JMLR: Workshop and Conference Proceedings 30:1–6, 2016 ICGI 2016

Sequence Prediction Using Neural Network Classifiers

Yanpeng Zhao ZHAOYP1@SHANGHAITECH.EDU.CN
Shanbo Chu CHUSHB@SHANGHAITECH.EDU.CN
Yang Zhou ZHOUYANG1@SHANGHAITECH.EDU.CN
Kewei Tu TUKW@SHANGHAITECH.EDU.CN
School of Information Science and Technology, ShanghaiTech University, Shanghai, China

Abstract

Being able to guess the next element of a sequence is an important question in many fields. In this paper we present our approaches used in the Sequence Prediction ChallengE (SPiCe), whose goal is to compare the different approaches to that problem on the same datasets. We model sequence prediction as a classification problem and adapt three different neural network models to tackle it. The experimental results show that our neural network based approaches produce better overall performance than the baseline approaches provided in the competition. In the actual competition, we won the second place using these approaches.

Keywords: Sequence prediction, classification, neural network classifiers

1. Introduction

SPiCe is an online competition about guessing the next element of a sequence[1], with the goal to compare different approaches to the sequence prediction problem on the same datasets. Participants are asked to learn a model on the training set consisting of whole sequences and rank the potential next symbols for a given prefix of a test sequence. Datasets used in the competition are either real-world data from different fields (natural language processing, biology, signal processing, software verification, etc.) or synthetic data especially created for SPiCe. All the information about the data sources is unknown to the participants.

For each problem instance in the competition, there are three sections of the data. One section is the training set containing complete sequences on which the model is learned. The other two sections are test sets including a public one that can be seen as a validation set and a private one on which the final score is calculated. After learning a model on the training set, participants need to submit a ranking of the top 5 most probable next symbols (including a special end-of-sequence symbol) for a given prefix of a test sequence. Baseline approaches such as 3-Gram and Spectral Learning (SL) are provided.

We model sequence prediction as a classification problem, in which the input to the classifier is the prefix and the class label is the next symbol. Since neural networks are impressive for their ability of transforming the original input space into a feature space in which it is easier to separate different classes and thus produce good classification performance, we use neural network as the classifier. To further explore the performance of different neural network models for sequence prediction, we adopt three different neural

1. http://spice.lif.univ-mrs.fr/index.php

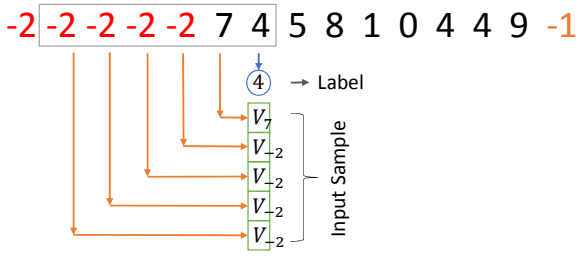

Figure 1: An example explaining how to transform the original discrete symbol sequence into the inputs to neural network classifiers. In this example, -2 represents the symbol DUMMY and -1 represents the symbol END. Assume we predict from previous $k = 5$ symbols, we first left pad the sequence with five -2 and append -1 to its end. For the symbol $s = 4$, we sequentially concatenate the word vectors $V_{-2}, V_{-2}, V_{-2}, V_{-2}, V_7$ of its previous 5 symbols to get the input sample and regard '4' as the class label.

network models, multilayer perceptron (MLP), convolutional neural network (CNN), and long short term memory (LSTM) networks.

The rest of the paper is organized as follows. Section 2 introduces the three different neural network models. Section 3 provides the details of our implementation and the comparisons between the results achieved by different models. Section 4 concludes this paper.

2. Neural Network Models

Neural networks have been achieving great success in natural language processing. Work in this field with neural networks is mostly based on learning continuous vector representations of words (Bengio et al., 2003; Mikolov et al., 2013). By representing words in a low-dimensional continuous embedding space, continuous word vectors can encode semantic and syntactic features of words and are supposed to be superior to conventional methods that treat words as discrete symbols or use discrete word representations such as one-hot vectors.

Our neural network based approaches to sequence prediction are developed on top of the continuous word vectors that are trained using Continuous Bag-of-Words Model (CBOW) proposed and implemented by Mikolov et al. (2013). With pretrained word vectors, we transform the original discrete symbol sequences as follows. Given a sequence of length L, we first right pad it with the symbol END, which indicates the end of the sequence. Assume that we predict the next symbol from the previous k symbols, we also need to left pad the sequence with k DUMMY symbols, so that we can construct the inputs to the classifier when predicting the first k symbols of the sequence. We then construct the training set for the neural network classifier: for each symbol s in the sequence (other than the DUMMY symbols), we sequentially concatenate the word vectors of its previous k symbols as the input and regard s as the label. Figure 1 gives an example of this procedure.

We use three different neural network classifiers as described below.

