VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

some kind of rnn/tensor mess

Paul Francis Cunninghame Mathews

Supervisors: Marcus Frean and David Balduzzi

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

A short description of the project goes here.

Acknowledgments

Any acknowledgments should go in here, between the title page and the table of contents. The acknowledgments do not form a proper chapter, and so don't get a number or appear in the table of contents.

Contents

Figures

Chapter 1

Introduction

This chapter gives an introduction to the project report.

In Chapter **??** we explain how to use this document, and the `vuwproject` style. In Chapter **??** we say some things about LaTeX, and in Chapter 6 we give our conclusions.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 Feed-Forward Neural Networks

This is to be pretty brief. Cover useful things – what they look like, gradient descent (in brief) and outline some results on the expressive power.

2.1.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) of the form considered here generalise feed-forward networks to address problems in which we wish to map a *sequence* of inputs $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_T)$ to a sequence of outputs $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots \mathbf{y}_T)$. They have been applied successful to a wide range of tasks which can be framed in this way including statistical language modelling [10] (including machine translation [3]), speech recognition [6], polyphonic music modelling [2], music classification [4], image generation [7] and more.

Original Formulation

An RNN is able to maintain context over a sequence by transferring its hidden state from one time-step to the next. We refer to the vector of states at time t as $\mathbf{h}_t$.

The classic RNN (often termed "vanilla") originally proposed in [5] computes its hidden states with the following recurrence:

$$\mathbf{h}_t = f(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \tag{2.1}$$

where $f(\cdot)$ is some elementwise non-linearity, often the hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

Equation (2.1) bears a striking resemblance to the building block of a feed-forward network. The key difference is the (square) matrix $\mathbf{W}$ which contains weights controlling how the previous state affects the computation of the new activations.

Training

We can train this (or any of the variants we will see subsequently) using back-propagation. Often termed "Back Propagation Through Time" [12] which requires using the chain rule to determine the gradients of the loss with respect to the network parameters in the same manner as for feedforward networks.

To understand what is required to perform this, consider a loss function for the whole sequence of the form

$$\mathcal{L}(\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_T, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T) = \sum_{i=1}^{T} \mathcal{L}_i(\hat{\mathbf{y}}_i, \mathbf{y}_i),$$

which is a sum of the loss accrued at each time-step. This captures all common cases including sequence classification or regression, as the $\mathcal{L}_i$ may simply return 0 for all but the last time-step. To find gradients of the loss with respect the parameters which generate the hidden states, we must first find the gradient of the loss with respect to the hidden states themselves. Choosing a hidden state i somewhere in the sequence we have:

$$\nabla_{\mathbf{h}_i} \mathcal{L} = \sum_{j=i}^{t} \nabla_{\mathbf{h}_i} \mathcal{L}_j$$

from the definition of the loss and the fact that a hidden state may affect all future losses. To determine each $\nabla_{\mathbf{h_i}} \mathcal{L}_j$ (noting $j \geq i$), we apply the chain rule, to back-propagate the error from time j to time i. This is the step from which the algorithm derives its name, and simply requires multiplying through adjacent timesteps. Let $\mathbf{z}_k = \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}$ be the pre-activation of the hidden states. Then

$$\begin{aligned}
\nabla_{\mathbf{h}_i} \mathcal{L}_j &= (\nabla_{\mathbf{h}_j})^{\top} \mathcal{L}_j \left(\prod_{k=i+1}^{j} \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}} \right) \\
&= (\nabla_{\mathbf{h}_j} \mathcal{L}_j)^{\top} \left(\prod_{k=i+1}^{j} \nabla_{\mathbf{z}_k} f \cdot \mathbf{W} \right). \qquad (2.2)
\end{aligned}$$

This has two key components: $\nabla_{\mathbf{h}_j} \mathcal{L}_j$ quantifies the degree to which the hidden states at time j affect the loss (computing this will most likely require further back-propagation through one or more output layers) while the second term in equation (2.2) measures how much the hidden state at time i affects the hidden state at time j.

We can now derive an update rule for the parameters by observing

$$\begin{aligned}
\nabla_{\mathbf{W}} \mathcal{L} &= \sum_{i=1}^{T} \nabla_{\mathbf{W}} \mathcal{L}_i \\
&= \sum_{i=1}^{T} \sum_{j=1}^{i} \nabla_{\mathbf{h}_j} \mathcal{L}_i \nabla_{\mathbf{W}} \mathbf{h}_j
\end{aligned}$$

shapes, get them right

and applying the above. For the input matrix and the bias the process is the same.

