Transformer & BERT Pre-training

Sebastian Hofstätter

sebastian.hofstaetter@tuwien.ac.at
/s_hofstaetter

TU WIEN

Today

Transformer & BERT Pre-training

1. **Transformer Architecture**
   - Self-attention
   - Positional encoding
2. **BERT Pre-training**
   - Masked language modelling task
   - The BERT model
   - HuggingFace: Sharing in the community
3. **Extractive QA**
   - One example of a downstream task (useful in IR)

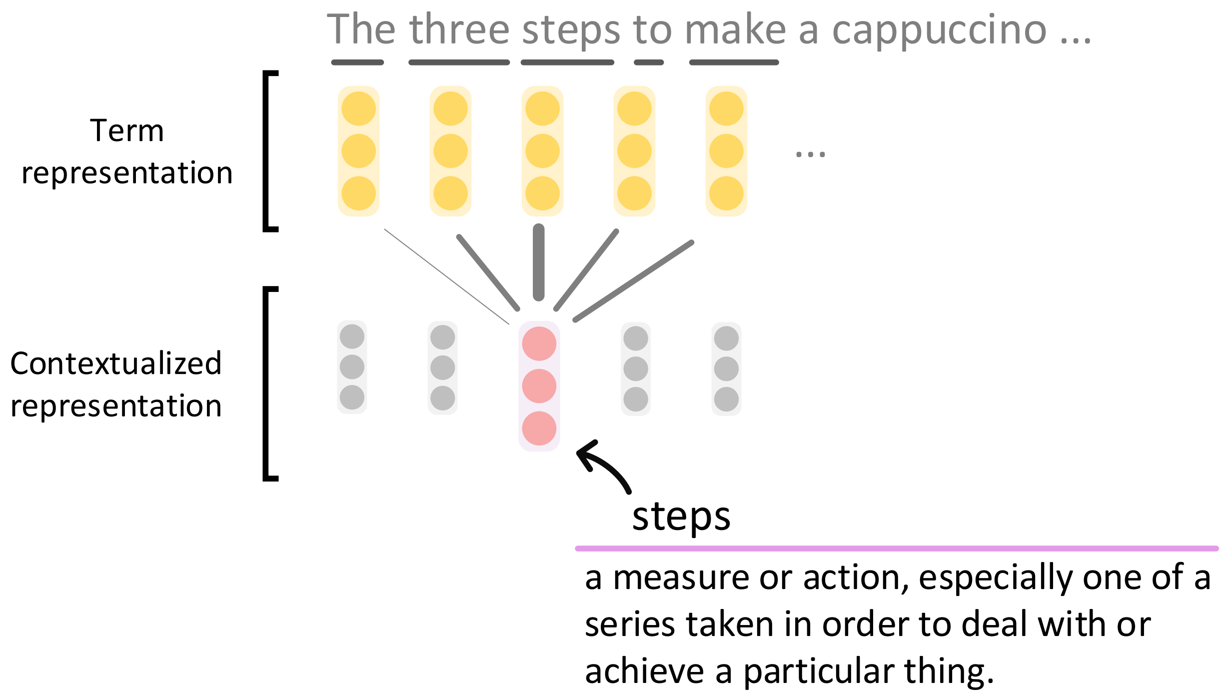
Another Versatile Building Block

- The Transformer architecture (as CNNs and RNNs) is not task specific
  - It operates on sequences of vectors, what we do with it is our choice
- Quickly gained huge popularity
  - Pre-trained Transformers are now ubiquitous in NLP & IR research, increasingly also in production systems
- Typical model sizes are not possible without modern hardware
  - Transformers are basically designed for what GPUs are best at: *large matrix multiplications*

Transformer

Contextualization via Self-Attention

Contextualization via Self-Attention



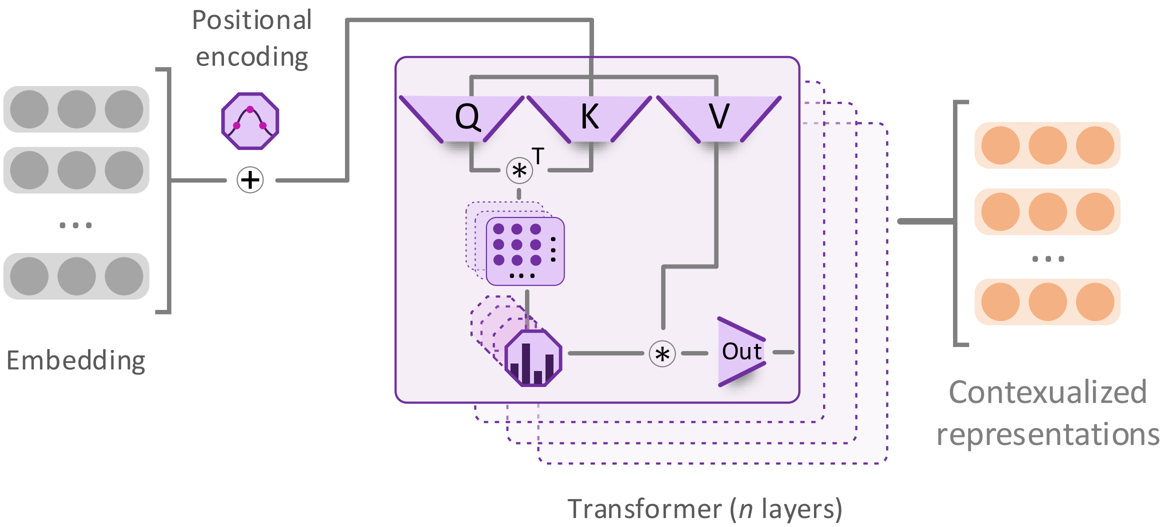
- Learn meaning based on surrounding context for every word occurrence
- This *contextualization* combines representations
- Context here is local to the sequence (not necessary a fixed window)
- Is computationally intensive $O(n^2)$
    - Every token attends to every other token

