Recurrent Neural Networks

08-07-2018

Do Minh Hai

@dominhhai

Outline

- Time Series problem
- Recurrent Neural Networks - RNN
  - Lost Function
  - Backpropagation Through Time - BPPT
- Vanishing and Exploding Gradient problem
- Long Short-Term Memory - LSTM
- Gated Reccurent Unit - GRU
- Bidirectional RNNs
- Deep RNNs

Time Series problem

- Input: variable-length sequences of dependent input variables

$$P(\mathbf{x}_t|\mathbf{x}_{t-1}, ..., \mathbf{x}_1)$$

- Output: variable-length sequences of dependent output values

$$P(\mathbf{y}_t|\mathbf{y}_{t-1}, ..., \mathbf{y}_1, \mathbf{x})$$

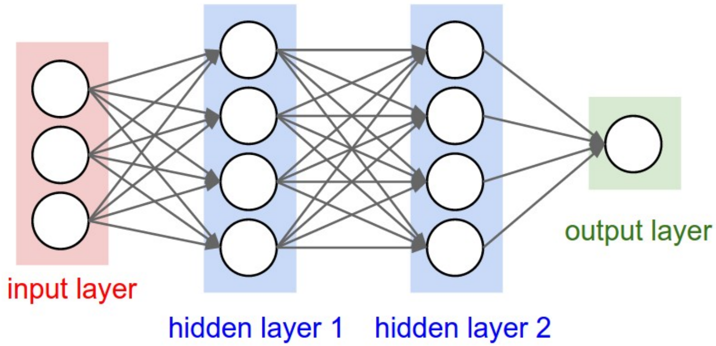
Language Model:

- Chữ tài đi với chữ tai một vần.
- He is Vietnamese. But he can not speak Vietnamese. 😳

Language Translation:

- Tao hôn nó. 😍 彼女にキスした。
- Nó hôn tao. 🙏 彼女からキスされる。

Time Series problem - FNN



FNN:

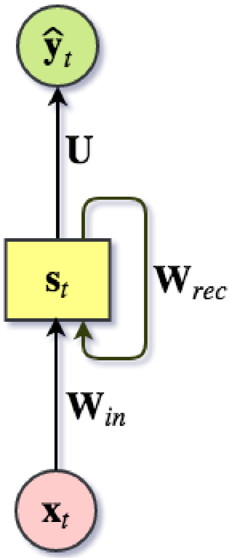
- Fixed input/output size
- Unordered input

Slide windows for sequences of inputs:

- window size may not fit
- window's weights are not shared

Figure: https://cs231n.github.io/neural-networks-1/

Recurrent Neural Networks - RNN



3 Node Types

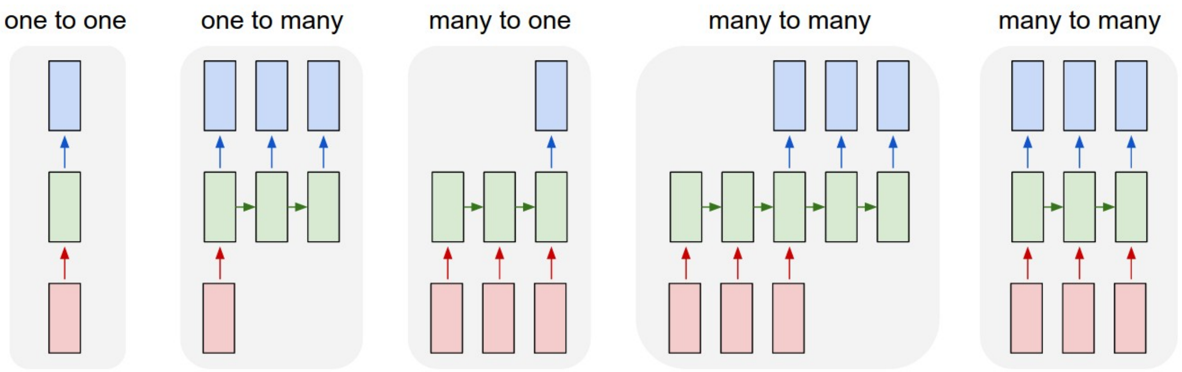
- Input Nodes: $\mathbf{x}_t$
- Recurrent Hidden Nodes: $\mathbf{s}_t$
  keeps order of hidden's state
- Output Nodes: $\hat{\mathbf{y}}_t$

