Recurrent Neural Networks and Encoder-Decoder Models

Diego Marcheggiani

University of Amsterdam
ILLC

Unsupervised Language Learning 2016

Outline

Word Embeddings Recap

Recurrent Neural Networks

Encoder Decoder Approach

Outline

Word Embeddings Recap

Recurrent Neural Networks

Encoder Decoder Approach

Word Representation

- In traditional NLP words are represented as a one-hot vector

Word Representation

- In traditional NLP words are represented as a one-hot vector
- Syntactically and semantically correlated words appear as completely different symbols

Word Representation

- In traditional NLP words are represented as a one-hot vector
- Syntactically and semantically correlated words appear as completely different symbols
- Add general features to the word, pos, prefixes, etc. to overcome the problem of data sparsity
- machine learning algorithms "understand" that `dog` is similar to `dogs`

Word Representation

- In traditional NLP words are represented as a one-hot vector
- Syntactically and semantically correlated words appear as completely different symbols
- Add general features to the word, pos, prefixes, etc. to overcome the problem of data sparsity
- machine learning algorithms "understand" that `dog` is similar to `dogs`

but what about `cat` and `dog` ? or `good` and `amazing` ?

Distributed Word Representation

"You shall know a word by the company it keeps!" (J. R. Firth, 1957)

Distributed Word Representation

"You shall know a word by the company it keeps!" (J. R. Firth, 1957)

- the meaning of a word is given by the context in which the word appears

Distributed Word Representation

"You shall know a word by the company it keeps!" (J. R. Firth, 1957)

- the meaning of a word is given by the context in which the word appears
- if two words appear in similar contexts (company), they have, to some extent, similar meanings.

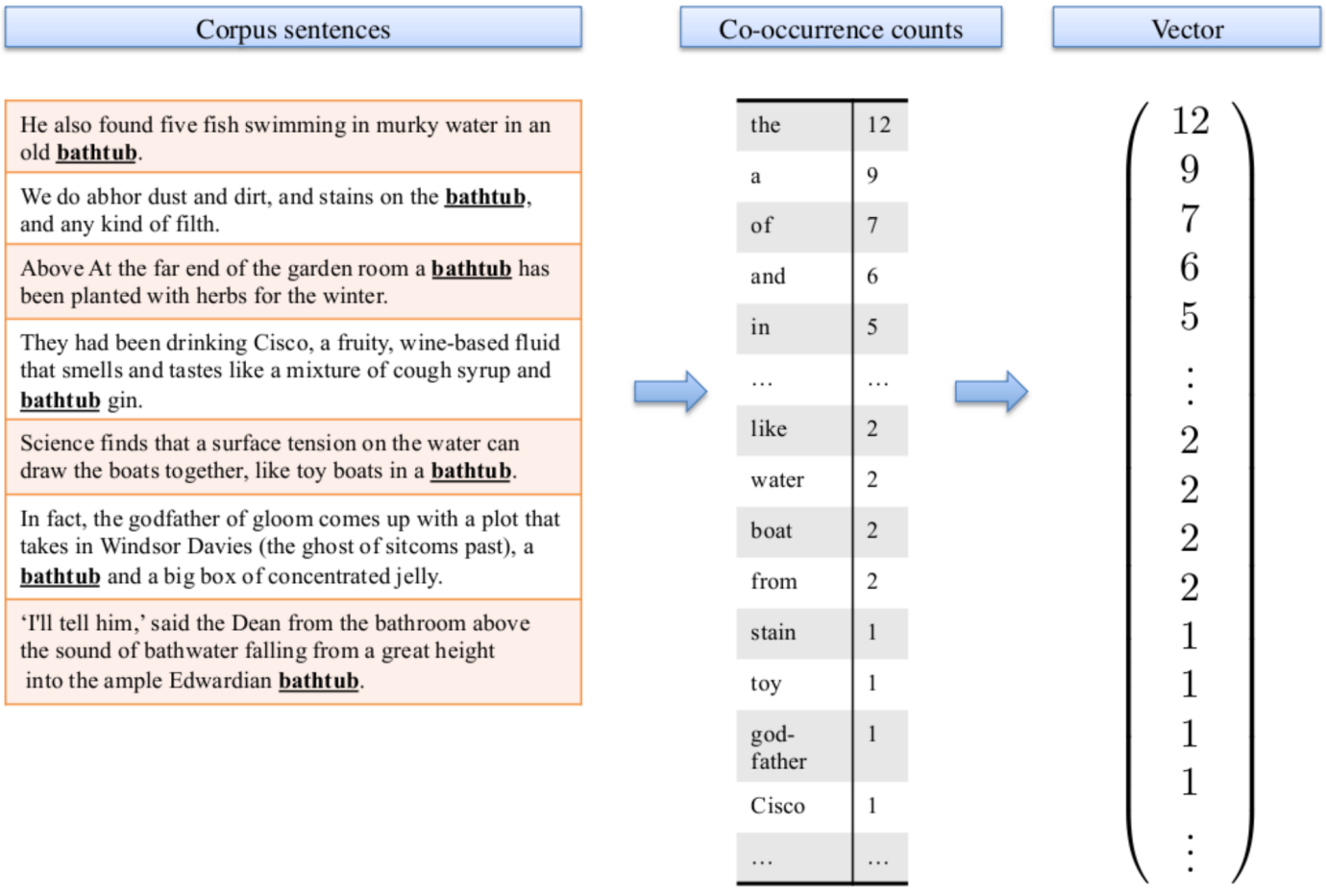
Distributed Word Representation

"You shall know a word by the company it keeps!" (J. R. Firth, 1957)

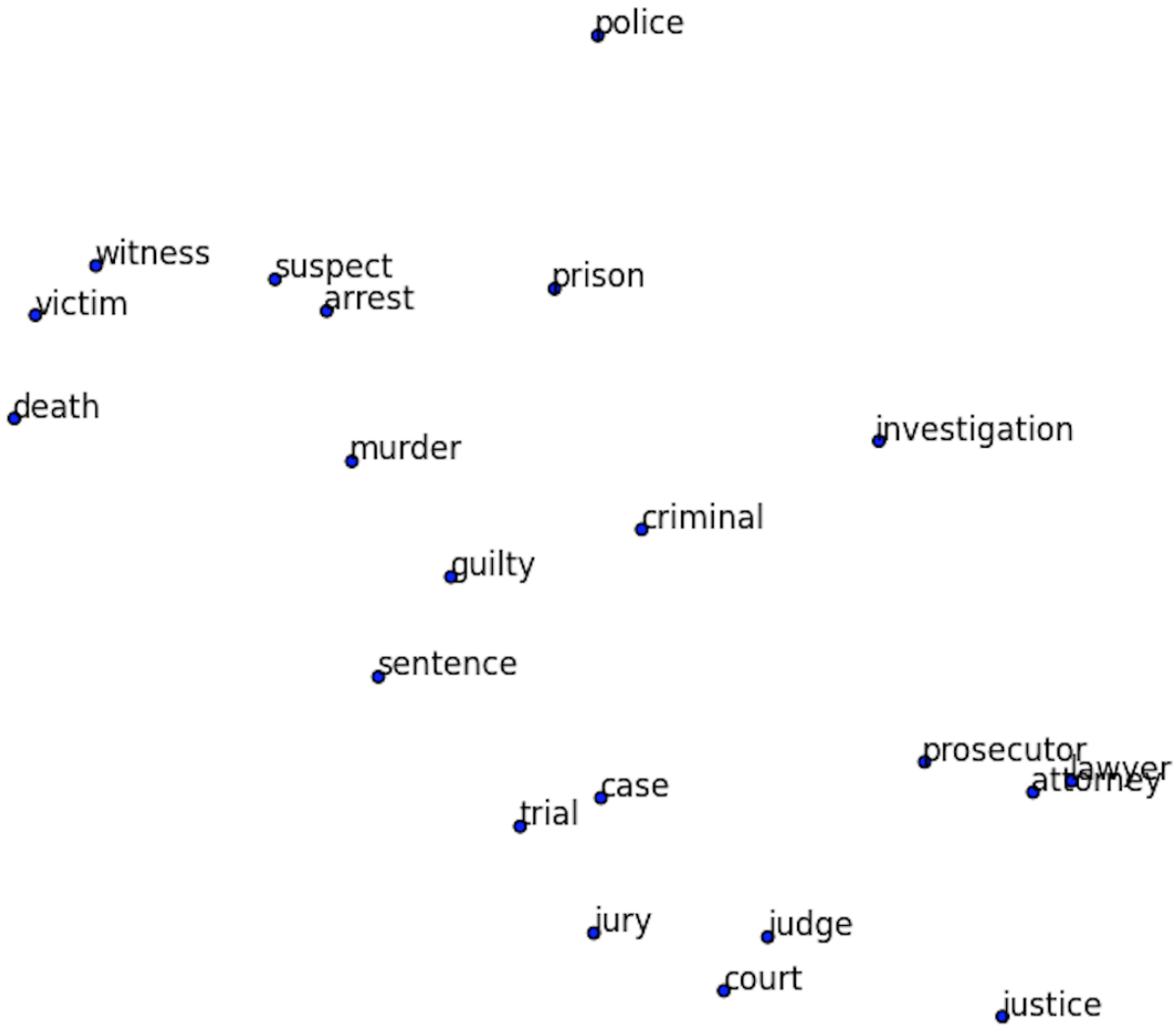
- the meaning of a word is given by the context in which the word appears
- if two words appear in similar contexts (company), they have, to some extent, similar meanings.

How can we formalize this intuition?

Co-occurence matrix



Word similarity



Learning word representations

- Vanilla count-based approaches results in huge sparse matrices

Learning word representations

- Vanilla count-based approaches results in huge sparse matrices
- They can be approximated to dense matrices using SVD.

Learning word representations

- Vanilla count-based approaches results in huge sparse matrices
- They can be approximated to dense matrices using SVD.
- Bengio et al. 2006 introduced a neural language model for learning word representations.

Learning word representations

- Vanilla count-based approaches results in huge sparse matrices
- They can be approximated to dense matrices using SVD.
- Bengio et al. 2006 introduced a neural language model for learning word representations. expensive
- Most popular model nowadays is Google's skip-gram word2vec [Mikolov et al. 2013].
- The idea is to predict context words of a given word.

Skip-gram word2vec

- $W \in \mathbb{R}^{|V| \times d}$ is the word embedding matrix
- $C \in \mathbb{R}^{|V| \times d}$ is the context embedding matrix
- V is the words dictionary
- d is the vector size of the word representation
- the size of the context window is given as hyperparameter

For each context, word pair $(c, w) \in \mathcal{D}$, we want to maximize

$$p(c|w) = \frac{\exp(C(c) \cdot W(w))}{\sum_{c'} \exp(C(c') \cdot W(w))}, \tag{1}$$

Skip-gram word2vec

For each context, word pair $(c, w) \in \mathcal{D}$, we want to maximize

$$p(c|w) = \frac{\exp(C(c) \cdot W(w))}{\sum_{c'} \exp(C(c') \cdot W(w))}, \tag{2}$$

Cons

- $\sum_{c'}$ is intractable, but negative sampling does the job.

Skip-gram word2vec

For each context, word pair $(c, w) \in \mathcal{D}$, we want to maximize

$$p(c|w) = \frac{\exp(C(c) \cdot W(w))}{\sum_{c'} \exp(C(c') \cdot W(w))}, \tag{2}$$

Cons

- $\sum_{c'}$ is intractable, but negative sampling does the job.

Pros

- Log-linear, easy to train!

Word representations

Ok, well done! Now what?

Word representations

Ok, well done! Now what?

- word embeddings as input to neural networks
- since deep learning models seek to solve non-convex optimization problems, starting from a good point in the parameters space usually helps.

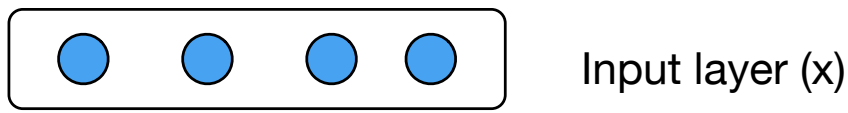
Outline

Word Embeddings Recap

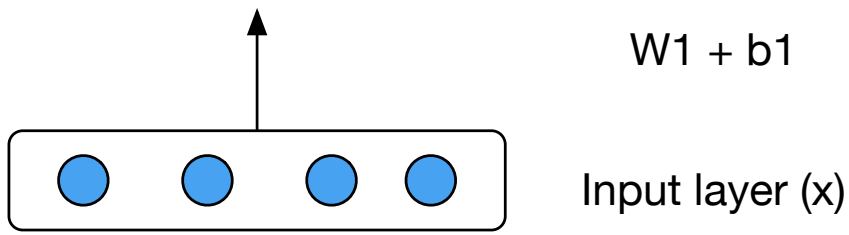
Recurrent Neural Networks

Encoder Decoder Approach

Neural networks recap

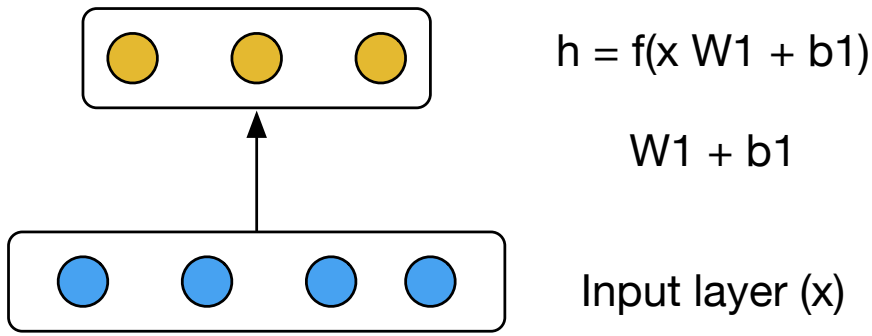


Neural networks recap



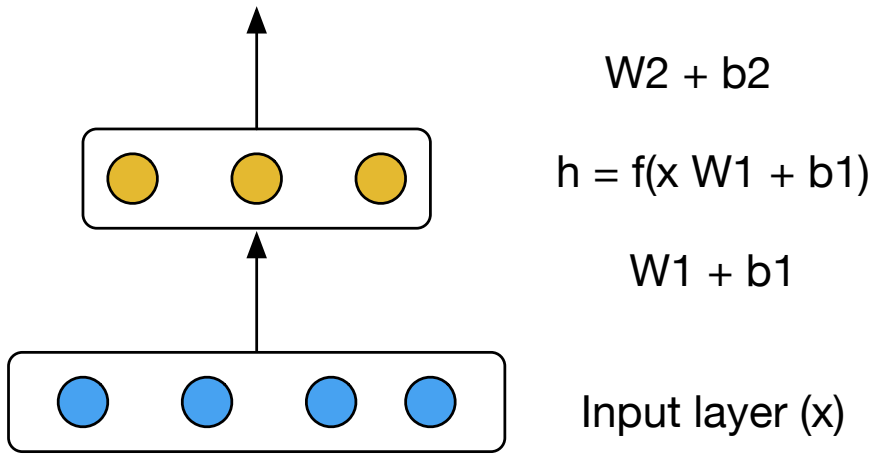
- W_1 is a parameter matrix and b_1 is the bias

Neural networks recap



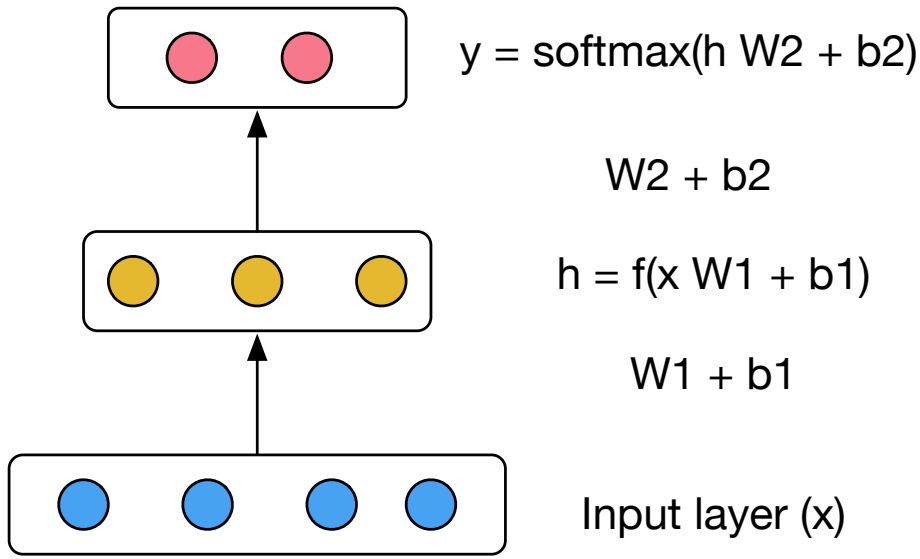
- W_1 is a parameter matrix and b_1 is the bias
- f is a non-linear activation function
- h is a hidden layer of the network

Neural networks recap



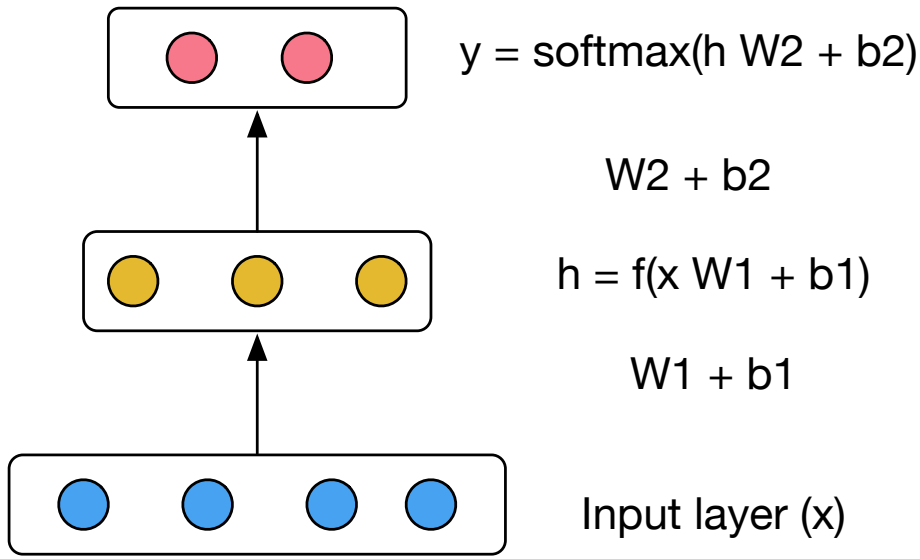
- W_1 is a parameter matrix and b_1 is the bias
- f is a non-linear activation function
- h is a hidden layer of the network
- W_2, b_2 more parameters

Multi-layer perceptron



- W_1 is a parameter matrix and b_1 is the bias
- f is a non-linear activation function
- h is a hidden layer of the network
- W_2, b_2 more parameters

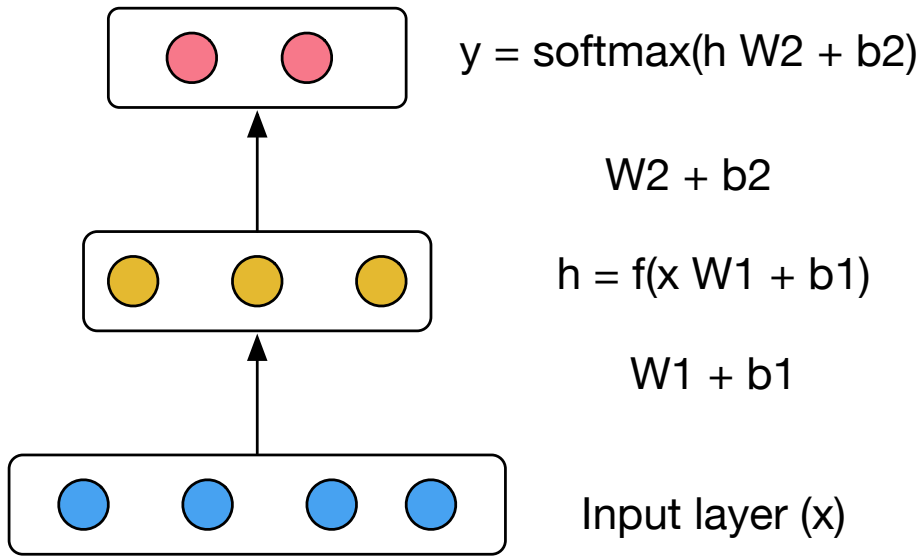
Multi-layer perceptron



- W_1 is a parameter matrix and b_1 is the bias
- f is a non-linear activation function
- h is a hidden layer of the network
- W_2, b_2 more parameters

We can add as many hidden layers as we like transforming it in a deep neural network.

Neural networks recap

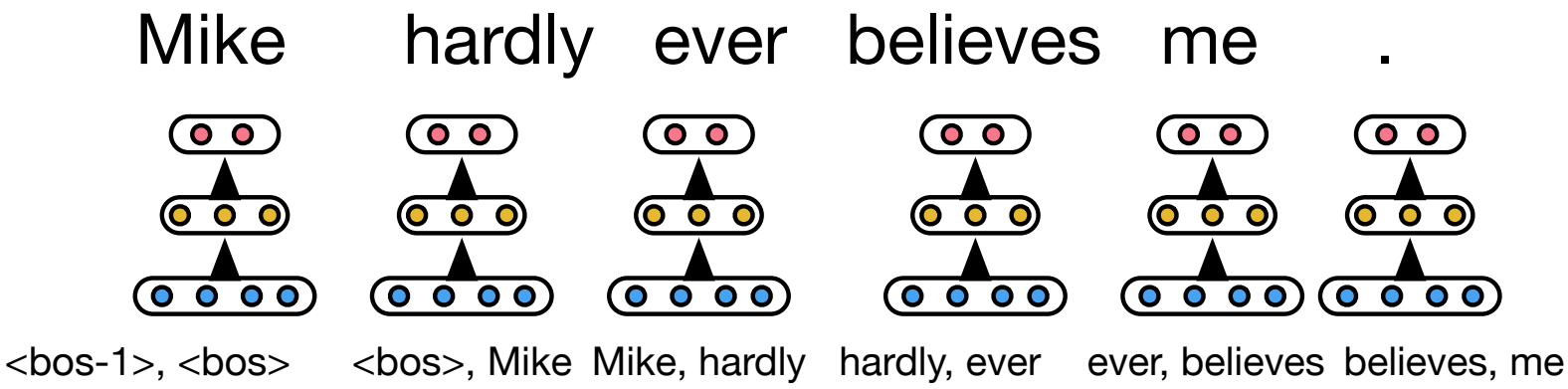


Standard feedforward neural networks cannot deal with variable length input.

Language modeling

Compute the probability of a sentence, or
given a sequence of words predicting the word that comes next.

- trigram language model $p(x_t|x_{t-1}, x_{t-2})$

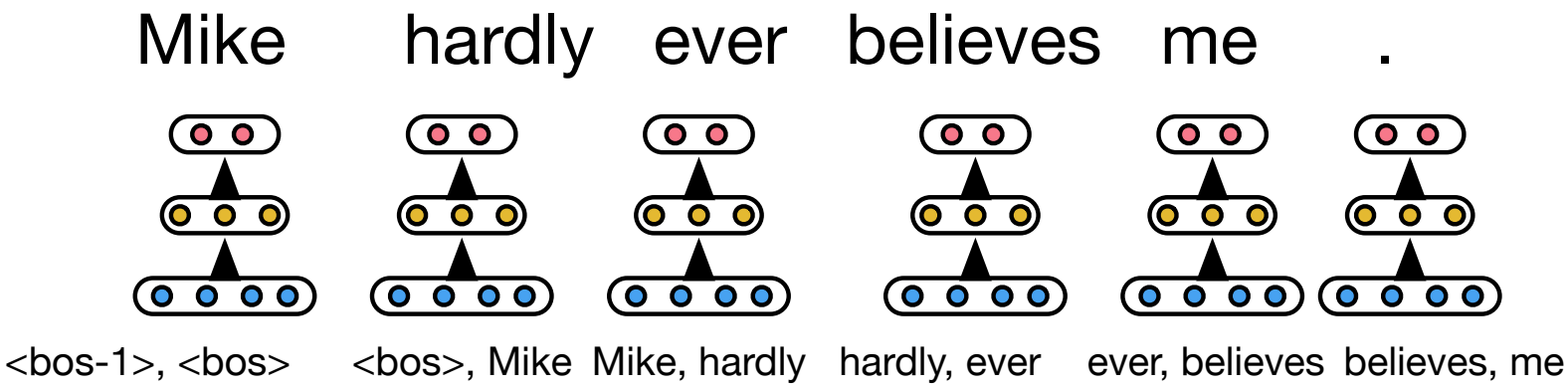


- cannot "see" the subject so cannot predict the right verb
- increase the size of the model.

Language modeling

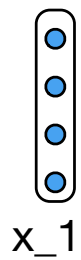
Compute the probability of a sentence, or
given a sequence of words predicting the word that comes next.

- trigram language model $p(x_t|x_{t-1}, x_{t-2})$

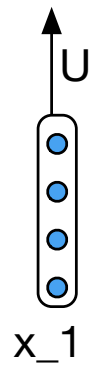


- cannot "see" the subject so cannot predict the right verb
- increase the size of the model.
- what about longer dependencies ?

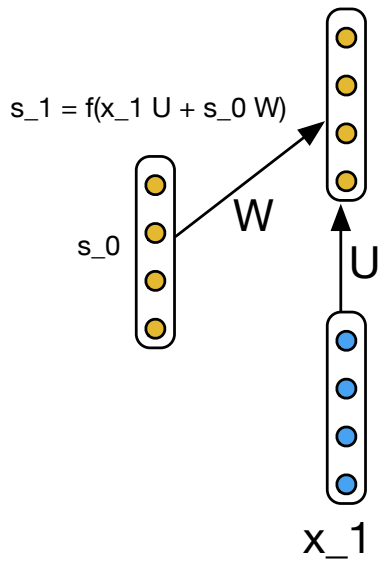
Recurrent Neural Networks



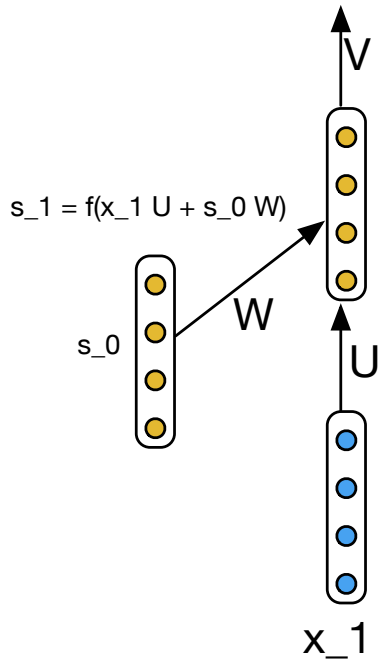
Recurrent Neural Networks



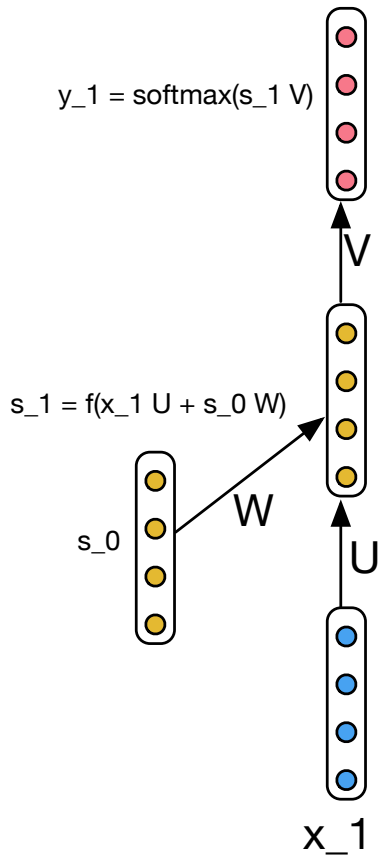
Recurrent Neural Networks



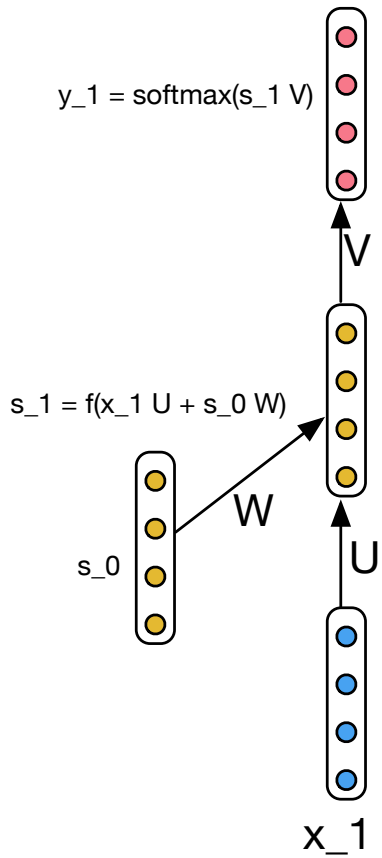
Recurrent Neural Networks



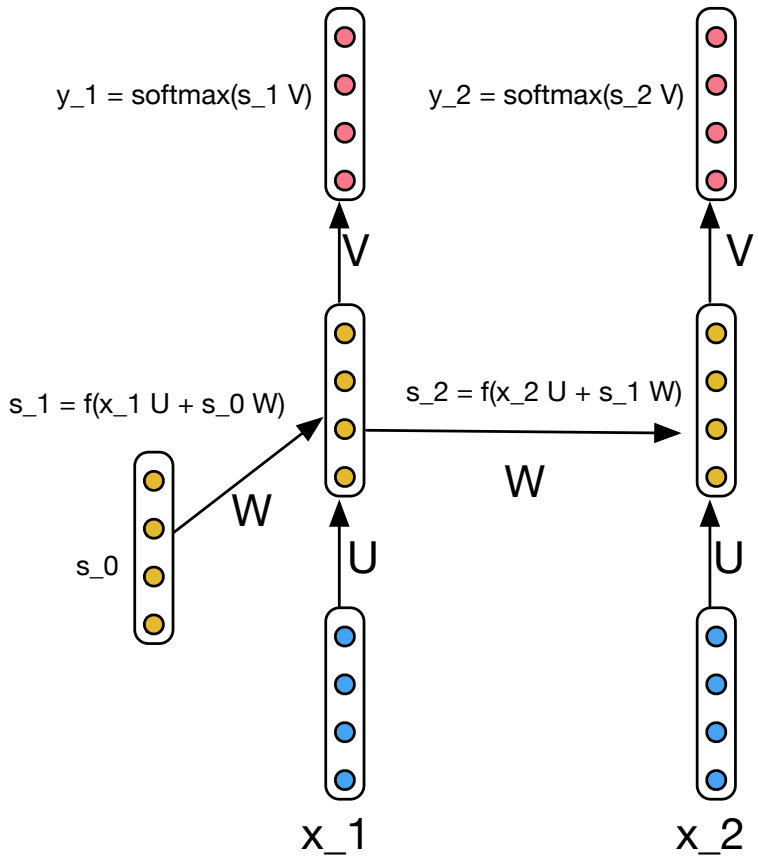
Recurrent Neural Networks



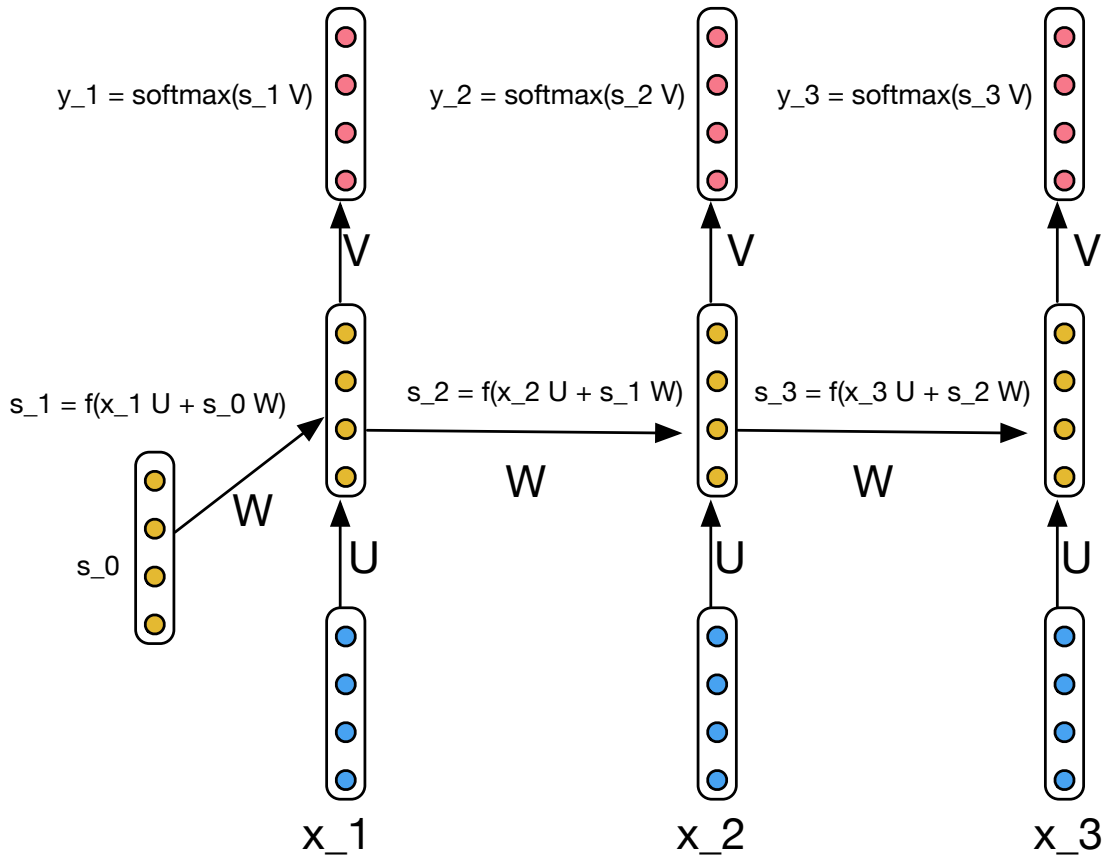
RNN step 1



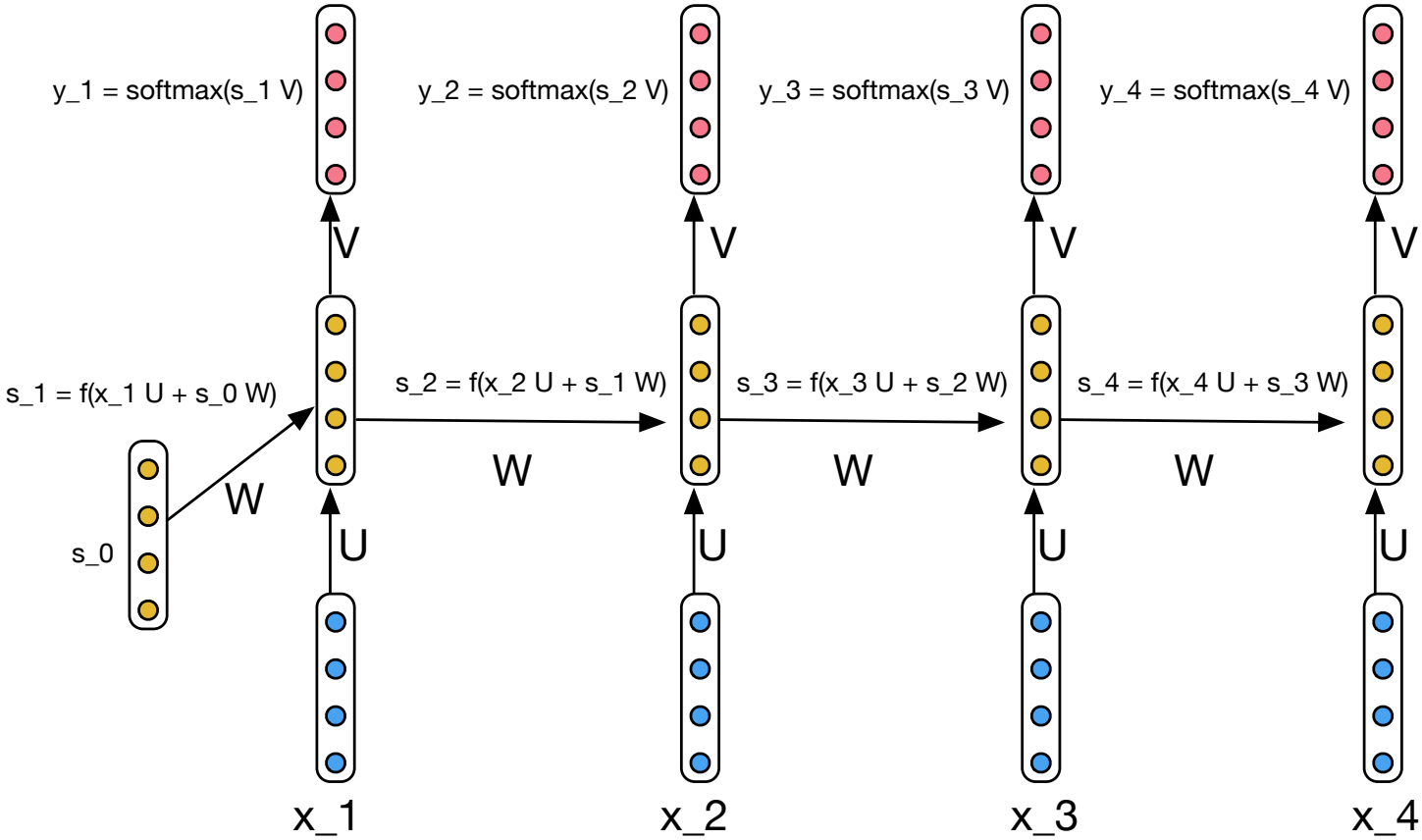
RNN step 2


y_1 = softmax(s_1 V)
y_2 = softmax(s_2 V)
V
V
s_1 = f(x_1 U + s_0 W)
s_2 = f(x_2 U + s_1 W)
W
W
s_0
U
U
x_1
x_2

RNN step 3

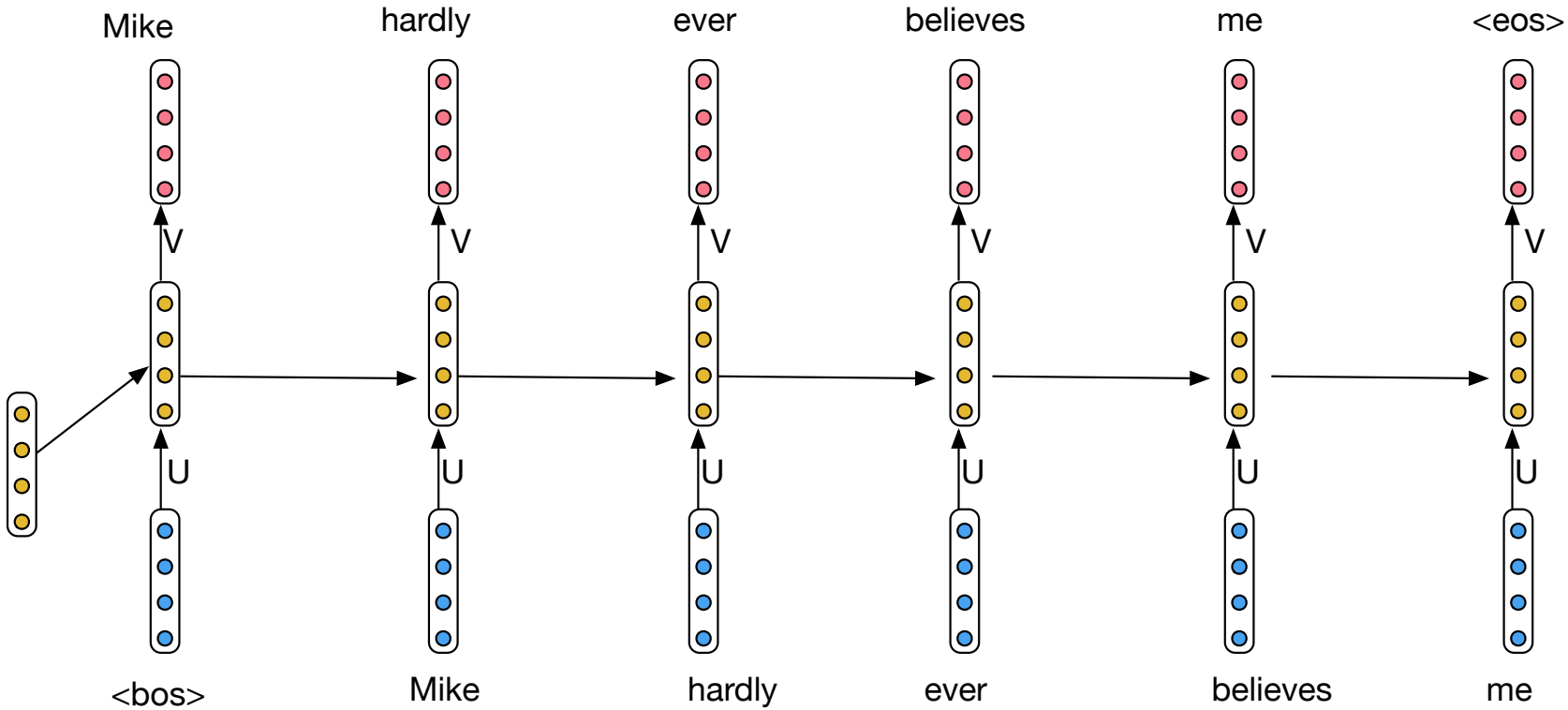

y_1 = softmax(s_1 V)
y_2 = softmax(s_2 V)
y_3 = softmax(s_3 V)
V
V
V
s_1 = f(x_1 U + s_0 W)
s_2 = f(x_2 U + s_1 W)
s_3 = f(x_3 U + s_2 W)
W
W
W
s_0
U
U
U
x_1
x_2
x_3

RNN step 4



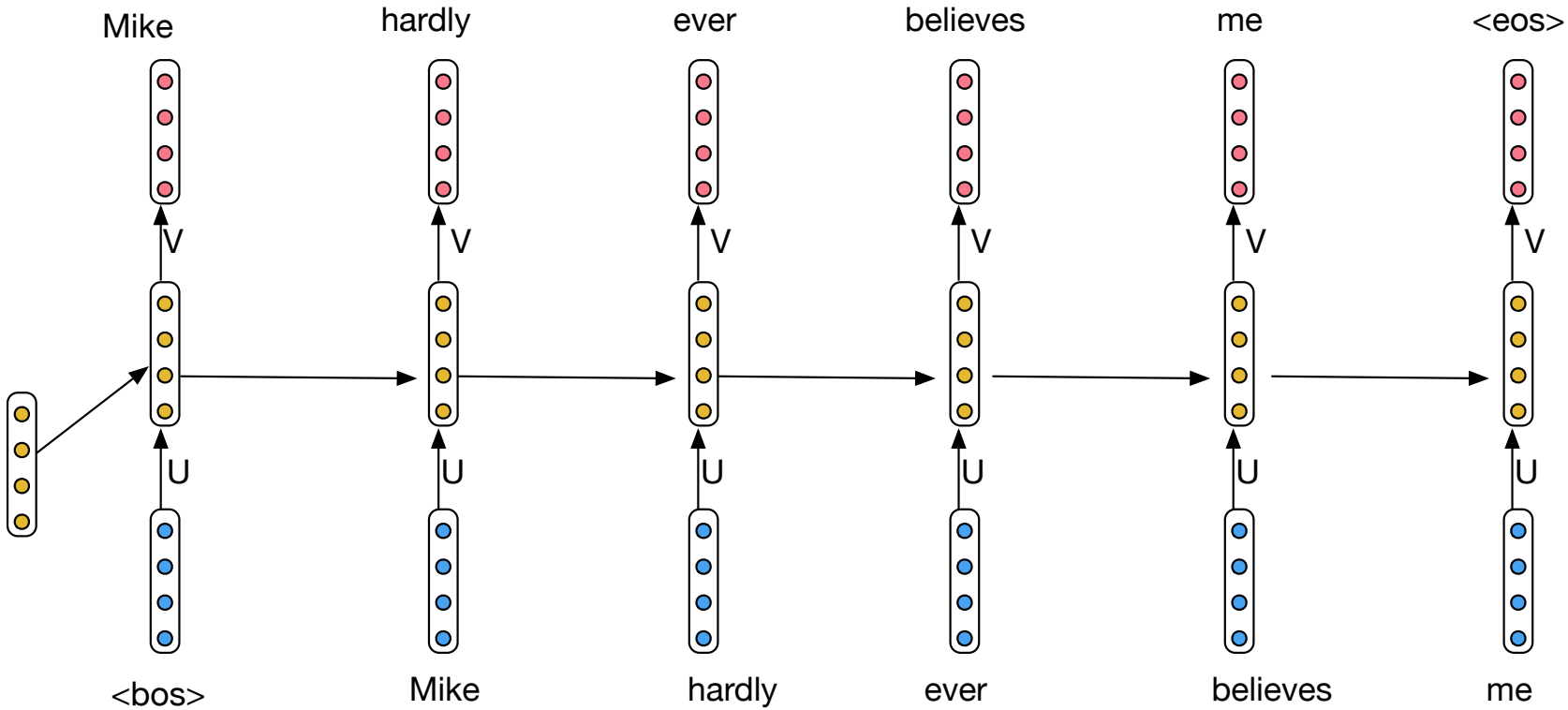
In each step we use information coming from all the previous steps.

Language modeling (reprise)



Long dependencies are captured

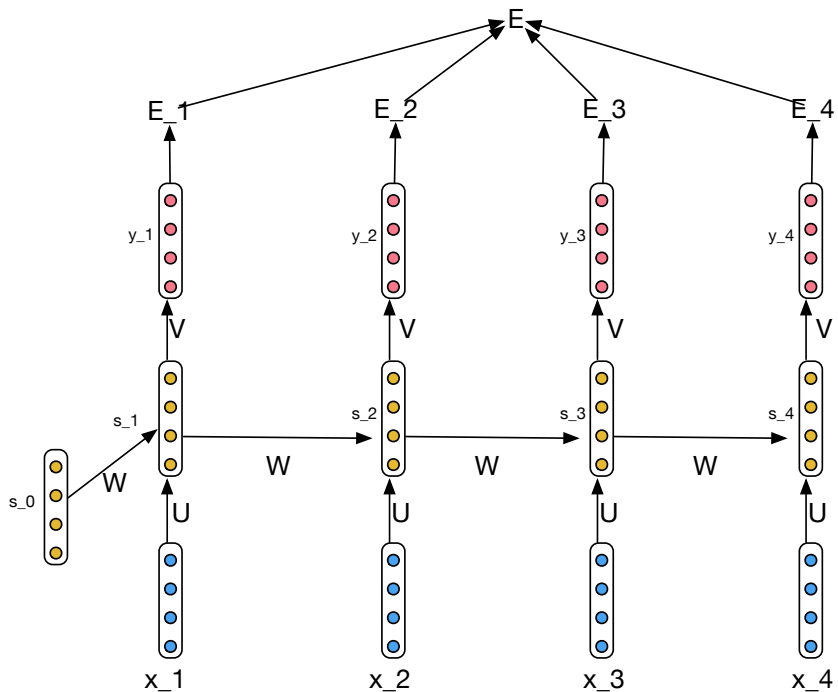
Language modeling (reprise)



Long dependencies are captured (at least in theory)

Training RNNs

- Total error is the sum of errors at each time step $\sum_t E(y_t^*, y_t)$
- Error is calculated as cross entropy loss $E(y_t^*, y_t) = -y_t^* \log y_t$
- Calculating the gradients for V depends only on the current time step t.
- For the recurrent parameters W is a bit different.



