Effective Incorporation of Term Importance and Attention in Designing Question Answering and Natural Language Inference

Suryadipto Sarkar

Department of Computer Engineering,
School of Computing, Informatics, Decision Systems Engineering (CIDSE),
Ira A. Fulton School of Engineering,
Arizona State University
ssarka34@asu.edu
(+1) 480-399-8867

***Abstract*—Question answering and natural language inference are integral tasks in the domain of natural language processing (NLP). From personal assistants like Echo and Siri, to search engines and database locators, question answering and NLI are two of the most widely used concepts in today's world of technology. Sentiment analysis, part of speech tagging, word embeddings and other essential underlying concepts of question answering have been discussed in great detail in this paper. Another aspect that has been taken into account is the concept of word embedding generation. Glove and FastText have been discussed in detail. The Infersent sentence embedding technique has been used along with Glove to train the HotpotQA complex question answering dataset, and the resulys seem promising.**

I. INTRODUCTION

The concept of "attention" is a relatively new one; but it is in pervasive use in most—if not all—of the present, state-of-the-art natural language processing applications. Starting from sentiment analysis, single- and multi-hop question answering, machine translation and automatic part-of-speech tagging, to the detection of fake documents from real ones, NLP engineers are on the constant lookout for more effective incorporation of attention.

Bi-directional knowledge transfer, and supervised word and sentence embedding generation are two other important breakthoughs in the field. For a long time, NLP engineers have been using conventional RNNs for speech and language processing tasks. However, with the advent of bi-directional LSTM and GRUs, knowledge and attention propagation have become especially convenient to implement.

Some of the datasets used in the project are:

1. The HotpotQA multi-hop questioning dataset[1]
2. SQuAD dataset[2]
3. SNLI dataset[3]
4. "Quora question-answering pair" dataset[4]
5. The sample dataset provided on the Infersent Github page[5].

II. PRE-PROCESSING THE DATASETS

Natural language cannot be directly used in model training. It must first be either converted to formal language (by means of transformations, lambda transitions et cetera); or, we need to adopt certain pre-processing methodologies to prepare the data as uniform, logically understandable components. This section shall delve into a discussion about the most common pre-processing techniques employed on natural languages before they can be trained for the purpose of question answering.

Three of the most commonly deployed pre-processing techniques in NLP are: tokenization, annotation and normalization. Note that there are lots of other pre-processing techniques, but these three are absolutely necessary to produce clean, uniform, trainable data.

1. Tokenization

Tokenization refers to the conversion of text data into smaller units. This involves how we handle apostrophes, hyphens et cetera. For example, the sentence "My name is Surya. I study in ASU." After tokenization shall be converted to ["My", "name", "is", "Surya", "I", "study", "in", "ASU"].

2. Normalization

Normalization refers to the conversion of multiple forms of a word into one uniform format. For example, 'Dog', 'DOG', and 'DOG' are all converted to "dog". Also, connectors and frequently used conjunctions are often ignored—but this is a tricky procedure, and must be done with care. It often requires a certain amount of trial and error before we can actually go ahead and convert determinants (a, and, the); prepositions (he, she,

them); connector verbs (is, are, am et cetera) into normalized form.

One problem with normalization is the loss of knowledge, sometimes even context. For example, DoG refers to "difference of gaussian", whereas dog refers to the animal. However, both shall be converted to "dog".

Normalization also takes care of preposition phrase variations. Most often, "is", "are", and "am" are converted to "be".

3. Annotation

Annotation refers to the action of segregating a word into multiple different possibilities. Very often, in articles and research papers, this is wrongly referred to as POS tagging. But this is in fact an enormous field of study in itself, part of speech tagging being just a small sub-field. This is an especially important step, and has the capability of affecting the performance of the model to a considerable extent. Two of the most common types of annotation techniques are:

a. POS tagging

Part-of speech tagging annotates the words with the corresponding part of speech, based on context. For the purpose of POS tagging, the context is the sentence that the word is present in.

For example, given two sentences:

Sentence-1: *The fly is buzzing around.*

Sentence-2: *It's time for me to fly to LA.*

After POS tagging, "fly" in the first sentence shall be converted to "fly/VB" [verb], and that in the second one shall become "fly/NN" [noun].

b. Word sense/ semantic tagging

As the name does suggest, word sense tagging—more commonly known as semantic tagging segregates similar words with different meanings. This comes in especially handy when we are trying to build a system that answers questions based on text comprehension. Some of the applications also include thesaurus word generation, and relationship score generation between word pairs.

