Lecture 13. Notes on recurrent models for time series forecasting

Alexey Artemov, Fintech CS HSE

May 31, 2018

1 The architecture of RNNs

2 Backpropagation through time (BPTT)

2.1 Forward propagation equations

Here we consider the case of a network that maps an input sequence to an output sequence of the same length.

$$\begin{aligned} \mathbf{h}^{(t)} &= \tanh(\mathbf{b} + \boldsymbol{W}\mathbf{x}_t + \boldsymbol{U}\mathbf{h}^{(t-1)}), \\ \mathbf{o}^{(t)} &= \mathbf{c} + \boldsymbol{V}\mathbf{h}^{(t)}, \\ \widehat{\mathbf{y}}^{(t)} &= \mathrm{softmax}(\mathbf{o}^{(t)}) \end{aligned}$$

Network parameters: $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}, \mathbf{b}, \mathbf{c}$.

2.2 The optimization setting

Consider a training sample $\mathbf{X}^\ell = \{(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\ell)}), (\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(\ell)})\}$. Suppose we have a loss function $L : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ (such as a cross-entropy loss). For "input sequence-output sequence" prediction, we compute the total loss as the sum of losses over all available time points:

$$L^{\text{total}}(\mathbf{X}^\ell) = \sum_{t=1}^{\ell} L(\mathbf{y}^{(t)}, \widehat{\mathbf{y}}^{(t)})$$

where $\widehat{\mathbf{y}}^{(t)}$ may generally be computed based on all $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)})$ up to time t.

2.3 The unrolling of RNNs

2.4 Backpropagation equations

To implement BPTT, we need to compute the derivatives of L w.r.t. network parameters $(\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}, \mathbf{b}, \mathbf{c})$ and all the variables $(\mathbf{x}^{(t)}, \mathbf{h}^{(t)}, \mathbf{o}^{(t)})$.

- *The gradient* $\nabla_{\mathbf{o}^{(t)}} L$:

$$\nabla_{\mathbf{o}^{(t)}} L = \widehat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}.$$

  Proof: use derivatives **A** and **B**.

- *The gradient* $\nabla_{\mathbf{h}^{(t)}} L$. For $t = \ell$ (end of sequence), as no $\mathbf{h}^{(\ell+1)}$ exists:

$$\nabla_{\mathbf{h}^{(\ell)}} L = \boldsymbol{V}^\intercal (\nabla_{\mathbf{o}^{(t)}} L).$$

  Proof: use chain rule ($\frac{\partial L}{\partial \mathbf{h}_k^{(t)}} = \frac{\partial L}{\partial \mathbf{o}_i^{(t)}} \frac{\partial \mathbf{o}_i^{(t)}}{\partial \mathbf{h}_k^{(t)}}$) and sum over $\mathbf{o}_i^{(t)}$.

  For $t < \ell$, as both $\mathbf{h}^{(t+1)}$ and $\mathbf{o}_k^{(t)}$ descend from $\mathbf{h}^{(t)}$:

$$\nabla_{\mathbf{h}^{(\ell)}} L = \boldsymbol{W}^\intercal (\nabla_{\mathbf{h}^{(t+1)}} L) \mathrm{diag}(1 - (\mathbf{h}^{(t+1)})^2) + \boldsymbol{V}^\intercal (\nabla_{\mathbf{o}^{(t)}} L).$$

  Proof: note that $\nabla_{\mathbf{h}^{(t)}} L$ is the sum of $\left(\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}}\right)^\intercal \nabla_{\mathbf{h}^{(t+1)}} L$ and $\left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}}\right)^\intercal \nabla_{\mathbf{o}^{(t)}} L$. Use derivative **C**.

- *The gradient* $\nabla_{\boldsymbol{V}} L$:

$$\nabla_{\boldsymbol{V}} L = \sum_t (\nabla_{\mathbf{o}^{(t)}} L)(\mathbf{h}^{(t)})^\intercal.$$

  Proof: for fixed t use chain rule ($\frac{\partial L}{\partial \boldsymbol{V}} = \frac{\partial L}{\partial \mathbf{o}_i^{(t)}} \nabla_{\boldsymbol{V}} \mathbf{o}_i^{(t)}$), sum over $\mathbf{o}_i^{(t)}$. Unroll to sum over t.

- *The gradient* $\nabla_{\boldsymbol{W}} L$:

$$\nabla_{\boldsymbol{W}} L = \sum_t \mathrm{diag}(1 - (\mathbf{h}^{(t)})^2)(\nabla_{\mathbf{h}^{(t)}} L)(\mathbf{h}^{(t-1)})^\intercal.$$

  Proof: for fixed t use chain rule ($\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \mathbf{h}_i^{(t)}} \nabla_{\boldsymbol{W}} \mathbf{h}_i^{(t)}$), sum over $\mathbf{h}_i^{(t)}$. Use derivative **C**. Unroll to sum over t.

- *The gradient* $\nabla_{\boldsymbol{U}} L$:

$$\nabla_{\boldsymbol{U}} L = \sum_t \mathrm{diag}(1 - (\mathbf{h}^{(t)})^2)(\nabla_{\mathbf{h}^{(t)}} L)(\mathbf{x}^{(t)})^\intercal.$$

  Proof: for fixed t use chain rule ($\frac{\partial L}{\partial \boldsymbol{U}} = \frac{\partial L}{\partial \mathbf{h}_i^{(t)}} \nabla_{\boldsymbol{U}} \mathbf{h}_i^{(t)}$), sum over $\mathbf{h}_i^{(t)}$. Use derivative **C**. Unroll to sum over t.

