Optimizing text embedding for Convolutional Neural Networks

학번: 2015410113, 이름: 신채호
지도교수: 이상근 (서명) [illegible]

2021.6.14

Abstract

We present a technique for optimizing word embedding vectors for convolutional neural networks. Many impressive advances in convolutional neural networks were made these days, but natural language processing tasks are currently not able to use these techniques. In this paper, we aim to fit word embedding vectors to convolutional neural networks by carefully looking for characterists of image inputs and applying them into word embedding model. By embedding word vectors in 2 or 3 ways, inputs could work like image inputs that have multi-channels. This embedding technique has improved performance of existing CNN models that deals with text classification tasks. Also new CNN architecture of decreasing convolution weight by increasing convolution filter number, one can extremely reduce training time.

1 Introduction (서론)

Advances in deep learning technology has made many changes in natural language processing. Especially for tasks that are relatively simple like text classification, advanced deep learning models has shown incredible results. In natural language processing tasks, text classifcation includes polarity detection, topic classification, spam detection, etc.
Convolutional neural networks are mostly used in text classification tasks since simple convolution cannot catch word's sequence information well. For this reason, CNN models in natural language processing usually deals with text classification tasks. For example, CNN for sentence classification (Yoon Kim, 2014) uses embedded word vectors with one layer convolution for polarity detection and topic classification. Character-level convolutional networks for text classification (X. Zhang, J. Zhao, Y. LeCun, 2015) uses six convolution layers with character level embedding for text classifcation. Main difference between these two models is word embedding method that the former uses pre-trained word vectors and the latter uses character-level one-hot encoding vectors. Since that each encoding scheme has its own advantages, we add both of them in two-channel vectors and our goal is to perfectly fit this two-channel embedded vector into convolutional neural networks.

2 Overview (개요)

2.1 Model discription
Our model adds one channel of character-level projected word vector to basic word vectors. Basic word vectors are made by well known embedding model, Mikolov et al., 2013. We only use continuous skip-gram model since that additional character-level encoding of input words is needed. We use same word to vector model as Mikolov et al., 2013, and train character-level projected vector in two ways.
First we make bag-of-characters with each word and train character vectors of size c (character dictionary size) as the same way that Mikolov et al., 2013, does. Bag-of-character vector will be character dictionary sized vector that characters included in that dictionary are marked as 1 and otherwise 0. Bag-of-character of each word is given as input and bag-of-characters of window words are given as output. By projecting this character-level vector into

projection size l_0, we can possibly get character-level projected vector.
The second way is to give input with word vectors trained by Mikolov et al., 2013. By predicting window word vectors with bag-of-charater vector input, we could also possibly get character-level projected vector.
Character-level projected vectors of size l_0 are made by these two methods and added as new channel of final embedded vector. Then these embedded vectors are passed to simple convolution network. To compare our model with previous works, we use simple CNN model, Yoon Kim et al., 2014, which has one layer of convolution and one layer of fully connected layer. For training convenience, we also have made Fast CNN model that can reduce training time by decreasing convolution weights and increasing convolution filter number. Detailed architecture will be mentioned later.

2.2 Dataset discription

Each dataset has about 5,000 to 10,000 categorized training data. Some datasets do not have separated test set but for more robust results we seprate each dataset into train set and test set, and detailed explanation will be described later. Below are the basic information of each dataset.

MR (Pang and Lee, 2005): Movie reviews dataset with positive/negative classes. Each classes have 5331 reviews with no separated test set.

Subj (Pang and Lee, 2004): Subjectivity dataset with subjective/objective classes. Each classes have 5000 datas with no separated test set.

TREC (Li and Roth, 2002): TREC question dataset with total 6 classes (Human, Abbreviation, Description, Entities, Locations, Numeric). Total 5500 questions for training and 500 questions for test.

