NMT

Mark Fishel, TartuNLP
14th MT Marathon
August 26, 2019

NMT

Questions:
sli.do/#MTM19

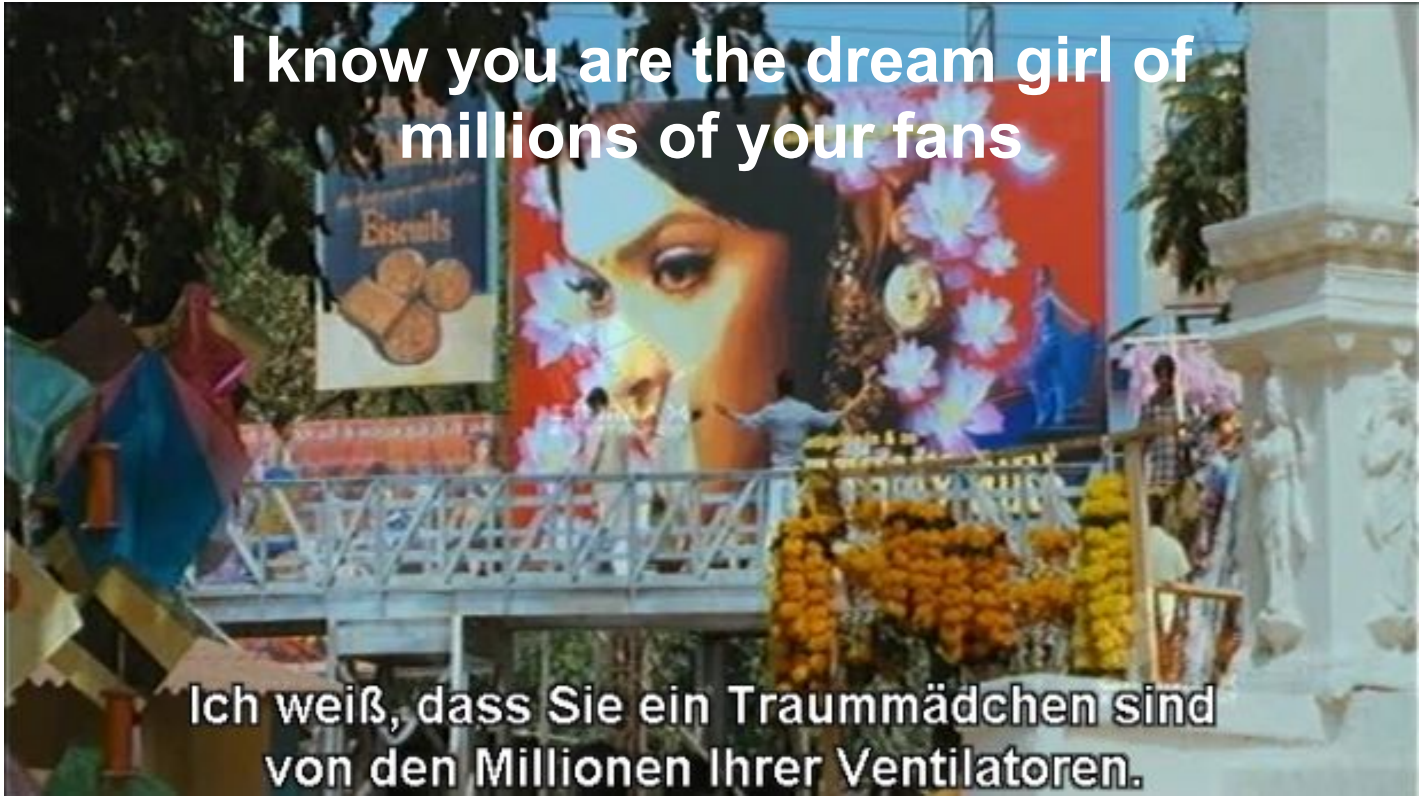
Slides:
bit.ly/2HpuwzW

Mark Fishel, TartuNLP
14th MT Marathon
August 26, 2019

Outline

1. why MT
2. how machines learn
3. how neural networks:
    - work ← algebra & geometry
    - learn ← calculus
    - handle language ← probabilities
    - translate ← black magic

Why MT?


I know you are the dream girl of
millions of your fans
Biscuits
Ich weiß, dass Sie ein Traummädchen sind
von den Millionen Ihrer Ventilatoren.

What is good MT?

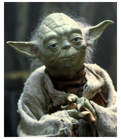
- **Human:** you must not go there
- **MT:** you must go there

What is good MT?

- **Human:** you must not go there
- **MT:** you must go there
  - meaning: very bad / correcting: easy

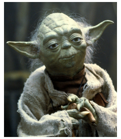
What is good MT?

- **Human:** you must not go there
- **MT:** you must go there
  - meaning: very bad / correcting: easy
- **MT:** head in there you should not



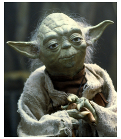
What is good MT?

- **Human:** you must not go there
- **MT:** you must go there
  - meaning: very bad / correcting: easy
- **MT:** head in there you should not
  - meaning: comprehensible / correcting: awful



What is good MT?

- **Human:** you must not go there
- **MT:** you must go there
  - meaning: very bad / correcting: easy
- **MT:** head in there you should not
  - meaning: comprehensible / correcting: awful



- **Human:** The battery lasts 6 hours and it can be fully recharged in 30 minutes.
- **MT:** Six hour accumulator, to recharge totally half-an-hour takes

MT: applications

- post-editing
  - automatic translation draft, to be fixed by human post-editors
  - can be more efficient than manual translation -- IF the MT quality is good enough

MT: applications

- post-editing
  - automatic translation draft, to be fixed by human post-editors
  - can be more efficient than manual translation -- IF the MT quality is good enough
- approximate understanding
  - browse the web
  - get the general idea of a text
  - ok IF errors do not hinder the meaning on average

MT: evaluation

1. Manual assessment
   - “is this translation adequate/fluent/etc?” × ∞
   - expensive but more reliable
     (though humans disagree)

MT: evaluation

2. Post-editing
   - manual post-editing
   - automatic comparison of pre-edit and post-edit
     - each fix is an error

   = "HTER" (human translation edit rate)

   - still expensive but reliable enough
     (humans disagree even more)

MT: evaluation

3. Automatic, via references
   - take a translated test set
   - automatically translate the source side
   - compare MT output (hypothesis) to etalon translation (reference) e.g. with `chrF`
     - or `BLEU`
   - cheap but approximate, single reference = bad, score often hard to interpret, etc. ad nauseam

MT: evaluation

4. Automatic, w/o references
    - take the source text and hypothesis translation
    - predict how good it is
    - how: black magic
        - and it doesn’t really work

MT: what makes it hard?

MT: what makes it hard?

- Ambiguity
  a. glasses:  or  ?

  b. The viking killed the pirate with a sword (?)

MT: what makes it hard?

- Ambiguity
  a. glasses:  or  ?
  b. The viking killed the pirate with a sword (?)

- Variation
  a. Notwithstanding the foregoing, the European Parliament or the Council may revoke the delegation of powers at any time. (legal text)
  b. The four guys we nabbed were just Lateesha's patsies. (TV subtitles)

MT: what makes it hard?

- Ambiguity
    a. glasses:  or  ?
    b. The viking killed the pirate with a sword (?)

- Variation
    a. Notwithstanding the foregoing, the European Parliament or the Council may revoke the delegation of powers at any time. (legal text)
    b. The four guys we nabbed were just Lateesha's patsies. (TV subtitles)

\+ Unseen words, rare languages, context -- and no AGI

MT: how?

1. Like any other programming task:
   - we explain to the computer, how we solve the task we want it to solve

MT: how?

1. Like any other programming task:
    - ~~we explain to the computer, how we solve the task we want it to solve~~
    - we express our knowledge of how we solve this task in a formal way (program/dictionary/rules/etc.)
    - do we know how humans translate?

MT: how?

1. Like any other programming task:
   - ~~we explain to the computer, how we solve the task we want it to solve~~
   - we express our knowledge of how we solve this task in a formal way (program/dictionary/rules/etc.)
   - do we know how humans translate?

~~No~~ Yesn't

MT: how?

1. Rule-based MT:

- replacing and reordering words/phrases
  - direct RBMT
- analyze input, transfer, generate output
  - transfer-based RBMT
- come up with an interlingua
  - interlingua-based RBMT

MT: how?

2. Show translation examples, let the program figure out how it is done

Un caffe per favore	One coffee please
Ciao, come stai?	Hi, how are you?
…	…

MT: how?

