Implementing and Analysing a Method for Reducing the Gender Bias in an LSTM Language Model

Paula Heigl[1], Imogen Hüsing[1], and
Supervisor: Leon Schmid[1]

[1]Institute of Cognitive Science, University Osnabrück

April 1, 2023

Abstract

Gender Biases in Languages can have effects on people in real life. Since natural language model have become increasingly integrated into our lives, it is important that they do not perpetuate or even amplify these biases. For our final project in the course *Implementing Artificial Neural Networks with TensorFlow*, we looked at different techniques to implement debiasing for language models. We chose one approach that applied soft debiasing during training to re-implement in TensorFlow. The word embeddings obtained through training were then tested for remaining indirect gender bias. The results suggest an immense improvement in indirect bias in the language model but are not reliable, as we had to simplify the model and stop training early to accommodate the time frame for this project.

1 Introduction

The Austrian-English Philosopher Ludwig Wittgenstein coined the phrase "Die Grenzen meiner Sprache sind die Grenzen meiner Welt" ("The borders of my language are the borders of my world") (18). One aspect of language that this phrase has been applied to is gender bias which arise in many languages. There are direct biases such as these German examples *Krankenschwester*, *Hausfrau* or *Feuerwehrmann* and *Müllmann* that have a gendered word directly integrated, drawing direct associations between that descriptor and that gender. Furthermore, there are descriptors that have gender-specific versions like in the English language *actor* and *actress* or in the Italian language *dottore* and *dottoressa*. Here, the male term is often used as the general form. These direct biases aside,

we can also find more indirect biases in a language, like always pairing male or female pronouns with one specific descriptor, e.g. connecting male pronouns to jobs like scientist, doctor or manager and female pronouns to jobs like assistant or nurse (10).

If we always use male pronouns when talking about doctors, scientists or CEOs, we will internalize that men perform these jobs. This verbal association is reflected in the actual numbers if we look at the statistics for these job groups: of the 500 companies with the highest revenue (according to the magazine Fortune), only a little over 10% (53) are led by female CEOs (Hinchliffe). This shows that language represents also the negative sides of our society, and that this negatively impacts women's career aspirations (5). Reducing gender bias in our language can help reduce gender inequality. Alongside gender bias, many more biases are prevalent in our language, like racial bias or ableist bias. Examples are *blacklist* and *whitelist*, and "falling on deaf ears". Since it is difficult to examine all biases simultaneously, we chose gender bias as our focus. And because English papers about this topic using English text data are the most accessible for us, we decided to confine our project to gender bias in the English language.

One of the first criteria of artificial intelligence was the performance of a machine in a conversation. Would it fool the human into thinking they were speaking to another human? Turing introduced this criterion in 1950 in his book *Computing Machinery and Intelligence* (15). In the 1960s, there was a massive wave of enthusiasm about AI after the first chatbot ELIZA was released in 1966 because, at first, it seemed like ELIZA could pass the Turing Test. But the vocabulary of phrases ELIZA used was limited, and conversations entered a loop after a short while (16).

As language models become better and better at understanding and creating natural-sounding text, public interest in them is on the rise. They can be the foundation of chatbots in customer support, be used in text correction software like Grammarly or produce different varieties like the currently highly discussed Chat-GPT. Powerful language models that can produce seemingly natural texts need large data sets to train on. Popular data sets include the Google Blogger Corpus, the PennTreebank or Project Gutenberg. Notably, these data sets have not been cleaned with regard to gender biases. When left unattended, these big and important language models learn and reproduce these biases and can even amplify them. This can become problematic when we give Natural Language models decision-making tasks like résumé screening where they apply the learned stereotypes (3). At the very least, as researchers in machine learning, we are obliged to prevent machine learning models from amplifying the gender bias in our society. One attempt to mitigate these issues is to reduce the language model's biases.

1.1 Task-Description

For our final project for the course *Implementing Artificial Neural Networks with TensorFlow*, we want to look at different approaches for reducing gender bias in language models and compare their performance. We take a look at one approach in particular, which was implemented in PyTorch and presented in a paper by Bordia and Bowman in 2019 (2). For this project, we built a model with a similar structure and the same gender debiasing regularization but adjusted the implementation to fit TensorFlow conventions. The resulting word embeddings are extracted from the model and compared to the debiasing attempts of other papers using different evaluation tests that examine the indirect bias remaining in the language models introduced in a paper by Gonen and Goldberg in 2019 (6). In summary, we build a language model inspired by Bordia and Bowman that implements bias regularization and evaluate the resulting embeddings using the tests developed by Gonen and Goldberg.

2 Background

2.1 Quantifying Gender Bias

Most language models translate words into embeddings of high-dimensional vectors. One common approach is the word2vec approach in which each word is represented through one specific vector of n dimensions. Such a model can, for example, complete an analogy task such as completing the sentence "man is to woman as king is to x" the model can take the embeddings of the words *man*, *woman* and *king* and, by using simple arithmetics, find *queen* as the word whose distance to *king* is most aligned with the distance of *man* to *woman*. According to Bolukbasi et al. these simple arithmetic relations become a problem when the word embeddings of the model represent stereotypes. A demonstrating example is completing the analogy "Man is to Computer Programmer as Woman is to x" with *Homemaker* (1). To reduce the stereotypical answers for these analogy tests, Bolukbasi et al. tried to establish a gender subspace representing the bias. To identify this subspace, they calculated the distance vectors for ten gender pairs (these include she – *he*, *daughter* – *son*, *Mary* – *John*, etc.) and extracted their principal components (PCs) by Singular Value Decomposition. Their result showed that the first principal component was much more pronounced than the following, and they, therefore, assumed this first principal component g to be representative of the gender subspace. With this gender subspace, they then developed a term that calculates the bias in terms of strictness c. This constant can be adjusted depending on the needs of the algorithm, with $c = 0$ being the strictest option (1).