Transformer

- Transformers contextualize with multi-head self-attention
  - Every token attends to every other token $O(n^2)$ complexity
- Commonly Transformers stack many layers
- Can be utilized as encoder-only or encoder-decoder combination
- Do not require any recurrence
  - The attention breaks down to a series of matrix multiplications over the sequence
- Initially proposed in translation
  - Now the backbone of virtually every NLP advancement in the last years

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, et al.
Attention is all you need. In NeurIPS. 2017.

Transformer – Architecture



- We embed (subword) tokens
- We add a positional encoding
- In each Transformer-Layer:
    - Project each vector with 3 linear layers to **Q**uery, **K**ey, **V**alue
    - Transform projections to another multi-head dimension
    - Matrix-multiply Query & Key
    - Get Q-K attention via softmax
    - Multiply attention with Values and project back to output

Nice detailed walkthrough code + paper:
https://nlp.seas.harvard.edu/2018/04/03/attention.html

Self-Attention Definition

- The Transformer Self-Attention is defined as:

$$\text{SelfAttention}(Q, K, V) \quad = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) * V$$

- Q, K, V are projections of the **same input** sequence
- This definition hides quite a bit of complexity, visible in the code

Q	Attention "Query"
K	Attention "Key"
V	Attention "Value"
d_k	Dimension of key embeddings

Transformer in PyTorch

- Native support in PyTorch
  - Brings many speed, stability, robustness improvements
  - Raw Transformer Encoder:

```
encoder_layer = nn.TransformerEncoderLayer(d_model=300,nhead=10,dim_feedforward=300)
transformer = nn.TransformerEncoder(encoder_layer, num_layers=2)

src = torch.rand(10, 32, 300)
out = transformer(src)
```

  - Can be a bit tricky to apply, especially masking & padding, & transposed input

Documentation: https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html
Tutorial: https://pytorch.org/tutorials/beginner/transformer_tutorial.html

Transformer – Positional Encoding

- Transformers add sinusoid curves to the input, before the attention
  - Informs about relative position inside the sequence
  - Removes need for explicit recurrence patterns

