<http://www.davidsbatista.net/blog/2018/12/06/Word_Embeddings/>

<https://towardsdatascience.com/elmo-contextual-language-embedding-335de2268604>

<https://allennlp.org/elmo>

## Static Word Embeddings

which can also be seen as static word embeddings since the same word will always have the same representation regardless of the context where it occurs

* word2vec
* GLOVE
* Fast-text

## Dynamic Word Embeddings

most of which make use of some language model to help modeling the representation of a word.

* ELMO
* Flair Embeddings
* BERT

# **Word2Vec**

Is a predictive model, meaning that it trains by trying to predict a target word given a context (CBOW) or the context words from the target (skip-gram). The model uses trainable embedding weights to map words to their corresponding embeddings. Which are used to help the model make predictions. The loss function for training model is related to how good the model’s predictions are, so as the model trains to make better predictions it will result in better embeddings.

## CBOW vs SKIP-Gram

CBOW is learning to predict the word by context, or maximize the probability of the target word by looking at the context, this happens to be a problem for rare words. For example given the context *‘yesterday was a really [..] day ‘* CBOW model will tell you that most probably the word is *‘beautiful’* or *‘nice’.* Words like *‘delightful’* will get much less attention of the model, because it is designed to predict the most probable word. This word will be smoothed over a lot of examples with more frequent words. several times faster to train than the skip-gram, slightly better accuracy for the frequent words.

On the other hand, the skip-gram model is designed to predict the context. Given the word *‘delightful’* it must understand it and tell us that there is a huge probability that the context is *‘yesterday was really [...] day’*, or some other relevant context. With skip-gram the word *‘delightful’* will not try to compete with the word beautiful but instead, *‘delightful + context’* pairs will be treated as new observations. works well with small amount of the training data, represents well even rare words or phrases.

# **ELMo**

The ELMo (Embeddings from Language Models) design uses a deep bidirectional LSTM language model (Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies) for learning words and their context. The deep BiLSTM (Bi-directional LSTM) architecture allows ELMo to learn more context-dependent aspects of word meanings in the higher layers along with syntax aspects in lower layers. This results in better word embeddings, and different representations of a word depending on the context it appears in (especially useful for homographs).

The main idea of ELMo can divided into two main tasks,

1. Train LSTM-based language model on some corpus
2. Use the hidden states of the LSTM for each token to generate a vector representation of each word

The language model is trained by reading sentences both forward and backward. That is in essence there are two language models, one that learns to predict the next word given the past words and another that learns to predict the past words given the future words.

Instead of using a single layer LSTM use a stacked multi-layer LSTM. A single layer LSTM takes the sequence of words as input, a multi-layer LSTM takes the output sequence of the previous LSTM-layer as input, the authors also mention the use of residual connections between the LSTM layers. The different layers of the LSTM language model learns different characteristics of language.

ELMo is a task specific combination of the intermediate layer representations in a bidirectional Language Model (biLM). That is, given a pre-trained biLM and a supervised architecture for a target NLP task, the end task model learns a linear combination of the layer representations.

ELMo code at AllenNLP github –

<https://github.com/allenai/allennlp/blob/master/tutorials/how_to/elmo.md>

AllenNLP models

<https://allennlp.org/models>

# **Flair Embeddings**

The contextualized character-level word embedding which captures word meaning in context and therefore produce different and therefore produce different embeddings for polysemous words depending on their context. It model words and context as sequence of characters, which aids in handling rare and misspelled words and captures sub word structures such as prefixes and endings.

Characters are the atomic units of language model, allowing text to be treated as a sequence of characters passed to an LSTM which at each point in the sequence is trained to predict the next character.

The model is a forward and backward model character language model, essentially the character-level language model is just ‘tuning’ the hidden states of the LSTM based on reading lots of sequence characters. The LSTM internal states will try to capture the probability distribution characters given the previous characters.(forward language model) and the upcoming characters (backward language model)

From this forward-backward LM, the authors concatenate the following hidden character states for each word:

* from the fLM, we extract the output hidden state after the last character in the word. Since the fLM is trained to predict likely continuations of the sentence after this character, the hidden state encodes semantic-syntactic information of the sentence up to this point, including the word itself.
* from the bLM, we extract the output hidden state before the word’s first character from the bLM to capture semantic-syntactic information from the end of the sentence to this character.

Both output hidden states are concatenated to form the final embedding and capture the semantic-syntactic information of the word itself as well as its surrounding context.

Flair-embeddings github repo

<https://github.com/zalandoresearch/flair>

# **BERT**

BERT or Bidirectional Encoder Representations from Transformers, is essentially new method of training language models.

Pre-trained word representations, can be ‘context-free’ (word2vec , GloVe, fastText) meaning that a single word representation is generated for each word in the vocabulary, or can also be contextual ( ELMo, flair) on which the word representations depends on the context where that word occurs, meaning that the same word in different contexts can have different representations.

Contextual representations can further be unidirectional or bidirectional. Note, even if a language model is trained forward or backward, is still considered unidirectional since the prediction of future words (or characters) is only base on past seen data.

In the sentence: “The cat sits on the mat”, the unidirectional representation of “sits” is only based on “The cat” but not on “on the mat”. Previous works train two representations for each word (or character), one left-to-right and one right-to-left, and then concatenate them together to a have a single representation for whatever downstream task.

BERT represents “sits” using both its left and right context — “The cat xxx on the mat” based on a simple approach, masking out 15% of the words in the input, run the entire sequence through a multi-layer bidirectional Transformer encoder, and then predict only the masked words.

BERT uses the transformer encoder to learn a language model.

Github BERT:

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

More:

<https://www.lyrn.ai/2018/11/07/explained-bert-state-of-the-art-language-model-for-nlp/>

<https://jalammar.github.io/illustrated-bert/>