Risk Manager/Quant world

- Our job is to deliver insights
    - to portfolio managers
    - traders
    - management
- Insights
    - into our portfolio
    - into the market
    - into trader behavior

Our insights are derived from *data analysis*

- price histories
- discovery of common factors
- trade performance

And lead to actionable decisions

- individual security buy/sell recommendations
- which sectors to over/under weight
- where (and to whom) to allocate capital
- what factors influence price
- under what conditions is a trader successful ?

The raw material for our analytics has, historically, been numeric

- But there is a lot of non-numeric data that may aid prediction
    - images
    - speech -text

Natural Language Processing is the set of tools/techniques that facilitate using text as raw material.

The world of text

- SEC filing
- analyst reports
- news articles
- tweets

There is a sequence of steps in dealing with text.

We will highlight the challenges (and some solutions) in creating an NLP pipeline/workflow

Our goals:

- Learn to analyze text
- Learn to *augment* numerical analyses with inputs derived from text
- Introduce open-source tools to enable us to deal with the challenges

Challenges: Overview

We briefly introduce the major components of an NLP pipeline.

After a brief intro to all, we will do a deeper dive on each.

- Obtaining text
    - web scraping
- Word recognition
    - tokenization, parsing
        - part of speech
        - entity recognition
- Representation of words
- Representation/meaning of sentences/documents

Challenge 1: Obtaining text

Documents are rarely presented to as a simple collection of words

- Structure documents
    - Web pages
    - Bloomberg articles
- Markup
    - needs to be removed (HTML) to extract the "text" parts

We will briefly describe Web-Scraping as a partial solution to this challenge.

Challenge 2: Word recognition

Text is more than words.

- word separators, punctuation
- Markup

Even after we have removed markup

- we have punctuation
    - Divides sentences, so has semantic meaning, not just syntactic
- Parsing strings into words
    - tokenization
- Word variants
    - plural
    - Capitlization

We will briefly describe some tools for processing text into words.

Challenge 3: Word representation

We motivate our study of NLP by enumerating key issues in dealing with text.

Notation

- Let $\mathbf{V}$ be the **vocabulary**
    - the ordered collection of distinct words in our universe
    - the i^{th} word is denoted $\mathbf{V}_i$

Text is not a number

- Models/algorithms deal with numbers: text is not a number !

How hard can it be to turn a word into a number ?

- i.e., assign an integer I_i to word $\mathbf{V}_i$

The naive method of arbitrarily mapping a word to an integer won't work.

- it will imply an order and a magnitude

Example

Linear regression:

$$\mathbf{y} = \Theta^T \mathbf{x}$$

Predict $\mathbf{y}$ given feature vector (attributes) $\mathbf{x}$

- by learning parameters Θ

- But our assignment of integers to words was arbitrary
    - multiply all integers by 10
    - permute the assignment

Suppose one feature $\mathbf{x}_j$ is the numeric encoding of a word

- if "apple" is encoded by integer 100 rather than integer 10
  - $I_{\text{apple}} = 100$ versus $I_{\text{apple}} = 10$
  - does it have 10 times the impact on $\mathbf{y}$?
  - according to the equation
    - impact is $\Theta_j * \mathbf{x}_j$
    - so impact is proportional to the numeric value of the word

What about the differential impact of $\mathbf{x}_j$ being different words ?

- $I_{\text{apple}} = 100$ versus $I_{\text{orange}} = 10$
- Are apples 10 times more important than oranges ?
    - impact is $\Theta_j * \mathbf{x}_j$ so, yes, that's what it implies

Words are categorical variables, not ordinal

- Ordinal
    - has magnitude
    - ordering relationships ($<$, $=$, $>$) is defined
- Categorical
    - no magnitude
    - nor ordering
        - Is the word "apples" > "apple" ?
        - Which is greater: "apples" or "oranges" ?

There are lots of words !

Representing categorical variables

- dummy/indicator variables

Create a binary indicator variable *for each word* in the vocabulary

- $\text{Is}_{\text{apple}}, \text{Is}_{\text{orange}}$

Linear Regression deals with Categoricals quite well

- the contribution to prediction $\mathbf{y}$ is
    - 0 if "apple" not present as a feature
    - increases by Θ_j if "apple" is word j

- Generalization: One Hot Encoding (OHE)
    - vector of length equal to vocabulary size $|V|$
    - all elements of vector 0 except for position j, for the $j^t h$ word in vocabulary
    - word i represented by a vector $\mathbf{v}$
        - $\mathbf{v}_j = 0, \forall j \neq i$
        - $\mathbf{v}_i = 1$

Let $\mathbf{v}_w$ denote the One Hot Encoding of word w (e.g., "apple").

The length of $\mathbf{v}_w$ is $|\mathbf{V}|$, the number of words in the vocabulary.

Easy enough to convert each word to a categorical variable.

Are we done ?

A vocaulary easily has thousands (more like tens of thousands) or words

- OHE are long !
- Stated another way:
    - the number of independent variables in our regression equation is $|V|$

It is unwieldy to deal with thousands of categorical variables

What's in a word ?

What is "orange"

- a color ?
- a fruit ?
- a brand name ?

The string of characters is not sufficient to convey full meaning !

- How do we discern "which orange" we mean ?
- The distinct meanings increases the size of $\mathbf{V}$ even more !

- Parts of speech
- Named Entity Recognition
    - recognizing a word as an instance of a "concept"
        - can replace the concrete word by the name of the concept in many cases
            - recognizing Apple as company
            - recognizing "11/04/2019" as a date

Challenge 4: Sentence/Document representation

Now that we addressed th represenation of words: how to we represent a sentence/document ?

Let $\mathbf{w}$ be the array of n words in a sentence.

The simplest way to represent a sentence would be the *set* of word representations.

This turns out to be impractical for the same reason that OHE representaion of words is impractical: size.

- Let $\text{rep}(\mathbf{w}_i)$ be a represention of word i
- $\{\text{rep}(\mathbf{w}_1), \text{rep}(\mathbf{w}_2), \ldots, \text{rep}(\mathbf{w}_n)\}$
- But if $\text{rep}(\mathbf{w}_i)$ is a OHE
    - total size $n * |V|$. Huge !

Sentences are of varying length

There is another issue with the array of words:

- sentences are of varying length

It would be highly undesirable if our algorithm was in any way influenced by the length of the sentence.

Moreover: many models can only deal with *fixed length* inputs

- pad short sequences
- truncate long sequences

Sentences/Documents are sequences not sets

To state the obvious: the ordering of words in a sentence is important.

- [..., "not", "like", ...] vs ["like", "me", ..., "not", "anyone, "else"]
    - consecutive "not like" is negative; independent is not necessarilty positive/negative

So our sentence representation must respect order.

Thus, an array of word representations is preferable.

$$[\text{rep}(\mathbf{w}_1), \text{rep}(\mathbf{w}_2), \ldots, \text{rep}(\mathbf{w}_n)]$$

The disadvantage, compared to a set

- the length of the sentence reprsentation depends on number of words in sentence
    - rather than number of *unique* words

Subsequences/n-grams

Words in isolation don't necesarily convey full meaning of a concept:

- "New York City" versus ["New", "York", "City"]
- A bigram is a two token sequence that encodes a single concept
- A trigram is a three token sequence

Dealing with the challenges

We now describe how to deal with each challenge.