2.2 Debiasing after Training

For debiasing the embedding, Bolukbasi et al. proposed two different approaches: Hard-Debiasing which is done through Neutralize and Equalize, and Soft-Debiasing. Bordia and Bowman use the Soft-Debiasing approach in their model, which we re-implemented and evaluated in this project. Therefore, we will go into some more details of this debiasing method.

Hard debiasing eliminates the gender difference between gendered words completely, whereas soft debiasing only reduces the gender difference while maintaining most of the original embedding. In Hard-Debiasing, "Neutralize" refers to adjusting the embedding to ensure that gender-neutral words have a value of zero in the gender subspace."Equalize" means to equalize the distance of a gender pair to neutral words outside the gender subspace. To illustrate what this entails: after performing the step equalize, the pair *gal* and *guy* would be equidistant to the word *babysit*. When both Neutralize and Equalize are applied to an embedding, then they call it Hard-Debiasing (1).

Soft-Debiasing is a minimization problem with regard to a linear transformation that should affect the pairwise inner products of all word vectors as little as possible while simultaneously reducing the projection of the words onto the gender subspace as much as possible. A lambda is introduced to regulate how strongly the model should be debiased. A high value for lambda leads to the same result as the Hard-Debiasing method explained in the previous paragraph. After debiasing the model found more adequate answers for previously stereotypically answered gender-neutral analogies but was still able to give gendered answers for analogies where it was relevant (e.g. she is to ovarian cancer as he is to prostate cancer). Bolukbasi et al. conclude that their debiasing attempt was successful (1).

2.3 Debiasing during Training

While Bolukbasi et al. applied their debiasing strategy that removes gender information from gender-neutral words after training directly on the word embeddings, Zhao et al. found a flexible approach to apply during training and to easily modify to fit different models and different biases (19). In contrast to Bolukbasi et al. (1) and Bordia and Bowman (2), Zhao et al. did not use a word2vec approach for their model but rather the Global Vector approach of representing words in a model. Global Vectors (GloVe) were developed by Pennington et al. (2014). Instead of focusing on the local co-occurrences of words from the vocabulary that can negatively influence the recognition of multiple meanings of a word in the model, GloVes are created by taking the whole data set into account. This is achieved by creating a word count matrix which counts the co-occurrences between words and factorizing over this matrix. Is assumed that each dimension in GloVes represents some semantic meaning of the vector (14). The idea of Zhao et al. was to implement the debiasing of the word vectors during

training but mask some of the dimensions in order not to remove gender information that is relevant to the meaning of the word. The debiasing is implemented by introducing a minimization objective that tries to restrict the overspill of gender information from the masked dimensions into the to be neutralized dimensions (19). Through the tests Zhao et al. performed on their trained model, they showed that their debiasing method using Gender Neutral GloVes mostly kept the gender attribute the same and separate from other latent aspects, performed well at estimating word proximities on benchmark datasets and that it reduced gender bias in actual application (19).

2.4 Effectiveness of Gender Debiasing Schemes

Although both gender debiasing schemes performed well in the tests by their developers, they both rely on a simplified concept of bias as it assumes that the bias is merely represented through the position of a word on a gender axis (1; 19). Gonen and Goldberg criticize in their paper from 2019 that both debiasing attempts merely hide the bias, as it can still be found and easily reconstructed by looking at the distances of technically gender-neutral words that hold strong stereotypical gender associations. When examining what clusters the 500 most biased female and male words create, it became apparent that even after debiasing, they cluster into two clusters representing the respectively associated gender. The words forming the two clusters are not explicitly marked with their gender but rather socially. They also calculate the Pearson correlation for the words with the original bias measure before and after debiasing and for both cases only register a reduction of about 0.04 (6). In another test, Gonen and Goldberg focus on different professions and track the amount of female and male neighbors in the embedding before and after debiasing. Especially for the Zhao et al. model, the reduction in the Pearson correlation was again small (6).

Bordia et al. developed a debiasing method based on the Soft-Debiasing approach introduced in the Bolukbasi paper (1) but applies the debiasing to the loss during training, similar to the model introduced by Zhao et al. (19). In this project, we want to investigate the performance of this debiasing approach using the experiments from Gonen and Goldberg (6).

3 Methods

3.1 Model

Merity et al. from the company Salesforce developed the LSTM and QRNN Toolkit which provides code to create an LSTM (Long-Short-Term-Memory Cell), a GRU (Gated Recurrent Unit), or a QRNN (Quasi-Recurrent Neural Network) and made the code for this toolkit publicly available under the

BDS 3-Clause License[1] (11; 12). Bordia and Bowman used this toolkit to create a three layer LSTM. An LSTM layer does not only a time step as input, but also a hidden and cell state corresponding to short and long-term memory respectively. To the three layer LSTM they added the soft debiasing method introduced in the paper by Bolukbasi et al. in the form of a bias regularization function that was added to the loss. Regularization functions are generally added to the loss function to prevent parameter magnitudes and prevent overfitting. This bias regularization function is not applied to the loss to reduce overfitting, but instead the gender bias of the model, but the principle is the same. They take the matrix that contains the embeddings for the words W and the set of all gender pairs D to calculate the stack of all difference vectors C between the pairs. Performing singular value decomposition on this stack, the gender subspace B is defined as the first k columns of the right singular vector that represent at least 50% of the gender variance. A matrix' singular vectors are the unit eigenvectors corresponding to the singular values, which quantify the amount of variation or importance of the data in the matrix. With this subspace and the matrix of the embeddings that should be debiased, the regularization term is computed using this function $\mathcal{L}_B = \lambda \|NB\|_F^2$ where N is the set of gender-neutral words, F stands for the Frobenius norm, and λ is a tuning parameter that regularizes in how far the embedding is debiased. This regularization term gets added to the loss during training. The effect of this bias regularization is represented in Figure 1.