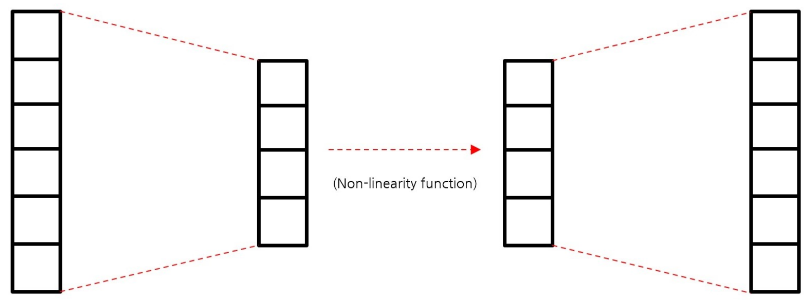
CR (Hu and Liu, 2004): Customer reviews dataset with positive/negative classes. Total 3775 reviews with no separated test set.

3 Problem (문제 정의)

Convolutional neural networks were used for some of the natural language processing tasks such as sentence classification. But ever since the attention mechanism solved problems that recursive neural networks had, e.g. long-term dependency problem, modern natural language processing techniques mostly deals with attention mechanism. On the contrary, many of the recent models for computer vision tasks still use convolution technique. Since that nearest pixels which compose specific feature are highly related to each other, CNN could catch these similarities well. By checking image with convolution filters, CNN architecture can easily catch these relationship. Images' 3-channel input, R, G, B, also perfectly fits CNN architecture and because of these reasons convolution technique could effect much more in computer vision tasks rather than natural language processing tasks. This is why models that were designed for natural language processing tasks such as transformer could be used in computer vision tasks but not vice versa. For natural languae processing tasks to use recent CNN architectures, embedded input vectors must have similar characteristics that pixels have. To use recent highly advanced CNN architectures in computer vision tasks, language input must be embedded in different way. Current embedding methods does not follow the frame of image pixels and in these reasons, we try to make language embedding vectors similar as pixel vectors.

4 Approach (아이디어)

We mainly focus on two characteristics of image inputs. First, pixels within one convolution filter are highly related. As stacking up convolution layers of network, convolution filters can catch wider relationships between distanced pixels. Word vectors Mikolov et al., 2013, are trained to catch these relationships. By training word vectors with input, output pair within the window size, this training method assumes that all the nearby words are highly related. From similar point of view, we assume that all characters within one word are highly related.



	Vector input	Projected vector		Output training vector
Original:	V x 1 (one-hot encoded)	l_0 x 1	l_0 x 1	V x 1 (one-hot encoded)
ChartoChar:	C x 1 (bag-of-characters)	l_0 x 1	l_0 x 1	C x 1 (bag-of-characters)
ChartoWord:	C x 1 (bag-of-characters)	l_0 x 1	l_0 x 1	l_0 x 1 (projected word vector)

Figure 1: Embedding model discription for original word to word, character to character and character to projected word.

Training character-level vectors with same model as Mikolov et al., 2013, vectors should be able to catch these relationships. There can be many ways of making character-level vectors, e.g. concatenating one-hot character vectors for each word, bag-of-characters, randomly initializing each vector, but since that each word should have fixed length character-level vector, we use bag-of-characters. Bag-of-characters method is making character-level vector of fixed size c (character dictionary size, 69 in our experiment). Character dictionary is made up of all 26 alphabet letters with some special characters. Special characters are needed to encode word in character-level because there are words that contain apostrophe (e.g. 've, 're, 's), hyphen (e.g. multi-channel, thirty-two) or any other special characters. These characters are ordered in character dictionary and has its own number. In bag-of-characters vectors, all characters in word are turned into 1 if the word contains that character, and 0 otherwise. In this way all words can be vectorized into fixed size c vectors. These vectors become input and output of projection model, Mikolov et al., 2013. We present two ways of making character-level projected word vectors, one is to put both input, output vectors with bag-of-characters, and the other one is to put bag-of-characters vector as input and trained word vector as output. These character-level vectors are projected into fixed size of l_0 (projection length, 100 in our experiment)during training and become final character-level projected word vectors. We see performance difference in two methods and check which one performs better in each dataset.

