Lecture 9-10-11: Deep Learning - Language Models

Advanced Practical Data Science, MLOps

AC295

Pavlos Protopapas
Institute for Applied Computational Science, Harvard

Outline

1. What are Language Models
2. Neural Networks for Language Modeling
3. Recurrent Neural Network
4. Seq2Seq + Attention
5. Self Attention
6. Transformers
7. Tutorial: SOTA Language Models

Outline

1. **What are Language Models**
2. Neural Networks for Language Modeling
3. Recurrent Neural Network
4. Seq2Seq + Attention
5. Self Attention
6. Transformers
7. Tutorial: SOTA Language Models

Language Models

Today, we heavily focus on Language Modelling (LM) because:

1. It’s foundational for nearly all NLP tasks.

2. LM approaches are generalizable to any type of data, not just text.

3. The data is readily available in huge quantities.

Language Models: Background

Regardless of how we model sequential data, keep in mind that we can estimate any time series as follows:

This compounds for all subsequent events, too

$$P(x_1, \dots, x_T) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \dots, x_1)$$

Joint distribution of all measurements

Conditional probability of an event, depends on all of the events that occurred before it.

Language Models: Example

If we want to know the probability of the the next on-screen Sesame Street character:

Scene 1

Scene 2

Scene 3

Language Models: Example

Remember that, when we are evaluate a distribution, we mean

$P($  $,$  $) = P(S_1 =$  $, S_2 =$  $)$

Language Models: Example

The probability of the the next on-screen Sesame Street character can be computed as

Scene 1

Scene 2

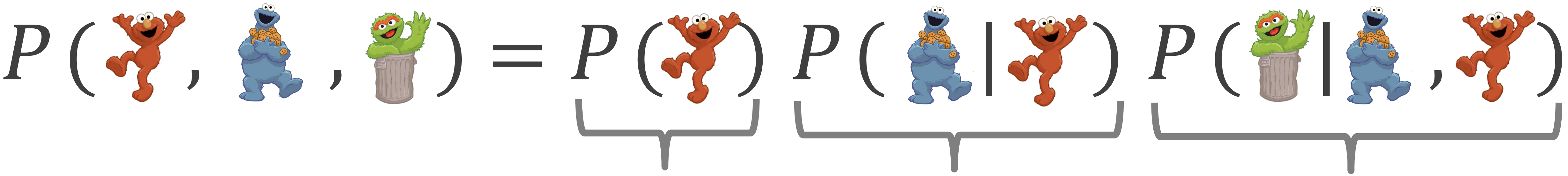
Scene 3

$P($  ,  ,  $) =$

Language Models: Example

The probability of the the next on-screen Sesame Street character can be computed as

Scene 1 Scene 2 Scene 3



Scene 1 Scene 2 Scene 3

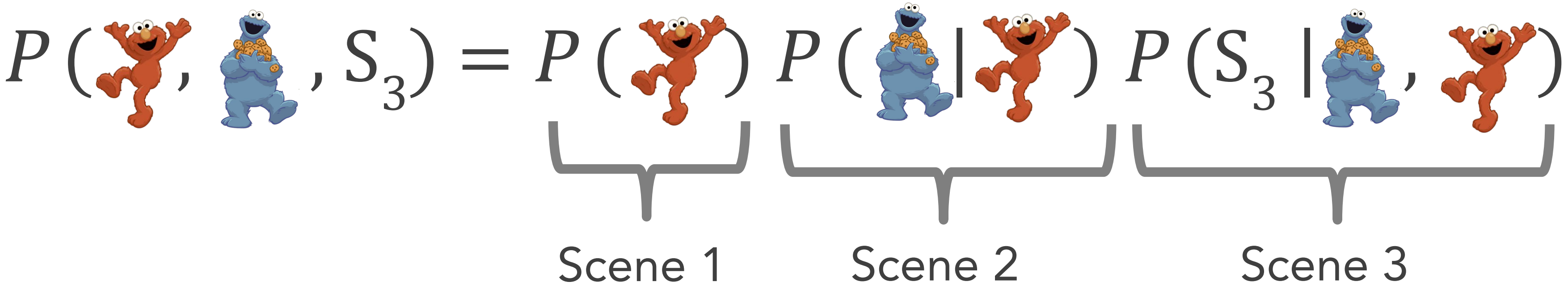
Language Models: Example

Why is it useful to accurately estimate the joint probability of any given sequence of length N?

Language Models: Background

Having learned a Language Model means that we know the behavior of the sequences.

If we have a sequence of length N, we can determine the most likely next event (i.e., sequence of length N+1).



Language Models: **Formal Definition**

A Language Model estimates the probability of any sequence of words

Let $\boldsymbol{X}$ = "Shiv was late for class"

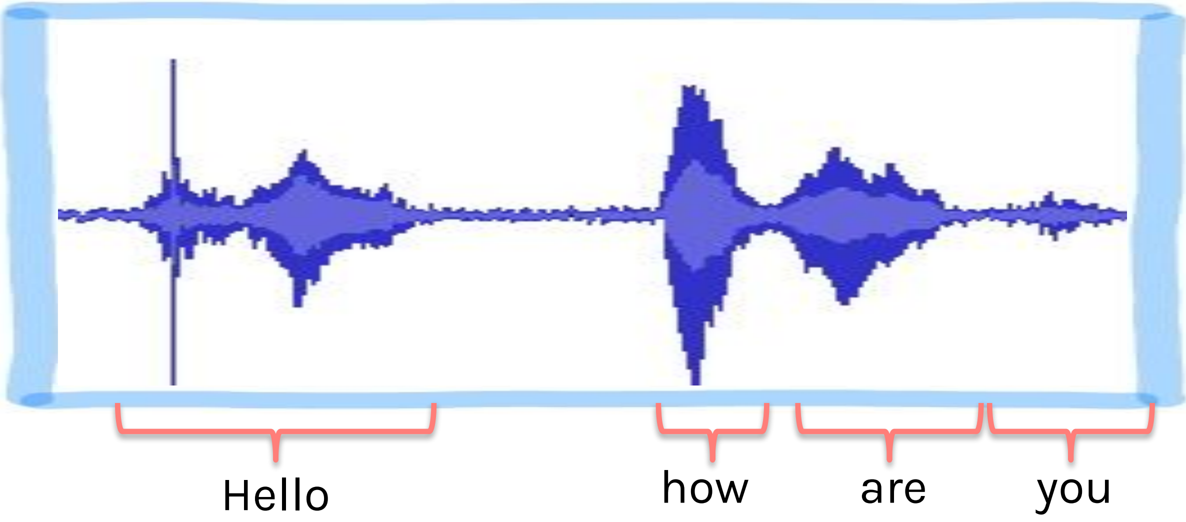
$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

$\mathrm{P}(\boldsymbol{X}) = P($"Shiv was late for class"$)$

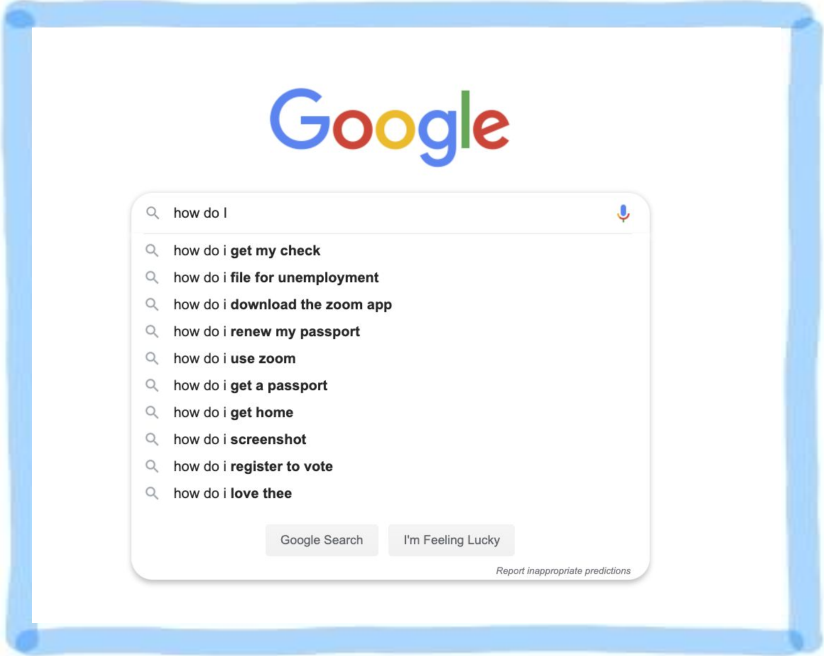
Language Models: Application

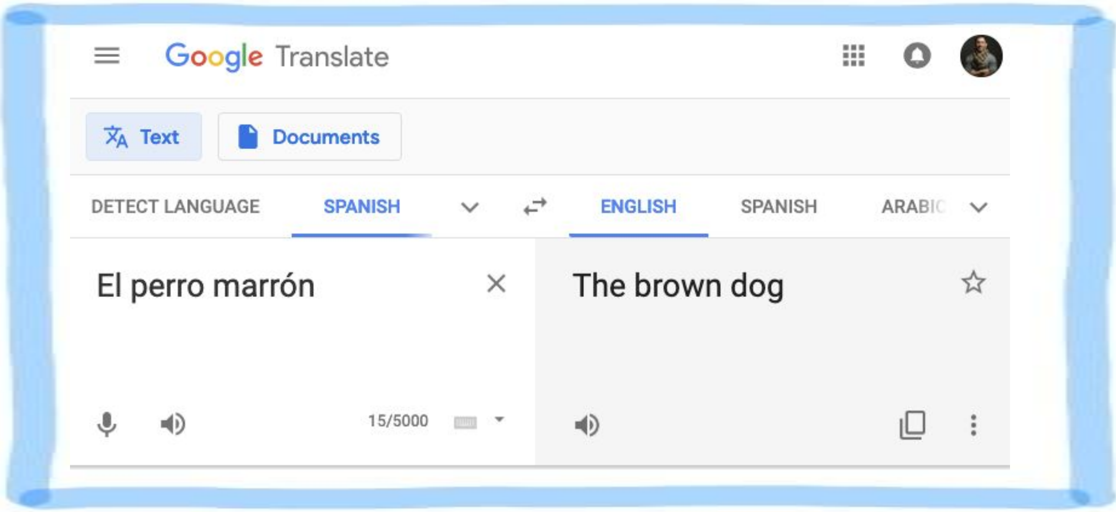
Text Recognition

Speech Recognition



Sentence Prediction





Translation

Outline

1. What are Language Models
2. **Neural Networks for Language Modeling**
3. Recurrent Neural Network
4. Seq2Seq + Attention
5. Self Attention
6. Transformers
7. Tutorial: SOTA Language Models

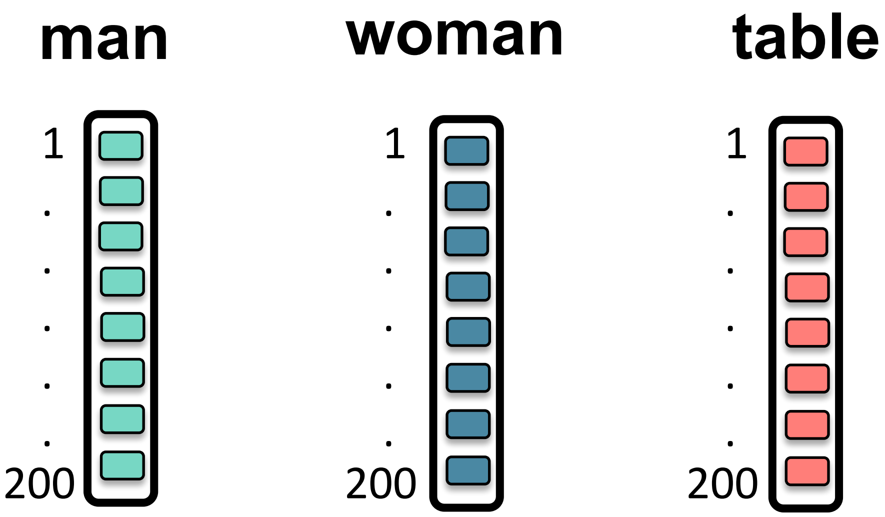
Neural Networks for Language Modeling

IDEA: Let’s use a **neural network**!

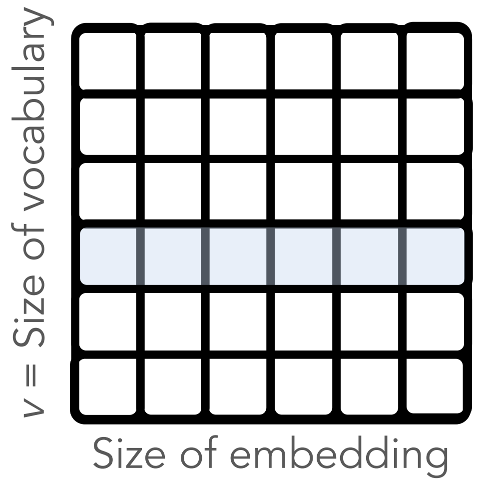
Neural Networks for Language Modeling

IDEA: Let's use a **neural network**!

First, each word is represented by a word embedding (e.g., vector of length 200)



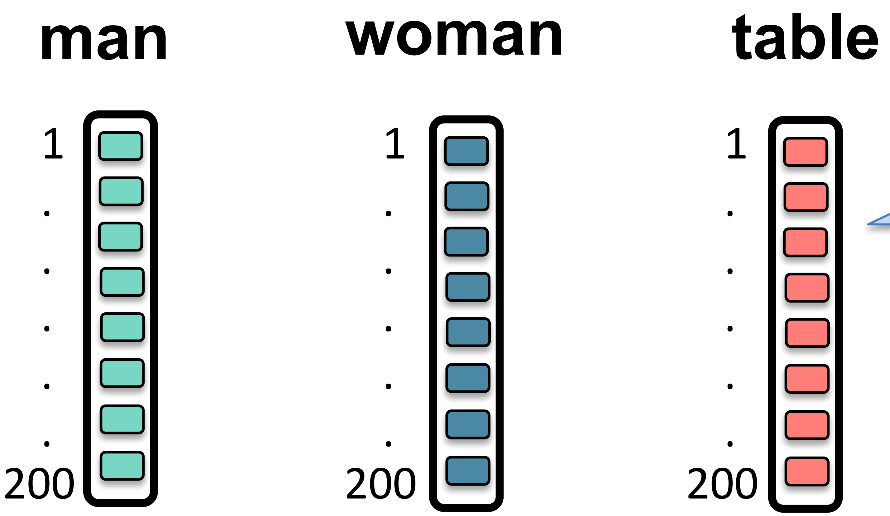
$X =$



Neural Networks for Language Modeling

IDEA: Let's use a **neural network**!

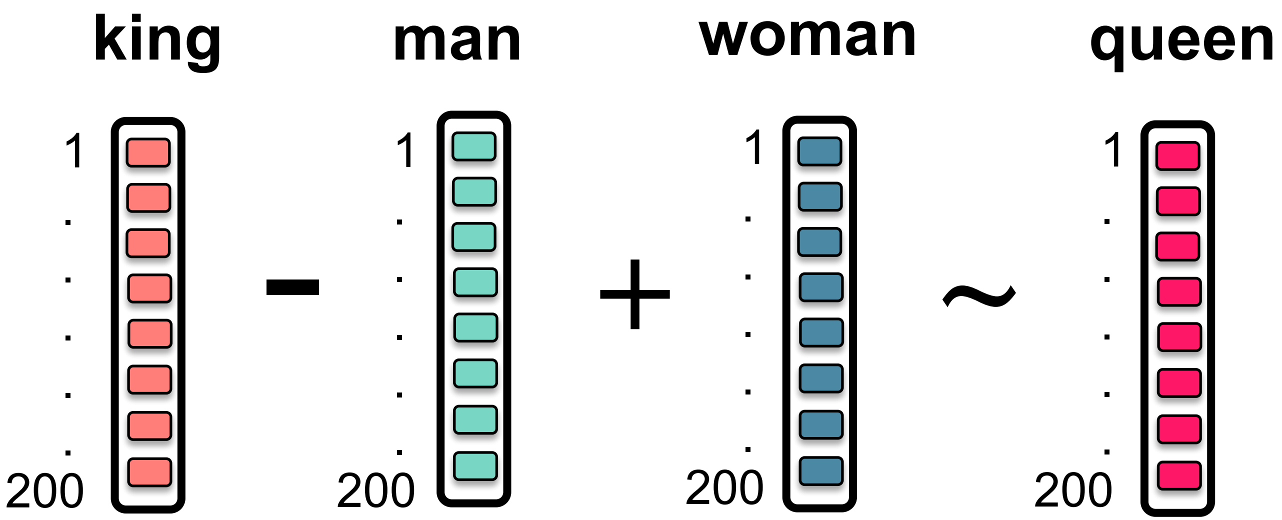
First, each word is represented by a word embedding (e.g., vector of length 200)



- Each rectangle is a *floating-point* scalar
- Words that are more *semantically similar* to one another will have **embeddings** that are also proportionally similar
- We can *use pre-existing* word embeddings that have been trained on gigantic corpora

Neural Networks for Language Modeling

These word embeddings are so rich that you get nice properties:



Word2vec: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
GloVe: https://www.aclweb.org/anthology/D14-1162.pdf

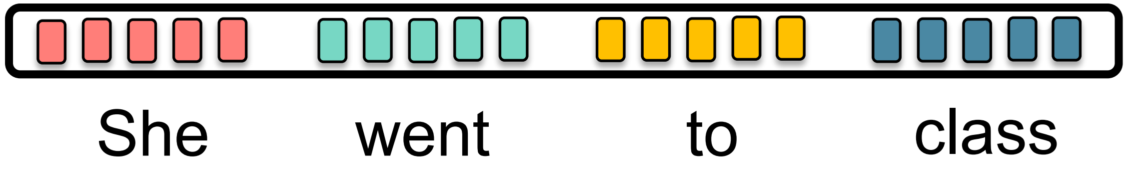
Neural Networks for Language Modeling

How can we use these embeddings to build a LM?

Remember, we only need a system that can estimate:

$$P(\underbrace{x_{t+1}}_{\text{next word}}|\underbrace{x_t, x_{t-1}, \dots, x_1}_{\text{previous words}})$$

