Applied Deep Learning for NLP

Week 7 - Transformers

Juan Carlos Medina Serrano

Technische Universität München
Hochschule für Politik
Political Data Science

Munich, 3. December 2020

Word Embeddings Shortcomings

With the word embeddings we had so far, each word had a **fixed** representation.

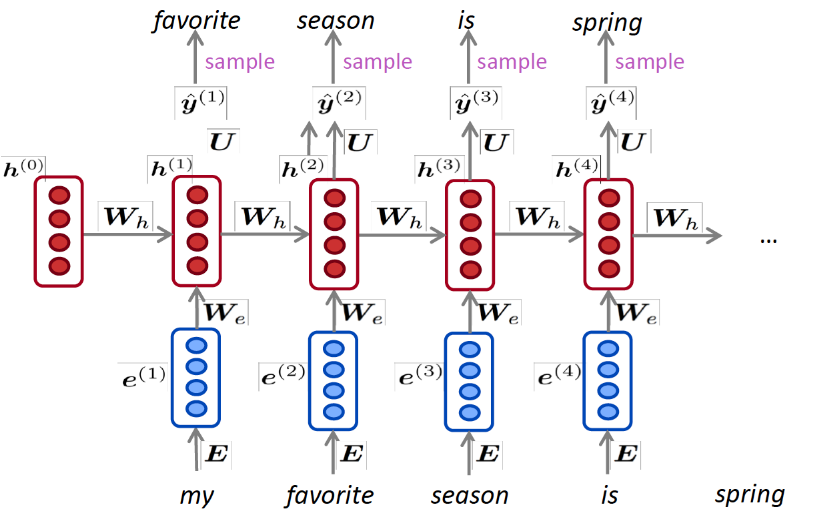
We don't take into account the context and words with different meanings (bank can be a bench and a money institution)

How do we obtain **contextual word embeddings**?

Language Models to the Rescue

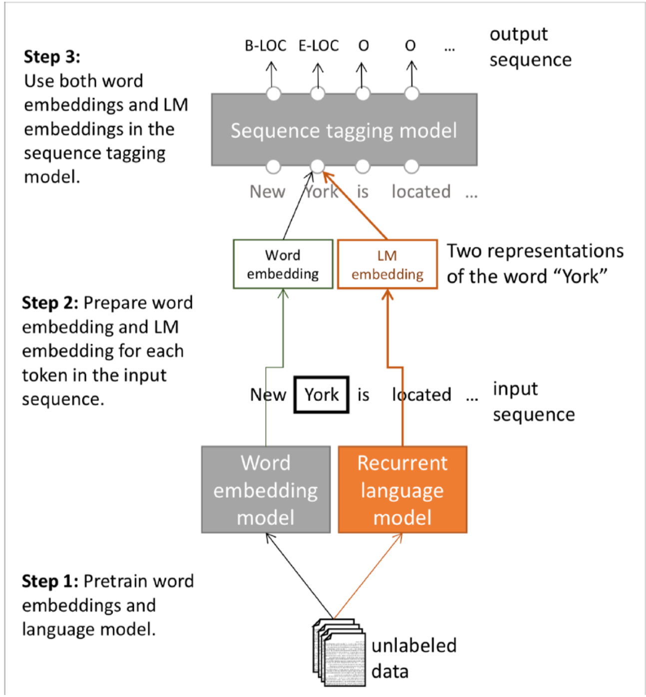
Did we all along have a solution to this shortcoming?

