

# TRANS·FOR·MERS FOR HY·PHEN·A·TION

A review of Seq2seq transformers in the context of Dutch Hyphenation, and other insights

# Short biography

- **François REMY // @FremyCompany**
  - Speech Recognition Team  
Internet and Data Science Lab  
University of Ghent
  - Interested in structured data extraction  
and question answering



# Motivation for this work

- **Language Model trained on a large text corpus**
  - Many words of unknown pronunciation
  - Guessing their pronunciation is known as grapheme-to-phoneme
  - In case of Dutch, transfer learning from hyphenation sounds plausible

Netwerkswitches  
≠ Networks + witches

# Motivation for this work

- **Language Model trained on a large text corpus**
  - Many words, more than is reasonable to have in a dictionary
  - Have to split them in smaller chunks called token
  - Techniques in use today don't care about splitting a syllable in two

Throat  $\neq$  Thro + at

# Byte Pair Encoding

- Start with words as a sequence of letter
- Merge the 2 letters that are the most often found near each other
- Repeat until you have merged  $N$  times,  $N \sim$  your vocab size

# Motivation for this work

- **Transformers-based models are popular**
  - But very few people seem to train them from scratch
  - Hence it's difficult to have a good intuition on how they're working
  - Learning experience

# Existing hyphenation tools

## ○ TeX and OpenOffice: Liang's TeX82

- Find patterns leading to hyphenation in stages
  - What character-patterns usually lead to hyphens?
  - What character-patterns usually prevent hyphens?
  - Repeat 6 times with patterns of increasing size, and less tolerance for imprecisions
- Works best for languages that don't form compounds, like French or English
- Good precision (>95%) and usually-acceptable recall (>90%)

```
.in-d .in-s .in-t .un-d b-s -cia con-s  
ex- -ful it-t i-ty -less l-ly -ment n-c  
n-v om-m -sion s-ly s-nes ti-ca x-p
```

# Side note on Morphessor

## ○ **Morphessor**

- Python program using Viterbi and Recursive Search to find split trees for strings
  - But is based on a “semantic” search rather than a pronunciation approach
  - Doesn't model letter sequence as well, not suitable for the task at hand
    - Ex: “Vriend-en” vs “Vrien-den”



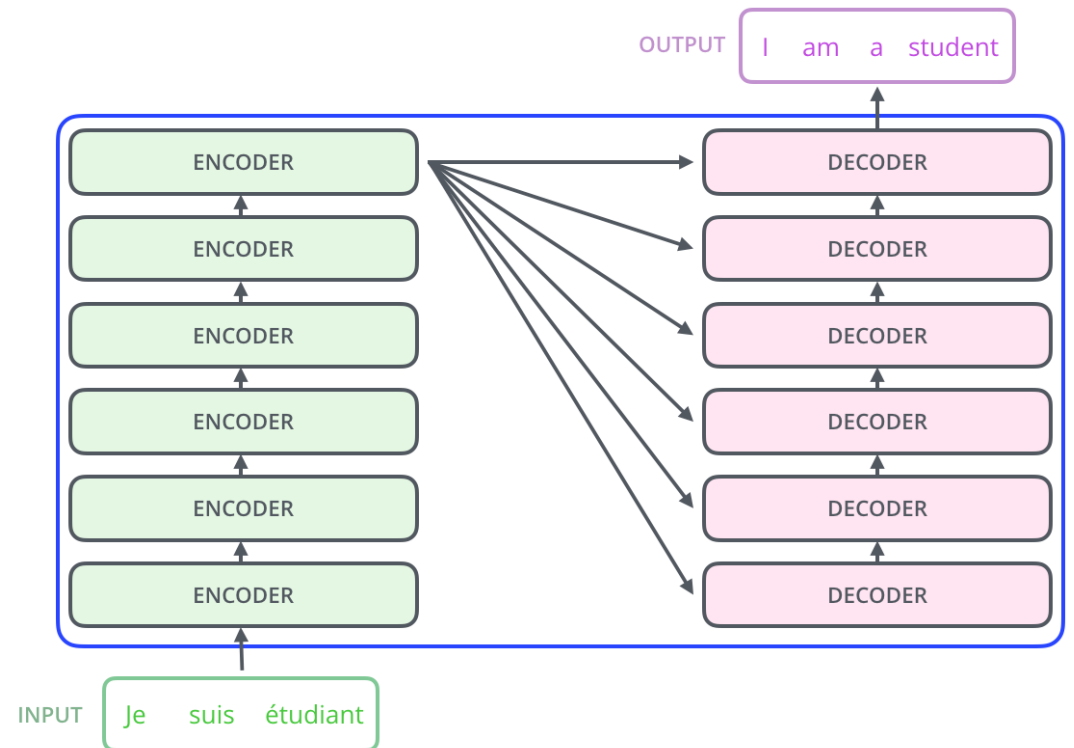
# Introduction of Transformers models

## ○ An encoder stack

- Maps each input in a sequence into an embedding representation

## ○ A decoder stack

- Generates a sequence output by output by combining the previously-generated embeddings
- **(Can perform Beam search)**



