Recurrent Neural Networks and Transformers

Shan-Hung Wu
shwu@cs.nthu.edu.tw

Department of Computer Science,
National Tsing Hua University, Taiwan

Machine Learning

Outline

Outline

Outline

Sequential Data

- So far, we assume that data points $(\boldsymbol{x}, \boldsymbol{y})$'s in a dataset are i.i.d

Sequential Data

- So far, we assume that data points $(\boldsymbol{x}, \boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications

Sequential Data

- So far, we assume that data points $(\boldsymbol{x}, \boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications
- ***Sequential data***: data points come in order and successive points may be dependent, e.g.,
  - Letters in a word
  - Words in a sentence/document
  - Phonemes in a spoken word utterance
  - Page clicks in a Web session
  - Frames in a video, etc.

Sequential Data

- So far, we assume that data points $(\boldsymbol{x}, \boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications
- ***Sequential data***: data points come in order and successive points may be dependent, e.g.,
  - Letters in a word
  - Words in a sentence/document
  - Phonemes in a spoken word utterance
  - Page clicks in a Web session
  - Frames in a video, etc.
- Dataset: $\mathbf{X} = \{\boldsymbol{X}^{(n)}\}_n \in \mathbb{R}^{N \times (D,K) \times T}$

Sequential Data

- So far, we assume that data points $(\boldsymbol{x}, \boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications
- ***Sequential data***: data points come in order and successive points may be dependent, e.g.,
  - Letters in a word
  - Words in a sentence/document
  - Phonemes in a spoken word utterance
  - Page clicks in a Web session
  - Frames in a video, etc.
- Dataset: $\mathbf{X} = \{\boldsymbol{X}^{(n)}\}_n \in \mathbb{R}^{N \times (D,K) \times T}$
  - $\boldsymbol{X}^{(n)} = \{(\boldsymbol{x}^{(n,t)}, \boldsymbol{y}^{(n,t)})\}_t$ a ***sequence***, where the superscript n can be omitted for simplicity

Sequential Data

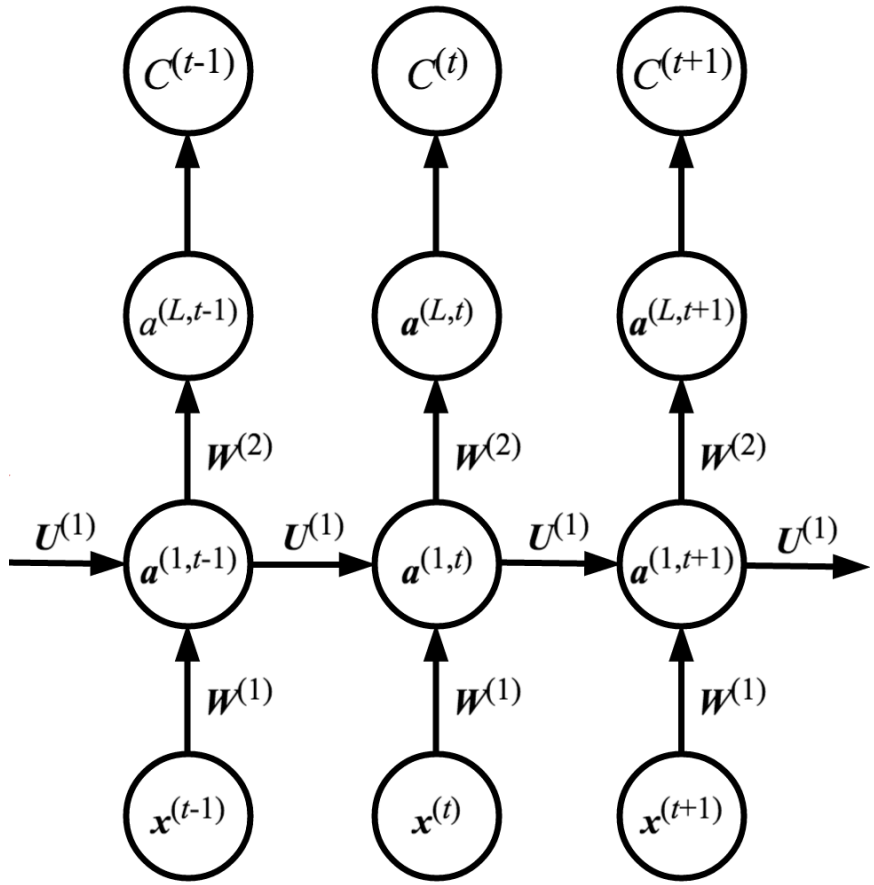
- So far, we assume that data points $(\boldsymbol{x}, \boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications
- ***Sequential data***: data points come in order and successive points may be dependent, e.g.,
  - Letters in a word
  - Words in a sentence/document
  - Phonemes in a spoken word utterance
  - Page clicks in a Web session
  - Frames in a video, etc.
- Dataset: $\mathbf{X} = \{\boldsymbol{X}^{(n)}\}_n \in \mathbb{R}^{N \times (D,K) \times T}$
  - $\boldsymbol{X}^{(n)} = \{(\boldsymbol{x}^{(n,t)}, \boldsymbol{y}^{(n,t)})\}_t$ a ***sequence***, where the superscript n can be omitted for simplicity
  - T is call the ***horizon*** and may be different between $\boldsymbol{x}^{(n)}$ and $\boldsymbol{y}^{(n)}$ and across data points n's

Sequence Modeling I

- How to model sequential data?

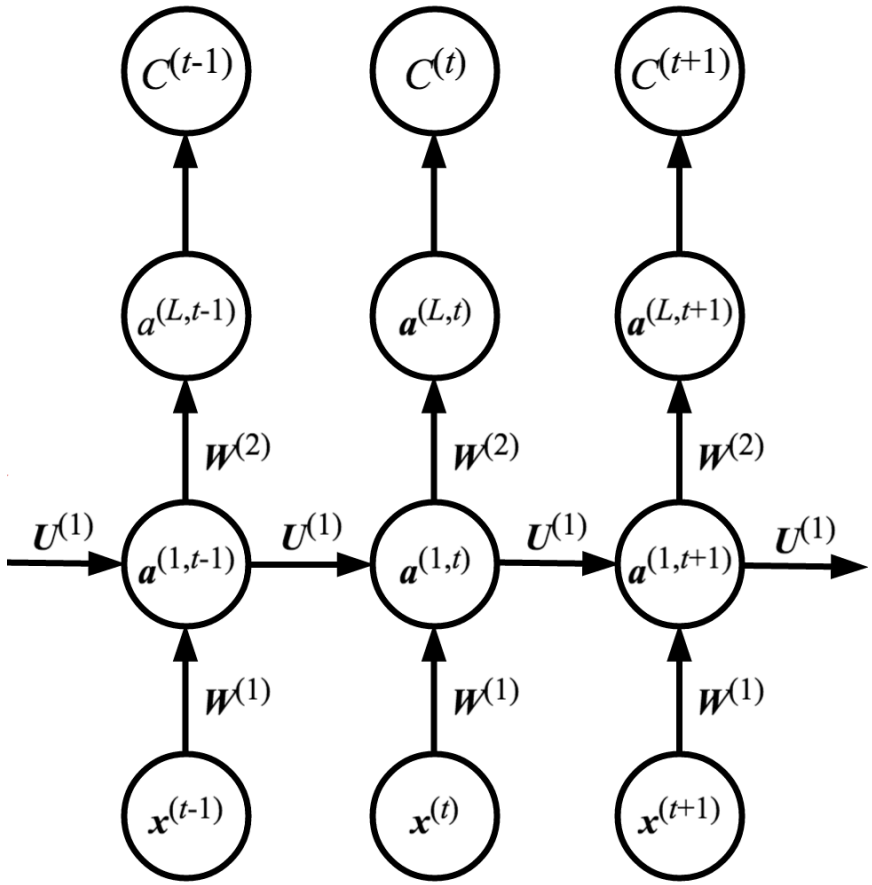
Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):



Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):
- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$

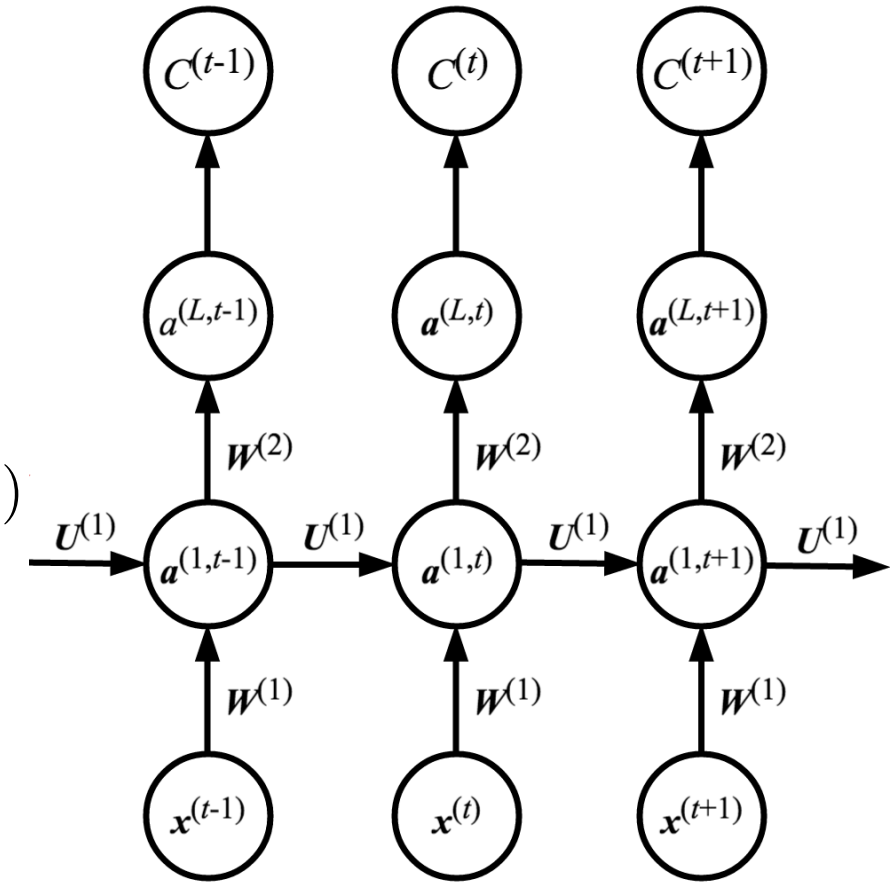


Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):
- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$
- Output $\mathbf{a}^{(L,t)}$ depends on hidden activations:

$$\begin{aligned}\mathbf{a}^{(k,t)} &= \text{act}(\mathbf{z}^{(k,t)}) \\ &= \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})\end{aligned}$$

  - Bias term omitted

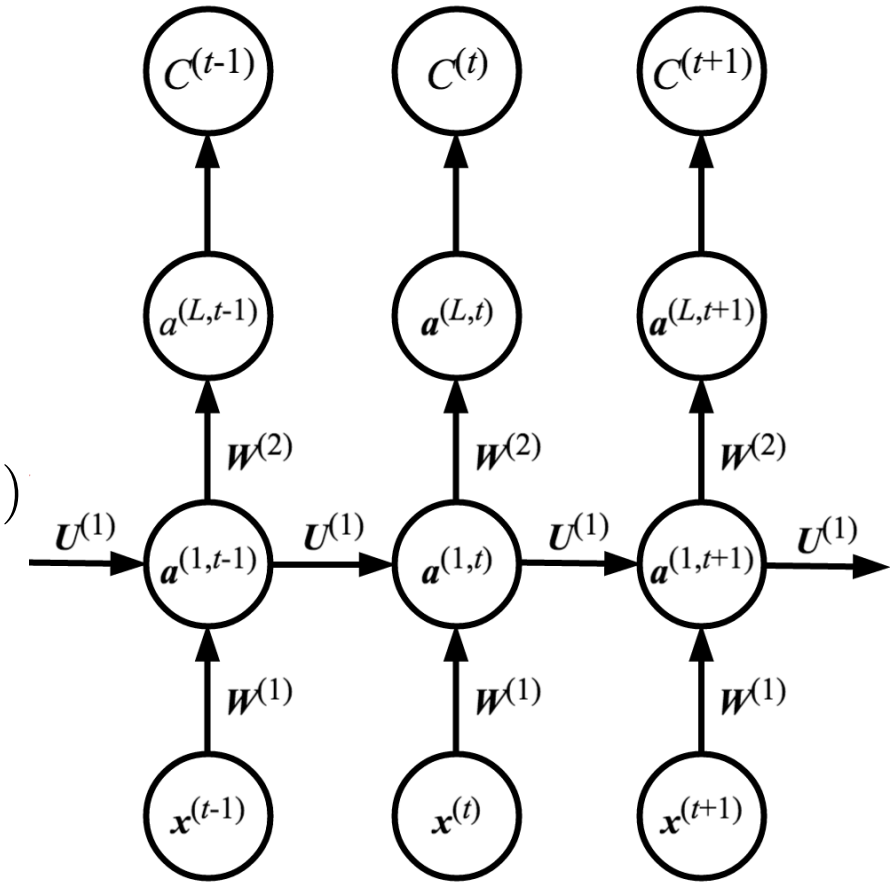


Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):
- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$
- Output $\mathbf{a}^{(L,t)}$ depends on hidden activations:

$$\begin{aligned}\mathbf{a}^{(k,t)} &= \text{act}(\mathbf{z}^{(k,t)}) \\ &= \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})\end{aligned}$$

  - Bias term omitted
- $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
  - Earlier points are less important

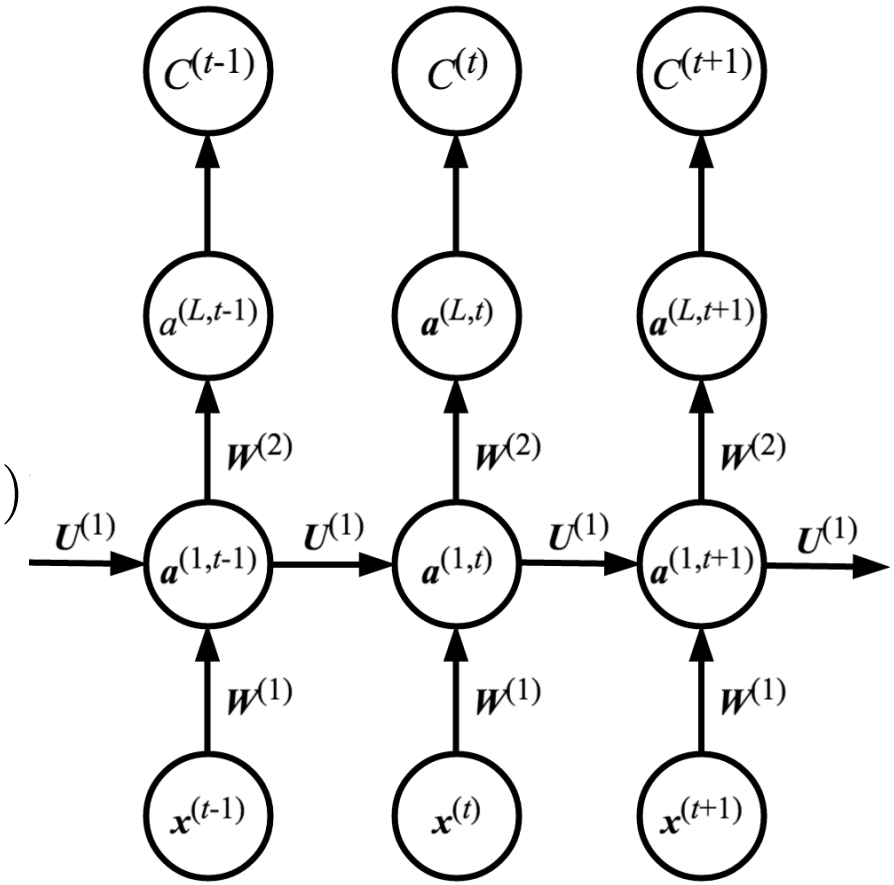


Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):
- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$
- Output $\mathbf{a}^{(L,t)}$ depends on hidden activations:

$$\begin{aligned}\mathbf{a}^{(k,t)} &= \text{act}(\mathbf{z}^{(k,t)}) \\ &= \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})\end{aligned}$$

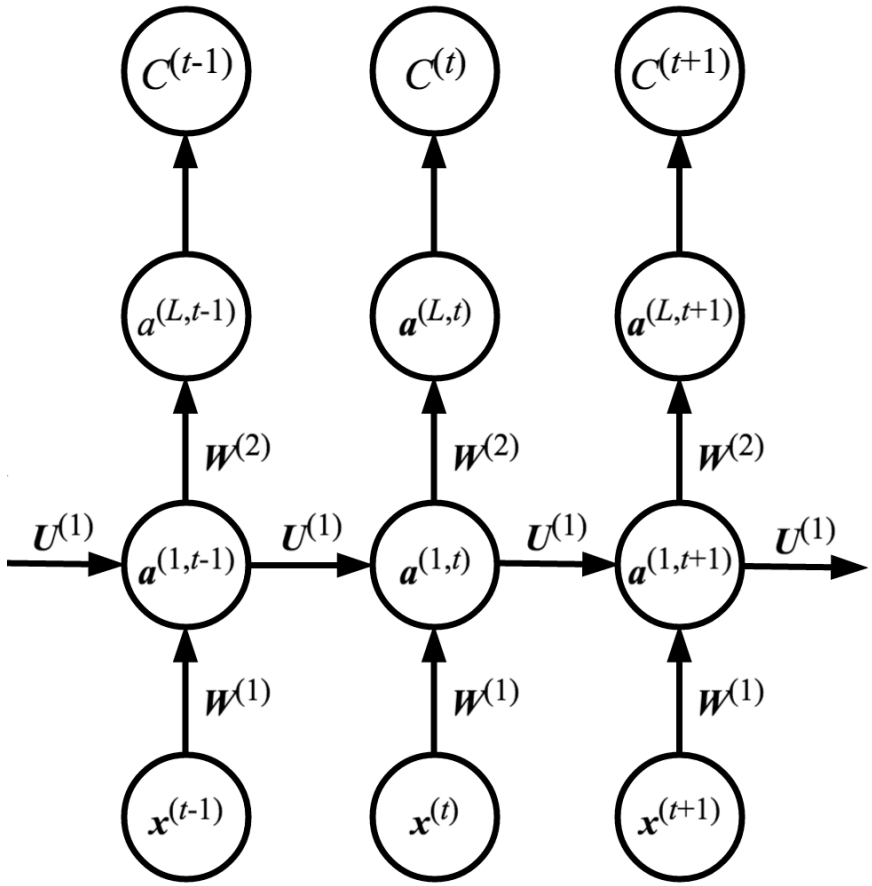
  - Bias term omitted
- $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
  - Earlier points are less important
- $\mathbf{a}^{(\cdot,t)}$'s at deeper layers give more abstract summarizations



Sequence Modeling II

$$\mathbf{a}^{(k,t)} = \text{act}(\mathbf{z}^{(k,t)}) = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

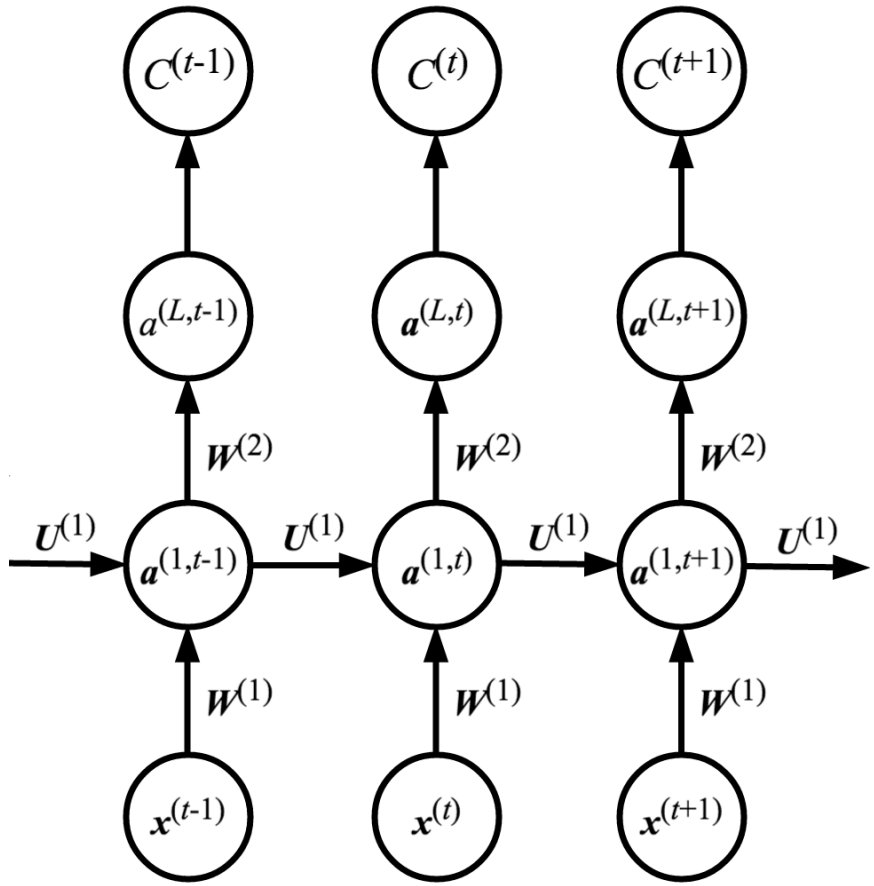
- Weights are ***shared*** across time instances



