

Recurrent Neural Networks

Shan-Hung Wu

shwu@cs.nthu.edu.tw

Department of Computer Science,
National Tsing Hua University, Taiwan

Machine Learning

Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Sequential Data

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

Sequential Data

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

Sequential Data

- So far, we assume that data points (x,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 (x,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: $\mathbf{X} = \{X^{(n)}\}_n \in \mathbb{R}^{N \times (D,K) \times T}$

Sequential Data

- So far, we assume that data points (x,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: $\mathbf{X} = \{X^{(n)}\}_n \in \mathbb{R}^{N \times (D,K) \times T}$
 - $X^{(n)} = \{(x^{(n,t)}, y^{(n,t)})\}_t$ a **sequence**, where the superscript n can be omitted for simplicity

Sequential Data

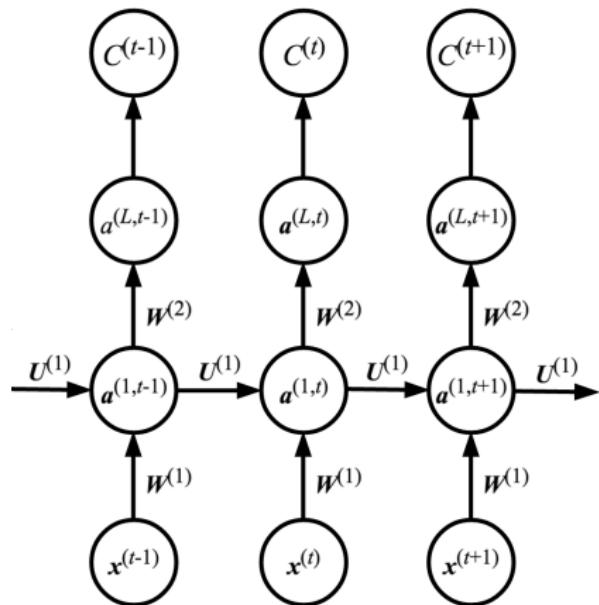
- So far, we assume that data points (x, y) 's in a dataset are i.i.d
- Does **not** hold in many applications
- **Sequential data:** data points come in order and successive points may be dependent, e.g.,
 - Letters in a word
 - Words in a sentence/document
 - Phonemes in a spoken word utterance
 -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

Sequence Modeling I

- How to model sequential data?

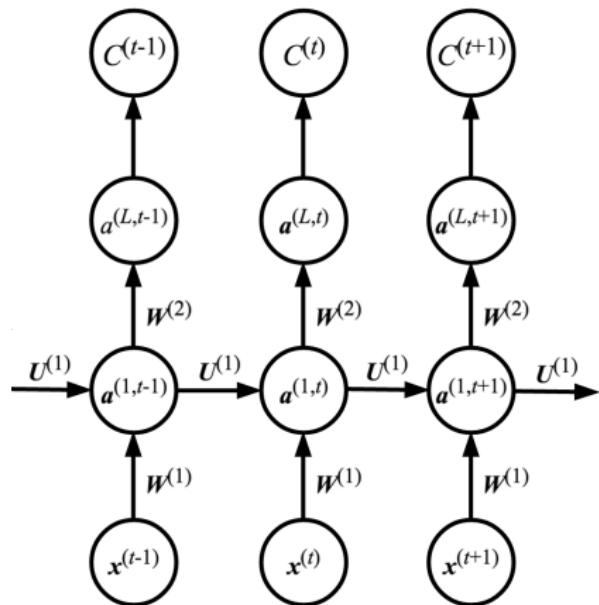
Sequence Modeling I

- 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 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}$

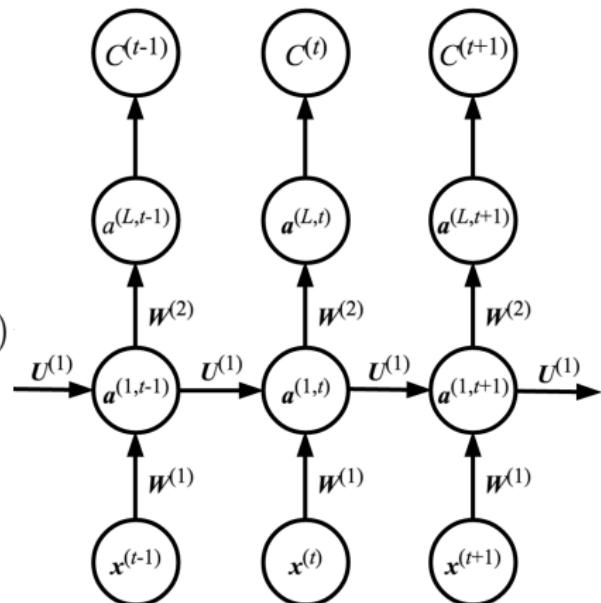


Sequence Modeling I

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

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

- Bias term omitted



Sequence Modeling I

- How to model sequential data?

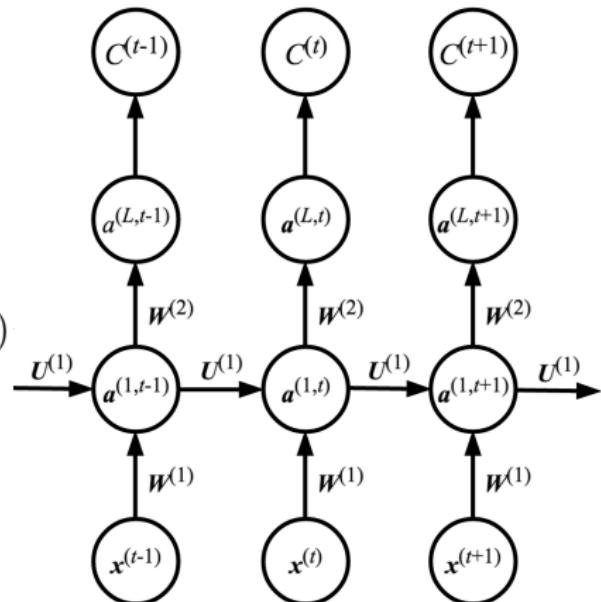
- **Recurrent neural networks**

(vanilla RNNs):

- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}$
- Output $\mathbf{a}^{(L,t)}$ depends on hidden activations:

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

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



Sequence Modeling I

- How to model sequential data?

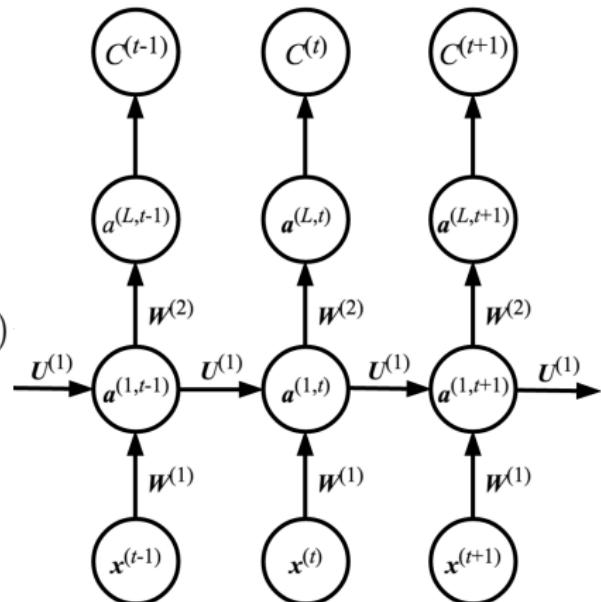
- **Recurrent neural networks**

(vanilla RNNs):

- $y^{(t)}$ depends on $x^{(1)}, \dots, x^{(t)}$
- Output $a^{(L,t)}$ depends on hidden activations:

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

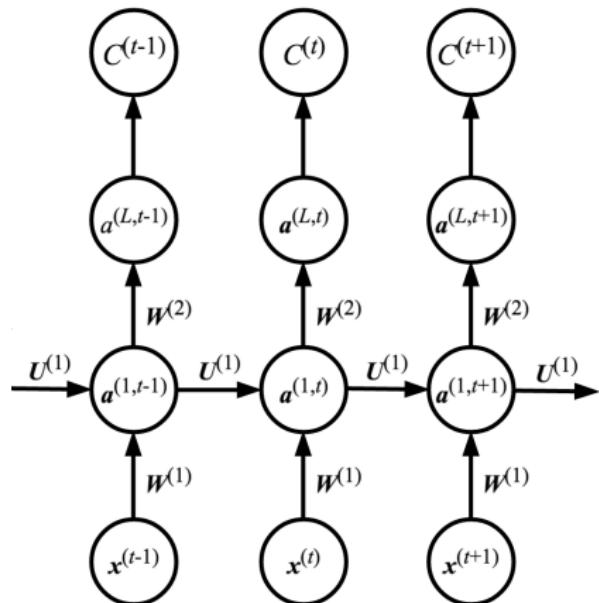
- Bias term omitted
- $\mathbf{a}^{(\cdot,t)}$ summarizes $x^{(t)}, \dots, x^{(1)}$
 - Earlier points are less important
- $\mathbf{a}^{(\cdot,t)}$'s at deeper layers give more abstract summarizations



Sequence Modeling II

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

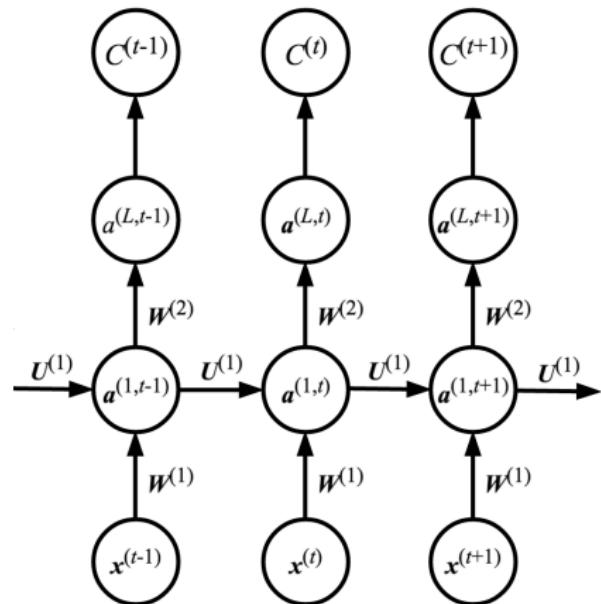
- Weights are *shared* across time instances



Sequence Modeling II

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

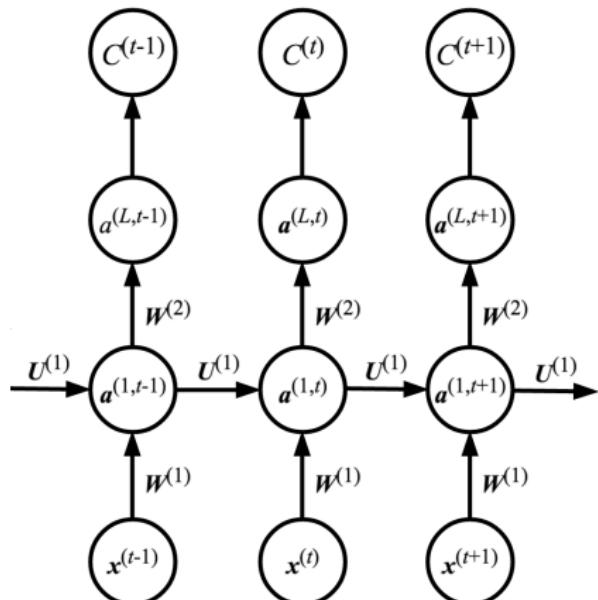
- Weights are *shared* across time instances
- Assumes that the “transition functions” are time invariant



Sequence Modeling II

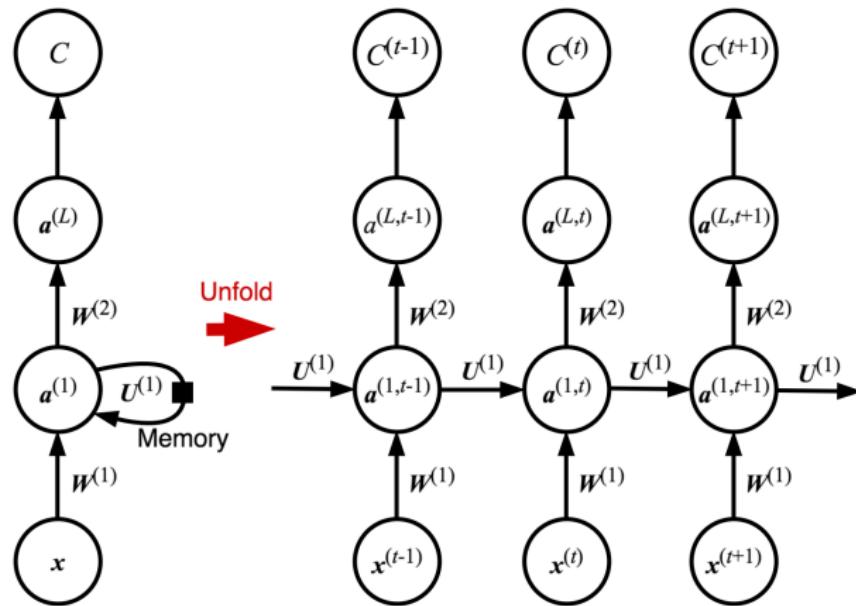
$$\begin{aligned}\mathbf{a}^{(k,t)} &= \text{act}(\mathbf{z}^{(k,t)}) \\ &= \text{act}(\mathbf{U}^{(k)} \mathbf{a}^{(k,t-1)} + \mathbf{W}^{(k)} \mathbf{a}^{(k-1,t)})\end{aligned}$$

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



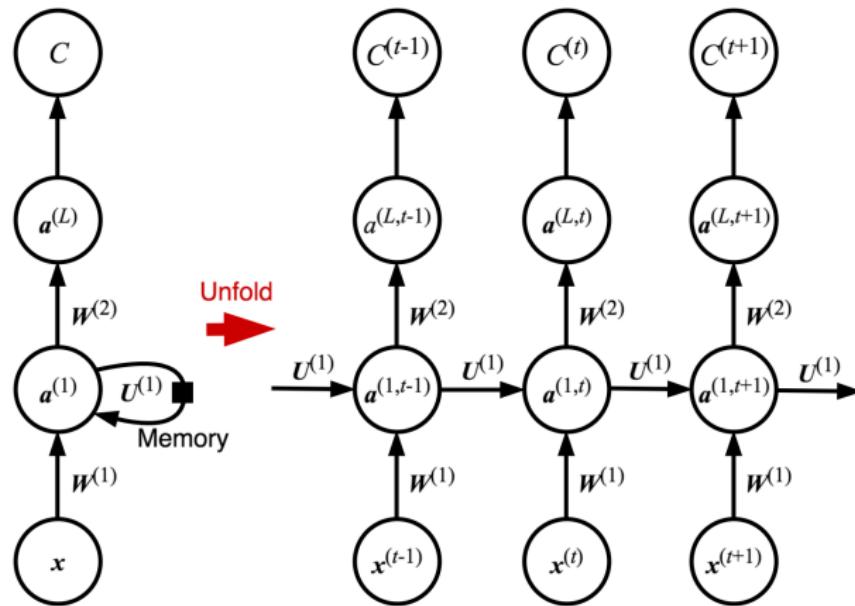
RNNs have Memory

- 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



Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

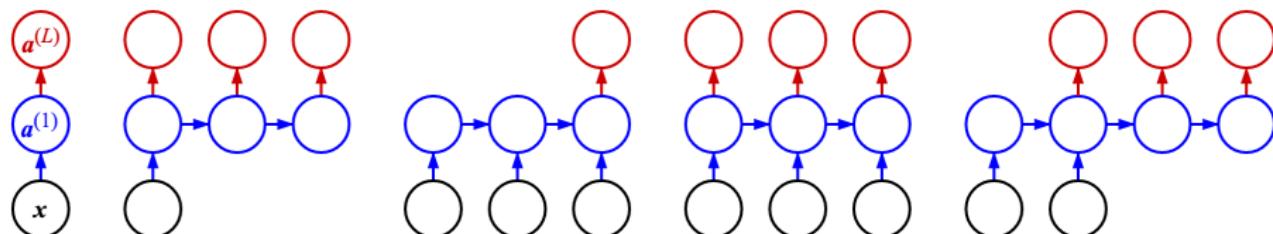
- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Input and Output

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



NN

One to Many

Many to One

Many to Many
(Synced)

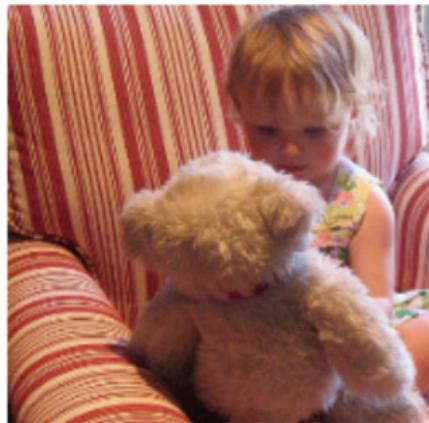
Many to Many
(Unsynced)

One2Many: Image Captioning

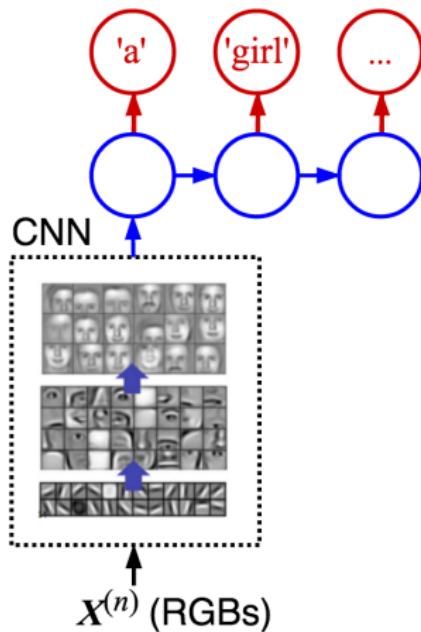


“A little girl sitting on a bed with a
teddy bear.”

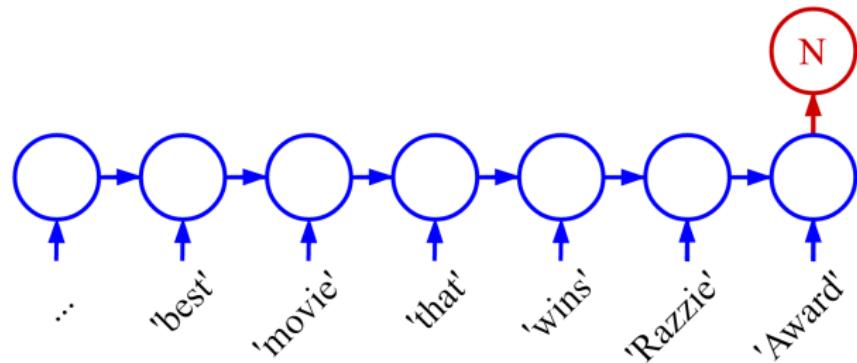
One2Many: Image Captioning



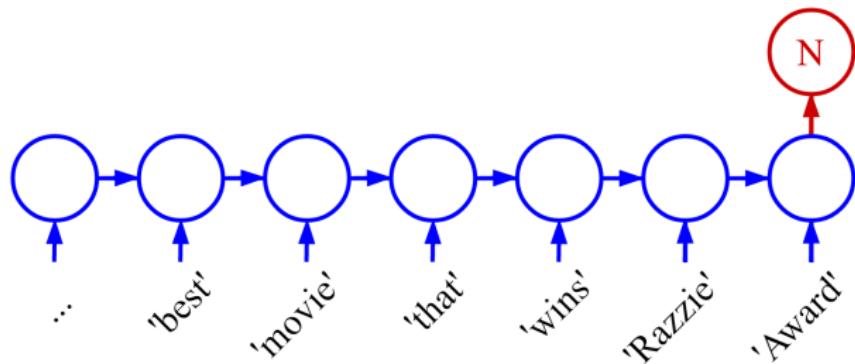
"A little girl sitting on a bed with a teddy bear."



