Recurrent Neural Networks

Lecture # 5

Recap: Language Model

Language model defines "how probable a sentence is"

John is driving a car vs. John is driving a cat

In other words, what is the probability to predict **cat** or **car** given the context "John is driving a"

Shortcomings of NNLM

Q: What are some shortcomings of the feed forward neural network language model that we have seenso far?

Shortcomings of NNLM

Q: What are some shortcomings of the feed forward neural network language model that we have seenso far?

- Independence assumption: We have a “hard” limit on the amount of context we see - bigram, trigram or some ngram.

Shortcomings of NNLM

Q: What are some shortcomings of the feed forward neural network language model that we have seenso far?

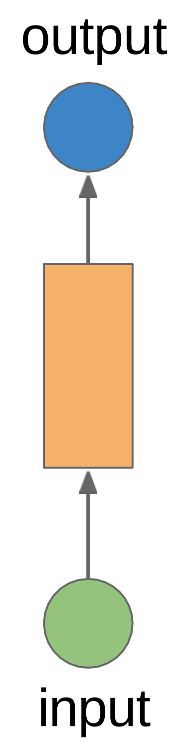
- Independence assumption: We have a “hard” limit on the amount of context we see - bigram, trigram or some ngram.
- Limit can *never* large enough

Shortcomings of NNLM

Q: What are some shortcomings of the feed forward neural network language model that we have seenso far?

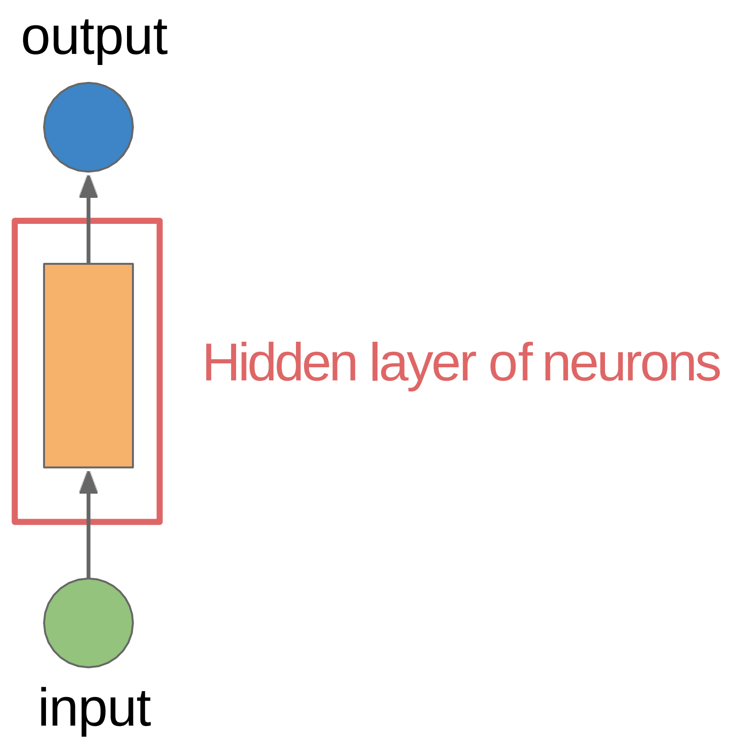
- Independence assumption: We have a “hard” limit on the amount of context we see - bigram, trigram or some ngram.
- Limit can *never* large enough
- It is not uncommon to have longer range dependencies in language!

Recurrent Neural Network



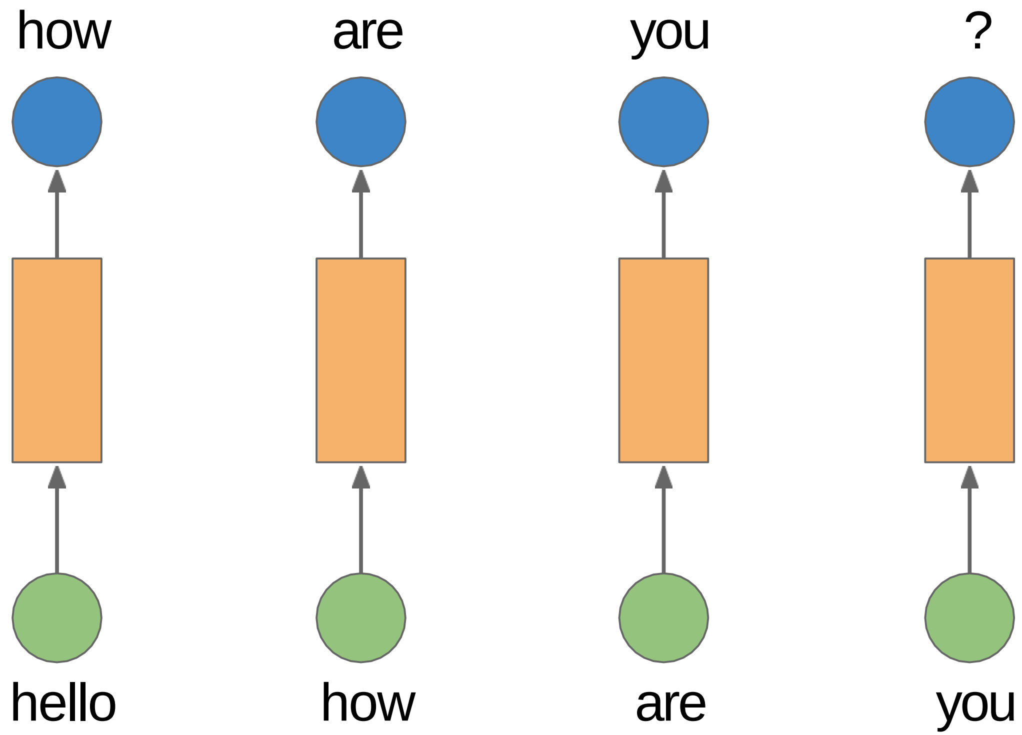
1-layer Feedforward Neural network

Recurrent Neural Network

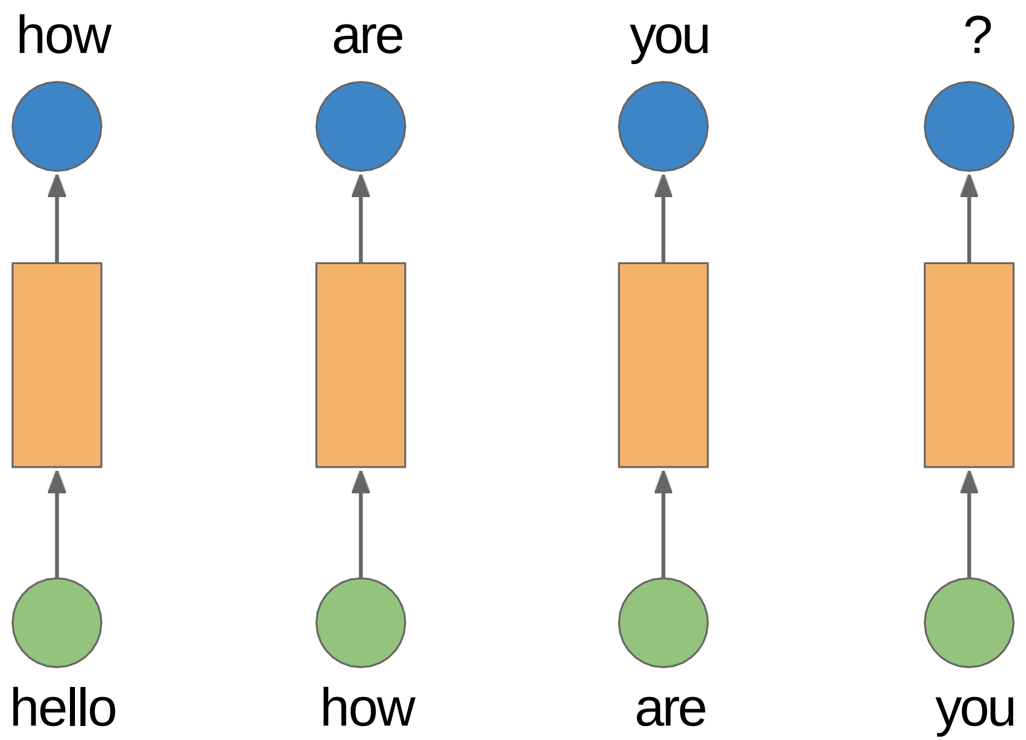


1-layer Feedforward Neural network

Recurrent Neural Network

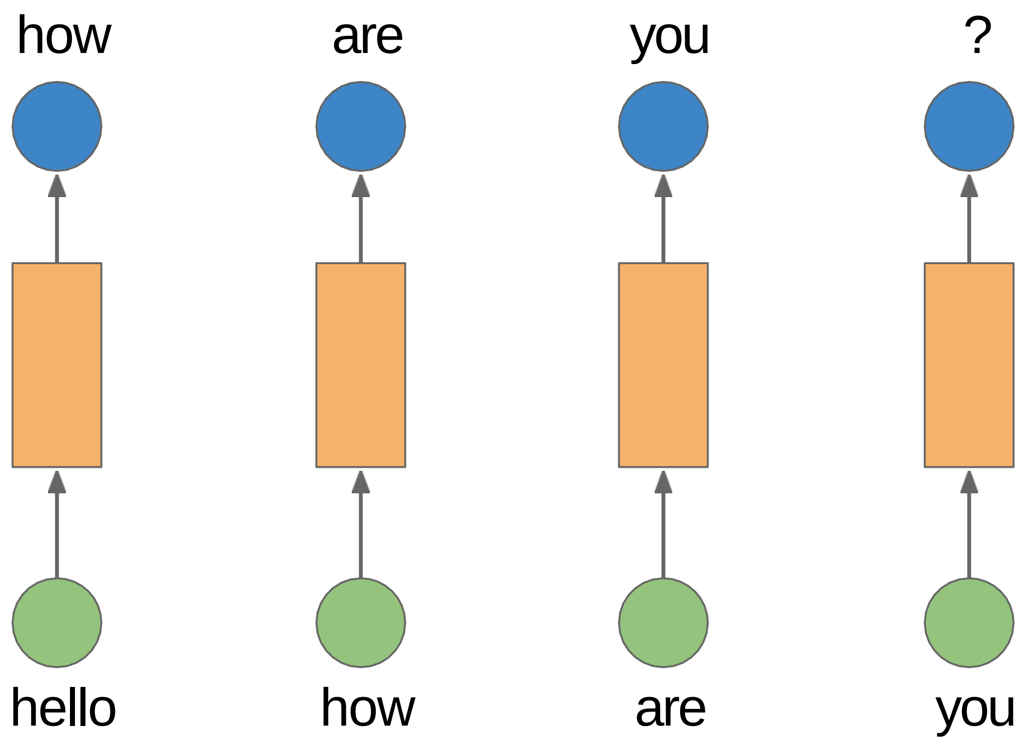


Recurrent Neural Network



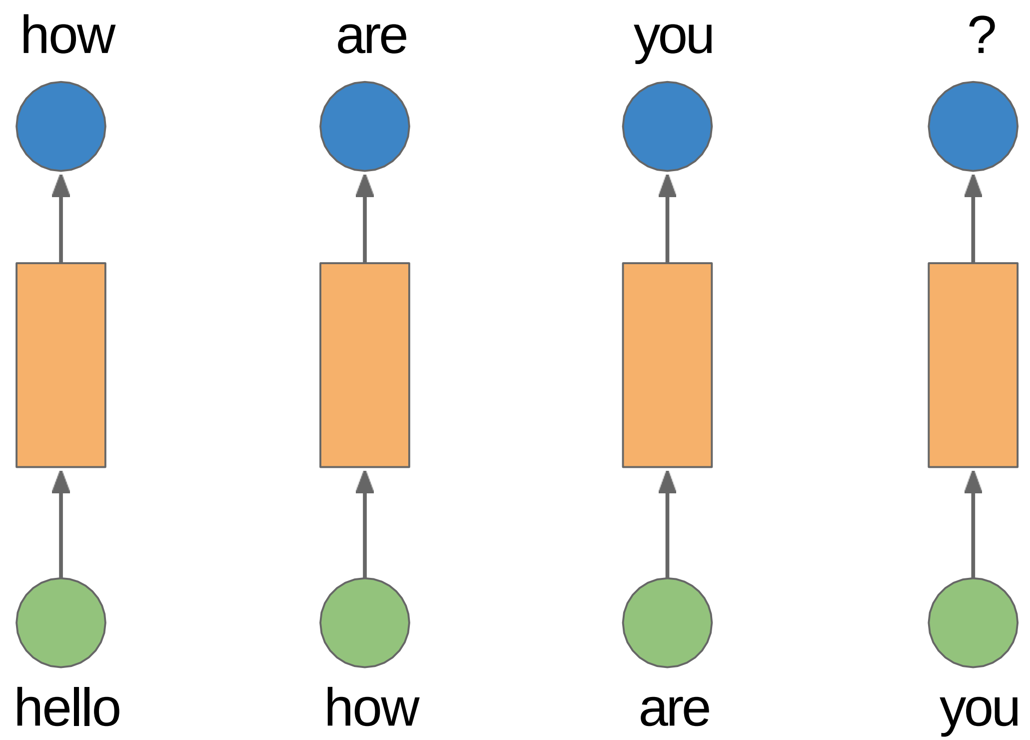
In the real world, we remember some history of previous words

Recurrent Neural Network



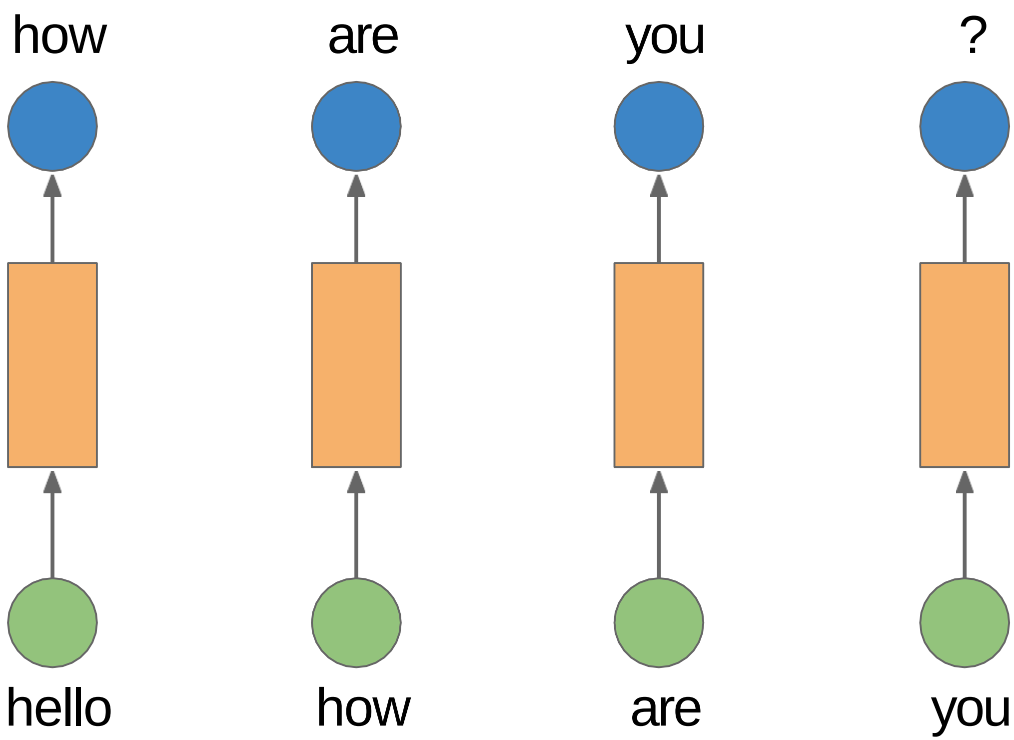
In our network here, each step is independent of the previous steps

Recurrent Neural Network



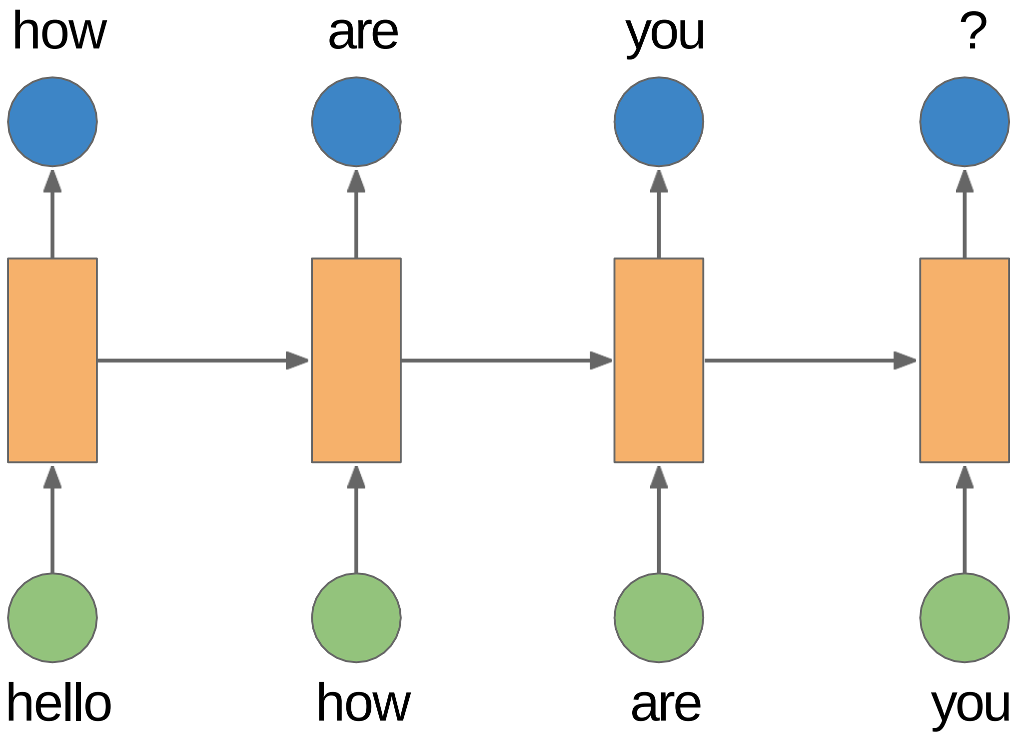
The only context available at every step is the input we provide to the network (bigram, trigram etc)

Recurrent Neural Network



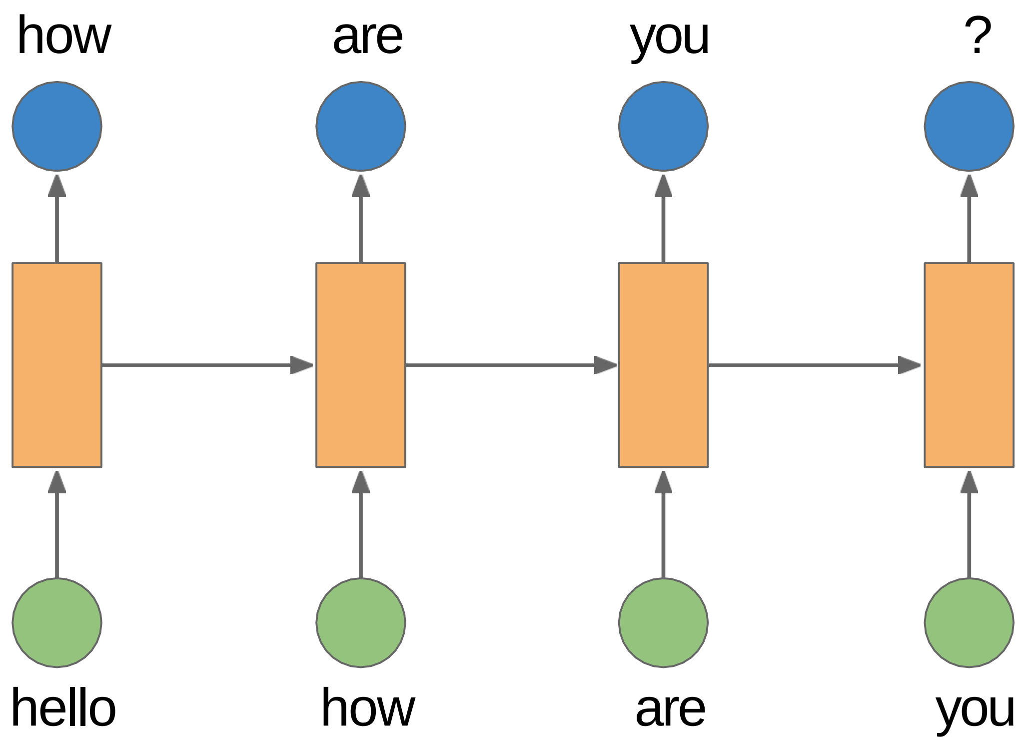
Why not connect these networks?

Recurrent Neural Network



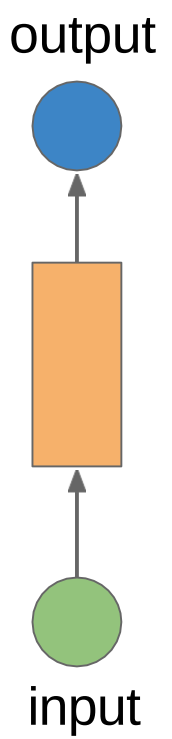
Why not connect these networks?

Recurrent Neural Network

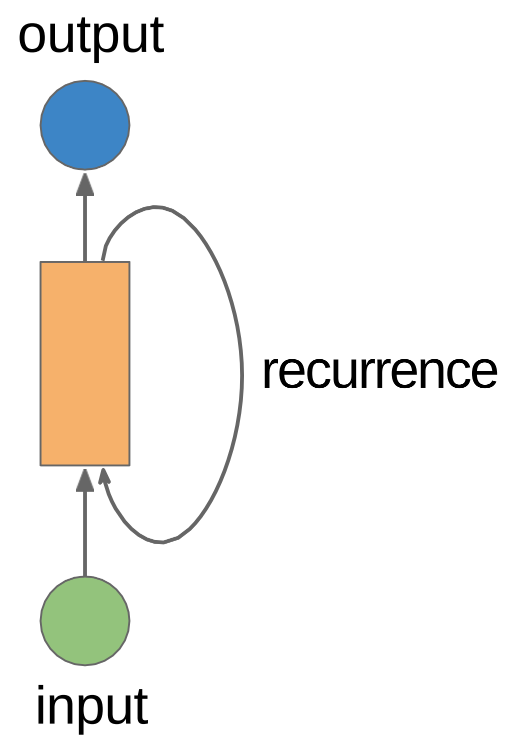


This is what recurrent neural networks do

Recurrent Neural Network

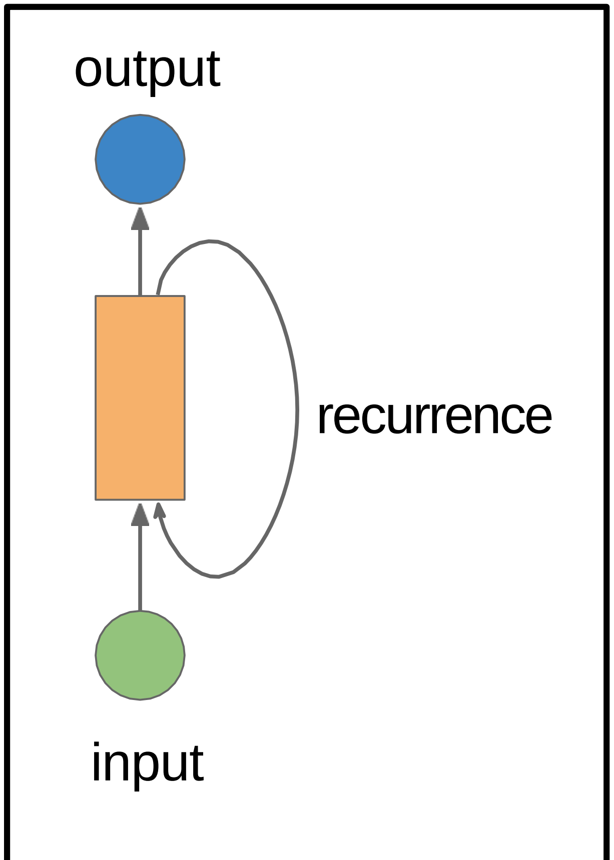


1-layer Feedforward
Neural network



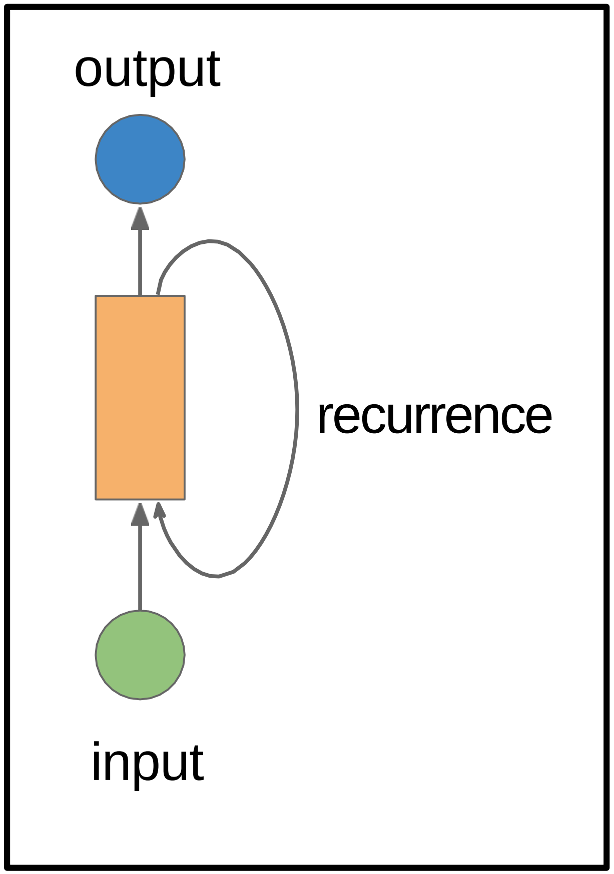
1-layer Recurrent
Neural network

Recurrent Neural Network

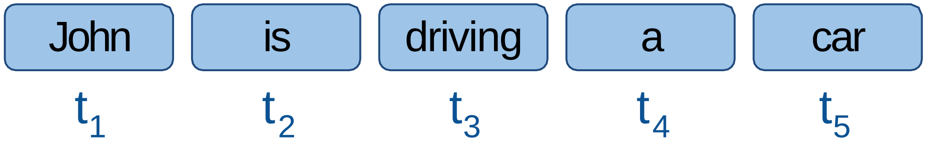


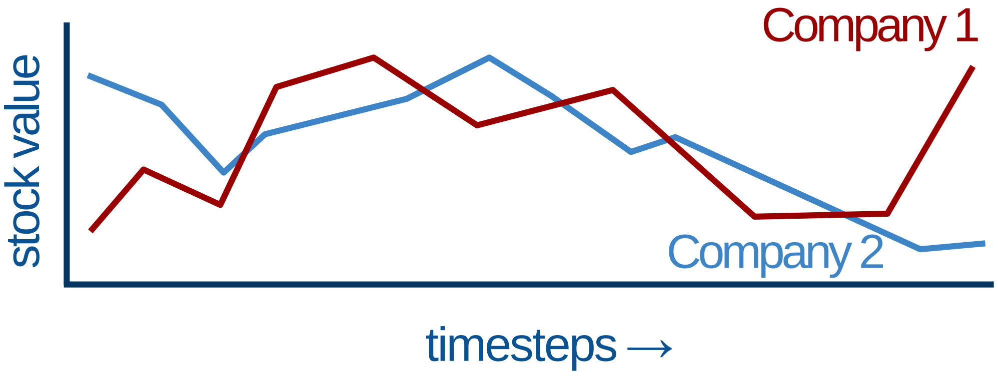
Recurrent units work very well for sequential information like a series of words, or knowledge across *timesteps*

Recurrent Neural Network

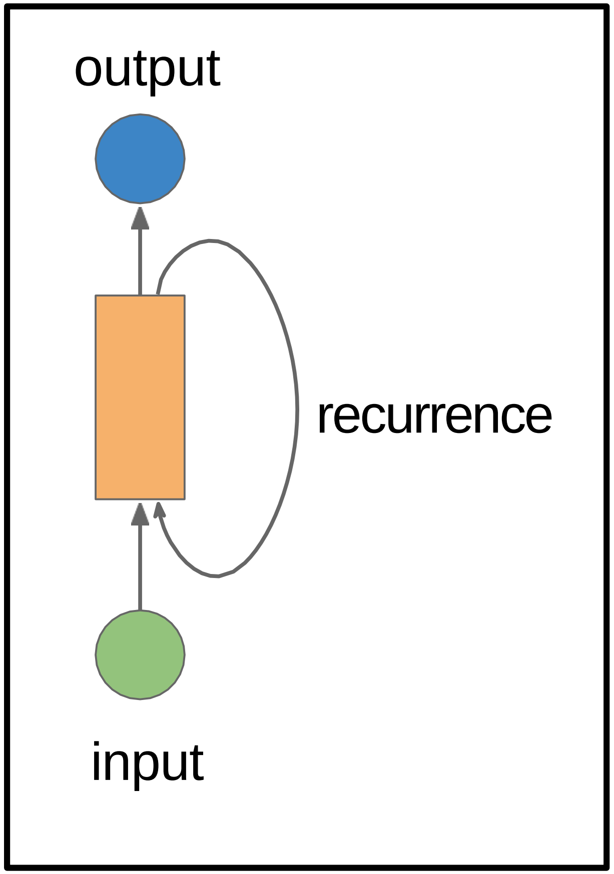


Recurrent units work very well for sequential information like a series of words, or knowledge across *timesteps*





Recurrent Neural Network



Recurrent units work very well for sequential information like a series of words, or knowledge across *timesteps*

The recurrence unit has two inputs:

1) x_i (input at time i)
2) h_{i-1} (input from previous state)

Recurrent Neural Network

Mathematically,

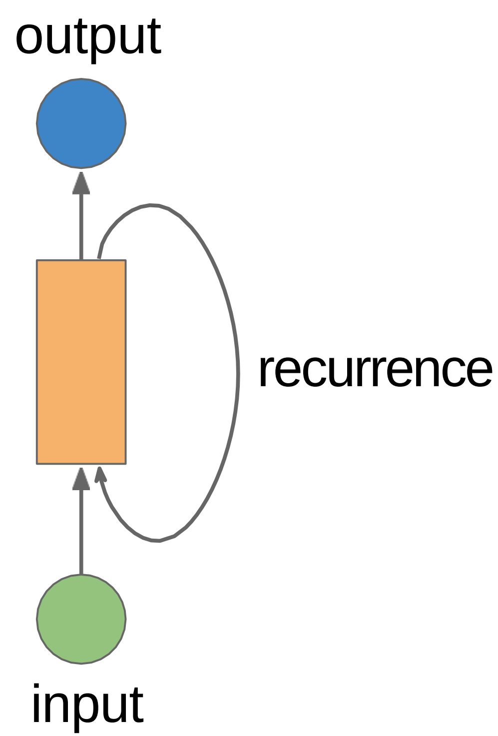
$$h = Wx + b \longrightarrow h_t = Wx + W_h h_{t-1} + b$$

Linear Recurrent

We have one additional set of parameters: W_h
which deals with the information transferred from the previous step