It's important to note that many choices are *not independent* of one another

- later stages of the pipeline are impacted by earlier choices
- adds complexity; no simple answer for each step

Response: Obtaining text challenge

Text data rarely comes to you in a neatly wrapped package.

Your first challenge is obtaining it

- PDF files
- Web pages

The data is typically in some structured form that includes both the text (that you want) and *mark-down* (e.g., HTML, which gives meaning to some text, but which you ultimately don't want).

Web scaping

The problem

Here is a typical web page, as rendered in your browswer

Web page

1:30 **Natural language processing as part of the quant process**

- Why is text data interesting?
- Why is text different? Challenges and solutions
- Classical machine learning methods for dealing with text
- Deep Learning methods for dealing with text word embedding's

- Convolutional Neural Networks
- Recursive Neural Networks
- The importance of attention

Ken Perry, founder, Slashrisk, adjunct professor, NYU Tandon School of Engineering

3:00 **Afternoon break**

3:30 **Machine learning in risk management**

- ML applications in risk management
- Analyze large amounts of data while maintaining granularity of analysis
- Tools to optimize and accelerate model risk management
- Reporting requirements within financial services
- Pre-trade risk controls and best execution analysis
- Data privacy, security and governance laws

Suppose we want the abstract, time and speakers for each session

- in this case, we preserve some of the structure

Here's what the web page's source looks like

Page source

```
    <p><strong>Ken Perry, </strong>founder, Slashrisk, adjunct professor, NYU Tandon School of Engineering</p>
    </td>
</tr>
<tr>
    <td style="width: 45px;">
    <p class="time">3:00</p>
    </td>
    <td colspan="2" style="width: 1188px;">
    <p><strong>Afternoon break</strong></p>
    </td>
</tr>
<tr>
    <td style="width: 45px;">
    <p class="time">3:30</p>
    </td>
    <td colspan="2" style="width: 1188px;">
    <p><strong>Machine learning in risk management</strong></p>

    <ul>
        <li>ML applications in risk management</li>
        <li>Analyze large amounts of data while maintaining granularity of analysis</li>
        <li>Tools to optimize and accelerate model risk management</li>
        <li>Reporting requirements within financial services</li>
        <li>Pre-trade risk controls and best execution analysis</li>
        <li>Data privacy, security and governance laws</li>
    </ul>
    </td>
</tr>
```

- It is highly structured
    - Good ! Can extract semantic concepts
        - time
        - speaker
    - Bad ! How do I deal with this ? I only want the text !

Solution

A Web-Scraper is a tool that can parse Page Source (HTML/XML) into units

- preserve structure
- remove markup
- left with text !

Beautiful Soup: a tool for document scraping

Beautiful Soup is a very popular, open-source tool for scraping.

https://www.crummy.com/software/BeautifulSoup/
(https://www.crummy.com/software/BeautifulSoup/#Download)

If you're serious about text, you should

- invest in learning a tool set
- **build your own** higher level tools for common tasks

This is a tool for programmers, not a GUI.

Here's some sample HTML source (taken from the documentation):

This is paragraph **one** .

This is paragraph **two** .

And some Python code to navigate/extract contents.

First: some imports and loading the document into BS.

from BeautifulSoup import BeautifulSoup # For processing HTML from BeautifulSoup import BeautifulStoneSoup # For processing XML import BeautifulSoup # To get everything import re # Here is the page source, already loaded into a string -- for convenience of presentation only doc = ['', '

This is paragraph **one**.', '

This is paragraph **two**.', ''] doc = ''.join(doc) soup = BeautifulSoup()

Let's walk through the document.

What's the first element ?

```
soup.contents[0].name # u'html'
```

The first element (html) is *itself* structured.

What is the first part (of the first element) ?

soup.contents[0].contents[0].name # u'head'

What about the first part of the next element (i.e., sibling of `html`)

head.nextSibling.contents[0] #

This is paragraph **one**.

Response: Word recognition challenge

Normalization

Put the raw words into a standard form (after removing markdown) by:

- tokenization
    - isolating strings of characters into "word" tokens
- word normalization
    - case
    - tense
    - stemming
    - lemmatization

Tokenization

This is not terribly complicated, nor is it completely trivial.

It usually involves some decent amount of pattern matching for which *regular expressions* are very useful.

Some questions

- how to separate words
    - word separators: space, line-end, punctuation
        - is language dependent: Mandarin, Japanese don't use spaces
        - punctuation may carry meaning
            - "Really ?", "Really !", "Really !!!"
    - special cases
        - punctuation as part of a word: U.S.A.

We won't dwell on this and will just use tokenizers from some standard libaries.

- `sklearn`
- `keras`
- `nltk`
- `spaCy`

Word normalization

We want to put words in a "standard" (canonical) form.

This can be problem dependent and **choices affect the entire downstream flow !**

- machine translation problems may benefit from minimal normalization
- text classification may benefit by reducing variation

Some issues

- Case sensitive or not ? Apple/apple
- U.S.A , USA, U.S, US $\mapsto$ US
- tense

Case folding

- Do we turn everything into lower case ?
    - Case sometimes conveys meaning
        - Proper names: Bill vs bill
        - Start of sentence indicator
        - Indicates emotion: NO WAY
    - where did the text come from ? Text messages often in all lower case

Stemming

- plural/singular
    - car/cars $\mapsto$ car
- tense
    - walk, walks, walking, walked $\mapsto$ walk

Lemmatization

More complex version of stemming: determines whether words have the same root

- am, is, are $\mapsto$ be

Stop words

Stop words are words with low information content: "a", "the"

Task specific (so no simple rule works for all tasks)

- remove for
    - classification
- don't remove for
    - translation
    - question answering

Sentence segmentation

Each sentence (should) represent a single thought.

Where does the sentence end ?

Not always clear

- role of semi-colon

Tagging/POS

For some problems, identifying parts of speech POS may be relevant.

- Bill (the person): proper noun
- Bill me later: verb, start of sentence
- send me the bill: noun

Named Entity Recognition

Entity recognition: a word as an instance of a "concept"; replace the concrete word with the concept.

- Alice, Bob are instances of the `Person` entity
    - does your task only need to know that the subject is a Person or a specific person ?
    - Ken Perry: multi-word proper name, `Person` entity
        - "In his talk Ken Perry said .."
        - "In his talk PERSON said .."
- 11/04/2019 is instance of a `Date` entity
- Google is instance of `Organization` entity

Note that recognizing entities may be affected by prior decisions: eliminating case

Toolkits for Natural Language Processing

Two popular toolkits to solve word-recognition (and more) problems

- nltk
- spaCy

NLTK

https://www.nltk.org/ (https://www.nltk.org/)

Let's convert a sentence into words (tokenization).

import nltk sentence = """At eight o'clock on Thursday morning ... Arthur didn't feel very good."""" tokens = nltk.word_tokenize(sentence) tokens ['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']

Lest you think that tokenization is trivial: notice the sophisticated way it handled

- "o'clock"
    - recognized as single word
- "didn't"
    - recognized as contraction of "did" and "not"

Let's identify parts of speech. We might use these to derive meaning.

tagged = nltk.pos_tag(tokens) tagged[0:6] [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN')]

Who knew their were so many parts of speech !

