Recurrent Neural Networks (RNN)

Dr. Josif Grabocka

ISMLL, University of Hildesheim

Recurrent Neural Networks

Unfolding Computational Graphs

- Activation in a recurrent network depend on the activation history

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$

- The unfolded recurrence after t steps with a function $g^{(t)}$:

$$\begin{aligned} h^{(t)} &= g^{(t)}(x^{(t)}, x^{(t-1)}, \ldots, x^{(1)}) \\ h^{(t)} &= f(h^{(t-1)}, x^{(t)}; \theta) \end{aligned}$$

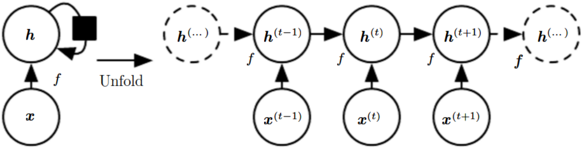

Figure 1: A recurrent computational graph, Source: Goodfellow et al., 2016

Recurrent Neural Networks (RNN)

- Regardless of sequence length the model has same input size
- It is possible to use the same transition function with same parameters
- RNNs have recurrent connections between hidden units

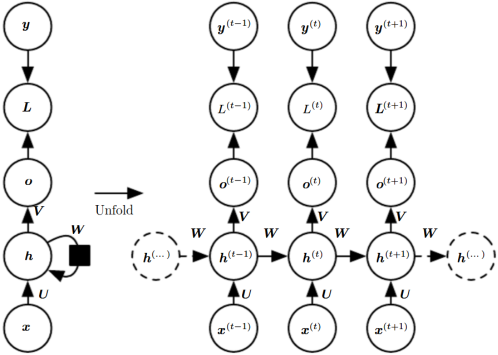

Figure 2: A recurrent neural network, Source: Goodfellow et al., 2016

RNN - Prediction Model (Single Layer)

- The aggregation $a \in \mathbb{R}^{N \times 1}$ depends on the previous activations $h^{(t-1)} \in \mathbb{R}^{N \times 1}$ and current input $x^{(t)} \in \mathbb{R}^{M \times 1}$:

$$a^{(t)} = b + W\, h^{(t-1)} + U\, x^{(t)}, \quad W \in \mathbb{R}^{N \times N}, U \in \mathbb{R}^{N \times M}$$

- The activations $h^{(t-1)} \in \mathbb{R}^{N \times 1}$ are non-linear firings:

$$h^{(t)} = \tanh(a^{(t)})$$

- The per-label outputs $o^{(t)} \in \mathbb{R}^{L \times 1}$ are:

$$o^{(t)} = c + V\, h^{(t)}, \quad V \in \mathbb{R}^{L \times N}$$

- And the predictions are the softmax of the per-label outputs:

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)}), \;\; \text{i.e.:} \;\; y_\ell^{(t)} = \frac{e^{o_\ell^{(t)}}}{\sum_{\ell'=1}^{L} e^{o_{\ell'}^{(t)}}}$$

RNN Loss

- The loss is defined as the negative likelihood of y^τ given $x^{(1)}, \ldots, x^{(\tau)}$

$$\begin{aligned} & \mathcal{L}\left(\left\{x^{(1)}, \ldots, x^{(\tau)}\right\},\ \left\{y^{(1)}, \ldots, y^{(\tau)}\right\}\right) \\ = \ & \sum_{t=1}^{\tau} \mathcal{L}^{(t)} \\ = \ & -\sum_{t=1}^{\tau} \log P\left(y^{(t)} \mid \left\{x^{(1)}, \ldots, x^{(\tau)}\right\}\right) \end{aligned}$$

- States computed during the $\mathcal{O}(\tau)$ forward pass needs to be stored for back-propagation through time (BPTT)