2. Show translation examples, let the program figure out how it is done
   - data-driven MT
     - Example-based / statistical / **neural MT**
   - need generalization = translate sentences unseen in the examples
   - still need to preset the “process outline”

Neural machine translation

1. let’s build a random text generator
    - it will learn to predict the next word after all previous words

would you like tea or …?

Neural machine translation

1. let’s build a random text generator
   - it will learn to predict the next word after all previous words

would you like tea or coffee?

dear …

Neural machine translation

1. let's build a random text generator
   - it will learn to predict the next word after all previous words

would you like tea or coffee?

dear (passengers / ladies and gentlemen / …)

Neural machine translation

1. let's build a random text generator
   - it will learn to predict the next word after all previous words

2. now let's teach it to generate random text (target language), given an input sentence (source language)

Neural machine translation

1. let’s build a random text generator
   ○ it will learn to predict the next word after all previous words

2. now let’s teach it to generate random text (target language), given an input sentence (source language)
   ○ … (lugupeetud reisijad)

Neural machine translation

1. let’s build a random text generator
   - it will learn to predict the next word after all previous words

2. now let’s teach it to generate random text (target language), given an input sentence (source language)
   - dear … (lugupeetud reisijad)

Neural machine translation

1. let’s build a random text generator
   - it will learn to predict the next word after all previous words

2. now let’s teach it to generate random text (target language), given an input sentence (source language)
   - dear passengers (lugupeetud reisijad)

Neural machine translation

1. let’s build a random text generator
   - it will learn to predict the next word after all previous words

2. now let’s teach it to generate random text (target language), given an input sentence (source language)
   - dear passengers ⟨/eos⟩ (lugupeetud reisijad)

Recap:

1. MT hard
   - but usable
2. Eval hard
3. NMT generates random text
   - one token at a time

TODO: check Slido

Machine learning

$$y = f(x)$$

Machine learning

$$y = f(x; \theta)$$

Machine learning

Machine learning: searching for θ automatically, while keeping f fixed.

$$y = f(x; \theta)$$

Machine learning: we need

1. Model space: set of possible values for θ

Machine learning: we need

1. Model space: set of possible values for θ
2. Quality measure: how good is a value of θ? (optimization criterion)
   - or error measure: how bad 😈 it is

Machine learning: we need

1. Model space: set of possible values for θ
2. Quality measure: how good is a value of θ? (optimization criterion)
   - or error measure: how bad 😈 it is
3. Search method: ~~brute force~~ a way to find a value for the model from the model space with a satisfactory quality/error measure

Machine learning

Programming ≈ writing a cooking recipe; e.g.

- bring water to boil
- add vegetables, bones
- add θ_1 tsp. salt and θ_2 tsp. pepper
- boil for 1 hour

Result = soup

Soup modelling

1. Model space
   - amount of salt and pepper: $\langle \theta_1 , \theta_2 \rangle$

2. Optimization criterion:
   - reduce number of complaints / aggression of clients

3. Search method:
   - make soup
   - react to client feedback

Modelling soup:

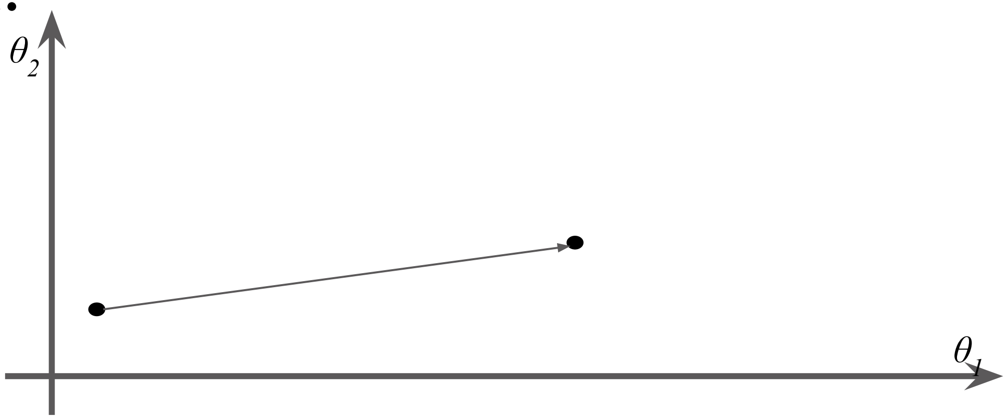
- parameters: $\theta = \langle \theta_1, \theta_2 \rangle$
  - θ_1 = amount of pepper
  - θ_2 = amount of salt
- search:

Modelling soup:

- parameters: $\theta = \langle\theta_1, \theta_2\rangle$
  - θ_1 = amount of pepper
  - θ_2 = amount of salt
- search:

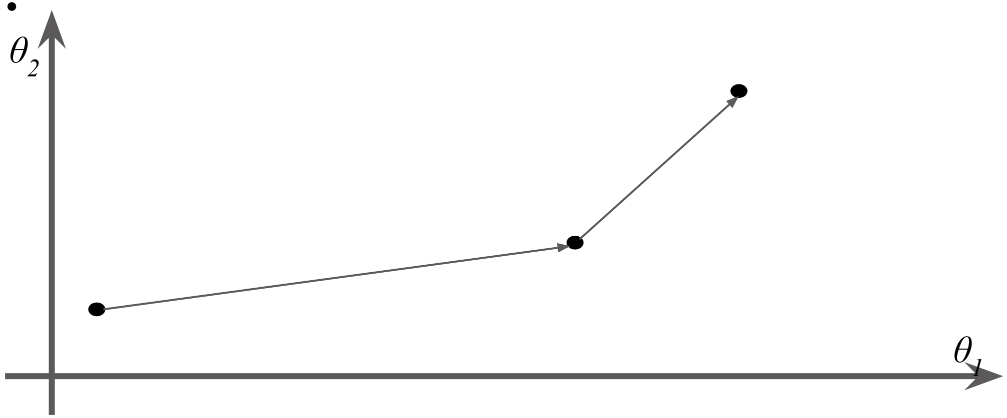
Modelling soup:

- parameters: $\theta = \langle \theta_1, \theta_2 \rangle$
  - θ_1 = amount of pepper
  - θ_2 = amount of salt
- search:



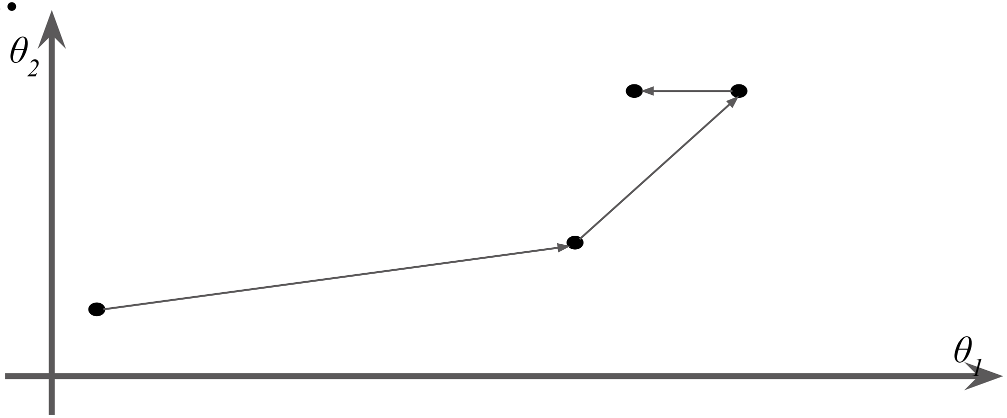
Modelling soup:

- parameters: $\theta = \langle \theta_1, \theta_2 \rangle$
  - θ_1 = amount of pepper
  - θ_2 = amount of salt
- search:



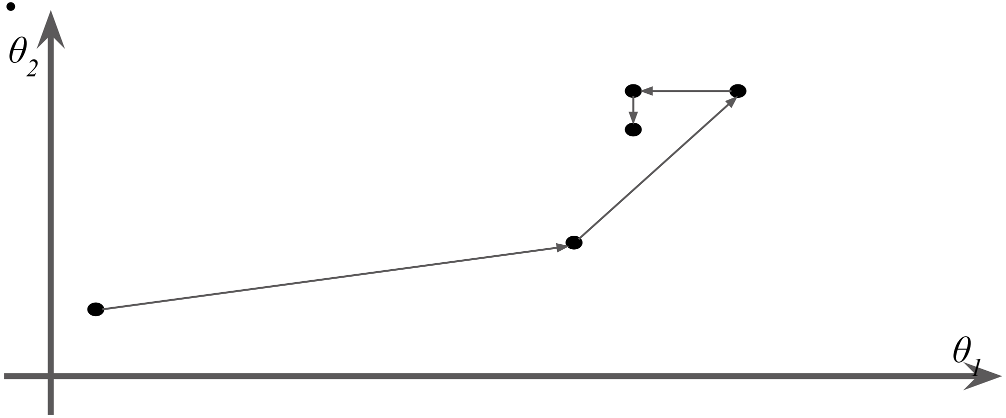
Modelling soup:

- parameters: $\theta = \langle\theta_1, \theta_2\rangle$
  - θ_1 = amount of pepper
  - θ_2 = amount of salt
- search:



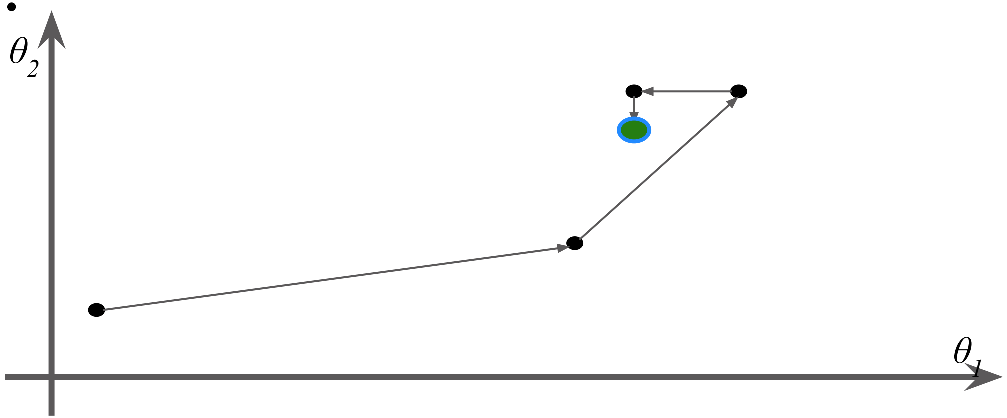
Modelling soup:

- parameters: $\theta = \langle \theta_1, \theta_2 \rangle$
  - θ_1 = amount of pepper
  - θ_2 = amount of salt
- search:



Modelling soup:

- parameters: $\theta = \langle \theta_1, \theta_2 \rangle$
  - θ_1 = amount of pepper
  - θ_2 = amount of salt
- search:



Machine learning: data

Training set: examples of input+output

Machine learning: data

Training set: examples of input+output

+ test set

Machine learning: data

Training set: examples of input+output

\+ test set

\+ dev set

Machine learning: underfitting

- keep getting bad results on the training set (and dev/test)
- model (f) too simplistic / not getting enough info on each input

Machine learning: overfitting

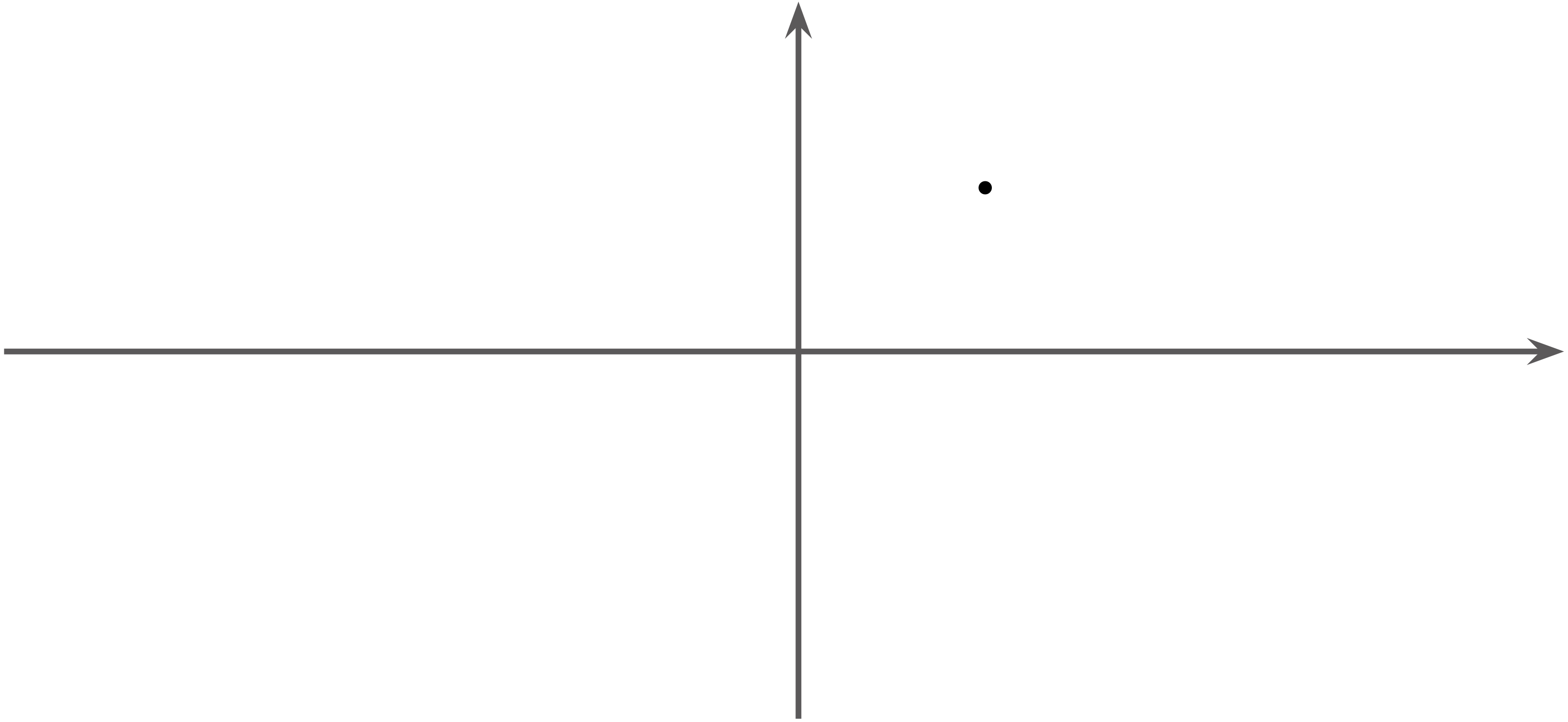
- great results on the training set
- bad results on dev/test
- model (f) too “smart”, memorizes training set
  - or training set too small

Recap:

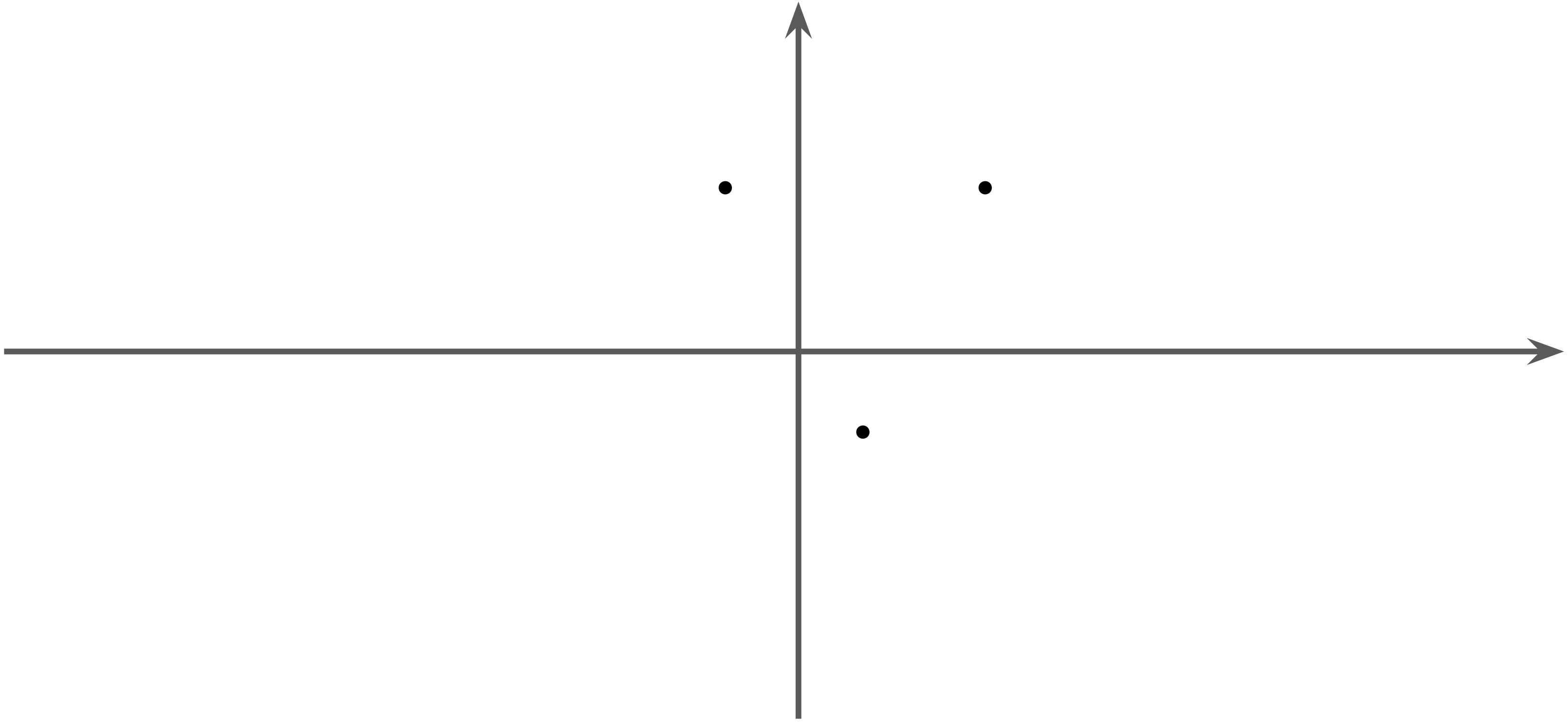
1. ML = finding the right parameters for a function
2. Need training data (+ dev and test)
3. Can solve obscure tasks
4. Not magic

