Natural Language Processing
ELMo and Transformers (BERT)

Tomasz Walkowiak

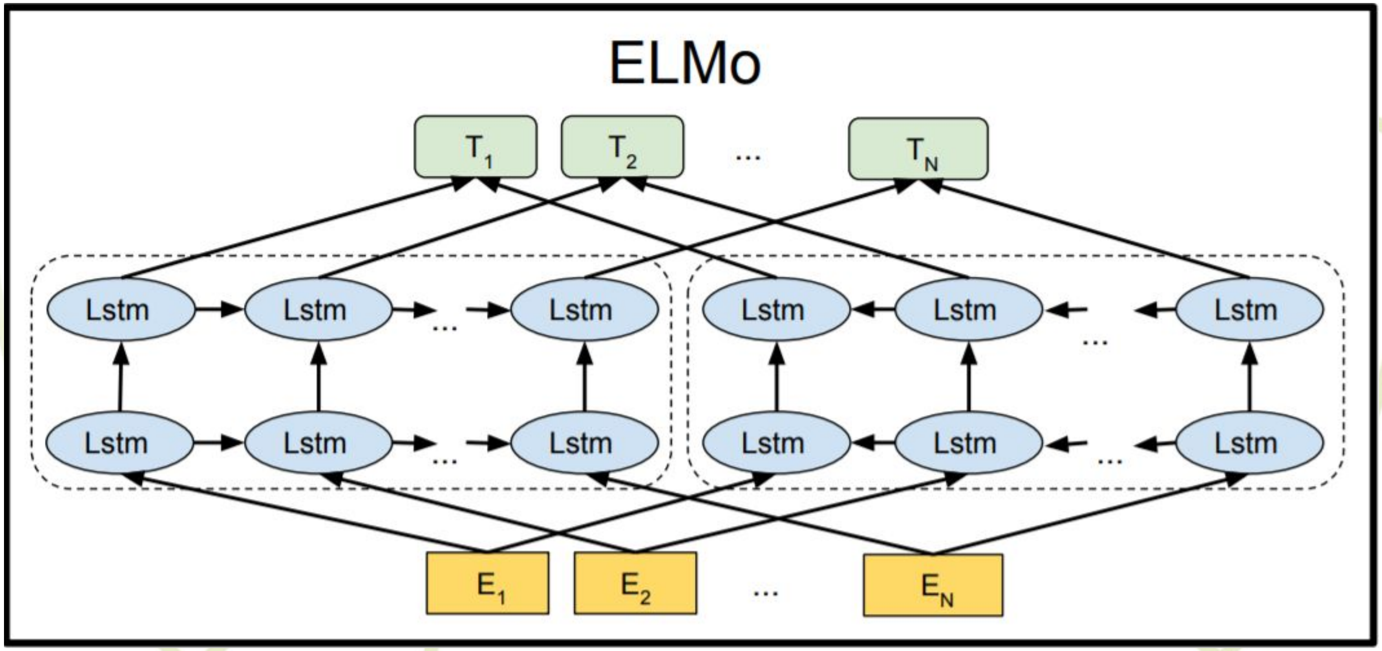
Wroclaw University of Science and Technology
CLARIN-PL
tomasz.walkowiak@pwr.edu.pl

Word embeddings problems

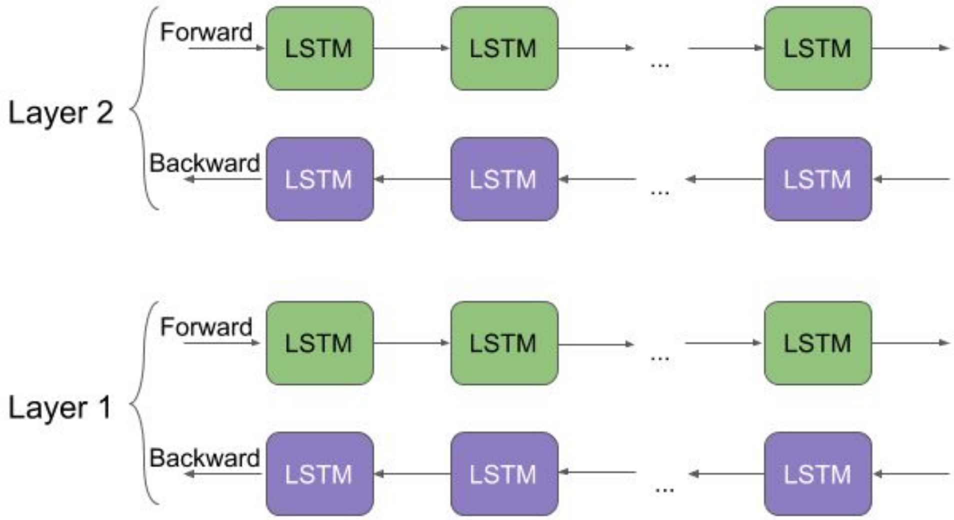
- context-free vectors, but the language is ambiguous (e.g., bank)
- how to turn sequences of embeddings into a single vector?
  - average of embeddings (fastText)
  - LSTM
- problems with LSTM
  - sequential processing, state-by-state, slow
- word2vec, fastText models are simple, not “deep”
  - one layer compared to 50-150 in ResNet (CNN for images)
  - just LUT for words or tokens (sub-words)

