Day 6 - From Recurrent Nets to Transformers

Advanced Text as Data: Natural Language Processing
Essex Summer School in Social Science Data Analysis

Burt L. Monroe (Instructor) & Sam Bestvater (TA)
Pennsylvania State University

August 3, 2021

Today

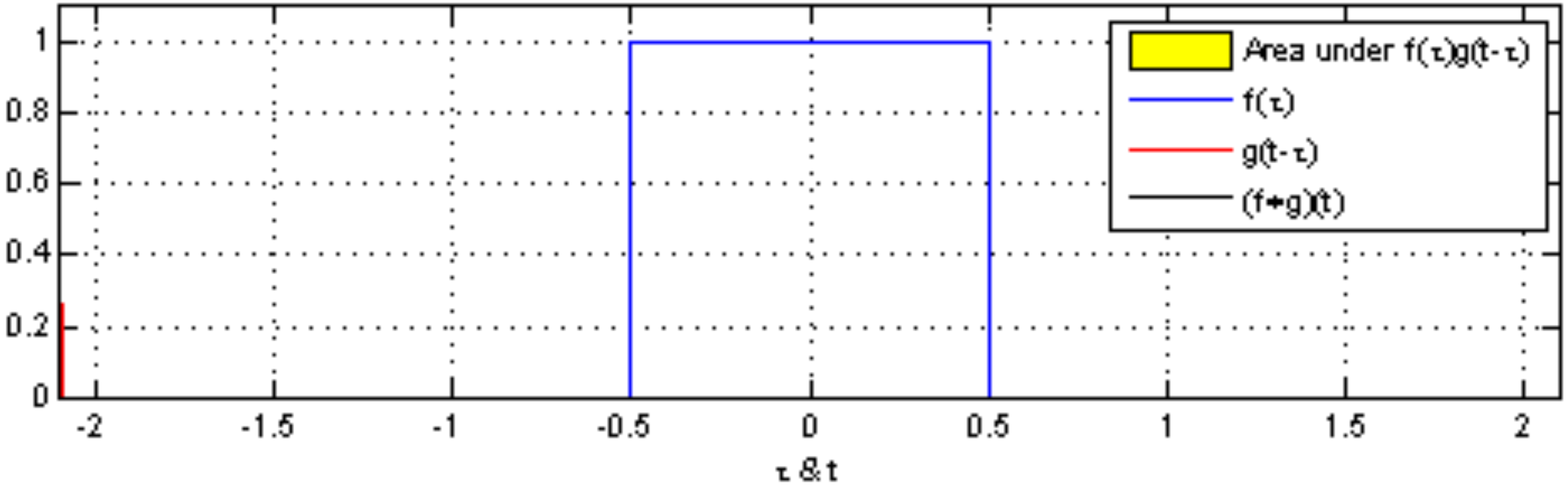
- Convolutional Neural Nets (CNNs) - Convolution, filters/kernels, higher-level features
- Recurrent Neural Nets (RNNs) - Recurrence / sequence, encoder-decoder seq2seq
- Gating in recurrent networks (LSTMs / bi-LSTMs)
- Attention mechanism in seq2seq models
- Self-attention & positional encodings (transformer)

Today (unlikely)

- Convolutional Neural Nets (CNNs) - Convolution, filters/kernels, higher-level features
- Recurrent Neural Nets (RNNs) - Recurrence / sequence, encoder-decoder seq2seq
- Gating in recurrent networks (LSTMs / bi-LSTMs)
- Attention mechanism in seq2seq models
- Self-attention & positional encodings (transformer)

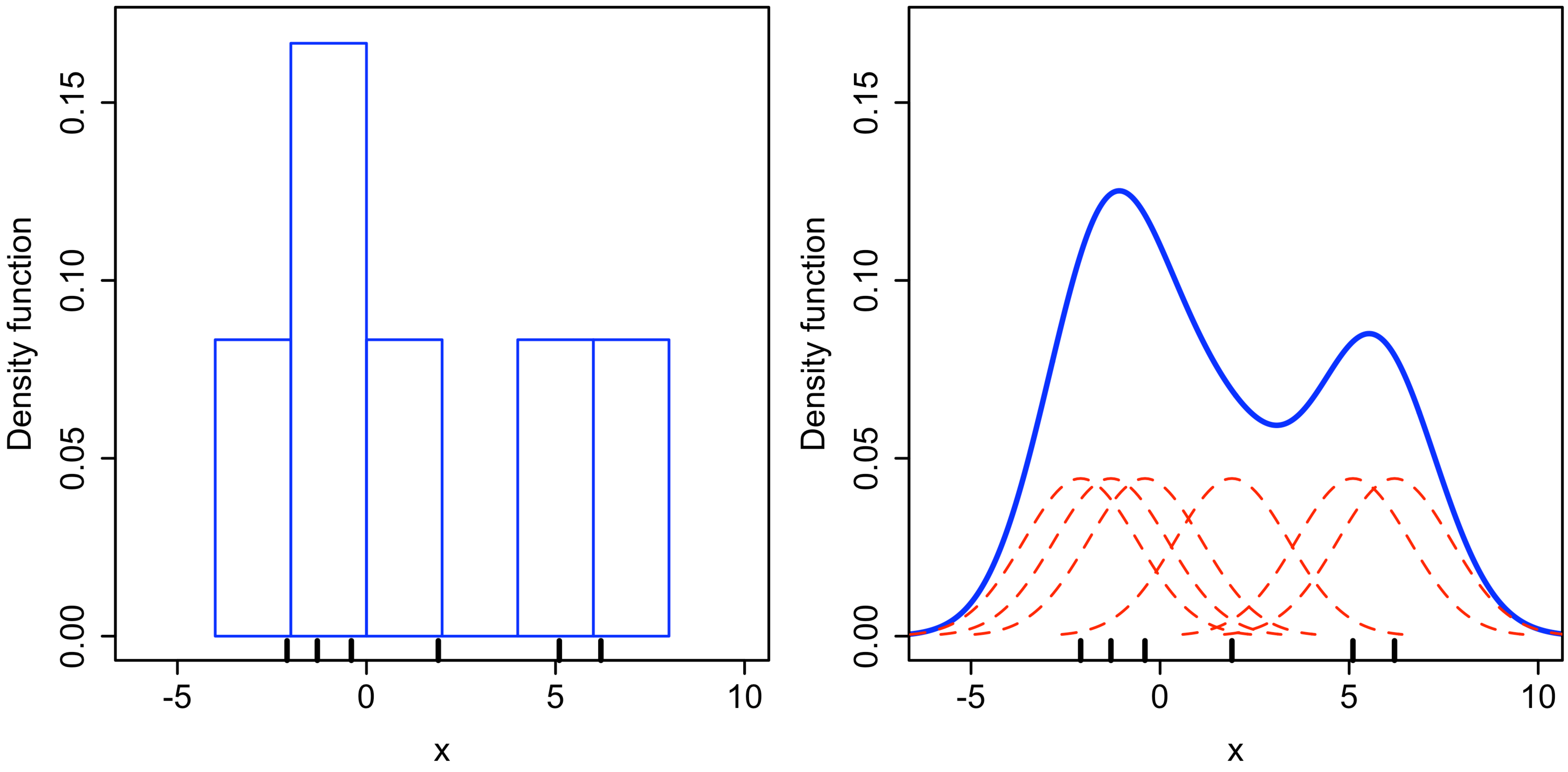
Convolution, Convolutional Neural Nets, and CNNs in NLP

Convolution



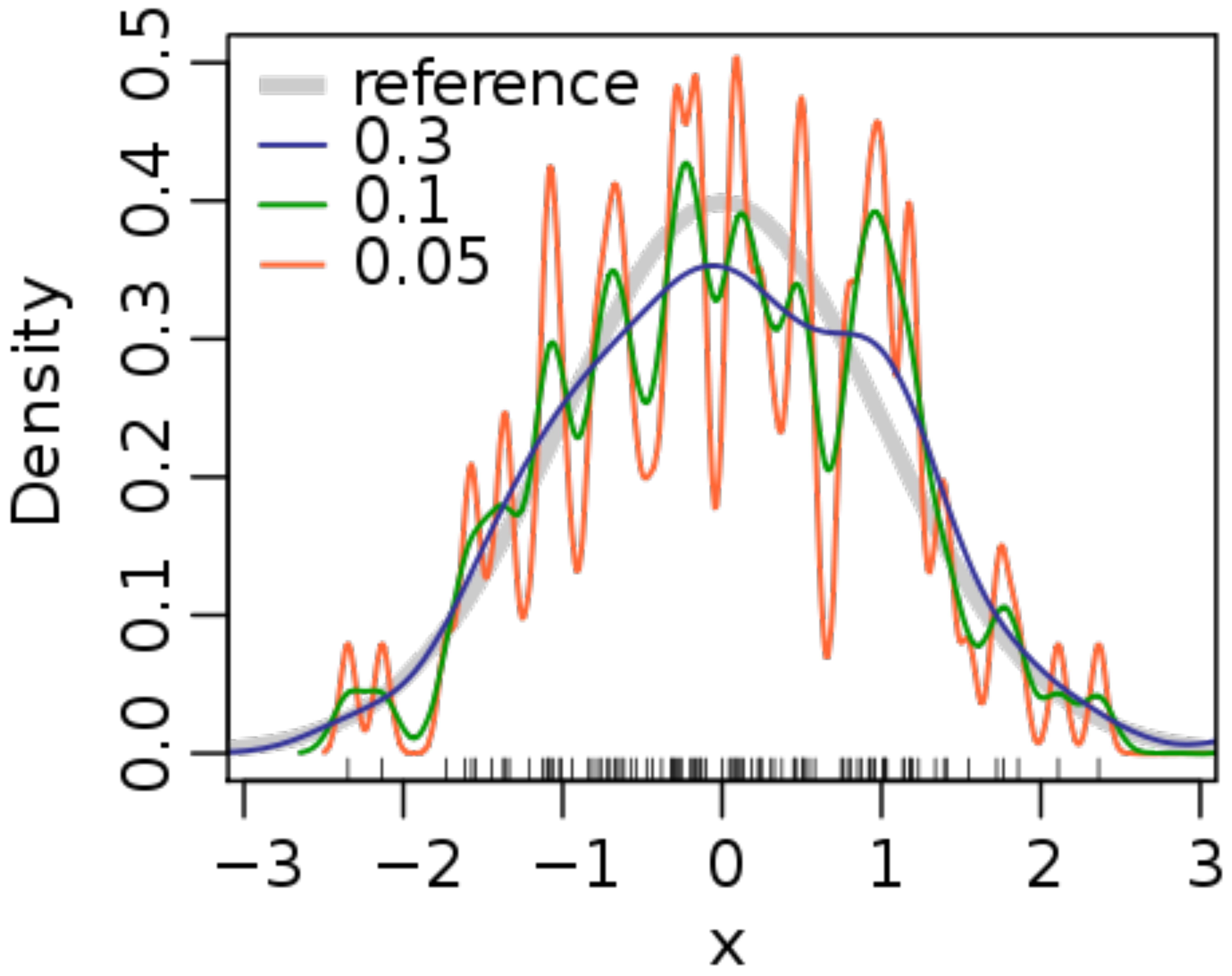
Source: Wikipedia, “Convolution”

Kernel density — smooth histogram by convolving a Gaussian over observations



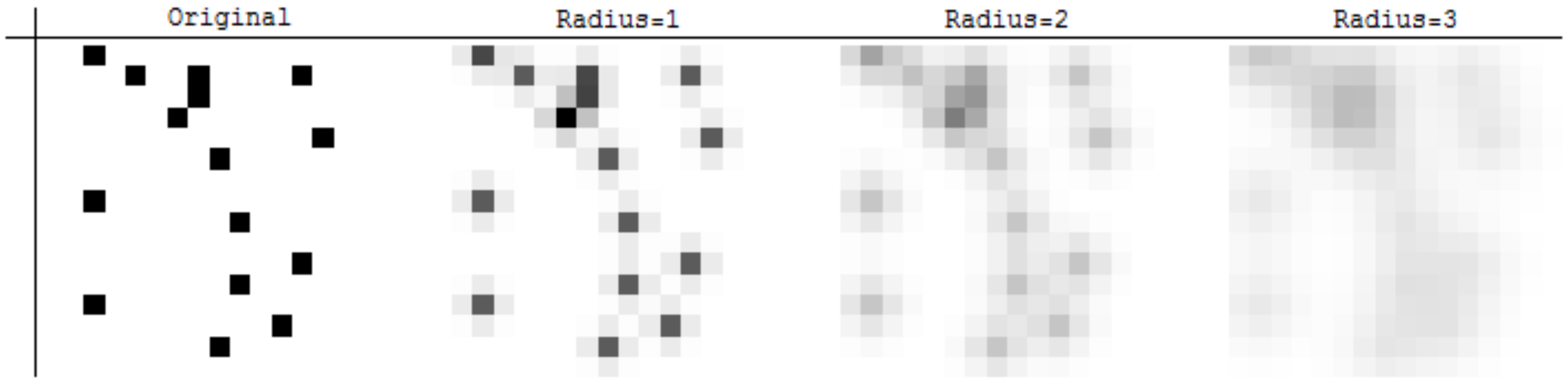
Source: Wikipedia, "Kernel Density"

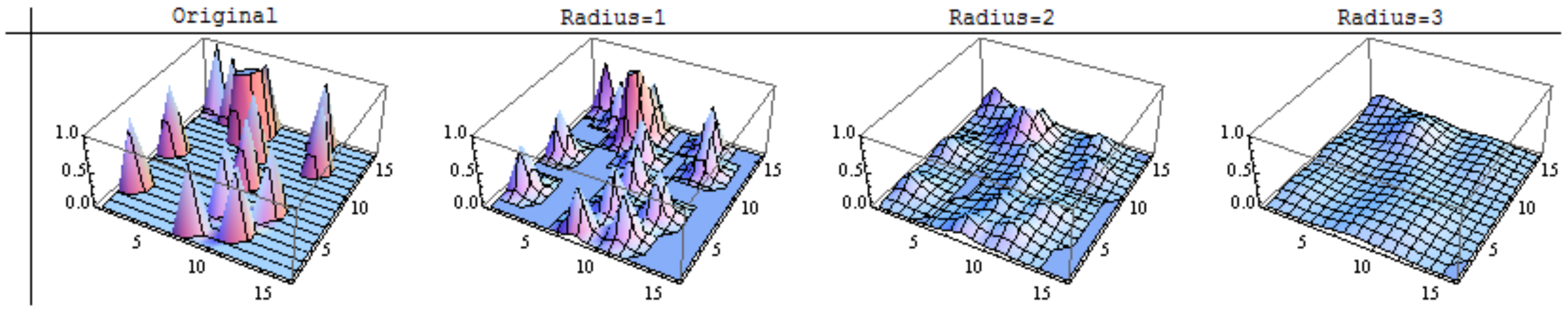
Kernel density — how smooth depends on variance / “width” of the Gaussian



Source: Wikipedia, “Kernel Density”

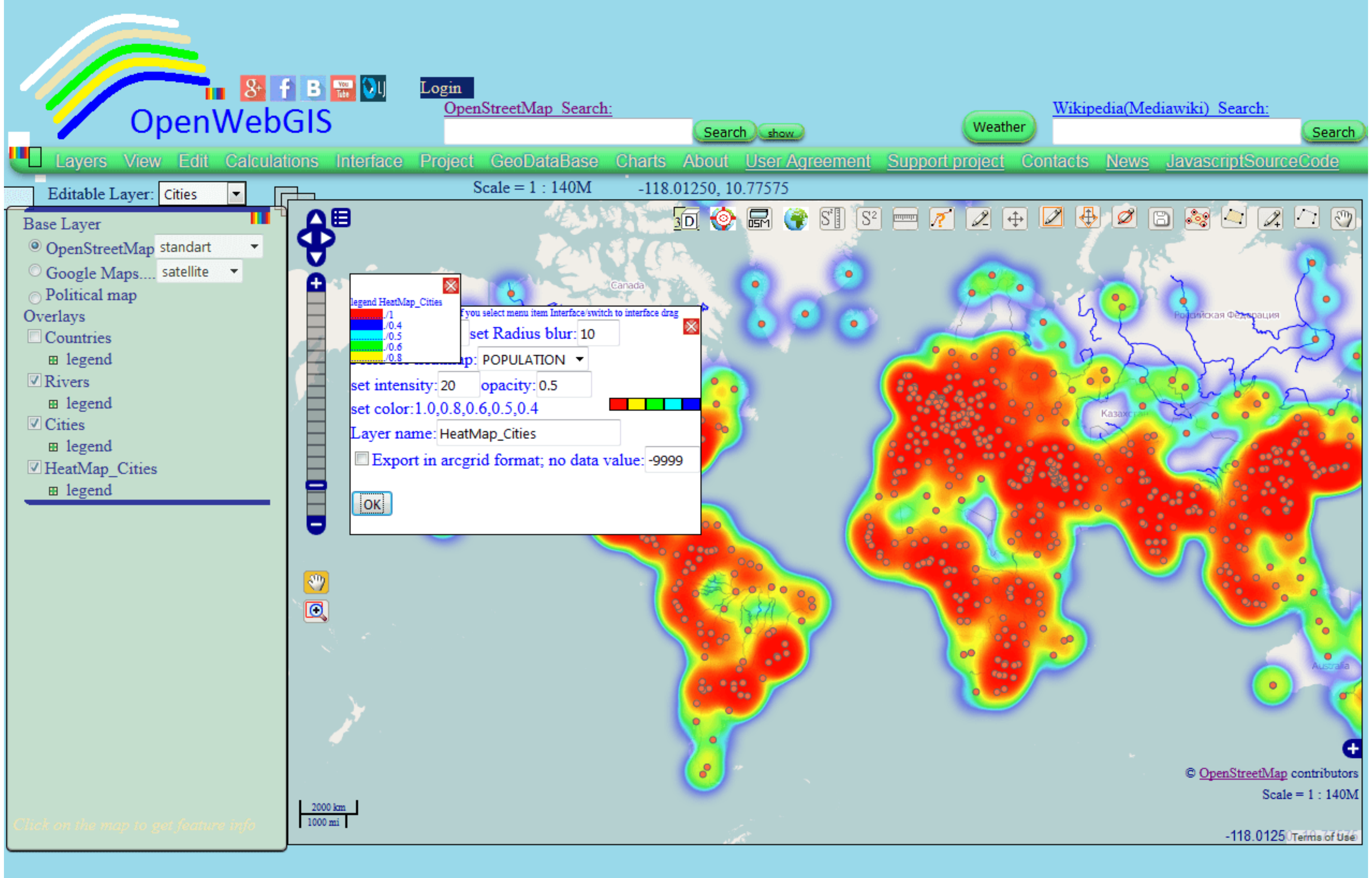
Kernel density — smooth in two dimensions





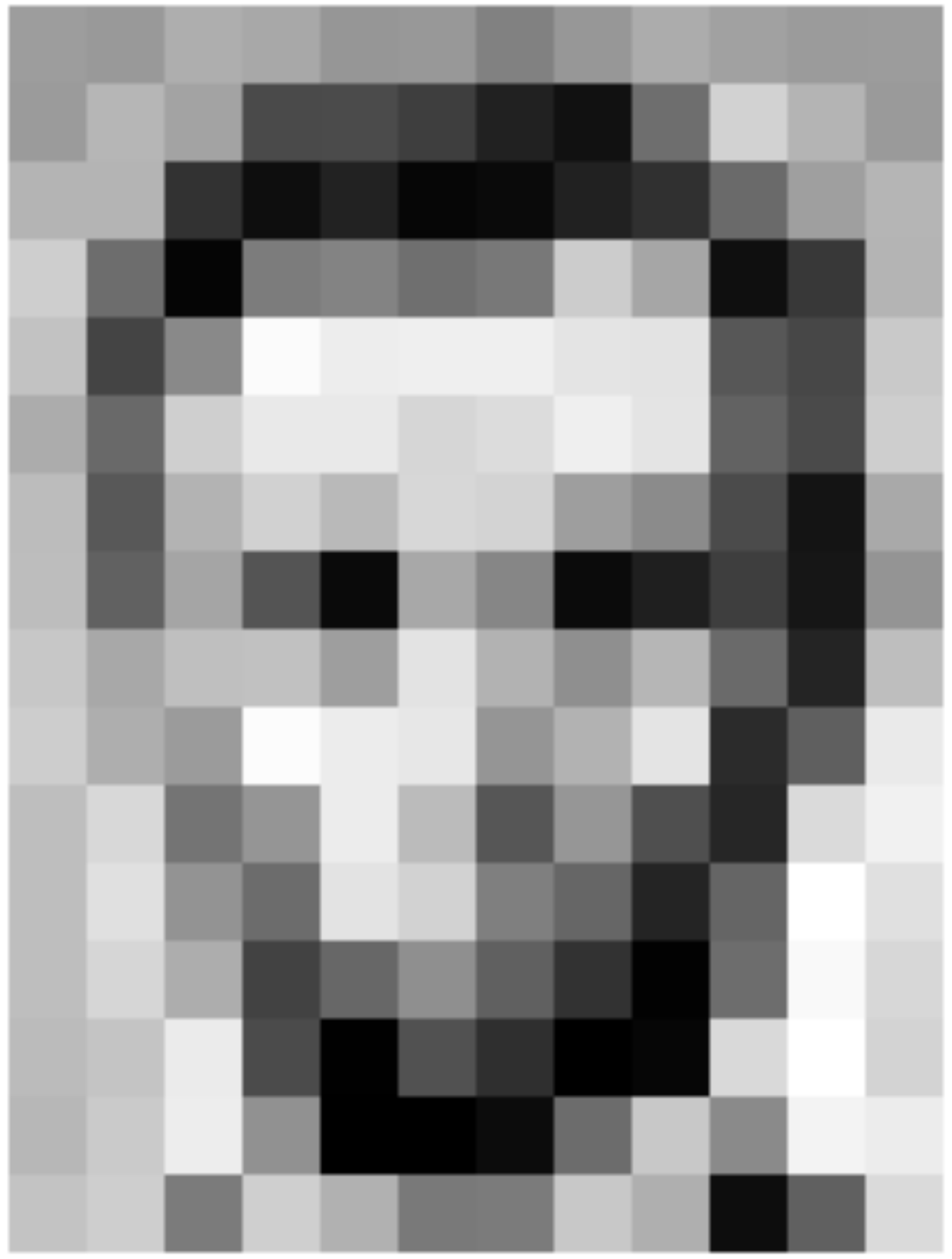
Source: Wikipedia, "Kernel Density"

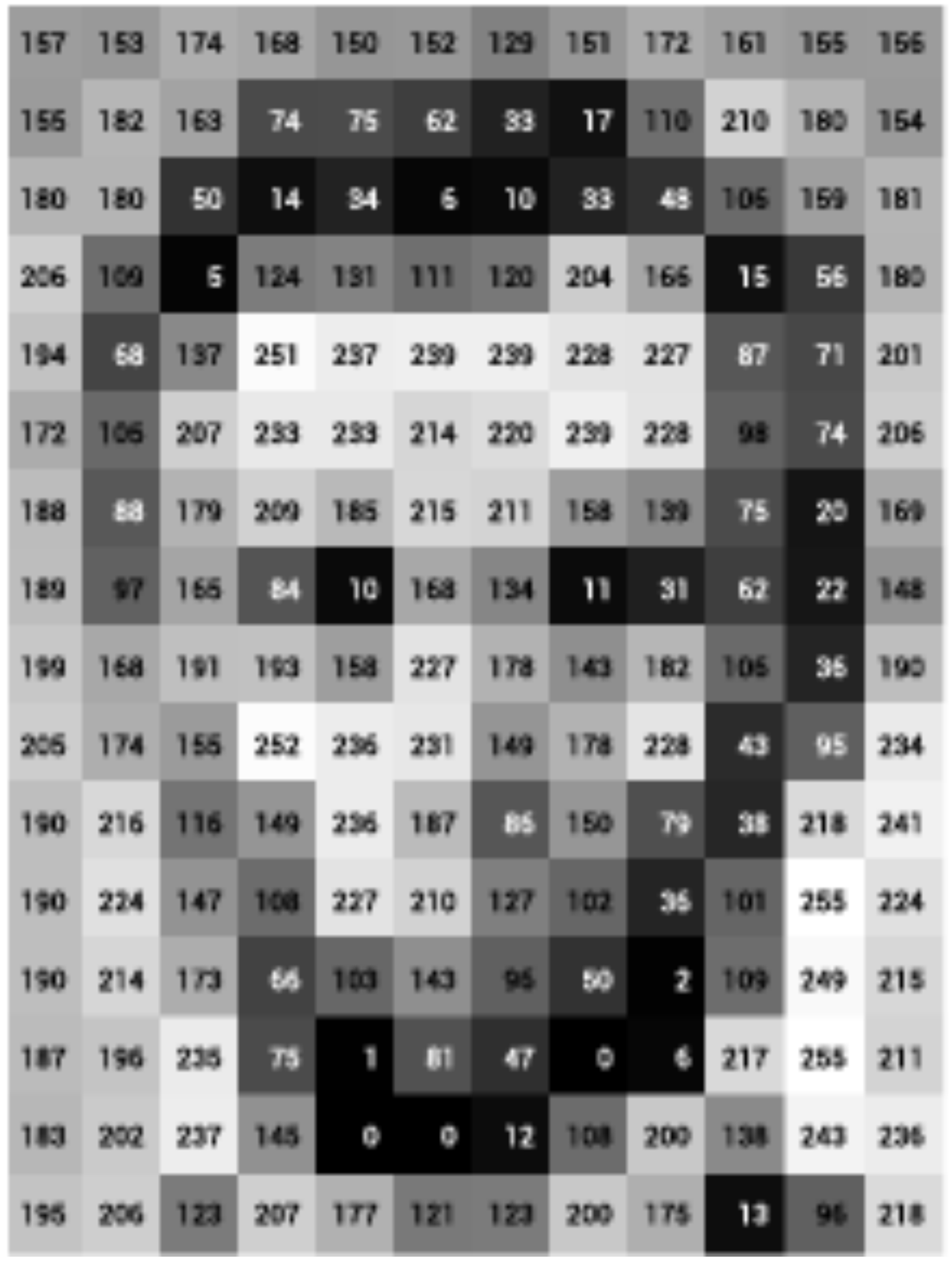
Kernel density — this is familiar in “heatmaps”



Source: Fedor Kolomeyko, www.digital-geography.com

Now imagine an image as two-dimensional data — a grid of pixel intensities

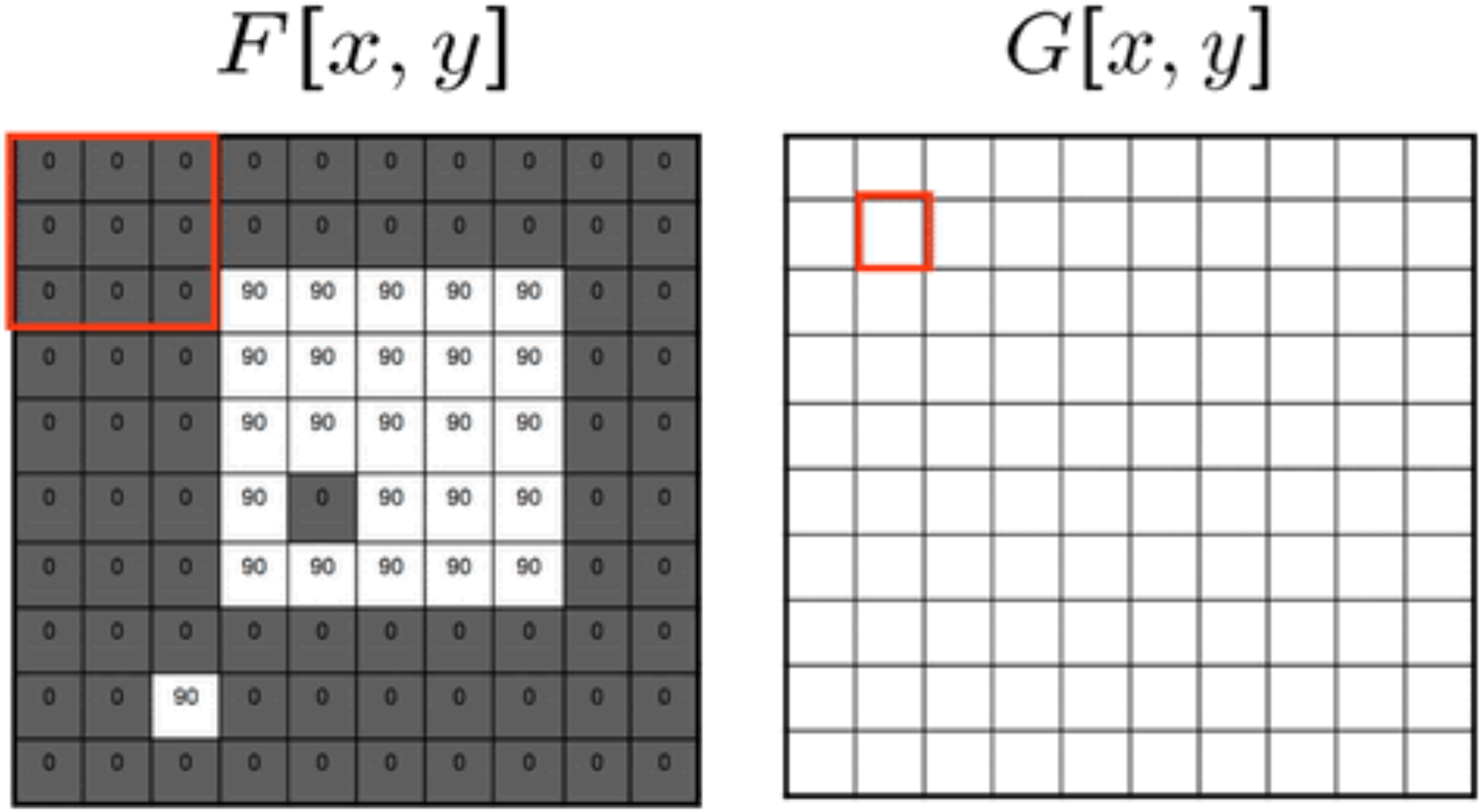




157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139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

Source: Stanford Artificial Intelligence Laboratory, *Introduction to Computer Vision*, “Image Filtering”

An image *filter* is a kernel - a small window we *convolve* over an image.
The filter illustrated here averages the nine pixels in the window.

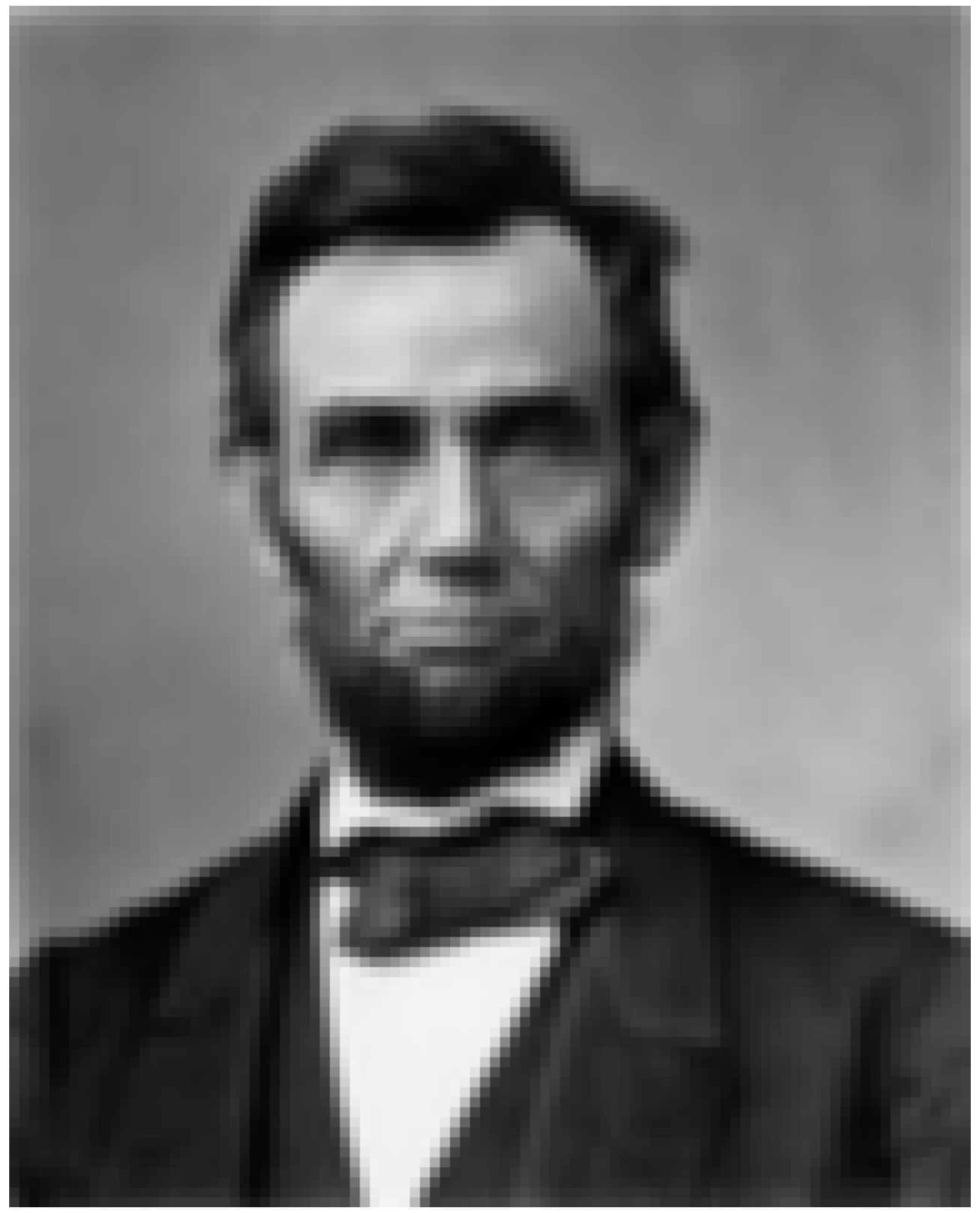


Source: Stanford Artificial Intelligence Laboratory, *Introduction to Computer Vision*, “Image Filtering”

A ~Gaussian kernel (high in the middle, lower away from the middle) acts as a smoothing or "blur filter"

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

(a) Blur kernel.

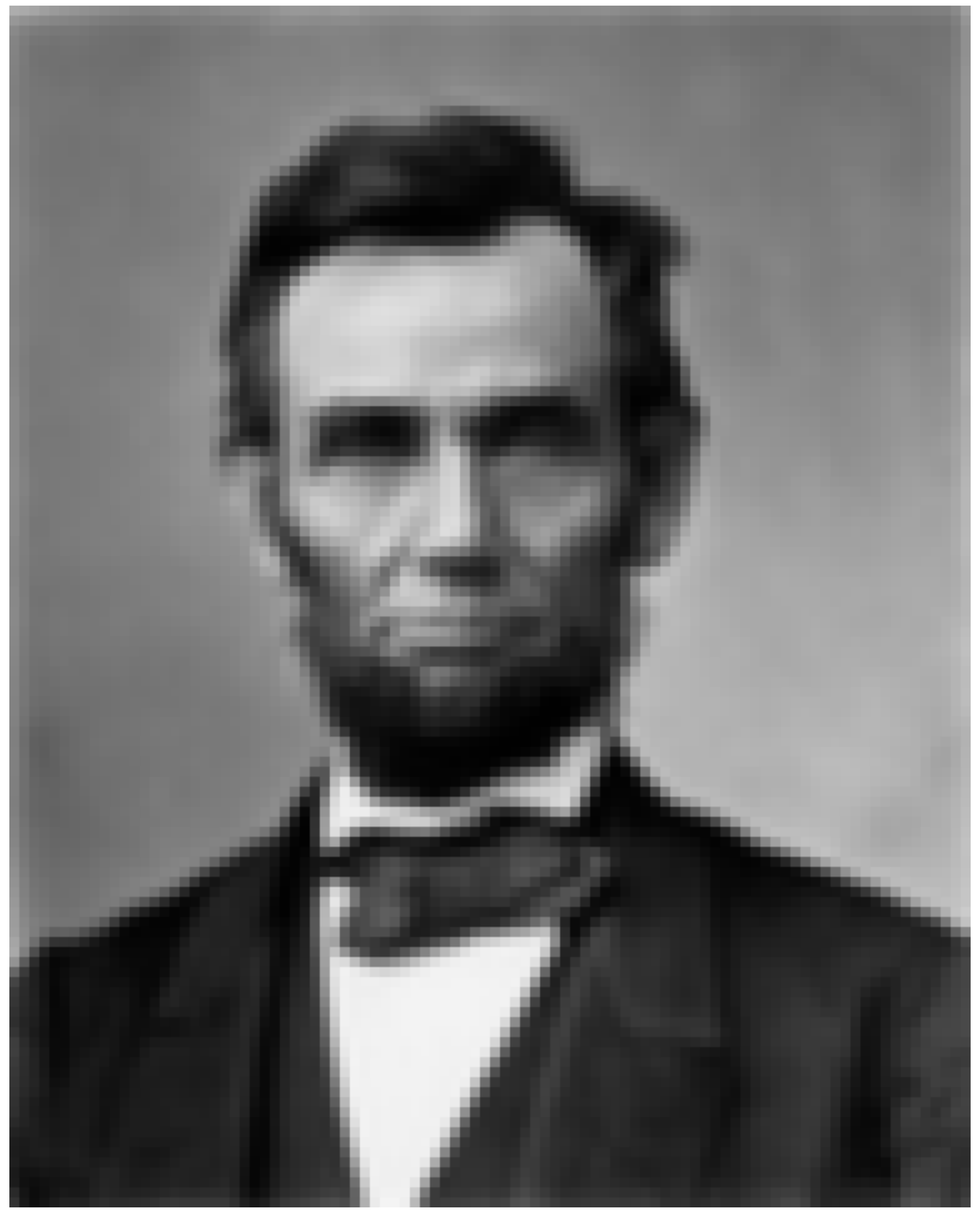


(b) Blur kernel applied.

The kernel on the right acts as an “edge filter”

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

(a) Blur kernel.



