Advances in Computer Science Research, volume 82
7th International Conference on Social Network, Communication and Education (SNCE 2017)

Online Learning Sum-Product Networks for Language Modeling

Yu Zhong Zhang[1, a *]

[1]School of Mathematics and Information Technology, Yuxi Normal University, Yuxi, Yunnan, China

[a]zh1011@yxnu.net

*The Corresponding author

Keywords: Sum-product networks; Language models; Oline learning; Deep Learning

Abstract. Sum-product networks (SPNs) have recently proposed as an remarkable representation due to their dual view as a special deep neural network with clear semantics and a probabilistic graphical model for which inference is always tractable. SPNs have been successfully applied in Computer Vision and Natural Language Processing. We used the hidden layers of SPNs to model complex dependencies among words and we used SPNs online learning algorithm to improve model learning speed and SPNs structure learning algorithm to improve modeling capabilities. Our empirical comparisons with other previous language models indicate that our online learning SPNs has better performance.

Introduction

Language model is the most important part of natural language processing, and has been widely used in text classification, machine translation, speech recognition, and information retrieval. In general, they model the probability distribution of the sequence of words w_1^m in a transcription as $P(w_1^m) \approx \prod_{k=1}^{m} P(w_k \mid w_{k-n+1}^{k-1})$ where w_i^j is a sequence of words $w_i, w_{i+1}, \cdots, w_{j-1}, w_j$ .A basic language model is the N-gram model，which predicts the next word based on previous $n-1$ words. However, N-gram model requires a considerable amount of training text to determine the parameters of the model. When n is large, the model's parameter space is too large. So the neural network language model and other high-level language model has gradually become a hot research.

More complex language models include the log-bilinear model [1], feedforward neural networks [2], and recurrent neural networks [3]. The log-bilinear model is a probabilistic graphical model that encodes the dependencies between all pairs of words in a vocabulary. It performs well but can not take advantage of the rich information that exists in three or more words. Bengio et al.[2] used a three layer feedforward neural networks to build a language model. They used word distributed representation to solve the sparse data on the impact of statistical modeling, while overcoming the dimension of the model parameters of disaster problems. Neural network uses only one hidden layer to capture the dependencies among words, without incurring too large a penalty in training time. Mikolov et al.[3] used a recurrent neural networks as a language model. An RNN is similar to a feedforward neural network in having an input layer of words that is connected to a hidden layer, which in turn is connected to an output layer representing a probability distribution over words. The biggest advantage of the recurrent neural network is that it can really make full use of all the above information to predict the next word. The use of recurrent neural network is very difficult to use, if the optimization is not good, long distance information will be lost, and even can not reach the window to see the effect of a number of words. In order to improve the RNN language models performance, Mikolov et al. [3] augmented them with contextual information via latent Dirichlet allocation [4] to obtain state-of-the-art results.

In recent years, there has been renewed interest in learning rich, tractable models, on which exact probabilistic inference can be performed in polynomial time(e.g. Sum-Product Networks [5]). Wei-Chen Cheng et al. [6] has firstly used SPNs to function as a language model. They proposed SPNs is able to encapsulate multiple hidden layers while maintaining tractable inference and training. Empirically, it achieves better predictive accuracy than the aforementioned language model. But they used a predefined SPNs architecture that can not fit the different types of data. And they used the

discriminate batch parameter leaning, this kind of parameter learning method causes the model to learn slower. In this paper, we use the SPNs structure learning algorithm to generate a model structure directly from the input data and we used SPNs online parameter learning algorithm to speed up model leaning time.

