Estimating Probabilities for Unbounded Categorization Problems [1]

James B. Henderson

Dpartement d'Informatique, Universit de Genve, Geneva, Switzerland

Abstract

We propose two output activation functions for estimating probability distributions over an unbounded number of categories with a recurrent neural network, and derive the statistical assumptions which they embody. Both these methods perform better than the standard approach to such problems, when applied to probabilistic parsing of natural language with Simple Synchrony Networks.

Key words: recurrent neural networks, probability estimation, natural language parsing

1 Introduction

Recurrent networks have the advantage over feed-forward networks that they can compute mappings from arbitrarily many input patterns to arbitrarily many output patterns. The work presented in this article focuses on the issues that arise with arbitrarily many output patters. In particular, we are concerned with the problem of estimating a probability distribution over an unbounded set of mutually exclusive categories. We propose two output activation functions which can be used for such tasks, and in each case derive the statistical assumptions which are necessary to prove that training networks which use these functions will result in estimates of the desired probability distributions. We then experimentally evaluate these activation functions in an application of a recurrent network architecture (Simple Synchrony Networks [9]) to probabilistic parsing of real natural language sentences. We find that both of these activation functions perform better than the standard approach of converting the task to a sequence of bounded categorization problems.

[1] This work was supported by the Centre Universitaire d'Informatique, Universit de Genve.

As an example of an unbounded categorization problem, consider the decision an incremental natural language parser must make when it reaches the last word of the sentence “John said Mary left yesterday.” The parser must choose whether “yesterday” modifies “said” or “left”. Each alternative modification can be thought of as a category, and they are mutually exclusive. In general there can be an unbounded number of verbs in such a sentence, so there can be an unbounded number of modifications which the parser must choose between. To make this choice, we want to be able to estimate a probability distribution over an unbounded number of mutually exclusive categories.

The reason a recurrent network can produce an unbounded number of outputs is that it can be run for an unbounded amount of time, and produce different outputs at each time.[2] We assume that one output y_i is produced at each time i, from one hidden layer activation pattern $\mathbf{z}_i$.[3] We will denote the total input as $\mathbf{x}$ and use Y_i to denote that the output category associated with time i is the correct output category. Our objective is to have y_i be an estimate of $P(Y_i|\mathbf{x})$. In the above example, the potential modifications are the different i, Y_i represents the correctness of modification i, and $\mathbf{x}$ is the whole sentence plus the question of how to parse “yesterday”. We want y_1 to be an estimate of the probability that “yesterday” modifies “said”, and y_2 to be an estimate of the probability that “yesterday” modifies “left”.

2 Using a Sequence of Bounded Categorizations

The standard method in statistical modeling and neural networks for handling unbounded categorization problems is to convert them into an unbounded sequence of bounded categorization problems. The first step in this process is to choose an ordering between the categories, which we will denote $c_1, \ldots, c_n$. The categorization probabilities which are then estimated are each conditional on the correct category not being any of those earlier in the sequence, namely $P(Y_{c_i}|\mathbf{x}, \neg Y_{c_1}, \ldots, \neg Y_{c_{i-1}})$. These probabilities can each be estimated independently, because the fact that the alternatives are mutually exclusive has been taken care of in the conditioning. The original probabilities can then be computed from these estimates by multiplying in accordance with the chain rule for conditional probabilities:[4]

$$P(Y_{c_2}|\mathbf{x}) = P(Y_{c_2}|\mathbf{x}, \neg Y_{c_1}) \times (1 - P(Y_{c_1}|\mathbf{x})) \tag{1}$$

[2] We are using the word “time” simply to refer to the different states which a recurrent network passes through. Everything discussed here is equally applicable to the graph-based generalization of time, as discussed in [6].

[3] All this work has been generalized to any finite number of output categories per time, but we do not present this extension here for simplicity.