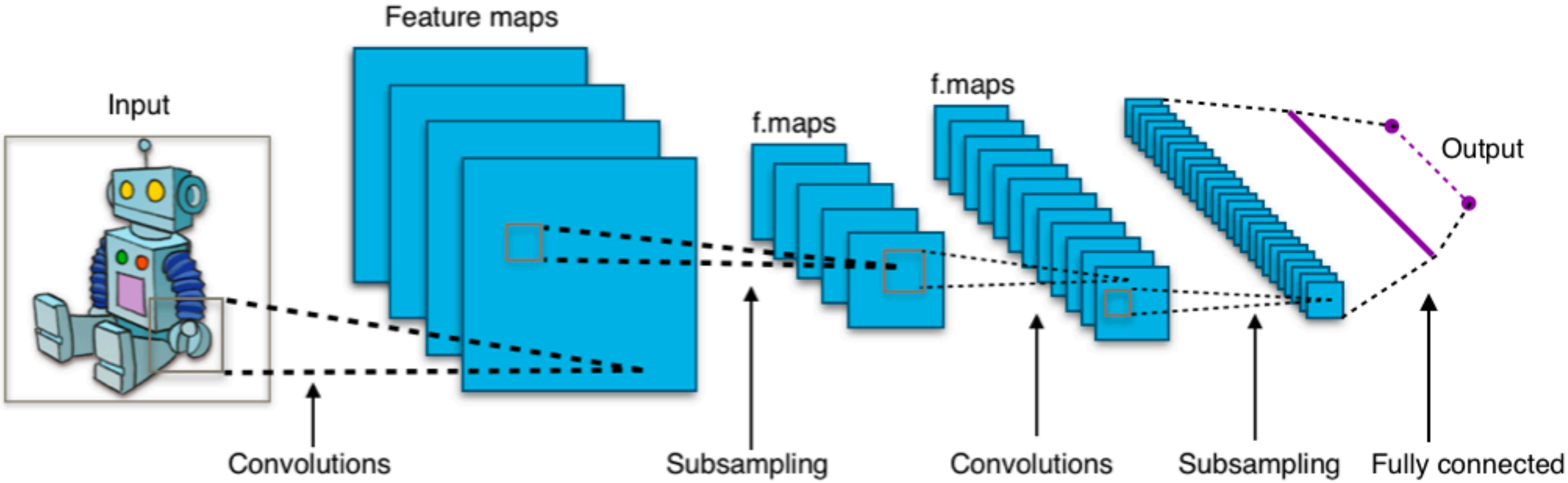
(b) Blur kernel applied.

-1	-1	-1
-1	8	-1
-1	-1	-1

(c) Edge kernel.

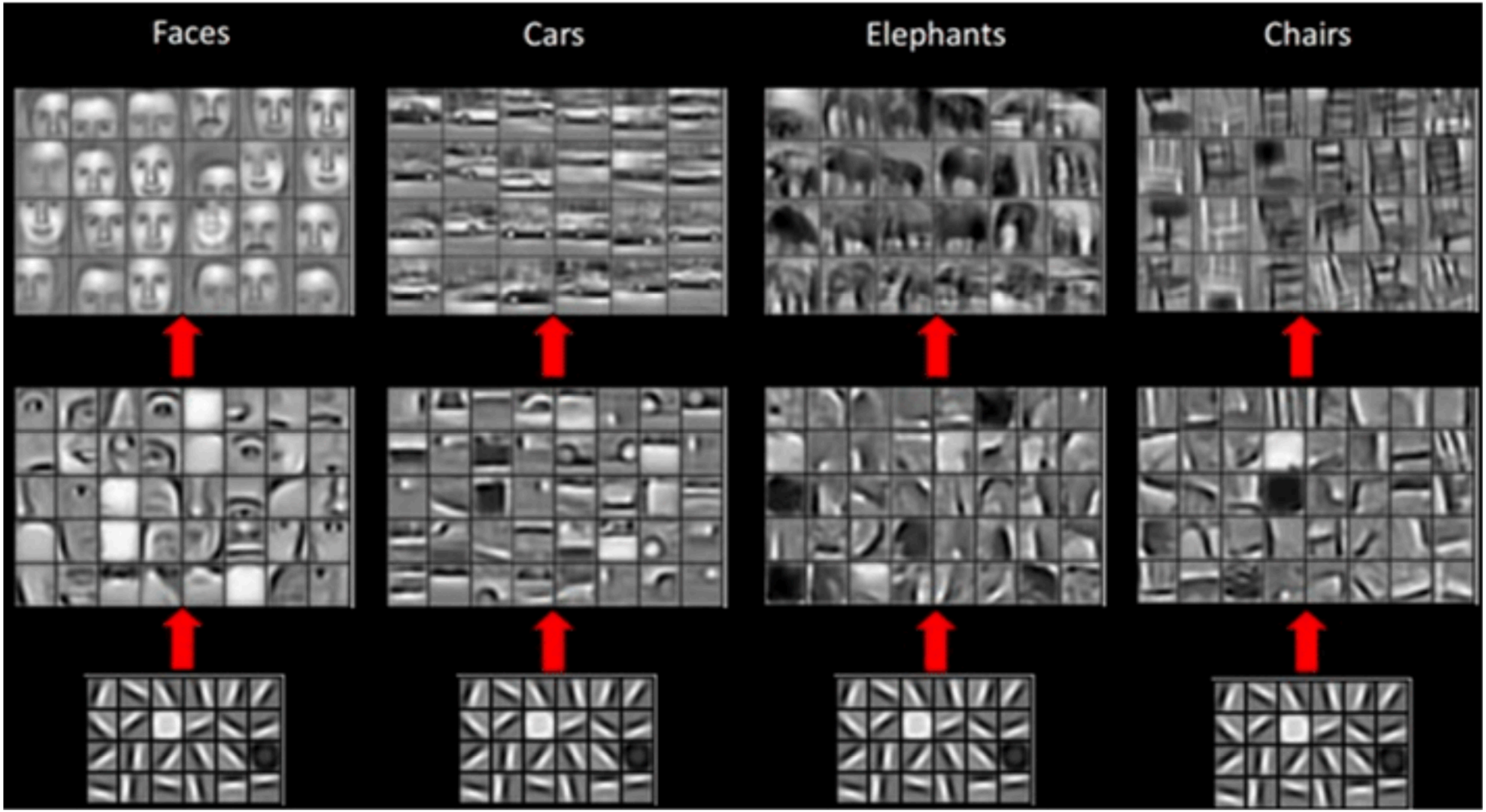
(d) Edge kernel applied.

Figure 8: Effect of convolutional image kernels.

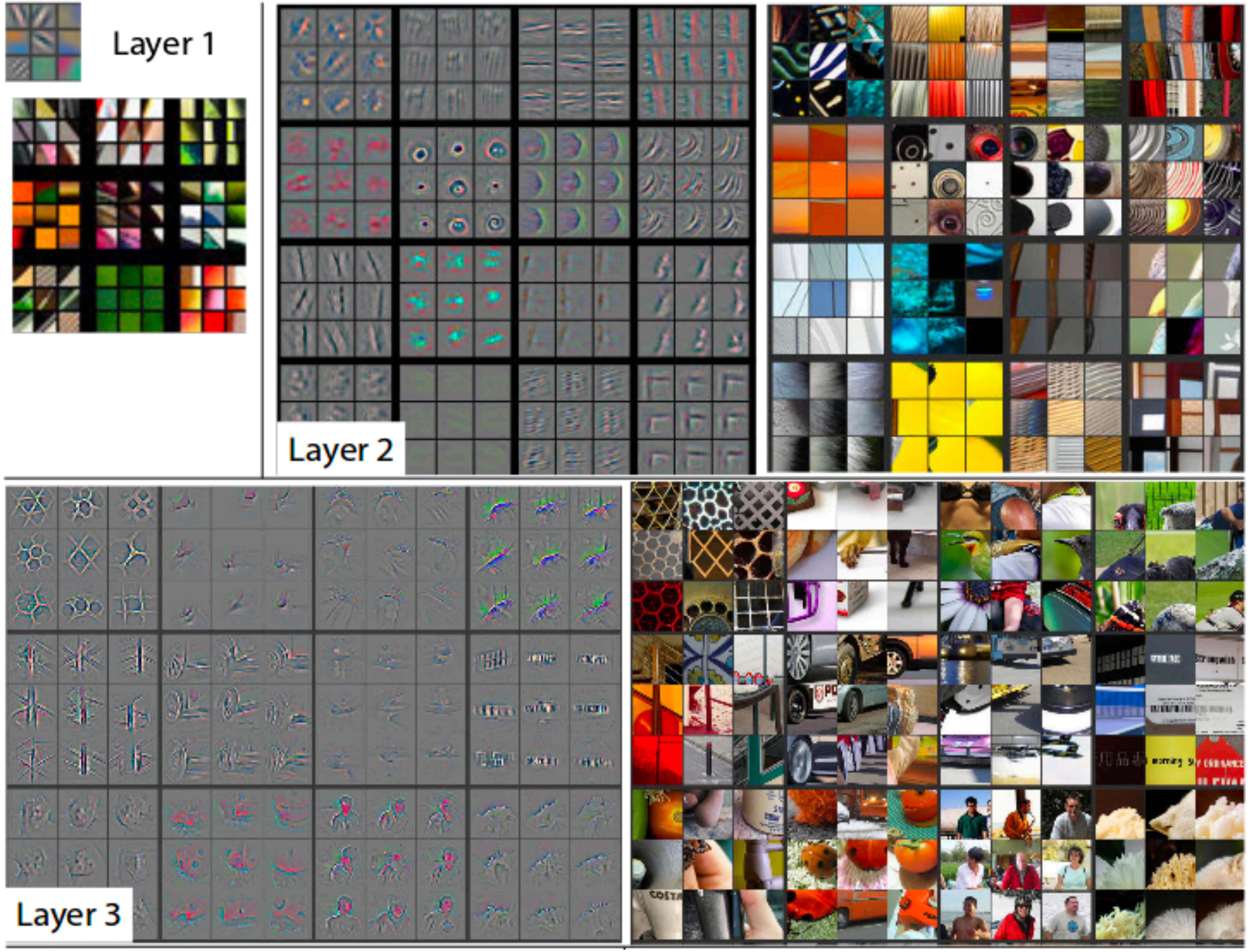


Source: Wikipedia, "Convolutional Neural Nets"

CNN layers learn filters to detect and combine higher level "features"



CNN layers learn filters to detect and combine higher level “features”



Convolutional Neural Network Visualization (Images)

http://scs.ryerson.ca/~aharley/vis/

Typical CNN architecture for NLP

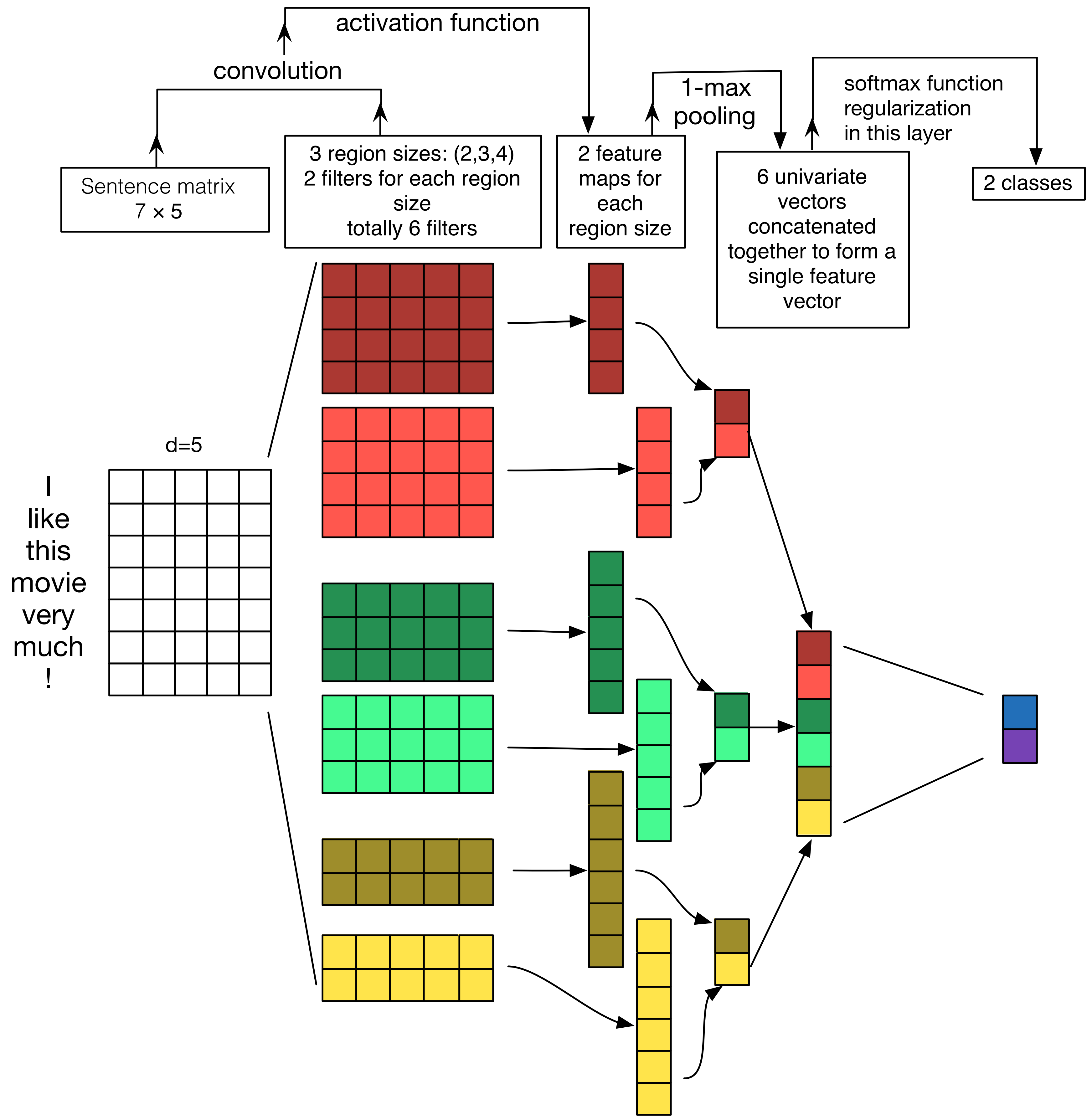


Figure 1: Illustration of a CNN architecture for sentence classification. We depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Filters perform convolutions on the sentence matrix and generate (variable-length) feature maps; 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states.

Source: Zhang and Wallace (2015)

7-gram features detected by CNN

POSITIVE

lovely	comedic	moments	and	several	fine	performances
good	script	,	good	dialogue	,	funny
sustains	throughout	is	daring	,	inventive	and
well	written	,	nicely	acted	and	beautifully
remarkably	solid	and	subtly	satirical	tour	de

NEGATIVE

,	nonexistent	plot	and	pretentious	visual	style
it	fails	the	most	basic	test	as
so	stupid	,	so	ill	conceived	,
,	too	dull	and	pretentious	to	be
hood	rats	butt	their	ugly	heads	in

'NOT'

n't	have	any	huge	laughs	in	its
no	movement	,	no	,	not	much
n't	stop	me	from	enjoying	much	of
not	that	kung	pow	is	n't	funny
not	a	moment	that	is	not	false

'TOO'

,	too	dull	and	pretentious	to	be
either	too	serious	or	too	lighthearted	,
too	slow	,	too	long	and	too
feels	too	formulaic	and	too	familiar	to
is	too	predictable	and	too	self	conscious

Source: Kalchbrenner, et al. (2014)

Typical CNN architecture for NLP

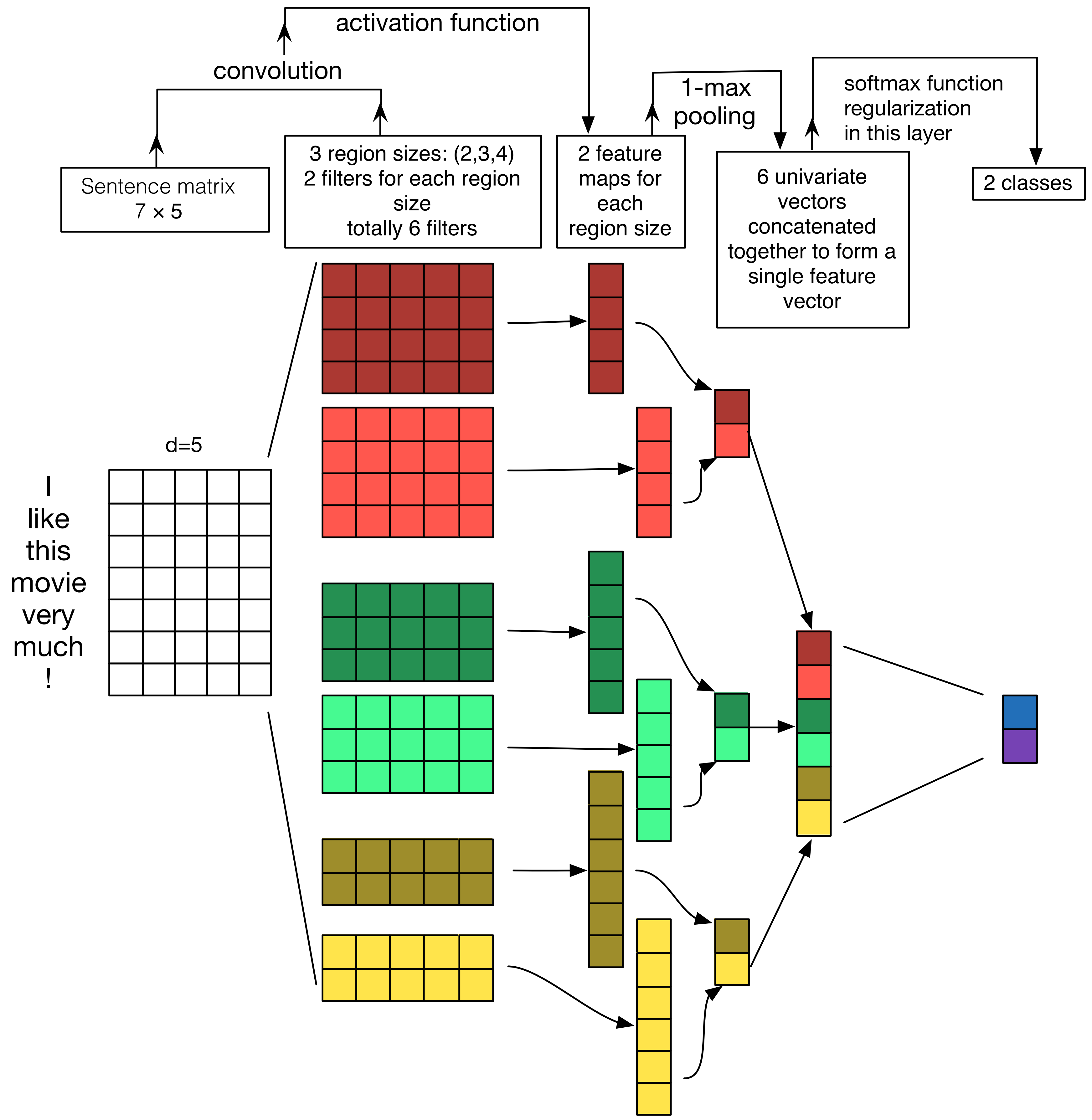


Figure 1: Illustration of a CNN architecture for sentence classification. We depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Filters perform convolutions on the sentence matrix and generate (variable-length) feature maps; 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states.

Source: Zhang and Wallace (2015)

7-gram features detected by CNN

POSITIVE

lovely	comedic	moments	and	several	fine	performances
good	script	,	good	dialogue	,	funny
sustains	throughout	is	daring	,	inventive	and
well	written	,	nicely	acted	and	beautifully
remarkably	solid	and	subtly	satirical	tour	de

NEGATIVE

,	nonexistent	plot	and	pretentious	visual	style
it	fails	the	most	basic	test	as
so	stupid	,	so	ill	conceived	,
,	too	dull	and	pretentious	to	be
hood	rats	butt	their	ugly	heads	in

'NOT'

n't	have	any	huge	laughs	in	its
no	movement	,	no	,	not	much
n't	stop	me	from	enjoying	much	of
not	that	kung	pow	is	n't	funny
not	a	moment	that	is	not	false

'TOO'

,	too	dull	and	pretentious	to	be
either	too	serious	or	too	lighthearted	,
too	slow	,	too	long	and	too
feels	too	formulaic	and	too	familiar	to
is	too	predictable	and	too	self	conscious

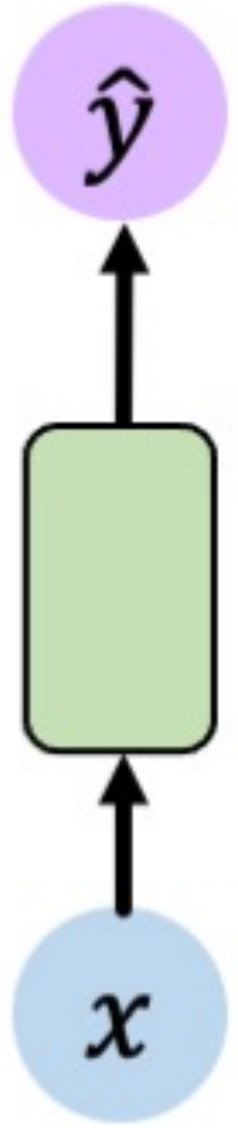
Source: Kalchbrenner, et al. (2014)

```python
# Build the model
inputs = keras.Input(shape=(None,), dtype="int32")
x = layers.Embedding(max_features, 16)(inputs) #
x = layers.Conv1D(filters=128, kernel_size=5, strides=1, padding='same', activation='relu')(x) #
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(16, activation = 'relu')(x)
outputs = layers.Dense(1, activation="sigmoid")(x) #
model = keras.Model(inputs, outputs) #
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding_3 (Embedding)      (None, None, 16)          80000
_________________________________________________________________
conv1d_3 (Conv1D)            (None, None, 128)         10368
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 16)                2064
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 17
=================================================================
Total params: 92,449
Trainable params: 92,449
Non-trainable params: 0
_________________________________________________________________
```

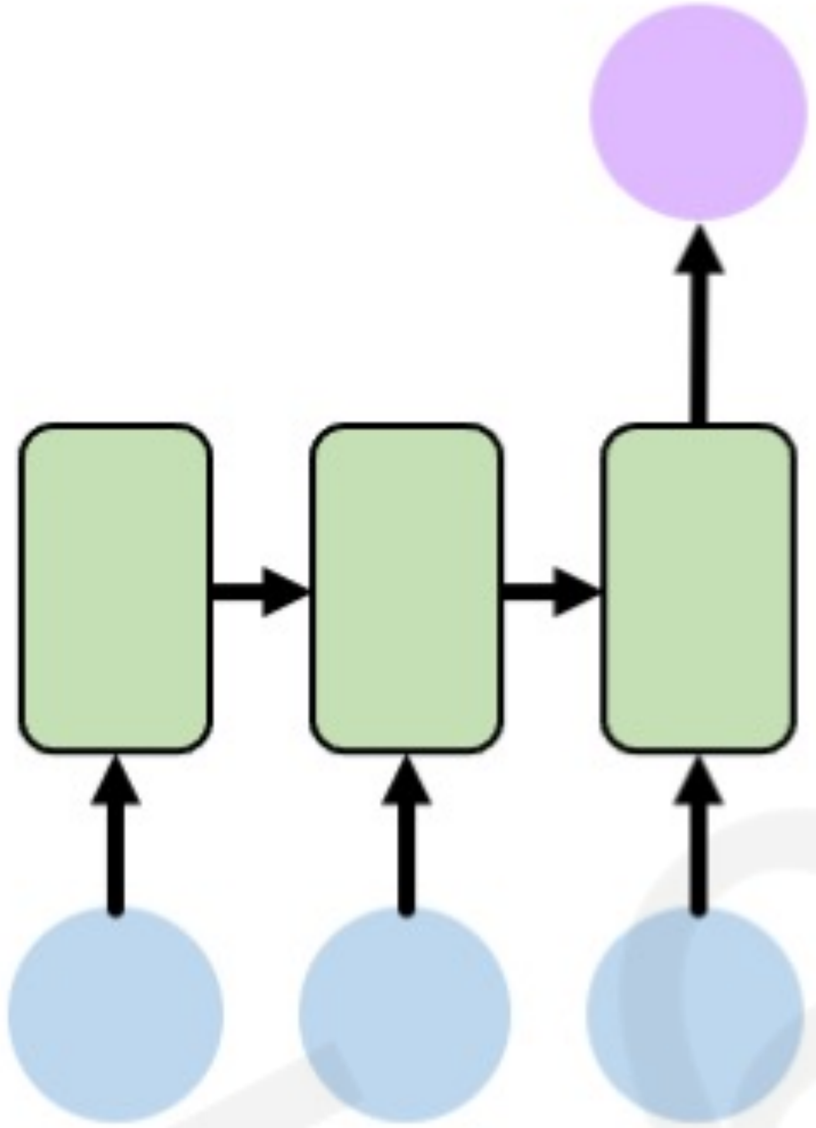
Modeling sequence with recurrence

Sequence Modeling Applications

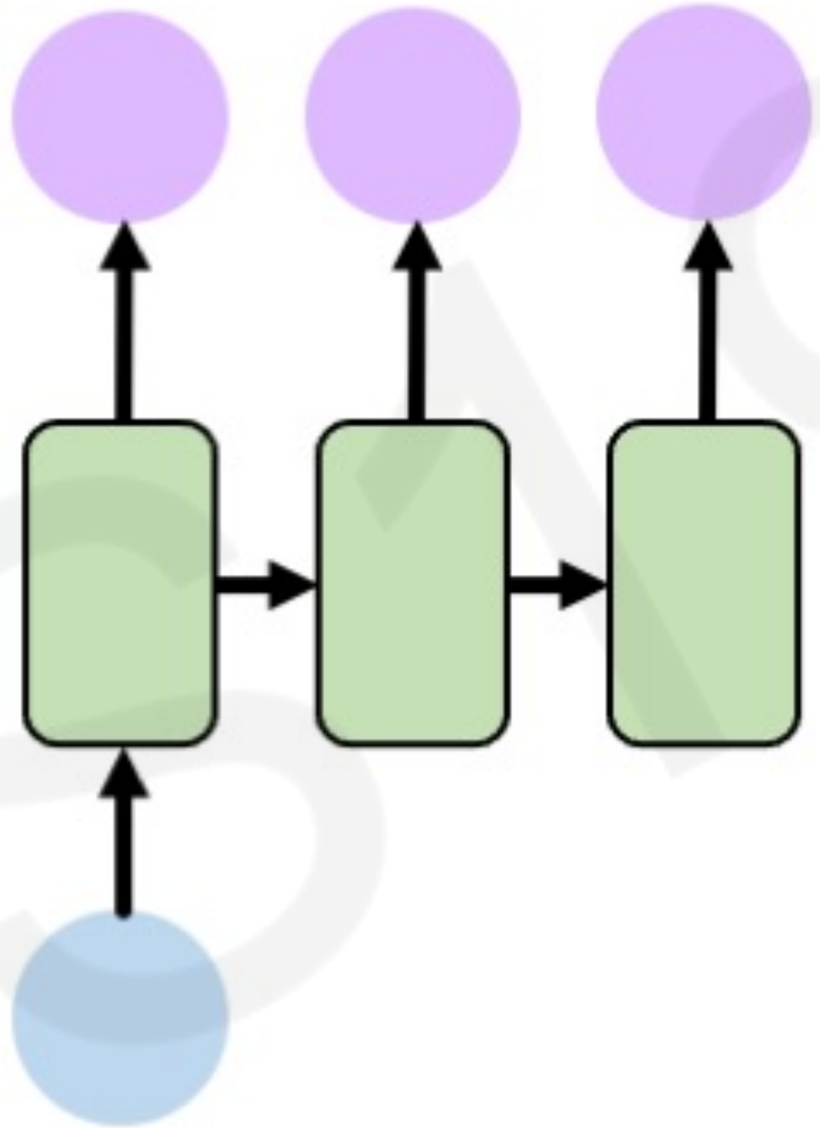


One to One
Binary Classification

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

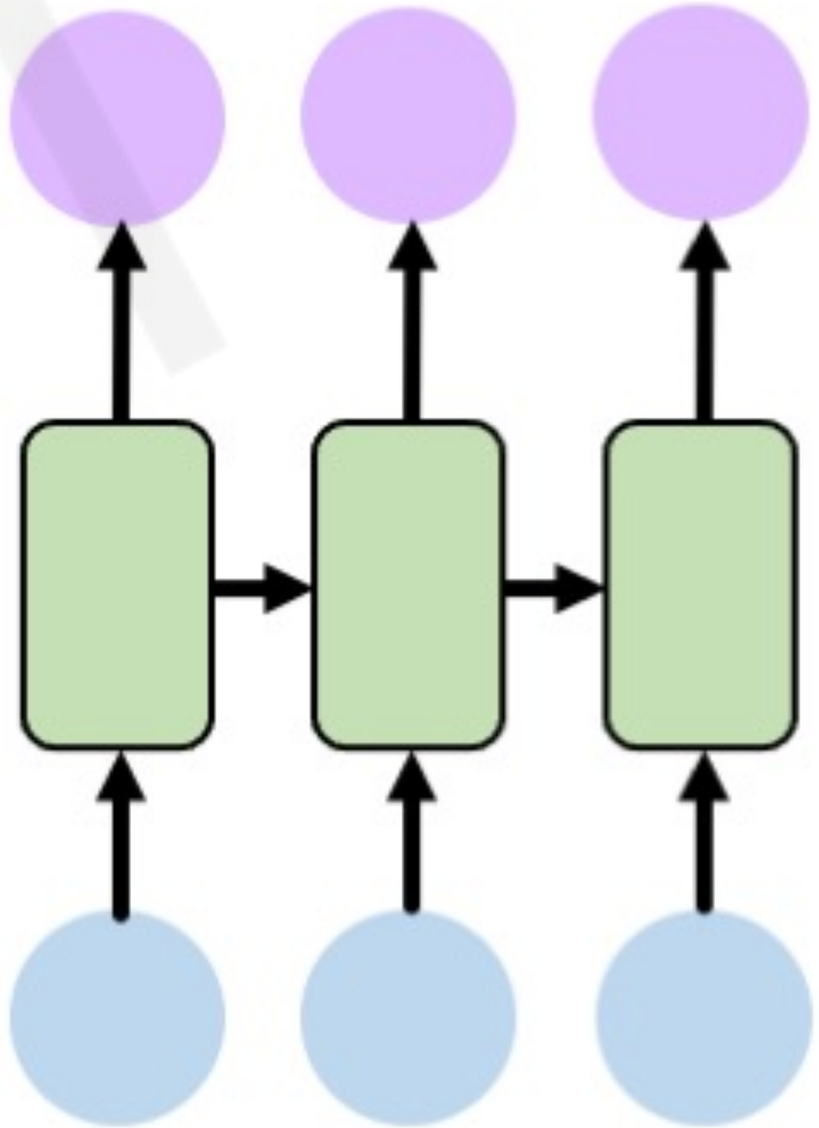


Many to One
Sentiment Classification



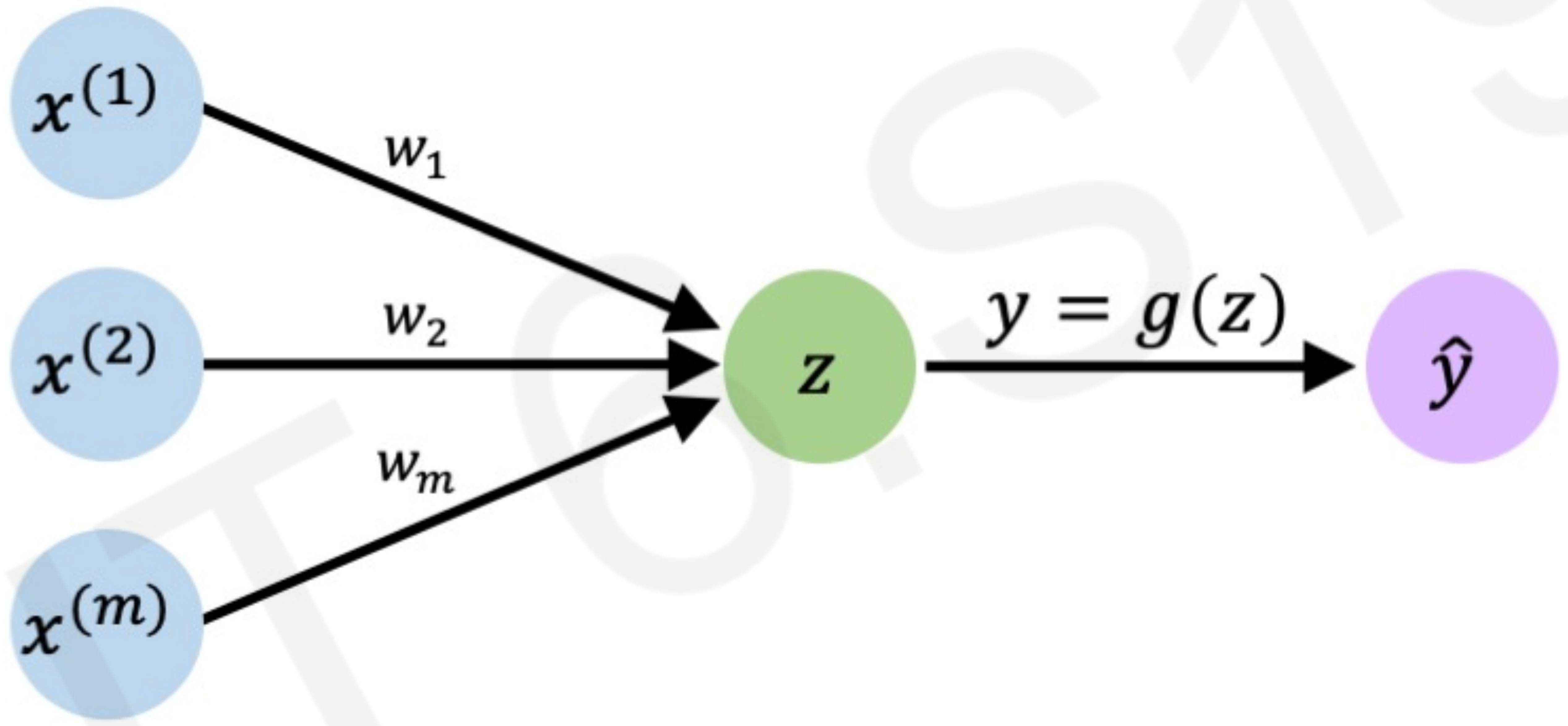
One to Many
Image Captioning

"A baseball player throws a ball."