Embeddings from Language Models

- Washington Uni./Microsoft/Allen Inst. -Peters et al. (2018)
- Contextual language model, Two-way, two-layer
- Separate LSTMs process forward and backward sequences and hidden layers at each stage are combined to form the cell output



ELMo - inference



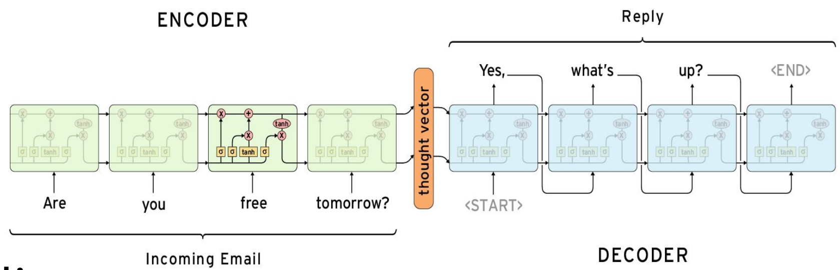
www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

Fune-tuning for downstream tasks

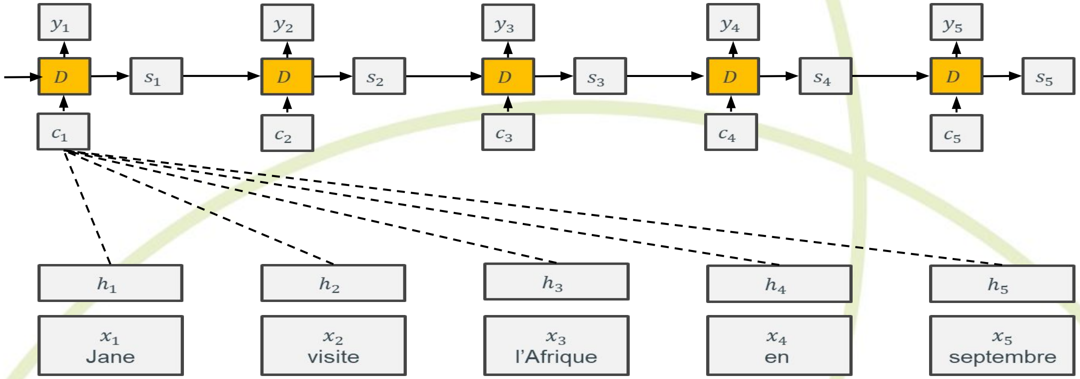
Problems

- sequential
- slow
- 13.6-93.6 mil. parameters

Attention



- RNNs are sequential and linear
- Sentences can be very long, language is not linear
- How to parallelize the process?
- How to calculate the input (context vector) to the decoder based on the encoder states?
- Solution:
  - Bahdanau'2014



  - Adding focus, attention => biological analogies
  - Context vector=weighted sum from hidden states (something like CNN)
  - alphas – by softmax (sum to 1) from a layer learned in parallel with the whole network

