Introduction

Discussing things, you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. Thus, building a model to detect the negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion) becomes meaningful.

The purpose of our project is to build a model to detect toxicity across a diverse range of conversations, which recognizes toxicity and minimizes this type of unintended bias with respect to mentions of identities. Thus, we have built an NLP model with DNN that recognizes toxicity and minimizes this type of unintended bias with respect to mentions of identities. The dataset would be labeled for identity mentions and optimizing a metric designed to measure unintended bias.

Our model will be separated to two part:

1) Data pre-processing. In this part, we will clean the data by delete some symbols like: “ /-'?! π‘₹´”. And we will also do the word embedding by converting the words to vectors. For the word embedding part, we will use the pretrained word vectors, GloVe which is trained by Jeffery Pennington, Richard Socher and Christopher D.Ming and FastText which is released by Facebook. Then we concatenate them to get a new word vector.

2) DNN. For the deep learning model part, we use two BLSTM as the main sequential training model and then we use two kinds of Global pooling layer to extract feature points. After pooling, we use the Res-Net to do the full connect nonlinear layer. At last, the output layer will use a sigmoid activation function

Background of word embedding

There are two main model families for learning word vectors are: global matrix factorization methods, such as latent semantic analysis (LSA), and local context window methods, such as the skip-gram model of Mikolov et al.

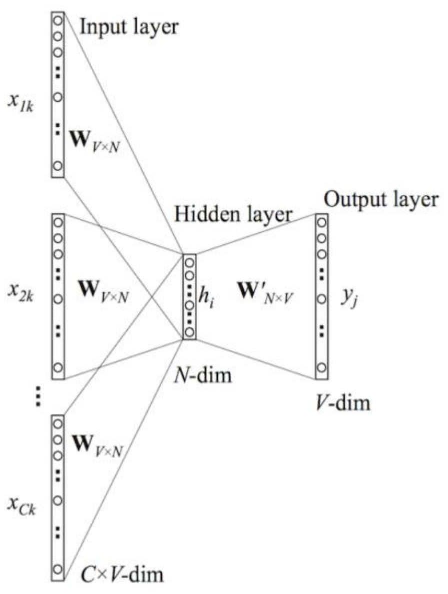
Matrix Factorization Methods. Matrix factorization methods for generating low-dimensional word representations have roots stretching as far back as LSA. These methods utilize low-rank approximations to decompose large matrices that capture statistical information about a corpus.

Shallow Window-Based Methods. Another approach is to learn word representations that aid in making predictions within local context win- dows. For example, Bengio et al. (2003) intro- duced a model that learns word vector representa- tions as part of a simple neural network architec- ture for language modeling. Collobert and Weston (2008) decoupled the word vector training from the downstream training objectives, which paved

Word2vec

Word2vec is a group of related models that are used to produce word embed- dings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

Continuous bag-of-words (CBOW) is a typical architecture of Word2vec. In CBOW, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction.



In the continuous bag of words model, context is represented by multiple words for a given target words. For example, we could use "cat" and "tree" as context words for "climbed" as the target word. This calls for a modification to the neural network architecture. The modification, shown below, consists of replicating the input to hidden layer connections C times, the number of context words, and adding a divide by C operation in the hidden layer neurons.

Our pretrained word vectors FastText is trained by this method.

GloVe

Global vectors for word representation using a combination of the Matrix and window size methods. The main method is that we set a window size and based on the size we calculate the frequency of word occurs with word j. We use Xij represent the frequency that word i and i are in the same window. Our purpose is to train a NN model to get a vector which has a close relationship with the frequency of Xij.

$$w_i^T \widetilde{w_k} + b_i + \tilde{b}_k = \log(X_{ik})$$

And we can find that wi and wk are symmetric vector. And bi and $\tilde{b}_k$are bias for wi and wk. Thus, for the whole model, our target function is:

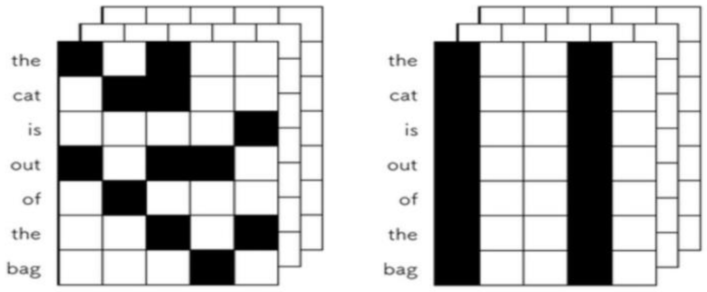
$$J = \sum_{i,j=1}^{V} f\left(x_{ij}\right)\left(w_i^T \widetilde{w}_i + b_i + \tilde{b}_j - logX_{ij}\right)^2$$

Where V is the size of the vocabulary. The weighting function should obey the following properties:

1. $f(0) = 0$. If f is viewed as a continuous function, it should vanish as $x \to 0$ fast enough that the $\lim_{x\to 0} f(x)\log^2 x$ is finite.
2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighed
3. $f(x)$ should be relatively small for large values of x, so that frequent co-occurrences are not overweighed.

SpatialDropout

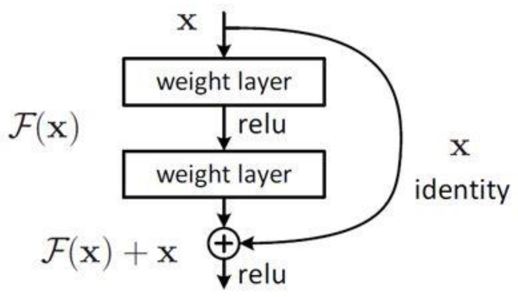
I use the spatial dropout instead of normal dropout to avoid overfitting. For normal dropout, we randomly select elements in tensor and set them to be zero. But for the spatial dropout, we will randomly select dimension and set all elements in that dimension to be zero.



The picture shows the difference between dropout and spatial dropout. As we can see that the left picture is the normal dropout, and the right part is spatial dropout.

Residual Network

Let us consider F(x) as an underlying mapping to be fit by a few stacked layers (not necessarily the entire net), with x denoting the inputs to the first of these layers. If one hypothesizes that multiple nonlinear layers can asymptotically approximate complicated functions2, then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, i.e., F(x) + x (the input and output are of the same dimensions). So rather than expect stacked layers to approximate F(x), we explicitly let these layers approximate a residual function H(x) = F(x) + x. The benefit of Res-Net is that we can increase the complexity of the model without increasing too many computations. And by increasing the complexity, we can get better result.


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Detailed individual work

1, Finishing the part of pre-processing.

2, Finishing the part of word-embedding.

3, Adding the Spatial Dropout layer to the model.

4, Adding the residual network to the model.

5, Wring code of BGRU instead of BLSTM as the model.

Results:

1) Choice of Word embedding library

Using FastText alone: public score will be 0.92870

Using GloVe alone: public score will be 0.92783

Concatenating GloVe and FastText: public score will be 0.93010

2) Choice of optimizer

Using Adam as optimizer: public score of 0.93010

Choosing AdaDelta as optimizer: public score of 0.90783

3) Size of mini-batch

Batch size of 1024: public score of 0.92825

Batch size of 256: public score of 0.93039

Batch size of 512: public score of 0.93010

4) BGRU VS BLSTM

BGRU: public score of 0.92943

BLSTM: public score of 0.93010

Code

The percentage of the code that I found or copied from the internet is almost 82%.

Summary and conclusions:

1, As we decrease the batch size, the model will generate better results, but the training time will be longer.

2, The best optimizer for the model is Adam, which has better performance than other optimizers.

3, BLSTM has better performance than BGRU, but longer training time.

4, Spatial dropout is better than dropout when we are dealing with the sequential data which has a close relationship with the data near it.

Improvements:

There are some methods which may improve our models' performance we can try in the future:

1, We can train the word vectors all by ourselves and using the vector as input to train our model.

2, We can use BERT as the transfer layer.

3, We can add attention layer before the BLST model to get the weight of the input, and then multiply the weight and word vector as the input to our training model.

Reference

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

2. Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

3. Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). Fasttext. zip: Compressing text classification models. arXiv preprint arXiv:1612.03651.

4. https://www.kaggle.com/bminixhofer/simple-lstm-pytorch-version