Chapter 1

Recurrent Neural Networks

1.1 Introduction

A **recurrent neural network (RNN)** [1], e.g. Fig. 1.1, is a class of artificial neural network for modelling time series. The structure of the network is similar to that of a standard feedforward neural networks, with the distinction that it allows connections among hidden units associated with a time delay. Through these connections the model can retain information about the past, enabling it to discover temporal correlations between events that are far away from each other in the data.

While in principle the recurrent network is a simple and powerful model, in practice, it is hard to train properly. Among the main reasons why this model is so unwieldy are the *vanish gradient* and *exploding gradient* problems described in [2]. In this report, we will analyse these issues thoroughly and summarise those relevant solutions.

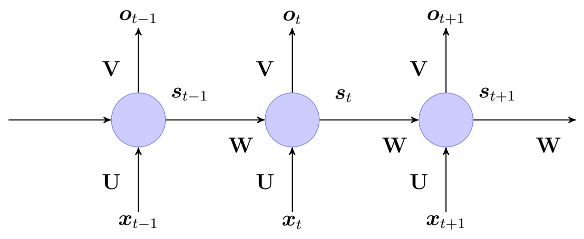


Figure 1.1: Schematic of an unrolled recurrent neural network. The recurrent connections in the hidden layer allow information to persist from one input to another.

1.2 Problem Formulation

We will make use of the following forward propagation formula[1] for a recurrent neural network with input $\boldsymbol{x}_t$ and state $\boldsymbol{s}_t$ at time step t:

$$\text{net}_t = \mathbf{U}\boldsymbol{x}_t + \mathbf{W}\boldsymbol{s}_{t-1} \tag{1.1}$$

$$\boldsymbol{s}_{t-1} = f(\text{net}_{t-1}) \tag{1.2}$$

where $\mathbf{U}$ and $\mathbf{W}$ are the weight matrices connecting the input $\boldsymbol{x}_t$ and the hidden state $\boldsymbol{s}_t$ at time step t, respectively. Also, the neuron output at time step t is represented as net_t.

1.2.1 Backpropagation Through Time (BPTT)

Element-wise calculus formulation

Let the jth element of the state vector $\boldsymbol{s}_t$ to be denoted as s^t_j, let the jth element of the input vector $\boldsymbol{x}_t$ be x^t_j, and similarly the ith neuron output is net^t_i. Then the above equation can be written equivalently as

$$\text{net}^t_i = u_{i\hat{j}} x^t_{\hat{j}} + w_{i\hat{j}} s^{t-1}_{\hat{j}} \tag{1.3}$$

$$s^{t-1}_j = f(\text{net}^{t-1}_j) \tag{1.4}$$

where the summation $\sum$ is omitted and by convention it is applied to the hatted indexes.

Let $\delta^t_i \equiv \frac{\partial \mathcal{E}_n}{\partial \text{net}^t_i}$, then the backpropagation through time (BPTT) update formula is derived as the following

$$\begin{aligned}\delta^t_i &= \frac{\partial \mathcal{E}_n}{\partial \text{net}^{t+1}_{\hat{j}}} \frac{\partial \text{net}^{t+1}_{\hat{j}}}{\partial \text{net}^t_i} \\ &= \delta^{t+1}_{\hat{j}} \frac{\partial \text{net}^{t+1}_{\hat{j}}}{\partial s^t_i} \frac{\partial s^t_i}{\partial \text{net}^t_i} \\ &= f^{'}(\text{net}^t_i)(w_{\hat{j}i}\delta^{t+1}_{\hat{j}}), \end{aligned} \tag{1.5}$$

or in matrix form as

$$\boldsymbol{\delta}_t = \text{diag}(f'(\text{net}^t))\mathbf{W}^\top \boldsymbol{\delta}^{t+1} \tag{1.6}$$

$$= (\prod_{l=t}^{T-1} \text{diag}(f'(\text{net}^l))\mathbf{W}^\top)\boldsymbol{\delta}_T \tag{1.7}$$

[1]For any model respecting Eqs. (1.1, 1.2), it is widely known as the following two recurrent formulations: $\boldsymbol{s}_t = f(\mathbf{U}\boldsymbol{x}_t + \mathbf{W}\boldsymbol{s}_{t-1})$ and $\text{net}_t = \mathbf{U}\boldsymbol{x}_t + \mathbf{W}f(\text{net}_{t-1})$.

Now we need to calculate the gradient with respect to each weight entry. Note that

$$\begin{aligned}\frac{\partial \mathcal{E}_n}{\partial w_{ij}} &= \frac{\partial \mathcal{E}_n}{\partial \mathrm{net}_i^{\hat{t}}} \frac{\partial \mathrm{net}_i^{\hat{t}}}{\partial w_{ij}} && \text{(summation is over } t\text{)} \\ &= \delta_i^{\hat{t}} s_j^{\hat{t}-1} && (1.8)\end{aligned}$$

From this equation, we can construct the gradient $\nabla_{\mathbf{W}} \mathcal{E}_n$ that are the sum of the gradients $\nabla_{\mathbf{W}_t} \mathcal{E}_n$ at all time step t.

