ECE 1508S2: Applied Deep Learning

Chapter 6: Recurrent NNs

Ali Bereyhi

`ali.bereyhi@utoronto.ca`

Department of Electrical and Computer Engineering

University of Toronto

Winter 2024

Learning from Sequence Data

In many applications, we have *sequence data*, *e.g.*,

- *speech data which is usually a long time series*
- *text data which is sequence of words and letters*
- *financial data that is typically a time-dependent sequence of values*

Learning from Sequence Data

In many applications, we have *sequence data*, *e.g.*,

- *speech data which is usually a long time series*
- *text data which is sequence of words and letters*
- *financial data that is typically a time-dependent sequence of values*

Learning from such data can be inefficient via FNNs, i.e., MLPs and CNNs

- *On one hand, we have long sequence*
  ↳ *This can easily make the NN size infeasible*

Learning from Sequence Data

In many applications, we have *sequence data*, *e.g.*,

- *speech data which is usually a long time series*
- *text data which is sequence of words and letters*
- *financial data that is typically a time-dependent sequence of values*

Learning from such data can be inefficient via FNNs, i.e., MLPs and CNNs

- *On one hand, we have long sequence*
  - ↳ *This can easily make the NN size infeasible*
- *On another hand, we do not have so much features*
  - ↳ *Just think of a long text, where we need to predict the next word in it*

Learning from Sequence Data

In many applications, we have *sequence data*, *e.g.*,

- *speech data which is usually a long time series*
- *text data which is sequence of words and letters*
- *financial data that is typically a time-dependent sequence of values*

Learning from such data can be inefficient via FNNs, i.e., MLPs and CNNs

- *On one hand, we have long sequence*
  - ↳ *This can easily make the NN size infeasible*
- *On another hand, we do not have so much features*
  - ↳ *Just think of a long text, where we need to predict the next word in it*

We need to develop some techniques to handle such data

First, let's see some examples!

Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

Learning from Sequence Data: *Example I*

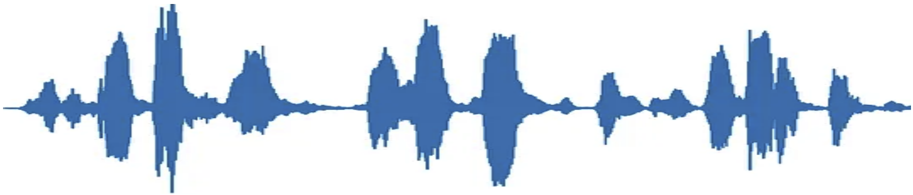
Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

- *This is a classification problem*
  - ↳ *The data-point, i.e., talk is classified either as sport or science*

Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

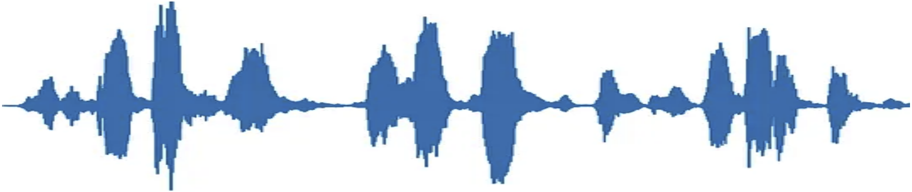
- *This is a classification problem*
  ↳ *The data-point, i.e., talk is classified either as sport or science*
- *What about the data?*
  ↳ *We sample the audio signal at rate 44.1 kHz and quantize the samples*



Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

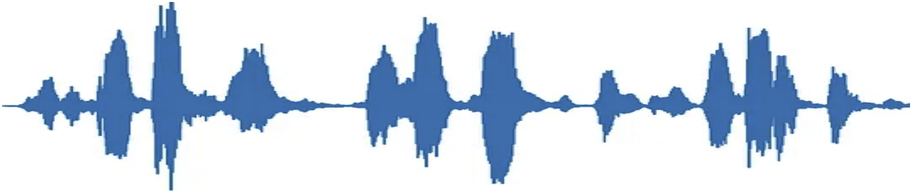
- *This is a classification problem*
  - ↳ *The data-point, i.e., talk is classified either as sport or science*
- *What about the data?*
  - ↳ *We sample the audio signal at rate 44.1 kHz and quantize the samples*
  - ↳ *We put every N successive samples in a frames $\equiv$ vector of samples*



Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

- *This is a classification problem*
  - ↳ *The data-point, i.e., talk is classified either as sport or science*
- *What about the data?*
  - ↳ *We sample the audio signal at rate 44.1 kHz and quantize the samples*
  - ↳ *We put every N successive samples in a frames ≡ vector of samples*
  - ↳ *We store the 15-minute talk as a sequence of time frames*

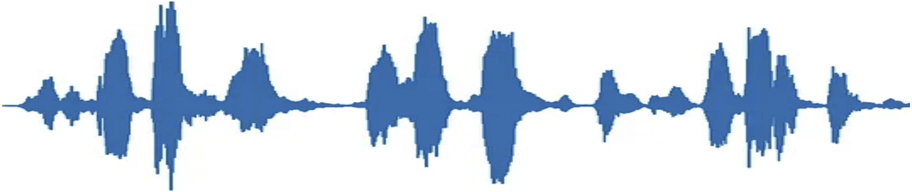


$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\mathbf{x}[3]$ $\cdots$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$ $\cdots$

Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

- *This is a classification problem*
  - ↳ *The data-point, i.e., talk is classified either as sport or science*
- *What about the data?*
  - ↳ *We sample the audio signal at rate 44.1 kHz and quantize the samples*
  - ↳ *We put every N successive samples in a frames $\equiv$ vector of samples*
  - ↳ *We store the 15-minute talk as a sequence of time frames*
  - ↳ *We may also store frequency frames from the Fourier transform*



$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\mathbf{x}[3]$ $\cdots$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$ $\cdots$

Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

- *This is a binary classification problem*
- *What about the data? each data-point is a sequence of vectors*

Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

- *This is a binary classification problem*
- *What about the data? each data-point is a sequence of vectors*

Say we make frames of 512 samples; then, we roughly have

77,587 frames = 39,690,000 samples

Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

- *This is a binary classification problem*
- *What about the data? each data-point is a sequence of vectors*

Say we make frames of 512 samples; then, we roughly have

77,587 frames = 39,690,000 samples

But do we need to pass them all together through an NN?

- *It does not seem to be!*

Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

- *This is a binary classification problem*
- *What about the data? each data-point is a sequence of vectors*

Say we make frames of 512 samples; then, we roughly have

77,587 frames = 39,690,000 samples

But do we need to pass them all together through an NN?

- *It does not seem to be!*
  - ↳ *We can classify based of simple words and expressions in the talk*
  - ↳ *Processing all samples together seems to be an unnecessary hardness*

Learning from Sequence Data: *Example I*

Assume we listen to a 15-minute talk: we want to find out whether it is about sport or science

- *This is a binary classification problem*
- *What about the data? each data-point is a sequence of vectors*

Say we make frames of 512 samples; then, we roughly have

$$77{,}587 \text{ frames} = 39{,}690{,}000 \text{ samples}$$

But do we need to pass them all together through an NN?

- *It does not seem to be!*
  - ↳ *We can classify based of simple words and expressions in the talk*
  - ↳ *Processing all samples together seems to be an unnecessary hardness*
- *It does not generalize well*
  - ↳ *We want to classify shorter and longer talks as well*

Learning from Sequence Data: *Example II*

Now let's consider another example: *we have a long text and want to learn what is the next word in the sentence*

- *This is a prediction task*
  - ↳ *Given previous text we predict the next outcome*

Learning from Sequence Data: *Example II*

Now let's consider another example: *we have a long text and want to learn what is the next word in the sentence*

- *This is a prediction task*
  - ↳ *Given previous text we predict the next outcome*
- *What about the data?*
  - ↳ *We parse the text into a sequence of words*
  - ↳ *We represent the letters of each word with their numeric, e.g., ASCII*
  - ↳ *We save each word into an N-dimensional frame*

Learning from Sequence Data: *Example II*

Now let's consider another example: *we have a long text and want to learn what is the next word in the sentence*

- *This is a prediction task*
  - ↳ *Given previous text we predict the next outcome*
- *What about the data?*
  - ↳ *We parse the text into a sequence of words*
  - ↳ *We represent the letters of each word with their numeric, e.g., ASCII*
  - ↳ *We save each word into an N-dimensional frame*

`...therapy.  UofT undergraduate students explore the use of AI to treat speech`

$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\mathbf{x}[3]$ $\cdots$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$ $\cdots$ $\mathbf{x}[T]$ $\mathbf{y} = ?$

Learning from Sequence Data: *Example III*

Another example: *we have a sequence of stock prices and are interested in the future price*

- *This is again a prediction task*
  ↳ *Given previous prices we predict the future price*

Learning from Sequence Data: *Example III*

Another example: *we have a sequence of stock prices and are interested in the future price*

- *This is again a prediction task*
  - ↳ *Given previous prices we predict the future price*
- *What about the data?*
  - ↳ *We put daily prices in form of a sequence*
  - ↳ *We collect every couple of prices along with other indicators into a vector*

Learning from Sequence Data: *Example III*

Another example: *we have a sequence of stock prices and are interested in the future price*

- *This is again a prediction task*
  ↳ *Given previous prices we predict the future price*
- *What about the data?*
  ↳ *We put daily prices in form of a sequence*
  ↳ *We collect every couple of prices along with other indicators into a vector*

In all these problems: *we have a sequence of data and we intend to learn from them in a generalizable way*

Learning from Sequence Data: *Example III*

Another example: *we have a sequence of stock prices and are interested in the future price*

- *This is again a prediction task*
  - ↳ *Given previous prices we predict the future price*
- *What about the data?*
  - ↳ *We put daily prices in form of a sequence*
  - ↳ *We collect every couple of prices along with other indicators into a vector*

In all these problems: *we have a sequence of data and we intend to learn from them in a generalizable way*

↳ *We clearly need a memory component that can potentially be infinite*

Learning from Sequence Data: *Example III*

Another example: *we have a sequence of stock prices and are interested in the future price*

- *This is again a prediction task*
  - ↳ *Given previous prices we predict the future price*
- *What about the data?*
  - ↳ *We put daily prices in form of a sequence*
  - ↳ *We collect every couple of prices along with other indicators into a vector*

In all these problems: *we have a sequence of data and we intend to learn from them in a generalizable way*

↳ *We clearly need a memory component that can potentially be infinite*

↳ *We should keep track of this infinitely large memory via limited storage*

Predicting Next Word

Let's start with a simple example: *we want to train a neural network that gets a sentence and complete the next word*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{y} = \mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Predicting Next Word

Let's start with a simple example: *we want to train a neural network that gets a sentence and complete the next word*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{y} = \mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

+ *How does the training dataset look like?*
– We are given with several long texts in the same context: *at each entry of sequence in each of these texts the whole sequence is the data-point and the next word is label*

Predicting Next Word

Let's make some specification to clarify the problem

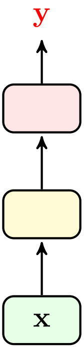
- *Each entry is a vector of dimension N, i.e., $\mathbf{x}[t] \in \mathbb{R}^N$*
- *We have an NN with some hidden layers to train*

Predicting Next Word

Let's make some specification to clarify the problem

- *Each entry is a vector of dimension N, i.e., $\mathbf{x}[t] \in \mathbb{R}^N$*
- *We have an NN with some hidden layers to train*

We show our NN compactly with the following diagram



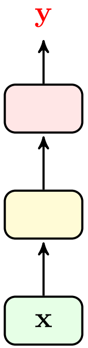
In this diagram

Predicting Next Word

Let's make some specification to clarify the problem

- *Each entry is a vector of dimension N, i.e., $\mathbf{x}[t] \in \mathbb{R}^N$*
- *We have an NN with some hidden layers to train*

We show our NN compactly with the following diagram



In this diagram

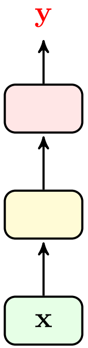
- *Green box shows the input layer*

Predicting Next Word

Let's make some specification to clarify the problem

- *Each entry is a vector of dimension N, i.e., $\mathbf{x}[t] \in \mathbb{R}^N$*
- *We have an NN with some hidden layers to train*

We show our NN compactly with the following diagram



In this diagram

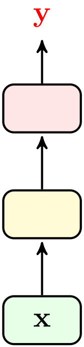
- *Green box shows the input layer*
- *Yellow box includes hidden layers*
  ↳ *It could be several layers*

Predicting Next Word

Let's make some specification to clarify the problem

- *Each entry is a vector of dimension N, i.e., $\mathbf{x}[t] \in \mathbb{R}^N$*
- *We have an NN with some hidden layers to train*

We show our NN compactly with the following diagram



In this diagram

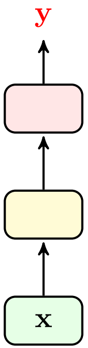
- *Green box shows the input layer*
- *Yellow box includes hidden layers*
  - ↳ *It could be several layers*
- *Red box is the output layer*

Predicting Next Word

Let's make some specification to clarify the problem

- *Each entry is a vector of dimension N, i.e., $\mathbf{x}[t] \in \mathbb{R}^N$*
- *We have an NN with some hidden layers to train*

We show our NN compactly with the following diagram

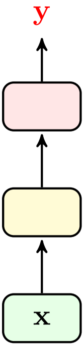


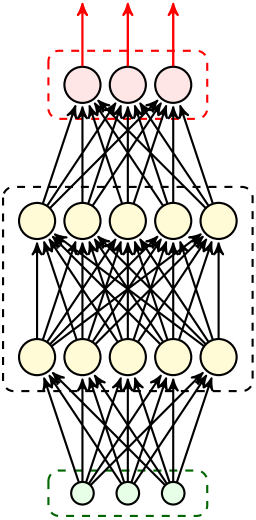
In this diagram

- *Green box shows the input layer*
- *Yellow box includes hidden layers*
  - ↳ *It could be several layers*
- *Red box is the output layer*
- *Arrows refer to all links between the layers*
  - ↳ *They could be learnable*

Predicting Next Word

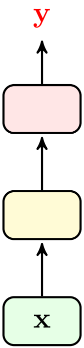
For instance, we could think of following equivalence





Predicting Next Word: *MLP*

Let's try solving this problem with a simple MLP



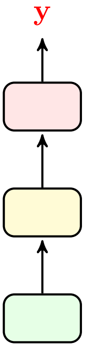
We have a fully-connected FNN that

- *takes N inputs, i.e., single entry*
- *returns N outputs, i.e., predicted next word*

We train this MLP

- *we go over all text*

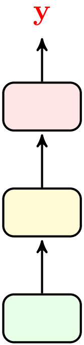
Predicting Next Word: *MLP*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

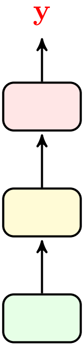
Predicting Next Word: *MLP*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

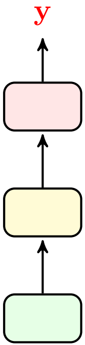
Predicting Next Word: *MLP*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

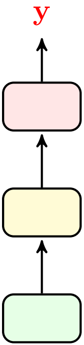
Predicting Next Word: *MLP*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

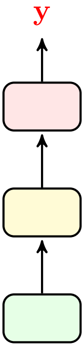
Predicting Next Word: *MLP*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

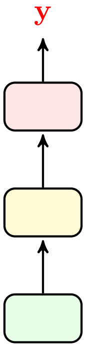
Predicting Next Word: *MLP*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

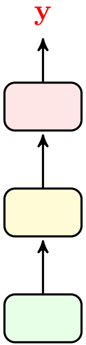
Predicting Next Word: *MLP*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Predicting Next Word: *MLP*

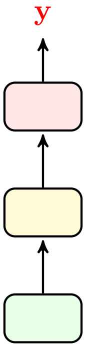


$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Does FNN predict "`her`"?

Predicting Next Word: *MLP*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Does FNN predict "`her`"? No! How can it remember we are talking about Julia?!

Predicting Next Word: *MLP*

$\mathbf{x}[t-6]$	$\mathbf{x}[t-5]$	$\mathbf{x}[t-4]$	$\mathbf{x}[t-3]$	$\mathbf{x}[t-2]$	$\mathbf{x}[t-1]$	$\mathbf{x}[t]$	$\mathbf{x}[t+1]$

... `Julia has been nominated to receive Alexander von Humboldt Prize for her`

Predicting Next Word: *MLP*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Each time the FNN gets trained with a new sample, *it forgets previous text*

Predicting Next Word: *MLP*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Each time the FNN gets trained with a new sample, *it forgets previous text*

- *At the end, it has a set of weights that are average over all predictions*
  - ↳ *many of these predictions are irrelevant, e.g.,*
    - ↳ "`nominated to`" *is followed by* "`receive`": *has nothing to say about* "`her`"

Predicting Next Word: *MLP*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Each time the FNN gets trained with a new sample, *it forgets previous text*

- *At the end, it has a set of weights that are average over all predictions*
  - ↳ *many of these predictions are irrelevant, e.g.,*
    - ↳ "`nominated to`" *is followed by* "`receive`": *has nothing to say about* "`her`"
- *By the time we get to* $\mathbf{x}[t]$*, the FNN gets no input that connects it to Julia*

Predicting Next Word: *MLP*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Each time the FNN gets trained with a new sample, *it forgets previous text*

- *At the end, it has a set of weights that are average over all predictions*
  - ↳ *many of these predictions are irrelevant, e.g.,*
    - ↳ "`nominated to`" *is followed by* "`receive`": *has nothing to say about* "`her`"
- *By the time we get to* $\mathbf{x}[t]$, *the FNN gets no input that connects it to Julia*

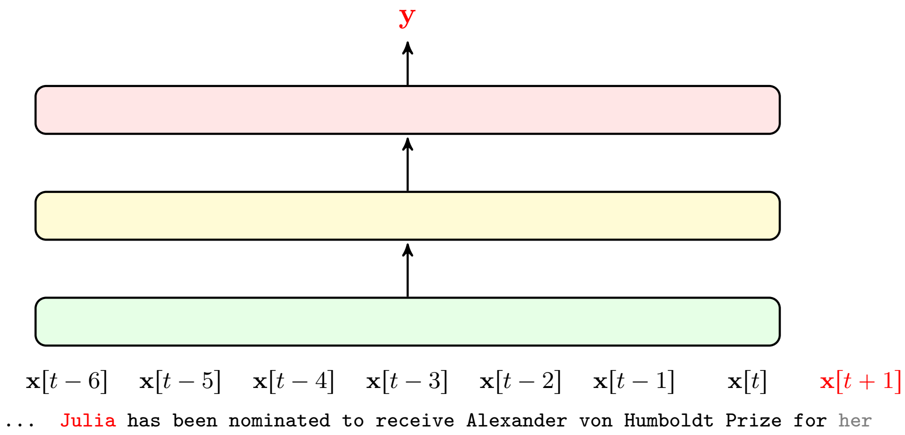
This indicates that we need to *make a memory component for our NN*

Predicting Next Word: *Large MLP*

Maybe we could give more inputs to the FNN!

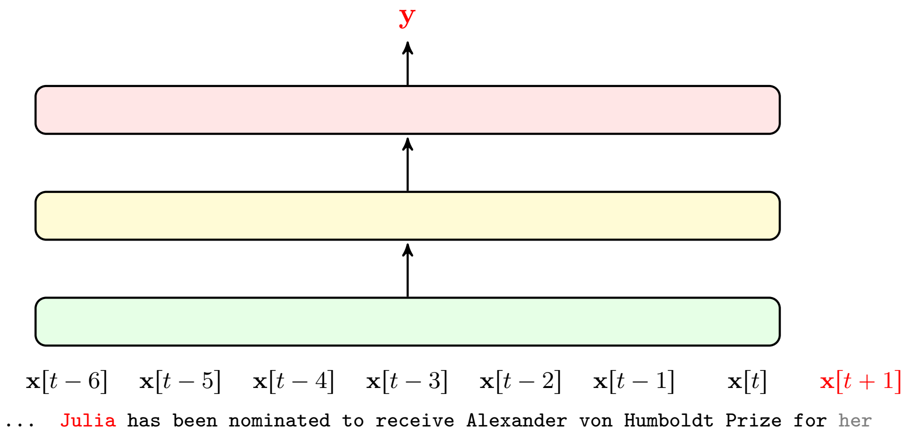
Predicting Next Word: *Large MLP*

Maybe we could give more inputs to the FNN!



Predicting Next Word: *Large MLP*

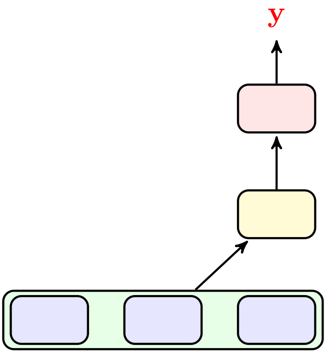
Maybe we could give more inputs to the FNN!



But, what is Julia has been mentioned 10 pages ago? Forget about large MLPs!

Predicting Next Word: CNN

Let's now think about CNNs

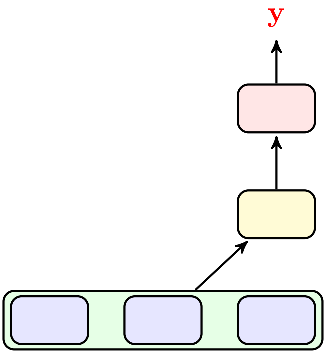


We have a fully-connected FNN that

- *takes N inputs, i.e., single entry*
- *returns N outputs, i.e., predicted next word*

Predicting Next Word: CNN

Let's now think about CNNs



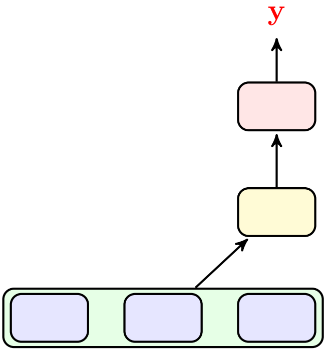
We have a fully-connected FNN that

- *takes N inputs, i.e., single entry*
- *returns N outputs, i.e., predicted next word*

We use convolution to

- *look into a larger input by a filter of size N*
- *extract features from a larger part of text and pass it to hidden layers*

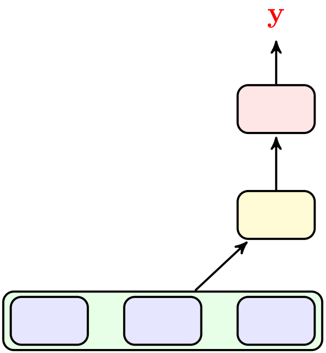
Predicting Next Word: *CNN*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

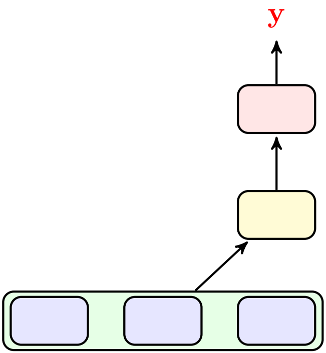
Predicting Next Word: *CNN*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

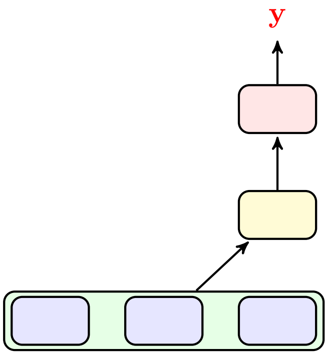
Predicting Next Word: *CNN*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

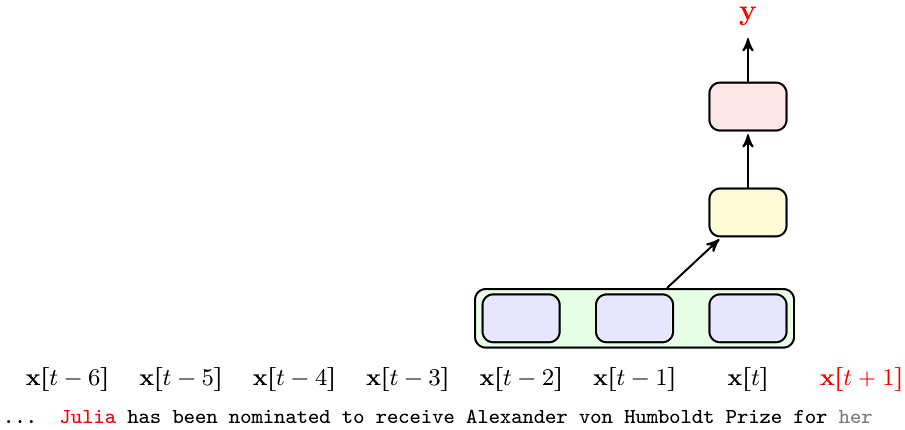
Predicting Next Word: *CNN*



$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

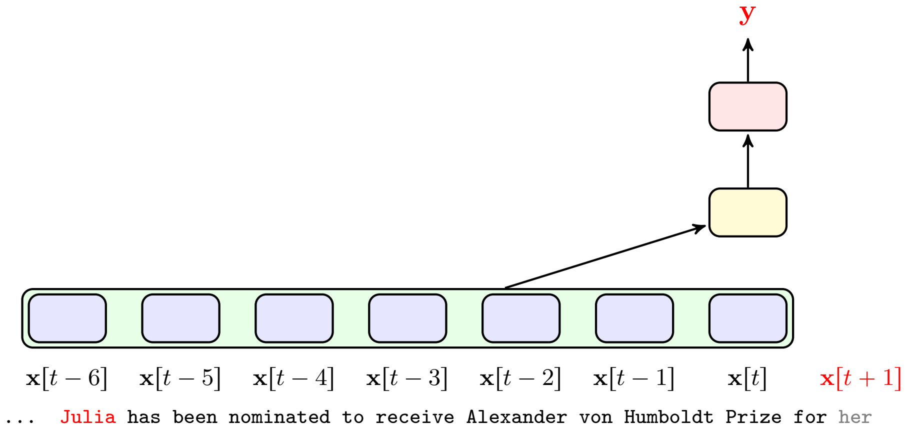
`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Predicting Next Word: *CNN*



Yet, it doesn't seem to remember Julia! Unless we slide over the whole text!

Predicting Next Word: *Large CNN*



Though better than MLP, it is still infeasible to track the whole text

Finite Memory: *Root Problem*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

The problem with *all architectures we know* is that *they have finite memory*

Finite Memory: *Root Problem*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

The problem with *all architectures we know* is that *they have finite memory*

- *They can only remember from their input*
  - ↳ *we always give them independent inputs with similar features*
  - ↳ *they gradually learn to connect any of such inputs toits label*

Finite Memory: *Root Problem*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

The problem with *all architectures we know* is that *they have finite memory*

- *They can only remember from their input*
  - ↳ *we always give them independent inputs with similar features*
  - ↳ *they gradually learn to connect any of such inputs toits label*
- *If we need to remember for long time we have to give them huge inputs*

Finite Memory: *Root Problem*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\mathbf{x}[t+1]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

The problem with *all architectures we know* is that *they have finite memory*

- *They can only remember from their input*
  - ↳ *we always give them independent inputs with similar features*
  - ↳ *they gradually learn to connect any of such inputs toits label*
- *If we need to remember for long time we have to give them huge inputs*
- *But the memory component does not seem to be so huge*
  - ↳ *We may only remember that Julia is a "single person" and "female"*
  - ↳ *If text now switches to Theodore we should refresh our memory that we are talking about a "single person" and "male"*

Finite Memory Component *with Infinite Response*

Component We Miss

We need to extract a memory component from our data that is finite in size but has been influenced (at least theoretically) infinitely

Finite Memory Component *with Infinite Response*

Component We Miss

We need to extract a memory component from our data that is finite in size but has been influenced (at least theoretically) infinitely

State-space model can help us building such memory component: *it is widely used in control theory to describe evolution of a system over time*

Finite Memory Component *with Infinite Response*

Component We Miss

We need to extract a memory component from our data that is finite in size but has been influenced (at least theoretically) infinitely

State-space model can help us building such memory component: *it is widely used in control theory to describe evolution of a system over time*

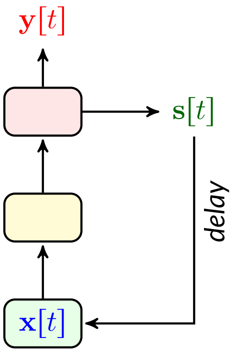
Assume $\mathbf{x}[t]$ is an input to a system at time t: the system returns an output $\mathbf{y}[t]$ to this input and a state variable $\mathbf{s}[t]$. The output in the next time, i.e., $t+1$, depends on the new input and current state, i.e.,

$$\mathbf{y}[t+1], \mathbf{s}[t+1] = f\left(\mathbf{x}[t+1], \mathbf{s}[t]\right)$$

In the above representation: *the state is a finite-size variable that carries information for infinitely long time*

State-Space Model for *NNs*

We can look at a NN as a state-dependent system

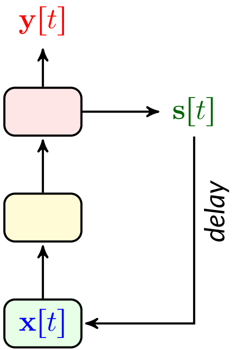


In this architecture, the NN

- *takes* $\mathbf{x}[t]$ *and previous state* $\mathrm{s}[t-1]$ *as inputs*
- *returns* $\mathbf{y}[t]$ *and new state* $\mathbf{s}[t]$ *as outputs*

State-Space Model for *NNs*

We can look at a NN as a state-dependent system



In this architecture, the NN

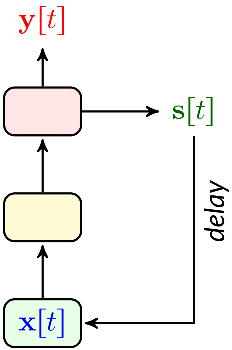
- *takes* $\mathbf{x}[t]$ *and previous state* $\mathbf{s}[t-1]$ *as inputs*
- *returns* $\mathbf{y}[t]$ *and new state* $\mathbf{s}[t]$ *as outputs*

This NN captures temporal behavior of data: starting from $t = 1$*, the NN*

- *initiates some* $\mathbf{s}[0]$

State-Space Model for *NNs*

We can look at a NN as a state-dependent system



In this architecture, the NN

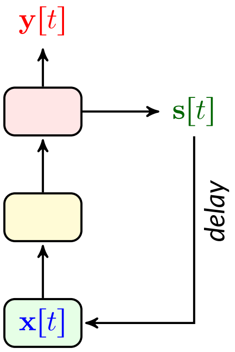
- *takes* $\mathbf{x}[t]$ *and previous state* $\mathbf{s}[t-1]$ *as inputs*
- *returns* $\mathbf{y}[t]$ *and new state* $\mathbf{s}[t]$ *as outputs*

This NN captures temporal behavior of data: starting from $t = 1$*, the NN*

- *initiates some* $\mathbf{s}[0]$
- *computes* $\mathbf{y}[1]$ *and* $\mathbf{s}[1]$ *from time sample* $\mathbf{x}[1]$ *and* $\mathbf{s}[0]$

State-Space Model for *NNs*

We can look at a NN as a state-dependent system



In this architecture, the NN

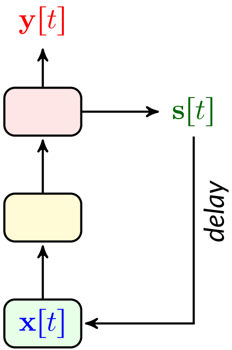
- *takes* $\mathbf{x}[t]$ *and previous state* $\mathbf{s}[t-1]$ *as inputs*
- *returns* $\mathbf{y}[t]$ *and new state* $\mathbf{s}[t]$ *as outputs*

This NN captures temporal behavior of data: starting from $t=1$*, the NN*

- *initiates some* $\mathbf{s}[0]$
- *computes* $\mathbf{y}[1]$ *and* $\mathbf{s}[1]$ *from time sample* $\mathbf{x}[1]$ *and* $\mathbf{s}[0]$
- *computes* $\mathbf{y}[2]$ *and* $\mathbf{s}[2]$ *from* $\mathbf{x}[2]$ *and* $\mathbf{s}[1]$

State-Space Model for *NNs*

We can look at a NN as a state-dependent system



In this architecture, the NN

- *takes* $\mathbf{x}[t]$ *and previous state* $\mathbf{s}[t-1]$ *as inputs*
- *returns* $\mathbf{y}[t]$ *and new state* $\mathbf{s}[t]$ *as outputs*

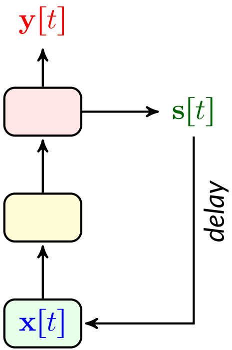
This NN captures temporal behavior of data: starting from $t = 1$*, the NN*

- *initiates some* $\mathbf{s}[0]$
- *computes* $\mathbf{y}[1]$ *and* $\mathbf{s}[1]$ *from time sample* $\mathbf{x}[1]$ *and* $\mathbf{s}[0]$
- *computes* $\mathbf{y}[2]$ *and* $\mathbf{s}[2]$ *from* $\mathbf{x}[2]$ *and* $\mathbf{s}[1]$

  $\ldots$

State-Space Model for *NNs*

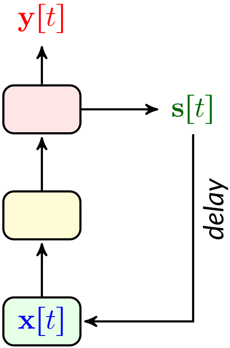
Theoretically, this NN has infinite time response



Say NN initiates with some $\mathbf{s}[0]$. *It takes only sample* $\mathbf{x}[1]$ *and no other time samples is given to it. At time* t,

State-Space Model for *NNs*

Theoretically, this NN has infinite time response

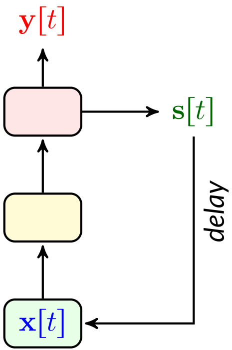


Say NN initiates with some $\mathbf{s}[0]$. *It takes only sample* $\mathbf{x}[1]$ *and no other time samples is given to it. At time* t,

- $\mathbf{y}[t]$ *depends on* $\mathbf{s}[t-1]$

State-Space Model for *NNs*

Theoretically, this NN has infinite time response

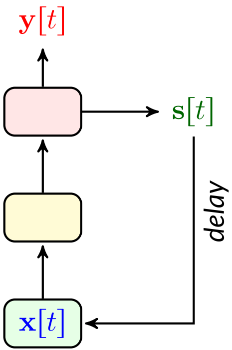


Say NN initiates with some $\mathbf{s}[0]$*. It takes only sample* $\mathbf{x}[1]$ *and no other time samples is given to it. At time* t,

- $\mathbf{y}[t]$ *depends on* $\mathbf{s}[t-1]$
- $\mathbf{s}[t-1]$ *depends on* $\mathbf{s}[t-2]$

State-Space Model for *NNs*

Theoretically, this NN has infinite time response



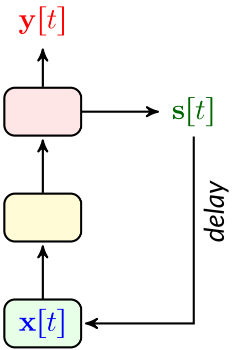
Say NN initiates with some $\mathbf{s}[0]$*. It takes only sample* $\mathbf{x}[1]$ *and no other time samples is given to it. At time* t,

- $\mathbf{y}[t]$ *depends on* $\mathbf{s}[t-1]$
- $\mathbf{s}[t-1]$ *depends on* $\mathbf{s}[t-2]$

$\cdots$

State-Space Model for *NNs*

Theoretically, this NN has infinite time response



Say NN initiates with some $\mathbf{s}[0]$. *It takes only sample* $\mathbf{x}[1]$ *and no other time samples is given to it. At time* t,

- $\mathbf{y}[t]$ *depends on* $\mathbf{s}[t-1]$
- $\mathbf{s}[t-1]$ *depends on* $\mathbf{s}[t-2]$

