Deep Learning and Natural Language Processing - Vector Semantics

Vincenzo Scotti,
Ph.D. student
vincenzo.scotti@polimi.it

arcslab
adaptable, relational and cognitive software environments

N.L.P. – A.Y. 20-21

Contents

This symbol means that the formulae in the slide aren't required to be known

VECTOR SEMANTICS

Vector semantics

- Build a new model of meaning focusing on **similarity**
  - Each element is represented (defined) by a d-dimensional **vector**
  - Similar elements are close in space
- This kind of representation is called **embedding**
  - Position in the embedding space represents meaning
- This concept is valid for words as well as for many other concepts

WORD EMBEDDINGS

Word embeddings overview

- Traditionally words have been represented using *sparse* encodings
  - Bag-of-Words (BoW)
  - Term Frequency–Inverse Document Frequency (TF-IDF)
- Such representations are uninformative, sparse and inefficient
  - Similar concepts are orthogonal in this representation (sparsity)
  - Words require very high-dimensional one-hot representations (uninformative and inefficient)
    - In BoW a word $\mathbf{o}_{BoW}$ is a one-hot vector of the size of the vocabulary $\mathscr{V}$ ($\|\mathscr{V}\| > 10^6$)
    - One-hot encoding requires that the values in all dimension but one are set to 0, the remaining to 1, hence $\mathbf{o}_{BoW} \in 1^{\|\mathscr{V}\|} \mid \sum_{i=1}^{\|\mathscr{V}\|} o_{BoW,(i)} = 1$ (where $1 \equiv \{0,1\}$

Word embeddings overview

- **Word embeddings** map words from their original high-dimensional sparse space to low-dimensional **dense** (**compact**) space
- Characteristics of word embeddings
  - **Compression** (**dimensionality reduction**): size d is way smaller than that of $\mathscr{V}$ (usually $50 \leq d \leq 1000$)
    - Easier to use these dimensions as *features* in machine learning
  - **Smoothing**: from discrete orthogonal representations to continuous
  - **Densification**: from sparse to dense representations

Word embeddings overview

- In embedding space similarity between words can now now measured as the distance between their vector representation
  - Many distance functions, most commonly used is cosine distance

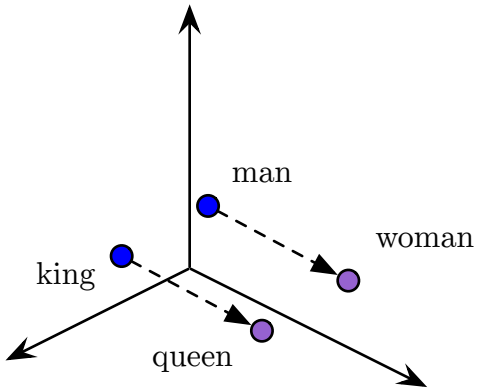
$$d_{cosine}(\mathbf{v}_1, \mathbf{v}_2) = \frac{\mathbf{v}_1^\top \cdot \mathbf{v}_2}{|\mathbf{v}_1| \cdot |\mathbf{v}_2|}$$

$$d_{cosine}(\mathbf{v}_1, \mathbf{v}_2) = \frac{\sum_{i=1}^{d} v_{1,(i)} v_{2,(i)}}{\sqrt{\sum_{i=1}^{d} v_{1,(i)}^2} \sqrt{\sum_{i=1}^{d} v_{2,(i)}^2}}$$

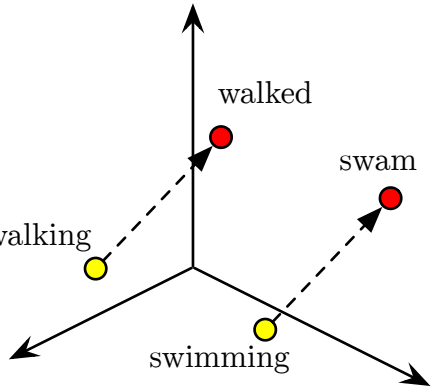
Word embeddings overview

- First works using dense representations come from the 80's (Deerwester, 1988)
- In early 2000 neural language models already showed to produce useful embeddings (Bengio et al., 2003)
- Few years later embeddings were shown to be crucial for a large number of NLP tasks (Collobert and Weston, 2007 and 2008; Weston, 2011)

