Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts

By:

C. Mohan Shashanka (A20365258)

Sachet Misra (A20362816)

Problem Statement:

The sentiment analysis of short texts such as single sentences from Twitter, microblogging, etc. is difficult because of low information which is contextual. Solving such problems might require us to take an approach which is better than the bag of words approach which combine text content with prior information. This will enable us to understand and classify the sentiment of the sentences which are short.

Proposed Network:

The proposed network is a deep convolution neural network which exploits character to sentence level information to perform sentiment analysis of short texts. The system uses two convolution layers to extract relevant features from words and sentences of any size. This network can easily explore the richness of word embeddings produced by unsupervised learning. In the paper, this network is called CharSCNN (Character to Sentence Convolutional Neural Network).

Neural Network Architecture:

Given a sentence, CharSCNN computes a score for each sentiment label $\tau \in T$. In order to score a sentence, the network takes as input the sequence of words in the sentence, and passes it through a sequence of layers where features with increasing levels of complexity are extracted. The network extracts features from the character-level up to the sentence-level. The main novelty in our network architecture is the inclusion of two convolutional layers, which allows it to handle words and sentences of any size.

Initial Representation Levels

The first layer of the network transforms words into real-valued feature vectors (embeddings) that capture morphological, syntactic and semantic information about the words. We use a fixed-sized word vocabulary V_{wrd}, and we consider that words are composed of characters from a fixed-sized character vocabulary V_{chr}. Given a sentence consisting of N words $\{w_1, w_2, ..., w_N\}$, every word w_n is converted into a vector $u_n = [r_{wrd}; r_{wch}]$, which is composed of two sub-vectors: the *word-level embedding* $r_{wrd} \in R_{dwrd}$ and the *character-level embedding* $r_{wch} \in R_{clu\,0}$ of w_n. While word-level embeddings are meant to capture syntactic and semantic information, character-level embeddings capture morphological and shape information.

We used randomised number set for this as we were not able to train it for the whole dataset. After having run the word2vec for about 1 hour, we felt that we cannot work on this anymore.

Word and Character-Level Embeddings

Word-level embeddings are encoded by column vectors in an embedding matrix $W^{wrd} \in \mathbb{R}^{d^{wrd} \times |V^{wrd}|}$. Each column $W^{wrd}_i \in \mathbb{R}^{d^{wrd}}$ corresponds to the word-level embedding of the *i*-th word in the vocabulary. We transform a word *w* into its word-level embedding r^{wrd} by using the matrix-vector product:

$$r^{wrd} = W^{wrd} v^w \quad (1)$$

where v^w is a vector of size V^{wrd} which has value 1 at index *w* and zero in all other positions.

The matrix W^{wrd} is a parameter to be learned, and the size of the word-level embedding d^{wrd} is a hyperparameter to be chosen by the user.Given a character *c*, its embedding r^{chr} is obtained by the matrix-vector product:

$$r^{chr} = W^{chr} v^c \quad (2)$$

where v^c is a vector of size V^{chr} which has value 1 at index *c* and zero in all other positions.

The input for the convolutional layer is the sequence of character embeddings $\{r_1^{chr}, r_2^{chr}, ..., r_M^{chr}\}$. The convolutional layer applies a matrix-vector operation to each window of size k^{chr} of successive windows in the sequence $\{r_1^{chr}, r_2^{chr}, ..., r_M^{chr}\}$. Let us define the vector $z_m \in \mathbb{R}^{d^{chr}k^{chr}}$ as the concatenation of the character embedding *m*, its

$(k^{chr} - 1)/2$ left neighbors, and

its $(k^{chr} - 1)/2$ rightneighbors1:

$$z_m = \left(r^{chr}_{m-(k^{chr}-1)/2}, \ldots, r^{chr}_{m+(k^{chr}-1)/2}\right)^T$$

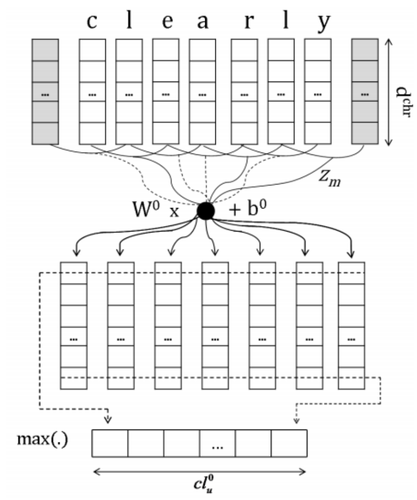

Figure 1: Convolutional approach to character-level feature extraction.

