NEURAL MODELS FOR WORD EMBEDDING

Prof. Eduardo Bezerra
(CEFET/RJ)
ebezerra@cefet-rj.br

Credit where credit is due

- The original papers from Mikolov et al.
- CS 224D: Deep Learning for NLP
  - https://cs224d.stanford.edu/lecture_notes/notes1.pdf
- McCormick, C. (2016). Word2Vec Tutorial.
  - http://www.mccormickml.com
- Understanding word vectors (Python notebook)
  - https://gist.github.com/aparrish/2f562e3737544cf29aaf1af30362f469

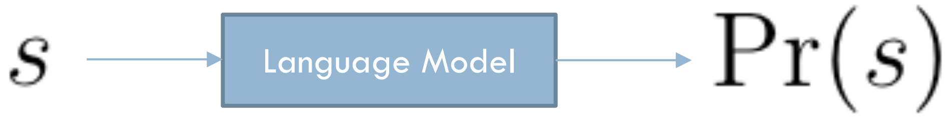
Summary

- Language Models
- Word2vec
  - Skip-gram

Language Models

Language Model

- Definition: A model that assigns a probability to a sequence of **tokens** (e.g., words or characters).
- A good language model gives...
  - ...(syntactically and semantically) valid sentences a high probability.
  - ...low probability to nonsense.

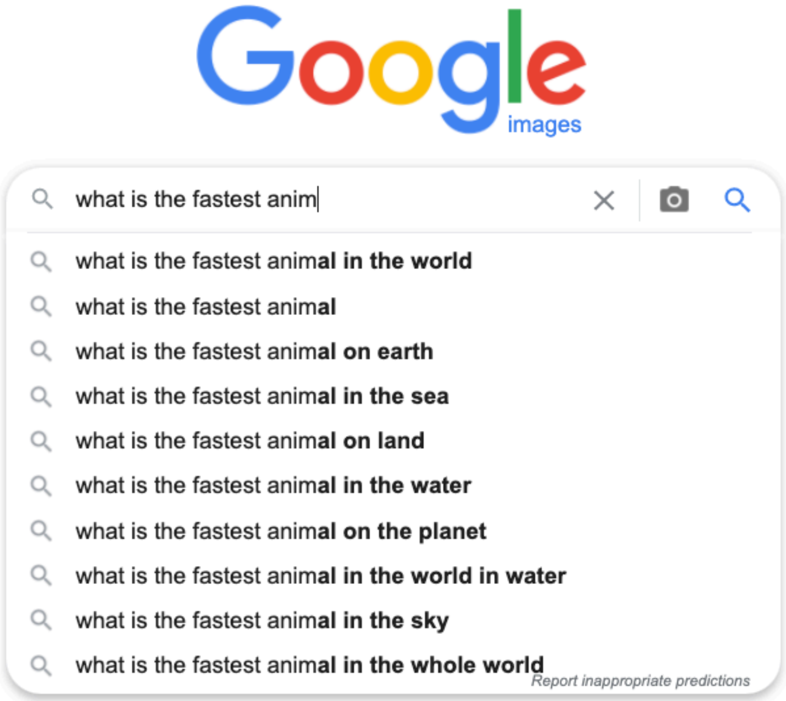


Language Model - applications

- NLP-based applications use language models for a variety of tasks:
  - audio to text conversion,
  - speech recognition,
  - sentiment analysis,
  - summarization,
  - spell correction,
  - etc.

Language Model (example application)

Example: query completion



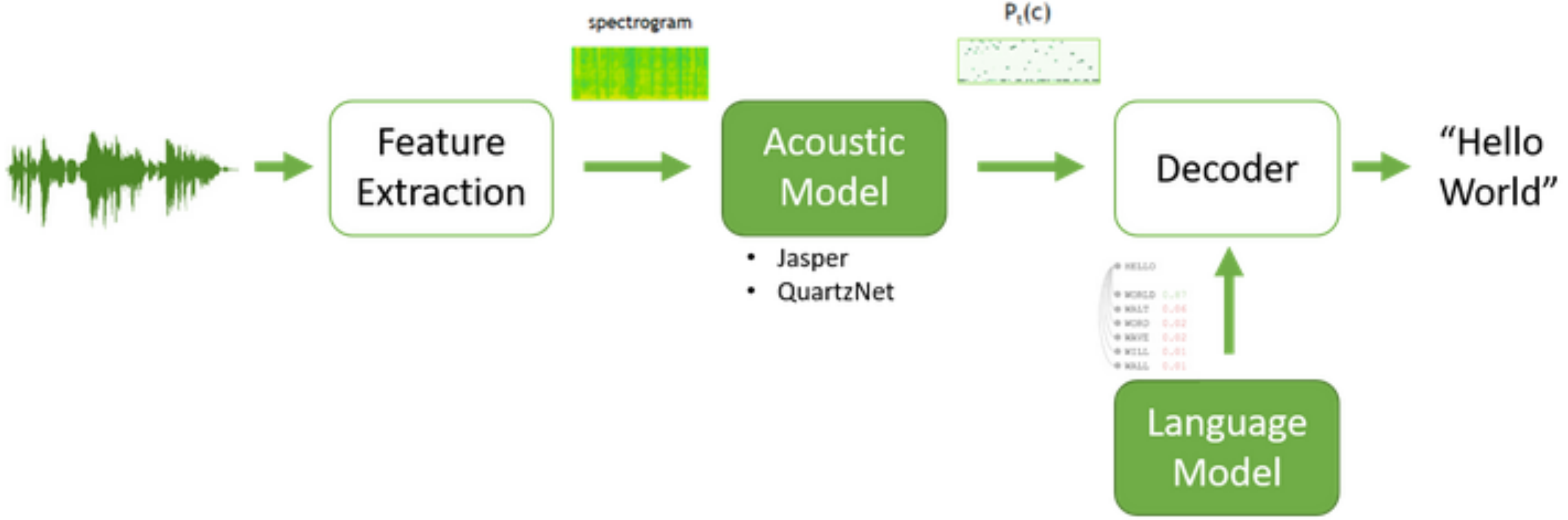
what is the fastest anim

what is the fastest anim**al in the world**
what is the fastest anim**al**
what is the fastest anim**al on earth**
what is the fastest anim**al in the sea**
what is the fastest anim**al on land**
what is the fastest anim**al in the water**
what is the fastest anim**al on the planet**
what is the fastest anim**al in the world in water**
what is the fastest anim**al in the sky**
what is the fastest anim**al in the whole world**

