Deep Learning in Natural Language Processing

02/12/2019

Andrea Zugarini, PhD Student
Marco Maggini, Advisor

Outline

- Introduction
- Word Representations
- Language Models
- Recurrent Neural Networks

Introduction

- Natural Language Processing (**NLP**) is the field in between computer science, statistics and linguistics.

- **Goal**: Design of algorithms that automatically **understand** human language in order to perform some tasks. E.g.:

  Spell checking, translation etc…

- Understanding the **meaning** of language is a challenging problem!

NLP Problems and Applications

- **Syntactic Analysis**
  - *Pos-tagging, Chunking*
- **Semantic Interpretation**
  - *Sentiment Analysis, Named Entity Recognition, Entity and Relation Extraction, Word Sense Disambiguation, ...*
- **Discourse Processing**
  - *Language Modelling, Machine Translation, Text Summarization, ...*

Growing **complexity** of such applications!

Rule-based or **statistical** approaches are **not adequate** to face the complexity of most of these challenges.

We need a different perspective!

Machine Learning and NLP

Alternatively, we could try to let computers learn directly from real **data** !!!

Machine Learning (ML) methods exploit **examples** to learn how to solve a problem.
The goal is to assign a **label** to each example.
An example is represented into the computer with **features**.
If we know the labels for some data, we can do **supervised learning**.

Many **NLP** problems can be formulated as **classification** problems.

Warning: Features are usually real numbers, while language is inherently **symbolic**! We need to handcraft ad-hoc solutions for each task.

Machine Learning and NLP

An Example: Sentiment Analysis

Goal: Decide whether a text is expressing a **positive** or **negative** opinion about something.

Plenty of useful applications, because people's feedbacks are important:

tripadvisor, e-commerce, movies….

It is a typical **binary** classification problem!

Naïve approach (see SentiWordNet):

- Define two sets of words with positive (*love*, *like*, *appreciated* …) and negative (*hate*, *lame*, *bad* …) meaning, respectively
- **Count** how many times each word in the sets appears in a text (our features)
- Feed a classifier with these features and make it learn.

Machine Learning and NLP

An Example: Sentiment Analysis

Review 1

My boyfriend and I went to watch The Guardian.At first I didn't want to watch it, but I loved the movie. It was definitely the best movie I have seen in sometime. I would suggest this movie for anyone to see. The ending broke my heart but I know why he did it.

positive

Review 2

For years, I've been a big fan of Park's work and "Old boy" is one of my all-times favorite. With lots of expectation I rented this movie, only to find the worst movie I've watched in awhile. It's not a proper horror movie; there's no suspense in it and even the "light" part is so lame, that I didn't know whether to laugh or cry. For me, an idol has fallen. If you loved movies like "Old boy", the Mr & Lady "Vengeance" , don't waste your time, the film's not worth it.

negative

Looks easy, right?

Machine Learning and NLP

An Example: Sentiment Analysis

Review 1

*My boyfriend and I went to watch The Guardian.At first I didn't want to watch it, but I **loved** the movie. It was definitely the **best** movie I have seen in sometime. I would suggest this movie for anyone to see. The ending **broke my heart** but I know why he did it.*

positive

Review 2

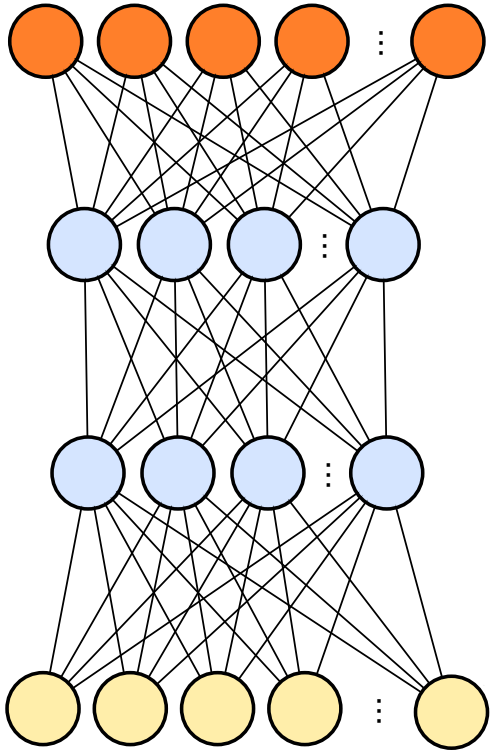
*For years, I've been a **big fan** of Park's work and "Old boy" is one of my all-times favorite. With lots of expectation I rented this movie, only to find the **worst** movie I've watched in awhile. It's not a proper horror movie; there's no suspense in it and even the "light" part is so **lame**, that I didn't know whether to laugh or cry. For me, an idol has fallen. If you **loved** movies like "Old boy", the Mr & Lady "Vengeance" , don't waste your time, the film's not worth it.*

negative

Not for a computer, positive words appear in negative reviews and viceversa!