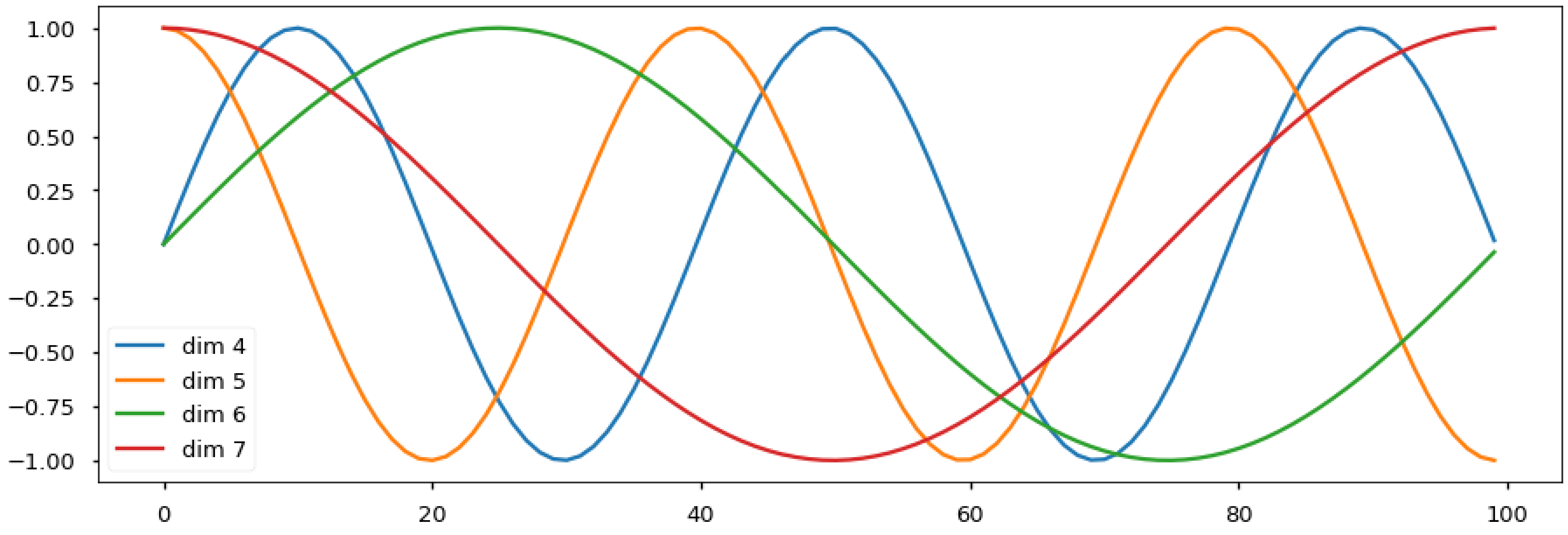


Figure from https://nlp.seas.harvard.edu/2018/04/03/attention.html

Transformer - Variations

- Non-exhaustive list of Transformer variants
- A lot focus on efficiency & long input
  - Break $O(n^2)$ runtime and memory requirement
  - Allow for thousands of input tokens
- Incredible speed of innovation

Overview of recent Transformer literature [Weng, 2020]
Attention is all you need [Vaswani et al., 2017]
Running self-attention on pre-segmented text [Al-Rfou et al., 2019, Hofstätter et al., 2020]
Localized Attention Span (Image Transformer) [Parmar et al., 2018]
Transformer-XL [Dai et al., 2019]
XLNet [Yang et al., 2019]
Gated Transformer-XL [Parisotto et al., 2019]
Reformer [Kitaev et al., 2019]
Reversible Residual Network [Gomez et al., 2017]
Routing Transformer [Roy et al., 2020]
Sparse Sinkhorn Attention [Tay et al., 2020]
Sparse Transformers [Child et al., 2019]
Megatron LM [Shoeybi et al., 2019]
Longformer [Beltagy et al., 2020]
Transformer-XH [Zhao et al., 2014]
Roberta [Liu et al., 2019]
Adaptive Attention Span [Sukhbaatar et al., 2019]
Adaptive Computation Time [Graves, 2016]
Universal Transformers [Dehghani et al., 2018]

More at:
https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

In-Depth Resources for Transformers

- Popularity naturally brings more educational content
  - More than we could cover today
- Here are some pointers, if you want to know more about Transformers:

  http://jalammar.github.io/illustrated-transformer/

  https://mccormickml.com/2019/11/11/bert-research-ep-1-key-concepts-and-sources/

  https://github.com/sannykim/transformers

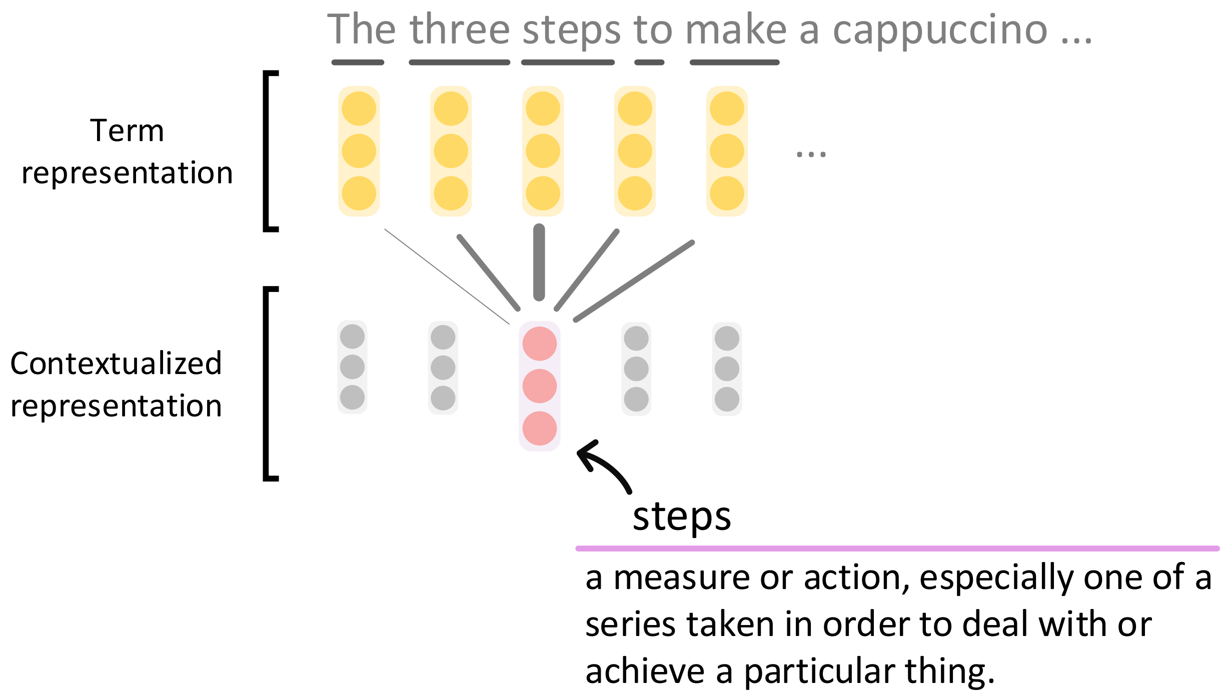
Pre-Training

Workflows, Tasks, Models

Pre-Training Motivation

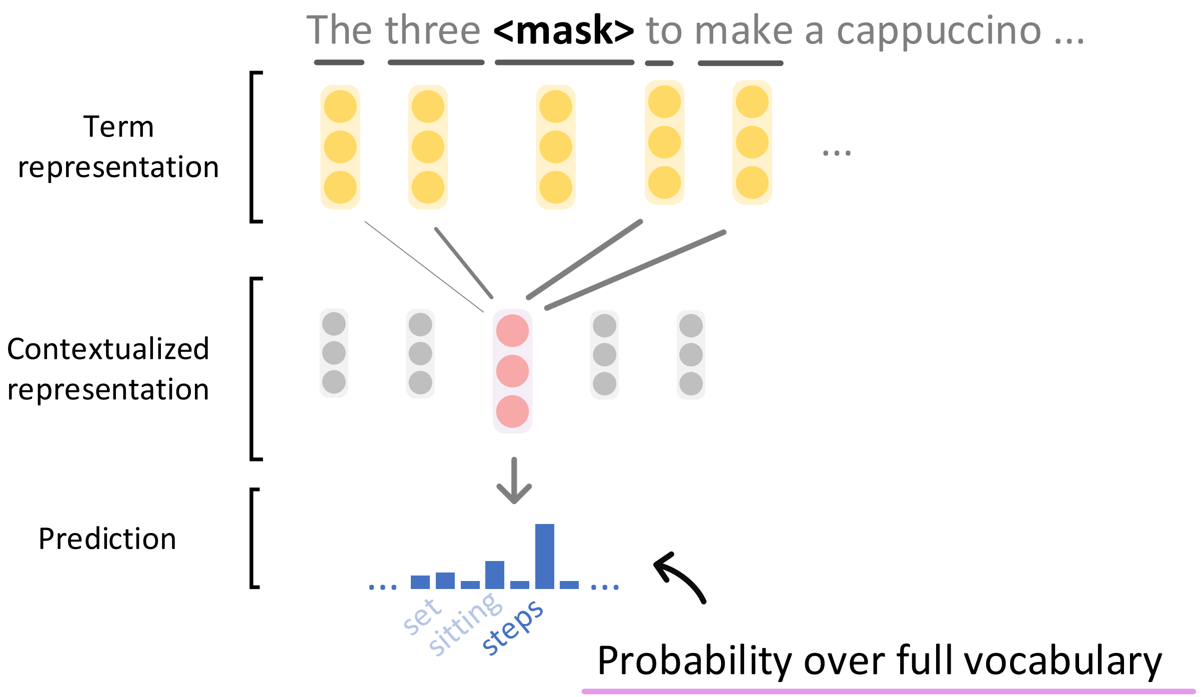
- Most tasks don’t come with huge training data
- Large (high capacity) models need a lot of data to work well
- Idea: Create a task-agnostic training that works unsupervised on large sets of text
  - Teaches the model about the meaning of words/patterns in the language
  - Unsupervised: We have no labels, but make predictions about words/sentence positions
- Continues the tradition of word2vec (albeit at a larger model scale)
- After a model is pre-trained it can be fine-tuned for a variety of tasks

Masked Language Modelling



- Recall our example:
  - We want a good context-dependent representation of “steps”
- Unsupervised Pre-training:
  - Take text and mask random words
  - Try to predict original word
  - Update weights based on loss of prediction vs. actual word

Masked Language Modelling



- Training procedure:
  - Take text and mask random words
  - Try to predict original word from context words
  - Update weights based on loss of prediction vs. actual word
- Loss requires prediction over vocabulary
  - Prohibitive for large vocabs
  - Models use WordPiece or BytePair splitting of infrequent terms