Many2One: Sentiment Analysis



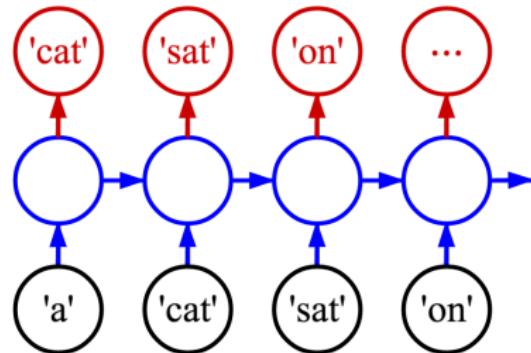
Many2One: Sentiment Analysis



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

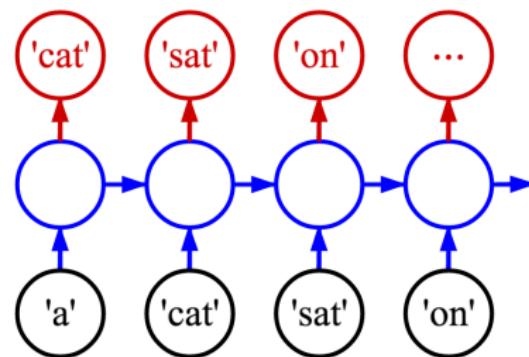
Many2Many (Synced): Language Modeling

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



Many2Many (Synced): Language Modeling

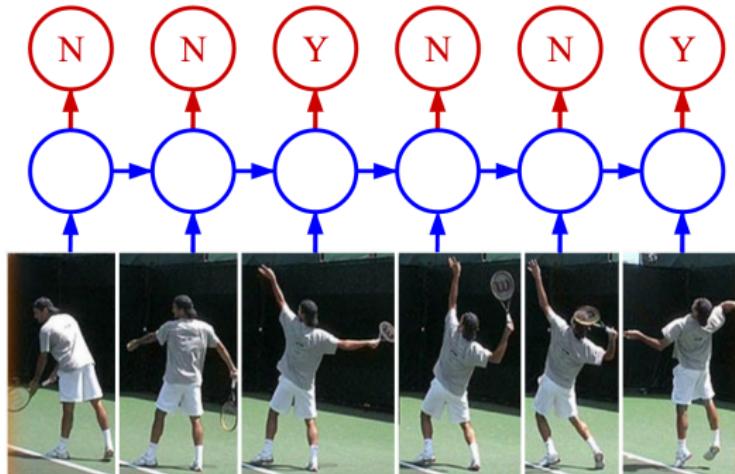
- **Language modeling**: predicting the next word based on the context



- Latent representation provides the context

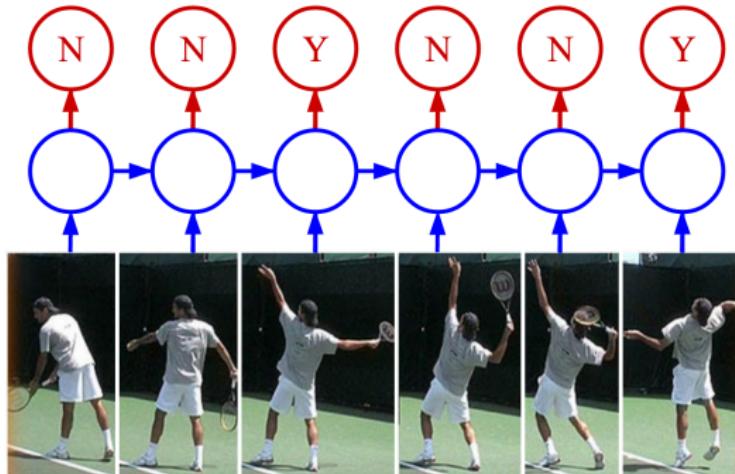
Many2Many (Synced): Video Keyframe Tagging

- Video frame annotation:



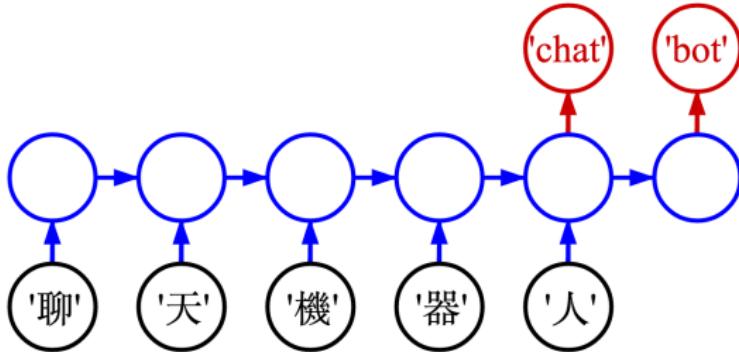
Many2Many (Synced): Video Keyframe Tagging

- Video frame annotation:



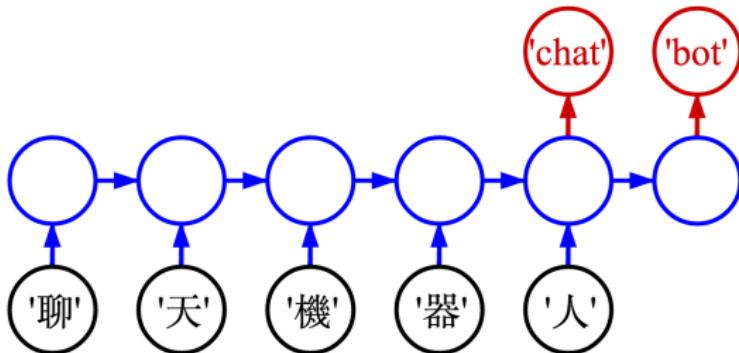
- Latent representation summarizes “what’s going on”

Many2Many (Unsynced): Machine Translation



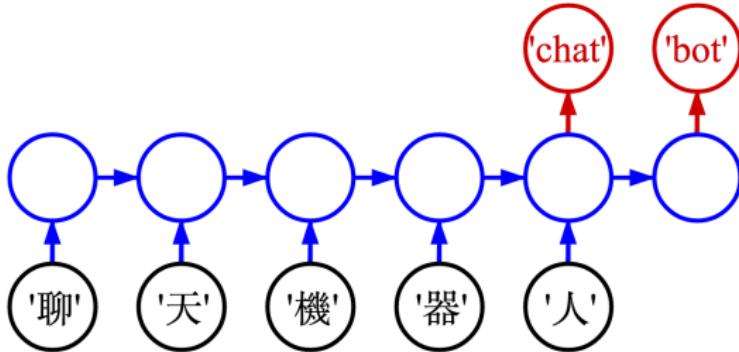
- Latent representation supports *encoding* first, and then *decoding*
 - Learns the structure difference
 - E.g., left-to-right vs. right-to-left languages

Many2Many (Unsynced): Machine Translation



- Latent representation supports *encoding* first, and then *decoding*
 - Learns the structure difference
 - E.g., left-to-right vs. right-to-left languages
- Also called *sequence to sequence* learning

Many2Many (Unsynced): Machine Translation



- Latent representation supports *encoding* first, and then *decoding*
 - Learns the structure difference
 - E.g., left-to-right vs. right-to-left languages
- Also called *sequence to sequence* learning
 - Has many applications, e.g., chat bots

Bidirectional RNNs

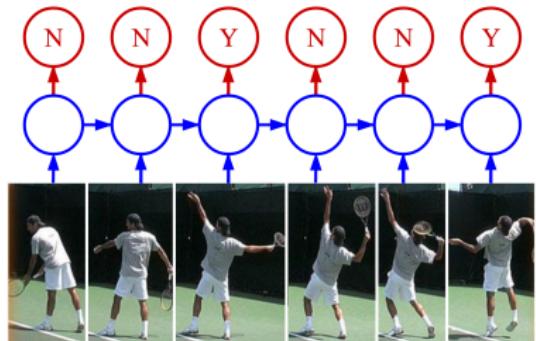
$$\mathbf{a}^{(k,t)} = \text{act}(\mathbf{U}^{(k)} \mathbf{a}^{(k,t-1)} + \mathbf{W}^{(k)} \mathbf{a}^{(k-1,t)})$$

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

Bidirectional RNNs

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

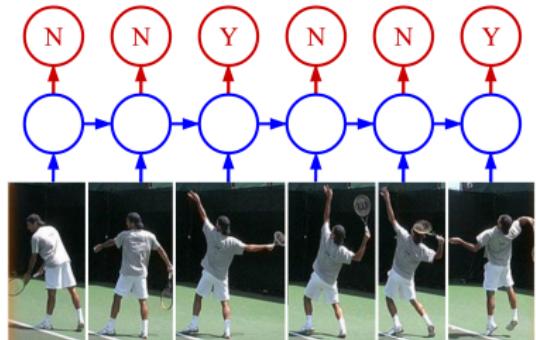
- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)}$
- A prediction is made by considering previous frames



Bidirectional RNNs

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

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

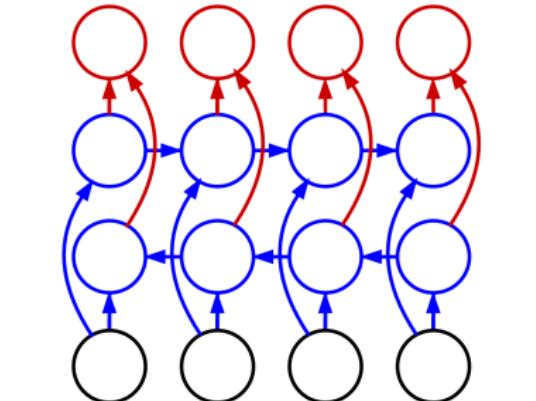
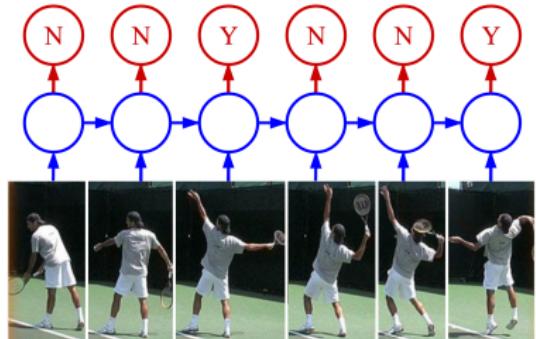


Bidirectional RNNs

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

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

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



Recursive RNNs I

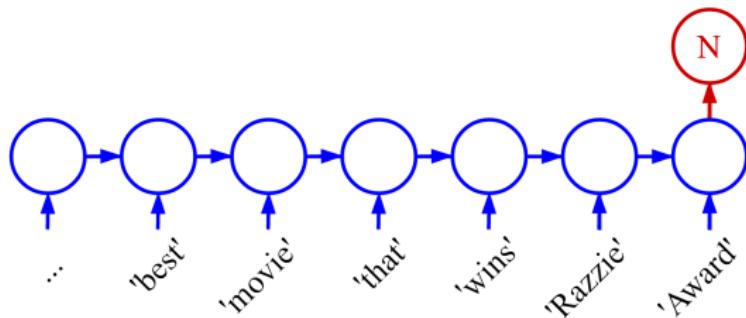
$$\mathbf{a}^{(k,t)} = \text{act}(\mathbf{U}^{(k)} \mathbf{a}^{(k,t-1)} + \mathbf{W}^{(k)} \mathbf{a}^{(k-1,t)})$$

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

Recursive RNNs I

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

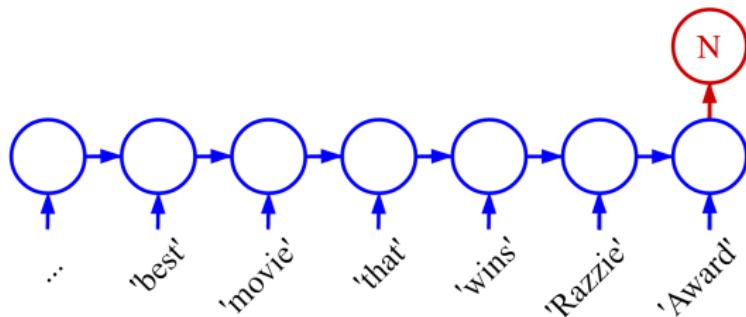
- The transition of hidden representation is invariant in time
 - Earlier input/representation is less important to current $\mathbf{a}^{(k,t)}$
- “Razzie” has less effect if it is far away from the prediction



Recursive RNNs I

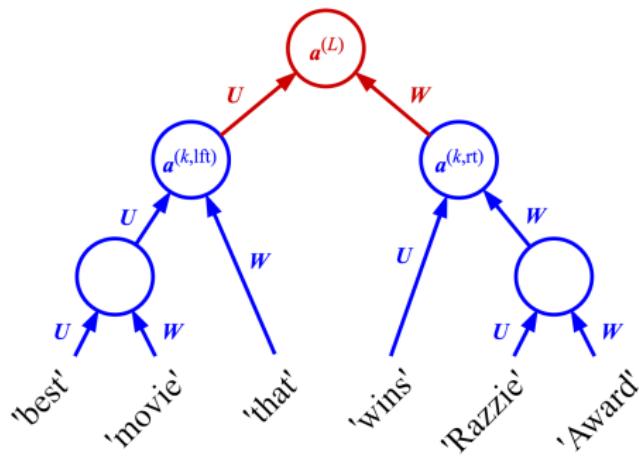
$$\mathbf{a}^{(k,t)} = \text{act}(\mathbf{U}^{(k)} \mathbf{a}^{(k,t-1)} + \mathbf{W}^{(k)} \mathbf{a}^{(k-1,t)})$$

- The transition of hidden representation is invariant in time
 - Earlier input/representation is less important to current $\mathbf{a}^{(k,t)}$
- “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 $X^{(n)}$ into a tree
 - Following grammar rules

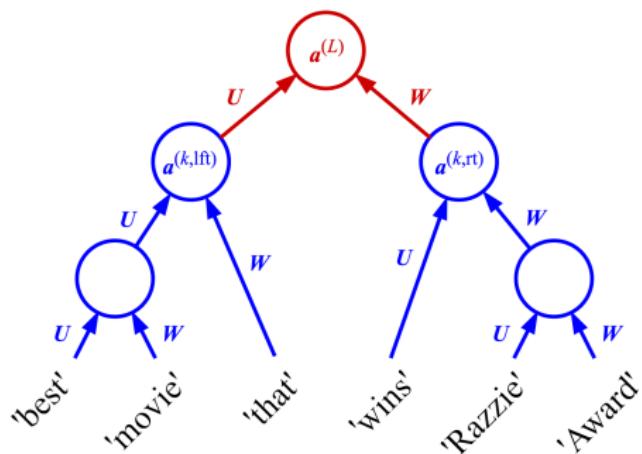


Recursive RNNs II

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

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

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

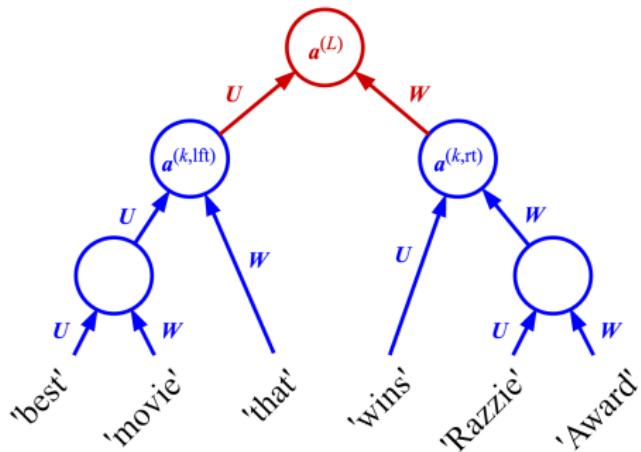


Recursive RNNs II

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

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

- \mathbf{U} and \mathbf{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

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Cost Function of Vanilla RNNs

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

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

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

Cost Function of Vanilla RNNs

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

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

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

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

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

Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

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

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

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

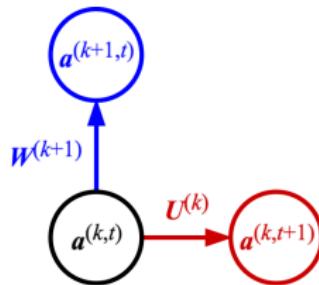
$$\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \text{act}'(z_j^{(k,t)})$$

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

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

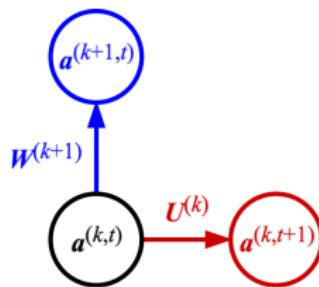


Backward Pass through Time I

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

- We have

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



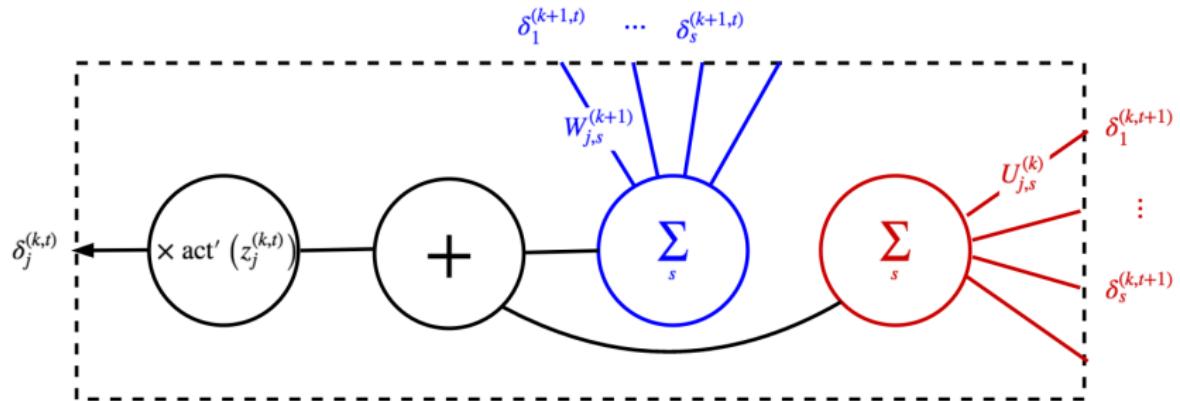
Backward Pass through Time II

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

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)

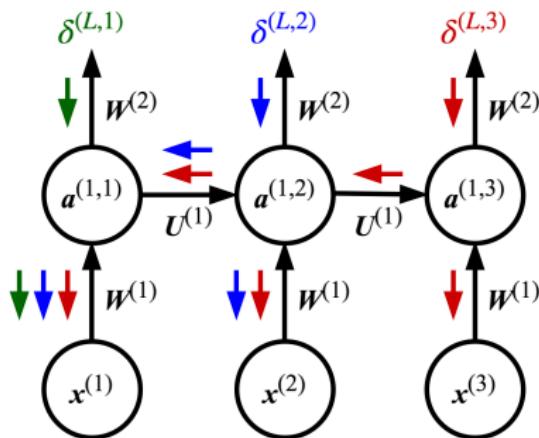
- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time
- However, for each sequence n , we have multiple $c^{(n,t)}$'s at different t 's
 - Their gradient need to be summed

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 t
- 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

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- **Optimization Techniques**
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Long-Term Dependencies

- Forward pass:

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

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

- Backward pass:

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

Long-Term Dependencies

- Forward pass:

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

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

- Backward pass:

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

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

Long-Term Dependencies

- Forward pass:

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

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

- Backward pass:

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

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

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

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

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

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

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

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

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

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

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

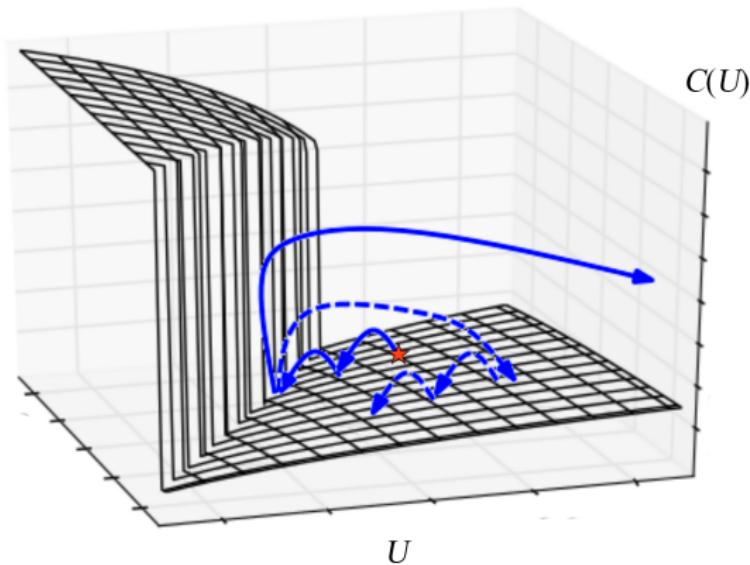
- Ignoring activation function and depth:

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

- Given eigendecomposition: $\mathbf{U}^{(k)} = \mathbf{Q} \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{D^{(k)}} \end{bmatrix} \mathbf{Q}^\top$
- We have: $(\mathbf{U}^{(k)})^{j-i} = \mathbf{Q} \begin{bmatrix} \lambda_1^{j-i} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \mathbf{Q}^\top$
- 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!

