Synthesis

Representations and language modeling

- Statistical Language Processing
    - Symbolic (rule-based): manipulation and processing of explicit symbols and rules
    - Stochastic: assign probabilities, data-driven (based on corpora), deep learning (ex.: HMM)
- Pattern matching: regular expressions: algebraic notation for characterizing a set of strings
    - Can be used to capture and substitute text
    - ELIZA: early NLP system with a series of cascade of regular expression substitution, matching and changing some part of the input lines
- Words:
    - Corpus: computer-readable collection of text
    - Word types: number of distinct words in a corpus, grouped in a set, the vocabulary (can be different word forms of a same lemma)
    - Word tokens: instances of types in the running text
- Tokenization: pre-process the text, segmenting into word tokens
    - Determines the vocabulary $\mathcal{V}$: the size of the vocabulary $|\mathcal{V}|$ has a huge impact on the cost of computation. Word normalization can be used to reduce it
    - Simplest approach: Space-based
    - Deal with punctuation
- Word normalization: standard format for words
    - Upper/lower-cases: depend on the task: information retrieval dos not need upper-cases, but information extraction does
    - Lemmatisation: represent words as their lemma, morphological parsing (necessary for some languages)
    - Stemming: crudely cutting words
- Word distance:
    - Minimum edit distance: minimum number of editing operations between two strings - Insertion, Deletion, Substitution - needed to transform one into another (adapting costs: substitutions costs 2: Levenshtein distance)
        - Less costly way to find this distance is searching for the lest costly path of edits in a huge space
        - The edit distance $D(i,j)$ between $X[1..i]$ and $Y[1..j]$
        - Algorithm for dynamic programming:
            - Initialize: $D(i,0)=i$ and $D(0,j)=j$
            - Recurrence:
                - For i from 1 to n:
                    - For j from 1 to m:

$$D(i,j)=\min\begin{cases}D(i-1,j)+1\\ D(i,j-1)+1\\ D(i-1,j-1)+\begin{cases}2 & \text{if } X[i]\neq Y[j]\\ 0 & \text{if } X[i]=Y[j]\end{cases}\end{cases}$$

- Bag of words: unordered frequency count of words in vocabulary
    - Assumes that position does not matter, hence it is indifferent to syntax and semantics, and is only lexical a feature
    - Main goal: text classification: rules based on words, classifier with word frequencies as features
    - Also useful for document clustering, information retrieval
- Naive Bayes: naively assumes independence between words
    - Goal: $\hat{c}=\underset{c\in\mathcal{C}}{\operatorname{argmax}}\mathbb{P}(c|d)$
    - Bayes rule and the independence assumption: $\hat{c}=\underset{c\in\mathcal{C}}{\operatorname{argmax}}\mathbb{P}(c)\prod_{i=1}^{n}\mathbb{P}(w_i|c)$
    - $\mathbb{P}(w|c)=\frac{\text{count}_{\mathcal{D}}(w,c)}{\sum_{w'\in\mathcal{V}}\text{count}_{\mathcal{D}}(w',c)}$
- Practice: log-probabilities (add 1 to each count)
    - Training:
        - Create the vocabulary $\mathcal{V}$ from documents $d\in\mathcal{D}$
        - From $\mathcal{D}$:
            - For each class $c\in\mathcal{C}$: compute the log-prior $\log\mathbb{P}(c)$
            - For each word $w\in\mathcal{V}$ compute the log-likelihood $\log\mathbb{P}(w|c)$
    - Inference
        - For each class $c\in\mathcal{C}$: $S(c)=\log\mathbb{P}(c)$
        - For each word $w_i\in d$: if $w_i\in\mathcal{V}$: $S(c)=S(c)+\log\mathbb{P}(w_i|c)$
        - Return $\underset{c\in\mathcal{C}}{\operatorname{argmax}}S(c)$
- Document similarity
    - Cosine similarity: $\cos(d_1,d_2)=\frac{d_1\cdot d_2}{\|d_1\|\cdot\|d_2\|}$
