Report of Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding

Cong Cheng CCHENG33@BUFFALO.EDU
CSE department, University at Buffalo

Abstract

The semi-supervised CNN[1] introduces an idea of learning the embeddings of regions of sentences instand of a word, and use those embeddings to do classifcation tasks, such as sentiment analysis. Moreover, they use not only the embeddings learned directly for the labled training data, but also those embeddings learned from a large amount of unlabled data by predicting context regions of the target region. The paper achieves state-of-the-art results on both sentiment analysis and topic classification tasks on IMDB, Elec, and RCV1 dataset. Thus, we wonder how it performs on the Stanford Sentiment Analysis TreeBank compared to the RNN model of Richard et al [2]. We reproduced the idea of Semi-supervided CNN and the its previous supervided CNN models [3] using Tensorflow[4]. In this report, we provide the results we obtain of these two models, compared to our simple model and the RNN model. We explain this paper in details, providing how to use chain-rule to get gradients of each layer and deriving the proof of tv-embedding idea in more detailed steps. Finally, we discuss the advantage and weakness of the paper, and propose a new model called "Phase Embedding" for text understanding.

1. Three CNN Models I implemented

1.1. BOW Sentence Embedding

1. This first one is 128_hidden_then_softmax.py, which is a model can be seen as BOW-Sentence-Embeding. In other words, each sentence is represented by a vecter of length $|V|$ (the size of the dictionary, say 15000). The indexes of words in the sentence are set to 1, and all the other places are zeros. Let's see an simple example of this "many-hot" representation. Suppose your dictionary only have 10 words, and words "this", "is", "a" and "test" are indexed at positions 0, 1, 2, 3 respectively. Then the we use a vector of 1111000000 to represent the sentence "this is a test". Then the input layer connects a hidden layer of 128 neurons, and finnaly we use softmax to connect the 128 neurons to the two output neurons (positive and negitive).

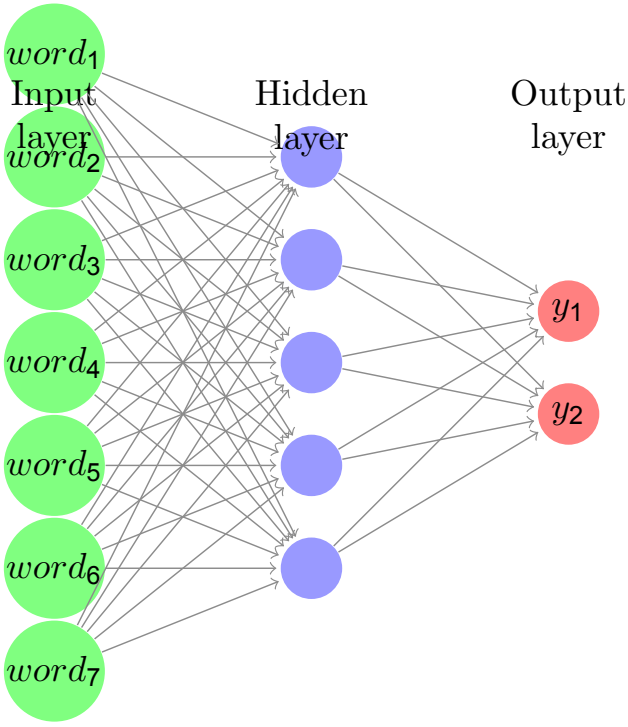

Figure 1. BOW Sentence CNN

Obviously, we lose the word order of the sentence, but it's worth to see how good/bad it can achieve, especially compared to the complexed models we are going to imprement next. We train this model on Stanford Sentiment Analysis TreeBank data, and limit the vocabulary size to 20000, so that only the most frequent words will be considered and all the other words will be replaced by the word "UNK",

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

i.e., zero vector. We use stochastic gradient descent to find the minimum cost of the cross-entropy between the predictions and true lables. In other words, in each iteration, we only use a small random batch of the training data, and pretend that is the whole dataset, and use the gradient of this small batch to update our parameters. This model achieves 74% accuaracy on development data set, and 75% on test data set. After we add another 10,000 training data from the PangLee Rotten Tomatoes dataset, we achieve 86% accuracy on development dataset and 88% on test dataset, which is no surprise since no model can guaranteed to predict something correctly if it never saw that kind of pattern.

you can run this model by just type "python 128_hidden_then_softmax.py".

