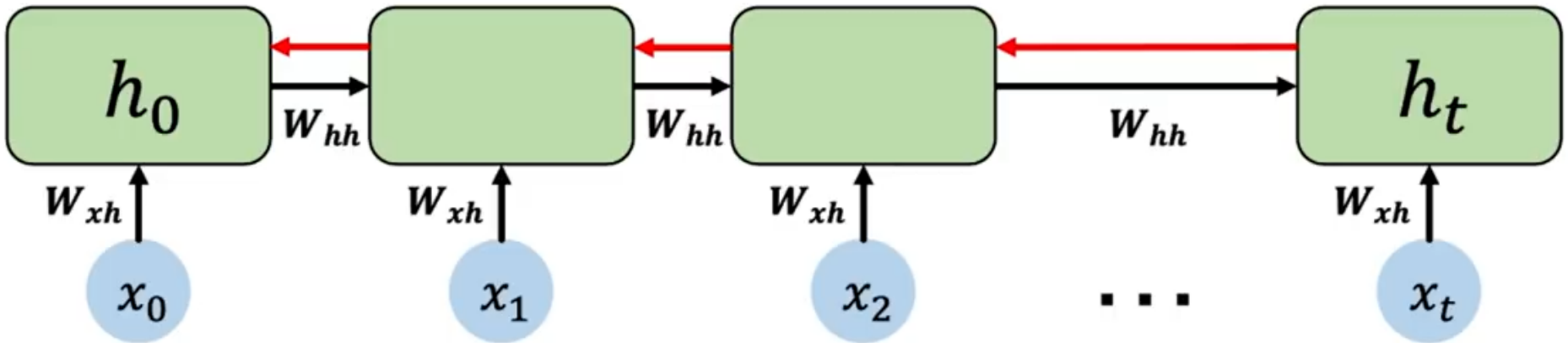
RNNs: Backpropagation Through Time



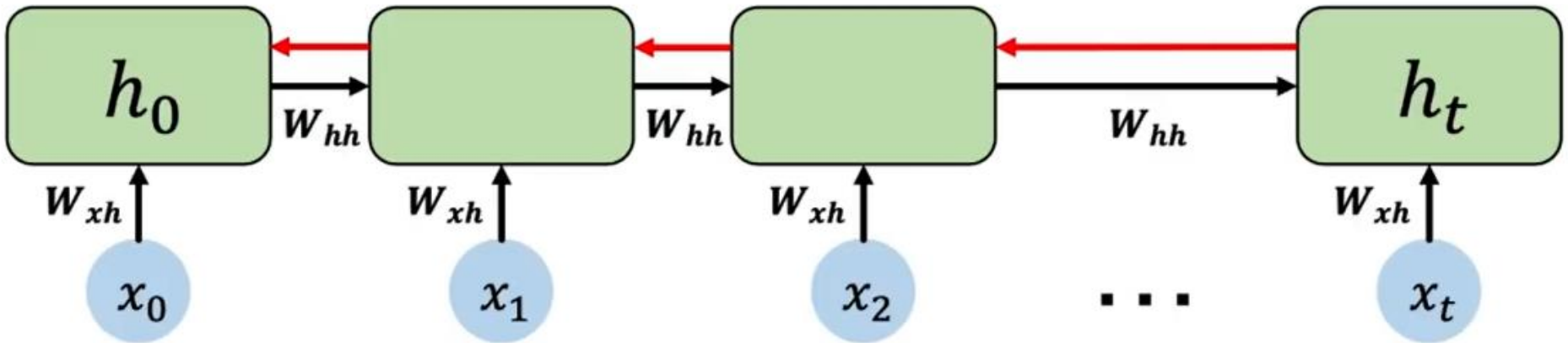
RNNs: Backpropagation Through Time



Standard RNN Gradient Flow

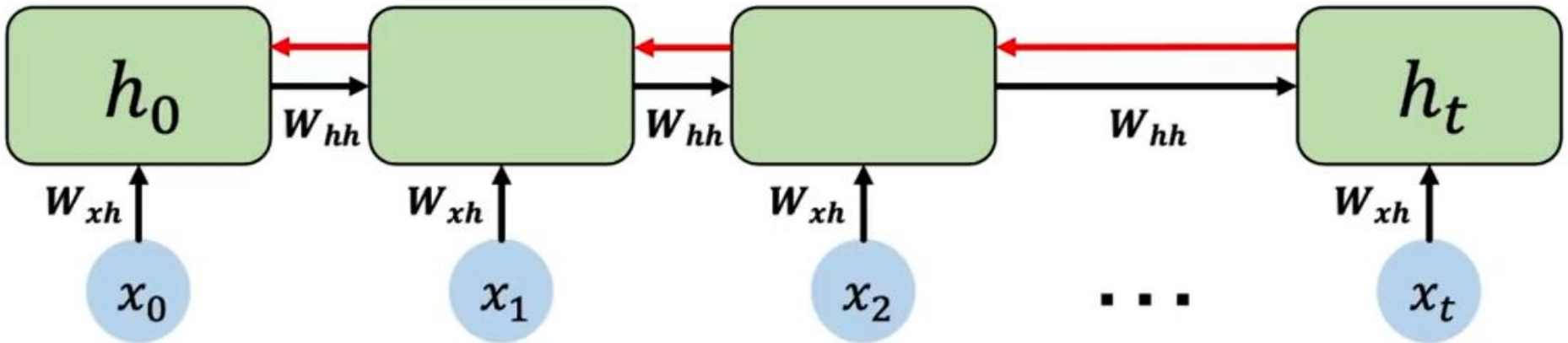


Standard RNN Gradient Flow



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

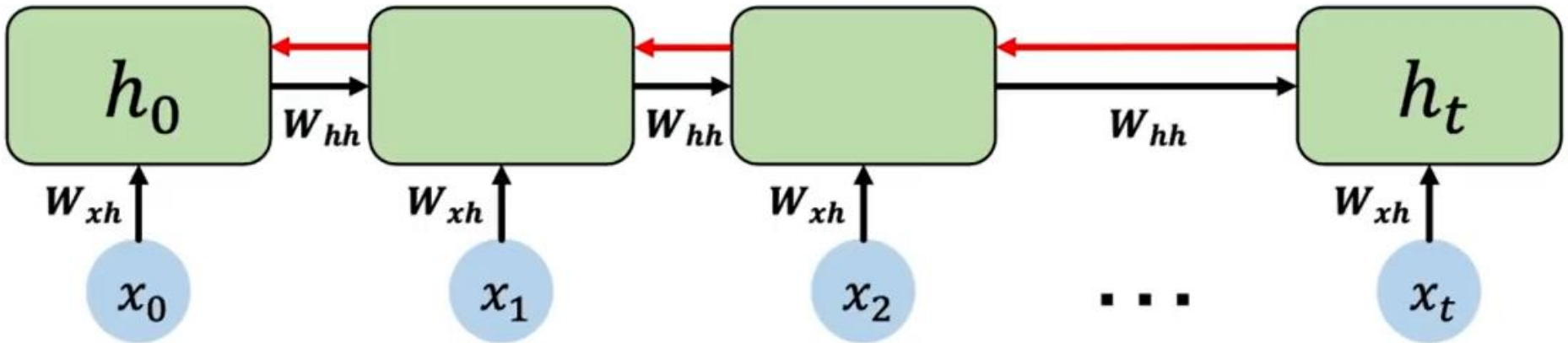
Standard RNN Gradient Flow: Exploding Gradients



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

Many values > 1:
exploding gradients

Standard RNN Gradient Flow: Exploding Gradients

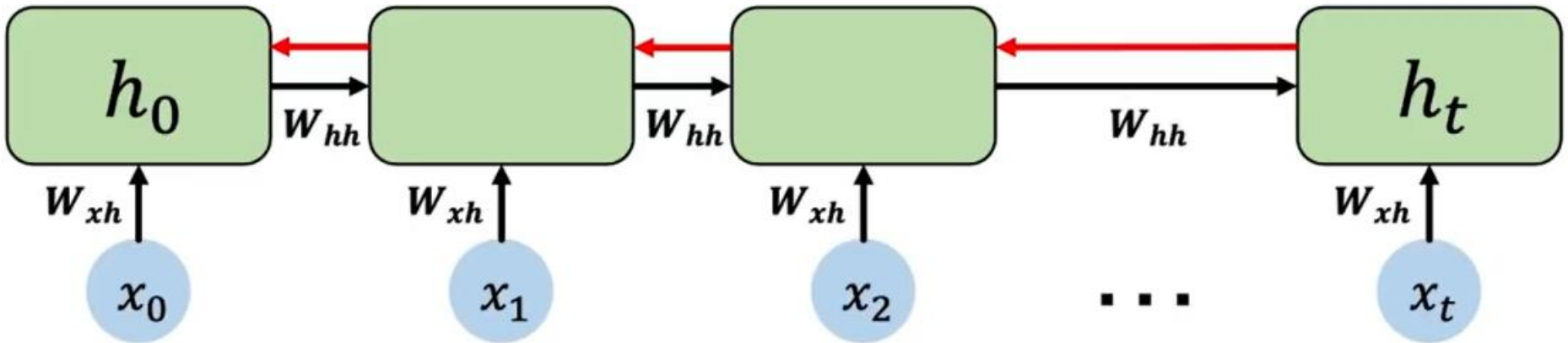


Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

Many values > 1:
exploding gradients

Gradient clipping to
scale big gradients

Standard RNN Gradient Flow: Vanishing Gradients



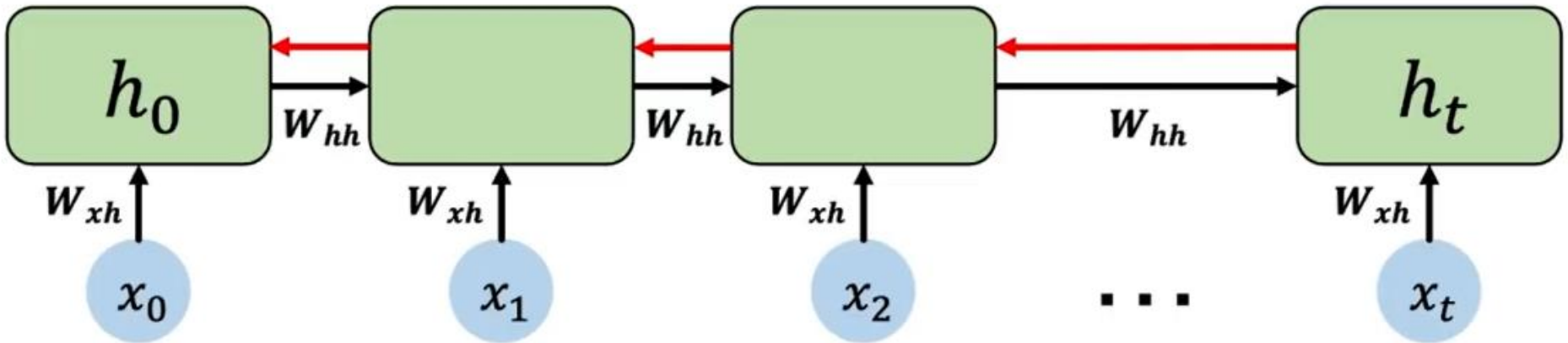
Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

Many values < 1:
vanishing gradients

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

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

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 Problem of Long-Term Dependencies

"The clouds are in the ___"

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 Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

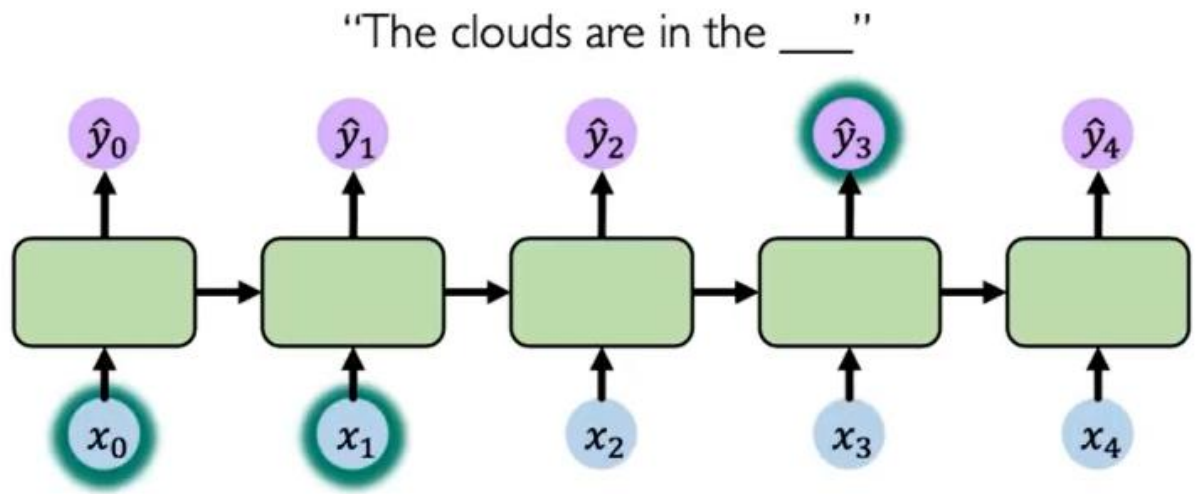
Multiply many **small numbers** together