- TF-IDF: remove frequent, but not significant words (stop words)
    - $\text{TF}(w,d)=\log_{10}(c(w,d)+1)$, $c(w,d)$ count of word w on document d
    - $\text{IDF}(w)=\log_{10}\left(\frac{N}{\text{cd}(w)}\right)$, $\text{cd}(w)$ count of documents w appears in, N the total number of documnets
    - $\text{TF}(w,d)\cdot\text{IDF}(w)$ is the weight given to word w in document d
- Classification with Logistic regression:
    - Discriminative linear classifier: learns directly $\mathbb{P}(c|d)$ computing a linear score and applying a logistic function
    - Learn a vector w and a bias b to maximize the likelihood of making a good classification
    - Binary case: sigmoid $\mathbb{P}(c=1)=\sigma(w\cdot d+b)$
        - Extended to a multinomial case using a matrix W, a vector b and the softmax function
    - Maximize the likelihood by minimizing the cross-entropy: $L(\hat{c},c)=-\log\mathbb{P}(c|d)=-[c\log\hat{c}+(1-c)\log(1-\hat{c})]$
    - Gradient descent
- For more complex NLP tasks
    - Segmentation: segment text into lexical units (words) (many possible uses for punctuation symbol, no typographic normalization, emoticons)
    - Lexical treatment: identify words (string to linguistic unit) and their properties, normalize text and deal with new words (with the help of morphology)
    - Syntax: constraints to obtain grammatically correct sentences (validity in position and agreement)
    - Semantics: meaning of statements, lexical (meaning of words) and compositional
    - Pragmatics: linked to the intent, statements are expected to be pertinent
- Solving ambiguities:
    - Ambiguity in written representation (speech synthesis)
    - Lexical ambiguity, on word grammatical properties (Part-of-speech tagging) or sense (Word sense disambiguation)
    - Syntactical ambiguity (parsing)
    - Ambiguity in interpreting a statement (sentiment classification, natural language inference)
- NLP applications: speech recognition, machine translation, grammar/spelling correction, ranking (assign larger probability to best option), optical character recognition, information retrieval, summarization, question answering, text generation
- Language model: estimates the probabilities of text sequences
    - Chain rule: decompose in smaller parts: $\mathbb{P}(w_1,\dots,w_m)=\prod_{i=1}^{m}\mathbb{P}(w_i|w_{<i})$
- Sequence of n successive items: n-gram
    - Ordered sequence looking $n-1$ words into the past
    - Depends on tokenization
    - Unigram (1-gram), bigram (2-gram), trigram (3-gram)…
    - Markov assumption: probability depends upon a fixed number of words (the order): order k $\mathbb{P}(w_i|w_1,\dots,w_{i-1})\approx\mathbb{P}(w_i|w_{i-k},\dots,w_{i-1})$
        - Order 0 (unigram), order 1 (bigram), order 2 (trigram)…
        - Trigram is arguably the most used: $\mathbb{P}(w_i,\dots,w_m)=\approx\mathbb{P}(w_1)\mathbb{P}(w_2|w_1)\prod_{i=3}^{m}\mathbb{P}(w_i|w_{i-2},w_{i-1})$
    - 4-gram are generally not sustainable due to the long-term dependencies, which are insufficient to the language model
    - n-gram model is parametric, with the number of parameters $\sim O(|\mathcal{V}|^n)$: defined by the values of $\theta=\{\mathbb{P}(w_n|w_1,\dots,w_{n-1}),\forall(w_1,\dots,w_n)\in\mathcal{V}^n\}$
    - $\mathbb{P}(w_n|w_{1:n-1})=\frac{C(w_1,\dots,w_n)}{\sum_{w'\in\mathcal{V}}C(w_1,\dots,w_{n-1},w')}$
- Model evaluation
    - Extrinsic evaluation: apply the model and use related metric
    - Intrinsic evaluation: $\text{Perplexity}(w_1,\dots,w_m)=\sqrt[m]{\prod_{i=1}^{m}\frac{1}{\mathbb{P}_\theta(w_i|w_{i-n+1},\dots,w_{i-1})}}$ (branching factor)