[4] Note that $P(Y_{c_n}|\mathbf{x}, \neg Y_{c_1}, \ldots, \neg Y_{c_{n-1}})$ is actually equal to 1 because c_n is the only possible alternative remaining.

$$\begin{aligned} & \dots \\ P(Y_{c_n}|\mathbf{x}) &= P(Y_{c_n}|\mathbf{x}, \neg Y_{c_1}, \dots, \neg Y_{c_{n-1}}) \times \\ & \quad (1 - P(Y_{c_1}|\mathbf{x})) \times \dots \times (1 - P(Y_{c_{n-1}}|\mathbf{x})) \end{aligned} \tag{2}$$

The advantage of this method is that each probability which needs to be estimated $P(Y_{c_i}|\mathbf{x}, \neg Y_{c_1}, \dots, \neg Y_{c_{i-1}})$ is for just a binary categorization problem; either the category is chosen or it is not. Thus the standard output activation function for estimating probabilities with neural networks can be used. The only complication is that there is an unbounded amount of information in the conditional of each probability. The information $\neg Y_{c_1}, \dots, \neg Y_{c_{i-1}}$ can be represented with the single fact $\neg\exists j{<}i(Y_{c_j})$. The unbounded amount of information in the input $\mathbf{x}$ can be handled by running some form of recurrent network across the input. The network defines a function from $\mathbf{x}$ to a hidden representation $\mathbf{z}_{c_i}$, which is trained to be a sufficient representation of $\mathbf{x}$ for estimating the probability:

$$P(Y_{c_i}|\mathbf{x}, \neg Y_{c_1}, \dots, \neg Y_{c_{i-1}}) = P(Y_{c_i}|\mathbf{z}_{c_i}, \mathbf{x}, \neg Y_{c_1}, \dots, \neg Y_{c_{i-1}}) \tag{3}$$

$$\approx P(Y_{c_i}|\mathbf{z}_{c_i}, \neg\exists j{<}i(Y_{c_j})) \tag{4}$$

The main disadvantage of this method is that the sequential ordering chosen for the conditioning biases the estimate. When the estimates are smoothed, as with neural networks, then small probabilities will tend to be overestimated, thereby reducing the probability assigned to categories later in the sequence, due to the $(1{-}P(Y_{c_j}|\mathbf{x}))$ terms. This will bias the estimates towards categories earlier in the sequence, and this bias will be detrimental for most tasks no matter what ordering we choose. In the next section we will propose methods which do not require any ordering, and thus do not introduce this bias.

3 Estimating the Probabilities Directly

The standard methods for estimating probabilities with neural networks apply only to feed-forward networks. However, for any (discrete time) recurrent network applied to a particular input, we can convert it into an equivalent feed-forward network with one copy of the recurrent network for each time step in the input [10]. We can use this fact to trivially extend the standard proofs for showing that a trained neural network will estimate a probability distribution to the use of recurrent networks discussed in the previous section. The difficulty arises because these standard proofs assume that all the outputs for a given probability distribution are computed from the same hidden layer activation pattern [1]. But if we want to estimate a probability distribution over an unbounded set of categories i then we need to produce their outputs y_i at different times using different hidden activation patterns $\mathbf{z}_i$. The fundamental issue in deriving new activation functions for estimating probability distributions over an unbounded set of categories is how to handle an unbounded number of hidden activation patterns.

