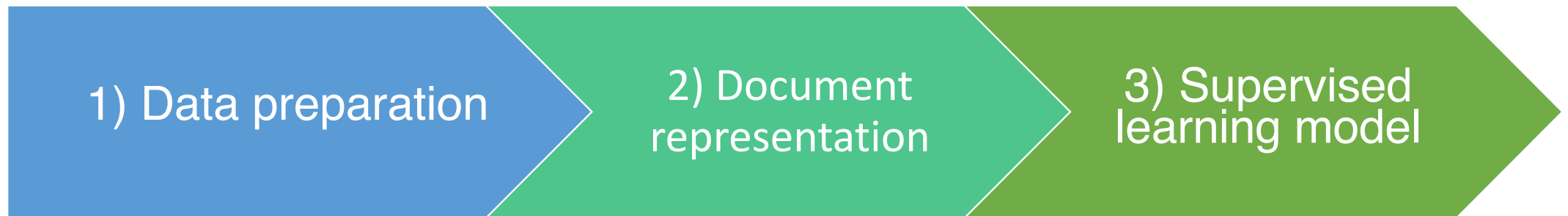


Text classification

(Common) Text classification pipeline



NBC Nightly News @nbcnightlynews
America's #1 evening news broadcast.
Tweets by @newsdel & @braddjaffy. Join us on Facebook <http://facebook.com/nbcnightlynews>

Following

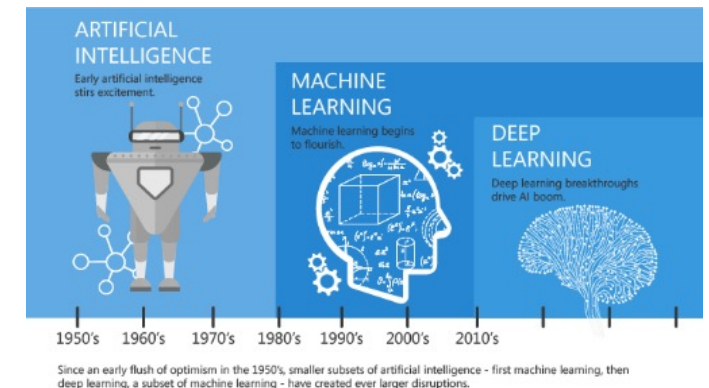
NBC News @NBCNews
A leading source of global news and information for more than 75 years. Have a news tip or question? Ask @rozzzy, @lou_dubois, @jbaiana or @anthonyquintano.

Following

CNN Breaking News @cnnbrk
CNN.com is among the world's leaders in online news and information delivery.

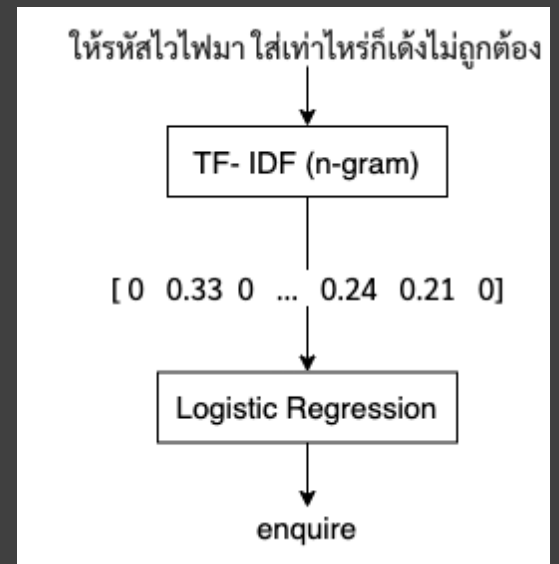
Following

Comments	Good	Like	Hate	Sentiment
Tweet1	7	8	0	😊
Tweet2	1	0	10	😡
Tweet3	2	9	1	😊



Part1: Traditional Approach

TF-IDF + Classifier



Sparse representation: Term Frequency (TF)

- Each row represents a word in the vocabulary and term-document matrix
- Each column represents a document.

vocabulary ↓	As You Like It	Twelfth Night	Julius Caesar	Henry V	document ←
battle	1	1	8	15	
soldier	2	2	12	36	
fool	37	58	1	5	
clown	5	117	0	0	

