Applied Deep Learning for NLP Applications

By Elvis Saravia

April 14, 2020

https://github.com/dair-ai/odsc_2020_nlp

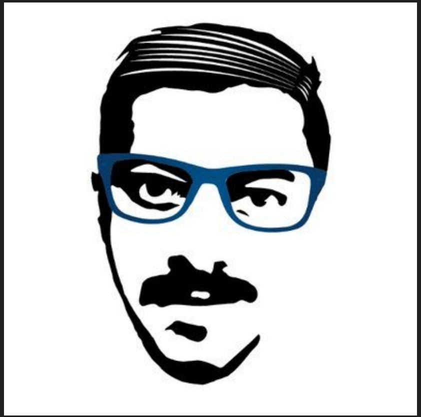
About me

Elvis Saravia

Education Engineer @ Elastic

Independent Research Scientist (ML and NLP)

Editor @ dair.ai



@omarsar0

Agenda

Part 1: Introduction to modern NLP

Part 2: Training and fine-tuning NLP Models

Part 3: Towards building real-world NLP powered applications

What this talk is about and what it is not about

What it is about:

- Hands-on training to learn about how to apply modern NLP techniques
- Brief introduction to important NLP and deep learning concepts

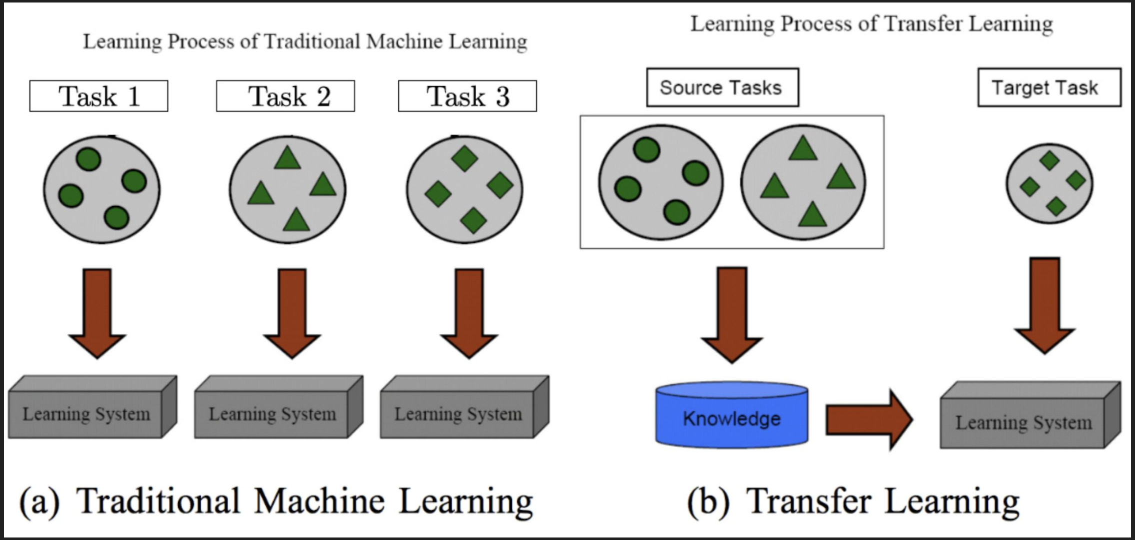
What it is not about:

- An introduction to NLP
- A complete course on NLP
- A course to learn the theoretical aspects of modern NLP techniques

