Recurrent Neural Networks

Changyou Chen

Department of Computer Science and Engineering
Universitpy at Buffalo, SUNY
changyou@buffalo.edu

March 26, 2019

Character-level Language Model: Trained on Shakespeare's "The Sonnets"

random generate

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

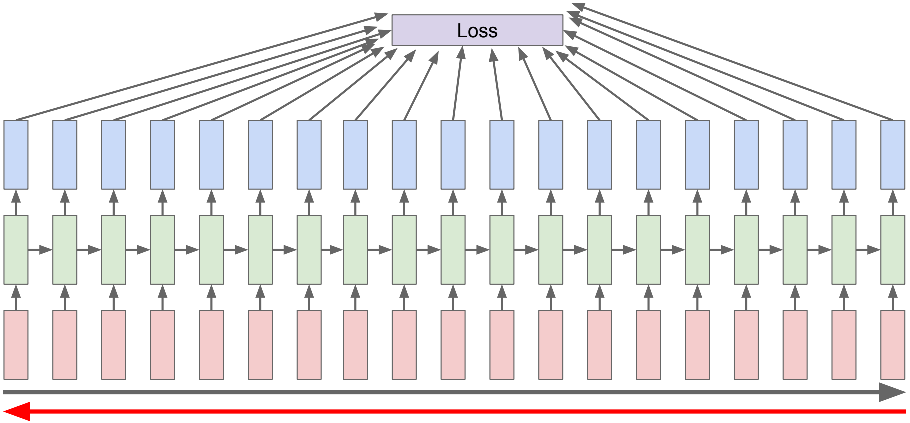
```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Learning in RNN

How to do BP?

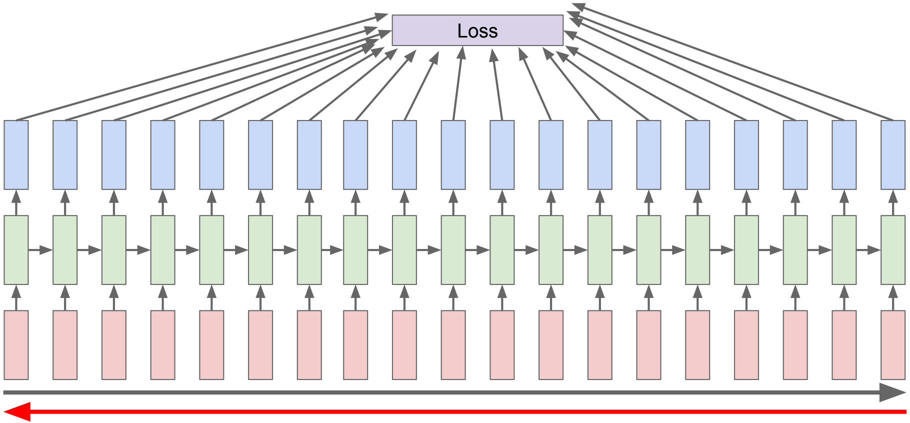
Backpropagation through Time

How is a weight correlated to local loss?



Backpropagation through Time

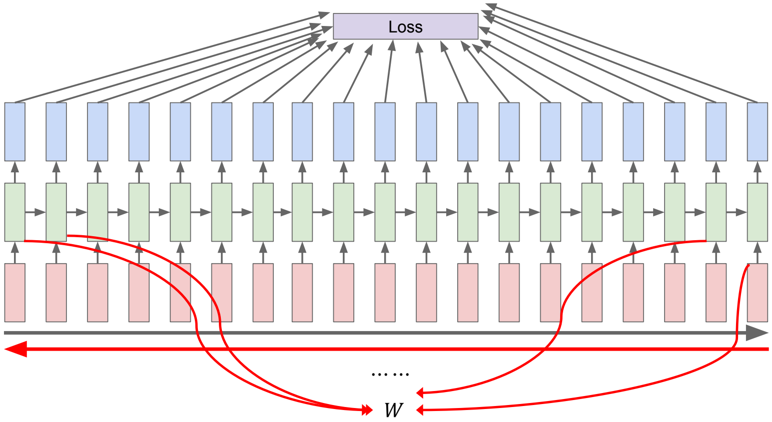
How is a weight correlated to local loss?



Forward though entire sequence to compute loss, then backward through entire sequence to compute gradient.

Backpropagation through Time

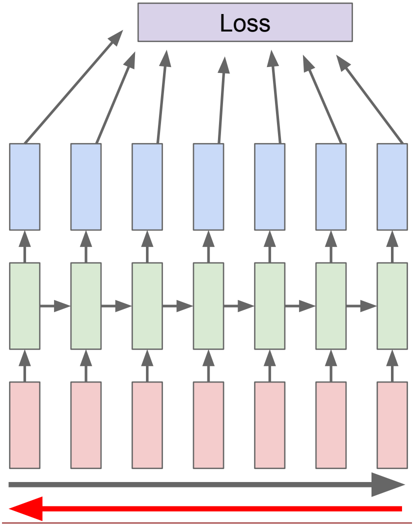
How is a weight correlated to local loss?



