Text Categorization with Convolutional Neural Networks

Mohit Sinha
email: sinha052@umn.edu

Vaibhav Sharma
email: sharm205@umn.edu

Abstract

We implemented a 1-D temporal convolutional neural network for categorizing Hindi text. Convolutional neural net (CNN) is a variant of feed-forward neural network which can make use of the internal structure of data. For text classification with CNN, the main idea is to exploit word sequence (1-D) information to generate feature vectors through the convolutional layers (interleaved with pooling) which then is fed to a fully-connected feed-forward neural network for classification. Two very recent works (Zhang et al [1] and Johnson et al [2]) advocate the use of directly using raw characters for text classification instead of low-dimensional word-vectors. Our goal was to evaluate this claim by directly using text of utf-8 encoded Hindi characters without any additional resources for classification purpose. We compared it to the standard bag-of-words model to assess the performance.

1 Introduction

Text categorization is the task of assigning pre-defined classes to documents written in natural languages. It is an essential component in many applications such as web searching and sentiment analysis and therefore has garnered considerable attention from researchers. A key challenge in text classification is feature representation, which is commonly based on the bag-of-words model or some exquisitely human-designed pattern. A major drawback of the bag-of-words approach or its variations is the loss of word order. Furthermore, looking at the frequency of the words or some other statistics of structures is not how we perceive language. So, we were interested in an approach that exploits structure and word sequence to train the system to classify as this seems more close to our understanding of the language. However this is not to imply that is all there is to “understanding of a language”. Anyone who has ever tried to pick up a foreign language knows how incredibly hard and complex it can be to understand language.

We were hugely intrigued by the hypothesis that Zhang and LeCun make, “when trained from raw characters, temporal convolutional neural network is able to learn the hierarchical representations of words, phrases and sentences in order to understand text.” They mention that, “One immediate advantage from our dictionary-free design is its applicability to other kinds of human languages. Our simple approach only needs an alphabet of the target language using one-of-n encoding” and emphasize “CNNs neither require knowledge of words nor syntax structures.” Now, these are grand claims. We wanted to see if these claims held water when a character-level input of Hindi text was provided as an input to the CNN. In the paper, when testing on the Chinese corpus, Zhang and LeCun phonetically transliterated Chinese to English characters. We went a step ahead and ignored even this layer and simply used utf-8 encoded character to use only “Hindi characters” to classify. We did not achieve as much information from our trial runs to conclusively validate or invalidate the claims and this work should be seen as a first step towards it. We were mostly constrained to a small dataset (9724 articles, 2 classes) which we built by scraping a newspaper archive and suffers from a major drawback that it is severely skewed.

Johshon et al [2] also use 1-D temporal CNN for text categorization and consider two variants : sequential-CNN and bag-of-words-CNN. Sequential-CNN considers $p|V|$ dimensional representation, where $|V|$ is the cardinality of vocabulary and p is a width that is fixed apriori. For instance, $p = 2$ the text region of two words will be represented by concatenating two vectors, each of which is a 1-of-$|V|$ encoded word. And the this window of 2 is slid to the right to generate another text region vector of size 2. As the dimensionality of such representation can get very high, they consider a bag-of-words-CNN, which considers $|V|$ dimensional representation. Here, both the words in the text region (for the $p = 2$ example) are represented in the same $|V|$ dimensional vector.

The manuscript is organized as follows. We begin by providing a brief overview of convolutional neural networks and its application for text classification in Section II. Following which, we provide details of our experiments and observations in Section III. Finally, we conclude our paper by highlighting a few directions for future work in Section IV.

2 Convolutional Neural Nets

We begin by a brief overview of convolutional neural nets which are biologically-inspired variants of the multi-layered feedforward neural networks. These fall under the so-called *deep learning* architectures which are composed of multi-level of nonlinear operations and are posited to be suitable for learning high-level abstractions such as vision, language and other AI tasks. [3]. In the implementation that we consider the neural network is 9 layers deep with 6 convolutional layers (interleaved with pooling) and 3 fully-connected layers.