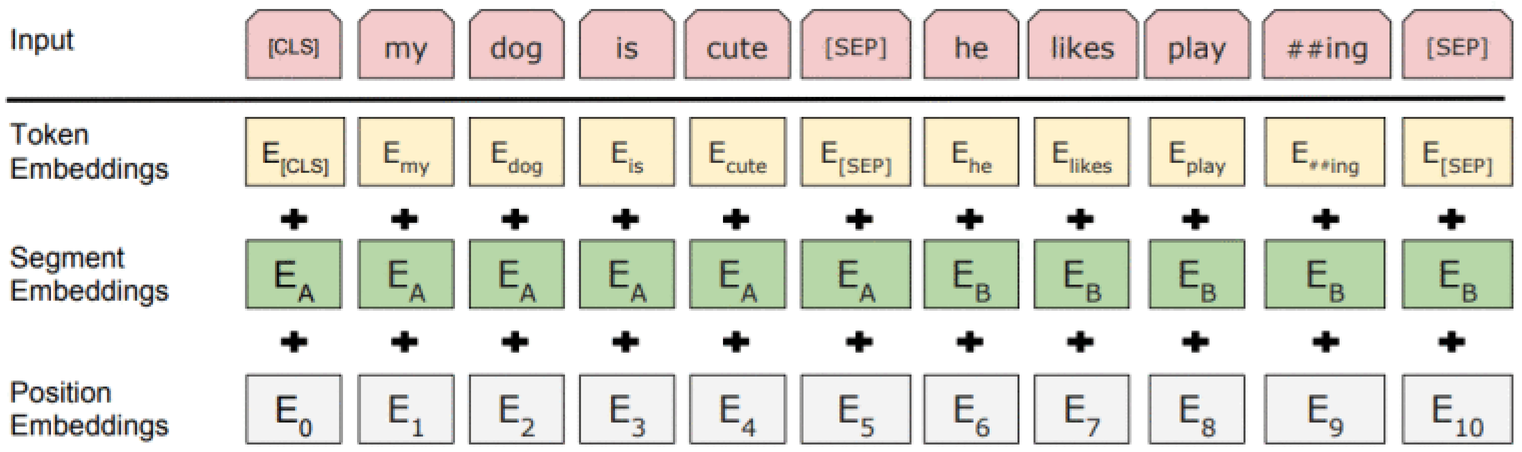
BERT

- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
- Large effectiveness gains on *all* NLP tasks
- Ingredients:
  - WordPiece Tokenization & Embedding (small vocab, covers infrequent terms)
  - Large model (many dimensions and layers – base: 12 layers and 768 dim.)
  - Special tokens (shared use between pre-training and fine-tuning)
    - **[CLS]** Classification token, used as pooling operator to get a single vector per sequence
    - **[MASK]** Used in the masked language model, to predict this word
    - **[SEP]** Used to indicate (+ sequence encodings) a second sentence
  - Long MLM pre-training (weeks if done on 1 GPU)

Devlin et al. 2019 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT - Input

- Either one or two sentences, always prepended with [CLS]
  - BERT adds trained position embeddings & sequence embeddings



More info: https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/

BERT - Model

- Model itself is quite simple: n Layers of stacked Transformers
  - Using LayerNorm, GeLU activations (like ReLU, but with a grace swing under 0)
  - Task specific heads on top to pool [CLS] or individual token representations
  - Every Transformer layer receives as input the output of the previous one
- The [CLS] token itself is only special because we train it to be
  - No mechanism inside the model that differentiates it from other tokens
- Novel contributions center around pre-training & workflow

BERT - Workflow

- Someone with lots of compute or time pre-trains a large model
  - BERT uses Masked Language Modelling [MASK] and Next Sentence Prediction [CLS]
- We download it and fine-tune on our task

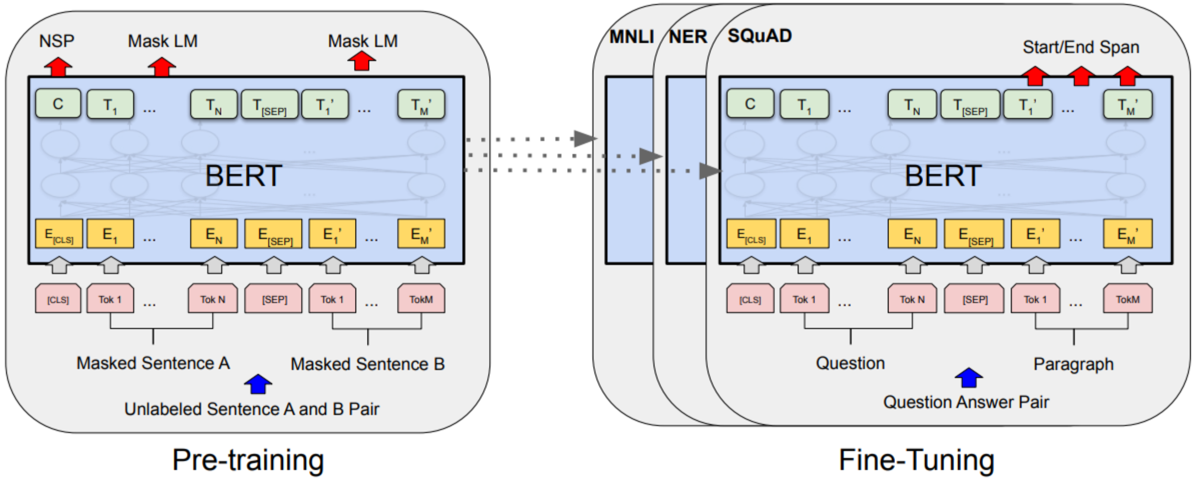


Figure taken from the BERT paper

BERT++

- Same as with Transformer variations, there are now many BERT variants
  - For many languages
  - Domains like biomedical publications
  - Different architectures, but similar workflow:
    Roberta, Transformer-XL, XLNet, Longformer …
- Main themes for adapted architectures:
  - Bigger
  - More efficient
  - Allowing for longer sequences (BERT is capped at 512 tokens in total)

