Deep Learning for NLP Final Project Report

Tamara Atanasoska
University of Potsdam
tamara.atanasoska
@uni-potsdam.de

Bhuvanesh Verma
University of Potsdam
bhuvanesh.verma
@uni-potsdam.de

Abstract

This document represents the report for the final project of the Deep Learning for Natural Language Processing class. The main task was to reproduce the ”Enriching Word Vectors with Subword Information” by Bojanowski et al. We have implemented the a Skipgram model, enriched it with subword information and provided semantic similarity on English and German datasets as well as analogy as methods of evaluation. When it comes to implementing the subword information, we have diverged from the paper by using the Byte Pair Encoding algorithm. Done with computational constraints and data size in mind, it enabled us to keep just the most meaningful subword information in our embeddings. In this document we present the process, methods and findings.

1 Introduction

The ”Enriching Word Vectors with Subword Information” (Bojanowski et al., 2017)) outlines a method to improve upon the Skipgram model by adding subword information. In this implementation, a word vector does not contain just the representation of the word itself, but rather a sum of the word and it's subwords. This in turn, improves the original Skipgram model, specifically the continuous Skipgram model by Mikolov (Mikolov et al., 2013b), by allowing the model to learn word representations by taking morphology into account.

The authors way of adding the subword information is to divide the words into n-grams. To achieve this, the word is first divided into characters, and then, depending on the n size of the n-gram specified, into n-sized n-grams until there are no more characters in the word. This is a simple, yet very effective approach.

We have, however, picked a slightly different approach. During the implementation, we reached computational limitations concerning our resources available to train and develop. The answer to this problem presented itself as the Byte Pair Encoding algorithm(BPE) (Sennrich et al., 2016).

BPE enabled us to grow the data by adding subword information, but only keeping the most important n-grams of a word. In the related work section below, we will discuss our particular implementation, how it differs as well as how it fulfils the same function and further elaborate on the decision.

With our base Skipgram model being easier to train, we achieve very good results, comparable to the results in the paper. Our subword addition is somewhat lacking in that regard, with the semantic similarity and other measurements discussed in the later parts of this report dropping once added and trained. We consider this to be in direct relation to our lack of resources to train large datasets for a longer amount of time. Despite the computational limitations, we spent considerable time utilising platforms for hyperparamter tuning, doing countless experiments and upgrading the code.

2 Related work

In this section we will discuss both work that led up to the paper we are reproducing, as well as related work that emerged after the publication. As we have chosen a slight divergence in the subword information methods, we will also look at papers that add variety in that direction.

Character-level subword information models. There has been a good amount of research proposing models that work on character-level. Operating on character-level helps us with rare words and taking advantage of subword information, as well as including a wider variety of languages. Some approaches use several layers of convolutional neural networks(CNNs) with the output give to a ”highway” long-short term memory(LSTM) layer like

(Kim et al., 2015). Some other approaches use bi-directionl LSTMs (Ling et al., 2015), likewise the Char2Vec model (Cao and Rei, 2016). When comparing the accuracy and performance of word-level vs character-level models, we can see that the word-level models outperform the character-level when it comes to accuracy, and have lower computational cost. However, charachter-level models definitely outperform word-level models when it comes to language with richer morphology, where the subword information is key. (Bojanowski et al., 2015) At the end, the success of the models and the picking the right level (word vs character) to operate on is largely tied to the task at hand and the data available.

BPE based subword information models. GMNT(Google Neural Machine Translation) (Wu et al., 2016) uses two different variants of BPE models: the SentencePiece Model (Kudo and Richardson, 2018) and the WordPiece Model (Schuster and Nakajima, 2012). The WordPiece works similarly like our own chosen algorithm. WordPiece tokenizes words first and then applies BPE to extract the subword information. Instead of using the subword frequency as we do in this implementation, it uses the log likelihood. This is slightly improved in the SentencePiece model which is made in a way to not require tokenization(thus making it language agnostic), and instead starts from a character level by also preserving the spaces. BERT (Devlin et al., 2018) uses a version of the WordPiece model.