Example input sentence

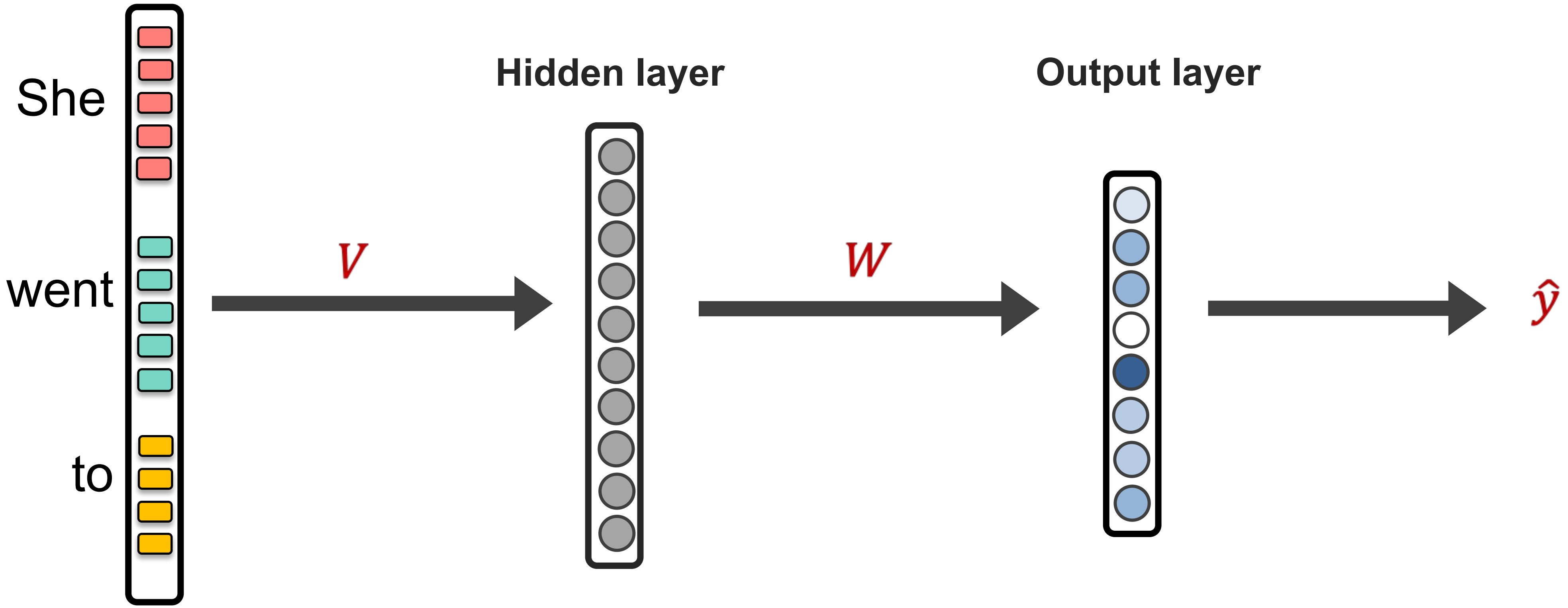


Neural Networks for Language Modeling

Neural Approach #1: Feed-forward neural net

General Idea: using *windows* of words, predict the next word

Example input sentence



Neural Networks for Language Modeling

$x = [x_1, x_2, x_3]$

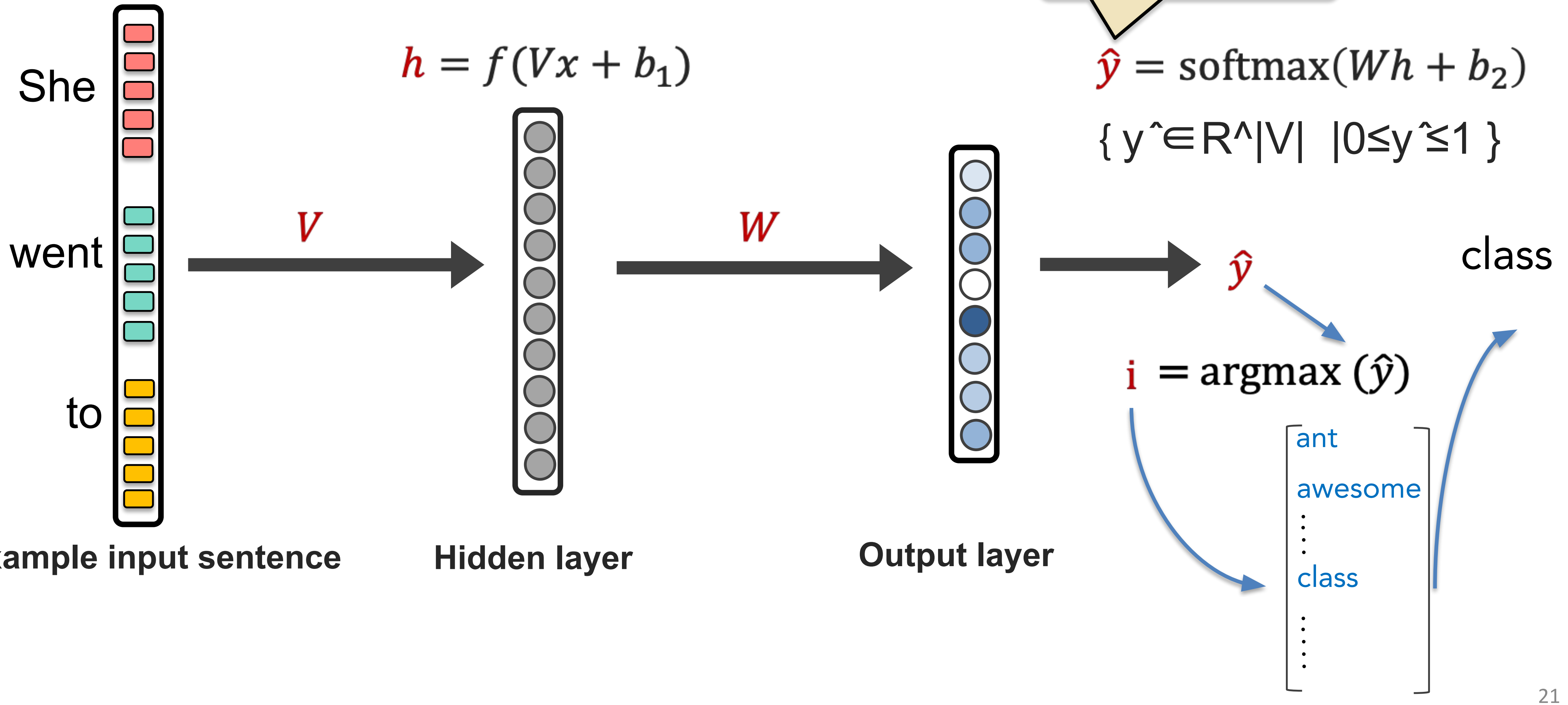
Concatenated word embeddings

$h = f(Vx + b_1)$

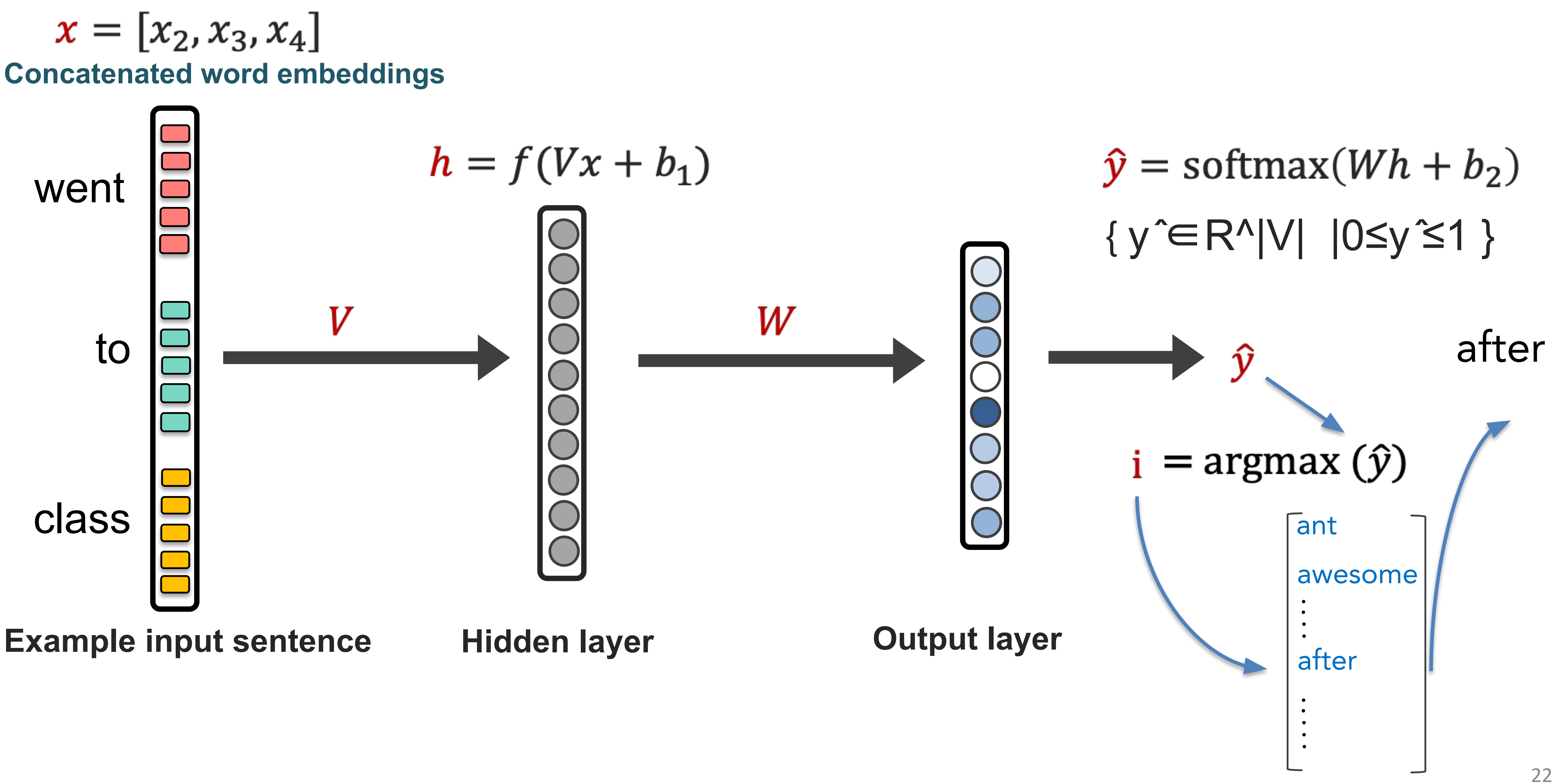
Probabilities

$\hat{y} = \text{softmax}(Wh + b_2)$

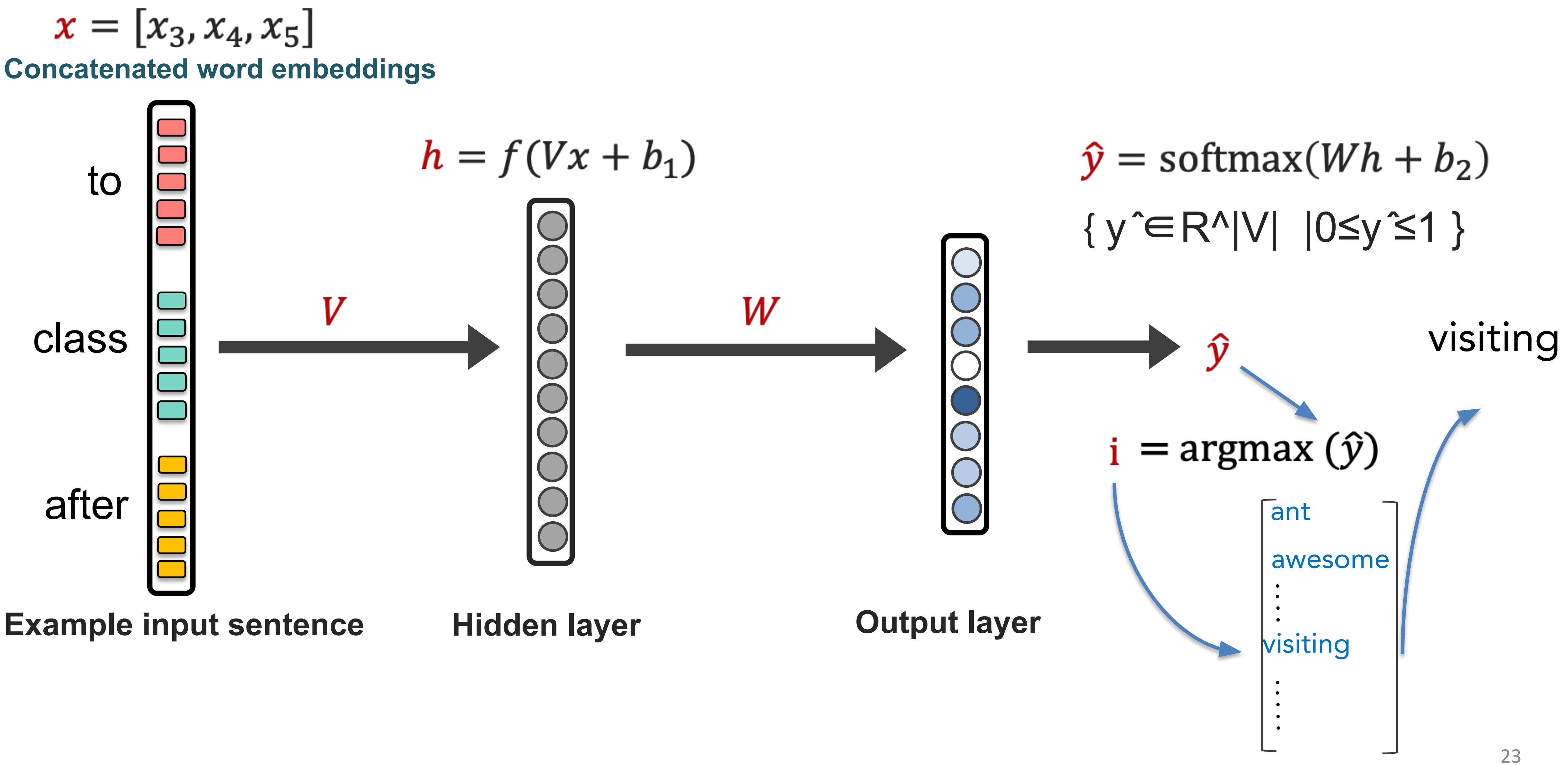
$\{\hat{y} \in R^{|V|} \mid 0 \le \hat{y} \le 1\}$



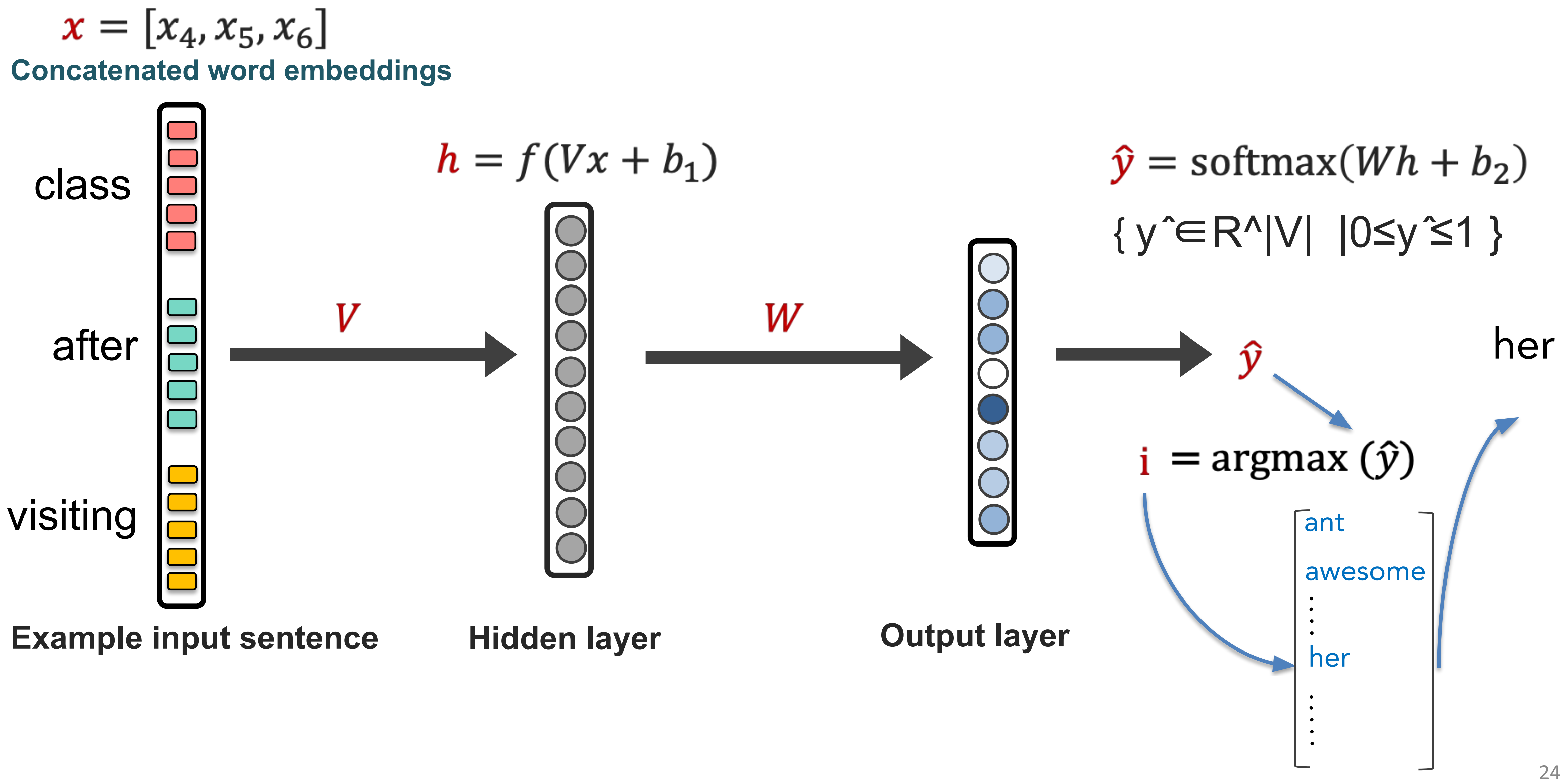
Neural Networks for Language Modeling



Neural Networks for Language Modeling

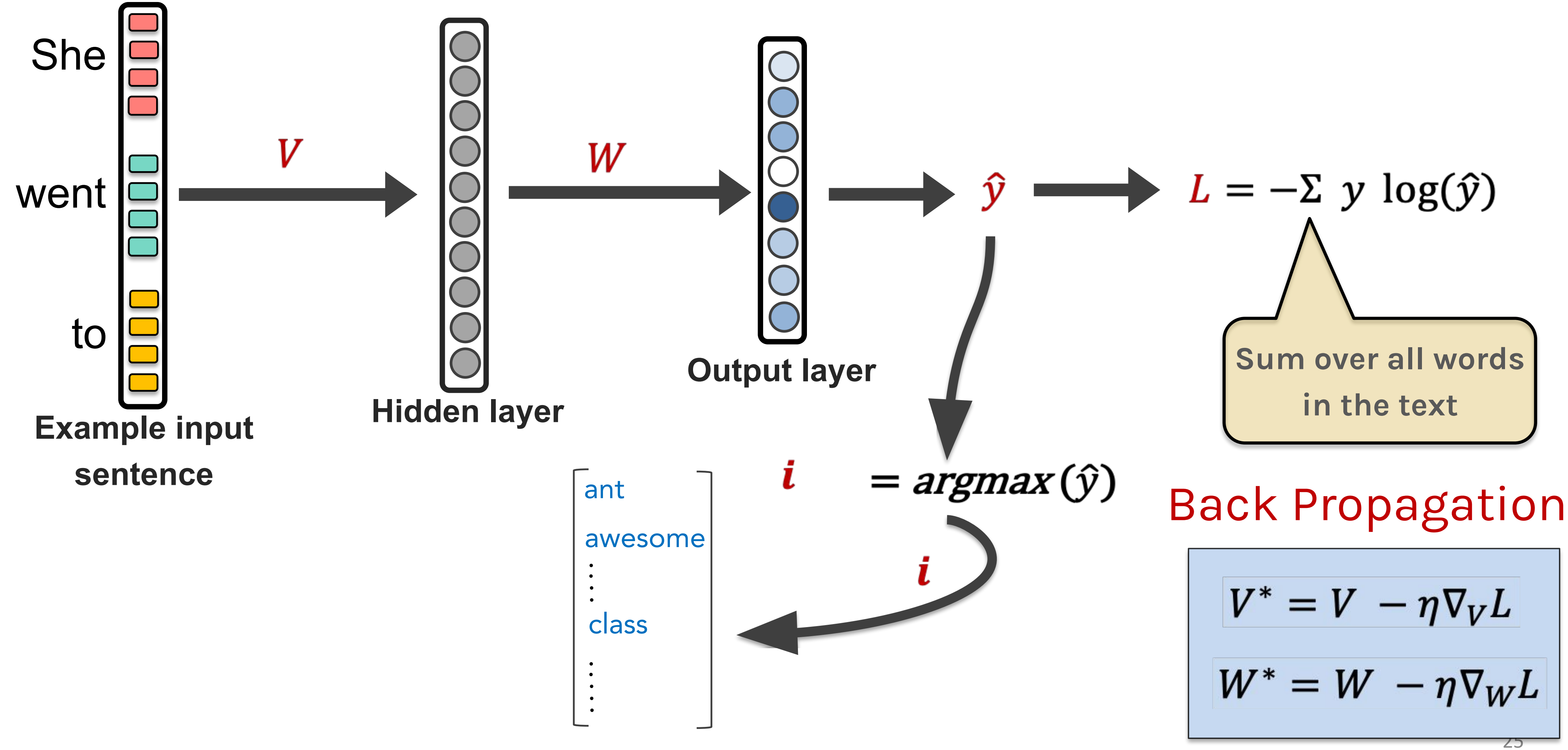


Neural Networks for Language Modeling



Neural Networks for Language Modeling (Training)

$x = [x_1, x_2, x_3]$ $\quad$ $h = f(Vx + b_1)$ $\quad$ $\hat{y} = \text{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$



Neural Networks for Language Modeling

FFNN Strength

- No sparsity issues (it's okay if we've never seen a word)
- No storage issues (we never store counts)

compared to traditional n-gram methods

Neural Networks for Language Modeling

FFNN Strength

- No sparsity issues (it's okay if we've never seen a word)
- No storage issues (we never store counts)

FFNN Issues

- Fixed-window size can never be big enough. Need more context
    - Requires inputting entire context just to predict one word
    - Increasing window size adds many more weights
- The weights awkwardly handle word position
- No concept of time

Neural Networks for Language Modeling

We especially need a system that:

- Has a concept of an “infinite” past, not just a fixed window
- For each new input, output the most likely next event (e.g., word)

Outline

1. What are Language Models
2. Neural Networks for Language Modeling
3. **Recurrent Neural Network**
4. Seq2Seq + Attention
5. Self Attention
6. Transformers
7. Tutorial: SOTA Language Models

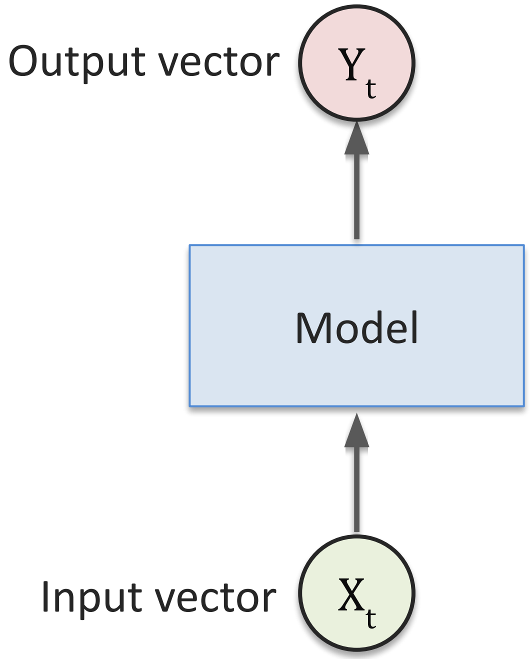
Recurrent Neural Network: Motivations

RNNs should exhibit the following advantages for sequence modelling:

- Handle variable-length sequences
- Keep track of long-term dependencies
- Maintain information about the order as opposed to FFNN
- Share parameters across the network

Recurrent Neural Network

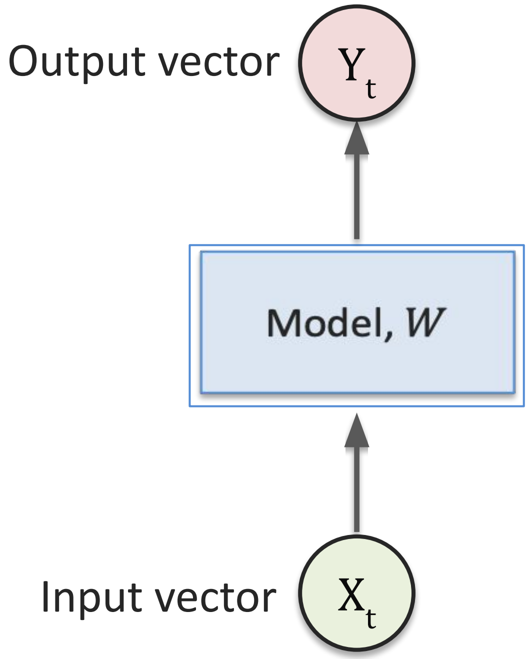
FEED FORWARD
NEURAL NETWORK



- Cannot maintain previous information

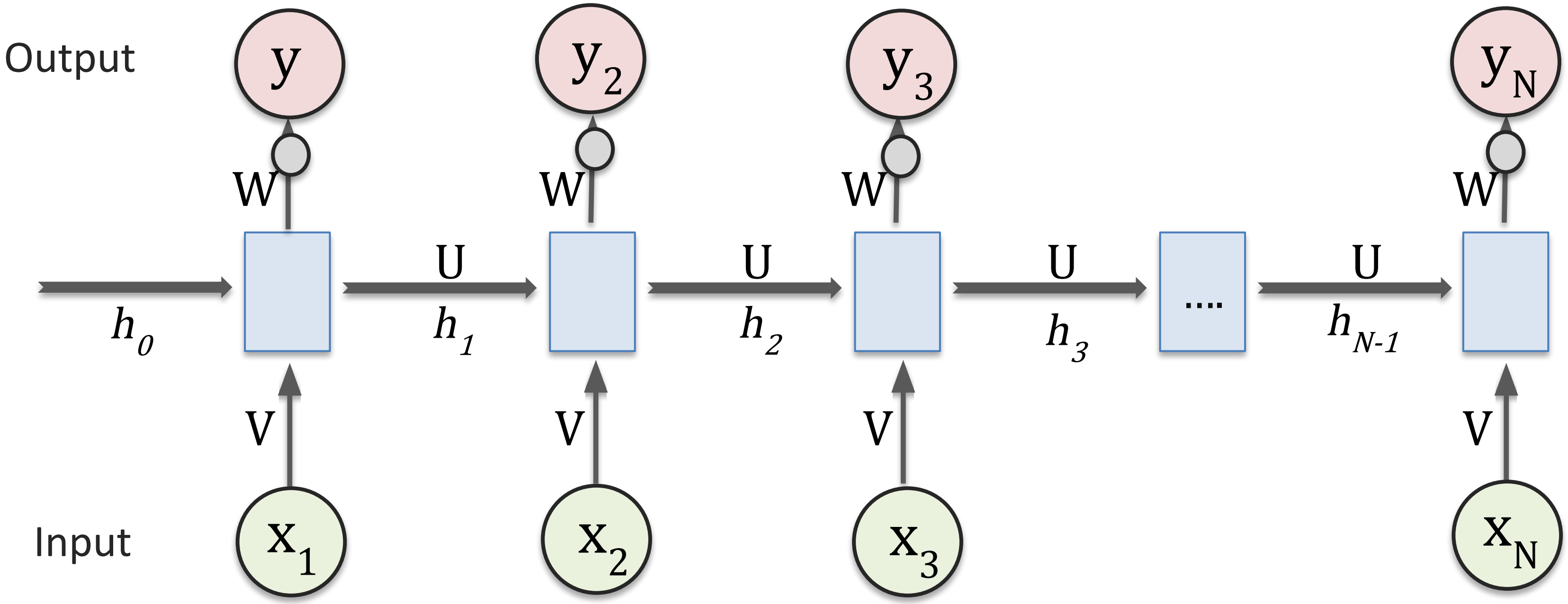
Recurrent Neural Network

FEED FORWARD NEURAL NETWORK



- Cannot maintain previous information

RECURRENT NEURAL NETWORK



The term **recurrent** comes from the fact that information is being passed from one time step to the next internally within the network.

