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

CSE847 (Machine Learning) Project Final Report

Eric Alan Wayman[1] Adi Mathew[1]

[1]Department of Computer Science and Engineering
Michigan State University

April 27, 2017

RNN basics

Definition

A *standard RNN* is defined by the following equations:

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

where we choose to minimize the cross-entropy cost function since we are performing a classification task:

$$E = -\ln(p) = -\sum_{\tau=1}^{t}\sum_{j=1}^{k} y_{\tau k}\ln\left(\widehat{y_{\tau j}}\right) = \sum_{\tau=1}^{t} E_t$$

RNN basics

For the associative retrieval task, we wish to only look at the end-of-sequence y_t so the error function becomes

$$E = -\ln(p) = \sum_{j=1}^{k} y_{tk} \ln\left(\widehat{y_{tj}}\right) = E_t$$

RNN basics

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

Note that W and U when chosen with gradient-training procedures will choose W and U that influence h_t such that E_t over the entire batch is minimized (i.e. the “average error”) is small. Can we do better?

Associative memory basics

Consider an associative memory which learns the single key pattern f and value g where both are column vectors (Anderson, *An Introduction to Neural Networks*, 1995). 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.

Associative memory basics

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,

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

Associative memory basics

Using outer products to create memory storage is referred to "the generalization of Hebb's postulate of learning" (Haykin, *Neural Networks and Learning Machines* 2009) since weight updates in Hebbian learning are calculated with outer products.

Note that generally, not all sets of key patterns would be orthogonal; we discuss the implications of this when we consider the associative memory structure from the Fast Weights paper.

Dynamic systems approach

The RNN is a nonlinear dynamic system defined by difference equations. We focus on the hidden states h_t. Since h_t changes not only due to past values of h_τ but also new inputs coming in at every time step, x_t. Thus it is a *nonautonomous* system and stability results for autonomous dynamic systems do not directly apply.

However the concepts of *attractors* and *basins of attraction* are useful to understand the behavior of the network. For the most simple type of attractor, the equilibrium point attractor, the *basin of attraction* of an equilibrium state $\overline{x}$ is the set of all x such that $x(t) \to \overline{x}$ when $t \to \infty$. We note that an attractor can consist of multiple points.

Dynamic systems approach

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.

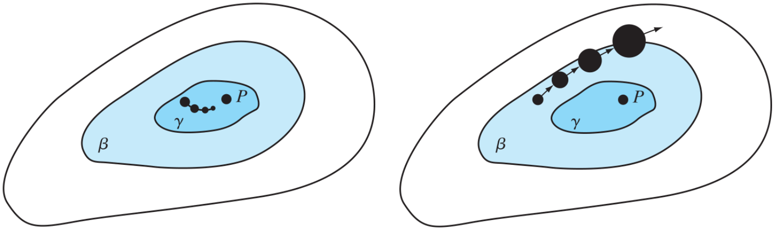
We consider

$$h(t) = \text{activ}(a(t)) = \text{activ}(b + Wh(t-1)) = Mh(t-1)$$

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).

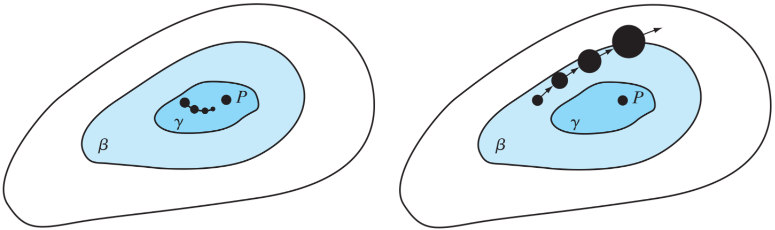
Vanishing gradients in RNNs

We summarize the argument of Bengio et al. 1994 which is restated in Hochreiter 2001. They show that even a nonautonomous system like the RNN will converge to the attractor as long as it begins in an area called the reduced attracting set γ. However, even if the state of the hidden units are in the basin of attraction β, if they are not in γ, the state may drift away from the attractor (figures from Haykin 2009).