Language Models produce context-specific word representations at each position (The hidden state of an RNN at each time step)



Language Models to the Rescue

SOLUTION: Use a semi-supervised approach where we train a Language Model on large unlabeled corpus. Then reuse the hidden states as embeddings in another task:



ELMo: Embeddings from Language Models

1) Trains a two layer bi-directional LSTM on a large corpus. 2) Freezes the embeddings 3) Concatenates them to a task-specific model (For example, a LSTM for classification)

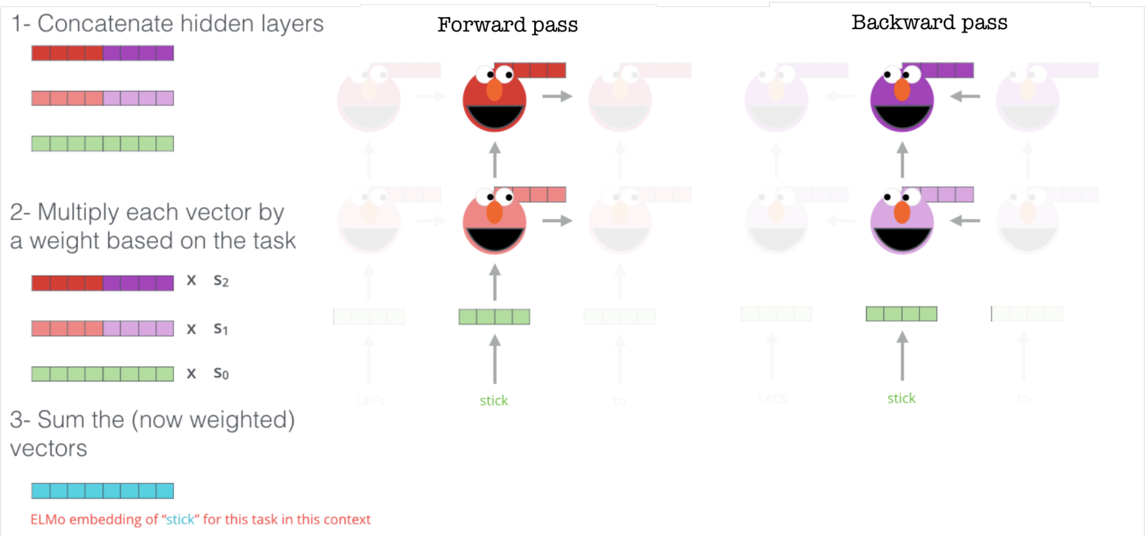
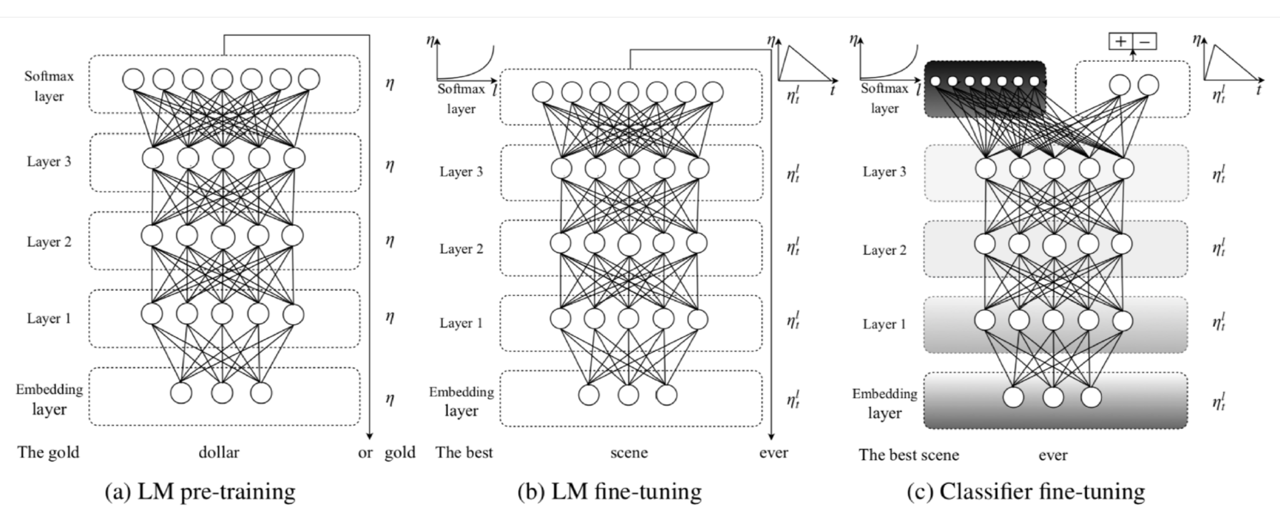


Image Source: `http://jalammar.github.io/illustrated-bert/`

ULMfit

Before we have fine-tuned pre-trained word embeddings for a specific task (for example, classification). What if we fine-tune the a complete pre-trained language model?