Shared Weights

- Input Weights: $\mathbf{W}_{in}$
- Recurrent Weights: $\mathbf{W}_{rec}$
- Output Weights: $\mathbf{U}$

Hopfield (1982), Rumelhart et al. (1986a), Jordan (1986), Elman (1990)

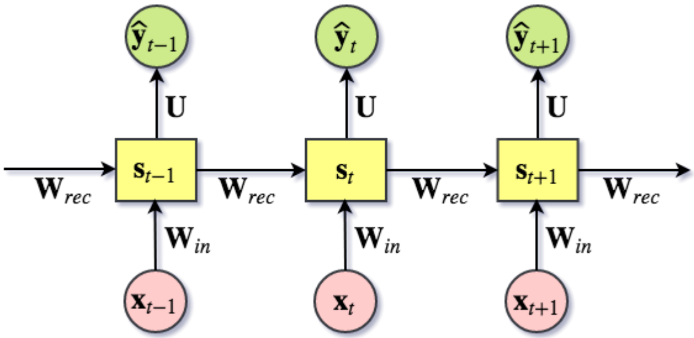
RNN - seq2seq



- Sequences in the input
- Sequences in the output
- Hidden Nodes is NOT fixed

Figure: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN - unroll



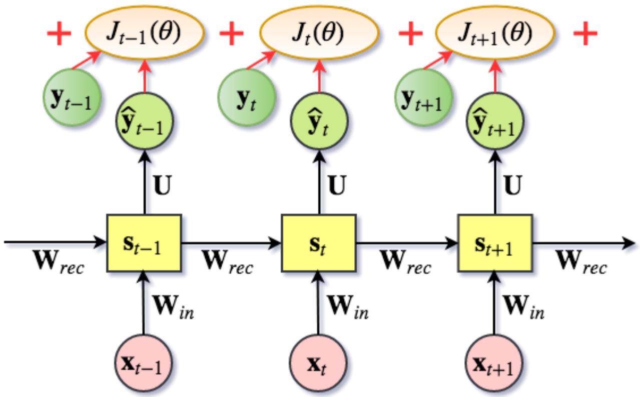
Calc Formulas

$$
\begin{aligned}
\mathbf{s}_t &= f(\mathbf{W}_{in}\mathbf{x}_t + \mathbf{W}_{rec}\mathbf{s}_{t-1} + \mathbf{b}_s) \\
\hat{\mathbf{y}}_t &= g(\mathbf{U}\mathbf{s}_t + \mathbf{b}_y)
\end{aligned}
$$

- $\mathbf{x}_t$: embedded word
- $\mathbf{s}_0$: 1st hidden state. Set to $\vec{\mathbf{0}}$ or *pre-trained* values.
- f : activation function. Usually the $\tanh$, ReLU, $sigmoid$.
- g : predict function. Such as, $softmax$ for language modeling.

Werbos (1990)

Lost Function



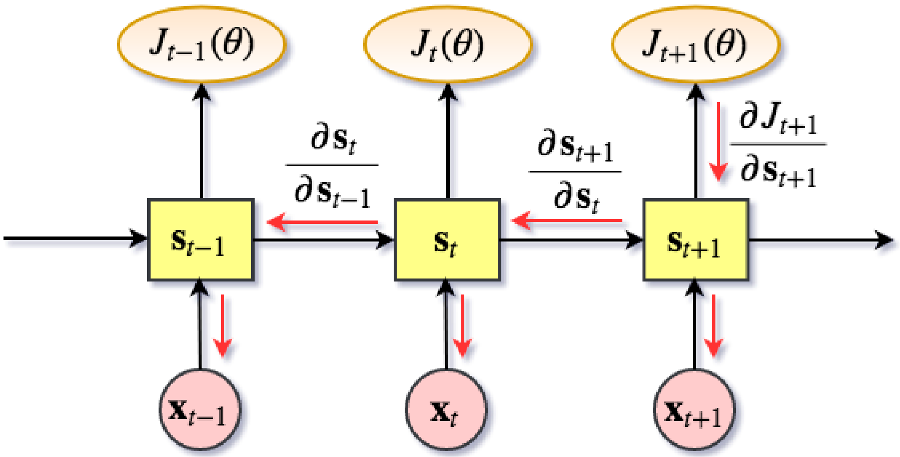
$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{N} y_{tj}\log \hat{y}_{tj}$$

Where:

- T: total time steps
- N: numbers of words
- $J_t(\theta)$: lost at step t

Werbos (1990)

Backpropagation Through Time - BPPT



Backprop over time steps $t = \overline{1, T}$ then summing gradient of each step.

