MAPPING OF AUTHOR CREATED LEARNING OBJECTIVES TO OFFICIAL COMMON ACADEMIC GOALS

Gandalf Saxe, Jacob J. Hansen, Jakob D. Havtorn and Yevgen Zainchkovskyy

Project supervised by prof. Ole Winther, DTU Compute

ABSTRACT

The Danish publisher Gyldendal has a set of learning objectives that they would like linked to the official learning objectives of the Ministry of Education in order to ensure that the material lives up to certain requirements. Different models within natural language processing, including LSI and fastText, were applied to the task and their performance evaluated. FastText, being the best model, was implemented into a reference web application, usable by Gyldendal.

***Index Terms*—** Natural language processing (NLP), LSA/LSI, fastText, word2vec, sequence mapping.

1. INTRODUCTION

Gyldendal has a portfolio of online learning material for use in the danish primary school. The Ministry of Education has set a series of Common Academic Goals (FFM, Fælles Faglige Mål), for each subject at each grade level (e.g "Danish, 3.- 4. grade). Gyldendal's authors have in most cases defined their own learning objectives which are generally more specific to the material itself, and thus might not be in line with the ones from the Ministry of Education. These are referred to as authors' learning objectives (LM). Thus Gyldendal needs a model to translate an LM to the most likely FFM(s).

At the current time, Gyldendal has very few labeled data. Therefore, an unsupervised model is the only choice and a small set of labeled data is provided for testing purposes. The overall goal is to construct a model, that can assist in creating a labeled data set under human supervision (through a webinterface). In time, a more advanced supervised model can be trained on this augmented data set, and potentially combined with the previous unsupervised one.

2. WORD REPRESENTATION

2.1. The Distributional Hypothesis and Word Embeddings

Much work in natural language processing (NLP) assumes the distributional hypothesis, i.e. words that are used and occur in the same contexts tend to purport similar meanings[1].

2.2. Word Embeddings

Word embeddings is a class of techniques in NLP in which words are mapped to continuous vectors of real numbers in some dimension. There are two broad categories of methods that leverage this principle:

1. Count-based methods Compute the statistics of how often some word co-occurs with its neighbor words in a large text corpus, then map these count-statistics down to a small dense vector for each word[1]. *Example: Latent Semantic Indexing (LSI), see sec. 4*.

2. Predictive methods Try to directly predict a word from its neighbors in terms of learned small dense embedding vectors[1]. *Example: Neural probabilistic language models*. In recent years, the predictive models (in particular neural networks), have been proven to be the most successful of the two [2, 3].

2.3. Word2vec: CBOW and skip-gram

Word2vec is a group of related predictive models used to produce word embeddings[4] that are particularly computationally efficient for learning word embeddings from raw text[1]. They're based on a shallow two-layer neural network that we will show in the next section. Word2vec comes in two variants:

1. Continuous bag-of-words (CBOW) model.
2. Skip-gram model.

Consider the phrase "the cat sat on the mat". Both CBOW and Skip-gram are algorithmically similar, but CBOW predicts target words (e.g. "mat") from the target context words ("the cat sat on the"), whereas skip-gram predicts context words (the, cat, sat, on, the) from the target word (mat). In practice it has been found that CBOW is better for smaller data sets, and skip-grams better for larger data sets. This is due to the fact that CBOW treats a context as one observation, whereas skip-gram treats each context-target pair as a new observation. The result is that CBOW smoothes over a lot of the distributional information by treating an entire context as one observation[1].

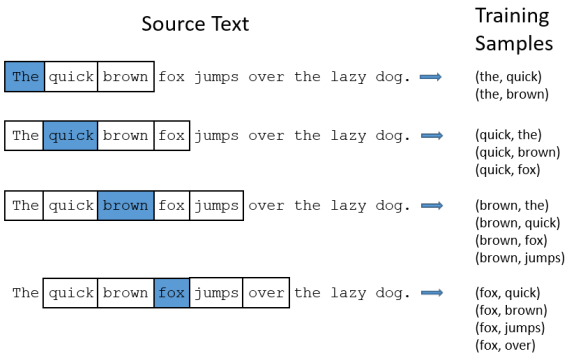

Fig. 1. How training samples are generated from the source text in the skip-gram model. Note that the distance of the context word from the target word is not taken into account in the simplest model, however more advanced skip-grams model does weight the words so that context words close to the target word is weighed higher during training. *Source: [1].*

The training corpus will determine the kind of language usage that the model will be suitable for representing. For many applications sufficiently large amounts of data are available that skip-grams is a popular choice today. An example of generation of training data from a sentence is shown in figure 1.

