epartamento de
ENGENHARIA INFORMÁTICA
Instituto Superior de Engenharia do Porto

isep
Instituto Superior de
Engenharia do Porto

Sequence to Sequence Tasks

Língua Natural e Sistemas Conversacionais

Luiz Faria

Adapted from Natural Language Processing course from HSE University

Introduction to Machine Translation

Machine Translation

- Machine translation is a task belonging to a category of tasks named *Sequence to Sequence tasks*: from a sequence of words in one language as an input, to a sequence of words in some other language as an output
- Summarization is another example of Sequence to Sequence tasks: we can think about it as machine translation but for one language, monolingual machine translation
- Two approaches to machine translation (some ideas are common to both):
  - Statistical machine translation
  - Neural machine translation

Parallel Data

Parallel corpora:

- Europarlament
- Movie subtitles
- Translated news, books
- Wikipedia (comparable but not parallel)
- `https://opus.nlpl.eu` – the open parallel corpus – collection of translated texts from the web

Lot's of problems with data for MT task:

- Noisy
- Specific domain (e.g. using movie subtitles to train a system for scientific papers translations)
- Rare language pairs
- Not aligned (correspondence between sentences, or between words and sentences)
- Not enough data

Evaluation

- How to compare two arbitrary translations?
- Low agreement rate even between reviewers
- BLEU score – a popular automatic technique – try to softly measure whether the system output is somehow similar to the reference translation

Evaluation

- How to compare two arbitrary translations?
- Low agreement rate even between reviewers
- BLEU score – a popular automatic translation score – value between 0 and 1 – a perfect match results in a score of 1.0

Reference: *E-mail was sent on Tuesday.*

System output: *The letter was sent on Tuesday.*

Brevity: used to penalize too short translations

1-grams: 4/6
2-grams: 3/5
3-grams: 2/4
4-grams: 1/3

$$BLEU = 1 \cdot \sqrt[4]{\frac{4}{6} \cdot \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3}}$$

Brevity: min(1, 6/5) (the length of the output divided by the length of the reference)

Papineni et. al. Bleu: A method for automatic evaluation of machine translation, 2002.

Approaches to MT

- Direct translation – translate word by word from source to target language – it does not work
- It would be better go into the syntactic level – we perform syntax analysis, and then we do the transfer and then we generate the target sentence by knowing how it should look like on the syntactic level
- Even better – It would be better go into semantic level – we analyze the source sentence and understand the meanings of parts of the sentence; then we transfer these meanings and generate syntactic structures with appropriate meaning – neural translation systems have mechanisms that somehow resembles this idea

First Approaches

1954 Georgetown IBM experiment Russian to English:

- Claimed that MT would be solved within 3-5 years.

1966 ALPAC report:

- Concluded that MT was too expensive and ineffective.

Two Main Approaches

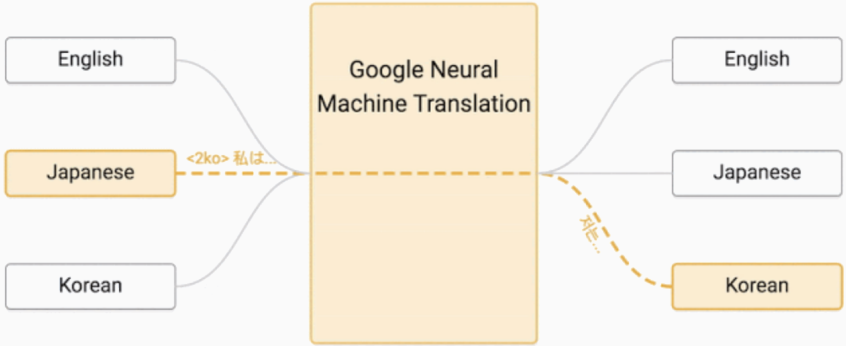
Statistical Machine Translation (SMT):

- 1988 – Word-based models (IBM models)
- 2003 – Phrase-based models (Philip Koehn)
- 2006 – Google Translate
- 2007 – Moses (a statistical machine translation system allowing to automatically train translation models for any language pair – the training uses parallel corpus)

Neural Machine Translation (NMT):

- 2013 – First papers on pure NMT
- 2015 – NMT enters shared tasks (WMT – workshop on machine translation)
- 2016 – Launched in production in companies

Zero-shot Translation



An encoder is used to encode a sentence in a source language to a hidden representation. Then the decoder takes that hidden representation and decodes it to the target sentence.

https://research.googleblog.com/2016/11/zero-shot-translation-with-googles.html

Noisy Channel Model

The Main Equation

- Given: French (foreign) sentence f,
- Find: English translation e:

$$e^* = \text{argmax}_{e \in E}\, p(e|f) = \text{argmax}_{e \in E}\, \frac{p(f|e)p(e)}{p(f)} = \text{argmax}_{e \in E}\, p(e)p(f|e)$$

$p(e|f) = \frac{p(f|e)p(e)}{p(f)} \rightarrow$ Bayes Rule

The denominator doesn't depend on the English sentence, so it will not influence the maximization of $p(e|f)$

1993 Brown et al., "The mathematics of statistical machine translation"

Why Is It Easier to Deal With?

$$e^* = \underset{e \in E}{\text{argmax}} \underbrace{p(e)p(f|e)}$$

Language model — Translation model

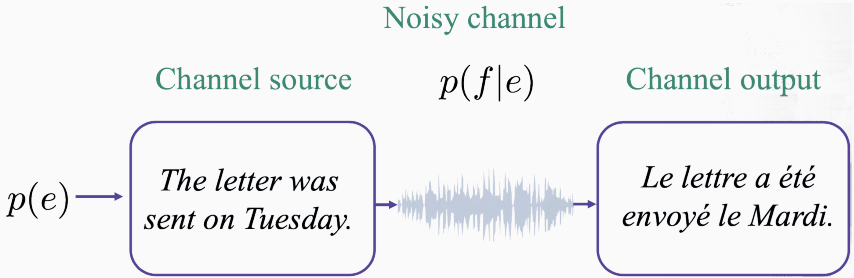
- $p(e)$ models the *fluency* of the translation
- $p(f|e)$ models the *adequacy* of the translation
- argmax is the search problem implemented by a *decoder*

This new formula allowed us to decouple a hard problem to two more simple problems:

- Language modeling – how to produce a meaningful sequence of words
- Translation model – isn't related with the coherence of the sentences – it deals with a good translation of e to f

The we use argmax to perform the search in the space and find the sentence in English that gives the best probability

Noisy Channel Model

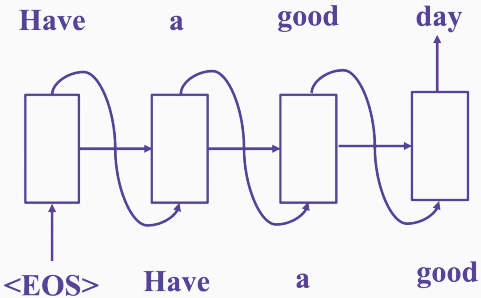


The Noisy Channel model is an interpretation of the previous formula.
We observe a distorted sentence *f* (foreign sentence). We have a model on how the sentence is distorted (translation model $p(f|e)$) and also a model on which original messages are probable (language model $p(e)$). Our goal is to recover the original sentence *e* (English sentence *e*).
"Source" in the noise channel model should be read as "original"

Language Model: $p(e)$

$$p(e) = p(e_1)p(e2|e1)\ldots p(e_k|e_1\ldots e_{k-1})$$

N-gram models or neural networks:



As we saw earlier, to compute the probability of a sentence of words, we apply the chain rule and then we know that we can factorize it into the probabilities of the next word given some previous history. We can use Markov assumption and then end up with n-gram language models. Alternatively, we can use a neural language model such as LSTM to produce the next word, given the previous words.