- "8" is a CD: cardinal digit
- "Thursday" is an NNP: proper noun, singular (sarah)
- "morning" is an NN: noun, singular (cat, tree)
- "At", "on" are IN: preposition/subordinating conjunction

spaCy

https://spacy.io/ (https://spacy.io/)

Response: Word representation challenge

Congratulations ! You have converted strings of characters "words".

These "words" are organized into larger collections

- a sentence is a *sequence* of words.
- a document is a sequence of sentences

Before we even get to how ML deals with sequences, let's talk about how we can turn individual words to numeric vectors.

- Let $\mathbf{V}$ be the **vocabulary**
    - the collection of distinct words in our universe
    - denoted by a vector $\mathbf{V}$ so that word i is denoted $\mathbf{V}_i$

Each word $v \in \mathbf{V}$ needs to be turned into a *feature vector* $\mathbf{v}$: an array of numbers encoding the word.

Note that that $\mathbf{v}$ is a vector (perhaps of length 1) because a word may turn out to be encoded by many features.

Aside: Expanded Vocabularies

It is not uncommon to expand the Vocabulary in order to capture some semantic information:

- tokens to denote position: <START>, <END>
- n-grams
    - pairs, triples, etc. of consecutive words
        - in this case may appear in each of these tuples

So when we refer to Vocabulary, this sometimes includes more than just the raw words.

Representing words by integers

Perhaps the simplest representation of v is as the index in $\mathbf{V}$ of v

- $\mathbf{v}$ where $\mathbf{V}_i$
  $= [i]$ $= v$

Properly speaking, words are most likely *categorical* (not *ordinal*, i.e., no ordering relationship between words).

So one should not make use of the ordinal relationship of the integer representation.

- `sklearn.preprocessing.LabelEncoder` (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)

Sparse Represention of words: One Hot Encoding (OHE)

The proper way to represent categorical variable v is via One Hot Encoding.

- $\mathbf{v}$ is of length $||\mathbf{V}||$
- $\mathbf{v}_i = 1$ for the i such that $\mathbf{V}_i = v$
- $\mathbf{v}_j = 0$ for $j \neq i$
- `sklearn.preprocessing.OneHotEncoder` (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#s

OHE is a *sparse* representation

- length of $\mathbf{v}$ is $||\mathbf{V}||$, yet only a single non-zero element
- problematic for large $\mathbf{V}$

Dense representation of words: Embeddings

Sparse encodings above essentially reduced the representation of a word to a single active feature.

This is called a *discrete* representation.

Categorical representations have a major drawback

- there is no meaningful metric of "distance" between the representation of words

Let

$$\mathrm{OHE}(w)$$

denote the One Hot Encoding of word w.

Using dot product (cosine similarity) as a measure of similarity

word	OHE(word)	Similarity
dog	[1,0,0,0]	OHE(word) · OHE(dog) = 1
dogs	[0,1,0,0]	OHE(word) · OHE(dog) = 0
cat	[0,0,1,0]	OHE(word) · OHE(dog) = 0
apple	[0,0,0,1]	OHE(word) · OHE(dog) = 0

Each pair of distinct words has 0 similarity

- no recognition of plural form
- no recognition of commonality (pets)

However, it's possible that there are many "dimensions" to a word, for example

- singular/plural
- entity type, e.g., Person
- positive/negative

Thus it is not unreasonable to represent a word as a vector of features where there is a numeric strength associated with the feature.

Ideally the features would be indepenent?

This is called a *continuous* word representation.

In the section on Embeddings, we will learn how to automatically construct dense vector representations in a way that capture properties of the words in $\mathbf{V}$.

Evaluating Dense vector representations

Word analogies

king:man :: ? : queen

Let

- $\mathbf{v}_w$ be the dense vector for word w
- $d(\mathbf{v}_w, \mathbf{v}_{w'})$ be some measure of the distance between the two vectors $\mathbf{v}_w, \mathbf{v}_{w'}$
    - e.g., (1- cosine similarity)

Using the distance metric, define the set of words in vocabulary $\mathbf{V}$ that are "closest" to a word w.

Let

- $\text{wv}_{n',d}(\mathbf{v}_w)$ be the dense vectors of the n' words in $\mathbf{V}$ closest to word w
$$\text{wv}_{n',d}(\mathbf{v}_w) = \{\mathbf{v}_{w'} | \text{rank}_V(d(\mathbf{v}_w, \mathbf{v}_{w'})) \leq n'\}$$
- $N_{n',d}(w)$ be the set of n' words in $\mathbf{V}$ closest in distance metric d to word w
$$N_{n',d}(w) = \{w' | w' \in \text{wv}_{n',d}(\mathbf{v}_w)\}$$

We can define approximate equality of two words w, w' if they are among the closest words

$$w \approx_{n',d} w' \quad \text{if } \mathbf{w}' \in N_{n',d}(w)$$

Finally, we can define word analogies:

a:b :: c:d

means

$$\mathbf{v}_a - \mathbf{v}_b \approx_{n',d} \mathbf{v}_c - \mathbf{v}_d$$

So to solve the word analogy for c:

$$\mathbf{v}_c \approx_{n',d} \mathbf{v}_a - \mathbf{v}_b + \mathbf{v}_d$$

To be concrete:

$$\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} \approx_{n',d} \mathbf{v}_{\text{queen}}$$

Why does adding 2 word vectors work

- Mikolov
    - Vector for a word reflects its context
    - Vector is log probability
        - so sum of log probabilities is log of product of probabilities
            - product is like a logical "and"

GloVe: Pre-trained embeddings

GloVe is a family of word embeddings that have been trained on large corpra

- GloVe6b
    - Trained on 6 Billion tokens
    - 400K words
    - Corpus: Wikipedia (2014) + GigaWord5 (version 5, news wires 1994-2010)
    - Many different dense vector lengths to choose from
        - 50, 100, 200, 300

We will illustrate the power of word embeddings using GloVe6b vectors of length 100.

$$\begin{aligned}
&\text{king- man + woman} && \approx_{n',d} && \text{queen} \\
&\text{man - boy + girl} && \approx_{n',d} && \text{woman} \\
&\text{Paris - France + Germany} && \approx_{n',d} && \text{Berlin} \\
&\text{Einstein - science + art} && \approx_{n',d} && \text{Picasso}
\end{aligned}$$

You can see that the dense vectors seem to encode "concept", that we can manipulate mathematically.

You may discover some unintended bias

$$\begin{aligned} \text{doctor - man + woman} &\approx_{n',d} \text{nurse} \\ \text{mechanic - man + woman} &\approx_{n',d} \text{teacher} \end{aligned}$$

Domain specific embeddings

Do we speak Wikipedia English in this room ?

Here are the neighborhoods of some financial terms:

$$
\begin{aligned}
N(\text{bull}) &= [\text{cow, elephant, dog, wolf, pit, bear, rider, lion, horse}] \\
N(\text{short}) &= [\text{rather, instead, making, time, though, well, longer, shorter, long}] \\
N(\text{strike}) &= [\text{workers, struck, action, blow, striking, protest, stoppage, walkout,} \\
N(\text{FX}) &= [\text{showtime, cnbc, ff, nickelodeon, hbo, wb, cw, vh1}]
\end{aligned}
$$

It may be desirable to create word embeddings on a narrow (domain specific) corpus.

Creating embeddings from word prediction problems

Word embeddings can be obtained as a by-product of a *word prediction* problem.

Let $\mathbf{w}$ be the sequence of n words $[\mathbf{w}_{(0)}, \mathbf{w}_{(1)}, \ldots, \mathbf{w}_{(n)}]$

A *word prediction* is a mapping from input to a probability distribtuion over vocabulary $\mathbf{V}$

- a vector of length $|\mathbf{V}|$ each value being in the range $[0, 1]$ and summing to 1.

Here are some simple word prediction problems:

predict next word from context	$p(\mathbf{w}_{(i)}\|\mathbf{w}_{(i-1)}, \ldots, \mathbf{w}_{(i-m)})$
predict a surrounding word	$p(\mathbf{w}_{(i')}\|\mathbf{w}_{(i)})$
predict center word from conext	$p(\mathbf{w}_{(i)}\|[\mathbf{w}_{(i-m)}, \ldots, \mathbf{w}_{(i-1)}, \mathbf{w}_{(i+1)}, \ldots, \mathbf{w}_{(i+m)}])$

word2vec: concrete example of creating word embeddings

Prediction problem for word2vec

word2vec is based on one of two prediction problems.

- predict center word given surrounding words as context
- precict which words can occur on either side of a given center words

- Vocabulary V
- window m on either side
- k length of embedding vector

Let $\mathbf{w}$ be the array of n words in a sentence.

For convenience we define two pseuduo-words to denote the start/end of the sentence

- $\mathbf{w}_0 = <\text{START}>$
- $\mathbf{w}_{n+1} = <\text{END}>$

The problems are framed as:

Prediction problem: Predict target word w_t given conditional word w_c

- $p(w_t|w_c)$

The first prediction problems is called *Skip gram*

- One (center word) to many (surrounding words)
    - w_c conditional word (input word) is center word
    - w_t is one of the "context" words in a window of size n from the center word
    - So given sentence fragment of size $2m + 1$:

$$w_{i-m}, \ldots, w_{i-1}, w_c, w_{i+1}, \ldots, w_{i+m}$$

        - training (input, label) pairs:

$$\{(w_c, w_j) \mid j \in \{(i-m), \ldots, (i+m)\} - \{i\}\}$$

    - w_c is a one-hot vector of length $|V|$
    - w_j is a one-hot vector of length $|V|$: probability vector

The second prediction problem is called *CBOW*

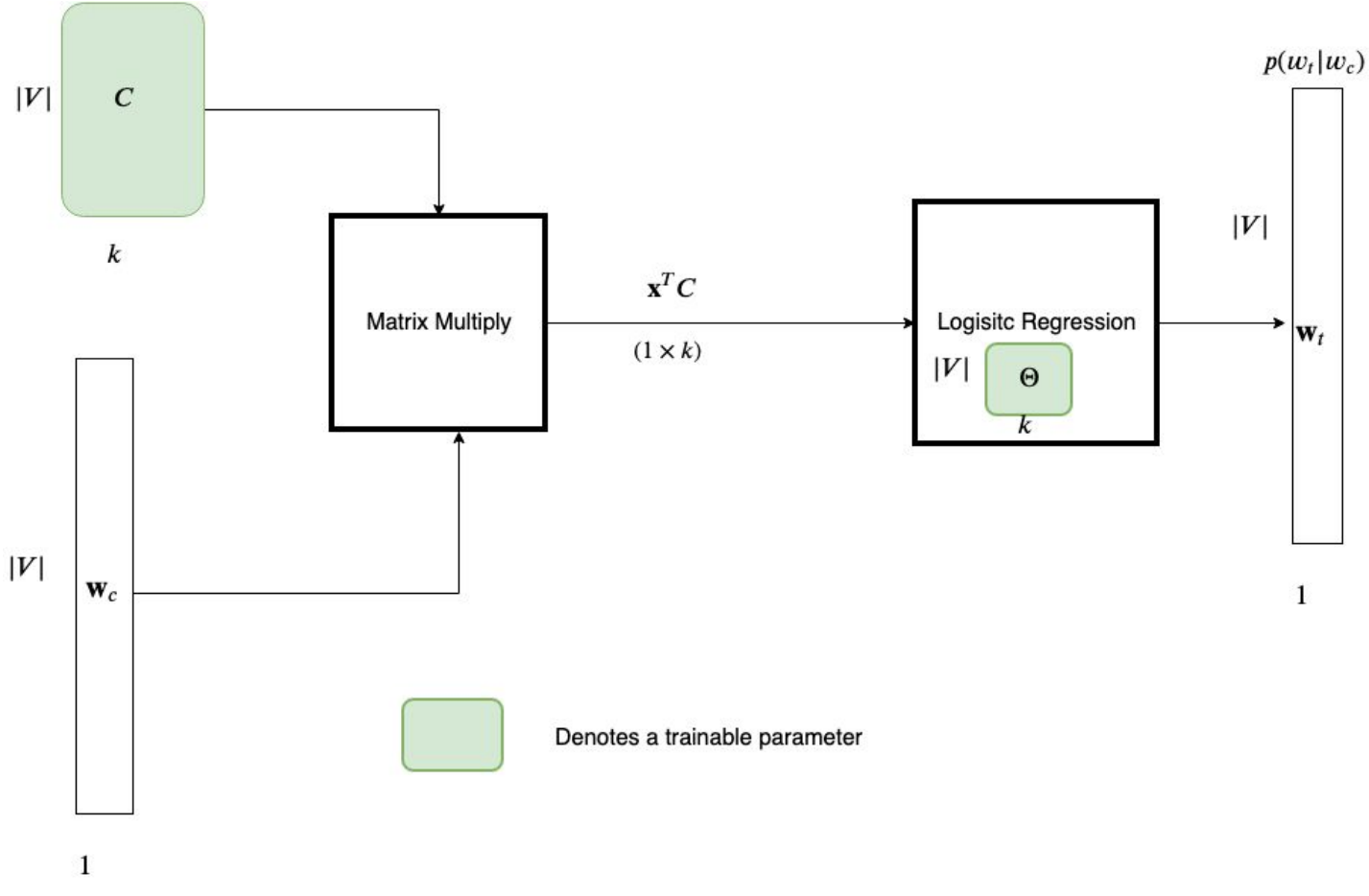
- many (surrounding words) to one (center word)
  - w_c: conditional words are words surrounding $\mathbf{w}_i$
  - $w_t = w_i$
  - So given sentence fragment of size $2m + 1$:

    $$w_{i-m}, \ldots, w_{i-1}, w_t, w_{i+1}, \ldots, w_{i+m}$$

    - training (input, label) pairs:
      $\{(\{w_j | j \in \{(i-m), \ldots, (i+m)\} - \{i\}\}, w_i) \mid \}$