Second, pixels are usually composed with multiple channels. By looking at a certain document in two ways, character-wise, word-wise, words can hopefully get embedded to vectors that work like a pixel. For character-level encoded vectors, we use projected character-level word vectors mentioned above. Word-level projected vector of size l_0 trained by Mikolov et al., 2013, and character-level projected vector of size l_0 trained with two different methods mentioned above, each vectors are given as each channel input and form two-channel embedded vector. With this final embedded vector, all the

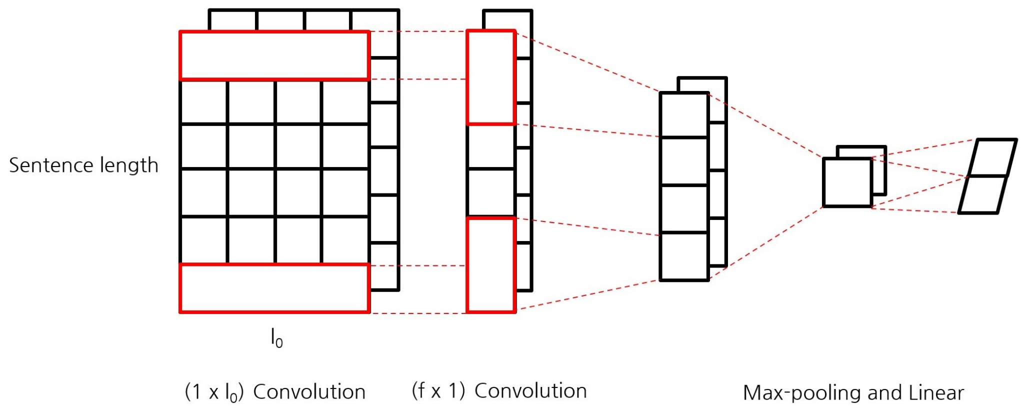


Figure 2: Fast CNN model architecture layout. Vector size of each layer is discribed with layer function name.

words in dataset are turned into two-channel vectors. Since that word input has to be vectorized into character-level once more, we only used continuous skip-gram model by Mikolov et al., 2013, to make input character-level encoding easier.

In embedding model, we also added one non-linearity function (ReLU) after first projection. Vectors are trained with non-linearity function between the fully connected layers but when getting the result vector after training, only the first projection fully connected layer is used. Convolution layers are based on model of CNN for sentence classification (Yoon Kim et al., 2014), one simple layer of convolution followed by max-over-time pooling. Outputs after applying convolution and pooling for each filter size are concatenated before getting into the fully connected layer. In this way, fully connected layer will have input size of (number of filters) * (number of filter type) and output will be number of classes. Before fully connected layer, dropout with 0.5 probability is added for regularization. Also after the fully connected layer, softmax is added so that final output can can be seen like probability of each classes.
For training speed, we have made variational CNN model. Current convolution layer has convolution filters with weight size of (filter size) * (projected vector length) and there are (number of filters) * (number of filter type) filters. Since each weight size is too big that training time takes long, we made model to have smaller weight size with more filters. We changed one colvolution layer model to two layer, first convolution layer deals with each word and second convolution layer deals with words within the filter size. After the input passes first convolution layer, each projected vector length word turns into size 1. Then this sequence of size 1 vectors are given as input of second convolution layer and checks words' relationship within filter size by convolutioning filter size of size 1 vectors. In this way, there will be two weight size filters, first layer with (projected vector length) and seconde layer with (filter size). But number of filters increase to first layer of (number of filters) and second layer of (number of filters) * (number of filter type). Although the number of filters increase by (number of filters), weight size decreases extremely from (filter size) * (projected vector length) to (filter size) + (projected vector length). This also decreases computation a lot in convolution layers which makes training time faster. We have compared two models, original CNN model and faster CNN model, by checking how model accuracy and training speed differs between two models.

5 Evaluation (실험 결과)

5.1. Experiment setup

Our main goal is to compare accuracy between three different embedding models, original word embedding, word and character to character 2-channel embedding, word and character to word 2 channel embedding. To check differences occured by embedding models, we set all different hyperparameters same.
For embedding model, all train data are trained 100 times (100 epochs) and the optimizer takes one step after the batch of one sentence. Since that batch of one sentence has its own input and output pair, different size of batches are made each sentence. Also we used normal stochastic gradient descent optimizer with initial learning rate of 0.01 and momentum of 0.9. All embedding models are trained with window size 1 (left, right each with total size 2) and the projected vector length is 100, so that computation can get easier. With these models, we make embedding vector table for each dataset and pass it to the next layer, CNN architecture layer.

