TEXT MINING

L07. NEURAL NLP AND TRANSFER LEARNING

SUZAN VERBERNE 2022

Universiteit Leiden

TODAY'S LECTURE

- Quiz about week 6
- Assignment 1
- Neural architectures for sequential data: RNNs and LSTMs
- Transformer models
- BERT and SBERT
- Transfer learning with neural language models

QUIZ ABOUT WEEK 6

- What are the strengths of CRF compared to a HMM for sequence labelling?
  a. It is multi-layered
  b. It can use features to represent tokens
  c. It takes a larger context

QUIZ ABOUT WEEK 6

- What are the strengths of CRF compared to a HMM for sequence labelling?
  a. It is multi-layered
  b. It can use features to represent tokens
  c. It takes a larger context

QUIZ ABOUT WEEK 6

- Why are part-of-speech tags informative for Named Entity Recognition?
  a. Because only nouns can be entities
  b. Because some word categories are more likely to be (part of an) entity
  c. Because the occurrence of subsequent entities in a row

QUIZ ABOUT WEEK 6

- Why are part-of-speech tags informative for Named Entity Recognition?
  a. Because only nouns can be entities
  b. Because some word categories are more likely to be (part of an) entity
  c. Because the occurrence of subsequent entities in a row

QUIZ ABOUT WEEK 6

- Why is distant supervision called distant?
  a. Because it can regonize relations between entities with a longer distance between them
  b. Because the supervision comes from a knowledge base, not a human
  c. Because the supervision is incomplete

QUIZ ABOUT WEEK 6

- Why is distant supervision called distant?
  a. Because it can regonize relations between entities with a longer distance between them
  b. Because the supervision comes from a knowledge base, not a human
  c. Because the supervision is incomplete

QUIZ ABOUT WEEK 6

- We have a text collection manually labelled with 1000 entities. Our BERT model identified 800 entities, 600 of which were also in the manual set. What is the recall?
  a. 800/1000
  b. 600/800
  c. 600/1000

QUIZ ABOUT WEEK 6

- We have a text collection manually labelled with 1000 entities. Our BERT model identified 800 entities, 600 of which were also in the manual set. What is the recall?
  a. 800/1000
  b. 600/800
  c. 600/1000

ASSIGNMENT 1 – TEXT CLASSIFICATION

- Grading (ongoing):
  - 5 criteria; max 2 points per criterion
    1. General: length correct (2-3 pages) and proper writing and formatting
    2. Experiments on 20 newsgroups
    3. Results table for 3 classifiers x 3 feature weights (counts, tf, and tf-idf)
    4. Results for a. lowercase; b. stop_words; c. analyzer (in combination with ngram_range); d. max_features
    5. Brief discussion on which classifier performs the best, with which features

ASSIGNMENT 1 – TEXT CLASSIFICATION

Some notes about your (previous year's) submissions:

- Please mention in your introduction what task you are addressing (text classification in 20 news categories)
- Please report what settings you compared and what the results were
- Please summarize the most important results in a neatly formatted table
  - It is good practice to highlight the best-in-class performances in boldface
- Grid search is a form of optimization and should not be done on the test set (but on a separate development set or in cross validation on the train set)
- *Never* copy text from (web) sources or other students. This is considered plagiarism and will be reported to the Board of Examiners.

EXAMPLE REPORT 1

Text categorisation

Victor Batenburg (s3263630) and Nathan Broughton (s1843850)

18 October 2021

1 Introduction

In this report we will be extending the 'working with text data' tutorial in scikit-learn. The focus will be on comparing three different classifiers on the multi-class classification task for different features. The data set consists of a collection of approximately 20,000 newsgroup documents. The data set is partitioned almost evenly across 20 different newsgroups. The goal of the classifier is to accurately classify each document to the correct newsgroup. The tutorial focuses on only 4 categories, however we will be addressing all 20 categories in the data set.

2 Methods

2.1 Feature spaces

We will compare and monitor the performance of three different feature spaces, word counts, TF and TF-IDF. The first feature space is word counts, which is a simple way of representing the presence of words in documents. When transforming our training data to word counts, we retrieve a matrix that contains the number of occurrences for specific words in each document. This matrix is sparse as many words in the vocabulary will not occur in all documents. The second feature space will be TF, or term frequency. The term frequency is the number of occurrences for a specific term in a document divided by the total number of terms in that documents. The third feature space is the TF-IDF space, which is the earlier mentioned TF multiplied by a weighted IDF. The IDF as calculated by scikit-Learn is defined as $log(\frac{n}{df(t)})+1$ where n is the number of documents, df the document frequency and t a specific term. We will also use smoothing, which means that both the numerator and denominator get an extra addition of 1 in order to prevent zero divisions.

2.2 Classifiers

As a baseline classifier, we will be using multi-class Naive Bayes. This classifier is based on Bayes' rule and estimates probabilities given an observation. We will use scikit-learn's MultiNB function to implement a Naive Bayes model. The second implemented model is a simple stochastic gradient descent classifier, for which we will use the SGDClassifier function. The final model tested is a linear support vector machine, implemented by scikit-learn's LinearSVC function.

3 Results

In order to properly compare the performance of all the models and feature spaces, we trained all combinations and monitored the precision, recall and F1 score on the test set. Based on the results, which can be found in table 1, we can conclude that TF-IDF is the best feature space to pick. In all models, it outperforms the other feature spaces on all metrics. As for the models, the SGD under performs relative to the Linear SVM on all features except TF-IDF, for which it is tied. Both SGD and the Linear SVM outperform the multi-class Naive Bayes on all metrics. The best combination of features and models would be the TF-IDF as feature space and SGD or Linear SVM as model.

We also performed a cross validated grid search on the parameters of CountVectorizer on all models to find the optimal settings. We used a 5-fold cross validation and then explored the CountVectorizer parameters. Their

	MultiNB			**SGD**			**LinearSVM**		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Counts	0.76	0.77	0.75	0.77	0.76	0.76	0.79	0.79	0.78
TF	0.79	0.71	0.69	0.81	0.81	0.80	0.83	0.83	0.82
TF-IDF	0.82	0.77	0.77	0.85	0.85	0.85	0.85	0.85	0.85

Table 1: Model performance on all feature spaces

values can be found in table 2. By building a scikit-learn pipeline, we were able to use the GridSearchCV function to explore all the possible parameter values and monitored the performance.

Parameter	**Possible values**
lowercase	True, False
stop_words	English, None
Analyzer	word, char, char_wb
ngram_range	(1, 1), (1, 2), (2, 2), (1, 3)
max_features	None, 5000, 10000, 50000

Table 2: Grid search parameters

Looking at the results of the cross validated grid search, we can immediately see that the linear SVM outperforms both other models. An interesting fact about the parameter settings is that the multi-class naive Bayes performs best for only bigrams, rather than the combination of unigrams, bigrams and trigrams. Furthermore, only the SGD model performs better when also using uppercase words. Parameters such as having no maximum number of features or using stop words are as expected. Overall, we conclude that the Linear SVM achieves the best performance, but is the most expensive model based on runtime.

Model	MultiNB	SGD	LinearSVM
Score	0.855	0.852	0.899
Analyzer	word	word	word
Lowercase	True	False	True
Max features	None	None	None
Ngram range	(2, 2)	(1, 3)	(1, 3)
Stop words	English	English	English
Runtime (minutes:seconds)	57:25	70:19	294:01

Table 3: Grid search parameters and performance

4 Discussion

After analysing the results of the three classifier we can conclude that the Linear SVM performs the best for our classification task. The performance of this classifier after finding its optimal parameters is 0.899, which is the best score we were able to obtain. However the time constraint of the cross validated grid search is a point which is worth mentioning. When the classification task uses a bigger data set it might be a better idea to use a different classifier. An interesting addition to the research would be to do a cross validated grid search for the TF-IDF parameters and see how much the classifier improves after that, as the TF-IDF feature is the best type of feature for this classification task.

EXAMPLE REPORT 2

Assignment I: Text Categorization

Yingjie Li (s3126161)
Tao Peng (s3076326)

I. Introduction

In this report, we try to perform a text categorization task with scikit-learn. In order to achieve this, we conducted two experiments. In the first experiment, we create and train three different classifiers and evaluate the performance with different metrics. In the second experiment, we modify the parameters of function CountVectorizer to observe the tokenizing result. The remainder of this report is sturctured as follows: Section II describes the data set we used to experiment with; Section III covers the details of three different classifiers; Section IV is about tokenizing with different parameter settings.

II. Experiments Setup

This section contains a description of the experimental setup. The training set and testing set are got from scikit-learn by function *fetch_20newsgroups*. From the description of the development documents, there are around 18000 newsgroups posts on 20 topics split in two subsets: training set and testing set. Instead of getting several samples in our experiment, we choose to load all the 20 newsgroups.

Then, after loading the data set, we transform the collected documents into a sparse matrix of token counts. And use 3 different types of features to represent: counts, tf, tf-idf.
To do this experiment, we choose different classifiers in the next section.
The code is implemented in Python 3.0 and all relevant functions are supported by scikit-learn 1.0.

III. Classifiers

In this experiment, we create three different classifiers, namely Navie Bayes(MultinomialNB), SGD Classifier and KNeighbors. For each classifier, three types of features(counts, tf and tf-idf) extracted from the data set are used as input data to train the model.

The first classifier is MultinomialNB, which is a Naive Bayes classifier for multinomial models. It is suitable for the task of classification with discrete features. Text classification is a typical task with discrete features. The second classifier we want to try is SGD classifier, which is a kind of linear classifiers with stochastic gradient descent (SGD) training. The model is updated with each sample and the learning rate is set with a decreasing strength schedule. The last classifier is KNeighbors classifier, which is based on a lazy learning algorithm and makes no assumption about data.

After training classifiers finished, we measure the performance of them in three metrics, namely F1, Precision and Recall. According to the description of the official documents, the data sets contain different categories evenly and there is no need to consider the weights. Therefore, the "average" parameters for all the three scores are set to "marco".

The main process of the program is described as follows:

- Step 1: Fetch the training and testing data set by function *fetch_20newsgroups*.
- Step 2: Tokenize and extract features from both the training and testing data set, using class *CountVectorizer* and *TfidfTransformer*.
- Step 3: Train the three classifiers with training features generated from Step 2.
- Step 4: Predict with the testing data set and measure the performance.

The results are shown in Table I, Table III and Table II. From these tables we can see that SGDClassifier earns the highest score in most of the tests, which means the best performance for classification. The performance of MultinomialNB is close to SGD Classifier and the KNeighbors Classifier works the worst.

Table I: F1 score on three classifiers with features counts, tf and tf-idf.

	MultinomialNB	SGDClassifier	KNeighborsClassifier
counts	0.745	0.756	0.356
tf	0.672	0.795	0.412
tf-idf	0.775	0.845	0.654

Table II: Recall score on three classifiers with features counts, tf and tf-idf.

	MultinomialNB	SGDClassifier	KNeighborsClassifier
counts	0.762	0.779	0.420
tf	0.792	0.805	0.480
tf-idf	0.825	0.850	0.667

Table III: Precision score on three classifiers with features counts, tf and tf-idf.

	MultinomialNB	SGDClassifier	KNeighborsClassifier
counts	0.763	0.751	0.350
tf	0.682	0.795	0.407
tf-idf	0.756	0.844	0.656

IV. Tokenizing

In this experiment, we investigate the parameters of *CountVectorizer*.
For "lowercase", if set this parameter to "True", all the words would be transformed to lowercase. Otherwise, the content of the documents would not be changed. The default value is "True" and we change it to "False". The shape of the features before and after the changing is (11314, 130107) and (11314, 155448). The number of terms increase a bit.

For "stop_words", it determines which stop words would be remove from the content. The default value of this parameter is "None", which means no stop words would be removed. We set it to "english" and the English stop words are removed. Then, the shape of features change to (11314, 129796). This is because the stop words only take a small proportion of the vocabulary.

For "analyzer", it can be set to three types: "word", "char", "char_wb". The first type means tokenizing with words, while the last two types mean tokenizing with character. In another words, the first type denotes the documents in words, and the others denote documents in character, like "a", "¶" and "ö". The default value is "word". At the same time, we can set the *ngram_range* to decide the maximal and minimal value of n-gram. We test three settings, (analyzer='char'), (analyzer='char_wb') and (analyzer='char_wb', ngram_range=(5,5)) and the shape of features change to (11314,108), (11314,104) and (11314,1638432) respectively. We can see that this parameter have a significant influence on the token matrix.

For "max_features", it means the function will return top max_features ordered by term frequency. The default value is "None" and we set it to 5000. Then, the shape of the features change to (11314,5000), as we expected.

At last, we test all above modifications on the MultinomialNB classifier with the counts feature and the result is shown in Table IV. From this table we can see that remove the stop words can increase the performance of a classifier.

Table IV: Performance of MultinomialNB classifier with different tokenizing parameter settings.

	F1	Recall	Precision
default	0.745	0.762	0.763
lowercase=False	0.739	0.758	0.758
stop_words='english'	0.778	0.813	0.794
analyzer='char'	0.151	0.170	0.163
analyzer='char_wb'	0.149	0.170	0.162
analyzer='char', ngram_range=(5, 5)	0.686	0.760	0.702
max_features=5000	0.569	0.581	0.581

V. Conclusion

From the results above, we can see that, in general, tf-idf performs the best on different classifiers. Tf performs a little better on naive bays and KNeighbors classifier on F1 score and precision than counts, but worse than counts on SGD Classifier for all the metrics. From the aspect of classifers, SGD Classifier is the best classifier of all the three for this text mining task and KNeighbors classifier is the worst one.

EXAMPLE RESULTS TABLES

	Precision	Recall	F1
Naive Bayes - count	0.76	0.76	0.75
Naive Bayes - tf	0.79	0.68	0.67
Naive Bayes - tf-idf	0.83	0.76	0.76
SVM - count	0.77	0.76	0.76
SVM - tf	0.81	0.8	0.8
SVM - tf-idf	**0.85**	**0.84**	**0.84**
Random Forest - count	0.77	0.75	0.75
Random Forest - tf	0.77	0.74	0.74
Random Forest - tf-idf	0.77	0.75	0.75

Classifier	Features	Mean Accuracy	Precision	Recall	F1
K-NN	Counts	0.352	0.421	0.352	0.357
	TF	0.408	0.489	0.408	0.417
	TF-IDF	0.659	0.674	0.659	0.66
Naive Bayes	Counts	0.773	0.762	0.773	0.751
	TF	0.705	0.785	0.705	0.692
	TF-IDF	0.774	0.822	0.774	0.768
SGDC	Counts	0.742	0.767	0.742	0.748
	TF	0.811	0.811	0.811	0.807
	TF-IDF	0.853	0.854	0.853	0.852

We tried different values for various parameters of the count-vectorizer. Results with tf-idf and the SVM I model on the same train-test sets[1] are shown in Table 2. Lowercasing the text and removing stop words has little effect on performance. Restricting the number of features to the n most common seems to have a detrimental effect. Lastly, when using character n-grams instead of word tokens, we observe that using larger ngrams improves scores. However, this is still inferior to using words.

Parameters	Precision	Recall	F1
`lowercase=False`	0.83	0.82	0.82
`lowercase=True`	0.83	0.82	0.82
`stop_words=None`	0.83	0.82	0.82
`stop_words='english'`	0.83	0.82	0.82
`analyzer='char', ngram_range=(1, 2)`	0.60	0.45	0.43
`analyzer='char', ngram_range=(2, 2)`	0.66	0.62	0.62
`analyzer='char_wb', ngram_range=(1, 2)`	0.59	0.34	0.35
`analyzer='char_wb', ngram_range=(2, 2)`	0.65	0.62	0.61
`maxfeatures=100`	0.25	0.22	0.19
`maxfeatures=500`	0.51	0.52	0.50
`max_features=1000`	0.64	0.65	0.63
`max_features=100000`	0.83	0.82	0.82
`max_features=None`	0.83	0.82	0.82

Table 2: Performance of SVM I using tf-idf with various `CountVectorizer` parameters. Any parameters not listed are at the defaults provided by `sklearn`

Universiteit Leiden

sklearn.metrics.classification_report has a parameter ‘digits’ for the number of decimals. General: use 3 digits: 0.763 or 76.3%

NEURAL MODELS FOR SEQUENTIAL DATA

J&M CHAPTER 9

Universiteit Leiden

Suzan Verberne 2022

RECURRENT NEURAL NETWORKS

- RNNs have connections between the hidden layers of subsequent 'time steps' (words in a text)
- RNNs have an internal state that is updated in every time step
- The hidden layer weights determine how the network should make use of past context in calculating the output for the current input
- As with the other weights in the network, these are trained via backpropagation