↓

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

↓

Bias parameters to capture short-term dependencies



The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

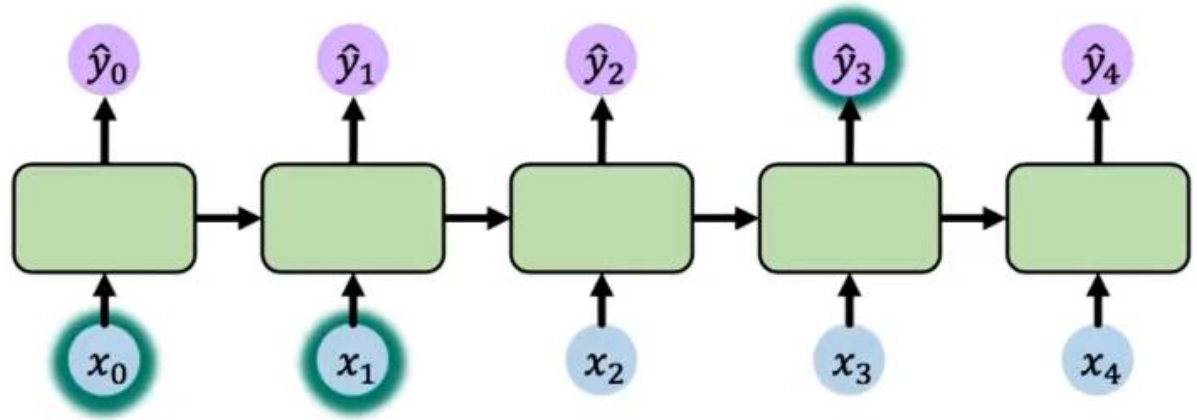
↓

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

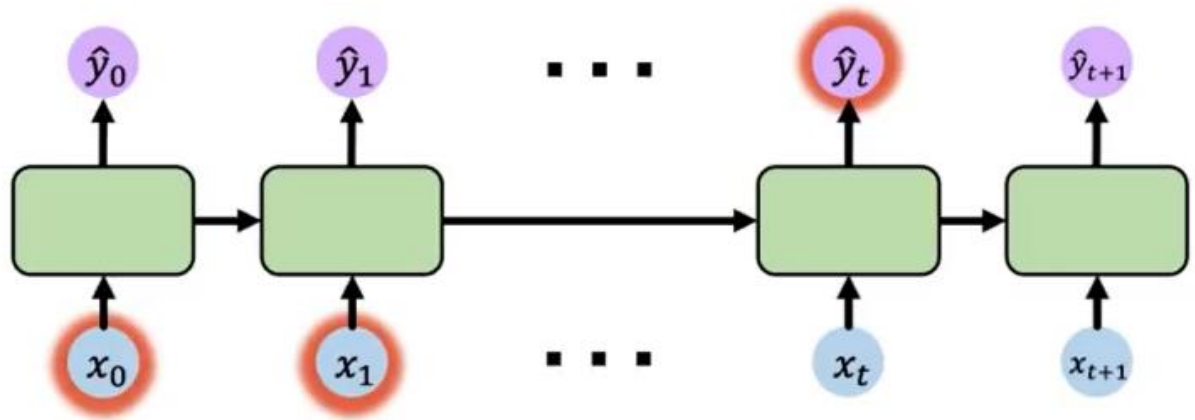
↓

Bias parameters to capture short-term dependencies

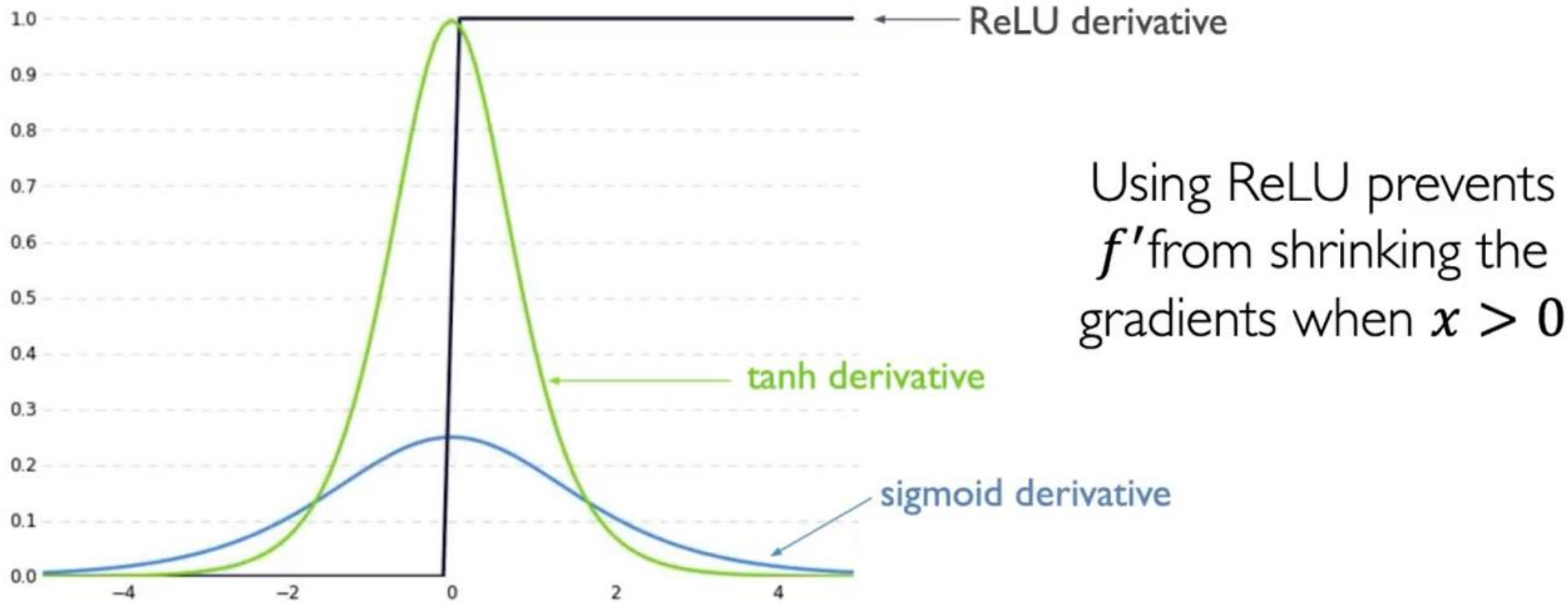
"The clouds are in the ___"



"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

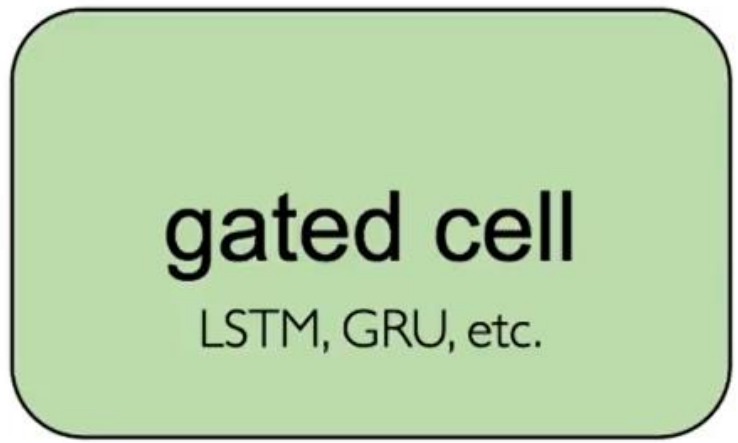
Initialize **biases** to zero

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

This helps prevent the weights from shrinking to zero.

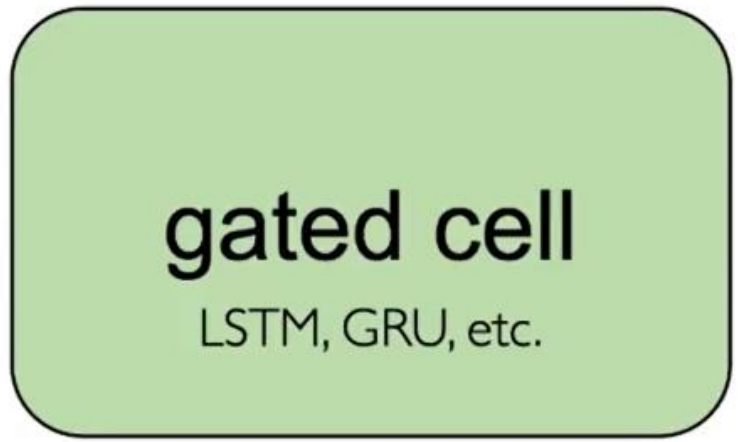
Solution #3: Gated Cells

Idea: use a more **complex recurrent unit with gates** to control what information is passed through



Solution #3: Gated Cells

Idea: use a more **complex recurrent unit with gates** to control what information is passed through

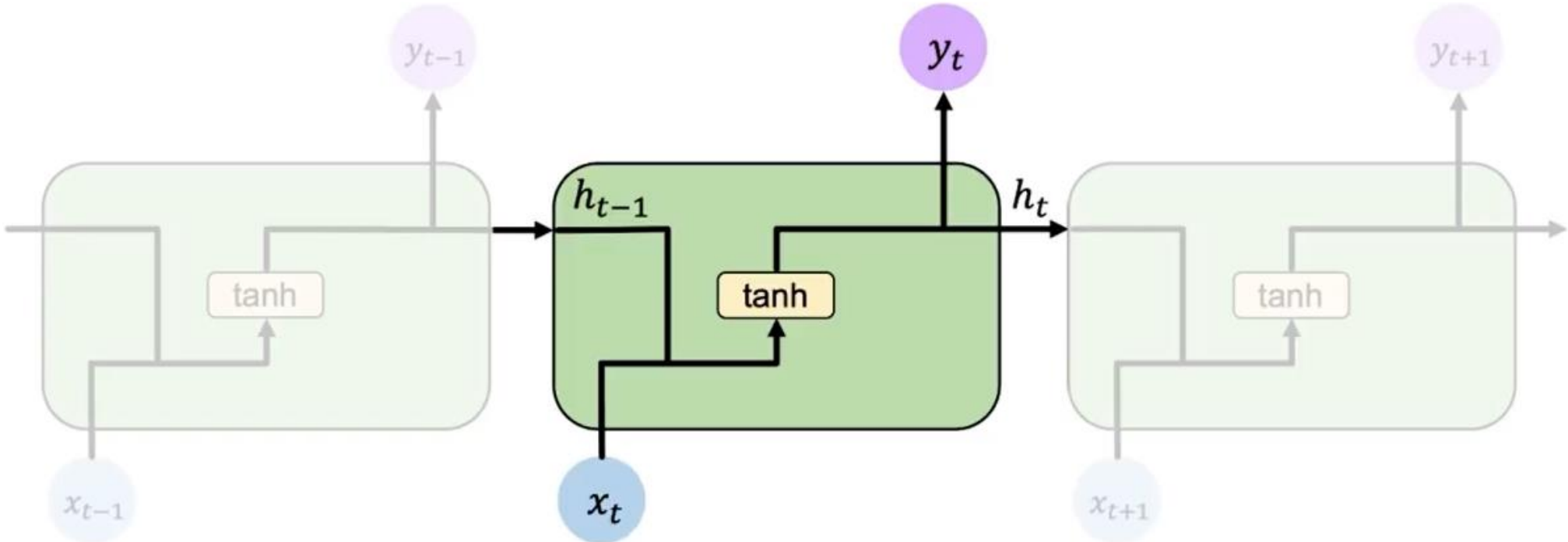


Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Long Short Term Memory (LSTM) Networks

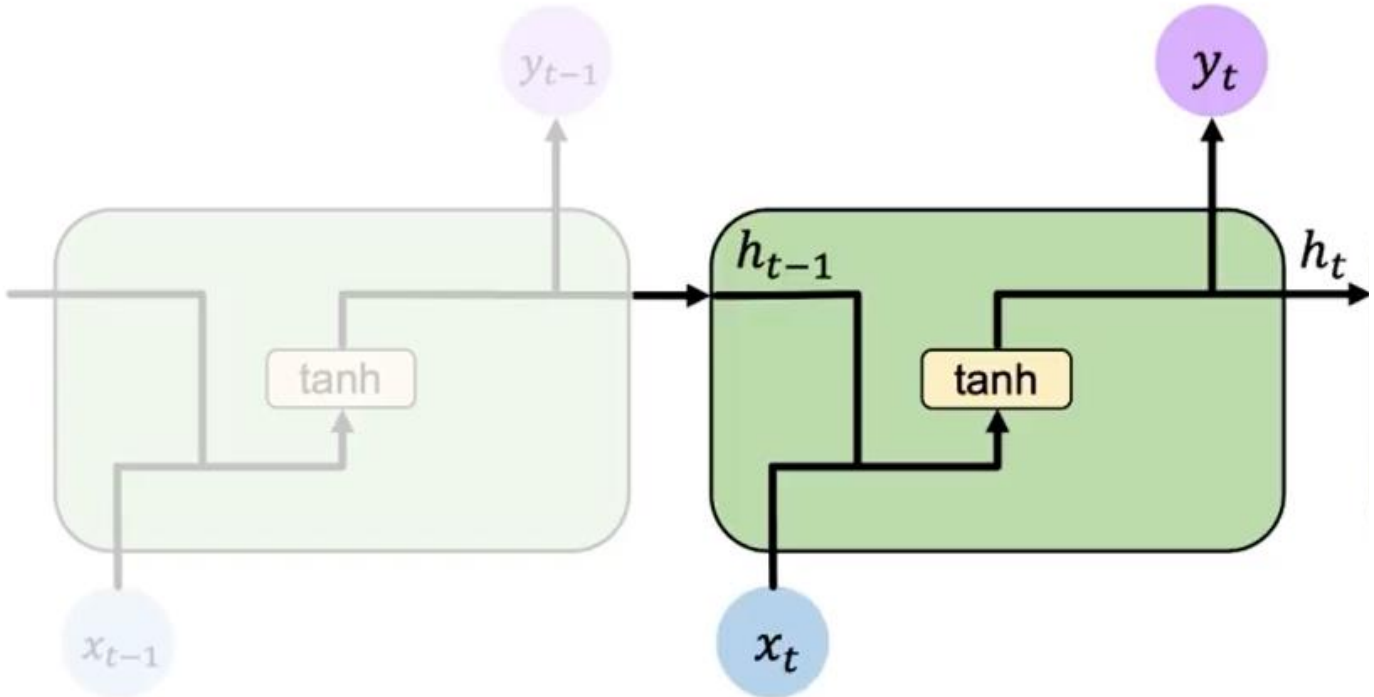
Standard RNN

In a standard RNN, repeating modules contain a **simple computation node**



Standard RNN

In a standard RNN, repeating modules contain a **simple computation node**



Output Vector

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

Update Hidden State

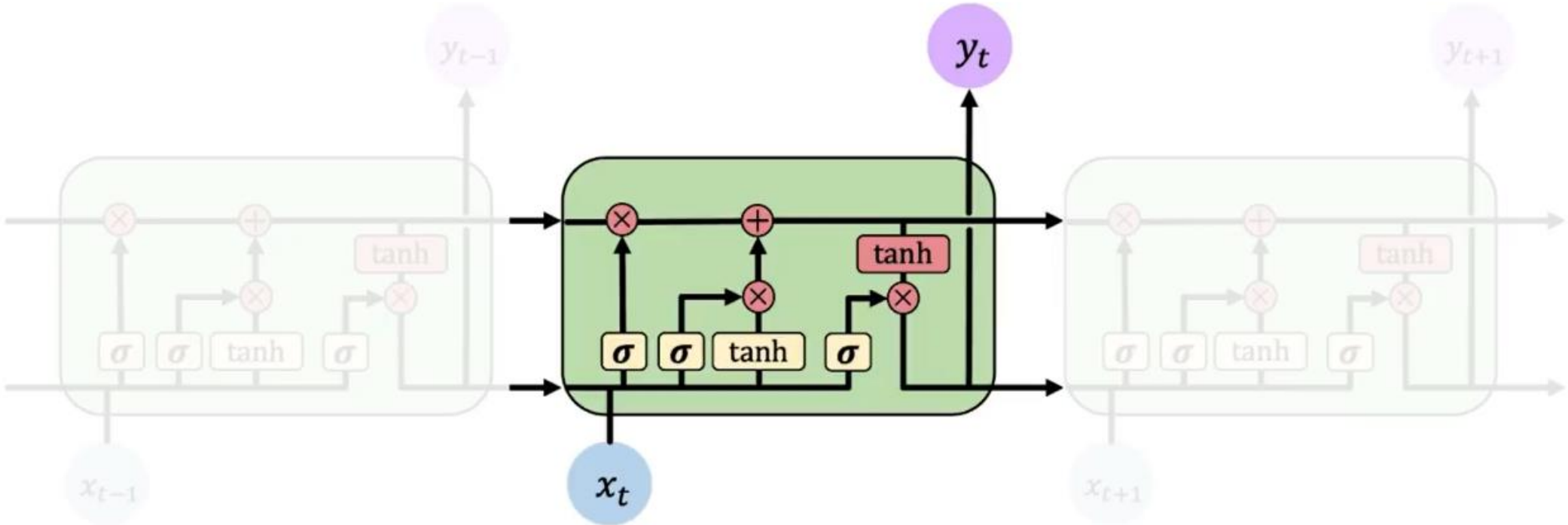
$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

Input Vector

$$x_t$$

Long Short Term Memory (LSTMs)

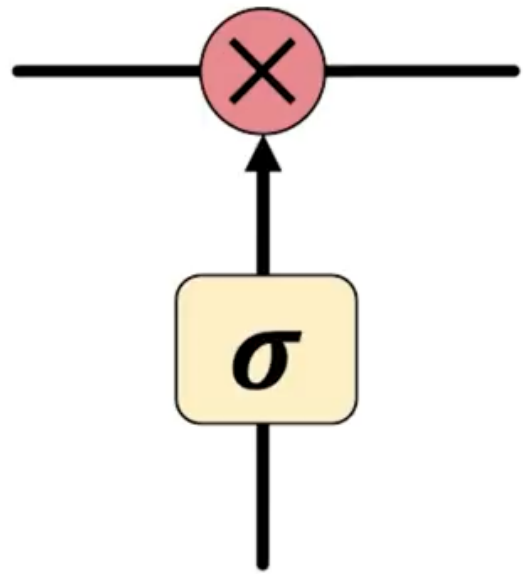
LSTM modules contain **computational blocks** that **control information flow**



LSTM cells are able to track information throughout many timesteps

Long Short Term Memory (LSTMs)

Information is **added** or **removed** through structures called **gates**

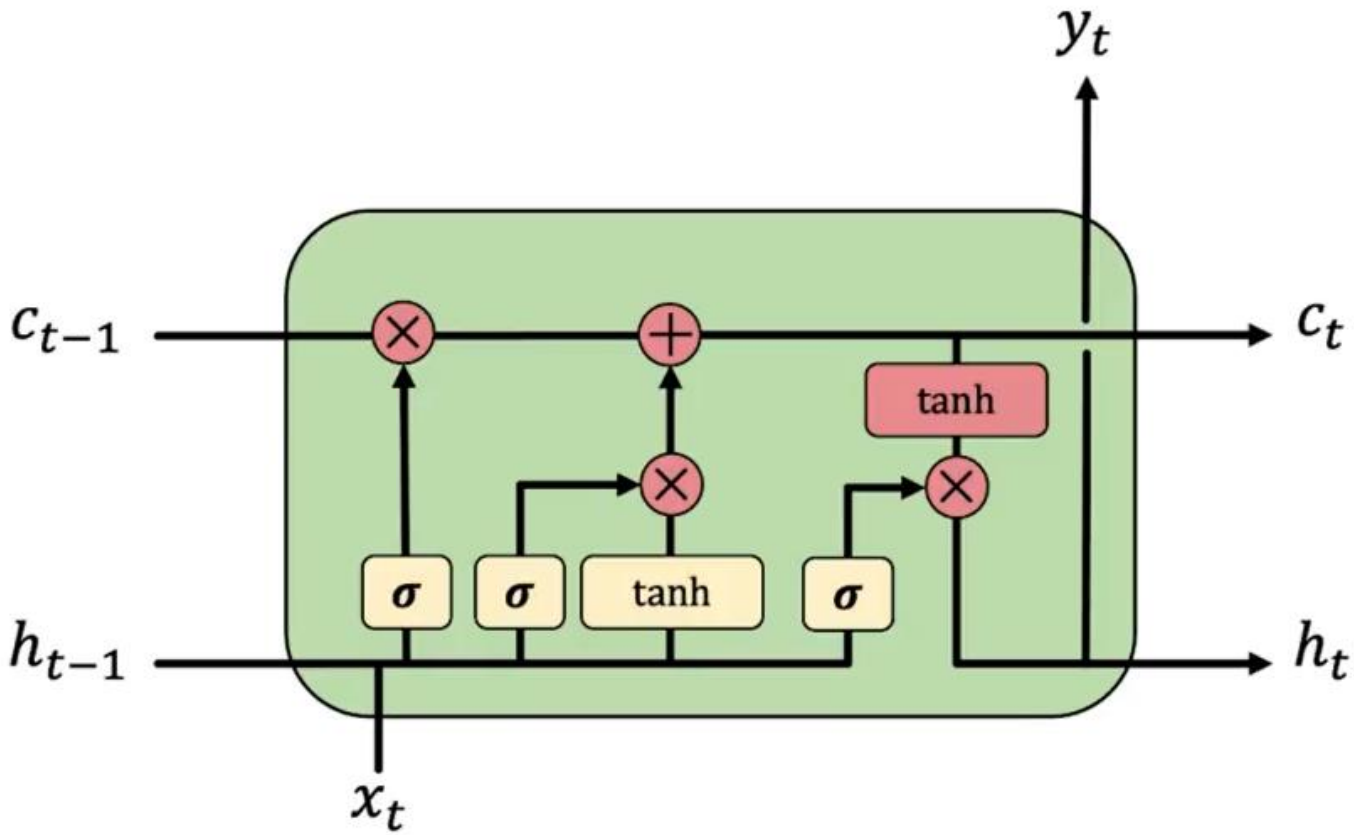


Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

Long Short Term Memory (LSTMs)

How do LSTMs work?