  $\cdots$

- $\mathbf{s}[1]$ *depends on* $\mathbf{x}[1]$

This means that $\mathbf{y}[t]$ *still remembers* $\mathbf{x}[1]$

State-Dependent NNs

It seems that for our purpose *state-space model helps extracting a good memory components. The challenge is to design a good state-dependent NN*

State-Dependent NNs

It seems that for our purpose *state-space model helps extracting a good memory components. The challenge is to design a good state-dependent NN*

+ *Why is it a challenge? We make an NN with input* $(\mathbf{x}, \mathbf{s})$ *and output* $(\mathbf{y}, \mathbf{s}')$ *and then train it!*
- That sounds easy, but have some challenges

State-Dependent NNs

It seems that for our purpose *state-space model helps extracting a good memory components. The challenge is to design a good state-dependent NN*

+ *Why is it a challenge? We make an NN with input* $(\mathbf{x}, \mathbf{s})$ *and output* $(\mathbf{y}, \mathbf{s}')$ *and then train it!*
– That sounds easy, but have some challenges

Design of state-dependent NNs has two main challenges

1. *Defining the state variable*
   ↳ *how should we specify the state as we do not have a clear clue about it*

State-Dependent NNs

It seems that for our purpose *state-space model helps extracting a good memory components. The challenge is to design a good state-dependent NN*

+ *Why is it a challenge? We make an NN with input* $(\mathbf{x}, \mathbf{s})$ *and output* $(\mathbf{y}, \mathbf{s}')$ *and then train it!*

– That sounds easy, but have some challenges

Design of state-dependent NNs has two main challenges

1. *Defining the state variable*
   ↳ *how should we specify the state as we do not have a clear clue about it*
2. *Training the NN over time: say we get a time sequence data and compute the empirical risk by averaging over time samples up to time* t
   ↳ *empirical risk depends on weights in the NN*
   ↳ *it also depends previous memory components which can be learnable*

State-Dependent NNs

It seems that for our purpose *state-space model helps extracting a good memory components. The challenge is to design a good state-dependent NN*

\+ *Why is it a challenge? We make an NN with input* $(\mathbf{x}, \mathbf{s})$ *and output* $(\mathbf{y}, \mathbf{s}')$ *and then train it!*

– That sounds easy, but have some challenges

Design of state-dependent NNs has two main challenges

1. *Defining the state variable*
   ↳ *how should we specify the state as we do not have a clear clue about it*
2. *Training the NN over time: say we get a time sequence data and compute the empirical risk by averaging over time samples up to time* t
   ↳ *empirical risk depends on weights in the NN*
   ↳ *it also depends previous memory components which can be learnable*
      ↳ *if we want to train the model efficiently, we need to get back over time and update all those memory components!*

State-Dependent NNs

Several attempts have been done: *we look briefly into two of them*

- *Jordan Network*
  - ↳ *proposed by Michael Jordan*[1] *in 1986*
  - ↳ *it computes the state variable to be a simple moving average*

[1]Professor at CS Department of University of Berkeley; check his page

[2]Late Professor at UCSD ('48 – '18); check his page

State-Dependent NNs

Several attempts have been done: *we look briefly into two of them*

- *Jordan Network*
  - ↳ *proposed by Michael Jordan*[1] *in 1986*
  - ↳ *it computes the state variable to be a simple moving average*
- *Elman Network*
  - ↳ *proposed by Jeffrey Elman*[2] *in 1990*
  - ↳ *it learns the state variable but does not track back completely over time*

[1] Professor at CS Department of University of Berkeley; check his page

[2] Late Professor at UCSD ('48 – '18); check his page

State-Dependent NNs

Several attempts have been done: *we look briefly into two of them*

- *Jordan Network*
  - ↳ *proposed by Michael Jordan*[1] *in 1986*
  - ↳ *it computes the state variable to be a simple moving average*
- *Elman Network*
  - ↳ *proposed by Jeffrey Elman*[2] *in 1990*
  - ↳ *it learns the state variable but does not track back completely over time*

These models are simple forms of what we nowadays know as

$$\textit{Recurrent NNs} \equiv \textit{RNNs}$$

[1]Professor at CS Department of University of Berkeley; check his page

[2]Late Professor at UCSD ('48 – '18); check his page

State-Dependent NNs

Several attempts have been done: *we look briefly into two of them*

- *Jordan Network*
  - ↳ *proposed by Michael Jordan*[1] *in 1986*
  - ↳ *it computes the state variable to be a simple moving average*
- *Elman Network*
  - ↳ *proposed by Jeffrey Elman*[2] *in 1990*
  - ↳ *it learns the state variable but does not track back completely over time*

These models are simple forms of what we nowadays know as

Recurrent NNs $\equiv$ *RNNs*

RNN

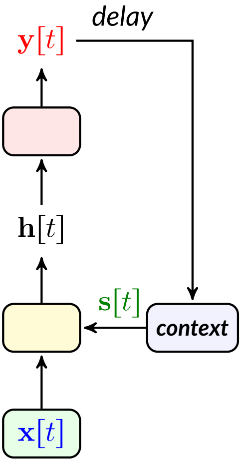
RNN is an NN with state-variable: the term recurrent refers to the connection between former state and new output

[1] Professor at CS Department of University of Berkeley; check his page

[2] Late Professor at UCSD ('48 – '18); check his page

Jordan Model

Jordan Network uses a simple moving average of output as the state

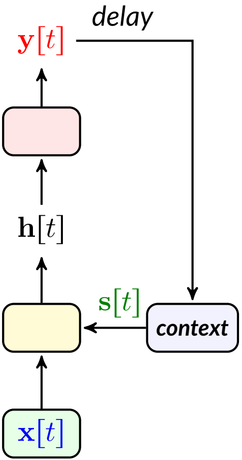


The proposal was a shallow NN

- *It starts with some initial state* $\mathbf{s}[1]$

Jordan Model

Jordan Network uses a simple moving average of output as the state



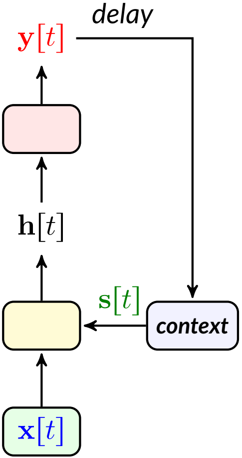
The proposal was a shallow NN

- *It starts with some initial state* $\mathbf{s}[1]$
- *In time* t*, it updates the state* $\mathbf{s}[t]$ *with fixed* μ *as*

$$\mathbf{s}[t] = \mu \mathbf{s}[t-1] + \mathbf{y}[t-1]$$

Jordan Model

Jordan Network uses a simple moving average of output as the state



The proposal was a shallow NN

- *It starts with some initial state $\mathbf{s}[1]$*
- *In time t, it updates the state $\mathbf{s}[t]$ with fixed μ as*

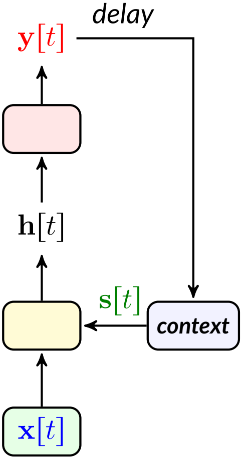
$$\mathbf{s}[t] = \mu \mathbf{s}[t-1] + \mathbf{y}[t-1]$$

- *The hidden layer then computes*

$$\mathbf{h}[t] = f\left(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_{\mathrm{m}} \mathbf{s}[t]\right)$$

Jordan Model

Jordan Network uses a simple moving average of output as the state



The proposal was a shallow NN

- *It starts with some initial state* $\mathbf{s}[1]$
- *In time* t, *it updates the state* $\mathbf{s}[t]$ *with fixed* μ *as*

$$\mathbf{s}[t] = \mu\mathbf{s}[t-1] + \mathbf{y}[t-1]$$

- *The hidden layer then computes*

$$\mathbf{h}[t] = f\left(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\mathrm{m}}\mathbf{s}[t]\right)$$

- *The output is*

$$\mathbf{y}[t] = f\left(\mathbf{W}_2\mathbf{h}[t]\right)$$

Jordan Network

Jordan Network has a memory component with infinte response: *say we set initial state to zero, give* $\mathbf{x}[1]$*, and keep the input zero for the rest of time; then,*

$$\mathbf{y}[1] = f\left(\mathbf{W}_2 f\left(\mathbf{W}_1 \mathbf{x}[1]\right)\right)$$
$$\mathbf{y}[2] = f\left(\mathbf{W}_2 f\left(\mathbf{W}_{\mathrm{m}} \mathbf{y}[1]\right)\right) = f\left(\mathbf{W}_2 f\left(\mathbf{W}_{\mathrm{m}} f\left(\mathbf{W}_2 f\left(\mathbf{W}_1 \mathbf{x}[1]\right)\right)\right)\right)$$
$$\vdots$$

Jordan Network

Jordan Network has a memory component with infinte response: *say we set initial state to zero*, *give* $\mathbf{x}[1]$, *and keep the input zero for the rest of time; then,*

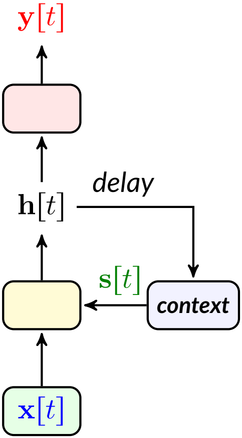
$$
\begin{aligned}
\mathbf{y}[1] &= f\left(\mathbf{W}_2 f\left(\mathbf{W}_1 \mathbf{x}[1]\right)\right) \\
\mathbf{y}[2] &= f\left(\mathbf{W}_2 f\left(\mathbf{W}_{\mathrm{m}} \mathbf{y}[1]\right)\right) = f\left(\mathbf{W}_2 f\left(\mathbf{W}_{\mathrm{m}} f\left(\mathbf{W}_2 f\left(\mathbf{W}_1 \mathbf{x}[1]\right)\right)\right)\right) \\
&\vdots
\end{aligned}
$$

But, the network *does not learn how to remember*

- *It takes a fixed moving average as memory component*
- *It only learns how to use this pre-defined memory for prediction*
  - ↳ *We could say that it implicitly learn to remember by learning* $\mathbf{W}_{\mathrm{m}}$

Elman Network

Elman Network uses output of hidden layer as state: also called hidden state

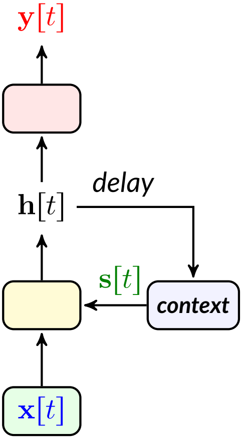


The proposal was a shallow NN

- *It starts with some initial hidden state* $\mathbf{h}[0]$

Elman Network

Elman Network uses output of hidden layer as state: also called hidden state



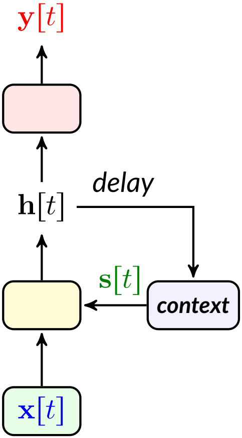
The proposal was a shallow NN

- *It starts with some initial hidden state* $\mathbf{h}[0]$
- *In time* t, *it updates the state* $\mathbf{s}[t]$ *as*

$$\mathbf{s}[t] = \mathbf{h}[t-1]$$

Elman Network

Elman Network uses output of hidden layer as state: also called hidden state



The proposal was a shallow NN

- *It starts with some initial hidden state* $\mathbf{h}[0]$
- *In time* t, *it updates the state* $\mathbf{s}[t]$ *as*

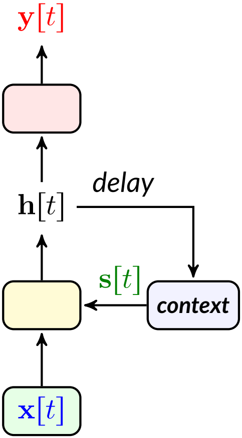
$$\mathbf{s}[t] = \mathbf{h}[t-1]$$

- *The hidden layer then computes*

$$\mathbf{h}[t] = f\left(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_{\mathrm{m}} \mathbf{s}[t]\right)$$

Elman Network

Elman Network uses output of hidden layer as state: also called hidden state



The proposal was a shallow NN

- *It starts with some initial hidden state* $\mathbf{h}[0]$
- *In time t, it updates the state* $\mathbf{s}[t]$ *as*

$$\mathbf{s}[t] = \mathbf{h}[t-1]$$

- *The hidden layer then computes*

$$\mathbf{h}[t] = f\left(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\rm m}\mathbf{s}[t]\right)$$

- *The output is*

$$\mathbf{y}[t] = f\left(\mathbf{W}_2\mathbf{h}[t]\right)$$

Elman Network

Similar to Jordan Network, Elman Network has a memory component with infinte response: *with zero initial hidden state, we have*

$$
\begin{aligned}
\mathbf{y}[1] &= f\left(\mathbf{W}_2 f\left(\mathbf{W}_1 \mathbf{x}[1]\right)\right)\\
\mathbf{y}[2] &= f\left(\mathbf{W}_2 f\left(\mathbf{W}_\mathrm{m} \mathbf{h}[1]\right)\right) = f\left(\mathbf{W}_2 f\left(\mathbf{W}_\mathrm{m} f\left(\mathbf{W}_1 \mathbf{x}[1]\right)\right)\right)\\
&\vdots
\end{aligned}
$$

Elman network also learns how to remember only implicitly

Challenge of Learning Through Time

Though Jordan and Elman Networks had memory, *they did not get trained accurately over time, i.e., they simplified the solution to the second challenge*

+ *What is really this challenge?*
– We are going to deal with it in next sections, but let's see it on these simple networks first

[3] We will see that this is not always this easy!

Challenge of Learning Through Time

Though Jordan and Elman Networks had memory, *they did not get trained accurately over time, i.e., they simplified the solution to the second challenge*

+ *What is really this challenge?*
- We are going to deal with it in next sections, but let's see it on these simple networks first

---

Let's assume a simple setting: *we are to train our NN on single data sequence*

- *We have the sequence* $\mathbf{x}[1], \ldots, \mathbf{x}[T]$ *as the data-point*

[3] We will see that this is not always this easy!

Challenge of Learning Through Time

Though Jordan and Elman Networks had memory, *they did not get trained accurately over time, i.e., they simplified the solution to the second challenge*

+ *What is really this challenge?*
– We are going to deal with it in next sections, but let's see it on these simple networks first

---

Let's assume a simple setting: *we are to train our NN on single data sequence*

- *We have the sequence* $\mathbf{x}[1], \ldots, \mathbf{x}[T]$ *as the data-point*
- *For each entry of this sequence we have the true label*
  - ↳ *We have sequence* $\mathbf{v}[1], \ldots, \mathbf{v}[T]$ *with* $\mathbf{v}[t]$ *being label of* $\mathbf{x}[t]$

[3] We will see that this is not always this easy!

Challenge of Learning Through Time

Though Jordan and Elman Networks had memory, *they did not get trained accurately over time, i.e., they simplified the solution to the second challenge*

+ *What is really this challenge?*
– We are going to deal with it in next sections, but let's see it on these simple networks first

---

Let's assume a simple setting: *we are to train our NN on single data sequence*

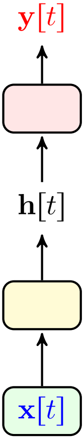
- *We have the sequence* $\mathbf{x}[1], \ldots, \mathbf{x}[T]$ *as the data-point*
- *For each entry of this sequence we have the true label*
  - ↳ *We have sequence* $\mathbf{v}[1], \ldots, \mathbf{v}[T]$ *with* $\mathbf{v}[t]$ *being label of* $\mathbf{x}[t]$
- *We are able to compute the loss between outputs and true labels as*[3]

$$\hat{R} = \sum_{t=1}^{T} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right)$$

[3] We will see that this is not always this easy!

Recap: *Basic FNN*

For sake if comparison, let's first train a basic FNN on this data sequence



We have a shallow FNN

- *The hidden layer computes*

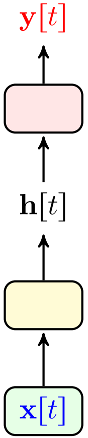
$$\mathbf{h}[t] = f\left(\mathbf{W}_1 \mathbf{x}[t]\right)$$

- *The output is*

$$\mathbf{y}[t] = f\left(\mathbf{W}_2 \mathbf{h}[t]\right)$$

Recap: *Basic FNN*

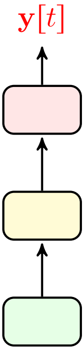
For sake if comparison, let's first train a basic FNN on this data sequence



How do we train this FNN?

- *We compute the gradients* $\nabla_{\mathbf{W}_1} \hat{R}$ *and* $\nabla_{\mathbf{W}_2} \hat{R}$
  ↳ *We do it via backpropagation*
- *We apply gradient descent*

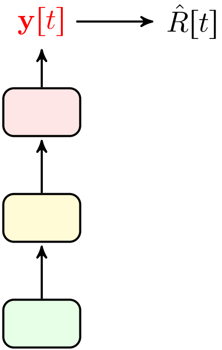
Learning Through Time: *FNNs*



$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\cdots$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\cdots$ $\mathbf{x}[T-1]$ $\mathbf{x}[T]$

$$\hat{R} = \sum_{t=1}^{T} \underbrace{\mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right)}_{\hat{R}[t]}$$

Learning Through Time: *FNNs*



$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\cdots$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\cdots$ $\mathbf{x}[T-1]$ $\mathbf{x}[T]$

$$\hat{R} = \sum_{t=1}^{T} \underbrace{\mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right)}_{\hat{R}[t]} = \sum_{t=1}^{T} \hat{R}[t] \rightsquigarrow \nabla_{\mathbf{w}_i} \hat{R} = \sum_{t=1}^{T} \nabla_{\mathbf{w}_i} \hat{R}[t]$$

Learning Through Time: *FNNs*

Let's learn $\mathbf{W}_1$ and to ease computation we use our cheating notation, i.e., *use $\circ$ to show any product:*

Learning Through Time: *FNNs*

Let's learn $\mathbf{W}_1$ and to ease computation we use our cheating notation, i.e., *use* $\circ$ *to show any product: to compute the gradient we start with the output*

$$
\begin{aligned}
\nabla_{\mathbf{W}_1} \hat{R}[t] &= \nabla_{\mathbf{W}_1} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right) \\
&= \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]
\end{aligned}
$$

Learning Through Time: *FNNs*

Let's learn $\mathbf{W}_1$ and to ease computation we use our cheating notation, i.e., *use $\circ$ to show any product: to compute the gradient we start with the output*

$$\begin{aligned}\nabla_{\mathbf{W}_1} \hat{R}[t] &= \nabla_{\mathbf{W}_1} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right) \\ &= \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]\end{aligned}$$

We know that $\mathbf{y}[t] = f\left(\mathbf{W}_2 \mathbf{h}[t]\right)$, so we can write

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{y}[t] &= \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \\ &= \left(\dot{f}\left(\mathbf{W}_2 \mathbf{h}[t]\right) \circ \mathbf{W}_2\right) \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]\end{aligned}$$

Learning Through Time: *FNNs*

Let's learn $\mathbf{W}_1$ and to ease computation we use our cheating notation, i.e., *use $\circ$ to show any product: to compute the gradient we start with the output*

$$\begin{aligned}\nabla_{\mathbf{W}_1} \hat{R}[t] &= \nabla_{\mathbf{W}_1} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right) \\ &= \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]\end{aligned}$$

We know that $\mathbf{y}[t] = f\left(\mathbf{W}_2 \mathbf{h}[t]\right)$, so we can write

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{y}[t] &= \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \\ &= \left(\dot{f}\left(\mathbf{W}_2 \mathbf{h}[t]\right) \circ \mathbf{W}_2\right) \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]\end{aligned}$$

What about $\nabla_{\mathbf{W}_1} \mathbf{h}[t]$? We keep on backward!

Learning Through Time: *FNNs*

Up to now, we have

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]$$

Learning Through Time: *FNNs*

Up to now, we have

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]$$

We use the fact that $\mathbf{h}[t] = f\left(\mathbf{W}_1 \mathbf{x}[t]\right)$

$$\nabla_{\mathbf{W}_1} \mathbf{h}[t] = \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t]\right) + \nabla_{\mathbf{x}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{x}[t]$$

Learning Through Time: *FNNs*

Up to now, we have

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]$$

We use the fact that $\mathbf{h}[t] = f\left(\mathbf{W}_1 \mathbf{x}[t]\right)$

$$\begin{aligned} \nabla_{\mathbf{W}_1} \mathbf{h}[t] &= \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t]\right) + \nabla_{\mathbf{x}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{x}[t] \\ &= \dot{f}\left(\mathbf{W}_1 \mathbf{x}[t]\right) \circ \mathbf{x}[t] + \left(\dot{f}\left(\mathbf{W}_1 \mathbf{x}[t]\right) \circ \mathbf{W}_1\right) \circ \underbrace{\mathbf{0}}_{\mathbf{x}[t] \text{ is not a function of } \mathbf{W}_1} \end{aligned}$$

Learning Through Time: *FNNs*

Up to now, we have

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]$$

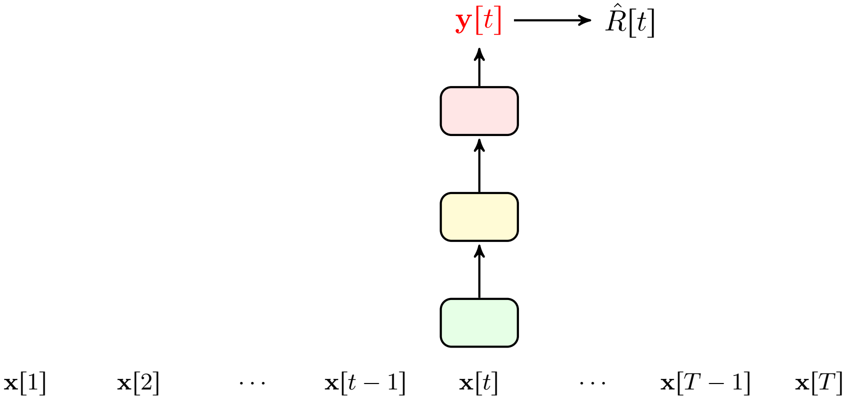
We use the fact that $\mathbf{h}[t] = f(\mathbf{W}_1 \mathbf{x}[t])$

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{h}[t] &= \nabla_{\mathbf{W}_1} f(\mathbf{W}_1 \mathbf{x}[t]) + \nabla_{\mathbf{x}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{x}[t] \\ &= \dot{f}(\mathbf{W}_1 \mathbf{x}[t]) \circ \mathbf{x}[t] + \left(\dot{f}(\mathbf{W}_1 \mathbf{x}[t]) \circ \mathbf{W}_1\right) \circ \underbrace{\mathbf{0}}_{\mathbf{x}[t] \text{ is not a function of } \mathbf{W}_1}\end{aligned}$$

Therefore, we end chain rule here

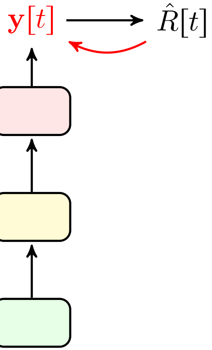
$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]$$

Learning Through Time: *FNNs*



$$\nabla_{\mathbf{W}_1} \hat{R}[t] =$$

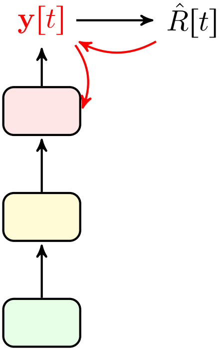
Learning Through Time: *FNNs*



$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\cdots$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\cdots$ $\mathbf{x}[T-1]$ $\mathbf{x}[T]$

$$\nabla_{\mathbf{w}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t]$$

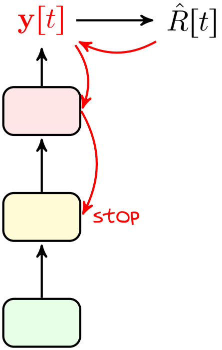
Learning Through Time: *FNNs*



$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\cdots$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\cdots$ $\mathbf{x}[T-1]$ $\mathbf{x}[T]$

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{h}[t]} \mathbf{y}[t]$$

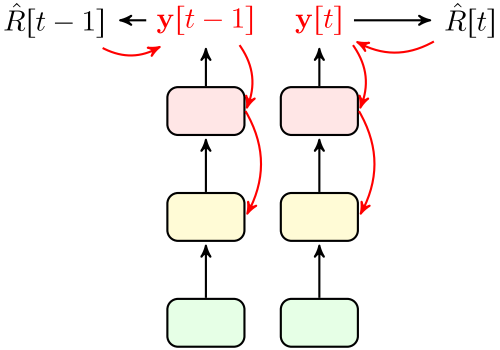
Learning Through Time: *FNNs*



$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\cdots$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\cdots$ $\mathbf{x}[T-1]$ $\mathbf{x}[T]$

$$\nabla_{\mathbf{w}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{w}_1} \mathbf{h}[t]$$

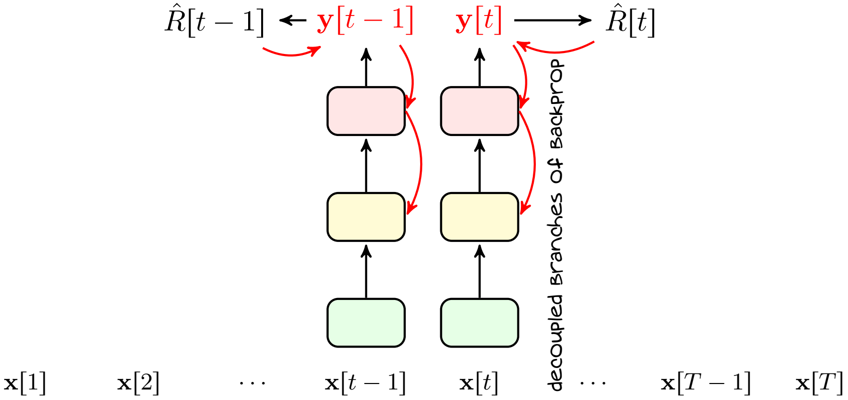
Learning Through Time: *FNNs*



$\mathbf{x}[1]$ $\mathbf{x}[2]$ $\cdots$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$ $\cdots$ $\mathbf{x}[T-1]$ $\mathbf{x}[T]$

$$\nabla_{\mathbf{w}_2} \hat{R} = \sum_{t=1}^{T} \nabla_{\mathbf{w}_2} \hat{R}[t]$$

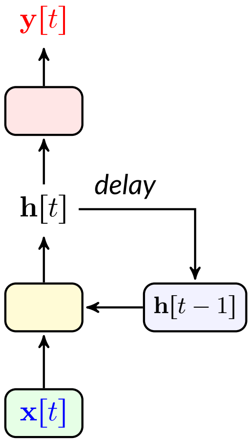
Learning Through Time: *FNNs*



$$\nabla_{\mathbf{w}_2} \hat{R} = \sum_{t=1}^{T} \nabla_{\mathbf{w}_2} \hat{R}[t]$$

Training a *Basic RNN*

Now, let's train Elman network on this sequence



The proposal was a shallow NN

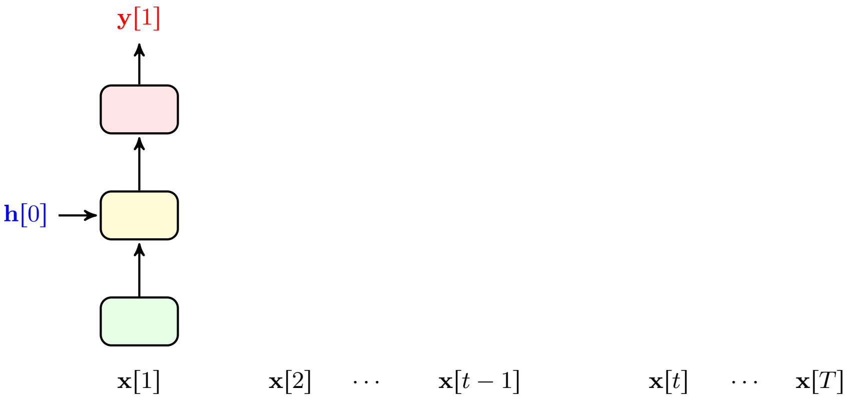
- *It starts with some initial hidden state* $\mathbf{h}[0]$
- *The hidden layer then computes*

$$\mathbf{h}[t] = f\left(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-1]\right)$$

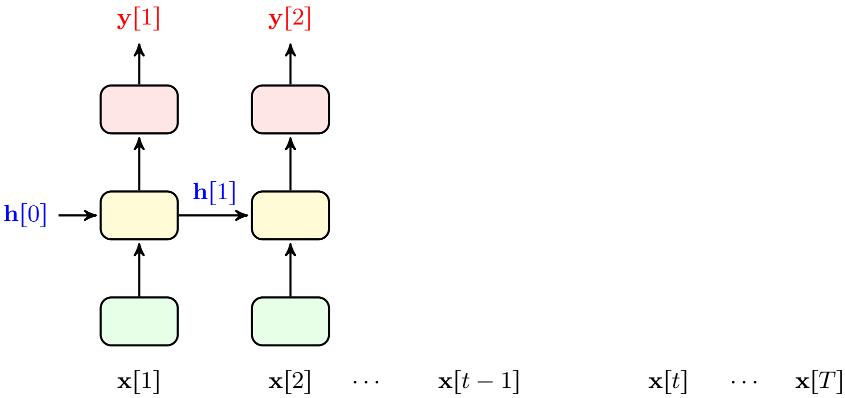
- *The output is*

$$\mathbf{y}[t] = f\left(\mathbf{W}_2 \mathbf{h}[t]\right)$$

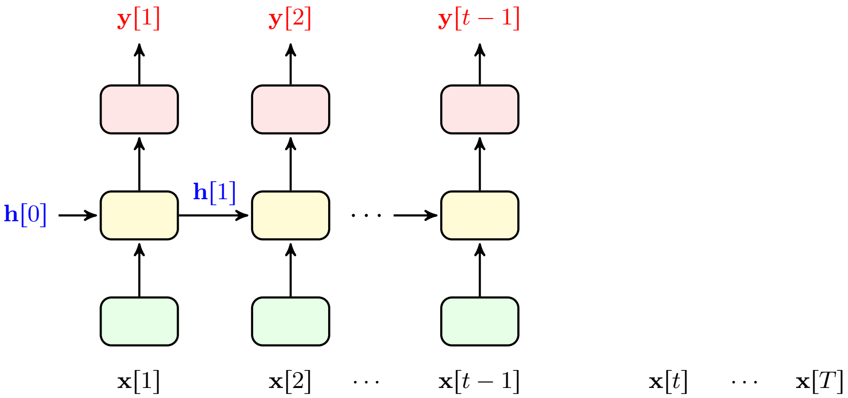
Inferring Through Time: *Elman Network*



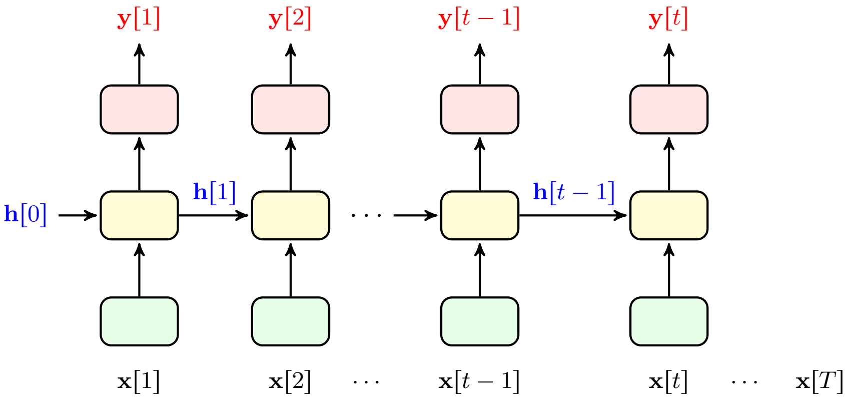
Inferring Through Time: *Elman Network*



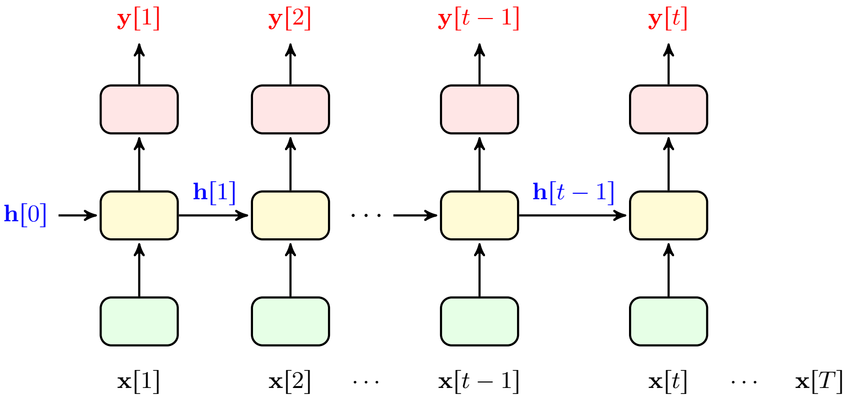
Inferring Through Time: *Elman Network*



Inferring Through Time: *Elman Network*

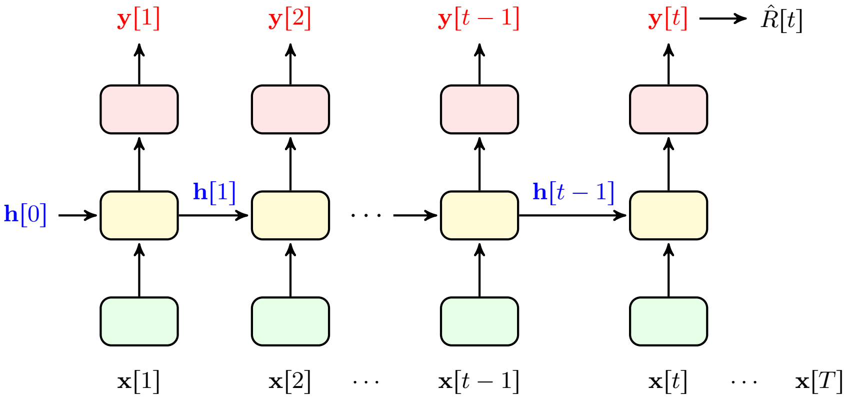


Inferring Through Time: *Elman Network*



$$\hat{R} = \sum_{t=1}^{T} \underbrace{\mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right)}_{\hat{R}[t]} = \sum_{t=1}^{T} \hat{R}[t]$$

Inferring Through Time: *Elman Network*



$$\hat{R} = \sum_{t=1}^{T} \underbrace{\mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right)}_{\hat{R}[t]} = \sum_{t=1}^{T} \hat{R}[t]$$

Learning Through Time: *Elman Network*

Let's again try to learn $\mathbf{W}_1$: *we start with the output*

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]$$

Learning Through Time: *Elman Network*

Let's again try to learn $\mathbf{W}_1$: *we start with the output*

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]$$

We know that $\mathbf{y}[t] = f\left(\mathbf{W}_2 \mathbf{h}[t]\right)$, *so we can write*

$$\nabla_{\mathbf{W}_1} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]$$

Learning Through Time: *Elman Network*

Let's again try to learn $\mathbf{W}_1$: *we start with the output*

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]$$

We know that $\mathbf{y}[t] = f(\mathbf{W}_2 \mathbf{h}[t])$, *so we can write*

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{y}[t] &= \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \\ &= \dot{f}(\mathbf{W}_2 \mathbf{h}[t]) \circ \mathbf{W}_2 \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]\end{aligned}$$

Learning Through Time: *Elman Network*

Let's again try to learn $\mathbf{W}_1$: *we start with the output*

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]$$

We know that $\mathbf{y}[t] = f(\mathbf{W}_2 \mathbf{h}[t])$, *so we can write*

$$\begin{aligned} \nabla_{\mathbf{W}_1} \mathbf{y}[t] &= \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \\ &= \dot{f}(\mathbf{W}_2 \mathbf{h}[t]) \circ \mathbf{W}_2 \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \end{aligned}$$

