METODE INTELIGENTE DE REZOLVARE A PROBLEMELOR REALE

Laura Dioşan
Text mining

Facultatea de Matematică și Informatică
Universitatea Babeș-Bolyai

- Natural Language Processing
  - Text classification
  - Language modelling
  - Machine translation
  - …

From Languages to Information

- Aim
  - Automatically extracting meaning and structure from:
    - Human language text and speech (news, social media, etc.)
    - Social networks
    - Genome sequences
  - Interacting with humans via language
    - Dialog systems/Chatbots
    - Question Answering
    - Recommendation Systems

From Languages to Information

- How?
  - Extracting information from language
    - Information Retrieval
    - Text Classification
    - Extracting Sentiment and Social Meaning
    - …
  - Interacting with humans via language
    - IBM's Watson
    - …
    - Counseling conversations
      - https://www.aclweb.org/anthology/Q16-1033.pdf
      - https://web.stanford.edu/class/cs124/lec/counseling_slides.pdf
    - Understanding police officer respect
      - https://www.pnas.org/content/114/25/6521
      - https://nlp.stanford.edu/robvoigt/124_lecture/

Text classification

- Definition
  - Text categorization
    - Assign (predefined) labels (categories) to some documents
    - Documents
      - Technical reports, web pages, messages, books, etc.
    - Categories:
      - Topics (art, economy)
      - Attributes (spams, ham)
- Data and various text classification problems
  - https://nlpprogress.com/english/text_classification.html

	Cuvinte	Documente
Învăţare supervizată	Etichetarea părţilor de vorbire	Clasificarea textelor, Filtrarea, Detectarea subiectelor
Învăţare nesupervizată	Indexarea semantică, construcţia automată a tezaurelor, extragerea cuvintelor cheie	Clusterizarea documentelor, Detectarea subiectelor

Text classification

- Automatic approaches
  - Based on knowledges
    - From experts
    - Encoded as rules
  - Based on learning
    - Experts label some training examples
    - The algorithm label the test examples
    - Learning
      - Supervised
      - Unsupervised
- Evaluation
  - Accuracy, Precision, Recall, AUC, etc.

Text classification

- Process
  - Data training analysis
    - **Document indexing**
      - Construct a document representation
        - Attributes / features
        - Weights of these features
      - Reduce the dimension of these representations
        - Feature selection
        - Feature extraction
    - Learning a classification model
  - Classification of test data
    - Text indexing
    - Apply the learnt model -> categories

Text classification

- Document indexing
  - Construct a document representation
    - Attributes / features
    - Weights of these features
  - 4 steps:
    1. Document linearization
    2. Filtering
    3. Canonical form
    4. Weighting

} Reduce the vocabulary size

Text classification – indexing

Step 1: linearization (segmentation)

- Reduce the document to a vector of terms (attributes)
  - *bag of words model*
  - A matrix
    - Documents -> lines
    - Terms -> colomns
    - Each cell 1/0 – if the current term belongs to the current document
- 2-stage process for term identification
  - Format removal
    - E.g. Elimination of HTML tags
  - *Tokenization*
    - Parsing (segmentation)
    - Removal of punctuation mark
    - Conversion of capital letters

Iniţial	Liniarizat
Interactive query expansion modifies queries using terms from a user. Automatic query expansion expands queries automatically.	interactive query expansion modifies queries using terms from a user automatic query expansion expands queries automatically

Text classification -> indexing

Paul 2: filtering

- Select some terms that are able to
  - Describe the content of the document
  - Make a difference between two documents
- Removal of stopwords
  - From a predefined list
  - Based on their frequencies (under a given threshold)

Segmentat	Filtrat
interactive query expansion modifies queries using terms from a user automatic query expansion expands queries automatically	interactive query expansion modifies queries terms automatic query expansion expands queries automatically

Text classification – indexing

- Step 3 – canonical form (normalization/standardisation)
  - Lematisation
    - Morphological analysis
    - Based on context
    - Ex. “better” → “good”
  - *Stemming*
    - Term-level
    - Ex. "computer", "computing", "compute" → "comput"
    - *stemming* algorithm
      - Martin Porter
      - WordNet

Filtrat	Redus
interactive query expansion modifies queries terms automatic query expansion expands queries automatically	interact queri expan modifi queri term automat queri expan expand queri automat

Text classification – indexing

Step 4: weighting

- Pre-determined (manual)
  - Ex. Bag of Words
- Automatic (learnt)
  - Ex. embedded representations

Text classification - indexing

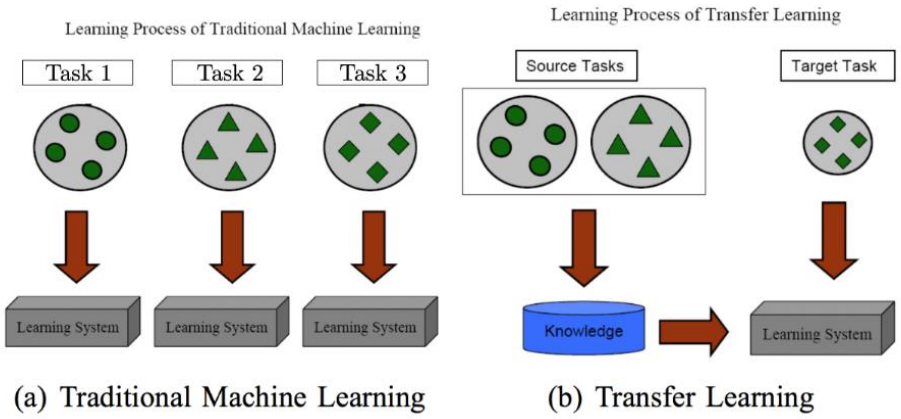
- Step 4: manual weighting
  - Usage of a particular model
  - Weights relative to
    - A single document
      - *term frequency – TF*
    - A collection of documents
      - *inverse document frequency – IDF*
    - hybrid *TF* and *IDF*
      - *TF* → cu cât un termen este mai frecvent într-un document, cu atât el este mai important pentru acel document
      - *IDF* → cu cât un termen apare în mai multe documente, cu atât el este mai puţin important în descrierea semanticii acelui document
  - Range of frequencies
    - Binary → presence / absence of a term (One-Hot encoding)
    - Real ([0,1]) → importance of that term
  - For a set of D documents and a set of T *termns, the weight* p_{ij} of term t_i in document d_j (i=1,2,...,$|T|$, j=1,2,...,$|D|$) can be:
    - binary: p_{ij} = 1, if t_i belongs to d_j
      0, otherwise
    - *TF*: $p_{ij}=tf_{ij}$ (no of appearances of term t_i in document d_j)
    - *TF.IDF*: $p_{ij}=tf_{ij}*\log_2(|D|/df_i)$, where df_i=no of documents where term t_i is found

Text classification – indexing

- Step 4 – weight learning
  - Unsupervised learnt representation
  - Classical models
    https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf
  - https://web.stanford.edu/class/cs124/lec/lm2021.pdf
  -
    - Probability of a target word based on k previous words
    - Document-level context
      - Semantic relation (boat – water)
    - Count-based models
      - Latent Semantic Analysis (LSA)
      - Latent Dirichlet Allocation (LDA)
  - Modern models
    - Word-level context
      - Semantic similarity (boat – sheep)

Text classification – indexing

- Step 4 – weight learning (word representations)
  - Methodologies
    - Traditional Machine Learning
    - Transfer learning https://www.cse.ust.hk/~qyang/Docs/2009/tkde_transfer_learning.pdf



(a) Traditional Machine Learning (b) Transfer Learning

Text classification – indexing

Learning word representations (vectors)

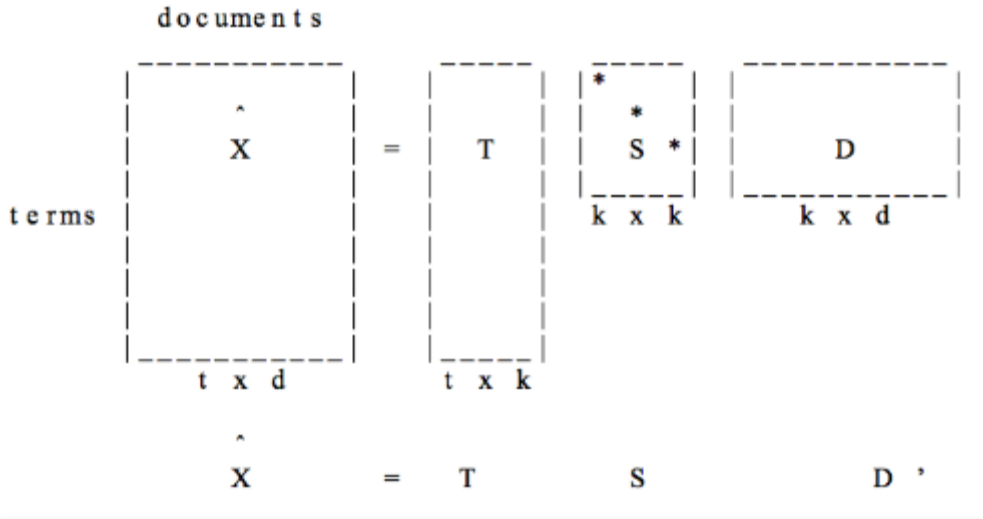
- Why?
  - Embeddings = parameters -> they can be learnt
  - Share representation across tasks
  - Lower dimensional space

- How?
  - Pre-NN
    - 1990 - Latent Semantic Analysis (LSA) http://lsa.colorado.edu/papers/JASIS.lsi.90.pdf
    - 1992 – n-gram models https://www.aclweb.org/anthology/J92-4003.pdf
    - 2003 - Latent Dirichlet Allocation (LDA)—Documents are mixtures of topics and topics are mixtures of words https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf
  - NN-based
    - Word-level (2003 - …)
    - Sentence (document) level (2014 - …)
    - Contextual word-vectors (Word vectors compress all contexts into a *single vector) (2016 - …)*

Text classification – indexing

Learning word representations (vectors)

- How?
  - Pre-NN
    - 1990 - Latent Semantic Analysis (LSA) http://lsa.colorado.edu/papers/JASIS.lsi.90.pdf



Text classification – indexing

Learning word representations (vectors)

- How?
  - Pre-NN
    - 1992 – n-gram models
      https://www.aclweb.org/anthology/J92-4003.pdf
    - https://web.stanford.edu/~jurafsky/slp3/3.pdf

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Text classification – indexing

Learning word representations (vectors)

- How? -> NN-based
  - Word-level (2003 - …)
    - 2003 - N-gram Neural language model (Montreal - Bengio)
      - https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
      - Probability of the target word based on previous k words
      - Estimation of the probability by an ANN -> prediction of the next word in a sequence
        - Embedding layer -> word embeddings
        - Intermediate layer(s) -> intermediate representation (non-linear); standard (HardTanh layer) or recurrent layers
        - softmax layer for producing probabilities over vocabulary) -> main bottleneck
        - Cross-entropy loss (max the probability of the next word)

Text classification – indexing

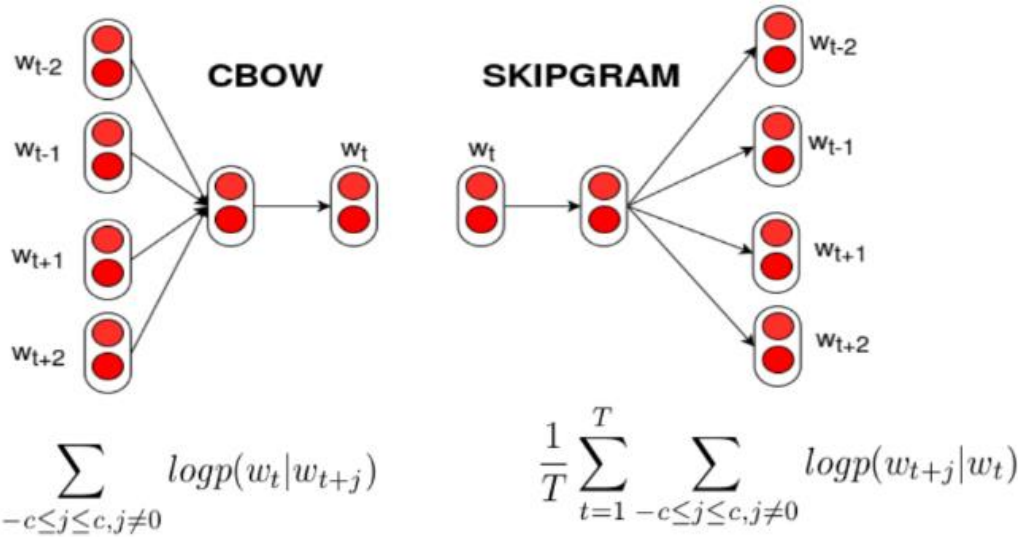
Learning word representations (vectors)