TODO: check Slido

Vectors

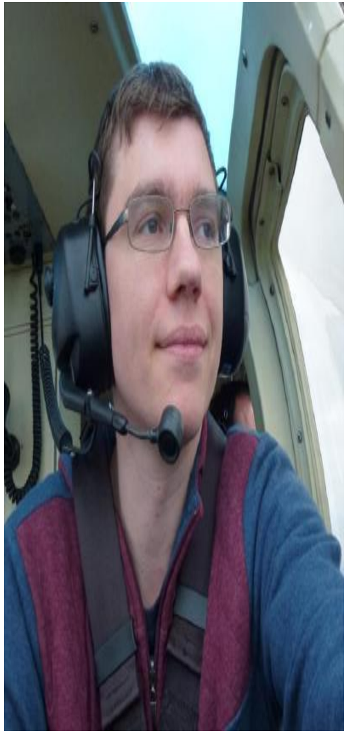


Vectors



Vectors

Vectors



Vectors

Vectors

Vectors

Vectors

Vectors

Vectors

Vectors

Vector transformations

vector: $\boldsymbol{x}^T = \langle x_1 , x_2 , \ldots \rangle$

vector functions: $\boldsymbol{g(x)}^T = \langle g(x_1) , g(x_2) , \ldots \rangle$

dot product: $\boldsymbol{x} \cdot \boldsymbol{y} = \sum_i x_i y_i = x_1 y_1 + x_2 y_2 + \ldots$

Vector transformations

matrix: $$\mathrm{A} = \begin{matrix} \langle a_{11}, a_{21}, a_{31}, \ldots \rangle \\ \langle a_{12}, a_{22}, a_{32}, \ldots \rangle \\ \langle \quad \ldots \quad \rangle \end{matrix} = \langle \boldsymbol{a}_{1\cdot}; \boldsymbol{a}_{2\cdot}; \boldsymbol{a}_{3\cdot}; \ldots \rangle$$

matrix product: $$\boldsymbol{x}^T\mathrm{A} = \langle \boldsymbol{xa}_1; \boldsymbol{xa}_2; \boldsymbol{xa}_3; \ldots \rangle$$

Vector transformations: matrix product

$$
\begin{array}{c|c}
 & \langle a_{11}, a_{21}, a_{31} \rangle \\
 & \langle a_{12}, a_{22}, a_{32} \rangle \\
\hline
\langle x_{12}, x_{22} \rangle & \langle y_1 \;, \; y_2, y_3 \;\rangle
\end{array}
$$

Vector transformations: matrix product

$$\begin{array}{c|c} & \langle a_{11}, a_{21}, a_{31} \rangle \\ & \langle a_{12}, a_{22}, a_{32} \rangle \\ \hline \boxed{\langle x_1, x_2 \rangle} & \langle \boxed{y_1}, y_2, y_3 \rangle \end{array}$$

$$x_1 a_{11} + x_2 a_{12}$$

Vector transformations: matrix product

$$
\begin{array}{c|c}
 & \langle a_{11}, a_{21}, a_{31} \rangle \\
 & \langle a_{12}, a_{22}, a_{32} \rangle \\
\hline
\langle x_1, x_2 \rangle & \langle y_1, y_2, y_3 \rangle
\end{array}
$$

$$x_1 a_{21} + x_2 a_{22}$$

Vector transformations: matrix product

$$\begin{array}{c|c} & \langle a_{11}, a_{21}, a_{31} \rangle \\ & \langle a_{12}, a_{22}, a_{32} \rangle \\ \hline \langle x_1, x_2 \rangle & \langle y_1, y_2, y_3 \rangle \end{array}$$

$$x_1 a_{31} + x_2 a_{32}$$

Vector transformations: matrix product

$$
\begin{array}{c|c}
 & \langle a_{11}, a_{21}, a_{31} \rangle \\
 & \langle a_{12}, a_{22}, a_{32} \rangle \\
\hline
\langle x_{11}, x_{21} \rangle & \langle y_{11}, y_{21}, y_{31} \rangle \\
\langle x_{12}, x_{22} \rangle & \langle y_{12}, y_{22}, y_{32} \rangle \\
\langle x_{13}, x_{23} \rangle & \langle y_{13}, y_{23}, y_{33} \rangle \\
\langle x_{14}, x_{24} \rangle & \langle y_{14}, y_{24}, y_{34} \rangle
\end{array}
$$

Vector transformations: matrix product

$$\begin{array}{c|c}
 & \langle a_{11}, a_{21}, \boxed{a_{31}} \rangle \\
 & \langle a_{12}, a_{22}, \boxed{a_{32}} \rangle \\
\hline
\langle x_{11}, x_{21} \rangle & \langle y_{11}, y_{21}, y_{31} \rangle \\
\boxed{\langle x_{12}, x_{22} \rangle} & \langle y_{12}, y_{22}, \boxed{y_{32}} \rangle \\
\langle x_{13}, x_{23} \rangle & \langle y_{13}, y_{23}, y_{33} \rangle \\
\langle x_{14}, x_{24} \rangle & \langle y_{14}, y_{24}, y_{34} \rangle
\end{array}$$

$$y_{32} = x_{12} a_{31} + x_{22} a_{3}$$

Vector transformations: matrix product

$$
\begin{array}{c|c}
 & \langle a_{11}, a_{21}, \boxed{a_{31}} \rangle \\
 & \langle a_{12}, a_{22}, \boxed{a_{32}} \rangle \\
\hline
\langle x_{11}, x_{21} \rangle & \langle y_{11}, y_{21}, y_{31} \rangle \\
\boxed{\langle x_{12}, x_{22} \rangle} & \langle y_{12}, y_{22}, \boxed{y_{32}} \rangle \\
\langle x_{13}, x_{23} \rangle & \langle y_{13}, y_{23}, y_{33} \rangle \\
\langle x_{14}, x_{24} \rangle & \langle y_{14}, y_{24}, y_{34} \rangle
\end{array}
$$

$$y_{32} = x_{12} a_{31} + x_{22} a_{3}$$

Vector transformations

- linear transformation: multiply with a matrix
  - incl. rotation, moving, scaling, skewing, mirroring **and their combinations**

Vector transformations

- linear transformation: multiply with a matrix
  - incl. rotation, moving, scaling, skewing, mirroring **and their combinations**
- non-linear transformation: everything else
  - much more and weirder
  - combinations make it more complex and weird

Neural networks:

- represent input info with a vector
  - just a list of numbers
- apply a sequence of transformations to it
  - matrices, functions
- get the output info (‘s vector) as a result

Neural networks:

- collect a dataset of input/output vectors
- use it to learn the linear transformations **automagically**
  - vectors are just lists of numbers
  - matrices are just tables of numbers
  - let's just learn these numbers! like with soup

Neural networks:

- the sequence of transformations and vectors: computation graph

Neural networks:

- the sequence of transformations and vectors: computation graph

$$\begin{array}{l} \\ \text{word: } \end{array} \begin{array}{l} \quad\ \ \text{the} \quad \text{of} \quad \text{on} \quad \text{a} \quad \text{you} \\ \boldsymbol{x}^T = \langle 0,\ 0,\ 1,\ 0,\ 0,\ \ldots \rangle \end{array}$$

$$\begin{array}{l} \\ \text{PoS-tag: } \end{array} \begin{array}{l} \quad\ \ \text{noun} \quad \text{verb} \quad \text{adj} \quad \text{prep} \\ \boldsymbol{y}^T = \langle 0,\ \ 0,\ \ 0,\ \ 1,\ \ \ldots \rangle \end{array}$$

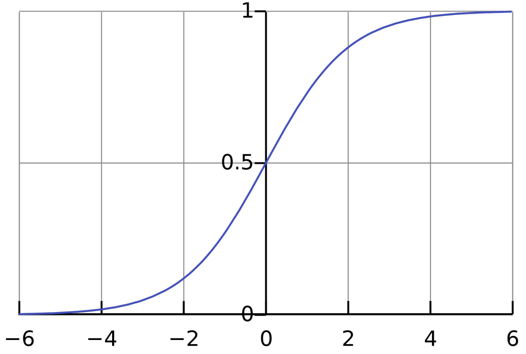
$$\boldsymbol{y} = \boldsymbol{\sigma}(\boldsymbol{\sigma}(\boldsymbol{x}\mathrm{W})\mathrm{V}) + \text{learn W and V}$$

Neural networks:

- the sequence of transformations and vectors: computation graph

$$\text{word: } \begin{array}{cccccccc} & & \text{the} & \text{of} & \text{on} & \text{a} & \text{you} & \\ \boldsymbol{x}^T & = & \langle 0, & 0, & 1, & 0, & 0, & \ldots \rangle \end{array}$$

$$\text{PoS-tag: } \begin{array}{ccccccc} & & \text{noun} & \text{verb} & \text{adj} & \text{prep} & \\ \boldsymbol{y}^T & = & \langle 0, & 0, & 0, & 1, & \ldots \rangle \end{array}$$



$$\boldsymbol{y} = \boldsymbol{\sigma}(\boldsymbol{\sigma}(\boldsymbol{x}\mathrm{W})\mathrm{V}) + \text{learn W and V}$$

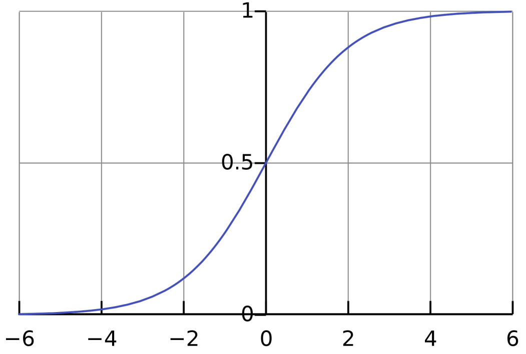
$$\sigma(t) = 1 \,/\, (1 + e^t)$$

Underfitting, anyone?

- the sequence of transformations and vectors: computation graph

word: $$\begin{array}{ccccccc} & \text{the} & \text{of} & \text{on} & \text{a} & \text{you} & \\ \boldsymbol{x}^T = \langle & 0, & 0, & 1, & 0, & 0, & \ldots \rangle \end{array}$$

PoS-tag: $$\begin{array}{cccccc} & \text{noun} & \text{verb} & \text{adj} & \text{prep} & \\ \boldsymbol{y}^T = \langle & 0, & 0, & 0, & 1, & \ldots \rangle \end{array}$$



$\boldsymbol{y} = \boldsymbol{\sigma}(\boldsymbol{\sigma}(\boldsymbol{x}\mathrm{W})\mathrm{V})$ + learn W and V

$\sigma(t) = 1 / (1 + e^t)$

Recap:

1. NN = sequence of automatically learned vector transformations
2. non-linearity matters
3. not magic
4. need computation graph

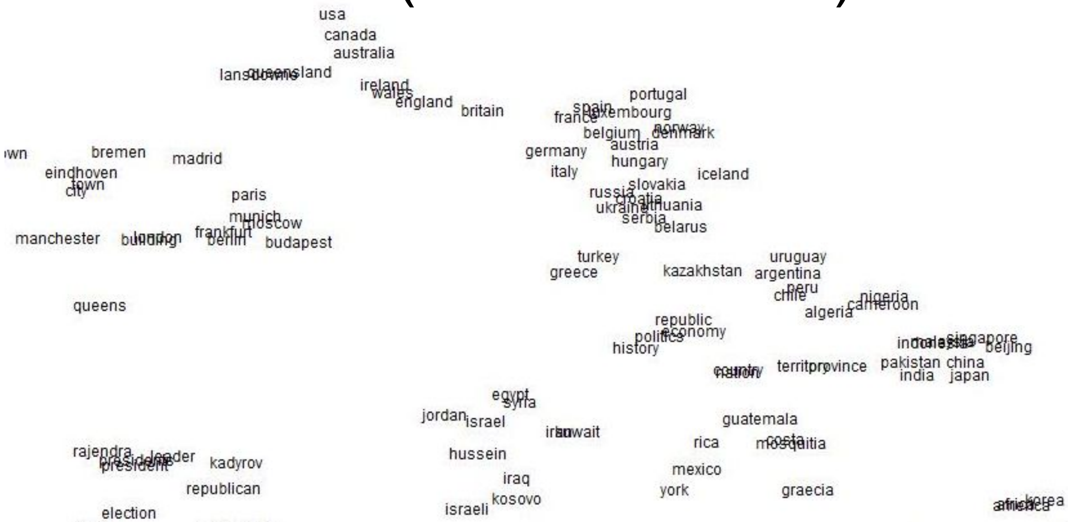
TODO: check Slido

Neural networks and words

- first transformation:
  - from sparse 0/1 vectors (size: tens of thousands)
  - into dense continuous vectors (size: hundreds)

Neural networks and words

- first transformation:
  - from sparse 0/1 vectors (size: tens of thousands)
  - into dense continuous vectors (size: hundreds)
- **embeddings**



Neural networks and words

- output trained with:
  $\langle 0,\ 0,\ 1,\ 0, \ldots \rangle$

- what is actually predicted
  $\langle 0.22,\ 0.10,\ 0.88,\ 0.27,\ \ldots \rangle$

- exponentiate, normalize (**SoftMax**):
  probability distribution!

Words vs vectors?

- number of different words in a language?

Words vs vectors?

- number of different words in a language? potentially infinite
  - or depends on training data
- vector size fixed

Words vs vectors?

Solutions:

- learn only K most frequent words
- translate character-by-character
- represent variable-size lexicon with a fixed-size encoding scheme

Words vs vectors?

Solutions:

- learn only K most frequent words
- translate character-by-character
- **represent variable-size lexicon with a fixed-size encoding scheme**

Splitting words

kassitoit	kassipiim	kassipoeg
koeratoit	koerapiim	koerapoeg
linnutoit	linnupiim	linnupoeg
kalatoit	kalapiim	kalapoeg

size of vocabulary?

Splitting words

kassi_ _toit	kassi_ _piim	kassi_ _poeg
koera_ _toit	koera_ _piim	koera_ _poeg
linnu_ _toit	linnu_ _piim	linnu_ _poeg
kala_ _toit	kala_ _piim	kala_ _poeg

size of vocabulary?

