Recurrent Architectures

CS60010: Deep Learning

Abir Das

IIT Kharagpur

Mar 17 and 19, 2022

Agenda

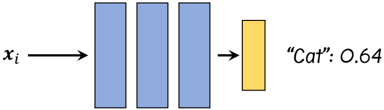
- Understand models involving sequential inputs and/or outputs.
- Using recurrent neural networks [RNNs] to deal with sequential inputs/outputs
- From RNNs to LSTMs.

Resources

§ CS W182 course by Sergey Levine at UC Berkeley. [Link] [Lecture 10]

What if we have variable size inputs?

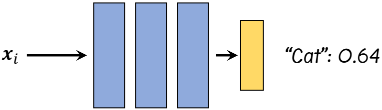
- Before,



Source: CS W182 course, Sergey Levine, UC Berkeley

What if we have variable size inputs?

§ Before,



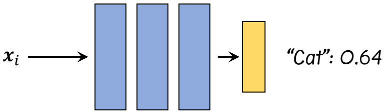
§ Now,

- $\mathbf{x}_1 = (\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \mathbf{x}_{1,3}, \mathbf{x}_{1,4})$
- $\mathbf{x}_2 = (\mathbf{x}_{2,1}, \mathbf{x}_{2,2}, \mathbf{x}_{2,3})$
- $\mathbf{x}_3 = (\mathbf{x}_{3,1}, \mathbf{x}_{3,2}, \mathbf{x}_{3,3}, \mathbf{x}_{3,4}, \mathbf{x}_{3,5})$

Source: CS W182 course, Sergey Levine, UC Berkeley

What if we have variable size inputs?

§ Before,



§ Now,

- $\mathbf{x}_1 = (\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \mathbf{x}_{1,3}, \mathbf{x}_{1,4})$
- $\mathbf{x}_2 = (\mathbf{x}_{2,1}, \mathbf{x}_{2,2}, \mathbf{x}_{2,3})$
- $\mathbf{x}_3 = (\mathbf{x}_{3,1}, \mathbf{x}_{3,2}, \mathbf{x}_{3,3}, \mathbf{x}_{3,4}, \mathbf{x}_{3,5})$

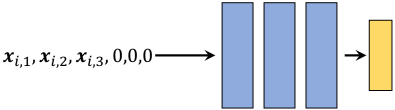
§ Example,

- classifying sentiment for a phrase (sequence of words)
- classifying the activity in a video (sequence of images)

Source: CS W182 course, Sergey Levine, UC Berkeley

Simple Idea

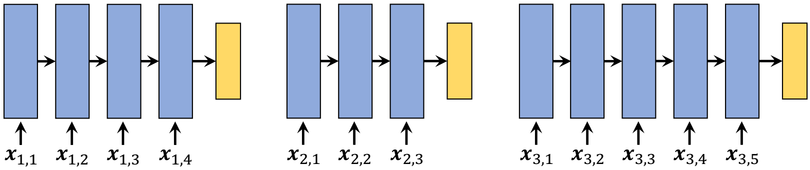
- Now,
  - $\mathbf{x}_1 = (\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \mathbf{x}_{1,3}, \mathbf{x}_{1,4})$
  - $\mathbf{x}_2 = (\mathbf{x}_{2,1}, \mathbf{x}_{2,2}, \mathbf{x}_{2,3})$
  - $\mathbf{x}_3 = (\mathbf{x}_{3,1}, \mathbf{x}_{3,2}, \mathbf{x}_{3,3}, \mathbf{x}_{3,4}, \mathbf{x}_{3,5})$
- Simple Idea: Zeropad up to length of longest sequence.



- Very simple, but doesn't scale well for very long sequences.

Source: CS W182 course, Sergey Levine, UC Berkeley

One Input per Layer?



- Each layer,

$$\bar{\mathbf{h}}^{(l-1)} = \begin{bmatrix} \mathbf{h}^{(l-1)} \\ \mathbf{x}_{i,t} \end{bmatrix}$$

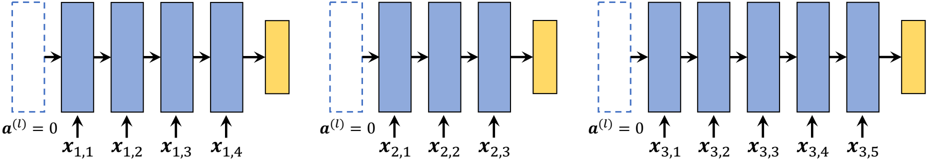
$$\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\bar{\mathbf{h}}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{h}^{(l)} = g\big(\mathbf{a}^{(l)}\big)$$

- Obvious question: What happens to the missing layers?

Source: CS W182 course, Sergey Levine, UC Berkeley

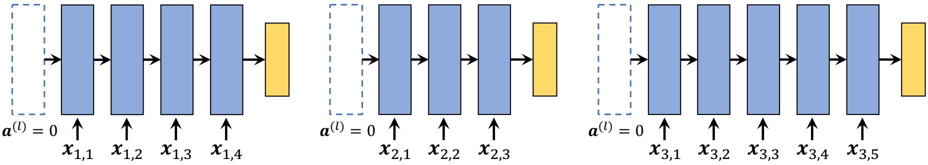
Zero Prior Activation



- All activations prior to the first word/layer are assummed to be zero.
- More efficient than always 0-padding the sequence up to max length. Each layer is much smaller than the giant first layer needed in case the whole sequence with zero padding is fed to the first layer.
- Later layers get more training.

Source: CS W182 course, Sergey Levine, UC Berkeley

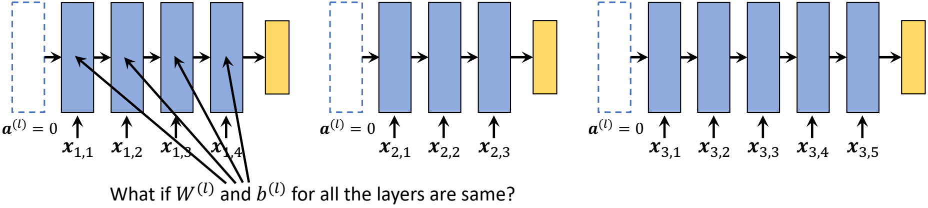
Zero Prior Activation



- All activations prior to the first word/layer are assumed to be zero.
- More efficient than always 0-padding the sequence up to max length. Each layer is much smaller than the giant first layer needed in case the whole sequence with zero padding is fed to the first layer.
- Later layers get more training.
- Total number of weight matrices increases with max sequence length!

Source: CS W182 course, Sergey Levine, UC Berkeley

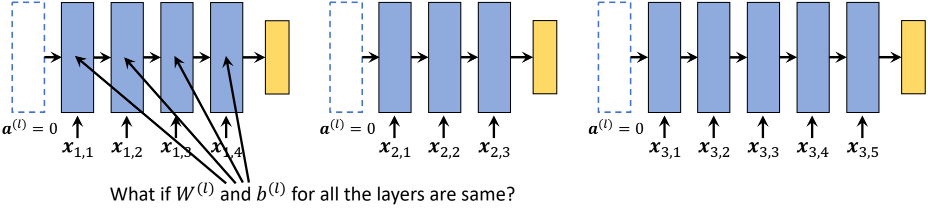
Can We Share Weight Matrices?



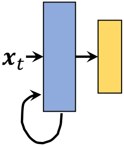
- This is called a *Recurrent Neural Network* (RNN).

Source: CS W182 course, Sergey Levine, UC Berkeley

Can We Share Weight Matrices?

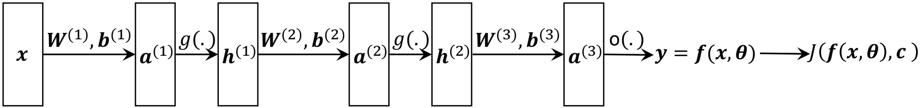


- This is called a *Recurrent Neural Network* (RNN).
- Another notion: a recurrent neural network extends a standard neural network along the time dimension.

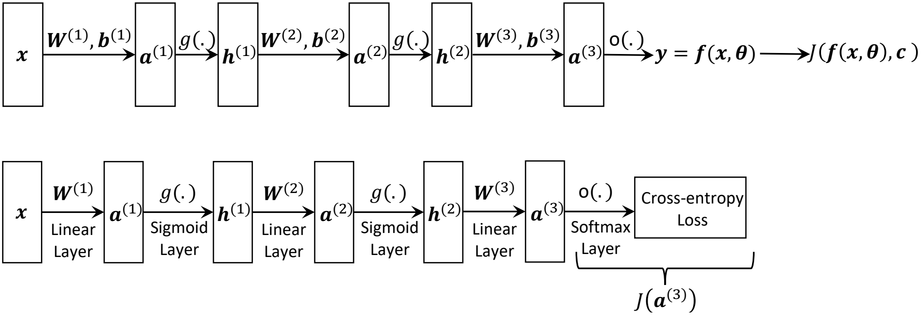


Source: CS W182 course, Sergey Levine, UC Berkeley

Backpropagation Revisit

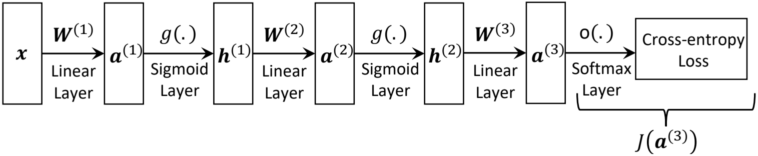


Backpropagation Revisit



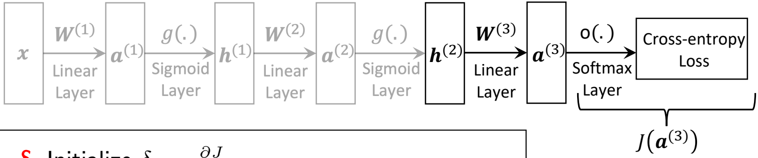
- For brevity, bias is not considered.
- Specific names of the activation functions are shown.
- Loss function is shown at the bottom as a function of output preactivation.

Backpropagation Revisit



- $\dfrac{\partial J}{\partial \mathbf{W}^{(3)}} = \dfrac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}} \dfrac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\dfrac{\partial J}{\partial W^{(2)}} = \dfrac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{W}^{(2)}} \dfrac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \dfrac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \dfrac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\dfrac{\partial J}{\partial W^{(1)}} = \dfrac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \dfrac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \dfrac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \dfrac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \dfrac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \dfrac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit

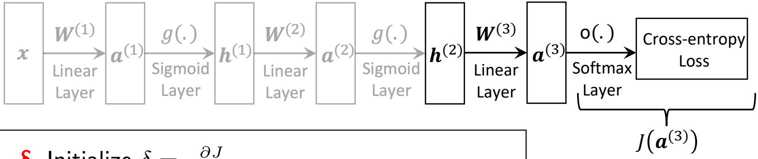


- Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(3)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}}\frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\frac{\partial J}{\partial \mathbf{W}^{(3)}} = \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}}\frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\delta \leftarrow \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}}\frac{\partial J}{\partial \mathbf{a}^{(3)}}$ (New δ for the layer to the left)

Backpropagation Revisit

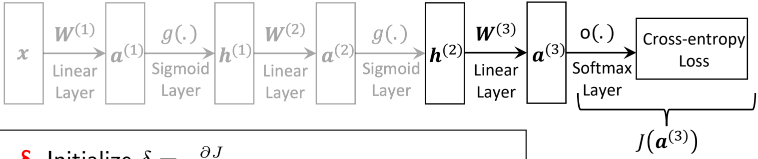


- Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta} \delta$
  - $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(3)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\frac{\partial J}{\partial \mathbf{W}^{(3)}} = \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\delta \leftarrow \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$ (New δ for the layer to the left)

Backpropagation Revisit

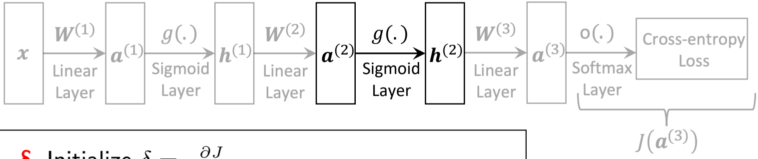


- Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(3)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\frac{\partial J}{\partial \mathbf{W}^{(3)}} = \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\delta \leftarrow \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$ (New δ for the layer to the left)

Backpropagation Revisit

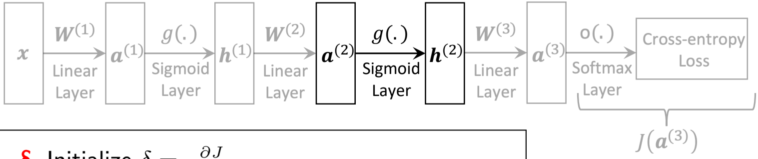


- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(2)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{W}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- § <No operation>
- § $\delta \leftarrow \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit

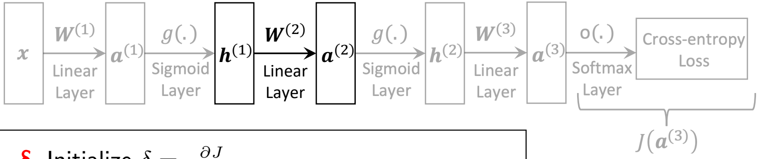


- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(2)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{W}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- § <No operation>
- § $\delta \leftarrow \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit

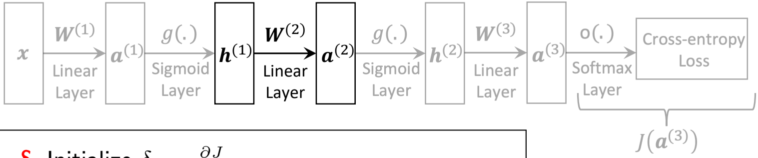


- Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(2)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{W}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- $\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{W}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\delta \leftarrow \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit

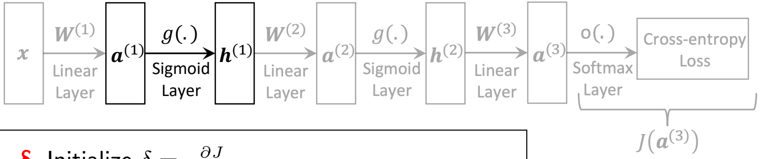


- Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(2)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{W}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- $\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{W}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- $\delta \leftarrow \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit

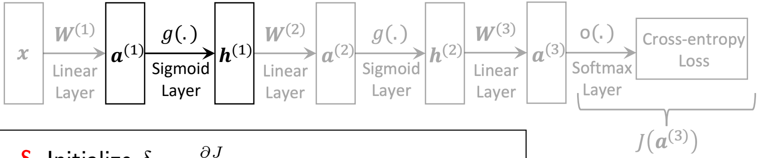


- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta} \delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(1)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- § <No operation>
- § $\delta \leftarrow \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit

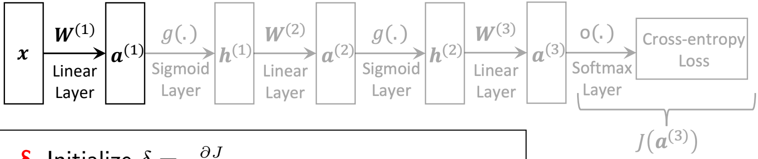


- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(1)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- § <No operation>
- § $\delta \leftarrow \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit

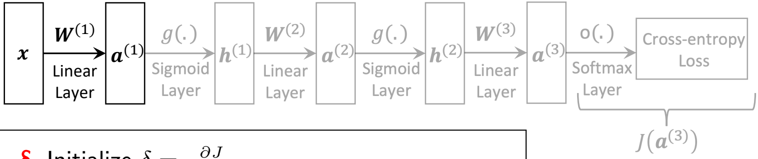


- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

To get $\frac{\partial J}{\partial \mathbf{W}^{(1)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- § $\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § $\delta \leftarrow \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{x}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit



- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

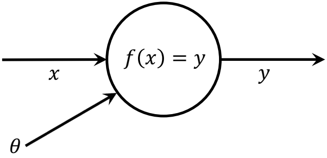
To get $\frac{\partial J}{\partial \mathbf{W}^{(1)}}$ (which is equal to $\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$),

- § $\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § $\delta \leftarrow \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{x}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial J}{\partial \mathbf{a}^{(3)}}$

Backpropagation Revisit

§ Neural network is a chain. We get δ from the next layer which is backpropagated into the previous layer.

- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

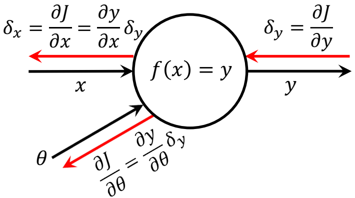


Source: CS W182 course, Sergey Levine, UC Berkeley

Backpropagation Revisit

§ Neural network is a chain. We get δ from the next layer which is backpropagated into the previous layer.

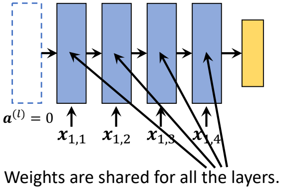
- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$



Source: CS W182 course, Sergey Levine, UC Berkeley

Backpropagation for Shared Weights

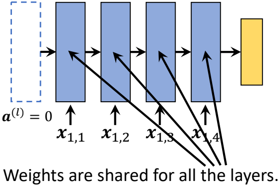
- RNN uses shared weights.



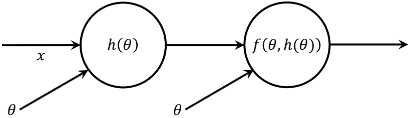
- What change to *backpropagation* is required?

Backpropagation for Shared Weights

- RNN uses shared weights.

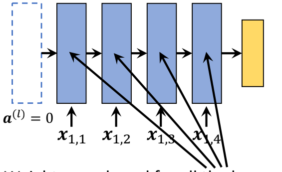


- What change to *backpropagation* is required?

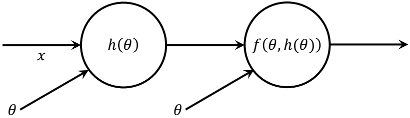


Backpropagation for Shared Weights

- RNN uses shared weights.



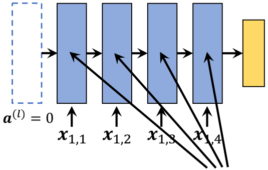
- What change to *backpropagation* is required?



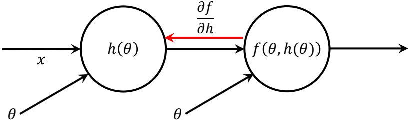
- (Remember:) If $u = f(x, y)$, where $x = \phi(t), y = \psi(t)$, then $\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial t}$
- $\frac{\partial f}{\partial \theta} = \frac{\partial f}{\partial \theta}\frac{\partial \theta}{\partial \theta} + \frac{\partial f}{\partial h}\frac{\partial h}{\partial \theta} = \frac{\partial f}{\partial \theta} + \frac{\partial h}{\partial \theta}\frac{\partial f}{\partial h}$

Backpropagation for Shared Weights

§ RNN uses shared weights.



§ What change to *backpropagation* is required?



§ (Remember:) If $u = f(x, y)$, where $x = \phi(t), y = \psi(t)$, then $\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial t}$

§ $\frac{\partial f}{\partial \theta} = \frac{\partial f}{\partial \theta}\frac{\partial \theta}{\partial \theta} + \frac{\partial f}{\partial h}\frac{\partial h}{\partial \theta} = \frac{\partial f}{\partial \theta} + \frac{\partial h}{\partial \theta}\frac{\partial f}{\partial h}$

§ ~~$\frac{\partial J}{\partial \theta} \leftarrow \frac{\partial \mathbf{y}}{\partial \theta}\delta$~~ Instead, use: $\frac{\partial J}{\partial \theta} += \frac{\partial \mathbf{y}}{\partial \theta}\delta$

§ "accumulate" the gradients during backward pass

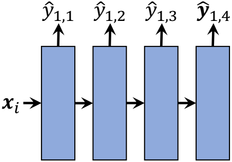
Backpropagation for Shared Weights

- § Initialize $\delta = \frac{\partial J}{\partial \mathbf{a}^{(3)}}$
- § For each layer with input $\mathbf{x}$ and output $\mathbf{y}$
  - ▶ if layer has learnable parameters θ
    - $\frac{\partial J}{\partial \theta} += \frac{\partial \mathbf{y}}{\partial \theta}\delta$
  - ▶ $\delta \leftarrow \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\delta$

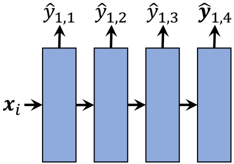
Variable Size Outputs

- **Image description or image captioning:** A crowd of people looking at giraffes in a zoo.

- **Before:** An input at every layer
- **Now:** An output at every layer



An Output Every Layer



At each step:

$$\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{h}^{(l)} = g(\mathbf{a}^{(l)})$$

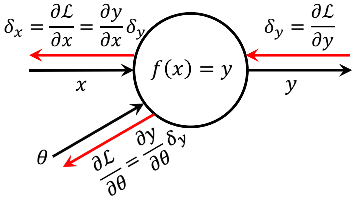
$$\hat{\mathbf{y}}_l = f(\mathbf{h}^{(l)})$$

- $f(.)$ at the end is some kind of *readout* function. Could be as simple as a linear layer + softmax.
- We have a loss on each $\hat{\mathbf{y}}_l$ (*e.g.*, cross-entropy).
- $\mathcal{L}(\hat{\mathbf{y}}_{1:T}) = \sum_l \mathcal{L}(\hat{\mathbf{y}}_l)$

Source: CS W182 course, Sergey Levine, UC Berkeley

Backpropagation with Output Every Layer

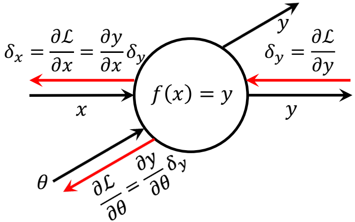
- This is what we saw previously.



Source: CS W182 course, Sergey Levine, UC Berkeley

Backpropagation with Output Every Layer

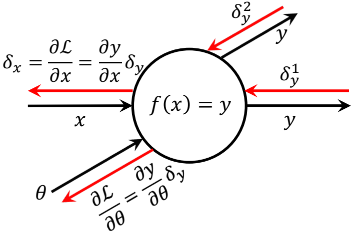
- Some nodes can have outputs going into multiple downstream nodes.



Source: CS W182 course, Sergey Levine, UC Berkeley

Backpropagation with Output Every Layer

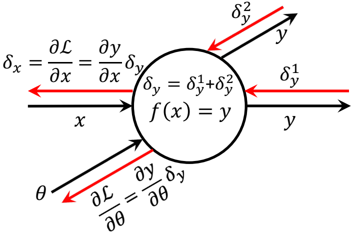
- Some nodes can have outputs going into multiple downstream nodes.
- During backpropagation two δ's coming in.
- Lets call them δ_y^1 and δ_y^2.



Source: CS W182 course, Sergey Levine, UC Berkeley

Backpropagation with Output Every Layer

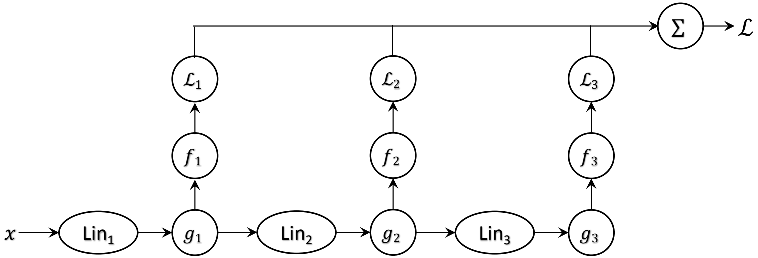
- Some nodes can have outputs going into multiple downstream nodes.
- During backpropagation two δ's coming in.
- Lets call them δ_y^1 and δ_y^2.
- Sum these two δ's for backpropagation.



Source: CS W182 course, Sergey Levine, UC Berkeley

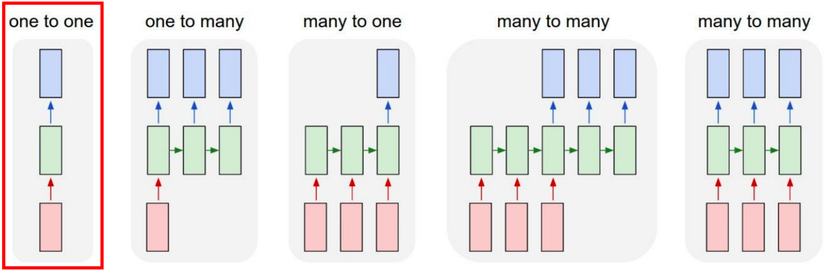
Backpropagation with Output Every Layer

- Very simple rule:
- For each node with multiple descendants in the computational graph:
- Simply add up the delta vectors coming from all of the descendants.



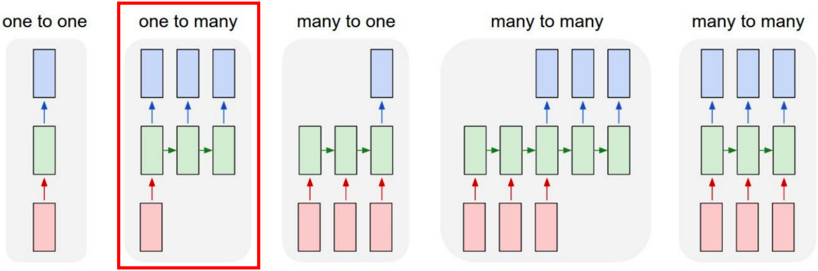
Source: CS W182 course, Sergey Levine, UC Berkeley

Sequence to Sequence



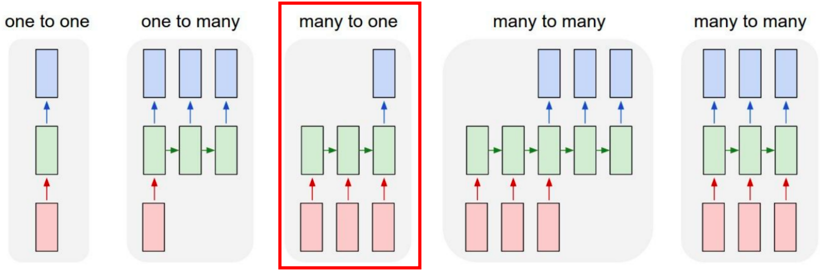
§ **One to one**: Image classification.

Sequence to Sequence



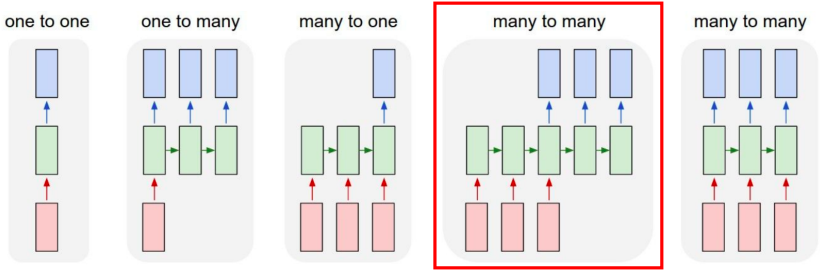
- **One to one**: Image classification.
- **One to many**: Image captioning.

Sequence to Sequence



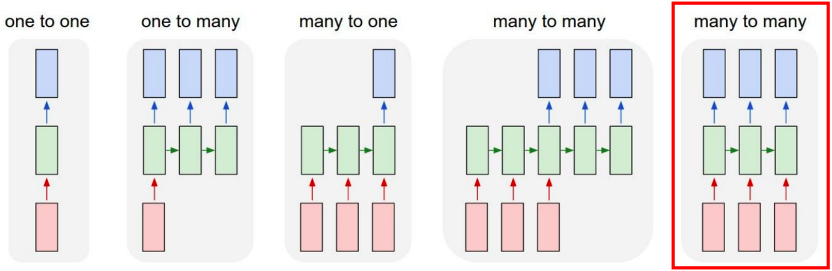
- **One to one**: Image classification.
- **One to many**: Image captioning.
- **Many to one**: Sentiment analysis, Video action recognition.

Sequence to Sequence



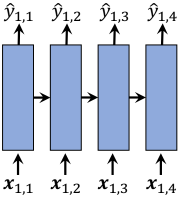
- **One to one**: Image classification.
- **One to many**: Image captioning.
- **Many to one**: Sentiment analysis, Video action recognition.
- **Many to many**: Machine translation.