- $(\mathbf{W}_{in}, \mathbf{W}_{rec}) = \mathbf{W} = \mathbf{W}^{(k)} \quad , k = \overline{1, T}$:

$$\frac{\partial J_t}{\partial \mathbf{W}} = \sum_{k=1}^{t} \frac{\partial J_t}{\partial \mathbf{W}^{(k)}} \frac{\partial \mathbf{W}^{(k)}}{\partial \mathbf{W}} = \sum_{k=1}^{t} \frac{\partial J_t}{\partial \mathbf{W}^{(k)}}$$

Werbos (1990)

Backpropagation Through Time - BPPT

Backprop over time steps $t = \overline{1, T}$ then summing gradient of each step.

Gradient Calc

$$\frac{\partial J}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial J_t}{\partial \theta}$$

- w.r.t $\mathbf{U}$:

$$\frac{\partial J_t}{\partial \mathbf{U}} = \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{U}} = (\hat{\mathbf{y}}_t - \mathbf{y}_t)\mathbf{s}_t^\intercal$$

- w.r.t $\mathbf{W}$ ($\mathbf{W}_{in}, \mathbf{W}_{rec}$):

$$\frac{\partial J_t}{\partial \mathbf{W}} = \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{s}_t} \frac{\partial \mathbf{s}_t}{\partial \mathbf{W}} = \sum_{k=1}^{t} \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{s}_t} \frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_k} \frac{\partial \mathbf{s}_k}{\partial \mathbf{W}}$$

Werbos (1990)

Vanishing and Exploding Gradient

Why do we have to care about it?

Exploding Gradient

- Norm of gradient increases exponentially
- Overflow when calc gradient

Vanishing Gradient

- Norm of gradient decrease exponentially (to 0)
- Can NOT learn long-term dependencies

Deep FNNs and RNNs are easy to stuck on these problems.

- product of matrices is similar to product of real numbers can to go zero or infinity.

$$\lim_{k \to \infty} \lambda^k = \begin{cases} 0 & \text{if } \lambda < 1 \\ \infty & \text{if } \lambda > 1 \end{cases}$$

Bengio et al. (1994), Pascanu et al. (2013)

Vanishing and Exploding Gradient - WHY

- Similar hidden state function

$$\mathbf{s}_t = F(\mathbf{s}_{t-1}, \mathbf{x}_t, \mathbf{W}) = \mathbf{W}_{rec} f(\mathbf{s}_{t-1}) + \mathbf{W}_{in}\mathbf{x}_t + \mathbf{b}_s$$

- Gradient w.r.t $\mathbf{W}$:

$$\frac{\partial J}{\partial \mathbf{W}} = \sum_{t=1}^{T} \frac{\partial J_t}{\partial \mathbf{W}}$$

- At step t:

$$\frac{\partial J_t}{\partial \mathbf{W}} = \sum_{k=1}^{t} \frac{\partial J_t}{\partial \mathbf{s}_t} \frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_k} \frac{\partial \mathbf{s}_k}{\partial \mathbf{W}}$$

- Error from step t back to k:

$$\frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_k} = \prod_{j=k}^{t-1} \frac{\partial \mathbf{s}_{j+1}}{\partial \mathbf{s}_j} = \prod_{j=k}^{t-1} \mathbf{W}_{rec}^{\intercal} \mathrm{diag}\big(f'(\mathbf{s}_j)\big)$$

Bengio et al. (1994), Pascanu et al. (2013)

Vanishing and Exploding Gradient - WHY

- $\mathbf{s}_k$ is vector, so $\dfrac{\partial \mathbf{s}_{j+1}}{\partial \mathbf{s}_j}$ is a Jacobian matrix

Let:

- $\gamma \in \mathbb{R}, \left\| \operatorname{diag}\left(f'(\mathbf{s}_j)\right) \right\| \leq \gamma$
- $\lambda_1 = \max\left(|eigenvalues(\mathbf{W}_{rec})|\right)$

We have:

$$\forall j, \left\| \frac{\partial \mathbf{s}_{j+1}}{\partial \mathbf{s}_j} \right\| \leq \|\mathbf{W}_{rec}^\intercal\| \left\| \operatorname{diag}\left(f'(\mathbf{s}_j)\right) \right\| \leq \lambda_1 \gamma$$