- How? -> NN-based
  - Word-level (2003 - …)
    - 2008 – multi-task model (Princeton – Collobert)
      - https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf
      - Probability of MORE target words (a sequence of words)
      - Estimation of the probability by an ANN –similar to Bengio's model, but
        - a pairwise ranking criterion
        - outputs a higher score for a correct word sequence than for an incorrect one

Text classification – indexing

Learning word representations (vectors)

- How? -> NN-based
  - Word-level (2003 - …)
    - 2013 – word2vec (Google – Mikolov)
      - https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf
      - Continous BOW model -> try to predict a word based on its context (neighbours); input = neighbour words, output = target word
      - Skip-gram model -> try to predict context (neighbours) of a word; input = target word; output = neighbour words;
      - Visualisation of embeddings https://ronxin.github.io/wevi/
      - Trained vectors https://radimrehurek.com/gensim/models/word2vec.html or https://github.com/3Top/word2vec-api#where-to-get-a-pretrained-models



Text classification – indexing

Learning word representations (vectors)

- How? -> NN-based
  - Word-level (2003 - …)
    - 2014 – GloVe (Stanford – Pennington, Manning)
      - https://nlp.stanford.edu/projects/glove/
    - 2017 - fastText (Facebook – Mikolov)
      - https://arxiv.org/abs/1607.04606
      - https://radimrehurek.com/gensim/models/fasttext.html

Text classification – indexing

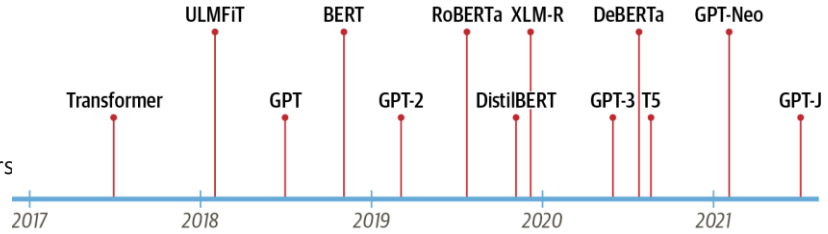
Learning word representations (vectors)

- How? -> NN-based
  - Word-level (2003 - …)
  - Sentence (document) level (2014 - …)
    - 2014 - Paragraph embedding
      - https://arxiv.org/abs/1405.4053
    - 2015 – skip-thought vectors
      - Predict previous / next sentence with seq2seq model)
      - https://arxiv.org/abs/1506.06726
    - 2015 – auto-encoders (semi or unsupervised)
      - https://arxiv.org/abs/1511.01432 ,
      - https://arxiv.org/pdf/1511.06349.pdf,
      - https://arxiv.org/abs/1602.03483

Text classification – indexing

Learning word representations (vectors)

- How? -> NN-based
  - Word-level (2003 - …)
  - Sentence (document) level (2014 - …)
  - Contextual word-vectors (Word vectors compress all contexts into a *single vector) (2016 - …)*
    - 2016 context2vec https://www.aclweb.org/anthology/K16-1006.pdf
    - 2017 tagLM https://arxiv.org/abs/1705.00108
    - 2017 CoVe https://papers.nips.cc/paper/2017/hash/20c86a628232a67e7bd46f76fba7ce12-Abstract.html
    - 2018 ELMo https://www.aclweb.org/anthology/N18-1202/
    - 2018 ULMFiT https://arxiv.org/abs/1801.06146
    - 2018 GPT (→ 2023 GPT4)
      - Generative Pre-trained Transformer
      - OpenAI
    - 2019 BERT
      - Bidirectional Encoder Representations from Transformers
      - Google
      - https://arxiv.org/abs/1810.04805
    - 2019 T5
      - Text-to-Text Transfer Transformer
      - Google



f(play | The kids play a game in the park.)

!=

f(play | The Broadway play premiered yesterday.)

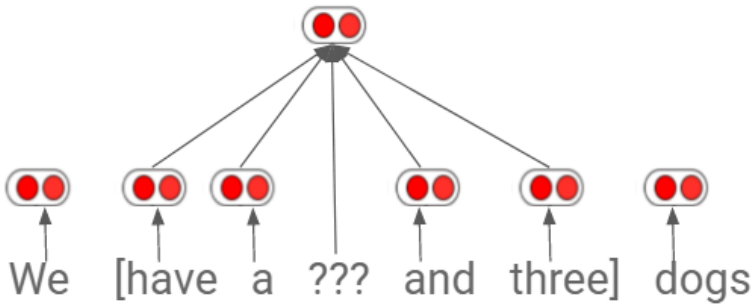
- cross-lingual pre-training https://arxiv.org/abs/1706.04902
  - Code example https://github.com/huggingface/naacl_transfer_learning_tutorial
  - See also https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

Text classification – indexing

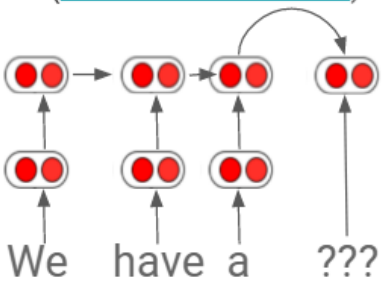
Learning word representations (vectors)

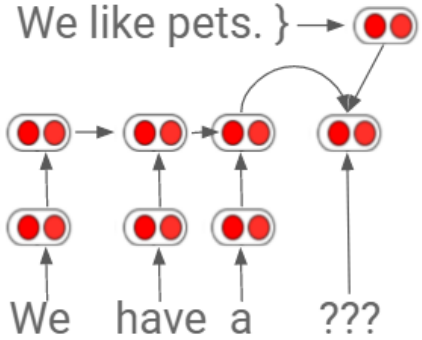
- How? -> NN-based

word2vec, Mikolov et al (2013)



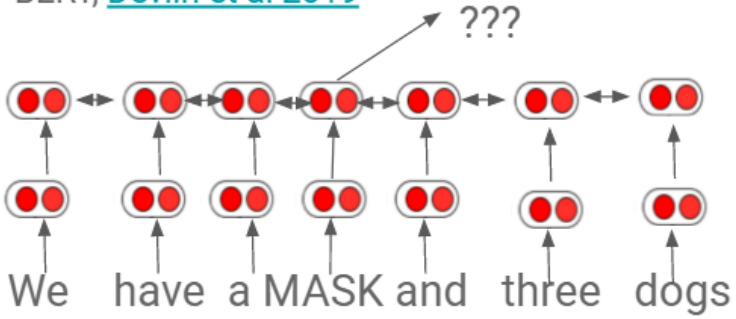
ELMo, Peters et al. 2018, ULMFiT (Howard & Ruder 2018), GPT (Radford et al. 2018)





Skip-Thought (Kiros et al., 2015)

BERT, Devlin et al 2019



Embeddings

- Process
  - Text representation (pre-processing)
    - Token-based models
    - Token-free models
  - Feature computation
    - Frequency-based embeddings
      - TF-IDF
      - Co-occurrence matrix
    - Prediction based embeddings
      - Word / subword embeddings
        - Word2vec (Skip-gram or Continuous Bag of Words (CBOW))

Embeddings

- Process → Text representation
  - Token-based models
    - White Space: Splits on spaces (simple but limited)
    - Word: Breaks into words (common for English)
    - Sentence: Divides text into sentences
    - Character: Splits into individual characters
    - N-gram: Creates sequences of n items/elements
      - fastText
    - Subword: Breaks words into smaller parts
      - Byte Pair Encoding (BPE): Merges common character pairs
      - WordPiece: Google's method for balancing words and subwords
      - Unigram

Method	Complexity	Best For	Drawback
White Space	Low	Simple tasks	Fails with some languages
Word	Low	General NLP	Struggles with contractions
Subword	High	Unknown words	More complex

  - Token-free models
    - CharFormer
    - byT5
    - MegaByte
    - Byte-latent transformer

FastText – 2016

- Text tokenization
  - Subwords = each word is represented as a bag of character n-grams in addition to the word itself.
  - E.g. “biking” would be represented by
    - the 2-grams: “bi”, “ik”, “ki”, “in”, “ng” and “<biking>”
    - The 3-grams: “bik”, “iki”, “kin”, “ing” and “<biking>”
  - Dictionary
    - union of the subwords of all words
    - Subwords for various n (n=3, 4, 5, 6)
    - Larger than for original 1-hot representation from Word2Vec → more parameters
  - Representation
    - Each word → 1-hot encoding for the extended dictionary
- Feature computation
  - Apply skip-gram model principles for training the weights
  - Each word is represented by the sum of its subword embeddings.
- Advantages
  - Handles Out-of-Vocabulary (OOV) Words (e.g bikelane)
  - Captures Morphological Information (e.g., “biking” and “biker”)
- Weaknesses
  - High memory requirement
- Example
  - Word: “bikelane“ (OOV)
  - Character 3-grams: “bik", “ike", “kel", “ela“, “lan”, “ane”
  - Embedding: The embedding for “bikelane" is the sum of the embeddings of its n-grams.
- More details
  - Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information (2016). arXiv preprint arXiv:1607.04606 link
  - https://fasttext.cc/docs/en/supervised-tutorial.html
  - https://github.com/facebookresearch/fastText/

BPE – 2016

- Text tokenization
  - subwords = each word is represented as a list of tokens composed of characters
  - a compression algorithm adapted for subword tokenization. It iteratively merges the most frequent pairs of characters or character sequences in a corpus to form subword units.
  - deterministic algorithm – based on the occurrences of words in the given text
  - Process of tokenization = building with Legos (from tiny pieces to bigger ones by various combinations)
    - Token learning = raw training text → learnt vocabulary (set of tokens)
      - Begin with single characters
      - Find the most common character pair
      - Merge that pair into a new token
      - Repeat until you hit your token limit
    - Segmenting / Processing = raw test text → list of tokens (based on the vocabulary)
      - Apply the merge rules learned in order on the individual characters of the given text

BPE – 2016

- Text tokenization
  - E.g.
    - Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars): (b, i, k, e), (b, i, k, e, r), (l, i, k, e)
    - Vocabulary of elements: b, i, k, e, r, l
    - Iter1
      - Form all possible pairs of 2 elements from the vocabulary and that are on consecutive position in the original text
        - b + i, i + k, k + e, e + r, l + i
      - Compute the score = the frequency of each pair
        - b + i → 2, i + k → 3, k + e -> 3, e + r → 1, l + i → 1
      - Greedy select the pair of the maximum score
        - b + i → 2, **i + k → 3**, k + e -> 3, e + r → 1, l + i → 1
      - Merge the 2 elements, store the merging rule, add the new element into the vocabulary
        - Vocab: b, i, k, e, r, l, **ik**
      - Update the segmented text based on the merge rule
        - (b, i, k, e), (b, i, k, e, r), (l, i, k, e) → (b, ik, e), (b, ik, e, r), (l, ik, e)
    - Iter2
      - Form all possible pairs of 2 elements from the vocabulary and that are on consecutive position in the original text
        - b + ik, ik + e, e + r, l + ik
      - Compute the score = the frequency of each pair
        - b + ik → 2, ik + e → 3, e + r → 1, l + ik → 1
      - Greedy select the pair of the maximum score
        - b + ik → 2, **ik + e → 3**, e + r → 1, l + ik → 1
      - Merge the 2 elements, store the merging rule, add the new element into the vocabulary
        - Vocab: b, i, k, e, r, l, ik, **ike**
      - Update the segmented text based on the merge rule
        - (b, ik, e), (b, ik, e, r), (l, ik, e) → (b, ike), (b, ike, r), (l, ike)
    - Iter3
      - Form all possible pairs of 2 elements from the vocabulary and that are on consecutive position in the original text
        - b + ike, ike + r, l + ike
      - Compute the score = the frequency of each pair
        - b + ike → 2, ike + r → 1, l + ike → 1
      - Greedy select the pair of the maximum score
        - **b + ike → 2**, ike + r → 1, l + ike → 1
      - Merge the 2 elements, store the merging rule, add the new element into the vocabulary
        - Vocab: b, i, k, e, r, l, ik, ike, **bike**
      - Update the segmented text based on the merge rule
      - (b, ike), (b, ike, r), (l, ike) → (bike), (bike, r), (l, ike)
    - Tokenize a new text:
      - Segment the text in iniatial elements (chars) and apply the "learnt" rules
      - E.g. biking
        - Start: b, i, k, l, n, g
        - First rule: b, ik, i, n, g
        - Second rule: b, ik, i, n, g
        - Third rule: b, ik, i, n, g
      - E.g. bikelane
        - Start: b, i, k, e, l, a, n, e
        - First rule: b, ik, e, l, a, n, e
        - Second rule: b, ike, l, a, n, e
        - Third rule: bike, l, a, n, e
    - Homework:
      - Repeat the training and the tokenization for the same two words by using as training data the text: {bike, biker, like, biking, riding, bike, running)}

