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 f and value g where both are column vectors (this example is from (Anderson 1995, p. 163-165)). We let the system be the matrix

$$A = \eta g f^T$$

The system performs perfectly:

$$g' = Af = \eta g f^T f \propto g$$

since the g' that is recalled is proportional to the value g associated with the input f.

Now consider a set of key patterns f_i and associated values g_i where all f_i are orthogonal (we write $f_i \to g_i$ to denote the associations). Letting

$$A_i = g_i f_i^T, \qquad A = \sum_i A_i$$

we see that again A performs recall perfectly since for all j,

$$\begin{aligned} Af_j &= \sum_i A_i f_j = \sum_{k \neq j} A_k f_j + A_j f_j \\ &= \sum_{k \neq j} g_k f_k^T f_j + \eta g_j \propto g_j \end{aligned}$$

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 $y \in \mathbb{R}^m$ and input $x \in \mathbb{R}^n$, 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 $\Delta t \to 0$,

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

or, writing out the matrices fully,

$$\begin{bmatrix} \frac{dw_{11}}{dt} & \cdots & \frac{dw_{m1}}{dt} \\ \vdots & \vdots & \vdots \\ \frac{dw_{1n}}{dt} & \cdots & \frac{dw_{mn}}{dt} \end{bmatrix} = \begin{bmatrix} x_1 x_1 & \cdots & x_n x_1 \\ \vdots & \cdots & \vdots \\ x_1 x_n & \cdots & x_n x_n \end{bmatrix} \begin{bmatrix} w_{11} & \cdots & w_{m1} \\ \vdots & \cdots & \vdots \\ w_{1n} & \cdots & w_{mn} \end{bmatrix}$$

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 OF 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

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

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

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

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

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

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 Stability for autonomous 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) or the x_j are not in polynomial form). We present some important definitions here regarding dynamic systems:

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

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 $x(t)$. If F only

depends on t through x, the system is *autonomous*, otherwise it is *nonautonomous* (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. This is easier to do with autonomous than nonautonomous systems (Beer 1995, p. 180).

Let be F autonomous. Then F continuous is sufficient for the existence of a solution (note that this is not true for nonautonomous systems). 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 $F(x)$ satisfies the Lipschitz condition" (Haykin 2009, p. 677). Once again we note that the above results do not hold generally for nonautonomous systems: similar results hold only in certain neighborhoods of equilibrium states (Rasmussen 2006, Definition A.2, p. 194).

5.1.1 Stability around an equilibrium state. The results in this subsection hold only for autonomous systems.

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 $\overline{x}$ (letting $\Delta x = x - \overline{x}$):

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

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.

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

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.

Note that the reason the above linearization is not possible for nonautonomous systems is that if $F = F(x(t), t)$, taking the Jacobian with respect to only x would ignore the direct effect of t on the system (i.e. the effect of t on F that does not go through x).

5.1.2 Types of stability of equilibrium states. Again, the results in this subsection hold only for autonomous systems.

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, applicable only to autonomous systems, 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. The theorems are part of the *direct method of Lyapunov* (Haykin 2009, p. 682), and the function $V(x)$ in the following theorems is called a Lyapunov function.

THEOREM 5.6. *The equilibrium state $\overline{x}$ is (Lyapunov) stable if, in a small neighborhood of $\overline{x}$ there exists a positive-definite function $V(x)$ such that its derivative with respect to time is negative semidefinite in that region.*

THEOREM 5.7. *The equilibrium state $\overline{x}$ is asymptotically stable if, in a small neighborhood of $\overline{x}$, there exists a positive-definite function $V(x)$ such that its derivative with respect to time is negative definite in that region (Haykin 2009, p. 682).*

In the Haykin book, the first theorem omits the word "Lyapunov" and does not define "stable": he means "Lyapunov stable." Note that in the definitions given, the initial time point is always taken to be $t_0 = 0$ so the distinction between Lyapunov stable and uniform stable is not made clear, and in any case, Lyapunov stability was not even defined. For the system to be Lyapunov stable, we have $\delta(\epsilon, t_0)$ where δ depends on t_0, whereas for the system to be uniform stable, the choice of δ is independent of t_0 (Yao 2008).