For CNN architectures, all train data are trained 1000 times (1000 epochs) and the optimizer takes one step after the batch size of 64. With embedding vector table that we get from above embedding models, we change all the words in the batch into 100 size vectors. We used Adadelta optimizer and gradient clipping method if l_2-norm of weight vectors exceed limit of 3, just like Yoon Kim et al., 2014. Both original CNN model and fast CNN model are evaluated by score of accuracy and time. By checking both accuracy and time, we can assume the trade-off between two values. All the datasets except for TREC dataset does not have seperated test set so we separated data into two groups, train and test. Below are the detail description of separation for each dataset.

MR: 5000 train set each class, 331 test set each class

Subj: 4500 train set each class, 500 test set each class

	MR	Subj	TREC	CR
Original	70.5	87.4	60.2	67.3
2-channel (char)	**73.7**	**88.1**	**84.2**	68.7
2-channel (word)	72.7	87.9	81.6	**69.7**

Table 1: Accuracy for each embedding model using same CNN architecture with only different embedding part. Best resulting models' accuracy are bolded.**Original**: Original word to vector model (Mikolov et al., 2013). **2-channel (char)**: Character-level projected word vector with bag-of-characters input and bag-of-characters output. **2-channel (word)**: Character-level projected word vector with bag-of-characters input and word-level projected word vector.

TREC: Total 5500 training set, total 500 test set (original seperated dataset)

CR: Total 3475 training set, total 300 test set with 200 positive class and 100 negative class

5.2. Different embedding channels

Accuracy results in different embedding models are listed in table 1. In all the datasets both of our 2-channel embedding model worked better than original model and between two 2-channel embedding methods, character to character method seems to work better than character to word method. For character to word method, pre-trained word vector needs to be trained well before training character-level. This can be the reason why character to character method works better and character to word method can be improved if globally pre-trained *word2vec* is used. Also difference in accuracy results were not too big (about 1 percent different) except for TREC dataset (about 24 percent different). Accuracy for original embedding model seems to be not catching word relationship information well in TREC dataset. This low accuracy can be caused by underfitted embedding model since that we only trained 100 times. Or projection length (100) can be too small to catch relationship between words because most of word to vector models use projection length of 300. Also all the window size were 1 (left, right each) and if sentences have wider relationship

	MR	Subj	TREC	CR
Original (multi)	73.7	88.1	84.2	68.7
Fast CNN	72	87.1	83.8	69.7

Table 2: Accuracy for each CNN model using same embedding (word and character to character 2-channel model) model with only different CNN architecture. **Original (multi)**: Original CNN model with one convolution layer and one fully connected layer (Yoon Kim et al., 2014). Input embedding vector is 2-channel (word and character to character) model. **Fast CNN**: Fast CNN model with two convolution layer and one fully connected layer. Input embedding vector is 2-channel (word and character to character) model.

	MR	Subj	TREC	CR
Original (multi)	1332	6823	4030	**2823**
Fast CNN	**835**	**782**	**460**	4109

Table 3: Time in seconds for training each CNN model using same embedding (word and character to character 2-channel model) model with only different CNN architecture. Each model is described in table 2.

of words, it can lead to low accuracy. But by just adding one more channel of character-level projected vector, accuracy becomes high just like well trained models. This addition of one more channel not only can work like a vector of 2 times length of projection (200) but also can catch character-wise meaning. Usual word dictionary that is trained by word to vector model has word size of more than 10,000 but with character-level encoding, one can change only input size of both input and output size to alphabet dictionary size which has about 100 characters in maximum. This can decrease time of training second channel embedding and adding one more channel of character-level encoding can lead to dramatic increase of accuracy if relationship of words are not caught well in only one-channel word embedding.

5.3. Different CNN models

Accuracy and training time results in original and Fast CNN architecture are listed in table 2, 3. For most of the datasets, Fast CNN model has reduced training time a lot except for CR dataset. But accuracy difference were not too big (about 1 2 percent) and training time took longer with Fast CNN in CR dataset but the accuracy was better. If increased filter number exceeds computational time of decreased weight size, Fast CNN can take more time just like CR dataset. So with different situations and dataset, if 1 or 2 percent difference of accuracy doesn't matter and have to get results quick, one can apply Fast CNN method. Also Fast CNN might not lead to reducing of training time but it can lead to performance increase.

