MIDDLE EAST TECHNICAL UNIVERSITY

MMI711 Sequence Models in Multimedia

LSTM & Introduction to Deep NLP

This week:

- We will first talk about LSTM a hand-crafted RNN design.
- Then we will delve into Deep NLP, start with Word Embeddings
- After a brief summary of LSTM encoder/decoder architectures, we will practice some live code.

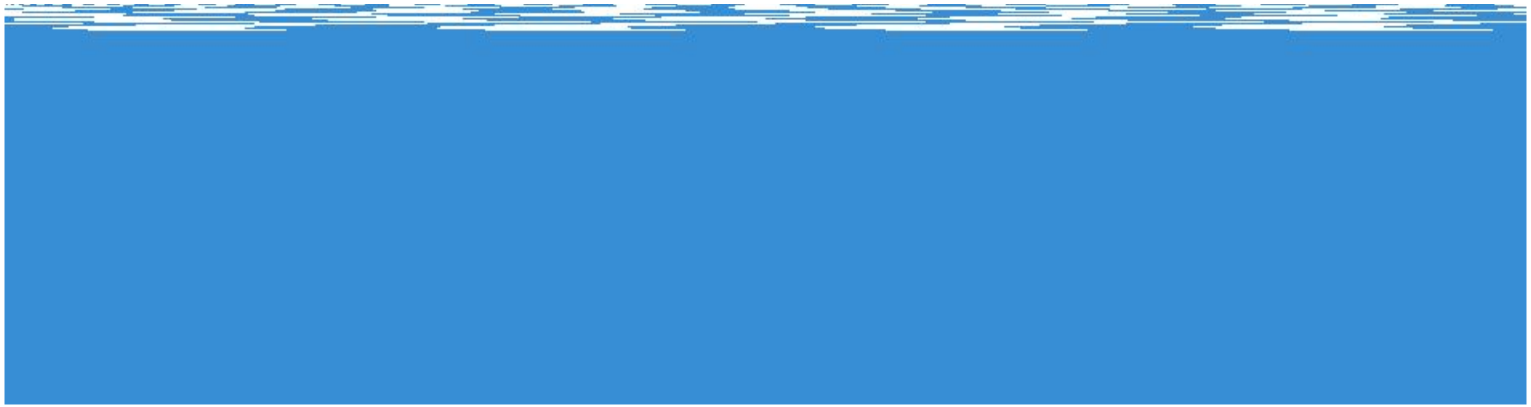
METU

from RNNs to Gated Recurrent Units

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- We will start with LSTM (1997), then continue with GRU (2014), in chronological order.
- Although GRU is newer (2014), it is architecture-wise simpler compared to the original LSTM (1997).
- So once again, RNNs in a nutshell:

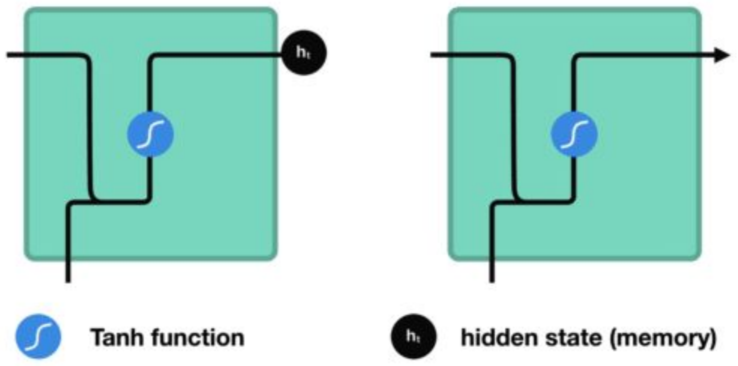


from RNNs to Gated Recurrent Units

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- RNNs, while processing, passes the previous hidden state to the next step of the sequence. The hidden state acts as the neural networks memory. It holds information on previous data the network has seen before



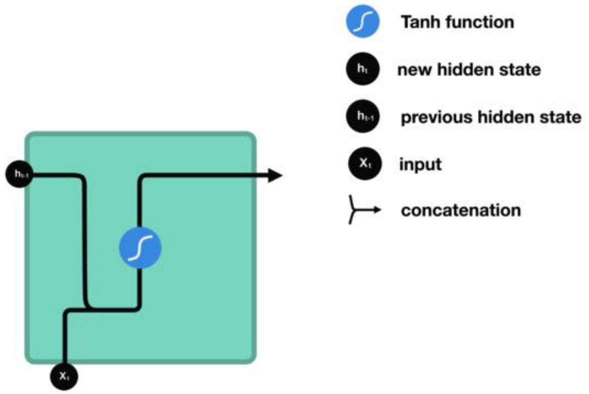
METU

from RNNs to Gated Recurrent Units

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- The input and previous hidden state are combined to form a vector.
- That vector now has information on the current input and previous inputs.
- The vector goes through the tanh activation, and the output is the new hidden state, or the memory of the network.



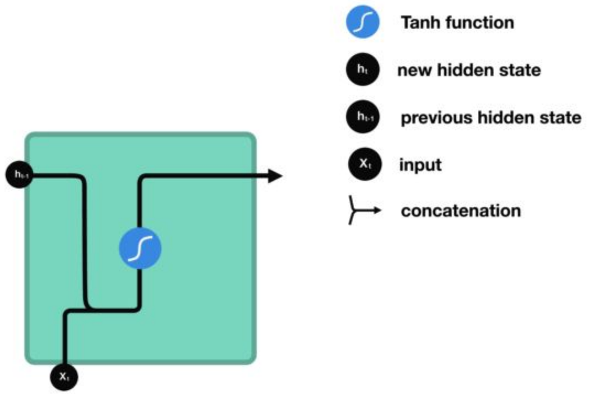
METU

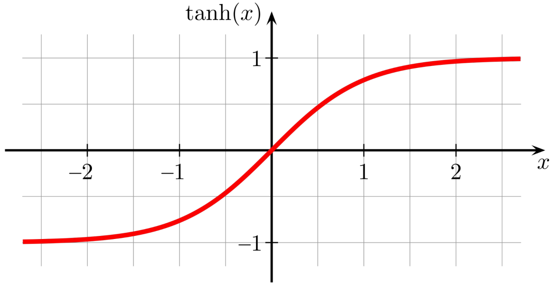
from RNNs to Gated Recurrent Units

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- Vanilla RNNs traditionally use $tanh$ as the activation function.
- Becasue the $tanh$ activation regulates the values flowing through the network.
- The $tanh$ function squishes values to always be between -1 and 1.





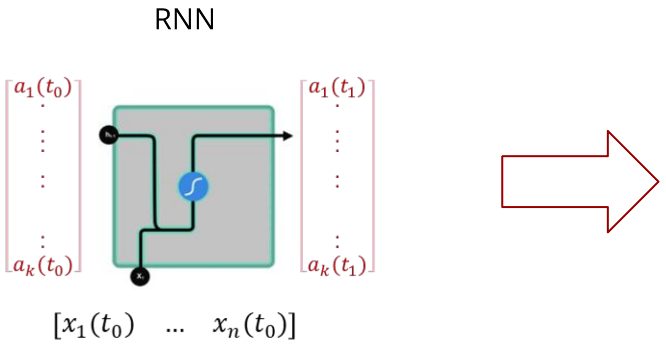
METU

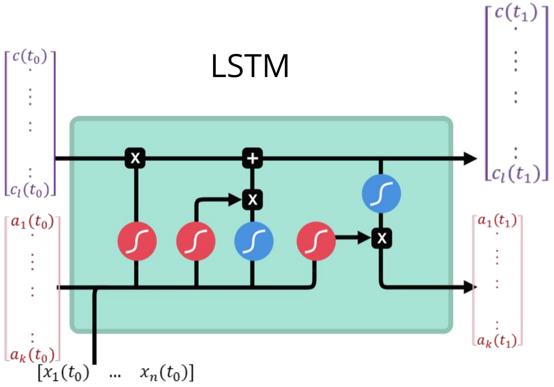
LSTM

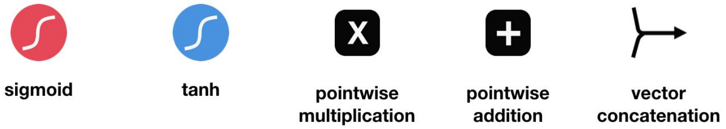
I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- The problem is (unlike ResNet) there is no identity function (i.e. the highway).
- How to add it: the answer is LSTM.
- An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells.





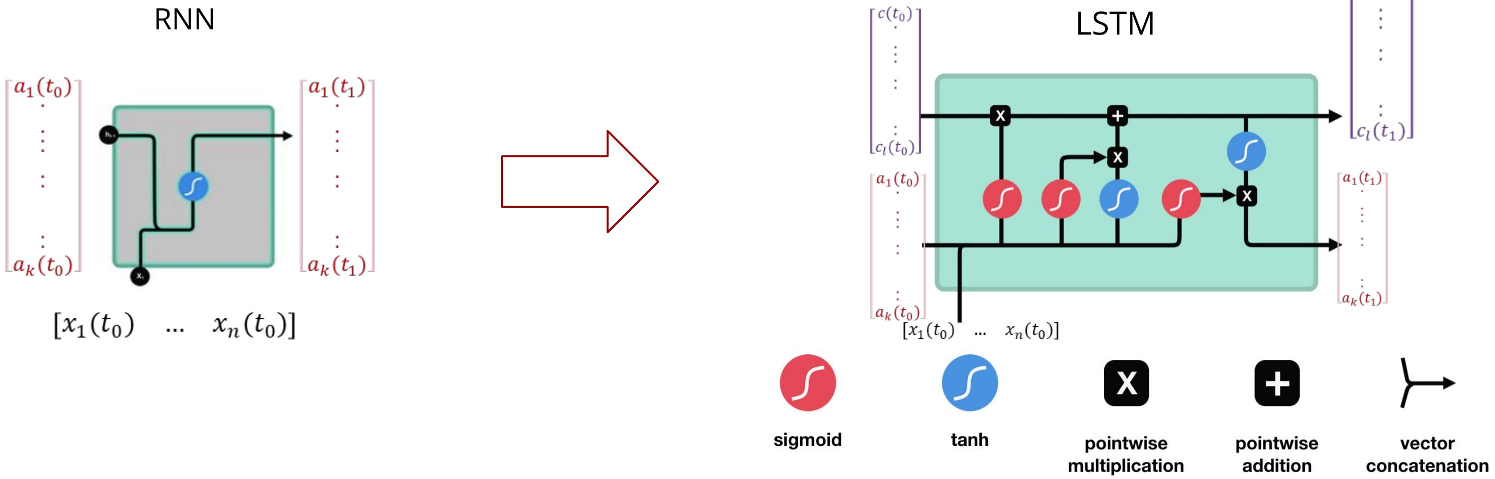


LSTM

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- Similar to the so-called highway, or the identity operation of ResNet, LSTMs have multiple mechanisms to control the information flow.
- These mechanisms are used to allow the LSTM to <u>**keep**</u> or <u>**forget**</u> information.