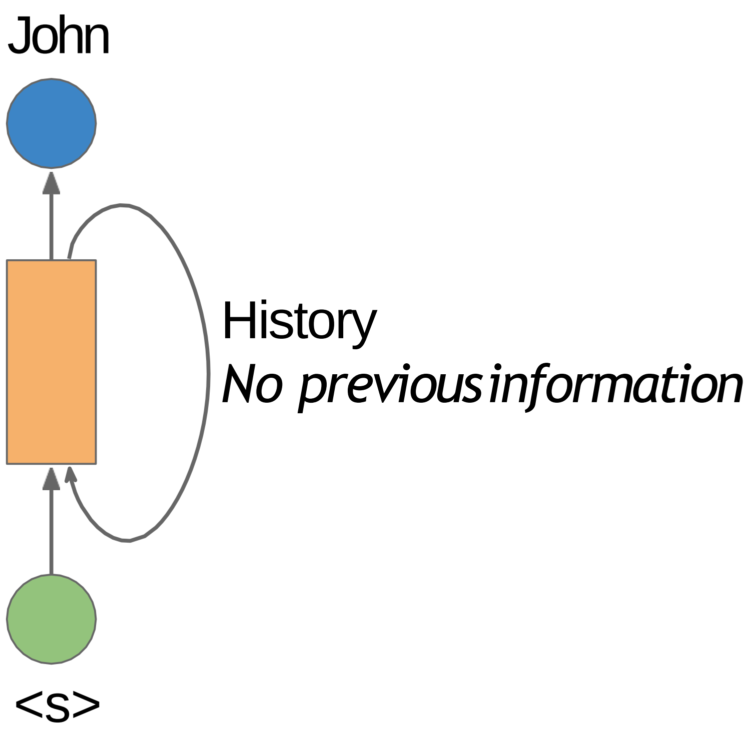
Recurrent Neural Network

Consider an example: <s> John is driving a car </s>



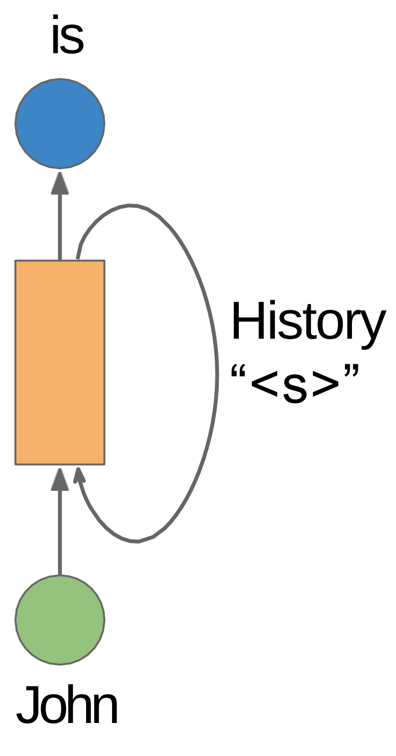
Recurrent Neural Network

Consider an example: <s> John is driving a car </s>



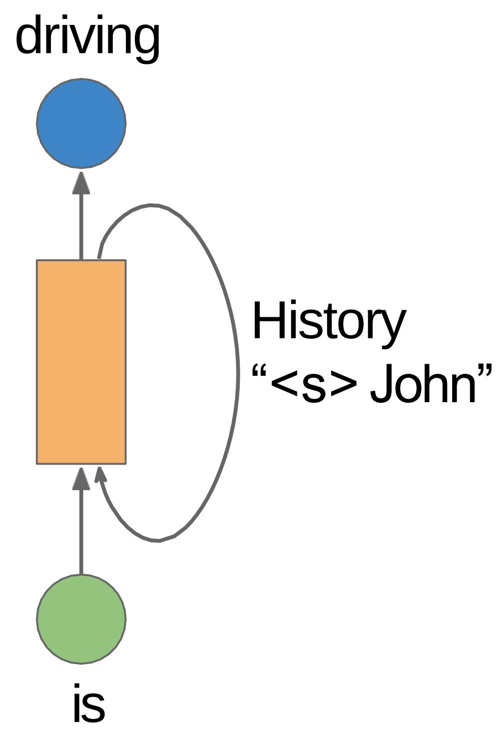
Recurrent Neural Network

Consider an example: <s> John is driving a car </s>



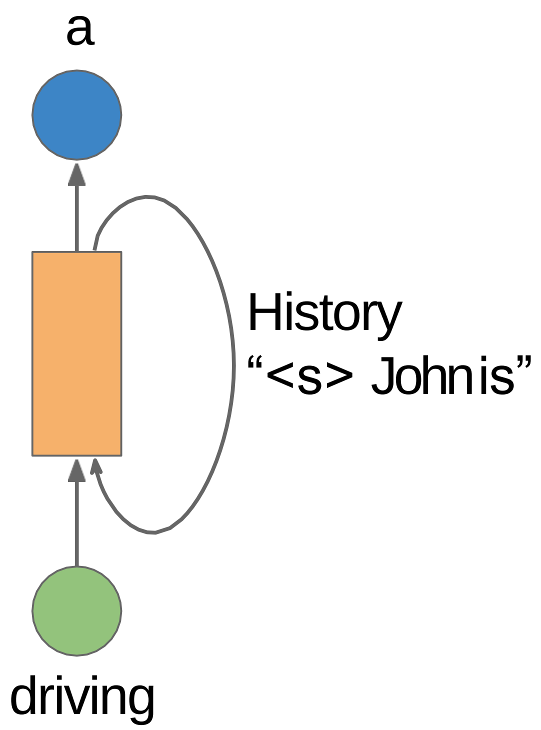
Recurrent Neural Network

Consider an example: <s> John is driving a car </s>



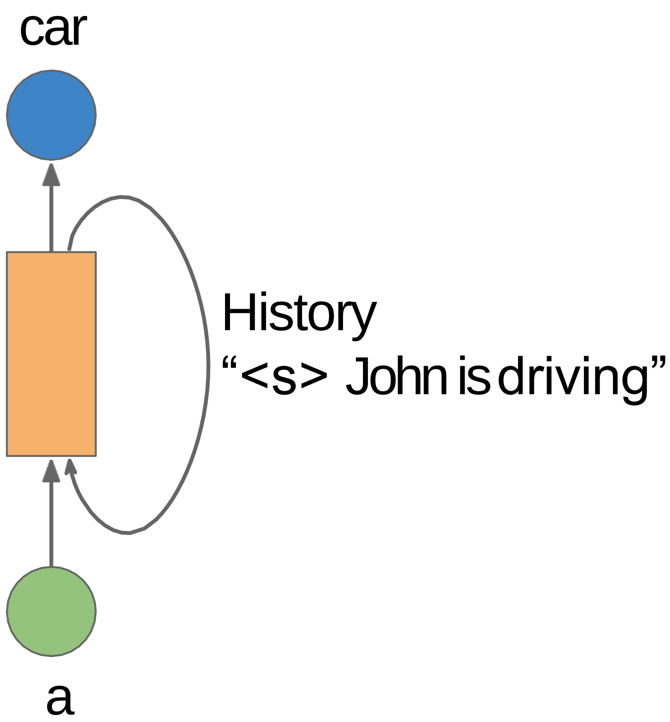
Recurrent Neural Network

Consider an example: <s> John is driving a car </s>



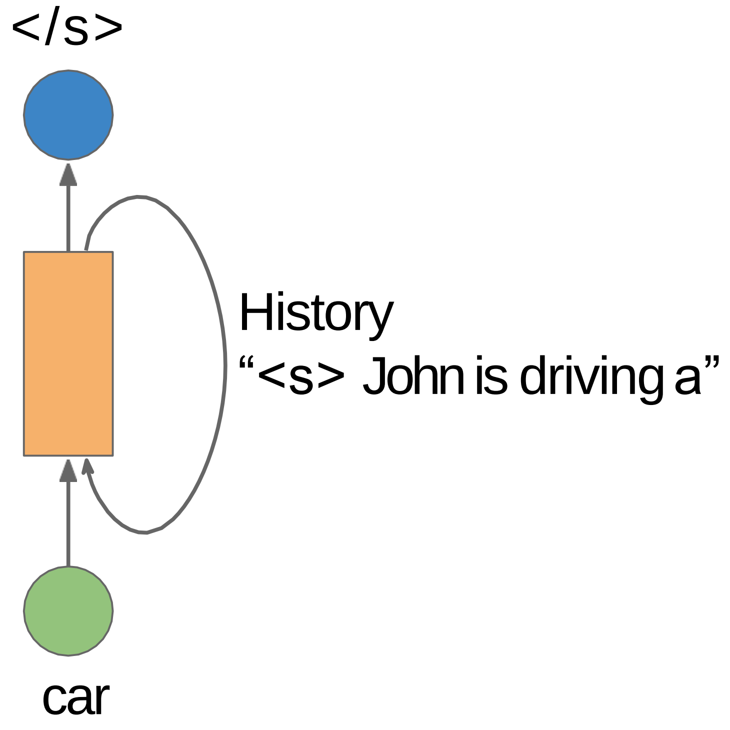
Recurrent Neural Network

Consider an example: <s> John is driving a car </s>



Recurrent Neural Network

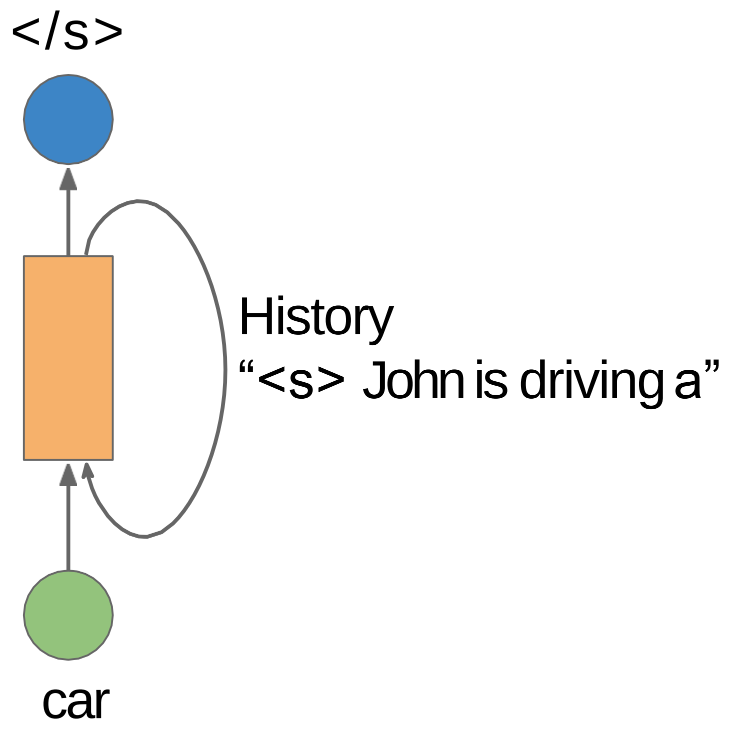
Consider an example: <s> John is driving a car </s>



Recurrent Neural Network

Consider an example: <s> John is driving a car </s>

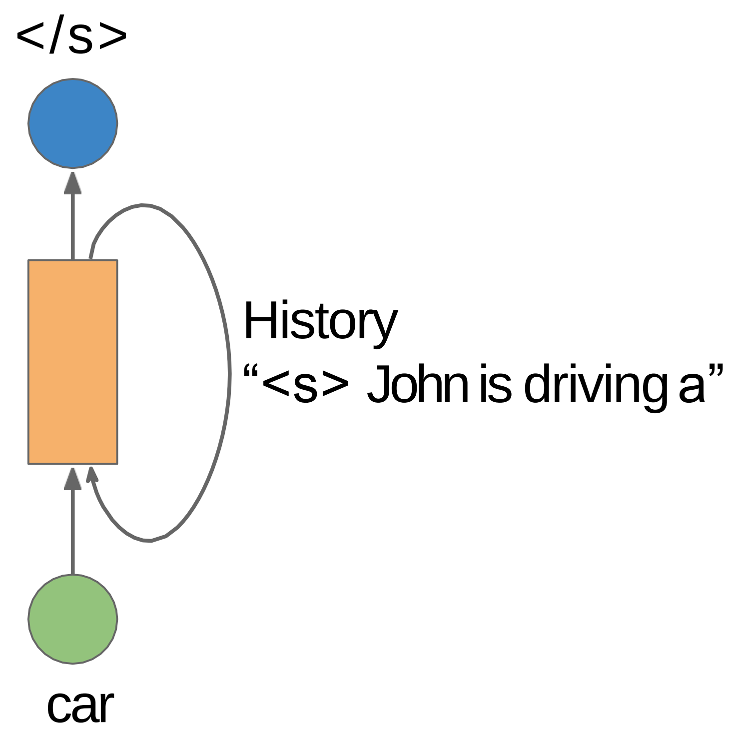
At the last timestep, the hidden state will have information about the entire sentence: **“John is driving a”** from history and **“car”** from the input



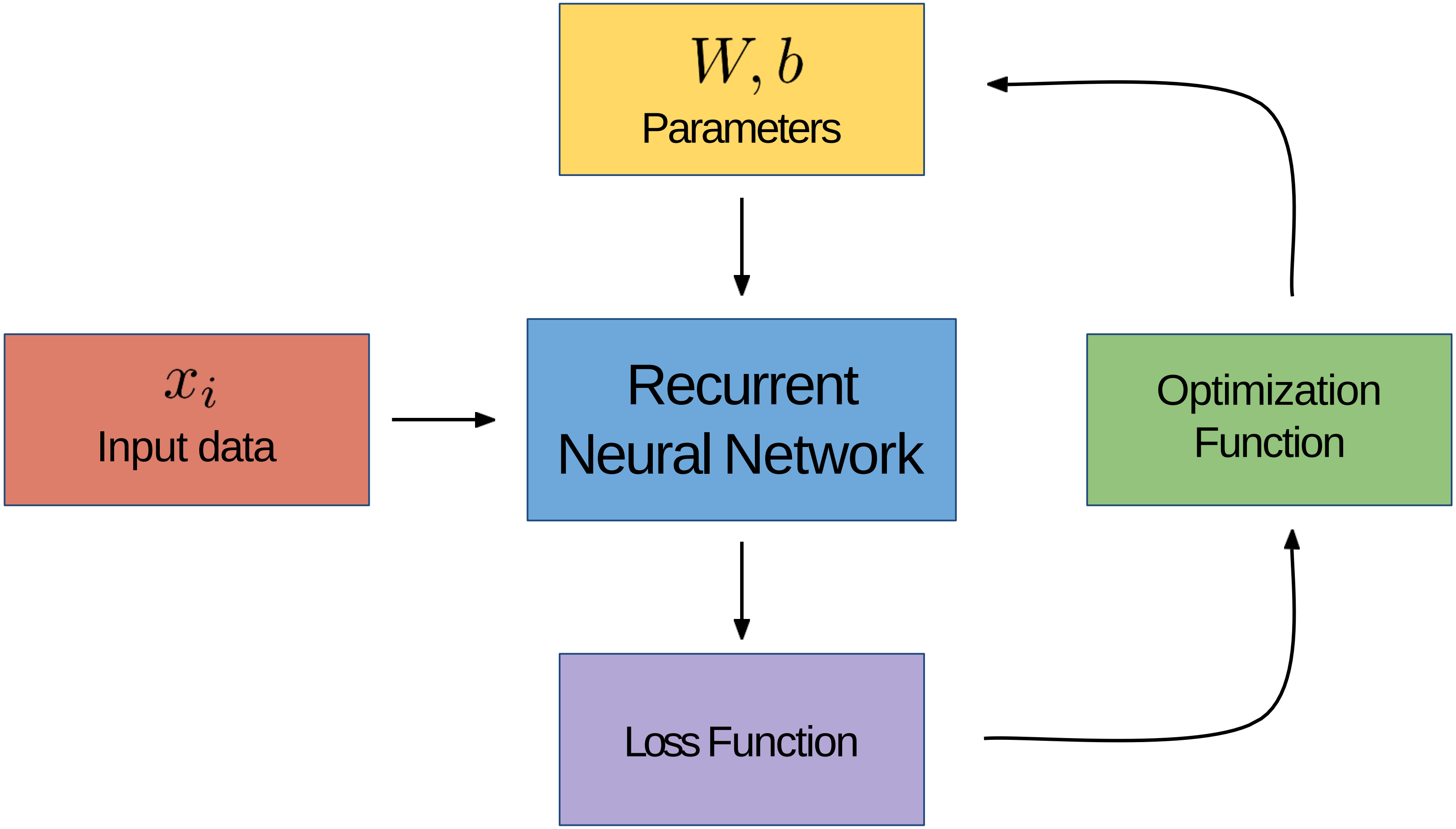
Recurrent Neural Network

Consider an example: <s> John is driving a car </s>

This hidden state can be considered as a “summary” of the entire sentence represented as a vector