2.4. Word2Vec: Basic Neural Network Model

Figure 2 shows the two-layer network. Words are fed into the neural network as a one-hot encoded vector. First a vocabolary is built by the unique words in the training corpus (after word stemming). In the example in figure 2 we have a vocabolary 10,000 words.

The input vector is connected to a hidden layer. There is no activation function on the hidden layer neurons. The vectors for the various words is found as the weights of the input and hidden layer. More specifically, the rows of the weight matrix between the first and second layer are the word vectors. Thus the hidden layer weight matrix acts as a simple lookup table of word vectors, see figure 3.

2.5. Basic Word2vec Optimizations: fastText

A number of modifications are made to the basic model above to achieve better predictive performance and speed of training. All of the following improvements to the basic Word2vec is used in fastText, a recent text representation and classification library by Facebook.

2.5.1. Subword sampling

In [5] describes the benefits of breaking down words into n-grams. By default, fastText uses $n \in 3, 4, 5, 6$. As we see in

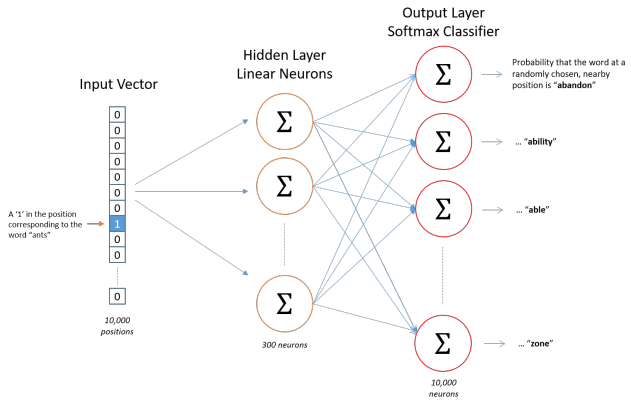

Fig. 2. Word2vec is a shallow two-layer neural network with one-hot encoded input vector, no activation function on hidden layer neurons, word vectors in the rows of the hidden layer weight matrix, and softmax function on the output layer neurons.

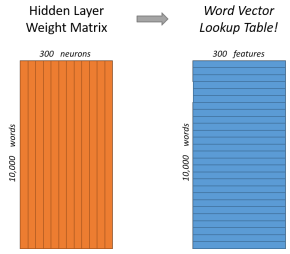

Fig. 3. Illustration of how the hidden layer weight matrix acts as a lookup table of word vectors. *Source: [1]*

figure 4, this scheme allows the capture of more syntactics out of the words, apart from just semantics[5]. The input layer in the neural as shown in figure 2 that was originally words is thus replaced with word fragments in the form of all 3-6 n-grams from the corpus.

2.5.2. Additional Modifications

In [6] further modifications to the basic Word2vec model are described :

1. Treating common word pairs or phrases as single "words" in the model.

2. Sub-sampling frequent words to decrease the number of training examples.

3. Modifying the optimization objective by a technique called "negative sampling", which causes each training sample to update only a small percentage of the model's weights. In essence, it means that we only update a small random number (on the order of 2-20 [7]) of output layer weights corresponding to the one context word we expect (whose weight is set to 1) in that word pair, and "negative" words which we know is not in the

target word context (whose weights we set to 0). This turns the basic skip-gram model into what's know as skip-gram negative sampling (SGNS) model.

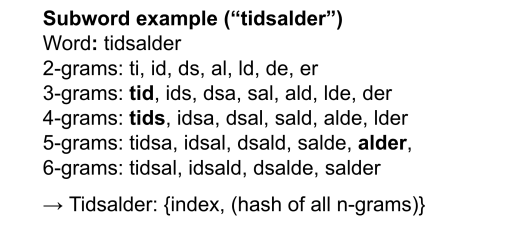

Fig. 4. Words are broken into many component n-grams, and the resulting "bag of n-grams" are used as training data instead of words. Here, the meaningful component subwords "tid", "tids" and "alder" is captured.

3. THE CLASSIFICATION PROBLEM

Gyldendal has a database of the LMs. This database does not currently feature associations from the LMs to any relevant FFMs. The LMs are however grouped by grade level and subject, similarly to the FFMs (e.g "Danish", "3. - 4. grade").