        - Perplexity is a simple function of the cross-entropy: $\text{Perplexity}(w_1,\dots,w_m)=2^{-\frac{1}{m}\sum_{i=1}^{m}\log\mathbb{P}_\theta(w_i|w_i-n+1,\dots,w_{i-1})}$
        - Maximizes likelihood, hence it minimizes perplexity
- Frequency is not the best measure of association between words
    - It is skewed: Zipf's law: frequency $\propto\frac{1}{\text{rank}}$
    - Very frequent words are rarely the most useful for classification
    - Zero probability n-grams: we use smoothing to solve it
        - Laplace Smoothing (add one): $\mathbb{P}(w_n|w_{1:n-1})=\frac{C(w_1,\dots,w_n)+1}{\sum_{w'\in\mathcal{V}}C(w_1,\dots,w_{n-1},w')+|\mathcal{V}|}$ (avoids making never-seen sequences impossible)
        - Backoff: when not enough information at order n back off to smaller orders
        - Interpolation: interpolate between smaller orders
- Pointwise Mutual information (PMI): evaluate how unexpected is the co-occurrence: $\text{PMI}(w_i,w_j)=\log\frac{\mathbb{P}(w_i,w_j)}{\mathbb{P}(w_i)\mathbb{P}(w_j)}=\log\frac{M_{w_i,w_j}\sum_{k=1}^{n}\sum_{l=1}^{n}M_{k,l}}{\sum_{l=1}^{n}M_{w_1,l}\sum_{k=1}^{n}M_{k,w_2}}$
    - Negative values are unreliable

Hidden Markov Models

- Stochastic process
    - Random state variable at time t: q_t or $q(t)$
    - Values of $q(t)$ belong to the finite set $S=1,\dots,Q$
    - Probability for observing state i at time t: $P(q_t=1)$
    - Evolution process: from initial state q_1, chain of state transitions $(t\le T)$
    - State sequence probability: $P(q_1,\dots,q_T)=P(q_1)P(q_2|q_1)P(q_3|q_1,q_2)\dots P(q_T|q_1,\dots,q_{T-1})$
    - Model: transition probabilities + initial state probability
- Markov chain (discrete time)
    - Markov property (order k): limit dependencies
        - $P(q_t|q_1,\dots,q_{t-1})=P(q_t|q_{t-k},\dots,q_{t-1})$
        - Bigram: $k=1$: $\mathbb{P}(q_t|q_1,\dots,q_{t-1})=\mathbb{P}(q_t|q_{t-1})$, $\mathbb{P}(q_t,\dots,q_T)=\mathbb{P}(q_1)\prod_{t=2}^{T}\mathbb{P}(q_t|q_{t-1})$
- Stationary Markov chain
    - Transitions do not depend on time: $\mathbb{P}(q_t=j|q_{t-1}=i)=\mathbb{P}(q_{t+k}=j|q_{t+k-1}=i)=a_{ij}$
    - Transition probability matrix $A=a_{ij},i=1,\dots,Q,j=1,\dots,Q$
    - Initial probability vector $\Pi=[\pi_i=P(q_1=i)],i=1,\dots,Q$
    - Constraints $0\le\pi_i\le1, 0\le a_{ij}\le1,\sum_{i=1}^{Q}\pi_i=1,\sum_{j=1}^{Q}a_{ij}=1$
- Model topology
    - Ergodic model (without constraints): A is full
    - Lef-right model: A is triangular
- Hidden Markov Model (HMM) for pattern recognition
    - Each class m is represented by an HMM model λ_m
    - Combination of 2 stochastic processes (one observed and one hidden)
    - State sequence hidden $\{q_1,\dots,q_T\}$
    - Observation generated by the states $\{o_1,\dots,o_T\}$
- Discrete HMM
    - Set of Q discrete states $\{1,\dots,Q\}$
    - Set of N observed symbols $\{s_1,\dots,s_N\}$
    - Observation probability matrix (probability of observing each symbol in each state): B
    - Defined by A, Π and B
- Continuous HMM
    - Probability of observing o_t in state i: $b_i(o_t)$
        - Gaussian mixture: $b_i(o_t)=\sum_{k=1}^{M}c_{ik}\,\mathcal{N}(o_t;\mu_{ik},\sigma_{ik})$
        - Gaussian model: $b_i(o_t)=\frac{1}{\sqrt{2\pi}\sigma_i}\exp-\frac{(o_t-\mu_i)^2}{2\sigma_i^2}$
    - Defined by A, Π, probability density function and $b_i(o_t)$
- HMM is a type of Dynamic Bayesian Network (DBN)
    - Independence of set of nodes: $P(o_1,\dots,o_T|q_1,\dots,q_T)=\prod_{t=1}^{T}P(o_t|q_t)$
    - Factorization: $P(o_1,\dots,o_T,q_1,\dots,q_T)=\pi_{q_1}b_{q_1}(o_1)\prod_{t=2}^{T}a_{q_{t-1},q_t}b_{q_t}(o_t)=P(o_1,\dots,o_T|q_1,\dots,q_T)P(q_1,\dots,q_T)$
- Generative HMM
- HMM decoding: assign the pattern to class $\hat{m}=\underset{m}{\operatorname{argmax}}P(o_1,\dots,o_T|\lambda_m)$