Forward though entire sequence to compute loss, then backward through entire sequence to compute gradient.

⇒ Too computationally expensive!

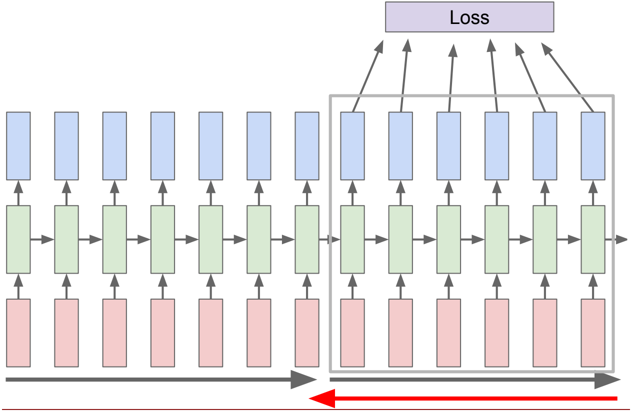
Backpropagation through Time



Run forward and backward through chunks of the sequence instead of whole sequence:

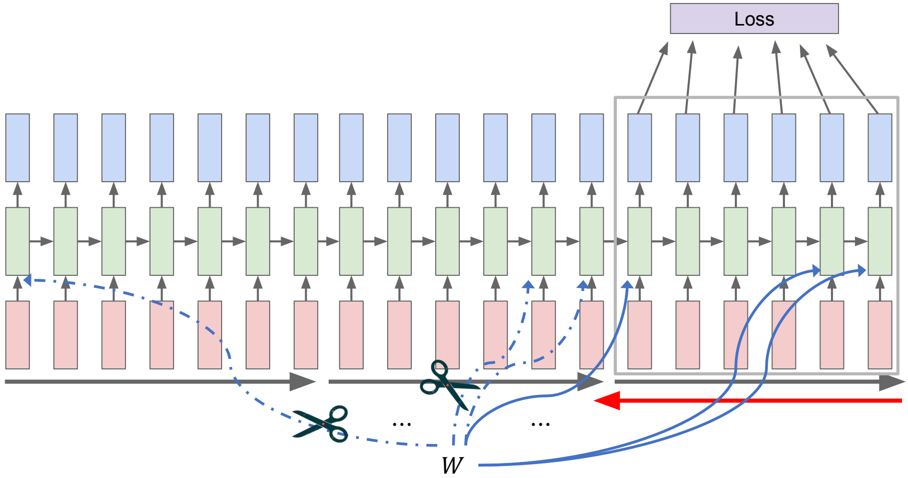
- greatly reduces computational time without too much loss in accuracy

Backpropagation through Time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps.

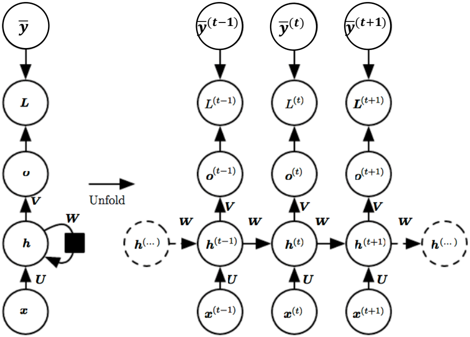
Backpropagation through Time



Example: Computing Gradients by BPTT

- Consider the following RNN model:

$$\mathbf{a}^{(t)} = \mathbf{W}\,\mathbf{h}^{(t-1)} + \mathbf{U}\,\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\,\mathbf{h}^{(t)} + \mathbf{c}$$

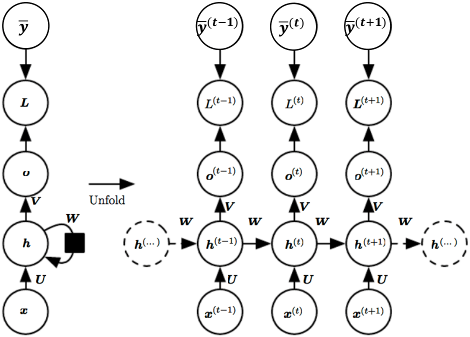


- For each node N, we need to compute the gradient $\nabla_N \mathcal{L}$ recursively, based on gradients at nodes that follow it in the graph.
- Assume the outputs $o^{(t)}$ are used as the arguments to the softmax function to obtain the vector $\hat{y}^{(t)}$ of probabilities over the output.