RNN Learning - BPTT

- Gradient of loss w.r.t. the output at time step t is:

$$\frac{\partial \mathcal{L}}{\partial o_\ell^{(t)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}^{(t)}} \frac{\partial \mathcal{L}^{(t)}}{\partial o_\ell^{(t)}} = \hat{y}_\ell^{(t)} - 1_{\ell, y^{(t)}}$$

- For the last sequence prediction at time τ:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial h_i^{(\tau)}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{L}^{(\tau)}} \frac{\partial \mathcal{L}^{(\tau)}}{\partial h_i^{(\tau)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}^{(\tau)}} \sum_{\ell=1}^{L} \frac{\partial \mathcal{L}^{(\tau)}}{\partial o_\ell^{(\tau)}} \frac{\partial o_\ell^{(\tau)}}{\partial h_i^{(\tau)}} \\ &= \sum_{\ell=1}^{L} \left(\hat{y}_\ell^{(\tau)} - 1_{\ell, y^{(\tau)}} \right) V_{\ell, i} \end{aligned}$$

- Back-propagate $\frac{\partial \mathcal{L}}{\partial h_i^{(t)}}$ to compute $\frac{\partial \mathcal{L}}{\partial h_i^{(t-1)}}$, for $t = \tau, \tau - 1, \ldots, 2$

RNN Learning - BPTT (2)

- Using previously computed $\frac{\partial \mathcal{L}}{\partial h^{(t+1)}}$ and stored $h^{(t+1)}, \hat{y}^{(t)}$
- For $1 < t < \tau$, note that $h_i^{(t)}$ contributes to all $h^{(t+1)} \in \mathbb{R}^N$ and all $o^{(t)} \in \mathbb{R}^L$, leading to:

$$\frac{\partial \mathcal{L}}{\partial h_j^{(t)}} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial h_i^{(t+1)}} \frac{\partial h_i^{(t+1)}}{\partial h_j^{(t)}} + \sum_{\ell=1}^{L} \frac{\partial \mathcal{L}}{\partial o_\ell^{(t)}} \frac{\partial o_\ell^{(t)}}{\partial h_j^{(t)}}$$

$$= \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial h_i^{(t+1)}} \left(1 - \left(h_i^{(t+1)}\right)^2\right) W_{i,j} + \sum_{\ell=1}^{L} \left(\hat{y}_\ell^{(t)} - 1_{\ell, y^{(t)}}\right) V_{\ell,j}$$

- Keep back-propagating $\frac{\partial \mathcal{L}}{\partial h^{(t)}}$ to compute $\frac{\partial \mathcal{L}}{\partial h^{(t-1)}}$ until $t = 1$

RNN Learning - BPTT (3)

- Using computed gradients $\frac{\partial \mathcal{L}}{\partial h^{(t)}}$ and $\frac{\partial \mathcal{L}}{\partial o^{(t)}}$, for $t = \tau, \ldots, 1$
- Then we can compute gradient w.r.t. parameters:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial c_\ell} &= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial o_\ell^{(t)}} \frac{\partial o_\ell^{(t)}}{\partial c_\ell^{(t)}} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial o_\ell^{(t)}} \\
\frac{\partial \mathcal{L}}{\partial b_i} &= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial h_i^{(t)}} \frac{\partial h_i^{(t)}}{\partial b_i^{(t)}} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial h_i^{(t)}} \left(1 - \left(h_i^{(t)}\right)^2\right) \\
\frac{\partial \mathcal{L}}{\partial V_{\ell,i}} &= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial o_\ell^{(t)}} \frac{\partial o_\ell^{(t)}}{\partial V_{\ell,i}^{(t)}} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial o_\ell^{(t)}} h_i^{(t)}
\end{aligned}
$$

RNN Learning - BPTT (4)

- Continuing with the activation parameters W, U:

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} = \sum_{t=2}^{\tau} \frac{\partial \mathcal{L}}{\partial h_i^{(t)}} \frac{\partial h_i^{(t)}}{\partial W_{i,j}^{(t)}} = \sum_{t=2}^{\tau} \frac{\partial \mathcal{L}}{\partial h_i^{(t)}} \left(1 - \left(h_i^{(t)}\right)^2\right) h_j^{(t-1)}$$

$$\frac{\partial \mathcal{L}}{\partial U_{i,m}} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial h_i^{(t)}} \frac{\partial h_i^{(t)}}{\partial U_{i,m}^{(t)}} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial h_i^{(t)}} \left(1 - \left(h_i^{(t)}\right)^2\right) x_m^{(t)}$$

- BPTT recap:
  - Forward step: Compute and store $h^{(t)}, \hat{y}^{(t)}$, for $t = 1, 2, \ldots, \tau$
  - Backward step: Compute and store $\frac{\partial \mathcal{L}}{\partial h^{(t)}}, \frac{\partial \mathcal{L}}{\partial o^{(t)}}$, for $t = \tau, \tau - 1, \ldots, 1$
  - Update step: Compute $\frac{\partial \mathcal{L}}{\partial c}, \frac{\partial \mathcal{L}}{\partial b}, \frac{\partial \mathcal{L}}{\partial V}, \frac{\partial \mathcal{L}}{\partial W}, \frac{\partial \mathcal{L}}{\partial U}$

Long-term Dependencies

- The RNN function composition resembles matrix multiplication:

$$\begin{aligned} h^{(t)} &= W^T h^{(t-1)} \\ h^{(t)} &= \left(W^t\right)^T h^{(0)} \end{aligned}$$

- If W admits an decomposition with orthogonal Q and diagonal λ:

$$W = Q \Lambda Q^T$$

- Then the recurrence can be expressed as :

$$h^{(t)} = Q \Lambda^t Q^T h^{(0)}$$

- Eigenvalues $\Lambda < 1$ will decay to zero (vanishing gradient problem), while $\Lambda > 1$ will explode to infinity

Illustrating Vanishing Gradients

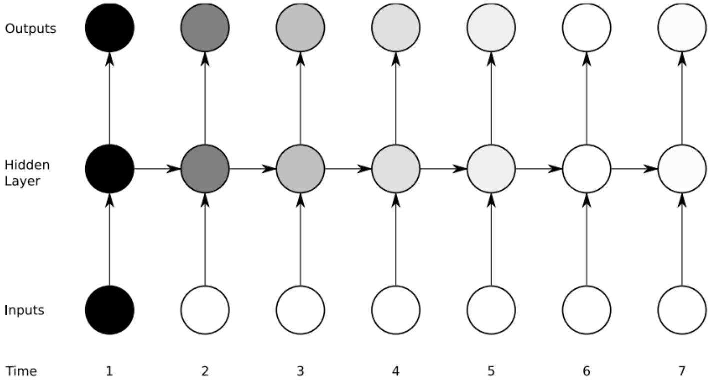

Figure 3: Sensitivity to the input at time one, Source: Graves 2008

Gating against Vanishing Gradients

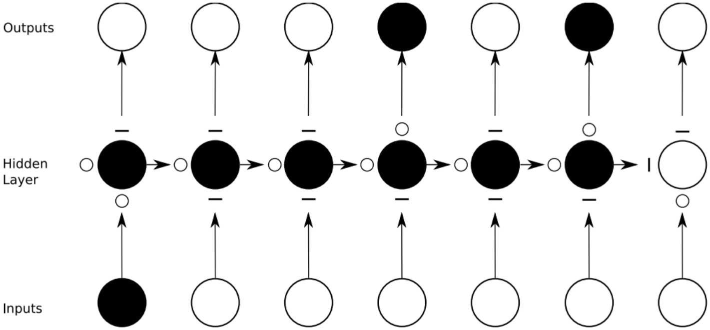

Figure 4: Gating helps remember, Source: Graves 2008

Long Short-Term Memory (LSTM)