Hybrid character-word models. An interesting example pushing the idea of subword information further are the hybrid models, that try to combine both character and word level representations. In essence the architecture has both character and word level, and it able to work with both. The character level is used for more rare and unseen words. An example of this is the Hybrid NMT model (Luong and Manning, 2016). As the discussion and results show, the approach outperforms the word-level models. In some cases, the performance lacks behind the character-level only models.

3 Model

In this section we present the reproduced model of the original paper (Bojanowski et al., 2017). For the basic model we use the Skipgram architecture, adding negative sampling following the example of the paper. Skipgram is a kind of Word2Vec model (Mikolov et al., 2013a).

On top of the basic model, we enrich the word representations with subword information using the Byte Pair encoding algorithm. We will discuss both the basic model and the BPE algorithm in detail in their respective sections below.

3.1 Skipgram model with negative sampling

As a baseline model before enriching with subword information we used a Skipgram model with negative sampling.

The Skipgram model is composed of two embedding layers, that have the vocabuary size as input and the embedding size, which is a hyperparameter ”EMBEDDING_DIM” as output. The words, already tokenized and converted into numerical representations are inserted into the layers in batches. This design is necessary to support the addition of negative sampling.

Negative sampling. As an addition to the simple Skipgram model, we use negative sampling. Negative sampling was described a paper by Mikolov et al. (Mikolov et al., 2013b) and used in the model we are replicating.

We will discuss shortly our particular implementation. Specified by the hyperparamter ”NG_WORDS” (”negative words”), we take k number of randomly chosen words to be our noise distribution. The probability that decides which words get chosen is arbitrary, and can be specified before the calculation. We use the same ”unigram distribution $U(w)$ raised to the 3/4 power” that is found best in the paper by Mikolov et al. (Mikolov et al., 2013b). Further mode, we utilise the two embeddings layers discussed above to facilitate the calculation. The loss is also adequately altered.

Subsampling. Subsampling is described in the same paper as negative sampling by Mikolov et al. (Mikolov et al., 2013b). By subsampling, we understand the removal of words that show up very often and don’t provide much contextual information. Those are words like ”the”, ”of” etc.

$$P(w_i) = 1 - \sqrt{\left(\frac{t}{f(w_i)}\right)}$$

In this equation, t is a threshold parameter, which can be set before the calculation. We calculate on every word in the dataset, w_i. The end probability $P(w_i)$ is the probability a word is discarded. The words are then discarded for the dataset before it is passed to the model.

We implemented subsampling as an optional preprocessing step and consequently we were able to establish that it improves the model when enabled.

3.2 Byte Pair Encoding algorithm for subword information

Byte Pair Encoding has been used in variety of ways in NLP for several years now. In the 2 Related Work section we mentioned a few different models that use BPE to obtain subword information. In this section, we would like to elaborate on the decision to use BPE cosidering it was not part of the original paper (Bojanowski et al., 2017) and talk about our particular implementation.

BPE implementation. Using BPE for obtaining subword information is in detailed described in the paper by Sennrich at al. (Sennrich et al., 2016), including the code for this particular task. In our approach we have kept to the proposed frequency of subword chunks when calculating, unlike some newer versions using the log likelihood (Schuster and Nakajima, 2012).

Despite having "bytes" in the name, BPE in the case of NLP doesn't use bytes, but instead takes advantage of characters and character n-grams. By using statistical analysis, BPE finds the most common characters in a word, as well as the combinations of such consecutive characters. The algorithm works with a set target vocabuary size, and we run the number of "merges" of the characters until this vocabulary size is reached. In our implementation, we specify the amount of operations with the hyperparameter "NUM_MERGES". Along the merges, BPE will merge the most frequent pair of characters to create new joint consecutive characters, resulting in subwords.

The subword information merges happen before the training. Before we pass the vocabulary to the layers, as the original paper suggests (Bojanowski et al., 2017) we sum the representation of the word and the representations of the subwords. We do not use their hashing technique, but instead just use the same numerical representation lookup as we do for the word vocabulary.