Example: Computing Gradients by BPTT

- Consider the following RNN model:

$$\mathbf{a}^{(t)} = \mathbf{W}\,\mathbf{h}^{(t-1)} + \mathbf{U}\,\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\,\mathbf{h}^{(t)} + \mathbf{c}$$

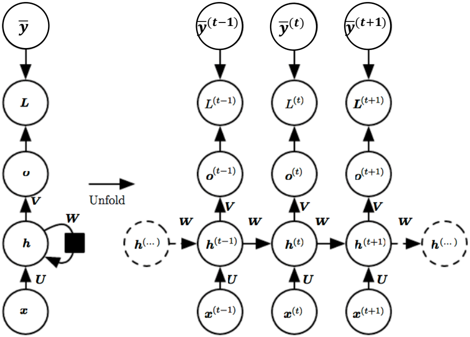


- For each node N, we need to compute the gradient $\nabla_N \mathcal{L}$ recursively, based on gradients at nodes that follow it in the graph.
- Assume the outputs $o^{(t)}$ are used as the arguments to the softmax function to obtain the vector $\hat{y}^{(t)}$ of probabilities over the output.

Example: Computing Gradients by BPTT

- Consider the following RNN model:

$$\mathbf{a}^{(t)} = \mathbf{W}\,\mathbf{h}^{(t-1)} + \mathbf{U}\,\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\,\mathbf{h}^{(t)} + \mathbf{c}$$



- For each node N, we need to compute the gradient $\nabla_N \mathcal{L}$ recursively, based on gradients at nodes that follow it in the graph.
- Assume the outputs $o^{(t)}$ are used as the arguments to the softmax function to obtain the vector $\hat{y}^{(t)}$ of probabilities over the output.

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} \ ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

$$(\nabla_{\mathbf{o}^{(t)}} \mathcal{L})_i = \frac{\partial \mathcal{L}}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \overset{\text{softmax property}}{=}$$

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

$$(\nabla_{\mathbf{o}^{(t)}} \mathcal{L})_i = \frac{\partial \mathcal{L}}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \overset{\text{softmax property}}{=} \hat{y}_i^{(t)} - \bar{y}_i^{(t)} .$$

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} \ ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

$$(\nabla_{\mathbf{o}^{(t)}} \mathcal{L})_i = \frac{\partial \mathcal{L}}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \overset{\text{softmax property}}{=} \hat{y}_i^{(t)} - \bar{y}_i^{(t)} \ .$$

$$\Rightarrow \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \hat{\mathbf{y}}^{(t)} - \bar{\mathbf{y}}^{(t)}$$

Example: Computing Gradients by BPTT

$$\mathbf{a}^{(t)} = \mathbf{W}\,\mathbf{h}^{(t-1)} + \mathbf{U}\,\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\,\mathbf{h}^{(t)} + \mathbf{c}$$

1. Iterate backwards. At the final time step t the gradient is (two paths for $\mathbf{h}^t$):

$$\nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \left(\frac{\partial\,\mathbf{h}^{(t+1)}}{\partial\,\mathbf{h}^{(t)}}\right)^T (\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}) + \left(\frac{\partial\,\mathbf{o}^{(t)}}{\partial\,\mathbf{h}^{(t)}}\right)^T (\nabla_{\mathbf{o}^{(t)}}\mathcal{L})$$
$$=$$

Example: Computing Gradients by BPTT

$$\mathbf{a}^{(t)} = \mathbf{W}\,\mathbf{h}^{(t-1)} + \mathbf{U}\,\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\,\mathbf{h}^{(t)} + \mathbf{c}$$

1. Iterate backwards. At the final time step t the gradient is (two paths for $\mathbf{h}^t$):

$$\nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \left(\frac{\partial\,\mathbf{h}^{(t+1)}}{\partial\,\mathbf{h}^{(t)}}\right)^T (\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}) + \left(\frac{\partial\,\mathbf{o}^{(t)}}{\partial\,\mathbf{h}^{(t)}}\right)^T (\nabla_{\mathbf{o}^{(t)}}\mathcal{L})$$
$$= \mathbf{W}^T \operatorname{diag}\left(1 - \tanh^2(\mathbf{a}^{(t+1)})\right) (\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}) + \mathbf{V}^T (\nabla_{\mathbf{o}^{(t)}}\mathcal{L})$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\begin{aligned}
\nabla_{\mathbf{c}}\mathcal{L} &= \\
\nabla_{\mathbf{b}}\mathcal{L} &= \\
\nabla_{\mathbf{V}}\mathcal{L} &= \\
\nabla_{\mathbf{W}}\mathcal{L} &= \\
\nabla_{\mathbf{U}}\mathcal{L} &=
\end{aligned}$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$
\begin{aligned}