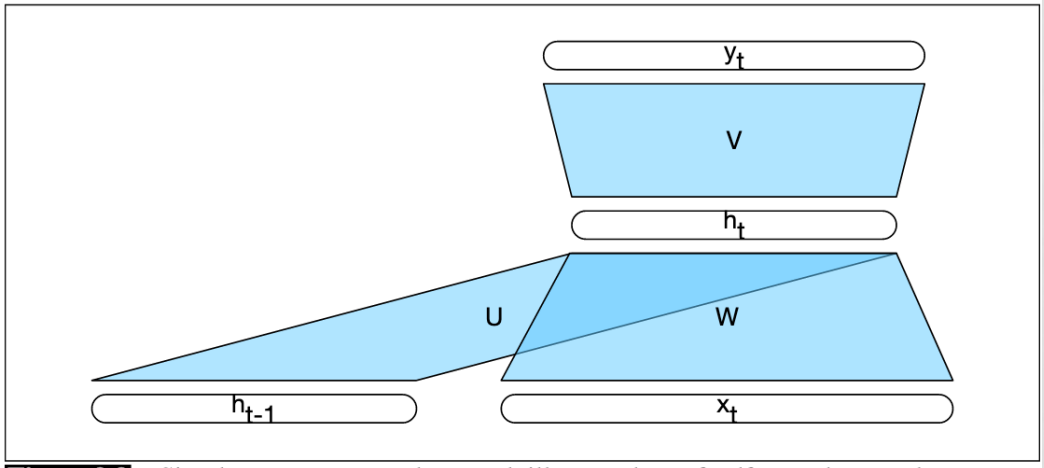


Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

RECURRENT NEURAL NETWORKS

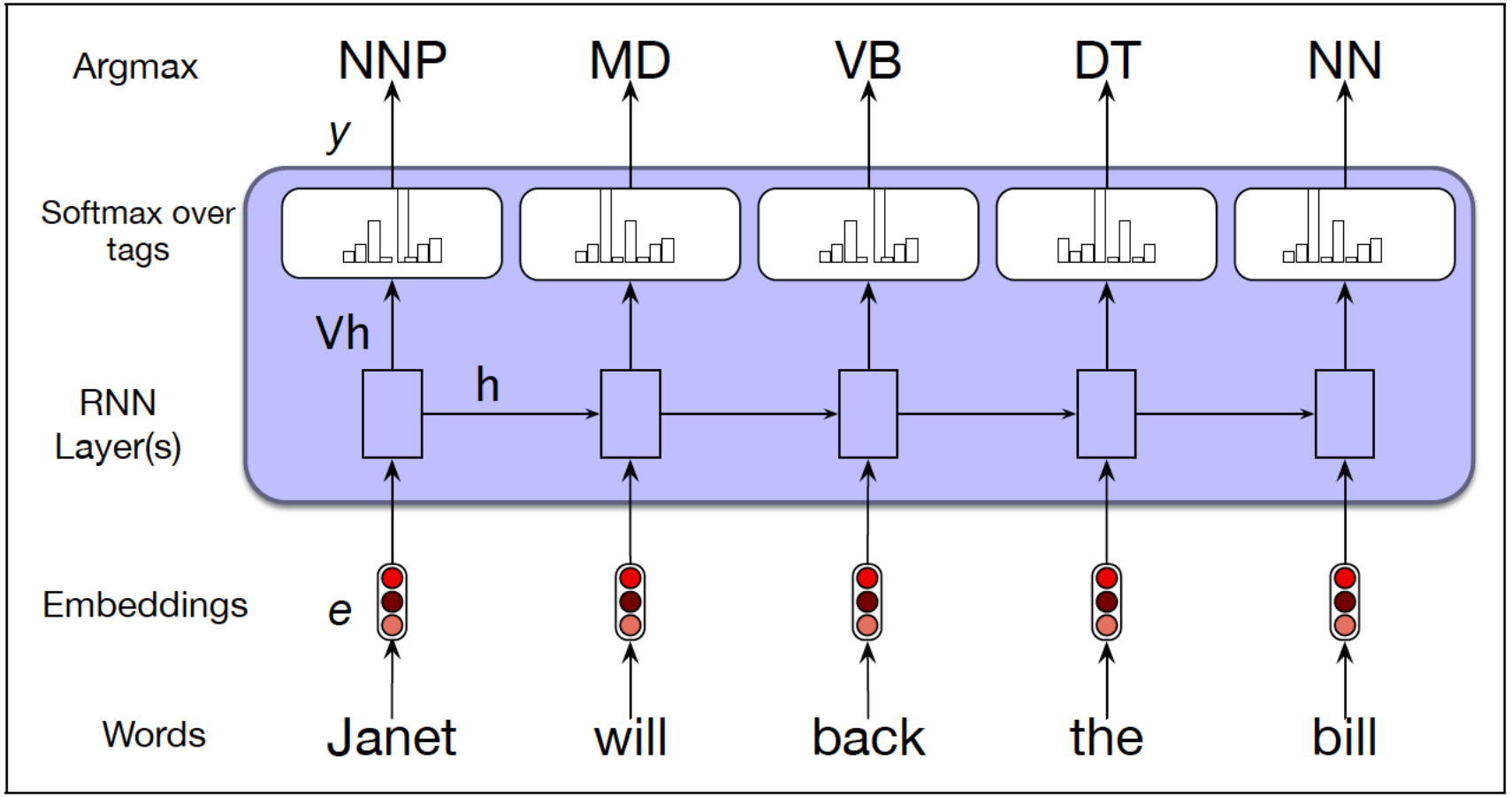


Figure 9.7 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

THE LSTM

- LSTMs are more powerful (and more complex) RNNs that take longer contexts into account
  - LSTM = Long Short-Term Memory
  - Longer context:
    "The lectures that I teach on Wednesday mornings are about Text Mining"
- The neural units used in LSTMs are more complex than those used in basic neural networks:
  - RNNs only have one type of nodes on the hidden layer
  - "LSTMs divide the context management problem into two sub-problems: removing information no longer needed from the context, and adding information likely to be needed for later decision making"

BI-LSTMS FOR SEQUENCE LABELLING

- Bidirectional neural model for NER: bi-LSTM
  - Bi-LSTM = Bidirectional Long Short-Term Memory
  - Word and character embeddings are computed for input word w_i and the context words
  - These are passed through a bidirectional LSTM, whose outputs are concatenated to produce a single output layer at position *i*
  - Simplest approach: direct pass to softmax layer to choose tag t_i