1. **MLP** is a feedforward artificial neural network model that consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Each node in intermediate layers (hidden layers) is a neuron with a nonlinear activation

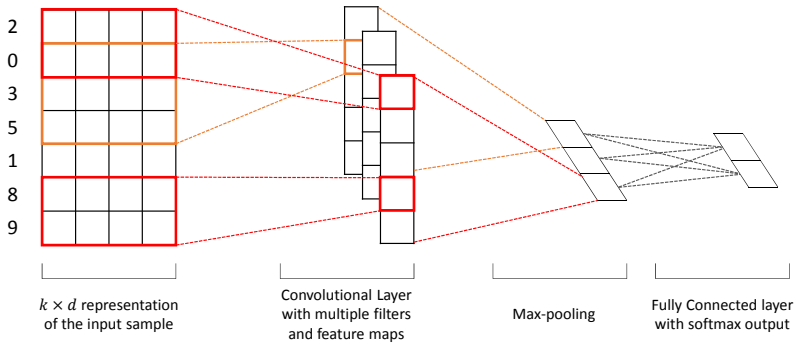

Figure 2: CNN model architecture adapted from Kim (2014). The input is a symbol sequence of length k represented by k stacked row vectors. Each row vector of dimension d is the continuous vector representation of the corresponding symbol.

function (tanh, ReLU, or sigmoid). The output layer is a softmax layer that implements softmax function for classification and cross-entropy cost function is used while training the MLP model.

2. **CNN** is a feedforward neural network that contains convolutional layers with a nonlinear activation function such as ReLU. Besides, CNN could also contain pooling layers that are usually used immediately after each convolutional layer. In SPiCe we employ the CNN model proposed by Kim (2014). For simplicity, the input layer only contains a static channel, and we do not enforce the L2-norm constraint on the weight vectors that parametrize the softmax function, which Zhang and Wallace (2015) found generally did not improve performance much. Figure 2 shows our CNN model architecture for sequence prediction.

3. **LSTM** is a special kind of recurrent neural network (RNN) architecture that was proposed by Hochreiter and Schmidhuber (1997) to combat the vanishing gradient problem. Featured with the gating mechanism, LSTM is well-suited to model long-term dependencies and has been successfully applied to language modeling. In SPiCe we use the standard LSTM architecture, and the hidden state of the LSTM model is fed to the logistic regression classifier.

3. Experiments and Results

We applied our neural network models to the SPiCe datasets and compared them with 3-Gram and SL baseline approaches. For each problem instance, we first learned vector representations of symbols (including the end-of-sequence symbol) using `word2vec`[2] on the training set. We set the vector dimension $d = 30$, which we found worked well. For symbols appearing in the test set but missing in the training set as well as the DUMMY symbol, we randomly fixed the vector of each of them. With learned word vectors, we then transformed the original datasets to construct the training set for the neural network models using the procedure described in Section 2, where k is set to 15. Each prefix in the test set is also padded with the symbol DUMMY on the left for a total length of 15 if it contains fewer

2. https://code.google.com/archive/p/word2vec/

than 15 symbols, and is truncated otherwise. Finally, we trained neural networks on the transformed training set and tested them on the public test set.

The MLP model used in our experiments has an input layer of dimension 450 (15 input symbols with a 30-dimension vector for each symbol), and two hidden layers with 750 and 1000 neurons respectively. The higher dimensions of the hidden layers mean that the inputs are consecutively projected to a higher-dimensional space twice. One reason for this choice is that samples could be easier to separate in the high-dimensional space than in the low-dimensional space when we get a large training set (note that the size of the transformed training set is very large since there is one training sample for each symbol in the original training set). The other reason is that adding further layers into the MLP does not improve the results but slows down training. Our MLP model used ReLU as the activation function.

As to CNN, we used filter windows (height) of 10, 11, 12, 13, 14, 15 with 200 feature maps each. The dropout rate was set to 0 and the activation function ReLU was used. For LSTM, we used tanh as the activation function. The number of time steps was set to 15 (number of input symbols) and the size of the output vector was 32. Both MLP and CNN were implemented in MxNet (Chen et al., 2015) and trained using SGD. LSTM was implemented in Tensorflow (Abadi et al., 2016) and trained using RMSProp. Other experiment settings can be found in Appendix A.

All our neural network models were trained on a CentOS 7.2 (64Bit) server equipped with Intel Xeon Processor E5-2697 v2 @ 2.70GHz and four Tesla K40Ms. Training and evaluating all the models on all datasets using multiple processes cost less than 18h (this is a rough estimation since we did not record the exact time).

We also tried the Weighted n-Gram Model (WnGM), which averaged predictions from n-Gram models with different n. We set n to 2, 3, 4, 5, 6 and associated them with weights 0.3, 0.2, 0.2, 0.15, 0.15 respectively. For 3-Gram and SL baseline approaches, we used the default settings in the source codes provided by SPiCe.