Sentence-Level Representation and Scoring

The second convolutional layer applies a matrix-vector operation to each window of size *kwrd* of successive windows in the sequence *{u*1*, u*2*, ..., uN}*. Let us define the vector *zn* ∈R(*dwrd*+*clu* 0)*kwrd* asthe concatenation of a sequence of *kwrd* embeddings, centralized in the *n*-th word2:

$$z_n = \left(u_{n-(k^{wrd}-1)/2}, \ldots, u_{n+(k^{wrd}-1)/2}\right)^T$$

Our network is trained by minimizing a negative likelihood over the training set *D*. Given a sentence *x*, the network with parameter set *ϑ* computes a score *sϑ*(*x*)*τ* for each sentiment label *τ* ∈*T*. In order to transform these scores into a conditional probability distribution of labels given the sentence and the set of network parameters *ϑ*, we apply a softmax operation over the scores of all tags *τ* ∈*T*:

$$p\left(\tau|x,\theta\right) = \frac{e^{s_\theta(x)_\tau}}{\sum\limits_{\forall i\in T} e^{s_\theta(x)_i}}$$

Taking the log, we arrive at the following conditional log-probability:

$$\log p\left(\tau|x,\theta\right) = s_\theta(x)_\tau - \log\left(\sum_{\forall i\in T} e^{s_\theta(x)_i}\right)$$

We use stochastic gradient descent (SGD) to minimize the negative log-likelihood with respect to ϑ:

$$\theta \mapsto \sum_{(x,y)\in D} -\log p(y|x,\theta)$$

where (*x, y*) corresponds to a sentence in the training corpus *D* and *y* represents its respective label.

DataSet:

We could not use the datasets present for many reasons. So we created our own datasets in order to run the program.

Vocabulary Creation:

We created a vocabulary of 5 words and characters. The following is the initialisation of the vocabulary:

Word vocabulary:

```
word_vocab = {'^*': 0, 'I': 1, 'am': 2, 'a': 3, 'good': 4,
'boy': 5}
```

Character vocabulary:

```
char_vocab = {'^': 0,'*': 1, 'I': 2, 'a': 3, 'm': 4, 'g': 5,
'o': 6, 'd': 7, 'b': 8, 'y': 9}
```

Sentences:

We are using two sentences to run our system as our dataset. The following are the two sentences used (from code with initialisation):

```
sent_1 = ['I', 'am', 'a', 'good', 'boy']
sent_2 = ['good', 'boy']
```

Code Explanation:

1. class CharSCNN -> Declares all the functions in the class

2. def __init__(self) -> hyper-paramenters declaration and initialization

3. def __init__(self, model_path) -> hyper-paramenters declaration and initialization
4. def load_model(self, model_path) -> load the model from folder
5. def save_model(self, model_path) -> saves the model in a new folder
6. def word_vector(self, word) -> Implementation for section 2.1.1 of the paper
7. def get_word_embedding(self, word) -> Implementation for section 2.1.2 of the paper
8. def get_char_embedding(self, word) -> Convolution Layer 1 : Convolution at the character level; Convolution Layer 1 : Max Pooling
9. def get_sent_representation(self, sent_array) -> Convolution Layer 2 : Max Pooling
10. def NN_Classifier(self, rxsent) -> Layer 1 &2 of NN Classifier
11. def softmax(self, x) -> Compute softmax values for each sets of scores in x.
12. def sgd_train(self, x_data, y_data) -> train the model and save parameters

Output

The output is the softmax scores of the sentence

Before Training

Sent 1 - **I am a good boy**
```
[[  7.65830706e-01]
 [  1.18327103e-04]
 [  2.34050967e-01]]
```

Sent 2 - **good boy**
```
[[  7.65830706e-01]
 [  1.18327103e-04]
 [  2.34050967e-01]]
```

After Training

Sent 1 - **I am a good boy**
```
[[  2.05235850e-04]
 [  9.99794056e-01]
 [  7.08616777e-07]]
```

Sent 2 - **good boy**
```
[[ 0.98240705]
 [ 0.00135633]
 [ 0.01623663]]
```

Initial values

Weight matrix for each convolution:

w0.txt w1.txt w2.txt w3.txt

Bias for each convolution:

b0.txt b1.txt b2.txt b3.txt

Final values (After training)

Weight matrix for each convolution:

w0.txt w1.txt w2.txt w3.txt

Bias for each convolution:

b0.txt b1.txt b2.txt b3.txt

The **Time taken** to run the code is **15.21 seconds**.

Challenges faced during Implementation:

1. Vocabulary creation: We needed a fixed length size of vocabulary which we could not create using the dataset.

2. Training the dataset took too much time to process. We have taken 15.21 seconds to process for training 2 sentences with 5 words and 9 characters. This is a long time for such a short dataset.
3. We could not find the right value of the learning rate. This caused us to iterate and use the learning rate to be 0.01.

References:

1. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. - C´ ıcero Nogueira dos Santos, Ma´ ıra Gatti
2. Automatic Text Reader Using Neural Networks - http://goo.gl/RXeQSh
3. Neural networks for sentiment analysis on Twitter - https://goo.gl/BoROYO
4. SentimentAnalysis Soumith Chintala - https://goo.gl/BoROYO
5. Andrei Alexandrescu and Katrin Kirchhoff. 2006. Factored neural language models.
6. Luciano Barbosa and Junlan Feng. 2010. Robust sentiment detection on twitter from biased and noisy data
7. James Bergstra, Olivier Breuleux, Fred ´ eric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, ´ Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler.