NLP

Davide Volpi

May 29, 2025

Contents

1 Text Preprocessing **4**
1.1 Tokenization . . . 4
1.1.1 BPE Tokeniser . . . 4
1.2 Normalization . . . 4
1.2.1 Porter Stemmer . . . 4
1.2.2 Lemmatizer . . . 4

2 Word Embeddings **5**
2.1 Bag of Words . . . 5
2.2 Term-Frequency Based Representation . . . 5
2.3 N-Grams . . . 5
2.4 One Hot Encoding . . . 6
2.5 Embeddings . . . 6
2.6 Distributional Semantics . . . 6
2.7 Cosine Similarity . . . 6
2.8 Term-Document Matrix . . . 6
2.9 Term-Term Matrix . . . 6
2.10 TF-IDF for Term-Document Matrices . . . 7
2.11 PPMI for Term Term Matrices . . . 7
2.12 Latent Semantic Analysis . . . 7
2.13 Word2vec: Skip-Gram Model . . . 7
2.13.1 Matrix Factorization . . . 8
2.14 Embeddings Evaluation . . . 8
2.15 Visualizing Word Embeddings . . . 8
2.16 Semantic Properties of Word Embeddings . . . 8
2.17 Diachronic Word Embeddings . . . 8
2.18 Embedding for Polysemous Words . . . 9
2.19 FastText . . . 9
2.20 Glove . . . 9
2.21 Continuous Bag of Words (CBOW) Model . . . 9

3 Language Models **10**
3.1 Estimating Sentence Probabilities . . . 10
3.2 N-gram Language Models . . . 10
3.2.1 Backoff and Interpolation . . . 10
3.2.2 Out of Vocabulary Words . . . 10
3.2.3 Evaluation of Language Models . . . 11
3.2.4 Output . . . 11

4 Neural Language Models **12**
4.1 Feedforward NLM . . . 12
4.1.1 Inference . . . 12
4.1.2 Training . . . 13
4.1.3 Discussion . . . 13
4.2 Text Generation . . . 14
4.2.1 Top-k Sampling . . . 14

4.2.2 Temperature . . . 14
4.3 Recurrent Neural Networks . . . 14
4.3.1 Inference . . . 14
4.3.2 Training . . . 15
4.3.3 Practical Issues . . . 15
4.4 Bidirectional RNNs . . . 15
4.5 LSTM . . . 15
4.5.1 Gates Equations . . . 16

5 Neural Architectures for NLP **17**
5.1 Encoder-Decoder Models . . . 17
5.1.1 Encoder-Decoder . . . 17
5.1.2 Translation with RNN . . . 17
5.1.3 Translation with Encoder Decoder . . . 17
5.1.4 Training Encoder-Decoder . . . 17
5.2 Attention . . . 18
5.2.1 Attention in Encoder-Decoder . . . 18
5.2.2 Attention Functions . . . 18
5.2.3 Reading Comprehension with Attention . . . 19
5.2.4 Sliding Window for Span-based QA . . . 19
5.3 Neural Approaches . . . 19
5.3.1 Attention Over Attention . . . 19
5.3.2 Multi-Step Reasoning . . . 19

6 Transformers **20**
6.1 Transformer-Encoder . . . 20
6.1.1 Attention . . . 20
6.2 Transformer-Decoder . . . 20
6.3 Pretraining . . . 20

7 Masked Language Models **22**
7.1 BERT (Bidirectional Encoder Representations from Transformers) . . . 22
7.1.1 BERT-NSP . . . 23
7.1.2 RoBERTa . . . 23
7.2 Contextual Embeddings . . . 23
7.2.1 Anisotropy . . . 23
7.3 Sentence Embeddings . . . 23
7.4 Fine-Tuning . . . 23
7.4.1 LORA . . . 24
7.5 Scaling Laws . . . 24

8 Decoder Only Models **25**
8.1 Generative Models . . . 25
8.2 Prompt Engineering . . . 25
8.2.1 Demonstrations . . . 25
8.2.2 Instruction Tuning . . . 25
8.2.3 Chain of Through Prompting . . . 25
8.2.4 Prompt Optimisation . . . 25
8.3 GPT (Generative Pretraining Transformer) . . . 26
8.3.1 GPT-3 . . . 26

9 Sequence Labelling **27**
9.1 Part of Speech . . . 27
9.1.1 Part of Speech Tagging . . . 27
9.1.2 Hidden Markov Models (HMM) . . . 27
9.1.3 Decoding . . . 28
9.1.4 Viterbi Algorithm . . . 28
9.1.5 POS Tagging Classifier . . . 28
9.1.6 Inference . . . 28

9.1.7 POS with Perceptron 28
9.2 Named Entity Recognition 28

10 Parsing **29**
10.1 Graph-Based Dependency Parsing 29
10.1.1 The Arc-Factored Model 29
10.2 Transition-Based Dependency Parsing 29
10.2.1 The Arc-Standard Algorithm 30
10.3 Neural Approaches 30

11 Sentiment Analysis **31**
11.1 Theory of Emotions 31
11.2 Lexicon-based Sentiment Analysis 31
11.2.1 Sentiment Lexicons 31
11.2.2 Learning Lexicons 31
11.2.3 Semantic Axis 31
11.2.4 Label Propagation 32
11.2.5 Using Lexicons 32
11.3 Entity-Based Sentiment Analysis 32
11.4 Sentiment Treebank 32

12 Question Answering **33**
12.1 Search Engines 33
12.1.1 Bi-Encoders 33
12.2 Machine Learning Comprehension 33
12.3 Language Models as Knowledge Bases 33
12.4 Retrieval-Augmented Generation 34
12.4.1 Evaluation 34
12.5 Neural Models 34

13 Sentence Embeddings **35**
13.1 Autoencoder 35
13.2 Skip-Thought Vectors 35
13.2.1 Out of Vocabulary (OOV) 35
13.3 SimCSE (Simple Contrastive Learning of Sentence Embeddings) 35
13.3.1 Unsupervised SimCSE 35
13.3.2 Supervised SimCSE 35
13.4 Sentence-BERT (SBERT) 35

14 Document Embeddings **36**
14.1 Compositional Approach 36
14.2 Paragraph Vectors 36

15 Coreference Resolution **37**
15.1 Evaluation 37
15.2 Algorithms for Coreference Resolution 37
15.2.1 Mention Identification 37
15.2.2 Mention Clustering Based Models 37
15.2.3 Entity Clustering Based Models 38

16 Semantic Role Labelling **39**
16.1 SRL as Classification 39
16.2 SRL Neural Approaches 39

1 Text Preprocessing

1.1 Tokenization

Tokenization: converting the text from a set of symbols to a sequence of word/tokens.
Subword tokenization: if the training corpus contains the words foot and ball but not the word football, the latter is a unknown word. It reduces vocabulary size, and has become the most common tokenization method for LLMs.

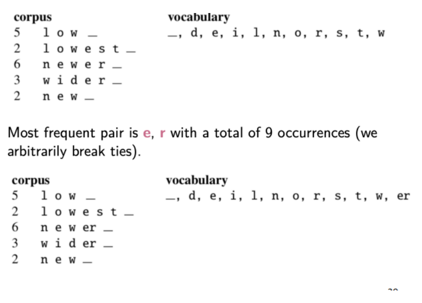
1.1.1 BPE Tokeniser

The BPE token learner is usually run inside words, not merging across word boundaries. To this end, use a specific end-of -word maker. The algorithm iterates through:

1. Begin with a vocabulary composed by all individual characters.
2. Choose the two symbols A,B that are most frequently adjacent.
3. Add a new merged symbol AB to the vocabulary.
4. Replace every adjacent A,B in the corpus with AB.

Stop when the vocabulary reaches size k, a hyperparameter.

- _ is the end of word maker
- After several iterations, BPE learns entire words.
- Most frequent units, useful for tokenizing unknown words.
- **WordPiece (BERT):** uses f(A,B)/f(a)f(b) to select the tokens to merge.



corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

Most frequent pair is e, r with a total of 9 occurrences (we arbitrarily break ties).

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

1.2 Normalization

Text normalization: string transformations that remove distinctions that are irrelevant to downstream applications.
An element of X is also called a **feature**. How discriminate a feature is for a (binary) supervised learning task depends on how imbalanced it is in the (two) classes:

- $|GREAT+| = 10; |GREAT-| = 1$
- $|great+| = 40; |great-| = 42$
- After lowercasing the feature great would be way less discriminative.

Curse op dimensionality: more features potentially require a larger number of examples to estimate correctly the density of the data.
Normalization reduces the size of the feature space, which can help in generalization. Risk of merging away linguistically meaningful distinctions.

1.2.1 Porter Stemmer

Stemming refers to the process of slicing a word with the intention of removing affixes. Stemming is problematic in the linguistic prospective, since it sometimes produces words that are not in the language, or else words that have a different meaning.

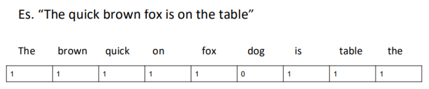
1.2.2 Lemmatizer

A **Lemma** is the canonical form of a word, its dictionary form.
Lemmatization depends on correctly identifying the intended part of speech and the meaning of a word in a sentence.

2 Word Embeddings

2.1 Bag of Words

A typical representation of the data is using a dictionary of words and their frequencies.

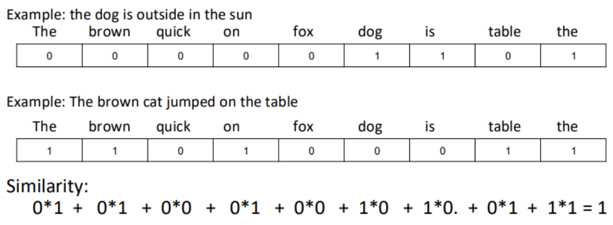


Es. "The quick brown fox is on the table"

The	brown	quick	on	fox	dog	is	table	the
1	1	1	1	1	0	1	1	1

Texts are compared by continuing the number of matchings between words.
The steps for building a BOW representation are:

1. Decide the size of the vector and what token is associate with each position in the vector.
   - Usually this is done by analysing a set of texts to extract the set of unique words.
   - The class CountVectorsizer of Kcikit-Learn does that with the method fit.
2. What function to use when we will fill the vectors (the choice depends on the task).
   - In the class CountVectorsizer of Kcikit-Learn the options when we create the object determine it ($v = CountVectorizer()$).
   - No options mean computing the frequency of a word (0/1).
3. Now, given a text, we can compute its BOW representation.



Example: the dog is outside in the sun

The	brown	quick	on	fox	dog	is	table	the
0	0	0	0	0	1	1	0	1

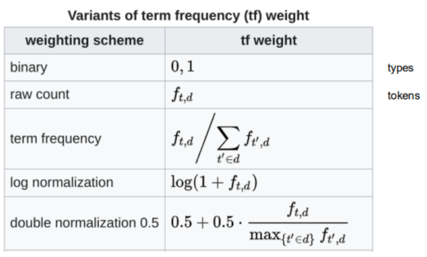
Example: The brown cat jumped on the table

The	brown	quick	on	fox	dog	is	table	the
1	1	0	1	0	0	0	1	1

Similarity:
0*1 + 0*1 + 0*0 + 0*1 + 0*0 + 1*0 + 1*0. + 0*1 + 1*1 = 1

We can compute the similarity between two texts by computing the dot product between the corresponding vectors.

2.2 Term-Frequency Based Representation

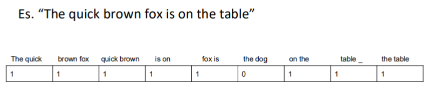


Variants of term frequency (tf) weight

weighting scheme	tf weight	
binary	$0, 1$	types
raw count	$f_{t,d}$	tokens
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$	
log normalization	$\log(1 + f_{t,d})$	
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$	

2.3 N-Grams

An **n-gram** is a sequence of n adjacent words in a text. N-grams are a viable alternative to bag of words.



