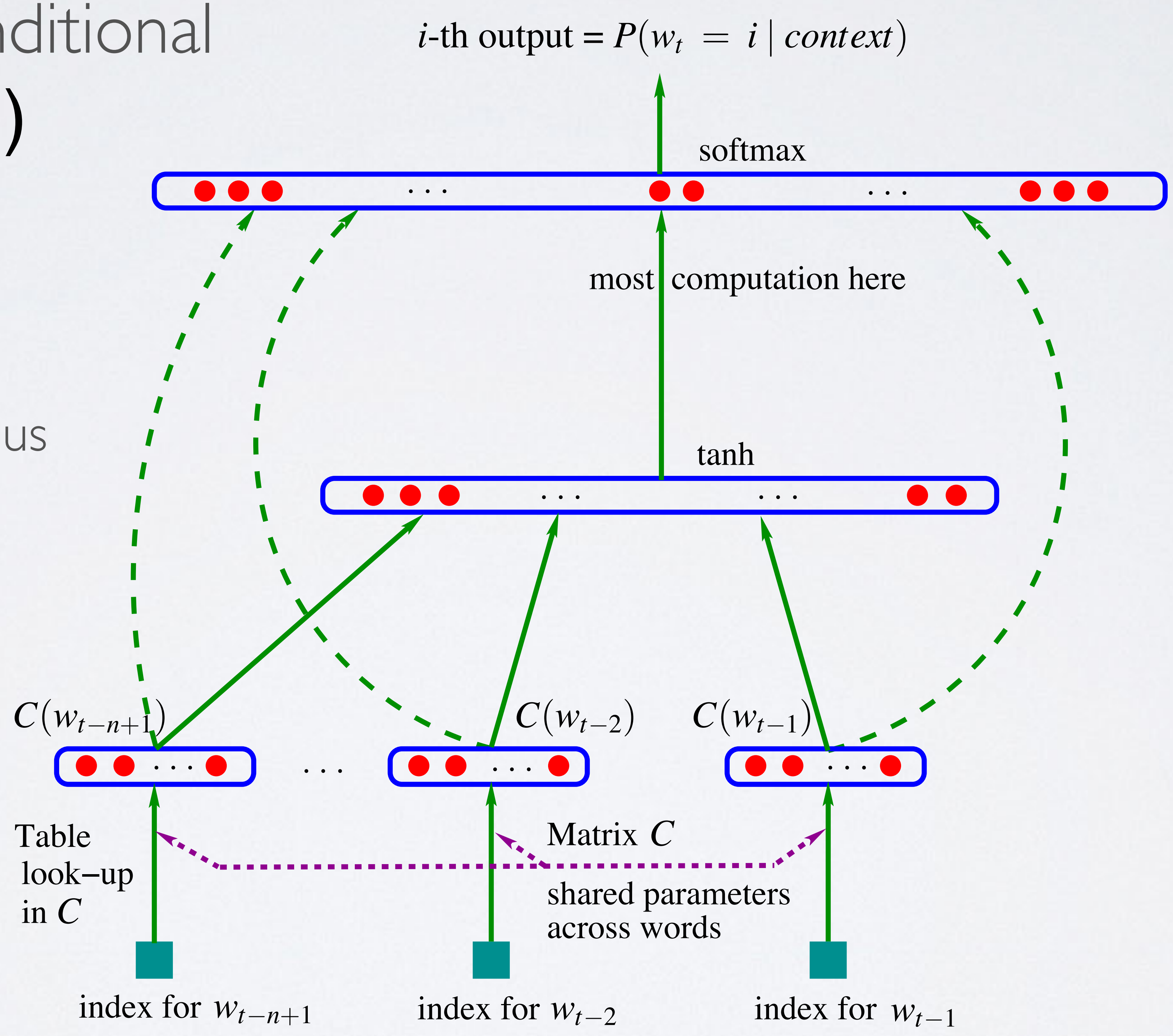
Recurrent neural networks

Slides by Hugo Larochelle - Google Brain,
lightly edited by Aaron Courville

NEURAL NETWORK LANGUAGE MODEL

Topics: neural network language model

- Solution: model the conditional $p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$ with a neural network
  - learn word representations to allow transfer to *n*-grams not observed in training corpus



Bengio, Ducharme, Vincent and Jauvin, 2003

LANGUAGE MODELING

Topics: language modeling

- An assumption frequently made is the n^{th} order Markov assumption

$$p(w_1, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$$

  - the t^{th} word was generated based only on the $n-1$ previous words
  - we will refer to $w_{t-(n-1)}, \ldots, w_{t-1}$ as the context

LANGUAGE MODELING

Topics: language modeling

- An assumption frequently made is the n^{th} order Markov assumption

$$p(w_1, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$$

  - the t^{th} word was generated based only on the $n-1$ previous words
  - we will refer to $w_{t-(n-1)}, \ldots, w_{t-1}$ as the context

Could we have a neural network that depends on the full previous context, i.e. that would model?

$$p(w_1, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_1, \ldots, w_{t-1})$$

RECURRENT NEURAL NETWORK (RNN)

Topics: RNN language model

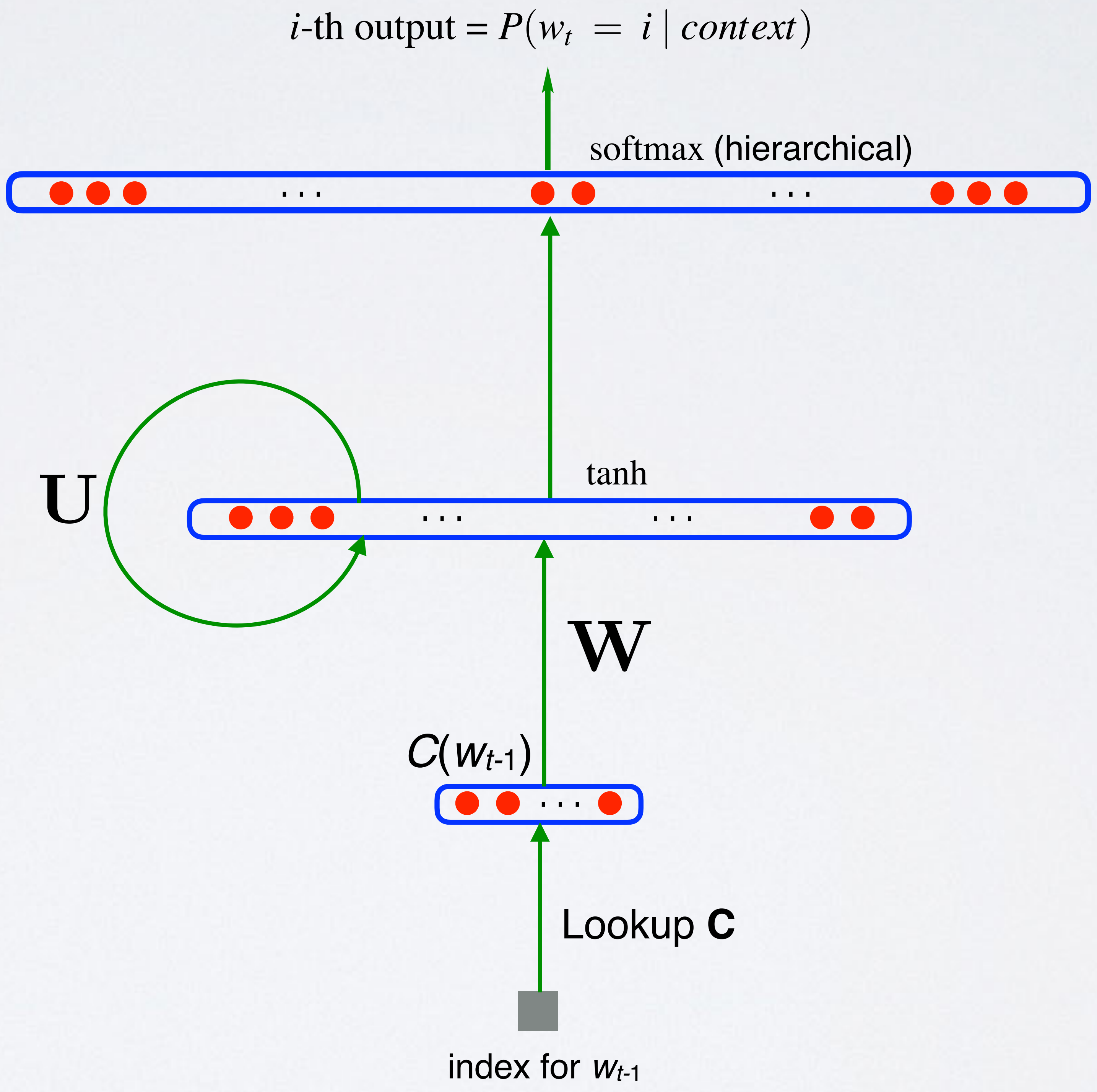
- Solution: recursively update a persistent hidden layer

$$\mathbf{h}_t = \tanh(\mathbf{b} + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}C(w_t))$$

- To compute

$$p(w_t \mid \underbrace{w_1, \ldots, w_{t-1}}_{context})$$

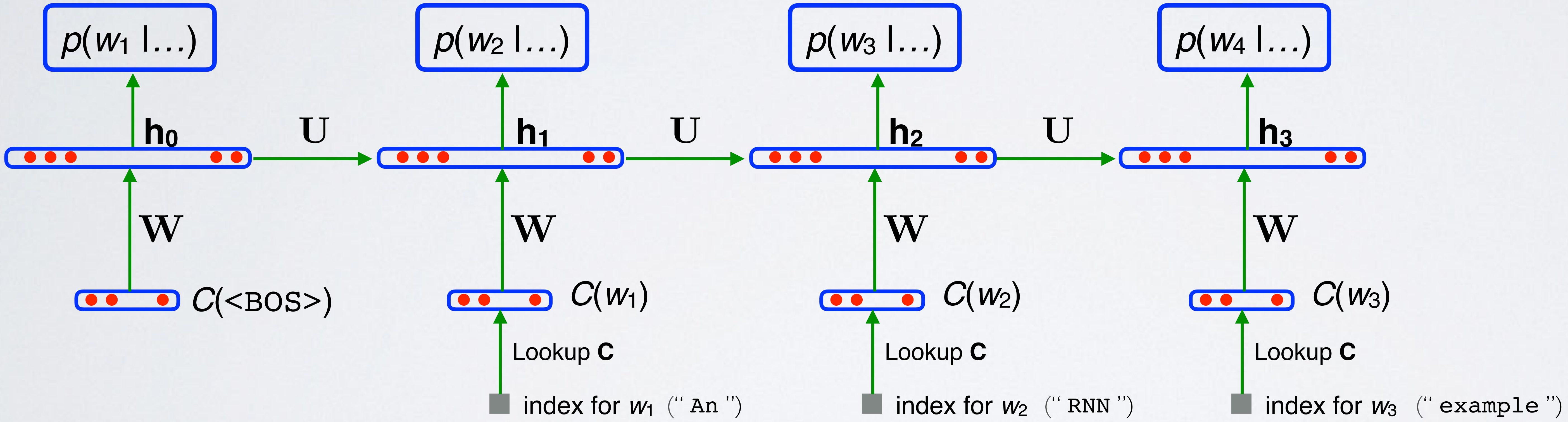
we use hidden layer $\mathbf{h}_{t\text{-}1}$



RECURRENT NEURAL NETWORK (RNN)

Topics: unrolled RNN

- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)

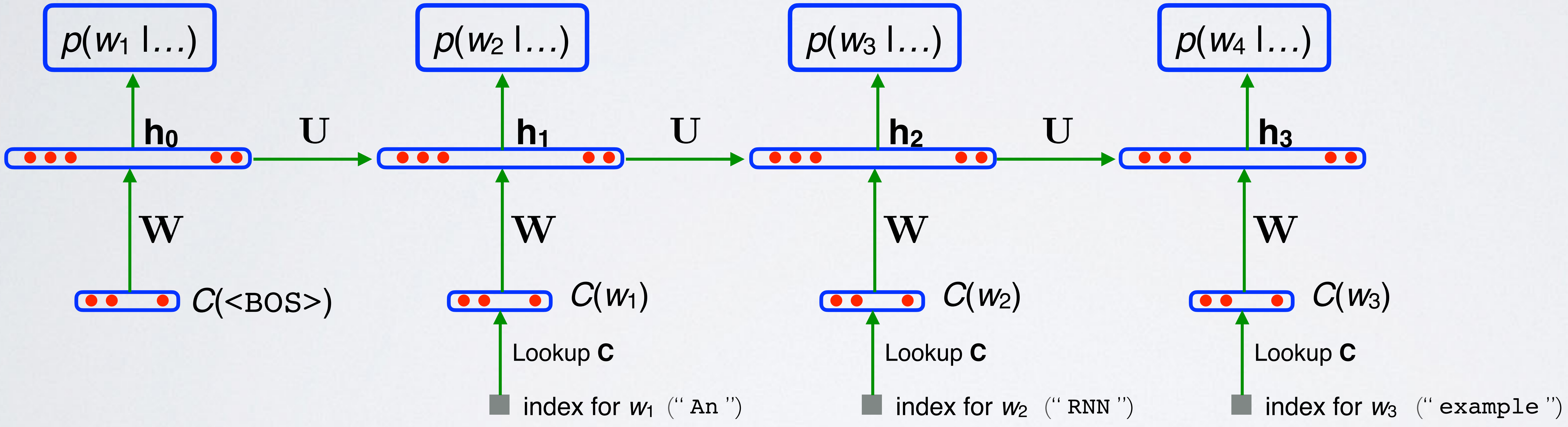


  - symbol " `.` " serves as an *end of sentence* symbol
  - $\mathbf{h}_0 = \tanh(\mathbf{b} + \mathbf{W}C(\text{<BOS>}))$, where $C(\text{<BOS>})$ is a unique embedding for the *beginning of sentence* position(<BOS> not included as possible output!)

DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

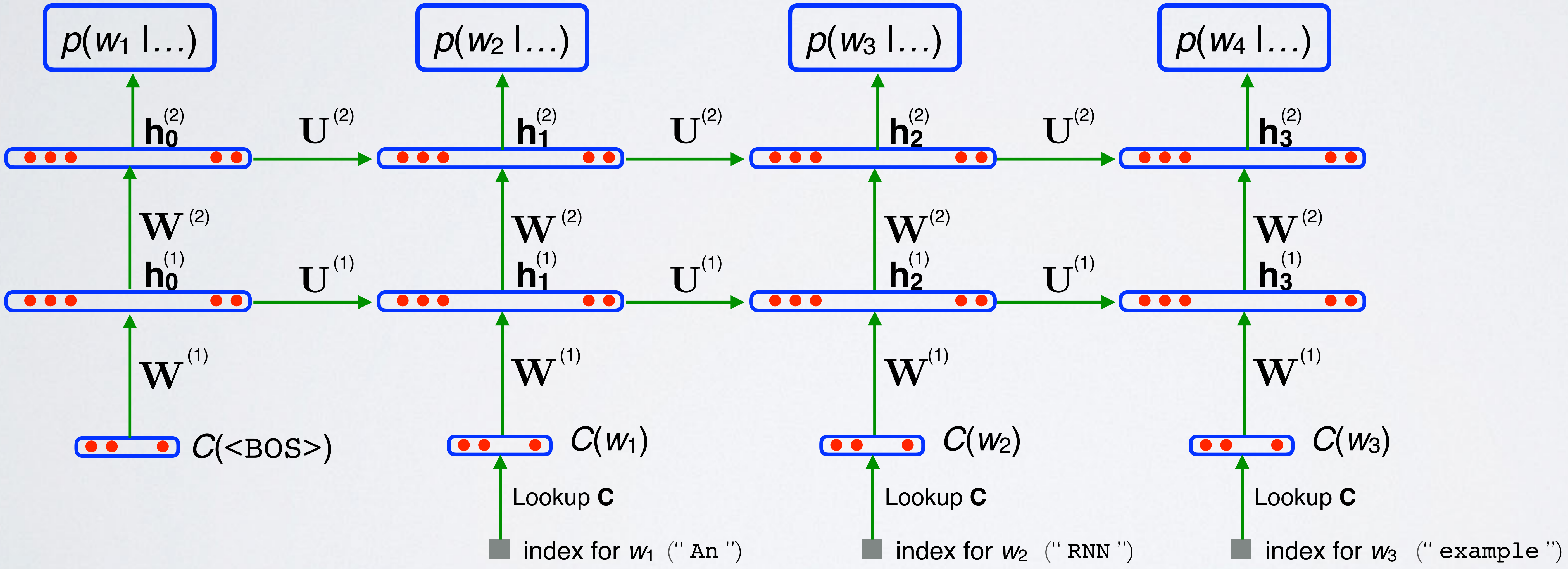
- Straightforward to make deep
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)



DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

- Straightforward to make deep
  - example: $\mathbf{w}$ = [“ `An` ”, “ `RNN` ”, “ `example` ”, “ `.` ”] ($T = 4$)



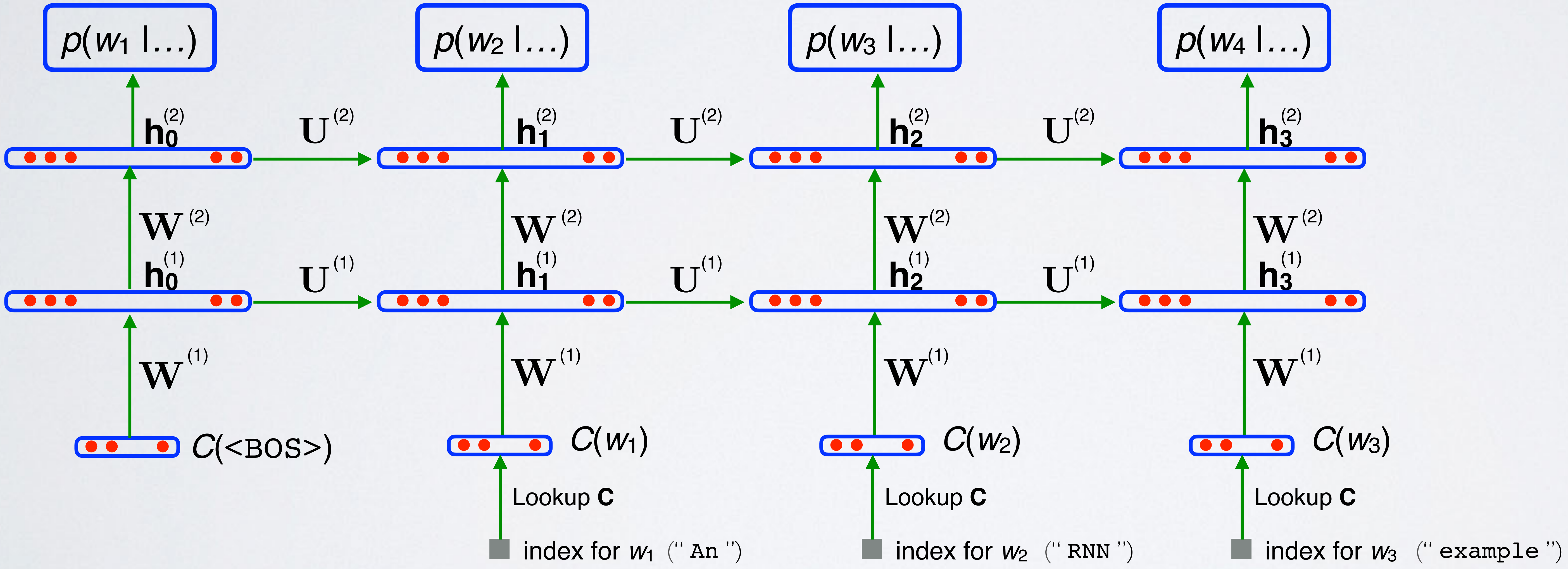
DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

- Straightforward to make deep
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)

$$\mathbf{h}_t^{(1)} = \tanh(\mathbf{b}^{(1)} + \mathbf{U}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{W}^{(1)}C(w_t))$$

$$\mathbf{h}_t^{(2)} = \tanh(\mathbf{b}^{(2)} + \mathbf{U}^{(2)}\mathbf{h}_{t-1}^{(2)} + \mathbf{W}^{(2)}\mathbf{h}_t^{(1)})$$

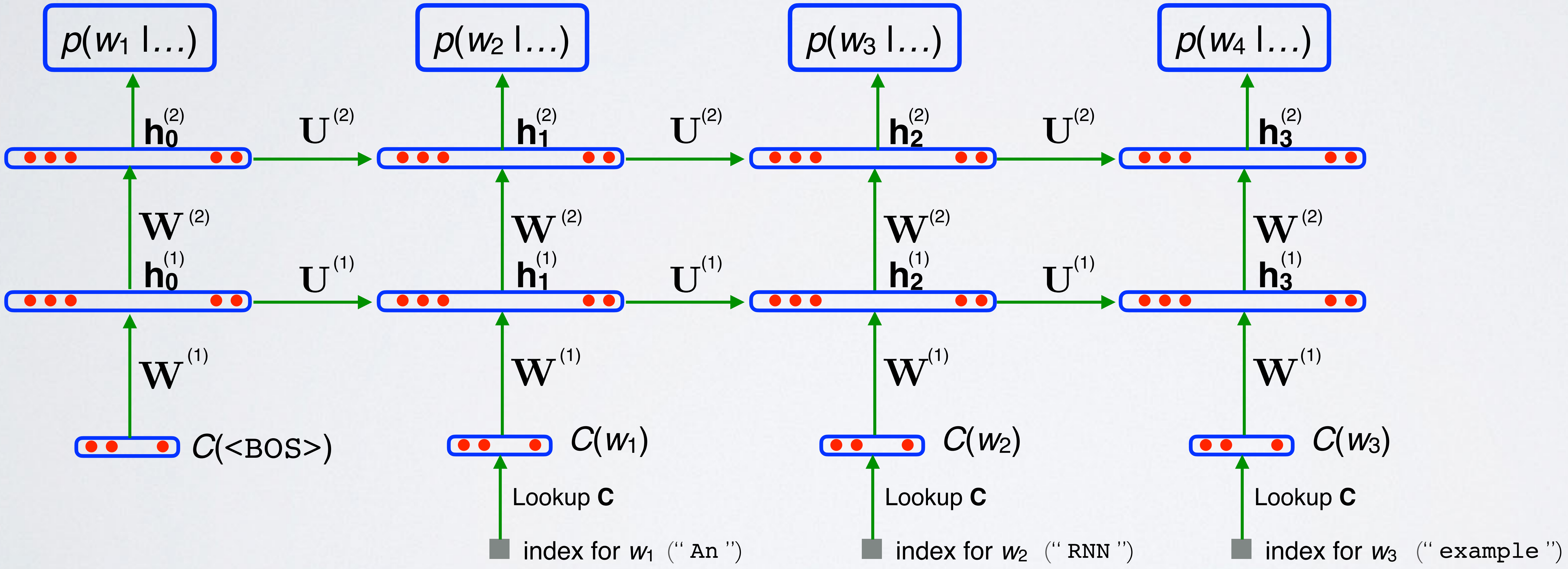


DEEP RECURRENT NEURAL NETWORK

$$\mathbf{h}_t^{(1)} = \tanh(\mathbf{b}^{(1)} + \mathbf{U}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{W}^{(1)}C(w_t))$$
$$\mathbf{h}_t^{(2)} = \tanh(\mathbf{b}^{(2)} + \mathbf{U}^{(2)}\mathbf{h}_{t-1}^{(2)} + \mathbf{W}^{(2)}\mathbf{h}_t^{(1)})$$

Topics: Deep RNN

- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)