3.1 An Unbounded Logistic Sigmoid Function

When we train a network to compute an output, the hidden layer learns a representation which is optimized for computing that particular output given the input. Thus it is natural to assume that a given output y_i can be accurately computed from its hidden activation pattern $\mathbf{z}_i$ without requiring additional information about the input $\mathbf{x}$. This independence assumption, plus the fact that $\mathbf{z}_i$ is a function of $\mathbf{x}$, results in the following approximation: [5]

$$\begin{aligned} P(Y_i|\mathbf{x}) &= P(Y_i|\mathbf{z}_i, \mathbf{x}) && (5) \\ &\approx P(Y_i|\mathbf{z}_i) = \frac{p(\mathbf{z}_i|Y_i)P(Y_i)}{p(\mathbf{z}_i)} && (6) \end{aligned}$$

In this model we are not assuming any ordering or other differentiation between the different category indexes i, so the prior probability distribution over these categories $P(Y_i)$ must be uniform. Therefore $P(Y_i) = \frac{1}{n}$, where n is the number of categories. Because each $\mathbf{z}_i$ is different, we cannot use it to normalize across outputs, but we can write it as a sum over output cases:

$$p(\mathbf{z}_i) = p(\mathbf{z}_i|Y_i)P(Y_i) + \sum_{j \neq i} p(\mathbf{z}_i|Y_j)P(Y_j) = \frac{p(\mathbf{z}_i|Y_i)}{n} + \sum_{j \neq i} \frac{p(\mathbf{z}_i|Y_j)}{n} \quad (7)$$

The first element of the sum is the numerator in the previous formula. The second element represents the probability of computing the hidden activation vector $\mathbf{z}_i$ in situations where Y_i is false. We assume that these later probabilities are only dependent on the fact that Y_i is false, and not dependent on the specific index j of the correct category. Thus:

$$p(\mathbf{z}_i) \approx \frac{p(\mathbf{z}_i|Y_i)}{n} + p(\mathbf{z}_i|\neg Y_i)\left(1 - \frac{1}{n}\right) \quad (8)$$

These assumptions allow us to reduce $P(Y_i|\mathbf{x})$ to a form similar to the activation function used in the two-category case for finite mappings:

$$P(Y_i|\mathbf{x}) \approx \frac{p(\mathbf{z}_i|Y_i)/n}{p(\mathbf{z}_i|Y_i)/n + p(\mathbf{z}_i|\neg Y_i)\left(1 - \frac{1}{n}\right)} \quad (9)$$

[5] Note that this assumption is stronger than the one made in section 2 because in this case we do not have as much information in the conditional.

$$\approx \frac{1}{1+(n-1)\exp(-a_i)} \quad \text{where } a_i = \ln\left(\frac{p(\mathbf{z}_i|Y_i)}{p(\mathbf{z}_i|\neg Y_i)}\right) \tag{10}$$

Equation 10 is the logistic sigmoid activation function, except that the weighting of $(n-1)$ has the effect of shifting the function to the right by $\ln(n-1)$. This shifting has little effect on estimates close to 1 or 0, but a large effect on estimates close to 0.5.

For the two-category case, the logistic sigmoid activation function allows a single output unit to estimate the desired posterior probability, assuming that the category-conditioned probability of the hidden activation vector is in a particular form of the exponential family of distributions [1]. Translating this to our case, we want to ensure that the output y_i can estimate $P(Y_i|\mathbf{x})$, assuming that both $p(\mathbf{z}_i|Y_i)$ and $p(\mathbf{z}_i|\neg Y_i)$ are in the exponential family:

$$p(\mathbf{z}_i|Y_i) = \exp(A(\theta_1) + B(\mathbf{z}_i, \phi_i) + \theta_1^T \mathbf{z}_i) \tag{11}$$
$$p(\mathbf{z}_i|\neg Y_i) = \exp(A(\theta_2) + B(\mathbf{z}_i, \phi_i) + \theta_2^T \mathbf{z}_i) \tag{12}$$

Note that the parameter ϕ_i must be the same for these two probabilities, and that both θ_1 and θ_2 must be the same for all output categories i. Combining these assumptions with the definition of a_i in equation 10 we get:

$$a_i = \ln\left(\frac{\exp(A(\theta_1) + B(\mathbf{z}_i, \phi_i) + \theta_1^T \mathbf{z}_i)}{\exp(A(\theta_2) + B(\mathbf{z}_i, \phi_i) + \theta_2^T \mathbf{z}_i)}\right) \tag{13}$$
$$= \mathbf{w}^T \mathbf{z}_i + w_0 \quad \text{where } \mathbf{w} = \theta_1 - \theta_2, \text{ and } w_0 = A(\theta_1) - A(\theta_2) \tag{14}$$

