

# RNN in action: Understanding sequences

We will study a toy example that is typical of many tasks involving sequences

- Given a prefix of a sequence
- Predict the next element

For example

- Predict the next word in a sentence
- Predict the next price in a timeseries of prices

Being able to predict the next element may be key to understanding the "logic" underlying a sequence

- You have to understand context and domain
- You have to understand how earlier elements influence latter elements

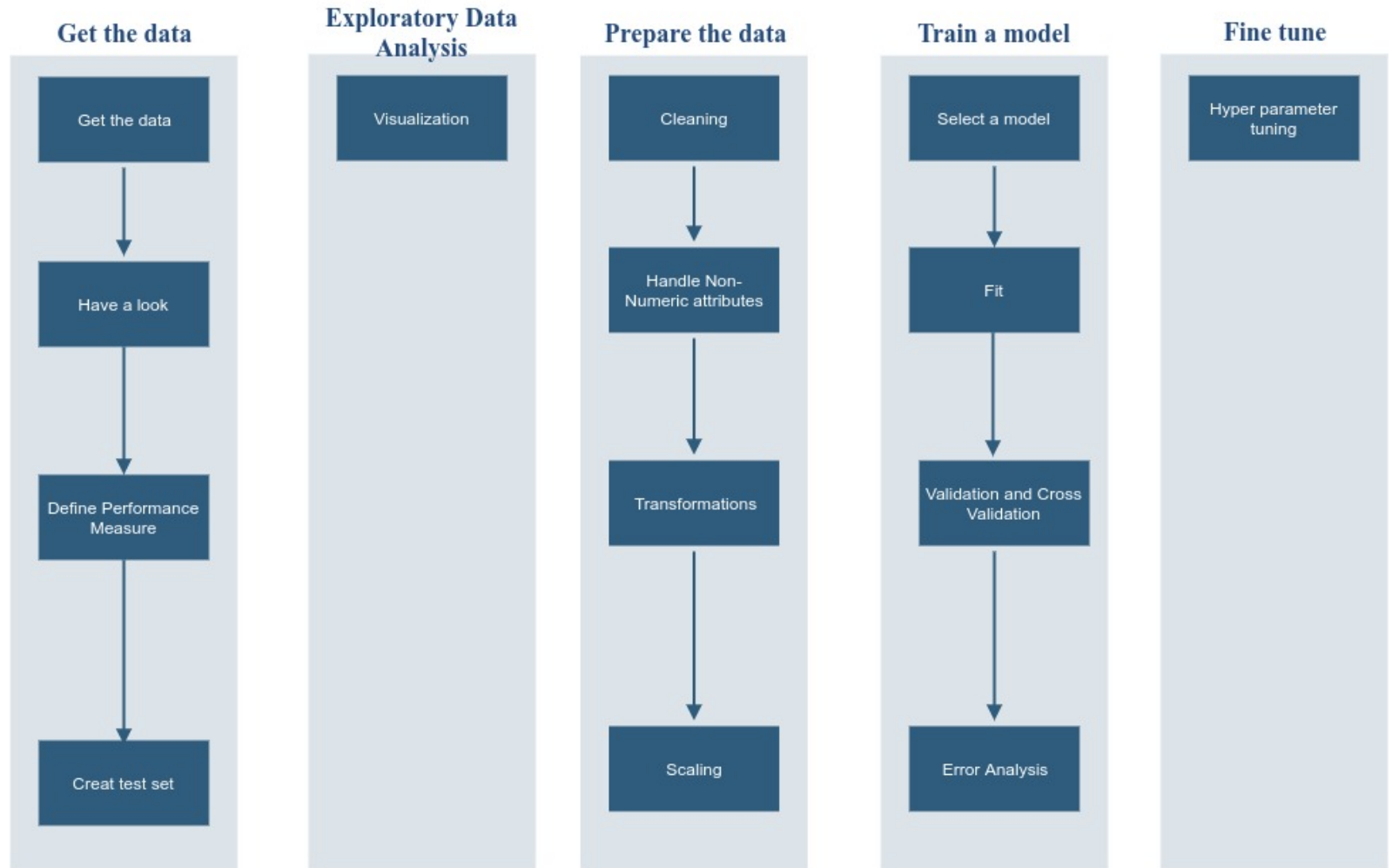
## Predict the next: Data preparation

It is our belief that Machine Learning is a *process* and not just a collection of models.