Translation Model: $p(f|e)$

$$p(f|e) = p(f_1, f_2, \ldots f_J|e_1, e_2, \ldots e_I)$$

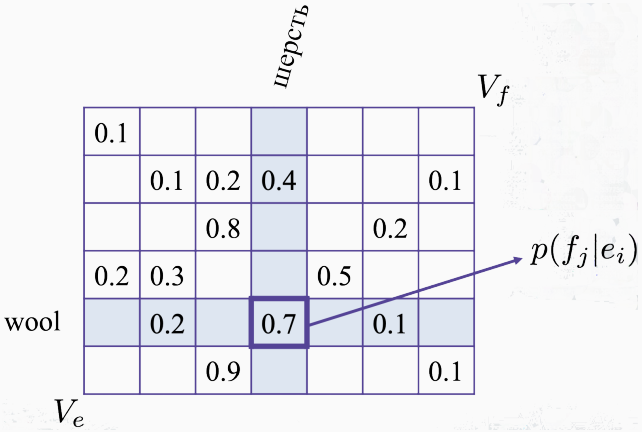
f (Foreign): Крику много, а шерсти мало.

e (English): Great cry and little wool.

The definition of the translation model is a hard task. This model produce the probability of a sequence or words in one language given a corresponding sequence of words in another language. To deal with this problem, we will get information about separate words in these sentences.

Translation Model: $p(f|e)$

We could learn translation probabilities for separate words:



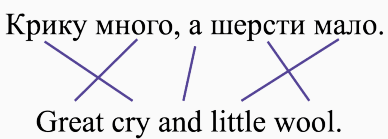
A translation table defines the probabilities of words in one language given some words in another language. Each row is normalized into one. This translations can be learnt or got from the dictionary.

Translation Model: $p(f|e)$

But how to build the probability for the whole sentences?

$$p(f|e) = \begin{matrix}\text{Some Magic} \\ \text{Factorization}\end{matrix} \left[p(f_i|e_i) \right]$$

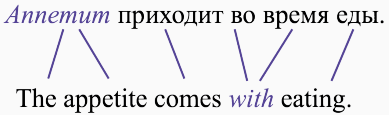
Reorderings:



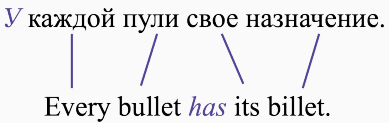
We need some word alignments. So, the problem is that we can have some reorderings in the language like the example, or even worse, we can have some one to many or many to one correspondence.

Word Alignments

One-to-many and many-to-one:



Words can disappear or appear from nowhere:



Word Alignment Models

Word Alignments

- Important subtask in machine translation, because different languages have different word order
- A word alignment model need to be learnt from the data

Word Alignment Task

Given a corpus of (e, f) sentence pairs:

- English, source: $e = (e_1, e_2, \ldots e_I)$
- Foreign, target: $f = (f_1, f_2, \ldots f_J)$

Predict:

- Alignments a between e and f:

e: The appetite comes with eating.

f: Аппетит приходит во время еды.

a?

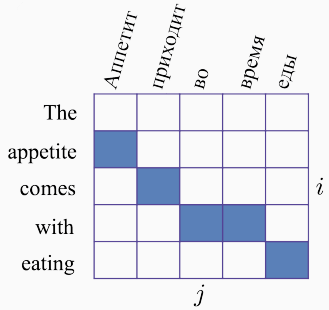
e is the sentence $e_1, e_2, \ldots, e_I$, and f is another sentence.
I and J are the lengths of sentences e and f.
a are the alignments between e and f that we must learn.
e is the source and f is target. Why if we are translating f to e? This is because we applied the Bayes rule to decouple the model in language and translation models. According to this rule, to build the system that translates from f to e, we need to model the probability of f given e.

Bayes Rule

$$e^* = \underset{e \in E}{\text{argmax}}\, \underbrace{p(e)}_{\text{Language model}} \underbrace{p(f|e)}_{\text{Translation model}}$$

- $p(e)$ models the *fluency* of the translation
- $p(f|e)$ models the *adequacy* of the translation
- argmax is the search problem implemented by a *decoder*

Word Alignment Matrix

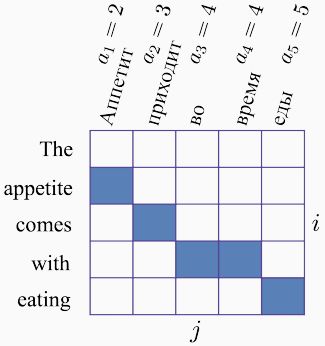


How do we represent word alignments?
The matrix is filled with zeros and ones. Cells with ones define the correspondence between words in source and target sentences.

If we consider different possible word orders, the number of matrices will be huge. To deal with this we will consider some simplifications: every target word is allowed to be aligned only to one source word – the example complies with this constraint.

Each target word *j* is allowed to have only one source *i*

Word Alignment Matrix



Notation

a_1 represent the number of the source word which is aligned to the first target word.

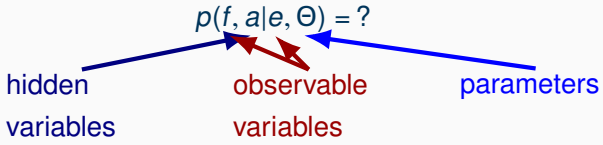
a_j represents the number of the source word which is aligned to the word in position j in the target sentence f.

Each target word j is allowed to have only one source i

Sketch of Learning Algorithm

1. Probabilistic model (generative story)

   Given e, model the generation of f:



The most creative step:

- How do we parametrize the model?
- Is it too complicated or too unrealistic?

Building the probabilistic model
e and *f* are our sentences, corresponding to the observable variables.
The goal is to model word alignments *a*. Since these alignments are not visible, these are the hidden variables.
Θ are the parameters of the model. These parameters must be defined in order to ensure the model will have some meaningful generative story. With too many parameters, it will be difficult to train. If we have too less parameters, the model will be not general enough to describe all the data.
Now, we have a probabilistic model of *f* and *a* given *e* and Θ.

Sketch of Learning Algorithm

1. Likelihood maximization for incomplete data:

$$p(f|e,\Theta) = \sum_a p(f,a|e,\Theta) \rightarrow \max_\Theta$$

The data can be used to estimate Θ using Likelihood maximization. However, since *a* is not observable, we don't know how to take derivatives to get maximum likelihood estimations. As *a* is not observable, we need to sum over all possible word alignments.
Likelihood maximization for incomplete data means that there are some hidden variables that we do not see. The EM-algorithm is used to solve this problem.

Sketch of Learning Algorithm

1. Likelihood maximization for the incomplete data:

$$p(f|e,\Theta) = \sum_a p(f,a|e,\Theta) \rightarrow \max_\Theta$$

2. EM-algorithm to solve likelihood maximization

Iterative process:

- E-step: estimates posterior probabilities for alignments
- M-step: updates Θ – parameters of the model

The EM-algorithm is an iterative process that has two steps: E-step and M-step. The E-step will get estimates the hidden variables – it will determine the best alignments that can be produced given the perimeters Θ.
The M-step will get the updates for the parameters that maximize the likelihood given the best guess for word alignments.