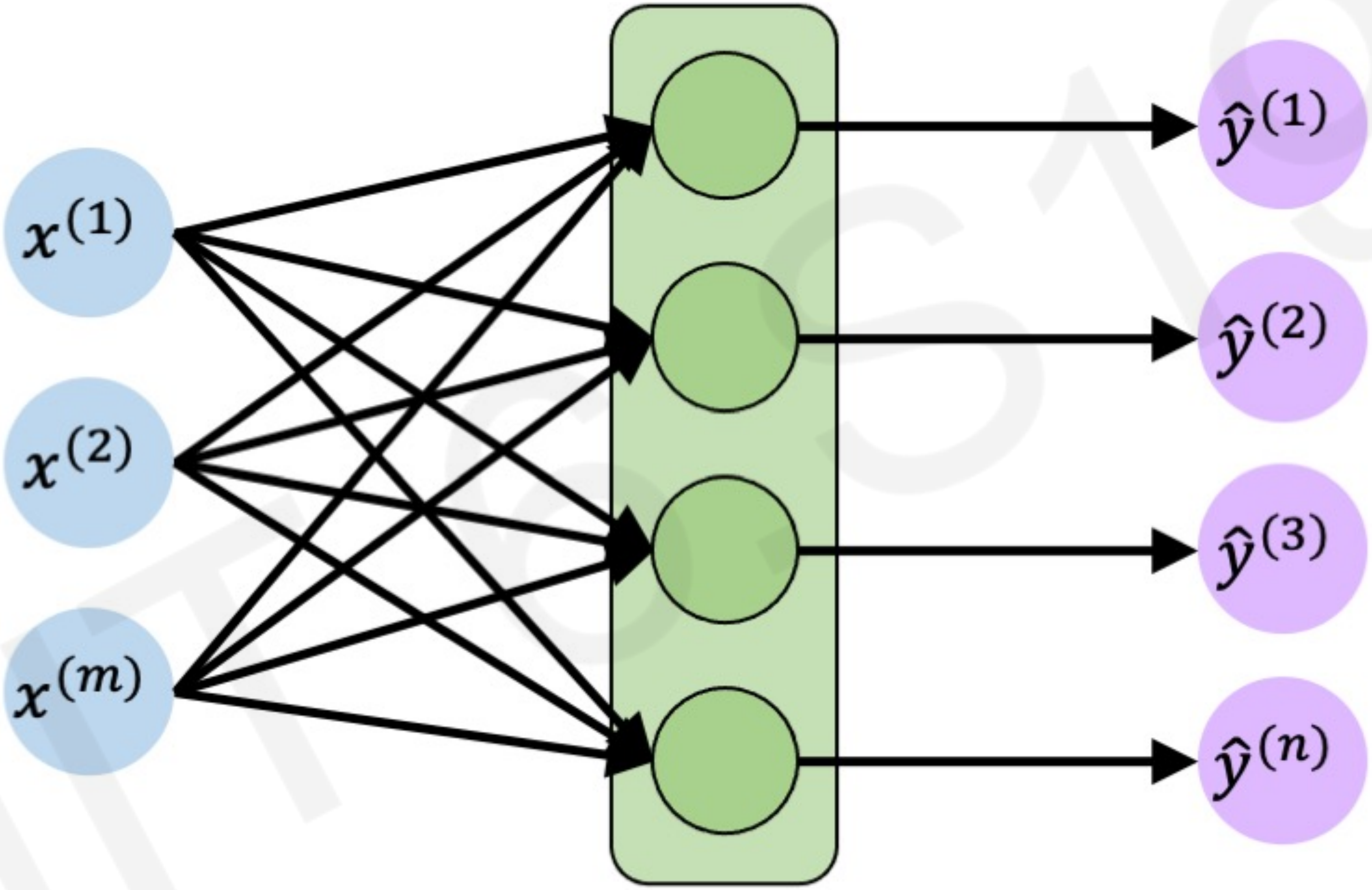


Many to Many
Machine Translation

The Perceptron Revisited

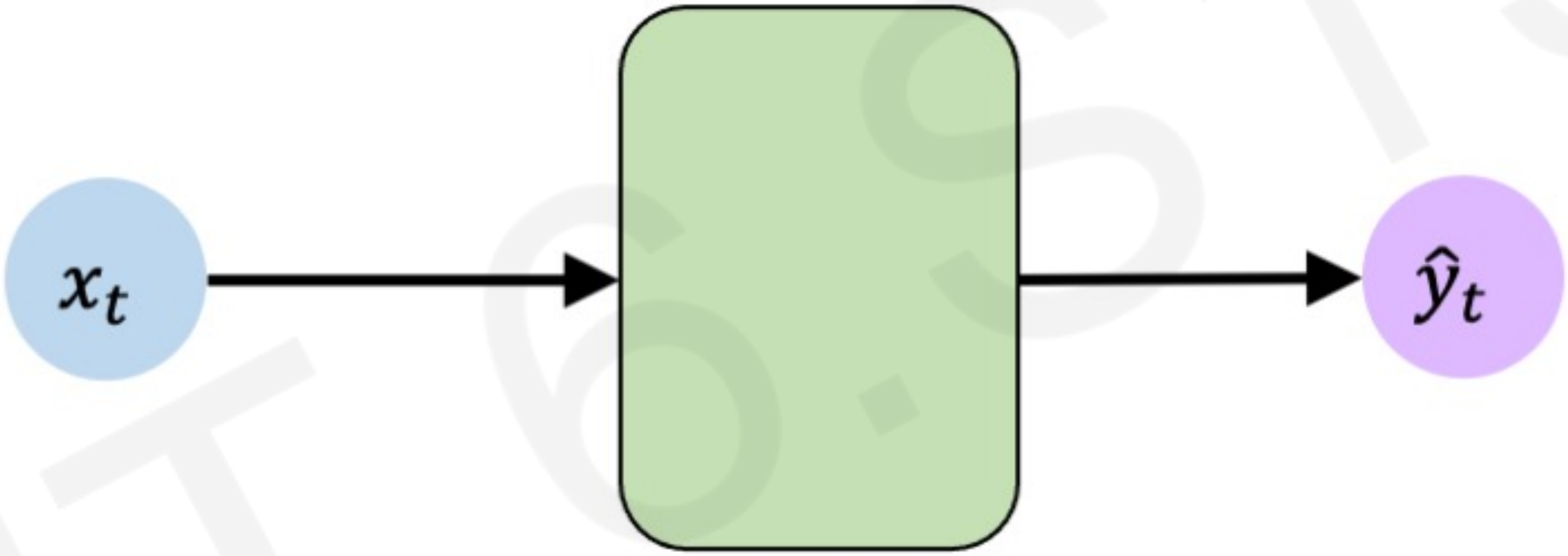


Feed-Forward Networks Revisited



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

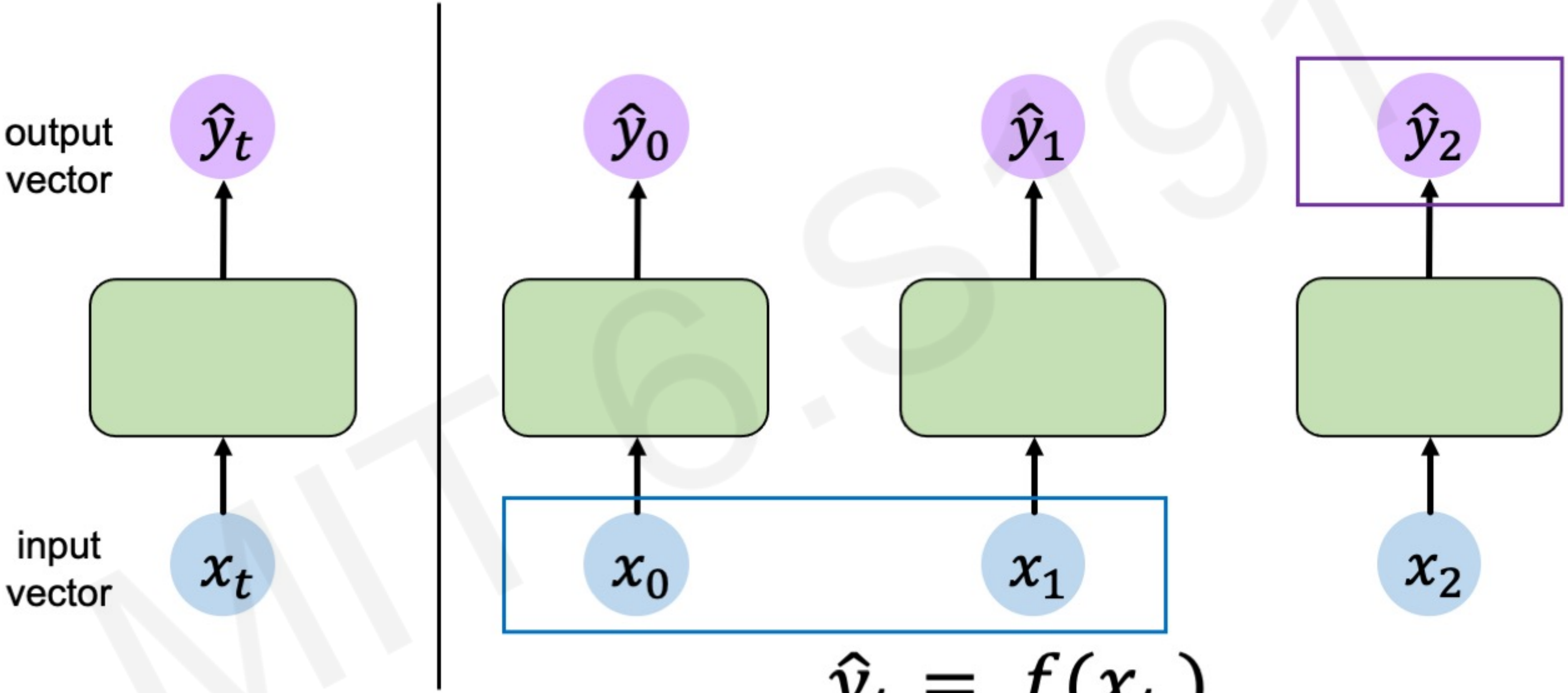
Feed-Forward Networks Revisited



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

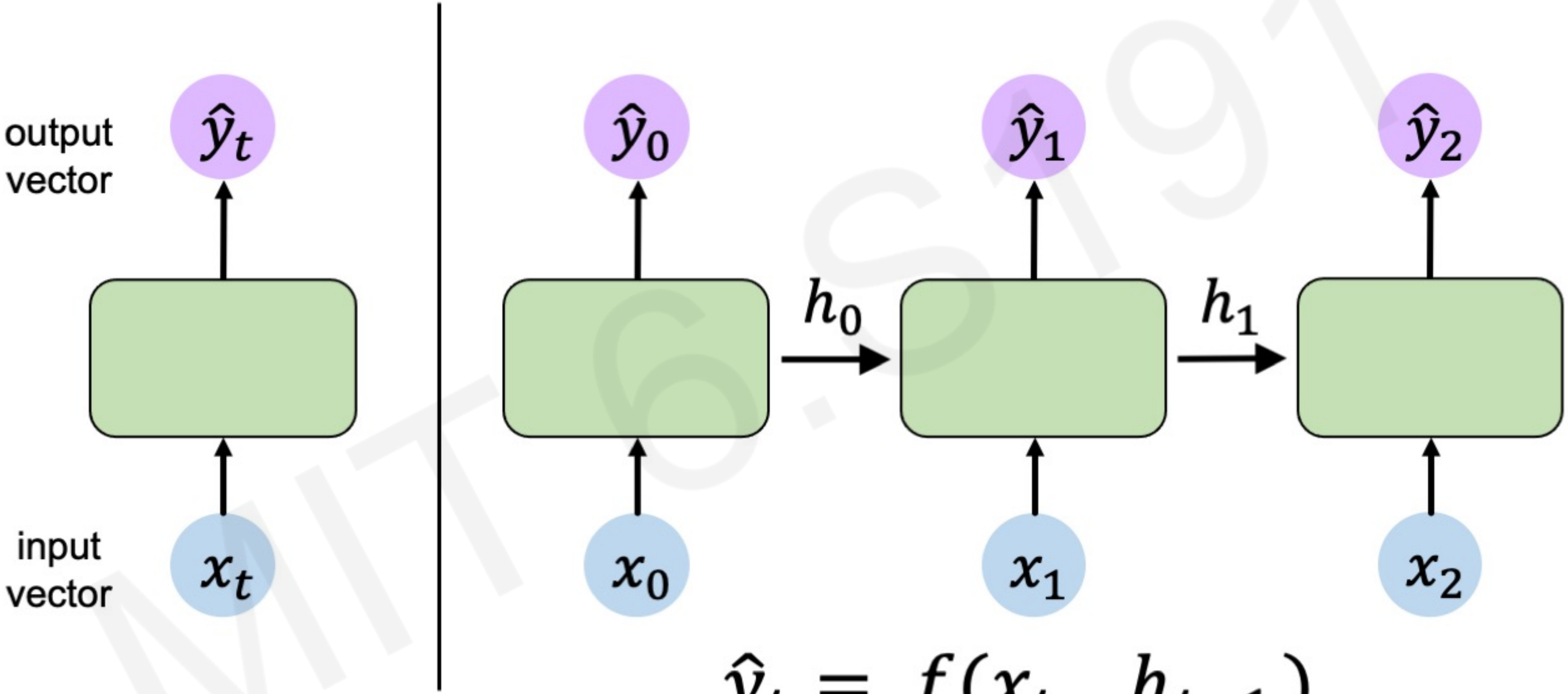
$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$

Handling Individual Time Steps



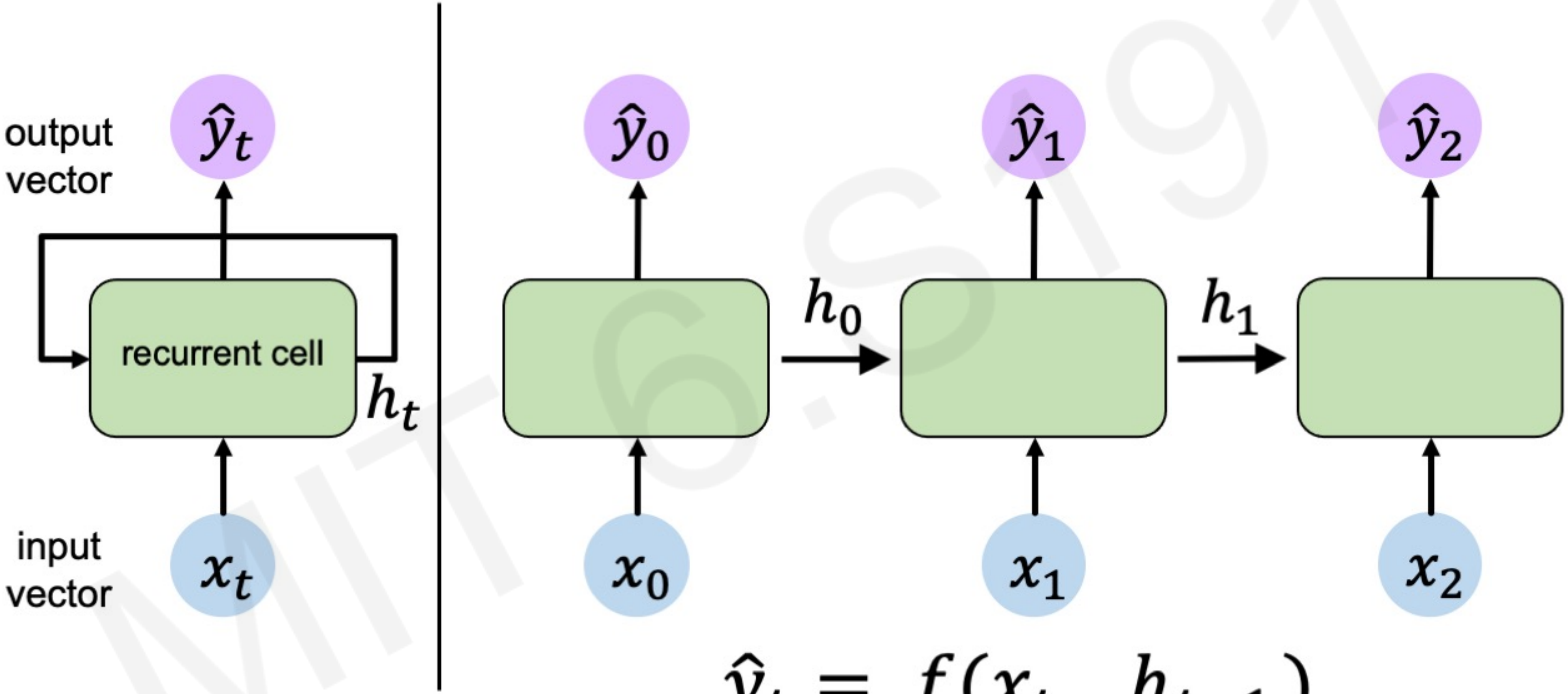
$$\hat{y}_t = f(x_t)$$

Neurons with Recurrence



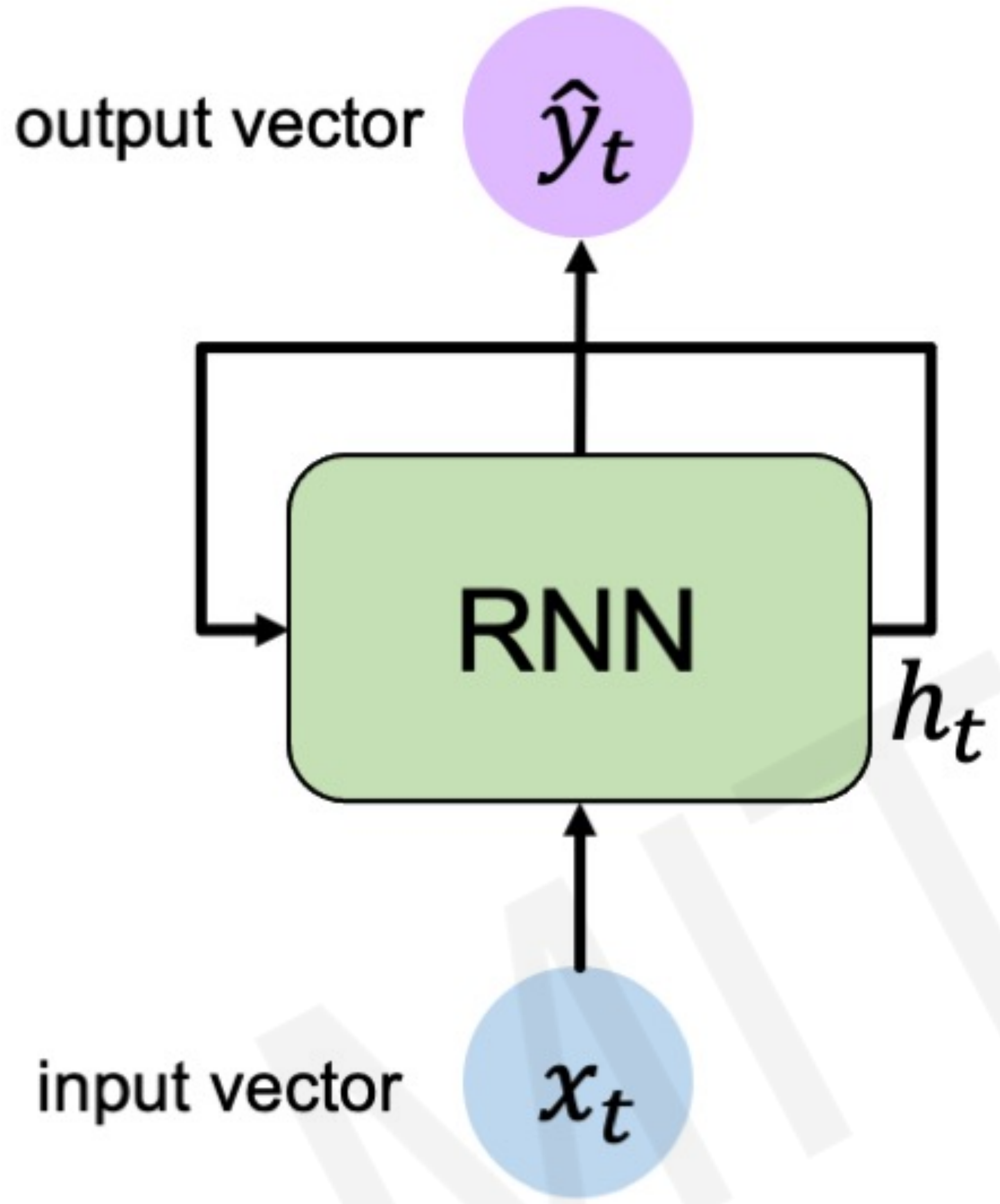
$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}},\ \underbrace{h_{t-1}}_{\text{past memory}})$$

Neurons with Recurrence



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

Recurrent Neural Networks (RNNs)



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

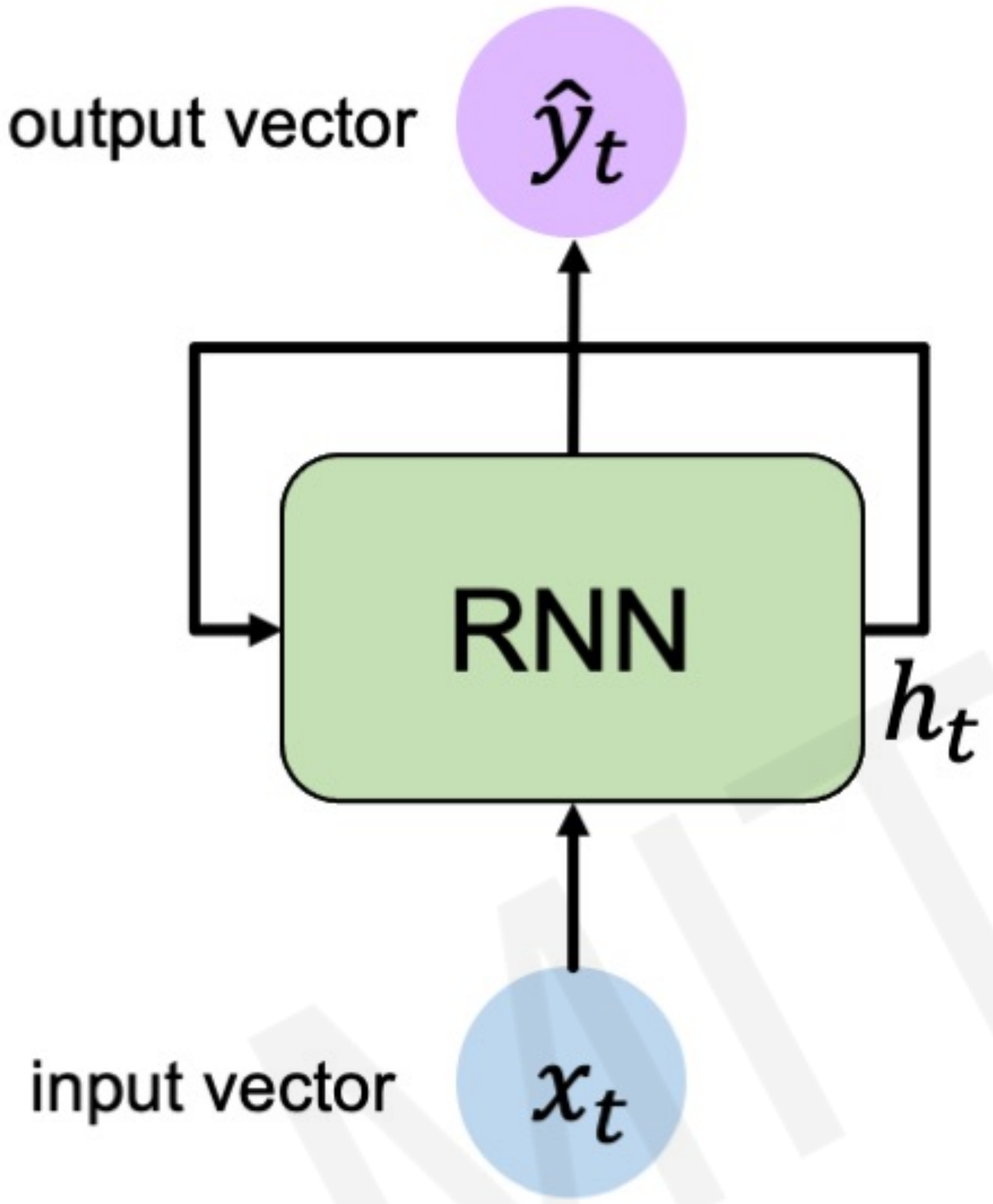
$$\boxed{h_t} = \boxed{f_W}(\boxed{x_t}, \boxed{h_{t-1}})$$

cell state | function with weights W | input | old state

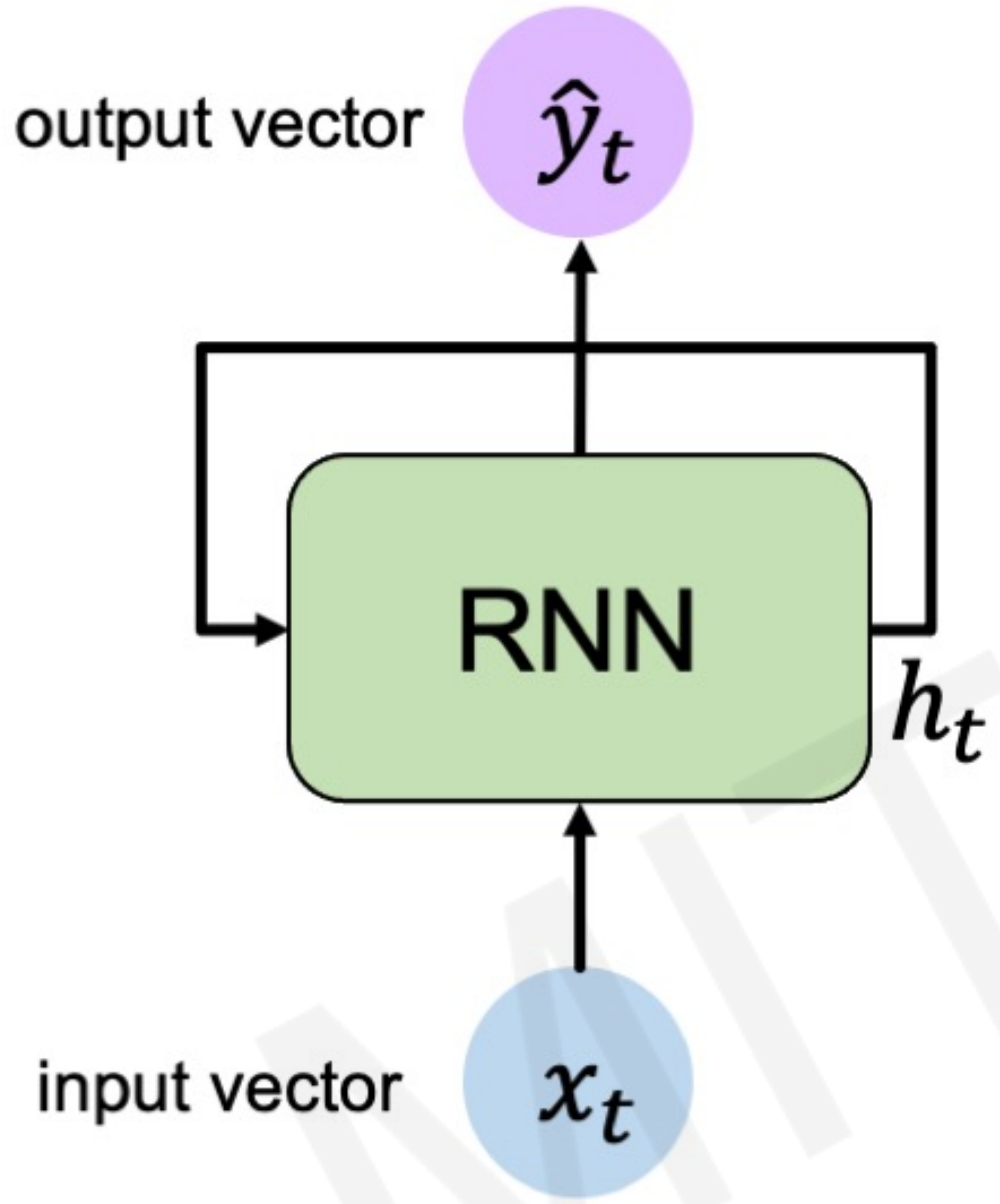
Note: the same function and set of parameters are used at every time step

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

RNN State Update and Output



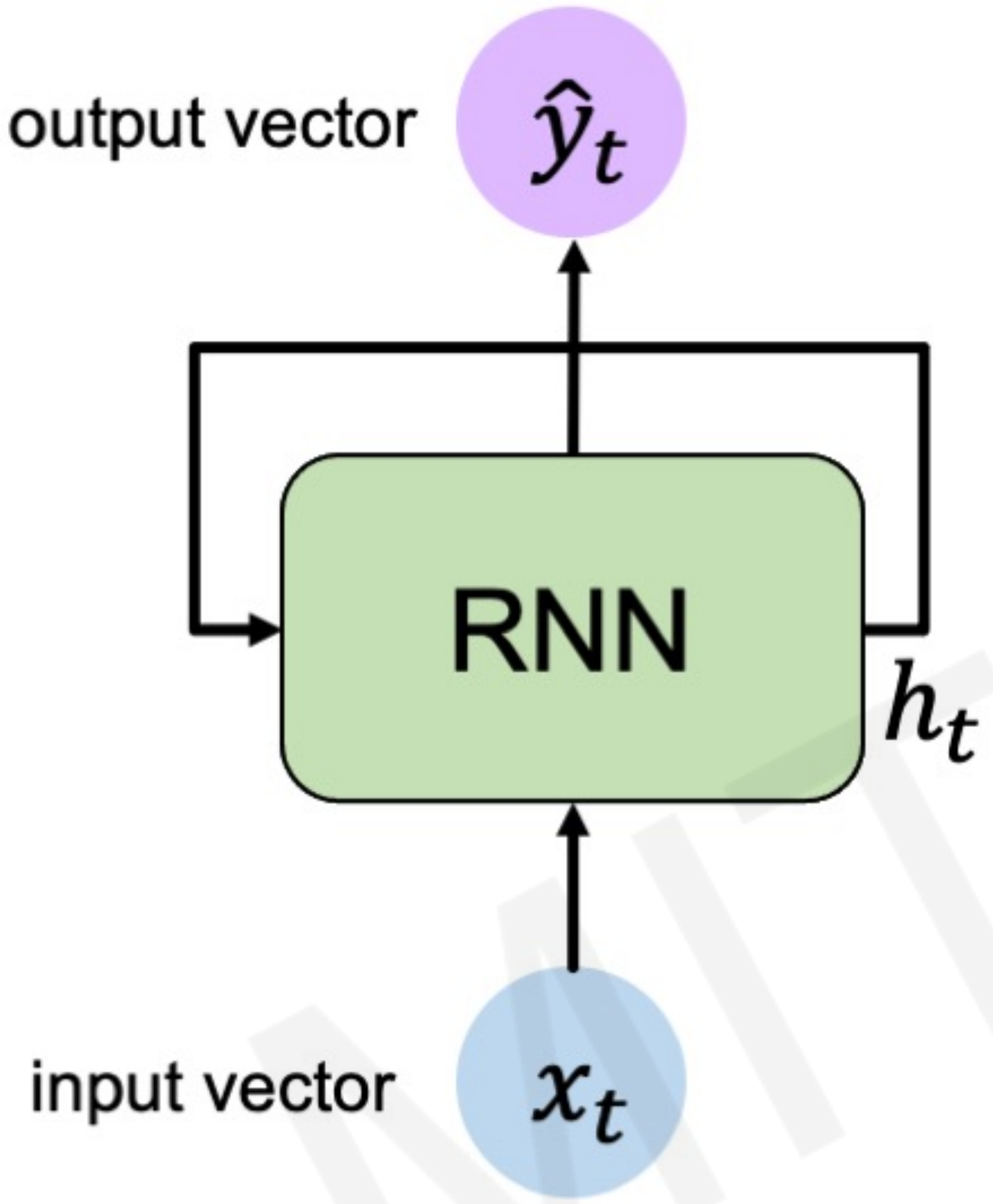
RNN State Update and Output



Input Vector

x_t

RNN State Update and Output



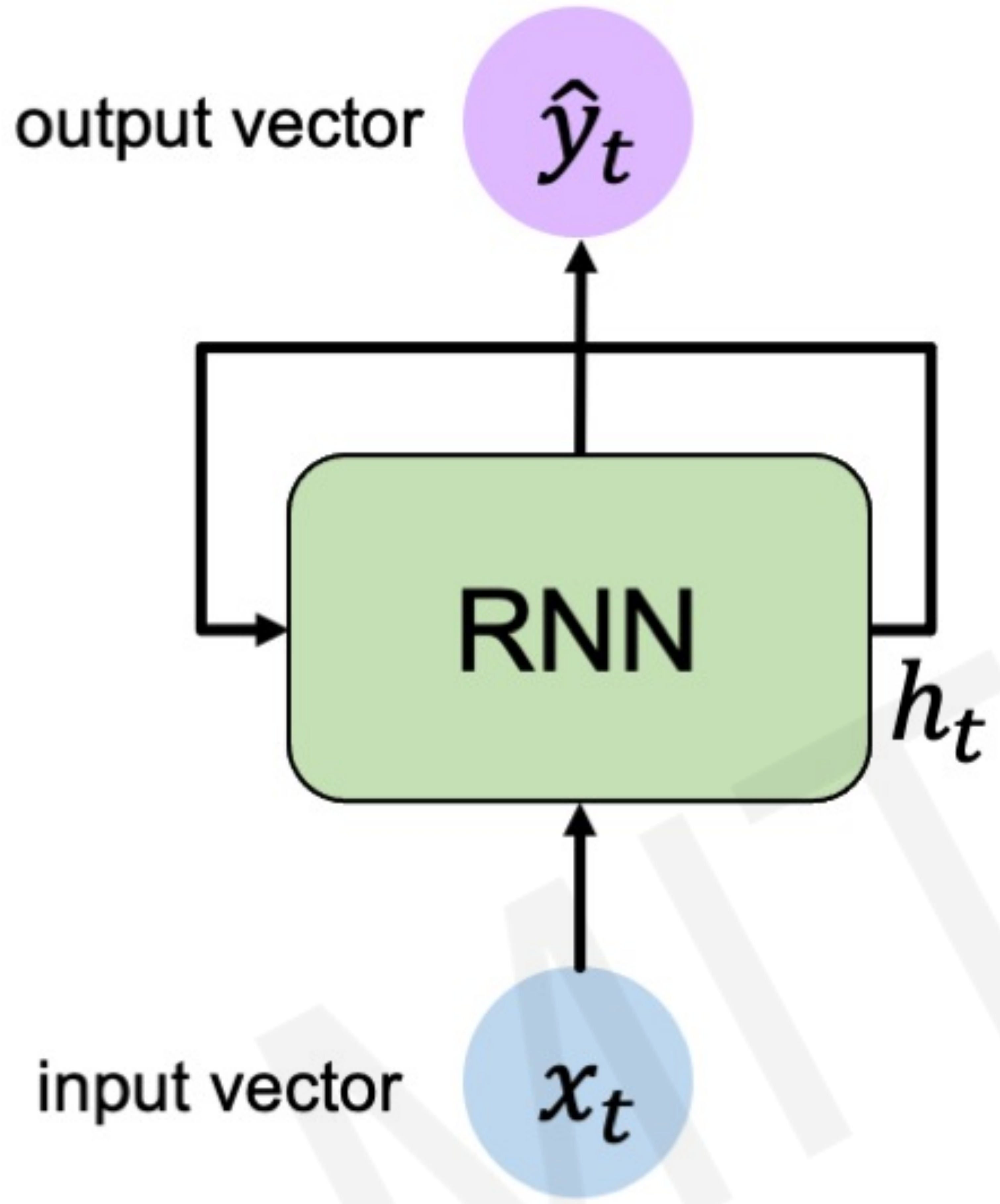
Update Hidden State

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

Input Vector

$$x_t$$

RNN State Update and Output



Output Vector

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

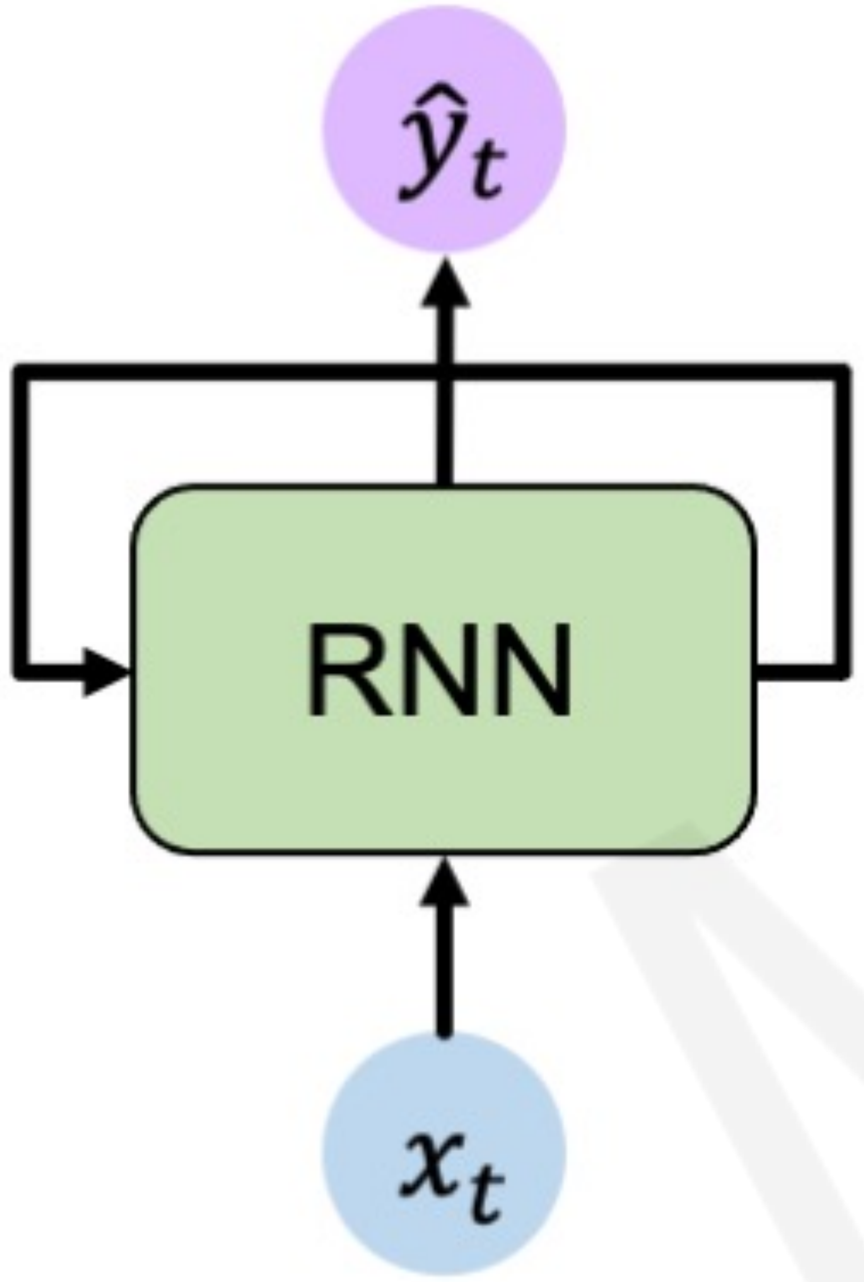
Update Hidden State

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

Input Vector

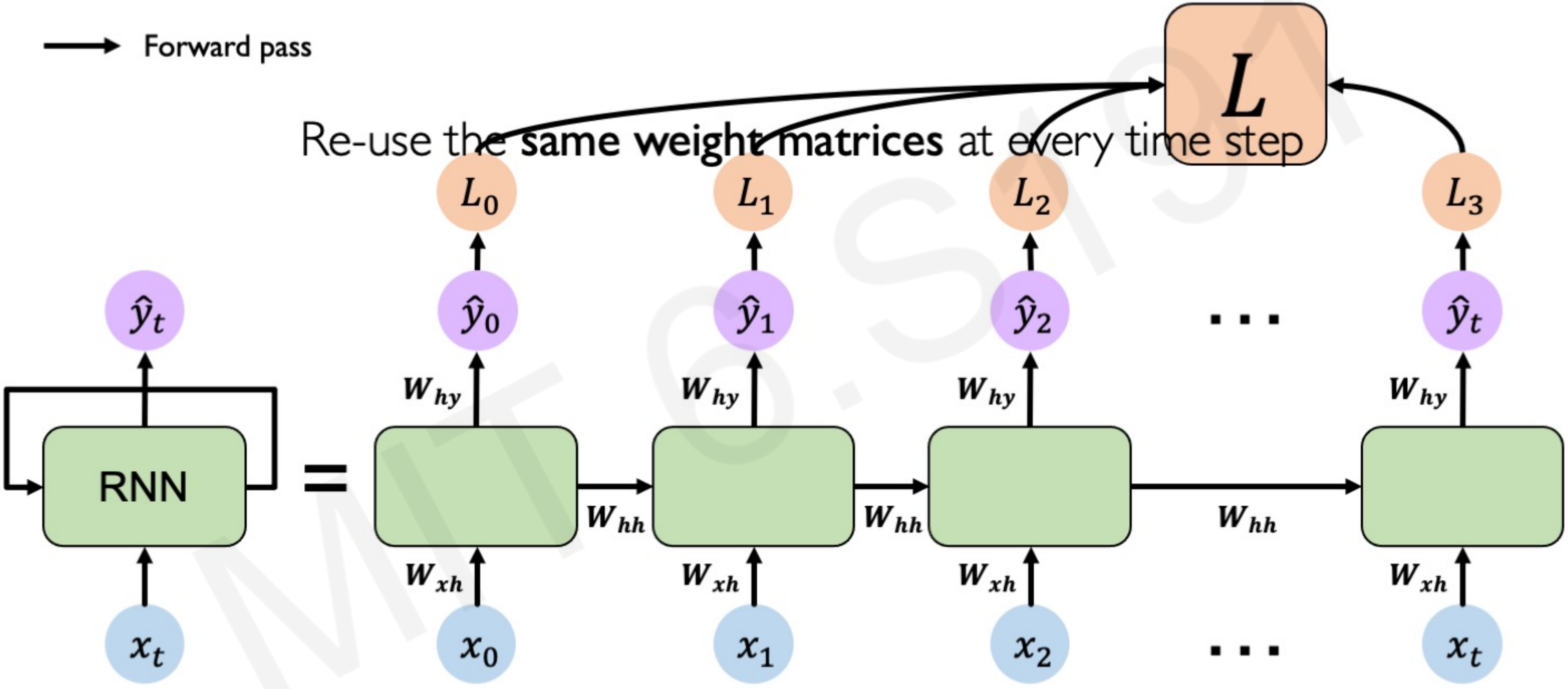
$$x_t$$

RNNs: Computational Graph Across Time



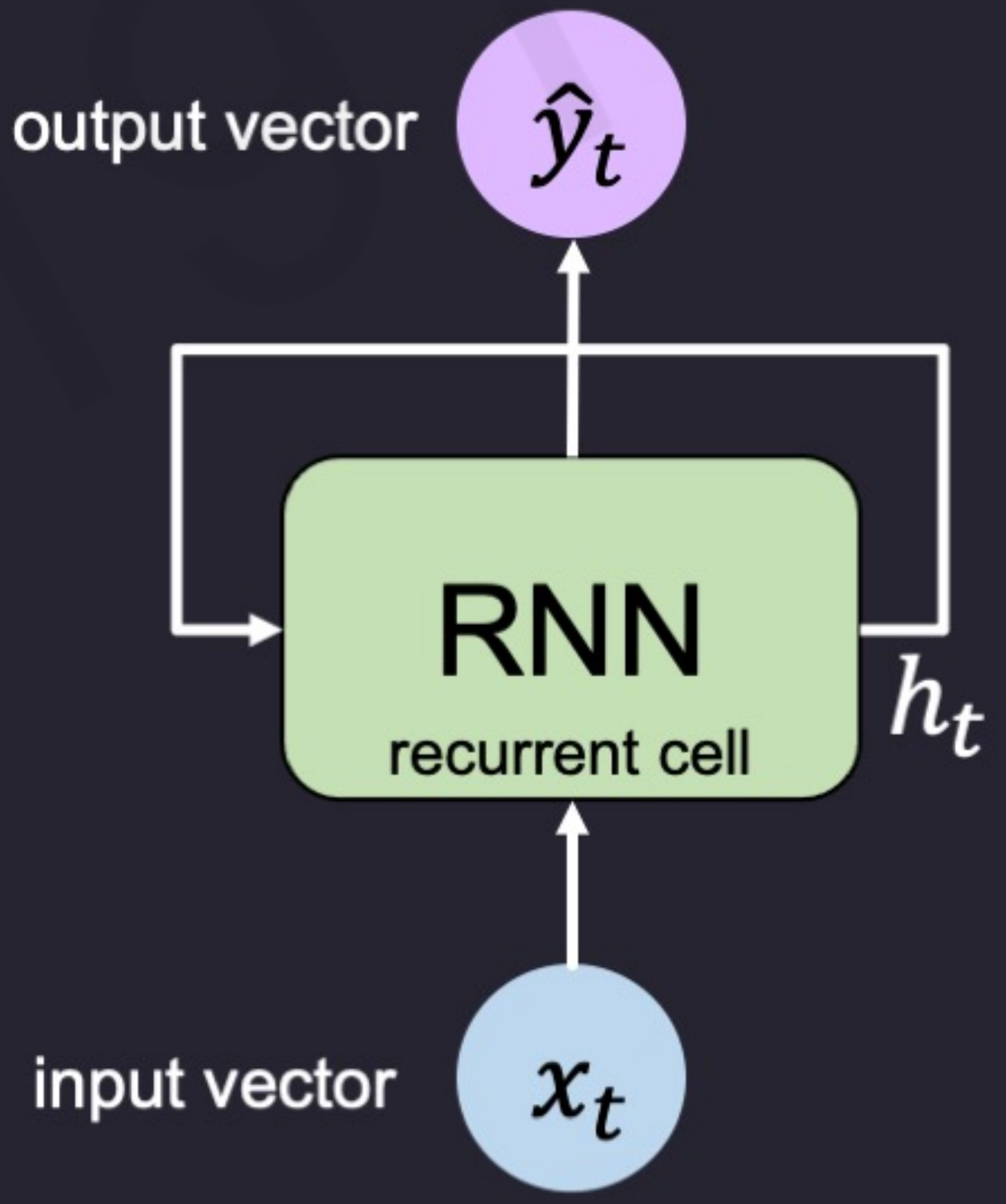
= Represent as computational graph unrolled across time