BPE – 2016

- Text tokenization
  - Dictionary
    - union of the subwords / tokens of all words
    - Larger than for original 1-hot representation from Word2Vec → more parameters
  - Representation
    - Each word → 1-hot encoding for the extended dictionary
- Feature computation
  - Apply skip-gram model principles for training the weights
  - Each word is represented by the sum of its subword embeddings.
- Advantages
  - BPE works best with languages that build long words from smaller ones. It's not as great for languages with simple word structures.
  - Scalability → Sequences of longer length can be encoded succinctly
  - Improved Efficiency for Large Datasets since the vocabulary size can be reduced significantly.
  - Flexibility → by handling of Rare Wordsand Out-of-Vocabulary Words → Better Handling of Domain-Specific Language (medical, legal, genomics texts etc.)
  - Efficient Vocabulary Size: BPE creates a fixed-size vocabulary of subword units, balancing between character-level and word-level representations.
- Weaknesses
  - Computational Complexity
- More details
  - Sennrich, R. (2015). Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909 link
  - BPE as data compression algorithm → Gage, P. (1994). A new algorithm for data compression. The C Users Journal, 12(2), 23-38. link
  - Byte-level BPE – GPT2, RoBERTa don't look at words as being written with Unicode characters, but with bytes. This way the base vocabulary has a small size (256), but every character you can think of will still be included and not end up being converted to the unknown token.
  - GPT-2 starts with 256 bytes base tokens, a special end-of-text token and the symbols learnt during 50 000 merges → 50 257 tokens
  - Comparison of more tokenization methods - Bostrom, K., & Durrett, G. (2020). Byte pair encoding is suboptimal for language model pretraining. arXiv preprint arXiv:2004.03720 link
  - HuggingFace docs https://huggingface.co/learn/nlp-course/chapter6/5

WordPiece – 2012

- Text tokenization
  - Similar to BPE - starts with individual characters and merges the most frequent pairs. It merges based on both frequency and likelihood of the formed subword
  - Process of tokenization = building with Legos (from tiny pieces to bigger ones by various combinations)
    - Token learning = raw training text → learnt vocabulary (set of tokens)
      - Begin with single characters
      - Find the pair that maximizes the likelihood of the training data once added to the vocabulary
        - Greedy search → deterministic algorithm
      - Merge that pair into a new token
      - Repeat until you hit your token limit
    - Segmenting / Processing = raw test text → list of tokens (based on the vocabulary)
      - Apply the merge rules learned in order on the individual characters of the given text

WordPiece – 2012

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and ##char): (b, ##i, ##k, ##e), (b, ##i, ##k, ##e, ##r), (l, ##i, ##k, ##e)
    - Vocabulary of elements: b, ##i, ##k, ##e, ##r, l
    - Iter1
      - Form all possible pairs of 2 elements from the vocabulary and that are on consecutive position in the original text
        - b + ##i, ##i + ##k, ##k + ##e, ##e + ##r, l + ##i
      - Compute the score = the frequency of each pair / (frequency of the first element * frequency of the second element
        - b + ##i → 2 / (2 + 3), ##i + ##k → 3 / (3 + 3), ##k + ##e → 3 / (3 + 3), ##e + ##r → 1 / (3 + 1), l + ##i → 1 / (1 + 3)
        - b + ##i → 0.4, ##i + ##k → 0.5, ##k + ##e → 0.5, ##e + ##r → 0.25, l + ##i → 0.25
      - Greedy select the pair of the maximum score
        - b + ##i → 0.4, **##i + ##k → 0.5**, ##k + ##e → 0.5, ##e + ##r → 0.25, l + ##i → 0.25
      - Merge the 2 elements, store the merging rule, add the new element into the vocabulary
        - Vocab: b, i, k, e, r, l, **##ik**
      - Update the segmented text based on the merge rule
        - (b, ##i, ##k, ##e), (b, ##i, ##k, ##e, ##r), (l, ##i, ##k, ##e) → (b, ##ik, ##e), (b, ##ik, ##e, ##r), (l, ##ik, ##e)
    - Iter2
      - Form all possible pairs of 2 elements from the vocabulary and that are on consecutive position in the original text
        - b + ##ik, ##ik + ##e, ##e + ##r, l + ##ik
      - Compute the score = the frequency of each pair / (frequency of the first element * frequency of the second element
        - b + ##ik → 2 / (2 + 3), ##ik + ##e → 3 / (3 + 2), ##e + ##r → 1 / (2 + 1), l + ##ik → 1 / (1 + 3)
        - b + ##ik → 0.4, ##ik + ##e → 0.6, ##e + ##r → 0.33, l + ##ik → 0.25
      - Greedy select the pair of the maximum score
        - b + ##ik → 0.4, **##ik + ##e → 0.6,** ##e + ##r → 0.33, l + ##ik → 0.25
      - Merge the 2 elements, store the merging rule, add the new element into the vocabulary
        - Vocab: b, i, k, e, r, l, ik, **##ike**
      - Update the segmented text based on the merge rule
        - (b, ##ik, ##e), (b, ##ik, ##e, ##r), (l, ##ik, ##e) → (b, ##ike), (b, ##ike, ##r), (l, ##ike)

WordPiece – 2012

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and ##char): (b, ##i, ##k, ##e), (b, ##i, ##k, ##e, ##r), (l, ##i, ##k, ##e)
    - Vocabulary of elements: b, ##i, ##k, ##e, ##r, l
    - Iter3
      - Form all possible pairs of 2 elements from the vocabulary and that are on consecutive position in the original text
        - b + ##ike, ##ike + ##r, l + ##ike
      - Compute the score = the frequency of each pair
        - b + ##ike → 2 / (2 + 3), ##ike + ##r → 1 / (3 +1), l + ##ike → 1 / (1 + 3)
        - b + ##ike → 0.4, ##ike + ##r → 0.25, l + ##ike → 0.25
      - Greedy select the pair of the maximum score
        - **b + ##ike → 0.4**, ##ike + ##r → 0.25, l + ##ike → 0.25
      - Merge the 2 elements, store the merging rule, add the new element into the vocabulary
        - Vocab: b, i, k, e, r, l, ik, ike, **bike**
      - Update the segmented text based on the merge rule
        - (b, ##ike), (b, ##ike, ##r), (l, ##ike) → (bike), (bike, ##r), (l, ##ike)
  - Tokenize a new text:
    - Search for the longest elements of the vocabulary that the text start with
    - E.g. biking
      - Vocab: b, i, k, e, r, l, ik, ike, bike
      - biking → [b, iking]
      - Iking → [ik, ing]
      - ing → [i, ng]
      - **biking → [b, ik, i, ng]**
    - E.g. bikelane
      - Vocab: b, i, k, e, r, l, ik, ike, bike
      - bikelane → [bike, lane]
      - lane -> l, ane
      - **bikelane → [bike, l, ane]**
  - Homework:
    - Repeat the training and the tokenization for the same two words by using as training data the text: {bike, biker, like, biking, riding, bike, running)}

WordPiece – 2012

- Text tokenization
  - Similar to BPE - starts with individual characters and merges the most frequent pairs. It merges based on both frequency and likelihood of the formed subword
  - E.g.
    - …
  - Dictionary
    - union of the subwords / tokens of all words
    - Larger than for original 1-hot representation from Word2Vec → more parameters
  - Representation
    - Each word → 1-hot encoding for the extended dictionary
- Feature computation
  - Apply skip-gram model principles for training the wei…
  - Each word is represented by the sum of its su…ord embeddings.
- Advantages
  - WordPiece works best when:
    - The training data is fixed
    - New data is similar to the training set
    - dealing with morphologically rich languages
- Weaknesses
  - Spelling mistakes can cause a massive decrease in model performance - Kumar, A., Makhija, P., & Gupta, A. (2020). Noisy text data: Achilles' heel of BERT. *arXiv preprint arXiv:2003.12932*. link
  - **WordPeice fails when it gets the word boundaries wrong (because it is greedy)**
- More details
  - Schuster, M., & Nakajima, K. (2012, March). Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 5149-5152). IEEE. Link
  - Pro & cons for BPE vs WordPiece https://medium.com/@atharv6f_47401/wordpiece-tokenization-a-bpe-variant-73cc48865cbf
  - TEnsorFlow https://www.tensorflow.org/text/guide/subwords_tokenizer#optional_the_algorithm
  - HuggingFace docs https://huggingface.co/learn/nlp-course/en/chapter6/6?fw=pt

"WordPiece ensures that the most common words are represented in the vocabulary as a single token while rare words are broken down into smaller subword tokens." - Schuster et al., 2012

Aspect	WordPiece	BPE
Selection	Maximizes data likelihood	Picks most frequent pair
Optimization	Dataset-specific	More general
Vocab Size	Usually smaller	Often larger
Training	Faster convergence	Can take longer

Unigram – 2018

- Text tokenization
  - uses a probabilistic model to determine the likelihood of subword sequences and selects the most probable subword segmentation
  - Process of tokenization = starts from a big vocabulary and removes tokens from it until it reaches the desired vocabulary size
    - Token learning = raw training text → learnt vocabulary (set of tokens)
      - Start with a large initial vocabulary that includes [all] possible subwords from the corpus → individual characters, words, and subwords.
      - Frequency Calculation: Calculate the frequency of each subword in the initial vocabulary based on the training corpus.
      - Loss Calculation: For each subword, compute the loss, which is a measure of how much the overall likelihood of the corpus decreases if that subword is removed from the vocabulary.
        - Loss = Σ freq * (- log(P(word))
      - Pruning: Remove the subwords that contribute the least to the overall likelihood (i.e., those with the smallest impact on the loss). This step is repeated iteratively until the desired vocabulary size is reached.
        - computes a loss over the corpus given the current vocabulary. Then, for each symbol in the vocabulary, the algorithm computes how much the overall loss would increase if the symbol was removed, and looks for the symbols that would increase it the least. Those symbols have a lower effect on the overall loss over the corpus, so in a sense they are “less needed” and are the best candidates for removal.
      - Repeat until you hit your token limit
    - Segmenting / Processing = raw test text → list of tokens (based on the vocabulary)
      - by selecting the most probable sequence of subwords that compose each word. This is typically done using a dynamic programming algorithm like the Viterbi algorithm.