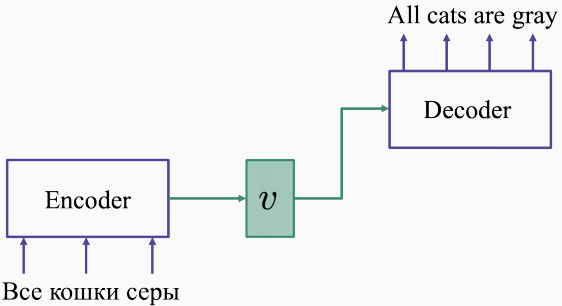
Sketch of Learning Algorithm

- The generation of the target sentence $p(f, a|e)$ include:
  - Generation of length of the foreign (target sentence) $p(J|e)$
  - Generation of an alignment for each word
  - Generation of the word
- Each of these steps envolves a complex formula
- Several different models assume certain simplifications in order to reduce the complexity of these models

Encoder-decoder Architecture

Encoder-decoder Architecture

- Encoder-decoder architecture was proposed for machine translation originally
- However, they are applied to many other tasks:
  - Summarization or simplification of the texts
  - Sequence to sequence chatbots
  - ...



Encoder-decoder Architecture

General idea of the architecture

- From an input sequence we would want to get some sequence as the output
- An encoder process the input sequence and produce hidden representation over the input sentence (green vector)
- The green hidden vector tries to encode the whole meaning of the input sentence
- Sometimes this vector is also called thought vector, because it encodes the thought of the sentence
- An decoder has the task to decode this thought vector or context vector into some output representation

Sequence to Sequence

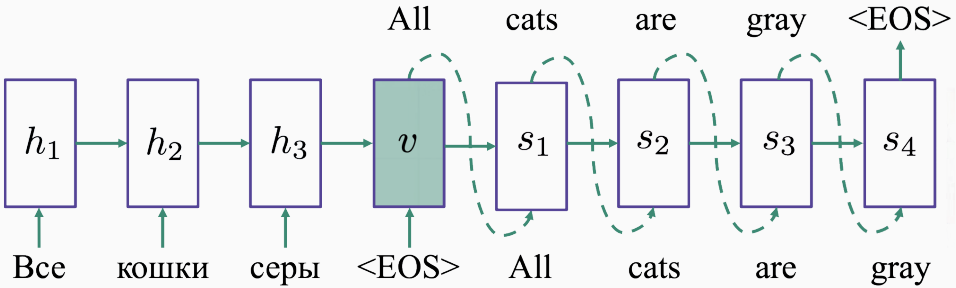
Types of encoders:

- Recurrent Neural Networks (RNN)
- Convolutional Neural Networks – can be very fast, and they can also encode the meaning of the sentence
- Recursive Neural Networks – use the syntax of the language and build the representation hierarchically from from bottom to the top

Sequence to Sequence

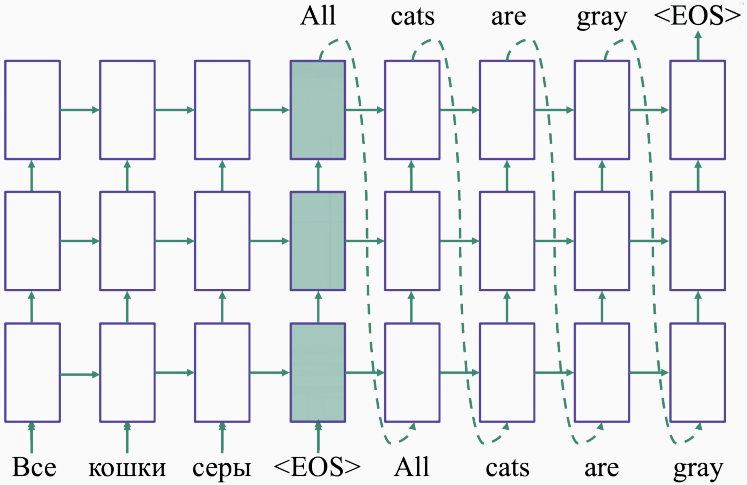
A RNN based encoder

- The RNN module encodes the input sentence until reach the < *EOS* > token
- At this point, the actual state is the thought vector or context vector
- Now, the decoding process starts – as any other language model, we fit the output of the previous state as the input to the next state, and generate the next words one by one



Sequence to Sequence

Stacking several LSTM layers

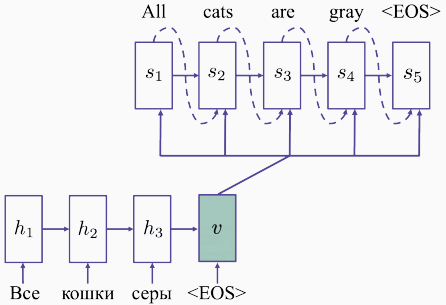


Sequence to Sequence

- The previous architectures present a shortcoming: the context vector can be forgotten
  - If we only feed *v* as the inputs to the first state of the decoder, then it is likely to forget about *v* when it come to the end of the output sentence
  - So it would be better to feed it at every moment

In this architecture, every stage of the decoder combines three sources:

- previous state
- context vector
- current input which is the output of the previous state



Cho et. al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014.

Sequence to Sequence

$$p(y_1, \ldots y_J | x_1, \ldots x_I) = \prod_{j=1}^{J} p(y_j | v, y_1, \ldots y_{j-1})$$

- Encoder: maps the source sequence to the hidden vector

  RNN: $h_i = f(h_{i-1}, x_i)$ $\quad v = h_I$

- Decoder: performs language modelling given this vector

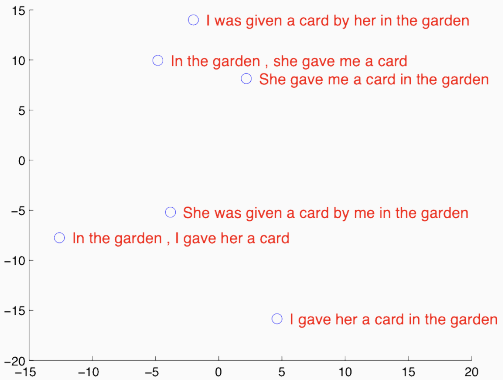
  RNN: $s_j = g(s_{j-1}, [y_{j-1}, v])$

- Prediction (using *softmax* to get the probability of the current word):

  $p(y_j | v, y_1, \ldots y_{j-1}) = softmax(Us_j + b)$

Meaning of Hidden Representations

Plotting of v vectors after dimension reduction to 2 dimensions:



Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Network, 2014.

Encoder-decoder Architecture

Attention Mechanism

Attention Mechanism Motivation

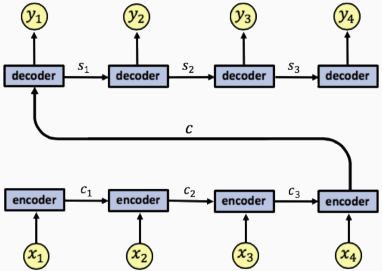
- RNN have been used successfully for many tasks involving sequential data such as machine translation, sentiment analysis, image captioning, time-series prediction, etc.
- Improved RNN models such as Long Short-Term Memory networks (LSTM) enable training on long sequences overcoming problems like vanishing gradients
- However, even LSTM have limitations to deal with long data sequences
- In machine translation, for example, the RNN has to find connections between long input and output sentences composed of dozens of words

Attention Mechanism Goal

Attention is a mechanism combined in the RNN allowing it to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling easier learning and of higher quality

Bahdanau et. al - Neural Machine Translation by jointly learning to align and translate, 2015.