Next, we note that $\mathbf{h}[t] = f(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_\mathrm{m} \mathbf{h}[t-1])$

$$\nabla_{\mathbf{W}_1} \mathbf{h}[t] =$$

Learning Through Time: *Elman Network*

Let's again try to learn $\mathbf{W}_1$: *we start with the output*

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]$$

We know that $\mathbf{y}[t] = f\left(\mathbf{W}_2 \mathbf{h}[t]\right)$, *so we can write*

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{y}[t] &= \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \\ &= \dot{f}\left(\mathbf{W}_2 \mathbf{h}[t]\right) \circ \mathbf{W}_2 \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]\end{aligned}$$

Next, we note that $\mathbf{h}[t] = f\left(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-1]\right)$

$$\nabla_{\mathbf{W}_1} \mathbf{h}[t] = \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-1]\right)$$

Learning Through Time: *Elman Network*

Let's again try to learn $\mathbf{W}_1$: *we start with the output*

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]$$

We know that $\mathbf{y}[t] = f(\mathbf{W}_2 \mathbf{h}[t])$, *so we can write*

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{y}[t] &= \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \\ &= \dot{f}(\mathbf{W}_2 \mathbf{h}[t]) \circ \mathbf{W}_2 \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t]\end{aligned}$$

Next, we note that $\mathbf{h}[t] = f(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-1])$

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{h}[t] = \nabla_{\mathbf{W}_1} & f(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-1]) \\ &+ \nabla_{\mathbf{x}[t]} \mathbf{h}[t] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{x}[t]}_{\mathbf{0}}\end{aligned}$$

Learning Through Time: *Elman Network*

Let's again try to learn $\mathbf{W}_1$: *we start with the output*

$$\nabla_{\mathbf{W}_1} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \hat{R}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{y}[t]$$

We know that $\mathbf{y}[t] = f(\mathbf{W}_2 \mathbf{h}[t])$, *so we can write*

$$\begin{aligned} \nabla_{\mathbf{W}_1} \mathbf{y}[t] &= \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \\ &= \dot{f}(\mathbf{W}_2 \mathbf{h}[t]) \circ \mathbf{W}_2 \circ \nabla_{\mathbf{W}_1} \mathbf{h}[t] \end{aligned}$$

Next, we note that $\mathbf{h}[t] = f(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_\mathrm{m} \mathbf{h}[t-1])$

$$\begin{aligned} \nabla_{\mathbf{W}_1} \mathbf{h}[t] = \nabla_{\mathbf{W}_1} f(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_\mathrm{m} \mathbf{h}[t-1]) \\ + \nabla_{\mathbf{x}[t]} \mathbf{h}[t] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{x}[t]}_{\mathbf{0}} \\ + \nabla_{\mathbf{h}[t-1]} \mathbf{h}[t] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{h}[t-1]}_{?} \end{aligned}$$

Learning Through Time: *Elman Network*

Well! We know that $\mathbf{h}[t-1] = f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-2]\right)$

$$\nabla_{\mathbf{W}_1} \mathbf{h}[t-1] =$$

Learning Through Time: *Elman Network*

Well! We know that $\mathbf{h}[t-1] = f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-2]\right)$

$$\nabla_{\mathbf{W}_1} \mathbf{h}[t-1] = \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-2]\right)$$

Learning Through Time: *Elman Network*

Well! We know that $\mathbf{h}[t-1] = f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-2]\right)$

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{h}[t-1] = \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-2]\right) \\ + \nabla_{\mathbf{x}[t-1]} \mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{x}[t-1]}_{\mathbf{0}}\end{aligned}$$

Learning Through Time: *Elman Network*

Well! We know that $\mathbf{h}[t-1] = f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_\mathrm{m} \mathbf{h}[t-2]\right)$

$$
\begin{aligned}
\nabla_{\mathbf{W}_1} \mathbf{h}[t-1] = \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_\mathrm{m} \mathbf{h}[t-2]\right) \\
+ \nabla_{\mathbf{x}[t-1]} \mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{x}[t-1]}_{\mathbf{0}} \\
+ \nabla_{\mathbf{h}[t-2]} \mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{h}[t-2]}_{?}
\end{aligned}
$$

Learning Through Time: *Elman Network*

Well! We know that $\mathbf{h}[t-1] = f\left(\mathbf{W}_1\mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}}\mathbf{h}[t-2]\right)$

$$\begin{aligned}\nabla_{\mathbf{W}_1}\mathbf{h}[t-1] = \nabla_{\mathbf{W}_1} & f\left(\mathbf{W}_1\mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}}\mathbf{h}[t-2]\right)\\ & + \nabla_{\mathbf{x}[t-1]}\mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1}\mathbf{x}[t-1]}_{\mathbf{0}}\\ & + \nabla_{\mathbf{h}[t-2]}\mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1}\mathbf{h}[t-2]}_{?}\end{aligned}$$

We are not still done! We know $\mathbf{h}[t-2] = f\left(\mathbf{W}_1\mathbf{x}[t-2] + \mathbf{W}_{\mathrm{m}}\mathbf{h}[t-3]\right)$

$$\nabla_{\mathbf{W}_1}\mathbf{h}[t-2] =$$

Learning Through Time: *Elman Network*

Well! We know that $\mathbf{h}[t-1] = f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-2]\right)$

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{h}[t-1] = & \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-2]\right) \\ & + \nabla_{\mathbf{x}[t-1]} \mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{x}[t-1]}_{\mathbf{0}} \\ & + \nabla_{\mathbf{h}[t-2]} \mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{h}[t-2]}_{?}\end{aligned}$$

We are not still done! We know $\mathbf{h}[t-2] = f\left(\mathbf{W}_1 \mathbf{x}[t-2] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-3]\right)$

$$\nabla_{\mathbf{W}_1} \mathbf{h}[t-2] = \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t-2] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-3]\right)$$

Learning Through Time: *Elman Network*

Well! We know that $\mathbf{h}[t-1] = f\left(\mathbf{W}_1\mathbf{x}[t-1] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-2]\right)$

$$\begin{aligned}\nabla_{\mathbf{W}_1}\mathbf{h}[t-1] = &\nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1\mathbf{x}[t-1] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-2]\right)\\ &+ \nabla_{\mathbf{x}[t-1]}\mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1}\mathbf{x}[t-1]}_{\mathbf{0}}\\ &+ \nabla_{\mathbf{h}[t-2]}\mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1}\mathbf{h}[t-2]}_{?}\end{aligned}$$

We are not still done! *We know* $\mathbf{h}[t-2] = f\left(\mathbf{W}_1\mathbf{x}[t-2] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-3]\right)$

$$\begin{aligned}\nabla_{\mathbf{W}_1}\mathbf{h}[t-2] = &\nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1\mathbf{x}[t-2] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-3]\right)\\ &+ \nabla_{\mathbf{x}[t-2]}\mathbf{h}[t-2] \circ \underbrace{\nabla_{\mathbf{W}_1}\mathbf{x}[t-2]}_{\mathbf{0}}\end{aligned}$$

Learning Through Time: *Elman Network*

Well! We know that $\mathbf{h}[t-1] = f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\rm m} \mathbf{h}[t-2]\right)$

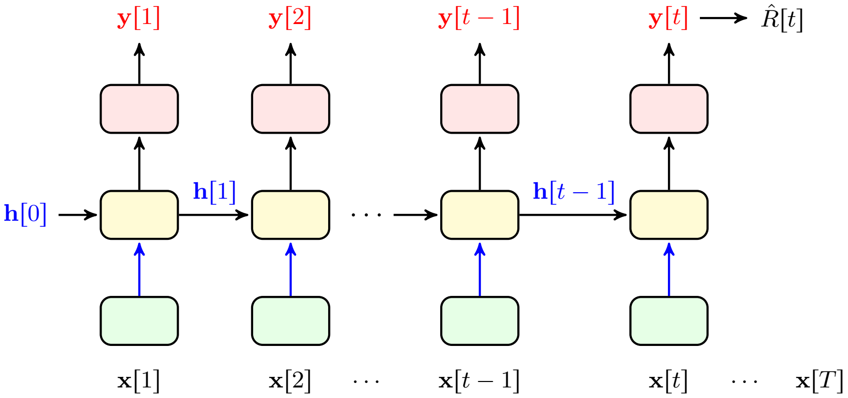
$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{h}[t-1] = \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t-1] + \mathbf{W}_{\rm m} \mathbf{h}[t-2]\right) \\ + \nabla_{\mathbf{x}[t-1]} \mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{x}[t-1]}_{\mathbf{0}} \\ + \nabla_{\mathbf{h}[t-2]} \mathbf{h}[t-1] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{h}[t-2]}_{?}\end{aligned}$$

We are not still done! We know $\mathbf{h}[t-2] = f\left(\mathbf{W}_1 \mathbf{x}[t-2] + \mathbf{W}_{\rm m} \mathbf{h}[t-3]\right)$

$$\begin{aligned}\nabla_{\mathbf{W}_1} \mathbf{h}[t-2] = \nabla_{\mathbf{W}_1} f\left(\mathbf{W}_1 \mathbf{x}[t-2] + \mathbf{W}_{\rm m} \mathbf{h}[t-3]\right) \\ + \nabla_{\mathbf{x}[t-2]} \mathbf{h}[t-2] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{x}[t-2]}_{\mathbf{0}} \\ + \nabla_{\mathbf{h}[t-3]} \mathbf{h}[t-2] \circ \underbrace{\nabla_{\mathbf{W}_1} \mathbf{h}[t-3]}_{?}\end{aligned}$$

Learning Through Time: *Elman Network*

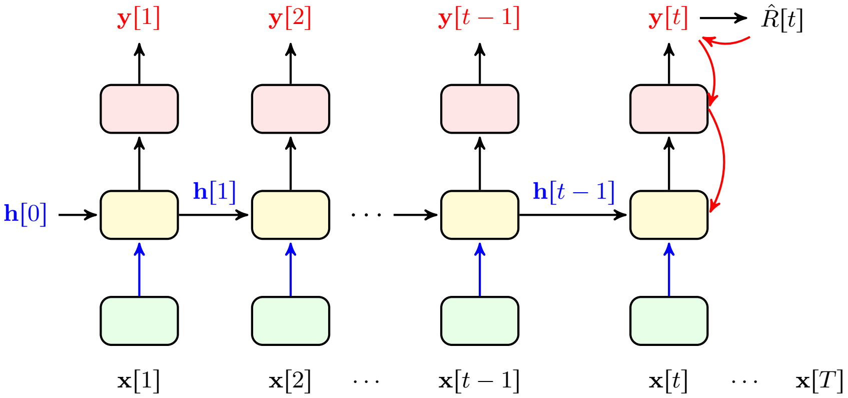
We should in fact pass all the way back to the initial time interval time!



Note that all blue edges are representing $\mathbf{W}_1$

Learning Through Time: *Elman Network*

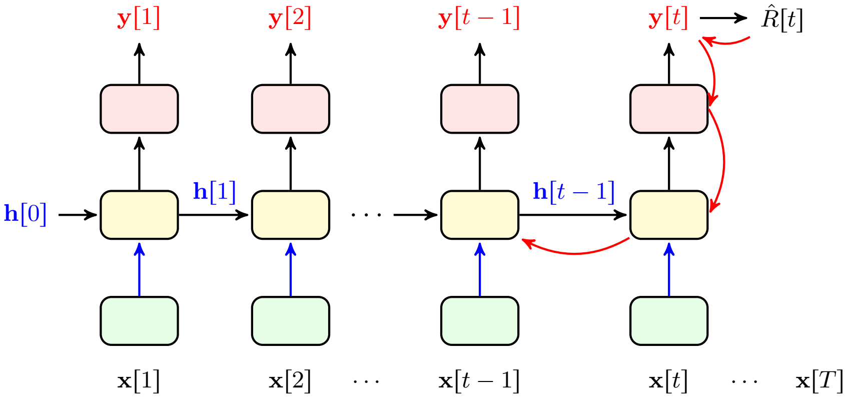
We should in fact pass all the way back to the initial time interval time!



Note that all blue edges are representing $\mathbf{W}_1$

Learning Through Time: *Elman Network*

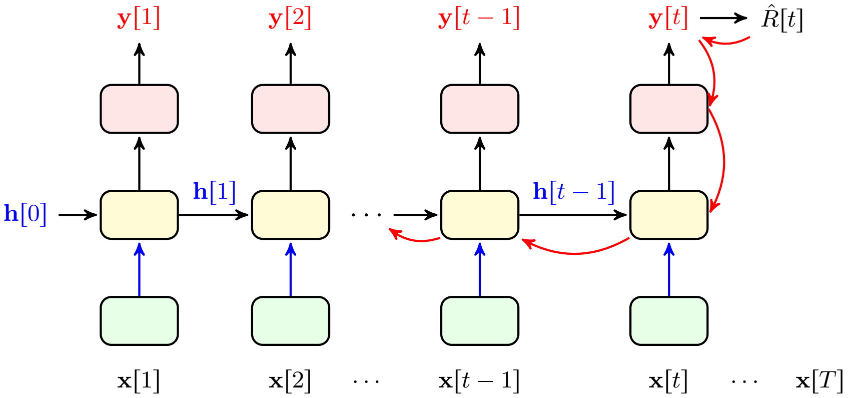
We should in fact pass all the way back to the initial time interval time!



Note that all blue edges are representing $\mathbf{W}_1$

Learning Through Time: *Elman Network*

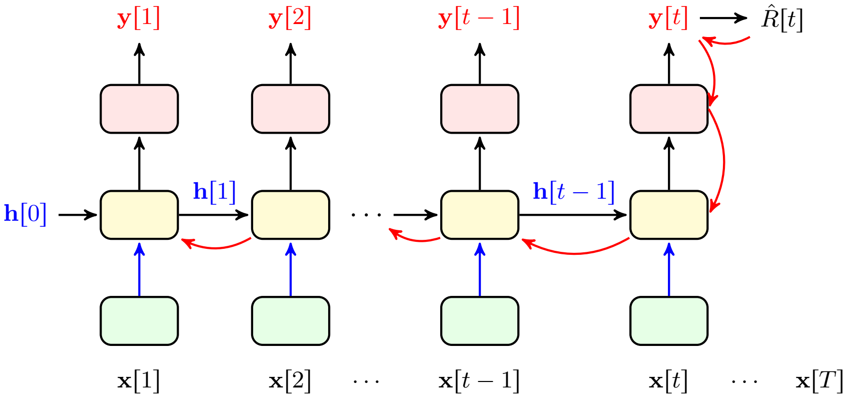
We should in fact pass all the way back to the initial time interval time!



Note that all blue edges are representing $\mathbf{W}_1$

Learning Through Time: *Elman Network*

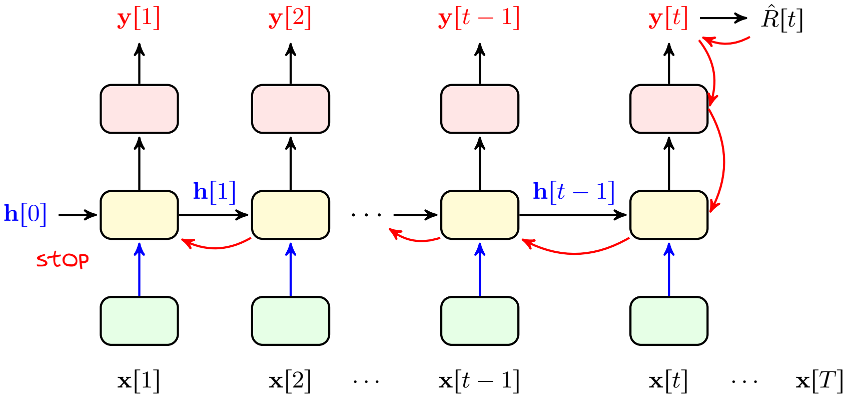
We should in fact pass all the way back to the initial time interval time!



Note that all blue edges are representing $\mathbf{W}_1$

Learning Through Time: *Elman Network*

We should in fact pass all the way back to the initial time interval time!



Note that all blue edges are representing $\mathbf{W}_1$

Learning Through Time

Moral of Story

To learn how to remember, we need to train our RNN through time: at each time interval, we should move all the way back to origin to find out how exactly we should change the weights!

Learning Through Time

Moral of Story

To learn how to remember, we need to train our RNN through time: at each time interval, we should move all the way back to origin to find out how exactly we should change the weights!

+ *Did Elman did so?*
– Not really!

Learning Through Time

Moral of Story

To learn how to remember, we need to train our RNN through time: at each time interval, we should move all the way back to origin to find out how exactly we should change the weights!

+ *Did Elman did so?*
– Not really!

For training Elman treated the hidden state $\mathbf{h}[t-1]$ *as a fixed variable, i.e., he assumed* $\nabla_{\mathbf{W}_1}\mathbf{h}[t-1] \approx \nabla_{\mathbf{W}_\mathrm{m}}\mathbf{h}[t-1] = \mathbf{0}$*! So, he did not need to move backward in time!*

Learning Through Time

Moral of Story

To learn how to remember, we need to train our RNN through time: at each time interval, we should move all the way back to origin to find out how exactly we should change the weights!

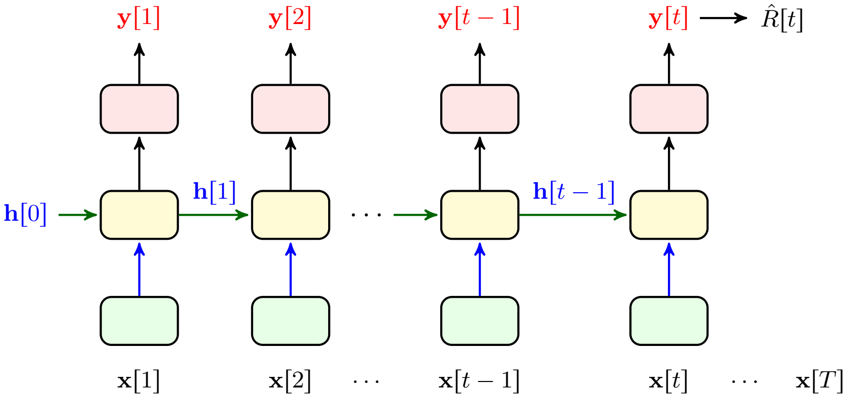
+ *Did Elman did so?*
– Not really!

For training Elman treated the hidden state $\mathbf{h}[t-1]$ as a fixed variable, i.e., he assumed $\nabla_{\mathbf{W}_1}\mathbf{h}[t-1] \approx \nabla_{\mathbf{W}_m}\mathbf{h}[t-1] = \mathbf{0}$! So, he did not need to move backward in time!

This means that Elman did not really addressed the second challenge!

Learning Through Time: *Elman's Approximation*

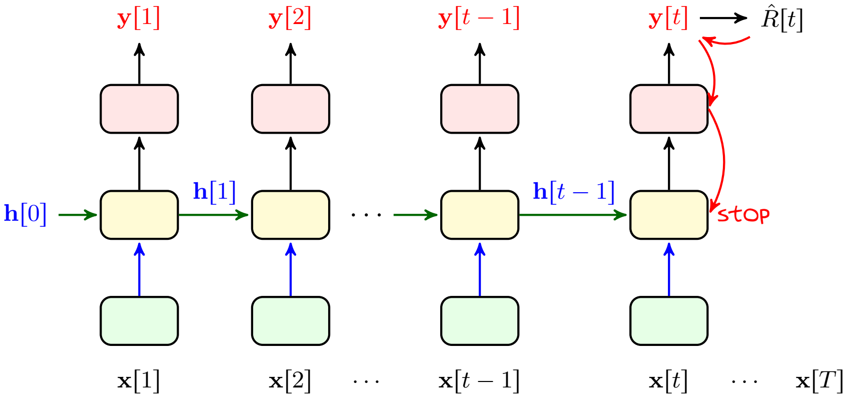
Elman treated it as a simple FNN with only one extra input!



Training $\mathbf{W}_1$ *is exactly as FNN. We just have one extra* $\mathbf{W}_\mathrm{m}$ *here!*

Learning Through Time: *Elman's Approximation*

Elman treated it as a simple FNN with only one extra input!



Training $\mathbf{W}_1$ *is exactly as FNN. We just have one extra* $\mathbf{W}_\mathrm{m}$ *here!*

RNNs: *Need to Learn Memory*

Though appreciated, Elman and Jordan Networks did not do the job

1. *Their memory component is rather simple*
   ↳ *We should use deeper models that enable advanced memory components*
2. *They do not really learn how to remember*
   ↳ *We should train the memory component over time*

RNNs: *Need to Learn Memory*

Though appreciated, Elman and Jordan Networks did not do the job

1. *Their memory component is rather simple*
   ↳ *We should use deeper models that enable advanced memory components*
2. *They do not really learn how to remember*
   ↳ *We should train the memory component over time*

These led us to development of RNNs!

RNN: *Less Generic Definition*

An RNN can be designed with any known architecture by letting NN also learn from its past features and outputs. This new enabling is called recurrence

RNN: *Dataset and Learning Setting*

We are now looking into a supervised learning problem where we are to

learn label from a sequence of data that has generally a temporal correlation

RNN: *Dataset and Learning Setting*

We are now looking into a supervised learning problem where we are to

learn label from a sequence of data that has generally a temporal correlation

Let's denote the sequence with $\mathbf{x}[1], \dots, \mathbf{x}[T]$

Temporal Correlation

By temporal correlation we mean that entries at other time instances carry information about one particular entry $\mathbf{x}[t]$

RNN: *Dataset and Learning Setting*

We are now looking into a supervised learning problem where we are to

learn label from a sequence of data that has generally a temporal correlation

Let's denote the sequence with $\mathbf{x}[1], \ldots, \mathbf{x}[T]$

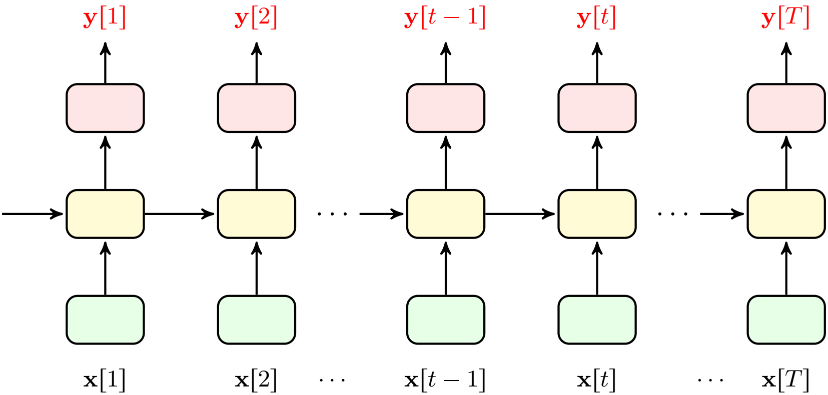
Temporal Correlation

By temporal correlation we mean that entries at other time instances carry information about one particular entry $\mathbf{x}[t]$

+ *But how is a label assigned to this sequence?*
– Well, that can be of various forms!

Types of Problems with Sequence Data

We considered a very simple case: many-to-many type I

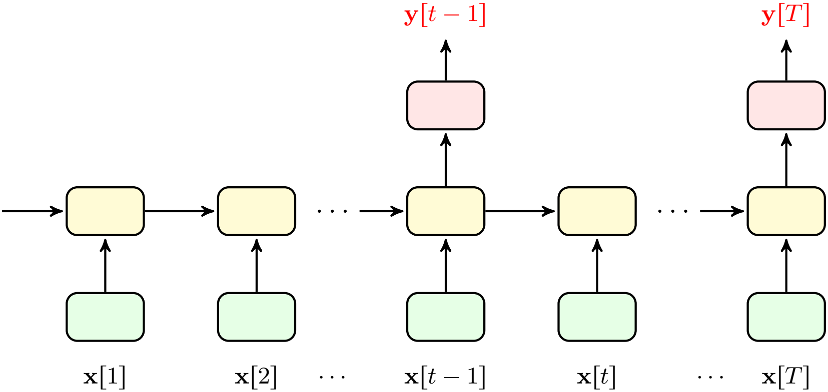


In this case, every entry has a label

↳ *speech tagging:* $\mathbf{x}[t]$ *is a part of speech and* $\mathbf{y}[t]$ *is its tag*

Types of Problems with Sequence Data

We considered a very simple case: many-to-many type II

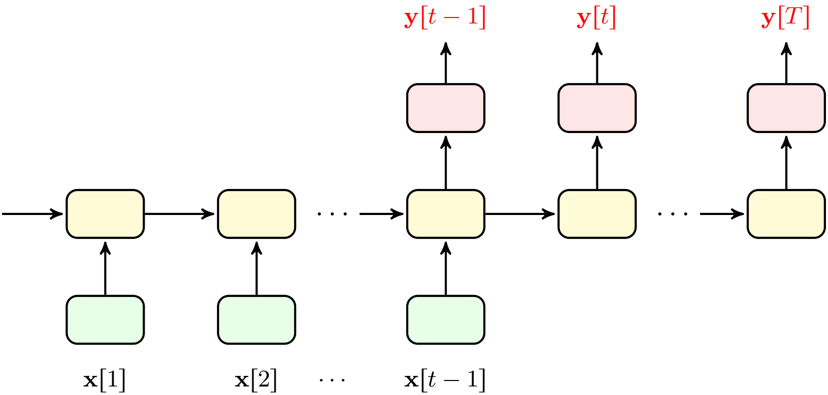


In this case, we get a label once after multiple entries

↳ *speech recognition: $\mathbf{x}[t]$ is a small part of speech and $\mathbf{y}[t]$ says what is a every couple of minutes about*

Types of Problems with Sequence Data

We considered a very simple case: many-to-many type III

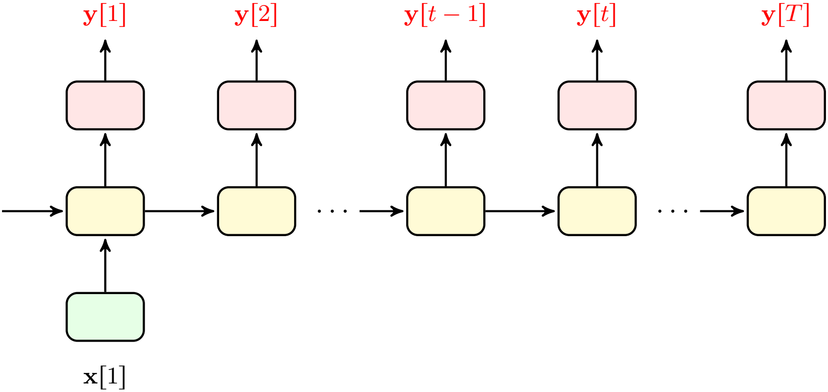


In this case, we start to get labels after some delay

↳ *language translation:* $\mathbf{x}[1], \ldots, \mathbf{x}[t-1]$ *is a sentence in German and* $\mathbf{y}[t-1], \ldots, \mathbf{y}[T]$ *is its translation to English*

Types of Problems with Sequence Data

We considered a very simple case: one-to-many

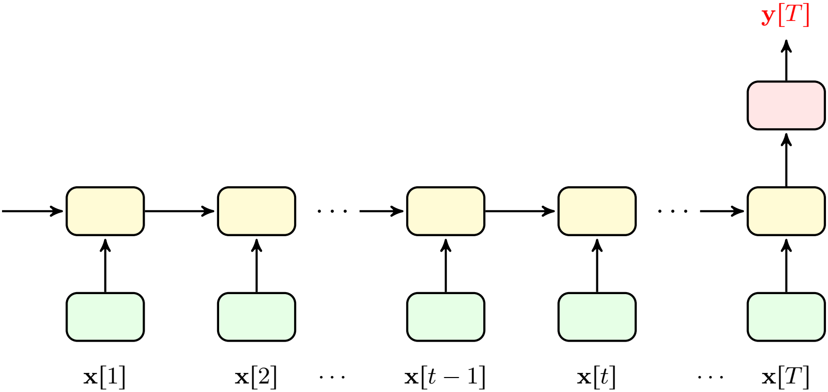


In this case, we get only one input data and have a sequence of labels

↳ *image captioning: $\mathbf{x}[1]$ is an image and $\mathbf{y}[1], \ldots, \mathbf{y}[T]$ is a caption describing what is inside the image*

Types of Problems with Sequence Data

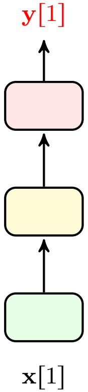
We considered a very simple case: many-to-one



In this case, we get only one label for a whole input sequence

↳ *sequence classification:* $\mathbf{x}[1], \dots, \mathbf{x}[T]$ *is a speech and* $\mathbf{y}[T]$ *says if this speech is constructive or destructive*

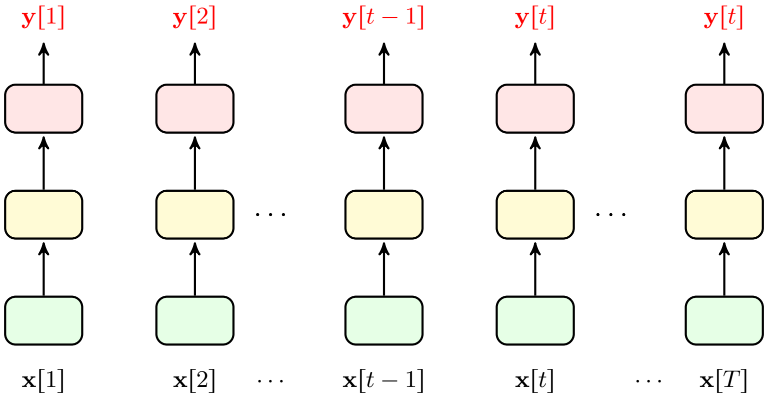
Types of Problems with Sequence Data: *FNNs*



In fact, FNNs are one-to-one RNNs

↳ *we can think of every data-point as a sequence of length one, or*

Types of Problems with Sequence Data: *FNNs*



In fact, FNNs are one-to-one RNNs

↳ *we can think of every data-point as a sequence of length one, or*

↳ *we may think of dataset as a long sequence with no temporal correlation*

RNN: *General Form*

To construct an RNN, we can use any module that we have learned:

- *we can use a fully-connected layer*
- *we can use a convolutional layer*
- *we can use a residual unit*
- *we may use an inception unit used in GoogLeNet*

  $\ldots$

RNN: *General Form*

To construct an RNN, we can use any module that we have learned:

- *we can use a fully-connected layer*
- *we can use a convolutional layer*
- *we can use a residual unit*
- *we may use an inception unit used in GoogLeNet*

  ...

But, classical implementations use fully-connected layers

RNN: *General Form*

To construct an RNN, we can use any module that we have learned:

- *we can use a fully-connected layer*
- *we can use a convolutional layer*
- *we can use a residual unit*
- *we may use an inception unit used in GoogLeNet*

  . . .

