FT FINANCIAL TIMES

Data Science in Media

Creating and assessing media article embeddings

Krum Arnaudov, Dec 2022

Agenda

1. Data Science at the FT and Project details
2. Theory overview
3. Setup and experiments
4. Summary of takeaways
5. Resources

FT

Self-intro

- Joined FT as Data Scientist in 2021
- Past jobs as DS at Amplify Analytics, before that Account & Ops Management at HPI
- ML knowledge largely self-taught via free online content + the AI specialisation at the SoftUni.
- Main hobbies:

< Picture of two kids>

Data Science at the FT

- London (5) + Sofia (3) + 2 ML Engineers
  - Diverse team
  - Diverse ML projects

- Main stack:
  - R + Python
  - BigQuery (but moving towards AWS)
  - Rstudio Connect + some AWS

- Big fellow data teams:
  - BI + Analytics folks (quite experienced)
  - Data Platform folks (quite experienced)

FT

Article Vectorisation - Project & Goals

Team:

- **A Lead DS**
- **3 DS**
- **1 ML Engineer**

- Collaboration between Data Science and Content Analytics
- Current methods - Doc2Vec for DS and Transformer for CA.
- Current use cases:
  - Breadth of Reading metric - measures how broadly a given user reads
  - Article Clustering model - assigns newly published articles to a ‘cluster’ with articles of similar content
  - Article Recommendation model - for a given user, suggests content in a ‘nearby’ cluster to what they usually read
  - Trending Topics - uses cluster ids from the Article Clustering model
  - Classification - predicts whether newly published articles are associated with a set of topics

Goal:

Unify and improve the existing vectorisation methods used by the respective teams. The resulting vectors need to be applicable for both clustering and classification, and potential further uses.

FT

AV Project Progress

Setup & Model Selection

Phase 1
Setting up Docker and Visual Studio code

Phase 2
Research the top methods: TF-IDF, SentenceTransformers, Doc2Vec, spaCy etc.

Phase 3
Set up a training dataset of articles

Phase 4
Build Python APIs for each method and produce vectors for the training dataset

Phase 5
Use different methods to evaluate the new vectors compared with the old vectors: similarity, clustering, classification

Phase 6
Choose winner - Congratulations all-MiniLM-L12-v2!!!😄

Model Evaluation & API Testing

Phase 7a
Deploy interim solution to the Content Analytics server, with DynamoDB storage

Phase 7b
Continue vector evaluation by plugging in new vectors into existing Data Science models

Downstream Tasks

- Article Clustering
- Article Recommendation
- CA Classifier
- Breadth of Readership
- Trending Topics

Final Model Deployment

Vectoriser API Final Deployment 🎉🎉🎉🎉🎉🎉🎉

Re-Deploy Models

- Article Recommendation
- Article Clustering
- Trending Topics
- Breadth of Readership
- CA Classifier

FT

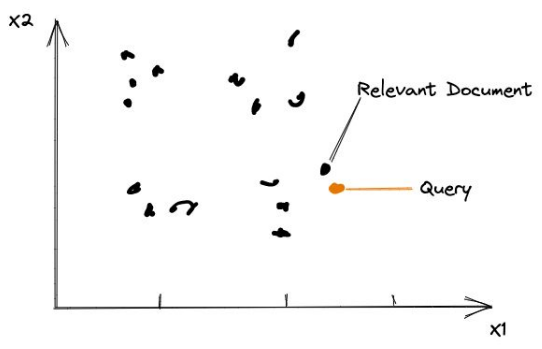
Long (ish) text vectorisation - key takeaways

- Plenty of great options to try:
  - ***SentenceTransformers*** (many, many submodels) - https://www.sbert.net/docs/pretrained_models.html
  - ***Doc2Vec*** - https://radimrehurek.com/gensim/models/doc2vec.html
  - ***Pooled Word Embeddings***
  - ***TF-IDF*** (seriously, oldie-but-goldie)
- ...But not all work well with long texts
- The text structure of FT articles makes this problem easier to solve
- No single model/solution that is best for all downstream tasks.
  - But SentenceTransformers are pretty good at all tasks
- Assess the options as you plan to use them.
- Follow the industry experts for the latest and greatest.
  - Nils Reimers for Sentence Transformers
  - Vincent Warmerdam for, well, everything practical with ML

FT

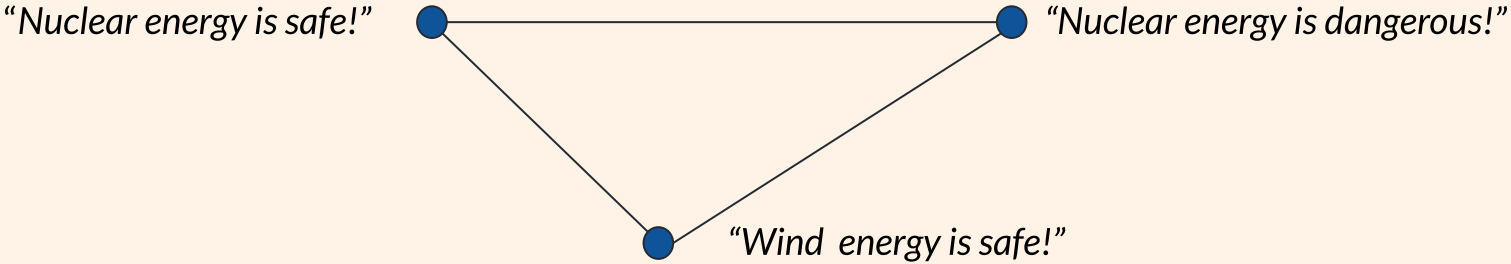
Research time

Task - Find numerical representation of the FT articles such that semantically similar articles are close.



What does semantically similar mean?

- No ***universal*** numerical text representation

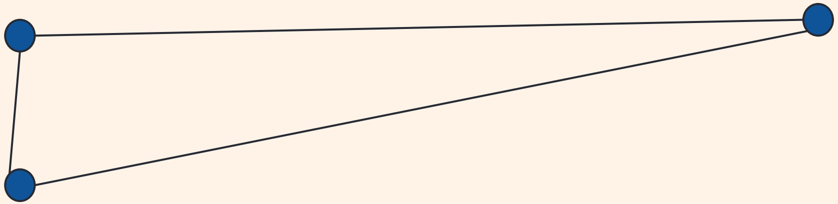


Semantically similar depends on the task!

