Generating Text with Recurrent Neural Networks

Ilya Sutskever ILYA@CS.UTORONTO.CA
James Martens JMARTENS@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
University of Toronto, 6 King's College Rd., Toronto, ON M5S 3G4 CANADA

Abstract

Recurrent Neural Networks (RNNs) are very powerful sequence models that do not enjoy widespread use because it is extremely difficult to train them properly. Fortunately, recent advances in Hessian-free optimization have been able to overcome the difficulties associated with training RNNs, making it possible to apply them successfully to challenging sequence problems. In this paper we demonstrate the power of RNNs trained with the new Hessian-Free optimizer (HF) by applying them to character-level language modeling tasks. The standard RNN architecture, while effective, is not ideally suited for such tasks, so we introduce a new RNN variant that uses multiplicative (or "gated") connections which allow the current input character to determine the transition matrix from one hidden state vector to the next. After training the multiplicative RNN with the HF optimizer for five days on 8 high-end Graphics Processing Units, we were able to surpass the performance of the best previous single method for character-level language modeling – a hierarchical non-parametric sequence model. To our knowledge this represents the largest recurrent neural network application to date.

1. Introduction

Recurrent Neural Networks (RNNs) form an expressive model family for sequence tasks. They are powerful because they have a high-dimensional hidden state with non-linear dynamics that enable them to remember and process past information. Furthermore, the gradients of the RNN are cheap to compute with backpropagation through time. Despite their attractive qualities, RNNs failed to become a mainstream tool in machine learning due to the difficulty of training them effectively. The cause of this difficulty is the very unstable relationship between the parameters and the dynamics of the hidden states, which manifests itself in the "vanishing/exploding gradients problem" (Bengio et al., 1994). As a result, there has been surprisingly little research on standard RNNs in the last 20 years, and only a few successful applications using large RNNs (Robinson, 2002; Pollastri et al., 2002), including a recent notable application of RNNs as a word-level language model (Mikolov et al., 2010).

Recently, Martens (2010) developed a greatly improved variant of Hessian-Free optimization (HF) which was powerful enough to train very deep neural networks from random initializations. Since an RNN can be viewed as an extremely deep neural network with weight sharing across time, the same HF optimizer should be able to train RNNs. Fortunately, Martens & Sutskever (2011) were able to show that this is indeed the case, and that this form of non-diagonal, 2nd-order optimization provides a principled solution to the vanishing gradients problem in RNNs. Moreover, with the addition of a novel damping mechanism, Martens & Sutskever (2011) showed that the HF optimizer is robust enough to train RNNs, both on pathological synthetic datasets known to be impossible to learn with gradient descent, and on complex and diverse real-world sequence datasets.

The goal of the paper is to demonstrate the power of large RNNs trained with the new Hessian-Free optimizer by applying them to the task of predicting the next character in a stream of text. This is an important problem because a better character-level language model could improve compression of text files (Rissanen & Langdon, 1979) and make it easier for people with physical disabilities to interact with computers (Ward et al., 2000). More speculatively, achieving the asymptotic limit in text compression requires an understanding that is "equivalent to intelligence" (Hutter, 2006). Good compression can be achieved by exploiting simple regularities such as the vocabulary and the syntax of the relevant languages and the shallow associations exem-

Appearing in *Proceedings of the 28^{th} International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

plified by the fact that the word "milk" often occurs soon after the word "cow", but beyond a certain point any improvement in performance must result from a deeper understanding of the text's meaning.

Although standard RNNs are very expressive, we found that achieving competitive results on character-level language modeling required the development of a different type of RNN that was better suited to our application. This new "MRNN" architecture uses multiplicative connections to allow the current input character to determine the hidden-to-hidden weight matrix. We trained MRNNs on over a hundred of megabytes of text for several days using 8 Graphics Processing Units in parallel to perform significantly better than one of the best word-agnostic single character-level language models: the sequence memoizer (Wood et al., 2009; Gasthaus et al., 2010), which is a hierarchical nonparametric Bayesian method. It defines a prior process on the set of predictions at every conceivable context, with judiciously chosen details that make approximate inference computationally tractable. The memoizer induces dependencies between its predictions by making similar predictions at similar contexts. Although intelligent marginalization techniques are able to eliminate all but a relatively small number of the random variables (so the datastructures used scale linearly with the amount of data), its memory requirements are still prohibitively expensive for large datasets, which is a direct consequence of its nonparametric nature.