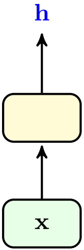
But, classical implementations use fully-connected layers

We can use one layer to make a shallow RNN or multiple to make a deep RNN

+ *Don't we do any change to them?*
- Not a serious change
  - ↳ *We may change the activation to* $\tanh(\cdot)$*: we will see later why*
  - ↳ *We expand the input dimension: since we need to also give memory as input*

Shallow RNN

Let's break it down a bit: *say the yellow box is a fully-connected layer*

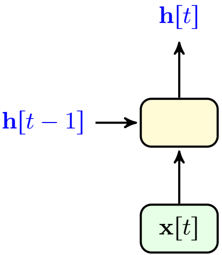


This layer gets input $\mathbf{x} \in \mathbb{R}^N$ and returns activated feature $\mathbf{h} \in \mathbb{R}^M$

- *We replace its activation by $\tanh(\cdot)$*
  - ↳ *Not necessary, but usually suggested*
- *We modify it to get a new input in $\mathbb{R}^{N+M}$*
  - ↳ *N entries for data inputs $\mathbf{x}$ in time t*
  - ↳ *M entries for features $\mathbf{h}$ in time $t-1$*

Shallow RNN

Let's break it down a bit: *say the yellow box is a fully-connected layer*

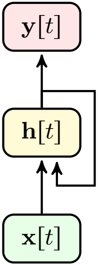


We show it this way now

- *We call $\mathbf{h}[t]$ usually the hidden state*
- *We pass the hidden state through an output layer*
  - ↳ *Output is not necessarily corresponding to label*

Shallow RNN

We may show our shallow RNN compactly via the following diagram

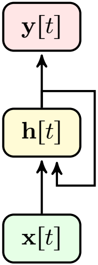


In this diagram

- *Each edge is a set of weights, e.g., a weight matrix*
- *The return edge also has a delay in time*

Shallow RNN

We may show our shallow RNN compactly via the following diagram



In this diagram

- *Each edge is a set of weights, e.g., a weight matrix*
- *The return edge also has a delay in time*

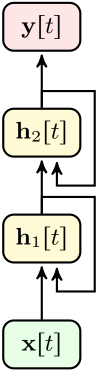
\+ *But, isn't that simply Elman Network?*

– If we use a fully-connected layer and sigmoid activation; then, Yes! *But, Remember that*

  - *Elman did not train it over time*
  - *Elman in its model used sigmoid activation*

Deep RNN

We can add more layers to make a deep RNN

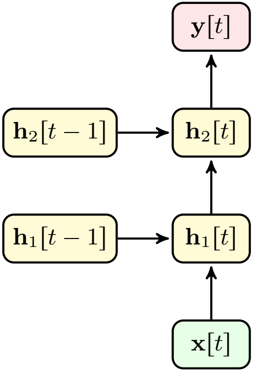


In this diagram

- *Each edge is a set of weights, e.g., a weight matrix*
- *The return edge also has a delay in time*
- *And, it's no more Elman Network*

Deep RNN

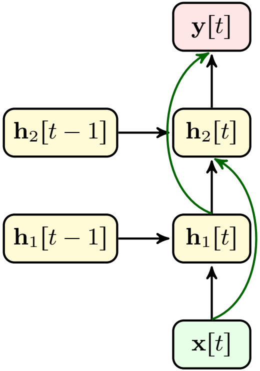
It might be easier to think of it as the following diagram



We can use any module that we like

Deep RNN

It might be easier to think of it as the following diagram

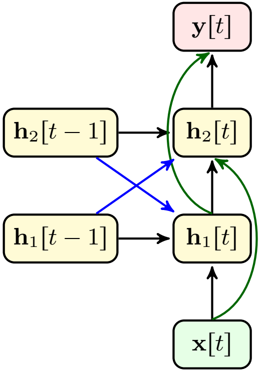


We can use any module that we like

- *We may add skip connection*

Deep RNN

It might be easier to think of it as the following diagram

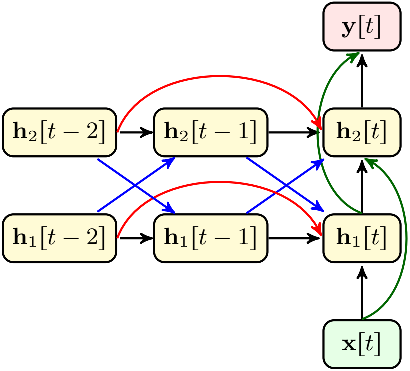


We can use any module that we like

- *We may add skip connection*
- *We could make dense connections*

Deep RNN

It might be easier to think of it as the following diagram

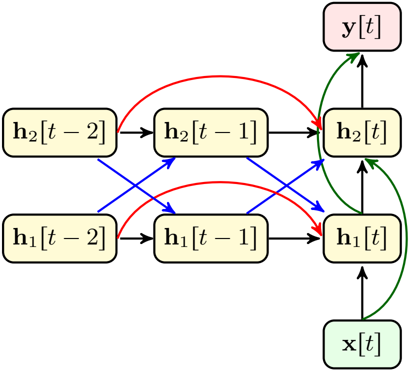


We can use any module that we like

- *We may add skip connection*
- *We could make dense connections*
- *We may skip over time*

Deep RNN

It might be easier to think of it as the following diagram



We can use any module that we like

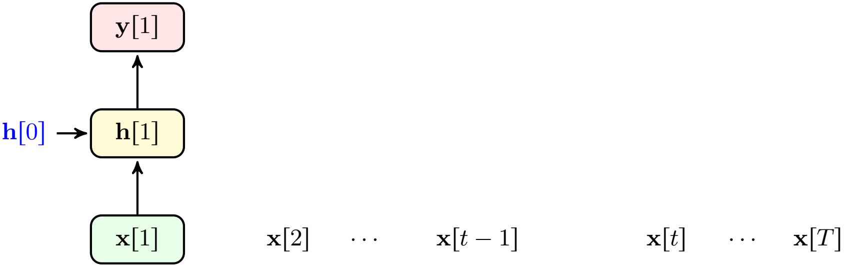
- *We may add skip connection*
- *We could make dense connections*
- *We may skip over time*
- *We may replace hidden layers with convolutions or anything else*

Shallow RNN: *Elman-Like Network*

Let's start with simple RNN: *a shallow RNN with fully-connected hidden layer*

+ *You mean Elman Network?*
– Yes, but *we now try to address the challenges that Elman did not address*

Let's look ts the flow of information once again

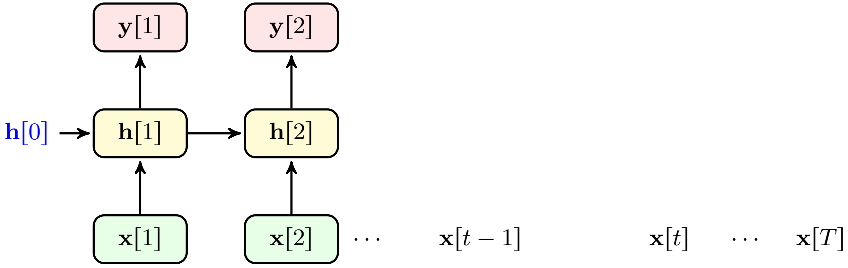


Shallow RNN: *Elman-Like Network*

Let's start with simple RNN: *a shallow RNN with fully-connected hidden layer*

+ *You mean Elman Network?*
- Yes, but *we now try to address the challenges that Elman did not address*

Let's look ts the flow of information once again

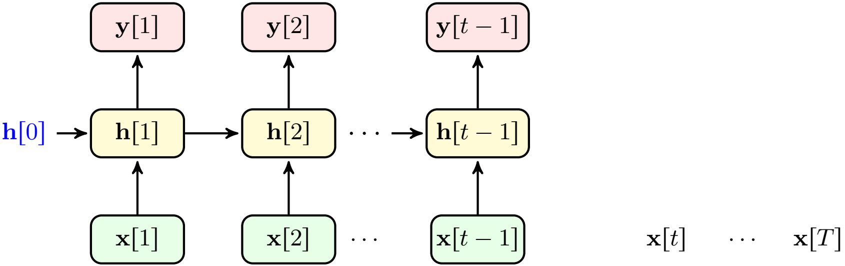


Shallow RNN: *Elman-Like Network*

Let's start with simple RNN: *a shallow RNN with fully-connected hidden layer*

+ *You mean Elman Network?*
– Yes, but *we now try to address the challenges that Elman did not address*

Let's look ts the flow of information once again

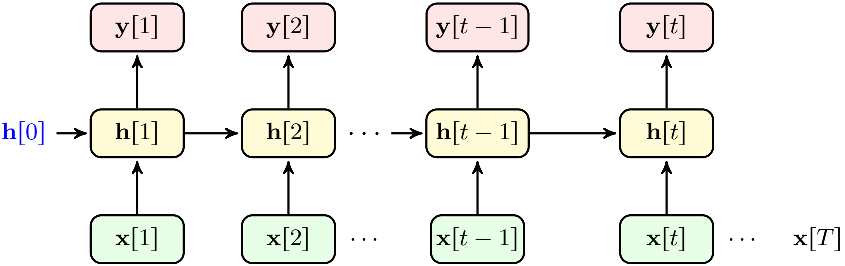


Shallow RNN: *Elman-Like Network*

Let's start with simple RNN: *a shallow RNN with fully-connected hidden layer*

- \+ *You mean Elman Network?*
- – Yes, but *we now try to address the challenges that Elman did not address*

Let's look ts the flow of information once again

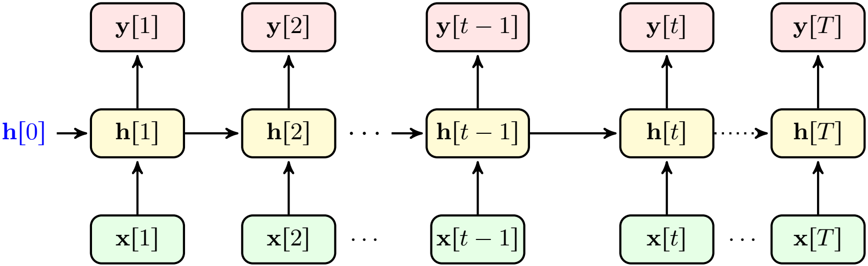


Shallow RNN: *Elman-Like Network*

Let's start with simple RNN: *a shallow RNN with fully-connected hidden layer*

+ *You mean Elman Network?*
– Yes, but *we now try to address the challenges that Elman did not address*

Let's look ts the flow of information once again



Shallow RNN: *Forward Propagation*

Let's specify the learnable parameters with some colors

$$\mathbf{y}[t]$$

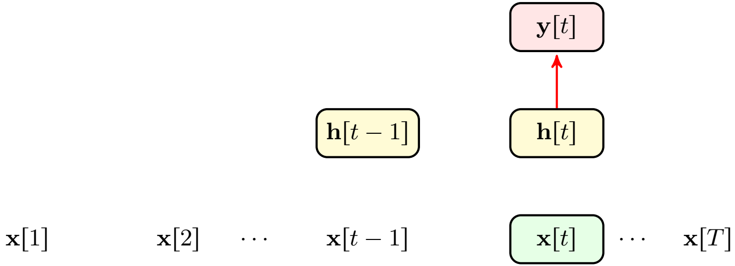
$$\mathbf{h}[t-1] \qquad \mathbf{h}[t]$$

$$\mathbf{x}[1] \qquad \mathbf{x}[2] \quad \cdots \quad \mathbf{x}[t-1] \qquad \mathbf{x}[t] \quad \cdots \quad \mathbf{x}[T]$$

Say we set the activation to $f(\cdot)$

Shallow RNN: *Forward Propagation*

Let's specify the learnable parameters with some colors

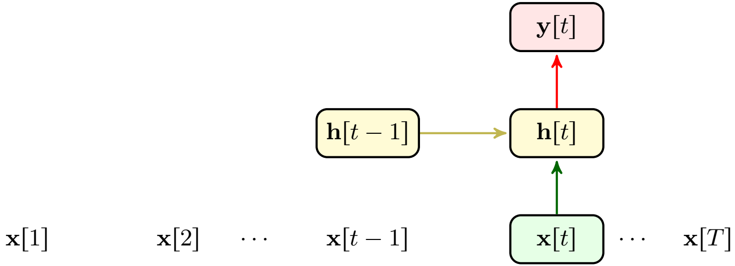


Say we set the activation to $f(\cdot)$

1. *We have* $\mathbf{y}[t] = f(\mathbf{W}_2\mathbf{h}[t])$: *we can learn* $\mathbf{W}_2$

Shallow RNN: *Forward Propagation*

Let's specify the learnable parameters with some colors

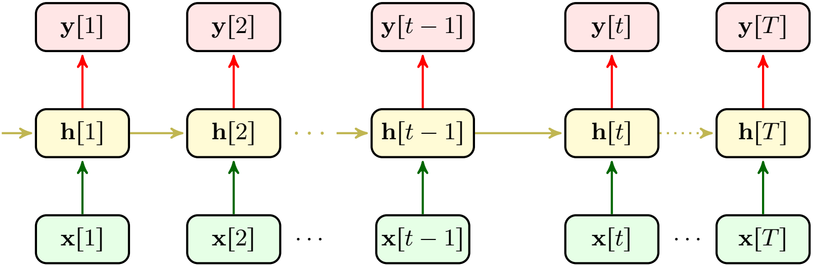


Say we set the activation to $f(\cdot)$

1. *We have* $\mathbf{y}[t] = f(\mathbf{W}_2\mathbf{h}[t])$: *we can learn* $\mathbf{W}_2$
2. *We have* $\mathbf{h}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-1])$: *we can learn* $\mathbf{W}_1$ *and* $\mathbf{W}_\mathrm{m}$

Shallow RNN: *Forward Propagation*

Let's specify the learnable parameters with some colors

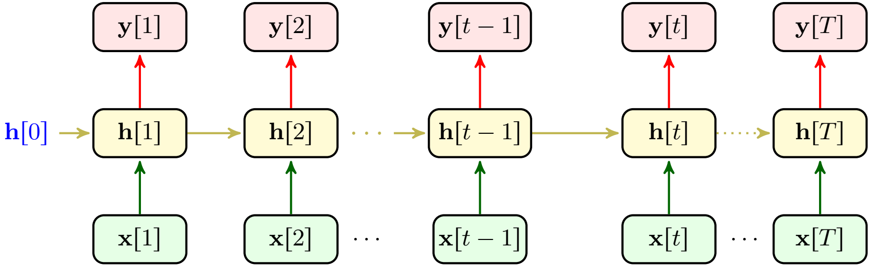


Say we set the activation to $f(\cdot)$

1. *We have* $\mathbf{y}[t] = f(\mathbf{W}_2\mathbf{h}[t])$: *we can learn* $\mathbf{W}_2$
2. *We have* $\mathbf{h}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-1])$: *we can learn* $\mathbf{W}_1$ *and* $\mathbf{W}_\mathrm{m}$

Shallow RNN: *Forward Propagation*

Let's specify the learnable parameters with some colors

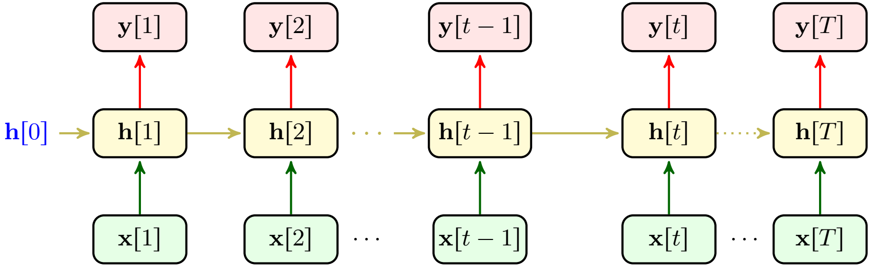


Say we set the activation to $f(\cdot)$

1. *We have $\mathbf{y}[t] = f(\mathbf{W}_2\mathbf{h}[t])$: we can learn $\mathbf{W}_2$*
2. *We have $\mathbf{h}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-1])$: we can learn $\mathbf{W}_1$ and $\mathbf{W}_\mathrm{m}$*
3. *We can start with any hidden state: this means we can learn $\mathbf{h}[0]$ too*

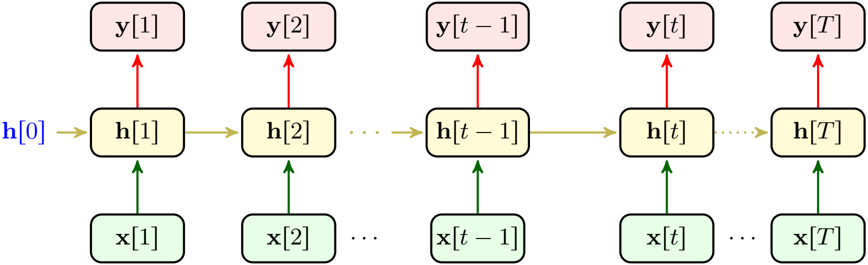
Training our Shallow RNN

We now want to train this basic RNN



Training our Shallow RNN

We now want to train this basic RNN



Let's consider training for only one sample

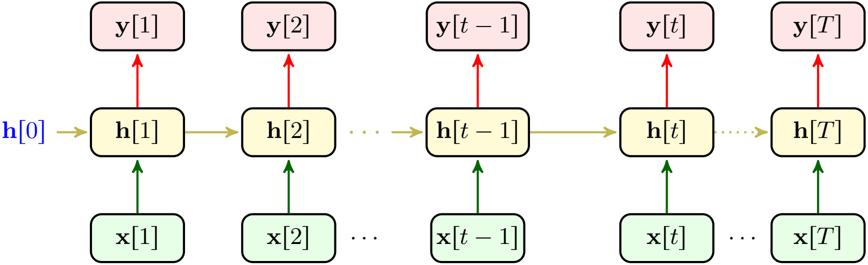
We assume that we have a sequence of labels $\mathbf{v}[1], \ldots, \mathbf{v}[T]$

1. *We know some $\mathbf{v}[t]$ could be empty, e.g., in many-to-one scenario*
2. *So could be some of $\mathbf{x}[t]$'s, e.g., in one-to-many scenario*

We however have no problem with that!

Training our Shallow RNN

We now want to train this basic RNN



Let's consider training for only one sample

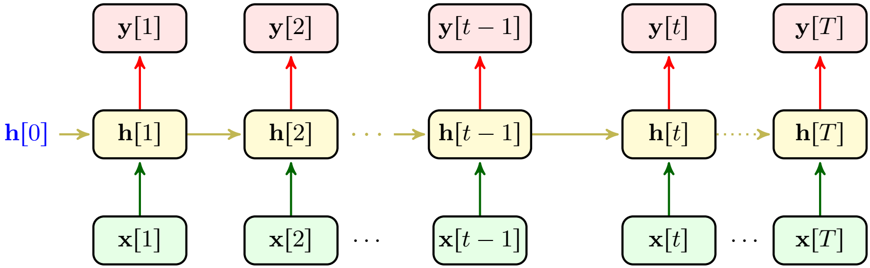
The loss in general can be written as

$$\hat{R} = \mathcal{L}\left(\mathbf{y}[1:T], \mathbf{v}[1:T]\right)$$

where we use shorten notation $\mathbf{y}[1:T] = \mathbf{y}[1], \ldots, \mathbf{y}[T]$

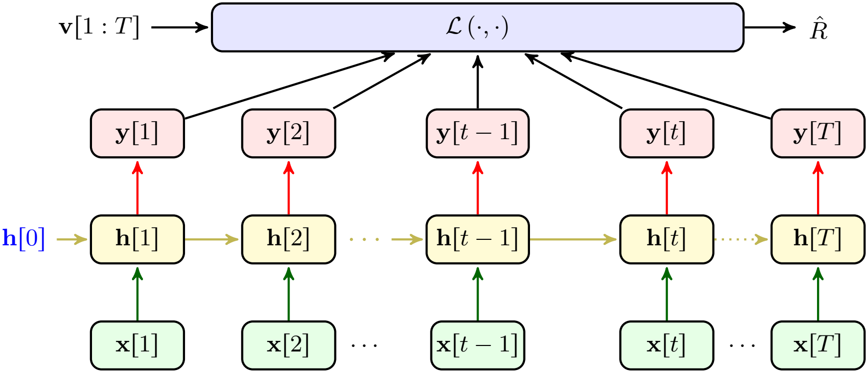
Training our Shallow RNN

We can think of such a diagram



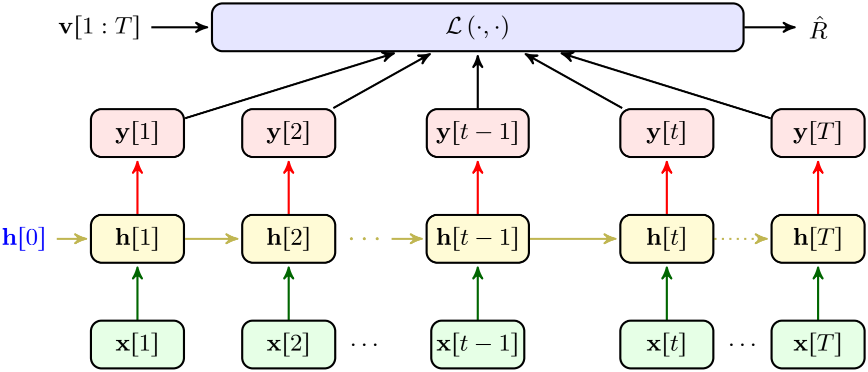
Training our Shallow RNN

We can think of such a diagram



Training our Shallow RNN

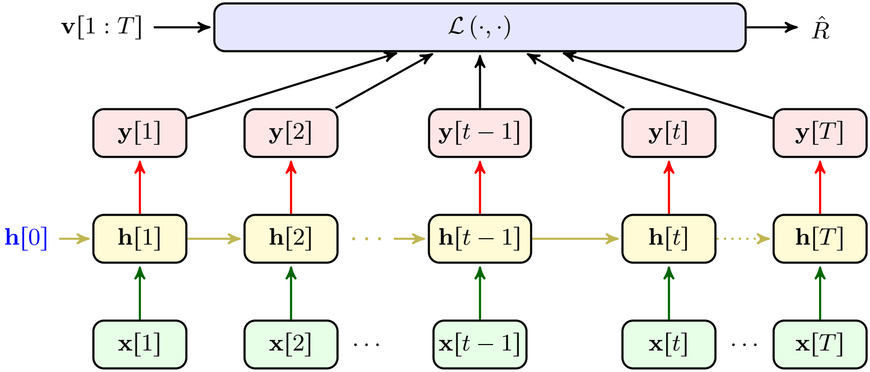
We can think of such a diagram



+ *It seems to be hard to get back from $\hat{R}$ to each $\mathbf{y}[t]$*
– Well! That's true, but we have some remedy for it

Training our Shallow RNN

We can think of such a diagram

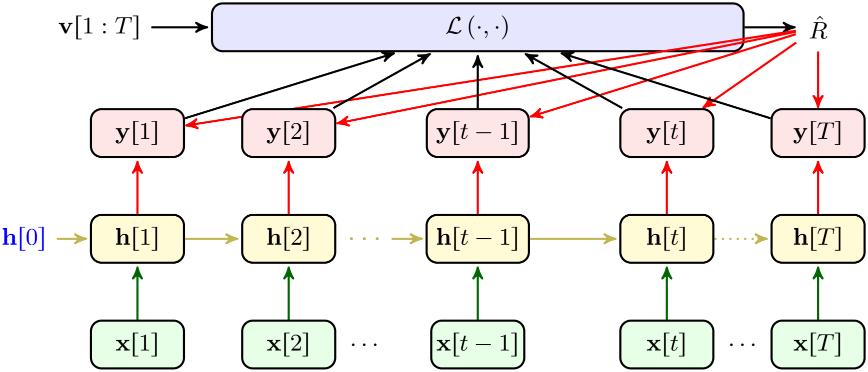


For the moment, we assume that we can

compute the $\nabla_{\mathbf{y}[t]} \hat{R}$ *for all* t

Training our Shallow RNN

We can think of such a diagram



For the moment, we assume that we can

compute the $\nabla_{\mathbf{y}[t]} \hat{R}$ for all $t \equiv$ move backward from $\hat{R}$ to any $\mathbf{y}[t]$

Backward Pass Through Time

Starting from $\hat{R}$, *say we want to find* $\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R}$

- *Since* $\hat{R}$ *is a function of* $\mathbf{y}[1:T]$, *we should write a vectorized chain rule*

$$\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R} =$$

Backward Pass Through Time

Starting from $\hat{R}$, *say we want to find* $\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R}$

- *Since* $\hat{R}$ *is a function of* $\mathbf{y}[1:T]$, *we should write a vectorized chain rule*

$$\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R} = \nabla_{\mathbf{y}[1]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[1]$$

Backward Pass Through Time

Starting from $\hat{R}$, *say we want to find* $\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R}$

- *Since* $\hat{R}$ *is a function of* $\mathbf{y}[1:T]$, *we should write a vectorized chain rule*

$$\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R} = \nabla_{\mathbf{y}[1]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[1] + \ldots + \nabla_{\mathbf{y}[T]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[T]$$

Backward Pass Through Time

Starting from $\hat{R}$, *say we want to find* $\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R}$

- *Since* $\hat{R}$ *is a function of* $\mathbf{y}[1:T]$, *we should write a vectorized chain rule*

$$
\begin{aligned}
\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R} &= \nabla_{\mathbf{y}[1]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[1] + \ldots + \nabla_{\mathbf{y}[T]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[T] \\
&= \sum_{t=1}^{T} \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[t]
\end{aligned}
$$

Backward Pass Through Time

Starting from $\hat{R}$, *say we want to find* $\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R}$

- *Since* $\hat{R}$ *is a function of* $\mathbf{y}[1:T]$, *we should write a vectorized chain rule*

$$\begin{aligned}\nabla_{\mathbf{W}_{\mathrm{m}}} \hat{R} &= \nabla_{\mathbf{y}[1]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[1] + \ldots + \nabla_{\mathbf{y}[T]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[T] \\ &= \sum_{t=1}^{T} \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[t]\end{aligned}$$

- *Since we assumed that we have* $\nabla_{\mathbf{y}[t]} \hat{R}$, *our main task is to find*

$$\nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[t]$$

for all t: *we could hence compute it for a general* t

↳ *This is a tensor-like gradient, i.e.,* $[\nabla_{\mathbf{W}_{\mathrm{m}}} y_1[t], \ldots, \nabla_{\mathbf{W}_{\mathrm{m}}} y_M[t]]$

Backward Pass Through Time

Starting from $\hat{R}$, *say we want to find* $\nabla_{\mathbf{w}_{\mathrm{m}}} \hat{R}$

- *Since* $\hat{R}$ *is a function of* $\mathbf{y}[1:T]$, *we should write a vectorized chain rule*

$$\begin{aligned}\nabla_{\mathbf{w}_{\mathrm{m}}} \hat{R} &= \nabla_{\mathbf{y}[1]} \hat{R} \circ \nabla_{\mathbf{w}_{\mathrm{m}}} \mathbf{y}[1] + \ldots + \nabla_{\mathbf{y}[T]} \hat{R} \circ \nabla_{\mathbf{w}_{\mathrm{m}}} \mathbf{y}[T] \\ &= \sum_{t=1}^{T} \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{w}_{\mathrm{m}}} \mathbf{y}[t]\end{aligned}$$

- *Since we assumed that we have* $\nabla_{\mathbf{y}[t]} \hat{R}$, *our main task is to find*

$$\nabla_{\mathbf{w}_{\mathrm{m}}} \mathbf{y}[t]$$

  for all t: *we could hence compute it for a general* t

  ↳ *This is a tensor-like gradient, i.e.,* $[\nabla_{\mathbf{w}_{\mathrm{m}}} y_1[t], \ldots, \nabla_{\mathbf{w}_{\mathrm{m}}} y_M[t]]$

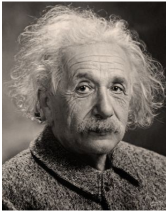
- *Apparently, we should apply chain rule for several times!*

Backward Pass Through Time

+ *But, this is going to be exhausting?*

Backward Pass Through Time

+ *But, this is going to be exhausting?*
– Well, we could again follow Albert Einstein advice!



"Everything should be made as simple as possible, but not simpler!"

Backward Pass Through Time: *Simple Example*

Let's consider a dummy RNN with all variables being scalar

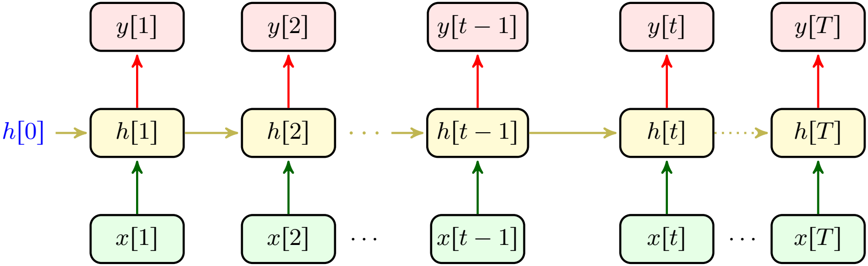
1. *We have* $y[t] = f(w_2 h[t])$
2. *We have* $h[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$
3. *We start with hidden state* $h[0]$

Backward Pass Through Time: *Simple Example*

Let's consider a dummy RNN with all variables being scalar

1. *We have* $y[t] = f(w_2 h[t])$
2. *We have* $h[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$
3. *We start with hidden state* $h[0]$

So the diagram gets simplified as below



Backward Pass Through Time: *Simple Example*

Starting from $\hat{R}$, say we want to find $\nabla_{w_{\mathrm{m}}} \hat{R}$: *our main task is to find*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}}$$

Let's start the computation:

Backward Pass Through Time: *Simple Example*

Starting from $\hat{R}$, say we want to find $\nabla_{w_{\mathrm{m}}} \hat{R}$: *our main task is to find*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}}$$

Let's start the computation: say we have passed forward through the RNN

- *we now have* $y[t]$, $h[t]$ *and* $x[t]$ *for all* t

Backward Pass Through Time: *Simple Example*

Starting from $\hat{R}$, say we want to find $\nabla_{w_\mathrm{m}} \hat{R}$: *our main task is to find*

$$\frac{\partial y[t]}{\partial w_\mathrm{m}}$$

Let's start the computation: say we have passed forward through the RNN

- *we now have* $y[t]$, $h[t]$ *and* $x[t]$ *for all* t

To go backward, we note that $y[t] = f(w_2 h[t])$

Backward Pass Through Time: *Simple Example*

Starting from $\hat{R}$, say we want to find $\nabla_{w_{\rm m}} \hat{R}$: *our main task is to find*

$$\frac{\partial y[t]}{\partial w_{\rm m}}$$

Let's start the computation: say we have passed forward through the RNN

- *we now have* $y[t]$, $h[t]$ *and* $x[t]$ *for all* t

To go backward, we note that $y[t] = f(w_2 h[t])$

- $y[t]$ *is a function of* $h[t]$*: so we write the chain rule as*

$$\frac{\partial y[t]}{\partial w_{\rm m}} = \frac{\partial y[t]}{\partial h[t]} \frac{\partial h[t]}{\partial w_{\rm m}}$$

↳ *we can compute the first term as* $\partial y[t] / \partial h[t] = w_2 \dot{f}(w_2 h[t])$
↳ *we should compute the second term by further chain rule*

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$

- $h[t]$ *is a function of* w_{m} *and* $h[t-1]$:[4]

[4]*We ignore* w_1 *and* $x[t]$*, as they are obviously not functions of* w_{m}

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$

- $h[t]$ *is a function of* w_{m} *and* $h[t-1]$:[4]
  ↳ *let's write* $h[t] = g\left(w_{\mathrm{m}}, h[t-1]\right)$

[4] *We ignore* w_1 *and* $x[t]$*, as they are obviously not functions of* w_{m}

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$

- $h[t]$ *is a function of* w_{m} *and* $h[t-1]$*:*[4]
  ↳ *let's write* $h[t] = g\left(w_{\mathrm{m}}, h[t-1]\right)$
- *So we write the chain rule as*

$$\frac{\partial h[t]}{\partial w_{\mathrm{m}}} =$$

[4] *We ignore* w_1 *and* $x[t]$*, as they are obviously not functions of* w_{m}

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$

- $h[t]$ *is a function of* w_{m} *and* $h[t-1]$:[4]
  - ↳ *let's write* $h[t] = g(w_{\mathrm{m}}, h[t-1])$
- *So we write the chain rule as*

$$\frac{\partial h[t]}{\partial w_{\mathrm{m}}} = \frac{\partial g}{\partial w_{\mathrm{m}}} \underbrace{\frac{\partial w_{\mathrm{m}}}{\partial w_{\mathrm{m}}}}_{1}$$

[4] *We ignore* w_1 *and* $x[t]$, *as they are obviously not functions of* w_{m}

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$

- $h[t]$ *is a function of* w_{m} *and* $h[t-1]$:[4]
  - ↳ *let's write* $h[t] = g(w_{\mathrm{m}}, h[t-1])$
- *So we write the chain rule as*

$$\frac{\partial h[t]}{\partial w_{\mathrm{m}}} = \frac{\partial g}{\partial w_{\mathrm{m}}} \underbrace{\frac{\partial w_{\mathrm{m}}}{\partial w_{\mathrm{m}}}}_{1} + \frac{\partial g}{\partial h[t-1]} \frac{\partial h[t-1]}{\partial w_{\mathrm{m}}}$$

[4] *We ignore* w_1 *and* $x[t]$, *as they are obviously not functions of* w_{m}

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$

- $h[t]$ *is a function of* w_{m} *and* $h[t-1]$*:*[4]
  ↳ *let's write* $h[t] = g\left(w_{\mathrm{m}}, h[t-1]\right)$
- *So we write the chain rule as*

$$\begin{aligned}\frac{\partial h[t]}{\partial w_{\mathrm{m}}} &= \frac{\partial g}{\partial w_{\mathrm{m}}} \underbrace{\frac{\partial w_{\mathrm{m}}}{\partial w_{\mathrm{m}}}}_{1} + \frac{\partial g}{\partial h[t-1]} \frac{\partial h[t-1]}{\partial w_{\mathrm{m}}} \\ &= \frac{\partial g}{\partial w_{\mathrm{m}}} + \frac{\partial g}{\partial h[t-1]} \frac{\partial h[t-1]}{\partial w_{\mathrm{m}}}\end{aligned}$$

[4] *We ignore* w_1 *and* $x[t]$*, as they are obviously not functions of* w_{m}

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(z[t])$

- *Let's define* $z[t] = w_1 x[t] + w_{\mathrm{m}} h[t-1]$

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(z[t])$

- *Let's define* $z[t] = w_1 x[t] + w_{\mathrm{m}} h[t-1]$
- $h[t] = g\left(w_{\mathrm{m}}, h[t-1]\right)$
  - ↳ *we can compute* $\partial g / \partial w_{\mathrm{m}} = h[t-1] \dot{f}(z[t])$
  - ↳ *we can compute* $\partial g / \partial h[t-1] = w_{\mathrm{m}} \dot{f}(z[t])$

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

We keep going backward, by noting that $h[t] = f(z[t])$

- *Let's define* $z[t] = w_1 x[t] + w_{\mathrm{m}} h[t-1]$
- $h[t] = g\left(w_{\mathrm{m}}, h[t-1]\right)$
  - ↳ *we can compute* $\partial g / \partial w_{\mathrm{m}} = h[t-1] \dot{f}(z[t])$
  - ↳ *we can compute* $\partial g / \partial h[t-1] = w_{\mathrm{m}} \dot{f}(z[t])$
- *So we can simplify the chain rule as*

$$\begin{aligned} \frac{\partial h[t]}{\partial w_{\mathrm{m}}} &= h[t-1] \dot{f}\left(z[t]\right) + w_{\mathrm{m}} \dot{f}\left(z[t]\right) \frac{\partial h[t-1]}{\partial w_{\mathrm{m}}} \\ &= \dot{f}\left(z[t]\right) \left(h[t-1] + w_{\mathrm{m}} \frac{\partial h[t-1]}{\partial w_{\mathrm{m}}} \right) \end{aligned}$$

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \dot{f}\left(z[t]\right) \left(h[t-1] + w_{\mathrm{m}} \frac{\partial h[t-1]}{\partial w_{\mathrm{m}}} \right)$$

We keep going backward: $h[t-1] = f\left(z[t-1]\right)$

↳ *Recall that* $z[t-1] = w_1 x[t-1] + w_{\mathrm{m}} h[t-2]$

Backward Pass Through Time: *Simple Example*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \dot{f}\left(z[t]\right) \left(h[t-1] + w_{\mathrm{m}} \frac{\partial h[t-1]}{\partial w_{\mathrm{m}}} \right)$$

We keep going backward: $h[t-1] = f\left(z[t-1]\right)$

↳ *Recall that* $z[t-1] = w_1 x[t-1] + w_{\mathrm{m}} h[t-2]$

- *We just need to replace* t *with* $t-1$ *in the last derivation*

$$\frac{\partial h[t-1]}{\partial w_{\mathrm{m}}} = \dot{f}\left(z[t-1]\right) \left(h[t-2] + w_{\mathrm{m}} \frac{\partial h[t-2]}{\partial w_{\mathrm{m}}} \right)$$

Backward Pass Through Time: *Simple Example*

Backpropagation at time t is hence described by a pair of recursive equations

- *At time t, we compute*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

Backward Pass Through Time: *Simple Example*

Backpropagation at time t is hence described by a pair of recursive equations

- *At time t, we compute*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

- *For each $i = t, t-1, \ldots, 1$, we use*

$$\frac{\partial h[i]}{\partial w_{\mathrm{m}}} = \dot{f}\left(z[i]\right) \left(h[i-1] + w_{\mathrm{m}} \frac{\partial h[i-1]}{\partial w_{\mathrm{m}}} \right)$$

Backward Pass Through Time: *Simple Example*

Backpropagation at time t is hence described by a pair of recursive equations

- *At time t, we compute*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

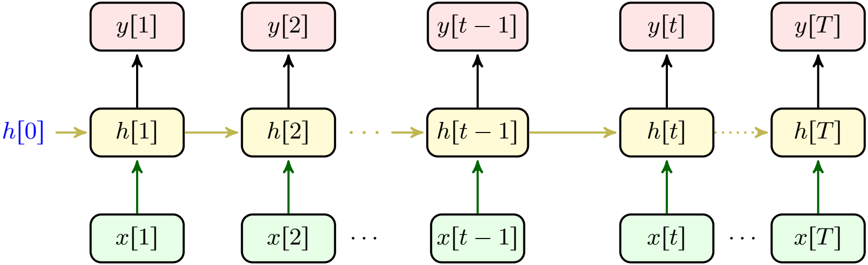
- *For each $i = t, t-1, \ldots, 1$, we use*

$$\frac{\partial h[i]}{\partial w_{\mathrm{m}}} = \dot{f}\left(z[i]\right)\left(h[i-1] + w_{\mathrm{m}} \frac{\partial h[i-1]}{\partial w_{\mathrm{m}}}\right)$$

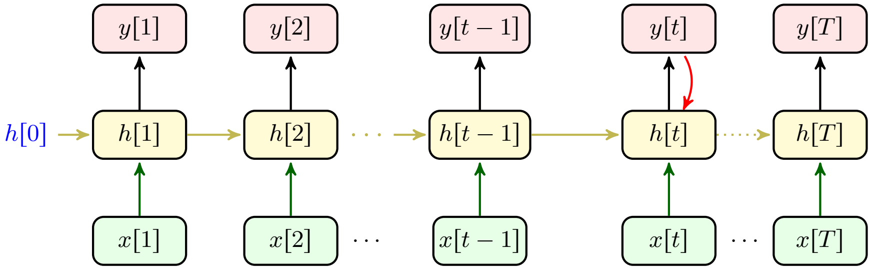
- *We stop at $i = 1$, where we get*

$$\frac{\partial h[1]}{\partial w_{\mathrm{m}}} = \dot{f}\left(z[1]\right)\left(h[0] + w_{\mathrm{m}} \underbrace{\frac{\partial h[0]}{\partial w_{\mathrm{m}}}}_{0}\right) = \dot{f}\left(z[1]\right) h[0]$$

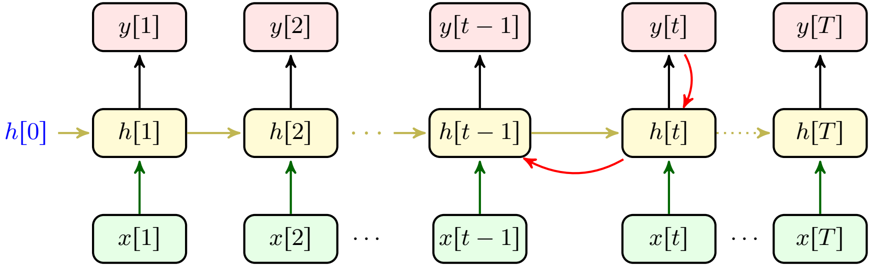
Backward Pass Through Time: *Simple Example*



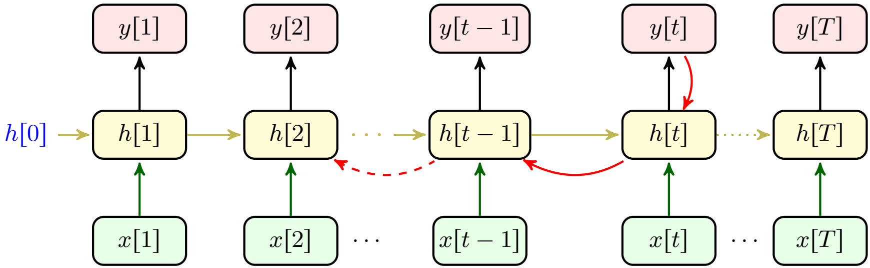
Backward Pass Through Time: *Simple Example*