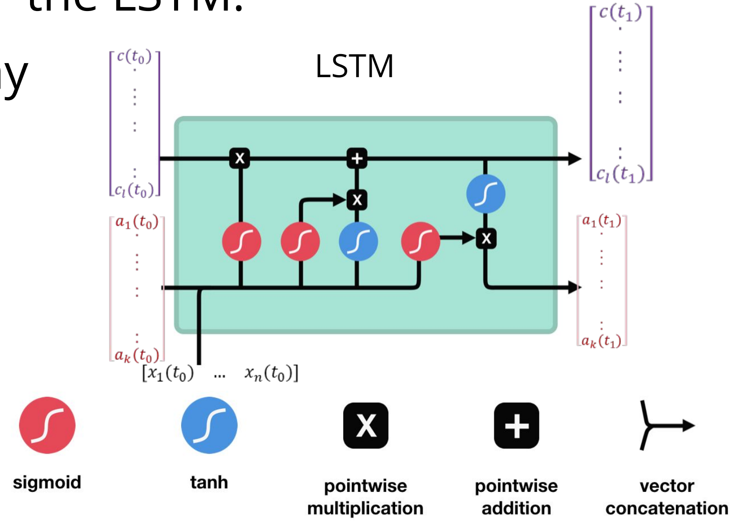


LSTM

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- **In addition to** hidden states, we have a concept of cell states in LSTMs,
- Hidden states of an LSTM are pretty much similar to hidden states of an RNN
- Cell states are related to the «gates» of the LSTM.
- The cell state act as a transport highway that transfers relative information all the way down the sequence chain.
- As the cell state goes on its journey, information gets added or removed to the cell state via gates.

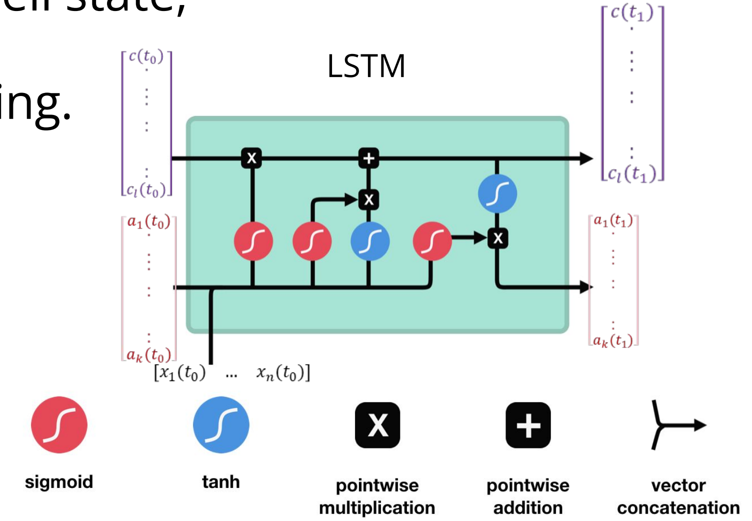


LSTM

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- Cell states are **long-term memory** of an LSTM
- Hidden states are the **working** (short-term) **memory.** They exist both in LSTM and RNN models
- The gates of an LSTM is to create the cell state, which can learn what information is relevant to keep or forget during training.

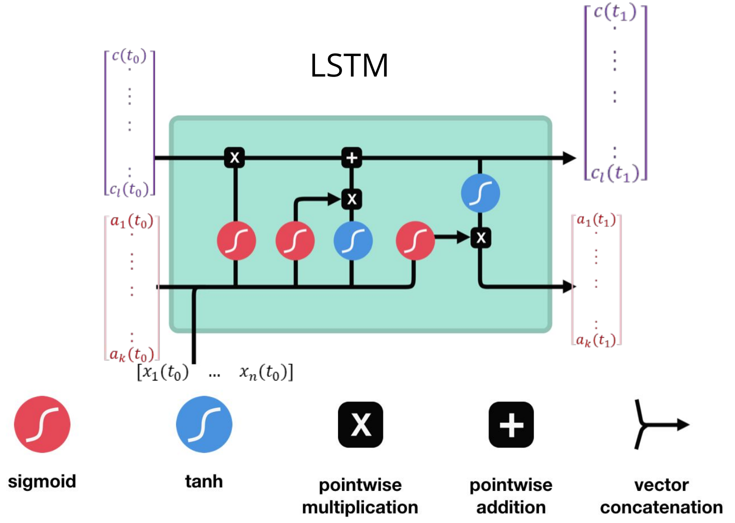


LSTM gates

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- In order to understand how the gates of an LSTM function, let's first remember $tanh$ and $sigmoid$ functions
- -1 $< tanh(x) <$ +1
- 0 $\leq sigmoid\ (x) <$ +1
- $sigmoid$ squishes values between 0 and 1. That is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappears or be "*forgotten.*"
- On the other hand, any number multiplied by 1 is the same value therefore that value stay's the same or is "*kept.*"



METU

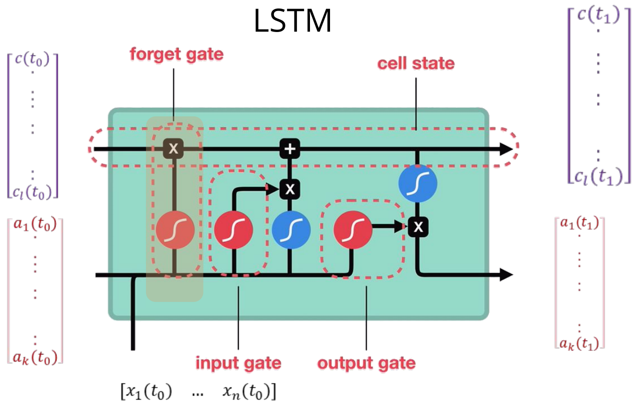
LSTM "forget" gate

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- This gate decides what information should be thrown away (forgotten) or kept.

- Information from the previous hidden state and information from the current input is passed through the sigmoid function.
- Values come out between 0 and 1.
- The closer to 0 means to forget, and the closer to 1 means to keep.



METU

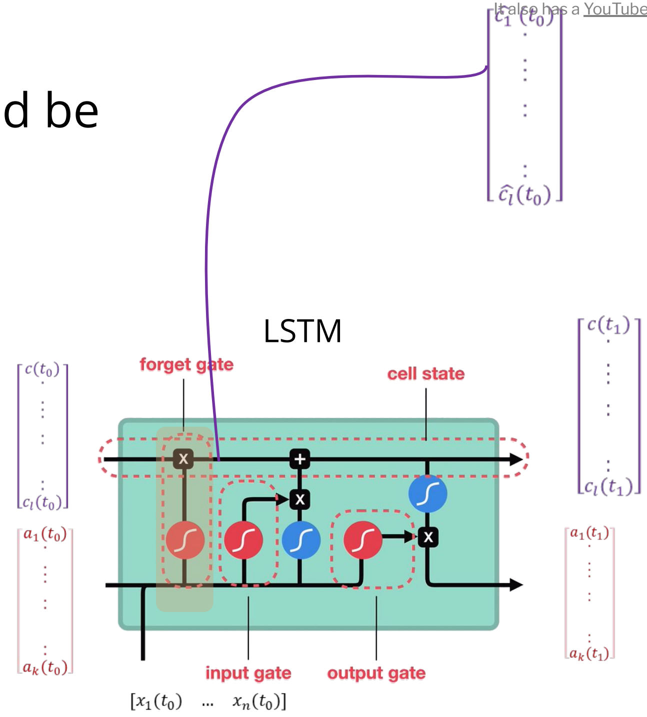
LSTM “forget” gate

- This gate decides what information should be thrown away (forgotten) or kept.
  - Forget gate, like all gate is a separate layer with individual weights.
  - $\hat{\mathbf{c}}_{l\times1} = \mathbf{c}_{l\times1} \odot \sigma\left(\mathbf{W}^{forget}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times1}(t_0)\\ \\ \mathbf{a}_{k\times1}(t_0)\end{bmatrix} + \mathbf{b}^{forget}_{l\times1}\right)$

    $\odot \rightarrow element - wise\ multiplication$
  - The gate outputs are vectoral, not scalar, so the updates on cell states are element-wise.
  - Some aspects of the cell state are forgotten (or kept), not all of them together.

I will be using the figures from this guy’s very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

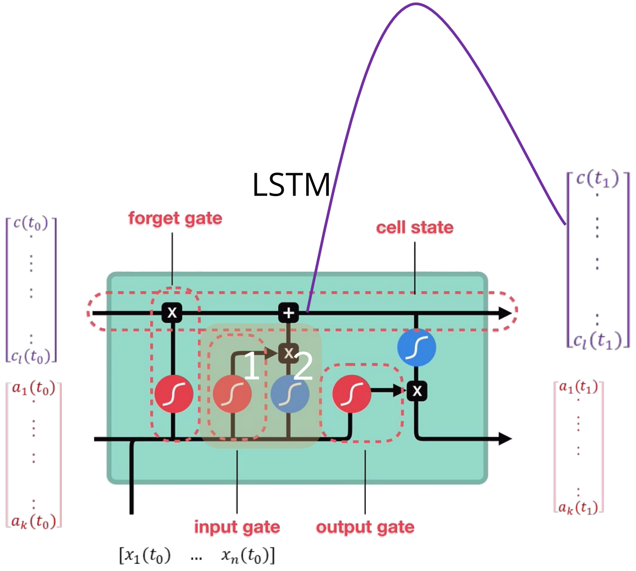


LSTM “input” gate

- In order to <u>update</u> the cell state, we have the input gate.
- Input gate does two things:
  1. *passes the previous hidden state and current input into a sigmoid function, that decides which values will be updated*
  2. *passes the hidden state and current input into a $tanh$ function to squish values between -1 and 1 to help regulate the network.*
- Then you multiply the $tanh$ output with the $sigmoid$ output. The sigmoid output will decide which information is important to keep from the $tanh$ output.

I will be using the figures from this guy’s very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi



METU

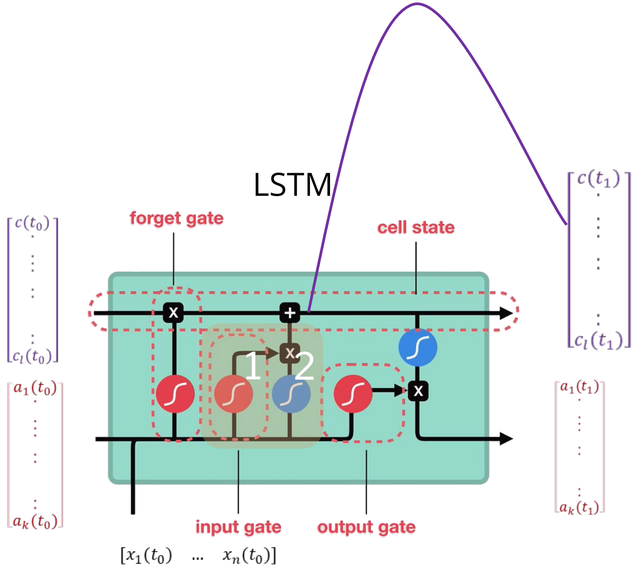
LSTM “input” gate

- How does the update take place?

