It is well known that it is extremely hard for standard feedforward neural nets to learn the parity function. However, using recurrent neural nets, observing one bit at a time, learning the parity function is a much easier task. We empirically validate this, visualise the learned memory state of the network and check how the performance is affected by various factors, such as the network architecture , types of hidden units.

Learning the Parity Function using Recurrent Neural Networks

Yutaro Yamada† and Uri Shaham‡,
Technical Report YALEU/DCS/TR-????

† Department of Computer Science, Yale University, New Haven CT 06511
‡ Department of Statistics, Yale University, New Haven CT 06511

Keywords: *the parity function, recurrent neural nets*

1 Introduction

The parity function is defined on the set $\{0, 1\}^*$ of finite binary strings and returns 1 of the sum of the bits is even and 0 otherwise. The ability of mahcine learning algorithms to learn the parity function is a long standing problem. Generally, the function is considered to be hard to learn as it very oscilatory wrt the Hamming distance - every single bit flip changes the value of the function.

While several works show that the n-bit parity function can be *represented* by fully connected neural nets (also called Multi Layer Perceptron, or MLP) with a single hidden layer with $O(n)$ units (see, for example, [6, 4, 1]), actually *learning* the parity function from a set of labeled examples by standard training methods is extremely difficult, and may result on a very long training time or even faliure to generalize to unseen examples.

Despite the great diffuculty of fully connected neural nets to learn the parity function, it has been observed [2, 5, 3] that Recurrent Neural Nets (RNNs) can easily learn the parity function. In this context, the main difference between RNNs and MLPs is that while MLPs are given the full string as an input, RNNs observe one but at a time, and are allowed to update the states of their hidden units based on the observed bit (and the current state). Effectively, this enables RNNs to work as a 2-state automaton, depicted in Figure **??**[[TODO: draw figure]], where the state at time t corresponds to the the parity of the substring of the first t bits of the string. This reduces learning the parity on arbitrarily long strings to learning it on 2-bit strings, which is an easy learning task.

In this manuscript we empirically validate this, and perform a series of experiments to investigate how the quality of learning depends on various factors: the network architecture (numbers and sizes of layers), the hidden units activation function, the number of training sequences, training protocol (number of time steps, single / sequential target). We also compare between RNNs and other learning algorithms.

The structure of this manuscript is as follows: in Section 2 we give a brief overview on RNNs. Experimental results are given in Section 3. Section 4 concludes the manuscript.

2 Recurrent Neural Nets

A quick description of RNNs, you can use chapter 10 in Bengio's book.

- main difference from MLP: MLPs are acyclic graphs, RNNs have cycles- hidden layers are have self connections.
- figures: rnn with cycels, expansion in time
- backprop through time (i.e., how to compute gradients for RNNs)
- LSTM - main idea, diagram.

Altogether, this section should take a single page at most.

3 Experiments

3.1 Setup

decribe the experimental setup

3.2 Visualization of Hidden Units

- how do the activations of the hidden units change with time, how does of correspond to the parity

3.3 Dependence on the Architecture

- number of layers
- number of units

3.4 Dependence on the Activations

- LSTM vs. sigmoid / tanh units
- which is faster / more accurate?

3.5 Dependence on the Sample Complexity

- performance vs. number of training sequences

3.6 Effect of training Protocol

- sequential targets vs. single target
- effect of number of time steps

3.7 Long Range Prediction

- quality of prediction vs. length of testing string

3.8 Comparison to other methods

- fully connected net with various numbers and sizes of hidden layers
- HMM (designed for sequential data)
- SVM

4 Discussion

References

[1] Leonardo Franco and Sergio A Cannas. Generalization properties of modular networks: Implementing the parity function. *Neural Networks, IEEE Transactions on*, 12(6):1306–1313, 2001.

[2] Sepp Hochreiter and Jurgen Schmidhuber. Bridging long time lags by weight guessing and long short-term memory. *Spatiotemporal models in biological and artificial systems*, 37:65–72, 1996.

[3] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *arXiv preprint arXiv:1507.01526*, 2015.

[4] Rudy Setiono. On the solution of the parity problem by a single hidden layer feedforward neural network. *Neurocomputing*, 16(3):225–235, 1997.

[5] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2368–2376, 2015.

[6] Bodgan M Wilamowski, David Hunter, and Aleksander Malinowski. Solving parity-n problems with feedforward neural networks. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 4, pages 2546–2551. IEEE, 2003.