\nabla_{\mathbf{c}}\mathcal{L} &= \sum_t \left(\frac{\partial\, \mathbf{o}^{(t)}}{\partial\, \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} \\
\nabla_{\mathbf{b}}\mathcal{L} &= \\
\nabla_{\mathbf{V}}\mathcal{L} &= \\
\nabla_{\mathbf{W}}\mathcal{L} &= \\
\nabla_{\mathbf{U}}\mathcal{L} &=
\end{aligned}
$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$
\begin{aligned}
\nabla_{\mathbf{c}}\mathcal{L} &= \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L} \\
\nabla_{\mathbf{b}}\mathcal{L} &= \\
\nabla_{\mathbf{V}}\mathcal{L} &= \\
\nabla_{\mathbf{W}}\mathcal{L} &= \\
\nabla_{\mathbf{U}}\mathcal{L} &=
\end{aligned}
$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$
\begin{aligned}
\nabla_{\mathbf{c}}\mathcal{L} &= \sum_t \left(\frac{\partial\, \mathbf{o}^{(t)}}{\partial\, \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L} \\
\nabla_{\mathbf{b}}\mathcal{L} &= \sum_t \left(\frac{\partial\, \mathbf{h}^{(t)}}{\partial\, \mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} \\
\nabla_{\mathbf{V}}\mathcal{L} &= \\
\nabla_{\mathbf{W}}\mathcal{L} &= \\
\nabla_{\mathbf{U}}\mathcal{L} &=
\end{aligned}
$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_t \left(\frac{\partial\,\mathbf{o}^{(t)}}{\partial\,\mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_t \left(\frac{\partial\,\mathbf{h}^{(t)}}{\partial\,\mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \sum_t \mathrm{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \nabla_{\mathbf{h}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{V}}\mathcal{L} =$$

$$\nabla_{\mathbf{W}}\mathcal{L} =$$

$$\nabla_{\mathbf{U}}\mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_t \left(\frac{\partial\, \mathbf{o}^{(t)}}{\partial\, \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_t \left(\frac{\partial\, \mathbf{h}^{(t)}}{\partial\, \mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \sum_t \mathrm{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \nabla_{\mathbf{h}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}}\right)^T \nabla_{\mathbf{V}} o^{(t)}$$

$$\nabla_{\mathbf{W}}\mathcal{L} =$$

$$\nabla_{\mathbf{U}}\mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_t \left(\frac{\partial\,\mathbf{o}^{(t)}}{\partial\,\mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_t \left(\frac{\partial\,\mathbf{h}^{(t)}}{\partial\,\mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \sum_t \mathrm{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \nabla_{\mathbf{h}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_t \left(\frac{\partial\mathcal{L}}{\partial o^{(t)}}\right)^T \nabla_{\mathbf{V}} o^{(t)} = \sum_t \left(\nabla_{\mathbf{o}^{(t)}}\mathcal{L}\right) \mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}}\mathcal{L} =$$

$$\nabla_{\mathbf{U}}\mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_t \left(\frac{\partial\, \mathbf{o}^{(t)}}{\partial\, \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_t \left(\frac{\partial\, \mathbf{h}^{(t)}}{\partial\, \mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \sum_t \mathrm{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \nabla_{\mathbf{h}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}}\right)^T \nabla_{\mathbf{V}} o^{(t)} = \sum_t \left(\nabla_{\mathbf{o}^{(t)}}\mathcal{L}\right) \mathbf{h}^{(t)T}$$

$$\nabla_{\mathbf{W}}\mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}}\right)^T \nabla_{\mathbf{W}} h^{(t)}$$

$$\nabla_{\mathbf{U}}\mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_t \left(\frac{\partial\, \mathbf{o}^{(t)}}{\partial\, \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_t \left(\frac{\partial\, \mathbf{h}^{(t)}}{\partial\, \mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \sum_t \mathrm{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \nabla_{\mathbf{h}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}}\right)^T \nabla_{\mathbf{V}} o^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}}\mathcal{L})\, \mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}}\mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}}\right)^T \nabla_{\mathbf{W}} h^{(t)} = \sum_t \mathrm{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) (\nabla_{\mathbf{h}^{(t)}}\mathcal{L})\, \mathbf{h}^{(t-1)\,T}$$

$$\nabla_{\mathbf{U}}\mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_t \left(\frac{\partial\, \mathbf{o}^{(t)}}{\partial\, \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_t \left(\frac{\partial\, \mathbf{h}^{(t)}}{\partial\, \mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \sum_t \mathrm{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \nabla_{\mathbf{h}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}}\right)^T \nabla_{\mathbf{V}} o^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}}\mathcal{L})\, \mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}}\mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}}\right)^T \nabla_{\mathbf{W}} h^{(t)} = \sum_t \mathrm{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) (\nabla_{\mathbf{h}^{(t)}}\mathcal{L})\, \mathbf{h}^{(t-1)\,T}$$

$$\nabla_{\mathbf{U}}\mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}}\right) \nabla_{\mathbf{U}^{(t)}} h^{(t)}$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_t \left(\frac{\partial\,\mathbf{o}^{(t)}}{\partial\,\mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_t \left(\frac{\partial\,\mathbf{h}^{(t)}}{\partial\,\mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \sum_t \operatorname{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \nabla_{\mathbf{h}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_t \left(\frac{\partial\mathcal{L}}{\partial\,\mathbf{o}^{(t)}}\right)^T \nabla_{\mathbf{V}}\,\mathbf{o}^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}}\mathcal{L})\,\mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}}\mathcal{L} = \sum_t \left(\frac{\partial\mathcal{L}}{\partial h^{(t)}}\right)^T \nabla_{\mathbf{W}} h^{(t)} = \sum_t \operatorname{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) (\nabla_{\mathbf{h}^{(t)}}\mathcal{L})\,\mathbf{h}^{(t-1)\,T}$$

$$\nabla_{\mathbf{U}}\mathcal{L} = \sum_t \left(\frac{\partial\mathcal{L}}{\partial h^{(t)}}\right) \nabla_{\mathbf{U}^{(t)}} h^{(t)} = \sum_t \operatorname{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) (\nabla_{\mathbf{h}^{(t)}}\mathcal{L})\,\mathbf{x}^{(t)\,T}$$

Summary: Gradients in RNN

$$\nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \hat{\mathbf{y}}^{(t)} - \bar{\mathbf{y}}^{(t)}$$

$$\nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \mathbf{W}^T \,\mathbf{diag}\left(1 - \tanh^2(\mathbf{a}^{(t+1)})\right)(\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}) + \mathbf{V}^T(\nabla_{\mathbf{o}^{(t)}}\mathcal{L})$$

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_t \left(\frac{\partial\,\mathbf{o}^{(t)}}{\partial\,\mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}}\mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_t \left(\frac{\partial\,\mathbf{h}^{(t)}}{\partial\,\mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \sum_t \mathbf{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right)\nabla_{\mathbf{h}^{(t)}}\mathcal{L}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_t \left(\frac{\partial\mathcal{L}}{\partial\,\mathbf{o}^{(t)}}\right)^T \nabla_{\mathbf{V}}\,\mathbf{o}^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}}\mathcal{L})\,\mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}}\mathcal{L} = \sum_t \left(\frac{\partial\mathcal{L}}{\partial h^{(t)}}\right)^T \nabla_{\mathbf{W}} h^{(t)} = \sum_t \mathbf{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right)(\nabla_{\mathbf{h}^{(t)}}\mathcal{L})\,\mathbf{h}^{(t-1)\,T}$$