BPE advantages for our particular case. One of the biggest advantages that BPE has when working with low computational resources is the ability to control the size of the vocabulary. The large datasets come with increased training time, as well as increased computational time when it comes to calculating all the n-grams of the dataset. Early in the development we reached a few limitations concerning processing the data, so we decided to try BPE to help with this issue. We still provide subword information, although it is obtained in a slightly different way than the paper suggests.

4 Experimental evaluation

Much like the evaluation in the original paper (Bojanowski et al., 2017), we used the measurements of semantic similarity and analogy. We also tried to follow and include some of the same datasets used in the evaluation. The biggest difference is that we only focused on English and German, while the original paper focuses on more languages.

4.1 Training and training limitations

Training limitations. The topic of computational limitations has been mentioned already throughout this report. We would like to shortly elaborate on this in this section.

Initially we intended to train on the large Wikipedia dataset, in full, and we attempted to write a system to accommodate this. It quickly proved to be difficult to achieve and we regularly ran out of memory during training on the GPU, or the training was very slow and with that problematic in the phase of development. Knowing that abandoning training on the full dataset that the original paper uses we will lose from accuracy and will make it harder to reach the numbers reached in the paper, we decided to use a subset of this data for training so we could completely the project successfully.

The UD Treebanks v2.8, both Engish and German, were much smaller datasets and were not presenting as such a problem. We focused on optimizing the model for the Wikipedia dataset, as it always gave better results than these two smaller datasets.

Overcoming the large dataset obstacle, we still reached computational limitations when trying to train models. The large batches and the learning rate required to have the model learn successfully and within a reasonable time to be able to develop and tune further were at odds with the subword computations in the model. Although appropriate measures were taken and safety code was put in place, the loss still sometimes becomes "nan" dur-

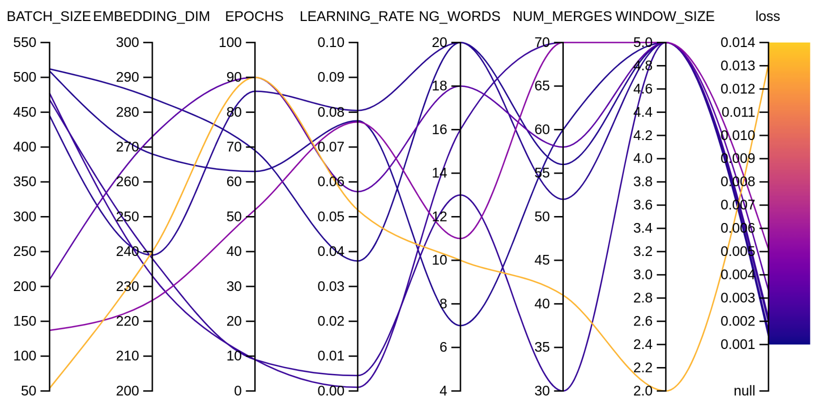

Figure 1: **Hyperparamter tuning in Weights & Biases for the German UD Treebanks v2.8.** We can see the different ranges of the hyperparamters, their mutual interplay and their relation to the loss. Each curvy line on the left represents a model, which upon training ends in the loss on the right.

ing training, thus breaking the training cycles and limiting the depth of the training. Our evaluation findings should be read with these limitations in mind, and we will mention them again further in the discussion.

Hyperparamter tuning. We tried both the platforms Tensorboard (ten) and Weights & Biases (wei) to tune our hyperparameters, learn about the models and eventually pick the best ones. This was heavily restricted due to the limitations mentioned above, but it was very informative. As Weights & Biases enabled better collaboration where we could share models and findings easier, we opted out for using it in all of our later experiments. An example can be seen in Figure 1. The discussion about the influence of the individual hyperparamters on performance can be found in the Discussion 4.6 section.