III. MATHEMATICAL PROCESSING AND ELEMENT WEIGHTING

In this project, I have made use of semi-automated mathematical processing and weight matrix generation.

Once the tokenization, annotation and normalization steps are complete, the frequency matrix of the vocabulary is created. However, we need to make a few considerations:

i. Uncommon words are more meaningful—that is, they hold more information—than common ones. For example, when searching for a document, the term "histogram of gradient" shall hold much more value than frequently occurring terms like *is*, *or*, *and* et cetera.

ii. The frequency of words plays an importance, but importance is not directly proportional to number of occurrences. That is, a document with more instances of the query keyword are considered to be more important. But if the query set appears twice, that doesn't necessarily mean that the search itself is twice as important.

To take these considerations into account more effectively, I have defined a customized version of the log-frequency weight matrix:

$$w_{t,d} = \begin{cases} 1 + log_{10} tf_{t,d}\ , & if\ tf_{t,d} > 0 \\ 0\ , & otherwise \end{cases}$$

$$Score = \sum_{t \epsilon q,d} 1 + log_{10} tf_{t,d}$$

Therefore, score is 0 if none of the query terms is present in the document. We shall be using this concept a lot throughout this project.

The idea of inverse document weight (IDF) is the same as the original formula. We want to alter the tf-idf—which is brought about by making changes to the term-frequency part of it, but without altering the document frequency definition. The reason behind this is that, the number of documents with the existing query terms is always going to be a constant, given a knowledge base.

The *idf-score* is thus defined as:

$$idf_t = log_{10}\left(\frac{N}{df_t}\right)$$

$$where, df_t\ is\ the\ document\ frequency\ matrix\ of\ the\ term\ t$$
$$N\ is\ the\ total\ number\ of\ documents$$

The *tf-idf* ranking score changes based on the new definition of *tf-score*:

$$w_{t,d} = \log\left(1 + tf_{t,d}\right) \times \log\left(N/df_t\right)$$

IV. METHODOLOGY

A. Improving attention

Formulating importance in a more intuitive, effective manner, and incorporating it in the model's attention mechanism, was my primary goal in this part of the project.

I used a one-layer bi-directional LSTM encoder in combination with a word-wise importance generator model. The attention layer, as obtained from the encoder using the hidden context vectors and the hidden question

vectors, are then scaled with the importance scores of the respective words.

From the set of encoder RNNs, we obtain the attention scores by performing the dot product of every one of the context vectors c_i with every one of the question vectors q_j. This results in a set of attention scores s_i, which can be given by the equation:

$$s_i = c_i \bullet q_j$$
$$= \left[c_i^T q_1, \dots, c_i^T q_M\right] \epsilon R^M$$
$$where, M\ is\ the\ number\ of\ words\ in\ the\ question$$

Next comes in the system that generates the word-wise importance score in the context. This model works in parallel with the encoder RNNs. It takes as input the context once again, and it generates scores using the SentEval toolkit which uses Facebook's Infersent sentence embedding.

Note that Infersent is a sentence embedding as opposed to a word embedding. Also, unlike other embedding techniques, Infersent is a supervised embedding generation technique, which trains itself to learn sentence embeddings from annotated Facebook data.

Many recent research papers have concluded that supervised embedding generation performs better—especially when it comes to longer texts like sentences, paragraphs et cetera, than unsupervised ones.

Also, Infersent doesn't require any training. You only need to feed in sufficient text. The Infersent embedding generation model builds a vocabulary out of the provided text, and performs sentence embeddings automatically.

For the purpose of this task, I have used the "sample text" file from the Infersent Github repository to build my model.

In my question-answering system, each sentence is used as a context. The words are fed sequentially into the RNN encoder. Note that every word, hidden state, and output from the RNN is a vector; and the general rules of vector mathematics apply.

The symbols that we shall be using are as follows:

1. Each word, which forms a part of the sentence context, is annotated as $c_i\ , i = 1\ to\ N$. That is, the size of the sentence in consideration comprises 'N' words.