Letting $\mathcal{U}$ be the small neighborhood of $\overline{x}$, if $V(x)$ is a Lyapunov function, the equilibrium state $\overline{x}$ is Lyapunov stable if $\frac{d}{dt}V(x) \leq 0$ for $x \in \mathcal{U} - \overline{x}$ and is asymptotically stable if $\frac{d}{dt}V(x) < 0$ for $x \in \mathcal{U} - \overline{x}$. Since $\frac{d}{dt}V(x) \leq 0$ in a neighborhood, if we define a surface (called a *Lyapunov surface*) $V(x) = c, c > 0$, once a trajectory crosses the surface, the trajectory stays within the set of points defined by $\{x \in \mathbb{R}^N : V(x) \leq c\}$ (note the mistake in Haykin here). If $\frac{d}{dt}V(x) < 0$, observe the trajectory continues to move closer and closer to $\overline{x}$, and by the second theorem, it approaches $\overline{x}$ as $t \to \infty$. (Note there is another mistake in Haykin: the book says "we cannot be sure that the trajectory will actually converge onto $\overline{x}$ as $t \to \infty$" but this is for the case of $\frac{d}{dt}V(x) \leq 0$, not strict inequality. For confirmation see, for example, Christofides and El-Farra 2005, p. 18-19.)

5.2 Attractors

Although the above exposition regarding Lyapunov surfaces is useful for intuitive understanding, there are several issues (leaving alone the fact that we are still only speaking of autonomous functions). One issue is we may not even have a Lyapunov function for our system. Since the existence of a Lyapunov function is a sufficient but not necessary condition for stability (Haykin 2009, p. 683), it seems sensible to have a definition of a region of stability without explicit reference to Lyapunov functions. We define the *basin of attraction* of an equilibrium state in the following manner:

Definition 5.8. The *basin of attraction* of an equilibrium state $\overline{x}$ is the set of all x such that $x(t) \to \overline{x}$ when $t \to \infty$ (Costin 2010).

Note that this definition can be applied to nonautonomous systems as well. We also note that basins of attraction are usually defined with respect to *attractors* of which there are four types, one of which is the equilibrium state (called an *equilibrium point attractor* in this context (Beer 1995, p. 179)). We restrict our consideration to equilibrium point attractors for the time being, which allows us to use the above more restricted definition.

We also observe an explicit connection between the Lyapunov surface and this more general concept of basin of attraction. For the small neighborhood $\mathcal{U}$ of $\overline{x}$, where $V(x)$ is negative definite, it can be shown that the sets $\mathcal{U}_c = \{x \in \mathcal{U} : V(x) \leq c\}$ are in the basin of attraction of $\overline{x}$ (Novozhilov 2015).

5.3 Information persistence

A standard RNN has the mathematical form (Goodfellow et al. 2016, p. 381)

$$\begin{aligned} a(t) &= b + Wh(t-1) + Ux(t) \\ h(t) &= \tanh(a(t)) \\ o(t) &= c + Vh(t) \\ \widehat{y}(t) &= \text{softmax}(o(t)) \end{aligned}$$

We choose our dimensions as:

Variable	Dimensions
y	$k \times 1$
o	$k \times 1$
V	$k \times m$
h	$m \times 1$
a	$m \times 1$
W	$m \times m$

When discussing stability for such a network, we are referring to the stability of the hidden unit h. For a sequence task, the hidden unit at each point in time should predict the desired output, and should incorporate information from as many previous time steps as will affect the desired output.

We consider

$$h(t) = \tanh(a(t)) = \tanh(b + Wh(t-1)) = Mh(t-1)$$

where M is the map taking $h(t-1)$ to $h(t)$ for the system where $Ux(t)$ is not included (so the system is autonomous).

If $h(0)$ begins within a basin of attraction then by our above analysis it will approach an attractor over time. In practice, if $h(0)$ and W are initialized properly, we may be able to guarantee h will begin in the correct basin of attraction (assuming the attractors even exist in the first place).