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

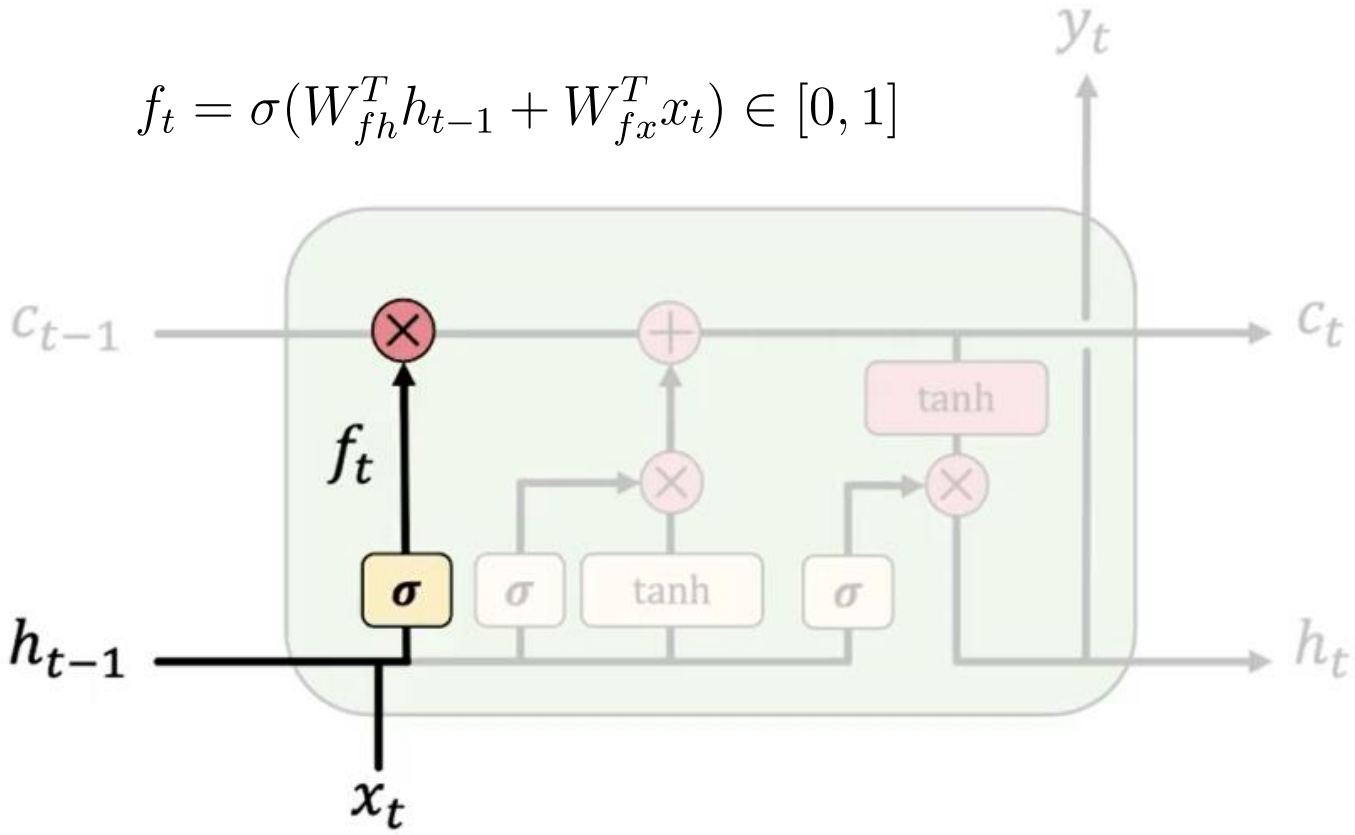


Long Short Term Memory (LSTMs)

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

LSTMs **forget irrelevant** parts of the previous state

$$f_t = \sigma(W_{fh}^T h_{t-1} + W_{fx}^T x_t) \in [0, 1]$$



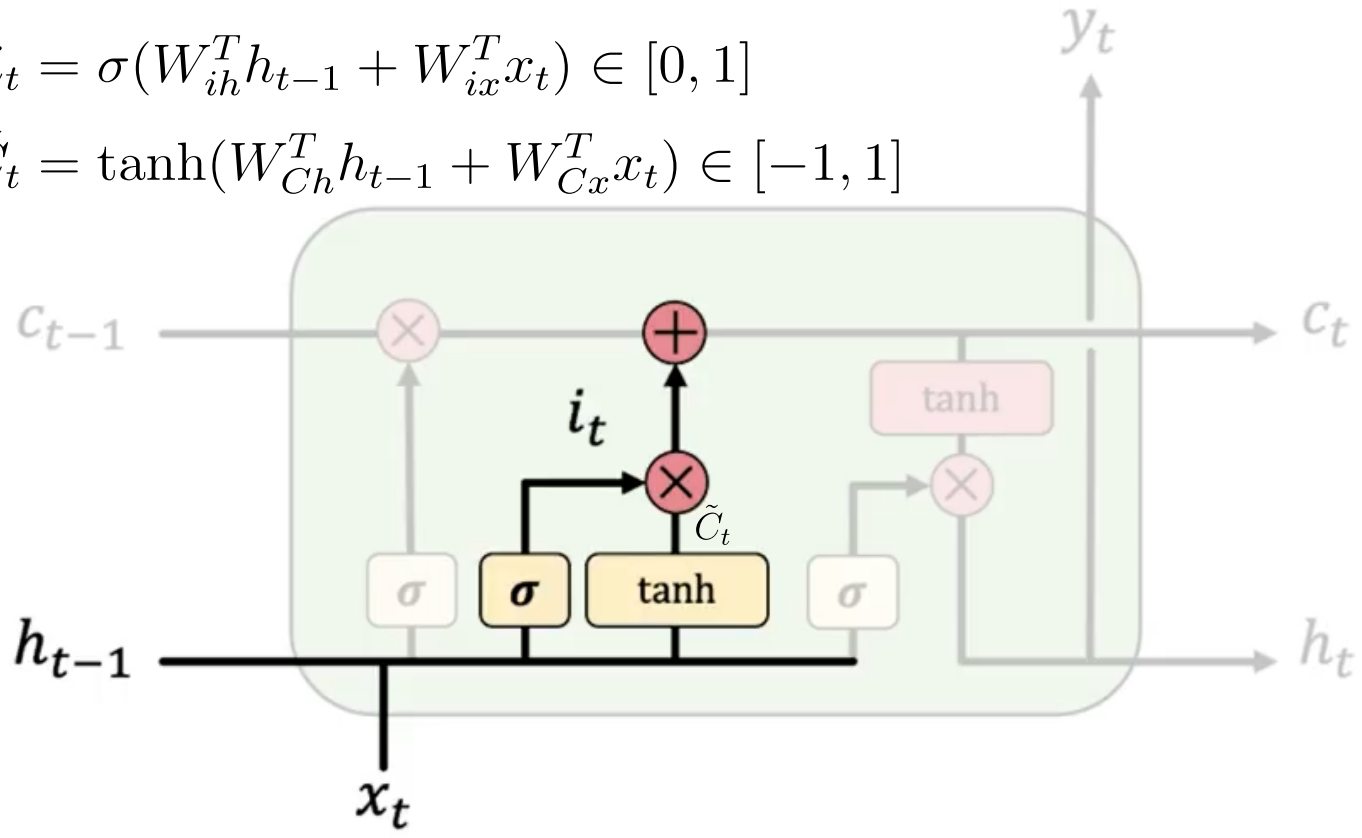
Long Short Term Memory (LSTMs)