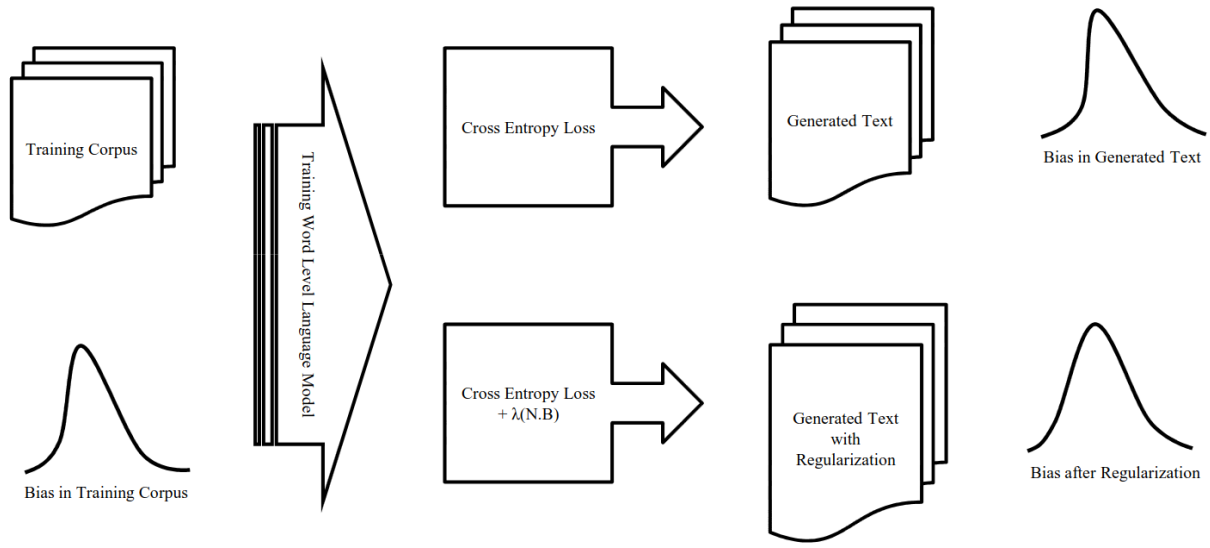

Figure 1: Bias Regularization is added to the Cross Entropy Loss during Training. Lambda controls the amount of regularization. Source: Bordia

The authors fail to specify clearly the optimizer they used in their paper or their code. However, we suspect they used Stochastic Gradient Descent, as it was given as a default value in the model's parameters and was shown to yield the best result for a word-level language model (17). Stochastic

[1]The code for the LSTM and QRNN Toolkit can be found under https://github.com/salesforce/awd-lstm-lm

Gradient Descent updates the model's parameters by subtracting the parameters' gradients multiplied by the given learning rate from the previous parameters. The weights and the learning rate of the optimizer decay during training.

3.2 Data

In their paper, Bordia et al. trained their model on three different data sets and compared the results. The first data set they used was the Penn Treebank. It consists of scientific abstracts, newspaper articles, book chapters, and the like. It contains over 4.5 million English words that have been tagged for Part-of-Speech (POS) by professional annotators. The POS tags help represent the grammatical behavior of a word in a text, which makes tagged datasets valuable tools for natural language processing (9). The Penn Treebank dataset was passed into the model with a batch size of 40 and the model, trained on the dataset for 750 epochs (2).

The second dataset Bordia and Bowman used is the WikiText-2 that includes approximately twice as many words as the Penn Treebank. The dataset consists of tokens taken from verified Wikipedia articles. The WikiText-2 model was also trained for 750 epochs but utilized a batch size of 80 (2).

The CNN/Dailymail Dataset consists of 300 thousand news articles written by CNN and The Daily Mail. The original version of the CNN/DailyMail dataset was created by Hermann et al. in 2015 and included over 1 million newspaper articles and some questions about key points of the articles. It was created to train of text comprehension models (7). The version used here is a smaller version that next to the queries also includes short summaries of the articles. As CNN is an American newsagent and the Daily Mail is British, the model includes British English and American English. Bordia et al. used subsampling with a factor of 100 to reduce the size of the dataset to make it manageable for experiments. As the CNN/DailyMail dataset was larger than the other two datasets, it was trained for only 500 epochs with a batch size of 80 (2).

For the bias calculation, defining sets were created for each corpus separately containing gender pairs due to vocabulary variations. While in the paper by Bolukbasi et al. a detailed description of the generation of the extensive list of defining sets was provided, we could not find more information on the methods with which Bordia and Bowman created their defining sets.

This chapter provided a more theoretical background for the code created for this project. The following chapter will show in detail how this theory was translated into TensorFlow code.

4 Implementation

The goal of our project was not to reimplement the model used by Bordia and Bowman one-to-one in TensorFlow, but rather to show how their debiasing method could be used in TensorFlow models and to evaluate the debiasing for capturing indirect gender bias. This allowed us to restructure some aspects of the code that did not translate into TensorFlow due to some inherent differences between the two libraries. In this chapter, we present the structure of our model[2] and the changes we decided to make in order to make the model follow the TensorFlow conventions we learned in this course. In the following subchapters, we use the same parameters used in the original PyTorch model to show how we would have run the model ideally. However, due to computational limitations, we had to change some parameters like the vocabulary size and the LSTM unit size. We will explain these changes in more detail in discussing of our results.

