Recurrent Neural Networks

Gregory Feldmann

July 6, 2020

Abstract

An overview of Recurrent Neural Network (RNN) architectures and training is given. Vanilla RNN cells and backpropagation through time (BPTT) are covered, as well as the issues with Vanilla RNNs that motivated Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) cells. Only single-node cells are considered to simplify the math, though generalisation to multiple nodes is quite simple.

1 Vanilla RNN Cells

1.1 Overview

The following definition is based on [3]. Suppose that at time t we have an input sequence $\mathbf{x}_t = (x_1, x_2, x_3, ...x_m)$ of length m. A simple RNN cell with one hidden unit h first does the following

$$a_{hidden,t} = \mathbf{w}_{in} \bullet \mathbf{x}_t + w_{hidden} h_{t-1} + b_{hidden} \tag{1}$$

$$h_t = f_{hidden}(a_{hidden,t}) \tag{2}$$

where h_t is the value of the hidden state h at time t, $\mathbf{w}_{in}$ is a vector of weights applied to $\mathbf{x}$, w_{hidden} is the weight applied to h_{t-1}, b_{hidden} is the bias and f_{hidden} is a non-linearity such as tanh. The output of the vanilla RNN cell is then computed as follows

$$a_{out} = w_{out} h_t + b_{out} \tag{3}$$

$$\hat{y}_t = f_{out}(a_{out}) \tag{4}$$

where $\hat{y}_t$ is the output at time t, w_{out} is the weight applied to h_t, b_{out} is the bias and f_{out} is an activation function.

We can see that h_t stores information about previous computations in the RNN. This allows it to 'remember' information and re-use it in future calculations.

1.2 Backpropagation Through Time

Backpropagation through time (BPTT) is a modification of backpropagation designed to accomodate the recurrent nature of RNNs. The basic idea is to 'unroll' the RNN cell by treating each step in time of the cell as a feed-forward layer taking in the input $\mathbf{x}_t$ and the previous hidden state h_{t-1} as inputs. Backpropagation is then done as per feedforward neural networks on the 'unrolled' RNN (see).

Consider a single-cell RNN (as above) with a loss function $l(\hat{y}_t, y_t)$, where y_t is the target value at time t that we are attempting to approximate with $\hat{y}_t$. Suppose we have a single input sequence $\mathbf{x}$ of length T, whose elements are denoted by x_t (i.e. t runs from 1 to T), and that the we only score the final output $\hat{y}_T$. This could represent, for example, forecasting the next value in a time series of length T. By unrolling the network for each step forward in the sequence, we end up with a length T feedforward network, to which we apply backpropagation to obtain $\partial l / \partial w_{hidden}$ [the following equations need fixing...I forgot to differentiate with respect to the linear combinations inside non-linearities]

$$\frac{\partial l}{\partial w_h} = \frac{\partial l}{\partial \hat{y}_T} . \frac{\partial \hat{y}_T}{\partial h_T} \frac{h_T}{w_h} \tag{5}$$

$$\frac{\partial h_T}{\partial w_h} = \frac{\partial}{\partial w_h} \Big(f_h(w_{in} x_T + w_h h_{T-1} + b_h) \Big) \tag{6}$$

The equation for $\partial h_T / \partial w_h$ presents us with a new challenge as h_{T-1} is a function of w_h. To tackle it, we use the product and chain rules of calculus

$$\frac{\partial}{\partial w_h} \Big(f_h(w_{in} x_T + w_h h_{T-1} + b_h) \Big) = \frac{\partial f_{h|T}}{\partial w_h} \left(h_{T-1} + w_h \frac{\partial h_{T-1}}{\partial w_h} \right) \tag{7}$$

The bracketed term on the right hand side can be re-expressed as

$$h_{T-1} + w_h \frac{\partial h_{T-1}}{\partial w_h} = \left. \frac{\partial h_T}{\partial w_h} \right|_{h_{T-1}} + \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial w_h} \tag{8}$$

This can be generalised to any t satisfying $0 < t \leq T$

$$\frac{\partial h_t}{\partial w_h} = \left. \frac{\partial h_t}{\partial w_h} \right|_{h_{t-1}} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h} \tag{9}$$

We can then use equation 9 to define equation 5 recursively, giving us the following

$$\frac{\partial l}{\partial w_h} = \frac{\partial l}{\partial \hat{y}_T} . \frac{\partial \hat{y}_T}{\partial h_T} \left(\frac{\partial f_{h|T}}{\partial w_h} \left(\left. \frac{\partial h_T}{\partial w_h} \right|_{h_{T-1}} + \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial w_h} \right) \right) \tag{10}$$

By continuing to 'unroll' the recursive elements using equation 9, we can re-express this as

$$\frac{\partial l}{\partial w_h} = \sum_{t=0}^{T-1} \frac{\partial l}{\partial \hat{y}_T} . \frac{\partial \hat{y}_T}{\partial h_T} \left(\prod_{\tau=0}^{t} \frac{\partial f_{h|T}}{\partial w_h} \frac{\partial h_{T-\tau}}{\partial h_{T-\tau-1}} \right) \left. \frac{\partial h_{T-t}}{\partial w_h} \right|_{h_{T-t-1}} \tag{11}$$

1.2.1 Truncated Backpropagation Through Time

1.3 Issues with Vanilla RNNs

While in theory Vanilla RNNs can learn to utilise information from previous time-steps arbitrarily into the past, they tend to be limited to utilising information from more recent time-steps due to vanishing and exploding gradients (see [1] and [2]).

Vanishing and exploding gradients are due to the recurrent nature of RNNs. During training, as the recurrent networks look further and further back in time, the gradient becomes a long chain of multiplications that is prone to going to either 0 or ∞. This can be made clear in equation 11 by focusing on the product term $\prod_{\tau=0}^{t} \frac{\partial h_{T-\tau}}{\partial h_{T-\tau-1}}$, which we can express as $\prod_{\tau=0}^{t} w_h = (w_h)^t$. When $|w_h| < 1$, $\lim_{t\to\infty}(w_h)^t = 0$ and when $|w_h| > 1$, $\lim_{t\to\infty}(w_h)^t \to \pm\infty$.

Extremely small gradients mean that learning will proceed slowly, if at all. Extremely large gradients lead to unstable learning that fails to converge. Either case makes training RNNs on long sequences quite difficult.

1.3.1 Gradient Clipping

Gradient clipping limits the maximum value a gradient can take, meaning it cannot increase without limit. Gradient norm clipping (see [5]) rescales the norm of a gradient to equal some value $\lambda_{threshold}$ whenever the norm of the gradient is greater than $\lambda_{threshold}$.

1.3.2 Moving Past Vanilla RNNs

While gradient clipping can help prevent exploding gradients, it doesn't help with vanishing gradients. More effective recurrent cells have been developped which address both vanishing and exploding gradients (to a degree); the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) cell.

2 Gated Recurrent Units

Gated recurrent units (GRUs), proposed in [4], are more complex than RNNs, but simpler than LSTMs (more on LSTMs below). For this reason they are discussed before LSTMs, despite being discovered after them. As GRUs have fewer trainable parameters than LSTMs, they tend to perform better on smaller datasets than LSTMs.

GRUs build on Vanilla RNNs by adding *reset* and *update* gates. As in the vanilla RNN case, we assume only one hidden unit. In this case, the reset gate is given by

$$r = f_\sigma\Bigg(\mathbf{w}_r \bullet \mathbf{x}_t + w_{reset}h_{t-1} + b_{reset}\Bigg) \tag{12}$$

where r is the value of the reset gate, $\mathbf{w}_r$ is a vector of weights applied to x_t, w_{reset} is the weights applied to the hidden state h_{t-1} and f_σ is the sigmoid activation. The update gate is given by

$$z = f_\sigma\Bigg(\mathbf{w}_z \bullet \mathbf{x}_t + w_{forget}h_{t-1} + b_{forget}\Bigg) \tag{13}$$

where z is the value of the update gate and the remaining variables are defined similarly to those in the reset gate.

The reset and update gates are then used to calculate the hidden state as follows

$$h_t = zh_{t-1} + (1-z)\tilde{h}_t \tag{14}$$

where $\tilde{h}_t$ is the *candidate hidden state* and is defined as follows

$$\tilde{h}_t = f_{hidden}(\mathbf{w}_{\tilde{h}} \bullet \mathbf{x}_t + w_{\tilde{h}}rh_{t-1} + b_{\tilde{h}}) \tag{15}$$

When the reset gate r is close to 0, $\tilde{h}$ becomes independant of h_{t-1} and there is greater emphasis on calculating $\tilde{h}$ using information from the input values $\mathbf{x}_t$. The reset gate thus determines how much to 'forget' about previously stored information [4]. This helps to capture short-term dependencies [**?**].