Byte-pair encoding

- split text into characters
  - vocabulary size = K (num. of characters)
- find most frequent adjacent pair
  - join it together
  - add to vocabulary
  - vocabulary size = K + 1
- repeat N times
  - vocabulary size = K + N

Recap:

1. first thing NNs learn:
   optimal representation (**embeddings**)
2. last thing NNs learn:
   probability estimation (**SoftMax**)
3. words can be butchered into a
   fixed-size representation (e.g. **BPE**)

TODO: check Slido

Neural Machine Translation

What we learned:

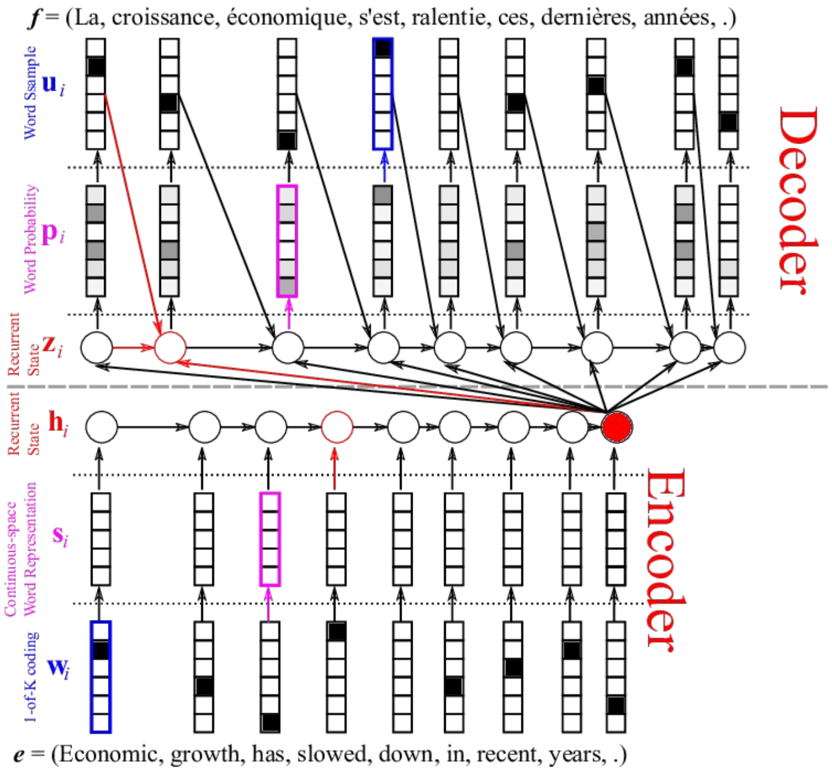
- Represent i/o words with vectors
- Come up with a crazy computation graph
- Learn the transformations
- Translate one (sub)word at a time
  - “autoregressive”

NMT: encoder-decoder

Encoding:
input sentence →
embeddings →
RNN encoding, context vectors →
input sentence vector

Decoding:
(input sentence vector, ⟨s⟩) → first word
(input sentence vector, first word) → second word
(input sentence vector, first two words) → third word …

NMT: encoder-decoder



NMT: enc-dec + attention

Encoding:
input sentence → embeddings → context vectors

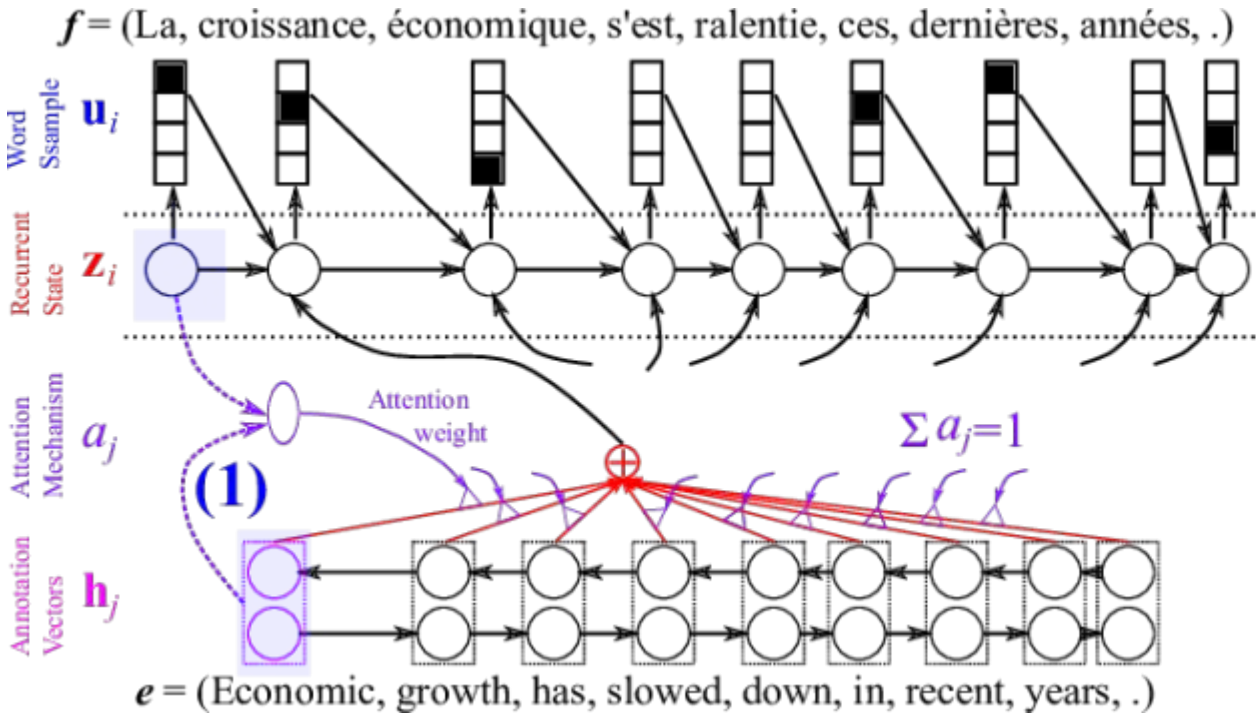
Decoding:
(ctx vecs + weights) → first word
(ctx vecs + new weights, first word) → second word
(ctx vecs + new weights, first two words) → third word
…

NMT: enc-dec + attention

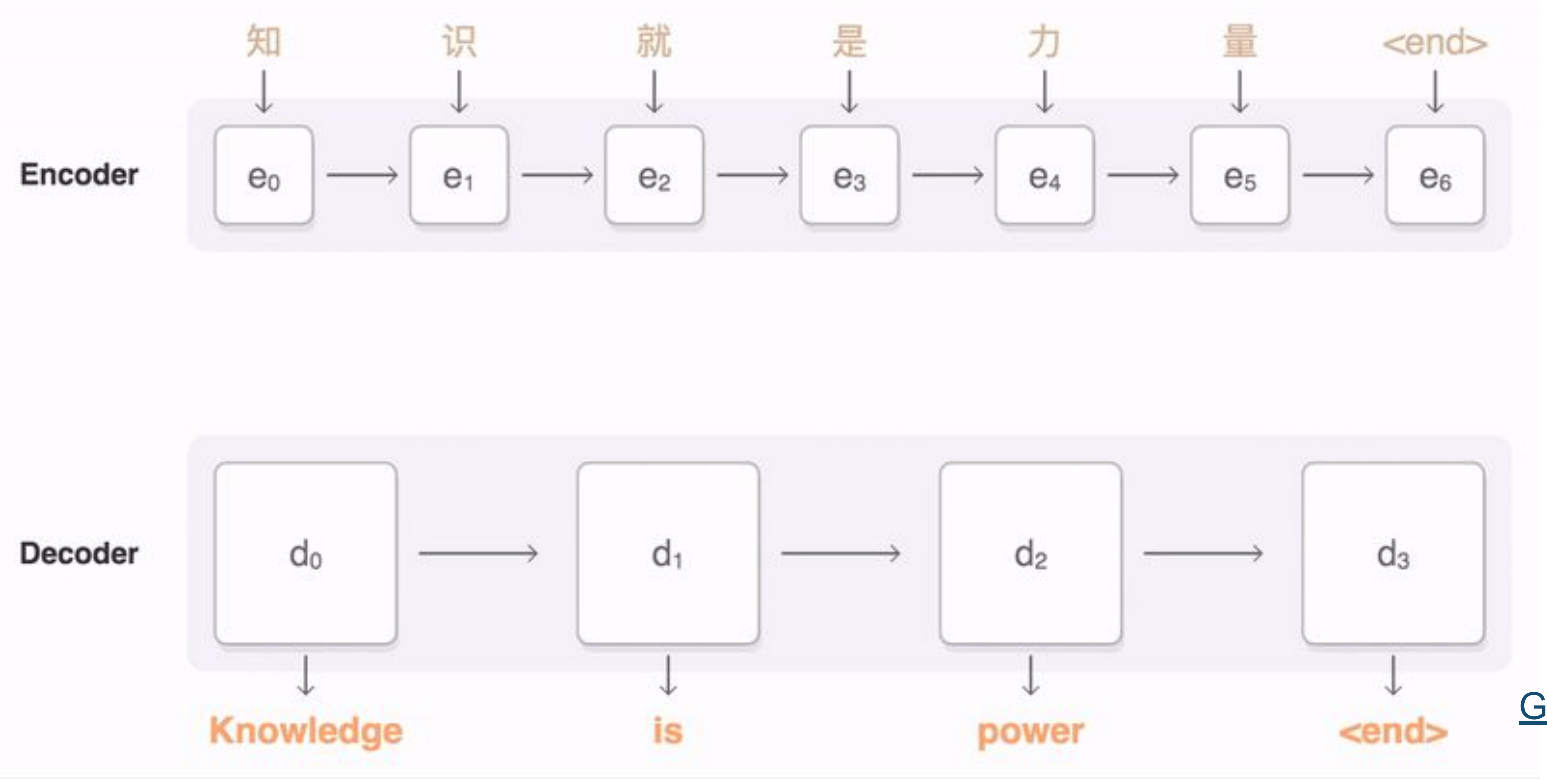
Encoding:
input sentence → embeddings → context embeddings

