Indian Research Journal on Advanced Technologies

Complete volume available at https://www.irjat.com/current-issue

Indian Research Journal on Advanced Technologies

Journal homepage: www.irjat.com

IRJV01I02P04

Convolutional Neural Network for Sentence Classification

Ritika Saxena

Abstract

We record a series of observations of pre-trained word vectors with Convolutional Neural Network (CNN) for sentence classification process. The work shows how a simple Convolutional Neural Network (CNN) model with minimal hyperparameter tuning achieves excellent results on various benchmarks. Learning of task specific vectors also offers gain in performance. Additionally, a simple modification in proposed in the architecture, allowing the use of both the task specific and the static vectors. The CNN model herein discussed, improves upon the state of art in 4 out of the 7 tasks (includes sentiment analysis and the question classification).

Introduction

The question is "How can an algorithm read words?" The suggested solution to this can be, transformation of words into vector to have numerical representation of them. Deep learning models have achieved astonishing results in speech recognition n (Krizhevsky et al., 2012) and computer vision n (Graves et al., 2013) in the past few years. In Natural Language Processing (NLP), much of the work is involved by Deep learning, which involves learning with word vector representation through natural language models and performing composition over the learned words vector for classification n (Collobert et al., 2011).

Word vectors, where words are projected from sparse, 1-of-V encoding (where V is the vocabulary size), into a low dimensional vector space via the hidden layer. These are the essential features extractors that encode semantic features in the dimensions. In such a dense representation, semantically close words are likewise close - in cosine or euclidean distance - in the lower dimensional vector space.

Originally CNN models were invented for computer vision, but have proved effective for NLP. They have also achieved remarkable results in semantic parsing (Yih et al., 2014), sentence modelling (Kalch-bernner et al., 2014), search query retrieval (Shen et al., 2014 and other traditional NLP techniques (Collobert et al., 2011). Convolutional Neural Network (CNN) utilizes layers of convolving filters that are applied to local features.

In this work, we train a simple CNN model with one layer of convolution on the top of the word vectors, which is obtained from an unsupervised neural language model. These vectors are trained on 100 billion Google News that were trained by Mikolov et al. (2013) and are publicly available. Initially, we keep the word vectors static and learn only the other parameters of the model.

In spite of little hyper-parameters, the simple model achieves excellent results on many benchmarks, suggesting that the pre-trained vectors are 'universal' feature extractors and can be utilized for various classification tasks. Further improvements are the result of learning task-specific vectors. Finally, we describe a simple modification to the architecture that allows the use of both, the pre-trained and the task specific vectors, by having multiple channels.

Model

The model architecture, shown in figure 1, is the slightly different from the CNN architecture oh Collobert et al. (2011).

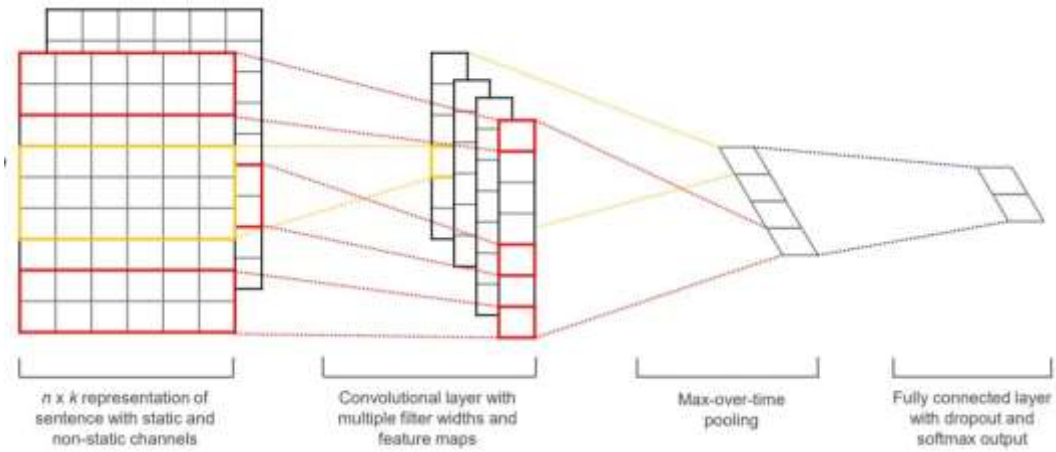

Figure 1: Model architecture with two channel for an example sentence

Let x_i belongs to R^k where k is the dimensional vector corresponding to the i-th word in the sentence.

A sentence of length n (is padded where necessary) is represented as

$X_{1:n} = X_1 \oplus X_2 \oplus \ldots \oplus Xn$ ---------- (1)

Here $\oplus$ represents the concatenation operator. Now, let $X_{i:i+j}$ refers to the concatenation of the words X_i , X_{i+1} , ...X_{i+j} .