The classification problem is to predict the most relevant FFMs when given a specific LM. The problem is thus a natural language processing problem based on sentence to sentence mapping. Since the LMs are generally more specific to the Gyldendal material while the FFMs are written in a more abstract language, the challenge lies in increasing the abstraction level of the given LM to match that of some of the FFMs. For example, a given LM could be "The student can explain the differences between peasants, servants and chiefs". A possible FFM mapping could then be "The student has knowledge of communities before and now". This illustrates the higher abstraction level of the FFM.

In order to achieve this understanding of abstraction, the language model must have some understanding of synonymy and polysemy.

4. LATENT SEMANTIC INDEXING

Latent semantic indexing (LSI), is a natural language processing technique which attempts to find the underlying concepts of a document [8]. It was examined as an initial model for the LM to FFM mapping task.

LSI works by first constructing a count matrix, $\mathbf{X}$, of the given documents. In this setting, the documents are the FFMs. The count matrix has FFMs mapped to columns, words mapped to rows and the number of occurrences of a word in a sentence as the elements. As such, the count matrix becomes sparse. The word occurrence weights are enhanced by the term frequency-inverse document frequency (tf-idf) weighting, which increases the importance of rare words and decreases it for common words.

The count matrix is then decomposed by the Singular Value Decomposition (SVD), given mathematically by

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T.$$

This effectively extracts the correlations between words and FFMs into the left and right singular vectors in the columns of $\mathbf{U}$ and $\mathbf{V}$, respectively, to form a semantic space. By choosing the the k largest singular values from $\mathbf{S}$ and the k first vectors of $\mathbf{U}$ and $\mathbf{V}$ a so-called rank-k approximation of the count matrix is obtained. This effectively means that only the k most important dimensions of the semantic space are used for modelling of FFMs. The LSI model used in this paper used $k = 12$.

A given LM can then be compared to the possible FFMs by constructing its semantic space representation and measuring the cosine similarity between it and the FFM representations. The most similar FFMs can then be presented to the author as possible matches.

5. FASTTEXT

FastText builds ontop of word2vec and enhances it as described in section 2. FastText was trained on the whole danish Wikipedia and it is then used to map one FM and the possible FFMs (in this case, "History 3.- 4. grade") into 300-dimensional vectors. The FFMs are then ranked using the cosine similarity between a FFM and the FM, with closest meaning most similar.

6. FASTTEXT ONLINE

In an attempt to further increase the accuracy of the aforementioned fastText model, an online-learning approach has been tried. We included already known mappings in the distance calculation routine, thus artificially boosting the database of known FFMs. Intuitively, an LM that is close to another, known LM, should point to the same FFM (fig 5).

However, after estimating performance based on an averaged 5-fold cross validation, a slight decrease in accuracy compared to the fastText model (see fig. 6) has been seen. We believe that this unsatisfying result is due to the low amounts and fragmentation of the provided labeled data.

7. MODEL PERFORMANCE AND COMPARISON

The performance of the different models are shown in fig 6, where each model ranks the 11 FFMs for every FM and then we check if the correct answer is in the top 1,2,3,...,8 suggestions. Random guessing is also shown for comparison.
Fasttext: Performs the best across the board, with up to 83% accuracy in top 5

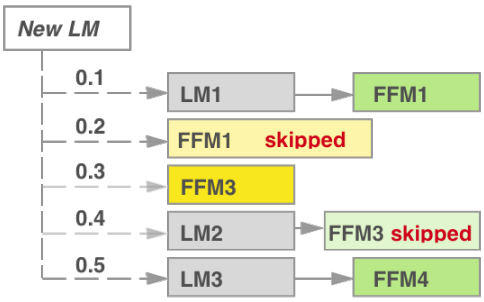

Fig. 5. Online model diagram showing how resulting suggestions `FF1`, `FFM3` and `FM4` are chosen based on ascending distance (0.1, 0.3, 0.5) and existing (e.g. `LM1` → `FFM1`) mappings.