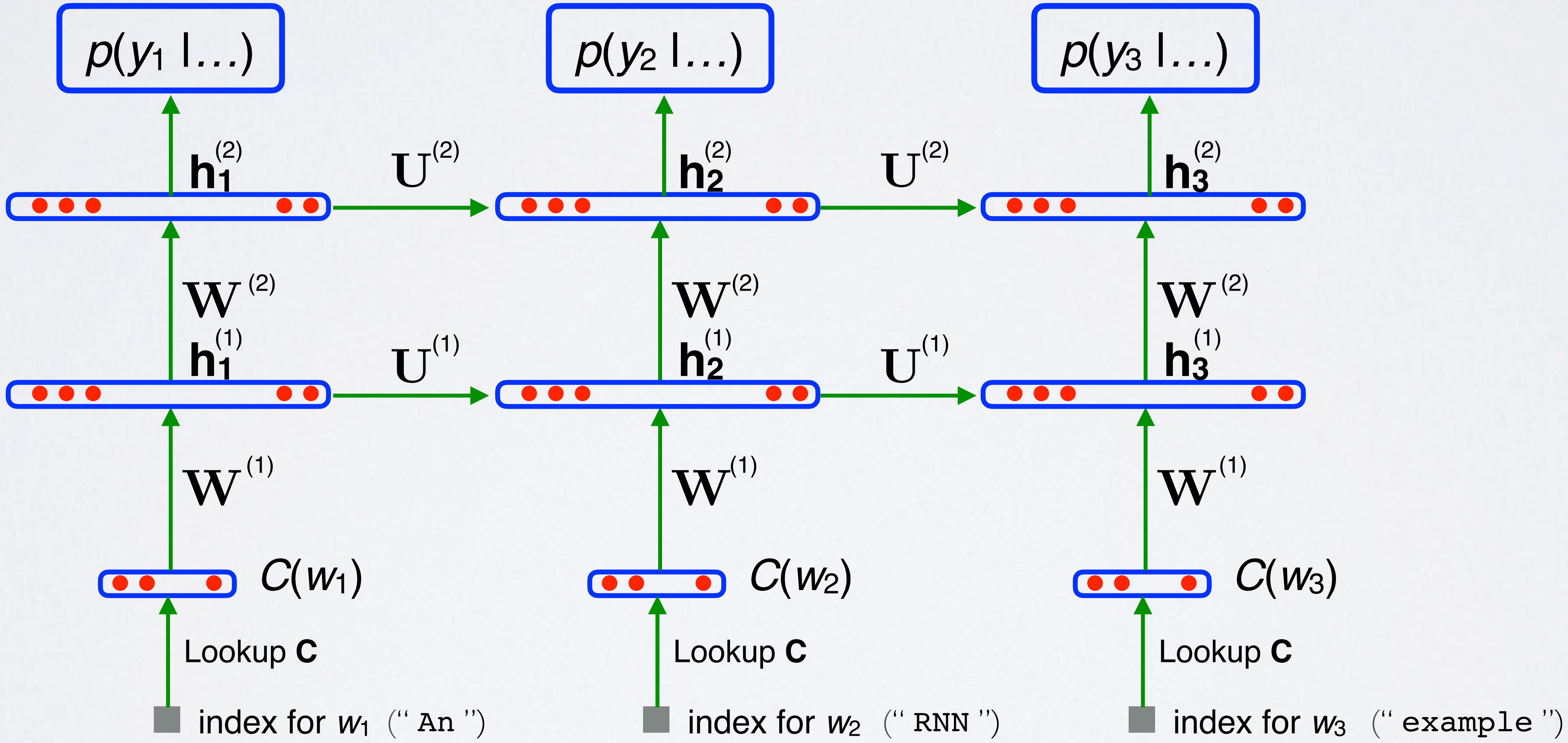


DEEP RECURRENT NEURAL NETWORK

$$\mathbf{h}_t^{(1)} = \tanh(\mathbf{b}^{(1)} + \mathbf{U}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{W}^{(1)}C(w_t))$$
$$\mathbf{h}_t^{(2)} = \tanh(\mathbf{b}^{(2)} + \mathbf{U}^{(2)}\mathbf{h}_{t-1}^{(2)} + \mathbf{W}^{(2)}\mathbf{h}_t^{(1)})$$

Topics: Deep RNN

- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)

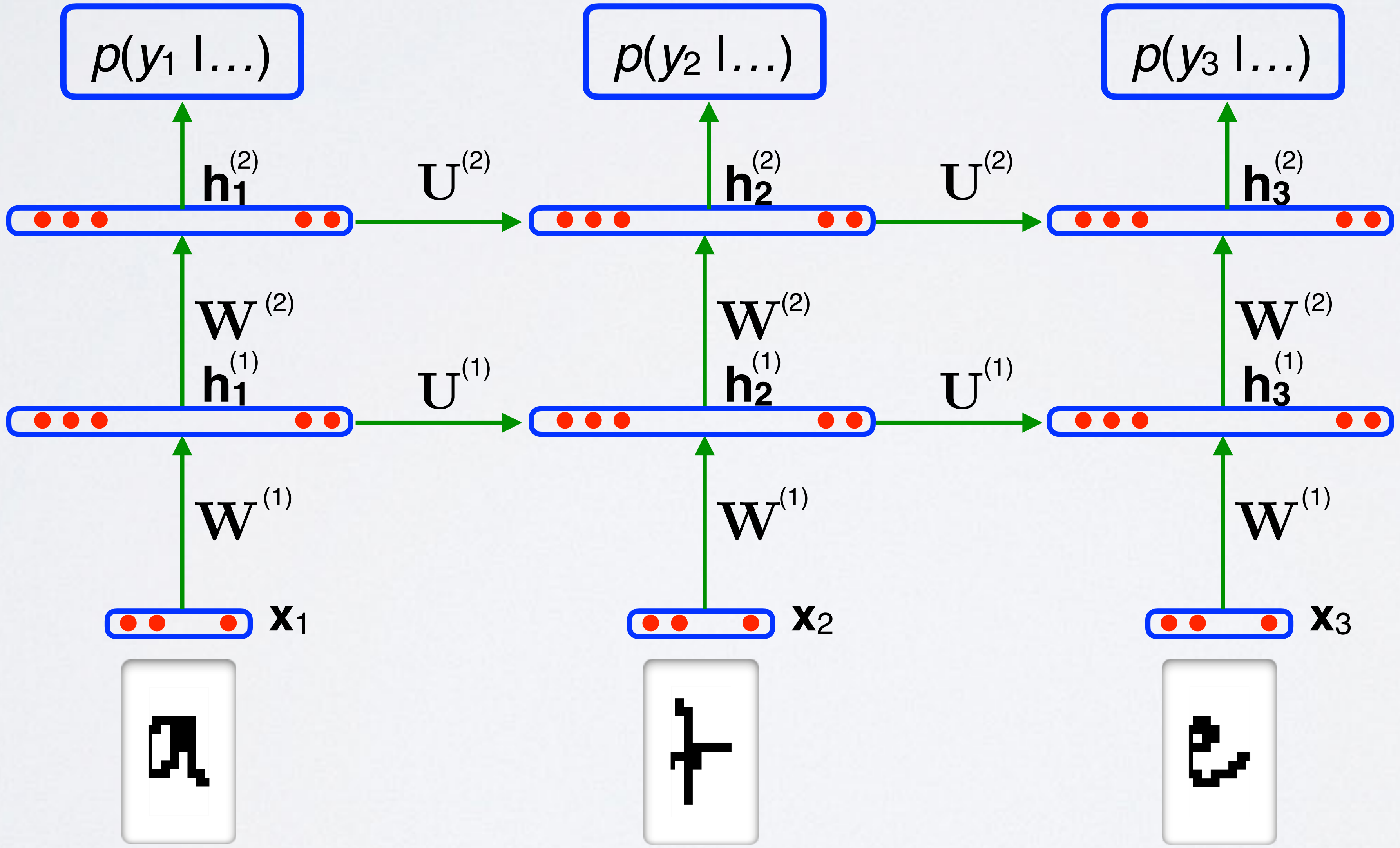


DEEP RECURRENT NEURAL NETWORK

$$\mathbf{h}_t^{(1)} = \tanh(\mathbf{b}^{(1)} + \mathbf{U}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{W}^{(1)}C(w_t))$$
$$\mathbf{h}_t^{(2)} = \tanh(\mathbf{b}^{(2)} + \mathbf{U}^{(2)}\mathbf{h}_{t-1}^{(2)} + \mathbf{W}^{(2)}\mathbf{h}_t^{(1)})$$

Topics: Deep RNN

- Useful beyond language modeling
  - sequence labeling in general (e.g. character recognition)



Recurrent neural networks

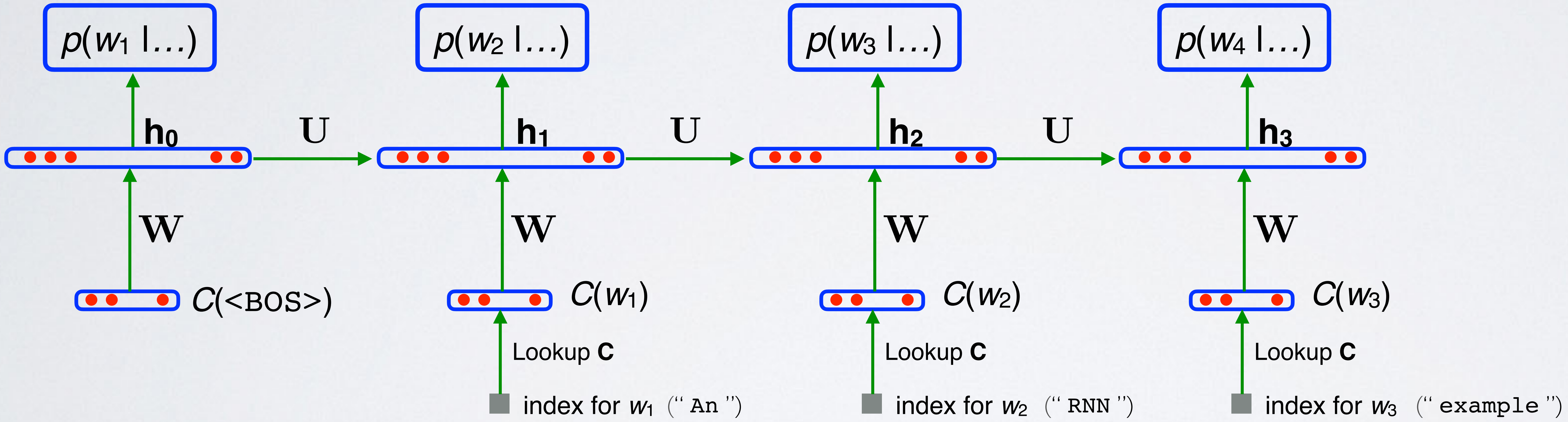
Backpropagation through time

RECURRENT NEURAL NETWORK (RNN)

REMINDER

Topics: unrolled RNN

- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)

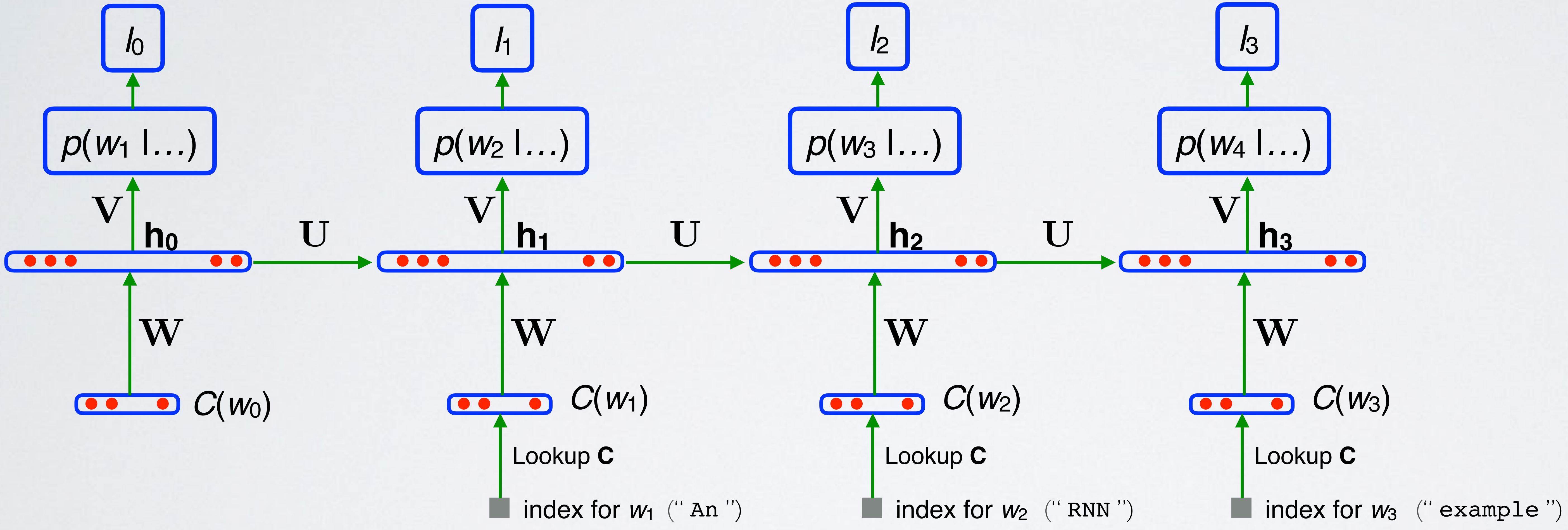


  - symbol " `.` " serves as an *end of sentence* symbol
  - alternative model for $\mathbf{h_0}$ is to set w_0 to a unique *beginning of sentence* symbol, with its own embedding $C(w_0)$ (but not included as possible output!)

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

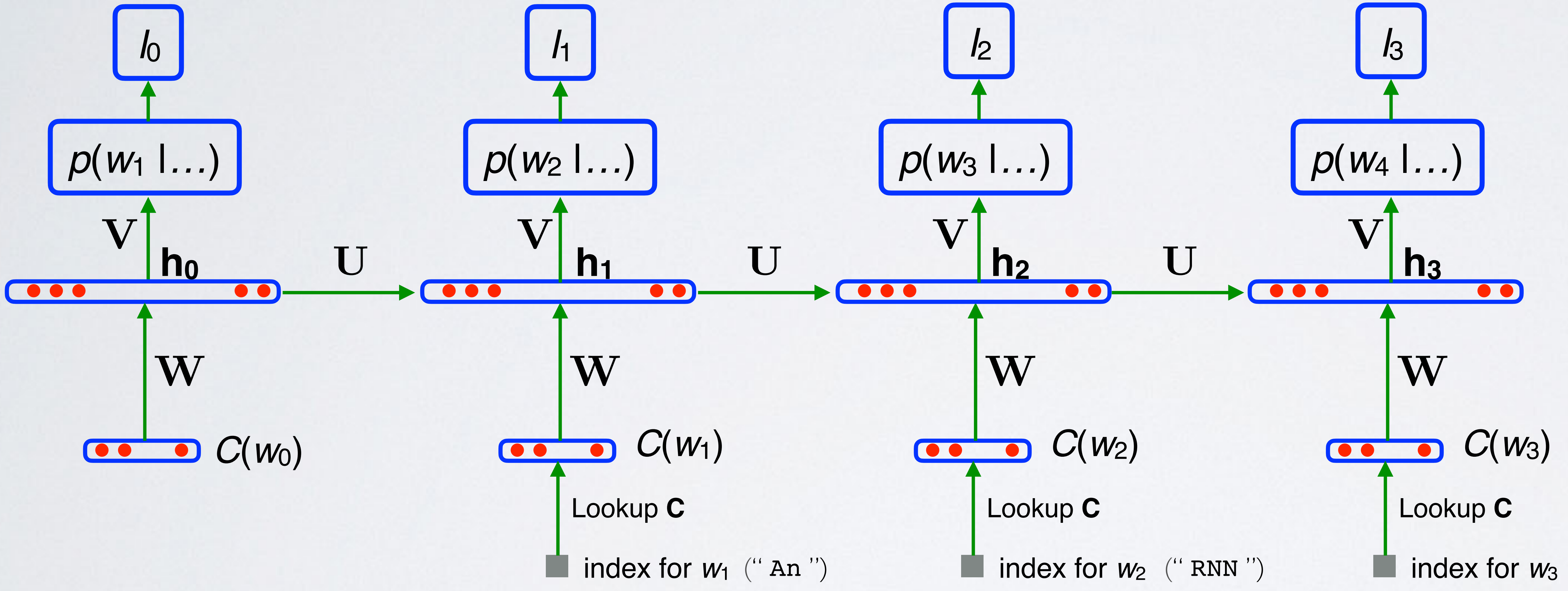


  - want to minimize sum of per step loss $l = \sum_{t=0}^{T-1} l_t$
  - for language modeling, $l_t = -\log p(w_{t+1} \mid \ldots)$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

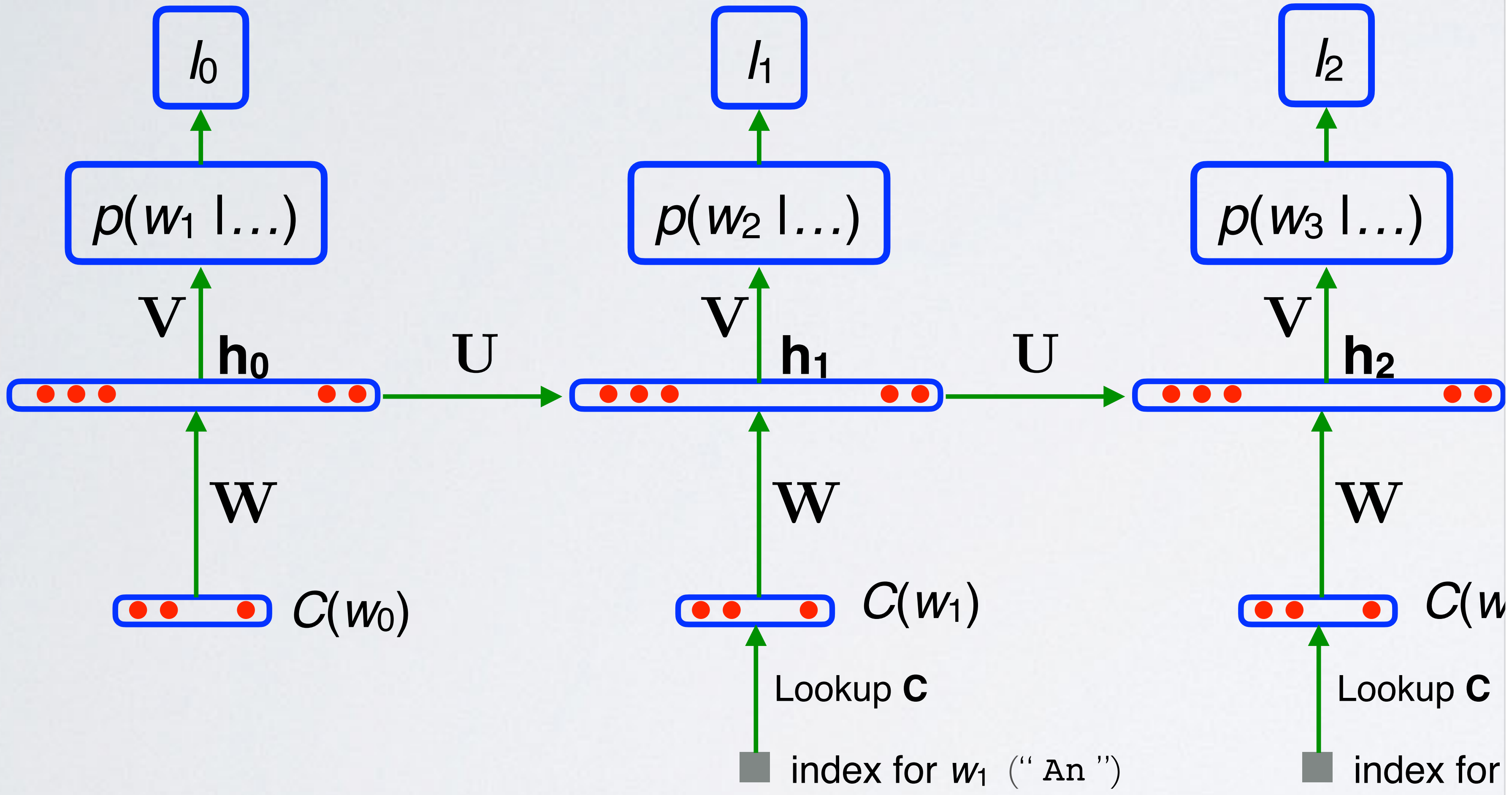


  - ***forward propagation:*** computation follows arrows in flow graph (forward in time)

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule



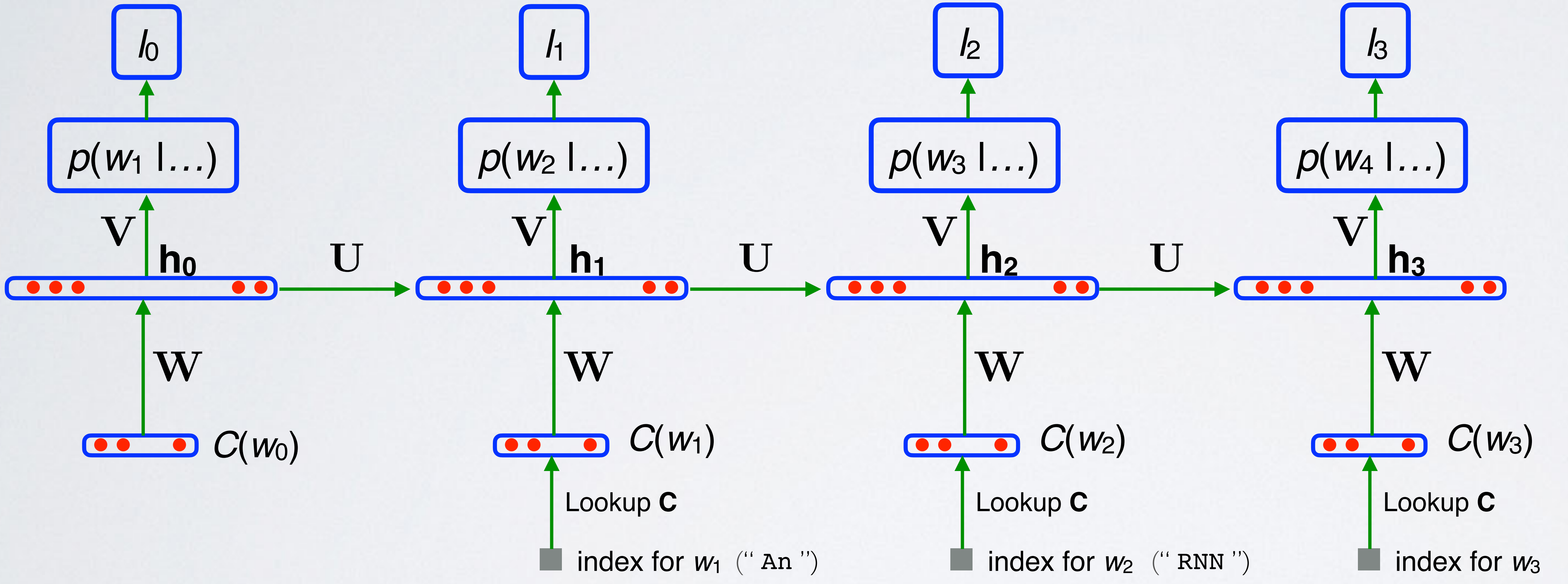
- initialize gradients
  - $\nabla_{\mathbf{V}} l \Leftarrow 0, \nabla_{\mathbf{W}} l \Leftarrow 0, \nabla_{\mathbf{U}} l \Leftarrow 0$
  - $\nabla_{\mathbf{h}_{T-1}} l \Leftarrow 0$
- for ***t*** from ***T*-1** to **0**
  - $\nabla_{\mathbf{V}} l \mathrel{+}= \nabla_{\mathbf{V}} l_t$
  - $\nabla_{\mathbf{h}_t} l \mathrel{+}= \nabla_{\mathbf{h}_t} l_t$
  - $\nabla_{\mathbf{a}_t} l \Leftarrow (1 - \mathbf{h}_t^2) \odot \nabla_{\mathbf{h}_t} l$
  - $\nabla_{\mathbf{W}} l \mathrel{+}= (\nabla_{\mathbf{a}_t} l)\; C(w_t)^\top$
  - $\nabla_{\mathbf{U}} l \mathrel{+}= (\nabla_{\mathbf{a}_t} l)\; \mathbf{h}_{t-1}^\top$
  - $\nabla_{\mathbf{h}_{t-1}} l \Leftarrow \mathbf{U}^\top \nabla_{\mathbf{a}_t} l$

- ***forward propagation:*** computation follows arrows in flow graph (forward in time)