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- $\hat{\mathbf{i}}_{l\times 1} = \sigma\left(\mathbf{W}^{inp1}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times 1}(t_0)\\ \mathbf{a}_{k\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{inp1}_{l\times 1}\right)$
- $\tilde{\mathbf{c}}_{l\times 1} = tanh\left(\mathbf{W}^{inp2}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times 1}(t_0)\\ \mathbf{a}_{k\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{inp2}_{l\times 1}\right)$



LSTM “forget+input” gate

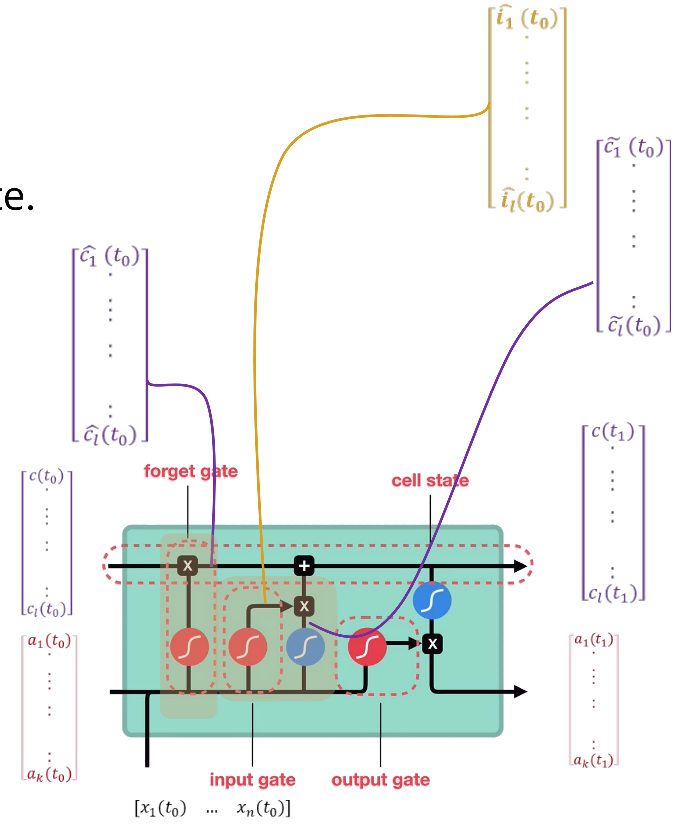
- Two gates combined, provide us the final cell state.

forget

- $\hat{\mathbf{c}}_{l\mathrm{x}1} = \mathbf{c}_{l\mathrm{x}1} \odot \sigma\left(\mathbf{W}^{forget}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times1}(t_0)\\ \mathbf{a}_{k\times1}(t_0)\end{bmatrix} + \mathbf{b}^{forget}_{l\times1}\right)$

input

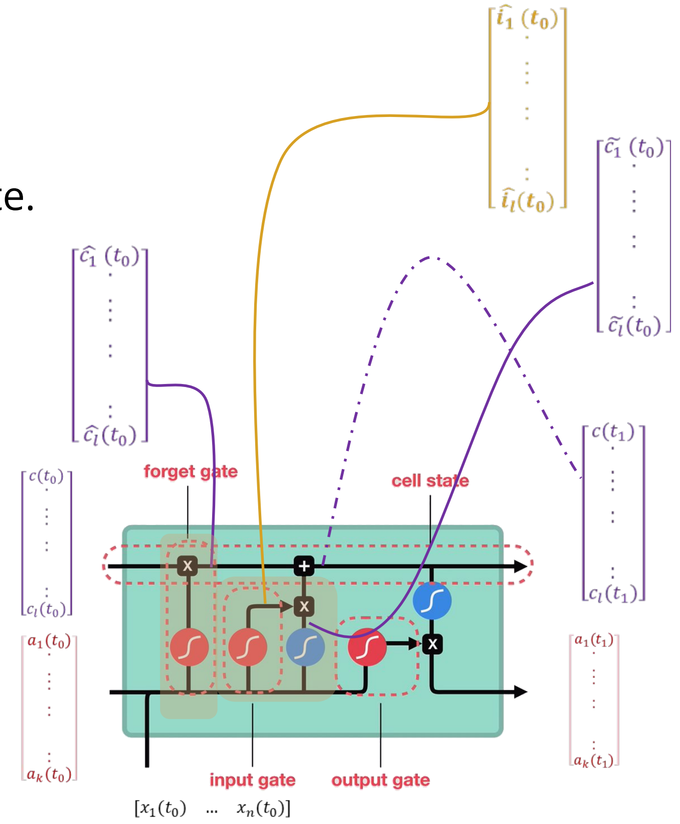
- $\hat{\mathbf{i}}_{l\mathrm{x}1} = \sigma\left(\mathbf{W}^{inp1}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times1}(t_0)\\ \mathbf{a}_{k\times1}(t_0)\end{bmatrix} + \mathbf{b}^{inp1}_{l\times1}\right)$
- $\tilde{\mathbf{c}}_{l\mathrm{x}1} = tanh\left(\mathbf{W}^{inp2}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times1}(t_0)\\ \mathbf{a}_{k\times1}(t_0)\end{bmatrix} + \mathbf{b}^{inp2}_{l\times1}\right)$


$[\hat{i}_1(t_0) \; \vdots \; \hat{i}_l(t_0)]$
$[\tilde{c}_1(t_0) \; \vdots \; \tilde{c}_l(t_0)]$
$[\hat{c}_1(t_0) \; \vdots \; \hat{c}_l(t_0)]$
$[c(t_0) \; \vdots \; c_l(t_0)]$
$[a_1(t_0) \; \vdots \; a_k(t_0)]$
forget gate
cell state
$[c(t_1) \; \vdots \; c_l(t_1)]$
$[a_1(t_1) \; \vdots \; a_k(t_1)]$
input gate
output gate
$[x_1(t_0) \; \ldots \; x_n(t_0)]$

LSTM “forget+input” gate

- Two gates combined, provide us the final cell state.
  - $$\hat{\mathbf{c}}_{l\times1} = \mathbf{c}_{l\times1} \odot \sigma\left(\mathbf{W}^{forget}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times1}(t_0)\\ \mathbf{a}_{k\times1}(t_0)\end{bmatrix} + \mathbf{b}^{forget}_{l\times1}\right)$$
  - $$\hat{\mathbf{i}}_{l\times1} = \sigma\left(\mathbf{W}^{inp1}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times1}(t_0)\\ \mathbf{a}_{k\times1}(t_0)\end{bmatrix} + \mathbf{b}^{inp1}_{l\times1}\right)$$
  - $$\tilde{\mathbf{c}}_{l\times1} = \tanh\left(\mathbf{W}^{inp2}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times1}(t_0)\\ \mathbf{a}_{k\times1}(t_0)\end{bmatrix} + \mathbf{b}^{inp2}_{l\times1}\right)$$

$$\mathbf{c}(t_{next})_{l\times1} = \hat{\mathbf{c}}_{l\times1} + \hat{\mathbf{i}}_{l\times1} \odot \tilde{\mathbf{c}}_{l\times1}$$



$\hat{i}_1(t_0)$ … $\hat{i}_l(t_0)$
$\tilde{c}_1(t_0)$ … $\tilde{c}_l(t_0)$
$\hat{c}_1(t_0)$ … $\hat{c}_l(t_0)$
$c(t_0)$ … $c_l(t_0)$
$c(t_1)$ … $c_l(t_1)$
$a_1(t_0)$ … $a_k(t_0)$
$a_1(t_1)$ … $a_k(t_1)$
forget gate
cell state
input gate
output gate
$[x_1(t_0)$ … $x_n(t_0)]$

METU

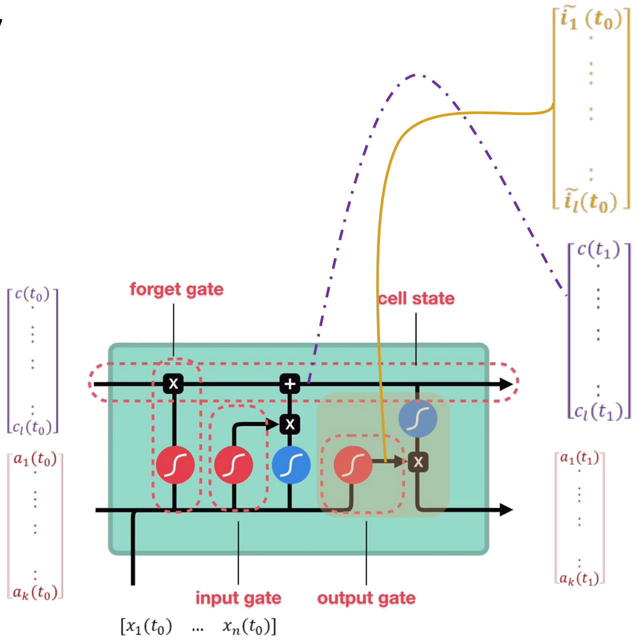
LSTM “output” gate

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- Finally, in order to obtain the hidden state **a(t$_{next}$)**, output gate uses
  - the new cell **c(t$_{next}$)**,
  - The previous hidden state **a(t$_{prevt}$)**,
  - and the current input **x**

  - $$\tilde{\mathrm{i}}_{\mathrm{l}x1} = \sigma\left(\mathbf{W}^{out1}_{\mathrm{l}\times(\mathrm{k}+\mathrm{n})} \cdot \begin{bmatrix} \mathbf{x}_{\mathrm{n}\times1}(t_0) \\ \mathbf{a}_{\mathrm{k}\times1}(t_0) \end{bmatrix} + \mathbf{b}^{out1}_{\mathrm{l}\times1}\right)$$
  - $$\mathbf{a}(t_{next})_{\mathrm{k}x1} = \tilde{\mathrm{i}}_{\mathrm{l}x1} \odot tanh\left(\mathbf{W}^{out2}_{\mathrm{k}\times\mathrm{l}} \cdot \mathbf{c}(t_{next})_{\mathrm{l}x1} + \mathbf{b}^{out2}_{\mathrm{k}\times1}\right)$$



$[\tilde{i}_1(t_0) \; \vdots \; \tilde{i}_l(t_0)]$
$[c(t_0) \; \vdots \; c_l(t_0)]$
forget gate
cell state
$[c(t_1) \; \vdots \; c_l(t_1)]$
$[a_1(t_0) \; \vdots \; a_k(t_0)]$
$[a_1(t_1) \; \vdots \; a_k(t_1)]$
input gate
output gate
$[x_1(t_0) \; \dots \; x_n(t_0)]$

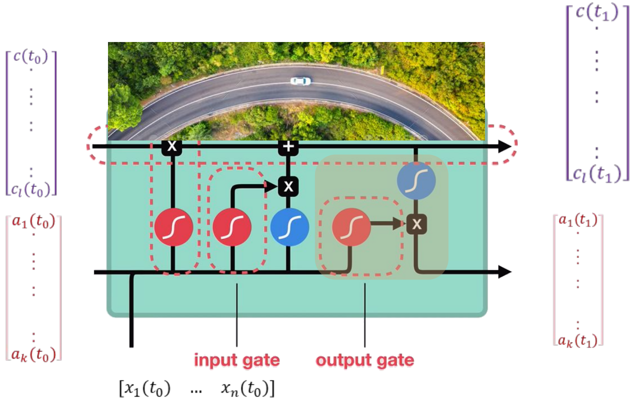
LSTM: observations

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- During backprop, for the cell state, there is an open highway. That's why we say, cell state can learn long term dependencies (i.e. Long term memory)

- During backprop, the hidden state is no different than a hidden state of a vanilla RNN. That's why it is prone to vanilla RNN problems, such as vanishing gradients etc.
  - It is for short term memory.
- **LSTM is long (the cell state), short (the hidden state), term memory.**



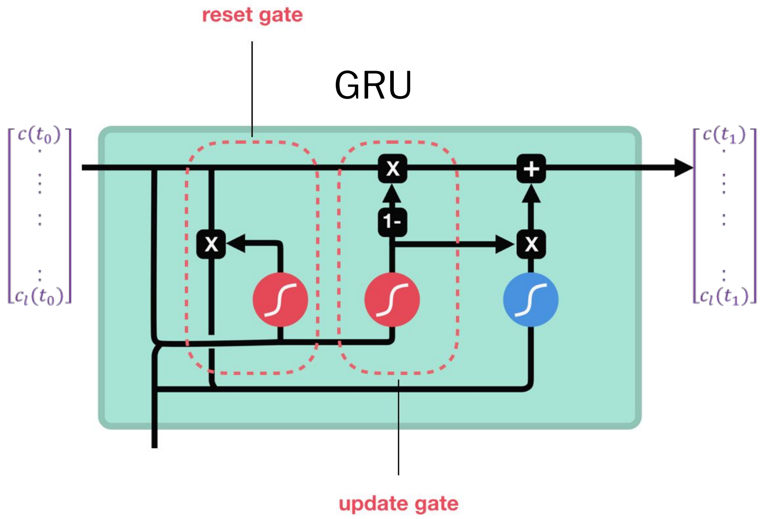
GRU: Gated Recurrent Unit

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- GRU is «LSTM Remastered – 2014». Like Iron Maiden's remastered studio albums. It is technically better, But it never gives you the same feeling. Or maybe it is just me.

