Deep Learning For Mathematical Functions

Kesinee Ninsuwan
Institute of Computational and Mathematical Engineering
Stanford University
Stanford, CA 94305
eveve@stanford.edu

Abstract

In this project, we are interested in applying deep learning technique to predict the output of a mathematical function given its inputs. In particular, we apply Recursive Neural Networks (RNN) to learn vector representation of inputs and predict function output. In this project, we restrict ourselves to integer-valued functions. We show the results for 1-layer RNN and 2-layer RNN with one linear function and one nonlinear function. The results show that 1-layer RNN perform better than 2-layer RNN in both cases. Moreover, 1-layer RNN gives higher accuracy in linear function than in nonlinear function.

1 Introduction

Deep learning has proven to be successful in Natural Language Processing. Vector representation of words could capture both semantic and syntactic behavior of the text. It is a powerful tool in many applications ranging from simple to very complex tasks. Some examples of these tasks are spell checking, keyword search, machine translation, and question answering.

Our interest in this project lies on applying deep learning technique in studying mathematical functions. As an analogy to vector representation of words, we attempt to learn vector representation of inputs of a function to predict the true function value given inputs. In the work of [6], Recursive Neural Networks (RNN) shows a very good result in discovering mathematical identities. Motivated by this work, we explore the performance of RNN in our task.

The following is the outline of this report. Section 2 describes the problem precisely as well as the evaluation. In Section 3, we present the model that will be applied to solve the problem. Section 4 discusses the experiments and results of the model. In Section 5, we discuss some restriction of the model. And Section 6 gives the conclusion.

2 Problem Statement

Given the inputs u, v of a function f, the task is to predict an accurate output $f(u, v)$. In this project, we restrict the functions of interest to be integer-valued functions. The training data consists of a set of data points in the form $(u, v, f(u, v))$, where u, v are input of function f and $f(u, v)$ is the output of function. The training set, dev set, and test set are generated by simple matlab code.

Definition 2.1 *For any input u, v, let $\hat{f}(u, v)$ be the output from the model and $f(u, v)$ be the true value of the function. The model gives a correct output for u, v if and only if*

$$\hat{f}(u, v) = f(u, v)$$

3 Technical Approach and Models

In this project, we restrict the functions of interest to be integer-valued functions that take 2 input integers such that the input u, v and output $f(u, v)$ are in $[0, 999]$. For each function, we apply Recursive Neural Networks (RNN) with ReLU activation layer(s) and one softmax layer. The model gives three predicted output $\hat{\mathbf{y}}^{(1)}, \hat{\mathbf{y}}^{(2)}, \hat{\mathbf{y}}^{(3)}$. Each $\hat{\mathbf{y}}^{(i)} \in \mathcal{R}^{10}$ gives the probability that digit i of function value is the numbers $0, 1, \ldots, 9$. Let $z^{(i)} = \text{argmax}\ \hat{\mathbf{y}}^{(i)}$. Then the predicted output is $\hat{f}(u, v) = 100 \cdot z^{(1)} + 10 \cdot z^{(2)} + z^{(3)}$. Here the cost function is Cross Entropy loss:

$$CE\left(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \mathbf{y}^{(3)}, \hat{\mathbf{y}}^{(1)}, \hat{\mathbf{y}}^{(2)}, \hat{\mathbf{y}}^{(3)}\right) = -\sum_{j=1}^{3}\sum_{i=1}^{10} y_i^{(j)} \log\left(\hat{y}_i^{(j)}\right),$$

where $\mathbf{y}^{(j)} \in \mathcal{R}^{10}$ is the one-hot label vector.

We use two RNN models in this project, one using one ReLU activation layer and another using two ReLU activation layers. For one ReLU layer, we have

$$h^{(1)} = \max\left(W^{(1)}\begin{bmatrix} L_{Left} \\ L_{Right} \end{bmatrix} + b^{(1)}, 0\right)$$
$$\hat{y}^{(j)} = \text{softmax}\left(U^{(j)} h^{(1)} + b^{(s,j)}\right), \quad \text{for } j = 1, 2, 3$$

where $L_i \in R^d, \forall i = 0, 1, \ldots, 999$. Here $W^{(1)} \in \mathcal{R}^{d \times 2d}, b^{(1)} \in \mathcal{R}^d, U^{(j)} \in \mathcal{R}^{10 \times d}, b^{s,j} \in \mathcal{R}^{10}$. Figure 1 shows the diagram of RNN used in this model.