Report inappropriate predictions

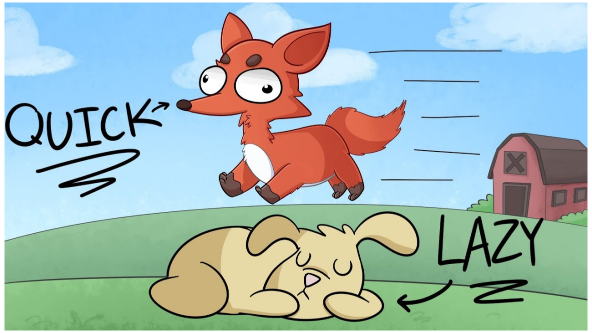
Language Model (example application)

Example: speech recognition

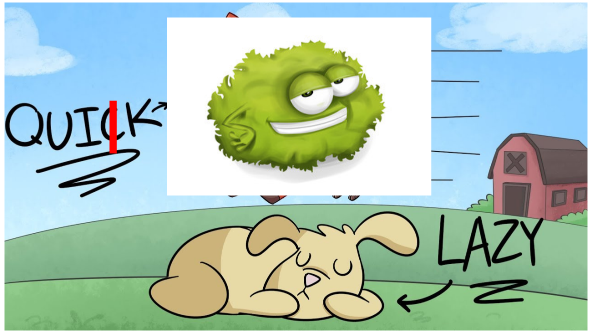


https://developer.nvidia.com/blog/how-to-build-domain-specific-automatic-speech-recognition-models-on-gpus/

Language Models (example)



$s_1 =$ The quick brown fox jumps over the lazy dog.



$s_2 =$ The quik brown lettuce over jumps the lazy dog

$$\boxed{\Pr(s_1) > \Pr(s_2)}$$

Language Models

- A (statistical) LM produces a probability to an input sequence by applying the <u>chain rule</u> from Probability Theory.

$$\begin{aligned}\Pr(w_1, w_2, \cdots, w_n) = {} & \Pr(w_1)\times \\ & \Pr(w_2 \mid w_1)\times \\ & \Pr(w_3 \mid w_1, w_2)\times \\ & \times \cdots \times \\ & \Pr(w_n \mid w_1, w_2, \cdots, w_{n-1})\end{aligned}$$

The chain rule tells us how to compute the joint probability of a sequence by using the conditional probabilities of each component word given previous words.

Language Models

$s_1 =$ The quick brown fox jumps over the lazy dog.

$$\begin{aligned}\Pr(\text{The}, \text{quick}, \cdots, \text{dog}) = {} & \Pr(\text{The}) \times \\ & \Pr(\text{quick} \mid \text{The}) \times \\ & \Pr(\text{brown} \mid \text{The}, \text{quick}) \times \\ & \times \cdots \times \\ & \Pr(\text{dog} \mid \text{The}, \text{quick}, \cdots, \text{lazy})\end{aligned}$$

n-gram models

- A challenge in building LMs from a corpus is data sparsity: most possible word sequences are not observed in training.
- One solution is to build a particular type a LM, an n-gram model.

n-grams (examples)

An n-gram is a contiguous sequence of n tokens (e.g., words).

- unigrams:

  (the), (quick), (brown), (fox), (jumped), (over), (the), (lazy), (dog)

- bigrams:

  (the quick), (quick brown), (brown fox), (fox jumped), (jumped over), (over the), (the lazy), (lazy dog)

- trigrams:

  (the quick brown), (quick brown fox), (brown fox jumped), (fox jumped over), (jumped over the), (over the lazy), (the lazy dog)

n-grams (example n=2)

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

n-gram models – Markov assumption

- n-gram models make the *Markov assumption*: “the probability of a word only depends on the previous n-1 words”.
  - This assumption drastically simplifies the estimation of conditional probabilities from data.

n-gram models – Markov assumption

$$\begin{aligned}\Pr(w_1, w_2, \cdots, w_n) = &\Pr(w_1)\times\\ &\Pr(w_2 \mid w_1)\times\\ &\Pr(w_3 \mid w_1, w_2)\times\\ &\times\cdots\times\\ &\Pr(w_n \mid w_1, w_2, \cdots, w_{n-1})\end{aligned}$$

Unigram model (aka *bag of words* model)

$$\Pr(w_1, w_2, \cdots, w_n) = \prod_{i=1}^{n} \Pr(w_i)$$

Bigram model

$$\Pr(w_1, w_2, \cdots, w_n) = \prod_{i=2}^{n} \Pr(w_i \mid w_{i-1})$$

n-gram models – Markov assumption

$$s_1 = \text{The quick brown fox jumps over the lazy dog.}$$

$$\begin{aligned}\Pr(\text{The}, \text{quick}, \cdots, \text{dog}) = & \Pr(\text{The})\times \\ & \Pr(\text{quick} \mid \text{The})\times \\ & \Pr(\text{brown} \mid \text{The}, \text{quick})\times \\ & \times \cdots \times \\ & \Pr(\text{dog} \mid \text{The}, \text{quick}, \cdots, \text{lazy})\end{aligned}$$

$$\begin{aligned}\Pr(\text{The}, \text{quick}, \cdots, \text{dog}) = & \Pr(\text{The})\times \\ & \Pr(\text{quick} \mid \text{The})\times \\ & \Pr(\text{brown} \mid \text{quick})\times \\ & \cdots \times \\ & \Pr(\text{dog} \mid \text{lazy})\end{aligned}$$

But, how to learn these conditional probabilities from data?

Learning n-gram models

- The simplest way to build an n-gram model from a corpus is through *maximum likelihood estimation* (count occurrences)

$$\boxed{\Pr(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) \approx \frac{\#(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\#(w_{i-(n-1)}, \ldots, w_{i-1})}}$$

$$\boxed{\Pr(\text{‘dog’} \mid \text{‘lazy’}) \approx \frac{\#(\text{‘lazy dog’})}{\#(\text{‘lazy’})}}$$

Example

statistical models → neural models

- Statistical LMs (such as n-gram models) were pervasive in the 80s and 90s.
- But in the 2000s, several neural language models started to appear.
- A **neural language model** is a language model built using Artificial Neural Networks.

Embedding models

- Embedding models are a kind of neural language model.
- They aim to learn a **continuous (vector) representation** for each token in a corpus.
- Once learnt, the continuous representations can be used in <u>downstream</u> machine learning tasks: classification, clustering, etc.

Embedding models

- Two types:
  - **word embedding**
  - **contextual embedding**

The kid dreamt of becoming an astronaut.
The child dreamt of becoming an astronaut.

I left the left door of the barn open.

Embedding models

- word2vec[1], GloVe[1]
- SkipThoughts
- Paragraph2Vec
- Doc2Vec
- FastText[1]
- ELMo[2], BERT[2], GPT[2]

AND MANY MORE!!