What does semantically similar mean?

- FT context:

"UK and France reach truce over fishing licence dispute"



"Theatre and the art of keeping the show on the road"

"UK talks with France fail to solve post-Brexit fishing dispute"

Map similar articles close together and pull dissimilar articles further apart.

Article vectorisation - NLP context

- Long texts vectorisation - challenges:
    - Multiple topics per document
    - Transformers token restriction - usually 512 tokens - ca. 400 words in English
    - TF-IDF able to capture specifics, but vocabulary grows enormous
    - Doc2Vec - worse, the longer the text

- FT articles - making it easier:
    - One topic per article.
    - Nicely structured, no typos, in English
    - Descriptive title and subtitle.
    - Frequently using past tense - same as much of the Transformers training data (think Wikipedia).
    - Usually, punchline at the beginning.

Article vectorisation - the contenders

- TF-IDF
- Pooled word embeddings
- Doc2vec (Paragraph-to-vector)
- SentenceTransformers (various options)

TF-IDF Intuition

- Per document:
  - Upscore words that are rather unique
  - Discount words that appear in all other docs

TF

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

IDF

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

TF-IDF

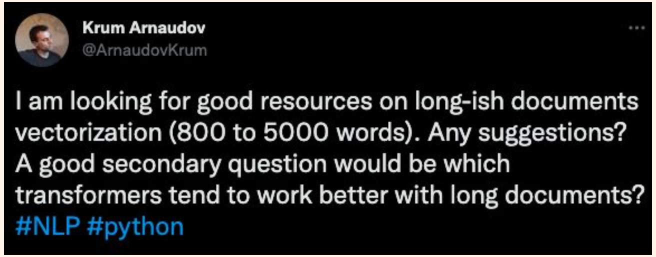
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Speech and Language Processing, D.Jurafsky, J.H.Martin

FT

TF-IDF Outlook

- PROs:
  - Fast to train, fast inference
  - Very interpretable
  - Covers the whole document
  - Order matters less with long documents
- CONs:
  - Requires a “vocabulary” of size (n_training docs, nr_tokens_retained) in memory e.g. (50 000, 100 000)
  - Huge vectors - tough for clustering tasks



(Pooled) Word Embeddings - Intuition

~~Deep~~ Shallow Learning Network

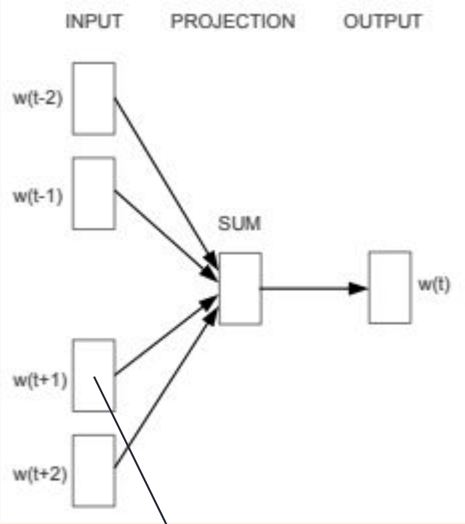
"He was an old man who fished alone in a skiff in the Gulf Stream and he had gone eighty-four days now without taking a fish."

He

was

old

man



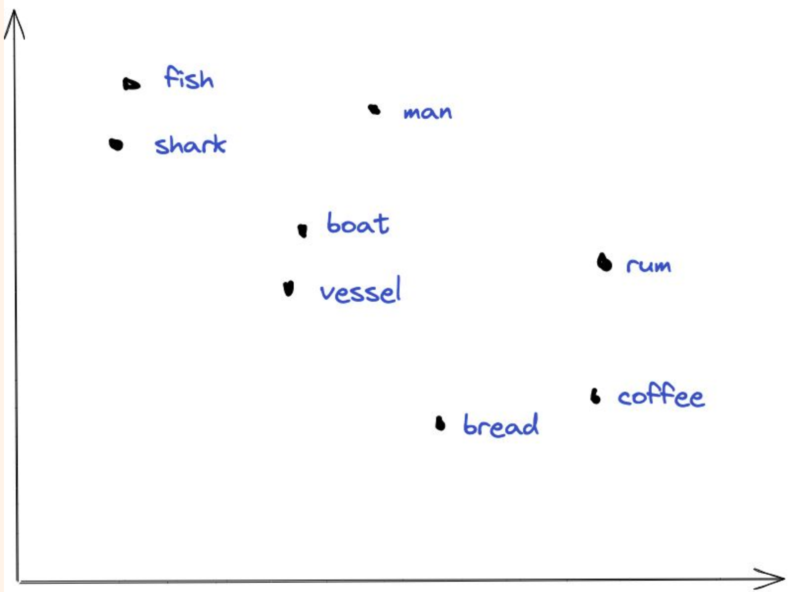
0
..
0.8 *an*
..
0
0
0.03 *the*
...
0.01
..
..
..

word embeddings

Mikolov et al, 2013

(Pooled) Word Embeddings - Intuition

$$\text{SHARK} = \begin{bmatrix} 0.28 \\ 0.79 \\ -0.17 \\ -0.10 \\ 0.10 \\ -0.54 \\ 0.34 \\ -0.19 \\ -0.43 \\ 0.26 \\ 0.01 \end{bmatrix} \qquad \text{FISH} = \begin{bmatrix} 0.14 \\ 0.54 \\ -0.25 \\ -0.12 \\ 0.16 \\ 0.43 \\ 0.34 \\ -0.21 \\ -0.32 \\ 0.23 \\ 0.06 \end{bmatrix}$$