- Viterbi decoding to Part of Speech (POS) tagging
    - Let $o=o_1,\dots,o_T$: $P(o|\lambda)=\sum_i P(o,q=i|\lambda)$ so search for $\hat{q}=\underset{q}{\operatorname{argmax}}P(o,q|\lambda)$ then estimate likelihood by $P(o|\lambda)\approx P(o,\hat{q}|\lambda)$
    - Joint probability of best partial state sequence ending at t on state i and corresponding to the partial observation sequence $o_1,\dots,o_t$: $\delta_t(i)=\max_{q_1,\dots,q_{t-1}}P(q_1,\dots,q_t=i,o_1,\dots,o_t|\lambda)$
        - Recurrence: $\delta_{t+1}(j)=b_j(o_{t+1})\max_i a_{ij}\delta_t(i)$
    - Algorithm:
        - 1st column: Initialization: $\delta_1(i)=b_i(o_1)\pi_i\quad i=1,\dots,Q$
        - Columns 2 to T:
            - Recursion $\delta_{t+1}(j)=b_j(o_{t+1})\max_i a_{ij}\delta_t(i)\quad t=1,\dots,T-1,\quad j=1,\dots,Q$
            - $\phi_{t+1}(j)=\underset{i}{\operatorname{argmax}}a_{ij}\delta_t(i)$ (Save best path)
        - Termination
            - $P(o,\hat{q}_T)=\max_j\delta_T(j)$
            - $\hat{q}_T=\underset{j}{\operatorname{argmax}}\delta_T(j)$
        - Backtrack: $\hat{q}_t=\phi_{t+1}(\hat{q}_{t+1})\quad t=T-1,T-2,\dots,1$
- Baum-Welch
    - $P(o,q_t=i|\lambda)=\beta_t(i)\alpha_t(i)$
        - Backward variable: $\beta_t(i)=P(o|q_t=i,\lambda)$
            - $\beta_T(i)=1$
            - $\beta_t(i)=\sum_{j=1}^{Q}a_{ij}\,b_j(o_{t+1})\beta_{t+1}(j)$
            - $P(o|\lambda)=\sum_{j=1}^{Q}\beta_1(j)\pi_j b_j(o_1)$
        - Forward variable: $\alpha_t(i)=P(o_1,\dots,o_t,q_t=i|\lambda)$
            - $\alpha_1(j)=b_j(o_1)\pi_j$
            - $\alpha_{t+1}(j)=b_j(o_{t+1})\sum_{i=1}^{Q}\alpha_t(i)a_{ij}$
            - $P(o|\lambda)=\sum_{j=1}^{Q}\alpha_T(j)$
        - $\gamma_t(i)=P(q_t=i|o)=\frac{\beta_t(i)\alpha_t(i)}{\sum_{j=1}^{Q}}\beta_t(j)\alpha_t(j)$
            - $\hat{q}_t=\underset{j}{\operatorname{argmax}}\gamma_t(j)$
        - $\xi_t(i,j)=P(q_t=i,q_{t+1}=j|o,\lambda)=\frac{\beta_{t+1}(j)b_j(o_{t+1})a_{ij}\alpha_t(i)}{\sum_{k=1}^{Q}\alpha_t(k)\beta_t(k)}$
- Training: complete data
    - Given L observation sequences and associated state sequences
    - $\hat{a}_{ij}=\frac{\sum_{l=1}^{L}\sum_{t=1}^{T(l)-1}1_{\{q_t^{(l)}=i,q_{t+1}^{(l)}=j\}}}{\sum_{l=1}^{L}\sum_{t=1}^{T(l)-1}1_{\{q_t^{(l)}=i\}}}$
    - $\hat{b}_i(s_k)=\frac{\sum_{l=1}^{L}\sum_{t=1}^{T(l)}1_{\{o_t^{(l)}=s_k,q_t^{(l)}=i\}}}{\sum_{l=1}^{L}\sum_{t=1}^{T(l)}1_{\{q_t^{(l)}=i\}}}$
    - $\hat{\mu}_i=\frac{\sum_{l=1}^{L}\sum_{t=1}^{T(l)}o_t^{(l)}1_{\{q_t^{(l)}=i\}}}{\sum_{l=1}^{L}\sum_{t=1}^{T(l)}1_{\{q_t^{(l)}=i\}}}$
    - $\hat{\sigma}_i^2=\frac{\sum_{l=1}^{L}\sum_{t=1}^{T(l)}(o_t^{(l)}-\hat{\mu}_i)^2 1_{\{q_t^{(l)}=i\}}}{\sum_{l=1}^{L}\sum_{t=1}^{T(l)}1_{\{q_t^{(l)}=i\}}}$
- Training: incomplete data
    - Given L observation sequences
    - Viterbi

    - Baum-Welch
        - $\hat{\pi}_i = \frac{\sum_{l=1}^{L} \gamma_1^{(l)}(i)}{L}$
        - $\hat{a}_{ij} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T(l)-1} \xi_t^{(l)}(i,j)}{\sum_{l=1}^{L} \sum_{t=1}^{T(l)-1} \gamma_t^{(l)}(i)}$
        - $\hat{b}_i(s_k) = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T(l)} \gamma_t^{(l)}(i) 1_{\{o_t^{(l)} = s_k\}}}{\sum_{l=1}^{L} \sum_{t=1}^{T(l)} \gamma_t^{(l)}(i)}$
        - $\hat{\mu}_i = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T} o_t \gamma_t(i)}{\sum_{l=1}^{L} \sum_{t=1}^{T} \gamma_t(i)}$
        - $\hat{\sigma}_i^2 = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T} (o_t - \hat{\mu}_i)^2 \gamma_t(i)}{\sum_{l=1}^{L} \sum_{t=1} \gamma_t(i)}$

Structured prediction in natural language processing

- Structured prediction refers to the task of predicting structured objects rather than independent labels such as scalar values or categorical quantities