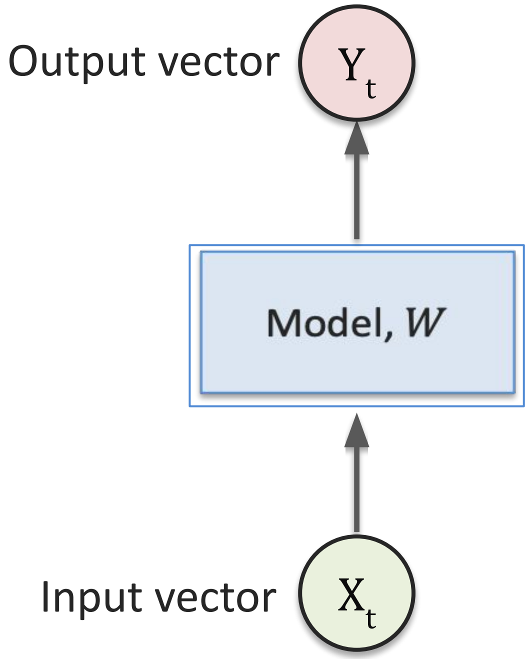
Network has loops for information to persist over time

Recurrent Neural Network

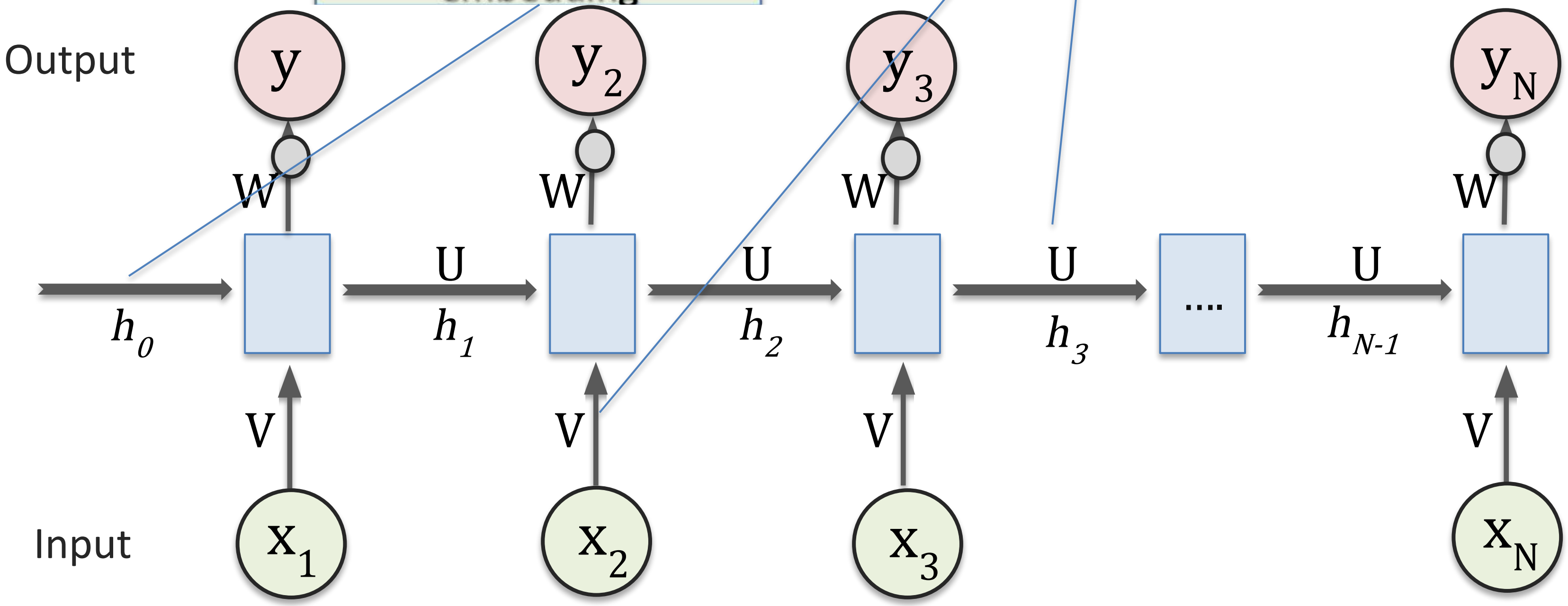
h_i:
memory
state
internal state
hidden state
latent variable
embedding

V, U, W: model parameters

FEED FORWARD
NEURAL NETWORK



T NEURAL NETWORK



- Cannot maintain previous information

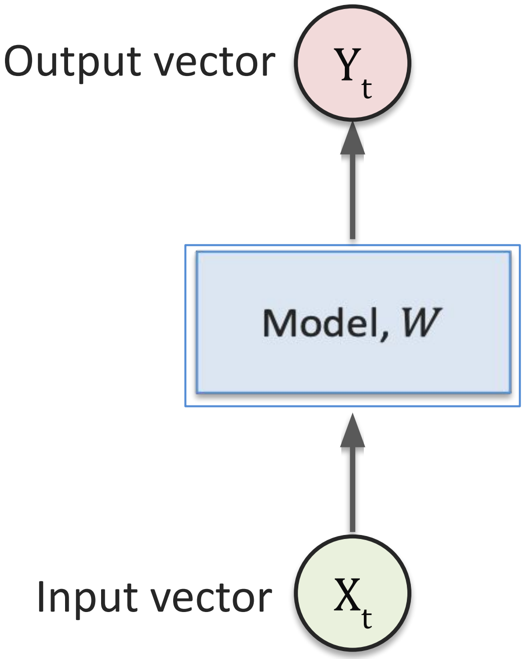
The term **recurrent** comes from the fact that information is being passed from one time step to the next internally within the network.

Network has loops for information to persist over time

Recurrent Neural Network

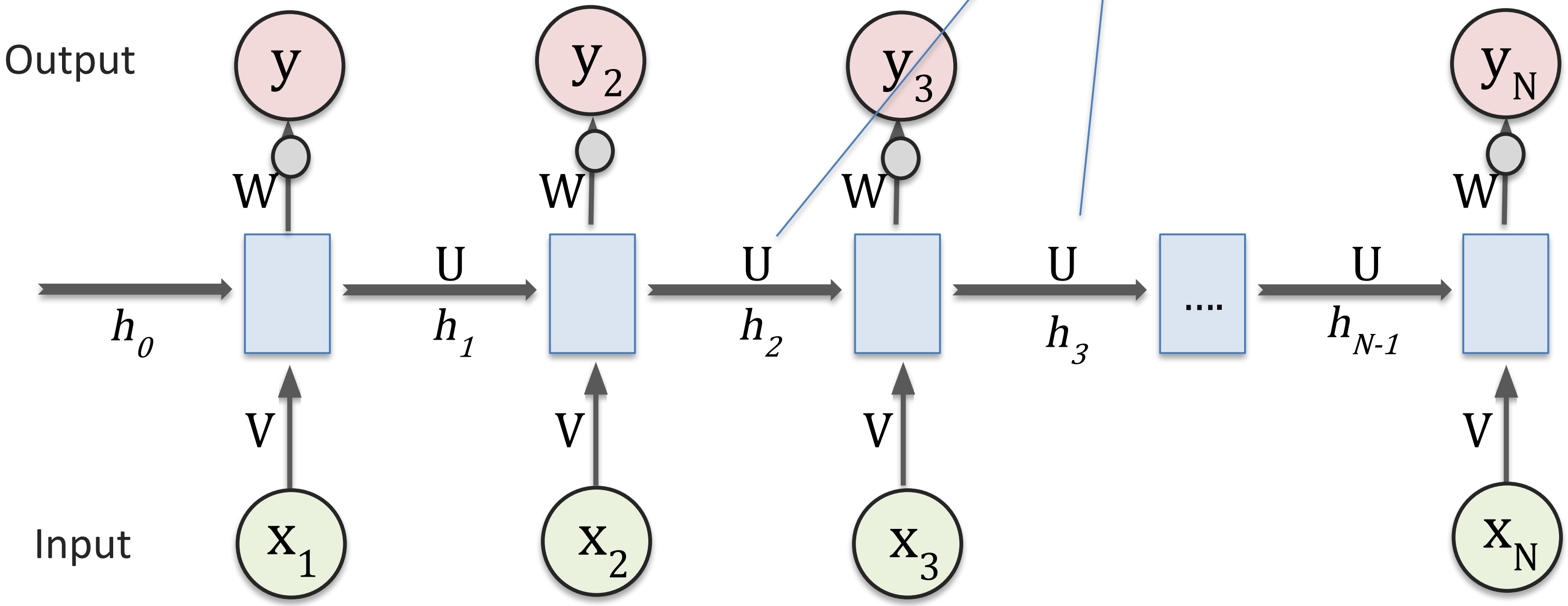
V, U, W: same for all times

FEED FORWARD NEURAL NETWORK



- Cannot maintain previous information

RECURRENT NEURAL NETWORK

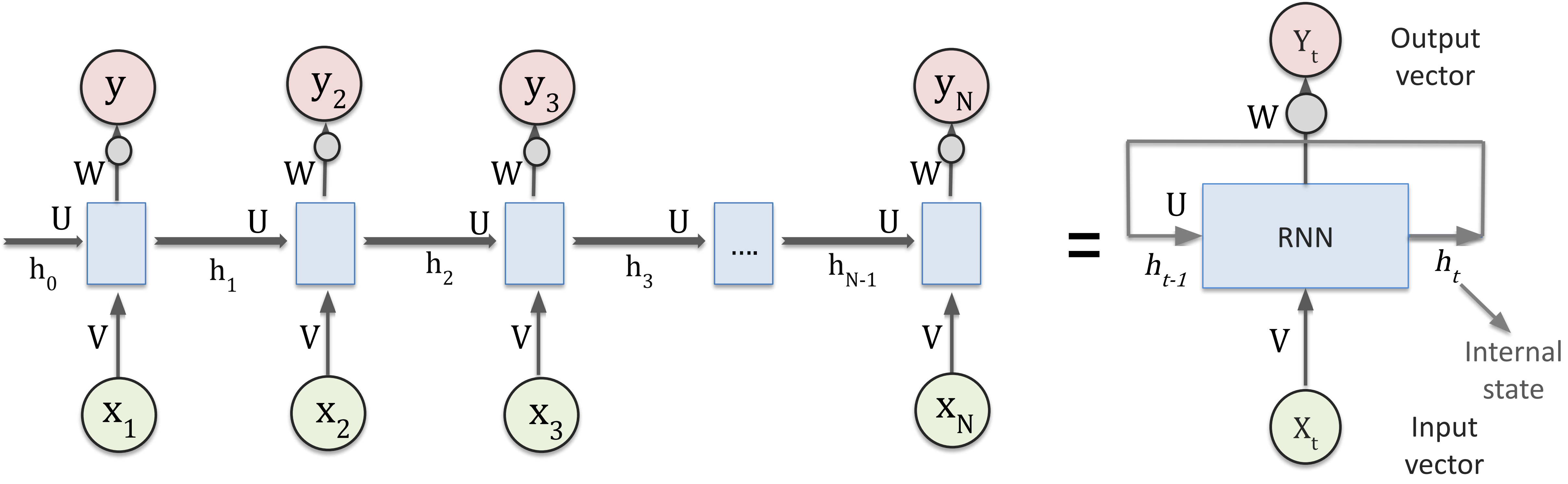


The term **recurrent** comes from the fact that information is being passed from one time step to the next internally within the network

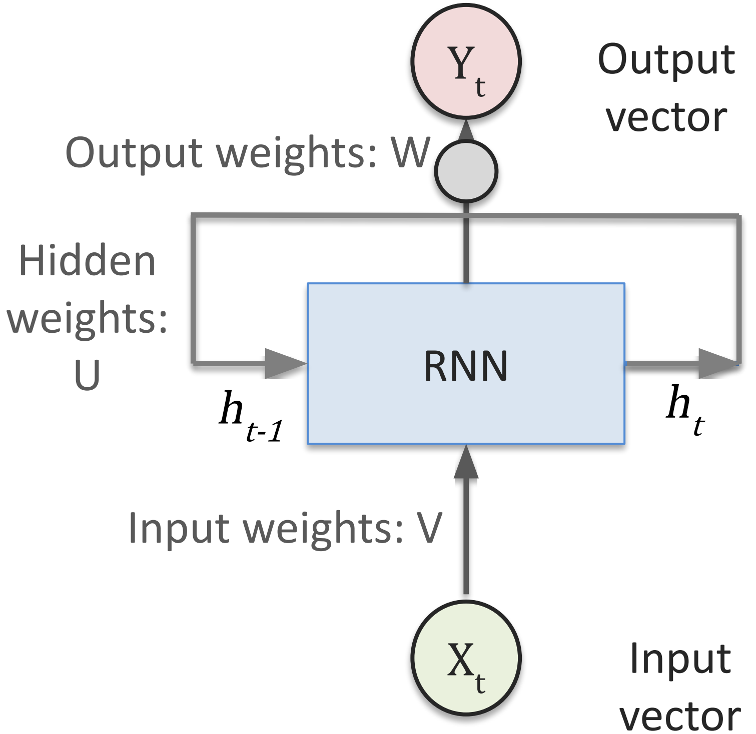
Network has loops for information to persist over time

Recurrent Neural Network

Alternative short representation:



Recurrent Neural Network

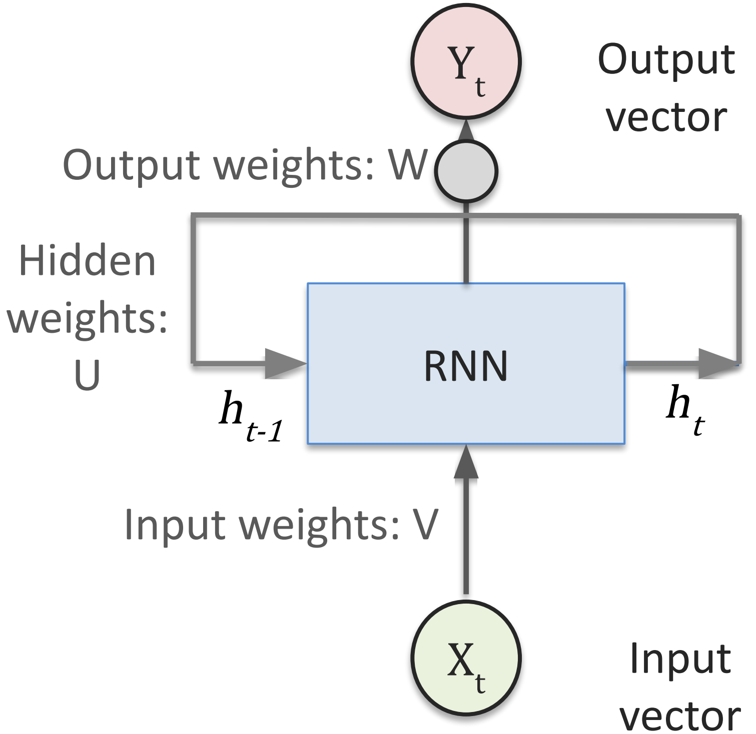


At each time step the RNN is fed the current input and the previous hidden state.

RNNs are governed by a **recurrence relation** applied at every time step for a given sequence.

$$h_t = f_{u,v}\,(\,h_{t-1}\,,\,x_t\,)$$

Recurrent Neural Network



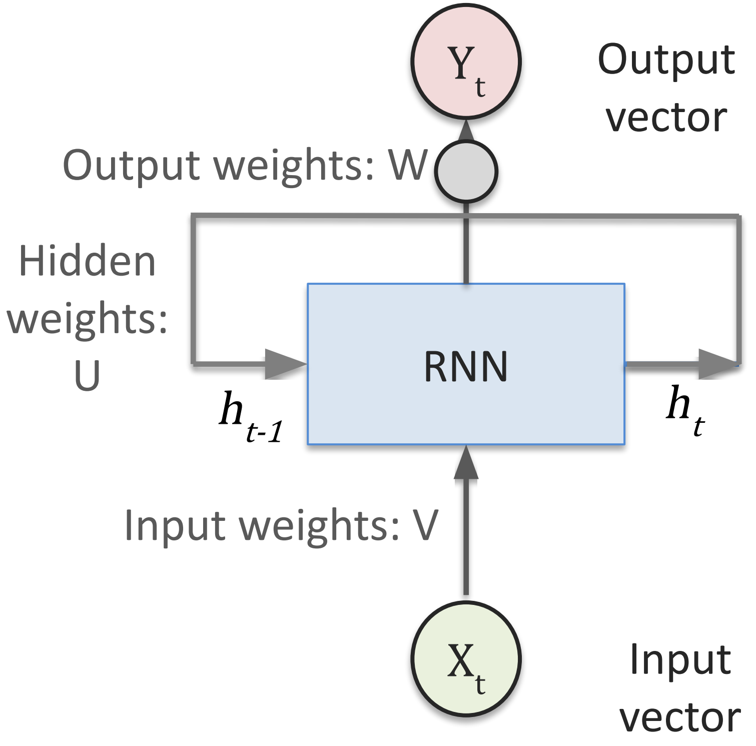
RNNs are governed by a **recurrence relation** applied at every time step for a given sequence.

$$\boxed{h_t} = \boxed{f_{u,v}}(\boxed{h_{t-1}}, \boxed{x_t})$$

State | Function parameterized by u,v | Old State | Input vector at time step t

The function $f_{u,v}$ and the parameters used for all time steps are learned during training.

Recurrent Neural Network



RNNs are governed by a **recurrence relation** applied at every time step for a given sequence.

Multiple names:
- Hidden state
- State
- Encoding
- Embedding

$$\boxed{h_t} = \boxed{f_{u,v}}(\boxed{h_{t-1}}, \boxed{x_t})$$

Stat

Function parameterized by u,v

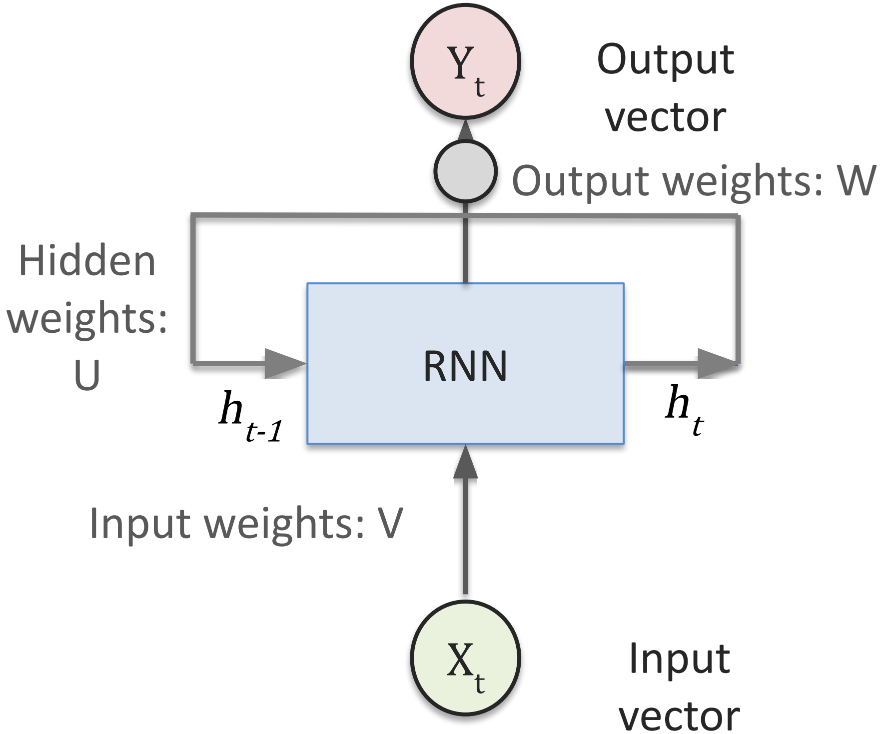
Old State

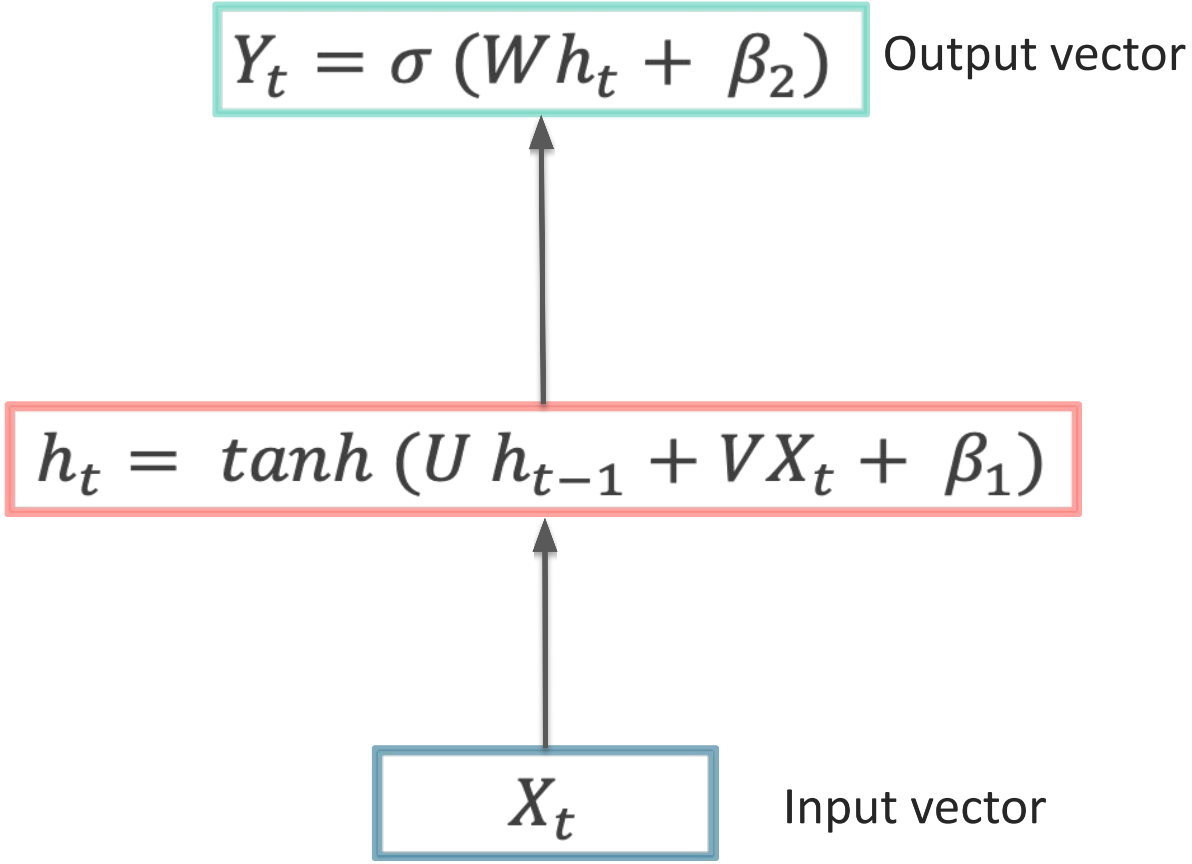
Input vector at time step t

We often ignore to mention the bias here. It should be: $f_{u,v,\beta}()$

The function $f_{u,v}$ and the parameters used for all time steps are learned during training.