FT

(Pooled) Word Embeddings - Intuition

- Creating a supervised method for an unsupervised learning algorithm.
  - You don't need labels
  - Guess the target word from the words before or after (CBOW) OR
  - Guess the words before and after based on the target word (Skip Gram)
- Think of the embedding layer as a dictionary - the keys are the words, the values are learned in the process
- The goal is the dictionary, not the prediction accuracy

The pooling just means a function to summarise the word embeddings of a document and map it to the same dimensions. You can:

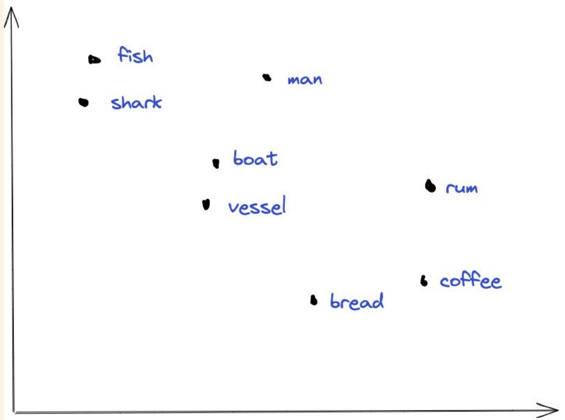
- Mean, Max, Min…
- Idea (not tried) - Weight by some TF-IDFy weights per document

FT

(Pooled) Word Embeddings - Outlook

- PROs:
  - Super fast
  - Good baseline

- CONs:
  - Weak theory for long documents (but good one for title + summary).
  - If you train yourself, too contextualised



FT

Doc2vec - Intuition

~~Deep~~ Shallow Learning Network

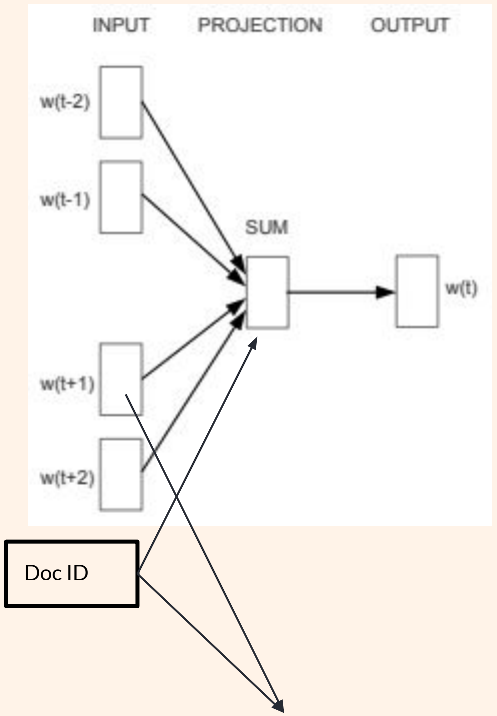
He

was

old

man

The Old Man and The sea



0
..
0.8 *an*
..
0
0
0.03 *the*
...
0.01
..
..
..

Word & Doc embeddings

Mikolov et al, 2014

Doc2vec - Outlook

- PROs:
  - Relatively fast to train/transform
  - Specific to the dataset
  - Built for document embeddings
  - Embedding size can be adjusted based on needs

- CONs:
  - Many parameters to tune
  - The gensim implementation has a (non-insignificant) stochastic element to it >> different vectors for the same params
  - Not transfer-learnable

Transformers

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

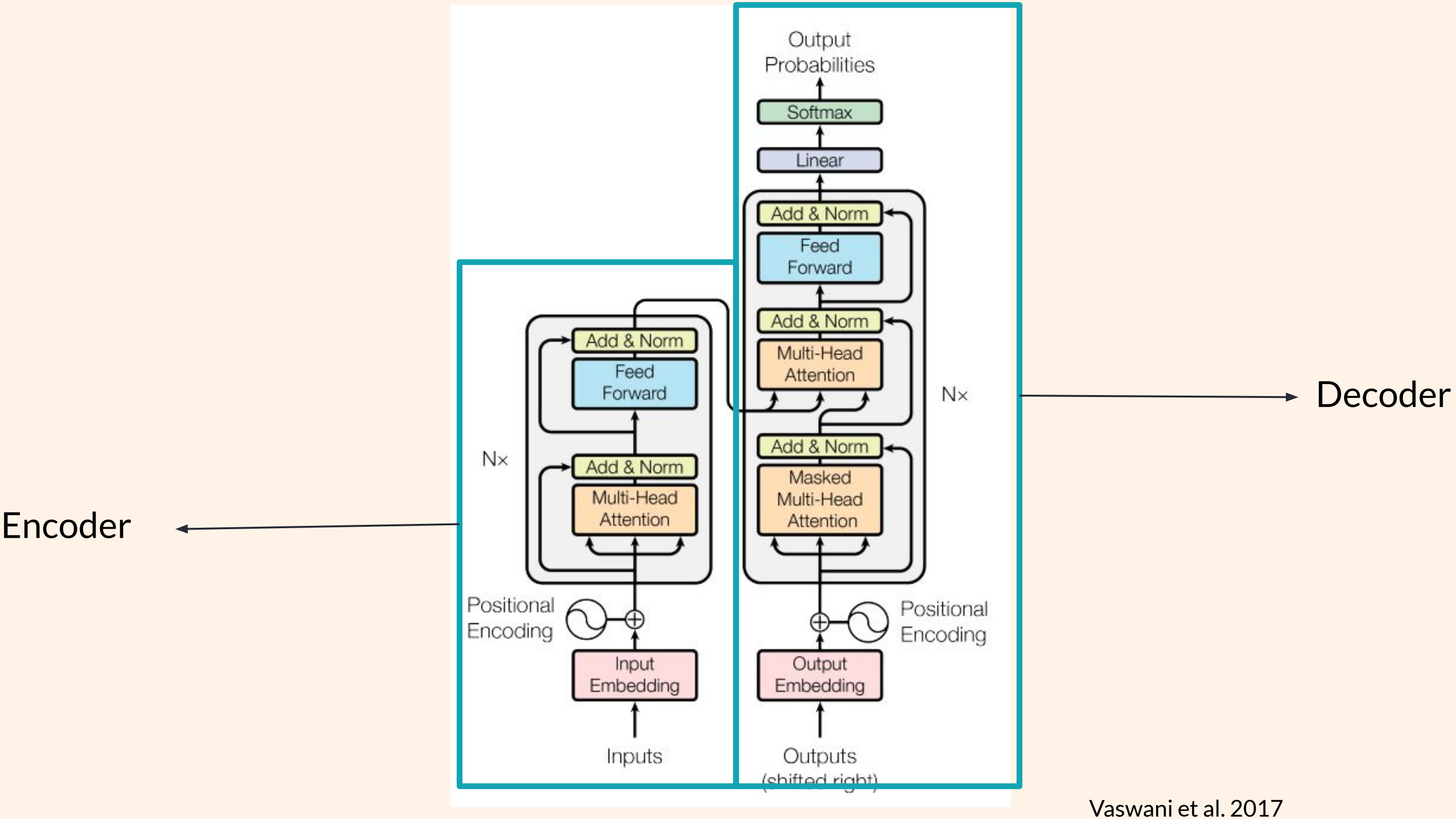
Aidan N. Gomez[*, †]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*, ‡]
illia.polosukhin@gmail.com