Now using equation 10 as our output activation function and $\mathbf{w}, w_0$ as our output weights and bias gives us the desired result.

$$y_i = \frac{1}{1+(n-1)\exp(-(\mathbf{w}^T \mathbf{z}_i + w_0))} \tag{15}$$
$$\approx P(Y_i|\mathbf{x}) \tag{16}$$

Here we can see why equations 11 and 12 had to have the same parameters θ_1 and θ_2 for all categories i. The different y_i are computed by the recurrent network at different times from different hidden activation patterns z_i, but they are all computed with the same link weights $\mathbf{w}, w_0$. Therefore it must be possible to define $\mathbf{w}, w_0$ independently of i, which requires θ_1 and θ_2 to be the same for all i.

The training algorithm for this model is the same as for the finite case, namely using the cross-entropy error function and back-propagation learning. The equivalence arises from the fact that equation 10 is the usual logistic sigmoid function applied to $b_i = a_i - \ln(n-1)$, and that $\frac{\partial b_i}{\partial a_i} = 1$, so $\frac{\partial E}{\partial a_i} = \frac{\partial E}{\partial b_i} = y_i - Y_i$ (where E is the error). We can use the same proof as in the finite case to show that the minimum of the cross-entropy error occurs where the output function is the desired probability function, assuming an infinite training set [1].

Because the independence assumptions in this model are not exactly correct, it is sometimes desirable to add a normalization factor by taking advantage of the fact that $\sum_k p(Y_k|\mathbf{x}) = 1$. To normalize, equation 10 simply needs to be divided by $\sum_k \frac{1}{1+(n-1)\exp(-a_k)}$. This is optional, but it has been found to help in the final stages of training. For the network whose results are reported in section 4, performance on the validation set increased by 0.2% by training with normalization after the training with the un-normalized version was complete.

$$p(Y_i|\mathbf{x}) \approx \frac{\frac{1}{1+(n-1)\exp(-a_i)}}{\sum_k \frac{1}{1+(n-1)\exp(-a_k)}} \tag{17}$$

In training, this requires a slight change to the back-propagation of error across the output layer.

$$\frac{\partial E}{\partial a_i} = (y_i - Y_i)\left(1 - \frac{1}{1+(n-1)\exp(-a_i)}\right) \tag{18}$$

3.2 An Unbounded Softmax Function

As an alternative to the unbounded sigmoid activation function, we consider what statistical assumptions would be necessary to derive a version of the normalized exponential, or “softmax”, activation function [2]. This is the proper activation function to use in the case of a bounded number of categories greater than 2, so it is natural to expect it would generalize to the unbounded case. For this derivation we do not assume that one hidden layer activation pattern is sufficient to estimate a given category’s probability, but instead make the much weaker assumption that the total set of all hidden patterns is sufficient. This alternative provides us with a normalization factor equal to $p(\mathbf{z}_1, \ldots, \mathbf{z}_n)$:

$$P(Y_i|\mathbf{x}) = P(Y_i|\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{x}) \tag{19}$$

$$\approx P(Y_i|\mathbf{z}_1, \ldots, \mathbf{z}_n) = \frac{p(\mathbf{z}_1, \ldots, \mathbf{z}_n|Y_i)P(Y_i)}{\sum_k p(\mathbf{z}_1, \ldots, \mathbf{z}_n|Y_k)P(Y_k)} \tag{20}$$

We then make the main independence assumption of this method; we assume that the probability distributions over hidden activation patterns are conditionally independent, given the correct category i:

$$p(\mathbf{z}_1, \ldots, \mathbf{z}_n|Y_i) \approx p(\mathbf{z}_i|Y_i) \times \prod_{j \neq i} p(\mathbf{z}_j|Y_i) \tag{21}$$

$$\approx \; p(\mathbf{z}_i|Y_i) \times \prod_{j \neq i} p(\mathbf{z}_j|\neg Y_j) \qquad (22)$$

In the second step we have assumed, as for the previous function, that the only information about Y_i which is relevant to $\mathbf{z}_j$ is the fact that Y_j is false.