1.2. Supervised Seq-CNN Model

The second model I implemented is the idea of the paper "Effective Use of Word Order for Text Categorization with Convolutional Neural Networks". Specifically, I implemented its seq-CNN for sentiment analysis. Each sentence is represented by a $|D|$*1 vector, where $|D|$ is the number of words in the sentence. Thus each word can be seen as a pixel in the image, and the only difference is that the width of this "sentence-image" is 1. Each pixel has $|V|$ channels (the size of dictionary), and only one of the channel is 1, and all the other channels are 0s. Then comes to the convolutional layer. we use 32 filters to each region of two consecutive words, learning 32 features of each region. Then comes to the max-pooling layer, and finnaly the softmax layer. Unlike the images, the length of sentence is a variable, but we still want the final full-connection layer has fixed number of neurons. Thus, we pad each sentence with 0s to the length of max sentence length in the dataset, i.e., 64.

After 20,000 iterations of SGD on the training data, we obtain 84% accuracy on the dev dataset, and 87% on the test dataset. You can run this model by simplely type "python seq_CNN.py".

1.3. Semi-supervised CNN for text categorizaition via region embedding

The third model I implemented is the idea of the paper "Semi-supervised convolutional neural networks for text categorization via region embedding", which is an improvement from model 1.2. Now it also uses unlabled data to learn the region embedding by predicting the context region based on the target region. Thus, when trying to predict the sentiment of a sentence, we have two embeddings of each region. One comes from the supervised part, same as the output of convoluation layer of model 1.2, and the other one comes from the unsupervised part.

The region embedding learning from unlabled data is a little bit different from the word embedding learning. For word embedding, the embedding of each word are stored in the parameter W between the input layer and hidden layer. W is a V*H tensor, where V is the dictionary size and H is the number of neurons in the hidden layer. Thus, each row i of W is the embedding for that word that indexed at i in the dictionary. However, for region embedding learning, we use the output the convolution layer as the embedding for each region. If we set the output channel of the convolution layer to 128, then we have a vection of length 128 to represent each region. Thus, we can first train our unsupervised region embedding model, and store the parameters between the input layer and convolution layer (checkpoint), and then when train our supervised model of this paper, we restore model of unsupervised part, and feed our seq-CNN or bow-CNN representation of the region to the input of unsupervised model, then we obtain u‘(x) or that region. Next, we combine it with model 2 to satisfy this equation: relu(W * r‘(x) + V * u‘(x) + b). Now our model will learning the parameters for W, V, b, and of course the parameters of other layers. The advantage of it is that we can also obtain the region vecter u‘(x) even though we didn't see it in the unlabled training data.

Due to limited time and computation resource, currently I don't have the result for this model. I already implemented the supervised part, and the unsupervised region embedding learning part, but haven't combined them together. The combination of the two model would take much time, but it takes time to train the unsupervised model, and tune the parameters.

2. Using chain rule to compute the gradient

Since CNN is a complex composite function, and the derivative of the composite function is the product of the derivatives of the components. Thus, we just need to use the chain-rule to compute the gradient for each layer, which is called back propagation in deep learning.

$$\sigma(\mathbf{W} \cdot \mathbf{r_l}(x) + \mathbf{V} \cdot \mathbf{u_l}(x) + b) \tag{1}$$