We summarize the argument of Bengio et al. 1994 which is restated in Hochreiter et al. 2001. Assuming we are in the desired basin of attraction of a hyperbolic attractor (either an equilibrium point attractor or a periodic (limit cycle) attractor), if we consider only the autonomous system, the system will converge to the attractor. If the system is made nonautonomous by the inclusion of the additive term $Ux(t)$, as long as $Ux(t)$ is in a subset of the *reduced attracting set* of the attractor, the system will still converge to the attractor (Bengio et al. 1994, p. 160). However, if this condition is violated, the state of the hidden units may drift away from the attractor even if the state is still in the basin of attraction. Bengio, Simard, and Frasconi show that the reduced attracting set is defined as $\|J_x M\| < 1$, i.e. the matrix norm of the Jacobian matrix of the mapping is less than 1 (for brevity, we ignore their discussion of uncertainty here). In short, when $Ux(t)$ is not in the reduced attracting set, the effect of such inputs $x(t)$ may be to push the state into a different basin of attraction than that of our desired hidden state. This may happen when, for example, the data is "noisy," so that not all the information of $x(t)$ leads the model to converge to the desired attractor.

The authors draw our attention to a dilemma: in the region where $\|J_x M\| < 1$, the gradient of the current state $h(t)$ with respect to $h(\tau), \tau \ll t$ decays quite rapidly (Hochreiter et al. 2001). This is an issue because gradient-based training methods, such as *back-propagation through time* or BPTT, the update to the weight matrix W is adjusted in proportion to these gradients. So, ironically, when the system is in the desired reduced attracting set, we cannot properly train the network, and when the system is not in that set, the weight matrix may change in undesired ways. Formally (Hochreiter et al. 2001),

$$\frac{\partial E(t)}{\partial W} = \sum_{\tau \le t} \frac{\partial E(t)}{\partial y(\tau)} \frac{\partial y(\tau)}{\partial W} = \sum_{\tau \le t} \frac{\partial E(t)}{\partial y(t)} \frac{\partial y(t)}{\partial y(\tau)} \frac{\partial y(\tau)}{\partial W}$$

where for $s < t$,

$$\frac{\partial y(t)}{\partial y(s)} = \frac{\partial y(t)}{\partial y(t-1)} \frac{\partial y(t-1)}{\partial y(t-2)} \cdots \frac{\partial y(s+1)}{\partial y(s)}$$

If the norm of each of the factors on the right hand side of this equation is less than 1, then $\frac{\partial y(t)}{\partial y(s)}$ converges to zero at an exponentially increasing rate as $t-s$ increases. Therefore looking at the equation for $\frac{\partial E(t)}{\partial W}$, for a single term of the summation with $\tau \ll t$,

$$\left| \frac{\partial E(t)}{\partial y(\tau)} \frac{\partial y(\tau)}{\partial W} \right| \to 0$$

Why do we say "we cannot properly train the network" due to the above property of the model? Thinking independently of the given model structure for a moment, assume only the following: that we wish to miminize $E = \sum_s E(s)$, the sum of prediction errors of the sequence produced by the $y(0), \ldots, y(T)$, and that the value of any $y(t)$ is determined by the values of $y(0), \ldots, y(t-1)$. Assume that for the true model which we don't know, for a given $\tau \ll t$, $\frac{\partial y(t)}{\partial y(\tau)}$ is large, in other words the relationship between the state at time τ has a strong effect on the state at time t. When we specify a model of the sequence, we would like our training of our proposed model to reflect this strong "dependency" of the states at these two time points. Assume that making a particular change in $y(\tau)$ leads to a decrease in $E(t)$ that outweighs the sum of all increases in $E(s), s \neq t$. We would like the training of this model to be able to lead to this change.

Now assume the model we specify is the RNN model from above. In that model, making a change to $y(\tau)$ requires a particular update to W. However, the adjustment to the weight matrix corresponding to the dependency $\frac{\partial y(t)}{\partial y(\tau)}$ is now the product of gradients less than 1, which is essentially 0 for $\tau \ll t$. Therefore the model structure has forced the term that would allow the weights to update to reflect the desired change in $y(\tau)$ to have almost no effect on the weight update. The weights will never change to give the desired effect (we note in passing that such a change to $y(\tau)$ leading to a change in $y(t)$ corresponds to the trajectory changing to a different basin of attraction). This flaw in RNNs is known as the *vanishing gradient problem.*

6 TRAINING RNNS

6.1 A note on gradients when matrices are involved

Many of the often-seen versions of matrix derivatives have shortcomings when generalizing them to cases when both a vector and a matrix or a matrix and a matrix are involved (Magnus and Neudecker 1999, p. 193-197). We use the definition recommended by Magnus and Neudecker (1999), for which all properties of Jacobian matrices are properly preserved. In order to state it, we first define the vec operator:

Definition 6.1. Let A be an $m \times n$ matrix and a_i its j-th column. Then vec A is the $mn \times 1$ vector

$$\text{vec } A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

(Magnus and Neudecker 1999, p. 34)

The definition of the Jacobian is as follows:

Definition 6.2. Let F be a differentiable $m \times p$ real matrix function of an $n \times q$ matrix of real variables X. The Jacobian matrix of F at X is the $mp \times nq$ matrix

$$DF(X) = \frac{\partial \text{vec } F(X)}{\partial (\text{vec } X)^T}$$

6.2 Derivation of gradients in BPTT

For $a = Wh$,

$$J_W(a) = \frac{\partial \text{vec}(Wh)}{\partial (\text{vec } W)^T}$$

We calculate $J_W(a)$ as follows:

$$W = \begin{pmatrix} w_1 & \dots & w_m \end{pmatrix}$$

where w_i are the columns of W. Then

$$\begin{aligned} (\text{vec}W)^T &= \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix}^T = \begin{pmatrix} w_1^T \cdots w_m^T \end{pmatrix} \\ &= \begin{pmatrix} w_{11} \cdots w_{m1} \cdots w_{1m} \cdots w_{mm} \end{pmatrix} \end{aligned}$$

$$Wh = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \vdots & \vdots \\ w_{m1} & \cdots & w_{mm} \end{pmatrix} \begin{pmatrix} h_1 \\ \vdots \\ h_m \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^m w_{1i} h_i \\ \vdots \\ \sum_{i=1}^m w_{mi} h_i \end{pmatrix}$$

Therefore

$$J_W(a) = \begin{pmatrix} h_1 I & h_2 I \dots h_m I \end{pmatrix}$$

where we have written the above result in a compact way due to formatting limitations; when deriving the above result we wrote m rows and the first m columns out fully. We further "simplify" the result by noting

$$J_W(a) = \begin{pmatrix} h_1 I & h_2 I \dots h_m I \end{pmatrix} = h^T \otimes I$$

where $\otimes$ is the Kronecker product.

$$J_W E_t = (J_{\widehat{y_t}} E_t)(J_{o_t} \widehat{y_t})(J_{h_t} o_t)(J_{a_t} h_t)(J_W a_t)$$

$J_{o_t} \widehat{y_t}$ is the Jacobian of the softmax:

$$\widehat{y_t} = \begin{pmatrix} \frac{\exp(o_1(t))}{\sum_{i=1}^k \exp(o_i(t))} & \cdots & \frac{\exp(o_k(t))}{\sum_{i=1}^k \exp(o_i(t))} \end{pmatrix}^T$$

We note that

$$\begin{aligned} J_{o_i}(y_i) = \frac{\partial y_i}{\partial o_i} &= \frac{(\sum \exp(o_j)) \exp(o_i) - (\exp(o_i))^2}{(\sum \exp(o_j))^2} \\ &= \frac{\exp(o_i)}{\sum \exp(o_j)} - \left(\frac{\exp(o_i)}{\sum \exp(o_j)} \right)^2 \\ &= y_i(1 - y_i) \end{aligned}$$

For $l \neq i$,

$$\begin{aligned} J_{o_l}(y_i) &= \frac{(\sum \exp(o_j))(o_l) - \exp(o_i) \exp(o_l)}{(\sum \exp o_j)^2} \\ &= \frac{-\exp(o_i) \exp(o_j)}{(\sum \exp o_j)^2} \\ &= -y_i y_l \end{aligned}$$

Therefore

$$\begin{aligned} J_{o_t} \widehat{y_t} &= \begin{pmatrix} y_1(1-y_1) & -y_1 y_2 & \cdots & -y_1 y_k \\ -y_2 y_1 & y_2(1-y_2) & \cdots & -y_2 y_k \\ \cdots & \cdots & \cdots & \cdots \\ -y_k y_1 & \cdots & \cdots & y_k(1-y_k) \end{pmatrix} \\ &= \text{diag}(y_t) - y_t y_t^T \end{aligned}$$

where diag indicates a square matrix with the elements of y_t on the diagonal.