We can construct a fairly simple Neural Network (NN) to

- solve the word prediction task
- obtain dense vector embeddings (of length k)


$|V|$
C
k
Matrix Multiply
$\mathbf{x}^T C$
$(1 \times k)$
Logisitc Regression
$|V|$
Θ
k
$p(w_t|w_c)$
$|V|$
$\mathbf{w}_t$
1
$|V|$
$\mathbf{w}_c$
1
Denotes a trainable parameter

The NN solves a maximization problem

- Maximize average log probability over the T examples in training set:

$$\mathcal{V} = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log(p(w_{t+j} | w_t))$$

- Maximiziation problem; Find C, Θ that maximizes the (log) likelihood

$$C, \Theta = \underset{C,\Theta}{\operatorname{argmax}}\ \mathcal{U}$$

In words:

- solve for the matrix C, and logistic regression parameters Θ that maximizes probability of predicting correctly

Note

- dimension of C : $(|V| \times k)$
  - row i: maps $\mathbf{w}_i$ to a dense vector of length k
- dimension of Θ : $(|V| \times k)$
  - row i: regression weights for binary classification of target being $\mathbf{w}_i$
- Construct E after the fact:
  - the i-th row of E is θ_i

So both C and Θ can be used to create the dense vectors of length k

- in practice: average C and Θ

Tricks for training word2vec

Negative sampling (Noise Contrastive Estimation)

- Softmax denominator involves $|V|$ terms
    - impractical for large V
        - $|V|$ gradients to compute for each (example, distance) from center pair
    - Negative sampling
        - Re-state each logistic as a binary classifier
            - "Is/Is not" a neighbor within window m
                - rather than "what words w_t are neighbors"
            - So given sentence fragment of size $2m+1$:

$$w_{i-m}, \ldots, w_{i-1}, w_c, w_{i+1}, \ldots, w_{i+m}$$

        - training (input, label) pairs becomes :

$$\{([w_c, w_j], \text{True}) \mid j \in \{(i-m), \ldots, (i+m)\} - \{i\}\}$$

        - need to add k negative examples to training set (can't learn from just positiv examples)

$$\{([w_c, w_{n_j}], \text{False}) \mid 1 \leq j \leq k\}$$

            - an choose w_{n_j} at random
                - with high probability w_n won't be a neighbor
        - objective function component for w_c becomes
            - maximize probability of true neighbor w_t , minimize probability of false neighbor

$$k$$

$$\log(p(w_t|w_c)) - \sum_{j=1} \log(p(w_{n_k}|w_c))$$

- Note: this is equal to log of ratio of probability of true neighbors to false neighbors
    - appears in the GloVe examples

$$\log\left(\frac{p(w_t|w_c)}{\sum_{j=1}^{k} p(w_{n_k}|w_c)}\right)$$

Sub-sampling

- High frequency words don't carray as much information as low frequency words
- Sub-sampling: under sample high frequency words

Embeddings: Other topics

Shared Embeddings

- joint embedding of words and images
    - come up with code for image
- joint embedding of two languages
    - embedding for English, embedding for Chinese
    - know correspondence of some words between languages
        - add constraint that Chinise word embedding is close to tranlsated English word embedding

Sub-word representations

- "unfortunately" ↦ ["un", "foruntate", "ly"]
    - divide into morphemes
    - other divisions possible
        - n-gram of letters

Sentence representation

We will present two classes of sentence representation

- convert the sequence to a fixed length
- models that take sequences as inputs

Response: Sentence representation challenge

Fixed length representaion of a sentence

One way to deal with a sequence $\mathbf{w}$ of words is to it to a vector $\mathbf{x}^{(\mathbf{w})}$ of **fixed length**.

Once the length is fixed, Classical and Deep Learning models taking fixed length inputs can work as usual.

Doing so usually involves losing the ordering information.

Note that $\mathbf{w}$ is a sequence (vector) whose elements $\mathbf{w}_j$ correspond to words. The individual words $\mathbf{w}_j$ have been given a representation as a vector so $\mathbf{w}_j$ is a vector, not a scalar.

Bag of Words (BOW): Pooling

We define a *reduction* operation CBOW that convert a sequence of length $||w||$ to a fixed length.

The fixed length is usually $||V||$ the number of words in the Vocabulary, and hence the reduced form is called a Bag of Words (BOW).

There are many operators to achieve the reduction, which we will group under the name *pooling*

Sum/Average

$$\text{CBOW}(\mathbf{w}) = \sum_{j=1}^{||\mathbf{w}||} \mathbf{w}_j$$

Remember, w_j is a vector (length l) so $\text{CBOW}(\mathbf{w})$ is a vector that is the result of element-wise addition of vectors.

- $||\text{CBOW}(\mathbf{w})|| = l$

We can easily turn the Sum into an average by dividing by $||w||$

Count vectorization: a special case of pooling

When the representaion of a word $\mathbf{w}_j$ is a OHE vector of length $||V||$ then the Sum reduction returns a vector $\mathbf{x}^{(\mathbf{w})}$ whose j^{th} element $\mathbf{x}_j^{(\mathbf{w})}$ is the number of occurences of word $\mathbf{V}_j$ in sequence $\mathbf{w}$.

This is often called Count Vectorization. (turning $\mathbf{w}$ into a vector of counts).

TF-IDF

This technique turns a sentence of length n into an array (of length $|V|$) of (word, count) pairs.

A vocabulary has many words, some more "important" than others in conveying meaning.

For example, in the English language "the" does not convey much meaning.

But the string "ML" might be more meaningful.

Part of this has to do with frequency of word occurrence: "the" occurs so frequently that it doesn't have much meaning.

In general purpose English "ML" occurs much less often and can be recognized as an abbreviation for Machine Learning.

Term Frequency, Inverse Document Frequency (TF-IDF) is based on the idea that a word that is *infequent* in the wide corpus but is frequent in a particular document in the corpus is very meaningul in the context of the document.

So a document in which "ML" occured a disproportionately high (relative to the broad corpus) number of times is likely to indicate that the document is dealing with the subject of Machine Learning.

Note A similar idea is behind many Web search algorithms (Google).

TF-IDF is similar to the Count Vectorizer, but with modified counts that are the product of

- the frequency of a word within a single document
- the inverse of the frequency of the word relative to all documents
- v is a word
- d is a document (collection of words) in set of documents D

$$\begin{aligned}
\text{tf}(v, d) &= \text{frequency of word } v \text{ in document } d && \text{(Term Frquency)} \\
\text{df}(v) &= \text{number of documents that contain word } v \\
\text{idf}(v) &= \log(\tfrac{||D||}{\text{df}(v)}) + 1 && \text{Inverse Document Fr} \\
\\
\text{tf-idf}(v, d) &= \text{tf}(v, d) * \text{idf}(v)
\end{aligned}$$

To be concrete, let's write $l = ||\mathbf{w}_j||$ to denote the length of the word representation.