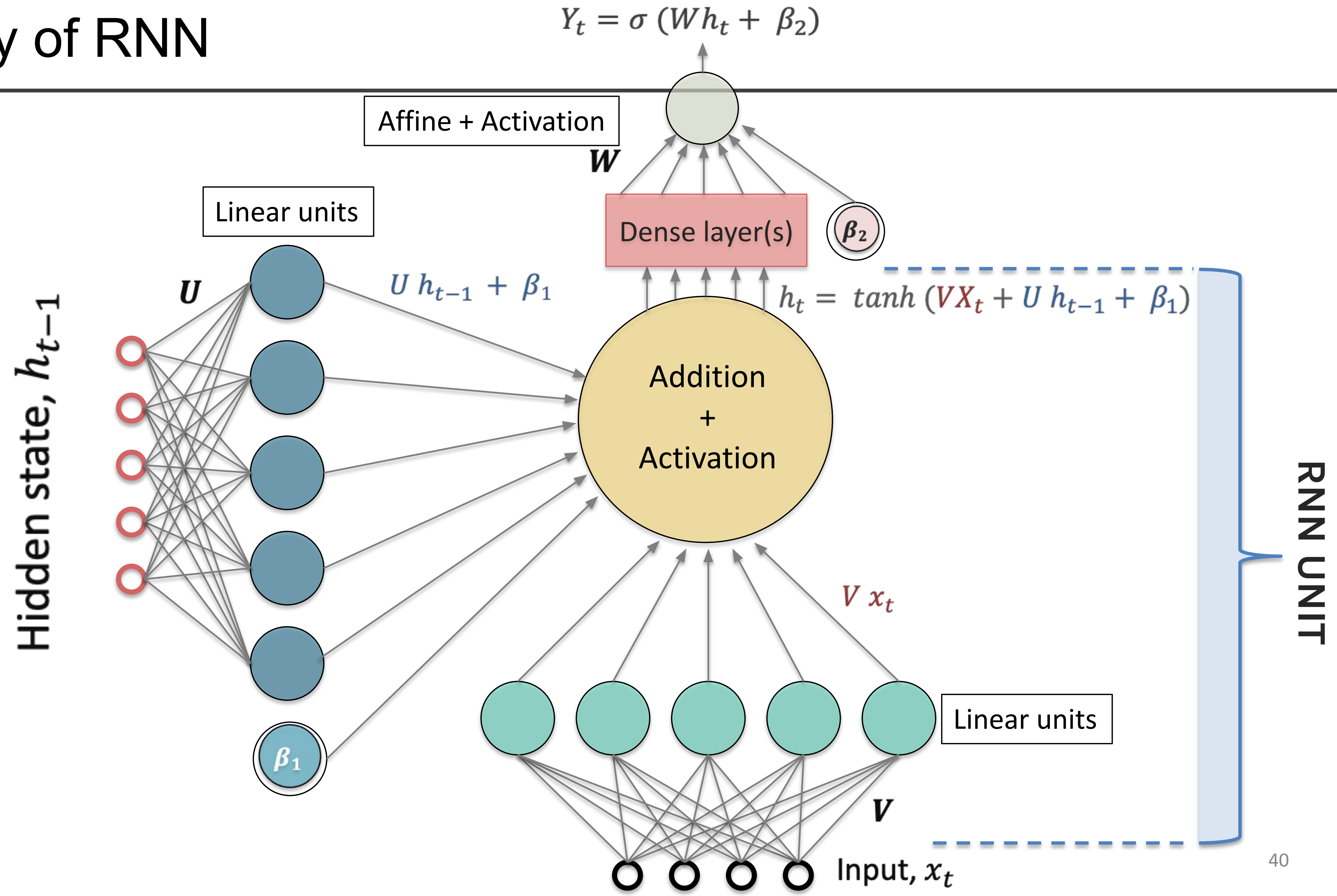
Recurrent Neural Network



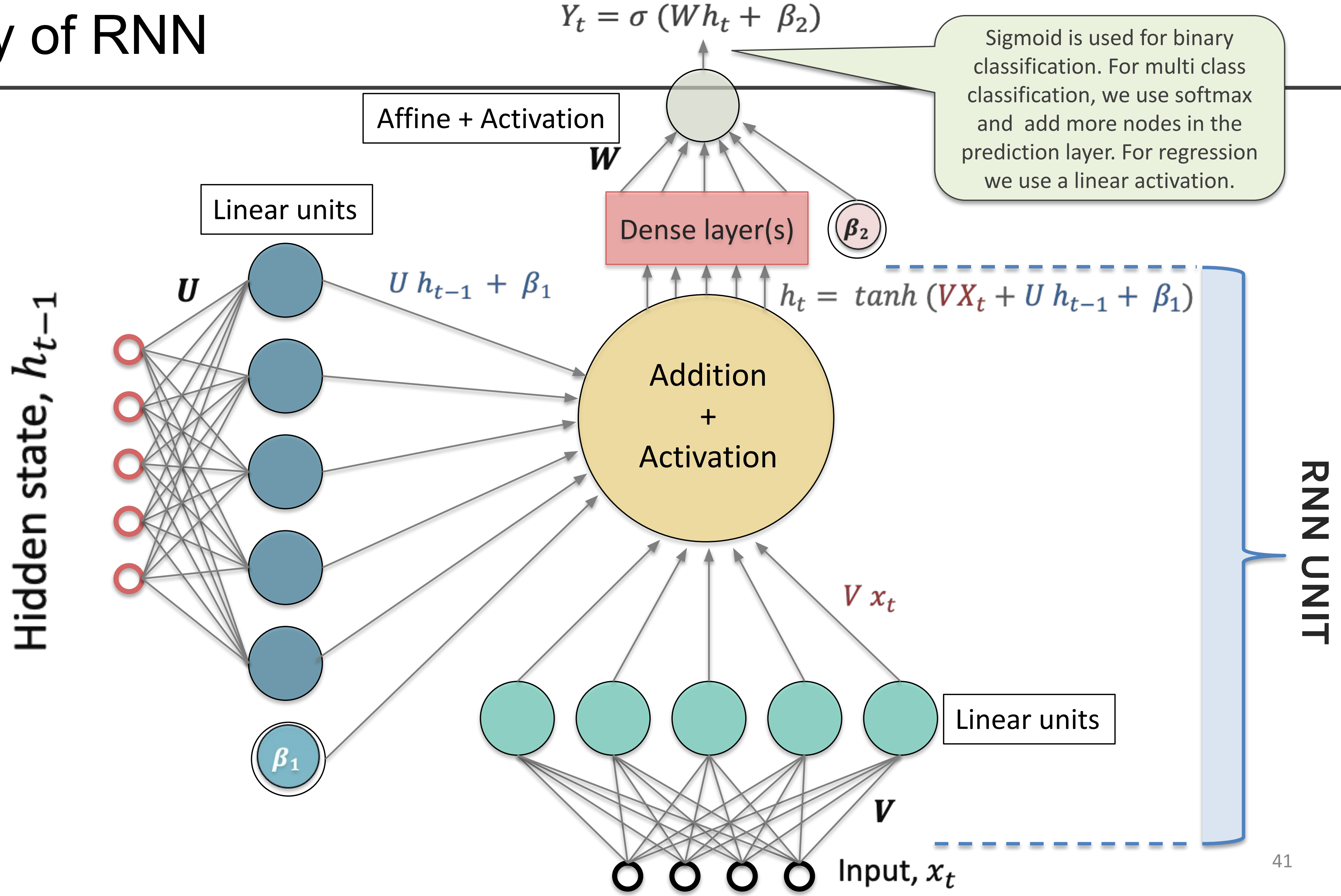


U, V and W are three different weight matrices learned during training

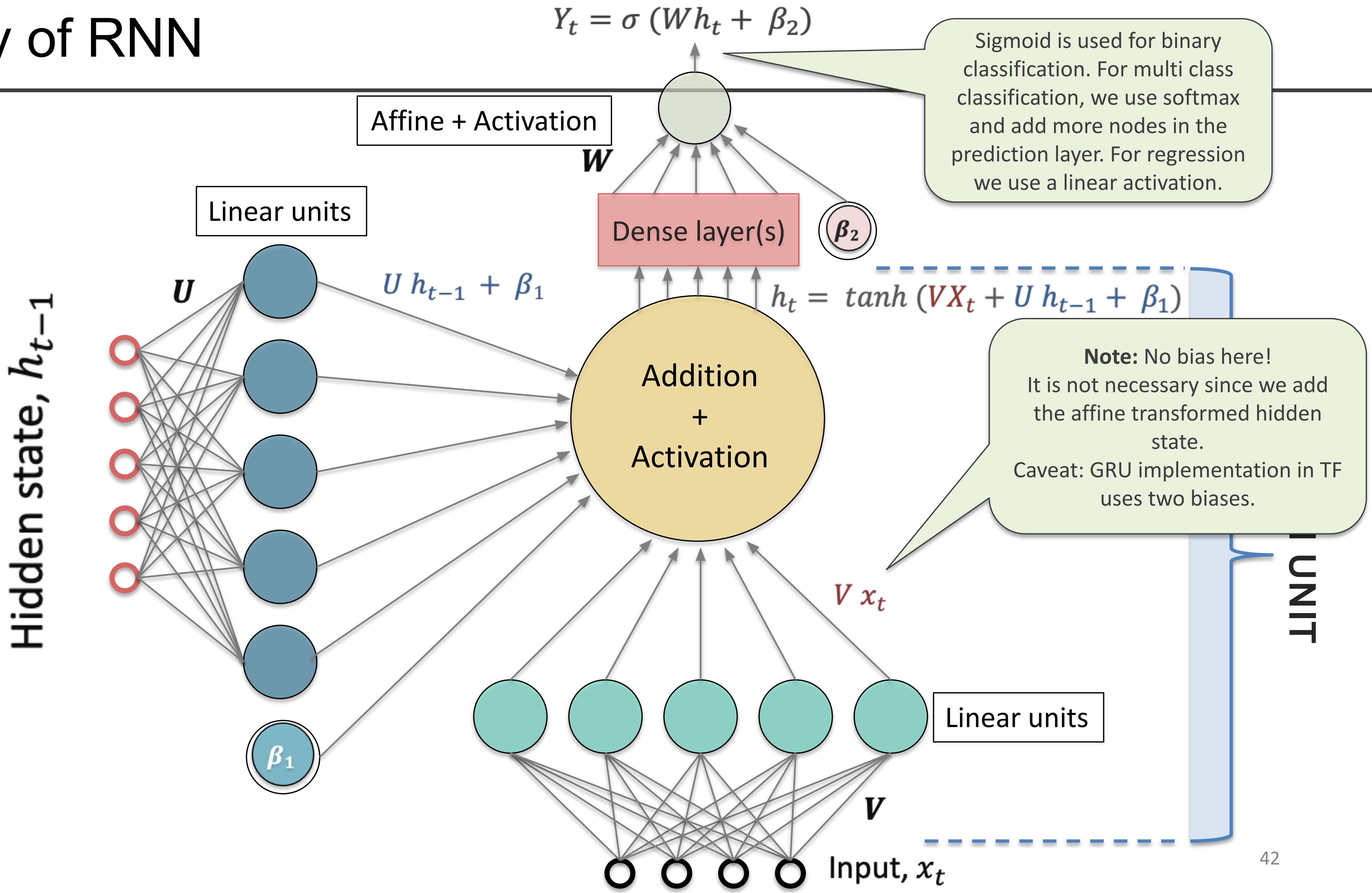
Anatomy of RNN



Anatomy of RNN

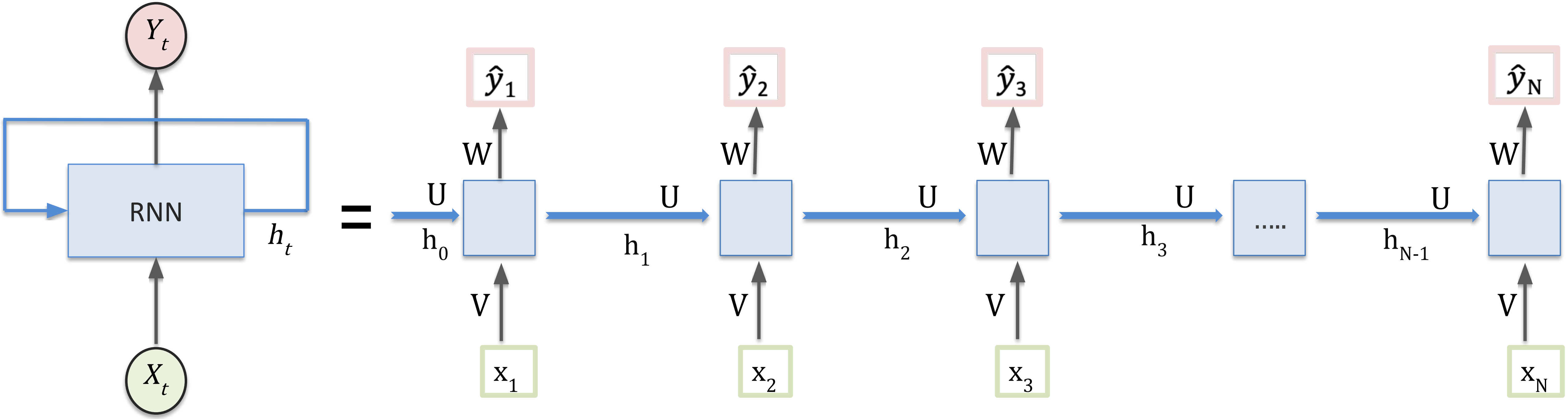


Anatomy of RNN



Training RNNs

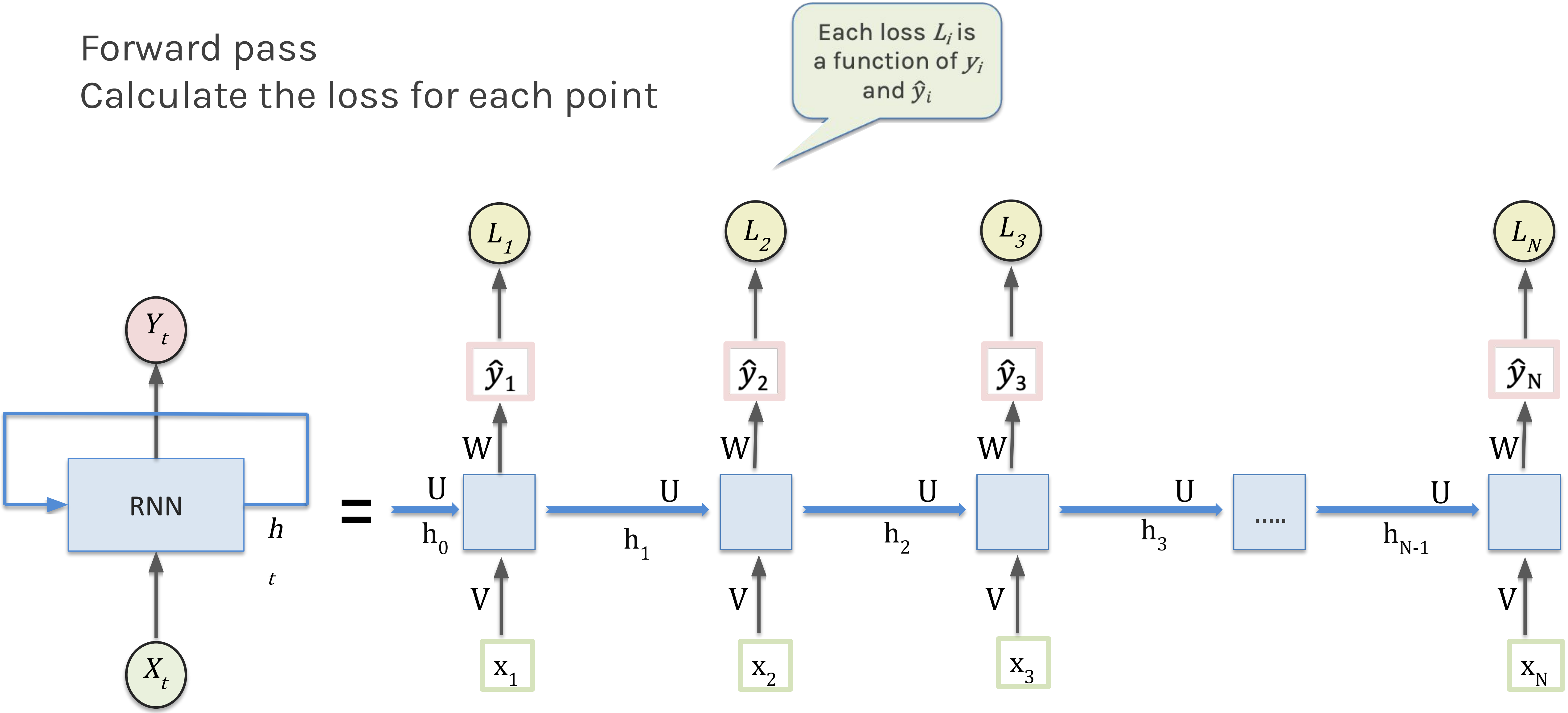
Forward pass



Training RNNs

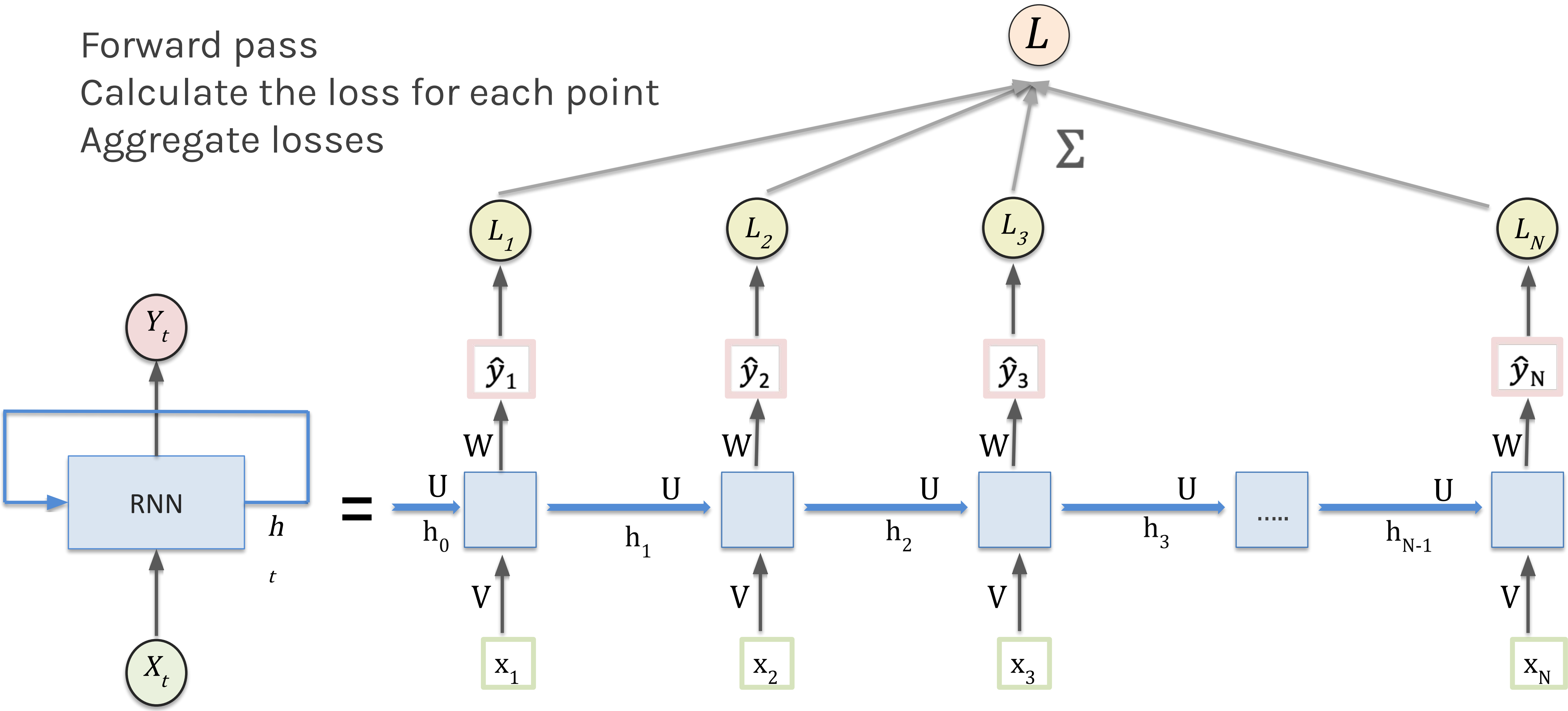
Forward pass

Calculate the loss for each point



Training RNNs

Forward pass
Calculate the loss for each point
Aggregate losses

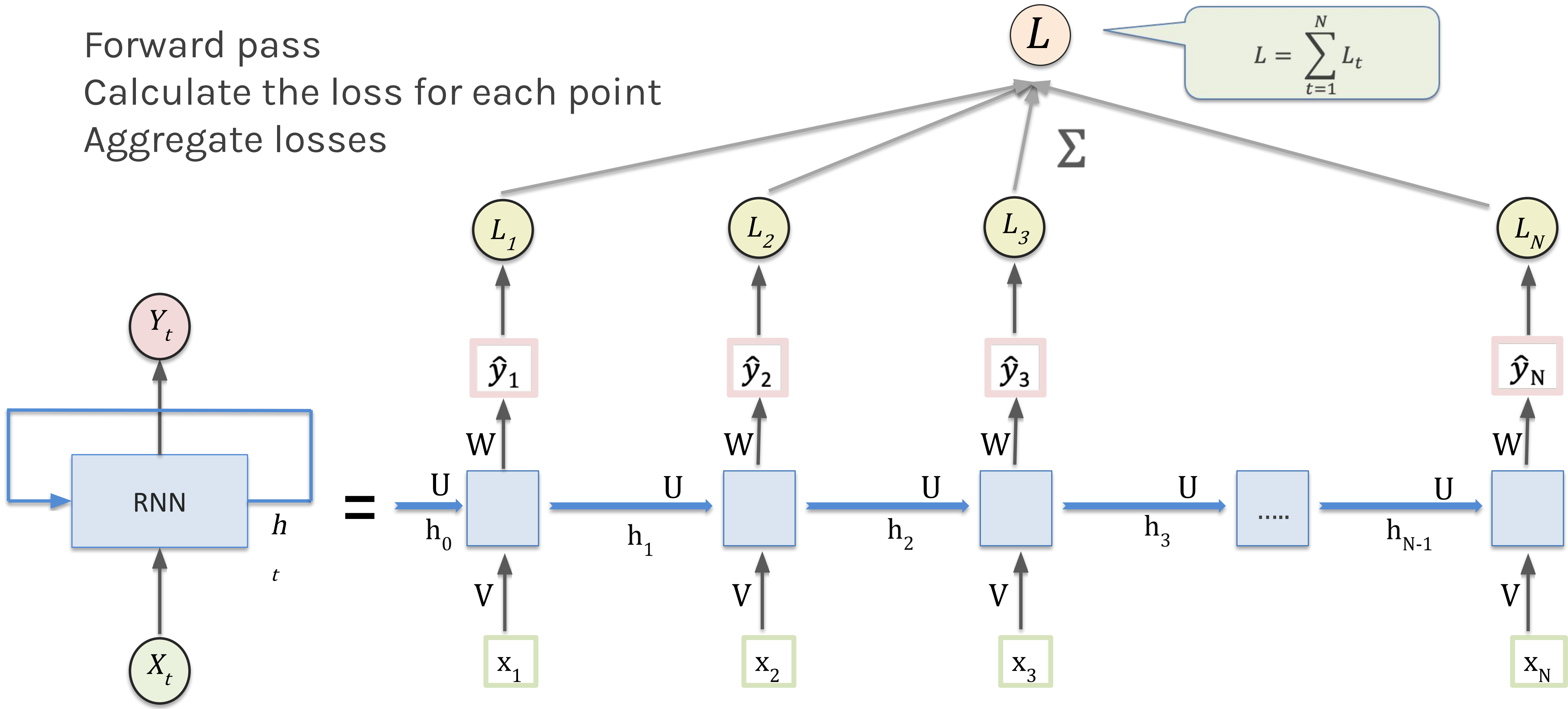


Training RNNs

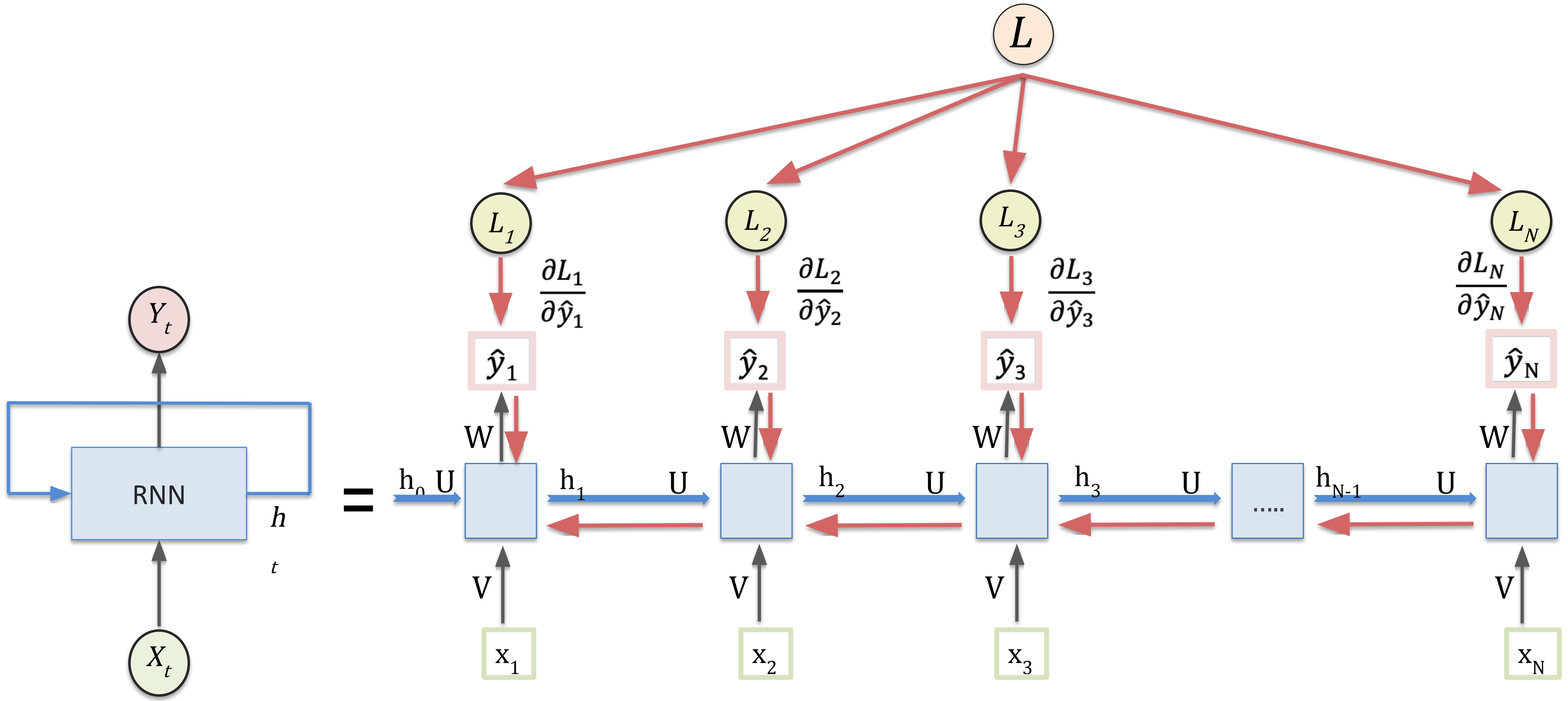
Forward pass

Calculate the loss for each point

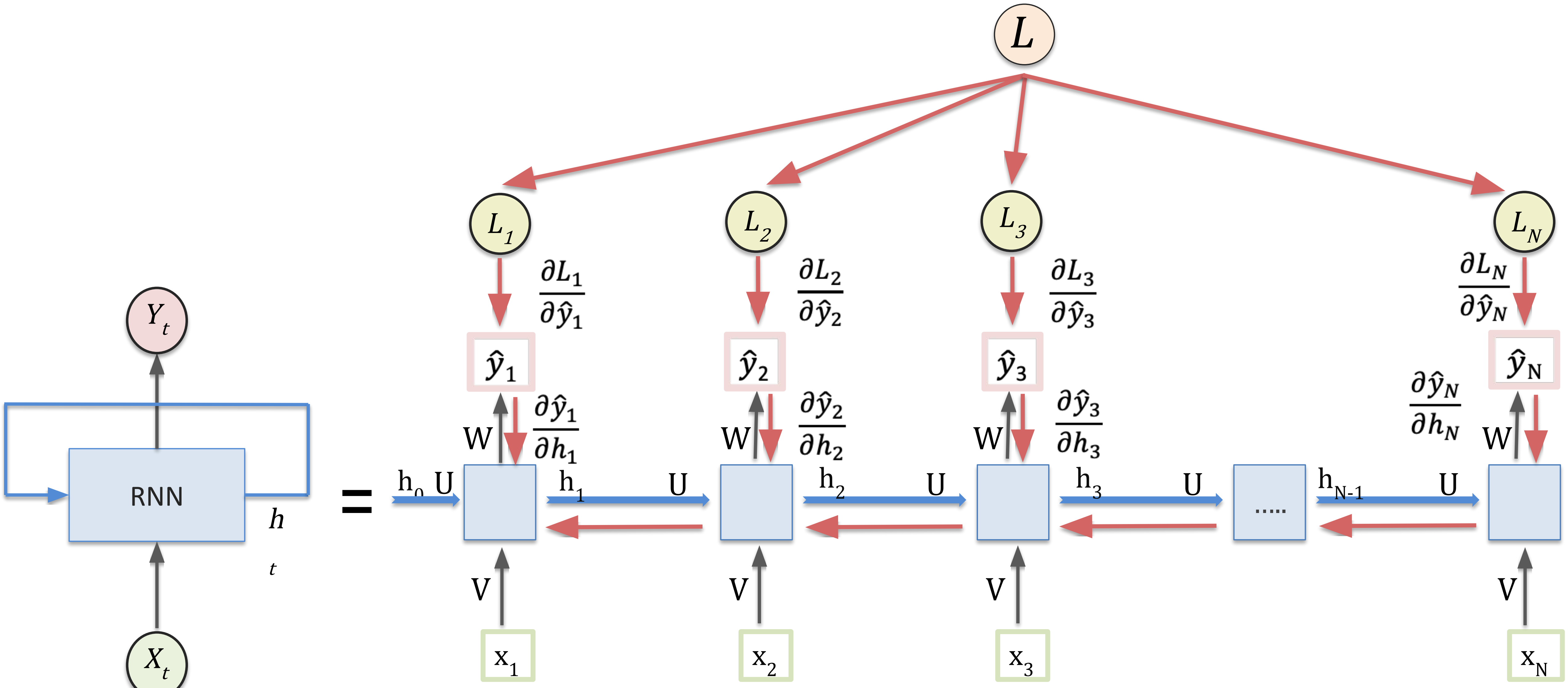
Aggregate losses



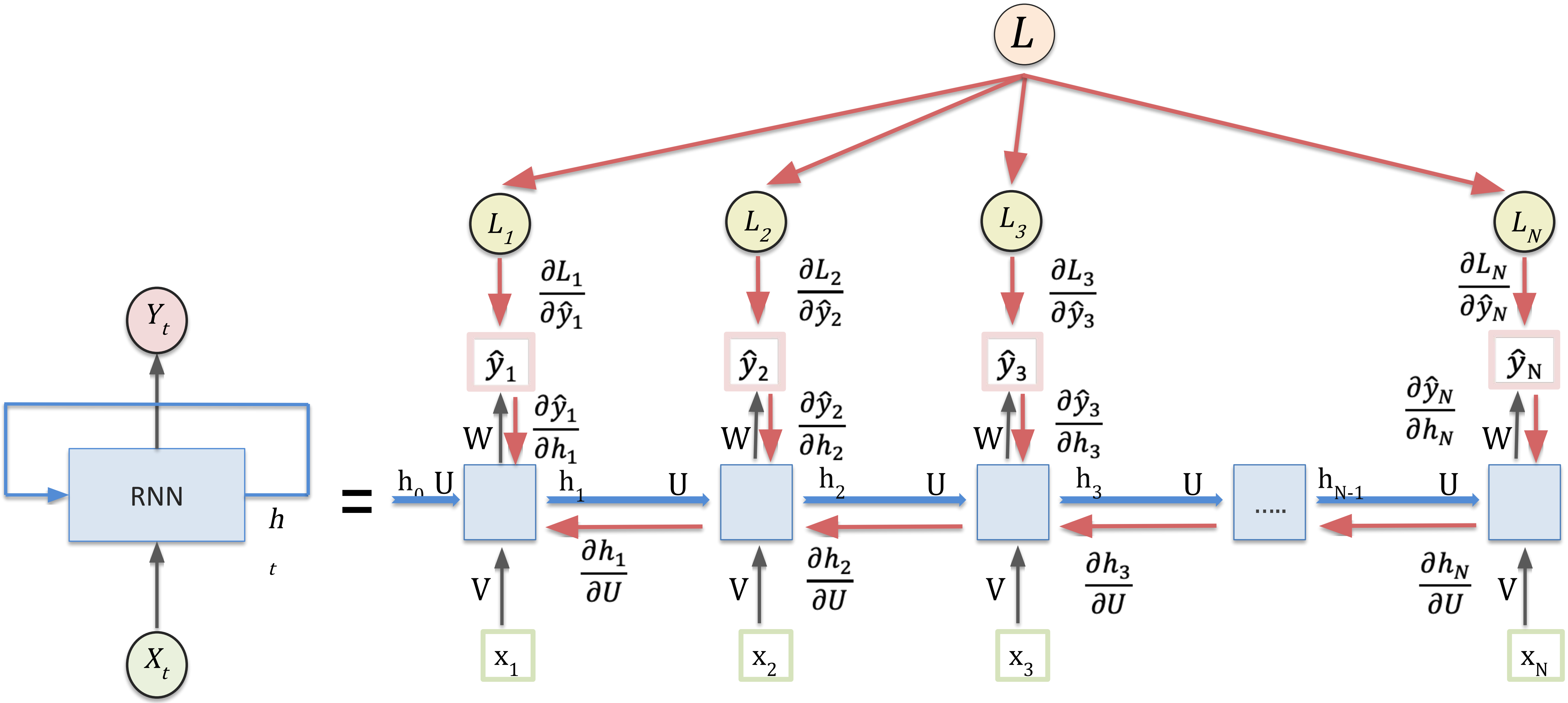
Training RNNs: Backpropagation



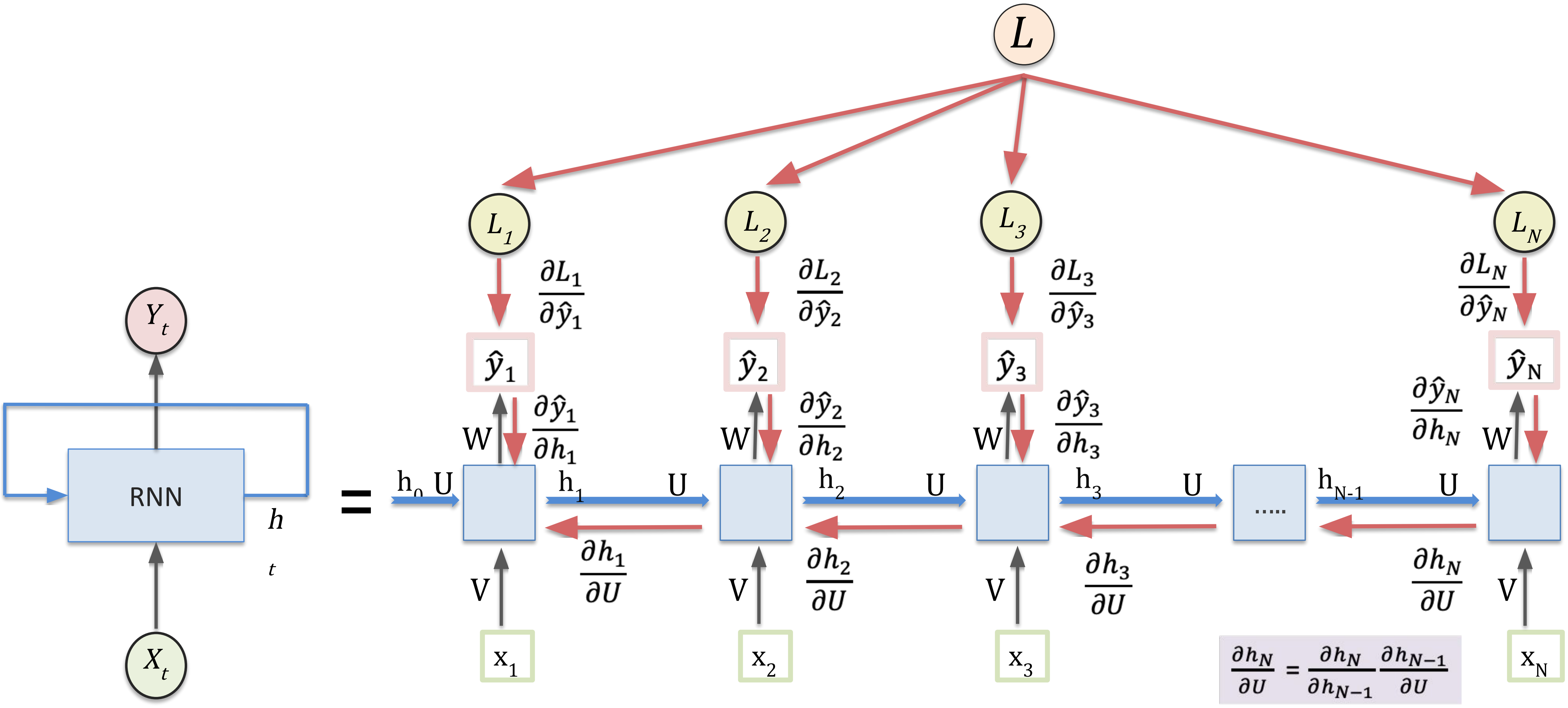
Training RNNs: Backpropagation



Training RNNs: Backpropagation



Training RNNs: Backpropagation



Training RNNs: Backpropagation

During backpropagation for each parameter at each time step *i*, a gradient is computed.

The individual gradients computed are then averaged at time step *t* and used to update the entire network.

The error flows back in time.

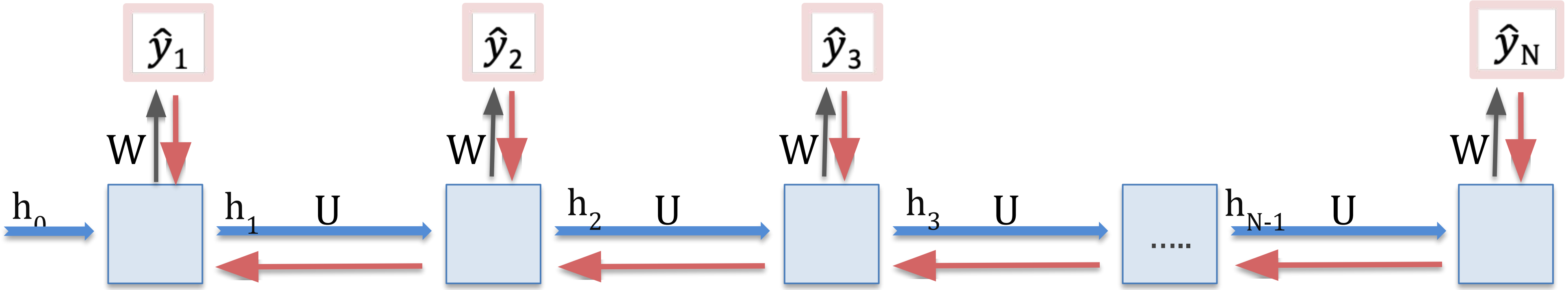
$$\frac{dL}{dU} = \sum_t \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial U}$$

$$\frac{\partial h_t}{\partial U} = \sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial U}$$