While our method performs at the state of the art for pure character-level models, its compression performance falls short of the best models which have explicit knowledge of words, the most powerful of these being PAQ8hp12 (Mahoney, 2005). PAQ is a mixture model of a large number of well-chosen context models whose mixing proportions are computed by a neural network whose weights are a function of the current context, and whose predictions are further combined with a neural-network like model. Unlike standard compression techniques, some of PAQ's context models not only consider contiguous contexts but also contexts with "gaps", allowing it to capture some types of longer range structures cheaply. More significantly, PAQ is not word-agnostic, because it uses a combination of character-level and word-level models. PAQ also preprocesses the data with a dictionary of common English words which we disabled, because it gave PAQ an unfair advantage over models that do not use such task-specific (and indeed, English-specific) explicit prior knowledge. The numerous mixture components of PAQ were chosen because they improved performance on a development set, so in this respect PAQ is similar in model complexity to the winning entry of the netflix prize (Bell et al., 2007).

Finally, language models can be used to "generate" language, and to our surprise, the text generated by the MRNNs we trained exhibited a significant amount of interesting and high-level linguistic structure, featuring a large vocabulary, a considerable amount of grammatical structure, and a wide variety of highly plausible proper names that were not in the training set. Mastering the vocabulary of English did not seem to be a problem for the MRNN: it generated very few uncapitalized non-words, and those that it did generate were often very plausible, like "homosomalist" or "un-ameliary". Of particular interest was the fact that the MRNN learned to balance parentheses and quotes over long distances (e.g., 30 characters). A character-level N-gram language model could only do this by modeling 31-grams, and neither Memoizer nor PAQ are representationally capable of balancing parentheses because of their need for exact context matches. In contrast, the MRNN's nonlinear dynamics enables it to extract higher level "knowledge" from the text, and there are no obvious limits to its representational power because of the ability of its hidden states to perform general computation.

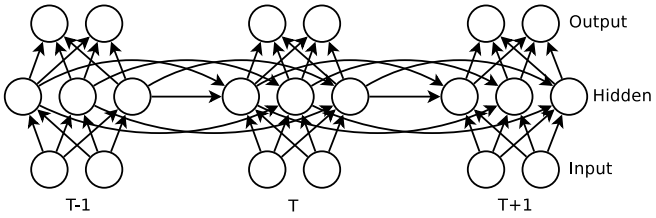

Figure 1. A Recurrent Neural Network is a very deep feedforward neural network whose weights are shared across time. The nonlinear activation function used by the hidden units is the source of the RNN's rich dynamics.

2. Recurrent Neural Networks

A Recurrent Neural Network is a straightforward adaptation of the standard feed-forward neural network to allow it to model sequential data. At each timestep, the RNN receives an input, updates its hidden state, and makes a prediction (fig. 1). The RNN's high dimensional hidden state and nonlinear evolution endow it with great expressive power, enabling the hidden state of the RNN to integrate information over many timesteps and use it to make accurate predictions. Even if the non-linearity used by each unit is quite simple, iterating it over time leads to very rich dynamics.

The standard RNN is formalized as follows: Given a sequence of input vectors $(x_1, \ldots, x_T)$, the RNN computes a sequence of hidden states $(h_1, \ldots, h_T)$ and a sequence of outputs $(o_1, \ldots, o_T)$ by iterating the following equations

for $t = 1$ to T:

$$h_t = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$
$$o_t = W_{oh}h_t + b_o \quad (2)$$

In these equations, W_{hx} is the input-to-hidden weight matrix, W_{hh} is the hidden-to-hidden (or recurrent) weight matrix, W_{oh} is the hidden-to-output weight matrix, and the vectors b_h and b_o are the biases. The undefined expression $W_{hh}h_{t-1}$ at time $t = 1$ is replaced with a special initial bias vector, h_{init}, and the $\tanh$ nonlinearity is applied coordinate-wise.

The gradients of the RNN are easy to compute via backpropagation through time (Rumelhart et al., 1986; Werbos, 1990)[1], so it may seem that RNNs are easy to train with gradient descent. In reality, the relationship between the parameters and the dynamics of the RNN is highly unstable which makes gradient descent ineffective. This intuition was formalized by Hochreiter (1991) and Bengio et al. (1994) who proved that the gradient decays (or, less frequently, blows up) exponentially as it is backpropagated through time, and used this result to argue that RNNs cannot learn long-range temporal dependencies when gradient descent is used for training. In addition, the occasional tendency of the backpropagated gradient to exponentially blow-up greatly increases the variance of the gradients and makes learning very unstable. As gradient descent was the main algorithm used for training neural networks at the time, these theoretical results and the empirical difficulty of training RNNs led to the near abandonment of RNN research.