Recap

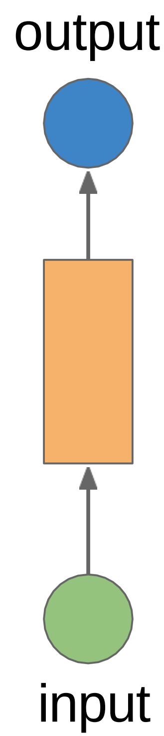


W, b
Parameters
x_i
Input data
Recurrent Neural Network
Optimization Function
Loss Function

Loss computation

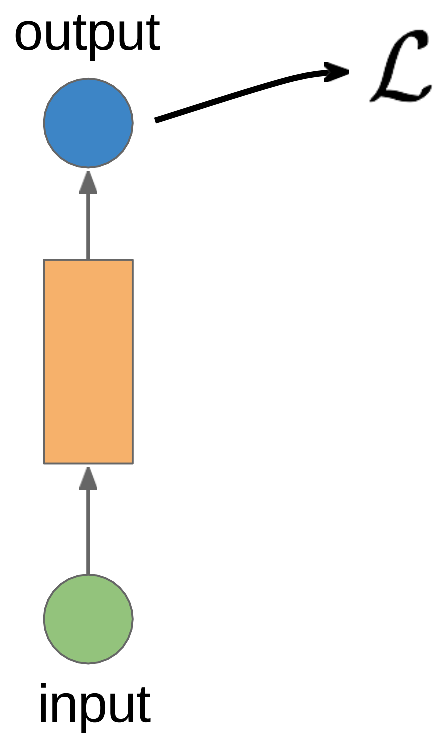
in recurrent neural networks

Loss Computation



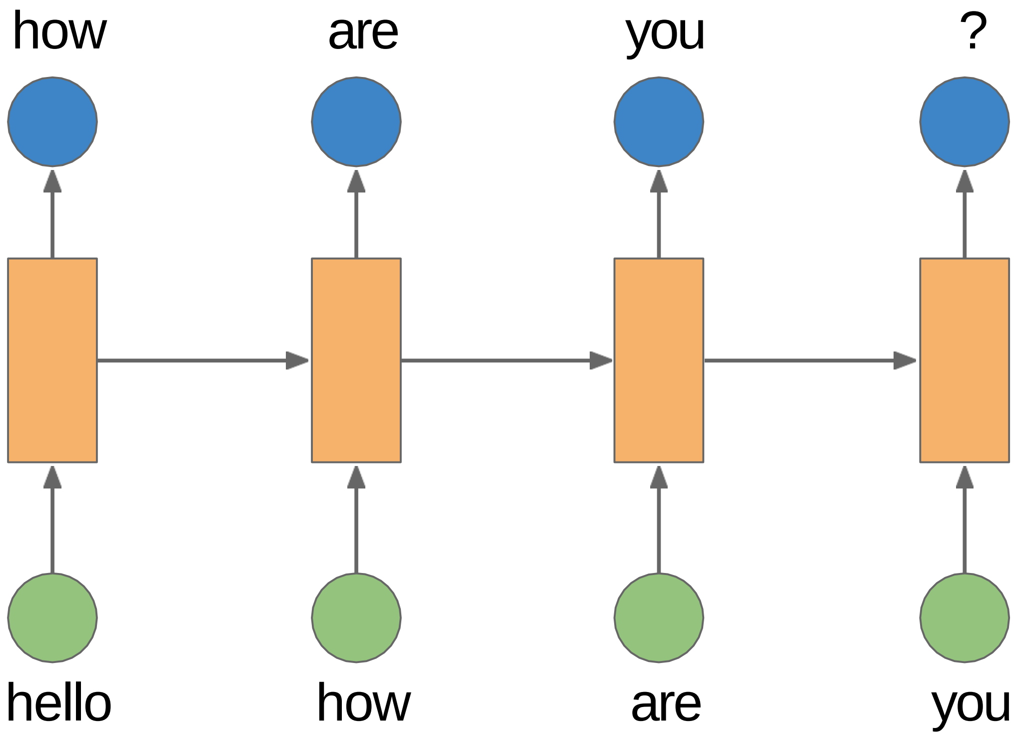
Recall that in a feed forward network, we have a single output

Loss Computation



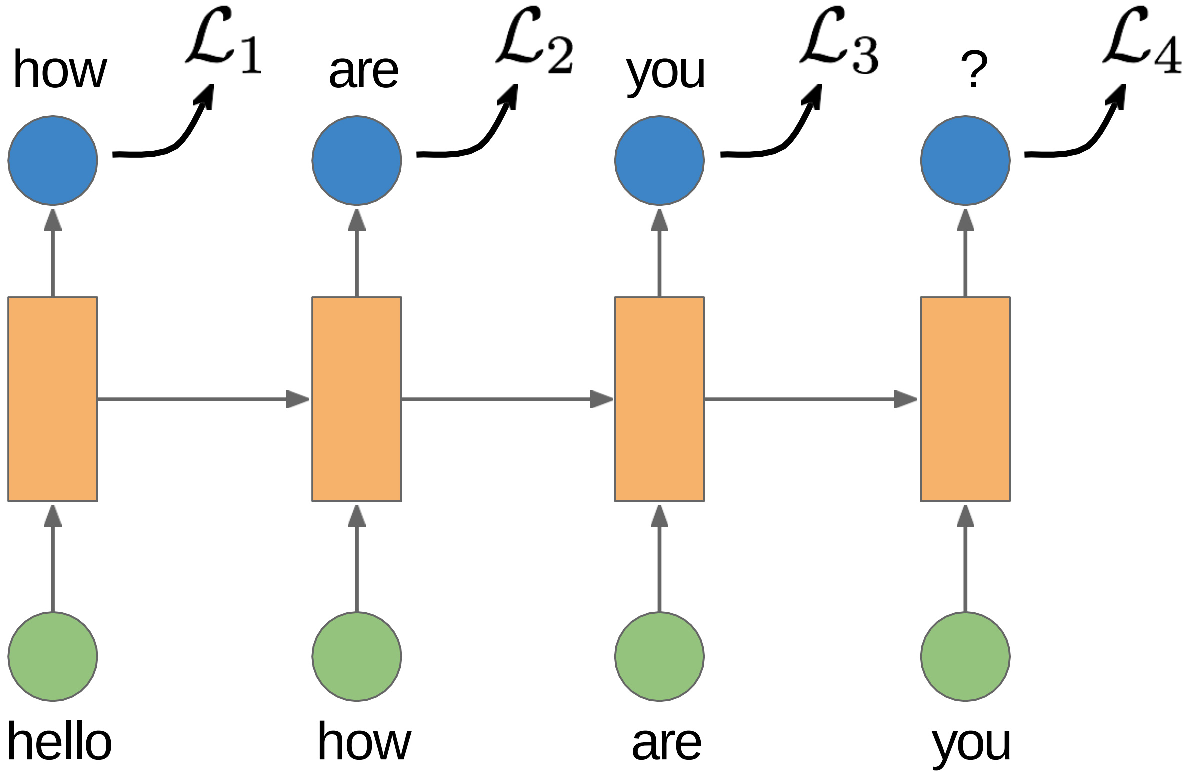
We compare this single output with the true label to get a loss value

Loss Computation



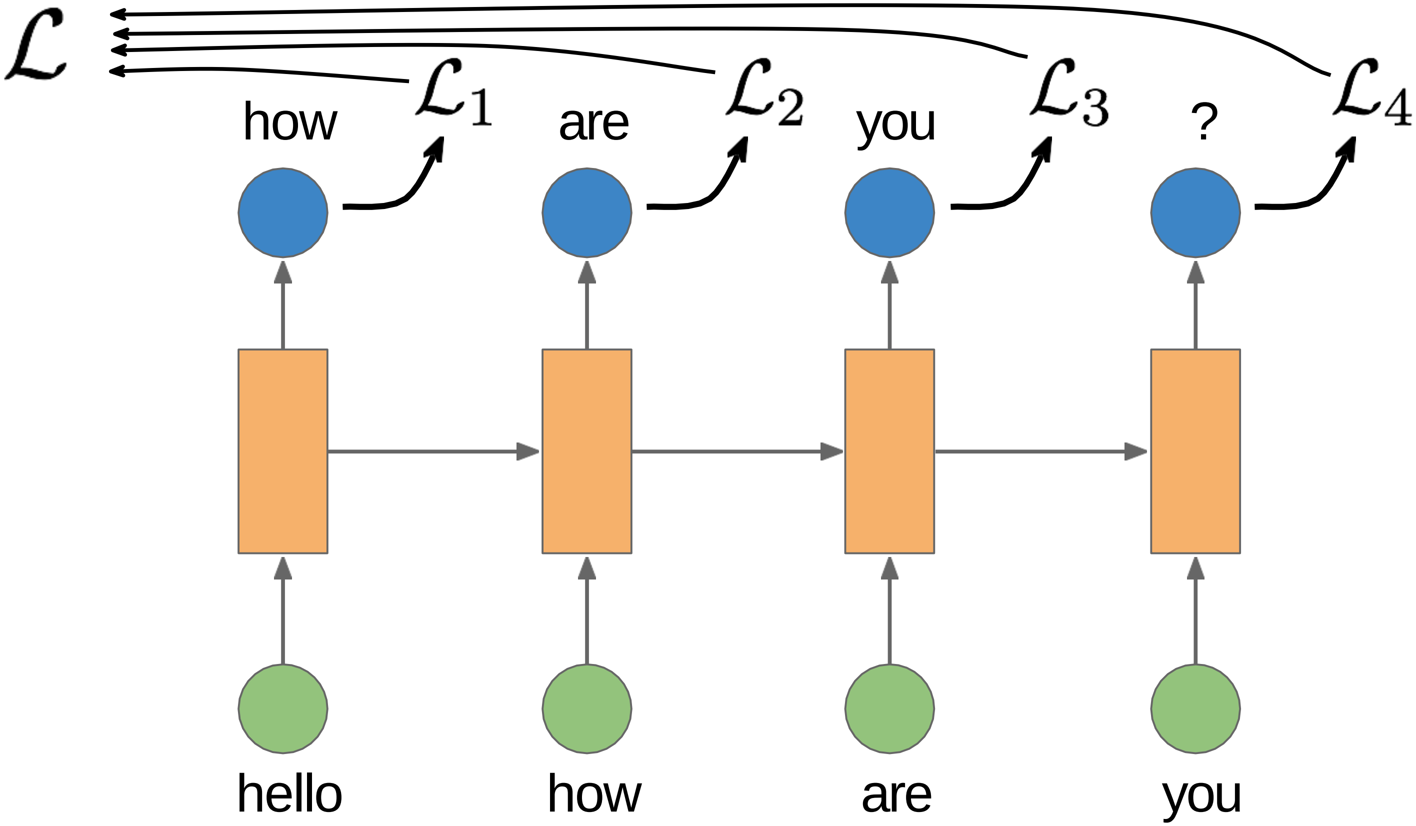
In the case of recurrent neural networks, we have an output per **timestep**

Loss Computation



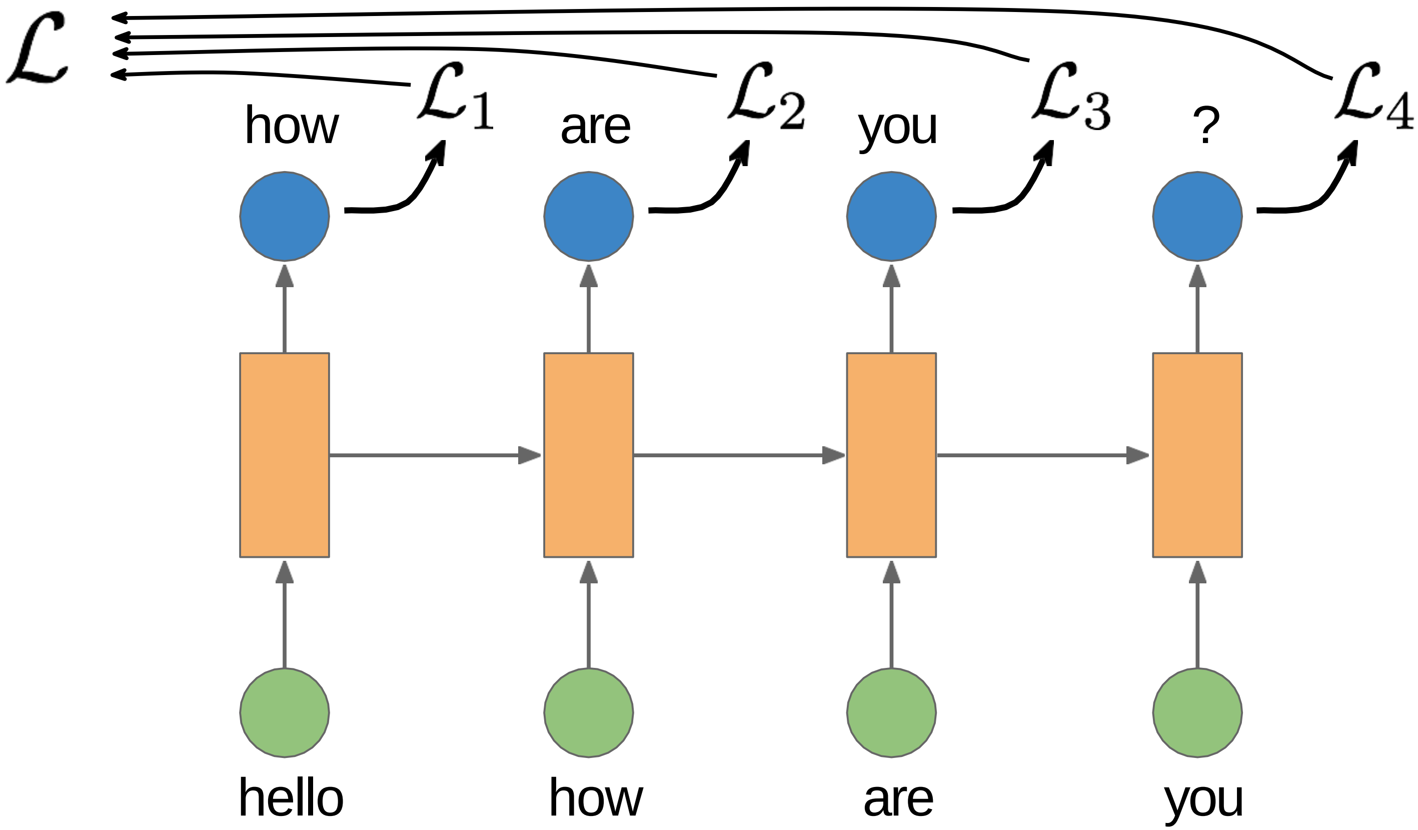
Each of these outputs can be used to get **one loss per timestep**