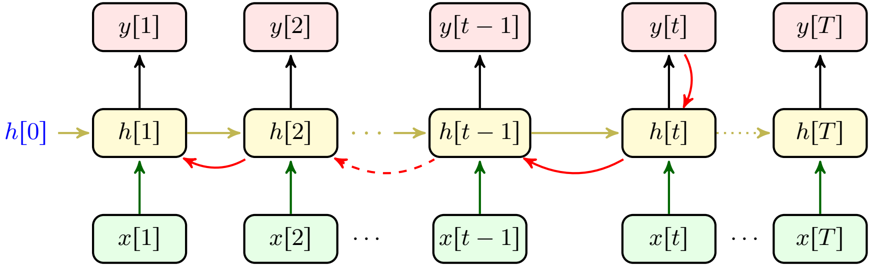
Backward Pass Through Time: *Simple Example*



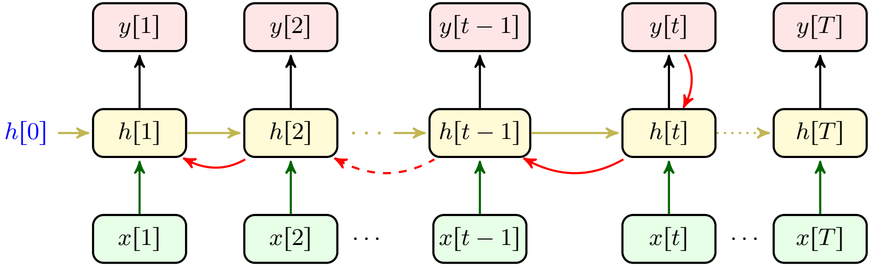
Backward Pass Through Time: *Simple Example*



Backward Pass Through Time: *Simple Example*



Backward Pass Through Time: *Simple Example*



Key Point

Propagating back in time is described via a recursive equation

Backward Pass Through Time: *Simple Example*

Recursive equation has an interesting feature

- *Say we have backpropagated from time t*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

↳ *We hence have $\partial h[t]/\partial w_{\mathrm{m}}, \partial h[t-1]/\partial w_{\mathrm{m}}, \ldots, \partial h[1]/\partial w_{\mathrm{m}}$*

Backward Pass Through Time: *Simple Example*

Recursive equation has an interesting feature

- *Say we have backpropagated from time t*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

  ↳ *We hence have $\partial h[t]/\partial w_{\mathrm{m}}, \partial h[t-1]/\partial w_{\mathrm{m}}, \ldots, \partial h[1]/\partial w_{\mathrm{m}}$*

- *Now, if we want to backpropagate from $t-1$*

  ↳ *We already have $\partial h[t-1]/\partial w_{\mathrm{m}}, \partial h[t-2]/\partial w_{\mathrm{m}}, \ldots, \partial h[1]/\partial w_{\mathrm{m}}$*

  ↳ *We do not need to backpropagate through time anymore*

Backward Pass Through Time: *Simple Example*

Recursive equation has an interesting feature

- *Say we have backpropagated from time t*

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \frac{\partial h[t]}{\partial w_{\mathrm{m}}}$$

  ↳ *We hence have $\partial h[t]/\partial w_{\mathrm{m}}, \partial h[t-1]/\partial w_{\mathrm{m}}, \ldots, \partial h[1]/\partial w_{\mathrm{m}}$*

- *Now, if we want to backpropagate from $t-1$*

  ↳ *We already have $\partial h[t-1]/\partial w_{\mathrm{m}}, \partial h[t-2]/\partial w_{\mathrm{m}}, \ldots, \partial h[1]/\partial w_{\mathrm{m}}$*

  ↳ *We do not need to backpropagate through time anymore*

Moral of Story

Pass forward till end of sequence and backpropagate to the beginning just once

Backpropagation Through Time (BPTT)

`BPTT():`

1. *Start at* t *with* $\nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{y}[t] = \mathbf{W}_2 \circ \dot{f}(\mathbf{W}_2 \mathbf{h}[t]) \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{h}[t]$
2. *Go back in time as* $\nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{h}[i] = \dot{f}\,(\mathbf{z}[i]) \circ (\mathbf{h}[i-1] + \mathbf{W}_{\mathrm{m}} \circ \nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{h}[i-1])$
3. *Stop at* $i = 1$ *with* $\nabla_{\mathbf{W}_{\mathrm{m}}} \mathbf{h}[1] = \dot{f}\,(\mathbf{z}[1]) \circ \mathbf{h}[0]$

Backpropagation Through Time (BPTT)

```
BPTT():
```

1. *Start at* t *with* $\nabla_{\mathbf{W}_\mathrm{m}} \mathbf{y}[t] = \mathbf{W}_2 \circ \dot{f}(\mathbf{W}_2 \mathbf{h}[t]) \circ \nabla_{\mathbf{W}_\mathrm{m}} \mathbf{h}[t]$
2. *Go back in time as* $\nabla_{\mathbf{W}_\mathrm{m}} \mathbf{h}[i] = \dot{f}\left(\mathbf{z}[i]\right) \circ \left(\mathbf{h}[i-1] + \mathbf{W}_\mathrm{m} \circ \nabla_{\mathbf{W}_\mathrm{m}} \mathbf{h}[i-1]\right)$
3. *Stop at* $i = 1$ *with* $\nabla_{\mathbf{W}_\mathrm{m}} \mathbf{h}[1] = \dot{f}\left(\mathbf{z}[1]\right) \circ \mathbf{h}[0]$

\+ *It looks very similar to backpropagation in deep NNs!*

– Exactly! *Even simple RNN is very deep through time*

\+ *Don't we experience vanishing or exploding gradient then!*

– Yes! Let's check it out

BPTT: *Vanishing Gradient*

Let's expand the gradient in our example

$$\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = w_2 \dot{f}(w_2 h[t]) \dot{f}\left(z[t]\right) h[t-1]$$

BPTT: *Vanishing Gradient*

Let's expand the gradient in our example

$$
\begin{aligned}
\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = \; & w_2 \dot{f}(w_2 h[t]) \dot{f}\left(z[t]\right) h[t-1] \\
& + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}} \dot{f}\left(z[t]\right) \dot{f}\left(z[t-1]\right) h[t-2]
\end{aligned}
$$

BPTT: *Vanishing Gradient*

Let's expand the gradient in our example

$$
\begin{aligned}
\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = \; & w_2 \dot{f}(w_2 h[t]) \dot{f}\left(z[t]\right) h[t-1] \\
& + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}} \dot{f}\left(z[t]\right) \dot{f}\left(z[t-1]\right) h[t-2] \\
& + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}}^2 \dot{f}\left(z[t]\right) \dot{f}\left(z[t-1]\right) \dot{f}\left(z[t-2]\right) h[t-3] \\
& + \cdots
\end{aligned}
$$

BPTT: *Vanishing Gradient*

Let's expand the gradient in our example

$$\begin{aligned}\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = \; & w_2 \dot{f}(w_2 h[t]) \dot{f}\left(z[t]\right) h[t-1] \\ & + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}} \dot{f}\left(z[t]\right) \dot{f}\left(z[t-1]\right) h[t-2] \\ & + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}}^2 \dot{f}\left(z[t]\right) \dot{f}\left(z[t-1]\right) \dot{f}\left(z[t-2]\right) h[t-3] \\ & + \cdots \\ & + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}}^{i-1} \left(\prod_{j=0}^{i-1} \dot{f}\left(z[t-j]\right) \right) h[t-i] \\ & + \cdots \end{aligned}$$

BPTT: *Vanishing Gradient*

Let's expand the gradient in our example

$$
\begin{aligned}
\frac{\partial y[t]}{\partial w_{\mathrm{m}}} = \; & w_2 \dot{f}(w_2 h[t]) \dot{f}\left(z[t]\right) h[t-1] \\
& + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}} \dot{f}\left(z[t]\right) \dot{f}\left(z[t-1]\right) h[t-2] \\
& + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}}^2 \dot{f}\left(z[t]\right) \dot{f}\left(z[t-1]\right) \dot{f}\left(z[t-2]\right) h[t-3] \\
& + \cdots \\
& + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}}^{i-1} \left(\prod_{j=0}^{i-1} \dot{f}\left(z[t-j]\right) \right) h[t-i] \\
& + \cdots \\
& + w_2 \dot{f}(w_2 h[t]) w_{\mathrm{m}}^{t-1} \left(\prod_{j=0}^{t-1} \dot{f}\left(z[t-j]\right) \right) h[0]
\end{aligned}
$$

BPTT: *Vanishing Gradient*

This says that gradient of hidden state in i steps back in time is multiplied by

$$w_2 w_{\mathrm{m}}^{i-1} \dot{f}(w_2 h[t]) \left(\prod_{j=0}^{i-1} \dot{f}\left(z[t-j]\right) \right)$$

BPTT: *Vanishing Gradient*

This says that gradient of hidden state in i steps back in time is multiplied by

$$w_2 w_{\mathrm{m}}^{i-1} \dot{f}(w_2 h[t]) \left(\prod_{j=0}^{i-1} \dot{f}\left(z[t-j]\right) \right)$$

Recall deep NNs: we could see two cases

- *If $\dot{f}(\cdot) > 1$ most of the time*
  - ↳ *very old hidden states can explode the gradient*
  - ↳ *this usually does not happen, because most activations are not like that*

BPTT: *Vanishing Gradient*

This says that gradient of hidden state in i steps back in time is multiplied by

$$w_2 w_{\mathrm{m}}^{i-1} \dot{f}(w_2 h[t]) \left(\prod_{j=0}^{i-1} \dot{f}\left(z[t-j]\right) \right)$$

Recall deep NNs: we could see two cases

- *If $\dot{f}(\cdot) > 1$ most of the time*
  - ↳ *very old hidden states can explode the gradient*
  - ↳ *this usually does not happen, because most activations are not like that*
- *If $\dot{f}(\cdot) < 1$ most of the time*
  - ↳ *very old hidden states have pretty much no impact on the gradient*
  - ↳ *this means that the RNN can train only up to a finite memory*
  - ↳ *this typically happens and we call it vanishing gradient through time*

Handling *Vanishing Gradient Through Time*

+ *Cant we use the same remedy as in deep NNs?*

Handling *Vanishing Gradient Through Time*

+ *Cant we use the same remedy as in deep NNs?*
– Yes and No!

Handling *Vanishing Gradient Through Time*

- + *Cant we use the same remedy as in deep NNs?*
- – Yes and No!

It is important to note that *vanishing gradient through time is a bit different: we are still updating weights with large enough gradients, but these gradients have no information of long-term memory*

Handling *Vanishing Gradient Through Time*

+ *Cant we use the same remedy as in deep NNs?*
– Yes and No!

It is important to note that *vanishing gradient through time* is *a* bit *different: we are still updating weights with large enough gradients, but these gradients have no information of long-term memory*

Solutions to vanishing gradient through time are mainly two approaches

- *Use* $\tanh(\cdot)$ *activation*
  ↳ $\tanh(\cdot)$ *is able to keep memory for a longer period*
- *Use truncated BPTT*
  ↳ *Repeat short-term BPTTs every couple of time intervals*

Handling *Vanishing Gradient Through Time*

+ *Cant we use the same remedy as in deep NNs?*
- Yes and No!

It is important to note that *vanishing gradient through time* is a bit *different*: *we are still updating weights with large enough gradients, but these gradients have no information of long-term memory*

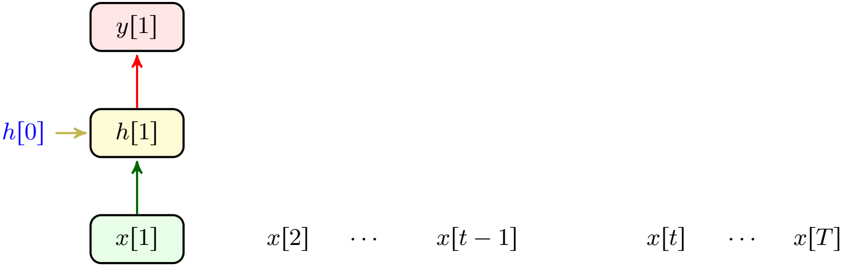
Solutions to vanishing gradient through time are mainly two approaches

- *Use* $\tanh(\cdot)$ *activation*
  ↳ $\tanh(\cdot)$ *is able to keep memory for a longer period*
- *Use truncated BPTT*
  ↳ *Repeat short-term BPTTs every couple of time intervals*
- *Invoke gating approach*
  ↳ *This is the most sophisticated approach*
  ↳ *It was known for a long time, but received attention much later!*

Principle of Gating

To understand the idea of *Gating*, let's *get back to our basic RNN*

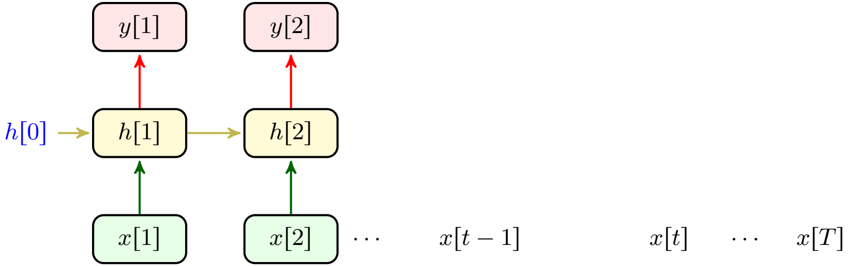
1. *We start with hidden state* $h[0]$
2. *We have* $h[t] = f(w_1 x[t] + w_m h[t-1])$
3. *We have* $y[t] = f(w_2 h[t])$



Principle of Gating

To understand the idea of *Gating*, let's *get back to our basic RNN*

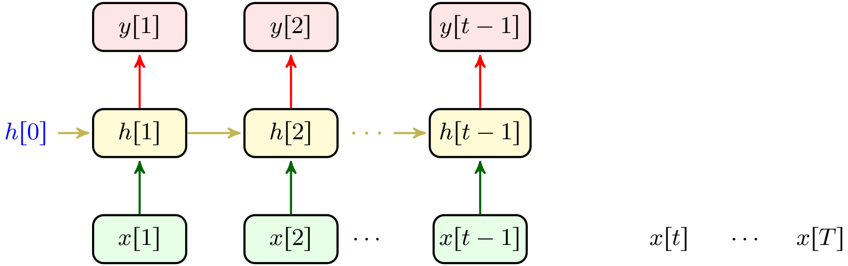
1. *We start with hidden state* $h[0]$
2. *We have* $h[t] = f(w_1 x[t] + w_{\rm m} h[t-1])$
3. *We have* $y[t] = f(w_2 h[t])$



Principle of Gating

To understand the idea of *Gating*, let's *get back to our basic RNN*

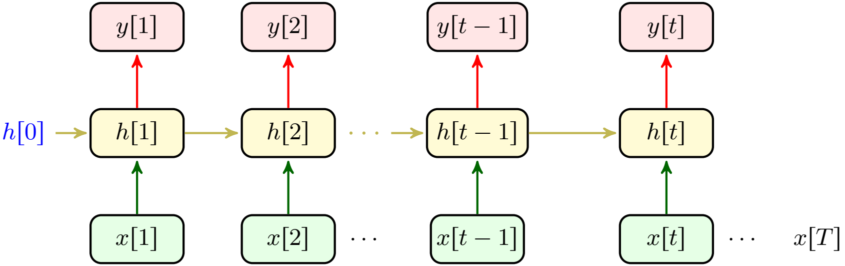
1. *We start with hidden state* $h[0]$
2. *We have* $h[t] = f(w_1 x[t] + w_m h[t-1])$
3. *We have* $y[t] = f(w_2 h[t])$



Principle of Gating

To understand the idea of *Gating*, let's *get back to our basic RNN*

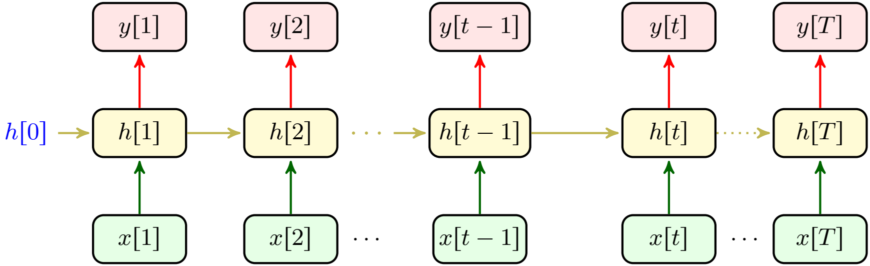
1. *We start with hidden state* $h[0]$
2. *We have* $h[t] = f(w_1 x[t] + w_{\rm m} h[t-1])$
3. *We have* $y[t] = f(w_2 h[t])$



Principle of Gating

To understand the idea of *Gating*, let's *get back to our basic RNN*

1. *We start with hidden state* $h[0]$
2. *We have* $h[t] = f(w_1 x[t] + w_m h[t-1])$
3. *We have* $y[t] = f(w_2 h[t])$



Looking at $h[t]$ *as memory, we can say we are always updating the memory*

Principle of Gating

Recall our motivating example: *we wanted to predict the next word*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Principle of Gating

Recall our motivating example: *we wanted to predict the next word*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Should we updating the memory all the way from Julia?

Principle of Gating

Recall our motivating example: *we wanted to predict the next word*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Should we updating the memory all the way from Julia?

- *Obviously No!*
  - ↳ *We should stop updating at Julia, since it is something we should remember*

Principle of Gating

Recall our motivating example: *we wanted to predict the next word*

$\mathbf{x}[t-6]$ $\mathbf{x}[t-5]$ $\mathbf{x}[t-4]$ $\mathbf{x}[t-3]$ $\mathbf{x}[t-2]$ $\mathbf{x}[t-1]$ $\mathbf{x}[t]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Should we updating the memory all the way from Julia?

- *Obviously No!*
  - ↳ *We should stop updating at Julia, since it is something we should remember*
- *How can we do it? Let's try a thought experiment*

Principle of Gating

Say we access to a sequence $u[t] \in [0, 1]$: *assume the following happens*

↳ We arrive at *"Julia"* at time t_0, i.e., $x[t_0] \propto$ *"Julia"*

↳ At time t_0, we have $u[t_0] = 1$

Principle of Gating

Say we access to a sequence $u[t] \in [0, 1]$: *assume the following happens*

- ↳ We arrive at *"Julia"* at time t_0, i.e., $x[t_0] \propto$ *"Julia"*
- ↳ At time t_0, we have $u[t_0] = 1$
- ↳ We want to predict at $t + 1$
- ↳ From $t_0 + 1$ to t, we have $u[t_0 + 1] = \cdots = u[t] = 0$

Principle of Gating

Say we access to a sequence $u[t] \in [0,1]$: *assume the following happens*

- ↳ We arrive at *"Julia"* at time t_0, i.e., $x[t_0] \propto$ *"Julia"*
- ↳ At time t_0, we have $u[t_0] = 1$
- ↳ We want to predict at $t+1$
- ↳ From t_0+1 to t, we have $u[t_0+1] = \cdots = u[t] = 0$

Now, we update our memory like this

1. *We start with hidden state* $h[0]$
2. *We compute* $\tilde{h}[t] = f(w_1 x[t] + w_{\mathrm{m}} h[t-1])$

4. *At each time, we have* $y[t] = f(w_2 h[t])$

Principle of Gating

Say we access to a sequence $u[t] \in [0,1]$: *assume the following happens*

↳ We arrive at *"Julia"* at time t_0, i.e., $x[t_0] \propto$ *"Julia"*

↳ At time t_0, we have $u[t_0] = 1$

↳ We want to predict at $t+1$

↳ From t_0+1 to t, we have $u[t_0+1] = \cdots = u[t] = 0$

Now, we update our memory like this

1. *We start with hidden state* $h[0]$
2. *We compute* $\tilde{h}[t] = f(w_1 x[t] + w_{\rm m} h[t-1])$
3. *We update* $h[t] = u[t]\tilde{h}[t] + (1-u[t])\, h[t-1]$
4. *At each time, we have* $y[t] = f(w_2 h[t])$

Let's see what happens to the memory

Principle of Gating

At time t_0, we can say

- *We have $x[t_0] \propto$ "Julia" and compute $\tilde{h}[t_0] = f(w_1 x[t_0] + w_{\rm m} h[t_0 - 1])$*
  - ↳ *$\tilde{h}[t_0]$ has fresh memory about "Julia"*
  - ↳ *Since $u[t_0] = 1$, we update as $h[t_0] = 1 \times \tilde{h}[t_0] + 0 \times h[t_0 - 1] = \tilde{h}[t_0]$*
  - ↳ *RNN has a fresh memory about "Julia"*

Principle of Gating

At time t_0, we can say

- *We have* $x[t_0] \propto$ *"Julia" and compute* $\tilde{h}[t_0] = f(w_1 x[t_0] + w_m h[t_0 - 1])$
  - ↳ $\tilde{h}[t_0]$ *has fresh memory about "Julia"*
  - ↳ *Since* $u[t_0] = 1$, *we update as* $h[t_0] = 1 \times \tilde{h}[t_0] + 0 \times h[t_0 - 1] = \tilde{h}[t_0]$
  - ↳ *RNN has a fresh memory about "Julia"*

At the next time, i.e., $t_0 + 1$, *we have*

- *No matter what* $\tilde{h}[t_0 + 1]$ *is we have* $u[t_0 + 1] = 0$
  - ↳ *We update as* $h[t_0 + 1] = 0 \times \tilde{h}[t_0 + 1] + 1 \times h[t_0] = h[t_0] = \tilde{h}[t_0]$
  - ↳ *We still have a fresh memory about "Julia"*

Principle of Gating

At time t_0, we can say

- *We have $x[t_0] \propto$ "Julia" and compute $\tilde{h}[t_0] = f(w_1 x[t_0] + w_m h[t_0 - 1])$*
  - ↳ *$\tilde{h}[t_0]$ has fresh memory about "Julia"*
  - ↳ *Since $u[t_0] = 1$, we update as $h[t_0] = 1 \times \tilde{h}[t_0] + 0 \times h[t_0 - 1] = \tilde{h}[t_0]$*
  - ↳ *RNN has a fresh memory about "Julia"*

At the next time, i.e., $t_0 + 1$, we have

- *No matter what $\tilde{h}[t_0 + 1]$ is we have $u[t_0 + 1] = 0$*
  - ↳ *We update as $h[t_0 + 1] = 0 \times \tilde{h}[t_0 + 1] + 1 \times h[t_0] = h[t_0] = \tilde{h}[t_0]$*
  - ↳ *We still have a fresh memory about "Julia"*

This repeats from $t_0 + 1$ till t, so at time t

- *No matter what $\tilde{h}[t]$, we have $u[t] = 0$*
  - ↳ *We update as $h[t] = 0 \times \tilde{h}[t] + 1 \times h[t - 1] = h[t - 1] = \ldots = \tilde{h}[t_0]$*
  - ↳ *We still have a fresh memory about "Julia"*

Principle of Gating: *Updating via Gates*

$u[t]$ *gates the memory: it decides how much memory we should pass and forget*

- *We update* $h[t] = u[t]\tilde{h}[t] + (1 - u[t])\, h[t-1]$

Principle of Gating: *Updating via Gates*

$u[t]$ *gates the memory: it decides how much memory we should pass and forget*

- *We update* $h[t] = u[t]\tilde{h}[t] + (1 - u[t])\, h[t-1]$

\+ *How does it help with vanishing gradient?*

– Well! *It is implicitly making skip connections through time*

Principle of Gating: *Updating via Gates*

$u[t]$ *gates the memory: it decides how much memory we should pass and forget*

- *We update* $h[t] = u[t]\tilde{h}[t] + (1 - u[t])\, h[t-1]$

+ *How does it help with vanishing gradient?*

– Well! *It is implicitly making skip connections through time*

Recall that *with standard BPTT, we have for* $i = t, t-1, \dots, t_0$

$$\frac{\partial h[i]}{\partial w_{\mathrm{m}}} = \dot{f}\left(z[i]\right) \left(h[i-1] + w_{\mathrm{m}} \frac{\partial h[i-1]}{\partial w_{\mathrm{m}}} \right)$$

Principle of Gating: *Updating via Gates*

$u[t]$ *gates the memory: it decides how much memory we should pass and forget*

- *We update* $h[t] = u[t]\tilde{h}[t] + (1 - u[t])\, h[t-1]$

+ *How does it help with vanishing gradient?*

– Well! *It is implicitly making skip connections through time*

Recall that *with standard BPTT, we have for* $i = t, t-1, \dots, t_0$

$$\frac{\partial h[i]}{\partial w_{\mathrm{m}}} = \dot{f}\left(z[i]\right)\left(h[i-1] + w_{\mathrm{m}} \frac{\partial h[i-1]}{\partial w_{\mathrm{m}}}\right)$$

But, now we skip multiple time slots, as we have for $i = t, t-1, \dots, t_0$

$$\frac{\partial h[i]}{\partial w_{\mathrm{m}}} = \frac{\partial h[i-1]}{\partial w_{\mathrm{m}}}$$

Principle of Gating: *A Generic Gate*

+ *Sounds inspiring! But, how could you get* $u[t]$*?*

[5] We are going to drop bias and you all know why!

Principle of Gating: *A Generic Gate*

+ *Sounds inspiring! But, how could you get* $u[t]$*?*
– Well! *Like what we did the whole time: we* learn *it!*

[5] We are going to drop bias and you all know why!

Principle of Gating: *A Generic Gate*

+ *Sounds inspiring! But, how could you get* $u[t]$*?*
- Well! *Like what we did the whole time: we learn it!*

Gate

Let $\mathbf{x}[t]$ *be input and* $\mathbf{h}[t-1]$ *be last hidden state: a gate* $\mathbf{\Gamma}[t]$ *is computed as*

$$\mathbf{\Gamma}[t] = \sigma\left(\mathbf{W}_{\Gamma,\mathrm{in}}\mathbf{x}[t] + \mathbf{W}_{\Gamma,\mathrm{m}}\mathbf{h}[t-1] + \mathbf{b}_{\Gamma}\right)$$

where $\sigma(\cdot)$ *is sigmoid function, and* $\mathbf{W}_{\Gamma,\mathrm{in}}$, $\mathbf{W}_{\Gamma,\mathrm{m}}$ *and* $\mathbf{b}_{\Gamma}$ *are learnable*[5]

[5] We are going to drop bias and you all know why!

Principle of Gating: *A Generic Gate*

+ *Sounds inspiring! But, how could you get $u[t]$?*
– Well! *Like what we did the whole time: we learn it!*

Gate

Let $\mathbf{x}[t]$ be input and $\mathbf{h}[t-1]$ be last hidden state: a gate $\mathbf{\Gamma}[t]$ is computed as

$$\mathbf{\Gamma}[t] = \sigma\left(\mathbf{W}_{\Gamma,\mathrm{in}}\mathbf{x}[t] + \mathbf{W}_{\Gamma,\mathrm{m}}\mathbf{h}[t-1] + \mathbf{b}_{\Gamma}\right)$$

where $\sigma(\cdot)$ is sigmoid function, and $\mathbf{W}_{\Gamma,\mathrm{in}}$, $\mathbf{W}_{\Gamma,\mathrm{m}}$ and $\mathbf{b}_{\Gamma}$ are learnable[5]

+ *Why do we use sigmoid function?*
- Simply because *it is between 0 and 1*
+ *What should we set the dimension of $\mathbf{\Gamma}[t]$?*
- Same as the *variable* (*memory component*) *that we want to gate*

[5] We are going to drop bias and you all know why!

Practical Gated Architectures

There are various *gated* architectures: *we look into two of them*

1. *Gated Recurrent Unit (GRU)*
2. *Long Short-Term Memory (LSTM)*

Practical Gated Architectures

There are various *gated* architectures: *we look into two of them*

1. *Gated Recurrent Unit (GRU)*
2. *Long Short-Term Memory (LSTM)*

Before we start, let's recall their basic RNN counterpart

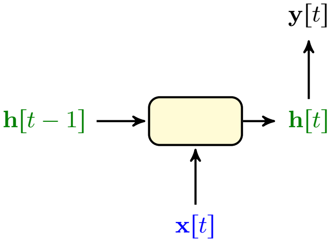
Basic RNN Counterpart

Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with an initial hidden state*
   ↳ *we can learn* $\mathbf{h}[0]$
2. *Compute memory as* $\mathbf{h}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-1])$
   ↳ *we can learn* $\mathbf{W}_1$ *and* $\mathbf{W}_\mathrm{m}$
3. *Compute output* $\mathbf{y}[t] = f_\mathrm{out}(\mathbf{W}_2\mathbf{h}[t])$ $\rightsquigarrow$ f_out and f could be different
   ↳ *we can learn* $\mathbf{W}_2$

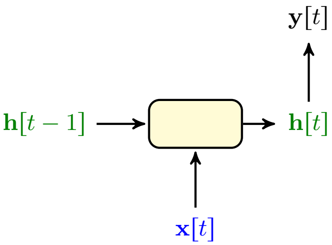
Classical Diagram: *Hidden Layer as Unit*

When we study a gated architecture: *it is common to look at the hidden layer as a unit which takes some inputs and returns some outputs*



Classical Diagram: *Hidden Layer as Unit*

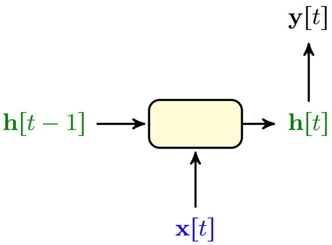
When we study a gated architecture: *it is common to look at the hidden layer as a unit which takes some inputs and returns some outputs*



We are mainly interested on this block: we want to know that given last state and new input

Classical Diagram: *Hidden Layer as Unit*

When we study a gated architecture: *it is common to look at the hidden layer as a unit which takes some inputs and returns some outputs*

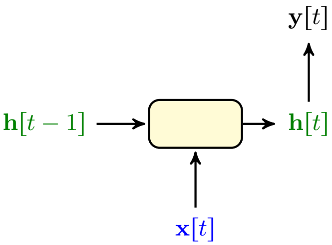


We are mainly interested on this block: we want to know that given last state and new input

1. *How does this unit update hidden state?*
2. *What components are passed to the next time interval?*

Classical Diagram: *Hidden Layer as Unit*

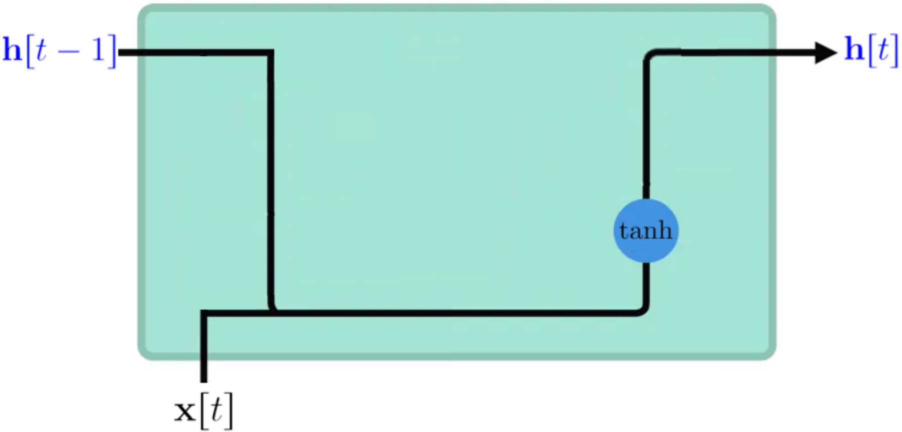
When we study a gated architecture: *it is common to look at the hidden layer as a unit which takes some inputs and returns some outputs*



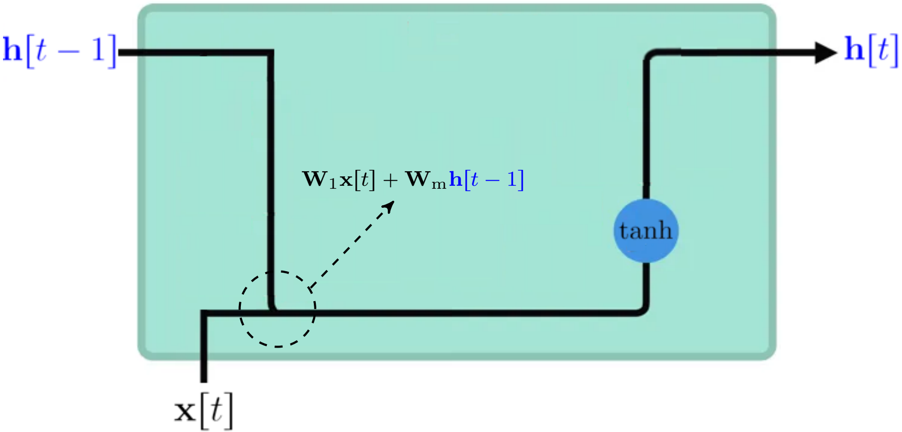
We are mainly interested on this block: we want to know that given last state and new input

1. *How does this unit update hidden state?*
2. *What components are passed to the next time interval?*
   - ↳ *Here, we have only* $\mathbf{h}[t]$
   - ↳ *But, we may have other components*
     - ↳ *We will see it in LSTM*

Classical Diagram: *Basic RNN*



Classical Diagram: *Basic RNN*



Practical Gated Architectures: *Gated Recurrent Unit*

Gated Recurrent Unit (GRU)

Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with an initial hidden state*
2. *Compute update gate* $\boldsymbol{\Gamma}_{\mathrm{u}}[t] = \sigma\left(\mathbf{W}_{\mathrm{u,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{u,m}}\mathbf{h}[t-1]\right)$
3. *Compute reset gate* $\boldsymbol{\Gamma}_{\mathrm{r}}[t] = \sigma\left(\mathbf{W}_{\mathrm{r,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{r,m}}\mathbf{h}[t-1]\right)$

Practical Gated Architectures: *Gated Recurrent Unit*

Gated Recurrent Unit (GRU)

Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with an initial hidden state*
2. *Compute update gate* $\mathbf{\Gamma}_{\rm u}[t] = \sigma\left(\mathbf{W}_{\rm u,in}\mathbf{x}[t] + \mathbf{W}_{\rm u,m}\mathbf{h}[t-1]\right)$
3. *Compute reset gate* $\mathbf{\Gamma}_{\rm r}[t] = \sigma\left(\mathbf{W}_{\rm r,in}\mathbf{x}[t] + \mathbf{W}_{\rm r,m}\mathbf{h}[t-1]\right)$
4. *Compute actual memory* $\tilde{\mathbf{h}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\rm m}\mathbf{\Gamma}_{\rm r}[t] \odot \mathbf{h}[t-1])$

Practical Gated Architectures: *Gated Recurrent Unit*

Gated Recurrent Unit (GRU)

Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with an initial hidden state*
2. *Compute update gate* $\mathbf{\Gamma}_{\rm u}[t] = \sigma\left(\mathbf{W}_{\rm u,in}\mathbf{x}[t] + \mathbf{W}_{\rm u,m}\mathbf{h}[t-1]\right)$
3. *Compute reset gate* $\mathbf{\Gamma}_{\rm r}[t] = \sigma\left(\mathbf{W}_{\rm r,in}\mathbf{x}[t] + \mathbf{W}_{\rm r,m}\mathbf{h}[t-1]\right)$
4. *Compute actual memory* $\tilde{\mathbf{h}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\rm m}\mathbf{\Gamma}_{\rm r}[t] \odot \mathbf{h}[t-1])$
5. *Update hidden state as* $\mathbf{h}[t] = (1 - \mathbf{\Gamma}_{\rm u}[t]) \odot \mathbf{h}[t-1] + \mathbf{\Gamma}_{\rm u}[t] \odot \tilde{\mathbf{h}}[t]$

Practical Gated Architectures: *Gated Recurrent Unit*

Gated Recurrent Unit (GRU)

