Grammar Inference Using Recurrent Neural Networks

Andrew Smith
Department of Computer Science
University of California, San Diego
La Jolla, CA 92037
atsmith@cs.ucsd.edu

Abstract

This paper compares the performance of two recurrent neural network models learning the grammar of simple English sentences and introduces a method for creating useful negative training examples from a set of grammatical sentences. The learning task is cast as a classification problem as opposed to a prediction problem; networks are taught to classify a given sequence as grammatical or ungrammatical. The two neural network models compared are the simple architecture proposed by Elman in [1], and the Long Short-Term Memory (LSTM) network from [4]. For comparison, a Naive Bayes (NB) classifier is also trained with bigrams to accomplish this task. Only the Elman network learns the training data better than NB, and neither network generalizes better than NB on sentences longer than the training sentences. The LSTM network does not generalize as well as the Elman network on a simple regular grammar, but shows comparable ability to generalize knowledge of English Grammar.

1 Background

Recurrent neural networks (RNNs) are widely used as sequence processing devices. More traditional models, such as the Hidden Markov Model (HMM), are confined to a discrete state space, whereas state is represented in a RNN by the pattern of activation of hidden units, each of which is typically composed of continuous values [2]. Because finite-state models cannot represent the complex structures commonly found in natural language, many researchers have developed RNN architectures to be used for grammatical inference [5]. RNNs exhibit complex dynamical behavior and have been shown capable of representing these structures [5].

There are currently many different architectures and types of RNNs in use. One issue this paper addresses is whether the purported benefits of recent RNN models outweigh the benefit of simplicity of older models in the domain of grammatical inference. Specifically two types of networks are compared, the simple recurrent network proposed by Elman and used in [1] and [5] to infer the grammar of natural language, and the Long Short-Term Memory networks used in [2] to learn simple grammars.

Classifying sentences as correct or incorrect English clearly requires memory. For example,

a classifier needs to remember whether a subject was plural or singular so when the verb appears, agreement can be checked. The recurrent backpropagation algorithm (BPTT) can take a very long time to teach a network to remember information. LSTM networks have been shown to learn to remember much more quickly and reliably, but it is not clear that parsing English requires memory over time intervals long enough to give LSTM networks an advantage over simple networks using BPTT [4]. To attempt to answer this question, the ability of the networks to learn to remember is directly investigated with a simple regular grammar. Then the networks are taught English grammar.

Another issue investigated here is the extent to which the grammar inferred from short sentences can be used to classify longer sentences. Certainly, longer sentences are likely to contain some structures not present in short sentences, such as nested clauses and phrases; however, in many cases, short sentences can be used to learn hierarchical structures that can be combined to form longer sentences. For example, a short sentence might contain a participial phrase as a subject and another short sentence might contain a participial phrase as a direct object. If this rule, that participial phrases can be used where nouns are used, is correctly learned, a longer sentence in which both the subject and direct object are participial phrases should be correctly classified. In [1] Elman investigates the ability of RNNs to learn these hierarchies, but not the extent to which they can be learned from short sentences and applied to long sentences.

2 Experimental design

The task of the networks is to decide if a given symbolic sequence is grammatical or ungrammatical. This differs from the usual approach ([1] and [2], for example) to grammatical inference which is to build a system that predicts the next input symbol based on the previous input symbols. The classification task is only to decide whether a finite input sequence was grammatical or not. The reason for this different approach is motivated by the fact that, given a sentence, any native speaker can immediately identify its grammatical status without consciously performing any continuous prediction task. To test the ability of the networks to generalize to longer sentences, the networks are trained on relatively short sequences and then tested with longer sequences.

The input to the networks is a sequence of symbols. Since the alphabet of symbols is small (6 for the regular grammar, and 58 for English grammar; see the Experiments section), a "one-hot" encoding of the input is used; there is a one-to-one mapping between symbols and input units, and only one input unit has high activation at a time. The networks have one output unit which is activated after the last symbol of a sequence is input to the network to indicate that the sequence was grammatical.

