Binarizing Word Embeddings using Straight-through estimators

Bachelor's Project Thesis

Robin Entjes, s2526883, r.entjes@student.rug.nl,
Supervisors: W. Mostard & M. Wiering

Abstract: Word embeddings are usually represented as real-valued vectors that contain semantic and syntactic information of a word. Words that are semantically similar have similar word embeddings. However, there are some downsides to using real-valued word embeddings. The calculations that are needed are computationally expensive and they require a large amount of memory. Therefore, we investigate the possibility of transforming real-valued word embeddings into binary-valued word embeddings. In this research we compare two different methods: an autoencoder that makes use of the Heaviside function and one autoencoder that was further extended with straight-through estimators. The two methods are compared using several standard word similarity tasks. Similar results are obtained for both binarization methods. However, the autoencoder using the straight-through estimators performed significantly better in the case of the SimLex dataset. It seems that it is possible to binarize real-valued embeddings without a great loss of semantic information.

1 Introduction

For a number of years now, word embeddings are getting more and more popular. They are useful in multiple downstream natural language processing tasks, such as word similarity tasks (Mikolov, Chen, Corrado, & Dean, 2013) and for classifying documents (Ahmad & Amin, 2016).
A word embedding is a numerical vector representation of a word. The vector contains semantic and syntactic information about a given word. Thus, if two words have word embeddings that are close to each other in the vector space, that would suggest that the words are similar in meaning. For example, if we consider the word embeddings for *car* and *automobile*, we can expect the cosine similarity between them to be higher than the cosine similarity between *car* and *fish*. Traditionally, these word embeddings are represented as real-valued vectors. However, there are a number of problems that come with representing these vectors with floating point numbers. The first problem is that it takes a significant amount of memory. If you for example would like to use it on phones, having a large vocabulary of words, it would require a significant chunk of memory. The second problem is that performing calculations on these word embeddings are computationally expensive, because it makes use of the cosine similarity metric, which involves the dot product. Since these computations are quite computationally expensive, we consider a different representation, namely binary valued vectors. The advantage of this approach is that we require a significantly smaller amount of memory. Also, if we wish to measure the similarity between two binary word embeddings, we can make use of the Hamming distance, which involves the bitwise XOR operator. This is a simple CPU instruction that is significantly faster than the dot product. So we can conclude from this that using binary word embeddings is something that can be quite useful.
Therefore, we try to transform the real-valued vectors into binary-valued vectors. We call this process binarization. By binarizing these word embeddings the memory needed and the computation time for calculating the similarity between two word embeddings are both greatly reduced. For the binarization of the embeddings we will make use of autoencoders. In this thesis two different methods will be compared.

In previous research there has been a method proposed for generating binary word embeddings which makes use of an autoencoder (Tissier, Gravier, & Habrard, 2019). An autoencoder is a type of neural network, where the goal is to compress the input into a smaller representation without a great loss of information (Goodfellow, Bengio, & Courville, 2016). The standard autoencoder is extended with the Heaviside step function to binarize the word embeddings. This comes with the problem that proper back-propagation is not possible, since the step function is non-differentiable. To overcome this problem, we will make use of a straight-through estimator (STE) (Bengio, Léonard, & Courville, 2013). This is a method to bypass the Heaviside step function and allows proper back-propagation. We will generate two sets of word embeddings. One set will be generated using the autoencoder without STE, and one set will be generated using the autoencoder with STE. Both autoencoders will be compared by testing the performance of both sets in a word-similarity task.

In this thesis we will hope to answer: Do binary word embeddings generated by an autoencoder using straight-through estimators have a higher performance in a word similarity task in comparison to binary word embeddings that are generated by an autoencoder without straight-through estimators?

In section 2 we will describe the workings of the autoencoder and the straight-through estimators. Furthermore, we will take a look at how GloVe is trained. GloVe is a model for training real-valued word embeddings. In section 3 we describe the hyperparameters for the autoencoders and also how both models will be evaluated. In section 4 we will describe the results, in section 5 we discuss the results and in section 6 we will have the conclusions.

2 Background

2.1 Autoencoder

To binarize the word embeddings we make use of an autoencoder, which is a form of a neural network (Kramer, 1991). It consists of one hidden layer h, an input layer x and an output layer x'. The autoencoder consists of two stages; the encoder and the decoder. The encoder transforms x into h, and the decoder transforms h into x'.The use of an autoencoder comes from the fact that the hidden layer typically has a smaller dimension than the input layer. Since the hidden layer has a smaller dimension than the input layer, it is forced to capture the most informative factors in the latent space (Goodfellow et al., 2016). In the case of an autoencoder we are not interested in the output of the model, but rather the vector from the hidden layer, since it is a more compact representation of the input. Therefore, such models can help finding a more compact representation of the input, without a great loss of information.

To enable this method to binarize word embeddings, we make use of the Heaviside step function within the autoencoder:

$$Heaviside(x) = \begin{cases} 0 & \text{if } x < \theta \\ 1 & \text{if } x \geq \theta \end{cases} \tag{2.1}$$

Where θ is the threshold, which will be set to 0 in this thesis. Using the step function leads to a different problem, since the function is non-differentiable. This makes it impossible to perform proper back-propagation. A workaround for this is to set the output weight matrix equal to the transpose of the input weight matrix (Tissier et al., 2019). This way, during training, the output weight matrix can be trained using back-propagation and the input weight matrix will be set to the transpose of the output weight matrix. This way it can still be implicitly trained. Now that we have a workaround we can look at how the binary word embeddings are generated. For that we will make use of the following formula:

$$b_i = h(W \cdot x_i^T) \tag{2.2}$$

In which b_i is the binary word embedding, x_i is a real-valued word embedding from the input, W is the weight matrix, and h is the Heaviside step function. This part is what would be called the encoder, since it transforms the original real-valued embedding into the binary word embedding. For the decoder, the following formula is used:

$$x_i' = tanh(W^T \cdot b_i + c) \tag{2.3}$$

Here x_i' represents the reconstructed vector, c represents the bias. And tanh is the hyperbolic tangent function, which is used to make sure that the values in the reconstructed vector are within the range of -1 and 1.

2.1.1 Objective Function

The objective function that is used to optimize the model contains two parts: a reconstruction loss and a regularization loss. These are defined as following:

$$l_{rec}(x_i) = \frac{1}{m}\sum_{k=1}^{m}(x_{i_k} - \hat{y}_{i_k})^2 \tag{2.4}$$

Here x_{i_k} represents attribute number k of vector x_i. The reconstruction loss is the mean squared error of the difference between the input vector and the reconstructed vector. Using solely the reconstruction loss will lead to good reconstructed vectors, but the generated binary word embeddings are not very well. This is because W was favoring the reconstruction, and therefore much similarity information was discarded. To deal with this, the regularization loss is introduced:

$$l_{reg} = \frac{1}{2}||W^T W - I||^2 \tag{2.5}$$

I represents the identity matrix. This term is used to minimize the correlations between the different features of the latent binary representations. The total objective function becomes:

$$\sum_{x_i \in X} l_{rec}(x_i) + \lambda_{reg} l_{reg} \tag{2.6}$$

Here λ_{reg} is a hyperparameter which can be tweaked if necessary. It is used to give more or less importance to the regularization loss. The default is 1.

2.2 Autoencoder with Straight-through Estimators

The previous method was able to achieve near-lossless performance in comparison to the original real-valued embeddings (Tissier et al., 2019). However, it seems to have its limitations. Adding extra layers into the model would not be possible since the decoder step has to be the exact transpose of the encoder step. This is because the transpose of W is used in the decoder. Also the quantization error could be quite high, which can lead to a loss of quality of the obtained embeddings for search applications (Mena & Nanculef, 2019). The autoencoder proposed in this thesis makes use of an adjustment, such that we can apply back-propagation in our model. This is done by making use of straight-through estimators (Bengio et al., 2013). The working can be found in Figure 2.1. During the forward pass, we still make use of the step function. However, during back-propagation, we will use the straight-through estimator. A straight-through estimator is a gradient, that will bypass the original gradient. During the backward pass the incoming gradient from the right is defined as following:

$$g_Y = \frac{\partial L}{\partial Y} \tag{2.7}$$

The derivative that we need is:

$$g_Z = \frac{\partial L}{\partial Z} \tag{2.8}$$

To bypass the Heaviside step function, we define the gradient as:

$$g_Z = g_Y \ .* \ 1_{|Z| \leq 1} \tag{2.9}$$

Here $.*$ is element wise multiplication (Courbariaux, Hubara, Soudry, El-Yaniv, & Bengio, 2016). In this function we make use of the indicator function:

$$1_A(x) := \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A. \end{cases} \tag{2.10}$$

The use of the indicator function, means that when the absolute value of Z is greater than 1, it will be set to 0, otherwise the indicator function will be equal to 1. So basically the gradient bypasses the step function and uses a different function instead. The gradient g_Z is what we call the straight-through estimator. This function is basically the same as the hardtanh function:

$$hardtanh(x) = \begin{cases} -1 & \text{if } x < -1 \\ 1 & \text{if } x > 1 \\ x & \text{otherwise} \end{cases} \tag{2.11}$$

This is the function that will be used in the code. The complete autoencoder with the straight-through estimator implemented can be found in Figure 2.2. We see that during the backward pass, the hardtanh function is used.

2.3 GloVe

For the input of both autoencoders GloVe is used (Pennington, Socher, & Manning, 2014). GloVe

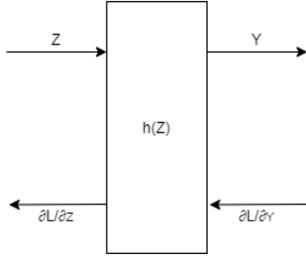

Figure 2.1: An image showing the workings of the straight-through estimator. h is the Heaviside step function. Z and Y are the vectors that are passed during the forward pass. During the backward pass, $\frac{\partial L}{\partial Z}$ and $\frac{\partial L}{\partial Y}$ are the gradients.

is a model which is used for producing word vectors. There are a number of pre-trained sets available. One of these sets, which is trained on Wikipedia2014 and Gigaword 5, will be used as the input for the autoencoders.
The word vectors in GloVe are trained based on co-occurrence of words within a text. It scans over a text using a context window. For every word in the text, the ten words to the left and ten words to the right are determined. In a matrix of co-occurrence all co-occurrences will be registered. Each co-occurrence will be weighted based on distance between two words using $1/d$, where d is the distance. So if a word is five places removed within a text, 1/5 will be added in the co-occurrence matrix.
To determine the word vectors, a function is wanted that can encapsulate the notion of co-occurrence. So a function of the following form is needed:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{2.12}$$

Here P_{ik} and P_{jk} are the probabilities of words i and k, and words j and k, respectively, co-occurring. w_i, w_j and $\tilde{w}_k$ are word vectors. A model which holds this property is as follows:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - logX_{ij})^2 \tag{2.13}$$

Here V is the size of the vocabulary, which in this case is 400,000. X_{ij} is matrix of word-word co-occurrence, whose entries denote the number of times word j occurs in the context of word i. b_i and $\tilde{b}_j$ are a bias, that is added for each word. $f(X_{ij})$ is a weighting function, which is used to make sure that rare and frequent co-occurrences are not over-weighted.

3 Methods

The two models both use the same setup. The model uses the entire GloVe database of 400,000 words as the input for training. These embeddings are pre-trained and have a dimension of 300. Three different sizes of binary embeddings are produced: 128, 256 and 512. These sizes are used, because the memory size needed for these embeddings are smaller than the original embeddings. The number of epochs that are used is set to 25. The learning rate for the model is 0.001 and the batch size that is used is 75. The hyperparameter λ_{reg} is held on 1.