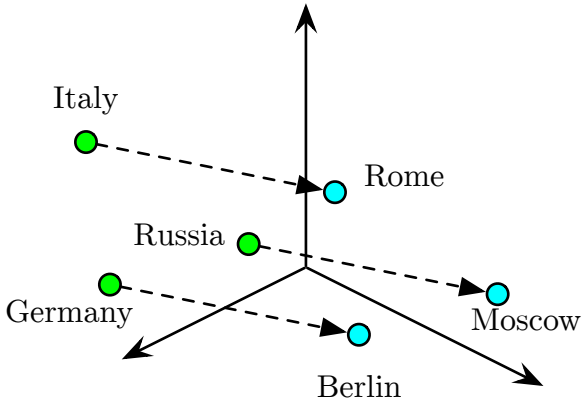
Word embeddings overview



Male-Female

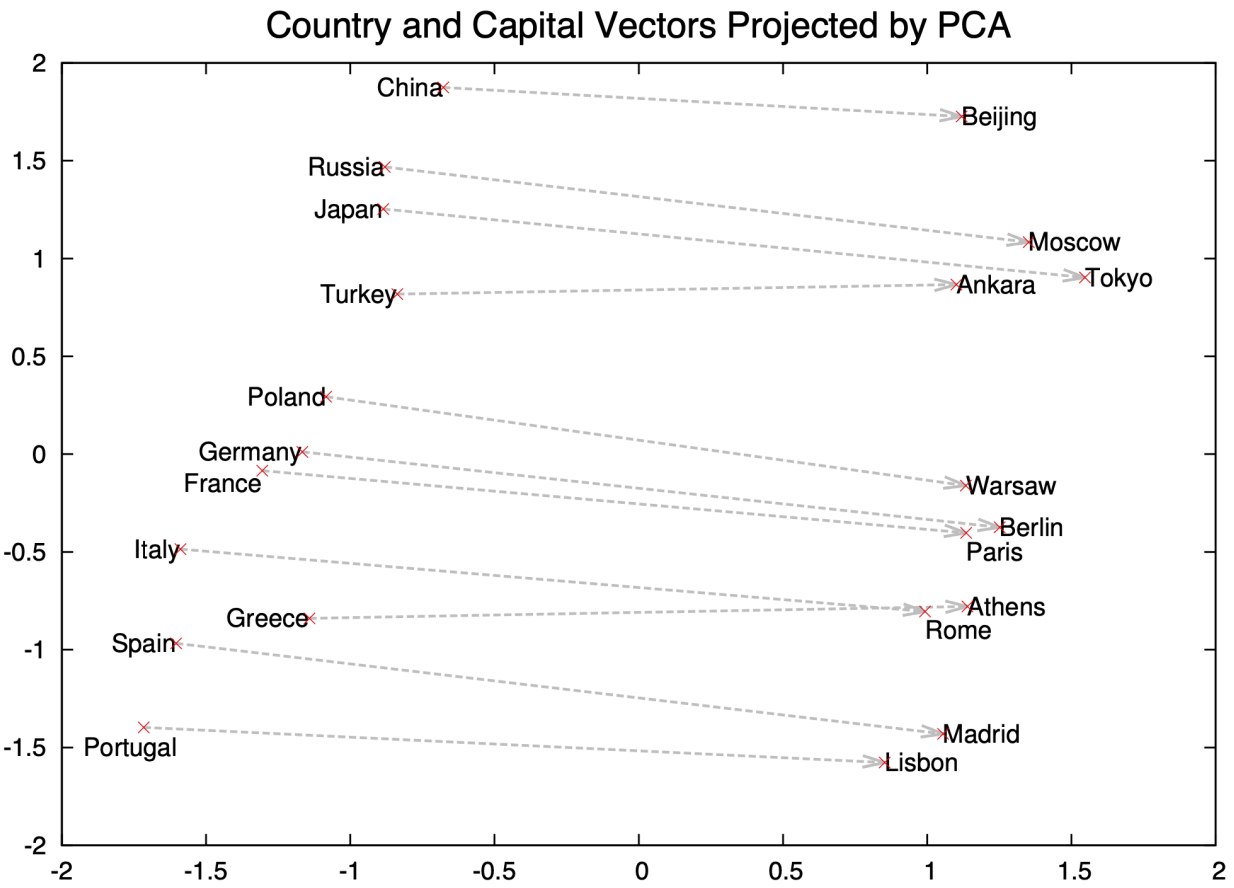


Verb tense

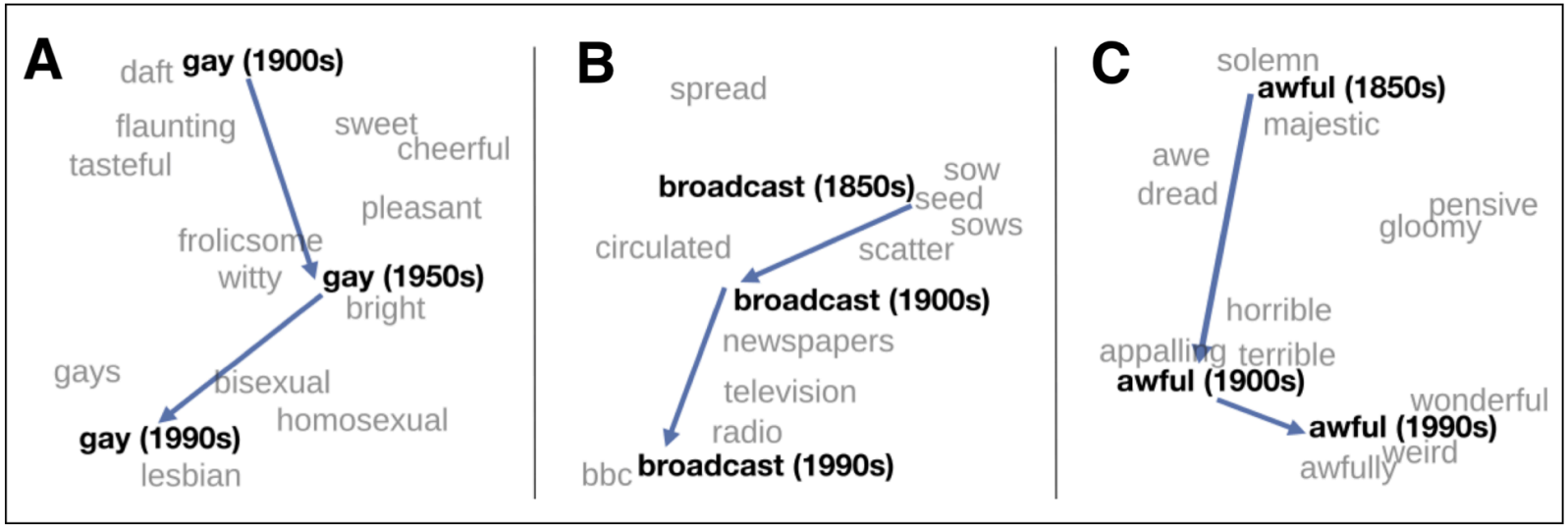


Country-Capital

Word embeddings overview



Word embeddings overview



Taxonomy

- Word embeddings models are organized according to two orthogonal dimensions:
  - **Prediction**-based vs **count**-based
    - Prediction: embeddings obtained through the training of models for next/missing word prediction given a context
    - Count: embeddings obtained leveraging words co-occurrence counts in a corpus
  - **Shallow** vs **deep** models
    - Shallow models: embeddings are extracted from a word embedding matrix $\mathbf{W} \in \mathbb{R}^{\|\mathscr{V}\| \times d}$ through lookup over the matrix rows
      - Each row of the embedding matrix corresponds to a word
      - In practice $\mathbf{w}_{\text{embedding}} = \mathbf{W}^{\top} \cdot \mathbf{o}_{\text{BoW}}$
    - Deep models: embeddings are extracted from a deeper model
      - In practice $\mathbf{w}_{\text{embedding}} = f\left(\mathbf{o}_{\text{BoW}}\right)$

Word2Vec

- First notable work using a wide corpus (Mikolov et al., 2013a,b)
  - Trained a shallow prediction model on a very large amount of data
- Prediction-based shallow model
- Two approaches were proposed to train Word2Vec embedding models
  - **Continuous Bag-of-Word**
  - **Skip-gram**
- Additionally it proposes a positive/negative sampling training (Mikolov et al., 2013b)

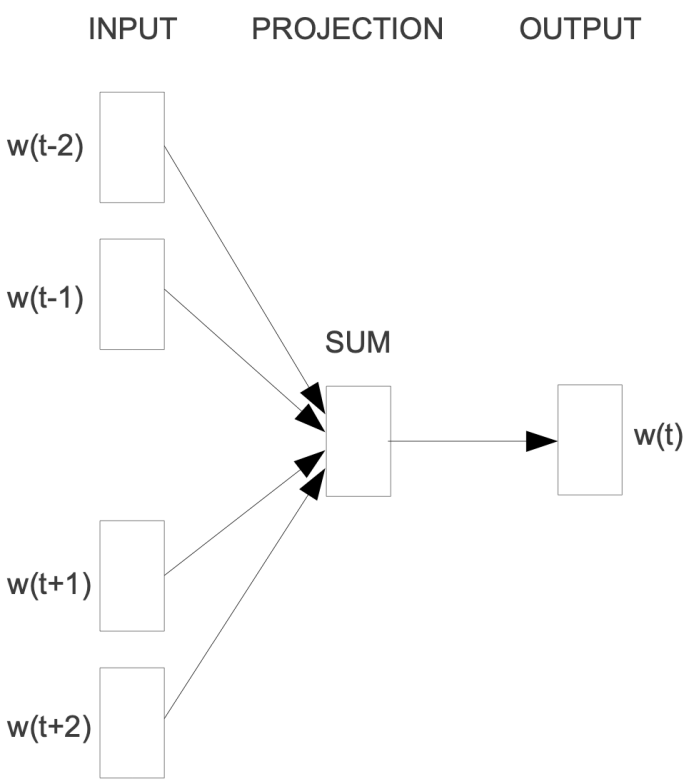
- "You shall know a word by the company it keeps" (Firth, 1957)

Word2Vec (CBoW)

- Train the model by learning to predict the missing word in a context
  - Context are the surrounding words in a sentence
- Model input: one-hot encoding of the n-gram words (except the central one) representing the context
- Model output: one-hot encoding of the word missing from the n-gram context

Word2Vec (CBoW)

- CBoW training visualized



Word2Vec (CBoW)

- Given a sequence of words (a sentence) expressed through one-hot encoding $\mathbf{S} = [\mathbf{o}_1, \ldots, \mathbf{o}_t] \in \mathbb{R}^{t \times \|\mathscr{V}\|}$
- Given a context of size c
- The model is trained to predict the missing word $\mathbf{o}_i$
  - The prediction is computed as a $\mathrm{softmax}()$ over the vocabulary

$$\mathbf{o}_j = \mathrm{softmax}\left(\mathbf{W}'^{\top} \cdot \left(\frac{1}{c}\mathbf{W}^{\top} \cdot \sum_{i \in \left[-\frac{c}{2}, \frac{c}{2}\right] \setminus \{0\}} \mathbf{o}_{j+i}\right)\right)$$

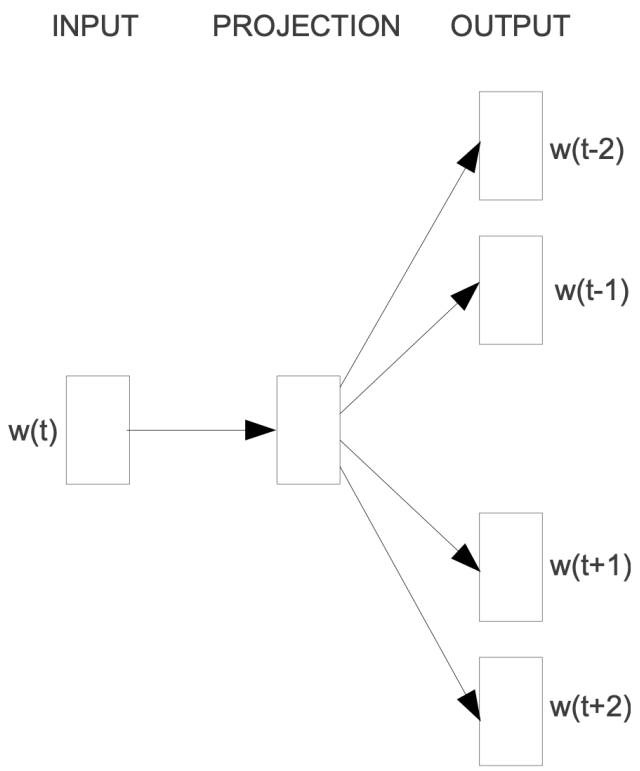
  - $\mathbf{W}$ is the embedding matrix to be trained
- The model is predicting the missing word from the context
  - Can be trained to minimize the cross-entropy between the prediction and the actual output

Word2Vec (SkipGram)

- Train the model by learning to predict the context words given the central word
  - Context are the surrounding words in a sentence
- Model input: one-hot encoding of the word missing from the n-gram context
- Model output: one-hot encoding of the n-gram words (except the central one) representing the context

Word2Vec (SkipGram)

- SkipGram training visualized



Skip-gram

Word2Vec (SkipGram)

- Given a sequence of words (a sentence) expressed through one-hot encoding $\mathbf{S} = [\mathbf{o}_1, \ldots, \mathbf{o}_t] \in \mathbb{R}^{t \times \|\mathscr{V}\|}$
- Given a context of size c
- The model is trained to predict the missing context
  - The prediction is computed as a $\mathbf{softmax()}$ over the vocabulary
    $$\forall i \in \left[-\frac{c}{2}, \frac{c}{2}\right] \backslash \{0\} \longrightarrow \mathbf{o}_{j+i} = \text{softmax}\left(\mathbf{W}'^{\mathsf{T}}_i \cdot \left(\mathbf{W}^{\mathsf{T}} \cdot \mathbf{o}_j\right)\right)$$
  - $\mathbf{W}$ is the embedding matrix to be trained
- The model is predicting separately the i-th context word from the central one
  - Can be trained to minimize the sum of cross-entropies between the prediction and the actual output context
  - Each different position in the context is reconstructed with a different matrix of weights $\mathbf{W}'_i$

Positive/negative sampling

- Positive/negative sampling is a technique to speed up training
- Conceptually:
  - Reward the model for correct predictions and punish for wrong ones
- Practically:
  - Loss is given by the sum of the cross-entropy between target output and prediction and the negative cross-entropy between a wrong target and the prediction

$$E(\vartheta) = \underbrace{-\mathbf{o}_{\text{target}} \log(\mathbf{o}_j)}_{\text{positive sample}} - \left(\underbrace{-\mathbf{o}_{\text{wrong}} \log(\mathbf{o}_j)}_{\text{negative sample}} \right)$$

- Compatible with both CBoW and Skip-Gram

GloVe

- Work on ratios of probabilities from the word-word co-occurrence matrix (Pennington et al., 2014)
- Count-based shallow model
- Explicitly encodes meaning as vector offsets in the embedding space
  - Word2Vec does so implicitly
  - Meaning is represented as the ratio of co-occurrence probability
- Still trained with CBoW or Skip-Gram approach

FastText

- FastText was designed to extend Word2Vec and help deal with missing words (Bojanowski et al., 2017)
  - Introduction of sub-word embeddings to take advantage of language morphology
- A word is now represented as itself plus its constituent character n-grams
- Trained with Skip-Gram approach