2. Each word in a question is annotated as $q_i\ , i = 1\ to\ N$. That is, the question corresponding to the present context being fed into the RNN as input, c_i, comprises a total of M words.

3. However, note that when c_i and q_j are denoted as such, that means that the question q_j does not correspond to the context word c_i. Such situations will arise when we perform dot product of the context and question vectors.

4. $x_{i:j}$ refers to a sequence of vectors from x_i to x_j. In other words, $x_{i:j} = [x_i, \dots, x_j]$

I designed the model to be able to accept two inputs, and produce two outputs at every state. The following RNN properties apply:

1. The initial hidden state is denoted as s_0. At every state i, a new hidden state s_i shall be generated, which shall be passed onto the next state.

2. The inputs at each state i, are represented by $\{c_i, q_i\}$. If the question has run out of words while the context is still being fed as input, that is not a problem because the RNN runs alright with uneven input sizes. All the weight calculations, that we will see later on, are generated as 0 by the RNN for missing inputs automatically.

The output of the bi-directional LSTM being used here—which is essentially an RNN network—is given by the following set of equations:

$$RNN(s_0\ , c_{1:n}\ , q_{1:n}) = s_{1:n}\ , y_{c_1:c_n}\ , y_{q_1:q_n}$$
$$s_i = R(s_{i-1}\ , c_i\ , q_i)$$
$$y_{c_1} = O_1(s_i)$$
$$y_{q_1} = O_2(s_i)$$
$$where, functions\ R, O$$
$$= \{O_1\ , O_2\}\ remain\ same\ for\ every\ state$$

We use bi-directional LSTMs because we want the words to be aware of all the other words that are not only ahead of them, but behind them as well. The output context vectors at each step are multiplied by the corresponding output question vectors. This is the step that first introduces the notion of "attention". By multiplying the context vector with the question vector, what we are doing is that we are telling the system what words in the question to pay attention to. Not only that, high product between the context and the question is also a way of measuring similarity between the two. The resulting weighted context vector is obtained by the following equation:

$$w_{c_i} = y_{c_i} \bullet y_{q_i}$$
$$where, w_{c_i} \epsilon R^N$$

Next, we generate a vector of N scores pertaining each one of the questions q_j , where $j = 1\ to\ n$, by feeding

each of the questions into the SentEval toolkit as described before.

The attention scores are generated as:

$$e_i = \left[y_{c_i}^T q_1, \ldots, y_{c_i}^T q_M\right] \epsilon\, R^M$$

Attention distributions per question are then generated as follows:

$$\alpha_i = softmax(e_i) \epsilon\, R^M$$

The entire process of generating attention distributions starting from the initial state s_0 and inputs $c_{1:n}$, $q_{1:n}$, is described in figure 1.

The final calculation of attention (a_i) is done by multiplying the attention distribution (α_i) with the corresponding question vector (α_i), as follows:

$$a_i = \sum_{j=1}^{M} \alpha_i q_j$$

Now, we know that $\alpha_i \epsilon\, R^M$. Therefore, the dimensions of a_i are the same as well. a_i is just the weighted sum of questions, where each question is weighted as per the corresponding probability of the answer to that question being in the particular context.

All we need to do is pass the attention score through another softmax layer. This gives an array of probabilities corresponding to each question q_j , $j = 1\ to\ M$. Now, for datasets that produce one-word answers (or two words, in case of people's names), we need to select the words with the top or top two probability. For datasets that provide answers in sentences, we need to select the output a little differently. After this point, it is just data processing, basically. Some applications even use a couple of feedforward layers in the end followed by a final softmax layer. But, I did not have to do that.

Some of the important contributions that the word-wise importance score generator had in improving the attention of the system were as follows:

1. It places greater importance (or relevancy) to rare words. The reason for this is because the model believes that more uncommon a word, the more information it holds; and vice versa. Figure 2 is a plot of generated importance scores for the sentence "This is all gobbledygook to me." Gobbledygook is a very uncommon term, but it is an actual word.