TRUNCATER BPTT

Topics: truncated BPTT

- Inappropriate for very long or infinite (e.g. online) sequences



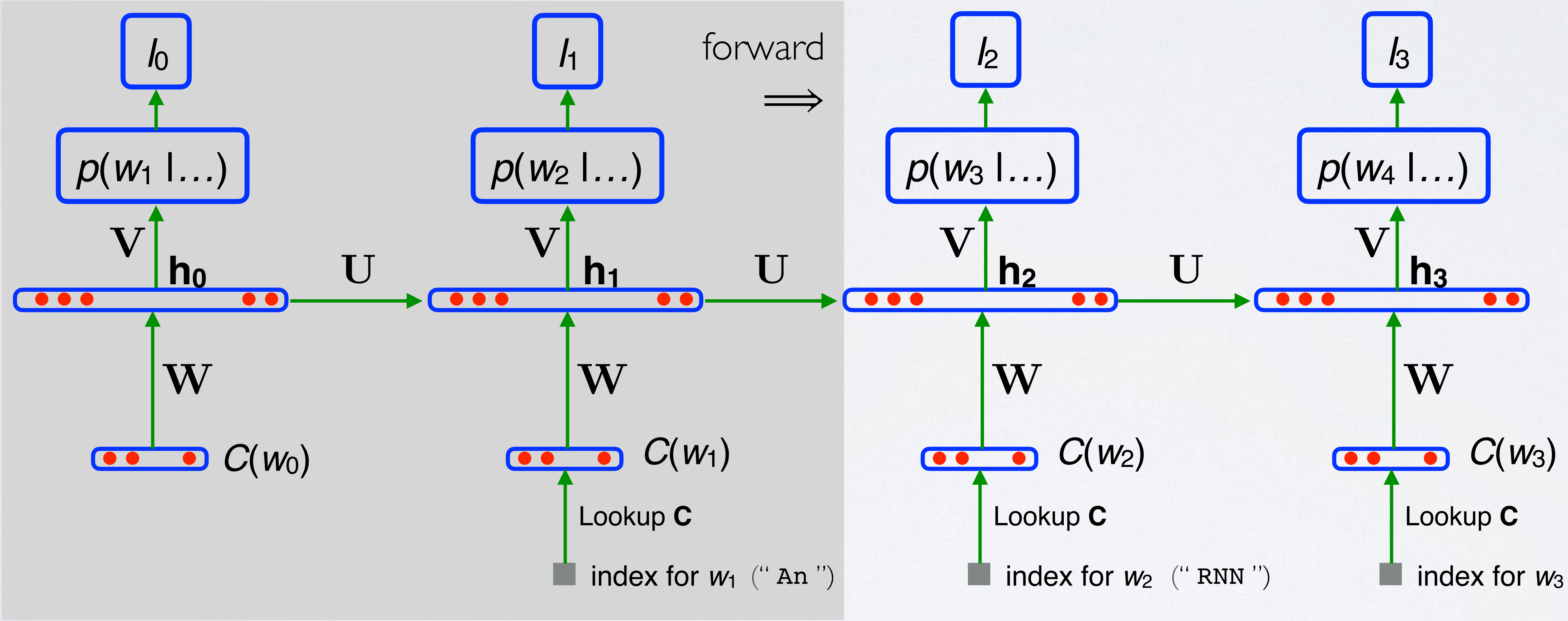
- **Truncated BPTT:** approximate BPTT by
  - performing forward pass k_1 steps at a time
  - running BPTT only k_2 steps backward and update (assuming earlier steps are fixed)

TRUNCATER BPTT

Topics: truncated BPTT

- Inappropriate for very long or infinite (e.g. online) sequences

Example with $k_1 = k_2 = 2$



- **Truncated BPTT:** approximate BPTT by
  - performing forward pass k_1 steps at a time
  - running BPTT only k_2 steps backward and update (assuming earlier steps are fixed)

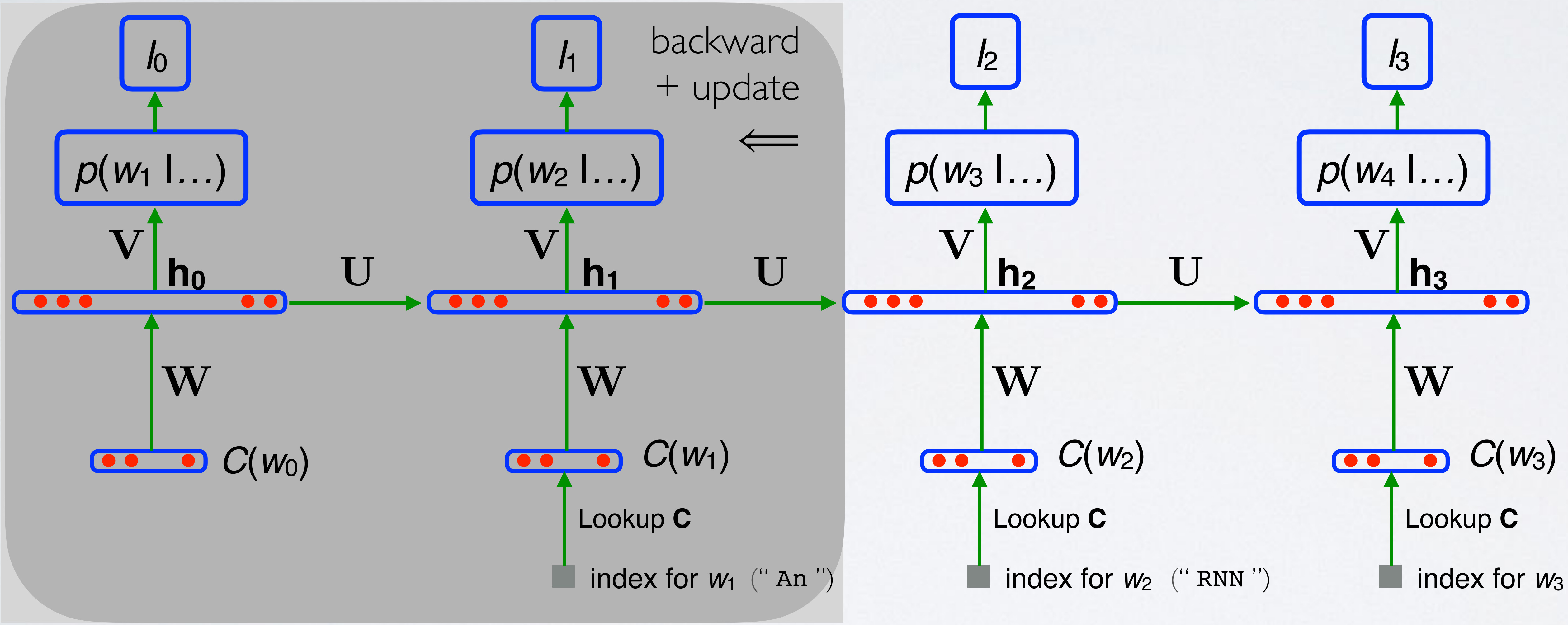
TRUNCATER BPTT

Topics: truncated BPTT

- Inappropriate for very long or infinite (e.g. online) sequences

Example with

k_1=k_2=2

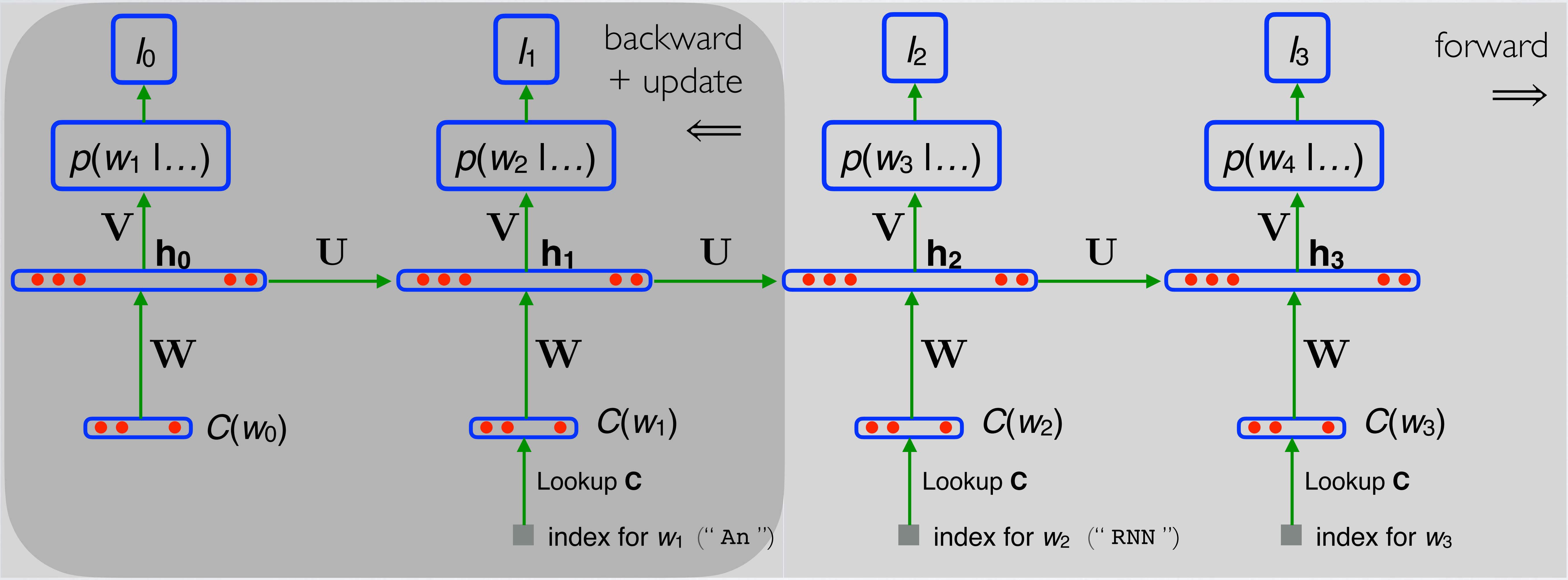


- **Truncated BPTT:** approximate BPTT by
  - performing forward pass k_1 steps at a time
  - running BPTT only k_2 steps backward and update (assuming earlier steps are fixed)

TRUNCATER BPTT

Topics: truncated BPTT

- Inappropriate for very long or infinite (e.g. online) sequences



Example with k_1=k_2=2

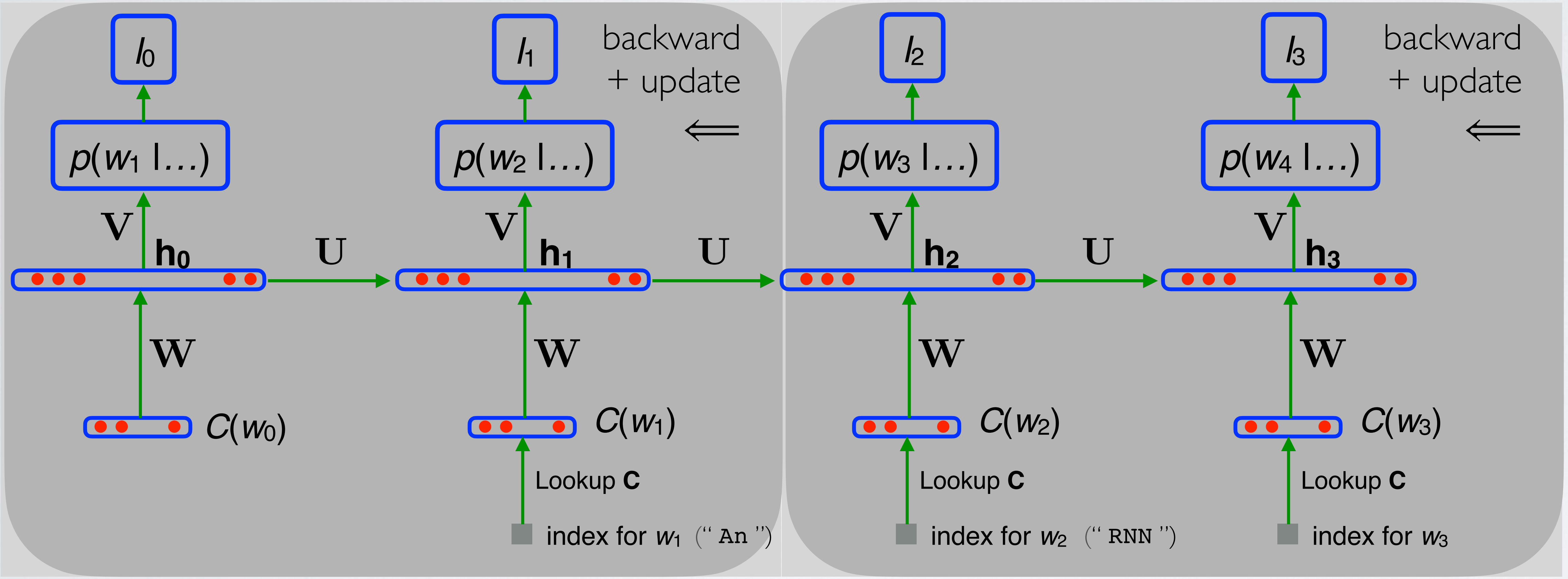
Computed from "pre-update" $\mathbf{h}_1$

- **Truncated BPTT:** approximate BPTT by
  - performing forward pass k_1 steps at a time
  - running BPTT only k_2 steps backward and update (assuming earlier steps are fixed)

TRUNCATER BPTT

Topics: truncated BPTT

- Inappropriate for very long or infinite (e.g. online) sequences



Example with k_1=k_2=2

Computed from "pre-update" $\mathbf{h}_1$

Stop BPTT at t=2

- **Truncated BPTT:** approximate BPTT by
  - performing forward pass k_1 steps at a time
  - running BPTT only k_2 steps backward and update (assuming earlier steps are fixed)

Recurrent neural networks

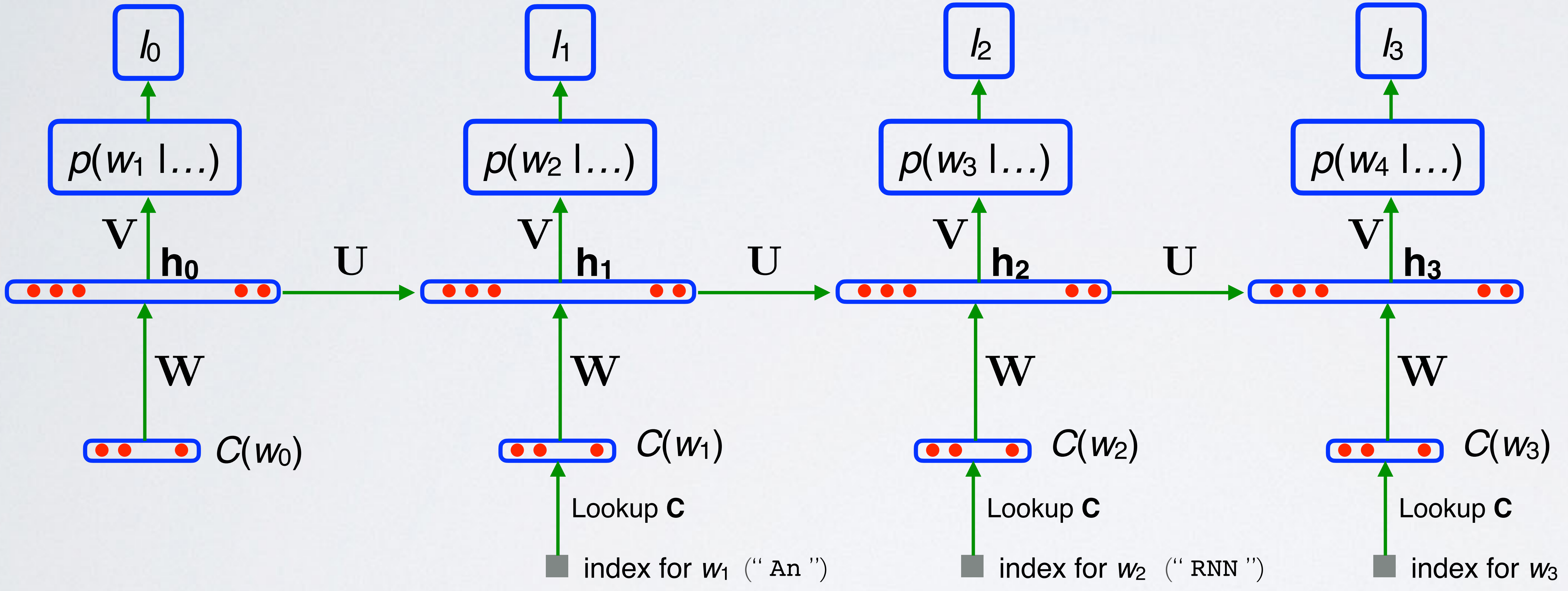
Exploding/vanishing gradient problem

BACKPROPAGATION THROUGH TIME

REMINDER

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

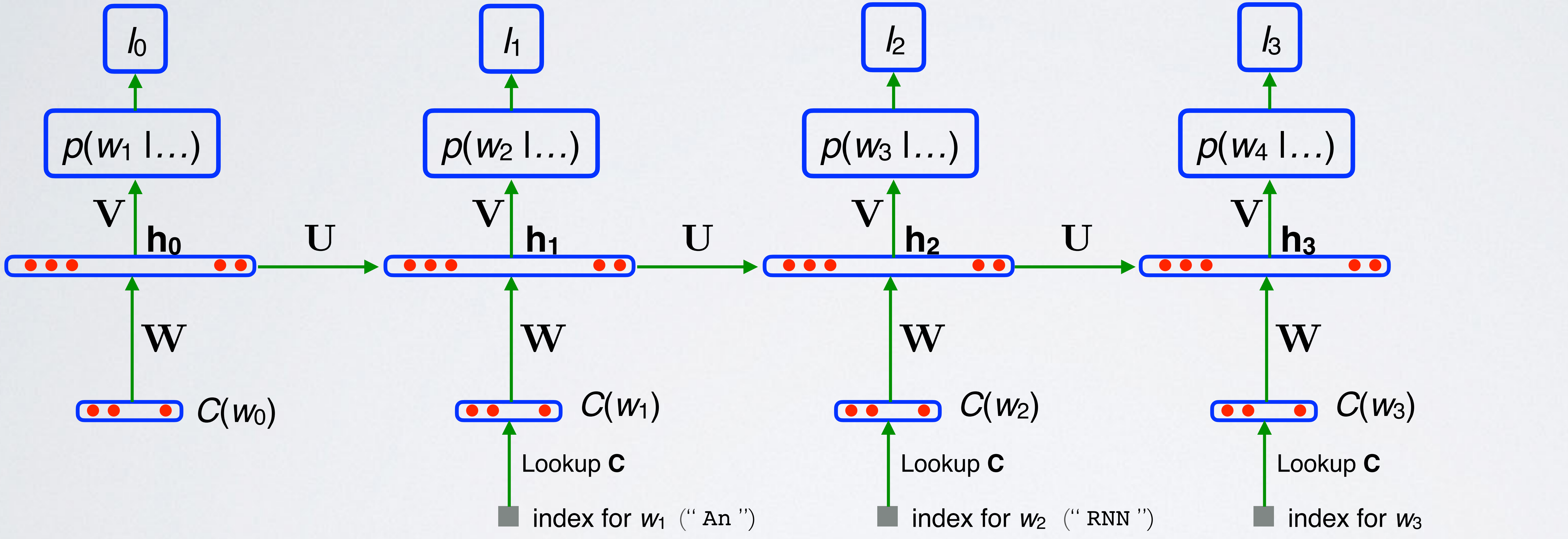


- ***forward propagation:*** computation follows arrows in flow graph (forward in time)

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

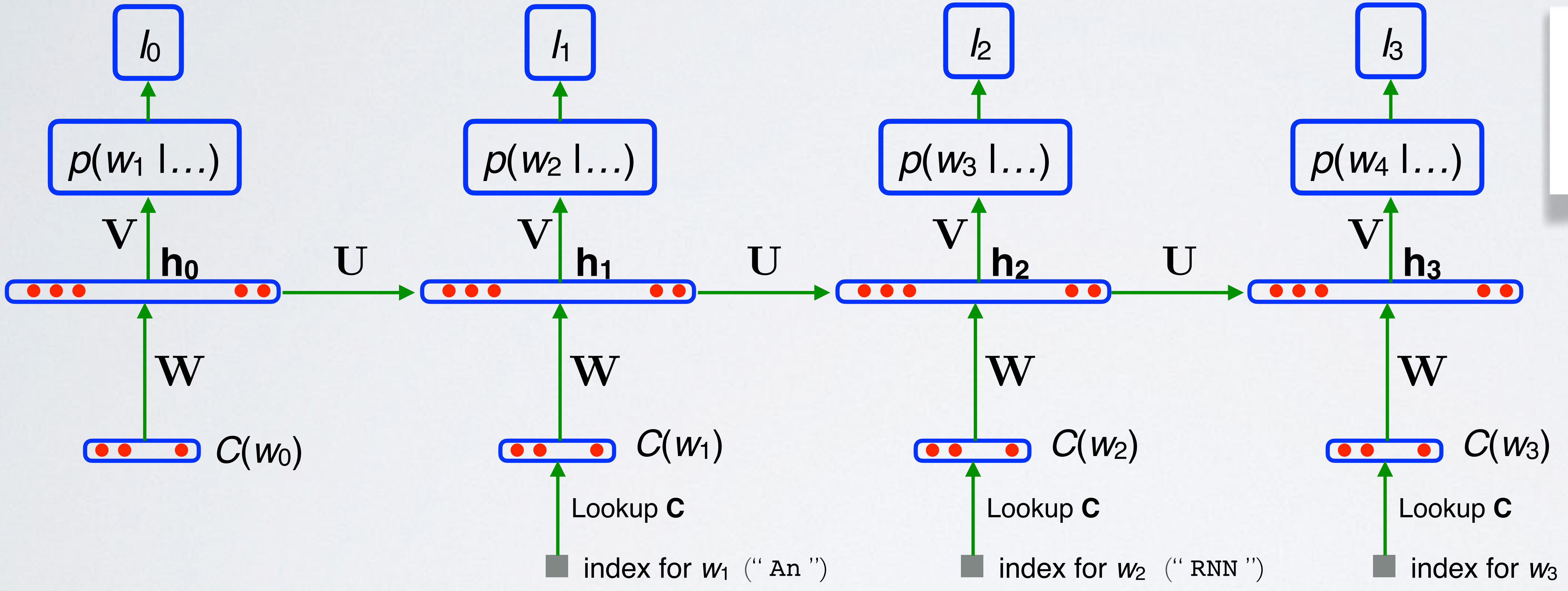


- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



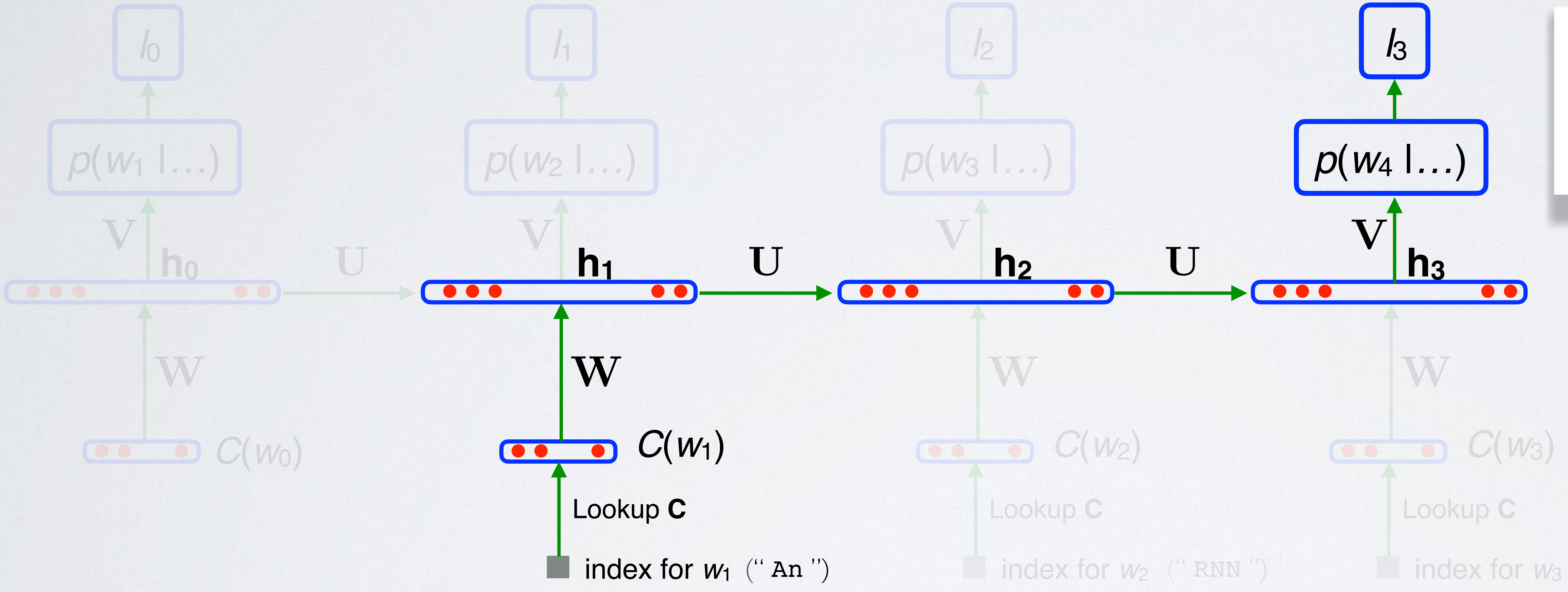
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