We will write $\mathbf{w}_{j,k}$ to refer to element k of the vector representing the j^{th} word $\mathbf{w}_j$.

This should be familiar to anyone using a programming language that represent multi-dimensional arrays as arrays containg elements that are arrays.

Variable length representation (sequence) of a sentence

Recurrent Neural Networks (RNN) and variants (LSTM)

- take variable length sequences as input
- fixed length output (final state)

They process the words in the sequence *in order* in order to arrive at the final state

- so final state is a representation of the entire sequence
- is of fixed length

Some sequence based models are *bi-directional*

- part of the model processes the sequence from start to end
- part of the model preocesses the sequence from end to start

This allows the model to extract greater meaning (particularly with some languages).

- English: The Big Apple
- French: Le Pomme Grand

Recurrent Neural Networks (and friends)

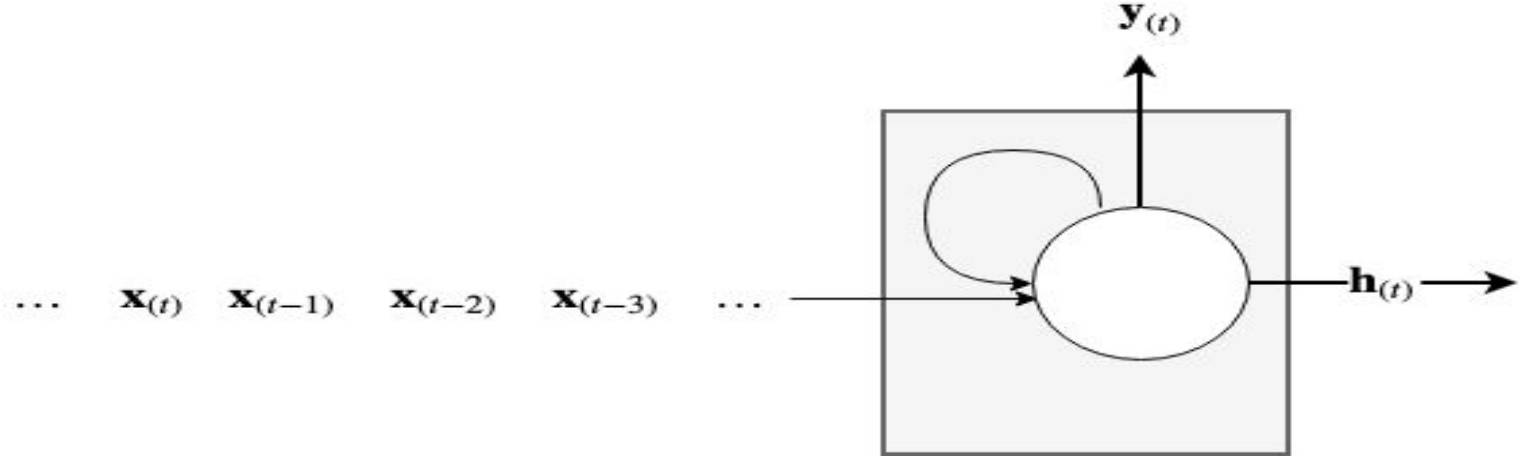
Recurrent Neural Networks (RNN) take *sequences* as inputs.

They are ideal for sentences of varying length.

There are advanced versions of RNN's

- LSTM
- GRU

We will use RNN generially.

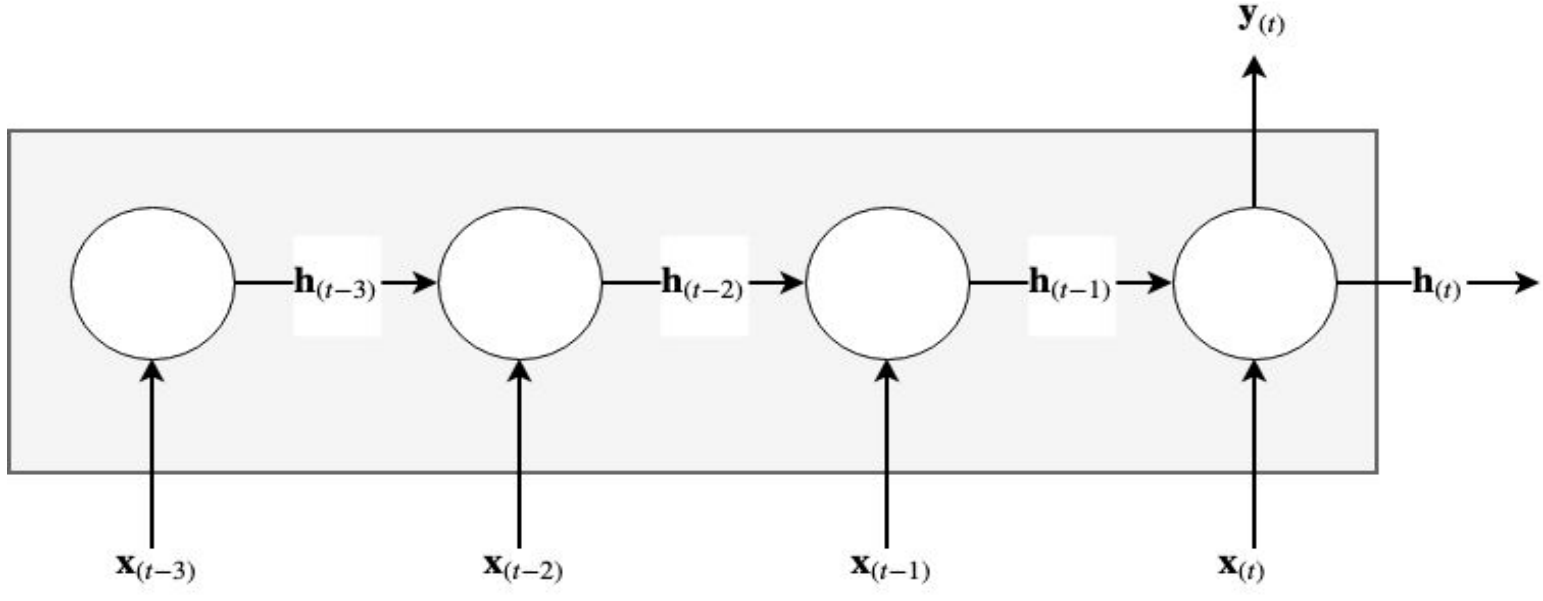

$\mathbf{y}_{(t)}$
$\mathbf{h}_{(t)}$
$\ldots$ $\mathbf{x}_{(t)}$ $\mathbf{x}_{(t-1)}$ $\mathbf{x}_{(t-2)}$ $\mathbf{x}_{(t-3)}$ $\ldots$