4.2 Evaluation methodology

We trained our model and obtained embeddings from the Wikipedia and the English and German UD version 2.8 treebanks. After training we ran the semantic similarity with human-judgement datasets and analogy evaluation tasks on those different emebeddings. We followed the suggestions provided in the task description for the final project. The different ways of evaluation and the results are discussed below.

4.2.1 Semantic similarity

Just like the paper, we take the Spearman's rank correlation coefficient (Spearman, 1961) between the human judgement score and the cosine similarity between vector representations. For German, we use three datasets: GRU65, GRU350 (Gurevych, 2005) and ZG222 (Zesch and Gurevych, 2006). For English also use three datasets: RG65 (Gurevych, 2005), RW (Sperr et al., 2013) and WS353 (Finkelstein et al., 2001).

We only evaluated for the words that are found in our vocabulary.

The semantic similarity was evaluated on the base Skipgram model with negative sampling and on the base model enriched with subword information.

4.2.2 Analogy

For the Google Analogy task we used the Google Analogy Test Set, introduced in the the paper by Mikolov at al. (Mikolov et al., 2013a), and it's German translation (Köper et al.).

We only evaluated for the words that are found in our vocabulary.

The analogy accuracy was evaluated on the base Skipgram model with negative sampling and on the base model enriched with subword information. We evaluated on the Wikipedia dataset, as well as the Treebanks datasets, both German and English. Unfortunately, with the amount of training we were able to do because of the unpredictable loss error, we didn't train long enough to see the analogy terms score high enough. Upon inspection, the desired words would appear often within the first 10 words predicted for the enriched subword model, however, they never reach the first place. For the base Skipgram model the performance was better, with the right word appearing in the first 5 predictions. This is in line with the rest of the evaluation findings.

We believe that training the model on a larger dataset, and not just a subset of the original Wikipedia dataset, as well as having the possibility and computational resources to optimize for learning rate and batch size and with that train longer, would have produced results that would be more agreeable to the task.

Athens is to Greece as Bern is to...
bern 0.8815888166427612
switzerland 0.31654098629951477
cantons 0.29430702328681946
greece 0.2695416510105133
Athens is to Greece as Cairo is to...
cairo 0.690380871295929
nasser 0.3314253091812134
egypt 0.3034687638282776
arab 0.2986876964569092

Table 1: **Top 4 predicted words for a given analogy followed by the similarity measurement**. The dataset was trained on the Wikipedia data, with the base Skip-gram model with negative sampling and it's most successful configuration. We can see that the right words comes really high, but is never the first one predicted.

4.3 Results

In this section, we will present some graphic and tabular presentation of the results obtained from the comparison with the semantic similarity datasets. We will also discuss the different configurations and training datasets and how they relate to the results. In the Analogy section 4.2.2 we quickly went over the results we obtained with that task. We will not mention those further.

The numbers in Table 2 and Table 3 are the best values we have gotten during out evaluation. Some are from different models, and we will carefully note those differences as they offer insights into the system.

4.4 Base model: Skipgram with negative sampling

The tabular results for the base model can be found both in Table 2 and Table 3. The base model is performing on pair with the numbers presented in the paper. We used an extra dataset, RG65, which is not featured in the original paper, which also has a high performance with the base model. The larges difference is when comparing the similarity of the RW, rare words dataset. We expect that a larger dataset and model training would increase these numbers too.

The results of the English Treebank dataset look quite different, with not much of disparity between the performance number for all the similarity datasets, including the RW dataset. The reason for this might be that this dataset is much smaller than any of the others, and contains different words than the ones in the Wikipedia dataset for comparison.

The base model requires it's own configuration that differs from the one that is the most successful for the enriched model. The "EMBEDDING_DIM" that fits best is of value 300, and the "NG_WORDS" were consistently better at a value of 5. A larger number of negative words hurt the performance, more in section 4.6.2. The best "WINDOW_SIZE" is 5. When it comes to the learning rate, we can be more flexible here since this model is less affected by the limitations discussed in section 4.1. A learning rate of 0.05 was used for most of the experiments. To achieve the high numbers presented in Table 2, a training time of 30 epochs is needed.