Introduction to modern NLP

Part 1

Transfer Learning in NLP

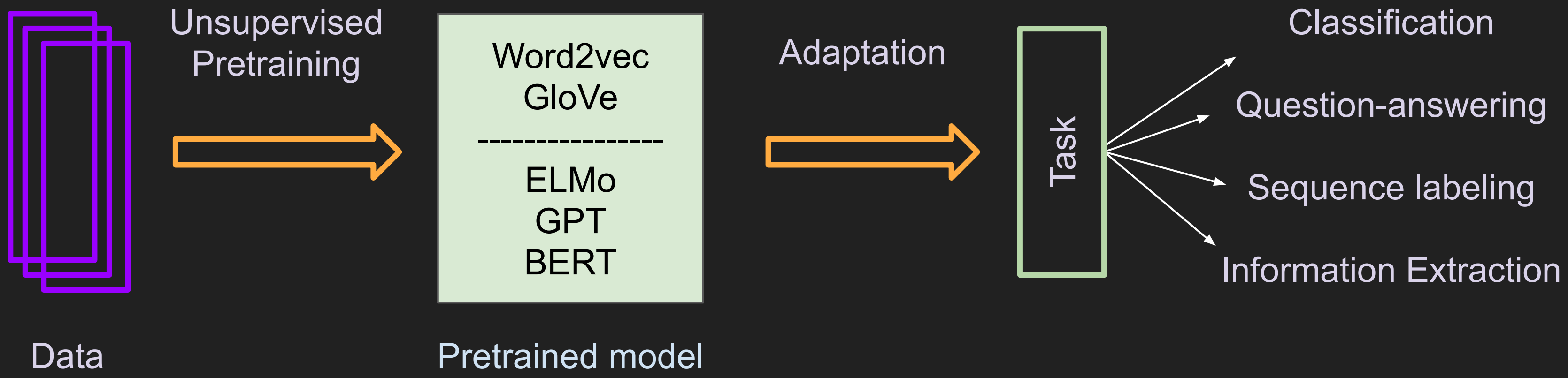


Pan and Yang (2010)

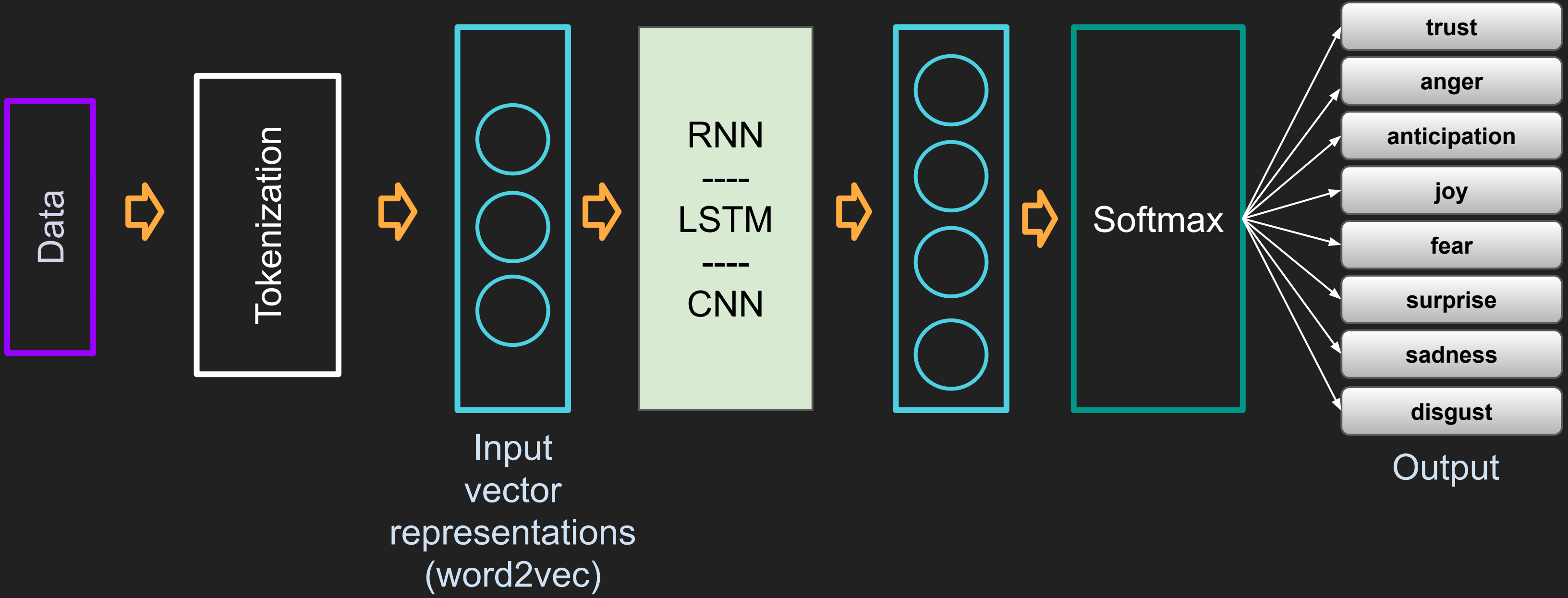
- Some NLP tasks share common knowledge about language
  - Structural similarities
  - Linguistic representations
  - Long-term dependencies
  - Negation
- Lack of annotated data
- Empirically results in SOTA on many supervised tasks

Ruder et al. (2019)

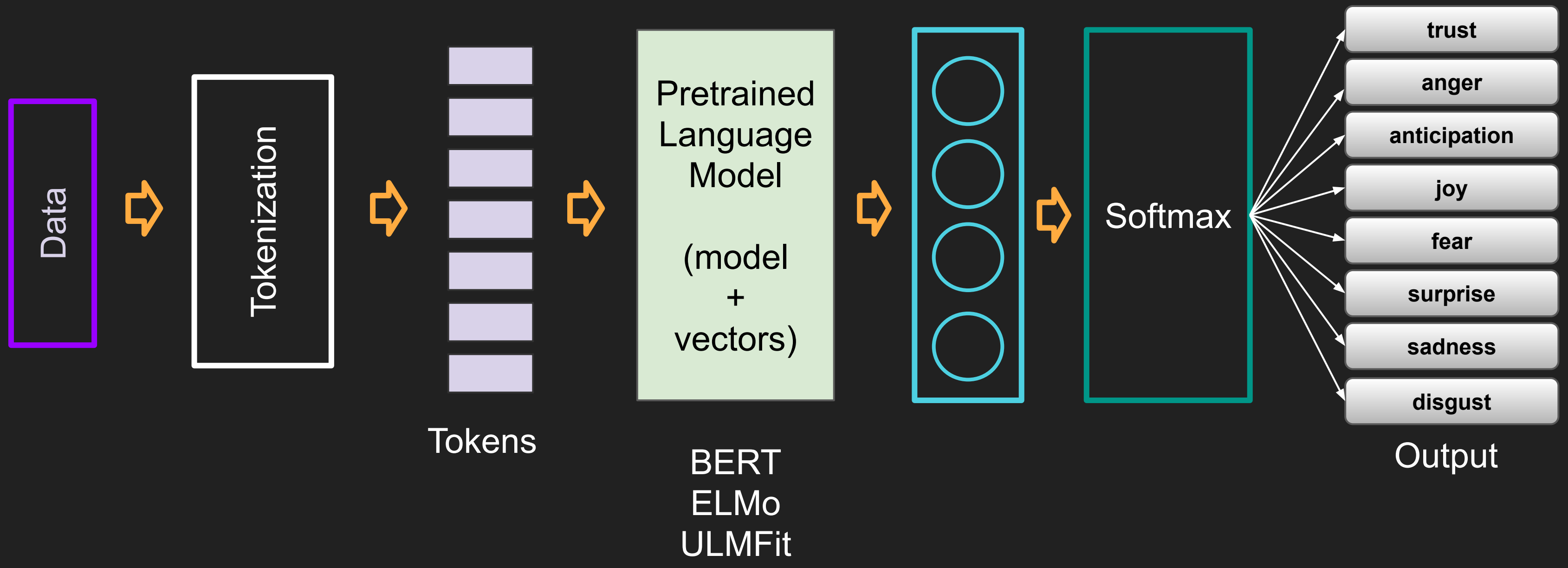
Sequential Transfer Learning



Example: Shallow approach



Example: Deep approach



Tokenization

Tokenization is the process of breaking down sequences into standard units

The models uses the input tokens to learn syntactic (lower levels) or semantic understanding (higher levels) via some neural architecture.

For modern NLP (deep learning NLP) we require two steps:

- Split sequence input into smaller units
- Represent inputs as vectors (e.g. contextualized embeddings) or pass tokens through entire LMs and output representations
- Train or fine-tune the model on the downstream task (e.g. classification)

Tokenization

Challenges when tokenizing your inputs using simple approaches:

- Big vocabularies (e.g. talk, talked, talking, etc.)
- Word combination (e.g. New York, sun, sunflower, check-in, etc.)
- Misspelled words (e.g. laugh, lauhg, etc)
- Abbreviated words (e.g. “ICYMI”, “IMO”, “FOMO”, etc.)
- Segmentation is difficult when working with different languages

Subword Tokenization

Reduces the entries needed in the vocab by representing the dataset with **the least amount of tokens**:

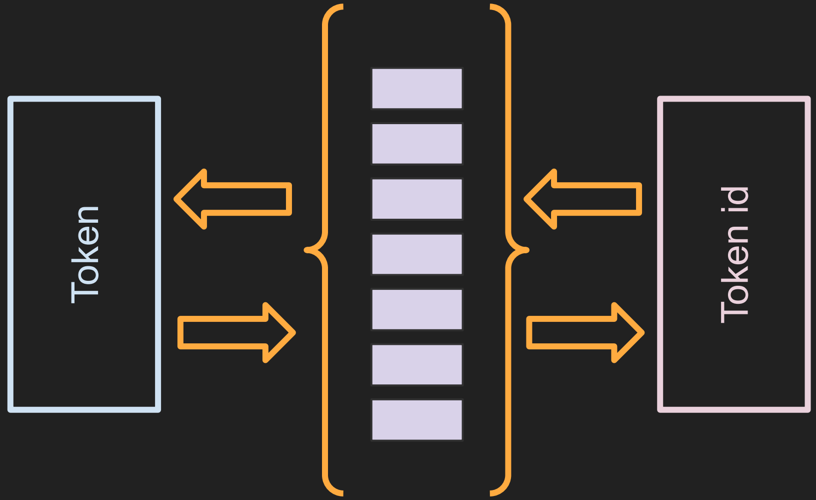
- E.g. the words “any” and “how” can make “anyplace”, “anyhow”, or “anybody”
- *Build a small collect of subword chunks that can cover most words in your dataset without needing to learn all the words in that vocabulary*
- Common words tokenized as whole words (the, at, and)
- Rarer words tokenized into smaller chunks
- Algorithms: BPE (GPT), WordPiece (BERT), SentencePiece (USE)

Read more

Vocabulary Training

The process of creating a vocabulary from the tokens generated in previous phase.

We can do this manually and automatically using high-level libraries like HuggingFace Tokenizers.



Language Modeling

The product of the probability of the words/sentence given their context.

- We leverage large-scale unlabeled dataset
- We train a model to learn $P_{\theta}(text)$ or $P_{\theta}(text \mid some\ other\ text)$
- The outputs of the model are then used to further train a downstream task:
  - Just the output vectors (Shallow approach)
  - Model + output vectors (Deep approach)

Language Modeling (Shallow to Deep)

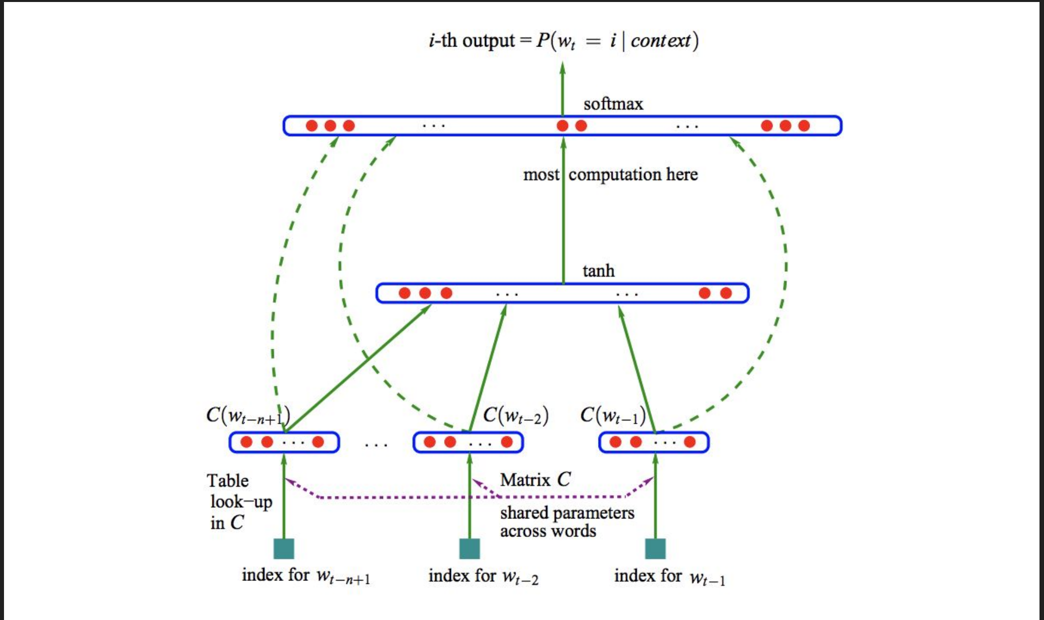
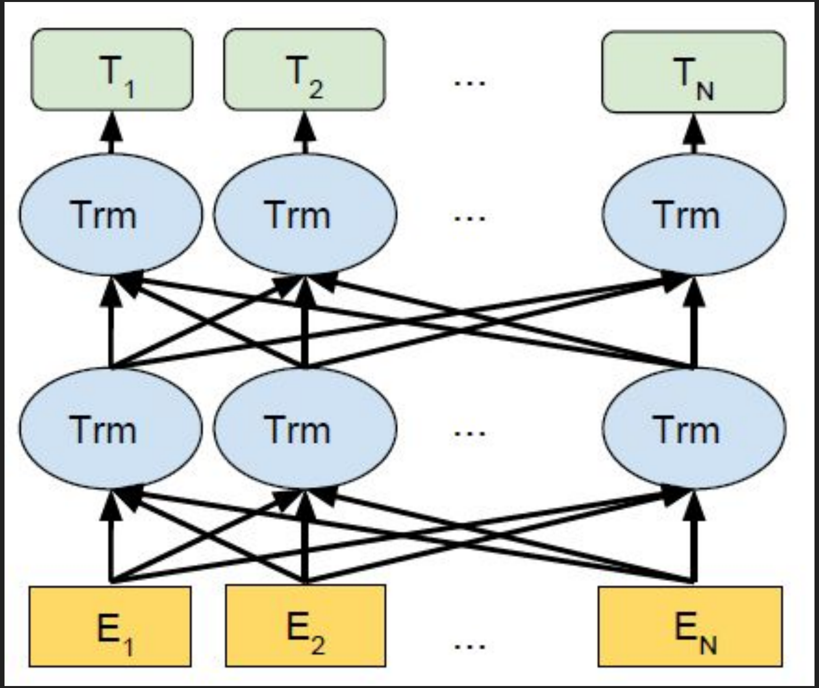


Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Bengio et al. 2003

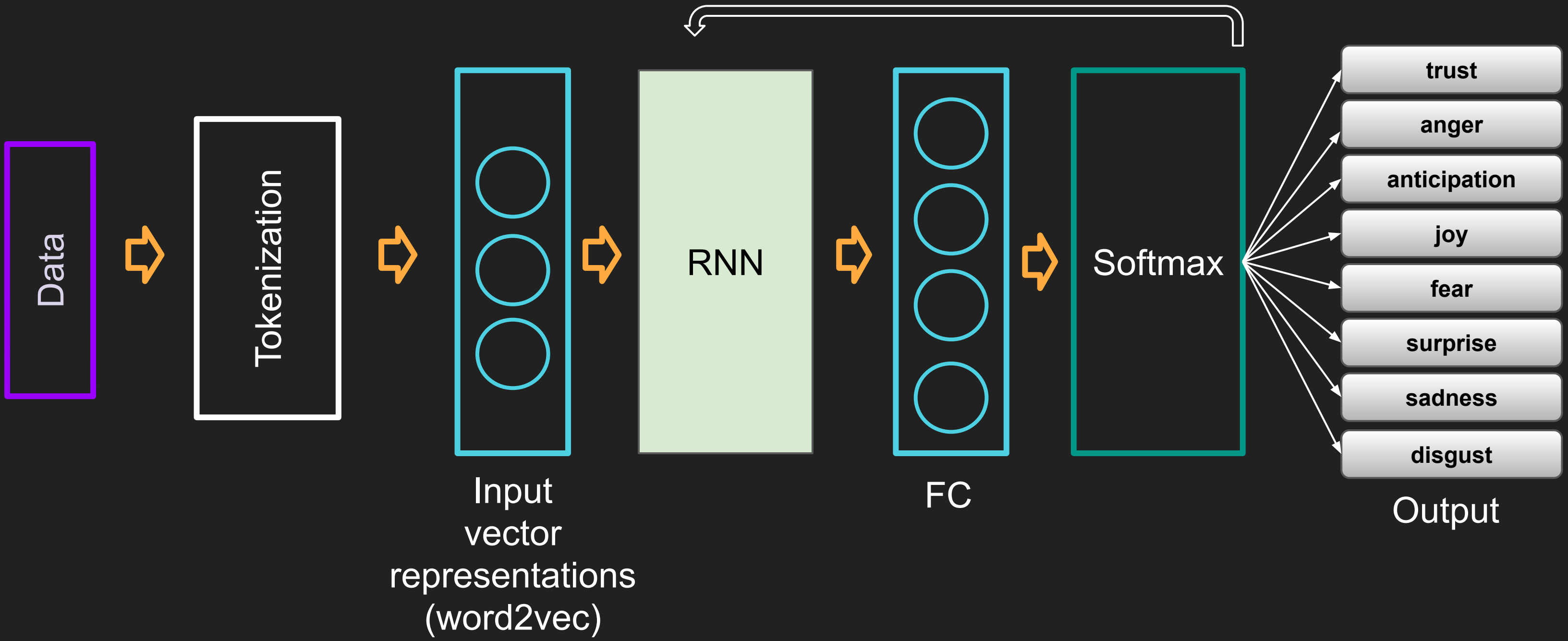


BERT - Devlin et al. 2019

Demo 1

https://github.com/dair-ai/odsc_2020_nlp

Exercise: Emotion Classifier



Training and fine-tuning NLP models

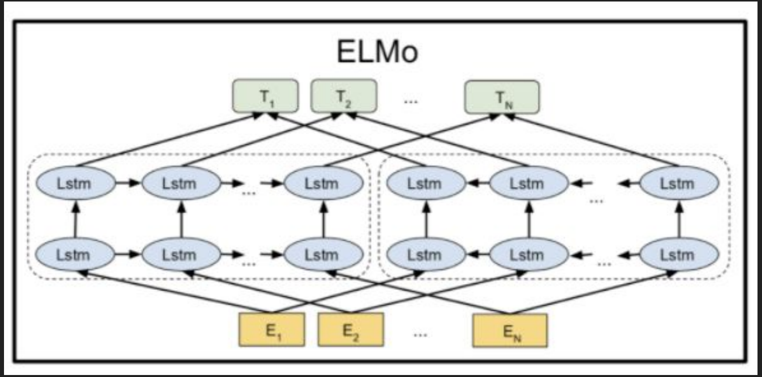
Part 2

Deep Learning for NLP

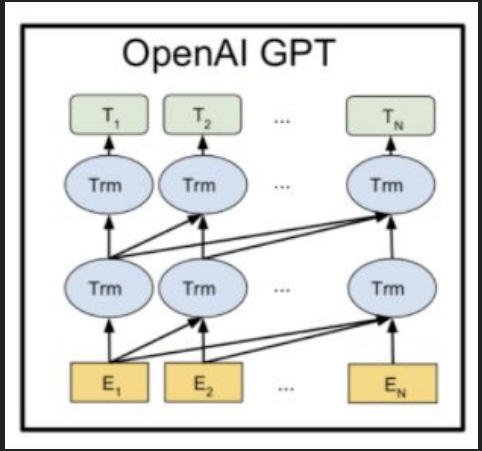
Brief History:

- Shallow: Train and share word embeddings
- Train a model with RNNs or CNNs; use word embeddings as input

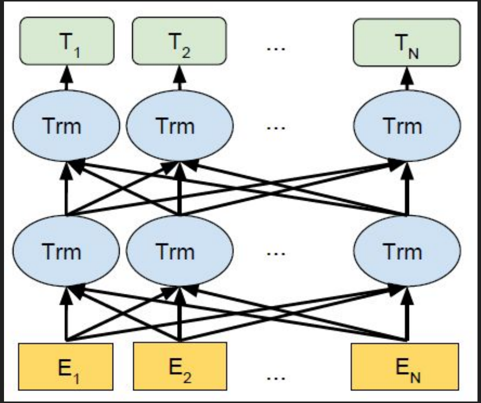
- Deeper: Training & sharing entire language models (ImageNet moment?)
- Fine-tune the models on downstream task (e.g. emotion classification, question-answering, etc.)