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \dots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \left(\frac{dh_t}{dU} + \frac{dh_t}{dh_{t-1}} \frac{dh_{t-1}}{dU} + \frac{dh_t}{dh_{t-1}} \frac{dh_{t-1}}{dh_{t-2}} \frac{dh_{t-2}}{dU} + \cdots \right)$$

Training RNNs: Backpropagation Issues



For longer sentences, we must backpropagate through more time steps.

This requires the gradient to be multiplied many times which causes the following issues:

If many values < 1, then the product, i.e., the gradient, will be close to zero. This is called the **vanishing gradient problem**.

This causes the parameters to update very slowly.

If many values > 1, then the product, i.e., the gradient, will explode. This is called the **exploding gradient problem**.

This causes an overflow problem.

Training RNNs: Backpropagation Issues

RNN Issues addressed by:

- GRU
- LSTMs
- Attention

Outline

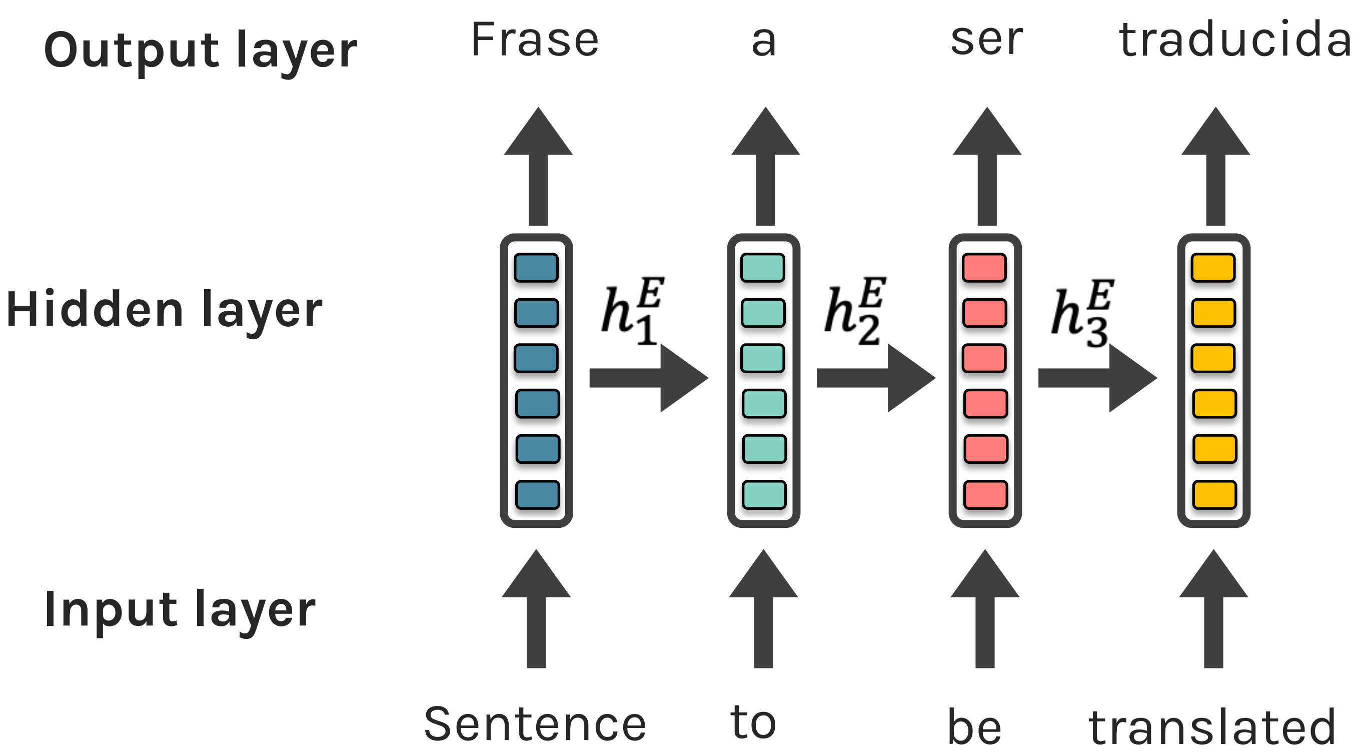
1. What are Language Models
2. Neural Networks for Language Modeling
3. Recurrent Neural Network
4. **Seq2Seq + Attention**
5. Self Attention
6. Transformers
7. Tutorial: SOTA Language Models

Sequence-to-Sequence (seq2seq)

- If our input is a sentence in Language A, and we wish to translate it to Language B, it is clearly suboptimal to translate word by word (like our current models are suited to do).

Sequence-to-Sequence (seq2seq)

- If our input is a sentence in Language A, and we wish to translate it to Language B, it is clearly suboptimal to translate word by word (like our current models are suited to do).