For $J_{h(t)}(o(t)) = J_{h(t)} Vh(t)$, we apply the definition of the Jacobian so

$$Vh(t) = \begin{pmatrix} \sum_{i=1}^k v_i h_i \end{pmatrix}$$

where v_i are the columns of V and h_i are the elements of $h(t)$. Then

$$J_{h_t}(o_t) = J_{h(t)} Vh(t) = \frac{\partial Vh(t)}{h(t)^T} = V$$

For $J_{a_t} h_t$, since $\frac{\partial \tanh(y(x))}{\partial x} = (1 - \tanh^2(x))$,

$$J_{a_t} h_t = \frac{\partial \tanh(h(t))}{\partial (a(t)^T)} = \text{diag}(1 - \tanh^2(a(t)))$$

Putting this all together and converting the Jacobian into a gradient we obtain

$$\begin{aligned} \nabla_W E_t &= (J_W E_t)^T \\ &= -\left(h(t)^T \otimes I \right)^T \text{diag}\left(1 - \tanh^2(a_t)\right) V^T \\ &\qquad (\text{diag}(y_t) - y_t y_t^T) \left(\frac{y_{tk}}{\widehat{y_{tk}}} \right) \\ &= -(h_t \otimes I) \text{diag}\left(1 - \tanh^2(a_t)\right) V^T \\ &\qquad (\text{diag}(y_t) - y_t y_t^T) \left(\frac{y_{tk}}{\widehat{y_{tk}}} \right) \end{aligned}$$

7 IRNN

Rectified linear units, or ReLUs, seem to have been introduced by Hahnloser et al. 2000. It has been shown that the use of the ReLU as the activation function for an RNN allows the learning of long-term dependencies given that the weight matrix is initialized as a scalar multiple of the identity matrix (Le et al. 2015, p. 2). The ReLU takes the form

$$f(x) = \max(0, x)$$

Ba et al. 2016a do not specify which cost function was used for training. Since a softmax function is used to produce the predicted values (Ba et al. 2016a, p. 5), the predicted values can be interpreted as probabilities and we can thus use the cross-entropy cost function for the multiclass classification problem (Bishop 2006),

$$p(y; U, W, V) = \prod_{\tau=1}^{t} \prod_{k=1}^{K} \widehat{y_{\tau k}}^{\, y_{\tau k}}$$

We minimize the negative logarithm of the cross entropy,

$$E = -\ln(p) = -\sum_{\tau=1}^{t} \sum_{k=1}^{K} \widehat{y_{\tau k}}^{\, y_{\tau k}}$$

8 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

Randall D. Beer. 1995. A Dynamical Systems Perspective on Agent-Environment Interaction. *Artif. Intell.* 72, 1-2 (1995), 173–215. DOI: http://dx.doi.org/10.1016/0004-3702(94)00005-L

Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks* 5, 2 (1994), 157–166. DOI:http://dx.doi.org/10.1109/72.279181

C.M. Bishop. 2006. *Pattern Recognition and Machine Learning.* Springer. https://books.google.com/books?id=kTNoQgAACAAJ

P.D. Christofides and N. El-Farra. 2005. *Control of Nonlinear and Hybrid Process Systems: Designs for Uncertainty, Constraints and Time-Delays.* Springer Berlin Heidelberg. https://books.google.com/books?id=P7QPCPwacGkC

Ovidiu Costin. 2010. Gradient and Hamiltonian systems. (2010). https://people.math.osu.edu/costin.9/820-10/p2.pdf

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.

Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* 405, 6789 (2000), 947–951.

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

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In *A Field Guide to Dynamical Recurrent Neural Networks*, Kremer and Kolen (Eds.). IEEE Press.

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

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. 2015. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *CoRR* abs/1504.00941 (2015). http://arxiv.org/abs/1504.00941

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

J.R. Magnus and H. Neudecker. 1999. *Matrix Differential Calculus with Applications in Statistics and Econometrics.* Wiley. https://books.google.com/books?id=0CXXdKKiIpQC

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

Artem Novozhilov. 2015. Lecuture notes for Math Methods of Bio Processes, Lecture 15: Limit sets, Lyapunov functions. (2015). "https://www.ndsu.edu/pubweb/~novozhil/Teaching/484Data/15.pdf"

Martin Rasmussen. 2006. *Attractivity and bifurcation for nonautonomous dynamical systems.* Ph.D. Dissertation. Uni Augsburg.

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. Yao. 2008. Lyapunov Stability - Stability of Equilibrium Points. (2008). https://engineering.purdue.edu/~byao/Research/Supplements/Lyapunov.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