2.1 Elman networks

In [1], Elman proposes a simple recurrent neural network model. The Elman network consists of an input layer, a fully connected hidden layer, and an output layer. The connections between the hidden units are the only recurrent connections, and the activations of the hidden units encode the internal state of the network. To determine the activations of the hidden units, each hidden unit calculates a weighted sum of the inputs, and a weighted sum of the activations of the hidden units during the previous time-step. (see figure 2)

To train an Elman network, the gradient of the error with respect to the weights is calculated, and the weights are incrementally adjusted to reduce the error. Backpropagation Through Time (BPTT) is used to calculate this gradient. Since the sequences are of finite length, the exact gradient can be used and there is no need for truncation.

To improve the efficiency of the training, several methods from [8] are implemented. [8]

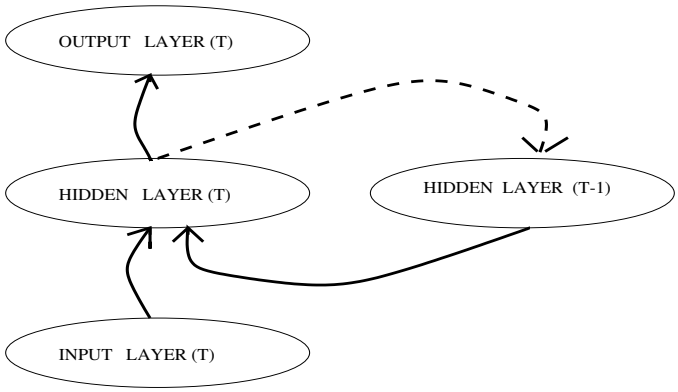

Figure 1: The Elman network can be interpreted as storing the current activations of the hidden units to be used in the calculation of the future activations of the hidden units. Solid arrows indicate weighted connections from every unit in the originating layer to every unit in the destination layer. The dashed arrow indicates the hidden layer storing its activations for use in the next time-step, not weighted connections. All non-input units have a connection from a bias unit, which is not shown.

suggests an activation function called "funny-tanh" leads to faster convergence, but the logistic sigmoid was experimentally found to work better. Target values of 0.8 and 0.2 are used instead of 1.0 and 0.0, to prevent the weights from being driven to infinity. Small momentum terms (usually .1) are added to dampen oscillations of the weight adjustments. Both batch and stochastic learning were implemented, but stochastic learning was found to be much faster and is used for all experiments. The learning rate schedule used in [5] was implemented, but found to be not much better than a constant learning rate $(= 0.5)$, which is used in all experiments. The initial weights are chosen randomly, but scaled so that units are expected to start out in the near-linear region of the logistic sigmoid.

2.2 Long short-term memory networks

One difficulty in training simple recurrent neural networks with BPTT is that the gradient of the error with respect to previous inputs quickly vanishes as the time lag between the output and the relevant input increases [2]. For this reason it can be difficult to teach a network to remember values for an extended period of time, especially when input values are always many time-steps before the outputs they affect. Long Short-Term Memory (LSTM) networks attempt to overcome this difficulty and have been shown to quickly learn to remember associations between outputs and relevant inputs even when the minimum time lag between the two is more than 1000 time-steps. LSTM networks are composed of special memory units with self-connections of weight 1.0 and a linear activation function (the identity function), called "constant error carousels" (CECs). This recurrent connection causes the flow of error, backwards through time, to remain constant. Special multiplicative units, called input and output gates, learn when to allow access to the CEC based on the current input and the memory cell output at the previous time-step [4]. Recently, [9] has suggested two improvements to the basic LSTM memory cell architecture, the addition of "forget gates" which can decide to reset the CEC, and allowing the gates to use the contents of the CEC as inputs, through "peephole connections." (see figure 2)

This paper uses the LSTM architecture extended with forget gates and peephole connections. This was found to converge faster than the basic LSTM architecture. Originally, BPTT was used to train LSTM networks by calculating the gradient of the output error