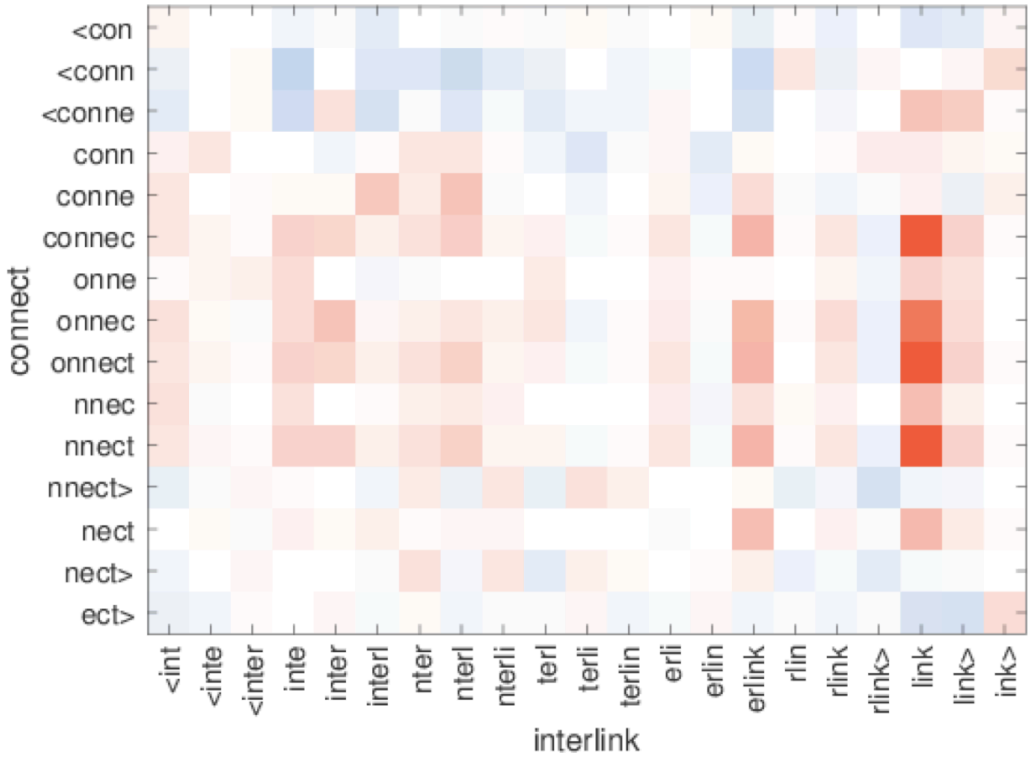
FastText

- Generating word embedding by summing whole word embedding plus the character n-grams composing it
  - Usually works with 3-gram
  - Characters representing a word are enclosed by special brackets
- The input representation is now a one-hot vector encoding the word, plus a binary vector over the possible 3-gram sequences
  - Note that if we consider only the alphabetical characters, the number of trigrams is $26^3 = 17576$, while the vocabulary is bigger of one order of magnitude usually
- The embedding becomes a concatenation of the one-hot word vector and the BoW representation of the sub-word 3-gram
- E.g. "word" is treated as $\langle$word$\rangle$

$\langle$ w o
w o r
o r d
r d $\rangle$

FastText

- Similarity of out-of-vocabulary words



Latent semantic analysis

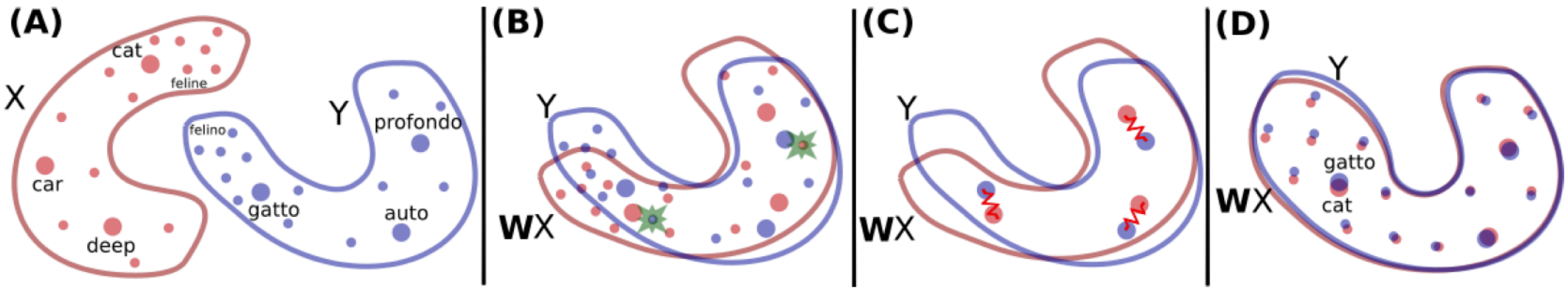
- In general training a word embedding model can be seen a factorization problem
  - Decompose the **co-occurrence matrix** $\mathbf{D} \in \mathbb{R}^{\|\mathscr{V}\| \times \|\mathscr{V}\|}$ produced by the context window sliding over the text in the corpus
  - $\mathbf{D}$ is such that the i-th column corresponds to the average context of the i-th word
    - This context can be extracted as $\mathbf{D} \cdot \mathbf{o}_{BoW}$
- Singolar Value Decomposition (SVD) can be used for this purpose
  - $\mathbf{D} = \mathbf{W}'\mathbf{\Sigma}\mathbf{W}^\top$ where $\mathbf{W}'$ and $\mathbf{W}^\top$ are two orthonormal matrices and $\mathbf{\Sigma}$ is a diagonal matrix
    - $\mathbf{W} \in \mathbb{R}^{\|\mathscr{V}\| \times \|\mathscr{V}\|}$, $\mathbf{\Sigma} \in \mathbb{R}^{\|\mathscr{V}\| \times \|\mathscr{V}\|}$ and $\mathbf{W}^\top \in \mathbb{R}^{\|\mathscr{V}\| \times \|\mathscr{V}\|}$
- $\mathbf{W}^\top$ is the matrix that goes from the input word space to and hidden space, $\mathbf{W}'$ goes from the hidden space to the output context space
- Rows in $\mathbf{W}^\top$ and columns in $\mathbf{W}'$ are presented in "importance" order with respect to the contribution to the reconstruction of $\mathbf{D}$
  - Keep only the first d rows of $\mathbf{W}^\top$ and use it as word embedding matrix

Multilingual unsupervised and supervised embeddings

- Multilingual unsupervised and supervised embeddings (MUSE) are a technique to map pre trained embeddings in different languages so that they work in the same space
  - The idea is to have word with same meaning map to same (or at least close) points in space
- Use adversarial learning to learn a rotation matrix that make the word embedding spaces overlap

Multilingual unsupervised and supervised embeddings

- MUSE embeddings construction



CONTEXTUAL EMBEDDINGS

Seq2Seq models

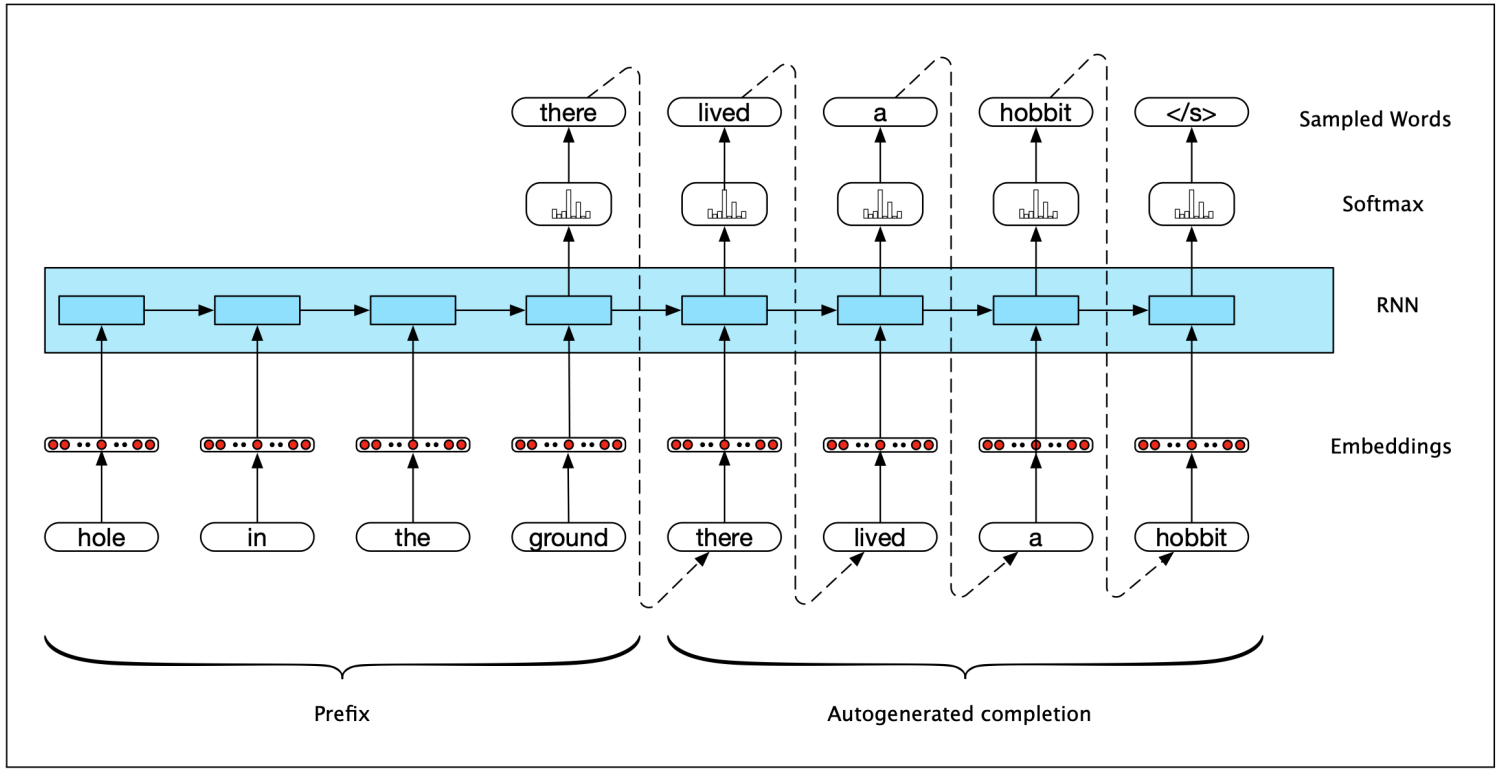
- Sequence-to-Sequence (**Seq2Seq**) identifies a class of neural network models widely used in NLP that work by taking an input sequence and *generating* another sequence as output
  - Sometimes these models are referred to as **Encoder-Decoder**
- Encoder network takes an input sequence and generate a contextualized version of it.
- Decoder networks use this representation as context to generate an output sequence according to the task it needs to accomplish
  - Usually decoding is autoregressive
  - Explicit use of ⟨start⟩ and ⟨end⟩ tokens to control decoding