3.1 Evaluation

For the evaluation of the trained embeddings we make use of a number of word similarity tasks. For this task we present the model with a number of word-pairs. These word-pairs have a score for how similar they are, according to human evaluation. With our trained binary embeddings we make use of the Hamming similarity to determine how similar the word pairs are according to our trained embeddings. The closer the Hamming similarity is to 1, the more similar the words should be. This measure can be used, since we make use of binary valued vectors. And since they are binary we can count the number of bits which are not the same. This is the same as using the XOR operator. Keeping this in mind, we get the following formula:

$$HammingSim(X, Y) = 1 - \frac{1}{N}\sum_{k=1}^{N} |X_k - Y_k| \tag{3.1}$$

Here X and Y represent the two word embeddings to be compared, X_k and Y_k are the values of the vector in position k, and N is the length of the vectors. The value of the similarity lies between 0 and 1. The higher the value is, the more similar the two words are. To determine the similarity of real-valued vectors, the cosine similarity is used, which

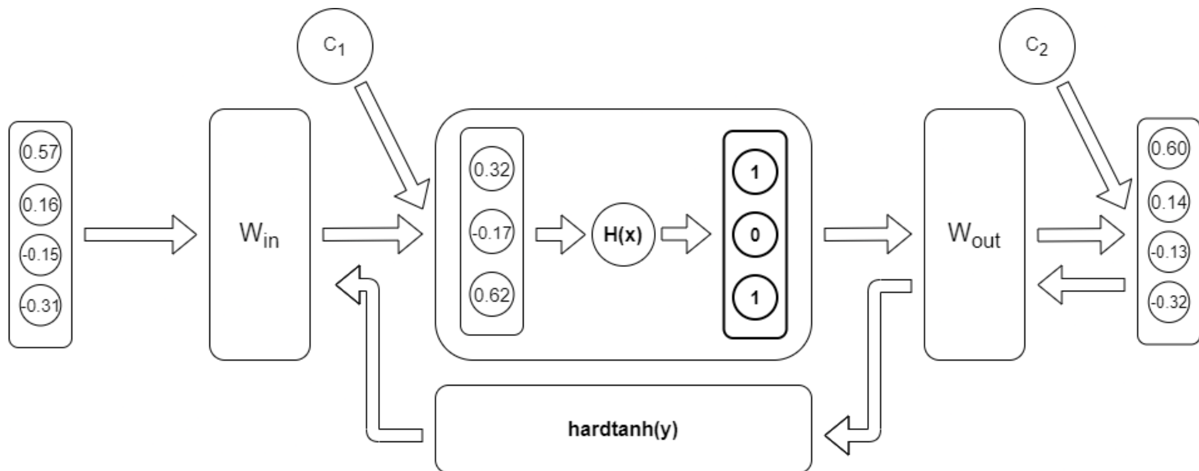

Figure 2.2: A visual representation of the workings of the autoencoder when a straight-through estimator is used. W_{in} and W_{out} are weight matrices and c_1 and c_2 are bias vectors. $H(x)$ is the Heaviside step function. The binary word embedding that we want to retrieve is in bold.

is defined as:

$$cosine(X,Y) = \frac{X \cdot Y}{||X||\,||Y||} \quad (3.2)$$

Here X and Y are the two vectors to be compared. We make use of four different datasets that contain a number of word-pairs with a score that determines how similar these words are. These scores are given by humans. The four datasets that are used are the Stanford Rare Word Similarity dataset (RW) (Luong, Socher, & Manning, 2013), the MEN dataset (Bruni, Tran, & Baroni, 2014), the SimLex-999 dataset (Hill, Reichart, & Korhonen, 2015) and the Wordsim353 dataset (Agirre et al., 2009). The Stanford Rare Word is a collection of word-pairs, with words that are not very common. The MEN dataset consists of 3000 word-pairs and during the scoring, the focus lied on relatedness. The SimLex-999 dataset contains 999 wordpairs, where focus lies more on similarity rather than relatedness. And finally the last dataset is the wordsim353 dataset, which contains 353 word-pairs. The Spearman's rank correlation is determined of both sets of embeddings against the scores given in the datasets. The Spearman's rank correlation is a non-parametric measure to determine the direction and strength of the correlation between two variables. This way we can measure how well both sets of embeddings perform in the word similarity tasks. To compare it to the original real-valued embeddings we measure the Spearman's rank correlation for those as well.

3.1.1 Binomial Test

To determine whether the word embeddings that are trained using STE are significantly better in comparison to the embeddings that are trained without STE (nSTE), we make use of a binomial test (Salzberg, 1997). For this we look at how many times the scores for the word pairs using STE embeddings, were more similar to the actual scores, in comparison to the nSTE scores. We will count the number of successes, which is the number of times where:

$$|S_h - S_{STE}| < |S_h - S_{nSTE}| \quad (3.3)$$

Here S_h is the human score for the word pair, S_{STE} is the similarity score for the word-pair based on the STE word embeddings, and S_{nSTE} is the similarity score based on the embeddings trained without STE. So this means that when S_{STE} is closer to S_h, it is considered a success. It is considered a failure if S_{nSTE} is closer to S_h. If the scores S_{STE} and S_{nSTE} are the same the word pair will not be considered.
The binomial distribution will be used to determine the probability of the STE word embeddings performing better in the word similarity task in com-

parison to the nSTE embeddings:

$$P = 2 \times \sum_{s}^{n} \frac{n!}{s!(n-s)!}(0.5)^n \tag{3.4}$$

Here n is the number of trials, and s is the number of successes. It will be multiplied by 2 since a 2-sided test is used, since we assume that both models would have a better score equally much. We will make use of an α of 0.05.
The different models and the evaluations that are made are publicly available on GitHub [*].

4 Results

The results of the Spearman's rank correlations are found in Table 4.1. For every generated set of word embeddings, we tested how well each set would perform in multiple word similarity tasks. What we see is that the real-valued embeddings from GloVe, perform best in all cases, which is as expected. When we look at the binary embeddings, we see that in most cases, each pairing of nSTE and STE embeddings have a very similar score, and thus have no significant results. However, if we look at the results in the word similarity task with the word-pairs from SimLex, we see that the results from the autoencoder with straight-through estimators, are significantly better, even though these are the lowest scores in general. The scores from the Rare Word set are also the lowest. In general the word embeddings perform the best when the word-pairs from the MEN dataset are presented.
When we take a look at the quantization error between the continuous and binary latent representation we see that the value lies around 0.50. There is an increase in the STE 128 set to 0.57.

4.1 Qualitative Results

In Table 4.2 we can find the ten most similar words to the word *car* according to each of the generated sets of word embeddings. Some of the interesting finds is, that if we look at the nSTE 128 results, we see that there are a number of words that does not seem to be directly related to the word *car*, such as *bomb*, *15* and *killed*. The same is true, for the STE embeddings with a dimension of 128, with words like *behind*, *into* and *victim*. The embeddings with higher dimensions, does not seem to have this problem.

[*] `https://github.com/robinentjes/bachelorproject`

5 Discussion

Overall, the auto-encoder did not seem to improve that much in comparison to the autoencoder without straight-through estimators. This can be caused by a number of things. Something that could be considered is that the word-pairs in the different datasets are scored based on different premises. The scores in the MEN dataset are mainly based on relatedness, whereas the other datasets focus more on similarity. The high correlation with the MEN set shows that the word embeddings give a better representation of relatedness, rather than similarity. This can be caused by the input of the autoencoders being the GloVe dataset. If a different set of real-valued embeddings was used for the input, it may lead to better results. Another option would be is to use a different set for the input in combination with the GloVe dataset. This could lead to word embeddings that both represent relatedness and similarity. This could lead to an increase in the correlation scores in the word similarity tasks.
What also could be considered is that all parameters were held the same in the model. This is useful in comparing both models. However, it may be the case that these parameters were not optimal in the model with straight-through estimators.
When we take a look at the quantization errors of both methods, we see that they both stay the same. To increase this we could make use of a variational autoencoder.

