CSE847 Project Intermediate Report

An analysis and verification of the efficacy of using Fast Weights with RNNs

Eric Alan Wayman
Department of Computer Science & Engineering
Michigan State University
waymane1@msu.edu

Adi Mathew
Department of Computer Science & Engineering
Michigan State University
mathewa6@msu.edu

ABSTRACT

This report describes our progress towards the goal of performing an analysis and reproduction of tests using the Fast Weights method (Ba et al. 2016a).

KEYWORDS

RNN, LSTM, Fast Weights, Memory

1 PROBLEM DESCRIPTION

Until recently, recurrent neural networks (RNNs), used for sequential processing, did not have a good way to share memory between sequences. The Fast Weights method introduced in the paper to be analyzed in this project (Ba et al. 2016a) addresses this limitation by providing the network with the capacity to store information about a given sequence during its duration to be used in the upcoming sequence. We will provide a full analysis and explanation of the methology, and replicate one of the empirical tests of the method, which compares its performance on an associative retrieval task to that of an iRNN and a long short-term memory network, or LSTM (Hochreiter and Schmidhuber 1997).

2 SURVEY OF PRIOR WORK

Recurrent neural networks (RNNs) are well-suited for learning from sequential data since weights are shared among different stages of the sequence (Goodfellow et al. 2016, p. 373). In particular, RNNs have been shown to perform well in tasks of speech-to-text conversion, creation of language models for both characters and words (Sutskever et al. 2011) and even frame by frame video analyses (Mnih et al. 2014). In RNNs, a given hidden state essentially acts as short-term memory for the previous state, determining the next hidden state together with the next input. One major issue in training RNNs with long input sequences is that the error gradients end up becoming very large or small (Schmidhuber 2015, p. 16) which implies that even if the network can be trained, the effect of an early hidden state on the current hidden state is practically non-existent. This problem was overcome by the introduction of the long short-term memory RNN (LSTM RNN), whose activation function has a constant derivative and thus does not explode or vanish (Schmidhuber 2015, p. 19). Unfortunately, the LSTM RNN's memory is still limited to an amount proportional to the number of hidden units in a sequence (Ba et al. 2016a, p. 1). Ba et al. propose the Fast Weights method to allow sequence-to-sequence memory in a recurrent network. We also note that Hopfield nets (MacKay 2002) implemented a similar storage rule (Ba et al. 2016a, p. 2) which we will review in our paper.

3 PRELIMINARY PLAN

Our term paper will first present the Fast Weights methodology and place it in the context of methods that led to its development. We will provide an extended description and derivation of the methodology for the purpose of verifying its properties. Our goal will be to also replicate Section 4.1 of the paper, which compares the Fast Weights' performance on an associative retrieval task with that of an Identity-RNN (iRNN) (Talathi and Vartak 2015), and an LSTM RNN (Ba et al. 2016a).

This project is intended to verify the foundational math and reasoning which justify the use of Fast Weights in a network. This will require us to retrace the work leading up to the introduction of Fast Weights. The initial stage of our project will be to perform a thorough proof and derivation of the equations for RNNs, and clearly explain the issues that led to the creation of LSTM RNNs. For instance, we will explain the "long-term memory issue" in RNNs beginning as follows. The expression of the hidden unit h_t at time t is:

$$h_t = g(\boldsymbol{W} \cdot x_t + \boldsymbol{U} \cdot h_{t-1} + b_h)$$

After t time steps, we get:

$$h_t = g(\boldsymbol{W} \cdot x_t + \boldsymbol{U} \cdot g(\cdots g(\boldsymbol{W} \cdot x_{t-T} + \boldsymbol{U} \cdot h_{t-T} + b_h) \cdots) + b_h)$$

Because of the T nested multiplications of h_{t-T} by $\boldsymbol{U}$, the effect of h_{t-T} on h_t is negligible (namely, the network does not have "long-term memory"). We will provide a full exposition of how this problem manifests itself when the network is trained.

The next stage of the project will involve explaining LSTM RNNs, their improvements over RNNs, and their limitations. We will then explain the mathematics of Fast Weights in

RNNs, as well as several methodologies used in their implementation in the paper being studied such as layer normalization (Ba et al. 2016b), grid search (Goodfellow et al. 2016), and the Adam optimizer (Kingma and Ba 2014).