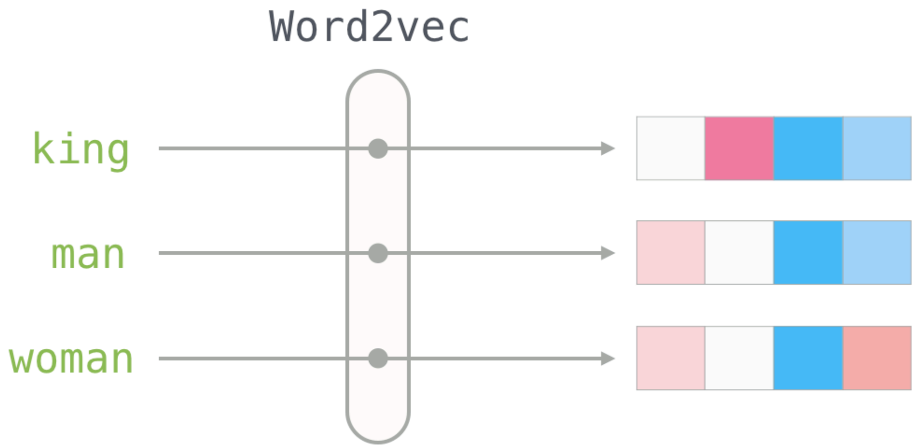
Currently, the distributional hypothesis through vector embeddings models generated by ANNs is used pervasively in NLP.

1) word embeddings (aka **global embeddings)**
2) contextual embeddings

word2vec

word2vec

- word2vec is a set of model architectures and optimizations to learn word embeddings from large corpora.



http://jalammar.github.io/illustrated-word2vec/

The word2vec trick

- In word2vec, a single hidden layer neural network is trained to perform a certain “fake” task.
  - But the resulting model is not actually used!
- Instead, the goal is to learn the weights of the hidden layer.
  - ***these weights are the “word vectors”***

The fake tasks

- **Skip-gram: predicting surrounding <u>context words</u> given a <u>center word</u>.**
- CBOW: predicting a center word from the surrounding context.

word2vec/skip-gram

Skip-gram

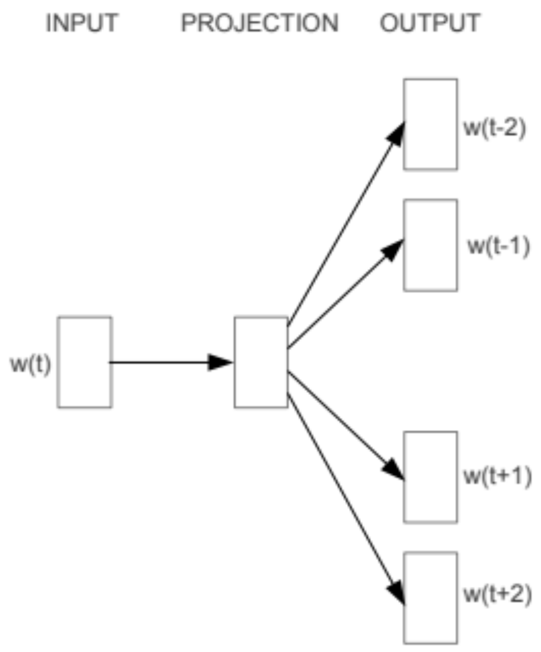
- The task: train a NN to predict whether a word is related to another word **w** given as input.
  - The network is trained to produce a probability distribution (one entry for every word in the vocabulary) words of being <u>nearby</u> **w**.
  - “nearby”: there is a "window size" (typical value: 5)

Skip-gram architecture

- Input layer size: |V|
- A single hidden layer, fully connected NN is used.
  - Neurons in the hidden layer are all linear.
  - Hidden layer size: set to the desired dimensionality of the resulting word vectors.
- Output layer size: |V|

V is the vocabulary set

Skip-gram architecture



Skip-gram

https://arxiv.org/pdf/1301.3781.pdf

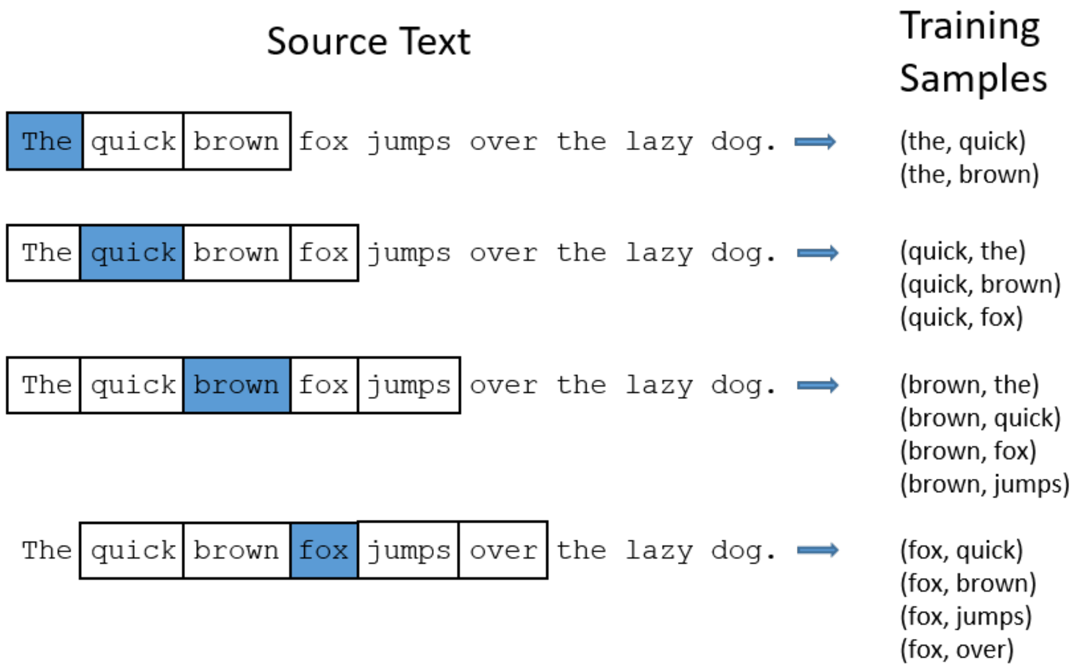
Skip-gram – output layer

- Probabilities produced in the output layer relate to how likely it is to find each vocabulary word nearby the input word **w**.
- For example, say **w** = “Africa”
  - output probabilities should be much higher for related words (e.g. “lion”, “zebra”) than for unrelated words (e.g. “bear”, “kangaroo”).

