A Little Survey on Traditional and Current Word and Sentence Embeddings

Simón Roca Sotelo (sroca@ing.uc3m.es)

June 26, 2018

Table of Contents

1. History
2. Word2Vec
3. FastText
4. ELMo
5. Introducing Sentence Embeddings
6. Implementing Word2Vec

Looking for contextual features

- *First clarification* – The term Word Embeddings comes from Deep Learning Community. However, other researchers may refer to them as: Distributional Semantic Models, Distributed Representations, Semantic Vector Space, Word Space...

Looking for contextual features

- *First clarification* – The term Word Embeddings comes from Deep Learning Community. However, other researchers may refer to them as: Distributional Semantic Models, Distributed Representations, Semantic Vector Space, Word Space...

Core idea

Distributional hypothesis – *You shall know a word by the company it keeps* (Firth, J. R. 1957:11)

Looking for contextual features

- *First clarification* – The term Word Embeddings comes from Deep Learning Community. However, other researchers may refer to them as: Distributional Semantic Models, Distributed Representations, Semantic Vector Space, Word Space...

Core idea

Distributional hypothesis – *You shall know a word by the company it keeps* (Firth, J. R. 1957:11)

- i.e., features capturing semantic information of words can arise from its context.

Approaches I

Automatic methods for learning contextual features were proposed in the 90's.

1. **Information Retrieval**: Latent Semantic Indexing/Analysis (LSI/LSA)

Approaches I

Automatic methods for learning contextual features were proposed in the 90's.

1. **Information Retrieval**: Latent Semantic Indexing/Analysis (LSI/LSA)
2. **Computational Linguistics**: Hyperspace Analogue to Language (HAL)

Approaches I

Automatic methods for learning contextual features were proposed in the 90's.

1. **Information Retrieval**: Latent Semantic Indexing/Analysis (LSI/LSA)
2. **Computational Linguistics**: Hyperspace Analogue to Language (HAL)
3. **Neural Networks**: Self Organized Maps (SOM) and Simple Recurrent Networks (SRN)

Approaches II

Later developments may be seen as refinements of the previous models.

1. **Information Retrieval**: Probabilistic LSA and Latent Dirichlet Allocation (LDA)
2. **Computational Linguistics**: Random Indexing and BEAGLE
3. **Neural Networks**: CNN and Autoencoders

Approaches II

Later developments may be seen as refinements of the previous models.

1. **Information Retrieval**: Probabilistic LSA and Latent Dirichlet Allocation (LDA)
2. **Computational Linguistics**: Random Indexing and BEAGLE
3. **Neural Networks**: CNN and Autoencoders

Intuition

- pLSA and LDA use documents as context. The later two lines use words as context.
- Typically the first captures semantic relatedness (boat-water), the later captures semantic similarity (boat-ship).

Skip-gram model

$$\max_{\Theta} \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c} \log p(w_{t+j} | w_t) \tag{1}$$

- The aim is to maximize the averaged likelihood of predicted context words, where c is the context window size.

Skip-gram model

$$\max_{\Theta} \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c} \log p(w_{t+j} | w_t) \tag{1}$$

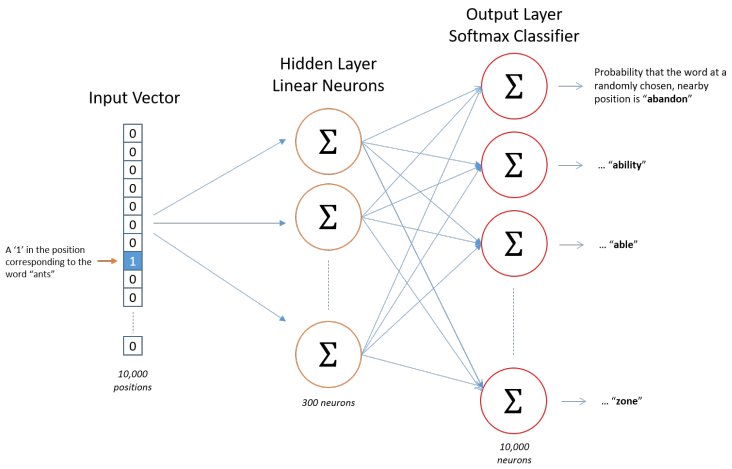
- The aim is to maximize the averaged likelihood of predicted context words, where c is the context window size.
- A neural network may be constructed. Input: a word, output: probabilities on each word that may appear in the input word context.