ULMfit (Universal Language Model Fine tuning) introduced the idea of **fine-tuning** to NLP: retrain the last few layers of a neural language model with your own data.



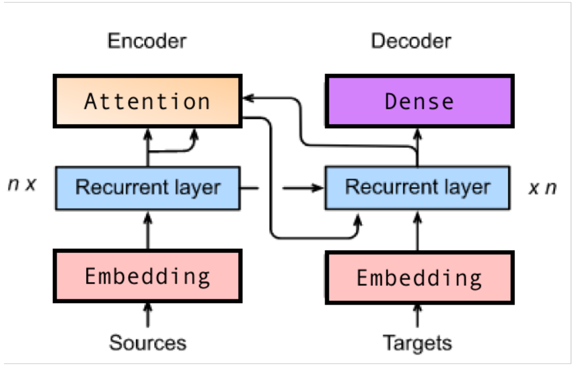
Motivation for Transformers

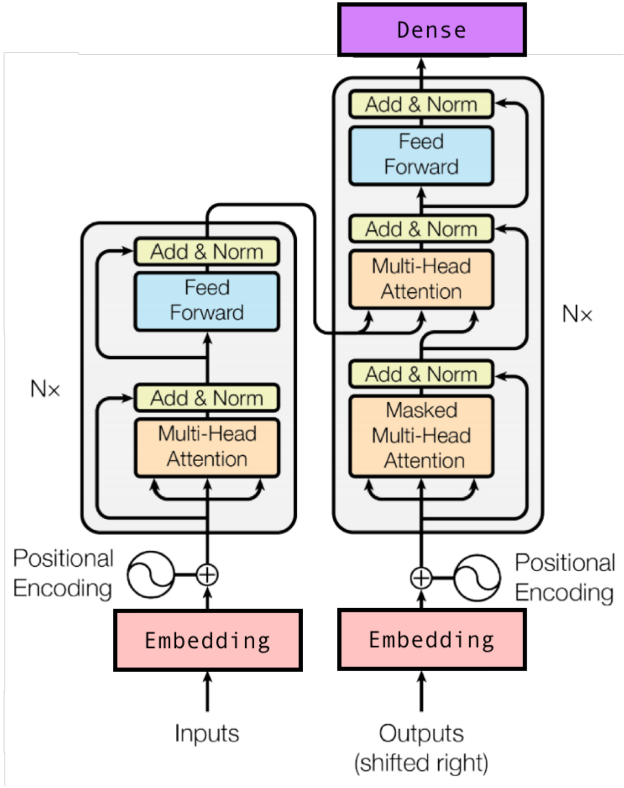
1) RNNs and co. are sequential => No way to use parallelization

2) Seq2Seq models with RNNs need **attention** to deal with long range dependencies

Let's TRANSFORM the Seq2Seq model for machine translation and don't use RNNs anymore!

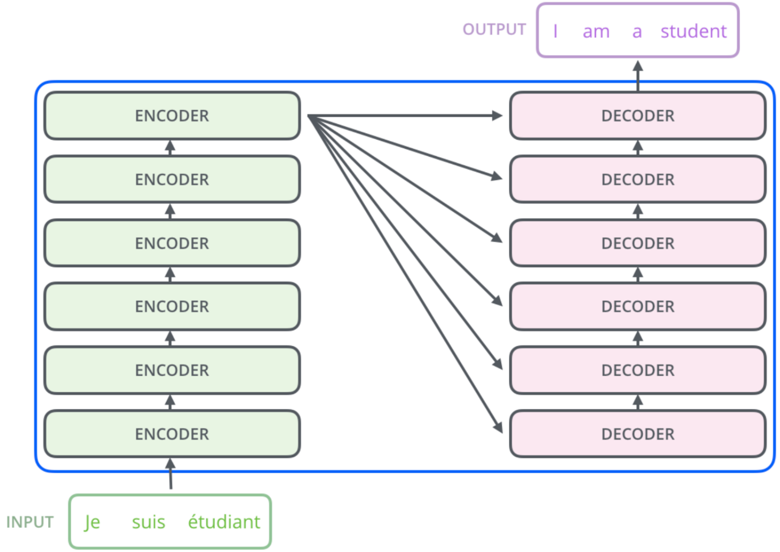
THE Transformer!