Following that, we will implement the Fast Associative Memory Network in MATLAB, and reproduce the analysis of Section 4.1 (Ba et al. 2016a) of the paper to confirm the performance of Fast Weights as compared to the iRNN and the LSTM RNN.

4 MEMORY AND RECURRENT NEURAL NETWORKS

Paragraph about the neural perspective.

Paragraph about treating as signals, etc.

There are two major ways of incorporating the temporal element of sequential data in neural network learning. One is providing a memory structure that can present data from multiple time steps to a network that does not itself depend on the time variable (e.g. the time element is handled externally) (Haykin 2009, p. 672-673). The other is incorporating the time element directly inside the network via the use of feedback. Feedback occurs in a system when the output of an element of the system eventually affects the input to that same element (Haykin 2009, p. 18). There are two main types of feedback: local feedback and global feedback. Local feedback occurs when the output from an element directly feeds into that element's input, and global feedback occurs when the output eventually affects the input after passing through other elements first (Haykin 2009, p. 673).

Recurrent neural networks are networks containing at least one feedback loop of either type (Haykin 2009, p. 23). Feedback loops are inherently time-delay elements, where the output from an element is fed into a new element with a delay (in other words, transmission from one node to another is not instantaneous).

One major use of recurrent neural networks is to provide *associative memory*. In the storage phase, an associative memory is presented with *key patterns* and stores memorized patterns or *values* which are implicitly associated with their key patterns. In the recall phase, when presented with a distorted or incomplete version of a key pattern, the memory produces the associated value pattern (Haykin 2009, p. 38).

Let $s+l=N$, where s is the total number of neurons used for storage of patterns and l is the number of neurons that can be used for learning. One metric used when designing an associative memory is the ratio $q=\frac{s}{N}=\frac{s}{s+l}$. The total machine capacity (N) is limited to some value by resource constraints. Note that for a small q, the performance may be very good, though s is small relative to l: in other words, the network must use a large number of neurons l to recall very well only a few patterns. Therefore we seek to make q as large as possible while still achieving good performance, i.e. the network should be able to recall correctly many patterns by using as few neurons for learning as possible (Haykin 2009, p. 39).

There are two main types of associative memory, *autoassociative* and *heteroassociative memory*: in autoassociative memory, the memorized patterns are the same as the key patterns, whereas in heteroassociative memory, the memorized patterns are different (Haykin 2009, p. 38). Different network structures may be more suited for one task than the other.

4.1 A simple example of an associative memory model

Consider an associative memory which learns the single key pattern $\boldsymbol{f}$ and value $\boldsymbol{g}$ where both are vectors (this example is from (Anderson 1995, p. 163-165)). We let the system be the matrix

$$\boldsymbol{A}=\eta \boldsymbol{g}\boldsymbol{f}^T$$

The system performs perfectly:

$$\boldsymbol{g}'=\boldsymbol{A}\boldsymbol{f}=\eta \boldsymbol{g}\boldsymbol{f}^T\boldsymbol{f}\propto \boldsymbol{g}$$

since the $\boldsymbol{g}'$ that is recalled is proportional to the value $\boldsymbol{g}$ associated with the input $\boldsymbol{f}$.

Now consider a set of key patterns $\boldsymbol{f}_i$ and associated values $\boldsymbol{g}_i$ where all $\boldsymbol{f}_i$ are orthogonal (we write $\boldsymbol{f}_i \rightarrow \boldsymbol{g}_i$ to denote the associations). Letting

$$\boldsymbol{A}_i=\boldsymbol{g}_i\boldsymbol{f}_i^T, \qquad \boldsymbol{A}=\sum_i \boldsymbol{A}_i$$

We see that again $\boldsymbol{A}$ performs recall perfectly since for all j,

$$\begin{aligned}\boldsymbol{A}\boldsymbol{f}_j&=\sum_i \boldsymbol{A}_i\boldsymbol{f}_j=\sum_{k\neq j}\boldsymbol{A}_k\boldsymbol{f}_j+\boldsymbol{A}_j\boldsymbol{f}_j\\&=\sum_{k\neq j}\boldsymbol{g}_k\boldsymbol{f}_k^T\boldsymbol{f}_j+\boldsymbol{g}_j=\boldsymbol{g}_j\end{aligned}$$

The above example suggests that outer products can be useful in constructing associative memory models. Using outer products to create memory storage is referred to "the generalization of Hebb's postulate of learning" (Haykin 2009, p. 698) since weight updates in Hebbian learning are calculated with outer products, as in the following example for a one-layer network (Fyfe 2000, p. 39-40). For output $\boldsymbol{y}\in\mathbb{R}^m$ and input $\boldsymbol{x}\in\mathbb{R}^n$, if

$$y_i=\sum_j w_{ij}x_j, \qquad \Delta w_{ij}=\alpha x_j y_i$$

then

$$\Delta w_{ij} = \alpha x_j \sum_k w_{ik} x_k = \alpha \sum_k w_{ik} x_j x_k$$

Letting $\alpha = \Delta t$,

$$\frac{\Delta w_{ij}}{\Delta t} = \sum_k w_{ik} x_j x_k$$

so as $\Delta t \to 0$,

$$\frac{d}{dt}\boldsymbol{W}(t) = \boldsymbol{C}\boldsymbol{W}(t)$$

or, writing out the matrices fully,

$$\begin{bmatrix} \frac{dw_{11}}{dt} & \cdots & \frac{dw_{m1}}{dt} \\ \vdots & \vdots & \vdots \\ \frac{dw_{1n}}{dt} & \cdots & \frac{dw_{mn}}{dt} \end{bmatrix} = \begin{bmatrix} x_1x_1 & \cdots & x_nx_1 \\ \vdots & \cdots & \vdots \\ x_1x_n & \cdots & x_nx_n \end{bmatrix} \begin{bmatrix} w_{11} & \cdots & w_{m1} \\ \vdots & \cdots & \vdots \\ w_{1n} & \cdots & w_{mn} \end{bmatrix}$$

Note that the simple example of autoassociative memory would correspond to the weights matrix initially being the zero matrix and having only one update, after which the memory can recall perfectly. In general the linear system just described will not converge without some constraints being imposed (Fyfe 2000, p. 40). Since it is known that given orthogonal key patterns only one weight update is needed, the algorithm can be run without considering issues of convergence or stability of the general case. However, stability of such systems is essential for learning algorithms based in such systems to exist; we will consider this point below in more detail.

Since not all sets of key values are orthogonal, the above system is inadequate for most cases. Before moving to other models we provide an overview of stability issues necessary for the design and evaluation of recurrent networks.

5 STABILITY, CONTROLLABILITY AND OBSERVABILITY IN DYNAMIC SYSTEMS

A *dynamic system* is a system whose state changes with time (Haykin 2009, p. 675). Above it was noted that feedback is a way of introducing time lags into a dynamic system. Depending on the particulars of the system, feedback can lead a system to stability or cause it to diverge. We consider a simple example (Haykin 2009, p. 18-21) of a dynamic system with feedback. From this example we will see that if the operator mapping input to output is a weight matrix, and outputs are mapped to inputs through a linear additive function, the values of the weights will determine whether the system is stable or diverges.

Consider the system defined by

$$y_k(n) = wx'_j(n), \qquad x'_j(n) = x_j(n) + z^{-1}[y_k(n)]$$

where z^{-1} is the unit time-delay operator, so $z^{-1}[y_k(n)] = y_k(n-1)$. Combining the two equations gives

$$y_k(n) = w\left(x_j(n) + z^{-1}[y_k(n)]\right)$$

$$y_k(n)(1 - wz^{-1}) = wx_j(n)$$

$$y_k(n) = w(1 - wz^{-1})^{-1}x_j(n)$$

Since

$$\sum_{l=0}^{\infty} w^l z^{-l} = \sum_{l=0}^{\infty} \left(\frac{w}{z}\right)^l = \frac{1}{(1 - wz^{-1})} = (1 - wz^{-1})^{-1}$$

we write

$$y_k(n) = w\sum_{l=0}^{\infty} w^l z^{-l} x_j(n) = \sum_{l=0}^{\infty} w^{l+1} x_j(n-l)$$

where we have applied the unit-time delay operator to the x_j term.

