On the effectiveness and simplicity of linear recursive neural network

Peng Xu
ICME, Stanford University
Stanford, CA 94305
pengxu@stanford.edu

Ruoxi Wang
ICME, Stanford University
Stanford, CA 94305
ruoxi@stanford.edu

Abstract

We address the simplicity and effectiveness of *linear* recursive neural network (RNN) for semantic compositionality. This approach, being very simple, achieves performance that is higher than commonly used models. In particular, it beats the RNN with nonlinear activation function in predicting compositional semantic effects. Further, we show that this model has very close performance to the best result achieved by recursive neural tensor network (RNTN), which takes a 3-5 hours to train. This is in contrast with the linear RNN which only takes a couple of minutes.

1 Introduction

We consider the problem of predicting compositional semantic effects. Sophisticated models rely on rich labeled dataset and powerful compositional functions to improve performance. There has been a large body of research using recursive neural networks related methods for sentiment analysis. The simplest method in the family is the standard recursive neural network. Improved versions have been proposed to achieve higher accuracy: Matrix-vector RNN (MV-RNN) [1] represents every word and phrase in the parse tree as a vector as well as a matrix. Recursive Neural Tensor Network (RNTN) [2] uses tensor-based composition function for all nodes. These methods have achieved higher accuracy than the standard RNN model, and other commonly used models. Though promising, these methods suffer from very expensive computational cost. In this paper, we will revisit the RNN method for sentiment analysis, and address that by using *linear* activation function, combined with the RNN model, one can already be very affective in capturing semantic compositionally, while being very simple.

When using the RNN, it is common to use nonlinear activation functions such as ReLU and tanh. Nonlinear activation functions are expected to introduce more complexity into the model, and have faster convergence rate than linear function. However, different from simple fully connected neural network–where using linear function for all layers is "equivalent" to a standard linear system–the RNN model itself exists more complexity. Further, nonlinearity always exist in the form of saturation, roundoff errors, and inconsistency of the linear slope. These have motivated us to investigate in the effect of using linear activation functions in the RNN setting.

In the remainder of the paper, we describe our approach in section 2, and show our experiment results in...

2 Approach

In this section, we describe our setting to address the effectiveness of linear RNN. We take advantage of a large labeled dataset: The Stanford Sentiment Treebank. We first introduce a two-layer RNN model.

$$
\begin{aligned}
h^{(1)} &= f(W^{(1)} \begin{pmatrix} h^{(1)}_{Left} \\ h^{(1)}_{Right} \end{pmatrix} + b^{(1)}) \\
h^{(2)} &= f(W^{(2)} h^{(1)} + b^{(2)}) \\
\hat{y} &= \text{softmax}(U h^{(1)} + b^{(s)})
\end{aligned}
$$

where $h^{(1)}_{Left}$ and $h^{(1)}_{Right}$ are output (or word vector) of the layer beneath it on the left and right, respectively. $W^{(1)} \in \mathbb{R}^{d \times 2d}, W^{(2)} \in \mathbb{R}^{d_{middle} \times d}$ are the weight matrices; f is the activation funciton; $\hat{y}$ is the predicted label at this layer. An example of a 2-layer RNN is shown in Figure 1.

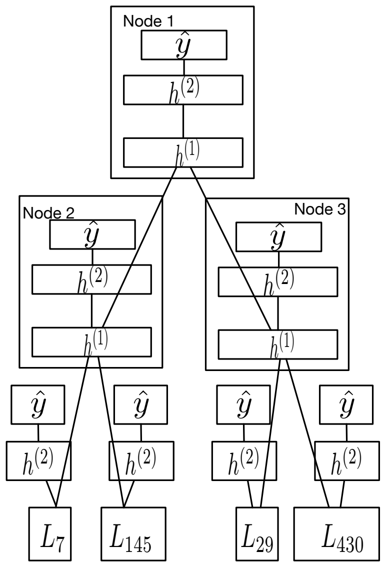

Figure 1: Example of recursive neural network (RNN)

Next, we consider five models, with all the combinations of nonlinearity and number of layers, to show the effect of using linear activation function. All models are based on the example in Figure 1.

- *model 1 (denoted as $h_1 L h_1$)*, two layer RNN with flow from $h^{(1)}$ to $h^{(1)}$. The activation function between hidden layers $h^{(1)}$ and $h^{(1)}$ is the identity function; while the activation function between hidden layer $h^{(1)}$ and $h^{(2)}$ are nonlinear function (ReLU). (see subplot (a) in Figure 2)
- *model 2 (denoted as $h_1 n L h_1$)*, two layer RNN with flow from $h^{(1)}$ to $h^{(1)}$. The activation function between hidden layers $h^{(1)}$ and $h^{(1)}$ is nonlinear; while the activation function between hidden layer $h^{(1)}$ and $h^{(2)}$ is the identity function. (see subplot (b) in Figure 2)
- *model 3 (denoted as $h_2 n L h_1$)*, two layer RNN with flow from $h^{(2)}$ to $h^{(1)}$. The activation function between hidden layers $h^{(2)}$ and $h^{(1)}$ is nonlinear; while the activation function between hidden layer $h^{(1)}$ and $h^{(2)}$ is the identity function. (see subplot (c) in Figure 2)
- *model 4 (denoted as $h_2 L h_1$)*, two layer RNN with flow from $h^{(2)}$ to $h^{(1)}$. The activation function between hidden layers $h^{(2)}$ and $h^{(1)}$ is the identity function; while the activation function between hidden layer $h^{(1)}$ and $h^{(2)}$ is nonlinear. (see subplot (d) in Figure 2)
- *model 5 (denoted as $h_1 L$)*, single layer RNN, the activation function between hidden layers is linear. (see subplot (e) in Figure 2)

$$\begin{tikzcd}
\boxed{L} \arrow[r, "L"] \arrow[d, "nL"] & \boxed{h^{(1)}} \arrow[r, "L"] \arrow[d, "nL"] & \boxed{h^{(1)}} \arrow[r, "L"] \arrow[d, "nL"] & {} \\
\boxed{h^{(2)}} \arrow[d] & \boxed{h^{(2)}} \arrow[d] & \boxed{h^{(2)}} \arrow[d] & \\
\boxed{\hat{y}} & \boxed{\hat{y}} & \boxed{\hat{y}} &
\end{tikzcd}$$

(a) model 1 ($h_1 L h_1$), two layers, linear between $h^{(1)}$ and $h^{(1)}$, nonliner between $h^{(1)}$ and $h^{(2)}$

$$\begin{tikzcd}
\boxed{L} \arrow[r, "nL"] \arrow[d, "L"] & \boxed{h^{(1)}} \arrow[r, "nL"] \arrow[d, "L"] & \boxed{h^{(1)}} \arrow[r, "nL"] \arrow[d, "L"] & {} \\
\boxed{h^{(2)}} \arrow[d] & \boxed{h^{(2)}} \arrow[d] & \boxed{h^{(2)}} \arrow[d] & \\
\boxed{\hat{y}} & \boxed{\hat{y}} & \boxed{\hat{y}} &
\end{tikzcd}$$

(b) model 2 ($h_1 nL h_1$), two layers, non-linear between $h^{(1)}$ and $h^{(1)}$, linear between $h^{(1)}$ and $h^{(2)}$

$$\begin{tikzcd}
\boxed{L} \arrow[r, "L"] & \boxed{h^{(2)}} \arrow[r, "nL"] \arrow[d] & \boxed{h^{(1)}} \arrow[r, "L"] & \boxed{h^{(2)}} \arrow[r, "nL"] \arrow[d] & {} \\
& \boxed{\hat{y}} & & \boxed{\hat{y}} &
\end{tikzcd}$$

(c) model 3 ($h_2 nL h_1$), two layers, linear between $h^{(1)}$ and $h^{(2)}$, nonlinear between $h^{(2)}$ and $h^{(1)}$

$$\begin{tikzcd}
\boxed{L} \arrow[r, "nL"] & \boxed{h^{(2)}} \arrow[r, "L"] \arrow[d] & \boxed{h^{(1)}} \arrow[r, "nL"] & \boxed{h^{(2)}} \arrow[r, "L"] \arrow[d] & {} \\
& \boxed{\hat{y}} & & \boxed{\hat{y}} &
\end{tikzcd}$$

(d) model 4 ($h_2 L h_1$), two layers, non-linear between $h^{(1)}$ and $h^{(2)}$, linear between $h^{(2)}$ and $h^{(1)}$

$$\begin{tikzcd}
\boxed{L} \arrow[r, "L"] & \boxed{h^{(1)}} \arrow[r, "L"] \arrow[d] & \boxed{h^{(1)}} \arrow[r, "L"] \arrow[d] & {} \\
& \boxed{\hat{y}} & \boxed{\hat{y}} &
\end{tikzcd}$$

(e) model 5 ($h_1 L$), single layer, linear between $h^{(1)}$ and $h^{(1)}$

Figure 2: Illustration of five models. The flow charts are based on the example shown in Figure 1. L means linear function, and nL means non-linear function. For linear, we used identity function; for nonlinear, we used ReLU.

3 Experiments

We implement all the RNN models described in the previous section on the Stanford Sentiment Treebank dataset. And we evaluated accuracy of the fine-grained sentiment classification.

For all the models, we used dev set and cross-validate over the length of the layers and the step size. The batch size is prefixed as 30 in all the experiments. From Table 1, we see that all the models we have tested achieve higher performance than those commonly used models (except RNTN). It is worth noting that the single layer linear RNN model (h_1L), which is the most simple model, achieves accuracy over 80% when the word vector length is only 10. This is very close to the best performance achieved by RNTN, which used 3-5 hours to train; and this is in contrast with the linear RNN model which only takes around 5 minutes to achieve 80.38% accuracy.

Table 1: Accuracy for fine grained (5-class) using different models. For the root accuracy, we only tested on model 5 (h_1L)

Model	VecAvg	RNN	MV-RNN	RNTN	h_1Lh_1	h_1nLh_1	h_2nLh_1	h_2Lh_1	h_1L
All	73.3	79.0	78.7	80.7	79.48	79.72	80.38	79.87	**80.38**
Root	32.7	43.2	44.4	45.7					43.3

References

[1] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211. Association for Computational Linguistics, 2012.

[2] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

Appendix

d	batch size	step size	running time per epoch	epochs	training accuracies	dev accuracies	test accuracies
5	30	2e-2	21.2	30	79.87	76.44	76.33
10	30	2e-2	22.3	20	87.00	80.11	79.98
15	30	2e-2	23.8	13	87.02	80.28	80.23
20	30	2e-2	24.1	9	85.39	80.38	80.01
25	30	2e-2	23.6	10	86.41	80.34	80.18
30	30	2e-2	25.7	10	86.24	80.51	80.40
35	30	2e-2	25.7	10	87.35	80.15	80.13
40	30	2e-2	28.7	10	87.01	80.46	80.38
45	30	2e-2	26.3	6	86.75	80.46	80.15

Table 2: single linear layer recursive neuron network

d	h	batch size	step size	running time per epoch	epochs	training accuracies	dev accuracies	test accuracies
5		30	5e-2	21.2		79.87	76.44	76.33
10		30	5e-2	22.3		87.00	80.11	79.98
15		30	5e-2	23.8		85.85	80.28	80.23
20								
25								
30	15	30	5e-2	34.3	9	86.24	80.01	80.15
30	10	30	5e-2	30.3	9	85.27	80.43	80.43
30	5	30	5e-2	32.9	15	86.34	80.19	79.82
35								
40								
45		30	2e-2	26.3	6	86.75	80.46	80.15
5	15	30	5e-2		6	77.43	77.76	80.01
10	15	30	5e-2		5	81.20	78.570	78.01
15	15	30	5e-2		9	83.52	76.71	75.99
20	15	30	5e-2		8	84.40	77.72	77.38
25	15	30	5e-2		9	84.67	80.30	80.34
30	15	30	5e-2		11	86.27	80.62	80.27
35	15	30	5e-2		10	86.42	80.59	80.44
40	15	30	5e-2		5	85.00	80.65	80.38
45	15	30	5e-2		9	85.92	80.60	80.35
change L with NL (L3)								
30	5	30	3e-2		30	83.79	78.11	78.27
30	10	30	3e-2		30	84.89	79.74	79.43
30	15	30	3e-2		29	85.96	79.40	79.27
30	20	30	3e-2		29	86.47	79.09	78.86
30	25	30	3e-2		21	86.61	79.54	79.39
30	30	30	3e-2					
30	35	30	3e-2		19	86.30	80.10	79.87
30	40	30	3e-2		21	86.24	78.70	78.26
30	45	30	3e-2					

Table 3: single linear layer recursive neuron network

d	h	batch size	step size	running time per epoch	epochs	training accuracies	dev accuracies	test accuracies
2L								
5	30	30	3e-2		28	85.41	77.68	77.51
10	30	30	3e-2		20	88.68	78.52	78.34
15	30	30	3e-2		18	89.34	79.23	78.89
20	30	30	3e-2		7	84.75	77.93	77.68
25	30	30	3e-2		9	90.29	79.80	79.61
30	30	30	3e-2		10	89.61	79.71	79.39
35	30	30	3e-2		8	89.62	79.69	79.65
40	30	30	3e-2					
45	30	30	3e-2		5	96.57	78.08	78.24
30	5	30	3e-2		11	84.62	76.06	75.85
30	10	30	3e-2		13	87.25	76.56	76.32
30	15	30	3e-2		10	89.38	79.27	79.04
30	20	30	3e-2		5	83.91	77.70	77.27
30	25	30	3e-2		26	92.65	78.84	78.83
30	35	30	3e-2		9	90.78	79.75	79.48
30	40	30	3e-2		6	94.37	77.94	78.17
30	45	30	3e-2		5	87.55	79.65	79.55
3L								
30	5	30	3e-2		27	80.96	78.09	77.97
30	10	30	3e-2		24	81.06	78.45	78.16
30	15	30	3e-2		27	82.67	79.52	79.30
30	20	30	3e-2		25	83.40	80.08	79.72
30	25	30	3e-2		22	81.21	79.14	78.85
30	30	30	3e-2					
30	35	30	3e-2					
30	40	30	3e-2					
30	45	30	3e-2					

Table 4: single linear layer recursive neuron network