What is Deep Learning?

- Is a subfield of Machine Learning!!!
- In **contrast** to standard ML, deep learning models automatically learn good features (or representations) from **raw input data**.
- **Neural Networks** are the dominant class of models in Deep Learning.
- How can automatically learn meaningful representation of text in NLP?



Words Representations

Words Representations

Words are **discrete symbols**.

Machine-Learning algorithms cannot process symbolic information as it is.

Same problem of any **categorical variable**, e.g.:

- Blood type of a person: {*A, B, AB, O*}
- Color of a flower: {*yellow, blue, white, purple, ...*}
- Country of citizenship: {*Italy, France, USA, ...*}

So, given the **set** of possible values S of the feature, the solution is to define an assignment function $H : S \rightarrow \mathbb{R}^d$ to map each symbol into a **real vector**.

One-hot Encoding

Without any other assumption, best way is to assign symbols to **one-hot** vectors, such that all the nominal values are **orthogonal**.

In Blood type example:

A: [1 0 0 0]
B: [0 1 0 0]
AB: [0 0 1 0]
O: [0 0 0 1]

Warning: the length *d* of the representation grows linearly with the cardinality of *S*.

In NLP, words are mapped to one-hot vectors with the size of the vocabulary.

One-hot Encoding

Given a vocabulary of 5 words *V*= {*hotel, queen, tennis, king, motel*}:

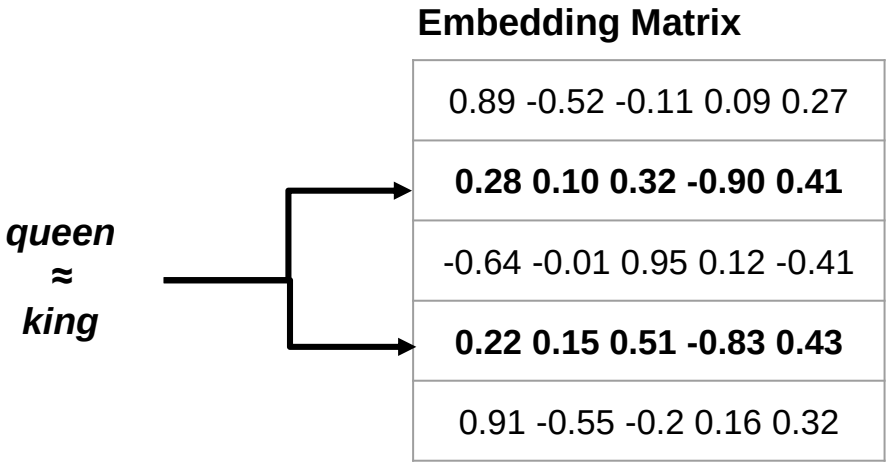
hotel: [1 0 0 0 0]
queen: [0 1 0 0 0]
tennis: [0 0 1 0 0]
king: [0 0 0 1 0]
motel: [0 0 0 0 1]

There is no notion of **similarity** between one-hot vectors!

queen: [0 **1** 0 0 0]
king: [0 0 0 **1** 0]
hotel: [**1** 0 0 0 0]

Word Embeddings

The idea is to assign each word to a **dense vector** with $d << |V|$, chosen such that similar vectors will be associated to words with similar meaning.
We must define an embedding matrix of size $|V| \times d$.
Each row is the embedding of a single word.



Word Embeddings

Word2vec

There are literally hundreds of methods to create dense vectors, however most of them are based on ***Word2vec*** framework (Mikolov et al. 2013).

Intuitive idea

> ***"You shall know a word by the company it keeps"*** *(J. R. Firth 1957)*

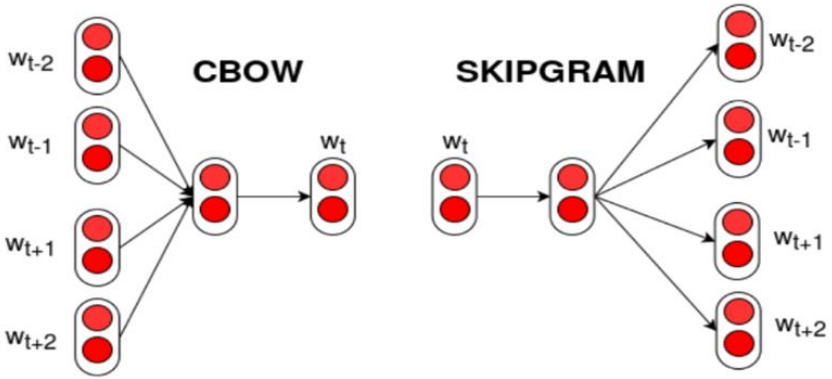
In other words, a **word's meaning** is given by the words in the **context** where it usually appears.