Loss Computation



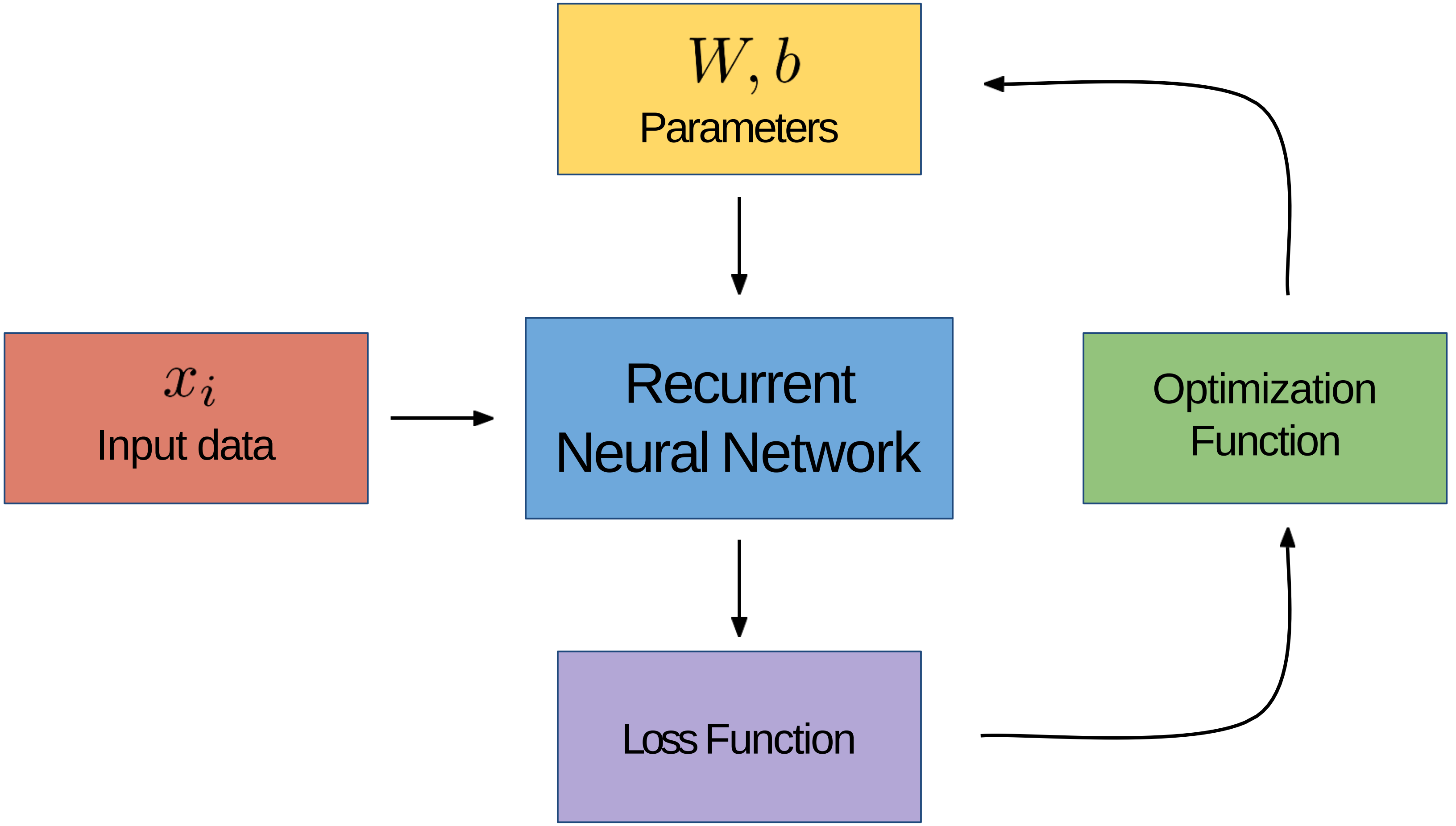
We *add all of these losses* together to get a single loss for our optimization algorithm

Loss Computation



Individual losses are still calculated as before - e.g. using cross entropy loss

Recap

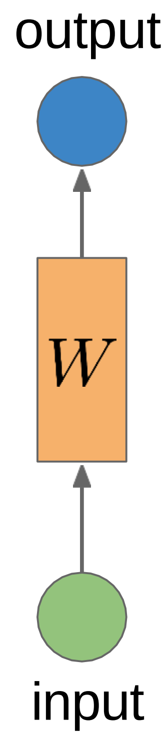


W, b
Parameters
x_i
Input data
Recurrent
Neural Network
Optimization
Function
Loss Function

Backpropagation through time

for recurrent neural networks

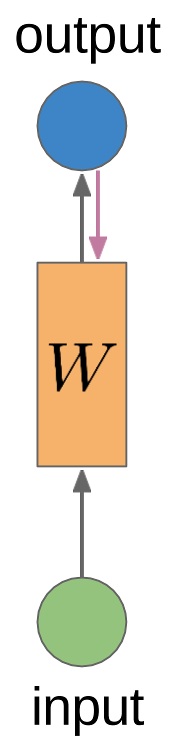
Backpropagation through time



Recall backpropagation in

Feedforward Neural network

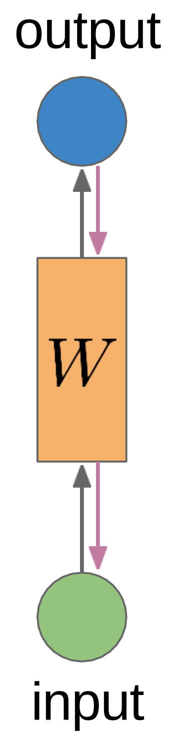
Backpropagation through time



Recall backpropagation in

Feedforward Neural network

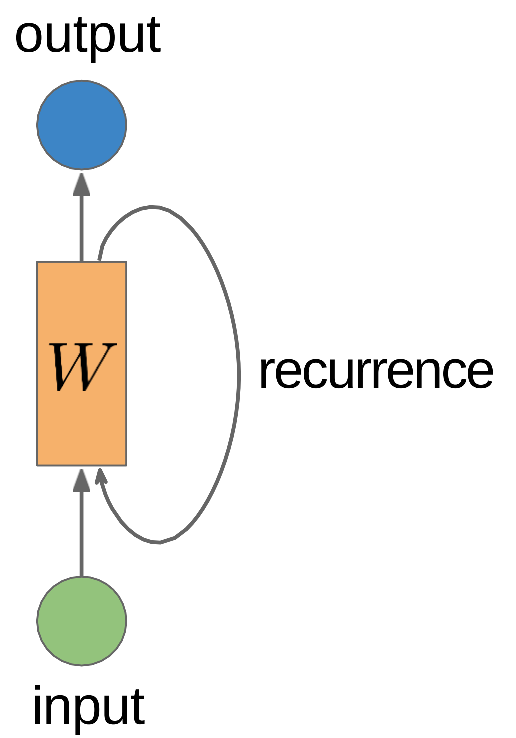
Backpropagation through time



Recall backpropagation in

Feedforward Neural network

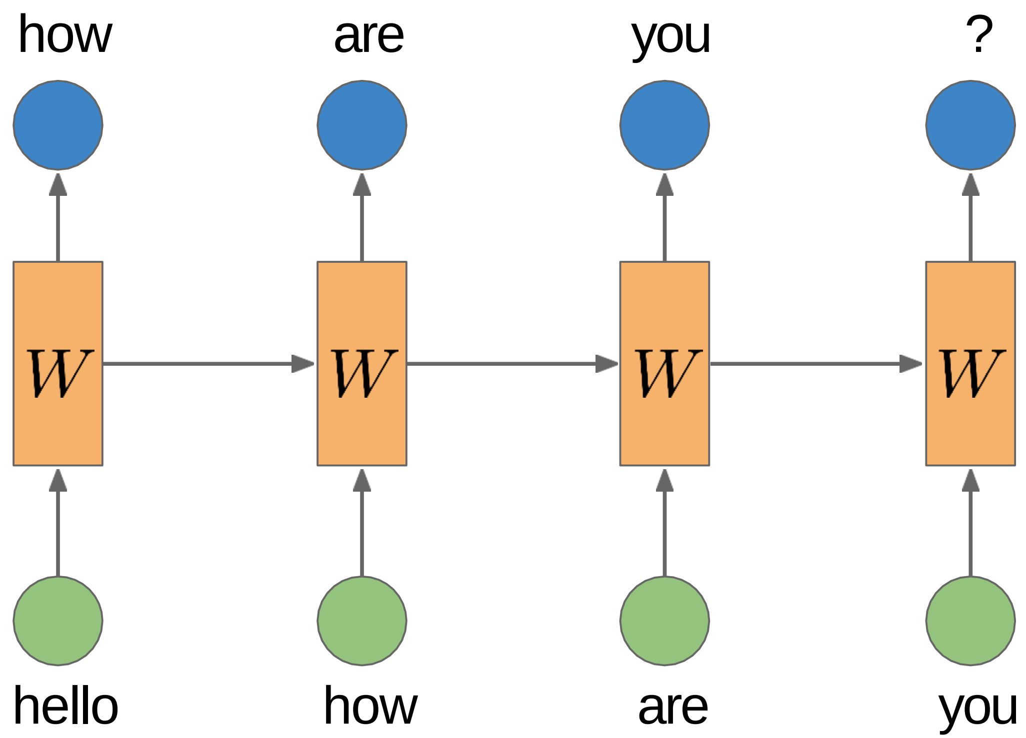
Backpropagation through time



What about backpropagation in recurrent neural networks?

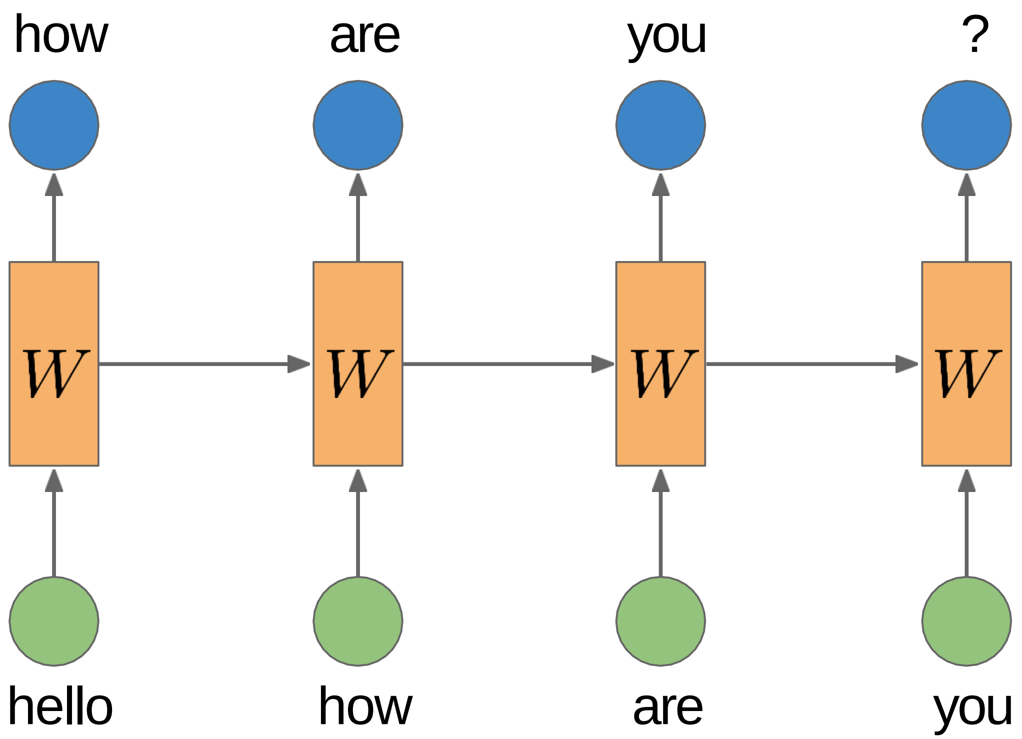
We now have an additional dimension of **time** but with a single **weight matrix** similar to the feedforward NN

Backpropagation through time



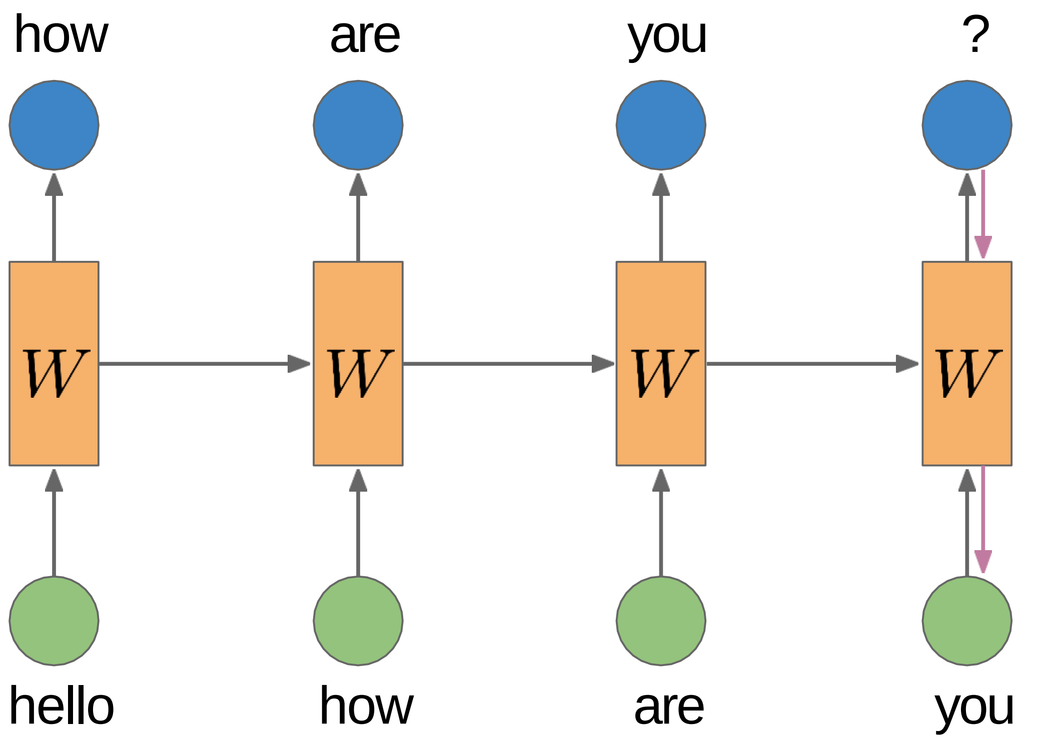
Easier to see when we have ***unrolled*** the RNN

Backpropagation through time



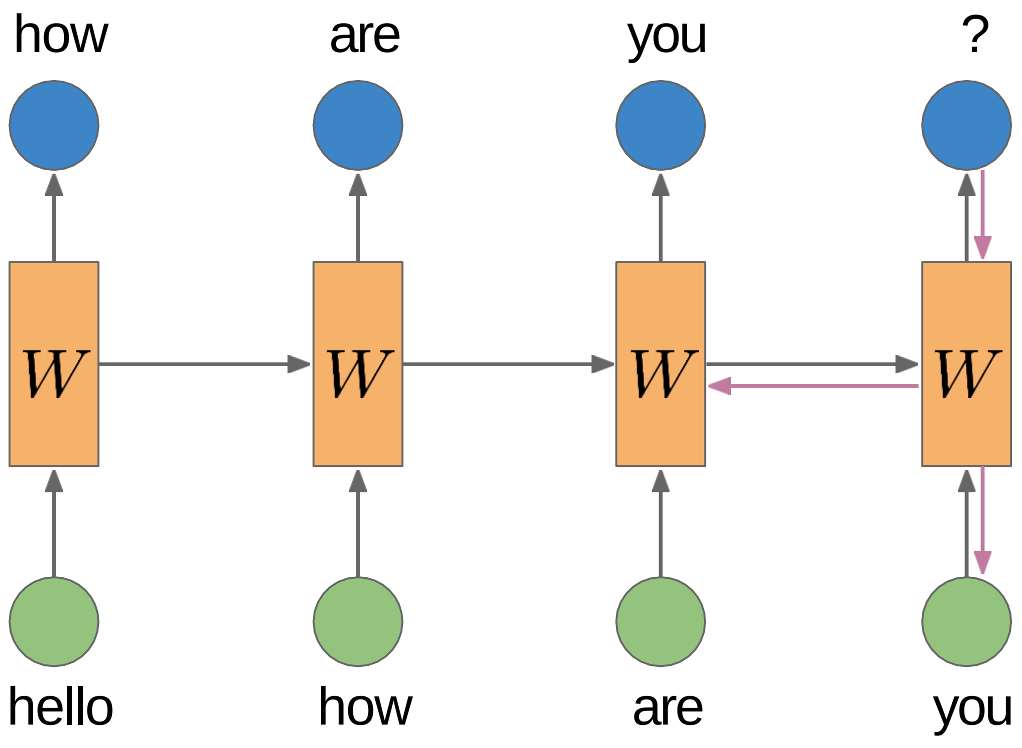
Loss from every timestamp is needed to update the weight parameter W