Using, as above, the uniform prior $P(Y_i) = \frac{1}{n}$, and substituting formula 22 into formula 20 and simplifying, we get:

$$P(Y_i|\mathbf{x}) \approx \frac{\frac{p(\mathbf{z}_i|Y_i)}{p(\mathbf{z}_i|\neg Y_i)}}{\sum_k \frac{p(\mathbf{z}_k|Y_k)}{p(\mathbf{z}_k|\neg Y_k)}} \qquad (23)$$

$$\approx \frac{\exp(a_i)}{\sum_k \exp(a_k)} \quad \text{where} \quad a_i = \ln\left(\frac{p(\mathbf{z}_i|Y_i)}{p(\mathbf{z}_i|\neg Y_i)}\right) \qquad (24)$$

Equation 24 is precisely the softmax activation function, applied to an unbounded number of categories. This activation function has also been used in other work on natural language parsing with neural networks [4], but without any theoretical or empirical justification.

When the softmax activation function is applied to a finite number of categories, the outputs can be shown to estimate the desired posterior probabilities, assuming that the category-conditioned probabilities of the hidden activation vectors are in a particular form of the exponential family of distributions [1]. We can extend this result to our case by making the same assumptions as given for the sigmoid activation function in equations 11 and 12. Combining these assumptions with the definition of a_i in equation 24 gives us the same result as for the sigmoid activation function in equation 14. Now using equation 24 as our output activation function and $\mathbf{w}, w_0$ as our output weights gives us the desired result.

$$y_i = \frac{\exp(\mathbf{w}^T\mathbf{z}_i + w_0)}{\sum_k \exp(\mathbf{w}^T\mathbf{z}_k + w_0)} \qquad (25)$$

$$\approx P(Y_i|\mathbf{x}) \qquad (26)$$

As with the sigmoid activation function, the training algorithm for this model is the same as for the finite case, namely using the cross-entropy error function and back-propagation learning. The same proofs as in the finite case apply to both showing that $\frac{\partial E}{\partial a_i} = y_i - Y_i$ and showing that the minimum of the cross-entropy error occurs where the output function is the desired probability function, assuming an infinite training set [1].

4 Experiments on Probabilistic Parsing

To empirically compare these three methods for estimating probabilities for unbounded categorization problems, we apply them to parsing natural language sentences. The input to this task is an unbounded sequence of words (the sentence), and the result is a tree of phrases with the words at its leaves. We will represent this tree as a set of phrases plus a set of parent-child relationships between these phrases and between these phrases and the words. In the previous example, the result of parsing "John said Mary left yesterday" would include two different phrases which are the parents of "said" and "left", respectively, and only one of these phrases can be parent of "yesterday".

The recurrent network architecture which we use in these experiments is Simple Synchrony Networks (SSNs) [9]. SSNs extend Simple Recurrent Networks [5] with the addition of Temporal Synchrony Variable Binding [11]. This extension allows them to model sets of sequences, rather than just individual sequences. At each position in the input sequence, a new sequence is added to the SSN's set of computed sequences, and one output pattern is computed for each of the computed sequences in the set at that position. This results in one output pattern for each pairing of a position in the input with a preceding position in the input, $(i, j), i \leq j$. This gives SSNs a number of output patterns which is quadratic in the length of the input, rather than just linear.

When applied to natural language parsing, the quadratic number of SSN outputs allows SSNs to compute one probability estimate for every possible parent of every word or phrase in the sentence. For real natural language, it is fairly simple to encode a phrase structure tree in such a way that a bounded number of phrases are associated with each position in the input sentence (in these experiments the bound is one), so that the number of possible parent relationships grows quadratically with the length of the sentence. It is thus possible to define a mapping from possible parent relationships to position-position pairs such that only a bounded number of relationships are mapped to every position pair. We then use the SSN output pattern for a given position pair to compute the bounded number of probability estimates for possible parent relationships which have been mapped to that position pair.

To define the mapping from parent relationships to position pairs, we first assign each phrase to a position in the sentence (in this work, the position of the first word of the phrase). We then assign each relationship to the pairing of the positions for the two phrases (or phrase and word) involved in the relationship. This results in each pairing of positions having four possible relationships assigned to it. For relationships between two phrases, there is the case where the parent is assigned to the earlier position, plus the case where the parent is assigned to the later position. For relationships between a phrase and a word, the phrase is always the parent and the phrase is never assigned to a position later than the word, but there is the case where the phrase is assigned to a position earlier than the word, and there is the case where the phrase is assigned to the position of the word itself.

Each one of these four cases has a different output unit which is trained to compute probability estimates for parent relationships of that type. By computing values for these four output units at all the output patterns computed by the SSN, we get probability estimates for every possible parent relationship in the sentence. In addition, the SSN's outputs estimate the probabilities for the possible phrase labels for each phrase (noun phrase, verb phrase, etc.). After the SSN computation is finished, the total set of probability estimates are used in a statistical parser to determine the most probable parse, and this parse is the output of the SSN parsing system (see [7] for more details).

Training the SSNs is done with standard back-propagation through time [10], but with SSNs we have two dimensions of time, one for the input sequence (as with Simple Recurrent Networks) and the other to cycle through the set of computed sequences (as with Temporal Synchrony Variable Binding). Thus we need to make one copy of the network's weights for each pairing of an input position with any of its preceding positions (i.e. one copy for each output pattern). With the version of SSNs used here, error flows backward through each individual sequence, but the only connection between different sequences is either pre-specified in the design of the network's input patterns or is a result of normalizing across different sequence's outputs.

In the experiments reported here, the same network architecture, training method, and statistical parser are applied for each probability estimation method. The only difference between the methods is the output activation function and its associated error function. For each method, we trained multiple networks with 60, 80, and 100 hidden units, using momentum and weight decay regularization. Training error was used to automatically reduce the learning rate, and a validation set was used to automatically reduce the regularization and to decide when to stop training. In each case the best network was chosen based on its performance on the validation set, and results were then computed on a held-out testing set. In each case one of the networks with 100 hidden units was chosen, but similar validation results were achieved by all the networks run with either 80 or 100 hidden units.

Table 1 shows the performance of the chosen parsers for the three methods. "Sequence" uses the standard method of converting the unbounded distribution into a sequence of bounded distributions, "Unb Sig" uses the unbounded sigmoid function, and "Unb Soft" uses the unbounded softmax function. The standard single measure for comparing parser performance is the F-measure applied to labeled phrases in the testing set, shown in the last column of the table. F-measure is a balanced combination ($\frac{2 \times r \times p}{r+p}$) of recall (percentage of correct phrases which are output) and precision (percentage of output phrases which are correct).