We have recently been emphasizing the models but let's review the process.

## Recipe for Machine Learning

---



It is usually the case that Sequence data involves substantial Data Preparation.

Suppose our task is to predict the next word in a sentence.

We are given (or must obtain) a collection of sentences (e.g., one or more documents) as our raw data.

But a sentence is not the format required for the training set of the "Predict the next word" task.

Data preparation is usually a substantial prerequisite for solving tasks involving sequences.

To be precise, the "Predict the next word" task involves

- Training a many to one RNN with examples created from a sequence.
- The elements of a single example are the prefix of a sentence
- The target of the example is the next word in the sentence



Let

$$[ \mathbf{s}_{(t)} | 1 \leq t \leq T ]$$

be the sequence of words in sentence  $\mathbf{s}$ .

We will prepare  $(T - 1)$  examples from this single sentence.

$$\langle \mathbf{X}, \mathbf{y} \rangle =$$

$i$	$\mathbf{x}^{(i)}$	$\mathbf{y}^{(i)}$
1	$\mathbf{s}_{(1)}$	$\mathbf{s}_{(2)}$
2	$\mathbf{s}_{(1),(2)}$	$\mathbf{s}_{(3)}$
$\vdots$		
$i$	$\mathbf{s}_{(1),\dots,(i)}$	$\mathbf{s}_{(i+1)}$
$\vdots$		
$(T-1)$	$\mathbf{s}_{(1),\dots,(T-1)}$	$\mathbf{s}_{(T)}$

For example

$\mathbf{s} = \text{"I am taking a class in Machine Learning"}$

$i$	$\mathbf{x}^{(i)}$	$\mathbf{y}^{(i)}$
1	[ I ]	am
2	[ I, am ]	taking
3	[ I, am, taking ]	a

## Predict the next: data shape

We had warned earlier about the explosion of the number of dimensions of our data. Now is a good time to take stock

- $\mathbf{X}$ , the training set, is a matrix with  $m$  rows
- Each row is an example  $\mathbf{x}^{(i)}$
- Each example is a sequence  $[ \mathbf{x}_{(t)}^{(i)} \mid 1 \leq t \leq ||\mathbf{x}^{(i)}|| ]$
- Each element  $\mathbf{x}_{(t)}^{(i)}$  of the sequence encodes a word
- A word is encoded as a One Hot Encoded binary vector of length  $||V||$  where  $V$  is the set of words in the vocabulary

Target  $\mathbf{y}^{(i)}$  is also a word (so is vector of length  $\|V\|$ ).

- Many to one: target is *not* a sequence

## Predict the next: training

Just like training any other type of layer, but more expensive

- Each example involves multiple time steps: forward pass time consuming
- The derivatives (needed for Gradient Descent) are more complex; backward pass complex and time consuming

# **RNN as a generative model (fun with RNN's)**

The "Predict the next" word task is interesting on its own

- But a slight twist will make it extremely interesting

Suppose

- We train the RNN on a large number of sentences of the same type (e.g., same author)
- Create a few words to create the prefix of a sentence
- Ask the RNN to predict the next word
- Append this word to the prefix
- Repeat !

Voilà: the RNN can *generate* a story in the same style as the training sentences.



Using Machine Learning to *create* data is called *generative*.

Using Machine Learning to classify/predict (as we've been doing thus far) is called *discriminative*.

# Architecture

How do we construct a model to solve this task ?