Let's expand (1) as,

$$\begin{aligned} y^t &= sigmoid(\sum_{h1=1}^{H} w_{1_{h1}} \cdot z^t_{l_{h1}} + \sum_{h2=1}^{H} v_{1_{h2}} \cdot z^t_{l_{h2}} + b_1) \\ &= \frac{1}{1 + exp[-\sum_{h1=1}^{H} w_{1_{h1}} \cdot z^t_{l_{h1}} - \sum_{h2=1}^{H} v_{1_{h2}} \cdot z^t_{l_{h2}} + b_1]} \end{aligned} \tag{2}$$

$$z_h = sigmoid(-\sum_{j=1}^{d} w_{0_{hj}} \cdot \mathbf{r}_{\mathfrak{l}}(x) - \sum_{j=1}^{d} v_{0_{hj}} \cdot \mathbf{r}_{\mathfrak{l}}(x) + b_0) = \frac{1}{1 + exp[-\sum_{j=1}^{d} w_{0_{hj}} \cdot \mathbf{r}_{\mathfrak{l}}(x) - \sum_{j=1}^{d} v_{0_{hj}} \cdot \mathbf{r}_{\mathfrak{l}}(x) + b_0]} \tag{3}$$

We know that the derivation of sigmoid function $y = sigmoid(\alpha)$ is,

$$\frac{dy}{d\alpha} = (\frac{1}{1 + exp(-\alpha)})' = y(1 - y) \tag{4}$$

In addition, we know that training a multilayer perceptron is a nonlinear function of the inputs thanks to the nonlinear basic function in the hidden units. Considering the hidden units as inputs, the second layer is a perceptron. Given inputs z_h, for the first-layer weights, $w_h j$, we can use chain rule (back-propagation) to calculate the gradient:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}} \tag{5}$$

Since the task is binary classification, namely, we are given a sample of two classes as, $\mathbf{X} = \{x^t, r^t\}$, where, $r^t = 1$ if $x \in positive$ and $r^t = 0$ if $\mathbf{x} \in negative$. We assume r^t, given $\mathbf{x}^t$, is Bernoulli with probability $y^t \equiv P(positive|\mathbf{x}^t)$ as calculated as $r^t|\mathbf{x}^t \sim Bernoulli(y^t)$.

As a discriminate-based approach, we model directly $r|\mathbf{x}$. The sample likelihood is

$$l(\mathbf{w_0}, \mathbf{w_1}, \mathbf{v_1}|\mathfrak{X}) = \prod_t (y^t)^{(r^t)} (1 - y^t)^{(1 - r^t)} \tag{6}$$

We now have a likelihood function to maximize. Next, we turn it to an error function to be minimized as $E = -logl$ with cross-entropy:

$$E(\mathbf{w_0}, \mathbf{w_1}, \mathbf{v_1}|\mathfrak{X}) = -\sum_t r^t log y^t + (1 - r^t) log(1 - y^t) \tag{7}$$

The update equations implementing gradient descent are

$$\bigtriangledown w_{1_{h_1}} = \eta \sum_t (r^t - y^t) z^t_{l_{h1}} \tag{8}$$

$$\bigtriangledown v_{1_{h_2}} = \eta \sum_t (r^t - y^t) z^t_{l_{h2}} \tag{9}$$

$$\bigtriangledown w_{0_{h_1 j}} = \eta \sum_t (r^t - y^t) w 1_{h_1} z^t_{l_{h_1}} r_l(x) \tag{10}$$

3. Detailed derivation of the theory 1 of tv-embedding

They first assume that X_1 contains d_1 elements, and X_2 contains d_2 elements, and $|H| = k$. The independence and rank condition in Assumption 1 implies the following decomposition

$$P(X_2|X_1) = \sum_{h \in \mathfrak{H}} P(X_2, h|X_1) = \sum_{h \in \mathfrak{H}} P(X_2|h, X_1) P(h|X_1) = \sum_{h \in \mathfrak{H}} P(X_2|h) P(h|X_1) \tag{11}$$

is of rank k. As we know, the dimension of $P(X_1, X_2)$ is d_1 and d_2 and similarly, $P(X_2|X_1)$ is a d_1 by d_2 matrix. It is not hard to see that $P(X_2|h)$ and $P(h|X_1)$ are a d_2 by k and a k by d_1 matrix respectively.