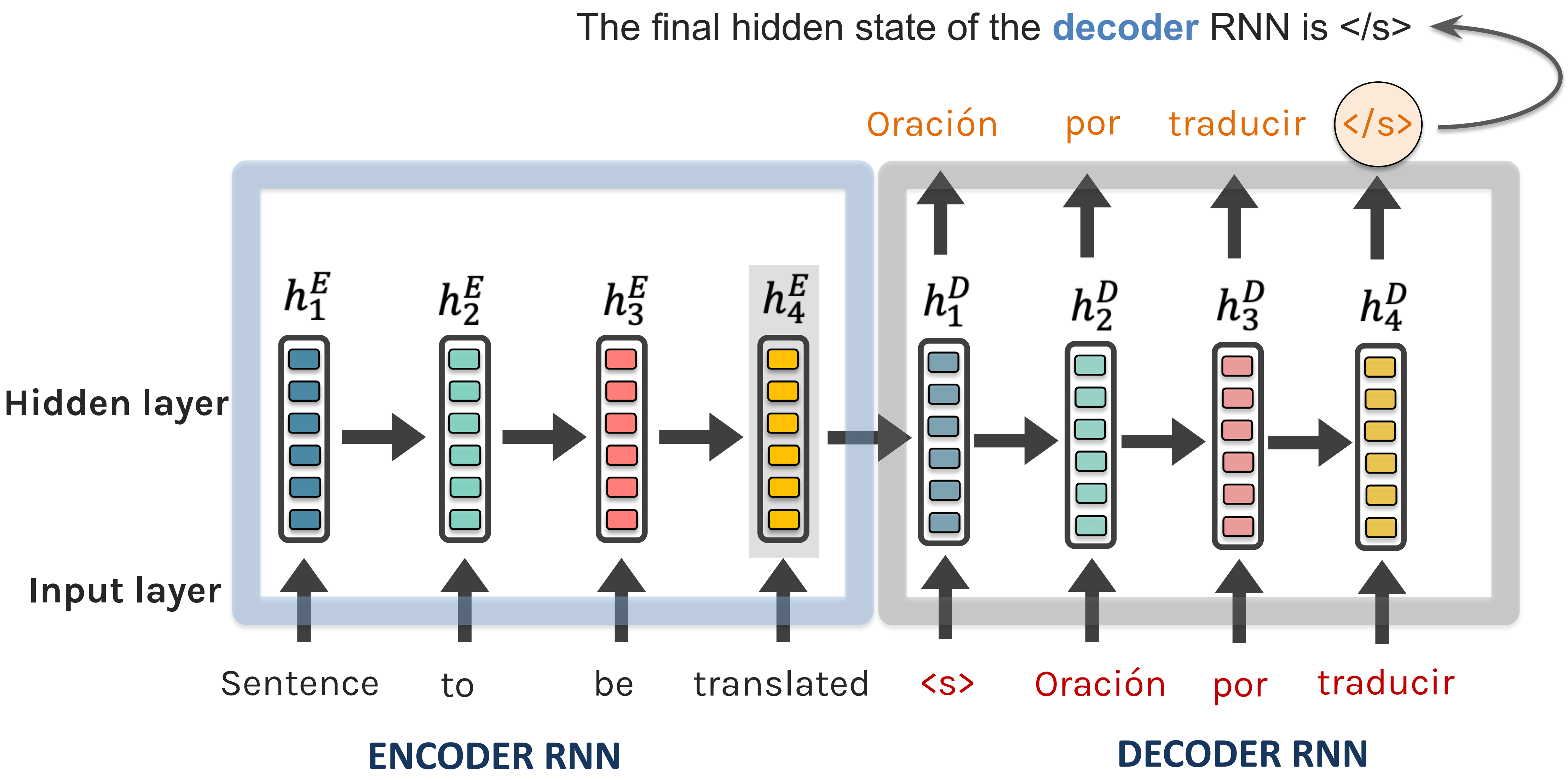
Sequence-to-Sequence (seq2seq)

- Instead, let a ***sequence*** of tokens be the unit that we ultimately wish to work with (a sequence of length **N** may emit a sequences of length **M**)

- Seq2seq models are comprised of **2 RNNs**: 1 encoder, 1 decoder

Sequence-to-Sequence (seq2seq)

The final hidden state of the **encoder** RNN is the initial state of the decoder RNN



Sequence-to-Sequence (seq2seq)

See any issues with this traditional **seq2seq** paradigm?

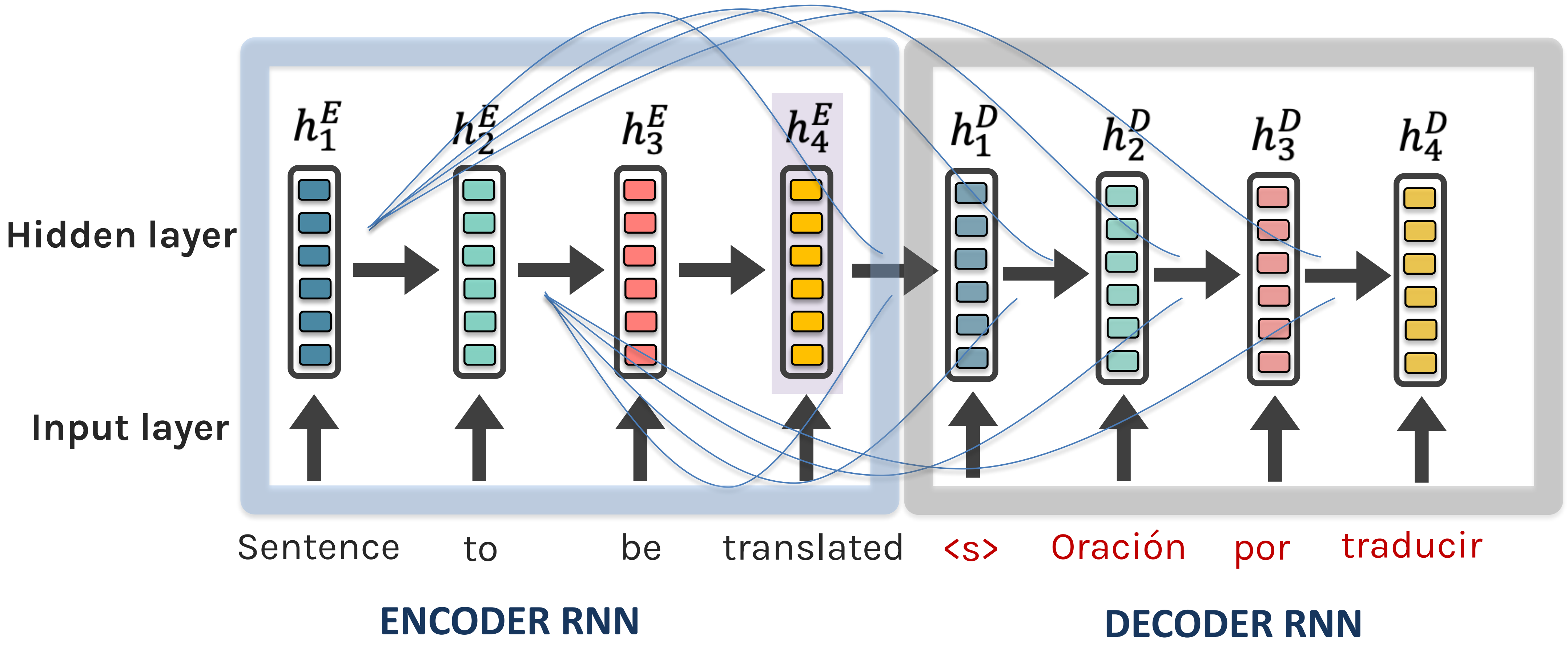
Sequence-to-Sequence (seq2seq)

It's crazy that the entire "meaning" of the 1st sequence is expected to be packed into one embedding, and that the encoder then never interacts w/ the decoder again. Hands free!

What other alternatives can we have?

Sequence-to-Sequence (seq2seq)

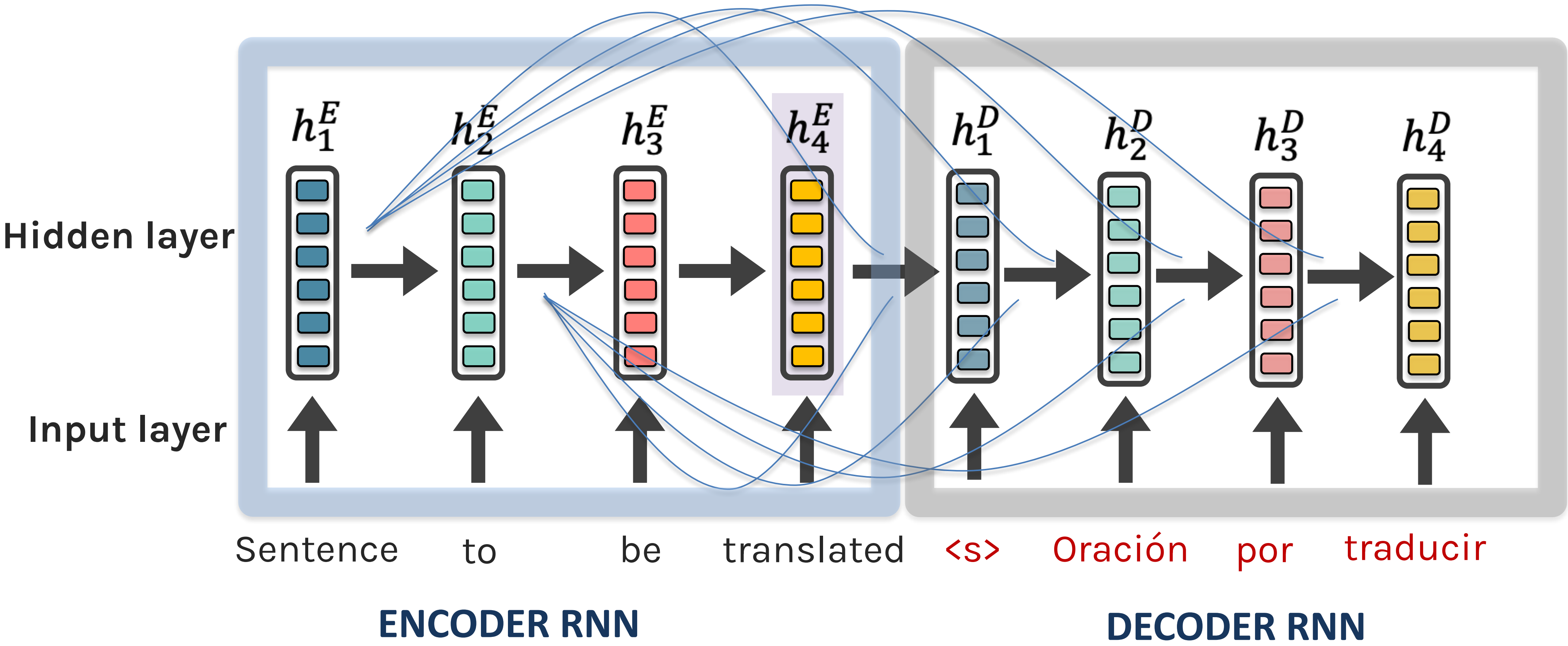
One alternative would be to pass every state of the encoder to every state of the decoder.



Sequence-to-Sequence (seq2seq)

But how much context is enough?

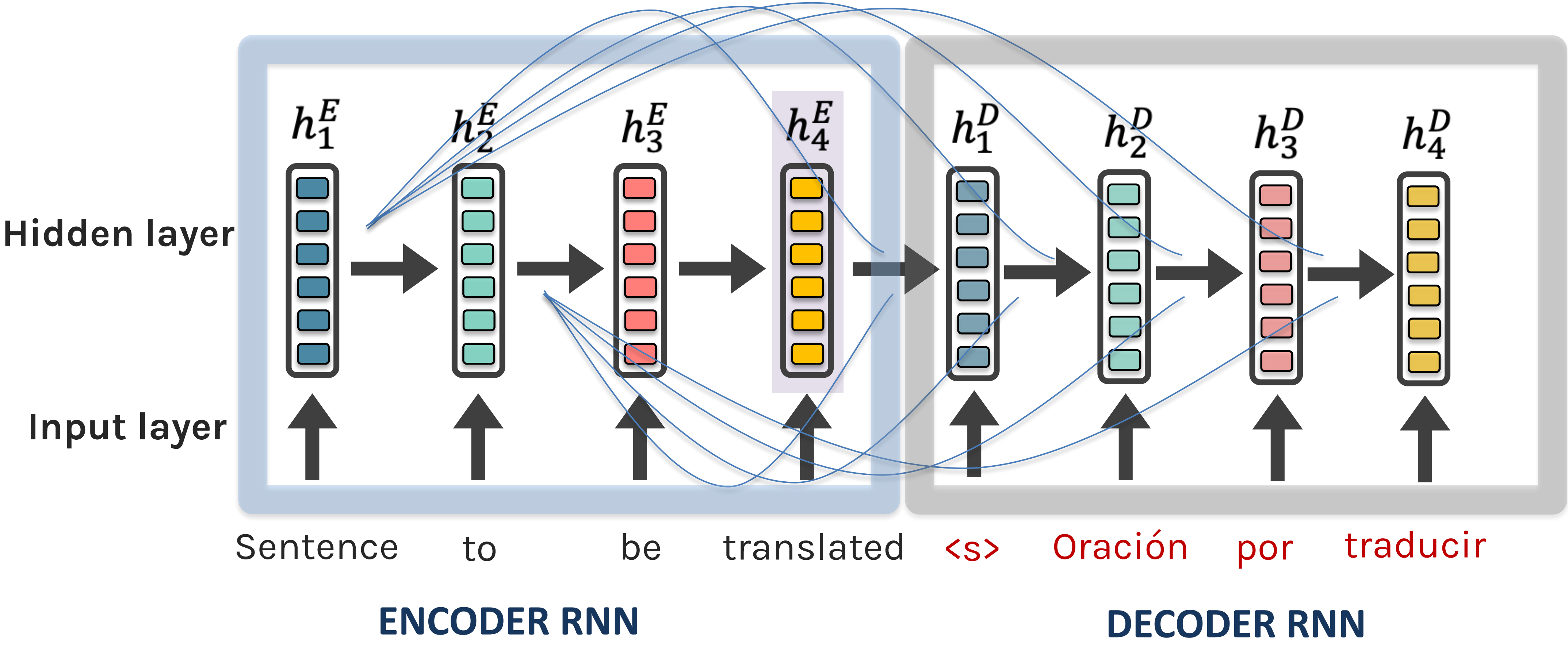
The number of states to send to the decoder can get extremely large.



Sequence-to-Sequence (seq2seq)

Another alternative could be to weight every state

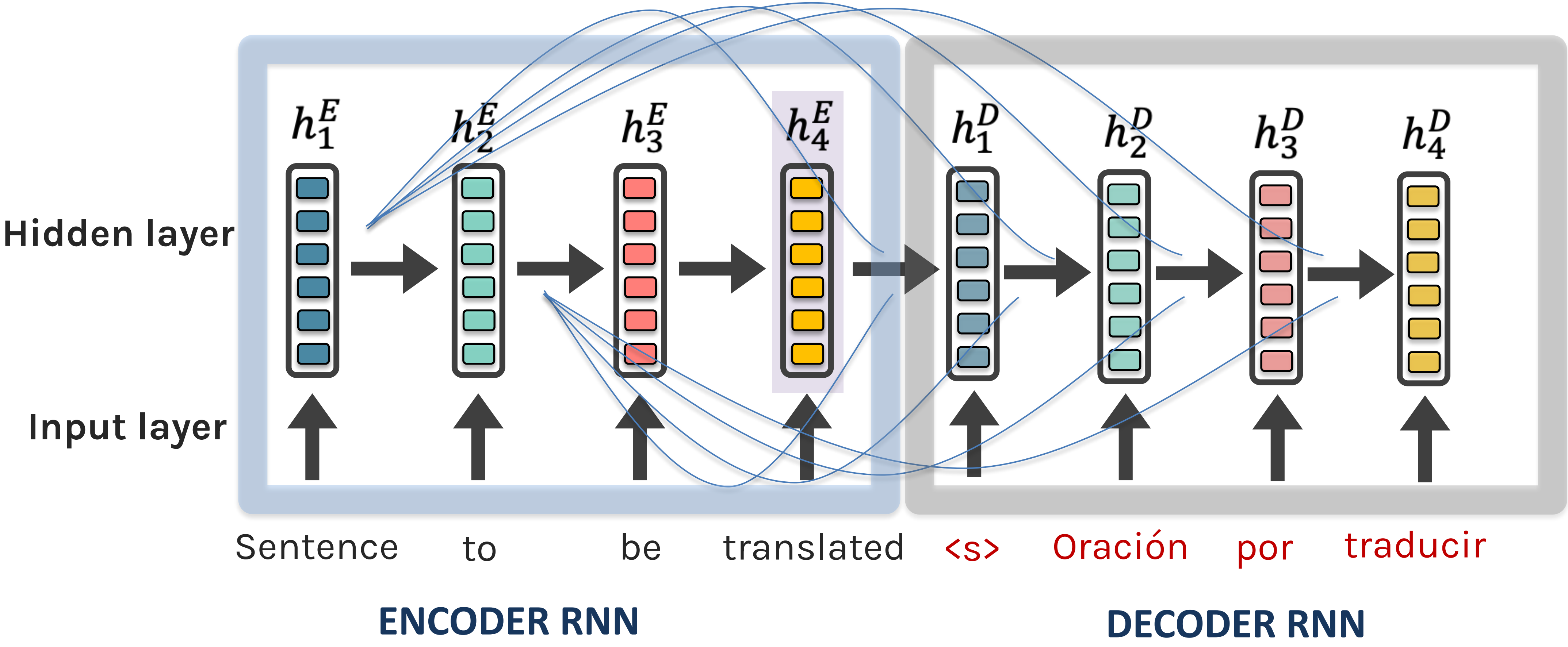
$$h_j^D = \tanh(Vx_j + Uh_{j-1}^D + W_1 h_1^E + W_2 h_2^E + W_3 h_3^E + \cdots)$$



Sequence-to-Sequence (seq2seq)

The **real problem** with this approach is that the weights $W_1, W_2, W_3, \ldots$ are fixed

$$h_j^D = \tanh(Vx_j + Uh_{j-1}^D + W_1 h_1^E + W_2 h_2^E + W_3 h_3^E + \cdots)$$



Sequence-to-Sequence (seq2seq)

The real problem with this approach is that the weights $W_1, W_2, W_3, \dots$ are fixed

$$h_j^D = \tanh(Vx_j + Uh_{j-1}^D + W_1 h_1^E + W_2 h_2^E + W_3 h_3^E + \cdots)$$

What we want instead is for the decoder, at each step, to decide how much attention to pay to each of the encoder's hidden states?

$$W_{ji} = g(\, h_i^E, X_j^D, h_{j-1}^D \,)$$

where g is a function parameterized by all the states of the encoder, the current input to the decoder and the state of the decoder. W indicates how much attention to pay to each hidden state of the encoder.

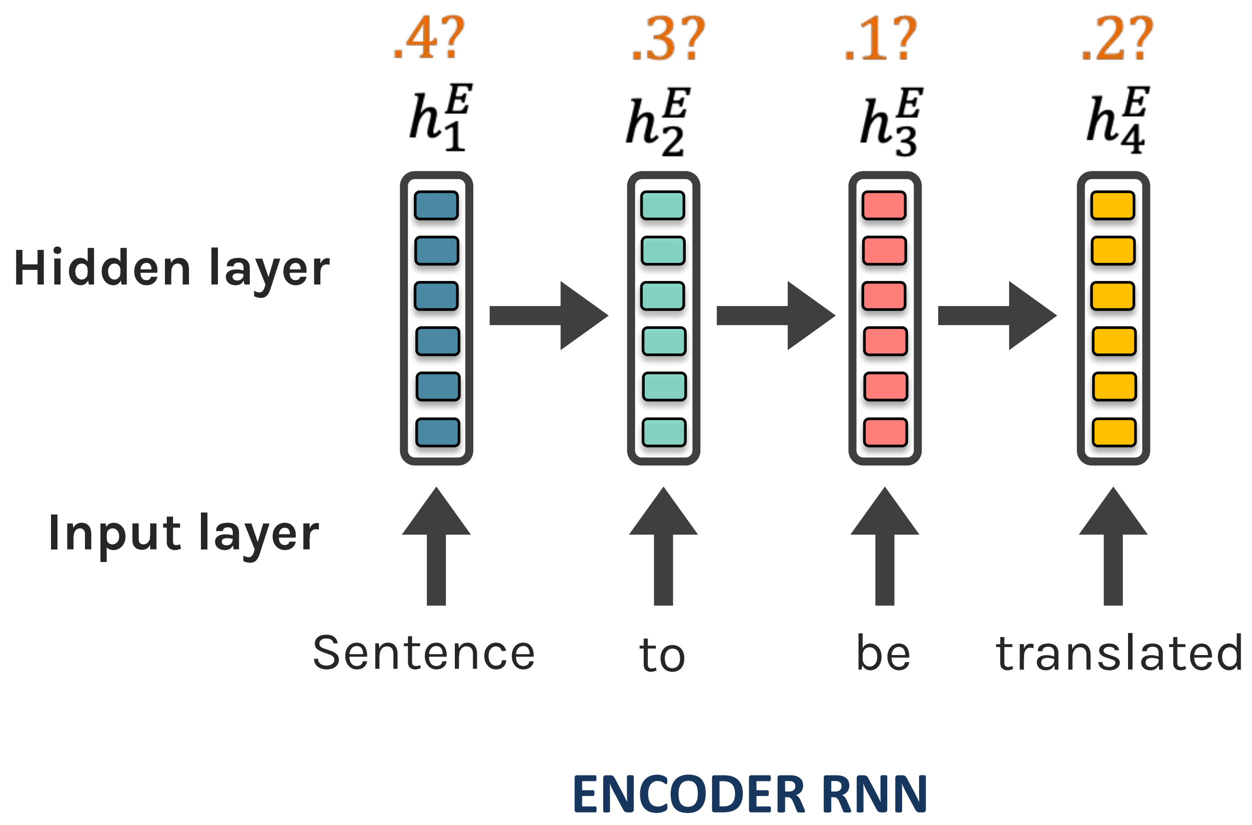
The function $\boldsymbol{g}$ gives what we call the attention.

Seq2Seq + Attention

Q: How do we determine how much attention to pay to each of the encoder's hidden states i.e. determine g(.) ?

Seq2Seq + Attention

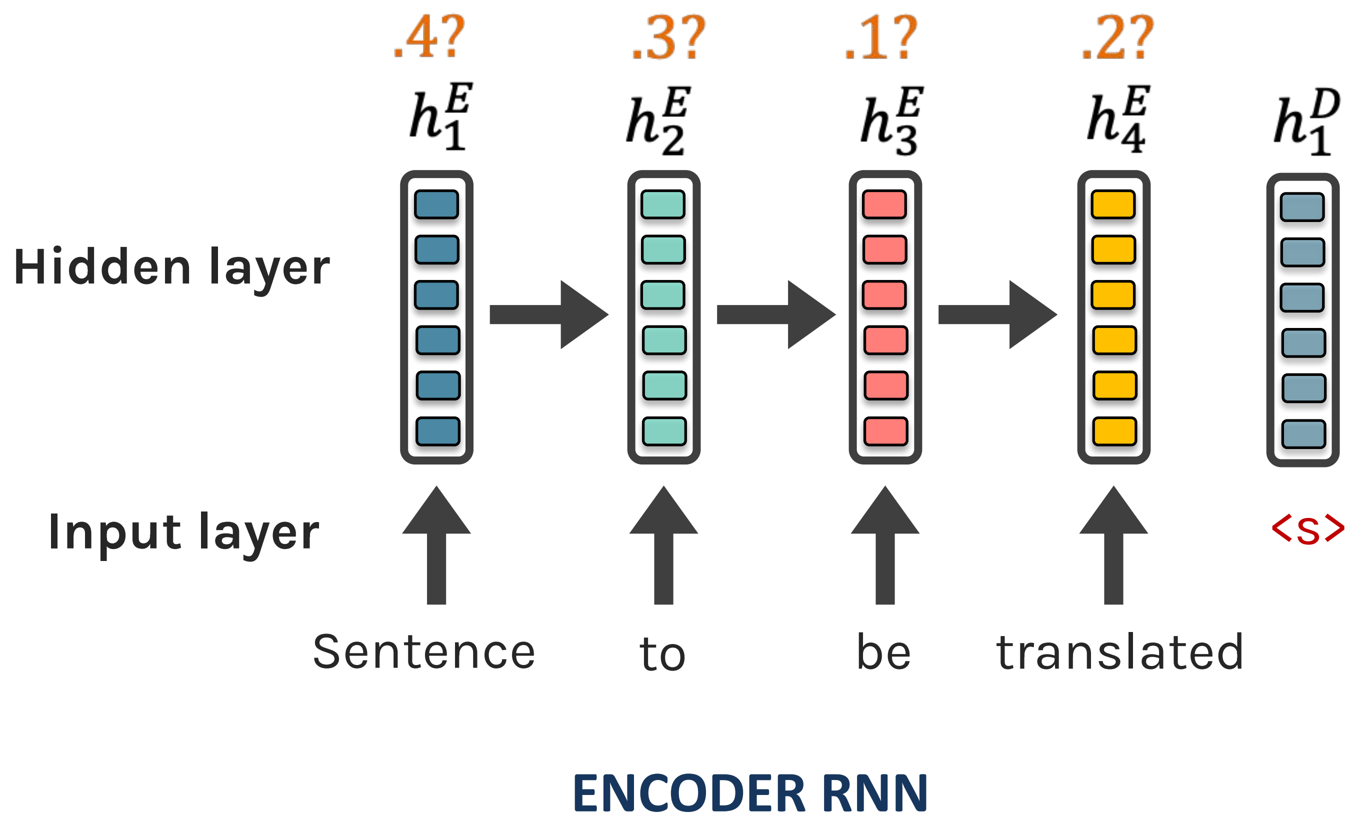
Q: How do we determine how much attention to pay to each of the encoder's hidden states i.e. determine g(.) ?