One way to deal with the inability of gradient descent to learn long-range temporal structure in a standard RNN is to modify the model to include "memory" units that are specially designed to store information over long time periods. This approach is known as "Long-Short Term Memory" (Hochreiter & Schmidhuber, 1997) and has been successfully applied to complex real-world sequence modeling tasks (e.g., Graves & Schmidhuber, 2009). Long-Short Term Memory makes it possible to handle datasets which require long-term memorization and recall but even on these datasets it is outperformed by using a standard RNN trained with the HF optimizer (Martens & Sutskever, 2011).

Another way to avoid the problems associated with backpropagation through time is the Echo State Network (Jaeger & Haas, 2004) which forgoes learning the recurrent connections altogether and only trains the non-recurrent output weights. This is a much easier learning task and it works surprisingly well provided the recurrent connections

[1]In contrast, Dynamic Bayes Networks (Murphy, 2002), the probabilistic analogues of RNNs, do not have an efficient algorithm for computing their gradients.

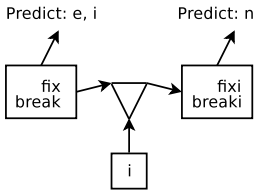

Figure 2. An illustration of the significance of the multiplicative connections (the product is depicted by a triangle). The presence of the multiplicative connections enables the RNN to be sensitive to conjunctions of context and character, allowing different contexts to respond in a qualitatively different manner to the same input character.

are carefully initialized so that the intrinsic dynamics of the network exhibits a rich reservoir of temporal behaviours that can be selectively coupled to the output.

3. The Multiplicative RNN

Having applied a modestly-sized standard RNN architecture to the character-level language modeling problem (where the target output at each time step is defined as the the input character at the next time-step), we found the performance somewhat unsatisfactory, and that while increasing the dimensionality of the hidden state did help, the per-parameter gain in test performance was not sufficient to allow the method to be both practical and competitive with state-of-the-art approaches. We address this problem by proposing a new temporal architecture called the Multiplicative RNN (MRNN) which we will argue is better suited to the language modeling task.

3.1. The Tensor RNN

The dynamics of the RNN's hidden states depend on the hidden-to-hidden matrix and on the inputs. In a standard RNN (as defined by eqs. 1-2), the current input x_t is first transformed via the visible-to-hidden weight matrix W_{hx} and then contributes additively to the input for the current hidden state. A more powerful way for the current input character to affect the hidden state dynamics would be to determine the entire hidden-to-hidden matrix (which defines the non-linear dynamics) in addition to providing an additive bias.

One motivation for this approach came from viewing an RNN as a model of an unbounded tree in which each node is a hidden state vector and each edge is labelled by a character that determines how the parent node gives rise to the child node. This view emphasizes the resemblance of an RNN to a Markov model that stores familiar strings of characters in a tree, and it also makes it clear that the RNN tree is potentially much more powerful than the Markov model because the distributed representation of a node allows dif-

ferent nodes to share knowledge. For example, the character string "ing" is quite probable after "fix" and also quite probable after "break". If the hidden state vectors that represent the two histories "fix" and "break" share a common representation of the fact that this could be the stem of a verb, then this common representation can be acted upon by the character "i" to produce a hidden state that predicts an "n". For this to be a good prediction we require the *conjunction* of the verb-stem representation in the previous hidden state and the character "i". One or other of these alone does not provide half as much evidence for predicting an "n": It is their conjunction that is important. This strongly suggests that we need a multiplicative interaction. To achieve this goal we modify the RNN so that its hidden-to-hidden weight matrix is a (learned) function of the current input x_t:

$$h_t = \tanh\left(W_{hx}x_t + W_{hh}^{(x_t)}h_{t-1} + b_h\right) \quad (3)$$

$$o_t = W_{oh}h_t + b_o \quad (4)$$

These are identical to eqs. 1 and 2, except that W_{hh} is replaced with $W_{hh}^{(x_t)}$, allowing each character to specify a different hidden-to-hidden weight matrix.