You can think of an RNN as a Neural Network in a loop:

- performs the same processing on each element of a sequence, one element at a time
- *has state*
    - encodes information on the prefix of the sequence encountered so far
    - so can change behavior on element t of the sequence, depending on elements $1 \ldots (t-1)$

It is helpful to "unroll" the loop and picture the RNN as a (variable length) sequence

RNN many to one



As you can see, the RNN at step t

- takes as inputs
    - $\mathbf{x_{(t)}}$, the t^th element of the sequence
    - $\mathbf{h}_{(t-1)}$, the prior state
        - creates $\mathbf{h_{(t)}}$, the new state

The final state $\mathbf{h}_{(n)}$, after processing the sequence of length n

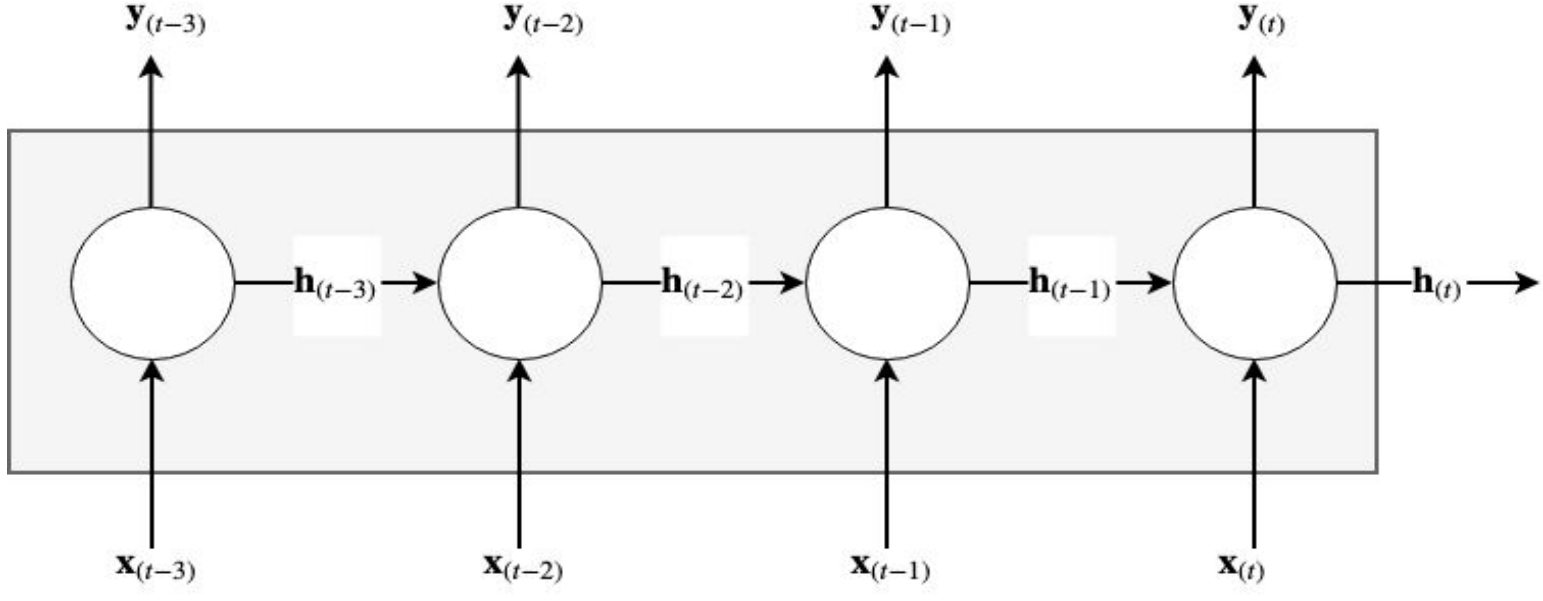
- is *an encoding of the entire sequence*
- may be used as a fixed length input to another model

This behavior is a *many to one* mapping.

One can also create a *many to many* mapping

- by making each hidden state $\mathbf{h_{(t)}}$ visible
- transforms input sequence to an output sequence of equal length

RNN many to many



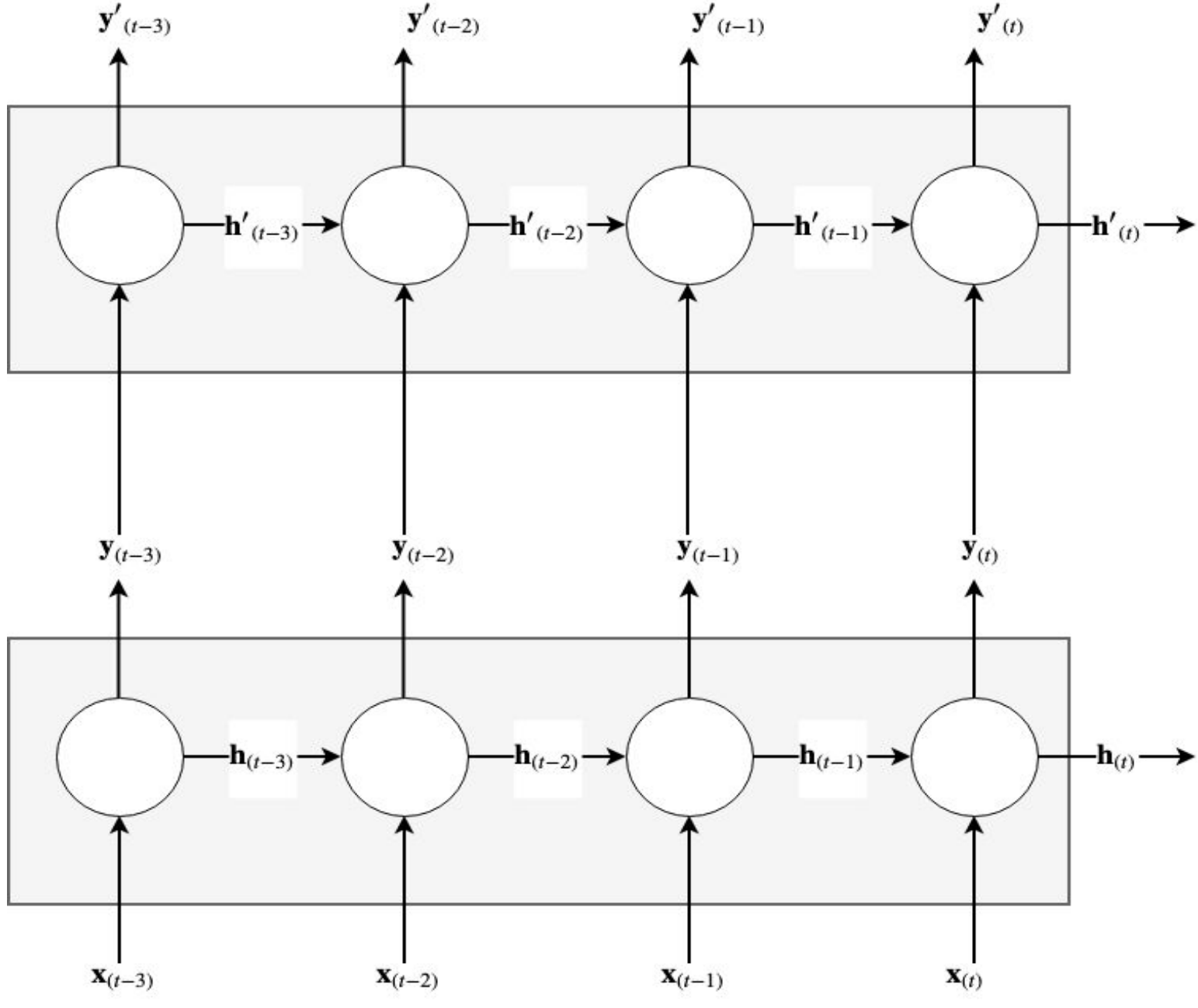
Producing a sequence is useful for several NLP tasks