We construct a two part model in what is known as an *Encoder/Decoder* architecture

The *Encoder* is a many to one RNN

- Takes the variable length "seed" sequence
- Outputs a fixed length representation of the seed
  - This is one of the strengths of an RNN

The *Decoder* is a one to many RNN

- Takes the fixed length representation of the seed produced by the Encoder
  - Used to initialize the Decoder's latent state  $\mathbf{h}_{(0)}$
- Outputs a variable length sequence

The only thing unusual about the Decoder is how its input sequence is constructed

- All the elements of its input  $\mathbf{x}$  are *generated* by the Decoder
- The first element of  $\mathbf{x}$  is set to a special "start of output" symbol

$$\mathbf{x}_{(1)} = \langle \text{START} \rangle$$

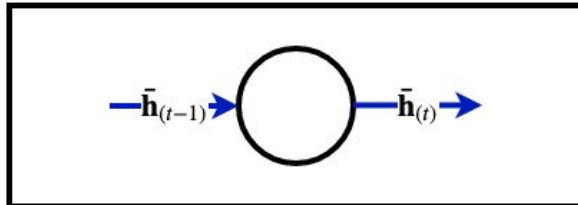
- $\mathbf{x}_{(t)}$  is extending *dynamically*, using the previous prediction  $\hat{\mathbf{y}}_{(t-1)}$

$$\mathbf{x}_{(t)}^{(i)} = \mathbf{y}_{(t-1)}$$

- The Decoder stops when it generates a special "end of output" symbol  $\langle \text{END} \rangle$

## Sequence to Sequence: inference

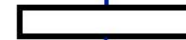
Encoder



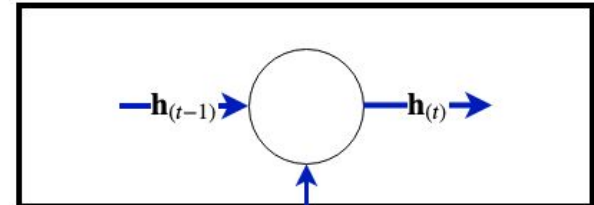
$\mathbf{x}_{(1)} \mathbf{x}_{(2)} \dots \mathbf{x}_{(\bar{T})}$

Decoder

$\hat{\mathbf{y}}_{(1)} \hat{\mathbf{y}}_{(2)} \dots \hat{\mathbf{y}}_{(t)} \dots \hat{\mathbf{y}}_{(\bar{T})}$



$\mathbf{h}_{(t)}, \mathbf{s}$



$\mathbf{h}_{(0)} = \bar{\mathbf{h}}_{(T)}$

$\mathbf{s} = \bar{\mathbf{h}}_{(\bar{T})}$

$\langle \text{Start} \rangle \hat{\mathbf{y}}_{(1)} \hat{\mathbf{y}}_{(2)} \dots \hat{\mathbf{y}}_{\bar{T}}$

That is

- The output is fed as the next input
- Thus extending the sequence
- And making sure that subsequent elements are influenced by all previously *generated* elements
- Continuing until the special  $\langle \text{END} \rangle$  symbol is generated

## Training: Teacher forcing

Some issues arise in using an RNN in the generative manner.

The first issue:

- Is the prediction a single word or a probability distribution over the vocabulary  $|V|$
- If it's a single word: the output is deterministic
  - Problematic once one word is wrong: the error propagates forward



The output of the multinomial classifier is a vector of length  $||V||$

- With values in the range  $[0, 1]$  that can be interpreted as probabilities
- Rather than choosing the single word with highest probability
- We *sample* one word at random, according to the probability distribution

This makes the output non-deterministic: running the model twice with the same "seed" may give different stories.