Training examples (word pairs)

- The training examples will be word pairs taken from the input corpus.
- For example, suppose the NN gets many more examples of (“dog”, “cat”) than (“dog”, “tiger”).
  - After training, if the NN is given the word “dog”, it will output a much higher probability for “cat” (or “pet”) than it will for “tiger”.

Positive training examples (word pairs)



"The quick brown fox jumps over the lazy dog."
window size = 2.

The word highlighted in blue is the input word **w**.

Negative training examples

- Negative training examples are built for an input word w by forming pairs (w, w_{out}), in which w_{out} is some word in the vocabulary not inside the context of w.

One hot encoding

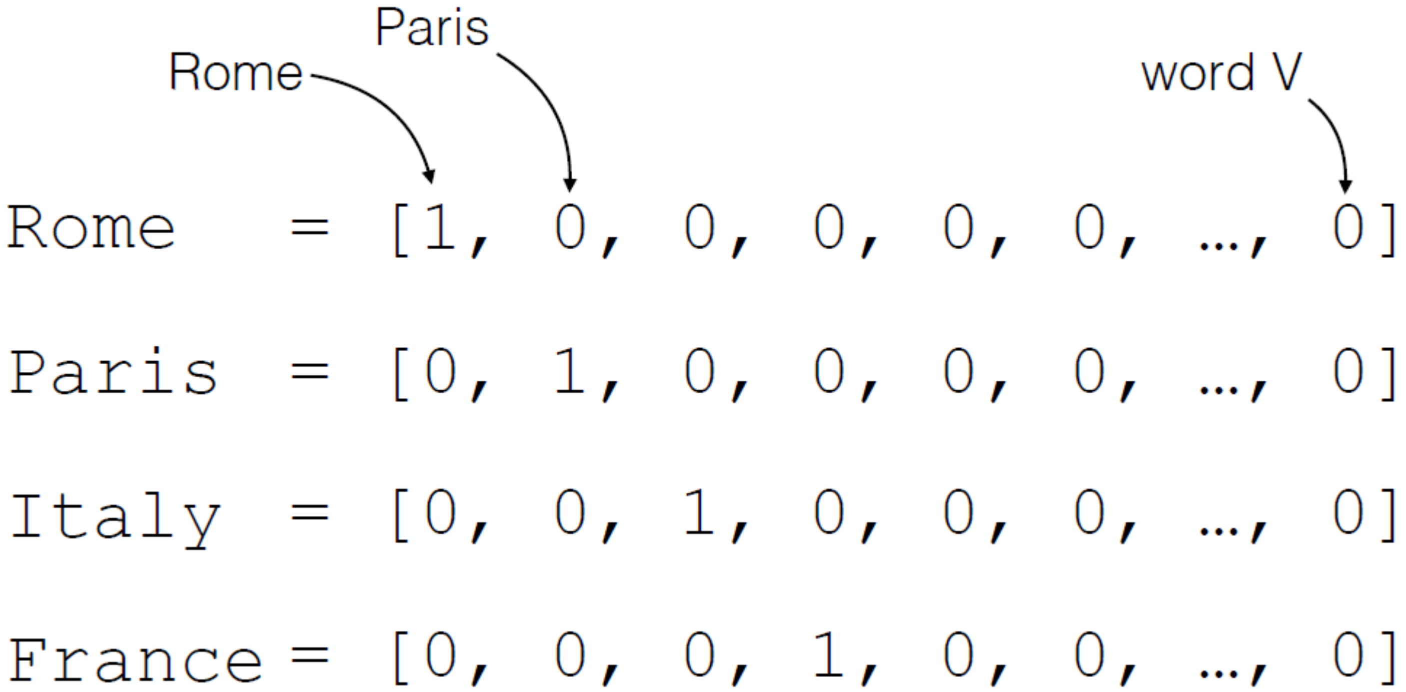
- During training, we cannot provide the NN with the text of the words.
  - Some vector representation for each word in V is needed → one hot encoding.

One hot encoding

```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

Credits: Marco Bonzaninin

One hot encoding