3 Difficulties with training RNNs

3.1 Useful mathematical concepts

Spectral radius

3.2 Vanishing and exploding gradients

Why gradients may vanish

Why gradients may explode

What to do about vanishing and exploding gradients

- Penalization of training with L1/L2 loss.
- Pre-programming the model to *behave* like the target.
- Teacher forcing: using targets for some or all of the hidden units.
- LSTM model for learnable control over

See (Pascanu, Mikolov, and Bengio 2013). (Bengio, Simard, and Frasconi 1994) considers in detail the most simple 1-hidden-neuron case. Outlined in (Hochreiter 1998) and

4 Long short-term memory networks

5 More reading for the curious

RNNs have been studied for over 25 years now, from very different perspectives. Reading older papers (from 1990s) is somewhat more laborious as they revolve around settings less relevant to current interest (back then people were "trying to make things learn" rather then trying to solve actual problems) and used the nowadays outdated terminology (usual for young areas of science). Nevertheless, the following papers can be valuable for in-depth understanding of RNNs:

- (Williams and Zipser 1995) provides an overview of gradient-based learning methods for RNNs.
- (Jaeger 2002) is a nice introductory tutorial covering BPTT, RTRL, and EKF algorithms for training RNNs.
- WildML has a hice series of tutorials on RNN/LSTM concepts, problems and implementation in both raw `numpy` and `theano`.

Appendix. Notation and useful mathematical derivations

This section is expected to be a mathematical reference. In all the formulas in this section, we only display derivatives w.r.t. the closest parameter variables. Where possible, we also write gradients. Obtaining more complicated derivatives is easy: use the chain rule.

Notation All vectors $\mathbf{x} \in \mathbb{R}^n$ are column vectors: shape of $\mathbf{x}$ is $n \times 1$.

A. Derivative of cross-entropy loss Cross-entropy loss is defined as:

$$L(\mathbf{y}, \widehat{\mathbf{y}}) = -\sum_{i=1}^{K} y_i \log \widehat{y}_i$$

(usually K is the number of classes in a classification problem).

We seek to derive $\frac{\partial L}{\partial \widehat{y}_i}$:

$$\frac{\partial L}{\partial \widehat{y}_i} = \frac{\partial}{\partial \widehat{y}_i}\Big[-\sum_{k=1}^{K} y_k \log \widehat{y}_k\Big] = -\frac{y_i}{\widehat{y}_i}.$$

Thus $\nabla_{\widehat{\mathbf{y}}} L = -\big(\frac{y_1}{\widehat{y}_1}, \ldots, \frac{y_K}{\widehat{y}_K}\big)$.

B. Derivative of softmax Softmax function is defined as:

$$\widehat{y}_j = \frac{\exp(z_j)}{\sum_{i=1}^{N} \exp(z_i)}$$

We seek to derive $\frac{\partial \widehat{y}_j}{\partial z_k}$. If $k = j$:

$$\begin{aligned}
\frac{\partial \widehat{y}_j}{\partial z_j} &= \frac{\partial}{\partial z_j}\Big[\frac{\exp(z_j)}{\sum_{i=1}^{N} \exp(z_i)}\Big] = \\
&= \frac{\exp(z_j)\sum_{i=1}^{N} \exp(z_i) - \exp(z_j)\exp(z_j)}{\big[\sum_{i=1}^{N} \exp(z_i)\big]^2} = \\
&= \frac{\exp(z_j)}{\sum_{i=1}^{N} \exp(z_i)} - \frac{\big[\exp(z_j)\big]^2}{\big[\sum_{i=1}^{N} \exp(z_i)\big]^2} = \\
&= \widehat{y}_j(1 - \widehat{y}_j).
\end{aligned}$$

Otherwise, if $k \neq j$:

$$\begin{aligned}
\frac{\partial \widehat{y}_j}{\partial z_k} &= \frac{\partial}{\partial z_k}\Big[\frac{\exp(z_j)}{\sum_{i=1}^{N} \exp(z_i)}\Big] = \\
&= \exp(z_j)\frac{-\exp(z_k)}{\big[\sum_{i=1}^{N} \exp(z_i)\big]^2} = \\
&= -\frac{\exp(z_j)}{\sum_{i=1}^{N} \exp(z_i)}\frac{\exp(z_k)}{\sum_{i=1}^{N} \exp(z_i)} = \\
&= -\widehat{y}_j\widehat{y}_k.
\end{aligned}$$

Thus, overall, $\frac{\partial \widehat{y}_j}{\partial z_k} = \widehat{y}_j(\delta_{jk} - \widehat{y}_k)$ where δ_{jk} is the Kronecker delta.

C. Derivative of tanh (hyperbolic tangent) Hyperbolic tangent function is defined as:

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}.$$

We seek to derive $\frac{\partial \tanh(x)}{\partial x}$:

$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x).$$

References

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Hochreiter, Sepp (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116.

Jaeger, Herbert (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. Vol. 5. GMD-Forschungszentrum Informationstechnik Bonn.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks". In: *International Conference on Machine Learning*, pp. 1310–1318.

Williams, Ronald J and David Zipser (1995). "Gradient-based learning algorithms for recurrent networks and their computational complexity". In: *Backpropagation: Theory, architectures, and applications* 1, pp. 433–486.