We consider the case where $x_j(k)$ are sampled from, say, a Gaussian distribution whose positive mean is much smaller than the value of $x_j(0)$. We observe three cases:

(1) If $|w| < 1$, the effect of the signal will decay towards 0.
(2) If $w = 1$, the system will diverge with the trend of $y_k(n)$ being linear.
(3) If $w > 1$, the system will diverge with the trend of $y_k(n)$ being exponential.

5.1 The direct method of Lyanpunov and definitions of aspects of dynamic systems

We note that system just described is a difference equation that is *linear* (namely, the exponents of x_j are to the first power only). Recurrent neural networks are generally *nonlinear* (the exponents of x_j are to powers greater than one) (Strogatz 2000, p. 6). To determine whether or not a general dynamic system (linear or nonlinear) is stable we utilize the direct method of Lyapunov (Haykin 2009, p. 674). We present the method and important definitions here:

Definition 5.1. State variables: $x_1(t), \ldots, x_N(t)$. Independent variable: t. Order of the system: N. State vector (a column vector N-dimensions): $x(t)$.

A nonlinear dynamic system thus consists of the equations

$$\frac{d}{dt}x_j(t) = F_j\left(x_j(t)\right), \qquad j = 1, 2, \ldots, N$$

or

$$\frac{d}{dt}x(t) = F\left(x(t)\right)$$

where F is non-linear and vector valued, with each element depending on the corresponding element of $x(t)$. We assume that F only depends on t through x, in other words the system is *autonomous* (Haykin 2009, p. 675)). The above equation is called the *state-space* equation (ibid.). We wish to establish the existence and possibly uniqueness of solutions to the state-space equation.

F continuous is sufficient for the existence of a solution; for the uniqueness of the solution we require the *Lipschitz condition*, which is the following: for column vectors x, u in an open set $\mathcal{M}$ in a normal state space, there exists a constant K such that

$$\|F(x) - F(u)\|_2 \leq K \|x - u\|_2$$

For autonomous systems the Lipschitz condition guarantees both the existence of a solution and its uniqueness. We also have that "if all partial derivatives $\partial F_i/\partial x_j$ are finite everywhere, then the function $\boldsymbol{F}(\boldsymbol{x})$ satisfies the Lipschitz condition" (Haykin 2009, p. 677).

5.1.1 Stability around an equilibrium state. We say a constant vector $\overline{x} \in \mathcal{M}$ is a an *equilibrium state* of the dynamic system if $F(\overline{x}) = 0$ (where of course 0 is a column vector). We examine the linearization of the state-space equation in a neighborhood of $\overline{x}$:

$$x(t) = \overline{x} + \Delta x(t)$$

We consider the Taylor expansion of $F(x)$. We define the Jacobian as $J_x y = \frac{\partial y}{\partial x^T}$ where x and y are column vectors.

$$\begin{aligned}\frac{d}{dt}x(t) &= F(x(t)) \\ &= F(\overline{x}) + J_x F(\overline{x})(x - \overline{x}) + \cdots \\ &= J_x F(\overline{x})(x - \overline{x}) + \cdots \\ &\approx J_x F(\overline{x})(x - \overline{x}) = J_x F(\overline{x})\Delta x\end{aligned}$$

where $\Delta x = x - \overline{x}$.

Since

$$\frac{d}{dt}\Delta x = \frac{d}{dt}x - 0 = \frac{d}{dt}x$$

We have

$$\frac{d}{dt}\Delta x(t) \approx J_x F(\overline{x})\Delta x$$

If $J_x F(\overline{x})$ is invertible, we can solve for Δx around $\overline{x}$, and the eigenvalues of $J_x F(\overline{x})$ determine the behavior of Δx in this neighborhood.

5.1.2 Types of stability of equilibrium states. The above analysis is limited in the sense that it says nothing about whether or not the system ever enters the neighborhood of the equilibrium point in the first place. We would like to be able to say definitively under what circumstances an system will approach an equilibrium state. To do this, we have several definitions, i.e. types, of stability.

Definition 5.2. The equilibrium state is said to be *uniformly stable* if, for any positive constant ϵ, there exists another positive constant $\delta = \delta(\epsilon)$ such that the condition

$$\|x(0) - \overline{x}\|_2 < \delta$$

implies

$$\|x(t) - \overline{x}\|_2 < \epsilon$$

(Haykin 2009, p. 681)

In other words, for a uniformly stable equilibrium state, we can guarantee that the state of the system at any time t will be within an arbitrary distance of the equilibrium state provided that the starting state is within a certain distance from the equilibrium state.