```
Rome   = [1, 0, 0, 0, 0, 0, ..., 0]
Paris  = [0, 1, 0, 0, 0, 0, ..., 0]
Italy  = [0, 0, 1, 0, 0, 0, ..., 0]
France = [0, 0, 0, 1, 0, 0, ..., 0]
```

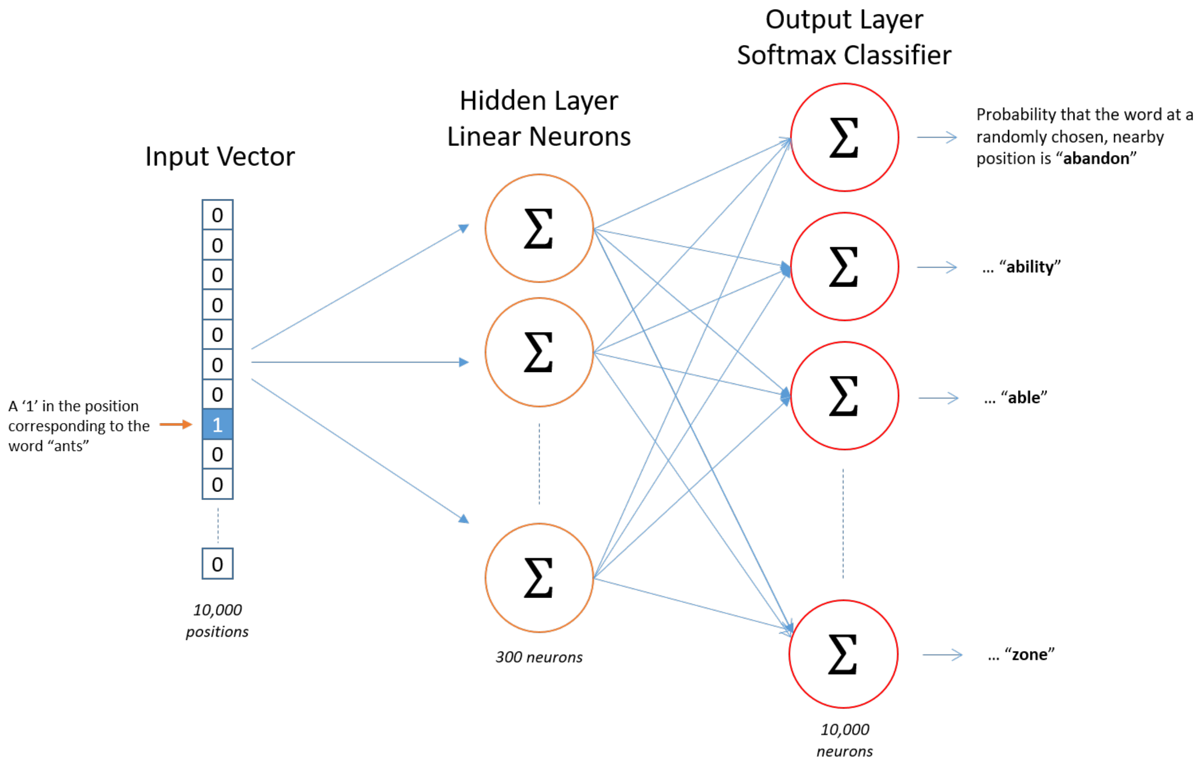
Credits: Marco Bonzaninin

One hot encoding

$$\begin{array}{lcl}
\text{Rome} & = & \overbrace{[1,\ 0,\ 0,\ 0,\ 0,\ 0,\ \ldots,\ 0]}^{V\ =\ \text{vocabulary size (huge)}} \\
\text{Paris} & = & [0,\ 1,\ 0,\ 0,\ 0,\ 0,\ \ldots,\ 0] \\
\text{Italy} & = & [0,\ 0,\ 1,\ 0,\ 0,\ 0,\ \ldots,\ 0] \\
\text{France} & = & [0,\ 0,\ 0,\ 1,\ 0,\ 0,\ \ldots,\ 0]
\end{array}$$

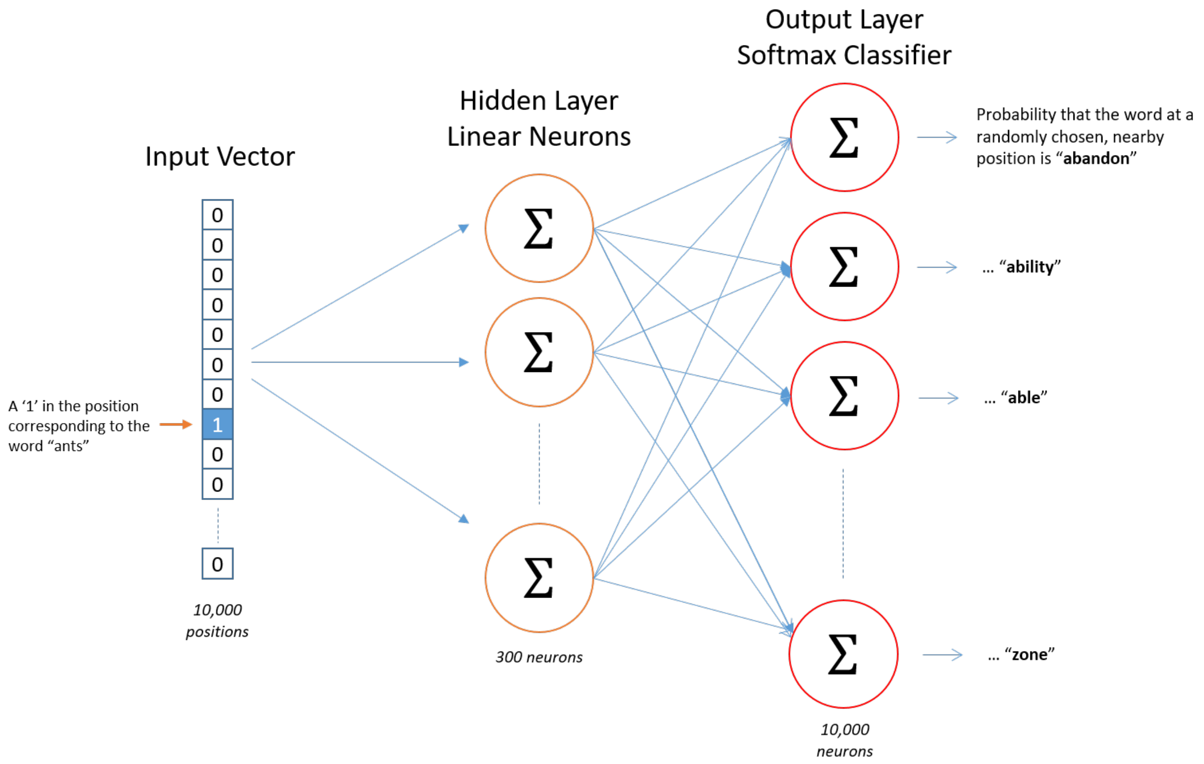
Credits: Marco Bonzaninin

Skip-gram NN architecture



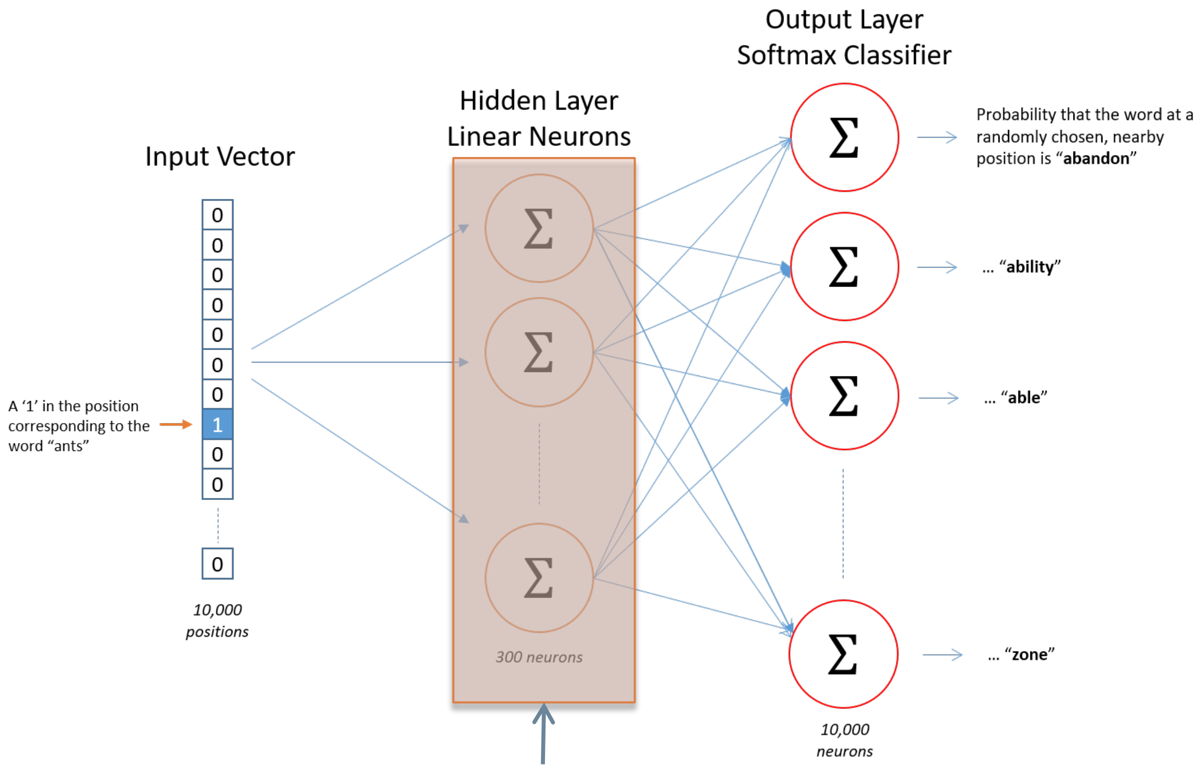
"Linear neurons" means there is no activation function on the hidden layer neurons, …

Skip-gram NN architecture



…, but the output neurons use softmax.

Skip-gram NN architecture



The amount of neurons in the hidden layer (a hyperparameter) determines de size of the embedding.

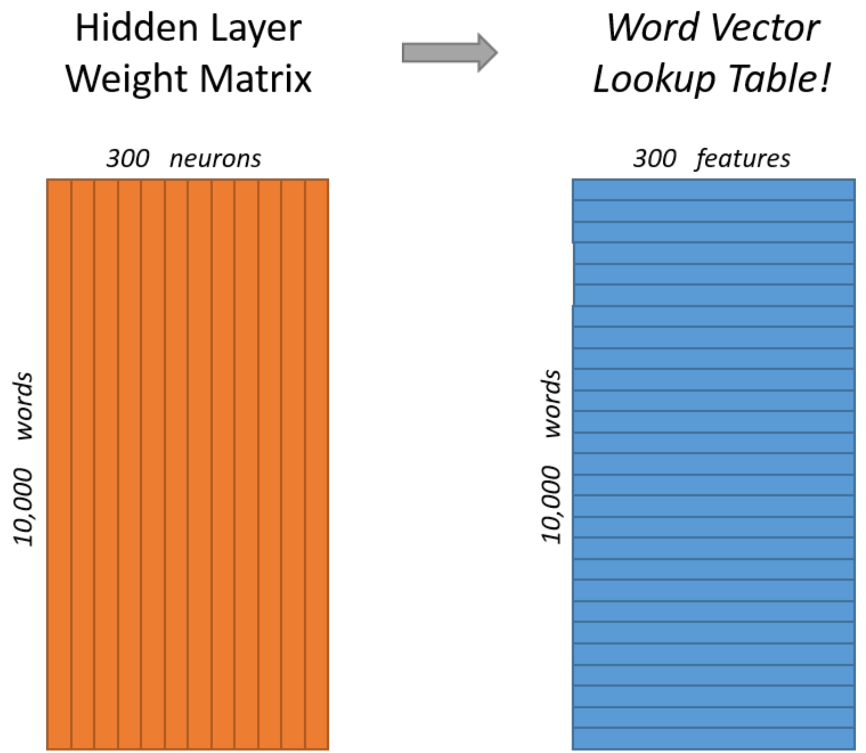
The hidden layer

- Multiplying a 1 x n one-hot vector **v** by an n x d matrix **W** (the hidden layer) will effectively *select* the row of **W** corresponding to the "1" position in **v**.

$$[0 \quad 0 \quad 0 \quad \boxed{1} \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10 \quad 12 \quad 19} \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

So, the output of the hidden layer is just the "word vector" for the input word → the hidden layer plays the role of a **lookup table**.

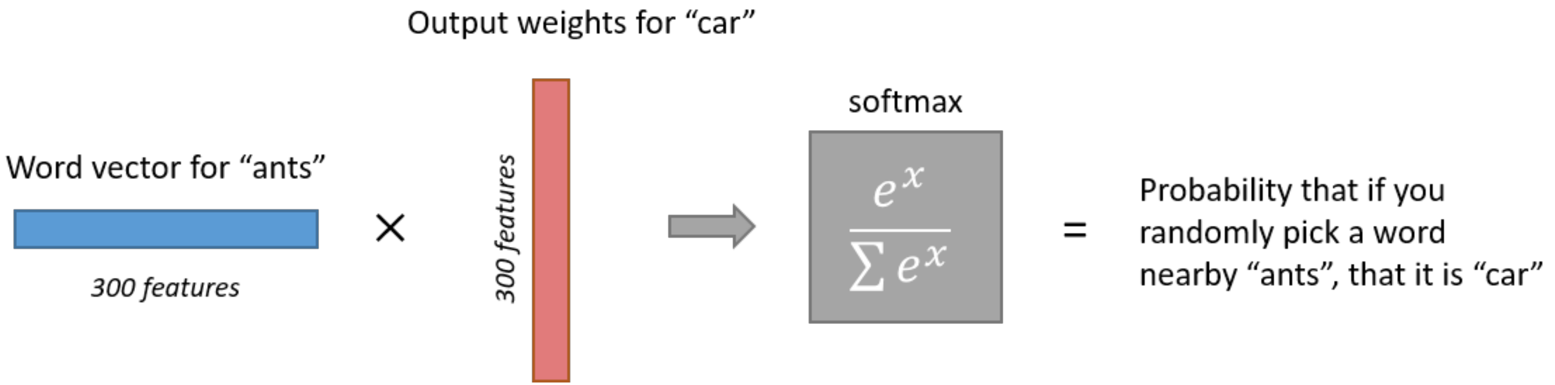
The hidden layer



The output of the hidden layer is just the "word vector" for the input word.
W is the matrix we really want!

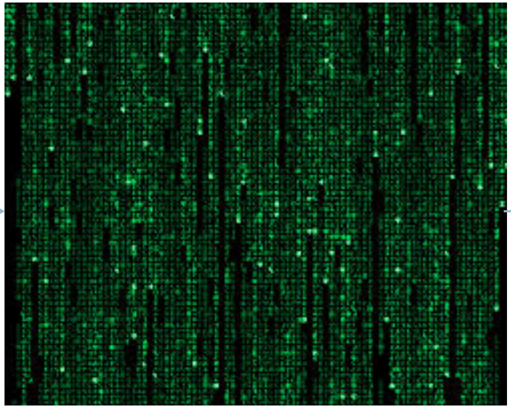
The output layer

- The 1 x d word vector (coming from the hidden layer) then gets fed to the output layer.
- The output layer has |V| neurons and is a softmax regression classifier.



word2vec