Seq2Seq models

- Many variants of this architecture
  - **Vanilla Seq2Seq** use a single network as encoder and decoder
    - E.g. an LSTM layer
  - **Encoder-decoder Seq2Seq** use two separate networks to encode input and decode output
    - E.g. two LSTMs where the second (the decoder) takes the last hidden state of the encoder as initial hidden state (often the encoder is a BiLSTM)
  - **Hierarchical encoder-decoder Seq2Seq** use an intermediate network between encoder and decoder to generate a context (compressed representation of the input sequence) to help the decoding the output
    - Like a sentence embedding model that predicts the embedding of the next sentence

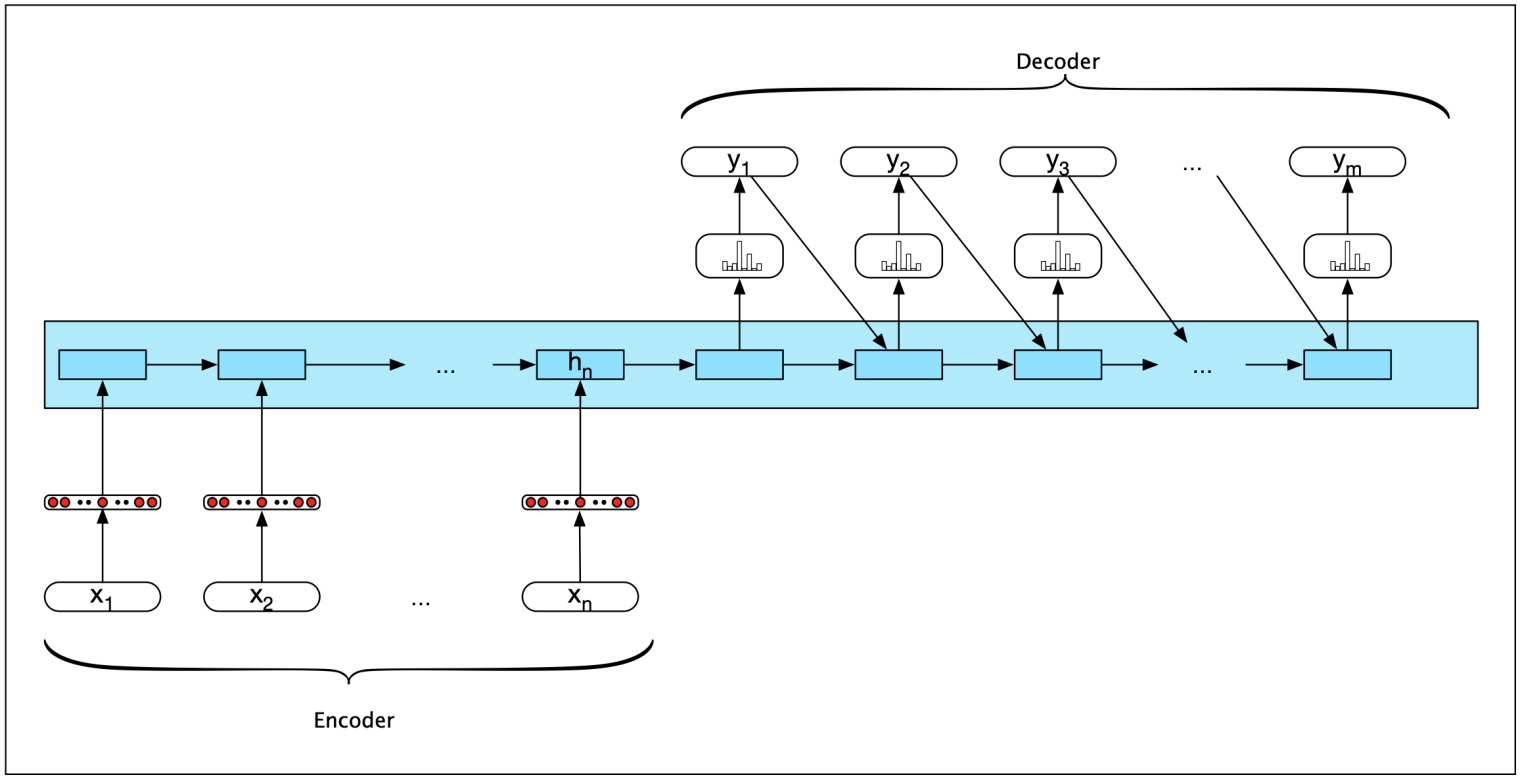
Seq2Seq models

- Vanilla



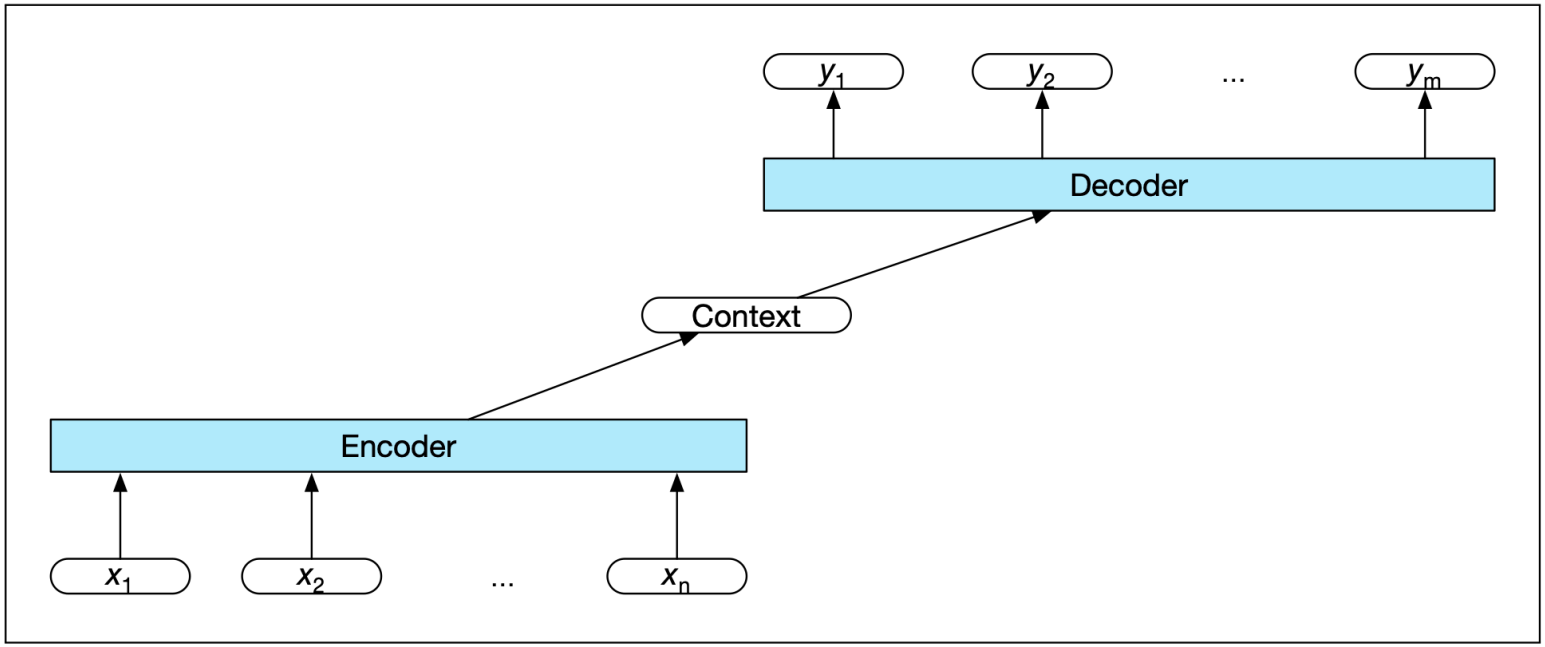
Seq2Seq models

- Encoder-decoder



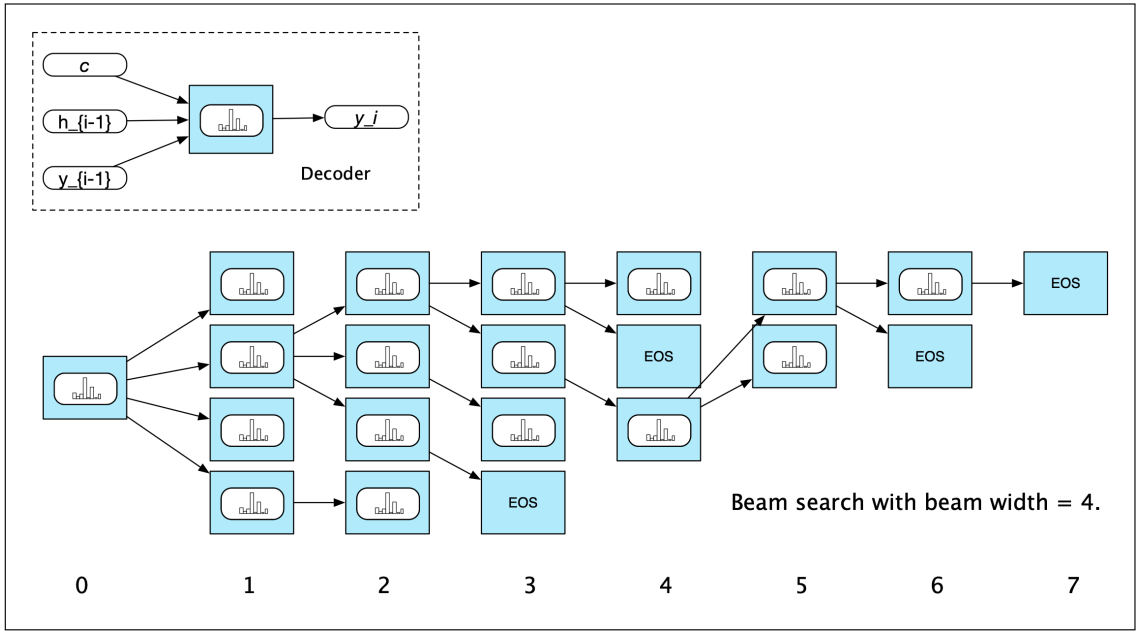
Seq2Seq models

- Hierarchical encoder-decoder



Seq2Seq models

- Decoding output probabilities
  - **Greedy decoding**: at each step select only most probable token
  - **Beam search**: at each step keep the b best decoded sequences