Figure 15.1 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, <https://web.stanford.edu/~jurafsky/slp3/>, August 2017

Sparse representation: TF-IDF

Need for normalization in TF

- Term Frequency (TF) – per each document

$$TF(w) = \frac{\text{Frequency of word } w \text{ in a document}}{\text{Total number of words in the document}}$$

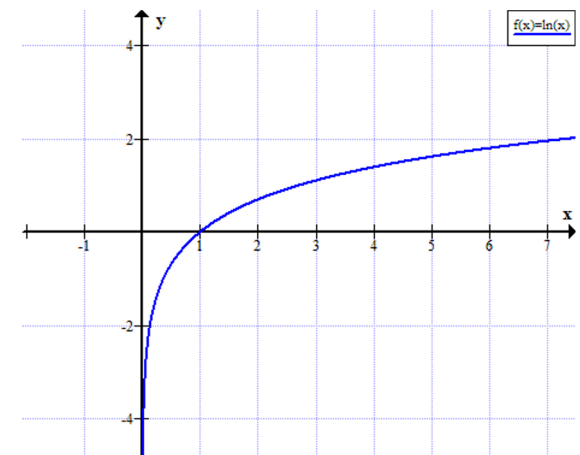
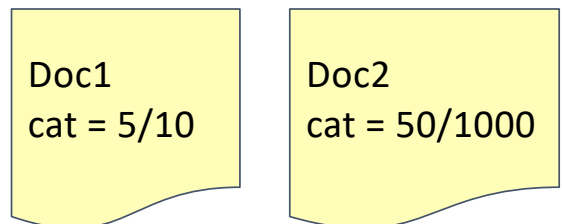
- Inverse Document Frequency (IDF) – per corpus (all documents)

$$IDF(w) = \log_e \left(\frac{\text{Total number of documents}}{\text{Number of documents that contain word } w} \right)$$

penalty score
i.e., a, an, the

- TF-IDF

$$TFIDF(w) = TF(w) * IDF(w)$$



```
1 # TF-IDF
2 tfidf = TfidfVectorizer(
3     ngram_range=(1,2),      # Use unigram and bigram
4     tokenizer=word_tokenize, # Use `word_tokenize` method from pythainlp for tokenizer
5     min_df=2,               # The word found less than three times in dataset is ignored
6     max_df=0.9,            # The word found more than 90% of entries is ignore
7     use_idf=True,
8     smooth_idf=True,
9     sublinear_tf=True
10 )
11 # Logistic regresstion
12 model = LogisticRegression(C=4, max_iter=300, random_state=42)
```

Sparse representation: TF-IDF (cont.)

TF

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	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

What classifier?

- Any classifier you like
- k-NN
- Naïve Bayes
- **Logistic regression**
- SVM
- Neural networks

```
1 # TF-IDF
2 tfidf = TfidfVectorizer(
3     ngram_range=(1,2),      # Use unigram and bigram
4     tokenizer=word_tokenize, # Use `word_tokenize` method from pythainlp for tokenizer
5     min_df=2,               # The word found less than three times in dataset is ignored
6     max_df=0.9,             # The word found more than 90% of entries is ignored
7     use_idf=True,
8     smooth_idf=True,
9     sublinear_tf=True
10 )
11 # Logistic regression
12 model = LogisticRegression(C=4, max_iter=300, random_state=42)
```

[Install](#) [User Guide](#) [API](#) [Examples](#) [More](#)

[Prev](#) [Up](#) [Next](#)

scikit-learn 0.22.1
[Other versions](#)

Please [cite us](#) if you use the software.