Unigram – 2018

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and bigrams): (b, i, k, e), (b, i, k, e, r), (l, i, k, e)
    - Vocabulary of elements: b, i, k, e, r, l, bi, ik, ke, er, li

    Iter1
    - Frequencies: b → 2, i → 3, k → 3, e → 3, r → 1, l → 1, bi → 2, ik → 3, ke → 3, er → 1, li → 1, sum of all frequencies = 23
    - For all the words, identify the segmentation of maximum probability
      - bike →
        - P([b, i, k ,e]) = 2 / 23 * 3 / 23 * 3 / 23 * 3 / 23 = (2 * 3 *3 * 3) / 23^4
        - P([bi, k, e]) = 2 / 23 * 3 / 23 * 3 / 23 = (2 * 3 * 3) / 23^3
        - P([b, ik, e]) = 2 / 23 * 3 / 23 * 3 / 23 = (2 * 3 * 3) / 23^3
        - P([b, i, ke]) = 2 / 23 * 3 /23 * 3 / 23 = (2 * 3 * 3) / 23^3
        - **P([bi, ke]) = 2/ 23 * 3 / 23 = (2 * 3) / 23^2**
      - Biker →
        - P([b, i, k ,e, r]) = 2 / 23 * 3 / 23 * 3 / 23 * 3 / 23 * 1 / 23= (2 * 3 *3 * 3 * 1) / 23^5
        - P([bi, k, e, r]) = 2 / 23 * 3 / 23 * 3 / 23 * 1 / 23 = (2 * 3 * 3 * 1) / 23^4
        - P([b, ik, e, r]) = 2 / 23 * 3 / 23 * 3 / 23 * 1 / 23 = (2 * 3 * 3 * 1) / 23^4
        - P([b, i, ke, r]) = 2 / 23 * 3 /23 * 3 / 23 * 1 / 23= (2 * 3 * 3 * 1) / 23^4
        - P([b, i, k, er]) = 2 / 23 * 3 /23 * 3 / 23 * 1 / 23= (2 * 3 * 3 * 1) / 23^4
        - **P([bi, ke, r]) = 2/ 23 * 3 / 23 * 1 / 23 = (2 * 3 * 1) / 23^3**
        - P([bi, k, er]) = 2/ 23 * 3 / 23 * 1 / 23 = (2 * 3 * 1) / 23^3
        - P([b, ik, er]) = 2/ 23 * 3 / 23 * 1 / 23 = (2 * 3 * 1) / 23^3
      - like
        - P([l, i, k, e]) = = 1 / 23 * 3 / 23 * 3 / 23 * 3 / 23 = (1 * 3 *3 * 3) / 23^4
        - P([li, k, e]) = 1 / 23 * 3 / 23 * 3 / 23 = (1 * 3 * 3) / 23^3
        - P([l, ik, e]) = 1 / 23 * 3 / 23 * 3 / 23 = (1 * 3 * 3) / 23^3
        - P([l, i, ke]) = 1 / 23 * 3 /23 * 3 / 23 = (1 * 3 * 3) / 23^3
        - **P([li, ke]) = 1 / 23 * 3 / 23 = (1 * 3) / 23^2**
    - Loss computation
      - Loss = Σ freq * (- log(P(word)) = 1 * (- log(6 / 23^2) + 1 * (- log(6 / 23^3) + 1 * (- log(3 / 23^2) = 7.589

		V					
Freq	Word	Score					
1	bike	0.0019					
1	biker	0.0006					
1	like	0.0068					
Loss		7.589					

Unigram – 2018

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and bigrams): (b, i, k, e), (b, i, k, e, r), (l, i, k, e)
    - Vocabulary of elements: b, i, k, e, r, l, bi, ik, ke, er, li

    Iter2
    - Frequencies: b → 2, i → 3, k → 3, e → 3, r → 1, l → 1, **bi → 0**, ik → 3, ke → 3, er → 1, li → 1, sum of all frequencies = 21
    - For all the words, identify the segmentation of maximum probability
      - Bike →
        - P([b, i, k ,e]) = 2 / 21 * 3 / 21 * 3 / 21 * 3 / 21 = $(2 * 3 * 3 * 3) / 21^4$
        - **P([bi, k, e]) = 0 / 21 * 3 / 21 * 3 / 21 = 0**
        - **P([b, ik, e]) = 2 / 21 * 3 / 21 * 3 / 21 = $(2 * 3 * 3) / 21^3$**
        - P([b, i, ke]) = 2 / 21 * 3 /21 * 3 / 21 = $(2 * 3 * 3) / 21^3$
        - **P([bi, ke]) = 0 / 21 * 3 / 21 = 0**
      - Biker →
        - P([b, i, k ,e, r]) = 2 / 21 * 3 / 21 * 3 / 21 * 3 / 21 * 1 / 21= $(2 * 3 * 3 * 3 * 1) / 21^5$
        - **P([bi, k, e, r]) = 0 / 21 * 3 / 21 * 3 / 21 * 1 / 21 = $(2 * 3 * 3 * 1) / 21^4$**
        - P([b, ik, e, r]) = 2 / 21 * 3 / 21 * 3 / 21 * 1 / 21 = $(2 * 3 * 3 * 1) / 21^4$
        - P([b, i, ke, r]) = 2 / 21 * 3 /21 * 3 / 21 * 1 / 21= $(2 * 3 * 3 * 1) / 21^4$
        - P([b, i, k, er]) = 2 / 21 * 3 /21 * 3 / 21 * 1 / 21= $(2 * 3 * 3 * 1) / 21^4$
        - **P([bi, ke, r]) = 0 / 21 * 3 / 21 * 1 / 21 = 0**
        - **P([bi, k, er]) = 0 / 21 * 3 / 21 * 1 / 21 = 0**
        - **P([b, ik, er]) = 2/ 21 * 3 / 21 * 1 / 21 = $(2 * 3 * 1) / 21^3$**
      - like
        - P([l, i, k, e]) = = 1 / 21 * 3 / 21 * 3 / 21 * 3 / 21 = $(1 * 3 * 3 * 3) / 21^4$
        - P([li, k, e]) = 1 / 21 * 3 / 21 * 3 / 21 = $(1 * 3 * 3) / 21^3$
        - P([l, ik, e]) = 1 / 21 * 3 / 21 * 3 / 21 = $(1 * 3 * 3) / 21^3$
        - P([l, i, ke]) = 1 / 21 * 3 /21 * 3 / 21 = $(1 * 3 * 3) / 21^3$
        - **P([li, ke]) = 1 / 21 * 3 / 21 = $(1 * 3) / 21^2$**
    - Loss computation
      - Loss = Σ freq * (- log(P(word)) = 1 * (- log($18 / 21^3$) + 1 * (- log($6 / 21^3$) + 1 * (- log($3 / 21^2$) = 8.05

		V	V – {bi}				
Freq	Word	Score	Score				
1	bike	0.0019	0.0113				
1	biker	0.0006	0.0004				
1	like	0.0068	0.0056				
Loss		7.589	8.050				

Unigram – 2018

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and bigrams): (b, i, k, e), (b, i, k, e, r), (l, i, k, e)
    - Vocabulary of elements: b, i, k, e, r, l, bi, ik, ke, er, li

    Iter3
    - Frequencies: b → 2, i → 3, k → 3, e → 3, r → 1, l → 1, bi → 2, **ik → 0**, ke → 3, er → 1, li → 1, sum of all frequencies = 20
    - For all the words, identify the segmentation of maximum probability
      - Bike →
        - P([b, i, k ,e]) = 2 / 20 * 3 / 20 * 3 / 20 * 3 / 20 = (2 * 3 *3 * 3) / 20^4
        - P([bi, k, e]) = 2 / 20 * 3 / 20 * 3 / 20 = (2 * 3 * 3) / 20^3
        - **P([b, ik, e]) = 2 / 20 * 0 / 20 * 3 / 20 = 0**
        - P([b, i, ke]) = 2 / 20 * 3 /20 * 3 / 20 = (2 * 3 * 3) / 20^3
        - **P([bi, ke]) = 2 / 20 * 3 / 20 = (2 * 3) / 20^2**
      - Biker →
        - P([b, i, k ,e, r]) = 2 / 20 * 3 / 20 * 3 / 20 * 3 / 20 * 1 / 20= (2 * 3 *3 * 3 * 1) / 20^5
        - P([bi, k, e, r]) = 2 / 20 * 3 / 20 * 3 / 20 * 1 / 20 = (2 * 3 * 3 * 1) / 20^4
        - **P([b, ik, e, r]) = 2 / 20 * 0 / 20 * 3 / 20 * 1 / 20 = 0**
        - P([b, i, ke, r]) = 2 / 20 * 3 /20 * 3 / 20 * 1 / 20= (2 * 3 * 3 * 1) / 20^4
        - P([b, i, k, er]) = 2 / 20 * 3 /20 * 3 / 20 * 1 / 20= (2 * 3 * 3 * 1) / 20^4
        - **P([bi, ke, r]) = 2/ 20 * 3 / 20 * 1 / 20 = (2 * 3 * 1) / 20^3**
        - P([bi, k, er]) = 2/ 20 * 3 / 20 * 1 / 20 = (2 * 3 * 1) / 20^3
        - **P([b, ik, er]) = 2/ 20 * 0 / 20 * 1 / 20 = 0**
      - like
        - P([l, i, k, e]) = = 1 / 20 * 3 / 20 * 3 / 20 * 3 / 20 = (1 * 3 *3 * 3) / 20^4
        - P([li, k, e]) = 1 / 20 * 3 / 20 * 3 / 20 = (1 * 3 * 3) / 20^3
        - **P([l, ik, e]) = 1 / 20 * 0 / 20 * 3 / 20 = 0**
        - P([l, i, ke]) = 1 / 20 * 3 /20 * 3 / 20 = (1 * 3 * 3) / 20^3
        - **P([li, ke]) = 1 / 20 * 3 / 20 = (1 * 3) / 20^2**
    - Loss computation
      - Loss = Σ freq * (- log(P(word)) = 1 * (- log(6 / 20^2) + 1 * (- log(6 / 20^3) + 1 * (- log(3 / 20^2) = 7.071

		V	V – {bi}	V – {ik}			
Freq	Word	Score	Score	Score			
1	bike	0.0019	0.0113	0.0150			
1	biker	0.0006	0.0004	0.0007			
1	like	0.0068	0.0056	0.0075			
Loss		7.589	8.050	7.071			

Unigram – 2018

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and bigrams): (b, i, k, e), (b, i, k, e, r), (l, i, k, e)
    - Vocabulary of elements: b, i, k, e, r, l, bi, ik, ke, er, li

    Iter4
    - Frequencies: b → 2, i → 3, k → 3, e → 3, r → 1, l → 1, bi → 2, ik → 3, **ke → 0**, er → 1, li → 1, sum of all frequencies = 20
    - For all the words, identify the segmentation of maximum probability
      - Bike →
        - P([b, i, k ,e]) = 2 / 20 * 3 / 20 * 3 / 20 * 3 / 20 = (2 * 3 *3 * 3) / 20^4
        - **P([bi, k, e]) = 2 / 20 * 3 / 20 * 3 / 20 = (2 * 3 * 3) / 20^3**
        - P([b, ik, e]) = 2 / 20 * 0 / 20 * 3 / 20 = (2 * 3 * 3) / 20^3
        - **P([b, i, ke]) = 2 / 20 * 3 /20 * 0 / 20 = 0**
        - **P([bi, ke]) = 2 / 20 * 0 / 20 = 0**
      - Biker →
        - P([b, i, k ,e, r]) = 2 / 20 * 3 / 20 * 3 / 20 * 3 / 20 * 1 / 20= (2 * 3 *3 * 3 * 1) / 20^5
        - P([bi, k, e, r]) = 2 / 20 * 3 / 20 * 3 / 20 * 1 / 20 = (2 * 3 * 3 * 1) / 20^4
        - P([b, ik, e, r]) = 2 / 20 * 3 / 20 * 3 / 20 * 1 / 20 = (2 * 3 * 3 * 1) / 20^4
        - **P([b, i, ke, r]) = 2 / 20 * 3 /20 * 0 / 20 * 1 / 20 = 0**
        - P([b, i, k, er]) = 2 / 20 * 3 /20 * 3 / 20 * 1 / 20 = (2 * 3 * 3 * 1) / 20^4
        - **P([bi, ke, r]) = 2/ 20 * 0 / 20 * 1 / 20 = 0**
        - **P([bi, k, er]) = 2/ 20 * 3 / 20 * 1 / 20 = (2 * 3 * 1) / 20^3**
        - P([b, ik, er]) = 2/ 20 * 3 / 20 * 1 / 20 = (2 * 3 * 1) / 20^3
      - like
        - P([l, i, k, e]) = = 1 / 20 * 3 / 20 * 3 / 20 * 3 / 20 = (1 * 3 *3 * 3) / 20^4
        - **P([li, k, e]) = 1 / 20 * 3 / 20 * 3 / 20 = (1 * 3 * 3) / 20^3**
        - P([l, ik, e]) = 1 / 20 * 3 / 20 * 3 / 20 = (1 * 3 * 3) / 20^3
        - **P([l, i, ke]) = 1 / 20 * 3 /20 * 0 / 20 = 0**
        - **P([li, ke]) = 1 / 20 * 0 / 20 = 0**
    - Loss computation
      - Loss = Σ freq * (- log(P(word)) = 1 * (- log(18 / 20^3) + 1 * (- log(6 / 20^3) + 1 * (- log(9 / 20^3) = 8.719

