CSCE 686 - Project Proposal - Recursive Neural Network Training Using the Hardest Walk Problem and Particle Swarm Optimization

Justin Fletcher

May 13, 2016

A recurrent neural network (RNN) is a neural network for which the underlying graph is weighted, directed, and cyclic. As with all neural networks, the purpose of an RNN is to map an input pattern to a desired output pattern. Assuming the topology and transfer function of RNN are fixed, the accuracy of this mapping is dependent only on the synaptic weights of the network, which coorespond to the edge weights in the underlying graph. The problem of selecting edge weights, such that the difference between computed and desired output patterns is minimized, is known as the *loading problem* []. The process of searching for a synaptic weight configuration which solves the loading problem is known as *training* the neural network.

We generally say that an feasible solution is one with error less than ϵ, where ϵ is an arbitrarily-small error value. The weight selection problem is often called the *loading problem*, as the choice of weights loads a representation of the mapping between input and output vectors into the RNN. Weight selection for an RNN is known to be PSPACE-complete [1]. As such, this problem is admissible as a CSCE-686 course project. In this proposal, a global depth-first search (DFS) and a stochastic search solution to the RNN weight selection problem are proposed.

Several methods exist for RNN training. In this work, a new method is proposed. The inspiration for this method is the back-propagation (BP) algorithm for feed-forward neural network (FFNN) weight selection. In BP, the error produced by the network for a given set of input-output pattern pairs propagated backward through the network, and a share of the credit for the produced error is allocated to each of the weights according to their "responsibility" for the error. This is called the *credit assignment problem.* It is not possible to use this procedure to assign blame for errors in a recurrent neural network because patterns propagate through the network every direction in the underlying graph, thereby intermingling the effects of many weights in the final output error. A potentially novel method for assigning credit for RNN output error is proposed in this work.

The output error of an RNN is a linear combination of error terms, each corresponding to an element of the output vector. Thus, to minimize the total error we must minimize the error of each output vector element. Observe that the output signal from a given neuron in an RNN is the sum of weighted signals into that neuron, transformed by the activation function. If we take the activation function to be fixed, and the input signals to be incidental to the operation of the network, then we find that the weight is the only feature of the network to which we may assign blame for the produced error. Thus, if a neuron outputs a signal which results in an error with a particular direction (positive or negative), then that error must be induced by those weights which are of the same direction. Further, the larger the error-inducing weight, the more it contributes to the error. Thus, by modifying the error-inducing weight, the error may be reduced. However, this procedure only applies to those weights for which the afferent neuron has an explicitly-defined desired output value. In order to apply this procedure to all weight in the network, a mechanism for defining the error contributions of other weights must be established. Applying inductive reasoning backward through the network, we observe that the maximum credit for the error will always be assigned to the largest weight with the same sign as the error, at each neuron. However, because the underlying graph contains cycles, there is no obvious way to conclude this induction.

A key insight from the RNN problem domain is required to proceed. First, we define a propagation to be the update of the output signals for all neurons in the network. A finite-time RNN produces an output vector after a fixed number of propagations, a quantity which we label L. Thus, the maximum length of a walk [2], taken by an input signal, from the input neuron to the output neuron is necessarily L. This constraint is sufficient to bound the induction over the structure of the graph; we know that the credit assignment path for a given output neuron must end at that neuron, begin at an input neuron, and contain L edge traversals. Thus, applying the inductive reasoning above, the weights of the edges which lie along maximum-weight walks from each input neuron to a particular output neuron constitute the greatest contributions to the error produced by that neuron. As such, these weights should be reduced proportionally to the error produced at that neuron. Repeating this procedure will reduce the output error of the network. The exact details of this procedure shall be established in the proposed project. This procedure may serve as a next-state-generation and selection function in the global DFS search algorithm, where the next state generated is the modified network, and that network is selected. This reduces the search tree to a path, and thus could equivalently be formulated as a gradient descent problem.

We now formalize the maximum-weight-walk-finding procedure. Given a graph $G = (V, E)$ find, for each input vertex $u \in V(G)$ and each output vertex $v \in V(G)$, a u, v-walk W entailing exactly L edge traversals, such that

$$\sum_{e \in W} w(e)$$

is maximized. An initial review of the available literature reveals no formulation of a problem matching this description. The most similar problem is the longest path problem, which is NP-hard. This suggests that this problem, too, is likely NP-hard. Further investigation is required to determine the complexity of the problem. The proposed project entails a complete and rigorous description of this new problem, which will henceforth be called the hardest walk problem (HWP).

The HWP is solved via depth-first search with backtracking. The root of the search tree is the initial endpoint of the walk, u. u may be adjacent to, at most, every other neuron in the network, thus there are $n = |V(G)|$ possible edge traversals. At the next level of the search tree, each search tree node again begets $n - 1$ nodes, thereby comprising a set of $(n-1)^2$ nodes. This tree growth repeats $L-1$ times. To construct the Lth layer, all vertices must follow an edge to vertex v. The underlying graph of the network is simple and fully connected, so the Lth layer has $(n-1)^{L-1}$ nodes, as each node must result in a walk which ends at v. Thus, the total number of nodes which must be searched is

$$(n-1)^{L-1} + \sum_{l=0}^{L-1} (n-1)^l. \tag{1}$$

Therefore, the running time complexity of computing an exact solution to the HWP using DFS/BT is $\mathcal{O}(n^L)$.

For the stochastic bio-inspired search algorithm, particle swarm optimization PSO will be employed. In the realization of PSO described in this work, each particle will be characterized by an individual RNN. The location in configuration space corresponds to the weight values of the RNN.

All experimental data on which the RNNs developed in this work are trained are drawn from standard experimental data sets. Each data set is publicly-available, and has been featured in a least one academic publication, in order to ensure a basis for comparison.

References

[1] J. Wiedermann, *Mathematical Foundations of Computer Science 1989: Porabka-Kozubnik, Poland August 28 – September 1, 1989 Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, ch. On the computational efficiency of symmetric neural networks, pp. 545–552.

[2] D. B. West, "Introduction to graph theory (second edition)," 2007.