A Discrete Probabilistic Memory Model for Discovering Dependencies in Time

Sepp Hochreiter and Michael C. Mozer

Department of Computer Science
University of Colorado
Boulder, CO 80309–0430
{hochreit,mozer}@cs.colorado.edu

Abstract. Many domains of machine learning involve discovering dependencies and structure over time. In the most complex of domains, *long-term temporal dependencies* are present. Neural network models such as LSTM have been developed to deal with long-term dependencies, but the continuous nature of neural networks is not well suited to discrete symbol processing tasks. Further, the mathematical underpinnings of neural networks are unclear, and gradient descent learning of recurrent neural networks seems particularly susceptible to local optima. We introduce a novel architecture for discovering dependencies in time. The architecture is formed by combining two variants of a hidden Markov model (HMM)—the factorial HMM and the input-output HMM—and adding a further strong constraint that requires the model to behave as a latch-and-store memory (the same constraint exploited in LSTM). This model, called an MIOFHMM, can learn structure that other variants of the HMM cannot, and can generalize better than LSTM on test sequences that have different statistical properties (different lengths, different types of noise) than training sequences. However, the MIOFHMM is slower to train and is more susceptible to local optima than LSTM.

1 Introduction

Many domains of machine learning involve discovering dependencies and structure over time. Example domains include speech recognition, process control, and time series prediction. In the most complex of domains, *long-term temporal dependencies* are present. A long-term dependency is one in which the observation at time t_u, $o(t_u)$, and the observation at some time in the future, $o(t_v)$, are dependent, where $t_v \gg t_u$, and there is no time t_w, $t_u < t_w < t_v$, such that the dependency between the $o(t_u)$ and $o(t_v)$ can be described in terms of the dependency between $o(t_u)$ and $o(t_w)$ plus the dependency between $o(t_w)$ and $o(t_v)$. To capture the structure present in the temporal sequence, it is therefore necessary to construct a memory holding information about $o(t_u)$ during the time intervening between the observations.

Hidden Markov Models (HMMs) and Recurrent Neural Networks (RNNs) are natural candidates to encode long-term dependencies. However, theoretical and

empirical work argues that learning these dependencies is difficult; for RNNs, see [8, 4, 6], and for HMMs, see [2]. A constrained form of the RNN architecture, called LSTM, has been proposed to learn long-term dependencies using standard learning procedures such as gradient descent [7]. LSTM succeeds because it imposes an inductive bias via hidden units with fixed linear self-recurrent connections of strength 1.0. These units behave as memory cells, responding to learned inputs, and then remaining active indefinitely.

LSTM has three weaknesses. First, LSTM was designed to address many tasks that are intrinsically discrete—they involve classifying sequences of input symbols. A neural network with continuous activation levels does not seem well suited to a discrete domain. Second, gradient-descent learning is slow and in the case of RNNs is particularly prone to encountering local optima. Third, the mathematical underpinnings of neural networks are shaky; for example, the semantics of "activation levels" are ill defined. None of these weaknesses are found in HMMs: HMMs are well suited for discrete inputs and outputs, they use EM procedures for training instead of gradient descent, the HMM has a probabilistic interpretation.

In this paper, we take the inductive bias provided by the LSTM model and incorporate it into a HMM, with the goal of obtaining the benefits of each. Rather than abandoning neural networks for the increasingly popular graphical models, we believe it valuable to exploit the inductive biases discovered by the RNN community in the design of constrained variations of HMMs. The constraint suggested by LSTM involves a fixed state transition probability matrix that implements a latch-and-hold memory.

2 A Discrete Probabilistic Memory Model

A standard HMM generates output sequences. To handle temporally-varying input as well as temporally-varying output, we use an extension known as an input-output HMM [3], in which the state at t, $s(t)$ is conditionally dependent on the previous state, $s(t-1)$, as well as the current input, $x(t)$, and the output, $y(t)$, is conditionally dependent on $s(t)$ and $x(t)$. Further, we allow for a state with compositional structure using a factorial HMM [5]. The particular sort of compositional state we explore in our model is one consisting multiple non-resettable flip-flops—memory elements which can be triggered by particular inputs and will remain unchanged in time thereafter; this same sort of *latch* is the heart of LSTM. Thus, our model is an *memory-based input-output factorial HMM*, which we shorten to MIOFHMM.

Consider factorizing the state into H components, denoted $s_1 \ldots s_H$, each of which we wish to behave as a latch-and-hold memory. Each component is a multinomial random variable with N values. Initially, all components have value $\mathbf{U}$ for "uncommitted"; various inputs can trigger the component to take on values $\mathbf{2}, \ldots, \mathbf{N}$. The constraint on the MIOFHMM is to fix the state transition function, $p(s_i(t) = a | s_1(t-1) = b_1, \ldots, s_H(t-1) = b_H, x(t) = c)$, to δ_{a,b_i} if $b_i \neq \mathbf{U}$, where δ is the Kronecker delta. Once component s_i takes on values

2...**N**, the component can not change its value—it behaves as a memory for the occurrence of an input event. Thus, it has $N-1$ *memory states.*

The restriction on the state transition function that allows each component to store its value indefinitely should have significant benefits in learning: the fixed transition probabilities prevent the transition matrix from becoming irreducible, and hence the limitations on learning temporal dependencies discussed in [2] are not applicable. Each component is further restricted in that it cannot be reset to **U** or any other value, and therefore cannot be re-used. However, we skirt this limitation by allowing multiple components that can be used to store different facets of the input sequence.

To avoid the possibility that all hidden variables become committed at a certain time point and the MIOFHMM becomes unable to track dynamics, we could add conventional hidden variables, i.e., hidden variables without constraints. Another possibility is to soften the constraints by adding a penalty for transitions that were neither 0 or 1, allowing learning to produce non-binary transition probabilities if it was warranted by improved performance.

2.1 Training the MIOFHMM

Training data for the MIOFHMM consists of a set of input and output sequence pairs. The goal of training is to determine model parameters—discrete conditional probability distributions—that maximize the likelihood of the training output sequences given the corresponding training input sequences.

We train the MIOFHMM using the Baum-Welch algorithm [1]. The complexity of the MIOFHMM training procedure is exponential in the number of memory components. Ignoring the memory constraint, the complexity of the Baum-Welsh algorithm for the MIOFHMM is $O\left(T\ N^{2H}\right)$, where T is the sequence length. However, exploiting the memory constraint reduces the complexity to $O(T\ [2N-1]^H)$ which is a savings of a factor $(N/2)^H$. Approximations to Baum-Welsh updating [5] might be used to further accelerate training, although we did not explore such approximations in the present work.

3 Experiments

We perform two sets of experiments. First, we compare our MIOFHMM to conventional IOHMMs and IOFHMMs on the detection of long-term dependencies. The tasks involve a nondeterministic mapping from input sequences to output sequences. Second, we compare the generalization performance of our MIOFHMM to the LSTM recurrent neural network. We use a classification task in which the model must produce an output indicating class membership following the entire input sequence. Each result we present is the average of twenty replications of a model, excluding replications that yielded local optima (as determined by a validation set). In all experiments, the HMM conditional distributions are initialized randomly.

3.1 Comparing the MIOFHMM, IOFHMM, and IOHMM

We begin with a study of learning long-term dependencies using a simple *latch task* that has been used to test various approaches to this problem, e.g., [4, 6]. The essence of the task is that a sequence of inputs are presented, beginning with one of two symbols, **A** or **B**, and after a variable number of time steps, the model must output a corresponding symbol—**U** if the original input was **A**, or **V** if the original input was **B**. Thus, the task requires memorizing the original input. The end of the input sequence is marked by the symbol **E**, and in the intervening time steps, symbols are chosen at random from $\{\mathbf{C}, \mathbf{D}\}$. Except for the final output symbol, any output from $\{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}$ is allowed. We vary T, the number of time steps intervening between the first input and the final output. A sample input sequence for $T = 6$ is **A**–**C**–**C**–**D**–**D**–**C**–**E**, and an allowed output sequence for this input is **Y**–**Z**–**X**–**Y**–**Z**–**Y**–**U**. Five hundred sequences were generated for training and for validation.

We compared the MIOFHMM against the IOHMM and the IOFHMM. The IOHMM is given 5 hidden states, and the IOFHMM and MIOFHMM are given 2 components with 5 hidden states each. (We also tested a version of the MIOFHMM with a single component—essentially an MIOHMM—and the performance was comparable to that of the MIOFHMM.) For each simulation, we record the number of updates required for the model to produce the correct output on the final time step for all examples in the the validation set. If a model does not process the validation set correctly within reasonable number of updates (multiple standard deviations above the mean), we treat the run as having become stuck in a local optimum. We report the mean number of update required for learning and the frequency of becoming stuck in local optima.

Figure 1 shows the number of updates required to train the three models, as a function of the sequence length T. Experiments with the IOHMM and IOFHMM with $T > 5$ were terminated due to lack of CPU cycles. The training time for the IOHMM and IOFHMM appears to scale exponentially with the sequence length for the IOHMM and IOFHMM, consistent with the theoretical results in [2], but training time for the MIOFHMM is flat. The IOHMM and IOFHMM also yielded many local optima: For $T = 5$, the IOHMM and IOFHMM discovered local optima on 35% and 85% of training runs, respectively, whereas the MIOFHMM yielded no local optima. The MIOFHMM clearly outperforms conventional HMM models on tasks involving long-term temporal dependencies. The key feature necessary for the success of the MIOFHMM is the constraint that the state components behave as memory, which is absent from the otherwise identical IOFHMM.

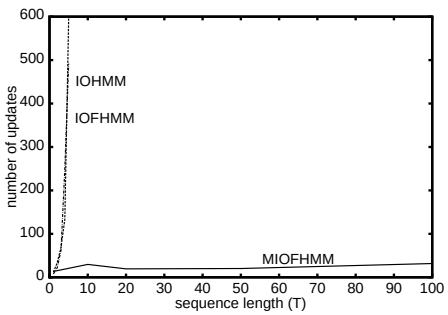

Fig. 1: Number of updates required to learn the latch task for three architectures, as a function of the sequence length (the number of time steps over which an input element must be remembered).

In a second study, we used another task that has previously been explored in the neural net community [6]. The task involves discovering a classification rule for input sequences that depends on the temporal order of events. Each sequence begins with the start symbol **S** and terminates with the end symbol **E**. Embedded in each sequence are two *critical* symbols chosen with replacement from $\{\mathbf{A}, \mathbf{B}\}$. All other input symbols are random, chosen with replacement from $\{\mathbf{C}, \mathbf{D}, \mathbf{F}, \mathbf{G}\}$. The input alphabet thus consists of eight symbols—two start symbols, two critical symbols, and four random symbols. A sample input sequence is **S**–**C**–**A**–**G**–**B**–**F**–**E**. The classification of the sequence depends on the identity and order of the two critical symbols: sequences containing an **A** followed by a **B** are assigned to class 1, a **B** followed by an **A** to class 2, an **A** followed by an **A** to class 3, and a **B** followed by a **B** to class 4. On receiving the final input, the task involved outputting the class label; prior to this input, the task required outputting a special "no class" label.

We compared the IOFHMM to the MIOFHMM. The models had two state components, each having two memory states. We trained the models with 500 examples, and used a validation set of 500 further examples to determine when the model had learned the task. The IOFHMM was never able to learn the task to a criterion of 0 classification errors. Although the MIOFHMM ran into local optima on 65% of trials, it needed only 162 model-parameter updates on average to learn on the remaining 35%. (The local optima obtained by the MIOFHMM in this and other experiments is actually a form of overfitting: the model performs very well on the training set, but not on the validation set. But we call this a local optimum nonetheless because there is a solution for which the model would perform better on training and validation set.) Regardless, the MIOFHMM can succeed on a difficult sequence-ordering task where the IOFHMM fails, due to the constraint imposed on the MIOFHMM that it implement a latch-and-hold memory.

To summarize our two experiments, the latch-and-hold memory constraint imposes a strong inductive bias on the MIOFHMM, which allows it to learn more efficiently and reliably than models such as the IOFHMM and IOHMM which do not exploit this constraint. Of course, the benefit extends only to tasks for which this bias is appropriate—tasks involving storing and remembering sequence elements and their ordering.

3.2 Comparison of MIOFHMMs and LSTM

The experiments in this section explore the generalization capabilities of the MIOFHMM as compared to those of LSTM [6], the neural network model with a latch-and-hold memory constraint. We consider a generalization task that is particularly difficult for machine learning systems, and for which no guarantees of good generalization are possible—where the distributions from which the training and test examples are drawn differ from one another.

In these experiments, we study a variation of the latch task. The input at each time consists of two real-valued elements, a *value* and a *marker*, both in [0,1]. The marker having value 1.0 indicates that the current value is to be stored, and

the marker having value 0.0 indicates that the previously stored value should be retrieved and outputted; a marker value of 0.5 indicates "no action".

Because the MIOFHMM is intrinsically discrete, input values were quantized into one of E equal width intervals in [0,1]. For example, with $E = 10$, the intervals were [0,.1], [.1,.2], etc. Each interval corresponded to a unique input value, which was crossed with the three distinct markers for a total of $3E$ input symbols. The output consisted of E symbols. The LSTM, in contrast, required only two continuous inputs and one continuous output. Its output was judged to be correct if it lay in the correct interval. Although the two architectures are quite different, it is not clear whether one has an advantage over the other on this task. The MIOFHMM benefits from the fact that it receives inputs that are quantized in a task-appropriate manner, whereas the LSTM benefits from the fact that its input has a compositional structure which is task appropriate.

In a first experiment, we trained the models on sequences with lengths between 2 and 10, sampled uniformly, with $E = 10$ intervals, and with the value to be stored always the first element of the sequence. Both models were supplied with 1000 training examples and 1000 validation examples. The models were tested on 1000 generalization examples for various lengths between 10 and 1000. Thus, the challenge was to extrapolate to longer sequences, and hence, to form a memory that could persist over long time intervals.

For this experiment, LSTM was provided with two memory cells. Weights in the LSTM were initialized randomly from $[-.1, .1]$, with an initial bias of -1.0 on each input gate. A learning rate of 0.1 was used. Following each sequence, the weights were updated and the network was reset. The MIOFHMM utilized one state component with 10 memory states. For both models, training continued until all examples in the validation set were classified correctly (i.e., in the correct interval); if this did not occur within a reasonable amount of time, then the training run was considered to have become stuck in a local optimum.

LSTM learned the task efficiently and reliably: training took was 36 seconds of CPU time on a 400 MHz PC (corresponding to 130 training epochs), and never encountered local optima. In contrast, the MIOFHMM required 77 minutes of CPU time (15 updates), and became stuck in local maxima on 47% of runs. In testing, however, the MIOFHMM outshone the LSTM. Figure 2 shows generalization error on test sequences with lengths ranging from 10 to 1000 elements. The test sequence length can be extended to 1000 without any affect on performance for the MIOFHMM, whereas the error rate increases rapidly for the LSTM for sequences having lengths greater than 30.

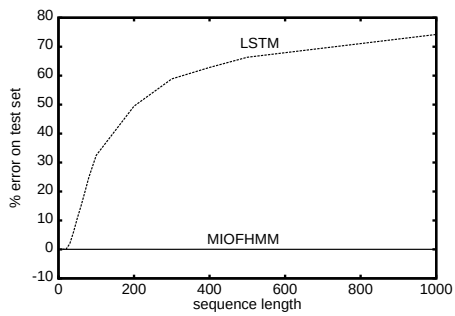

Fig. 2: Generalization error of the LSTM and the MIOFHMM on a latch task with test sequences longer than the longest training sequence.

In a second experiment, we simplified the latch task by presenting sequences whose element to be stored had only $E = 2$ discriminable values, but made the task more difficult in that the value to be stored could occur on any of the first 5 sequence elements. Sequences ranged in length from 5 to 20. As in the first experiment, a marker input of 1.0 was a signal to store an input, and a marker input of 0.0 was a signal to retrieve the stored value. However, we modified the task by replacing the neutral marker value of 0.5 with values ranging from 0.025 to 0.975. In the training set, the neutral marker value was randomly chosen from a uniform distribution over 39 discrete values evenly spaced in [0.025, 0.975]. In the test set, the neutral marker value was randomly chosen from a rectified, discretized Gaussian distribution over the 39 discrete values. The variance of the Gaussian was chosen based on a parameter a, such that with 99% probability the a largest values will be chosen. Consequently, as a is decreased, more marker values in the sequence will become confused with the store (1.0) markers.

As in the first experiment, LSTM training was faster and more reliable: LSTM required 7.75 minutes on average to train (976 epochs), whereas the MIOFHMM required 19 hours (50 updates). LSTM never encountered local optima, whereas MIOFHMM did on 15% of trials. However, in terms of generalization performance, MIOFHMM once again beat out LSTM. Figure 3 shows percentage error on a test set as a function of the noise parameter a. Even for large values of a, MIOFHMM produces fewer errors than LSTM, but as a is decreased, LSTM errors increase dramatically. For small a, the neutral marker was more likely to be a value close to that of the store marker, and consequently, acted as a lure to confuse LSTM. MIOFHMM benefits from the fact that the marker 1.0 and the marker 0.975 are two different symbols, and the similarity structure of the numerical values is therefore irrelevant to performance.

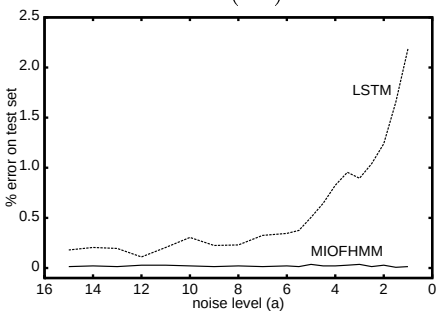

Fig. 3: Generalization error of the LSTM and the MIOFHMM on a latch task with increased noise.

To summarize these two experiments, the discrete nature of the MIOFHMM allows it to reliably hold information for longer periods of time than the continuous LSTM, and also prevents the MIOFHMM from becoming confused by noise, even noise whose statistics in the training and test sets are quite different. Although the two experimental tasks we presented are somewhat contrived, they emphasize that the discrete nature of the MIOHMM can be a virtue that distinguishes from any continuous recurrent neural network model.

4 Conclusions

In this paper, we have introduced a novel architecture for classifying input sequences and for mapping input sequences to output sequences, the MIOFHMM. The MIOFHMM combines two of the virtues of hidden Markov models—the explicit probabilistic framework and the powerful Baum-Welsh training procedure—with a form of inductive bias found to be valuable in recurrent neural network models such as LSTM. The bias forces the MIOFHMM to behave as a latch-and-store memory. We explored four tasks that involved discovering key elements in an input sequence whose detection or temporal order was critical to performance. The MIOFHMM performed well on all these tasks. In contrast, variants of the architecture without the memory constraint—the IOHMM and the IOFHMM—scaled poorly as a function of the temporal span of dependencies. Further, the discrete nature of the MIOFHMM makes it particularly well suited to reaching stable, robust fixed points in state space, and consequently, it appears less susceptible than a continuous model like LSTM to disruption by noise, either in the form of interspersed irrelevant sequence elements or variability in the input at a particular point in time. The one weakness of the MIOFHMM is that—in contrast to our original intuitions—it takes significantly longer than LSTM to train and is more susceptible to local optima.

Acknowledgments
The work was supported by the *Deutsche Forschungsgemeinschaft* (Ho 1749/1-1), McDonnell-Pew award 97-18, and NSF award IBN-9873492.

References

1. L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3:1–8, 1972.
2. Y. Bengio and P. Frasconi. Diffusion of context and credit information in markovian models. *Journal of Artificial Intelligence Research*, 3:249–270, 1995.
3. Y. Bengio and P. Frasconi. An input output HMM architecture. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 427–434. MIT Press, Cambridge MA, 1995.
4. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. on Neural Networks*, 5(2):157–166, 1994.
5. Z. Ghahramani and M. I. Jordan. Factorial hidden markov models. *Machine Learning*, 29:245, 1997.
6. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
7. S. Hochreiter and J. Schmidhuber. LSTM can solve hard long time lag problems. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 473–479. MIT Press, Cambridge MA, 1997.
8. M. C. Mozer. Induction of multiscale temporal structure. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 275–282. San Mateo, CA: Morgan Kaufmann, 1992.