		V	V – {bi}	V – {ik}	V – {ke}		
Freq	Word	Score	Score	Score	Score		
1	bike	0.0019	0.0113	0.0150	0.0022		
1	biker	0.0006	0.0004	0.0007	0.0007		
1	like	0.0068	0.0056	0.0075	0.0011		
Loss		7.589	8.050	7.071	8.719		

Unigram – 2018

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and bigrams): (b, i, k, e), (b, i, k, e, r), (l, i, k, e)
    - Vocabulary of elements: b, i, k, e, r, l, bi, ik, ke, er, li

    Iter5
    - Frequencies: b → 2, i → 3, k → 3, e → 3, r → 1, l → 1, bi → 2, ik → 3, ke → 3, **er → 0**, li → 1, sum of all frequencies = 22
    - For all the words, identify the segmentation of maximum probability
      - Bike →
        - P([b, i, k ,e]) = 2 / 22 * 3 / 22 * 3 / 22 * 3 / 22 = (2 * 3 *3 * 3) / 22^4
        - P([bi, k, e]) = 2 / 22 * 3 / 22 * 3 / 22 = (2 * 3 * 3) / 22^3
        - P([b, ik, e]) = 2 / 22 * 3 / 22 * 3 / 22 = (2 * 3 * 3) / 22^3
        - P([b, i, ke]) = 2 / 22 * 3 /22 * 3 / 22 = (2 * 3 * 3) / 22^3
        - **P([bi, ke]) = 2 / 22 * 3 / 22 = (2 * 3) / 22^2**
      - Biker →
        - P([b, i, k ,e, r]) = 2 / 22 * 3 / 22 * 3 / 22 * 3 / 22 * 1 / 22= (2 * 3 *3 * 3 * 1) / 22^5
        - P([bi, k, e, r]) = 2 / 22 * 3 / 22 * 3 / 22 * 1 / 22 = (2 * 3 * 3 * 1) / 22^4
        - P([b, ik, e, r]) = 2 / 22 * 3 / 22 * 3 / 22 * 1 / 22 = (2 * 3 * 3 * 1) / 22^4
        - P([b, i, ke, r]) = 2 / 22 * 3 /22 * 3 / 22 * 1 / 22 = (2 * 3 * 3 * 1) / 22^4
        - **P([b, i, k, er]) = 2 / 22 * 3 /22 * 3 / 22 * 0 / 22 = 0**
        - **P([bi, ke, r]) = 2/ 22 * 3 / 22 * 1 / 22 = (2 * 3 * 1) / 22^3**
        - **P([bi, k, er]) = 2/ 22 * 3 / 22 * 0 / 22 = 0**
        - **P([b, ik, er]) = 2/ 22 * 3 / 22 * 0 / 22 = 0**
      - like
        - P([l, i, k, e]) = = 1 / 22 * 3 / 22 * 3 / 22 * 3 / 22 = (1 * 3 *3 * 3) / 22^4
        - P([li, k, e]) = 1 / 22 * 3 / 22 * 3 / 22 = (1 * 3 * 3) / 22^3
        - P([l, ik, e]) = 1 / 22 * 3 / 22 * 3 / 22 = (1 * 3 * 3) / 22^3
        - P([l, i, ke]) = 1 / 22 * 3 /22 * 3 / 22 = (1 * 3 * 3) / 22^3
        - **P([li, ke]) = 1 / 22 * 3 / 22 = (1 * 3) / 22^2**
    - Loss computation
      - Loss = Σ freq * (- log(P(word)) = 1 * (- log(6/ 22^2) + 1 * (- log(6 / 22^3) + 1 * (- log(3 / 22^2) = 7.362

		V	V – {bi}	V – {ik}	V – {ke}	V – {er}	
Freq	Word	Score	Score	Score	Score	Score	
1	bike	0.0019	0.0113	0.0150	0.0022	0.0123	
1	biker	0.0006	0.0004	0.0007	0.0007	0.0005	
1	like	0.0068	0.0056	0.0075	0.0011	0.0061	
Loss		7.589	8.050	7.071	8.719	7.362	

Unigram – 2018

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and bigrams): (b, i, k, e), (b, i, k, e, r), (l, i, k, e)
    - Vocabulary of elements: b, i, k, e, r, l, bi, ik, ke, er, li

    Iter6
    - Frequencies: b → 2, i → 3, k → 3, e → 3, r → 1, l → 1, bi → 2, ik → 3, ke → 3, er → 1, **li → 0,** sum of all frequencies = 22
    - For all the words, identify the segmentation of maximum probability
      - Bike →
        - P([b, i, k ,e]) = 2 / 22 * 3 / 22 * 3 / 22 * 3 / 22 = (2 * 3 *3 * 3) / 22^4
        - P([bi, k, e]) = 2 / 22 * 3 / 22 * 3 / 22 = (2 * 3 * 3) / 22^3
        - P([b, ik, e]) = 2 / 22 * 3 / 22 * 3 / 22 = (2 * 3 * 3) / 22^3
        - P([b, i, ke]) = 2 / 22 * 3 /22 * 3 / 22 = (2 * 3 * 3) / 22^3
        - **P([bi, ke]) = 2 / 22 * 3 / 22 = (2 * 3) / 22^2**
      - Biker →
        - P([b, i, k ,e, r]) = 2 / 22 * 3 / 22 * 3 / 22 * 3 / 22 * 1 / 22= (2 * 3 *3 * 3 * 1) / 22^5
        - P([bi, k, e, r]) = 2 / 22 * 3 / 22 * 3 / 22 * 1 / 22 = (2 * 3 * 3 * 1) / 22^4
        - P([b, ik, e, r]) = 2 / 22 * 3 / 22 * 3 / 22 * 1 / 22 = (2 * 3 * 3 * 1) / 22^4
        - P([b, i, ke, r]) = 2 / 22 * 3 /22 * 3 / 22 * 1 / 22 = (2 * 3 * 3 * 1) / 22^4
        - P([b, i, k, er]) = 2 / 22 * 3 /22 * 3 / 22 * 1 / 22 = (2 * 3 * 3 * 1) / 22^4
        - **P([bi, ke, r]) = 2/ 22 * 3 / 22 * 1 / 22 = (2 * 3 * 1) / 22^3**
        - P([bi, k, er]) = 2/ 22 * 3 / 22 * 1 / 22 = (2 * 3 * 1) / 22^3
        - P([b, ik, er]) = 2/ 22 * 3 / 22 * 1 / 22 = (2 * 3 * 1) / 22^3
      - like
        - P([l, i, k, e]) = = 1 / 22 * 3 / 22 * 3 / 22 * 3 / 22 = (1 * 3 *3 * 3) / 22^4
        - **P([li, k, e]) = 0 / 22 * 3 / 22 * 3 / 22 = 0**
        - **P([l, ik, e]) = 1 / 22 * 3 / 22 * 3 / 22 = (1 * 3 * 3) / 22^3**
        - P([l, i, ke]) = 1 / 22 * 3 /22 * 3 / 22 = (1 * 3 * 3) / 22^3
        - **P([li, ke]) = 0 / 22 * 3 / 22 = 0**
    - Loss computation
      - Loss = Σ freq * (- log(P(word)) = 1 * (- log(6/ 22^2) + 1 * (- log(6 / 22^3) + 1 * (- log(9 / 22^3) = 8.228

		V	V – {bi}	V – {ik}	V – {ke}	V – {er}	V – {li}
Freq	Word	Score	Score	Score	Score	Score	Score
1	bike	0.0019	0.0113	0.0150	0.0022	0.0123	0.0123
1	biker	0.0006	0.0004	0.0007	0.0007	0.0005	0.0005
1	like	0.0068	0.0056	0.0075	0.0011	0.0061	0.0008
Loss		7.589	8.050	7.071	8.719	7.362	8.228

Unigram – 2018

- Text tokenization
  - E.g. Training text = {bike, biker, like}
    - Segmented the text by using elementary units (e.g. chars and bigrams): (b, i, k, e), (b, i, k, e, r), (l, i, k, e)
    - Vocabulary of elements: b, i, k, e, r, l, bi, ik, ke, er, li
    - **Choose to eliminate the token with the minimum loss → ik**
      - Vocabulary of elements: b, i, k, e, r, l, bi, ke, er, li
  - Tokenize a new text (e.g. bike):
    - identify the segmentation of maximum probability
    - Frequencies: b → 2, i → 3, k → 3, e → 3, r → 1, l → 1, bi → 2, ke → 3, er → 1, li → 1, sum of all frequencies = 20
      - kerb →
        - P([k, e, r, b]) = 3 / 20 * 3 / 20 * 1 / 20 * 2 / 20 = (3 * 3 * 1 * 2) / 20^4
        - **P([ke, r, b]) = 3 / 20 * 1 / 20 * 2 / 20 = (3 * 1 * 2) / 20^3**
        - P([k, er, b]) = 3 / 20 * 1 / 20 * 2 / 20 = (3 * 1 * 2) / 20^3
        - P([k, e, rb]) = 3 / 20 * 3 / 20 * 0 / 20 = 0
        - P([ke, rb]) = 3 / 20 * 0 / 20 = 0
  - A faster approach for reducing the vocabulary
    - Prunning by Expectation Maximisation algorithm combined with Viterbi algorithm

		V	V – {bi}	**V – {ik}**	V – {ke}	V – {er}	V – {li}
Freq	Word	Score	Score	**Score**	Score	Score	Score
1	bike	0.0019	0.0113	**0.0150**	0.0022	0.0123	0.0123
1	biker	0.0006	0.0004	**0.0007**	0.0007	0.0005	0.0005
1	like	0.0068	0.0056	**0.0075**	0.0011	0.0061	0.0008
Loss		7.589	8.050	**7.071**	8.719	7.362	8.228

Unigram – 2018

- Text tokenization
- Feature computation
  - Apply skip-gram model principles for training the weights
  - Each word is represented by the sum of its subword embeddings.
- Advantages
  - Language-Agnostic Handle
  - Lossless tokenization
- Weaknesses
  - Tokenisation of numbers (left-to-right vs. right-to-left)
- More details
  - Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*. Link
  - https://everdark.github.io/k9/notebooks/ml/natural_language_understanding/subword_units/subword_units.nb.html#121_expectation-maximization
  - https://guillaume-be.github.io/2020-05-30/sentence_piece
  - https://huggingface.co/spaces/huggingface/number-tokenization-blog

Token-based models

Model	BPE	WordPiece	Unigram
Training	Starts from a small vocabulary and learns rules to merge tokens	Starts from a small vocabulary and learns rules to merge tokens	Starts from a large vocabulary and learns rules to remove tokens
Training step	Merges the tokens corresponding to the most common pair	Merges the tokens corresponding to the pair with the best score based on the frequency of the pair, privileging pairs where each individual token is less frequent	Removes all the tokens in the vocabulary that will minimize the loss computed on the whole corpus
Learns	Merge rules and a vocabulary	Just a vocabulary	A vocabulary with a score for each token
Encoding	Splits a word into characters and applies the merges learned during training	Finds the longest subword starting from the beginning that is in the vocabulary, then does the same for the rest of the word	Finds the most likely split into tokens, using the scores learned during training

- Integration in LLMs
  - WordPiece
    - BERT (VocabSize = 30 000 tokens)
  - BPE
    - GPT, GPT2, GPT3 (VocabSize = 175 000 tokens), GPT4, RoBERTa
  - Unigram
    - T5, AlBERT, mBART, Big Bird, XLNet

Embeddings

- Process → Text representation
  - Token-based models
    - White Space: Splits on spaces (simple but limited)
    - Word: Breaks into words (common for English)
    - Sentence: Divides text into sentences
    - Character: Splits into individual characters
    - N-gram: Creates sequences of n items/elements
      - fastText
    - Subword: Breaks words into smaller parts
      - Byte Pair Encoding (BPE): Merges common character pairs
      - WordPiece: Google's method for balancing words and subwords
      - Unigram

Method	Complexity	Best For	Drawback
White Space	Low	Simple tasks	Fails with some languages
Word	Low	General NLP	Struggles with contractions
Subword	High	Unknown words	More complex

  - **Token-free models**
    - **CharFormer**
    - **byT5**
    - **MegaByte**
    - **Byte-latent transformer**

Token-free models

- Token-free models
  - ByT5
    - https://arxiv.org/abs/2105.13626
  - Charformer
    - https://arxiv.org/abs/2106.12672
  - MegaByte (OpenAI, 2023)
    - Yu, L., Simig, D., Flaherty, C., Aghajanyan, A., Zettlemoyer, L., & Lewis, M. (2023). Megabyte: Predicting million-byte sequences with multiscale transformers. *Advances in Neural Information Processing Systems*, *36*, 78808-78823. link
    - GitHub - lucidrains/MEGABYTE-pytorch: Implementation of MEGABYTE, Predicting Million-byte Sequences with Multiscale Transformers, in Pytorch
  - SpaceByte (2024)
    - Slagle, K. (2024). SpaceByte: Towards Deleting Tokenization from Large Language Modeling. *arXiv preprint arXiv:2404.14408*. link
    - GitHub - jxiw/MambaByte: [CoLM 24] Official Repository of MambaByte: Token-free Selective State Space Model
  - Byte-latent transformer (MetaAI, 2024)
    - Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman†, Srinivasan Iyer (2024) Byte Latent Transformer: Patches Scale Better Than Tokens link
    - https://github.com/facebookresearch/blt
    - GitHub - waefrebeorn/bytropix: Combining the XJDR's Entropix Sampler and Model code with Meta's BYTE paper idea on a basic 0-255 byte demo

Byte-latent transformer

- It works directly with raw bytes (it skips tokenization step)
  - Better for new words (OOV problem)
- No predefined vocabularies
- Memory-efficient and scalable (compact design)
- Main idea
  - Dynamic patching → different representation of information (based on its complexity)
  - 3 components
    - Encoder:
      - Aim: raw bytes → dynamic patches (segments of variable length) = local byte-level representations
      - Architecture: a transformer with few layers that learns the byte embeddings
    - Global latent transformer
      - Aim: sequences of patches → features
      - Architecture: a large autoregressive transformer
    - Decoder
      - Aim: features → sequences of bytes

- Tokenisation based models
  - Fixed set of tokens (chosen offline)
  - Tokens compress bytes by heuristics methods
  - A rigid separation between raw input and the transformer
- BLT
  - Patches are formed dynamically (larger for repetitive/predictable input or smaller for unfamiliar input
  - A strong connection between raw input and the transformer
- More details
  - https://github.com/facebookresearch/blt

Embeddings

- Process
  - Text representation (pre-processing)
    - Token-based models
    - Token-free models
  - Feature computation
    - Frequency-based embeddings
      - TF-IDF
      - Co-occurrence matrix
    - Prediction based embeddings
      - Word / subword embeddings
        - **Word2vec** (Skip-gram or Continuous Bag of Words (CBOW))

Word2Vec

- Principles
- Examples
- Advantages
- Weaknesses

De ce reprezentari vectoriale ale intelesului unui cuvant/text?

- Permit determinarea similaritatii intre cuvinte/texte
  - ***fast*** is similar to ***rapid***
  - ***tall*** is similar to ***height***

Question answering:

- *Q: "How **tall** is Mt. Everest?"*
  *Candidate A: "The official **height** of Mount Everest is 29029 feet"*

Intuitia din spatele similaritatii

- Exemplu: Ce este o Sirrus X 3.0*?*

> *The **Sirrus X 3.0** invites you to explore beyond boundaries. With confidence-inspiring tires, an upright riding position, and intuitive components, **Sirrus X 3.0** is your ticket to adventure.*

- Din context, o persoana poate ghici ca Sirrus X 3.0 este un model de bicicleta

- Pentru un algoritm, intuitia e ca **doua cuvinte sunt similar daca ele sunt folosite in context similare**

Diferite reprezentari vectoriale pentru text

- Reprezentari rare (*sparse*):
  1. Mutual-information weighted word co-occurrence matrices

- Reprezentari dense (compacte) :
  2. Singular Value Decomposition (si Latent Semantic Analysis)
  3. Neural-network-inspired models (skip-grams, CBOW)
  4. Altele (e.g. brown clusters)

Vectori rari *vs*. densi

- Vectorii → matricea de co-ocurenta a termenilor
  - **lungi** (length |V|= 20,000 -> 50,000)
  - **rari** (f multe elemente sunt 0)
- Alternativa: vectori invatati (prin AI/ML)
  - scurti (length 200-1000)
  - **densi** (multe elemente nu sunt 0)
- De ce vectori densi?
  - Vectorii scurti -> folositi mai usor ca si *features* in algoritmii de invatare (mai putini coeficienti de invatat)
  - Vectorii densi pot generaliza mai bine, captand sinonimia termenilor
    - Bike – scooter
    - Car – automobile
    - House – apartment

Modele de predictive (invatare) a reprezentarilor

- **Modelul Word2vec**
  - **Skip-gram** (Mikolov et al. 2013a), **CBOW** (Mikolov et al. 2013b)
  - Se invata reprezentari, numite embeddings, ca parte din procesul de predictive/generare a textului
  - Se antreneaza o retea neuronala pentru prezicerea urmatorului cuvant
  - Avantaje
    - Rapid, simplu de antrenat
    - Modele gata antrente disponibile online

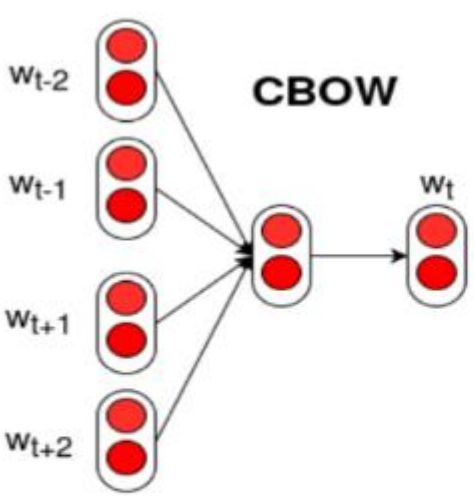
Invatare supervizata fara etichetare manuala de date

- Text: "*She rides a bike. He rides a scooter in park. My father drives a motorcycle. …*"
- Vocabular V = {*she, ride, bike, he, scooter, park, my, father, drive, motorcycle*}
- Perechi (context, cuvant)
  - E.g. context = cele 2 cuvinte precedente cuvantului curent

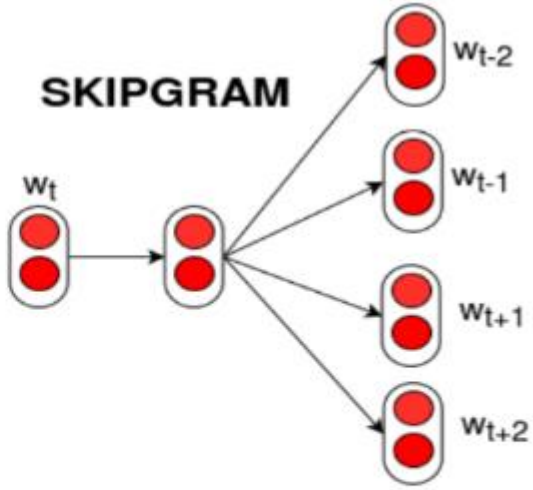
Negative sampling

[she ride] bike	1		[she ride] he	0
[ride bike] he	1		[she ride] scooter	0
[he ride] scooter	1		[she ride] park	0
[ride scooter] park	1		[she ride] my	0
[my father] drive	1		[she ride] father	0
[father drive] motorcycle	1		[she ride] drive	0
			[ride bike] scooter	0
			[ride bike] park	0
			…	
			[father drive] she	0

Arhitecturi



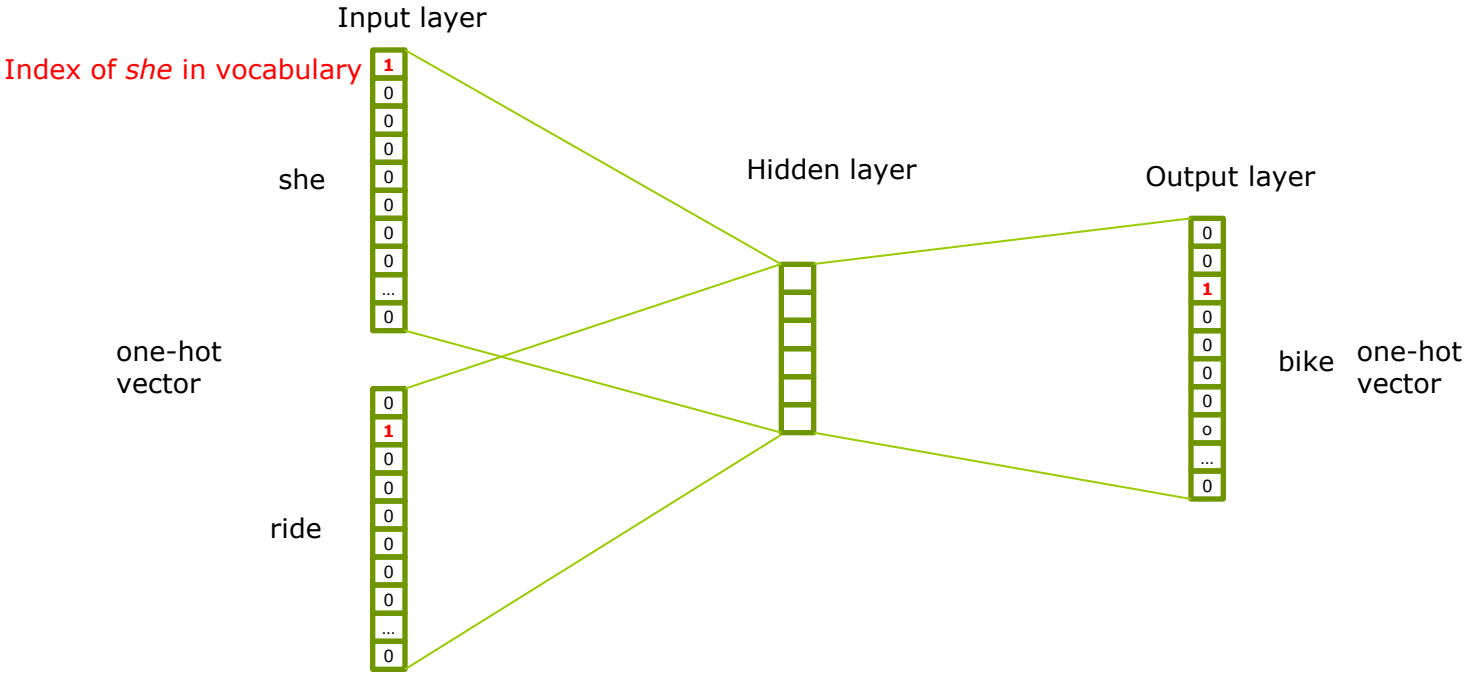
$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_t|w_{t+j})$$