- It is a simpler version, designed to be more efficient.
- GRU has only two gates, a reset gate and update gate. No output gate!
- No output gate because, GRU has only a single state. In a way the cell state and hidden state are one.



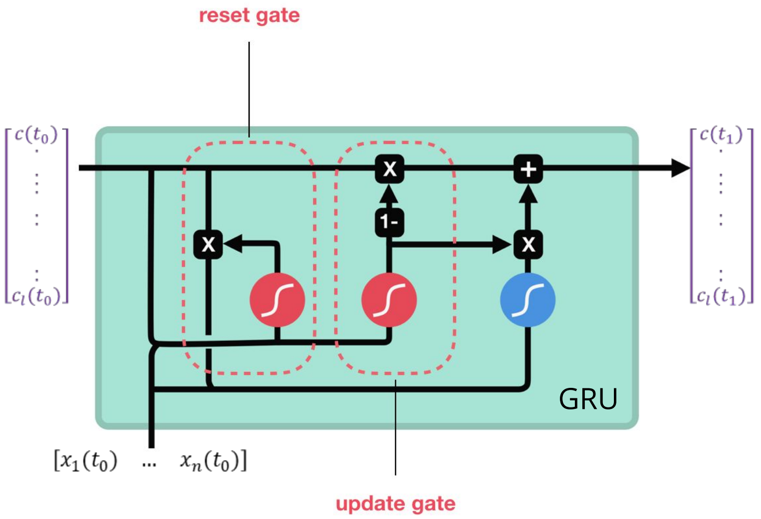
METU

GRU vs LSTM Gates

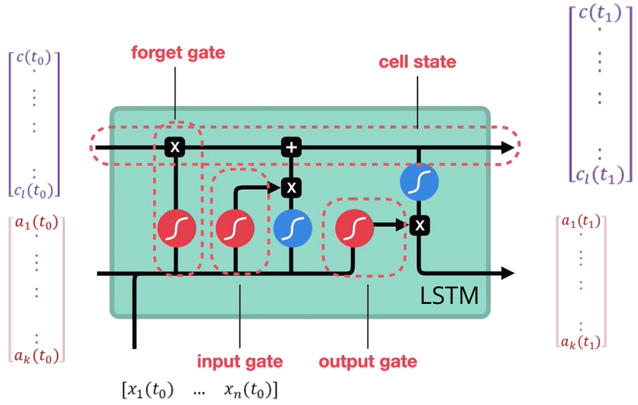
I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- There are only two gates: reset and output.
  - GRU Reset Gate ≈ LSTM forget gate
  - GRU update Gate ≈ LSTM update gate



What about the output gate?



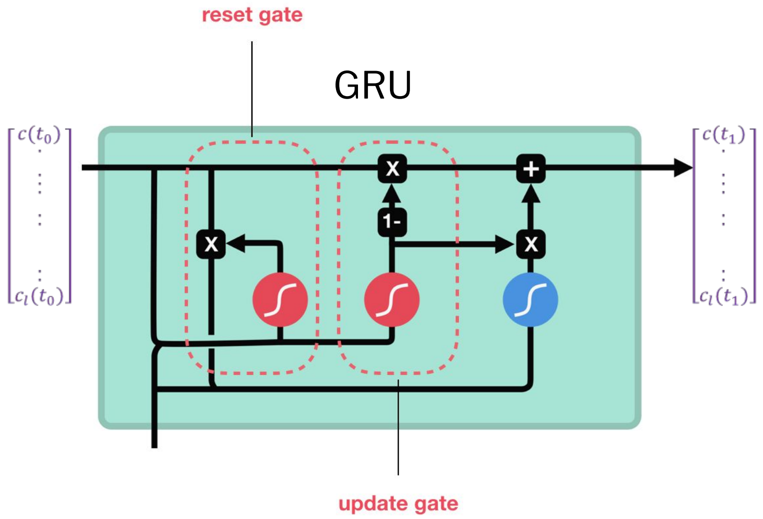
METU

GRU: no output gate

Some say GRU has a hidden state but no cell state. I'll explain now.

You may have already realized that there is no hidden state **a(t_{next})** in GRU.

- And remember that LSTM output gate's purpose was to calculate the hidden state **a(t_{next})**.
  - **Hence, no output gate!**
- But why no hidden state then?

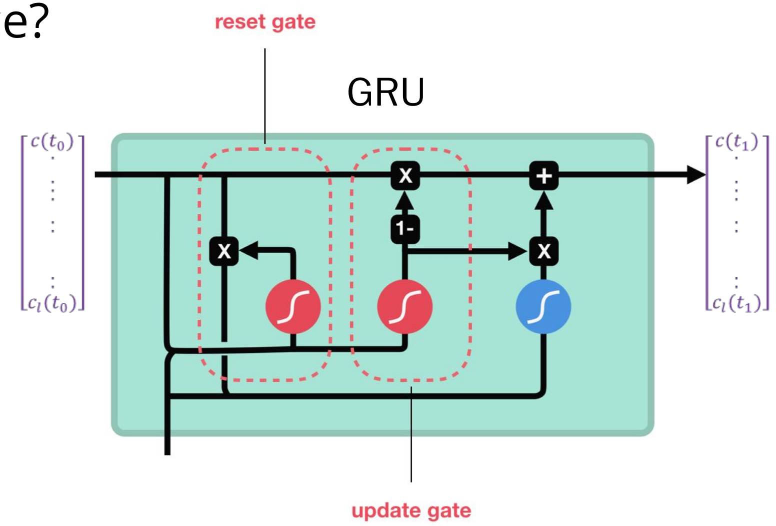


GRU: no output gate

I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- GRU's single cell/hidden state is vectoral.
- The elements of this single vector include both the long and short term components of the sequence.
- So how does it compare to LSTM in practice?

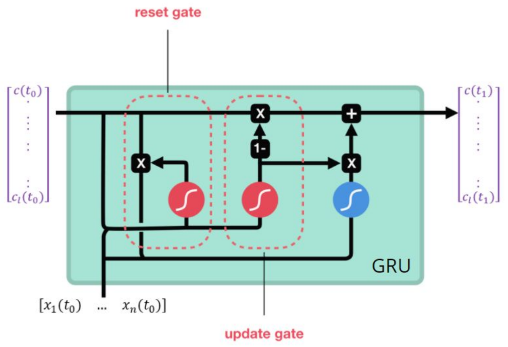


LSTM vs GRU

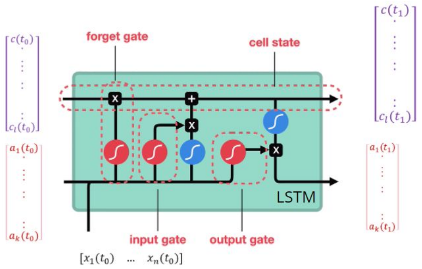
I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

- LSTM is more «transparent». We can observer what is long, what is short term memory.
- GRU is more efficient. Only 3 activations (wrt to 5 activations in LSTM). Hence, GRUs have been shown to exhibit better performance on certain smaller and less frequent datasets
- In practice, they perform similar (although LSTM is believed to perform slightly better.)
- Their comparison is a detailed subject.

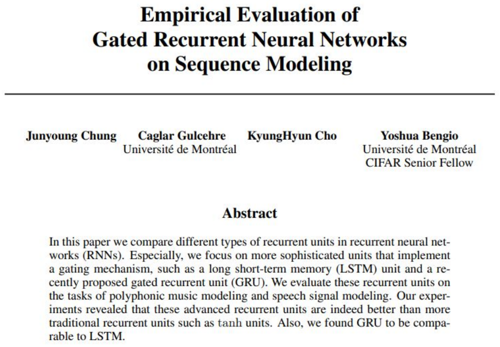


vs.



METU

LSTM vs GRU



Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Junyoung Chung Caglar Gulcehre KyungHyun Cho
Université de Montréal

Yoshua Bengio
Université de Montréal
CIFAR Senior Fellow

Abstract

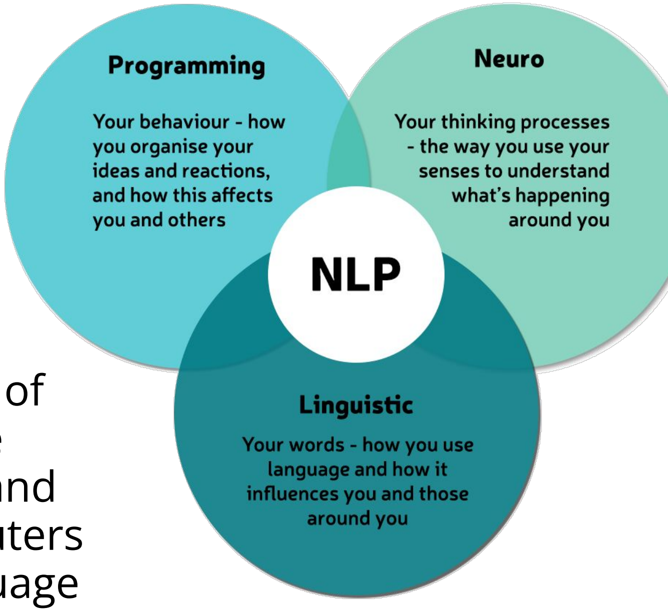
In this paper we compare different types of recurrent units in recurrent neural networks (RNNs). Especially, we focus on more sophisticated units that implement a gating mechanism, such as a long short-term memory (LSTM) unit and a recently proposed gated recurrent unit (GRU). We evaluate these recurrent units on the tasks of polyphonic music modeling and speech signal modeling. Our experiments revealed that these advanced recurrent units are indeed better than more traditional recurrent units such as tanh units. Also, we found GRU to be comparable to LSTM.

- There is a nice paper discussing which gated recurrent unit to use and why (by Bengio's team).
- Here is the link. And there is a blog discussing that discussion.
- The empirical results do not seem to be enough to declare a winner between LSTMs and GRUs; this suggests that one or the other might be best suited to a given task based on the description of the task.

Natural Language Processing (Wiki)

- NLP is Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.
- The result is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them.
- The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.

Natural Language Processing (Wiki)


Programming
Your behaviour - how you organise your ideas and reactions, and how this affects you and others
Neuro
Your thinking processes - the way you use your senses to understand what's happening around you
NLP
Linguistic
Your words - how you use language and how it influences you and those around you

- NLP is Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.
- The result is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them.
- The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.

NLP History

- Three main eras
    - Symbolic NLP (50s – to – early 90s)
    - Statistical NLP (90s – to – early 10s)
    - Neural (Deep) NLP (present!)

METU

NLP History

- Three main eras
  - Symbolic NLP (50s – to – early 90s)
    - It is learning the language semantics as a computational system. also called “deterministic”, the idea is to teach the machine how to understand languages in the same way as we, humans, have learned how to read and how to write. In order to do so, we went to school and we learned how to structure language through rules, grammar, conjugation, and vocabulary. Computational linguists do exactly the same: they use rules, lexicon and semantic in order to teach the bot’s engine how to understand a language.
  - Statistical NLP (90s – to – early 10s)
  - Neural NLP (present!)