Rogers et al. A Primer in BERTology: What we know about how BERT works https://arxiv.org/abs/2002.12327

Pre-Training Ecosystem

- With simple 1-word-1-vector embeddings (word2vec) sharing was as simples as a single text file containing both vocab + weights
  - We could simply load the weight matrix into bigger models
  - Mostly whitespace tokenization meant very little complexity

- BERT et al. re-use requires:
  - Exact model architecture (specific code and config) for hundreds of details
  - Weights for 100+ modules
  - Specific tokenizer rules for sub-word tokenization and special token handling
  - A single text file doesn’t work here anymore …

HuggingFace: Transformers Library

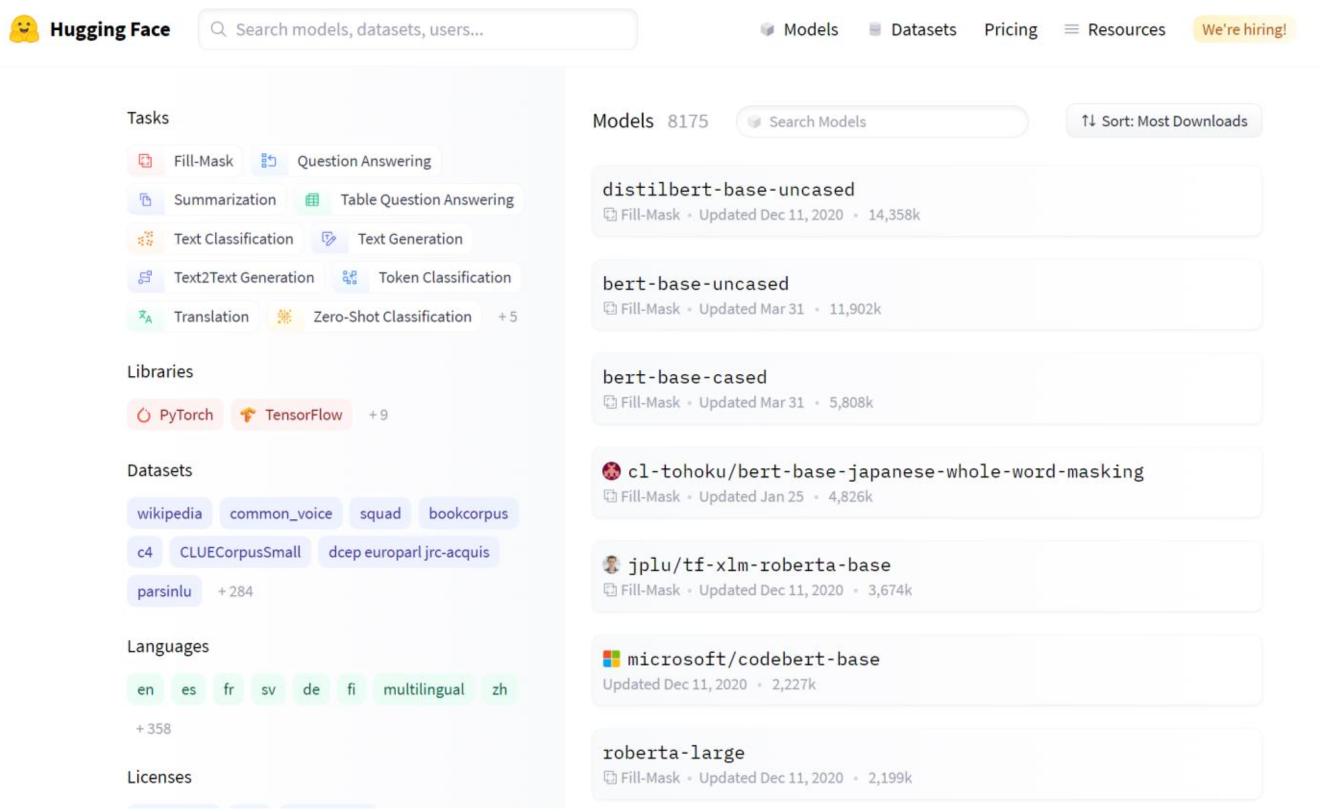
- Started as a port of TensorFlow implementation of BERT to PyTorch
- Quickly morphed into a multi-use, multi-model, multi-framework library
  - Out-of-the-box support for: tokenization, BERT architectures, many NLP tasks (not yet for neural re-ranking and only spotty dense retrieval*)
  - Expanding to even more use cases quickly (f.e. speech recognition)

- Gained huge popularity, because it really is easy to use
  - The pre-training ecosystem needs this for broad access

*As of April 2021; To the code: https://github.com/huggingface/transformers/

HuggingFace: Model Hub

- Not only one-way model code, but a hub for:
  - Model definitions
  - Trained models
- Everyone can upload models
  - Already thousands of entries
- Data is hosted by HuggingFace
  - Don't have to worry about public storage

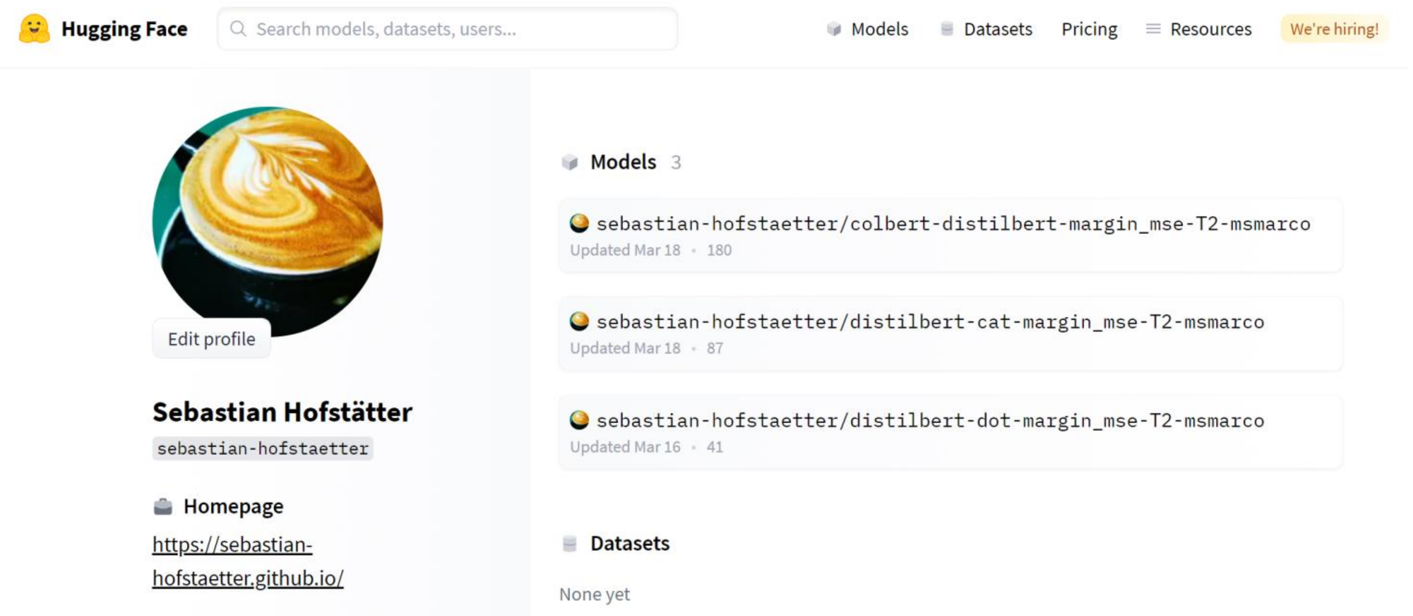


URL: https://huggingface.co/models

HuggingFace: Model Hub

- Each model is packaged in a library defined format and uploaded & versioned via git-lfs

- Readme (like GitHub) is displayed as model card to be able to explain what is trained here



Our models: https://huggingface.co/sebastian-hofstaetter

HuggingFace: Getting Started

- Getting Started is easy, load the model & tokenizer:

```python
from transformers import AutoTokenizer, AutoModel

pre_trained_model_name = "sebastian-hofstaetter/distilbert-dot-margin_mse-T2-msmarco"
tokenizer = AutoTokenizer.from_pretrained(pre_trained_model_name)
bert_model = AutoModel.from_pretrained(pre_trained_model_name)
```

- Tokenize & encode some text:

```python
passage_input = tokenizer("We are very happy to show you the 🤗 Transformers library for pre-trained language models 🔥.",
                          return_tensors="pt")

passage_encoded = bert_model(**passage_input)
```

- Now, we can do something with the encoded representations

The full example: https://github.com/sebastian-hofstaetter/neural-ranking-kd/blob/main/minimal_bert_dot_usage_example.ipynb

Extractive QA

One NLP task example out of many possible using BERT

Soooo many tasks are solvable with BERT

- Original BERT paper evaluates on:
  - GLUE, SQuAD, SWAG, CoNLL
- Now at ~18 thousand citations, we can assume some more are evaluated
  - As long as your text input is <512 tokens & you can pool the CLS token or learn per-term predictions you can use BERT

- HuggingFace model Hub alone provides out-of-the-box support for dozens of tasks & lists 200+ datasets used by the trained models.

Extractive Question Answering

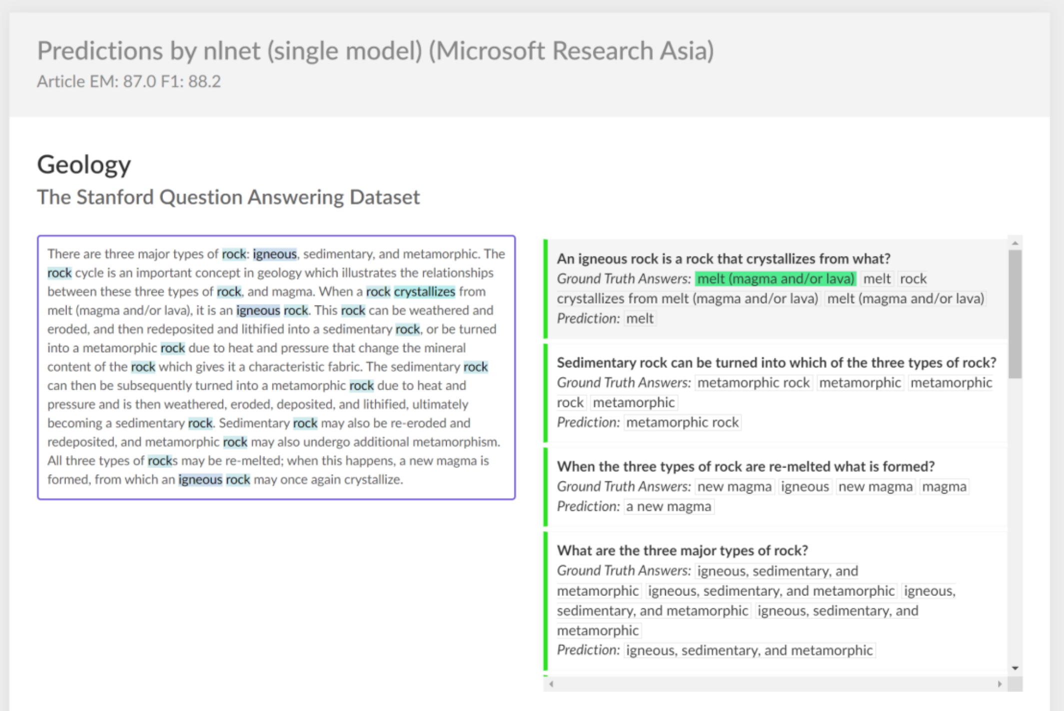
- Given a query and a passage/document:
  Select the words in the passage that answer the query
  - We want to select at least 1 span with a start and end position, that we then can extract
  - Use extracted text in highlighted UI (with surrounding text), chatbot, or audio-based assistant
  - Not perfect: query type must be specific to be answerable with fixed text

- Differs from *Generative* Question Answering
  - Models are tasked to create new text (with new words, more natural conversational style)
  - More complex, more potential for error and biases

Extractive QA: Datasets

- Popular datasets include: SQuAD & NaturalQuestions
  - Both are based on Wikipedia text
  - SQuAD contains artificially created queries, NQ google search queries
  - Both come with fairly large training and evaluation sets

- Many pre-trained models are available for both



Example: https://rajpurkar.github.io/SQuAD-explorer/explore/v2.0/dev/

More datasets: https://huggingface.co/datasets?filter=task_ids:extractive-qa

Extractive QA: Training

- For BERT we concatenate query and passage
  - With the pre-trained special tokens
- Per term output (of BERT) of the passage
  predicts if this token is a start or end token of the answer
  - End tokens are trained with gold-label start positions
  - Beam search can be used to find the best combination
- Loss is based on CrossEntropy of prediction vs ground-truth label
  - Potentially also includes a non-answerable prediction for the passage as a whole (SQuAD 2.0)

Getting Started: https://huggingface.co/transformers/task_summary.html#extractive-question-answering

IR + QA = Open Domain QA

- Having a passage guaranteed to contain the answer is somewhat artificial
- More realistic scenario: we have a collection, and we need to generate candidates first with our IR system
  - Often referred to as Open Domain QA or “retrieve and read”
- Can be separate systems our jointly learned
  - Def. makes evaluation and analysis more complex, as man more moving parts are involved
- Fulfills the initial idea of the immediate answer – search engine presented in the course introduction

Summary: Transformers & BERT

1. Transformers apply self-attention to contextualize a sequence

2. BERT pre-trains Transformers for easy downstream use

3. An open and sprawling ecosystem lowers the barrier of entry

1. Transformers apply self-attention to contextualize a sequence
2. BERT pre-trains Transformers for easy downstream use
3. An open and sprawling ecosystem lowers the barrier of entry

Thank You