Decoding:
(ctx vecs + weights) → first word
(ctx vecs + new weights, first word) → second word
(ctx vecs + new weights, first two words) → third word
…

NMT: enc-dec + attention

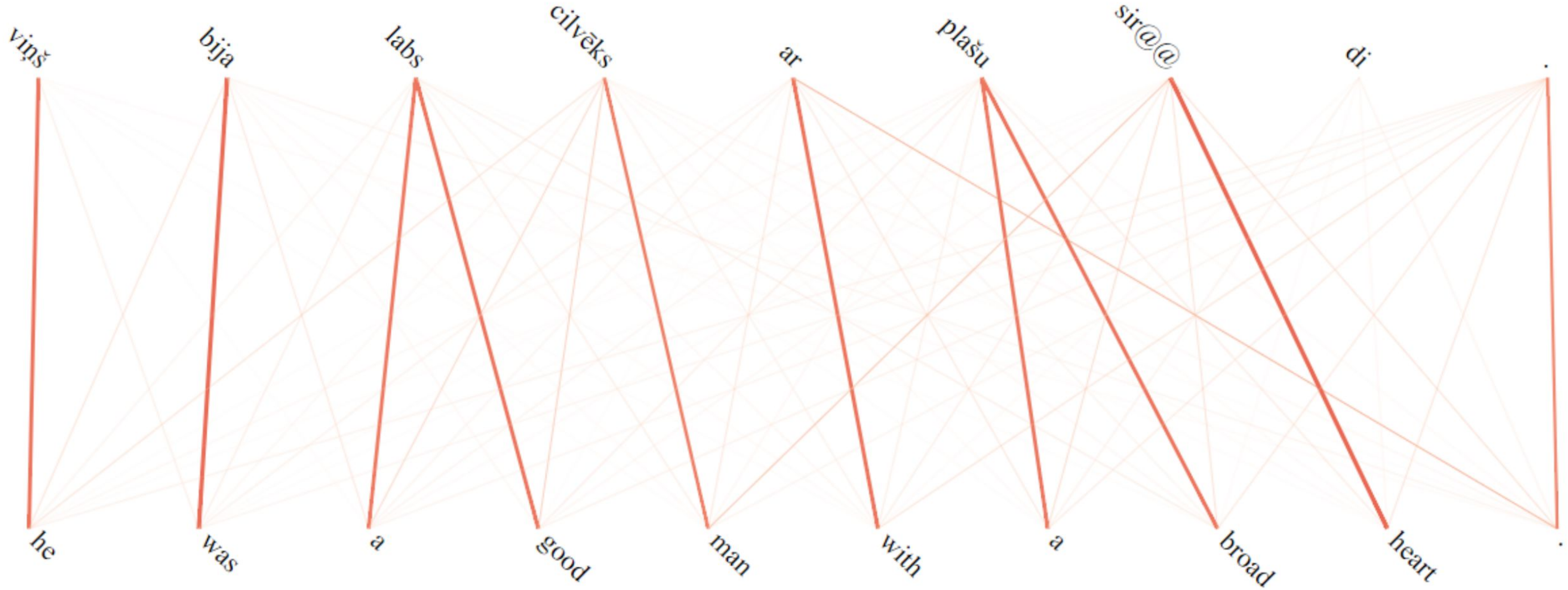


NMT: enc-dec + attention

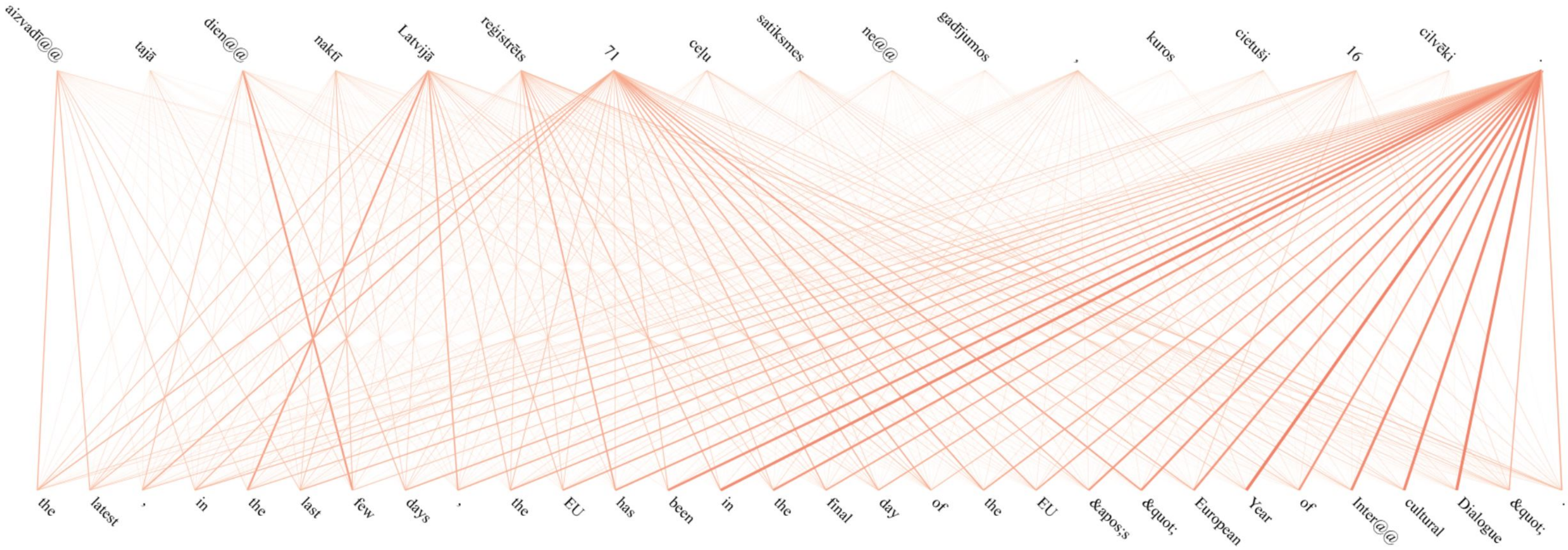


Src: [Google AI Blog]

NMT: enc-dec + attention



NMT: enc-dec + attention



NMT: attention is all you need!

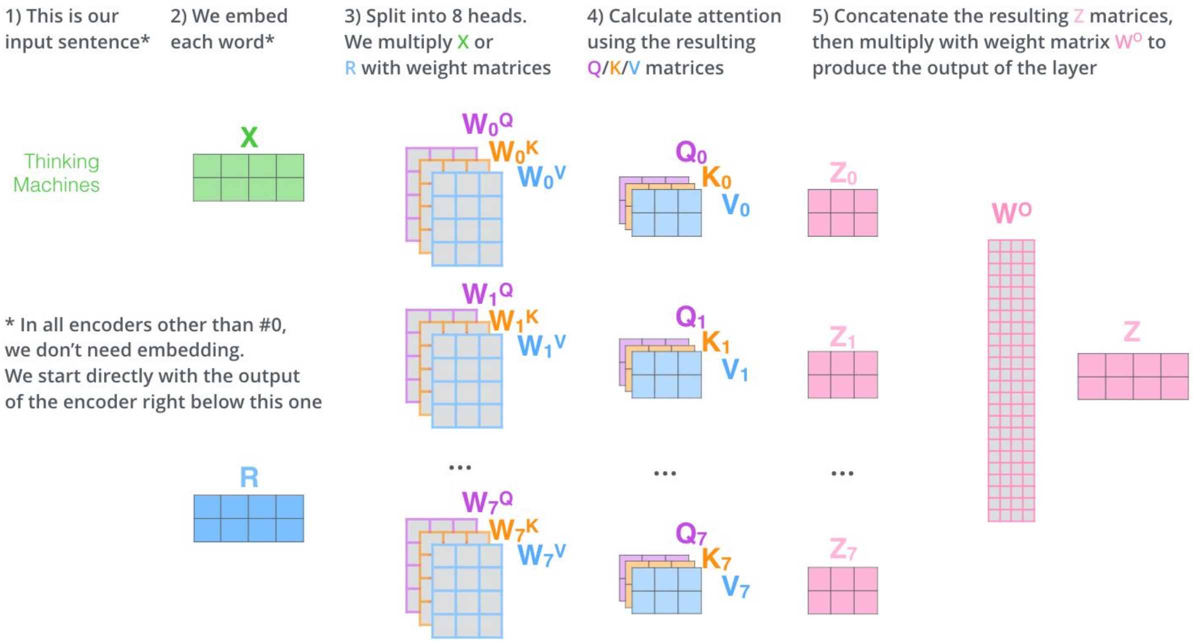
Encoding:
input sentence → embeddings → attention over itself (repeat) → context vectors

Decoding:
attentioned ctx vecs, attention over its own prev. output → (repeat) → next word

NMT: attention is all you need!