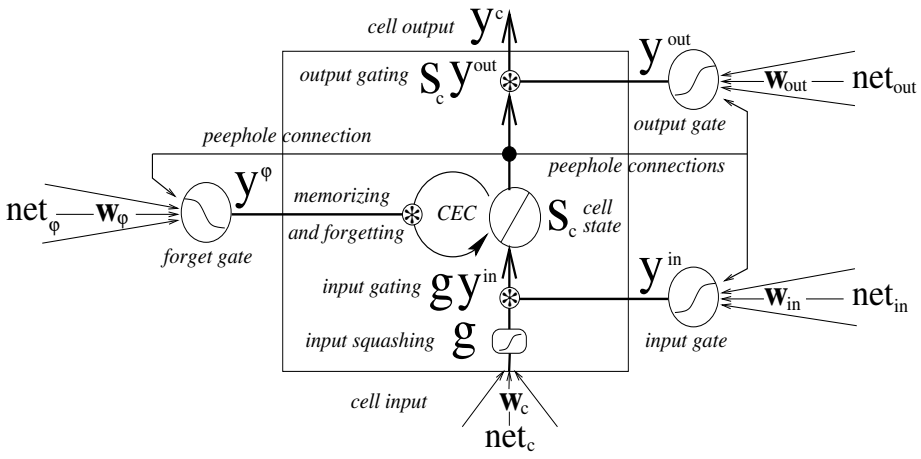

Figure 2: This shows one memory cell of the LSTM network with peephole connections and forget gates. All incoming connections are from the input layer, the output of the memory cells during the previous time-step, and a bias unit (with activation 1.0). The subscript ϕ indexes forget gates, **in** indexes input gates, **out** indexes output gates, and **c** indexes memory cells. The input to the memory cell $\mathbf{g}(net_c)$ is multiplied by the activation of the input gate $\mathbf{y}^{in}$. Similarly, the output of the memory cell is the internal state of the memory cell $\mathbf{S}_c$ times the activation of the output gate $\mathbf{y}^{out}$. (figure from [9])

with respect to the weights. In the development of this algorithm, the gradient was compared to an approximation by weight perturbations to ensure BPTT was correctly modified to account for multiplicative gates. Updating the weights of the network by simply moving them against the error gradient did not teach the networks any more than the output bias, even after hours of training. For this reason the LSTM training procedure presented in [9], which is a fusion of truncated BPTT and a modified version of real-time recurrent learning (RTRL), is used for all experiments.

As with Elman networks, a scheduled learning rate did not seem to increase performance, so a constant learning rate $(= 0.1)$ is used. Bias weights to the input gates, forget gates, and output gates are initialized to 1.0, 2.0, and -2.0 respectively, which was empirically found in [2] to increase performance.

2.3 Naive Bayes

To provide a baseline for comparison, a simple Naive Bayes (NB) classifier is trained to learn English grammar. The sequences to be classified are sequences of word tags, drawn from a set of 58 tags (see the next section). The type of NB classifier is the multi-variate Bernoulli model described in [10]. The features are ordered pairs of adjacent word tags. Given this set F of 58^2 features, each dimension of the feature space $t, t \in \{1, ..., |F|\}$ corresponds to an ordered pair of word tags, f_t. We associate with each sentence s_i the set of indicator variables B_{it} which equals 1 if sentence s_i contains feature f_t, and 0 otherwise. To classify a sentence, we make the naive Bayes assumption: the probability of each word tag pair occurring in a sentence is independent of the occurrence of other word tag pairs in the sentence, given the class. The probability of a sentence s_i being generated by class c_j is then the product of the probability of each feature over the entire set of features

$$P(s_i|c_j,\theta) = \prod_{t=1}^{|F|}(B_{it}P(f_t|c_j,\theta) + (1 - B_{it})(1 - P(f_t|c_j,\theta))) \quad (1)$$

The parameters of the model, θ, are the probabilities $P(f_t|c_j,\theta)$ for each feature f_t and each class c_j. The maximum a posteriori estimates of these probabilities are the commonly used frequency counts

$$P(f_t|c_j,\theta) = \frac{.1 + \sum_{i=1}^{|S|} B_{it}P(s_i|c_j)}{.2 + \sum_{i=1}^{|S|} P(s_i|c_j)} \quad (2)$$

where $P(s_i|c_j)$ is one if s_i is in class c_j and zero otherwise. The .1 and .2 are used to define the Laplace priors, and were determined experimentally to work well. These values prevent probabilities from vanishing if a novel sentence contains a feature never seen in the training set, while emphasizing the importance of the observed word pair tags more than the unobserved ones. For this paper, two NB classifiers were trained, one for the grammatical sentences, and one for the ungrammatical sentences. A novel sentences is classified as grammatical if the probability of it being generated by the grammatical sentence classifier is higher than the probability of it being generated by the ungrammatical sentence classifier. (See [10] for a more detailed discussion of multi-variate Bernoulli NB classifiers.)