Es. "The quick brown fox is on the table"

The quick	brown fox	quick brown	is on	fox is	the dog	on the	table _	the table
1	1	1	1	1	0	1	1	1

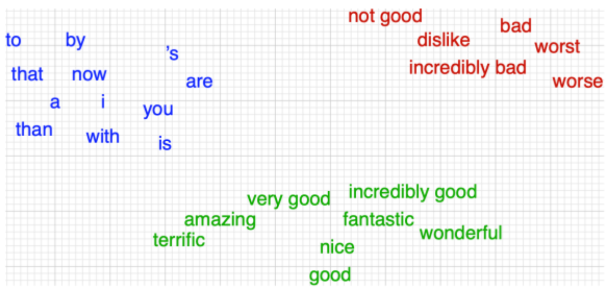
Texts are compared by continuing the number of matching n-grams (dot products of the vectors above).

2.4 One Hot Encoding

In one hot word encoding the representation for each word is orthogonal to each other.
All words are equally dissimilar to each other, but ”dog” should be closer to ”cat” than to ”table”.

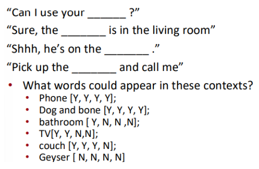
2.5 Embeddings

Ideally we would like to come up with a vectorial representation, allowing us to determine the degree of similarity between words.
We will call the resulting (dense) vectors **embeddings**.



2.6 Distributional Semantics

Consists of guessing the meaning of a term by analysing the context of the text.
The dot product gives us a similarity measure. This measure increases with the number of contexts.



“Can I use your ______ ?”
“Sure, the _______ is in the living room”
“Shhh, he’s on the _______ .”
“Pick up the _______ and call me”

- What words could appear in these contexts?
  - Phone [Y, Y, Y, Y];
  - Dog and bone [Y, Y, Y, Y];
  - bathroom [Y, N, N ,N];
  - TV[Y, Y, N,N];
  - couch [Y, Y, Y, N];
  - Geyser [N, N, N, N]

2.7 Cosine Similarity

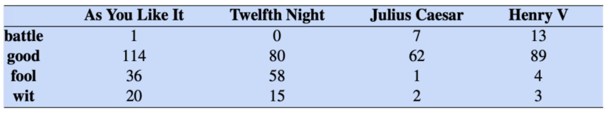
The cosine similarity metric between two vectors is computes as:

$$cosine(v, w) = \frac{v \cdot w}{|V||w|} = cost(\theta) \qquad |v| = \sqrt{\sum_{i=1}^{N} v_i^2} = \sqrt{v \cdot v}$$

It is a normalised version of the dot product.

2.8 Term-Document Matrix

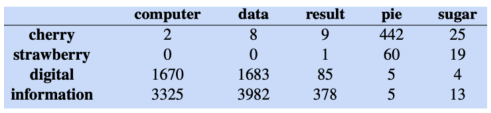
Words in rows and docs in columns, each element i, j corresponds to the frequency of word i in the document j.



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

2.9 Term-Term Matrix

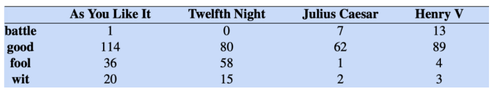
Each column now refers to a word: for each word in a row i, we count, for each column j, the number of times the column word appear inside a window of size L, at the left or at the right, in any document.



	computer	data	result	pie	sugar
cherry	2	8	9	442	25
strawberry	0	0	1	60	19
digital	1670	1683	85	5	4
information	3325	3982	378	5	13

2.10 TF-IDF for Term-Document Matrices

TF (Term Frequency): $tf_{t,d} = \log_{10}(count(t,d) + 1)$
IDF (Inverse Document Frequency): $idf_t = \log_{10}(\frac{N}{df_t})$
N: tot documents, df_t**:** number of docs in which t appear.
TF-IDF: $w_{t,d} = tf_{t,d} \times idf_t$



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

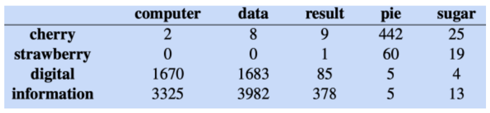
2.11 PPMI for Term Term Matrices

PMI: measures how often two events x and y occur, compared with what we would expect if they were independent

$$PMI(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

PMI values range from negative to positive infinity, **PPMI** restricts to positive values only

$$PPMI(w,c) = max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$



	computer	data	result	pie	sugar
cherry	2	8	9	442	25
strawberry	0	0	1	60	19
digital	1670	1683	85	5	4
information	3325	3982	378	5	13

2.12 Latent Semantic Analysis

Representing a word with a term-term matrix generate long vectors. Consider a matrix C where $c_{i,j}$ represent the occurrence of term i in context j. It can be written as a product of 3 matrices: $C = USV^T$, where $C = m \times n$, $U = m \times I$, $S = |x|$, $V = I \times n$. S is a diagonal matrix, the values in the diagonal matrix are the singular values.
The goal is to approximate C with a smaller matrix:

$$\tilde{C} \rightarrow \min_{u,v} ||C - \tilde{C}(u,v)||_F$$

where the Frobenius norm is

$$||A||_F = \sqrt{\sum_i^m \sum_j^n |a_{ij}|^2}$$

The two matrices have the same dimensions, but we will express $\tilde{C}$ in terms of two submatrices v,u:

- **Context vectors:** v (columns)
- **Term vectors:** u (rows)

2.13 Word2vec: Skip-Gram Model

We want our classifier to estimate the probability that the word is in the context: $P(+|w,c)$.
The idea is to express the probability in terms of the similarity of the two words: $Similarity(w,c) \sim c \cdot w$.

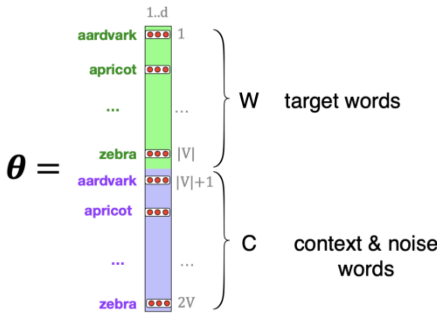
$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)} \qquad \rightarrow \qquad \log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log(\sigma(c_i \cdot w))$$

We want to maximise the probability of context words and minimise the probability of non-context words:

$$L(\theta) = \sum_{(t,c)\in +}$$

Each word is represented by a d-dimensional vector, and has 2 representations: as w and as c(we have $d * 2|V|$ parameters in our model)
After learning we can sum the vectors w_i and c_i for word i or just use w_i. The resulting vector are dense.
For each positive pair, we create k negative pairs by randomly selecting a context word with probability P_α (α usually set to 0.75):

$$P_\alpha(w) = \frac{coount(w)^\alpha}{\sum_{w^t} count(w')^\alpha}$$



2.13.1 Matrix Factorization

Skip-gram with negative sampling model is connected to matrix factorization.

$$C_{i,j} = PMI(i,j) - \log k$$

with PMI of word i in context j and k is the number of negative examples sampled.
We can use PPMI to avoid -infinite values when $count(i,j) = 0$.

2.14 Embeddings Evaluation

- **Intrinsic Evaluation:** comparison between dimensionality and training words, cosine similarity between words.
- **Extrinsic Evaluation:** as compare word embeddings and n-gram as input to a number of downstream tasks.

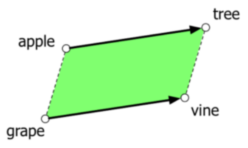
2.15 Visualizing Word Embeddings

Word embeddings can be used to visualise the meaning of a word by:

- Listing the words in V with highest cosine similarity with the word.
- Project the d dimensions of a word embedding down into 2 dimensions (t-distributed stochastic neighbour embedding (t-SNE)).

2.16 Semantic Properties of Word Embeddings

A property of embeddings is their ability to capture relational meanings. Word embeddings can be use to solve analogy problems. The parallelogram can be used for solving such problems:



2.17 Diachronic Word Embeddings

It is possible to compare embeddings trained on different data (but related). We need to align the two embeddings before comparing them

$$R^{(t)} = arg \min_{Q^TQ=I} ||W^{(t)}Q - W^{(t+1)}||_F$$

where Q is a transformation (rotation and alignment) matrix.
This is called the orthogonal Procrustes problem and its solution is UV^T where U,V are the result of the SVD decomposition of $W^{t+1}(W^t)T$.

2.18 Embedding for Polysemous Words

For the embedding of words with multiple meaning, the resulting embedding vector is a linear combination of the vectors of the multiple means.

2.19 FastText

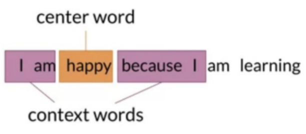
Useful when words have internal structure, like in rich morphological languages.
FastText compute representations for char n-grams and represent a word by the sum of the embeddings of the char n-gram comprising it.
The best n depends on the language and the task.

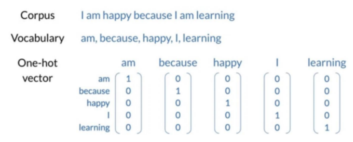
2.20 Glove

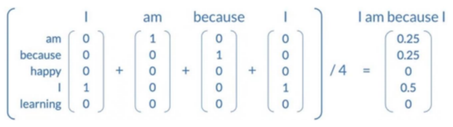
Combines models like skip-gram that use local information on the context window with global information.
With some degrees of semplification, the embeddings are initialised based on global co-occurrence statistics between words.

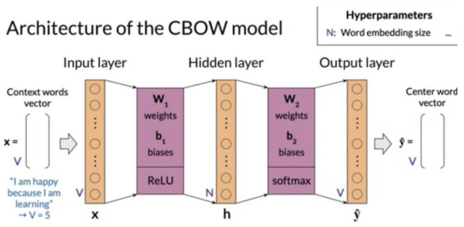
2.21 Continuous Bag of Words (CBOW) Model

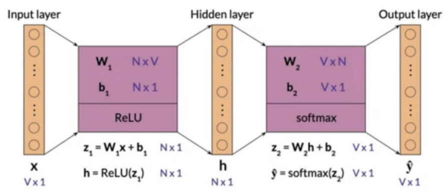
1. Take a huge text corpus.
2. Go over the test with a sliding window, moving one woed at a time.
   - At each step, there is a center word and context words
   - Predict the center word probabilities given the context words.
3. Adjust the vectors to increase these probabilities.




Corpus
I am happy because I am learning
Vocabulary
am, because, happy, I, learning
One-hot vector
am
because
happy
I
learning

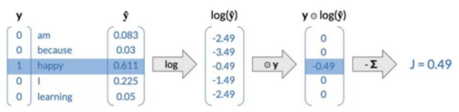


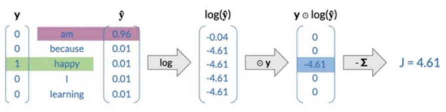




1. Each column of W_1 is the embedding of the corresponding word.
2. Each row of W_2 is the embedding of the corresponding word.
3. The average of W_1 and W_2 transposed

Context words are one hot encodings averaged and N is the size of the embeddings.
The ordering of the words is not modelled.





The loss function is a Cross Entropy Loss.

3 Language Models

LMs estimate the probability of different linguistic units: symbols, tokens, token sequences.

3.1 Estimating Sentence Probabilities

Any model that computes the probabilities to see the sentence formed by some words is called a Language Model

$$P(W) = P(w_1, \ldots, w_n) \; or \; P(w_4|w_1, w_2, w_3)$$

This estimator is correct in the limit. We need a lot of data to correctly estimate a sentence.

3.2 N-gram Language Models

- n: model's order
- V the vocabulary
- $P(y|u)$ how likely it is to observe y after having seen u of size n?

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

If n is too small, it might fail to model important dependencies.
If n is set too small, it might fail to model important dependencies

$$P(w_1, 1_2, \ldots, w_n) \approx \prod_i (P(w_i)$$

To avoid this problem, we use $\log(p)$.
We have to deal with the **sparsity problem:**

- Some sequences may not be appear in the corpus we use for estimating the data.
- $P(w|\textit{students opened their}) = 0$, we will never generate the sentence "students opened their books".
- What if $N(\textit{students opened their}) = 0$?
- Recovering sparsity issues by enlarging the corpus used for estimation.