$\mathbf{v}_{\text{onehot}} \rightarrow$



$\rightarrow \mathbf{v}_{\text{embed}}$

$\mathbf{W}$

$$\text{e.g., } V = 50000, D = 300$$

$$\mathbf{W} \in \Re^{300 \times 50000}$$

$$\mathbf{v}_{\text{onehot}} \in \Re^{50000 \times 1}$$

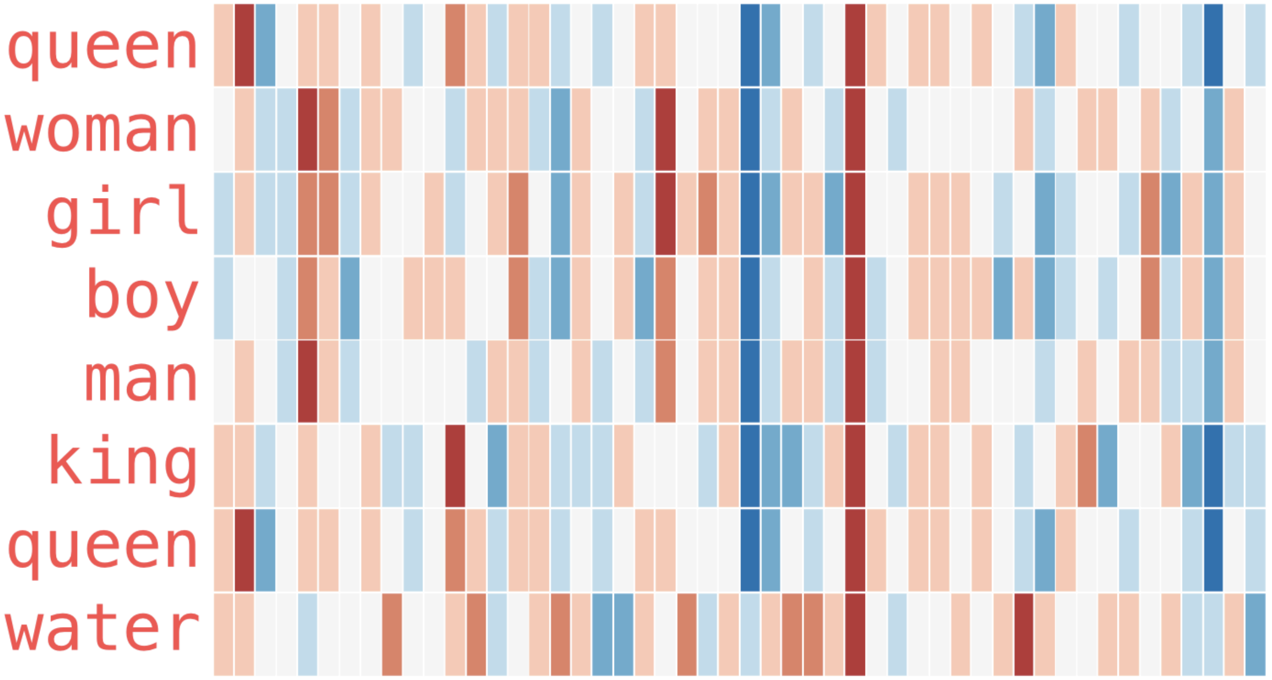
$$\mathbf{v}_{\text{embed}} = \mathbf{W} \times \mathbf{v}_{\text{onehot}}$$

$$\mathbf{v}_{\text{embed}} \in \Re^{300 \times 1}$$

word2vec

- word2vec captures context similarity:
  - If words w_i and w_k have similar contexts, then the model needs to output very similar results for them.
    - One way for the network to do this is to make the word vectors for w_i and w_k very similar.
  - So, if two words have similar contexts, the network is motivated to learn similar word vectors for them.

word2vec



Credits: http://jalammar.github.io/illustrated-word2vec/

Skip-gram captures context similarity

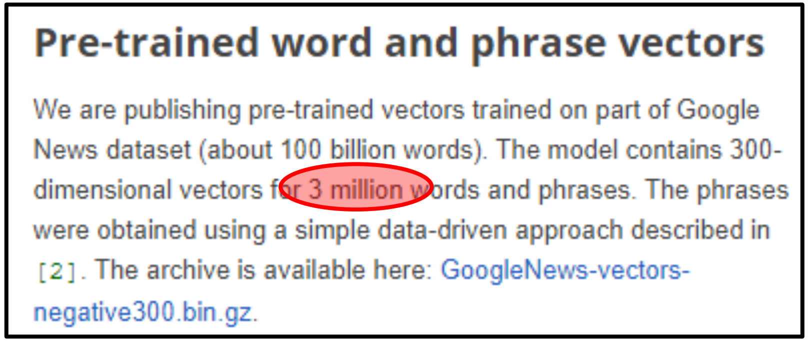
- If two words w_i and w_k have similar "contexts" (i.e., similar words are likely to appear around them), then the model needs to output very similar results for them.
  - One way for the network to do this is if *the word vectors for* w_i and w_k *are similar*.
- So, if two words have similar contexts, our network is motivated to learn similar word vectors for them.

Final Remarks

Take away notes

- SOTA results in <u>most</u> NLP is currently neural-based.
- Neural-based NLP is recent but relies on older ideas.
- Attention mechanism is a novel and very promising idea.

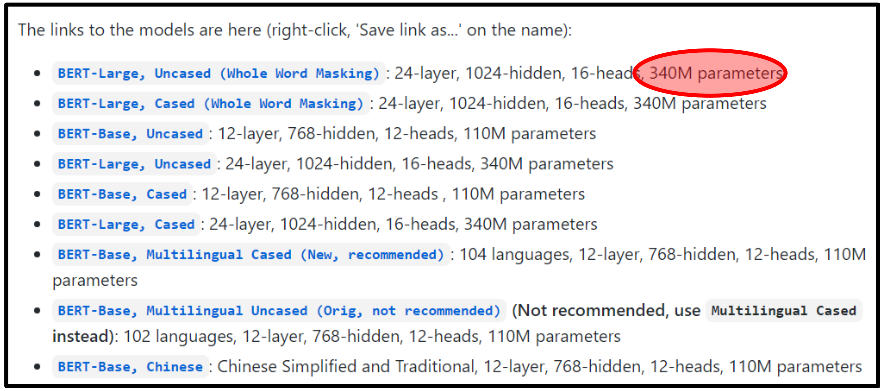
Pretrained models



Pre-trained word and phrase vectors

We are publishing pre-trained vectors trained on part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. The phrases were obtained using a simple data-driven approach described in [2]. The archive is available here: GoogleNews-vectors-negative300.bin.gz.

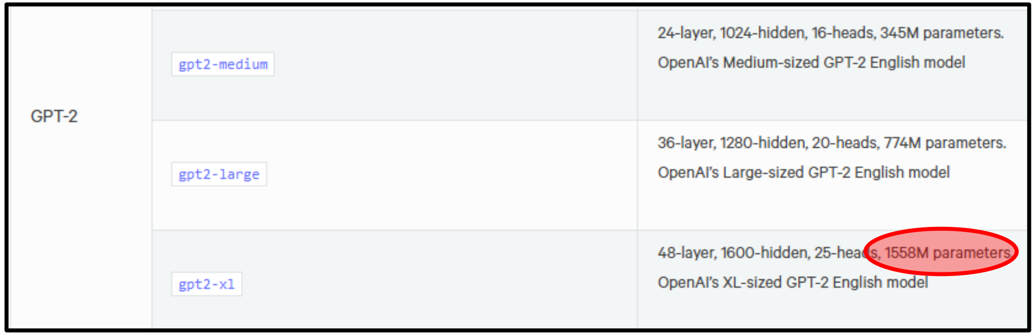
https://code.google.com/archive/p/word2vec/