One of the most successful ideas in Natural Language Processing! Embeddings are learnt in an **unsupervised** way.

Word Embeddings

Word2vec

- Consider a large corpus of text (billions of words).
- Define a vocabulary of words and associate each word to a row of the embedding matrix initialized at **random**.
- Go through each position in the text, which has a **center** word and a **context** around it (fixed **window**). Two conceptually equivalent methods:
  - (**CBOW**) Estimate the probability of the center word given its context.
  - (**SKIPGRAM**) Estimate the probability of context given the center word.
- Adjust word vectors to **maximize** the probability.



Word Embeddings

Issues

Results are impressive, but keep in mind that there are still open points:

- **Multi-sense** words. There are words with multiple senses. E.g. **bank**:
  "Cook it right on the ***bank*** of the river"
  "My savings are stored in the ***bank*** downtown"

- **Fixed size** vocabulary, i.e. new words are not learned
- **Out Of Vocabulary** words are represented with the same dense vector.
- No information about sub-word structure, so **morphology** is completely unexploited.

Possible solutions:

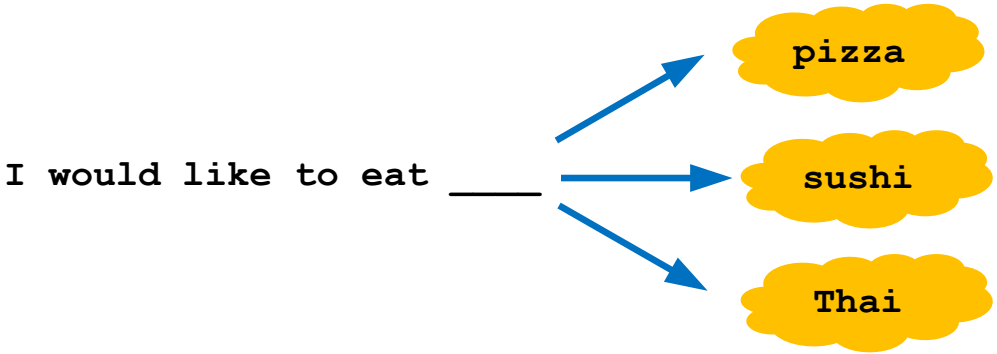
- Multi-sense word embeddings
- Character-based word representations

However, **Word2vec** embeddings work pretty well for common tasks such as Language Modeling.

Language Modeling

Language Modeling

Language Modelling is the problem of **predicting** what word comes next



Formally, a sentence of words $w_1, \ldots, w_n$ is characterized by a **probability distribution**:

$$p(w_1, \ldots, w_n) = p(w_1)p(w_2|w_1)\ldots p(w_n|w_1, \ldots, w_{n-1})$$

where, the equivalence comes directly from the chain rule.