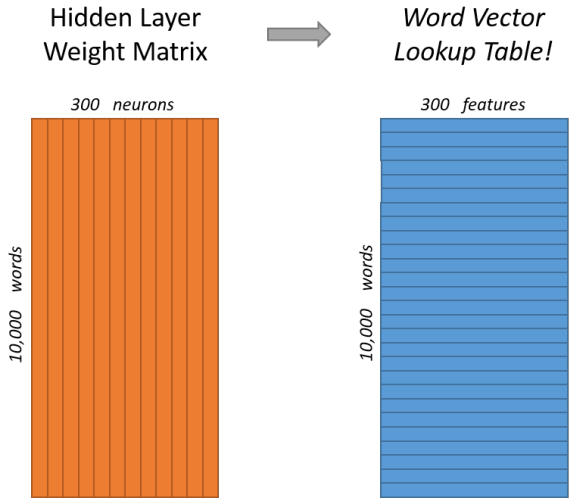
Skip-gram model

$$\max_{\Theta} \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c} \log p(w_{t+j} | w_t) \tag{1}$$

- The aim is to maximize the averaged likelihood of predicted context words, where c is the context window size.
- A neural network may be constructed. Input: a word, output: probabilities on each word that may appear in the input word context.
- Hidden layer won't apply activation. Output will be converted into a probability with softmax function.

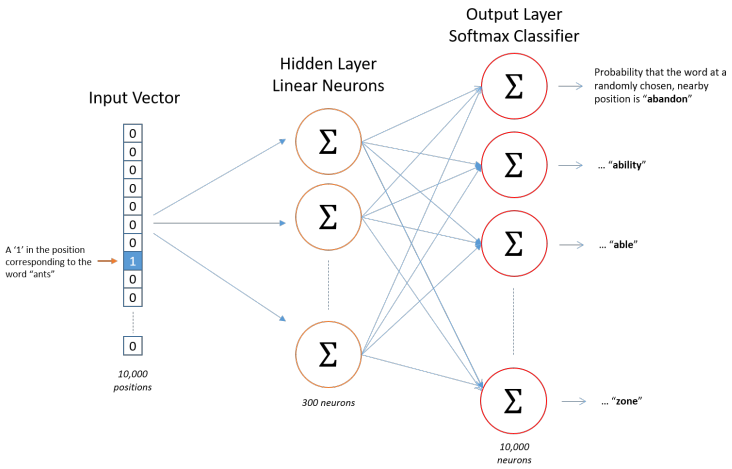
Skip-gram model

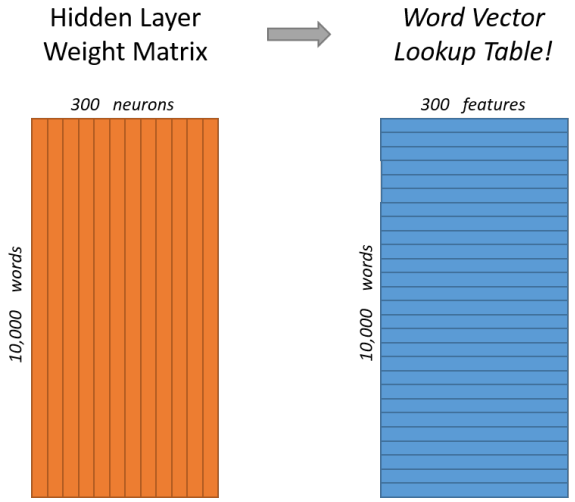




- One-hot vectors as inputs select one row in the input weight matrix – **the word contextual vector!**