A High-Level View

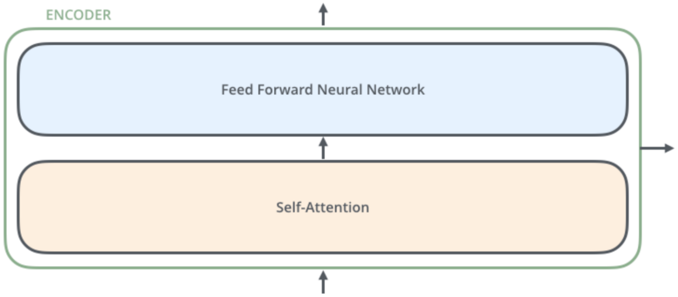
It all starts with stacks of Transformer encoders and Transformer decoders



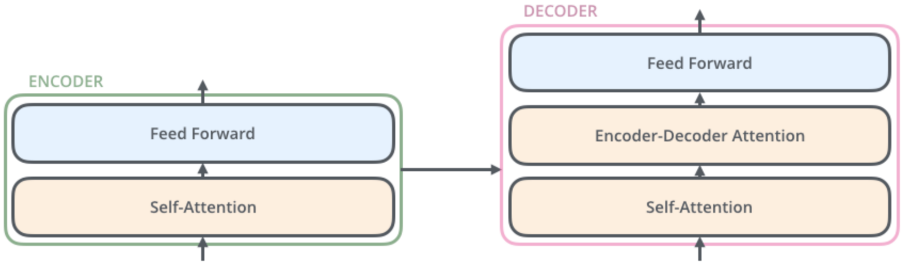
The following slides contain images from this AMAZING Transformer tutorial:
`http://jalammar.github.io/illustrated-transformer/`

A High-Level View

A Transformer Encoder has a **self-Attention** and a **pointwise** FNN layer.