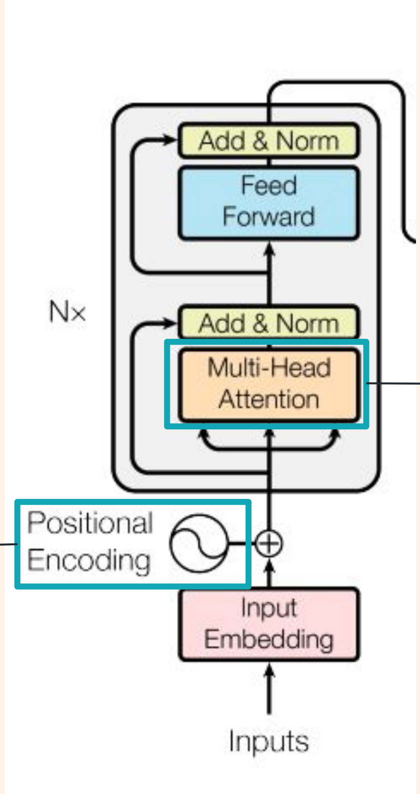
Vaswani et al. 2017

Transformers



Vaswani et al. 2017

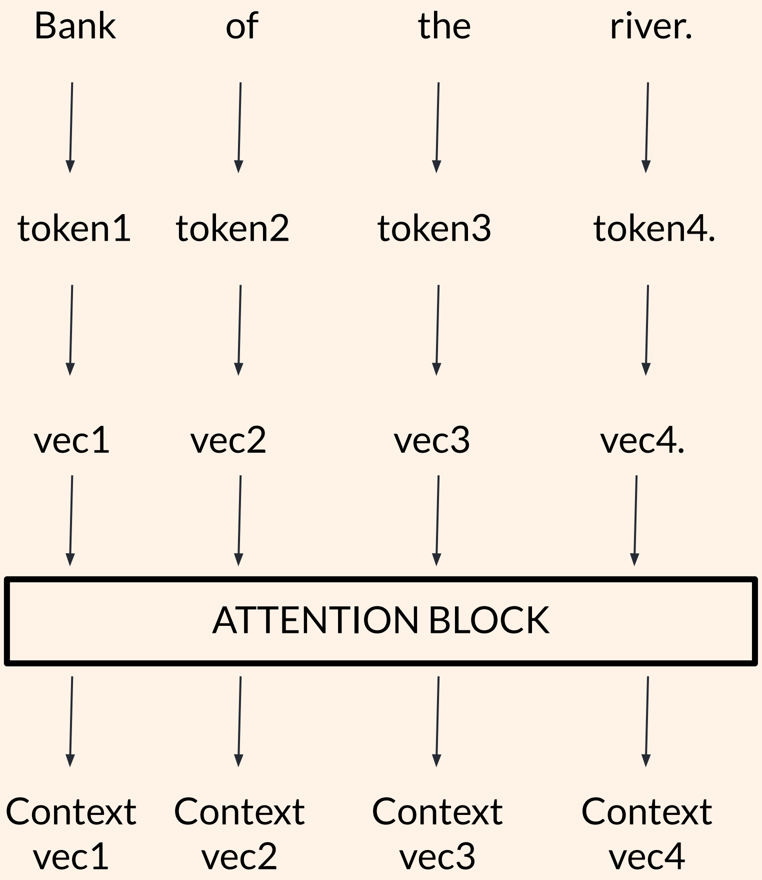
Transformers



Attention >> Self-Attention (for Encoders) >> Multi-Head Attention

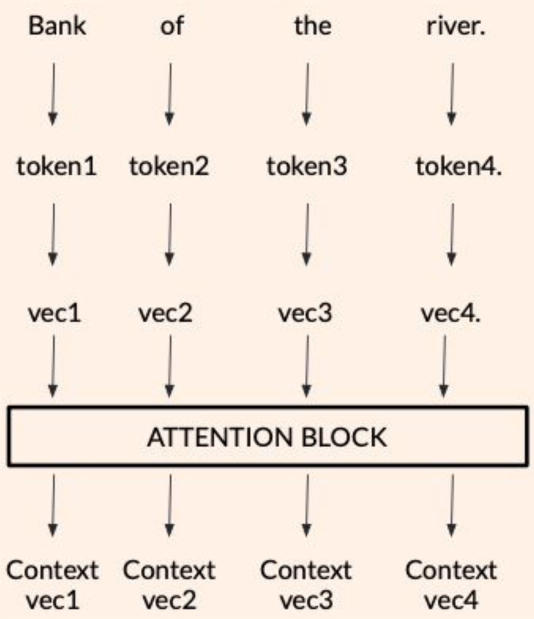
Positional Encoding

Transformers - Self-Attention



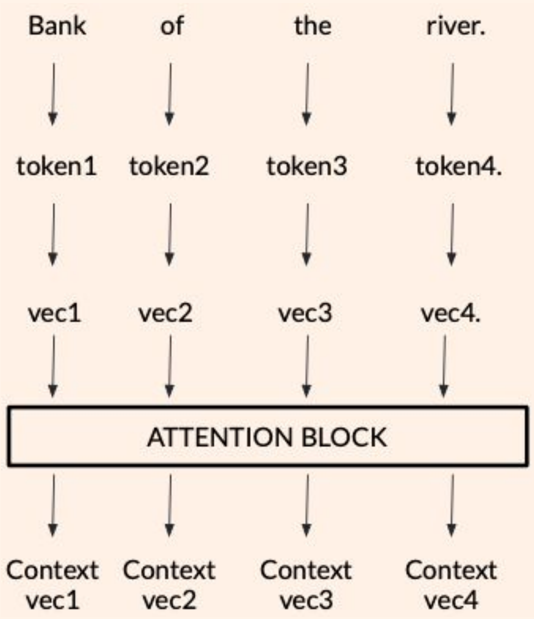
Vincent Warmerdam, 2021

Transformers - Self-Attention




```
vectors = [vec1, vec2, vec3, vec4]
for query in vectors:

    # Similarity - dot product of query with the rest
    dot_product_scores = dot_product(query, vectors)

    # Normalise the scores  so that they sum to one
    normalised_scores = normalise(dot_product_scores)

    # Weight original vectors
    weighted_vectors = normalised_scores*vectors

    # Context vector for query word i
    # weighted average of the original vectors
    context_vec = sum(weighted_vectors)
```

Transformers - Self-Attention

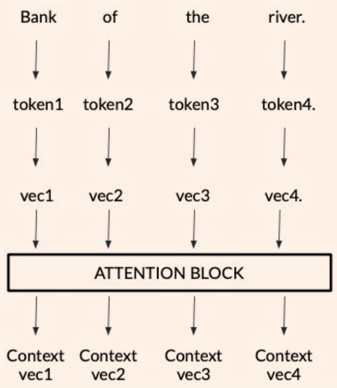



```
vectors = [vec1, vec2, vec3, vec4]
for query in vectors:

        # 1xk vector times kxk matrix
        updated_query = query*query_matrix
        updated_keys= vectors*key_matrix
        updated_values= vectors*value_matrix

        # dot product of query with the rest
        dot_product_scores = dot_product(updated_query, updated_keys)

        # normalise the scores so that they sum to one
        normalised_scores = normalise(dot_product_scores)

        # weight original vectors
        weighted_vectors = normalised_scores*updated_values

        # Context vector for query word i