6 Conclusion

In this thesis we looked at a different method to train binary word embeddings, which involved an autoencoder with straight-through estimators. This allows us to perform full end-to-end training, while still making use of the non-differentiable Heaviside step function. It was expected that this would perform better, since the input weight matrix could be properly trained. When the trained word embeddings were presented in a word-similarity task, the embeddings that were trained with STE were

	RW			MEN			SimLex			WordSim		
	nSTE	**STE**	**GloVe**	**nSTE**	**STE**	**GloVe**	**nSTE**	**STE**	**GloVe**	**nSTE**	**STE**	**GloVe**
128	0.280	0.305		0.566	0.580		0.239	**0.240**		0.401	0.444	
256	0.342	0.342		0.664	0.673		0.299	**0.340**		0.510	0.511	
512	0.368	0.369		0.691	0.699		0.324	**0.341**		0.554	0.555	
			0.412			0.749			0.371			0.601

Table 4.1: The results of the the Spearman's rank correlation tests from the different methods against the human evaluations. Significant results, according to the Bernoulli trials, are shown in bold. The p-value that is used is 0.05

nSTE	128	256	512	**STE**	128	256	512
	car	car	car		car	car	car
	truck	vehicle	vehicle		cars	cars	cars
	cars	cars	cars		vehicle	driver	vehicle
	vehicle	truck	driver		truck	truck	truck
	bomb	vehicles	truck		driver	vehicle	motorcycle
	driver	driver	driving		behind	driving	driver
	vehicles	driving	vehicles		driving	automobile	vehicles
	15	garage	bus		engines	suv	driving
	killed	door	automobile		into	time	jeep
	train	drive	drivers		victim	drivers	parked

Table 4.2: A table containing the top ten most similar words to the word *car* according to the generated binary word embeddings.

only significantly better with one dataset (the SimLex dataset). This can be caused by the fact that the SimLex dataset was mainly focused on scores based on relatedness and not as much focused on actual word similarity. This probably comes from the fact that the GloVe embeddings represent relatedness. Furthermore, we see that in all cases the performance increases, when the vector size of the binary embeddings is increased. This makes sense, since vectors with higher dimensions can hold more information.

Only adjusting the quantization layer does not seem to give a significant difference. However, the model could be further extended. Using a non-linear model could increase the performance. This is an advantage of making use of straight-through estimators, since we are not limited by this architecture. This is something that could be studied in future work. Furthermore, the model could be extended to a variational autoencoder, which could lead to a decrease in the quantization error. Overall, the autoencoder with straight-throught estimators shows potential, since it could be further extended to different architectures. Without these variations it already gives similar scores to the previous method.

References

Agirre, E., Alfonseca, E., Hall, K., J., K., Pasca, M., & Soroa, A. (2009). A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of naacl-hlt.*

Ahmad, A., & Amin, M. (2016). Bengali word embeddings and it's application in solving document classification problem. In *2016 19th international conference on computer and information technology (iccit)* (p. 425-430). doi: 10.1109/ICCITECHN.2016.7860236

Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR, abs/1308.3432*. Retrieved from `http://arxiv.org/abs/1308.3432`

Bruni, E., Tran, N., & Baroni, M. (2014). Multimodal distributional semantics. In *Journal of artificial intelligence research 49* (p. 1-47).

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks: Training neural networks with weights and activations constrained to +1 or -1. *CoRR*, *abs/1602.02830*. Retrieved from http://arxiv.org/abs/1602.02830

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (http://www.deeplearningbook.org)

Hill, F., Reichart, R., & Korhonen, A. (2015, December). SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, *41*(4), 665–695. Retrieved from https://aclanthology.org/J15-4004 doi: 10.1162/COLI_a_00237

Kramer, M. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE*, *37, Issue 2*. doi: https://doi.org/10.1002/aic.690370209

Luong, M., Socher, R., & Manning, C. (2013). Better word representations with recursive neural networks for morphology. In *Conll.* Sofia, Bulgaria.

Mena, F., & Nanculef, R. (2019, 10). A binary variational autoencoder for hashing. In (p. 131-141). doi: 10.1007/978-3-030-33904-3_12

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. (arXiv:1301.3781)

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation.

Salzberg, S. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery 1*, 317-328. doi: https://doi.org/10.1023/A:1009752403260

Tissier, J., Gravier, C., & Habrard, A. (2019, Jul). Near-lossless binarization of word embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 7104–7111. Retrieved from http://dx.doi.org/10.1609/aaai.v33i01.33017104 doi: 10.1609/aaai.v33i01.33017104