BI-LSTM-CRF FOR NER

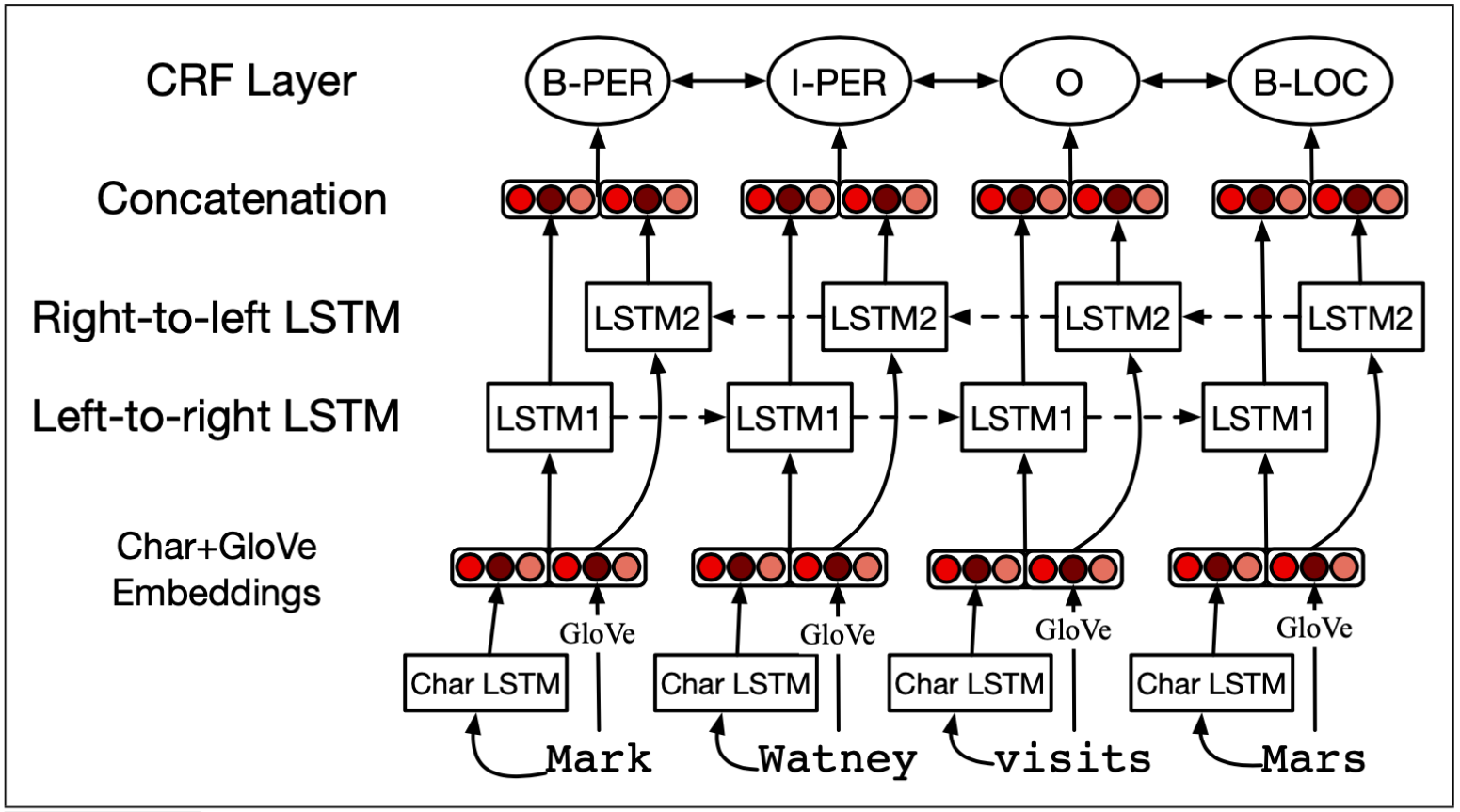


Figure 18.8 Putting it all together: character embeddings and words together in a bi-LSTM sequence model. After Lample et al. (2016).

BI-LSTMS

- For NER the softmax approach is insufficient:
  - strong constraints for neighboring tokens needed (e.g., the tag I-PER must follow I-PER or B-PER)
  - Use CRF layer on top of the bi-LSTM output: biLSTM-CRF

- BiLSTM-CRF was the state of the art for NER for some years and is still used in combination with other architectures (Transformers)
  - For example in the package Flair: https://github.com/flairNLP/flair

TRANSFORMER MODELS

Universiteit Leiden

Suzan Verberne 2022

TRANSFORMER MODELS

- LSTMs are inefficient to train
  - Recurrent networks are sequential: computation cannot be parallalized
- Breakthrough in neural sequence architectures: the Transformer

Attention is all you need

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

… the number of **attention** heads and the **attention** key and value dimensions, keeping the amount of computation constant, as described in Section 3.2.2. While single-head **attention** is 0.9 …

☆ Save Cite Cited by 53874 Related articles All 46 versions »

http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

TRANSFORMER MODELS

- The Transformer architecture is an encoder-decoder architecture
- Much more efficient than BiLSTMs and other RNNs because input is processed in parallel
- Can model longer-term dependencies because the complete input is processed at once
  - *"The lectures that I teach to master students in the Gorlaeus building on Wednesday mornings are about Text Mining"*
- But it uses a lot of memory because of quadratic complexity: $O(n^2)$ for input length of n items

THE ATTENTION MECHANISM (J&M 9.7)

- When processing each item in the input, the model has access to all of the input items
- Self-attention: each input token is compared to all other input tokens
  - Comparison: dot product

$$score(x_i, x_j) = x_i \cdot x_j$$

  - The result of a dot product is a scalar value ranging from $-\infty$ to ∞;
  - The larger the value the more similar the vectors that are being compared

SMALL EXERCISE

- Compute the dot product between:

a) $(2, -1,4)$ and $(3, -2,5)$

b) $(2, -1,4)$ and $(-1,4,3)$

Which vectors are more similar?

a) $(2, -1,4) \cdot (3, -2,5) = 6 + 2 + 20 = 28$

b) $(2, -1,4) \cdot (-1,4,3) = -2 - 4 + 12 = 6$

Pair a) is more similar

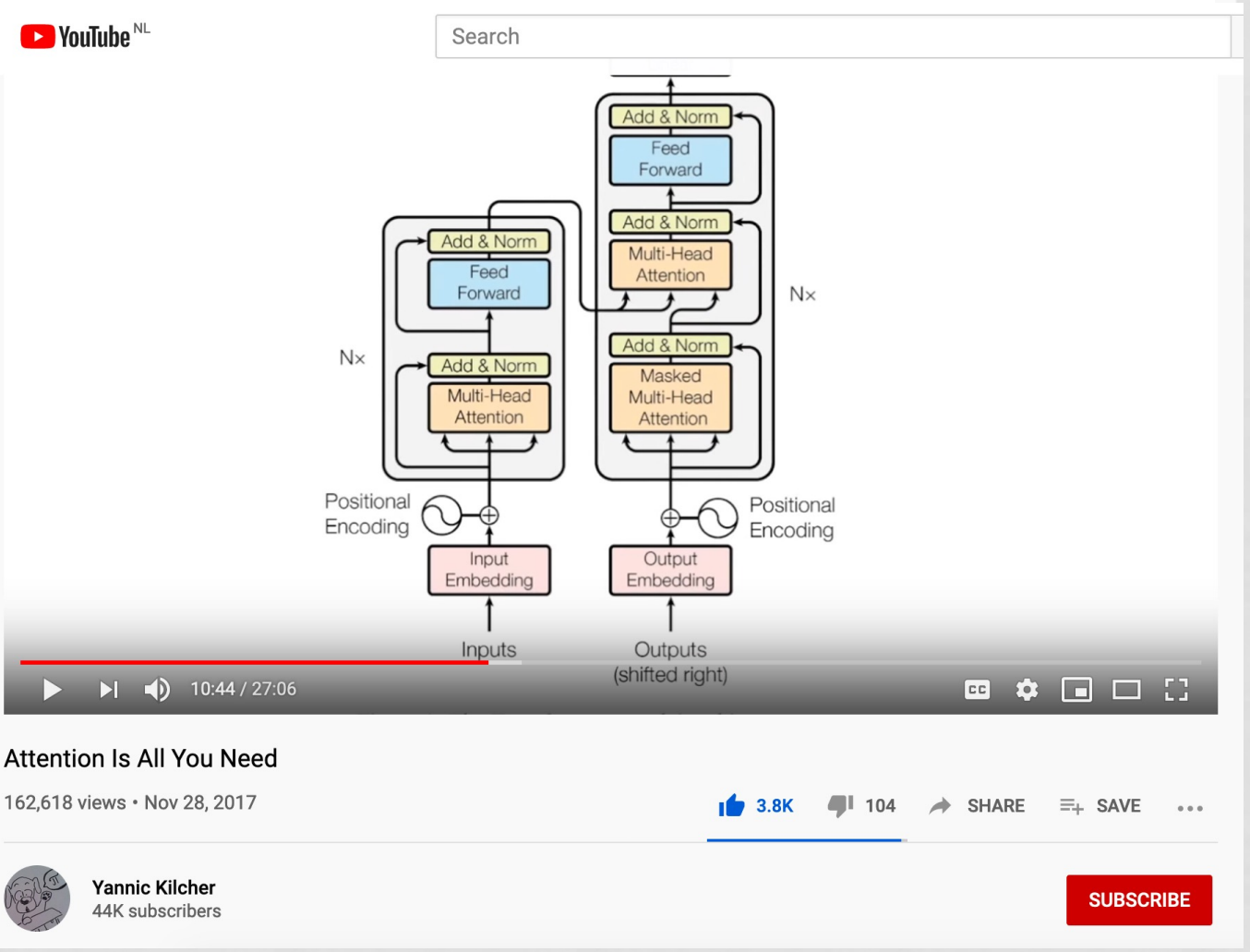
https://en.wikipedia.org/wiki/Dot_product

THE ATTENTION MECHANISM (J&M 9.7)

- Self-attention represents how words contribute to the representation of longer inputs
- And how strongly words are related to each other
- This allows us to model longer-distance relations between words
- Disadvantage: attention is quadratic in the length of the input, since at each layer we need to compute dot products between each pair of tokens in the input
- Therefore, input is maximized to 512 tokens

TRANSFORMER MODELS

- If you are more interested:



https://www.youtube.com/watch?v=iDulhoQ2pro

Universiteit Leiden

BERT

PRE-TRAINING OF DEEP BIDIRECTIONAL TRANSFORMERS FOR LANGUAGE UNDERSTANDING

Universiteit Leiden

BERT

- The Transformer led to a breakthrough in NLP:

Bert: Pre-training of deep **bidirectional transformers** for language understanding

J Devlin, MW Chang, K Lee, K Toutanova - arXiv preprint arXiv …, 2018 - arxiv.org

… called **BERT**, which stands for **Bidirectional** Encoder Representations from **Transformers**. … 2018), **BERT** is designed to **pretrain** deep **bidirectional** representations from unlabeled text …

☆ Save 99 Cite Cited by 49694 Related articles All 44 versions »

https://www.aclweb.org/anthology/N19-1423.pdf

BERT INTUITION

Pre-training of Deep Bidirectional Transformers for Language Understanding

1. Pre-training: language modelling
2. Bidirectional: Predicting randomly masked words in context
3. Transformers: efficient neural architectures with self-attention
4. Language understanding: encoding, not decoding (not generation)

BERT PRINCIPLES

- Transformer = encoder-decoder (sequence-to-sequence)
- BERT = “encoder half ” of the transformer
  - (GPT is a decoder-only transformer)
- Core idea of BERT: self-supervised pretraining based on language modelling

Lin, J., Nogueira, R., & Yates, A. (2021). Pretrained transformers for text ranking: Bert and beyond. *Synthesis Lectures on Human Language Technologies*, *14*(4), 1-325.

LANGUAGE MODELLING

- Masked language modelling:
  1. Predicting randomly masked words in context
     - To capture the meaning of words
  2. Next-sentence classification
     - To capture the relationship between sentences
- Both are trained in parallel

LANGUAGE MODELLING

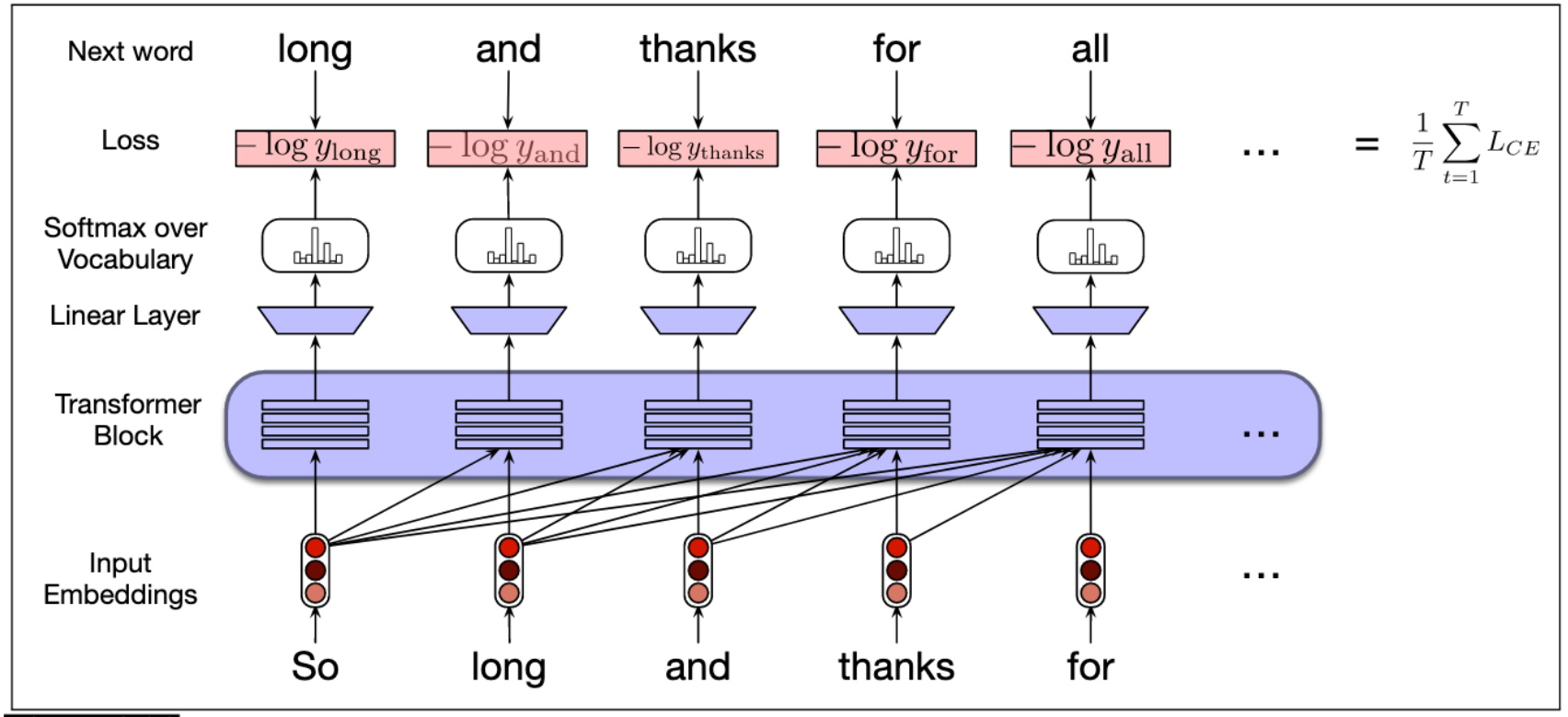


Figure 9.21 Training a transformer as a language model.

BERT ARCHITECTURE

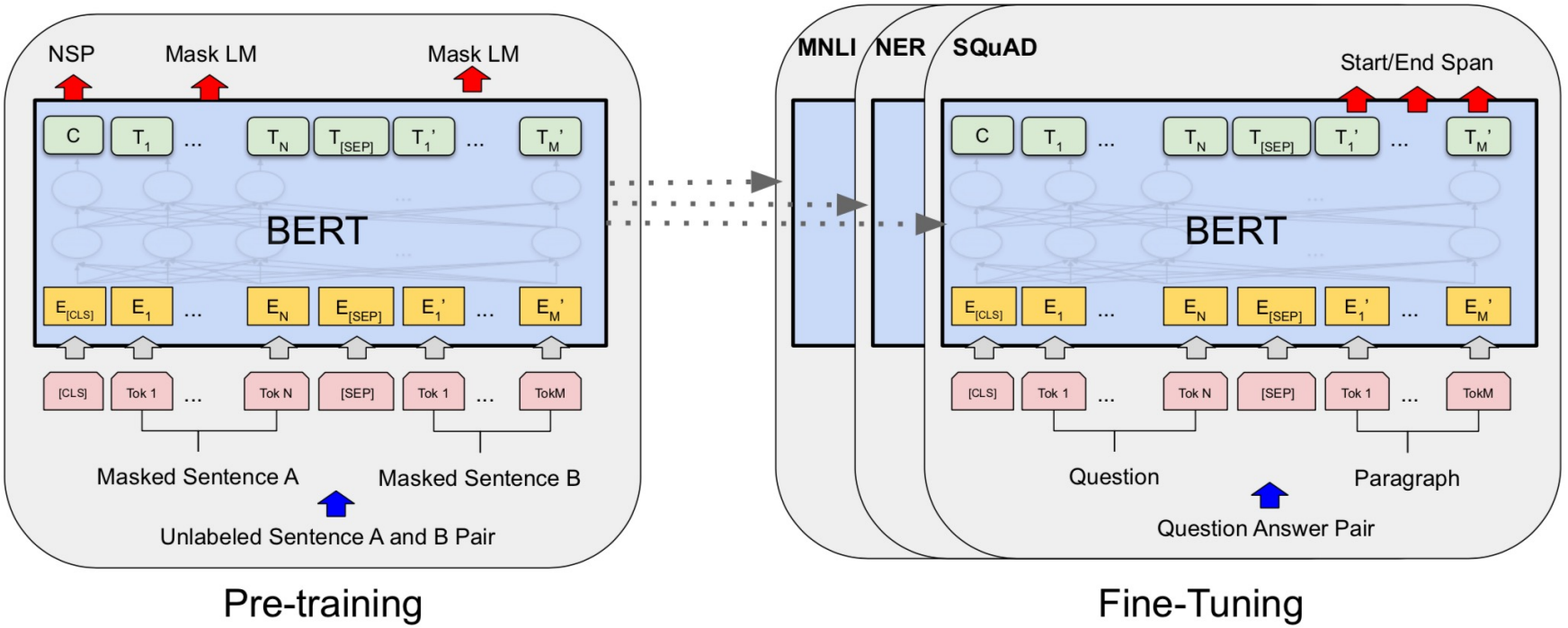


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

BERT INPUT

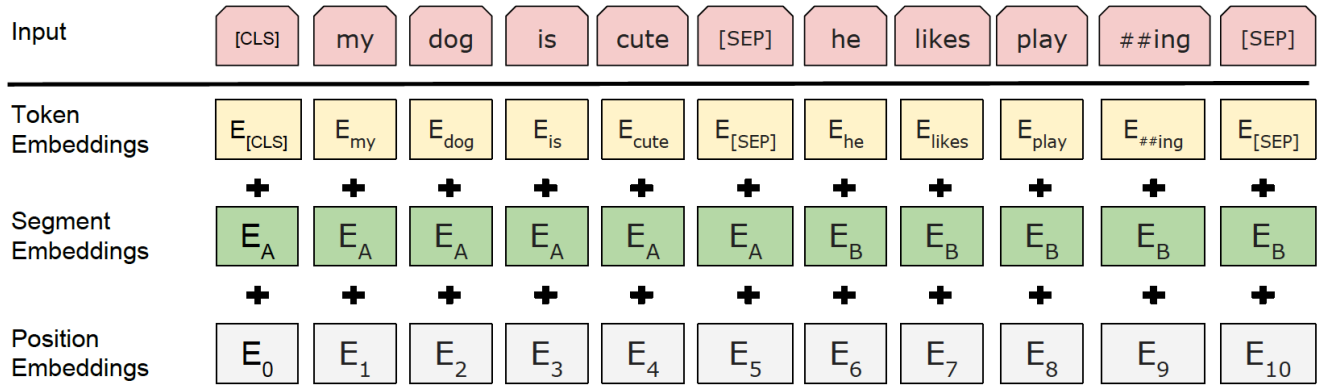


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Universiteit Leiden

Devlin et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the NAACL-HLT,* pp. 4171-4186

BERT INPUT

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

BERT uses subtokens for words that are not in the vocabulary

BERT INPUT

- BERT use a specific type of tokenization: WordPiece
- With WordPiece, a fixed-size vocabulary (e.g., 30,000 wordpieces) is defined to model huge corpora
- The WordPiece vocabulary is optimized to cover as many words as possible
  - Frequent words are single tokens, e.g. “walking” and “talking”
  - Less frequent words are split into subwords, e.g. “bi” + “##king”, “bio” + “##sta” + “##tist” + “##ics”
  - This is not linguistically motivated, but purely computationally

Lin, J., Nogueira, R., & Yates, A. (2021). Pretrained transformers for text ranking: Bert and beyond. *Synthesis Lectures on Human Language Technologies*, *14*(4), 1-325.

BERT ARCHITECTURE

- Two models presented in the BERT paper
  - $BERT_{BASE}$ (L=12, H=768, A=12, Total Parameters= 110M)
  - $BERT_{LARGE}$ (L=24, H=1024, A=16, Total Parameters=340M)
- (L: number of layers; H: dimensionality of hidden layers; A: the number of attention heads)
- "Compared to pre-training, fine-tuning is relatively inexpensive. All of the results in the paper can be replicated in at most 1 hour on a single Cloud TPU, or a few hours on a GPU, starting from the exact same pre-trained model"

https://cloud.google.com/tpu/docs/colabs

SUCCESS OF BERT

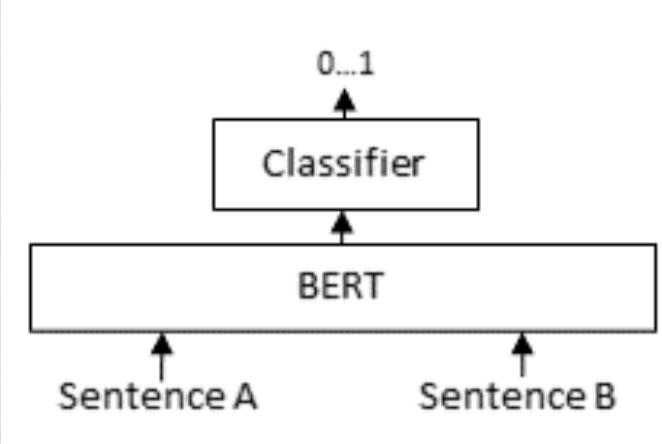
- Achieves state-of-the-art results on a large range of tasks and even in a large range of domains
- Pre-trained models can easily be fine-tuned
- Pre-trained models are available for many languages,
- as well as domain-specific pre-trained BERT models: bioBERT, sciBERT, clinicalBERT (even archeoBERTje)

SUCCESS OF BERT

- The authors (from Google) open-sourced the model implementation
- And publicly release pretrained models (which are computationally expensive to pretrain from scratch)
- https://huggingface.co/ is a the standard implementation package for training and applying Transformer models, supporting both PyTorch and TensorFlow

BERT FOR SIMILARITY

- With BERT, if we want to compute the similarity (or some other relation) between two sentences, we concatenate them in the input and then feed them to the BERT encoder
- "Finding the most similar pair in a collection of 10,000 sentences takes about 65 hours with BERT."



https://www.sbert.net

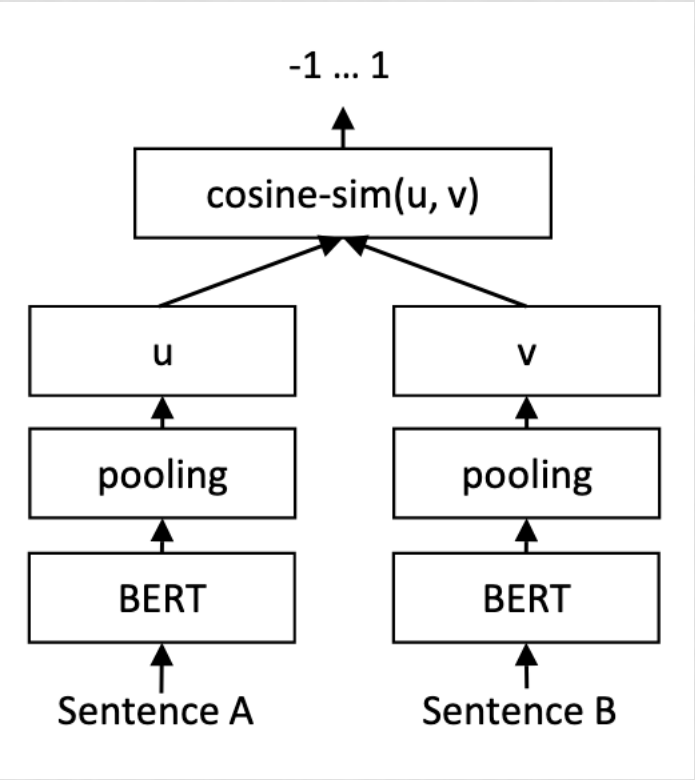
Universiteit Leiden

Reimers et al. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the EMNLP* (pp. 671-688).

SBERT

Sentence-BERT:

- Independent encoding of the two sentences with a BERT encoder
- Then measure similarity between the two embeddings
- “This reduces the effort for finding the most similar pair from 65 hours with BERT / RoBERTa to about 5 seconds with SBERT, while maintaining the accuracy from BERT”



https://www.sbert.net

Reimers et al. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the EMNLP* (pp. 671-688).

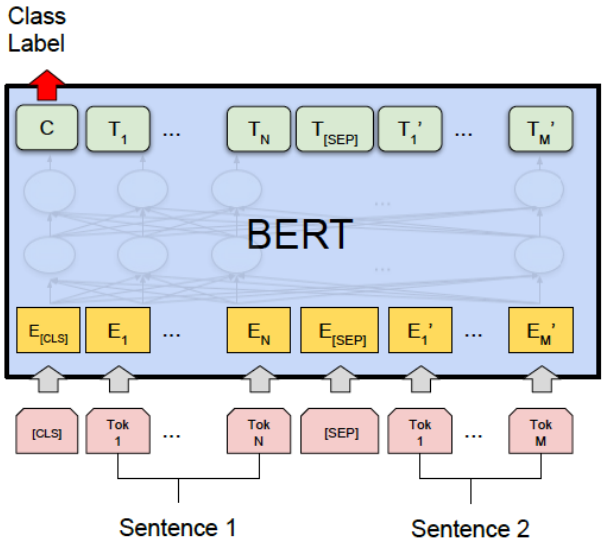
TRANSFER LEARNING

Universiteit Leiden

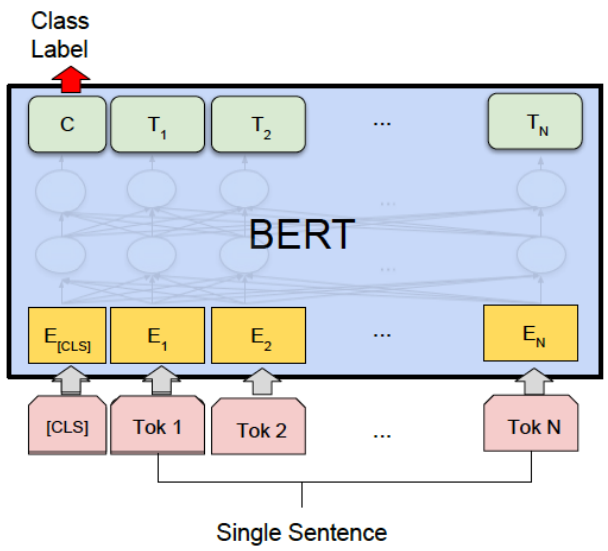
TRANSFER LEARNING WITH NEURAL LANGUAGE MODELS

- Inductive transfer learning: transfer the knowledge from pre-trained language models to any NLP task
  1. During pre-training, the model is trained on unlabeled data (self-supervision) over different pre-training tasks.
  2. For finetuning, the BERT model is first initialized with the pre-trained parameters,
  3. All of the parameters are fine-tuned using labeled data from the downstream tasks (supervised learning).
- Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters.

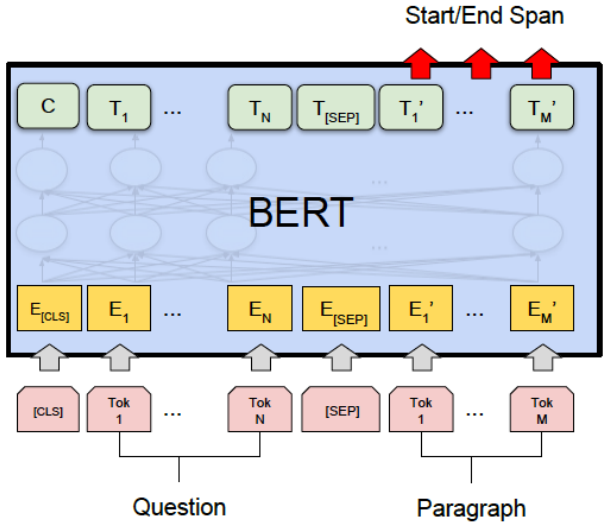
BERT FINE-TUNING ON DIFFERENT TASKS



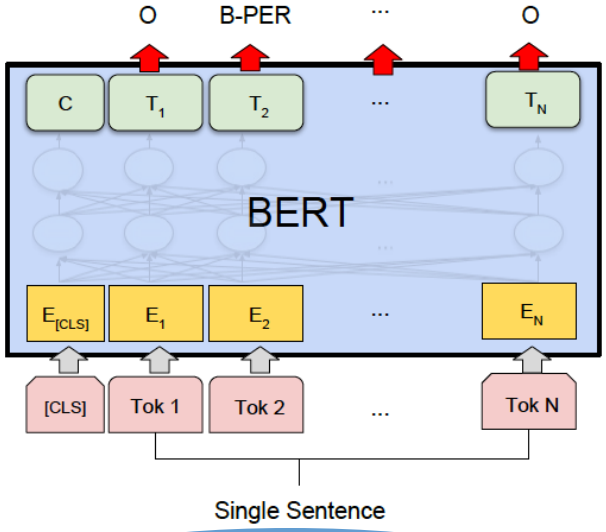
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA

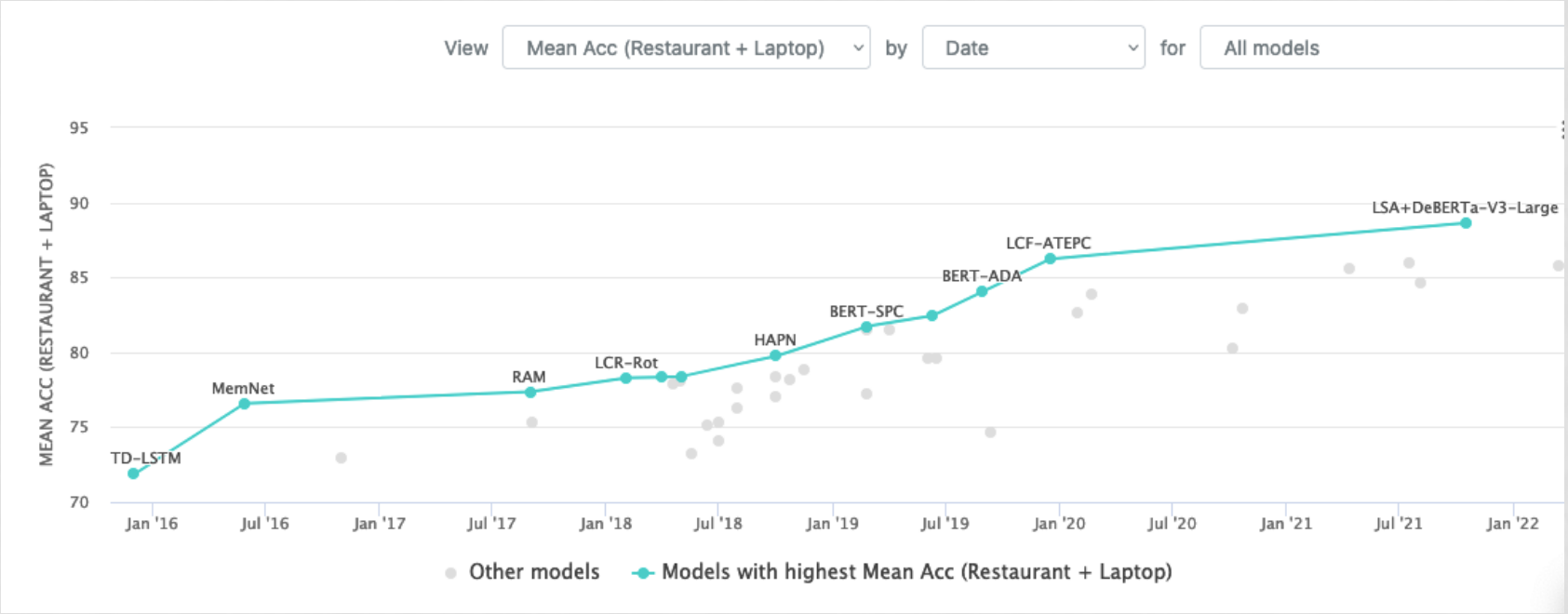


(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

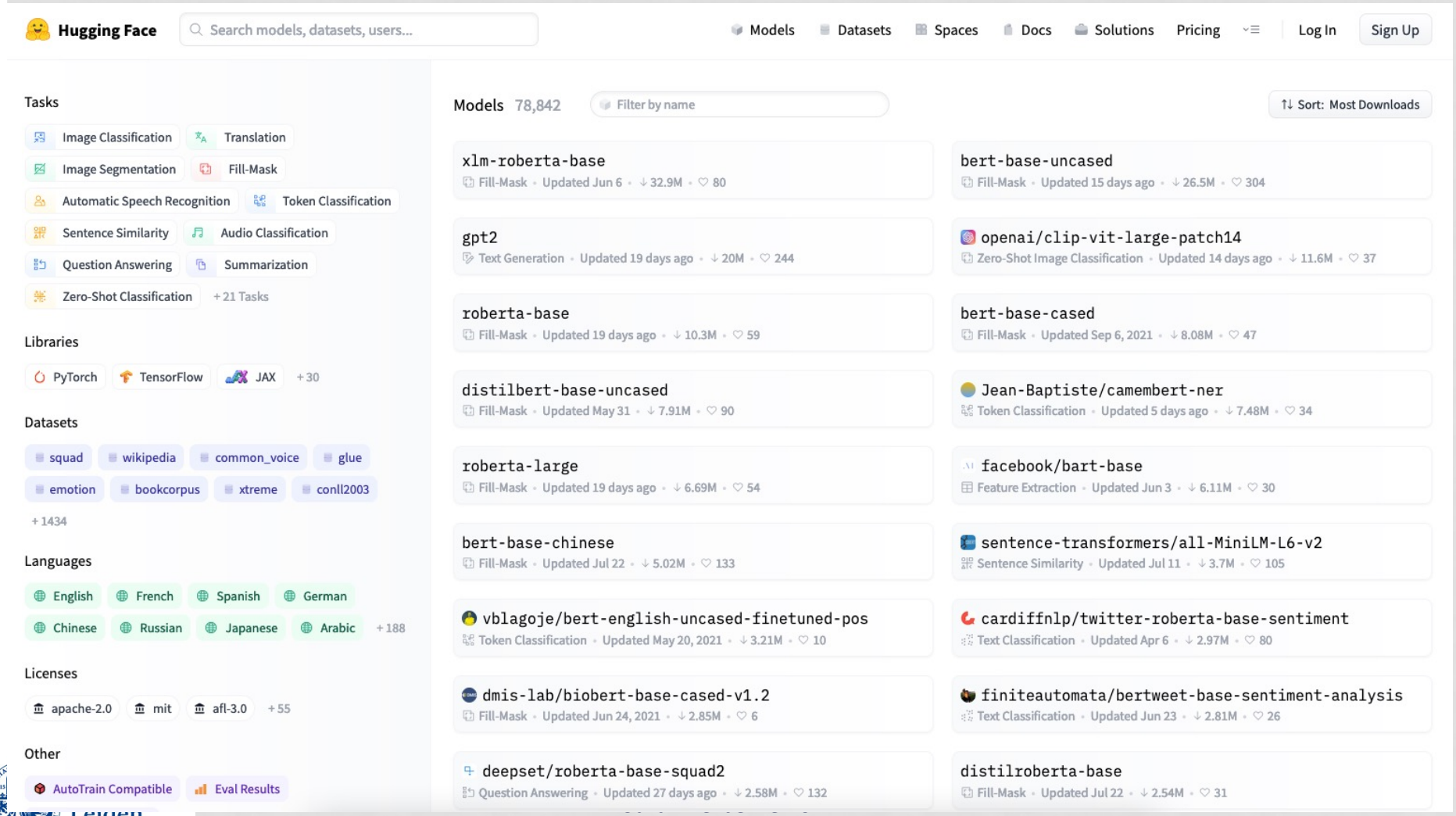
THE SUCCESS OF TRANSFER LEARNING



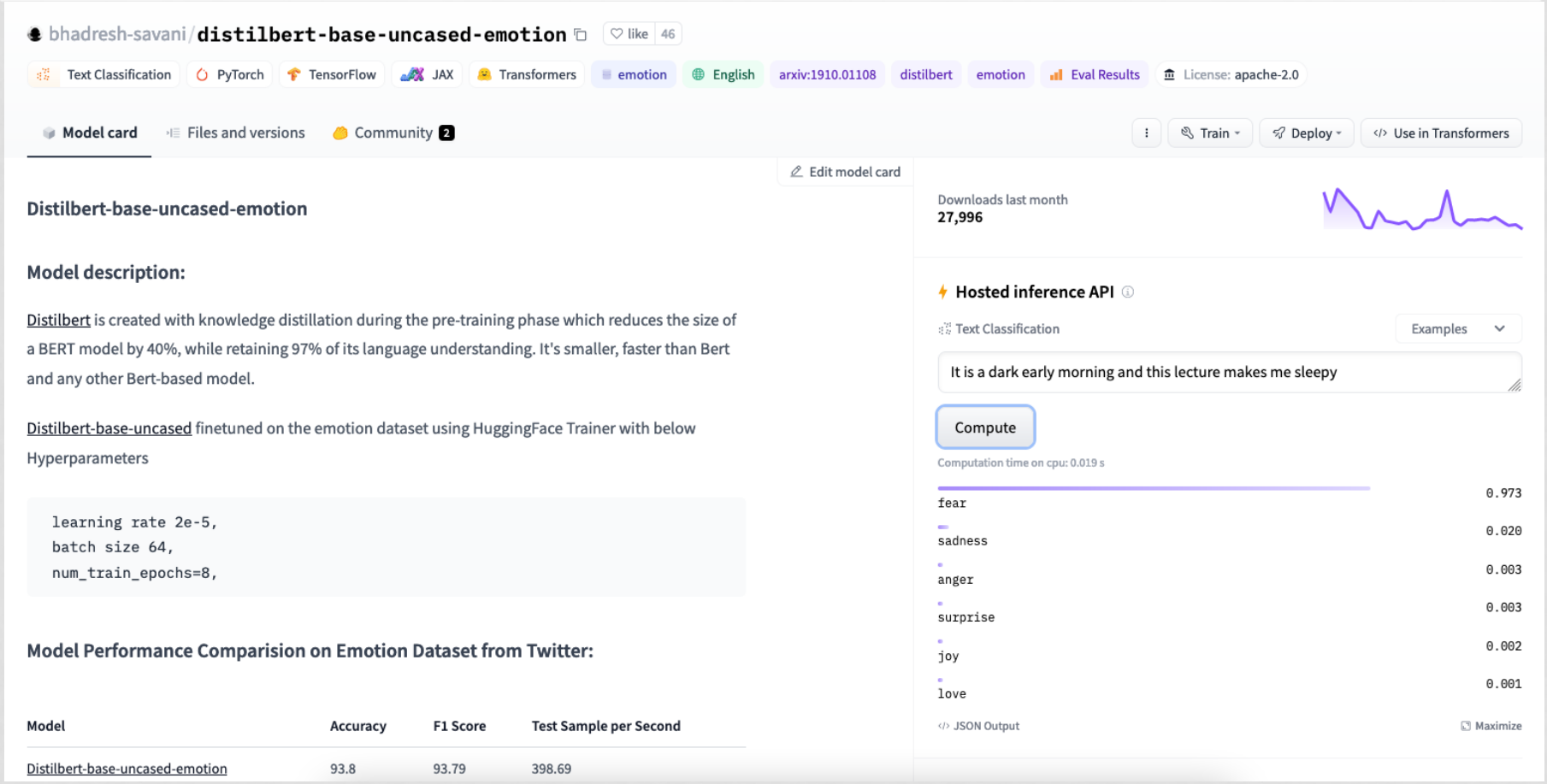
- Performance on aspect-based sentiment analysis over time

https://paperswithcode.com/sota/aspect-based-sentiment-analysis-on-semeval

PRACTICAL USE



PRACTICAL USE



Universiteit Leiden

https://huggingface.co/bhadresh-savani/distilbert-base-uncased-emotion?text=It+is+a+dark+early+morning+and+this+lecture+makes+me+sleepy

TRANSFER LEARNING WITH NEURAL LANGUAGE MODELS

- If we use a pre-trained model without fine-tuning, this is called zero-shot use
- We also use the term 'zero-shot' for the use of models that were fine-tuned by someone else or on a different task, e.g.
  - trained on sentence similarity, used for ontology mapping
  - trained on newspaper benchmark, applied to twitter data
  - trained on English, used for Dutch
- Few-shot learning = fine-tuning with a small number of samples

CHALLENGES OF STATE-OF-THE-ART METHODS

- Time and memory expensive:
  - Pre-training takes time (days) and computing power
  - Fine-tuning takes time (hours) and computing power
  - Inference (use of a fine-tuned model) needs computing power
- Hyperparameter tuning:
  - optimization on development set (takes time)
  - adoption of hyperparameters from pre-training task (might be sub-optimal)
- Interpretation/explainability: additional effort

CHALLENGES OF STATE-OF-THE-ART METHODS

- Traditional models are also still commonly used in classification tasks
  - Typically, BERT is better on the larger categories and SVM is better on the smaller categories
- Sklearn can efficiently process text data into a term-document matrix and use a range of classifiers for the learning task
- The research community has almost completely moved to neural models, but in applied contexts you will still find the traditional models

CONCLUSIONS

SUZAN VERBERNE 2022

Universiteit Leiden

HOMEWORK

- Read:
  - J&M chapter 9. Deep Learning Architectures for Sequence Processing
    - Note that the chapters refer to chapter 5 (Logistic Regression), sometimes with broken references: ??

HOMEWORK

- Exercises week 6 and 7:
  - Tutorial (required): Sequence labelling with CRFsuite (week 6). Deadline assignment 2: November 14
    https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html
  - Tutorial (optional): Fine-Tuning BERT for Text Classification
    https://towardsdatascience.com/fine-tuning-bert-for-text-classification-54e7df642894

AFTER THIS LECTURE…

- You can explain Recurrent Neural Networks and LSTMs on a conceptual level
- You can explain how self-attention works and define its computational complexity
- You can explain the difference between pre-training and fine-tuning and between self-supervised and supervised learning
- You can list the strengths and challenges of BERT models
- You can explain the difference between regular tokenization and WordPiece tokenization
- You can explain the difference between BERT and SBERT for sentence similarit tasks
- You can explain how transfer learning from pre-trained language models is used for text mining tasks