3 Experiments

3.1 Simple regular grammar

The first test is to distinguish between two similar regular grammars, $a(e|f)^*c \mid b(e|f)^*d$ and $a(e|f)^*d \mid b(e|f)^*c$. This task requires one bit of memory, whether the first character was an a or a b. For a neural network to correctly classify a sequence, some internal representation of that bit of knowledge must persist through the presentation of an arbitrary number of "distractor symbols," the e's and the f's, until the presentation of the final symbol.

3.2 English grammar

The second test is to learn the grammar of simple English sentences. The sentences are drawn from the SUSANNE corpus, a 130,000 word subset of the BROWN corpus consisting of written American English. The sentences used for this experiment are simple, grammatical, non-questions without punctuation. The SUSANNE corpus is tagged according to the scheme defined in [7] which consists of 353 different word tags. For this experiment, the SUSANNE tagset is reduced to 58 tags by grouping the original tags into logical categories. This reduced tagset consists of the commonly accepted parts of speech divided into sub-categories. For example, nouns are divided into three categories singular, plural, and ambiguous (ie "fish") and pronouns are divided by person and case. More than half of these 58 tags are verbs. In order for the reduced tagset to be descriptive enough to contain basic grammatical information, verbs are divided by conjugation or participle, tense, and obligatory transitive or intransitive (or ambiguous). There are also special tags for forms of the verb "be," "have," and "do," as well as tags for modal verbs and other special verbs.

To construct ungrammatical sentences for use in training, each grammatical sentences is altered in the following way. For each tag in the sentence, the tagset is partitioned into

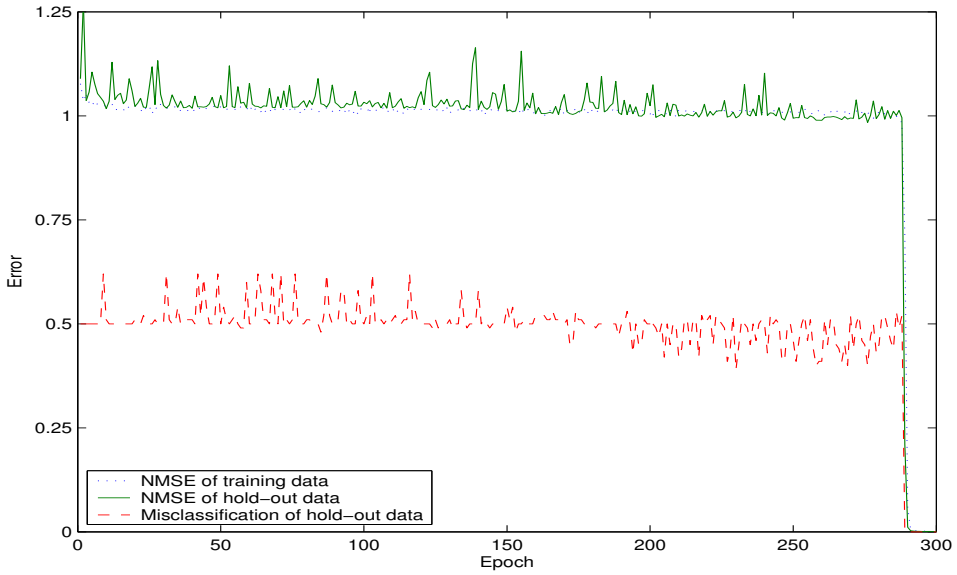

Figure 3: This figure shows the progress of an Elman network learning the simple regular grammar.

those tags that could legally replace it, and those tags that could not, given all other tags in the sentence. This method produces many examples of ungrammatical tag sequences and it has the side-effect of producing more positive training examples. For example, the SUSANNE corpus contains 37 distinct grammatical tag sequences of length three. Using the method outlined above, this is extended to 622 more grammatical tag sequences, and 4274 ungrammatical tag sequences.