A Basic RNN

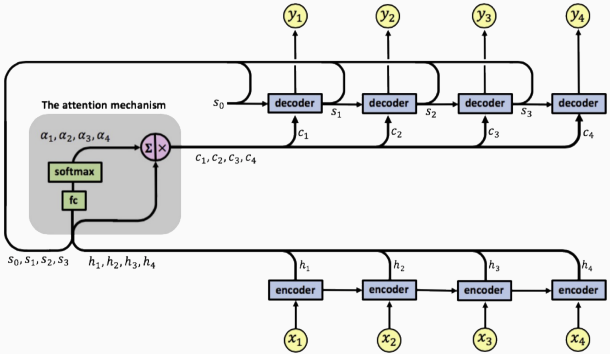


$x1$, $x2$, $x3$, $x4$: encoder input sequence
$c1$, $c2$, $c3$: encoder states
c: single output vector which is passed as input to the decoder
$s1$, $s2$, $s3$: decoder states (also a single-layered RNN)
$y1$, $y2$, $y3$, $y4$: decoder input sequence

A RNN encoder-decoder architecture with 4 time steps

"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 (c). 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"

RNN with Attention Mechanism



The attention mechanism is located between the encoder and the decoder. The attention's input is composed of the encoder's output vectors $h1$, $h2$, $h3$, $h4$ and the states of the decoder $s0$, $s1$, $s2$, $s3$. The attention's output is a sequence of vectors called context vectors denoted by $c1$, $c2$, $c3$, $c4$.

Context Vectors

- The context vectors enable the decoder to focus on certain parts of the input when predicting its output
- Each context vector is a weighted sum of the encoder's output vectors $h1, h2, h3, h4$
- Each vector c_i contains information about the whole input sequence (since it has access to the encoder states during its computation) with a strong focus on the parts surrounding the j-th vector of the input sequence
- The vectors $h1, h2, h3, h4$ are scaled by weights α_{ij} capturing the degree of relevance of input x_j to output at time i, y_i

The context vectors $c1, c2, c3, c4$ are given by:

$$c_i = \sum_{j=1}^{4} \alpha_{ij} h_j$$

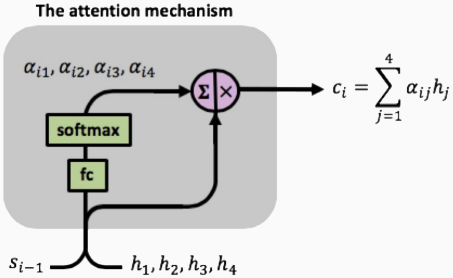
Learning Attention Weights

The attention weights are learned using an additional fully-connected network, *fc*

Computation of the attention weights is given by:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{4} \exp e_{ik}}, e_{ij} = fc(s_{i-1}, h_j)$$

The attention weights are learned using the attention fully-connected network and a softmax function:

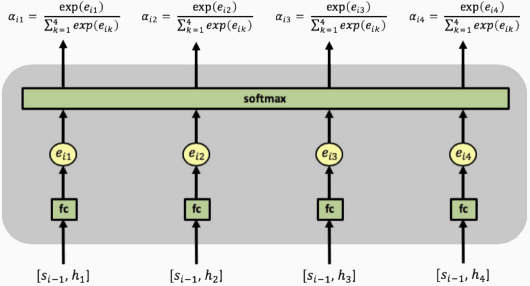


At time step *i*, the mechanism has *h*1, *h*2, *h*3, *h*4 and s_{i-1} as inputs. It uses the *fc* NN and the softmax function to compute the attention weights $\alpha_{i1}, \alpha_{i2}, \alpha_{i3}, \alpha_{i4}$. The weights are then used in the computation of the context vector c_i.

Learning Attention Weights

The fully-connected network *fc* receives the concatenation of vectors $[s_{i-1}, h_j]$ as input at time step i. The network has a single fully-connected layer, the outputs of the layer, denoted by e_{ij}, are passed through a softmax function computing the attention weights, which lie in $[0, 1]$.

The same *fc* network is used for all the concatenated pairs $[s_{i-1}, h_1]$, $[s_{i-2}, h_2]$, $[s_{i-3}, h_3]$, $[s_{i-4}, h_4]$, meaning there is **a single network learning the attention weights**.



Learning Attention Weights

The attention weights α_{ij} reflect the importance of h_j with respect to the previous hidden state s_{i-1} in deciding the next state s_i and generating y_i. A large α_{ij} attention weight causes the RNN to focus on input x_i (represented by the encoder's output h_j), when predicting the output y_i

Training: The *fc* network is trained along with the encoder and decoder using backpropagation – the RNN prediction error terms are backpropagated backward through the decoder, then through the *fc* attention network and from there to the encoder

Dimensions of the weight matrix: A RNN with 4 input time steps and 4 output time steps will have a 4×4 of the attention matrix, connecting between every input to every output

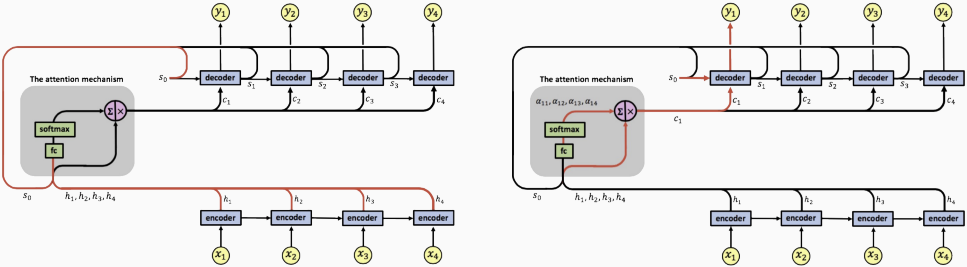
Attention Mechanism

"The most important distinguishing feature of this approach from the basic encoder–decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed-length vector. We show this allows a model to cope better with long sentences."