Sequence Modeling II

$$\mathbf{a}^{(k,t)} = \text{act}(\mathbf{z}^{(k,t)}) = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

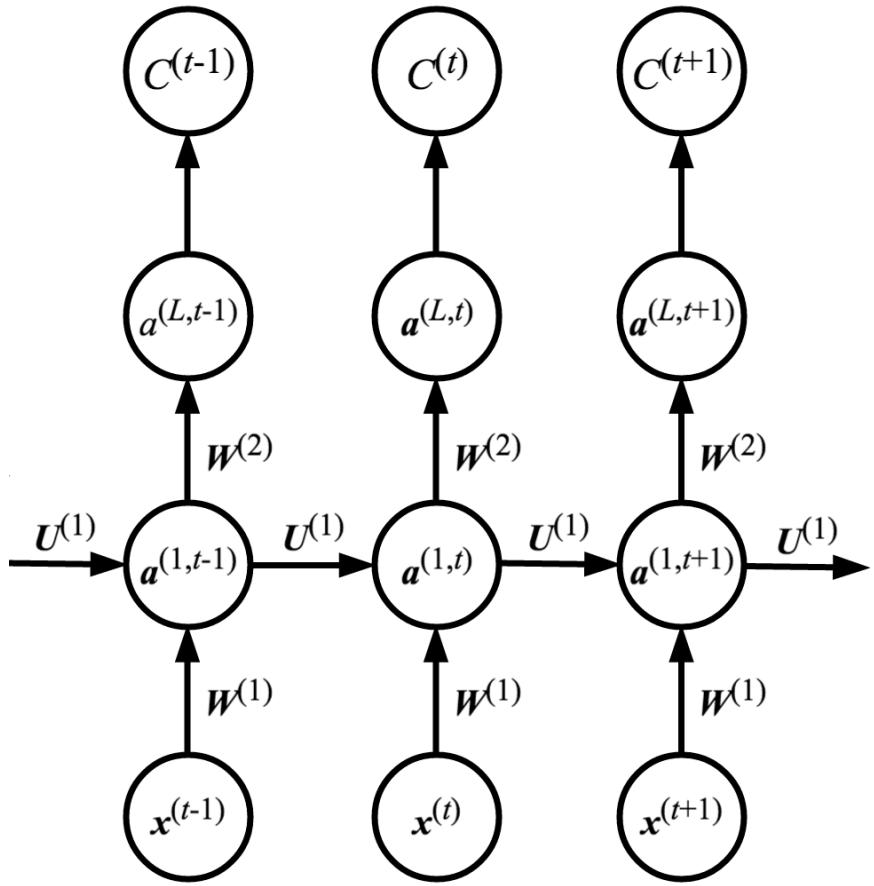
- Weights are ***shared*** across time instances
- Assumes that the "transition functions" are time invariant



Sequence Modeling II

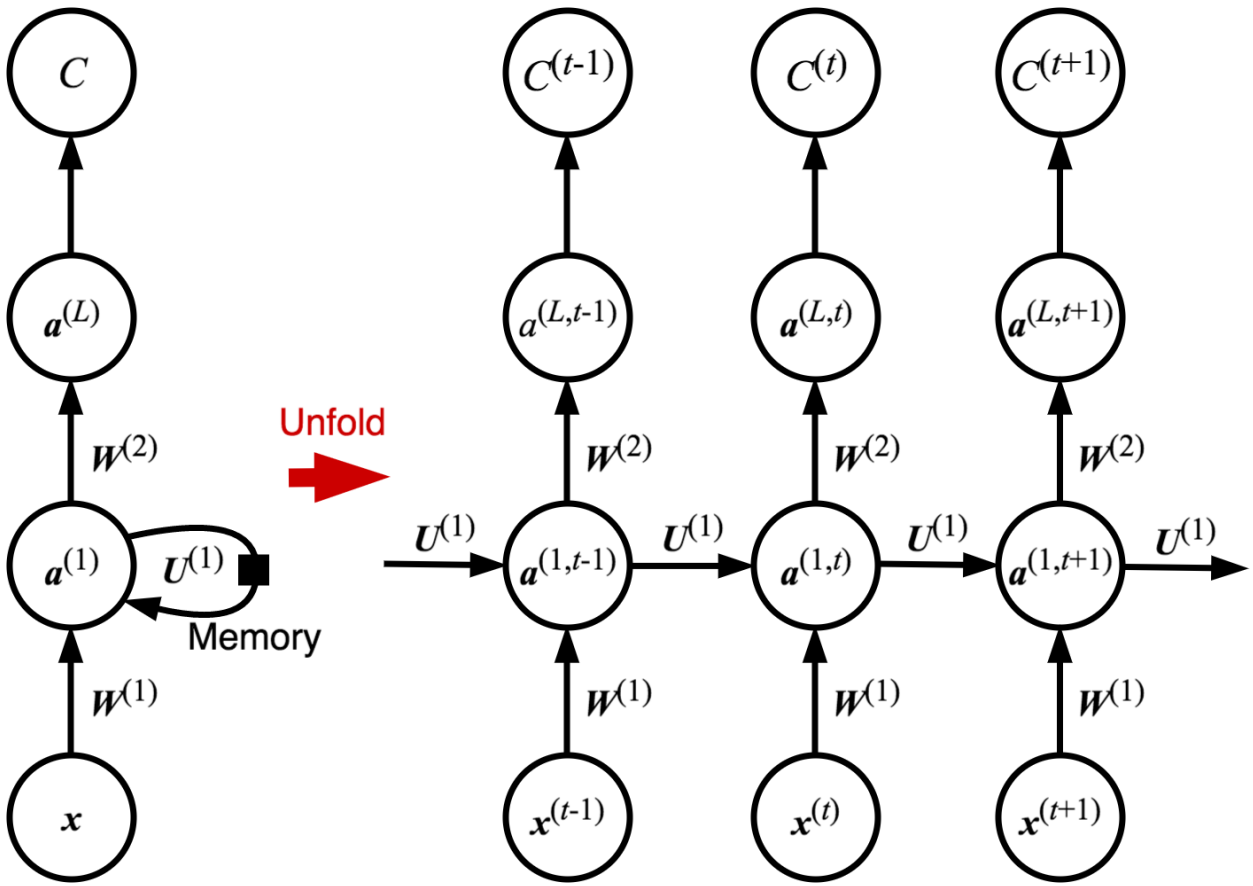
$$\mathbf{a}^{(k,t)} = \text{act}(\mathbf{z}^{(k,t)}) = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- Weights are ***shared*** across time instances
- Assumes that the "transition functions" are time invariant
- Our goal is to learn $\boldsymbol{U}^{(k)}$'s and $\boldsymbol{W}^{(k)}$'s for $k = 1, \cdots, L$



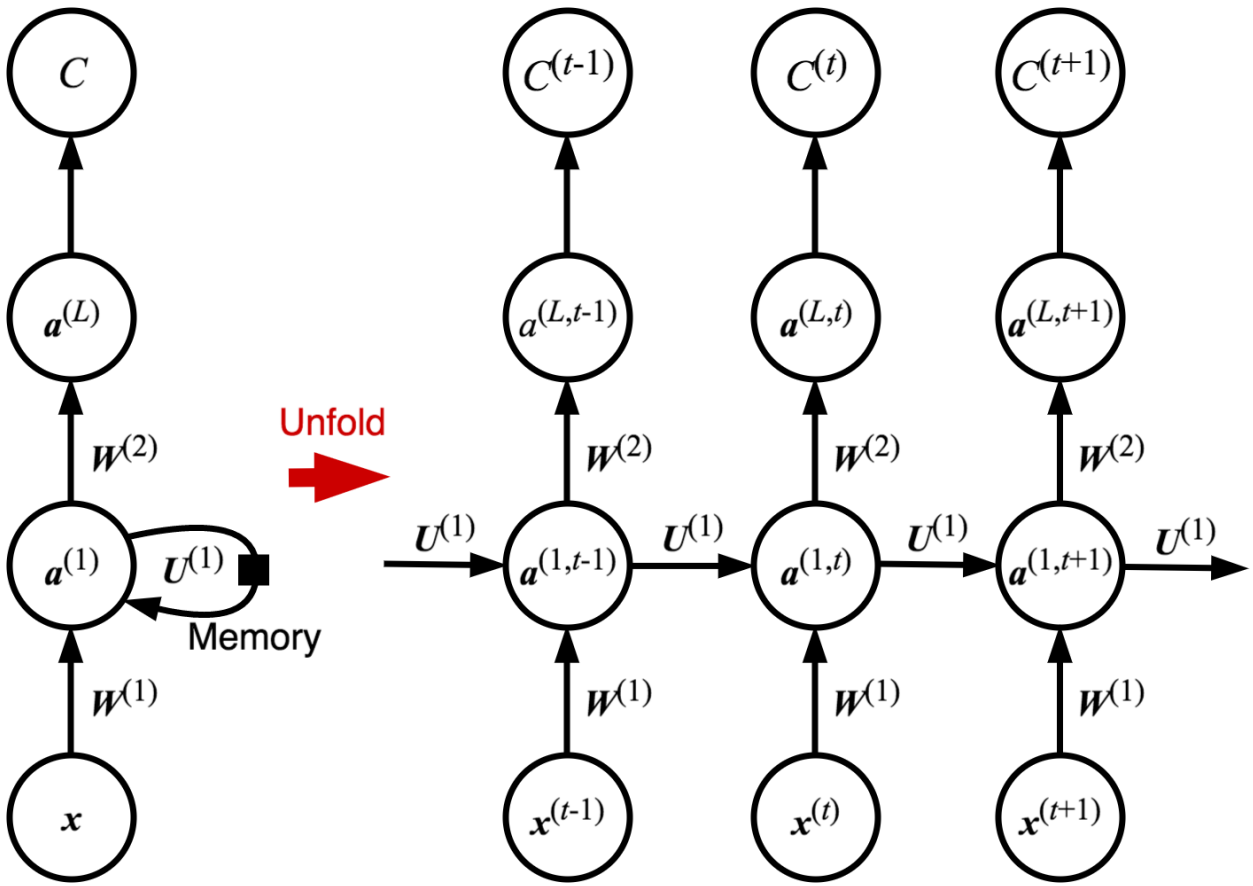
RNNs have Memory

- The computational graph of an RNN can be ***folded*** in time



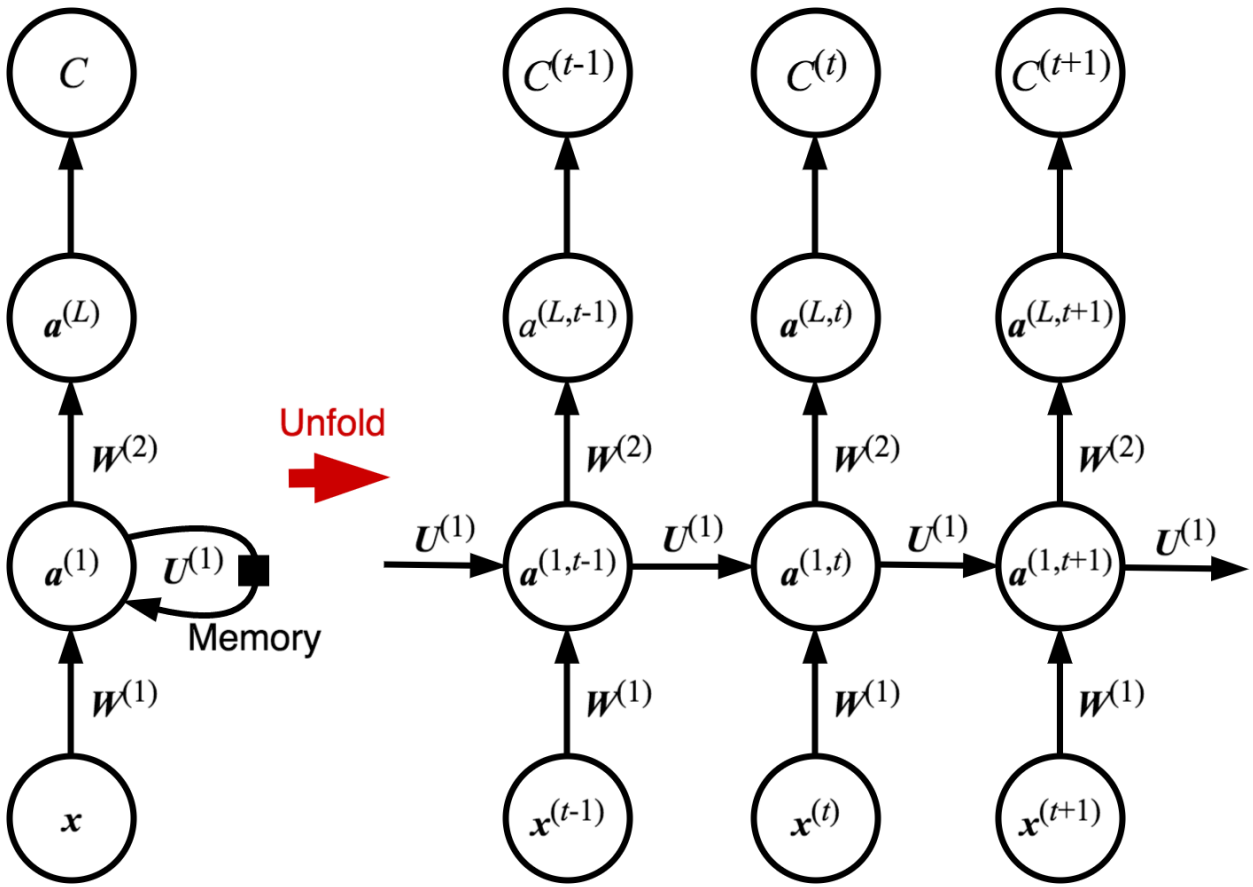
RNNs have Memory

- The computational graph of an RNN can be ***folded*** in time
- Black squares denotes ***memory*** access



Output Layer (1/2)

- With multi-class $\mathbf{y}^{(t)}$,
  - $\mathbf{a}^{(L,t)}$ represents the probability of each class
  - $C^{(t)}$ is cross entropy
- How to obtain $\hat{\mathbf{y}}^{(t)}$ rom $\mathbf{a}^{(L,t)}$ at inference time?



Output Layer (2/2)

- Output sampling for multi-class tasks:
  - Greedy: sample $\hat{\mathbf{y}}^{(t)}$ from $\mathbf{a}^{(L,t)}$
  - Bean search: sample $\hat{\mathbf{y}}^{(t)}$ from the most probable paths of the join distribution $(\mathbf{a}^{(L,t)}, \mathbf{a}^{(L,t-1)}, \cdots \mathbf{a}^{(L,t-b)})$, where b is bean size
  - Noisy: sample $\hat{\mathbf{y}}^{(t)}$ based on $\mathbf{a}^{(L,t)}$ + noise

Output Layer (2/2)

- Output sampling for multi-class tasks:
  - Greedy: sample $\hat{\mathbf{y}}^{(t)}$ from $\mathbf{a}^{(L,t)}$
  - Bean search: sample $\hat{\mathbf{y}}^{(t)}$ from the most probable paths of the join distribution $(\mathbf{a}^{(L,t)}, \mathbf{a}^{(L,t-1)}, \cdots \mathbf{a}^{(L,t-b)})$, where b is bean size
  - Noisy: sample $\hat{\mathbf{y}}^{(t)}$ based on $\mathbf{a}^{(L,t)}$ + noise
- Problem: out of vocabulary or high dimensional $\mathbf{a}^{(L,t)}$

- Solution?

Output Layer (2/2)

- Output sampling for multi-class tasks:
  - Greedy: sample $\hat{\mathbf{y}}^{(t)}$ from $\mathbf{a}^{(L,t)}$
  - Bean search: sample $\hat{\mathbf{y}}^{(t)}$ from the most probable paths of the join distribution $(\mathbf{a}^{(L,t)}, \mathbf{a}^{(L,t-1)}, \cdots \mathbf{a}^{(L,t-b)})$, where b is bean size
  - Noisy: sample $\hat{\mathbf{y}}^{(t)}$ based on $\mathbf{a}^{(L,t)}$ + noise
- Problem: out of vocabulary or high dimensional $\mathbf{a}^{(L,t)}$

- Solution? ***Subword tokenization*** (to be discussed later)

RNNs vs CNNs for Sequential Data

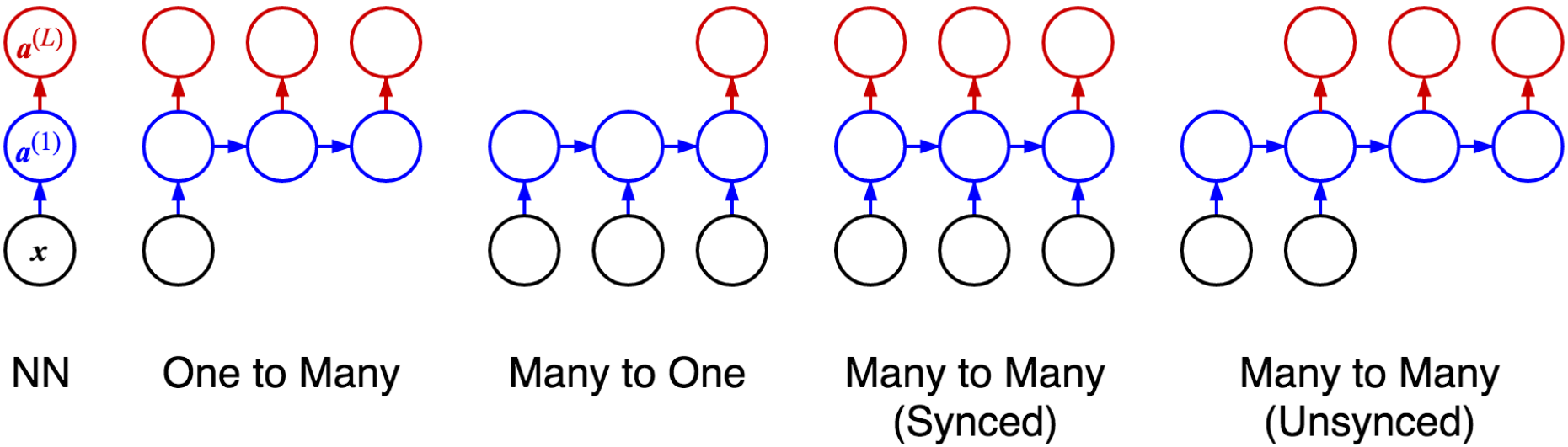
- On processing a sequence of length T at each layer with
  - D-dimensional point input and output
  - F = the CNN filter/kernel size
  - #CNN filters = D

	#Weights	Computation	Autoregressive	Point Distance
CNN	$O(FD^2)$	$O(TFD^2)$	No	$O(\log_F T)$
RNN	$O(D^2)$	$O(TD^2)$	Yes	$O(T)$

Outline

Input and Output

- $\boldsymbol{x}^{(t)}$'s and $\boldsymbol{y}^{(t)}$'s do ***not*** need to have one-to-one correspondence:

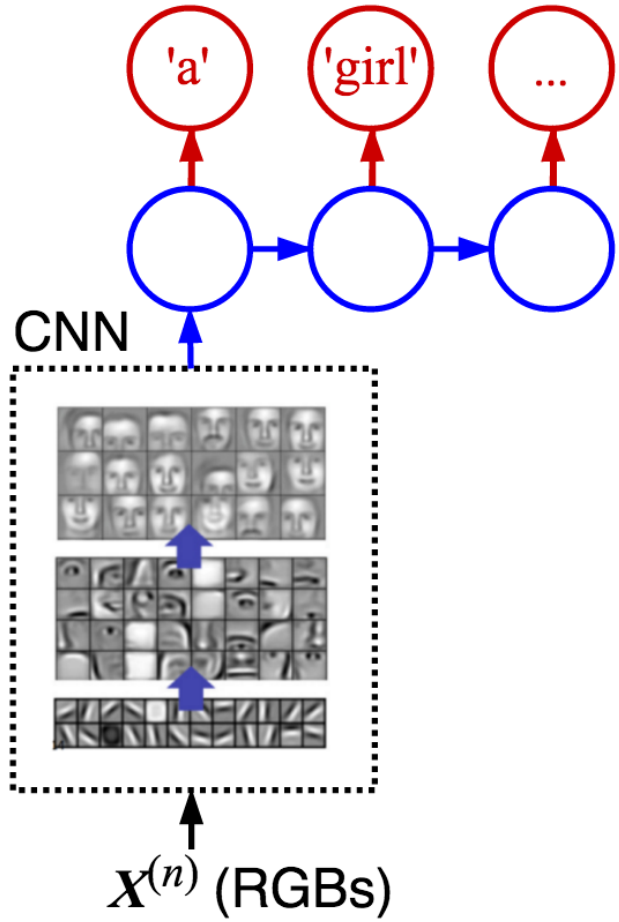


One2Many: Image Captioning

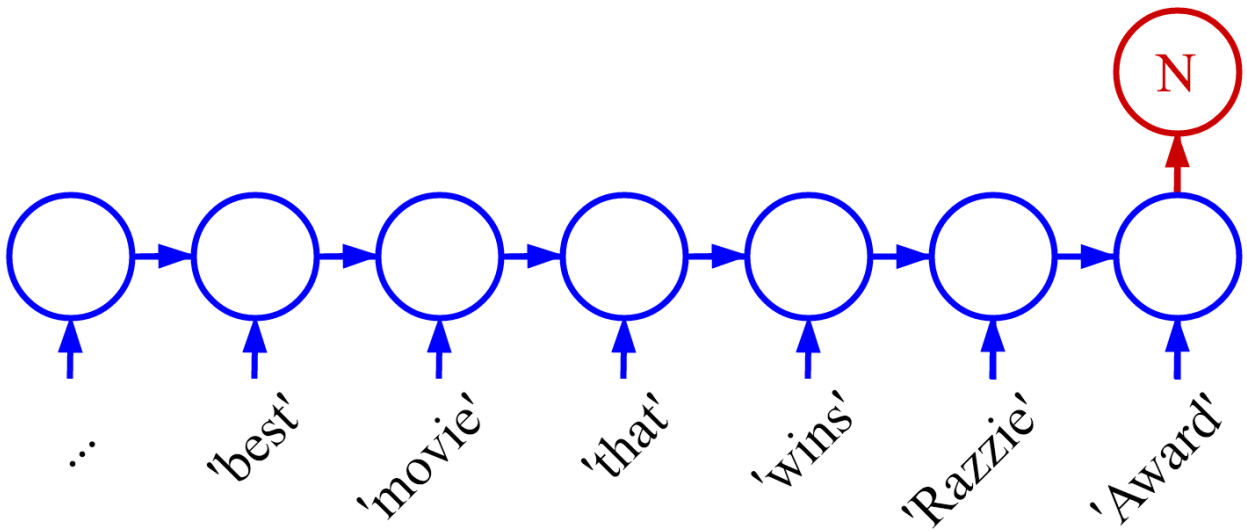
"A little girl sitting on a bed with a teddy bear."