Sequence to Sequence



- § **One to one**: Image classification.
- § **One to many**: Image captioning.
- § **Many to one**: Sentiment analysis, Video action recognition.
- § **Many to many**: Machine translation.
- § **Many to many**: Tracking, Image classification (with attention)

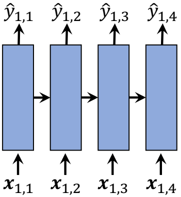
Gradient Flow Problem in RNNs



- RNNs are **extremely deep** networks.
- For a 1000 length sequence, this means backpropagating through 1000 layers.
- $\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \cdots \frac{\partial J}{\partial \mathbf{a}^{(n)}}$

Source: CS W182 course, Sergey Levine, UC Berkeley

Gradient Flow Problem in RNNs



- RNNs are **extremely deep** networks.
- For a 1000 length sequence, this means backpropagating through 1000 layers.
- $\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(2)}} \cdots \frac{\partial J}{\partial \mathbf{a}^{(n)}}$
- Multiplying many many numbers together means,
  - If most of the numbers are < 1, we get 0. (vanishing gradients)
  - If most of the numbers are > 1, we get ∞. (exploding gradients)
  - If all numbers are close to 1, then we get a reasonable answer.
- Exploding gradients could be fixed with gradient clipping.
- Vanishing gradients are bigger problem.
- Intuitively, vanishing gradients prevents gradient signals from later steps reach earlier steps.
- This prevents the RNN from *remembering* things from the begining.

Source: CS W182 course, Sergey Levine, UC Berkeley

Promoting Better Gradient Flow

- Basic idea: We would like the gradients to be close to 1.
- For Jacobians, this means the eigenvalues to be close to 1.

Source: CS W182 course, Sergey Levine, UC Berkeley

Promoting Better Gradient Flow

- Basic idea: We would like the gradients to be close to 1.
- For Jacobians, this means the eigenvalues to be close to 1.
- But first, bit of notations.
- Each timestep,

$$\underbrace{\bar{\mathbf{h}}_{t-1} = \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix};\ \mathbf{a}_t = \mathbf{W}\bar{\mathbf{h}}_{t-1} + \mathbf{b};\ \mathbf{h}_t = g\big(\mathbf{a}_t\big)}_{\mathbf{h}_t = q(\mathbf{h}_{t-1}, \mathbf{x}_t);\ \text{RNN dynamics}}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

Promoting Better Gradient Flow

- Basic idea: We would like the gradients to be close to 1.
- For Jacobians, this means the eigenvalues to be close to 1.
- But first, bit of notations.
- Each timestep,

$$\underbrace{\bar{\mathbf{h}}_{t-1} = \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix};\ \mathbf{a}_t = \mathbf{W}\bar{\mathbf{h}}_{t-1} + \mathbf{b};\ \mathbf{h}_t = g(\mathbf{a}_t)}_{\mathbf{h}_t = q(\mathbf{h}_{t-1}, \mathbf{x}_t);\ \text{RNN dynamics}}$$

- Best gradient flow is when dynamics Jacobian $\frac{\partial q}{\partial \mathbf{h}_{t-1}} = \mathbf{I}$
- However, it 'depends' on whether you want the RNN to 'forget' the past or not.

Source: CS W182 course, Sergey Levine, UC Berkeley

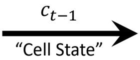
Promoting Better Gradient Flow

- We want $\frac{\partial q_i}{\partial \mathbf{h}_{t-1,i}} \approx 1$ if we choose to *remember* $\mathbf{h}_{t-1,i}$.
- A little "neural circuit" decides whether to remember or overwrite.

Source: CS W182 course, Sergey Levine, UC Berkeley

Promoting Better Gradient Flow

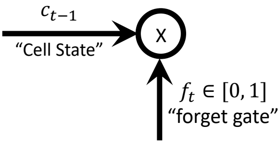
- We want $\frac{\partial q_i}{\partial \mathbf{h}_{t-1,i}} \approx 1$ if we choose to *remember* $\mathbf{h}_{t-1,i}$.
- A little "neural circuit" decides whether to remember or overwrite.



Source: CS W182 course, Sergey Levine, UC Berkeley

Promoting Better Gradient Flow

- We want $\frac{\partial q_i}{\partial \mathbf{h}_{t-1,i}} \approx 1$ if we choose to *remember* $\mathbf{h}_{t-1,i}$.
- A little "neural circuit" decides whether to remember or overwrite.

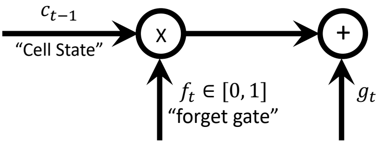


- $f_t \to 0$ means $\mathbf{c}_{t-1}$'s value is forgotten.
- $f_t \to 1$ means $\mathbf{c}_{t-1}$'s value is remembered.

Source: CS W182 course, Sergey Levine, UC Berkeley

Promoting Better Gradient Flow

- We want $\frac{\partial q_i}{\partial \mathbf{h}_{t-1,i}} \approx 1$ if we choose to *remember* $\mathbf{h}_{t-1,i}$.
- A little "neural circuit" decides whether to remember or overwrite.

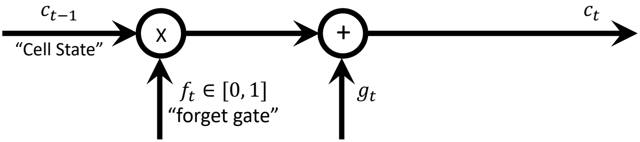


- $f_t \to 0$ means $\mathbf{c}_{t-1}$'s value is forgotten and overridden by g_t.
- $f_t \to 1$ means $\mathbf{c}_{t-1}$'s value is remembered and additively modified by g_t.

Source: CS W182 course, Sergey Levine, UC Berkeley

Promoting Better Gradient Flow

- We want $\frac{\partial q_i}{\partial \mathbf{h}_{t-1,i}} \approx 1$ if we choose to *remember* $\mathbf{h}_{t-1,i}$.
- A little "neural circuit" decides whether to remember or overwrite.

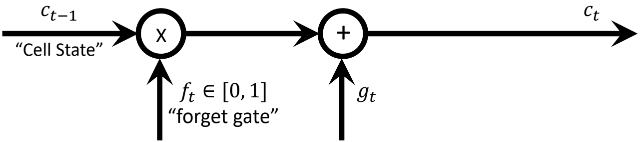


- $f_t \to 0$ means $\mathbf{c}_{t-1}$'s value is forgotten and overridden by g_t.
- $f_t \to 1$ means $\mathbf{c}_{t-1}$'s value is remembered and additively modified by g_t.
- c_t is the new cell state.
- $c_t = f_t c_{t-1} + g_t$ with $f_t \in [0, 1]$.
- $\frac{\partial q_i}{\partial \mathbf{c}_{t-1,i}} = f_t \in [0, 1]$

Source: CS W182 course, Sergey Levine, UC Berkeley

Promoting Better Gradient Flow

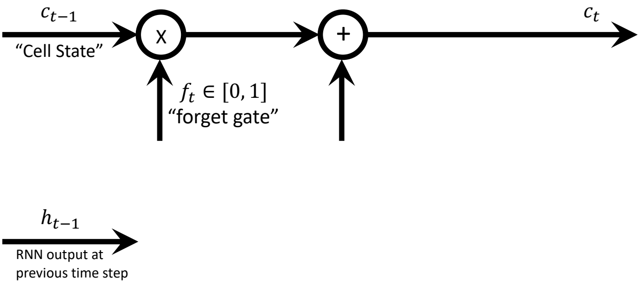
- We want $\frac{\partial q_i}{\partial \mathbf{h}_{t-1,i}} \approx 1$ if we choose to *remember* $\mathbf{h}_{t-1,i}$.
- A little "neural circuit" decides whether to remember or overwrite.



- $f_t \to 0$ means $\mathbf{c}_{t-1}$'s value is forgotten and overridden by g_t.
- $f_t \to 1$ means $\mathbf{c}_{t-1}$'s value is remembered and additively modified by g_t.
- c_t is the new cell state.
- $c_t = f_t c_{t-1} + g_t$ with $f_t \in [0, 1]$.
- $\frac{\partial q_i}{\partial \mathbf{c}_{t-1,i}} = f_t \in [0, 1]$
- Where do we get f_t and g_t?

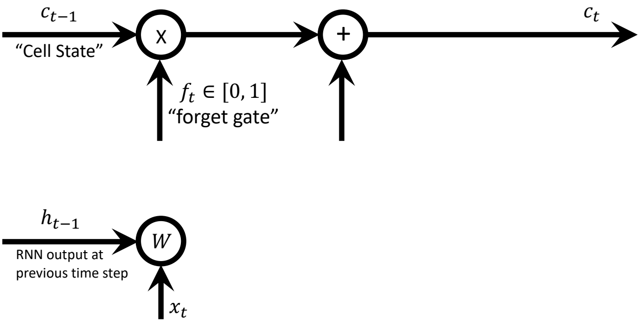
Source: CS W182 course, Sergey Levine, UC Berkeley

LSTM Cells



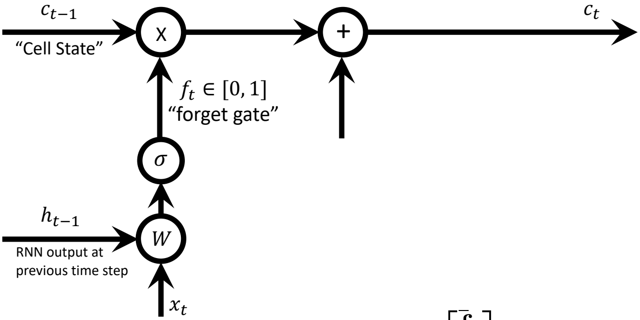
Source: CS W182 course, Sergey Levine, UC Berkeley

LSTM Cells



Source: CS W182 course, Sergey Levine, UC Berkeley

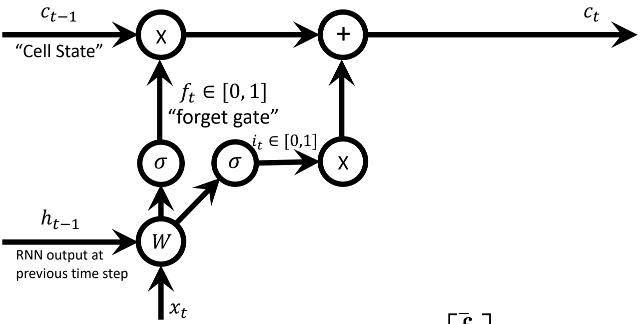
LSTM Cells



$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b} = \begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

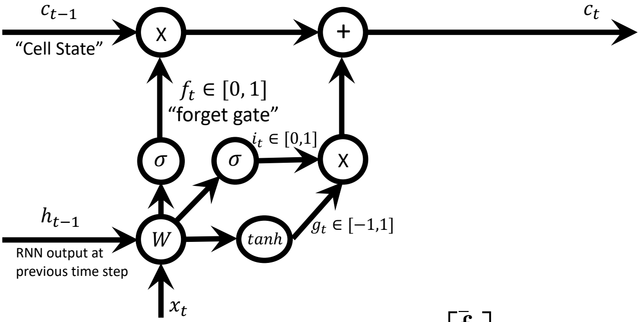
LSTM Cells



$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b} = \begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

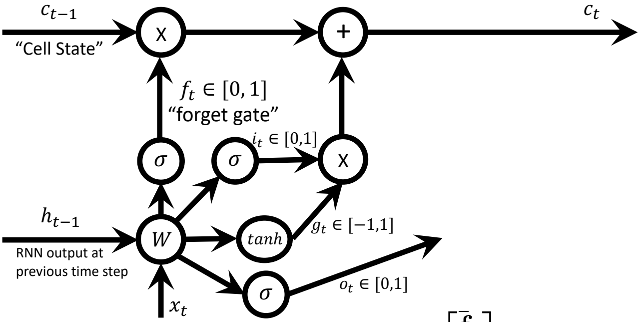
LSTM Cells



$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix}+\mathbf{b}=\begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

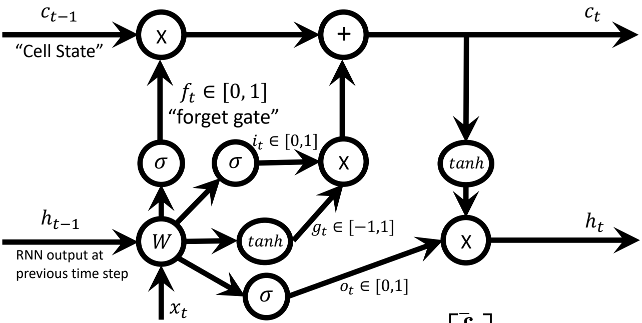
LSTM Cells



$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b} = \begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