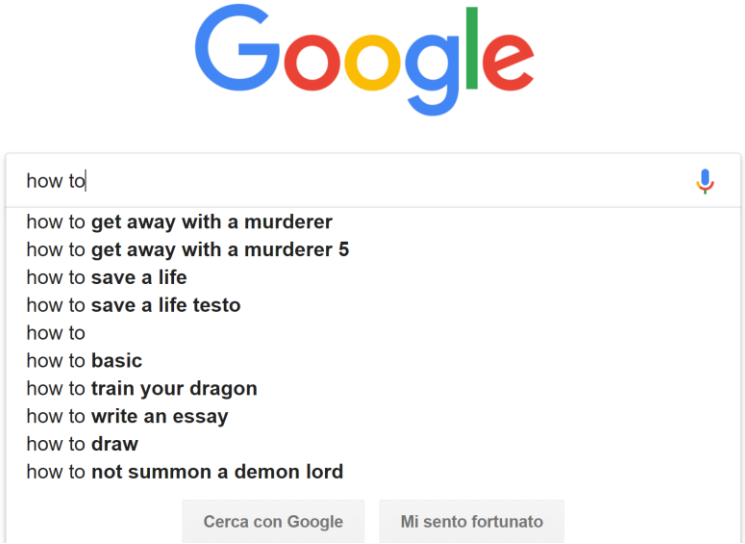
Motivation

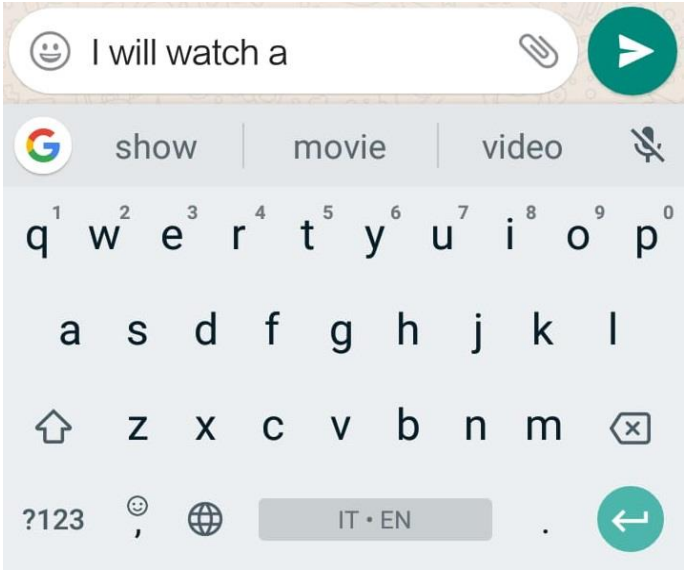
Language Modeling is considered a benchmark to evaluate progresses on language understanding.

LMs are involved on several NLP tasks:

- **Language Generation**
- **Representation Learning**
- Speech Recognition
- Spell Correction
- Machine Translation
- ….

Some Examples





N-gram Language Models

How to estimate $p(w_t|w_1,\ldots,w_{t-1})$? Just learn it from **observations**!

1) Get a **huge** collection of textual documents.
2) Retrieve the set ***V*** of all the words in the corpora, known as Vocabulary.
3) For **any** sub-sequence of words, **estimate** $p(w_t|w_1,\ldots,w_{t-1})$ by counting the number of times w_t appears in context $w_1 \ldots w_{t-1}$ over the number of times the context appeared overall, i.e.:

$$p(w_t|w_1,\ldots,w_{t-1}) = \frac{\#(w_1,\ldots,w_{t-1},w_t)}{\sum_{w_i \in V} \#(w_1,\ldots,w_{t-1},w_i)}$$

Easy, right?

N-gram Language Models

Considering **all the possible sub-sequences** is infeasible in terms of computation and memory.

N-gram models approximate $p(w_t|w_1,\ldots,w_{t-1})$ assuming:

$$p(w_t|w_1,\ldots,w_{t-1}) \approx p(w_t|w_{t-N+1},\ldots,w_{t-1})$$

When **N** increases, approximation is more precise, **but** complexity grows exponentially.

Viceversa, when **N=1**, uni-gram models requires few resources but performances are poors.

Bi-grams are usually a good tradeoff.

N-gram Language Models

Limitations

- N-gram models **do not generalize** to unseen word sequences, that is partially alleviated by **smoothing** techniques. The longer the sequence, the higher the probability to discover an unseen one.
- Despite the choice of **N**, it will always be bounded.
- The exponential complexity of the model limits **N** to be rather small (usually 2 or 3) that leads to not good enough performances.

How about using a *Machine-Learning model*?

Neural Language Model

Fixed Window (Bengio et al. 03)

Neural Networks require a **fixed-length** input. Hence, we need to set a window of words with length **N**.

Concatenated word embeddings of the last **N** words are the input of an MLP with one hidden layer.

Advantages over N-gram models:

- Neural networks have better **generalization** capabilities => NO SMOOTHING required.
- Model size increases **linearly** O(N), not exponentially O(exp(N)).

Still open **problems**:

- History length is **fixed**.
- Weights are **not shared** across the window!