4.5 Base model + Byte Pair Encoding subword information

The tabular results for the base model can be found both in Table 2 and Table 3. Our enriched model has a lower performance on all accounts comparing to the base model. There are several reasons that we would like to discuss here.

Normalisation. To be able to consistently deal with the large values of the enriched model, the data is normalised. This might reduce the performance, as once normalisation is applied to the base model the performance drops to similar scores as the performance of the enriched model.

Dataset limitations. The model might need more data to be able to get a better value from the added subword information. We were able to successfully train on only a subset of the data that the authors in the original paper used. Furthermore, by adding the subword information and thus increasing the vocabulary and the size of the embedding matrix, the computational resources were an even more acute problem.

The results obtained and presented in Table 2 and Table 3 are coming from slightly different configurations. Some of the most notable differences come with using different a different amount of BPE merge operations, discussed in detail in

Dataset	SGNS	SGNS + BPE
WS353	75	56
RW	22	17
RG65	55	42

Dataset	SGNS	SGNS + BPE
WS353	31	17
RW	31	33
RG65	30	21

Table 2: **English datasets**. Presenting the correlation between human judgement and similarity scores on word similarity datasets. We trained models both only on the word-level, Skipgram with negative sampling(SGNS) and models that add the subword information to that base model using BPE segmentation. On the left is the Wikipedia dataset, on the right English UD Treebanks v 2.8

4.6.1 section. While the rest of the scores for the WS353 dataset and the RG65 dataset come from a low merge number of 30-50, the RW performance was only able to jump to a score of 17 once the merge numbers were taken to a higher extreme of 300-500.

The subword model proved more successful with a lower "EMBEDDING_DIM" of 250 comparing to the base model. We used double the amount of "NG_WORDS" to assist negative sampling when training the subword models, more on this in section 4.6.2. The "WINDOW_SIZE" was consistently 5, and the best learning rate 0.05. To avoid the loss issue discussed in section 4.1 we used a learning rate of 0.003 in some of the later experiments to be able to train longer. Most of the experiments were trained between 5-15 epochs.

Dataset	SGNS	SGNS + BPE
GRU65	54	41
GRU350	39	30
ZG222	31	23

Table 3: **German dataset**. Presenting the correlation between human judgement and similarity scores on word similarity datasets. Training same as in Table 2. Dataset used is the UD German Treebank v2.8.

4.6 Discussion

In this section, we will discuss the various effects of some of the hyperparameters have on our system, as well as some core questions about the system design that stem from the evaluation results presented in the previous section.

4.6.1 The effect of the number of merges in Byte Pair Encoding

Thu number of merges ("NUM_MERGES") is a hyperparameter that lets us pick the amount of time we would like to perform merge operations and join the existing ngrams. With each merge, the ngrams' frequency is evaluated and the ngrams then joined if needed, for each word, creating new larger ngrams. The frequency is relative to the current dataset.

We did many experiments with many different values of the the number of merges. The initial hyperparameter tuning that oriented itself by optimizing for the loss picked 50 as the ideal number(from a list we provided). Following, we used 50 for the majority of our early experiments. Once we implemented the various ways of measuring similarity, we questioned the values that the loss oriented hyperparameter tuning brought forward. One interesting perspective opened when we decided to expand these experiments to cover more extreme values, mainly to inspect the influence of this particular parameter on the whole system.

What we learned is also conceptually and linguistically interesting. If one uses low values, like 20 or 30 for this parameter, the ngrams, which start from separated characters, look still very sparse. There are very few ngrams larger than two characters, a lot of the letters stand for themselves or are paired with another. These type of division still gives good results, and specifically for the WS353 similarity dataset, which has the persistently the highest values anyway, even with just the base model.