Computation of the Attention Weights and Context Vectors

Time step 1:

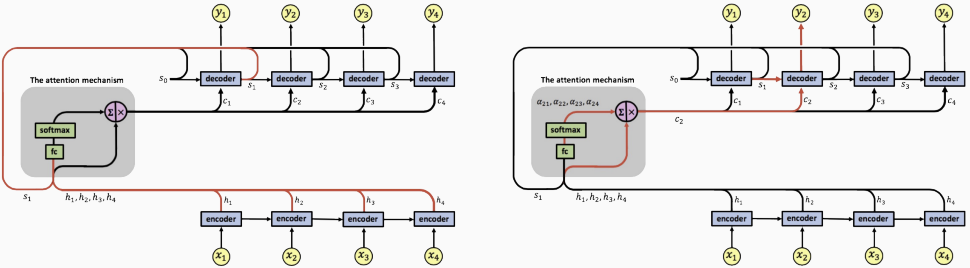
- Computation of vectors h_1, h_2, h_3, h_4 by the encoder
- The attention mechanism computes the first set of attention weights α_{11}, α_{12}, α_{13}, α_{14}, using the first attention input sequence $[s_0, h_1], [s_0, h_2], [s_0, h_3], [s_0, h_4]$, enabling the computation of the first context vector c_1
- The decoder now uses $[s_0, c_1]$ and computes the first RNN output y_1



Computation of the Attention Weights and Context Vectors

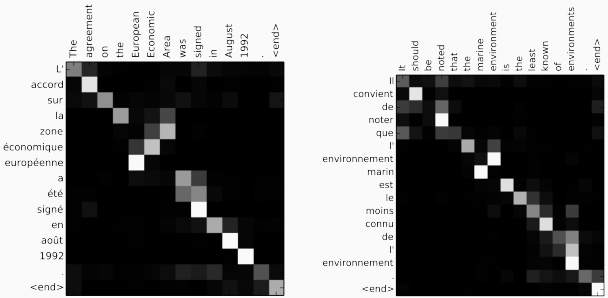
Time step 2:

- The attention mechanism has as input the sequence $[s_1, h_1], [s_1, h_2], [s_1, h_3], [s_1, h_4]$
- The second set of attention weights α_{21}, α_{22}, α_{23}, α_{24} enabling the computation of the second context vector c_2
- The decoder uses $[s_1, c_2]$ and computes the second RNN output y_2

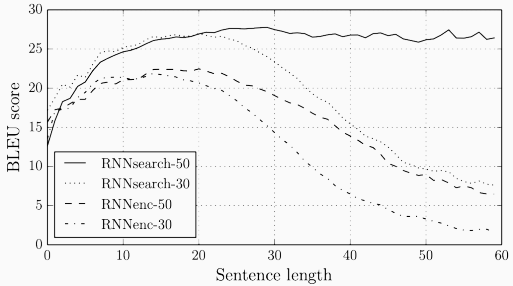


Example: Attention (Alignments)

- English-French machine translation
- Two alignments found by the attention RNN – The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively
- Each pixel shows the weight α_{ij} of the j-th source word and the i-th target word, in grayscale (0: black, 1: white)



Results



The *RNNenc* models refer to models based on an encoder-decoder architecture without attention mechanism. The *RNNsearch* models use an attention mechanism. The numbers 30 and 50 refer to the maximum sequence lengths used in the training dataset.

Bahdanau et. al - Neural Machine Translation by jointly learning to align and translate, 2015.

How to Deal with the Vocabulary in MT

Outline

- Computing softmax for a large vocabulary is slow
  - **Hierarchical softmax**
- Even a large vocabulary has *OOV words*:
  - Copy mechanism
  - Sub-word modeling
    - Word-character hybrid models

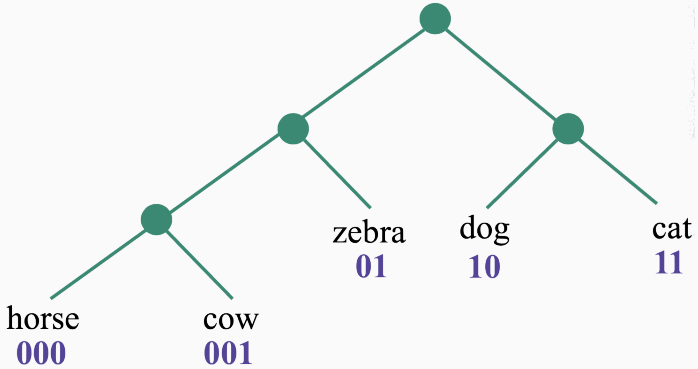
Hierarchical Softmax

A procedure to compute softmax in a fast way

Each word is uniquely represented by a binary code:

- 0 means “go left”, 1 means “go right”

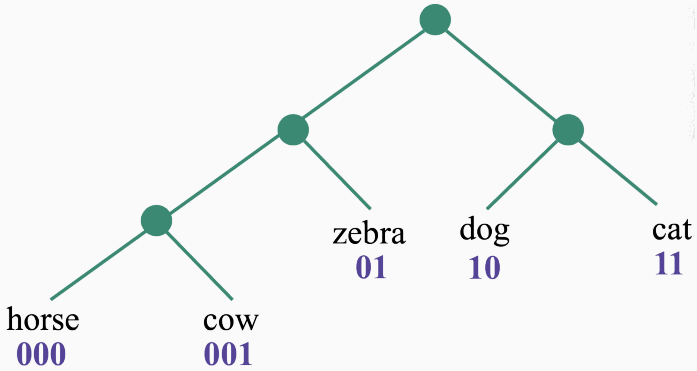
The idea is to build a binary tree for the words. For example, the zebra will have code 01, which means that first, we go to the left, it is 0, and then we go to the right, it is 1.



Hierarchical Softmax

E.g. for zebra the code is $d = (0, 1) \rightarrow d_1 = 0, d_2 = 1$

Each word will have an unique code



Scaling Softmax

Express the probability of a word (zebra) as a product of probabilities of the binary decisions along the path (d1 , d2).

$$p(w_n = w|w_1^{n-1}) = \prod_i p(d_i|w_1^{n-1})$$

The probabilities of words sum to 1.

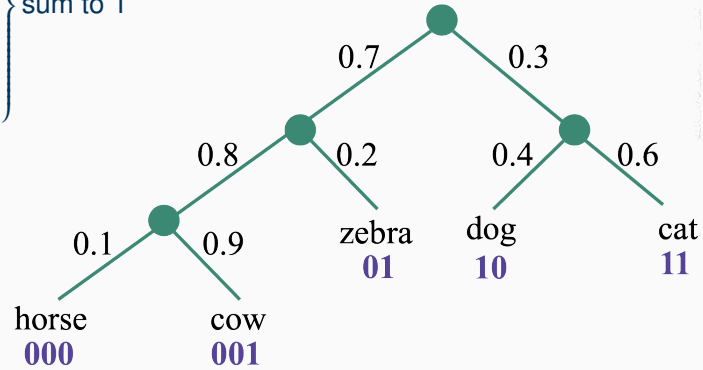
Rong, X. - word2vec Parameter Learning Explained, 2016.

Computing the probabilities for words using their binary codes
Instead of computing the probability of the word and normalizing it across the whole vocabulary, we split it into separate terms.
Each term is the probability of the digit (probability of the next decision in the tree, whether to go to the left or to the right). These probabilities are used in the product to estimate the probability of the whole word.