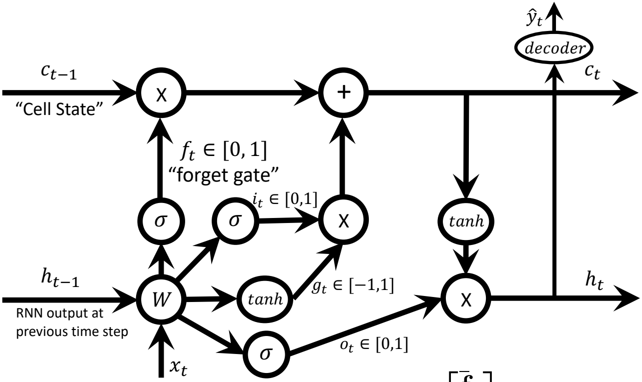
LSTM Cells



$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b} = \begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

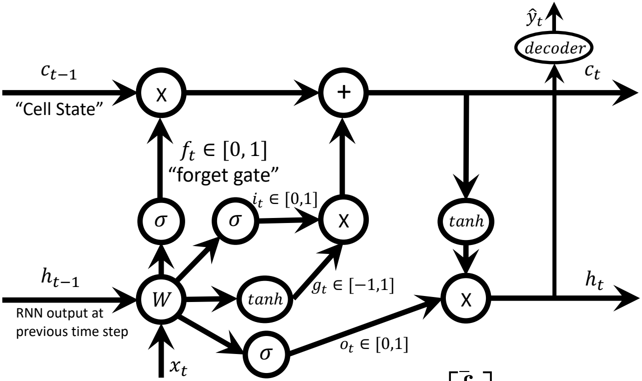
LSTM Cells



$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix}+\mathbf{b}=\begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

LSTM Cells

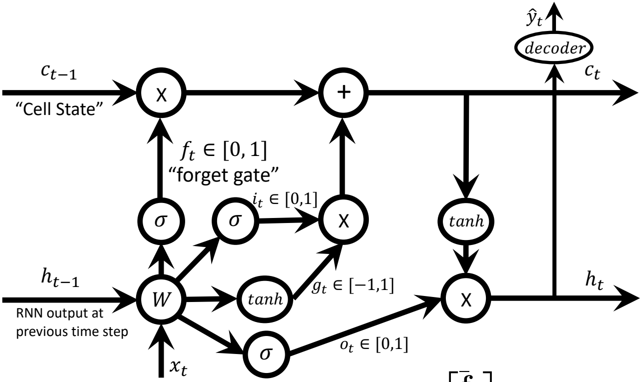


$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b} = \begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

Isn't all these a little arbitrary?

Source: CS W182 course, Sergey Levine, UC Berkeley

LSTM Cells



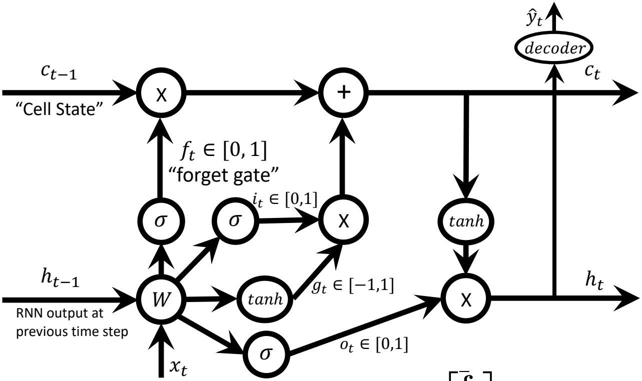
$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix}+\mathbf{b}=\begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

Isn't all these a little arbitrary?

Yes, but it ends up working quite well in practice and much better than a naive RNN.

Source: CS W182 course, Sergey Levine, UC Berkeley

LSTM Cells



$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix}+\mathbf{b}=\begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

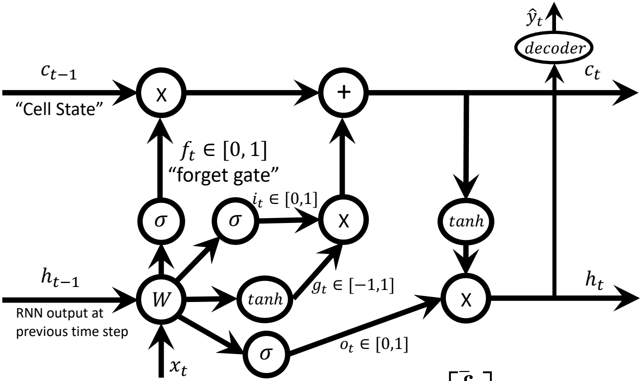
$c_t = f_t c_{t-1} + g_t$

Changes very little step to step!

Long term memory.

Source: CS W182 course, Sergey Levine, UC Berkeley

LSTM Cells



$$\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b} = \begin{bmatrix}\bar{\mathbf{f}}_t\\ \bar{\mathbf{i}}_t\\ \bar{\mathbf{g}}_t\\ \bar{\mathbf{o}}_t\end{bmatrix}$$

$c_t = f_t c_{t-1} + g_t$
Changes very little step to step!
Long term memory.

Changes all the time (multiplicative) *short term* memory.

Source: CS W182 course, Sergey Levine, UC Berkeley

Some Practical Notes

- In practice, naive RNNs almost never work.
- LSTM units dramatically improve over naive RNNs.
- Requires way more hyperparameter tuning than standard fully connected or conv-nets.
- Some modifications and alternatives work better for sequences.
  - Transformers
  - Gated recurrent unit (GRU)

Source: CS W182 course, Sergey Levine, UC Berkeley