Feed-Forward Networks with Attention Can Solve Some Long-Term Memory Problems

Colin Raffel
LabROSA, Columbia University
craffel@gmail.com

Daniel P. W. Ellis
LabROSA, Columbia University
dpwe@ee.columbia.edu

Abstract

Recurrent neural networks (RNNs) have proven to be powerful models in problems where variable-length sequences are projected into a fixed-dimensional embedded space. Recently, RNNs have been augmented with “attention” mechanisms which allow the network to focus on different parts of an input sequence when computing their output. We propose a simplified model of attention which is applicable to feed-forward neural networks and demonstrate that it can solve some long-term memory problems (specifically, those where temporal order doesn’t matter). In fact, we show empirically that our model can solve these problems for sequence lengths which are both longer and more widely varying than the best results attained with RNNs.

1 Models for Sequential Data

Many problems in machine learning are best formulated using sequential data, i.e. data where a given observation may be dependent on previous observations. Such problems can be coarsely classified as sequence transduction (producing a new sequence given an input sequence), sequence embedding (producing a single label, value, or vector from an entire sequence), or sequence generation (producing a sequence from no input). Appropriate models for these tasks must be able to capture temporal dependencies in sequences, potentially of arbitrary length.

1.1 Recurrent Neural Networks

One such class of models are recurrent neural networks (RNNs), which can be considered a learnable function f whose output $h_t = f(x_t, h_{t-1})$ at time t depends on input x_t and the previous state h_{t-1}. In the supervised setting, the parameters of f are optimized with respect to a loss function which measures f’s performance. A common approach is to use backpropagation through time [1], which “unrolls” the RNN over time steps to compute the gradient of the parameters of f with respect to the loss. Because the same function f is applied repeatedly over time, this gradient can easily explode or vanish [2, 3, 4]. The use of gating architectures [3, 5], sophisticated optimization techniques [6, 7], gradient clipping [2, 8], and/or careful initialization [7, 9, 10, 11] can help mitigate this issue and has facilitated the success of RNNs in a variety of fields (see e.g. [12, 13] for an overview). However, these approaches don’t *solve* the problem of vanishing and exploding gradients, and as a result RNNs are in practice typically only applied in tasks where sequential dependencies span at most hundreds of time steps [6, 7, 11, 3]. Very long sequences can also make training computationally inefficient due to the fact that RNNs must be evaluated sequentially and cannot be fully parallelized.

1.2 Attention

A recently proposed method for easier modeling of long-term dependencies is “attention”. Attention mechanisms allow for a more direct dependence between the state of the model at different points in time. Following the definition from [14], given a model which produces a hidden state h_t at each time step, attention-based models first compute a "context" vector c_t as the weighted mean of the state sequence h by

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$$

where T is the total number of time steps in the input sequence and α_{tj} is a weight computed at each time step t for each state h_j. These context vectors are then used to compute a new state sequence s, where s_t depends on s_{t-1}, c_t and, for sequence prediction, the model's output at $t-1$. The weightings α_{ij} are then computed by

$$\begin{aligned} e_{tj} &= a(s_{t-1}, h_j) \\ \alpha_{tj} &= \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})} \end{aligned}$$

where a is a learned function which can be thought of as computing a scalar importance value for h_j given the value of h_j and the previous state s_{t-1}. This formulation allows the new state sequence s to have more direct access to the entire state sequence h. Attention-based RNNs have proven effective in a variety of sequence transduction tasks [14, 13]. Attention can be seen as analogous to the "soft addressing" mechanisms of the recently proposed Neural Turing Machine [15] and End-To-End Memory Network [16] models.

1.3 Feed-Forward Attention

A straightforward simplification to the attention mechanism described above which would allow it to be applied to sequence embedding tasks could be formulated as follows: Instead of a sequence of context vectors, we produce a single context vector c as

$$\begin{aligned} e_t &= a(h_t) \\ \alpha_t &= \frac{\exp(e_t)}{\sum_{k=1}^{T} \exp(e_k)} \\ c &= \sum_{t=1}^{T} \alpha_t h_t \end{aligned} \tag{1}$$

As before, a is a learnable function, but it now only depends on h_t. In this formulation, attention can be seen as producing a fixed-length embedding c of the input sequence by computing an adaptive weighted average of the state sequence h. A schematic of this form of attention is shown in Figure 1.

A consequence of using an attention mechanism is that the context vector c can model temporal dependencies because attention performs integration over time. It follows that by using this simplified form of attention, a model could perform sequence embedding even if the calculation of h_t was feed-forward, i.e. $h_t = f(x_t)$. Using a feed-forward f could also result in large efficiency gains as the computation could be completely parallelized. While using a feed-forward model for sequential modeling sacrifices the ability to solve some problems, we show that for certain tasks, feed-forward networks with attention can perform arbitrary-length sequence embedding as effectively than RNNs.

We note here that feed-forward models without attention can be used for sequence embedding when the sequence length T is fixed, but when T varies across sequences, some form of temporal integration is necessary. An obvious straightforward choice, which can be seen as an extreme oversimplification of attention, would be to compute c as the unweighted average of the state sequence h_t, i.e.

$$c = \frac{1}{T} \sum_{t=1}^{T} h_t \tag{2}$$

This form of integration is similar to the "global temporal pooling" described in [18], which is based on the "global average pooling" technique of [19]. We will also explore the effectiveness of this approach for sequence embedding.

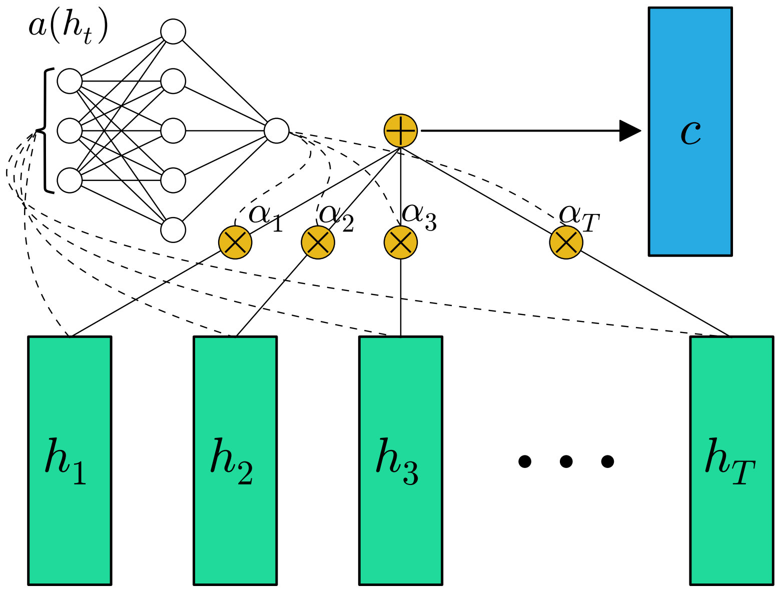

Figure 1: Schematic of our proposed "feedforward" attention mechanism (cf. [17] Figure 1). Vectors in the hidden state sequence h_t are fed into the learnable function $a(h_t)$ to produce a probability vector α. The vector c is computed as a weighted average of h_t, with weighting given by α.

2 Toy Long-Term Memory Problems

A common way to measure the long-term memory capabilities of a given model is to test it on the synthetic problems originally proposed in [3]. In this paper, we will focus on the "addition" and "multiplication" problems, which can be summarized as follows: The input is a two dimensional sequence, where one dimension is a random sequence sampled uniformly from $[0, 1]$ and the other dimension is a "mask" sequence. At two time steps, one in the first ten sequence steps and one before the sequence's midpoint, the "mask" signal is 1; at the first and last time step it is -1; and at all other time steps it is 0. The goal is to perform addition or multiplication on the two values in the noise dimension which co-occur with the 1s in the mask dimension, which is meant to require that a model be able to store the correct values for the duration of the sequence. A sequence is considered correctly processed if the absolute difference between the predicted and target values is less than .04. Slight variants of these tasks have also been used [7, 11, 9, 6], but we will follow the original definition from [3]. These two tasks are only a subset of the synthetic long-term memory problems which have been proposed; we focus on them here because they are the most commonly used and discuss the applicability of feed-forward attention on the remaining problems in Section 3.

2.1 Model Details

For all experiments, we used the following model: First, the state h_t was computed from the input at each time step x_t by

$$h_t = \max(W_{xh}x_t + b_{xh}, 0.01)$$

where $W_{xh} \in \mathbb{R}^{D\times 2}, b_{xh} \in \mathbb{R}^D$. We tested models where the context vector c was then computed either as in Equation 1, with

$$a(h_t) = \tanh(W_{hc}h_t + b_{hc})$$

where $W_{hc} \in \mathbb{R}^{1\times D}, b_{hc} \in \mathbb{R}$, or simply as the unweighted mean of h as in Equation 2. We then computed an intermediate vector

$$s = \max(W_{cs}c + b_{cs}, 0.01)$$

where $W_{cs} \in \mathbb{R}^{D\times D}, b \in \mathbb{R}^D$ from which the output was computed as

$$y = \max(W_{sy}s + b_{sy}, 0.01)$$

Addition						
Sequence length (T_0)	50	100	500	1000	5000	10000
Attention	1	1	1	1	2	3
Unweighted	1	1	1	2	8	17

Multiplication						
Sequence length (T_0)	50	100	500	1000	5000	10000
Attention	1	2	4	2	15	6
Unweighted	2	2	8	33	89.8%	80.8%

Table 1: Number of epochs required to achieve perfect accuracy, or accuracy after 100 epochs (greyed-out values), for the experiment described in Section 2.2.

where $W_{sy} \in \mathbb{R}^{1\times D}$, $b_{sy} \in \mathbb{R}$. The "leaky rectifier" nonlinearity $\text{LReLU}(x) = \max(x, .01)$ was proposed in [20]; we found that it improved early convergence so we used it in all of our models. For all experiments, we set $D = 100$.

We used the squared error of the output y against the target value for each sequence as an objective. Parameters were optimized using "adam", a recently proposed stochastic optimization technique [21], with the optimization hyperparameters β_1 and β_2 set to the values suggested in [21] (.9 and .999 respectively). All weight matrices were initialized with entries drawn from a Gaussian distribution with mean zero and, for a matrix $W \in \mathbb{R}^{M\times N}$, a standard deviation of $1/\sqrt{N}$. All bias vectors were initialized with zeros. We trained on mini-batches of 100 sequences and computed the accuracy on a held-out test set of 1000 sequences every epoch, defined as 1000 parameter updates. We stopped training when either 100% accuracy was attained on the test set, or after 100 epochs. All networks were implemented using Lasagne [22], which is built on top of Theano [23, 24].

2.2 Fixed-Length Experiment

Traditionally, the sequence lengths tested in each task vary uniformly between $[T_0, 1.1T_0]$ for different values of T_0. As T_0 increases, the model must be able to handle longer term dependencies. The largest value of T_0 for which high accuracy was attained on a held-out test set varies across models, tasks and papers. In the original paper describing the long short-term memory (LSTM) architecture [3], a small LSTM network was shown to be able to solve the addition problem up to $T_0 = 1000$ and the multiplication problem up to $T_0 = 100$. [6] later showed that a "vanilla" (single-layer, densely-connected) recurrent network could solve both the addition and multiplication problems for $T_0 = 200$ when Hessian-Free optimization was used. The same model was used in [7], where Nesterov's Accelerated Gradient was used instead for optimization which allowed the model to solve the addition and multiplication problems for $T_0 = 80$. By initializing a similarly structured recurrent network's hidden-to-hidden weights to an identity matrix and using a rectifier activation function, [11] solved the addition problem up to $T_0 = 300$, which they claimed outperformed an LSTM model. More recently, [25] utilized a regularizer which encourages subsequent hidden states to have similar norms, allowing a vanilla recurrent network to achieve some success on the addition problem for $T_0 = 400$. Related work in [26] showed that enforcing an orthogonality constraint on the hidden-to-hidden weight matrix of a vanilla RNN allowed it to nearly solve the addition problem for $T_0 = 400$ and achieve some success for $T_0 = 750$. Finally, in [9] the addition and multiplication tasks were solved for $T_0 = 10000$ and $T_0 = 1000$ respectively by using a recurrent network with a very large and carefully initialized hidden-to-hidden weight matrix which was not optimized, which sidesteps the issue of vanishing and exploding gradients.