Vanishing gradients in RNNs

The authors show that unfortunately, in the region γ, the gradient of the current state $h(t)$ with respect to $h(\tau), \tau \ll t$ decays quite rapidly. This is an issue because gradient-based training methods, 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.



Backpropagation in RNNs

$$\begin{aligned} J_W E_t &= (J_{a_t} E_t)(J_W a_t) \\ &= (J_{a_t} E_t)(h_t + J_W h_{t-1}) \\ &= (J_{a_t} E_t) h_t + (J_{a_t} E_t)(J_w h_{t-1}) \end{aligned}$$

The second term cannot be reformulated to include $J_{a_{t-1}} E_t$ since the chain of derivatives has been "broken" (i.e. $J_{h_{t-1}} a_t$, the "connecting term," does not appear).

A solution to this is to use the notation W_t to represent the W appearing at each time step (Goodfellow et al. 2016, p. 386). Using this notation, $J_W E_t$ becomes

$$J_W E_t = \sum_{\tau=2}^{t} (J_{a_\tau} E_t)(J_{W_\tau} a_\tau)$$

Backpropagation in RNNs

$$J_{a_t} E_t = \delta_t^t$$

$$J_{a_{t-1}} E_t = (J_{a_t} E_t)(J_{h_{t-1}} a_t)(J_{a_{t-1}} h_{t-1}) = \delta_t^t (J_{h_{t-1}} a_t)(J_{a_{t-1}} h_{t-1}) = \delta_{t-1}^t$$

$$\begin{aligned} J_{a_{t-2}} E_t &= (J_{a_{t-1}} E_t)(J_{h_{t-2}} a_{t-1})(J_{a_{t-2}} h_{t-2}) \\ &= \delta_{t-1}^t (J_{h_{t-2}} a_{t-1})(J_{a_{t-2}} h_{t-2}) = \delta_{t-2}^t \end{aligned}$$

So once δ_{t-1}^t has been calculated, the calculation of the Jacobian with respect to the preceding time step requires the calculating only three Jacobians, rather an ever-expanding product of Jacobians.

Fast weights augmentation to RNNs

The Fast Weights augmented RNN has h_t of the form:

$$h_t^{s+1} = \text{activ}\left(LN(b + Wh_{t-1} + Ux_t) + A_t h_{t-1}^s \right)$$

where the superscript s on h_{t-1} is not a power but an index indicating the number of iterations for the inner loop, and where LN indicates the layer normalization procedure. The A_t term is the fast weights matrix.

Fast weights augmentation to RNNs

$A_t = \lambda A_{t-1} + \eta h_t h_t^T$

$$A_{t-1} h_t = \eta \sum_{\tau=1}^{t-1} \lambda^{(t-1)-\tau} h_\tau \left(h_\tau^T h_t \right)$$

This result is the sum of all past h_τ, $\tau < t$, where each h_τ is weighted by two quantities:

- The further in the past is τ, the smaller the magnitude of the resulting vector ($\lambda^{(t-1)-\tau}$ with $\lambda < 1$)
- The less similar are h_t and h_τ, the smaller the positive magnitude of the resulting vector (the multiple of h_τ ranges from 1 down to -1, when the vectors are pointed in exactly the opposite direction).

So $A_{t-1} h_t$ represents the weighted sum of all h_τ, $1 \le \tau \le t-1$ with more recent vectors and vectors more similar to h_t being given higher (more largely positive) weight.

Fast weights augmentation to RNNs

What the above loop does is to influence h_t in a direction towards the vector that results from this sum. Note that without this term, h_t would be influenced by h_{t-1} only through W, and influenced by h_τ, $\tau < t-1$ only indirectly through W. Thus the effect of the addition of this term causes a h_t to be influenced by all $h_\tau, \tau < t$ directly and in proportion to how recent and how similar each h_τ is to h_t.

Associative retrieval task results