2. It has the capability of excluding words with spelling mistakes, or non-words, even though they might be uncommon. Fig 3 corresponds to the sentence "Give me a shpen." Notice how it excludes the word "shpen", and returns scores for all of the other words (including ".").
3. Names of places are given extremely high importance, and names of people are given pretty high importance (not as much as names of places, though). Figure 4 gives a plot of generated word-wise importance scores for the sentence "Jack is going to Japan tonight. Mary just returned from Nairobi." Notice how "Japan" and "Nairobi" have the highest importance score, followed by "Jack" and "Mary".
4. It can differentiate very uncommon names from non-names. (Therefore, it goes without saying that it can identify common names with ease.) Figure 5 depicts the word importance score corresponding to the sentences: 1. "Cassandra and Epictetus are having a fight." 2. "Bibbob gave me some money."
   Note that "Cassandra" and "Epictetus" are very uncommon names, whereas "Bibbob" is a non-name.
5. The model is good at differentiating between context-wise repetitions of the same word. "The flies are buzzing. He flies to Japan tonight."

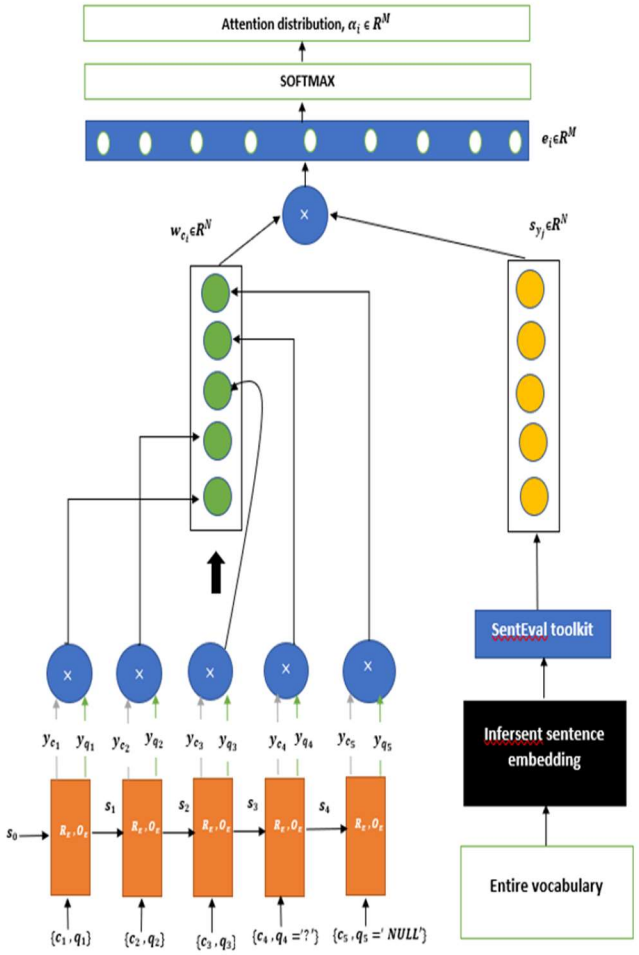

Fig 1. The process of generating attention distributions starting from the initial state s_0 and inputs $c_{1:n}$, $q_{1:n}$

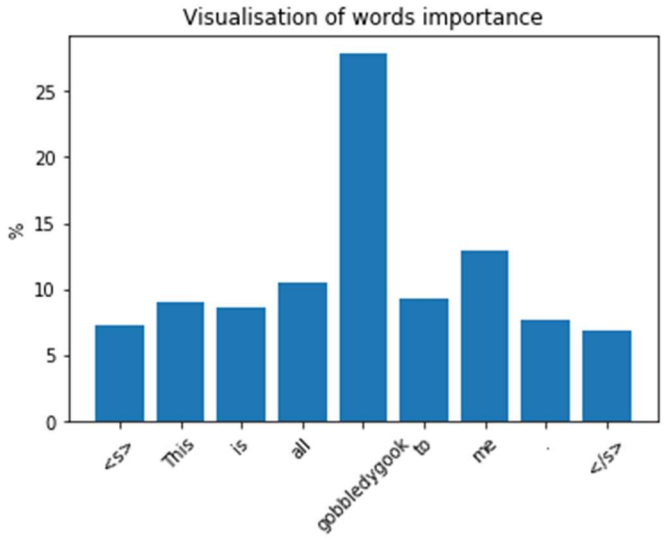

Fig 2. Importance score generation for "This is all gobbledygook to me."

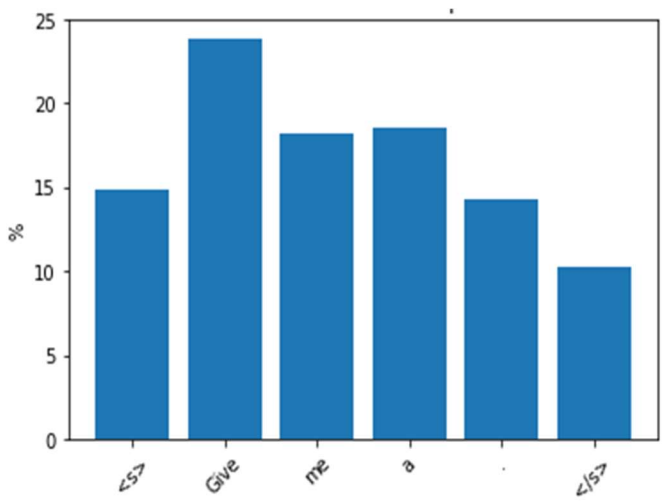

Fig 3. Importance score generation for "Give me a shpen."

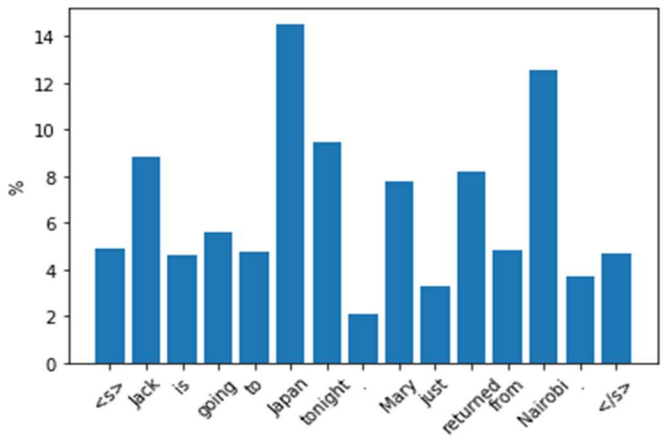

Fig 4. Importance score generation for "Jack is going to Japan tonight. Mary just returned from Nairobi."

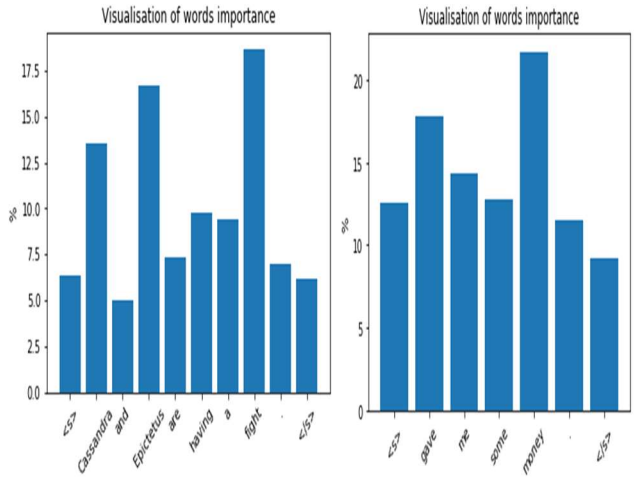

Fig 5. Importance scores for: 1. "Cassandra and Epictetus are having a fight." 2. "Bibbob gave me some money."

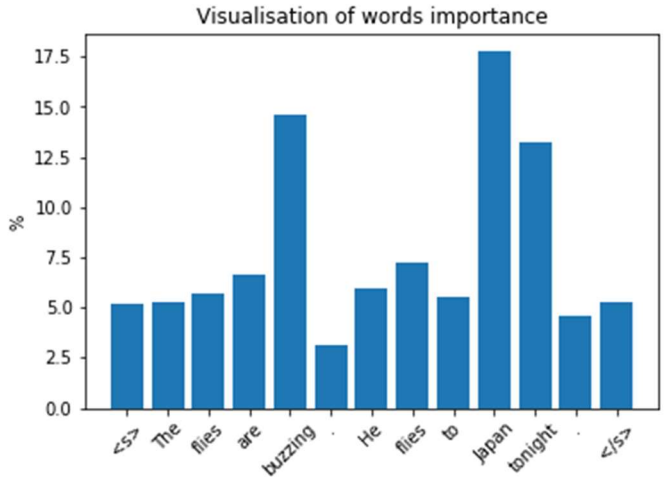

Fig 6. Importance score plot for "The flies are buzzing. He flies to Japan tonight."