This method of creating negative training examples has the property that each ungrammatical tag sequence differs from a grammatical sequence in only one tag. These negative training examples lie near the boundary between grammatical and ungrammatical sequences, and are therefore more useful from a learning perspective than other ungrammatical sequences, such as five coordinating conjunctions in a row.

4 Results

4.1 Simple regular grammar

Figure 3 shows the progress of an Elman network learning to distinguish the simple regular grammars described in the previous section. The network has ten hidden units and is trained with a learning rate of 0.5 and a momentum factor of 0.1. The training set consists of 1000 sequences of uniform random length from 2 to 15, divided evenly between the two grammars. The hold-out set consists of 200 sequences of uniform random length from 2 to 30, also divided evenly. After training, this network generalizes perfectly, correctly classifying all test sequences, which range in length from 50 to 10,000. The network learns the ability to store values indefinitely until the input becomes relevant.

Figure 4 shows the progress of a LSTM network learning the simple regular grammar. The network has three memory cells and is trained with a learning rate of 0.1 and a momentum factor of 0.1. A LSTM network with three memory cells contains approximately the same number of adjustable parameters as an Elman network with ten hidden units (used in figure

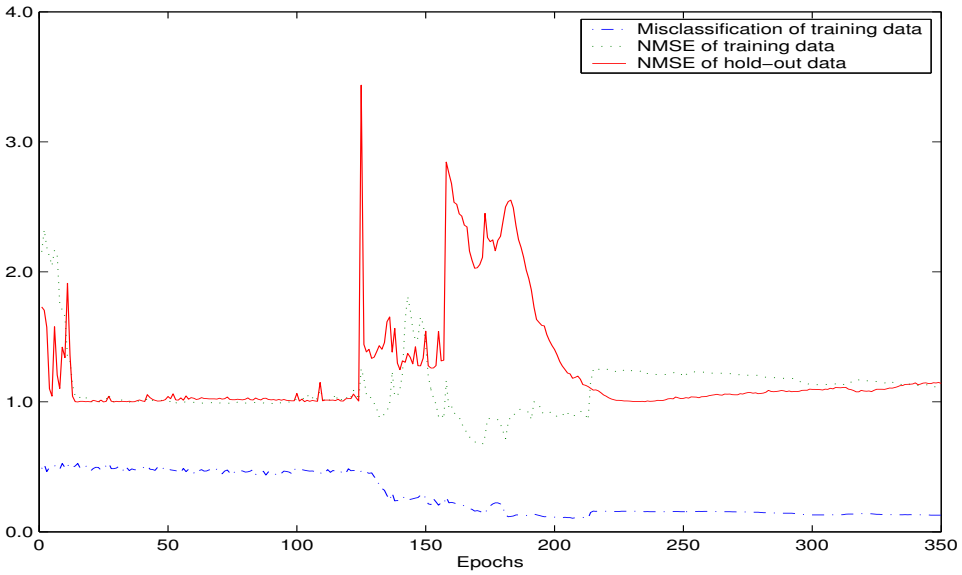

Figure 4: This figure shows the progress of a LSTM network learning the simple regular grammar.

3). The same training and hold-out sets are used as with the Elman network. Despite the fact that the normalized mean-squared-error (NMSE) never drops much below 1.0, the misclassification rate of the training data approaches a minimum of about 20 percent. After training, this network shows virtually no ability to generalize to longer sequences.

Clearly a Naive Bayes classifier cannot learn this grammar for sequences of length greater than two, so that method of classification is not tested.

4.2 English grammar

Figure 5 shows the progress of an Elman network learning English grammar. The training set consists of approximately 12,000 grammatical and ungrammatical sentences of length two, three, or four. The hold-out set consists of 230 sentences of length five. The network has twenty three hidden units and is trained with a learning rate of 0.1 and a momentum factor of 0.1. Despite classifying the training data with only two percent error, the hold-out data is never classified more accurately than with 22 percent error (at the 62^{nd} epoch).