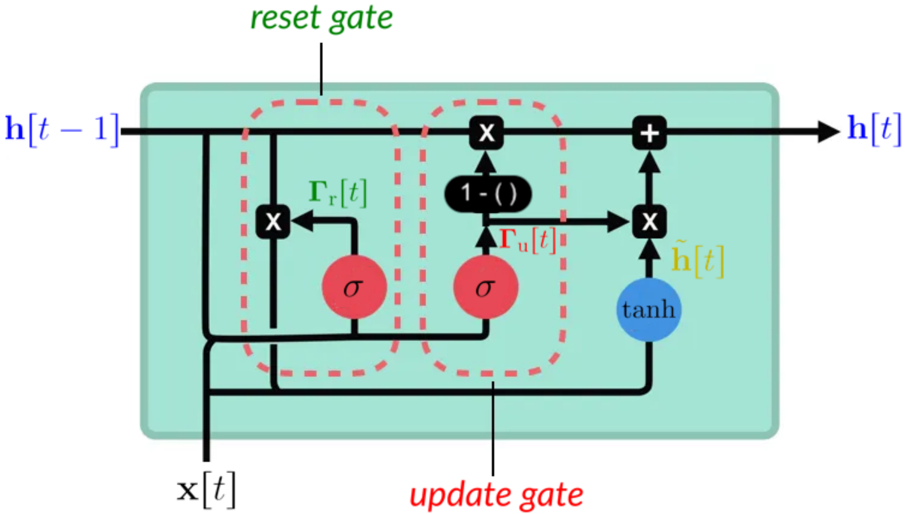
Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with an initial hidden state*
2. *Compute update gate* $\mathbf{\Gamma}_{\rm u}[t] = \sigma\left(\mathbf{W}_{\rm u,in}\mathbf{x}[t] + \mathbf{W}_{\rm u,m}\mathbf{h}[t-1]\right)$
3. *Compute reset gate* $\mathbf{\Gamma}_{\rm r}[t] = \sigma\left(\mathbf{W}_{\rm r,in}\mathbf{x}[t] + \mathbf{W}_{\rm r,m}\mathbf{h}[t-1]\right)$
4. *Compute actual memory* $\tilde{\mathbf{h}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\rm m}\mathbf{\Gamma}_{\rm r}[t] \odot \mathbf{h}[t-1])$
5. *Update hidden state as* $\mathbf{h}[t] = (1 - \mathbf{\Gamma}_{\rm u}[t]) \odot \mathbf{h}[t-1] + \mathbf{\Gamma}_{\rm u}[t] \odot \tilde{\mathbf{h}}[t]$
6. *Compute output* $\mathbf{y}[t] = f_{\rm out}(\mathbf{W}_2\mathbf{h}[t])$ $\rightsquigarrow$ $f_{\rm out}$ and f could be different

Or we could give $\mathbf{h}[t]$ *to a new layer: for instance a new GRU whose input is* $\mathbf{h}[t]$ *and has its own state*

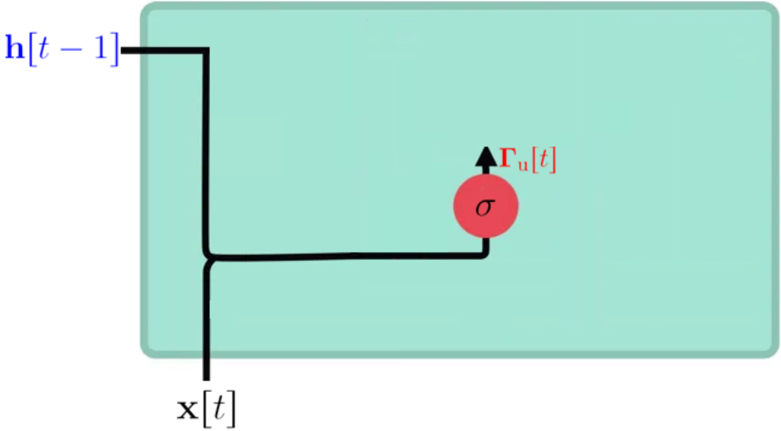
Practical Gated Architectures: *GRU*

This is what's going on in a GRU cell



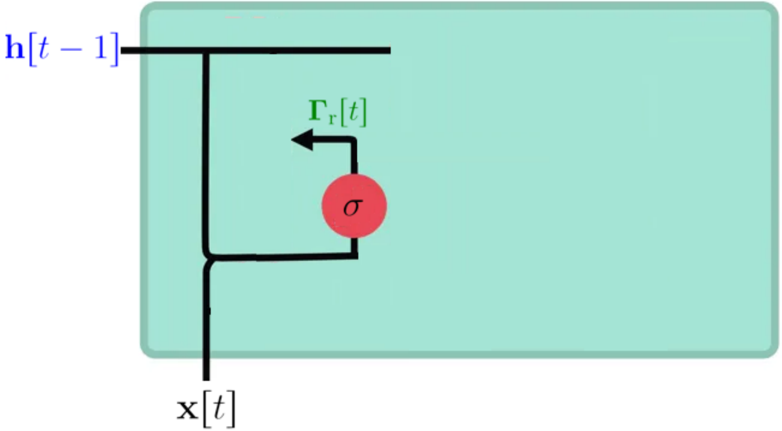
Practical Gated Architectures: *GRU*

Compute update gate $\mathbf{\Gamma}_{\rm u}[t] = \sigma\left(\mathbf{W}_{\rm u,in}\mathbf{x}[t] + \mathbf{W}_{\rm u,m}\mathbf{h}[t-1]\right)$



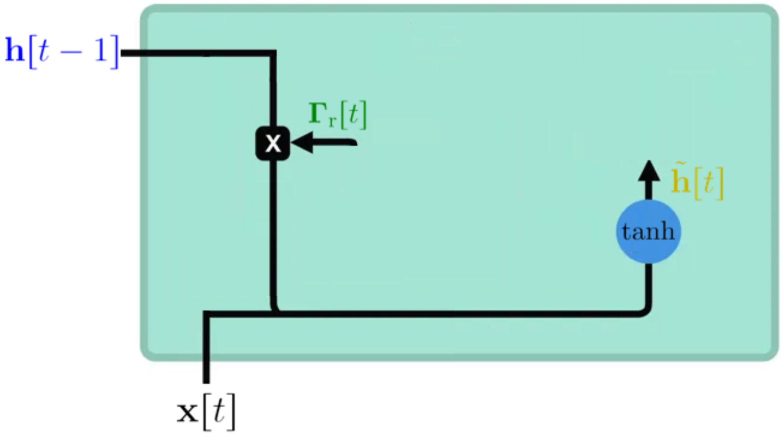
Practical Gated Architectures: *GRU*

Compute reset gate $\boldsymbol{\Gamma}_{\rm r}[t] = \sigma\left(\mathbf{W}_{\rm r,in}\mathbf{x}[t] + \mathbf{W}_{\rm r,m}\mathbf{h}[t-1]\right)$



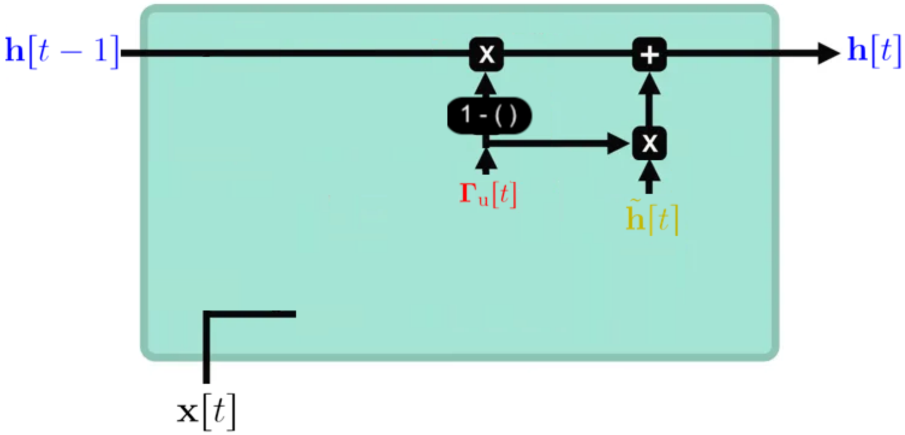
Practical Gated Architectures: *GRU*

Compute actual memory $\tilde{\mathbf{h}}[t] = f(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_\mathrm{m} \boldsymbol{\Gamma}_\mathrm{r}[t] \odot \mathbf{h}[t-1])$



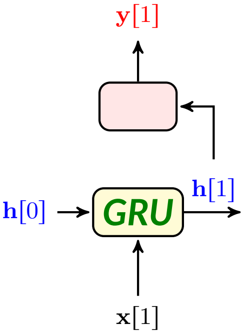
Practical Gated Architectures: *GRU*

Update hidden state as $\mathbf{h}[t] = (1 - \mathbf{\Gamma}_{\mathrm{u}}[t]) \odot \mathbf{h}[t-1] + \mathbf{\Gamma}_{\mathrm{u}}[t] \odot \tilde{\mathbf{h}}[t]$



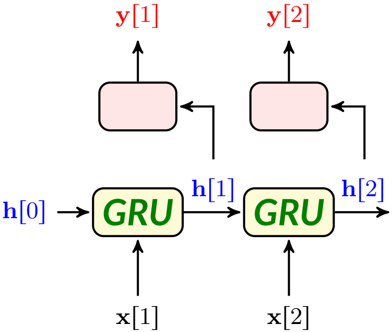
GRU: *Forward Pass*

Starting from an initial state: GRU applies the first 5 steps each time



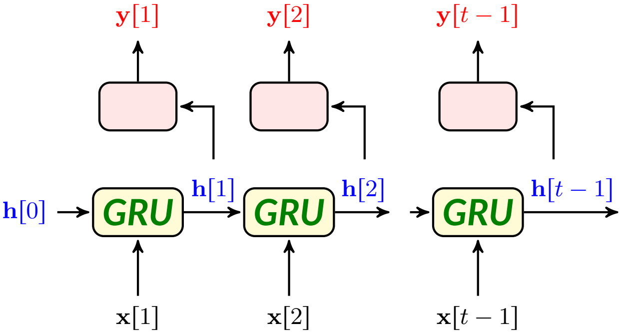
GRU: *Forward Pass*

Starting from an initial state: GRU applies the first 5 steps each time



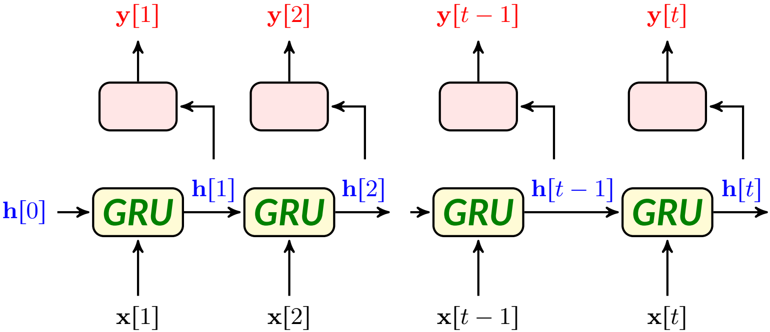
GRU: *Forward Pass*

Starting from an initial state: GRU applies the first 5 steps each time



GRU: *Forward Pass*

Starting from an initial state: GRU applies the first 5 steps each time



GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$:

GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$*

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. *We know that* $\mathbf{h}[t] = (1 - \mathbf{\Gamma}_{\mathrm{u}}[t]) \odot \mathbf{h}[t-1] + \mathbf{\Gamma}_{\mathrm{u}}[t] \odot \tilde{\mathbf{h}}[t]$

$$\nabla_{\mathbf{W}} \mathbf{h}[t] =$$

GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. *We know that* $\mathbf{h}[t] = (1 - \boldsymbol{\Gamma}_{\mathrm{u}}[t]) \odot \mathbf{h}[t-1] + \boldsymbol{\Gamma}_{\mathrm{u}}[t] \odot \tilde{\mathbf{h}}[t]$

$$\nabla_{\mathbf{W}} \mathbf{h}[t] = \nabla_{\boldsymbol{\Gamma}_{\mathrm{u}}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \boldsymbol{\Gamma}_{\mathrm{u}}[t]$$

GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. *We know that* $\mathbf{h}[t] = (1 - \mathbf{\Gamma}_{\mathrm{u}}[t]) \odot \mathbf{h}[t-1] + \mathbf{\Gamma}_{\mathrm{u}}[t] \odot \tilde{\mathbf{h}}[t]$

$$\nabla_{\mathbf{W}} \mathbf{h}[t] = \nabla_{\mathbf{\Gamma}_{\mathrm{u}}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \mathbf{\Gamma}_{\mathrm{u}}[t] + \nabla_{\mathbf{h}[t-1]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t-1]$$

GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. *We know that* $\mathbf{h}[t] = (1 - \mathbf{\Gamma}_{\mathrm{u}}[t]) \odot \mathbf{h}[t-1] + \mathbf{\Gamma}_{\mathrm{u}}[t] \odot \tilde{\mathbf{h}}[t]$

$$\begin{aligned} \nabla_{\mathbf{W}} \mathbf{h}[t] = & \nabla_{\mathbf{\Gamma}_{\mathrm{u}}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \mathbf{\Gamma}_{\mathrm{u}}[t] + \nabla_{\mathbf{h}[t-1]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t-1] \\ & + \nabla_{\tilde{\mathbf{h}}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \tilde{\mathbf{h}}[t] \end{aligned}$$

GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. *We know that* $\mathbf{h}[t] = (1 - \mathbf{\Gamma}_{\mathrm{u}}[t]) \odot \mathbf{h}[t-1] + \mathbf{\Gamma}_{\mathrm{u}}[t] \odot \tilde{\mathbf{h}}[t]$

$$\begin{aligned}\nabla_{\mathbf{W}} \mathbf{h}[t] =& \nabla_{\mathbf{\Gamma}_{\mathrm{u}}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \mathbf{\Gamma}_{\mathrm{u}}[t] + \nabla_{\mathbf{h}[t-1]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t-1] \\ &+ \nabla_{\tilde{\mathbf{h}}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \tilde{\mathbf{h}}[t]\end{aligned}$$

3. $\ldots$

Practical Gated Architectures: *Long Short-Term Memory*

Long Short-Term Memory (LSTM)

Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with initial hidden state and cell state*
2. *Compute forget gate* $\boldsymbol{\Gamma}_{\rm f}[t] = \sigma\left(\mathbf{W}_{\rm f,in}\mathbf{x}[t] + \mathbf{W}_{\rm f,m}\mathbf{h}[t-1]\right)$
3. *Compute input gate* $\boldsymbol{\Gamma}_{\rm i}[t] = \sigma\left(\mathbf{W}_{\rm i,in}\mathbf{x}[t] + \mathbf{W}_{\rm i,m}\mathbf{h}[t-1]\right)$
4. *Compute output gate* $\boldsymbol{\Gamma}_{\rm o}[t] = \sigma\left(\mathbf{W}_{\rm o,in}\mathbf{x}[t] + \mathbf{W}_{\rm o,m}\mathbf{h}[t-1]\right)$

Practical Gated Architectures: *Long Short-Term Memory*

Long Short-Term Memory (LSTM)

Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with initial hidden state and cell state*
2. *Compute forget gate* $\mathbf{\Gamma}_{\rm f}[t] = \sigma\left(\mathbf{W}_{\rm f,in}\mathbf{x}[t] + \mathbf{W}_{\rm f,m}\mathbf{h}[t-1]\right)$
3. *Compute input gate* $\mathbf{\Gamma}_{\rm i}[t] = \sigma\left(\mathbf{W}_{\rm i,in}\mathbf{x}[t] + \mathbf{W}_{\rm i,m}\mathbf{h}[t-1]\right)$
4. *Compute output gate* $\mathbf{\Gamma}_{\rm o}[t] = \sigma\left(\mathbf{W}_{\rm o,in}\mathbf{x}[t] + \mathbf{W}_{\rm o,m}\mathbf{h}[t-1]\right)$
5. *Compute actual cell state* $\tilde{\mathbf{c}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\rm m}\mathbf{h}[t-1])$
6. *Update cell state as* $\mathbf{c}[t] = \mathbf{\Gamma}_{\rm f}[t]\mathbf{c}[t-1] + \mathbf{\Gamma}_{\rm i}[t] \odot \tilde{\mathbf{c}}[t]$

Practical Gated Architectures: *Long Short-Term Memory*

Long Short-Term Memory (LSTM)

Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with initial hidden state and cell state*
2. *Compute forget gate* $\mathbf{\Gamma}_{\mathrm{f}}[t] = \sigma\left(\mathbf{W}_{\mathrm{f,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{f,m}}\mathbf{h}[t-1]\right)$
3. *Compute input gate* $\mathbf{\Gamma}_{\mathrm{i}}[t] = \sigma\left(\mathbf{W}_{\mathrm{i,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{i,m}}\mathbf{h}[t-1]\right)$
4. *Compute output gate* $\mathbf{\Gamma}_{\mathrm{o}}[t] = \sigma\left(\mathbf{W}_{\mathrm{o,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{o,m}}\mathbf{h}[t-1]\right)$
5. *Compute actual cell state* $\tilde{\mathbf{c}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\mathrm{m}}\mathbf{h}[t-1])$
6. *Update cell state as* $\mathbf{c}[t] = \mathbf{\Gamma}_{\mathrm{f}}[t]\mathbf{c}[t-1] + \mathbf{\Gamma}_{\mathrm{i}}[t] \odot \tilde{\mathbf{c}}[t]$
7. *Update hidden state as* $\mathbf{h}[t] = \mathbf{\Gamma}_{\mathrm{o}}[t] \odot f\left(\mathbf{c}[t]\right)$

Practical Gated Architectures: *Long Short-Term Memory*

Long Short-Term Memory (LSTM)

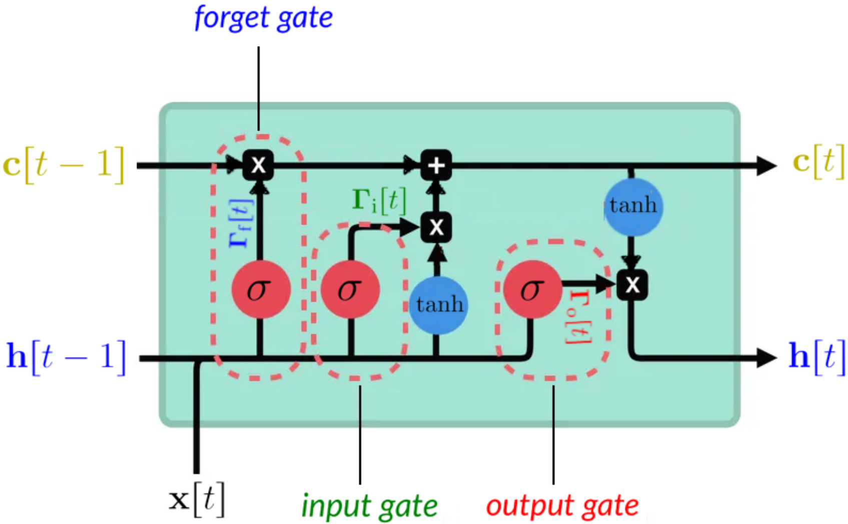
Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with initial hidden state and cell state*
2. *Compute forget gate* $\mathbf{\Gamma}_{\mathrm{f}}[t] = \sigma\left(\mathbf{W}_{\mathrm{f,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{f,m}}\mathbf{h}[t-1]\right)$
3. *Compute input gate* $\mathbf{\Gamma}_{\mathrm{i}}[t] = \sigma\left(\mathbf{W}_{\mathrm{i,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{i,m}}\mathbf{h}[t-1]\right)$
4. *Compute output gate* $\mathbf{\Gamma}_{\mathrm{o}}[t] = \sigma\left(\mathbf{W}_{\mathrm{o,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{o,m}}\mathbf{h}[t-1]\right)$
5. *Compute actual cell state* $\tilde{\mathbf{c}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\mathrm{m}}\mathbf{h}[t-1])$
6. *Update cell state as* $\mathbf{c}[t] = \mathbf{\Gamma}_{\mathrm{f}}[t]\mathbf{c}[t-1] + \mathbf{\Gamma}_{\mathrm{i}}[t] \odot \tilde{\mathbf{c}}[t]$
7. *Update hidden state as* $\mathbf{h}[t] = \mathbf{\Gamma}_{\mathrm{o}}[t] \odot f\left(\mathbf{c}[t]\right)$
8. *Compute output* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2\mathbf{h}[t])$ $\rightsquigarrow$ f_{out} and f could be different

Or we could give $\mathbf{h}[t]$ *to a new layer: for instance a new LSTM whose input is* $\mathbf{h}[t]$ *and has its own hidden and cell states*

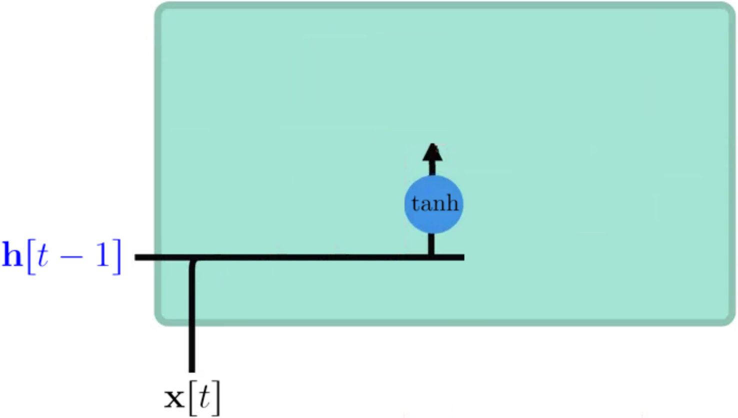
Practical Gated Architectures: *LSTM*

This is how inside an LSTM unit looks like



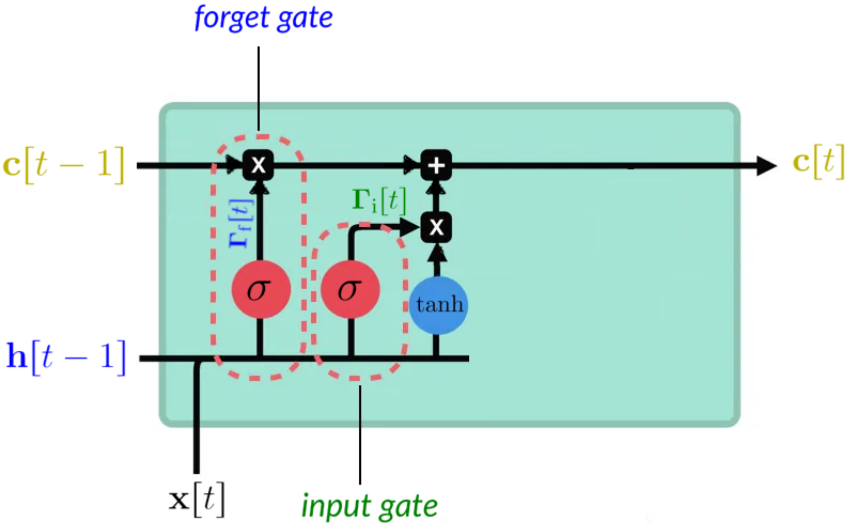
Practical Gated Architectures: *LSTM*

Actual cell state $\tilde{\mathbf{c}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-1])$



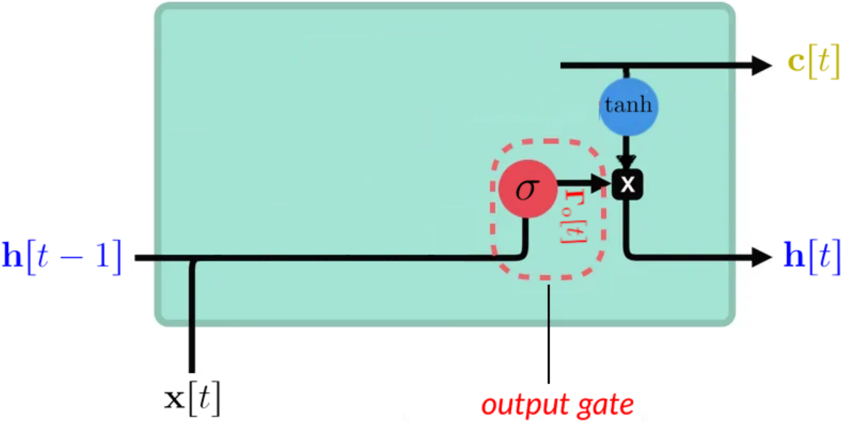
Practical Gated Architectures: *LSTM*

We use forget gate and update gate to update cell state



Practical Gated Architectures: *LSTM*

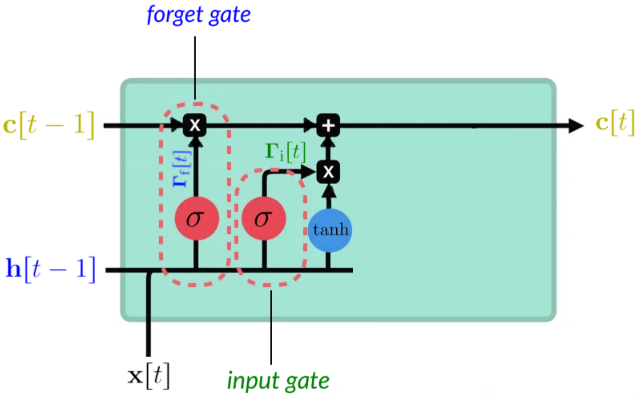
We use output gate to control fellow of memory to the hidden state



Practical Gated Architectures: *LSTM*

Intuitively, the gates in LSTM impact *the flow of information as follows*

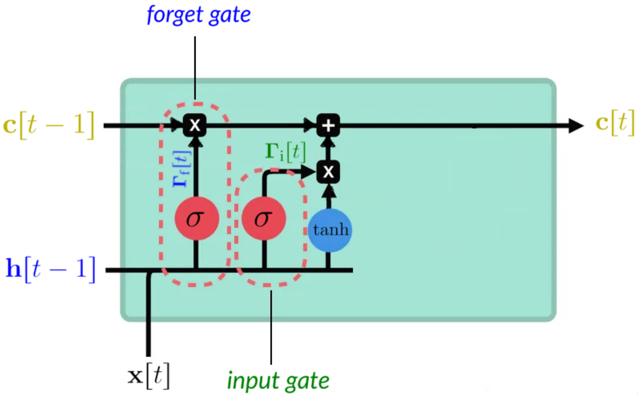
- *Forget gate controls how much we forget from last state*
  ↳ *Assume* $\mathbf{\Gamma}_{\mathrm{f}}[t] = \mathbf{0}$*: then, we remember nothing of* $\mathbf{c}[t-1]$



Practical Gated Architectures: *LSTM*

Intuitively, the gates in LSTM impact *the flow of information as follows*

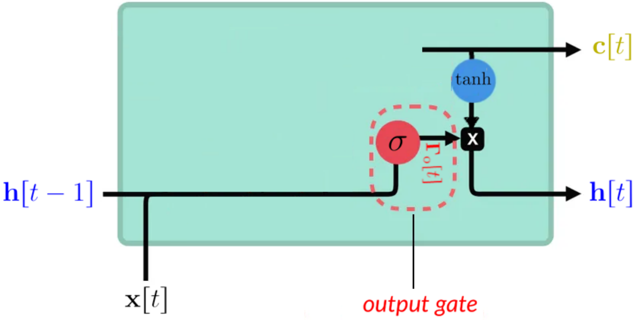
- *Forget gate controls how much we forget from last state*
  - ↳ *Assume* $\mathbf{\Gamma}_{\rm f}[t] = \mathbf{0}$*: then, we remember nothing of* $\mathbf{c}[t-1]$
- *Input gate controls how much we remember from new cell state*
  - ↳ *Assume* $\mathbf{\Gamma}_{\rm i}[t] = \mathbf{0}$*: then, we remember nothing of* $\tilde{\mathbf{c}}[t]$



Practical Gated Architectures: *LSTM*

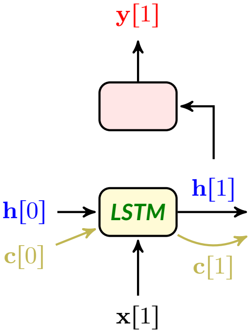
Intuitively, the gates in LSTM impact *the flow of information as follows*

- *Output gate controls how much we let from updated state to go out*
  - ↳ *Assume* $\boldsymbol{\Gamma}_0[t] = \mathbf{0}$: *then, we send nothing of* $\mathbf{c}[t]$ *out*



LSTM: *Forward Pass*

Starting from initial hidden and cell state: LSTM passes forward as

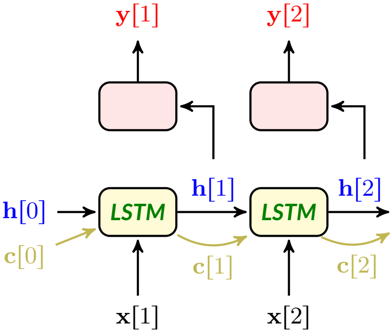


Pay Attention

Note that unlike other architectures, LSTM does not keep all memory inside hidden state but it carries it also in cell state. This state is only for memory and is not directly used by higher layers, e.g., output layer of the NN

LSTM: *Forward Pass*

Starting from initial hidden and cell state: LSTM passes forward as

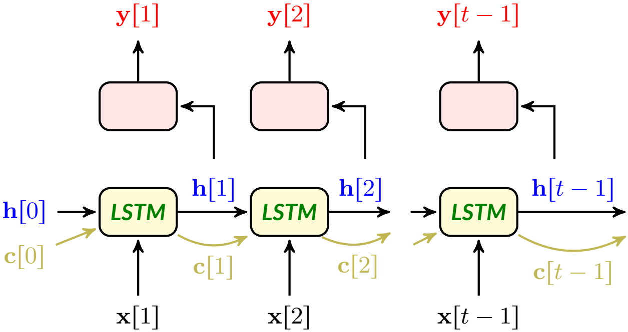


Pay Attention

Note that unlike other architectures, LSTM does not keep all memory inside hidden state but it carries it also in cell state. This state is only for memory and is not directly used by higher layers, e.g., output layer of the NN

LSTM: *Forward Pass*

Starting from initial hidden and cell state: LSTM passes forward as

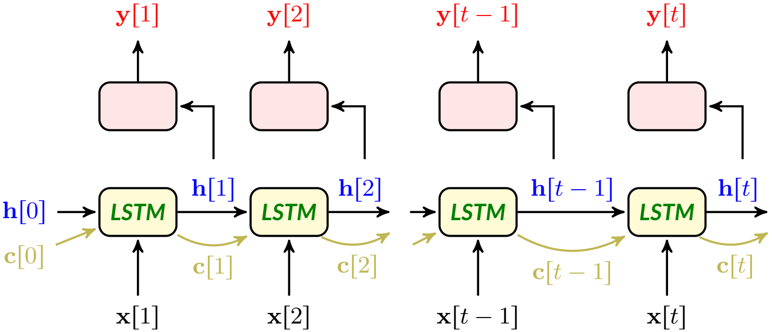


Pay Attention

Note that unlike other architectures, LSTM does not keep all memory inside hidden state but it carries it also in cell state. This state is only for memory and is not directly used by higher layers, e.g., output layer of the NN

LSTM: *Forward Pass*

Starting from initial hidden and cell state: LSTM passes forward as



Pay Attention

Note that unlike other architectures, LSTM does not keep all memory inside hidden state but it carries it also in cell state. This state is only for memory and is not directly used by higher layers, e.g., output layer of the NN

LSTM: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside LSTM, e.g., $\mathbf{W}_{\mathrm{i,m}}$:

LSTM: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside LSTM, e.g., $\mathbf{W}_{\mathrm{i,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

LSTM: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside LSTM, e.g., $\mathbf{W}_{\mathrm{i,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$*

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

LSTM: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside LSTM, e.g., $\mathbf{W}_{\mathrm{i,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. ...

LSTM: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside LSTM, e.g., $\mathbf{W}_{\mathrm{i,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. $\ldots$

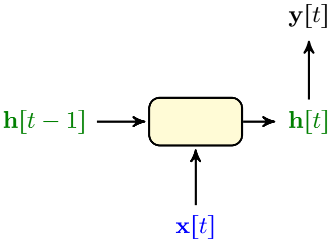
Suggestion

Try writing it once to see the impact of gates!

Bidirectional RNNs

We have up to now considered *unidirectional* RNNs

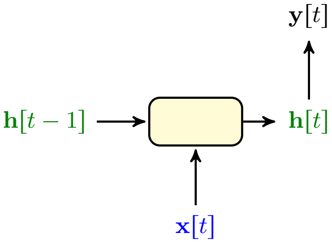
we start from beginning of the sequence and move in one direction



Bidirectional RNNs

We have up to now considered *unidirectional* RNNs

we start from beginning of the sequence and move in one direction



But, can't we learn from future input as well?

Bidirectional RNNs

Future entries can have information about past: *say our RNN wants to fill the empty field*

... the color $\boxed{}$ that many people assume is the color of sun ...

Obviously, future input in the sequence is helping in this example!

Bidirectional RNNs

Future entries can have information about past: *say our RNN wants to fill the empty field*

... the color $\boxed{}$ *that many people assume is the color of sun ...*

Obviously, future input in the sequence is helping in this example!