Sum-Product Networks

Sum-product networks (SPNs) were first proposed by Poon & Domingos [5] as a new type of deep architecture include a rooted acyclic directed graph with interior nodes that are sum nodes and product nodes while the leaves are tractable distributions, including Bernoulli distributions for discrete SPNs and Gaussian distributions for continuous SPNs. The edges emanating from sum nodes are labeled with non-negative weights w. An SPN encodes a function $f(X=x)$ that takes as input a variable assignment $X=x$ and produces an output at its root. This function is defined recursively at each node n as follows:

$$f_{root(x)}(x)=\begin{cases}\Pr(X_n=x_n) & \text{if is Leaf(n)}\\ \sum_i w_i f_{child(n)}(x) & \text{if is Sum(n)}\\ \prod_i f_{child_i(n)}(x) & \text{if is Product(n)}\end{cases} \quad (1)$$

Here, $X_n=x_n$ denotes the variable assignment restricted to the variables contained in the leaf n.If none of the variables in leaf n are instantiated by $X=x$ then $\Pr(X_n=x_n)=\Pr(\varnothing)=1$. Note also that if leaf n contains continuous variables, then $\Pr(X_n=x_n)$ should be interpreted as $pdf(X_n=x_n)$.

An SPN can also be viewed as encoding a joint distribution over the random variables in its leaves when the network structure satisfies certain conditions. These conditions are often defined in terms of the notion of scope.

Definition 1 (Scope). The scope(n) of a node n is the set of variables that are descendants of n.

Definition 2 (Completeness) An SPN is complete if all children of the same sum node have the identical scope (i.e, $\forall c \in children(Sum),\quad scope(c)=scope(c')$)

Definition 3 (Decomposability) An SPN is decomposable if all children of the same product node have disjoint scope(i.e $\forall c,c' \in children(Sum)$ and c ≠ c', $scope(c)\wedge scope(c')=\varnothing$)

The decomposability here allows us to interpret the product nodes as computing factored distributions with respect to disjoint sets of variables, thus ensuring an efficient distribution of the product within the sub-level. Similarly, completeness allows us to interpret the sum node as a mixture of distributions of children encoded because they all have the same range. Each child is a mixture component whose mixing probability is proportional to its weight. Thus, in a complete and decomposable SPN, the sub-SPN according to each node can be interpreted as encoding (non-normalized) joint distribution over its range. And as long as all the nodes of SPNs set value 1, SPNs can effectively calculate the partition function so that it can perform a exact inference in the size of networks.

When all sum nodes are complete and all product nodes are decomposable, an SPN is called valid, which ensures that the SPN value for some evidence is proportional to the probability of the evidence (i.e, $\Pr(x)\propto f_{root}(x)$). This means that we can answer the inference queries $\Pr(x_1\mid x_2)$ by computing two SPN evaluations $f_{root}(x_1)$ and $f_{root}(x_2)$:

$$\Pr(x_1\mid x_2)=\frac{f_{root}(x_1,x_2)}{f_{root}(x_2)} \quad (2)$$

SPNs can be understood as a deeply computable probabilistic graphical model, unlike Bayesian networks and Markov networks where inference may be exponential in the size of the network, inference in SPNs is in time linear in the size of the network. SPNs are deep neural networks restricted to sum and product operators. It is easy to see that the node calculates a linear combination of its child nodes. A product node can be interpreted as the sum of its child nodes in the log domain. Thus, the

product network can be viewed as a neural network with logarithmic and exponential activation functions.