Figure 6 shows the progress of a LSTM network learning English grammar. The same training and hold-out sets are used as with the Elman network. The network has six memory cells and is trained with a learning rate of 0.1 and a momentum factor of 0.1. A LSTM network with six memory cells contains approximately the same number of adjustable parameters as an Elman network with twenty three hidden units (used in figure 5). This network does not learn the training data nearly as well as the Elman network, but the best generalization to the hold-out data, at the 42^{nd} epoch, has a comparable 23 percent error.

Figure 7 shows the ability of both networks to generalize to longer sentences. For each sentence length, the same number of grammatical and ungrammatical sequences are tested (so guessing randomly would yield an accuracy of 0.5). The accuracy of a Naive Bayes classifier trained on bigrams of adjacent word tags is also shown for comparison. Neither network clearly generalizes better than the Naive Bayes classifier, though the Elman

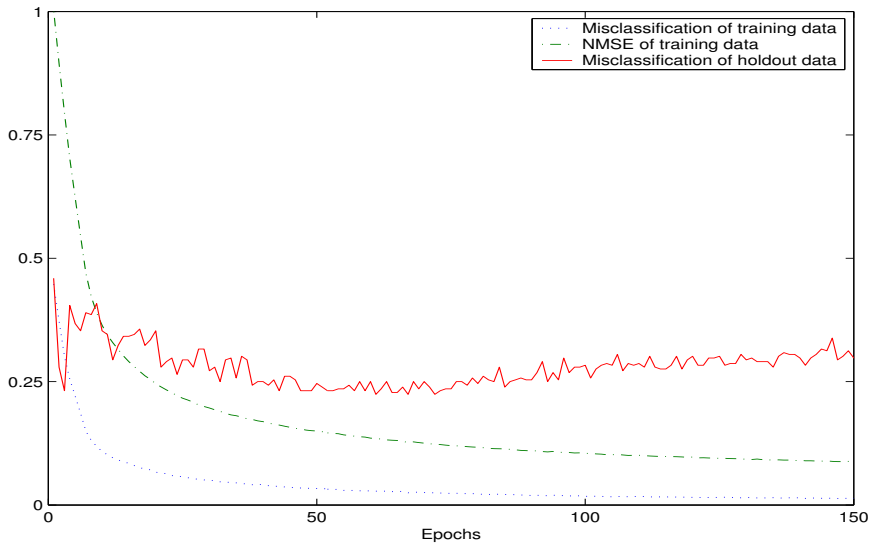

Figure 5: This figure shows the progress of an Elman network learning English grammar.

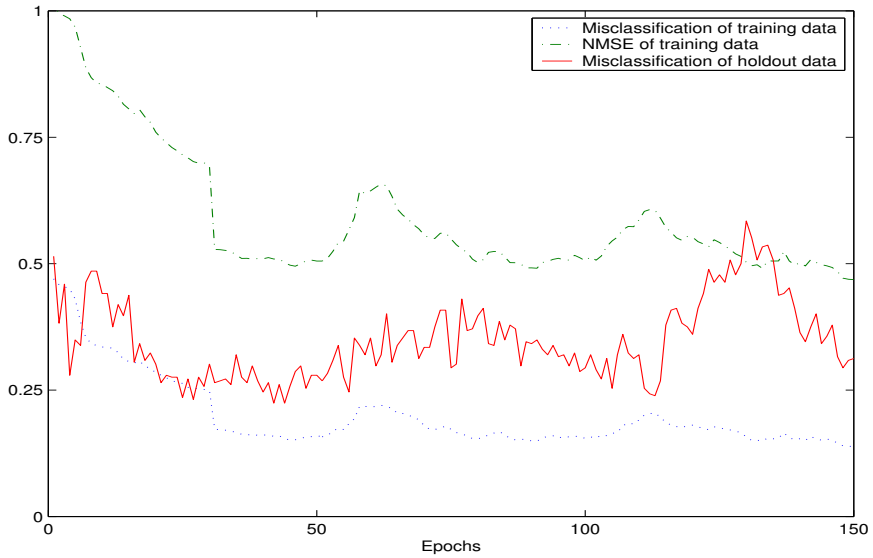

Figure 6: This figure shows the progress of a LSTM network learning English grammar.