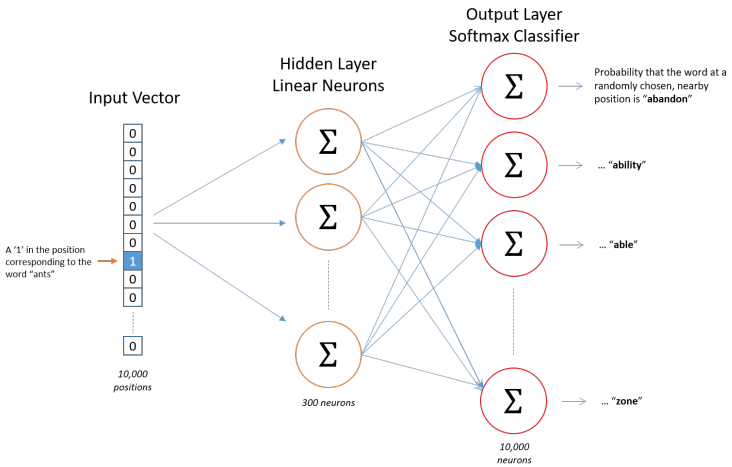
Skip-gram model

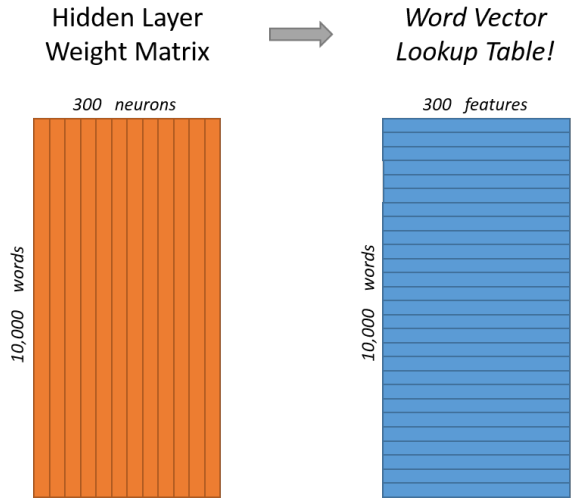




- One-hot vectors as inputs select one row in the input weight matrix – **the word contextual vector!**
- Training examples are pairs of words (input, random context word, 'skipping' some combinations).

Skip-gram model





- One-hot vectors as inputs select one row in the input weight matrix – **the word contextual vector!**
- Training examples are pairs of words (input, random context word, 'skipping' some combinations).

Key point

Similar words are expected to have similar contexts (similar probabilities placed on same context words). This necessarily implies that their vectors should be similar as well.

Tricks for efficiency

- Considering we have 10 000 words and 300 features, we are learning 3M weights for each layer.
- Some tricks should be done to improve efficiency:

Tricks for efficiency

- Considering we have 10 000 words and 300 features, we are learning 3M weights for each layer.
- Some tricks should be done to improve efficiency:
  - Subsampling frequent words.
  - Hierarchical Softmax.
  - Negative Sampling.

Subsampling frequent words

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) * \frac{0.001}{z(w_i)} \quad (2)$$

where:

- $P(w_i)$ is the probability of discarding an instance of a given word.

Subsampling frequent words

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) * \frac{0.001}{z(w_i)} \quad (2)$$

where:

- $P(w_i)$ is the probability of discarding an instance of a given word.
- $z(w_i)$ is the fraction of words in vocabulary corresponding to a specific term.