Backpropagation through time



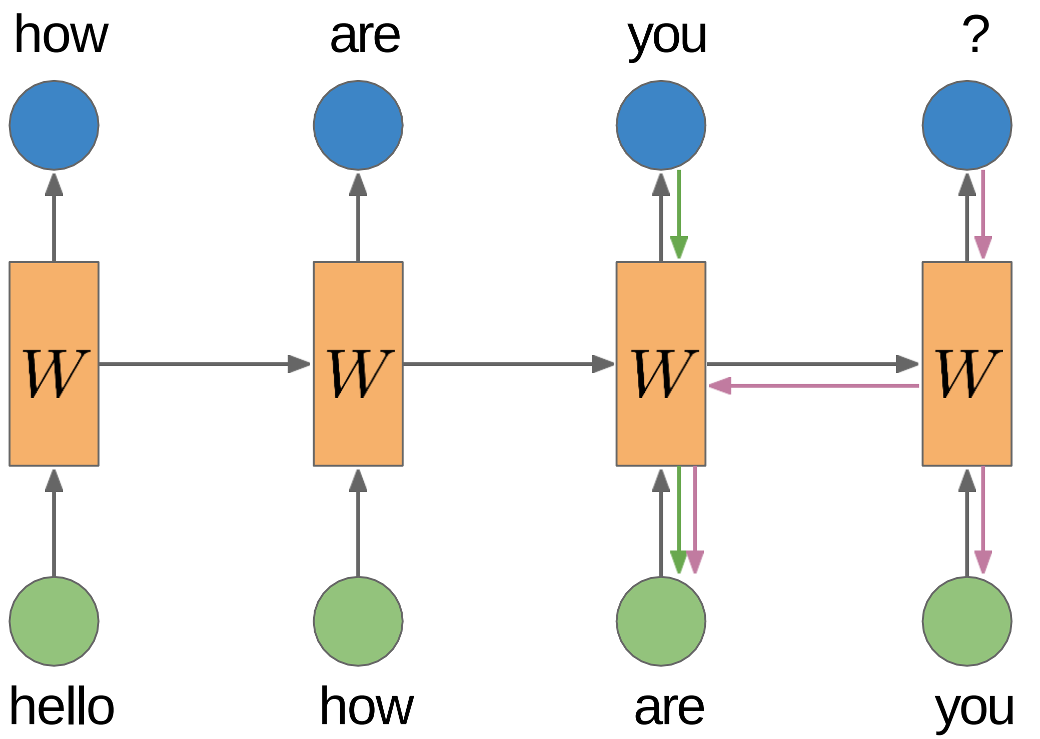
The last timestep propagates its gradient as usual

Backpropagation through time



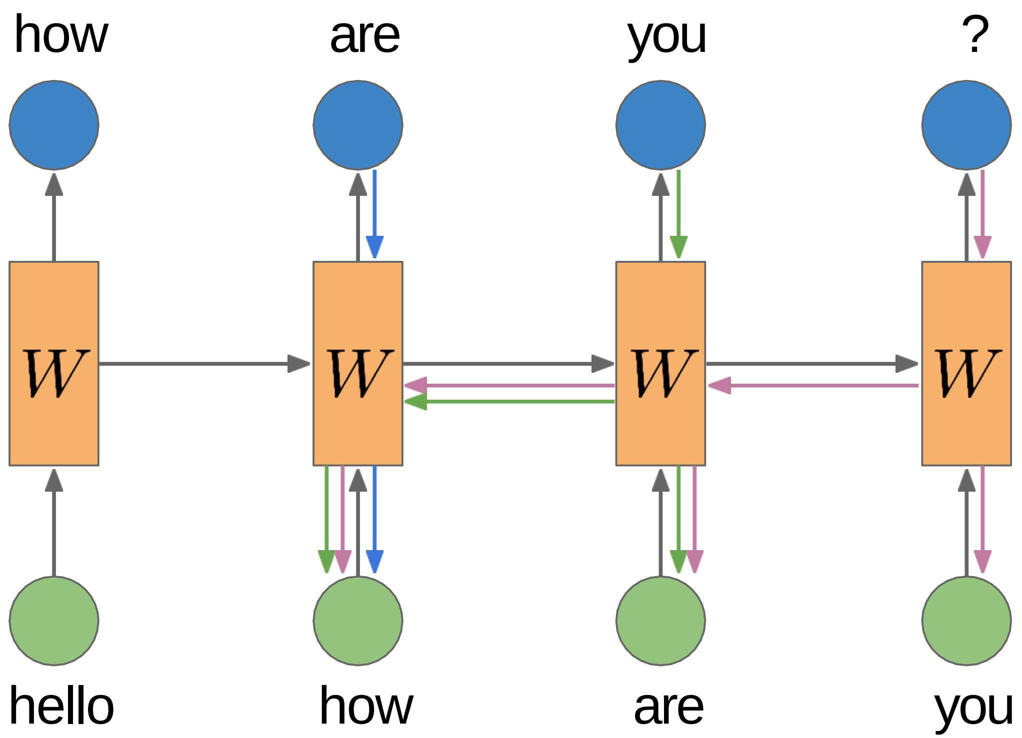
This time, we also propagate the gradient of the last timestep to timestep t - 1

Backpropagation through time



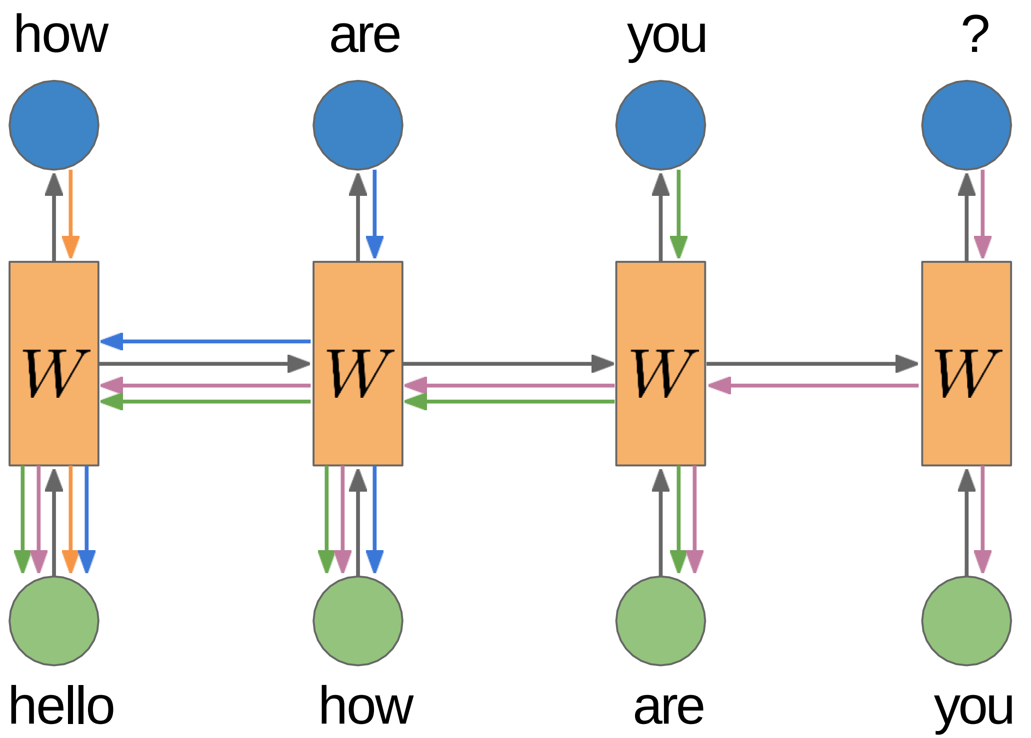
Timestep t - 1 gets gradients from both the output of timestep t - 1 and t !

Backpropagation through time



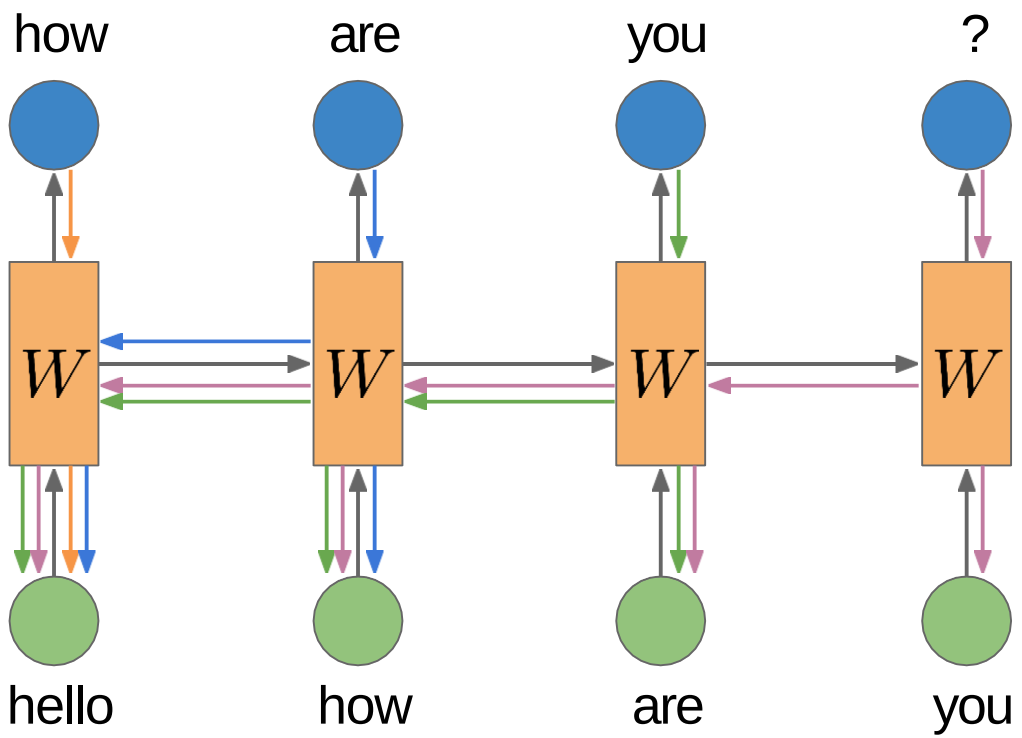
Timestep t - 2 gets gradients from all future timesteps

Backpropagation through time



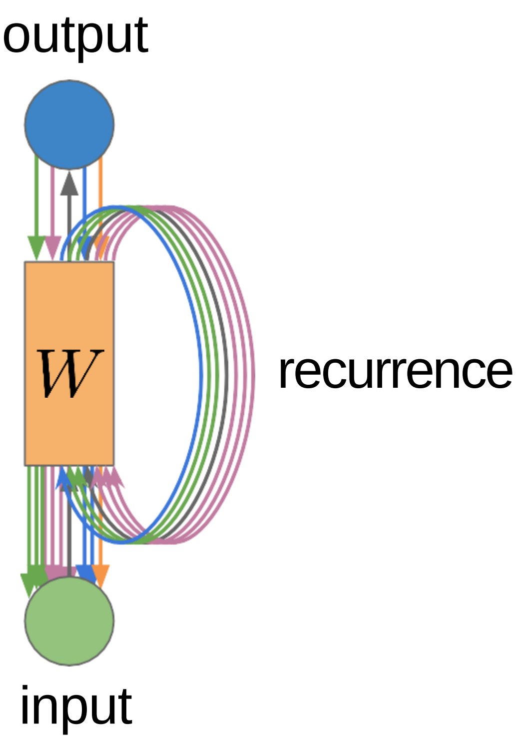
Timestep 1 gets gradients from all future timesteps

Backpropagation through time



Remember, this is an unrolled network - so the parameters are the same in each of the hidden units!

Backpropagation through time



A bit difficult to see in the ***rolled*** RNN...

Issues with vanilla RNN

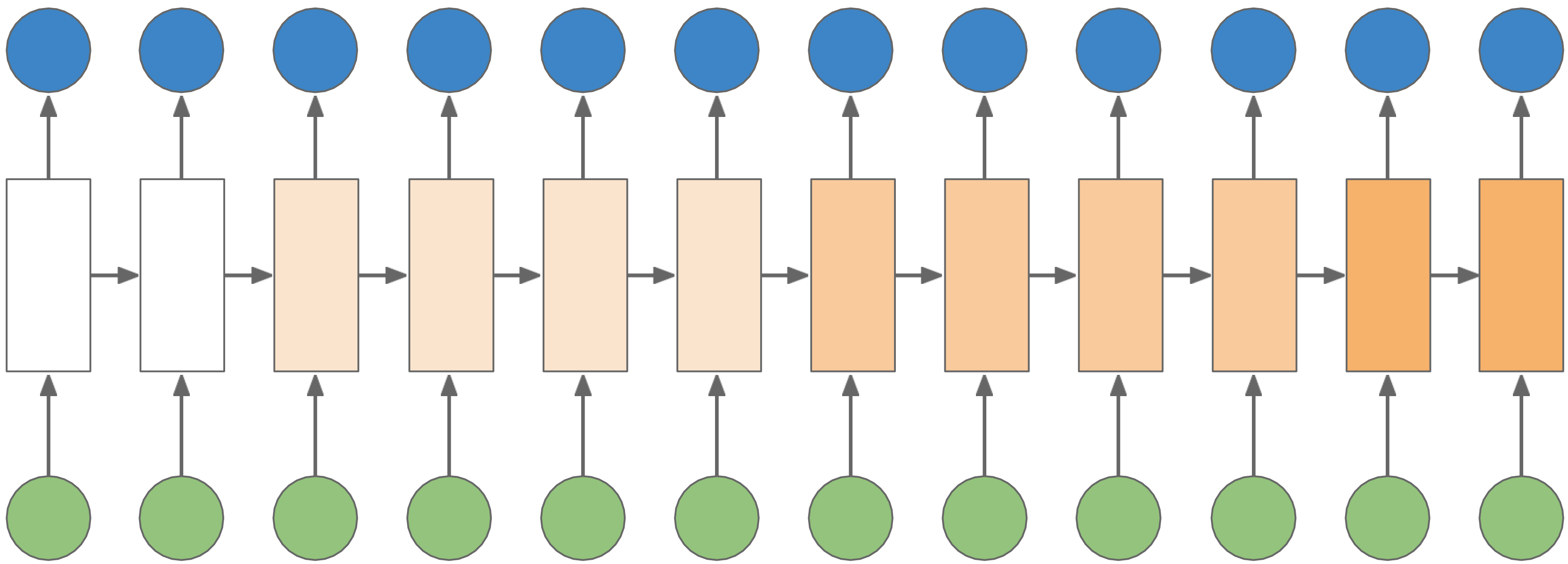
Issues with Vanilla RNN

- Information decay
  - long-term dependencies
- Vanishing gradients
- Exploding gradients