$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

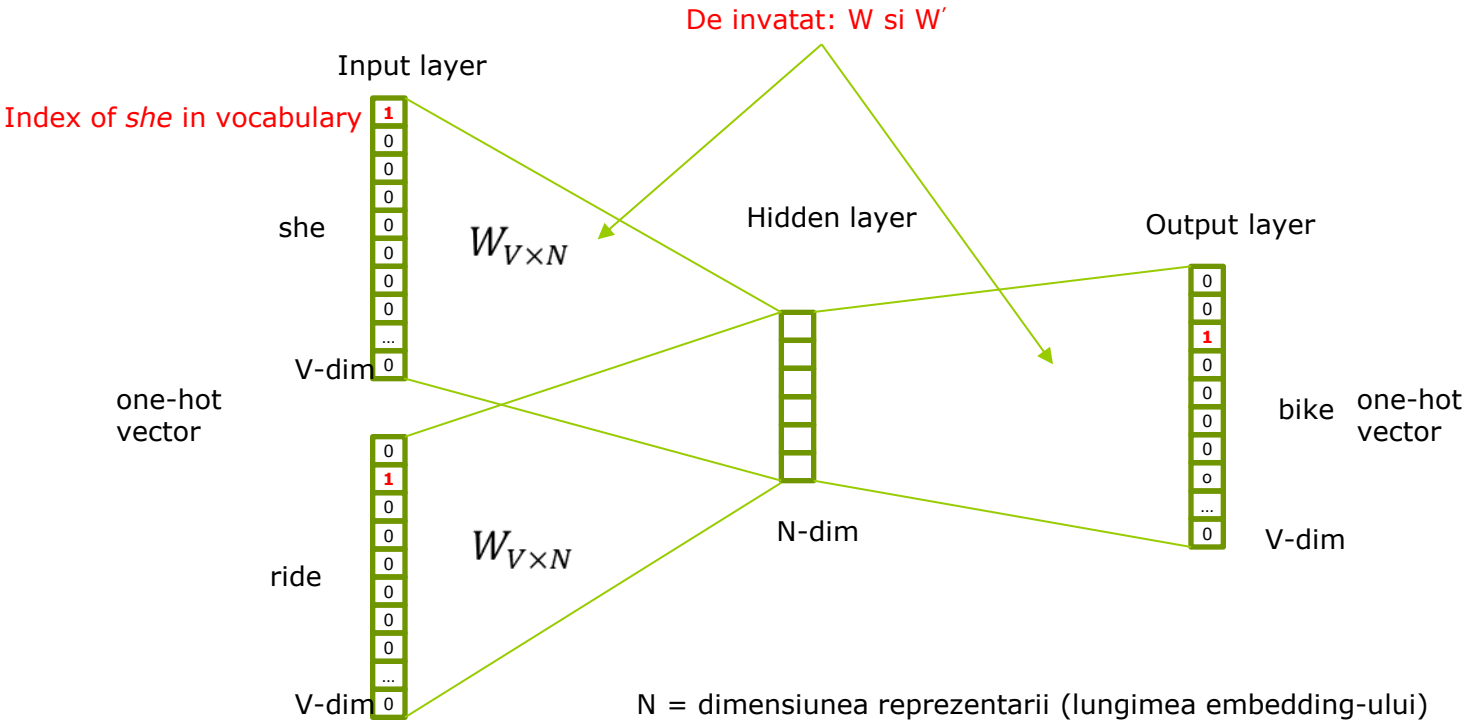
CBOW (Continuous Bag of Words)

	she	ride	bike	he	scooter	park	my	father	drive	motorcycle
She	1	0	0	0	0	0	0	0	0	0
Ride	0	1	0	0	0	0	0	0	0	0
Bike	0	0	1	0	0	0	0	0	0	0

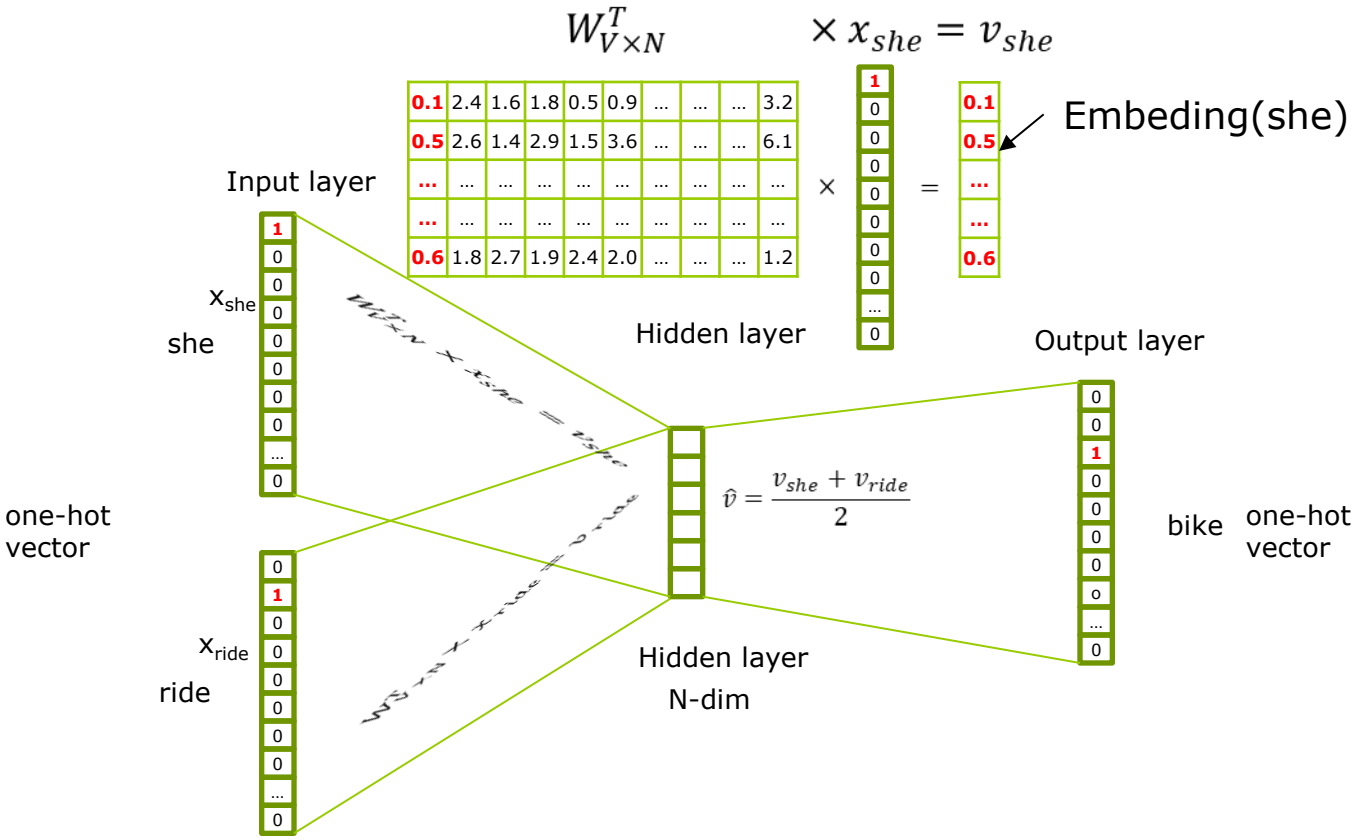


CBOW (Continuous Bag of Words)

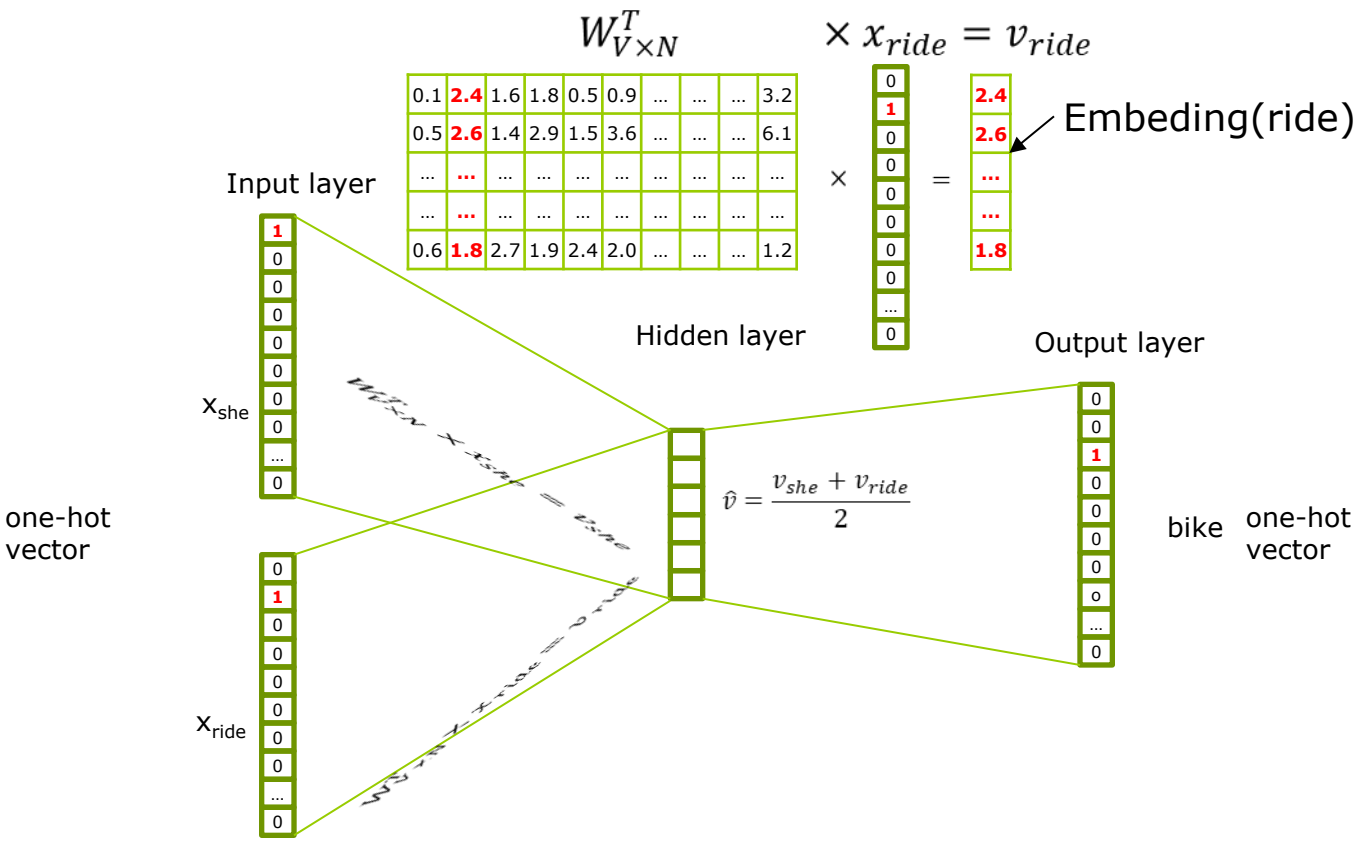
	she	ride	bike	he	scooter	park	my	father	drive	motorcycle
She	1	0	0	0	0	0	0	0	0	0
Ride	0	1	0	0	0	0	0	0	0	0
Bike	0	0	1	0	0	0	0	0	0	0



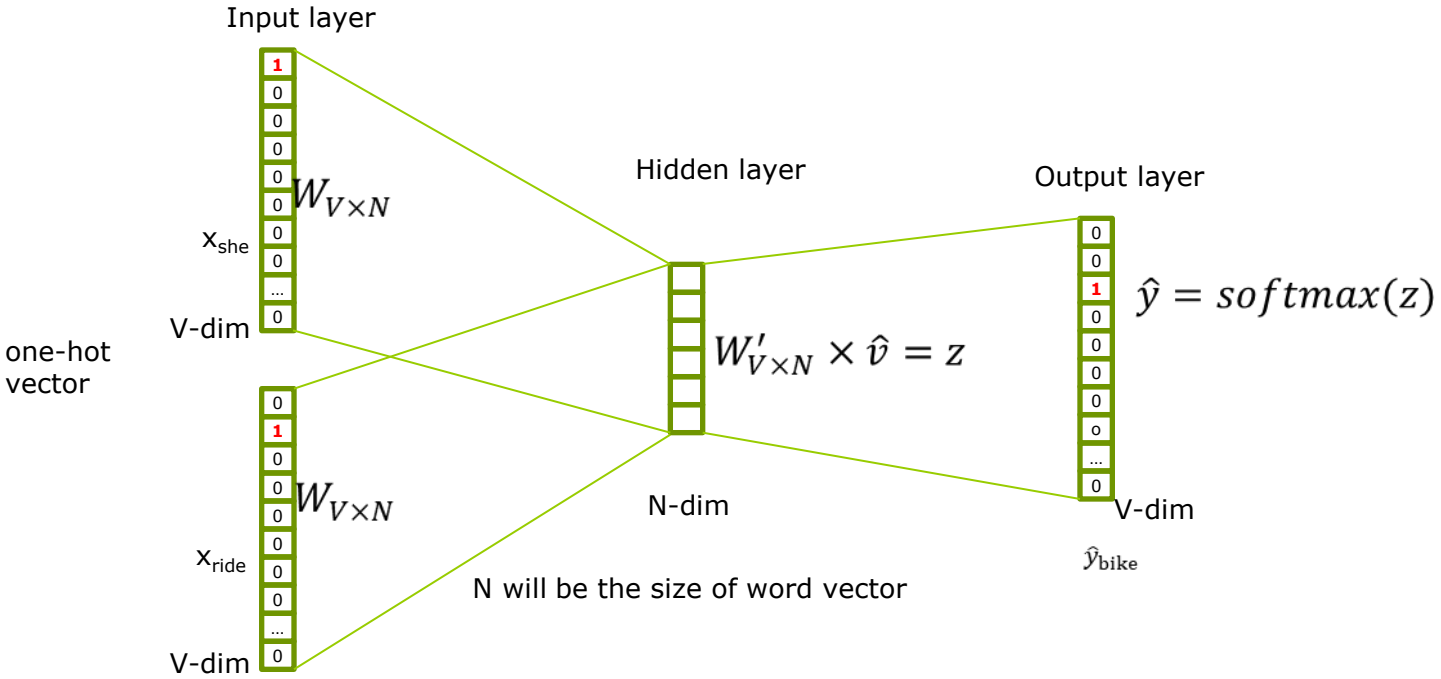
CBOW (Continuous Bag of Words)



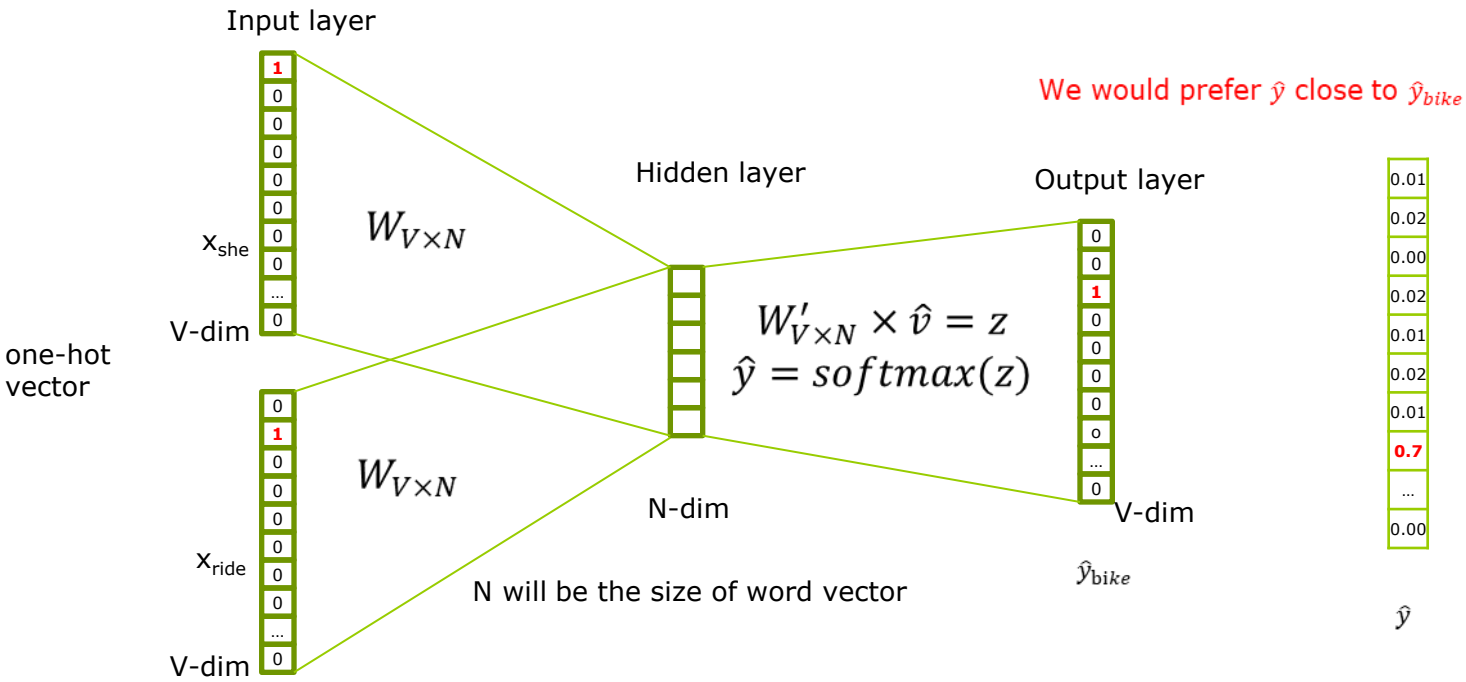
CBOW (Continuous Bag of Words)



CBOW (Continuous Bag of Words)

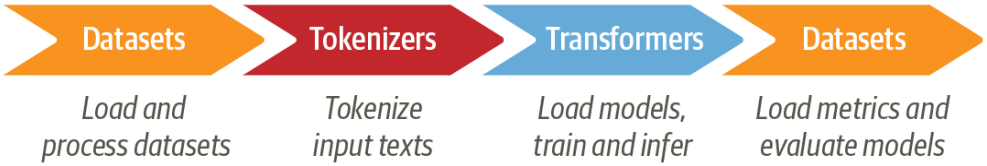


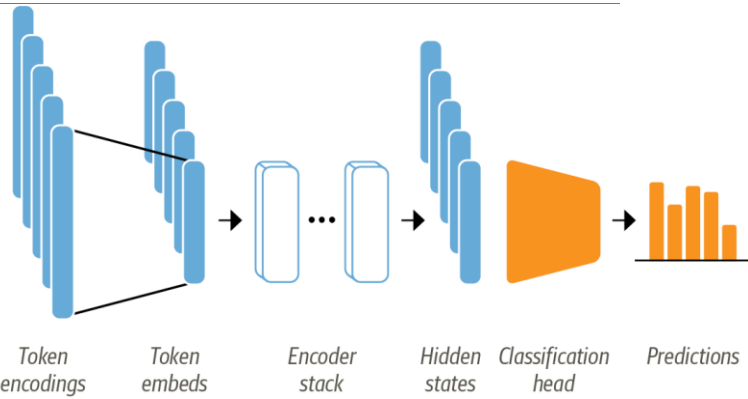
CBOW (Continuous Bag of Words)



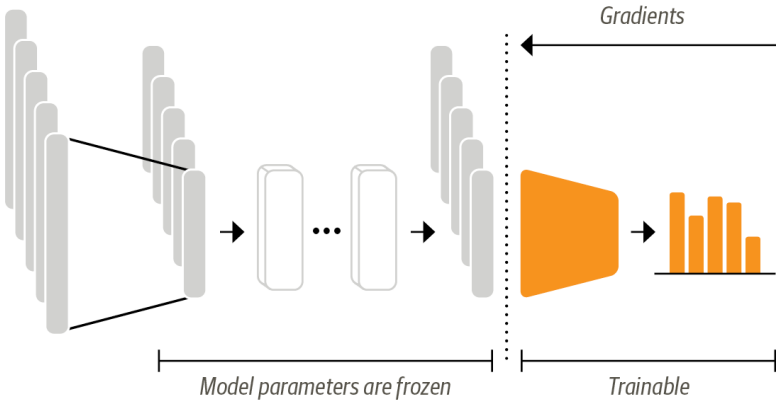
Going further to text classification

- Pipeline



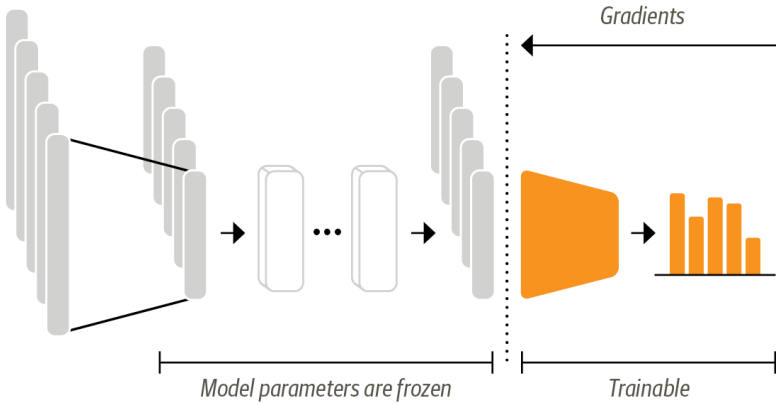


- Scenarios
  - Transformers as feature extractors

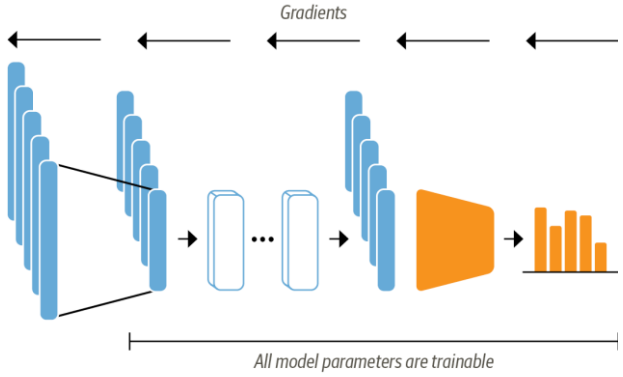


Going further to text classification

- Scenarios
  - Transformers as feature extractors

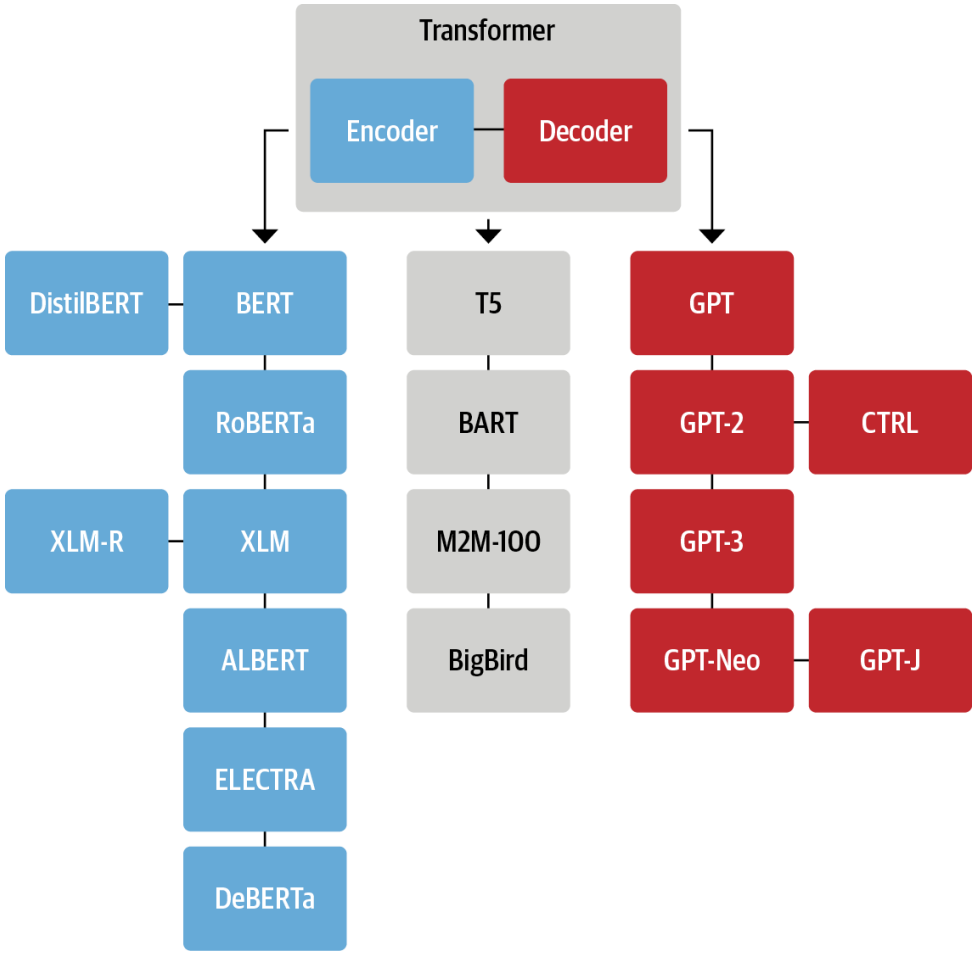


  - Fine-tuning transformers



Going further to text classification

- Architectures



Going further to text classification

- Tokenisation and Vocabulary
- Pre-training objectives
  - BERT → masked LM & Next Sentence Prediction
  - GPT → causal LM (next token prediction)
  - T5 → denoising objective (span corruption)
- Input representation
  - BERT → [CLS] + Token + [SEP] + Token + [SEP]
  - GPT → [start] + Token1 + Token2 + …
  - T5 → task prefix followed by input text
- Attention mechanism
  - BERT → absolute positional encodings and bidirectional attention (left and right)
  - GPT → absolute positional encodings and unidirectional attention (left)
  - T5 → relative positional biases
- Architecture
  - BERT → Encoder-only
  - GPT → decoder-only
  - T5 → encoder-decoder