    - Not structured prediction: sentiment analysis, text classification,chat bots, autocorrection, speech recognition
    - Structured prediction: tagging, parsing, coreference resolution, text summarization
    - Depends: machine translation, natural language understanding, natural language inference
- Part of Speech (POS) tagging
    - English: ~ 15% of words are ambiguous
    - Example: Noun (N), Proper noun (PN), Transitive Verb (TV), Adjective (Adj), Determiner (D), Noun Phrase (NP: D + N), Verb Phrase (VP: TV + NP), Sentence (S: NP + VP)
    - Syntactic tree
    - Computational linguistics tasks: linguistic change, linguistic variation, comparison and control
    - NLP tasks: syntactic parsing, machine translation, sentiment analysis, text-to-speech
    - Algorithms: HMM, Conditional Random Fields (CRF), Maximum Entropy Markov Models (MEMM), Neural sequence models (RNNs or Transformers), Finetuned LLMs (ex: BERT)
        - Required a human-labeled training set
- Named Entity Recognition (NER)
    - Named Entity (people (PEO), location (LOC), organization (ORG), geo-political entity (GPE), dates, times, prices (AMOUNT))
    - Applications: sentiment analysis, question answering, information extraction
    - Algorithms for NER: Beginning-inside-outside (BIO) tagging, HMM, Conditional Random Fields (CRF), Maximum Entropy Markov Models (MEMM), Neural sequence models (RNNs or Transformers), Fine-tuned LLMs (ex: BERT)
        - Required a human-labeled training set
- Coreference resolution (CR): find all linguistic expressions in a given text that refer to the same entity
- Sequence labeler: model that assigns a label to each unit in a sequence, mapping a sequence of observations to a sequence of labels of the same length
- Goal: add information about the context in which a word appears
    - Conditional Random Fields (CRF) and Maximum Entropy Markov Models (MEMM) allow integration of rich features for better accuracy than HMM, however they require much slower training
- Dependency parsing: the syntactic structure of a sentence is described in terms of directed binary grammatical relations between the words
    - The arcs go from heads to dependents; the parse is a typed dependency structure
        - Dependency structure shows which words depend on (modify, attach to, or are arguments of) which other words
            - A dependency structure can be represented as a directed graph $G = (V, A)$:
                - a set of vertices V (~ set of words in a given sentence + punctuation)
                - a set of ordered pairs of vertices A (arcs)
            - Can have constraints: must be connected, must have root node, must be acyclic or planar
    - Computational linguistics tasks: Human communication, Linguistic variation
    - NLP tasks: Syntactic parsing for semantic parsing Machine translation, Sentiment analysis, Text-to-speech
    - Phrase attachment ambiguities
    - Three components: a stack, on which the parse is built; a buffer, containing the tokens to be parsed; a parser which takes actions on the parse via a predictor: an oracle, which requires supervised training for which it is necessary data
    - Algorithm:
        - The parser: walks through the sentence left-to-right, shifting items from the buffer onto the stack
        - At each time point: the top two elements on the stack are examined, the oracle makes a decision about what transition to apply to build the parse:
            - LEFTARC: assign the current word as the head of some previously seen word
            - RIGHTARC: assign some previously seen word as the head of the current word;
            - SHIFT: postpone dealing with the current word, storing it for later processing.