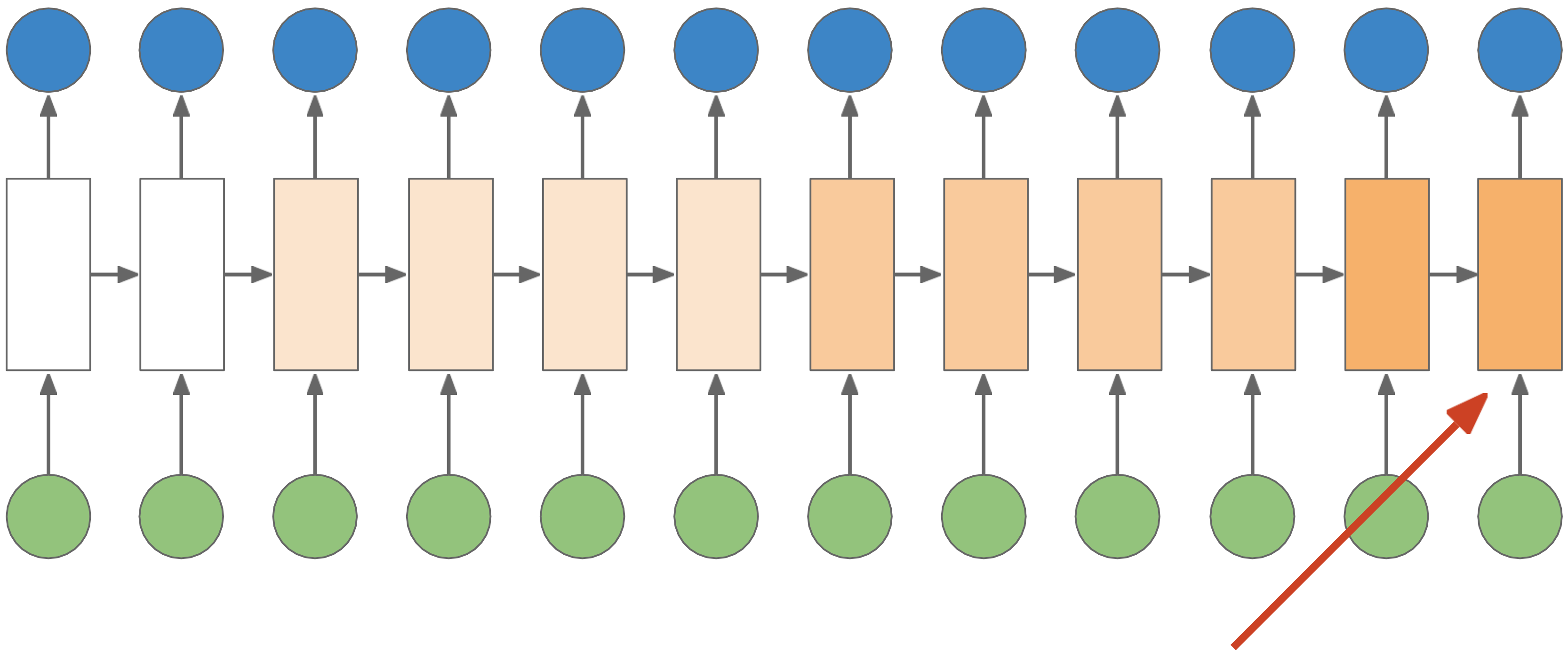
Issues with Vanilla RNN

- Information decay
  - long-term dependencies
- Vanishing gradients
- Exploding gradients

Information decay

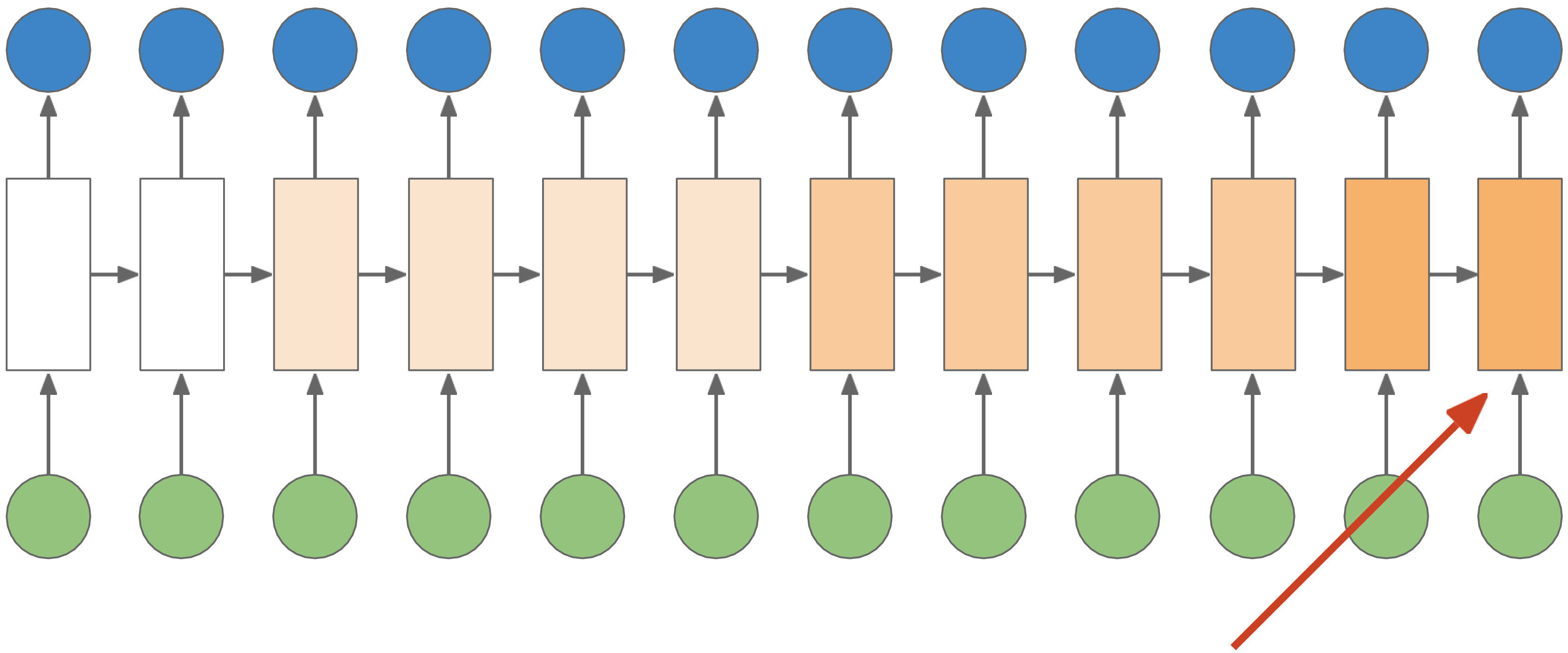


Information decay



The last timestep remembers very little about older timesteps, since it needs to remember information from recent history and the current timestep

Information decay



Remember, the output of the hidden layer is a **fixed length vector**

The network can only remember a limited amount of total information!

Long-term Dependencies

In theory, RNNs are capable of remembering long distance information

Practically, they start forgetting information over long distances as we have seen with the information decay problem

Long-term Dependencies

Words can have long-term dependency on previous words

Ansehen

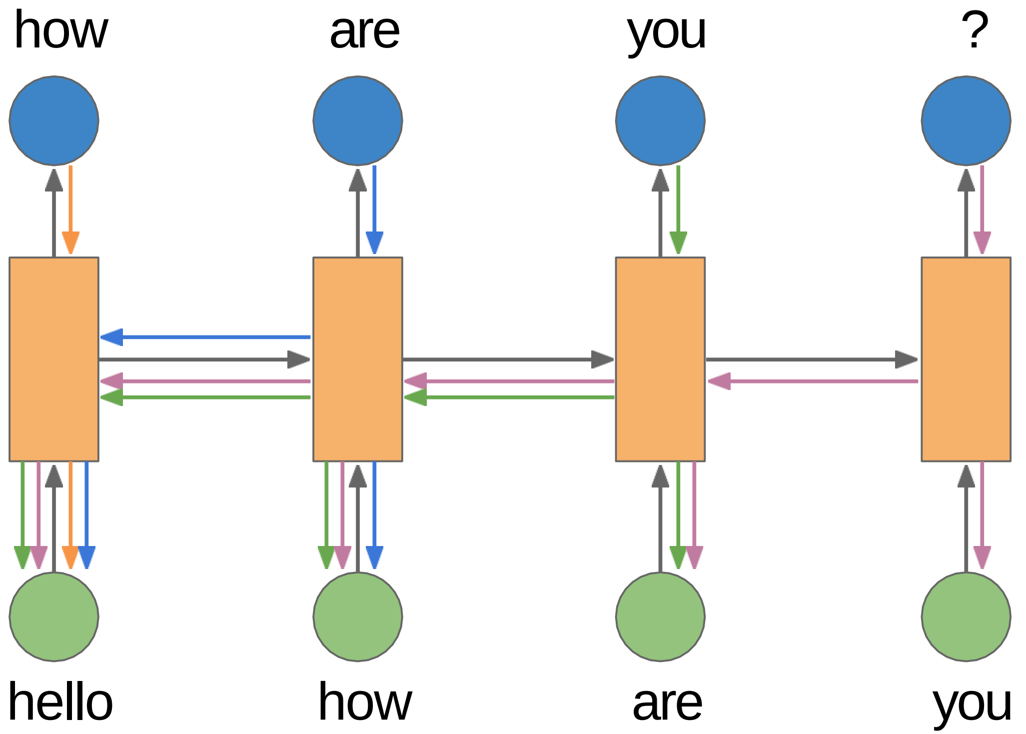
Anna ***sieht*** sich die Talkshow **an**

If the distance between “an” and “sieht” becomes long, the RNN may forget to correctly learn the relationship

Issues with Vanilla RNN

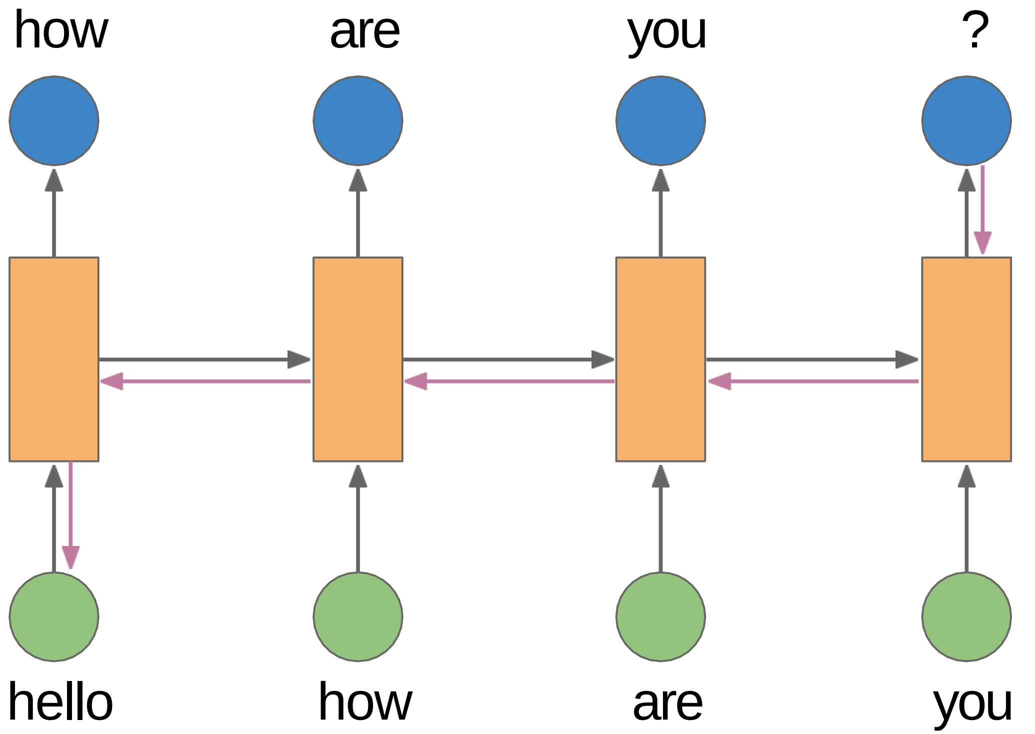
- Information decay
  - long-term dependencies
- Vanishing gradients
- Exploding gradients

Vanishing Gradients



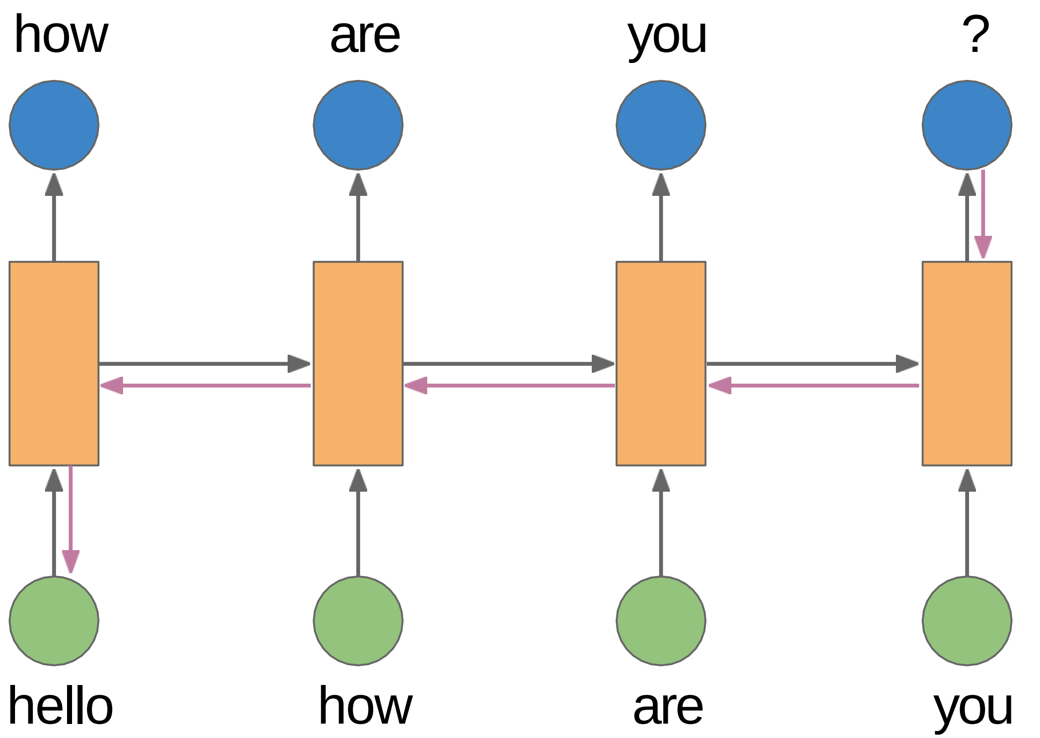
Consider the backpropagation through time

Vanishing Gradients



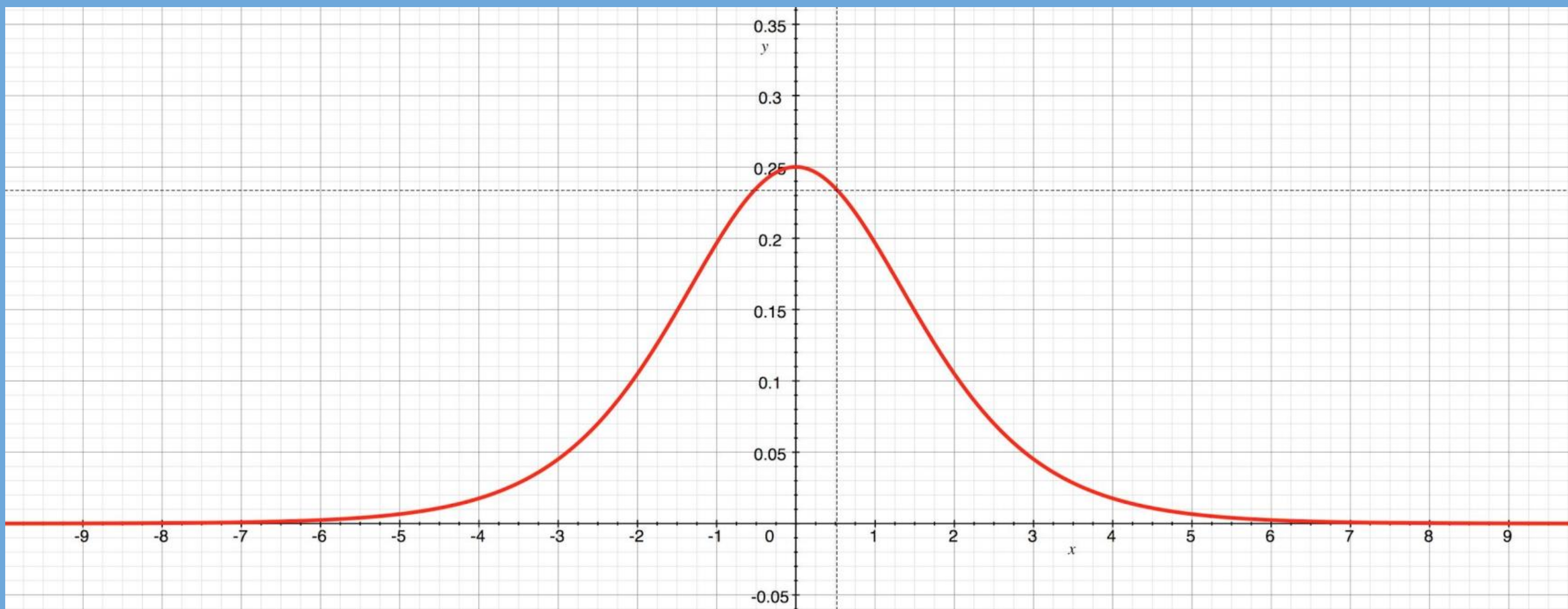
Consider the backpropagation through time