The links to the models are here (right-click, 'Save link as...' on the name):

- `BERT-Large, Uncased (Whole Word Masking)`: 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Large, Cased (Whole Word Masking)`: 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Uncased`: 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Large, Uncased`: 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Cased`: 12-layer, 768-hidden, 12-heads , 110M parameters
- `BERT-Large, Cased`: 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Multilingual Cased (New, recommended)`: 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Multilingual Uncased (Orig, not recommended)` (Not recommended, use `Multilingual Cased` instead): 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Chinese`: Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

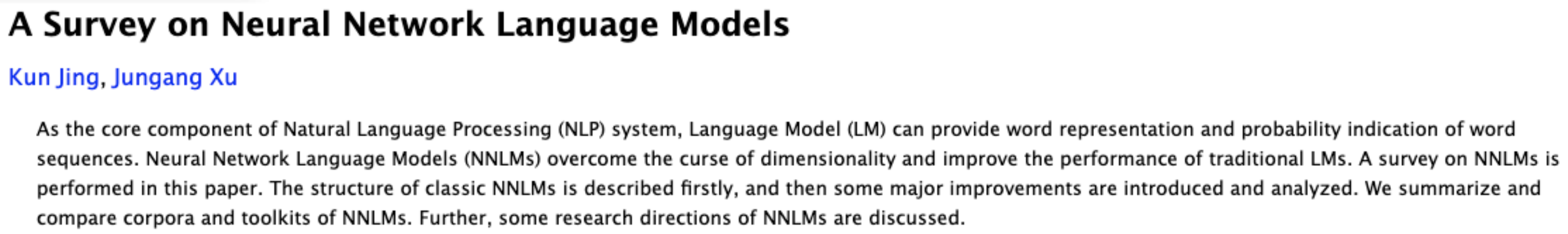
https://github.com/google-research/bert



GPT-2	`gpt2-medium`	24-layer, 1024-hidden, 16-heads, 345M parameters. OpenAI's Medium-sized GPT-2 English model
	`gpt2-large`	36-layer, 1280-hidden, 20-heads, 774M parameters. OpenAI's Large-sized GPT-2 English model
	`gpt2-xl`	48-layer, 1600-hidden, 25-heads, 1558M parameters. OpenAI's XL-sized GPT-2 English model

To learn more – neural language models

Learn More



A Survey on Neural Network Language Models

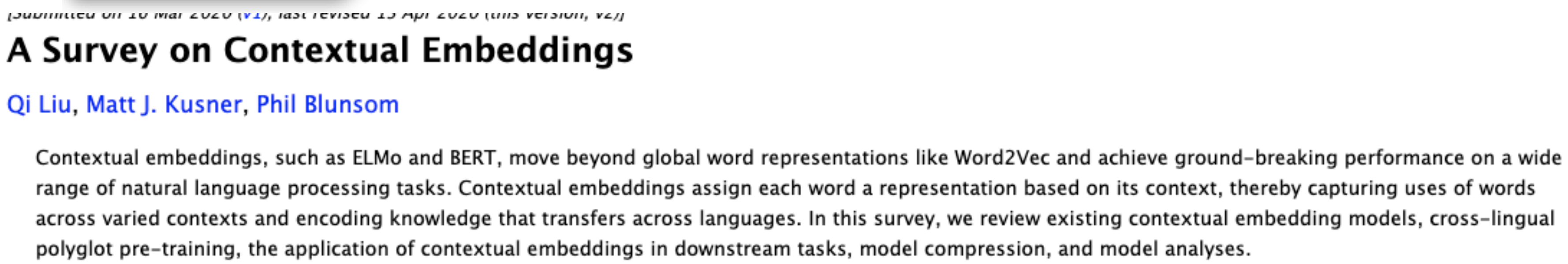
Kun Jing, Jungang Xu

As the core component of Natural Language Processing (NLP) system, Language Model (LM) can provide word representation and probability indication of word sequences. Neural Network Language Models (NNLMs) overcome the curse of dimensionality and improve the performance of traditional LMs. A survey on NNLMs is performed in this paper. The structure of classic NNLMs is described firstly, and then some major improvements are introduced and analyzed. We summarize and compare corpora and toolkits of NNLMs. Further, some research directions of NNLMs are discussed.