Seq2Seq + Attention

Q: How do we determine how much attention to pay to each of the encoder's hidden states i.e. determine g(.) ?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden states!



Seq2Seq + Attention

Q: How do we determine how much attention to pay to each of the encoder's hidden states i.e. determine g(.) ?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden states!

For this, we define a context vector c_i, computed as a weighted sum of the encoder states h_i^E.

$$\boxed{c_i = \sum_{j=1}^{T_x} \alpha_{ij}(h_{i-1}^D, h_j^E) h_j^E}$$

NN: Is a FCNN with weights W_a

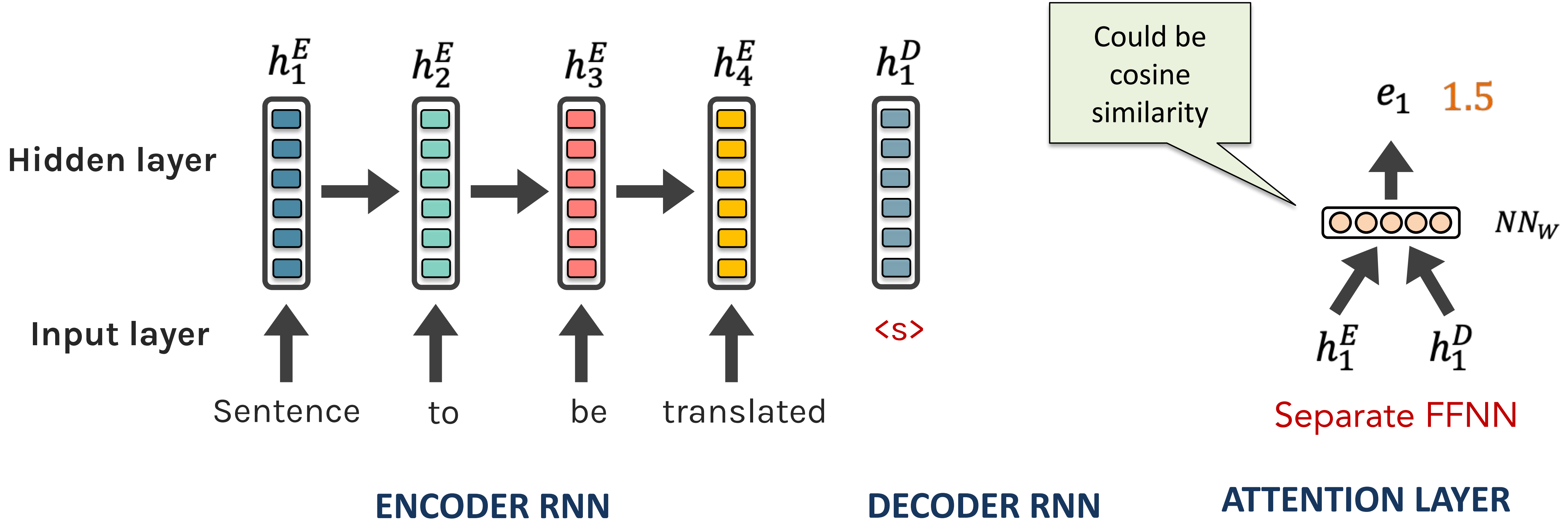
The weight α_{ij} for each state h_j^E is computed as $\alpha_{ij} = \dfrac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$

where $e_{ij} = NN_{W_a}(h_{i-1}^D, h_j^E)$

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden states?

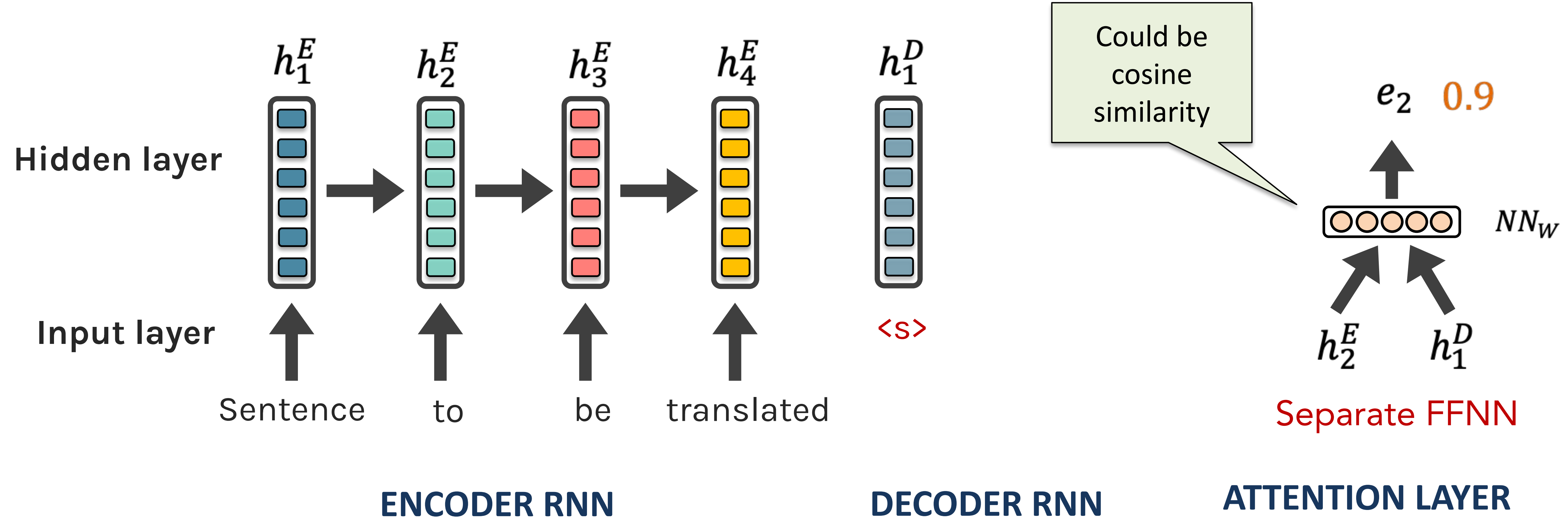
A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden states! We want to measure similarity between decoder hidden state and encoder hidden states in some ways.



Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden states?

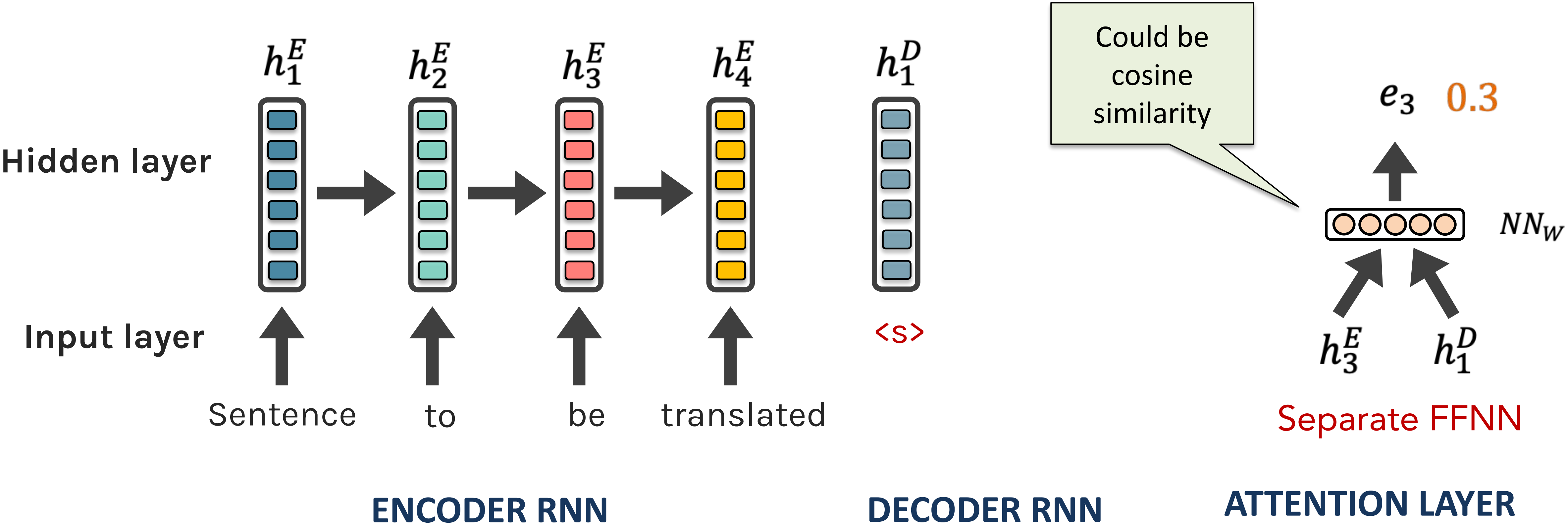
A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden states! We want to measure similarity between decoder hidden state and encoder hidden states in some ways.



Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden states?

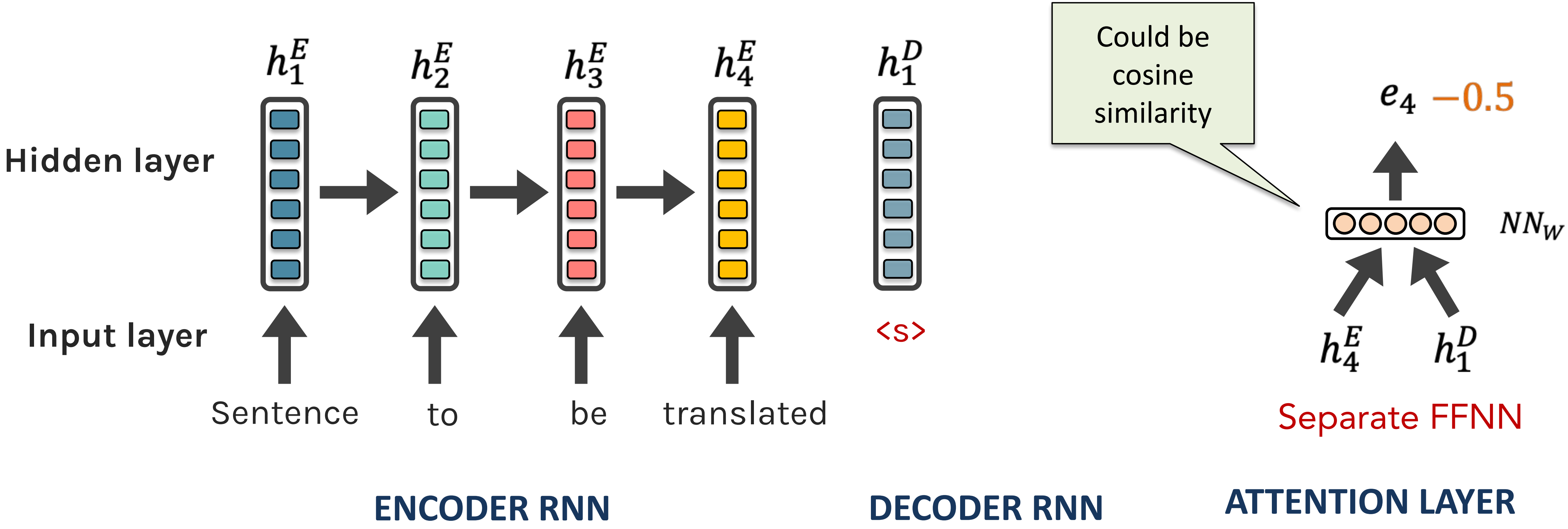
A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden states! We want to measure similarity between decoder hidden state and encoder hidden states in some ways.



Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden states?

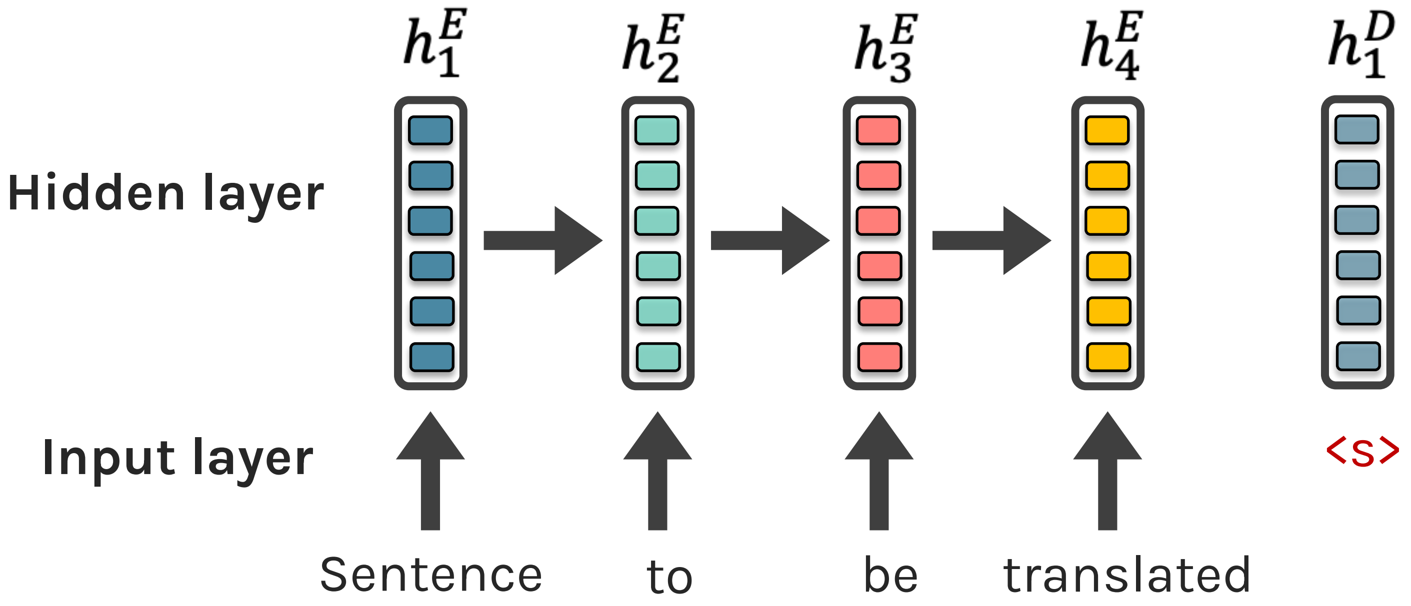
A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden states! We want to measure similarity between decoder hidden state and encoder hidden states in some ways.



Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden states?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden states! We want to measure similarity between decoder hidden state and encoder hidden states in some ways.



Attention (raw scores)

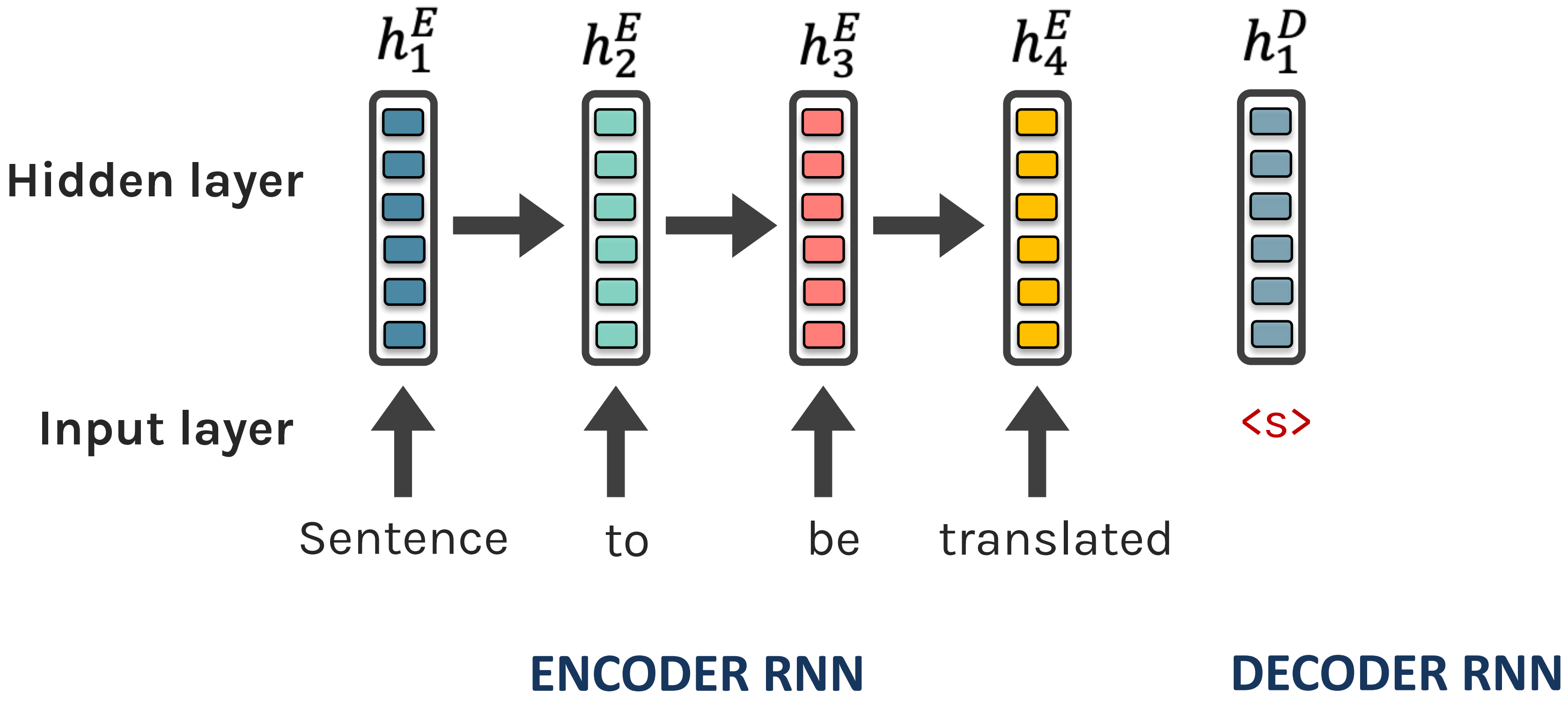
e_1 1.5
e_2 0.9
e_3 0.3
e_4 -0.5

ENCODER RNN **DECODER RNN** **ATTENTION LAYER**

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden states?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden states! We want to measure similarity between decoder hidden state and encoder hidden states in some ways.



Attention (raw scores)

e_1 1.5

e_2 0.9

e_3 0.3

e_4 −0.5

Attention (softmax'd)

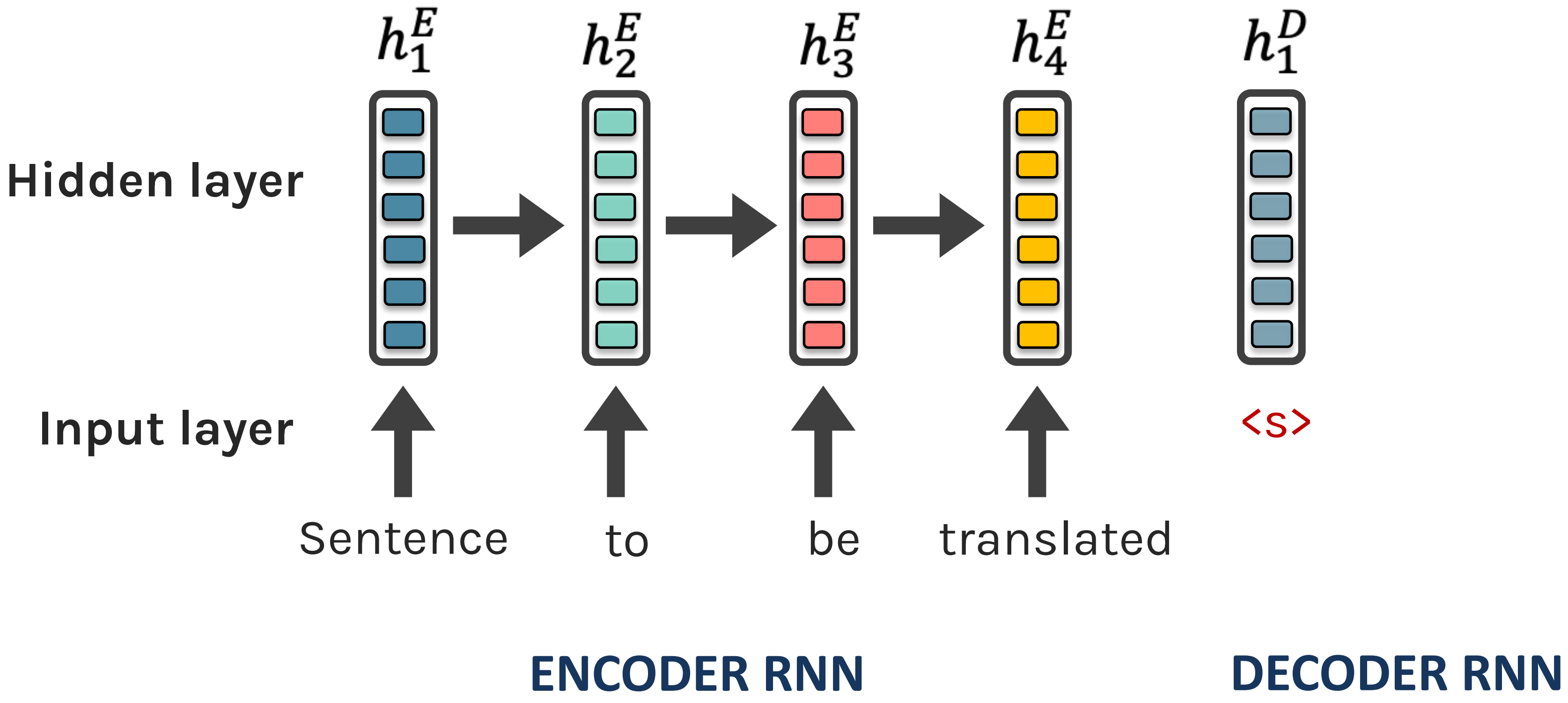
$$\alpha_i^1 = \frac{\exp(e_i)}{\sum_i^N \exp(e_1)}$$

ATTENTION LAYER

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder’s hidden states?

A: Let’s base it on our decoder’s previous hidden state (our latest representation of meaning) and all of the encoder’s hidden states! We want to measure similarity between decoder hidden state and encoder hidden states in some ways.



Attention (raw scores)

e_1 1.5

e_2 0.9

e_3 0.3

e_4 -0.5

Attention (softmax’d)

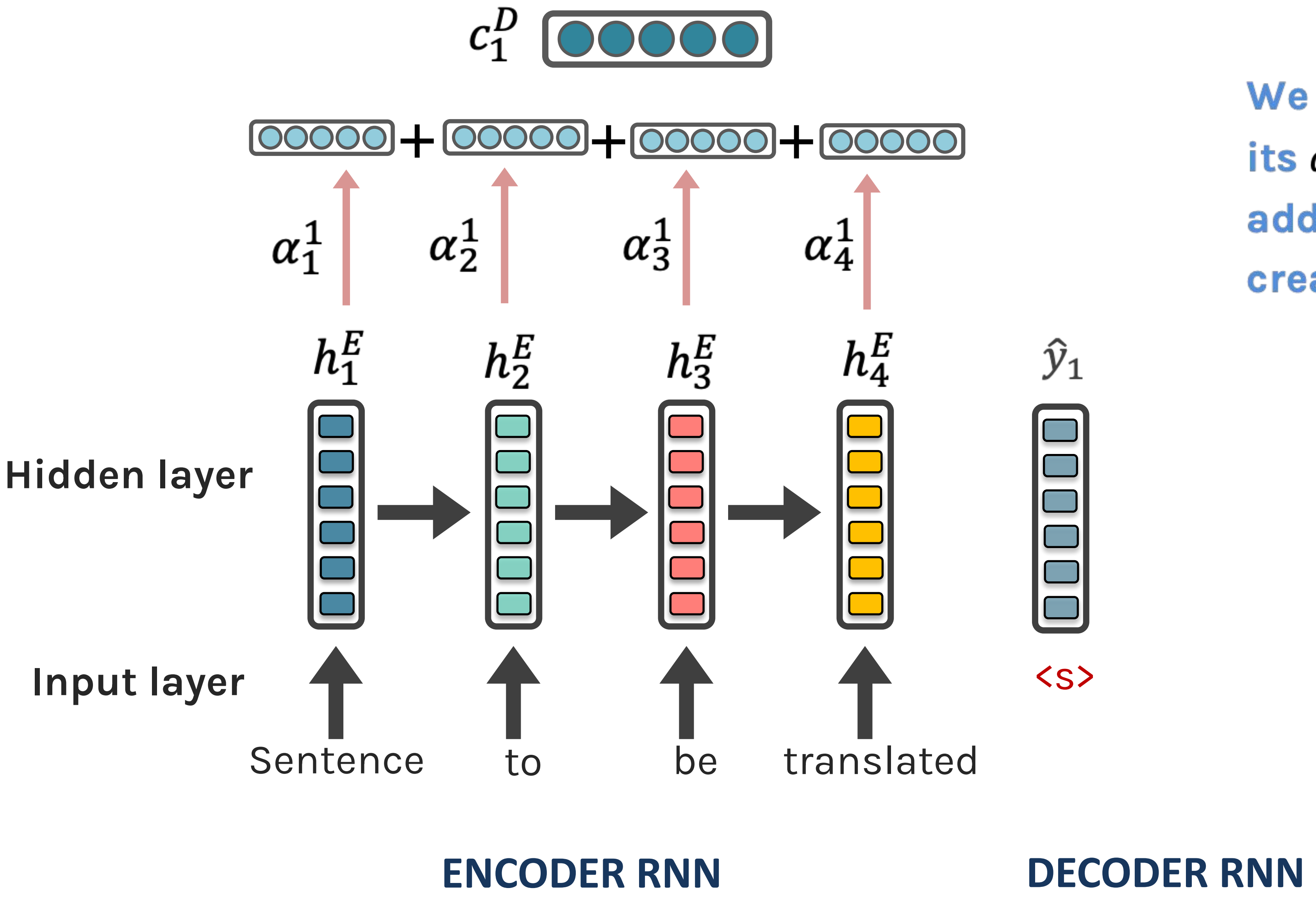
$\alpha_1^1 = 0.51$

$\alpha_2^1 = 0.28$

$\alpha_3^1 = 0.14$

$\alpha_3^1 = 0.07$

Seq2Seq + Attention



We multiply each hidden state by its α_i^1 attention weights and then add the resulting vectors to create a context vector c_1^D.

Attention (softmax'd)

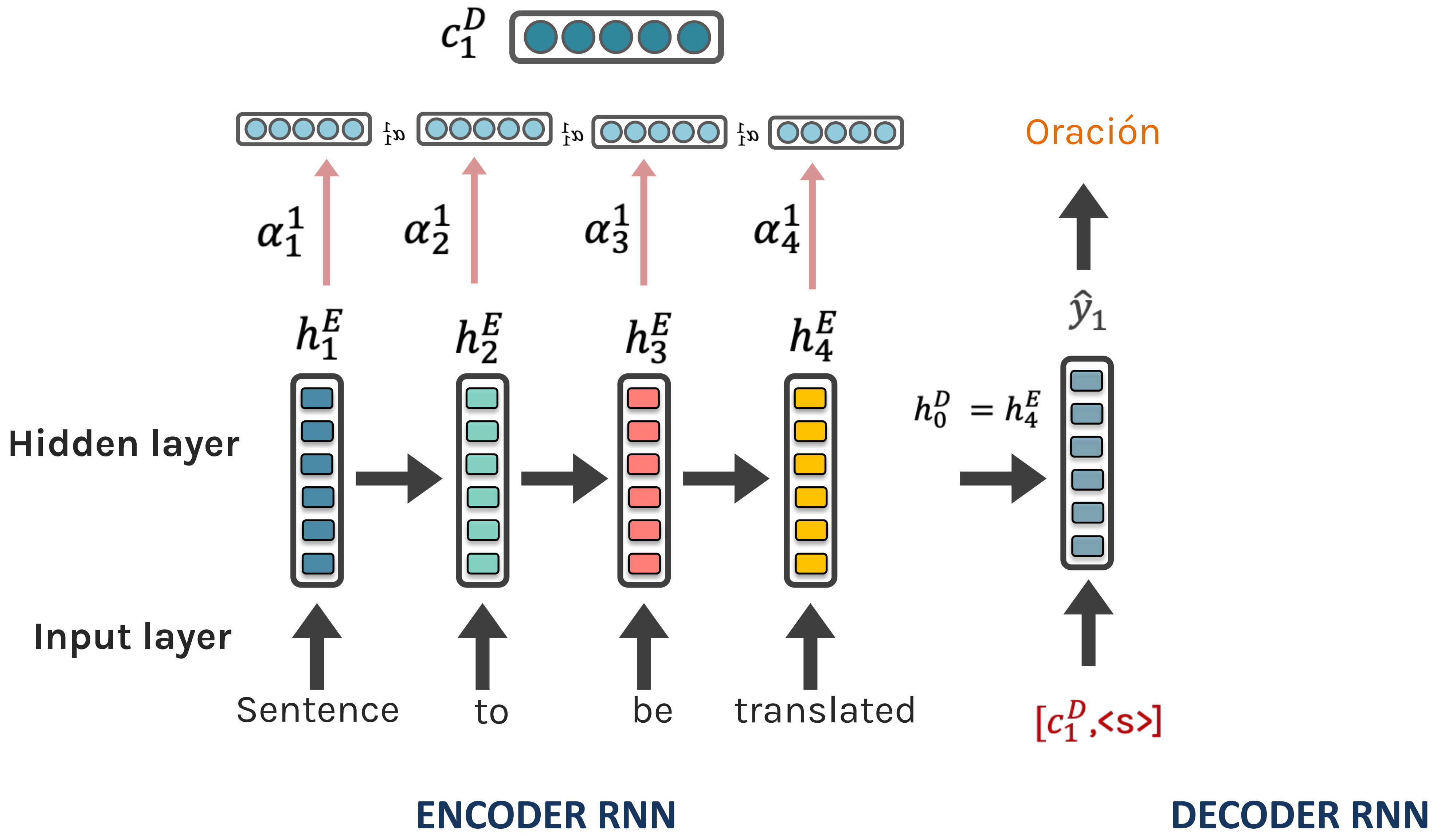
$\alpha_1^1 = 0.51$

$\alpha_2^1 = 0.28$

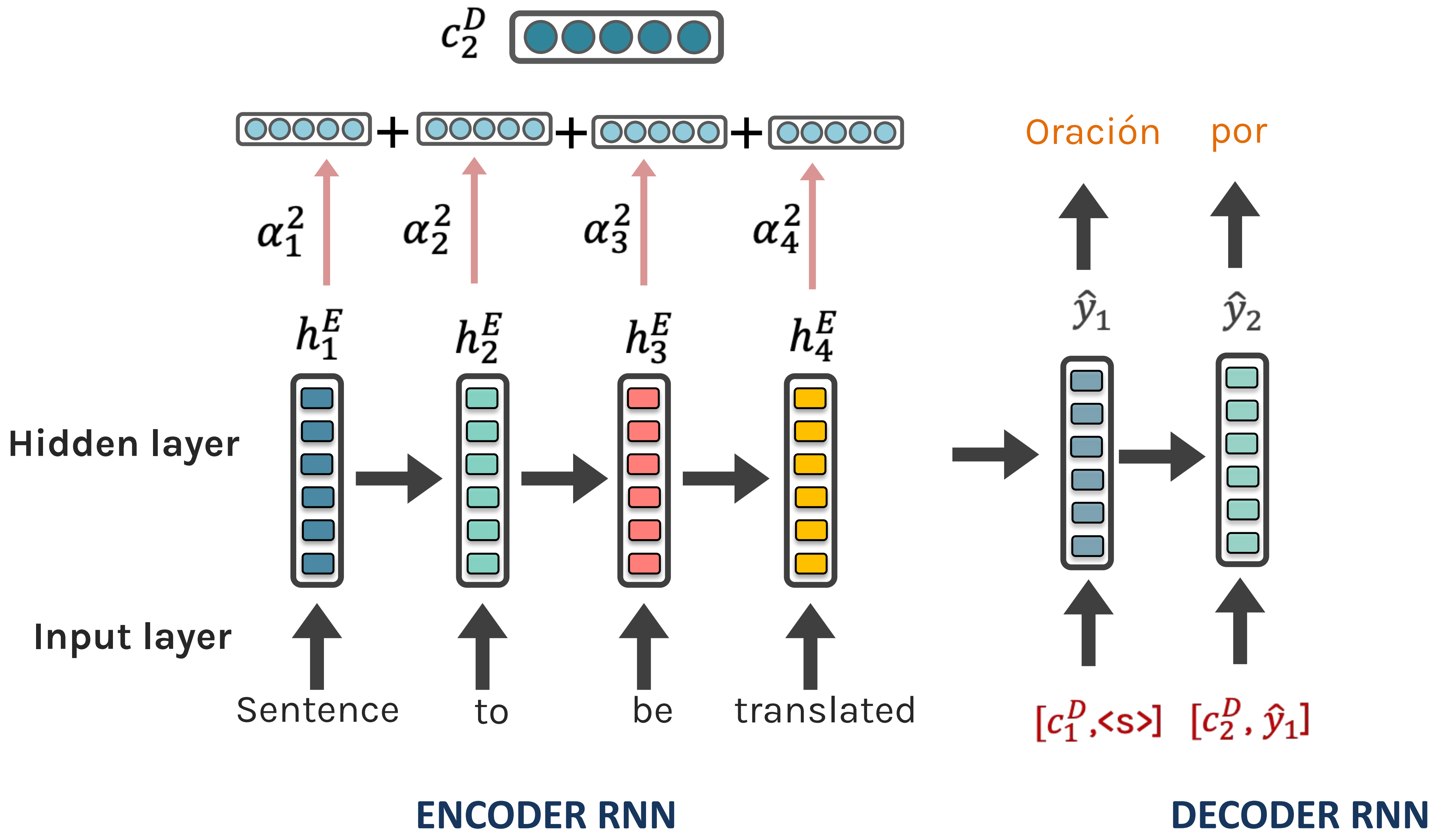
$\alpha_3^1 = 0.14$

$\alpha_3^1 = 0.07$

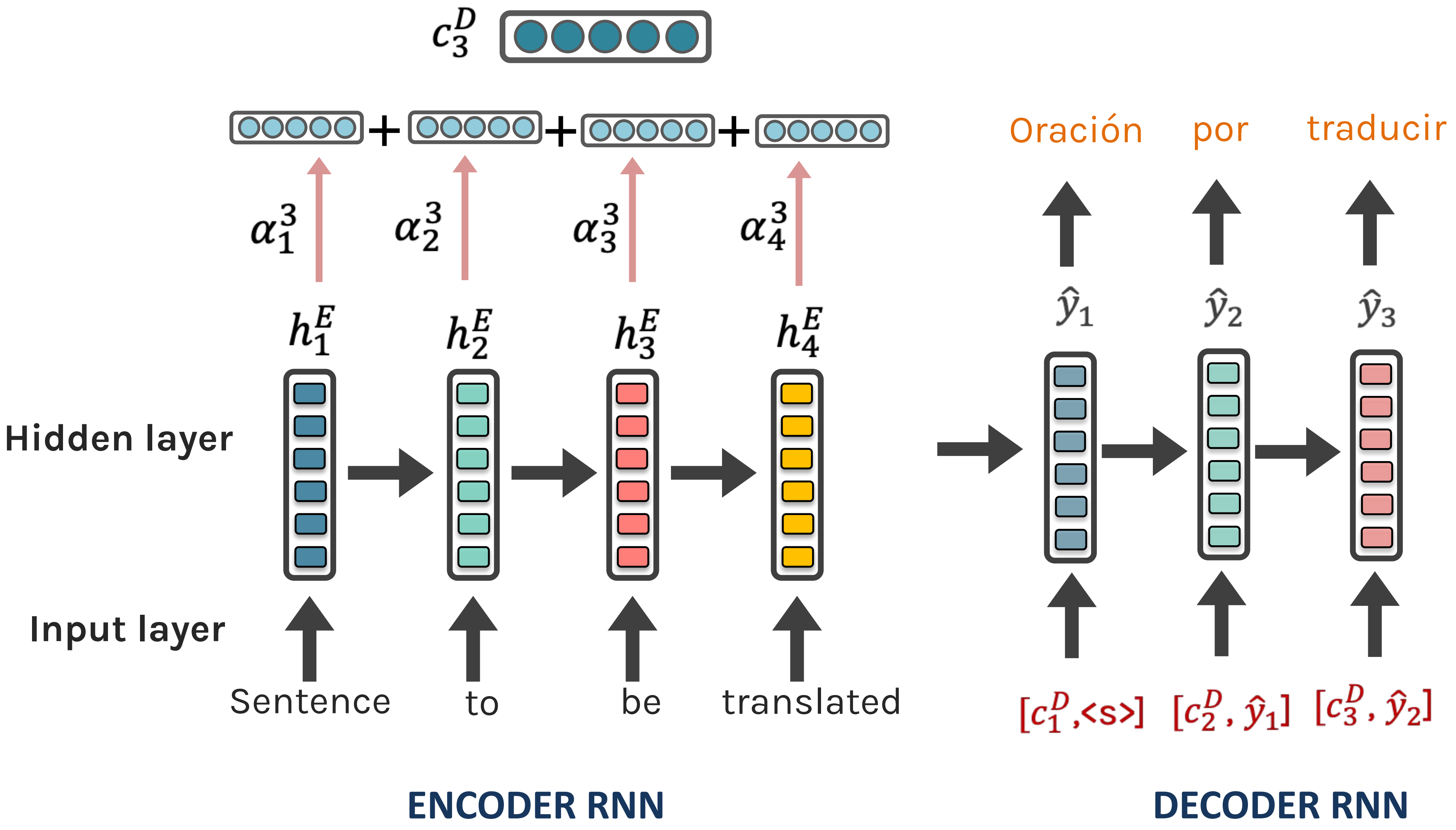
Seq2Seq + Attention



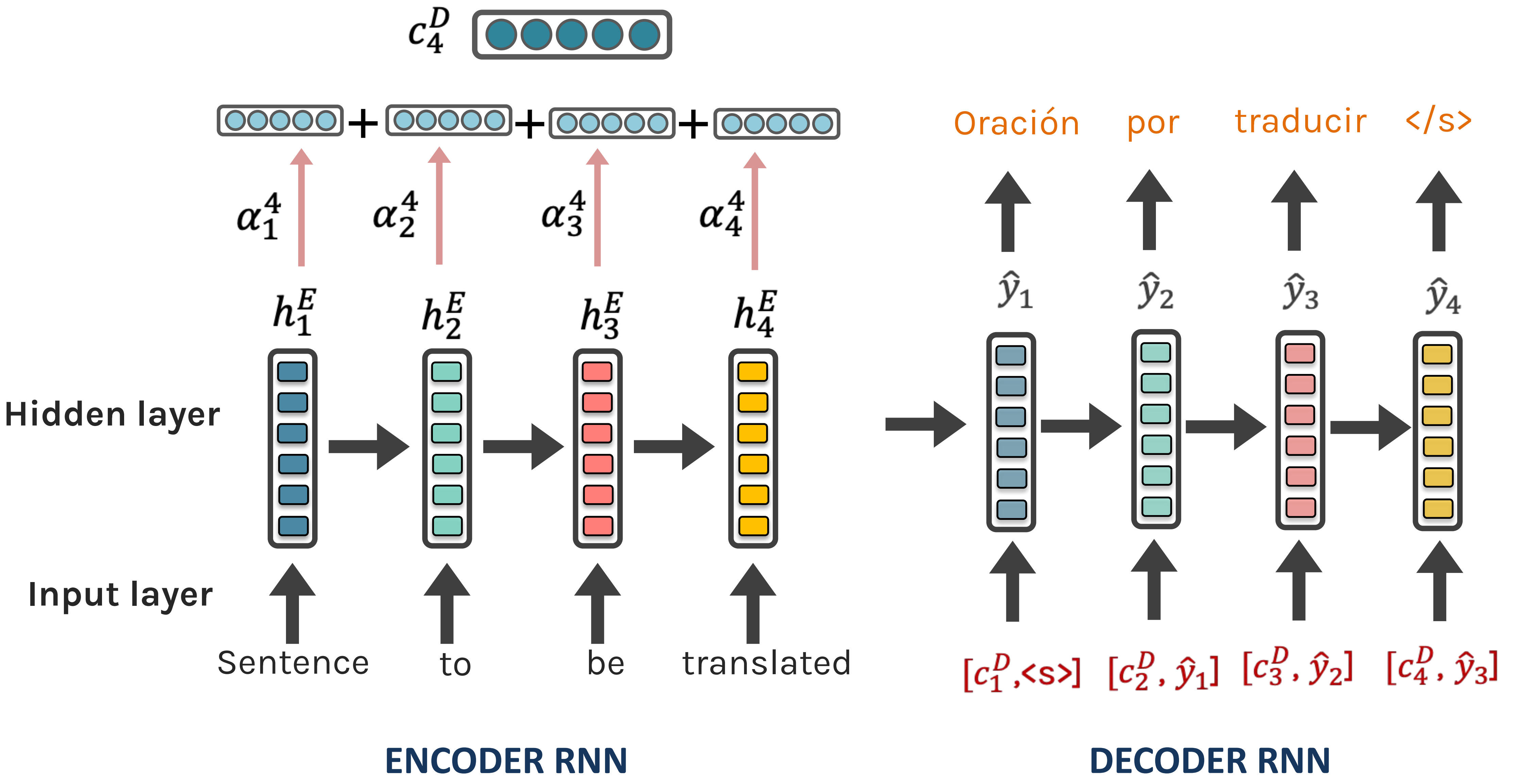
Seq2Seq + Attention



Seq2Seq + Attention



Seq2Seq + Attention



Seq2Seq + Attention

Attention:

- greatly improves seq2seq results
- allows us to visualize the contribution each word gave during each step of the decoder

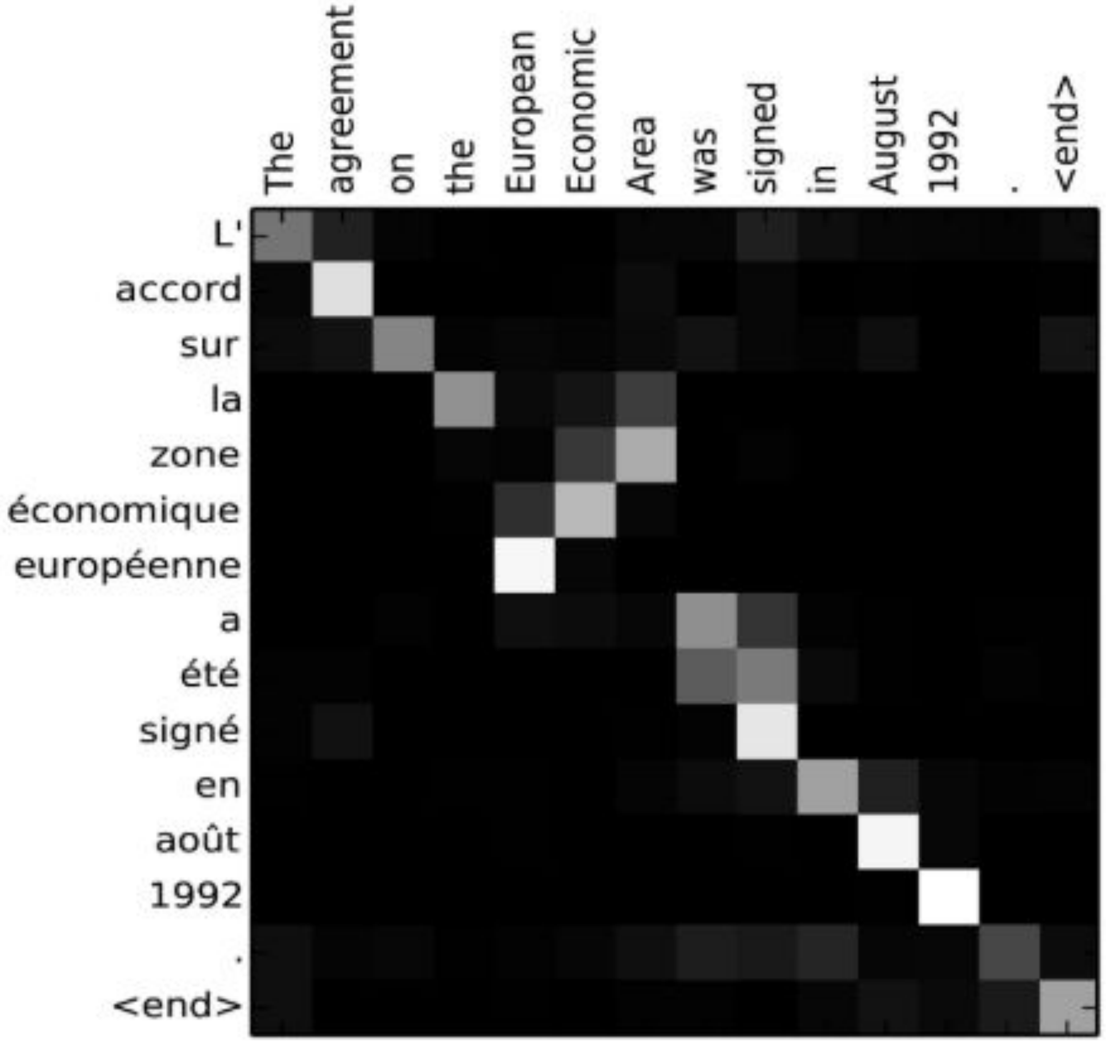


Image source: Fig 3 in Bahdanau et al., 2015

Next Class

1. What are Language Models
2. Neural Networks for Language Modeling
3. Recurrent Neural Network
4. Seq2Seq + Attention
5. **Self Attention**
6. **Transformers**
7. **Tutorial: SOTA Language Models**

THANK YOU