+ *But, how can we get information from future?*
- Well, we have the whole sequence: *we could move once from left to right and once from right to left*

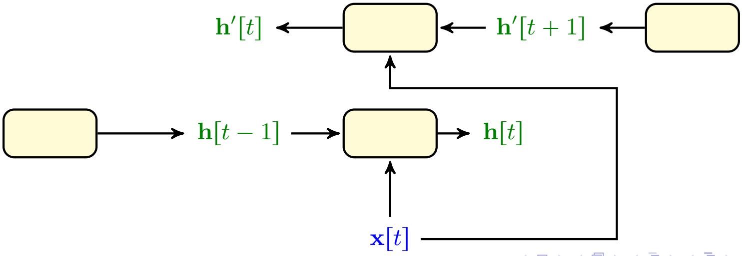
Bidirectional RNNs

Bidirectional RNNs

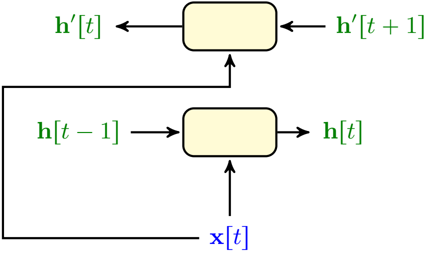
A bidirectional RNN (BRNN) consists of two RNNs

- *one that starts with an initial hidden state at* $t = 0$ *and computes* $\mathbf{h}[t]$ *from* $\mathbf{h}[t-1]$ *and* $\mathbf{x}[t]$
- *another that starts with an initial hidden state at* $t = T + 1$ *and computes* $\mathbf{h}'[t]$ *from* $\mathbf{h}'[t+1]$ *and* $\mathbf{x}[t]$

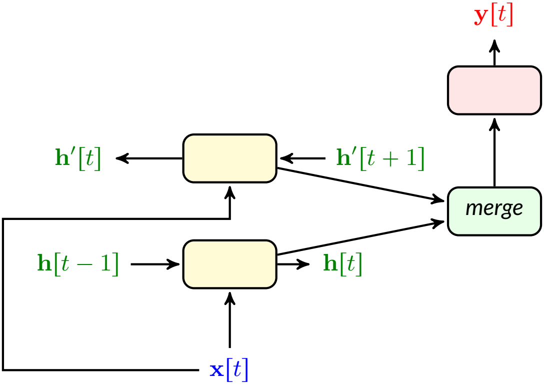
Output at time t *is determined from merged version of the two hidden states*



Bidirectional RNNs

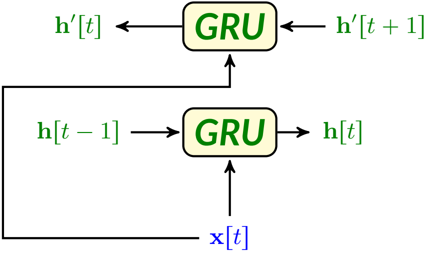


Bidirectional RNNs



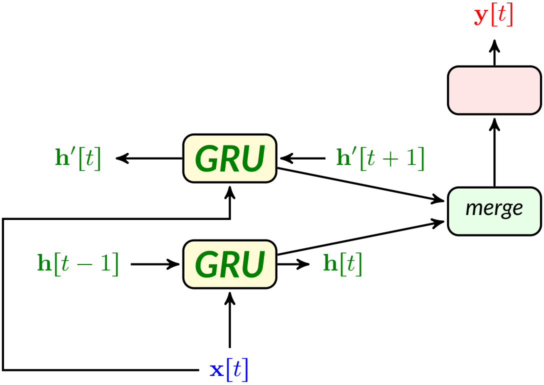
+ *What exactly is this merge block?*
– It gets the two states and *returns a vector that matches output layer*

Bidirectional GRU



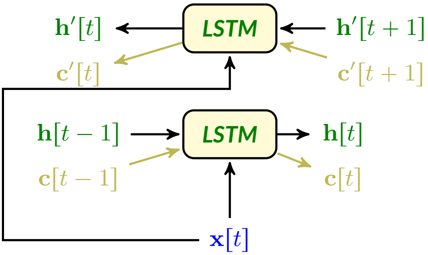
+ *Should we use any RNN here?*
– Sure! *We may use GRU*

Bidirectional GRU



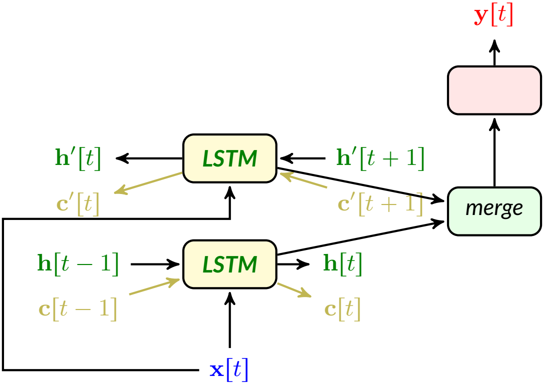
+ *Should we use any RNN here?*
– Sure! *We may use GRU*

Bidirectional LSTM



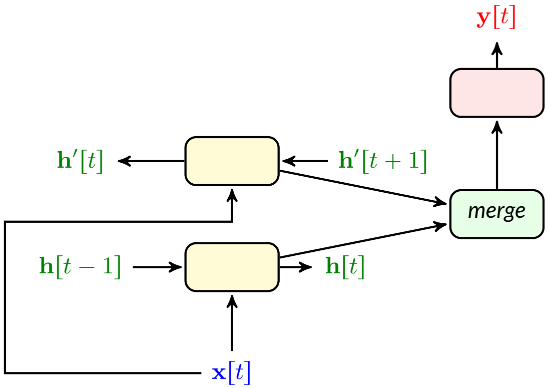
+ *Should we use any RNN here?*
– Sure! *We may use LSTM*

Bidirectional LSTM



+ *Should we use any RNN here?*
– Sure! *We may use LSTM*

RNNs in PyTorch



+ *Any suggestion for merging the hidden states?*
– Sure! Let's see some code

RNNs in PyTorch

In PyTorch, we can *access a basic RNN in* `torch.nn` *module*

```
torch.nn.RNN()
```

We can make it deep by simply choosing `num_layers` *more than one and bidirectional by setting* `bidirectional` *to* `True`.

RNNs in PyTorch

In PyTorch, we can *access a basic RNN in* `torch.nn` *module*

```
torch.nn.RNN()
```

We can make it deep by simply choosing `num_layers` *more than one and bidirectional by setting* `bidirectional` *to* `True`. *Same with GRU and LSTM*

```
torch.nn.LSTM()
torch.nn.GRU()
```

RNNs in PyTorch

In PyTorch, we can *access a basic RNN in* `torch.nn` *module*

```
torch.nn.RNN()
```

We can make it deep by simply choosing `num_layers` *more than one and bidirectional by setting* `bidirectional` *to* `True`. *Same with GRU and LSTM*

```
torch.nn.LSTM()
torch.nn.GRU()
```

In bidirectional case, we get access to both states. To merge them, we could

- *add the two states*
- *average them*
- *concatenate them, i.e.,* $\mathbf{h}_{\mathrm{c}}[t] = (\mathbf{h}[t], \mathbf{h}'[t])$

or do any other operation that we find useful

Computing Loss: *Challenge*

We mentioned several times in this chapter that *we assume*

we can compute the loss between RNN's output sequence and *label sequence*

However, it is in general a challenge!

Computing Loss: *Challenge*

We mentioned several times in this chapter that *we assume*

we can compute the loss between RNN's output sequence and label sequence

However, it is in general a challenge!

+ *Why is it a challenge? We did it easily in FNN and CNN chapters!*
– Because the problem there *was already properly segmented!*
+ *What do you mean by segmented?*
– Let's break it down!

Computing Loss: *Motivating Example*

Let's consider a simple example: *we have an image that includes a sequence of handwritten digits, e.g.,*

- *The sequence includes five digits*
- *Each digit is either 1, 2, 3, or 4*

Computing Loss: *Motivating Example*

Let's consider a simple example: *we have an image that includes a sequence of handwritten digits, e.g.,*

- *The sequence includes five digits*
- *Each digit is either 1, 2, 3, or 4*

Our task is to recognize this sequence, i.e., return the five digits in correct order

- *This is a classification task*
- *How can we do it? We use NNs*
  - ↳ *We train an NN over lots of images: we have lots of images of sequence of digits*
  - ↳ *We then use it to recognize new images*

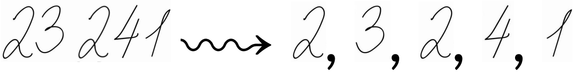
$\rightsquigarrow$ 23241

Let's say we are going to use a CNN

Computing Loss: *Motivating Example*

To use a CNN, we need to specify our input size

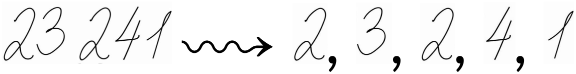
- *We segment an input image into a sequence of five images*
  - ↳ *These images are all as large as CNN's input size*



Computing Loss: *Motivating Example*

To use a CNN, we need to specify our input size

- *We segment an input image into a sequence of five images*
  - ↳ *These images are all as large as CNN's input size*



- *We label each image with its label, e.g.,*  is *labeled as 2*

Computing Loss: *Motivating Example*

To use a CNN, we need to specify our input size

- *We segment an input image into a sequence of five images*
  - ↳ *These images are all as large as CNN's input size*

- *We label each image with its label, e.g.,*  *is labeled as 2*
- *We give these five images to our CNN and get five outputs*
  - ↳ *Assume we use softmax at the output layer*
  - ↳ *For each image, we get a vector of size 4 as output*
    - ↳ *Each entry represents probability of image being one of digits 1, 2, 3, and 4*

Computing Loss: *Motivating Example*

To use a CNN, we need to specify our input size

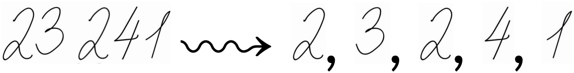
- *We segment an input image into a sequence of five images*
  - ↳ *These images are all as large as CNN's input size*

- *We label each image with its label, e.g.,*  *is labeled as 2*
- *We give these five images to our CNN and get five outputs*
  - ↳ *Assume we use softmax at the output layer*
  - ↳ *For each image, we get a vector of size 4 as output*
    - ↳ *Each entry represents probability of image being one of digits 1, 2, 3, and 4*

Computing Loss: *Motivating Example*

To use a CNN, we need to specify our input size

- *We segment an input image into a sequence of five images*
  - ↳ *These images are all as large as CNN's input size*

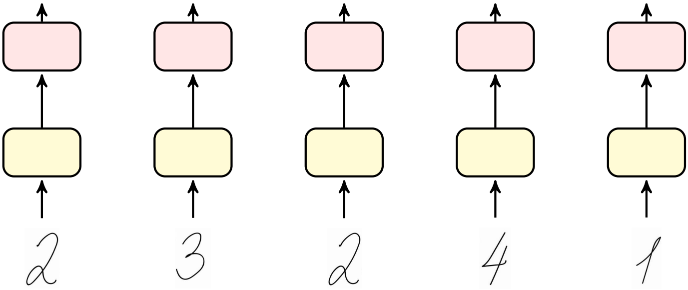


- *We label each image with its label, e.g.,*  *is labeled as 2*
- *We give these five images to our CNN and get five outputs*
  - ↳ *Assume we use softmax at the output layer*
  - ↳ *For each image, we get a vector of size 4 as output*
    - ↳ *Each entry represents probability of image being one of digits 1, 2, 3, and 4*
- *To compute loss, we compare each output with its corresponding label*

Computing Loss: *Motivating Example*

$$\mathbf{v}[1] = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{v}[2] = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{v}[3] = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{v}[4] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{v}[5] = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} y_1[1] \\ y_2[1] \\ y_3[1] \\ y_4[1] \end{bmatrix} = \mathbf{y}[1] \quad \mathbf{y}[2] \quad \mathbf{y}[3] \quad \mathbf{y}[4] \quad \mathbf{y}[5]$$



↳ $y_j[t]$ *is the probability of digit in time t being j*

Computing Loss: *Motivating Example*

Here, we already have the data segmented into

a sequence that for each time step has a label

Computing Loss: *Motivating Example*

Here, we already have the data segmented into

a sequence that for each time step has a label

So, computing loss is easy as pie!

$$\hat{R} = \mathcal{L}\left(\mathbf{y}[1], \ldots, \mathbf{y}[5], \mathbf{v}[1], \ldots, \mathbf{v}[5]\right)$$
$$= \sum_{t=1}^{5} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right) = \sum_{t=1}^{5} \hat{R}[t]$$

Computing Loss: *Motivating Example*

Here, we already have the data segmented into

a sequence that for each time step has a label

So, computing loss is easy as pie!

$$\hat{R} = \mathcal{L}\left(\mathbf{y}[1], \ldots, \mathbf{y}[5], \mathbf{v}[1], \ldots, \mathbf{v}[5]\right)$$
$$= \sum_{t=1}^{5} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right) = \sum_{t=1}^{5} \hat{R}[t]$$

When we compute gradients, we note that only $\hat{R}[t]$ depends on $\mathbf{y}[t]$: so, for a given output at time $t = i$ we can simply write

$$\nabla_{\mathbf{y}[i]} \hat{R} = \sum_{t=1}^{5} \nabla_{\mathbf{y}[i]} \hat{R}[t] = \nabla_{\mathbf{y}[i]} \hat{R}[i] = \nabla_{\mathbf{y}[i]} \mathcal{L}\left(\mathbf{y}[i], \mathbf{v}[i]\right)$$

Computing Loss: *One-to-One Correspondence*

+ *But is it practical to do segmentation by hand?*
– No! This is why we built RNNs!

Computing Loss: *One-to-One Correspondence*

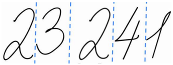
+ *But is it practical to do segmentation by hand?*
– No! This is why we built RNNs!

With RNNs, we address this learning task as bellow

- *We look at the complete image as a sequence of data*
  ↳ *We divide input into multiple equal-size frames*
- *We go over each frame separately*
  ↳ *We give the frame as the input along with previous state*
  ↳ *We compute a new state which can potentially give us the output*

Computing Loss: *One-to-One Correspondence*

If we are extremely lucky; then, our segmentation looks like this

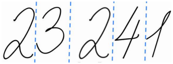


$$\rightsquigarrow \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3], \mathbf{x}[4], \mathbf{x}[5]$$

and we have a label for each time step

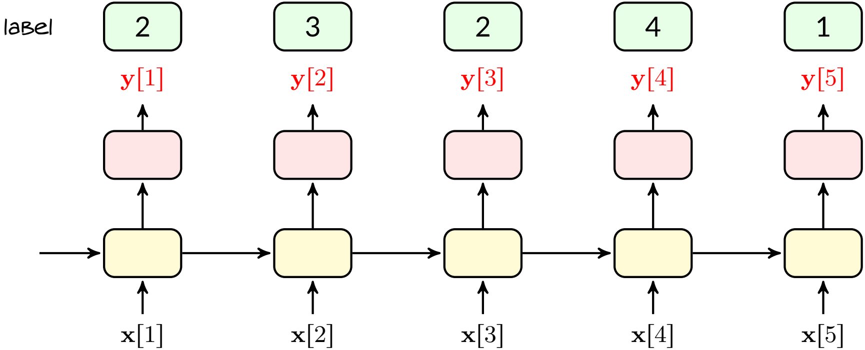
Computing Loss: *One-to-One Correspondence*

If we are extremely lucky; then, our segmentation looks like this



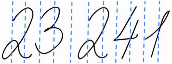
$$\rightsquigarrow \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3], \mathbf{x}[4], \mathbf{x}[5]$$

and we have a label for each time step



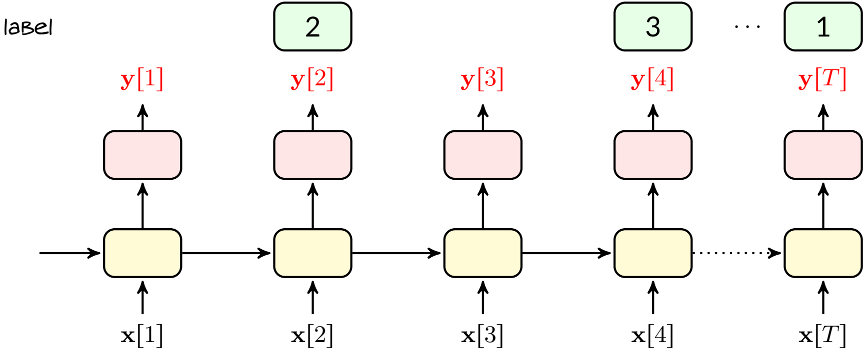
Computing Loss: *One-to-One Correspondence*

But, that's too good to happen! Usually we have



$$\rightsquigarrow \mathbf{x}[1], \mathbf{x}[2], \ldots, \mathbf{x}[T]$$

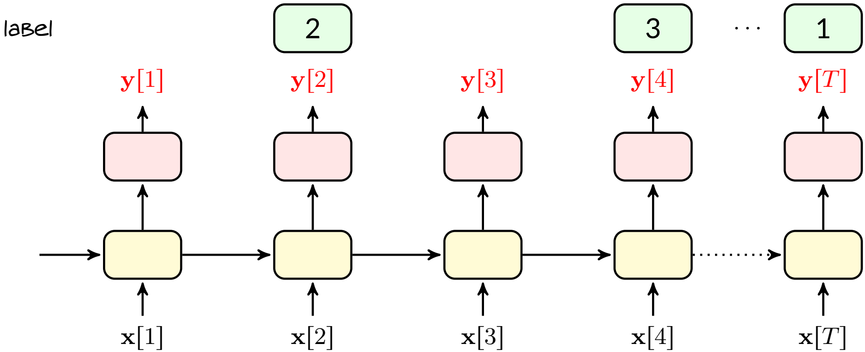
and we have a label in some time steps



Computing Loss: *One-to-One Correspondence*

In this typical case, two questions seem non-trivial

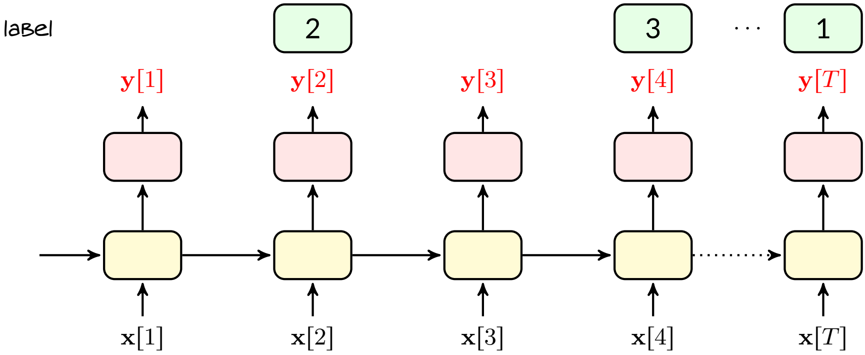
1. *Where should we put each label?* $\equiv$ *Where should we read each label?*



Computing Loss: *One-to-One Correspondence*

In this typical case, two questions seem non-trivial

1. *Where should we put each label?* $\equiv$ *Where should we read each label?*
2. *What should we do with non-labeled outputs, e.g.,* $\mathbf{y}[1]$?



Computing Loss: *One-to-One Correspondence*

The key challenge in computing the loss is that *we do not have necessarily one-to-one correspondence with sequence data*

Computing Loss: *One-to-One Correspondence*

The key challenge in computing the loss is that *we do not have necessarily one-to-one correspondence with sequence data*

Correspondence Problem

With sequence data, we could have a data-sequence of time T that is labeled by a sequence of size $K < T$ where

no time index is specified for any label in the K-long label sequence

Computing Loss: *One-to-One Correspondence*

The key challenge in computing the loss is that *we do not have necessarily one-to-one correspondence* with *sequence data*

Correspondence Problem

With sequence data, we could have a data-sequence of time T that is labeled by a sequence of size $K < T$ where

no time index is specified for any label in the K-long label sequence

Correspondence problem exists pretty much in all practical sequence data

- *In speech recognition, multiple time frames correspond to a single word*
- *In text recognition, multiple image frames correspond to a single letter*
- ...

Correspondence Problem: *Formulation*

Let's formulate the problem clearly: *Say we have*

A sequence of data

$$\mathbf{x}[1:T] = \mathbf{x}[1], \ldots, \mathbf{x}[T]$$

that is labeled with the sequence of K true labels

$$\mathbf{v}[1:K] = \mathbf{v}[1], \ldots, \mathbf{v}[K]$$

where K and T can be different

Correspondence Problem: *Formulation*

Let's formulate the problem clearly: *Say we have*

A sequence of data

$$\mathbf{x}[1:T] = \mathbf{x}[1], \dots, \mathbf{x}[T]$$

that is labeled with the sequence of K true labels

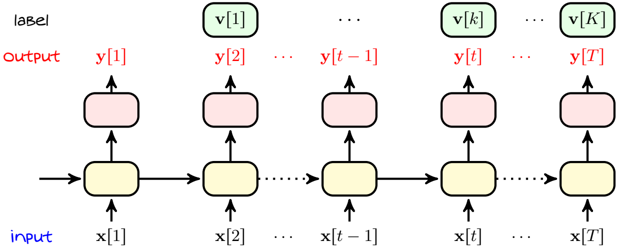
$$\mathbf{v}[1:K] = \mathbf{v}[1], \dots, \mathbf{v}[K]$$

where K and T can be different

For this setting, we want to train an RNN with this data sequence: starting with an initial state, this RNN returns an output sequence

$$\mathbf{y}[1:T] = \mathbf{y}[1], \dots, \mathbf{y}[T]$$

Correspondence Problem: *Formulation*

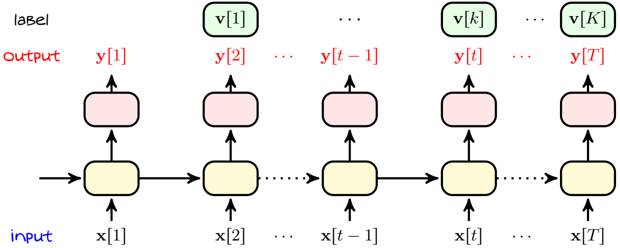


To be able to train this RNN, we need to

1. *define a loss function that computes* $\hat{R} = \mathcal{L}\left(\mathbf{y}[1:T], \mathbf{v}[1:K]\right)$
   ↳ *We need this loss function to be differentiable with respect to all outputs*

$$\nabla_{\mathbf{y}[1]} \hat{R}, \dots, \nabla_{\mathbf{y}[T]} \hat{R}$$

Correspondence Problem: *Formulation*



To use this RNN after training, i.e., for inferring, we need to

2. *know how to map outputs to predicted labels*
   ↳ *We need to extract K labels from $\mathbf{y}[1:T]$, i.e.,*

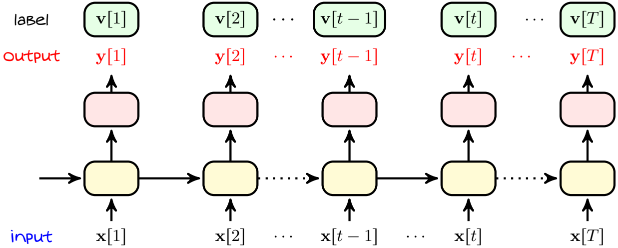
$$\mathbf{y}[1], \dots, \mathbf{y}[T] \mapsto \hat{\mathbf{v}}[1], \dots, \hat{\mathbf{v}}[K]$$

Let's look into different settings

Setting I: *Perfectly Segmented*

In some problems, *we have our data perfectly segmented*

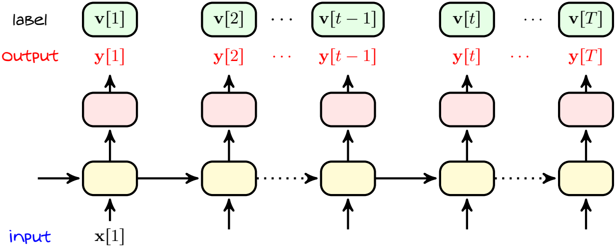
- *There is a separate label for each time step, i.e.,* $K = T$
  - ↳ *many-to-many type I and one-to-many*



Setting I: *Perfectly Segmented*

In some problems, *we have our data perfectly segmented*

- *There is a separate label for each time step, i.e.,* $K = T$
  ↳ *many-to-many type I* and *one-to-many*



Attention

We can always treat a non-existing input entry as an empty

↳ *We are good as long as we have a label at each time* t

Setting I: *Defining Loss*

In such settings, we define the loss *to be aggregated loss over time*

$$\hat{R} = \sum_{t=1}^{T} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right)$$

for some loss function $\mathcal{L}(\cdot, \cdot)$

Setting I: *Defining Loss*

In such settings, we define the loss *to be aggregated loss over time*

$$\hat{R} = \sum_{t=1}^{T} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right)$$

for some loss function $\mathcal{L}(\cdot, \cdot)$

The gradients are then trivially computed

Gradient with respect to particular output $\mathbf{y}[t]$ *is*

$$\nabla_{\mathbf{y}[t]} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R}[t] = \nabla_{\mathbf{y}[t]} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[t]\right)$$

Setting I: *Inference*

Inference in such setting is performed *by one-to-one mapping: at time* t*, we predict based on* $\mathbf{y}[t]$

$$\mathbf{y}[1] \mapsto \hat{\mathbf{v}}[1], \ldots, \mathbf{y}[T] \mapsto \hat{\mathbf{v}}[T]$$

Setting I: *Inference*

Inference in such setting is performed *by one-to-one mapping: at time t, we predict based on $\mathbf{y}[t]$*

$$\mathbf{y}[1] \mapsto \hat{\mathbf{v}}[1], \ldots, \mathbf{y}[T] \mapsto \hat{\mathbf{v}}[T]$$

For instance, assume $\mathbf{y}[t]$ is output of a softmax activation; then, we set

$$\hat{\mathbf{v}}[t] = \operatorname{argmax} \mathbf{y}[t]$$

where argmax *returns the index of the largest entry, e.g.,*

$$\operatorname{argmax} \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \\ 0 \end{bmatrix} = 2$$

Setting II: *Known Segments*

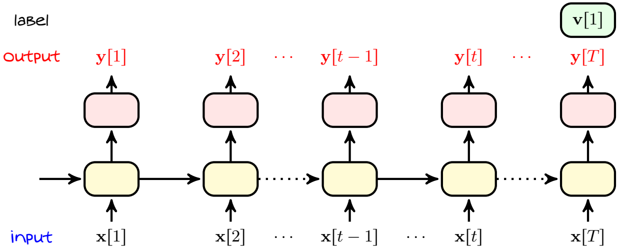
In some problems, *we have only one label for the whole sequence, i.e.,* $K = 1$

↳ *It corresponds to many-to-one type of problems*

↳ *This can be that we have really only one label, e.g., content classification*

↳ *It can be that we know the time index t at which each label is assigned*

↳ *We split data to sub-sequences with each sub-sequence having only one label*



Setting II: *Defining Loss for Dumb NN*

A naive approach to define loss is *to set it be the loss between last output and label*, i.e.,

$$\hat{R} = \mathcal{L}\left(\mathbf{y}[T], \mathbf{v}[1]\right)$$

Setting II: *Defining Loss for Dumb NN*

A naive approach to define loss is *to set it be the loss between last output and label*, i.e.,

$$\hat{R} = \mathcal{L}\left(\mathbf{y}[T], \mathbf{v}[1]\right)$$

With this loss, *gradient with respect to particular output* $\mathbf{y}[t]$ *is*

$$\nabla_{\mathbf{y}[t]} \hat{R} = \begin{cases} \nabla_{\mathbf{y}[T]} \mathcal{L}\left(\mathbf{y}[T], \mathbf{v}[1]\right) & t = T \\ 0 & t \neq T \end{cases}$$

Setting II: *Defining Loss for Dumb NN*

A naive approach to define loss is *to set it be the loss between last output and label*, i.e.,

$$\hat{R} = \mathcal{L}\left(\mathbf{y}[T], \mathbf{v}[1]\right)$$

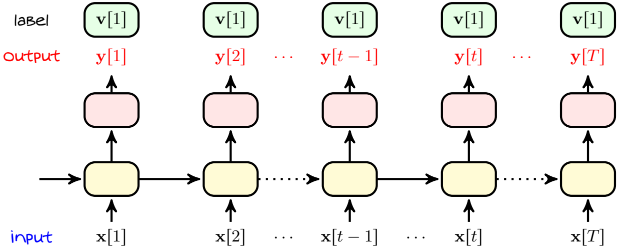
With this loss, *gradient with respect to particular output* $\mathbf{y}[t]$ *is*

$$\nabla_{\mathbf{y}[t]} \hat{R} = \begin{cases} \nabla_{\mathbf{y}[T]} \mathcal{L}\left(\mathbf{y}[T], \mathbf{v}[1]\right) & t = T \\ 0 & t \neq T \end{cases}$$

+ *But does it make sense to ignore all other outputs?*
- Not at all! *We are training a dumb NN that can respond only when it's over with the whole sequence!*

Setting II: *Loss for Smarter Training*

An extremely smart NN is the one who knows the label before the input speaks!

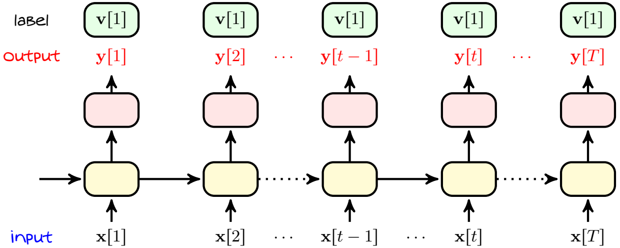


For this NN, the loss is

$$\hat{R} = \sum_{t=1}^{T} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[1]\right)$$

Setting II: *Loss for Smarter Training*

An extremely smart NN is the one who knows the label before the input speaks!



For this NN, the loss is

$$\hat{R} = \sum_{t=1}^{T} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[1]\right)$$

But, we should be careful! We should not expect NN to know everything from potentially irrelevant input!

Setting II: *Defining Proper Loss*

A realistic approach is *to define the loss via a weighted sum*, i.e.,

$$\hat{R} = \sum_{t=1}^{T} w_t \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[1]\right)$$

where w_t is the weight at time t

- *initially w_t is small*
  - ↳ *we do not expect the NN to know everything from very beginning*
- *it gradually increases up to its maximum w_T*
  - ↳ *by time T the NN should know the label*

Setting II: *Defining Proper Loss*

A realistic approach is *to define the loss via a weighted sum*, i.e.,

$$\hat{R} = \sum_{t=1}^{T} w_t \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[1]\right)$$

where w_t is the weight at time t

- *initially w_t is small*
  - ↳ *we do not expect the NN to know everything from very beginning*
- *it gradually increases up to its maximum w_T*
  - ↳ *by time T the NN should know the label*

With this loss, *gradient with respect to particular output $\mathbf{y}[t]$ is*

$$\nabla_{\mathbf{y}[t]} \hat{R} = w_t \nabla_{\mathbf{y}[t]} \hat{R}[t] = w_t \nabla_{\mathbf{y}[t]} \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[1]\right)$$

Setting II: *Inference*

Inference in such setting is performed *by many-to-one mapping: we only predict based on* $\mathbf{y}[1:T]$

$$\mathbf{y}[1:T] \mapsto \hat{\mathbf{v}}[1]$$

Setting II: *Inference*

Inference in such setting is performed *by many-to-one mapping: we only predict based on* $\mathbf{y}[1:T]$

$$\mathbf{y}[1:T] \mapsto \hat{\mathbf{v}}[1]$$

For instance, assume $\mathbf{y}[t]$ *is output of a softmax activation; then, we set*

$$\tilde{\mathbf{v}}[t] = \operatorname{argmax} \mathbf{y}[t]$$

and then take a (potentially weighted) majority vote: $\hat{\mathbf{v}}[1]$ *is the class that most often estimated with occurrence at each time being weighted by some weight*

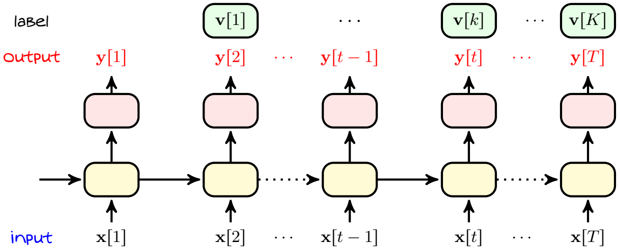
Setting III: *Unknown Segments*

Most common case is that *we have a label sequence shorter than our data*

↳ *Each label in this sequence is corresponding to a segment of input*

  ↳ *We do not know where this segment begins and where it ends*

    ↳ *There might be even no clear answer to that!*



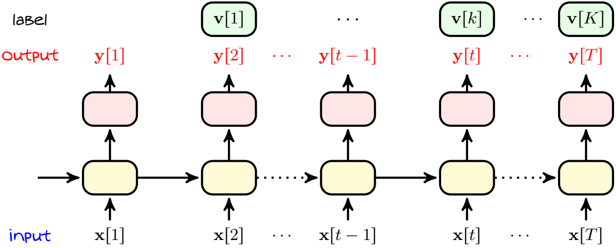
Setting III: *Unknown Segments*

Most common case is that *we have a label sequence shorter than our data*

↳ *Each label in this sequence is corresponding to a segment of input*

↳ *We do not know where this segment begins and where it ends*

↳ *There might be even no clear answer to that!*



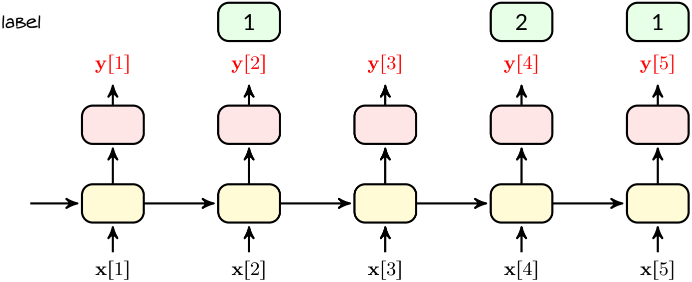
Note that we are dealing with a sequence to sequence model: we want to learn

relation between sequence $\mathbf{x}[1:T]$ and sequence $\mathbf{v}[1:K]$!

Setting II: *Example*

Assume we have image  *that is divided into a sequence of five pixel vectors*

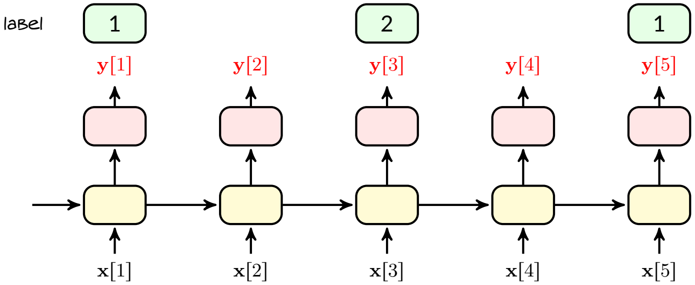
- *Since it is a training data, it is labeled as 121*
  - ↳ *We do not know after which output we should expect RNN to know first, second* or *third digit!*



Setting II: *Example*

Assume we have image  *that is divided into a sequence of five pixel vectors*

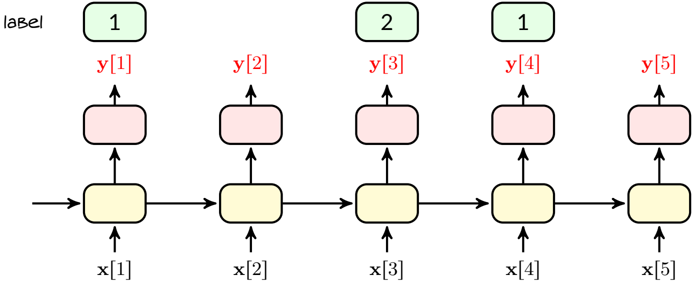
- *Since it is a training data, it is labeled as 121*
  - ↳ *We do not know after which output we should expect RNN to know first, second* or *third digit!*



Setting II: *Example*

Assume we have image  *that is divided into a sequence of five pixel vectors*

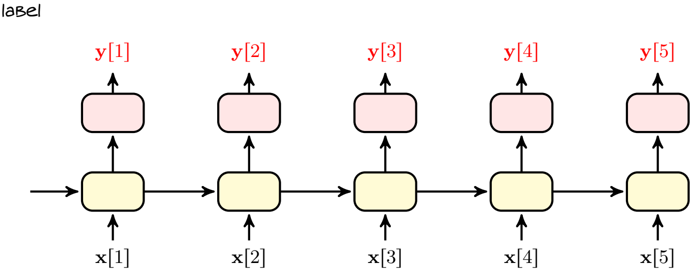
- *Since it is a training data, it is labeled as 121*
  - ↳ *We do not know after which output we should expect RNN to know first, second* or *third digit!*



Setting II: *Example*

Assume we have image  *that is divided into a sequence of five pixel vectors*

- *Since it is a training data, it is labeled as 121*
  - ↳ *We do not know after which output we should expect RNN to know first, second or third digit!*

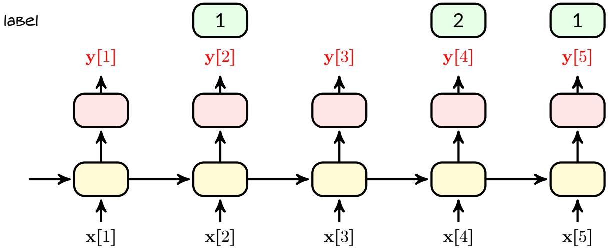


+ *Sounds impossible!*

– *Only impossible is impossible!* Let's carry on and see what we can do!