B. *Improving context*

The HotpotQA dataset has been used for this part of the implementation. In this section, I shall walk you through the semi-automated embedding generation technique that I incorporated using an ensemble of Glove[6] and Fasttext[7], and how it compares with the currently available state-of-the-art embeddings.

I wanted to focus on small datasets, therefore only used the first 1000 samples from the HotpotQA multi-hop question answering dataset.

What Glove does is, it uses bigrams to learn relationship between words. This means that it generates an enormous frequency matrix, which is computationally very expensive. Also, it does not learn the data all that well because it does not take into account the concept of "context". (It iterates through the model, and tries to learn the essence.) Another problem with this approach is that it has a hard time generalizing from bigrams to n-grams.

I trained all of the questions whose answers involved two letters (names), and numbers (date, year et cetera, but not exceeding two separate words) using Glove—since it uses bigram training; and everything else using Fasttext.

This approach has resulted in:

1. An improved answering accuracy
2. Decreased processing and word embedding generation time.

Table 1 gives a set of accuracy metrics as obtained from the baseline code[8], with and without the use of the proposed embedding generation technique.

Note that the model has been trained on the "fullwiki" setting only, and not the "distractor" one because my intention is to use a very complex dataset with small training samples to check how nuanced the model actually that—that is the primary reason that, for this particular task, I decided to go with a tough to answer dataset like Hotpot instead of SQuAD, ConsmosQA or CommonsenQA.

Also note that the F1-scores generated by other models, as reported in table1, have been taken from the respective papers, which have been trained on the entire data. The results that I obtain using the proposed embeddings only takes into account the first 700 samples from the HotpotQA dataset for training. It would be reasonable to assume that if I were to train it on the entire dataset, it would yield results that might be significantly better.

Table 2 gives a set of F1-scores of other state-of the models.

Model	F1-score (%)
BERT (1-hop trained) + proposed embeddings	33
BERT (1-hop trained)	29.97
BiDAF + proposed embeddings	35
BiDAF	32.89

Table 1. F1 scores of the baseline models with and without the proposed word embedding generation technique. All of the models have been trained and evaluated in the "fullwiki" setting.

Model	F1-score
DecompRC	40.65
MultiQA	40.23
QFE	38.06
GRN	36.48
BiDAF	32.89
SemanticRetrievalMRS	45.32
Quark + SemanticRetrievalMRS IR	55.50

Table 2. F1-scores of other state-of-the models trained on the HotpotQA multi-hop question-answering data in the "fullwiki setting."

V. RESULTS

The Stanford Natural Language corpus is a collection of 570,000 human-written, annotated, English sentence pairs. Each pair is labelled "entailment", "contradiction" or "neutral". This SNLI dataset has been used by me for natural language inference classification. The model yields an accuracy of 73%. Table 3 gives the model-wise accuracies of some of the best performing LSTM and bi-directional LSTM models. Note that the other models have not been implemented by me; the accuracy values are as they have been presented in the respective papers.

Model	Accuracy(%)
LSTM	80
biLSTM-Mean	78.2

Table 3. model-wise accuracies of some of the best performing LSTM and bi-directional LSTM models, for the NLI task on the SNLI dataset.

REFERENCES

[1] Yang, Zhilin, et al. "Hotpotqa: A dataset for diverse, explainable multi-hop question answering." *arXiv preprint arXiv:1809.09600* (2018).

[2] Rajpurkar, Pranav, et al. "Squad: 100,000+ questions for machine comprehension of text." *arXiv preprint arXiv:1606.05250* (2016).

[3] Bowman, Samuel R., et al. "A large annotated corpus for learning natural language inference." *arXiv preprint arXiv:1508.05326* (2015).

[4] Sharma, Lakshay, et al. "Natural Language Understanding with the Quora Question Pairs Dataset." *arXiv preprint arXiv:1907.01041* (2019).

[5] facebook/Infersent/GitHub link: https://github.com/facebookresearch/InferSent

[6] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

[7] Word vectors for 157 languages: https://fasttext.cc/docs/en/crawl-vectors.html