When the update gate is close to 1, the hidden state becomes independant of $\tilde{h}$. In other words, the update gate determines how much of the old state to retain in the new state. This helps to capture long-term dependencies [**?**].

2.1 Minimal GRUs

A reduced version of the GRU, known as the Minimal GRU, was put forth in [7]. The Minimal GRU consolidates the update and reset gates of the GRU into a single gate by requiring $reset_t = forget_t$. The Minimal GRU first calculates the reset/forget gate, z, as follows

$$z_t = f_\sigma\Bigg(\mathbf{w}_z \bullet \mathbf{x}_t + w_{forget}h_{t-1} + b_{forget}\Bigg) \tag{16}$$

followed by the candidate hidden state and final hidden state

$$\tilde{h}_t = f_{hidden}(\mathbf{w}_{\tilde{h}} \bullet \mathbf{x}_t + w_{\tilde{h}}z_th_{t-1} + b_{\tilde{h}}) \tag{17}$$

$$h_t = zh_{t-1} + (1-z)\tilde{h}_t \tag{18}$$

These are seen to be of the same form as the GRU equations, with z substituted in place of r.

3 Long Short-Term Memory Cells

Long short-term memory (LSTM) neural networks are more complex than GRUs, mainly due to the addition of a memory cell and the output and hidden states being distinct. The basic LSTM includes an input gate, output gate, hidden state and a memory cell [6]. A more complex form also includes a forget gate.

$$I_t = f_{sigma}(\mathbf{w}_{input} \bullet \mathbf{x}_t + w_{input} h_{t-1} + b_{input}) \tag{19}$$

$$O_t = f_{sigma}(\mathbf{w}_{output} \bullet \mathbf{x}_t + w_{output} h_{t-1} + b_{output}) \tag{20}$$

The candidate memory cell is given by

$$\tilde{C}_t = f_{tanh}(\mathbf{w}_{\tilde{C}_t} \bullet \mathbf{x}_t + w_{\tilde{C}} h_{t-1} + b_{\tilde{C}_t}) \tag{21}$$

and the memory cell by

$$C_t = C_{t-1} + I_t \tilde{C}_t \tag{22}$$

Finally, the hidden state is given by

$$H_t = O_t f_{tanh}(C_t) \tag{23}$$

3.1 LSTM with Forget Gate

$$I_t = f_{sigma}(\mathbf{w}_{input} \bullet \mathbf{x}_t + w_{input} h_{t-1} + b_{input}) \tag{24}$$

$$F_t = f_{sigma}(\mathbf{w}_{forget} \bullet \mathbf{x}_t + w_{forget} h_{t-1} + b_{forget}) \tag{25}$$

$$O_t = f_{sigma}(\mathbf{w}_{output} \bullet \mathbf{x}_t + w_{output} h_{t-1} + b_{output}) \tag{26}$$

The candidate memory cell is given by

$$\tilde{C}_t = f_{tanh}(\mathbf{w}_{\tilde{C}_t} \bullet \mathbf{x}_t + w_{\tilde{C}} h_{t-1} + b_{\tilde{C}_t}) \tag{27}$$

and the memory cell by

$$C_t = F_t C_{t-1} + I_t \tilde{C}_t \tag{28}$$

Finally, the hidden state is given by

$$H_t = O_t f_{tanh}(C_t) \tag{29}$$

See the Dive into Deep Learning book for an excellent explanation of LSTMs.

4 Bi-directional RNNs

References

[1] Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE International Conference on Neural Networks*, pages 1183–1188 vol.3, 1993.

[2] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[3] John A Bullinaria. Recurrent neural networks neural computation : Lecture 12, 2015. Available at https://www.cs.bham.ac.uk/ jxb/INC/l12.pdf.

[4] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[5] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–1310–III–1318. JMLR.org, 2013.

[6] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[7] Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou. Minimal gated unit for recurrent neural networks, 2016.