From the matrix equation $A = BC$, we obtain $C = B^{-1}A$. Further, we know that $(B^T B)^{-1} B^T = B^{-1} (B^{-1})^T B^T = B^{-1}$. Therefore, $C = (B^T B)^{-1} B^T A$. If we set $U = (B^T B)^{-1} B^T$, U is a k by d_2 matrix. Then, we can write up C as $C = UA$. This implies that there exists a function $\mu(h, X_2)$ as,

$$\forall h \in \mathfrak{H} : P(h|X_1) = \sum_{X_2 \in \mathfrak{X}_2} P(X_2|X_1) \mu(h, X_2) \tag{12}$$

According to the definition 1 for tv-embedding, A function f_1 is a tv-embedding of X_1 w.r.t. X_2 if there exists a function g_1 such that $P(X_2|X_1) = g_1(f_1(X_1), X_2)$ for any $(X_1\ X_2) \in \mathfrak{X}_1 \times \mathfrak{X}_2$. Using this definition, we obtain,

$$P(h|X_1) = \sum_{X_2 \in \mathfrak{X}_2} g_1(f_1(X_1), X_2) \mu(h, X_2) \tag{13}$$

Define $t_1(\alpha_1, h) = \sum_{x_2} g_1(\alpha_2, h) \mu(h, X_2)$ such that for any $h \in \mathfrak{H}$

$$P(h|X_1) = t_1(f_1(X_1), h) \tag{14}$$

Similarly, there exists a function $t_2(\alpha_2, h) = \sum_{x_2} g_1(\alpha_2, h) \mu(h, X_2)$ such that for any $h \in \mathfrak{H}$

$$P(h|X_2) = t_2(f_2(X_2), h) \tag{15}$$

where $\alpha_1 = f_1(X_1)$ and $\alpha_1 = f_2(X_2)$.

Now, we include Y into this derivation process, we firstly can obtain,

$$P(Y|X_1) = \sum_{h\in\mathfrak{H}} P(Y,h|X_1) = \sum_{h\in\mathfrak{H}} P(Y|h,X_1)P(Y|X_1)$$
$$= \sum_{h\in\mathfrak{H}} P(Y|h)P(h|X_1) \tag{16}$$

Using (3), we obtain

$$P(Y|X_1) = \sum_{h\in\mathfrak{H}} P(Y|h)t_1(f_1(X_1)) \tag{17}$$

If we can define $q_1(\alpha_1, Y) = \sum_{h\in\mathfrak{H}} t_1(\alpha, h)P(Y|h)$, we obtain,

$$P(Y|X_1) = q_1(f_1(X_1), Y) \tag{18}$$

This proves the first part of the Theorem 1.

Further, we can use similar logic to derive $P(Y|X_1, X_2)$

$$P(Y|X_1,X_2) = \sum_{h\in\mathfrak{H}} P(Y,h|X_1,X_2)$$
$$= \sum_{h\in\mathfrak{H}} P(h|X_1,X_2)P(Y|h,X_1,X_2) \tag{19}$$
$$= \sum_{h\in\mathfrak{H}} P(h|X_1,X_2)P(Y|h)$$

where equation (19) has used the assumptions that Y is independent of X_1 and X_2 given h.

The first term of equation (19) can be reorganized through following steps as products of a few functions as following,

$$P(h|X_1,X_2) = \frac{P(h,X_1,X_2)}{P(X_1,X_2)} = \frac{P(h,X_1,X_2)}{\sum_{h'\in\mathfrak{H}} P(h',X_1,X_2)}$$
$$= \frac{P(h)P(X_1,X_2|h)}{\sum_{h'\in\mathfrak{H}} P(h')P(X_1,X_2|h')}$$
$$= \frac{P(h)P(X_1|h)P(X_2|h)}{\sum_{h'\in\mathfrak{H}} P(h')P(X_1|h')P(X_2|h')}$$
$$= \frac{P(h,X_1)P(h,X_2)/P(h))}{\sum_{h'\in\mathfrak{H}} P(h',X_1)P(h',X_2)/P(h'))}$$
$$= \frac{P(h|X_1)P(X_1)P(h|X_2)P(X_2)/P(h)}{\sum_{h'\in\mathfrak{H}} P(h'|X_1)P(X_1)P(h'|X_2)P(X_2)/P(h')}$$
$$= \frac{P(h|X_1)P(h|X_2)/P(h)}{\sum_{h'\in\mathfrak{H}} P(h'|X_1)P(h'|X_2)/P(h')}$$
$$= \frac{t_1(f_1(X_1),h)t_2(f_2(X_2),h)}{\sum_{h'\in\mathfrak{H}} t_1(f_1(X_1),h')t_2(f_2(X_2),h')} \tag{20}$$

where we use the assumption that X_1 and X_2 are independent given h to derive the fourth equation from third equation and the last equation has used and . The last equation shows that $P(h|X_1, X_2)$ is a funtion of $(f_1(X_1), f_2(X_2), h)$. Therefore, there exists a function $\tilde{t}$ such that $P(h|X_1, X_2) = \tilde{(}f_1(X_1), f_2(X_2), h)$. Put (20) back to (19), we obtain,

$$P(Y|X_1,X_2) = \sum_{h'\in\mathfrak{H}} \tilde{(}f_1(X_1), f_2(X_2), h)P(Y|h) \tag{21}$$

Now, if we define $q(\alpha_1, \alpha_2, Y) = \sum_{h'\in\mathfrak{H}} \tilde{(}f_1(X_1), f_2(X_2), h)P(Y|h)$, we can get the conclusion that there exists a function q such that $P(Y|X_1, X_2) = q(f_1(X_1), f_2(X_2), Y)$. The second part of the Theorem 1 is proved.

4. Discussion

The idea of the learning region embeddings is very useful, since "no context, no meaning". We cannot determine the meaning of of a word without any context information. For example, it is hard to say what does "bank" means if we only see this word. We cannot even determine whether it is a noun and a verb. however, we know "bank" means an institute if we see "go to the bank" or "bank of america", and means a mass of a particular substance when we see "a bank of cloud". This is the main reason the region embedding is better than word embedding. Word embedding cannot solve the problem of Polysemy, because each word only have one vector representation regardless of how many meanings that word can have.

However, for the size of each region, the one-size-fits-all policy harms the models significantly. In some regions, there are more than one phrase, such as "vector representation regardless", and some other regions only contains a part of the phrase, such as "university of New". Thus, we think a better way is "phase embedding learning", where we first parse the sentence and the use phases as regions, and try to learn the representation of each phase. Although parse tree have a little bit difficult to determine the ambuguity of the sentence, but it would not affect our phase embedding learning since we may still have the same phases of the sentence regards of different parse method. For example, let's consider the sentence "John saw the pirate with a telescope". For phase embedding, we can use "John", "saw the pirate" and "with a telescope" to represent this sentence. The parser may have a distribution on who has the telescope, but we obtain the three same phases anyway. This is more natural since people also understand sentence based on phases, and understand phases based on words, instead of consective regions such as "pirate with a".

Due to limited time, currently we cannot prove the expe-

rient result for phase embedding model, but it is comming soon.

5. Reference

[1] Johnson, Rie, and Tong Zhang. "Semi-supervised convolutional neural networks for text categorization via region embedding." Advances in neural information processing systems. 2015.

[2] Socher, Richard, et al. "Recursive deep models for semantic compositionality over a sentiment treebank." Proceedings of the conference on empirical methods in natural language processing (EMNLP). Vol. 1631. 2013.

[3] Johnson, Rie, and Tong Zhang. "Effective use of word order for text categorization with convolutional neural networks." arXiv preprint arXiv:1412.1058 (2014).

[4] Abadi, Martn, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

[5] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

[6] Geoffrey Hinton, "Neural Networks for Machine Learning", https://www.coursera.org/course/neuralnets, 2012

[7] Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXi, 1408.5882 (2014).