## A second issue

- If a wrong word is chosen at step  $t$ : it affects the generation of all words at step  $t > t'$
- This is particularly problematic at *training* time: makes learning difficult

The solution for training an RNN for this task is a method known as *teacher forcing*

- Rather than extending the seed example  $\mathbf{x}^{(i)} = [ \mathbf{x}_{(t)}^{(i)} \mid 1 \leq t \leq t' ]$ 
  - With  $\hat{\mathbf{y}}_{(t)}$ , the *predicted*  $t^{th}$  word for  $(t > t')$
  - Which is what would happen at inference/test time
  - Extend it with  $\mathbf{y}_{(t)}$ , the *target* (i.e., correct  $t^{th}$  word)

In other words: to speed up training

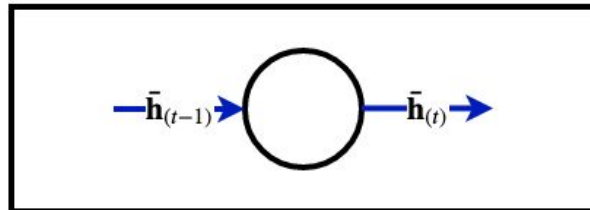
- When extending the prefix
- A teacher forces the student (model) to continue with the *correct* answer
- Rather than the student's answer

$$\mathbf{x}_{(t)}^{(i)} = \mathbf{y}_{(t-1)}$$

for  $t > t'$ .

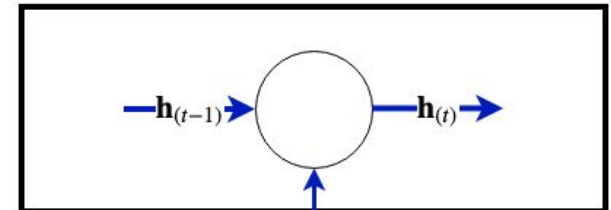
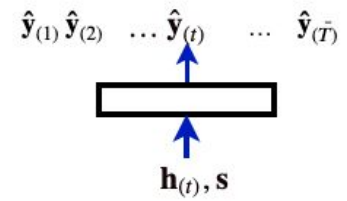
Sequence to Sequence: training (teacher forcing)

Encoder



$\mathbf{x}_{(1)} \mathbf{x}_{(2)} \dots \mathbf{x}_{(\bar{T})}$

Decoder



$\mathbf{h}_{(0)} = \bar{\mathbf{h}}_{(T)}$

$\mathbf{s} = \bar{\mathbf{h}}_{(\bar{T})}$

**<Start>**  $\mathbf{y}_{(1)} \mathbf{y}_{(2)} \dots \mathbf{y}_{\bar{T}}$

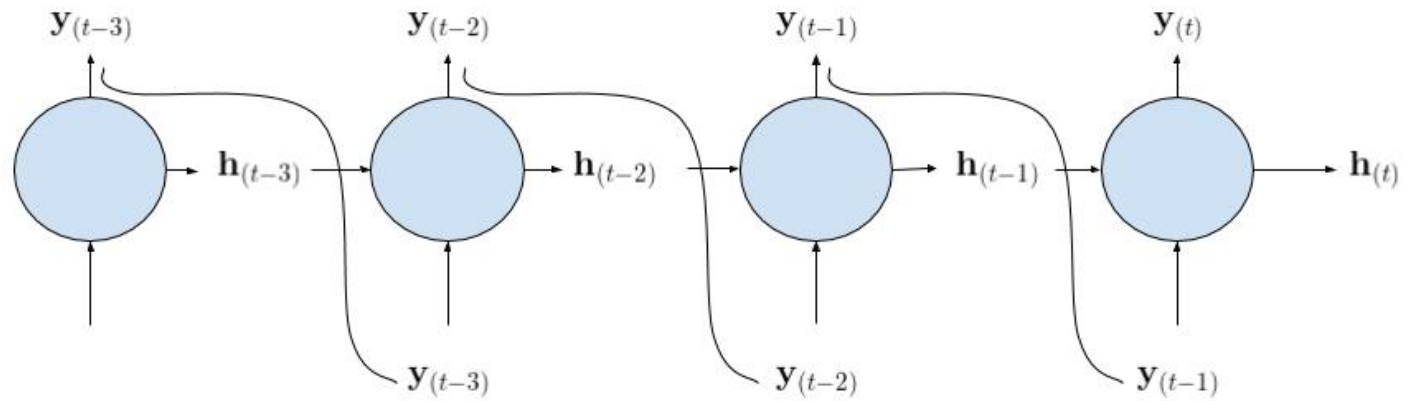
Teacher forcing is indicated in red

- Predictions  $[ \hat{\mathbf{y}}_{(t)} \mid 1 \leq t \leq T ]$  **are not** used as input (lower right)
- Only correct targets  $[ \mathbf{y}_{(t)} \mid 1 \leq t \leq T ]$  are used