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

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

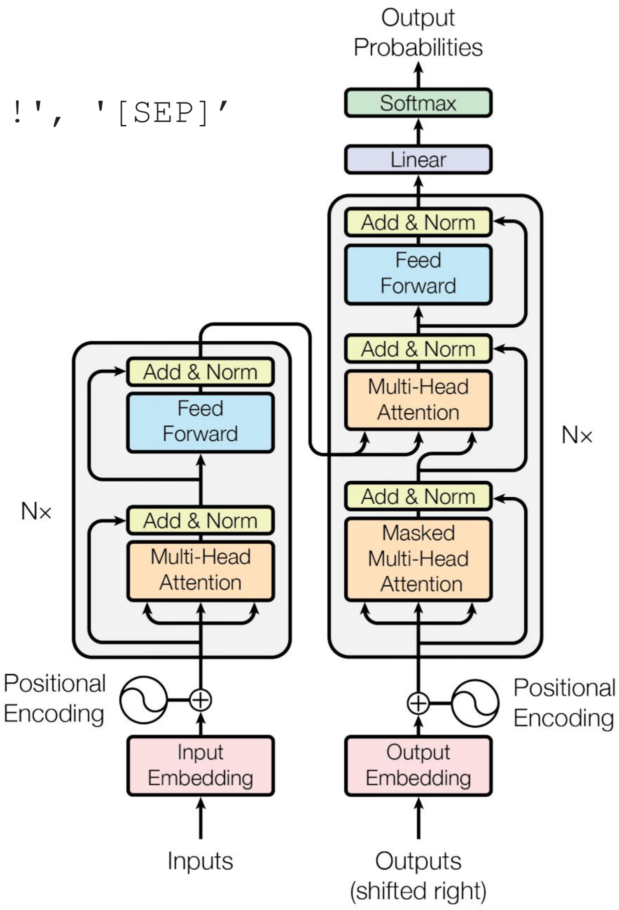
$$e_{ij} = a(s_{i-1}, h_j)$$

Transformer

- „Attention Is All You Need”, Google, 2017, for machine translation
- http://nlp.seas.harvard.edu/annotated-transformer/
- Not patented !!!!, no recursion
- Tokenizer
  - statistically learned, text -> list of tokens
  - `Tomasz lives in Vienna!`
  - `'[CLS]', 'tomas', '##z', 'lives', 'in', 'vienna', '!', '[SEP]'`
  - `[101, 12675, 2480, 3268, 1999, 6004, 999, 102]]`

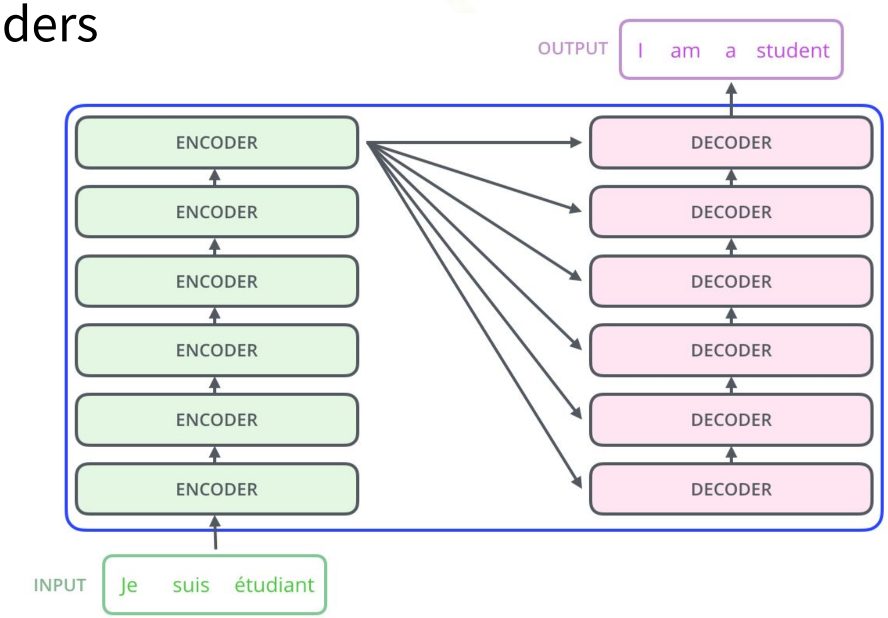
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

- Position encoding
- Token encoding
  - input and output (learned LUT)
  - text-> 2D array (tokens, encodings)
  - with max. length
- Uses 3 types of attention
  - Encoder self-attention
  - Decoder self-attention
  - Multi-head for encoder and decoder

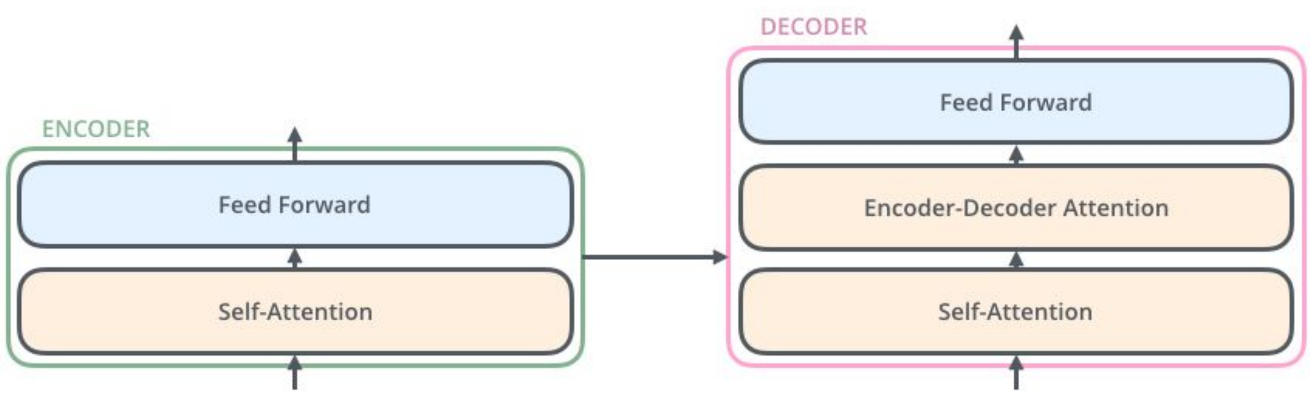


Transformer - architecture

- several layers (6) of encoders and decoders



- Encoder and decoder consist of :
  - self-attention and feed-forward (linear + RELu) layers



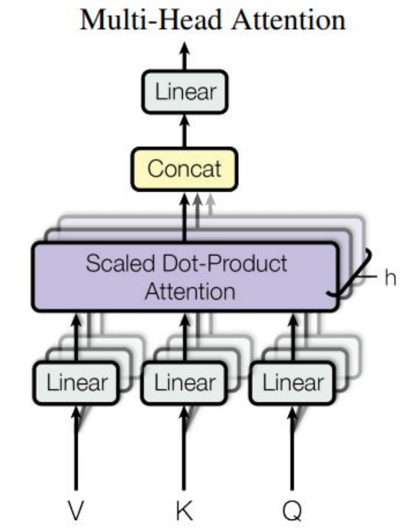
Attention

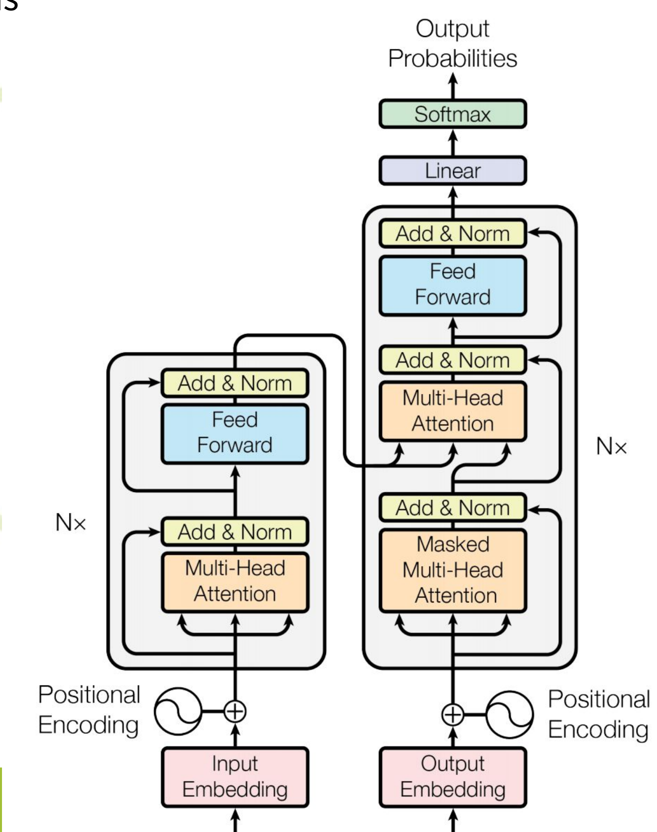
- Input –matrixes (512 x number of tokens)
  - **K**ey , **V**alue, **Q**uery (Q,K – dimension d_k)
- Scaled dot-product attention