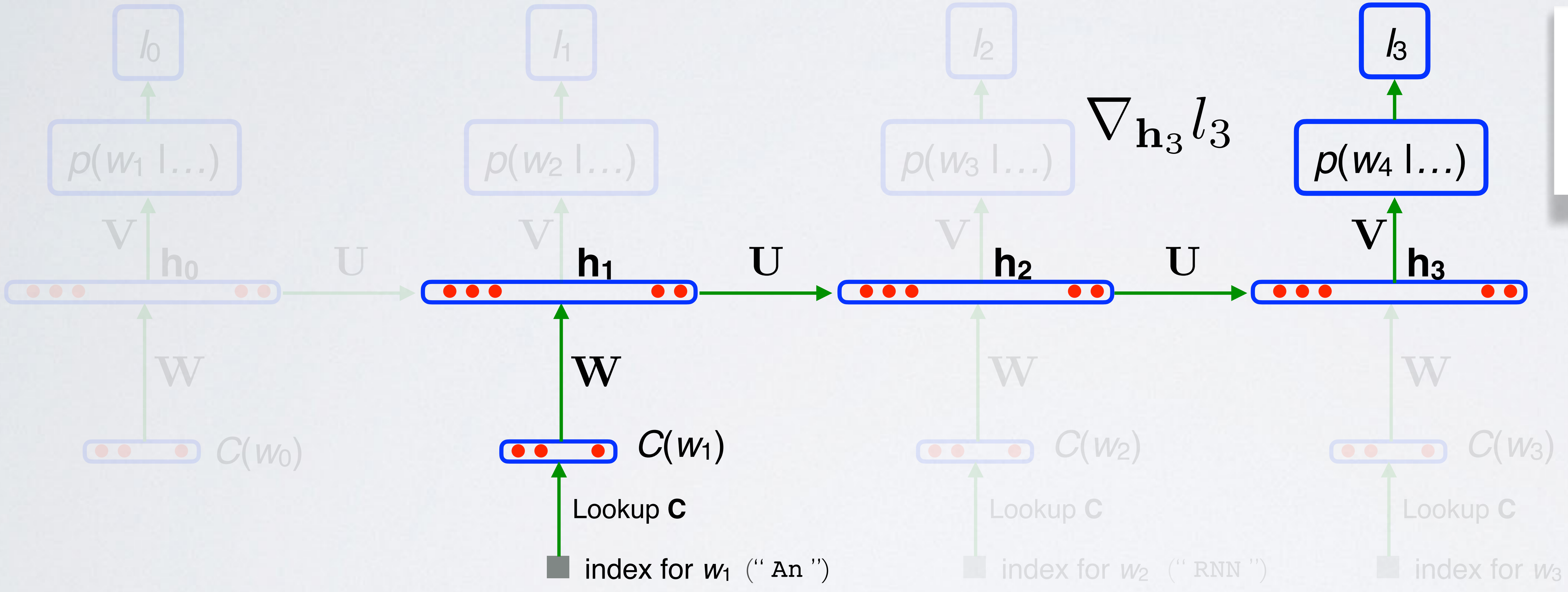
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



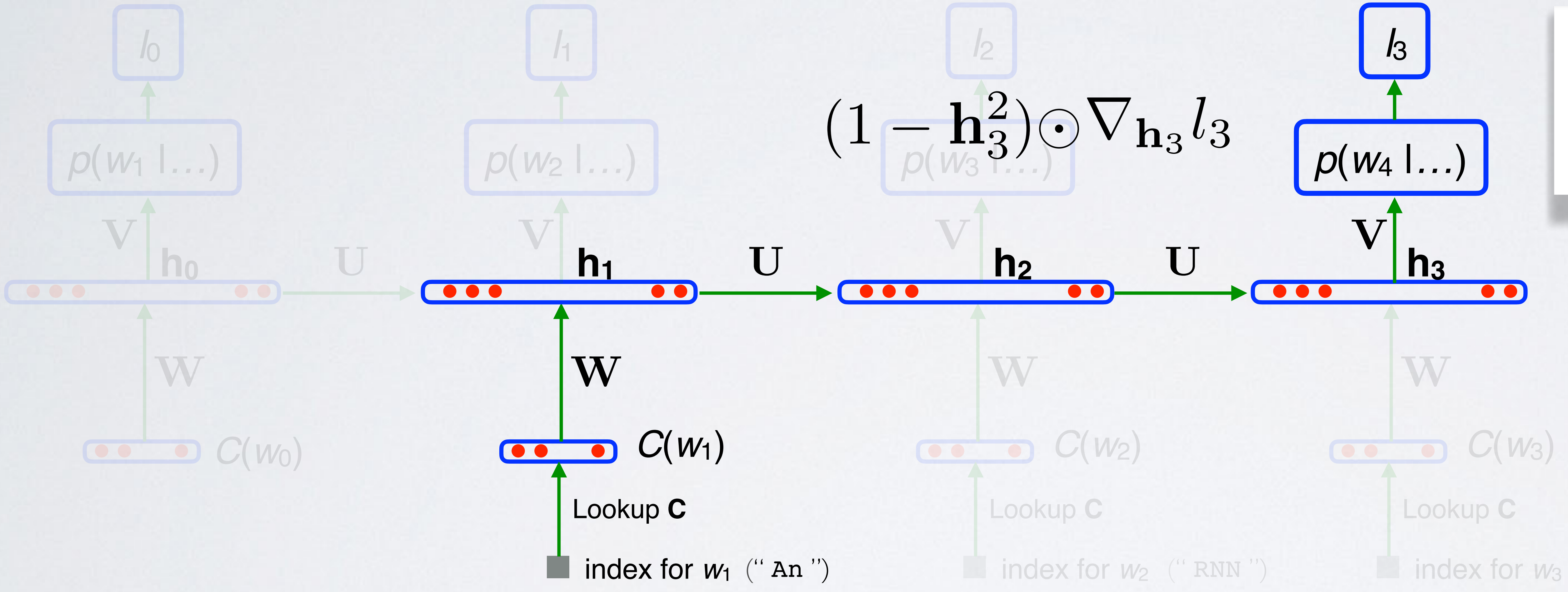
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



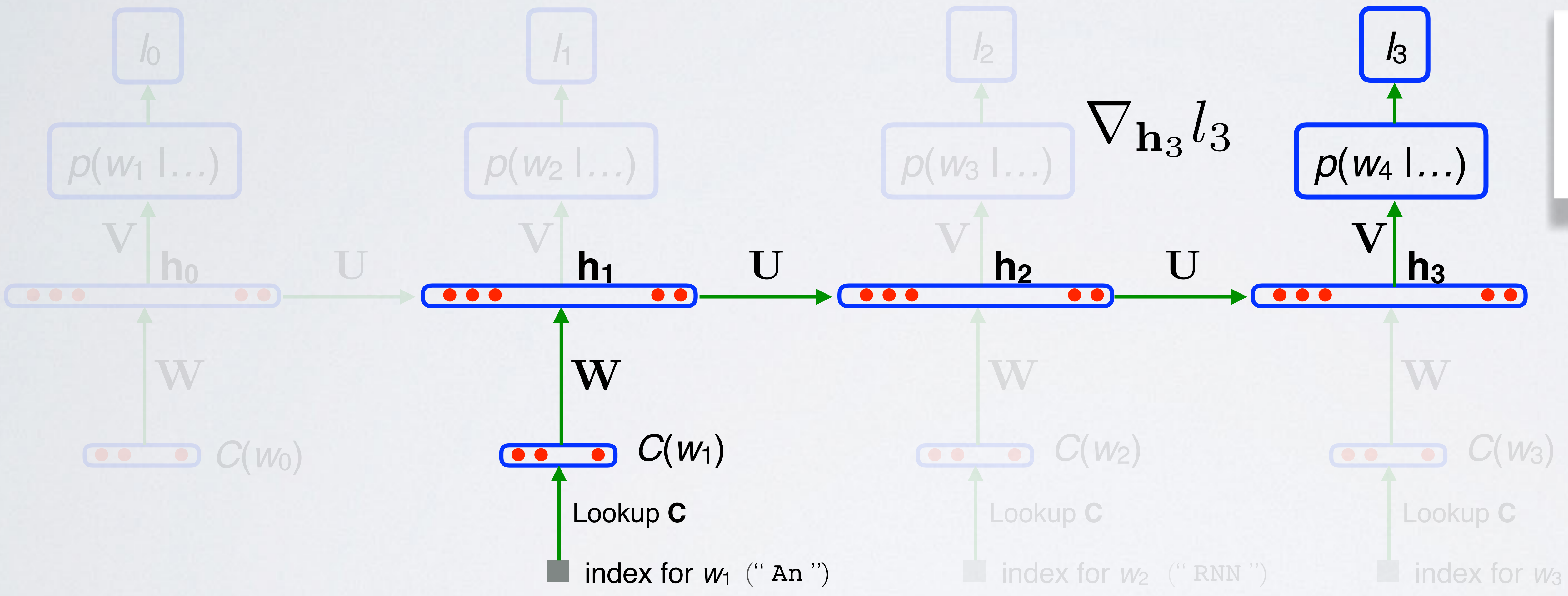
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



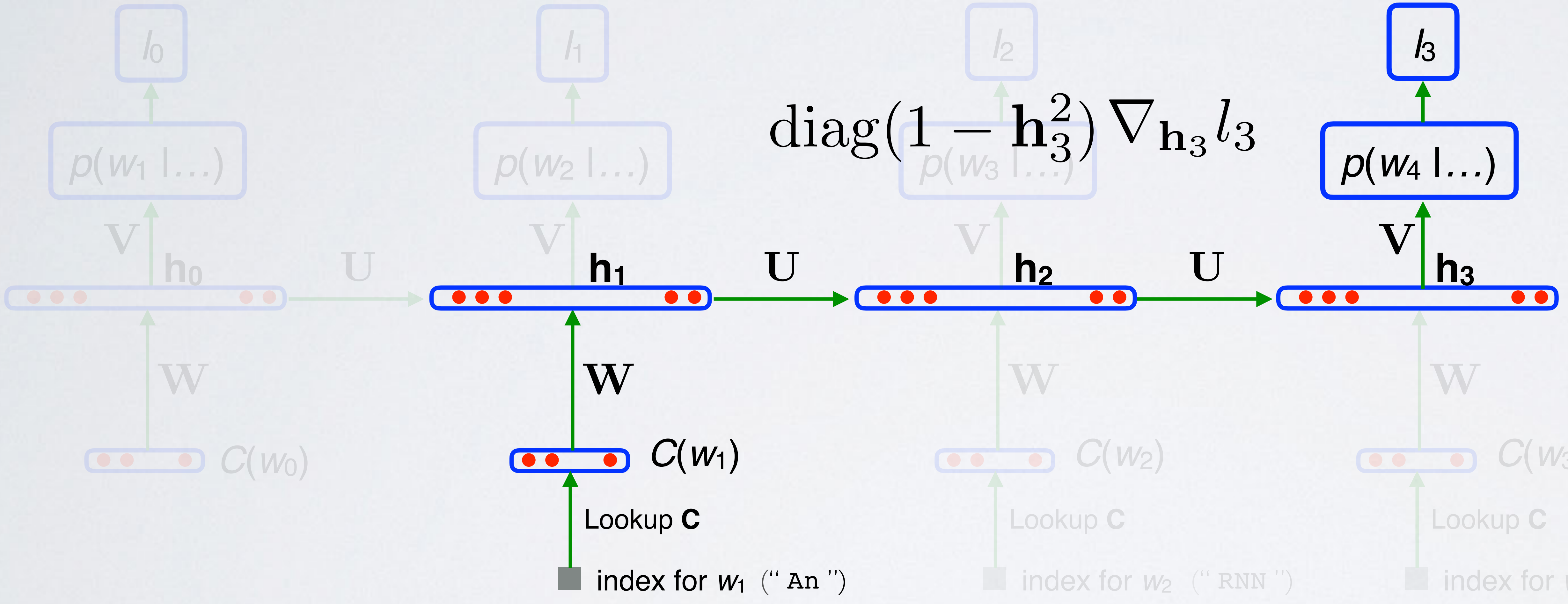
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



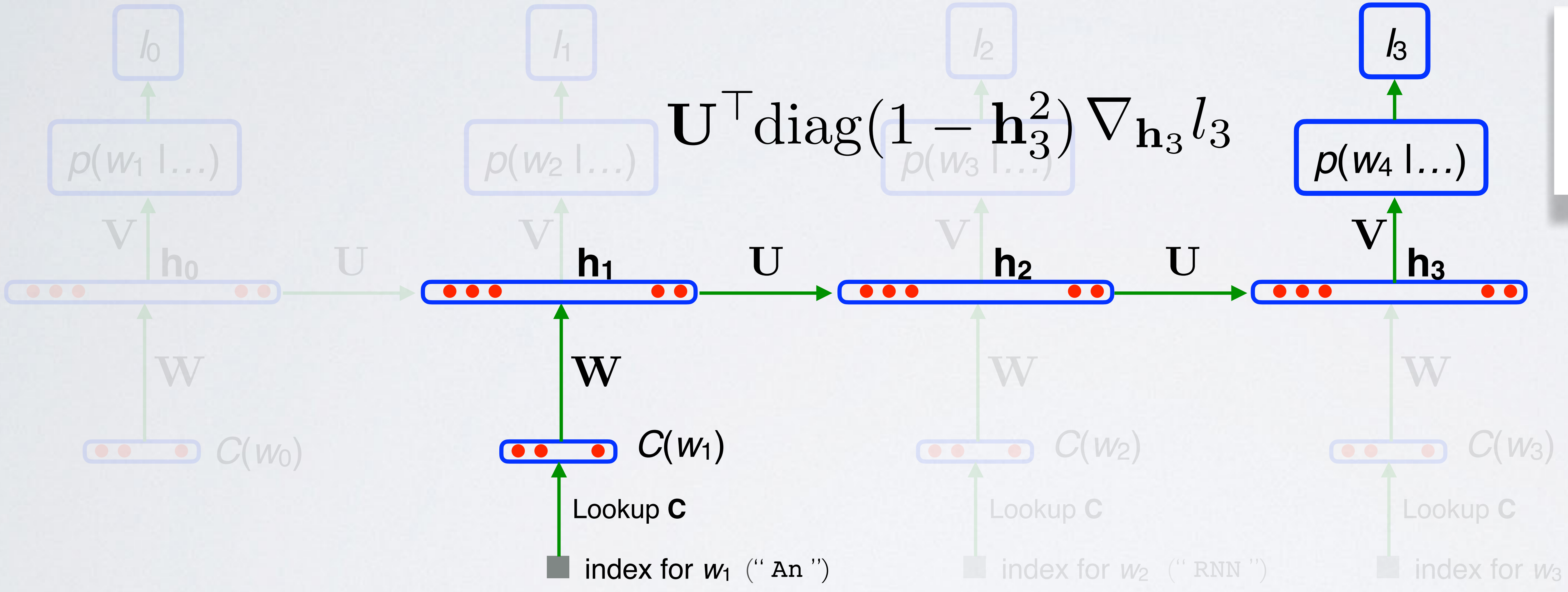
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



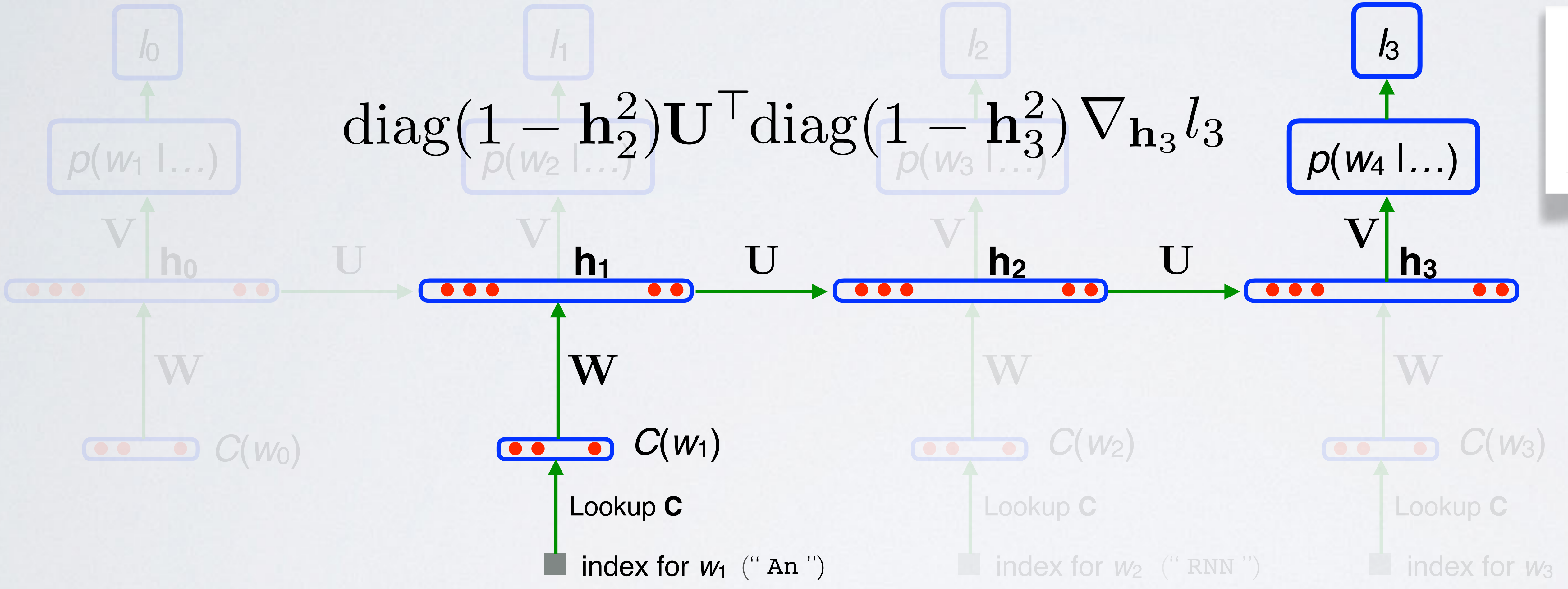
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

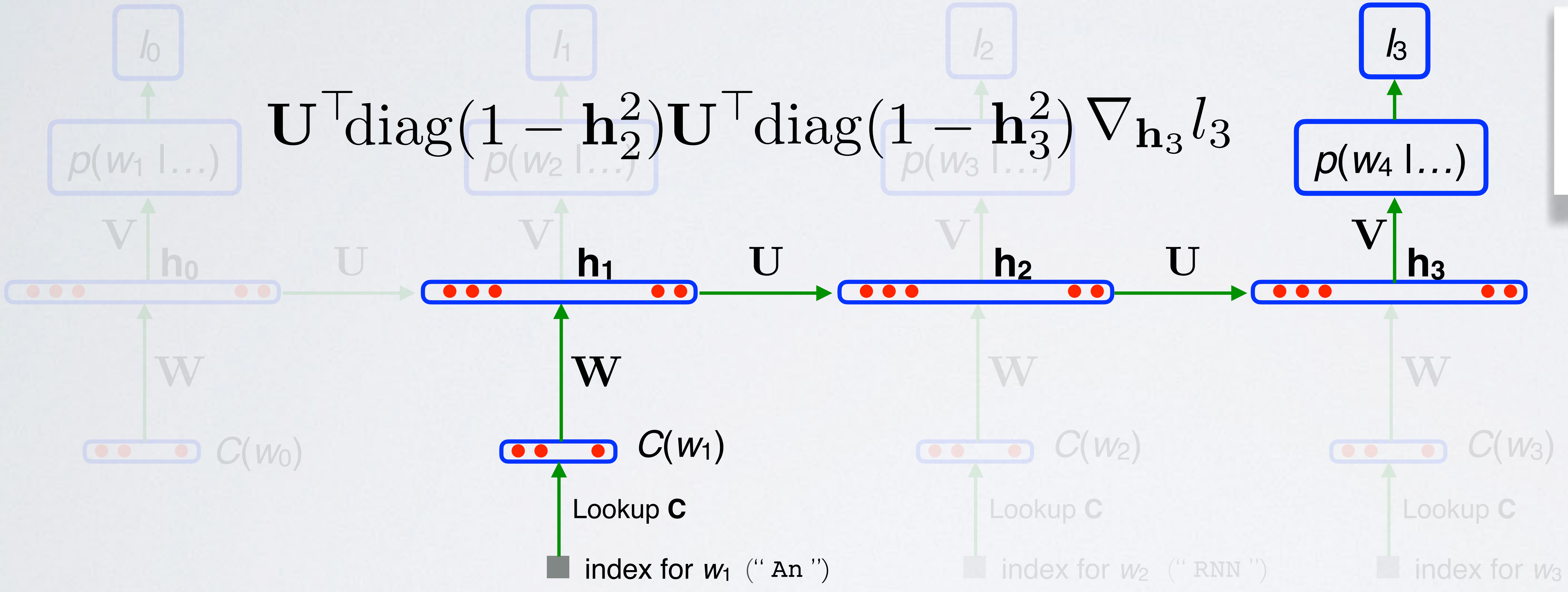
- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

$$\mathbf{U}^\top \mathrm{diag}(1-\mathbf{h}_2^2)\mathbf{U}^\top \mathrm{diag}(1-\mathbf{h}_3^2)\, \nabla_{\mathbf{h}_3} l_3$$



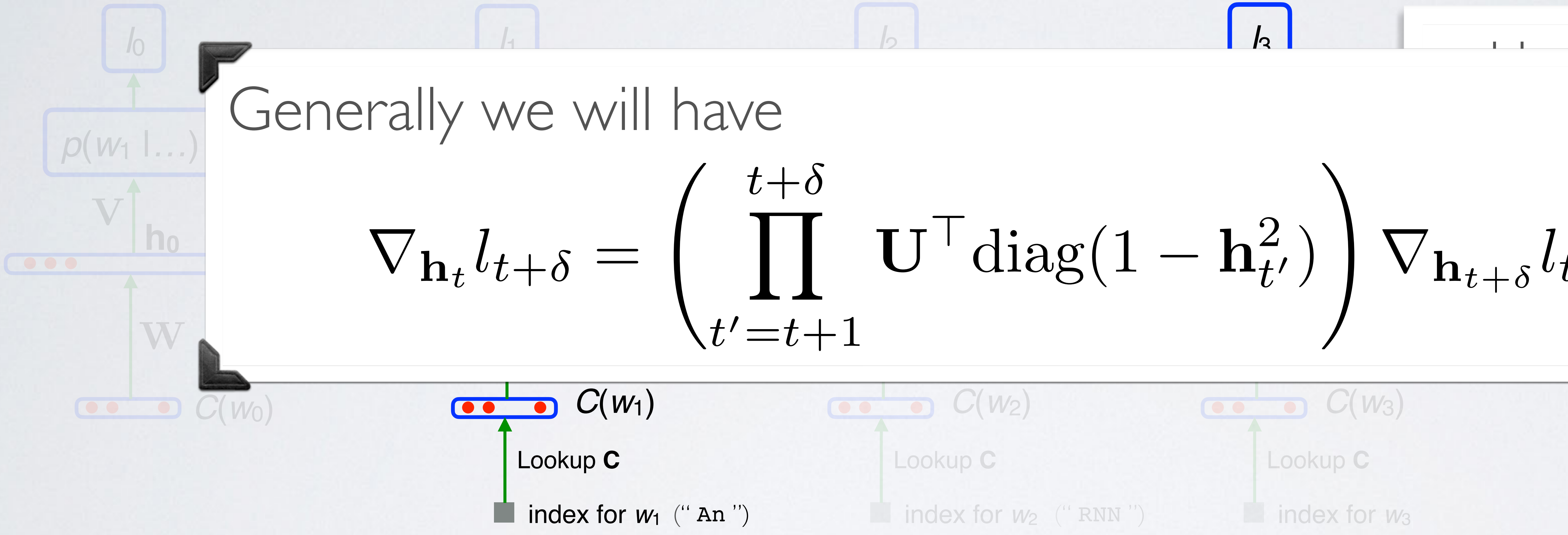
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