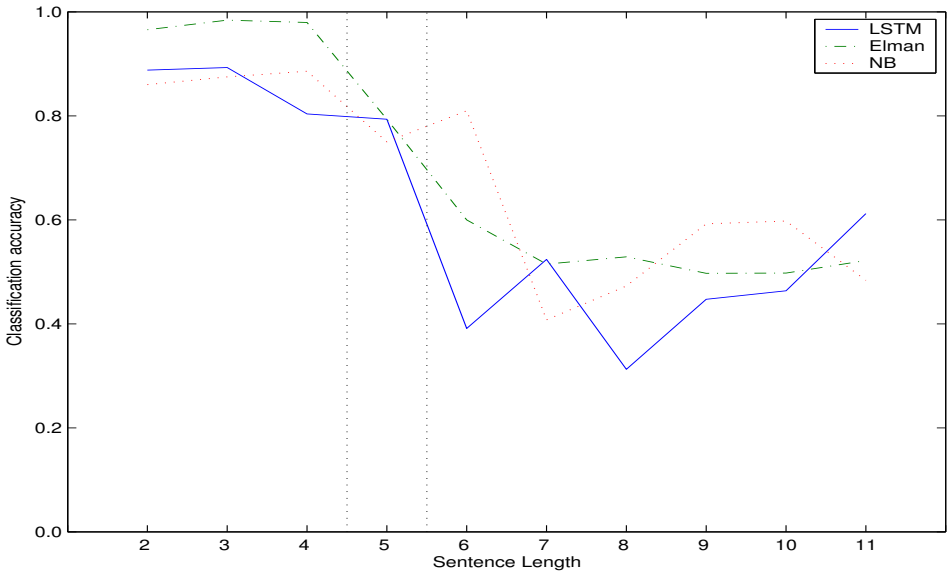

Figure 7: This figure shows the ability of trained Elman and LSTM networks to generalize to sentences longer than those in the training set. The vertical dotted lines separate the classification accuracy of the training data, on the left, the classification accuracy of the hold-out data, and the classification accuracy of novel data, on the right.

network shows somewhat more graceful degradation.

5 Conclusions

The results of the networks learning the English grammar and being able to generalize were generally much less successful than those reported in [1] and [5]. Clearly, the NB classifier is insufficient for recognizing general grammatical structures spanning more than two words. That the networks did not classify long sentences better than the NB classifier indicates that the hierarchical grammatical relationships in [1] and [5] were not learned. This is possibly due to the fact that the task in this paper is different in several respects from the tasks in those papers.

First, as previously stated, the task at hand is to classify sequences as grammatical or not after their presentation to the network. While this may seem like an easier task than continuously predicting the next symbol of a sequence, it allows for fewer opportunities to modify the network behavior. An error signal is generated whenever the network output differs from the target output, if one exists. In a continuous prediction task a target output exists for every time-step, resulting in an error signal being generated and weights being updated every time-step. However, in a sequence classification task, only one error signal is generated every time a complete sequence is presented to the network. The result is that the classification task provides fewer opportunities to correct the network behavior.

Another difference is that the task attempted here uses many more word tags than those used in [1], [2], and [5], which used about 20 different input symbols. The goal was to provide a minimal set of tags that could encode as much grammatical information as possible. The task in this paper was to learn a subset of English grammar that was much less restrictive than those in [1] and [5], and was therefore more difficult.

The nature of the generalization that was tested in this paper was different from that of [1] and [5]. Those papers trained and tested networks on sentences of comparable length. Here, the goal was to provide a training set consisting of short sentences with as many different grammatical structures as possible, and then test the ability of the networks to apply knowledge of those structures to classify longer sentences. The networks were not able to classify long sentences any better than guessing randomly. Future work should include an exploration of trends in the misclassification of longer sentences; do misclassified sentences contain structures not seen in the training sentences?

Perhaps the length of the training sentences was too small to contain a large sample of the common grammatical structures found in English. This possibility is supported by the fact that the Elman network learned the English training data so well, generalized reasonably well to sentences one word longer, but did not generalize well to longer sentences; sentences differing in length by only one word seem likely to contain comparable grammatical structures. This could be explored by increasing the length of sentences in the training and test sets.