4.1 Overall Structure

The general structure of the program follows the structure of the code provided in the lectures and the homework of this course. It therefore differs in some parts quite heavily from the structure of the original PyTorch code. First, we load and preprocess the data in a data pipeline. We model class is created with all necessary layers, the call method, and a train and test step. The data and model get passed on to a training loop, which calls the train and test step. There, the forward pass is made using gradient tape, the loss is computed, and afterward backpropagated through the model. During backpropagation, the weights and biases of the model are updated. The *training_loop()* and the *train_step()* and *test_step()* functions are built exactly like the ones presented in the lecture, only with minor additions in the training loop like an early stopping condition and in the train step like the debiasing method.

4.2 Model

Overall, the model is implemented as a *tf.keras.Model* class that contains the already mentioned three LSTM layers with 1150 units except for the last layer which has as many as the embedding has output dimensions, in our case 400. The recurrent layers are preceded by an embedding layer and followed by a dense layer. In Figure 2 the model's general structure is shown. For counteracting overfitting and enhancing the model's performance, dropout is applied to the weights matrices and the output of those layers. Specifically, weights are dropped for the weights matrix of the embedding layer and the hidden weights matrices of the three LSTM layers. Furthermore, a dropout mask is applied to

[2]The code we wrote for this project can be found here: https://github.com/alinakrause/IANNeTF/tree/main/FinalProject

the outputs also of the embedding and the LSTM layers. The two types of dropouts are visualized in 3. In the following, we will elaborate on how the model was implemented by Bordia and Bowman, or originally by Salesforce, and explain how we adapted it to TensorFlow.

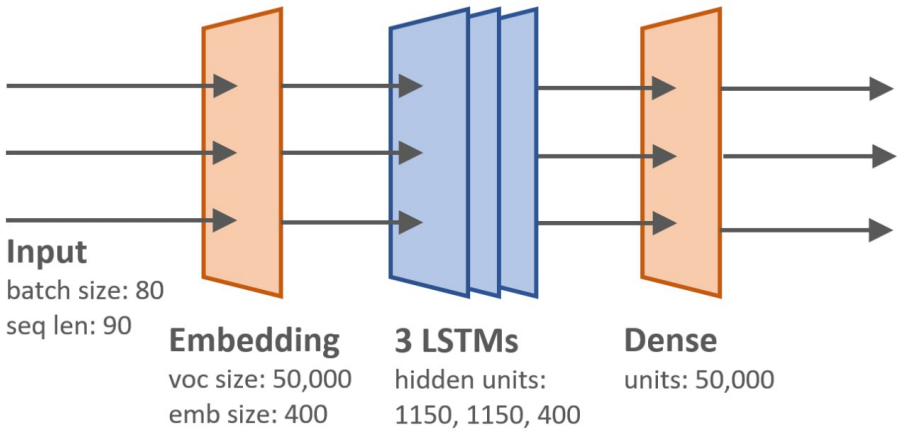

Figure 2: The overall structure of the model: visualizes the input and different layers, with its main parameters affecting the dimensions of the underlying matrices.

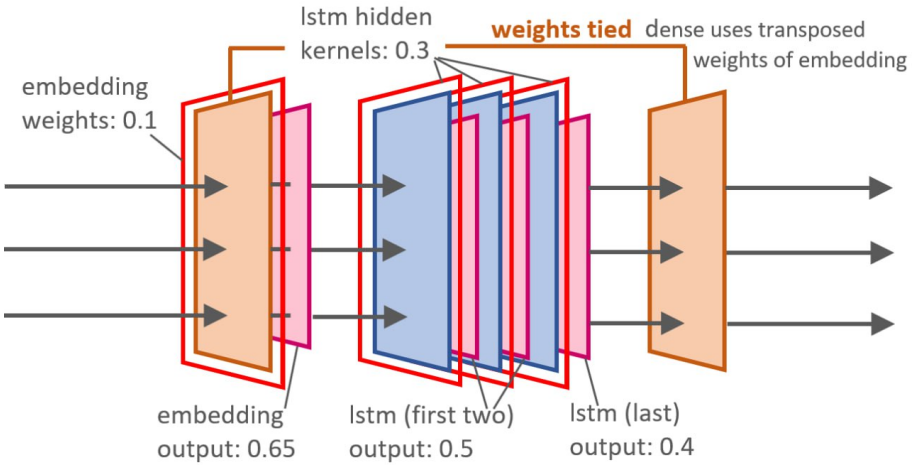

Figure 3: The more in-depth structure of the model: visualizes the dropout layers with its dropout rate. There are two types of dropout, one directly dropping weights of the weights matrices of the layers (red), and the applied to the output of a layer. Additionally, it shows the tied weights of embedding and dense layer (encoder and decoder).

In the PyTorch code, the weights dropout of the embedding layer is implemented via the method *embedded_dropout* that is called on the encoder object. They directly access the weights matrix of the embedding layer, apply a Bernoulli dropout mask, and then retrieve the word embeddings using these weights. But *get_weights()* and *set_weights()* do not run on the TensorFlow function graph, and accessing and changing a layer's weights from the outside is impossible in TensorFlow. Directly accessing the weights would also be needed for tying the weights of the embedding and the linear

layer. Therefore, we wrote a new layer class from scratch called *EncoderDecoder* which acts as an embedding and dense layer in one. In the embedding call, a dropout mask is generated and applied to the weights matrix, and the embeddings are generated. As mentioned, in the Salesforce model, the weights of the linear output layer and the embedding layer are tied (see Figure 3), meaning the dense layer uses the transposed weights of the embedding. In order to implement the tying weights, the custom *EncoderDecoder* layer contains a method *decode()*, which shares almost all of its functions with the call method of a dense layer with a softmax activation but uses the weight matrix of the layer that the embedding call also uses, only in *decode()* it is transposed.