One2Many: Image Captioning

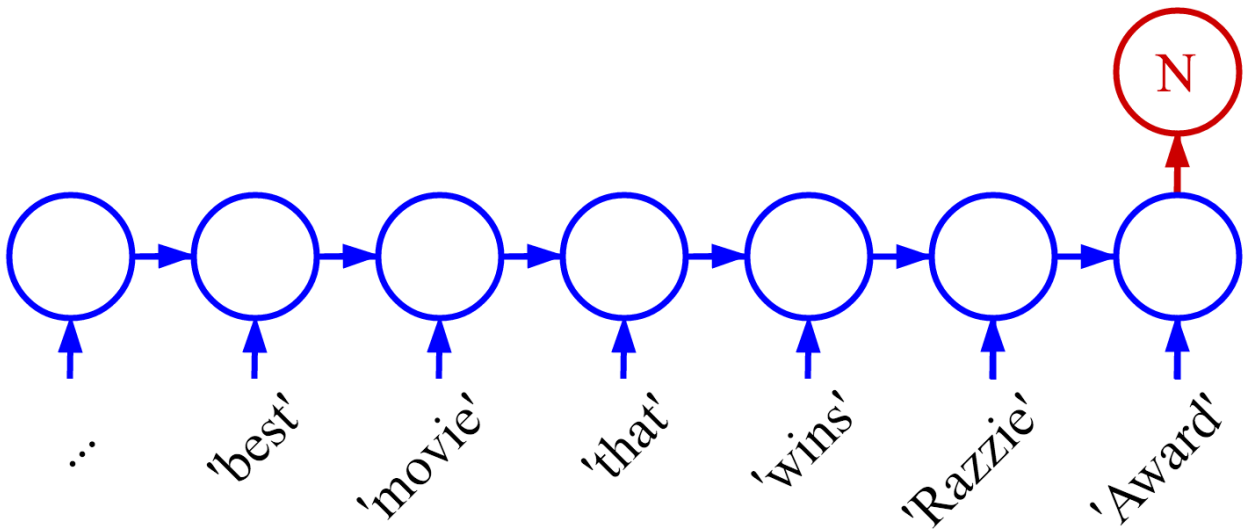
"A little girl sitting on a bed with a teddy bear."



Many2One: Sentiment Analysis



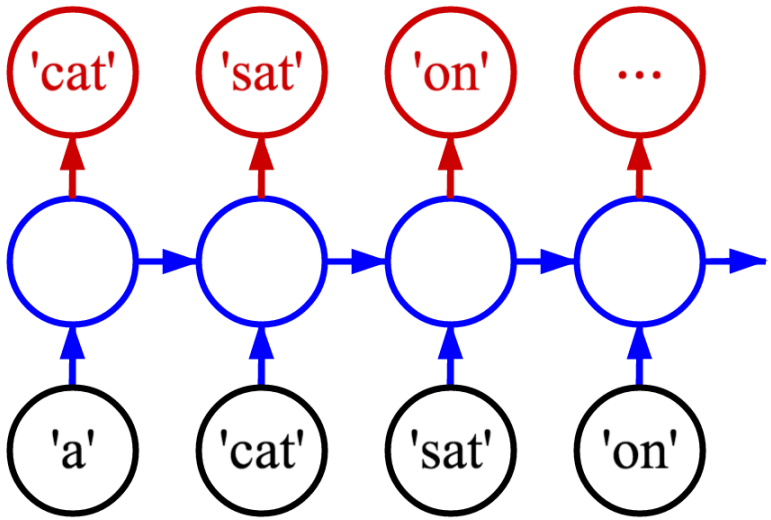
Many2One: Sentiment Analysis



- A single word (e.g., "Razzie") can negate the entire input sentence

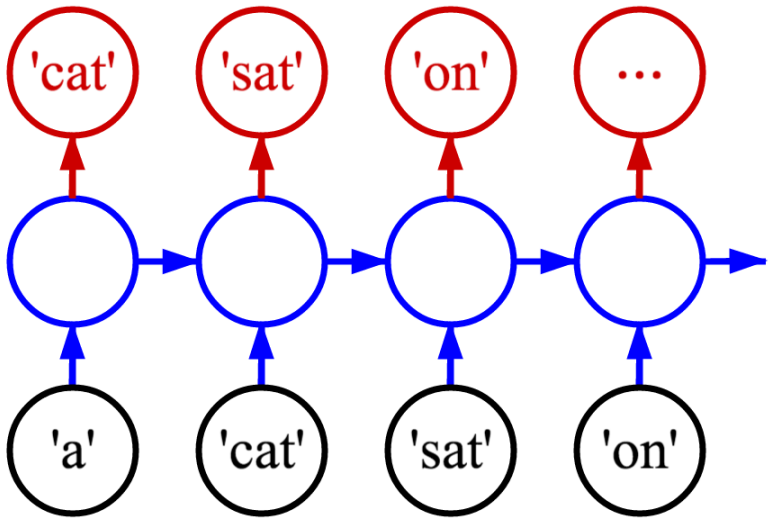
Many2Many (Synced): Language Modeling

- ***Language modeling***: predicting the next/nearby word based on the context



Many2Many (Synced): Language Modeling

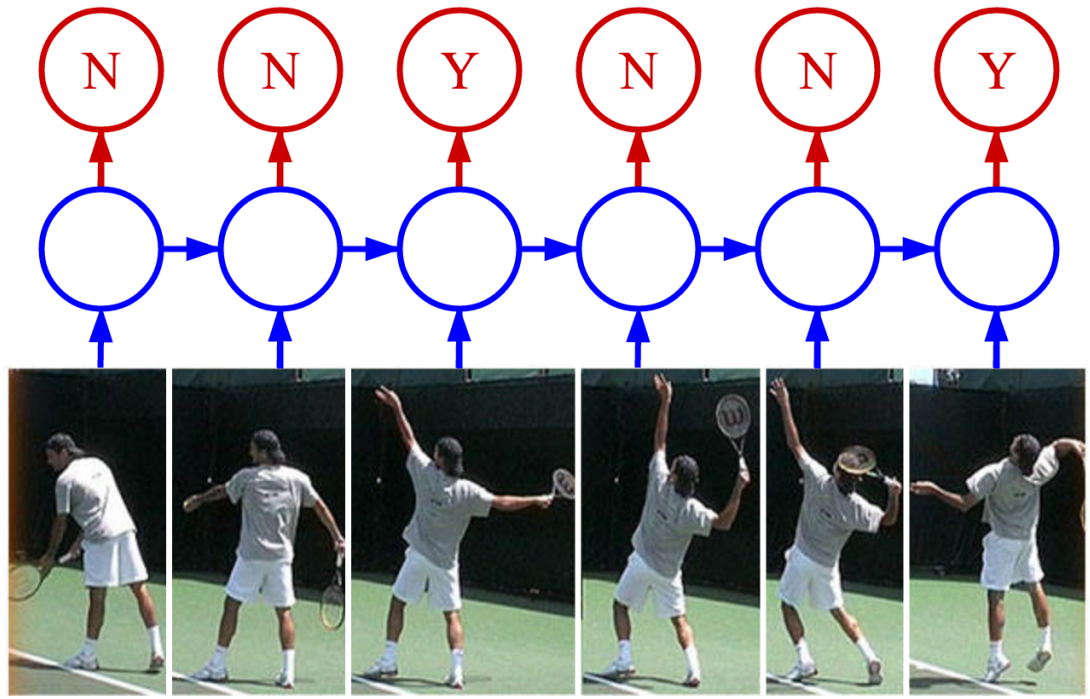
- ***Language modeling***: predicting the next/nearby word based on the context



- Latent representations of RNN provide the context

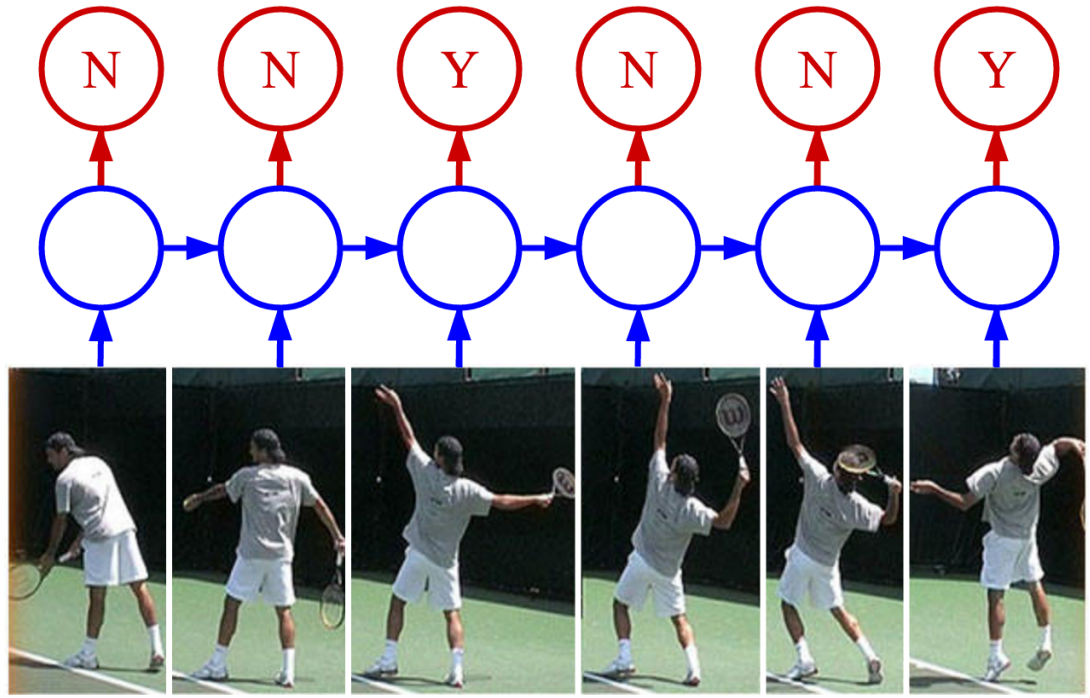
Many2Many (Synced): Video Keyframe Tagging

- Video frame annotation:



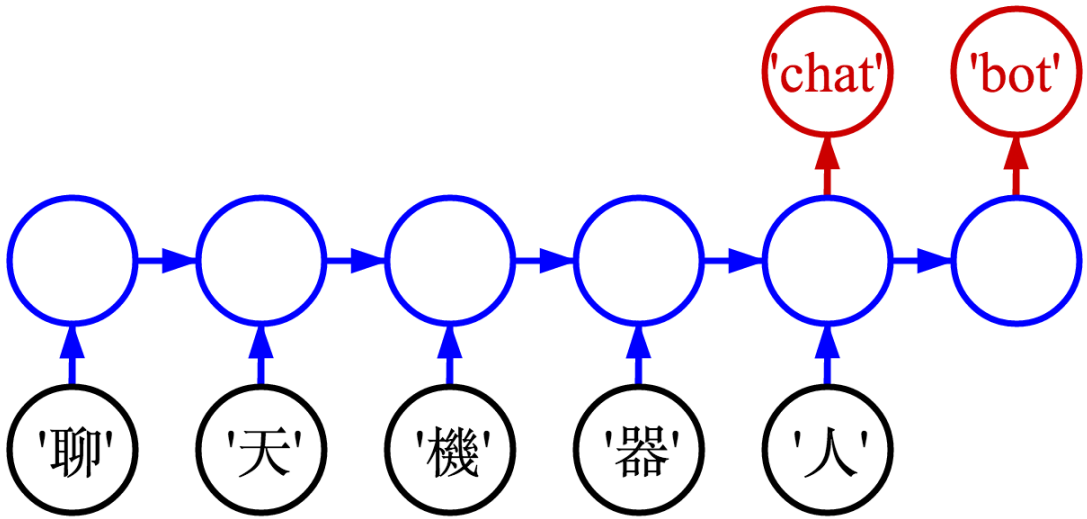
Many2Many (Synced): Video Keyframe Tagging

- Video frame annotation:



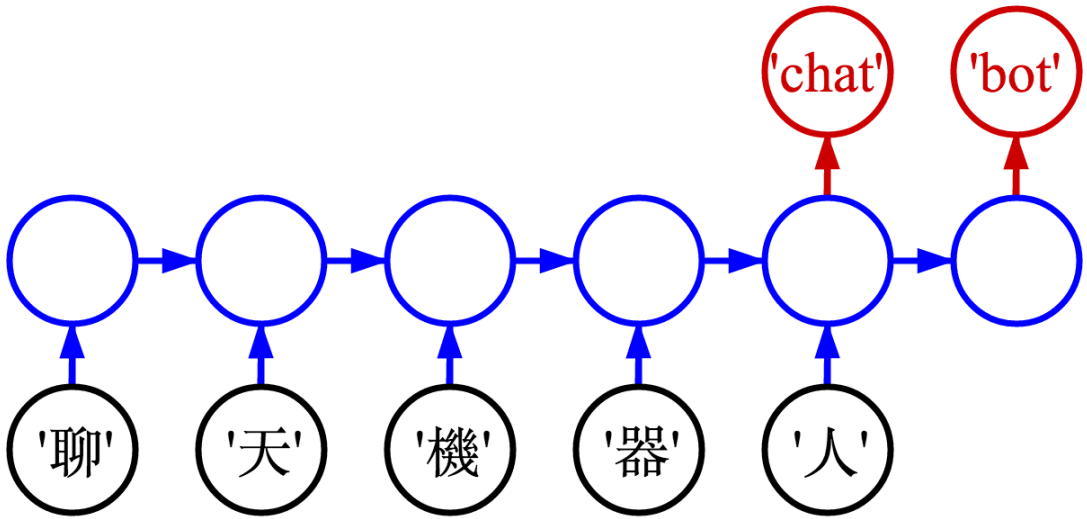
- Latent representations summarize "what's going on"

Many2Many (Unsynced): Machine Translation



- Latent representations support ***encoding*** first, and then ***decoding***
- RNN learns the structure difference

Many2Many (Unsynced): Machine Translation



- Latent representations support ***encoding*** first, and then ***decoding***
- RNN learns the structure difference
- Also called ***sequence to sequence*** learning
  - Also used in other applications, e.g., chat bots

Bidirectional RNNs

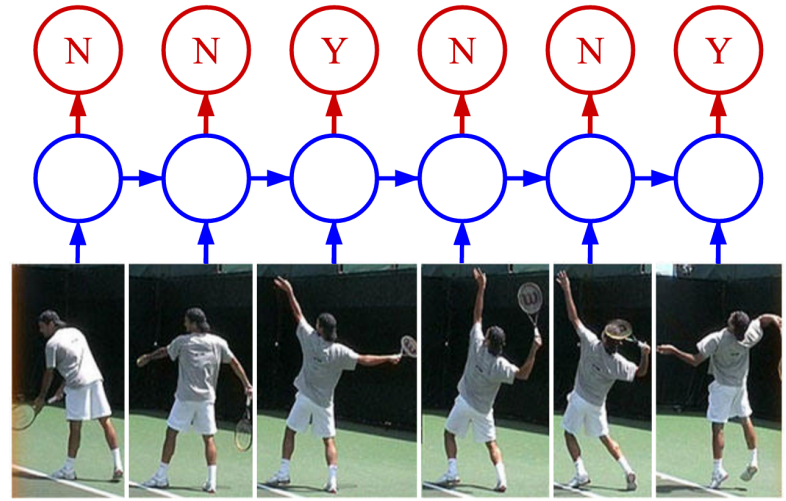
$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$

Bidirectional RNNs

$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

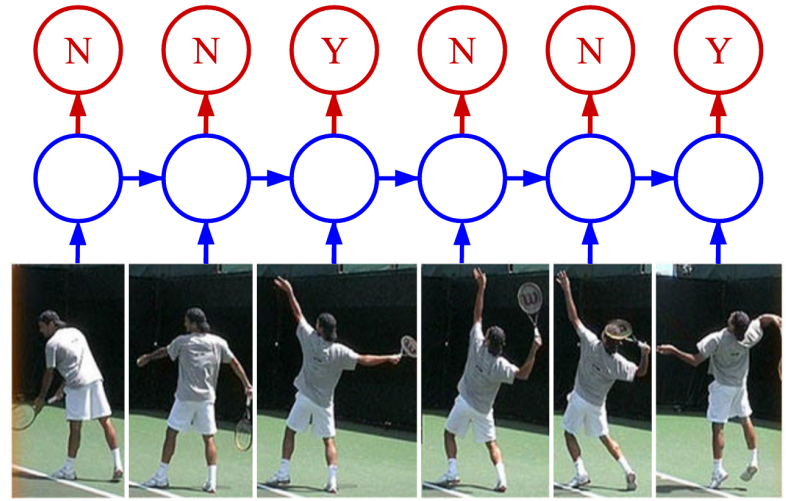
- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
- A prediction is made by considering previous frames



Bidirectional RNNs

$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
- A prediction is made by considering previous frames
- Can we take ***future*** frames into account?

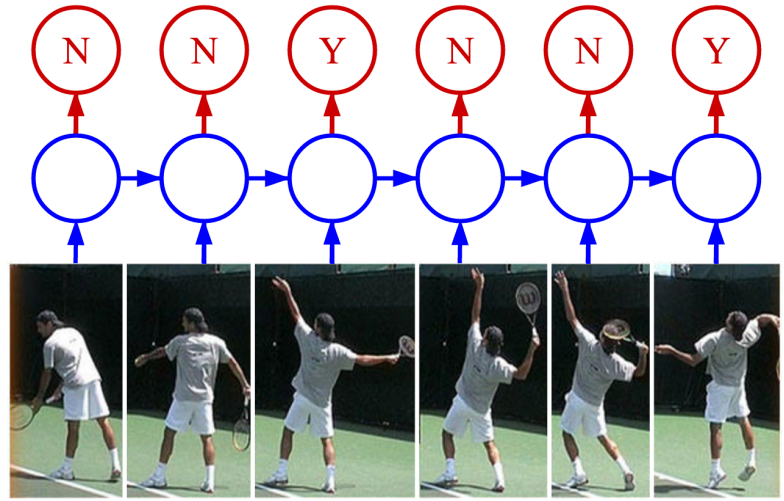


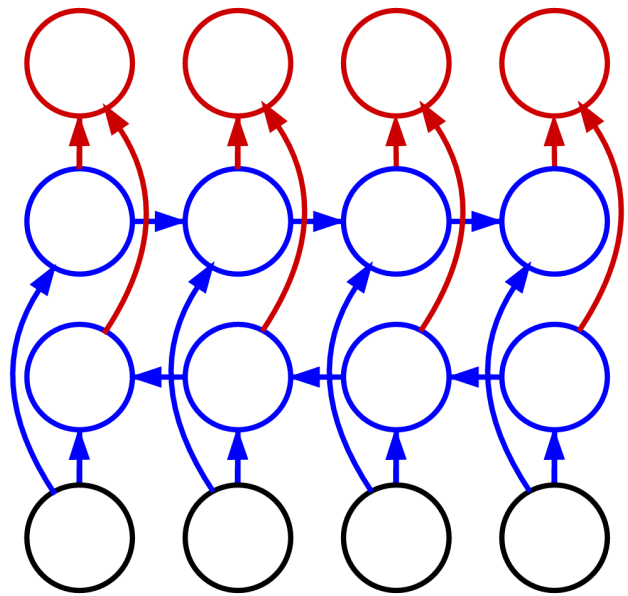
Bidirectional RNNs

$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
- A prediction is made by considering previous frames
- Can we take ***future*** frames into account?
- ***Bidirectional RNNs***: output $\mathbf{a}^{(L,t)}$ depends on both $\mathbf{a}^{(k,t)}$'s and $\tilde{\mathbf{a}}^{(k,t)}$'s

$$\tilde{\mathbf{a}}^{(k,t)} = \text{act}(\tilde{\boldsymbol{U}}^{(k)}\tilde{\mathbf{a}}^{(k,t+1)} + \tilde{\boldsymbol{W}}^{(k)}\tilde{\mathbf{a}}^{(k-1,t)})$$





Recursive RNNs I

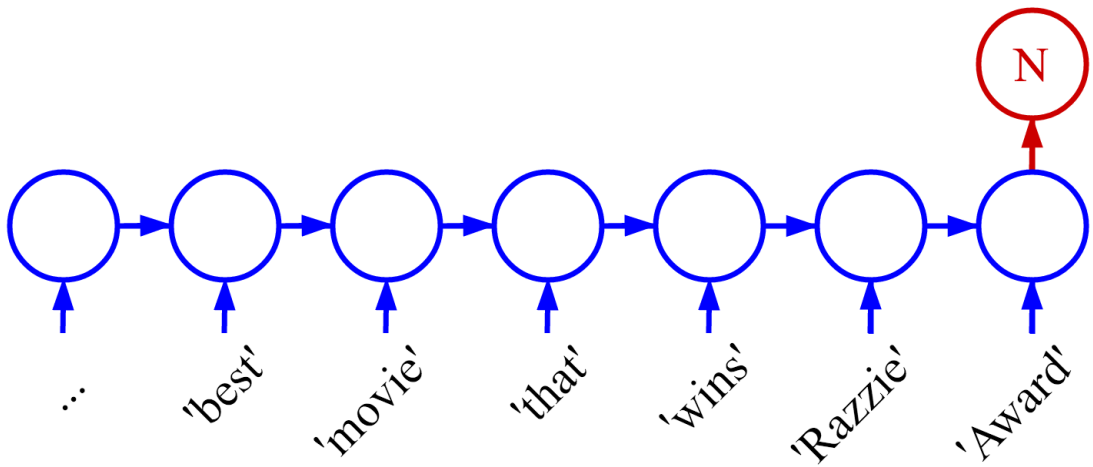
$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- The transition of hidden representation is invariant in time
  - Earlier input/representation is less important to current $\mathbf{a}^{(\cdot,t)}$

Recursive RNNs I

$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

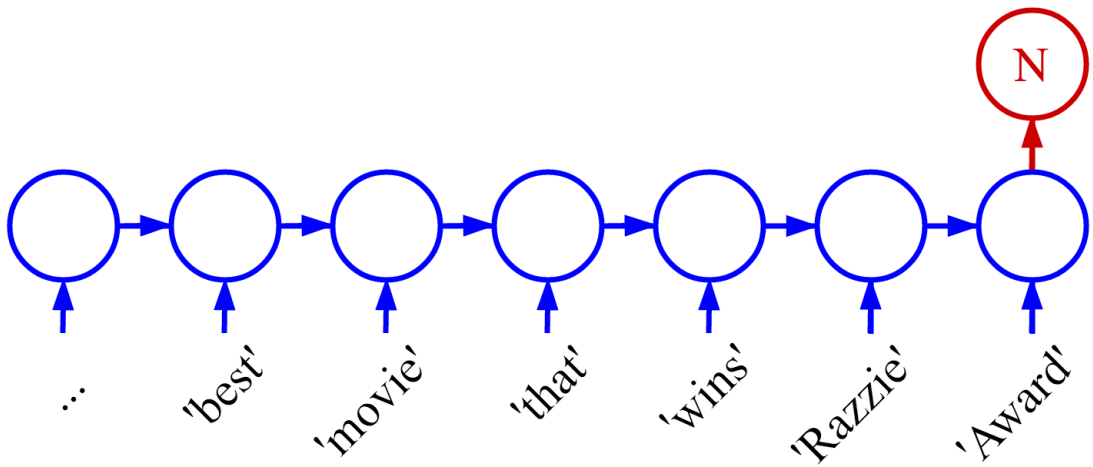
- The transition of hidden representation is invariant in time
  - Earlier input/representation is less important to current $\mathbf{a}^{(\cdot,t)}$
- "Razzie" has less effect if it is far away from the prediction



Recursive RNNs I

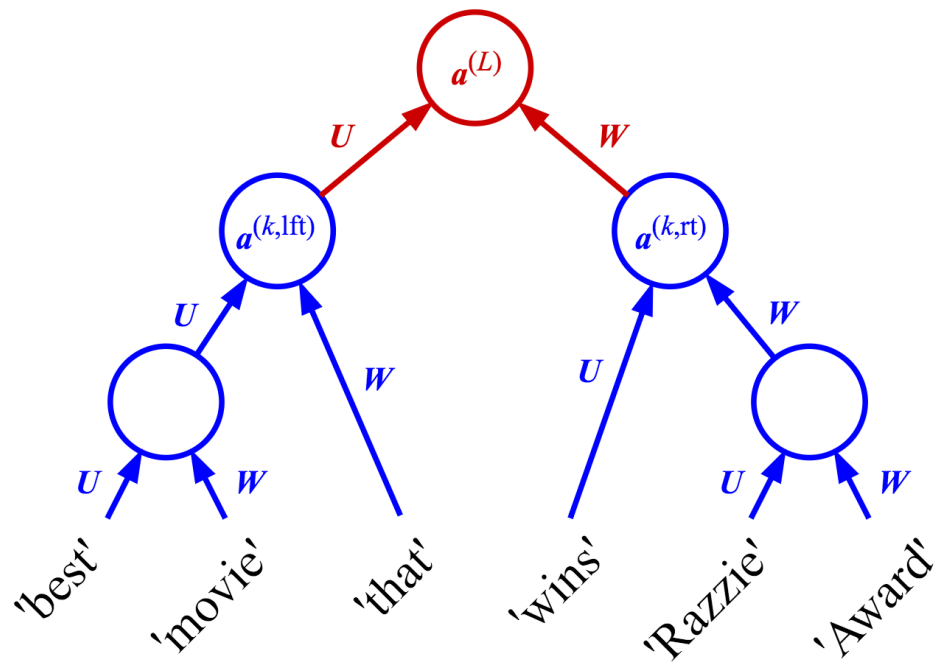
$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- The transition of hidden representation is invariant in time
  - Earlier input/representation is less important to current $\mathbf{a}^{(\cdot,t)}$
- "Razzie" has less effect if it is far away from the prediction
- In some applications, transitions are invariant in terms of other concepts



Recursive RNNs II

- In natural language processing (NLP), we can parse the input sentence $\boldsymbol{X}^{(n)}$ into a tree
  - Following gramma rules

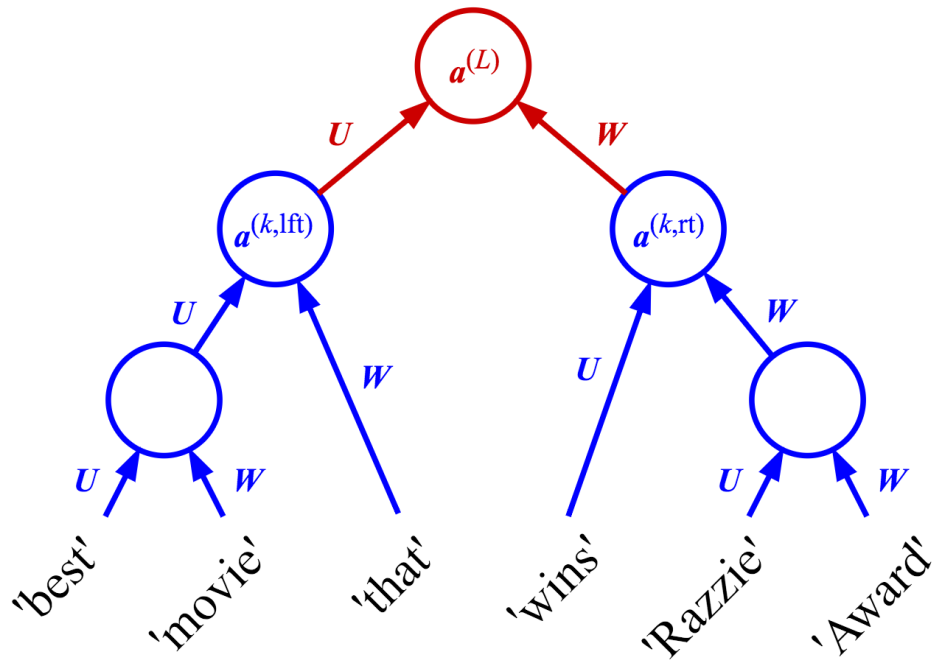


Recursive RNNs II

- In natural language processing (NLP), we can parse the input sentence $\boldsymbol{X}^{(n)}$ into a tree
  - Following gramma rules
- ***Recursive RNNs***: "subtree merges" are invariant

$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}\mathbf{a}^{(k-1,\text{ left })} + \boldsymbol{W}\mathbf{a}^{(k-1,\text{ right })})$$

  - $\boldsymbol{U}$ and $\boldsymbol{W}$ are shared recursively in subtrees

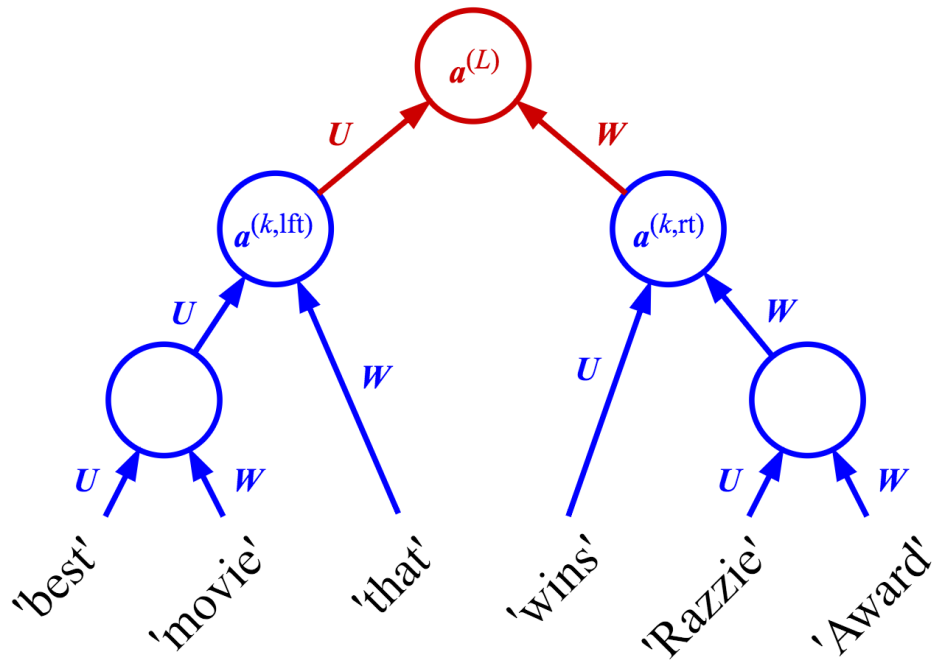


Recursive RNNs II

- In natural language processing (NLP), we can parse the input sentence $\boldsymbol{X}^{(n)}$ into a tree
  - Following gramma rules
- ***Recursive RNNs***: "subtree merges" are invariant

$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}\mathbf{a}^{(k-1,\text{ left })} + \boldsymbol{W}\mathbf{a}^{(k-1,\text{ right })})$$

  - $\boldsymbol{U}$ and $\boldsymbol{W}$ are shared recursively in subtrees
- Given sentence length T, $\mathbf{a}^{(L)}$ and $\mathbf{a}^{(1,\cdot)}$ can be $O(\log T)$ away in the best case



Outline

Cost Function of Vanilla RNNs

- Parameters to learn: $\Theta = \{\boldsymbol{W}^{(k)}, \boldsymbol{U}^{(k)}\}_k$ (bias terms omitted)
- Maximum likelihood:

$$\begin{aligned}
&\arg\min_{\Theta} C(\Theta) \\
&= \arg\min_{\Theta} -\log \mathrm{P}(\mathbf{X} \mid \Theta) \\
&= \arg\min_{\Theta} -\textstyle\sum_{n,t} \log \mathrm{P}(\boldsymbol{y}^{(n,t)} \mid \boldsymbol{x}^{(n,t)}, \cdots, \boldsymbol{x}^{(n,1)}, \Theta) \\
&= \arg\min_{\Theta} -\textstyle\sum_{n,t} C^{(n,t)}(\Theta)
\end{aligned}$$

  - $\mathbf{y}^{(t)}$ depends only on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$

Cost Function of Vanilla RNNs

- Parameters to learn: $\Theta = \{\boldsymbol{W}^{(k)}, \boldsymbol{U}^{(k)}\}_k$ (bias terms omitted)
- Maximum likelihood:

$$\begin{aligned}
&\arg\min_\Theta C(\Theta) \\
&= \arg\min_\Theta -\log \mathrm{P}(\mathbf{X} \mid \Theta) \\
&= \arg\min_\Theta -\textstyle\sum_{n,t} \log \mathrm{P}(\boldsymbol{y}^{(n,t)} \mid \boldsymbol{x}^{(n,t)}, \cdots, \boldsymbol{x}^{(n,1)}, \Theta) \\
&= \arg\min_\Theta -\textstyle\sum_{n,t} C^{(n,t)}(\Theta)
\end{aligned}$$

  - $\mathbf{y}^{(t)}$ depends only on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$
- For example, in binary classification:
- Assuming $\mathrm{P}(\mathbf{y}^{(n,t)} = 1 \mid \boldsymbol{x}^{(n,t)}, \cdots, \boldsymbol{x}^{(n,1)}) \sim \mathrm{Bernoulli}(\rho^{(t)})$, we have

$$C^{(n,t)}(\Theta) = (\boldsymbol{a}^{(L,t)})^{y^{(n,t)}} (1 - \boldsymbol{a}^{(L,t)})^{(1-y^{(n,t)})}$$

  - $\boldsymbol{a}^{(L,t)} = \rho^{(t)}$ are based on $\boldsymbol{a}^{(\cdot,t)}$'s, which summarize $\boldsymbol{x}^{(n,t)}, \cdots, \boldsymbol{x}^{(n,1)}$

Outline

SGD-based Training

- RNN optimization problem can be solved using SGD:

$$\Theta^{(s+1)} \leftarrow \Theta^{(s)} - \eta \nabla_{\Theta} \sum_{n,t} C^{(n,t)}(\Theta^{(s)})$$

- Let $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$, our goal is to evaluate $\frac{\partial c^{(n,t)}}{\partial U_{i,j}^{(k)}}$ and $\frac{\partial c^{(n,t)}}{\partial W_{i,j}^{(k)}}$
- Evaluation of $\frac{\partial c^{(n,t)}}{\partial W_{i,j}^{(k)}}$ is similar to that in DNNs and omitted
- We focus on:

$$\frac{\partial c^{(n,t)}}{\partial U_{i,j}^{(k)}} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} \cdot \frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}} = \delta_j^{(k,t)} \frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}}$$

Forward Pass through Time

- The second term: $\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}}$

Forward Pass through Time

- The second term: $\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}}$
- We have $z_j^{(k,t)} = \sum_i W_{i,j}^{(k)} a_i^{(k-1,t)} + \sum_i U_{i,j}^{(k)} a_i^{(k,t-1)}$ and

$$\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}} = a_i^{(k,t-1)}$$

Forward Pass through Time

- The second term: $\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}}$
- We have $z_j^{(k,t)} = \sum_i W_{i,j}^{(k)} a_i^{(k-1,t)} + \sum_i U_{i,j}^{(k)} a_i^{(k,t-1)}$ and

$$\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}} = a_i^{(k,t-1)}$$

- We can get all second terms starting from the most shallow layer and ***earliest time***

Backward Pass through Time I

- The first term (error signal): $\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}}$
- We have

$$\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k,t)}}$$

Backward Pass through Time I

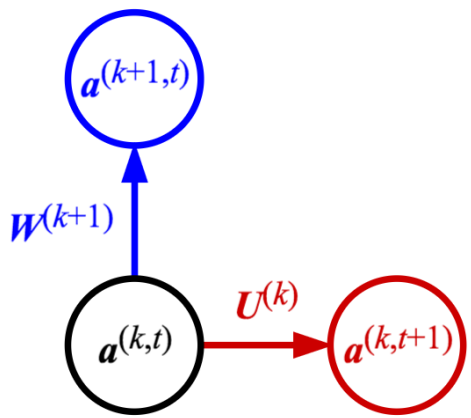
- The first term (error signal): $\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}}$
- We have

$$\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \text{act}'(z_j^{(k,t)})$$

Backward Pass through Time I

- The first term (error signal): $\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}}$
- We have

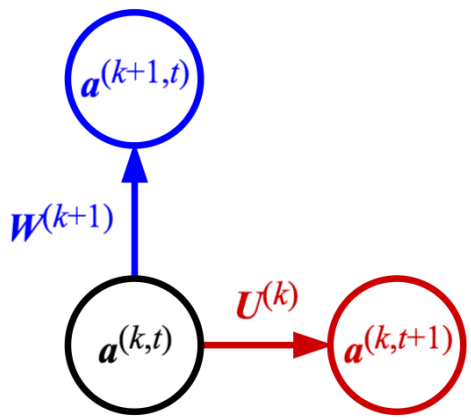
$$\begin{aligned}\delta_j^{(k,t)} &= \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \text{act}'(z_j^{(k,t)}) \\ &= \left(\sum_s \frac{\partial c^{(n,t)}}{\partial z_s^{(k+1,t)}} \cdot \frac{\partial z_s^{(k+1,t)}}{\partial a_j^{(k,t)}} + \sum_s \frac{\partial c^{(n,t)}}{\partial z_s^{(k,t+1)}} \cdot \frac{\partial z_s^{(k,t+1)}}{\partial a_j^{(k,t)}} \right) \text{act}'(z_j^{(k,t)})\end{aligned}$$



Backward Pass through Time I

- The first term (error signal): $\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}}$
- We have

$$
\begin{aligned}
\delta_j^{(k,t)} &= \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \mathrm{act}'(z_j^{(k,t)}) \\
&= \left(\sum_s \frac{\partial c^{(n,t)}}{\partial z_s^{(k+1,t)}} \cdot \frac{\partial z_s^{(k+1,t)}}{\partial a_j^{(k,t)}} + \sum_s \frac{\partial c^{(n,t)}}{\partial z_s^{(k,t+1)}} \cdot \frac{\partial z_s^{(k,t+1)}}{\partial a_j^{(k,t)}} \right) \mathrm{act}'(z_j^{(k,t)}) \\
&= \left(\sum_s \delta_s^{(k+1,t)} W_{j,s}^{(k+1)} + \sum_s \delta_s^{(k,t+1)} U_{j,s}^{(k)} \right) \mathrm{act}'(z_j^{(k,t)})
\end{aligned}
$$



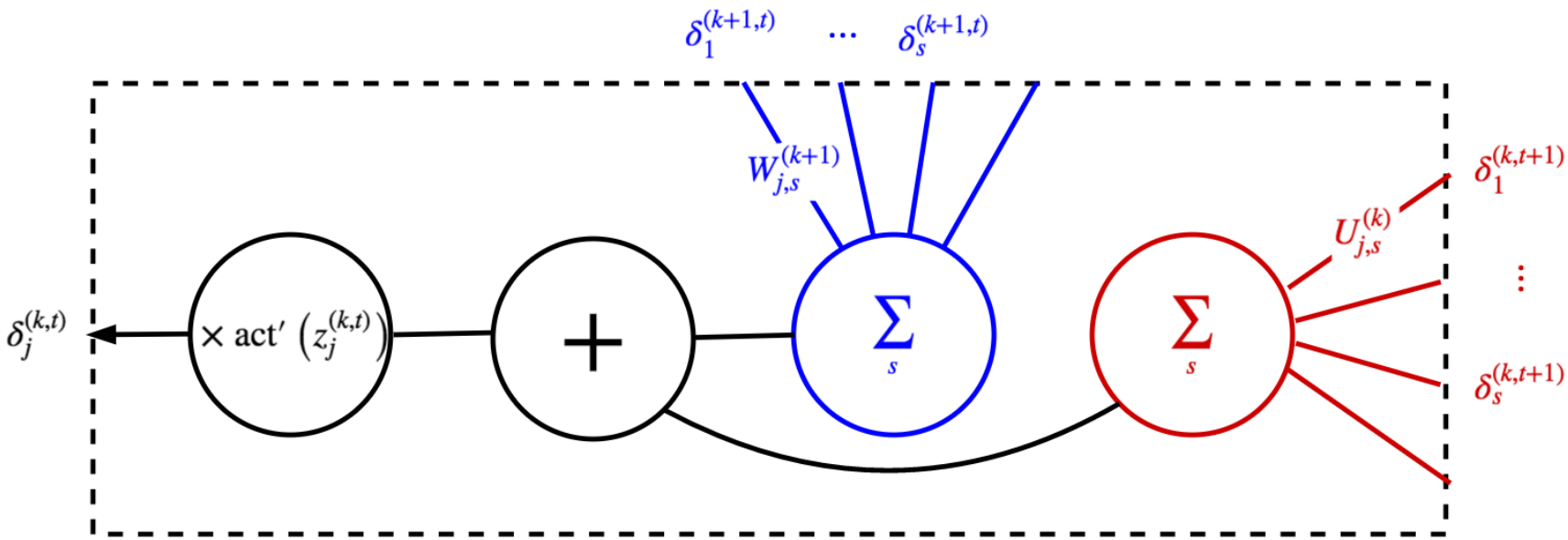
Backward Pass through Time II

$$\delta_j^{(k,t)} = \left(\sum_s \delta_s^{(k+1,t)} W_{j,s}^{(k+1)} + \sum_s \delta_s^{(k,t+1)} U_{j,s}^{(k)} \right) \text{act}'(z_j^{(k,t)})$$

Backward Pass through Time II

$$\delta_j^{(k,t)} = \left(\sum_s \delta_s^{(k+1,t)} W_{j,s}^{(k+1)} + \sum_s \delta_s^{(k,t+1)} U_{j,s}^{(k)} \right) \text{act}'(z_j^{(k,t)})$$

- We can evaluate all $\delta_j^{(k,t)}$'s starting from the deepest layer and ***latest time***



Backprop through Time (BPTT)

- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time

Backprop through Time (BPTT)

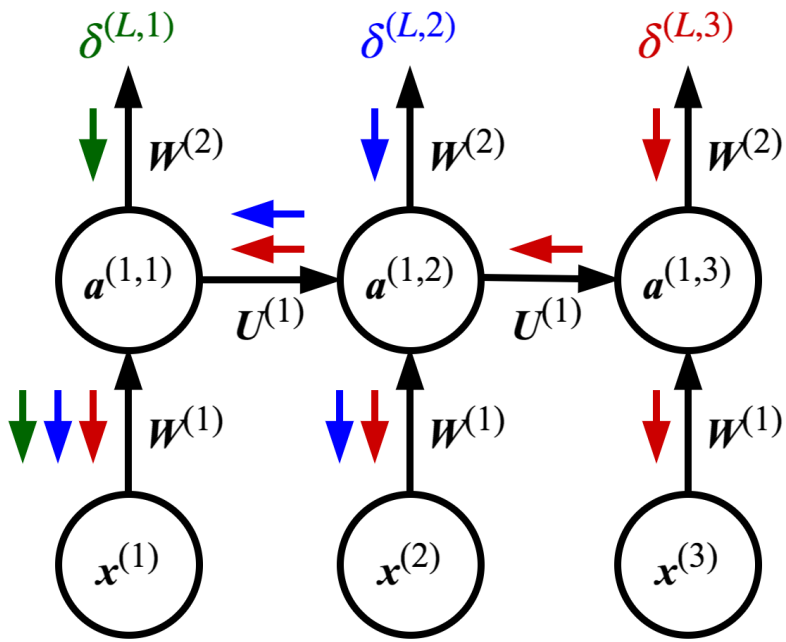
- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time
- However, for each sequence n, we have multiple $c^{(n,t)}$'s at different t's
  - Their gradient need to be summed

Backprop through Time (BPTT)

- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time
- However, for each sequence n, we have multiple $c^{(n,t)}$'s at different t's
  - Their gradient need to be summed
- For different $c^{(n,\cdot)}$'s, the forward pass can be shared

Backprop through Time (BPTT)

- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time
- However, for each sequence n, we have multiple $c^{(n,t)}$'s at different t's
  - Their gradient need to be summed
- For different $c^{(n,\cdot)}$'s, the forward pass can be shared
- ***BPTT***: single forward pass, ***multiple*** backward passes



Outline

Long-Term Dependencies

- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top} \boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top} \boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

- Backward pass:

$$\delta^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)} \delta^{(k,t+1)} + \boldsymbol{W}^{(k+1)} \delta^{(k+1,t)})$$

Long-Term Dependencies

- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

- Backward pass:

$$\delta^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\delta^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\delta^{(k+1,t)})$$

- The dependency between $\boldsymbol{a}^{(k,i)}$ (resp. $\delta^{(k,i)}$) and $\boldsymbol{a}^{(k,j)}$ (resp. $\delta^{(k,j)}$) are maintained by $\left(\boldsymbol{U}^{(k)}\right)^{(j-i)}$
  - Ignoring activation function and depth, we have $\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{j-i}\boldsymbol{a}^{(k,i)}$ and $\delta^{(k,i)} = (\boldsymbol{U}^{(k)})^{j-i}\delta^{(k,j)}$

Long-Term Dependencies

- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

- Backward pass:

$$\delta^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\delta^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\delta^{(k+1,t)})$$

- The dependency between $\boldsymbol{a}^{(k,i)}$ (resp. $\delta^{(k,i)}$) and $\boldsymbol{a}^{(k,j)}$ (resp. $\delta^{(k,j)}$) are maintained by $\left(\boldsymbol{U}^{(k)}\right)^{(j-i)}$
  - Ignoring activation function and depth, we have $\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{j-i}\boldsymbol{a}^{(k,i)}$ and $\delta^{(k,i)} = (\boldsymbol{U}^{(k)})^{j-i}\delta^{(k,j)}$
- If $\boldsymbol{a}^{(k,i)}$ and $\boldsymbol{a}^{(k,j)}$ are far away in time, their long-term dependency causes optimization problems

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

$$\boldsymbol{a}^{(k,j)} = \left(\boldsymbol{U}^{(k)\top}\right)^{(j-i)} \boldsymbol{a}^{(k,i)} \text{ and } \boldsymbol{\delta}^{(k,i)} = \left(\boldsymbol{U}^{(k)}\right)^{(j-i)} \boldsymbol{\delta}^{(k,j)}$$

- Given eigendecomposition: $\boldsymbol{U}^{(k)} = \boldsymbol{Q} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}} \end{bmatrix} \boldsymbol{Q}^\top$

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

$$\boldsymbol{a}^{(k,j)} = \left(\boldsymbol{U}^{(k)\top}\right)^{(j-i)} \boldsymbol{a}^{(k,i)} \text{ and } \boldsymbol{\delta}^{(k,i)} = \left(\boldsymbol{U}^{(k)}\right)^{(j-i)} \boldsymbol{\delta}^{(k,j)}$$

- Given eigendecomposition: $\boldsymbol{U}^{(k)} = \boldsymbol{Q} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}} \end{bmatrix} \boldsymbol{Q}^\top$

- We have: $\left(\boldsymbol{U}^{(k)}\right)^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

$$\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{(j-i)} \boldsymbol{a}^{(k,i)} \text{ and } \boldsymbol{\delta}^{(k,i)} = (\boldsymbol{U}^{(k)})^{(j-i)} \boldsymbol{\delta}^{(k,j)}$$

- Given eigendecomposition: $\boldsymbol{U}^{(k)} = \boldsymbol{Q} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}} \end{bmatrix} \boldsymbol{Q}^\top$
- We have: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$
- When $j-i$ is large, λ_s^{j-i} is either very large or small
  - $\lambda_s = 1.01 \Rightarrow \lambda_s^{1000} \approx 20,000$
  - $\lambda_s = 0.99 \Rightarrow \lambda_s^{1000} \approx 0$

Exploding/Vanishing Gradient Problem

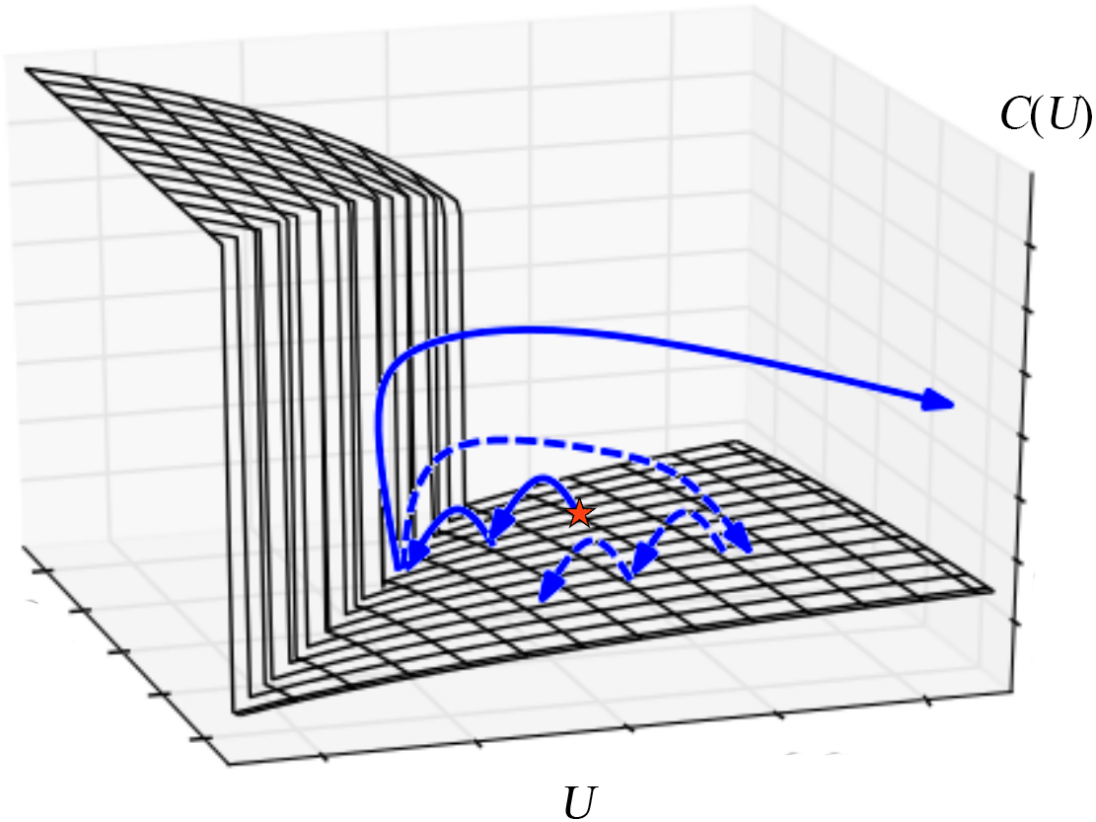
- Ignoring activation function and depth:

$$\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{(j-i)} \boldsymbol{a}^{(k,i)} \text{ and } \boldsymbol{\delta}^{(k,i)} = (\boldsymbol{U}^{(k)})^{(j-i)} \boldsymbol{\delta}^{(k,j)}$$

- Given eigendecomposition: $\boldsymbol{U}^{(k)} = \boldsymbol{Q} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}} \end{bmatrix} \boldsymbol{Q}^\top$
- We have: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$
- When $j-i$ is large, λ_s^{j-i} is either very large or small
  - $\lambda_s = 1.01 \Rightarrow \lambda_s^{1000} \approx 20,000$
  - $\lambda_s = 0.99 \Rightarrow \lambda_s^{1000} \approx 0$
- Exploding or vanishing gradients!

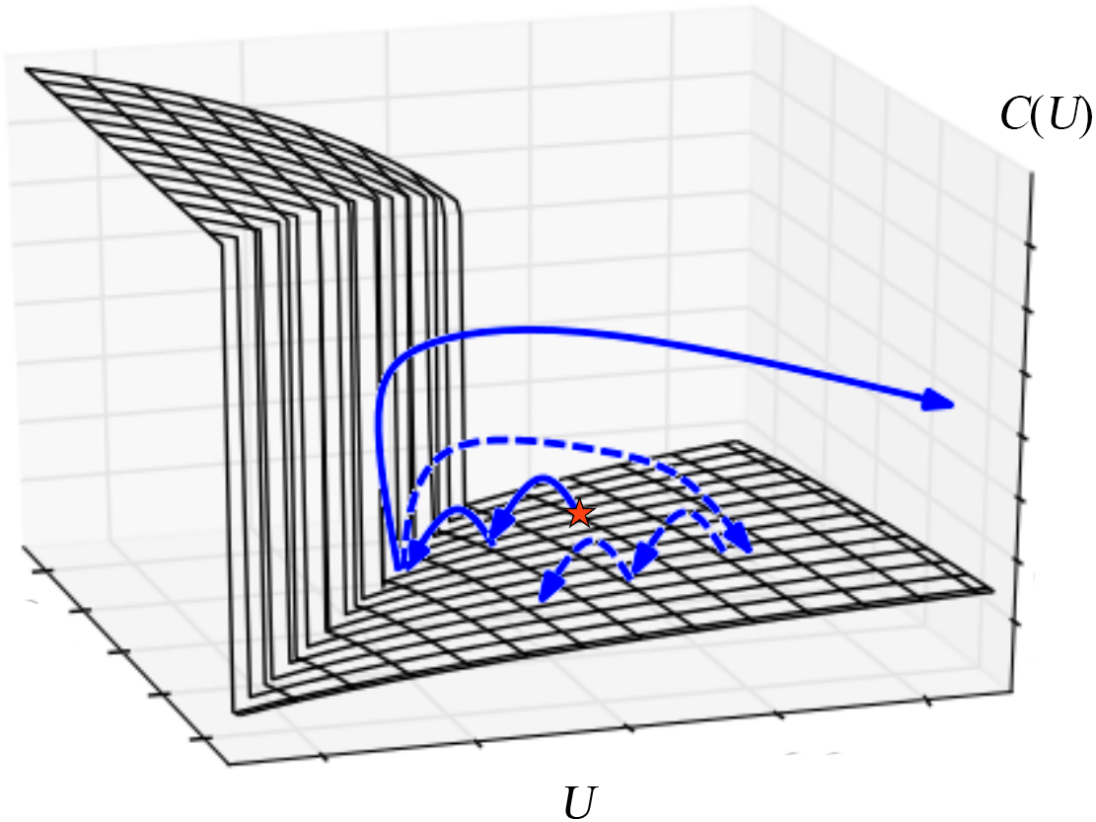
Cost Surface

- The cost surface of C is either very flat or steep
- Hard for gradient-based optimization



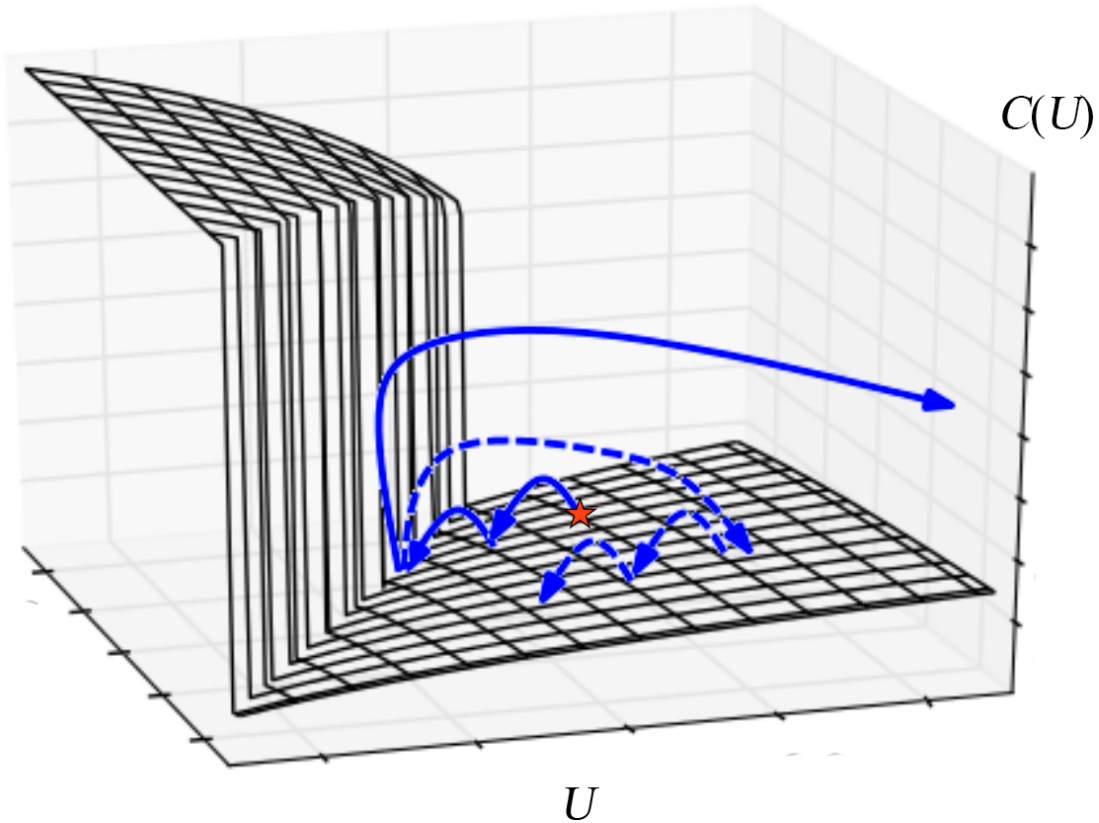
Cost Surface

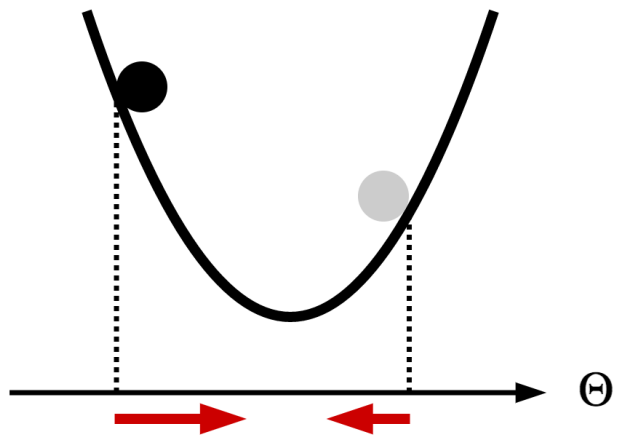
- The cost surface of C is either very flat or steep
- Hard for gradient-based optimization
- Optimization techniques?



Nesterov Momentum

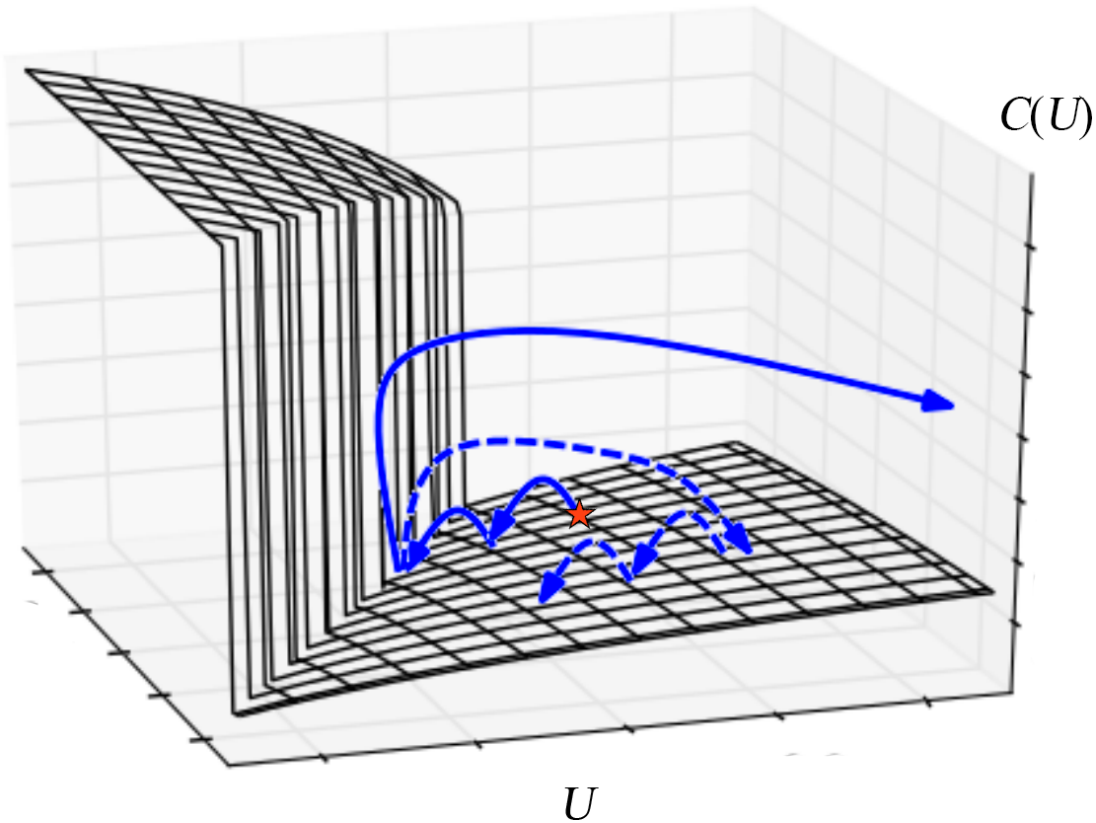
- Use Nesterov momentum to "brake" before hitting the wall





Gradient Clipping

- A simple way is to avoid the exploding gradient problem is to ***clip*** a gradient if it exceeds a predefined threshold
- Very effective in practice

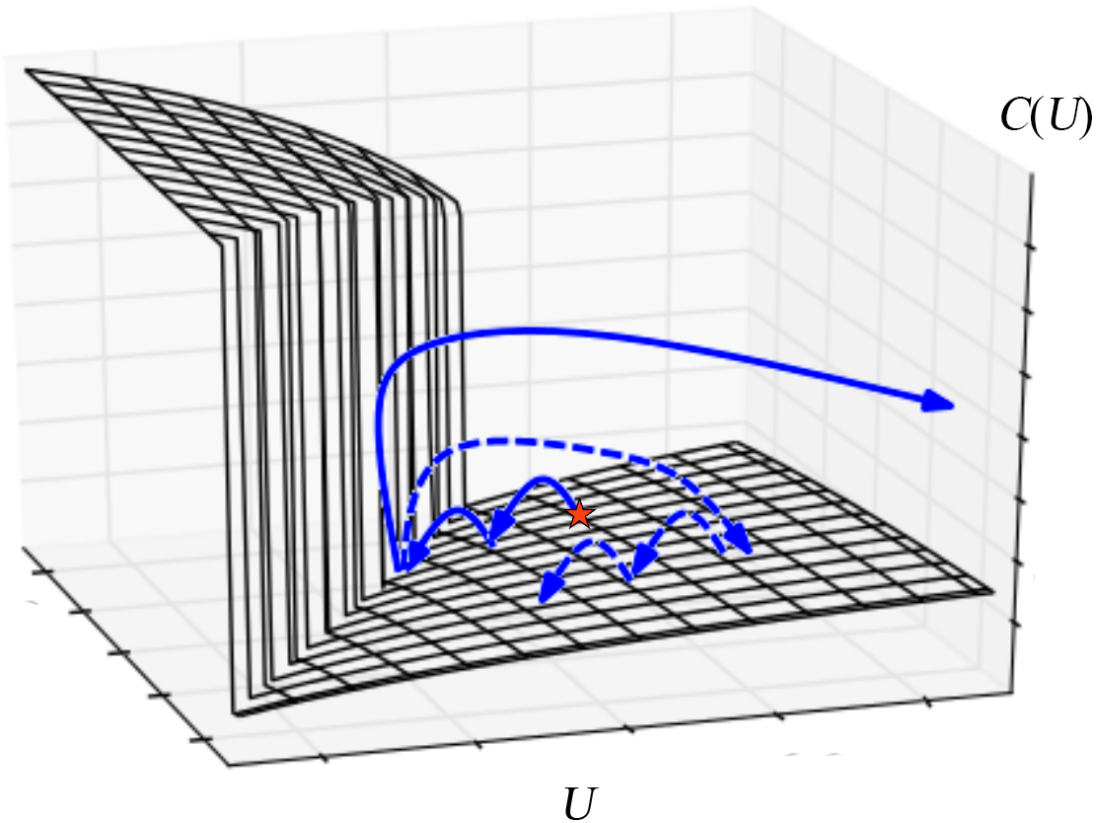


RMS Prop

- Adaptive learning rate based on statistics of recent gradients:

$$\boldsymbol{r}^{(t+1)} \leftarrow \lambda \boldsymbol{r}^{(t)} + (1-\lambda)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{r}^{(t+1)}}} \odot \boldsymbol{g}^{(t)}$$



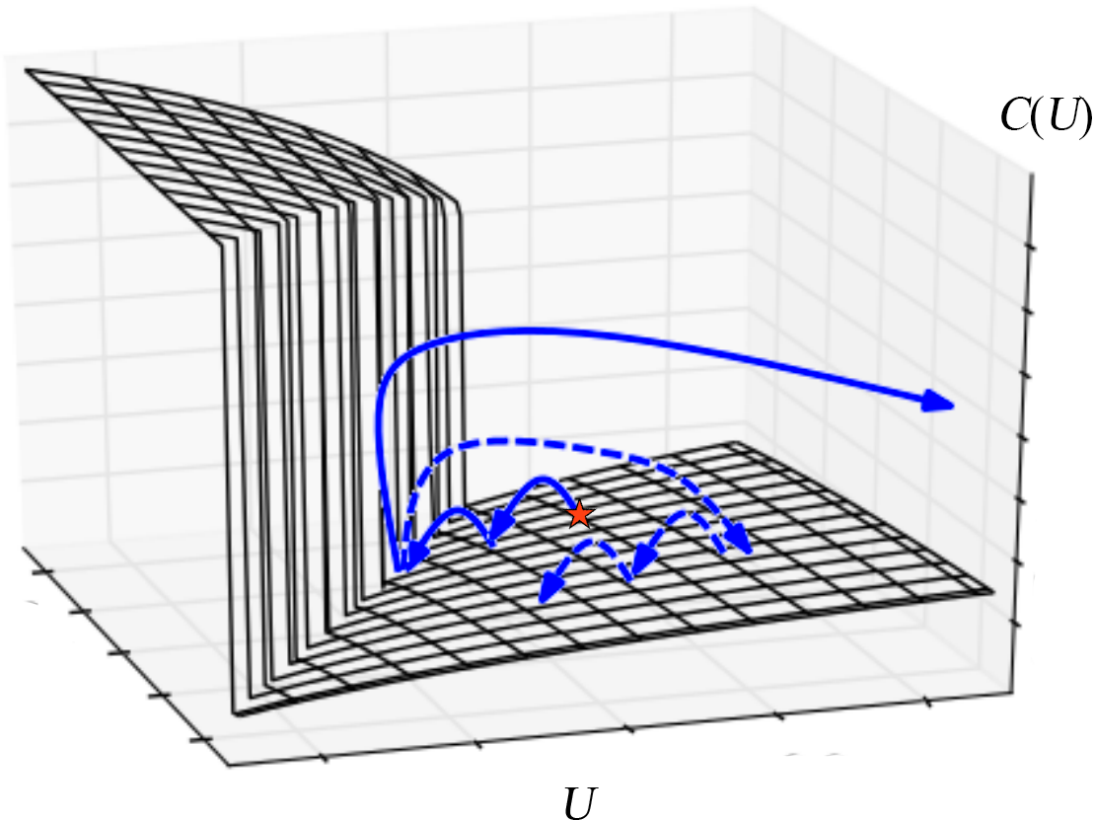
RMS Prop

- Adaptive learning rate based on statistics of recent gradients:

$$\boldsymbol{r}^{(t+1)} \leftarrow \lambda \boldsymbol{r}^{(t)} + (1-\lambda)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{r}^{(t+1)}}} \odot \boldsymbol{g}^{(t)}$$

- Reduce λ



Outline

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?
- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

Backward pass:

$$\boldsymbol{\delta}^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\boldsymbol{\delta}^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\boldsymbol{\delta}^{(k+1,t)})$$

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?
- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \text{act}(\boldsymbol{z}^{(k,t)})$$

  Backward pass:

$$\delta^{(k,t)} \leftarrow \text{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\delta^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\delta^{(k+1,t)})$$

- Sigmoid activation:
  - Bounded range in the forward pass
  - $\text{act}'(\cdot) < 1$ in the backward pass

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?
- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

  Backward pass:

$$\delta^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\delta^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\delta^{(k+1,t)})$$

- Sigmoid activation:
  - Bounded range in the forward pass
  - $\mathrm{act}'(\cdot) < 1$ in the backward pass
- Mitigates the exploding (but not vanishing) gradient problem for $\boldsymbol{U}^{(k)}$'s

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?
- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

  Backward pass:

$$\delta^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\delta^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\delta^{(k+1,t)})$$

- Sigmoid activation:
  - Bounded range in the forward pass
  - $\mathrm{act}'(\cdot) < 1$ in the backward pass
- Mitigates the exploding (but not vanishing) gradient problem for $\boldsymbol{U}^{(k)}$'s
- Introduces vanishing gradients of $\boldsymbol{W}^{(k)}$'s

Learning Unitary $\boldsymbol{U}^{(k)}$'s

- Long-term dependency: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{\boldsymbol{D}^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$

Learning Unitary $\boldsymbol{U}^{(k)}$'s

- Long-term dependency: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{\boldsymbol{D}^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$
- Why not make $\boldsymbol{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?

Learning Unitary $\boldsymbol{U}^{(k)}$'s

- Long-term dependency: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$
- Why not make $\boldsymbol{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [5] propose IRNN:
  - ***Initializes*** $\boldsymbol{U}^{(k)} = \boldsymbol{I}$ in SGD
  - Uses ***ReLU*** hidden units

Learning Unitary $\boldsymbol{U}^{(k)}$'s

- Long-term dependency: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$
- Why not make $\boldsymbol{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [5] propose IRNN:
  - ***Initializes*** $\boldsymbol{U}^{(k)} = \boldsymbol{I}$ in SGD
  - Uses ***ReLU*** hidden units
  - Empirically, requires a very small learning rate (e.g., 10^{-8}) to work well
  - Simple, but very slow

Learning Unitary $\boldsymbol{U}^{(k)}$'s

- Long-term dependency: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$
- Why not make $\boldsymbol{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [5] propose IRNN:
  - ***Initializes*** $\boldsymbol{U}^{(k)} = \boldsymbol{I}$ in SGD
  - Uses ***ReLU*** hidden units
  - Empirically, requires a very small learning rate (e.g., 10^{-8}) to work well
  - Simple, but very slow
- Krueger et al. [4] add a term $\sum_{k,t}(\|\boldsymbol{a}^{(k,t)}\| - \|\boldsymbol{a}^{(k,t-1)}\|)^2$ to IRNN cost to stabilize the norms of $\boldsymbol{a}^{(k,t)}$'s in time

Learning Unitary $\boldsymbol{U}^{(k)}$'s

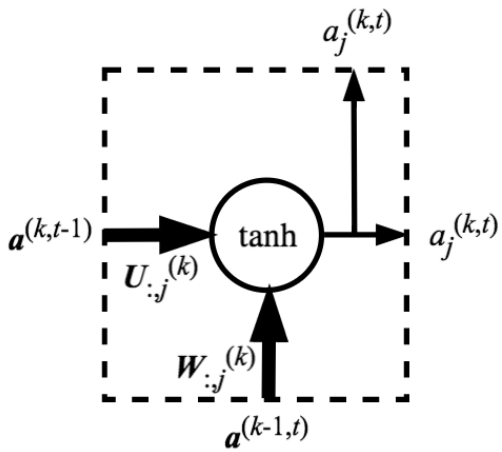
- Long-term dependency: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$
- Why not make $\boldsymbol{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [5] propose IRNN:
  - ***Initializes*** $\boldsymbol{U}^{(k)} = \boldsymbol{I}$ in SGD
  - Uses ***ReLU*** hidden units
  - Empirically, requires a very small learning rate (e.g., 10^{-8}) to work well
  - Simple, but very slow
- Krueger et al. [4] add a term $\sum_{k,t}(\|\boldsymbol{a}^{(k,t)}\| - \|\boldsymbol{a}^{(k,t-1)}\|)^2$ to IRNN cost to stabilize the norms of $\boldsymbol{a}^{(k,t)}$'s in time
- Bengio et al. [1] propose uRNN that learns unitary $\boldsymbol{U}^{(k)}$'s explicitly

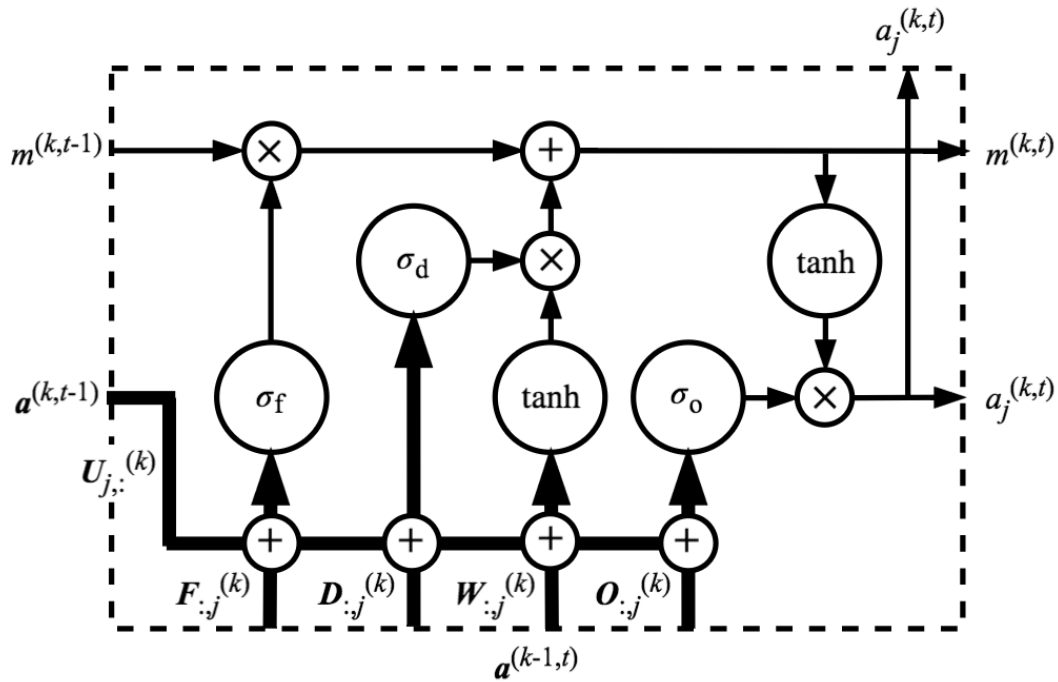
Long Short-Term Memory (LSTM)

- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly

Long Short-Term Memory (LSTM)

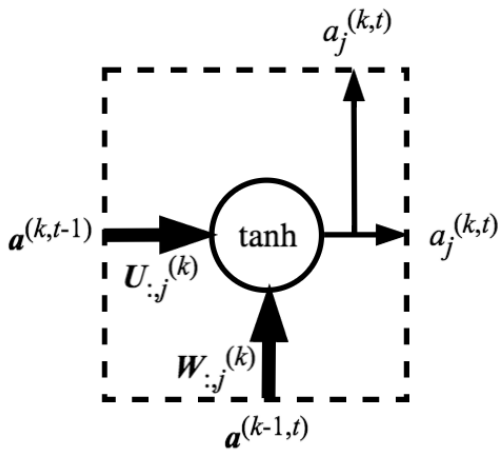
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:

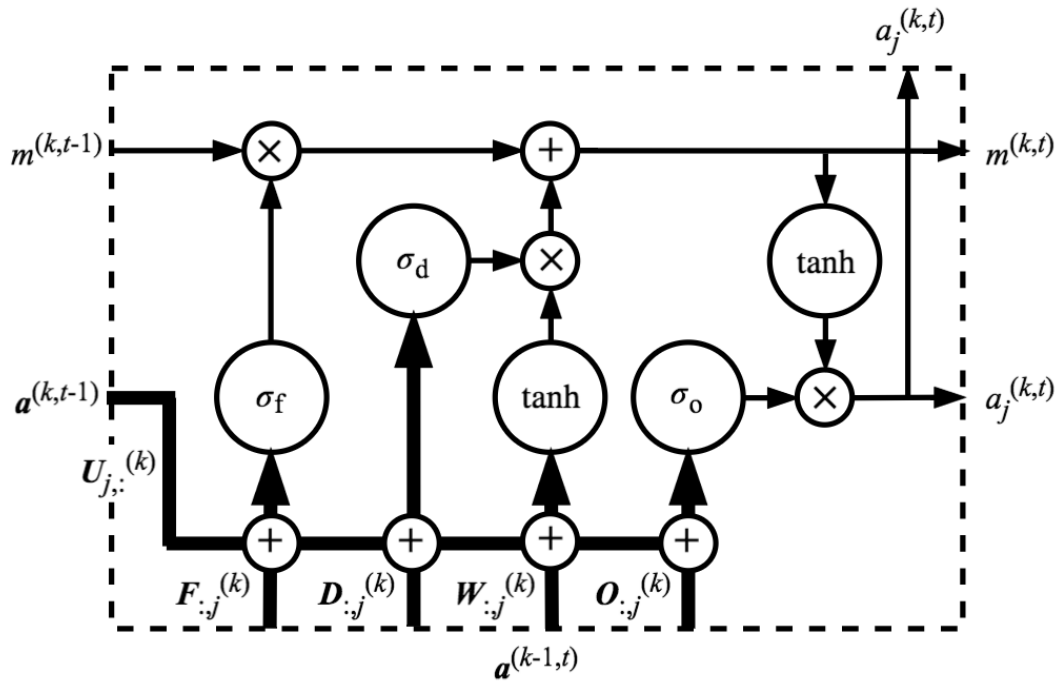




Long Short-Term Memory (LSTM)

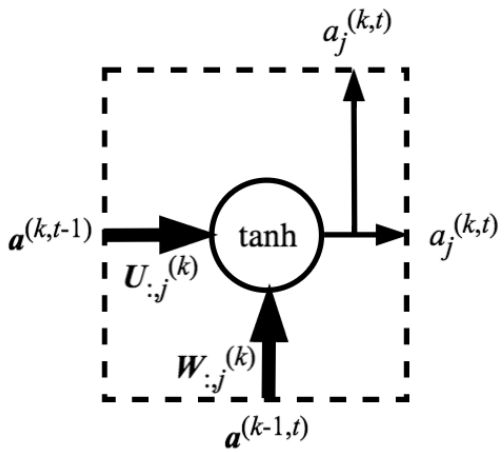
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:
  - First $\tanh$ activation to be added into the ***memory cell*** $m^{(k,t)}$
  - Second $\tanh$ activation as the final activation $a_j^{(k,t)}$

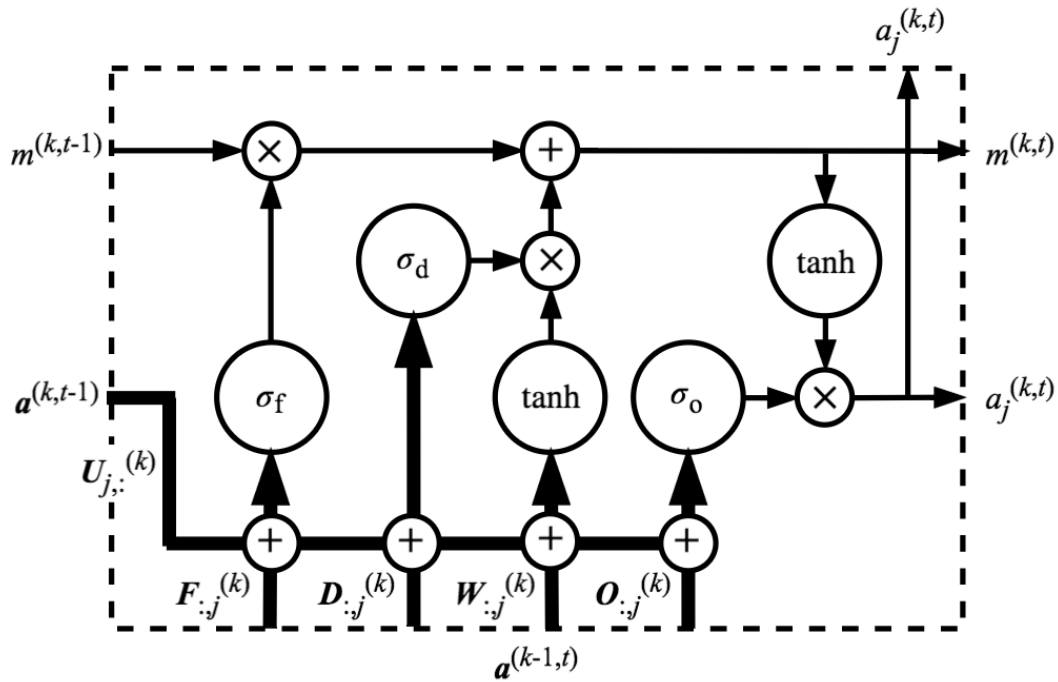




Long Short-Term Memory (LSTM)

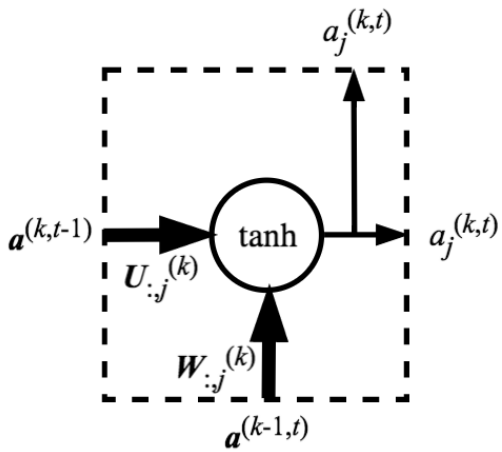
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:
  - First tanh activation to be added into the ***memory cell*** $m^{(k,t)}$
  - Second tanh activation as the final activation $a_j^{(k,t)}$
  - Forget gate σ_f: whether to forget $m^{(k,t-1)}$

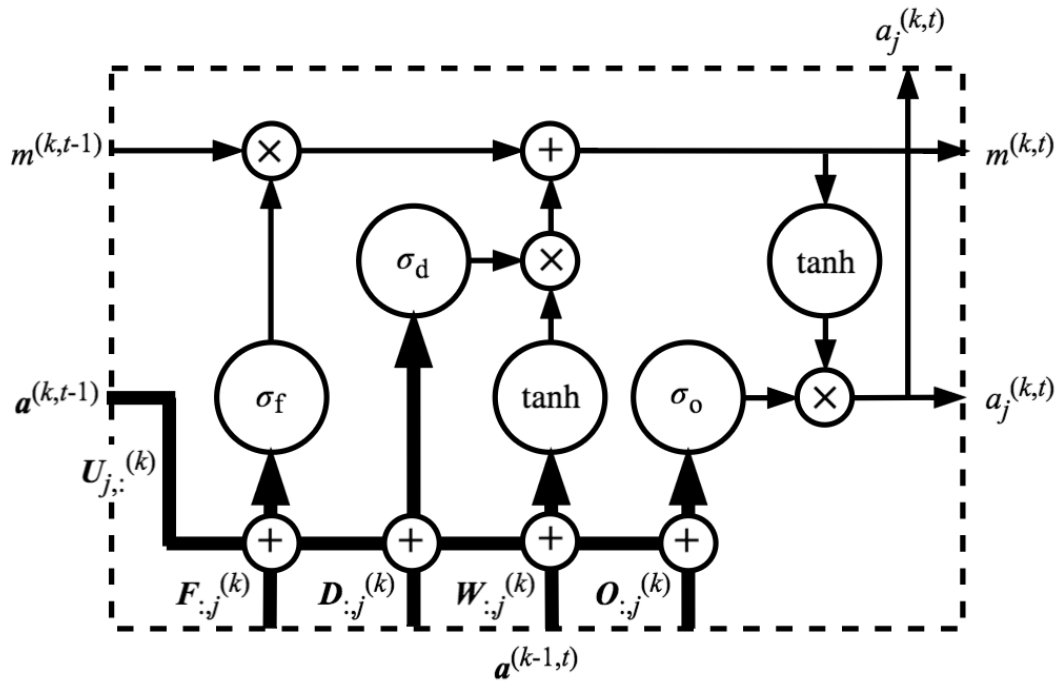




Long Short-Term Memory (LSTM)

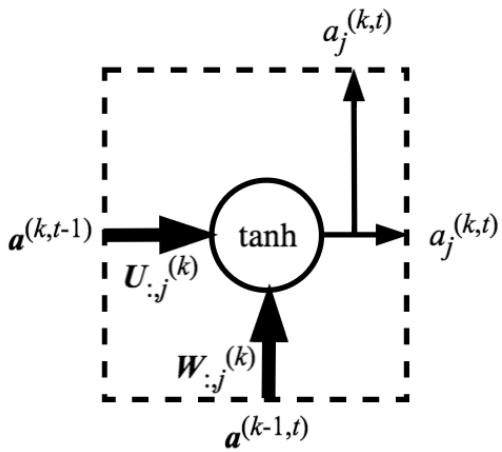
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:
  - First $\tanh$ activation to be added into the ***memory cell*** $m^{(k,t)}$
  - Second $\tanh$ activation as the final activation $a_j^{(k,t)}$
  - Forget gate σ_f: whether to forget $m^{(k,t-1)}$
  - Input gate σ_d: whether to store the first activation into $m^{(k,t)}$

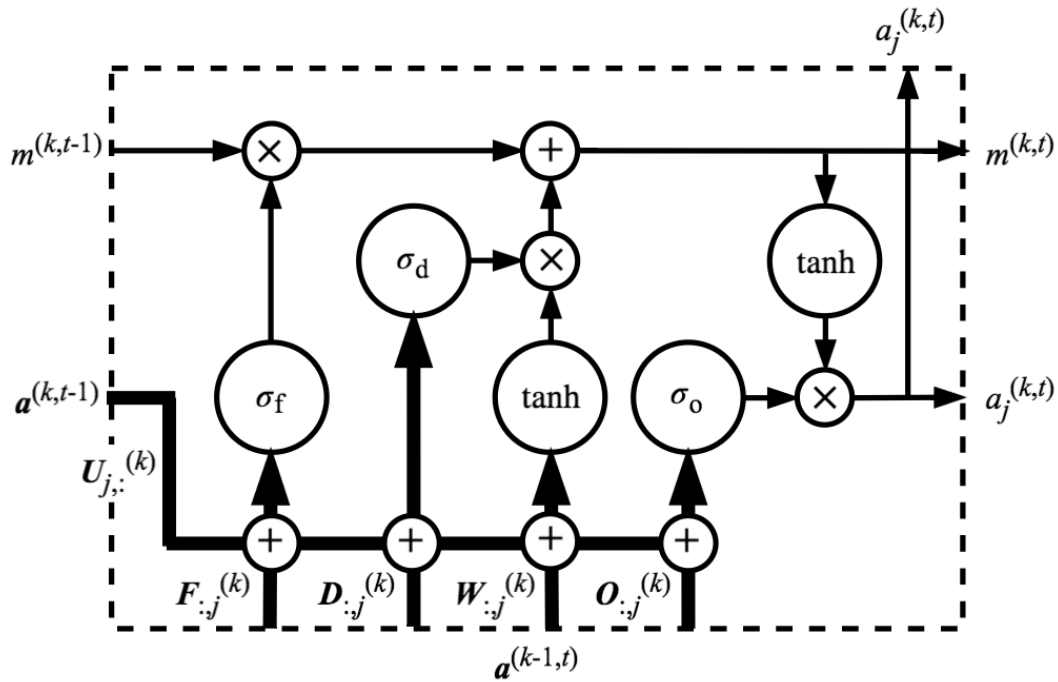




Long Short-Term Memory (LSTM)

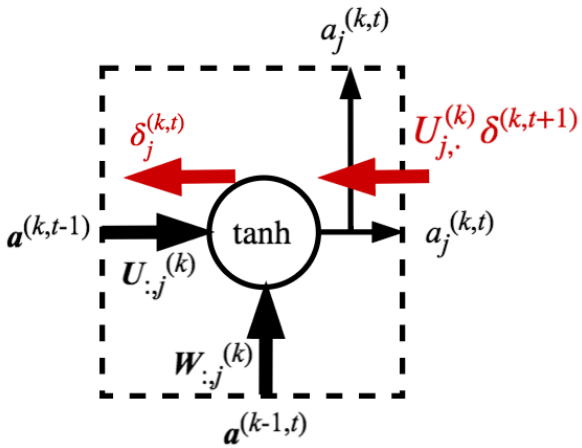
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:
  - First $\tanh$ activation to be added into the ***memory cell*** $m^{(k,t)}$
  - Second $\tanh$ activation as the final activation $a_j^{(k,t)}$
  - Forget gate σ_f: whether to forget $m^{(k,t-1)}$
  - Input gate σ_d: whether to store the first activation into $m^{(k,t)}$
  - Output gate σ_o: whether to output the second activation

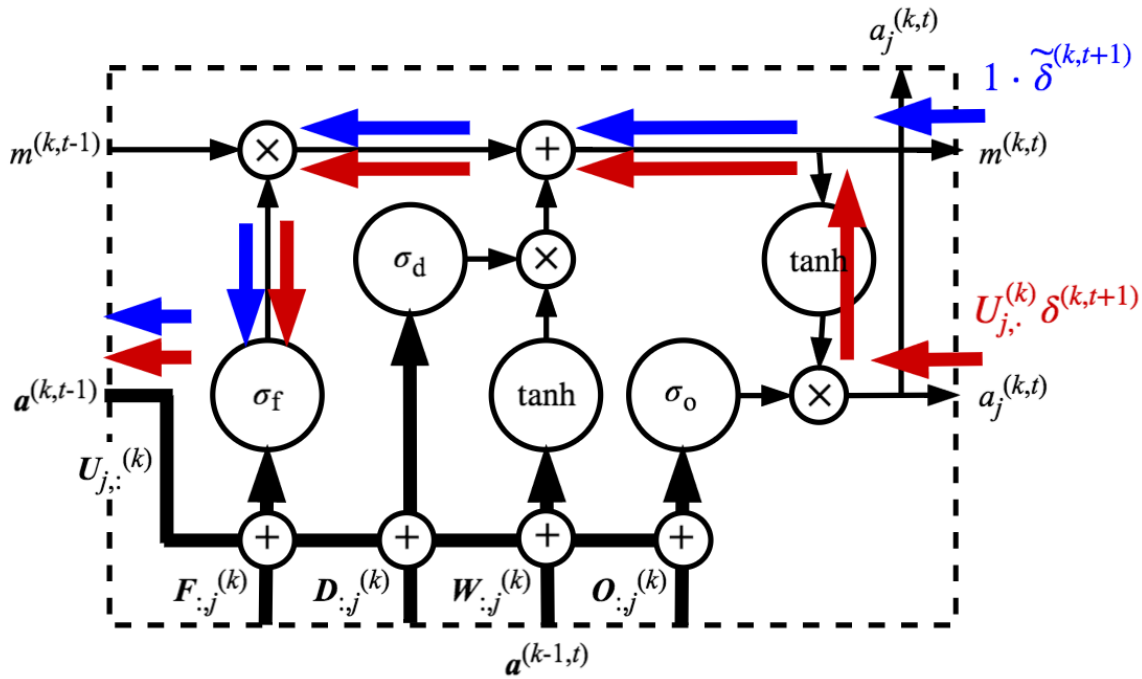




Error Signals

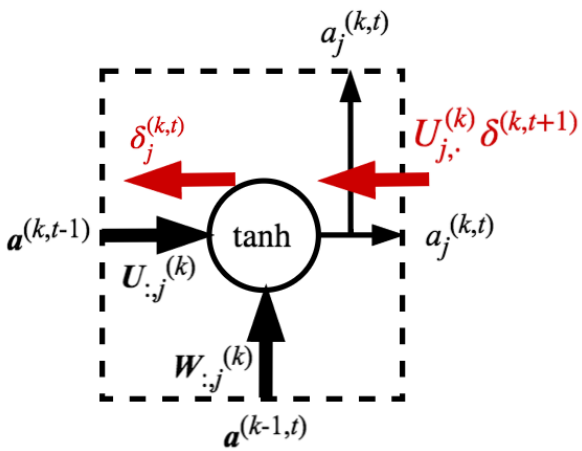
- Error signals now have a second path
  - If the forget gate is open, error signals won't decay (***blue arrows***)

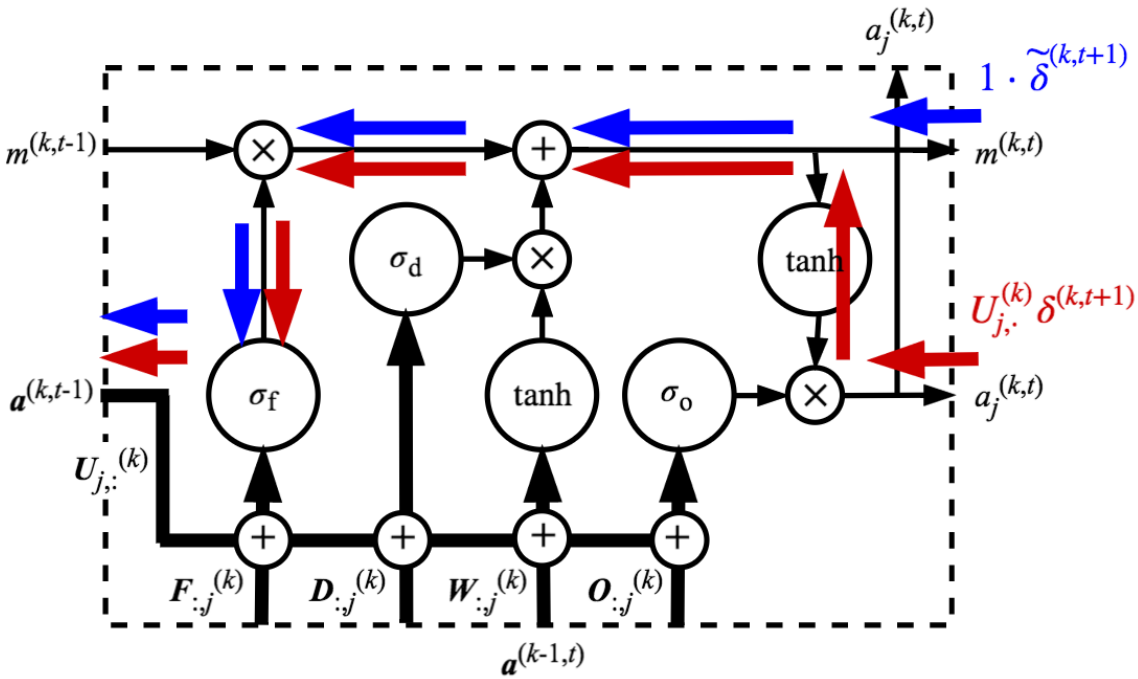




Error Signals

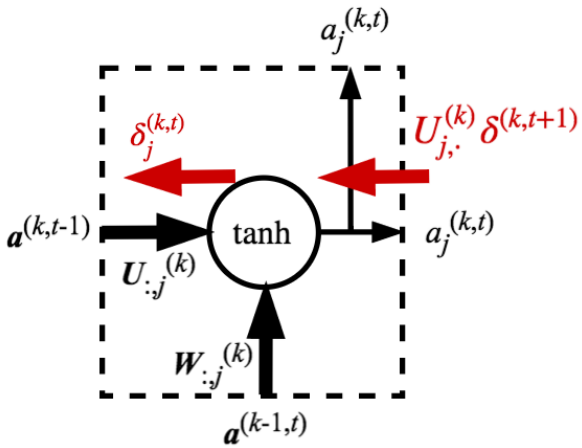
- Error signals now have a second path
  - If the forget gate is open, error signals won't decay (***blue arrows***)
  - Avoids the vanishing gradients (but not exploding ones)

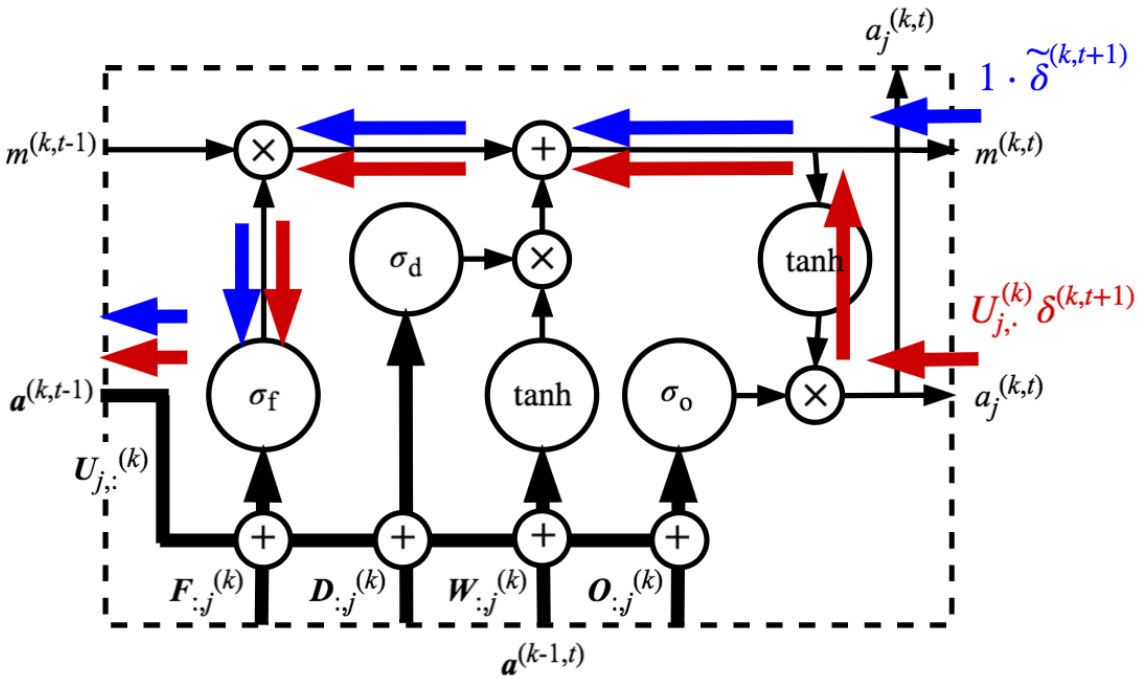




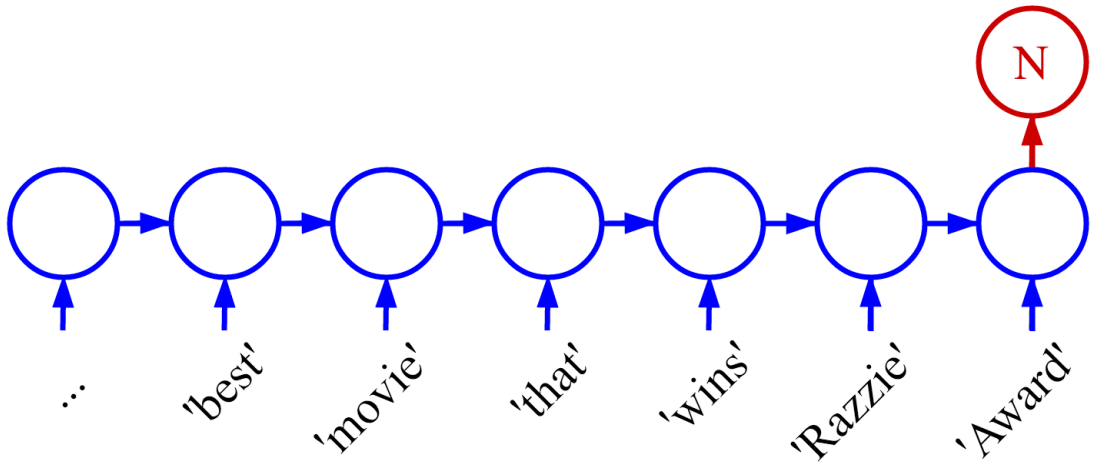
Error Signals

- Error signals now have a second path
  - If the forget gate is open, error signals won't decay (***blue arrows***)
  - Avoids the vanishing gradients (but not exploding ones)
- When NN decides to close the forget gate, the vanishing gradient problem is irrelevant
- In practice, LSTM + gradient clipping works well together



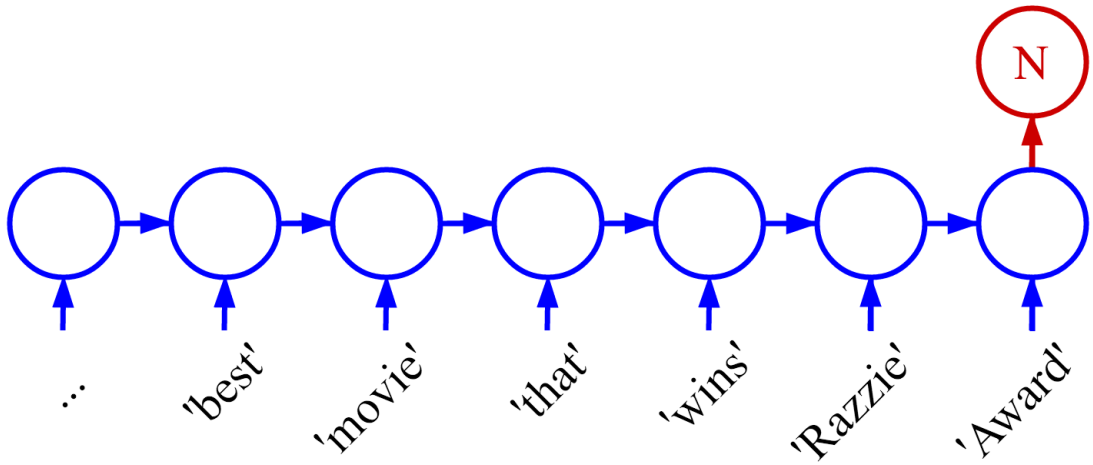


Dynamic Representations for Sentiment Analysis



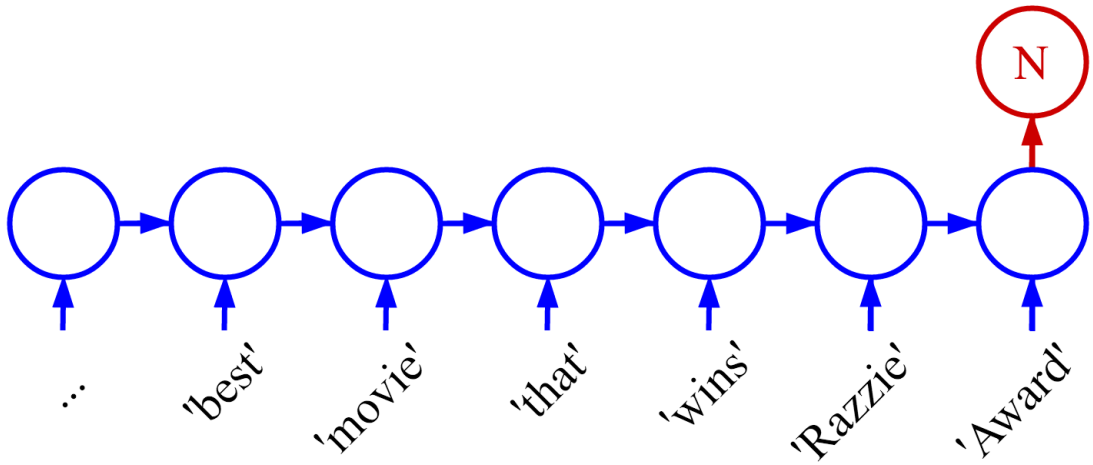
- LSTM units learn dynamic representations

Dynamic Representations for Sentiment Analysis



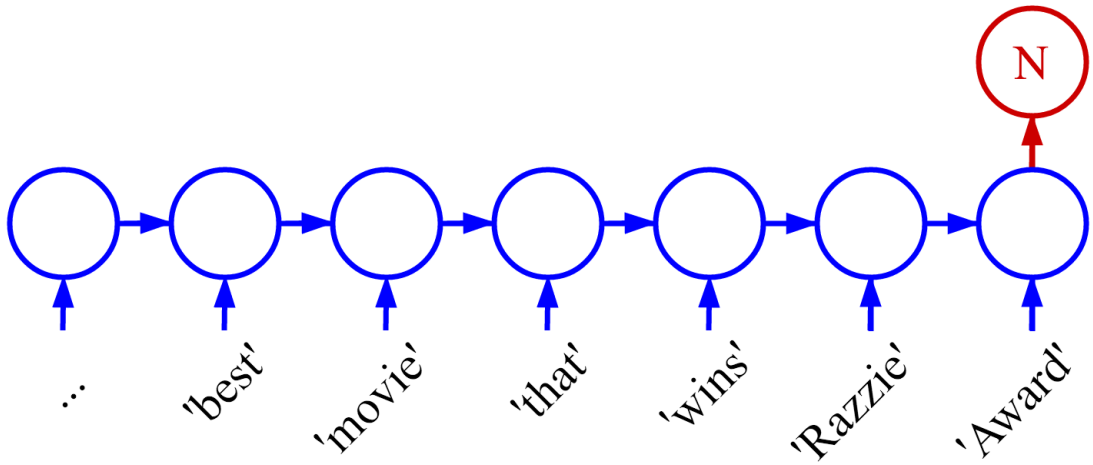
- LSTM units learn dynamic representations
- Closing forget gate for "Razzie"
  - To correct previous summarization and/or shift focus

Dynamic Representations for Sentiment Analysis



- LSTM units learn dynamic representations
- Closing forget gate for "Razzie"
  - To correct previous summarization and/or shift focus
- Closing input gate for "movie"
  - Not to learn, to keep the same summarization

Dynamic Representations for Sentiment Analysis

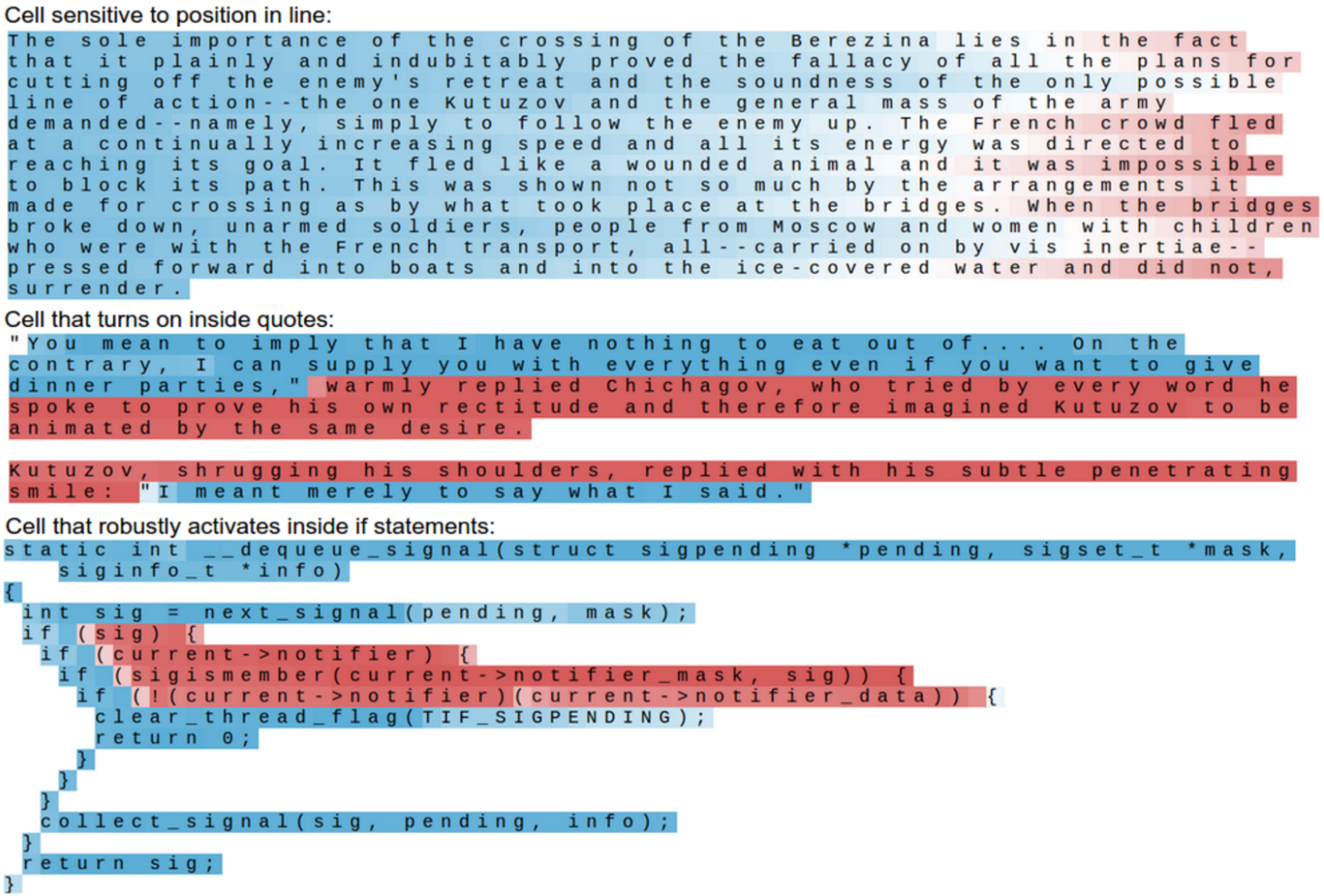


- LSTM units learn dynamic representations
- Closing forget gate for "Razzie"
  - To correct previous summarization and/or shift focus
- Closing input gate for "movie"
  - Not to learn, to keep the same summarization
- Closing output gate for "that"
  - To let the next neuron decide the activation/gate values by its own

Dynamic Representations for Language Models

Dynamic Representations for Language Models

- Neuron activations for language modeling [3] ▸ Interactive Tool



Learning Process of LSTMs

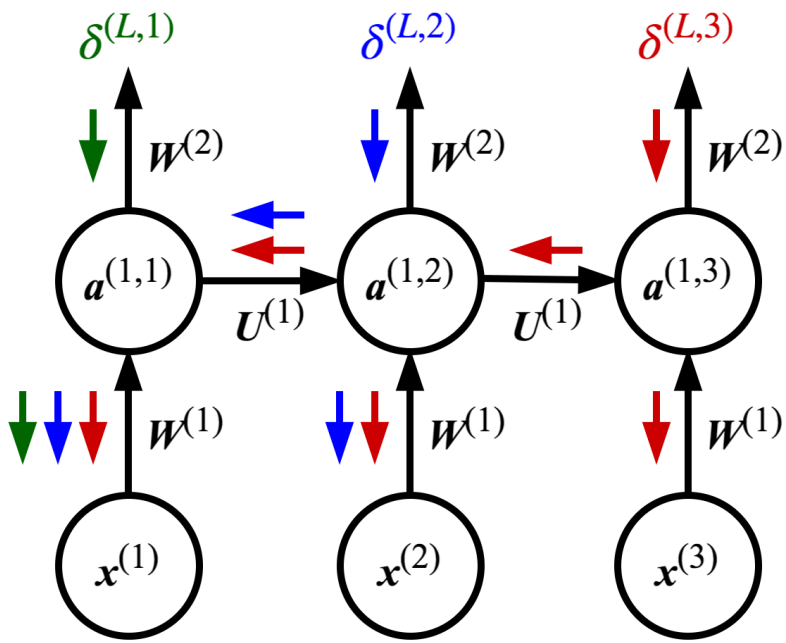
- Output at epoch 100:

  "*... tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh*..."

Outline

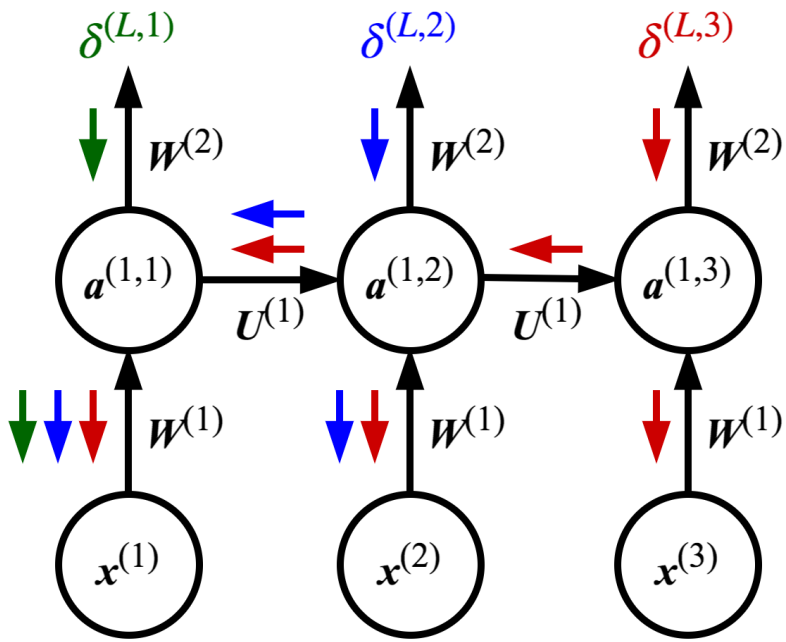
Parallelism

- A forward/backward pass through time in BPTT cannot be parallelized



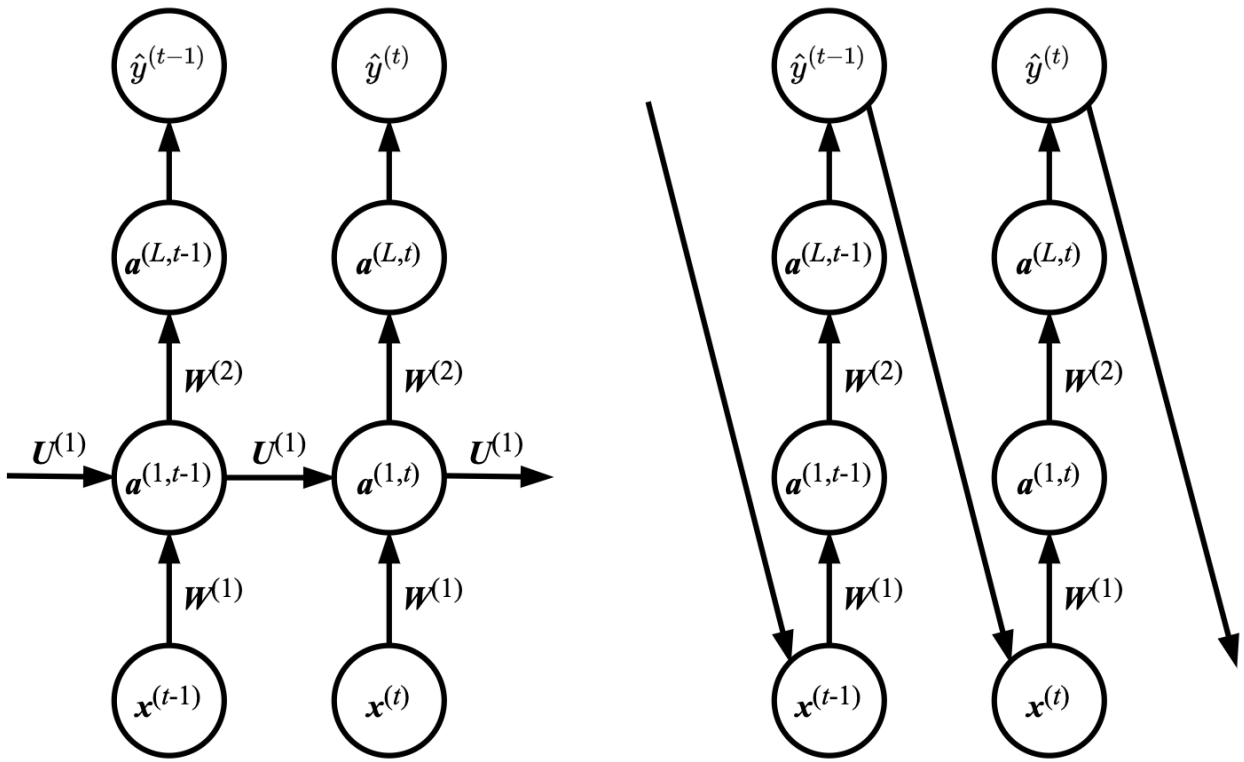
Parallelism

- A forward/backward pass through time in BPTT cannot be parallelized
- The ***hidden-to-hidden*** recurrent connections in a vanilla RNN create dependency between
  - $\boldsymbol{a}^{(k,t)}$'s in forward pass
  - $\delta^{(k,t)}$'s in backward pass



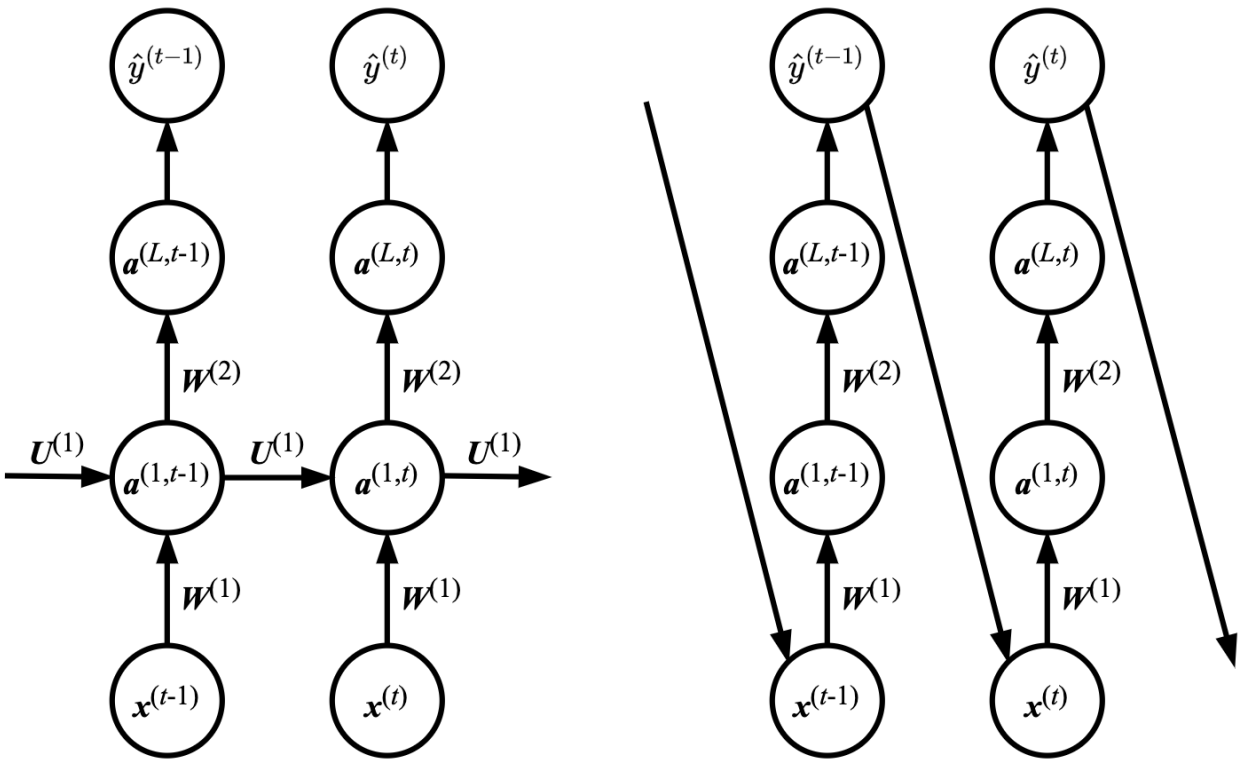
Output Recurrence and Teacher Forcing

- ***Teacher forcing***: replace hidden-to-hidden recurrence with output-to-hidden or output-to-input recurrence



Output Recurrence and Teacher Forcing

- ***Teacher forcing***: replace hidden-to-hidden recurrence with output-to-hidden or output-to-input recurrence
- At training time, use ***correct labels*** $\boldsymbol{y}^{(\cdot)}$'s to train the model
  - So, the forward/backward pass through time can be parallelized
- At test time, switch back to using model output $\hat{\boldsymbol{y}}^{(\cdot)}$'s



Exposure Bias

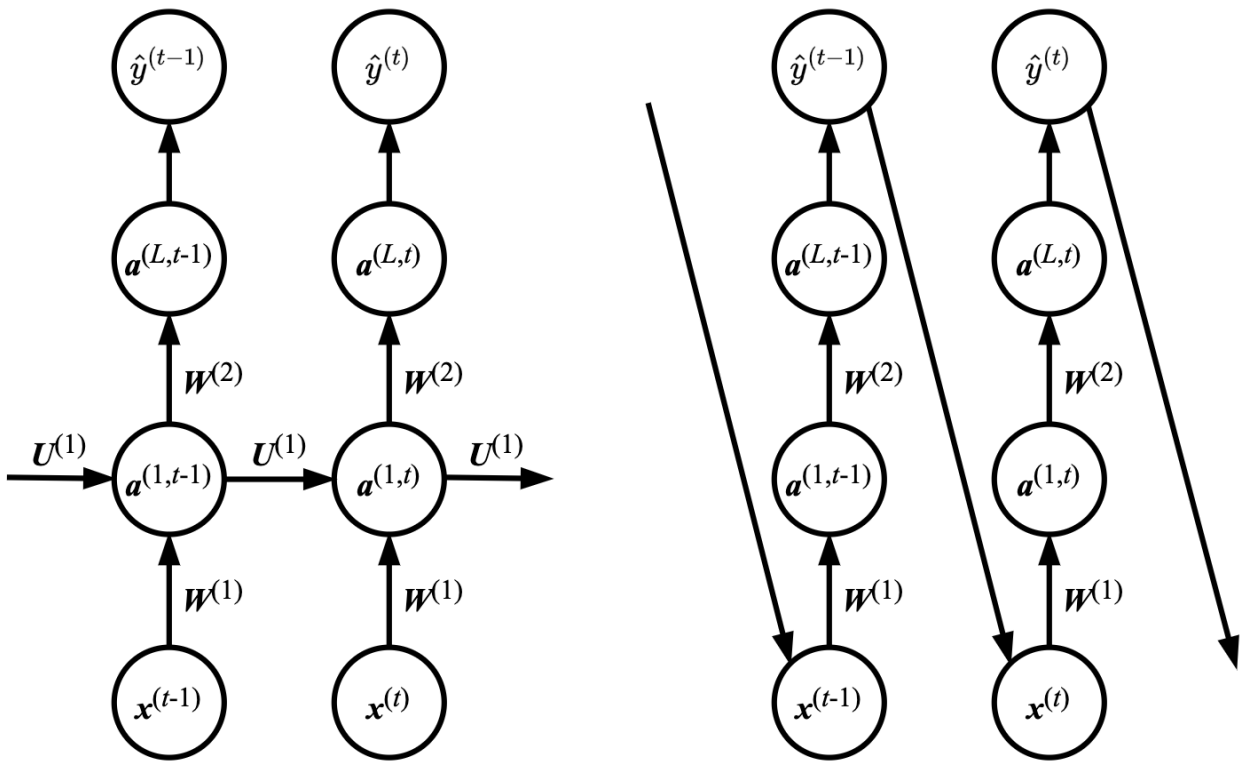
- Mismatch between $\boldsymbol{y}^{(\cdot)}$'s and $\hat{\boldsymbol{y}}^{(\cdot)}$'s hurts RNN performance
- Solution?

Exposure Bias

- Mismatch between $\boldsymbol{y}^{(\cdot)}$'s and $\hat{\boldsymbol{y}}^{(\cdot)}$'s hurts RNN performance
- Solution? Scheduled sampling
- At training time,
  1. Use $\boldsymbol{y}^{(\cdot)}$'s initially
  2. Gradually mix in $\hat{\boldsymbol{y}}^{(\cdot)}$'s later

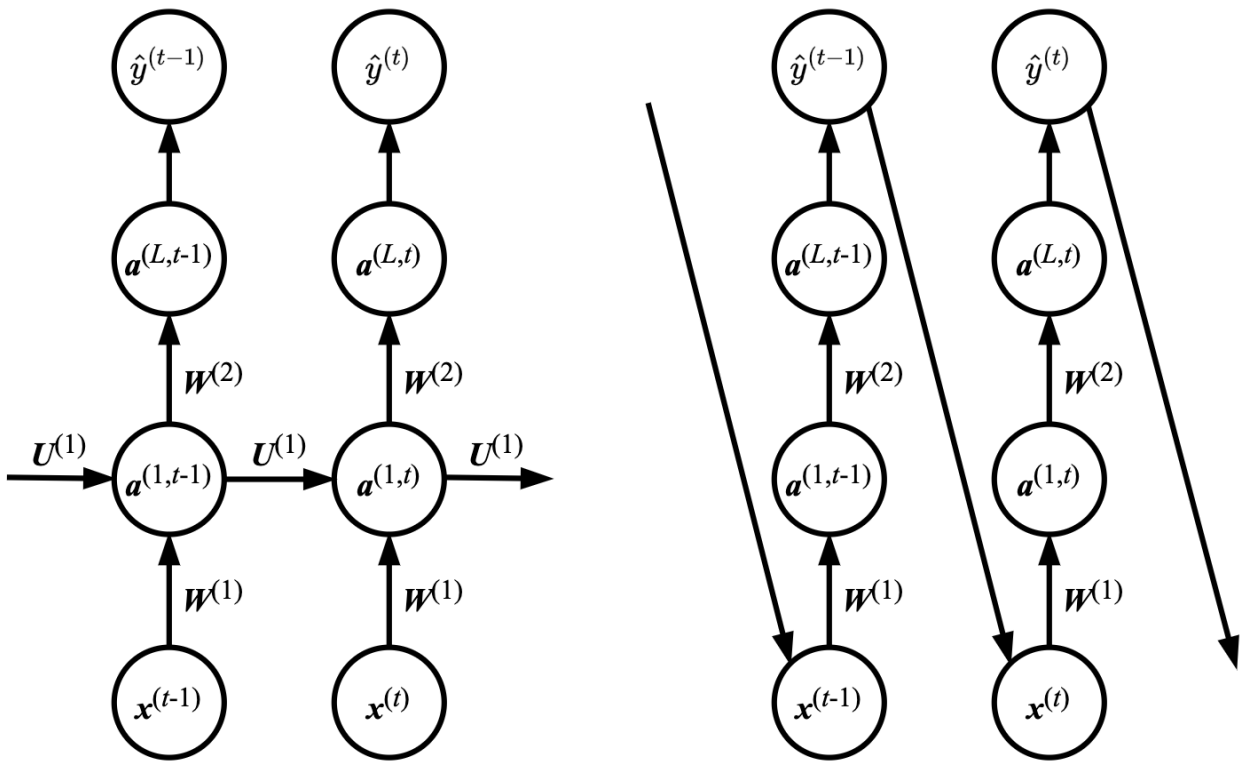
Reduced Expressiveness

- The vanilla RNNs are universal in the sense that they can simulate Turing machines [6]



Reduced Expressiveness

- The vanilla RNNs are universal in the sense that they can simulate Turing machines [6]
- Output-recurrent RNNs ***cannot*** simulate Turing machines and are strictly less powerful



Reduced Expressiveness

- The vanilla RNNs are universal in the sense that they can simulate Turing machines [6]
- Output-recurrent RNNs ***cannot*** simulate Turing machines and are strictly less powerful
- The output $\boldsymbol{a}^{(L,\cdot)}$'s are explicitly trained to match training targets
  - Cannot capture all required information in the past to predict the future