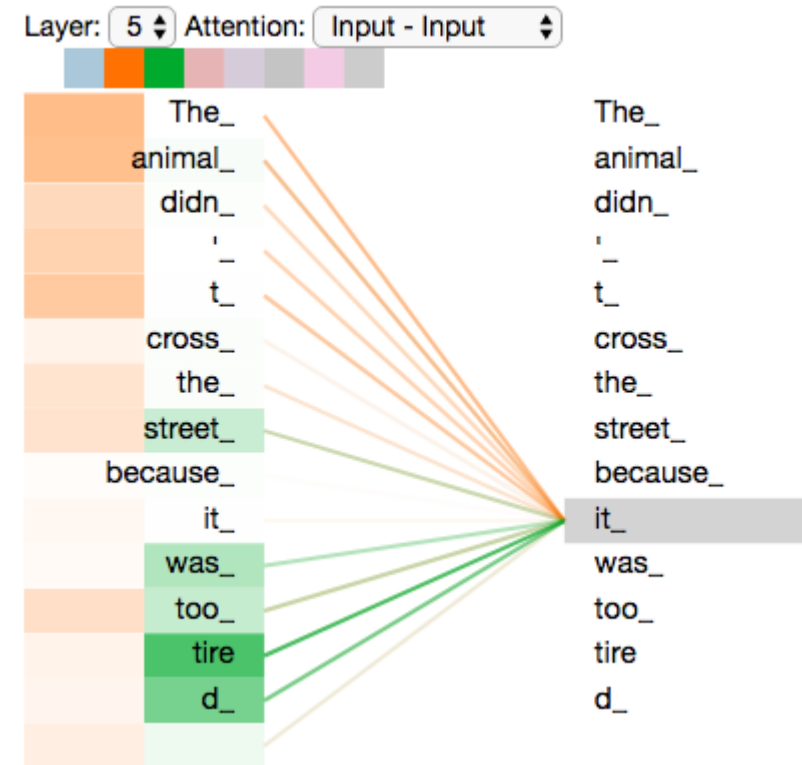
# Introduction of Transformers models

## ○ Each input unit

- attends to every other input, creates a complex representation for itself

## ○ Each output unit

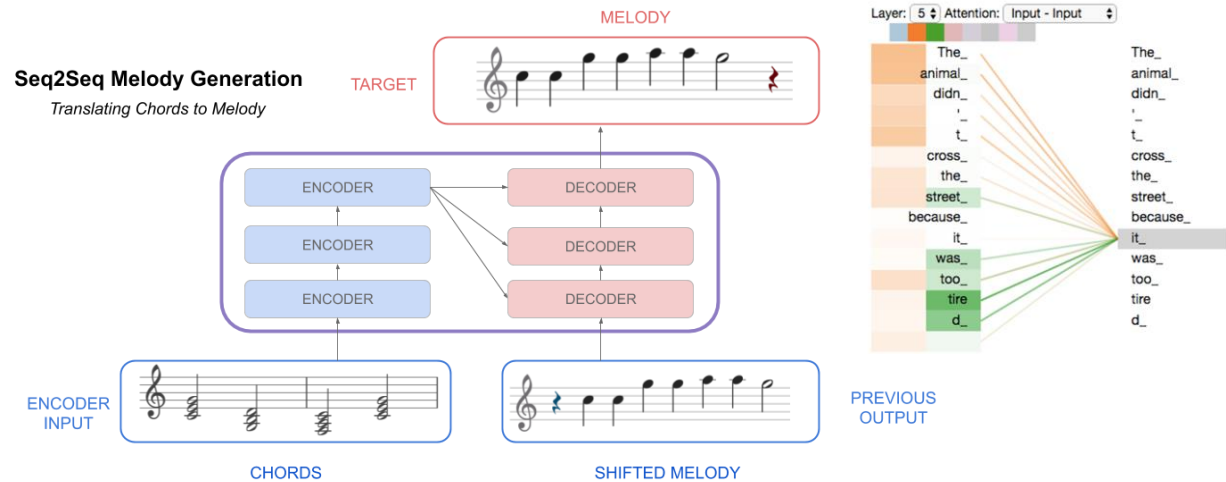
- attends to every input and every previously-generated output



# Introduction of Transformers models

## ○ State of the art performance

- In translation
- In language modelling
- In differential equation solving!