Once the number of merges is taken to the other extreme, we tested between the values of 300-500, the subwords look slightly different. There are considerably more groups that look like more meaningful divisions, composed of 3-4 characters as well as smaller words that have no merged completely and have no subwords. The ngrams with the high number of merges look more like morphological meaningful units. This is also accompanied by a considerable jump in the accuracy for the RW, rare words, dataset, which linguistically would make sense. When compared to the approach in the original paper, the high merges resemble more the type of ngrams that would fit the 3-6 character divisions that the authors did.

Choosing the right value for the merges is also

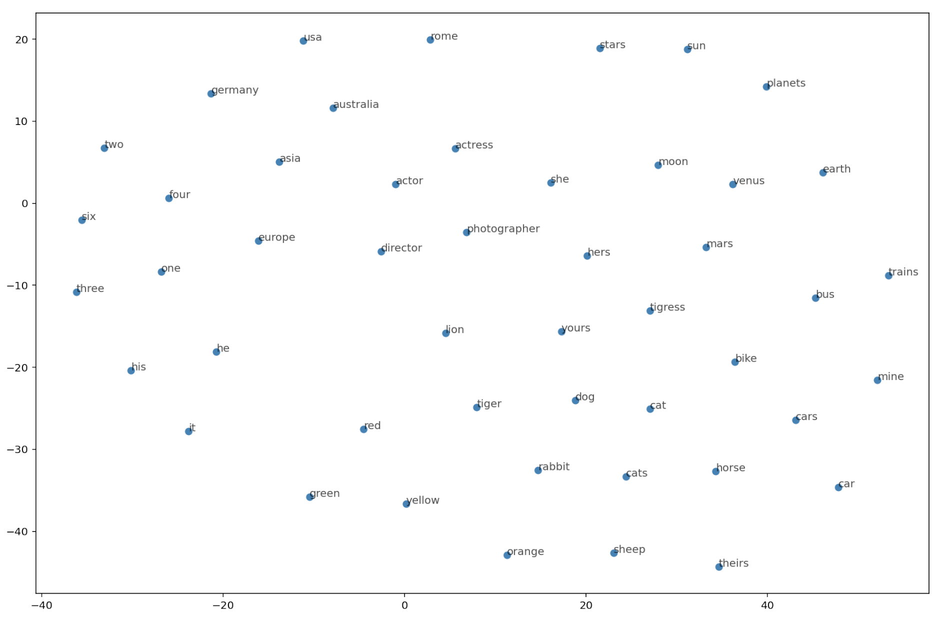

Figure 2: **Graphical representation of words close to each other by their cosine similarity.** Estimated on embeddings trained on the Wikipedia dataset. We can observe that the semantic groupings take shape, although the boundaries and separations are not as perfect.

dependent on the size of the dataset, and there are other considerations. The decision around the number of merges has a big impact on the systems and different systems have different needs (Ding et al., 2019).

4.6.2 The effect of the number of noise words for negative sampling

The noise words for the negative sampling loss are the randomly chosen words to contrast the meaning of the context word. We use a probability distribution to pick then. The hyperparameter for these words in our system is called ”NG_WORDS” meaning ”negative words”. We used a platform for hyperparameter tuning which returned different suggestions for both of the different models we have, the base model and the subword model.

The base model performs better with 5 negative words, and if more are added, the performance drops considerably. On the other hand, the performance is not that considerably hurt when the same is applied to the subword model.

We trained models that have both 10 and 5 negative words for the subword model. In our more recent experiments, the lower number of negative words had a slight positive effect on the similarity measurements.

As a conclusion, taken all of the the above into consideration, increasing the number of the negative words will not improve the system in most cases.

5 Future work and improvements

It has been a few years since the original paper (Bojanowski et al., 2017) has been published, so if we talk about future work, we talk about the future that has already happened. In the years that have passed, different architecture came to dominant the deep learning space (Reis et al., 2021), and with it, bring improvements to the existing ways of learning word representations.

The transformers architecture (Vaswani et al., 2017) gave rise to models like ELMO (Peters et al., 2018), BERT (Devlin et al., 2018) and GPT-2 (Radford et al., 2019).