Let $\eta = \lambda_1 \gamma$ and $l = t - k$:

$$\frac{\partial J_t}{\partial \mathbf{s}_t} \frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_k} = \frac{\partial J_t}{\partial \mathbf{s}_t} \prod_{j=k}^{t-1} \frac{\partial \mathbf{s}_{j+1}}{\partial \mathbf{s}_j} \leq \eta^l \frac{\partial J_t}{\partial \mathbf{s}_t}$$

Bengio et al. (1994), Pascanu et al. (2013)

Vanishing and Exploding Gradient - WHY

$$\frac{\partial J_t}{\partial \mathbf{s}_t}\frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_k} \le \eta^l \frac{\partial J_t}{\partial \mathbf{s}_t}$$

With $(t-k)$ is large (*long-term dependencies*) :

- $\lambda_1 < \dfrac{1}{\gamma}$ or $\eta < 1$: *sufficient* condition for vanishing gradient problem
- $\lambda_1 > \dfrac{1}{\gamma}$ or $\eta > 1$: *neccessary* condition for exploding gradient problem

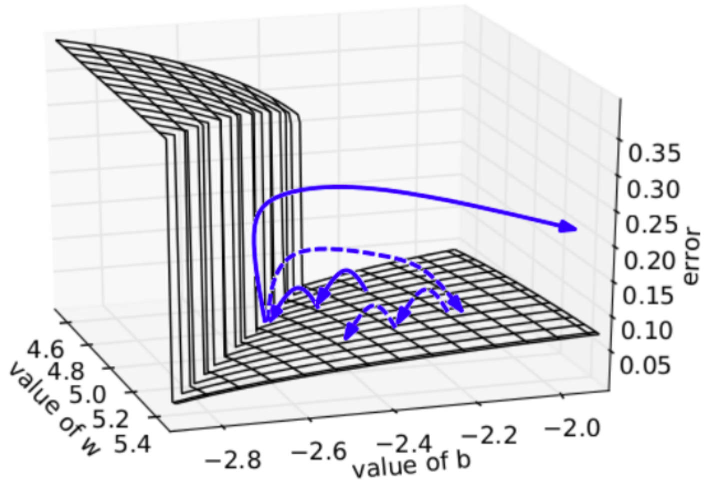
E.x, gradient will shrink to zero when:

- $\lambda_1 < 1$ if f is tanh because $\gamma = 1$
- $\lambda_1 < 4$ if f is sigmoid because $\gamma = 0.25$

Bengio et al. (1994), Pascanu et al. (2013)

Gradient Clipping

- Solution to exploding gradient problem: Rescale gradients



Error surface of a single hidden unit recurrent network

Where:

- Solid lines: standard gradient descent
- Dashed lines: rescaled gradient descent

Pascanu et al. (2013)

Gradient Clipping

- Add `threshold` hyper-parameter to clip norm of gradients

$$
\begin{aligned}
&\text{— — — — — — — — — — — — — — — — — — —}\\
&\hat{g} = \frac{\partial J}{\partial \mathbf{W}}\\
&\textbf{if} \quad \|\hat{g}\| \geq threshold \quad \textbf{then}\\
&\qquad \hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|}\hat{g}\\
&\textbf{end if}\\
&\text{— — — — — — — — — — — — — — — — — — —}
\end{aligned}
$$

- Usually, threshold $\in [1, 5]$
- Simple, Effective

Pascanu et al. (2013)

Long Short-Term Memory - LSTM

- Constant Error Flow of Identity Relationship doesn't decay:

$$\mathbf{s}_t = \mathbf{s}_{t-1} + f(\mathbf{x}_t) \implies \frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_{t-1}} = 1$$

- Key idea: Use *Constant Error Carousel* - **CEC** to prevent from gradient decay

  - **Memory Cell** $\mathbf{c}_t$: indentity relationship
  - Compute new state by difference from before time step *[s]*