        # weighted average of the original vectors
        context_vec = sum(weighted_vectors)
```

Transformers - Multi Head Attention

“I gave my dog Charlie some food.”

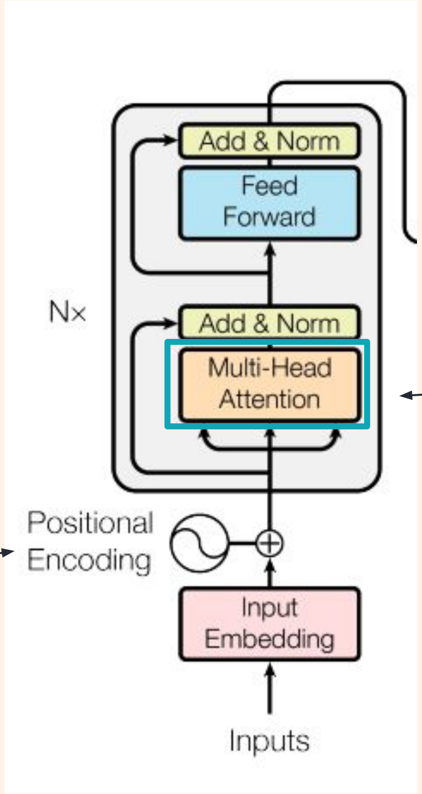
- Have several attention layers
  - At the end, concat and pass through dense, so that the shape aligns with the shape of the input.
- Lets network learn multiple different semantic meanings of attention:
  - E.g. one for grammar, one for vocabulary, etc…




```
vectors = [vec1, vec2, vec3, vec4]
for query in vectors:

    # 1xk vector times kxk matrix
    updated_query = query*query_matrix
    updated_keys= vectors*key_matrix
    updated_values= vectors*value_matrix

    # dot product of query with the rest
    dot_product_scores = dot_product(updated_query,
updated_keys)

    # normalise the scores so that they sum to one
    normalised_scores = normalise(dot_product_scores)

    # weight original vectors
    weighted_vectors = normalised_scores*updated_values

    # Context vector for query word i
    # weighted average of the original vectors
    context_vec = sum(weighted_vectors)
```

Vincent Warmerdam, 2021

Transformers - Encoder



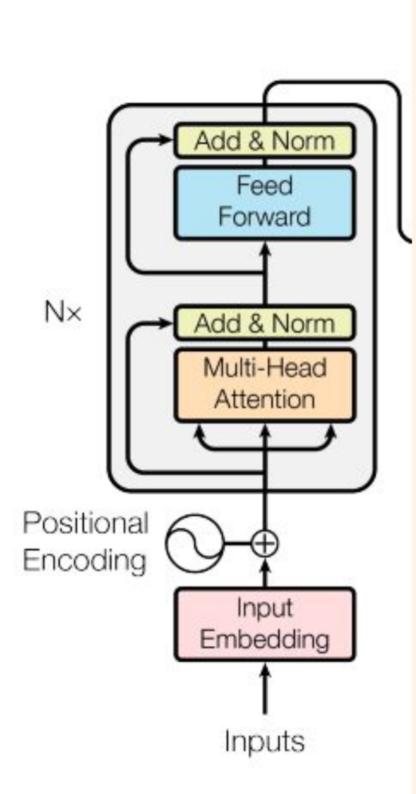
Attention does not care about order.
Good? Bad?

Push the encoder to care about position.

FT

Transformers - Positional encoding

- Set of vectors, encoding the position of each word in the input sentence.
- Same dimensions as the word vectors and are added element-wise to the word vectors to create the final input representation for each word.



Article vectorisation - Why not BERT?