In the Salesforce model, the dropout on the hidden weights matrices of the LSTM layers are realized as a wrapper class for each RNN layer called *WeightDrop()*. In this class, the hidden weights matrix is deleted, then set up again with a new matrix, which, even according to the authors, is a non-ideal, temporary solution. Afterward, a dropout mask is applied to the weights before calling the LSTM. We decided against writing this custom wrapper class but instead use the *recurrent_dropout* parameter of the keras LSTM layer that automatically applies dropout on the *recurrent_kernel* weight matrix. The dropout for the output of both the embedding and the LSTM layers is implemented in the Bordia code by a custom Layer class called *LockedDropout* that generates a Bernoulli-distributed dropout mask and applies it to the input that it gets. We directly translated this class into TensorFlow.

Our model's loss function is the cross-entropy. Originally, in the Salesforce code they wrote a custom split cross entropy loss which was prone to errors and improved the performance only marginally, according to Git Issues (octavian ganea). To simplify, we decided to use the TensorFlow native *SparseCategoricalCrossEntropy()*. Furthermore, the model gets an SGD (Stochastic Gradient Descend) optimizer. In the Salesforce model, they manually scale the learning rate down in the *train()* method relative to the sequence length. We implemented this with a learning rate scheduler with exponential decay. The initial learning rate is 30 and then stepwise decayed by a factor of 0.9.

4.3 Debiasing

The implementation of the bias regularization in our version is realized like in the original code by computing a bias regularization loss term. For this, it needs a defining set D containing all the gender pairs and a set N containing all the gender-neutral words. At the beginning of the training, these two sets are generated by calling the method *get_gender_pairs()*. The gender pairs are read from the provided file, tokenized, and added to the defining set if the words are within the set vocabulary size. The same happens to the remaining words in the corpus, which are added to the neutral set. These two sets get passed on to the bias regularization method. The following pseudocode illustrates how this method was implemented. An explanation of the underlying math is in chapter 3.1.

Pseudocode:

```
def bias_regularization_encoder(model, D (defining set), N (neutral set),
                                variance_ratio, lambda, normalization=True):
    get weights W from embedding layer -> shape of W:(voc_size, embed_feat)
    if normalization:
        normalize weights

    variable C for storing the gender pair difference vectors
    iterate over axis 0 of defining_set (over all gender pairs):
        get tokens of words in gender pair
        get word vector u for female token from embedding W
        get word vector v for female token from embedding W
        compute vector difference: (u - v)/2
        add difference to C
    -> shape of C: (number of gender pairs, embedding length)

    S, U, V = singular value decomposition of C
    -> shape of S: (number of gender pairs,)

    compute normalized cumulative sum of the squared elements of S
    filter out elements that are < variance_ratio
    find index k of smallest entry: k = index_of_min(filtered_cumul_var)
    -> for our D, S only one-dimensional, therefore k is always 0

    retrieve gender subspace B: get all rows and first k+1 columns of V

    get word vectors for neutral words (N) from embedding W
    get their projection onto gender subspace: word vectors @ B
    loss = lambda * squared L2 norm of projection tensor

    return loss
```

This bias regularization loss term is computed in the train step after the model has been called. It gets added to the calculated model loss before backpropagation. Similarly, the original authors compute an L2 activation regularization and a slowness activation loss term from the individual of the LSTM layers and add onto the loss. However, instead of manually calculating the regularization and for that using lists in the call and train step methods, we decided to use the *tensorflow.keras.regularizer.L1L2* which we passed to each LSTM layer upon instantiation.

4.4 Data Preprocessing

Because of the limited time and computing ressources that we had available for this project, we decided to use only the CNN/DailyMail datatset as it is the largest corpus and the ratio of female and male-gendered words is more balanced compared to the first two corpora (2). We downloaded the data from (Cho). The dataset came with all articles being stored in separate STORY files. To get the articles into a format that is easier to handle in the preprocessing, we wrote a script to extract the articles from the STORY files and to save them all in one cumulative text file. Later, when we tried to actually train the model, we had to resort to using only the CNN articles because training on the whole dataset exceeded the RAM that we had available.

In the original paper, the data is batched before the training starts, but the generation of the sequences is part of the $train()$ function. The sequencing of the already batched training data happens inside the training loop, in addition to the forward step, the loss calculation and the backward step. This is done with a while-loop, iterating over the batches. The sequence length is randomized using a normal distribution, with the backpropagation through time variable ($bptt$) indicating the center of the distribution and the standard deviation being initialized with 5. In about 5 percent of the sequences, the $bptt$ variable is halved. The sequence length had a minimum value of 5 to prevent too short or even negative sequences, but it still allows extremely long sequences in rare cases. With the sequence length, the batch is sequenced. The learning rate is also adjusted using the ratio of the sequence length to $bptt$. The model is then called with the newly sequenced batch and the new learning rate, and the raw loss is calculated. As already mentioned, to the raw loss the L2 and the slowness regularizer, and the bias regularization are added before the loss is propagated back through the model to update the weights. Afterward, it goes into the next batch, and the sequencing process starts again.

For our code, we decided to separate the sequencing from the rest of the training. In PyTorch, a batch has the structure (time steps, batches), which makes it possible to do the sequencing as described above. As the dimensions are flipped in TensorFlow (batches, time steps), it is not possible to directly translate the sequencing. To stick to the TensorFlow conventions taught in this course,

we decided to move the sequencing into the data preprocessing before training.