It is natural to define $W_{hh}^{(x_t)}$ using a tensor. If we store M matrices, $W_{hh}^{(1)}, \ldots, W_{hh}^{(M)}$, where M is the number of dimensions of x_t, we could define $W_{hh}^{(x_t)}$ by the equation

$$W_{hh}^{(x_t)} = \sum_{m=1}^{M} x_t^{(m)} W_{hh}^{(m)} \quad (5)$$

where $x_t^{(m)}$ is the m-th coordinate of x_t. When the input x_t is a 1-of-M encoding of a character, it is easily seen that every character has an associated weight matrix and $W_{hh}^{(x_t)}$ is the matrix assigned to the character represented by x_t. [2]

3.2. The Multiplicative RNN

The above scheme, while appealing, has a major drawback: Fully general 3-way tensors are not practical because of their size. In particular, if we want to use RNNs with a large number of hidden units (say, 1000) and if the dimensionality of x_t is even moderately large, then the storage required for the tensor $W_{hh}^{(x_t)}$ becomes prohibitive.

It turns out we can remedy the above problem by factoring the tensor $W_{hh}^{(x)}$ (e.g., Taylor & Hinton, 2009). This is done by introducing the three matrices W_{fx}, W_{hf}, and W_{fh}, and reparameterizing the matrix $W_{hh}^{(x_t)}$ by the equation

$$W_{hh}^{(x_t)} = W_{hf} \cdot \mathrm{diag}(W_{fx}x_t) \cdot W_{fh} \quad (6)$$

[2] The above model, applied to discrete inputs represented with their 1-of-M encodings, is the nonlinear version of the Observable Operator Model (OOM; Jaeger, 2000) whose linear nature makes it closely related to an HMM in terms of expressive power.

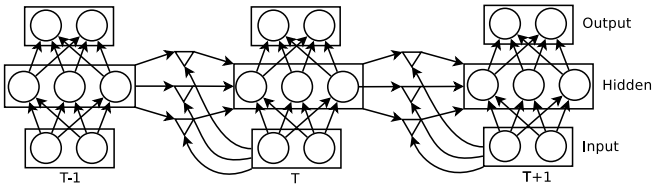

Figure 3. The Multiplicative Recurrent Neural Network "gates" the recurrent weight matrix with the input symbol. Each triangle symbol represents a factor that applies a learned linear filter at each of its two input vertices. The product of the outputs of these two linear filters is then sent, via weighted connections, to all the units connected to the third vertex of the triangle. Consequently every input can synthesize its own hidden-to-hidden weight matrix by determining the gains on all of the factors, each of which represents a rank one hidden-to-hidden weight matrix defined by the outer-product of its incoming and outgoing weight-vectors to the hidden units. The synthesized weight matrices share "structure" because they are all formed by blending the same set of rank one matrices. In contrast, an unconstrained tensor model ensures that each input has a completely separate weight matrix.

If the dimensionality of the vector $W_{fx}x_t$, denoted by F, is sufficiently large, then the factorization is as expressive as the original tensor. Smaller values of F require fewer parameters while hopefully retaining a significant fraction of the tensor's expressive power.

The Multiplicative RNN (MRNN) is the result of factorizing the Tensor RNN by expanding eq. 6 within eq. 3. The MRNN computes the hidden state sequence $(h_1, \ldots, h_T)$, an additional "factor state sequence" $(f_1, \ldots, f_T)$, and the output sequence $(o_1, \ldots, o_T)$ by iterating the equations

$$f_t = \mathrm{diag}(W_{fx}x_t) \cdot W_{fh}h_{t-1} \quad (7)$$

$$h_t = \tanh(W_{hf}f_t + W_{hx}x_t) \quad (8)$$

$$o_t = W_{oh}h_t + b_o \quad (9)$$

which implement the neural network in fig. 3. The tensor factorization of eq. 6 has the interpretation of an additional layer of multiplicative units between each pair of consecutive layers (i.e., the triangles in fig. 3), so the MRNN actually has two steps of nonlinear processing in its hidden states for every input timestep. Each of the multiplicative units outputs the value f_t of eq. 7 which is the product of the outputs of the two linear filters connecting the multiplicative unit to the previous hidden states and to the inputs.

We experimentally verified the advantage of the MRNN over the RNN when the two have the same number of parameters. We trained an RNN with 500 hidden units and an MRNN with 350 hidden units and 350 factors (so the RNN has slightly more parameters) on the "machine learning" dataset (dataset 3 in the experimental section). After extensive training, the MRNN achieved 1.56 bits per character and the RNN achieved 1.65 bits per character on the

test set.