RNNs: Computational Graph Across Time

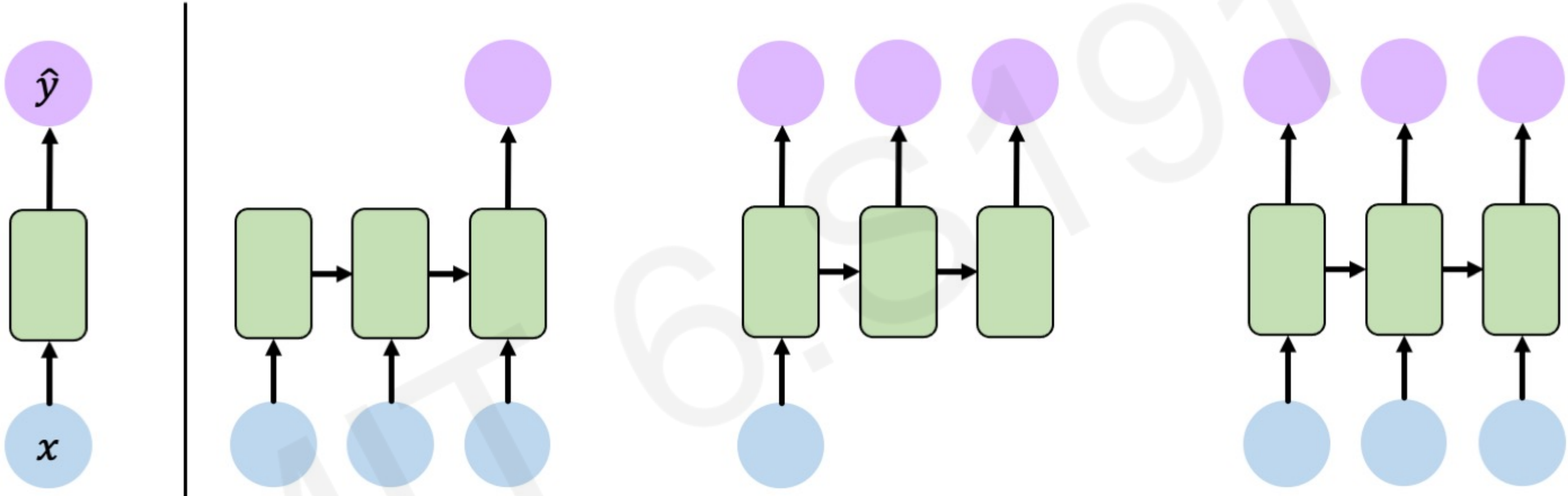


RNN Implementation in TensorFlow

```
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

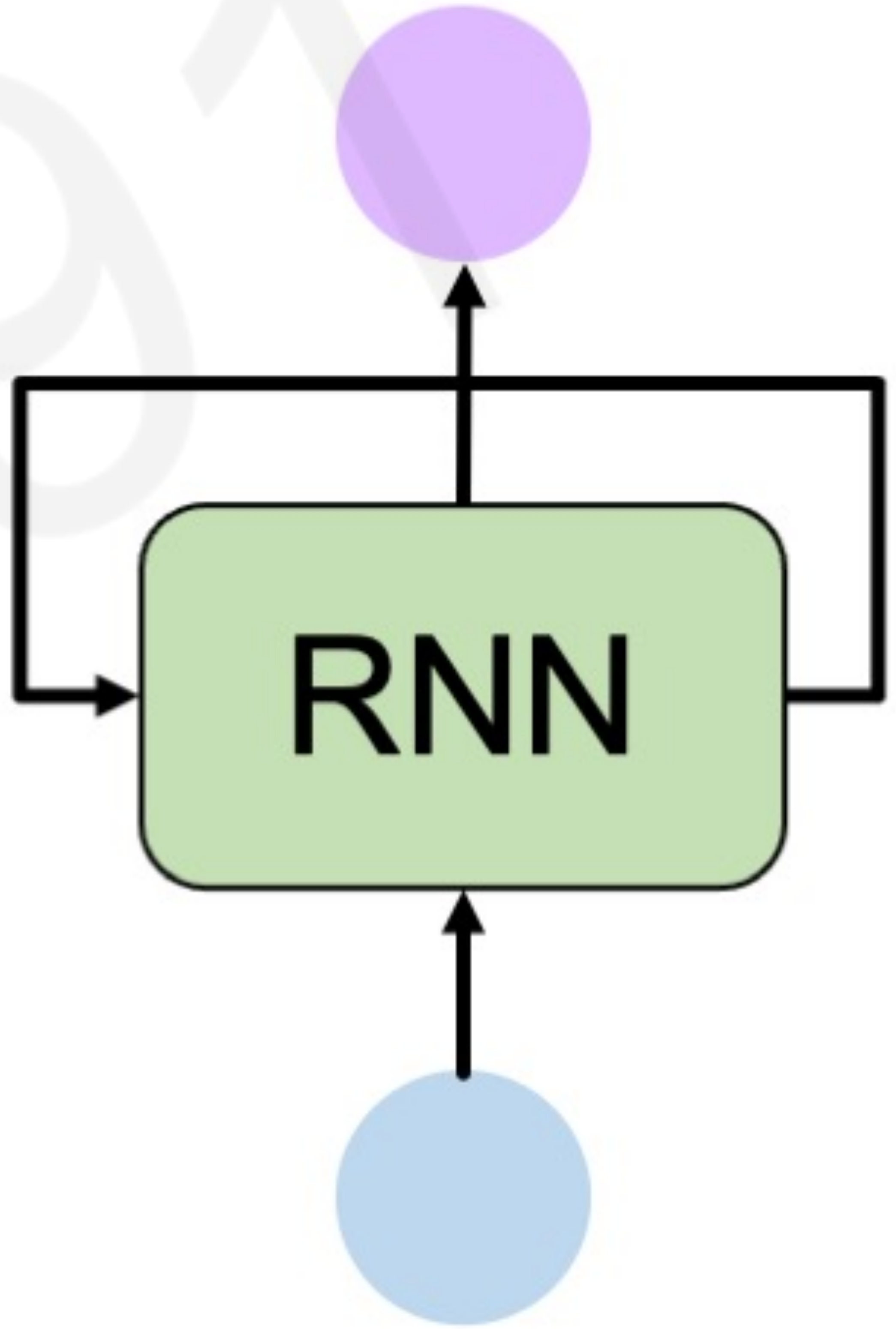
… and many other architectures and applications

6.S191 Lab!

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

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

A Sequence Modeling Problem: Predict the Next Word

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

given these words

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

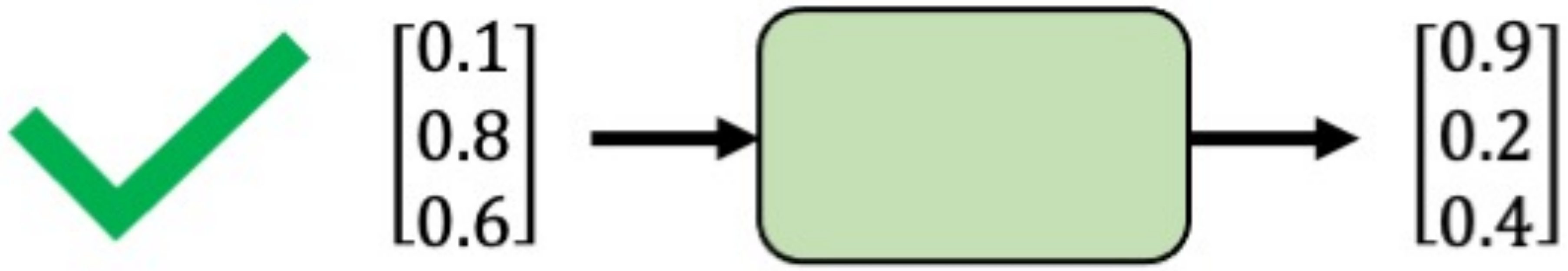
"This morning I took my cat for a walk."

given these words

predict the next word

Representing Language to a Neural Network

Neural networks cannot interpret words



Neural networks require numerical inputs

A Sequence Modeling Problem: Predict the Next Word

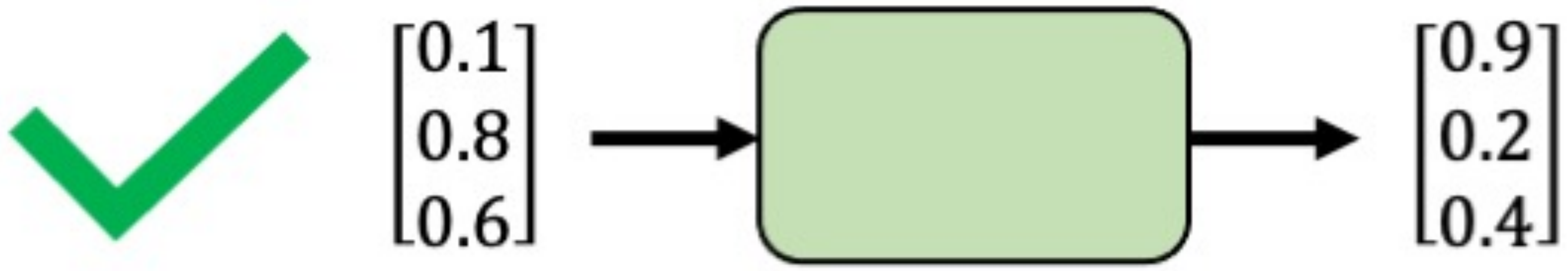
"This morning I took my cat for a walk."

given these words

predict the next word

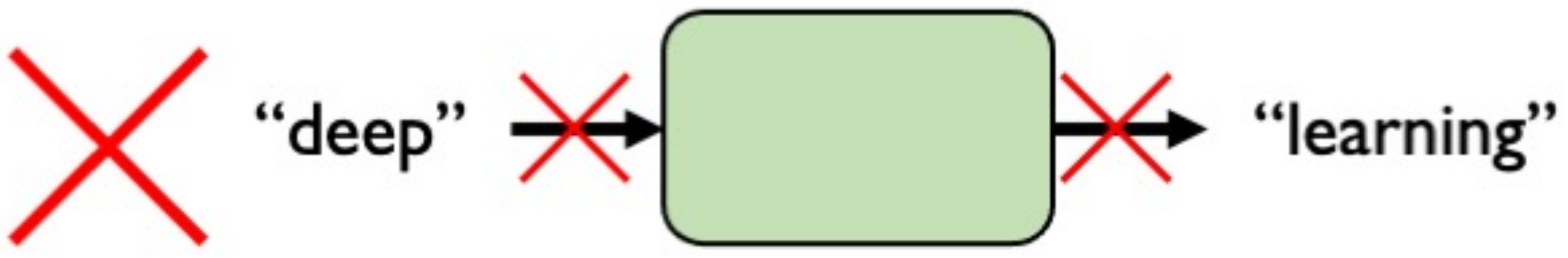
Representing Language to a Neural Network

Neural networks cannot interpret words

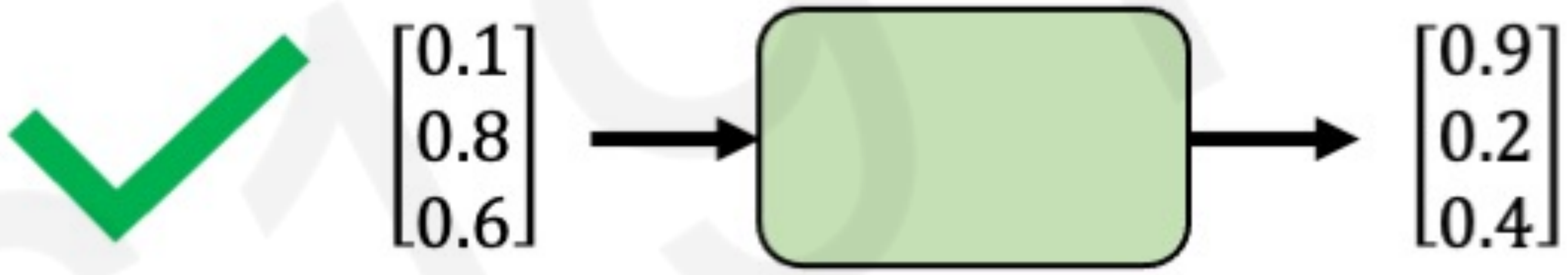


Neural networks require numerical inputs

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.

this cat for took my walk I a morning

1. Vocabulary:
Corpus of words

a → 1
cat → 2
… …
walk → N

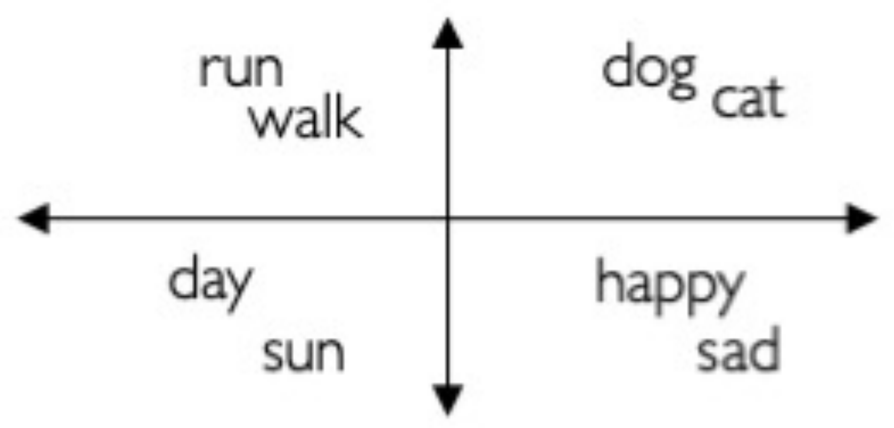
2. Indexing:
Word to index

One-hot embedding

"cat" = [0, 1, 0, 0, 0, 0]

i-th index

Learned embedding



3. Embedding:
Index to fixed-sized vector

Handle Variable Sequence Lengths

The food was <u>great</u>

vs.

We visited a restaurant for <u>lunch</u>

vs.

We were hungry but cleaned the house before <u>eating</u>

Model Long-Term Dependencies

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

J'aime 6.S191!

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.

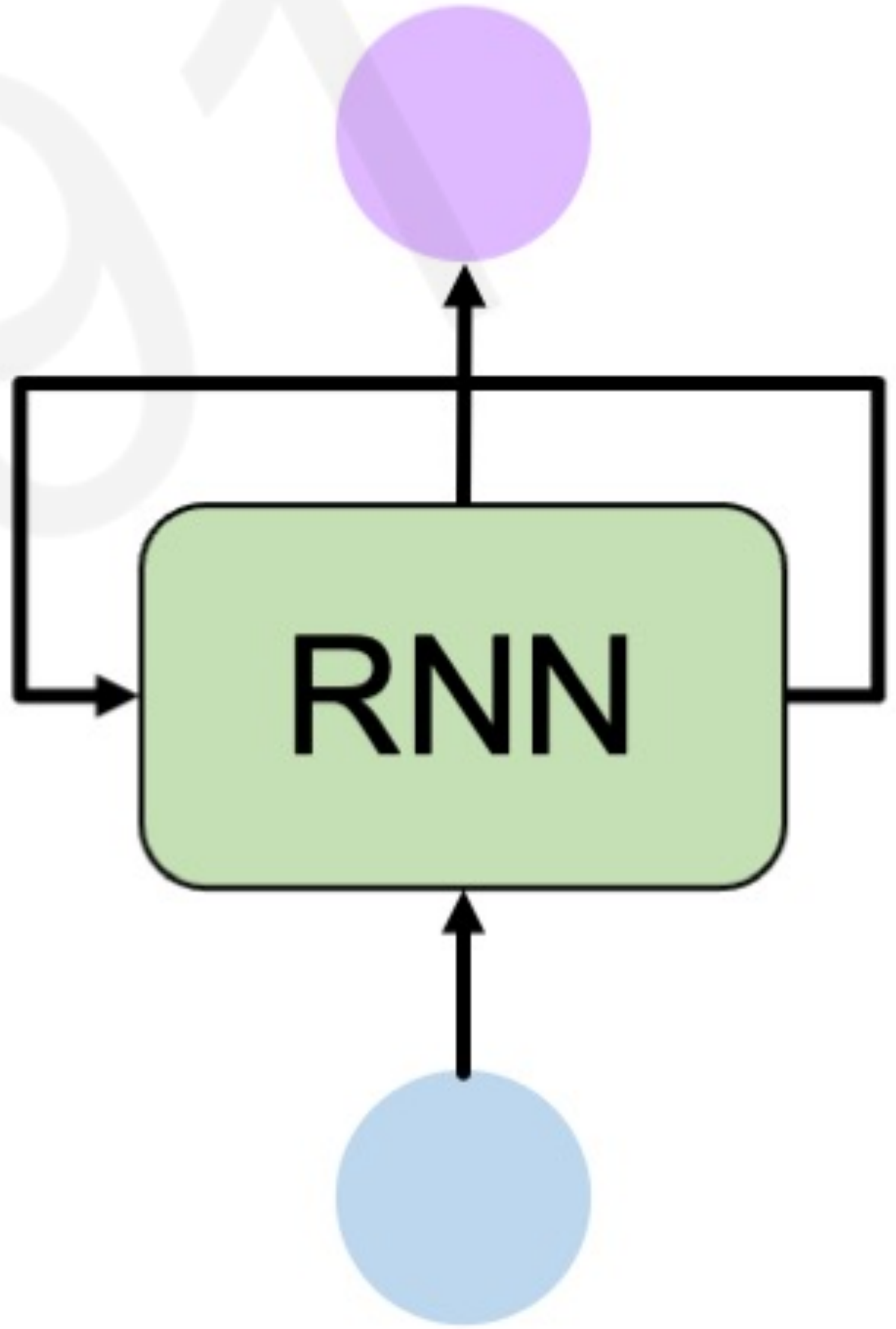
vs.

The food was bad, not good at all.

Sequence Modeling: Design Criteria

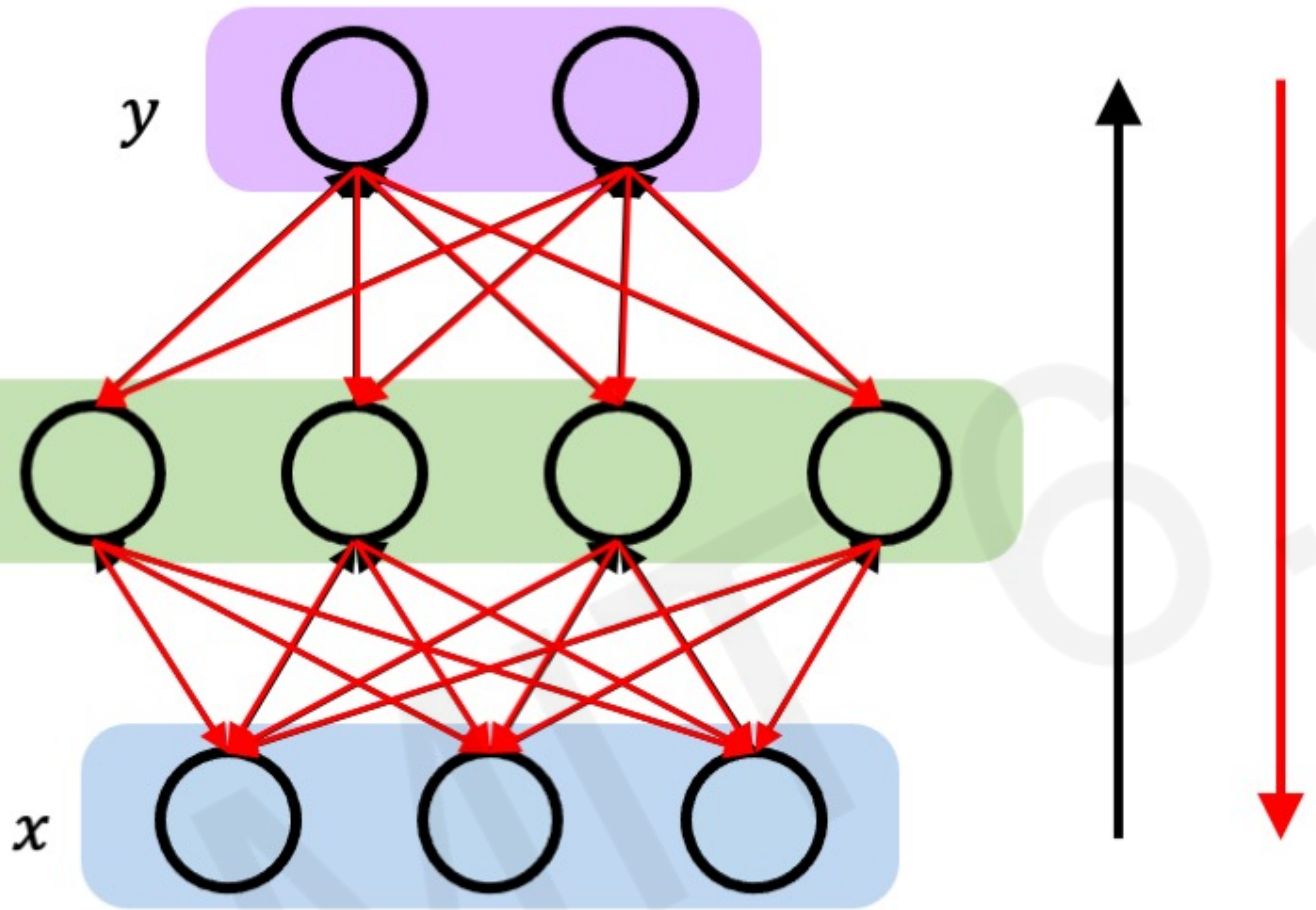
To model sequences, we need to:

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

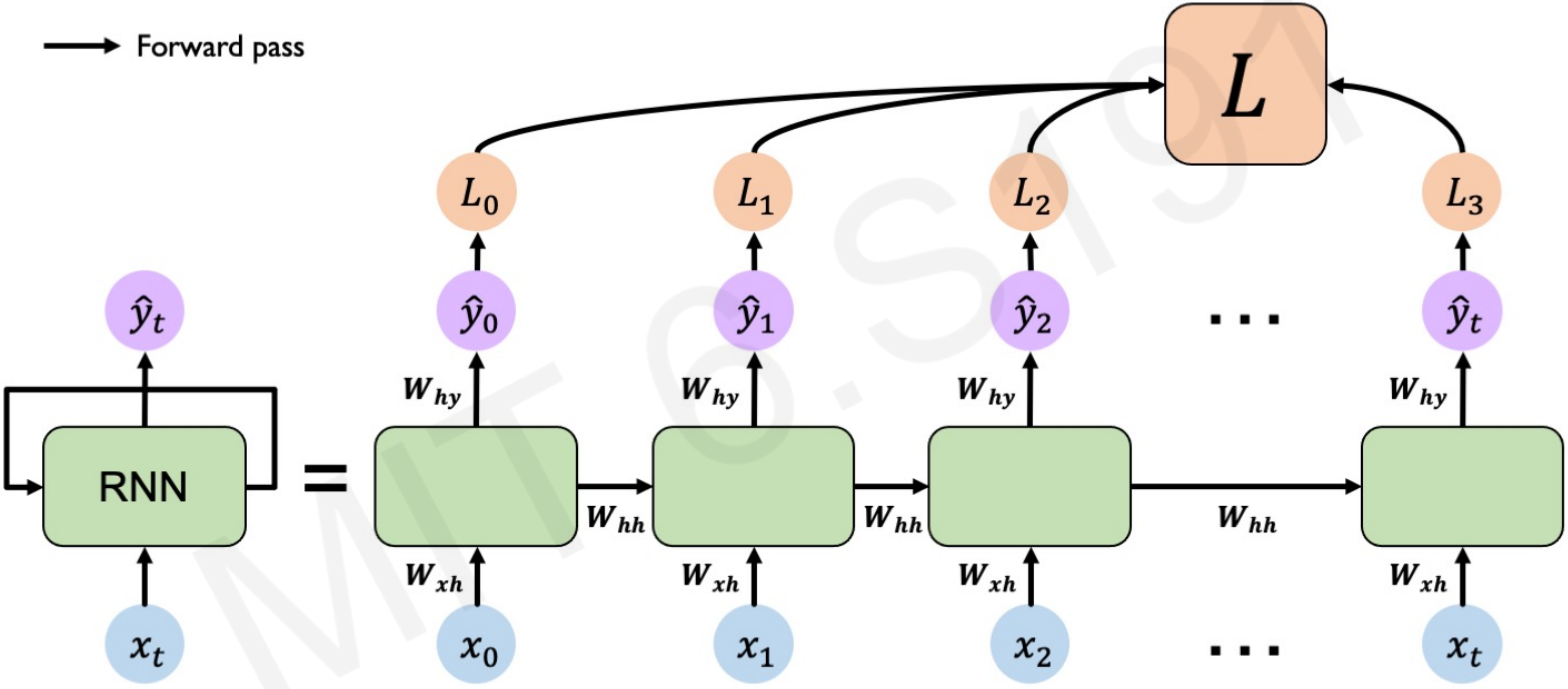
Recall: Backpropagation in Feed Forward Models



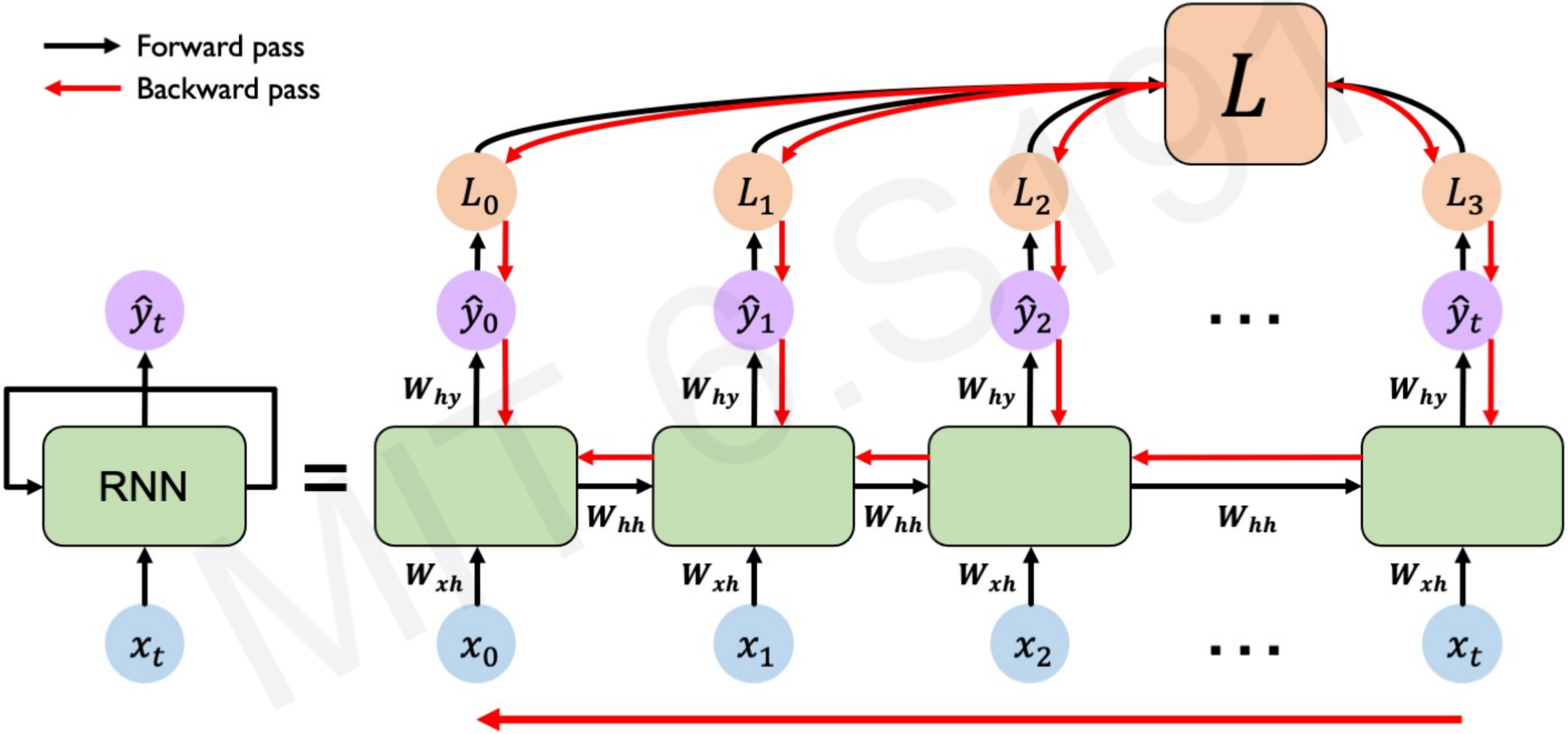
Backpropagation algorithm:

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

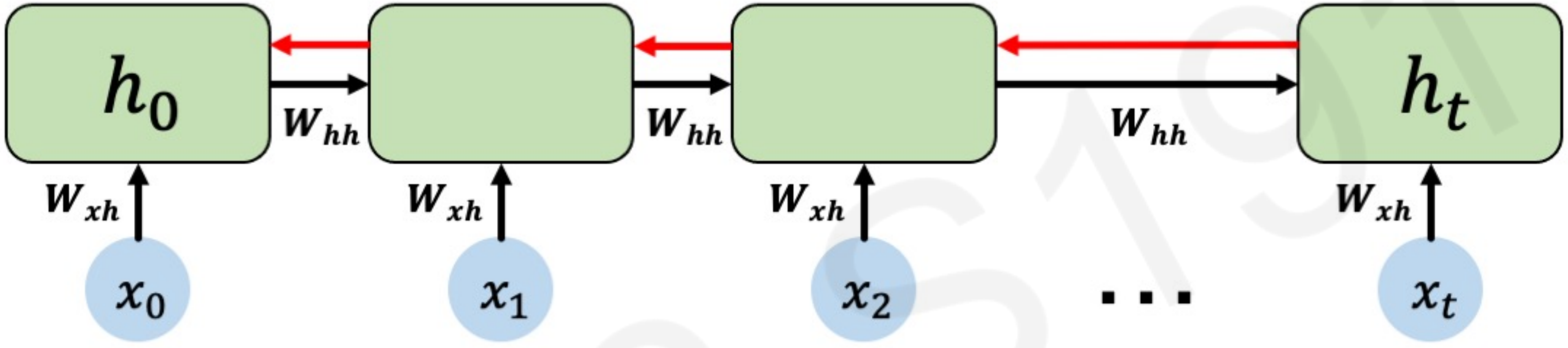
RNNs: Backpropagation Through Time



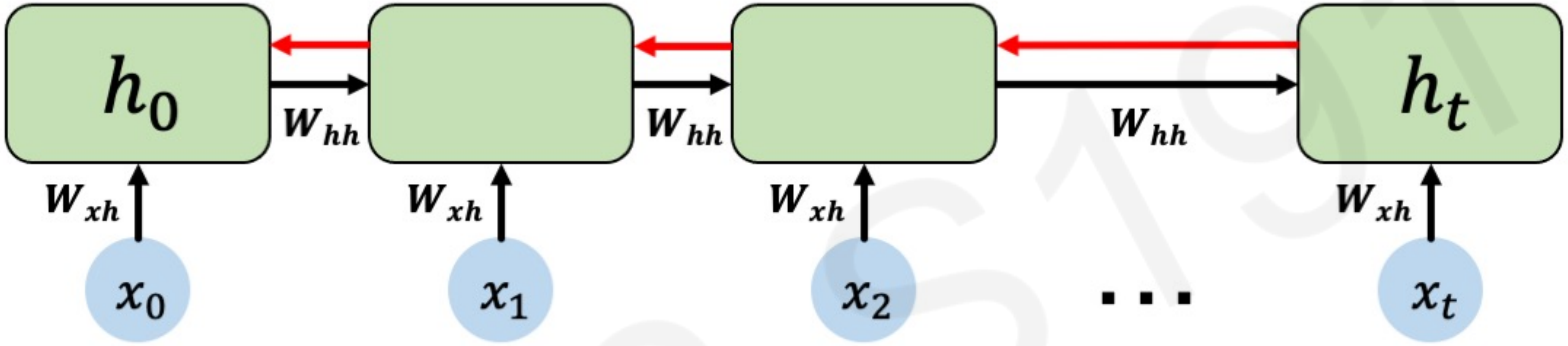
RNNs: Backpropagation Through Time



Standard RNN Gradient Flow

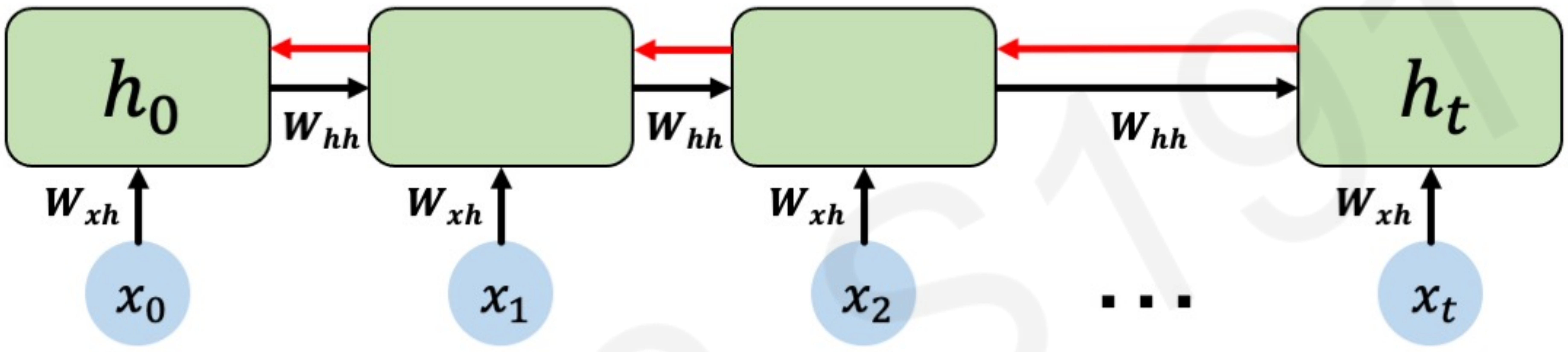