Generally we will have

$$\nabla_{\mathbf{h}_t} l_{t+\delta} = \left(\prod_{t'=t+1}^{t+\delta} \mathbf{U}^\top \mathrm{diag}(1 - \mathbf{h}_{t'}^2) \right) \nabla_{\mathbf{h}_{t+\delta}} l_{t+\delta}$$

- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: exploding gradient

Generally we will have

$$\nabla_{\mathbf{h}_t} l_{t+\delta} = \left(\prod_{t'=t+1}^{t+\delta} \mathbf{U}^\top \mathrm{diag}(1 - \mathbf{h}_{t'}^2) \right) \nabla_{\mathbf{h}_{t+\delta}} l_{t+\delta}$$

- What could go wrong, as δ increases?
  - if **U** is "large", then as δ grows, $\nabla_{\mathbf{h}_t} l_{t+\delta}$ will grow too (exponentially!)
  - if $\nabla_{\mathbf{h}_t} l_{t+\delta}$ is large, then so will the gradients on **W** and … **U** !
  - this is known as the **exploding gradient problem**

BACKPROPAGATION THROUGH TIME

Topics: gradient clipping

Generally we will have

$$\nabla_{\mathbf{h}_t} l_{t+\delta} = \left(\prod_{t'=t+1}^{t+\delta} \mathbf{U}^\top \mathrm{diag}(1 - \mathbf{h}_{t'}^2) \right) \nabla_{\mathbf{h}_{t+\delta}} l_{t+\delta}$$

- Solution: **gradient clipping**
  - let θ be any of parameter matrix/vector of the model (e.g. **W**, **U** or **V**)
  - before update, if the norm of $\nabla_\theta l$ is larger than some threshold C, rescale

$$\nabla_\theta l \Leftarrow C \times \frac{\nabla_\theta l}{||\nabla_\theta l||}$$

  - often applied where θ is the concatenation of *all* parameters of the model

BACKPROPAGATION THROUGH TIME

Topics: vanishing gradient

Generally we will have

$$\nabla_{\mathbf{h}_t} l_{t+\delta} = \left(\prod_{t'=t+1}^{t+\delta} \mathbf{U}^\top \mathrm{diag}(1 - \mathbf{h}_{t'}^2) \right) \nabla_{\mathbf{h}_{t+\delta}} l_{t+\delta}$$

- What could go wrong, as δ increases?
  - if **U** is "small" or hidden units $\mathbf{h}_{t'}$ are saturated, then $\nabla_{\mathbf{h}_t} l_{t+\delta}$ will shrink (exponentially!)
  - if $\nabla_{\mathbf{h}_t} l_{t+\delta}$ is small, then ***can't learn long term dependencies***
  - this is known as the **vanishing gradient problem**

BACKPROPAGATION THROUGH TIME

Topics: orthogonal initialization

Generally we will have

$$\nabla_{\mathbf{h}_t} l_{t+\delta} = \left(\prod_{t'=t+1}^{t+\delta} \mathbf{U}^\top \mathrm{diag}(1 - \mathbf{h}_{t'}^2) \right) \nabla_{\mathbf{h}_{t+\delta}} l_{t+\delta}$$

- Solution: **orthogonal initialization**
  - initialize **U** as a random orthogonal matrix (i.e. $\mathbf{U}^\top \mathbf{U} = \mathbf{U}\mathbf{U}^\top = \mathbf{I}$)
    - then multiplying with $\mathbf{U}^\top$ doesn't change the norm, since $(\mathbf{U}^\top \mathbf{v})^\top (\mathbf{U}^\top \mathbf{v}) = \mathbf{v}^\top \mathbf{U}\mathbf{U}^\top \mathbf{v} = \mathbf{v}^\top \mathbf{v}$
  - initialize as usual **W** (small random entries)
    - then $\mathrm{diag}(1 - \mathbf{h}_{t'}^2)$ is close to **I**
  - also useful to avoid exploding gradient (at least initially)

Recurrent neural networks

Long short-term memory network

BACKPROPAGATION THROUGH TIME

REMINDER

Topics: vanishing gradient

Generally we will have

$$\nabla_{\mathbf{h}_t} l_{t+\delta} = \left(\prod_{t'=t+1}^{t+\delta} \mathbf{U}^\top \mathrm{diag}(1 - \mathbf{h}_{t'}^2) \right) \nabla_{\mathbf{h}_{t+\delta}} l_{t+\delta}$$

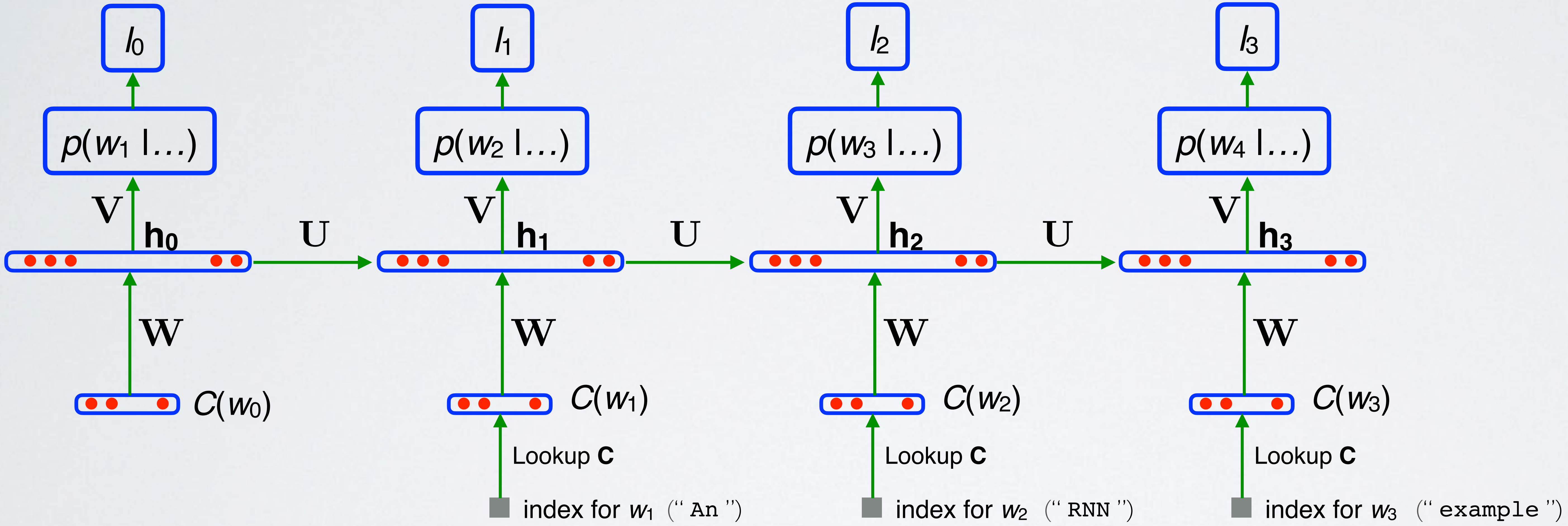
- What could go wrong, as δ increases?
  - if **U** is "small" or hidden units $\mathbf{h}_{t'}$ are saturated, then $\nabla_{\mathbf{h}_t} l_{t+\delta}$ will shrink (exponentially!)
  - if $\nabla_{\mathbf{h}_t} l_{t+\delta}$ is small, then ***can't learn long term dependencies***
  - this is known as the **vanishing gradient problem**

BACKPROPAGATION THROUGH TIME

REMINDER

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

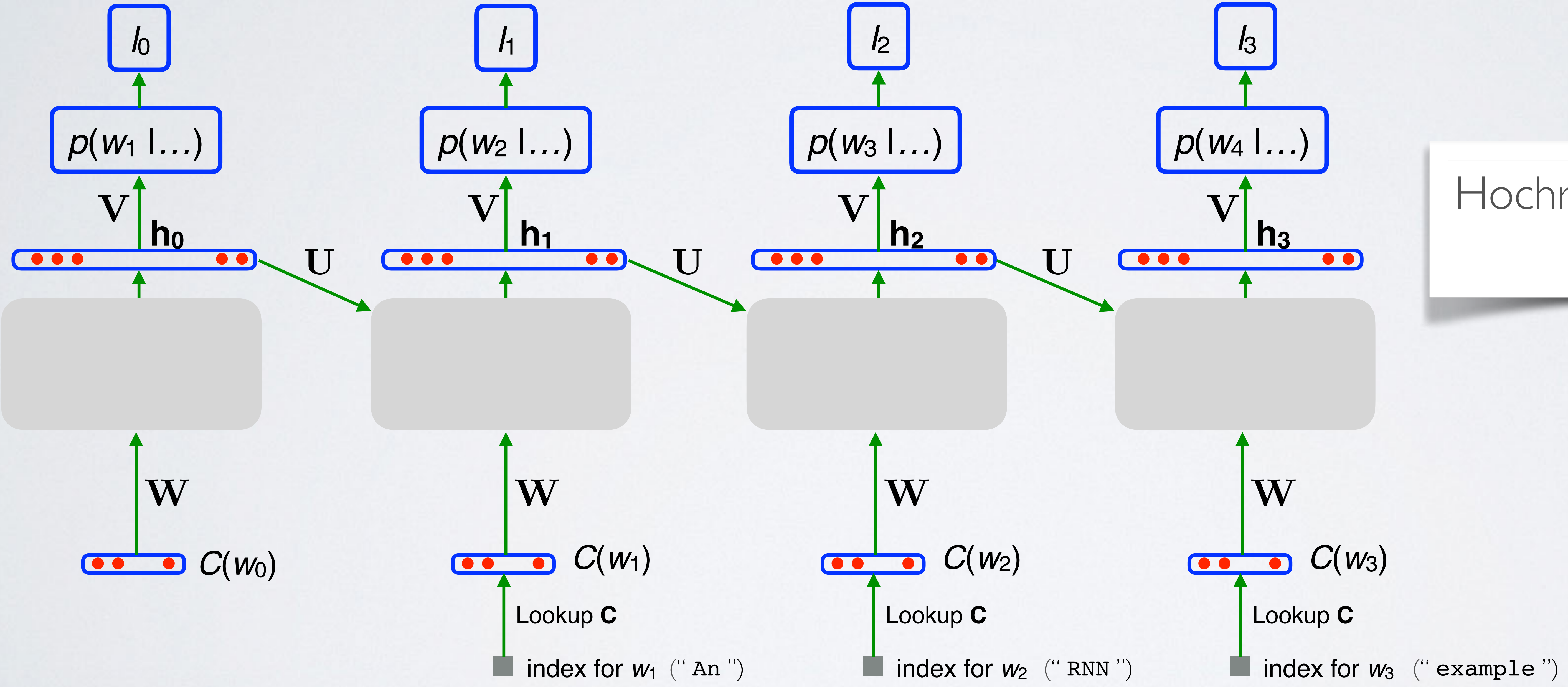


- want to minimize sum of per step loss $l = \sum_{t=0}^{T-1} l_t$
- for language modeling, $l_t = -\log p(w_{t+1} \mid \ldots)$

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**



Hochreiter, Schmidhuber
1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

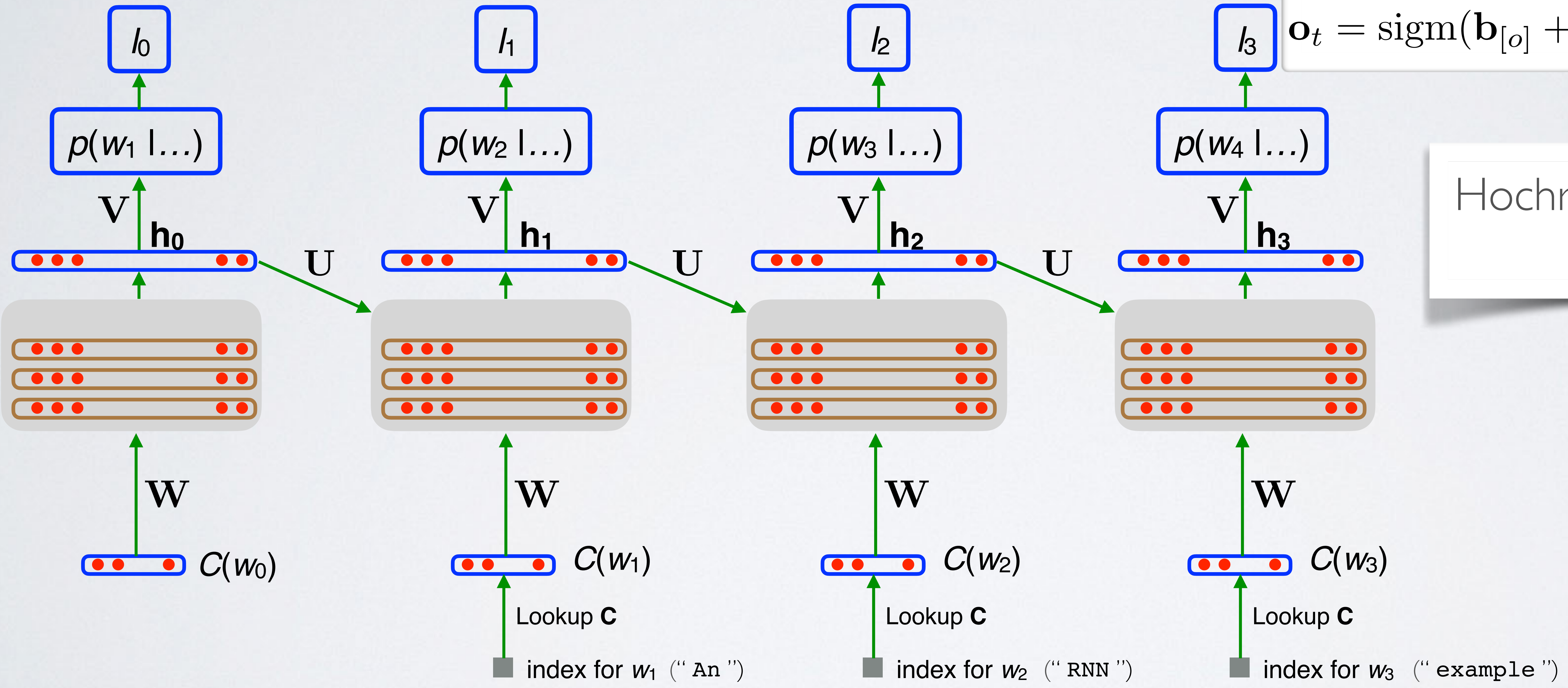
- Layer $\mathbf{h}_t$ is a function of **memory cells**

Input, forget, output gates:

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

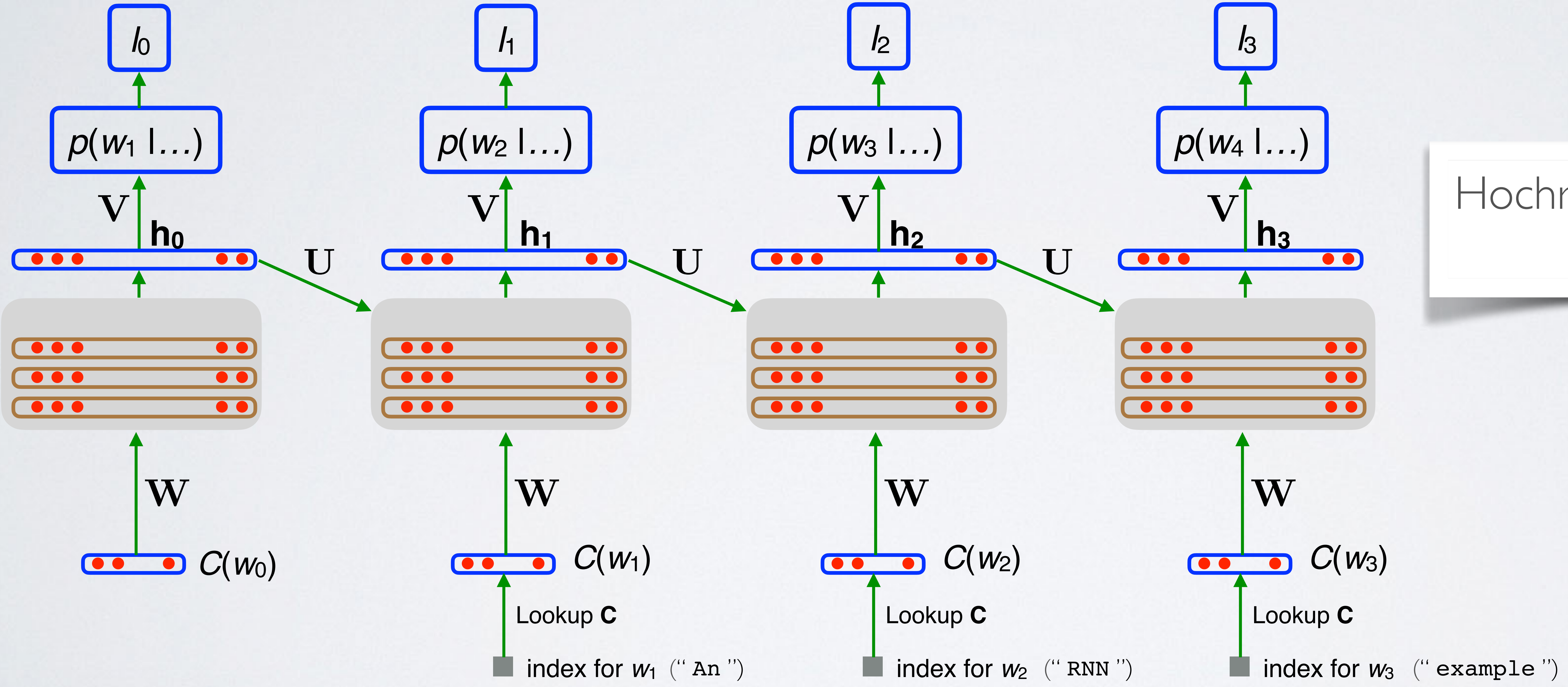


Hochreiter, Schmidhuber
1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**



Hochreiter, Schmidhuber
1995

LONG SHORT-TERM MEMORY NETWORK

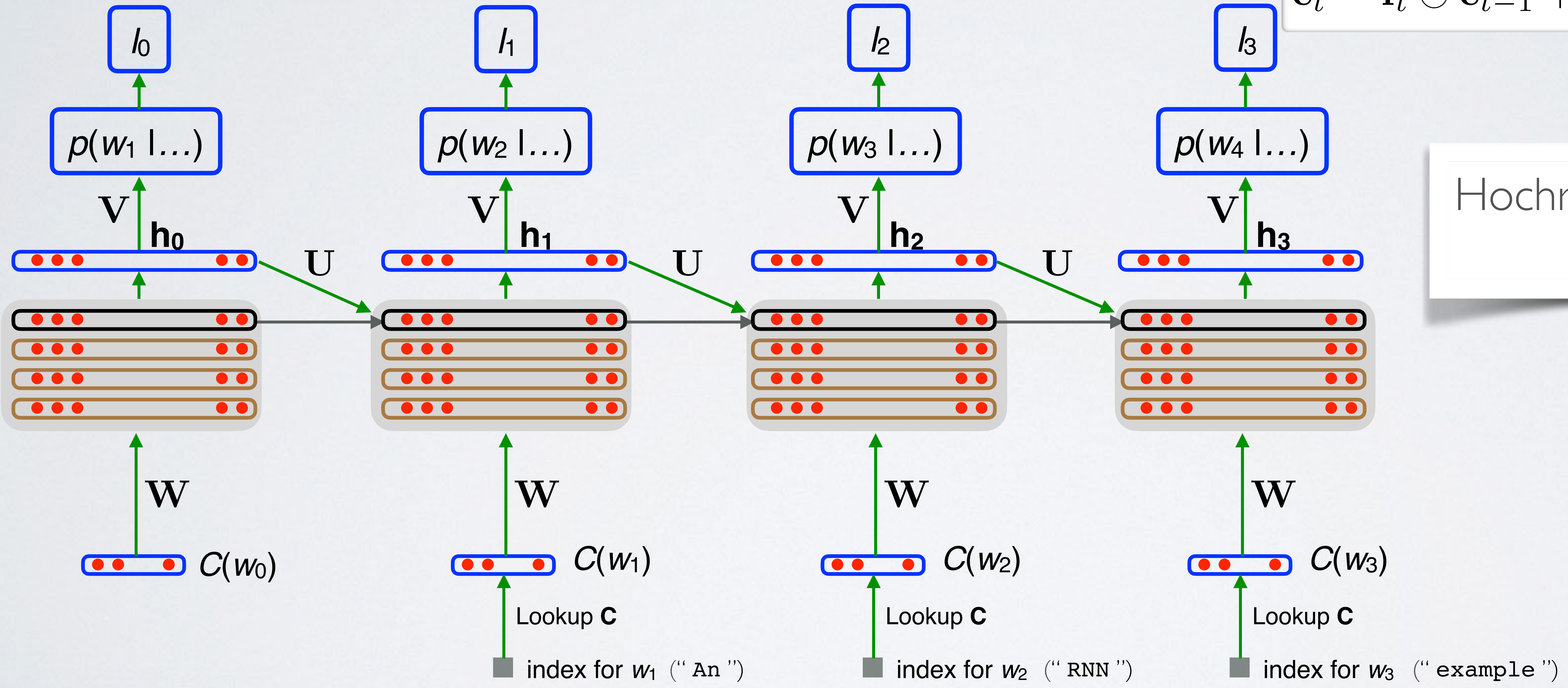
Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**

Cell state:

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

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

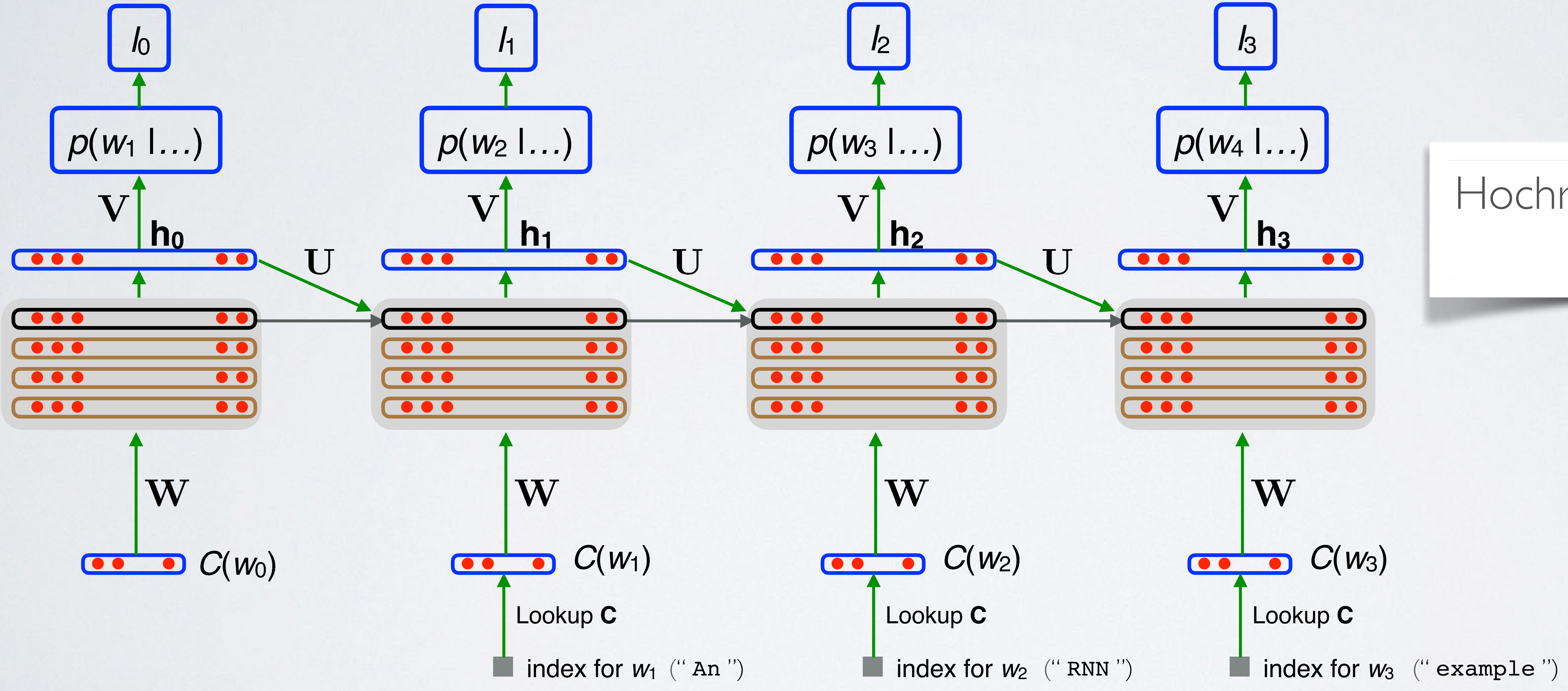


Hochreiter, Schmidhuber
1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**



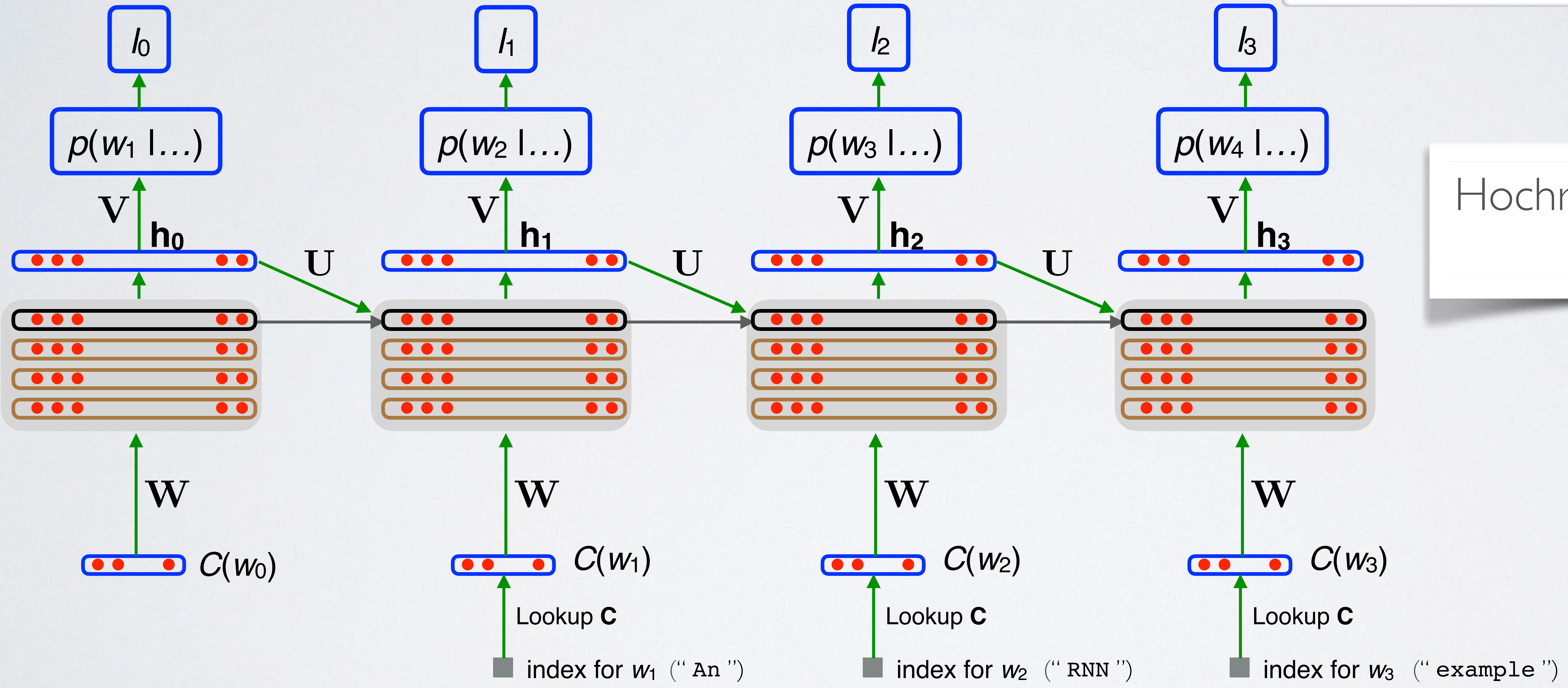
Hochreiter, Schmidhuber
1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**

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



Hochreiter, Schmidhuber
1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

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

Hidden layer:

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

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

The **gates** control the flow of information in ($\mathbf{i}_t$, $\mathbf{f}_t$) and out ($\mathbf{o}_t$) of the cell

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

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

Hidden layer:

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

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

The **gates** control the flow of information in ($\mathbf{i}_t$, $\mathbf{f}_t$,) and out ($\mathbf{o}_t$) of the cell

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

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

The **cell state** maintains information on the input

Hidden layer:

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

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

The **gates** control the flow of information in ($\mathbf{i}_t$, $\mathbf{f}_t$) and out ($\mathbf{o}_t$) of the cell

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

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

The **cell state** maintains information on the input

Hidden layer:

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

The **hidden layer** sees what passes through the output gate

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$
$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$
$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

Hidden layer:

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

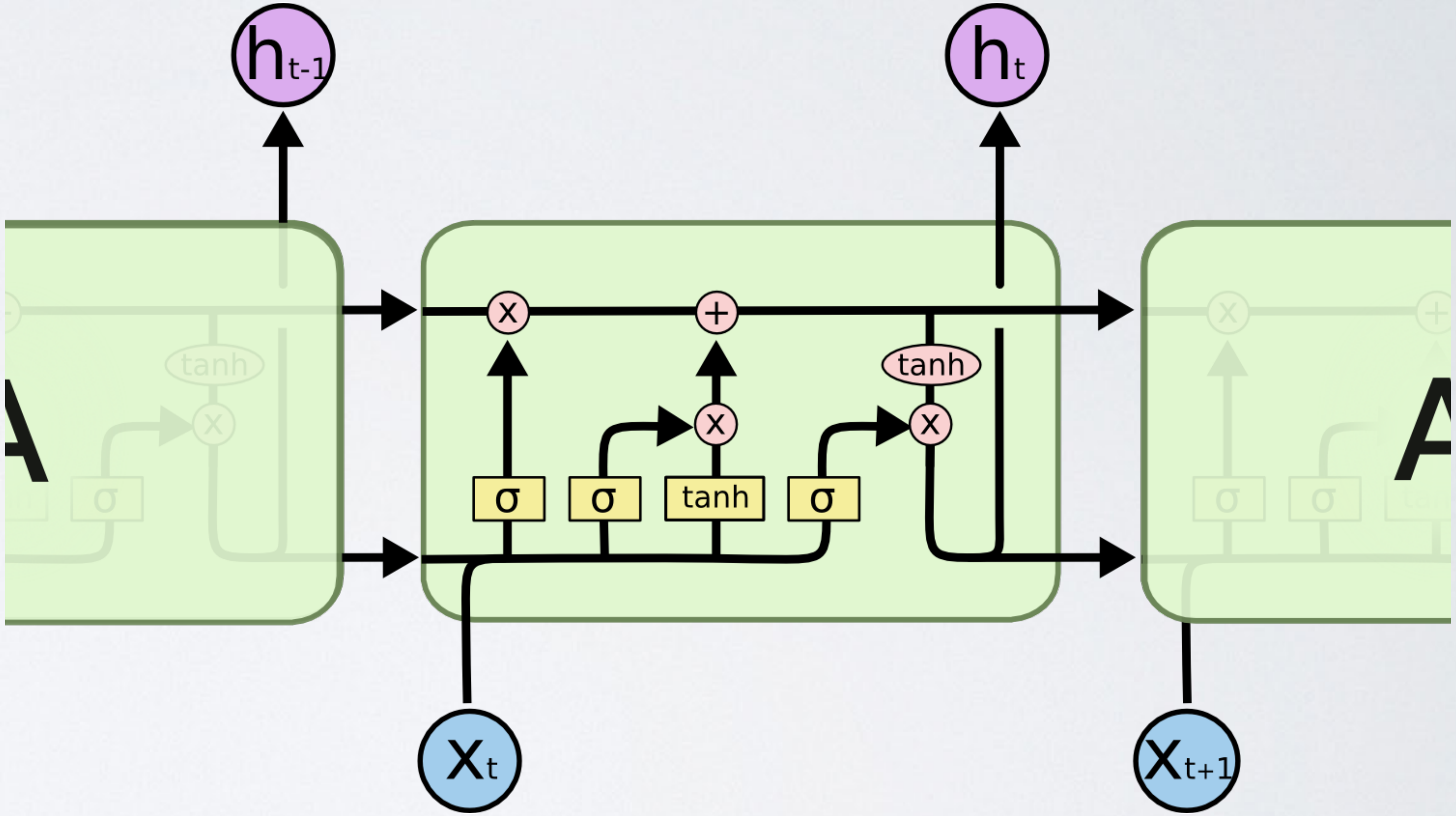


Image from Chris Olah's Blog post on Understanding LSTMs

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

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

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

$$\mathbf{c}_t = \ \mathbf{f}_t \odot \mathbf{f}_{t-1} \odot \mathbf{c}_{t-2} + \mathbf{f}_t \odot \mathbf{i}_{t-1} \odot \widetilde{\mathbf{c}}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

$$\mathbf{c}_t = \sum_{t'=0}^{t} \mathbf{f}_t \odot \cdots \odot \mathbf{f}_{t'+1} \odot \mathbf{i}_{t'} \odot \widetilde{\mathbf{c}}_{t'}$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

$$\mathbf{c}_t = \sum_{t'=0}^{t} \mathbf{f}_t \odot \cdots \odot \mathbf{f}_{t'+1} \odot \mathbf{i}_{t'} \odot \widetilde{\mathbf{c}}_{t'}$$

- As long as forget gates are open (close to 1), gradient may pass into $\widetilde{\mathbf{c}}_t$ over long time gaps
  - saturation of forget gates doesn't stop gradient flow
  - suggests that a better initialization of forget gate bias $\mathbf{b}_{[f]}$ is $\gg 0$ (e.g. 1)
- Learning with BPTT more effective
  - easy to compute gradients with automatic differentiation

LONG SHORT-TERM MEMORY NETWORK

Topics: LSTM variations

- Can add "peephole connections" to the gates

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$
$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$
$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

- For historical perspective and empirical ablation analysis

  LSTM: A Search Space Odyssey
  Greff, Srivastava, Koutník, Steunebrink, Schmidhuber
  2015

  - forget gate are crucial, as well as output gates if cell state unbounded
  - coupling the input and forget gate ($\mathbf{f}_t = 1 - \mathbf{i}_t$) can also work well

LONG SHORT-TERM MEMORY NETWORK

Topics: LSTM variations

- Can add "peephole connections" to the gates

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{V}_{[i]}\mathbf{c}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$
$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{V}_{[f]}\mathbf{c}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$
$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{V}_{[o]}\mathbf{c}_{t} + \mathbf{W}_{[o]}C(w_t))$$

- For historical perspective and empirical ablation analysis

  LSTM: A Search Space Odyssey

  Greff, Srivastava, Koutník, Steunebrink, Schmidhuber

  2015

  - forget gate are crucial, as well as output gates if cell state unbounded
  - coupling the input and forget gate ($\mathbf{f}_t$=1 - $\mathbf{i}_t$) can also work well

Recurrent neural networks

Gated recurrent units network

LONG SHORT-TERM MEMORY NETWORK

REMINDER

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

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

Hidden layer:

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

GATED RECURRENT UNITS NETWORK

Topics: gated recrurrent units (GRU) network

Cho, Merrienboer, Bahdanau, Bengio 2014

LSTM

Input, forget, output gates:

$$\mathbf{i}_t = \mathrm{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \mathrm{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \mathrm{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

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

Hidden layer:

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

GRU

Update, reset gates:

$$\mathbf{z}_t = \mathrm{sigm}(\mathbf{b}_{[z]} + \mathbf{U}_{[z]}\mathbf{h}_{t-1} + \mathbf{W}_{[z]}C(w_t))$$

$$\mathbf{r}_t = \mathrm{sigm}(\mathbf{b}_{[r]} + \mathbf{U}_{[r]}\mathbf{h}_{t-1} + \mathbf{W}_{[r]}C(w_t))$$

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{[c]}C(w_t))$$

$$\mathbf{c}_t = (1 - \mathbf{z}_t) \odot \mathbf{c}_{t-1} + \mathbf{z}_t \odot \widetilde{\mathbf{c}}_t$$

Hidden layer:

$$\mathbf{h}_t = \mathbf{c}_t$$

GATED RECURRENT UNITS NETWORK

Cho, Merrienboer, Bahdanau, Bengio 2014

Topics: gated recrurrent units (GRU) network

- To sum up:

Update, reset gates:

$$\mathbf{z}_t = \mathrm{sigm}(\mathbf{b}_{[z]} + \mathbf{U}_{[z]}\mathbf{h}_{t-1} + \mathbf{W}_{[z]}C(w_t))$$

$$\mathbf{r}_t = \mathrm{sigm}(\mathbf{b}_{[r]} + \mathbf{U}_{[r]}\mathbf{h}_{t-1} + \mathbf{W}_{[r]}C(w_t))$$

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{[c]}C(w_t))$$

$$\mathbf{c}_t = (1 - \mathbf{z}_t) \odot \mathbf{c}_{t-1} + \mathbf{z}_t \odot \widetilde{\mathbf{c}}_t$$

Hidden layer:

$$\mathbf{h}_t = \mathbf{c}_t$$

GATED RECURRENT UNITS NETWORK

Cho, Merrienboer, Bahdanau, Bengio 2014

Topics: gated recrurrent units (GRU) network

- To sum up:

Update, reset gates:

$$\mathbf{z}_t = \mathrm{sigm}(\mathbf{b}_{[z]} + \mathbf{U}_{[z]}\mathbf{h}_{t-1} + \mathbf{W}_{[z]}C(w_t))$$

$$\mathbf{r}_t = \mathrm{sigm}(\mathbf{b}_{[r]} + \mathbf{U}_{[r]}\mathbf{h}_{t-1} + \mathbf{W}_{[r]}C(w_t))$$

Fewer **gates**, thus fewer parameters and computations

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{[c]}C(w_t))$$

$$\mathbf{c}_t = (1 - \mathbf{z}_t) \odot \mathbf{c}_{t-1} + \mathbf{z}_t \odot \widetilde{\mathbf{c}}_t$$

Hidden layer:

$$\mathbf{h}_t = \mathbf{c}_t$$

GATED RECURRENT UNITS NETWORK

Cho, Merrienboer, Bahdanau, Bengio
2014

Topics: gated recrurrent units (GRU) network

- To sum up:

Update, reset gates:

$$\mathbf{z}_t = \mathrm{sigm}(\mathbf{b}_{[z]} + \mathbf{U}_{[z]}\mathbf{h}_{t-1} + \mathbf{W}_{[z]}C(w_t))$$

$$\mathbf{r}_t = \mathrm{sigm}(\mathbf{b}_{[r]} + \mathbf{U}_{[r]}\mathbf{h}_{t-1} + \mathbf{W}_{[r]}C(w_t))$$

Fewer **gates**, thus fewer parameters and computations

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{[c]}C(w_t))$$

$$\mathbf{c}_t = (1 - \mathbf{z}_t) \odot \mathbf{c}_{t-1} + \mathbf{z}_t \odot \widetilde{\mathbf{c}}_t$$

Update gate within **cell update**
Coupling of forget and input gates

Hidden layer:

$$\mathbf{h}_t = \mathbf{c}_t$$

GATED RECURRENT UNITS NETWORK

Cho, Merrienboer, Bahdanau, Bengio 2014

Topics: gated recrurrent units (GRU) network

- To sum up:

Update, reset gates:

$$\mathbf{z}_t = \mathrm{sigm}(\mathbf{b}_{[z]} + \mathbf{U}_{[z]}\mathbf{h}_{t-1} + \mathbf{W}_{[z]}C(w_t))$$

$$\mathbf{r}_t = \mathrm{sigm}(\mathbf{b}_{[r]} + \mathbf{U}_{[r]}\mathbf{h}_{t-1} + \mathbf{W}_{[r]}C(w_t))$$

Fewer **gates**, thus fewer parameters and computations

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{[c]}C(w_t))$$

$$\mathbf{c}_t = (1 - \mathbf{z}_t) \odot \mathbf{c}_{t-1} + \mathbf{z}_t \odot \widetilde{\mathbf{c}}_t$$

Update gate within **cell update**
Coupling of forget and input gates

Hidden layer:

$$\mathbf{h}_t = \mathbf{c}_t$$

Hidden layer *is* the cell state, so fewer computations there too

Recurrent neural networks

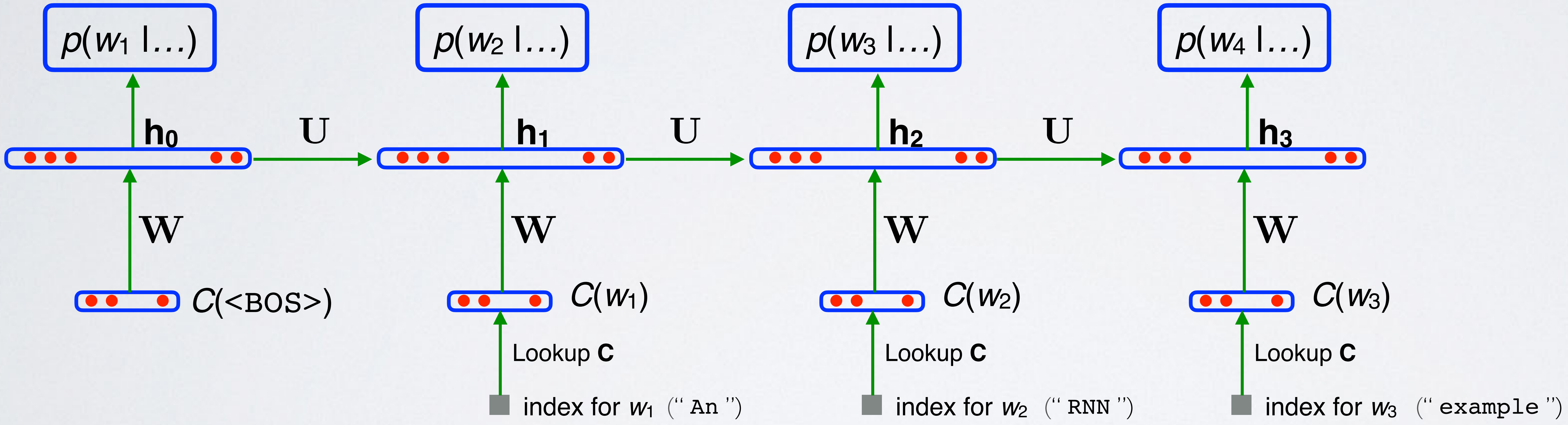
Sequence to sequence learning

RNN LANGUAGE MODEL

REMINDER

Topics: unrolled RNN

- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [“ `An` ”, “ `RNN` ”, “ `example` ”, “ `.` ”] ($T = 4$)



  - symbol “ `.` ” serves as an *end of sentence* symbol
  - $\mathbf{h}_0 = \tanh(\mathbf{b} + \mathbf{W}C(\texttt{<BOS>}))$, where $C(\texttt{<BOS>})$ o is a unique embedding for the *beginning of sentence* position(<BOS> not included as possible output!)

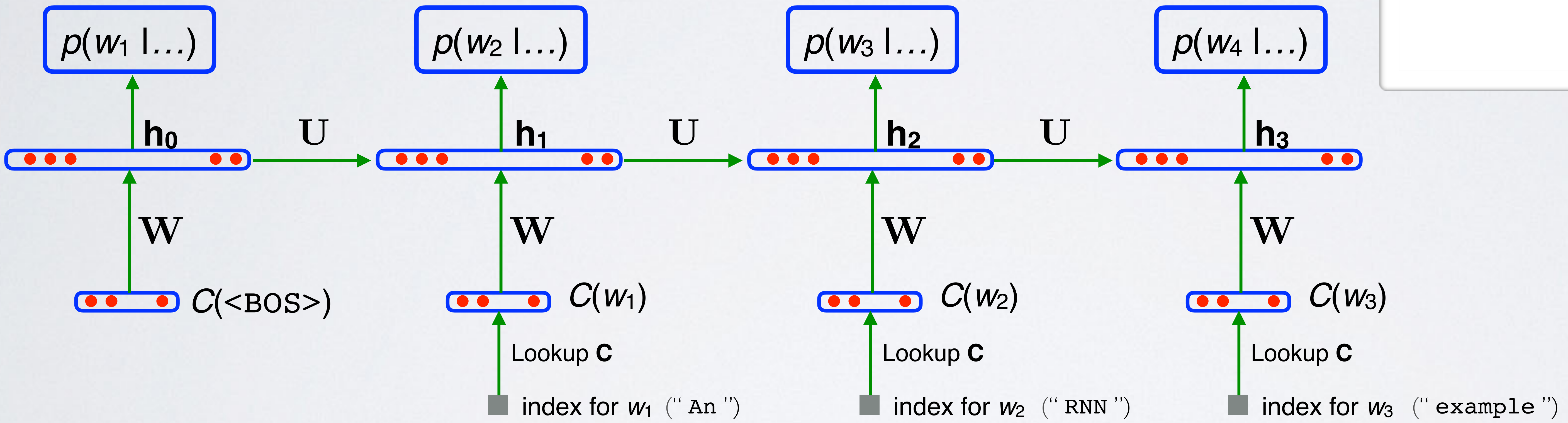
RNN LANGUAGE MODEL

REMINDER

Topics: unrolled RNN

- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)

$$p(\mathbf{w}) = p(w_1) \times p(w_2 \mid w_1) \times p(w_3 \mid w_1, w_2) \times p(w_4 \mid w_1, w_2, w_3)$$



  - symbol " `.` " serves as an *end of sentence* symbol
  - $\mathbf{h}_0 = \tanh(\mathbf{b} + \mathbf{W}C(\text{<BOS>}))$, where $C(\text{<BOS>})$ o is a unique embedding for the *beginning of sentence* position(<BOS> not included as possible output!)