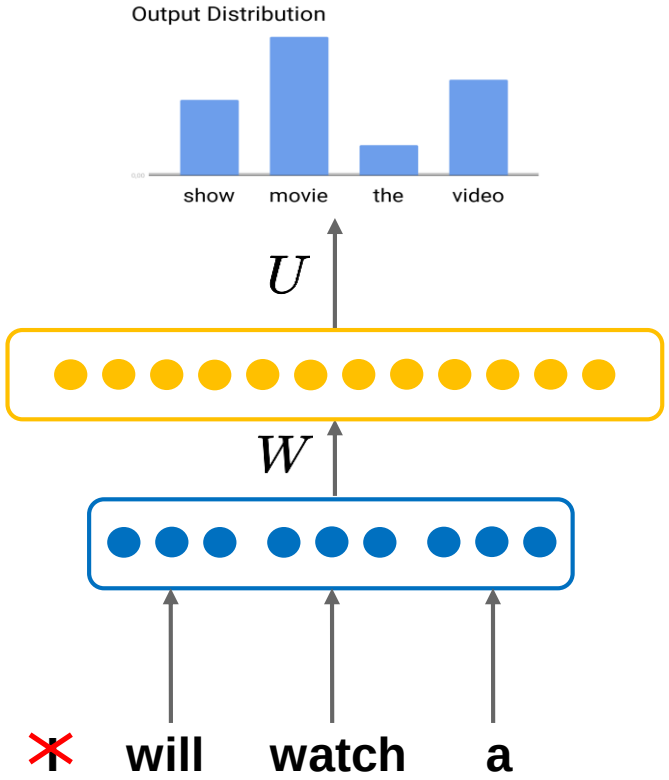
Neural Language Model

Fixed Window (Bengio et al. 03)

$$\hat{y} = softmax(Uh + b_2) \in \mathbb{R}^{|V|}$$

$$h = \sigma(We + b_1)$$

$$e = [e^{(t-N)}, \ldots, e^{(t-1)}]$$



Case of window size **N=3.** Only the last 3 words are taken into account.

Recurrent Neural Networks

Recurrent Neural Networks

Feedforward networks just define a *mapping* between inputs to outputs. This behaviour does not depend on the order in which inputs are presented. Time is then not considered, that's why feedforward networks are said to be **static** or **stationary**.

$$Y = f(X)$$

Recurrent Neural Networks (RNN) are a family of architectures that **extend** standard feedforward neural networks to process input sequences, in principle, of **any length**. They are also known as **dynamic** or **non-stationary** networks. Patterns are sequences of vectors.

$$Y(t) = f(X(t))$$

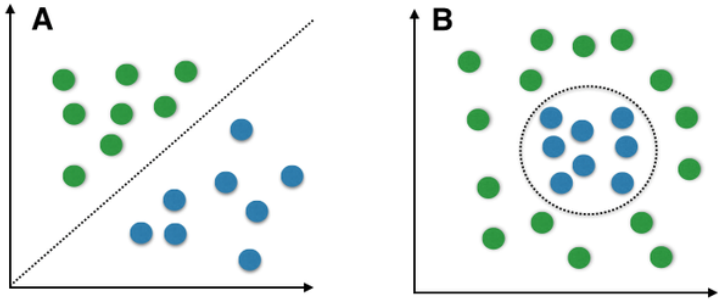
Recurrent Neural Networks

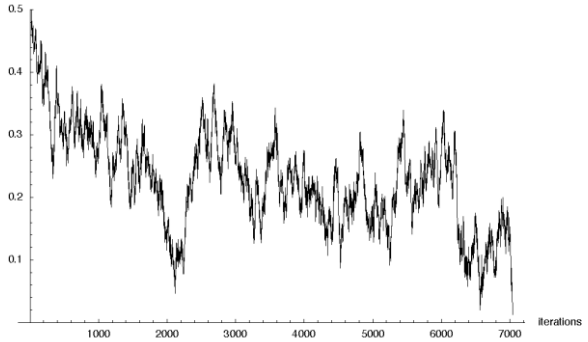
Feedforward Networks

- It models static systems.
- **Good** for traditional classification and regression tasks.

Recurrent Networks

- Whenever there is a temporal dynamic on the patterns.
- **Good** for Time series, Speech Recognition,**Natural Language Processing**, etc…





Recurrent Neural Networks

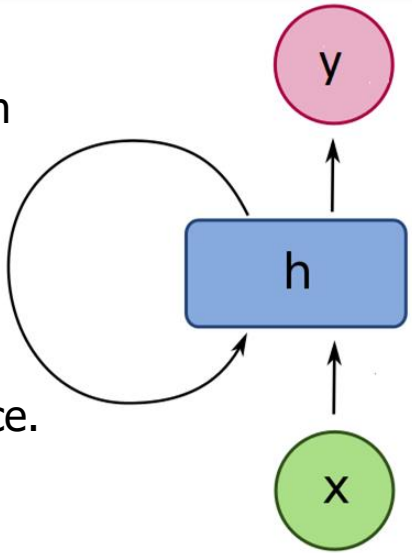
Two functions ***f*** and ***g***, compute the hidden state and the output of the network, respectively.