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

  $\mathbf{h}_t$ *is the output at time step t*

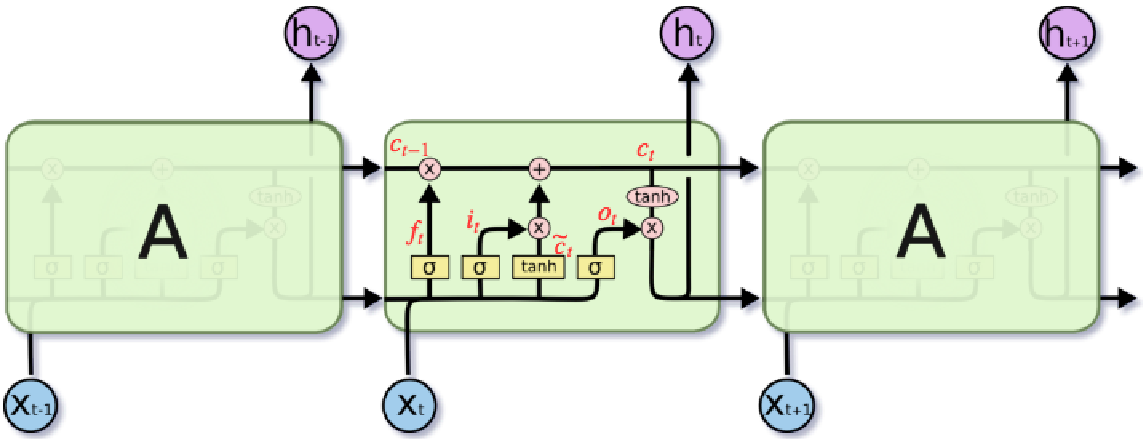
- Weights conflict:

  - Input Weights: Same weights for *"write operations"*
  - Output Weights: Same weights for *"read operations"*

==> Use **Gates Units** to control conflicting

Hochreiter (1997)

LSTM



- Gate Units:
  - sigmoid function $\sigma \in [0, 1]$ controls how much info can be through
- Gate Types:
  - Forget Gate $\mathbf{f}_t$ (*Gers et al. (1999)*)
  - Input Gate $\mathbf{i}_t$
  - Output Gate $\mathbf{o}_t$
- CEC: center $\oplus$ act as linear function

Figure: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM - Forward

- Forget Gate:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

- Input Gate:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

- Output Gate:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

- New State:

$$\begin{aligned}\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)\\ \mathbf{c}_t &= \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{c}}_t\end{aligned}$$

- Cell's Output:

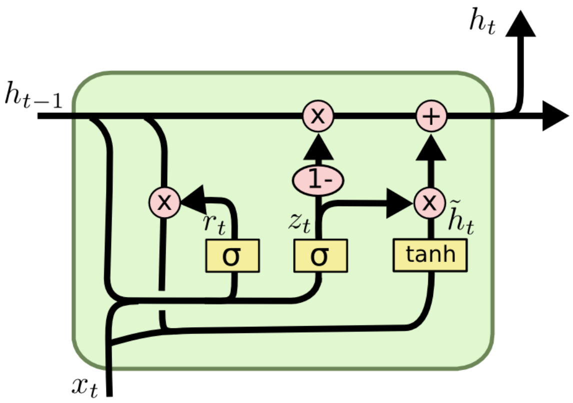
$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{c}_t)$$

Hochreiter (1997)

LSTM - Backward

- Cell's Output: $\delta h_t = \partial J_t / \partial \mathbf{h}_t$
- Output Gate: $\delta o_t = \delta h_t * \tanh(\mathbf{c}_t)$
  - Compute: $\delta W_o^{(t)}, \delta b_o^{(t)}$
- New State: $\delta c_t = \color{blue}{\delta c_t} + \delta h_t * \delta o_t * (1 - \tanh^2(\mathbf{c}_t))$
- Previous State: $\delta c_{t-1} = \delta c_t * \mathbf{f}_t$
- Input Gate: $\delta i_t = \delta c_t * \tilde{\mathbf{c}}_t$
  - Compute: $\delta W_i^{(t)}, \delta b_i^{(t)}$
- Forget Gate: $\delta f_t = \delta c_t * \mathbf{c}_{t-1}$
  - Compute: $\delta W_f^{(t)}, \delta b_f^{(t)}$
- External Input: $\delta \tilde{c}_t = \delta c_t * \mathbf{i}_t$
  - Compute: $\delta W_c^{(t)}, \delta b_c^{(t)}$