Hierarchical Softmax

$0.7 \cdot 0.8 \cdot 0.1$
$0.7 \cdot 0.8 \cdot 0.9$
$0.7 \cdot 0.2$
$0.3 \cdot 0.4$
$0.3 \cdot 0.6$
} sum to 1

In the hierarchical softmax model, there is no output vector representation for words. Instead, each of the $V - 1$ inner units (green nodes) has an output vector, where V is the number of words in the vocabulary



Hierarchical Softmax

Model binary decisions along the path in the tree:

$$p(w_n = w|w_1^{n-1}) = \prod_i p(d_i|w_1^{n-1})$$

How to construct a tree (balanced vs. semantic approaches):

- Based on some pre-built ontology
- Based on semantic clustering from data
- Huffman tree (based on the frequency of the words – the most frequent words stay closer to the root of the tree)
- Random

Outline

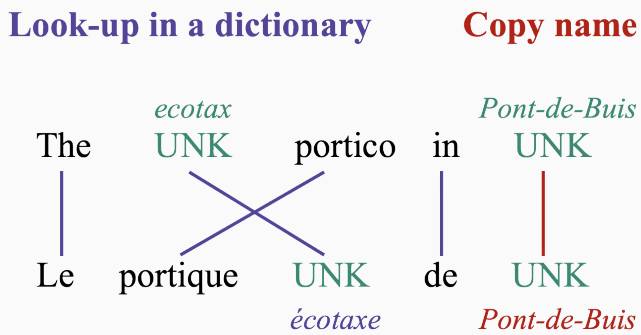
- Computing softmax for a large vocabulary is slow
  - Hierarchical softmax
- Even a large vocabulary has *OOV words*:
  - **Copy mechanism**
  - Sub-word modeling
    - Word-character hybrid models

Copy Mechanism

- Scaling *softmax* is insufficient
- What do we do with OOV words?
  - Names, numbers, rare words ...

Copy Mechanism

- Scaling *softmax* is insufficient
- What do we do with OOV words?
  - Names, numbers, rare words . . .



Copy Mechanism

Algorithm:

- Learn relative positions for UNK tokens with Neural Machine Translation
- Post-process the translation:
  - Copy the source word
  - Look up in a dictionary

Open Vocabulary

Still problems:

- Multi-word alignment: Solar system → Sonnensystem
- Rich morphology: nejneobhospodařovávatelnějšímu
- Informal spelling: goooooood morning !!!!!

Outline

- Computing softmax for a large vocabulary is slow
  - Hierarchical softmax
- Even a large vocabulary has *OOV words*:
  - Copy mechanism
  - **Sub-word modeling**
    - Word-character hybrid models

Outline

- Computing softmax for a large vocabulary is slow
  - Hierarchical softmax
- Even a large vocabulary has *OOV words*:
  - Copy mechanism
  - Sub-word modeling
    - **Word-character hybrid models**

Character-based Models

Character-based encoder is good for source languages with rich morphology

- CNNs on characters

  For example, let's consider *drinkable* as OOV - we can use a CNN that has convolutions that represent the meaning of *drink*, and other convolutions that represent the meaning of *able* - we can build the meaning of the whole word

- Bi-LSTMs to build word embeddings from characters

  Bidirectional LSTMs can be used on word level as well as on character level

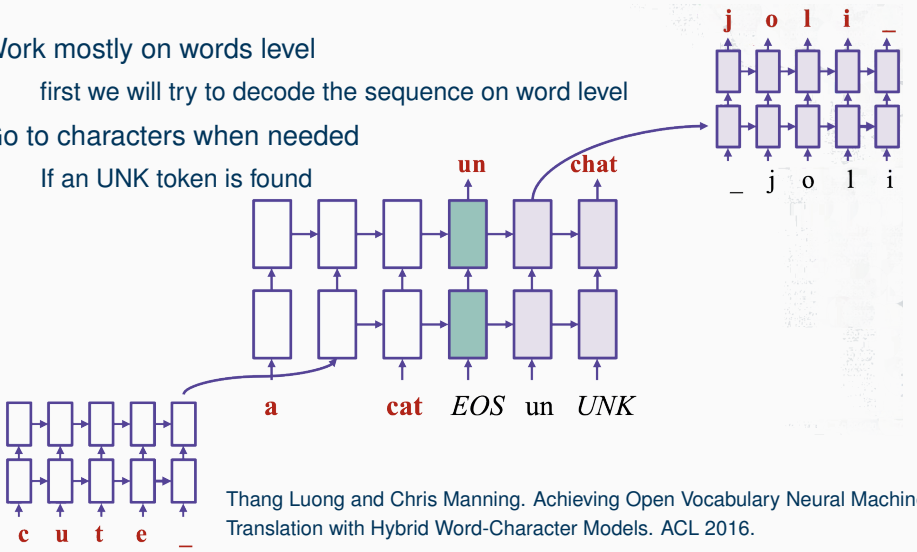
Hybrid Models

- Work mostly on words level

  first we will try to decode the sequence on word level

- Go to characters when needed

  If an UNK token is found



Thang Luong and Chris Manning. Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models. ACL 2016.

Encode-decode Attention Architecture to Implement a Chatbot

Chatbots

Goal-oriented:

- Narrow domain
- Specific questions and tasks
- *Example*: call center to help custumers
- *Model*: usually retrieval-based ranked from a pre-defined responses repository

Chit-chat:

- General conversation
- Human-like experience
- *Example*: entertaining bot
- *Model*: generative to produce new answers

Models Pros and Cons

Retrieval-based models

- use a repository of predefined responses
- no grammatical mistakes
- unable to handle unseen cases

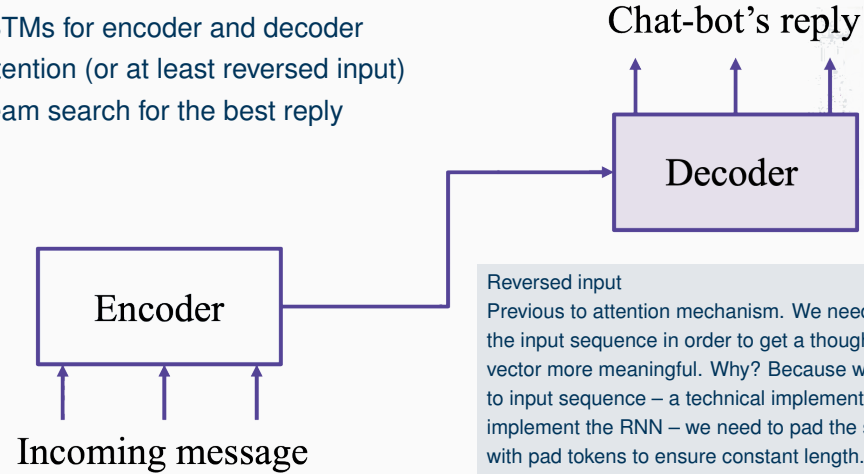
Generative models

- generate new responses from scratch
- can make mistakes (especially on longer sentences)
- impression we are talking to a human

In this chapter we will deal only with generative models for conversational chatbots.

Sequence to Sequence

- LSTMs for encoder and decoder
- Attention (or at least reversed input)
- Beam search for the best reply



Reversed input

Previous to attention mechanism. We need to reverse the input sequence in order to get a thought (context) vector more meaningful. Why? Because we have to pad to input sequence – a technical implementation detail to implement the RNN – we need to pad the sequences with pad tokens to ensure constant length. If we tag the sequence at the end, the end the sequence is absolutely not meaningful, and the context vector will not be usefull.

Padding

EOS : End of sentence

PAD : Filler

GO : Start decoding

UNK : Unknown; word not in vocabulary

Q: How are you?

A: I am fine

Q: [PAD,PAD,PAD,PAD,PAD,PAD,“?”,“you”,“are”, “How”]

A: [GO, “I”, “am”, “fine”, “.”, EOS, PAD, PAD, PAD, PAD]

Bucketing

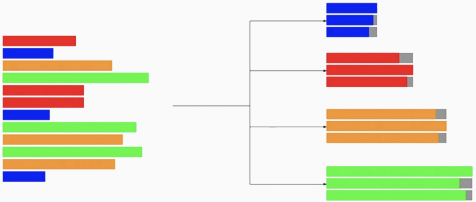
Bucketing: alternative to padding.