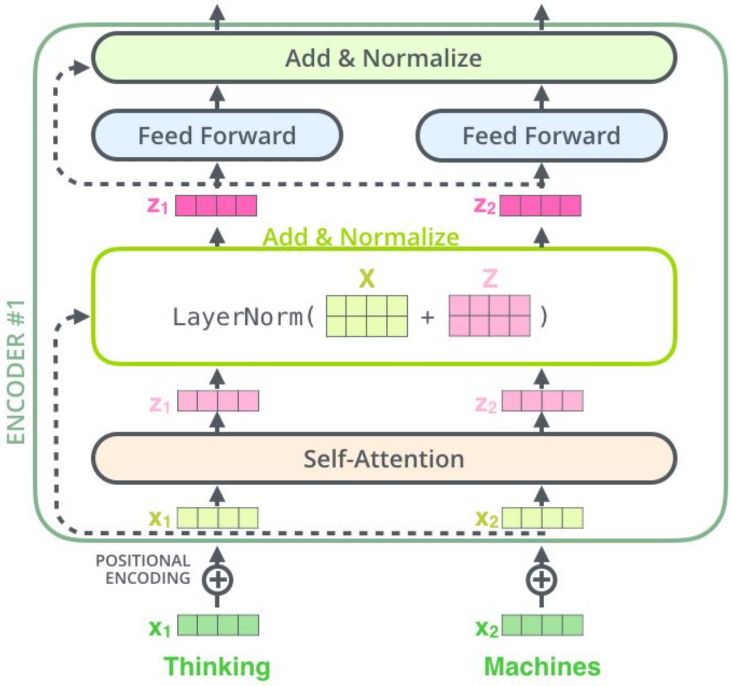
Src: Google AI Blog

Transformer: self-attention with keys, values and queries



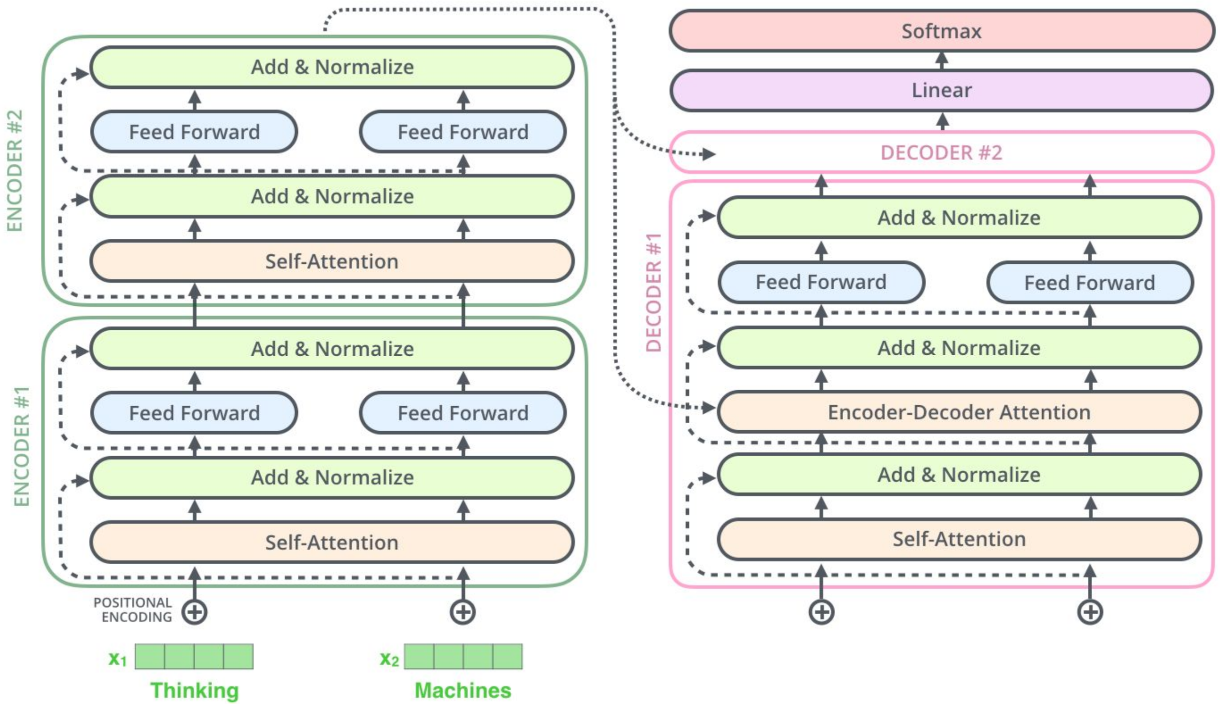
Src: [Illustrated Transformer]

Transformer: one encoder



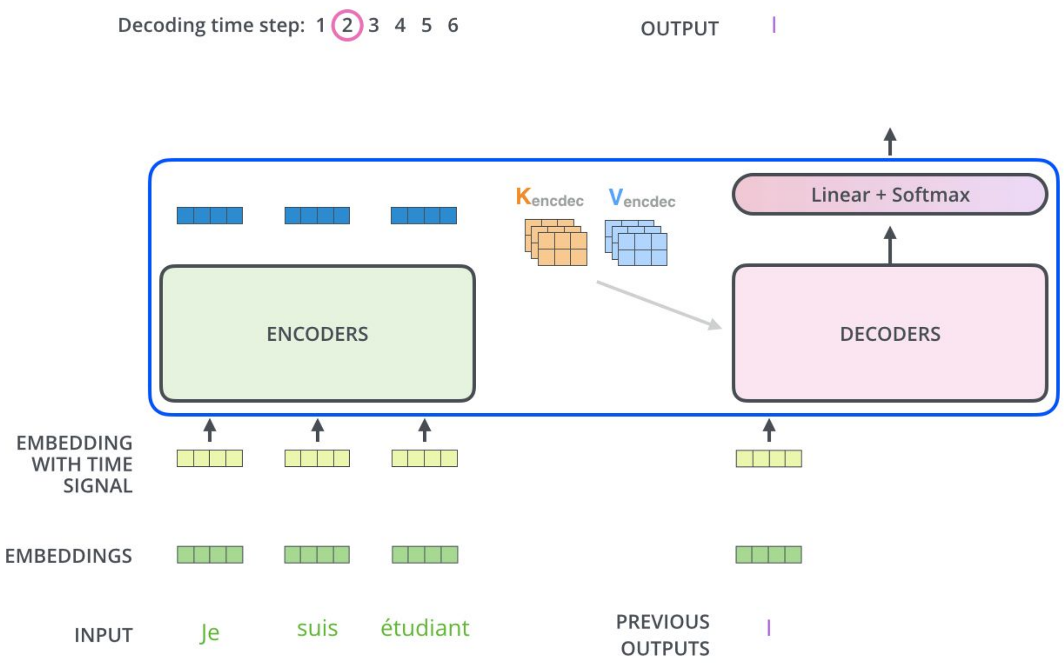
Src: [Illustrated Transformer]()

Transformer: encoder+decoder



Src: [Illustrated Transformer]

Transformer



Src: [Illustrated Transformer]

Magic?

- NNs sensitive to meta-parameters
  - learning rate, etc.
  - these cannot be automatically tuned (reasonably)
- computation graph encodes our bias
  - narrowing down the model to a particular task
- no magic

Magic:

- wget http://statmt.org/europarl/v7/cs-en.tgz
- gunzip, paste+shuf+cut+head into train/dev/test
- pip install sockeye sentencepiece
- sentencepiece the data into subwords
- sockeye-train the NMT model
- sockeye-translate your texts!

(need a GPU and several days to wait)


"That's all Folks!"