Standard RNN Gradient Flow



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

Standard RNN Gradient Flow: Exploding Gradients

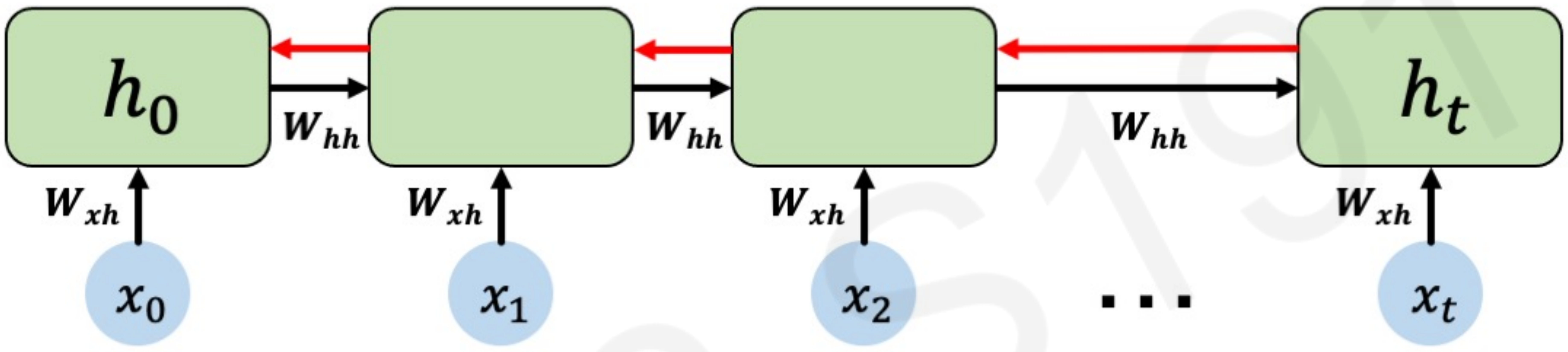


Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

Many values > 1:
exploding gradients

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?

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

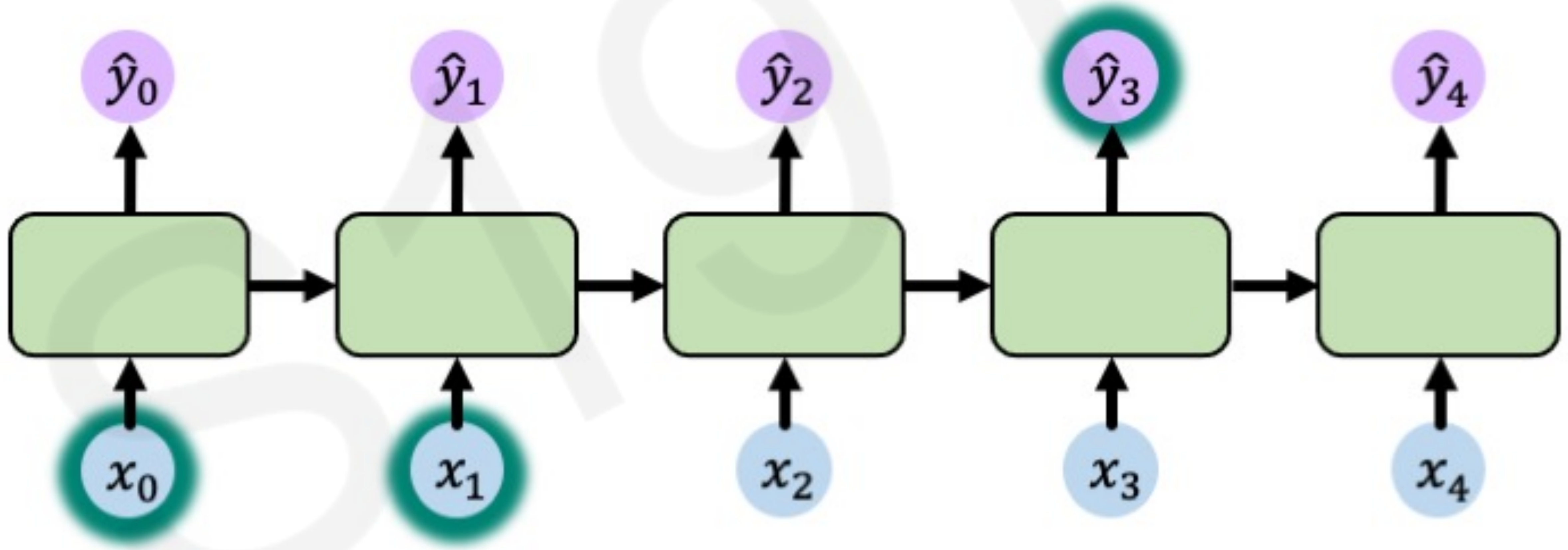
↓

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

↓

Bias parameters to capture short-term dependencies

"The clouds are in the ___"



The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

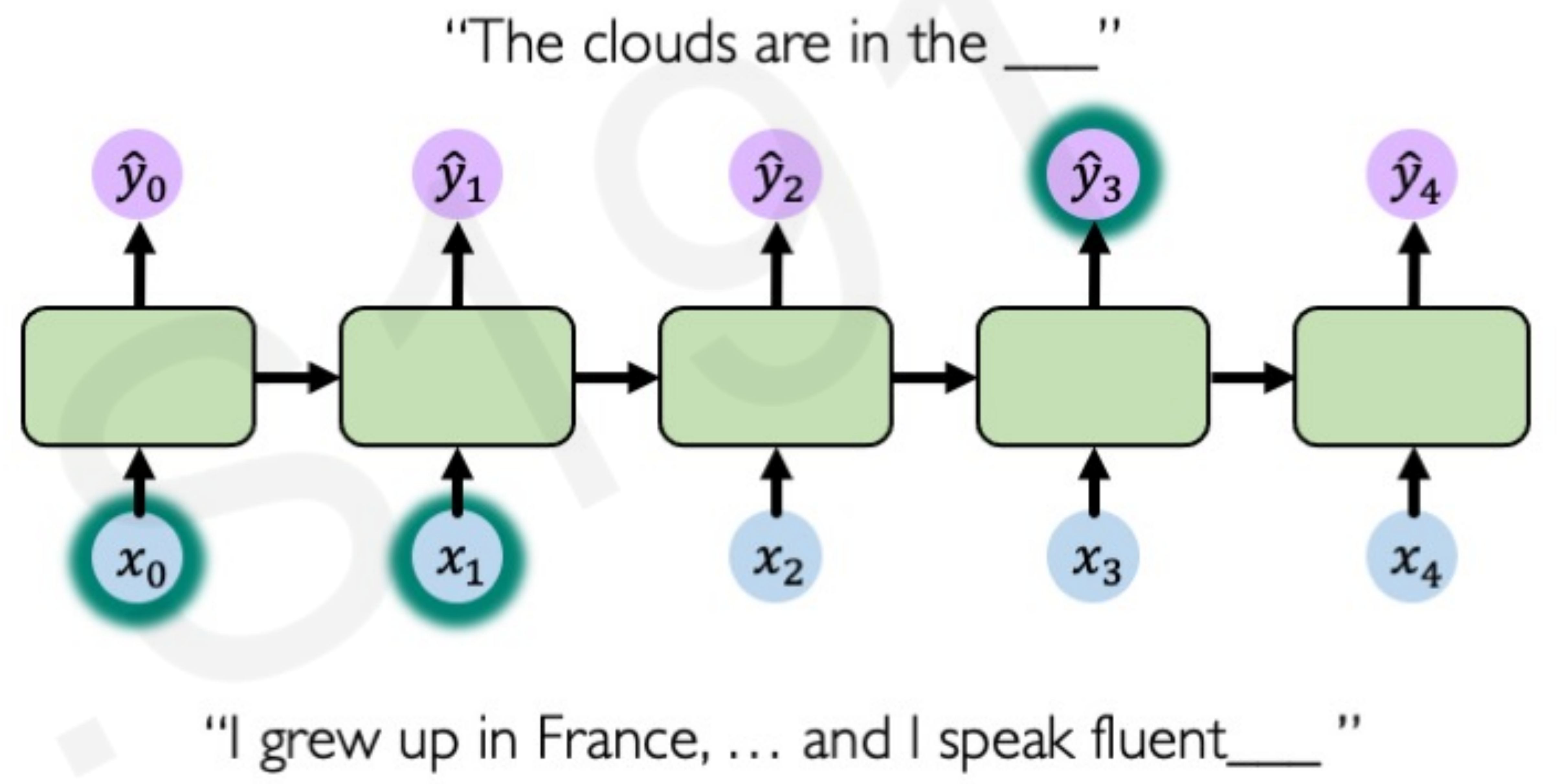
Multiply many **small numbers** together

↓

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?

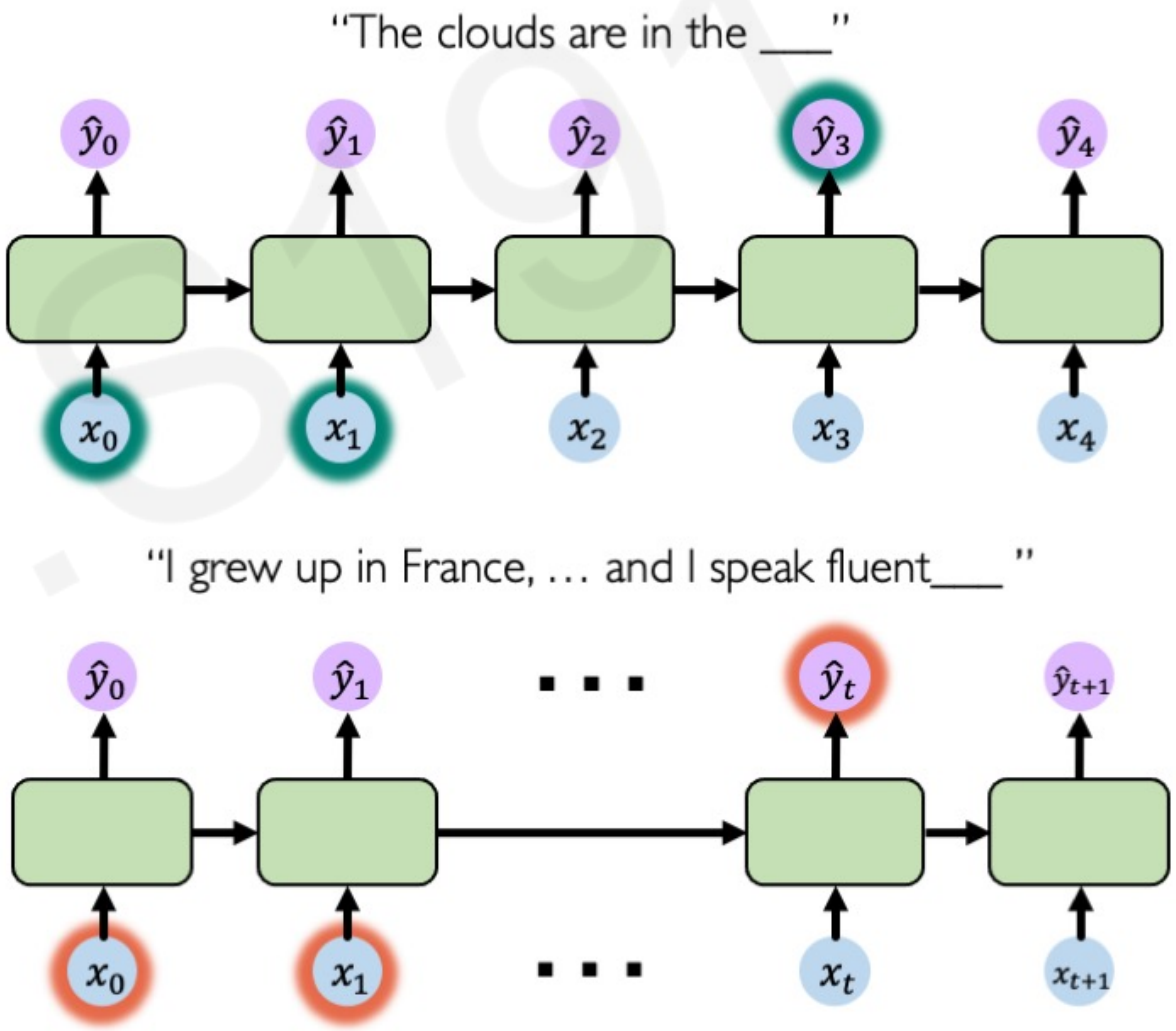
Multiply many **small numbers** together

↓

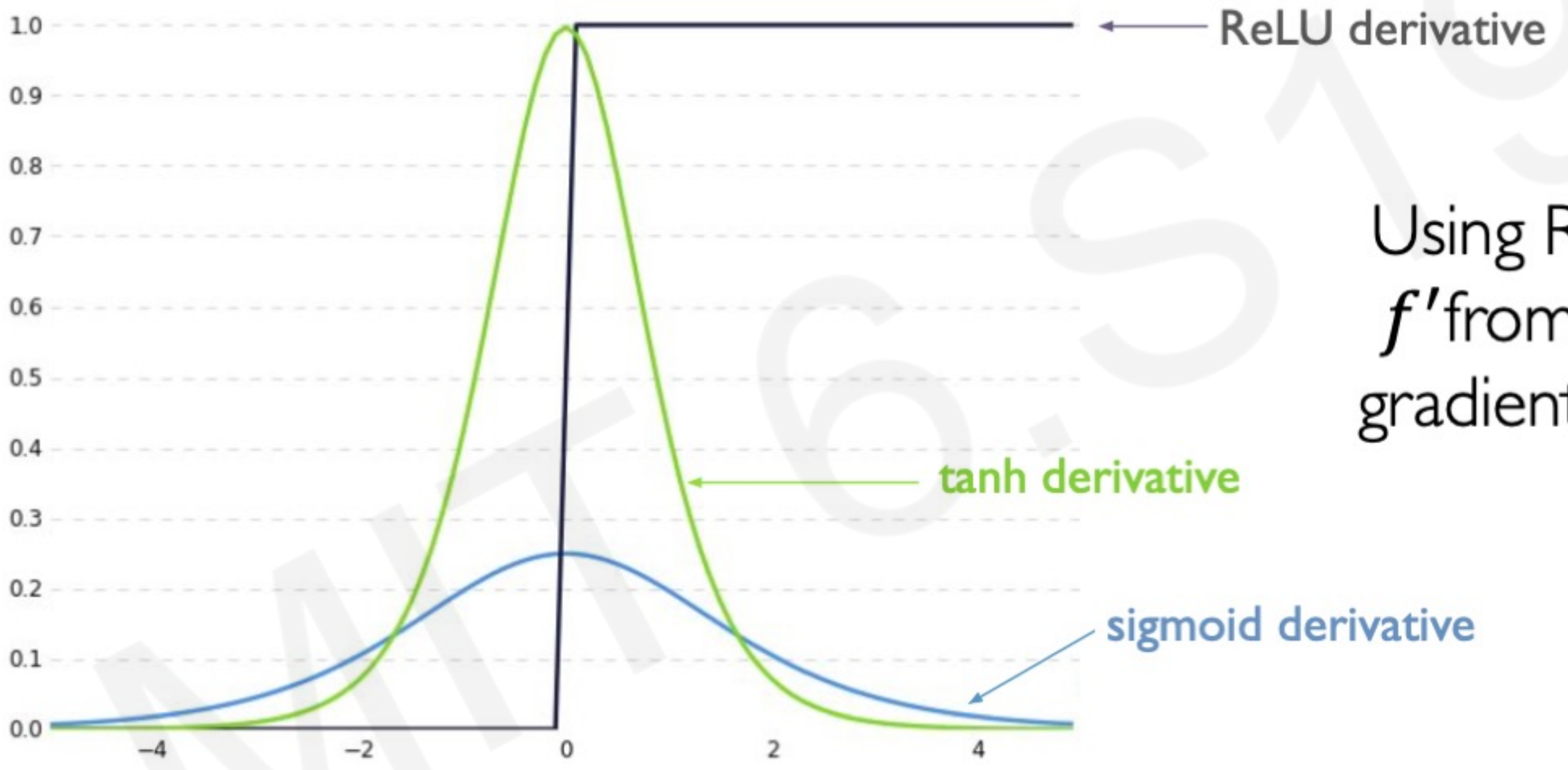
Errors due to further back time steps have smaller and smaller gradients

↓

Bias parameters to capture short-term dependencies



Trick #1: Activation Functions



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

Trick #2: Parameter Initialization

SKIP

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

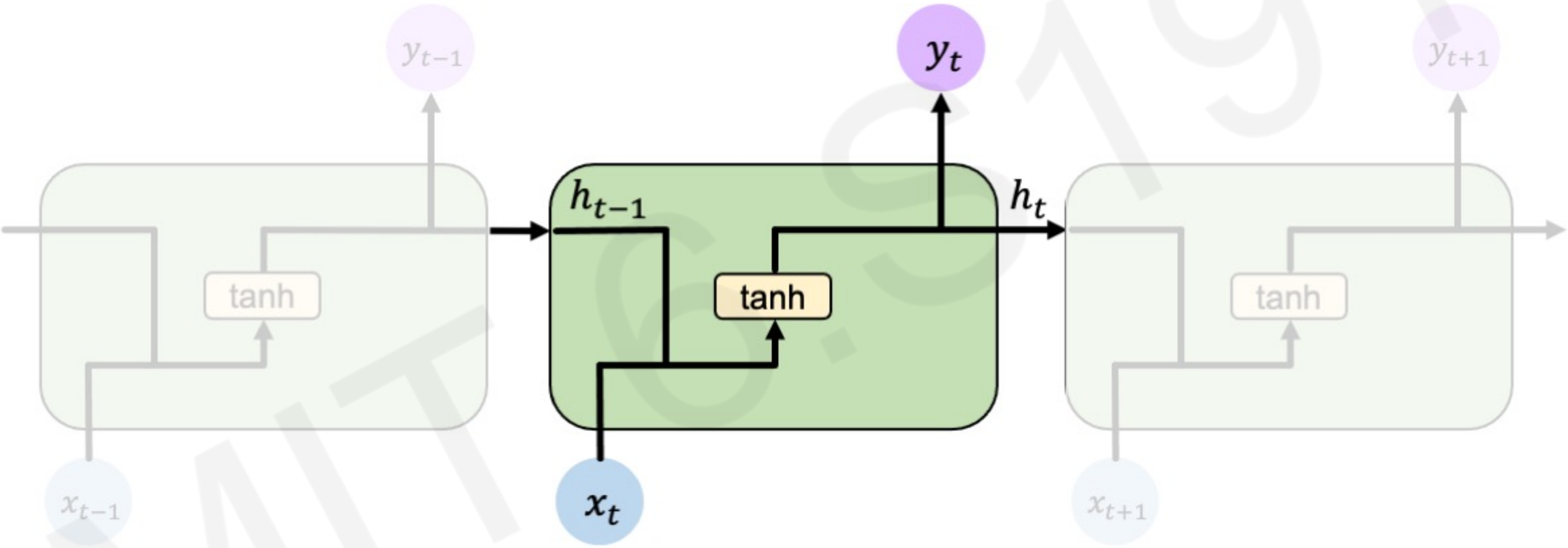
gated cell

LSTM, GRU, etc.

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

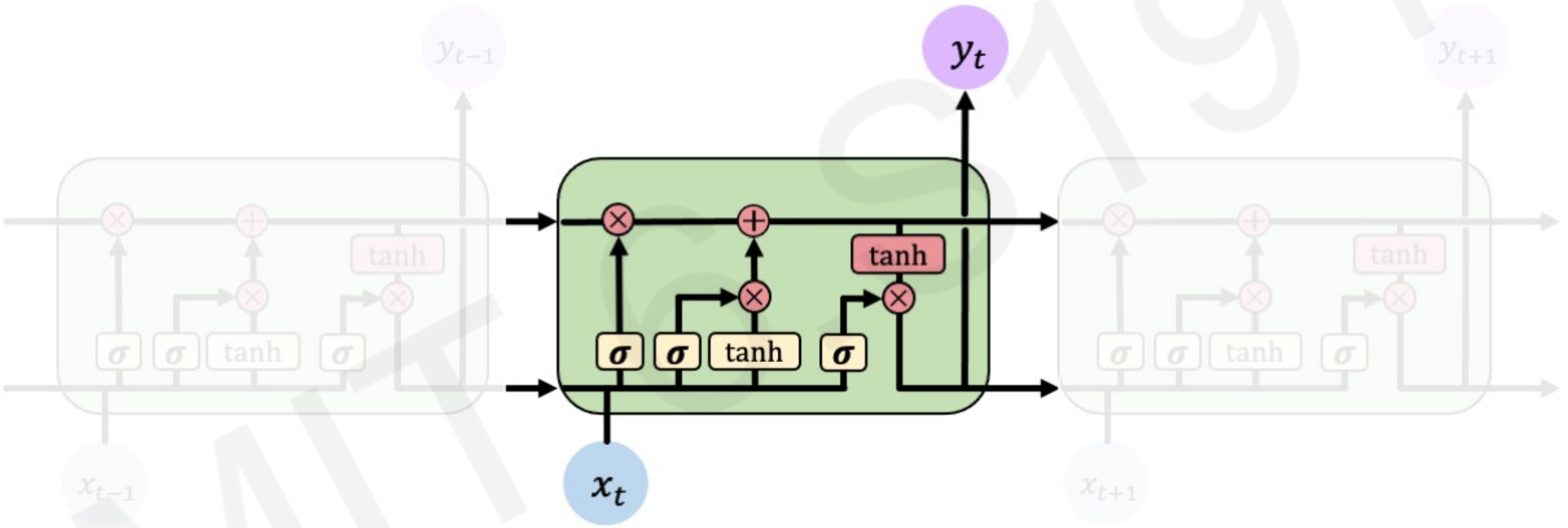
Standard RNN

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



Long Short Term Memory (LSTMs)

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

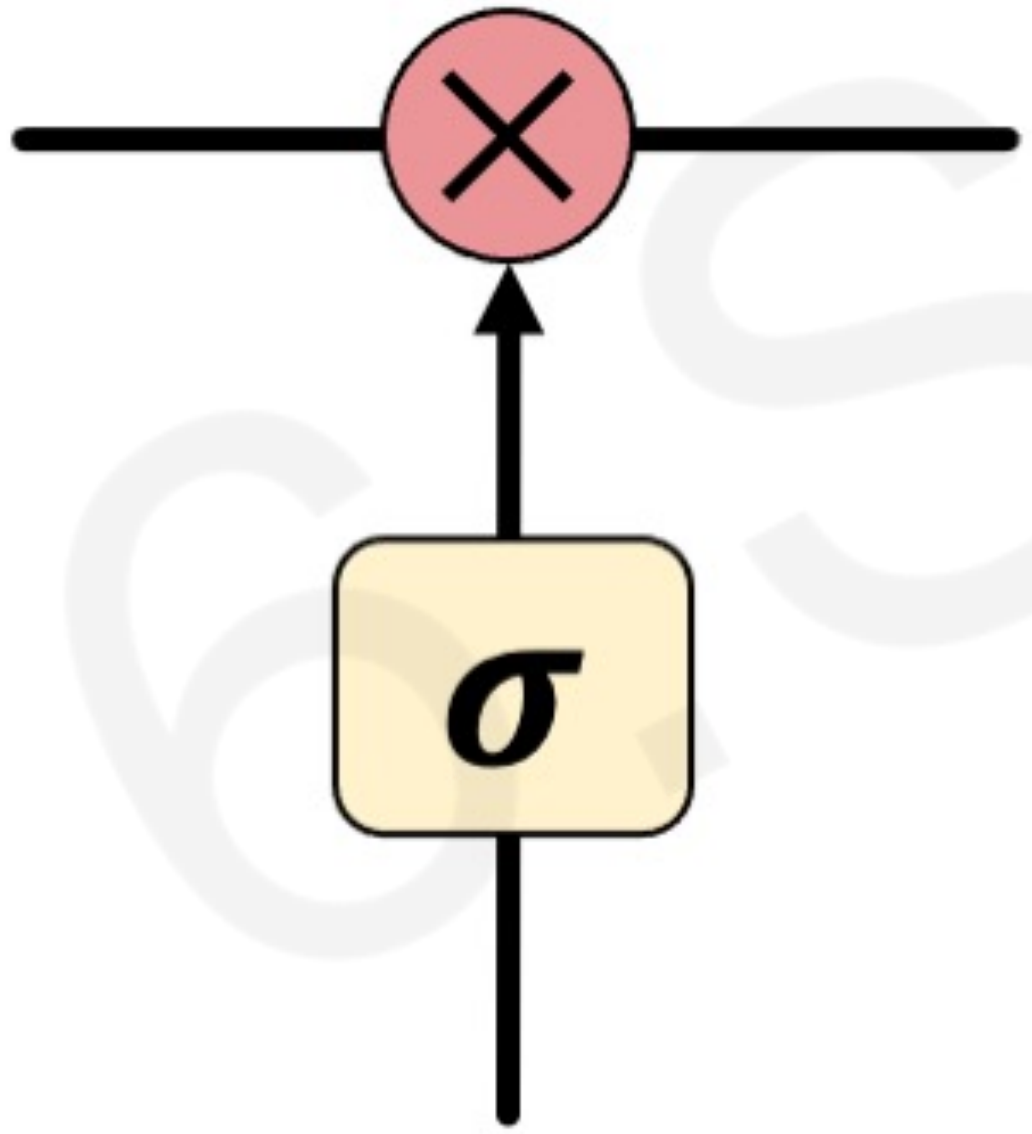


LSTM cells are able to track information throughout many timesteps

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

Long Short Term Memory (LSTMs)

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

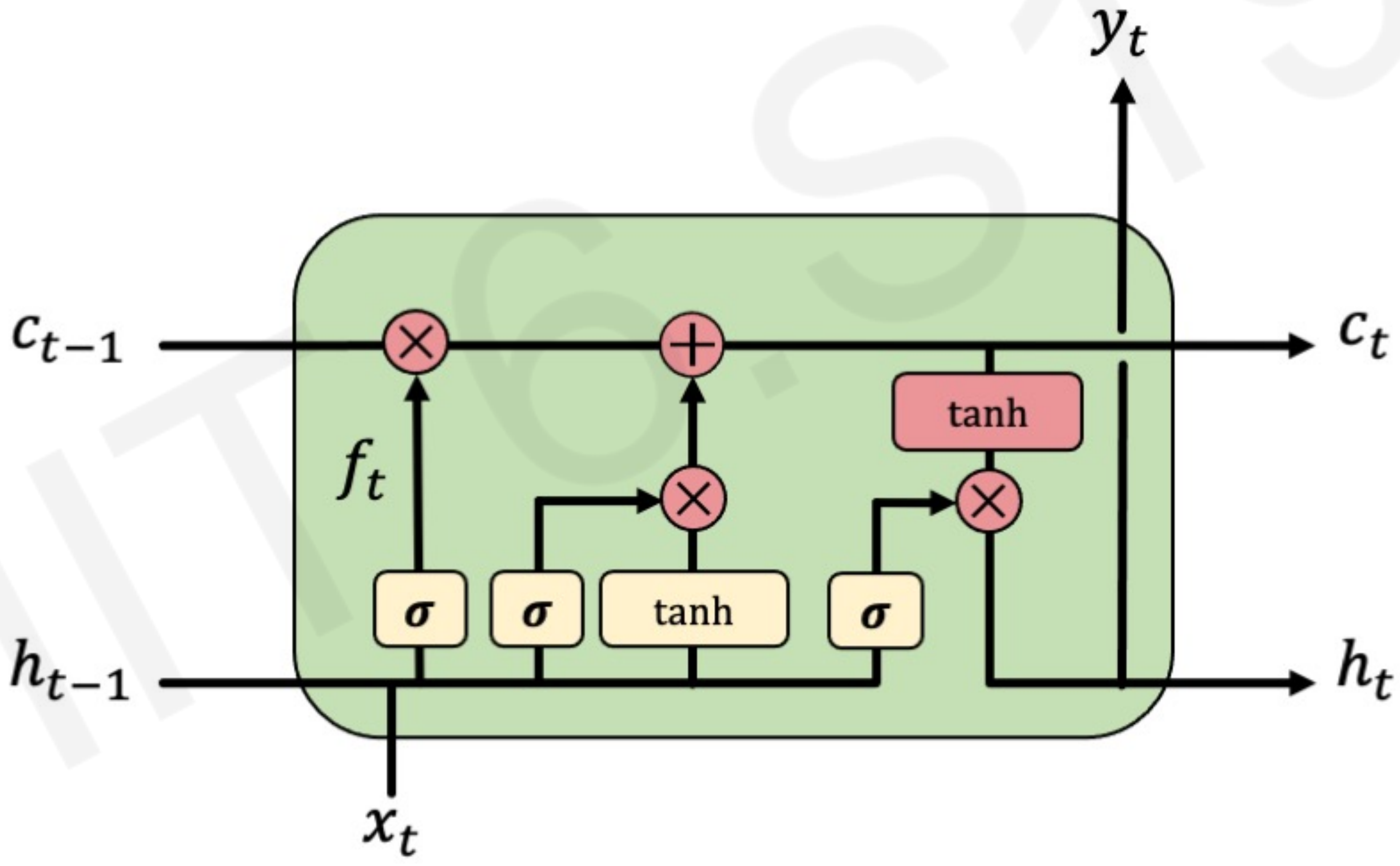


Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

Long Short Term Memory (LSTMs)

How do LSTMs work?

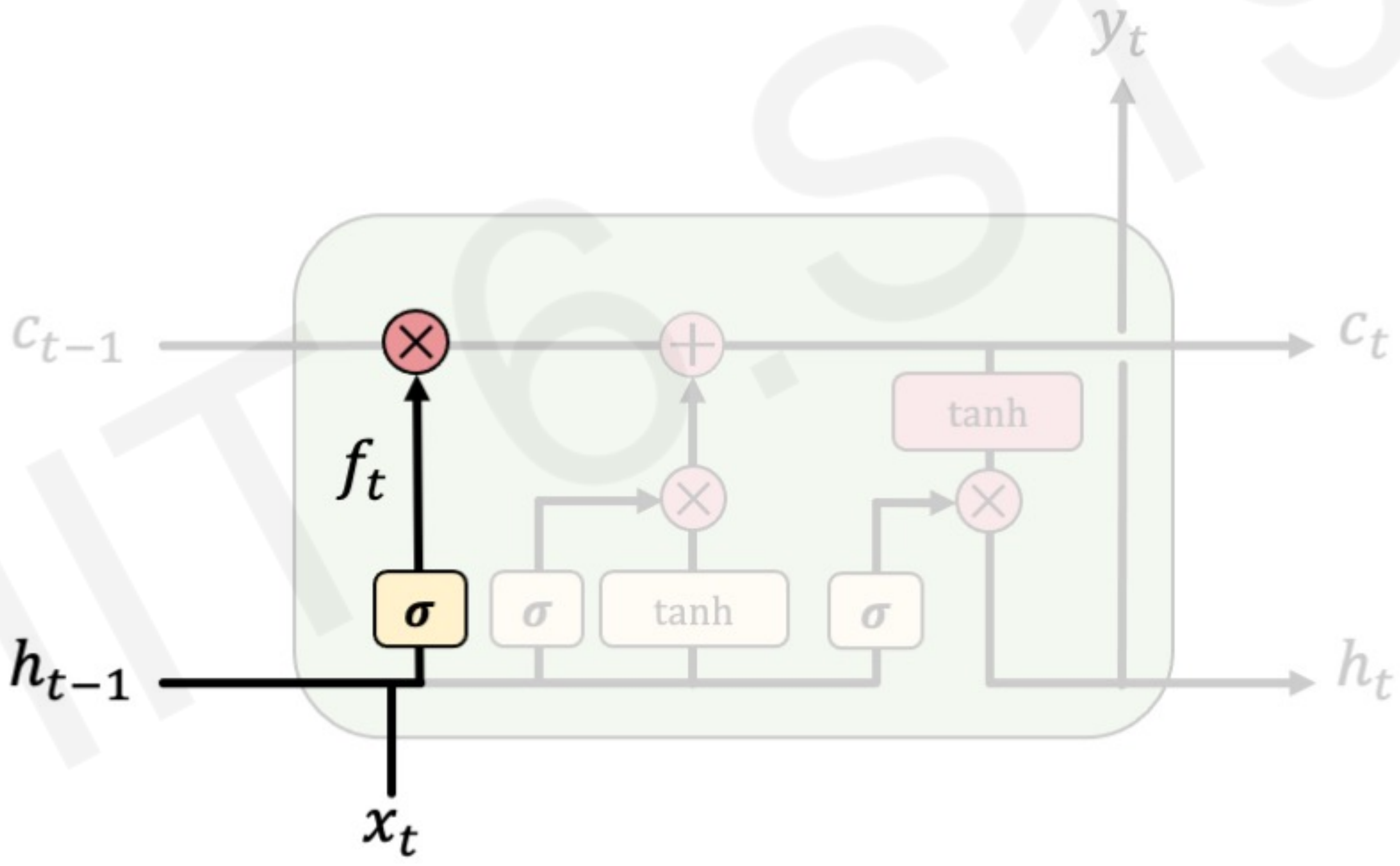
1) Forget 2) Store 3) Update 4) Output



Long Short Term Memory (LSTMs)

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

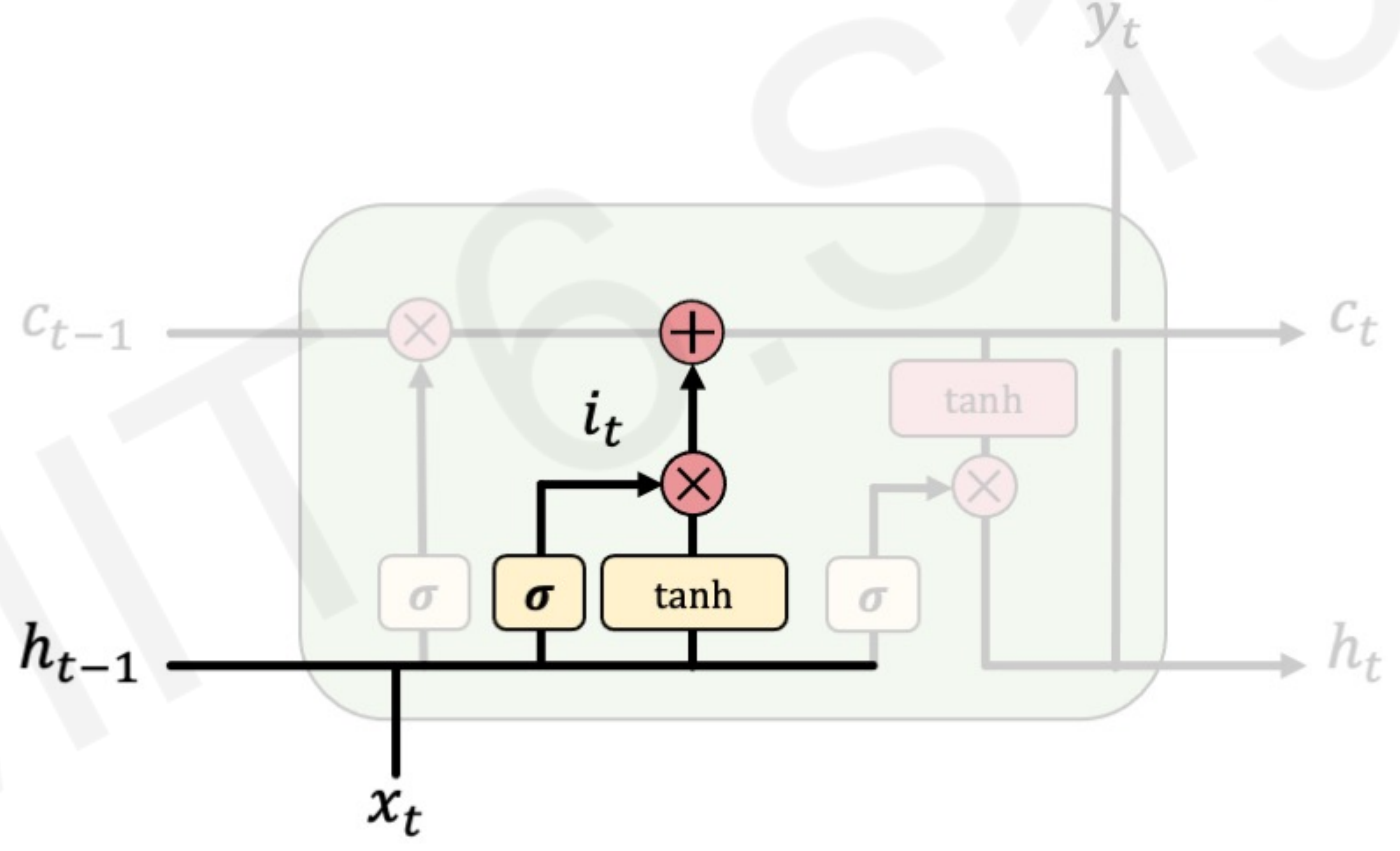
LSTMs **forget irrelevant** parts of the previous state



Long Short Term Memory (LSTMs)

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

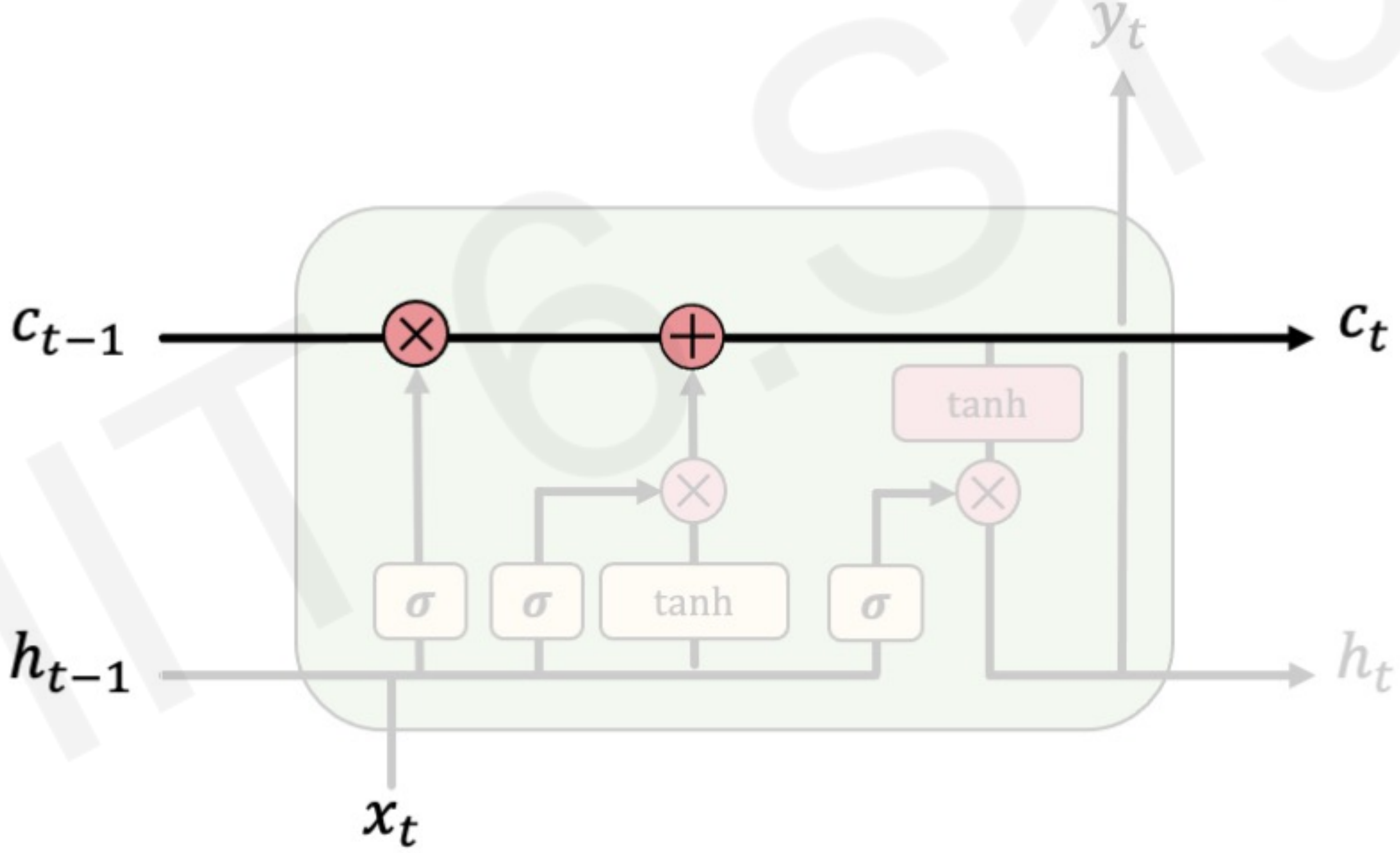
LSTMs **store relevant** new information into the cell state



Long Short Term Memory (LSTMs)

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

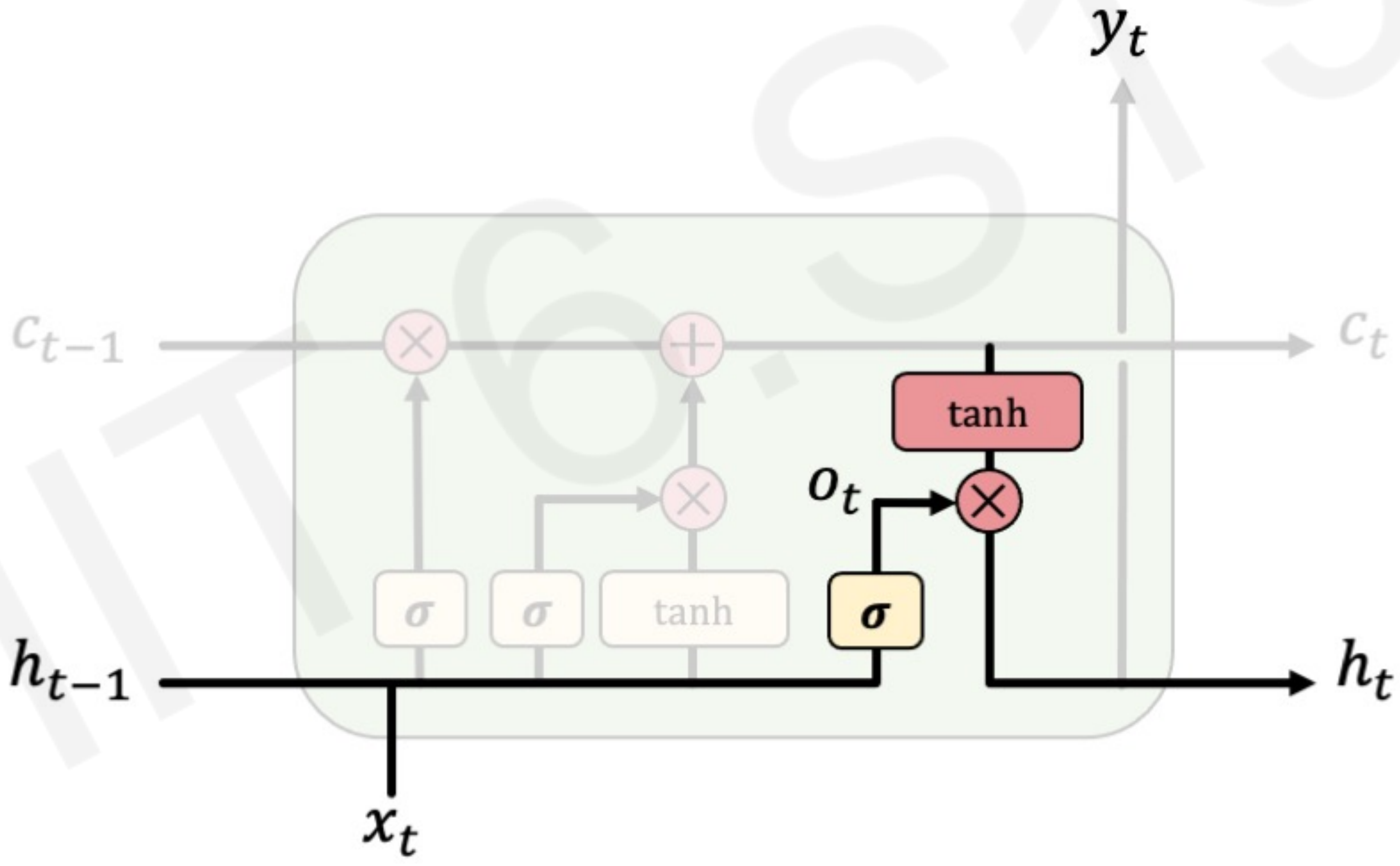
LSTMs **selectively update** cell state values



Long Short Term Memory (LSTMs)

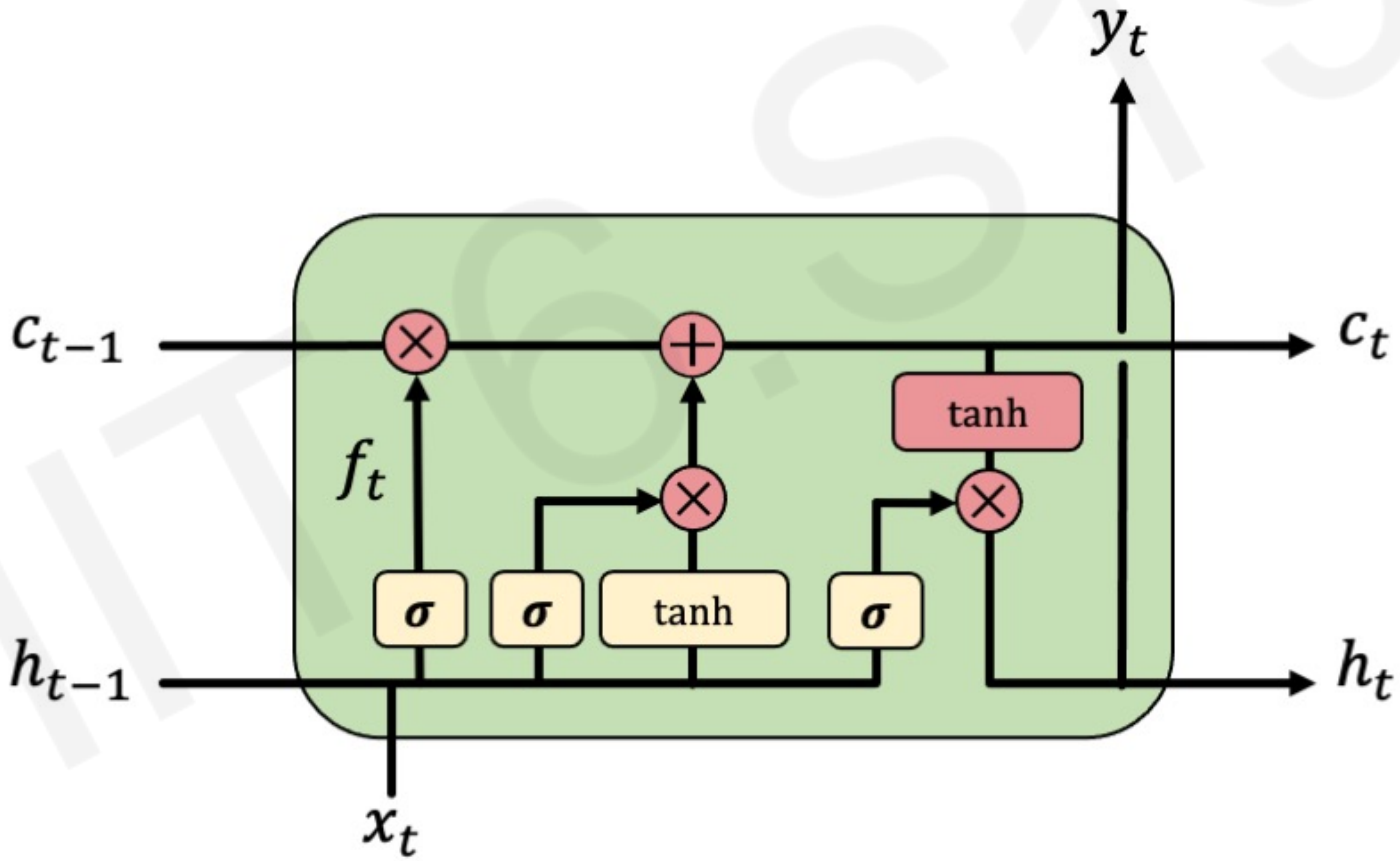
1) Forget 2) Store 3) Update **4) Output**

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



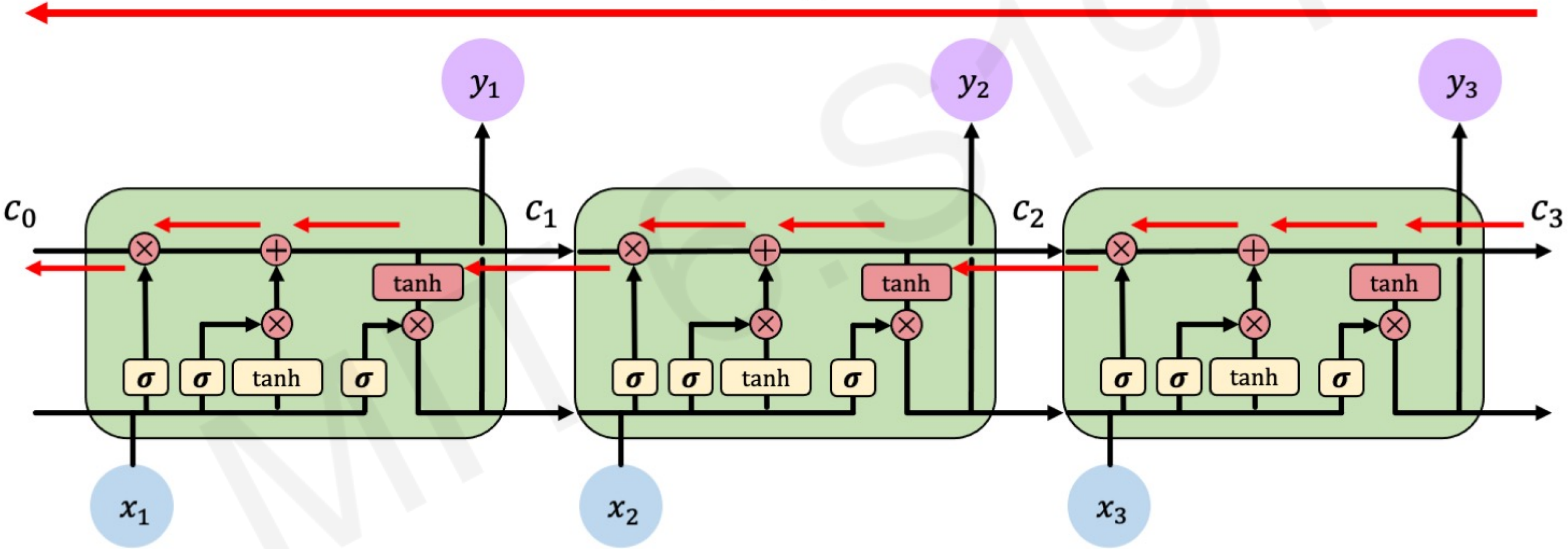
Long Short Term Memory (LSTMs)

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



LSTM Gradient Flow

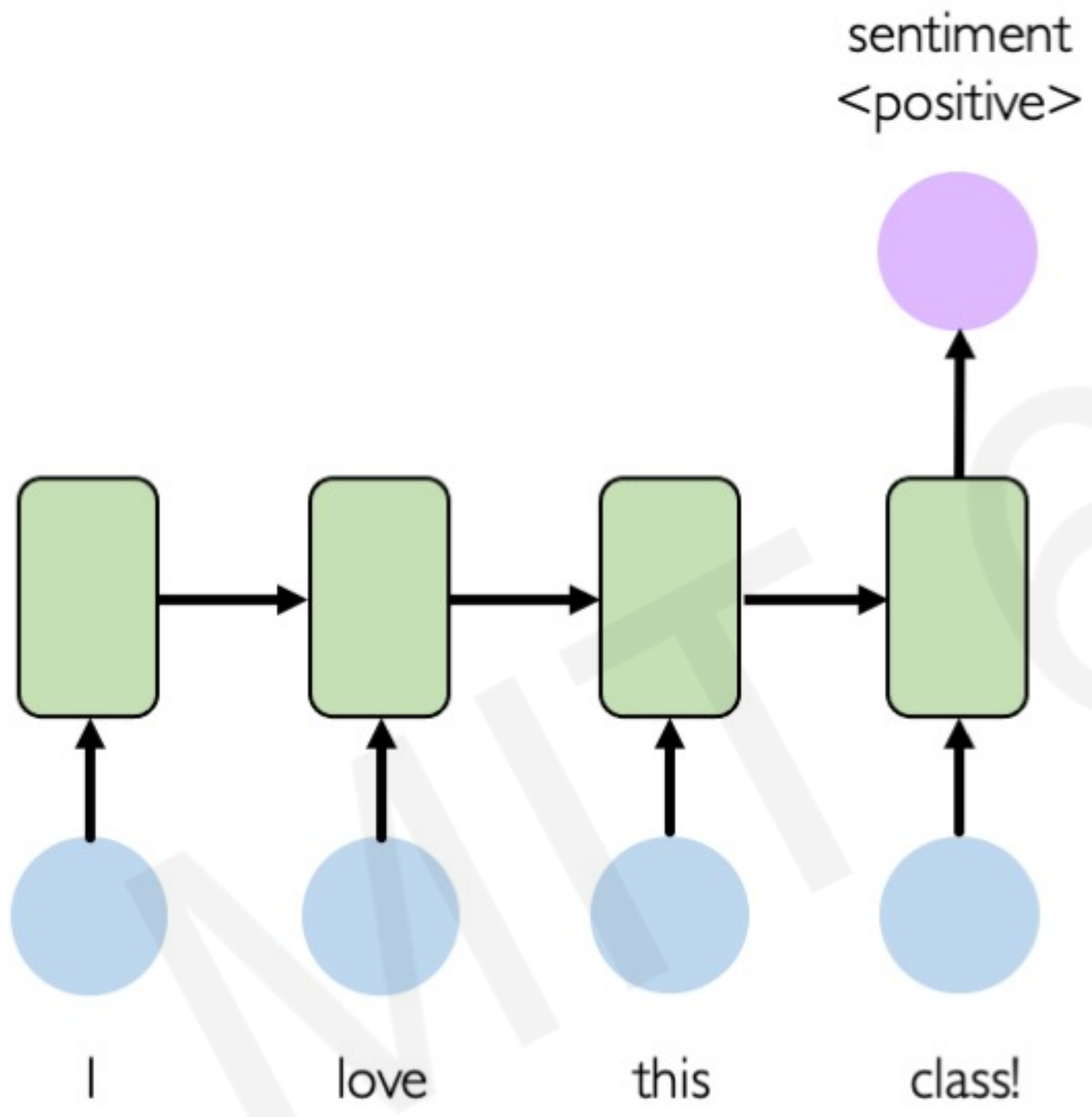
Uninterrupted gradient flow!



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**

Example Task: Sentiment Classification

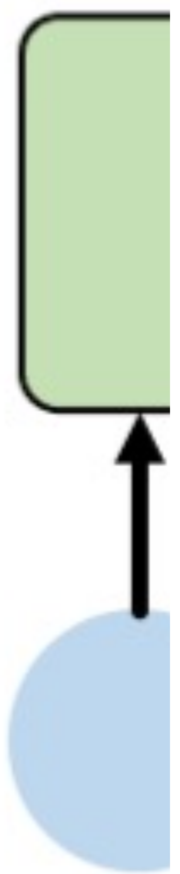


Tweet sentiment classification

Example Task: Sentiment Classification

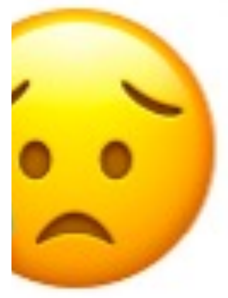
```
[ ] inputs = keras.Input(shape=(None,), dtype="int32")
    x = layers.Embedding(max_features, 16)(inputs) # define a 16-dimensional Embedding layer that acts on "input:
    x = layers.LSTM(16)(x) # Add a 16-node LSTM that acts on the output of the Embedding layer
    outputs = layers.Dense(1, activation="sigmoid")(x) # define a 1-node sigmoid / classifier layer that acts on
    model = keras.Model(inputs, outputs) # define the model as inputs -> outputs
    model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding_1 (Embedding)      (None, None, 16)          80000
_________________________________________________________________
lstm_1 (LSTM)                (None, 16)                2112
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 82,129
Trainable params: 82,129
Non-trainable params: 0
_________________________________________________________________
```



I love this class!

is
nd



e
this

winter! :(

2:19 AM - 25 Jan 2019

Example Task: Sentiment Classification

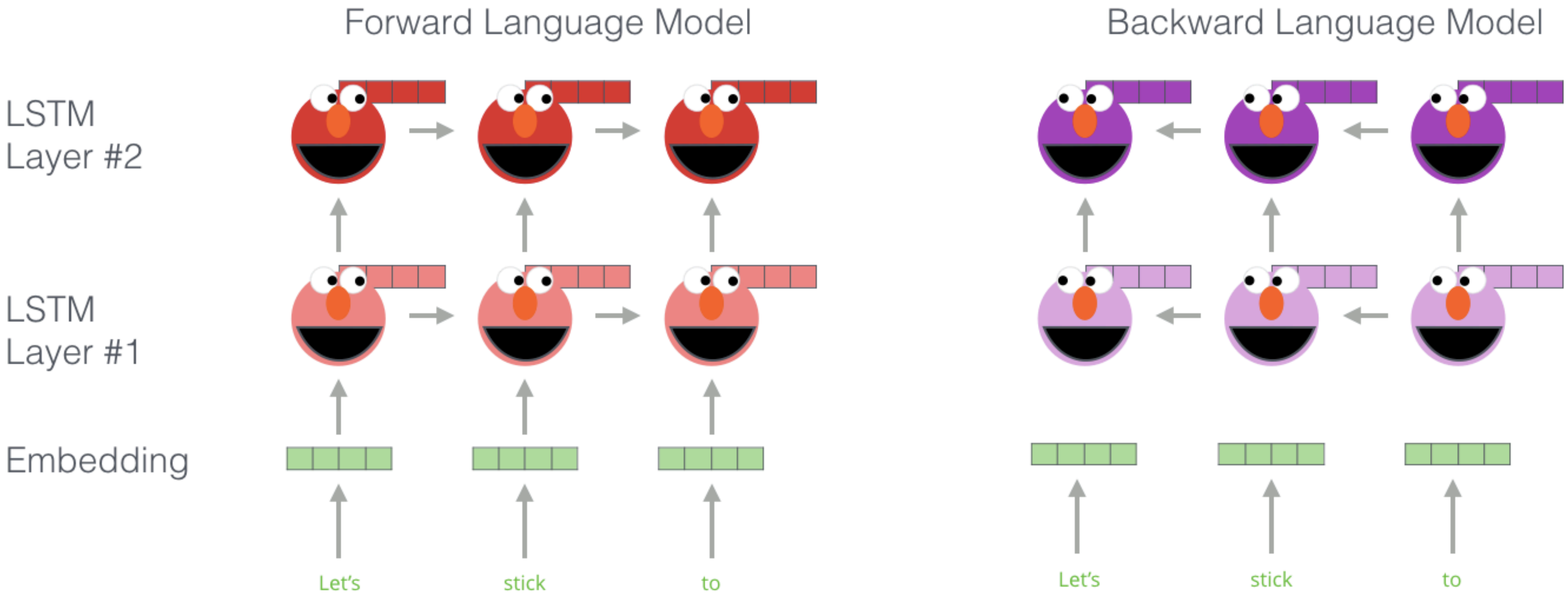
```
[ ] inputs = keras.Input(shape=(None,), dtype="int32")
    x = layers.Embedding(max_features, 16)(inputs) # define a 16-dimensional Embedding layer that acts on "input:
    x = layers.Bidirectional(layers.LSTM(16))(x) # Add a 16-node bi-LSTM that acts on the output of the Embeddin
    outputs = layers.Dense(1, activation="sigmoid")(x) # define a 1-node sigmoid / classifier layer that acts on
    model = keras.Model(inputs, outputs) # define the model as inputs -> outputs
    model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding_2 (Embedding)      (None, None, 16)          80000
_________________________________________________________________
bidirectional (Bidirectional (None, 32)                4224
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 84,257
Trainable params: 84,257
Non-trainable params: 0