[sklearn.linear_model.LogisticRegression](#)
Examples using
[sklearn.linear_model.LogisticRegression](#)

sklearn.linear_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None) #
```

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default.** It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Read more in the [User Guide](#).

Parameters: **penalty** : {'l1', 'l2', 'elasticnet', 'none'}, default='l2'
Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. 'elasticnet' is only supported by the 'saga' solver. If 'none' (not supported by the liblinear solver), no regularization is applied.

New in version 0.19: l1 penalty with SAGA solver (allowing 'multinomial' + L1)

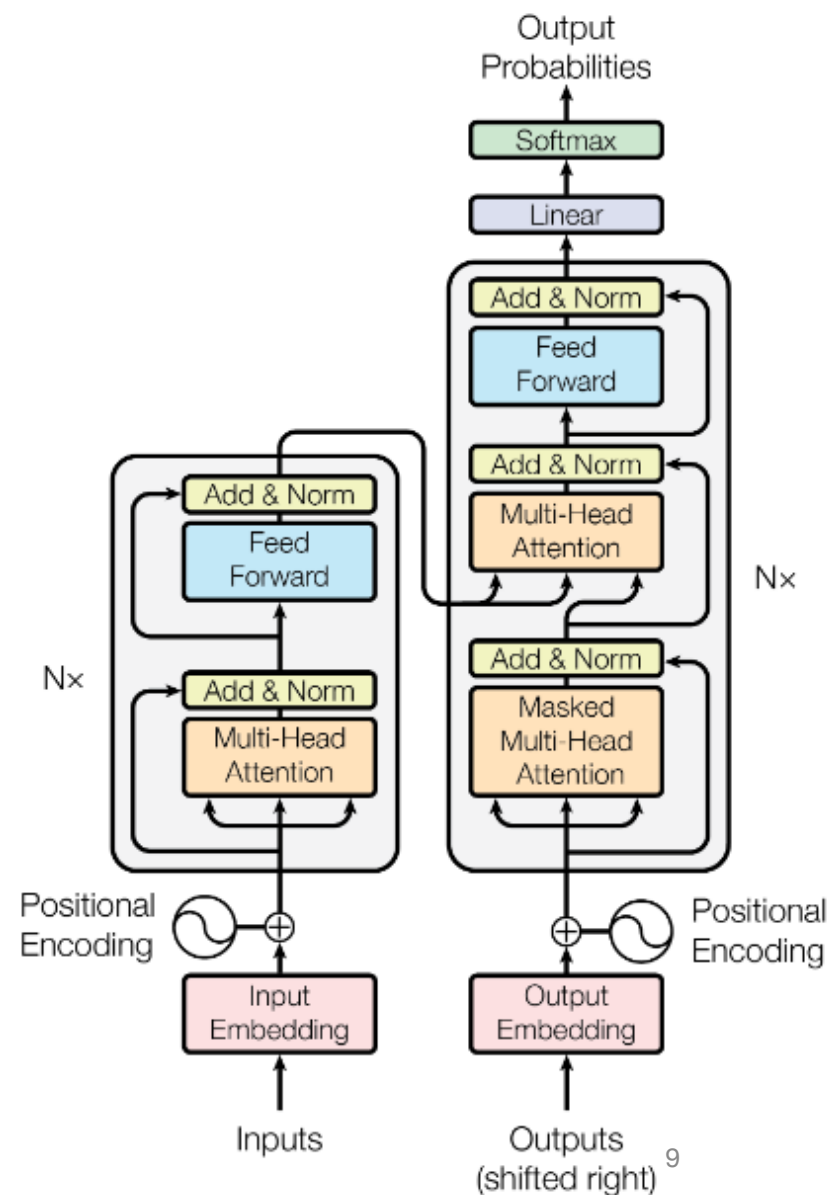
Part2: Transformer-based models

Transformer-based models

Transformer [Google Brain, 2017]

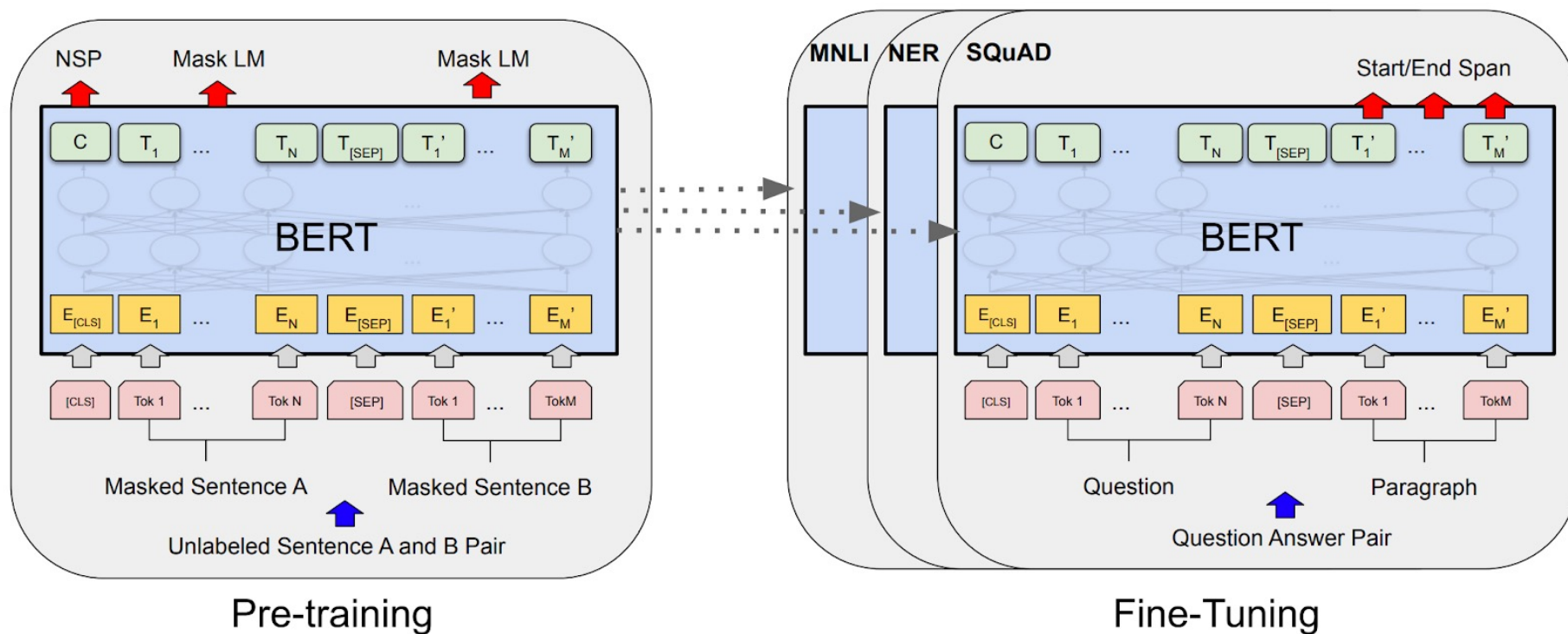
- A model based on **attention mechanism**
 - Gaining popularity in many application domain (NLP, speech, vision, bioinformatics, Reinforcement learning, Recommendation systems, etc.)

Attention is all you need 2017 <https://arxiv.org/abs/1706.03762>



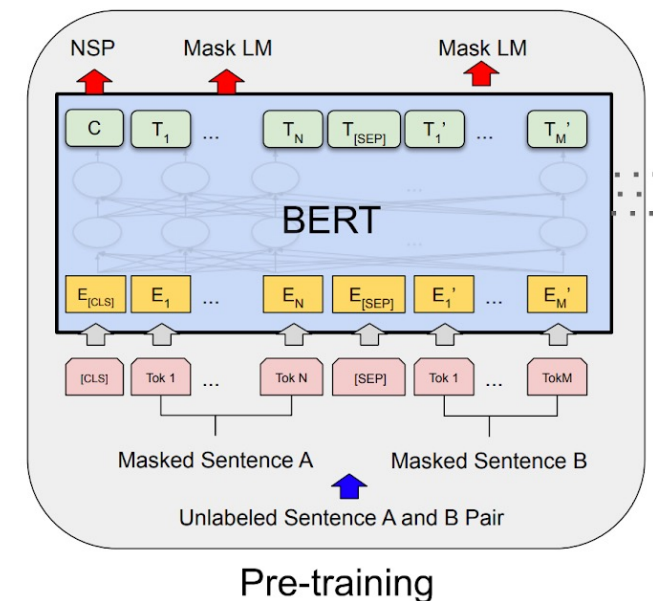
BERT [Google, 2018]

- Bidirectional Encoder Representations from Transformers
- Pretrained language model based on transformers
 - Can be used in many NLP tasks



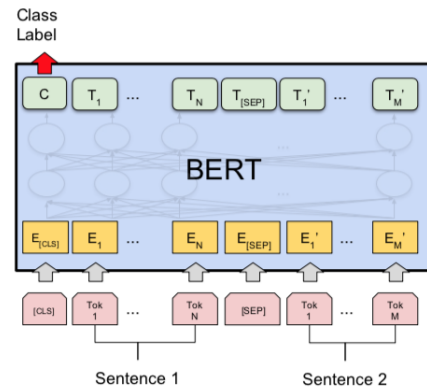