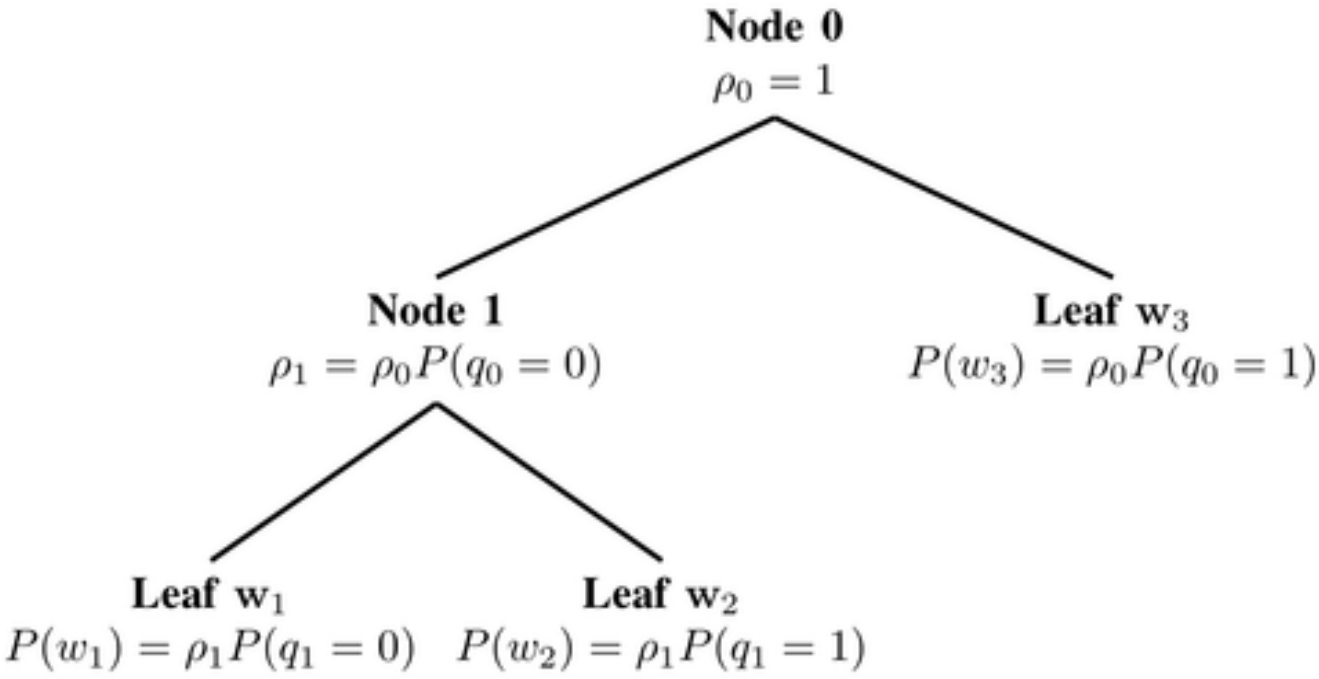
Subsampling frequent words

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) * \frac{0.001}{z(w_i)} \quad (2)$$

where:

- $P(w_i)$ is the probability of discarding an instance of a given word.
- $z(w_i)$ is the fraction of words in vocabulary corresponding to a specific term.
- 0.001 is a parameter that can be modified to subsample more words (or less).

Hierarchical Softmax



By using a binary tree structure for the softmax layer, we enforce words output to follow paths in which each level probability is already normalized, resulting in a faster training.

Negative Sampling

- It works as an approximation to Noise Contrastive Estimation, which approximates the loss of the softmax function.
- Instead of updating all weights each time, it focuses on k 'negative' samples (words out of context, choosen at random).
- The task becomes distinguish the target word from noisy words.

Negative Sampling

- It works as an approximation to Noise Contrastive Estimation, which approximates the loss of the softmax function.
- Instead of updating all weights each time, it focuses on k 'negative' samples (words out of context, choosen at random).
- The task becomes distinguish the target word from noisy words.

$$J_\Theta = \log \sigma(v'^{\mathsf{T}}_{w_O} v_{w_I}) + \sum_{i=1}^{k} \mathop{\mathbb{E}}_{w_i \sim P_n(w)} [\log \sigma(-v'^{\mathsf{T}}_{w_i} v_{w_I})] \tag{3}$$

$$P_n(w_i) = \frac{z(w_i)^{3/4}}{\sum_{j=0}^{n} z(w_j)^{3/4}} \tag{4}$$

FastText: improving Word2Vec

- Word2Vec has at least one problem: there is no vector associated to unseen words.
- FastText provides a solution: learning vectors on subword segment using n-grams.

Splitting words in FastText

For instance, 'where':

Splitting words in FastText

For instance, 'where':

1. Beginning and ending of the word is made explicit: <where>

Splitting words in FastText

For instance, 'where':

1. Beginning and ending of the word is made explicit: <where>
2. Get the n-grams: <wh, whe, her, ere, re>

Splitting words in FastText

For instance, 'where':

1. Beginning and ending of the word is made explicit: <where>
2. Get the n-grams: <wh, whe, her, ere, re>
3. Consider the whole word <where>

Splitting words in FastText

For instance, 'where':

1. Beginning and ending of the word is made explicit: <where>
2. Get the n-grams: <wh, whe, her, ere, re>
3. Consider the whole word <where>

- Finally, the word vector is represented as the sum of the vector representation of its n-grams.

Splitting words in FastText

For instance, 'where':

1. Beginning and ending of the word is made explicit: <where>
2. Get the n-grams: <wh, whe, her, ere, re>
3. Consider the whole word <where>