Definition 5.3. The equilibrium state is said to be *convergent* if there exists a positive constant δ such that the condition

$$\|x(0) - \overline{x}\|_2 < \delta$$

implies

$$\lim_{t\to\infty} x(t) = \overline{x}$$

(Haykin 2009, p. 681)

In other words, as long as the initial state is within a certain distance from the equilibrium state, the state of the system will eventually approach the equilibrium state (Haykin 2009, p. 681).

Definition 5.4. The equilibrium state is *asymptotically stable* if it is both uniformly stable and convergent. (Wilson 2010)

Definition 5.5. The equilibrium state is *globally asymptotically stable* if it is both uniformly stable and all trajectories of the system converge to $\overline{x}$ as $t \to \infty$. (Wilson 2010)

Global asymptotic stability implies the system has only a single equilibrium state (Haykin 2009, p. 681).

5.1.3 Theorems regarding stability. Now we present theorems which give conditions under which stability of various types is achieved. These theorems are useful because they save us from having to solve for the solutions of the state-space equations. Rather, the theorems say that if we can find a certain function of the state variable that satisfies certain properties, then the equilibrium state is stable in a certain way.

6 THE HOPFIELD NETWORK AS AN EXAMPLE OF AN AUTOASSOCIATIVE MEMORY MODEL

We examine below the Hopfield network, a well-known model that is suited for the autoassociative memory task and which is an application of the additive model of the neuron. (Yegnanarayana 2004).

REFERENCES

J. A. Anderson. 1995. *An Introduction to Neural Networks.* MIT Press, Cambridge, MA.

Jimmy Ba, Geoffrey E. Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. 2016a. Using Fast Weights to Attend to the Recent Past. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain.* 4331–4339. http://papers.nips.cc/paper/6057-using-fast-weights-to-attend-to-the-recent-past

Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016b. Layer Normalization. *CoRR* abs/1607.06450 (2016). http://arxiv.org/abs/1607.06450

Colin Fyfe. 2000. Artificial Neural Networks and Information Theory. (2000). Edition 1.2.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Simon Haykin. 2009. *Neural Networks and Learning Machines (3rd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. DOI:http://dx.doi.org/10.1162/neco.1997.9.8.1735

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980

David J. C. MacKay. 2002. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, New York, NY, USA.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and others. 2014. *Recurrent models of visual attention.* 2204–2212 pages.

Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117. DOI:http://dx.doi.org/10.1016/j.neunet.2014.09.003

Steven H. Strogatz. 2000. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*. Westview Press.

Ilya Sutskever, James Martens, and Geoffrey E. Hinton. 2011. Generating Text with Recurrent Neural Networks. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011.* 1017–1024.

Sachin S. Talathi and Aniket Vartak. 2015. Improving performance of recurrent neural network with relu nonlinearity. *CoRR* abs/1511.03771 (2015). http://arxiv.org/abs/1511.03771

Bill Wilson. 2010. Neurodynamics Notes. (2010). https://www.cse.unsw.edu.au/~billw/cs9444/Neurodynamics-10s2-4up.pdf

B. Yegnanarayana. 2004. *Artificial Neural Networks.* Prentice-Hall of India Pvt.Ltd.

Table 1: *Project timeline*

Week	Dates	Task	Deliverable
Week 1	(2/6 - 2/13)	Background reading	
Week 2	(2/13 - 2/20)	Background reading	Proposal
Week 3	(2/20 - 2/27)	Data set construction	
Week 4	(2/27 - 3/6)	Background, foundational proofs	
Week 5	(3/6 - 3/13)	Partial implementation and preliminary run	
Week 6	(3/20 - 3/27)	Compose Intermediate Report	Intermediate Report
Week 7	(4/3 - 4/10)	Full implementation of networks	
Week 8	(4/10 - 4/17)	Complete empirical analysis	
Week 9	(4/17 - 4/24)	Compose Final Report	
Week 10	(4/24 - 5/1)	Finish Final Report and rehearse Presentation	Final Report Presentation