A convolution operation involves a filter w, which belongs to R^{hk} , which is applied to a window of h words to produce a new feature.

For example, a feature named c_i is generated from a window of words:

$X_{i:i+h-1}$ by $c_i = f(w \cdot x_{i:i+h-1} + b)$---------- (2)

Here $b \in R$ ia a bias term and f is a non-linear function such as the hyperbolic tangent. The filter is applied to each and every possible window in the sentence $\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$ to produce the feature map: $c = [c_1, c_2, \ldots, c_{n-h+1}]$ ---------- (3) (where $c \in R^{n-h+1}$).

Then we apply max-overtime pooling operation (Collobert et al, 2011) over the feature map and we take the $\hat{c} = \max\{c\}$ as the feature corresponding to this particular filter. The main idea is to capture, the most important feature, the one with the highest value for each feature map.

The mentioned pooling scheme deals naturally with the sentences of variable lengths.

Hereby, we describe the process by which one feature can be extracted from one filter. The model will use multiple filters, with varying window sizes, to obtain multiple features.

Now, these features from the penultimate layer are passed to a fully connected softmax layer whose output is the probability distribution over labels. In one of the model variants, we experiment with having two “channels” of the vectors. One that is kept static throughout the training and other one that is fine-tuned via backpropagation. In multichannel architecture, figure 1, we note that each filter is applied to both channels and the results are added to calculate c_i in equation no. (2).

Regularization

We additionally constrain l2-norms of the weight vectors by re-scaling w to have $||w||2 = s$ whenever $||w||2 > s$ after a gradient descent step. Dropout on the penultimate layer

is employed, for regularization, with a constraint on l_2-norms of the weight vectors. Co-adaptaion of hidden units are prevented by dropout, by randomly dropping out, that is., setting to zero, a propotion p of the hidden units during forward backpropagation. That is, given the penultimate layer $z = [\hat{c}_1, \hat{c}_m]$, where m is the number of filters.

Instead of using: $y = w \cdot z + b$ ---------- (4)

for output unit y in forward propagation, dropout uses: $y = w \cdot (z \circ r) + b$ ---------- (5)

Here ◦ denotes the element wise multiplication

n operator and $r \in R^m$ is a 'masking' vector of Bernoulli random variables with probability p of being 1.

The gradients are backpropagated only through unmasked units. At the testing time the, the learned weight vectors are scaled by p such that: $\hat{w} = pw$

$\hat{w}$ is used without dropout, to score unseen sentences.

Data	c	l	N	|V|	|Vpre|	Test
MR	2	20	10662	18765	16448	CV
SST-1	5	18	11855	17836	16262	2210
SST-2	2	19	9613	16185	14838	1821
Subj	2	23	10000	21323	17913	CV
TREC	6	10	5952	9592	9125	500
CR	2	19	3775	5340	5046	CV
MPQA	2	3	10707	6246	6083	CV

Table 1 summarizes the statistics for the data-set after tokenisation. Here:

c : represents number of target classes, L

: represents average sentence length, N :

represents data-set size.

|V| : it is the vocabulary size

$|V_{pre}|$: it is the number of words present in the set of pre trained word vectors.

Test : test set size

CV : it means there was no standard train or test split.

Experimental Setup and Datasets

We test our model on various benchmarks. The above table shows the summary statistics of the data-set.

- Movie Review (MR): Movie reviews involves one sentence per review. The classification involves detection of positive or negative reviews.
- SST-1: Stanford Sentiment Treebank, it is the extension of movie review but train / dev/test splits are provided and the fine grained labels including very positive, positive neutral, neutral negative, very negative.
- SST-2: Stanford Sentiment Treebank, it is same as SST-1 but the neutral reviews are removed and and the binary labels are also removed from the data-set.
- Subj: It is the subjective data-set, where the task is to classify a sentence as being subjective or objective.
- CR: CR stands for Customer Reviews. CR of various products like cameras, MP3s, etc. are taken into count. The task is to predict that whether, the reviews given by the customer, is positive or negative.
- MPQA: Opinion polarity detection is the subtask of the MCQA data-set (Wiebe et al., 2005)

Hyperparameters and Training

For all the data-sets the following are used:

- Rectified linear units
- Filter windows (h) of 3, 4, 5 with 100 feature maps each,
- Dropout rate (p) of 0.5
- L_2 constraint (s) of 3
- Mini batch size of 50.

There values are chosen from a grid search on the Stanford Sentiment Treebank-2 dev set. We do not perform any tuning on data-set other than early stopping of dev sets. We randomly selected 10% training data for the data-sets without a standard dev set. The training is done through stochastic gradient descent over shuffled mini batches with Adadelta update rule.

