Recurrent nets and LSTM

Nando de Freitas
Outline of the lecture

This lecture introduces you sequence models. The goal is for you to learn about:

- Recurrent neural networks
- The vanishing and exploding gradients problem
- Long-short term memory (LSTM) networks
- Applications of LSTM networks
  - Language models
  - Translation
  - Caption generation
  - Program execution
A simple recurrent neural network

\[ h_t = \theta \phi(h_{t-1}) + \theta_x x_t \]

\[ y_t = \theta_y \phi(h_t) \]
Vanishing gradient problem

\[
\begin{align*}
\mathbf{h}_t &= \underbrace{\theta \phi(\mathbf{h}_{t-1})} + \theta_x \mathbf{x}_t \\
\mathbf{y}_t &= \theta_y \phi(\mathbf{h}_t)
\end{align*}
\]

\[
\frac{\partial E}{\partial \theta} = \sum_{t=1}^{S} \frac{\partial E_t}{\partial \theta}
\]

\[
\frac{\partial E_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \theta}
\]

[Yoshua Bengio et al]
Vanishing gradient problem

\[
\frac{\partial E_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial \theta}
\]

\[
\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} \theta^T \text{diag}[\phi'(h_{i-1})]
\]

\[
\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \left\| \theta^T \right\| \left\| \text{diag}[\phi'(h_{i-1})] \right\| \leq \gamma \theta \gamma \phi
\]

\[
\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq (\gamma \theta \gamma \phi)^{t-k}
\]
Simple solution

\[ c_t = \Theta \cdot c_{t-1} + \Theta_s \cdot g_t \]

\[ h_t = \text{Tanh} (c_t) \]
LSTM

Input gate: scales input to cell (write)
Output gate: scales output from cell (read)
Forget gate: scales old cell value (reset)

[Alex Graves]
LSTM

\[ i_t = \text{Sigm}(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i) \]

\[ f_t = \text{Sigm}(\theta_{xf}x_t + \theta_{hf}h_{t-1} + b_f) \]

\[ o_t = \text{Sigm}(\theta_{xo}x_t + \theta_{ho}h_{t-1} + b_o) \]

\[ g_t = \text{Tanh}(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g) \]

\[ c_t = f_t \odot c_{t-1} + i_t \odot g_t \]

\[ h_t = o_t \odot \text{Tanh}(c_t) \]
Entry-wise multiplication layer

\[ z = f(x_1, x_2) = x_1 \odot x_2 \]

\[ \frac{\partial E}{\partial x_1} = \left( \frac{\partial E}{\partial z} \right) \frac{\partial z}{\partial x_1} = \frac{\partial E}{\partial z} \odot x_2 \]

\[ z_i = f(x_{1i}, x_{2i}) = x_{1i} x_{2i} \]

\[ \frac{\partial E}{\partial x_{1i}} = \sum_j \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial x_{1i}} = \frac{\partial E}{\partial z_i} x_{2i} \]
LSTM cell in Torch

```plaintext
local function make_lstm_step(opt, input, prev_h, prev_c)
local function new_input_sum()
    local x_to_h = nn.Linear(opt.rnn_size, opt.rnn_size)
    local h_to_h = nn.Linear(opt.rnn_size, opt.rnn_size)
    return nn.CAddTable()({ x_to_h(input), h_to_h(prev_h) })
end

local in_gate = nn.Sigmoid()(new_input_sum())
local forget_gate = nn.Sigmoid()(new_input_sum())
local cell_gate = nn.Tanh()(new_input_sum())
local next_c = nn.CAddTable()({
    nn.CMulTable()({{forget_gate, prev_c}},
    nn.CMulTable()({{in_gate, cell_gate}})})
local out_gate = nn.Sigmoid()(new_input_sum())
local next_h = nn.CMulTable()({out_gate, nn.Tanh()(next_c)})
return next_h, next_c
end
```
LSTM column in Torch

```python
local function make_lstm_network(opt)
    local n_layers = opt.n_layers or 1

    local x = nn.Identity()
    local prev_s = nn.Identity()
    local splitted_s = {prev_s:split(2 * n_layers)}
    local next_s = {}
    local inputs = {[0] = x}
    for i = 1, n_layers do
        local prev_h = splitted_s[2 * i - 1]
        local prev_c = splitted_s[2 * i]
        local next_h, next_c = make_lstm_step(opt, inputs[i - 1], prev_h, prev_c)
        next_s[#next_s + 1] = next_h
        next_s[#next_s + 1] = next_c
        inputs[i] = next_h
    end
    local module = nn.gModule({x, prev_s}, {inputs[n_layers], nn.Identity() (next_s)})
    module:getParameters():uniform(-0.08, 0.08)
    module = cuda(module)
    return module
end
```
LSTMs for sequence to sequence prediction

[Ilya Sutskever et al]
LSTMs for sequence to sequence prediction

80k softmax by 1000 dims
This is very big!

1000 LSTM cells
2000 dims per timestep

2000 x 4 = 8k dims per sentence

160k vocab in input language
Learning to parse

John has a dog. →  
[S NP VP NP NNP VBZ NP DT NN]$_{NP}$ VP . ]$_{VP}$ S

John has a dog. →  
(S (NP NNP)$_{NP}$ (VP VBZ (NP DT NN)$_{NP}$)$_{VP}$ . )$_{S}$
Learning to execute

Input:

```python
j = 8584
for x in range(8):
    j += 920
b = (1500 + j)
print((b + 7567))
```

Target: 25011.

[Wojciech Zaremba and Ilya Sutskever]
Video prediction

Real

Generated

Karol Gregor, Ivo Danihelka, Andriy Mnih, Daan Wierstra…

Google DeepMind
Hand-writing recognition and synthesis

Which is Real?

from his travels it might have been
from his travels it might have been
from his travels it might have been
from his travels it might have been
from his travels it might have been

[Alex Graves]
Neural Turing Machine (NTM)

External Input

Controller

Read Heads

Write Heads

Memory

External Output

[Alex Graves, Greg Wayne, Ivo Danihelka]
Neural Turing Machine (NTM)

\[ r_t \leftarrow \sum_i w_t^k(i) M_t(i) \quad \text{Read} \]

\[ \tilde{M}_t(i) \leftarrow M_{t-1}(i) \left[ 1 - w_t^w(i) e_t \right] \quad \text{Erase} \]

\[ M_t(i) \leftarrow \tilde{M}_t(i) + w_t^w(i) a_t \quad \text{Write} \]
Neural Turing Machine (NTM)
Translation with alignment (Bahdanau et al)

\[ p(y_i | y_1, \ldots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i) \]

\[ s_i = f(s_{i-1}, y_{i-1}, c_i) \]

context vector \[ c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \]

\[ \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \]

\[ e_{ij} = a(s_{i-1}, h_j) \]
Show, attend and tell

1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

[Kelvin Xu et al, 2015]
Show, attend and tell

\[ a = \{ a_1, \ldots, a_L \}, \quad a_i \in \mathbb{R}^D \]

\[ \hat{z}_t = \phi (\{ a_i \}, \{ \alpha_i \}) = \sum_{i=1}^{L} \alpha_i a_i \]

\[ e_{ti} = f_{\text{att}}(a_i, h_{t-1}) \]

\[ \alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})} \]

\[ y = \{ y_1, \ldots, y_C \}, \quad y_i \in \mathbb{R}^K \]
In the next lecture, we will look techniques for unsupervised learning known as autoencoders. We will also learn about sampling and variational methods.

I **strongly recommend** reading Kevin Murphy’s variational inference book chapter prior to the lecture.