Basically, convolutional neural nets can be thought of as a kind of neural network that uses many identical copies of the same neuron i.e. these neurons have the same parameters, known as convolutional *weight-tying*. This allows the network to have large number of neurons while keeping the parameters that need to be learned comparatively small [4]. This trick is roughly analogous to abstraction of functions in mathematics. [5] So, a convolutional neural net can learn a neuron once and use it in many places. This identical group of neurons look at local data segments (in the sense of space or time) to compute certain *features*. In the case of text, this convolutional layer tries to derive features based on word sequences (to extract contextual information in the text). The output of this *convolutional layer* is generally interleaved with a pooling layer. The pooling layer takes small blocks from the convolutional layer and subsamples it to produce a single output from that block. There are many ways this can be done, such as pooling the average or the maximum. A max-pooling layer, the one that we consider here, takes the maximum of features. Thus, pooling layers kind of "zoom out" and allow later convolutional layers to work on larger sections of the data, because a small patch after the pooling layer corresponds to a much larger patch before it. These pooling layers do not do any learning themselves and only reduce the size of the problem. Next, we elaborate on implementation of each of the layers which is identical to the implementation in [1].

2.1 1-D Convolution

1-D convolution forms the central component of the central layer. Suppose we have a discrete input function $f : [1,l] \to \mathbb{R}$ and a discrete kernel function $g : [1,k] \to \mathbb{R}$ then the convolution

$h : [1, \lfloor l-k+1 \rfloor] \to \mathbb{R}$ is defined as:

$$h(x) = \sum_{y=1}^{k} g(y) f(x-y+c) \tag{1}$$

where $c = k$ is an offset constant. One way to think about this is that we're sliding the kernel over the input sequence. For each position of the kernel, we multiply overlapping values of the kernel and the word sequence together, and add up the results.

2.2 Thresholding

The thresholding function $h(x) = \max(0, x)$ is always applied to the output of the convolutional or linear (of the fully-connected part) layer. This is inspired from [6] and [7] where they show adding this thresholding function improves the performance of the hidden units.

2.3 Temporal Max Pooling

Given the discrete input function $f : [1, l] \to \mathbb{R}$, the max-pooling function $p : [1, \lfloor l-k+1 \rfloor] \to \mathbb{R}$ is given by :

$$p(y) = \max f(y-x+c) \quad \forall x \in [1, k]. \tag{2}$$

where $c = k$ is an offset. The max pooling function allows us to train deep models by reducing the size of the problem by introducing sparsity over the hidden representation by erasing all the non-maximal values in non-overlapping regions. [8]

2.4 Backpropagation

We use stochastic gradient descent (SGD) to run backpropagation. It is mainly motivated by the high cost of running backpropagation over the full training set. SGD follows a negative gradient of the objective after only seeing a few examples and can therefore overcome the high cost and lead to a faster convergence comparatively. The new update is given by [9]:

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)}) \tag{3}$$

where $(x^{(i)}, y^{(i)})$ represent a training example, α is the learning rate and $J(\theta)$ represents the error in terms of the weights θ, which in our case is cross-entropy error on which we elaborate in the next error. This parameter update in SGD is computed with respect to a minibatch as it reduces variance in the parameter update and lead to a more stable convergence. The minibatch size in our is 128.

Furthermore, the objectives of deep architectures tend to have long "narrow ravines" near their local optima and can cause SGD to oscillator around the optimum and thus lead to a slower convergence. Momentum is one method to remedy this. The momentum update is given by [9]:

$$\begin{aligned} v &= \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \\ \theta &= \theta - v \end{aligned} \tag{4}$$

v is the so-called velocity vector which has same dimensions as θ and γ determines gradients of how many past iterations are incorporated in the current update. We use $\gamma = 0.9$ and $\alpha = 0.01$ which is halved after every 3 epoches for 10 times.

One final point regarding SGD is the order in which we present the data to the algorithm. It can potentially bias the gradient and lead to poor convergence. To avoid this, we randomly shuffle the data prior to each epoch of training [9].

2.5 Cross-Entropy Loss

The cross entropy loss function computes is given by

$$J(\theta) = \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{N} ln(o_i) t_i \tag{5}$$

where n is the number of training samples, N is the length of the output vector and target vector, o_i represents the ith predicted output and t_i is the ith entry of the target vector. Thus, the cross-entropy is positive, and tends toward zero as the neuron gets better at computing the desired output for all training inputs. While the mean-squared error also has these properties, the learning slows down for the quadratic error as it reduces which is avoided by this cross-entropy error.