We therefore tested our proposed feed-forward attention models for $T_0 \in \{50, 100, 500, 1000, 5000, 10000\}$. The required number of epochs or accuracy after 100 epochs for each task, sequence length, and temporal integration method (adaptively weighted attention or unweighted mean) is shown in Table 1. For fair comparison, we report the best result achieved using any learning rate in $\{.0003, .001, .003, .01\}$. From these results, it's clear that the feed-forward attention model can quickly solve these long-term memory problems for longer sequence lengths than have been demonstrated with RNNs. Our model is also extremely efficient: Processing one epoch of 100,000 sequences with $T_0 = 5000$ took about two minutes using an

NVIDIA GTX 980 Ti GPU. In addition, there is a clear benefit to using the attention mechanism of Equation 1 instead of a simple unweighted average over time, which only incurs a marginal increase in the number of parameters (10,602 vs. 10,501, or less than 1%).

2.3 Variable-length Experiment

Because the range of sequence lengths $[T_0, 1.1T_0]$ is small compared to the range of T_0 values we evaluated, we further tested whether it was possible to train a single model which could cope with sequences with highly varying lengths. To our knowledge, such an experiment has not been conducted with RNNs. We trained models of the same architecture as used in the previous experiment on minibatches of sequences whose lengths were chosen uniformly at random between 50 and 10000 time steps. Using the attention mechanism of Equation 1, on held-out test sets of 1000 sequences, our model achieved 99.9% accuracy on the addition task and 99.4% on the multiplication task after training for 100 epochs. This suggests that a single feed-forward network with attention can simultaneously model both short-term and very long-term dependencies, with a marginal decrease in accuracy. Using an unweighted average over time, we were only able to achieve accuracies of 77.4% and 55.5% on the variable-length addition and multiplication tasks, respectively.

3 Discussion

A clear limitation of our proposed model is that it will fail on any task where temporal order matters because computing an average over time discards temporal information. For example, on the two-symbol temporal order task [3] where a sequence must be classified in terms of whether two symbols X and Y appear in the order X, X; Y, Y; X, Y; or Y, X, our model can differentiate between the X, X and Y, Y cases perfectly but cannot differentiate between the X, Y and Y, X cases at all. Nevertheless, we submit that for some real-world tasks involving sequential data, temporal order is substantially less important than being able to handle very long sequences. For example, in Joachims' seminal paper on text document categorization, he posits that "word stems work well as representation units and that their ordering in a document is of minor importance for many tasks" [27]. We also have shown in parallel work that our proposed feed-forward attention model can be used effectively for pruning large-scale (sub)sequence retrieval searches, even when the sequences are very long and high-dimensional [28]. Our results here further demonstrate that our model can be substantially more efficient than recurrent models. Further investigation on more real-world problems is warranted; for interested researchers, all of the code used in this experiment is available online.[1]

References

[1] Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[2] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv:1211.5063*, 2012.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv:1406.1078*, 2014.

[6] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1033–1040, 2011.

[7] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013.

[1] https://github.com/craffel/sequence-embedding/tree/master/toy_problems

[8] Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.

[9] Herbert Jaeger. Long short-term memory in echo state networks: Details of a simulation study. Technical Report 27, Jacobs University, February 2012.

[10] Tomas Mikolov, Armand Joulin, Sumit Chopra, et al. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*, 2014.

[11] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv:1504.00941*, 2015.

[12] Alex Graves. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[13] Kyunghyun Cho, Aaron C. Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoder-decoder networks. *arXiv:1507.01053*, 2015.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

[15] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

[16] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *arXiv:1503.08895*, 2015.

[17] Kyunghyun Cho. Introduction to neural machine translation with GPUs (part 3). `http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/`, 2015.

[18] Sander Dieleman. Recommending music on Spotify with deep learning. `http://benanne.github.io/2014/08/05/spotify-cnns.html`, 2014.

[19] Min Lin, Qiang Chen, and Shuicheng Yen. Network in network. *arXiv preprint arXiv:1312.4400*, 2014.

[20] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[21] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Sander Dieleman, Jan Schlüter, Colin Raffel, et al. Lasagne: First release. `https://github.com/Lasagne/Lasagne`, 2015.

[23] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, et al. Theano: new features and speed improvements. In *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*, 2012.

[24] James Bergstra, Olivier Breuleux, Frédéric Bastien, et al. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, 2010.

[25] David Krueger and Roland Memisevic. Regularizing RNNs by stabilizing activations. *arXiv preprint arXiv:1511.08400*, 2015.

[26] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *arXiv preprint arXiv:1511.06464*, 2015.

[27] Thorsten Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[28] Colin Raffel and Daniel P. W. Ellis. Pruning subsequence search with attention-based embedding. In *Proceedings of the 41st IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2016 (to appear).