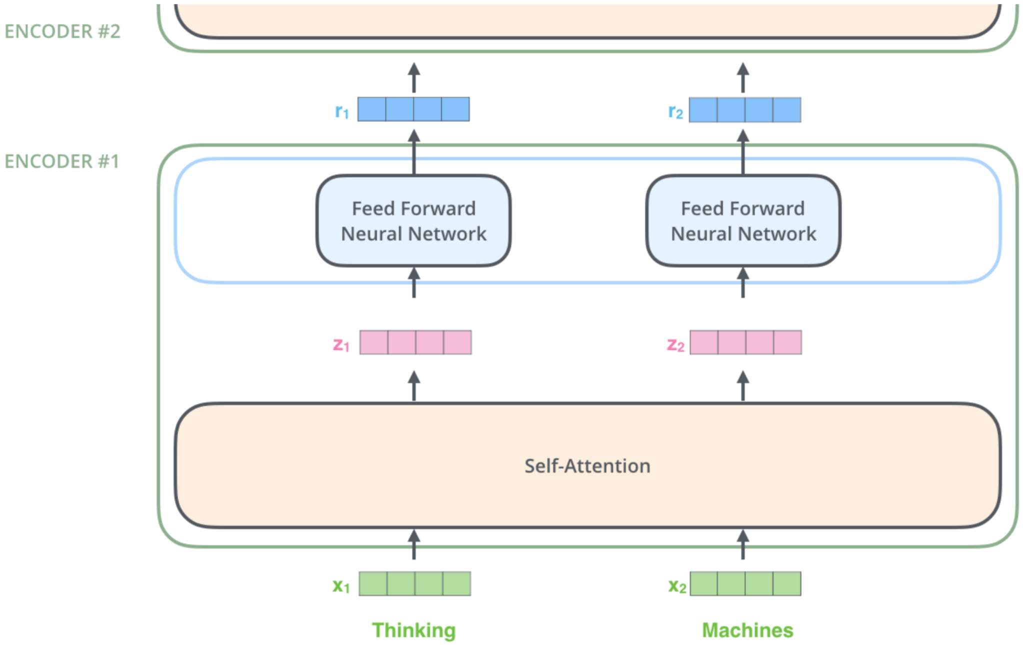


A Transformer Decoder has additional attention layer with the same purpose as the Seq2Seq attention.



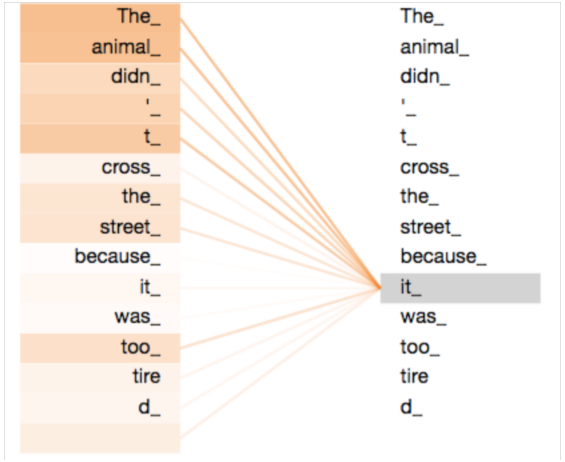
Pointwise FNN layer

Pointwise FNN layer means that every word embedding goes separately through a dense layer with ReLU (with a higher dimension than the input) and a dense layer (with the same dimension as the input) (Why do we do this? Wait for a couple of weeks...)



Self-Attention

It allows to look at other positions in the input sequence for clues that can help lead to a better encoding for this word. Similar to generating contextual embeddings



Self-Attention

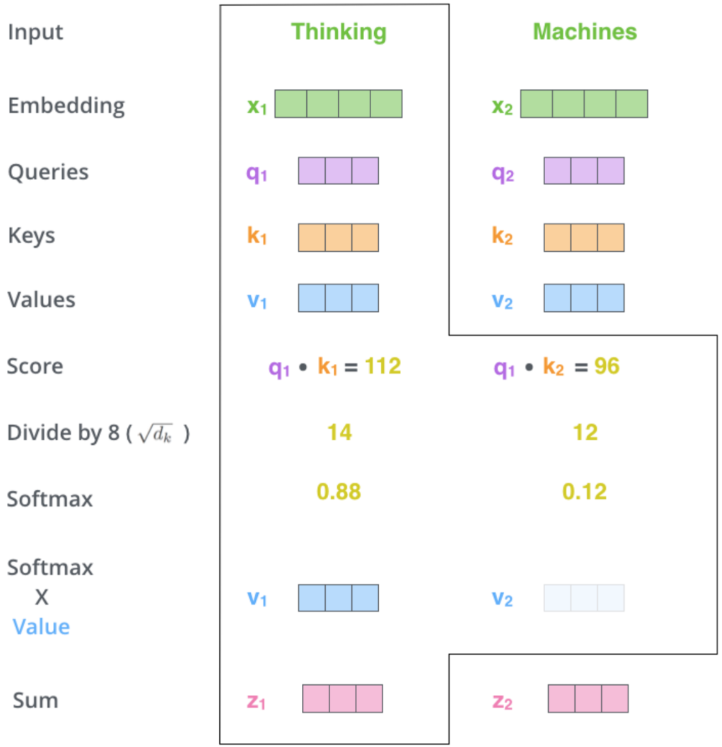
Three important vectors:

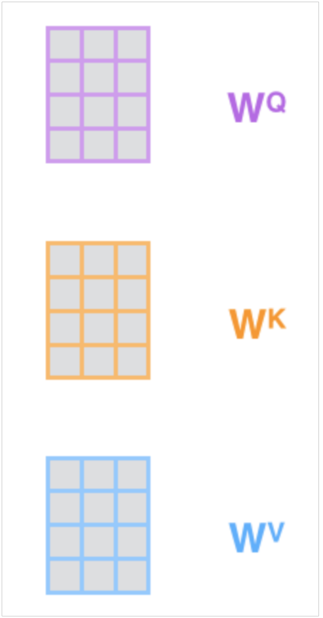
- **Query** Equivalent to the Decoder hidden state (times a W matrix)
- **Value** Equivalent to the Encoder hidden state (times a W matrix)

  Normal attention would compute a score with the dot product between Query and Value. However, we introduce a third vector:

- **Key** Think of a key-value dictionary: *{ 'subject':'They', 'verb':'played'}*.
  The key tries to identify the semantics of a language, the query represents the same in the other language.
  Dot products between keys and queries will find similarity measures of semantics in sentences.

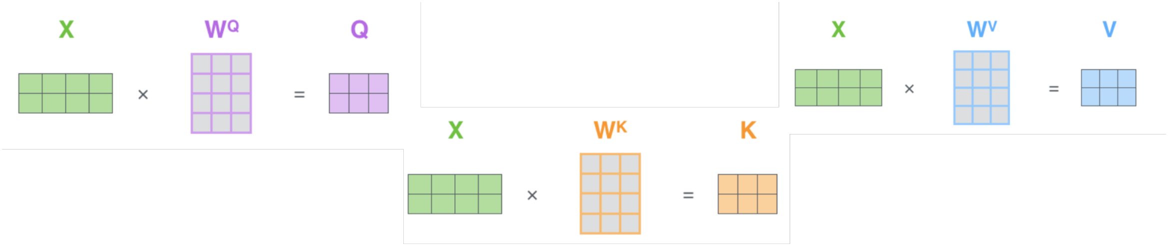
Self-Attention

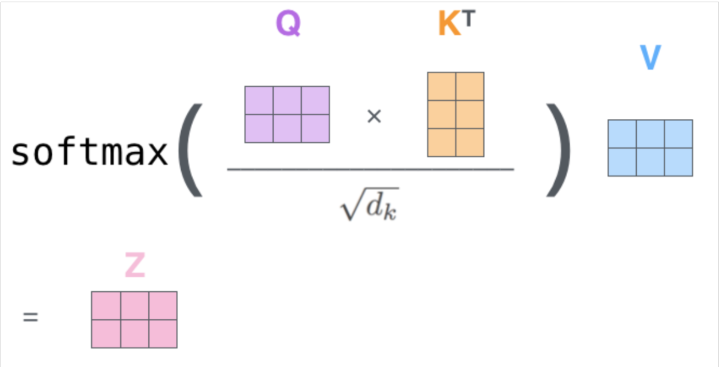




Self-Attention

Input sentence is processed in parallel with matrix multiplications:

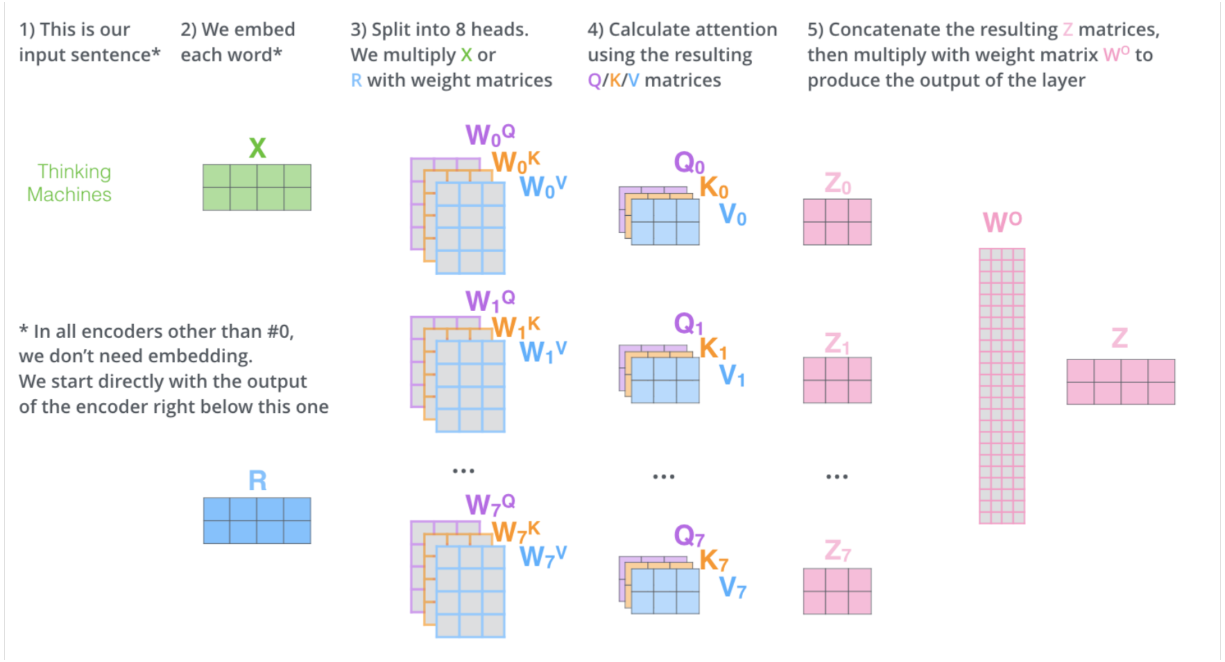




Important: W matrices are linear transformations = Dense layer without activation function

Multi-Head Attention

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions: **Idea: Multiple Self-Attentions in parallel at different positions of the embedding!**



Important: The dimension of Q_0, Q_1 ... is the embedding dimension of the word divided by the number of heads

Masked Multi-Head Attention

The decoder has first a **masked** multi-head attention.
It works the same BUT each word is only allowed to attend to words located **before** it (setting future positions to -inf)

This is important given that at translation time, we don't have the true translation!

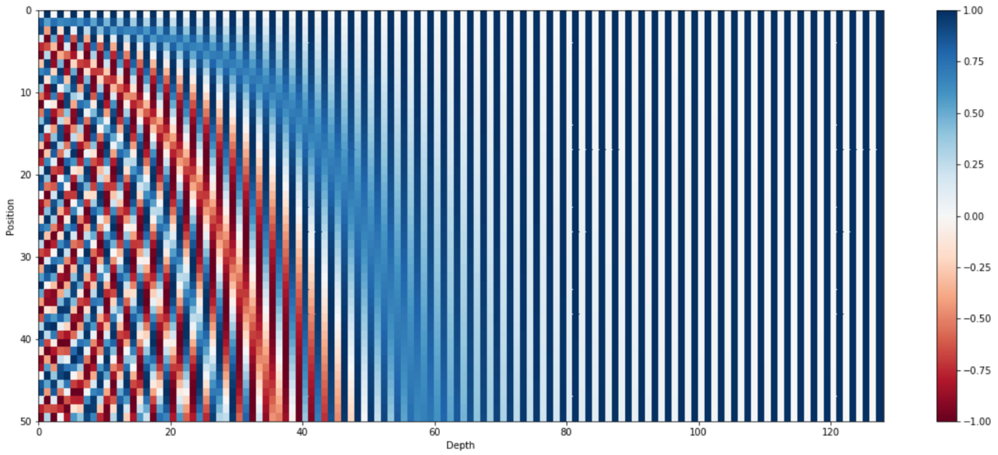
Positional Encodings

Since we don't have RNNs anymore, there is no way for the Transformer to know the position of the words in a sentence. How do we give this information to the Transformer?