Equation	Solution
$y' = \frac{16x^3 - 42x^2 + 2x}{(-16x^8 + 112x^7 - 204x^6 + 28x^5 - x^4 + 1)^{1/2}}$	$y = \sin^{-1}(4x^4 - 14x^3 + x^2)$
$3xy \cos(x) - \sqrt{9x^2 \sin(x)^2 + 1}y' + 3y \sin(x) = 0$	$y = c \exp(\sinh^{-1}(3x \sin(x)))$
$4x^4yy'' - 8x^4y'^2 - 8x^3yy' - 3x^3y'' - 8x^2y'^2 - 6x^2y'y'' - 9xy' - 3y = 0$	$y = \frac{c_1 + 3x + 3 \log(x)}{x(c_2 + 4x)}$

Table 5: Examples of problems that our model is able to solve, on which Mathematica and Matlab were not able to find a solution. For each equation, our model finds a valid solution with greedy decoding.

# G2P Seq2Seq: Letters as input

## Sequence-to-Sequence G2P toolkit

---

The tool does Grapheme-to-Phoneme (G2P) conversion using transformer model from tensor2tensor toolkit [1]. A lot of approaches in sequence modeling and transduction problems use recurrent neural networks. But, transformer model architecture eschews recurrence and instead relies entirely on an attention mechanism to draw global dependencies between input and output [2].

**WER: 20% Relative Improvement vs Baseline**

<https://github.com/cmuspinx/g2p-seq2seq>


# Dataset extracted from WikiWoordenboek

- **Around 350K words with audio**  
(good quality)
- **Around 40k words with IPA representation**  
(but of lesser quality)
- **Around 400k words with hyphenation**  
(very good quality!)

## *Nederlands*

---

### *Uitspraak*

- Geluid:  bestaan
- IPA: /bə'stan/

### *Woordafbreking*

- be·staan

# TRAINING INSIGHTS

Get the most out of your transformer model training time

# Positional encoding isn't ideal

- **Transformers don't rely on an order between tokens, the tokens embed their position in themselves**
  - This can cause the decoder to “skip” or “rewind back” accidentally
    - Especially at lengths which haven't been seen much during the training, and longer lengths in general

Hottententententoonstelling  
> Hot-ten-ten-ten-ten-ten-tens <

# Dealing with positional encoding

- **If possible, keep input and output length identical**
  - This will help the decoder be stricter on the positional encoding match

be-vrien-den

-vs-

BeVrienDen



# Transformers are data-hungry

- **Needs to be bucketized by word length to avoid loops or omissions near the end of long words, but this splits the dataset even further**
  - Usage of data augmentation to produce new possible long words, in order to increase the dataset set (up to 8 million words)
    - Upside: much better results on test set thanks to more data points
    - Downside: learns bias in data augmentation that are detrimental: we have now moved the problem to good and well-proportioned data augmentations

When do you insert a tussen-s?

# Dutch compounds: tussen-s

De regel voor de tussen-s is: als een woord het best klinkt met een tussen-s, dan mag je de tussen-s ook opschrijven. In de meeste gevallen kun je dus op je gehoor afgaan. Bij een woord als *kwaliteit(s)standaard* is niet te horen of er één of twee s'en geschreven moeten worden. Dat komt doordat *standaard* ook al met een s begint. Om te bepalen wat de beste vorm is, kun je twee dingen doen:

1. Maak een samentrekking. Als je twijfelt over *kwaliteit(s)standaard*, kun je nagaan welke samentrekking voor jou het best klinkt: *kwaliteits- en levensstandaard* of *kwaliteit- en levensstandaard*. Als het eerste het best klinkt, pleit dat voor *kwaliteitsstandaard*.
2. Vervang het tweede deel door een woord dat niet met een s-klank begint. In *kwaliteitsbewaking* is een tussen-s duidelijk het gangbaarst. Dan kun je ook kiezen voor *kwaliteitsstandaard*.

Er zijn dus geen officiële goed-foutregels voor het wel of niet invoegen van de tussen-s. Het is dan ook geen groot spellingprobleem, omdat bij de meeste woorden duidelijk is welke vorm het gebruikelijkst is. En als dat niet zo is, zijn die twee vormen voor het gevoel van de meeste mensen ook echt even acceptabel.