3 Case Study : Hindi Language Dataset

3.1 Dataset

We built a Hindi Language dataset by scraping the archives of Hindi newspaper *Dainik Bhaskar* [10]. The dataset is grouped into many regional categories which have news from that local region such as *Ratlam* (a small town in India) as well as national and international categories which have news of national importance and from around the globe. We partitioned our dataset into two classes based on this : Regional and Non-Regional. Our dataset has 9724 articles with headlines and text from the article out of which we use 80% for training and 20% testing. Our training dataset is highly skewed with 8928 entries from Regional class and 596 entries from Non-Regional class. We however took equal entries of both classes to create the test data. *Character quantization:* We used regular expression to clean the data and contain only valid Hindi characters with spaces. Hindi characters were not transliterated to English and as read as "utf-8" encoded characters. The Hindi alphabet (in utf-8 form) was provided and was encoded to generate 1-of-n encoding for every character.

3.2 Torch Deep Learning Library

Torch is a scientific computing framework, which is based on Lua language, with packages of neural networks, optimization and other prominent machine learning algorithms. We used the open-source code provided by Zhang and LeCun for their implementation of 1-D temporal convolutional neural network. We changed parameters and alphabet to suit our implementation. As we implemented this on CPU, the training time was huge. For even a small dataset with 9724 articles, one run of training and testing took more than 10 hours (with 100 epochs) on a 128 GB 40-core cluster.

3.3 Bag-of-Words Model

As bag-of-words model is one of the standard tools used for text categorization, we thought of comparing the performance of our very naive implementation of convolutional neural net with it. We built of a vocabulary of $16,325$ words which came more than 3 times in each of the article.

Then, we created a *term document matrix* which counts the number of occurrences of each words in the vocabulary for every article. Indeed, the matrix is very sparse and this has been mentioned in literature as a cause of deterioration in the classification accuracy []. We used the *feature* matrix so generated to do a two-class classification using a Support Vector Machine (SVM) with a linear kernel. We used scikit-learn python library to implement SVM.

3.4 Results

Both CNN and Bag-of-words did not perform well, both only doing slightly better than a random choice. However, even this naive implementation of CNN marginally outperforms Bag-of-words model. However, the skewed classes in the training set leads to major predictions to be of that class in both the classification algorithms. In both the convolutional net implementations in [1, 2], comparatively huge datasets were provided for the classification and were trained for larger number of epochs. The results are summarized in the tables below.

CNN

Epochs	Train Error	Test Error
1	6.7%	42.07%
100	6.6%	42.15%
200	6.25%	42.15%

Bag-of-words

Train Error	Test Error
99.97%	48%

Table 1: Performance on the newspaper Dainik Bhaskar data

4 Conclusions

Deep learning models have garnered much attention due to their overwhelming success in computer vision and speech recognition. How good these models are for text categorization is an interesting question to ask and formed the mainstay of this project. We implemented a 1-D temporal CNN to perform a two-class text classification and showed that CNN marginally outperformed the bag-of-words based SVM classifier. Next, we intend to perform a multi-class classification with a bigger and balanced dataset to see if the CNN model can perform close to the performance in [1]. Further, it would be interesting to draw comparisons with the variations of CNN model proposed in [2].

References

[1] X. Zhang and Y. LeCun, “Text understanding from scratch,” *arXiv preprint arXiv:1502.01710*, 2015.

[2] R. Johnson and T. Zhang, “Effective use of word order for text categorization with convolutional neural networks,” *arXiv preprint arXiv:1412.1058*, 2014.

[3] Y. Bengio, “Learning deep architectures for ai,” *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[4] J. Ngiam, Z. Chen, D. Chia, P. W. Koh, Q. V. Le, and A. Y. Ng, “Tiled convolutional neural networks,” in *Advances in Neural Information Processing Systems*, pp. 1279–1287, 2010.

[5] `http://colah.github.io/posts/2014-07-Conv-Nets-Modular/`.

[6] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, 2010.

[7] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?,” in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2146–2153, IEEE, 2009.

[8] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, “Stacked convolutional auto-encoders for hierarchical feature extraction,” in *Artificial Neural Networks and Machine Learning–ICANN 2011*, pp. 52–59, Springer, 2011.

[9] `http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/`

[10] `http://www.bhaskar.com/archives/`.