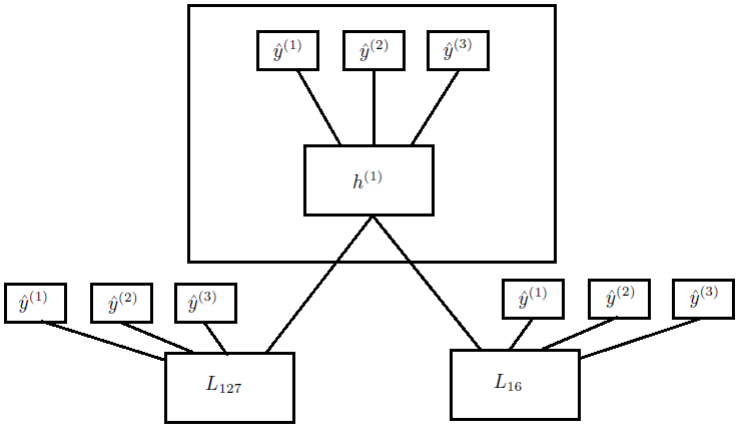

Figure 1: RNN with one ReLu layer and one softmax layer.

For 2 ReLU layers, we use the same loss function. The networks can be written as

$$h^{(1)} = \max\left(W^{(1)}\begin{bmatrix} L_{Left} \\ L_{Right} \end{bmatrix} + b^{(1)}, 0\right)$$
$$h^{(2} = \max\left(W^{(2)} h^{(1)} + b^{(2)}, 0\right)$$
$$\hat{y}^{(j)} = \text{softmax}\left(U^{(j)} h^{(2)} + b^{(s,j)}\right), \quad \text{for } j = 1, 2, 3$$

where $W^{(2)} \in \mathcal{R}^{middle \times d}, b^{(2)} \in \mathcal{R}^{middle}$, and $U^{(j)} \in \mathcal{R}^{10 \times middle}$.

Note that one advantage of this model is that we do not assume any linearity of the functions.

4 Experiments and Results

We show the result for the model with one linear function $f(u, v) = u+v$ and one nonlinear function $f(u, v) = u^2 + v$. The training set contains $60,000$ data points. The dev set and test set contain $20,000$ data points each. We train the model by picking the data points at random from training set with *epoch* schedules. To train each model, we first train over different epochs, and choose epoch that gives highest accuracy for dev set. For 1-layer RNN, we then train with different word vector dimension d (wvecdim) and test with dev set. We choose wvecdim that gives highest accuracy. For 2-layer RNN, we train to pick dimension of middle activation layer that gives highest accuracy. After we find optimal values for parameters for each model, we test it with test set to evaluate the performance.

First we show the result for linear function $f(u, v) = u+v$. Figure 2 (left) shows the plot of accuracy vs epoch. It shows that the accuracy of dev set increases with epoch. Due to the limit in computation power, we take 100 epochs as optimal value and use it with different values of dimension d of word vector for inputs. The plot of accuracy of dev set vs word vector dimension (wvecdim) in Figure 2 (right) shows that the accuracy increases with wvecdim. However, it tends to stay about the same after wvecdim = 45. Therefore, for 1-layer RNN with $f(u, v) = u + v$, we train the model with epoch = 100 and wvecdim = 45.

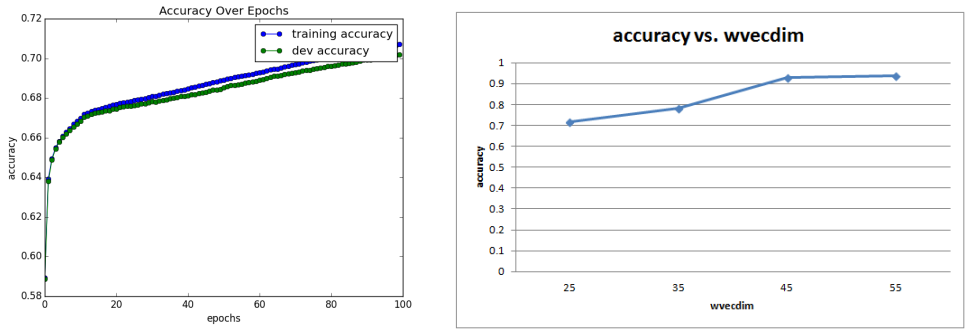

Figure 2: For linear function $f(u, v) = u + v$: (Left) the plot of accuracy vs epoch for 1-layer RNN model, (Right) the plot of accuracy of dev set vs word vector dimension d.

The result for 2-layer RNN applied to linear function $f(u, v) = u + v$ is shown in Figure 3. Here we use wvecdim = 30. Similar to 1-layer RNN, accuracy of dev set increases with epochs. So we take optimal epoch to be 100. In the plot of accuracy vs middledim in Figure 3 (right), the accuracy is highest at middledim = 45.

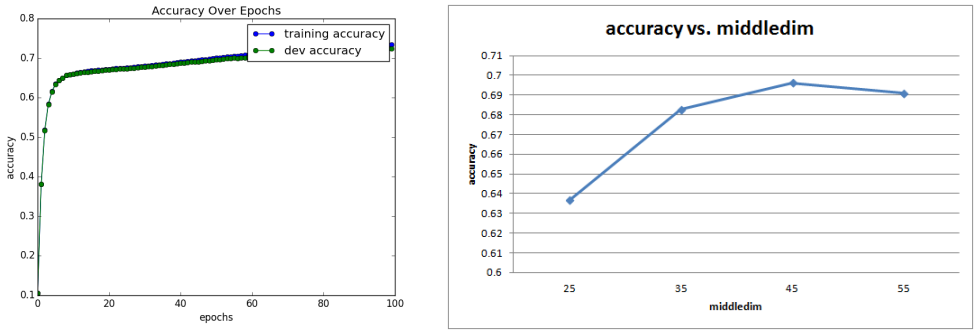

Figure 3: For linear function $f(u, v) = u + v$: (Left) the plot of accuracy vs epoch for 2-layer RNN model, (Right) the plot of accuracy of dev set vs middle dimension of activation layer

Next we show the result for non linear function $f(u, v) = u^2 + v$. The plots of result for 1-layer RNN is shown in Figure 4. And the plots result for 2-layer RNN is shown in Figure 5. Similar to the result for linear function, accuracy of dev set increases with epoch for both models. For 1-layer RNN, wavecdim = 45 gives highest accuracy. For 2-layer RNN, optimal middledim = 35.

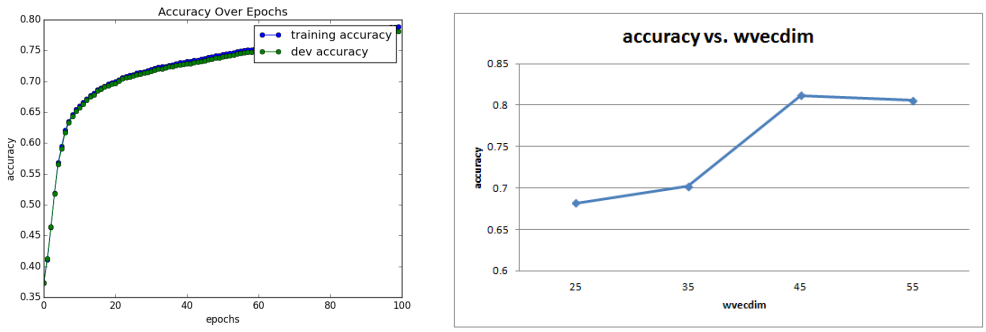

Figure 4: For nonlinear function $f(u, v) = u^2 + v$: (Left) the plot of accuracy vs epoch for 1-layer RNN model, (Right) the plot of accuracy of dev set vs word vector dimension