$$\nabla_{\mathbf{U}}\mathcal{L} = \sum_t \left(\frac{\partial\mathcal{L}}{\partial h^{(t)}}\right) \nabla_{\mathbf{U}^{(t)}} h^{(t)} = \sum_t \mathbf{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right)(\nabla_{\mathbf{h}^{(t)}}\mathcal{L})\,\mathbf{x}^{(t)\,T}$$

Request

- Please do course evaluation!

Quiz

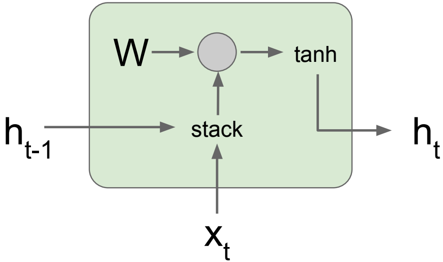
- You have an input volume that is $63 \times 63 \times 16$, and convolve it with 32 filters that are each $7 \times 7 \times 16$, using a stride of 2 and no padding. 1) What is the output volume? 2) What is the total number of parameters (ignore the bias parameter)?

Long Short Term Memory[1]

[1] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Vanilla RNN Gradient Flow

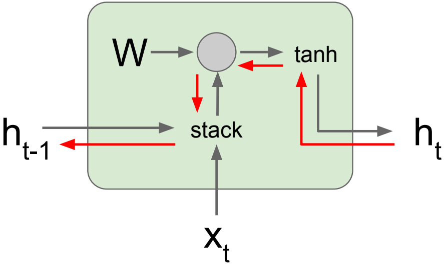
$$
\begin{aligned}
\mathbf{h}_t &= \tanh\left(\mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{W}_{xh}\,\mathbf{x}_t\right) \\
&= \tanh\left(\begin{bmatrix}\mathbf{W}_{hh} & \mathbf{W}_{xh}\end{bmatrix}\right)\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix} \\
&= \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix}\right)
\end{aligned}
$$



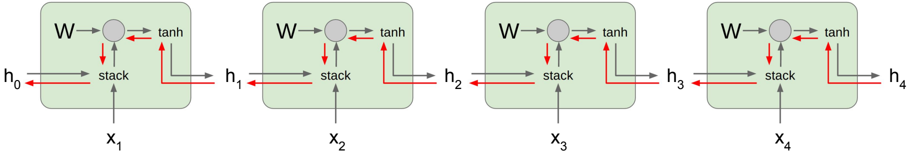
Vanilla RNN Gradient Flow

$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{W}_{xh}\,\mathbf{x}_t\right)$$
$$= \tanh\left(\begin{bmatrix}\mathbf{W}_{hh} & \mathbf{W}_{xh}\end{bmatrix}\right)\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix}$$
$$= \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix}\right)$$

Backpropagation from h_t to h_{t-1} multiplies by W (actually $W_{hh}{}^T$)



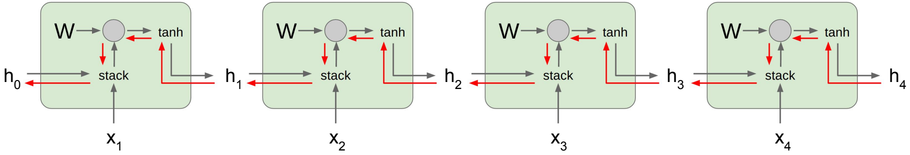
Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow

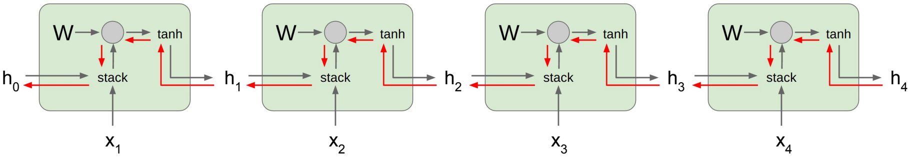


Computing gradient of h_0 involves many factors of W (and repeated tanh)

$$\nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \mathbf{W}^T \operatorname{diag}\left(1 - \tanh^2(\mathbf{a}^{(t+1)})\right)(\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}) + \mathbf{V}^T(\nabla_{\mathbf{o}^{(t)}}\mathcal{L})$$

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



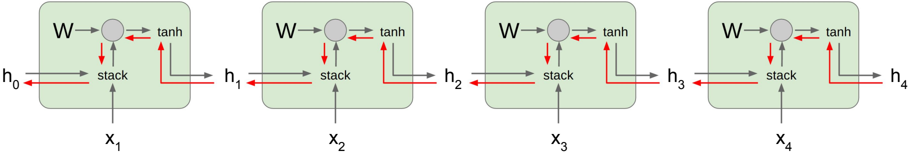
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

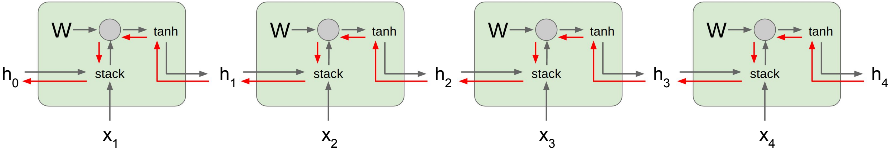
Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Why?

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

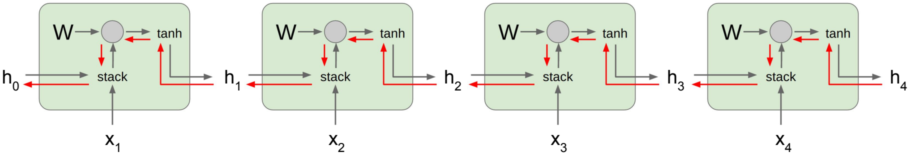
Largest singular value < 1:
Vanishing gradients

Gradient clipping: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

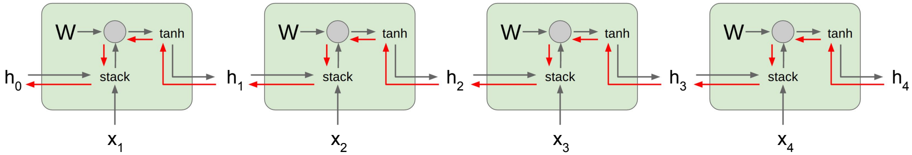
Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

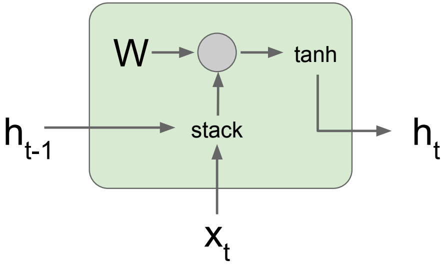
→ Change RNN architecture

Long Short Term Memory!

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN

$$\begin{aligned}
\mathbf{h}_t &= \tanh\left(\mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{W}_{xh}\,\mathbf{x}_t\right) \\
&= \tanh\left(\begin{bmatrix}\mathbf{W}_{hh} & \mathbf{W}_{xh}\end{bmatrix}\right)\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix} \\
&= \tanh\left({\color{red}\mathbf{W}}\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix}\right)
\end{aligned}$$



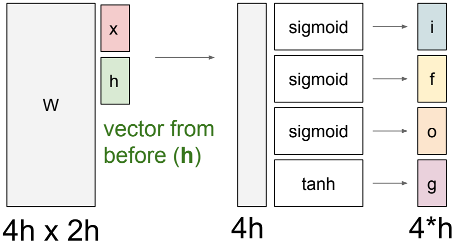
Long Short Term Memory (LSTM)

- **f**: *Forget gate*, whether to erase cell
- **i**: *Input gate*, whether to write to cell
- **g**: *New content*, how much to write to cell
- **o**: *Output gate*, how much to reveal cell

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left[\mathbf{W} \begin{pmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{pmatrix} \right]$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g}$$

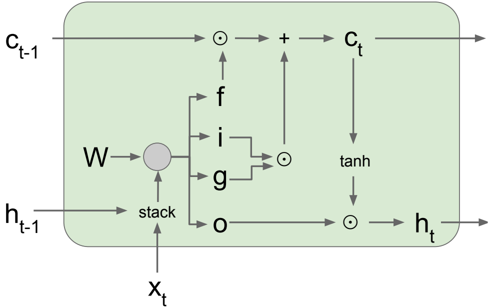
$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$



Hochreiter & Schmidhuber, Neural Computation 1997

Long Short Term Memory (LSTM)

Create an additional internal hidden state $\mathbf{c}_t$ for each time t.