To preprocess the data, it is read line by line from a text file and tokenized using *tf.keras.preprocessing.text.Tokenizer* with vocabulary size set to 50,000. We split the tokenized data into train, validation, and test sets according to the commonly used 0.6-0.2-0.2 ratio. Iterating over the lists of tokens, it first generates a random sequence length as described above. Even though the original code did not implement an upper limit for the sequence length, we added it to prevent overly long sequences and therefore a slower training. Then, one sequence of that length is sliced off the tokens list. The sequence length generation and slicing are repeated until all data is sequenced. The sequences are padded to create sequences of equal length for batching. Afterward, the dataset is created from the sequences and split into input and target. In the end, the dataset is shuffled, batched, and prefetched. The data was not shuffled in the original code, but since this is important to prevent overfitting and for the model to remain general, we decided to include it. For further tests, comparing the difference between the shuffled and the not-shuffled data set would be interesting.

4.5 Training

In the training loop of our code, for each epoch, it iterates over all batches in the training data and performs the train step on it. As mentioned above, the actual training happens in the train step defined in the model. In the train step, we use a gradient tape to call the model and generate the predictions from the inputs. The loss is calculated with the predictions and targets corresponding to the inputs and the bias regularization added. Afterward, the metrics are updated. Every 500 batches, the class *SerializeEmbedding()* saves the weights matrix of the embedding layer. At the end of each epoch, the model is evaluated by calling the model on the validation data and calculating the loss (without the bias regularization). Before entering the next epoch, the code checks whether the conditions for early stopping are met and ends the training if they are.

The embeddings that have been serialized during the training are used to evaluate the model's ability to filter out gender bias while learning with a language corpus. We perform the experiments from Gonen and Goldberg (6) to specifically test whether the debiasing method also captures the indirect gender bias. We will not go into the details of the implementation of these experiments since we used the code for the most part as it was provided by the authors of the paper and adjusted it only slightly to fit our embeddings[3]. In the next chapter, we will explain the experiments and our results.

[3]The code for the tests can be found here: https://github.com/gonenhila/gender$_b ias_l ipstick$

5 Evaluation of the Model

Gonen and Goldberg conducted five experiments on the word embeddings of Bolukbasi et al.(1) and Zhao et al.(19) with and without debiasing. The experiments use two different bias definitions, bias-by-projections, and bias-by-neighbors. According to the first definition, the bias is calculated from the difference of the projection of a target word vector onto the word vector of the gendered word *he* and its projection onto the *she* vector. Both Bolukbasi et al., Zhao et al., and later Bordia et al., use this definition for their debiasing method. For the second definition, proposed by Gonen and Goldberg, the bias is obtained by looking at the counts of male and female-biased words (according to the previous definition) among the k nearest neighbors of the target word.

5.1 Tests

The first experiment is a clustering experiment. The 500 most female and 500 most male-biased words before debiasing according to the bias-by-projection definition are extracted. Using *KMeans* and *TSNE* from the *sklearn* library, the word vectors of those words get clustered into two clusters, once for the biased word embeddings, and also for the debiased embeddings. Afterward, the accuracy of the clustering is calculated.

The second experiment measures the Pearson correlation of the counts of male neighbors (bias-by-neighbors) for all words in the vocabulary and their bias-by-projection score of the original, biased embeddings. Again, they do this once for the neighbor counts of the debiased words and also for the original, biased once for comparison.

Next, they measure the same correlations, but look only at the word embeddings of words for professions, retrieved from the list of professions used in Bolukbasi et al.(1) and Zhao et al.(19). For visualization purposes, they plot the professions and their bias-by-neighbors counts before and after debiasing.

The fourth experiment replicates the gender related association experiments conducted by Caliskan et al.(3). The tests evaluate the associations between female and male names and certain groups of words like arts and mathematics or family and career. Since the word corpus we used for our training does not include all words that are used in the experiments, we excluded this experiment from our measurements, and therefore, we will not go into further detail explaining the procedure.

The last experiment Gonen and Goldberg conducted is classifying the 5,000 most biased words according to the bias-by-projection definition with an RBF-kernel SVM classifier. First, it is trained on 500 randomly chosen word embeddings, each of the female and male most biased words. After training, its generalization accuracy is evaluated on the remaining 4,000 words' embeddings.

Debiasing Method	Experiment 1		Experiment 2		Experiment 3		Experiment 5	
	Cluster Acuracy		**Correlation btw. Bias Definitions**		**Correlation for Professions**		**Classifier Accuracy**	
	debiased	original	**debiased**	original	**debiased**	original	**debiased**	original
Zhao	**85.6%**	100%	**0.736**	0.773	**0.792,**	0.820	**96.25%**	98.65%
Bolukbasi	**92.5%**	99.9%	**0.686**	0.741	**0.606**	0.747	**88.88%**	98.25%
Bordia	**49.8%**	100%	**-0.009**	0.743	**0.158**	0.770	**50.43%**	99.4%

Table 1: Comparision of the results of the Gonen and Goldberg experiments (except Experiment 4) for the three debiasing methods: The in-training GN-GloVe model by Zhao et al., the post-processing hard-debiasing method by Bolukbasi et al., and the in-training soft-debiasing method implemented by us but originally by Bordia et al.. The measurements for Zhao and Bolukbasi are taken from Gonen et al.(6), Bordia measurements are obtained by us.

5.2 Results and Interpretation

hyperref Due to computational limitations, we had to restrict the training process of our model. The data corpus, the model's hyperparameters, and all the training variables were described in chapters 4 and 5 as we intended them for training this model. Even though we already tried several attempts to make it work, a Lenovo Legion 5 with a Geforce GTX 2060 and 16 GB RAM was not strong enough to get the training running. Therefore, we had to resort to Colab Pro. Even then, we had to limit the training to get the embeddings within reasonable time to evaluate them within the scope of this project. We trained the embeddings we used for these experiments only on the CNN articles and not the whole data, which reduces the number of unique words in the corpus from 4 million to 1.2 million. We restricted the vocabulary size to 10,000 words and set the number of hidden units of the LSTM layers (except for the last, which was still 400) to 756 instead of 1150. The model was trained on the data for around 10 hours, once with the bias regularization enabled and once without it. Because of these restrictions, we had to put on the training due to the time and computational limitations, and the results of the experiments on the word embeddings are unfortunately not significant.