Vanishing Gradients



When the gradients pass through each time step, we have to multiply it with the derivative of the activation function

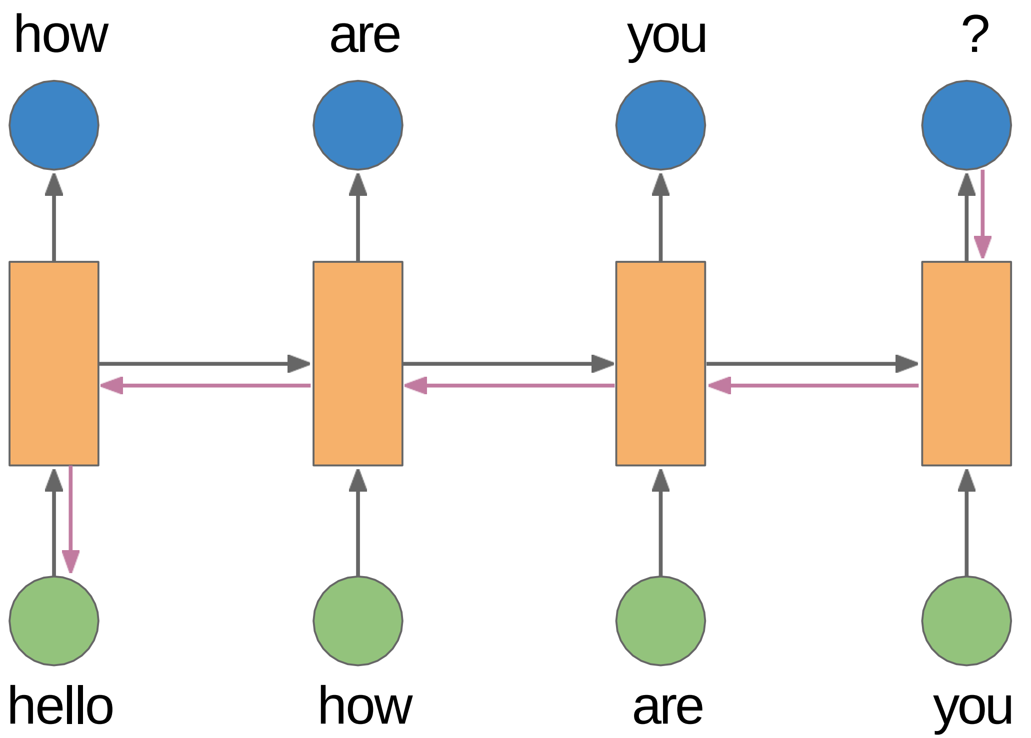
Vanishing Gradients



Gradient of sigmoid function

When the gradients pass through each time step, we have to multiply it with the derivative of the activation function

Vanishing Gradients



Since the gradient of the sigmoid activation is atmost 0.25, we will be multiplying the gradient of the final timestep repeatedly by 0.25

Vanishing Gradients

Since the gradient of the sigmoid activation is atmost `0.25`, we will be multiplying the gradient of the final timestep repeatedly by `0.25`

Vanishing Gradients

Since the gradient of the sigmoid activation is atmost 0.25, we will be multiplying the gradient of the final timestep repeatedly by 0.25

Multiplying several small numbers would result in even smaller numbers!

$$0.25 \times 0.25 \times 0.25 = 0.0156$$

Vanishing Gradients

Since the gradient of the sigmoid activation is atmost 0.25, we will be multiplying the gradient of the final timestep repeatedly by 0.25

Multiplying several small numbers would result in even smaller numbers!

$$0.25 \times 0.25 \times 0.25 = 0.0156$$

Similarly, our gradient from the final timestep becomes very small by the time it reaches a few steps in the beginning

Vanishing Gradients

Since the gradient of the sigmoid activation is atmost 0.25, we will be multiplying the gradient of the final timestep repeatedly by 0.25

Multiplying several small numbers would result in even smaller numbers!

$$0.25 \times 0.25 \times 0.25 = 0.0156$$

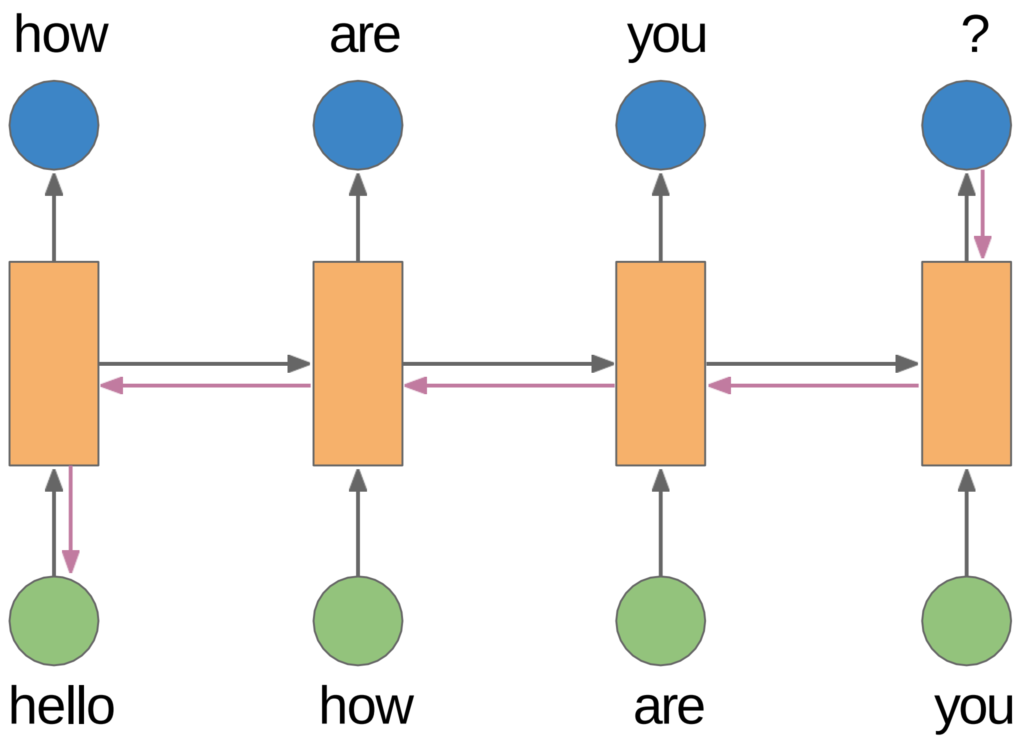
Similarly, our gradient from the final timestep becomes very small by the time it reaches a few steps in the beginning

Hence, our parameters do not change over long distances - but language has a lot of long range dependencies!

Issues with Vanilla RNN

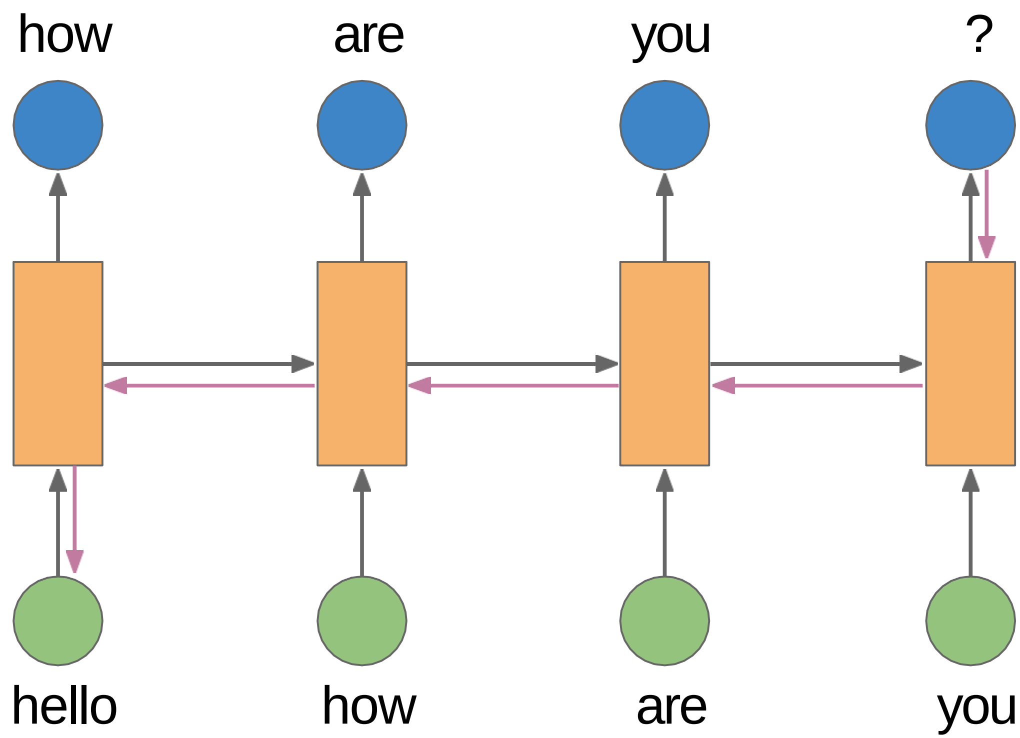
- Information decay
  - long-term dependencies
- Vanishing gradients
- Exploding gradients

Exploding Gradients



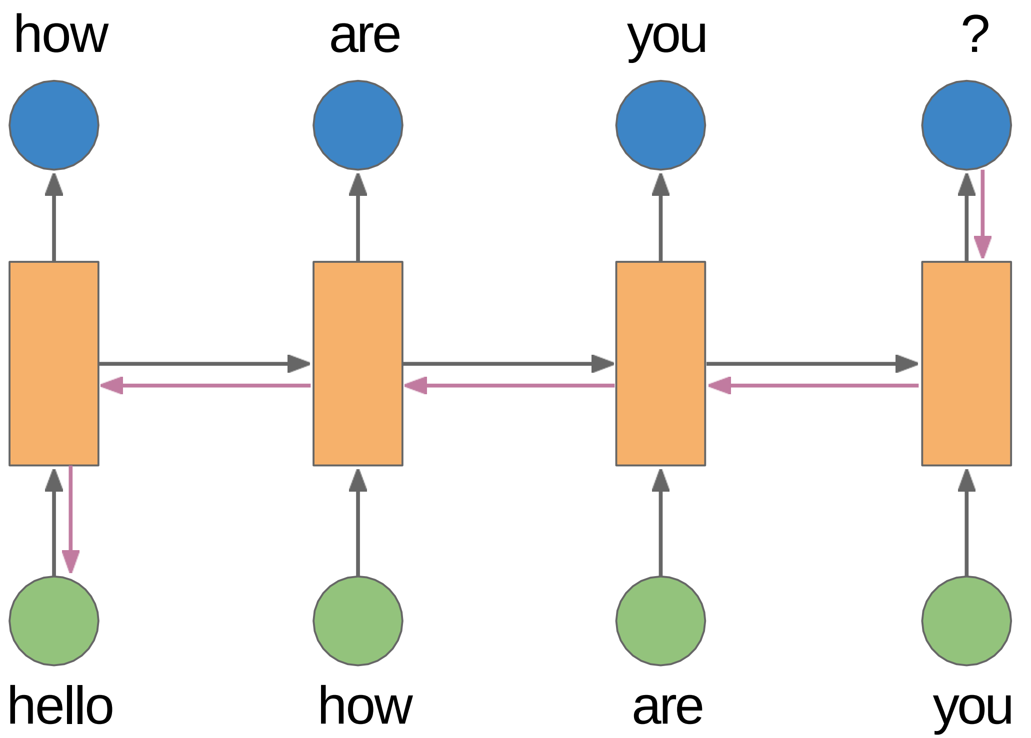
Similar to vanishing gradients, we can also have the problem of an **exploding gradient**

Exploding Gradients



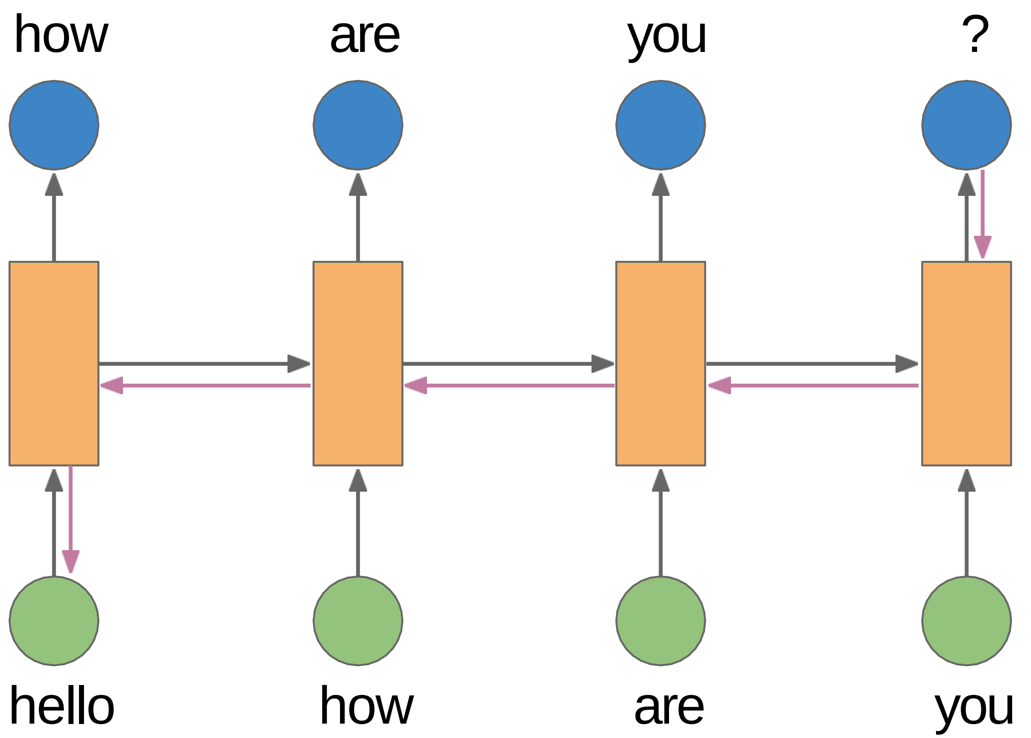
We have activation functions where the gradient can be greater than 1. Our weights themselves can also be greater than 1

Exploding Gradients



In very long sequences, multiplying a lot of large numbers can result in our gradient becoming too large very quickly!

Exploding Gradients



A lot of the time, the problem surfaces as our gradient becomes NaN, and so does our loss!

Issues with Vanilla RNN

- Information decay
  - long-term dependencies
- Vanishing gradients
- Exploding gradients

One potential solution:

Long short-term memory (LSTM)