        - LEFTARC cannot be applied when ROOT is the second element of the stack.
        - LEFTARC and RIGHTARC require two elements to be on the stack to be applied.
- Dependency tree: directed graph that satisfies the following constraints:
    - Constraints
        - There is a single designated root node that has no incoming arcs
        - Appart from the root node, each vertex has exactly one incoming arc
        - There is a unique path from the root node to each vertex in V
    - ROOT: Root of the tree, head of the entire structure
    - NSUBJ: Nominal subject
    - OBJ: Direct object
    - NMOD: Nominal modifier
    - DET: Determiner
    - CASE: Prepositions, postpositions, other case markers
    - Universal Dependencies project
- Dependency treebanks: human annotated tree datasets
    - Bilexical affinities, Dependency distance, Intervening material,Valency of heads
- Evaluate parsing:
    - Exact match (EM): how many sentences are parsed correctly
    - Labeled attachment score (LAS): is a word properly assigned to its head with the correct dependency relation?
    - Unlabeled attachment score (UAS): is a word properly assigned to its head? (ignoring the dependency relation)
    - Label accuracy score (LS): what is the percentage of tokens with correct labels? (ignoring where the relations come from)

Neural Language Models and Word Embeddings

- Difficulties with counting: models are huge, vectors are sparse
    - Use dense, distributed representations
        - Change the context for counting words: use surrounding words: co-occurrence matrix
        - Word meaning (distribution in the neighborhood): word embeddings: vector describing the distribution of other words in the neighborhood (cosine distance can be used to compute word similarity but it does not work well: all dimensions matter have the same weight)
    - Reduce the vocabulary (lemmatisation, TF-IDF)
    - Re-weight vectors
- Latent Semantic Analysis: Singular Value Decomposition (SVD): $M = U\lambda V^T$
    - λ diagonal matrix, with eigenvalues - ordered
    - U, V orthogonal; eigenvectors
    - Keep the k first columns of U, for the k largest eigenvalues, to obtain embeddings $\in \mathbb{R}^k$
    - Very costly
    - New space is interpreted as topics
- Topic modeling: mostly generative models
    - Probabilistic LSA: generation of words follows a mixture of conditionally independent multinomial distributions, given topics: $P(w|d) = \sum_t P(t)P(d|t)P(w|t) = P(d) \sum_t P(t|d)P(w|t)$
        - $P(d|t)$ relates to V, $P(w|t)$ relates to U
    - Latent Dirichlet Allocation (LDA): topic distribution is assumed to have a Dirichlet prior
- n-gram neural models
    - Teach a neural network to predict probability $\mathbb{P}(w_i|w_{i-n+1}, \dots, w_{i-1})$
    - Divided in two parts: processing the context words, obtaining output probability for the next word
- Neural Probabilistic Language Model (NLPM)
    - Continuous word vectors: Each input and output word is represented by a vector of dimension $d \ll |\mathcal{V}|$ taking values in $\mathbb{R}$, rather than being discrete
    - Continuous probability function: The probability of the next word is expressed as a continuous function of the features of the word in the current context - using a neural network
    - Joint learning: The parameters of the word representations, and the probability function are learnt jointly
    - Projecting words: input one-hot encoded words to a densely connected to a smaller layer to smaller layer of dimension d_w: the weight matrix are word embeddings
    - Obtaining scores:
        - Context c is the concatenation of the smaller representation of $n-1$ words