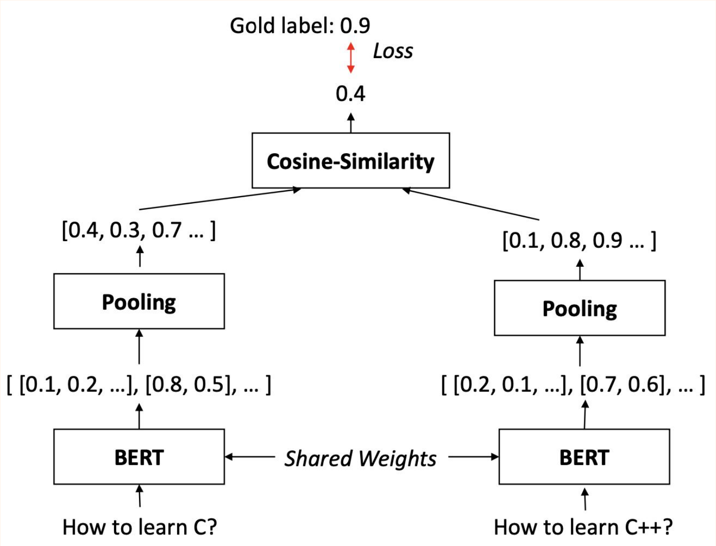
- No concept of “document” embeddings, just better word embeddings:
  - Need to either pool all embeddings OR
  - Use the “CLS” (classification token)
  - Neither work well for clustering, semantic search, etc…
    - Although, they work well for classification!
- Very slow at inference time

Solution - Sentence Transformers

FT

SentenceTransformers - Intuition

- Transformers built specifically for good text embeddings
- Siamese architecture originally - training specifically for semantic similarity



Nils Reimers, Iryna Gurevych, Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

SentenceTransformers - Loss Functions

- Specific loss functions:
  - Constrative loss - pull together positive pairs in the vector space, push away negative pairs in the vector space.
- Triplet loss - anchor, positive and negative (3 things).
  - Results in three spaces - hard negatives, semi-hard negatives (within a margin) and easy negative.
  - Requires good triples (if trained on obvious easy negatives, not much learning is done).
- Multiple Negative Ranking Loss with hard negatives:
  - Train with tuples - (a1, p1, n1) where n1 should be similar to p1 but not match a1
  - Then push the positive towards the anchor and push the negative and ALL the other positives away from the anchor.

  a: How many people live in London?
  p: Around 9 milion people live in London
  n: Around 1 milion people live in Birmingham, second to London

  -

SentenceTransformers - Loss Functions

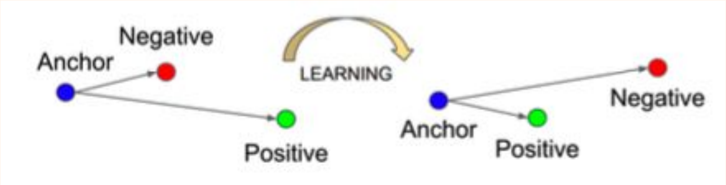
Datasetstructures to train your SentenceTransformers model		
Datasetstructure	Example datasets(repo id in Hugging Face Hub)	Loss functions(imported from sentence_transformers)
Pair of sentences and a label indicating how similar they are	snli	ContrastiveLoss; SoftmaxLoss; CosineSimilarityLoss
Pair of positive (similar) sentences without a label	embedding-data/flickr30k_captions_quintets; embedding-data/coco_captions_quintets	MultipleNegativesRankingLoss; MegaBatchMarginLoss
Single sentence with an integer label	trec; yahoo_answers_topics	BatchHardTripletLoss; BatchAllTripletLoss; BatchHardSoftMarginTripletLoss; BatchSemiHardTripletLoss
Triplet (anchor, positive, negative) sentences	sembedding-data/QQP_triplets	TripletLoss

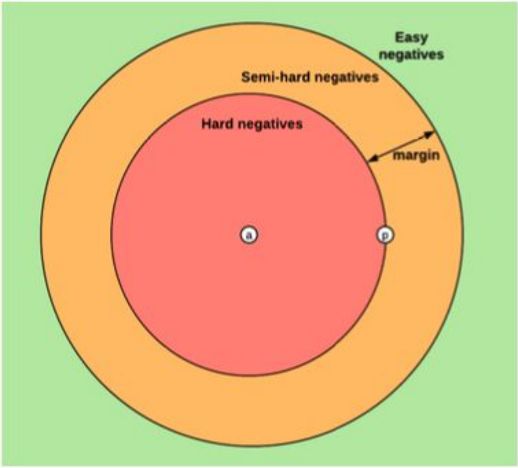
https://huggingface.co/blog/how-to-train-sentence-transformers

FT

SentenceTransformers - Triplet loss example

```
{'set':
 {'query': 'What can I do to get better grades?',
  'pos':
['How do I improve my grades?'],
  'neg':
['Why do I get bad grades even though I study a lot?',
   'How can I get better grades in maths?',
   'How serious is forging high school grades?']
}}
```





SentenceTransformers - Choices

All models

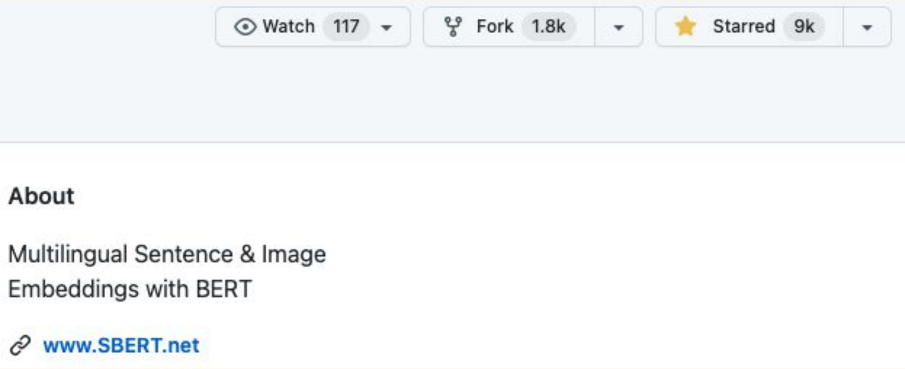
Model Name	Performance Sentence Embeddings (14 Datasets)	Performance Semantic Search (6 Datasets)	Avg. Performance	Speed	Model Size
all-mpnet-base-v2	69.57	57.02	63.30	2800	420 MB
multi-qa-mpnet-base-dot-v1	66.76	57.60	62.18	2800	420 MB
all-distilroberta-v1	68.73	50.94	59.84	4000	290 MB
all-MiniLM-L12-v2	68.70	50.82	59.76	7500	120 MB
multi-qa-distilbert-cos-v1	65.98	52.83	59.41	4000	250 MB
all-MiniLM-L6-v2	68.06	49.54	58.80	14200	80 MB
multi-qa-MiniLM-L6-cos-v1	64.33	51.83	58.08	14200	80 MB
paraphrase-multilingual-mpnet-base-v2	65.83	41.68	53.75	2500	970 MB
paraphrase-albert-small-v2	64.46	40.04	52.25	5000	43 MB
paraphrase-multilingual-MiniLM-L12-v2	64.25	39.19	51.72	7500	420 MB
paraphrase-MiniLM-L3-v2	62.29	39.19	50.74	19000	61 MB
distiluse-base-multilingual-cased-v1	61.30	29.87	45.59	4000	480 MB
distiluse-base-multilingual-cased-v2	60.18	27.35	43.77	4000	480 MB

FT

Sentence Transformers Outlook

- PROs:
    - Built for the task
    - State-of-the-art, well-researched
    - Great transfer learning options per task
    - OK latency

- CONs:
    - Only up-to 512 tokens (400 words)
    - Slow-ish on CPU



Code Time

Project setup

Project setup

- VSCode + Remote Containers
- Data + Models + Experiments - on AWS S3
  - Access via the pins package

- Different APIs
  - Every library, different input/output API, easy to get wrong
  - Solution: write sklearn API wrappers & build a package

Seriously, gensim?

```
X_transformed = [
    gensim.models.doc2vec.TaggedDocument(
        gensim.utils.simple_preprocess(article), [i]
    )
    for i, article in enumerate(X)
]

doc2vec_model = gensim.models.doc2vec.Doc2Vec(
    vector_size=self.vector_size,
    min_count=self.min_count,
    epochs=self.epochs,
    dm=self.dm,
    **self.other_gensim_args
)

doc2vec_model.build_vocab(X_transformed)
doc2vec_model.train(
    X_transformed,
    total_examples=doc2vec_model.corpus_count,
    epochs=doc2vec_model.epochs,
)
```

Code Time

Experiments and assessment

Candidates

- Baseline - GloVe 840B 300 dimensions via the Sentence Transformers
- TF-IDF - via scikit-learn:
  - Try uni- vs. bigram
  - max_df
- Doc2Vec via gensim:
  - Embedding dimensions
  - Number of epochs
  - Min_count
- Sentence Transformers:
  - all-MiniLM-L12-v2 - fast, allrounder
  - all-mpnet-base-v2 - best in class, slower
  - multi-qa-mpnet-base-dot-v1 - semantic search expert, slower

FT

Summary of evaluation tasks

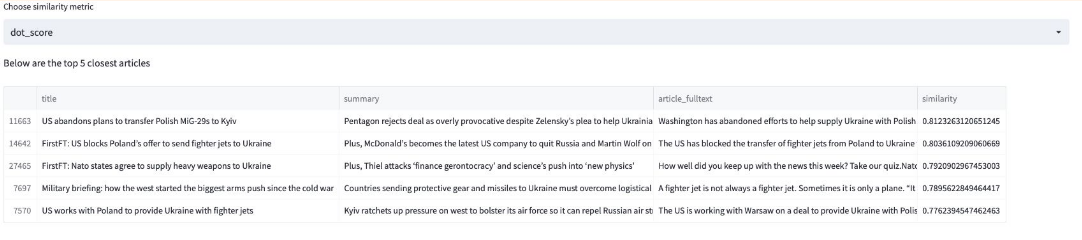
Task	Metric	TF-IDF	Doc2Vec	all-mpnet-base-v2	average_word_embeddings_glove.840B.300d	multi-qa-mpnet-base-dot-v1	all-MiniLM-L12-v2
Similarity dataset	Vector Creation Time	00:00:09	00:00:05	00:02:27	0:00:004	00:02:27	00:01:03
	No. of Rows Cosine(1a-1b) > Cosine(1a-2)	98 / 101	93 / 101	96 / 101	88 / 101	91 / 101	99 / 101
	% Rows Cosine(1a-1b) > Cosine(1a-2)	97.03	92.08	95.05	87.13	90.10	98.02
(ESG) Classification	Precision	0.95	0.94	0.95	0.95	0.95	0.94
	Recall	0.98	0.96	0.97	0.97	0.97	0.97
	F1-score	**0.96**	**0.95**	**0.96**	**0.96**	**0.96**	**0.96**
Sector classification	Precision	0.94	0.93	0.94	0.93	0.94	0.93
	Recall	0.97	0.93	0.96	0.97	0.96	0.96
	F1-score	**0.95**	**0.93**	**0.95**	**0.95**	**0.95**	**0.95**
Corporate Event classification	Precision	0.9	0.9	0.92	0.88	0.93	0.9
	Recall	0.91	0.89	0.9	0.88	0.92	0.89
	F1-score	**0.9**	**0.9**	**0.91**	**0.88**	**0.92**	**0.9**
Clustering & Topic modelling	Number of suggested clusters (bertopic)	NA	403	625	413	591	591
	Number of outlier articles (bertopic)	NA	11295	9836	12892	10261	9753
	Mean cluster size (bertopic)	NA	52	36	47	37	38
	silhouette score (kmeans)	NA	0.030	0.053	0.077	0.033	0.043
Similarity Search	**Qualitative manual observations**		Very good "closest article". Low spread of scores vs. the sentence transformers.	Best in the test "closest article". Great close articles, logical cosine similarity scores.			Very close to all-mpnet-base-v2, more logical score distribution than Doc2Vec

FT

Evaluation - getting the feel of the vectors

- Streamlit dashboards:
  - Similarity search
  - UMAP + Cluestar
  - BERTopic

Try to get a direct feel of method performance.

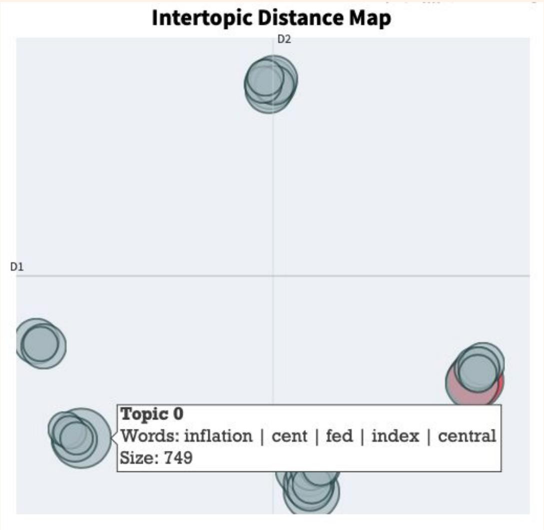


Choose similarity metric

dot_score

Below are the top 5 closest articles

	title	summary	article_fulltext	similarity
11663	US abandons plans to transfer Polish MiG-29s to Kyiv	Pentagon rejects deal as overly provocative despite Zelensky's plea to help Ukrainia	Washington has abandoned efforts to help supply Ukraine with Polish	0.8123263120651245
14642	FirstFT: US blocks Poland's offer to send fighter jets to Ukraine	Plus, McDonald's becomes the latest US company to quit Russia and Martin Wolf on	The US has blocked the transfer of fighter jets from Poland to Ukraine	0.8036109209060669
27465	FirstFT: Nato states agree to supply heavy weapons to Ukraine	Plus, Thiel attacks 'finance gerontocracy' and science's push into 'new physics'	How well did you keep up with the news this week? Take our quiz.Natc	0.7920902967453003
7697	Military briefing: how the west started the biggest arms push since the cold war	Countries sending protective gear and missiles to Ukraine must overcome logistical	A fighter jet is not always a fighter jet. Sometimes it is only a plane. "It	0.7895622849464417
7570	US works with Poland to provide Ukraine with fighter jets	Kyiv ratchets up pressure on west to bolster its air force so it can repel Russian air st	The US is working with Warsaw on a deal to provide Ukraine with Polis	0.7762394547462463



The results!

- All methods are reasonable.
- Sentence Transformers are jack of all trades, master of some.
- Doc2vec - baby ST, good, but with some cons
- TF-IDF - best for classification, but tough for clustering.

The winner:
Sentence Transformer(all-MiniLM-L12-v2)

- Faster and smaller than other STs, similar results

FT

MTEB

- Massive Text Embedding Benchmark (MTEB)

Overall | Bitext Mining | Classification | Clustering | Pair Classification | Retrieval | Reranking | STS | Summarization

Overall MTEB English leaderboard 🔮

- Metric: Various, refer to task tabs
- Languages: English, refer to task tabs for others

Rank	Model	Embedding Dimensions	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)	Pair Classification Average (3 datasets)	Reranking Average (4 datasets)	Retrieval Average (15 datasets)	STS Average (10 datasets)	Summarization Average (1 dataset)
1	sentence-t5-xxl	768	59.51	73.42	43.72	85.06	56.42	42.24	82.63	30.08
2	gtr-t5-xxl	768	58.97	67.41	42.42	86.12	56.66	48.48	78.38	30.64
3	SGPT-5.8B-weightedmean-msmarco-specb-bitfit	4096	58.81	68.13	40.34	82	56.56	50.25	78.1	24.75
4	e5-small	384	58.78	71.67	39.51	85.08	54.45	46.01	80.87	24.94
5	gtr-t5-xl	768	58.42	67.11	41.51	86.13	55.96	47.96	77.8	30.21
6	gtr-t5-large	768	58.28	67.14	41.6	85.32	55.36	47.42	78.19	29.5
7	sentence-t5-xl	768	57.87	72.84	42.34	86.06	54.71	38.47	81.66	29.91
8	all-mpnet-base-v2	768	57.78	65.07	43.69	83.04	59.36	43.81	80.28	27.49

Lessons Learned

- Plenty of great options to try:
  - *SentenceTransformers* (many, many submodels) - https://www.sbert.net/docs/pretrained_models.html
  - *Doc2Vec* - https://radimrehurek.com/gensim/models/doc2vec.html
  - *Pooled Word Embeddings*
  - *TF-IDF* (seriously, oldie-but-goldie)
- ...But not all work well with long texts
- The text structure of FT articles makes this problem easier to solve
- No single model/solution that is best for all downstream tasks.
  - But SentenceTransformers are pretty good at all tasks
- Assess the options as you plan to use them.
- Follow the industry experts for the latest and greatest.
  - Nils Reimers for Sentence Transformers
  - Vincent Warmerdam for, well, everything practical with ML

FT

Key resources

- Vincent Warmerdam's Whiteboard - https://youtube.com/playlist?list=PL75e0qA87dlG-za8eLI6t0_Pbxafk-cxb
  - And a list of his projects - https://github.com/koaning
  - Great short, calm courses - https://calmcode.io/
- Great Free Stanford Textbook - https://web.stanford.edu/~jurafsky/slp3/ed3book_jan122022.pdf
  - With a course - https://web.stanford.edu/class/cs224n/index.html#coursework
- Nils Reimers sites:
  - https://www.nils-reimers.de/ - check the Talks section for video/slides
  - https://www.sbert.net/
- Huggingface blogs on Sentence Transformers:
  - https://huggingface.co/blog/how-to-train-sentence-transformers
  - https://huggingface.co/blog/playlist-generator
- If you'd like to play along:
  - Playground vectoriser package - https://github.com/krumeto/articlevectorizer
  - Notebooks to play around - https://github.com/krumeto/article-vectorisation-eda

Thank you!