METU

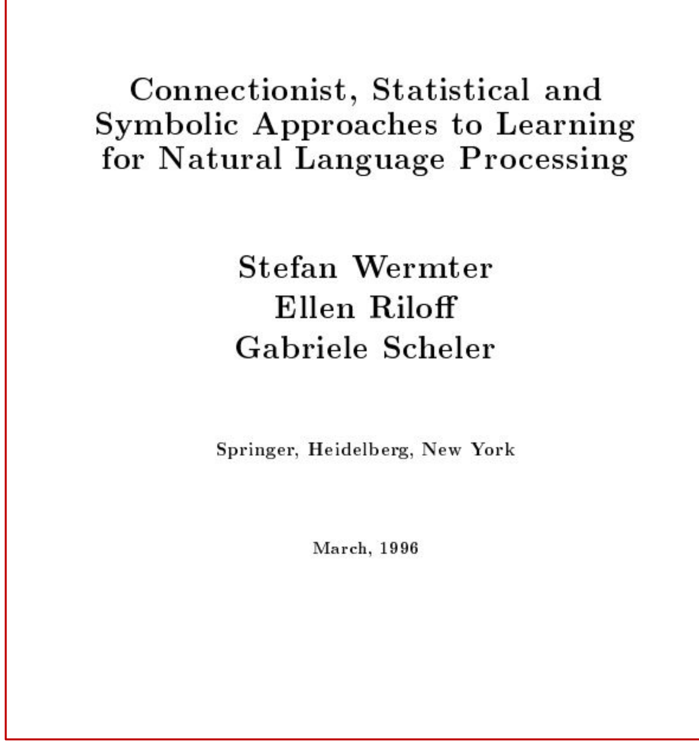
NLP History

- Three main eras
  - Symbolic NLP (50s – to – early 90s)
  - Statistical NLP (90s – to – early 10s)
    - Statistical NLP has been the most widely used term to refer to non-symbolic and non-logical work on NLP over the past decade. Statistical NLP comprises all quantitative approaches to automated language processing, including probabilistic modeling and information theory.
  - Neural NLP (present!)

METU

NLP History

- Three main eras
  - **Symbolic NLP (50s – to – early 90s)**
  - Statistical NLP (90s – to – early 10s)
  - Neural NLP (present!)



Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing

Stefan Wermter
Ellen Riloff
Gabriele Scheler

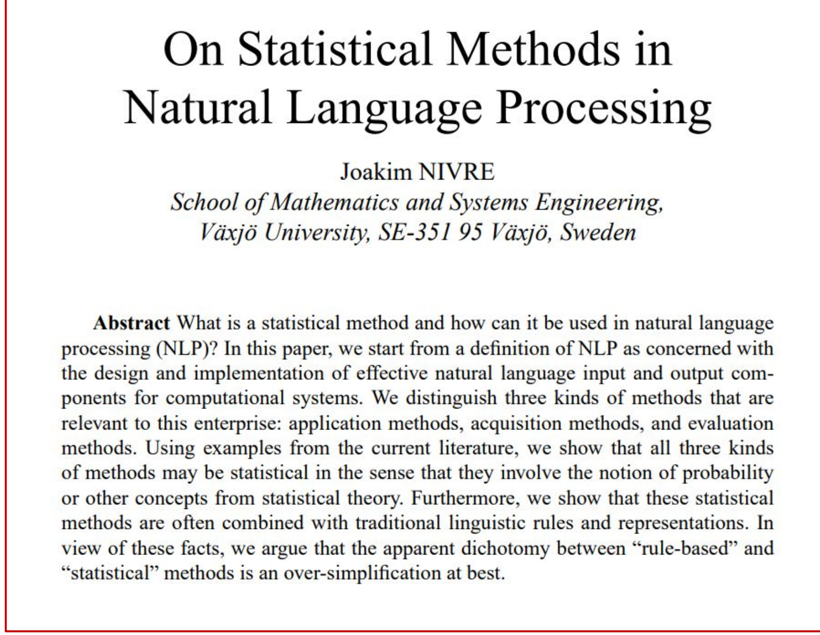
Springer, Heidelberg, New York

March, 1996

METU

NLP History

- Three main eras
  - Symbolic NLP (50s – to – early 90s)
  - **Statistical NLP (90s – to – early 10s)**
  - Neural NLP (present!)



On Statistical Methods in Natural Language Processing

Joakim NIVRE

School of Mathematics and Systems Engineering, Växjö University, SE-351 95 Växjö, Sweden

Abstract What is a statistical method and how can it be used in natural language processing (NLP)? In this paper, we start from a definition of NLP as concerned with the design and implementation of effective natural language input and output components for computational systems. We distinguish three kinds of methods that are relevant to this enterprise: application methods, acquisition methods, and evaluation methods. Using examples from the current literature, we show that all three kinds of methods may be statistical in the sense that they involve the notion of probability or other concepts from statistical theory. Furthermore, we show that these statistical methods are often combined with traditional linguistic rules and representations. In view of these facts, we argue that the apparent dichotomy between "rule-based" and "statistical" methods is an over-simplification at best.

NLP History

- Three main eras
  - Symbolic NLP (50s – to – early 90s)
  - Statistical NLP (90s – to – early 10s)
  - Neural NLP (present!)
    - In the 2010s, representation learning and deep neural network-style machine learning methods became widespread in natural language processing, due in part to a flurry of results showing that such techniques can achieve state-of-the-art results in many natural language tasks, for example in language modeling, parsing, and many others.
      - This method uses RNNS, LSTMs and further architectures (such as Transformers)

METU

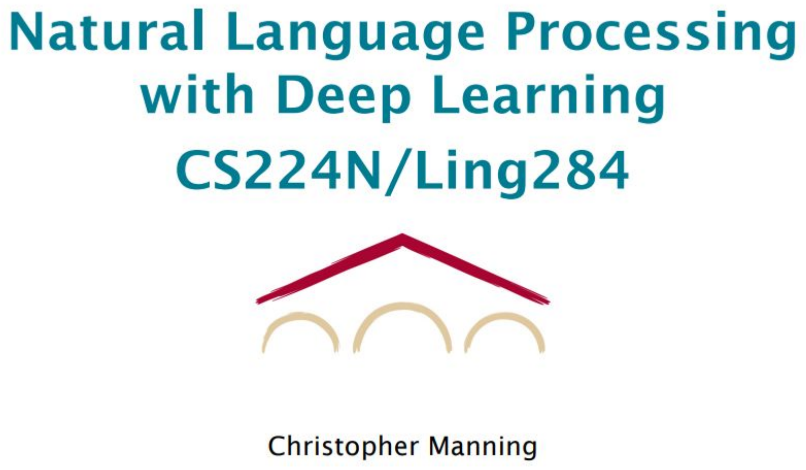
NLP History

- Three main eras
  - Symbolic NLP (50s – to – early 90s)
  - Statistical NLP (90s – to – early 10s)
  - Neural NLP (present!)
    - In the 2010s, representation learning and deep neural network-style machine learning methods became widespread in natural language processing, due in part to a flurry of results showing that such techniques can achieve state-of-the-art results in many natural language tasks, for example in language modeling, parsing, and many others.
      - This method uses RNNS, LSTMs and further architectures (such as Transformers)

METU

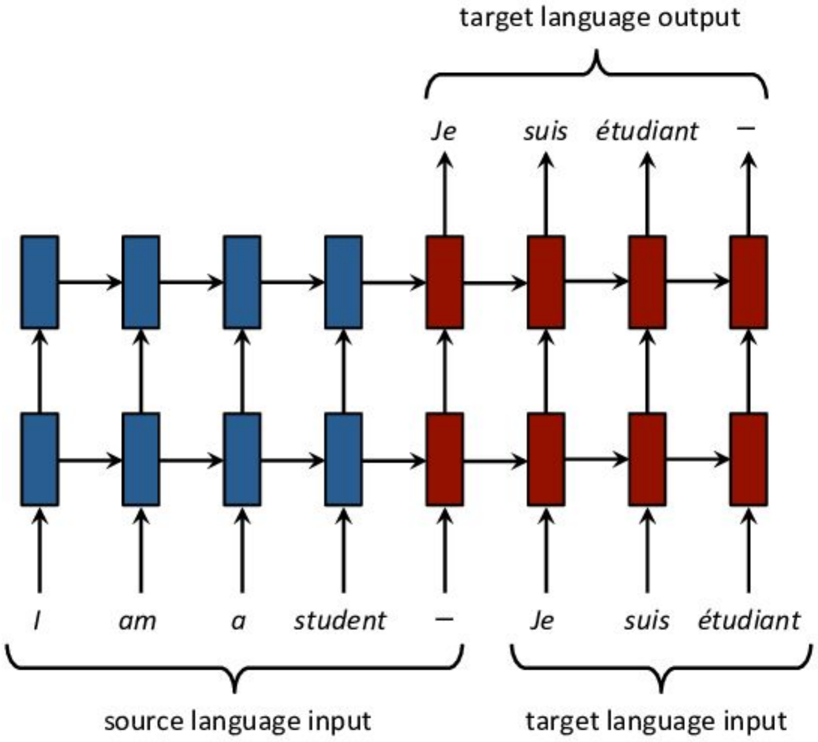
NLP History

- Three main eras
  - Symbolic NLP (50s – to – early 90s)
  - Statistical NLP (90s – to – early 10s)
  - Neural NLP (present!)
    - In the 2010s, representation learning and deep neural network-style machine learning methods became widespread in natural language processing, due in part to a flurry of results showing that such techniques can achieve state-of-the-art results in many natural language tasks, for example in language modeling, parsing, and many others.



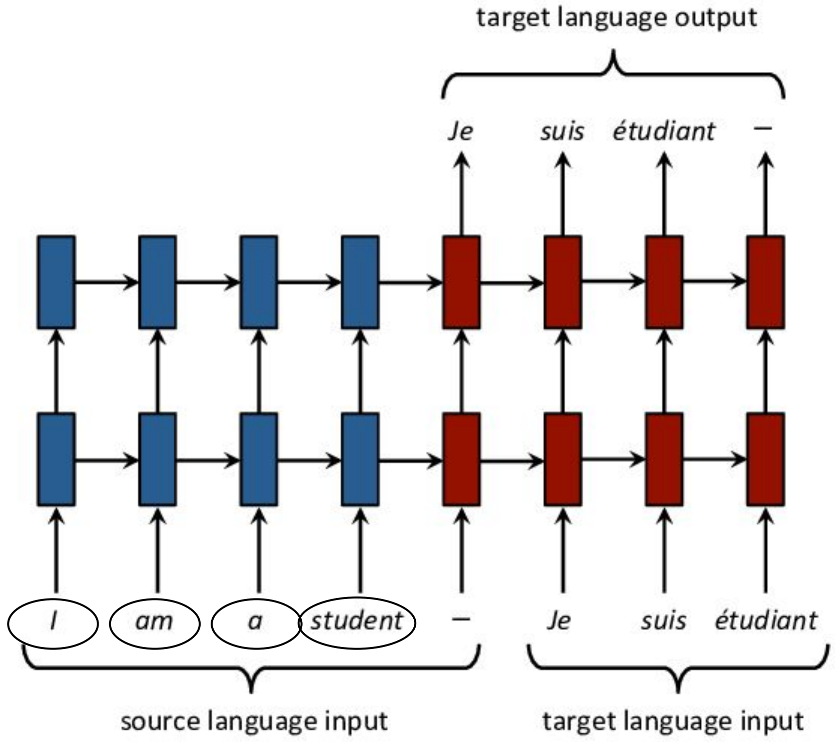
Neural NLP

- So neural NLP it is!
- We simply treat the language as a sequence model.
  - Is it that simple?