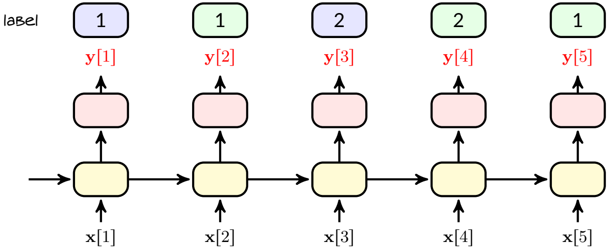
Setting II: *Genie-Defined Loss*

Assume a genie has told us end of each segment



Setting II: *Genie-Defined Loss*

Assume a genie has told us end of each segment

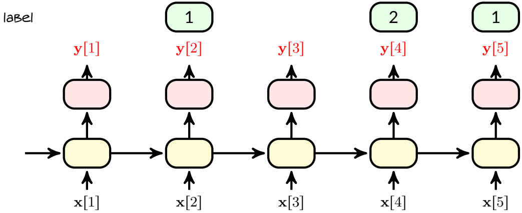


We can fill the empty labels with repetition, and then define the loss as

$$\hat{R} = \sum_{k=1}^{K} \sum_{t=i_{k-1}+1}^{i_k} w_t \mathcal{L}\left(\mathbf{y}[t], \mathbf{v}[k]\right)$$

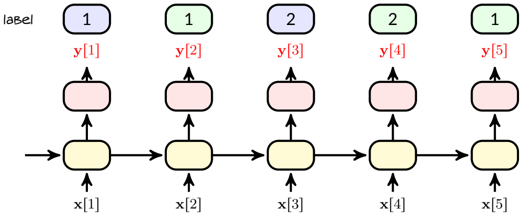
where i_k *is where label* $\mathbf{v}_k$ *ends, e.g., in above diagram* $i_1 = 2$

Setting II: *Defining Loss*



We don't have the genie:

Setting II: *Defining Loss*



We don't have the genie: we could assume that i_k is something to learn!

$$\hat{R}(\mathbf{i}) = \sum_{k=1}^{K} \sum_{t=i_{k-1}+1}^{i_k} w_t \mathcal{L}(\mathbf{y}[t], \mathbf{v}[k])$$

where $\mathbf{i} = [0, i_1 \ldots, i_K]$ is something we need to learn

Setting II: *Optimal Segmentation*

+ *How could we learn* $\mathbf{i}$*? Should we compute also* $\nabla_{\mathbf{i}} \hat{R}$*?*
- Well! You may try! *But, obviously* i_k *is an integer!*

Setting II: *Optimal Segmentation*

+ *How could we learn* $\mathbf{i}$*? Should we compute also* $\nabla_{\mathbf{i}} \hat{R}$*?*
– Well! You may try! *But, obviously* i_k *is an integer!*

Optimal Segmentation

Optimal approach for finding $\mathbf{i}$ *is to train the NN for all possible choice for* $\mathbf{i}$ *and then find the final training loss* $\hat{R}(\mathbf{i})$*. The optimal segmentation is then given by*

$$\mathbf{i}^\star = \operatorname*{argmin}_{\mathbf{i}} \hat{R}(\mathbf{i})$$

Setting II: *Optimal Segmentation*

+ *How could we learn* $\mathbf{i}$*? Should we compute also* $\nabla_{\mathbf{i}} \hat{R}$*?*
– Well! You may try! *But, obviously* i_k *is an integer!*

Optimal Segmentation

Optimal approach for finding $\mathbf{i}$ *is to train the NN for all possible choice for* $\mathbf{i}$ *and then find the final training loss* $\hat{R}(\mathbf{i})$*. The optimal segmentation is then given by*

$$\mathbf{i}^\star = \underset{\mathbf{i}}{\operatorname{argmin}}\, \hat{R}(\mathbf{i})$$

+ *Is it computationally feasible?*
– No! *The number of possible choice for* $\mathbf{i}$ *grows exponentially with* T*! We need to go for sub-optimal approaches*

Setting II: *Number of Possible Segmentations*

+ *How is it exponentially large?*
– Let's look at our example

In our last example, *we should assign label sequence 121 to a sequence of length 5: each entry of output sequence in this case can be labeled by 1 (the first one), 2 or 1 (the last one). This means that we have 3 choices of label for each time interval; thus, the total number of possible segmentations is around 3^5.*

$$\boxed{\text{In general number of segmentations grows exponentially with } T}$$

+ *But wait a moment! We have also counted the case of labeling all outputs with 1! This cannot be the case!*
– This is right! *It is in general much less than 3^5 but it's still exponential*

Let's see the exact possible segmentations!

Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ with a label

- *We know that the label sequence is 121*

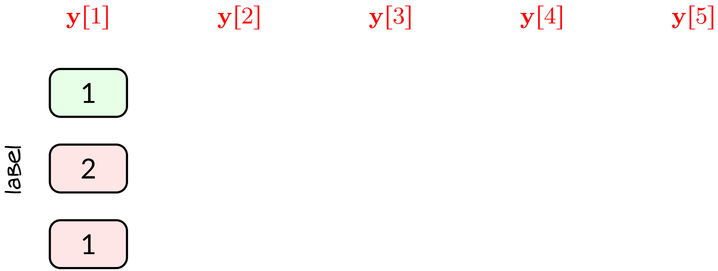
$\mathbf{y}[1]$ $\mathbf{y}[2]$ $\mathbf{y}[3]$ $\mathbf{y}[4]$ $\mathbf{y}[5]$

label

Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ *with a label*

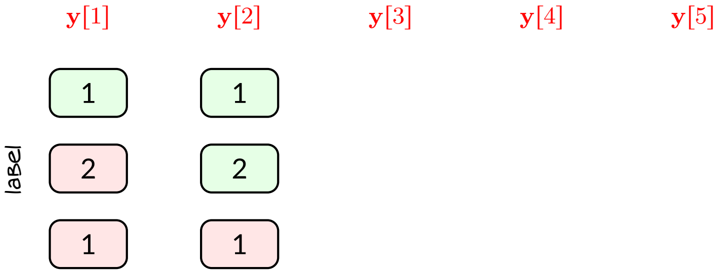
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ with a label

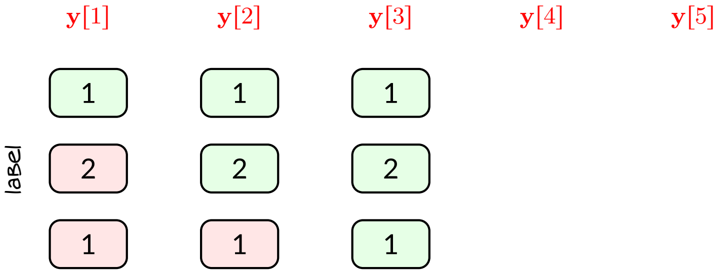
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ *with a label*

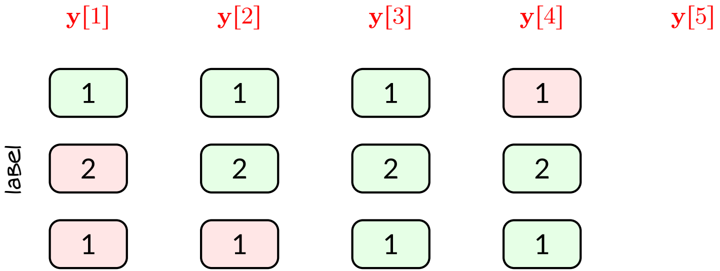
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \dots, \mathbf{y}[5]$ *with a label*

- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ with a label

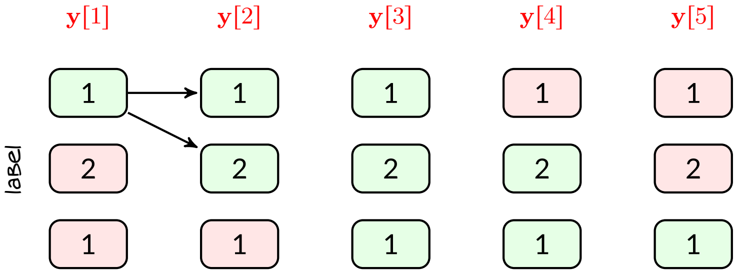
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*
  - ↳ *Last output could be only in the third segment*

label	$\mathbf{y}[1]$	$\mathbf{y}[2]$	$\mathbf{y}[3]$	$\mathbf{y}[4]$	$\mathbf{y}[5]$
	1	1	1	1	1
	2	2	2	2	2
	1	1	1	1	1

Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ with a label

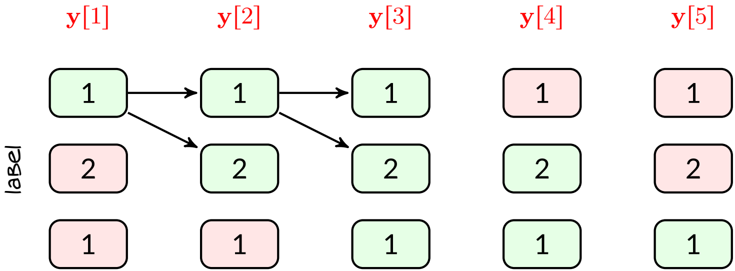
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*
  - ↳ *Last output could be only in the third segment*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ *with a label*

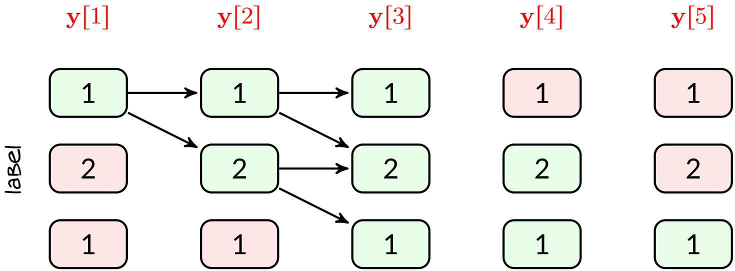
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*
  - ↳ *Last output could be only in the third segment*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ *with a label*

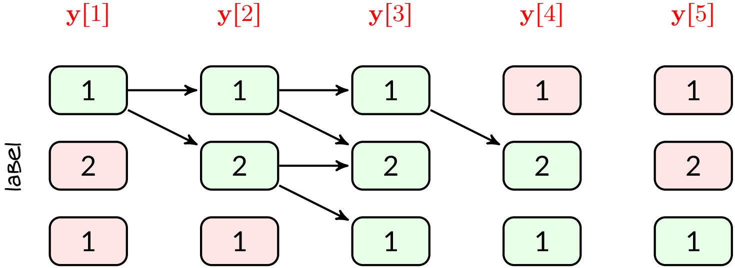
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*
  - ↳ *Last output could be only in the third segment*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ *with a label*

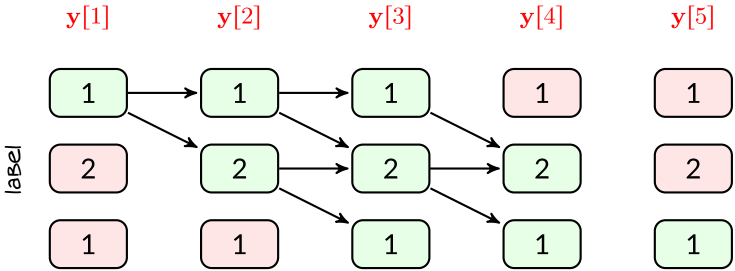
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*
  - ↳ *Last output could be only in the third segment*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \dots, \mathbf{y}[5]$ *with a label*

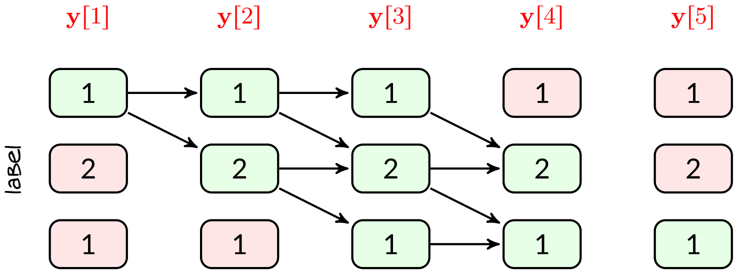
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*
  - ↳ *Last output could be only in the third segment*



Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ with a label

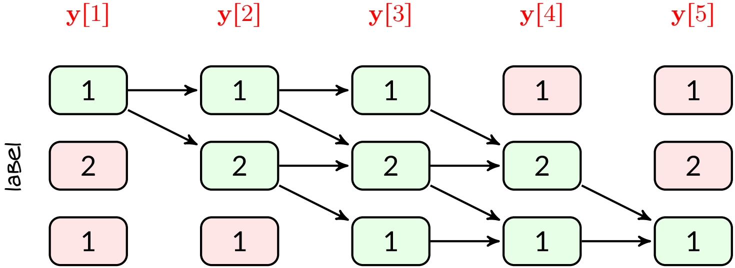
- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*
  - ↳ *Last output could be only in the third segment*



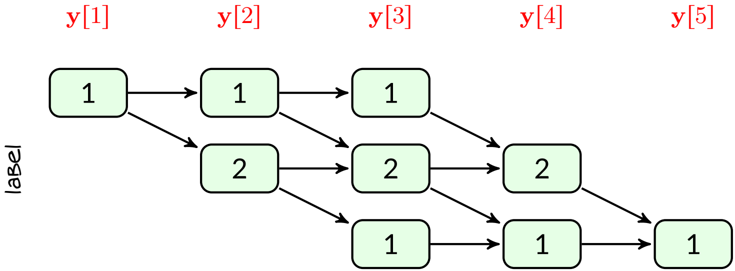
Setting II: *Number of Possible Segmentations*

We intend to compare each of $\mathbf{y}[1], \ldots, \mathbf{y}[5]$ *with a label*

- *We know that the label sequence is 121*
  - ↳ *First output is definitely in the first segment: it's label is definitely 1*
  - ↳ *Second output could be still in the first segment or in the second segment*
  - ↳ *Third output could be in the first, second, or third segment*
  - ↳ *Our labels should finish by the end of output sequence: fourth output cannot be in first segment*
  - ↳ *Last output could be only in the third segment*

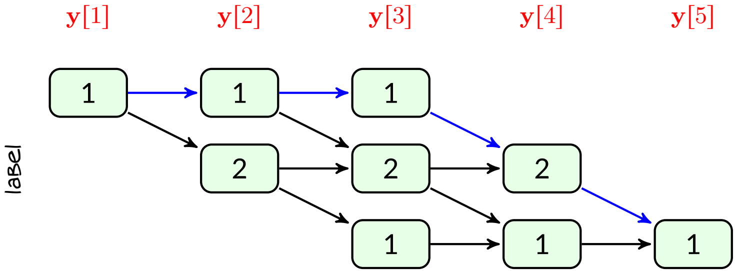


Setting II: *Showing Segmentations on Graph*



Though it's exponentially large: *we see that each segmentation corresponds to one path on this graph*

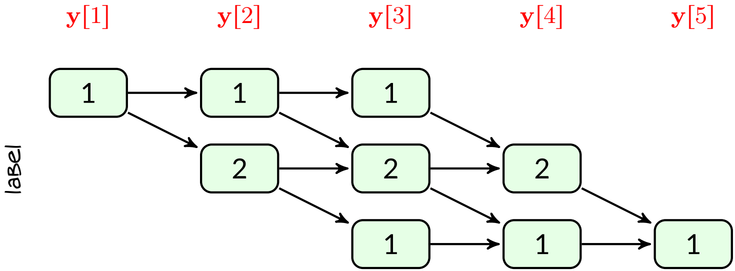
Setting II: *Showing Segmentations on Graph*



Though it's exponentially large: *we see that each segmentation corresponds to one path on this graph*

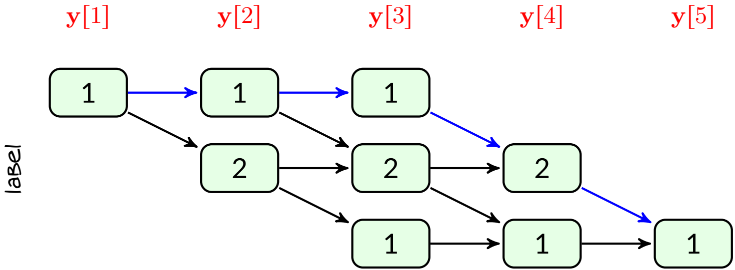
Blue path corresponds to $i_1 = 3$, $i_2 = 4$, *and* $i_3 = 5$, *i.e.*, $\mathbf{i} = [0, 3, 4, 5]$

Setting II: *Loss on Segmentation Graph*



We can compute the loss for each segmentation directly on this graph: let's say that we have L different paths on the graph. For each path, we can write an expanded label sequence, e.g.,

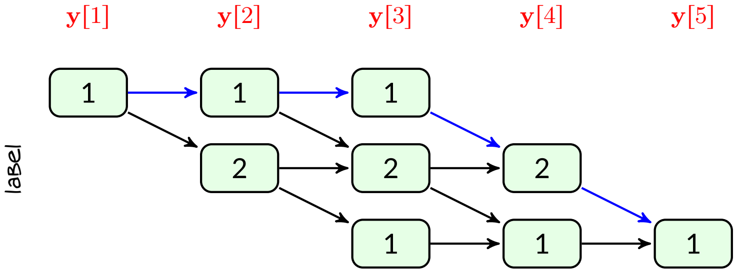
Setting II: *Loss on Segmentation Graph*



We can compute the loss for each segmentation directly on this graph: let's say that we have L different paths on the graph. For each path, we can write an expanded label sequence, e.g.,

> *Expanded label sequence of blue path is { 1, 1, 1, 2, 1}*

Setting II: *Loss on Segmentation Graph*

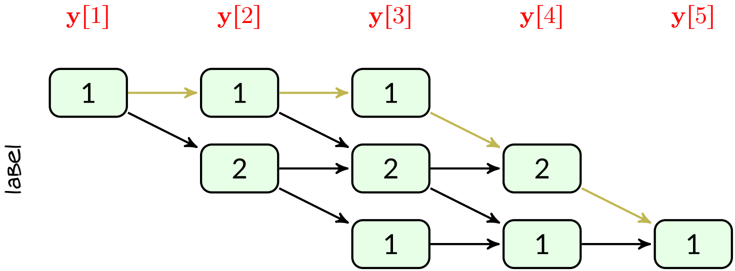


We can compute the loss for each segmentation directly on this graph: let's say that we have L different paths on the graph. For each path, we can write an expanded label sequence, e.g.,

> *Expanded label sequence of blue path is { 1, 1, 1, 2, 1}*

This sequence is of length T and we show it for path ℓ with $\tilde{\mathbf{v}}_\ell[t]$

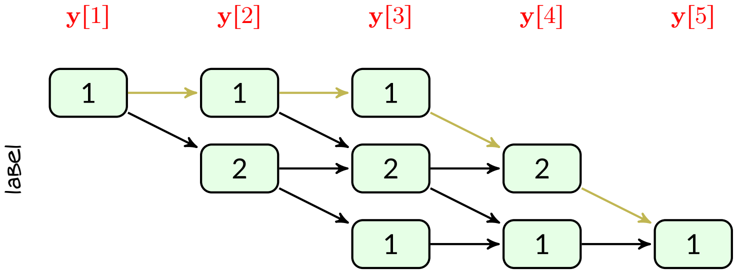
Setting II: *Loss on Segmentation Graph*



For each path $\ell = 1, \ldots, L$, the loss is computed by aggregating the losses between outputs and extended labels

$$\hat{R}_\ell = \sum_{t=1}^{T} w_t \mathcal{L}\left(\mathbf{y}[t], \tilde{\mathbf{v}}_\ell[t]\right) =$$

Setting II: *Loss on Segmentation Graph*



For each path $\ell = 1, \ldots, L$, the loss is computed by aggregating the losses between outputs and extended labels

$$\hat{R}_\ell = \sum_{t=1}^{T} w_t \mathcal{L}\left(\mathbf{y}[t], \tilde{\mathbf{v}}_\ell[t]\right) = \sum_{t=1}^{T} \hat{R}_\ell[t]$$

It again decomposes into sum of T terms with only one being function of $\mathbf{y}[t]$

Setting II: *Optimal Segmentation on Graph*

We can represent the optimal segmentation on the graph as below

```
OptimalSegmentTraining():
```

1: Initiate with $\hat{R} = +\infty$ and some random $\ell^\star = \varnothing$
2: **for** $\ell = 1, \ldots, L$ **do**
3: Let the loss be $\hat{R}_\ell$
4: Train for sufficient epochs
5: **if** *After training* $\hat{R}_\ell < \hat{R}$ **then**
6: $\hat{R} \leftarrow \hat{R}_\ell$ and $\ell^\star \leftarrow \ell$
7: **end if**
8: **end for**
9: Return learnable parameters and $\ell^\star$

\+ *Say we could be over with this infeasible training! How do we use the trained RNN for inference?*

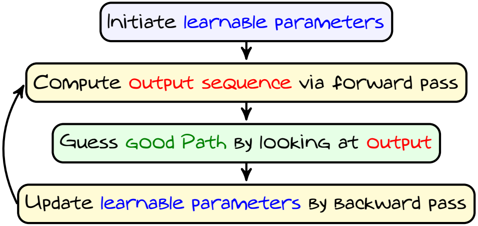
– In this case, *we have* $\ell^\star$ *which gives us optimal segmentation: we infer label of each segment based on its corresponding outputs*

Setting II: *Maximum-Likelihood Segmentation*

Since optimal segmentation is infeasible, people uses *maximum-likelihood approach* that is *well-known in detection and coding theory*

Maximum-Likelihood Segmentation

Start with an initial guess for optimal path on segmentation graph and do one step of training; then, improve the guess based on the outputs of next forward pass and go for next step of training



Let's look at its pseudo-code

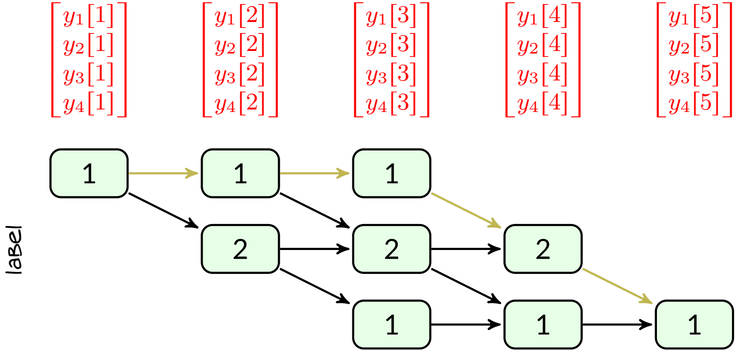
Setting II: *Maximum-Likelihood Segmentation*

`MaxLikelihoodTraining():`

1: **for** Iteration $i = 1, \ldots, I$ **do**
2: Pass forward through time: Compute output sequence $\mathbf{y}[1:T]$
3: Compute $p(\tilde{\mathbf{v}}_\ell[1:T] | \ell)$ for each path ℓ on segmentation graph ⊛
4: Update $\ell^* = \operatorname{argmax}_\ell p(\tilde{\mathbf{v}}_\ell[1:T] | \ell)$ ⊛
5: Set loss to $\hat{R}_{\ell^\star}$ and backpropagate over RNN
6: Update learnable parameters
7: **end for**
8: Return learnable parameters and ℓ^*

+ *Why we call it maximum likelihood?*
– Because we *guess* path *by maximizing the likelihood* $p(\tilde{\mathbf{v}}_\ell[1:T] | \ell)$
+ *But how can find likelihood of a path?*
– We can use output sequence $\mathbf{y}[1:T]$

Setting II: *Finding Likelihood on Segmentation Graph*

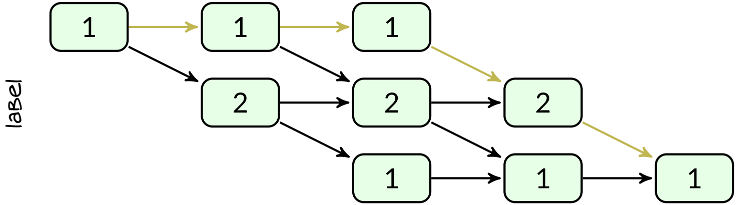


Assume that each label could be 1, 2, 3, or 4: *at each time t the RNN returns a 4-dimensional vector whose entries are probability of each class*

we can multiply the probabilities of classes on the path

Setting II: *Finding Likelihood on Segmentation Graph*

$$\begin{bmatrix} y_1[1] \\ y_2[1] \\ y_3[1] \\ y_4[1] \end{bmatrix} \quad \begin{bmatrix} y_1[2] \\ y_2[2] \\ y_3[2] \\ y_4[2] \end{bmatrix} \quad \begin{bmatrix} y_1[3] \\ y_2[3] \\ y_3[3] \\ y_4[3] \end{bmatrix} \quad \begin{bmatrix} y_1[4] \\ y_2[4] \\ y_3[4] \\ y_4[4] \end{bmatrix} \quad \begin{bmatrix} y_1[5] \\ y_2[5] \\ y_3[5] \\ y_4[5] \end{bmatrix}$$

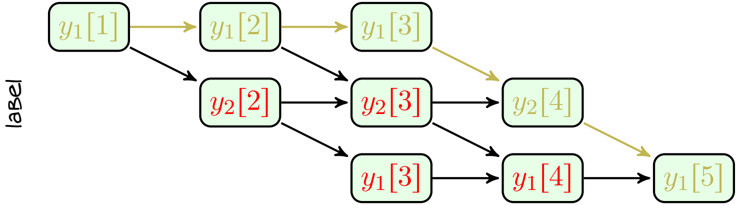


For instance, the yellow path has a likelihood

$$p\left(\tilde{\mathbf{v}}_\ell[1:T] | \ell\right) = \prod_{t=1}^{T} p\left(\tilde{\mathbf{v}}_\ell[t] | \ell\right) = y_1[1] y_1[2] y_1[3] y_2[4] y_1[5]$$

Setting II: *Finding Likelihood on Segmentation Graph*

$$\begin{bmatrix} y_1[1] \\ y_2[1] \\ y_3[1] \\ y_4[1] \end{bmatrix} \quad \begin{bmatrix} y_1[2] \\ y_2[2] \\ y_3[2] \\ y_4[2] \end{bmatrix} \quad \begin{bmatrix} y_1[3] \\ y_2[3] \\ y_3[3] \\ y_4[3] \end{bmatrix} \quad \begin{bmatrix} y_1[4] \\ y_2[4] \\ y_3[4] \\ y_4[4] \end{bmatrix} \quad \begin{bmatrix} y_1[5] \\ y_2[5] \\ y_3[5] \\ y_4[5] \end{bmatrix}$$



Or better to say: we just put output entries in graph and move on the path

$$p\left(\tilde{\mathbf{v}}_\ell[1:T]|\ell\right) = \prod_{t=1}^{T} y_{\tilde{v}_\ell[t]}[t] = y_1[1]y_1[2]y_1[3]y_2[4]y_1[5]$$

Setting II: *Maximum-Likelihood Segmentation*

+ *OK! We can find the likelihood, but how can we maximize it? It's again an exponentially large search!*

$$\ell^* = \operatorname*{argmax}_{\ell} p\left(\tilde{\mathbf{v}}_{\ell}[1:T] | \ell\right)$$

– Well! If we only need the maximum, *it turns not to be exponential*

Setting II: *Maximum-Likelihood Segmentation*

+ *OK! We can find the likelihood, but how can we maximize it? It's again an exponentially large search!*

$$\ell^* = \operatorname*{argmax}_{\ell} p\left(\tilde{\mathbf{v}}_{\ell}[1:T]|\ell\right)$$

– Well! If we only need the maximum, *it turns not to be exponential*

We can readily show that finding maximum likelihood on the graph is a dynamic programming problem and can be solved by the Viterbi algorithm

Maximum likelihood training can be implemented efficiently

Setting II: *Maximum-Likelihood Inference*

```
MaxLikelihoodTraining():
```

1: **for** Iteration $i = 1, \dots, I$ **do**
2: Pass forward through time: Compute output sequence $\mathbf{y}[1:T]$
3: Compute $p\left(\tilde{\mathbf{v}}_{\ell}[1:T] | \ell\right)$ for each path ℓ on segmentation graph ⊛
4: Update $\ell^* = \operatorname{argmax}_{\ell} p\left(\tilde{\mathbf{v}}_{\ell}[1:T] | \ell\right)$ ⊛
5: Set loss to $\hat{R}_{\ell^\star}$ and backpropagate over RNN
6: Update learnable parameters
7: **end for**
8: Return learnable parameters and ℓ^*

+ *How can we use our RNN for inference after training via maximum likelihood segmentation?*
– We have access to ℓ^*: *we predict the label of each segment based on its corresponding outputs*

Setting II: *Connectionist Temporal Classification*

It turns out that *maximum-likelihood could stick to a bad local minimum, i.e.,*

it quickly converges to a path ℓ^* *that is much different from* $\ell^\star$

- + *Is there any solution to this?*
- – Yes! We can use *connectionist temporal classification (CTC) loss*

CTC Loss

Instead of searching for a best segmentation and then minimizing its loss, we learn directly from unsegmented data by minimizing the average loss over all possible segmentations, i.e., we define loss to be

$$\hat{R} = \mathbb{E}_\ell \left\{ \hat{R}_\ell \right\} = \sum_{\ell=1}^{L} p\left(\ell | \tilde{\mathbf{v}}_\ell[1:T]\right) \hat{R}_\ell$$

and train the RNN by finding learnable parameters that minimize this loss

Setting II: *CTC Loss*

+ *But, why should it be a better choice of loss?*
– Because *we are sure that optimal segmentation is contributing to our loss*

$$\hat{R} = \sum_{\ell=1}^{L} p\left(\ell | \tilde{\mathbf{v}}_\ell[1:T]\right) \hat{R}_\ell = p\left(\ell^\star | \tilde{\mathbf{v}}_\ell[1:T]\right) \hat{R}_{\ell^\star} + \sum_{\ell \neq \ell^\star} p\left(\ell | \tilde{\mathbf{v}}_\ell[1:T]\right) \hat{R}_\ell$$

+ *Agreed! Now, how should we determine* $p\left(\ell^\star | \tilde{\mathbf{v}}_\ell[1:T]\right)$ *?*
– Just use the Bayes rule!
+ *What about the expectation? It is at the end sum of exponentially large number of terms!*
– We can again *go on the graph and determine it via dynamic programming*

Setting II: *CTC Loss*

The CTC loss can be written as

$$\hat{R} = \sum_{\ell=1}^{L} p\left(\ell | \tilde{\mathbf{v}}_\ell[1:T]\right) \hat{R}_\ell = \sum_{\ell=1}^{L} p\left(\ell | \tilde{\mathbf{v}}_\ell[1:T]\right) \sum_{t=1}^{T} w_t \mathcal{L}\left(\mathbf{y}[t], \tilde{\mathbf{v}}_\ell[t]\right)$$

$$= \sum_{t=1}^{T} w_t \underbrace{\sum_{\ell=1}^{L} p\left(\ell | \tilde{\mathbf{v}}_\ell[1:T]\right) \mathcal{L}\left(\mathbf{y}[t], \tilde{\mathbf{v}}_\ell[t]\right)}_{\breve{R}[t]} = \sum_{t=1}^{T} w_t \breve{R}[t]$$

This has been shown that $\breve{R}[t]$ can be recursively computed[6]:

by some approximation we are able to readily compute $\nabla_{\mathbf{y}[t]} \breve{R}[t]$

and we set $\nabla_{\mathbf{y}[t']} \breve{R}[t] \approx \mathbf{0}$ for $t' \neq t$

[6] Check out the original paper

Setting II: *Training with CTC Loss*

```
CTC_Training():
```

1: **for** iteration $i = 1, \dots, I$ **do**
2: Pass forward through time: Compute output sequence $\mathbf{y}[1:T]$
3: Compute CTC loss $\hat{R}$ and $\nabla_{\mathbf{y}[t]} \hat{R}$ by recursion
4: Backpropagate through time and update learnable parameters
5: **end for**
6: Return learnable parameters

This looks like standard training loop now

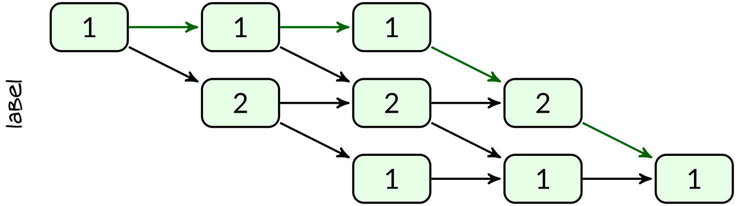
the loss is only replaced with CTC loss

+ *What about inference?*
– Well! We should figure it out, *since the training loop does not compute any segmentation path!*

Setting II: *Inference with CTC-Trained RNN*

Let's get back to our simple example: *assume that after training with CTC loss we give an image of handwritten* 121 to the RNN

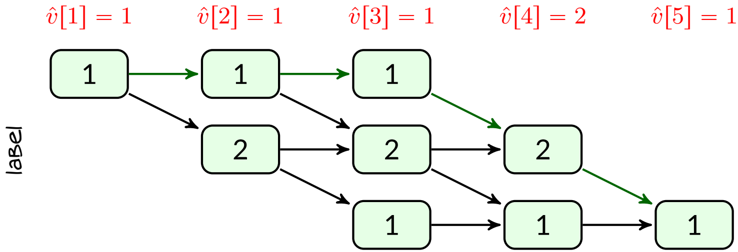
- *RNN divides it into 5 frames and is able to track optimal segmentation*
  - ↳ *The first three frames belong to the first segment*
  - ↳ *The remaining frames belong to the second and third segments*



Setting II: *Inference with CTC-Trained RNN*

Let's get back to our simple example: *assume that after training with CTC loss we give an image of handwritten* 121 to the RNN

- *RNN divides it into 5 frames and is able to track optimal segmentation*
  - ↳ *The first three frames belong to the first segment*
  - ↳ *The remaining frames belong to the second and third segments*
- *RNN infers from output sequence* $\hat{v}[1:5]$ *but does not return optimal path*



Setting II: *Inference with CTC-Trained RNN*

We can conclude from $\hat{v}[1:5]$ that the sequence is {1,2,1} *if we are sure the sequence has no repetition*

Label Encoding and Decoding

CTC uses this fact and constructs following encoding and decoding method: it introduces a new label called "blank:-" which does not belong to set of classes

Setting II: *Inference with CTC-Trained RNN*

We can conclude from $\hat{v}[1:5]$ that the sequence is {1,2,1} *if we are sure the sequence has no repetition*

Label Encoding and Decoding

CTC uses this fact and constructs following encoding and decoding method: it introduces a new label called "blank:-" which does not belong to set of classes

- *While training, it adds blank between any two repetitions*
  - ↳ *For instance, we encode 112 ↦ 1-12, or 111 ↦ 1-1-1*

Setting II: *Inference with CTC-Trained RNN*

We can conclude from $\hat{v}[1:5]$ that the sequence is {1,2,1} *if we are sure the sequence has no repetition*

Label Encoding and Decoding

CTC uses this fact and constructs following encoding and decoding method: it introduces a new label called "blank:-" which does not belong to set of classes

- *While training, it adds blank between any two repetitions*
  - ↳ *For instance, we encode 112 ↦ 1-12, or 111 ↦ 1-1-1*
- *For inference, it removes any repetition in inferred sequence $\hat{v}[1:T]$ and then drops blanks*
  - ↳ *For instance, we decode 1-11-312 ↦ 11312, or 3333-3121 ↦ 33121*

Setting II: *Training and Inference with CTC*

```
CTC_Training():
```

1: **for** iteration $i = 1, \dots, I$ **do**
2: Add blanks to the label sequences with repetition
3: Pass forward through time: Compute output sequence $\mathbf{y}[1:T]$
4: Compute CTC loss $\hat{R}$ and $\nabla_{\mathbf{y}[t]} \hat{R}$ by recursion
5: Backpropagate through time and update learnable parameters
6: **end for**
7: Return learnable parameters

Setting II: *Training and Inference with CTC*

```
CTC_Training():
```

1: **for** iteration $i = 1, \dots, I$ **do**
2: Add blanks to the label sequences with repetition
3: Pass forward through time: Compute output sequence $\mathbf{y}[1:T]$
4: Compute CTC loss $\hat{R}$ and $\nabla_{\mathbf{y}[t]} \hat{R}$ by recursion
5: Backpropagate through time and update learnable parameters
6: **end for**
7: Return learnable parameters

```
CTC_Inference():
```

1: Pass forward through time the input and compute output $\mathbf{y}[1:T]$
2: Infere encoded sequence $\hat{v}[1:T]$ from $\mathbf{y}[1:T]$
3: Remove repetitions from $\hat{v}[1:T] \mapsto \hat{v}[1:T']$
4: Remove blanks from $\hat{v}[1:T'] \mapsto \hat{v}[1:K]$
5: Return $\hat{v}[1:K]$

In PyTorch: *CTC Loss*

We can access CTC loss in `torch.nn` module as

```
torcn.nn.CTCLoss()
```

Few notes about CTC loss implementation

- *We need to specify the index of blank label*
  - ↳ *It should be out of our set of classes*
  - ↳ *By default, it is set to* `blank = 0`
- *When we define our model, we should always take blank label into account*
  - ↳ *If we do classification with C classes, model should return $C+1$ classes with blank being one of them*
- *PyTorch considers cross-entropy loss function, i.e.,* $\mathcal{L}(\mathbf{y}, \tilde{\mathbf{v}}) = \mathrm{CE}(\mathbf{y}, \tilde{\mathbf{v}})$
- *As input to CTC loss:* $\mathbf{y}$ *should be logarithm of probabilities*
  - ↳ *We can activate the output layer with logarithmic softmax*