3.2.1 Backoff and Interpolation

Backoff and interpolation techniques deal with words that are in our vocabulary, but in the test set combine to form previously unseen contexts.
Backoff:

- If a n-gram is not present, use the probability of the n-1-gram, etc ...
- Stupid backoff: if $P(n - gram) = 0$ then return $kP((n-1) - gram)$, for sufficiently small k.

Interpolation:

- $a \times P(w_n|w_{n-1}, w_{n-2}) + b \times P(w_n|w_{n-1} + c \times P(w_n)$
- $a + b + c = 1$

3.2.2 Out of Vocabulary Words

A new text may contain words that are out-of-vocabulary. The idea is to restrict the vocabulary to the most frequent words, and to convert all other tokens to a special 'unknown word' token, UNK. When we compute the probability of a new text, we first replace every out-of-vocabulary word with UNK.

3.2.3 Evaluation of Language Models

- **Intrinsic Evaluation:** how does the model scores with respect to a given evaluation function.
- **Extrinsic Evaluation:** how does the method help with the application is is embedded in?

Held out: to data not used during training.
What is the probability that our model generated that data?

$$P(w_1, w_2, \ldots, w_n) = \prod_{n=1}^{m} p(w_1|w_{j<i}$$

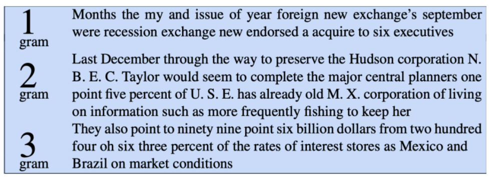
$$l(w) = \sum_{m=1}^{M} \log p(w_m|w_{m-1}, \ldots, w_1)$$

M is the number of tokens in the held-out corpus, and $-l(w)$ is the cross entropy loss.

$$PERPLEXITY(w_1, \ldots, w_m) = 2\frac{-l(w)}{M} = 2^{-\frac{1}{M}\sum_{i=1}^{M} \log_2 p(w_t|w_{j<t})}$$

If the model chooses randomly, $PERPLEXITY = |V|$ where V is the output vocabulary.
The $PERPLEXITY$ of k means that at each step the model has k equally likely best options.

3.2.4 Output



1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

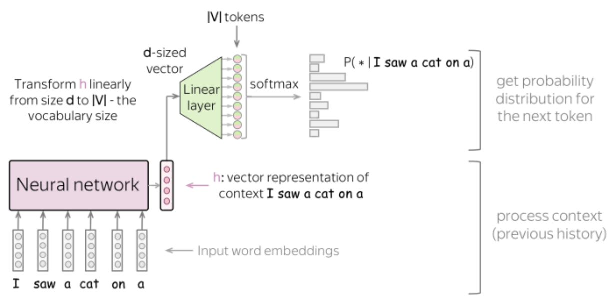
In N-gram models the choice of n has a profound impact on the system. If n is too high, it is hard to get meaningful estimations for the conditional probabilities. If n is set too small, it might fail to model important dependencies.
Unfortunately the choice of n is global (not optimal for every context).
Neural models are more flexible: given a large context, they learn to focus on those small subsets of words that are more interesting.

4 Neural Language Models

The idea is to:

1. Get a vector representation for the previous context.
2. Generate a probability distribution for the next token.

First bullet depends on NN architecture, second bullet is model-agonistic.
Most natural choice for NN architecture is RNN but FNN and CNN have also been exploited.



|V| tokens
d-sized vector
Transform h linearly from size d to |V| - the vocabulary size
Linear layer
softmax
P(* | I saw a cat on a)
get probability distribution for the next token
Neural network
h: vector representation of context I saw a cat on a
Input word embeddings
process context (previous history)
I saw a cat on a

4.1 Feedforward NLM

4.1.1 Inference

$$P(w_t|w_{1:t-1}) \approx P(w_t|w_{t-N+1:t-1})$$

For $w \in V$, let $ind(w) \in [1 \dots]$ be the index associated with w.
We represent each input word w_t as a **one-hot vector** x_t of size $|V|$, defined as: element $x_t[ind(w_t)]$ is set to one and all the other elements of x_t are set to zero.
At the first layer:

1. We convert one-hot vectors for the words in thee N-1 window into word embeddings of size d.
2. We concatenate the N-1 embeddings.
3. The first hidden layer equation is $e_t = [Ex_{t-3}; Ex_{t-2}; Ex_{t-1}]$

Where:

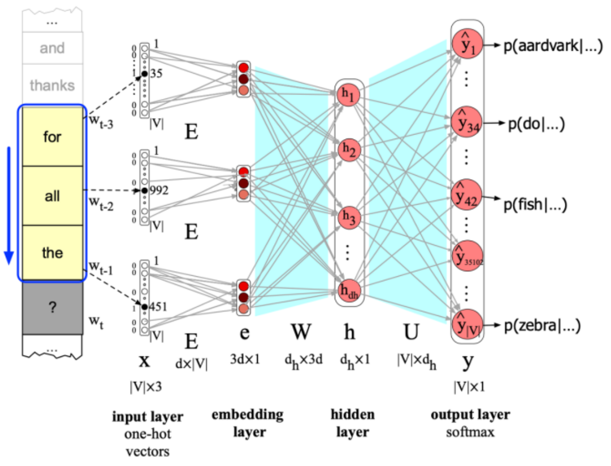
- $E : d \times |V|$ is a learnable matrix with the word embeddings.
- $x_{t-i} : |V| \times 1$ are 1-hot representation of word w_{t-1}.
- $e_t : 3d \times 1$ is the concatenation of the embeddings of the N-1 previous words.

The model equations for the remaining layers are:

- $h_t = g(We_t + b)$
- $z_t = Uh_t$
- $\hat{y_t} = (z_t)$

Where:

- $W : d_h \times 3d$ is a learnable matrix, d_h the size of the second hidden vector representation.
- $b : d_h \times 1$ is a learnable vector.
- $h_t : d_h \times 1$ is obtained through some activation function g.
- $U : |V| \times d_h$ is a learnable matrix.
- $z_t, \hat{y_t} : |V| \times 1$ scores and distribution.



The vector z_t can be thought as a set of scores over $|V|$, also called **logits**: raw predictions that a classification models generates. Passing these scores through the softmax function normalizes into a probability distribution.
The element of $\hat{y}_t$ with index $ind(w_t)$ is the probability that the next word is w_t:

$$\hat{y}_t[ind(w_t)] = P(w_t w_{t-3:t-1})$$

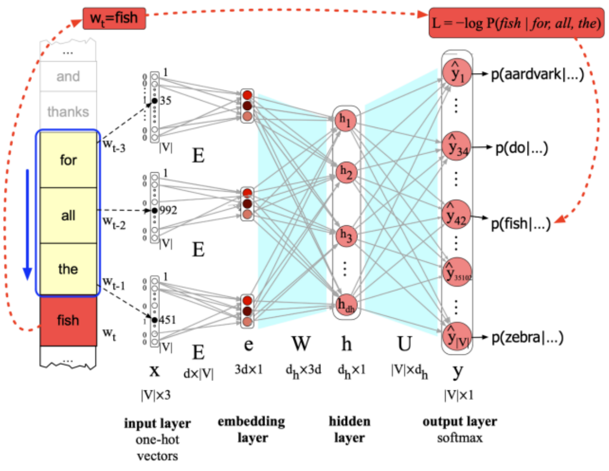
4.1.2 Training

The parameters of the model are $\theta = E, W, U$. The number of parameters is $|V|$, since d is a constant.
Let w_t be the word at position t in the training data. Then the **true distribution** y_t for the word at position t is a 1-hot-vector of size $|V|$ with:

- $y_t[ind(w_t)] = 1$
- $y_t[k] = 0$ everywhere else.

We use the cross-entropy loss for training the model:

$$L_{CE}(\hat{y}_t, y_t) = -\sum_{k=1}^{|V|} y_t[k] \log \hat{y}_t[k] = -\log P(w_t w_{t-N+1:t-1})$$



4.1.3 Discussion

Feedforward NLM learns word embeddings simultaneously with training the network.
The text generated by sampling our NLM should be **coherent** and **diverse**, but there is a tradeoff between the two.
Contrastive evaluation is used to test specific linguistic constructions in NLM.

4.2 Text Generation

4.2.1 Top-k Sampling

Select the k most likely words, renormalize the probability and then apply random sampling.
Works if the top k words include most of the probability mass. If the distribution is almost flat, the top-k represent only a small portion of the probability mass.
Select the smallest set of words $V^{(P)}$ such that:

$$\sum_{w \in V^{(P)}} P(w|w_{<t}) \geqslant p$$

The goal is to have a measure that is able to capture the most likely word such that their probability mass is maximised.

4.2.2 Temperature

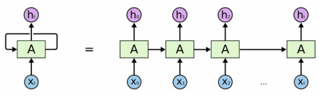
Temperature reshapes the probability distribution:

$$y = softmax(V/\tau) = \frac{\exp(\frac{v_i h_t}{\tau})}{\sum_j \exp(\frac{v_j h_t}{\tau})}$$

where V_i denotes the i-th row of V and $\tau = 1$ leaves y unchanged, $\tau > 1$ flattens it, $\tau < 1$ makes it skewed.

4.3 Recurrent Neural Networks

- Unfolding the network reach horizontal deepness.
- Every time step uses the same shared weight.
- The output can be influenced by the whole sequence seen so far.
- The hidden state can be a summary of the input sequence.



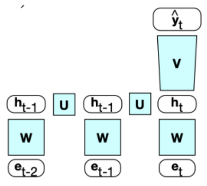
The standard way to perform training of RNNs parameters is via the back-propagation through time (BPTT). Basically is a standard back propagation on the unrolled network. The shared weights are updated by summing over the gradients computed for each position.

4.3.1 Inference

RNN language models process the input one word at a time, predicting the next word from the **current word and the previous hidden state**.
RNNs can model probability distribution $P(w_t|w_{1:t-1})$ without the $N-1$ window approximation of feedforward NLM.
The hidden state of the RNN model can in principle represent information about all of the preceding words.

The model equations are (for some h_0):

- $e_t = E_{x_t}$
- $h_t = g(Uh_{t-1} + We_t)$
- $\hat{y_t} = softmax(Vh_t) = softmax(E^T h_t)$ since $E^T = V$.



where (d_a is the size of the hidden vectors)

- $x_t : |V| \times 1$ is a 1-hot representation of word w_t.
- $E : d \times |V|$ is a learnable matrix with the word embeddings.
- $U, W : d \times d$ are learnable matrices.

- $h_t : d \times 1$ is the hidden vector at step t.
- $V : |V| \times d$ is a learnable matrix.
- $\hat{y_t} : |V| \times 1$ is a probability distribution.

The vector resulting from Va_t records the **logits** over the vocabulary, given the evidence provided by a_t. The softmax normalized the logits, resulting in the **estimated distribution** $\hat{y_t}$ for the word at step t.

4.3.2 Training

The number of parameters of the model is $O(|V|)$, since d is a constant.
Let y_t be the true distribution at step t. This is a 1-hot vector over, obtained from the training set.
We train the model to minimize the error in predicting the true next word w_{t+1} in the training sequence, using cross-entropy as the loss function.
At each step t during training:

- Prediction is based on the correct sequence of tokens $w_{1:t}$.
- We ignore what the model predicted at previous steps.
- The idea that we always givve the model the correct history sequence to predict the next word is called **teacher forcing**.
  - Teacher forcing has some disadvantages: the model is never exposed to prediction mistakes; therefore, at inference time the model is not able to recover from errors.