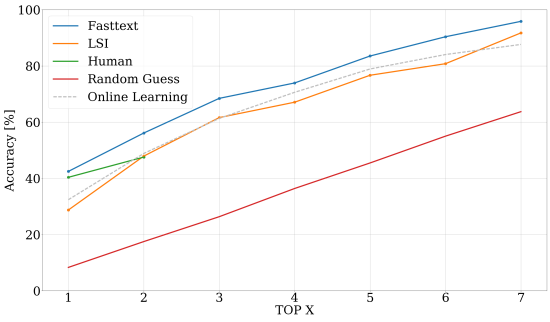

Fig. 6. Accuracy of models, comparing if the right answer is within the top X suggestions for all LM's

LSI: Also a solid model, but with worse performance compared to fastText, especially its first suggestion is bad, with only 29% accuracy compared to fastText's 42%
Online Learning: As mentioned earlier, this actually performs a little worse than fastText with the current data.
Human: To get a feel for the task at hand, we tried personally to classify the 77 LMs. The achieved accuracy of 40% on the first suggestion and about 47% on top 2 shows that it is not by any means a trivial task to perform quality mappings, even for a human.

It should be noted that the labelled data is sparse and that the labelling of an LM with a certain FFM in many cases has no clear ground truth. As such, the numerical values of the model accuracies are somewhat arbitrary, but the relative model performances remain a valid grounds for comparison. As noted, the human mapping only agreed on 40% of the labeled data and it was also noted, that many LMs were matched even when the matching between an LM and any of the FFMs made no apparent sense. Looking forward, a more diverse test-set has to be made, over different subjects, grades and annotated by more than just one person to give a more precise estimate of the accuracy of the models.

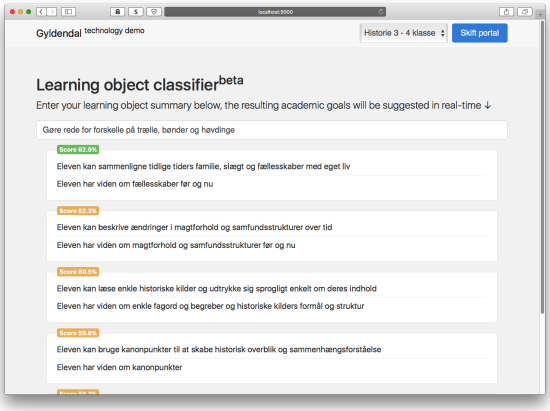

Fig. 7. Screenshot of a reference web-based implementation of our fastText model.

8. WEB APPLICATION

Finally, as this project is a collaboration with an industry partner, and to ease the adoption and implementation of our solution at Gyldendal, we have created a functioning web application to showcase our model. It's front-end consists of a simple real-time interface with an input field, where authors enter learning objectives (LM's) and are presented with a distance sorted list of the 5 best matching FFM's (see fig. 7).

The demo uses our best performing fastText model and is based on a Python fastText library with an addition of several other libraries to ease web implementation: Flask[1], jQuery[2] and Bootstrap by Twitter [3].

9. CONCLUSION

In this project, a real-world problem for a company was sought solved. Data was mined from Gyldendals web services and different models were fitted to the data. Facebook's fastText proved to be the best classifier after it had been trained on the whole Danish Wikipedia. This model was then implemented as a web application to be used for reference by Gyldendal. The final model is an unsupervised one since the provided data did not contain enough labelled entries to both construct and validate a supervised model. A final product has been made, capable of assisting Gyldendal in creating a larger and more diverse annotated dataset, enabling us to both compare our unsupervised model to a better ground truth, and to begin constructing a model which learns from the annotated links.

[1]Flask - web framework for Python `http://flask.pocoo.org`
[2]jQuery javascript library `https://jquery.com`
[3]Twitter Bootstrap CSS framework `http://getbootstrap.com`

10. REFERENCES

[1] "Tensorflow Word2vec," https://www.tensorflow.org/tutorials/word2vec, Accessed: 22-05-2016.

[2] Marek Rei, "Don't count, predict," http://www.marekrei.com/blog/dont-count-predict/, 2014, Accessed: 22-05-2016.

[3] Marco Baroni and Georgiana Dinu, "Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors," 2015.

[4] "Word2vec," https://en.wikipedia.org/wiki/Word2vec, Accessed: 22-05-2016.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov, "Enriching word vectors with subword information," 2016.

[6] Tomas Mikolov, Google Inc, Ilya Sutskever, Google Inc, Kai Chen, Google Inc, Greg Corrado, Google Inc, and Jeffrey Dean, "Distributed representations of words and phrases and their compositionality," 2016.

[7] Chris McCormick, "Word2Vec Tutorial Part 2 - Negative Sampling," http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/, Accessed: 22-05-2016.

[8] Kevin P Murphy, "Machine learning: a probabilistic perspective," 2012.