The biggest improvement the transformers architecture brings is context - thus enabling context-dependent word representations. There is a limit of how much we can learn about the meaning of a word without taking it’s context in a sentence into consideration. The Skipgram model is taking each for independently, in a bag-of-words way, which means that the surrounding words don’t influence the meaning of the word. The context-dependent word representations help us go over this hurdle and obtain more meaningful embeddings.

When it comes to our implementation specifically, we would have loved to have the opportunity and time to implement both the n-gram subword division proposed originally in the paper and the Byte Pair Encoding algorithm. The comparison of the evaluation outcomes would have been interesting. Concerning ideas about extending the evaluation, applying the learned embeddings in tasks like language modeling would have been great for evaluating the results further.

6 Conclusion

The Skipgram model with the negative sampling addition is very powerful, can generalise and learn just by using word representations. Implementing subword information can be done in multiple ways, with their own drawbacks and advantages. Using Byte Pair Encoding to sample for subword information is one of those ways, and it is very effective as well. Picking the right number of merges for a Byte Pair Encoding implementation is very dataset, system and task dependent. Furthermore, when working with large values, whether negative or positive and their summation, various strategies need to be employed to be able to successfully train the model. And lastly, computational limitations are a real obstacle and we need to create systems that can work within those limitations.

This project was a great learning experience. We explored tooling, platforms, the computational and our own practical and conceptual knowledge limitations. We created two separate models, working with various success and explored the topic in and out to the best of our ability.

Acknowledgments

We would like to thank the Teaching Assistant Patrick Kahardipraja for being available for every question and doubt, and the lecturer Dr. Sharid Indhira Loáiciga Sánchez for her guidance and support.

References

TensorBoard: TensorFlow's visualization toolkit tensorboard description. https://www.tensorflow.org/tensorboard?hl=en. Accessed: 2021-08-16.

Weights & Biases home page. https://wandb.ai/site. Accessed: 2021-08-16.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Piotr Bojanowski, Armand Joulin, and Tomás Mikolov. 2015. Alternative structures for character-level rnns. *CoRR*, abs/1511.06303.

Kris Cao and Marek Rei. 2016. A joint model for word embedding and word morphology. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 18–26, Berlin, Germany. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Shuoyang Ding, Adithya Renduchintala, and Kevin Duh. 2019. A call for prudent choice of subword merge operations in neural machine translation. *Machine Translation Summit XVII*, page 204.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414.

Iryna Gurevych. 2005. Using the structure of a conceptual network in computing semantic relatedness. In *International conference on natural language processing*, pages 767–778. Springer.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2015. Character-aware neural language models. corr abs/1508.06615.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Maximilian Köper, Christian Scheible, and Sabine Schulte im Walde. Google semantic/syntactic analogy datasets german translation resources list. https://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/analogies//. Accessed: 2021-08-16.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramón Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics.

Minh-Thang Luong and Christopher D Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv:1604.00788*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Eduardo Souza Dos Reis, Cristiano André Da Costa, Diórgenes Eugênio Da Silveira, Rodrigo Simon Bavaresco, Rodrigo Da Rosa Righi, Jorge Luis Victória Barbosa, Rodolfo Stoffel Antunes, Márcio Miguel Gomes, and Gustavo Federizzi. 2021. Transformers aftermath: current research and rising trends. *Communications of the ACM*, 64(4):154–163.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers) 1715–1725 association for computational linguistics https://www. aclweb. org/anthology. *P16-1162*.

Charles Spearman. 1961. The proof and measurement of association between two things.

Henning Sperr, Jan Niehues, and Alex Waibel. 2013. Letter n-gram-based input encoding for continuous space language models. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pages 30–39, Sofia, Bulgaria. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Torsten Zesch and Iryna Gurevych. 2006. Automatically creating datasets for measures of semantic relatedness. In *Proceedings of the Workshop on Linguistic Distances*, pages 16–24, Sydney, Australia. Association for Computational Linguistics.