Step1: Pre-training BERT [unsupervised]

- Predicting masked words in a sentence
 - The quick brown fox jumps over the [MASK]
 - Variants: predict correct word or not, predict swapped words, etc.
- Next sentence prediction
 - A: The cat is scared. B: It hides under the table.
 - A: The apple is on the table. B: It always rain.
 - Variants: sentence order prediction

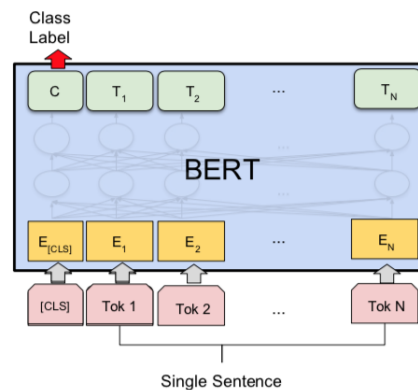


Step2: Downstream tasks with BERT

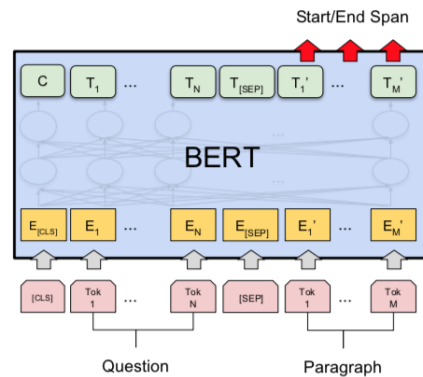
[supervised]



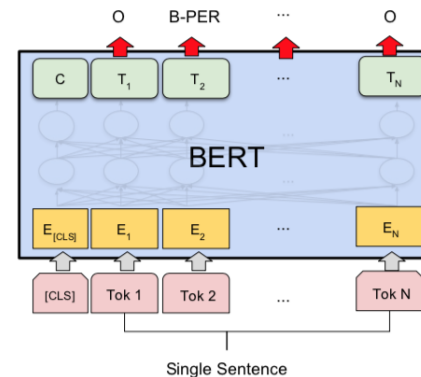
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



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



(c) Question Answering Tasks:
SQuAD v1.1



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

Current BERTrends: many variations in 2019 - 2020

- Transformers are notorious for requiring large resources
- Newer models focus on
 - Better size/compute
 - Longer context

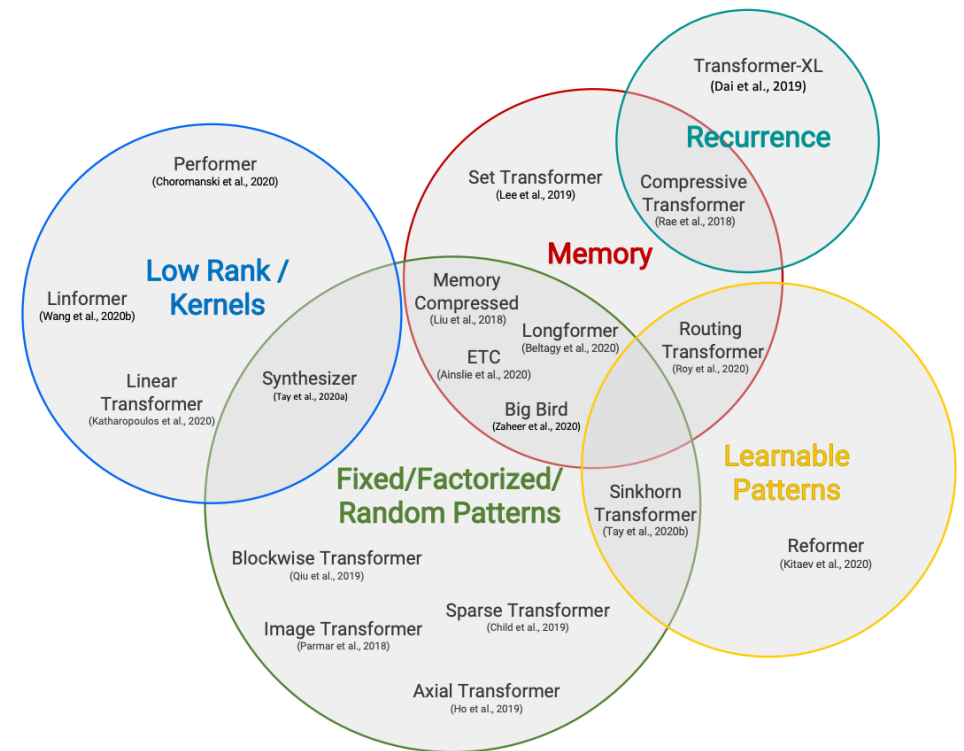


Figure 2: Taxonomy of Efficient Transformer Architectures.

Interesting BERT models

Model Name	Team	Year	Paper	Code
BERT	Google AI Language	2018	https://arxiv.org/abs/1810.04805	https://github.com/google-research/bert
RoBERTa	Facebook AI	2019	https://arxiv.org/abs/1907.11692	https://github.com/facebookresearch/fairseq/tree/main/examples/roberta