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

LSTMs **store relevant** new information into the cell state

$$i_t = \sigma(W_{ih}^T h_{t-1} + W_{ix}^T x_t) \in [0, 1]$$

$$\tilde{C}_t = \tanh(W_{Ch}^T h_{t-1} + W_{Cx}^T x_t) \in [-1, 1]$$

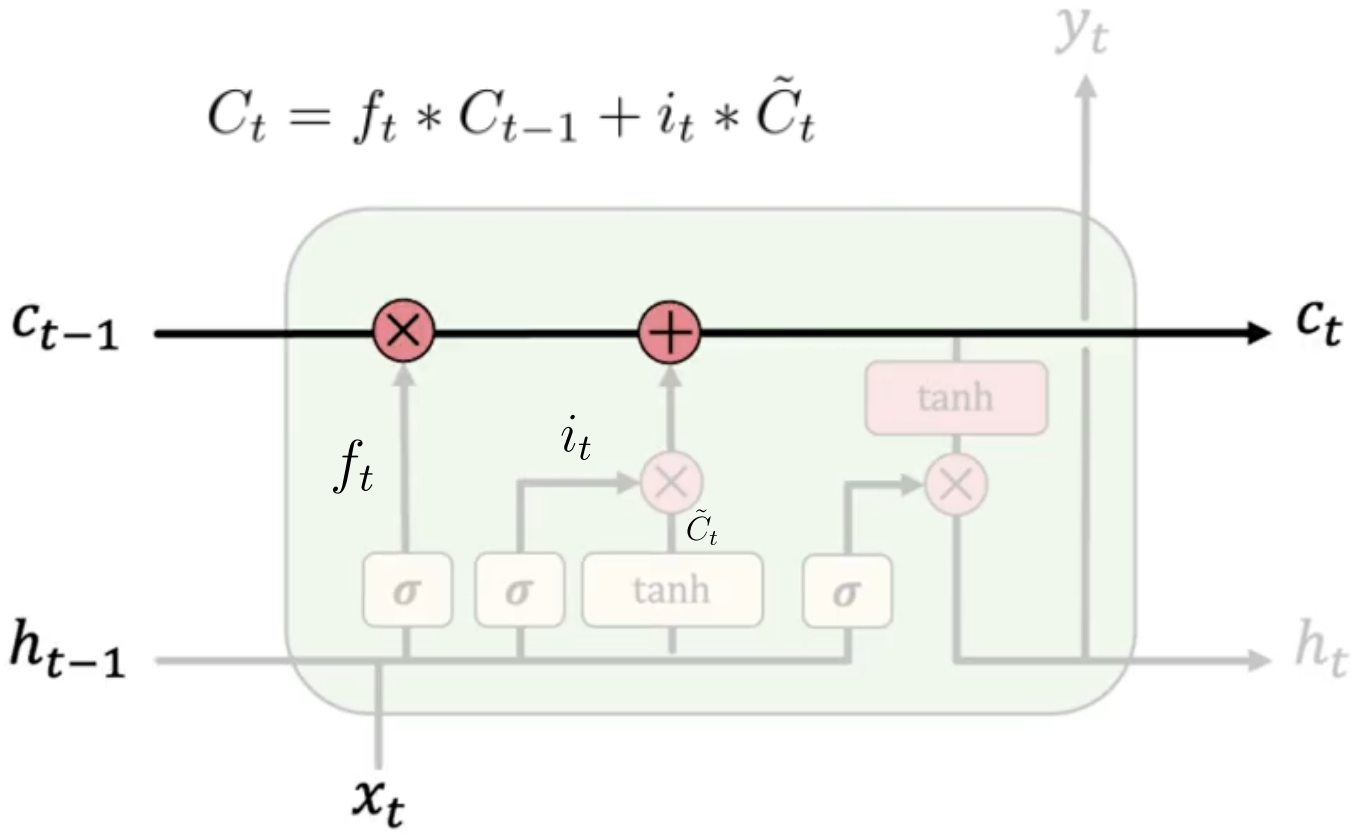


Long Short Term Memory (LSTMs)

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

LSTMs **selectively update** cell state values

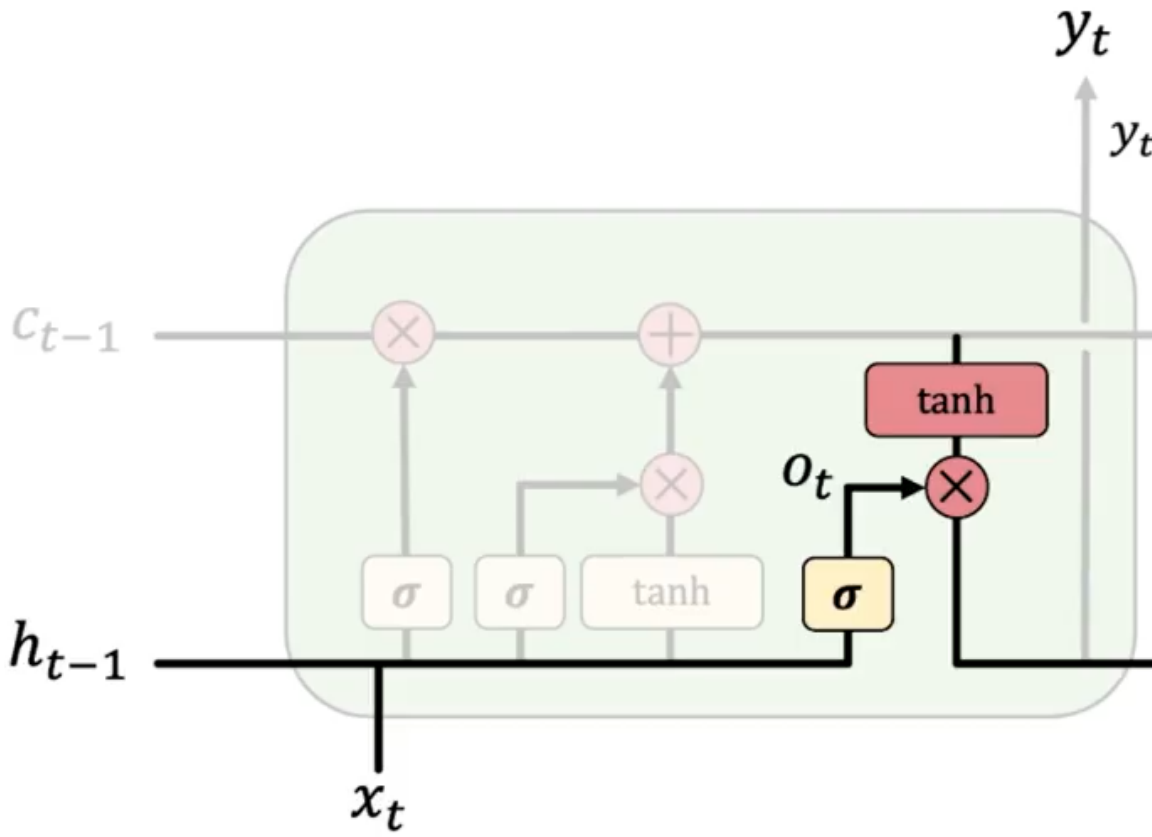
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



Long Short Term Memory (LSTMs)