Table 1
Parsing results: percent labeled phrase recall, precision, and F-measure

	Training			Validation			Testing		
	Rec	Prec	$F_{\beta=1}$	Rec	Prec	$F_{\beta=1}$	Rec	Prec	$F_{\beta=1}$
Sequence	60.2	62.3	61.3	57.9	60.0	58.9	58.7	59.9	59.3
Unb Sig	69.6	71.8	70.7	62.9	64.3	63.6	65.1	66.6	65.8
Unb Soft	68.7	70.6	69.6	63.6	64.7	64.1	64.8	65.9	65.4

These results are all good considering the nature of the datasets. For comparison, the testing results for an un-smoothed statistical model (a Probabilistic Context Free Grammar, PCFG) were 29.2% recall, 53.7% precision, and 37.8% F-measure. The PCFG performs poorly because of the relatively small datasets used here, even though we combined the training and validation sets to train the PCFG. The SSN models all handle this problem robustly, finding appropriate ways to smooth the probabilities from the training cases to the novel testing cases (see [7] for a discussion). When a more specialized version of an SSN parser is trained using a much larger dataset (one million words, versus the 26,480 words used here), performance reaches 83.8% F-measure [8], compared to only 72.1% F-measure for a smoothed PCFG trained on the same large dataset [3]. Other work on broad coverage parsing with neural networks has only achieved 61.0% F-measure on this dataset [4], even worse than when we used our small dataset and the SSN architecture with the same output function as in that work ("Unb Soft" in table 1). All the above results are for models which only include the part-of-speech tags of words in their input, rather than the specific words. State-of-the-art statistical parsers, which all use specific words, reach performance levels around 89% or 90% F-measure. When an SSN parser is trained using a relatively small vocabulary of words, results reach 89.1% F-measure [8].

5 Conclusions and Future Research

The experimental results show a clear advantage for both of the unbounded activation functions we have proposed in this article over the standard method. This shows that estimating an unbounded probability distribution directly does impose fewer undesirable biases than converting the unbounded distribution into a sequence of bounded distributions and then estimating the bounded distributions. This improvement also suggests that these alternative statistical estimation methods could improve the performance of many applications in unbounded domains, in particular statistical parsers. The unbounded sigmoid function does slightly better than the unbounded softmax function on the testing set, but the results on the validation set show that this is not likely to be a reliable difference. The fact that two quite different sets of statistical assumptions lead to roughly equivalent performance suggests that the training is able to produce hidden layer representations which fit the statistical properties required by the output layer computations. This is particularly interesting for the unbounded softmax function, since its statistical assumptions are less easy to motivate.

In on-going research we have looked at other output activation functions which weaken some of statistical assumptions made in this article. In particular we have looked at weakening the assumption $p(\mathbf{z}_i|Y_j) \approx p(\mathbf{z}_i|\neg Y_i)$, replacing $\neg Y_i$ with a finite number of relationships between i and the correct category (such as i is a phrase which is above or below the correct phrase in the parse tree). These modifications resulted in networks which did not exhibit stable convergence during training. We hypothesize that this is caused by multiple conflicting statistical properties being required of the hidden layer representations, which training is not able to satisfy. Future research could try to solve this problem, and could address weakening other statistical assumptions, such as the uniform prior used for $P(Y_i)$. The promising results obtained here suggest further work is justified.

References

[1] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK, 1995.

[2] J.S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fogelman Soulié and J. Hérault, editors, *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236. Springer-Verlag, New York, 1990.

[3] Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18(4):33–44, 1997.

[4] F. Costa, V. Lombardo, P. Frasconi, and G. Soda. Wide coverage incremental parsing by learning attachment preferences. In *Proc. of the Conf. of the Italian Association for Artificial Intelligence*, 2001.

[5] Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

[6] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9:768–786, 1998.

[7] James Henderson. A neural network parser that handles sparse data. In *Proc. 6th Int. Workshop on Parsing Technologies*, pages 123–134, Trento, Italy, 2000.

[8] James Henderson. Structural bias in inducing representations for probabilistic natural language parsing. In *Proc. 13th Int. Conf. on Artificial Neural Networks*, Istanbul, Turkey, 2003.

[9] Peter Lane and James Henderson. Incremental syntactic parsing of natural

language corpora with simple synchrony networks. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):219–231, 2001.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing, Vol 1*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[11] Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16:417–451, 1993.