Seq2Seq models

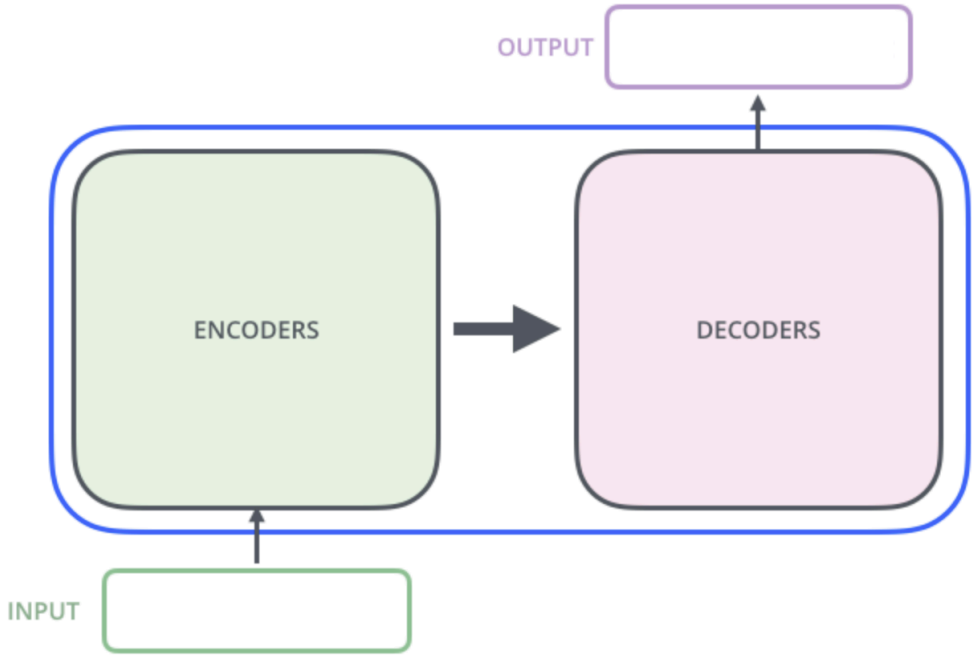
- For long time these models have been implemented using recurrent networks
  - Even LSTMs fail in front of very long contexts
  - Hierarchical variant helped cope with the problem, but still persisted
- Introducing an attention layer from encoder hidden representation to decoder last hidden vector helped cope with the problem
  - The decoder now selects at each step which information retain from the encoder

Transformer Seq2Seq models

- Attention layers can substitute completely recurrent layers
  - Completely parallel computation
    - Requires positional information to be encoded with the input
  - Longer memory limit
- Attention can be used to compute context as well as encoder and decoder functions
  - **Self-attention** on input to build the encoder
  - **Attention** between encoder and decoder (encoder output serves as key and values, decoder current last token as query) to compute context
  - **Masked self-attention** on output to build the decoder
    - Zero out future words to avoid decoder see them (remember that attention goes in all directions)
- This is called **Transformer** architecture

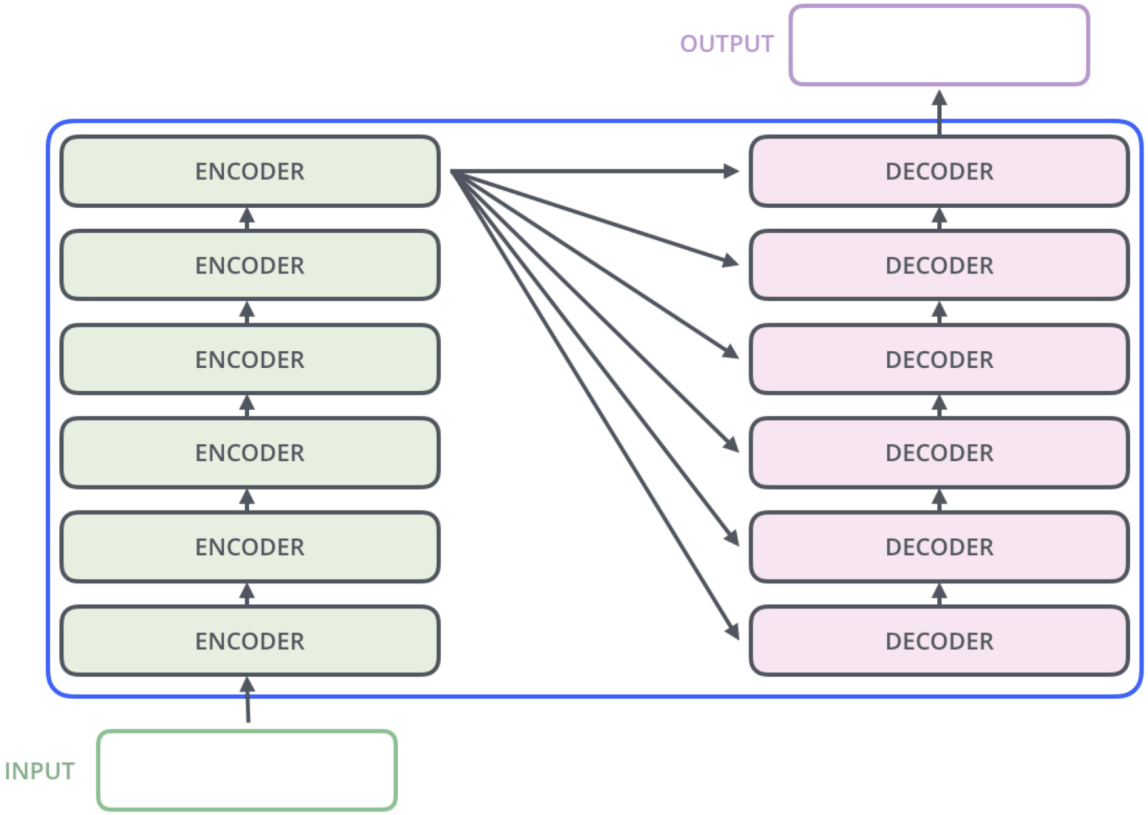
Transformer Seq2Seq models

- Transformer architecture



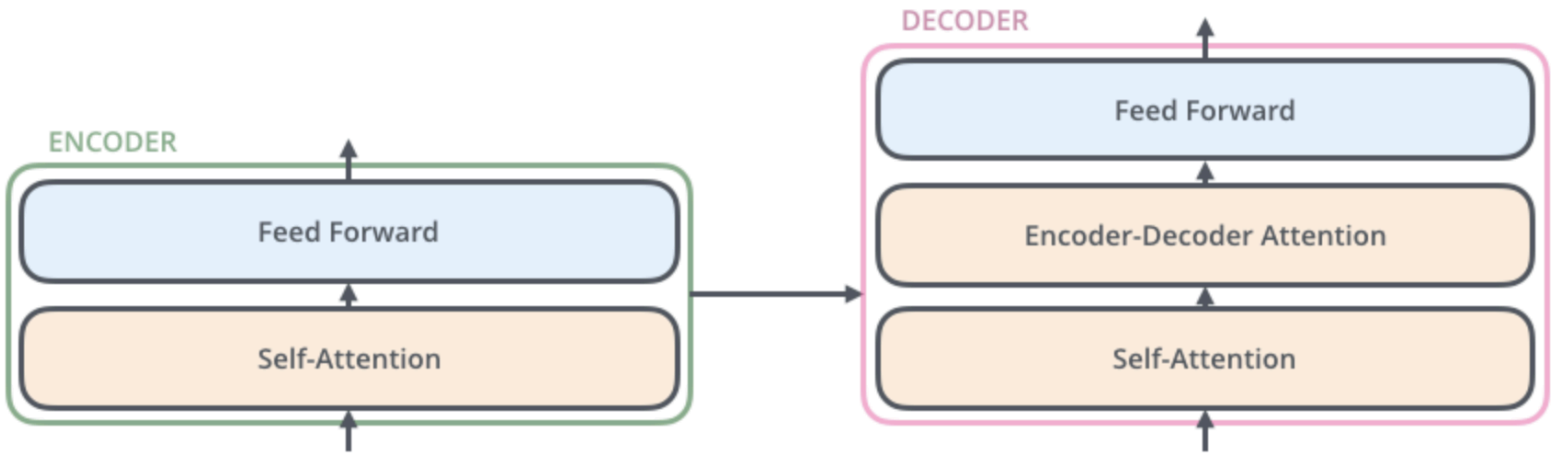
Transformer Seq2Seq models

- Transformer architecture



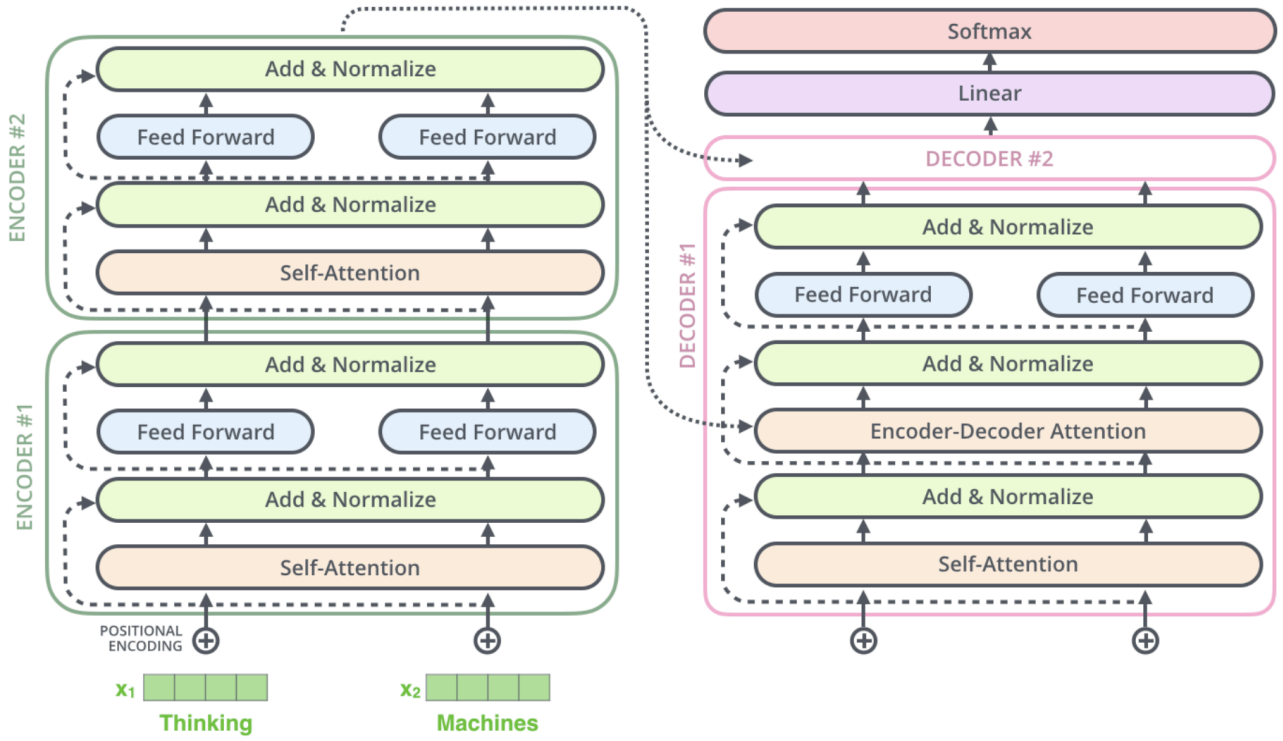
Transformer Seq2Seq models

- Transformer architecture



Transformer Seq2Seq models

- Transformer architecture



Transformer Seq2Seq models (byte pair encoding)

- Usually transformers are fed with the Byte Pair Encoding (BPE) of the input string rather then the one-hot encoding
  - This approach comes from lossless compression
- Mine a corpus by recursively finding the most common consecutive characters (byte pair) and replace them with a new symbol
- E.g.
  Input string: “I like studying NLP”
  Intermediate tokenization: [“I”, “like”, “studying”, “NLP”]
  BPE: [“I”, “like”, “study”, “ing”, “NLP”]

  Each of the elements is treated as a separate symbol

Contextual embeddings overview

- Build vector representation of words using the entire sentence (or document) where it appears
- The embedding of the same word changes at runtime depending on the context
- Encoder-decoder Seq2Seq architectures can be used to learn these embeddings
  - Actually the entire architecture in not required

Learning contextual embeddings

- In general all these models for contextual embeddings are trained over a language modeling task
  - **Unsupervised pre-training** over a generic task like this helps produce very general features
  - Easy access to wide data sets
    - No need of labelling
- They use either the entire architecture or a single part
  - Usually encoder learns good features and decoder a good generative function
    - However these tasks are interchangeable

Learning contextual embeddings

- Three pre-training tasks:
  - **Prefix language modeling**
    - Predict rest of the word sequence starting from the first part (The prefix)
    - Used to train encoder-decoder models
  - **Causal language modeling**
    - Predict next word(s) using the previously predicted one as input at each step
    - Used to train decoder-only models
  - **Masked language modeling**
    - Predict missing word(s) leveraging information from the rest of the input text
    - Used to train encoder-only models
- After pre-training remove the final prediction layer to have the embeddings

Taxonomy

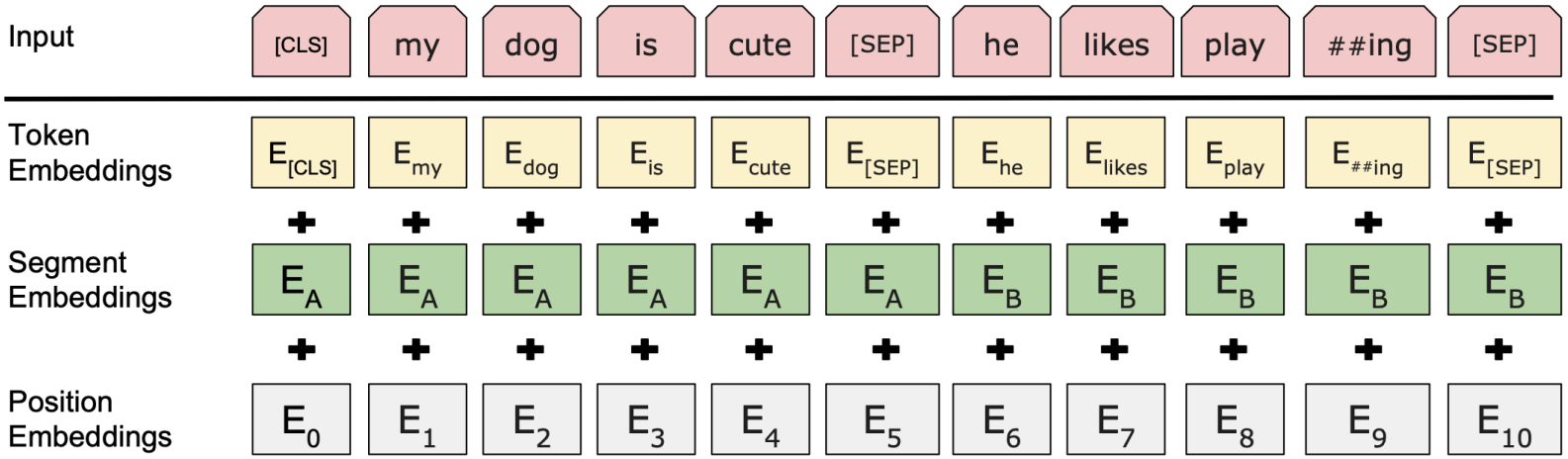
- Different architecture and pre-training objective induce the taxonomy of these contextual embedding models
  - Encoder-only
    - Like ELMo, BERT, ...
  - Decoder-only
    - Like GPT, GPT-2, ...
  - Encoder-decoder
    - Like T5

BERT

- The main transformer encoder architecture for contextual embeddings is Bidirectional Encoder Representations from Transformers (BERT)
- It is a stack of self attention layers trained on masked language modeling task
  - In practice it is an auto-encoder
  - Takes the same text as input and output
  - Some words in the input text are obscured (masked)
- Through this unsupervised task the model automatically learns good hidden features for contextual embeddings

BERT

- Input representation

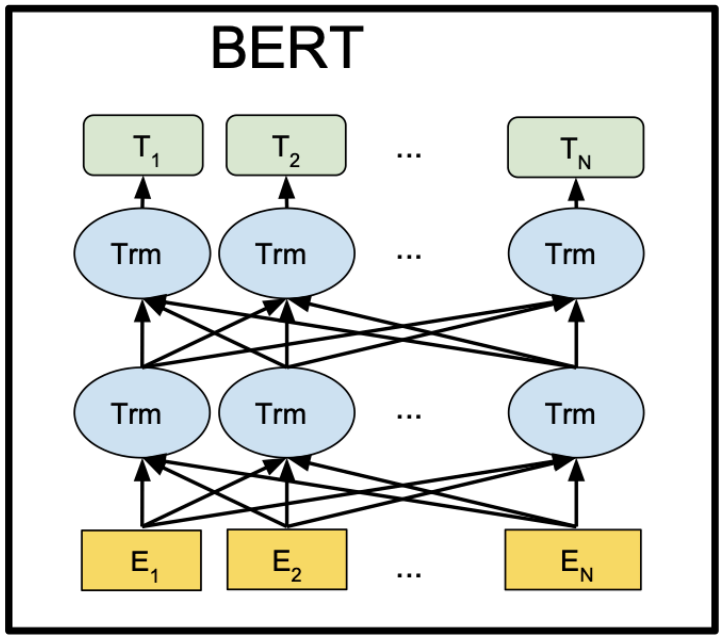


BERT (architecture)

- Input is a sequence of one-hot encoded symbols, these symbols are passed through and embedding layer and the positional encoding is added to the embeddings
- Hidden layers are a stack of attention and fully connected layers
- Output layer is a fully connected with a $\mathbf{softmax()}$ activation over the symbols dictionary

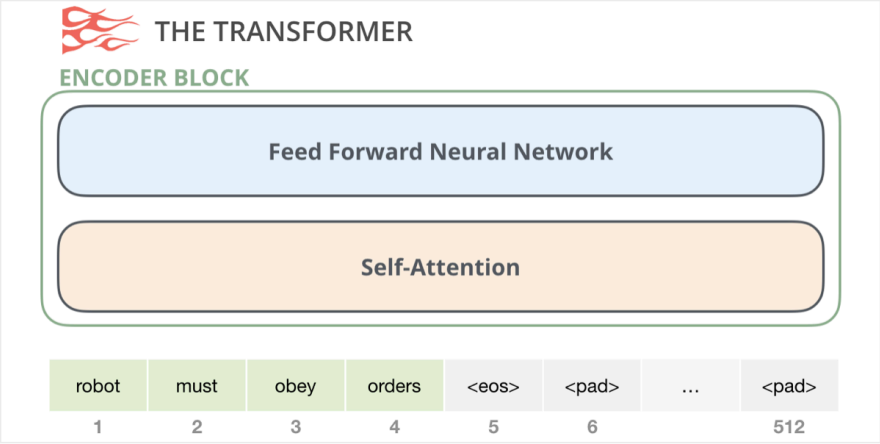
BERT (architecture)

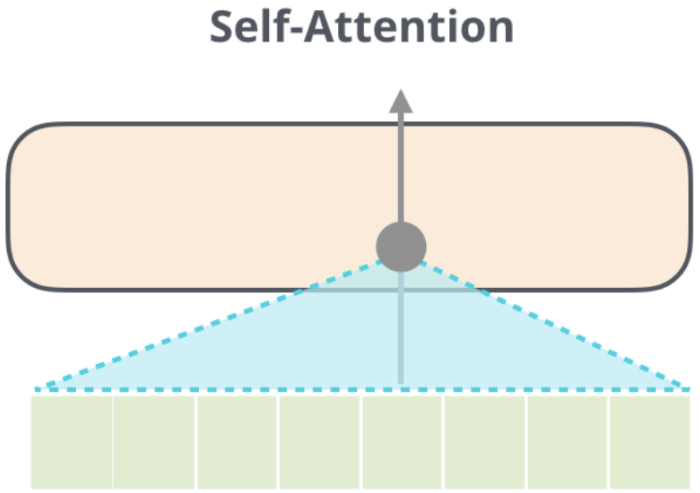
- High level architecture



BERT (architecture)

- Transformer encoder block and self-attention





BERT

- To extract the contextual embeddings the last layer is removed
- At inference time, to compute the embeddings only the hidden features maps are considered
  - Input text isn't masked anymore

GPT

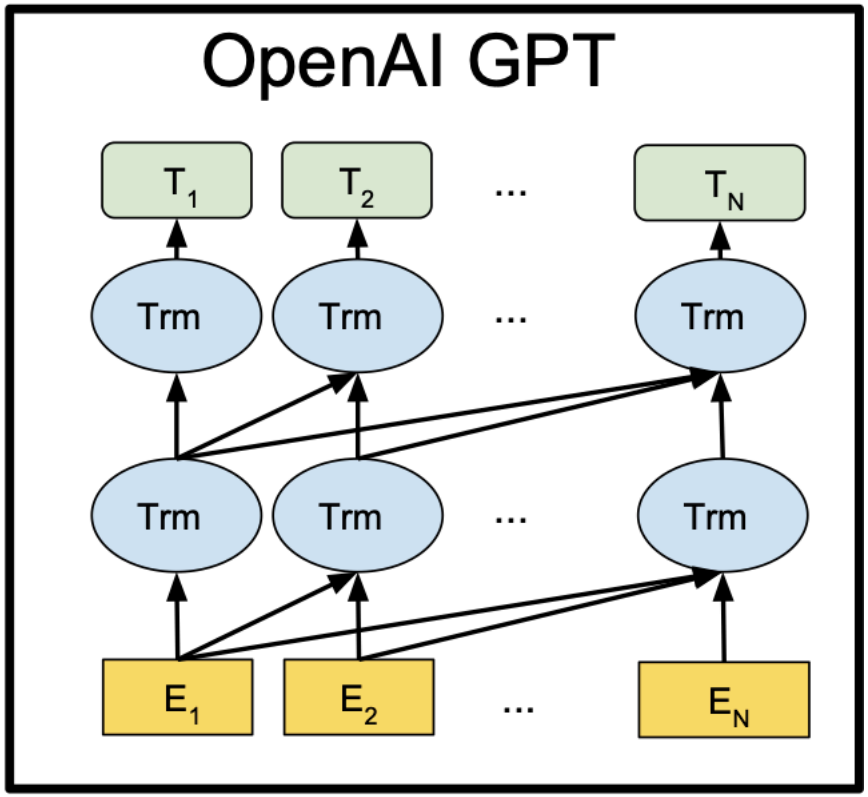
- The main transformer decoder architecture for contextual embeddings is Generative Pre-Training (GPT)
  - We're at the third version of this architecture
- It is a stack of masked self-attention layers trained on causal language modeling task
  - In practice it is a decoder
  - Takes a text as input and the same text shifted on the left as output
  - Must predict next token
- Through this unsupervised task the model automatically learns good hidden features for contextual embeddings

GPT (architecture)

- Input is a sequence of one-hot encoded symbols, these symbols are passed through and embedding layer and the positional encoding is added to the embeddings
- Hidden layers are a stack of masked attention and fully connected layers
  - Masked attention prevents separately each feature vector in the input to attend those on its right
  - In this way language modeling is causal (only tokens on the left are used to predict the next one on the right)
- Output layer is a fully connected with a $\mathrm{softmax}()$ activation over the symbols dictionary

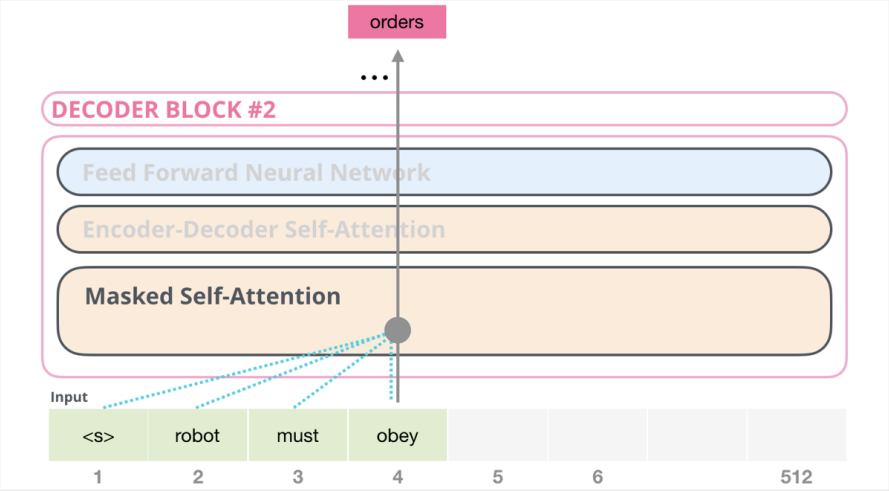
GPT (architecture)

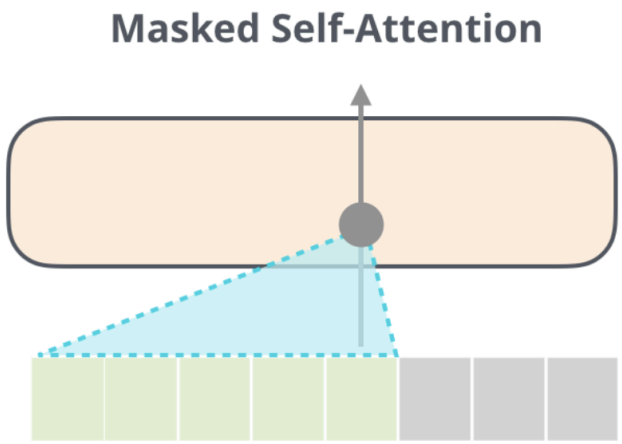
- High level architecture



GPT (architecture)

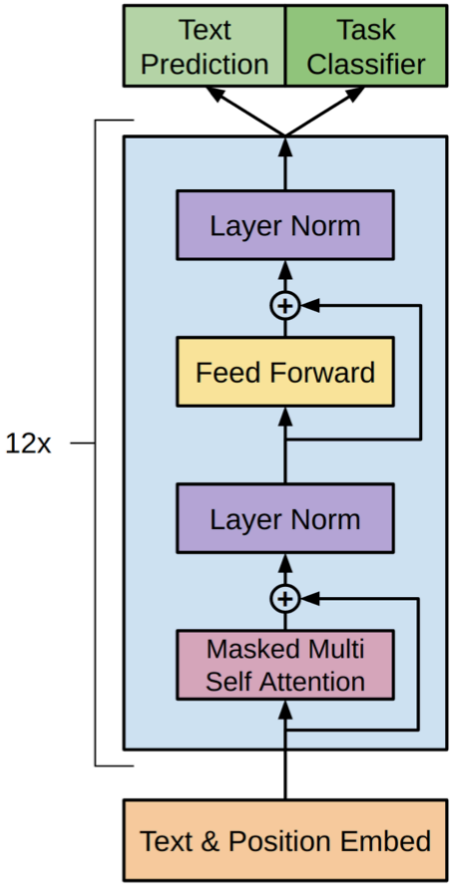
- Transformer decoder-only block and masked self attention





GPT (architecture)

- Attention block detail



GPT

- To extract the contextual embeddings the last layer is removed
- At inference time, to compute the embeddings only the hidden features maps are considered
  - Attention isn't masked anymore

Other transformer architectures for contextual embeddings

- Same concepts, very little variations
- Encoder: BERT, RoBERTa, DistilBERT, XLNet
- Decoder: GPT (1, 2 and 3), MegatronLM, Turing-NLG
- Encoder Decoder: T5

SENTENCE EMBEDDINGS

Sentence embeddings overview

- The inherent hierarchical structure of the human language makes it hard to understand a text from single words
  - Thus, the birth of higher level semantic representations for sentences, the **sentence embeddings**
- Provide a fixed-size vector representation of the entire sentence that preserves syntactic and semantic information

Taxonomy

- Sentence embedding models are organized into two categories
- **Parametric** vs **non-parametric**
  - Parametric models are explicitly trained to generate the embedding starting from words
  - Non-parametric models leverage pre-defined techniques to combine lower level representations without the need of specific training

Sent2Vec

- Extension of Word2Vec to sentences
- Learn word and sentence representation in an unsupervised manner
  - This is done by learning a word representation compatible with aggregation through average in a sentence representation
  - Average the word embeddings in the entire sentence to compute that of the sentence
- Can be trained in an unsupervised manner using CBoW approach
  - Instead of a fixed size context the entire sentence is used
  - The objective during training is still to reconstruct the missing word given the rest of the sentence

Sent2Vec

- Given a sentence $\mathcal{S}$ it employs all the unigrams and bigrams inside it $\mathscr{R}(\mathcal{S})$ to build the representation
- At train time both the embedding matrix $\mathbf{V}$ and the "inverse" embedding matrix $\mathbf{U}$
  - $\mathbf{V} \in \mathbb{R}^{\|\mathscr{V}\| \times d}$ and $\mathbf{U} \in \mathbb{R}^{\|\mathscr{V}\| \times d}$
  - As premised the model is trained with a CBoW approach
- At inference time, only $\mathbf{V}$ is used to extract the embeddings to be averaged in the sentence representation

Sent2Vec

- To compute the embedding of the sentence $\mathbf{v}_S$

$$\mathbf{v}_{\mathcal{S}} = \frac{1}{|\mathcal{R}(\mathcal{S})|}\mathbf{V}^{\top} \cdot \mathbf{o}_{\mathcal{S}} = \frac{1}{|\mathcal{R}(\mathcal{S})|} \sum_{w \in \mathcal{R}(\mathcal{S})} \mathbf{V}^{\top} \cdot \mathbf{o}_{w} = \frac{1}{|\mathcal{R}(\mathcal{S})|} \sum_{w \in \mathcal{R}(\mathcal{S})} \mathbf{v}_{w}$$

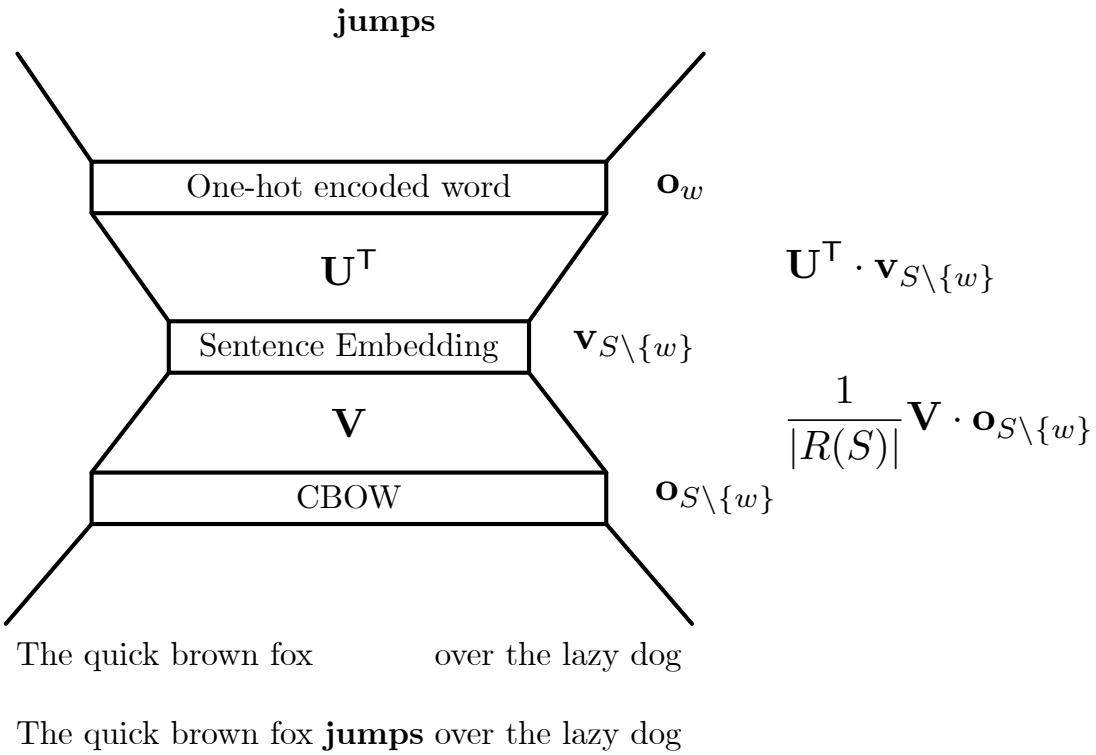
  - Where $\mathbf{o}_{\mathcal{S}}$ is the BoW representation of the unigrams and bigrams in the sentence
  - $\mathbf{o}_{w}$ is the BoW representation of the single unigram or bigram in the sentence

Sent2Vec

- To train the model compute the loss on the prediction of the missing word
  - Minimize reconstruction error of $\mathbf{o}_w$, with $w \in \mathcal{S}$
    - $\mathbf{y} = \mathbf{U} \cdot \mathbf{v}_{\mathcal{S}\setminus\{w\}}$ is the prediction of the model that should match the missing word
    - Notice that $\mathbf{U}$ can be fetched as $\mathbf{V}$ to retrieve the vector corresponding to a given word w
      - No need of $\mathbf{softmax()}$ over entire vocabulary, just match the vector $\mathbf{u}_w$ of $\mathbf{U}$ corresponding to the output to $\mathbf{v}_{\mathcal{S}\setminus\{w\}}$ using the dot product
  - Use positive/negative sampling to speed up training
    - Also called **contrastive loss**

Sent2Vec

- Training approach



Sent2Vec

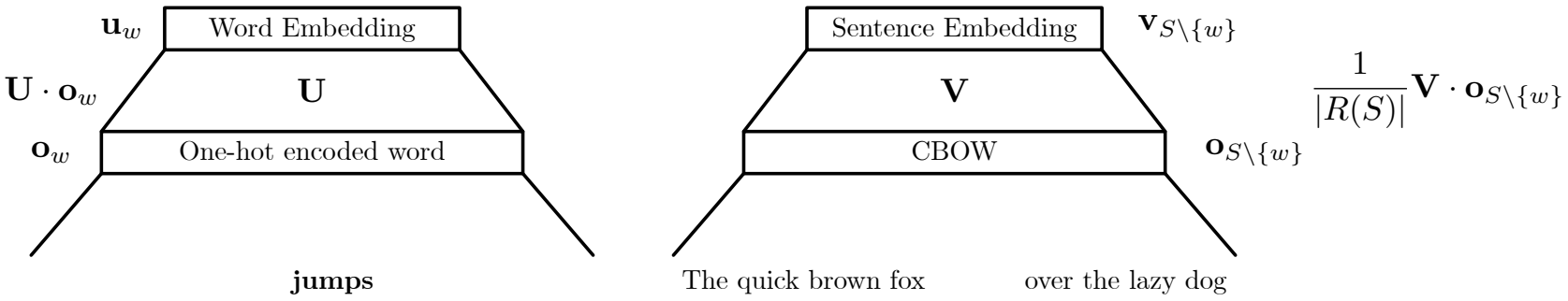
- The unsupervised training objective is given by

$$\min_{\mathbf{V},\mathbf{U}} \sum_{\mathcal{S}\in\mathscr{D}} \sum_{w\in\mathcal{S}} \left(q_p(w)\, l\left(\mathbf{u}_w^\top \mathbf{v}_{\mathcal{S}\setminus\{w\}}\right) + |N_w| \sum_{w'\in V} q_n(w')\, l\left(-\mathbf{u}_{w'}^\top \mathbf{v}_{\mathcal{S}\setminus\{w\}}\right) \right)$$

- Where:
  - $\mathscr{D}$ is the corpus
  - $q_p(w)$and $q_n(w)$ are respectively the positive and negative sampling probability
  - N_w is the set of words sampled negatively for word w
  - $l(x)$is the binary logistic loss function $l(x) = \log(1 + \exp(-x))$

Sent2Vec

- Word embeddings for the target word are fetched directly from the embedding matrix
- Same applies for the other positive/negative samples
- Only a part of the weights is trained at each step



The quick brown fox **jumps** over the lazy dog

Smooth Inverse Frequency

- Smooth Inverse Frequency (**SIF**) proposes a simple algorithm to aggregate word embeddings into a sentence representation
  - The model in this case in **non-parametric**
- Besides the word embeddings, the algorithm takes as input $q(w)$, the frequency of word w in vocabulary $\mathcal{V}$, and a smoothing term α (usually $\alpha = 10^{-3}$)

Smooth Inverse Frequency

- To compute the embedding of sentence $\mathcal{S}$
  - First a weighted mean of the words appearing in the sentence is computed
    $$\mathbf{v}_{\mathcal{S}} = \frac{1}{|\mathcal{S}|} \sum_{w \in \mathcal{S}} \frac{\alpha}{\alpha + q(w)} \mathbf{W}^{\top} \cdot \mathbf{o}_w$$
  - Then the "low frequency" components are removed from the sentences
    - The idea is to remove some sort of **common discourse vector $\mathbf{u}$**
    - $\mathbf{u}$ is extracted as the first eigenvector of a matrix whose columns are the weighed means of the sentences to compare using SVD
    - Finally the sentence embedding is given by
      $$\mathbf{v}_{\mathcal{S}} = \mathbf{v}_{\mathcal{S}} - \mathbf{u}\mathbf{u}^{\top}\mathbf{v}_{\mathcal{S}}$$

REFERENCES

References

- https://web.stanford.edu/~jurafsky/slp3/6.pdf
- http://lsa.colorado.edu/papers/JASIS.lsi.90.pdf
- https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
- https://jmlr.org/papers/volume12/collobert11a/collobert11a.pdf
- https://arxiv.org/pdf/1301.3781.pdf
- https://arxiv.org/pdf/1310.4546.pdf
- https://nlp.stanford.edu/pubs/glove.pdf
- https://arxiv.org/pdf/1607.04606.pdf
- https://arxiv.org/pdf/1710.04087.pdf
- https://web.stanford.edu/~jurafsky/slp3/10.pdf
- http://jalammar.github.io/illustrated-transformer/
- http://jalammar.github.io/illustrated-gpt2/

References

- https://arxiv.org/pdf/1706.03762.pdf
- https://arxiv.org/pdf/1810.04805v2.pdf
- https://arxiv.org/pdf/1907.11692.pdf
- https://arxiv.org/pdf/1910.01108.pdf
- https://arxiv.org/pdf/1906.08237.pdf
- https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
- https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- https://arxiv.org/pdf/2005.14165.pdf
- https://arxiv.org/pdf/1909.08053.pdf
- https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/
- https://arxiv.org/pdf/1910.10683.pdf
- https://arxiv.org/pdf/1703.02507.pdf
- https://openreview.net/pdf?id=SyK00v5xx