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

The **output gate** controls what information is sent to the next time step

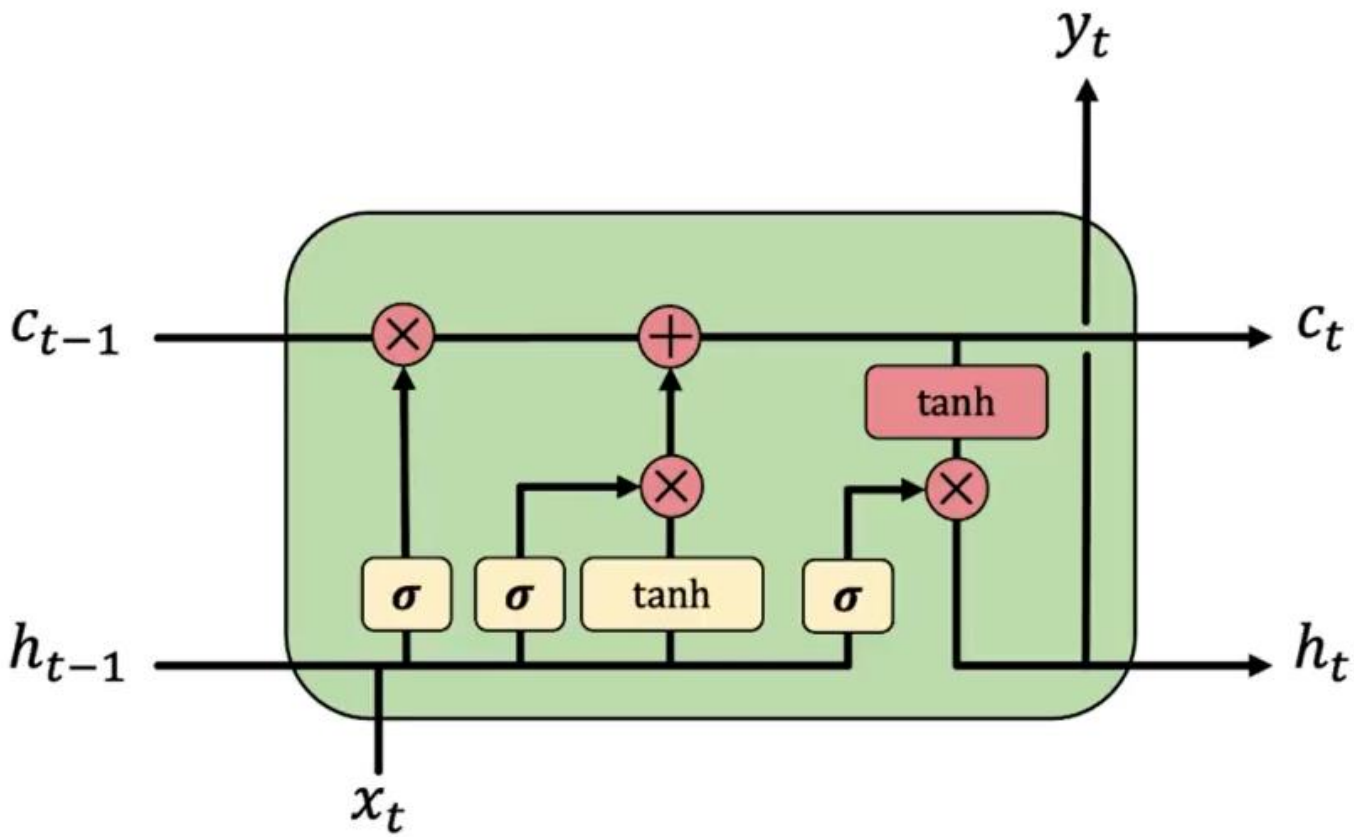


$$o_t = \sigma(W_{oh}^T h_{t-1} + W_{ox}^T x_t) \in [0, 1]$$

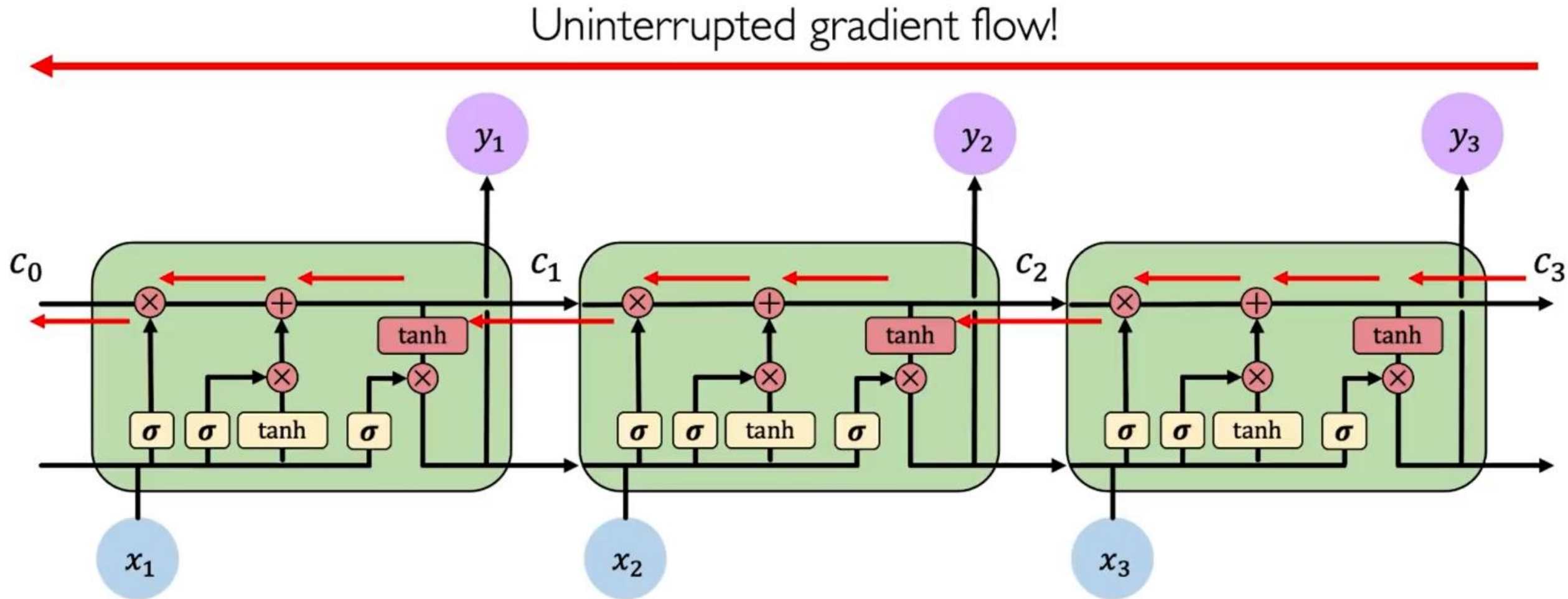
$$h_t = o_t * \tanh(C_t)$$

Long Short Term Memory (LSTMs)

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



LSTM Gradient Flow

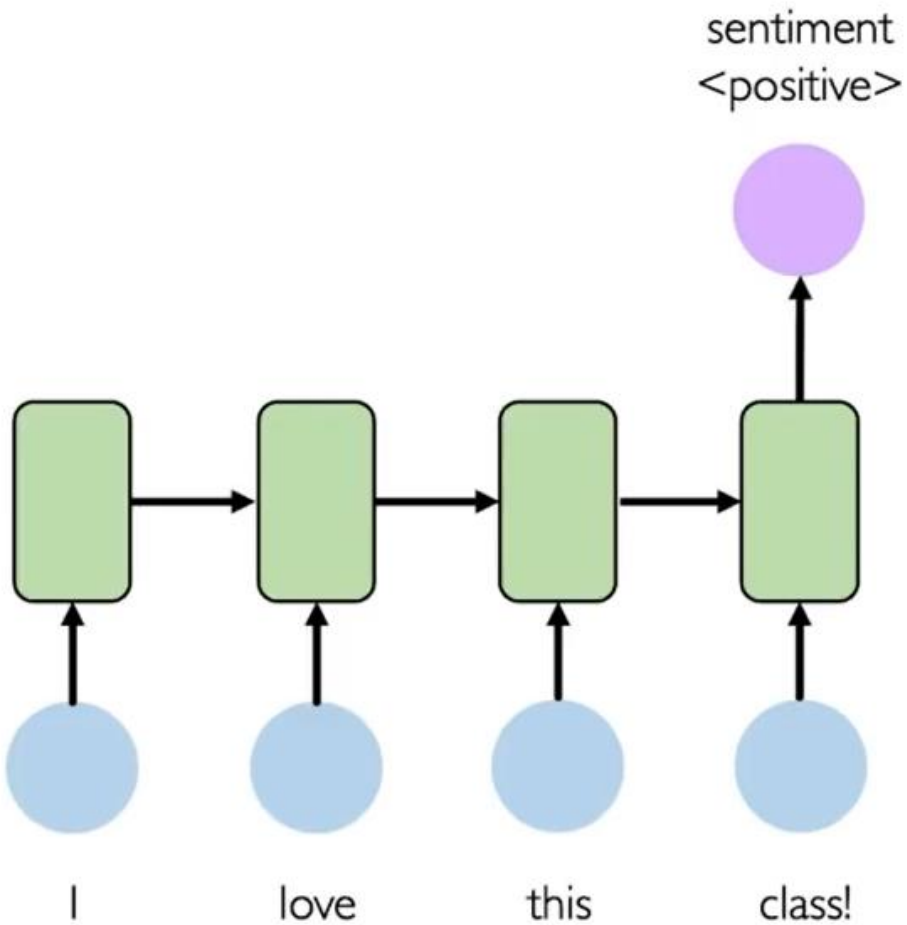


LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
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 **uninterrupted gradient flow**