- ELMo: Bidirectional multilayer LSTMs
- Outputs contextualized embeddings
- Feed them to task-specific models



- GPT: multilayer transformer decoder
- Fine-tune base model
- Use last layer output of last token
- Add one new trainable weight matrix to predict distribution over class labels



BERT: multilayer transformer decoder
Fine-tune base model
Use final hidden state of first token [CLS] and multiply by small weight matrix to get a prediction

Read More

Demo 2

https://github.com/dair-ai/odsc_2020_nlp

Exercise: Emotion Classifier based on Pretrained models

Task 1: Train emotion classifier by fine-tuning a BERT-based model.

- Download the pretrained model
- Fine-tune it on the downstream task (emotion classification)
- Build a classification model
- Use advanced deep learning tools like Transformers and PyTorch

Towards building real-world NLP-powered applications

Part 3

NLP Applications

All sorts of interesting applications are possible with NLP technology:

- Healthcare
- Finance
- Improving Ads (personalization)
- Improving recommendation systems
- Smart search
- ...

Smart Search

We want to leverage language models and their outputs to be able to render representations that help us to build a semantic search engine to improve document exploration.

We are going to use Elasticsearch to index the vector representations (sentence embeddings) and leverage the search capabilities of Elasticsearch for relevance ranking.

We should be able to retrieve similar phrases or sentences (e.g. questions, abstracts, etc.)

Sentence Embeddings

Embeddings are vector representations that models the semantic meaning of text.

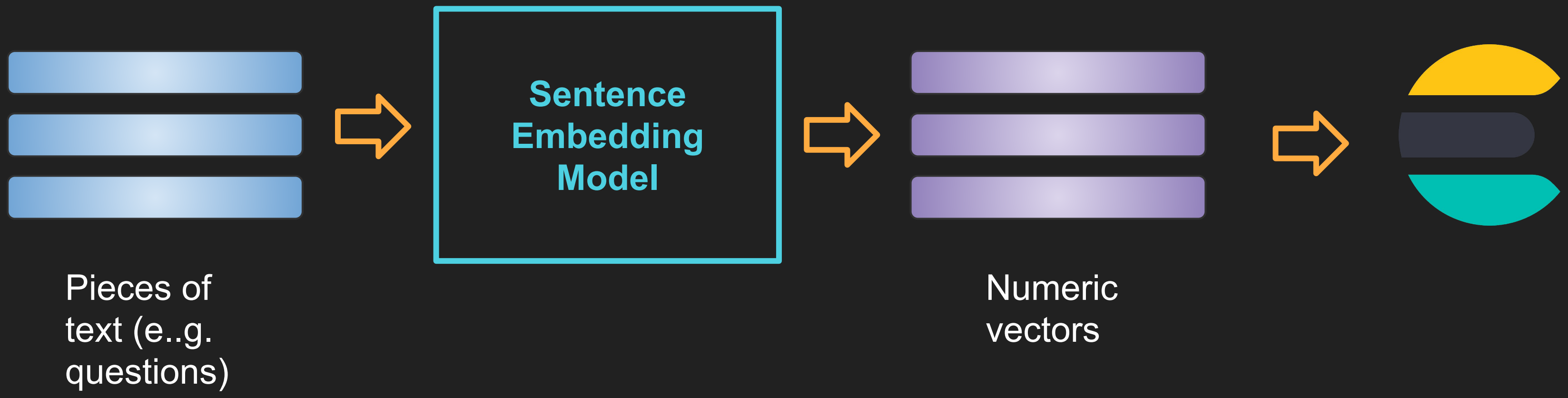
Sentence embeddings can take word order into account when learning vector representations (“tune in” vs. “in tune”)

Limited to short paragraphs; don’t generalize well to very large text

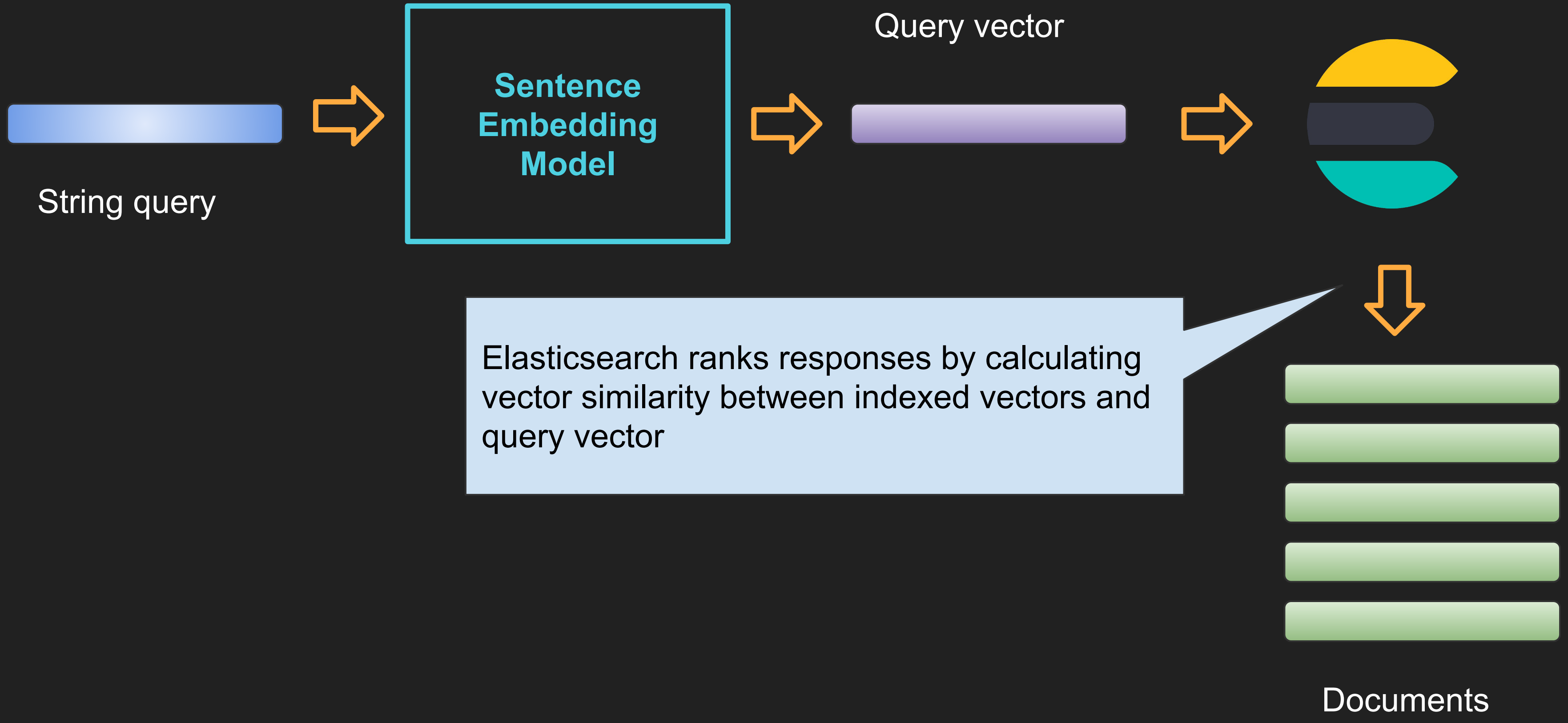
Popular methods:

- InferSent
- Universal Sentence Encoder
- ELMo
- BERT and its variants

Framework (Indexing/Storing)



Framework (Querying)



Elasticsearch ranks responses by calculating vector similarity between indexed vectors and query vector

Demo 3

https://github.com/dair-ai/odsc_2020_nlp

Next Steps

- Experiment more with the search application and try to do some fine-tuning to help improve the outputs of the LMs and improve search

- Try other domain-specific LMs such as BioBERT and SciBERT

- Build a small web application and provide an online demo for others to experiment with.

References

- Text similarity search in Elasticsearch using vector fields
- このスクリプト
- COVID-19 Open Research Dataset Challenge (CORD-19)
- Transfer Learning in Natural Language Processing