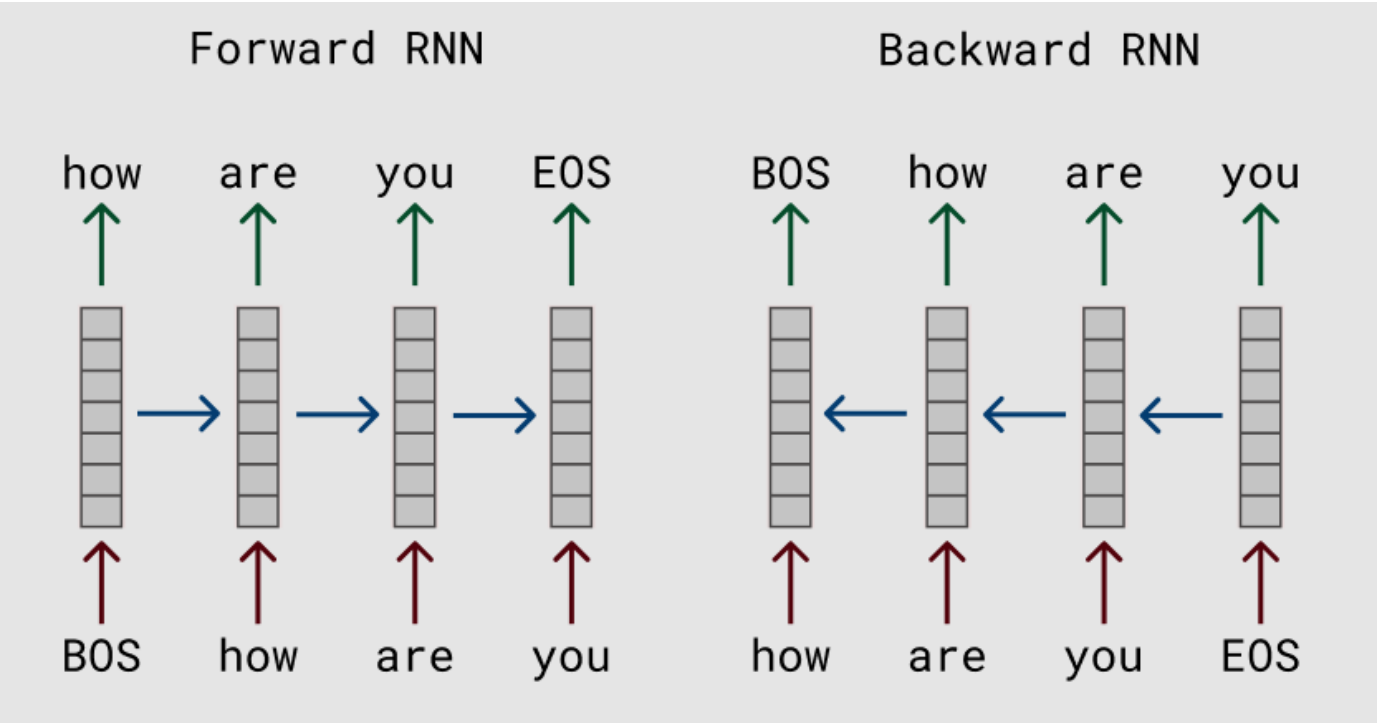
- Finally, the word vector is represented as the sum of the vector representation of its n-grams.
- n-grams are hashed to a fixed number K to control memory costs.

What is ELMo?

- Embeddings for Language Models (ELMo) claims to learn features that are function of the entire corpus itself.
- Vectors are computed on top of two-layer bidirectional language models (biLMs) with character convolution.
- By 2018, improved the SoA of several tasks, let's see how.

ELMo: Bidirectional LM

Given a sequence of N tokens, $(t_1, t_2, ..., t_N)$, we can model probabilities 'looking forward' or 'looking backwards':



ELMo: Bidirectional LM

ELMo 'ties' two LSTMs (forward and backward), by using same word vectors and Softmax Layers.

$2L + 1$ representations

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

where $\mathbf{h}_{k,0}^{LM}$ is the token layer and $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$, for each biLSTM layer.

Then, ELMo vectors for each task are obtained from each layer output:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

What information is codifying ELMo

- Inputs x_k^{LM} can be word vectors, or even char-gram vectors (CNN over chars). They call it context-independent.

What information is codifying ELMo

- Inputs x_k^{LM} can be word vectors, or even char-gram vectors (CNN over chars). They call it context-independent.
- It better captures multiple meanings of a word with respect to its context.

What information is codifying ELMo

- Inputs x_k^{LM} can be word vectors, or even char-gram vectors (CNN over chars). They call it context-independent.
- It better captures multiple meanings of a word with respect to its context.
- Each layer is expected to learn different textual information, and all the information is explored.

ELMo can be use to update pre-trained BiLMs, by freezing the weights and concatenating the ELMo vectors at different points of the network (typically at the input).

ELMo and SOA

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

Introducing sentence embeddings

Can we learn features characterizing a full sentence/document?

	Words Embed.	**Sentences Embed.**
Strong baselines	FastText	Bag-of-Words
State-of the-art	ELMo	**Unsupervised** Uses unannotated or weakly-annotated dataset: Skip-Thoughts, Quick-Thoughts, DiscSent, Google's dialog input-output **Supervised** Uses annotated dataset: InferSent, Machine translation **Multi-task learning** Uses **several** annotated or unannotated datasets: MILA/MSR's General Purpose Sent., Google's Universal Sentence Enc. recent trend

Sentence Embeddings I: Unsupervised

- **Bag of words (baseline)**: for a text segment, just averaging word vectors of the words of a sentence works fairly well for some tasks.