A pattern x is a sequence of vectors:

$$x = \{x_1, \ldots, x_T\}$$

The hidden state has **feedback connections** that passes information about the past to the next input.

$$h_t = f(W_h \cdot h_{t-1} + W \cdot x_t + b_1)$$



Output can be produced at any step or only at the end of the sequence.

$$y_t = g(U \cdot h_t + b_2)$$

Learning in Recurrent Networks

Backpropagation Through Time

How to train **RNNs** ?

Feedback connections creates **loops**, that are a problem since the update of a weight depends on itself at previous time step.

Solution: a recurrent neural network processing a sequence of length ***T*** is **equivalent** to a feedforward network obtained by the **unfolding** of the RNN ***T*** times.

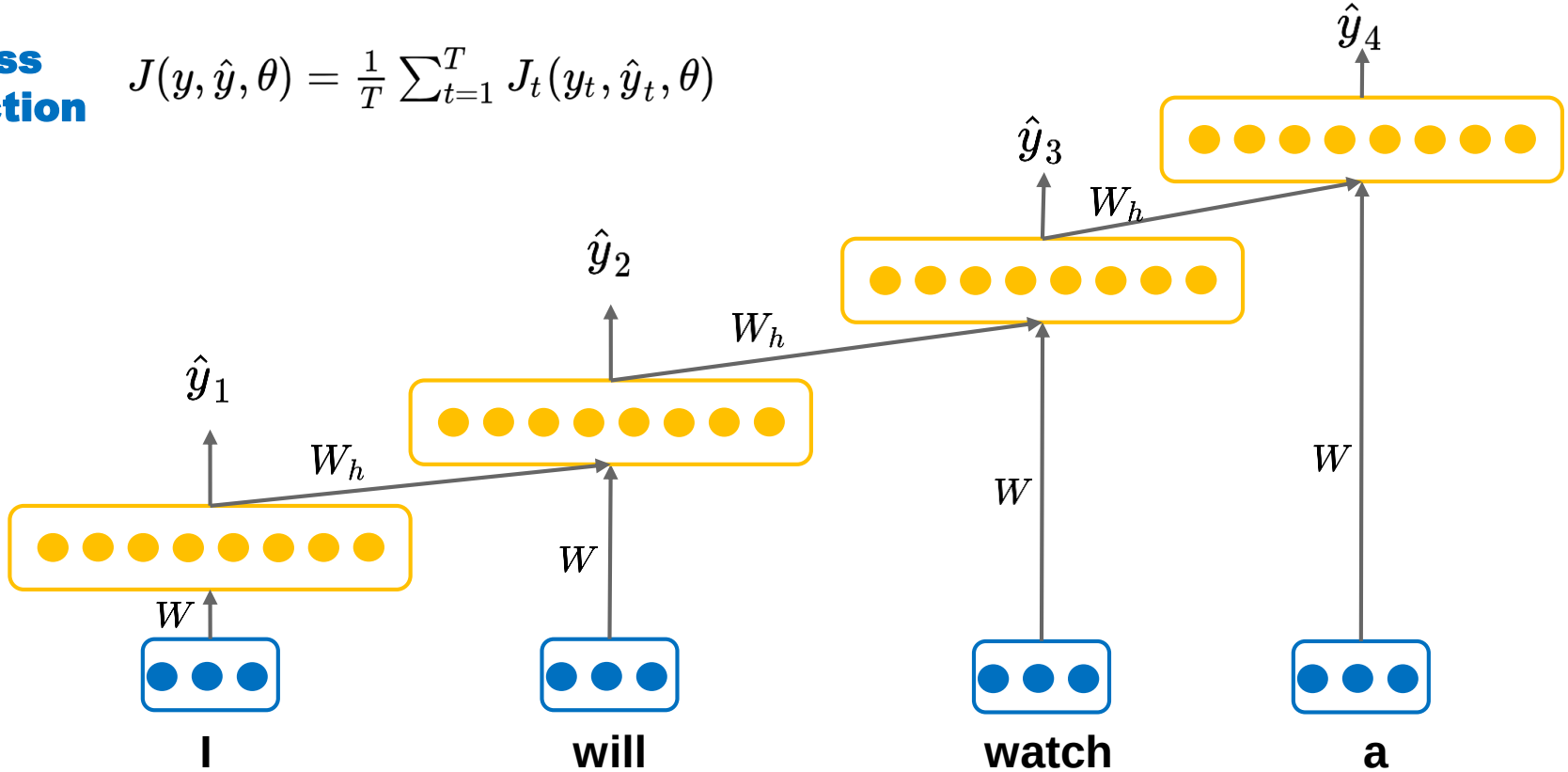
The unfolded network is trained with standard **backpropagation** with **weight sharing**.