Issues

Equation (2.2) reveals a key pathology of the vanilla RNN – vanishing gradients. This occurs when the gradient of the loss vanishes to a negligibly small value as we propagate it backward in time, leading to a negligible update to the weights. $(\nabla_{\mathbf{h}_j} \mathcal{L}_j)^{\top}$ (a vector) is multiplied by a long product of matrices, alternating between $\nabla_{\mathbf{z}_k} f$ and $\mathbf{W}$. If we assume for illustrative purposes that $f(\cdot)$ is the identity function (so we have a linear network), then the loss vector is multiplied by $\mathbf{W}$ taken to the $(j-i)$th power. If the largest eigenvalue of $\mathbf{W}$ is large, then this will cause the gradient to eventually explode. If the largest eignevalue is small, then the gradient will vanish. This issue was first presented in 1994 [1], while a thorough treatment including necessary conditions for vanishing and the complementary exploding problem, see [11].

In the non-linear case, this remains a serious issue. While exploding gradients are often mitigated by using a *saturating* non-linearity so that the gradient tends to zero as the hidden states grow, this only exacerbates the vanishing problem.

A second issue when training RNNs can be illustrated by viewing them as iterated non-linear dynamical systems and thus susceptible to the "butterfly effect": seemingly negligible changes in initial conditions can lead to catastrophic changes after a number of iterations [9]. In RNNs this manifests as near-discontinuity of the loss surface [11] as a change (for example, to a weight during back-propagation) which may even reduce the loss for a short period can cause instabilities further on which lead to steep increases in loss. This problem is not as well studied as vanishing gradients although some partial solutions exist such as clipping the norm of gradients [11] or using a regulariser to encourage gradual changes in hidden state [8].

Alternate Architectures

To address these fundamental problems a number of alternate architectures have been proposed. Here we will outline two popular variants: the Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU). Both of these belong to a class of *gated* RNNs, which have a completely novel method of computing a new state.

The LSTM was proposed to alleviate the vanishing gradient problem. It computes a new state in a fundamentally different manner

2.2 Related Work

2.2.1 Long Time Dependencies

Summary of approaches to fix vanishing gradients.

2.2.2 Memory

Summary of approaches to augmenting RNNs with extra memory, or other approaches to better use memory.

2.2.3 Tensors in Neural Networks

Including gated networks, MRNN and so on.

Chapter 3

Tensors

This chapter discusses some necessary/useful multi-linear algebra which we use later.

3.1 Definitions

3.2 Bilinear Products

3.3 Tensor Decompositions

3.3.1 CANDECOMP/PARAFAC

3.3.2 Tensor Train, Tucker

3.4 Learning decompositions by gradient descent

Multiplicative dynamics = instability?

Chapter 4

Proposed Architectures

4.1 Incorporating tensors for expressivity

4.2 Gates and Long Time Dependencies

4.3 Proposed RNNs

Chapter 5

RNN Experiments (better title plz)

5.1 Synthetic Tasks

Pathological, exercise specific features of the architecture.

5.1.1 Addition

5.1.2 Variable Binding

5.1.3 MNIST

is really dumb

5.2 Real-world Data

Mostly testing rank as regulariser

5.2.1 Polyphonic Music

5.2.2 PTB

5.2.3 War and Peace

Chapter 6

Conclusions

The conclusions are presented in this Chapter.

Bibliography

[1] BENGIO, Y. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166.

[2] BOULANGER-LEWANDOWSKI, N., VINCENT, P., AND BENGIO, Y. Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. *Proceedings of the 29th International Conference on Machine Learning (ICML-12)* Cd (June 2012), 1159–1166. arXiv: `1206.6392`.

[3] CHO, K., ET AL. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (June 2014), 1724–1734. arXiv: `1406.1078`.

[4] CHOI, K., ET AL. Convolutional Recurrent Neural Networks for Music Classification. *arXiv preprint* (Sept. 2016). arXiv: `1609.04243`.

[5] ELMAN, J. l. Finding Structure in Time. *COGNITIVE SCIENCE* 14 (1990), 179–211.

[6] GRAVES, A., ET AL. Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks. *Proceedings of the 23rd international conference on Machine Learning* (2006), 369–376.

[7] GREGOR, K., ET AL. DRAW: A Recurrent Neural Network For Image Generation. *Icml-2015* (Feb. 2015), 1462–1471. arXiv: `1502.04623`.

[8] KRUEGER, D., AND MEMISEVIC, R. Regularizing RNNs by Stabilizing Activations. *International Conference On Learning Representations* (Nov. 2016), 1–8. arXiv: `1511.08400`.

[9] LORENZ, E. N. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences* 20, 2 (Mar. 1963), 130–141.

[10] MIKOLOV, T. Statistical Language Models Based on Neural Networks. PhD thesis. 2012, 1–129. arXiv: `1312.3005`.

[11] PASCANU, R., MIKOLOV, T., AND BENGIO, Y. On the difficulty of training recurrent neural networks. *Proceedings of The 30th International Conference on Machine Learning* 2 (2012), 1310–1318. arXiv: `arXiv:1211.5063v2`.

[12] WERBOS, P. J. Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE* 78, 10 (1990), 1550–1560.