6 Related Work (관련 연구)

Efficient Estimation of Word Representations in Vector Space (Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean 2013)

Efficient Estimation of Word Representations in Vector Space (Mikolov et al., 2013) proposes two deep learning architecture for computing continuous vector representations of words. By changing word vectorizing task to prediction task, words‘ similarity could be represented in projected word vectors. The base model has input layer, projection layer and output layer. All the words are one-hot encoded in word dictionary and given as input and output. CBOW architecture predicts current word (output in the base model) based on the context (words within the window, input in the base model). Skip-gram architecture predicts surrounding words (words within the window, output in the base model) based on the current word (input in the base model). After training a deep learning model by predicting words, each vector can be projected into vectors by trained model. In this way, projected vectors can still have word similarities and can work as more powerful embedding word vector.

CNN for sentence classification (Yoon Kim, 2014)

CNN for sentence classification (Yoon Kim et al., 2014)uses word vectors(Mikolov et al., 2013) trained on 100 billion words of Google News. In a sentence, by making a model that predicts words from surrounding words or vice versa, words' similarities can be represented in word vectors. All the words in a sentence are embedded into fixed length vector and given as an input. These inputs then go through one convolutional layer, one max pooling layer and one fully connected layer. Model has four variations which are CNN-rand, CNN-static, CNN-non-static, CNN-multichannel. CNN-rand uses random initialization of word vectors and modified during training. Both CNN-static and CNN-non-static use pre-trained word vectors but the former one is not modified and the latter one is fine-tuned during training. CNN-multichannel uses both static and non-static channels. Without random initialization each model variations had achieved state of the art in different tasks. This concludes that quality of embedded words effects model's performance a lot and it differs in tasks. Also with good quality of embedded words, simple one layer of convolutional network can perform well.

Character-level Convolutional Networks for Text Classification (X. Zhang, J. Zhao, Y. Le-Cun, 2015)
Character-level convolutional networks for text classification (X. Zhang et al., 2015) usese character-level encoding. First, character-level encoding defines character dictionary that includes total 70 characters in only lower alphabet case. Then each characters in a sentence are quantized using one-hot encoding (1-of-70 encoding in lower alphabet only case). All the sentences are vectorized only with 1014 characters, padded if necessary, with one-hot encoded vector each. Final dimension of each sentences will be 70x1014 (in lower alphabet only case) and this matrix is given as input of convolutional network. 1-D convolution is used in total 6 convolutional layers and 3 fully-connected layers after that. This model uses deeper model than word embedding model so that relationship within long input feature could be captured. Also model needs big size data to work well. This concludes that character level relationship also could be captured with large size datasets without knowledge about syntactic or semantic structure.

7 Conclusion (결론)

In this paper we have presented two different embedding technique that can be added as second channel for convolution networks. This experiment shows that adding a channel of character-level projected word vector can make previous word embedding models work better. Even when word vectors cannot catch similarities because of some reasons, by simply adding a second-channel character-level projected vector can lead to accuracy increase. These results show that putting two-channel embedding vector in CNN architecture can lead to performance increase and can get more accurate results. Also by comparing time and accuracy results of two different CNN architectures, we can assume when to apply each model. We hope this work can change the view of CNN models in natural language processing tasks.

Future works
In the future, we hope to train *word2vec* vectors in different parameters. Word vectors trained in our experiments are trained in minimum condition, projection size of 100 and window size 1 each direction. The reason of setting on minimum condition was basic word2vec model needs to have (vocabulary size) * (projection size) size of weights each layer and vocabulary size usually exceeds at least 10,000. So the number of computation becomes too big and takes a lot of training time as projection size and window size increases. Also we hope to do the experiments with pretrained global word vectors with vector size 300. Current experiments are all based on the datasets which do not have that many data and results in low accuracy. On the contrary, publicly availabe *word2vec* vectors are trained on 100 billion words from Google News. Based on these vectors we can possibly add second-channel character-level projected vector and see the results. These are the possible works that we could do in the future for finiding more optimal embedding technique for convolutional neural networks in natural processing tasks.