Learning in Recurrent Networks

Unfolding through time

Loss Function

$$J(y, \hat{y}, \theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(y_t, \hat{y}_t, \theta)$$



Learning in Recurrent Networks

Vanishing Gradient Problem

Sequences can be much longer than the one seen in the examples.

When the sequences are too long gradients steps tends to **vanish**, because the squashing functions have gradient always < 1.
So learning **long-term dependencies** between inputs of a sequence is difficult (Bengio et al. 1994).

Intuitive Idea
RNNs have problems to remember information coming from very **old past**.

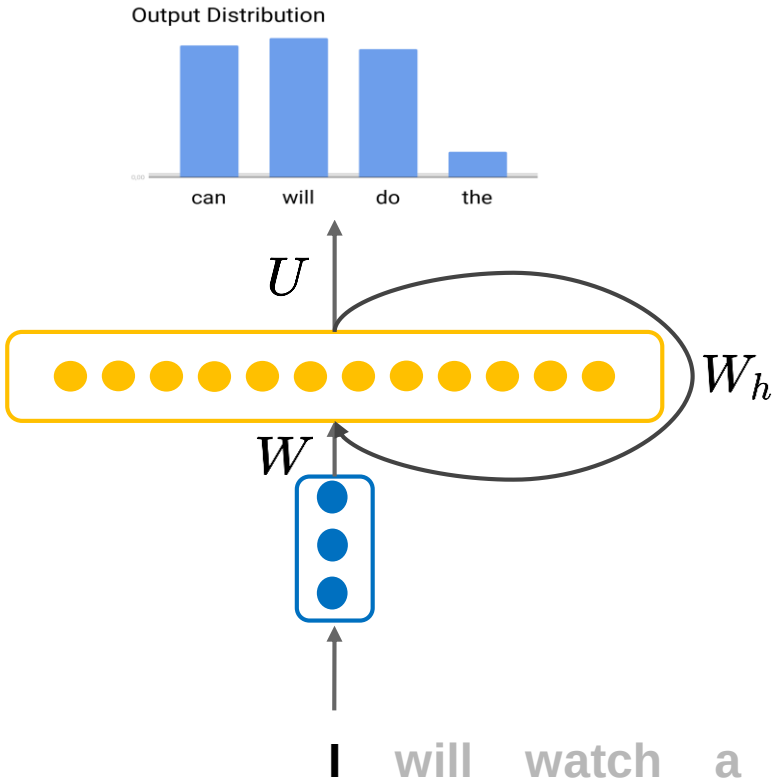
Learning in Recurrent Networks

Vanishing Gradient Problem

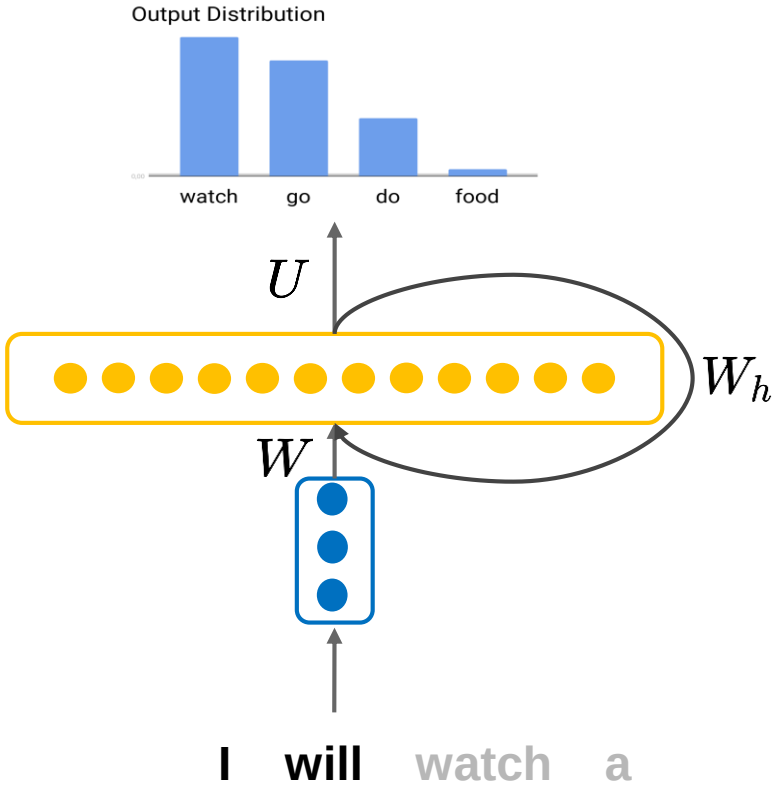
There are ways to **alleviate** this issue:

- Use of **ReLu** activation functions, but there is the risk of gradient exploding (opposite problem).
- Good initialization of the weights (e.g. **Xavier**), always a best practice.
- Other variants of recurrent networks Long-Short Term Memory (**LSTM**) networks, Gated Recurrent Units (GRU), have been designed precisely to mitigate the problem.