These negative results should not be interpreted as evidence that recurrent neural networks cannot learn English (indeed, human brains demonstrate otherwise), but they suggest that the specific task required of the networks in this paper is inherently difficult. When sentences are reduced to sequences of word tags, the task of learning a natural language is changed to the point that any biological inspiration for training artificial neural networks no longer applies; word tags are not the linguistic data on which biological neural networks naturally operate. As previously stated, any native speaker can quickly classify sentences as grammatical or ungrammatical, however classification of word tag sequences is much slower. Incidentally, this made the creation of the data set used for English grammar learning a very slow process. The "point" of language is to represent ideas, and therefore the ultimate goal of a natural language model should be to infer the ideas and concepts expressed in a sentence, and inferring the grammar of tag sequences does not necessarily bring us closer to this goal.

Identifying the grammatical structure of a sentence is certainly a great step towards inferring its meaning, however doing so using only word tags ignores the interactions between the semantics and the syntax [1]. Consider the fragment "natural language processing system." Depending on the context, this can mean either a system for processing natural language, or a natural system (ie biological) for processing language. Deciding which noun "natural" modifies is impossible with just word tags. To completely infer the meaning of the fragment, some semantic knowledge of the context must be learned.

The previous example illustrates the role of semantic interactions in grammatical disambiguation. Another type is of semantic interaction is when it "facilitates [the] parsing [of] structures which are otherwise hard to process [1]." Consider two example sentences from [1]: "The cat the dog the mouse saw chased ran away" and "The planet the astronomer the university hired saw exploded." The second is much easier to parse because we know that planets explode, astronomers see, and universities hire, whereas in the first sentence semantic interactions do not help. Cats, dogs, and mice can all see, chase, and run. A complete model of language should be able to use the semantic interactions to aid parsing. This example also illustrates that the difficulties humans have parsing grammatically unambiguous sentences can be highly dependent on semantic interactions.

Another linguistic task that computational language models must explain is the resolution of ambiguous pronouns. This task is also greatly facilitated by considering semantic interactions. For example, in the sentence "I dropped the glass on the floor and it broke." the pronoun "it" most likely refers to "glass" and not "floor," but discovering that reference requires the knowledge that "glass" and "break" is a much more probable subject-predicate pair than "floor" and "break." This resolution would also be impossible using only word

tags.

Future natural language processing models must eventually include the capability to learn semantic interactions and context. The restriction of a model to part of speech information clearly precludes this capability and has no obvious biological counterpart.

References

[1] J.L. Elman (1991) Distributed Representations, simple recurrent networks, and grammatical structure. *Machine Learning 7, p. 195*

[2] F.A. Gers, J. Schmidhuber (2001) LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages *IEEE Transactions on Neural Networks*

[3] C.L. Giles, C.B.Miller (1992) Extracting and Learning an Unknown Grammar with Recurrent Neural Networks, *Advances in Neural Information Processing Systems 4, J.E. Moody, S.J. Hanson and R.P. Lippman (Eds)*

[4] S. Hochreiter, J. Schmidhuber (1997) Long Short-Term Memory, *Neural Computation 9, 8 1735-1780*

[5] Steve Lawrence, C. Lee Giles, Sandiway Fong (1998) Natural Language Grammatical Inference with Recurrent Neural Networks, *IEEE Transactions on Knowledge and Data Engineering*

[6] Michael C. Mozer (1994) Neural Net Architectures for Temporal Sequence Processing, *Predicting the Future and Understanding the Past, A. Weigent and N. Gershenfeld (Eds)*

[7] G. Sampson (1995) English for the Computer The SUSANNE Corpus and Analytic Scheme, *Clarendon Press, Oxford*

[8] Y. LeCunn, L. Bottou, G.B. Orr, K.R. Müller (1998) Efficient Backprop *Neural Networks Tricks of the Trade, chapter 1., G.B. Orr and K.R. Müller (Eds)*

[9] F.A. Gers and J. Schmidhuber (2000) Recurrent nets that time and count, *Prsc. IJCNN'2000, Int. Joint Conf. on Neural Networks* IEEE Computer Society, 2000.

[10] A. McCallum and K. Nigam (1998) A comparison of event models for naive bayes text classification, *AIII-98 Workshop on Learning for Text Catigorization, p. 41-48*