- translating between languages
- captioning images

Encoder/Decoder architecture

An *Encoder/Decoder* is a two part Neural Network that is applied to many NLP tasks

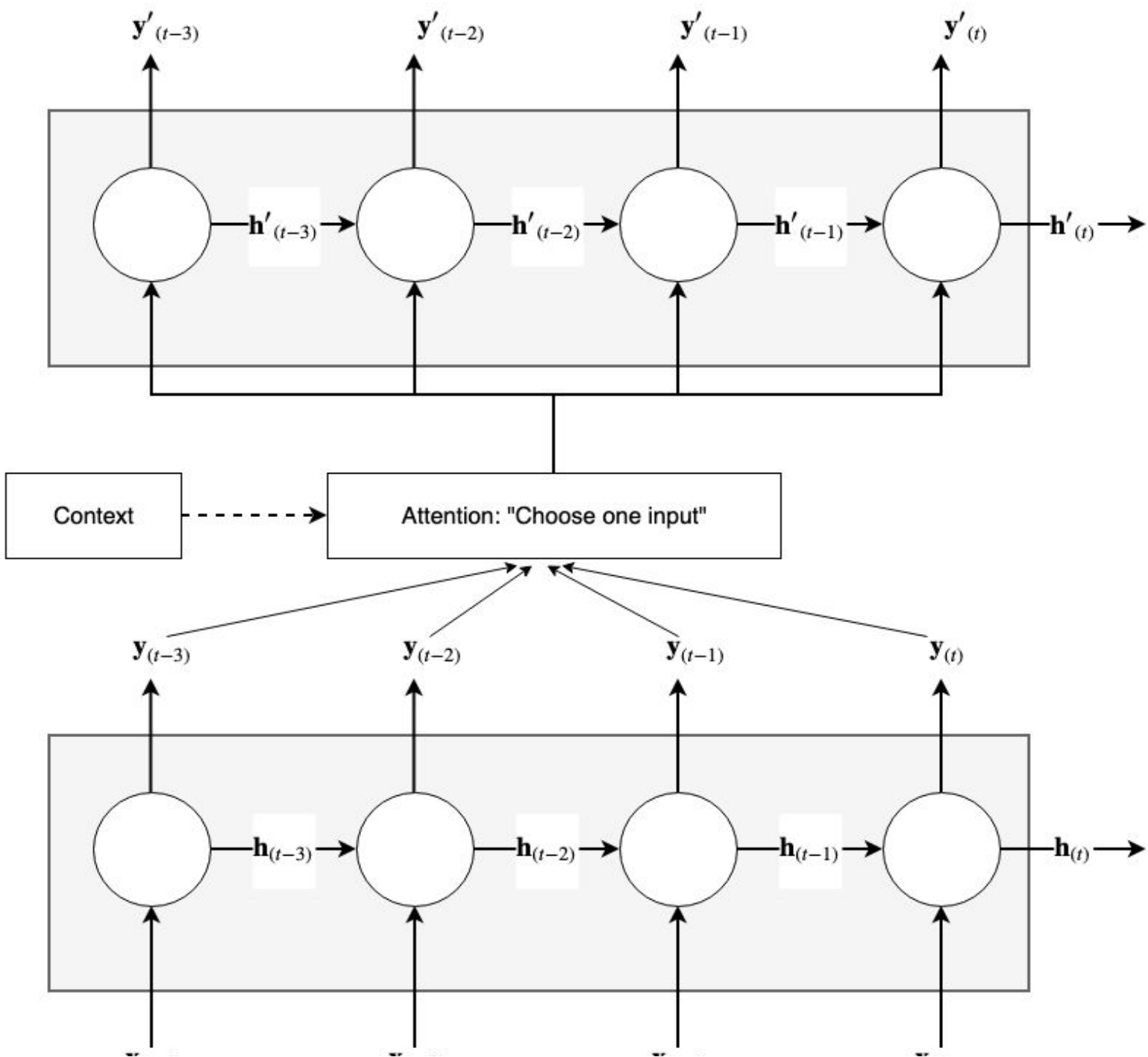
- *Encoder* converts sequence (sentence) into intermediate representation (sequence)
- *Decoder* converts intermediate sequence to final sequence


$\mathbf{y}'_{(t-3)}$
$\mathbf{y}'_{(t-2)}$
$\mathbf{y}'_{(t-1)}$
$\mathbf{y}'_{(t)}$
$\mathbf{h}'_{(t-3)}$
$\mathbf{h}'_{(t-2)}$
$\mathbf{h}'_{(t-1)}$
$\mathbf{h}'_{(t)}$
$\mathbf{y}_{(t-3)}$
$\mathbf{y}_{(t-2)}$
$\mathbf{y}_{(t-1)}$
$\mathbf{y}_{(t)}$
$\mathbf{h}_{(t-3)}$
$\mathbf{h}_{(t-2)}$
$\mathbf{h}_{(t-1)}$
$\mathbf{h}_{(t)}$
$\mathbf{x}_{(t-3)}$
$\mathbf{x}_{(t-2)}$
$\mathbf{x}_{(t-1)}$
$\mathbf{x}_{(t)}$

Examples

- Translating between languages
    - Encoder: encode source language
    - Decoder: decode into target language
- Image captioning
    - Encoder: encodes a stream of video frames
    - Decoder: generate description of the scene

Attention: an enhancement to sequence models


$\mathbf{y}'_{(t-3)}$
$\mathbf{y}'_{(t-2)}$
$\mathbf{y}'_{(t-1)}$
$\mathbf{y}'_{(t)}$
$\mathbf{h}'_{(t-3)}$
$\mathbf{h}'_{(t-2)}$
$\mathbf{h}'_{(t-1)}$
$\mathbf{h}'_{(t)}$
Context
Attention: "Choose one input"
$\mathbf{y}_{(t-3)}$
$\mathbf{y}_{(t-2)}$
$\mathbf{y}_{(t-1)}$
$\mathbf{y}_{(t)}$
$\mathbf{h}_{(t-3)}$
$\mathbf{h}_{(t-2)}$
$\mathbf{h}_{(t-1)}$
$\mathbf{h}_{(t)}$

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

```
Done
```