DistilBERT	Hugging Face	2019	https://arxiv.org/abs/1910.01108	https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification
TinyBERT	Huawei Noah's Ark Lab	2019	https://arxiv.org/abs/1909.10351	https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/TinyBERT
ALBERT	Google Research, Toyota Tech Institute	2019	https://arxiv.org/abs/1909.11942	https://github.com/google-research/albert
ELECTRA	Google Research	2020	https://arxiv.org/abs/2003.10555	https://github.com/google-research/electra
DeBERTa	Microsoft Research	2020	https://arxiv.org/abs/2006.03654	https://github.com/microsoft/DeBERTa

Huggingface

- An opensource library for transformer-related models
- Has datasets, models, scripts, deployment solutions
- New official online course
<https://huggingface.co/course/chapter1>



**The AI community
building the future.**

Build, train and deploy state of the art models powered by
the reference open source in natural language processing.

 Star 47,792

Transformers



V4.46.2

EN



135,083

Share your model

Agents 101

Agents, supercharged - Multi-agents, External tools, and more

Generation with LLMs

Chatting with Transformers

TASK GUIDES

NATURAL LANGUAGE PROCESSING

Text classification

Token classification

Question answering

Causal language modeling

Masked language modeling

Translation

Summarization

Multiple choice

https://huggingface.co/docs/transformers/en/tasks/sequence_classification