- Idea 1: Assign a number to each time-step within the [0, 1] range in which 0 means the first word and 1 is the last time-step
  Problem: Time-step delta doesn't have consistent meaning across different sentence
- Idea 2: Assign a number to each time-step linearly (1,2,3..)
  Problem: Model can face sentences longer than the ones in training

Positional Encodings

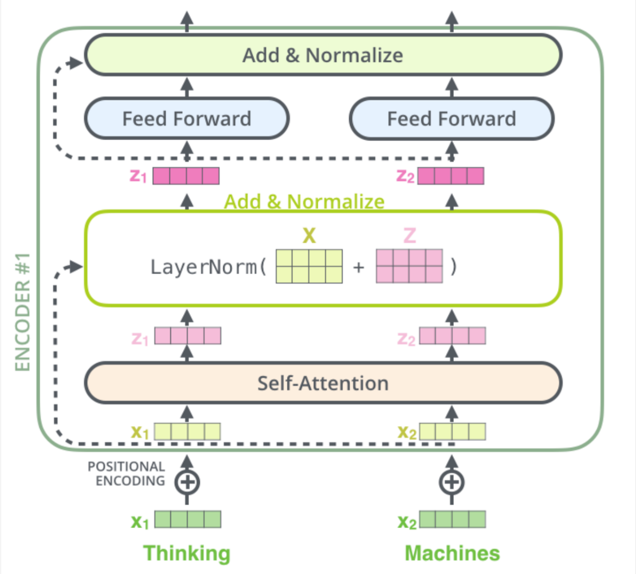
Solution: Add a **fixed** positional embedding (a dense vector that encodes the position in a sentence) to the input word embedding



50 words sentence with embedding dimension of 140.

There is not enough time to explain why this works, if you are curious read here:
`https://kazemnejad.com/blog/transformer_architecture_positional_encoding/`

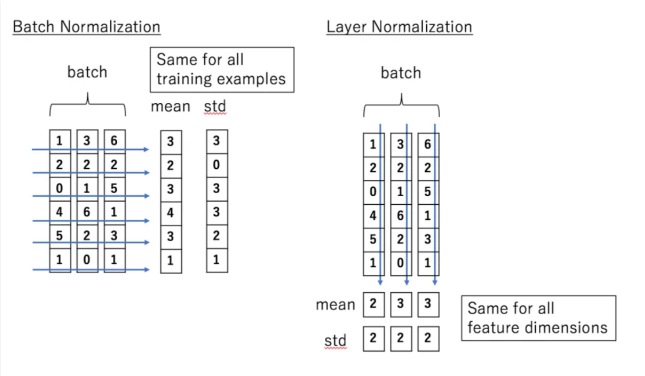
What are we missing?



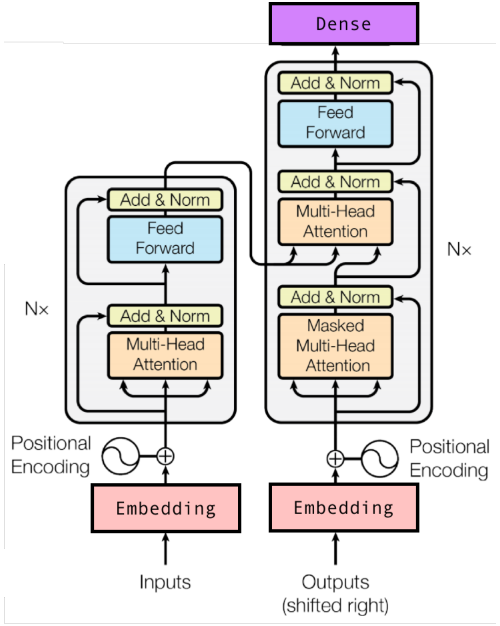
- **Residual connections:** (Shortcut connections) The input of a layer is also addded to the output of the layer. Introduced first in CNNs (ResNet). Helps to train faster and helps against vanishing gradients.

What are we missing?

- **Layer Normalization:** Improvement of Batch Normalization that is independent of the size of the mini batch. It normalizes the inputs across the feature and not the batch dimension.



THE Transformer!



- During training, the embeddings flow all through the transformer in parallel.
- At inference time, decoding id one one step at a time until the end of sentence symbol is reached (like normal seq2seq translation)