## Summary

Here is an unrolled graph at inference/test time

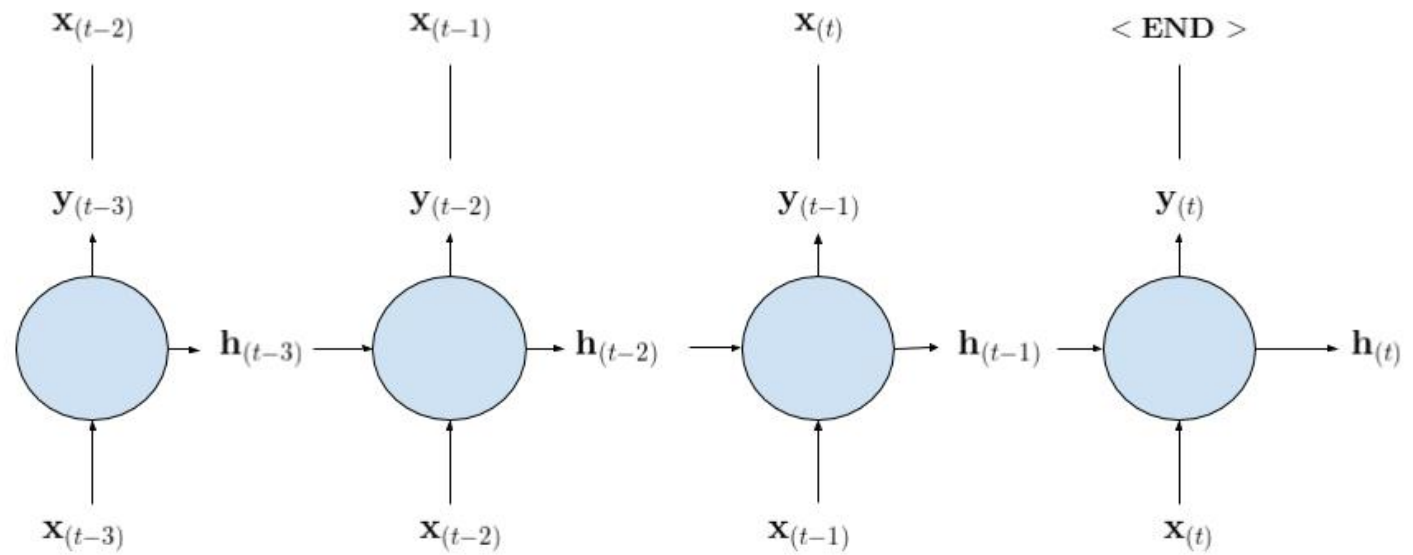
Inference





And here is a depiction of the graph used at *training* time

Training targets:



## **Generating strange things**

Generating stories from seeds was very popular a few years back.

Let's look at some examples.

But first, a surprise:

- Rather than solving a "predict the next word" task
- All of the following examples were generated by a "predict the next **character**" task !

It is somewhat amazing that what is generated

- Has correctly spelled words/keywords
- Is Syntactically correct (sentences end with a ".", parentheses/brackets are balanced)
- Is meaningful: the elements/words are arranged in a logical order

Even though

- We have not explicitly identified any of these concepts
- Nor forced training to respect them (via a loss function)

Remember

- All of this behavior was "learned" by identifying the correct next **character**

- Fake Shakespeare (<http://karpathy.github.io/2015/05/21/rnn-effectiveness/#shakespeare>), or fake politician-speak
- Fake code
- Fake math textbooks (<http://karpathy.github.io/2015/05/21/rnn-effectiveness/#algebraic-geometry-latex>)
- Click bait headline generator (<http://clickotron.com/about>)

In [2]: `print("Done")`

Done