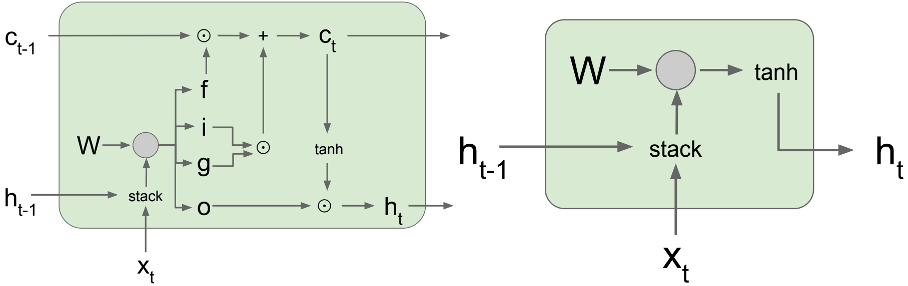
$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left[\mathbf{W} \begin{pmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{pmatrix} \right]$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g}$$

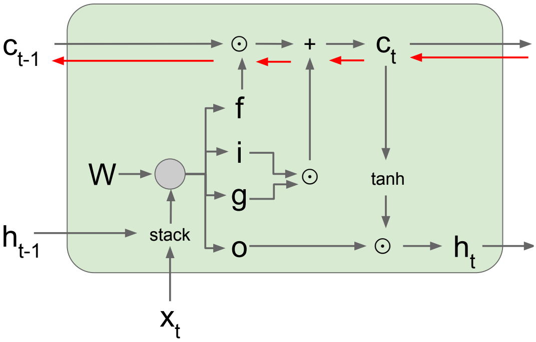
$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

Hochreiter & Schmidhuber, Neural Computation 1997

Long Short Term Memory (LSTM)



Long Short Term Memory: Gradient Flow



Backpropagation from $\mathbf{c}_t$ to $\mathbf{c}_{t-1}$ only elementwise multiplication by $\mathbf{f}$, no direct matrix multiply by $\mathbf{W}$ $\Rightarrow$ no gradient vanishing.

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left[\mathbf{W} \begin{pmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{pmatrix} \right]$$

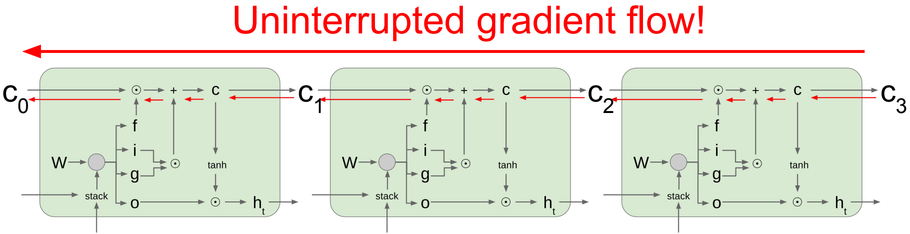
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g}$$

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

Hochreiter & Schmidhuber, Neural Computation 1997

Long Short Term Memory: Gradient Flow

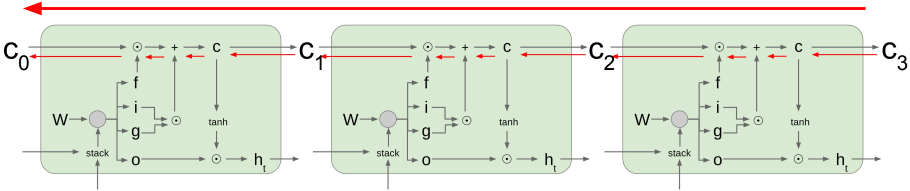
Gradients directly backpropagate through time.



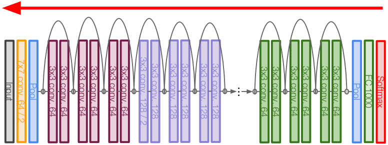
Long Short Term Memory: Gradient Flow

Gradients directly backpropagate through time.

Uninterrupted gradient flow!