- Gates: Nonlinear switch functions $\mathbb{R} \rightarrow [0, 1]$
- State := State · State_gate + Input · Input_gate
- Output := f(State) · Output_gate

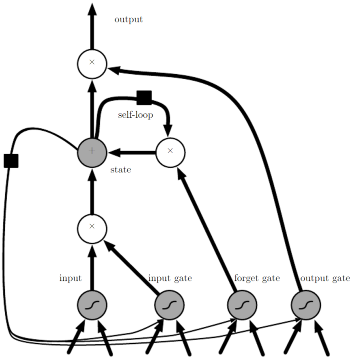

Figure 5: A LSTM neuron, Source: Goodfellow et al., 2016

LSTM (2)

- First of all, the input is gated as:

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_{m=1}^{M} U_{i,m}^g x_m^{(t)} + \sum_{j=1}^{N} W_{i,j}^g h_j^{(t-1)}\right)$$

- The state gate is also known as forget gate:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_{m=1}^{M} U_{i,m}^f x_m^{(t)} + \sum_{j=1}^{N} W_{i,j}^f h_j^{(t-1)}\right)$$

- Leading to a forget state with gated input:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_{m=1}^{M} U_{i,m} x_m^{(t)} + \sum_{j=1}^{N} W_{i,j} h_j^{(t-1)}\right)$$

LSTM (3)

- Finally the activation is a gated firing of state:

$$
\begin{aligned}
h_i^{(t)} &= \tanh\left(s_i^{(t)}\right)\ q_i^{(t)} \\
q_i^{(t)} &= \left(b_i^q + \sum_{m=1}^{M} U_{i,m}^q x_m^{(t)} + \sum_{j=1}^{N} W_{i,j}^q h_j^{(t-1)}\right)
\end{aligned}
$$

- There are four types of parameters in a LSTM neuron/cell:
  - Input: b, U, W
  - Input gate: b^g, U^g, W^g
  - State/forget gate: b^f, U^f, W^f
  - Output gate: b^q, U^q, W^q

Alternative: Gated Recurrent Unit

A simplified version of LSTM is the Gated Recurrent Unit:

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + \left(1 - u_i^{(t-1)}\right) \sigma \left(b_i + \sum_m U_{i,m} x_m^{(t)} + \sum_j W_{i,j} r_j^{(t-1)} \, h_j^{(t-1)} \right)$$

It utilizes u-update and r-reset gates:

$$u_i^{(t)} = \left(b_i^u + \sum_{m=1}^{M} U_{i,m}^u x_m^{(t)} + \sum_{j=1}^{N} W_{i,j}^u h_j^{(t-1)} \right)$$

$$r_i^{(t)} = \left(b_i^r + \sum_{m=1}^{M} U_{i,m}^r x_m^{(t)} + \sum_{j=1}^{N} W_{i,j}^r h_j^{(t-1)} \right)$$

What happens with $u_i^{(t)} = 0$ and $r_i^{(t)} = 1$? What about $u_i^{(t)} = 1$?

Clipping gradients

RNN produces strongly nonlinear loss functions which create cliffs:

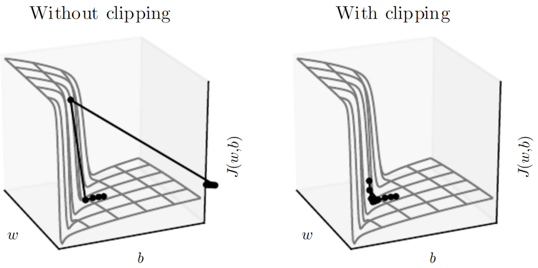

Figure 6: Clipping can avoid exploding gradients, Source: Goodfellow et al., 2016

A simple solution is the gradient clipping heuristic:

$$\text{if } ||g|| > v \text{ then } g \leftarrow \frac{gv}{||g||}$$