Pre-trained Word Vectors

In the absence of large supervised training data the most popular method to improve the performance of the model, is to, initialize word vectors with those obtained from unsupervised neural language. We used word2vec vectors, that are trained on 100 billion Google News. These are publicly available. The vectors have the dimensionality of 300 and are trained using the continuous bag of words architecture. Words which are not present in the pre-trained words, are initialized randomly.

Model Variation

Experiments were also done with the several variants of the model.

- CNN-rand: This is the baseline where all the words are randomly initialized and they are modified during the training process.
- CNN-static the model with pre-trained vectors from word2vec. All the words, including the unknown ones are initialized randomly and are kept static. Then the other parameters of the model are learned.
- CNN-non-static: just like the CNN-static the pre-trained vectors are fine-tunes for each task.
- CNN-multichannel: The model has a set of two word vectors. The filter is applied to both the "channels" (here the vectors are treated as a "channel"). Despite of the two channels, the gradients are back propagated only through one of the channels. Hence the model can fine tune one set of vectors, keeping the other one static. Both the channels are initialized with word2vec.

Model	MR	SST-1	SST-2	SUBJ	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	89.6
CNN-non-static	81.5	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	88.1	93.2	92.2	85.0	89.4

Table 2

We stamp out other sources of randomness, like, CV-fold assignment, initialization of the unknown word vectors and initialization of CNN-parameters, by keeping them uniform within each data-set. This is done in order to remove the above variations.

Results and Discussion

The results are compared with other methods and are shown in table 2. The baseline model with the randomly initialization of words, CNN-rand, does not perform well on itself. We are surprised with the magnitude of the grains, because we used the pre-trained vectors.

A simple model with static vectors, like CNN-static, it performs excellently by giving us the competitive results against more sophisticated deep learning model. The deep learning models in them utilizes complex pooling layers or uses parse trees to compute beforehand.

Hence the results shows that the pre-trained vectors are good "universal" features extractors and they can also be utilized across data-sets. Further improvements are

achieved by fine tuning of the pre-trained vectors for each task (CNN-non-static is one of the example).

Single Channels v. Multichannel Models

Initially, we hoped that, on small data-sets multichannel architecture would prevent over fitting and works better than single channel model. The reason being, the learned vectors will not deviate too far from the original values. However, the results are mixed. For instance, in place of additional channels being used for a non-static portion, we could maintain a single channel and employ extra dimensions. They can also be modified during the training process.

Non-static v. Static Representation

Multichannel models are able to fine tune the non-static channels in order to make them more specific to task-at-hand. For example, bad is similar to good in word2vec, the reason can be, they are syntactically equivalent.

But this is not the case for the vectors in non-static channels that are fine tuned in SST-2(table 3). Similarly good is arguably far from great than it is to nice for expressing sentiment, and this indeed is reflected in learned vectors.

Fine tuning allows randomly initialized tokens (not in the set of pre-trained vectors) to learn more meaningful representations: the network learned that the commas are associated with conjunctive and that exclamations are effusive expression(table 3).

	most similar words for	
	static channel	non-static channel
bad	good terrible horrible lousy	terrible horrible lousy stupid
good	great bad terrific decent	nice decent solid terrific
n't	os ca ireland wo	not never nothing neither
!	2500 entire jze changer	2500 lush beautiful terrific

Table 3: It shows the top 4 neighbouring words based on cosine similarity.

Other Observations

Other than multichannel or single channel models and static and non-static representations, some other observations are:

- Dropout proved to us a good regularize, and can be used with larger networks also. It consistently added 2 to 4 percent to the relative performance.
- When the randomly initialized words are not present in word2vec, we obtained slight improvements, by sampling each and every dimension from U[-a,a]; where we choose a such that, randomly initialized vectors can have the same variance as the pre-trained ones.
- We also experimented, briefly, with publicly available vectors (trained by Collobert (2011)). The results given by word2vec are far superior that the expectations.

Conclusion

In our present work, a series of experiments are discussed with CNNs build on the top of word2vec. Despite of some hyperparameter tuning, a Convolutional Neural Network with single layer of convolution performs remarkably well. Unsupervised pre-training of the word vectors are the important ingredient for Natural Language Processing(NLP) and our results add up to this evidence.

References

- Y. Bengio, R. Ducharme, P. Vincent. 2003. Neural Probabilitistic Language Model.
- Journal of Machine Learning Research 3:1137–1155.
- R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuglu, P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research 12:2493–2537.
- J. Duchi, E. Hazan, Y. Singer. 2011 Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12:2121–2159.
- L. Dong, F. Wei, S. Liu, M. Zhou, K. Xu. 2014. A Statistical Parsing Framework for Sentiment Classification. CoRR, abs/1401.6330.
- A.S. Razavian, H. Azizpour, J. Sullivan, S. Carlsson 2014. CNN Features off-the-shelf: an Astounding Baseline. CoRR, abs/1403.6382.