RNN Applications

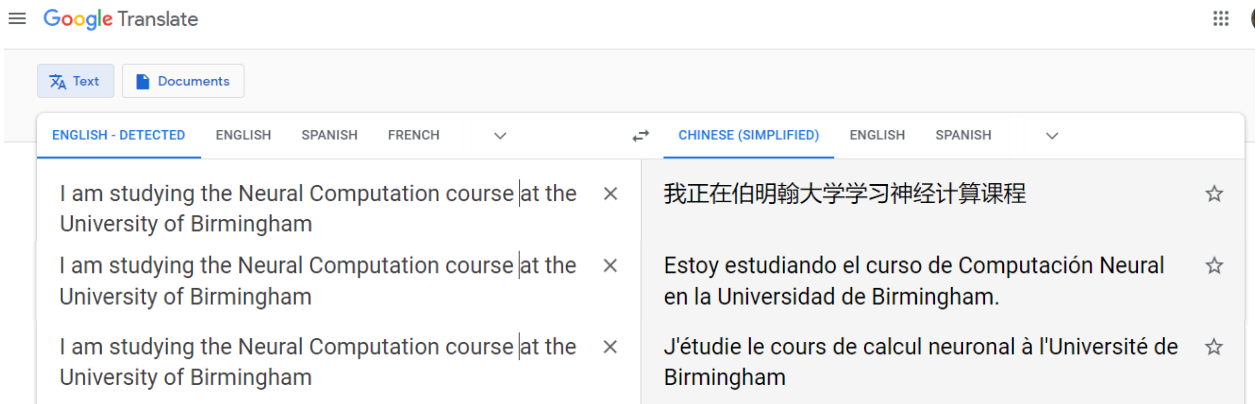
Example Task: Sentiment Classification

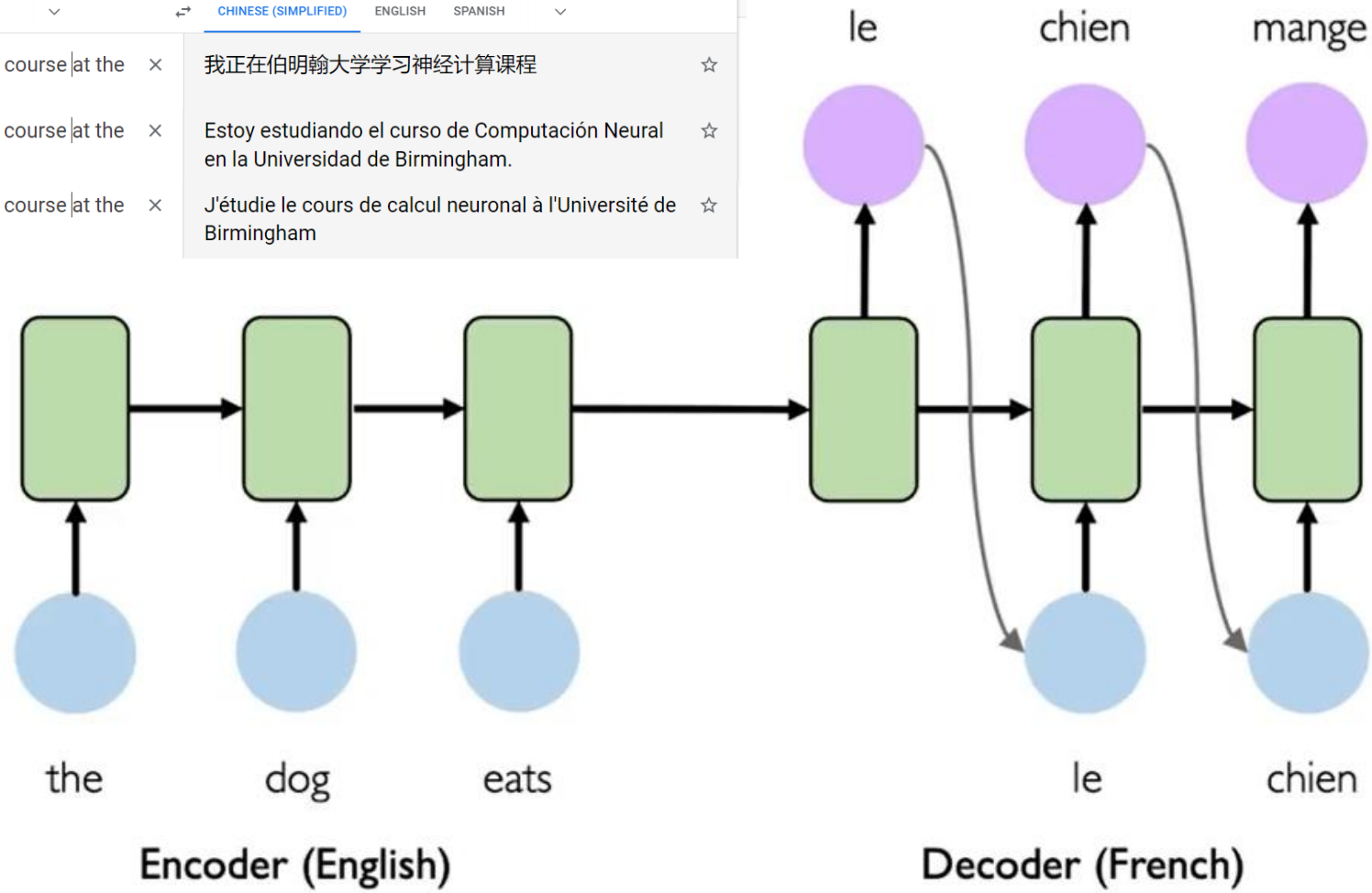


Input: sequence of words

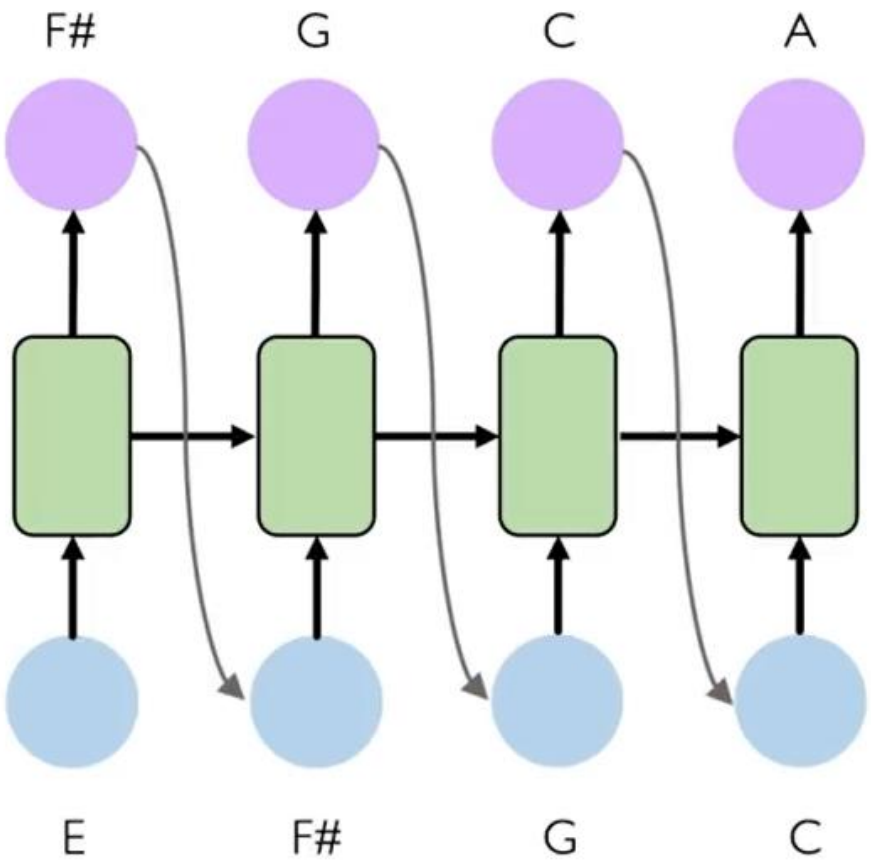
Output: probability of having positive sentiment

Example Task: Machine Translation



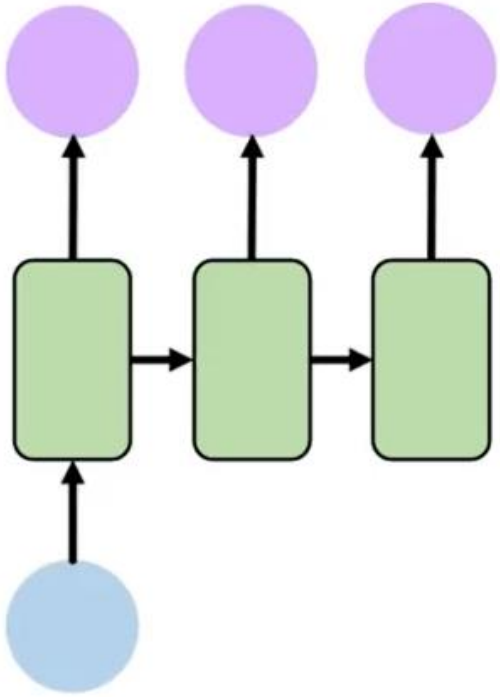


Example Task: Music Generation



Chopin Music Generation

Example Task: Image Captioning



One to Many
Text Generation
Image Captioning

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. Gated cells like **LSTMs** let us model **long-term dependencies**

References

- http://introtodeeplearning.com/
- http://cs231n.stanford.edu/schedule.html
- https://www.youtube.com/watch?v=j60J1cGINX4&t=25s
- https://www.youtube.com/watch?v=40riCqvRoMs&t=10s
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/