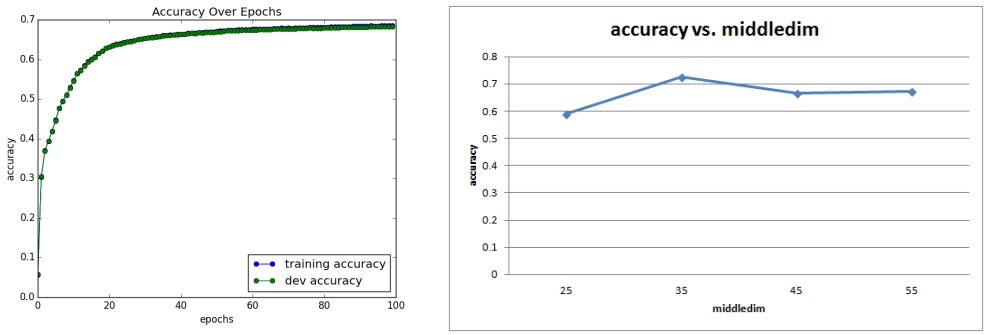

Figure 5: For nonlinear function $f(u, v) = u^2 + v$: (Left) the plot of accuracy vs epoch for 2-layer RNN model, (Right) the plot of accuracy of dev set vs middle dimension of activation layer

Table 1 shows the accuracy for each model with test set. For both functions, 1-layer RNN performs better than 2-layer RNN. For 1-layer RNN, the model gives higher accuracy for linear function.

Table 1: The accuracy of each model for the test set

	1-layer RNN	**2-layer RNN**
$f(u, v) = u + v$	0.93	0.70
$f(u, v) = u^2 + v$	0.81	0.72

5 Challenges and Restrictions

In Section 3, we describe the model for functions that take two integers as input. We also applied it to only functions that take two inputs in this project. Extending this model to use with functions that take more than two inputs is trivial. It is precisely the framework of Recursive Neural Networks [3]. In particular, the network will have more than one layer of nodes.

The model presented in this project still has a lot of restrictions. First restriction is that we require the integer-valued functions with inputs and output in $[0, 999]$. One extension for this work is to modify

the model to use with real-valued functions. For the model presented in this paper, extending the range of output increases dimension of parameters.

With limited computational power, we could not train the model with higher epochs. The plots of accuracy vs epoch in Section 4 shows that accuracy of dev set increases with epoch. Therefore, one way to improve the accuracy is to train with more values of epoch to find optimal value.

Although the model we presented in this paper still has string restriction, the results will be the first step toward applying deep learning to study more complex functions with larger domain and larger images.

6 Conclusion

In this paper, we give a model based on Recursive Neural Networks to solve the problem of predicting the value of a function give its inputs. One advantage of this model is that we do not assume the linearity for other properties of the functions. The result show a better performance of 1-layer RNN over 2-layer RNN when applying to both linear and nonlinear functions.

Some future study includes modifying the model using more activation layers, adding regularization techniques, and changing nonlinear functions in activation layers. In establish stronger conclusion, we must test the model with more classes of functions. It is also interesting to study the behavior of the model when we train with training set with different distribution.

Acknowledgments

The author would like to acknowledge Mike Phulsuksombati for the discussion about the project idea and all the data used in this project. He also gives an insightful advice throughout the process. In addition, the author would like to thank the instructor Dr. Richard Socher and all the staff of class CS224D at Stanford University for all wonderful advices and guidance.

References

[1] Benign, Y. (2012). Practical recommendations for gradient-based training of deep architectures. arXiv:1206.5533.

[2] Socher, R., Bauer, J., Manning, C.D., Ng, A.Y. Parsing with Compositional Vector Grammars.

[3] Socher, R., Lin, C.C., Ng, A.Y., and Manning, C.D. Parsing Natural Scenes and Natural Language with Recursive Neural Networks.

[4] Socher, R., et al. (2013) Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Vol. 1631. 2013.

[5] Pennington, J., Socher, R., Manning, C.D. GloVe: Global Vectors for Word Representation.

[6] Zaremba, W., Kurach, K., and Fergus, R. (2014). Learning to Discover Efficient Mathematical Identities. *arXiv:1406.1584v3*.

[7] Zaremba, W. and Sutskever, I. (2015). Learning to Execute. arXiv:1410.4615v3.