        - Create hidden representation $h = \phi(W^{ih}c)$
        - Obtain scores for all words in $\mathcal{V}$ given h: $s = W^{hs}h$ (output word embeddings to upscale the hidden representation to the predicted word dimension)
            - Compute probabilities: softmax $\mathbb{P}(o|w_{i-n+1,\dots,w_{i-1}}) = \frac{\exp(h_i^T \mathbf{w}_o^{hs})}{\sum_{l=1}^{|\mathcal{V}|} \exp(h_i^T \mathbf{w}_l^{hs})}$
    - Training: minimize negative log-likelihood: $\mathrm{NLL}(\theta) = -\sum_{i=1}^{m} \log(\mathbb{P}_\theta^{(i)}(w_i|w_{<i}))$
        - Equivalent to minimizing the Kullback-Leibler divergence
    - Updates: $\frac{\partial}{\partial\theta}\log(\mathbb{P}_\theta(w_{i+j}|w_{<i})) = \frac{\partial}{\partial\theta}s_{w_{i+j}} - \sum_{k=1}^{|\mathcal{V}|} \mathbb{P}_\theta(w_k|w_j)\frac{\partial}{\partial\theta}s_w$
        - First term increases the conditional log-likelihood of w_{i+j} given w_i
        - Second term decreases the conditional log-likelihood of all $w_k \in \mathcal{V}$
        - Learning bottleneck due to softmax: Making hierarchical predictions: replace complexity in $O(|\mathcal{V}|)$ by $O(\log|\mathcal{V}|)$; Sampling based methods: sum over k samples
- Learning word embeddings
    - Continuous bag of words (CBOW): simplifying architecture: $h = C\sum_{j,j\neq 0} w_{i+j}$, $\mathbb{P}(w_i|w_j \in \mathcal{C}_w) = o_w = \frac{\exp(h^T \mathbf{w}_{w_i}^{hs})}{\sum_{l=1}^{|\mathcal{V}|} \exp(h^T \mathbf{w}_l^{hs})}$
        - Parameters: $\theta = \{W, C\}$
        - Training: $d(\mathbb{P}_*^w, P_\theta) = -\log\mathbb{P}_\theta(w) = -\log o_w$
        - Objective function: $J_{MLE}^{CBOW}(\theta) = -\sum_{i=1}^{N} \log P_\theta(w_i|w_{i-m}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+m})$
    - Skip-gram
        - Objective function: $J_{MLE}^{SG}(\theta) = -\sum_{i=1}^{N} \sum_{j\neq 0}^{-m<j<m} \log \mathbb{P}_\theta(w_{i+j}|w_i)$
        - Better handling of infrequent words
    - Avoid computing $\sum_{k=1}^{|\mathcal{V}|}$
        - Replace task by binary classification: predicting the right word $\mathbb{P}_\theta(w_{i+j}|w_i) = \sigma(s_{w_{i+j}})$
        - Only take into consideration the positive contribution: $\frac{\partial}{\partial\theta}\log(\mathbb{P}_\theta(w_{i+j}|w_i)) = (1 - \sigma(s_{w_{i+j}}))\frac{\partial}{\partial\theta}s_{w_{i+j}}$
        - Negative contribution by sampling $k \ll |\mathcal{V}|$ wrong words: $J_{MLE}^{SG}(\theta) = -\sum_{i=1}^{N} \sum_{j\neq 0}^{-m<j<m} \left(\log\sigma(s_{w_{i+j}}) + \sum_{l=1}^{k} \log\sigma(s_{w_l}^{\text{noise}})\right)$
            - Requires subsampling of frequent words in the noise distribution
    - GloVe: learn word embeddings by predicting word co-occurrence counts
        - $J_{\text{GloVe}}(\theta) = \sum_{w_i, w_j \in \mathcal{V}} f(M_{w_i,w_j})(w_{w_j}^T w_{w_j} - \log M_{w_i,w_j})^2$
            - $f(x) = \begin{cases} (\frac{x}{x_{\max}})^\alpha & x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$
        - Very fast training
        - Linear word representation relationships can capture meaning
    - Often reflect bias (gender bias)
    - Lexical ambiguity: polysemous, homonyms imply word embeddings with conflation
        - Solutions: sense embeddings, sparse coding, contextualized embeddings
- Prediction-based: fast and scale well with available data; dense and capture complex patterns; but require a lot of data and do not use all statistical information available
- Count-based: can also give dense representation (SVD to a PMI matrix); relatively fast; uses efficiently all information available and works well with little data
- Sub-word models: decompose to solve closure of vocabulary
    - Phonems, morphemes