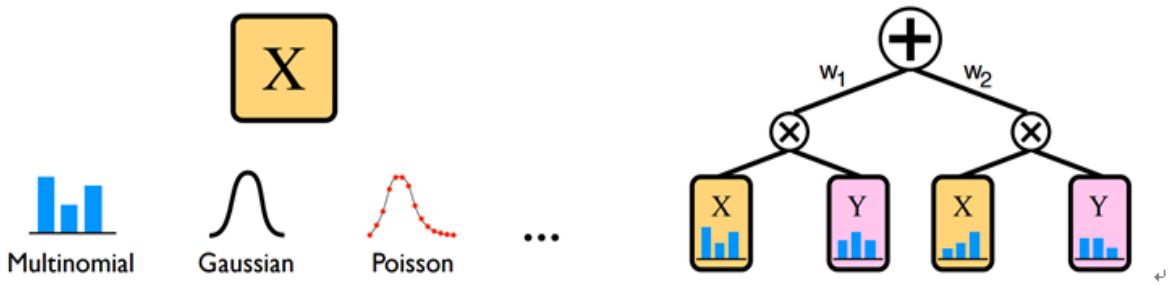

Figure 1. Finite A simple SPNs architecture with the distribution of leave nodes

SPNs Parameter Learning

The weights of an SPNs are its parameters. They can be estimated by maximizing the likelihood of a dataset (generative training) [5] or the conditional likelihood of some output features given some input features (discriminative training) by Stochastic Gradient Descent (SGD) [7]. Wei-Chen Cheng et al. [6] used the SGD to train the SPNs parameter. Their parameter learning methods are not suitable for large text dataset and streaming text data. In this paper, we used SPNs online learning algorithm to improve model learning speed. we used the online Bayesian moment matching technique [8] for SPNs parameter learning. Bayesian learning begin wih a prior $\Pr(w)$ over the weights, learning corresponds to computing the posterior distribution $\Pr(w \mid data)$ based on the data observed according to Bayes' theorem:

$$\Pr(w \mid data) \propto \Pr(w)\Pr(data \mid w) \tag{3}$$

Since the data consists of a set of instances $X^{1:N} = \{x^1, \ldots, x^N\}$, We can rewrite the Bayesian theorem in a recursive way, which helps to incremental online learning:

$$\Pr(w \mid x^{1:n+1}) \propto \Pr(w \mid x^{1:n})\Pr(x^{n+1} \mid w) \tag{4}$$

The parameters of the SPN are made up of weights associated with the edges emitted from each and the nodes. The first step is to define a prior over the weights. We start with a prior that consists of a product of Dirichlets with respect to the weights $w_{i.} = \{w_{ij} \mid j \in children(i)\}$ of each sum node i:

$$\Pr(w) = \prod_{i \in SumNodes} Dir(w_{i.} \mid \alpha_i) \tag{5}$$

The posterior is obtained by multiplying the prior by the likelihood $f_{root(x)}$ of each data instance:

$$\Pr(w \mid x^{1:n+1}) \propto \Pr(w \mid x^{1:n}) f_{root}(x^{n+1}) \tag{6}$$

Moment matching is a popular technique to estimate the parameters of a distribution based on the empirical moments of a dataset. Moment matching to approximate mixtures of products of Dirichlets obtained after processing each data instance by a single product of Dirichlets. When SPN is a tree, all time linear moments can be calculated simultaneously on the size of the network. The key is to compute two coefficients $coef_i^0, coef_i^1$. Once we have the coefficients, we can compute each moment as follows:

$$M_{posterior}(w_{ij}^k) = \int_{w_{i.}} w_{ij}^k Dir(w_{i.} \mid \alpha_{i.})(coef_i^0 + coef_i^1 \sum_{j'} w_{ij'} V_{j'}(e(x)))dw_{i.} \tag{7}$$

SPNs Structual Learning

Since it is difficult to specify network structures for SPNs that satisfy the completeness and decomposability properties, several automated structure learning techniques have been proposed [9] [10] [11]. The most prominent general SPNs structure learning algorithm was proposed by Gens [9]. Their structure learning techniques are top down approaches that alternate between instance clustering to construct sum nodes and variable partitioning to construct product nodes.

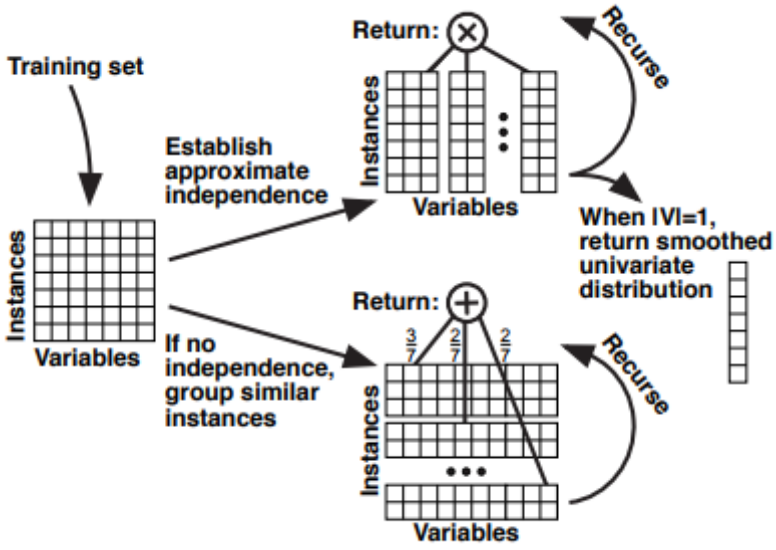

Figure 2. Finite Gens recursive algorithm for learning structure of SPNs

But their structure learning algorithms cluster instances without ensuring that the clustering respects context-specific independences (independences that hold only among instances of a specific context or cluster). We used a new SPNs structure learning method that are based on recursively extracting rank-one (SVD) submatrices from data. The search for rank-1 submatrices occurs jointly over variables and instances, whereas clustering and identifying independencies are two unrelated procedures.

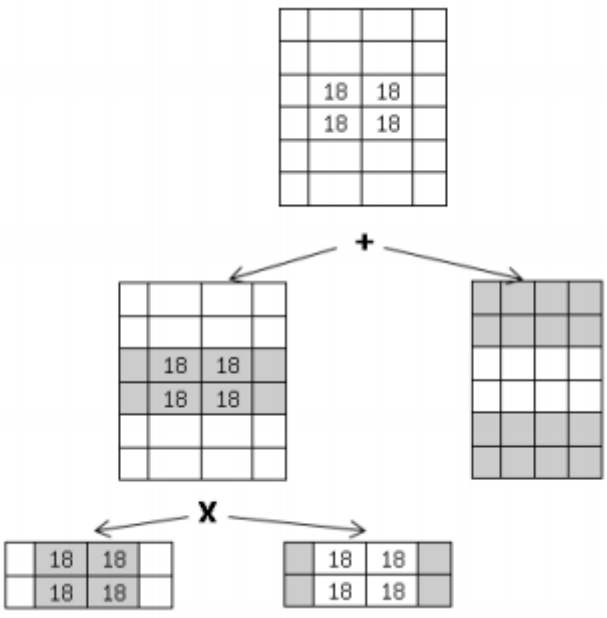

Figure 3. Finite SVD based SPNs structure learning

SPNs for Language Modeling

We used SPNs for language modeling. Using a query variable to represent a predict word, we use its previous N words as evidence in SPNs. Each previous word is represented by a K-dimensional vector where K is the number of words in a vocabulary, we used a bag-of-word(BoW) model to represen a word vector. Each vector has exactly 1 at the index corresponding to the word it represents, and is 0 in other places. When we predict the i^{th} words, we have a vector v_{i-j} ($1 \le j \le N$) at the leaf layer for each of the previous N words.

Firstly, we used SPNs structure learning algorithms to generate a model structure from input train dataset matrix. The input data matrix is a word vector matrix that contains the relationship between the words. The whole process of taking into account the input data of the original relevant information. Each hidden layer of SPNs can mixture the word information. We used the structure learning algorithm of SPNs based on SVD. This method is more general than the Wei[6] predefined SPNs language model structure, which can fit very well into different types of input data and not bother to design each layer of model structure. Secondly, we used online parameter learning algorithm of SPNs to train the weight of network. Using SPNs online Bayesian moment matching technique can greatly improve the speed of model training . Finally, we have a SPNs language model, we use the MAP [7] inference method to predict the conditional probability of the i^{th} word given its previous N words.