4.3.3 Practical Issues

Suffers from the **vanishing gradient** problem:

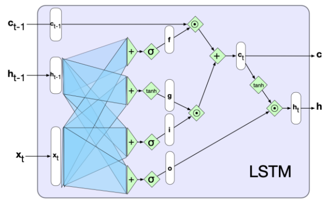
- Past events have weights that decrease exponentially with the distance from actual word w_t.
- Gated recurrent units (GRU) and long-short term memory (LSTM) neural networks are better in capturing long distance relations.

4.4 Bidirectional RNNs

- Combines two independent RNNs, one where the input is processed from the start to the end, and the other from the end to the start.
- We then concatenate the two representations computed by the networks into a single vector $h_t = [h_t^f ; h_t^b]$.

4.5 LSTM

Current RNN models tend to use the **local context information** and have difficulties exploiting distant information.
LSTM network has the goal to decide what information keep for the future.
LSTM introduces gates for that purpose: feedforward NN followed by a sigmoid activation, followed by an element-by-element multiplication:



- **Forget gate:** deletes information from the context that is no longer needed.
- **Add gate:** chooses the information to add to the context.
- **Output gate:** decides what information is required for the current hidden state.

c_{t-1} is the previous context, h_{t-1} is the previous hidden state and x_t the current input.
It outputs the current hidden state and context.

4.5.1 Gates Equations

The forget gate is:

$$f_t = \sigma(U_f h_{t-1} + W_f x_t) \quad k_t = c_{t-1} \odot f_t$$

where k_t is the previous context with some information removed.
The add gate is:

$$g_t = tanh(U_g h_{t-1} + W_g x_t) \quad i_t = \sigma(U_i h_{t-1} + W_i x_t) \quad j_t = g_t \odot i_t$$

where j_t is the information from the previous hidden state that will added to the context.
Finally, the new context is $c_t = j_t + k_t$ and the output gate is:

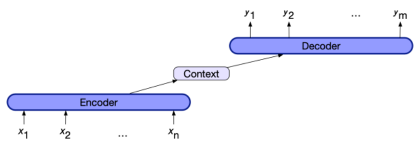
$$o_t = \sigma(U_o h_{t-1} + W_o x_t) \quad h_t = o_t \odot tanh(c_t)$$

5 Neural Architectures for NLP

5.1 Encoder-Decoder Models

5.1.1 Encoder-Decoder

- **Encoder:** $x_1, \ldots, x_n \rightarrow h_1, \ldots, h_n$.
- **Context:** $h_1, \ldots, h_n \rightarrow c$ a compressed representation of $h_1, \ldots, h_n$ into a fixed size vector.
- **Decoder:** $c \rightarrow h_1, \ldots, h_m$ (m might be a different that n).

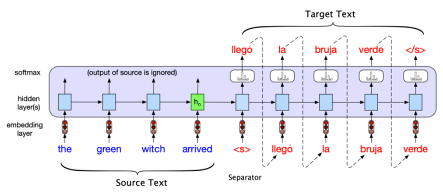


5.1.2 Translation with RNN

A RNN can be used to translate a sentence x in a specific language into a sequence y in another language if we "include" the input text in the context:

$$p(y|x) = p(y_1|x)p(y_2|y_1, x) \ldots p(y_m|y_1, \ldots, y_{m-1}, x)$$

We need to discriminate x (text in the original language) and y (partial translation).
We add a separator, say $< s >$, at the end of the input sentence and we start generating text after that (or we ignore any output before).



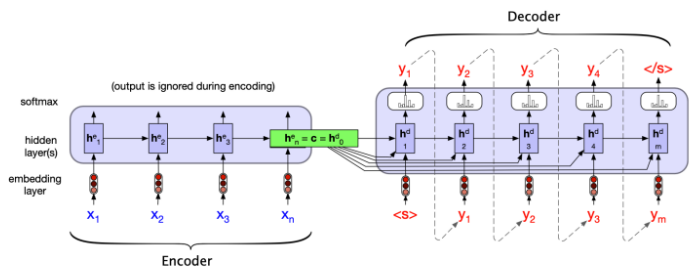
5.1.3 Translation with Encoder Decoder

Following the encoder-decoder structure, we first compute c based on the input sentence, then we start generating y_t based on $y_{<t}$ and c.
In order to avoid polluting c at every time step, we do not update it during generation (each y_t is conditioned also on the same c vector):

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$$

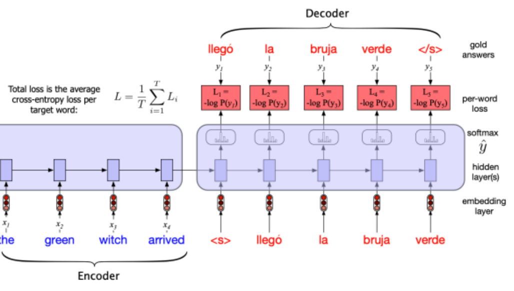
Formally we have:

- $c = h_n^e$
- $h_0^d = c$
- $h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$
- $\hat{y}_t = softmax(h_t^d)$



5.1.4 Training Encoder-Decoder

An input example is a pair of sentences separated by a specific token.
Training is performed as for any NN.
During inference, after an error the decoder might deviate more and more from the correct translation.



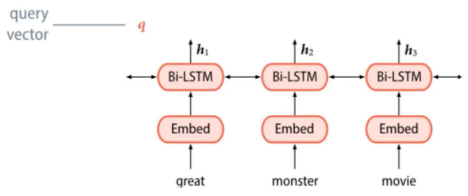
5.2 Attention

Mimics the retrieval of a value v_i for a query q based on a key (address) k_i in a database (memory):

$$attention(q,k,v) = \sum_i similarity(q,k_i)v_i$$

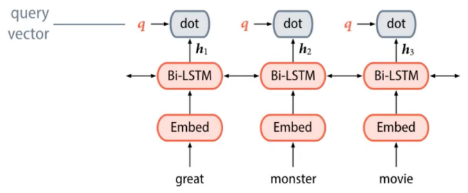
The RNN encoder architecture may not preserve information about all words, but the latest words tend to have higher impact on the summary vector.

Idea: construct a summary vector of all words that is tailored to our goal.
Weight each word according to its relevance to the task and use a query vector q that represent the prototype of a sentiment indicator.

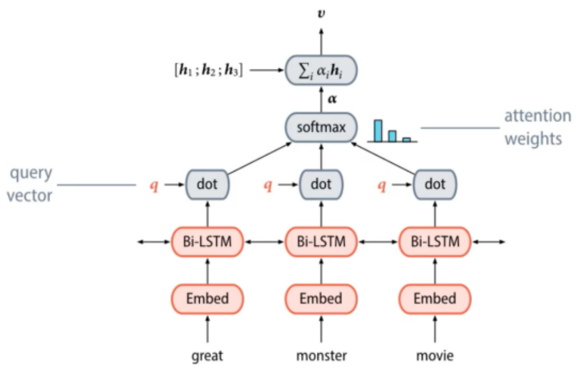


Then, compute the similarity between q and each word:

$$attention(q,k,v) = \sum_i similarity(q,k_i)v_i$$



Compute the similarity between q and each word (h_i).
Compute the softmax of the similarity values and obtain scores α.
The hidden states are composed together with their scores.
We have obtained an alternative vector v which depends on the query q.
The vector q can be learned as any other parameter of the network.



5.2.1 Attention in Encoder-Decoder

Instead of having one single context vector c that summarises the information of the encoder as before, c depends also on the previous state of the decoder (h^d_{i-1}):

$$c_i = f(h^e_1 \dots h^e_n, h^d_{i-1})$$

In the decoding phase

$$h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c_i)$$

The hidden state of the decoding phase depends on a different context for each token. It allows each $\hat{y}_i$ to focus on each encoder state via the custom c_i.
Attention assumes that all hidden states are explicitly available and many-to-many relationship between output and input-hidden states.

5.2.2 Attention Functions

- $similarity(q,k_i) = q^T k_i$
- $similarity(q,k_i) = q^T W k_i$ (weights W to be learned, useful when q,k are not in the same space i.e question and answer).
- $similarity(q,k_i) = w_q^T tanh(W[q;k_i])$
- A kernel function could be used.

5.2.3 Reading Comprehension with Attention

Form two encodings, one for the query and one for each token in the document. The representation r of the document d is formed by a weighted sum of the token vectors. The weights are interpreted as the model's attention.
Define the joint document and query embedding via non linear combination:

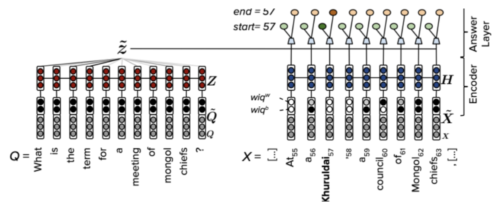
$$g(q, d) = tanh(W_q r(d) + W_2 r(q))$$

5.2.4 Sliding Window for Span-based QA

Sliding window predicts an answer based on a simple lexical information in a sliding window. It maximize the BOW similarity between the answer and the sliding window in the given passage.
The Logistic Regression (SQuAD baseline) using features from the candidates including lengths, bigram frequencies, word frequencies, span POS tags, lexical features, dependency tree path features . . .

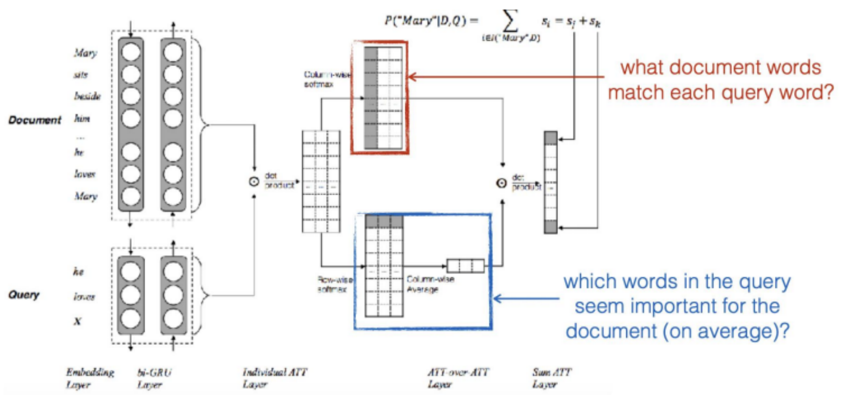
5.3 Neural Approaches

FastQA: basic features indicate whether a token in a passage appears in the question.
The features are weighted w.r.t. the similarity of the context and the query.


end = 57
start= 57
Answer Layer
Encoder
$\tilde{z}$
Z
H
$\tilde{Q}$
Q
$\tilde{X}$
X
wiq^w
wiq^b
Q = What is the term for a meeting of mongol chiefs ?
X = [...] At55 a56 Khuruldai57 58 a59 council60 of61 Mongol62 chiefs63 , [...]

5.3.1 Attention Over Attention

Which document words match best with the most important words in the query?



5.3.2 Multi-Step Reasoning

It might become clear that more information is necessary post-facto.
A general formulation of models that access external memory through attention and specific instantiation for document-level QA.
Idea: compute attention also on the sentence selected at the previous step.
When to stop reasoning?

- After a predefined number of steps.
- When we attend to a "stop reasoning" symbol.
- Have an explicit "stop reasoning" predictor.

6 Transformers

6.1 Transformer-Encoder

The Transformer architecture ditches recurrence and focuses on attention.
It is still a encoder-decoder network that process the whole sentence in parallel.
Information about the position of the words is recovered via positional encodings.
The add step biases the representation to be similar to the input itself.
The normalisation step avoid the system to reach states where the gradient is too small.
The whole block is repeated Nx times in order to connext the "group of words" selected at the previous stage.

6.1.1 Attention

Through the self-attention mechanism an encoder computes a representation of the input that depends on the context.
Scaled Attention: query and key are here pairs of the input words

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V$$

For each word, attention creates an embedding that depends on the other words in the sentence.
Multi-Head attention multiple attention matrices are computed (with different initialization weights) and then combined.

