Recurrent Neural Networks

Advanced Topics in High-Performance Computing

Faisal Qureshi
Copyright information

These slides draw heavily upon works of many individuals, notably among them are:

- Nando de Freitas
- Fei-Fei Li
- Andrej Karpathy
- Justin Johnson
Recurrent Neural Networks (RNN)

- one to one: image classification
- one to many: image captioning
- many to one: sentiment analysis
- many to many: machine translation
- many to many: video understanding

[From A. Karpathy Blog]
Sequential processing of fixed inputs

- Multiple object recognition with visual attention, Ba et al.
Sequential processing of fixed outputs

- **DRAW**: a recurrent neural network for image generation, Gregor et al.
Recurrent Neural Network

- $h_t = \phi_1(h_{t-1}, x_t)$
- $\hat{y}_t = \phi_2(h_t)$

Where

- $x_t =$ input at time $t$
- $\hat{y}_t =$ prediction at time $t$
- $h_t =$ new state
- $h_{t-1} =$ previous state
- $\phi_1$ and $\phi_2 =$ functions with parameters $W$s that we want to train

Subscript $t$ indicates sequence index.

Example

\[
    h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)
\]
\[
    \hat{y}_t = \text{softmax}(W_{hy}h_t)
\]
Parameters $W_{xh}$, $W_{hh}$ and $W_{hy}$ are tied over time

Cost: $E = \sum_t E_t$, where $E_t$ depends upon $y_t$

Training: minimize $E$ to estimate $W_{xh}$, $W_{hh}$ and $W_{hy}$
Recurrent Neural Network: Loss

When dealing with output sequences, we can define loss to be a function of the predicted output $\hat{y}_t$ and the expected value $y_t$ over a range of times $t$

$$E(y, \hat{y}) = \sum_{t} E_t(y, \hat{y})$$

Example: using cross-entropy for k-class classification problem

$$E(y, \hat{y}) = -\sum_{t} y_t \log \hat{y}_t$$
Recurrent Neural Networks: Computing Gradients

We need to compute \( \frac{\partial E}{\partial W_{xh}} \), \( \frac{\partial E}{\partial W_{hh}} \), and \( \frac{\partial E}{\partial W_{hy}} \) in order to train an RNN.

Example

\[
\frac{\partial E_3}{\partial W_{hh}} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}
\]
Recurrent Neural Networks: Vanishing and Exploding Gradients

We can compute the highlighted term in the following expression using \textit{chain-rule}

\[
\frac{\partial E_3}{\partial W_{hh}} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}
\]

Applying the chain-rule

\[
\frac{\partial h_3}{\partial h_k} = \prod_{j=k+1}^{3} \frac{\partial h_j}{\partial h_{j-1}}
\]

Or more generally

\[
\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}
\]
Recurrent Neural Networks: Difficulties in Training

\[
\frac{\partial E_t}{\partial W_{hh}} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \left( \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W_{hh}}
\]

\[
\frac{\partial h_i}{\partial h_{i-1}}
\]

is a Jacobian matrix.

For longer sequences

- if \( \left| \frac{\partial h_i}{\partial h_{i-1}} \right| < 0 \), the gradients vanish
  - Gradient contributions from “far away” steps become zero, and the state at those steps doesn’t contribute to what you are learning.
  - Long short-term memory units are designed to address this issue

- if \( \left| \frac{\partial h_i}{\partial h_{i-1}} \right| > 0 \), the gradients vanish
  - Clip gradients at a predefined threshold

- See also, On the difficulty of training recurrent neural networks, Pascanu et al.
Image Captioning

Convolutional network trained for image classification

Recurrent Neural Network trained to generate image captions
For the image captioning example shown in the previous slide, $h_t$ is defined as follows:

$$h_t = \text{Tanh}(W_{hh}h_{t-1} + W_{xh}x + W_{ih}v)$$

$$\hat{y}_t = \text{softmax}(W_{hy}h_t)$$
Image Captioning

- Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
- Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy - and Fei-Fei
- Show and Tell: A Neural Image Caption Generator, Vinyals et al.
- Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
- Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick
Dealing with Vanishing Gradients

Change of notation

\[ c_t = \theta c_{t-1} + \theta_g g_t \]
\[ h_t = \tanh(c_t) \]
Long Short Term Memory (LSTM)
Long Short Term Memory (LSTM)

- Input gate: scales input to cell (write operation)
- Output gate: scales input from cell (read operation)
- Forget gate: scales old cell values (forget operation)

\[
\begin{align*}
i_t &= \text{sigm}(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i) \\
f_t &= \text{sigm}(\theta_{xf}x_t + \theta_{hf}h_{t-1} + b_f) \\
o_t &= \text{sigm}(\theta_{xo}x_t + \theta_{ho}h_{t-1} + b_o) \\
g_t &= \tanh(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{align*}
\]

- represent element-wise multiplication
RNN vs. LSTM

Check out the video at https://imgur.com/gallery/vaNahKE
Summary

- RNN
  - Allow a lot of flexibility in architecture design
  - Very difficult to train in practice due to vanishing and exploding gradients
  - Control gradient explosion via clipping
  - Control vanishing gradients via LSTMs

- LSTM
  - Very powerful architecture for dealing with sequences (input/output)
  - Works rather well in practice