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

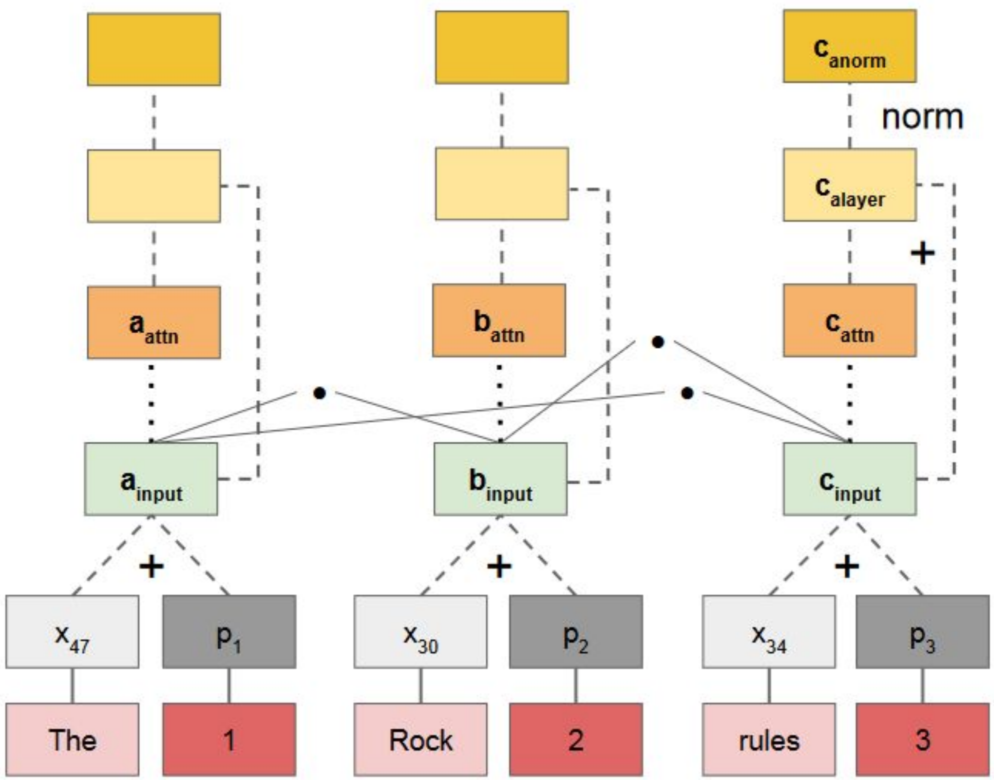
weighted average after tokens

- Multi-head
  - h – various linear input transformations (h=8) 512 to 64
  - scaled dot-product attention (d_k=64)
  - concatenation: 8*64 =512
  - additional linear transform
  - K,V,Q in the encoder are identical
  - inputs (for each token) or output(s) from the previous encoder
- In the decoder
  - the same (the first), but masking future tokens
  - K,V from the last encoder, Q - from the decoder





Encoder (simplified)



$$\mathrm{c}_{\mathrm{anorm}} = \frac{\mathrm{c}_{\mathrm{alayer}} - \mathbf{mean}(\mathrm{c}_{\mathrm{alayer}})}{\mathbf{std}(\mathrm{c}_{\mathrm{alayer}}) + \varepsilon}$$

$$\mathrm{c}_{\mathrm{alayer}} = \mathrm{c}_{\mathrm{attn}} + \mathbf{Dropout}(\mathrm{c}_{\mathrm{input}})$$

$$\mathrm{c}_{\mathrm{attn}} = \mathbf{sum}\left([\alpha_1 \mathrm{a}_{\mathrm{input}}, \alpha_2 \mathrm{b}_{\mathrm{input}}]\right)$$

$$\alpha = \mathbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{{\mathrm{c}_{\mathrm{input}}}^{\top} \mathrm{a}_{\mathrm{input}}}{\sqrt{d_k}}, \frac{{\mathrm{c}_{\mathrm{input}}}^{\top} \mathrm{b}_{\mathrm{input}}}{\sqrt{d_k}}\right]$$

$$\mathrm{c}_{\mathrm{input}} = x_{34} + p_3$$

A look at the Transformer inference

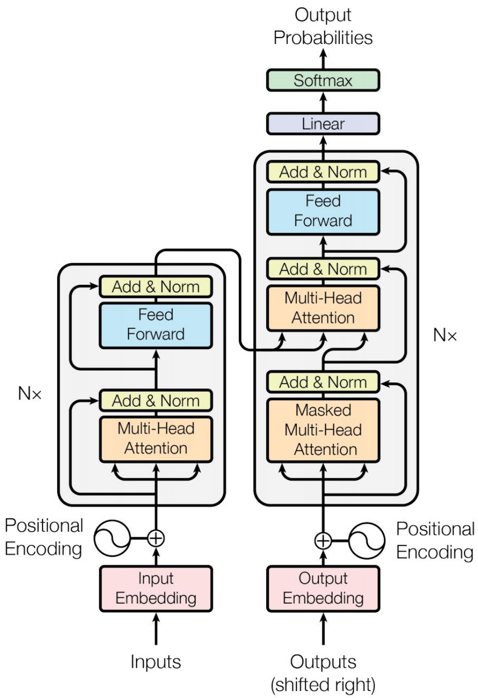
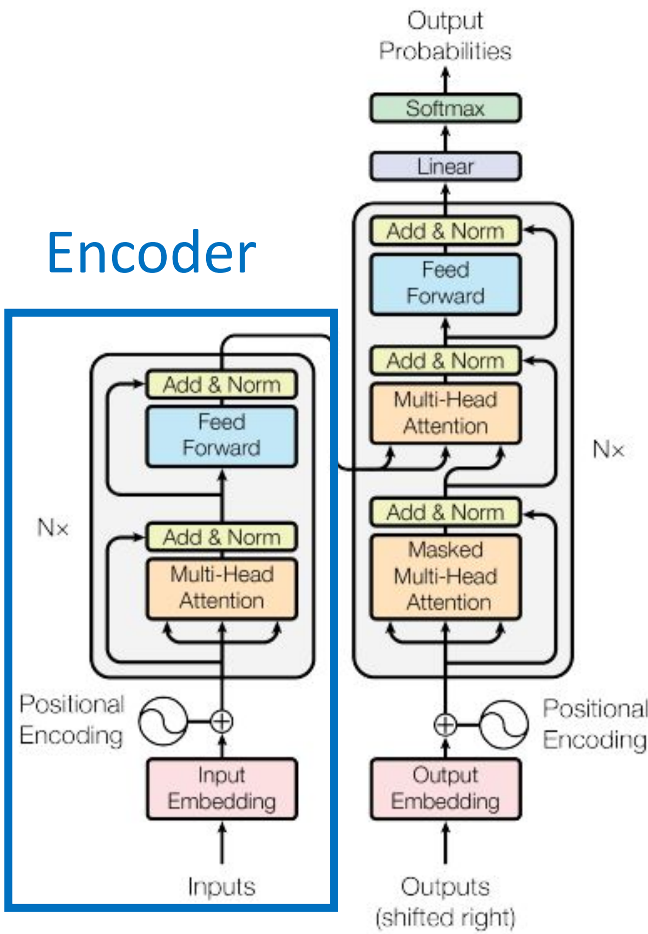


Figure 1: The Transformer - model architecture.

decoder is autoregressive at inference time
and non-autoregressive at training time

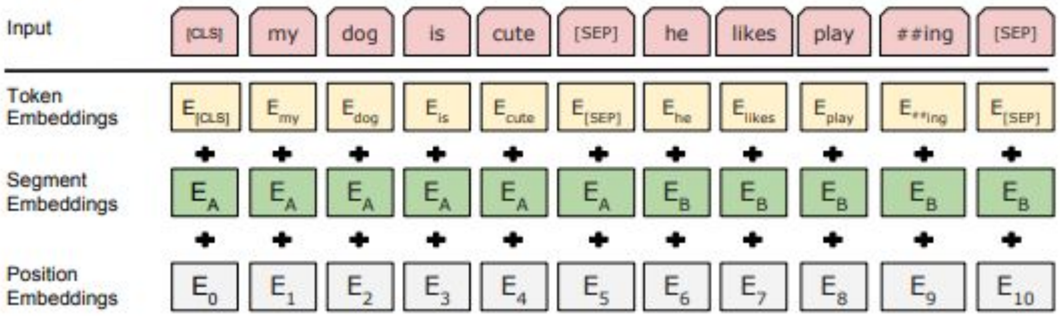
BERT

- Google - 2018
- BERT is an encoder from a transformer
  - 12 (24) encoders, size 768,1024, head 12,16
- Language model learned
  - on a very large but unannotated corpus
  - but unlike the other models
    - simultaneously on two tasks
  - and tuned in downstream tasks
- Ready models, to be downloaded
- In the original work
  - https://github.com/google-research/bert
  - Monolingual: English, Chinese
  - multilingual: simultaneously for 104 languages
- Now
  - Huggingface - everything , including