Gated Reccurent Unit - GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

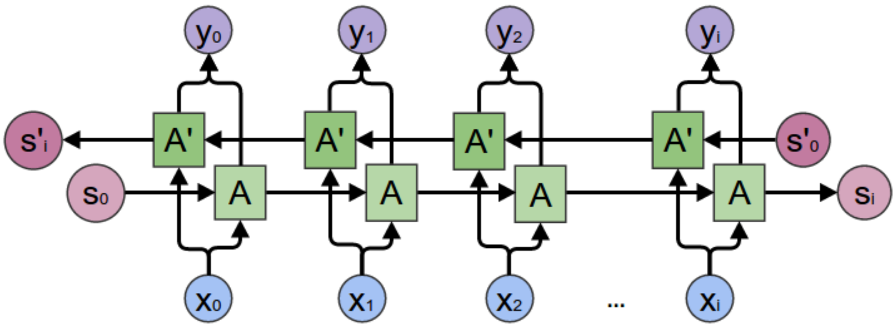
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Cell State h_t
  - Cell State & Hidden State
- Update Gate z_t
  - Forget Gate & Input Gate
- Reset Gate r_t

Cho et al. (2014), Figure: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Bidirectional RNNs



- Previous Dependencies (left → right):

$$\mathbf{s}_t = f(\mathbf{W}_{in}\mathbf{x}_t + \mathbf{W}_{rec}\mathbf{s}_t + \mathbf{b}_s)$$

- Following Dependencies (right → left):

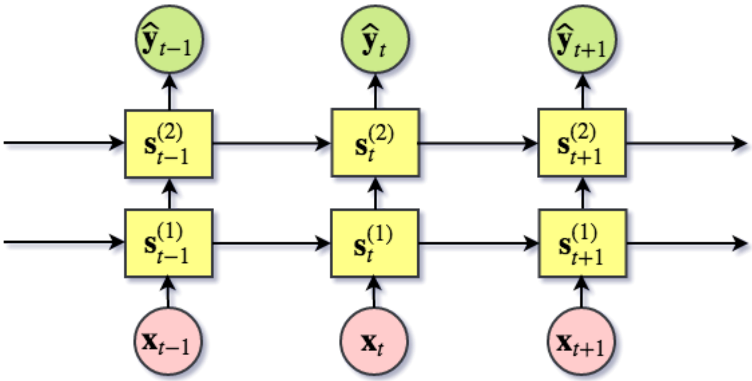
$$\mathbf{s}'_t = f(\mathbf{W}'_{in}\mathbf{x}_t + \mathbf{W}'_{rec}\mathbf{s}_t + \mathbf{b}'_s)$$

- Output:

$$\mathbf{y}_t = g(U[\mathbf{s}_t, \mathbf{s}'_t] + \mathbf{b}_y)$$

Figure: https://colah.github.io/posts/2015-09-NN-Types-FP/

Deep RNNs



- Layer 0 (Input):

$$\mathbf{s}_t^{(0)} = \mathbf{x}_t$$

- Layer $l = \overline{1, L}$:

$$\mathbf{s}_t^{(l)} = f(\mathbf{W}_{in}^{(l)}\mathbf{s}_t^{(l-1)} + \mathbf{W}_{rec}^{(l)}\mathbf{s}_{t-1}^{(l)} + \mathbf{b}_s^{(l)})$$

- Output:

$$\hat{\mathbf{y}}_t = g(\mathbf{U}\mathbf{s}_t^{(L)} + \mathbf{b}_y)$$

Summary

- RNNs
    - Variable-length In/Output
    - Train with BPPT
    - Vanishing & exploding gradient problem
- Gradient Clipping
    - Rescale gradients to prevent from exploding gradient problem
- LSTM
    - Memory Cell: Keep linear relationship between state
    - Gate Units: control through info with sigmoid function $\sigma \in [0, 1]$
    - Time step lags > 1000
    - Local in space and time: $\Theta(1)$ per step and weight
- GRU
    - Merge cell state and hidden state
    - Combine forget gate and input gate into update gate
- RNNs variants: Bidirectional RNNs, Deep RNNs

References

- [1] *Hopfield (1982).* Neural networks and physical systems with emergent collective computational abilities
- [2] *Rumelhart et al. (1986a).* Learning representations by back-propagation errors
- [3] *Jordan (1986).* Serial order: A parallel distributed processing approach
- [4] *Elman (1990).* Finding structure in time
- [5] *Werbos (1990).* Backpropagation Through Time: What It Does and How to Do It
- [6] *Bengio et al. (1994).* Learning Long-Term Dependencies with Gradient Descent is Difficult
- [7] *Pascanu et al. (2013).* On the difficulty of training Recurrent Neural Networks
- [8] *Hochreiter et al. (1997).* Long Short-Term Memory
- [9] *Greff et al. (2017).* LSTM: A search space odyssey
- [10] *Jozefowics et al. (2015).* An Empirical Exploration of Recurrent Network Architectures
- [11] *Cho et al. (2014).* Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation