Recurrent neural networks

Slides: H. Larochelle, J. Pineau (McGill COMP-551), F. Li & J. Johnson & S. Yeung (Stanford CS231)

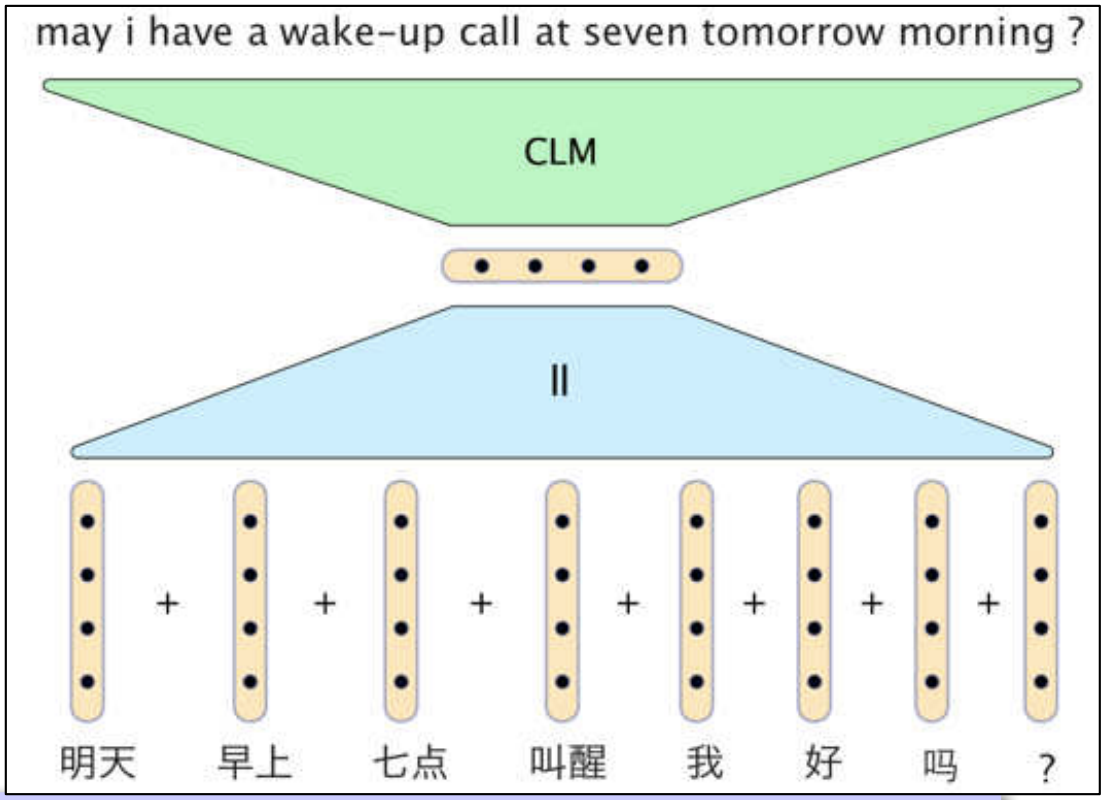
Time series/sequence data

- Very important in practice:
  - Speech recognition
  - Text processing (taking into account the sequence of words)
  - DNA analysis
  - Heart-rate monitoring
  - Financial market forecasting
  - Mobile robot sensor processing
  - ...
- Does this fit the machine learning paradigm as described so far?
  - The sequences are *not all the same length* (so we cannot just assume one attribute per time step)
  - The data at each time slice/index is *not independent*
  - The data distribution *may change over time*

Neural models for sequences

- Several datasets contain sequences of data (e.g. time-series, text)
- Bag-of-words assumption looses the ordering information.

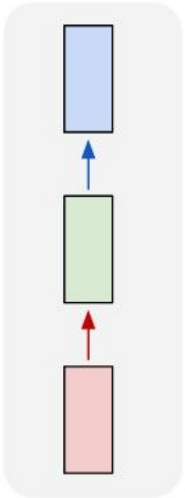
- E.g. Machine translation



From Phil Blumson's slides:

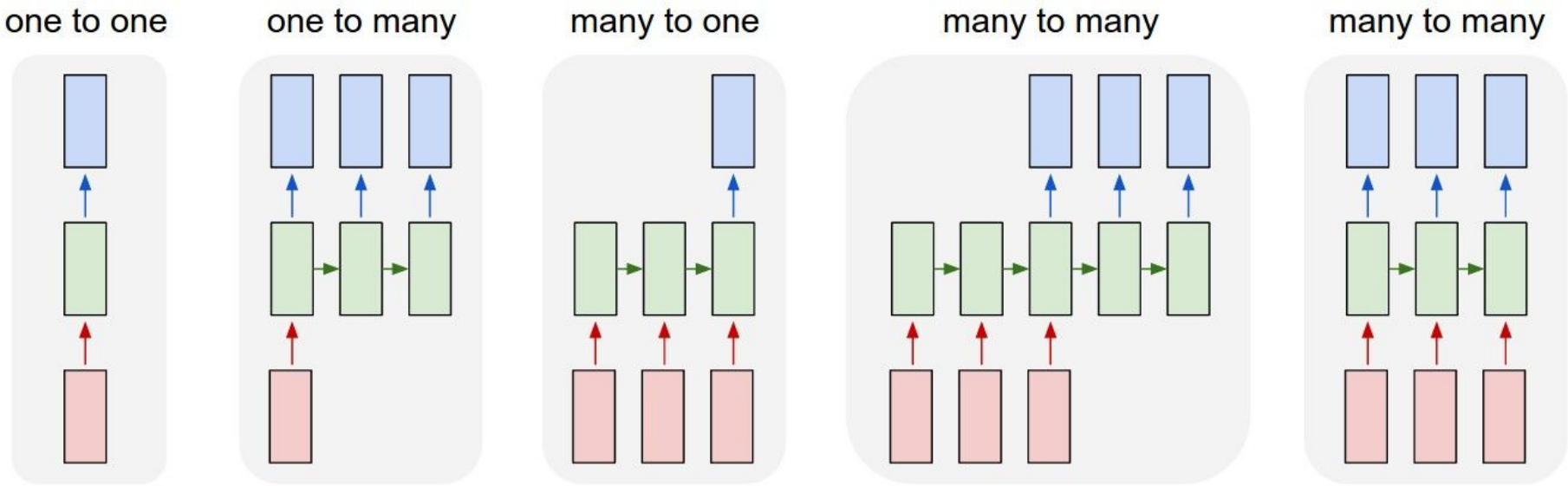
“Vanilla” Neural Network

one to one



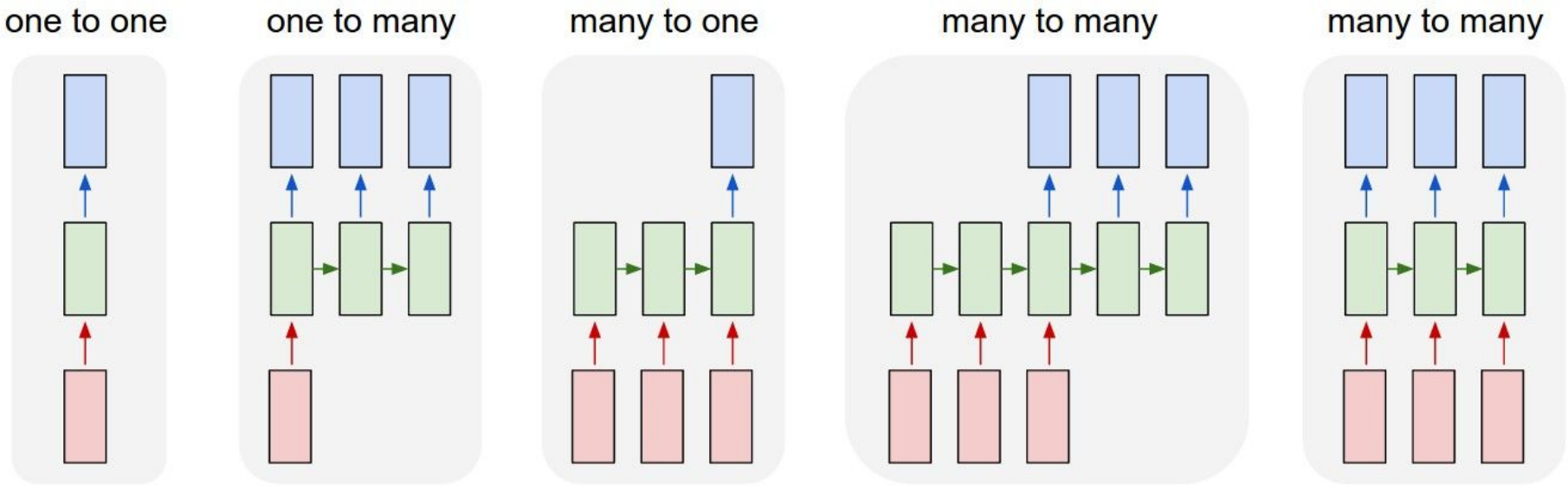
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



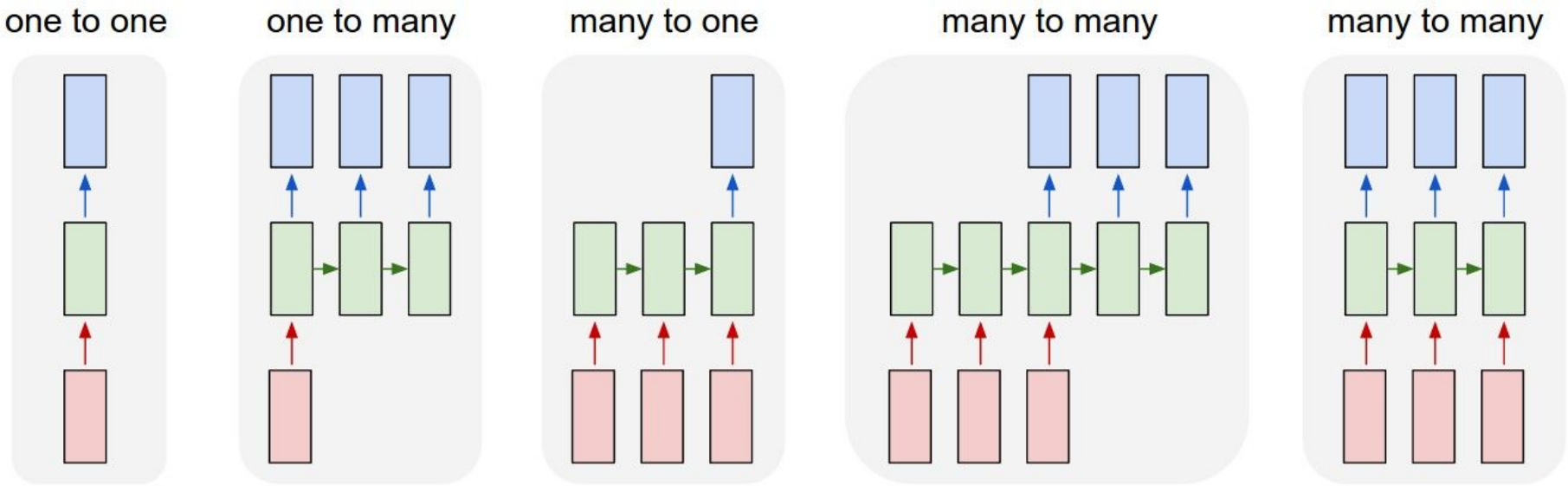
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



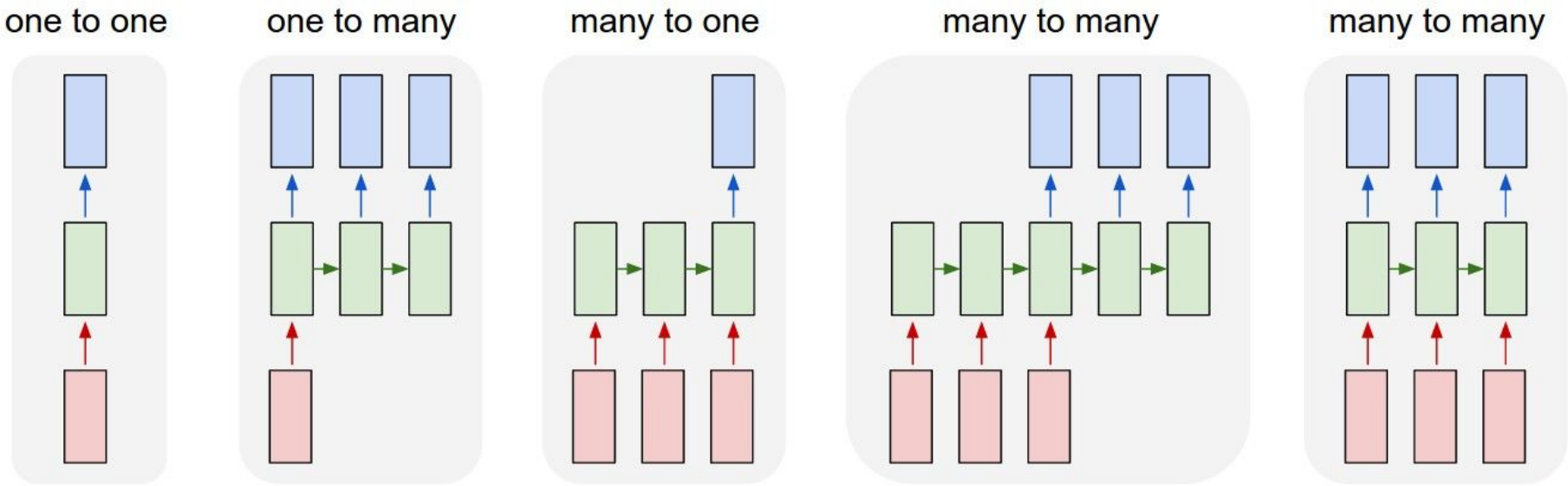
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Neural Networks: Process Sequences



e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Neural Networks: Process Sequences

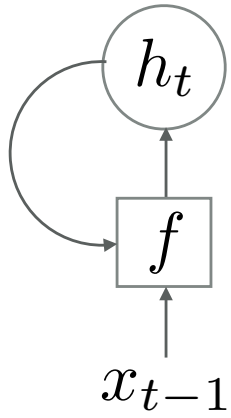


e.g. **Video classification on frame level**

Language modelling from recursion

- Directly model the conditional probabilities.

$$p(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



- Recursive Construction:

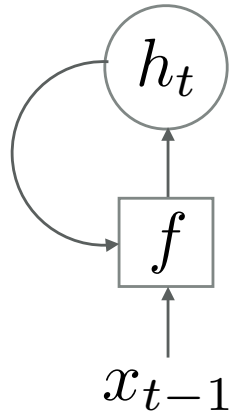
Initial Condition: $h_0 = 0$

Recursion: $h_t = f(x_{t-1}, h_{t-1})$

Language modelling from recursion

- Directly model the conditional probabilities.

$$p(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



- Recursive Construction:

Initial Condition: $h_0 = 0$

Recursion: $h_t = f(x_{t-1}, h_{t-1})$

Example: $p(\text{eating}|\text{the, cat, is})$

(1) Initialization: $h_0 = 0$

(2) Recursion

(1) $h_1 = f(h_0, \text{the})$

(2) $h_2 = f(h_1, \text{cat})$

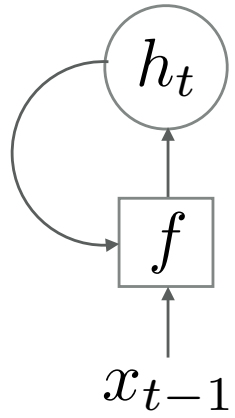
(3) $h_3 = f(h_2, \text{is})$

(3) Readout: $p(\text{eating}|\text{the, cat, is}) = g(h_3)$

Language modelling from recursion

- Directly model the conditional probabilities.

$$p(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



- Recursive Construction:

Initial Condition: $h_0 = 0$

Recursion: $h_t = f(x_{t-1}, h_{t-1})$

Example: $p(\text{eating}|\text{the, cat, is})$

(1) Initialization: $h_0 = 0$

(2) Recursion

(1) $h_1 = f(h_0, \text{the})$

(2) $h_2 = f(h_1, \text{cat})$

(3) $h_3 = f(h_2, \text{is})$

(3) Readout: $p(\text{eating}|\text{the, cat, is}) = g(h_3)$

We call h_t an internal hidden state,

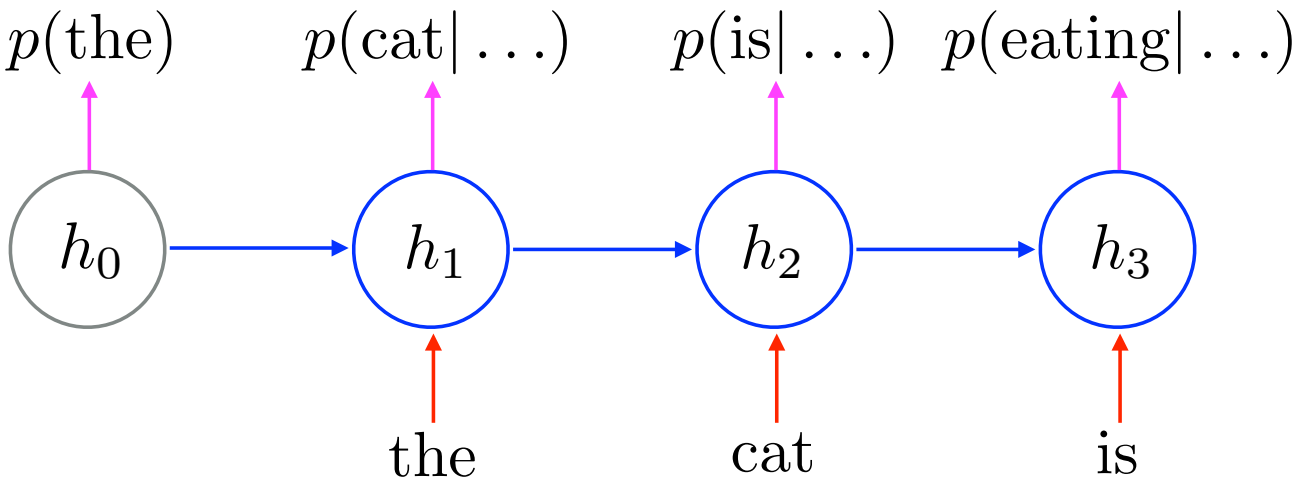
or **memory,** which summarizes history from x_1 up to x_{t-1}.

Recurrent neural language model

Transition Function $h_t = f(h_{t-1}, x_{t-1})$

Output/Readout Function $p(x_t = w|x_1, \ldots, x_{t-1}) = g_w(h_t)$

Example: $p(\text{the}, \text{cat}, \text{is}, \text{eating})$



Training an RNN language model

- Loss function:

Log-Probability of a sentence $(x_1, x_2, \ldots, x_T)$

$$\log p(x_1, x_2, \ldots, x_T) = \sum_{t=1}^{T} \log p(x_t \mid x_1, \ldots, x_{t-1})$$

Training an RNN language model

- Loss function:

Log-Probability of a sentence $(x_1, x_2, \ldots, x_T)$

$$\log p(x_1, x_2, \ldots, x_T) = \sum_{t=1}^{T} \log p(x_t \mid x_1, \ldots, x_{t-1})$$

- Train an RNN LM to maximize the log-prob's of training sentences.

Given a training set of N sentences: $\{(x_1^1, \ldots, x_{T_1}^1), \ldots, (x_1^N, \ldots, x_{T_N}^N)\}$

$$\text{maximize}_\Theta \frac{1}{N} \sum_{n=1}^{N} \log p(x_1^n, \ldots, x_{T_n}^n)$$

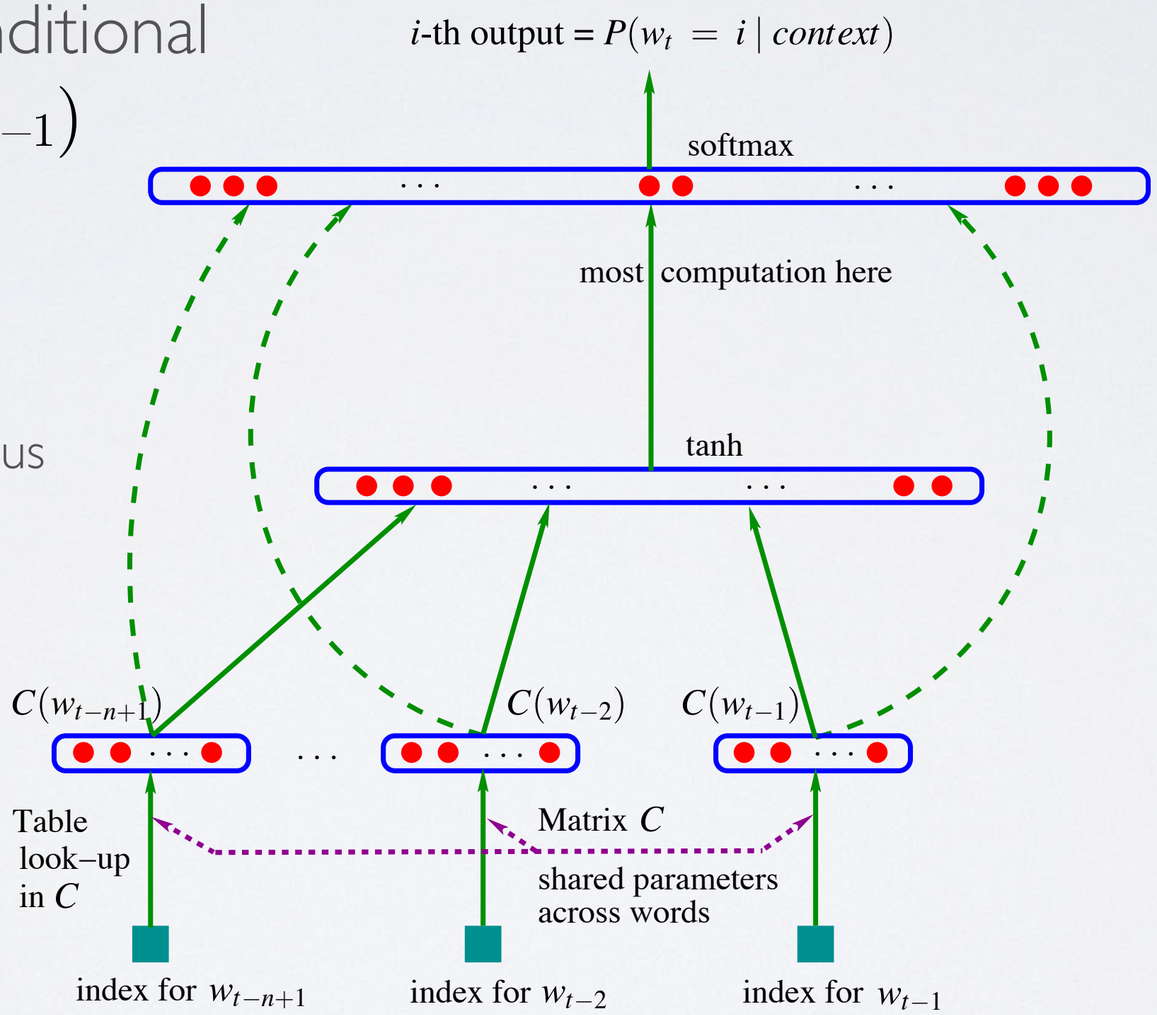
$$\iff \text{minimize}_\Theta J(\Theta) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log p(x_t^n | x_1^n \ldots, x_{t-1}^n)$$

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

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

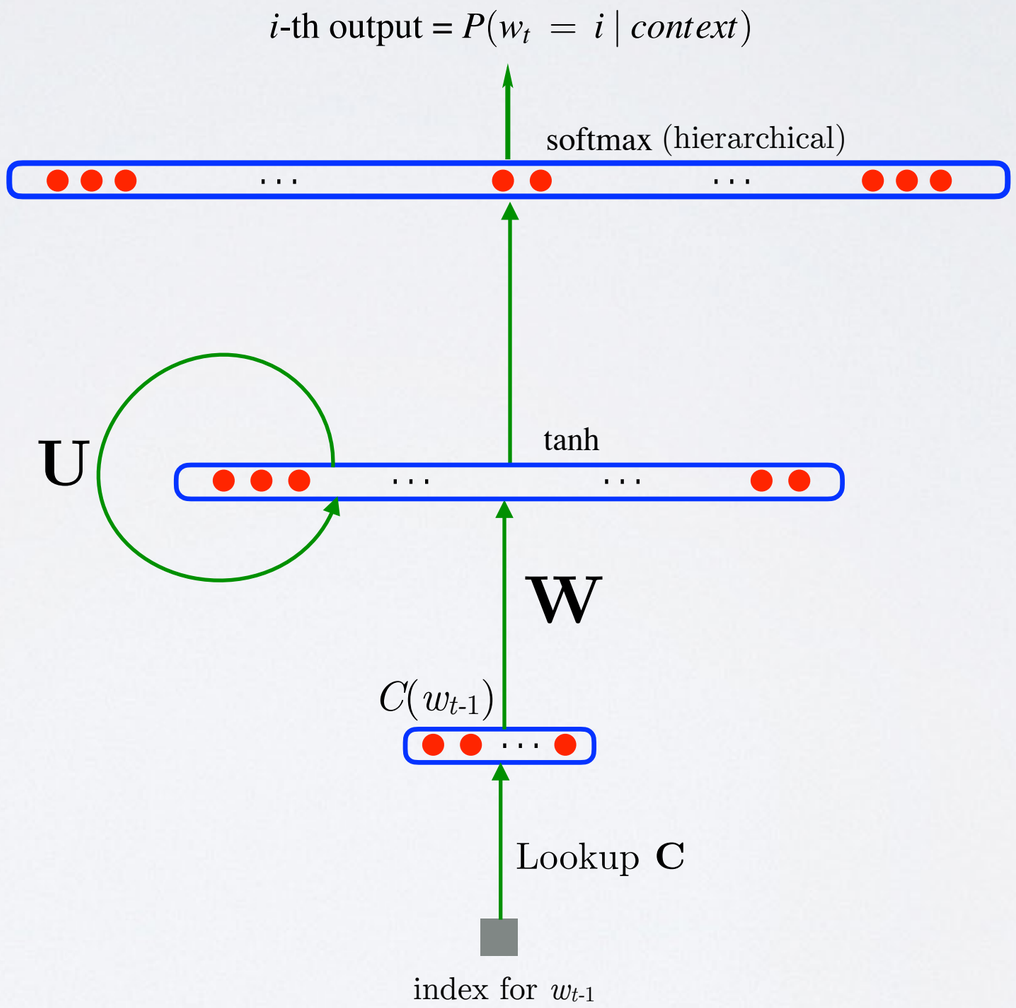
- Solution: recursively update a persistent hidden layer

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

- To compute

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

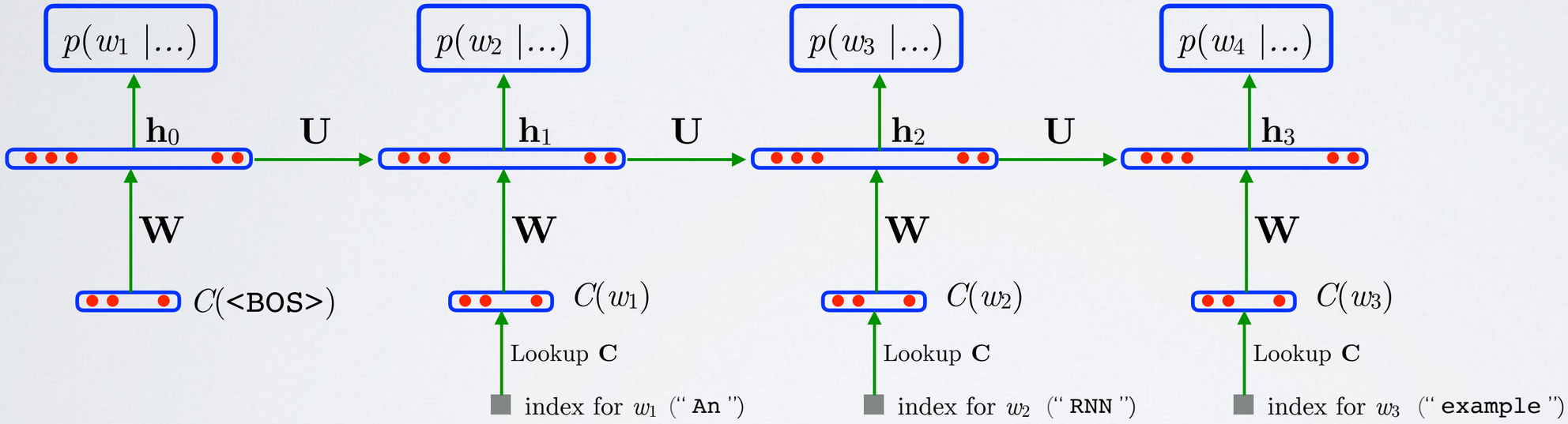
we use hidden layer $\mathbf{h}_{t\text{-}1}$



RECURRENT NEURAL NETWORK (RNN)

Topics: unrolled RNN

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

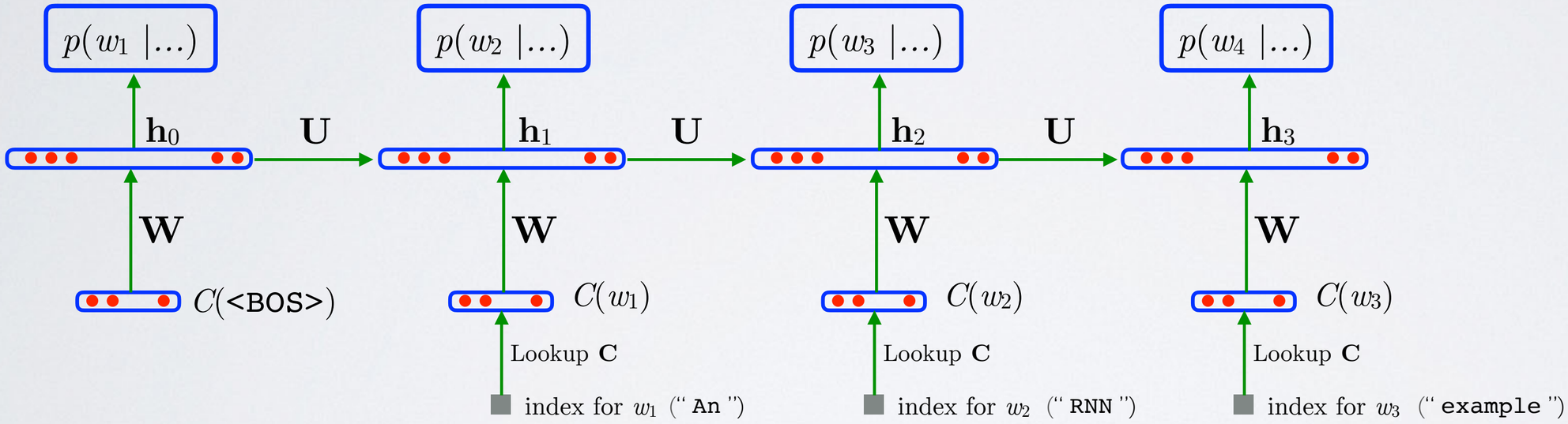


  - symbol “ `.` ” serves as an *end of sentence* symbol
  - $\mathbf{h}_0 = \tanh(\mathbf{b} + \mathbf{W}\, C(\texttt{<BOS>}))$, where $C(\texttt{<BOS>})$ o is a unique embedding for the *beginning of sentence* position(`<BOS>` not included as possible output!)

DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

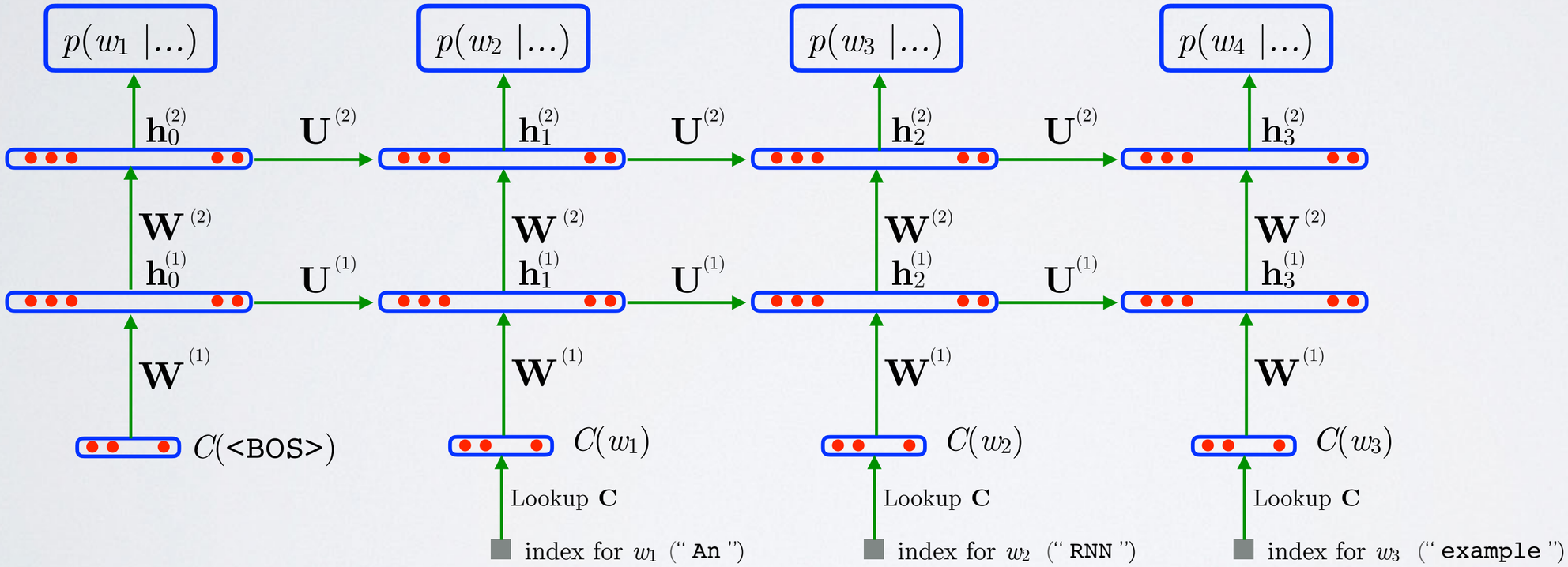
- Straightforward to make deep
  - example: $\mathbf{w} = [\text{“}\texttt{An}\text{”}, \text{“}\texttt{RNN}\text{”}, \text{“}\texttt{example}\text{”}, \text{“}\texttt{.}\text{”}]$ $(T = 4)$



DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

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



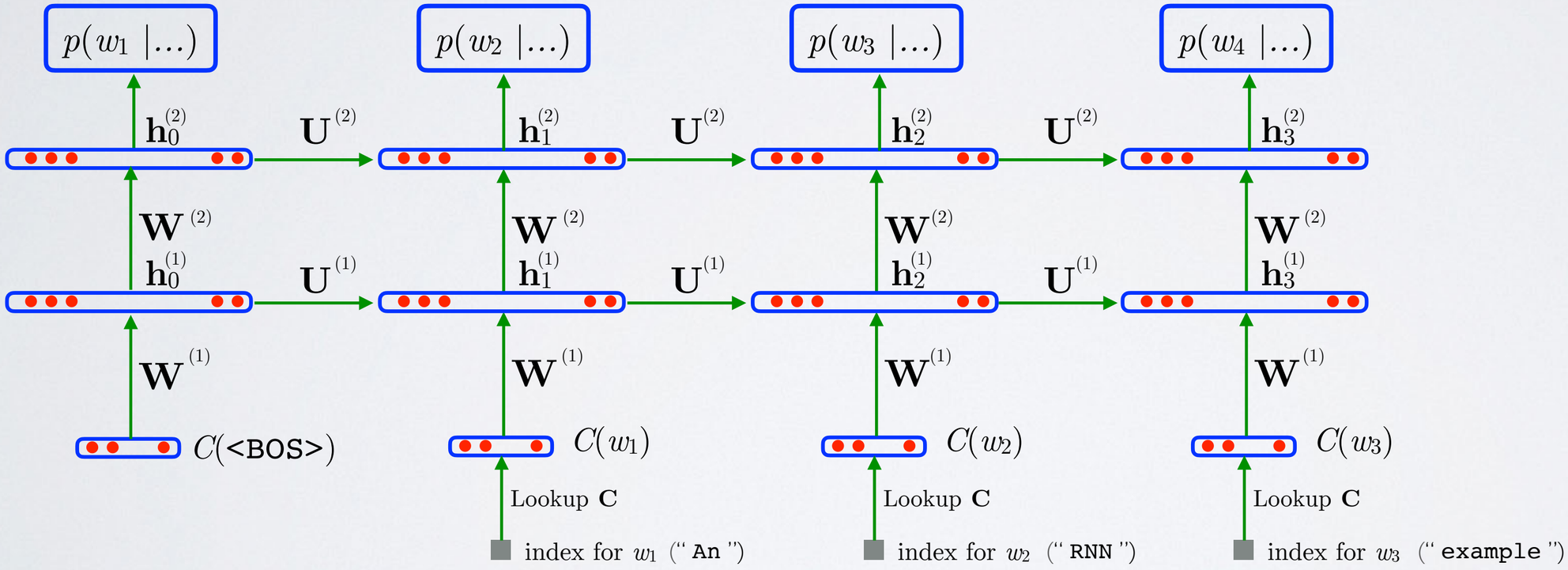
DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

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

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

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



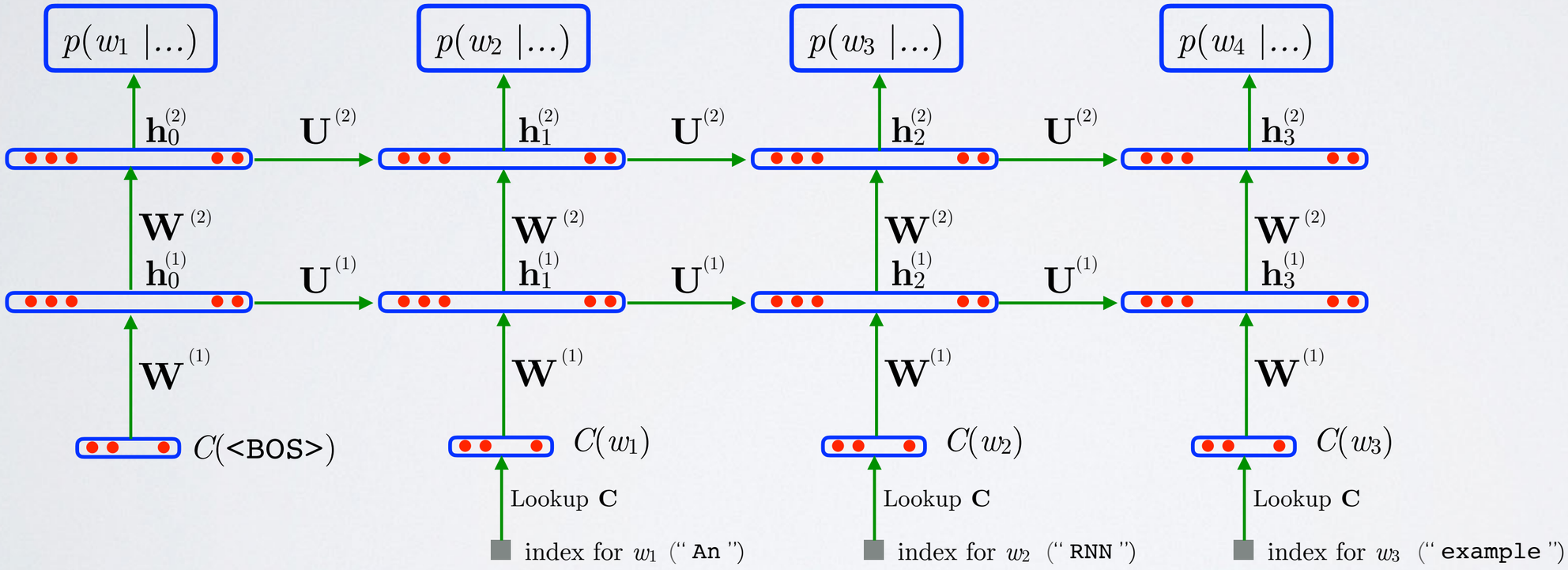
DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

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

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

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



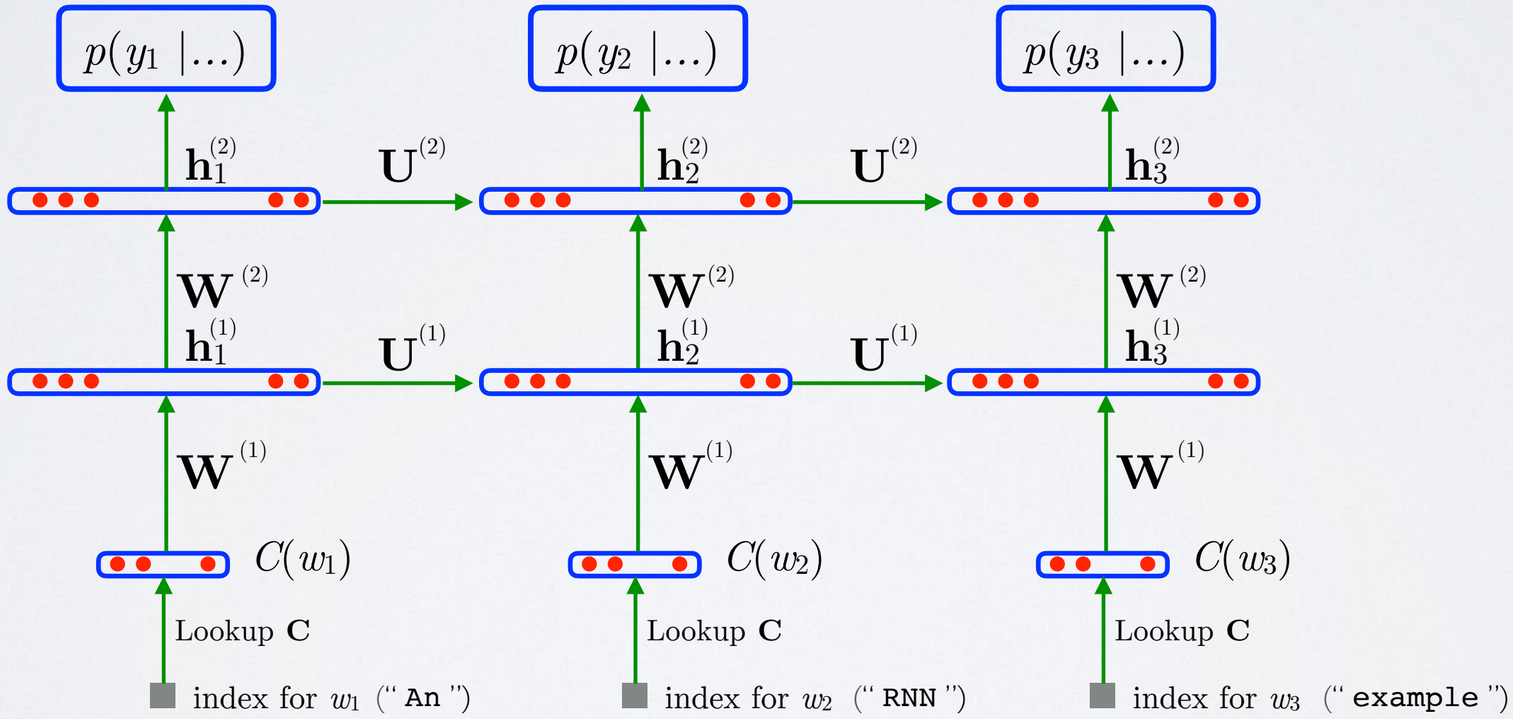
DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

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

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

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



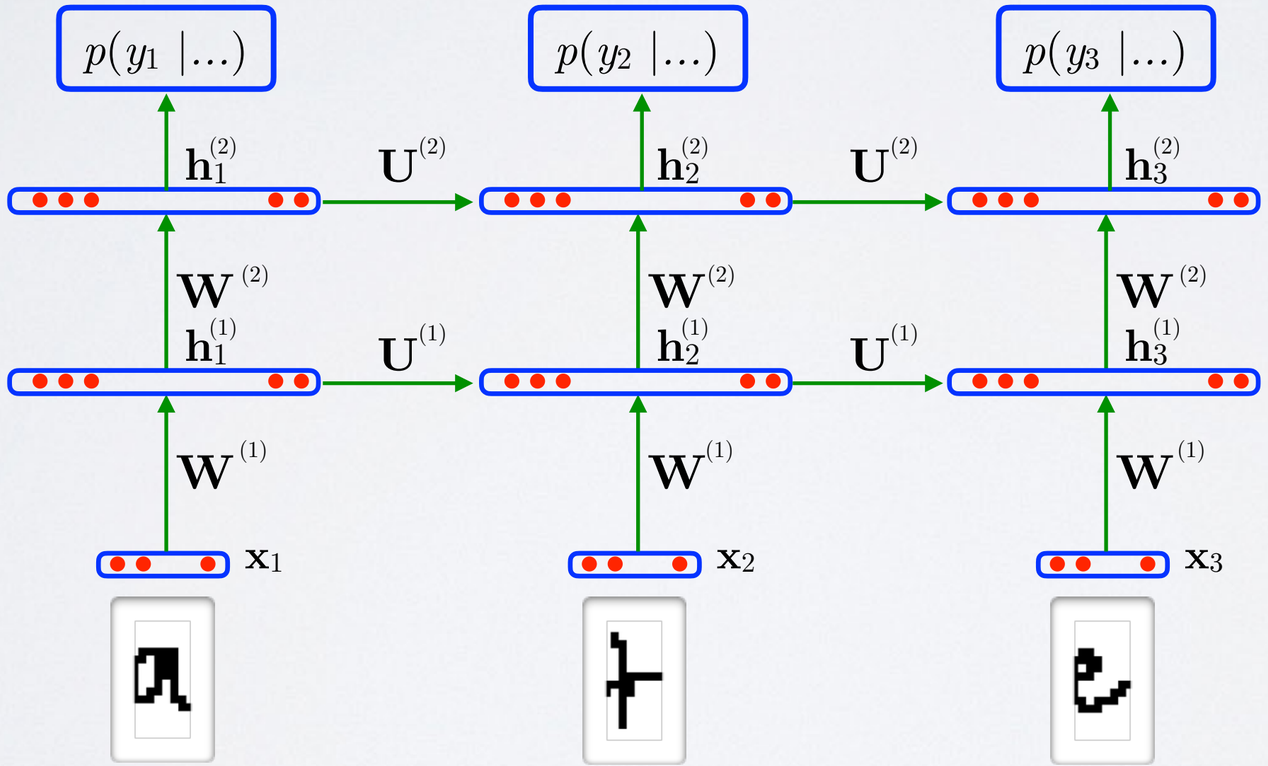
DEEP RECURRENT NEURAL NETWORK

Topics: Deep RNN

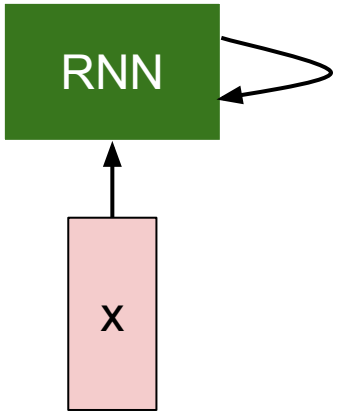
- Useful beyond language modeling
  - sequence labeling in general (e.g. character recognition)

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

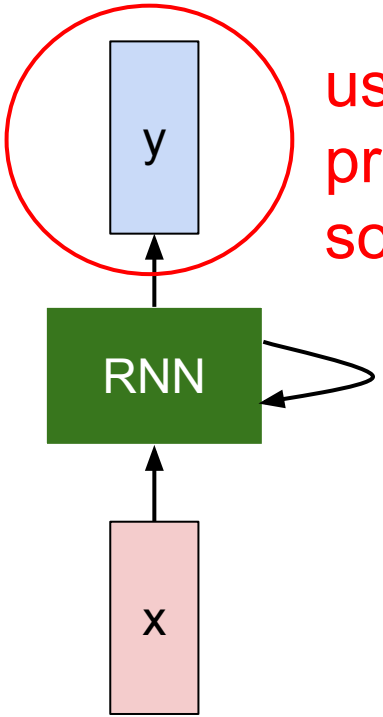
$$\mathbf{h}_t^{(2)} = \tanh(\mathbf{b}^{(2)} + \mathbf{U}^{(2)}\mathbf{h}_{t-1}^{(2)} + \mathbf{W}^{(2)}\mathbf{h}_t^{(1)})$$



Recurrent Neural Network



Recurrent Neural Network



usually want to predict a vector at some time steps

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

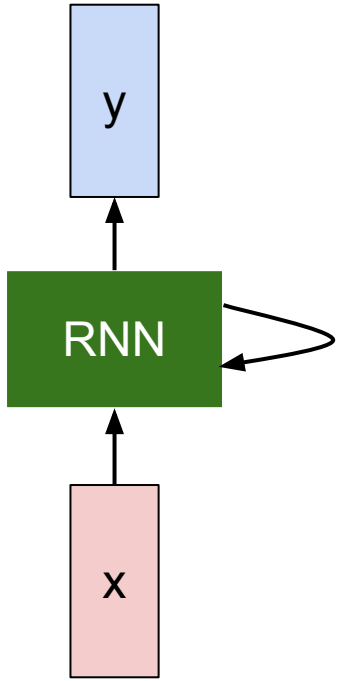
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

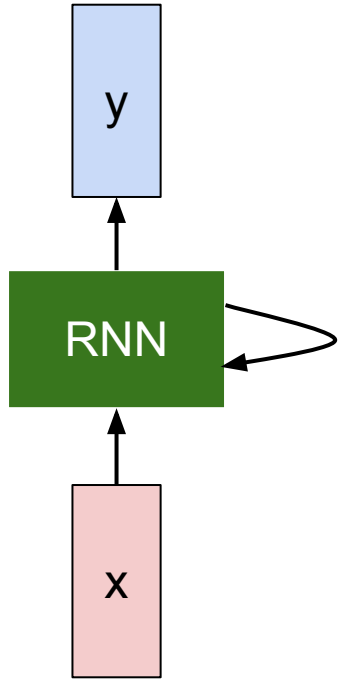
input vector at some time step



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

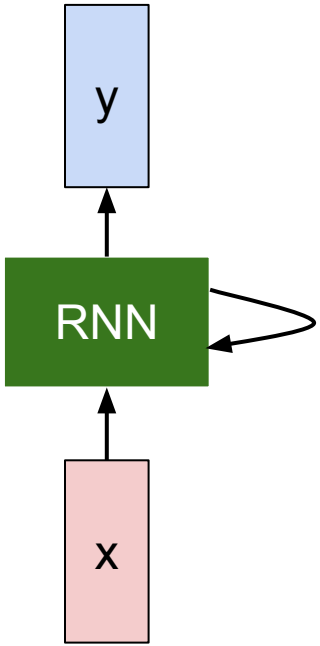
$$h_t = f_W(h_{t-1}, x_t)$$



Notice: the same function and the same set of parameters are used at every time step.

(Simple) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



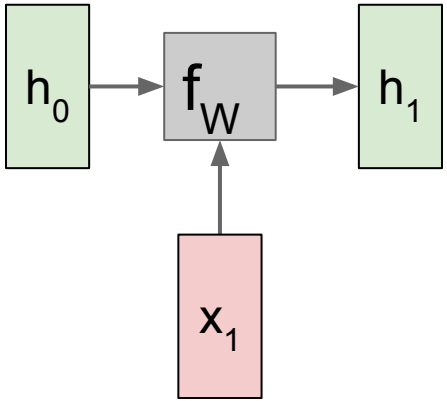
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

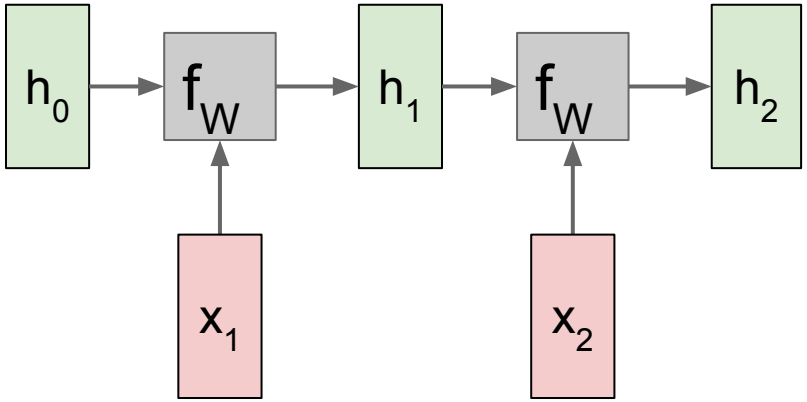
$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

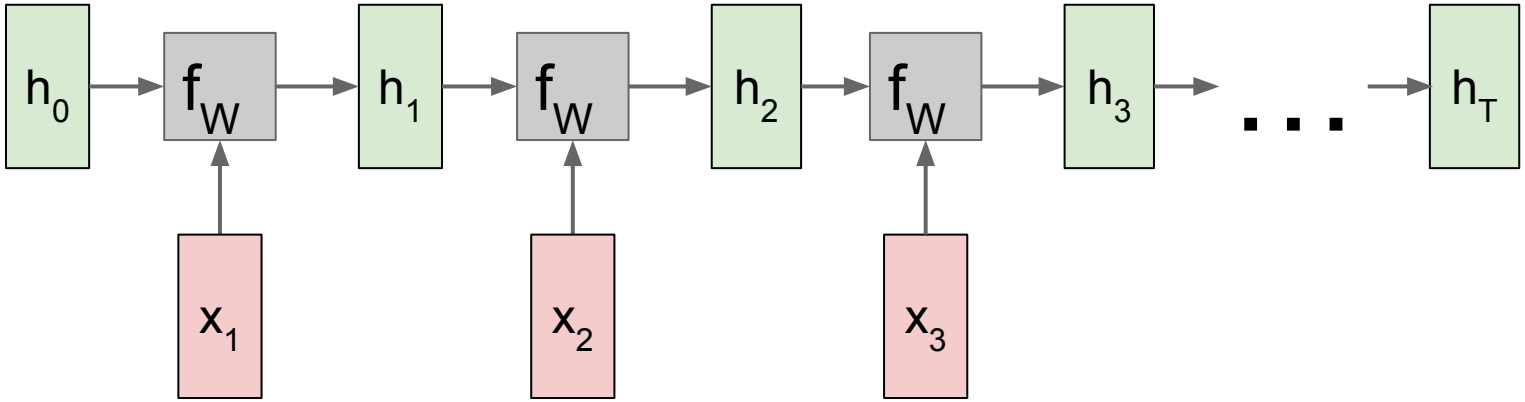
RNN: Computational Graph



RNN: Computational Graph

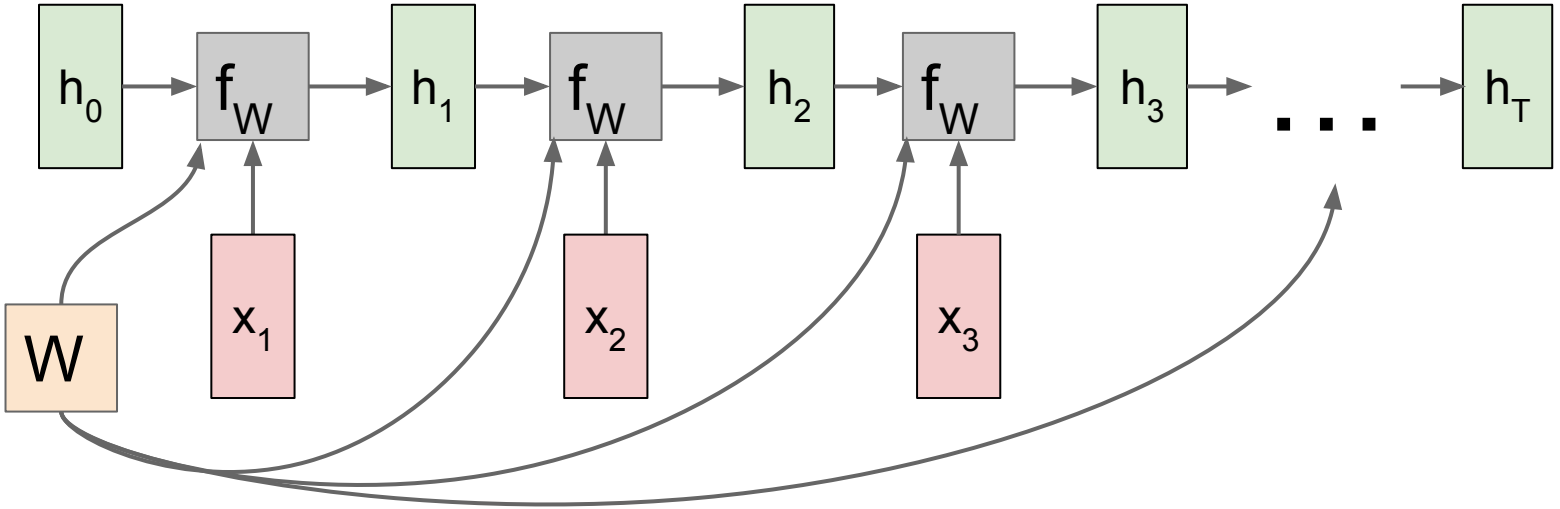


RNN: Computational Graph

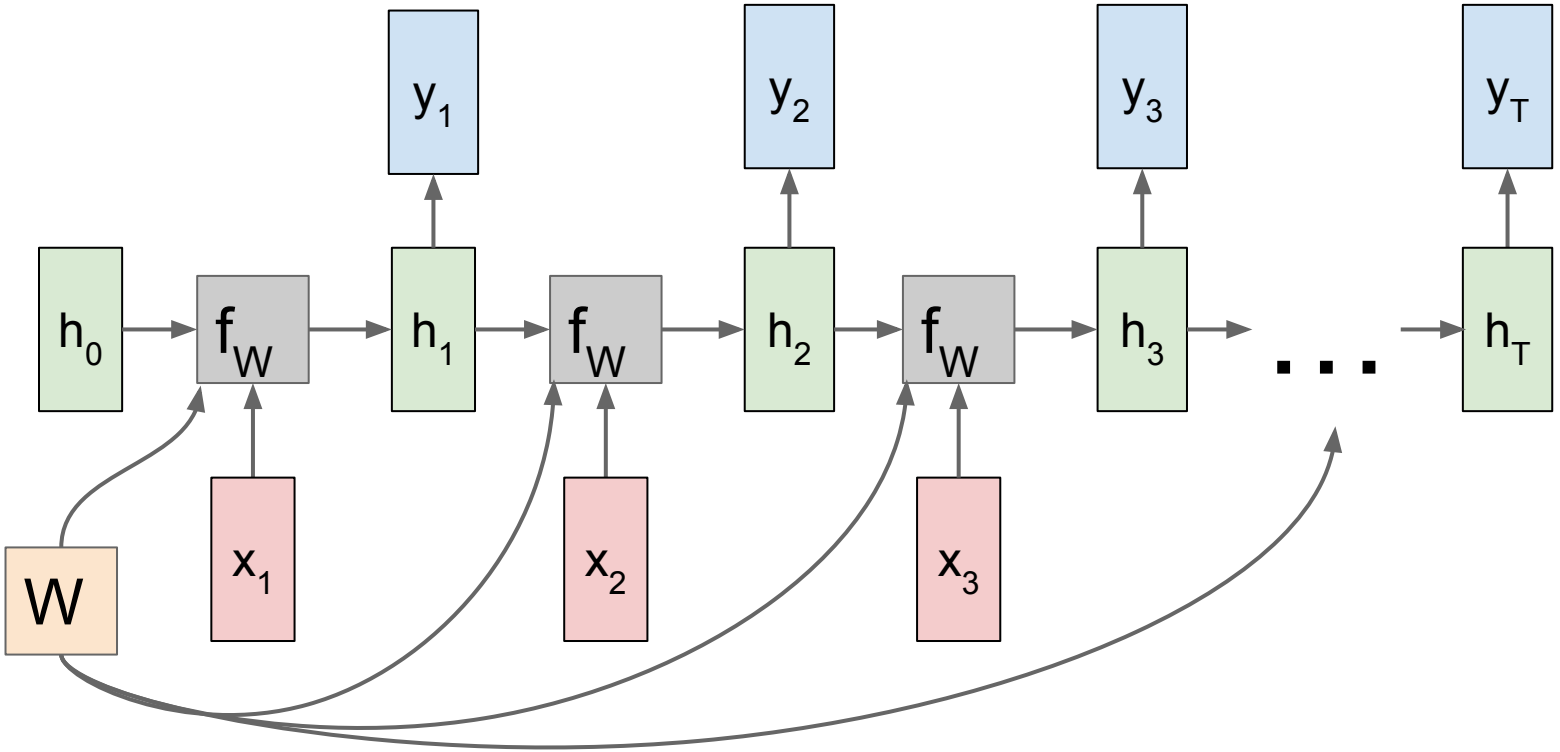


RNN: Computational Graph

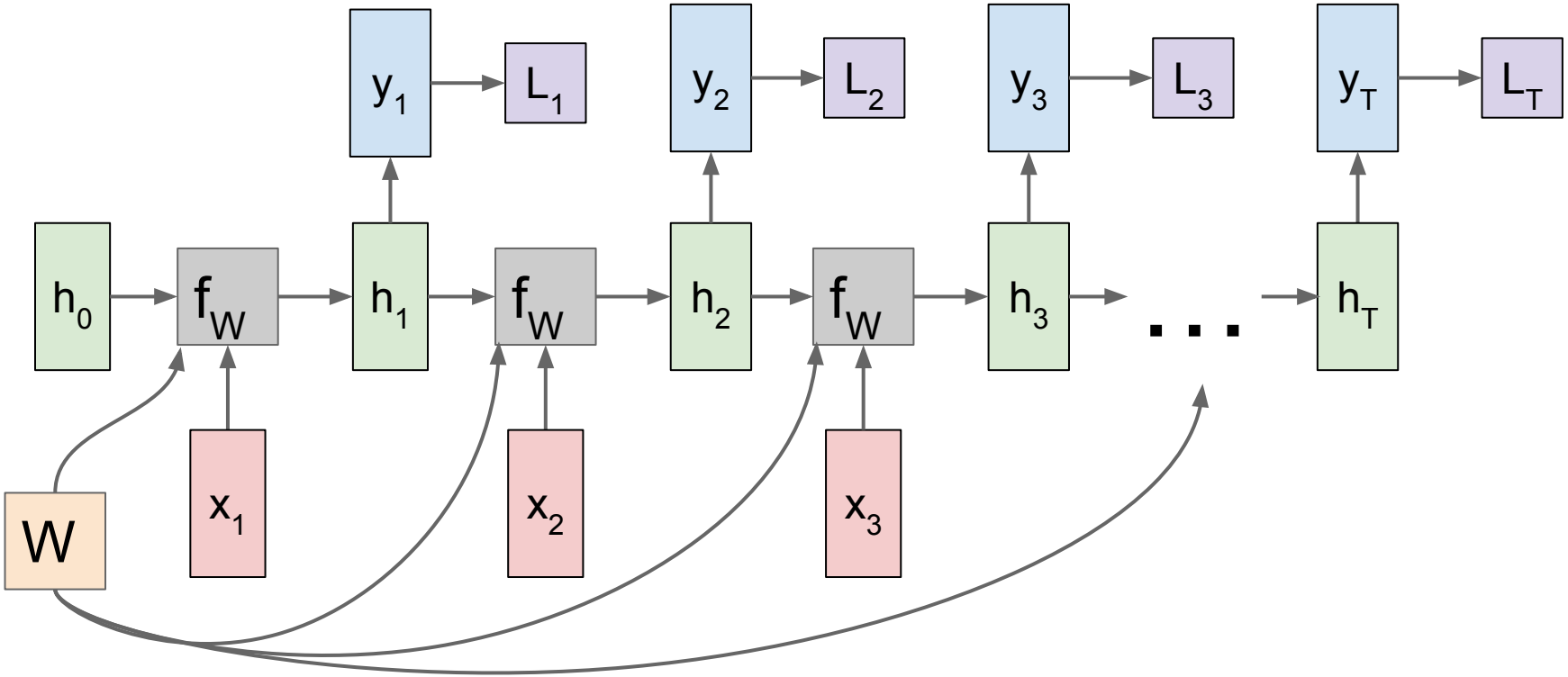
Re-use the same weight matrix at every time-step



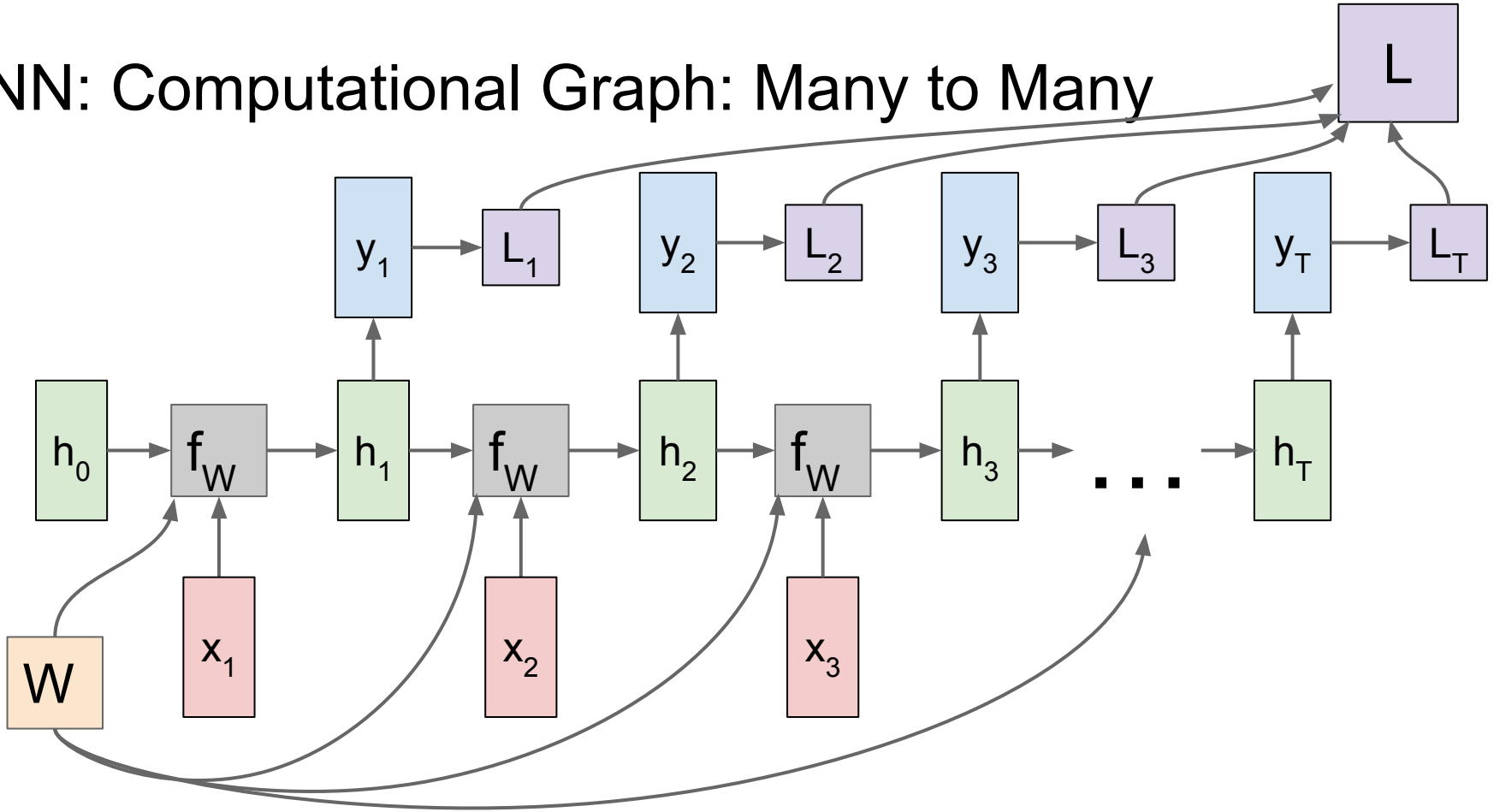
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to Many



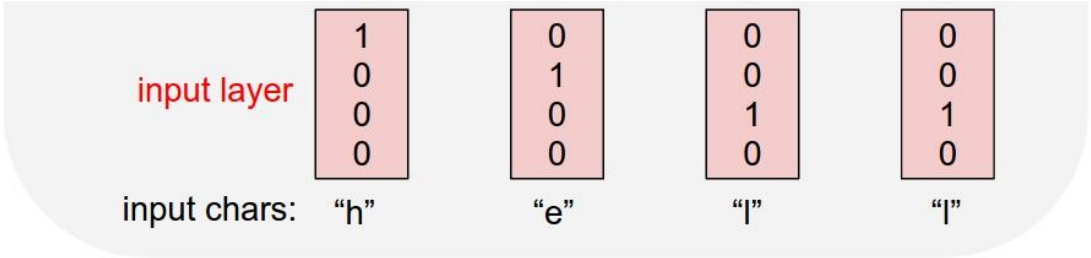
RNN: Computational Graph: Many to Many



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

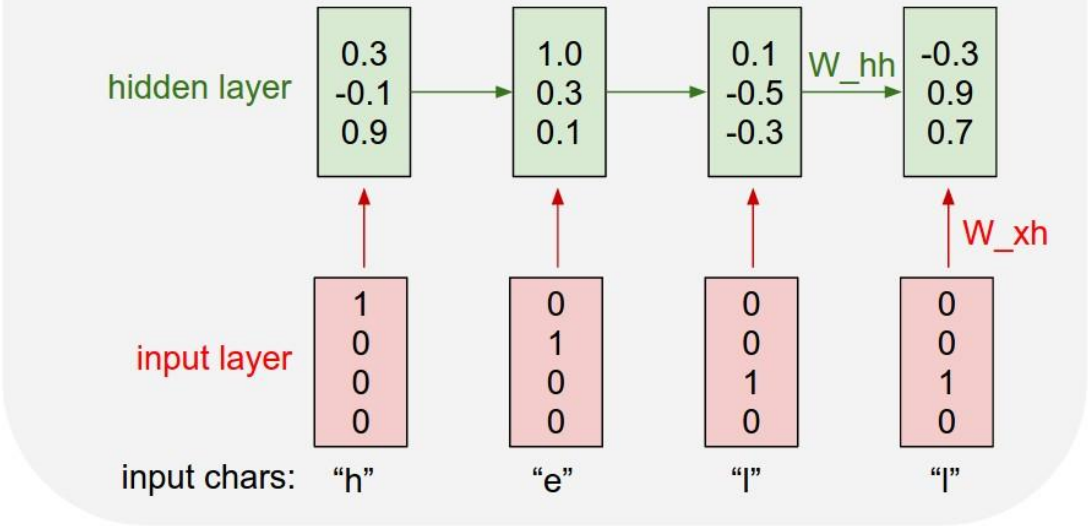


Example: Character-level Language Model

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Vocabulary:
[h,e,l,o]

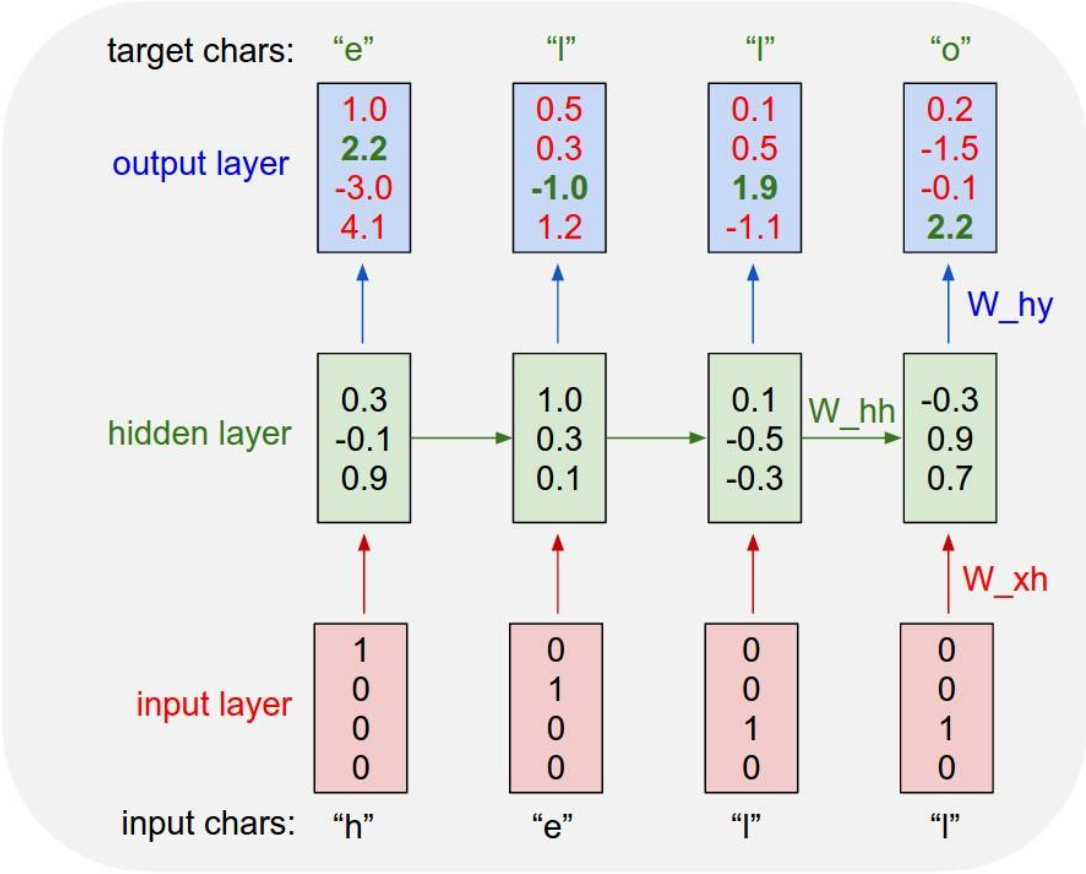
Example training sequence:
“hello”



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

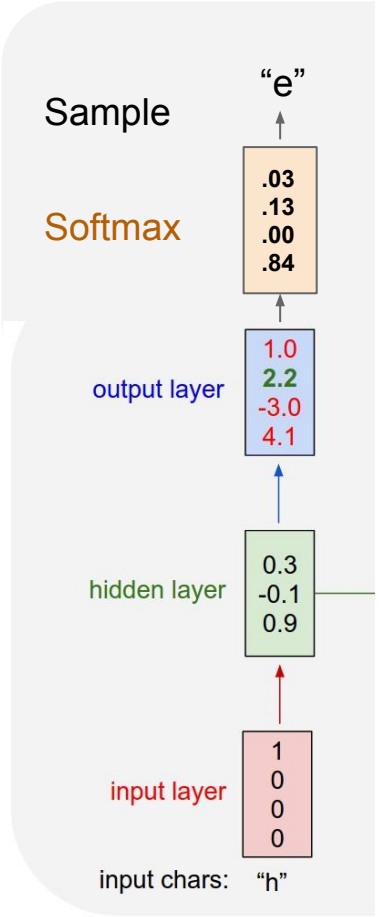
Example training sequence:
“hello”



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

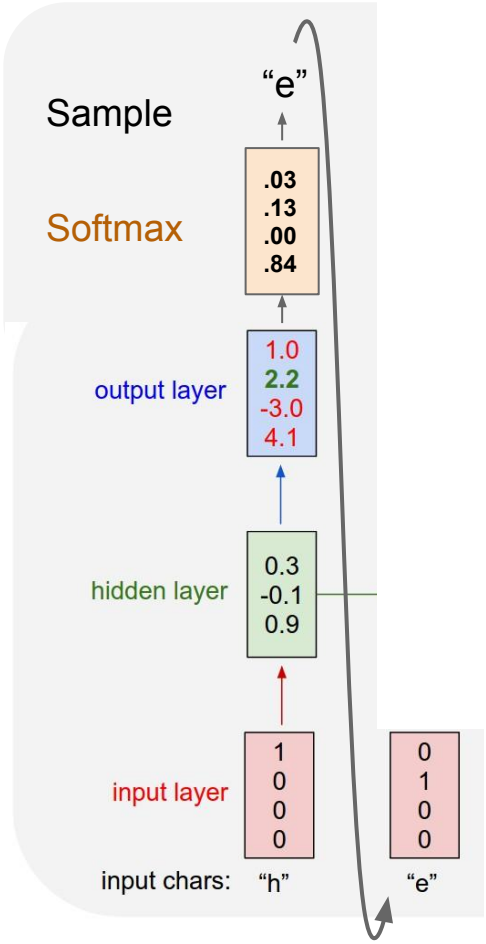
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

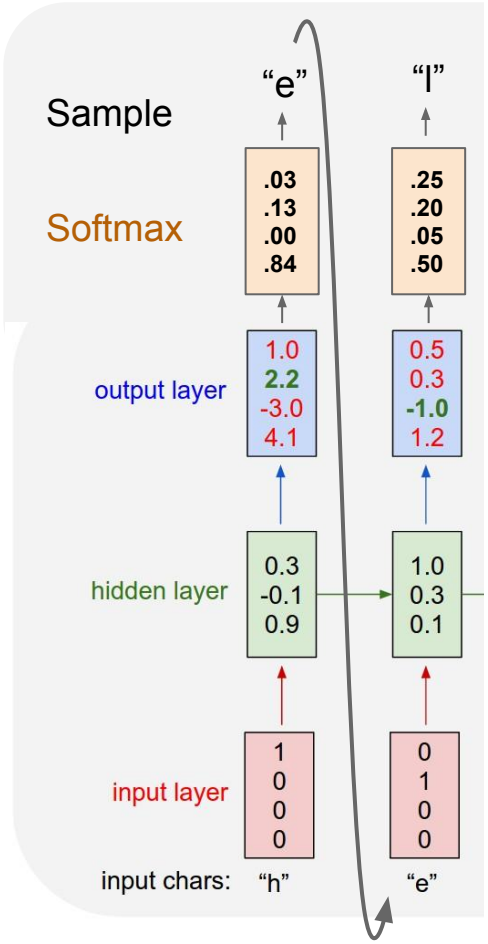
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

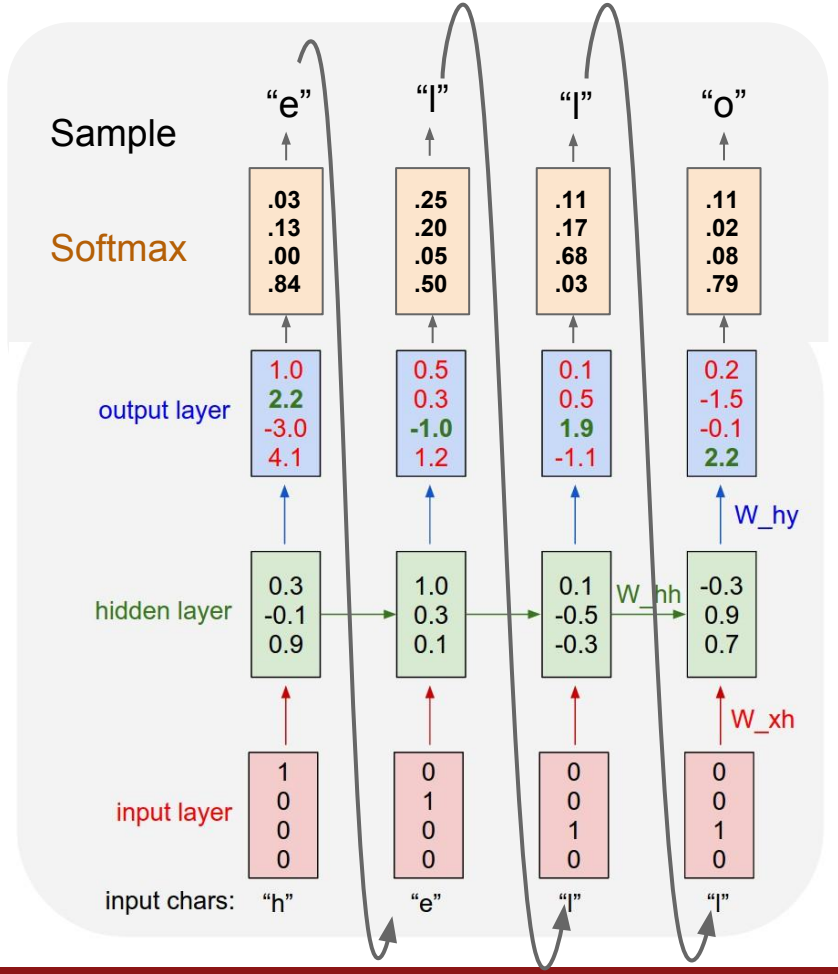
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

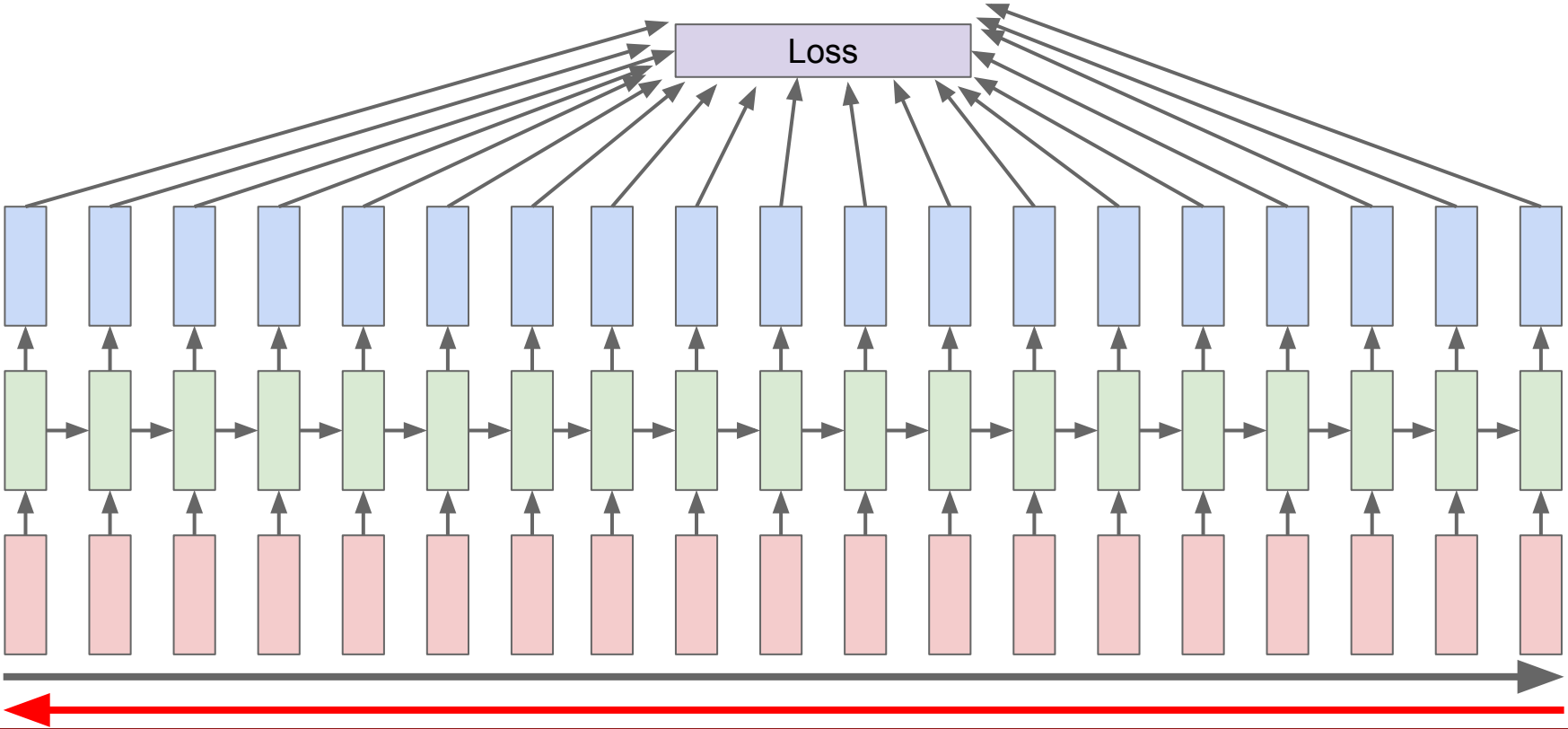
Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

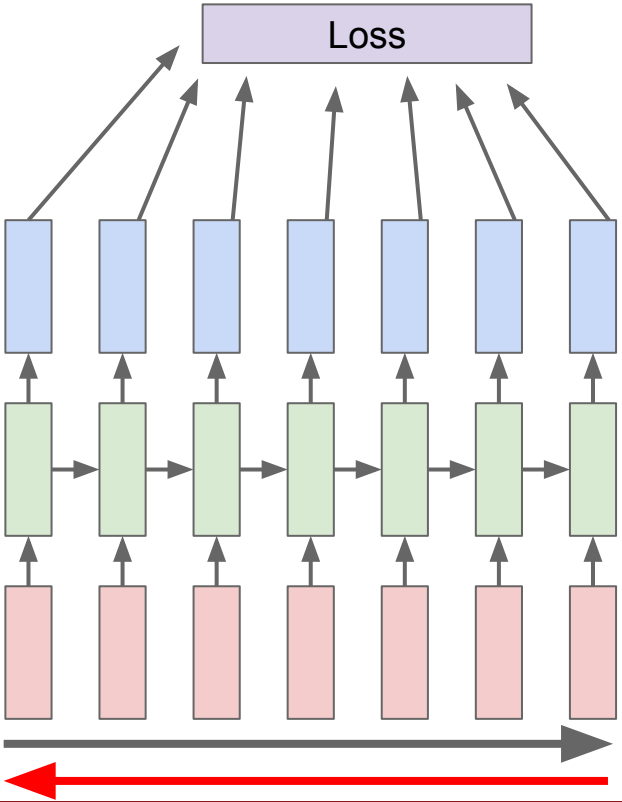


Backpropagation through time

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

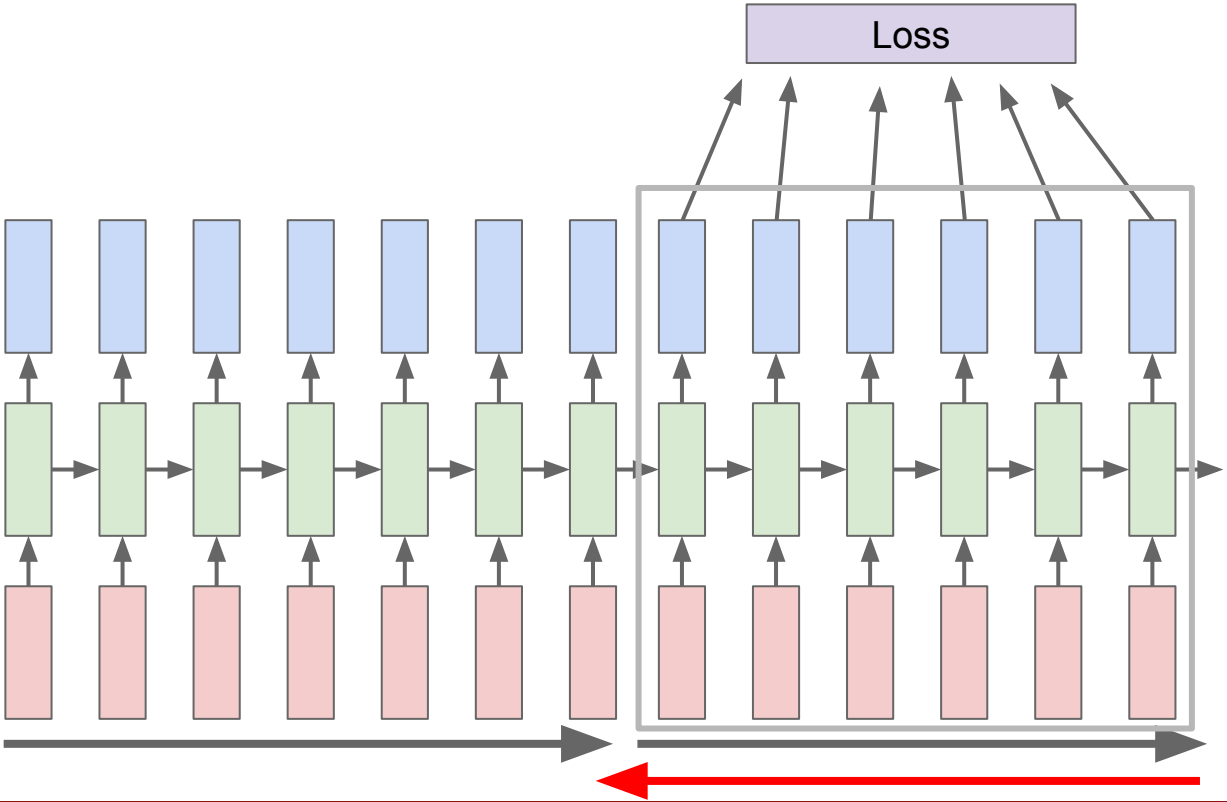


Truncated Backpropagation through time



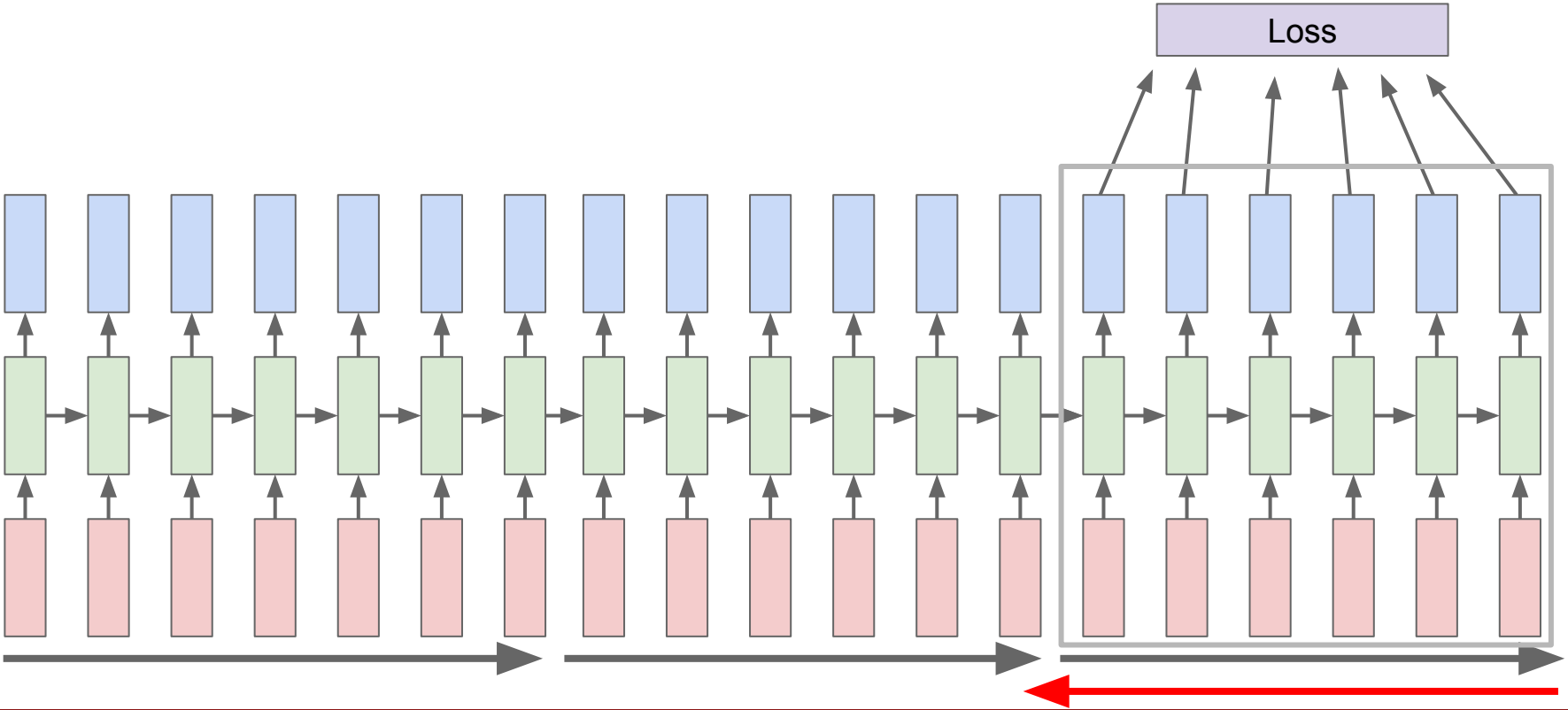
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



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

Truncated Backpropagation through time

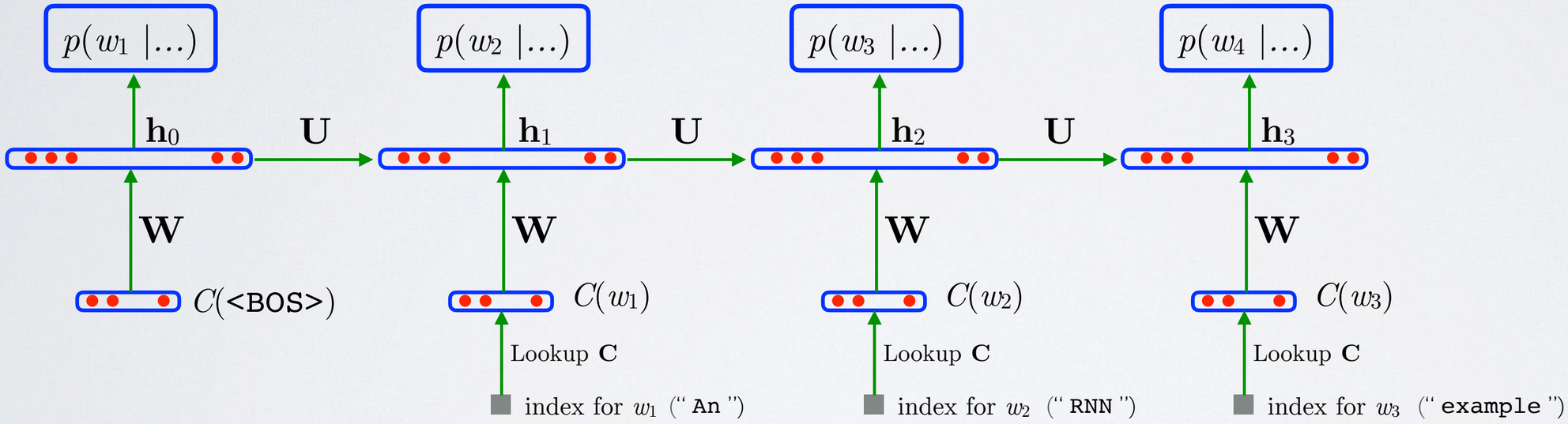


RECURRENT NEURAL NETWORK (RNN)

REMINDER

Topics: unrolled RNN

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

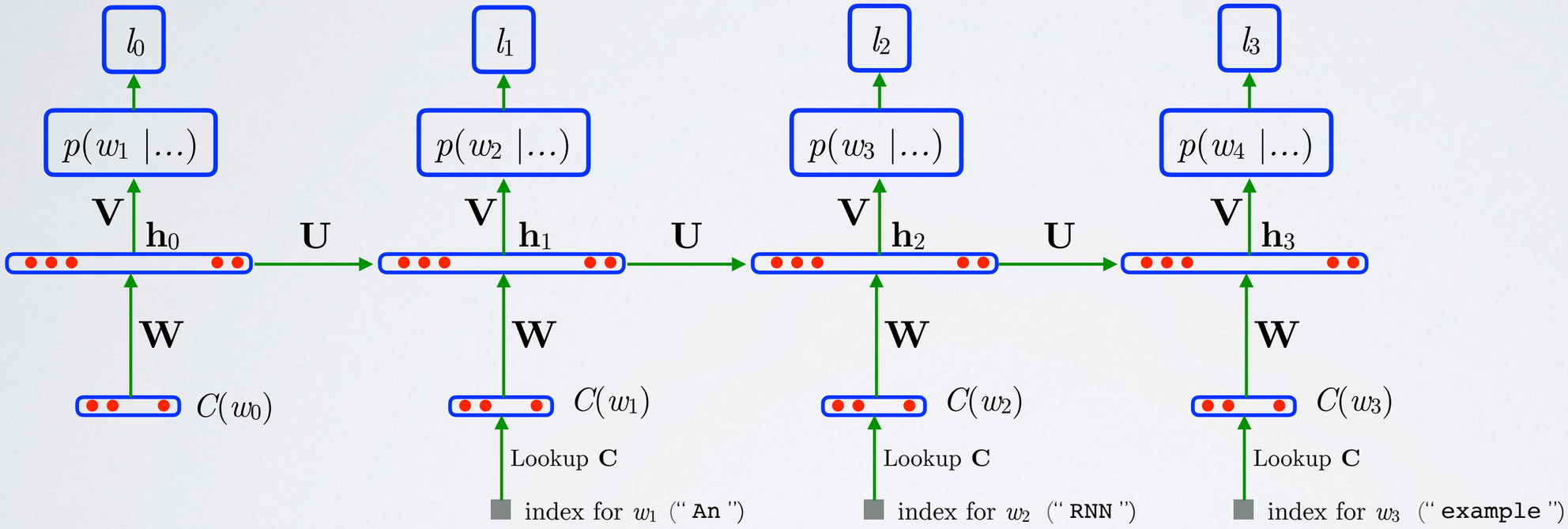


  - symbol “ `.` ” serves as an *end of sentence* symbol
  - alternative model for $\mathbf{h}_0$ is to set w_0 to a unique *beginning of sentence* symbol, with its own embedding $C(w_0)$ (but not included as possible output!)

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

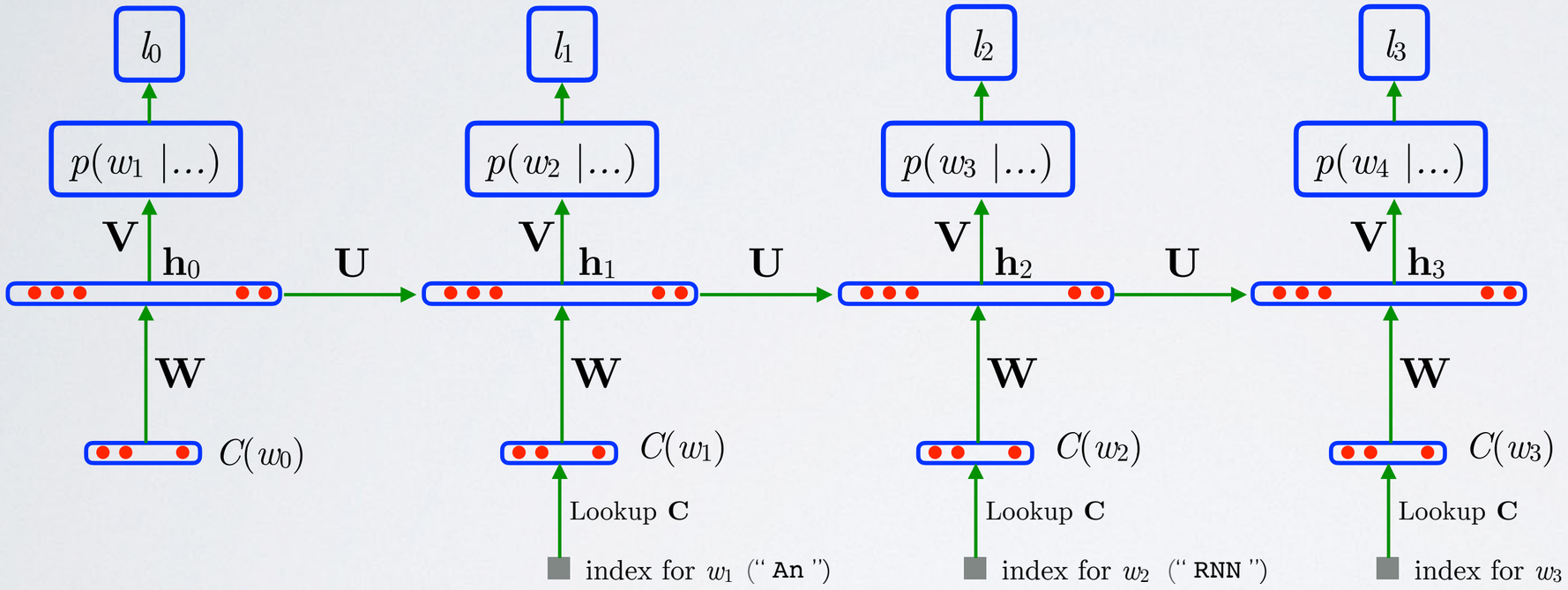


- want to minimize sum of per step loss $l = \sum_{t=0}^{T-1} l_t$
- for language modeling, $l_t = -\log p(w_{t+1} \mid \ldots)$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

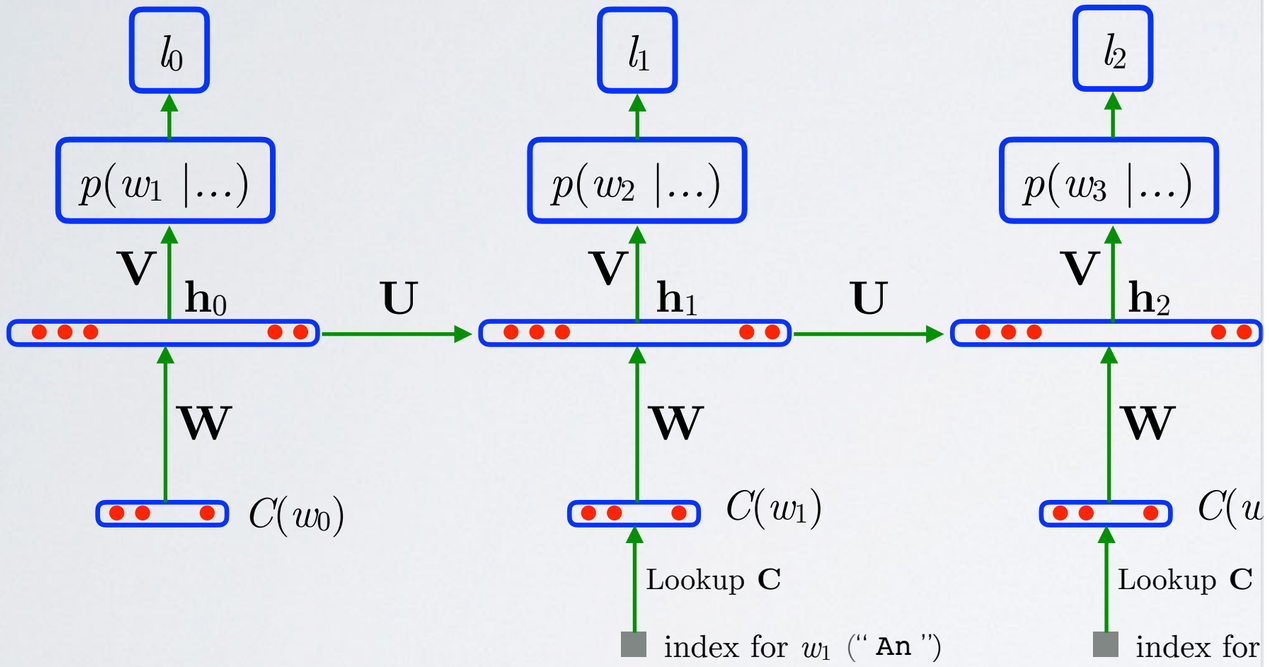


- ***forward propagation:*** computation follows arrows in flow graph (forward in time)
- ***backpropagation:*** computation goes in reverse order (backward in time)

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule



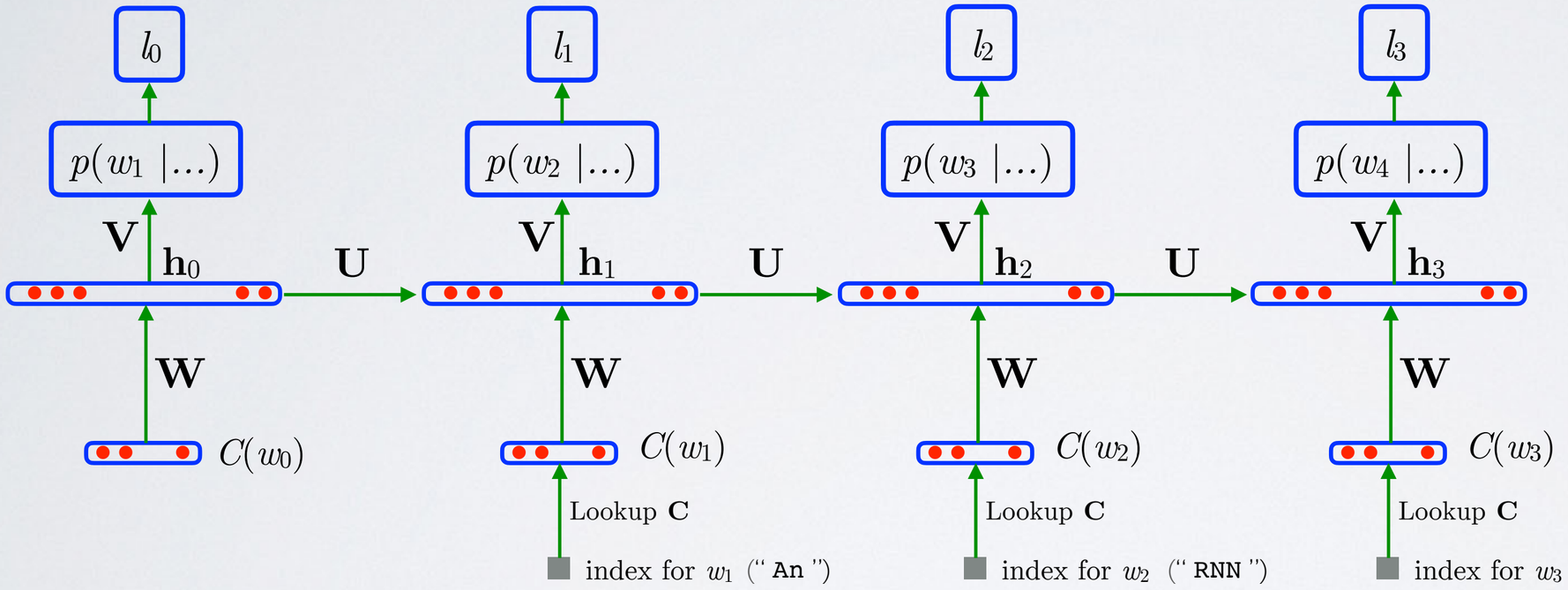
- initialize gradients
  - $\nabla_{\mathbf{V}} l \Leftarrow 0, \nabla_{\mathbf{W}} l \Leftarrow 0, \nabla_{\mathbf{U}} l \Leftarrow 0$
  - $\nabla_{\mathbf{h}_{T-1}} l \Leftarrow 0$
- for t from T-1 to 0
  - $\nabla_{\mathbf{V}} l \mathrel{+}= \nabla_{\mathbf{V}} l_t$
  - $\nabla_{\mathbf{h}_t} l \mathrel{+}=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

- ***forward propagation:*** computation follows arrows in flow graph (forward in time)
- ***backpropagation:*** computation goes in reverse order (backward in time)

TRUNCATED BPTT

Topics: truncated BPTT

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



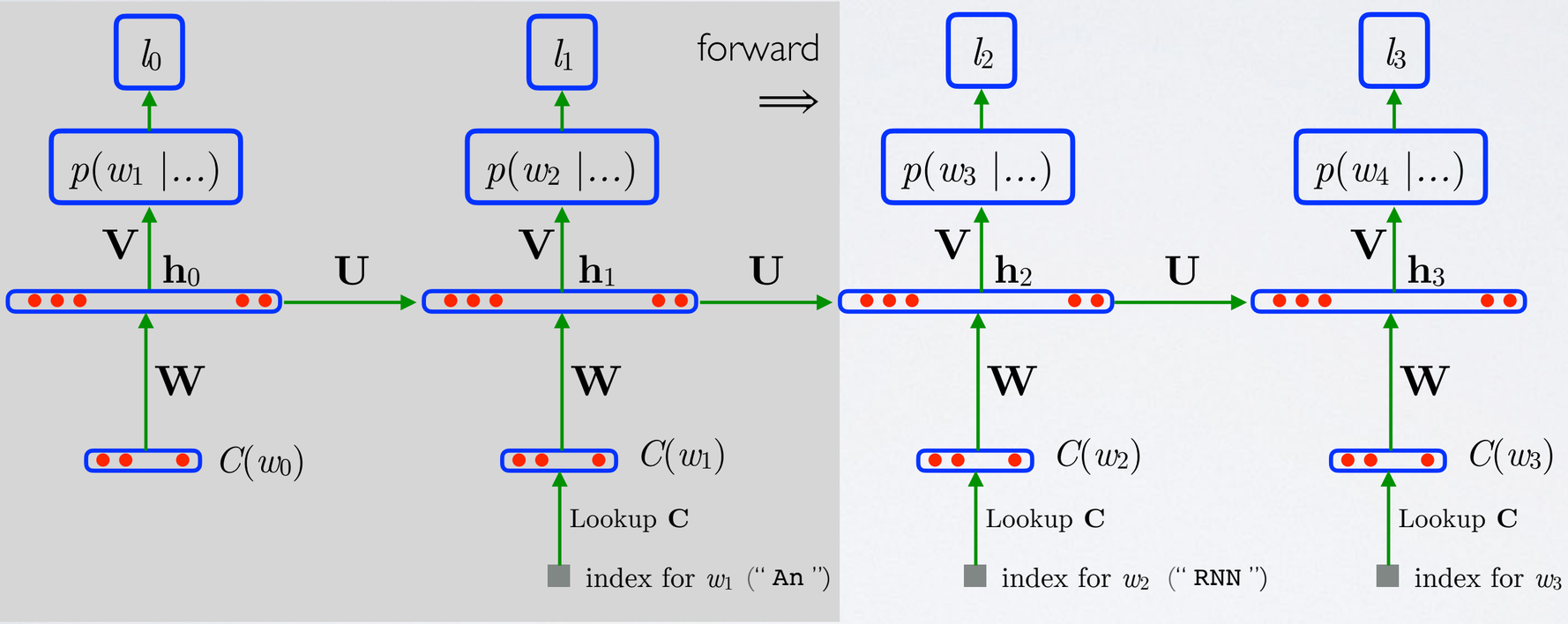
- **Truncated BPTT:** approximate BPTT by
  - performing forward pass k_1 steps at a time
  - running BPTT only k_2 steps backward and update (assuming earlier steps are fixed)

TRUNCATED BPTT

Topics: truncated BPTT

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

Example with
$k_1 = k_2 = 2$



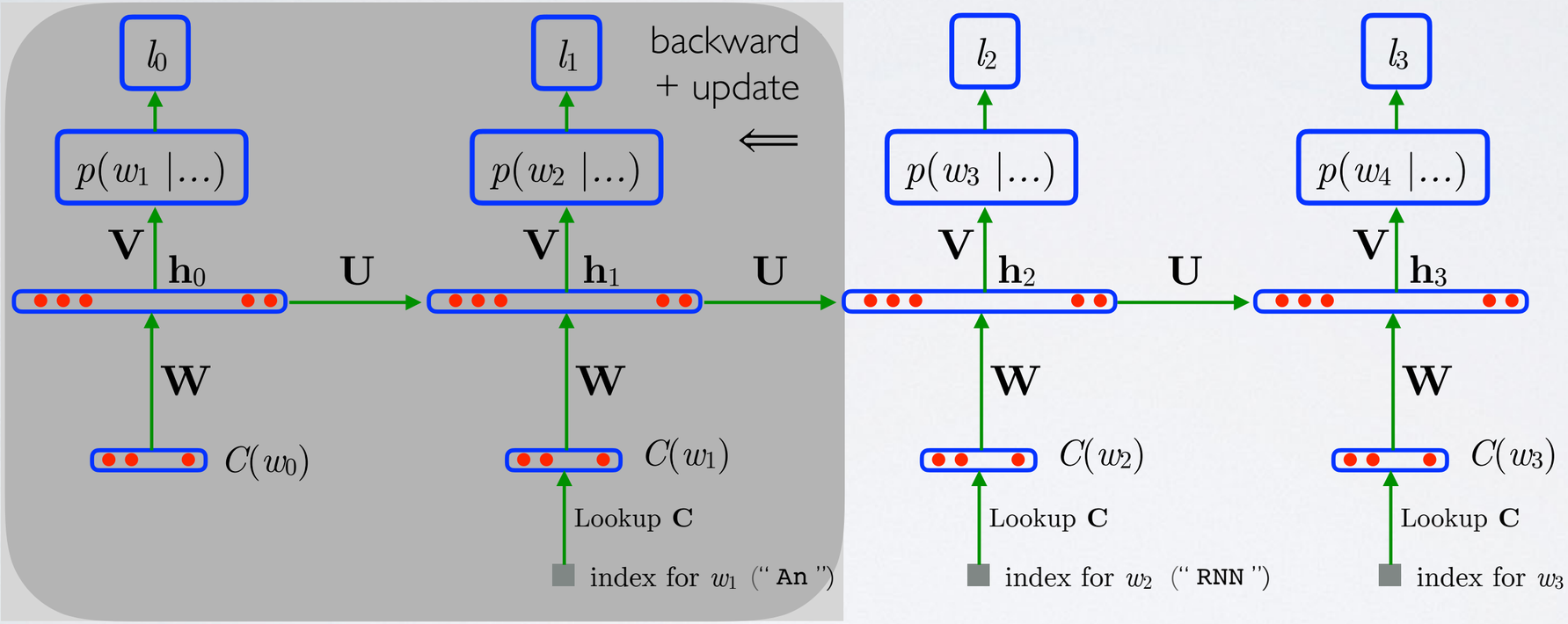
- **Truncated BPTT:** approximate BPTT by
  - performing forward pass k_1 steps at a time
  - running BPTT only k_2 steps backward and update (assuming earlier steps are fixed)

TRUNCATED BPTT

Topics: truncated BPTT

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

Example with
$k_1 = k_2 = 2$

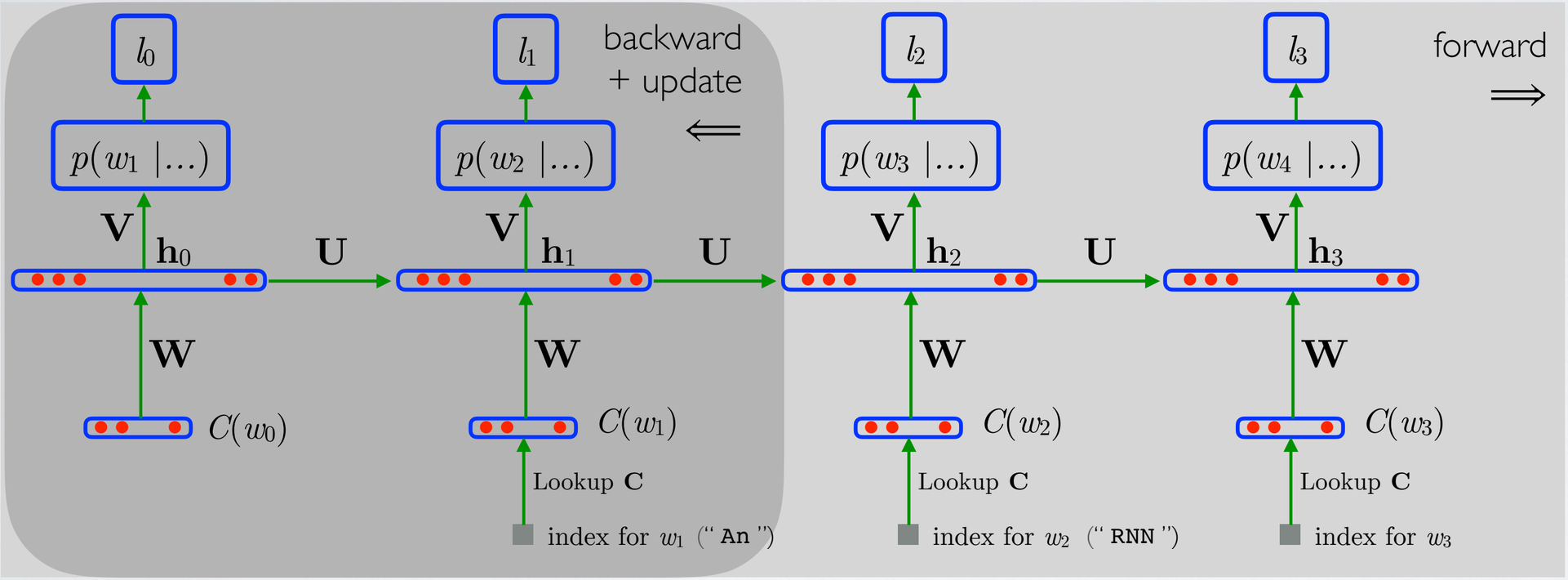


- **Truncated BPTT:** approximate BPTT by
  - performing forward pass k_1 steps at a time
  - running BPTT only k_2 steps backward and update (assuming earlier steps are fixed)

TRUNCATED BPTT

Topics: truncated BPTT

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



Example with $k_1 = k_2 = 2$

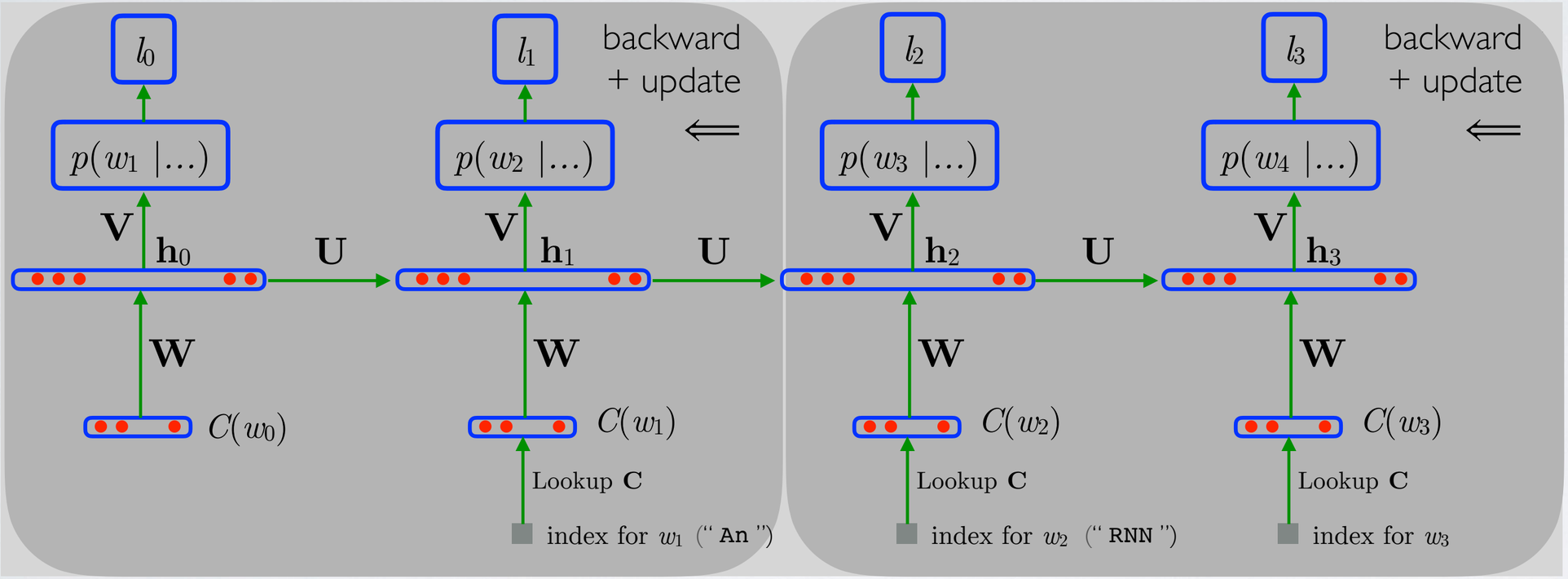
Computed from "pre-update" $\mathbf{h}_1$

- **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)

TRUNCATED 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)

min-char-rnn.py gist: 112 lines of Python

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

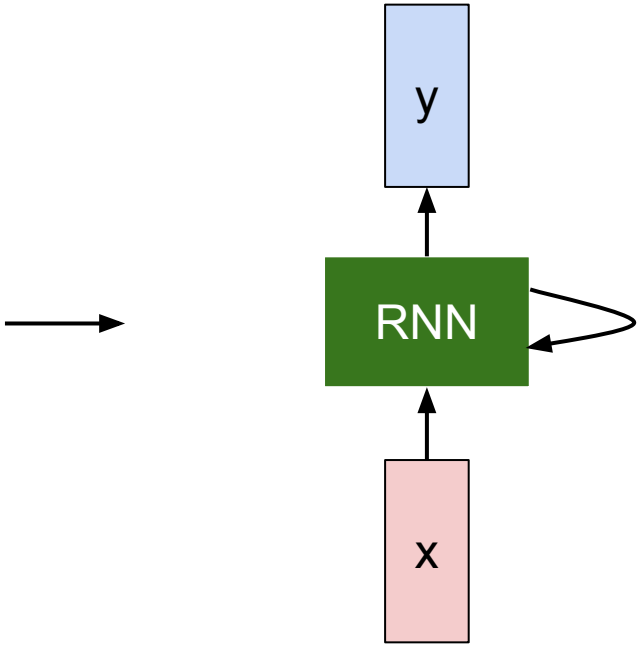
(https://gist.github.com/karpathy/d4dee566867f8291f086)

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.



at first:

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

train more

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

train more

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

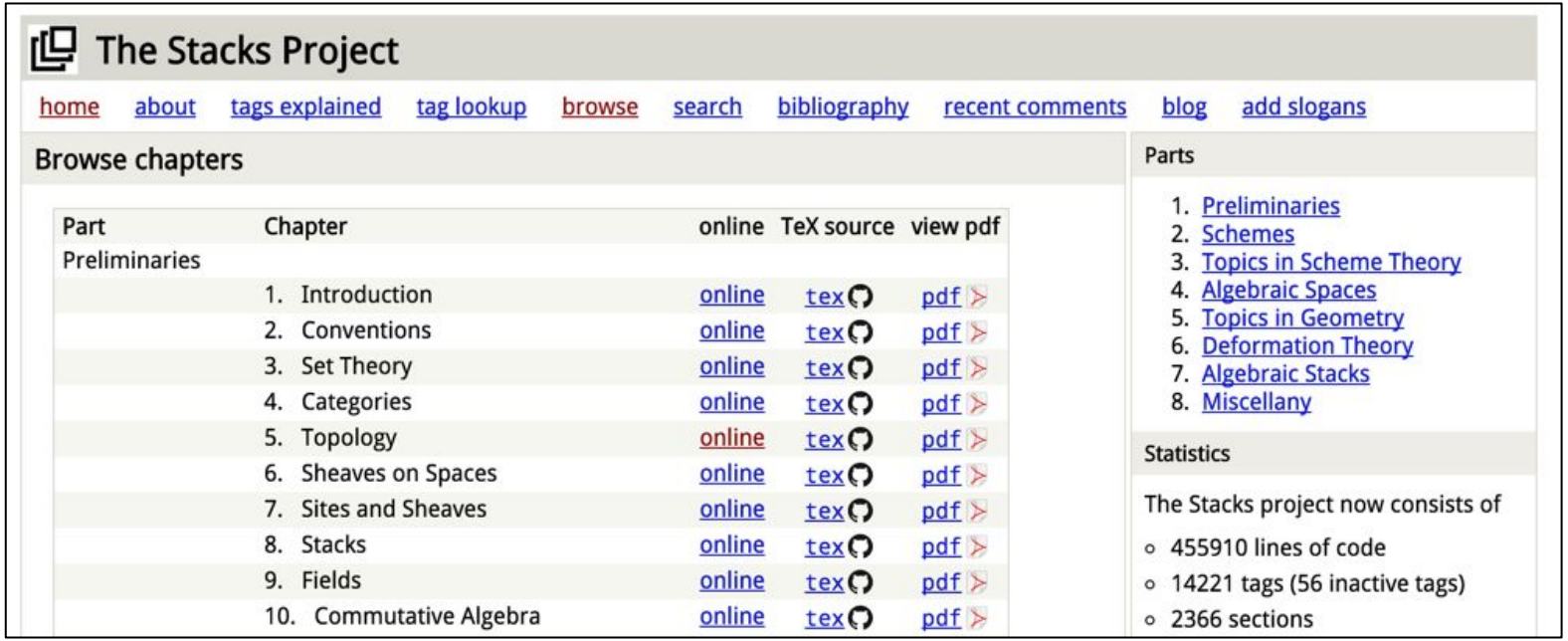
train more

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

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

The Stacks Project: open source algebraic geometry textbook



The Stacks Project

home about tags explained tag lookup browse search bibliography recent comments blog add slogans

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	online	tex	pdf
	2. Conventions	online	tex	pdf
	3. Set Theory	online	tex	pdf
	4. Categories	online	tex	pdf
	5. Topology	online	tex	pdf
	6. Sheaves on Spaces	online	tex	pdf
	7. Sites and Sheaves	online	tex	pdf
	8. Stacks	online	tex	pdf
	9. Fields	online	tex	pdf
	10. Commutative Algebra	online	tex	pdf

Parts

1. Preliminaries
2. Schemes
3. Topics in Scheme Theory
4. Algebraic Spaces
5. Topics in Geometry
6. Deformation Theory
7. Algebraic Stacks
8. Miscellany

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source

http://stacks.math.columbia.edu/
The stacks project is licensed under the GNU Free Documentation License

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciy in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation ??. Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

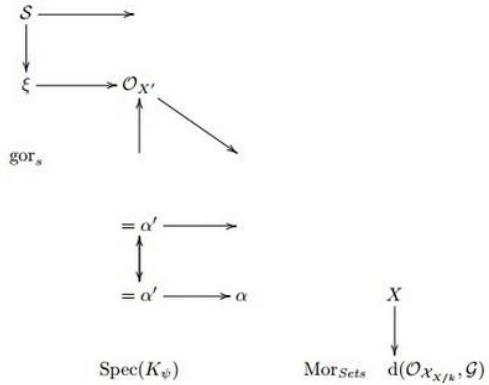
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f_\star$. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

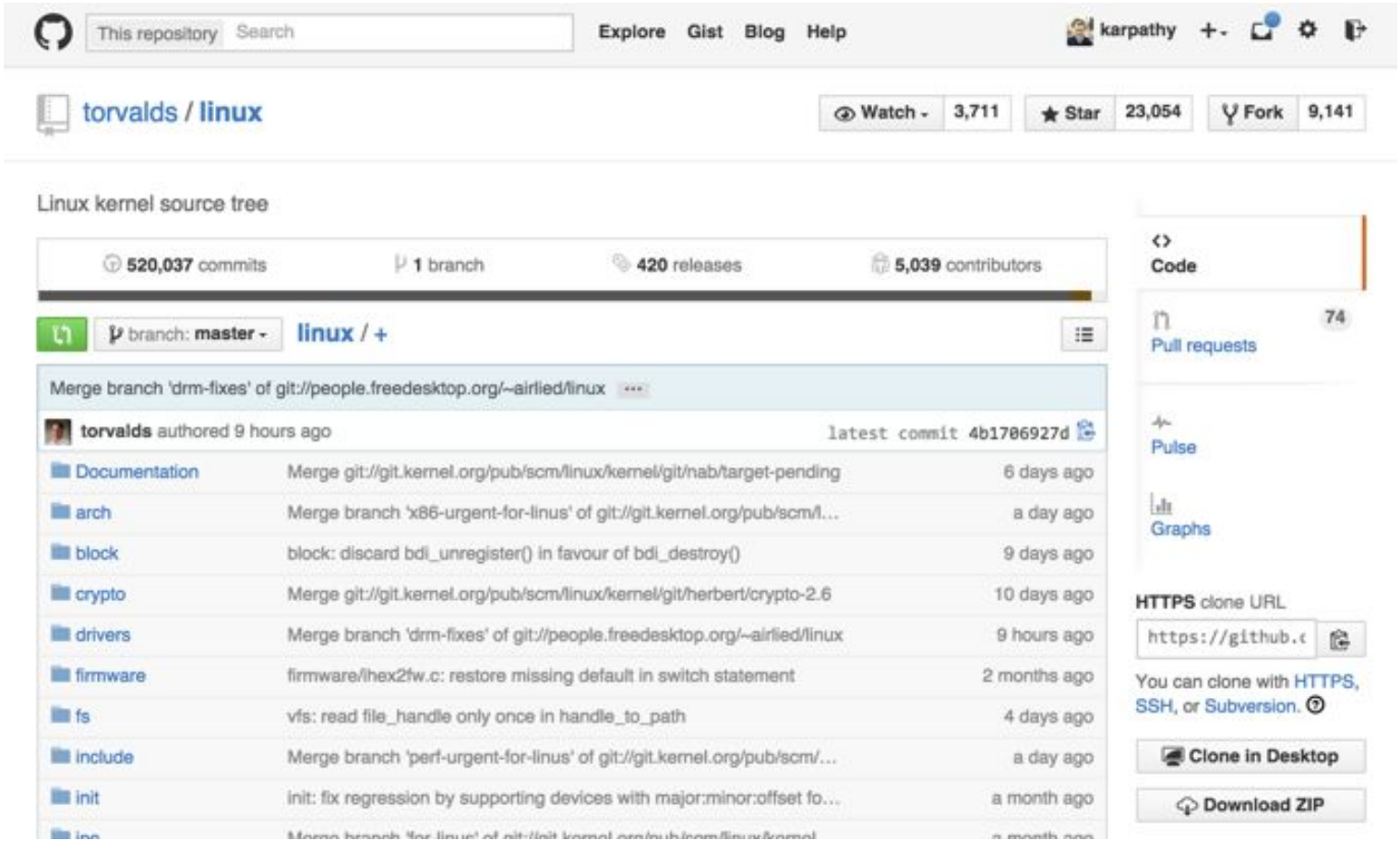
Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_t}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.


This repository Search
Explore Gist Blog Help
karpathy
torvalds / linux
Watch 3,711
Star 23,054
Fork 9,141
Linux kernel source tree
520,037 commits
1 branch
420 releases
5,039 contributors
branch: master
linux / +
Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux
torvalds authored 9 hours ago
latest commit 4b1706927d
Documentation Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending 6 days ago
arch Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l... a day ago
block block: discard bdi_unregister() in favour of bdi_destroy() 9 days ago
crypto Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6 10 days ago
drivers Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux 9 hours ago
firmware firmware/ihex2fw.c: restore missing default in switch statement 2 months ago
fs vfs: read file_handle only once in handle_to_path 4 days ago
include Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/... a day ago
init init: fix regression by supporting devices with major:minor:offset fo... a month ago
Code
74
Pull requests
Pulse
Graphs
HTTPS clone URL
https://github.c
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000ffffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Generated C code

```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

Image Captioning

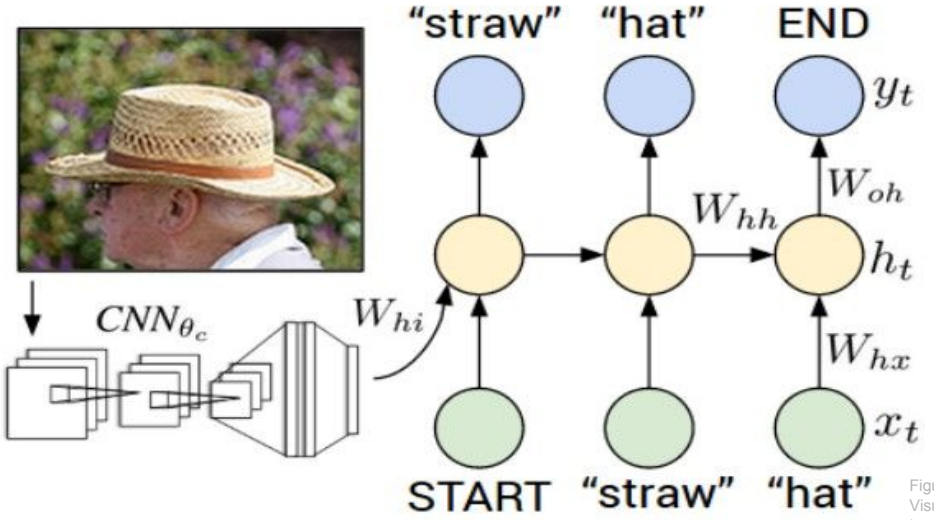
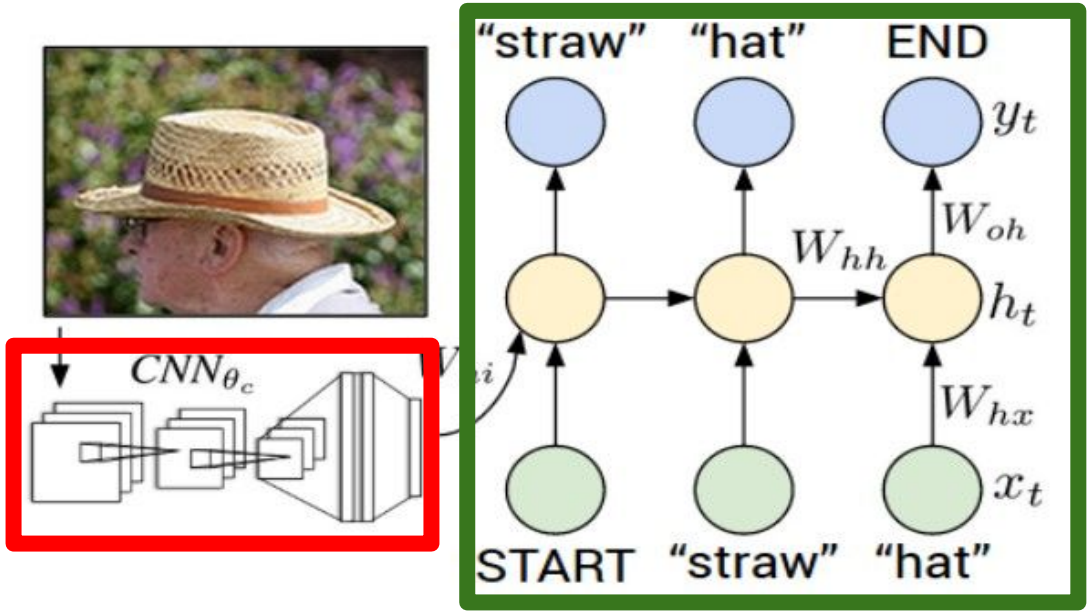


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network



Convolutional Neural Network

test image

image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096
FC-1000
softmax

test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax

test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

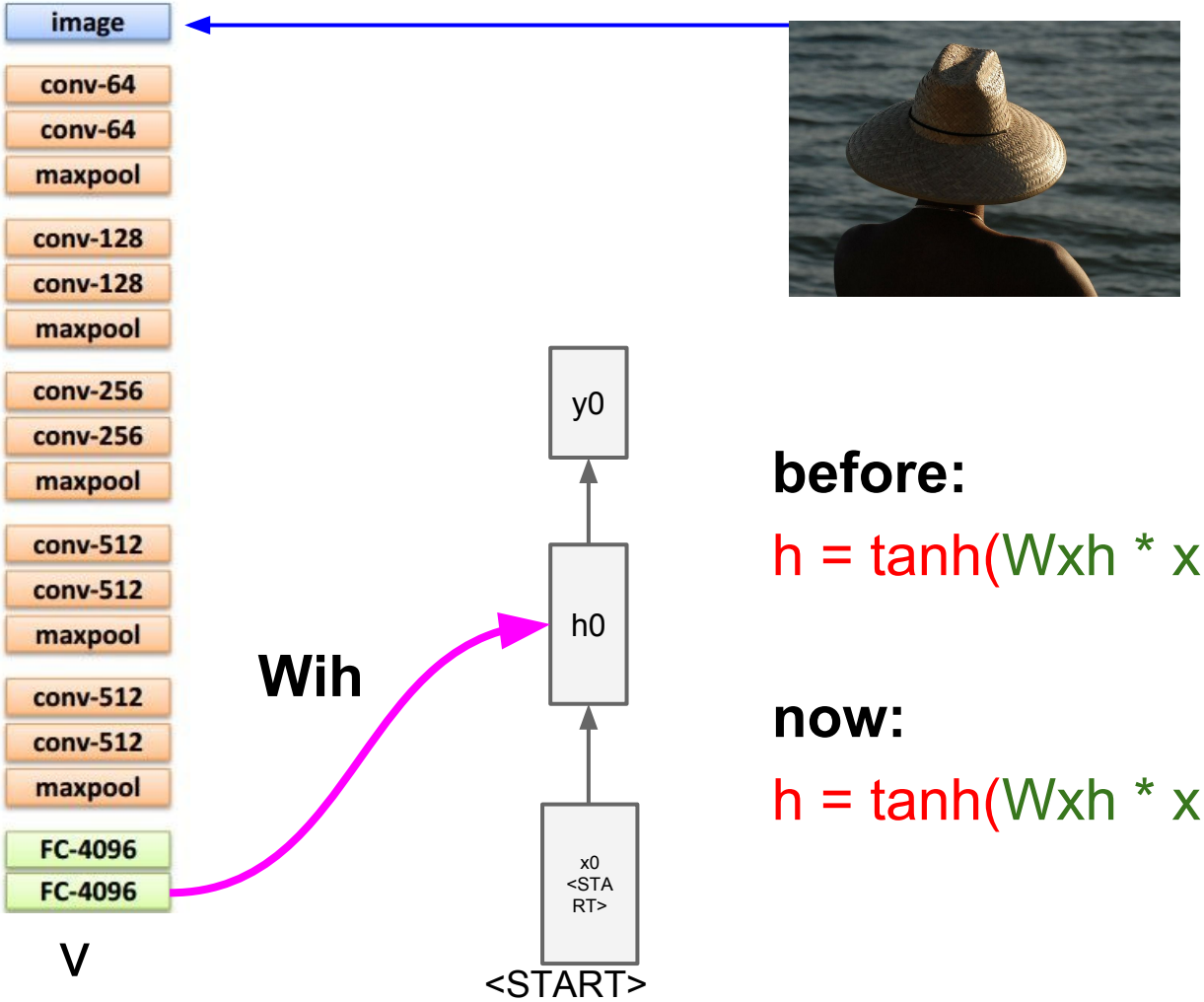
FC-4096

FC-4096

test image

x0
<STA
RT>

<START>



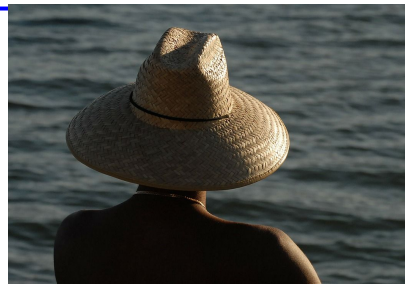
test image

before:

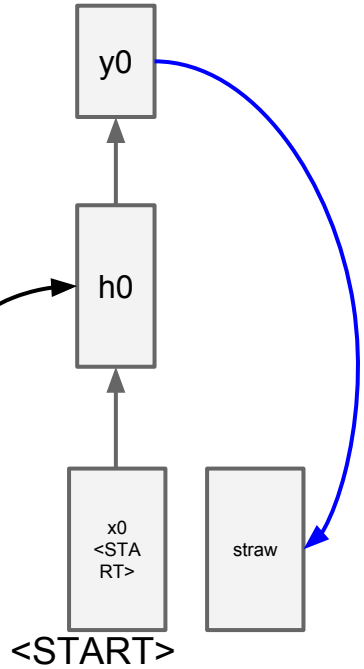
h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h **+ Wih * v**)



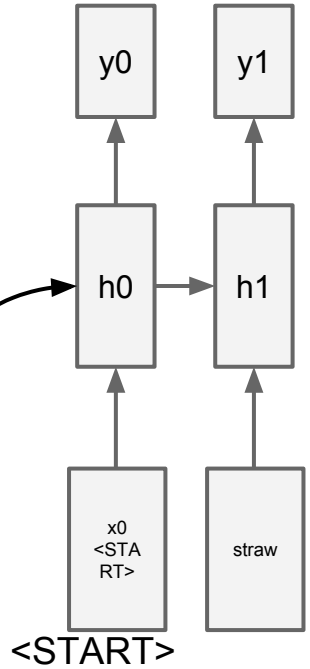
test image

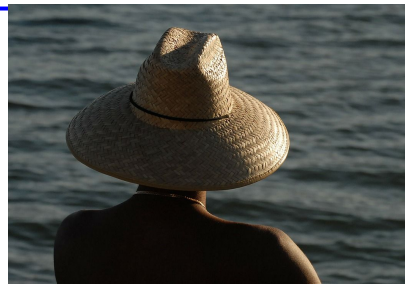


sample!

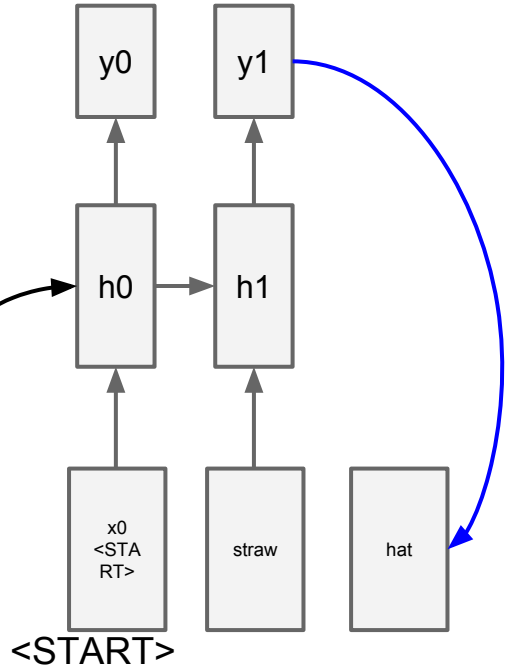

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

test image


y0
y1
h0
h1
x0
<STA
RT>
straw
<START>



test image



sample!

image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

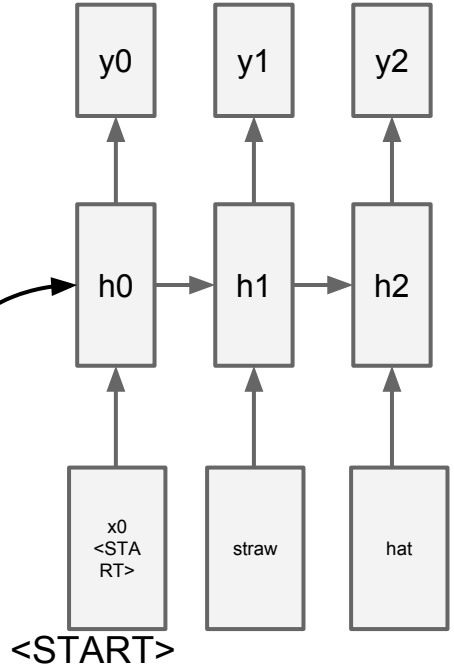
conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096

test image



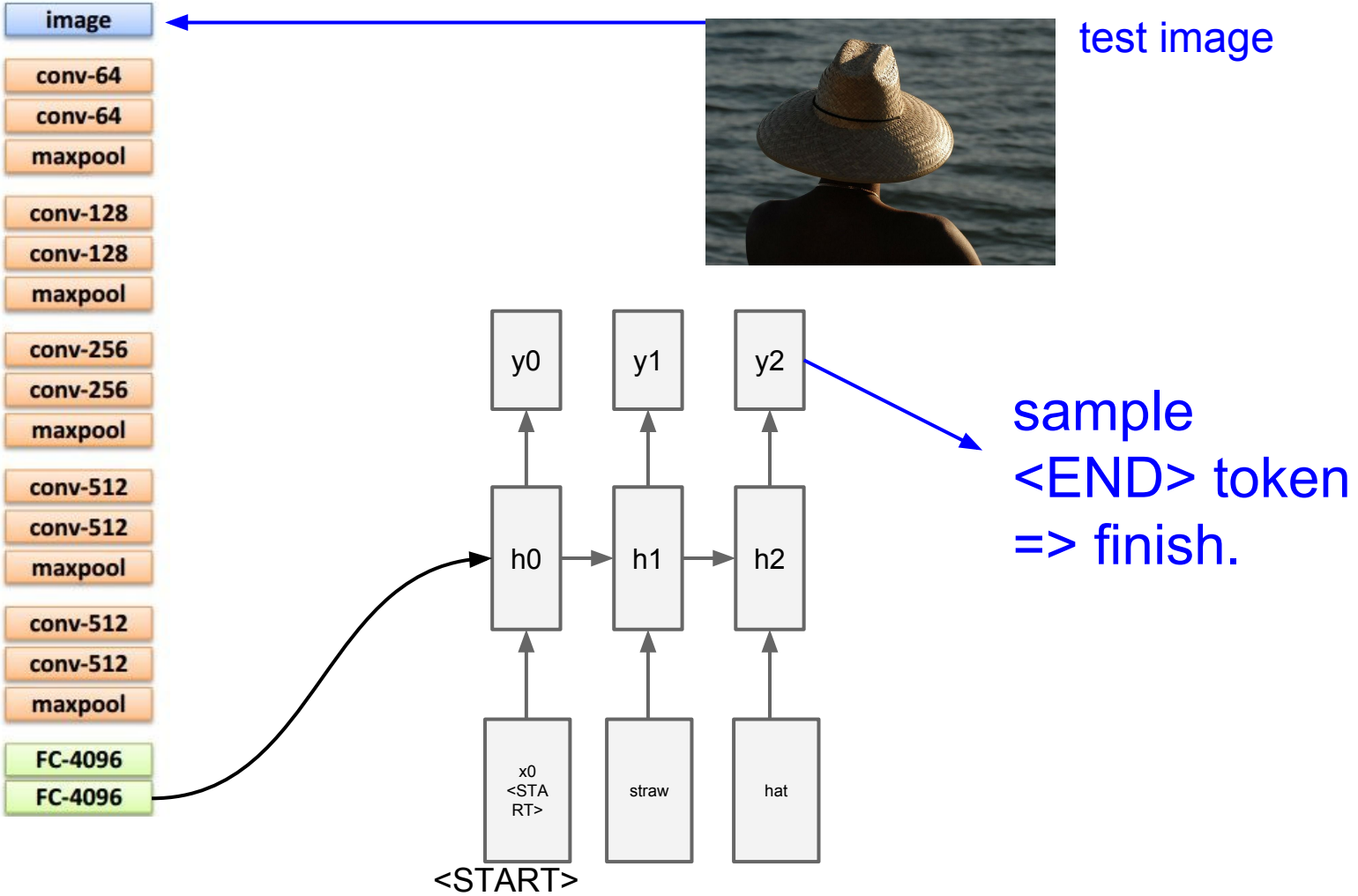


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0
<STA
RT>
straw
hat
<START>
sample
<END> token
=> finish.

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

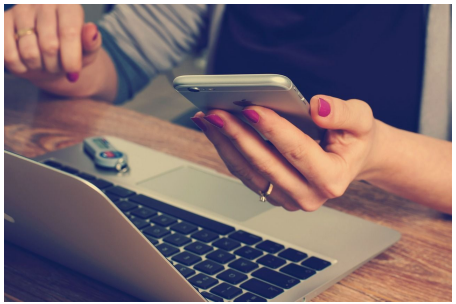
Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

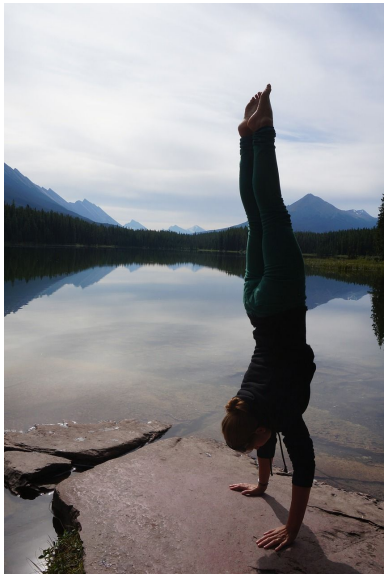
Image Captioning: Failure Cases

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand



A person holding a computer mouse on a desk



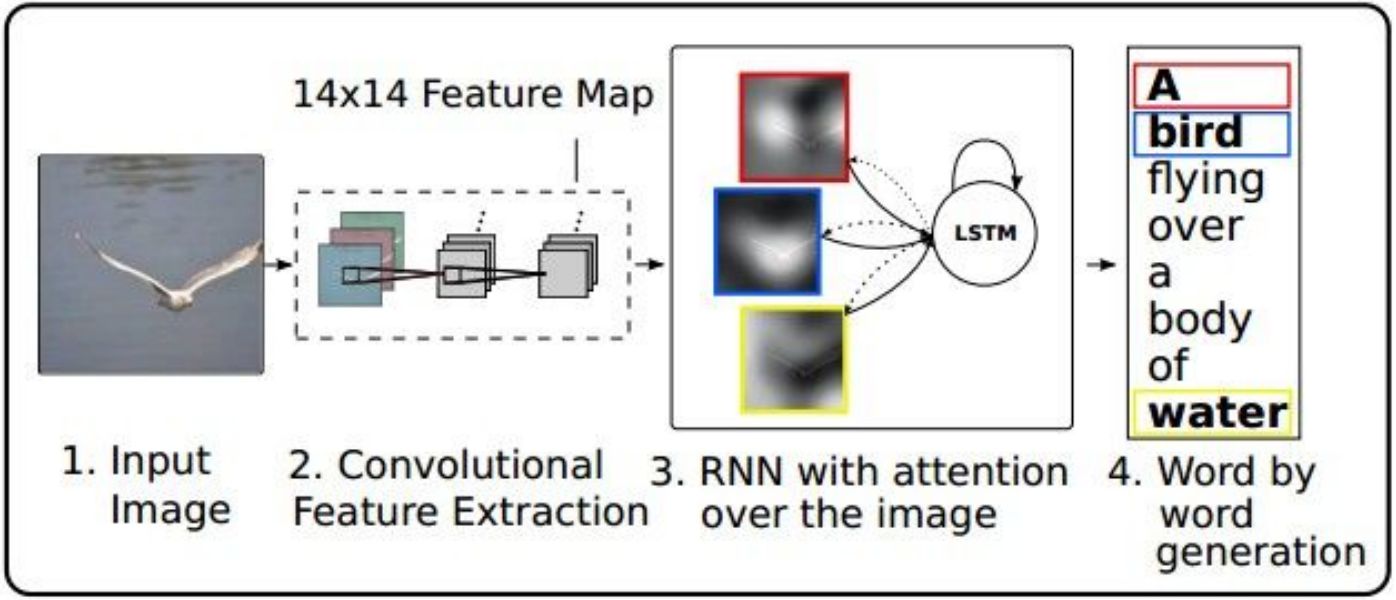
A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

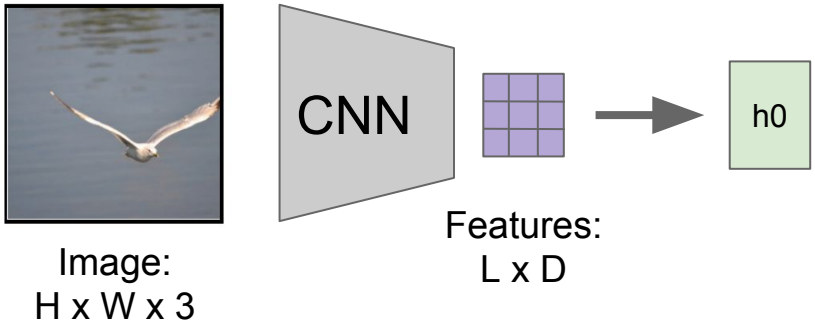
Image Captioning with Attention

RNN focuses its attention at a different spatial location when generating each word



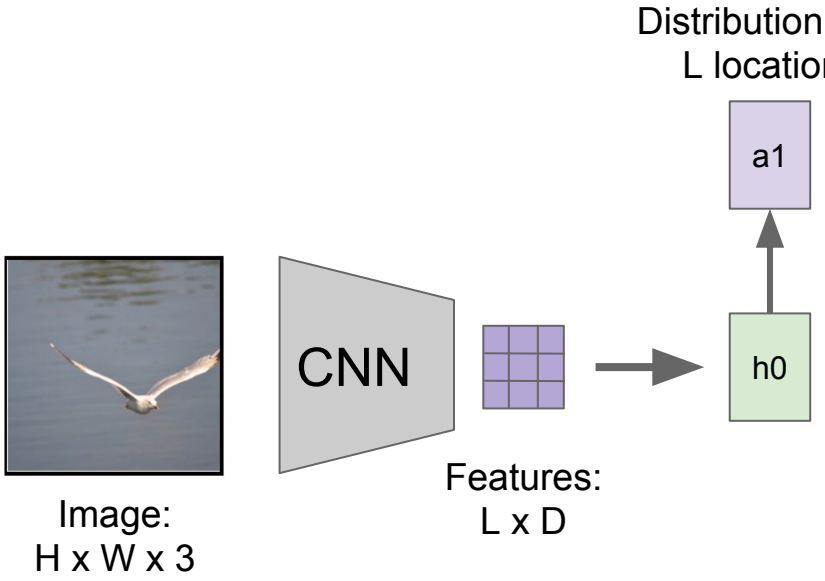
Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with Attention



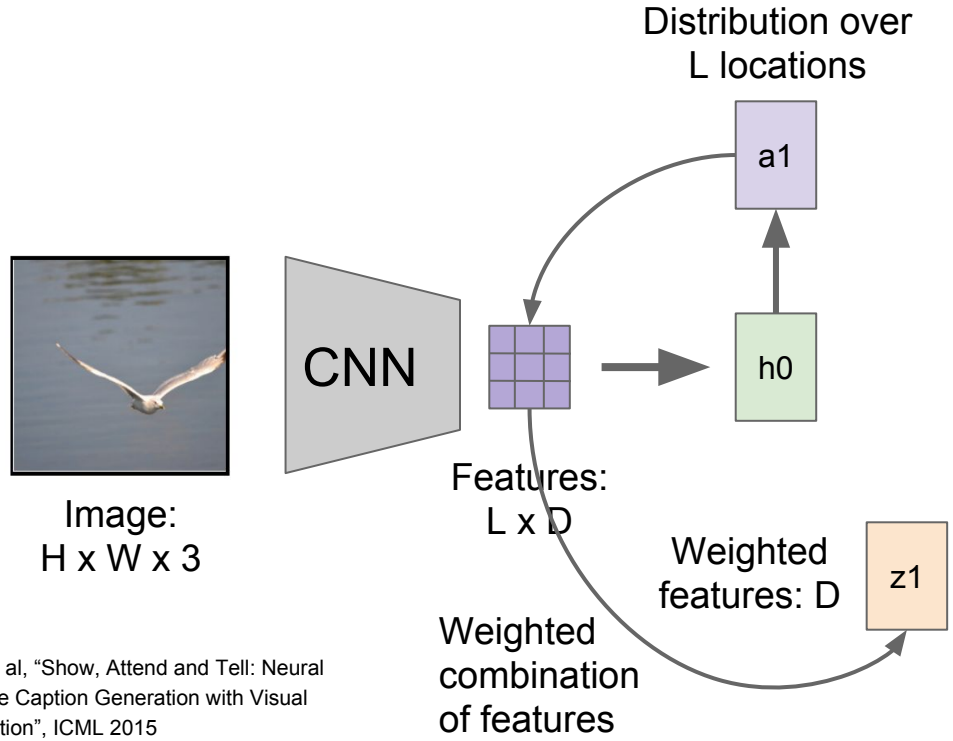
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with Attention



Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

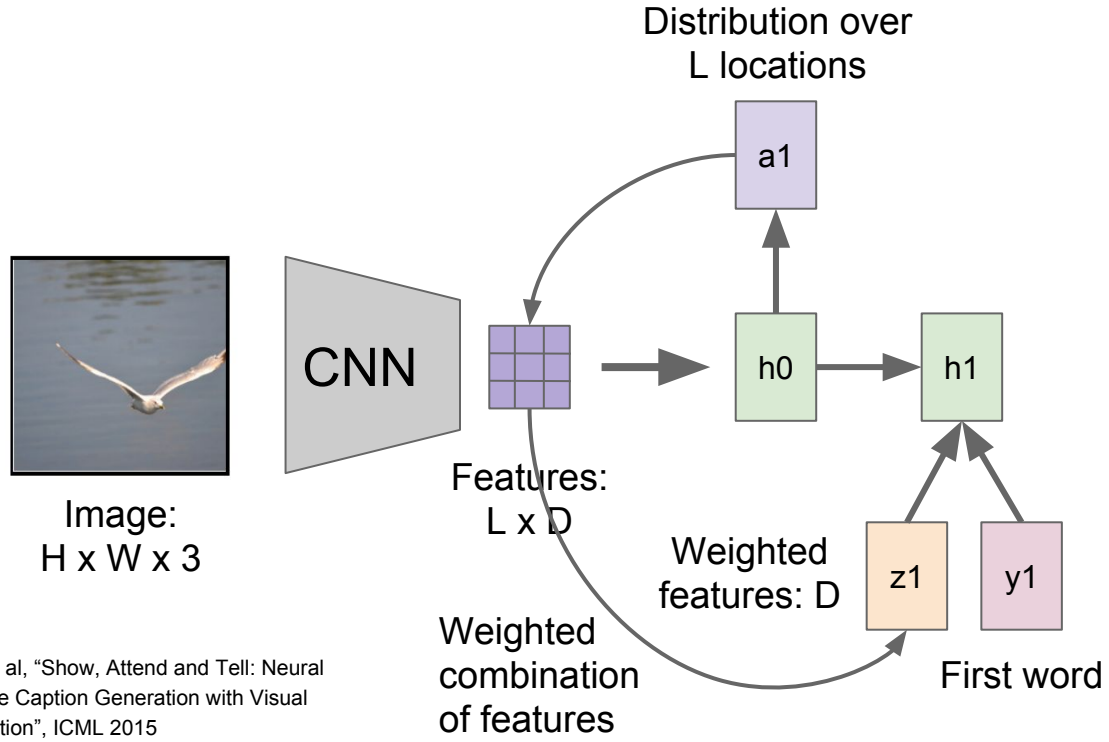
Image Captioning with Attention



$$z = \sum_{i=1}^{L} p_i v_i$$

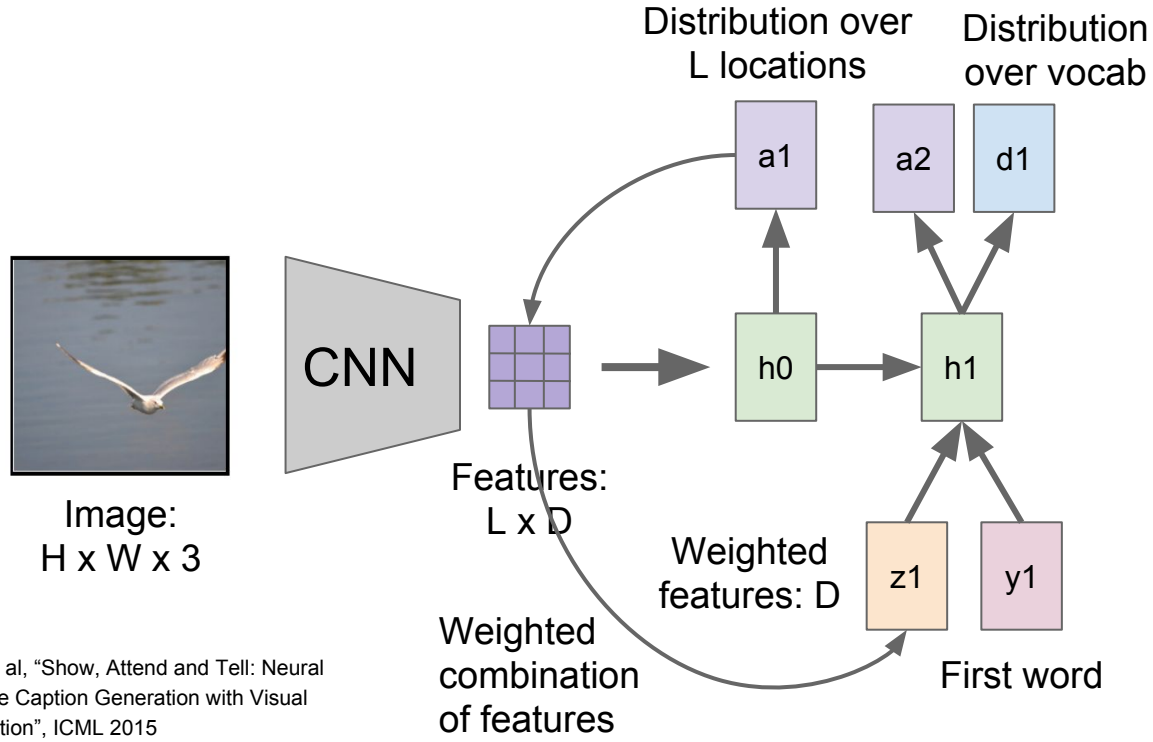
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention



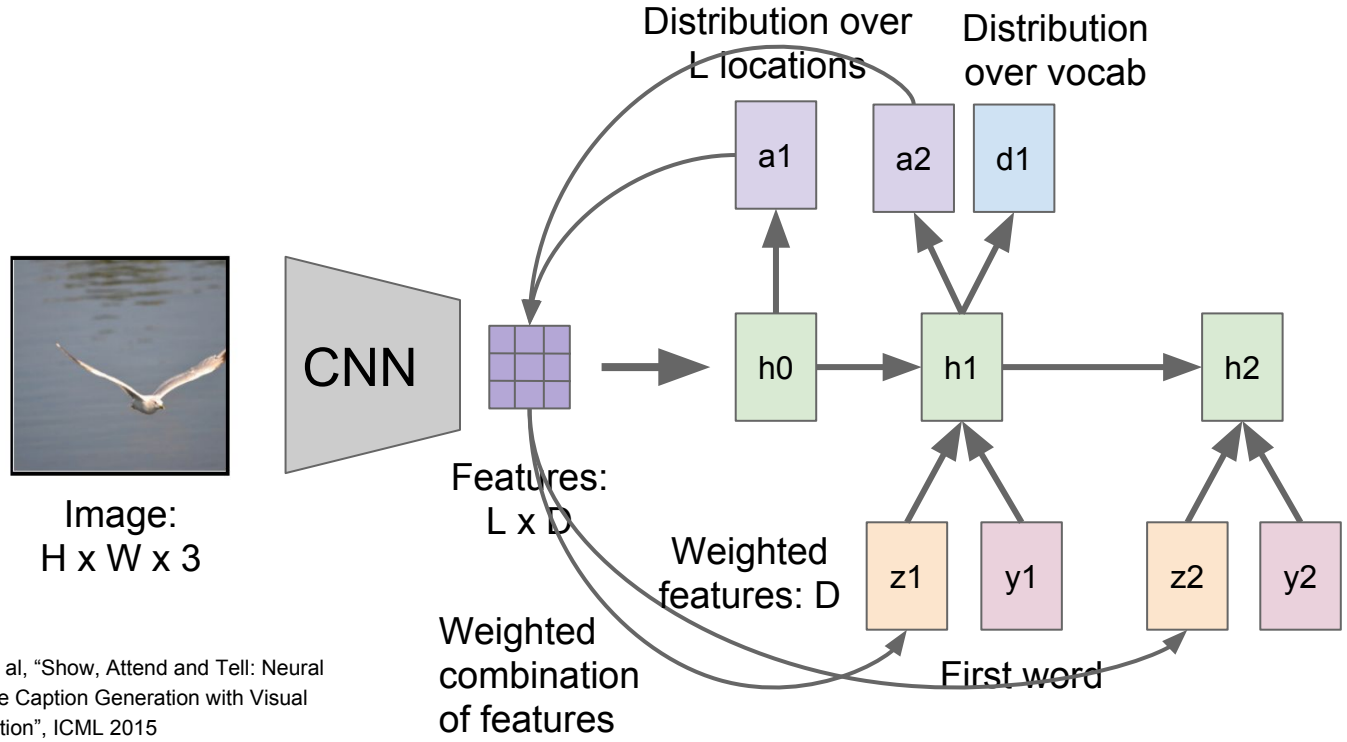
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention



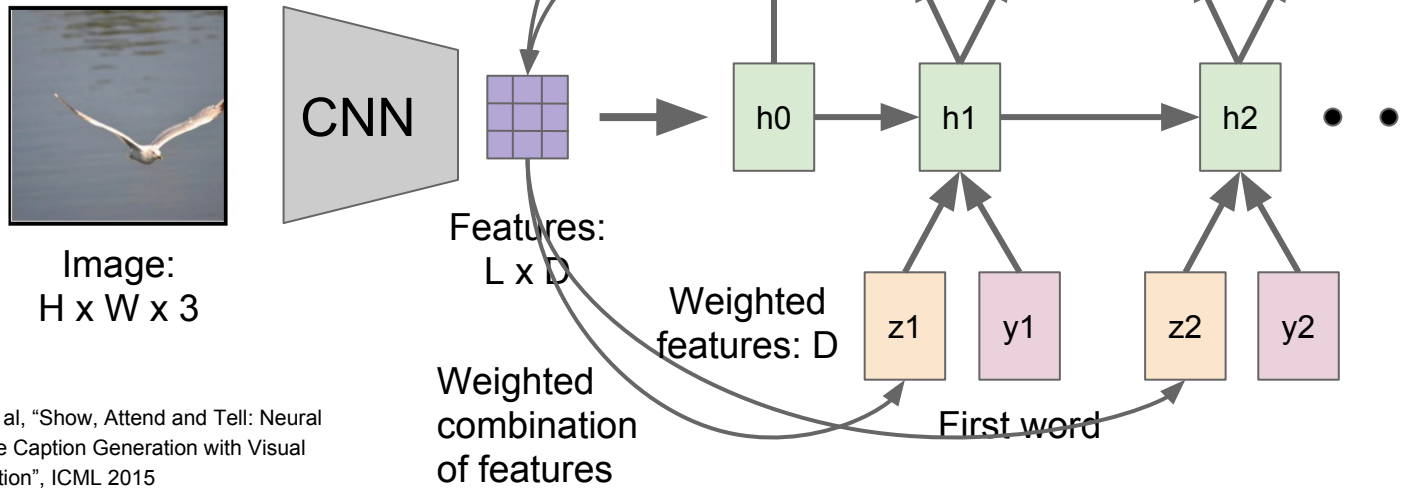
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention



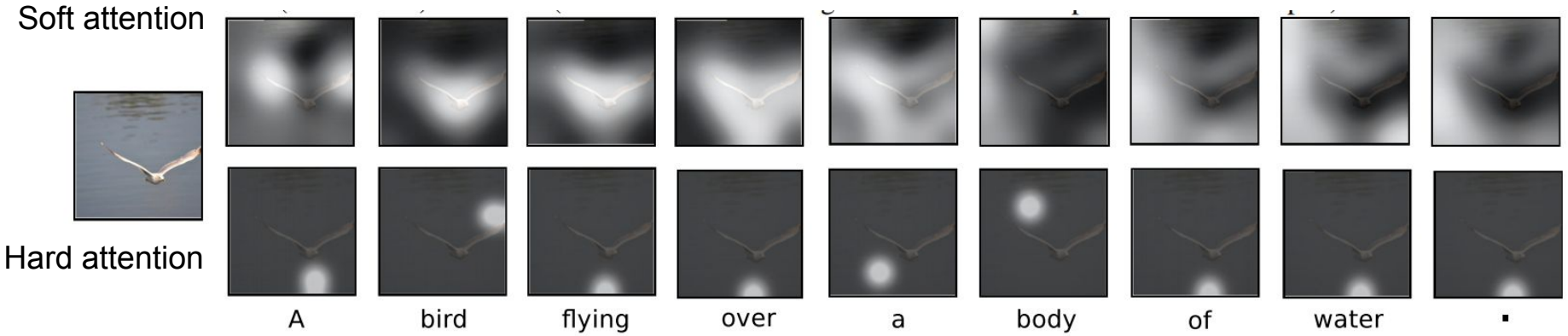
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention



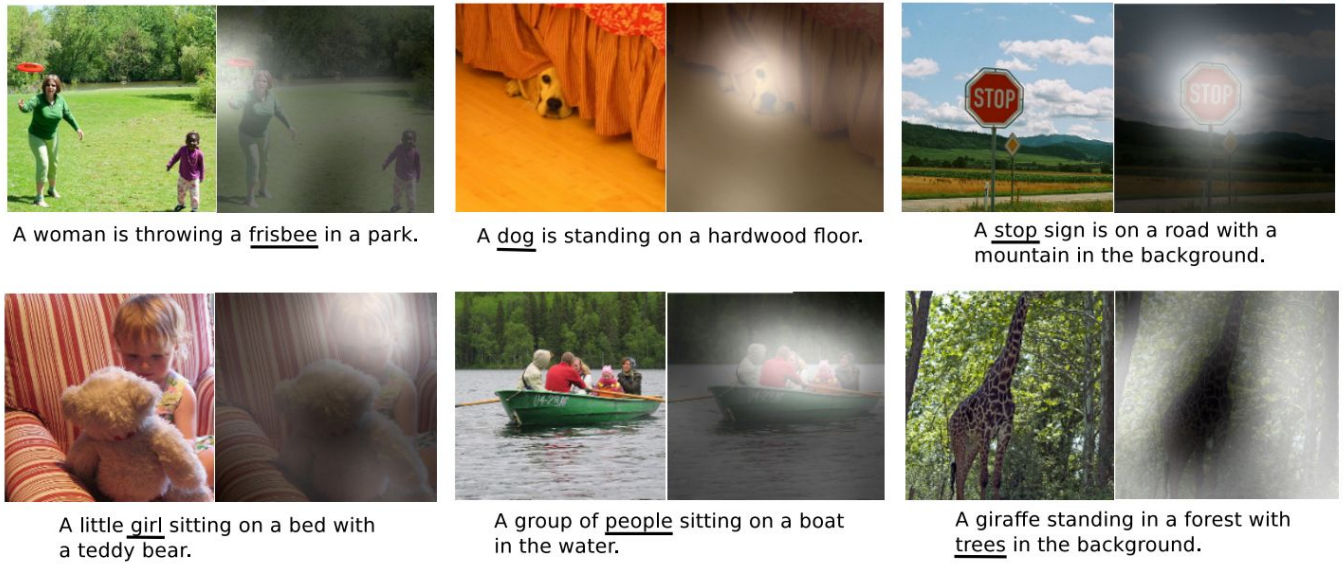
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention



Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with Attention



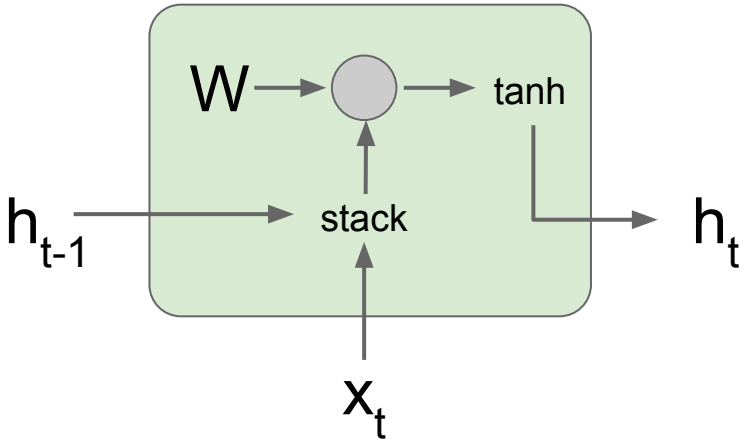
Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Recurrent neural networks

Exploding/vanishing gradient problem

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

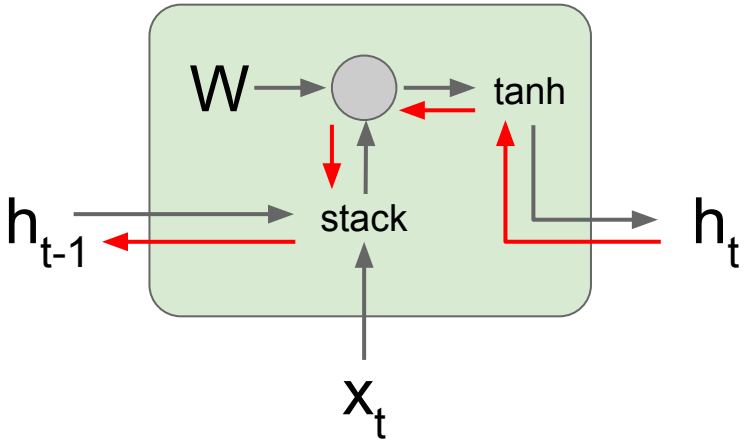


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

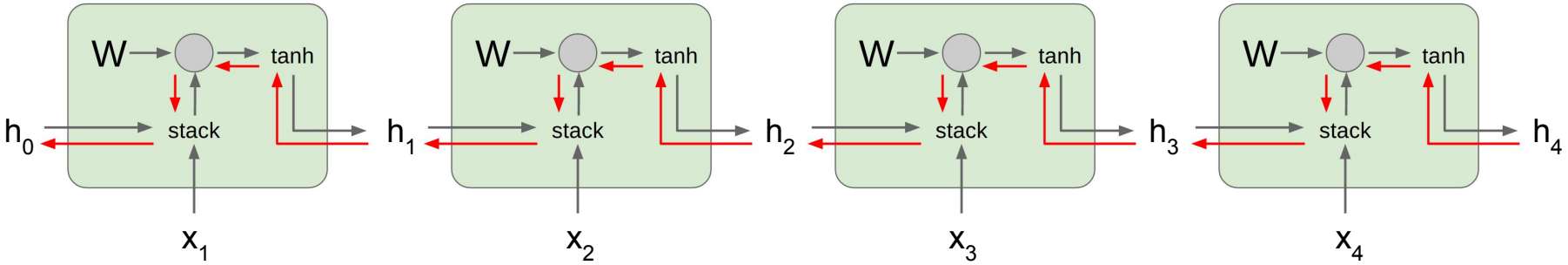
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

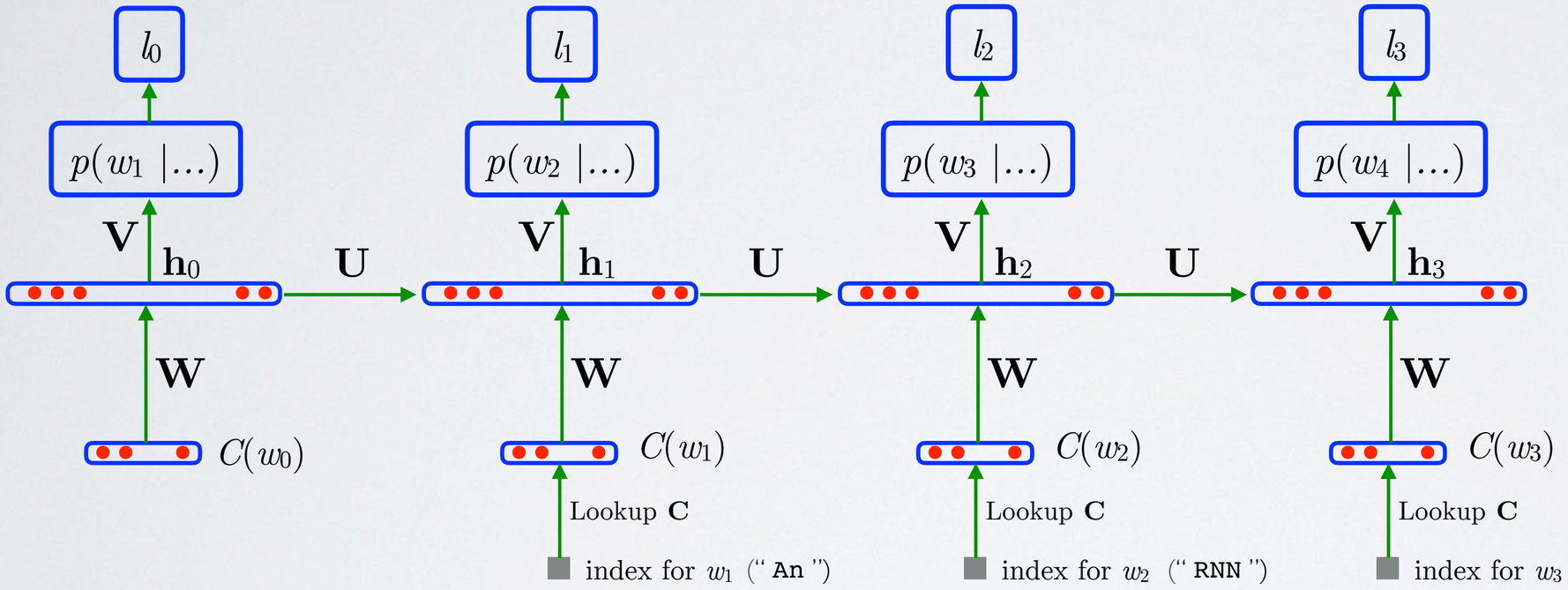


Computing gradient of h_0 involves many factors of W (and repeated tanh)

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

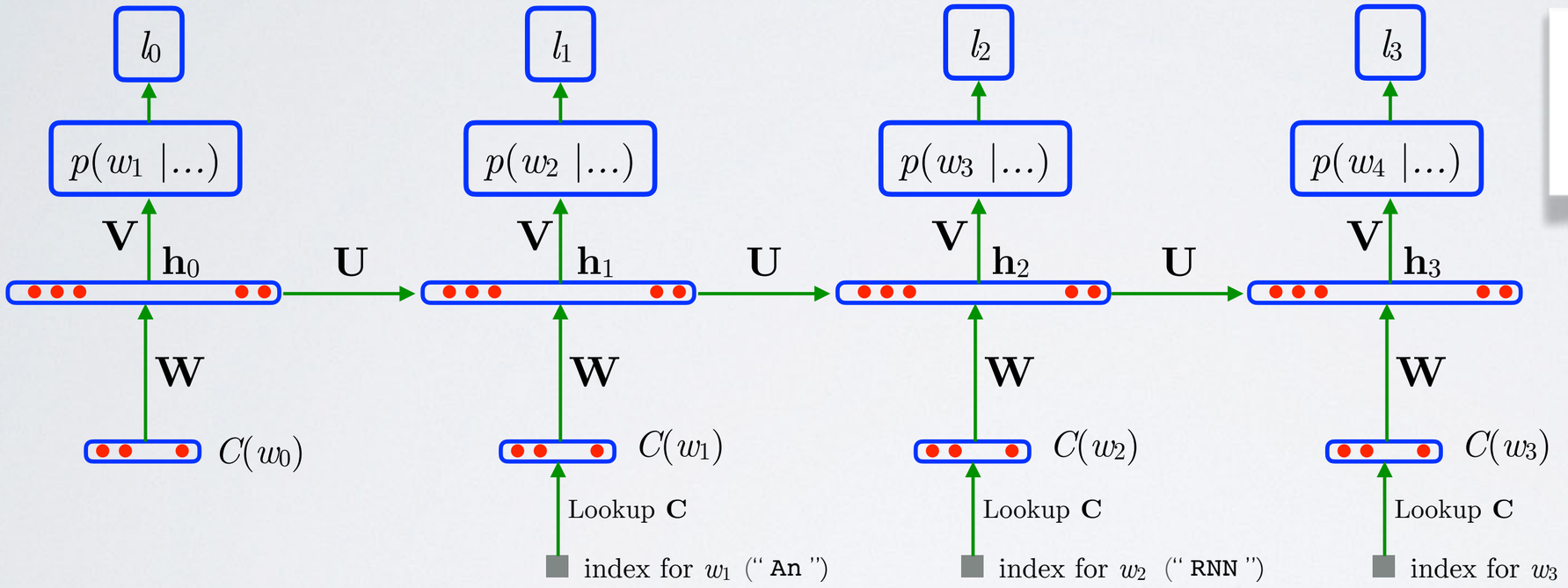


- Consider the gradient with respect to $\mathbf{h}_t$: $\nabla_{\mathbf{h}_t} l = \sum_{\delta=0}^{T-\delta-1} \nabla_{\mathbf{h}_t} l_{t+\delta}$

BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph



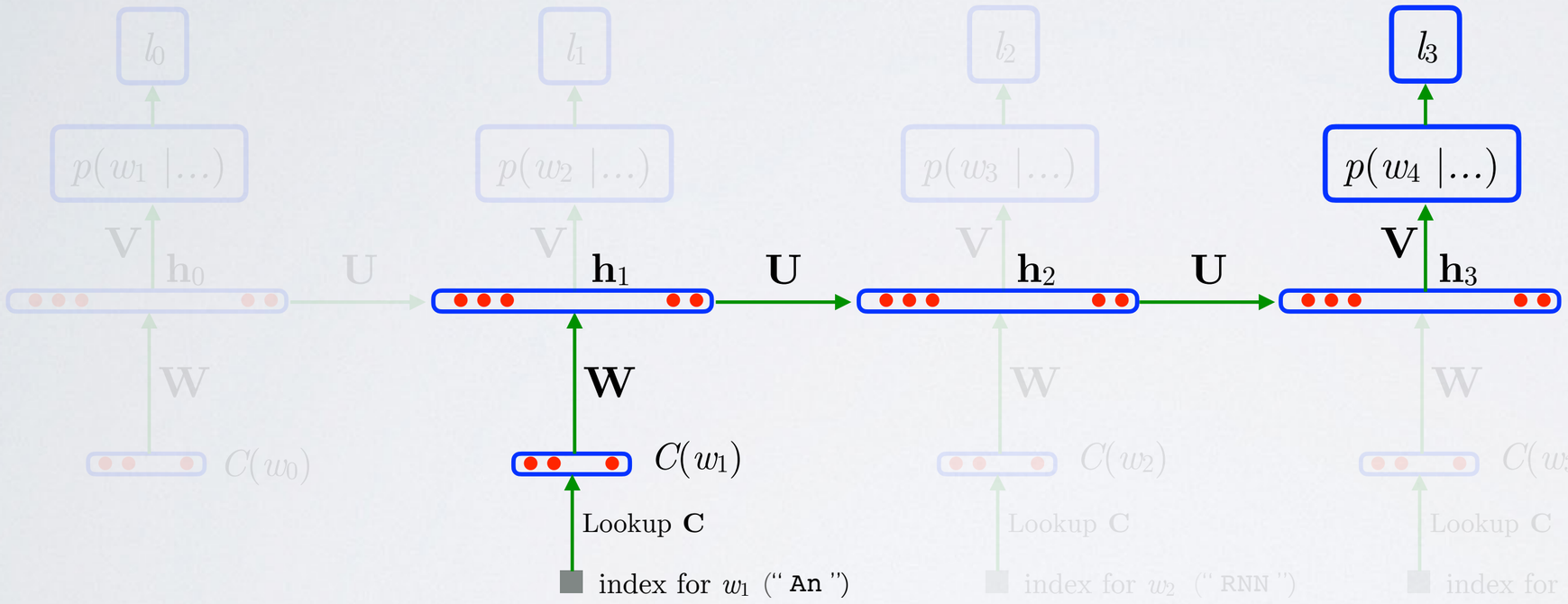
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- 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



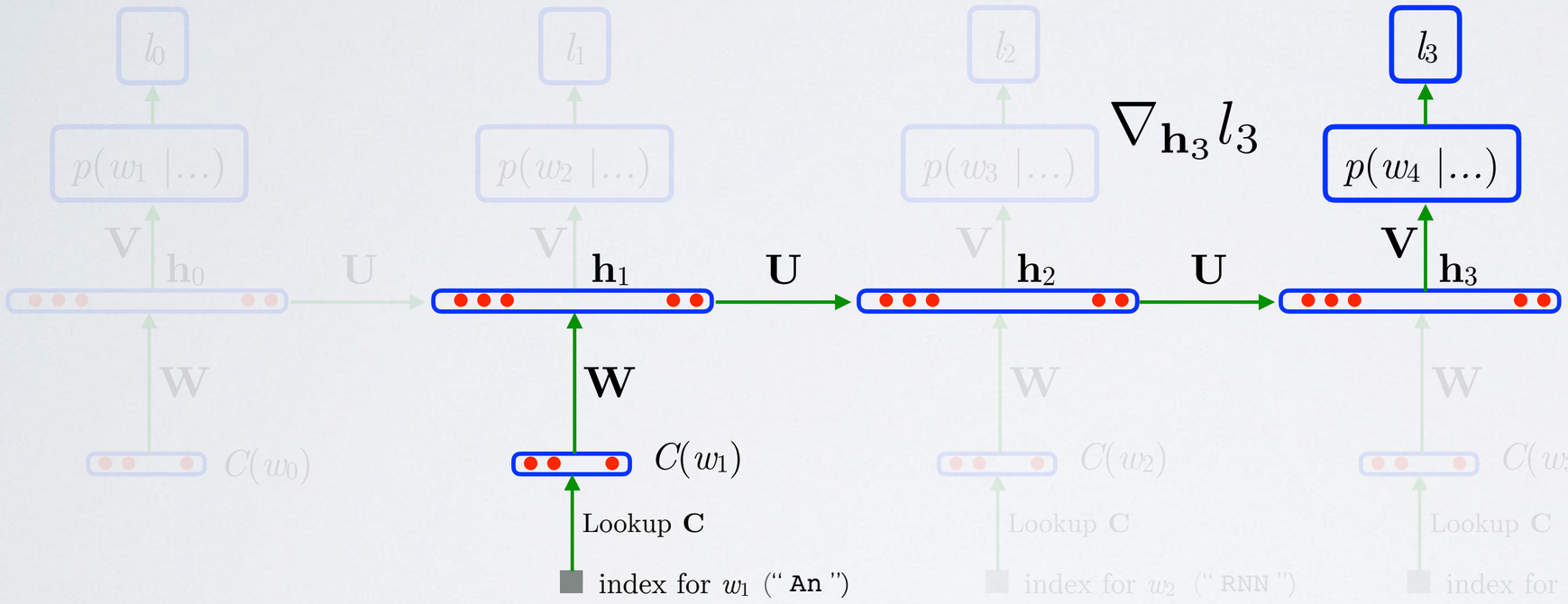
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- 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



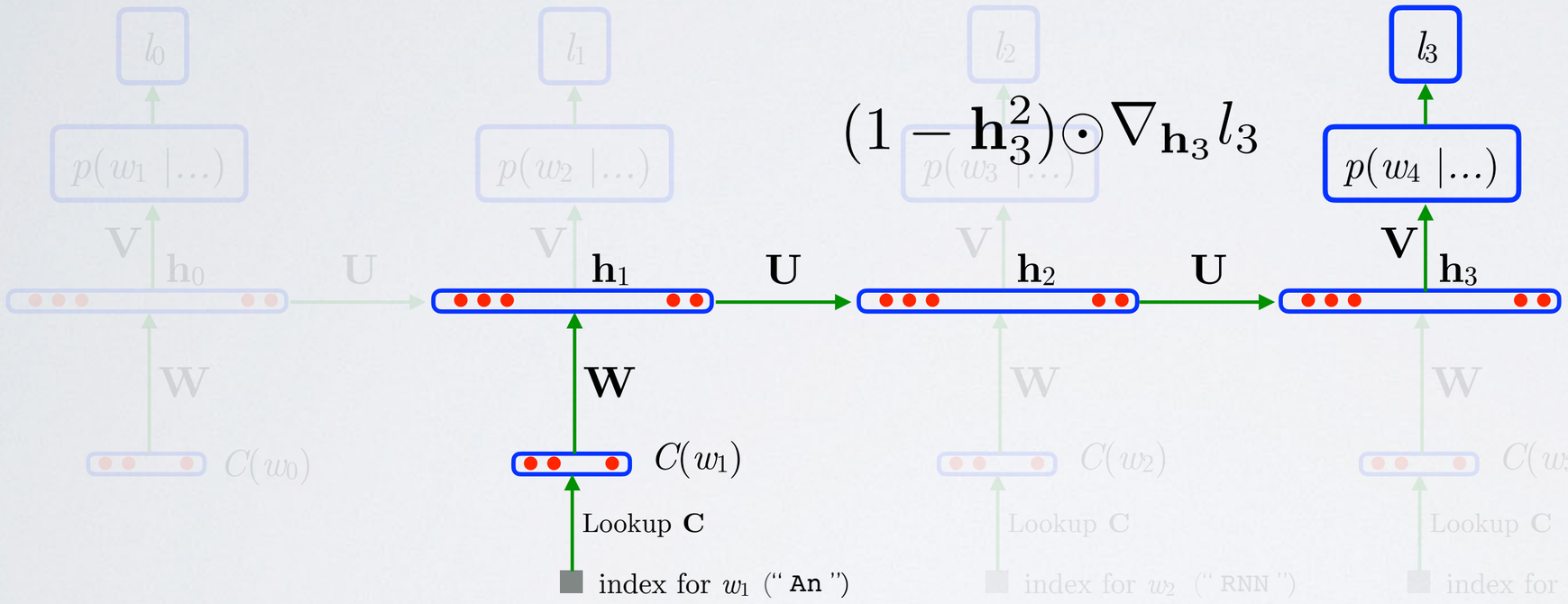
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- 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



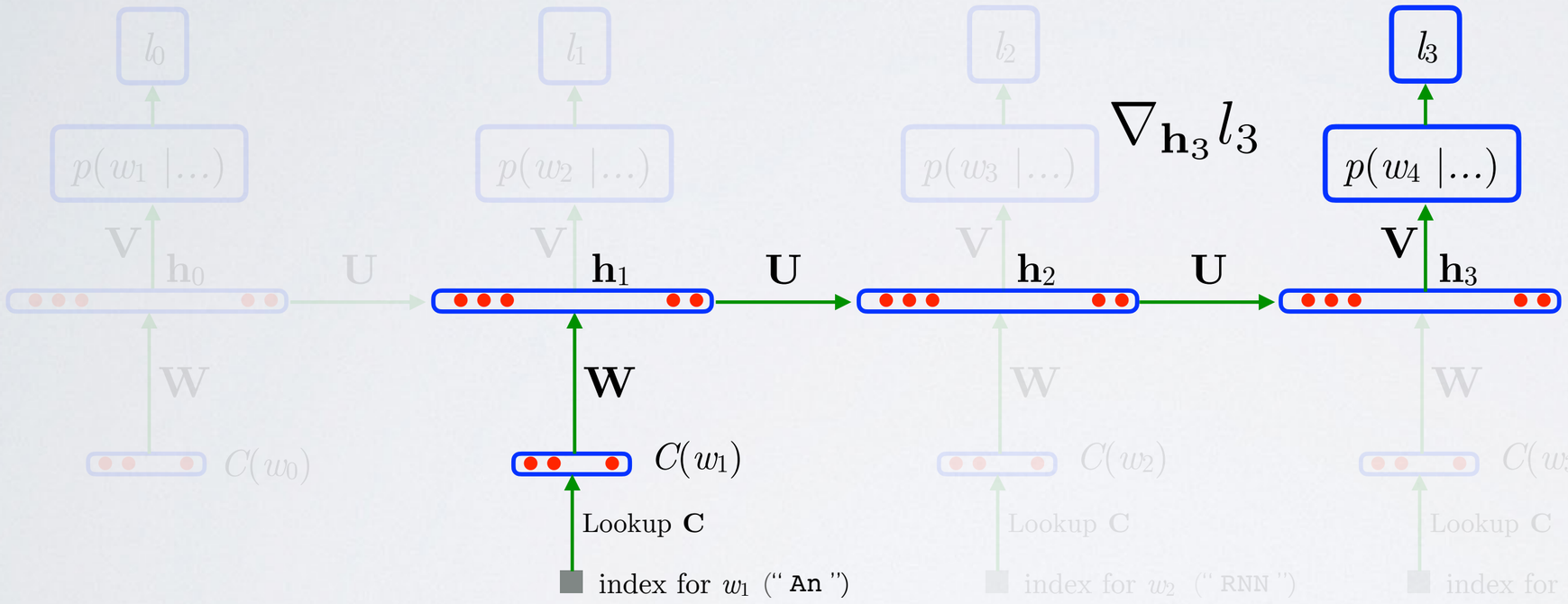
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- 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



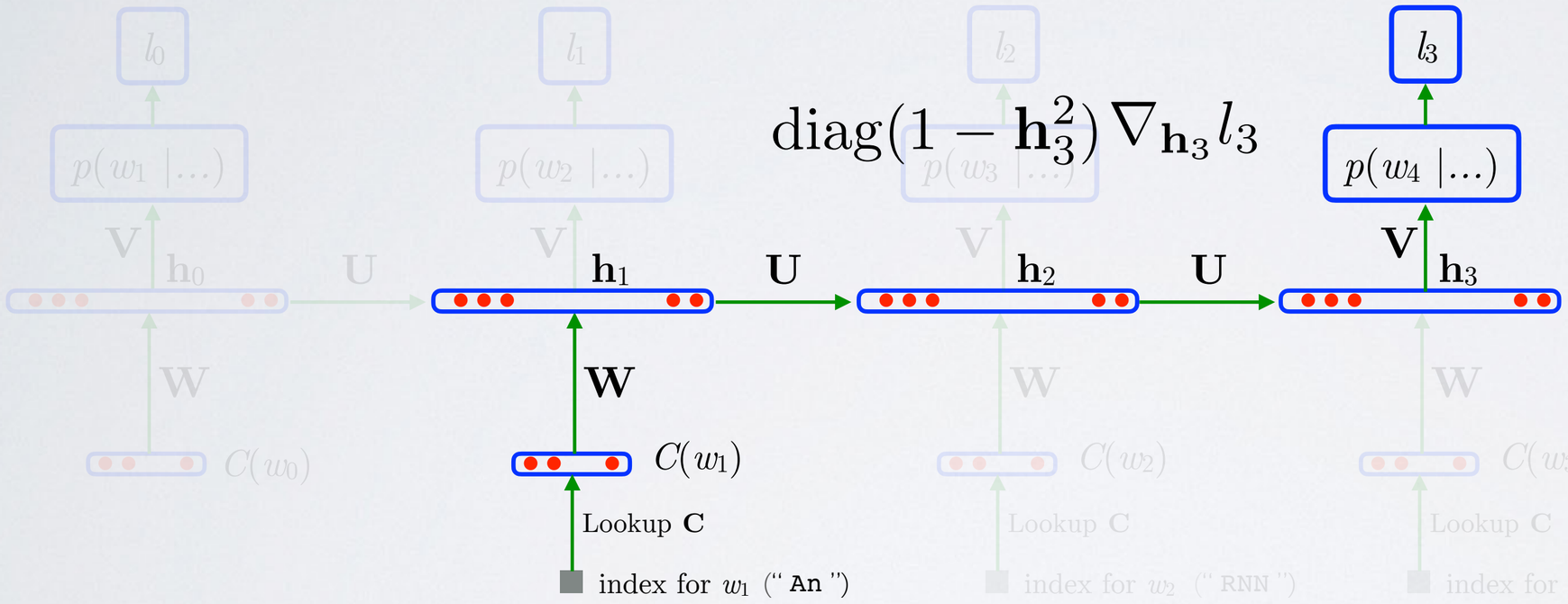
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- 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



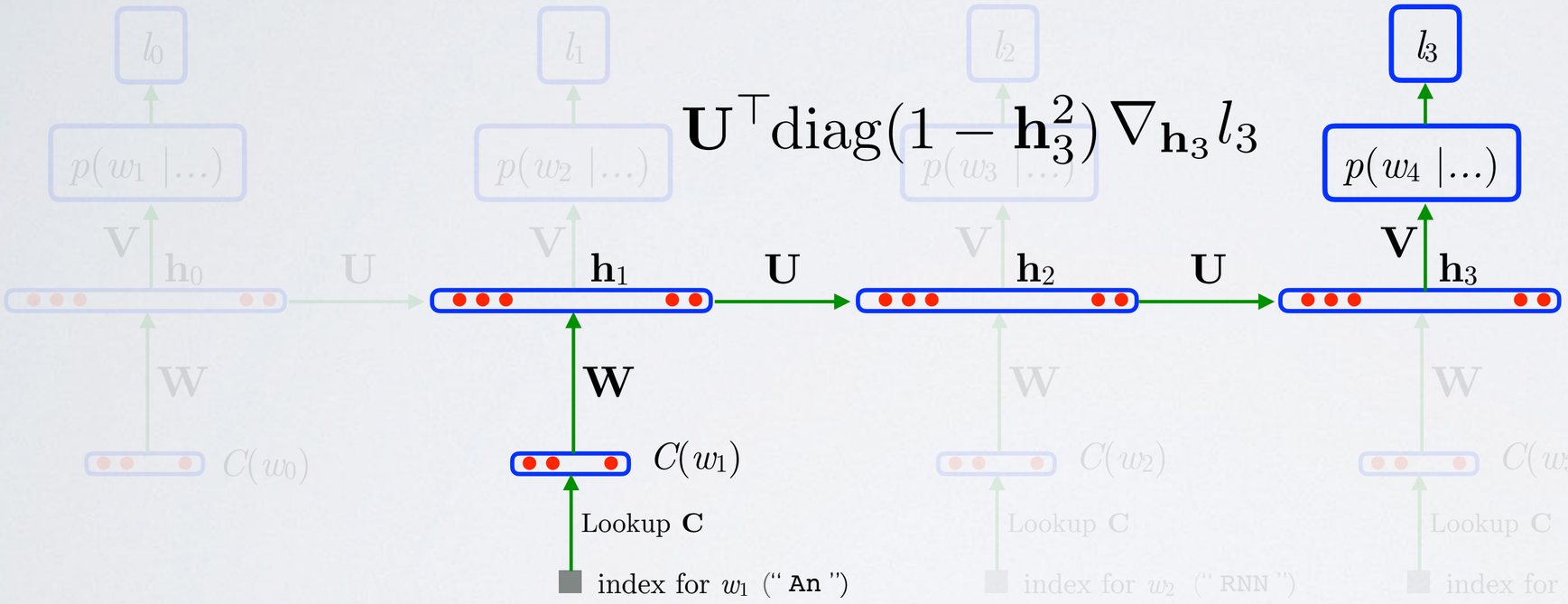
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- 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



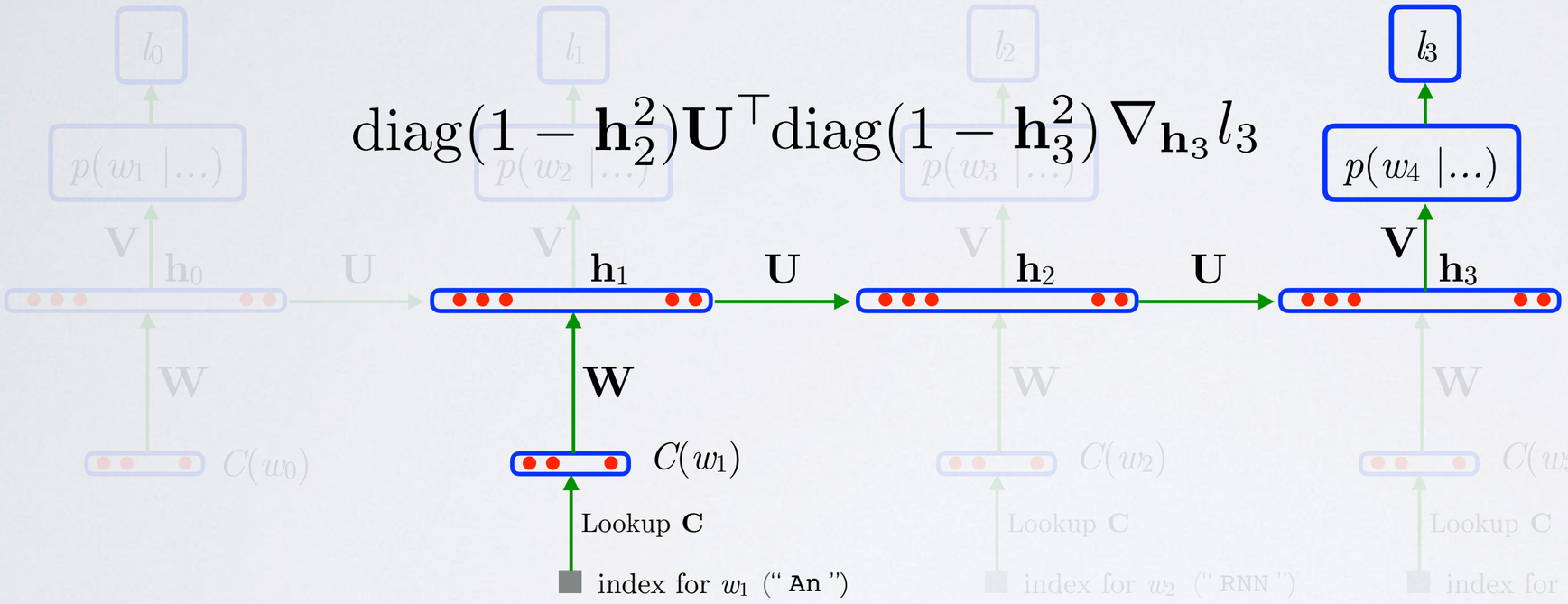
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- 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



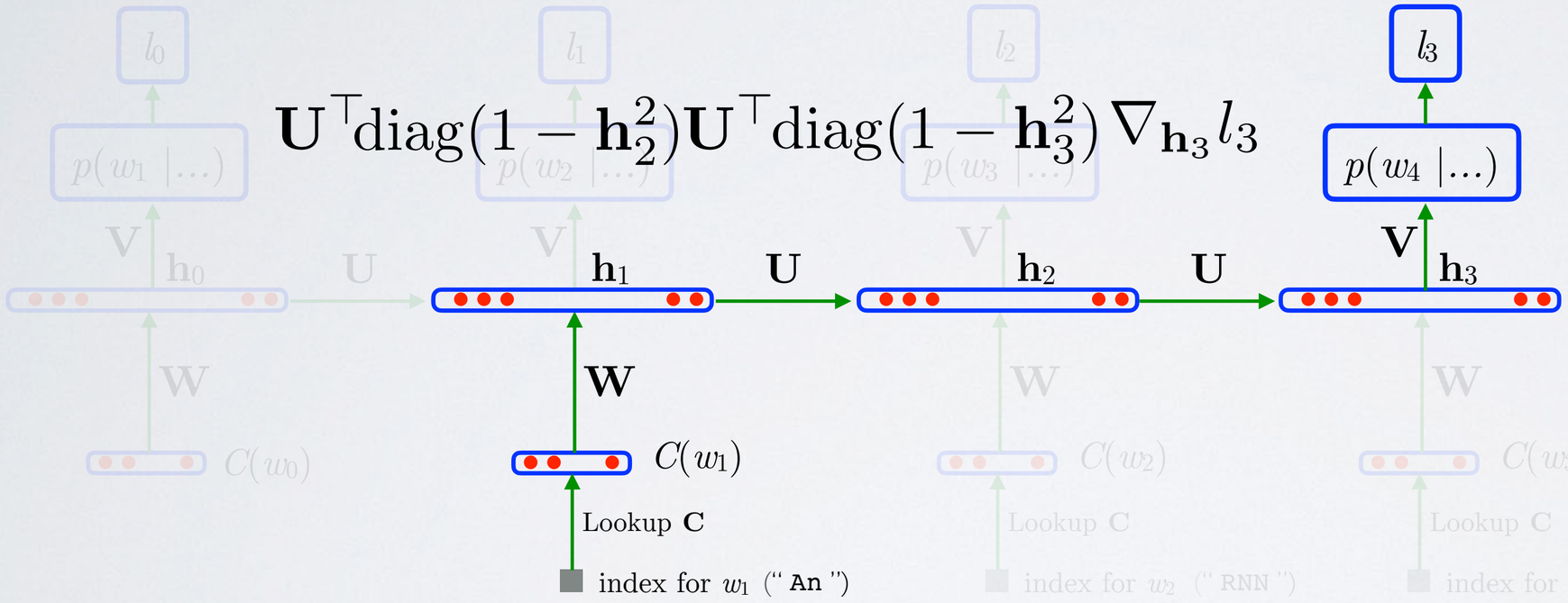
How does $\nabla_{\mathbf{h}_t} l_{t+\delta}$ vary with the time gap δ ?

- 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



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

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

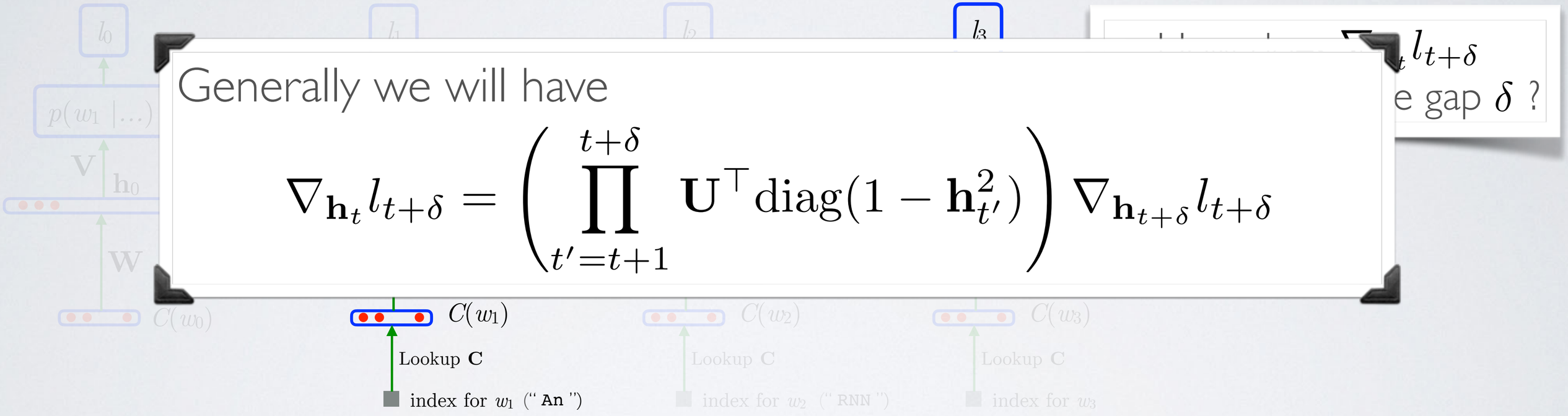
BACKPROPAGATION THROUGH TIME

Topics: backpropagation through time (BPTT)

- Gradients obtained by applying chain rule on unrolled graph

Generally we will have

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



$l_{t+\delta}$

e 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: exploding gradient

Generally we will have

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

- What could go wrong, as δ increases?
  - if $\mathbf{U}$ 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 $\mathbf{W}$ and … $\mathbf{U}$!
  - this is known as the **exploding gradient problem**

BACKPROPAGATION THROUGH TIME

Topics: vanishing gradient

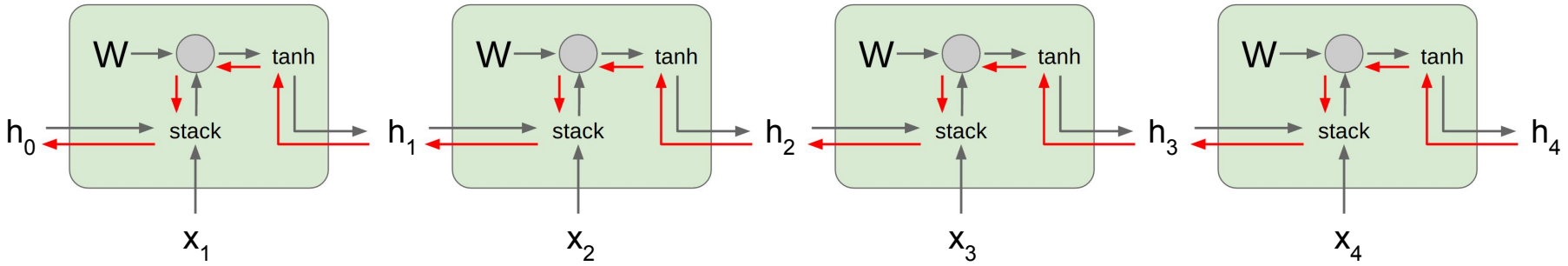
Generally we will have

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

- What could go wrong, as δ increases?
  - if $\mathbf{U}$ 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**

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



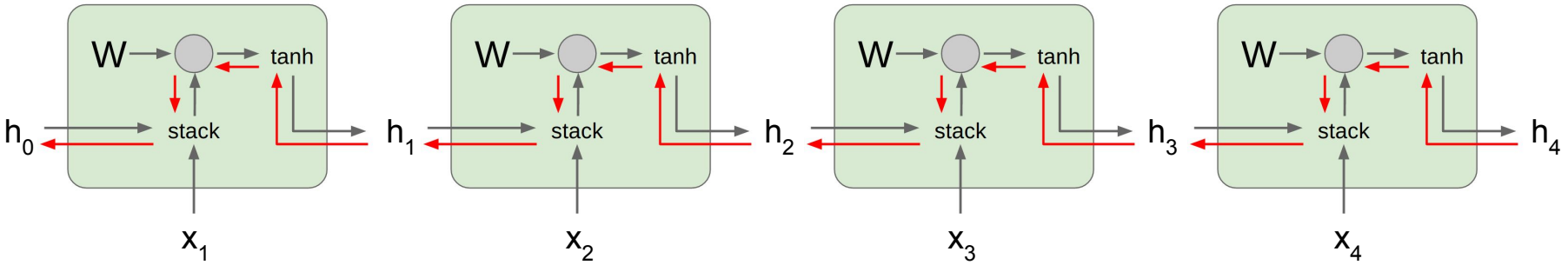
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Largest singular value > 1:
Exploding gradients

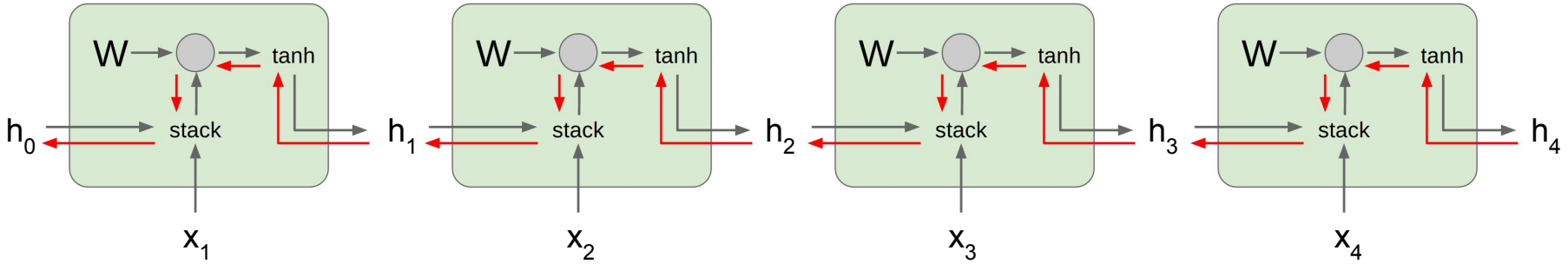
→ **Gradient clipping**: Scale gradient if its norm is too big

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

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

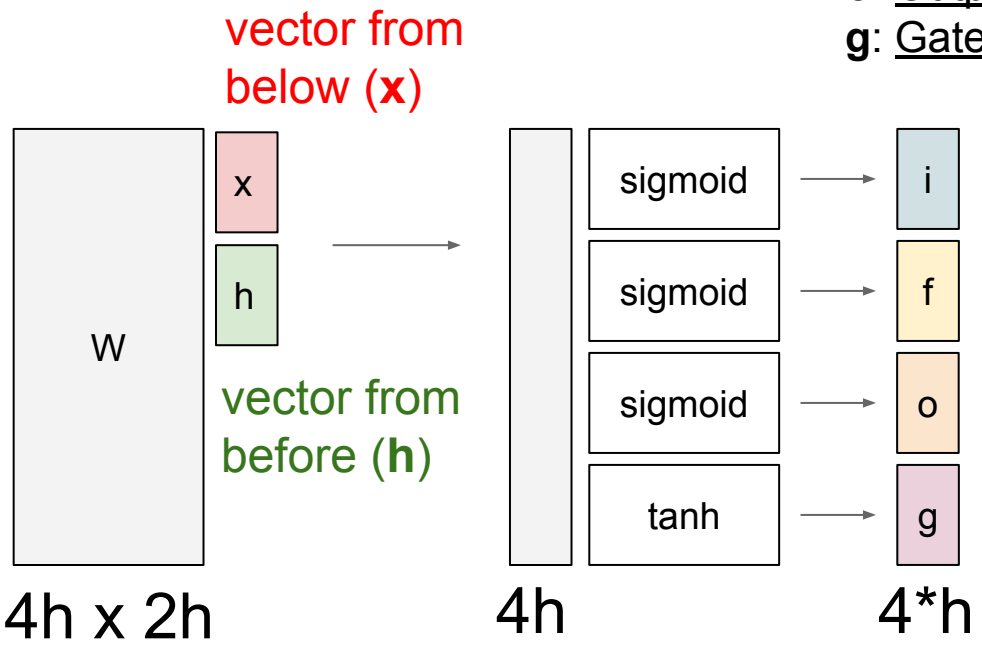
$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell



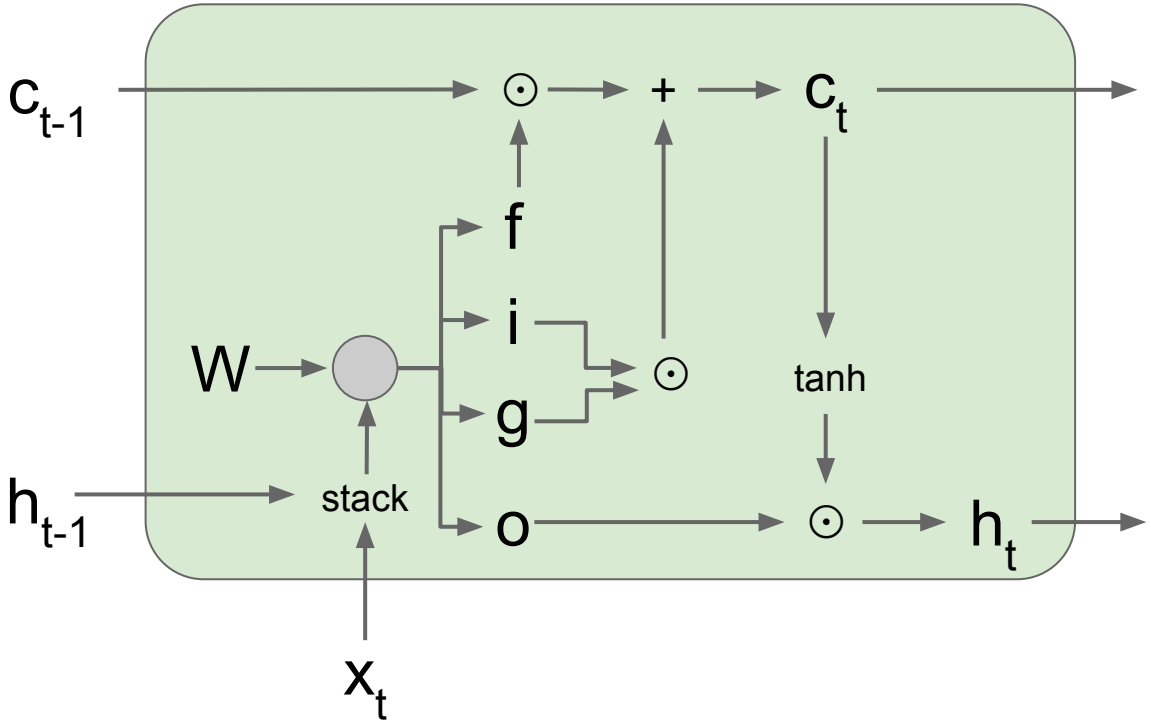
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



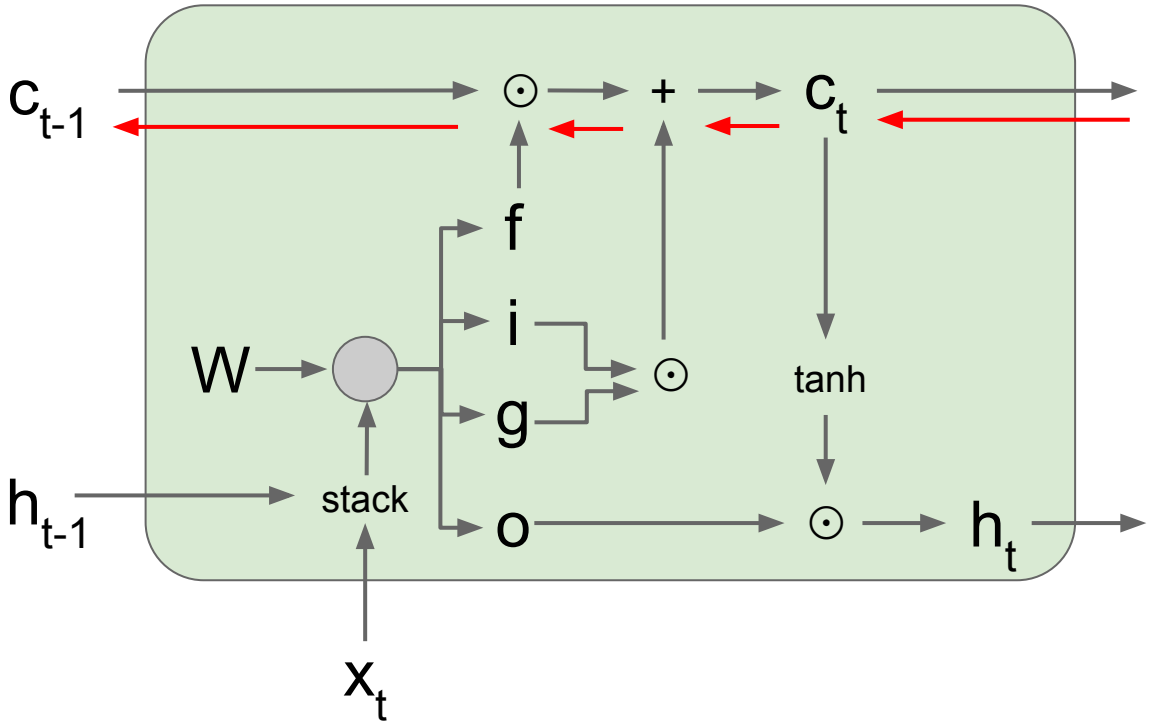
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

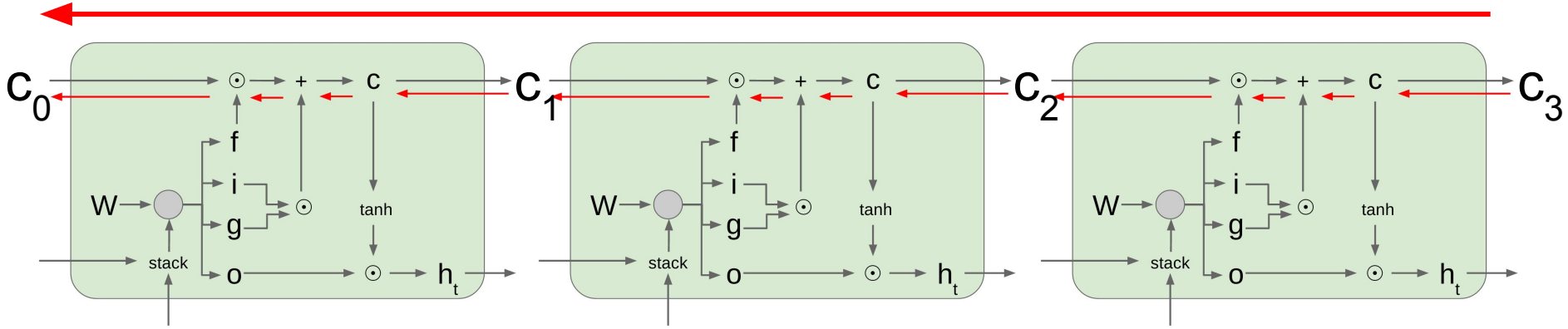
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

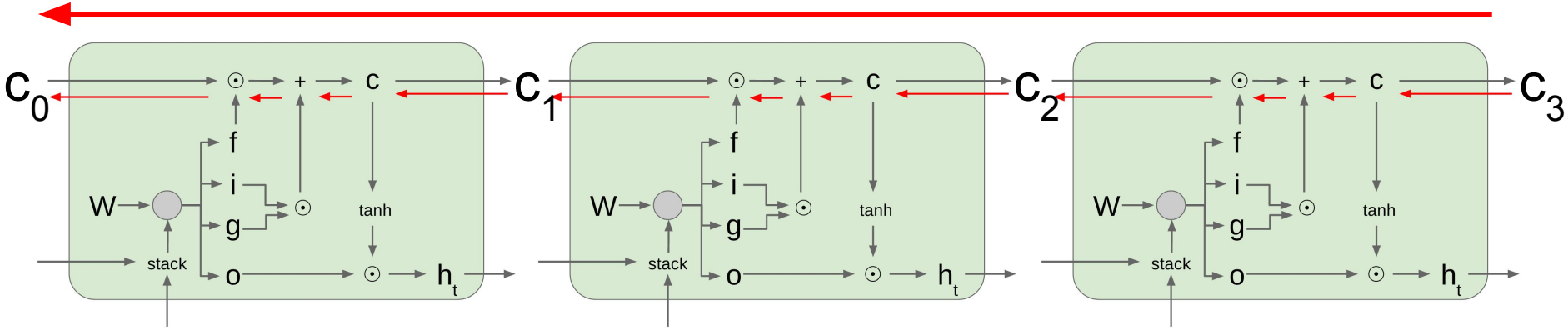
Uninterrupted gradient flow!



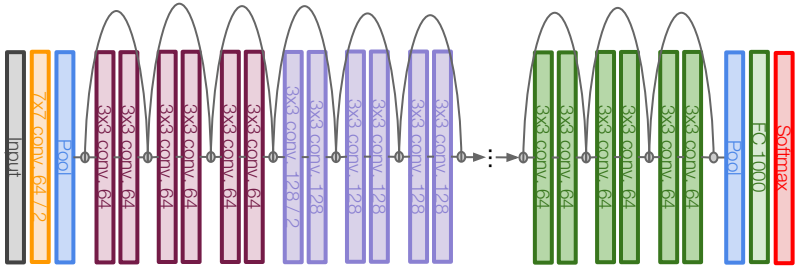
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Similar to ResNet!



In between:

Highway Networks

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

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

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

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

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

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:

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

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

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

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:

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

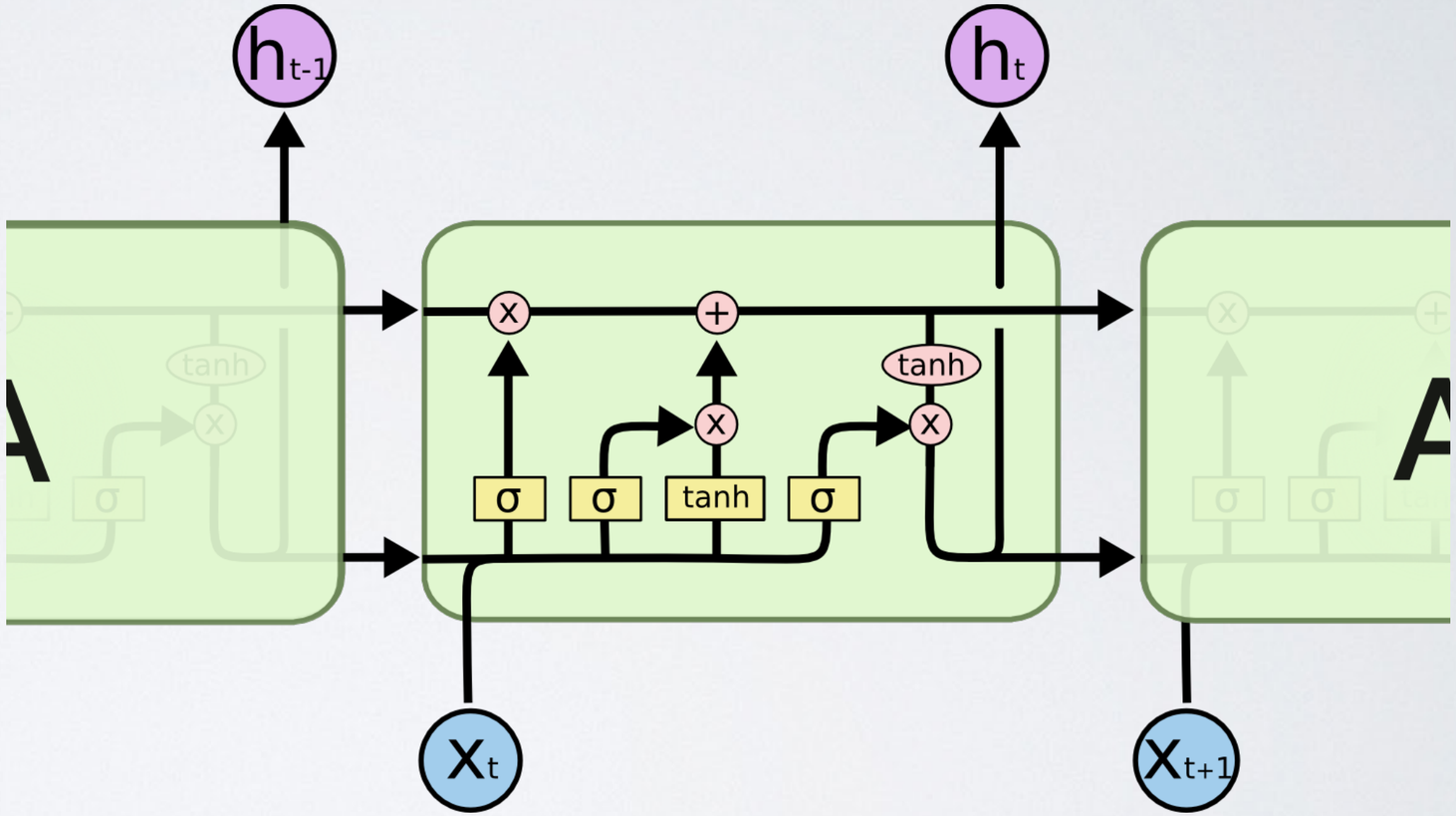


Image from Chris Olah's Blog post on Understanding LSTMs

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:

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

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

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

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:

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

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

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

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:

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

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

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:

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

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

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:

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

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

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:

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

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

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:

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

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

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

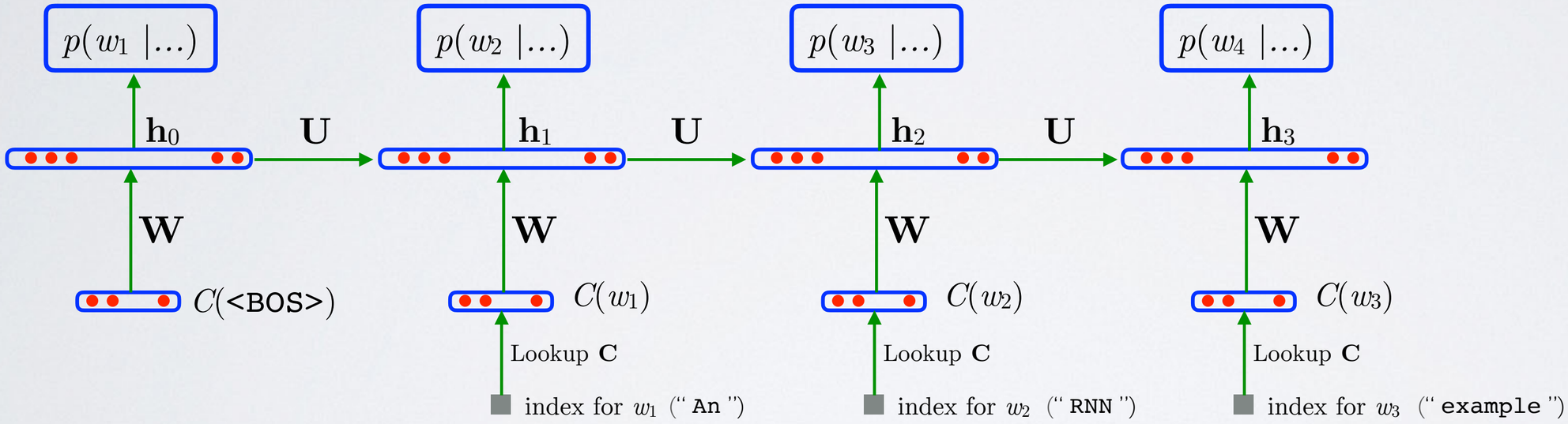
Sequence to sequence learning

RNN LANGUAGE MODEL

REMINDER

Topics: unrolled RNN

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



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!)

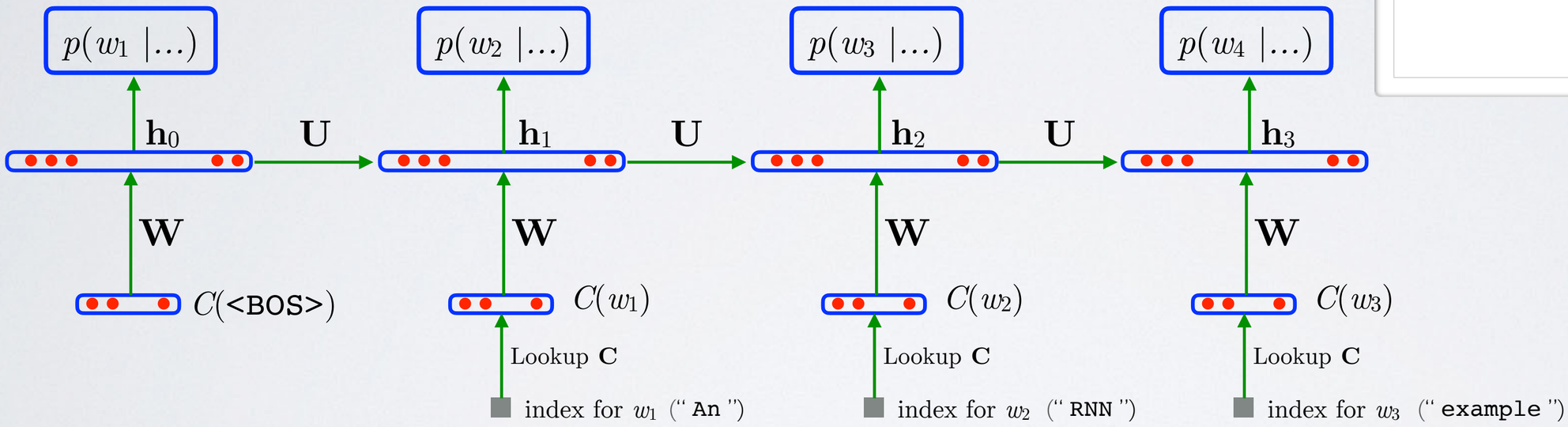
RNN LANGUAGE MODEL

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

SEQUENCE TO SEQUENCE LEARNING

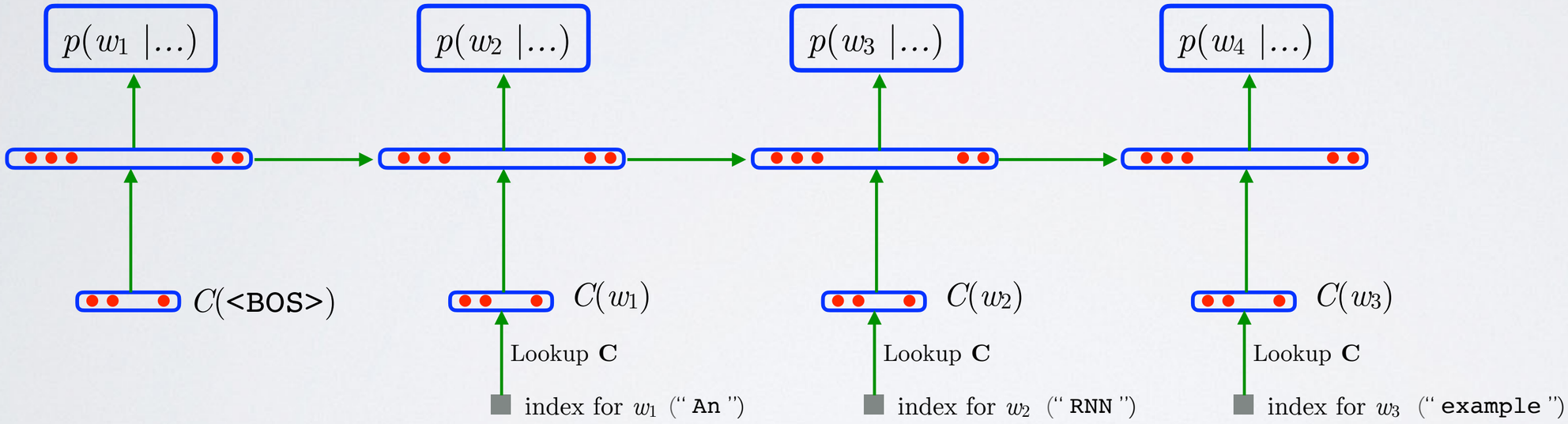
Topics: sequence to sequence (Seq2Seq) learning

- An RNN (LSTM or GRU) gives us good models over a target sequence space $\mathbf{w}$
- What if we wanted to predict the target sequence $\mathbf{w}$ from some input sequence $\mathbf{x}$
  - example application: machine translation
  - can't assume that $\mathbf{w}$ has same size as $\mathbf{x}$ 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

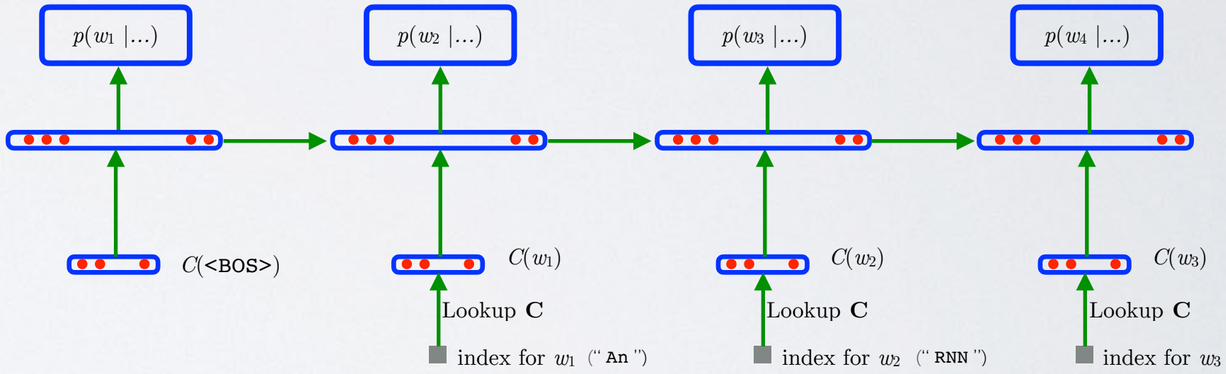
- View of RNN unrolled through time
  - example: $\mathbf{w}$ = [“ `An` ”, “ `RNN` ”, “ `example` ”, “ `.` ”] ($T = 4$)



SEQUENCE TO SEQUENCE LEARNING

Topics: sequence to sequence (Seq2Seq) learning

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

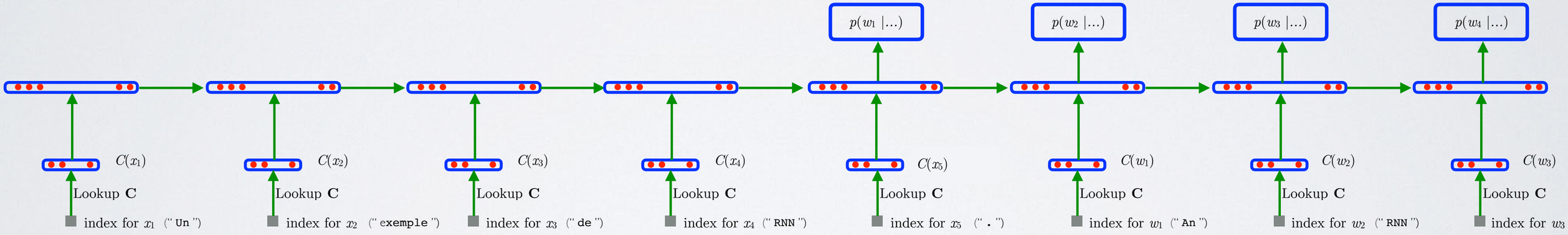


SEQUENCE TO SEQUENCE LEARNING

Topics: sequence to sequence (Seq2Seq) learning

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

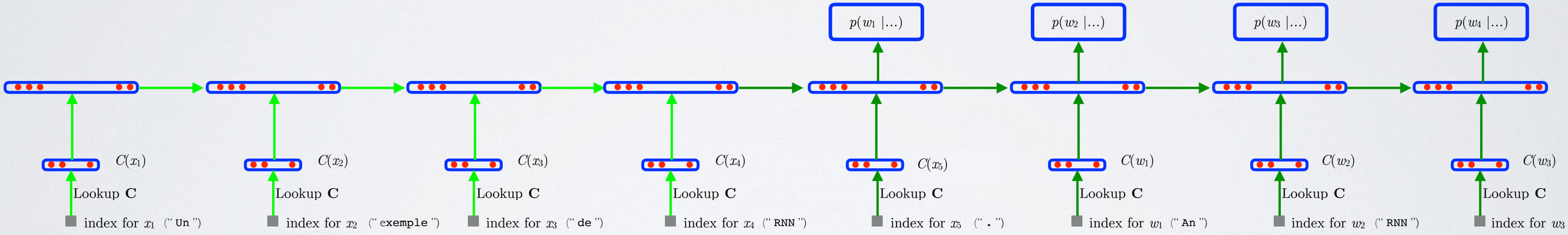
    $\mathbf{x} = [\text{“}\texttt{Un}\text{”}, \text{“}\texttt{exemple}\text{”}, \text{“}\texttt{de}\text{”}, \text{“}\texttt{RNN}\text{”}, \text{“}\texttt{.}\text{”}]\ (T_{\mathbf{x}} = 5)$



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` ", " `.` "] ($T_{\mathbf{x}} = 5$)



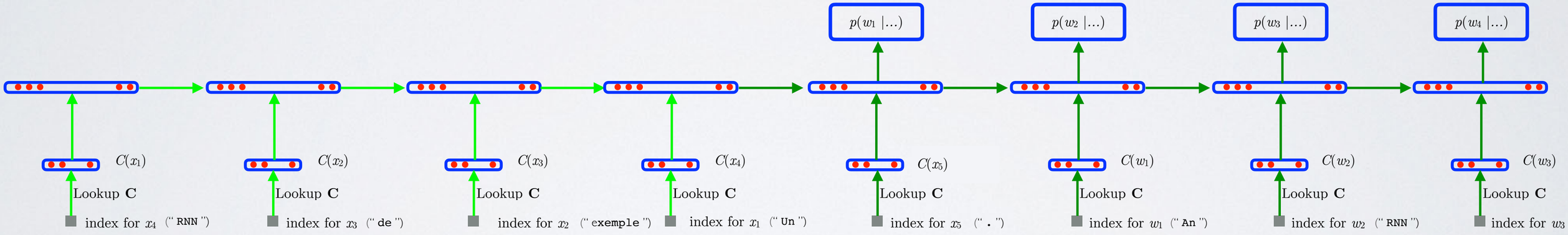
  - may work better by using different RNN parameters to process $\mathbf{x}$

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` ”, “ `.` ”] ($T_{\mathbf{x}} = 5$)



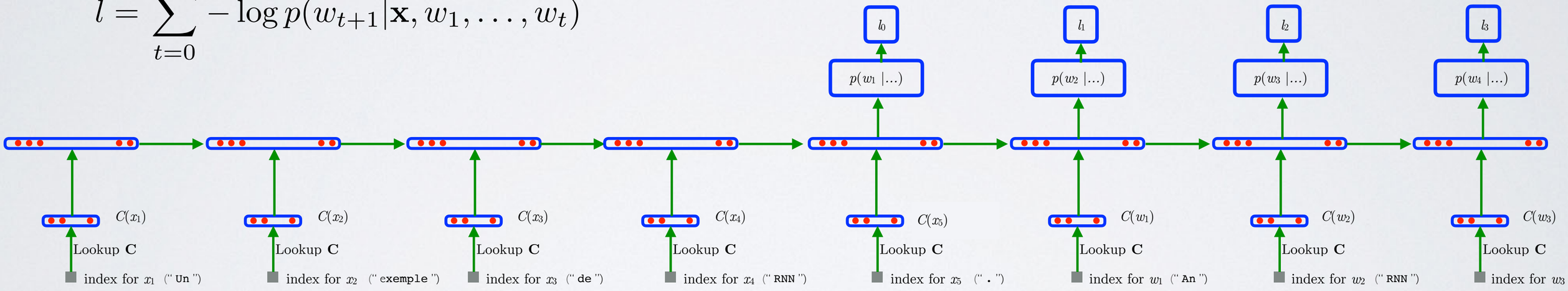
  - may work better by using different RNN parameters to process $\mathbf{x}$
  - may work better by processing sequence $\mathbf{x}$ in reverse order

SEQUENCE TO SEQUENCE LEARNING

Topics: training

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

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

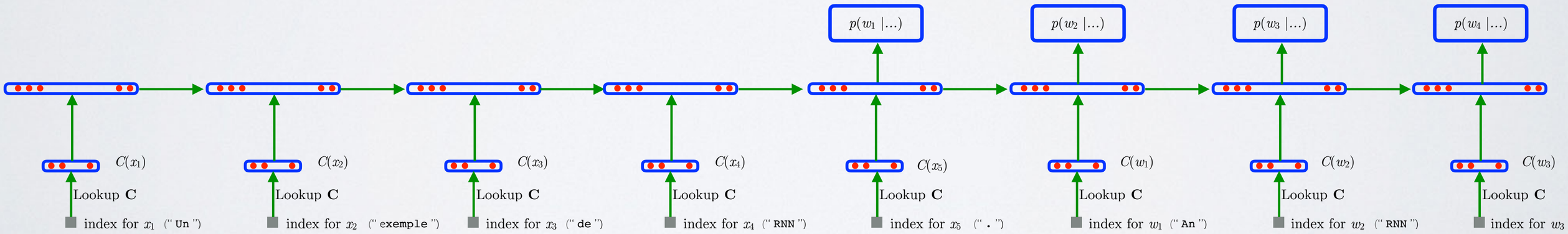


  - in practice, group examples into mini-batches of sequences with similar sizes

SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

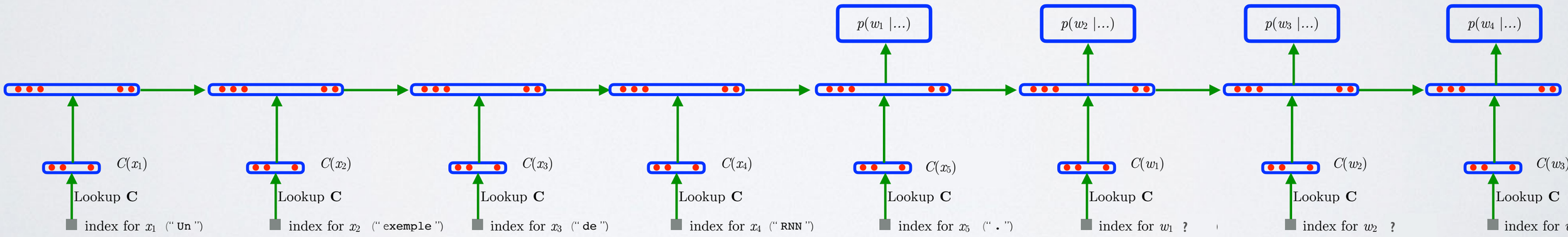
- At test time, must find $\operatorname{argmax} p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain k best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
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

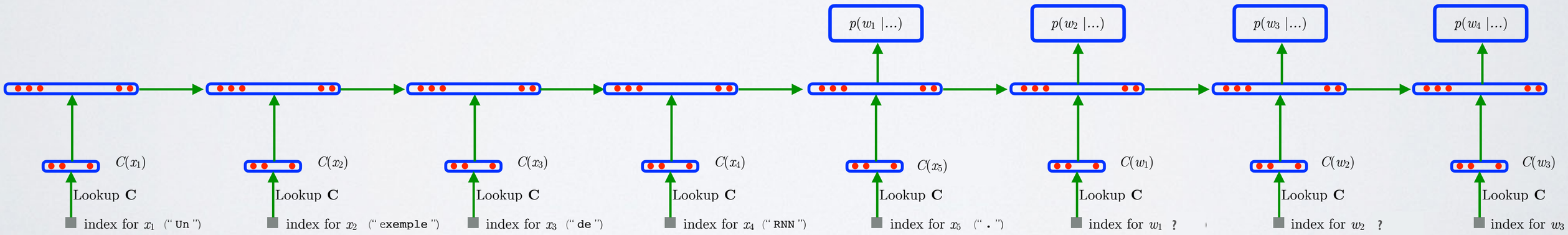


SEQUENCE TO SEQUENCE LEARNING

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 $k=2$ {

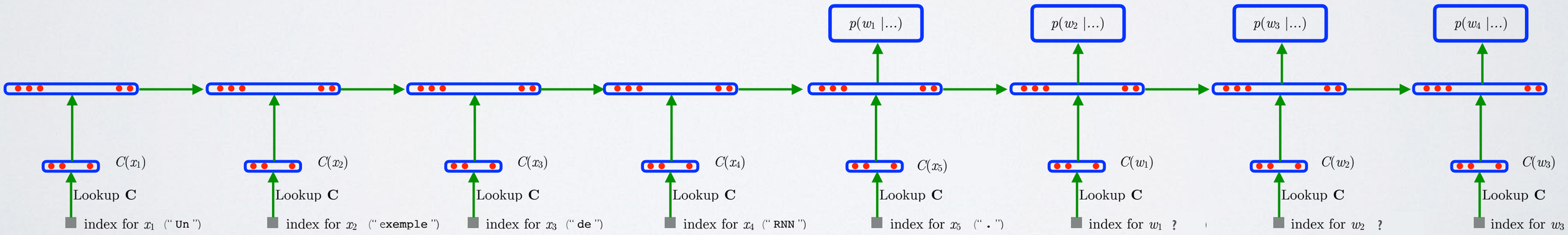


SEQUENCE TO SEQUENCE LEARNING

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. …

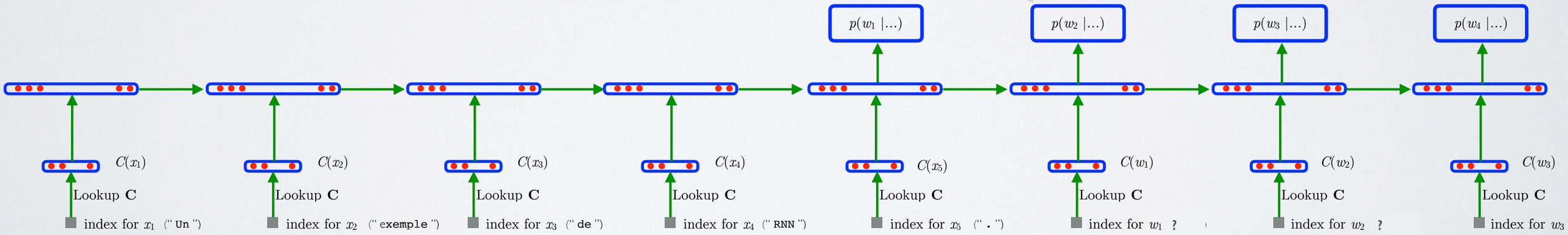


SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

- At test time, must find $\operatorname{argmax} p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- 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

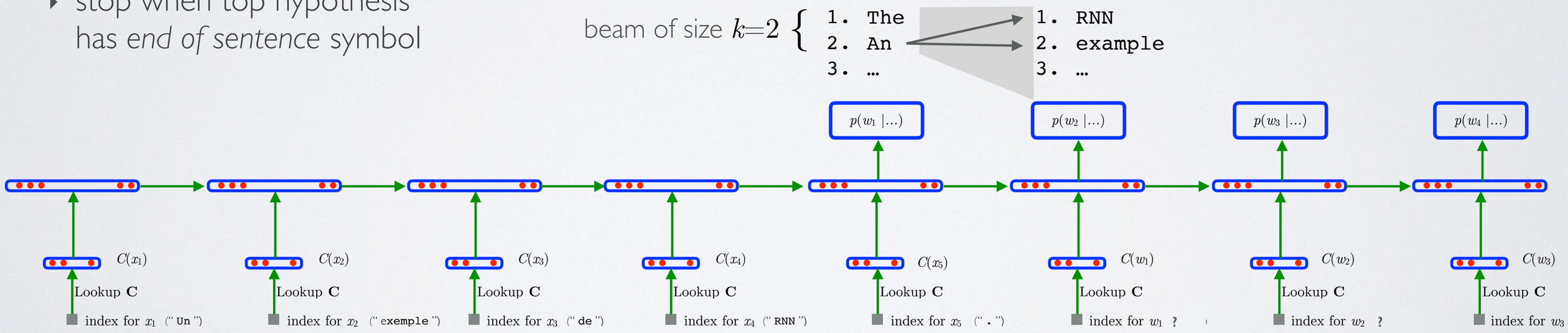
beam of size k=2 {
1. The
2. An
3. …



SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

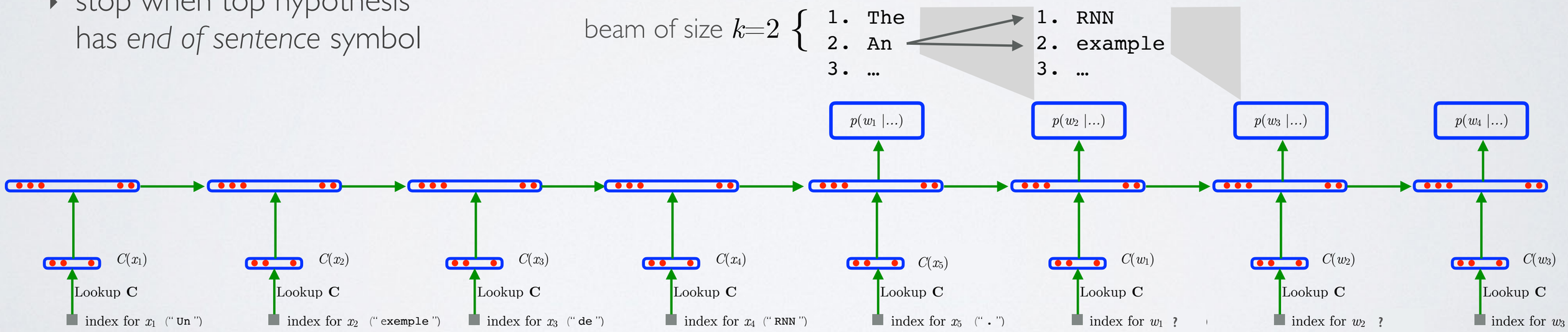
- At test time, must find $\operatorname{argmax} p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain k best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
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



SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

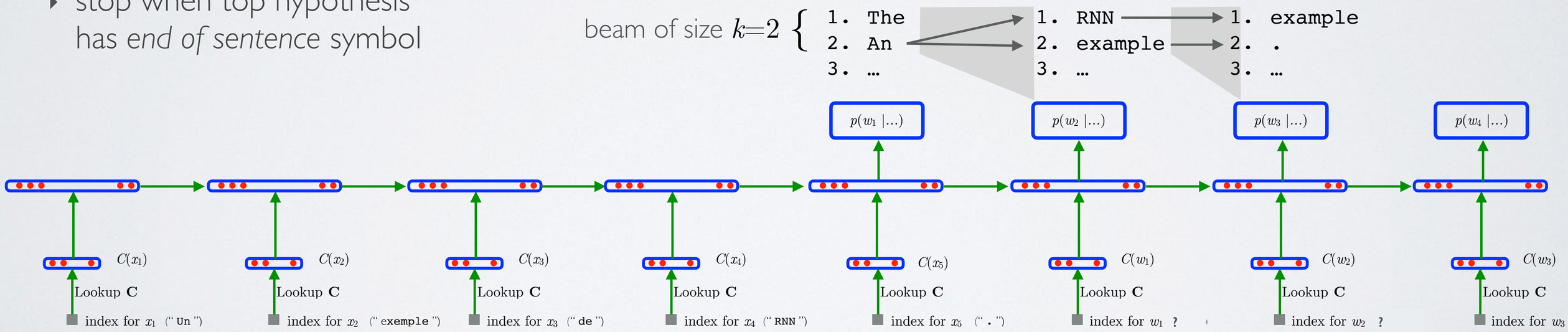
- At test time, must find $\operatorname{argmax} p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain k best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
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



SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

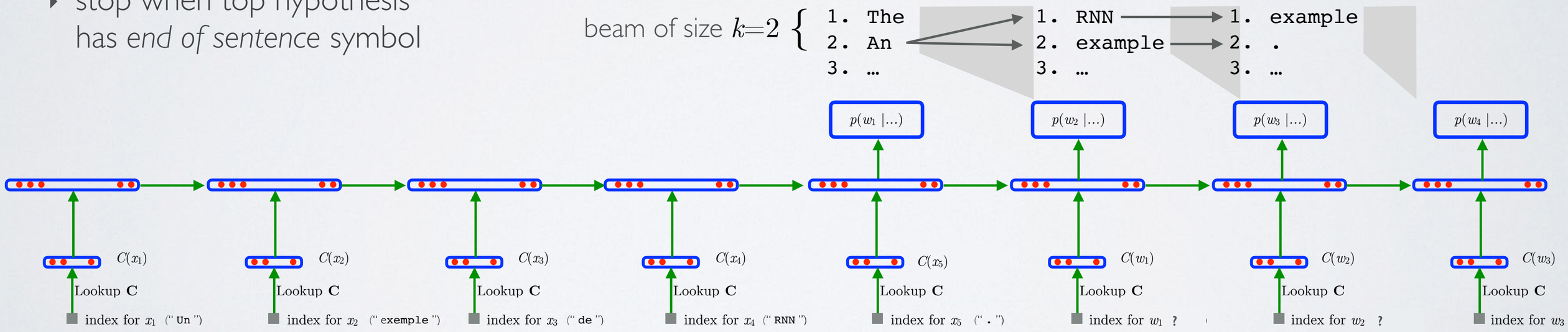
- At test time, must find $\operatorname{argmax} p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain k best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
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



SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

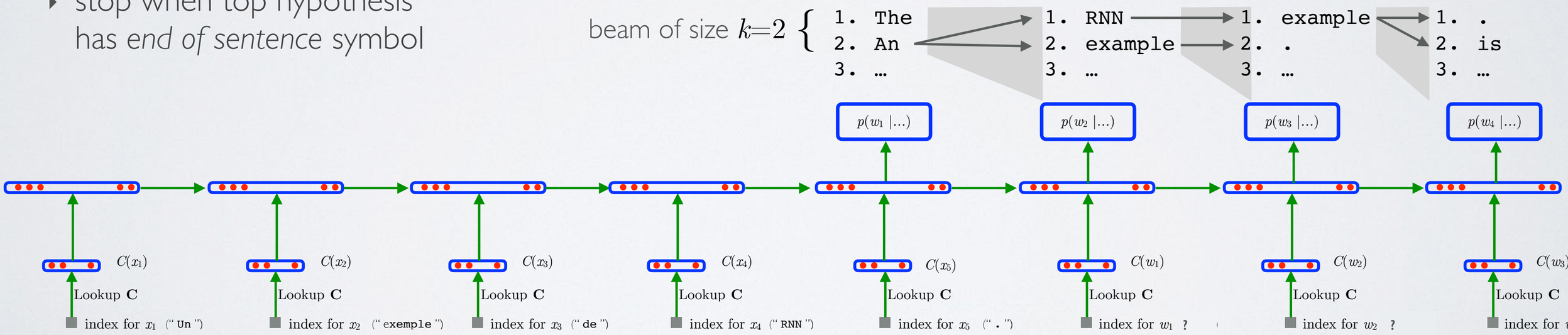
- At test time, must find $\operatorname{argmax} p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain k best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
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



SEQUENCE TO SEQUENCE LEARNING

Topics: beam search

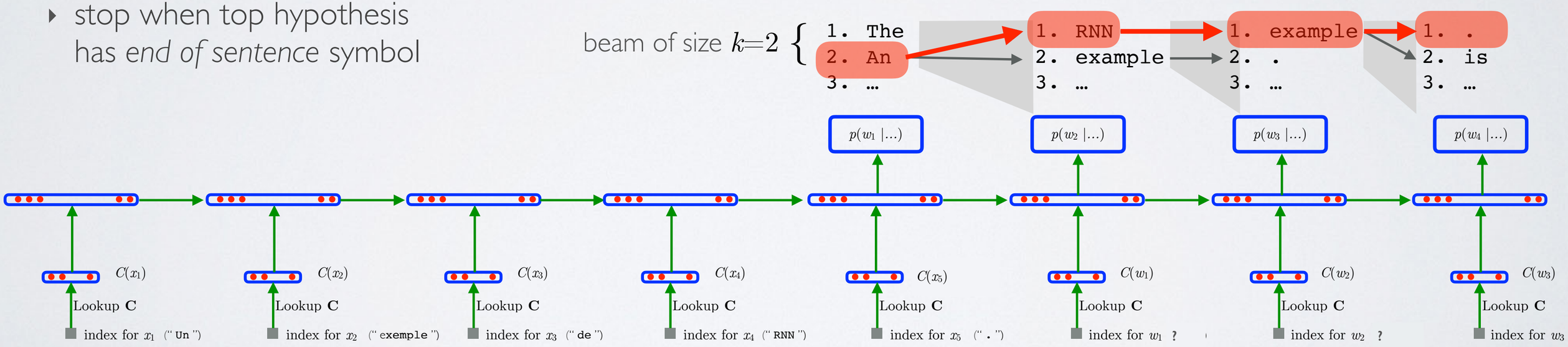
- At test time, must find $\operatorname{argmax} p(\mathbf{w}|\mathbf{x}) = \operatorname{argmax} \sum_{t=0}^{T-1} \log p(w_{t+1}|\mathbf{x}, w_1, \ldots, w_t)$
- Use beam search as approximation
  - maintain k best sequences ("hypotheses")
  - hypotheses ranked based on subsequence log-probability
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

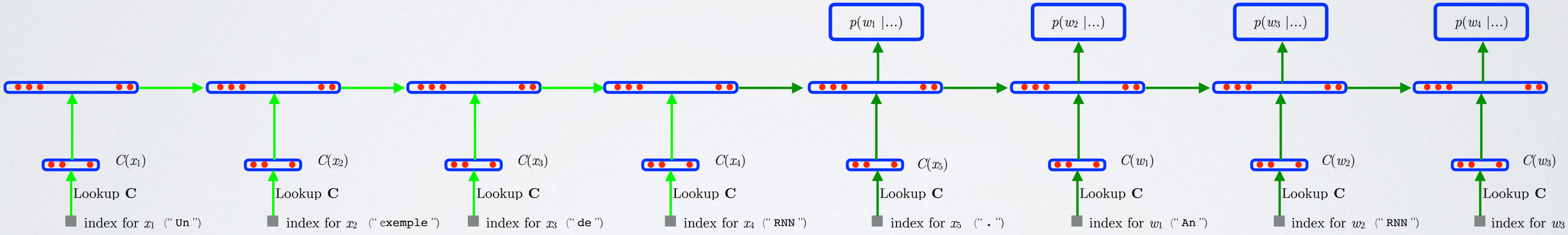
SEQUENCE TO SEQUENCE LEARNING

REMINDER

Topics: sequence to sequence (Seq2Seq) learning

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

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



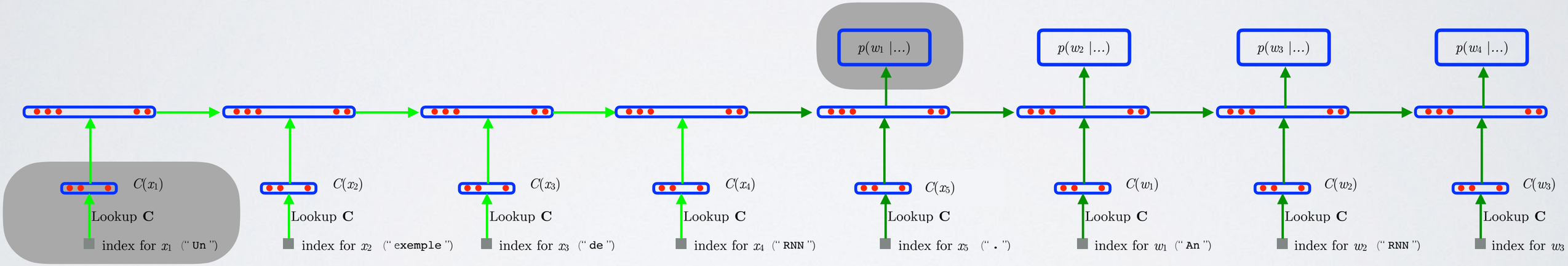
SEQUENCE TO SEQUENCE LEARNING

REMINDER

Topics: sequence to sequence (Seq2Seq) learning

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

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



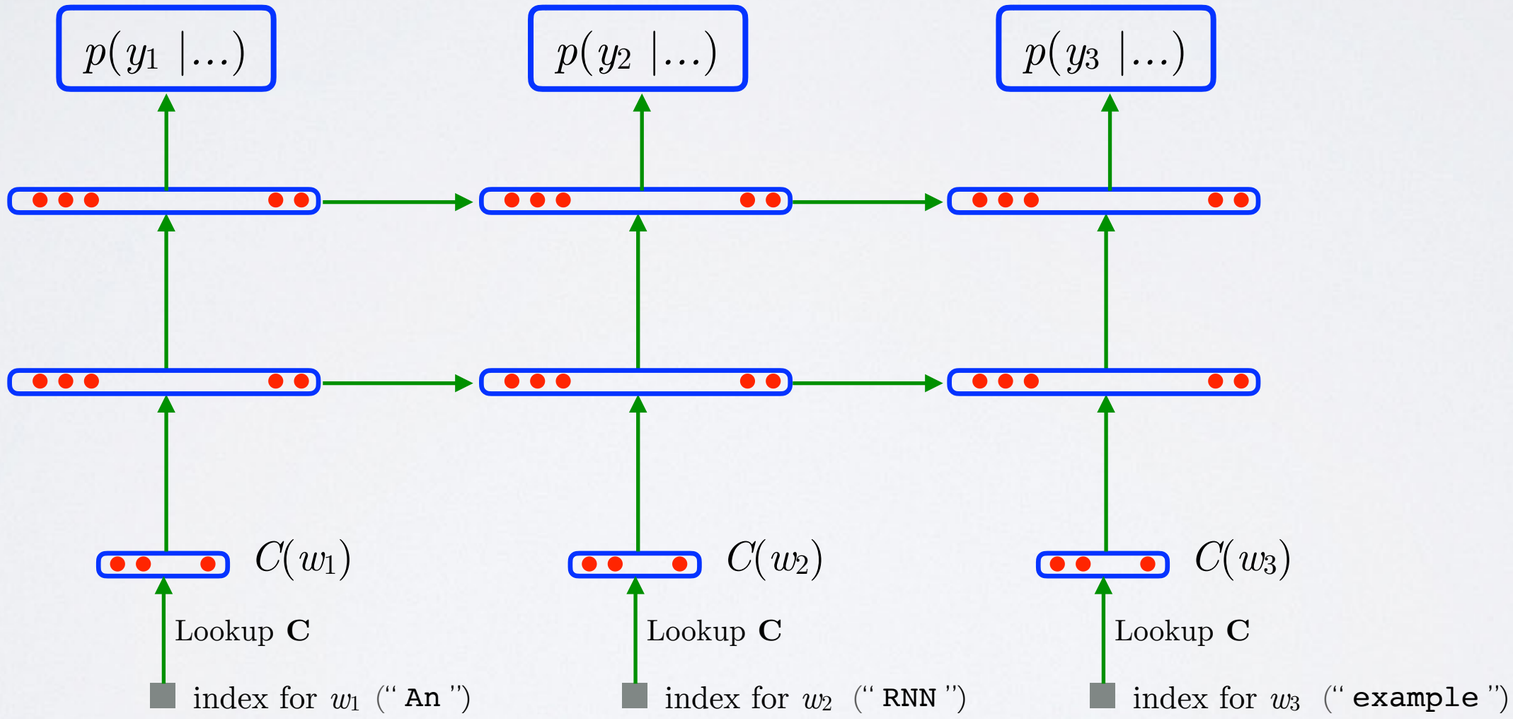
Capturing long-term dependencies is crucial

DEEP RECURRENT NEURAL NETWORK

REMINDER

Topics: Deep RNN

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



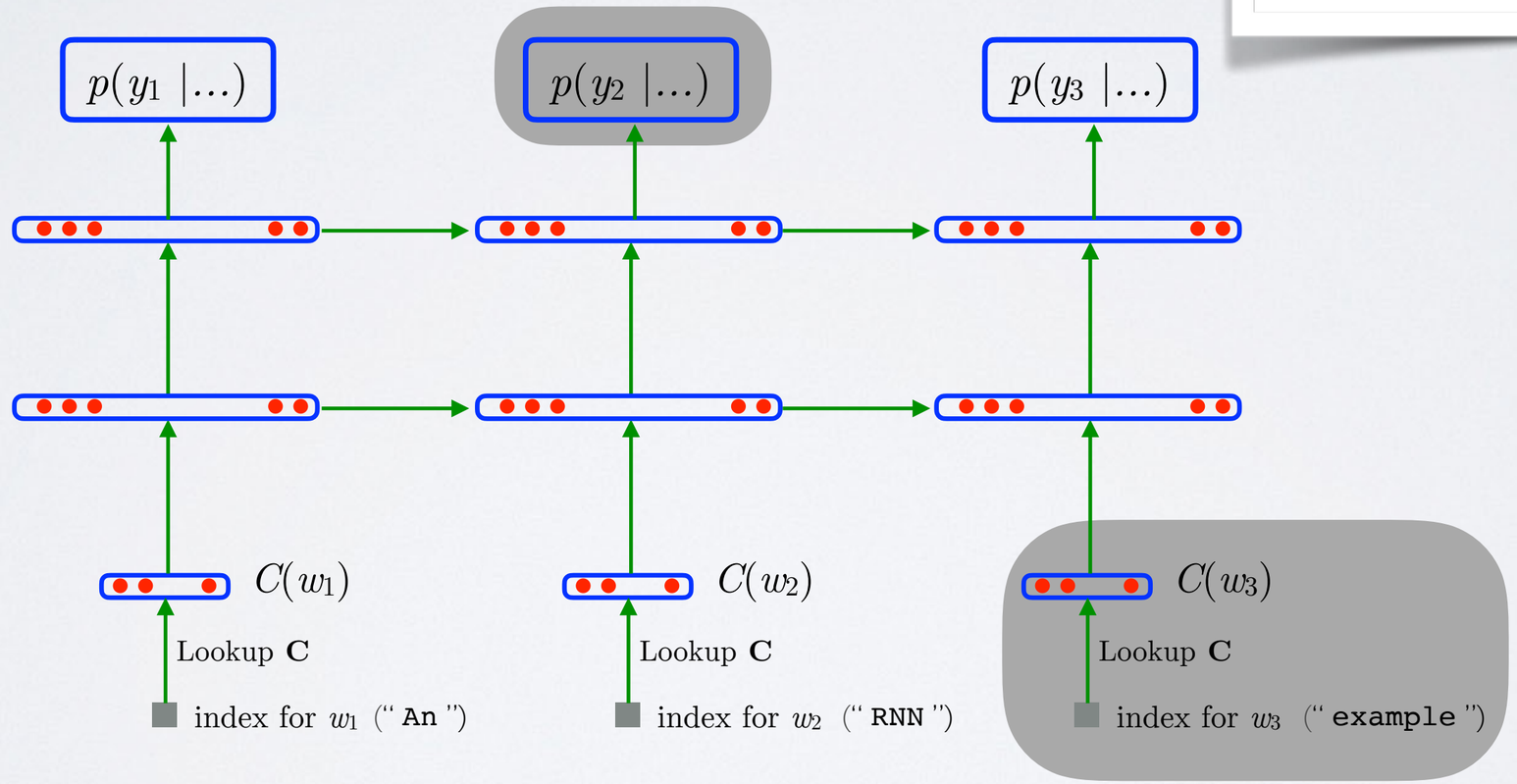
DEEP RECURRENT NEURAL NETWORK

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

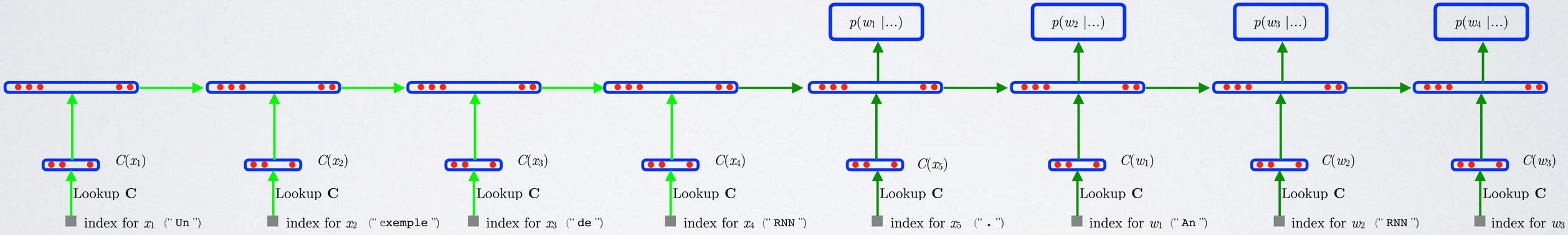
- Bidirectional RNNs exploit this observation
  - have one RNNs traverse the sequence left-to-right
  - have *another* RNN traverse the sequence right-to-left
  - use concatenation of hidden layers as representation

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` ”, “ `.` ”] $(T_{\mathbf{x}} = 5)$

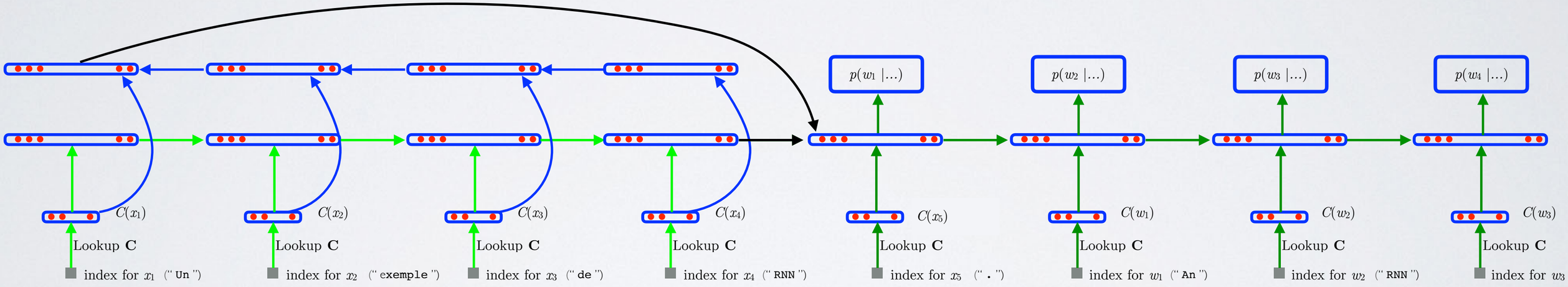


SEQUENCE TO SEQUENCE LEARNING

Topics: bidirectional Seq2Seq

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

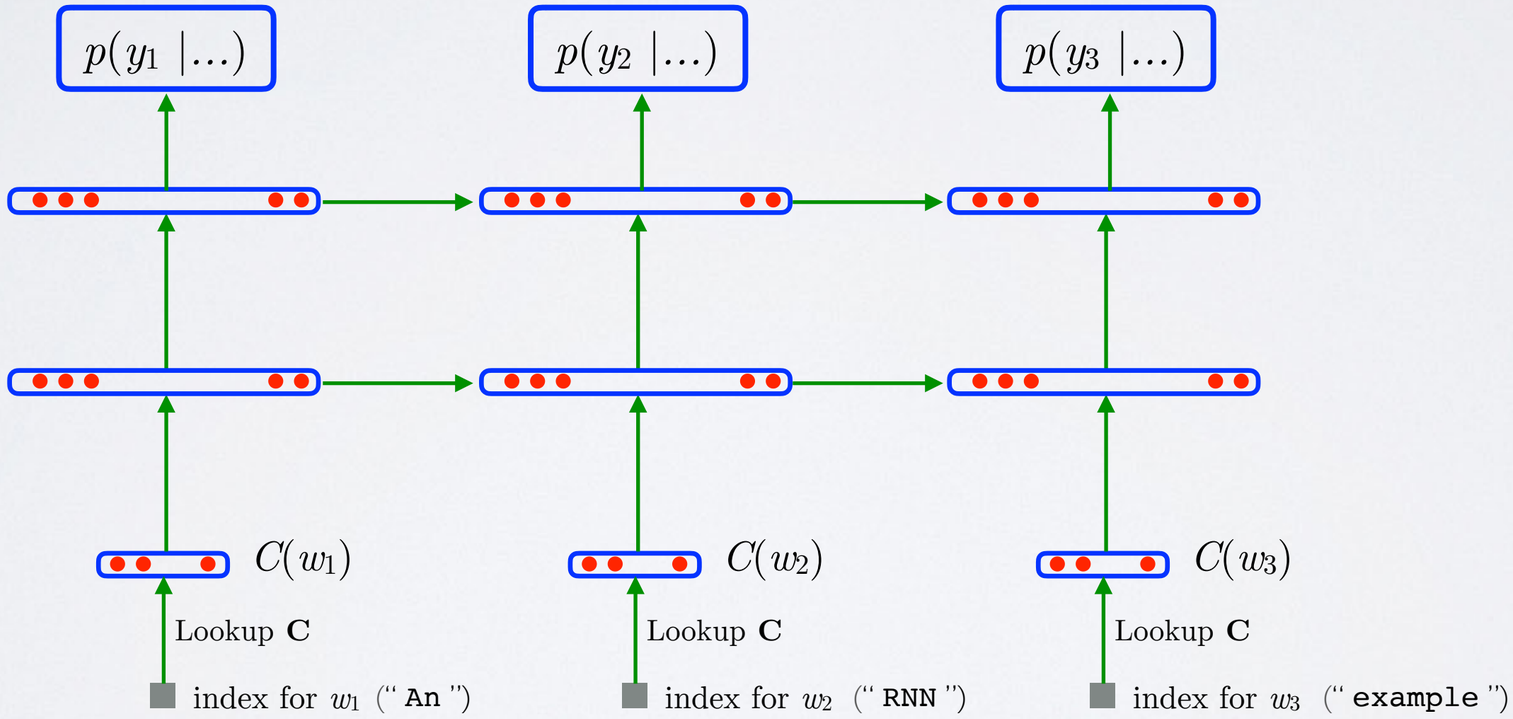
    $\mathbf{x} = [\text{“}\texttt{Un}\text{”}, \text{“}\texttt{exemple}\text{”}, \text{“}\texttt{de}\text{”}, \text{“}\texttt{RNN}\text{”}, \text{“}\texttt{.}\text{”}]\ (T_{\mathbf{x}} = 5)$



DEEP RECURRENT NEURAL NETWORK

Topics: Bidirectional deep RNN

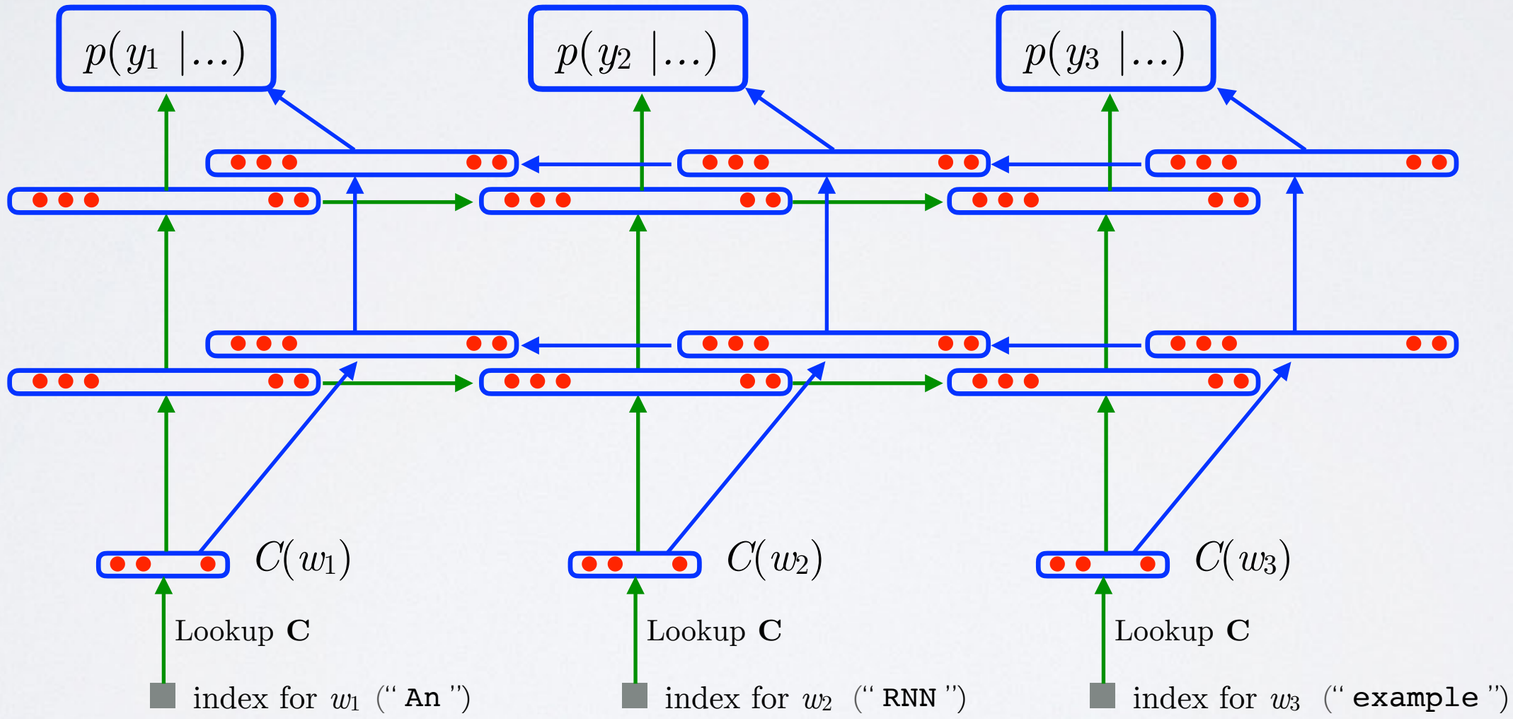
- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)



DEEP RECURRENT NEURAL NETWORK

Topics: Bidirectional deep RNN

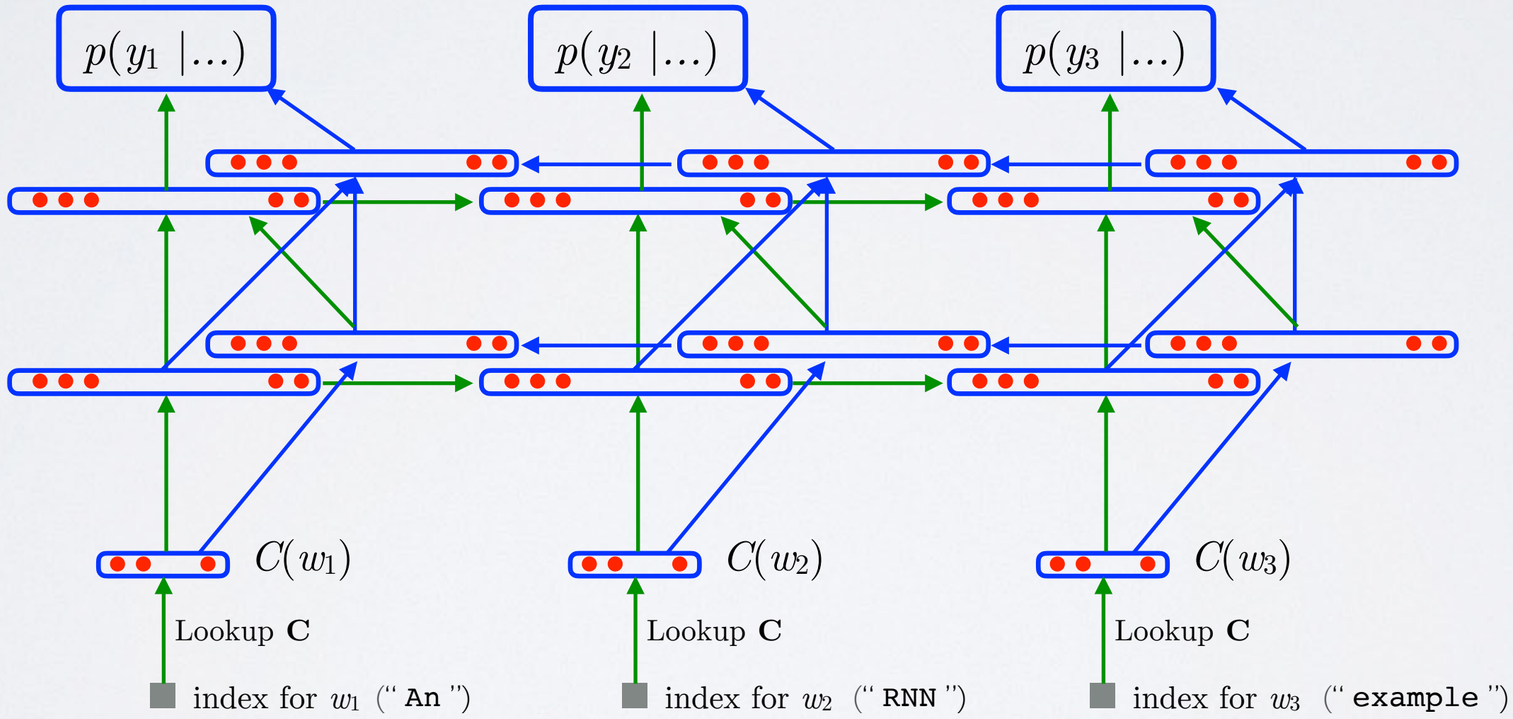
- Useful beyond language modeling
  - word tagging (e.g. part-of-speech tagging, named entity recognition)



DEEP RECURRENT NEURAL NETWORK

Topics: Bidirectional deep RNN

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



RECURSIVE NEURAL NETWORKS

Topics: Recursive NN (Socher, Manning and Ng, ICML 2011)

- Alternative to RNN and convolutional NN for sequence modeling.
- Repeating left/right branching structure: Parameters are shared across layers
- Network depth depends on the length of the sequence.

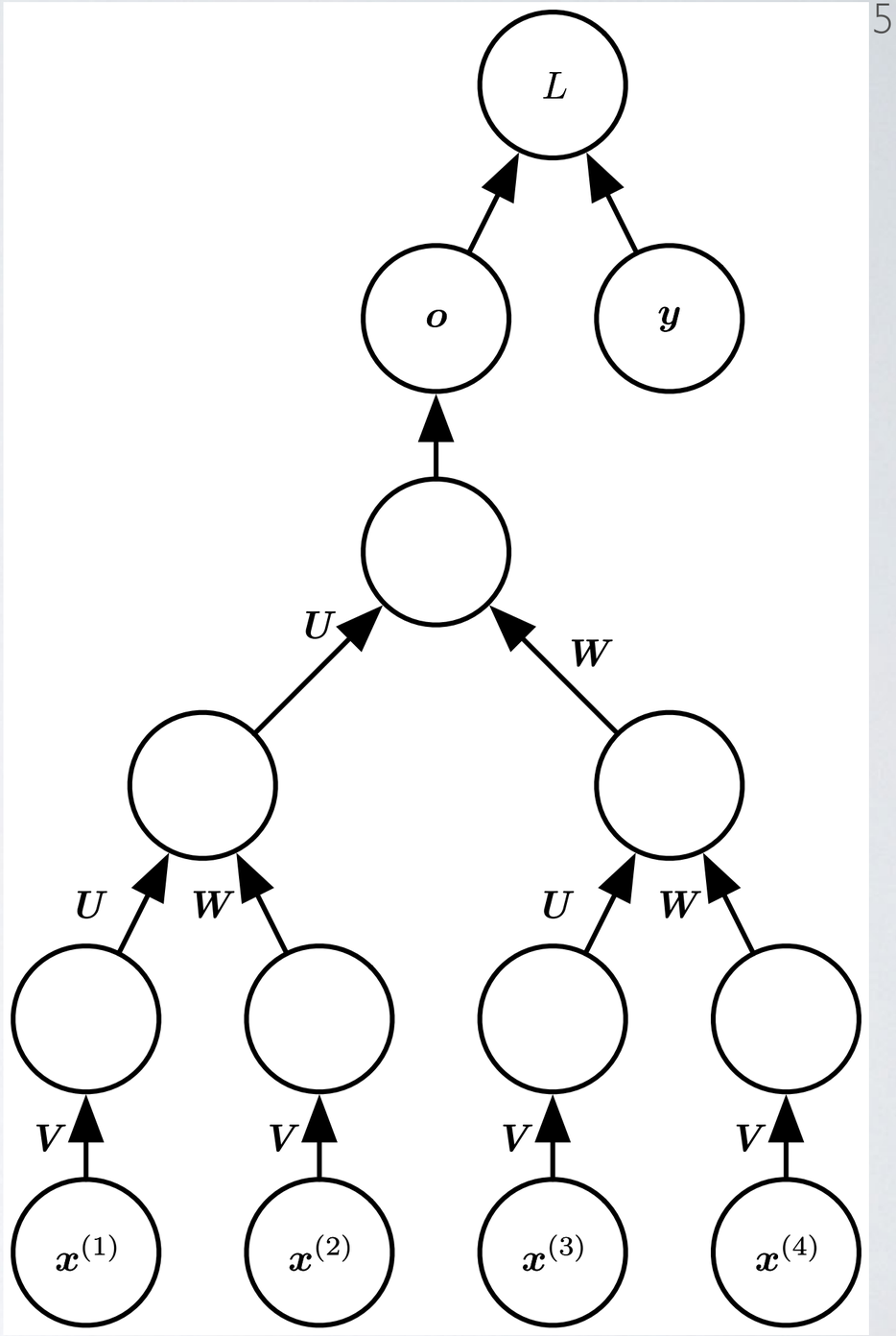
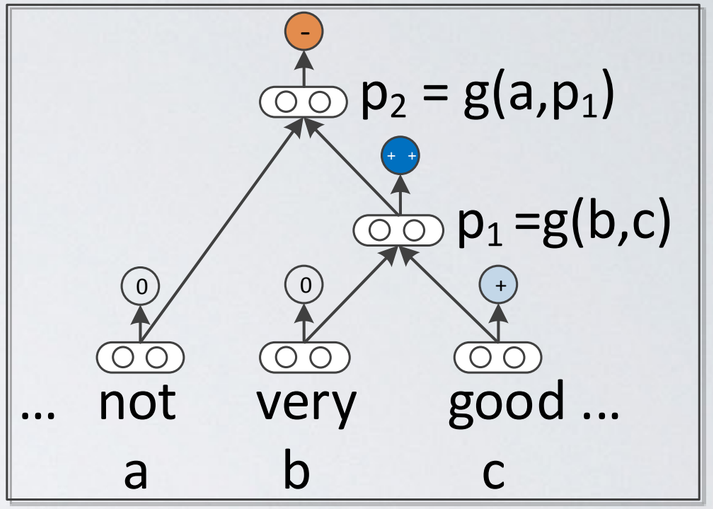


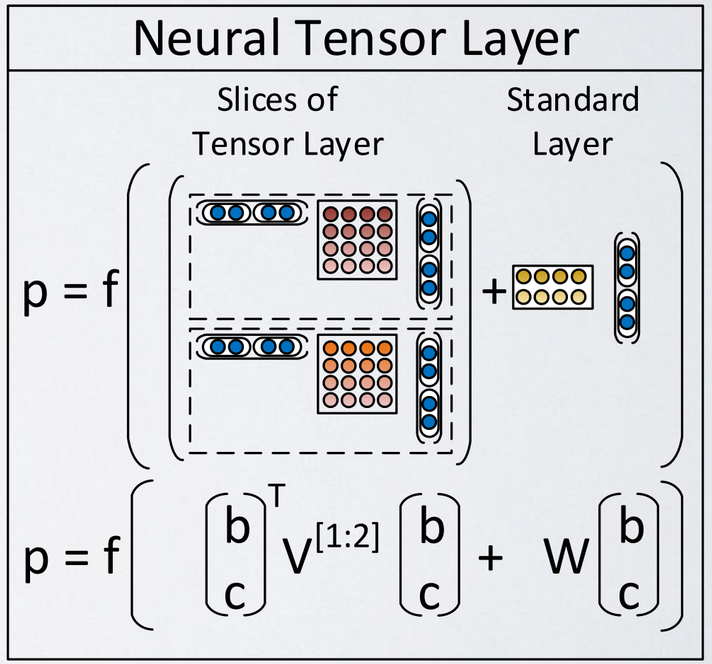
Image from Goodfellow et al. (2016)

RECURSIVE NETWORKS FOR SENTIMENT ANALYSIS

Topics: Recursive NN

- Application of Recursive NN to movie review sentiment analysis (Socher et al. EMNLP 2013).
- Stanford Parser is used to recover the tree structure (Klein and Manning, 2003)
- Trained with supervised (sub)phrase sentiment labels.
- Generalize pair-wise interactions to Recursive Neural Tensor Networks (RNTNs).





Images from Socher et al. (2013)

RECURSIVE NETWORKS FOR SENTIMENT ANALYSIS

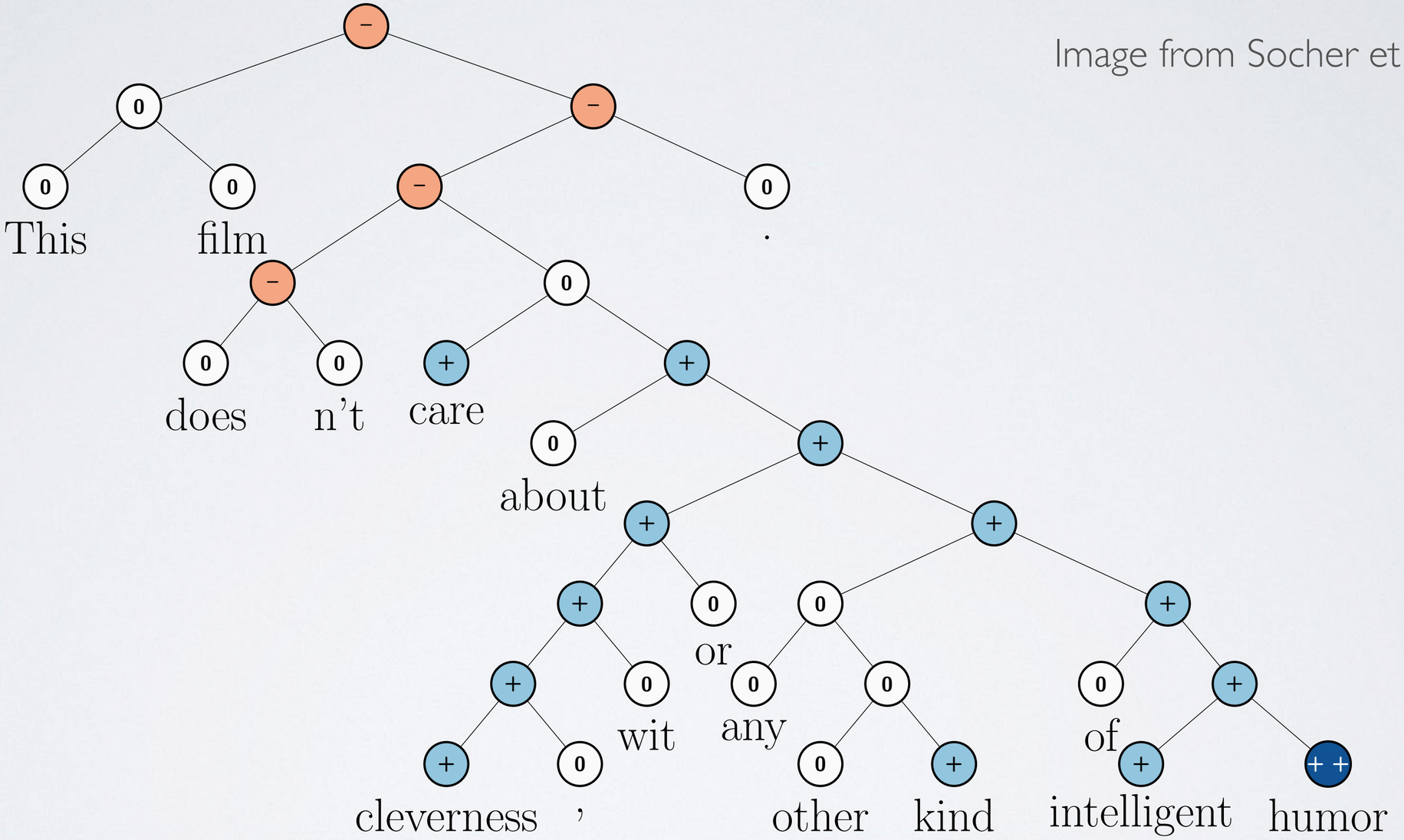


Image from Socher et al. (2013)