_________________________________________________________________
```

2:19 AM - 25 Jan 2019

One big bi-LSTM success was ELMo — contextual embeddings

Embedding of “stick” in “Let’s stick to” - Step #1



Embedding of "stick" in "Let's stick to" - Step #2

1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task

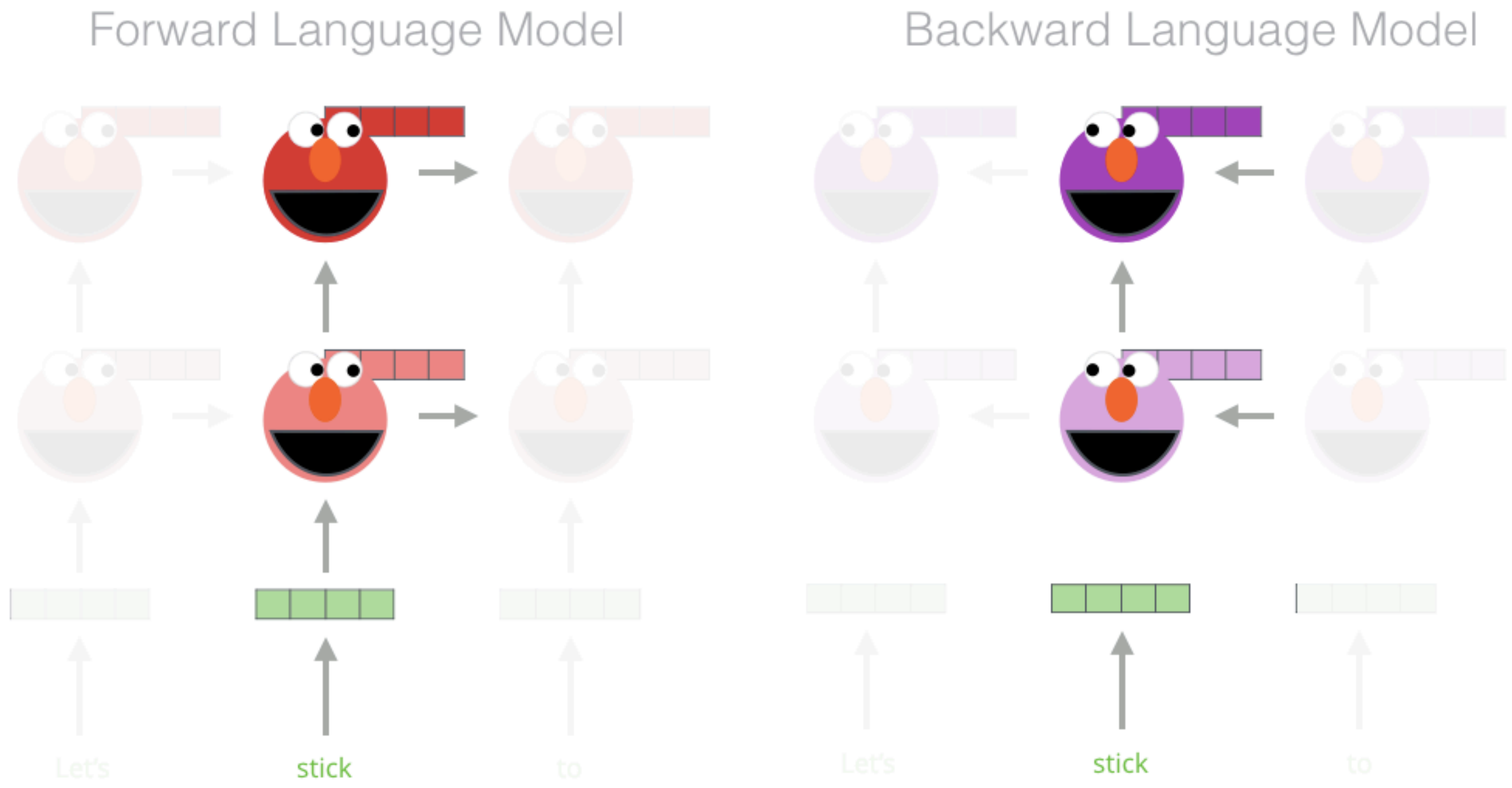
x s_2

x s_1

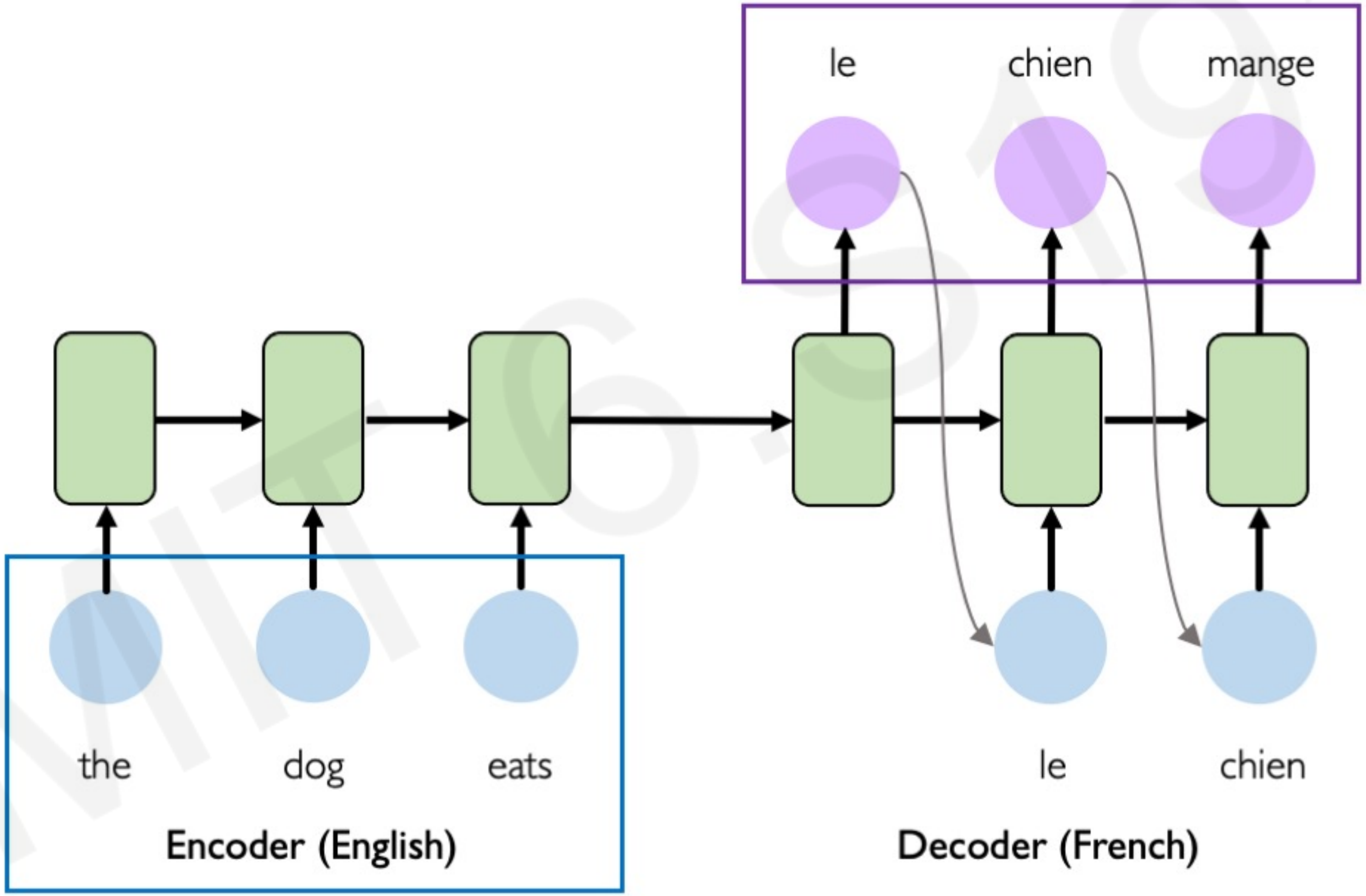
x s_0

3- Sum the (now weighted) vectors

ELMo embedding of "stick" for this task in this context

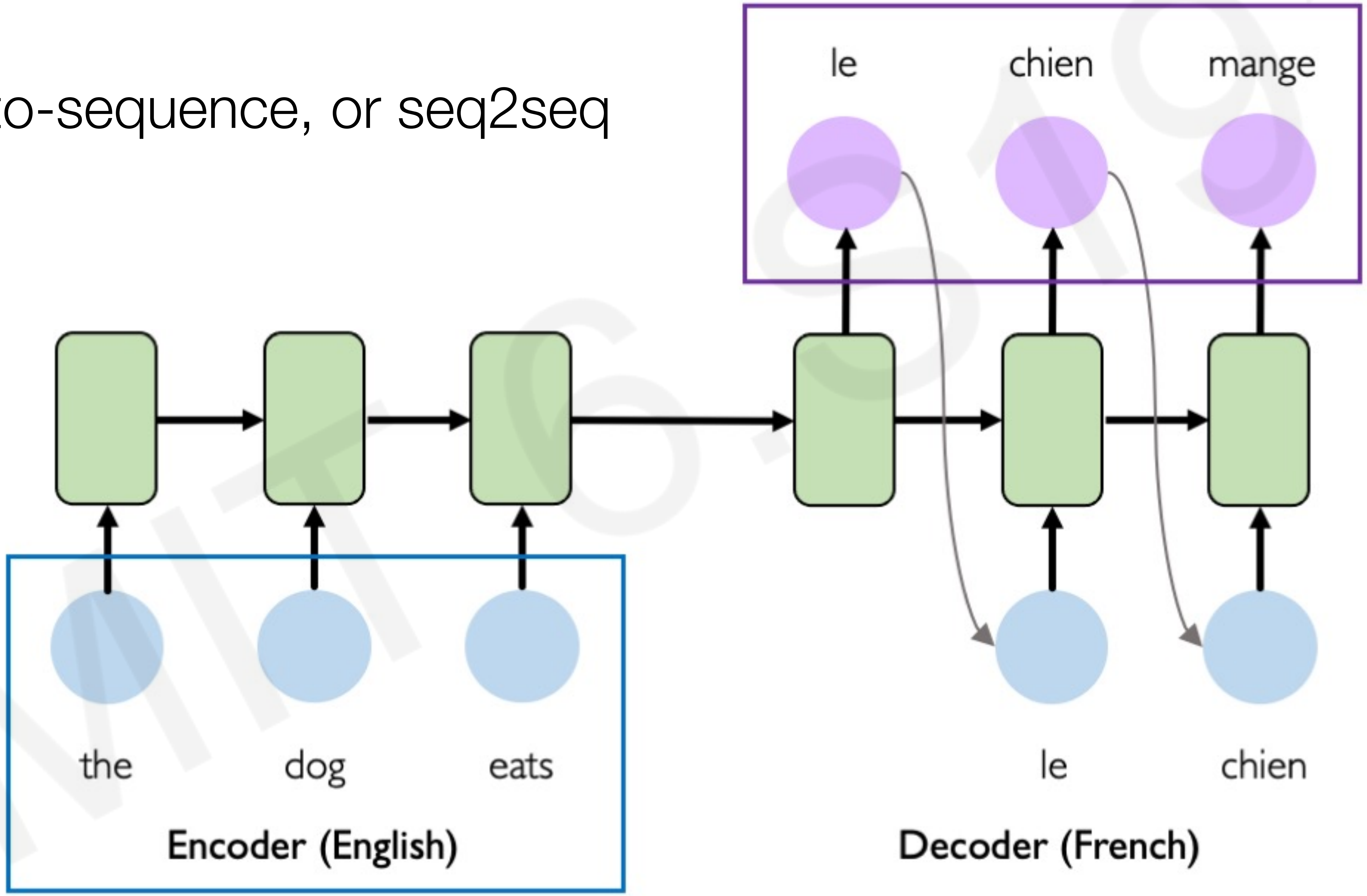


Example Task: Machine Translation



Example Task: Machine Translation

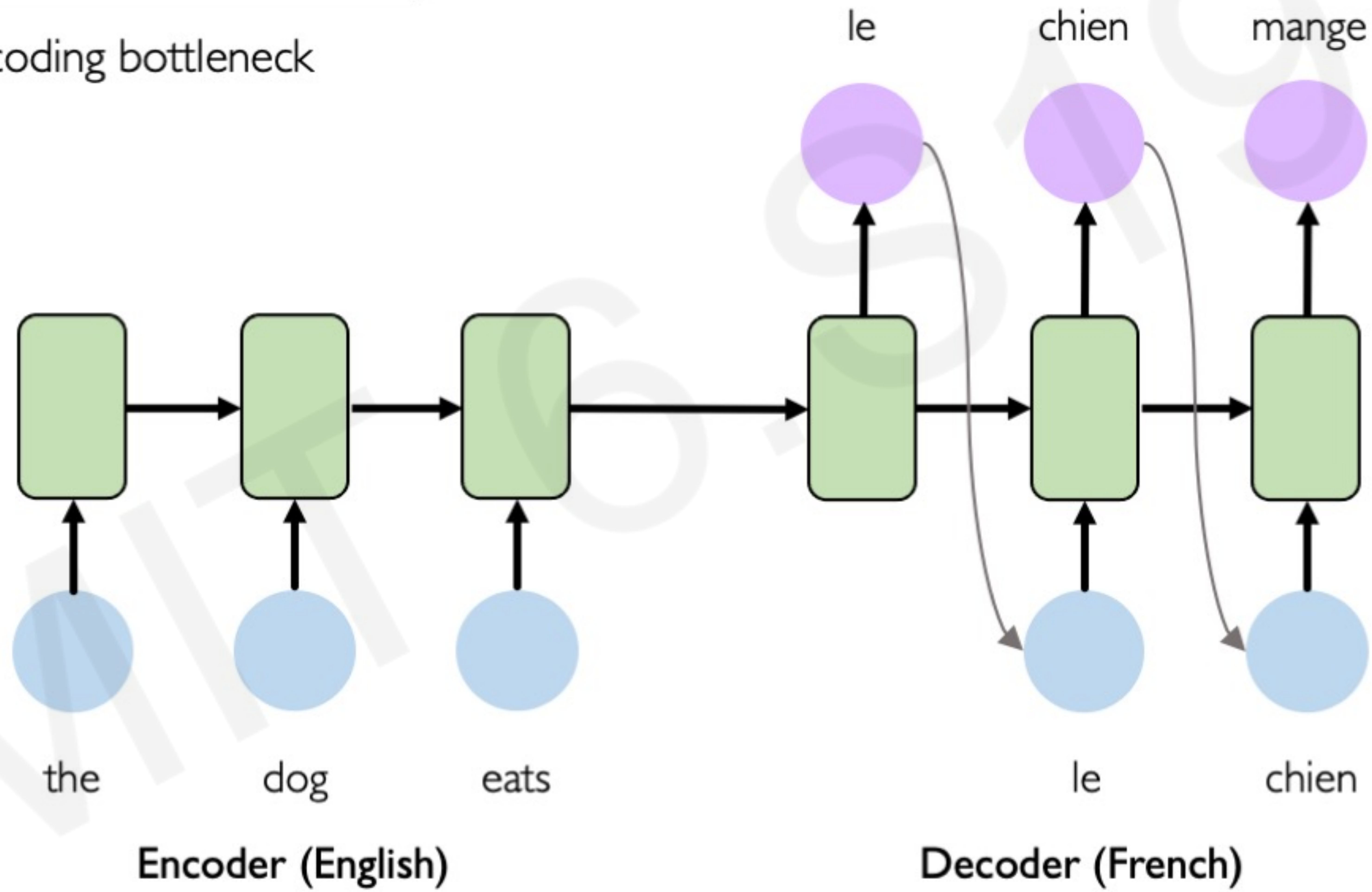
Sequence-to-sequence, or seq2seq



Example Task: Machine Translation

Potential Issues

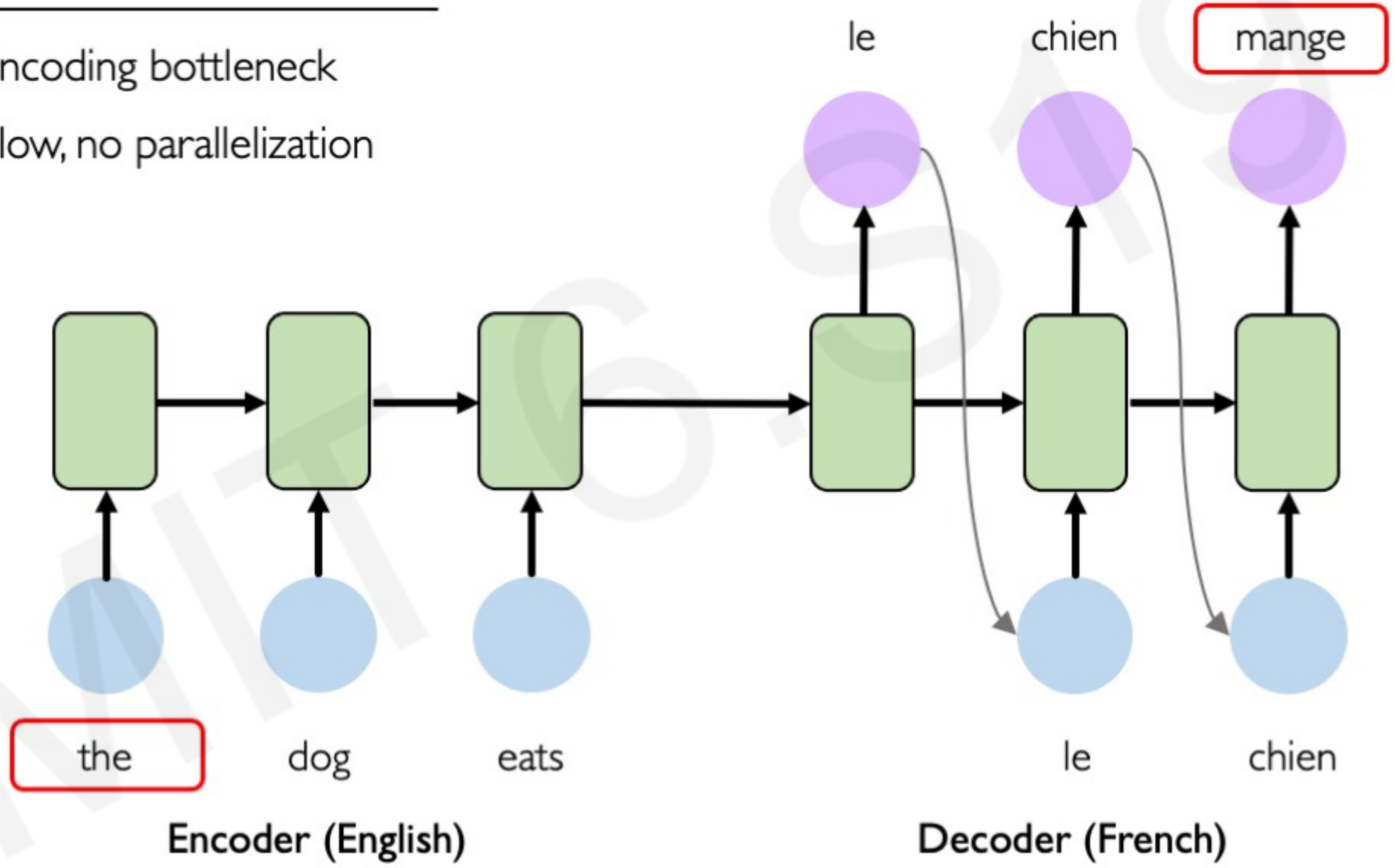
- Encoding bottleneck



Example Task: Machine Translation

Potential Issues

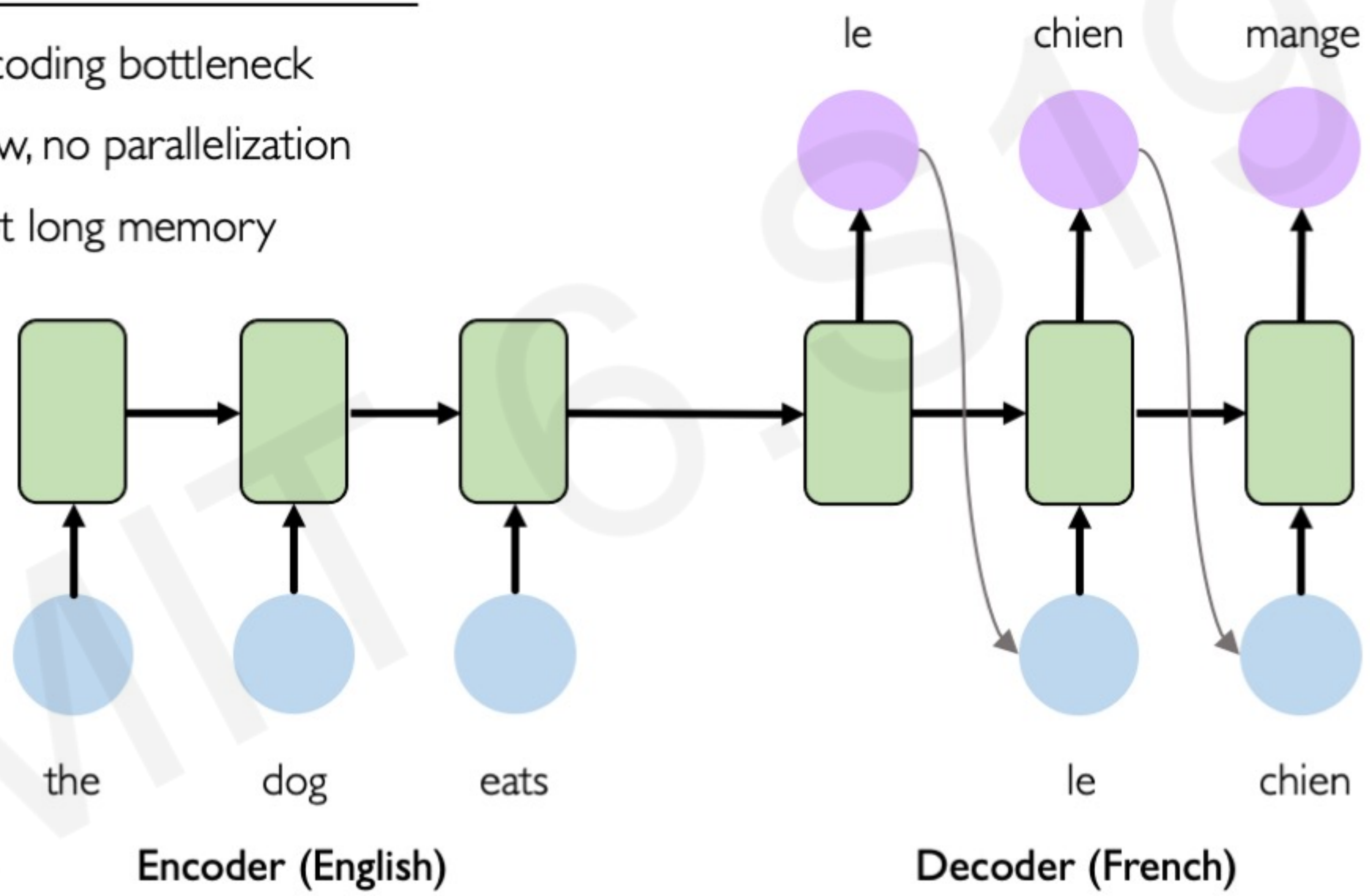
- Encoding bottleneck
- Slow, no parallelization



Example Task: Machine Translation

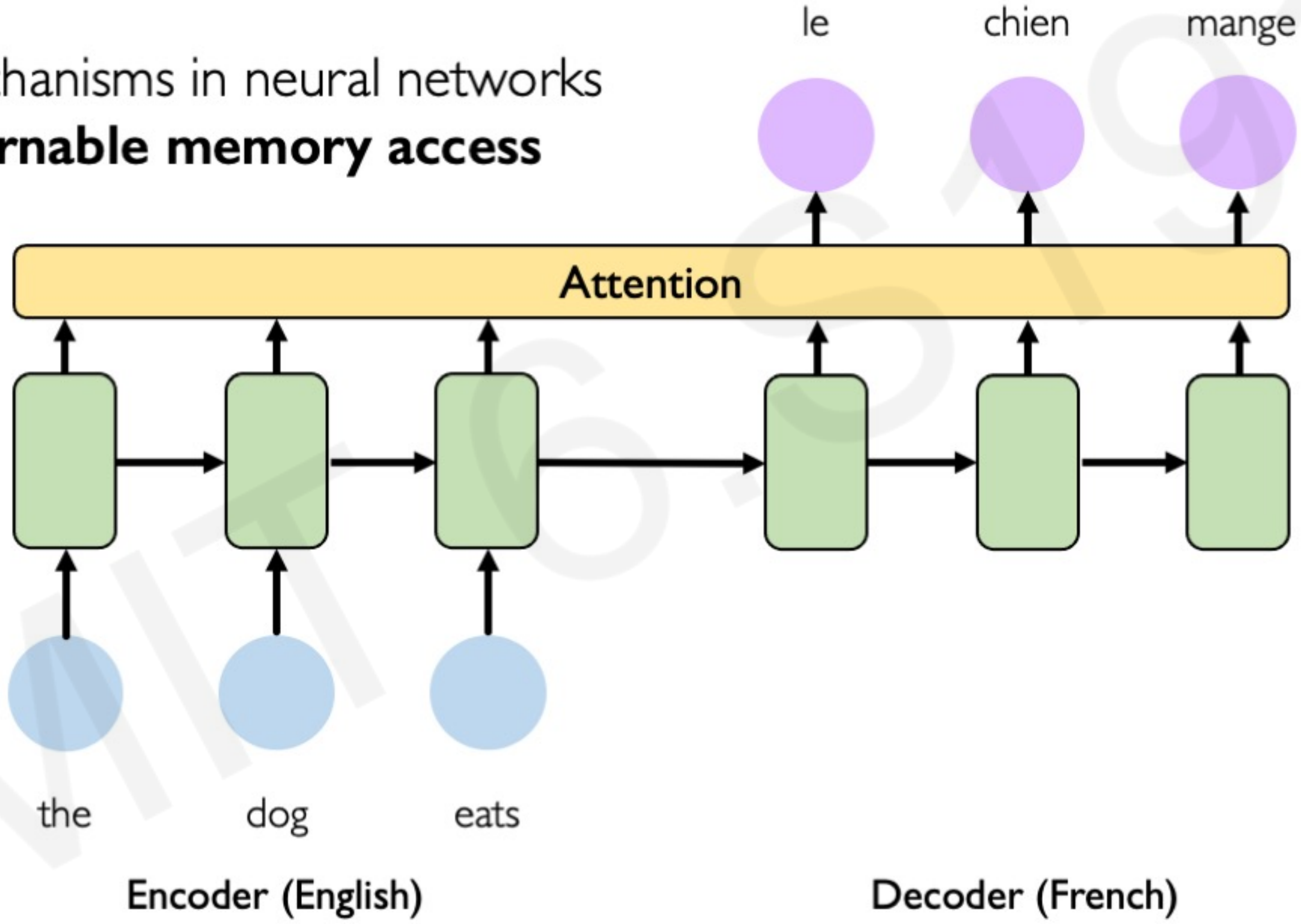
Potential Issues

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



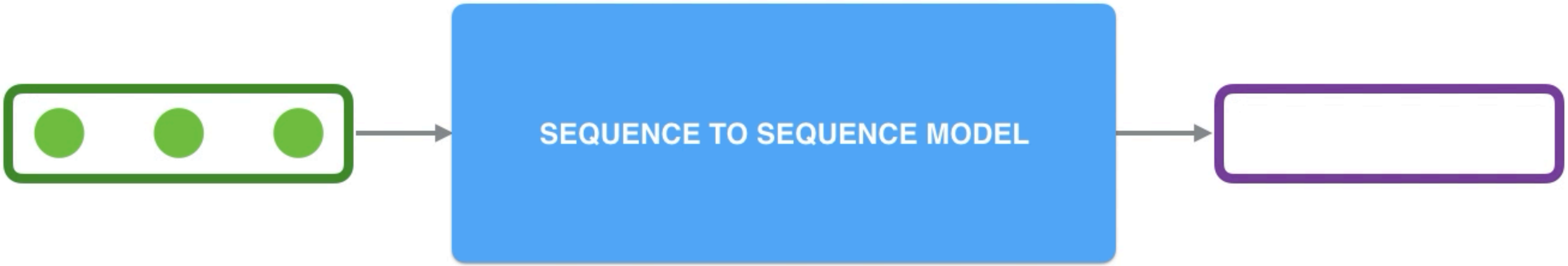
Example Task: Machine Translation

Attention mechanisms in neural networks provide **learnable memory access**



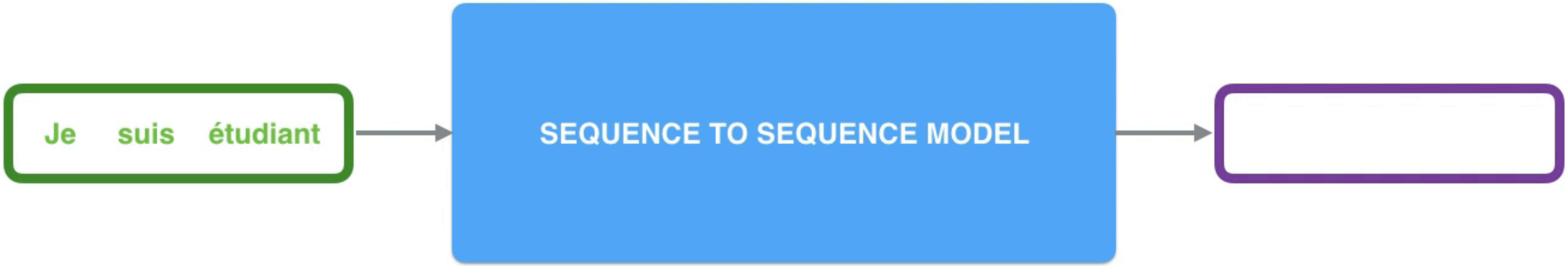
Attention

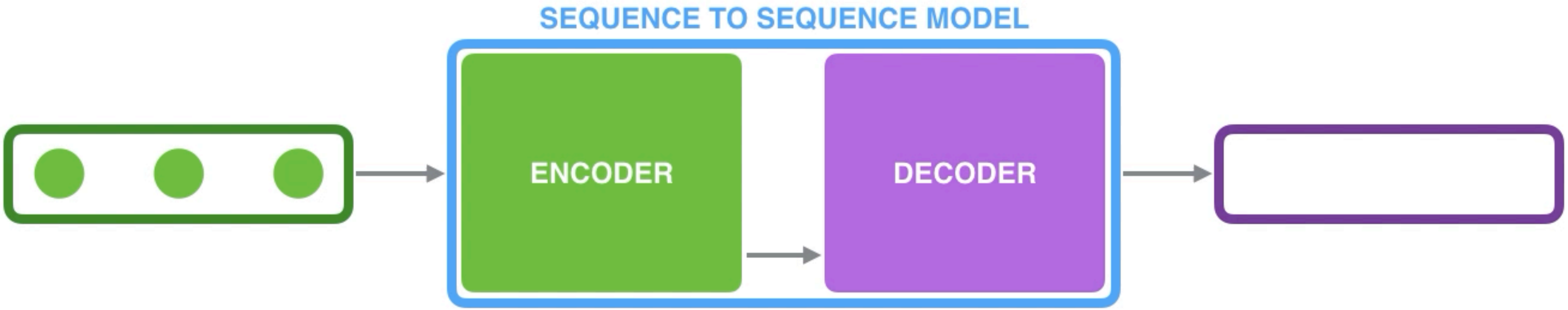
Following images/videos from Jay Alammar, "The Illustrated Transformer" and "Visualizing a Neural Machine Translation Model (Mechanics of Seq2seq Models with Attention")



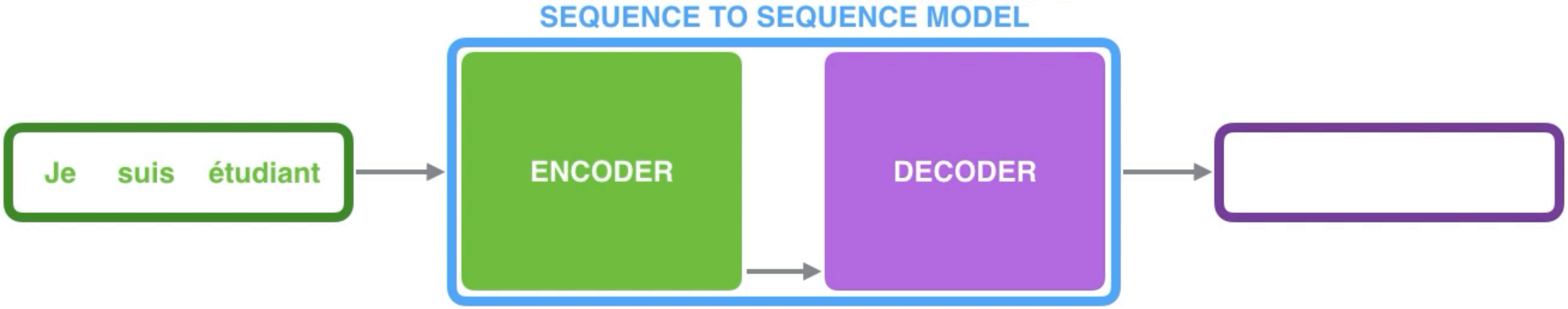
Neural Machine Translation

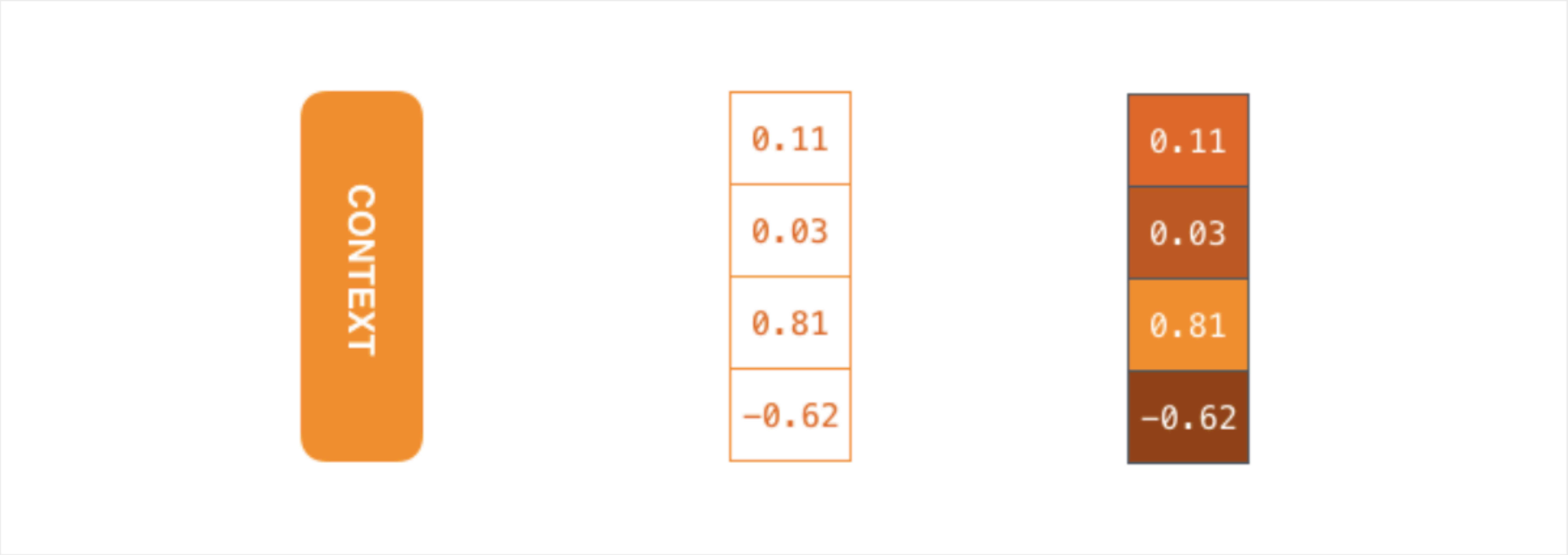
SEQUENCE TO SEQUENCE MODEL



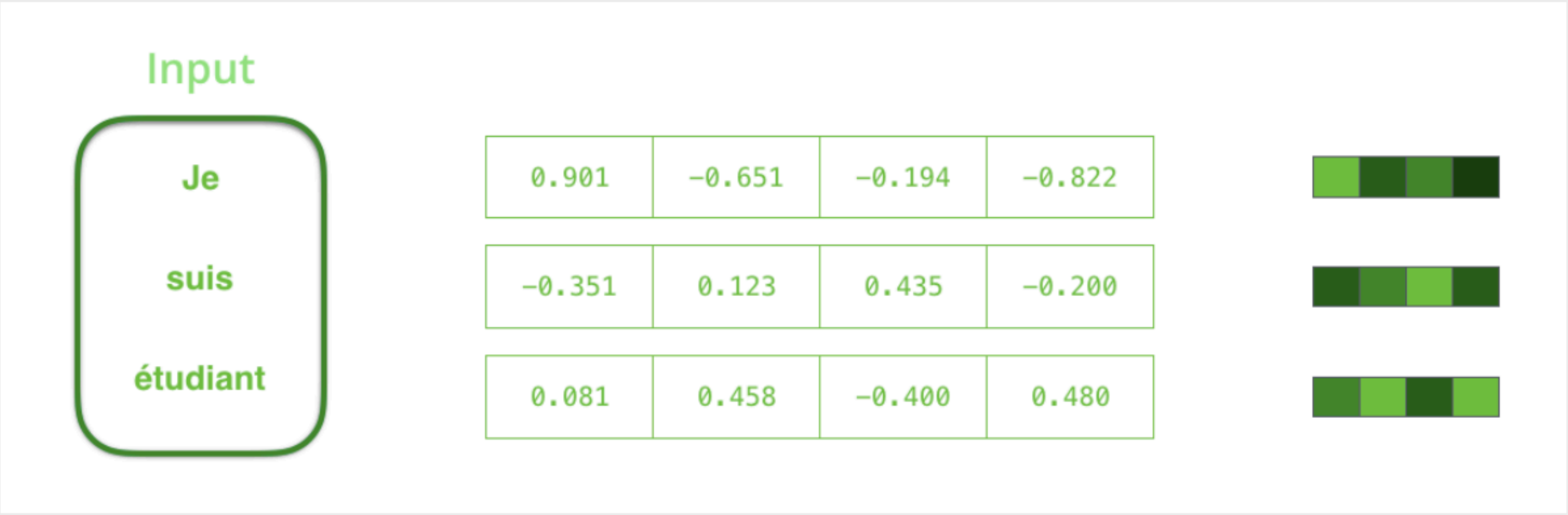


Neural Machine Translation





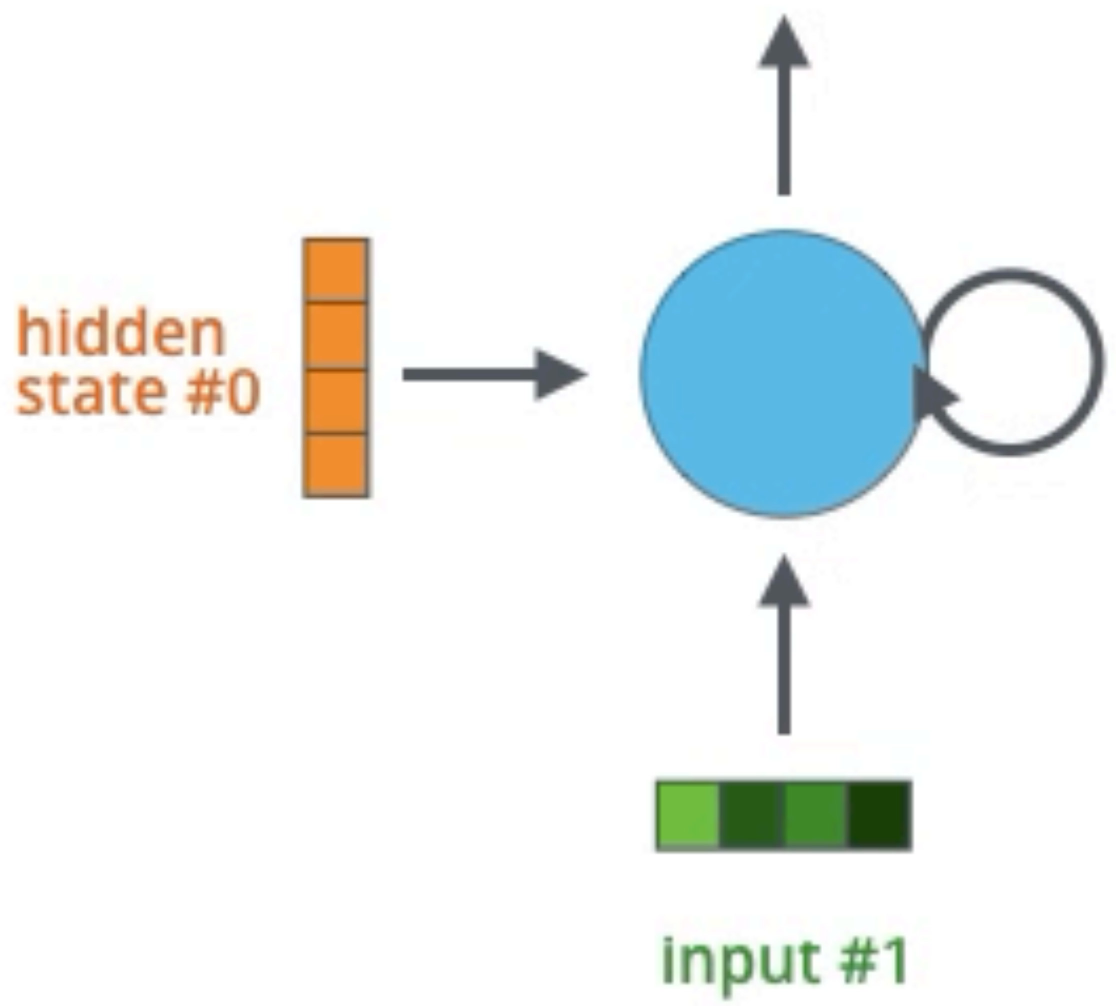
The context is a vector of floats. Later in this post we will visualize vectors in color by assigning brighter colors to the cells with higher values.



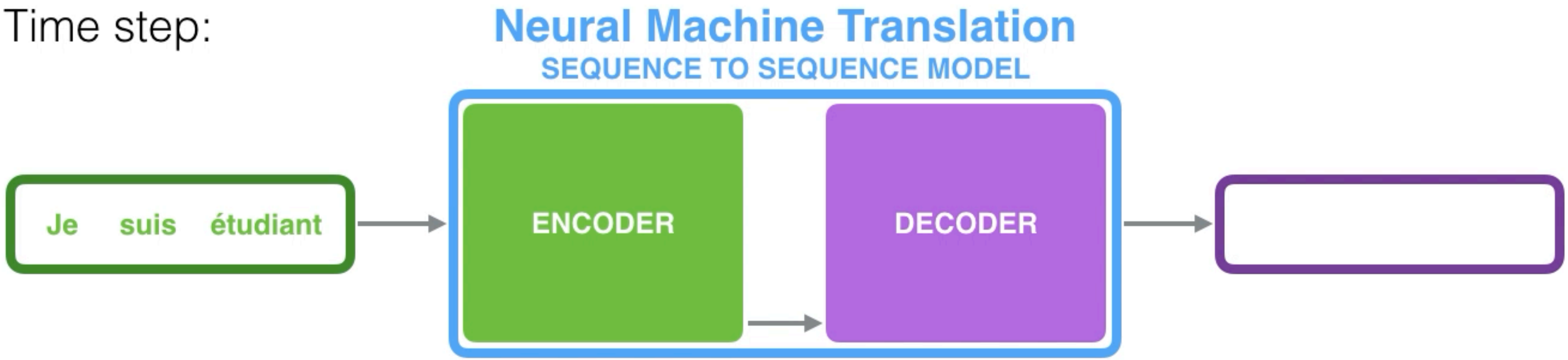
We need to turn the input words into vectors before processing them. That transformation is done using a word embedding algorithm. We can use pre-trained embeddings or train our own embedding on our dataset. Embedding vectors of size 200 or 300 are typical, we're showing a vector of size four for simplicity.

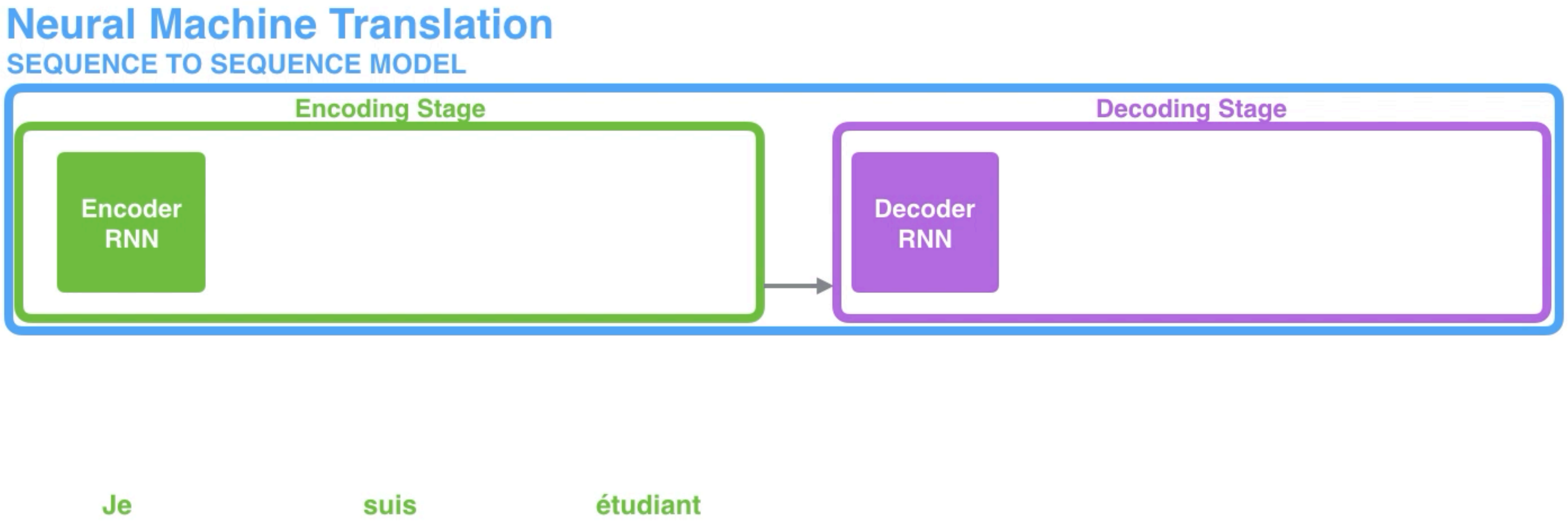
Recurrent Neural Network

Time step #1:
An RNN takes two input vectors:



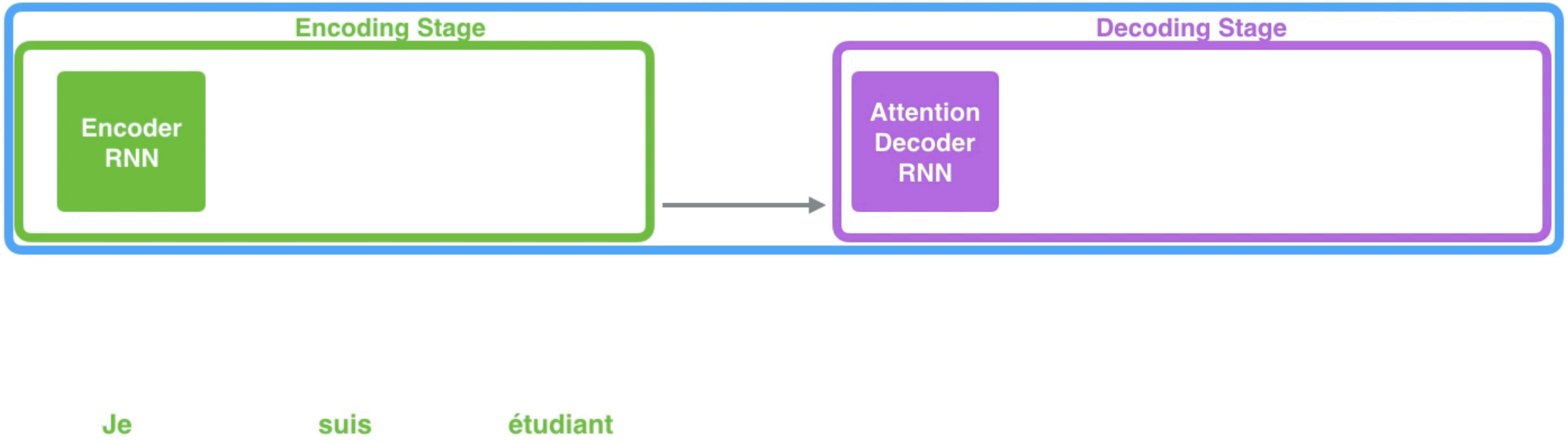
Time step:


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL
Je suis étudiant