Neural NLP

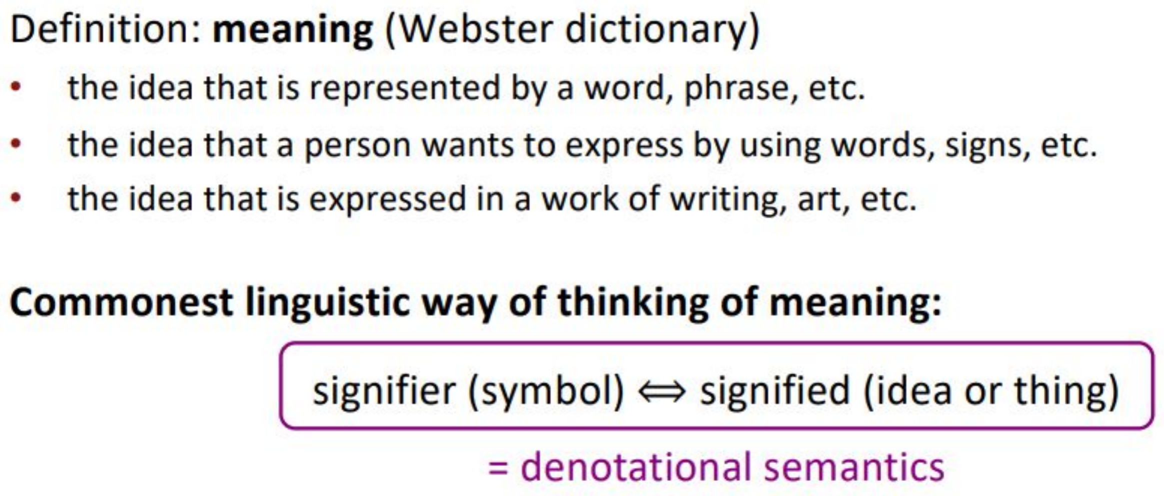
- So neural NLP it is!
- We simply treat the language as a sequence model.
  - Is it that simple?
- How do we represent the meaning of a word?



METU

Meaning of a Word!

- «How do we represent the meaning of a word?» is a fundamental question.



Definition: **meaning** (Webster dictionary)

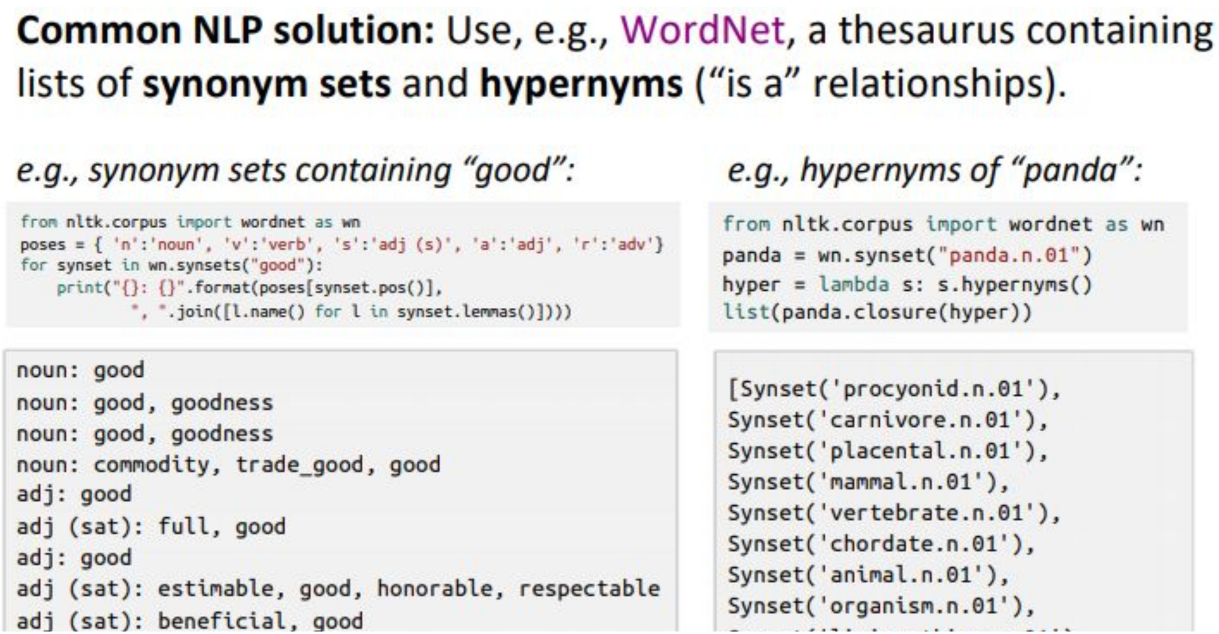
- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

$\boxed{\text{signifier (symbol)} \Leftrightarrow \text{signified (idea or thing)}}$

= denotational semantics

Meaning of a Word!



Common NLP solution: Use, e.g., WordNet, a thesaurus containing lists of **synonym sets** and **hypernyms** ("is a" relationships).

e.g., synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

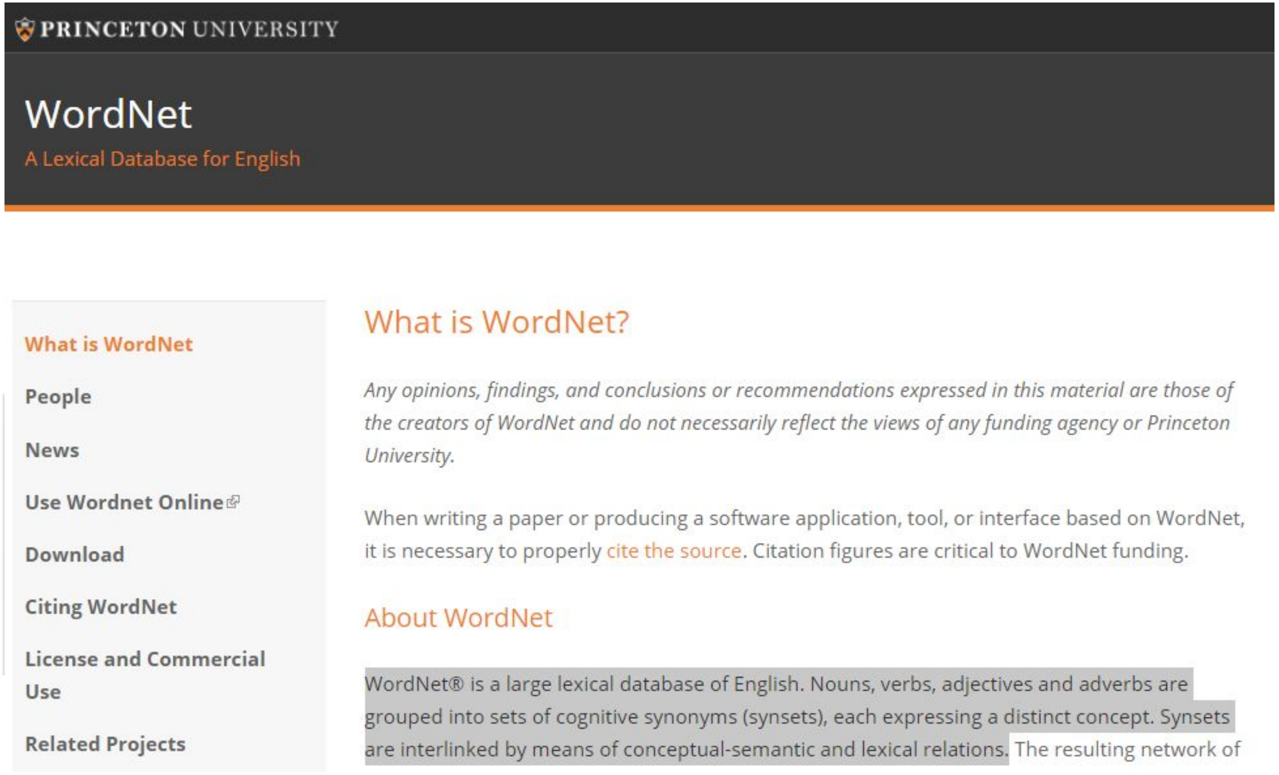
```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
```