Bucketing solves the problem of high variance of the length of sentences by putting them into buckets of different sizes.

Buckets for the length intervals]0, 5],]5, 10],]10, 20], . . .

Q: How are you?

A: I am fine



Q: [PAD,“?”,“you”,“are”, “How”]

A: [GO, “I”, “am”, “fine”, “.”, EOS, PAD,

Chatbots’ Problems

Trained on Movies Subtitles

A human tool machine talk, trained for movie subtitles:

Human: what is the purpose of living?

Machine: to live forever.

Human: what is the purpose of existence?

Machine: to find out what happens when we get to the planet earth.

Human: where are you now?

Machine: i'm in the middle of nowhere.

It seems to work, but has some problems though:

- Unrealistic topics
- Too dramatic responses

The behaviour is dependent of the domain data.

Oriol Vinyals, Quoc Le. A Neural Conversational Model, 2015.
https://www.wired.com/2015/06/google-made-chatbot-debates-meaning-life/

Trained on Calls

Machine: hi
Human: hi
Machine: could you please let me know what is the operating system you are using?
Human: linux
Machine: ok this is the issue with the system performance, nasser, tools.
Human: what is that ?
Machine: is it a windows machine?
Human: i am using linux
Machine: let me take a remote of your machine and check if issue has been pushed properly.
Human: ok

The chatbot discourse isn't consistent. The system does not track the intent and the context of the conversation.

Oriol Vinyals, Quoc Le. A Neural Conversational Model, 2015.
https://www.wired.com/2015/06/google-made-chatbot-debates-meaning-life/

Context of the Conversation

The previous behaviour can be improved with a **RNN to keep track of the intents:**

Human: hi

Machine: how may i help you today?

Human: my computer is not responding

Machine: oh ! i sorry to hear that. are you getting an error message?

Human: yes

Machine: what recent changes did you made with this computer?

Human: i updated my computer

Machine: i see . is this on windows 7?

Human: no it is on linux

Machine: ok . are you using the affected computer right now?

Human: no

Yao et. al. Attention with Intention for a Neural Network Conversation Model, 2015.

Coherent Personality

4-layer seq2seq trained on 25 million Twitter snippets:

Human: Where do you live now?
Machine: I live in Los Angeles.
Human: In which city do you live now?
Machine: I live in Madrid.
Human: In which country do you live now?
Machine: England, you?

Persona-based model to memorize a personality (the bot is trained with coherent pieces of dialogues from different persons):

Human: Where are you from?
Machine: I'm from England.
Human: In which city do you live now?
Machine: I live in London.

Li et. al. A Persona-Based Neural Conversation Model, 2016.

Diversity of the Responses

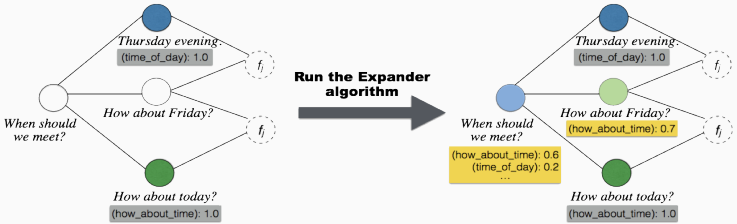
Top-3 suggestions of response in an email smart reply system:

- How about tomorrow?
- Wanna get together tomorrow?
- I suggest we meet tomorrow.

All the alternatives are similar – there is no diversity.

Kannan et. al. Smart Reply: Automated Response Suggestion for Email, KDD 2016.

Intents Clustering



Intent clustering for the responses. A supervised dataset about the types of the responses is used. For example, we can have the label, “how_about_time” for “How about Friday”. So we have a graph of different responses, and similarities between the responses and labels built by a distributional semantics model. Now, we have some labeled nodes in this graph from the supervised data, and we want to propagate this knowledge to other labels of the graph. This technique is called label propagation, and Expander is a library that implements it. The labels of the responses will be propagated in such a way that close responses will get close labels. Then we pick up one example from every cluster, and suggest it for the user.

Kannan et. al. Smart Reply: Automated Response Suggestion for Email, KDD 2016.

Google Smart Reply

Query	Top generated responses
Hi, I thought it would be great for us to sit down and chat. I am free Tuesday and Wenesday. Can you do either of those days? Thanks! –Alice	I can do Tuesday. I can do Wednesday. How about Tuesday? I can do Tuesday! I can do Tuesday. What time works for you? I can do Wednesday! I can do Tuesday or Wednesday. How about Wednesday? I can do Wednesday. What time works for you? I can do either.

Kannan et. al. Smart Reply: Automated Response Suggestion for Email, KDD 2016.

Conclusion

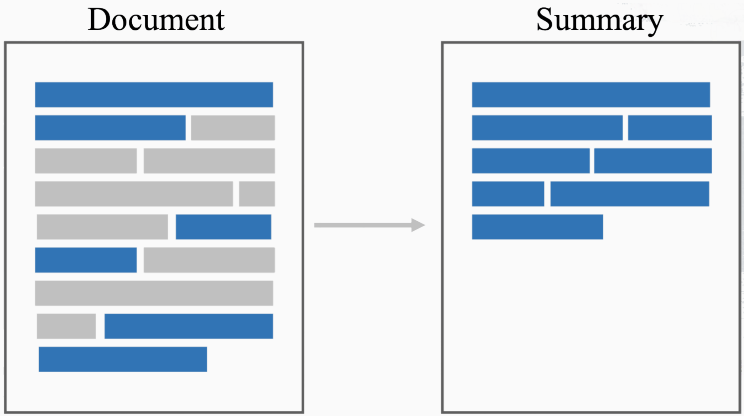
- Even though the bots can try to have some meaningful conversation, there are still so many problems with them
- It is easy to understand that we are speaking to a bot and not to a human
- Current models are still not humans

Other Sequence to Sequence Tasks

Sequence to Sequence

- Machine Translation
- Summarization
- Text simplification
- Language to code
- Chit-chat bot
- Question answering
- Listen, attend and spell: speech recognition
- Show, attend and tell: image caption generation
- ...

Summarization



Summarization

Original Text

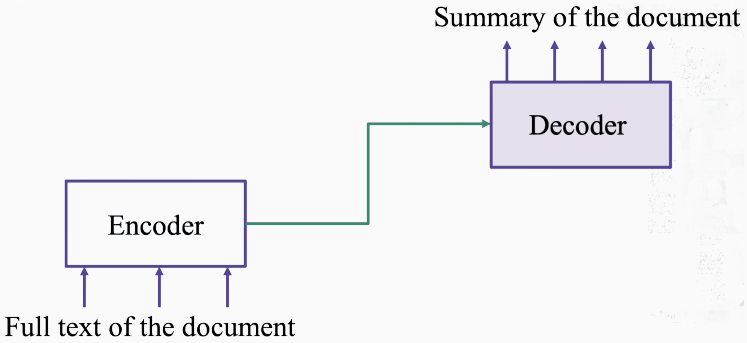
Alice and Bob took the train to visit the zoo. They saw a baby giraffe, a lion, and a flock of colorful tropical birds.