Join the Hugging Face community

and get access to the augmented documentation experience


Collaborative
dataset

Sign Up

```
>>> from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer

>>> model = AutoModelForSequenceClassification.from_pretrained(
...     "distilbert/distilbert-base-uncased", num_labels=2, id2label=id2label, label2id=label2id
... )
```

Text classification

[Open in Colab](#)
[Open in Studio Lab](#)


Thai pre-trained BERTs

- WangchanBERTa [2021] is a Thai language model developed by AIResearch.
 - Built upon the [RoBERTa-base architecture](#), it is specifically designed to address the unique characteristics and challenges of the Thai language.
 - Pretrained on a substantial dataset of [78.5 GB](#), comprising diverse sources such as social media posts, news articles, and other publicly available Thai texts.
- PhayaThaiBERT [2023] is a Thai language model developed to enhance the understanding of unassimilated [loanwords—foreign words](#), particularly English, that are integrated into Thai text without orthographic changes.
 - Building upon the foundation of [WangchanBERTa](#)
 - Pretrained on [a dataset larger](#) than that used for WangchanBERTa, providing a more comprehensive linguistic foundation

<https://huggingface.co/airesearch/wangchanberta-base-att-spm-uncased>

<https://huggingface.co/Pavarissy/phayathaibert-thainer>