Matrix calculus formulation

In this section, we provide a tensor calculus formulation [3] for the BPTT method. We summarise some of the tensor notations as follows.

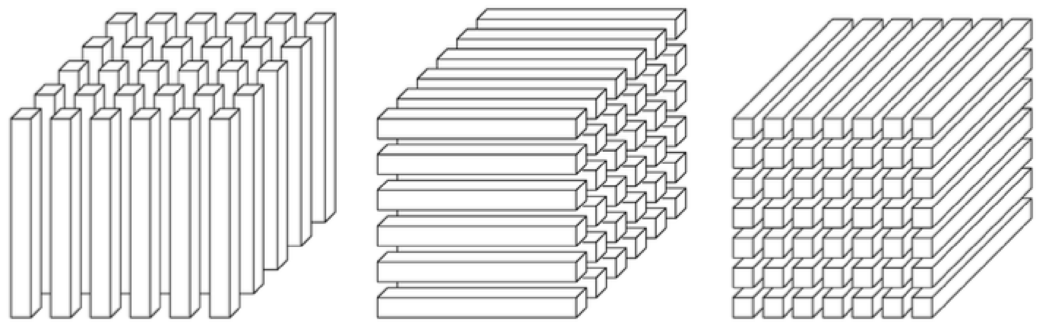
The number of dimensions of a tensor is called as its *order*, it is also known as ways or modes. Higher-order tensors (order three or higher) are denoted by boldface Euler script letters, e.g., $\boldsymbol{\mathcal{X}}$. And the element (i, j, k) of a third-order tensor $\boldsymbol{\mathcal{X}}$ is denoted by x_{ijk}. Indices typically range from 1 to their capital version, e.g., $i = 1, \ldots, I$. The nth element in a sequence is denoted by a superscript in parentheses, e.g., $\mathbf{A}^{(n)}$ denotes nth matrix in a sequence.

Fibers are the higher-order analogue of matrix rows and columns. A fiber is defined by fixing every index but one. A matrix column is a mode-1 fiber and a matrix row is a mode-2 fiber. Third-order tensors have column, row, and tube fibers, denoted by $\boldsymbol{x}_{:ik}, \boldsymbol{x}_{i:k}$ and $\boldsymbol{x_{ij:}}$, respectively; see Figure 1.2(a). When extracted from the tensor, fibers are always assumed to be oriented as column vectors.

Slices are two-dimensional sections of a tensor, defined by fixing all but two indices. Figure 1.2(b) shows the horizontal, lateral, and frontal slides of a third-order tensor $\boldsymbol{\mathcal{X}}$, denoted by $\mathbf{X}_{i::}, \mathbf{X}_{:j:}$, and $\mathbf{X}_{::k}$, respectively.

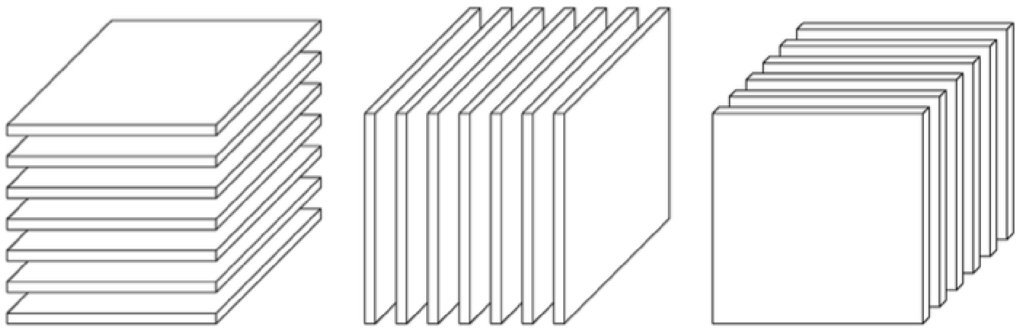
Now we return to our discussion of BPTT method. Let the error signal $\boldsymbol{\delta}_t$ at time step t to be defined as $\frac{\partial \mathcal{E}_n}{\partial \mathrm{net}_t}$, we then have

$$\begin{aligned}\boldsymbol{\delta}_t &= \frac{\partial \mathcal{E}_n}{\partial \mathrm{net}_{t+1}} \frac{\partial \mathrm{net}_{t+1}}{\partial \mathrm{net}_t} && \text{(numerator layout)} \\ &= \boldsymbol{\delta}_{t+1} \frac{\partial \mathrm{net}_{t+1}}{\partial \boldsymbol{s}_t} \frac{\partial \boldsymbol{s}_t}{\partial \mathrm{net}_t} \\ &= \boldsymbol{\delta}_{t+1} \mathbf{W} \mathrm{diag}(f'(\mathrm{net}_t)) && (1.9) \\ &= \boldsymbol{\delta}_T (\prod_{l=t}^{T-1} \mathbf{W} \mathrm{diag}(f'(\mathrm{net}_t))) && (1.10)\end{aligned}$$



(a) Mode-1 (column) fibers: $\mathbf{x}_{:jk}$ (b) Mode-2 (row) fibers: $\mathbf{x}_{i:k}$ (c) Mode-3 (tube) fibers: $\mathbf{x}_{ij:}$

(a) Tensor fibers



(a) Horizontal slices: $\mathbf{X}_{i::}$ (b) Lateral slices: $\mathbf{X}_{:j:}$ (c) Frontal slices: $\mathbf{X}_{::k}$ (or $\mathbf{X}_k$)

(b) Tensor slices

Figure 1.2: Fibers and slices of a 3rd-order tensor

The update for the weight matrix $\mathbf{W}$ is calculated as

$$\begin{aligned}
\frac{\partial \mathcal{E}_n}{\partial \mathbf{W}} &= \frac{\partial \mathcal{E}_n}{\partial \text{net}_T}\frac{\partial \text{net}_T}{\partial \mathbf{W}} \\
&= \frac{\partial \mathcal{E}_n}{\partial \text{net}_T}\frac{\partial (\mathbf{U}\boldsymbol{x}_T + \mathbf{W}f(\text{net}_{T-1}))}{\partial \mathbf{W}} \\
&= \boldsymbol{\delta}_T^\top \left[\frac{\partial \mathbf{W}}{\partial \mathbf{W}}\boldsymbol{s}_{T-1} + \frac{\partial \text{net}_T}{\partial \text{net}_{T-1}}\frac{\partial \text{net}_{T-1}}{\partial \mathbf{W}}\right] \\
&= \boldsymbol{\delta}_T^\top \frac{\partial \mathbf{W}}{\partial \mathbf{W}}\boldsymbol{s}_{T-1} + \frac{\partial \mathcal{E}_n}{\partial \text{net}_{T-1}}\frac{\partial \text{net}_{T-1}}{\partial \mathbf{W}} \\
&= \sum_{t=1}^{T} \boldsymbol{\delta}_t^\top \frac{\partial \mathbf{W}}{\partial \mathbf{W}}\boldsymbol{s}_{t-1}. \qquad (1.11)
\end{aligned}$$

Similarly, we have

$$\begin{aligned}
\frac{\partial \mathcal{E}_n}{\partial \mathbf{U}} &= \frac{\partial \mathcal{E}_n}{\partial \text{net}_T}\frac{\partial \text{net}_T}{\partial \mathbf{U}} \\
&= \frac{\partial \mathcal{E}_n}{\partial \text{net}_T}\frac{\partial (\mathbf{U}\boldsymbol{x}_T + \mathbf{W}f(\text{net}_{T-1}))}{\partial \mathbf{U}} \\
&= \boldsymbol{\delta}_T^\top \left[\frac{\partial \mathbf{U}}{\partial \mathbf{U}}\boldsymbol{x}_T + \frac{\partial \text{net}_T}{\partial \text{net}_{T-1}}\frac{\partial \text{net}_{T-1}}{\partial \mathbf{U}}\right] \\
&= \boldsymbol{\delta}_T^\top \frac{\partial \mathbf{U}}{\partial \mathbf{U}}\boldsymbol{x}_T + \frac{\partial \mathcal{E}_n}{\partial \text{net}_{T-1}}\frac{\partial \text{net}_{T-1}}{\partial \mathbf{U}} \\
&= \sum_{t=1}^{T} \boldsymbol{\delta}_t^\top \frac{\partial \mathbf{U}}{\partial \mathbf{U}}\boldsymbol{x}_t. \qquad (1.12)
\end{aligned}$$

Note that $\frac{\partial \mathbf{W}}{\partial \mathbf{W}}$ is a four way tensor. We denote this tensor as $\mathcal{Y}$, then its (i,j,k,l)th element is computed as $y_{ijkl} = \frac{\partial w_{ij}}{\partial w_{lk}}$. The generalised formula $\boldsymbol{a}^\top \mathcal{Y}\boldsymbol{b}$ is thus a matrix of J rows and K columns. This jkth element is calculated as

$$\begin{aligned}
a_{\hat{i}} y_{\hat{i}jk\hat{l}} b_{\hat{l}} &= a_{\hat{i}} \frac{\partial w_{\hat{i}j}}{\partial w_{\hat{l}k}} b_{\hat{l}} \\
&= a_{\hat{i}} (\frac{\partial w_{\hat{i}j}}{\partial w_{\hat{l}k}} b_{\hat{l}}) = a_{\hat{i}} \frac{\partial w_{\hat{i}j}}{\partial w_{jk}} b_j = a_k \frac{\partial w_{kj}}{\partial w_{jk}} b_j \quad \text{(eliminating the zero terms)} \\
&= a_k b_j \qquad (1.13)
\end{aligned}$$

From here, we conclude that the equivalent result of Eq. (1.8) is obtained.

1.2.2 Gradient Vanish and Gradient Exploding

Bibliography

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533 EP –, 10 1986. [Online]. Available: http://dx.doi.org/10.1038/323533a0

[2] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Trans Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. [Online]. Available: https://ieeexplore.ieee.org/document/279181/

[3] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” *Siam Review*, vol. 51, no. 3, pp. 455–500, 2009.