e.g., hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
```

Meaning of a Word!



PRINCETON UNIVERSITY

WordNet

A Lexical Database for English

- What is WordNet
- People
- News
- Use Wordnet Online
- Download
- Citing WordNet
- License and Commercial Use
- Related Projects

What is WordNet?

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the creators of WordNet and do not necessarily reflect the views of any funding agency or Princeton University.

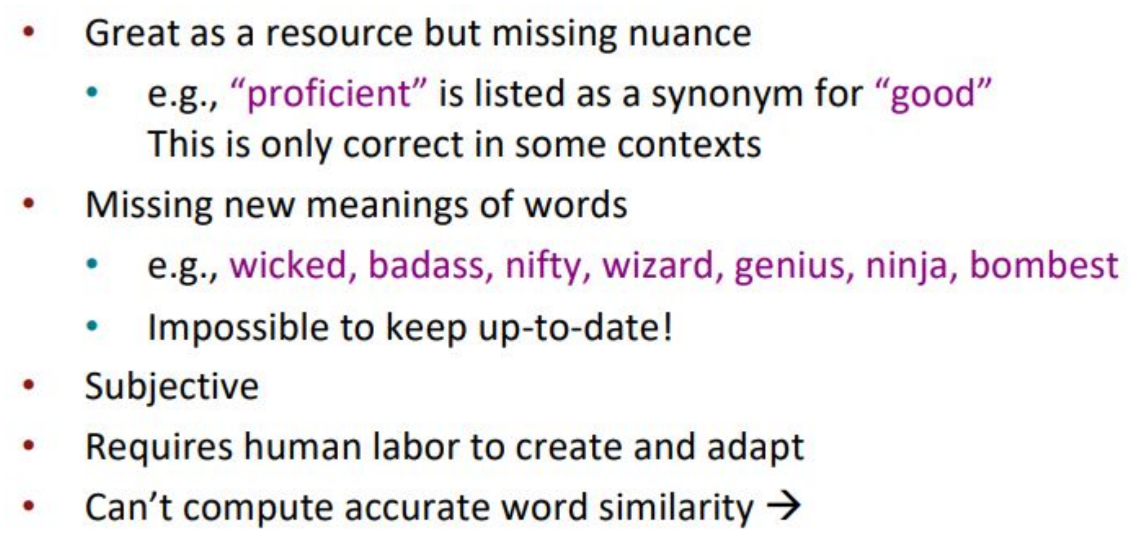
When writing a paper or producing a software application, tool, or interface based on WordNet, it is necessary to properly cite the source. Citation figures are critical to WordNet funding.

About WordNet

WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of

METU

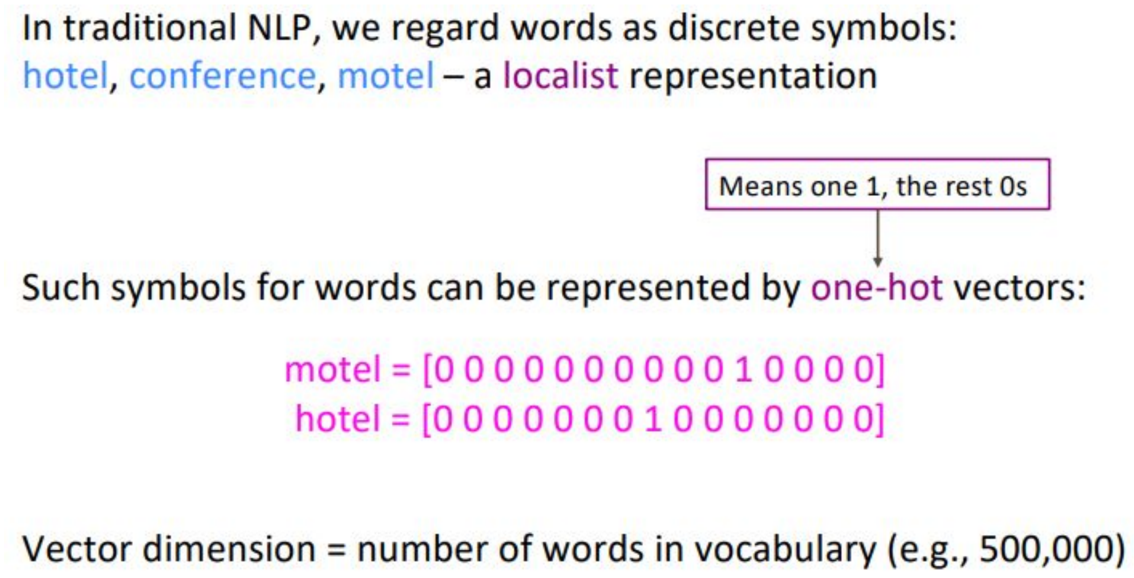
WordNet good but eh!



- Great as a resource but missing nuance
  - e.g., “proficient” is listed as a synonym for “good”
    This is only correct in some contexts
- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can’t compute accurate word similarity →

METU

Word similarity



In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

METU

Word similarity

Example: in web search, if user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal

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

Solution:

- Could try to rely on WordNet’s list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

METU

Word similarity

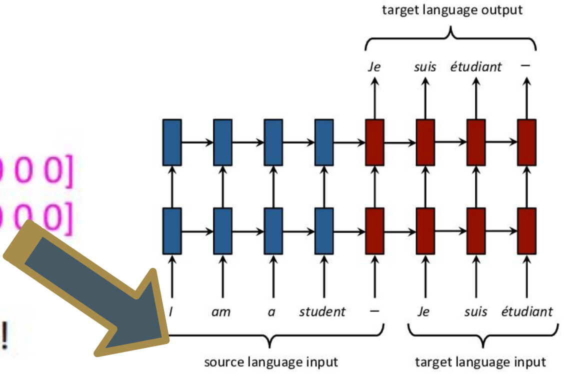
Example: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal

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



Solution:

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

Word vectors

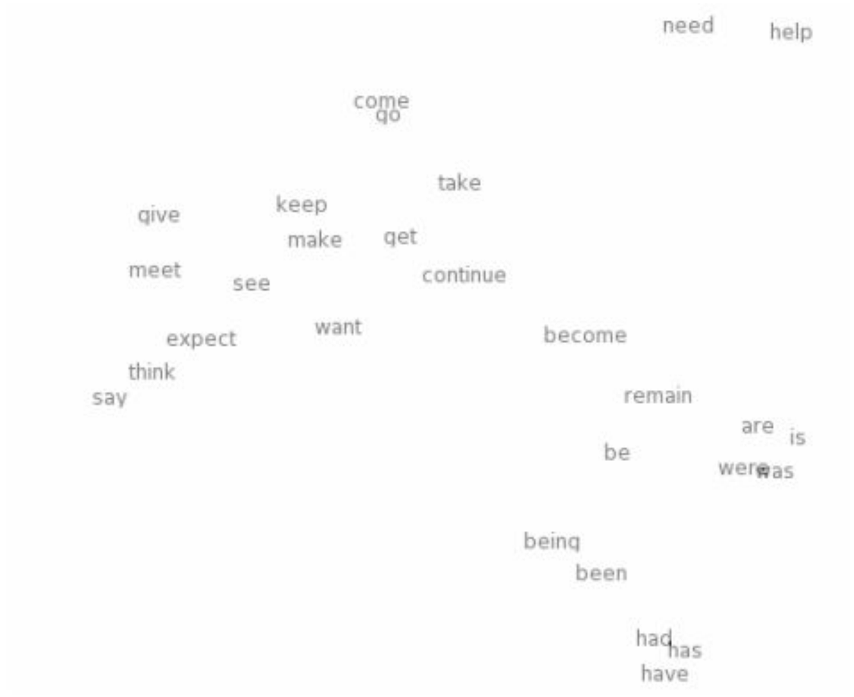
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Note: word vectors are also called word embeddings or (neural) word representations
They are a distributed representation

Word vector space
(2D projection)

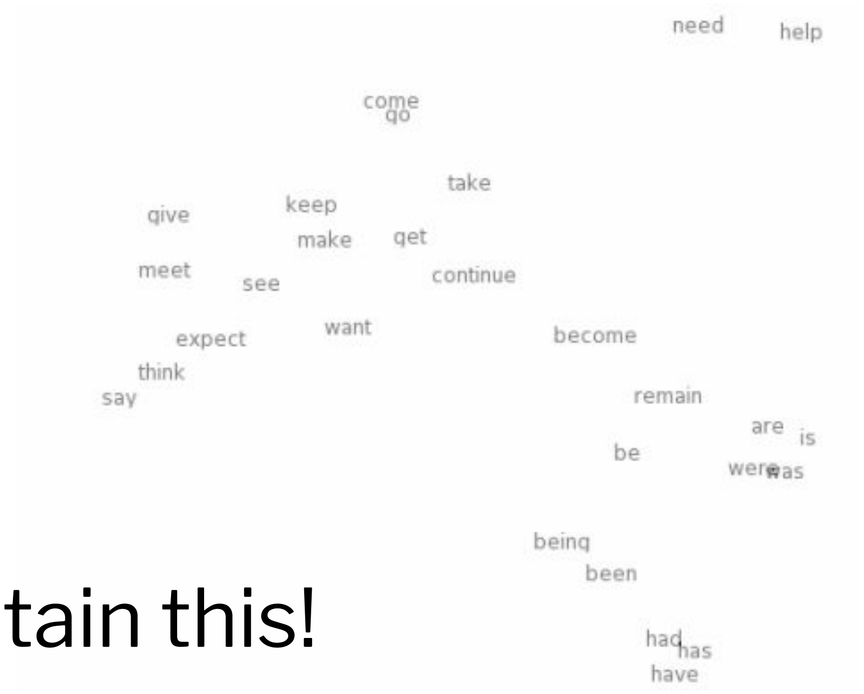
$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



METU

Word vector space
(2D projection)

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



But how to obtain this!

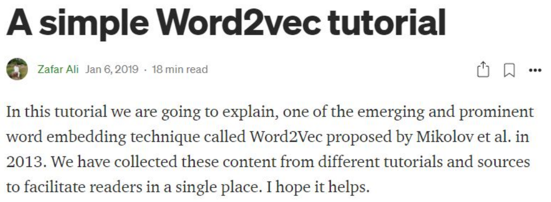
Word-2-Vec

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus ("body") of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

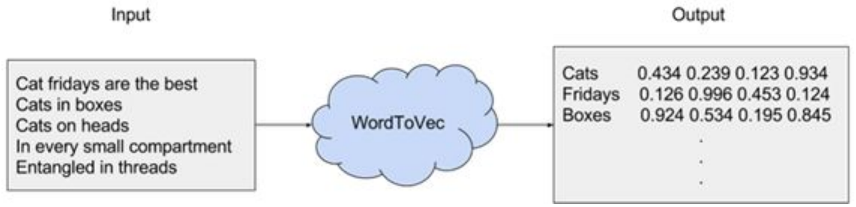
Word-2-Vec


A simple Word2vec tutorial

Zafar Ali Jan 6, 2019 · 18 min read

In this tutorial we are going to explain, one of the emerging and prominent word embedding technique called Word2Vec proposed by Mikolov et al. in 2013. We have collected these content from different tutorials and sources to facilitate readers in a single place. I hope it helps.

- Word2vec is a combination of models used to represent distributed representations of words in a corpus C.
- Word2Vec (W2V) is an algorithm that accepts text corpus as an input and outputs a vector representation for each word, as shown in the diagram below:


Input
Cat fridays are the best
Cats in boxes
Cats on heads
In every small compartment
Entangled in threads
WordToVec
Output
Cats 0.434 0.239 0.123 0.934
Fridays 0.126 0.996 0.453 0.124
Boxes 0.924 0.534 0.195 0.845
.
.
.

Word-2-Vec

- There are two common approaches of this algorithm namely:
    - **Skip-Gram**. Given a set of sentences (also called corpus) the model loops on the words of each sentence and either tries to use the current word *w* in order to predict its neighbors.
    - **CBoWs.** or it uses each of these neighbors to predict the current word *w*, in that case the method is called “Continuous Bag Of Words” (CBOW). In order to limit the number of words in each neighborhood, a parameter called “window size” is used.

Word-2-Vec

- There are two common approaches of this algorithm namely:
  - **Skip-Gram**. Given a set of sentences (also called corpus) the model loops on the words of each sentence and either tries to use the current word *w* in order to predict its neighbors.
  - **CBoWs.** or it uses each of these neighbors to predict the current word *w*, in that case the method is called “Continuous Bag Of Words” (CBOW). In order to limit the number of words in each neighborhood, a parameter called “window size” is used.

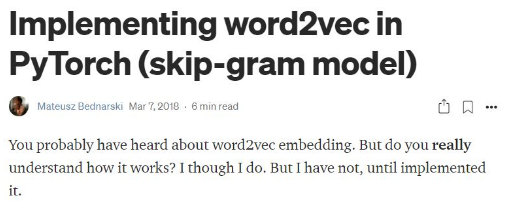
Details of W2V algorithms are beyond the scope of DI504!

Word-2-Vec (let's discuss)

- The output of Word2vec, i.e. -the vectors we use to represent words are called neural word embeddings.
- The representations are strange. One thing describes another, even though those two things are radically different.
- Word2vec "vectorizes" about words, and by doing so it makes natural language computer-readable — we can start to perform powerful mathematical operations on words to detect their similarities.
- So, a neural word embedding represents a word with numbers.
- Word2vec is similar to an auto-encoder, encoding each word in a vector. Like a feature extractor.

W2V implementations

- There are many implementations for various Python libraries.



Implementing word2vec in PyTorch (skip-gram model)

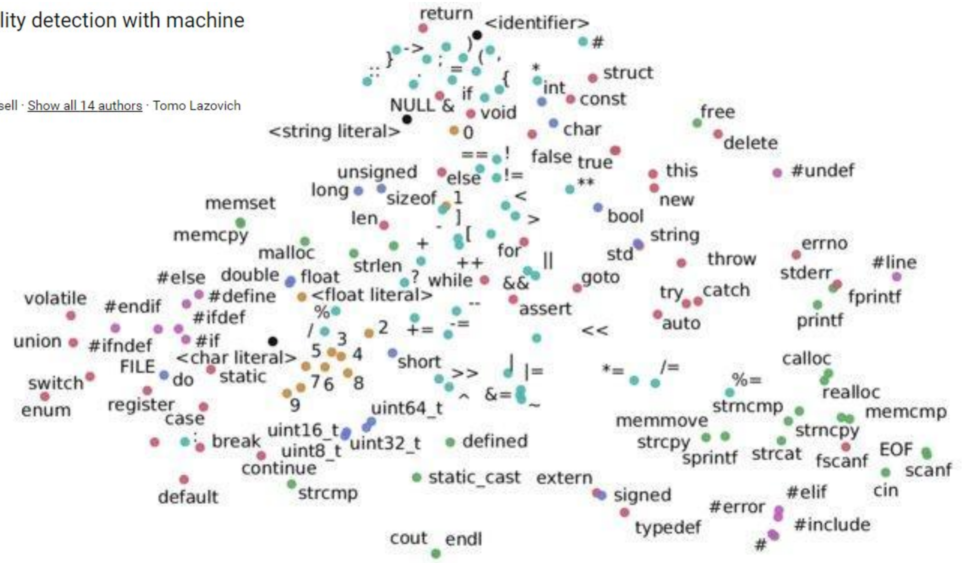
Mateusz Bednarski Mar 7, 2018 · 6 min read

You probably have heard about word2vec embedding. But do you **really** understand how it works? I though I do. But I have not, until implemented it.

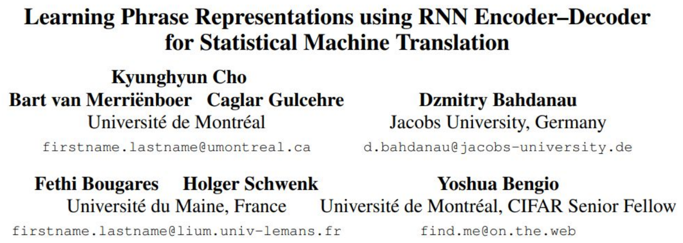
Automated software vulnerability detection with machine learning

February 2018

Jacob A. Harer · Louis Kim · Rebecca L. Russell · Show all 14 authors · Tomo Lazovich



Seq2Seq by EDs


Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Kyunghyun Cho
Bart van Merriënboer **Caglar Gulcehre**
Université de Montréal
firstname.lastname@umontreal.ca

Dzmitry Bahdanau
Jacobs University, Germany
d.bahdanau@jacobs-university.de

Fethi Bougares **Holger Schwenk**
Université du Maine, France
firstname.lastname@lium.univ-lemans.fr

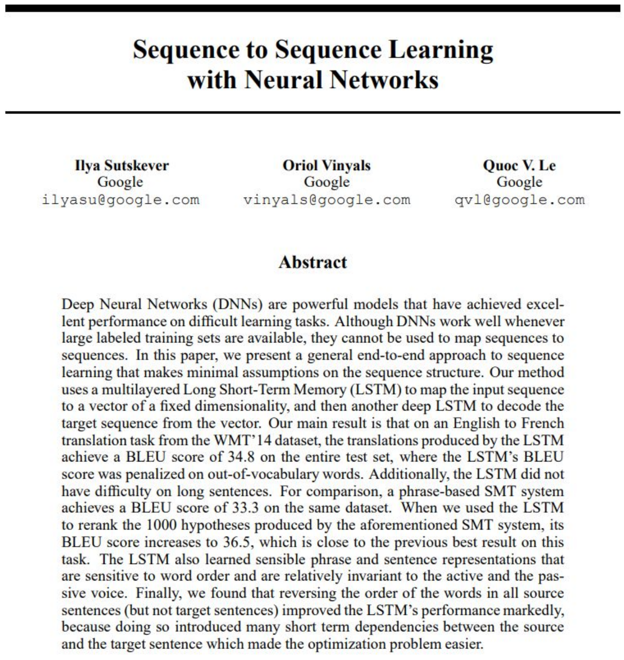
Yoshua Bengio
Université de Montréal, CIFAR Senior Fellow
find.me@on.the.web

- One of earliest RNN Encoder/Decoder was proposed by [Cho et al., 2014]
  (team led by Bengio, ~10k citations)
- They proposed the very first RNN Encoder-Decoder that consists of two recurrent neural networks (RNN).
  - One RNN encodes a sequence of symbols into a fixed-length vector representation,
  - and the other decodes the representation into another sequence of symbols.

METU

Seq2Seq by EDs

- The first «successful» application of the ED idea to language translation (i.e. NLP in general) was by Google, the very same year [Sutskever et al. 2014] (~10k citations)
- The method used a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector.
- That main result was an English to French translation.



Sequence to Sequence Learning with Neural Networks

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT'14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.

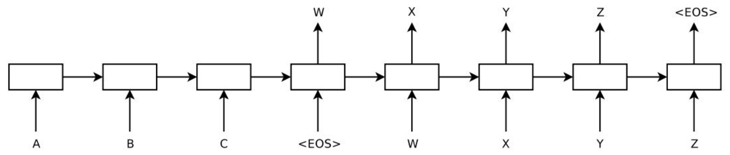
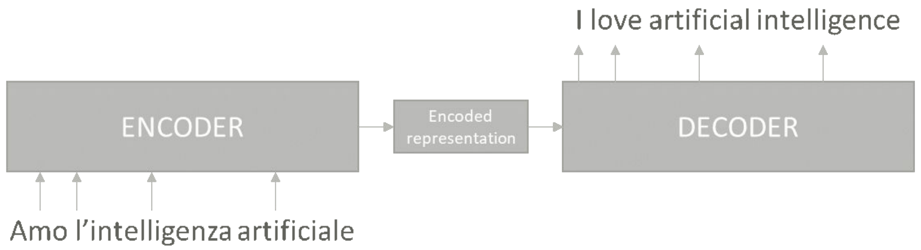


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

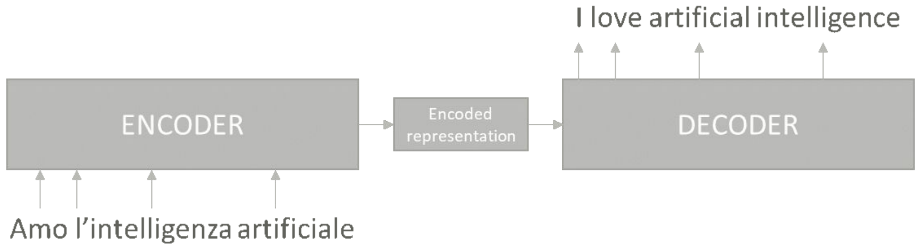
Encoder/Decoders for NLP

- This is how the great journey of Neural NLP started, with EDs.

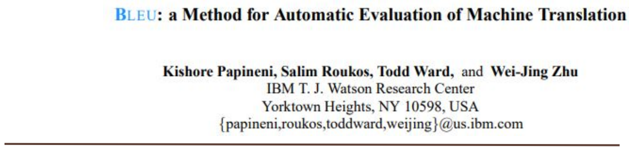

I love artificial intelligence
ENCODER
Encoded representation
DECODER
Amo l'intelligenza artificiale

Encoder/Decoders for NLP

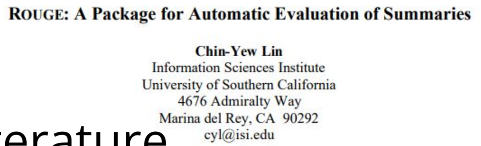
- This is how the great journey of Neural NLP started, with EDs.
- One of the important problems of this problem was the cost/loss function.
- How to calculate a loss for a wrong translation?


I love artificial intelligence
ENCODER
Encoded representation
DECODER
Amo l'intelligenza artificiale

METU


BLEU: a Method for Automatic Evaluation of Machine Translation

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
{papineni,roukos,toddward,weijing}@us.ibm.com


ROUGE: A Package for Automatic Evaluation of Summaries

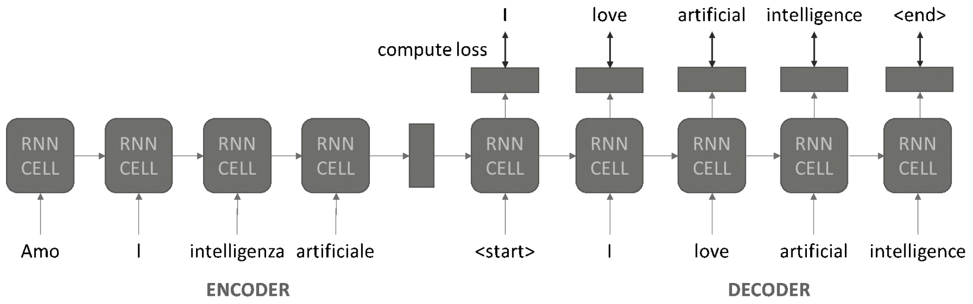
Chin-Yew Lin
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
cyl@isi.edu

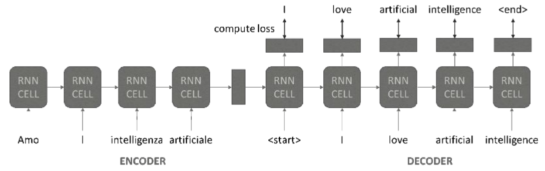
Loss functions for NLP

- There are some common loss functions use din the literature, such as BLEU [Papineni et al. 2002], ROUGE [Lin 2004], (and some other less famous)
- In general:
  - **BLEU (**Bilingual Evaluation Understudy**) measures precision**: how much the words (and/or n-grams) in the *machine generated summaries* appeared in the human reference summaries.
  - **ROUGE (**Recall-Oriented Understudy for Gisting Evaluation**) measures recall**: how much the words (and/or n-grams) in the *human reference summaries* appeared in the machine generated summaries.

Encoder/Decoders for NLP

- So the loss is calculated (somehow). Let's get deeper into the ED architecture.
- This kind of approach **allows the input and the output sequence to have a great flexibility** in length and opened up the way to deep learning inside the automatic translation field of research.

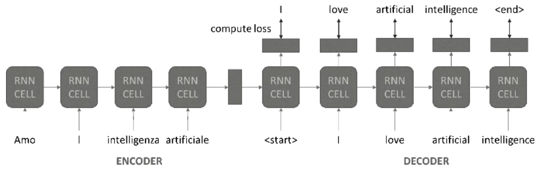




Encoder/Decoders for NLP

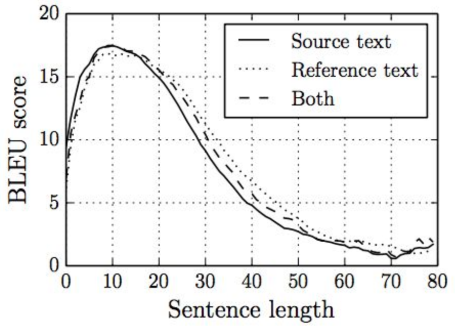
- However this approach still presents some intuitive downsides.
- As [Bahdanau et al. 2015] puts it:
  - «A potential issue with this encoder–decoder approach is that **a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector**.»
  - «This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus.
  - «Cho et al.(2014b) showed that indeed the performance of a basic encoder–decoder deteriorates rapidly as the length of an input sentence increases.»

Encoder/Decoders for NLP



- In other words:
  - Even if LSTM recurrent neural networks are able to persist information taken from the beginning of the input sentence up to the end i.e. up to the encoded array, the available space where this information can be stored is limited by the fixed dimension of the encoded array.
  - **If the input sentence is too long with respect to this fixed dimension, then we cannot avoid the loss of some important information** and the translation will be affected from it.

Figure from Cho et al., 2014



Additional Reading & References

- https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html (please continue to read this)
- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be
- https://wiki.tum.de/display/lfdv/Deep+Residual+Networks
- https://medium.com/paper-club/grus-vs-lstms-e9d8e2484848
- https://towardsdatascience.com/recurrent-neural-networks-part-4-39a568034d3b
- https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
- https://datascience.stackexchange.com/questions/61358/relu-for-combating-the-problem-of-vanishing-gradient-in-rnn
- https://arxiv.org/pdf/1503.04069.pdf
- https://arxiv.org/abs/1504.00941
- https://www.researchgate.net/publication/320511751_A_comparative_performance_analysis_of_different_activation_functions_in_LSTM_networks_for_classification

Additional Reading & References

- https://www2.informatik.uni-hamburg.de/wtm/ps/book96.pdf
- https://cl.lingfil.uu.se/~nivre/docs/statnlp.pdf
- http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture01-wordvecs1.pdf
- https://www.youtube.com/watch?v=8rXD5-xhemo
- https://medium.com/@Capeai/natural-language-processing-a-brief-history-7811f0727f44
- https://wordnet.princeton.edu/
- https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1
- https://towardsdatascience.com/implementing-word2vec-in-pytorch-skip-gram-model-e6bae040d2fb