SEQUENCE TO SEQUENCE LEARNING

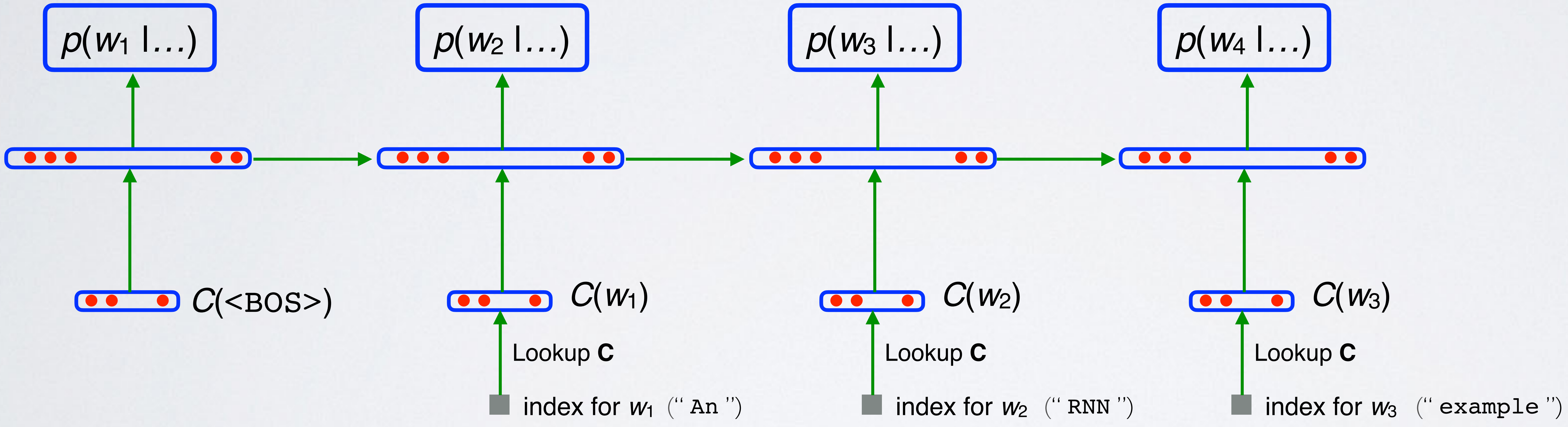
Topics: sequence to sequence (Seq2Seq) learning

- An RNN (LSTM or GRU) gives us good models over a target sequence space **w**
- What if we wanted to predict the target sequence **w** from some input sequence **x**
    - example application: machine translation
    - can't assume that **w** has same size as **x** or are aligned, so could not treat as a simpler tagging problem
- Referred to as **sequence to sequence** (**Seq2Seq**) learning
    - RNNs can be used to construct a model for Seq2Seq

SEQUENCE TO SEQUENCE LEARNING

Topics: sequence to sequence (Seq2Seq) learning

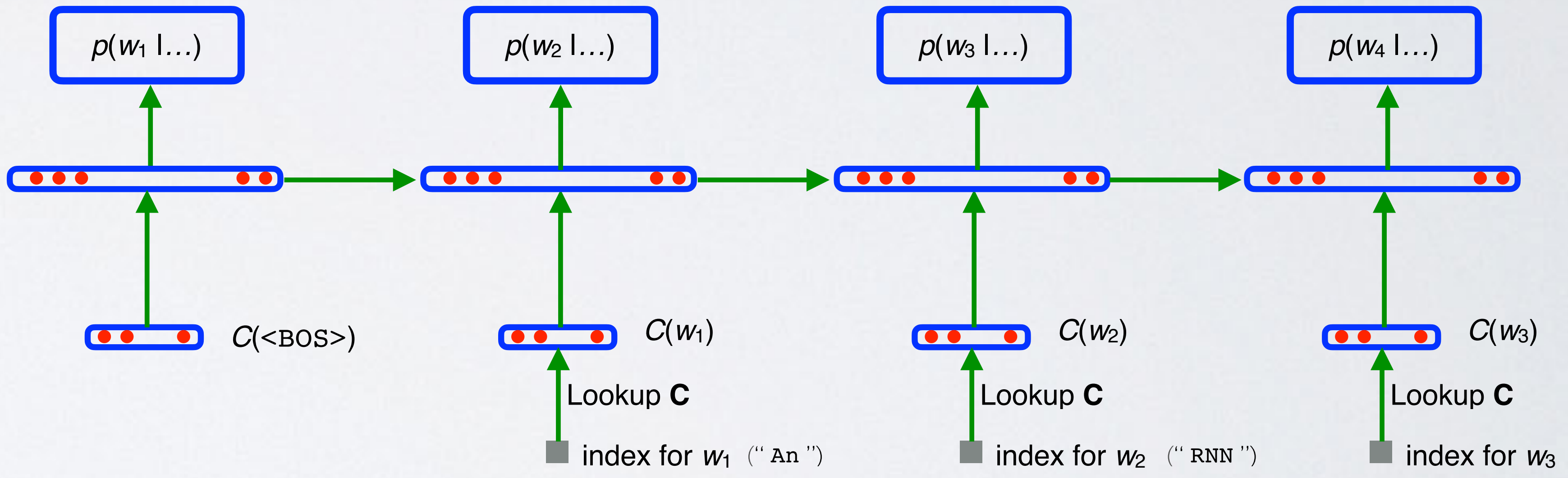
- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)



SEQUENCE TO SEQUENCE LEARNING

Topics: sequence to sequence (Seq2Seq) learning

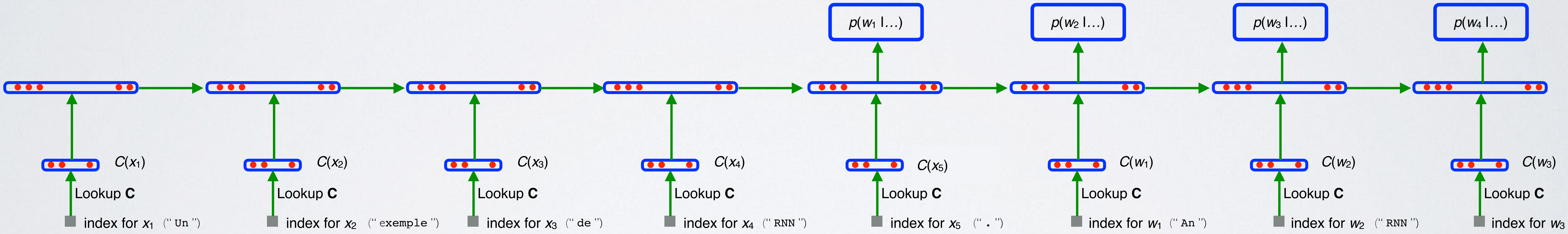
- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)



SEQUENCE TO SEQUENCE LEARNING

Topics: sequence to sequence (Seq2Seq) learning

- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)
    $\mathbf{x}$ = [" `Un` ", " `exemple` ", " `de` ", " `RNN` ", " `.` "] ($T_{\mathbf{x}} = 5$)

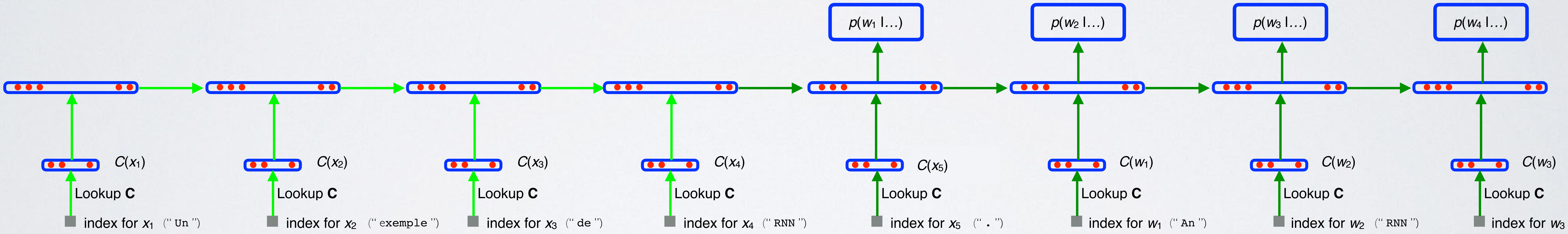


SEQUENCE TO SEQUENCE LEARNING

Topics: sequence to sequence (Seq2Seq) learning

- View of RNN unrolled through time
  - example: $\mathbf{w} = [\text{``}\texttt{An}\text{''}, \text{``}\texttt{RNN}\text{''}, \text{``}\texttt{example}\text{''}, \text{``}\texttt{.}\text{''}]$ $(T = 4)$

    $\mathbf{x} = [\text{``}\texttt{Un}\text{''}, \text{``}\texttt{exemple}\text{''}, \text{``}\texttt{de}\text{''}, \text{``}\texttt{RNN}\text{''}, \text{``}\texttt{.}\text{''}]$ $(T_{\mathbf{x}} = 5)$

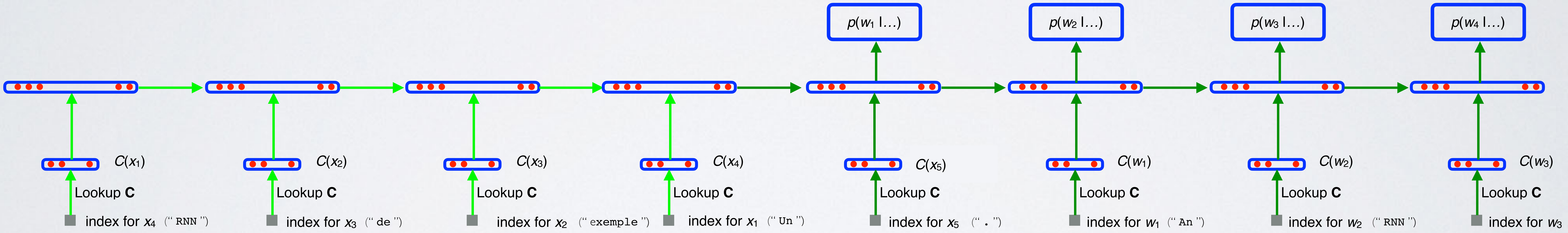


  - may work better by using different RNN parameters to process $\mathbf{x}$

SEQUENCE TO SEQUENCE LEARNING

Topics: sequence to sequence (Seq2Seq) learning

- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)
    $\mathbf{x}$ = [" `Un` ", " `exemple` ", " `de` ", " `RNN` ", " `.` "] ($T_{\mathbf{x}} = 5$)



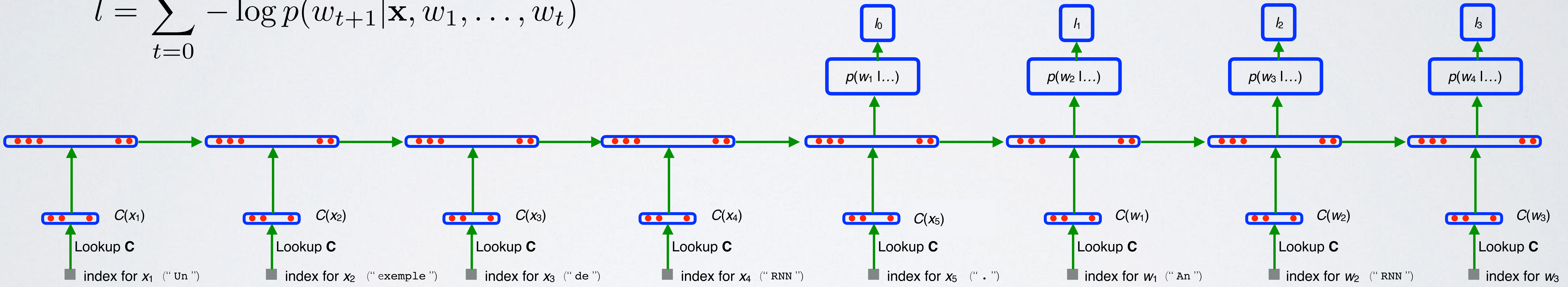
  - may work better by using different RNN parameters to process $\mathbf{x}$
  - may work better by processing sequence $\mathbf{x}$ in reverse order

SEQUENCE TO SEQUENCE LEARNING

Topics: training

- Provides a model for $p(\mathbf{w}|\mathbf{x})$
  - trained with BPTT and gradient descent on loss:

$$l = \sum_{t=0}^{T-1} -\log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$$

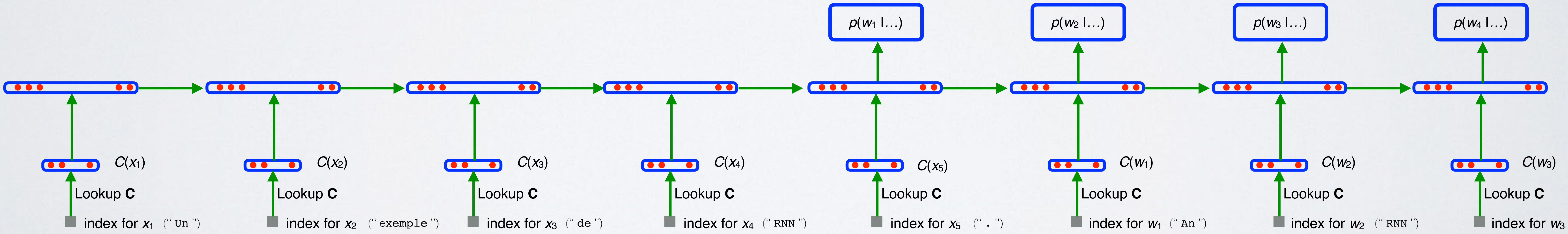


  - in practice, group examples into mini-batches of sequences with similar sizes

SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

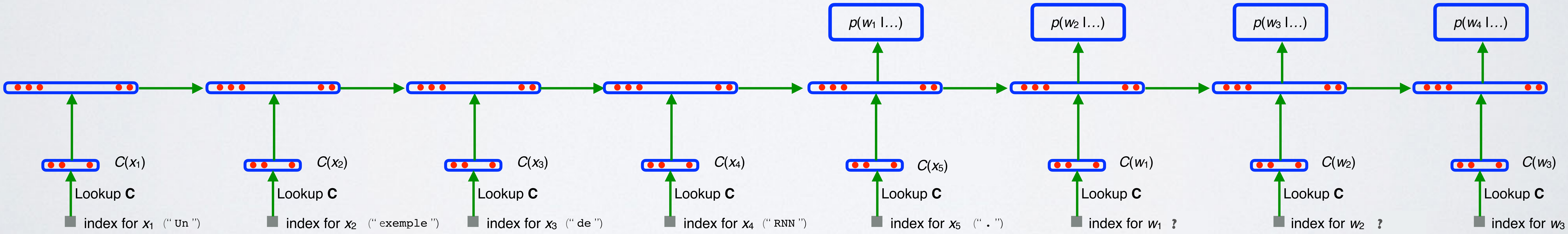
- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences (“hypotheses”)
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol



SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol

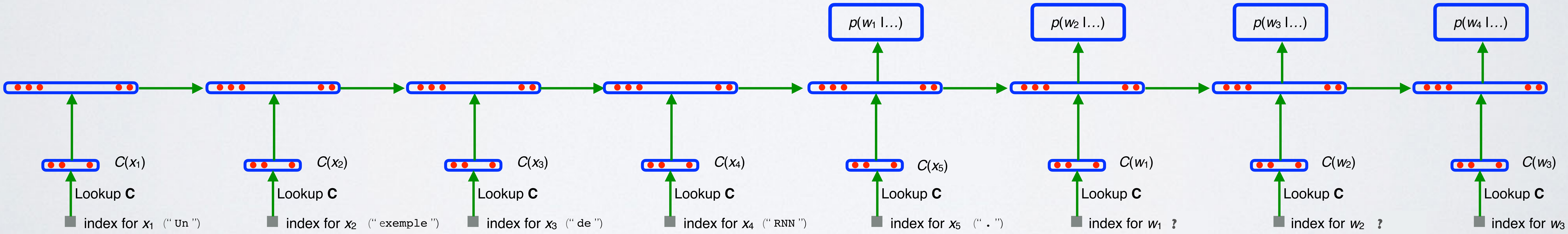


SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol

beam of size *k*=2 {

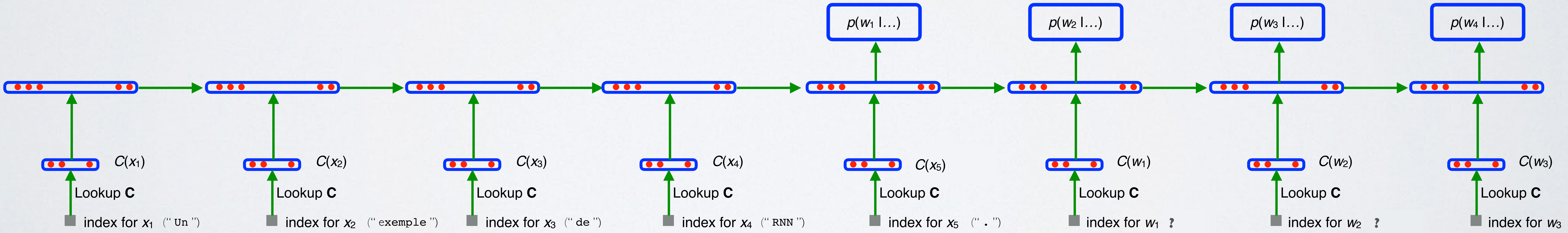


SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol

beam of size *k*=2 {
1. The
2. An
3. …

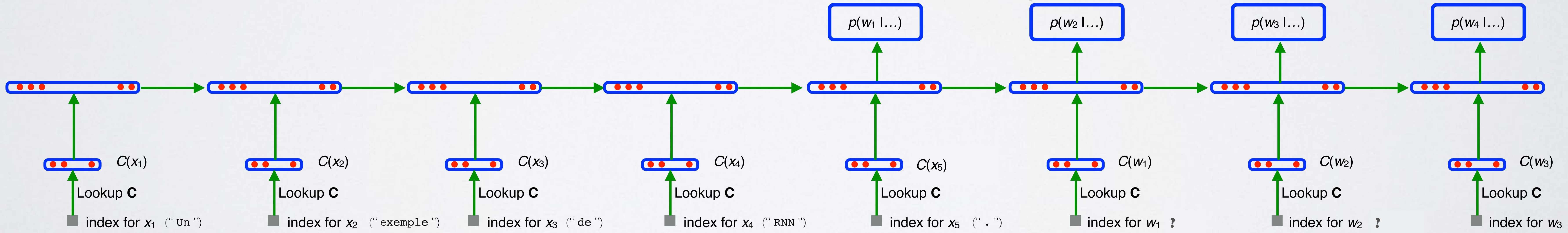


SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

- At test time, must find $\text{argmax}\ p(\mathbf{w}|\mathbf{x}) = \text{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol

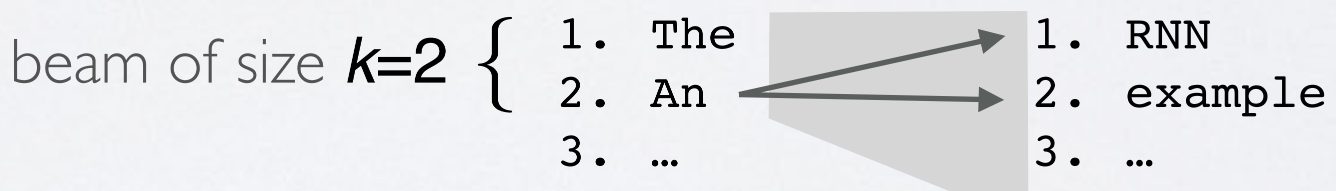
beam of size *k*=2 { 1. The
2. An
3. …

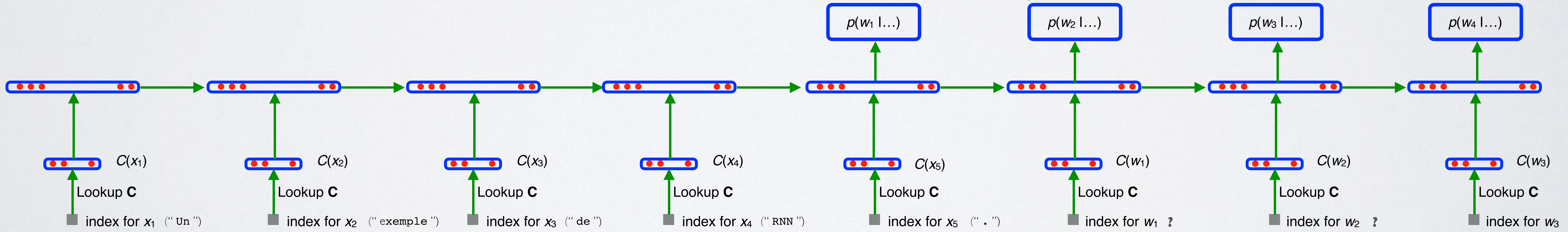


SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol

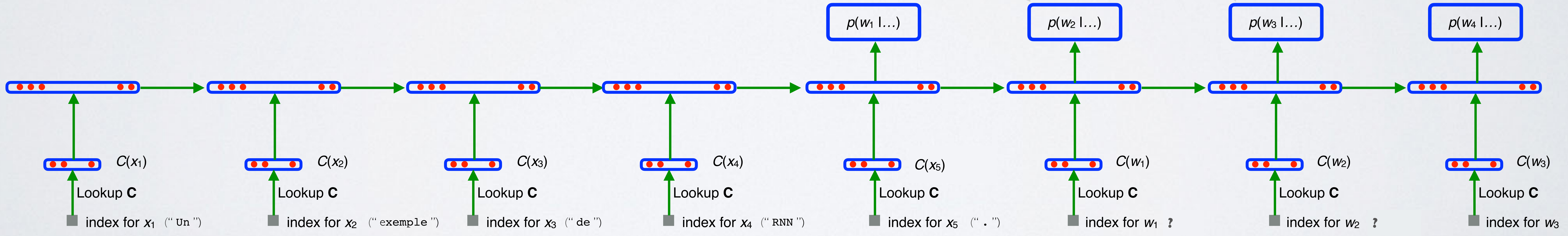




SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

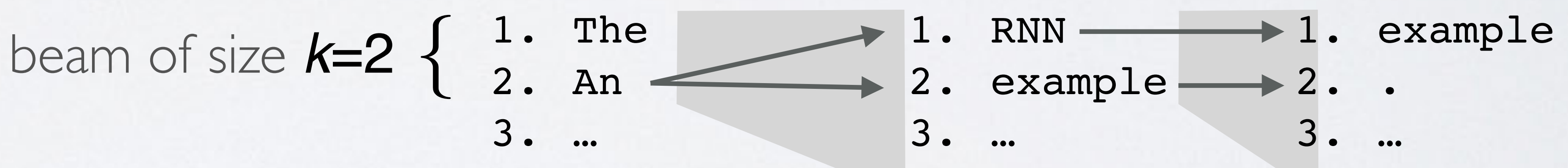
- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol

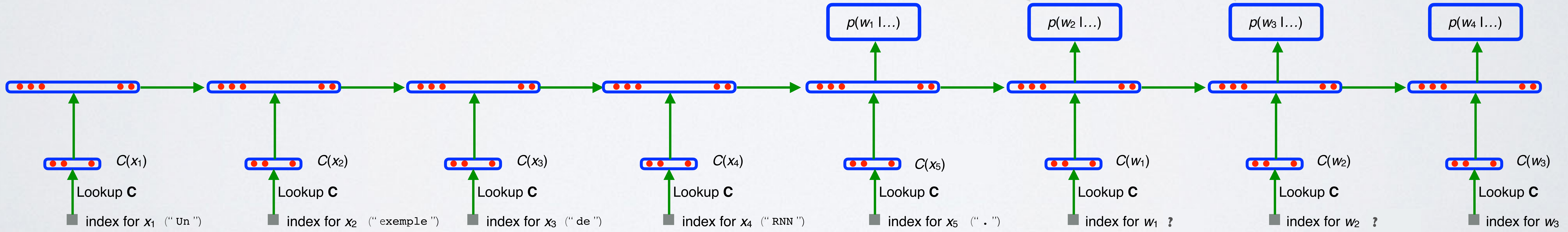


SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol

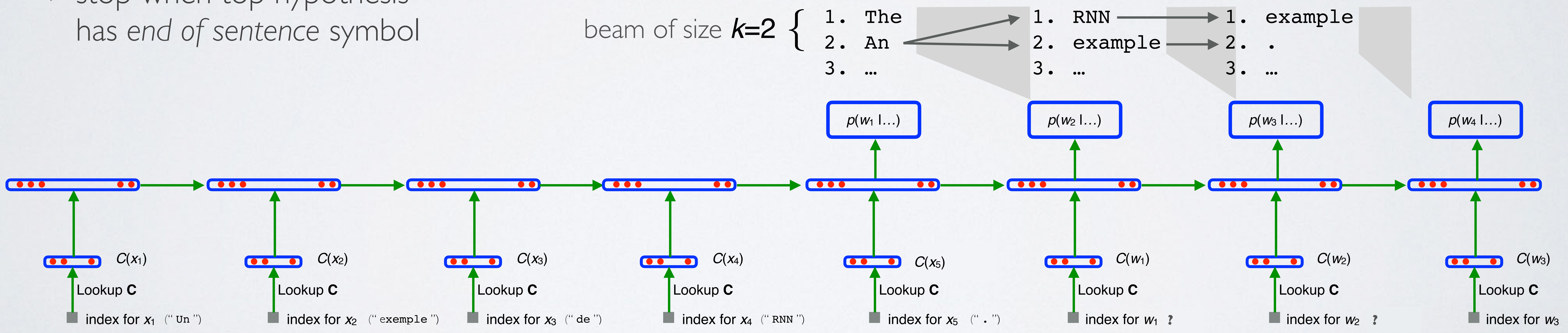




SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

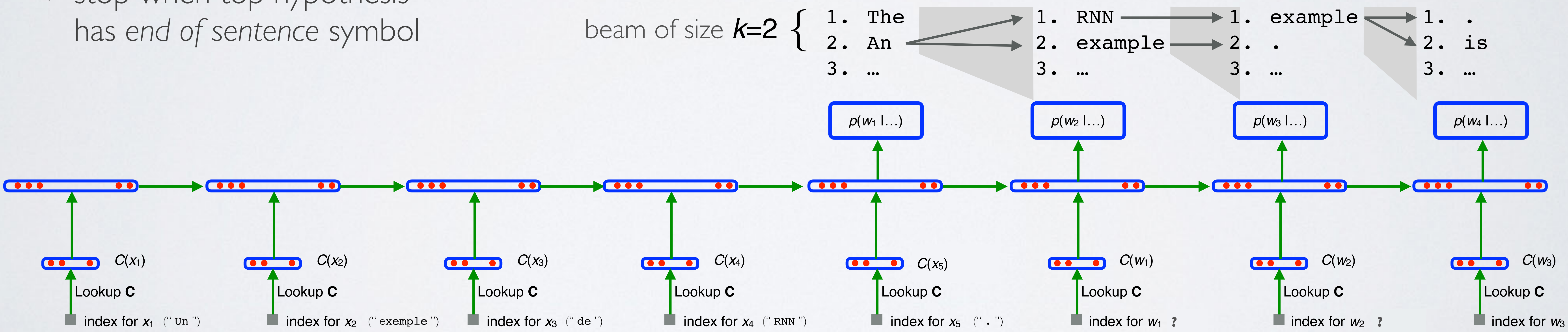
- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences (“hypotheses”)
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol



SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

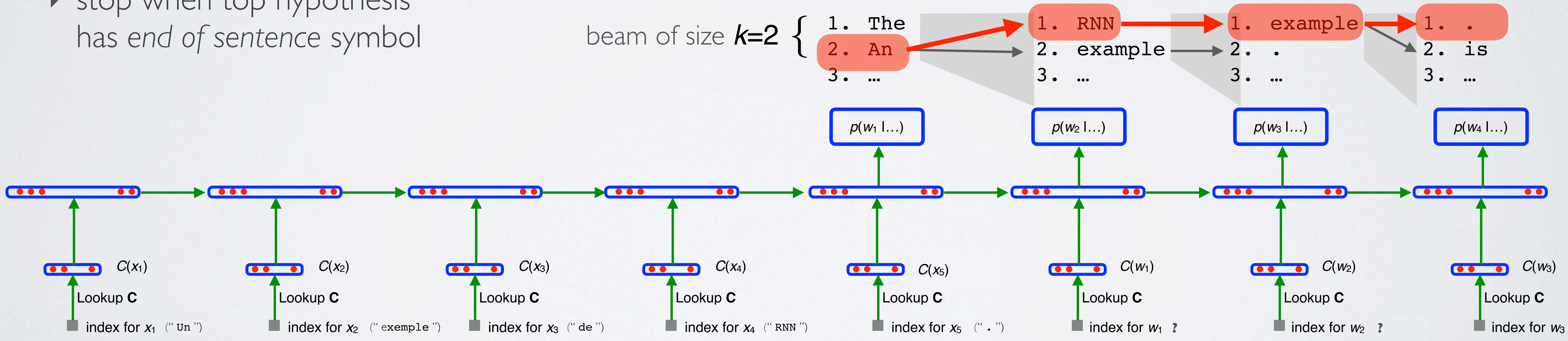
- At test time, must find $\operatorname{argmax} p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol



SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

- At test time, must find $\operatorname{argmax}\, p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain *k* best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
  - stop when top hypothesis has *end of sentence* symbol



SEQUENCE CLASSIFICATION

Topics: sequence classification

- Sequence classification can be seen as special case of Seq2Seq
  - corresponds to case where target sequence has only one word $\mathbf{w}=w_1$
  - w_1 corresponds to the input sequence's label

- RNN models allow us to represent sequences into fixed size sequences

Recurrent neural networks

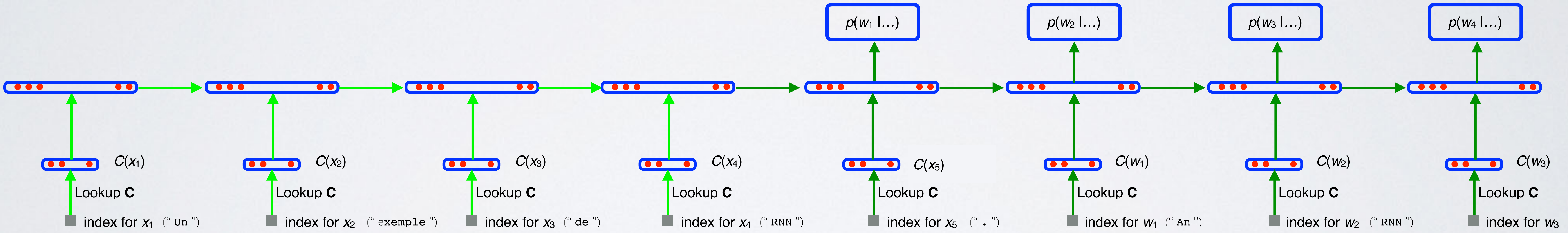
Bidirectional RNN

SEQUENCE TO SEQUENCE LEARNING

REMINDER

Topics: sequence to sequence (Seq2Seq) learning

- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)
    $\mathbf{x}$ = [" `Un` ", " `exemple` ", " `de` ", " `RNN` ", " `.` "] ($T_{\mathbf{x}} = 5$)

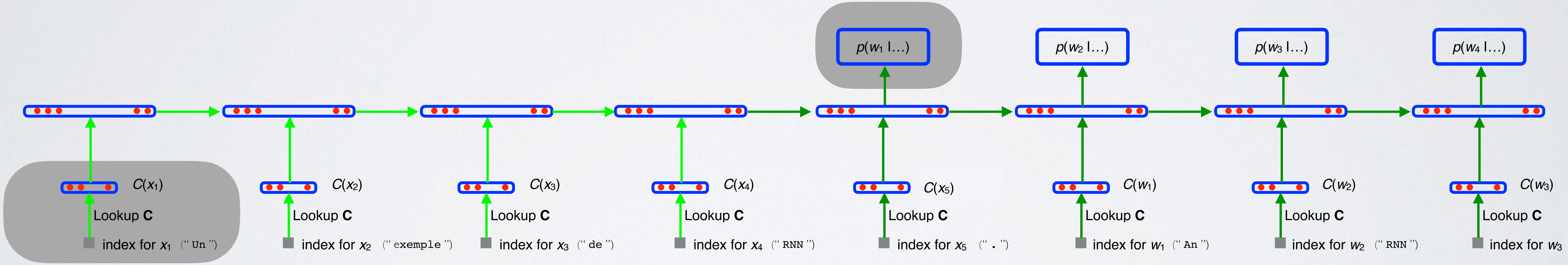


SEQUENCE TO SEQUENCE LEARNING

REMINDER

Topics: sequence to sequence (Seq2Seq) learning

- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)
    $\mathbf{x}$ = [" `Un` ", " `exemple` ", " `de` ", " `RNN` ", " `.` "] ($T_{\mathbf{x}} = 5$)



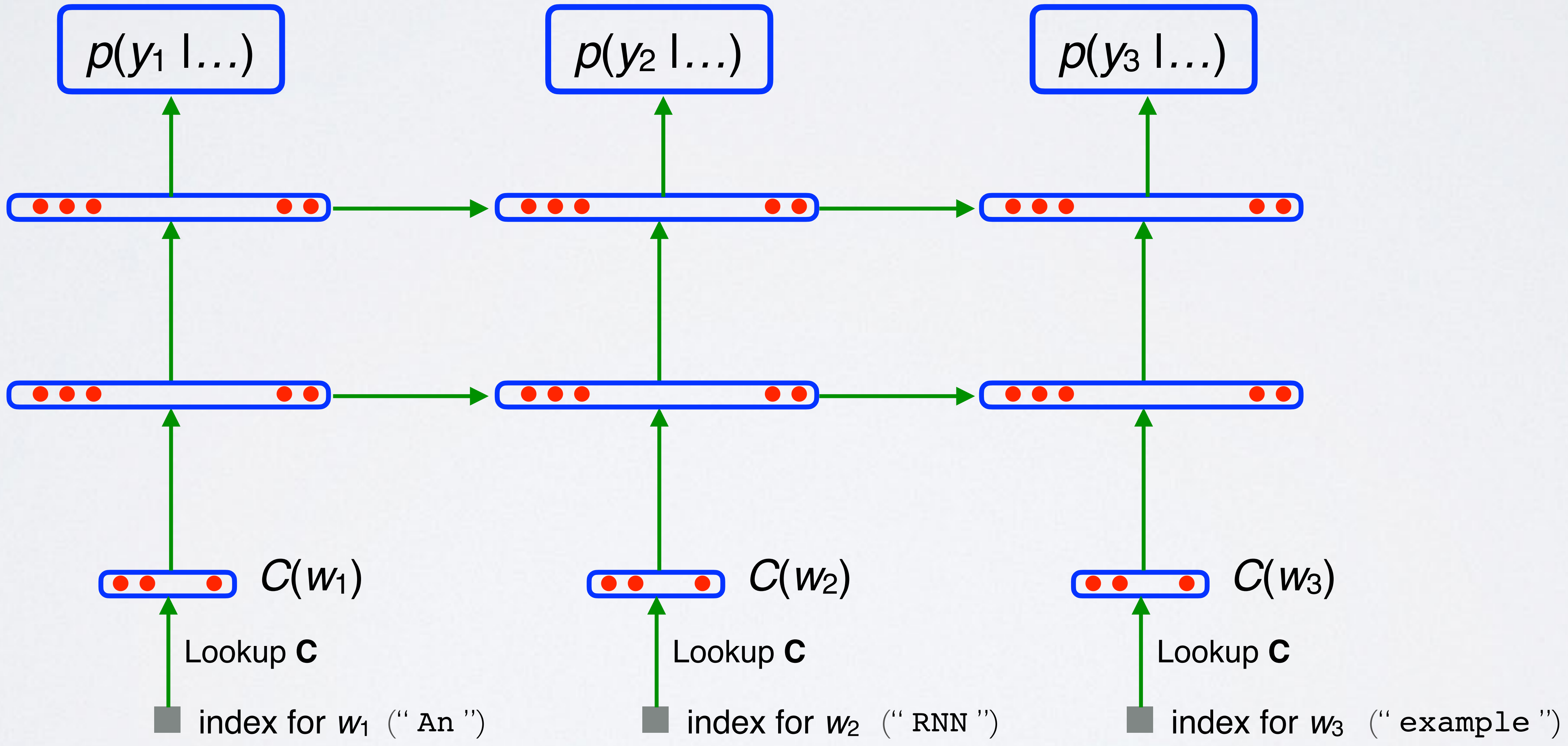
Capturing long-term dependencies is crucial

DEEP RECURRENT NEURAL NETWORK

REMINDER

Topics: Deep RNN

- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)



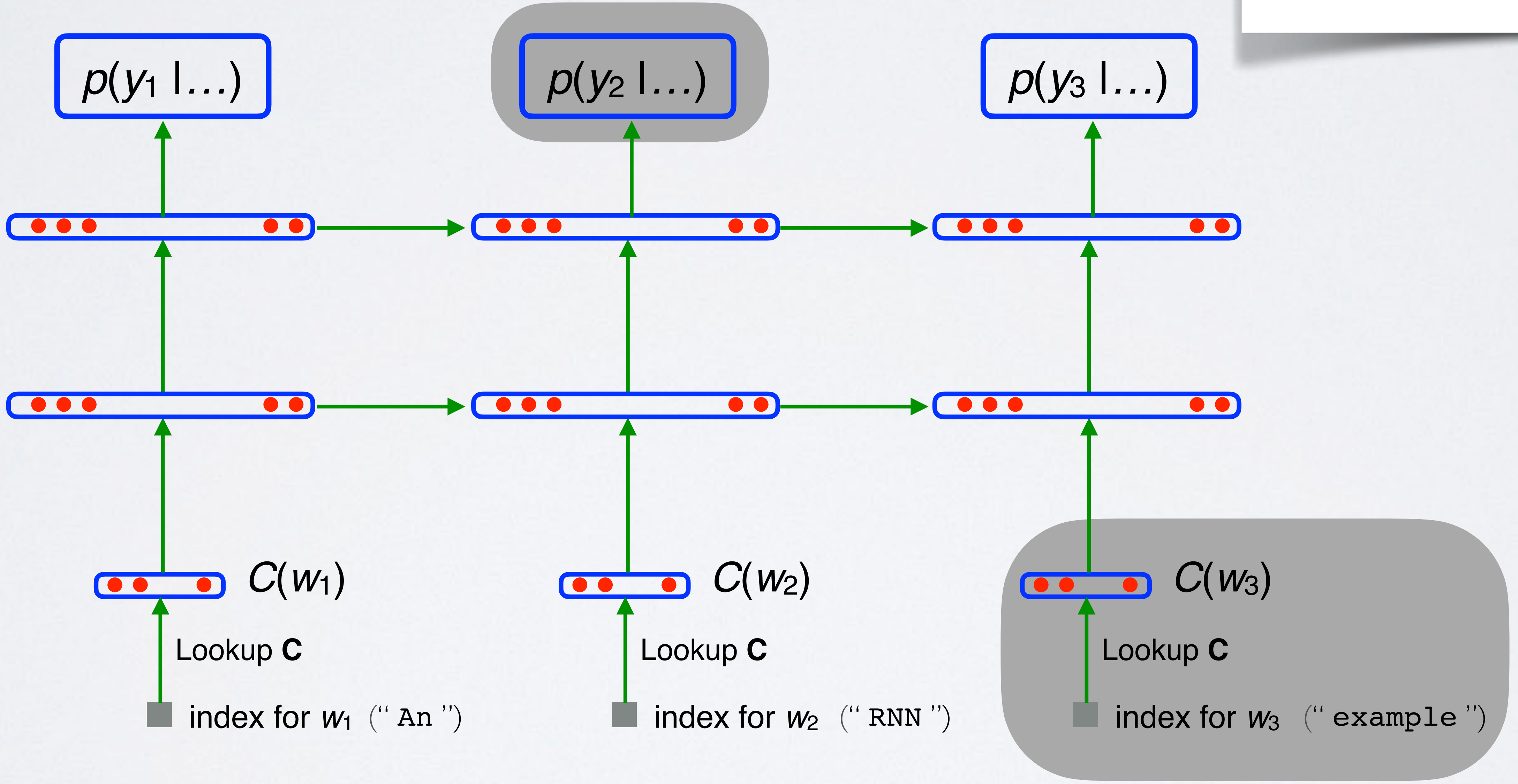
DEEP RECURRENT NEURAL NETWORK

REMINDER

Topics: Deep RNN

- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)

Cannot use information at following time steps



BIDIRECTIONAL RNNS

Topics: bidirectional RNNs

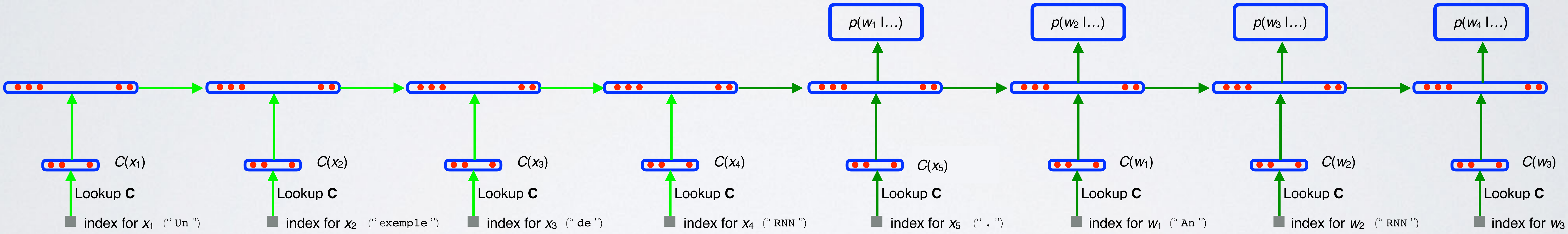
- When conditioning on a full input sequence, no obligation to only traverse left-to-right

- Bidirectional RNNs exploit this observation
  - have one RNNs traverse the sequence left-to-right
  - have *another* RNN traverse the sequence right-to-left
  - use concatenation of hidden layers as representation

SEQUENCE TO SEQUENCE LEARNING

Topics: bidirectional Seq2Seq

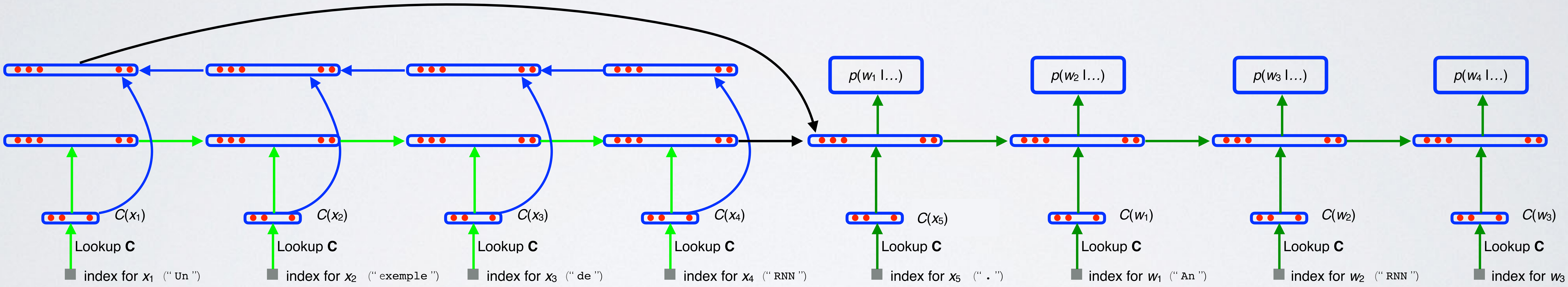
- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [“ `An` ”, “ `RNN` ”, “ `example` ”, “ `.` ”] ($T = 4$)
    $\mathbf{x}$ = [“ `Un` ”, “ `exemple` ”, “ `de` ”, “ `RNN` ”, “ `.` ”] ($T_\mathbf{x} = 5$)



SEQUENCE TO SEQUENCE LEARNING

Topics: bidirectional Seq2Seq

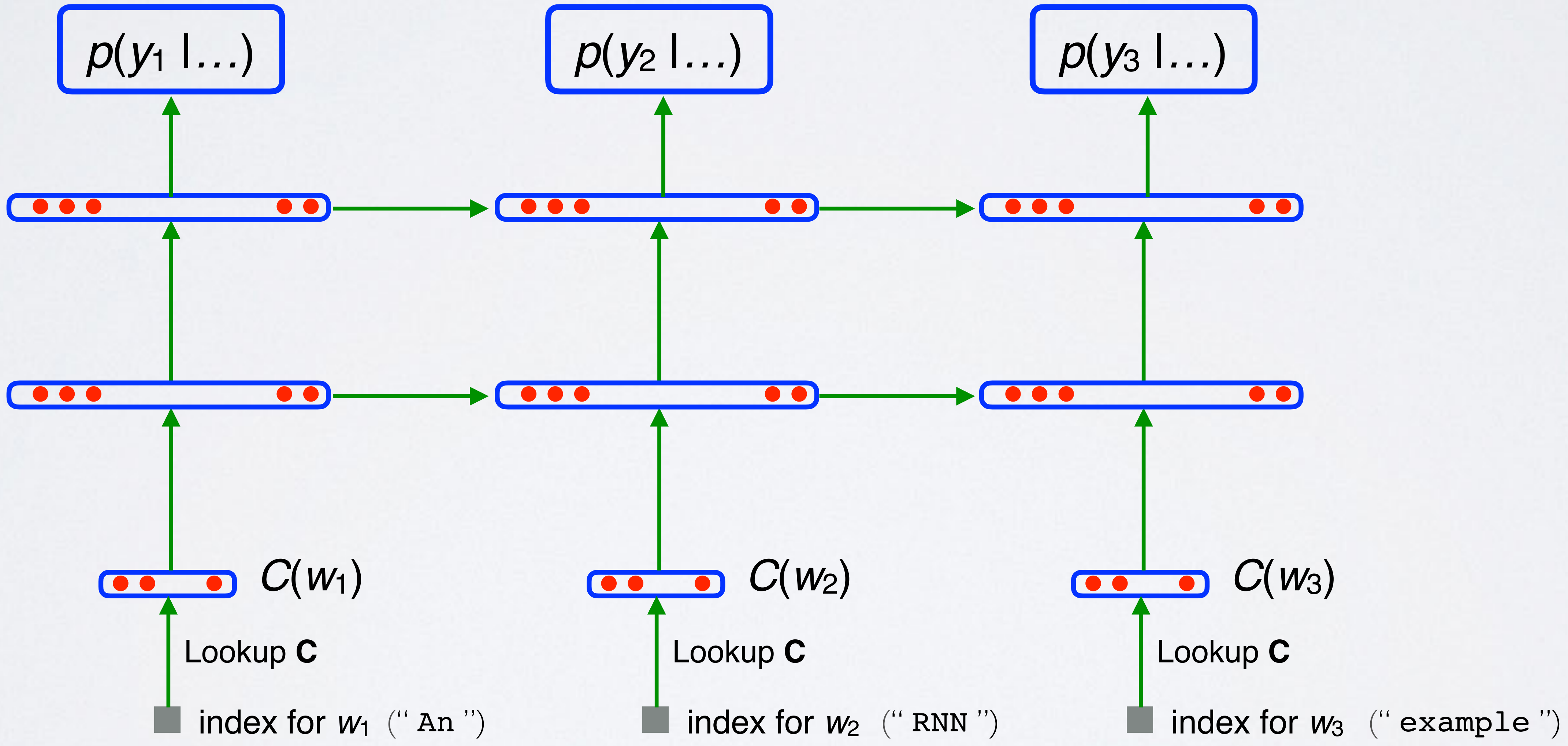
- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [" `An` ", " `RNN` ", " `example` ", " `.` "] ($T = 4$)
    $\mathbf{x}$ = [" `Un` ", " `exemple` ", " `de` ", " `RNN` ", " `.` "] ($T_{\mathbf{x}} = 5$)



DEEP RECURRENT NEURAL NETWORK

Topics: Bidirectional deep RNN

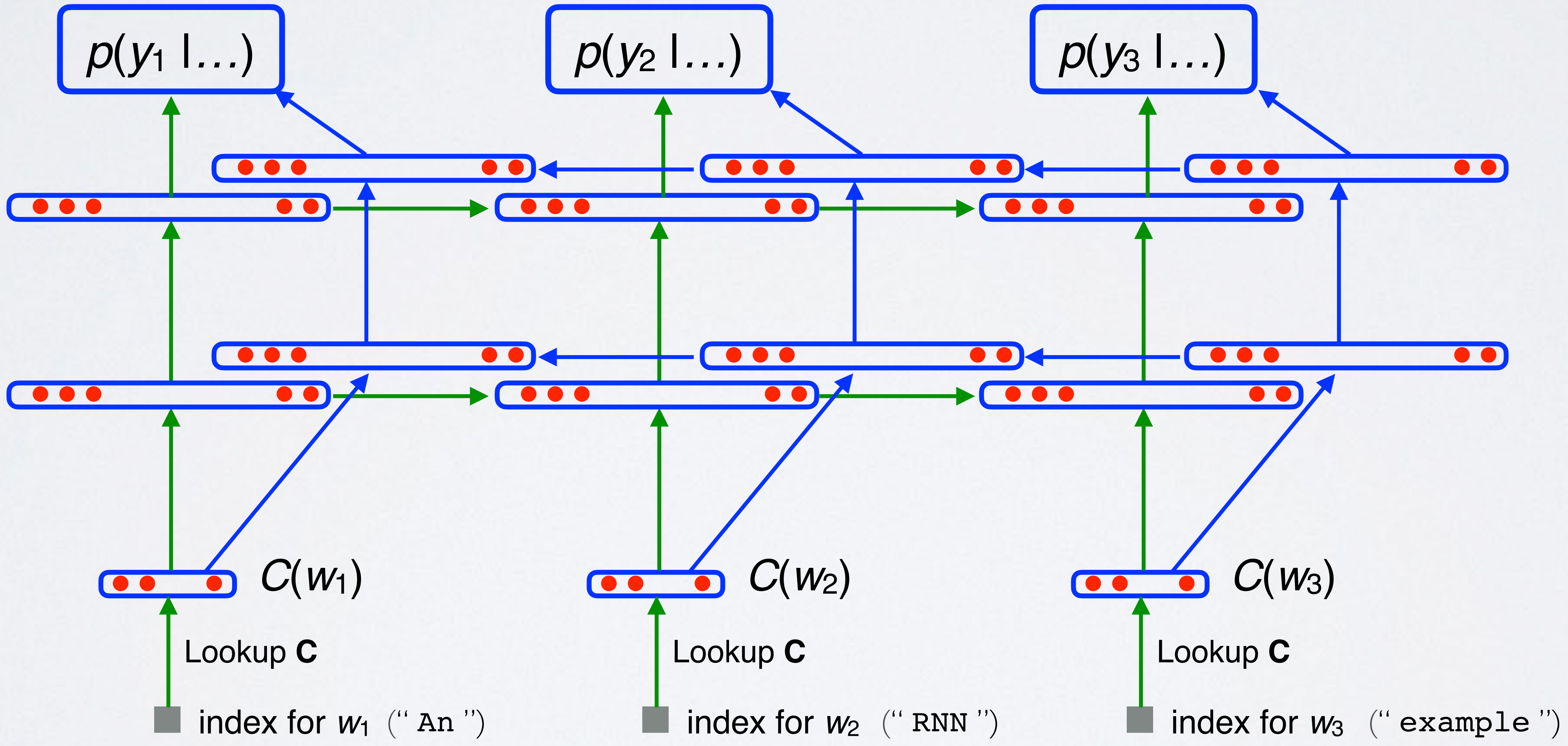
- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)



DEEP RECURRENT NEURAL NETWORK

Topics: Bidirectional deep RNN

- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)



DEEP RECURRENT NEURAL NETWORK

Topics: Bidirectional deep RNN

- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)