The results of our experiments are presented in Table 1. The plotted clusters of experiment 1 are shown in Figure 4. Before debiasing the clusters are split almost evenly in the middle, while for the debiased version the elements of the clusters are much more intermixed. For experiment 2, we get a Pearson correlation of -0.009 with a p-value of 0.356 for the debiased embeddings (compared to 0.743 and p-value=0.0 for the biased embeddings). Figure 5 plots the profession from experiment 3, and the change in gender bias with and without debiasing according to the bias-by-neighbors definition. The Pearson correlation of this measurement is 0.158 and a p-value of 0.096 (compared to 0.770, p-value < 3.5e-23). And in the final classifier experiment, the accuracy drops from 99.4 to 50.4%.

The results of our experiments are ambivalent. On one hand, they seem to reveal a significant

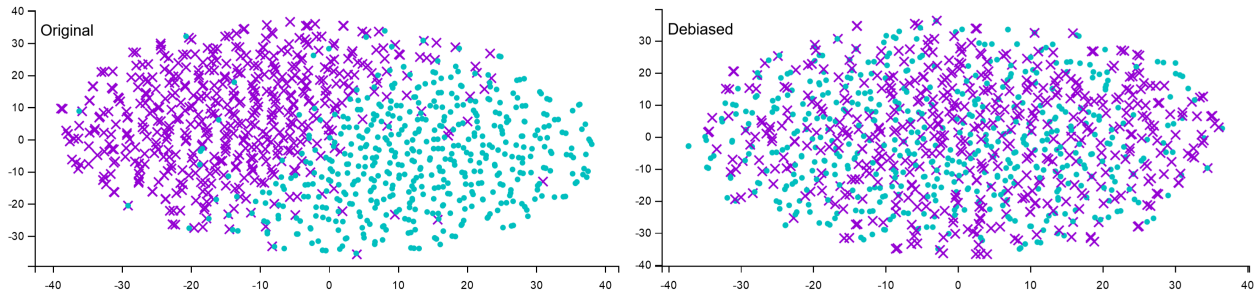

Figure 4: Clustering the most 1,000 biased words, without (left) and with debiasing (right).

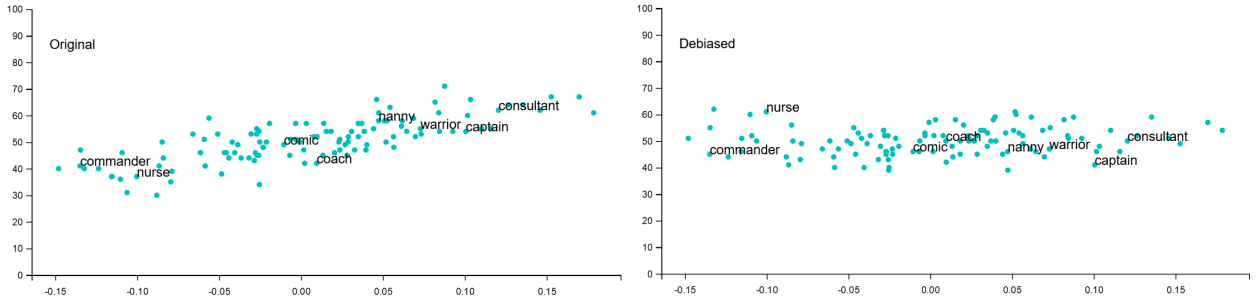

Figure 5: The number of male neighbors for the professions with (left and without debiasing (right). Axis X is the bias-by-projection from the biased embeddings and axis Y is the number of male neighbors (bias-by-neighbor definition) of the biased embeddings (left) and the debiased embeddings (right). A limited number of professions is shown on the plot to make it readable.

reduction of the indirect gender bias in our debiasing language model. A cluster and classifier accuracy of around 50%, indicates that the gender bias in the embeddings could be removed almost entirely. Moreover, both debiased Pearson correlations of the general the professions experiment have decreased compared to the original, biased ones. However, the p-values of those Pearson correlations are all >0.05, therefore they fail to be significant. But as already mentioned, the results of all experiments are unreliable overall because the training of the embeddings was not done as intended. Re-doing the experiments with properly trained word embeddings could yield more convincing results. Possibly there is an actual significant improvement of indirect gender bias of the in-training debiasing method by Bordia et al. (2) compared to the post-processing method by Bolukbasi et al. (1).

6 Conclusion

For the final project of the course *Implementing Artificial Neural Networks with TensorFlow* we implemented an LSTM language model with a bias regularization function based on an already existing model by Bordia and Bowman (2). We tested the word embeddings received through training the model for remaining indirect bias. The test results suggest great improvement in removing indirect

gender bias of the word embeddings but should be regarded with caution as the training could not be performed as intended. Further work could include completing the training as described in this term paper and evaluate the final embeddings using the tests proposed by Gonen and Goldberg (6). If the results of the tests remain similar to the results we got from the simplified version, this debiasing method is a significant improvement to the other techniques summarized in this term paper. Secondly, since some changes were made to the model structure in this TensorFlow version, the hyperparameters should be fine-tuned to get optimal model performance. Furthermore, it would be interesting to compare the embeddings of the original PyTorch version to the embeddings of this to TensorFlow adapted version to check if our adaptions resulted in relevant differences in the model.

References

[1] Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., and Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems*, 29.