Types of summarization:

- **Extractive Summary** – extract some pieces of the original text
  Alice and Bob visit the zoo. saw a flock of birds.
- **Abstractive summary** – generate a summary that is not necessarily from the input text, but summarizes the whole text
  Alice and Bob visited the zoo and saw animals and birds.

https://research.googleblog.com/2016/08/text-summarization-with-tensorflow.html

Sequence to Sequence



Extractive summary with sequence to sequence model – from a full text as the input, we get a summary as the output

Google Research Blog

Dataset: Annotated English Gigaword (examples and their headlines) – 10 million documents

`catalog.ldc.upenn.edu/LDC2012T21`

Model: sequence to sequence with attention + beam search

Code: open-source TF implementation

`https://github.com/tensorflow/models/tree/master/research/textsum`

`https://git.dst.etit.tu-chemnitz.de/external/tf-models/-/tree/master/research/textsum`

Google Research Blog

Input: Article 1st sentence	Model-written headline
metro-goldwyn-mayer reported a third-quarter net loss of dlrs 16 million due mainly to the effect of accounting rules adopted this year	mgm reports 16 million net loss on higher revenue
starting from july 1, the island province of hainan in southern china will implement strict market access control on all incoming livestock and animal products to prevent the possible spread of epidemic diseases	hainan to curb spread of diseases
australian wine exports hit a record 52.1 million liters worth 260 million dollars (143 million us) in september, the government statistics office reported on monday	australian wine exports hit record high in september

https://research.googleblog.com/2016/08/text-summarization-with-tensorflow.html

Simplification

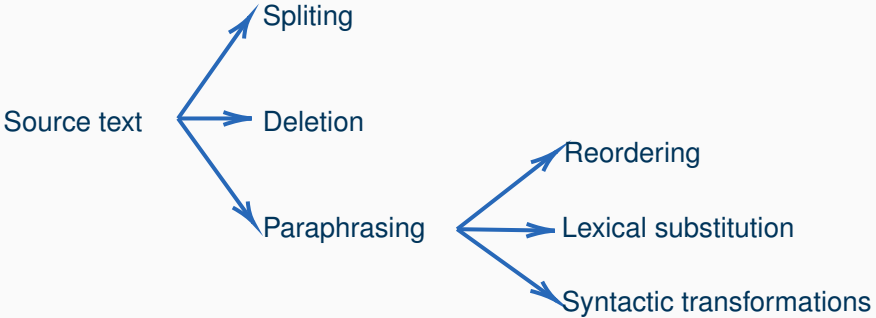
Text simplification – reducing the lexical and syntactical complexity of text

Simple Wikipedia can be used as dataset to train simplification task

a.	Normal:	As Isolde arrives at his side, Tristan dies *with her name on his lips*.
	Simple:	As Isolde arrives at his side, Tristan dies *while speaking her name*.
b.	Normal:	Alfonso Perez *Munoz, usually referred to as Alfonso,* is a former Spanish *footballer, in the striker position.*
	Simple:	Alfonso Perez is a former Spanish *football player*.
c.	Normal:	Endemic types *or species* are *especially* likely to develop on islands because *of their geographical isolation*.
	Simple:	Endemic types are *most* likely to develop on islands because *they are isolated*.
d.	Normal:	The reverse process, producing electrical energy from mechanical, energy, is accomplished by a generator or dynamo.
	Simple:	A dynamo or an electric generator does the reverse: it changes mechanical movement into electric energy.

Coster et. al. Simple English Wikipedia: A New Text Simplification Task, 2011.

Operations to Simplify Text



Operations to simplify text

We can delete, paraphrase, or split one sentence into two simpler smaller sentences. Paraphrasing can reorder words, or do some syntactic analysis of the sentence in order to substitute one syntactic structure by other.

Coster et. al. Simple English Wikipedia: A New Text Simplification Task, 2011.

Rule-based Approach for Paraphrasing

Lexical	[RB]	solely	→	only
	[NN]	objective	→	goal
	[JJ]	undue	→	unnecessary
Phrasal	[VP]	accomplished	→	carried out
	[VP/PP]	make a significant contribution	→	contribute greatly
	[VP/S]	is generally acknowledged that	→	is widely accepted that
Syntactic	[NP/VP]	the manner in which NN	→	the way NN
	[NP]	NNP 's population	→	the people of NNP
	[NP]	NNP 's JJ legislation	→	the JJ law of NNP

Example paraphrase rules in the Paraphrase Database (PPDB) that result in simplifications of the input. The rules are synchronous context-free grammar (SCFG) rules where uppercase indicates non-terminal symbols. Non-terminals can be complex symbols like VP/S which indicates that the rule forms a verb phrase (VP) missing a sentence (S) to its right. The final syntactic rule both simplifies and reorders the input phrase.

- Synchronous context-free grammar (SCFG) rules
- Uppercase indicates non-terminal symbols
- Paraphrase Database `http://www.cis.upenn.edu/~ccb/ppdb/`

Xu et. al. Optimizing Statistical Machine Translation for Text Simplification, 2016.

Simplification with Deep Learning

Encoder-decoder framework – However the network might learn just to **copy** the content because it doesn't have any simplification objective built in. How do we force it to **simplify**?

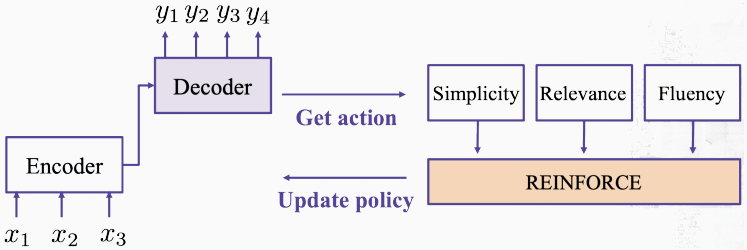
Reinforcement learning can be used to do **weak supervision**.

- **Action:** output next word y_j
- **Policy:** $p(y_j|\mathbf{x}, y_1, \ldots y_{j-1})$
- **Reward:** Adequacy + Fluency + Simplicity
  - Adequacy is about whether the simplified sentence is still about the same fact as the original sentence
  - Fluency is about the coherence of the sentence and the language model
  - Simplicity is whether the simplified version is indeed simpler than the original one

Rewards come only when the whole sequence is generated

Zhang, Lapata. Sentence Simplification with Deep Reinforcement Learning, 2017.

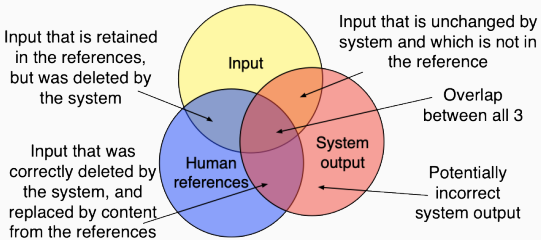
Simplification



The encoder-decoder agent generate some sequences. Every output sequence get some reward based on simplicity, relevance and fluency. This reward is used by reinforce algorithm to update the policy of the agent. The agent, on the next step, will be more likely to generate those actions that give higher rewards. It is similar to gradient descent, but the diference is that the rewards are usually not differentiable. So reinforcement learning is helpful when we cannot use a lost function and optimize it.

Zhang, Lapata. Sentence Simplification with Deep Reinforcement Learning, 2017.

How to Measure Simplicity



SARI (system against references and input) – arithmetic average of n-gram precision and recall of operations:

- addition
- copying
- deletion

Xu et. al. Optimizing Statistical Machine Translation for Text Simplification, 2016.