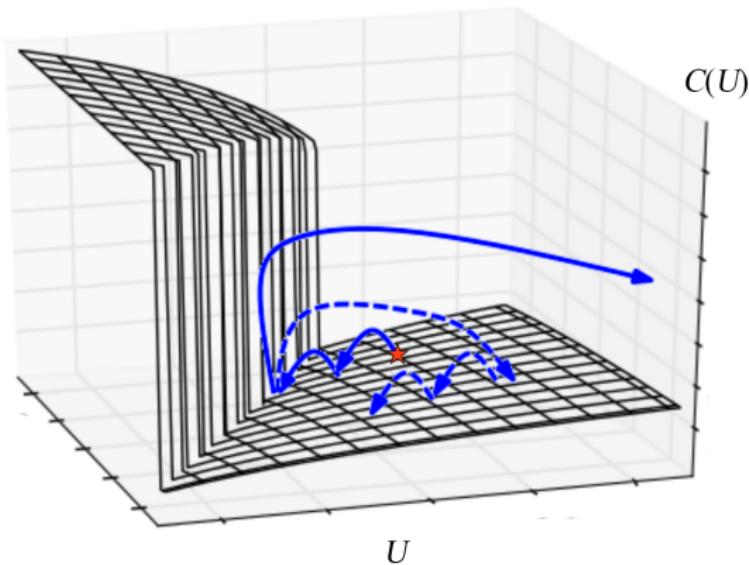
Cost Surface

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



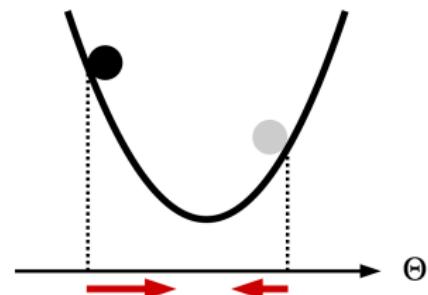
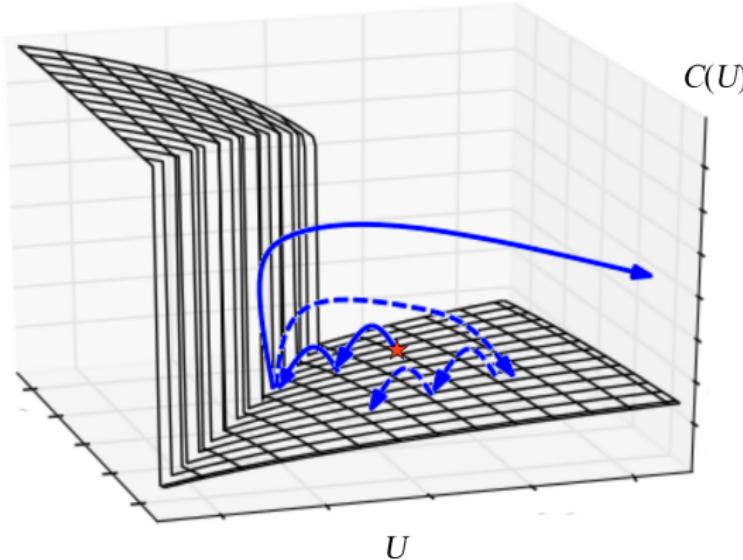
Cost Surface

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



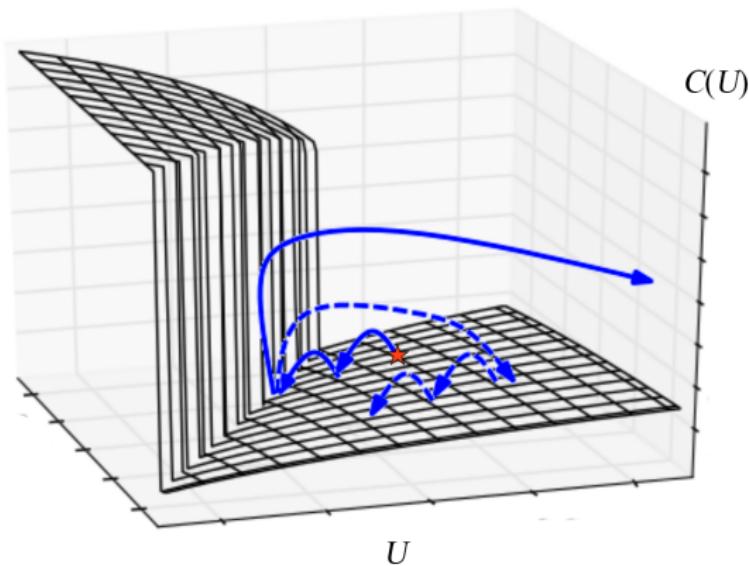
Nesterov Momentum

- 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

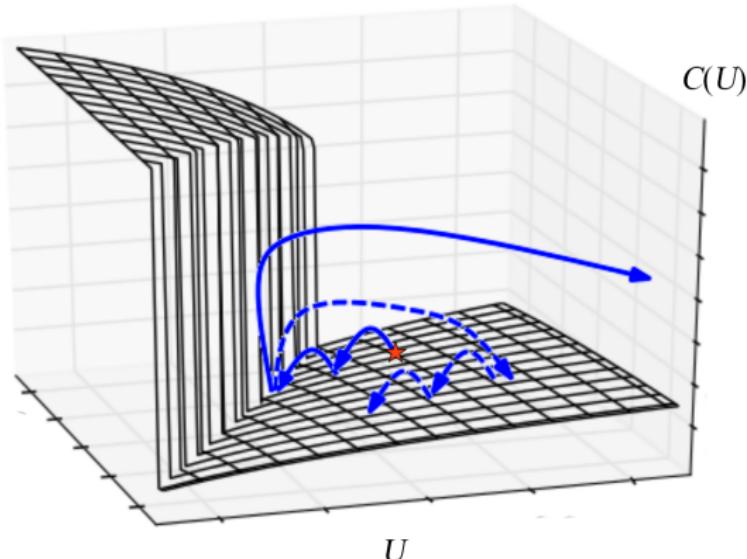


RMS Prop

- Adaptive learning rate based on statistics of recent gradients:

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

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



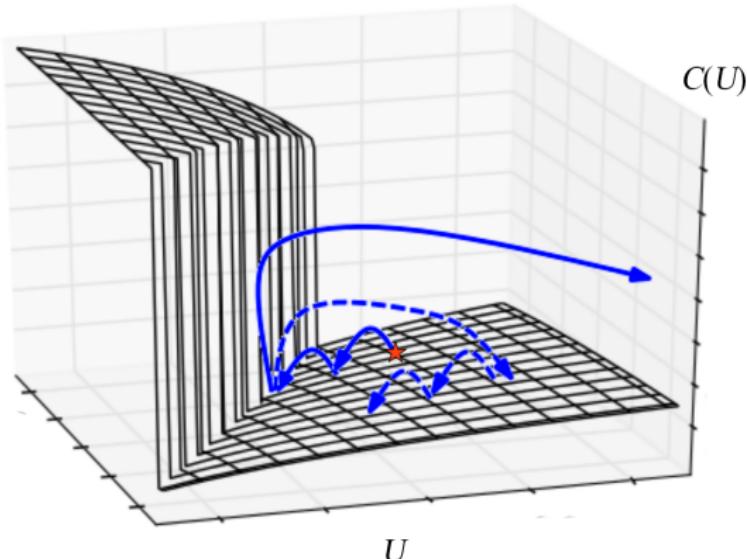
RMS Prop

- Adaptive learning rate based on statistics of recent gradients:

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

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

- Reduce λ



Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

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:

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

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

Backward pass:

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

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:

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

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

Backward pass:

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

- 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:

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

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

Backward pass:

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

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

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

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

Backward pass:

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

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

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

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

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

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

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

$$\begin{bmatrix} \lambda_1^{j-i} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \mathbf{Q}^\top$$

- Long-term dependency: $(\mathbf{U}^{(k)})^{j-i} = \mathbf{Q} \begin{bmatrix} \lambda_1^{j-i} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \mathbf{Q}^\top$
- Why not make $\mathbf{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [10] propose IRNN:
 - **Initializes** $\mathbf{U}^{(k)} = \mathbf{I}$ in SGD
 - Uses **ReLU** hidden units

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

- Long-term dependency: $(U^{(k)})^{j-i} = Q \begin{bmatrix} \lambda_1^{j-i} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} Q^\top$
- Why not make $U^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [10] propose IRNN:
 - **Initializes** $U^{(k)} = I$ 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 $U^{(k)}$'s

$$(\mathbf{U}^{(k)})^{j-i} = \mathbf{Q} \begin{bmatrix} \lambda_1^{j-i} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \mathbf{Q}^\top$$

- Long-term dependency: Why not make $\mathbf{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [10] propose IRNN:
 - **Initializes** $\mathbf{U}^{(k)} = \mathbf{I}$ 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. [8] add a term $\sum_{k,t} (\|\mathbf{a}^{(k,t)}\| - \|\mathbf{a}^{(k,t-1)}\|)^2$ to IRNN cost to stabilize the norms of $\mathbf{a}^{(k,t)}$'s in time

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

$$(\mathbf{U}^{(k)})^{j-i} = \mathbf{Q} \begin{bmatrix} \lambda_1^{j-i} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \mathbf{Q}^\top$$

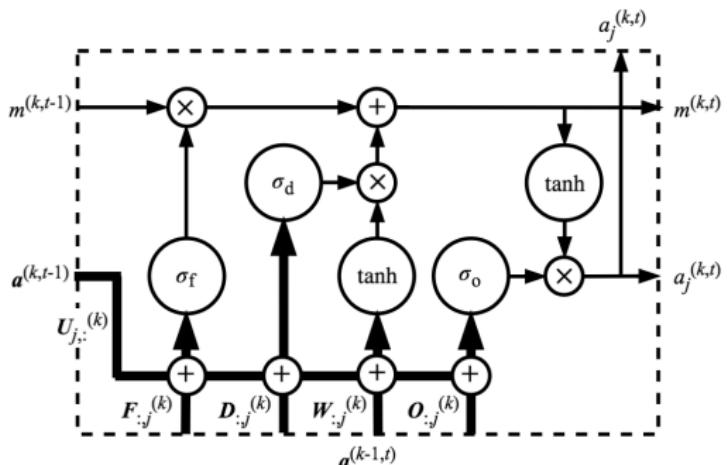
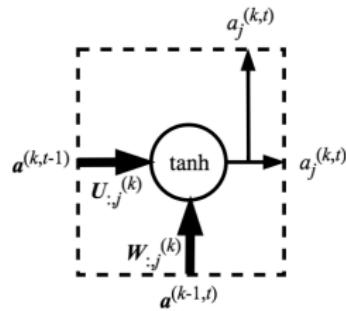
- Long-term dependency: Why not make $\mathbf{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [10] propose IRNN:
 - **Initializes** $\mathbf{U}^{(k)} = \mathbf{I}$ in SGD
 - Uses **ReLU** hidden units
 - Empirically, requires a very small learning rate (e.g., 10^{-8}) to work well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

Long Short-Term Memory (LSTM)

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

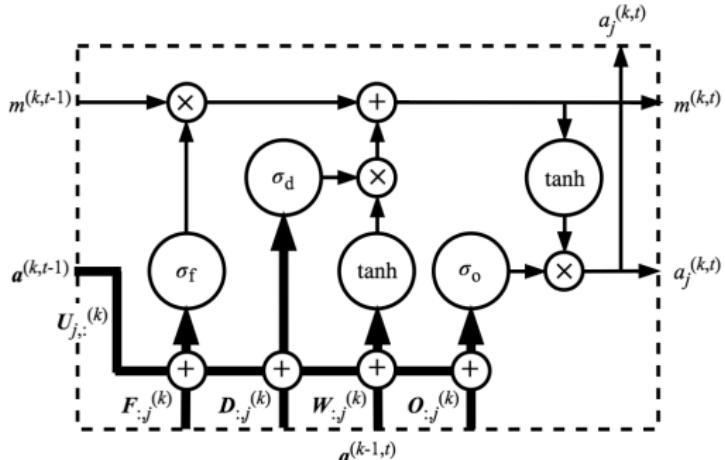
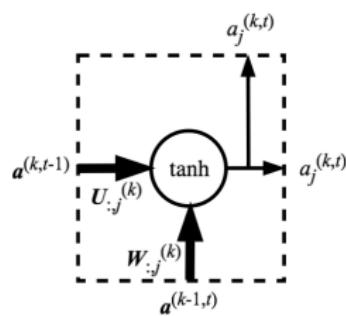
Long Short-Term Memory (LSTM)

- 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 :



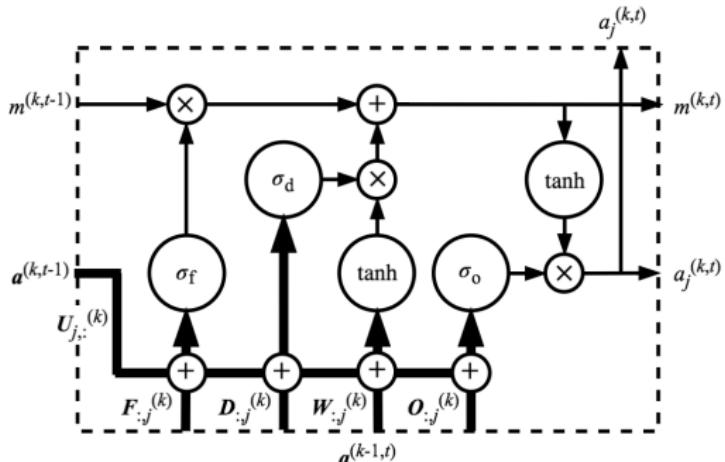
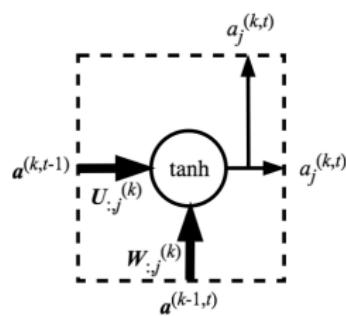
Long Short-Term Memory (LSTM)

- 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 $a_j^{(k,t)}$



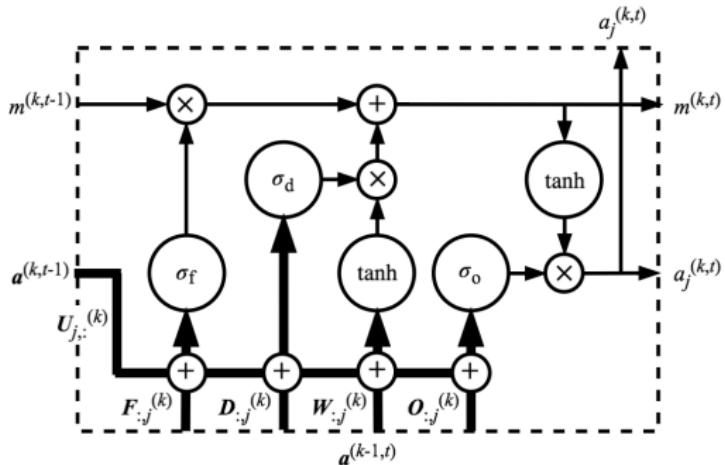
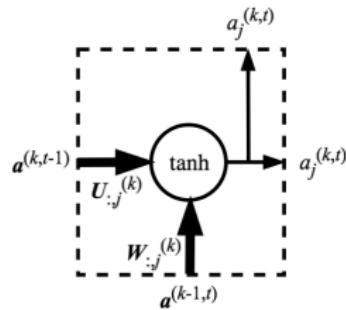
Long Short-Term Memory (LSTM)

- 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 $m^{(k,t-1)}$



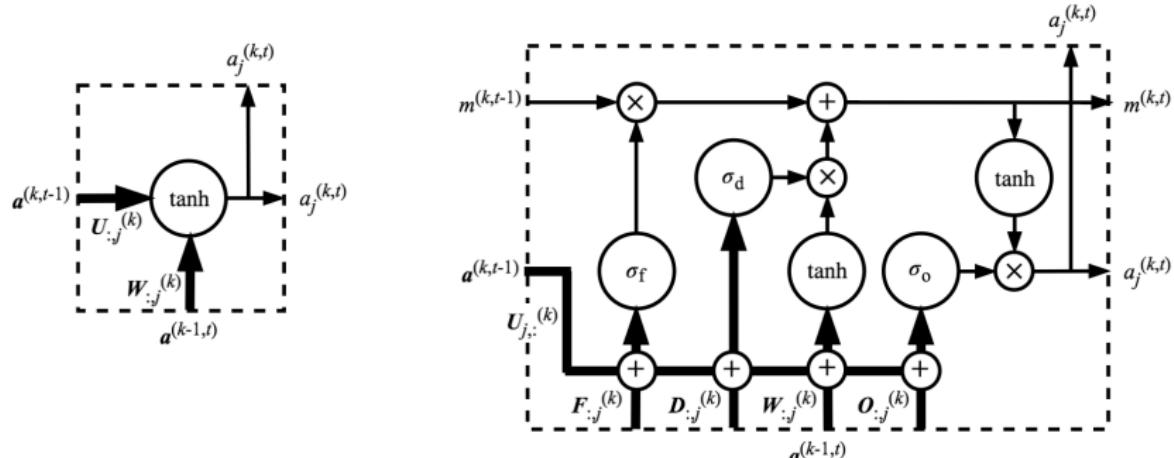
Long Short-Term Memory (LSTM)

- 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 $m^{(k,t)}$



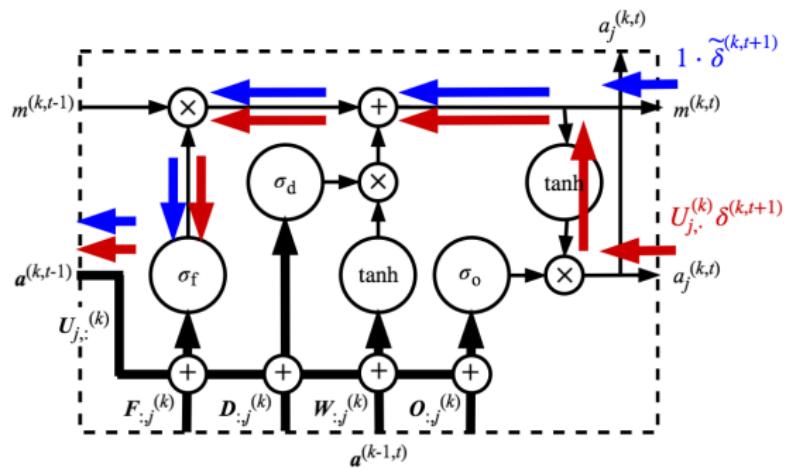
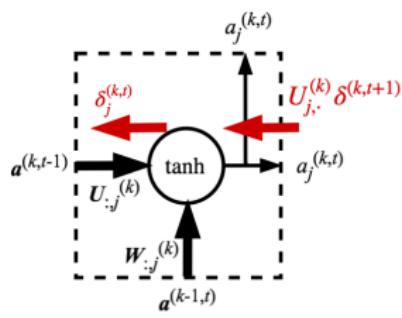
Long Short-Term Memory (LSTM)

- 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



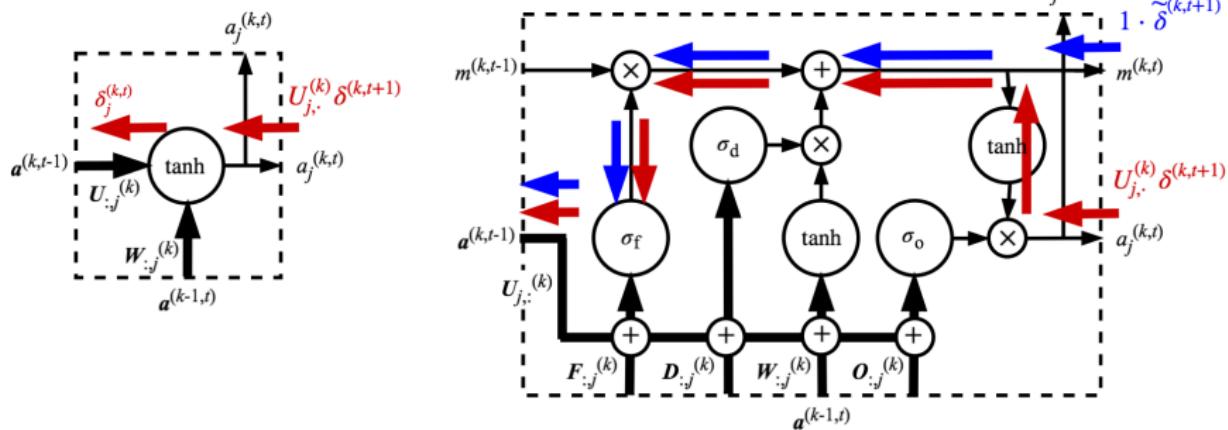
Error Signals

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



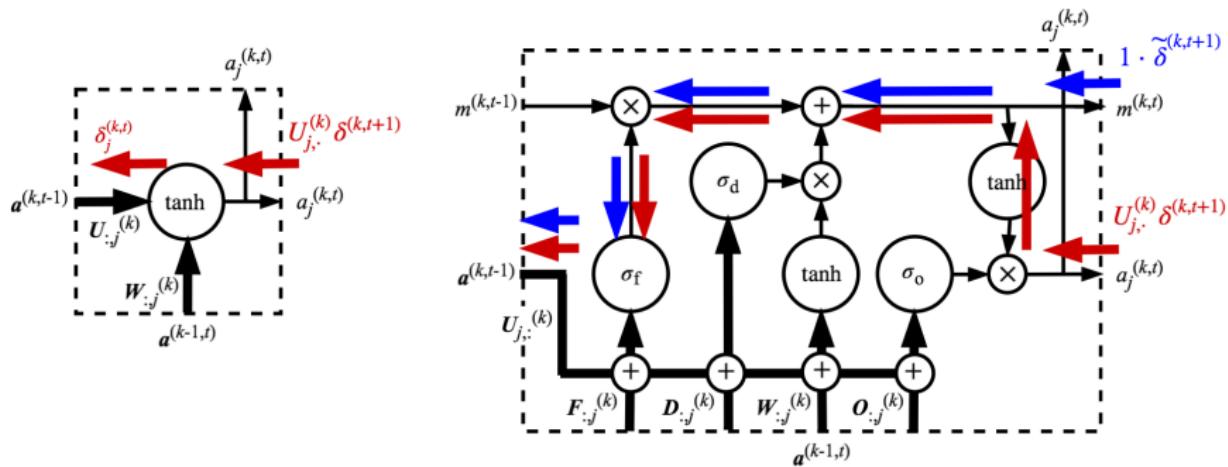
Error Signals

- 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)

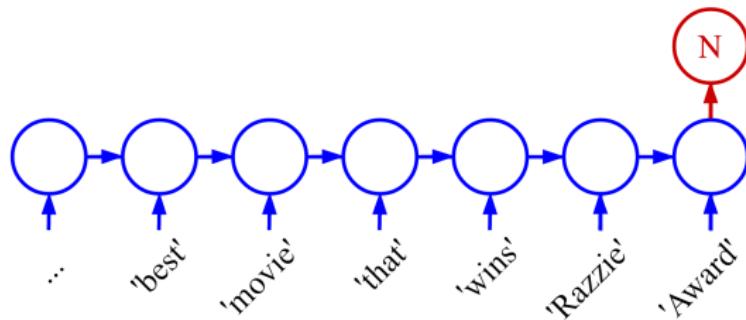


Error Signals

- 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

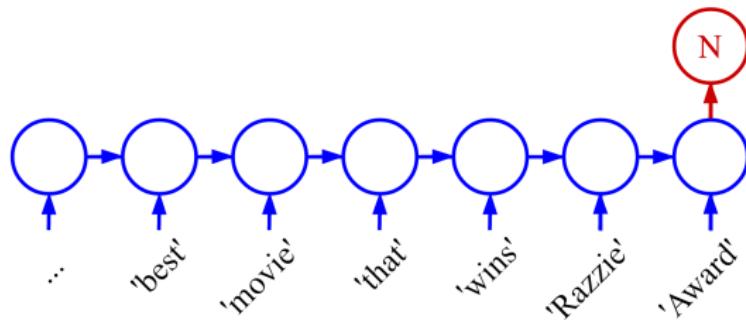


Dynamic Representation



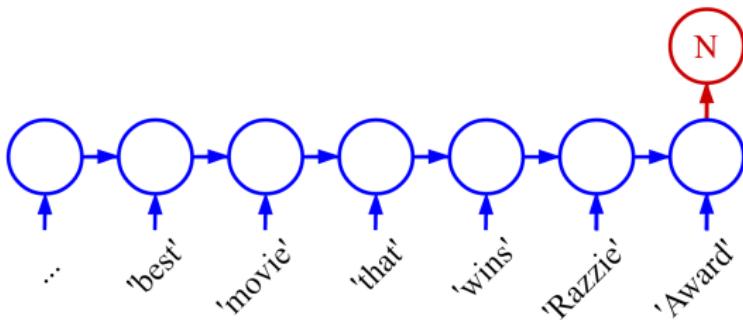
- LSTM units learn dynamic representations