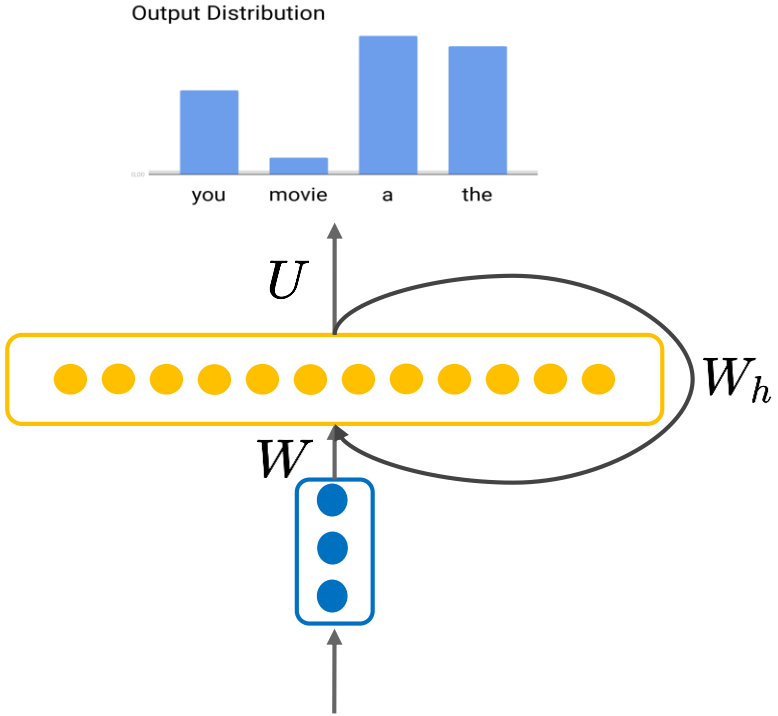
RNN Language Model



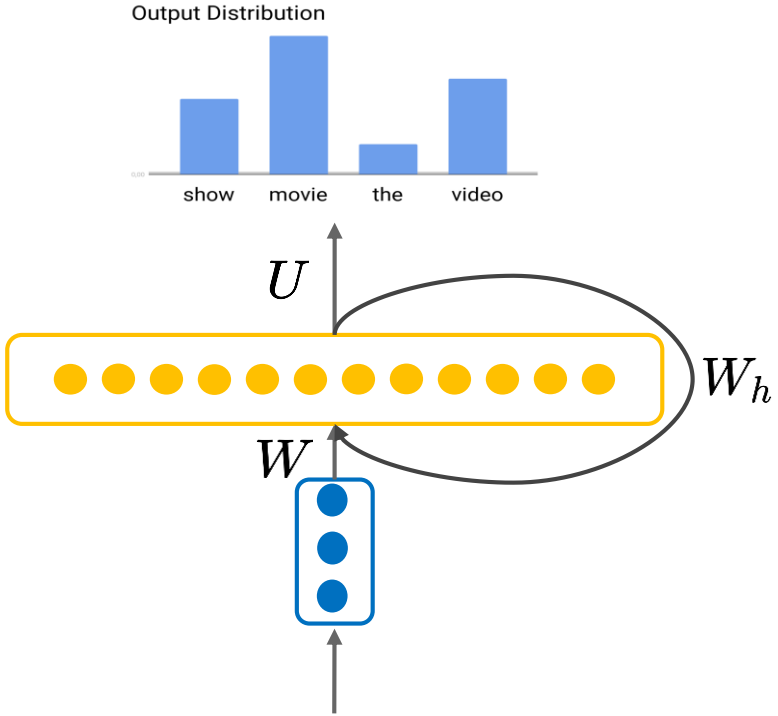
RNN Language Model



RNN Language Model



RNN Language Model



Language Modeling

Comparison

n-gram model →

Increasingly complex RNNs ↓

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

↓ Perplexity improves (lower is better)

Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

References

- Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 2003.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint* 2013.
- Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994.
- S Hochreiter, J Schmidhuber *Long short-term memory.*
- Material on Deep Learning in NLP, http://web.stanford.edu/class/cs224n/syllabus.html .