ENCODER
DECODER


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL
Encoding Stage
Decoding Stage
Encoder RNN
Decoder RNN
Je
suis
étudiant

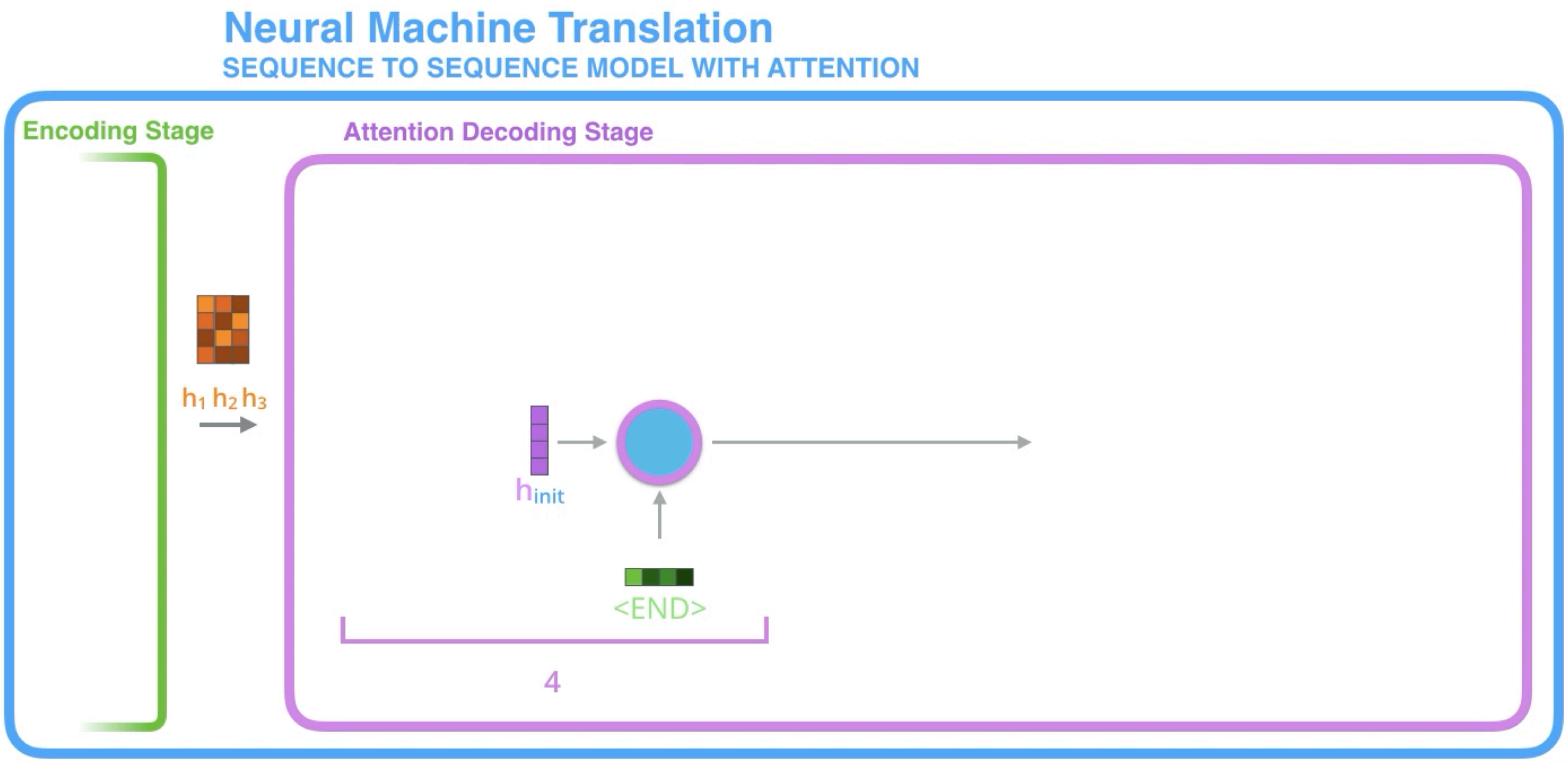
Neural Machine Translation

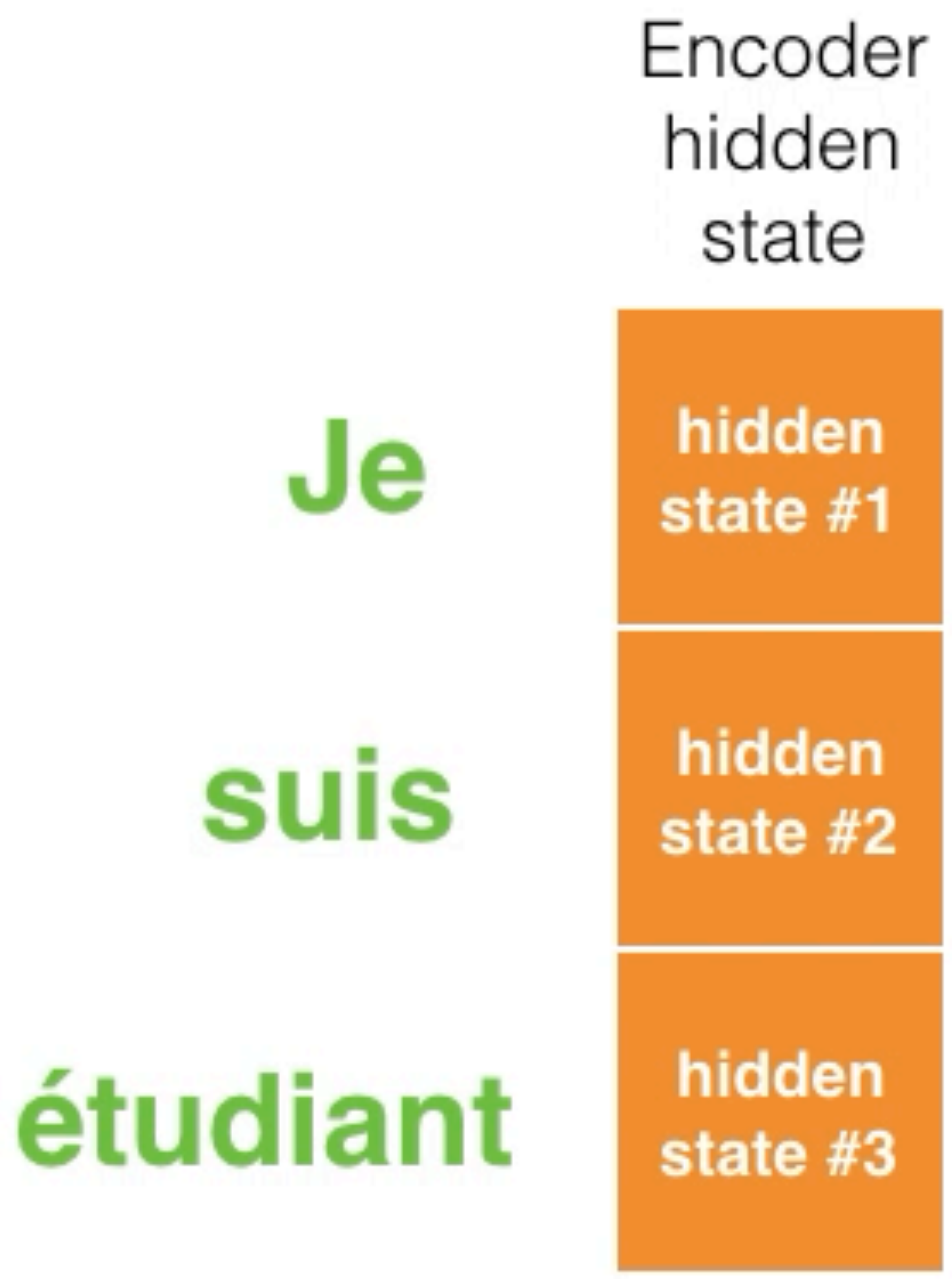
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION

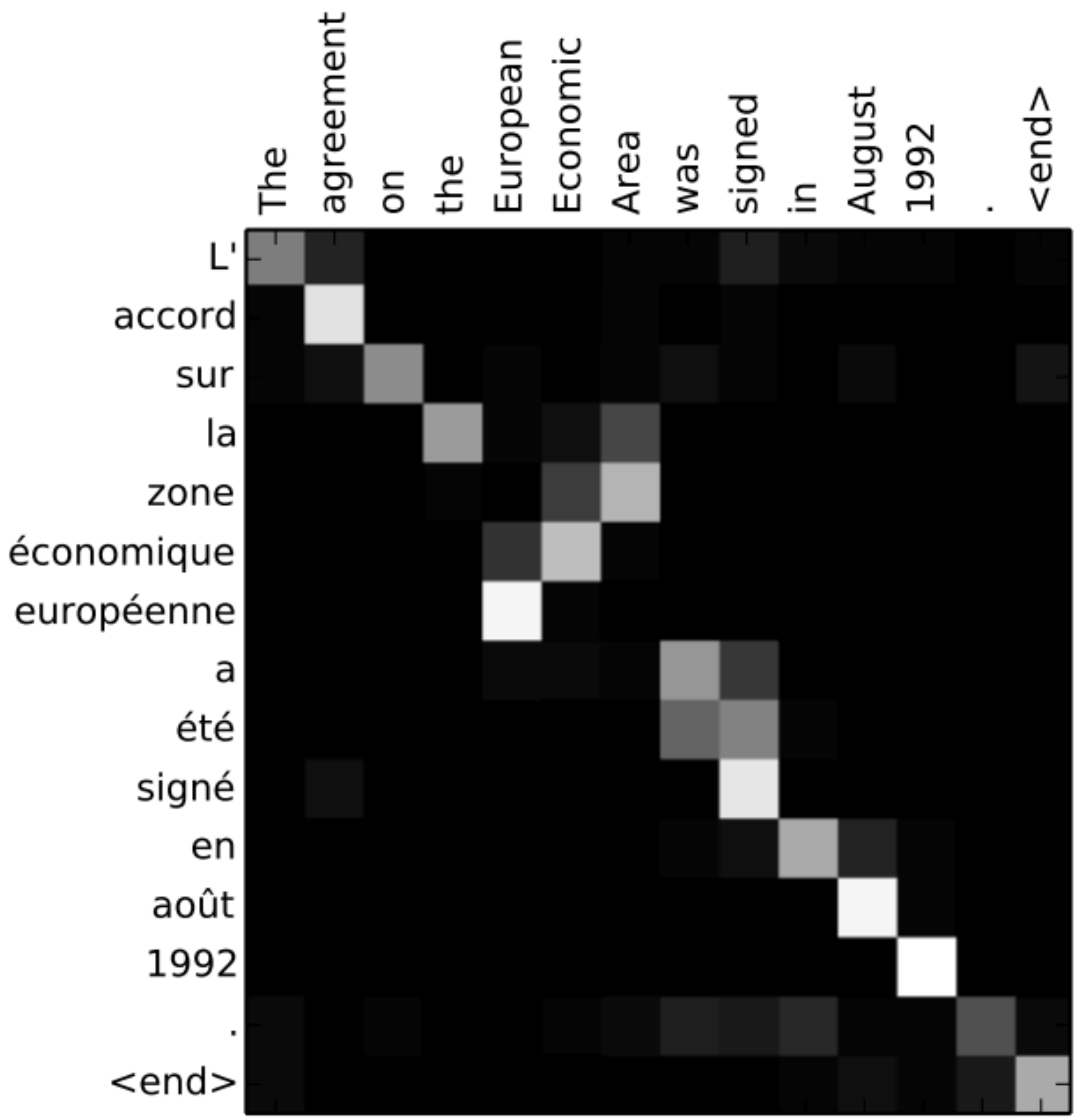


Attention at time step 4

1. The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
2. The RNN processes its inputs, producing an output and a new hidden state vector (h4). The output is discarded.
3. Attention Step: We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
4. We concatenate h4 and C4 into one vector.
5. We pass this vector through a feedforward neural network (one trained jointly with the model).
6. The output of the feedforward neural networks indicates the output word of this time step.
7. Repeat for the next time steps

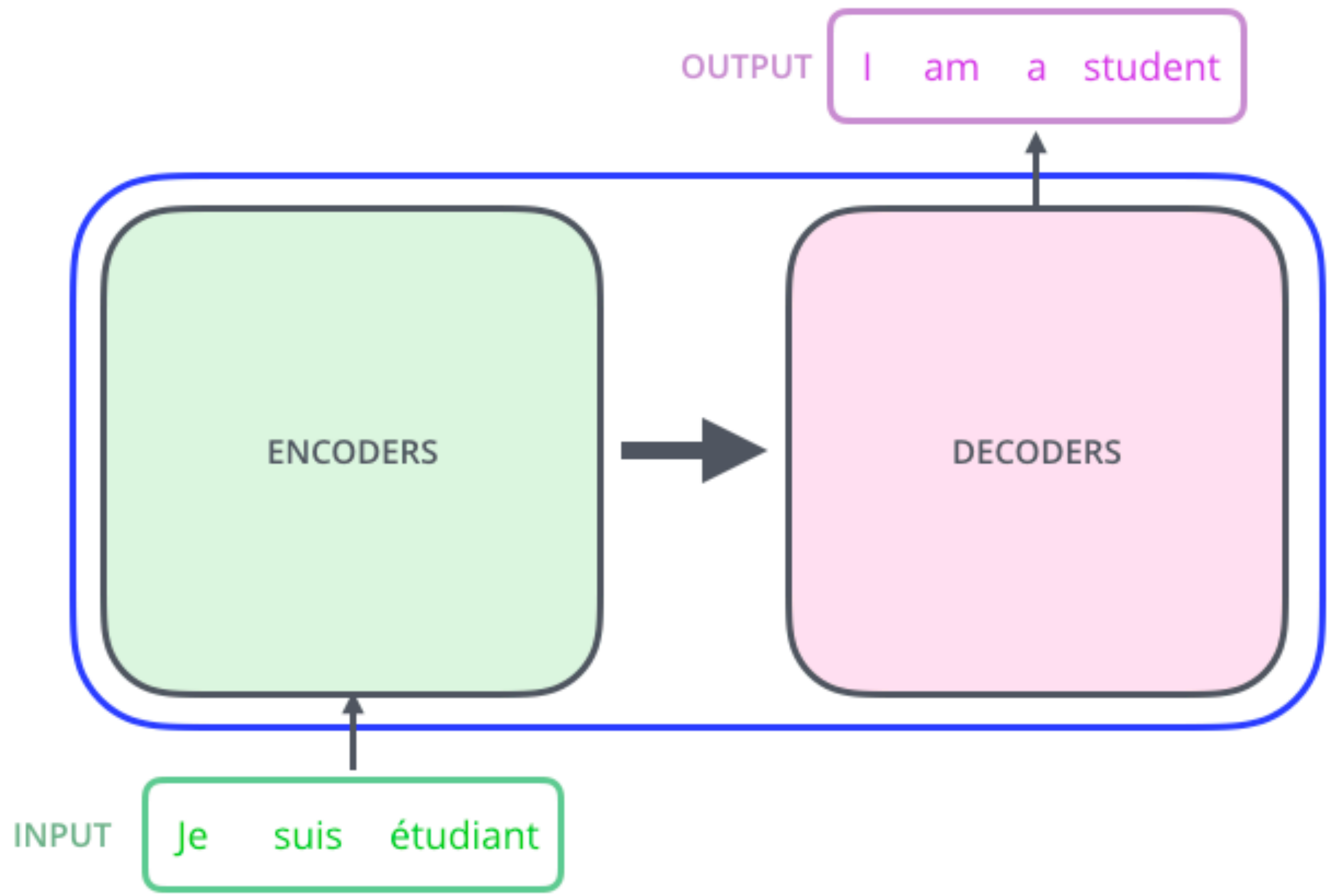


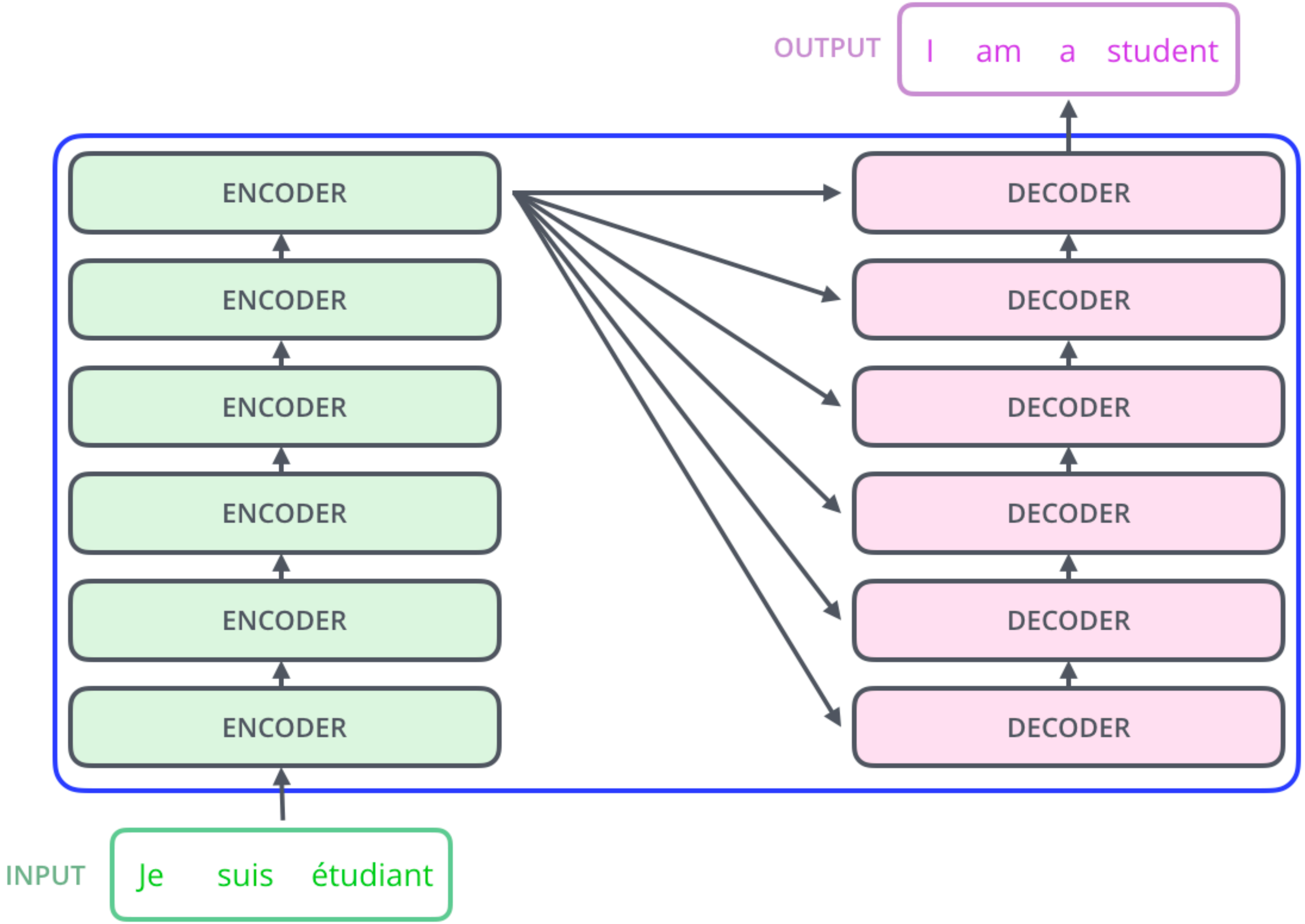

Encoder hidden state
Je
suis
étudiant
hidden state #1
hidden state #2
hidden state #3

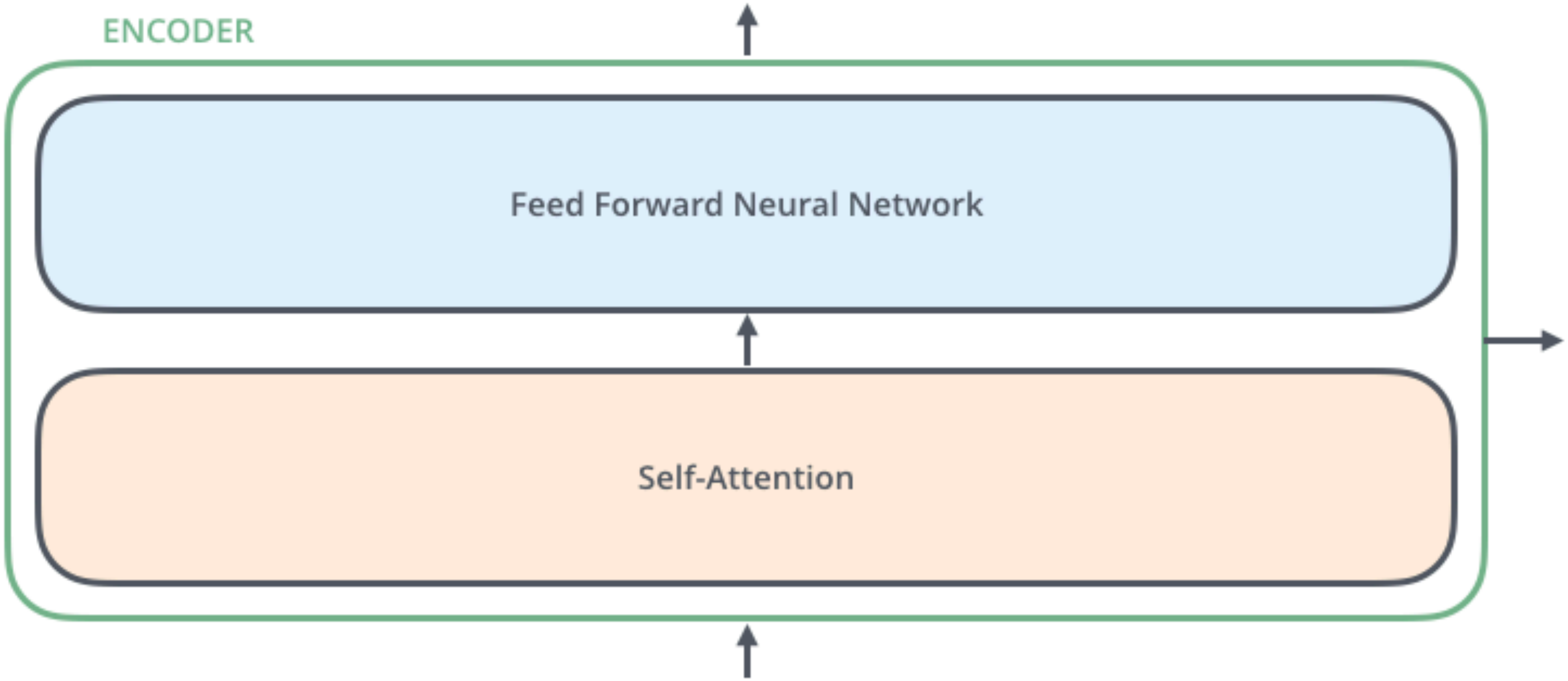

The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

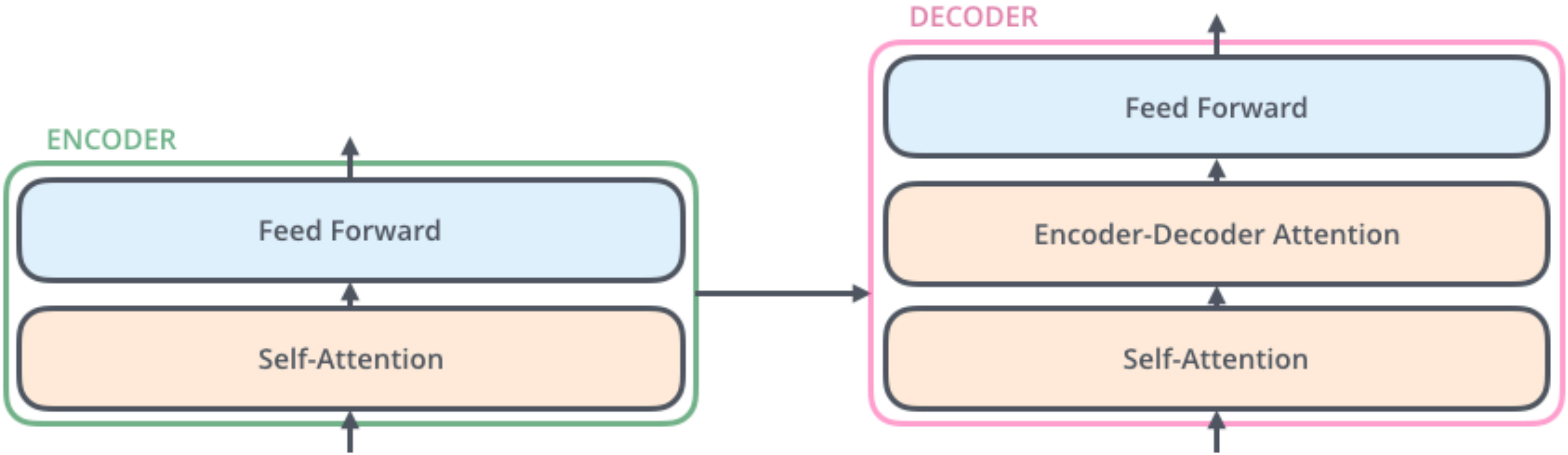
The transformer (“Attention is All You Need”)

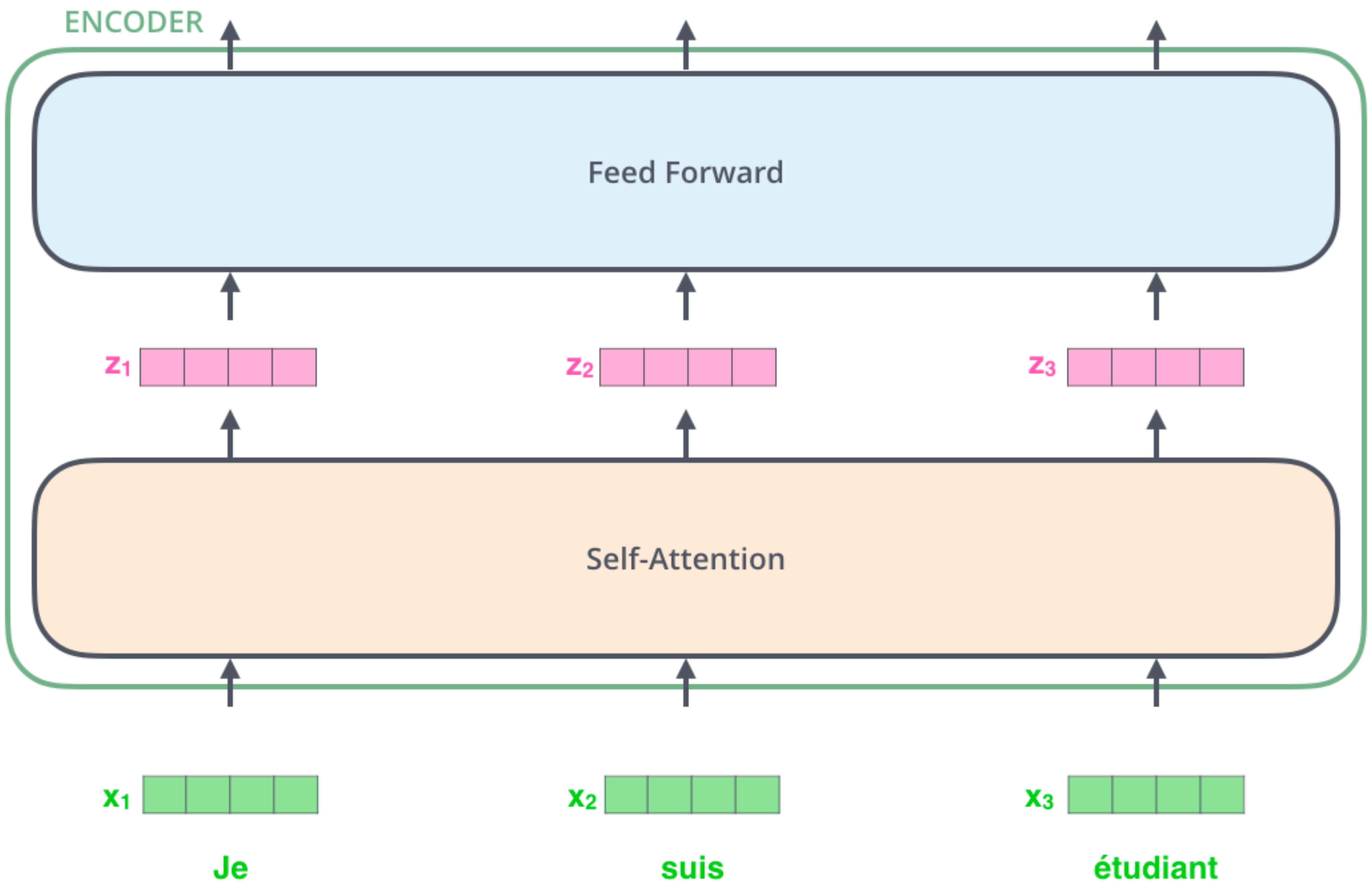

INPUT
Je suis étudiant
THE
TRANSFORMER
OUTPUT
I am a student

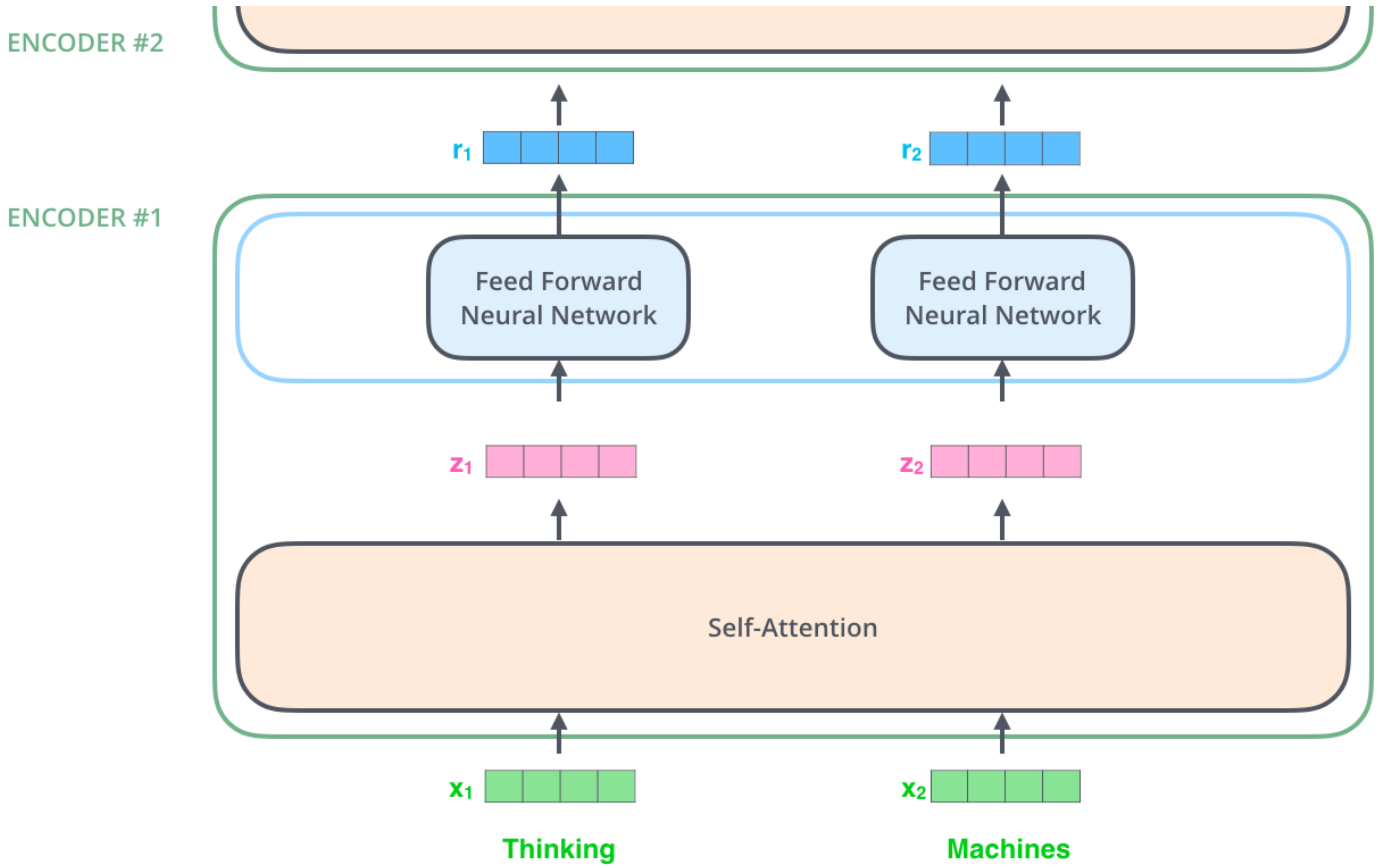

OUTPUT
I am a student
ENCODERS
DECODERS
INPUT
Je suis étudiant

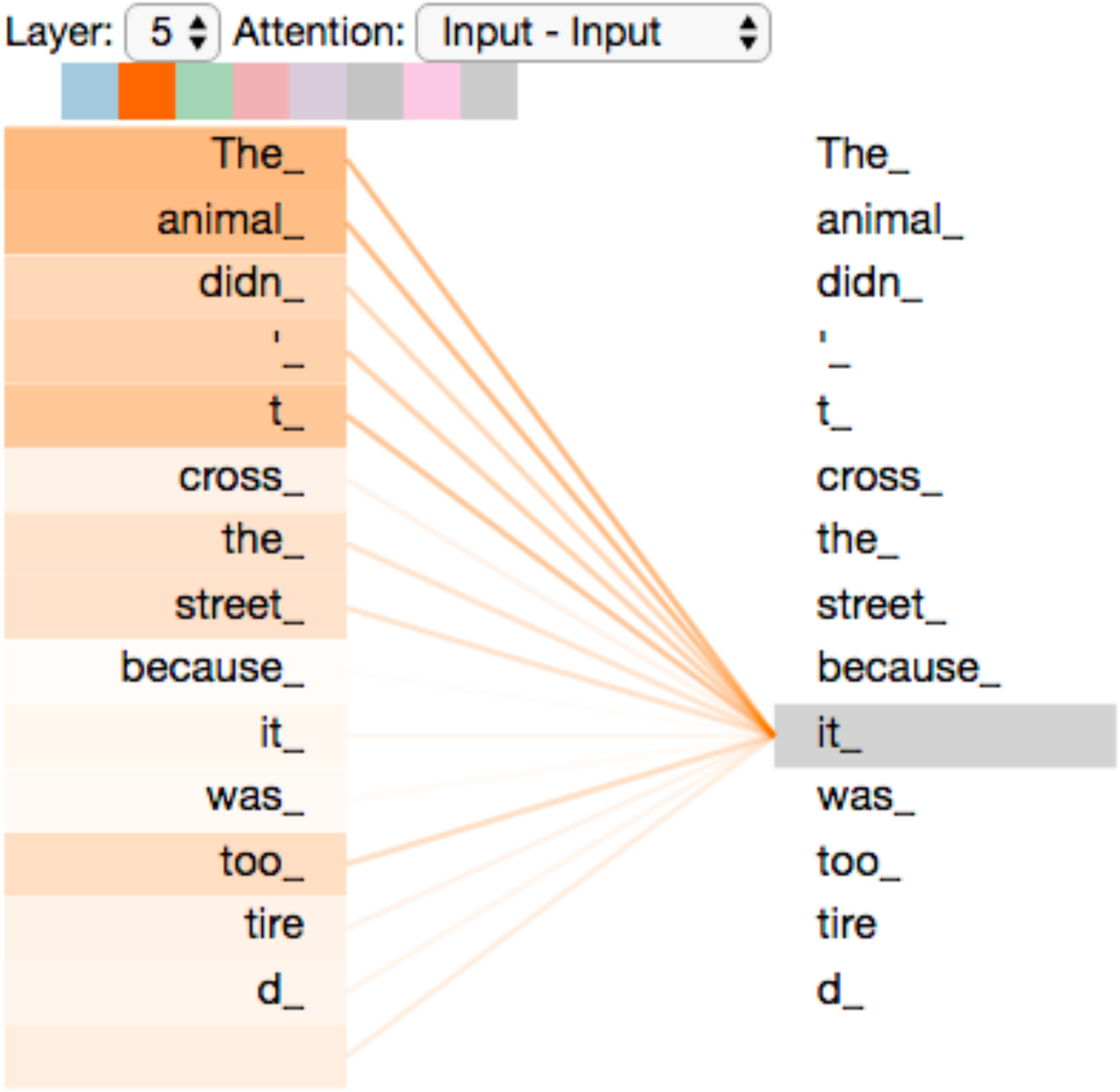

OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant


ENCODER
Feed Forward Neural Network
Self-Attention


DECODER
Feed Forward
Encoder-Decoder Attention
Self-Attention
ENCODER
Feed Forward
Self-Attention


ENCODER
Feed Forward
z_1
z_2
z_3
Self-Attention
x_1
x_2
x_3
Je
suis
étudiant

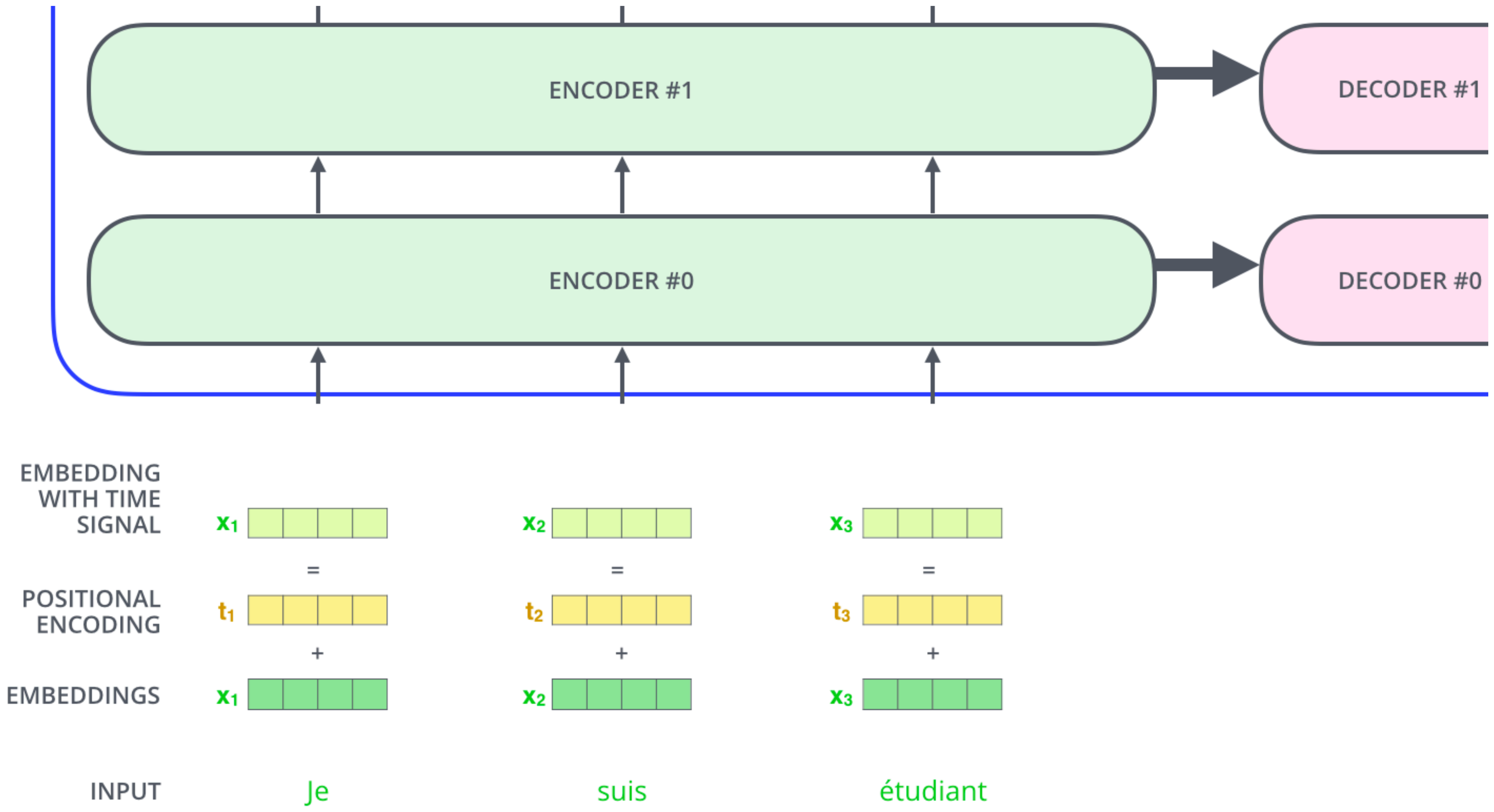

ENCODER #2
r1
r2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

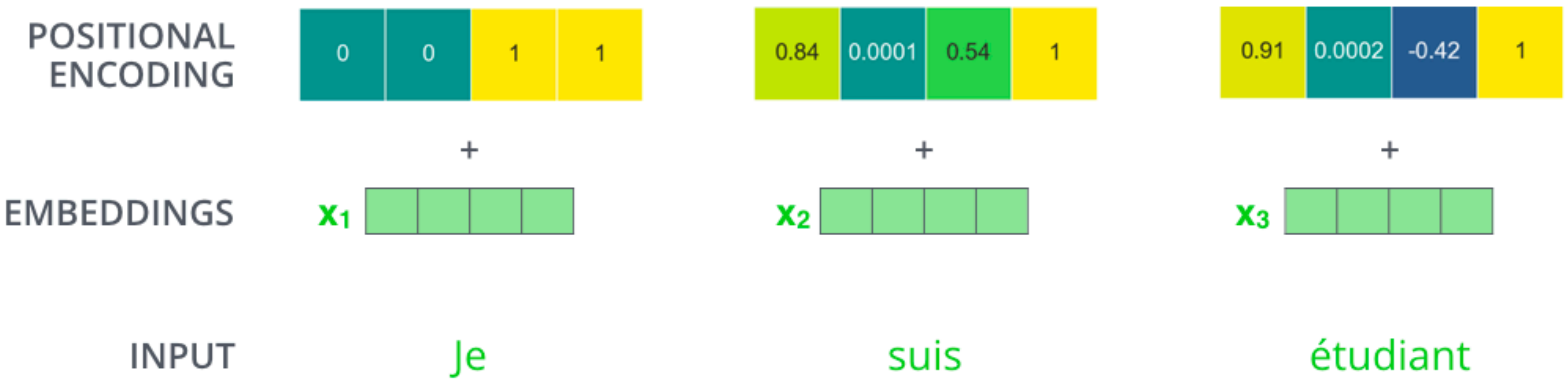


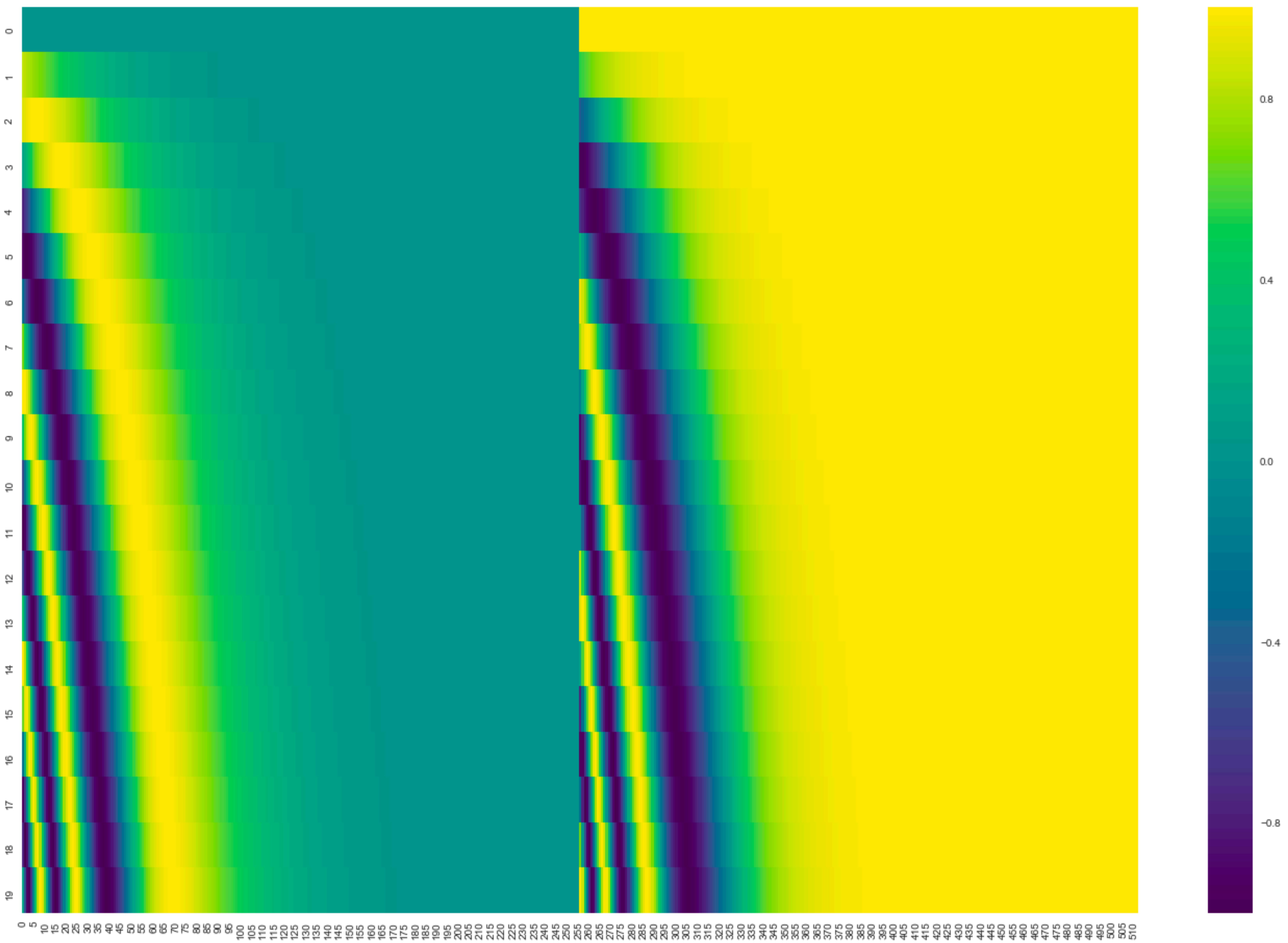
Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

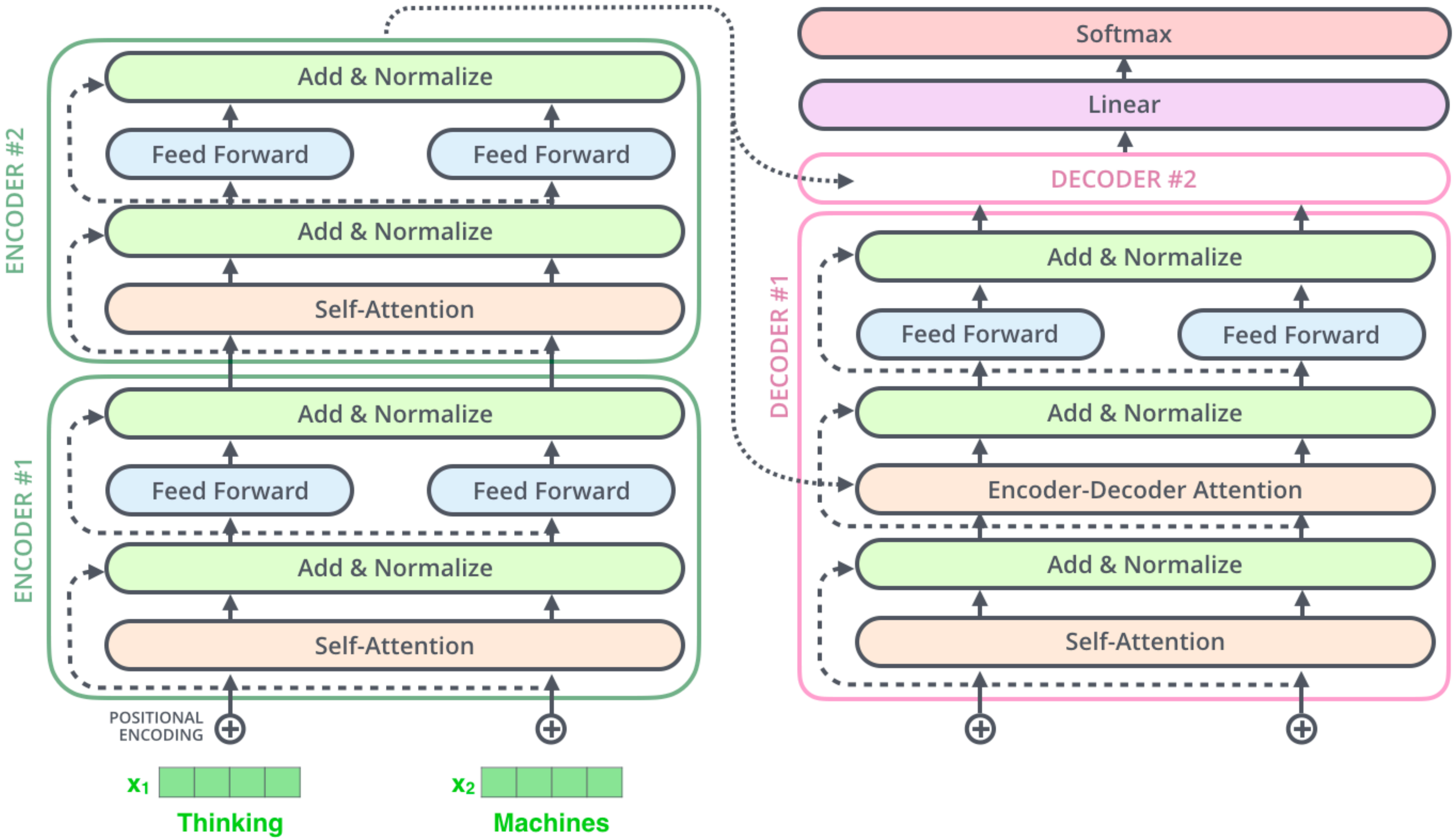
Matrix math of self-attention - yada yada yada

Multi-headed attention - blah blah blah


ENCODER #1
DECODER #1
ENCODER #0
DECODER #0
EMBEDDING WITH TIME SIGNAL
x_1
x_2
x_3
=
POSITIONAL ENCODING
t_1
t_2
t_3
+
EMBEDDINGS
x_1
x_2
x_3
INPUT
Je
suis
étudiant


POSITIONAL ENCODING
0
0
1
1
0.84
0.0001
0.54
1
0.91
0.0002
-0.42
1
+
+
+
EMBEDDINGS
x_1
x_2
x_3
INPUT
Je
suis
étudiant




Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x_1
Thinking
x_2
Machines