[2] Bordia, S. and Bowman, S. R. (2019). Identifying and reducing gender bias in word-level language models. *arXiv preprint arXiv:1904.03035*.

[3] Caliskan, A., Bryson, J. J., and Narayanan, A. (2017). Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186.

[Cho] Cho, K. Deepmind qa dataset.

[5] Correll, S. J. (2004). Constraints into preferences: Gender, status, and emerging career aspirations. *American sociological review*, 69(1):93–113.

[6] Gonen, H. and Goldberg, Y. (2019). Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. *arXiv preprint arXiv:1903.03862*.

[7] Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28.

[Hinchliffe] Hinchliffe, E. Women ceos run more than 10% of fortune 500 companies for the first time in history.

[9] Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank.

[10] Menegatti, M. and Rubini, M. (2017). Gender bias and sexism in language. In *Oxford research encyclopedia of communication*.

[11] Merity, S., Keskar, N. S., and Socher, R. (2017). Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*.

[12] Merity, S., Keskar, N. S., and Socher, R. (2018). An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*.

[octavian ganea] octavian ganea. awd-lstm-lm git issue 62.

[14] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[15] Turing, A. M. (2009). *Computing machinery and intelligence*. Springer.

[16] Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

[17] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, 30.

[18] Wittgenstein, L. J. J. (1971). Prototractatus: an early version of tractatus logico-philosophicus.

[19] Zhao, J., Zhou, Y., Li, Z., Wang, W., and Chang, K.-W. (2018). Learning gender-neutral word embeddings. *arXiv preprint arXiv:1809.01496*.

Appendix

Meeting Protocols

Meeting 1: March 8th, 2023

- Instructions for how to arrange meetings in the future: rather via sending a calender invite, not mail
- We presented the idea for our project: to translate the PyTorch model by Bordia and Bowman with their in-training debiasing method to TensorFlow, and afterward, apply the experiments by Gonen and Goldberg on the trained embeddings to measure the debiasing method's ability of also capturing an indirect gender bias.
- Is the training of the language model possible with our provided computational resources or at university/IKW?
  - → Yes, LSTM language model is doable with Imogen's notebook resources.
  - → There are computers at the IKW which we could use, but difficulties with setting everything up can arise.
- Translation from PyTorch to Tensorflow should not be too complicated, since we don't use a pre-trained model and don't have to transfer any objects.

Meeting 2: March 13th, 2023

- We were confused about sequencing in train() method in the PyTorch code and what the methods batchify() and getbatch() exactly do.
  - → In PyTorch, the dimensions for time steps and batches are switched, meaning it's (time steps, batches) not (batches, time steps) like it is in Tensorflow. Spaghetti cutting analogy.
  - → batchify() turns data in into batches which is called before the training loop, getbatch() creates the sequences which happens during the training.
- How should we do the batching and sequencing, especially the random sequence lengths that are generated while training?
  - → Make a proper TensorFlow data pipeline, just like we did it in the homework. Keep the randomization of the sequence lengths, but do it all before the actual training.

- In general, should we try to translate the PyTorch code "1-zu-1" or rather keep the structure and details and implement the model in TensorFlow more freely?

  → The latter, stick to the TensorFlow conventions we learned in the course and only keep the important structures of the network intact (e.g. the randomization of the sequence length).

  → Ideally, in the end, compare the PyTorch and the TensorFlow model using a subset of the data, overall and with respect to changes we made to the implementation. E.g. in the original code they do not shuffle, so we could compare it with and without shuffling and evaluate any differences. But this is not mandatory.

Meeting 3: March 21st, 2023

- Should we make changes that seem logical, but are not part of the original PyTorch code?

  → Shuffle the data: Ideally, run it with and without shuffling and compare the results. Otherwise, for a reimplementation study, do it like the original, but note that usually it would and should be included.

  → Upper bound for the random sequence length generation: Yes.

  → Add missing gender pairs (e.g. Mr, Mrs) to the list: No, use the original list, but maybe look up the origins or generation process of that list.

- In PyTorch, setting the gradients to zero with optimizer.zero-grad(), does not specifically have to be implemented in TensorFlow, does it?

  → No, because gradients and backpropagation works differently in TensorFlow.

Meeting 4: March 24th, 2023

- @tf.function / graph mode does not work yet, because we use non-TensorFlow-things inside the train/test step. The custom dropout method that they use in the PyTorch code accesses and changes the weights of the embedding layer, but get and setweights() uses numpy arrays. And also tying the weights between the dense output layer and the embedding layer is a problem. We don't know how exactly to make the code purely TensorFlow-y or what's the best option to do so.

  → Write a custom embeddings layer from scratch that has the basic functionalities of an embedding layer but applies a dropout mask to the weights before computing the embedding

output for the given input. And this layer also has a dense layer function which uses the same weights, but transposed. Explanation of how to implement such a custom layer.

- Warning message, that the LSTM layers don't have gradients.

  → The reason for that are the lists in the call and train/test step method for hidden states, cell states and outputs. Replace absolutely necessary lists with tf.TensorArray. But the lists mostly should not be necessary, e.g. the activation and slowness regularization can be replaced by implementing a tensorflow.keras.regularizers.Regularizer, specifically the L1L2 one since the activation regularization seems to be a L2 regularizer and the slowness regularization a L1 regualrizer.

Meeting 5: March 27th, 2023

- Problem: Imogen's notebook is too weak for the training, after all. We have to do it in Colab now, but the training the way we planned to do it is way too slow.

  → We can change the model, specifically reduce the LSTM units, the vocabulary size and maybe even the number of LSTM layers, in order to get some training done, but we have to state the changes in the report.

  → Offer: train the model on Leon's computer, but only overnight (28.-29.03.). And we could get an extension of the deadline up to 5 days, but we should notify Leon in advance.