Experiments

We performed our experiments on the commonly used Penn Treebank corpus [12], and adhered to the experimental setup used in previous work[3], We used sections 0-20, sections 21-22, and sections 23-24 respectively as training, validation and test sets. We treated punctuation as words, and used the 10,000 most frequent words in the corpus to create a vocabulary. To evaluate its performance on the test set, we used the standard (per-word) perplexity measure. The perplexity (PPL) on a sequence of words $w_1,\ldots,w_M$ is given by:

$$PPL = \sqrt[M]{\prod_{i=1}^{M} \frac{1}{P(w_i \mid w_1,\ldots,w_{i-1})}} \tag{8}$$

We estimated the probability $P(w_i \mid w_1,\ldots,w_{i-1})$ in PPL as $P(w_i \mid w_1,\ldots,w_{i-N})$ that is given by SPNs.

Table 1 shows the results of our experiments. The scores of comparison systems are obtained from [13], The "Individual PPL" column shows the perplexity score of the respective systems. The "+KN5" column shows the perplexity score after taking a weighted average of a system's predictions and KN5's predictions (both equally weighted).

Table 1 Perplexity scores (PPL) of different language models

Model	Individual PPL	+KN5
KN5[14]	141.2	
Log-bilinear model[1]	144.5	115.2
Feedforward neural network[2]	140.2	116.7
RNN[3]	124.7	105.7
LDA-augmented RNN[4]	113.7	98.3
WeiSPNs[6]	107.6	82.4
OnlineSPNs	102.5	81.1

OlineSPNs outperform LDA-augmented RNN and WeiSPN by 11.2% and 5.1% respectively on "Individual PPL", and by 17.2% and 1.3% respectively on "+KN5". And WeiSPNs spent 40 hours training, our OnlineSPNs only spent only 2 hours traing.

Summary

We presented the OnlineSPNs that is used for language modeling. Our empirical comparisons with five previous language models on the standard Penn Treebank corpus demonstrate the effectiveness of our OnlineSPNs. As future work, we want to create a "recurrent" OnlineSPNs to capture long range dependencies in word sequences and apply OnlineSPNs to text multi-label classification task.

References

[1] Mnih A,Hinton G.Three new graphical models for statistical language modelling[C]// Proceedings of the 24th international conference on Machine learning. ACM, 2007: 641-648.
[2] Bengio Y, Ducharme R, Vincent P, et al. A neural probabilistic language model[J]. Journal of machine learning research, 2003, 3(Feb): 1137-1155.
[3] Mikolov T, Karafiát M, Burget L, et al. Recurrent neural network based language model[C]//Interspeech. 2010, 2: 3.
[4] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation[J]. Journal of machine Learning research, 2003, 3(Jan): 993-1022.
[5] Poon H, Domingos P. Sum-product networks: A new deep architecture[C]//Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on. IEEE, 2011: 689-690.
[6] Cheng W C, Kok S, Pham H V, et al. Language modeling with sum-product networks[C]//Fifteenth Annual Conference of the International Speech Communication Association. 2014.
[7] Gens R, Domingos P. Discriminative learning of sum-product networks[C]//Advances in Neural Information Processing Systems. 2012: 3239-3247.
[8] Broderick T, Boyd N, Wibisono A, et al. Streaming variational bayes[C]//Advances in Neural Information Processing Systems. 2013: 1727-1735.
[9] Gens R, Pedro D. Learning the structure of sum-product networks[C]//International Conference on Machine Learning. 2013: 873-880.
[10] Dennis A, Ventura D. Learning the architecture of sum-product networks using clustering on variables[C]//Advances in Neural Information Processing Systems. 2012: 2033-2041.
[11] Dennis A W, Ventura D. Greedy Structure Search for Sum-Product Networks[C]//IJCAI. 2015, 15: 932-938.
[12] Marcus M P, Marcinkiewicz M A, Santorini B. Building a large annotated corpus of English: The Penn Treebank[J]. Computational linguistics, 1993, 19(2): 313-330.
[13] Mikolov T, Zweig G. Context dependent recurrent neural network language model[J]. SLT, 2012,12:234-239.
[14] Kneser R, Ney H. Improved backing-off for m-gram language modeling[C]//Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on. IEEE, 1995, 1: 181-184