6.2 Transformer-Decoder

Masked attention: some of the values are forced not to be used fot computing attention

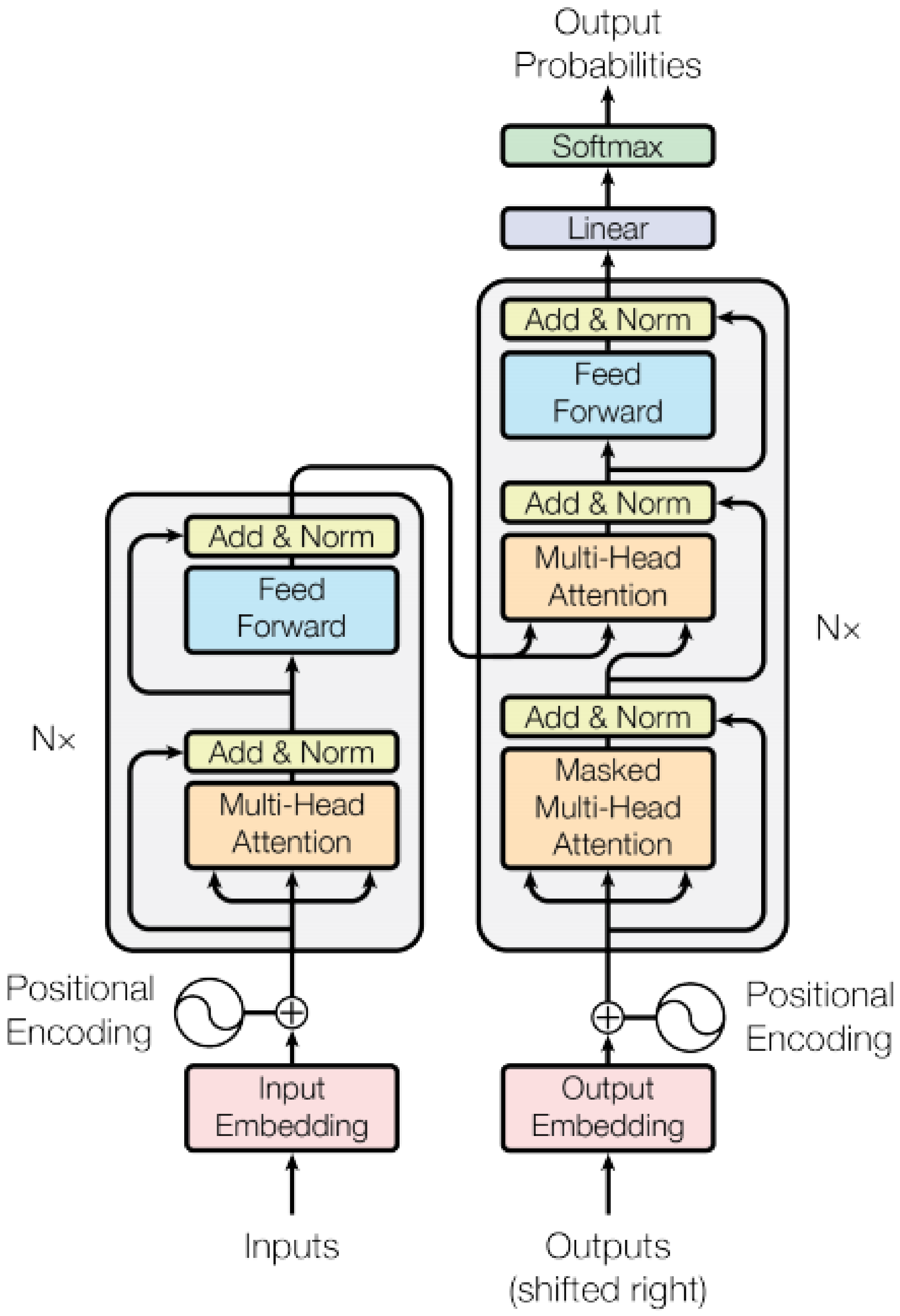
$$maskedAttention(Q, K, V) = softmax(\frac{Q^\top K + M}{\sqrt{d_k}})V$$

Entries that we want to discard are set to -inf in M.
In the multi-head attention the queries are:

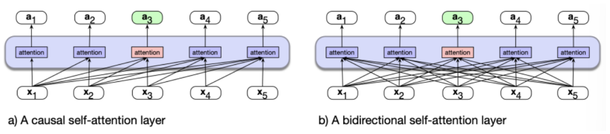
- The previous outputs in the prediction phase.
- The gold labels during training.

6.3 Pretraining

Despite very minor technical differences, the pretraining is basically the same as the Feedforward NLM. Since these models tend to be much larger in size, they are trained on extremely large collections of data.


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

7 Masked Language Models


a) A causal self-attention layer
b) A bidirectional self-attention layer

Besides the difference over which attention is computed, the same transformer architecture is used.
The input sentence first has to be tokenized (BPE tokenizer), and all further processing takes place on subword tokens rather than words.

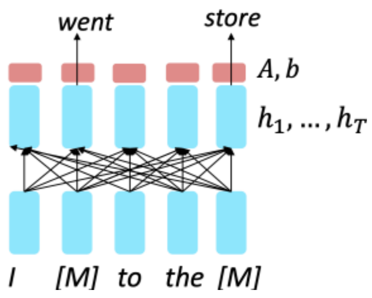
7.1 BERT (Bidirectional Encoder Representations from Transformers)

Is a pre training of Deep Bidirectional Transformers for Language Understanding.
The idea is to replace some fraction of words in the input with a special $[MASK]$ token; predict these words:

$$h_1, \dots, h_t = Encoder(w_1, \dots, w_t)$$

$$y \sim Ax_i + b$$

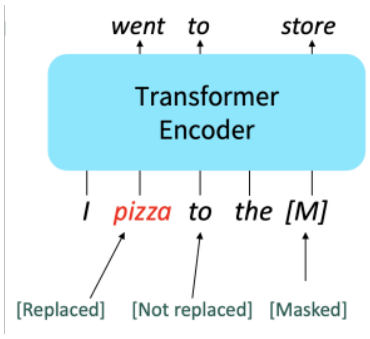
Only add loss terms from words that are masked out. If x' is the masked version of x, we're learning $p_\theta(x|x')$, called **masked LM**.



Predict a random 15% of (sub) word tokens:

- Replace input word with $[MASK]$ 80% of the time.
- Replace input word with a random token 10% of the time.
- Leave input word unchanged 10% of the time, but still predict it.

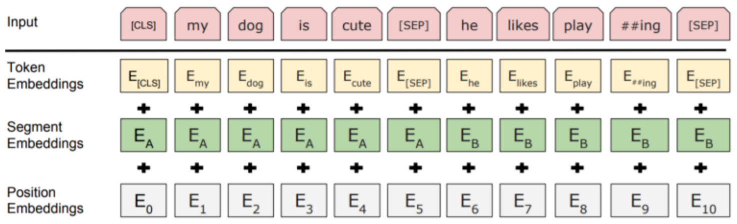
No $[MASK]$ is seen at fine-tuning time because bias the representation towards the actual word.



There are many downstream tasks between pairs of sentences such as: paraphrase detection, semantic textual similarity, sentence entailment, answer sentence selection. This is not directly captured by language modelling.
In order to train a model that understands sentence relationships, BERT introduces a second learning objective called next sentence prediction (NSP). NSP is a binary decision task that can be trivially generated from any monolingual corpus. The NSP input to BERT:

- tokens $[CLS]$ prepended to the first sentence.
- token $[SEP]$ placed between the two sentences and after the rightmost token of the second sentence.
- segment embedding, indicating which sentence the token belongs to.

The pretraining input to BERT was two separate contiguous sentences:


Input
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
Token Embeddings
E_[CLS] E_my E_dog E_is E_cute E_[SEP] E_he E_likes E_play E_##ing E_[SEP]
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_10

The second task is to predict whether one sentence follows the order or is randomly sampled.

7.1.1 BERT-NSP

The training set is composed of pairs of sentences:

- An actual pair of adjacent sentences from the trailing corpus (50%).
- A pair of unrelated sentences randomly selected (50%).

The output embedding associated with the $[CLS]$ token represents the next sentence prediction.
After the training on MLM and NSP, the embeddings are selected from a combination of the topmost layers of the network.
$[CLS]$ embeddings sometimes are also used as sentence embeddings.

7.1.2 RoBERTa

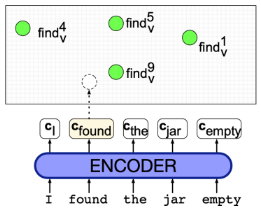
Robustly Optimized BERT Approach is based on a novel recipe for training BERT:

- Longer training over more data.
- No more NSP task.
- Dynamically changing the masking pattern.

XML-RoBERTa is a version trained on data from multiple languages.

7.2 Contextual Embeddings

While static (word2vec) embeddings represent the meaning of word types (vocabulary entries), contextual embeddings represent the meaning of **word instances**.
Contextual embeddings in tasks like measuring the semantic similarity of two words in context.
Since there are several layers in a transformer architecture, there is not a unique choice for extracting embeddings, many times a combination of the latest 4 layers are used.



If we compute the embeddings of all the word senses from a controlled dataset, when we want to establish the meaning of a word instance, we simply compare it with the template word vectors we computed above.

7.2.1 Anisotropy

We can compare two contextual embeddings with the cosine similarity but it has been empirically observed that any two embeddings from BERT have similarity close to 1. Timekey et al propose to standardise the vectors to make them more isotopic:

$$\mu = \frac{1}{|C|}\sum_{x \in C} x \quad \sigma = \sqrt{\frac{1}{|C|}\sum_{x \in C}(x-\mu)^2} \quad \rightarrow \quad z = \frac{x-\mu}{\sigma}$$

7.3 Sentence Embeddings

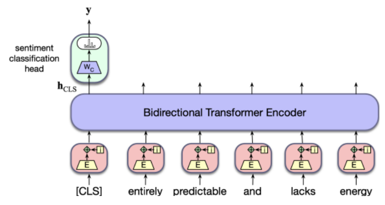
If we have word embeddings of size k for each of the n words in a sentence, we apply a function on vectors

$$f(W) = s$$

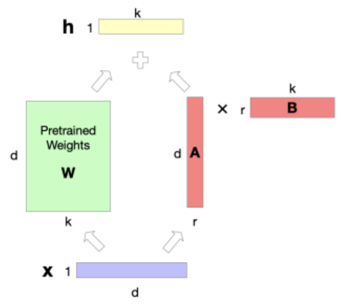
$$W = n \times k; s = 1 \times k$$

7.4 Fine-Tuning

Sequence Classification (text classification): we can fine-tune BERT by adding a classifier (NN) on top of the CLS token and use a loss to adapt the weights of the network.
One way to tackle fine-tuning tasks is to feed the sentences into BERT, separated by the token [SEP], and then add again a classifier over the CLS token.



7.4.1 LORA



Finetuning by keep pretraining on large language models is useful but expensive. Attention computation can be cast as matrix multiplication and thus parallelised over tokens.
Say we need to update a matrix $W : n \times n$. We can be approximated by AB (low rank approximation) of size $A : N \times r$ and $B : r \times d$.
We apply gradient descent to A and B and then we update $W = W + AB$.

7.5 Scaling Laws

The performance (cross-entropy loss) for a transformer architecture depends on N (size of the model vs performance does not seem to depend on the exact architecture), D (dataset size), C (amount of computing).
Performance has roughly a power law relationship with these elements N^a.
Performance improves predictably as long as we scale up N and D in tandem. Every time we increase the model size $8x$, we only need to increase thee data by roughly $5x$ to avoid penality.
Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps and using fewer data points.
If C is fixed, we attain optimal performance by training very large models and stopping significantly short of convergence.

8 Decoder Only Models

8.1 Generative Models

An alternative way to solve learning tasks is to tweak the output of a language model so that, given x_i as context, y_i is output.
In addition to x_i, we can simply describe what we want in natural language → **prompts**.

8.2 Prompt Engineering

Prompts need to be designed unambiguously, any reasonable continuation would accomplish the task.
We can make contrasting prompts if we want to use an LLM for a classification task on multiple instances.
When using prompting on lots of data, we should expect to do some postprocessing of the outputs.
Given an {input} text, prompt examples are:

- Summarization
- Translation
- Sentiment
- Fine-Grained-Sentiment

If we want to address multiple tasks with one prompt we can use templates for the input.
With prompts we can also: restrict a summary to be a particular length, have an answer generated according to some kind of persona or role, specify a more structured output using a specific format.
Few-shot prompting: prompting with instructions and some examples as input.
Zero-shot prompting: prompting with just instructions as input.

8.2.1 Demonstrations

We can also provide examples (**demonstrations**) to help make the instructions clearer.
The primary benefit in examples is to demonstrate the task to be performed to the LLM and the format of the output sequence.
Adding too many examples seems to cause the model to overfit to details of the exact examples chosen and generalize poorly.
Using demonstrations that are similar to the current input seems to improve performance.
We can refer to prompting as **in-context learning**.
A **Limitation of prompting** is that generating continuations consistent with its context may not always be forced with prompting to generate the output we want.

8.2.2 Instruction Tuning

The models are finetuned on a corpus of instructions and questions with their corresponding responses.
Is a form of supervised learning where the training data consists of instructions (instruction prompts and input/output demonstrations pairs).
We continue training the model on them using the same language modelling objective used to train the original model and we penalise outputs of the LLMs that do not correspond exactly to the one provided as example.

8.2.3 Chain of Through Prompting

The goal is to improve the performance on reasoning tasks.
The intuition is that people solve these tasks by breaking them down into steps.
We encourage language models to break them down in the same way and one way to do it is to add to each of the demonstrations in the few-shot prompt some text explaining the reasoning steps.

8.2.4 Prompt Optimisation

1. Start state (prompt human or machine generated).
2. Scoring function (depends on the task we are solving).
3. Expansion method (how to generate variations of the prompts).

8.3 GPT (Generative Pretraining Transformer)

Is a left-to-right (casual) language model. Is based on transformer's decoder-only, no cross-attention layer and the layer normalization is moved to the input of each sub-block.

8.3.1 GPT-3

Is the same model with the difference that includes (casual) attention sparsity at half of the layers, computing only a subset of the elements of the attention matrix.
Few-shot learning: ability of an LLM to solve a problem on the basis of a few examples.
Zero-shot learning: ability of an LLM to solve a new problem solely on the basis of NL instructions.

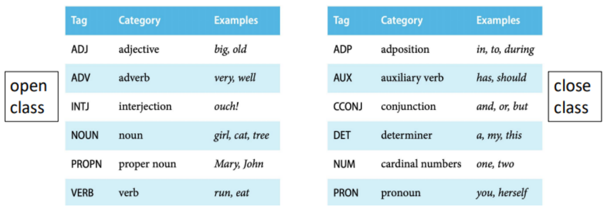
9 Sequence Labelling

Assign discrete labels to discrete elements in a sequence. There are two types of approaches:

- **Local search:** sequence of classification problems where we aim to predict the label for each position in the input sequence.
- **Global search:** combinatorial optimisation problem over the full set of candidate sequences.

9.1 Part of Speech

A part of speech is a category of words that play similar roles within the syntactic structure of a sentence. All speakers share the words in the closed POS classes, might have different words for the open ones. Two popular **tagsets** are the *Penn Treebank POS tags* and the *Universal Dependencies*.



open class

Tag	Category	Examples
ADJ	adjective	*big, old*
ADV	adverb	*very, well*
INTJ	interjection	*ouch!*
NOUN	noun	*girl, cat, tree*
PROPN	proper noun	*Mary, John*
VERB	verb	*run, eat*

close class

Tag	Category	Examples
ADP	adposition	*in, to, during*
AUX	auxiliary verb	*has, should*
CCONJ	conjunction	*and, or, but*
DET	determiner	*a, my, this*
NUM	cardinal numbers	*one, two*
PRON	pronoun	*you, herself*

9.1.1 Part of Speech Tagging

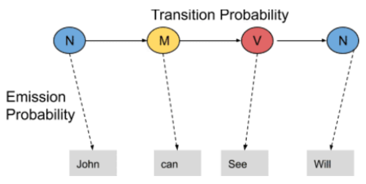
The process of assigning a part-of-speech or lexical class marker to each word in a corpus.
We need a disambiguation task since terms might have more than one POS.
The rule based approach:

1. Start with a dictionary.
2. Assign all possible tags to words from the dictionary.
3. Write rules by hand to selectively remove tags.
4. Leave the correct tag for each word.

9.1.2 Hidden Markov Models (HMM)

We cannot use Markov Chains to compute probabilities of POS tags because we do not observe them. Rather, we see words, we must infer the tags from the word sequence. Therefore we call the tags **hidden**.

$Q = q_1, q_2, \dots, q_N$ a set of N **states**.
$A = a_{11} \dots a_{ij} \dots a_{NN}$ a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j.
$B = b_i(o_t)$ a sequence of **emission probabilities**, expressing the probability of an observation o_t being generated from a state q_i.
$\pi = \pi_1, \dots, \pi_N$ an **initial probability distribution over states**, where π_i is the probability that the chain will start in state i.



The emission probability depends only on the state producing it (independent from the other states). Given A and B estimated from a training corpus. We want to:

- Estimating the probability of a tag t_i following t_{i-1}:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

- Estimating the probability of a word w_i given a tag t_i:

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

9.1.3 Decoding

Is the task of determining the hidden variables sequence corresponding to the sequence of observation. HMM makes two assumptions: the Markov Assumption $P(t)$ depends only on the previous tag and $P(w_i)$ depends only on the emitting tag t_i

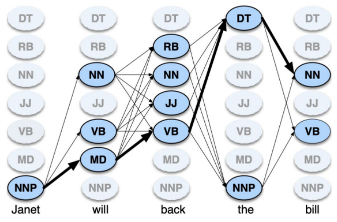
$$\underset{t_1 \ldots t_n}{argmax} P(t_1 \ldots t_n | w_1 \ldots w_n) \approx \underset{t_1 \ldots t_n}{argmax} \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1})$$

where the term inside the production is the **emission transition**.

9.1.4 Viterbi Algorithm

Given an input sentence and an HMM, finds the most probable sequence of POS tags.

It builds a matrix of states (POS tags), one for each input word.
For each state we consider the most probable path that lead to it.
For example there are different ways to reach RB, we keep only the highest probability path.
This allow us to avoid checking all possible combinations of tags.
As with the n-gram language models, there are a number of limitations, for example unknown words.



9.1.5 POS Tagging Classifier

Sliding Window Classifier: each token is classified independently, but features are derived from surrounding tokens.

9.1.6 Inference

We could phrase the problem of a global labelling as:

$$\hat{y} = \underset{y \in \gamma(w)}{argmax} \Psi(w, y)$$

However, if we consider the length of the sentence and the number of POS tags, the size of the space Y grows exponentially wrt them.
If the scorer function is decomposable we can find faster algorithms such as the Viterbi algorithm.

9.1.7 POS with Perceptron

1. Predict the best tag sequence z using Viterbi with current settings.
2. For each feature vector connected to a wrongly predicted tag (wrt y):
   (a) Increase the score of the correct features, decrease the score of the incorrect features.

9.2 Named Entity Recognition

Name Entity Recognition (NER): find spans of text that constitute proper names and tag the type of the entity:

- **PER** (person), **LOC** (location), **ORG** (organization), or **GPE** (geo-political entity).
- Other type of entities.

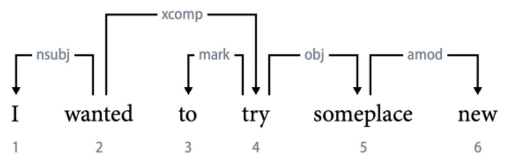
The task is at token-level. One waw to tackle it is to use **BIO Tagging** (begin, inside, outside), where we transform our input so that each token gets associated a BIO tag:

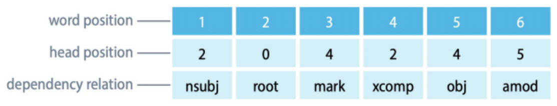
- Begin: if it is the starting token of an entity.
- Inside: if is not the first token of an entity.
- Outside: if it does not belong to an entity.

To tackle it, we can add a classifier on top of each input token whose output is either BIO and then compare it with the expected BIO tag.

10 Parsing

Constituent: group of words within utterances that can be shown to act as a single unit.
Constituency Tree: shows how a phrase is formed, but contains many grammatically correct but semantically unreasonable parses.
Dependency Parser: is an acyclic graph with a root node. Words appear in internal nodes and descendants modify (enrich) the head node.
Dependency Tree: Highlights important semantic information. The head is connected to dependent words and plays the role of the central organizing word, and the dependent as a kind of modifier. Edges might be labelled with the type of relation.





word position	1	2	3	4	5	6
head position	2	0	4	2	4	5
dependency relation	nsubj	root	mark	xcomp	obj	amod

10.1 Graph-Based Dependency Parsing

- Cast parsing as a combinatorial optimisation problem over a set of dependency trees.
- Full trees as input.

Given a sentence x and a set $Y(x)$ of candidate dependency trees for x, we want to find a highest-scoring tree $\hat{y} \in Y(x)$:

$$\hat{y} = \underset{y \in Y(x)}{argmax} score(x, y)$$

The computational complexity depends on the choice of $Y(x)$ and the scoring function.
Under the arc-factored model, the highest-scoring dependency tree can be found in $O(n^3)$ time (n = sentence length).
Extending arc-factored model makes the problem intractable.

10.1.1 The Arc-Factored Model

The score of a dependency tree is expressed as the sum of the scores of its arcs:

$$\hat{y} = \underset{y \in Y(x)}{argmax} \sum_{a \in y} score(x, a) \quad \leftarrow \textit{head-dependent arc}$$

The score of a single arc can be computed by any learning algorithm that receives the head and the dependent as input.

10.2 Transition-Based Dependency Parsing

- Cast parsing as a sequence of local classification problems; at each point in time, predict one of several parser actions.
- Builds the tree step by step.

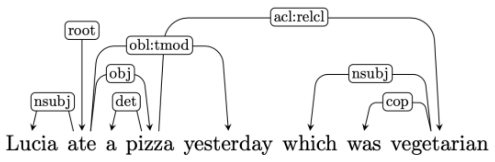
Use local classifier to build one single dependency tree step by step and usually linear w.r.t. the length of the input sentence.

1. Given an input sentence, start with an empty dependency tree.
2. Then calls a classifier, which predicts the transition that the parser should make the mode to a next configuration (extend the partial dependency tree).
3. The process is repeated until the parser reaches a terminal configuration (complete dependency tree).

10.2.1 The Arc-Standard Algorithm

Can only predict projective dependency trees:

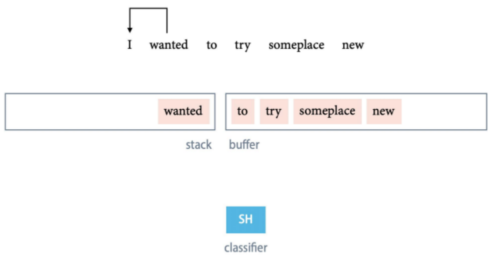
- Given a head and a dependant, there is a path to every word between the head and the dependant.
- Non projective tree: pizza is not connected to yesterday.



Arc-standard consists of:

- **Buffer:** contains those words in the sentence that still need to be processed.
- **Stack:** contains those words in the sentence that are currently being processed.
- **Partial Dependency Tree:** initially, this tree contains all the words of the sentence, but no dependency arc.

Shift transition (SH): removes the frontmost word from the buffer and pushes it to the top of the stack.
Left-arc transition (LA): created a dependency from thee topmost word on the stack to the second-topmost word, and pops the second-topmost word.
Right-arc transition (RA): created a dependency from the second-topmost word on the stack to the topmost word, and pops the topmost word.

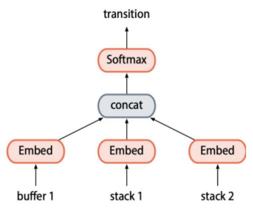


At every stage, a classifier decides the best action between SH, LA or RA.
SH is valid if the buffer contains at least one word.
LA/RA are valid if the stack contains at least two words.
The number of transitions that the arc-standard algorithm takes to build a tree for a sentence with n words is $2n-1$.
Soundness: every valid transition sequence that starts in the initial configuration and ends in some terminal configuration builds a projective dependency tree.
Completeness: every projective dependency tree can be built by some valid transition sequence that starts in the initial configuration and ends in some terminal configuration.

The classifier can use the following features:

- The words in the buffer.
- The words on the stack.
- The partial dependency tree.

The classifier has been implemented as a FFNN.



10.3 Neural Approaches

Kiperwasser et al. embeds the representation of the top 3 words in the stack and one word in the buffer. The algorithm has been extended to be a scorer for graph-based approaches. Given a pair of words, score them as a (head,dependant) pair. The loss maximises margin (y^* gold label):

$$L(\theta) = \max(0, 1 + \underset{y \neq y^*}{maxscore}(x, y) - score(x - y^*))$$

Dozat et al. used two FNN specialised representations of each word as head and dependent. Specialised representations are then scored to get a final score for the arc.

$$h_i = FNN_h(v_i) \quad d_i = FNN_d(v_i)$$

11 Sentiment Analysis

11.1 Theory of Emotions

- **Elkman theory:** there are 6 emotions, supposedly present in all cultures.
- **Plutchik theory:** there are 8 emotions in 4 opposing pairs.
- **Others:** identify 3 axis as valence, the pleasantness of the stimulus; arousal, the intensity of emotion provoked by the stimulus; dominance, the degree of control exerted by the stimulus.

11.2 Lexicon-based Sentiment Analysis

When dealing with documents, combining BOW representation with the sentiment of the words could suffice. We would expect the words associated with its true sentiment to overwhelm the others.
For sentences or short documents this is less effective but a partial solution can be computed by using n-grams and compute sentiment for each n-gram or use the syntax tree to establish relationships between negation and the object negated.

11.2.1 Sentiment Lexicons

Binary lexicons representing the sentiment (valence) of words and also lexicons considering further aspects.

- **LIWC, Linguistic Inquiry and Word Count:** widely set of 73 lexicons containing over 2300 words (include lexicons category).
- **Non Binary Lexicons:** the NRC Valence, Arousal and Dominance (VAD) assigns values to 20000 words.
  - The annotation was made by asking to select best/worst among four terms. The score of the term is the percentage of times it has been best-percentage of times it has been worst.

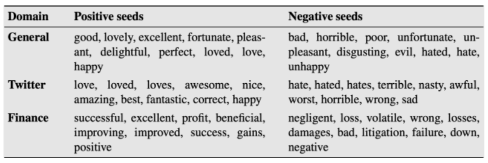
11.2.2 Learning Lexicons

Lexicons might need to be adapted to the current domain, culture or context. Hence the need to learn/ fine tune them from data.

11.2.3 Semantic Axis

1. Define positive and negative vectors in a word embedding space.
2. Define the axis between such vectors.
3. Project words into the axis and determine whether the positive or the negative is closer.

The positive and negative vectors are dominated on the basis of a set of initial seed words.
Words having different sentiment in different contexts can be left to the learning algorithm to deal with.
Different seed words can be provided for each context.



Domain	Positive seeds	Negative seeds
General	good, lovely, excellent, fortunate, pleasant, delightful, perfect, loved, love, happy	bad, horrible, poor, unfortunate, unpleasant, disgusting, evil, hated, hate, unhappy
Twitter	love, loved, loves, awesome, nice, amazing, best, fantastic, correct, happy	hate, hated, hates, terrible, nasty, awful, worst, horrible, wrong, sad
Finance	successful, excellent, profit, beneficial, improving, improved, success, gains, positive	negligent, loss, volatile, wrong, losses, damages, bad, litigation, failure, down, negative

Given a set of embeddings for positive (negative) seed words, we compute $V^+(V^-)$ the vector representing positive (negative) sentiment:

$$V^+ = \frac{1}{n}\sum_{i=1}^{n} E(w_i^+)$$

The axis is defined as

$$V_{axis} = V^+ - V^-$$

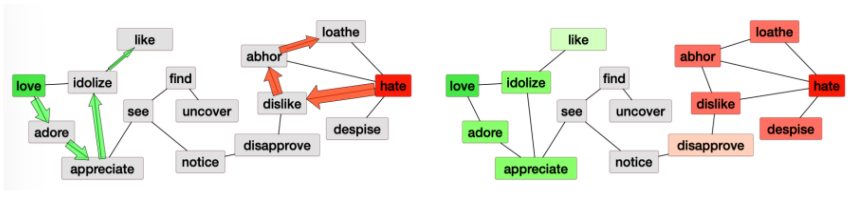
A new word is scored as

$$score(w) = \frac{E(w)\cdot V_{axis}}{||E(w)||||V_{axis}||}$$

11.2.4 Label Propagation

1. Define a graph: one node per word, an edge for each of the k most similar words.
2. Define seed words.
3. Propagate polarities on the graph from seed words: perform random walks starting from each node and moving to a neighbouring node with probability proportional to the edge weight.

Score the nodes in the graph: count the percentage of times a node has been reached from any walk starting from a positive labelled node w.r.t. the total number it has been visited.
Confidence values for the scores can be obtained by repeating the walks from different seed sets.



11.2.5 Using Lexicons

Extracting the number of words of the lexicon appearing in the document:

$$f_L = \sum_{w \in L} count(w)$$

Extracting the weighted count of the words appearing in the lexicon

$$f_L = \sum_{w \in L} \theta_w^L count(w)$$

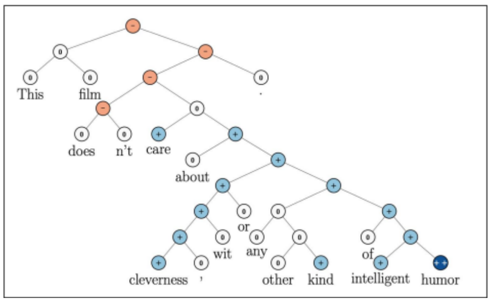
11.3 Entity-Based Sentiment Analysis

To train a system for scoring contextualised word embeddings (Field and Tsvetkov 2019):

1. Get all contextualised word embeddings for a word, compute the average E.
2. Get the VAD scores for the word; train 3 systems to learn a mapping from E to these scores.
3. At prediction time, use the trained systems to get the VAD scores.
4. Now we are able to get the VAD scores for each entity. Average the values for each mention of an entity.

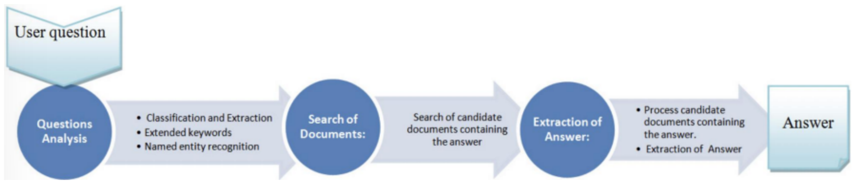
11.4 Sentiment Treebank

- Give a parse tree of a sentence, each node is annotated for Sentiment.
- The sentiment of a node can be studied in terms of the sentiment of the child nodes.



12 Question Answering

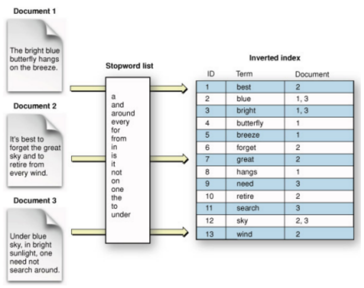
Question answering is the task of automatically providing an answer for a question asked in NL.


User question
Questions Analysis
• Classification and Extraction
• Extended keywords
• Named entity recognition
Search of Documents:
Search of candidate documents containing the answer
Extraction of Answer:
• Process candidate documents containing the answer.
• Extraction of Answer
Answer

The QA subtasks are: identifying the source/context, identifying the answer type, the reading comprehension, the visual QA, the information retrieval and the community answering.

12.1 Search Engines

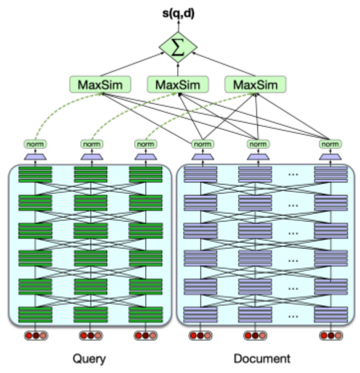
Given a query and a collection of documents/passages, rank the documents/passages with respect to the relevance to the query.
Search engines analyse documents and build an inverted index.
Terms are weighted with tf-idf and documents get scored as the sum of the weights of the matching terms.
Query and document terms match if their word forms are exactly the same. Synonymity is handled using embeddings.


Document 1
The bright blue butterfly hangs on the breeze.
Document 2
It's best to forget the great sky and to retire from every wind.
Document 3
Under blue sky, in bright sunlight, one need not search around.
Stopword list
a
and
around
every
for
from
in
is
it
not
on
one
the
to
under
Inverted index

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

12.1.1 Bi-Encoders

Encoding together (q, d) allows for better representation, but the computation has to be repeated for each (q, d) pair.
If we encode separately the query and the passages we obtain less accurate results but way more efficient
Use a fast method like the standard document scoring through inverted index to retrieve a subset of interesting documents, then apply the most expensive BERT.
ColBERT: separately encode the query and the passage, keeping the representation of each token.
For each token in q, ColBERT finds the most contextually similar token in d, and then sums up these similarities.

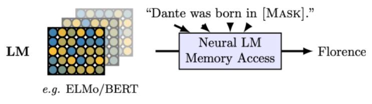


12.2 Machine Learning Comprehension

A machine comprehends a passage of text if, for any question regarding that text that can be answered correctly by a majority of native speakers, that machine can provide a string which those speakers would agree both answers that question, and does not contain information irrelevant to that question.
There are a few corpora for reading comprehension such as CNN/DailyMail, CBT and SQuAD.

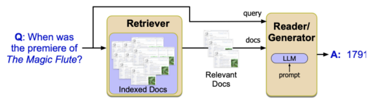
12.3 Language Models as Knowledge Bases

Particularly used for factoid questions (easy to formulate a template answer).
LLM might hallucinate they make up answers that sound reasonable because they are not trained specifically to give correct answers and it might work if the relevant data is public, recent, and it has been used to train the LLM.



12.4 Retrieval-Augmented Generation

In order to avoid the issues of LLM not using recent and private data, RAG is used and relevant informations is first retrieved and then feed to the LLM together with the question and template answer.



- **Retriever:** retrieving relevant passages from a text collection. The retrieved passages are added to the prompt of the LLM.
- **Reader:** augmenting generation with information from the extracted passages. Has a lot of issues:
  - The size of the context window for the LLM determines the number of relevant passages that can be provided to the LLM.
  - If the dense vectors are used to select the passages, relatively short passages should be encoded, otherwise it is complex to keep their semantic.
  - Searching among a huge number of embeddings may not meet the efficiency requirements of the application.

12.4.1 Evaluation

- **F1-Score:** average token overlap between predicted and gold answers.
- **MRR:** is used when QA systems return a short ranked list of answers

$$MRR : \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

- **BLEU:** the idea is to compare the text with a reference one

$$p_n = \frac{\textit{number of n-grams appearing both on gold and hypothesis}}{\textit{number of n-grams appearing in the hypothesis}} \qquad BLEU : \exp\frac{1}{N}\sum_{n=1}^{N}\log p_n$$

- **Rouge-n:** overlap of n-grams between the system and reference summaries.
- **Rouge-L:** longest common subsequence based

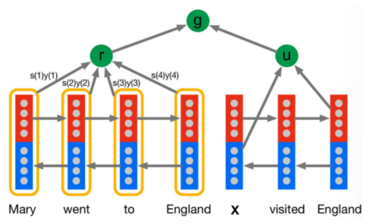
$$R_{ICS} = \frac{LCS(X,Y)}{m}$$

- **BERTScore:** given two sentences, compute the contextual embedding for each token of each sentence, obtaining two lists $x, \hat{x}$. Then every token is matched with the one having largest cosine similarity

$$R_{BERT}(x,\hat{x}) = \frac{\sum_{x_i\in x} idf(x_i)\max_{\hat{x}_j\in\hat{x}} x_i^\top \hat{x}_j}{\sum_{x_i\in x} idf(x_i)}$$

12.5 Neural Models

1. Read (encode) context document and question.
2. Read question first, then attend to context.
3. Use joint representation to generate answer:
   - Predict based on attention map.
   - Generate conditioned on joint representation.
   - Classify over set of candidate answers.

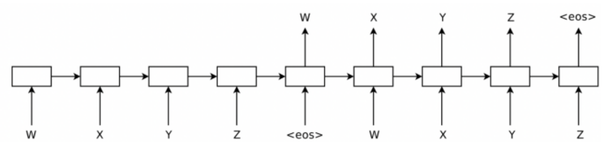


13 Sentence Embeddings

13.1 Autoencoder

Dai et al. used an encoder-decoder RNN-based architecture to reproduce the input sentence as output:

- Use a RNN network as an encoder to read in an input sequence into a hidden state, which is the input to a decoder recurrent network that predicts the output sequence.
- The goal is to produce a compressed representation that is able to minimize the information loss.



The use of unsupervised data allows to improve the performance on several tasks.

13.2 Skip-Thought Vectors

- LSTM encoder-decoder model trained on the BookCorpus dataset.
- The goal is to reproduce the previous and the next sentence.
- s_{i-1} = "I got back home"; s_i = "I could see the cat on the steps", s_{i+1} ="This was strange".
- The encoder builds a representation for one sentence; a pair of sentences is represented as :

$$[|u - v|; u \cdot v]$$

13.2.1 Out of Vocabulary (OOV)

The idea is to learn a mapping that transfer word representations from one model to another.
Learn a linear mapping from a word in word2vec space to a word in the encoder's vocabulary space.
Train on all words that are shared between vocabularies.

13.3 SimCSE (Simple Contrastive Learning of Sentence Embeddings)

Contrastive learning: given a batch of examples, push together semantically related examples, by minimising L. Make sure the embeddings are uniformly distributed or push apart unrelated sentences.

13.3.1 Unsupervised SimCSE

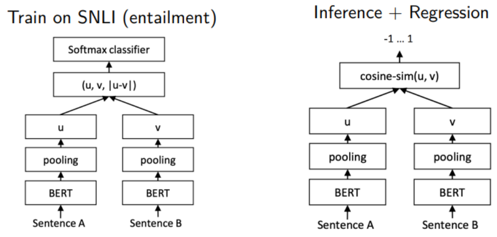
The positive pairs are selected by passing the same input sentence to the pre-trained encoder twice applying independently sampled dropout masks. Obtain two embeddings as "positive pairs".

13.3.2 Supervised SimCSE

Use sentences in Entailment as positive labels and use contradicting sentences as hard negatives.

13.4 Sentence-BERT (SBERT)

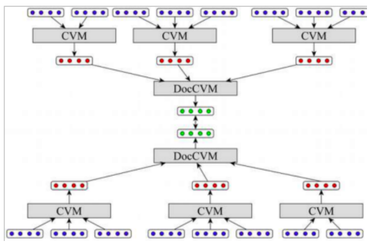
SBERT uses a Siamese network (shared weights).
Pooling: mean of all output vectors.
Uses a triplet loss (left) or a MSE (right).



14 Document Embeddings

For documents, we typically need to capture aboutness.
With BOW approaches we can express the document as a weighted combination of the word vectors or a combination of the n-grams.

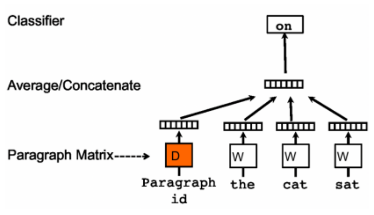
14.1 Compositional Approach



Hermann et al. proposed a multi-level approach for document representation where:

- Parallel documents in different languages are matched by composing the representations of the sentences.
- CVM stands for any compositional function for sentence encoding.

14.2 Paragraph Vectors



Paragraph vector is an extension of a language model with an additional element to keep info on the full context.
Is used to give context to the next-word prediction task.
Such information becomes the representation of the document

15 Coreference Resolution

Grouping spans of text (mentions) that refer to a single underlying entry or event (referent).
Task: the input is a raw text, and systems must detect mentions and then link them into clusters.

15.1 Evaluation

B^3**:** given mention i, let R be the reference (gold) set that includes i, and H the hypothesis (prediction) set that includes i

$$Precision = \frac{H \cap R}{H} \quad Recall = \frac{H \cap R}{R} \quad F_1 = 2\frac{Precision \times Recall}{Precision + Recall}$$

15.2 Algorithms for Coreference Resolution

15.2.1 Mention Identification

Heuristic: Consider all NP (noun phrases) and names entries (sometimes all n-grams). We can have Nested NP with the same head, Numerical entities, Non referential pronouns. Any false negative error cannot be recovered in the next phases (so usually generous). It is possible to consider all possible spans (n-gramms) and perform mention identification and mention clustering at the same time. **Non-Referential Pronouns:** Pronouns may not refer to entities, may have generic referents and may not refer to anything.
Referential vs Non-referential Pronouns: we can check the distributional statistics of similar pronouns in the same context.

15.2.2 Mention Clustering Based Models

Each pair of mentions is scored independently and a clustering algorithm clusters mentions together. Is the fastest approach but may lead to incoherent results.
Mention Pair Models: annotate pairs of mentions with a binary label (1 / -1)

- **Positive examples:** for each mention in position j, find the most recent coreference mention i and assign to that pair the label 1.
- **Negative examples:** for each entity k, $i < k < j$, assign a label -1 to the pair.
- Any supervised classification algorithm can now be applied as long as we define a representation for the mentions.
- Proper nouns often corefer with other proper nouns. The idea is to match the syntactic head words of the reference with the referent: for sequences of proper nouns, the head word will be the final token, but it doesn't work all the time.
- Nominals are any NP that is not a pronoun or a proper noun.
- **Mentions Features:** POS, number of tokens, Lexical features, Morphosyntactic features.
- **Mention Pair Features:** Combination of mention features, Distance, String matching, Binary features, Gazetteers, Lexical semantics.
- Once the classifier is trained, it could act as a distance function for a clustering algorithm. For each mention i in a document, the classifier considers each of the prior mentions.
- **Closest-First clustering:** the classifier is run right to left and the first antecedent with probability $> .5$ is linked to i.
- **Best-First clustering:** the classifier is run on all $i-1$ antecedents and the most probable preceding mention is chosen as the antecedent for i.
- The transitive closure of the pairwise relation is taken as the cluster.

15.2.3 Entity Clustering Based Models

The whole group of mentions is scored together. In general is more accurate but is the slowest.

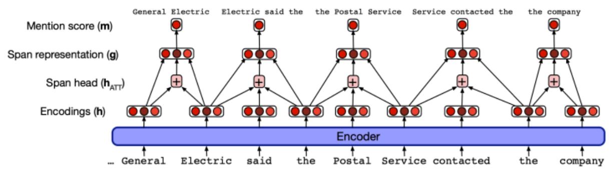
$$\max_z \sum_{e=1} \psi_E(\{i : z_i = e\})$$

where the scoring function applied to all mentions i that are assigned to entity e.
IN order to reduce the computational complexity, incremental search strategies are implemented:

- When a mention is first encountered in a text, it is matched with all clusters already formed; it is added only if it is compatible with all the elements of the cluster.
- It is possible to make mistakes early on: beam search is used to maintain a set of solutions.

End to End Models:

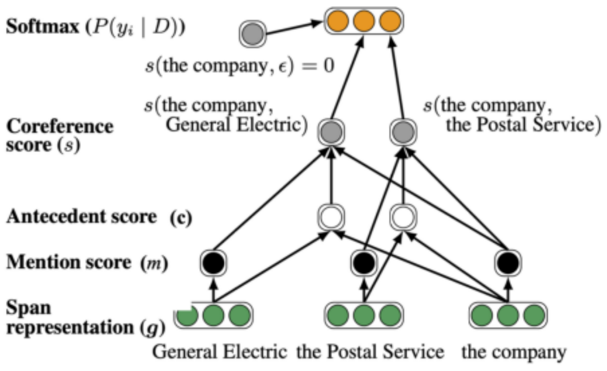
- Neural e2e-coref algorithms: consider all possible spans.
- For each pair of spans *(i,j)*, compute *s(i,j)* (higher if they corefer).
- $s(i,j) = f(m(i), m(j), c(i,j))$
  - $m(x)$ →span x is a mention.
  - $c(i,j) \rightarrow j$ is the antecedent of i.
- A span is represented by the embeddings of: the first word, the last word, the embedding of the most important words in the span.
- To compute *m()* and *c()* we need to represent a span. Use an encoder (BERT) to generate (contextual) embeddings; the attention one is the sum of the attention-weighted embeddings of all the words in the span. Then concatenate the representation of the three embeddings.



- A span i combined the representation of the 3 tokens

  $$g_i = [h_{START(i)}; h_{END(i)}, h_{ATT(i)}]$$

  Given all the scores an antecedent is selected, then a transitive closure of all mentions gives a final clustering.


Softmax ($P(y_i \mid D)$)
s(the company, ϵ) = 0
s(the company, General Electric)
s(the company, the Postal Service)
Coreference score (s)
Antecedent score (c)
Mention score (m)
Span representation (g)
General Electric
the Postal Service
the company

- $c(i,j)$ input: representations of the spans and a combined representation of the two.
- Normalize the scores for each antecedent of span i: $\frac{\exp(s(i,j))}{\sum_j \exp(s(i,j))}$

16 Semantic Role Labelling

Semantic roles, express the role that arguments of a predicate take in the event. Help generalize over different surface realizations of predicate arguments.
SRL act as a shallow meaning representation that can let us make simple inferences that aren't possible from the pure surface string of words, or event from the parse tree.
A SRL system is required to:

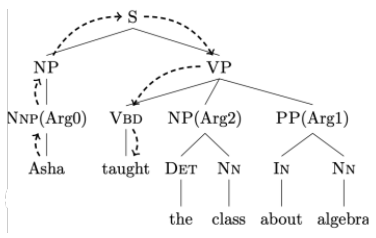
1. Identify all predicates.
2. Specify the spans of text that fill each role

The SRL roles do not necessarily coincide with the grammatical ones.
Can we define generic categories (roles) associated with the predicates? Yes, with tools such as:

- **VerbNet:** categories are organised in a hierarchy and verbs are grouped by meaning.
- **PropBank:** allows for shallow semantic analysis.
- **FrameNet:** aims to generalise across verbs. Semantic frames are descriptions of situations or events, organised in a complex structure.

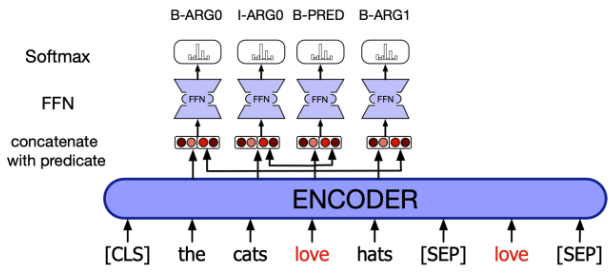
16.1 SRL as Classification

Features: syntactic path from the argument to the predicate (can also be used the dependency path).



16.2 SRL Neural Approaches

We use BIO encoding and the input is *the sentence + [SEP] + the predicate.*


B-ARG0
I-ARG0
B-PRED
B-ARG1
Softmax
FFN
concatenate with predicate
ENCODER
[CLS]
the
cats
love
hats
[SEP]
love
[SEP]