Dynamic Representation



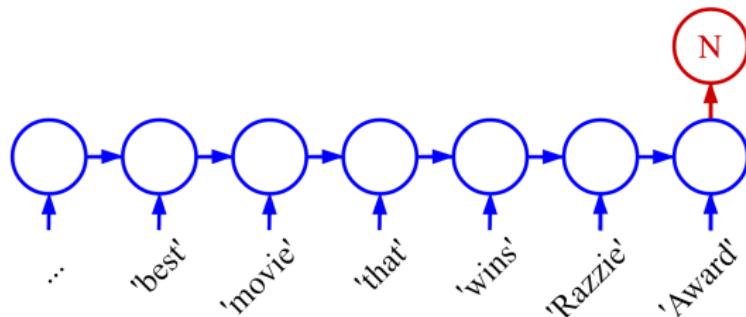
- LSTM units learn dynamic representations
- Closing forget gate for “Razzie”
 - To correct previous summarization and/or shift focus

Dynamic Representation



- 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 Representation



- 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

Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

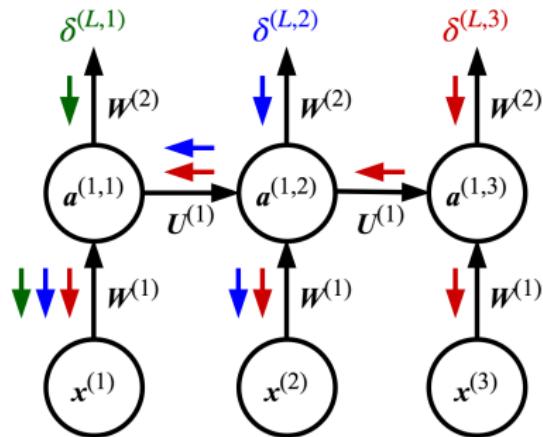
- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

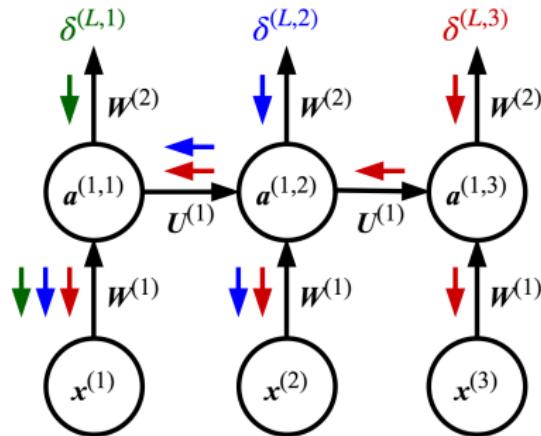
Parallelism

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



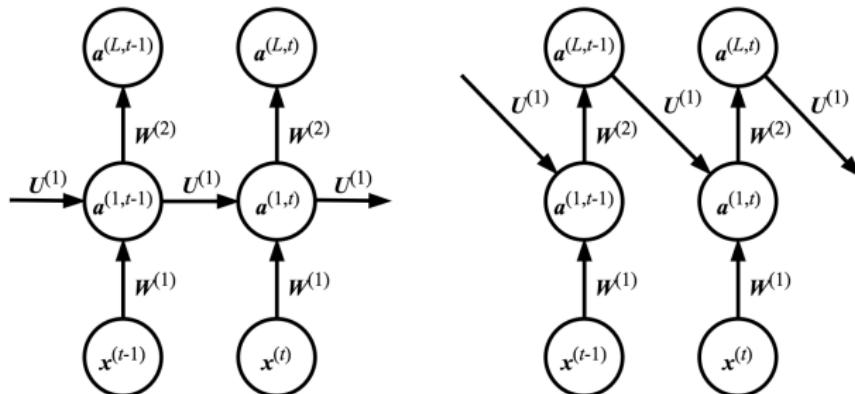
Parallelism

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



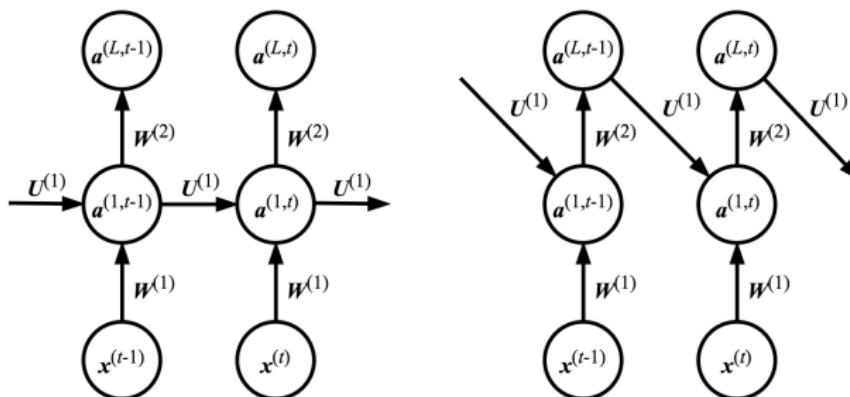
Output Recurrence and Teacher Forcing

- We can instead use a model with **output-to-hidden** recurrence



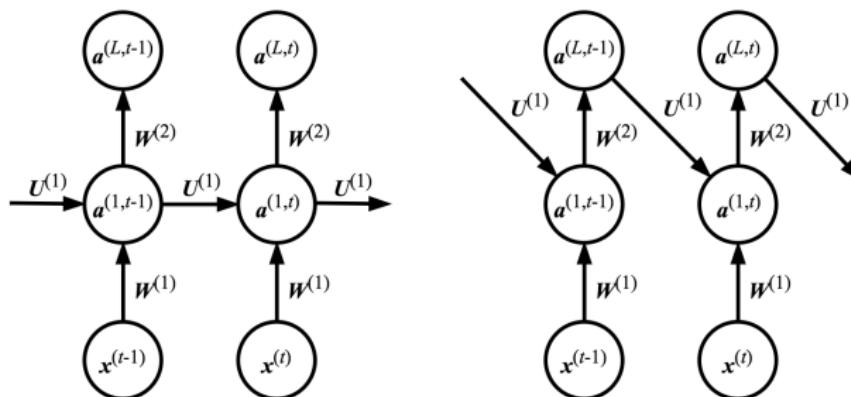
Output Recurrence and Teacher Forcing

- We can instead use a model with ***output-to-hidden*** recurrence
- ***Teacher forcing***: during training time, we replace $a^{(L,t)}$ with $y^{(n,t)}$ from training set for each t



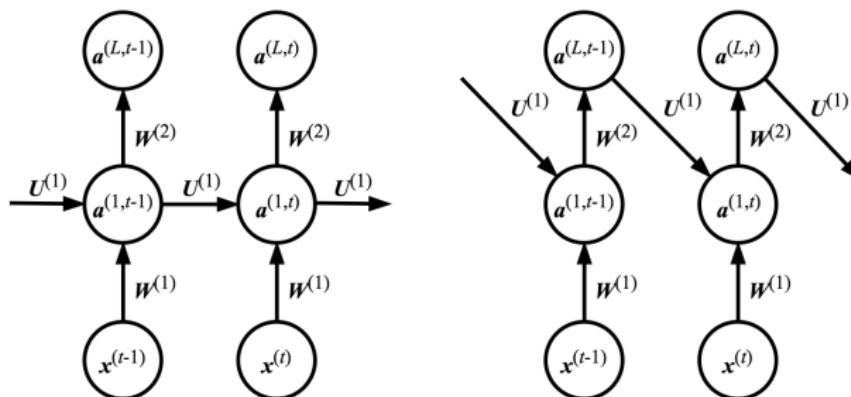
Output Recurrence and Teacher Forcing

- We can instead use a model with ***output-to-hidden*** recurrence
- ***Teacher forcing***: during training time, we replace $a^{(L,t)}$ with $y^{(n,t)}$ from training set for each t
 - There is no dependency between hidden neurons in time



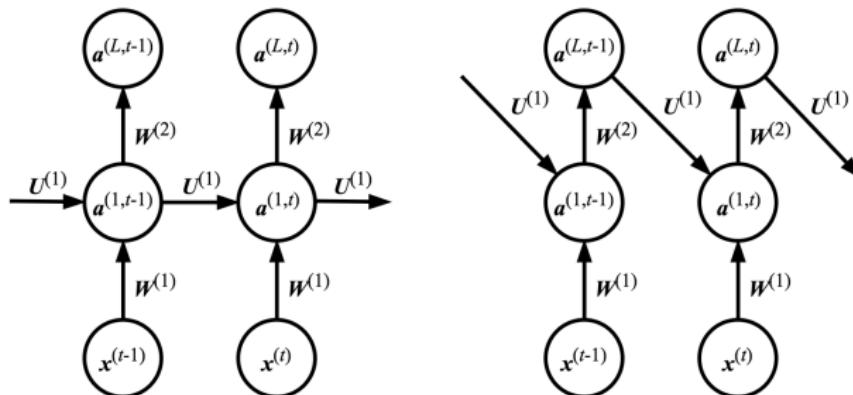
Output Recurrence and Teacher Forcing

- We can instead use a model with ***output-to-hidden*** recurrence
- ***Teacher forcing***: during training time, we replace $a^{(L,t)}$ with $y^{(n,t)}$ from training set for each t
 - There is no dependency between hidden neurons in time
 - A forward/backward pass through time can be parallelized



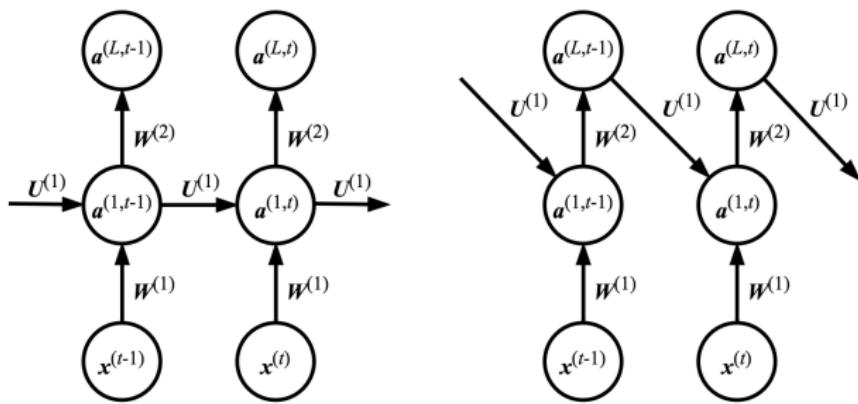
Output Recurrence and Teacher Forcing

- We can instead use a model with ***output-to-hidden*** recurrence
- ***Teacher forcing***: during training time, we replace $a^{(L,t)}$ with $y^{(n,t)}$ from training set for each t
 - There is no dependency between hidden neurons in time
 - A forward/backward pass through time can be parallelized
 - At test time, we switch back to using model output $a^{(L,\cdot)}$'s



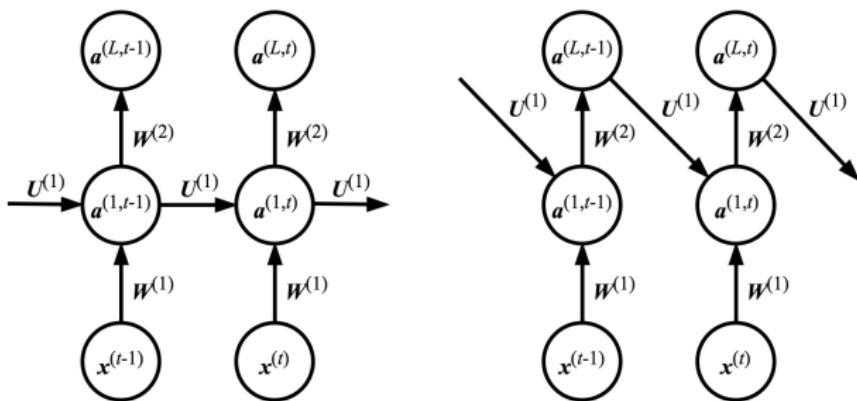
Reduced Expressiveness

- The vanilla RNNs are universal in the sense that they can simulate Turing machines [12]



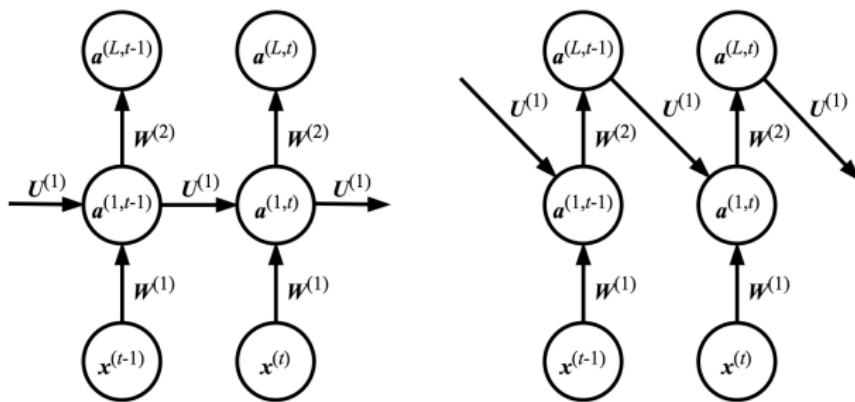
Reduced Expressiveness

- The vanilla RNNs are universal in the sense that they can simulate Turing machines [12]
- 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 [12]
- Output-recurrent RNNs **cannot** simulate Turing machines and are strictly less powerful
- The output $a^{(L,\cdot)}$'s are explicitly trained to match training targets
 - Cannot capture all required information in the past to predict the future



Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

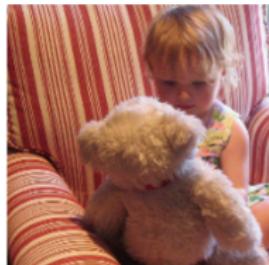
- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

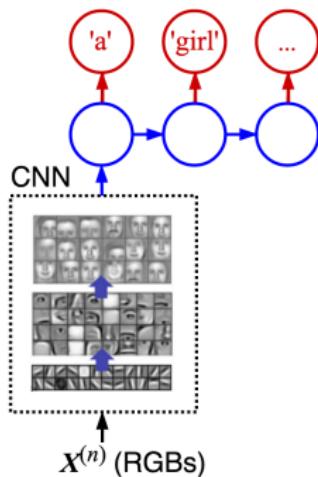
- Visualization
- Memory Networks
- Google Neural Machine Translation

Limited Representation Size

- In some RNNs, a hidden representation $\mathbf{a}^{(\cdot,t)}$ needs to support:
 - Current prediction $\mathbf{a}^{(L,t)}$, and
 - All** future predictions $\mathbf{a}^{(L,t+1)}, \dots, \mathbf{a}^{(L,T)}$

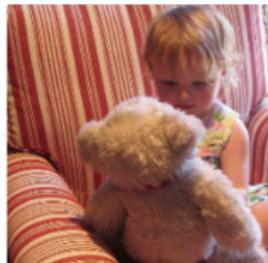


“A little girl sitting on a bed
with a teddy bear.”

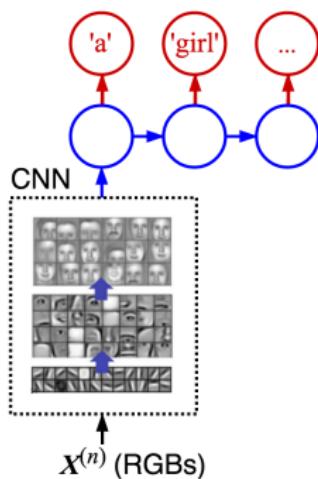


Limited Representation Size

- In some RNNs, a hidden representation $\mathbf{a}^{(\cdot,t)}$ needs to support:
 - Current prediction $\mathbf{a}^{(L,t)}$, and
 - **All** future predictions $\mathbf{a}^{(L,t+1)}, \dots, \mathbf{a}^{(L,T)}$
- The fixed-size $\mathbf{a}^{(t)}$ faces a trade-off between:
 - Representing face features for current prediction ("girl")
 - Representing other features for future predictions ("bear")

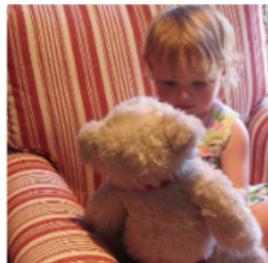


"A little girl sitting on a bed
with a teddy bear."

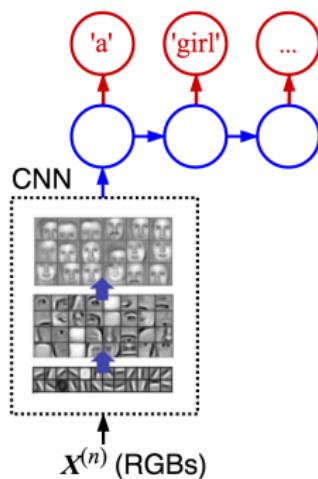


Limited Representation Size

- In some RNNs, a hidden representation $\mathbf{a}^{(\cdot,t)}$ needs to support:
 - Current prediction $\mathbf{a}^{(L,t)}$, and
 - **All** future predictions $\mathbf{a}^{(L,t+1)}, \dots, \mathbf{a}^{(L,T)}$
- The fixed-size $\mathbf{a}^{(t)}$ faces a trade-off between:
 - Representing face features for current prediction ("girl")
 - Representing other features for future predictions ("bear")
- Can we ease the job of $\mathbf{a}^{(\cdot,t)}$?

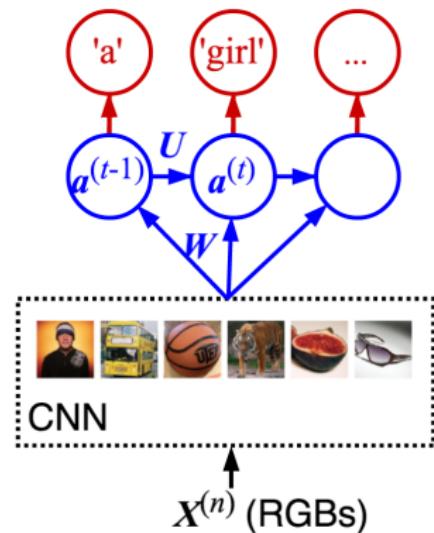


"A little girl sitting on a bed
with a teddy bear."



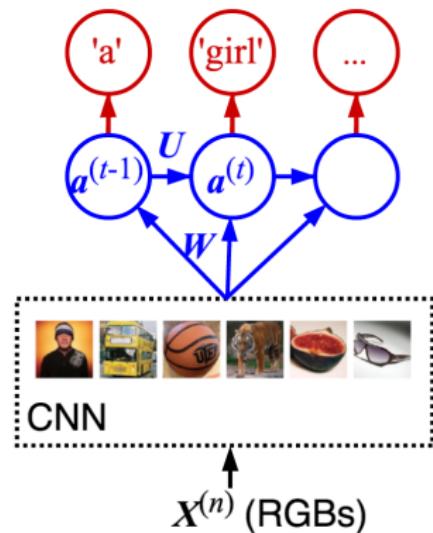
Input Recurrence

- **Input recurrence**: to feed the entire input \mathbf{X} to all $\mathbf{a}^{(1,t)}$'s, $\forall t$



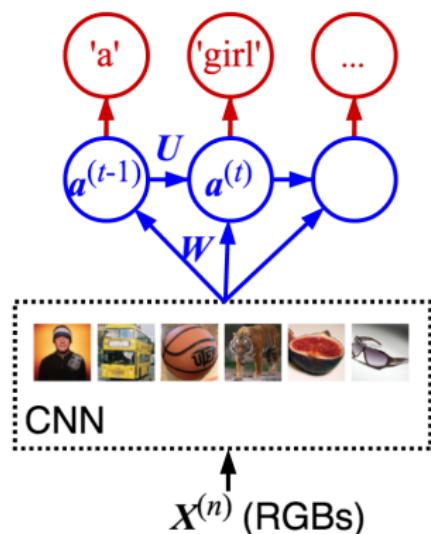
Input Recurrence

- **Input recurrence**: to feed the entire input \mathbf{X} to all $\mathbf{a}^{(1,t)}$'s, $\forall t$
- Now, $\mathbf{a}^{(\cdot,t)}$ only needs to:
 - Support current prediction $\mathbf{a}^{(L,t)}$, and
 - Provide context to the **next** representation $\mathbf{a}^{(\cdot,t+1)}$



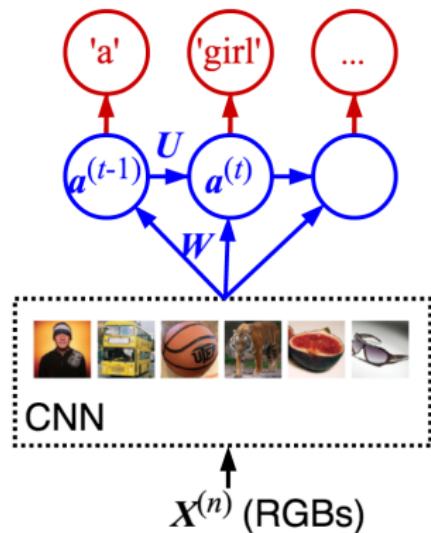
Input Recurrence

- **Input recurrence**: to feed the entire input \mathbf{X} to all $\mathbf{a}^{(1,t)}$'s, $\forall t$
- Now, $\mathbf{a}^{(\cdot,t)}$ only needs to:
 - Support current prediction $\mathbf{a}^{(L,t)}$, and
 - Provide context to the **next** representation $\mathbf{a}^{(\cdot,t+1)}$
- E.g., when predicting “girl,” $\mathbf{a}^{(\cdot,t)}$ may pay attention to only few face-related images features of current input $X^{(n)}$



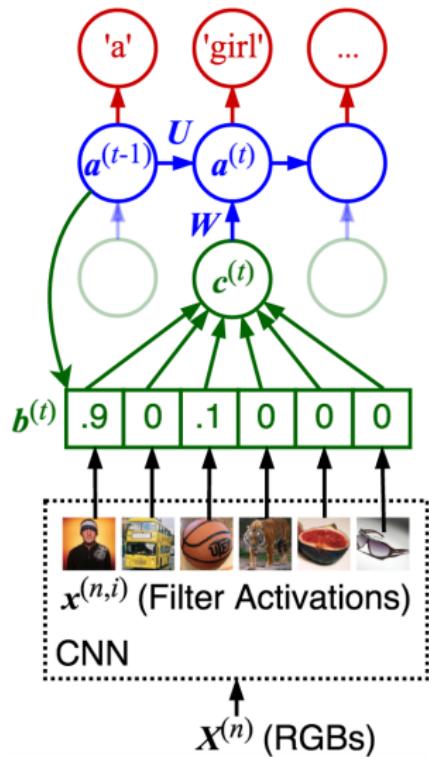
Input Recurrence

- **Input recurrence**: to feed the entire input \mathbf{X} to all $\mathbf{a}^{(1,t)}$'s, $\forall t$
- Now, $\mathbf{a}^{(\cdot,t)}$ only needs to:
 - Support current prediction $\mathbf{a}^{(L,t)}$, and
 - Provide context to the **next** representation $\mathbf{a}^{(\cdot,t+1)}$
- E.g., when predicting “girl,” $\mathbf{a}^{(\cdot,t)}$ may pay attention to only few face-related images features of current input $X^{(n)}$
- Why not model the attention explicitly?
 - So we can see where $\mathbf{a}^{(\cdot,t)}$ is “looking at”



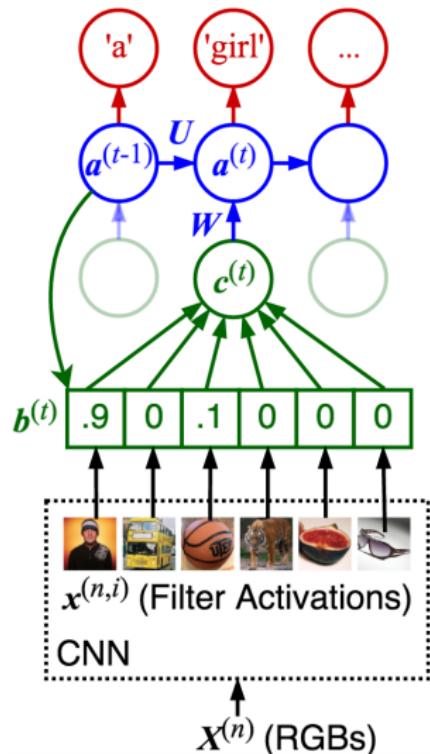
Attention Mechanism

- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into “parts”
 - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter



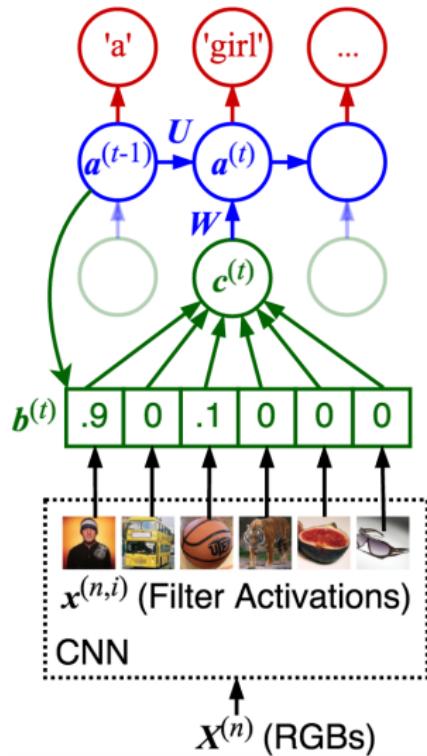
Attention Mechanism

- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into “parts”
 - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter
- For each timestamp t :
 - Obtain from $\mathbf{a}^{(L-1,t-1)}$ an **attention vector** $\mathbf{b}^{(t)}$, $\sum b_i^{(t)} = 1$



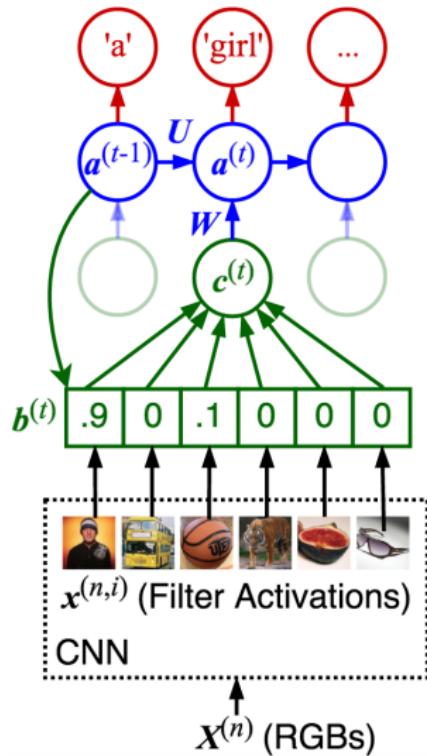
Attention Mechanism

- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into “parts”
 - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter
- For each timestamp t :
 - Obtain from $\mathbf{a}^{(L-1,t-1)}$ an **attention vector** $\mathbf{b}^{(t)}$, $\sum b_i^{(t)} = 1$
 - Feed $\mathbf{a}^{(1,t)}$ with the weighted input $\mathbf{c}^{(t)} = \sum_i b_i^{(t)} \mathbf{x}^{(i)}$



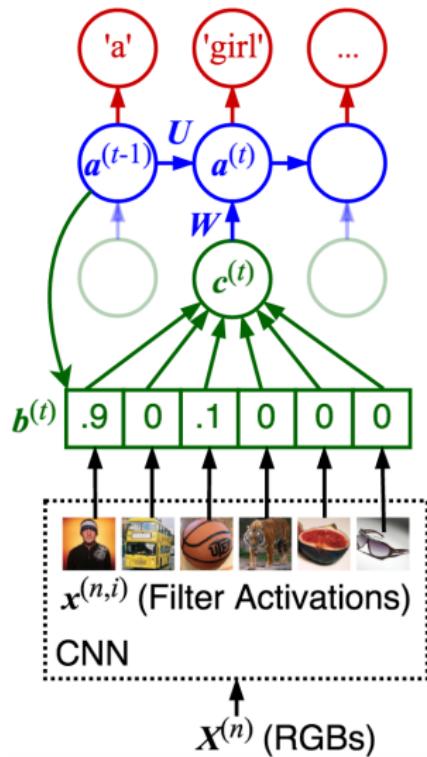
Attention Mechanism

- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into “parts”
 - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter
- For each timestamp t :
 - Obtain from $\mathbf{a}^{(L-1,t-1)}$ an **attention vector** $\mathbf{b}^{(t)}$, $\sum b_i^{(t)} = 1$
 - Feed $\mathbf{a}^{(1,t)}$ with the weighted input $\mathbf{c}^{(t)} = \sum_i b_i^{(t)} \mathbf{x}^{(i)}$
- Reduces size of \mathbf{W} from $O(|\mathbf{X}| \cdot |\mathbf{a}^{(\cdot,t)}|)$ to $O(|\mathbf{x}^{(i)}| \cdot |\mathbf{a}^{(\cdot,t)}|)$



Attention Mechanism

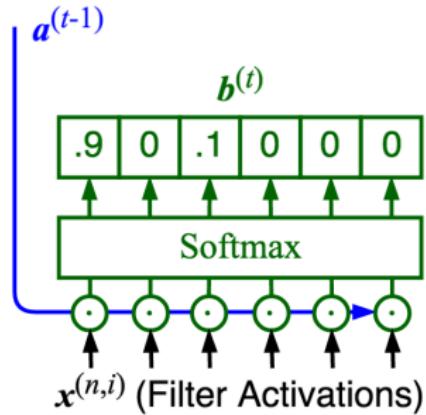
- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into “parts”
 - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter
- For each timestamp t :
 - Obtain from $\mathbf{a}^{(L-1,t-1)}$ an **attention vector** $\mathbf{b}^{(t)}$, $\sum b_i^{(t)} = 1$
 - Feed $\mathbf{a}^{(1,t)}$ with the weighted input $\mathbf{c}^{(t)} = \sum_i b_i^{(t)} \mathbf{x}^{(i)}$
- Reduces size of \mathbf{W} from $O(|\mathbf{X}| \cdot |\mathbf{a}^{(\cdot,t)}|)$ to $O(|\mathbf{x}^{(i)}| \cdot |\mathbf{a}^{(\cdot,t)}|)$
- How to obtain $\mathbf{b}^{(t)}$?



Computing Attention Vector

- ① Use $\mathbf{a}^{(L-1,t-1)}$ as a “query” to get a match score for each input part by using, e.g., a simple NN [2, 15]:

$$z_i = \text{act}(\mathbf{p}^\top \mathbf{a}^{(L-1,t-1)} + \mathbf{q}^\top \mathbf{x}^{(i)} + r)$$



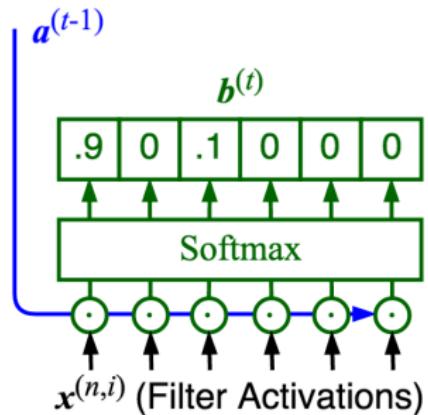
Computing Attention Vector

- ① Use $\mathbf{a}^{(L-1,t-1)}$ as a “query” to get a match score for each input part by using, e.g., a simple NN [2, 15]:

$$z_i = \text{act}(\mathbf{p}^\top \mathbf{a}^{(L-1,t-1)} + \mathbf{q}^\top \mathbf{x}^{(i)} + r)$$

- ② Normalize and concentrate on few larger scores by:

$$b_i = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



Computing Attention Vector

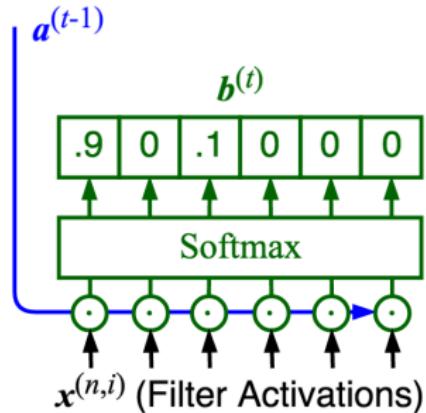
- ① Use $\mathbf{a}^{(L-1,t-1)}$ as a “query” to get a match score for each input part by using, e.g., a simple NN [2, 15]:

$$z_i = \text{act}(\mathbf{p}^\top \mathbf{a}^{(L-1,t-1)} + \mathbf{q}^\top \mathbf{x}^{(i)} + r)$$

- Jointly trained with the main RNN

- ② Normalize and concentrate on few larger scores by:

$$b_i = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



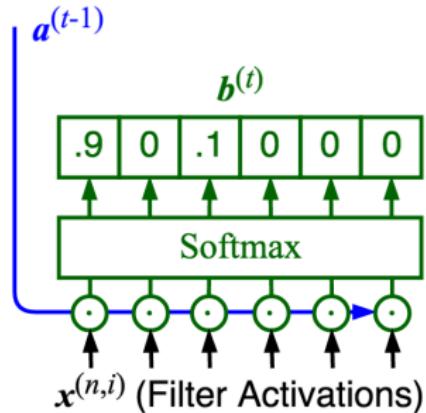
Computing Attention Vector

- ① Use $\mathbf{a}^{(L-1,t-1)}$ as a “query” to get a match score for each input part by using, e.g., a simple NN [2, 15]:

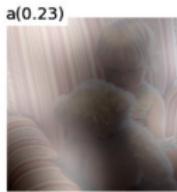
$$z_i = \text{act}(\mathbf{p}^\top \mathbf{a}^{(L-1,t-1)} + \mathbf{q}^\top \mathbf{x}^{(i)} + r)$$

- Jointly trained with the main RNN
 - \mathbf{p} , \mathbf{q} , and r are shared by different i 's and t 's (weight tying)
- ② Normalize and concentrate on few larger scores by:

$$b_i = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



Visualizing Attention



- How to draw a mask?

Visualizing Attention



- How to draw a mask? Threat $\mathbf{c}^{(t)} = \sum_i b_i^{(t)} \mathbf{x}^{(i)}$ as image and enlarge it

Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

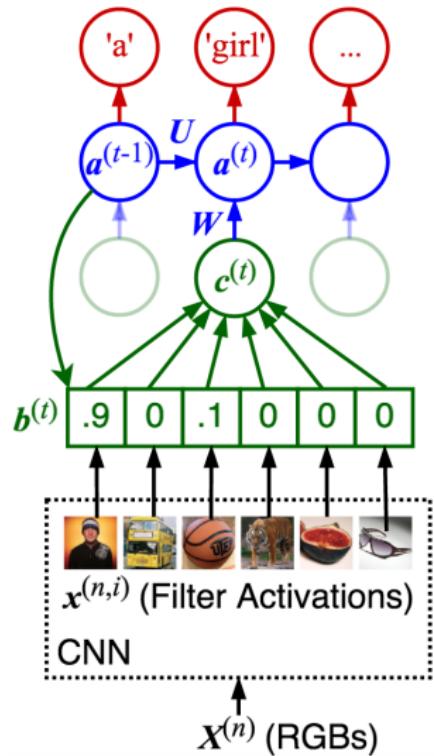
- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

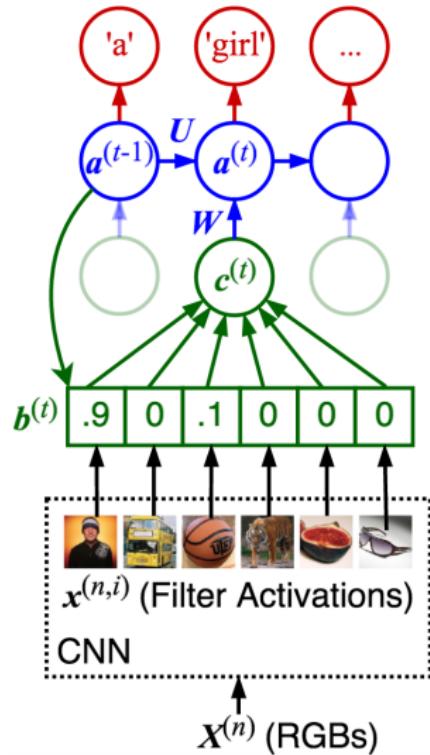
Explicit Memory

- We can regard \mathbf{X} as an *external/explicit* memory



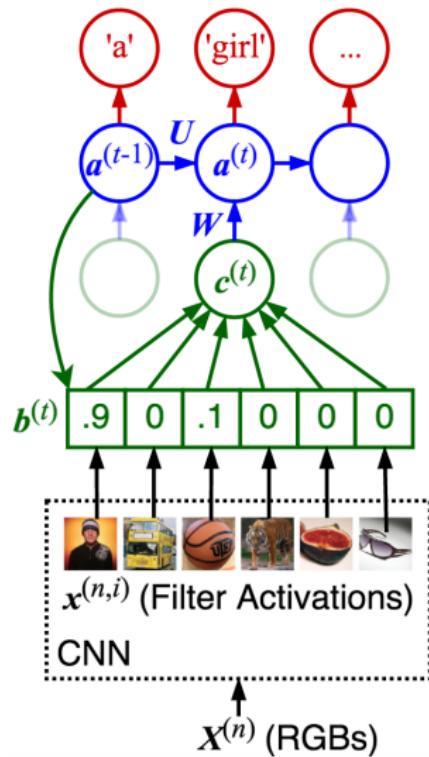
Explicit Memory

- We can regard \mathbf{X} as an *external/explicit* memory
 - Simplify the *internal/implicit* memory $\mathbf{a}^{(\cdot,t)}$

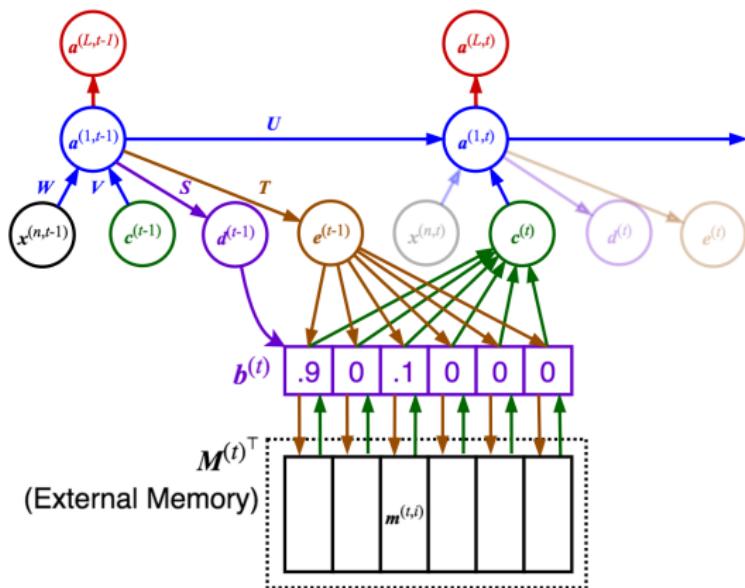


Explicit Memory

- We can regard \mathbf{X} as an *external/explicit* memory
 - Simplify the *internal/implicit* memory $\mathbf{a}^{(\cdot,t)}$
- Why not let the NN decide the content of external memory?

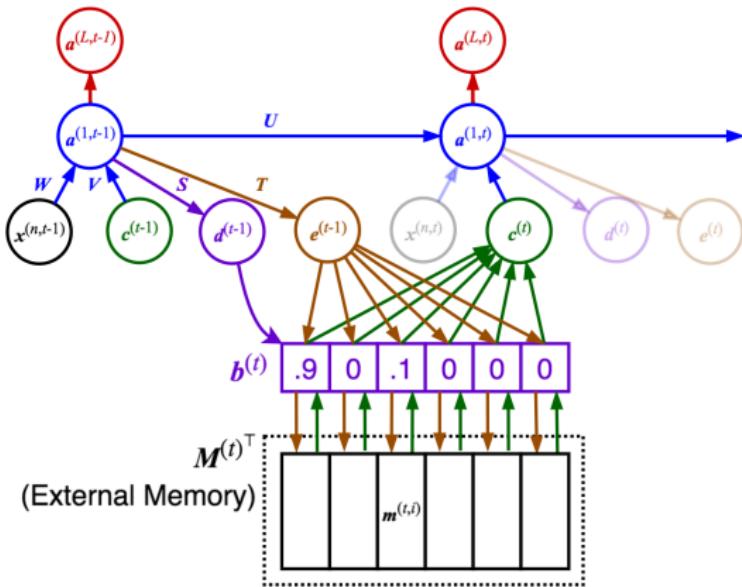


Neural Turing Machines



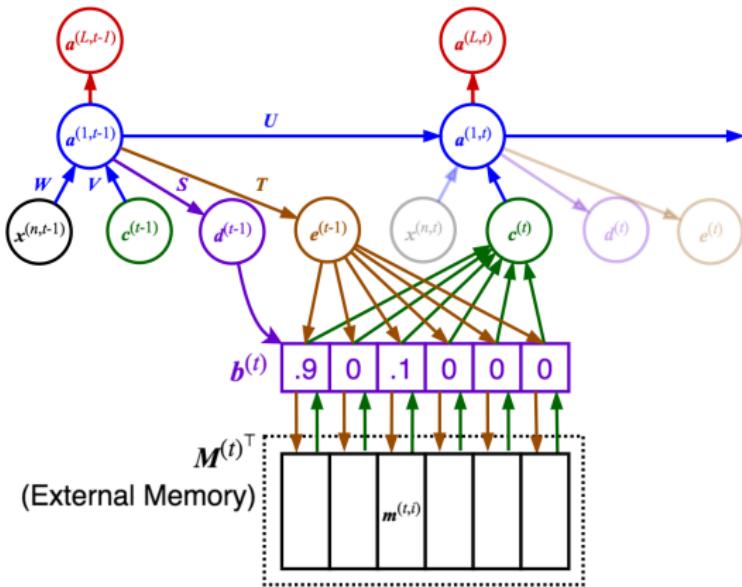
- Attention (contention-based): $b_i^{(t)} = \text{softmax}(\mathbf{M}^{(t-1)} \mathbf{d}^{(t-1)})_i$

Neural Turing Machines



- Attention (contention-based): $b_i^{(t)} = \text{softmax}(\mathbf{M}^{(t-1)} \mathbf{d}^{(t-1)})_i$
 - Simplified, see [4] for location-based attention
- **Write:** $\mathbf{m}^{(i,t)} = (1 - \alpha)\mathbf{m}^{(i,t-1)} + \alpha b_i^{(t)} \mathbf{m}^{(i,t)}, \forall i$ (simplified)

Neural Turing Machines



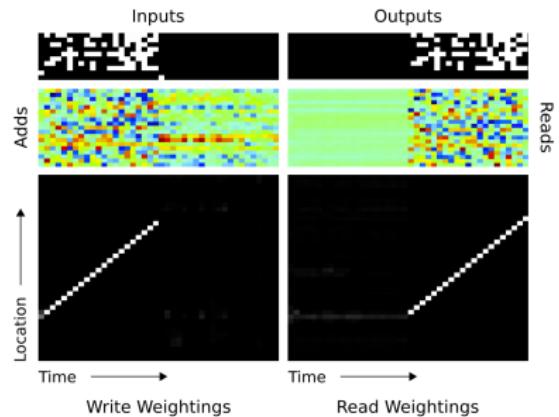
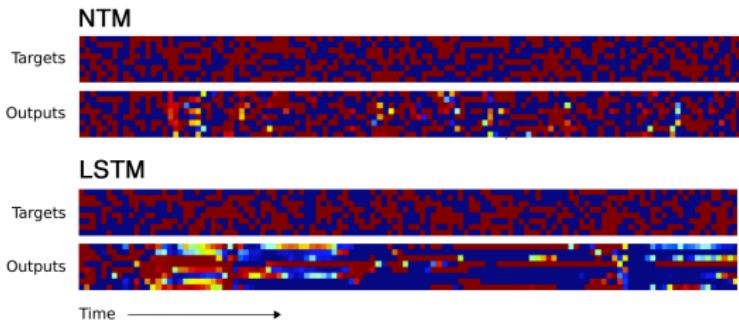
- Attention (contention-based): $b_i^{(t)} = \text{softmax}(M^{(t-1)} d^{(t-1)})_i$
 - Simplified, see [4] for location-based attention
- **Write:** $\mathbf{m}^{(i,t)} = (1 - \alpha)\mathbf{m}^{(i,t-1)} + \alpha b_i^{(t)} \mathbf{m}^{(i,t)}, \forall i$ (simplified)
- Read: $\mathbf{c}^{(t)} = \sum_i b_i^{(t)} \mathbf{m}^{(i,t)}$

Experimental Results

- Task: seq2seq copy
 - Training: length 20
 - Testing: length 120

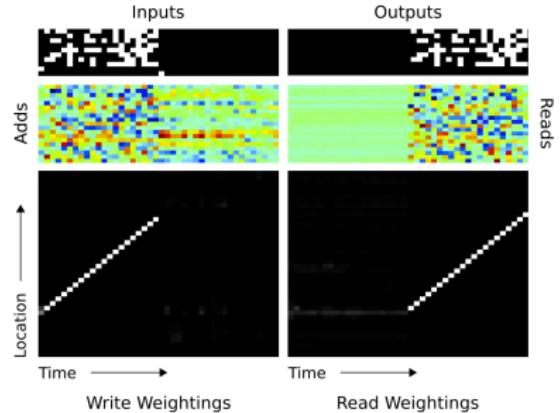
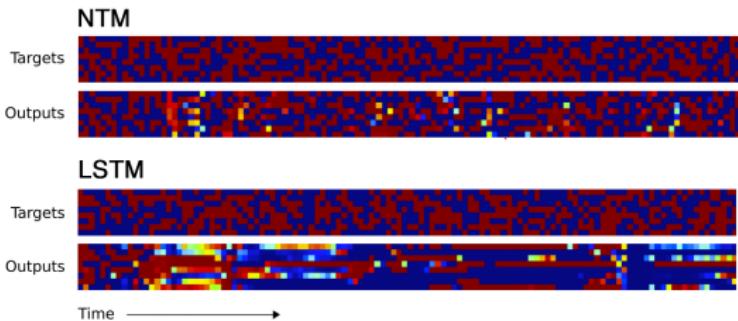
Experimental Results

- Task: seq2seq copy
 - Training: length 20
 - Testing: length 120
- NTM generalizes better



Experimental Results

- Task: seq2seq copy
 - Training: length 20
 - Testing: length 120
- NTM generalizes better
- External read/write weights explain how NN learns

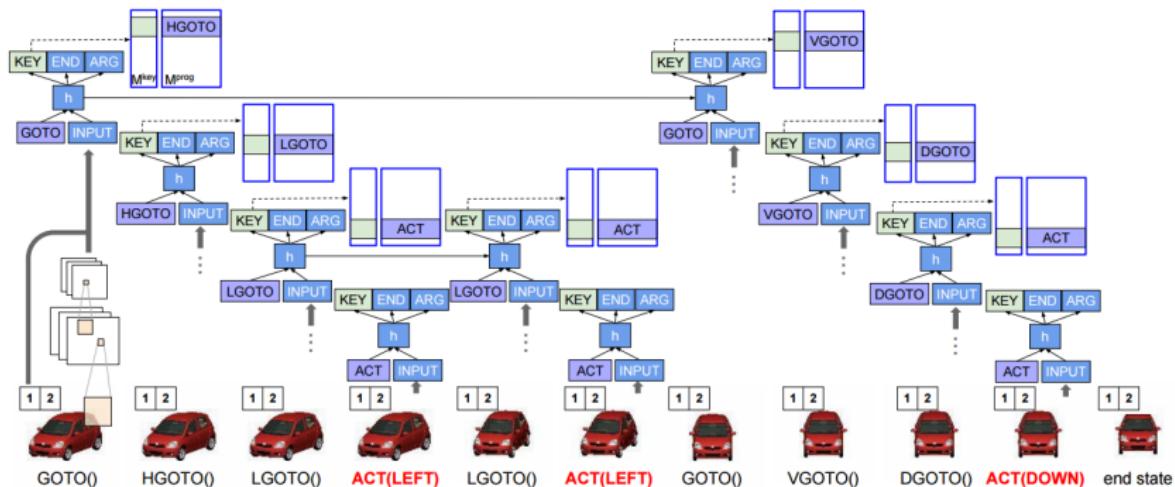


Modeling Priors via Memory Accessing

- Memory structure: pointers [9], stacks/queues [5, 6]

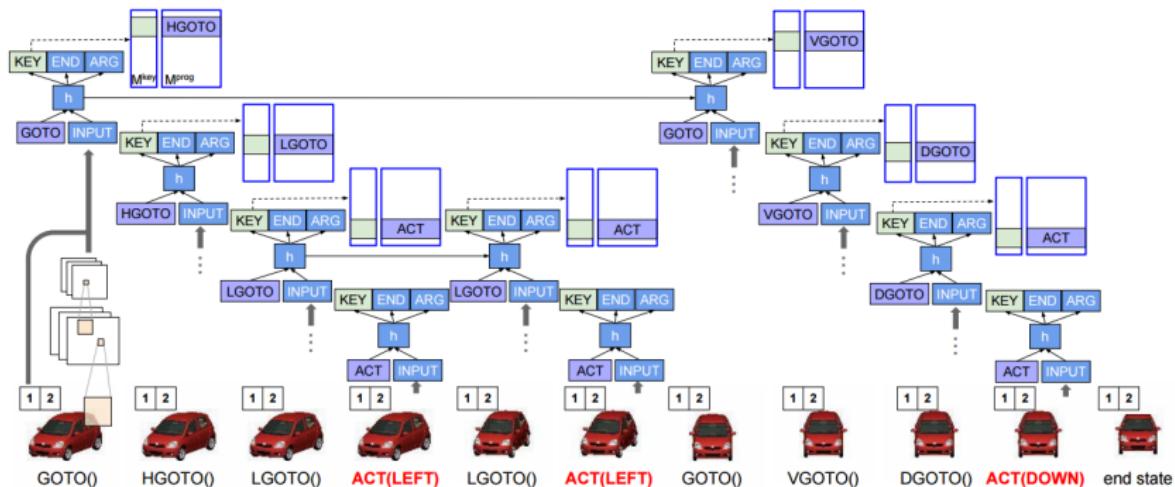
Modeling Priors via Memory Accessing

- Memory structure: pointers [9], stacks/queues [5, 6]
- Memory access: neural programer-interpreter (NPI) [11]
 - Input/output: current/next instructions (synced many2many)



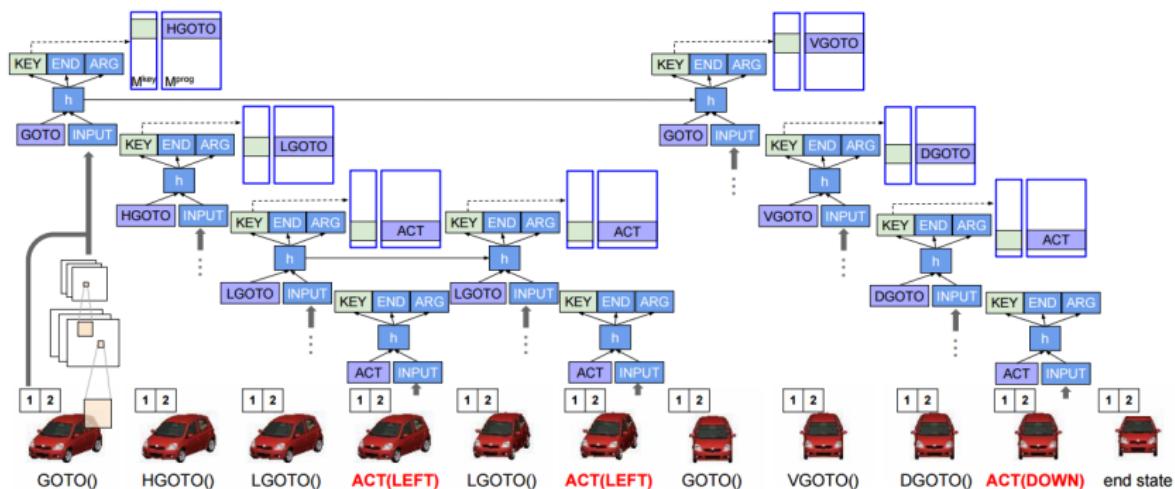
Modeling Priors via Memory Accessing

- Memory structure: pointers [9], stacks/queues [5, 6]
- Memory access: neural programer-interpreter (NPI) [11]
 - Input/output: current/next instructions (synced many2many)
 - RNN models the calling stack



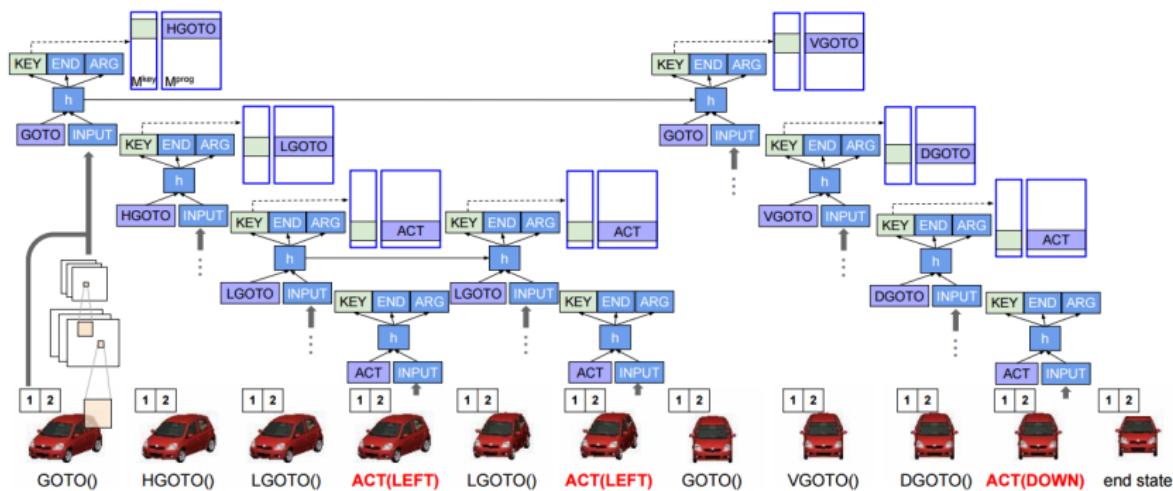
Modeling Priors via Memory Accessing

- Memory structure: pointers [9], stacks/queues [5, 6]
- Memory access: neural programmer-interpreter (NPI) [11]
 - Input/output: current/next instructions (synced many2many)
 - RNN models the calling stack
 - Semi-supervised program memory



Modeling Priors via Memory Accessing

- Memory structure: pointers [9], stacks/queues [5, 6]
- Memory access: neural programmer-interpreter (NPI) [11]
 - Input/output: current/next instructions (synced many2many)
 - RNN models the calling stack
 - Semi-supervised program memory (NPI **adds learned subroutines**)
 - Same NPI trained for different tasks



Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Adaptive Computation Time

- Standard RNNs do the same amount of computation at each time step

Adaptive Computation Time

- Standard RNNs do the same amount of computation at each time step
- Why not “think more” when inputs/predictions are hard?

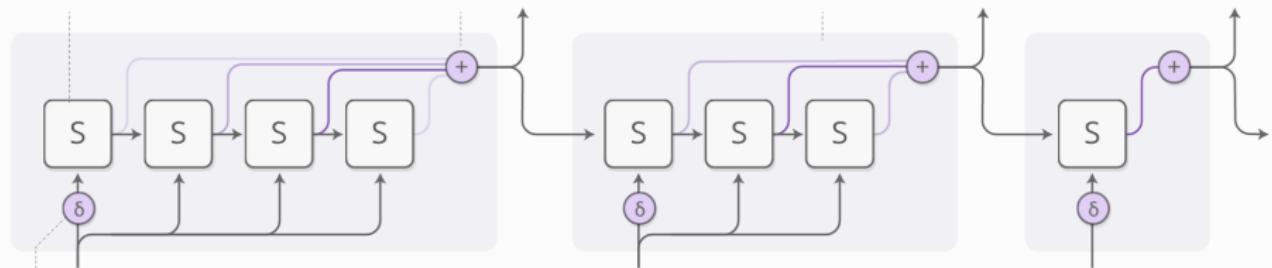
Adaptive Computation Time

- Standard RNNs do the same amount of computation at each time step
- Why not “think more” when inputs/predictions are hard?
- Graves et al. [3] propose the **adaptive computation**:
 - Output at time t is a weighted combination of “states” ($a^{(t,s)}$ ’s)
 - Numbers of states can be different at different t ’s

For every time step the RNN can do multiple computation steps.

The output is a weighted combination of the computation step outputs

The process is repeated for each time step



A special bit is set to denote the first computation step

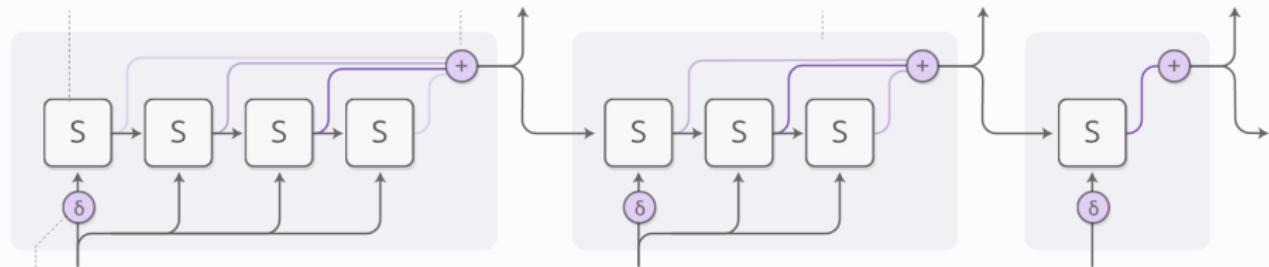
Adaptive Computation Time

- Standard RNNs do the same amount of computation at each time step
- Why not “think more” when inputs/predictions are hard?
- Graves et al. [3] propose the **adaptive computation**:
 - Output at time t is a weighted combination of “states” ($a^{(t,s)}$ ’s)
 - Numbers of states can be different at different t ’s
- Why weighted average?

For every time step the RNN can do multiple computation steps.

The output is a weighted combination of the computation step outputs

The process is repeated for each time step



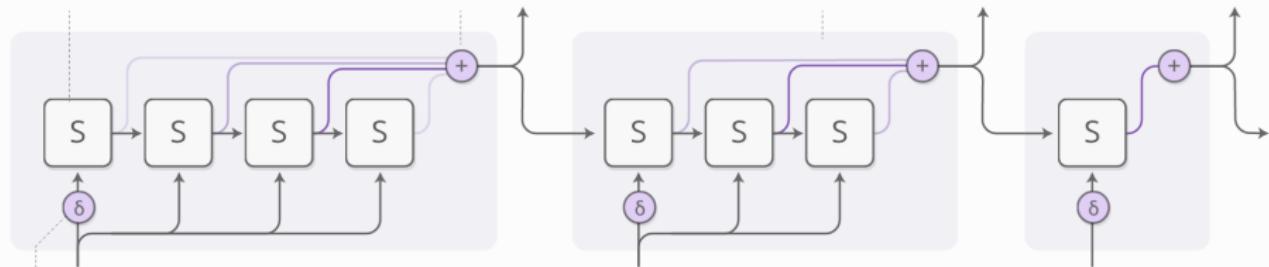
Adaptive Computation Time

- Standard RNNs do the same amount of computation at each time step
- Why not “think more” when inputs/predictions are hard?
- Graves et al. [3] propose the **adaptive computation**:
 - Output at time t is a weighted combination of “states” ($a^{(t,s)}$ ’s)
 - Numbers of states can be different at different t ’s
- Why weighted average? Like (soft) attention, state output with **soft halting probabilities**

For every time step the RNN can do multiple computation steps.

The output is a weighted combination of the computation step outputs

The process is repeated for each time step



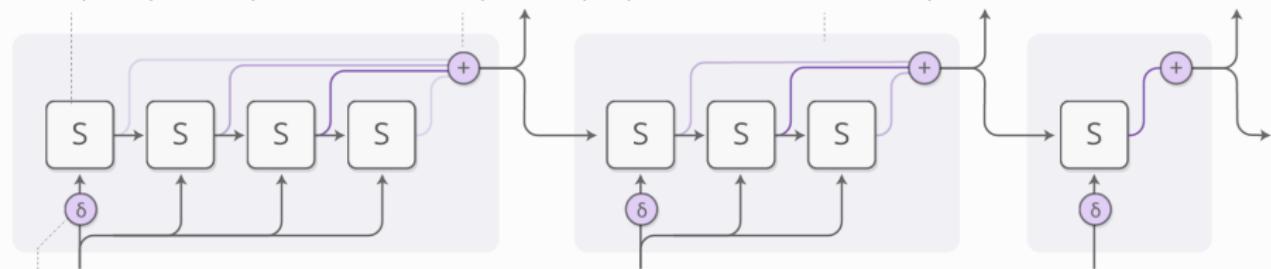
Adaptive Computation Time

- Standard RNNs do the same amount of computation at each time step
- Why not “think more” when inputs/predictions are hard?
- Graves et al. [3] propose the **adaptive computation**:
 - Output at time t is a weighted combination of “states” ($a^{(t,s)}$ ’s)
 - Numbers of states can be different at different t ’s
- Why weighted average? Like (soft) attention, state output with **soft halting probabilities**
- δ augments input so NN can distinguish repeated inputs from repeated computation

For every time step the RNN can do multiple computation steps.

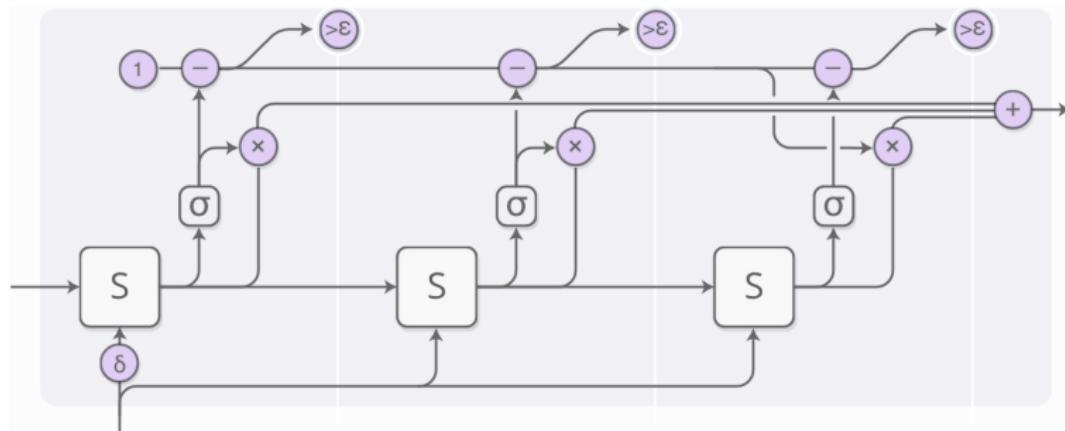
The output is a weighted combination of the computation step outputs

The process is repeated for each time step

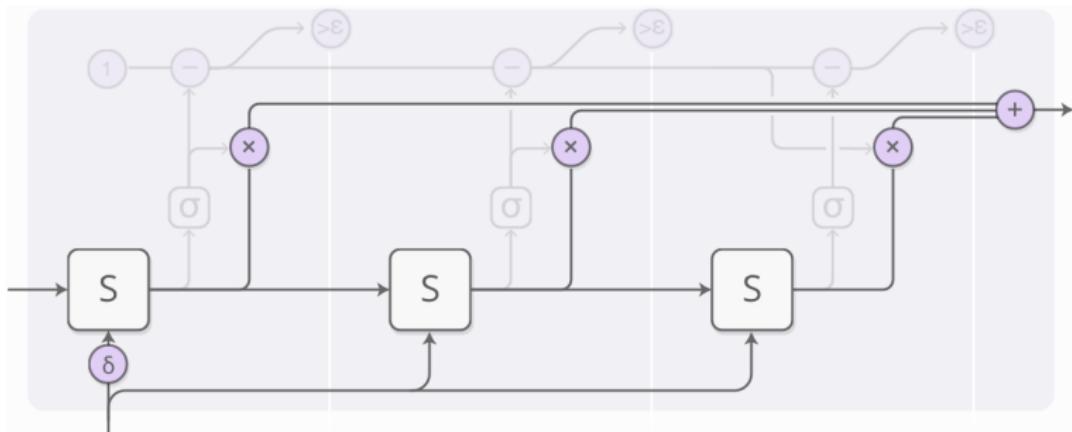


A special bit is set to denote the first computation step

Budgeted Computation at a time Step

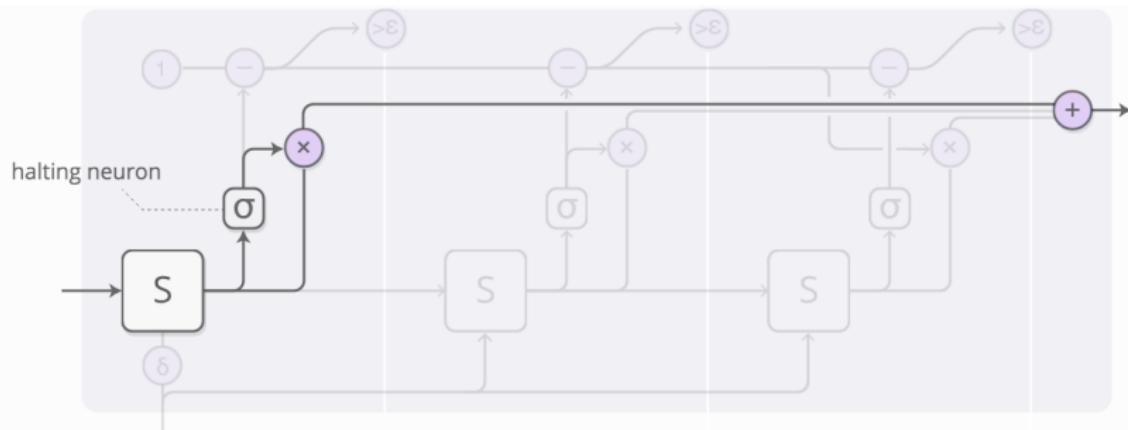


Budgeted Computation at a time Step



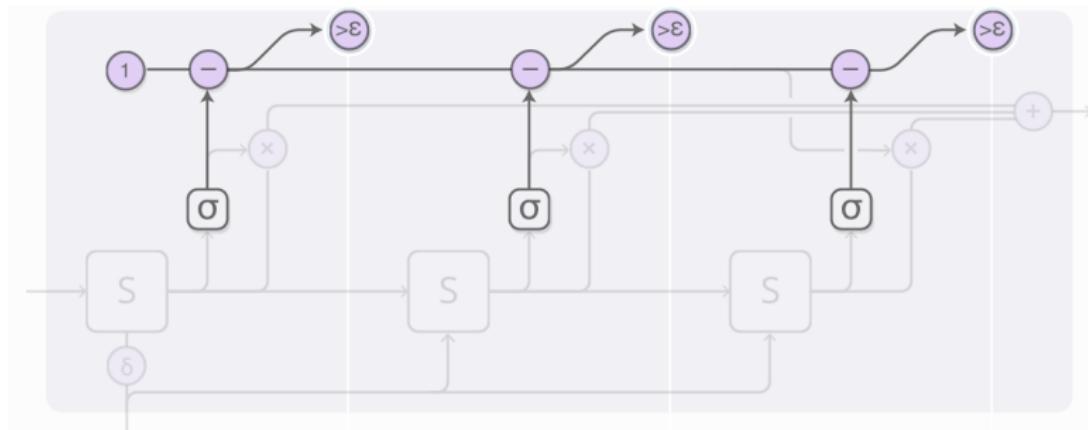
- ① Output is a weighted combination of only the first few states

Budgeted Computation at a time Step



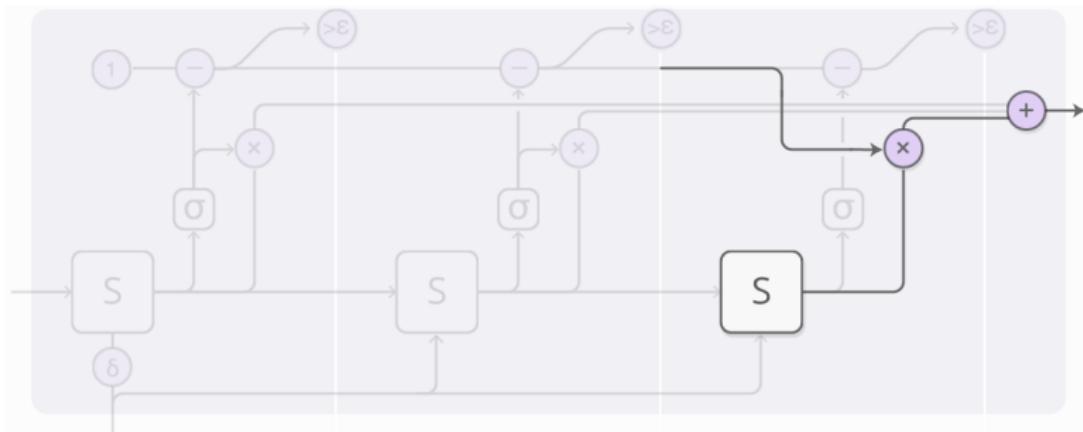
- ① Output is a weighted combination of only the first few states
- ② Weight for each state is determined by a **halting neuron**
 - $h^{(t,s)} = \sigma(w^{\text{halt}\top} a^{(t,s)} + b^{\text{halt}})$

Budgeted Computation at a time Step



- ① Output is a weighted combination of only the first few states
- ② Weight for each state is determined by a *halting neuron*
 - $h^{(t,s)} = \sigma(w^{\text{halt}\top} a^{(t,s)} + b^{\text{halt}})$
- ③ Total budget for halting weights is 1
 - Eventually, there will be a state $\mu^{(t)}$ that runs out of budget,
$$\mu^{(t)} = \min\{\mu : \sum_{s=1}^{\mu} h^{(t,s)} > 1 - \varepsilon\}$$

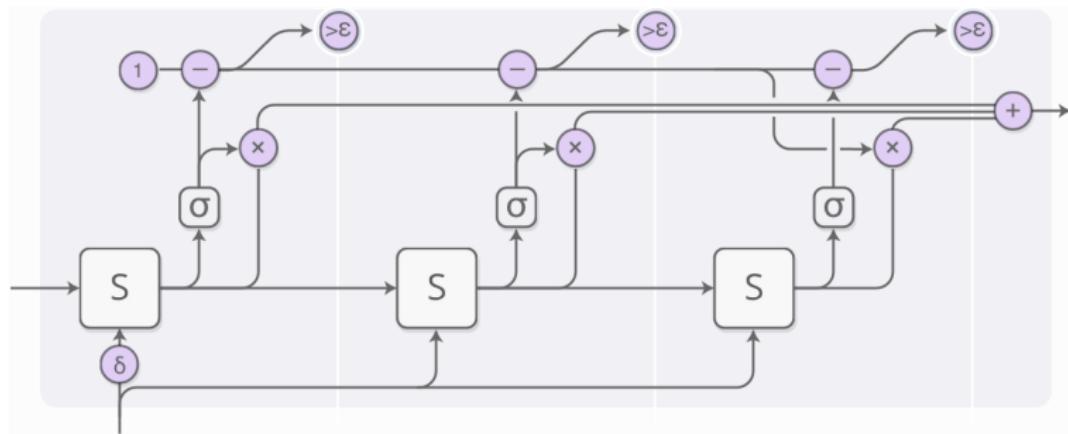
Budgeted Computation at a time Step



- ① Output is a weighted combination of only the first few states
- ② Weight for each state is determined by a *halting neuron*
 - $h^{(t,s)} = \sigma(w^{\text{halt}}{}^\top a^{(t,s)} + b^{\text{halt}})$
- ③ Total budget for halting weights is 1
 - Eventually, there will be a state $\mu^{(t)}$ that runs out of budget,
$$\mu^{(t)} = \min\{\mu : \sum_{s=1}^{\mu} h^{(t,s)} > 1 - \varepsilon\}$$
- ④ Assign to state $\mu^{(t)}$ the left over halting budget $1 - \sum_{s=1}^{\mu^{(t)}-1} h^{(t,s)}$

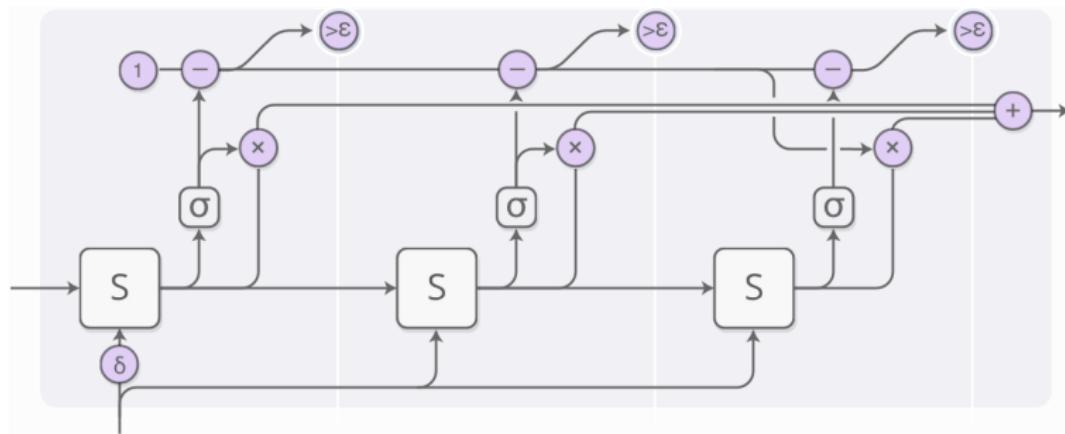
The Ponder Cost

- Without any regularization, states tend to “ponder” each input for as long as possible
 - To make use of as many states as possible to reduce error



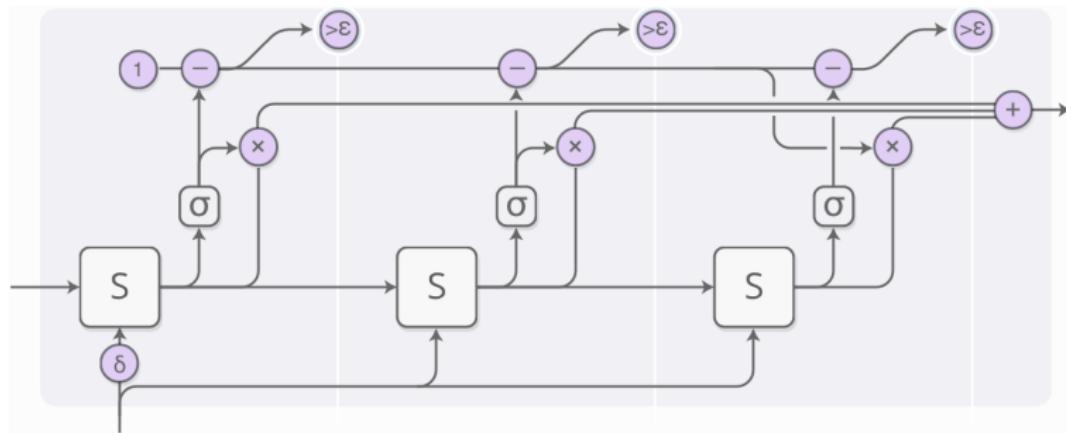
The Ponder Cost

- Without any regularization, states tend to “ponder” each input for as long as possible
 - To make use of as many states as possible to reduce error
- Adds a “ponder penalty” to cost: $\sum_t \left(1 - \sum_{s=1}^{\mu^{(t)}-1} h^{(t,s)}\right)$
 - Discourages the left over budget (for **any** state and input at time t)



The Ponder Cost

- Without any regularization, states tend to “ponder” each input for as long as possible
 - To make use of as many states as possible to reduce error
- Adds a “ponder penalty” to cost: $\sum_t \left(1 - \sum_{s=1}^{\mu^{(t)}-1} h^{(t,s)} \right)$
 - Discourages the left over budget (for **any** state and input at time t)
 - States end up using budget as early as possible



Experimental Results I

- Task: seq2seq sorting
 - Input seq: unsorted numbers
 - Output seq: sort indices

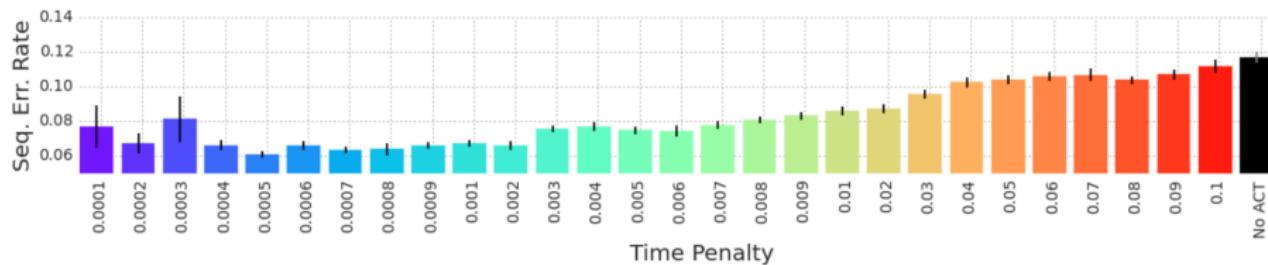
Experimental Results I

- Task: seq2seq sorting
 - Input seq: unsorted numbers
 - Output seq: sort indices
- Reduced error rate at lower ponder cost (higher computation)

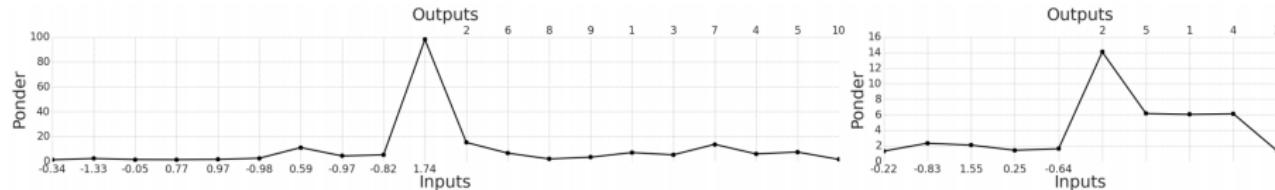


Experimental Results I

- Task: seq2seq sorting
 - Input seq: unsorted numbers
 - Output seq: sort indices
- Reduced error rate at lower ponder cost (higher computation)



- Ponder time (#states at each time) provides insights into when comparisons take place

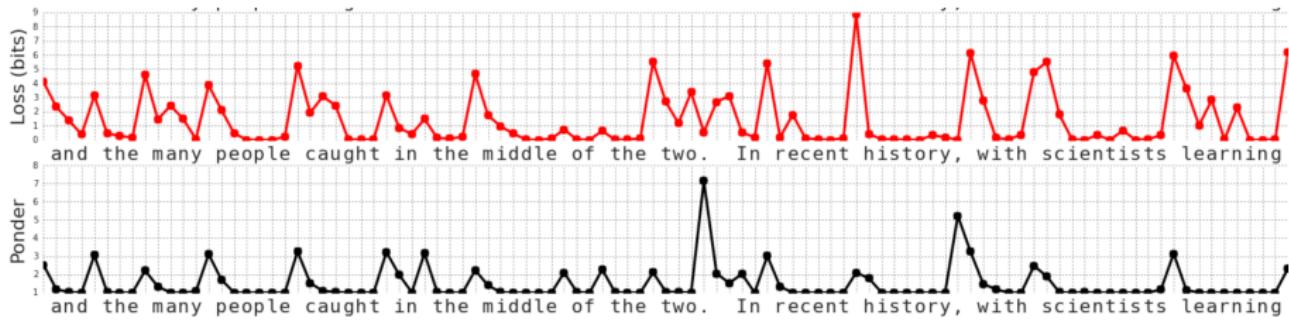


Experimental Results II

- Task: language modeling
- Dataset: Wiki

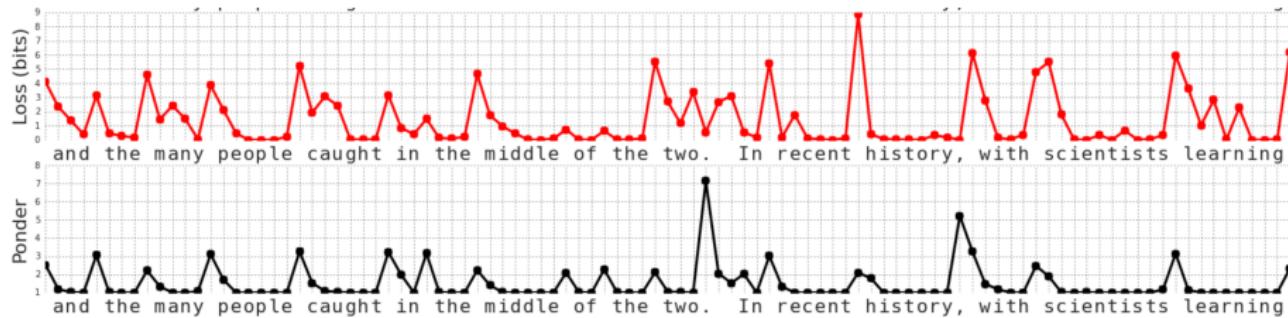
Experimental Results II

- Task: language modeling
- Dataset: Wiki
- Longer ponder time (#states at each time) before predicting the next “chunk”



Experimental Results II

- Task: language modeling
- Dataset: Wiki
- Longer ponder time (#states at each time) before predicting the next “chunk”
- Learning implicit boundaries/transitions in sequence (unsupervised)
 - More robust than learning from prediction losses/errors



Outline

1 Design

- Vanilla RNNs
- Design Alternatives

2 Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

3 Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

4 Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

Outline

1 Design

- Vanilla RNNs
- Design Alternatives

2 Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

3 Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

4 Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

What Do RNN Neurons Learn?

What Do RNN Neurons Learn?

- Neuron activations for language modeling [7] [Interactive Tool](#)

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact  
that it plainly and indubitably proved the fallacy of all the plans for  
cutting off the enemy's retreat and the soundness of the only possible  
line of action--the one Kutuzov and the general mass of the army  
demanded--namely, simply to follow the enemy up. The French crowd fled  
at a continually increasing speed and all its energy was directed to  
reaching its goal. It fled like a wounded animal and it was impossible  
to block its path. This was shown not so much by the arrangements it  
made for crossing as by what took place at the bridges. When the bridges  
broke down, unarmed soldiers, people from Moscow and women with children  
who were with the French transport, all--carried on by vis inertiae--  
pressed forward into boats and into the ice-covered water and did not,  
surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the  
contrary, I can supply you with everything even if you want to give  
dinner parties," warmly replied Chichagov, who tried by every word he  
spoke to prove his own rectitude and therefore imagined Kutuzov to be  
animated by the same desire.
```

```
Kutuzov, shrugging his shoulders, replied with his subtle penetrating  
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,  
    siginfo_t *info)  
{  
    int sig = next_signal(pending, mask);  
    if (sig) {  
        if (current->notifier) {  
            if (sigismember(current->notifier_mask, sig)) {  
                if (!!(current->notifier)(current->notifier_data)) {  
                    clear_thread_flag(TIF_SIGPENDING);  
                    return 0;  
                }  
            }  
        }  
        collect_signal(sig, pending, info);  
    }  
    return sig;  
}
```

When Did RNN Neurons Learn?

- Output at epoch 100:

“... tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh...”

When Did RNN Neurons Learn?

- Output at epoch 100:
“... *tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh...*”
- At 300 (spaces and periods):
“... *Phe lism thond hon at. MeiDimorotion in ther...*”

When Did RNN Neurons Learn?

- Output at epoch 100:
“... tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh...”
- At 300 (spaces and periods):
“... Phe lism thond hon at. MeiDimorotion in ther...”
- At 500 (common words “we,” “he,” etc.):
“... we counter. He stutn co des. His stanted out one...”

When Did RNN Neurons Learn?

- Output at epoch 100:
“... *tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh...*”
- At 300 (spaces and periods):
“... *Phe lism thond hon at. MeiDimorotion in ther...*”
- At 500 (common words “we,” “he,” etc.):
“... *we counter. He stutn co des. His stanted out one...*”
- At 700 (English-like structure):
“... *Aftair fall unsuch that the hall for Prince Velzonski's...*”

When Did RNN Neurons Learn?

- Output at epoch 100:
“... tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh...”
- At 300 (spaces and periods):
“... Phe lism thond hon at. MeiDimorotion in ther...”
- At 500 (common words “we,” “he,” etc.):
“... we counter. He stutn co des. His stanted out one...”
- At 700 (English-like structure):
“... Aftair fall unsuch that the hall for Prince Velzonski's...”
- At 1200 (quotations and longer words):
“... “Kite vouch!” he repeated by her door...”

When Did RNN Neurons Learn?

- Output at epoch 100:
“... tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh...”
- At 300 (spaces and periods):
“... Phe lism thond hon at. MeiDimorotion in ther...”
- At 500 (common words “we,” “he,” etc.):
“... we counter. He stutn co des. His stanted out one...”
- At 700 (English-like structure):
“... Aftair fall unsuch that the hall for Prince Velzonski's...”
- At 1200 (quotations and longer words):
“... “Kite vouch!” he repeated by her door...”
- At 2000 (topics and longer-term dependencies):
“... “Why do what that day,” replied Natasha, ...”

Outline

1 Design

- Vanilla RNNs
- Design Alternatives

2 Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

3 Augmented RNNs

- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

4 Case Studies

- Visualization
- **Memory Networks**
- Google Neural Machine Translation

Reading Comprehension & QA

“Joe went to the kitchen. Fred went to the kitchen.

Joe picked up the milk. Joe travelled to the office.

Joe left the milk. Joe went to the bathroom.”

- Where is the milk now?
- Where is Joe?
- Where was Joe before the office?

Reading Comprehension & QA

"Joe went to the kitchen. Fred went to the kitchen.

Joe picked up the milk. Joe travelled to the office.

Joe left the milk. Joe went to the bathroom."

- Where is the milk now? **A: office**
- Where is Joe? **A: bathroom**
- Where was Joe before the office? **A: kitchen**

Memory Networks

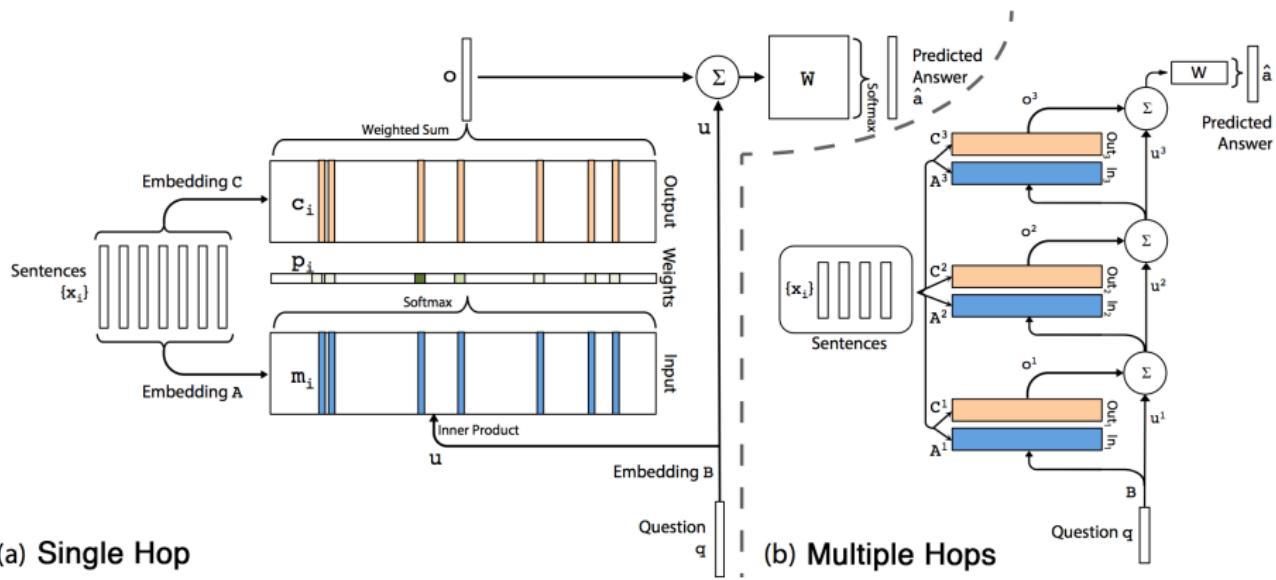
“Joe went to the kitchen. ... to the office. ... to the bathroom.”

- One-time attention may not be sufficient

Memory Networks

“Joe went to the kitchen. ... to the office. ... to the bathroom.”

- One-time attention may not be sufficient
- **Hopping** in memory networks [14, 13]:



(a) Single Hop

(b) Multiple Hops

Outline

① Design

- Vanilla RNNs
- Design Alternatives

② Training

- Backprop through Time (BPTT)
- Optimization Techniques
- Optimization-Friendly Models & LSTM
- Parallelism & Teacher Forcing

③ Augmented RNNs

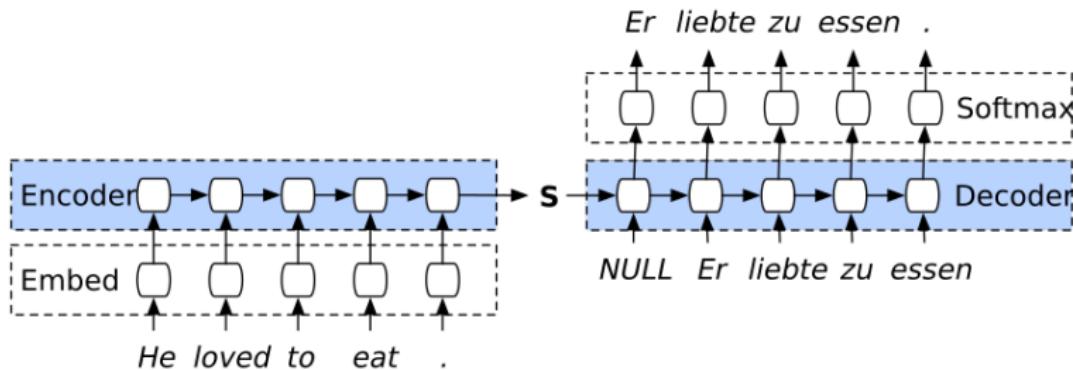
- Attention
- Explicit Memory
- Adaptive Computation Time (ACT)

④ Case Studies

- Visualization
- Memory Networks
- Google Neural Machine Translation

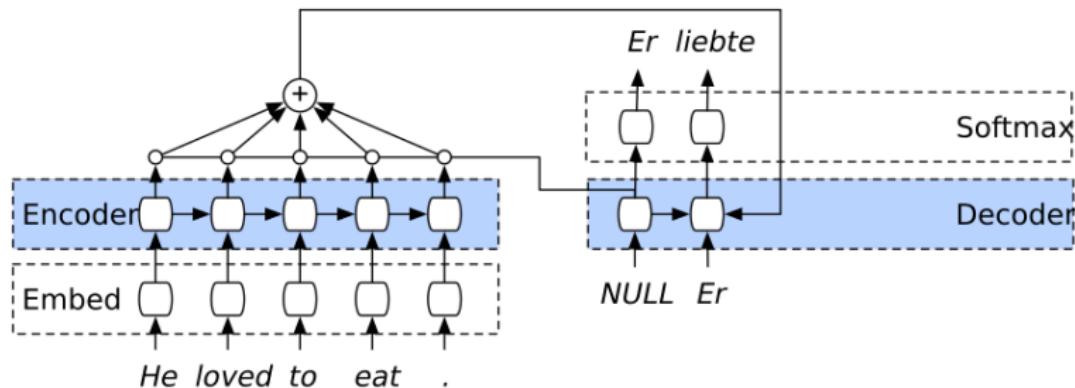
V1: Encoder-Decoder

- LSTM-based RNN
- Hidden state S encodes an entire input sequence
- Then supplied to the decoder



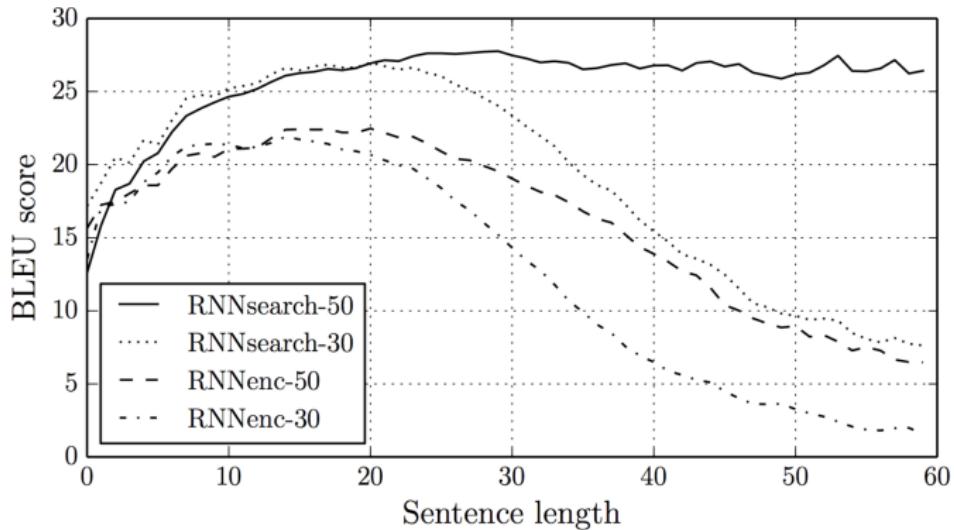
V2: Attention-based Encoder-Decoder

- Allows for retrieving different parts of input sentence depending on decoding context
- No direct connection between the encoder and decoder



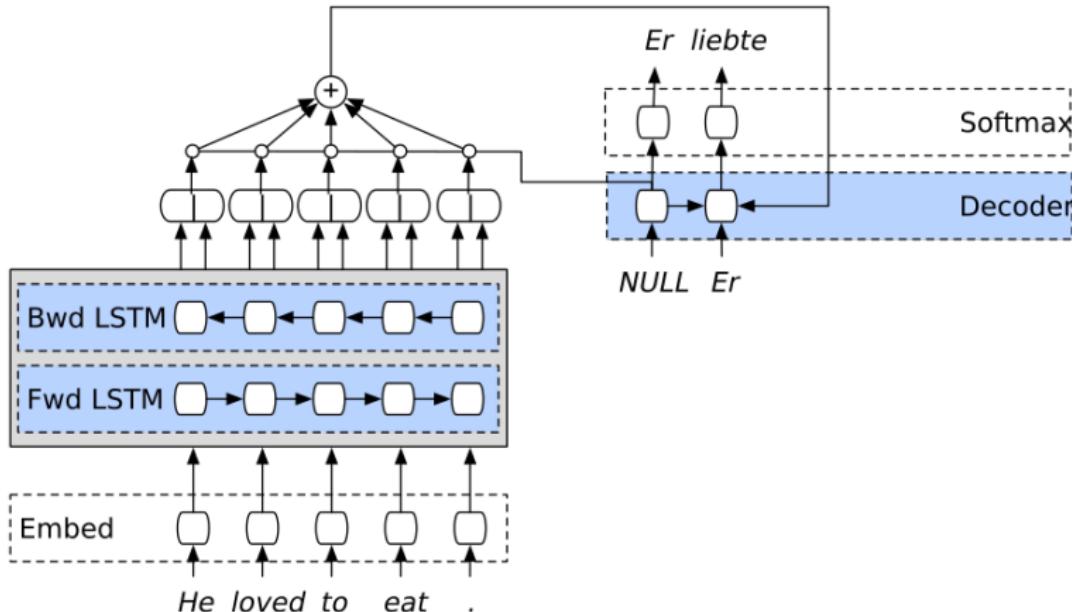
Long Sequences

- Attention-based model generates long sequences better



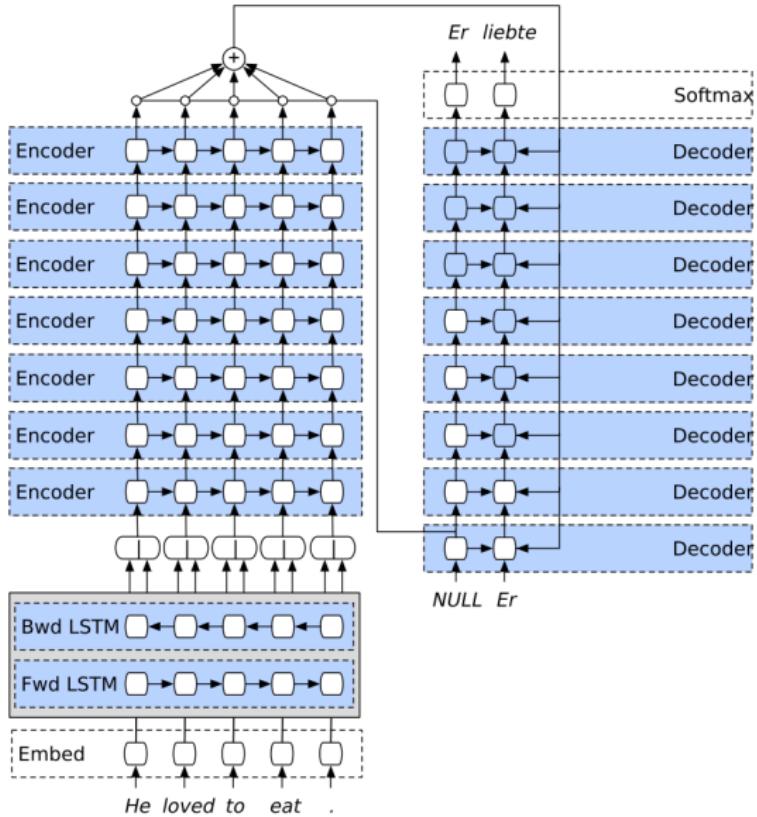
V3: Bidirectional Encoder Layer

- Takes into account future words when summarizing input sequence
- Better determines the meaning/context



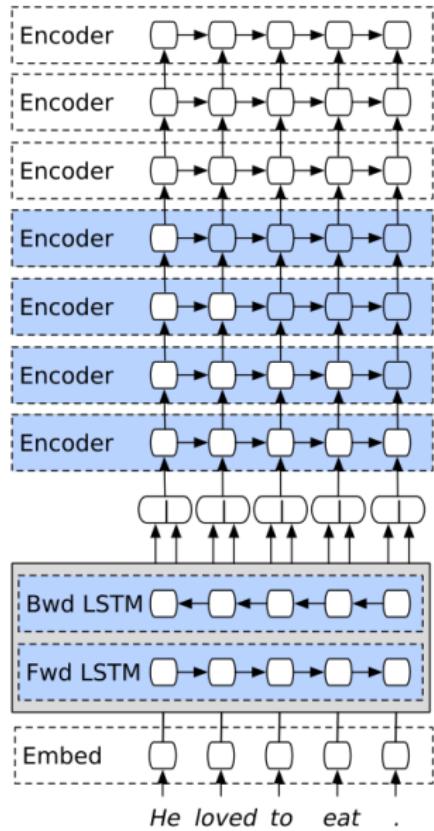
V4: Going Deep

- Encoder:
 - 1 bi-directional layer
 - 7 uni-directional layers
- Decoder:
 - 8 uni-directional layers
 - Lowest** decoder layer for querying attention

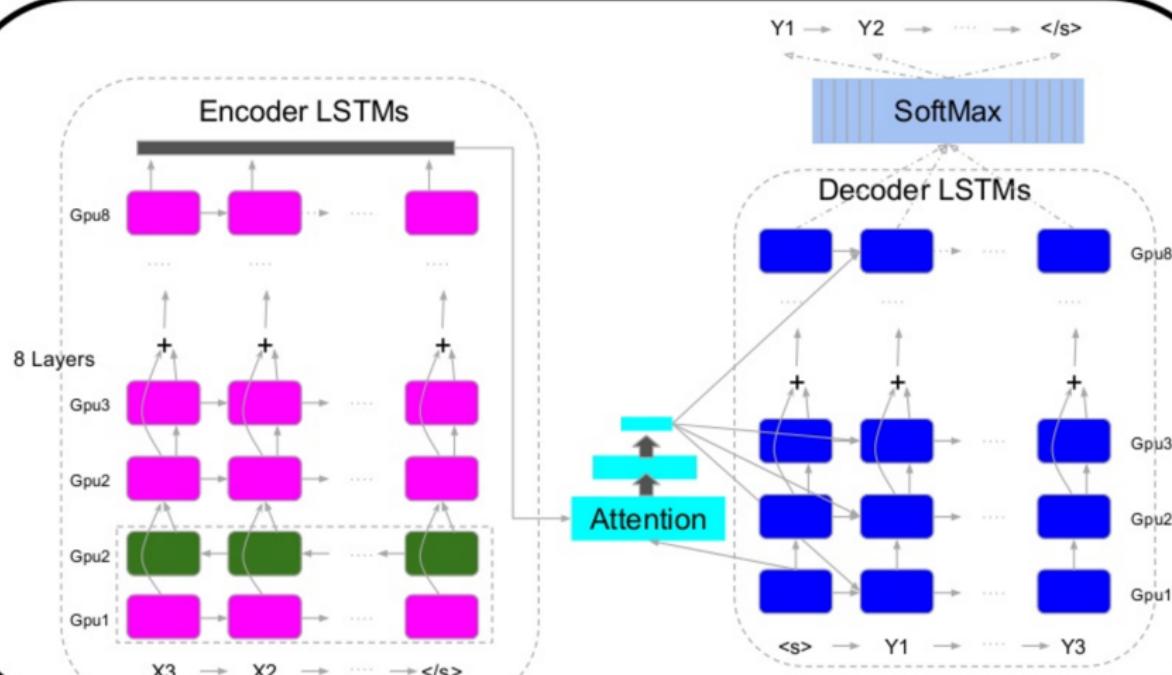


V5: Parallelization

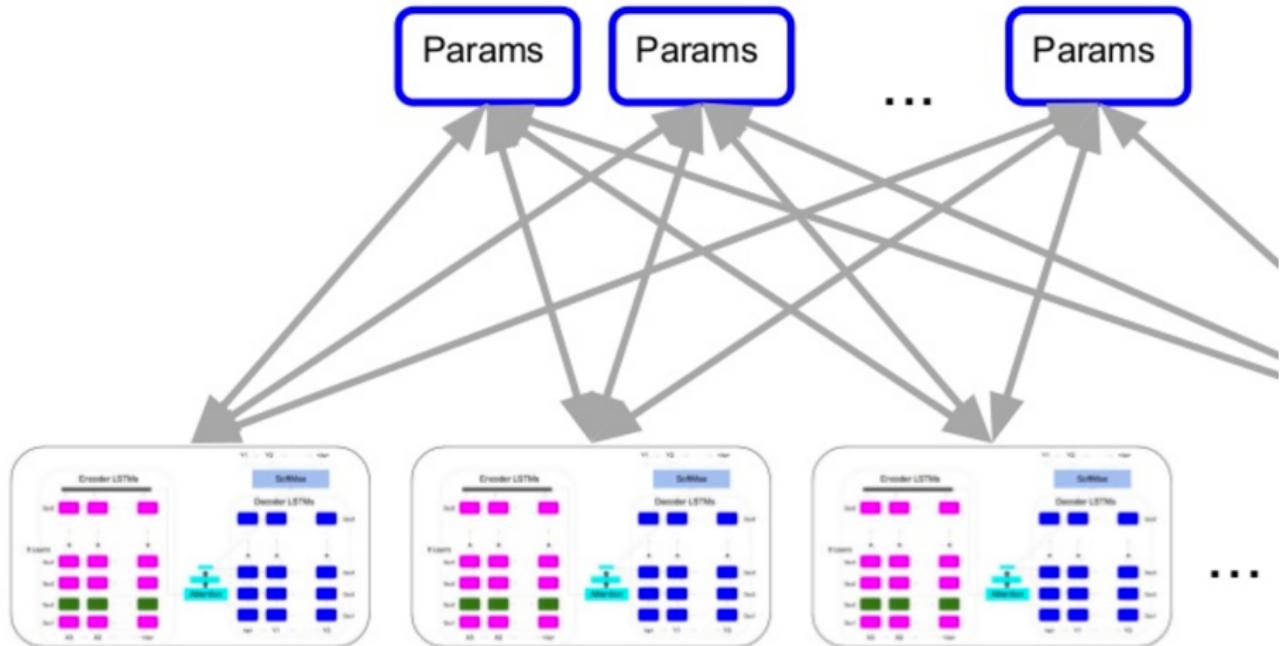
- Each layer trained by a GPU
- Forward pass:
 - Encoder: 7 uni-directional encoder layers trained in a pipeline
 - Decoder: pipeline starts as soon as encoder layers are ready
- Backward pass:
 - Teacher forcing



Model Parallelism (1 Machine, 8 GPUs)

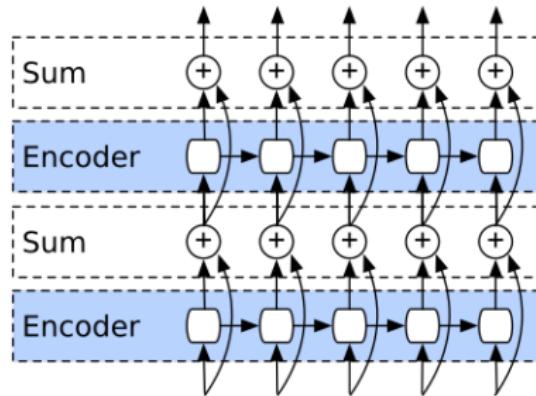


Data Parallelism (Multiple Param Servers)



V6: Residuals

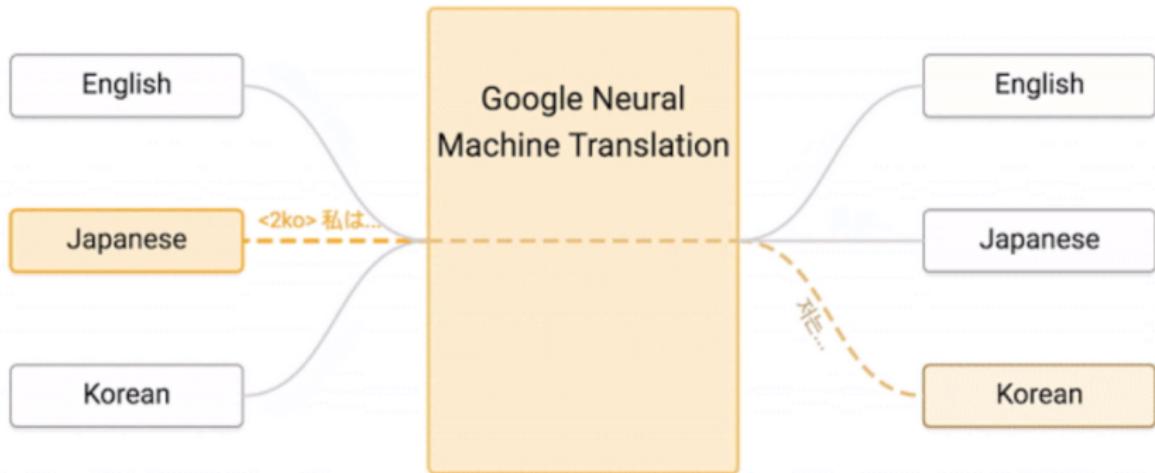
- Upper layer learns the *delta* function to the lower one
- Easier to train a deep NN



V7: Multilingual & Zero-Shot Translation

“Google AI invented its own secret language? ”

Zero-shot



Reference I

- [1] Martin Arjovsky, Amar Shah, and Yoshua Bengio.
Unitary evolution recurrent neural networks.
arXiv preprint arXiv:1511.06464, 2015.
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.
Neural machine translation by jointly learning to align and translate.
arXiv preprint arXiv:1409.0473, 2014.
- [3] Alex Graves.
Adaptive computation time for recurrent neural networks.
arXiv preprint arXiv:1603.08983, 2016.
- [4] Alex Graves, Greg Wayne, and Ivo Danihelka.
Neural turing machines.
arXiv preprint arXiv:1410.5401, 2014.

Reference II

- [5] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom.
Learning to transduce with unbounded memory.
In *Advances in Neural Information Processing Systems*, pages 1828–1836, 2015.
- [6] Armand Joulin and Tomas Mikolov.
Inferring algorithmic patterns with stack-augmented recurrent nets.
In *Advances in Neural Information Processing Systems*, pages 190–198, 2015.
- [7] Andrej Karpathy, Justin Johnson, and Li Fei-Fei.
Visualizing and understanding recurrent networks.
arXiv preprint arXiv:1506.02078, 2015.
- [8] David Krueger and Roland Memisevic.
Regularizing rnns by stabilizing activations.
arXiv preprint arXiv:1511.08400, 2015.

Reference III

- [9] Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever.
Neural random-access machines.
arXiv preprint arXiv:1511.06392, 2015.
- [10] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton.
A simple way to initialize recurrent networks of rectified linear units.
arXiv preprint arXiv:1504.00941, 2015.
- [11] Scott Reed and Nando de Freitas.
Neural programmer-interpreters.
arXiv preprint arXiv:1511.06279, 2015.
- [12] Eduardo D Sontag.
On the computational power of neural nets'.
JOURNAL OF COMPUTER AND SYSTEM SCIENCES, 50:132–150,
1995.

Reference IV

- [13] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al.
End-to-end memory networks.
In *Advances in neural information processing systems*, pages 2440–2448, 2015.
- [14] Jason Weston, Sumit Chopra, and Antoine Bordes.
Memory networks.
arXiv preprint arXiv:1410.3916, 2014.
- [15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio.
Show, attend and tell: Neural image caption generation with visual attention.
arXiv preprint arXiv:1502.03044, 2(3):5, 2015.