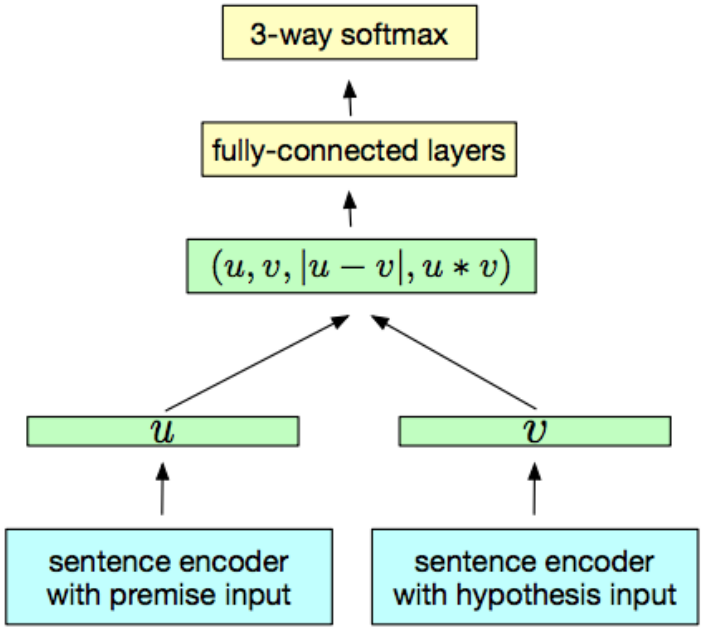
Sentence Embeddings I: Unsupervised

- **Bag of words (baseline)**: for a text segment, just averaging word vectors of the words of a sentence works fairly well for some tasks.
- **Skip-thoughts vectors**: similar to skip-gram word2vec, 'given a sentence, predict surrounding ones'. RNN-based encoder-decoder. They learn a linear regressor to map from w2v vectors to skip-thought vectors. This allows to introduce new words learned by external embeddings.

Sentence Embeddings I: Unsupervised

- **Bag of words (baseline)**: for a text segment, just averaging word vectors of the words of a sentence works fairly well for some tasks.
- **Skip-thoughts vectors**: similar to skip-gram word2vec, 'given a sentence, predict surrounding ones'. RNN-based encoder-decoder. They learn a linear regressor to map from w2v vectors to skip-thought vectors. This allows to introduce new words learned by external embeddings.
- **Quick-thoughts vectors**: an order of magnitude faster than the previous method. Decoder now becomes a classifier for candidate output sentences.

Sentence Embeddings II: Supervised



Infersent starts with a huge annotated corpus, with sentences which are entailed, contradicted or neutral. Then, two candidate sentences are encoded and different operations on its vectors are an input to a deep network which tries to classify its relation. BiLSTM is a typical choice for encoders, where each vector $\mathbf{h_t}$ is used. Learned features are suitable for Recognizing Textual Entailment (RTE), generalizing well for many other tasks.

Sentence Embeddings II: Supervised

name	N	task	C	examples
MR	11k	sentiment (movies)	2	"Too slow for a younger crowd , too shallow for an older one." (neg)
CR	4k	product reviews	2	"We tried it out christmas night and it worked great ." (pos)
SUBJ	10k	subjectivity/objectivity	2	"A movie that doesn't aim too high , but doesn't need to." (subj)
MPQA	11k	opinion polarity	2	"don't want"; "would like to tell"; (neg, pos)
TREC	6k	question-type	6	"What are the twin cities ?" (LOC:city)
SST	70k	sentiment (movies)	2	"Audrey Tautou has a knack for picking roles that magnify her [..]" (pos)

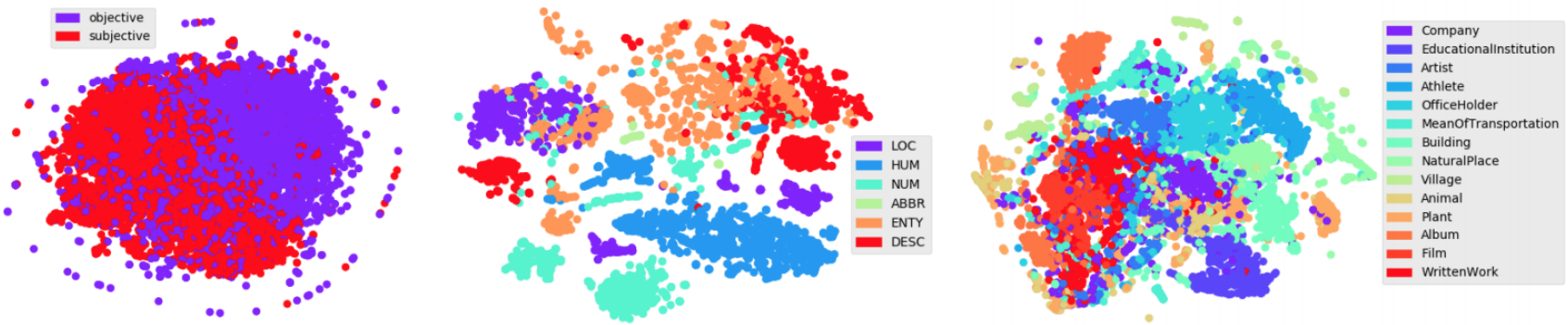
Table 1: **Classification tasks**. C is the number of class and N is the number of samples.

name	task	N	premise	hypothesis	label
SNLI	NLI	560k	"Two women are embracing while holding to go packages."	"Two woman are holding packages."	entailment
SICK-E	NLI	10k	A man is typing on a machine used for stenography	The man isn't operating a stenograph	contradiction
SICK-R	STS	10k	"A man is singing a song and playing the guitar"	"A man is opening a package that contains headphones"	1.6
STS14	STS	4.5k	"Liquid ammonia leak kills 15 in Shanghai"	"Liquid ammonia leak kills at least 15 in Shanghai"	4.6

Table 2: **Natural Language Inference and Semantic Textual Similarity tasks**. NLI labels are contradiction, neutral and entailment. STS labels are scores between 0 and 5.

Sentence Embeddings III: Universal

Universal Sentence Encoder uses a bidirectional GRU (Gated Recurrent Unit) encoder and a unidirectional GRU encoder. Several tasks can be casted as sentence-to-sentence. Different tasks datasets are used to learn a comprised set of features. Decoders are task-specific, while there's only one encoder.



Sentence Embeddings III: Universal

Universal Sentence Encoder (Google) presents two models in which the user can tune some parameters trading-off regarding accuracy and complexity.The **Transformer** encoding model is the most successful (and expensive) in many tasks, using Attention. On the other hand, a **Deep Averaging Network** is used, averaging word-level embeddings at its input, reducing compute time. Both encoders are designed to learn very general features.

Sentence Embeddings III: Universal

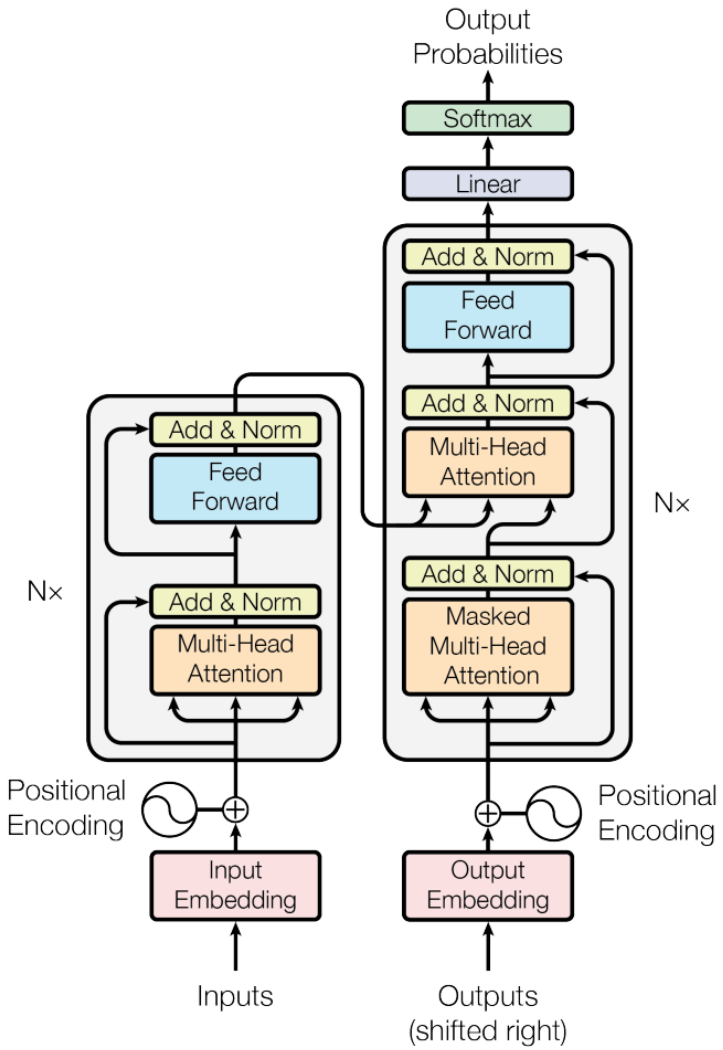
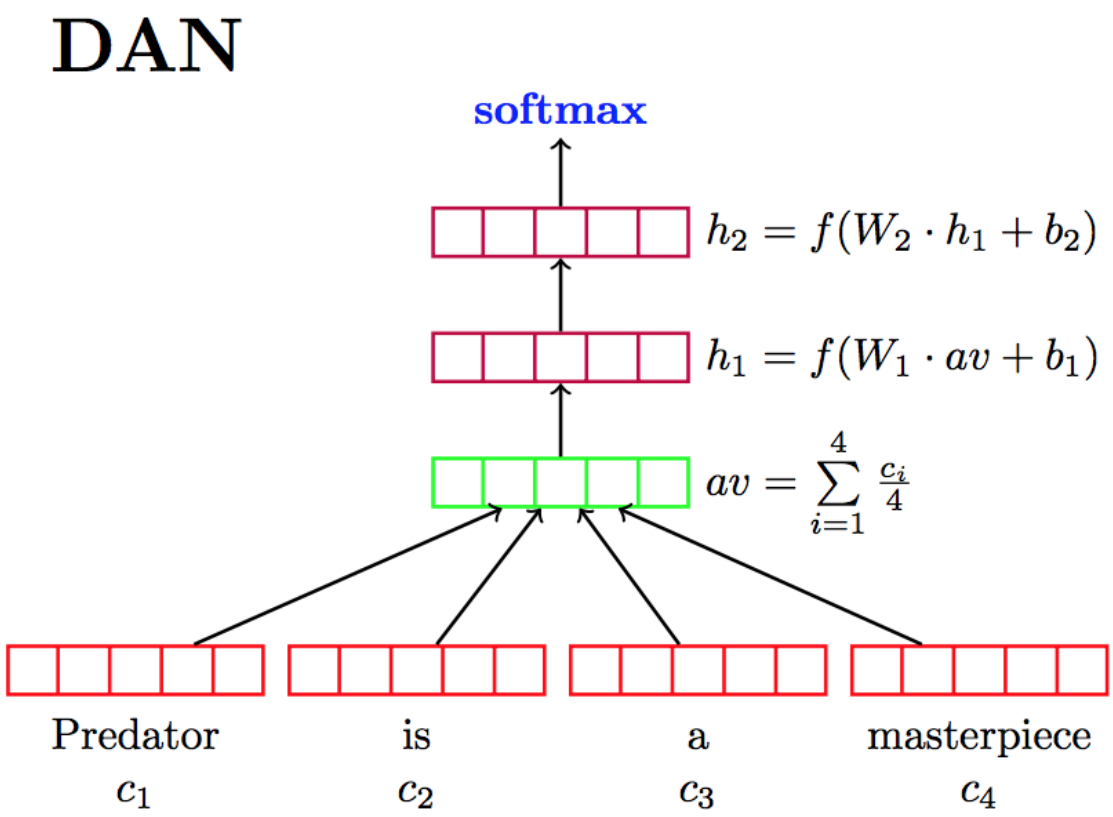


Figure 1: The Transformer - model architecture.

Sentence Embeddings III: Universal



Warning

Word2Vec and other models implementations are public, with wrappers for several languages. They typically provide pre-trained vectors on huge datasets, or methods to train your own embeddings probably in the most efficient way.

We're taking a look into a word2vec tutorial in Python3 with TensorFlow. Concretely, we will:

- generate batches using parameters we've learned.
- create a Tensor Model graph.
- Compare running time for Softmax and Negative Sampling.

- The Bidirectional Language Model - https://medium.com/@plusepsilon/the-bidirectional-language-model-1f3961d1fb27
- The Current Best of Universal Word Embeddings and Sentence Embeddings - https://medium.com/huggingface/universal-word-sentence-embeddings-ce48ddc8fc3a
- Word2Vec Tutorials (I and II). Chris McCormick - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- Word2Vec Tutorial Part I: The Skip Gram Model. Alex Minaar (gradients for Word2Vec).

- A Brief History of Word Embeddings - https://www.gavagai.se/blog/2015/09/30/a-brief-history-of-word-embeddings/
- On Word Embeddings. - http://ruder.io/word-embeddings-1/
- Word2Vec word embedding tutorial in Python and TensorFlow - http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/
- Fast Text and Skip Gram (Fast Text in Keras) - http://debajyotidatta.github.io/nlp/deep/learning/word-embeddings/2016/09/28/fast-text-and-skip-gram/