## Vaak geen tussen-s

De meeste mensen gebruiken geen tussen-s:

- als het eerste deel een zelfstandig naamwoord is dat niet verwijst naar een levend wezen en (ook) een meervoud heeft op -s: *motorboot, televisieprogramma, theatergezelschap*;
- als het eerste deel een zelfstandig naamwoord is dat eindigt op -ier en geen persoon aanduidt: *alvleesklierontsteking, spierkracht*;
- als het eerste deel een werkwoordstam is: *babbelkous, loopafstand, rekenhulp* (uitzonderingen: *leidsman, scheidsrechter*);
- als het woord eindigt op -schap (in de betekenis 'hoedanigheid, het zijn van'): *blijdschap, ondernemerschap, slachtofferschap*;
- als het eerste woorddeel een stof aanduidt en niet telbaar is: *aluminiumfolie, bierglas, koffieboon* (maar wel meestal *houtskool, bloedsomloop*).

# Transformer decoders work start-to-end

- **In some context, you might want to reverse the order in which you want the transformer to guess your output sequence.**
  - For instance, in the case of word hyphenation, it might be better to reverse the sequence and feed the transformer the letters starting from the last one, instead of starting from the first one.

be-vrien-den

-vs-

ned-neirv-eb

# Transformer models are slow

- **For hyphenation, using a complex transformer model is overkill**
  - Nowadays, it's better to ship with a large hyphenation dictionary, and treat unseen words using a simpler Tex82 or decision tree, rather than use a complex transformer model
  - Transformer models can however be used to generate data to train the trees.

# DATA AUGMENTATION FOR HYPHENATION

A quick review of other techniques used to expand the transformer training set

# Semi-supervised data augmentation

- **To improve the quality of the model, we used a dictionary of words to be hyphenated, and tried to detect compounds in them for which both sides already had a known hyphenation**
  - To ensure that the splits obtained this way are not accidental, we used words obtained from a large news corpus, with at least 33 occurrences in the corpus;
  - Two types of word2vec embeddings have been computed on the corpus (small window to capture syntactic properties & large window to capture semantic properties) and a compound split was only validated if there was evidence the split matched reasonably well on both sides.

# Using embeddings to validate word splits

- **Verkeersprobleem : verkeer[s]-probleem ?**
  - $Sem\_Sim \leftarrow SemanticEmbeddings.Similarity( Verkeer, Verkeersprobleem )$
  - $Syn\_Sim \leftarrow SyntacticEmbeddings.Similarity( Verkeersprobleem, Probleem )$
  - $GetSplitScore(Sem\_Sim, Syn\_Sim) > Threshold$

Can be used to decide at which point embeddings stop to be of high quality

# DISTILLATION OF TRANSFORMER INTO RULE-BASED MODEL

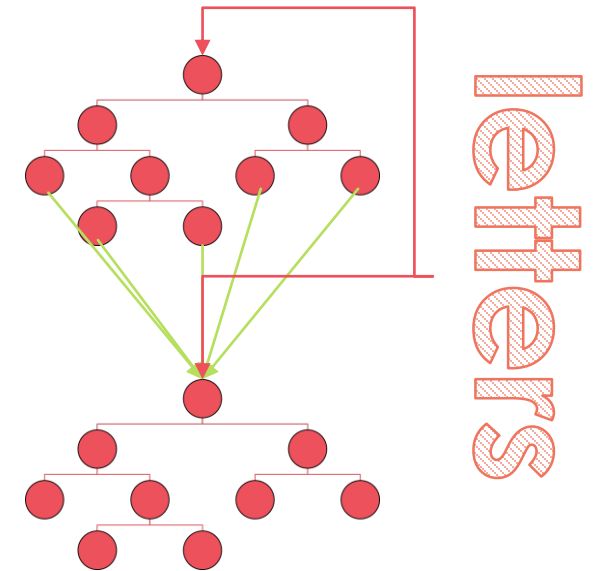
As mentioned before, transformer models are too slow in practice, but they can be distilled



# Distillation: Recursive Decision Trees

## ○ Decision tree input:

- Letters around the considered hyphenation boundary
  - All letters before and after
  - Only close letters
  - Causal
- Embedding of the word to hyphenate (if known)
- How long since the last hyphen was inserted
  - Makes the decision trees recursive like an LSTM
  - Generate a forest by training on cross-validation folds



# Using embeddings in Decision Trees

- **Embeddings are not very useful for decision trees that can look backward and forward far enough**
  - They are however very useful for “causal” models, which only know which letters have been processed so far (but cannot look at letters at the future).
  - Combining causal models with embeddings and non-causal models improves the chances of finding hyphenations at word boundaries, thanks to the embedding “understanding” of the word at hand.

# CONCLUSION

Ending remarks

[twitter.com/FremyCompany](https://twitter.com/FremyCompany)