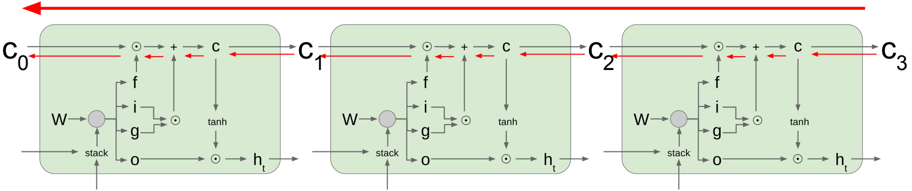
Similar to ResNet!



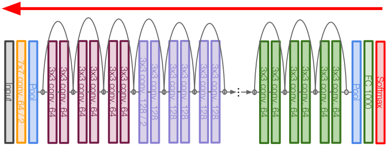
Long Short Term Memory: Gradient Flow

Gradients directly backpropagate through time.

Uninterrupted gradient flow!



Similar to ResNet!



In between:

Highway Networks

$$g = T(x, W_T)$$

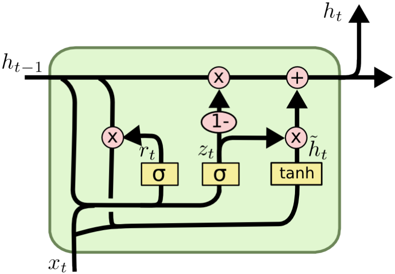
$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, "Highway Networks", ICML DL Workshop 2015

Other RNN Variants

Gated Recurent Unit (GRU) [Cho et al., 2014]

$$\mathbf{r}_t = \sigma\left(\mathbf{W}_{xr}\,\mathbf{x}_t + \mathbf{W}_{hr}\,\mathbf{h}_{t-1} + \mathbf{b}_r\right)$$
$$\mathbf{z}_t = \sigma\left(\mathbf{W}_{xz}\,\mathbf{x}_t + \mathbf{W}_{hz}\,\mathbf{h}_{t-1} + \mathbf{b}_z\right)$$
$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}_{xh}\,\mathbf{x}_t + \mathbf{W}_{hh}\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{b}_h\right)$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \tilde{\mathbf{h}}_{t-1} + (1 - \mathbf{z}_t) \odot \mathbf{h}_t$$



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

Other RNN Variants [Jozefowicz et al., 2015]

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(\mathbf{W}_{xz}\,\mathbf{x}_t + \mathbf{b}_z\right)\\
\mathbf{r}_t &= \sigma\left(\mathbf{W}_{xr}\,\mathbf{x}_t + \mathbf{W}_{hr}\,\mathbf{h}_t + \mathbf{b}_r\right)\\
\mathbf{h}_{t+1} &= \tanh\left(\mathbf{W}_{hh}\left(\mathbf{r}_t \odot \mathbf{h}_t\right) + \tanh(\mathbf{x}_t)\right.\\
&\quad \left. + \mathbf{b}_h\right)\mathbf{z}_t + \mathbf{h}_t \odot (1 - \mathbf{z}_t)
\end{aligned}$$

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(\mathbf{W}_{xz}\,\mathbf{x}_t + \mathbf{W}_{hz}\,\mathbf{h}_t + \mathbf{b}_z\right)\\
\mathbf{r}_t &= \sigma\left(\mathbf{x}_t + \mathbf{b}_r\right)\\
\mathbf{h}_{t+1} &= \tanh\left(\mathbf{W}_{hh}\left(\mathbf{r}_t \odot \mathbf{h}_t\right) + \mathbf{W}_{xh}\,\mathbf{x}_t\right.\\
&\quad \left. + \mathbf{b}_h\right)\mathbf{z}_t + \mathbf{h}_t \odot (1 - \mathbf{z}_t)
\end{aligned}$$

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(\mathbf{W}_{xz}\,\mathbf{x}_t + \mathbf{W}_{hz}\tanh(\mathbf{h}_t) + \mathbf{b}_z\right)\\
\mathbf{r}_t &= \sigma\left(\mathbf{W}_{xr}\,\mathbf{x}_t + \mathbf{W}_{hr}\,\mathbf{h}_t + \mathbf{b}_r\right)\\
\mathbf{h}_{t+1} &= \tanh\left(\mathbf{W}_{hh}\left(\mathbf{r}_t \odot \mathbf{h}_t\right) + \mathbf{W}_{xh}\,\mathbf{x}_t\right.\\
&\quad \left. + \mathbf{b}_h\right)\mathbf{z}_t + \mathbf{h}_t \odot (1 - \mathbf{z}_t)
\end{aligned}$$

Summary

1. RNNs allow a lot of flexibility in architecture design.
2. Vanilla RNNs are simple but don't work very well.
3. Common to use LSTM or GRU: their additive interactions improve gradient flow.
4. Backward flow of gradients in RNN can explode or vanish:
   - exploding is controlled with gradient clipping.
   - vanishing is controlled with additive interactions (LSTM).
5. Better/simpler architectures are a hot topic of current research.
6. Better understanding (both theoretical and empirical) is needed.