3.3. The difficulty of learning multiplicative units

In an MRNN, the effective weight $W_{ij}^{(c)}$ [3] from hidden unit i to hidden unit j contributed by character c is given by:

$$W_{ij}^{(c)} = \sum_f W_{if} W_{fc} W_{fj} \tag{10}$$

This product of parameters makes gradient descent learning difficult. If, for example, W_{if} is very small and W_{fj} is very large we get a very large deriviative for the very small weight and a very small derivative for the very large weight. Fortunately, this type of difficulty is exactly what second-order methods are good at handling, so multiplicative units should be better handled by a 2nd-order approach like the HF optimizer.

4. The RNN as a Generative Model

The goal of character-level language modeling is to predict the next character in a sequence. More formally, given a training sequence $(x_1, \ldots, x_T)$, the RNN uses the sequence of its output vectors $(o_1, \ldots, o_T)$ to obtain a sequence of predictive distributions $P(x_{t+1}|x_{\leq t}) = \text{softmax}(o_t)$, where the softmax distribution is defined by $P(\text{softmax}(o_t) = j) = \exp(o_t^{(j)})/\sum_k \exp(o_t^{(k)})$. The language modeling objective is to maximize the total log probability of the training sequence $\sum_{t=0}^{T-1} \log P(x_{t+1}|x_{\leq t})$, which implies that the RNN learns a probability distribution over sequences. Even though the hidden units are deterministic, we can sample from an MRNN stochastically because the states of its output units define the conditional distribution $P(x_{t+1}|x_{\leq t})$. We can sample from this conditional distribution to get the next character in a generated string and provide it as the next input to the RNN. This means that the RNN is a directed non-Markov model and, in this respect, it resembles the sequence memoizer (Wood et al., 2009).

5. The experiments

The goal of our experiments is to demonstrate that the MRNN, when trained by HF, learns high-quality language models. We demonstrate this by comparing the MRNN to the sequence memoizer and to PAQ on three real-world language datasets. After splitting each dataset into a training and test set, we trained a large MRNN, a sequence memoizer[4], and PAQ, and report the bits per character (bpc) each model achieves on the test set.

[3] We slightly abuse notation, using $W_{ij}^{(c)}$ to stand for $W_{hh\ ij}^{(c)}$.

[4] Which has no hyper-parameters and strictly speaking isn't 'trained' but rather conditioned the training set.

Owing to its nonparametric nature and the nature of the data-structures it uses, the sequence memoizer is very memory intensive, so it can only be applied to training datasets of roughly 130MB on a machine with 32GB of RAM. In contrast, the MRNN can be applied to datasets of unlimited size although it typically requires considerably more total FLOPS to achieve good performance (but, unlike the memoizer, it is easily parallelized). However, to make the experimental comparison fair, we train the MRNN, the memoizer, and PAQ on datasets of the same size.

5.1. The datasets

We now describe the datasets. Each dataset is a long string of characters from an 86-character alphabet of about 100MB that includes digits and punctuation, together with a special symbol which indicates that the character in the original text was not one of the other 85 characters in our alphabet. The last 10 million characters of each dataset are used as a test set.

1. The first dataset is a sequence of characters from the English Wikipedia. We removed the XML and the Wikipedia markup to clean the dataset. Since Wikipedia is extremely nonuniform, we randomly permuted its articles before partitioning it into a train and a test set.

2. The second dataset is a collection of articles from the New York Times (Sandhaus, 2008).

3. The third dataset is a corpus of machine learning papers. We constructed this dataset by downloading every NIPS and JMLR paper, and converting them to plain text using the pdftotext utility. We then translated a large number of special characters to their ascii equivalents (including non-ascii punctuation, greek letters, and the "fi" and "if" symbol) to clean the dataset, and removed most of the unstructured text by using only sentences consisting of at least 70% alphanumeric characters. Finally, we randomly permuted the papers.

The first two corpora are subsets of larger corpora (over 1GB large), but the semi-online nature of our optimizer makes it easy to train the MRNN on a dataset of any size.

5.2. Training details

To compute the exact gradient of the log probability of the training set (eq. 4), the MRNN needs to process the entire training set sequentially and store the hidden state sequence in order to apply backpropagation-through-time. This is infeasible due to the size of the training set but it is also unnecessary: Training the MRNN on many shorter sequences is just as effective, provided they are several hundred characters or more long. If the sequences are too short, we fail to utilize the ability of the HF optimizer to capture long-

Table 1. This table shows the test bits per character for each experiment, with the training bits in brackets (where available). The MRNN achieves lower bits per character than the sequence memoizer but higher than PAQ on each of the three datasets. The MRNN (full set) column refers to MRNNs trained on the larger (1GB) training corpora (except for the ML dataset which is not a subset of a larger corpus). Note, also, that the improvement resulting from larger dataset is modest, implying that the an MRNN with 1500 units and factors is fairly well-trained with 100MB of text.

DATA SET	MEMOIZER	PAQ	MRNN	MRNN (FULL SET)
WIKI	1.66	1.51	1.60 (1.53)	1.55 (1.54)
NYT	1.49	1.38	1.48 (1.44)	1.47 (1.46)
ML	1.33	1.22	1.31 (1.27)	

term dependencies spanning hundreds of timesteps.

An advantage of using a large number of relatively short sequences over using a single long sequence is that the former is much easier to parallelize. This is essential, since our preliminary experiments suggested that HF applied to MRNNs works best when the gradient is computed using millions of characters and the curvature-matrix vector products are computed using hundreds of thousands of characters. Using a highly parallel system (consisting of 8 high-end GPUs with 4GB of RAM each), we computed the gradient on 160·300=48000 sequences of length 250, of which 8·300=2400 sequences were used to compute the curvature-matrix vector products that are needed for the HF optimizer (Martens & Sutskever, 2011) (so each GPU processes 300 sequences at a time).

The first few characters of any sequence are much harder to predict because they do not have a sufficiently large context, so it is not beneficial to have the MRNN spend neural resources predicting these characters. We take this effect into account by having the MRNN predict only the last 200 timesteps of the 250-long training sequences, thus providing every prediction with at least 50 characters of context.

The Hessian-Free optimizer (Martens, 2010) and its RNN-specialized variant (Martens & Sutskever, 2011) have a small number of meta-parameters that must be specified. We set the structural damping coefficient μ to 0.1, and initialized λ to 10 (see Martens & Sutskever (2011) for a description of these meta-parameters). Our HF implementation uses a different subset of the training data at every iteration, so at a coarse temporal scale it is essentially online. In this setup, training lasted roughly 5 days for each dataset.

We found that a total of 160·150 weight updates was sufficient to adequately train an MRNN. More specifically, we used 160 steps of HF, with each of these steps using a maximum of 150 conjugate gradient iterations to approach the minimum of the quadratic Gauss-Newton-based approximation to the objective function, which remains fixed during the conjugate gradient iterations. The small number of weight updates, each requiring a massive amount of computation, makes the HF optimizer much easier to parallelize than stochastic gradient descent.

In all our experiments we use MRNNs with 1500 hidden units and 1500 factors (F), which have 4,900,000 parameters. The MRNNs were initialized with sparse connections: each unit starts out with 15 nonzero connections to other units (see Martens & Sutskever, 2011). Note that if we unroll the MRNN in time (as in fig. 3) we obtain a neural network with 500 layers of size 1500 if we view the multiplicative units f_t as layers. This is arguably the deepest and largest neural network ever trained.

5.3. The results

The main experimental results are shown in table 5.2. We see that the MRNN predicts the test set more accurately than the sequence memoizer but less accurately than the dictionary-free PAQ on the three datasets.

5.4. Debagging

It is easy to convert a sentence into a bag of words, but it is much harder to convert a bag of words into a meaningful sentence. We name the latter the debagging problem. We perform an experiment where a character-level language model evaluates every possible ordering of the words in the bag, and returns and the ordering it deems best. To make the experiment tractable, we only considered bags of 7 words, giving a search space of size 5040.

For our experiment, we used the MRNN and the memoizer[5] to debag 500 bags of randomly chosen words from "Ana Karenina". We use 11 words for each bag, where the first two and the last two words are used as context to aid debagging the middle seven words.

We say that the model correctly debags a sentence if the correct ordering is assigned the highest log probability. We found that the wikipedia-trained MRNN recovered the correct ordering 34% of the time, and the wikipedia-trained memoizer did so 27% of the time. Given that the problem is "word-level", utilizing large character contexts is essential to achieve good performance.

[5]We were unable to modify the implementation of PAQ to make debagging feasible.

6. Qualitative experiments

In this section we qualitatively investigate the nature of the models learned by the MRNN.

6.1. Samples from the models

The simplest qualitative experiment is to inspect the samples generated by the three MRNNs. The most salient characteristic of the samples is the richness of their vocabularies. Further inspection reveals that the text is mostly grammatical, and that parentheses are usually balanced over many characters. The artifacts of the generated text, such as consecutive commas or quotes, are the result of the data preprocessing and are frequently found in the training set.

6.1.1. Samples from the Wikipedia model

We now present a sample from the Wikipedia model. We use ? to indicate the "unknown" character. The sample below was obtained by running the MRNN less than 10 times and selecting the most intriguing sample. The beginning of the paragraph and the parentheses near the end are particularly interesting. The MRNN was initialized with the phrase "The meaning of life is":

The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. In the show's agreement unanimously resurfaced. The wild pasteured with consistent street forests were incorporated by the 15th century BE. In 1996 the primary rapford undergoes an effort that the reserve conditioning, written into Jewish cities, sleepers to incorporate the .St Eurasia that activates the population. Mar??a Nationale, Kelli, Zedlat-Dukastoe, Florendon, Ptu's thought is. To adapt in most parts of North America, the dynamic fairy Dan please believes, the free speech are much related to the

6.1.2. Samples from the NYT model

Below is a sample from the model trained on the full NYT dataset, where the MRNN was initialized with a single space. The spaces surrounding the punctuation are an artifact of the preprocessing.

while he was giving attention to the second advantage of school building a 2-for-2 stool killed by the Cultures saddled with a half-suit defending the Bharatiya Fernall 's office . Ms . Claire Parters will also have a history temple for him to raise jobs until naked Prodiena to paint baseball partners , provided people to ride both of Manhattan in 1978 , but what was largely directed to China in 1946 , focusing on the trademark period is the sailboat yesterday and comments on whom they obtain overheard within the 120th anniversary , where many civil rights defined , officials said early that forms , " said Bernard J. Marco Jr. of Pennsylvania , was monitoring New York

6.1.3. Samples form the ML model

Finally, we generate text from an MRNN trained on the ML corpus conditioned on the string "Recurrent". This MRNN is also able to balance parentheses (e.g., the third line of the sample):

Recurrent network with the Stiefel information for logistic regression methods Along with either of the algorithms previously (two or more skewprecision) is more similar to the model with the same average mismatched graph. Though this task is to be studied under the reward transform, such as (c) and (C) from the training set, based on target activities for articles a ? 2(6) and (4.3). The PHDPic (PDB) matrix of cav'va using the three relevant information contains for tieming measurements. Moreover, because of the therap tor, the aim is to improve the score to the best patch randomly, but for each initially four data sets. As shown in Figure 11, it is more than 100 steps, we used ?? \to \infty with 1000

6.2. Structured sentence completion

In this section, we investigate the MRNN's response in various situations by sampling from the MRNN's distribution conditioned on a prefix. The goal is to see whether the MRNN is able to generate "plausible" continuations to the initial strings. In our first experiment, we use the Wikipedia MRNN to complete the string "England, Spain, France, Germany,":

England, Spain, France, Germany, and Massachusetts.
England, Spain, France, Germany, cars, and direct schools
England, Spain, France, Germany, , or New Orleans and Uganda.
England, Spain, France, Germany, , Westchester,
Jet State, Springfield, Athleaves and Sorvinhee

In the above completions, the MRNN correctly interpreted the string to be a list of locations, so the generated text was also a part of a list.

Next, we performed a similar experiment using the ML model and the pair of strings "(ABC et al" and "ABC et al". The system has never seen the string "(ABC et al" in its training set (simply because there is no machine learning author named ABC, and its capitalization is particularly uncommon for a citation), so the MRNN needed to generalize over an entirely new author name:

(ABC et al., 2003), ?13?, and for a supervised Mann-¡Whitnaguing
(ABC et al., 2002), based on Lebanon and Haussler, 1995b)
ABC et al. (2003b), or Penalization of Information
ABC et al. (2008) can be evaluated and motivated by
providing optimal estimate

This example shows that the MRNN is sensitive to the initial bracket before "ABC", illustrating its representational power. The above effect is extremely robust. In contrast, both N-gram models and the sequence memoizer cannot make such predictions unless these exact strings (e.g., "(ABC et al., 2003)") occur in the training set, which cannot be counted on. In fact, *any* method which is based on precise context matches is *fundamentally incapable* of utilizing long contexts, because the probability that a long context occurs more than once is vanishingly small. We experimentally verified that neither the sequence memoizer

nor PAQ are not sensitive to the initial bracket.

7. Discussion

Modeling language at the character level seems unnecessarily difficult because we already know that morphemes are the appropriate units for making semantic and syntactic predictions. Converting large databases into sequences of morphemes, however, is non-trivial compared with treating them as character strings. Also, learning which character strings make words is a relatively easy task compared with discovering the subtleties of semantic and syntactic structure. So, given a powerful learning system like an MRNN, the convenience of using characters may outweigh the extra work of having to learn the words All our experiments show that an MRNN finds it very easy to learn words. With the exception of proper names, the generated text contains very few non-words. At the same time, the MRNN also assigns probability to (and occasionally generates) plausible words that do not appear in the training set (e.g., "cryptoliation", "homosomalist", or "un-ameliary"). This is a desirable property which enabled the MRNN to gracefully deal with real words that it nonetheless didn't see in the training set. Predicting the next word by making a sequence of character predictions avoids having to use a huge softmax over all known words and this is so advantageous that some word-level language models actually make up binary "spellings" of words so that they can predict them one bit at a time (Mnih & Hinton, 2009).

MRNNs already learn surprisingly good language models using only 1500 hidden units, and unlike other approaches such as the sequence memoizer and PAQ, they are easy to extend along various dimensions. If we could train much bigger MRNNs with millions of units and billions of connections, it is possible that brute force alone would be sufficient to achieve an even higher standard of performance. But this will of course require considerably more computational power.

Acknowledgements

This work was supported by a Google fellowship and NSERC. The experiments were implemented with software packages by Tieleman (2010) and Mnih (2009).

References

Bell, R.M., Koren, Y., and Volinsky, C. The BellKor solution to the Netflix prize. *KorBell Team's Report to Netflix*, 2007.

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Gasthaus, J., Wood, F., and Teh, Y.W. Lossless compression based on the Sequence Memoizer. In *Data Compression Conference (DCC), 2010*, pp. 337–345. IEEE, 2010.

Graves, A. and Schmidhuber, J. Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in Neural Information Processing Systems*, 21, 2009.

Hochreiter, S. *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis*. PhD thesis, Institut fur Informatik, Technische Universitat Munchen, 1991.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. ISSN 0899-7667.

Hutter, M. The Human knowledge compression prize, 2006.

Jaeger, H. Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398, 2000.

Jaeger, H. and Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78, 2004.

Mahoney, M. Adaptive weighing of context models for lossless data compression. *Florida Inst. Technol., Melbourne, FL, Tech. Rep. CS-2005-16*, 2005.

Martens, J. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*. ICML 2010, 2010.

Martens, J. and Sutskever, I. Training Recurrent Neural Networks with Hessian-Free optimizaiton. *ICML 2011*, 2011.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. Recurrent Neural Network Based Language Model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

Mnih, A. and Hinton, G. A scalable hierarchical distributed language model. *Advances in Neural Information Processing Systems*, 21:1081–1088, 2009.

Mnih, Volodymyr. Cudamat: a CUDA-based matrix class for python. Technical Report UTML TR 2009-004, Department of Computer Science, University of Toronto, November 2009.

Murphy, K.P. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, Citeseer, 2002.

Pollastri, G., Przybylski, D., Rost, B., and Baldi, P. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2):228–235, 2002.

Rissanen, J. and Langdon, G.G. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, 1979.

Robinson, A.J. An application of recurrent nets to phone probability estimation. *Neural Networks, IEEE Transactions on*, 5 (2):298–305, 2002. ISSN 1045-9227.

Rumelhart, D.E., Hintont, G.E., and Williams, R.J. Learning representations by back-propagating errors. *Nature*, 323(6088): 533–536, 1986.

Sandhaus, E. The new york times annotated corpus. *Linguistic Data Consortium, Philadelphia*, 2008.

Taylor, G.W. and Hinton, G.E. Factored conditional restricted boltzmann machines for modeling motion style. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1025–1032. ACM, 2009.

Tieleman, T. Gnumpy: an easy way to use GPU boards in Python. Technical Report UTML TR 2010-002, University of Toronto, Department of Computer Science, 2010.

Ward, D.J., Blackwell, A.F., and MacKay, D.J.C. Dasher–a data entry interface using continuous gestures and language models. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pp. 129–137. ACM, 2000.

Werbos, P.J. Backpropagation through time: What it is and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Wood, F., Archambeau, C., Gasthaus, J., James, L., and Teh, Y.W. A stochastic memoizer for sequence data. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1129–1136. ACM, 2009.