https://arxiv.org/abs/1906.03591

To learn more – contextual embeddings

Learn More



[Submitted on 16 Mar 2020 (v1), last revised 13 Apr 2020 (this version, v2)]

A Survey on Contextual Embeddings

Qi Liu, Matt J. Kusner, Phil Blunsom

Contextual embeddings, such as ELMo and BERT, move beyond global word representations like Word2Vec and achieve ground-breaking performance on a wide range of natural language processing tasks. Contextual embeddings assign each word a representation based on its context, thereby capturing uses of words across varied contexts and encoding knowledge that transfers across languages. In this survey, we review existing contextual embedding models, cross-lingual polyglot pre-training, the application of contextual embeddings in downstream tasks, model compression, and model analyses.

https://arxiv.org/abs/2003.07278

To learn more - word2vec

Learn More

https://research.fb.com/people/mikolov-tomas/

- Efficient Estimation of Word Representations in Vector Space, September 7th, 2013.
- Distributed Representations of Words and Phrases and their Compositionality, October 16th, 2013.

https://code.google.com/archive/p/word2vec/

These slides are available at http://eic.cefet-rj.br/˜ebezerra/

Eduardo Bezerra (ebezerra@cefet-rj.br)

Backup slides

A drawback of skip-gram

- The resulting trained model contains a huge amount of weights.
  - e.g., $d = 300$ and $|V| = 10{,}000$, that's 3M weights in the hidden layer and output layer each!
- Training (optimizing) a skip-gram model on a large corpus would be prohibitive.
- In order to circumvent this, some innovations…

Three inovations

1. Learn phrases from the corpus.
2. Subsample frequent words to decrease the number of training examples.
3. Modify the optimization objective with a technique called "Negative Sampling", which causes each training sample to update only a small percentage of the model's weights.