Table 1 shows total scores achieved by our models discussed above and the baseline approaches (the scores of different models on individual datasets are shown in Appendix A). We can see that all our models produce better scores than baseline approaches. MLP, the simplest neural network model, produced the best score. We believe it is because MLP makes the best use of the order of the word vectors in the input. We also experimented with the sum, instead of the concatenation, of word vectors as the input and found that hurt the model performance. The mediocre performance of CNN may be due to that the original model architecture is proposed for sentence classification, which is different from our task, and thus directly simplifying the model and applying it to sequence prediction may not be appropriate. Iyyer et al. (2015) found that syntactically-ignorant deep averaging network (DAN) could perform comparably with more complicated neural networks that explicitly model semantic and syntactic compositionality. Therefore, we conjecture that a deeper network with proper input might be helpful.

Table 1: Total scores achieved by different models on public test sets.

Problem ID	3-Gram	SL	MLP	CNN	WnGM	LSTM
Total Score	8.666182	7.443699	9.802395	9.593389	9.236514	9.325444

When participating in the competition, to obtain a better overall score, we tuned model parameters by the total score achieved on public test sets. For each problem, the best model that produced the best score on the corresponding public test set was used. The total score we achieved on private test sets is **10.160324**. Note that this score is different from that (10.019871) we achieved online, since we tried new models and tuned them after the competition. You can find what model and model parameters we used for each problem in Appendix B.

4. Conclusion

In this paper we modeled sequence prediction as a classification problem and developed neural network based approaches to it. Taking advantage of continuous word vectors, we transformed original discrete symbol sequences to continuous inputs for neural networks and employed three different neural network models, MLP, CNN, and LSTM, for classification. The experimental results show that our approaches produced a better overall performance than the 3-Gram and Spectral Learning baseline approaches. One shortcoming of our approaches is that they do not explicitly make use of the potential grammatical structures within the sequences. A direction for future work is to integrate neural networks into probabilistic grammatical models in sequence prediction.

References

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155, 2003.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Mohit Iyyer, Varun Manjunatha, and Jordan Boyd-Graber. Deep unordered composition rivals syntactic methods for text classification. 2015.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

Appendix A. Detailed Results on Public Test Sets

Table 2 presents scores by different models on each public test set. For both 3-Gram and SL baseline approaches, we used the default settings. For MLP, CNN, and LSTM, we used adaptive learning rate with initial values of 0.1, 0.002, and 0.1 respectively, and these three models were trained with 5, 10k, and 40k epoches respectively.

Table 2: Results achieved by different models on public test sets.

Problem ID	3-Gram	SL	MLP	CNN	WnGM	LSTM
1	0.842825	0.874877	0.894517	0.877351	0.841534	**0.903276**
2	0.817825	0.87404	0.904835	0.897068	0.817638	**0.910337**
3	0.775519	0.828139	0.86131	0.850179	0.779835	**0.874903**
4	0.537655	0.431704	0.528351	0.531538	**0.588389**	0.422016
5	0.598158	0.292452	**0.761046**	0.743044	0.690147	0.685619
6	0.674949	0.429366	0.705684	0.667631	0.759378	**0.769689**
7	0.442091	0.324983	**0.685551**	0.6506	0.503653	0.5767
8	0.590873	0.491134	**0.615426**	0.603933	0.61359	0.574858
9	0.858806	0.741451	0.921942	0.904356	0.896695	**0.952874**
10	0.413004	0.256069	**0.556514**	0.553421	0.44162	0.528451
11	0.38855	0.329707	**0.51323**	0.499009	0.441006	0.443673
12	0.700295	0.55699	0.715182	0.667296	**0.747883**	0.664733
13	0.435876	0.328572	**0.524219**	0.521	0.492081	0.385525
14	0.338599	**0.406484**	0.350904	0.357211	0.362929	0.366087
15	0.251157	**0.277731**	0.263684	0.269752	0.260136	0.266703
Total Score	8.666182	7.443699	9.802395	9.593389	9.236514	9.325444

numbers in bold indicate the best scores achieved on public test sets.

Appendix B. Results on Private Test Sets

Table 3 presents final scores achieved by the best models on each private test set. Unless otherwise specified, model configurations in this section are the same as that discussed in Section 3. We used WnGM on test sets 4, 12, MLP model on test sets 5, 7, 8, 10, 11, 13, and LSTM model on test sets 1, 2, 3, 6, 9. For test sets 14, 15, we tuned the other LSTM model on public test sets that performed best on these two problems. The new LSTM model used adaptive learning rate initialized to 0.02, the size of its output vector was 2000 and the model was trained with 2k epochs.

Table 3: Final scores achieved by the best models on each private test set.

Problem ID	1	2	3	4	5	6	7	8
Score	0.905435	0.909176	0.881451	0.579369	0.777499	0.754421	0.684236	0.629736
Problem ID	9	10	11	12	13	14	15	Total Score
Score	0.953808	0.52535	0.511706	0.749253	0.520801	0.470133	0.307950	**10.160324**