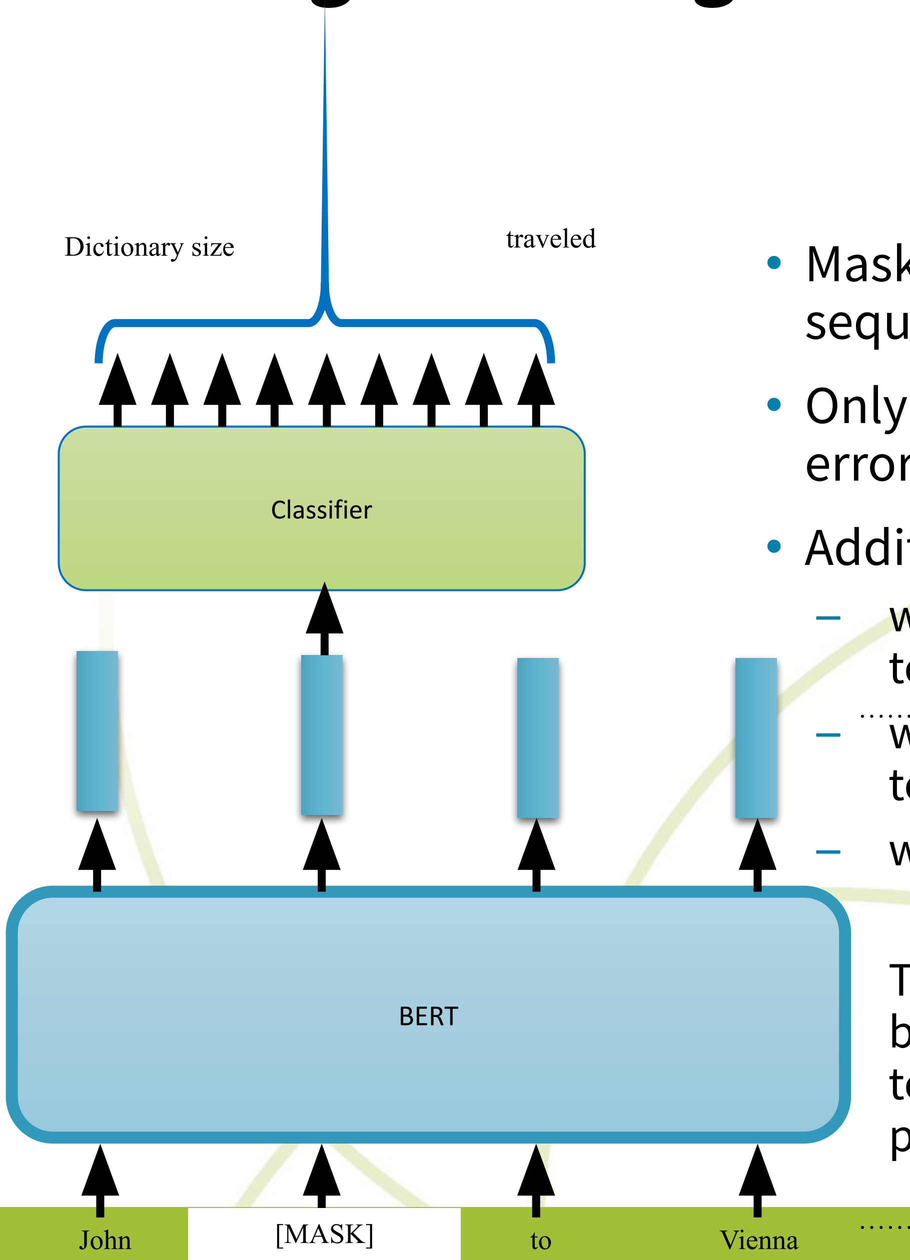
BERT input

- Sum of 3 embeddings



- BERT contains its own tokenizer: 'em', '##bed', '##ding', '##s'
  - List of words, word fragments, characters
  - Unknown word is broken into fragments and/or characters
  - For multilingual version is in the dictionary: 120k entries, for English 30k
  - Token->vector (size 768 or 1024)
  - Text -> array (768/1024 by 512)
  - All embeddings are learned !!!

Training - masking



- Mask randomly 15% of the tokens in each sequence, [MASK].
- Only answers for [MASK] are taken to the error
- Additional heuristics
  - with probability 80%, replace the selected token with [MASK];
  - with probability 10%, replace with a random token;
  - with probability 10%, don't change it.

The model predicts only the missing token, but there is no information about which tokens are replaced or which tokens are predicted

Training – next sentence

- binary classifier
- whether one sentence is the next sentence of another
- additional tokens
  - [CLS] - for the output of the classifier
  - [SEP] - sentence boundary
- the learning error is the sum of the average masked LM probability and the average probability of predicting the next sentence

Inference

- input: token id-s, token masks, sentence ids
- output: embedding (768-dimensional vector) for each token

Practical usage (downstream tasks)

- the classification layer is added
- BERT + extra layers are tuned !!!
- we use batches (samples on which the model makes predictions in one forward pass) – GPU architecture
- we need padding (the same length) and truncation (BERT size is limited)

Deep language models summary ELMo, BERT

- Important features
  - based on very large datasets that do not require labeling
  - supervised learning - despite formal lack of labels
  - publicly available models
  - can be re-used in tasks where annotated sets (costly) are small (transfer learning)
  - SoA for NLP (BERT)
  - BERT and ELMo are contextual, works with polysemy and pronouns
- Problems
  - size of models
    - BERT 345 million parameters, 1.4 GB
    - but Sent2vec for PL: 20 GB
  - have a lot of complexity even when generating vectors
  - they require GPUs (even during inference)
    - in BERT it is not as bad as in ELMo
    - usage of ONNX

What happened after 2018 ?

- BERT modifications
  - RoBERTa - Facebook
  - DistilBERT – distilled version
    - HuggingFace, ½ of BERT weights, 95% effectiveness
  - SentenseTransformers – metric learning
  - LongFormers
- Text -> Text (text generation)
  - GPT (Open AI Transformer)
    - only decoders
    - GPT-2 (2019) – 1.5 billion par., 8 mil. web pages
    - GPT-3 (2020) – 175 billion parameters, 96 attention layers
    - ChatGPT (2022) GPT3.5
  - T5 (Google)
    - Encoders and decoders
    - From 60 million do 11 billion parameters
    - C4 data set
  - BART

Bert for text classification

- Fine-tuned on downstream tasks, transfer learning
  - pick up a pre-trained BERT
  - add a dense layer (classifier) to token- 0 (pooling operation)
  - tune (on our task) all weights (except embeddings)
- How
  - Transformers from HuggingFace (PyTorch)

HUGGING FACE

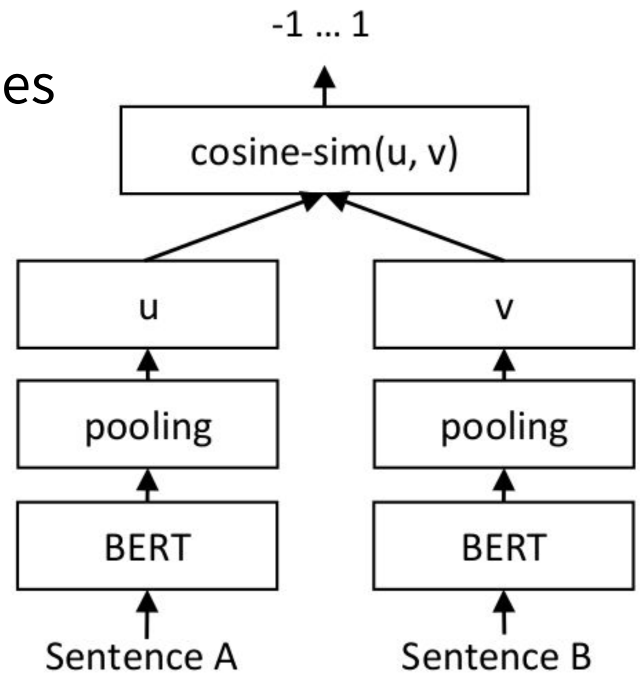
```
from transformers import …
model_name = "bert-base-uncased”
tokenizer = BertTokenizer.from_pretrained(model_name)
train_encodings = tokenizer(train_texts, truncation=True,…)
model=BertForSequenceClassification.from_pretrained(model_name, num_labels=…))
train_dataset = Dataset(train_encodings, labels)
trainer = Trainer( model=model,args=…,train_dataset=train_dataset)
trainer.train()

… = trainer.predict(test_dataset)
```

Example results for subject classification (for Polish)

Data set	TF-IDF		doc2vec		fastText		Polbert		HerBERT	
	mean	std	mean	std	mean	std	mean	std	mean	std
Wiki	0.8302	0.0038	0.9044	0.004	0.8816	0.0055	0.943	0.0064	**0.9494**	0.0066
Press	0.9438	0.0056	0.9559	0.0038	0.9587	0.0049	0.9704	0.0068	**0.9718**	0.0091
Web	0.5627	0.0098	0.5764	0.0274	0.5451	0.0221	0.5533	0.0194	**0.5952**	0.0324
Rev	0.9425	0.0046	0.917	0.0036	**0.9659**	0.0037	0.9524	0.0077	0.9555	0.0106
Qual	0.7943	0.0165	0.8023	0.0116	0.8155	0.011	**0.83**	0.0137	0.8221	0.0253

Sentence Transformers

- Sentence embedding based on text similarities
- Twin/Siamese networks
- Metric learning
- Good vectors for
  - clustering
  - searches (Q&A)
  - **visualization**




```
from sentence_transformers
        import SentenceTransformer
sentences = ["This is an example sentence.", "Each…"]
model = SentenceTransformer
('sentence-transformers/distiluse-base-multilingual-cased-v1')
embeddings = model.encode(sentences)
```