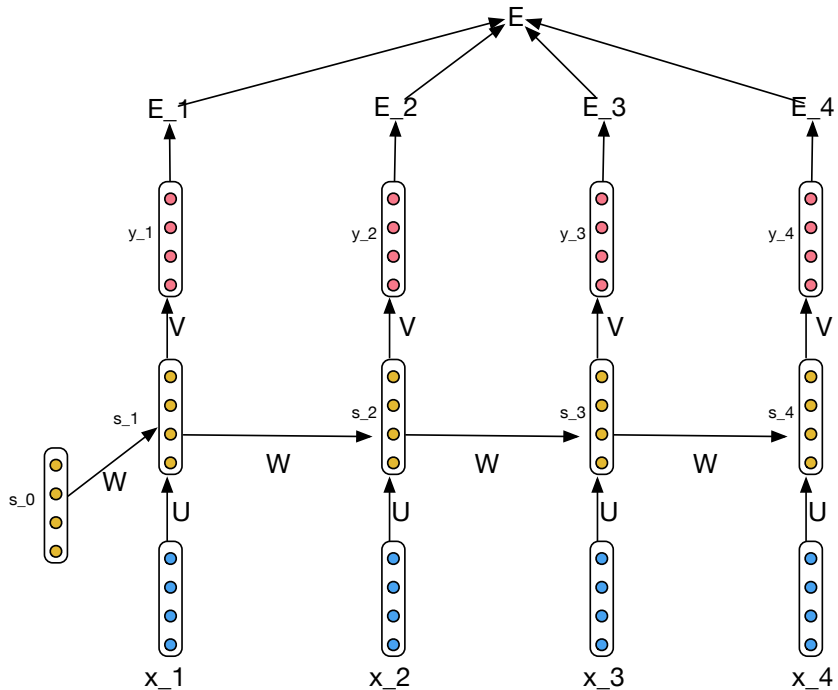
Backpropagation through time

Since we sum errors, we also sum gradients at each time step.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

Following standard backprop derivation, for each time step t we end-up having:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t-1} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}$$



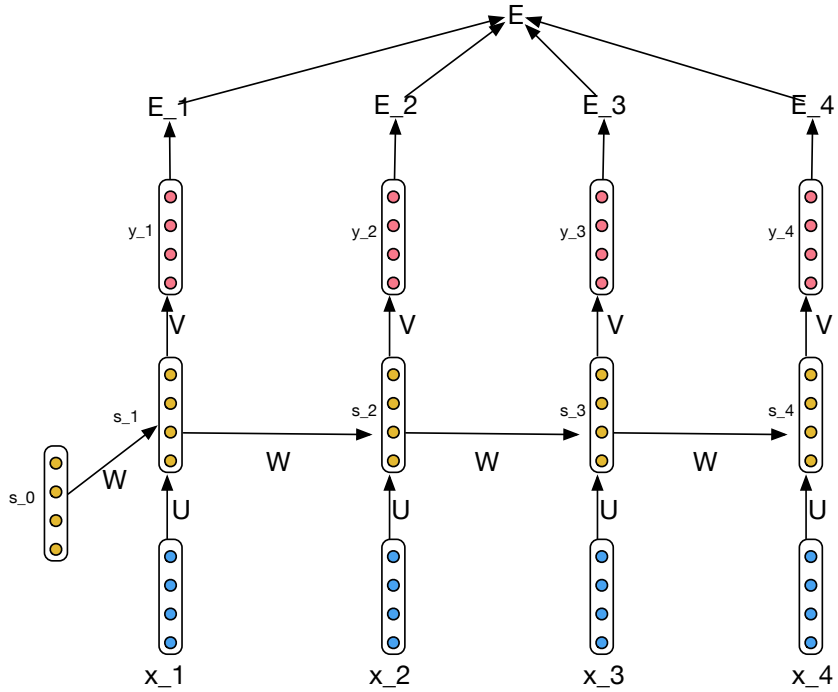
Backpropagation through time

For t = 3 we have:

$$\frac{\partial E_3}{\partial W} = \sum_{k=1}^{2} \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

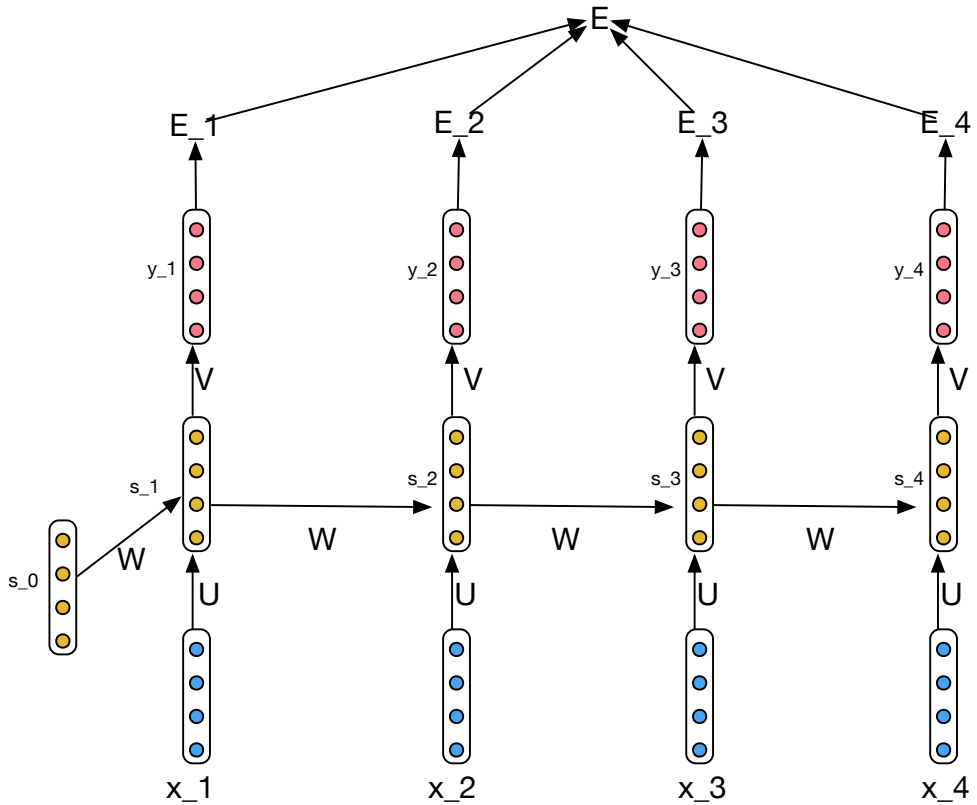
that is unfolded as:

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W}$$



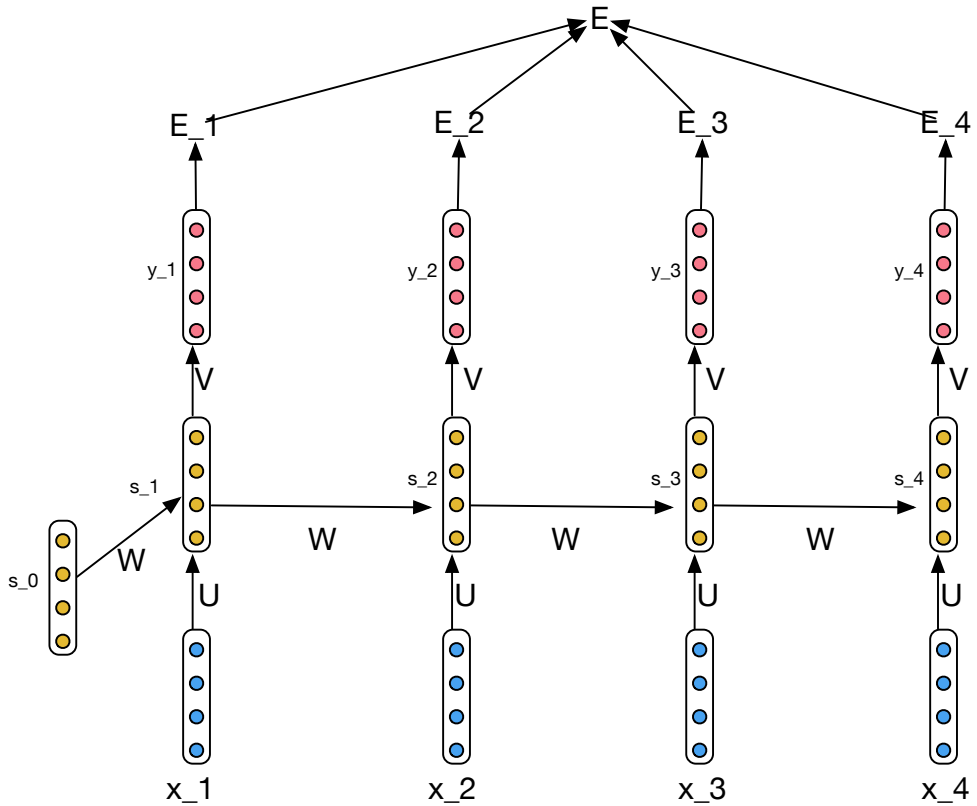
Backpropagation through time

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}$$



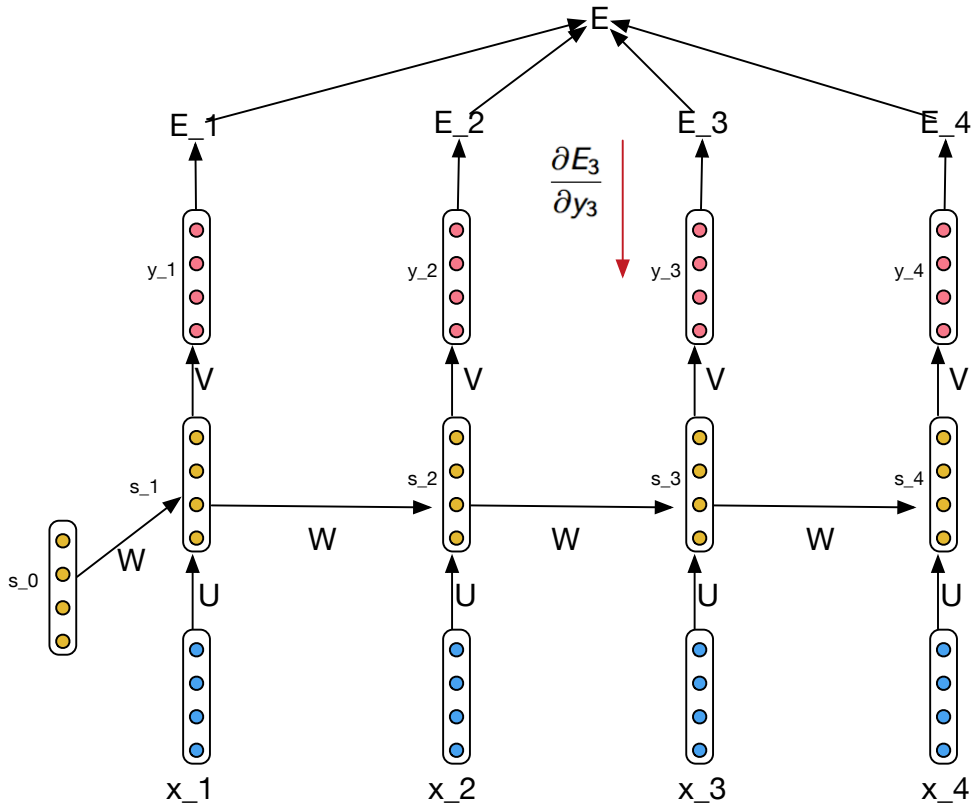
Backpropagation through time

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}$$



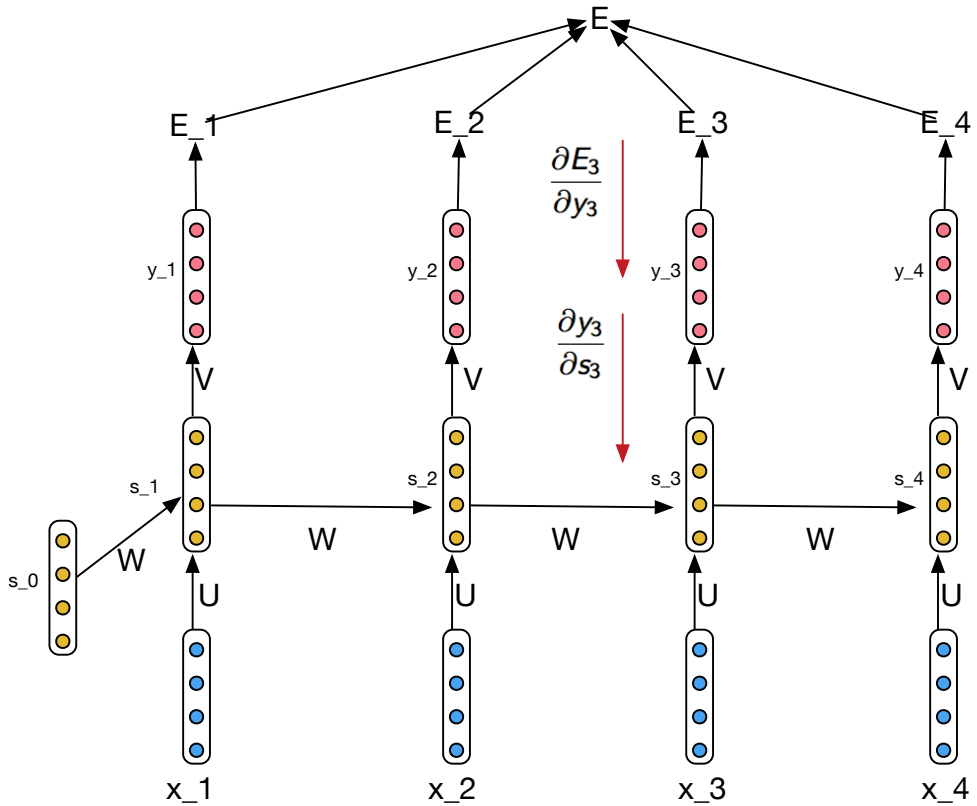
Backpropagation through time

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}$$



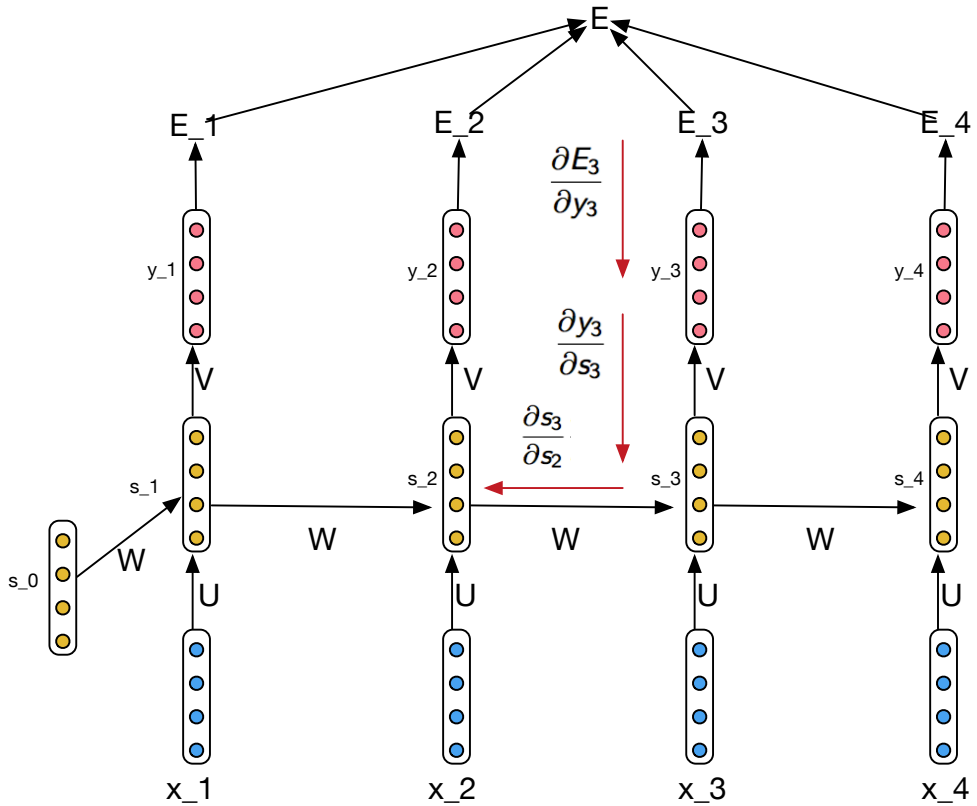
Backpropagation through time

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}$$



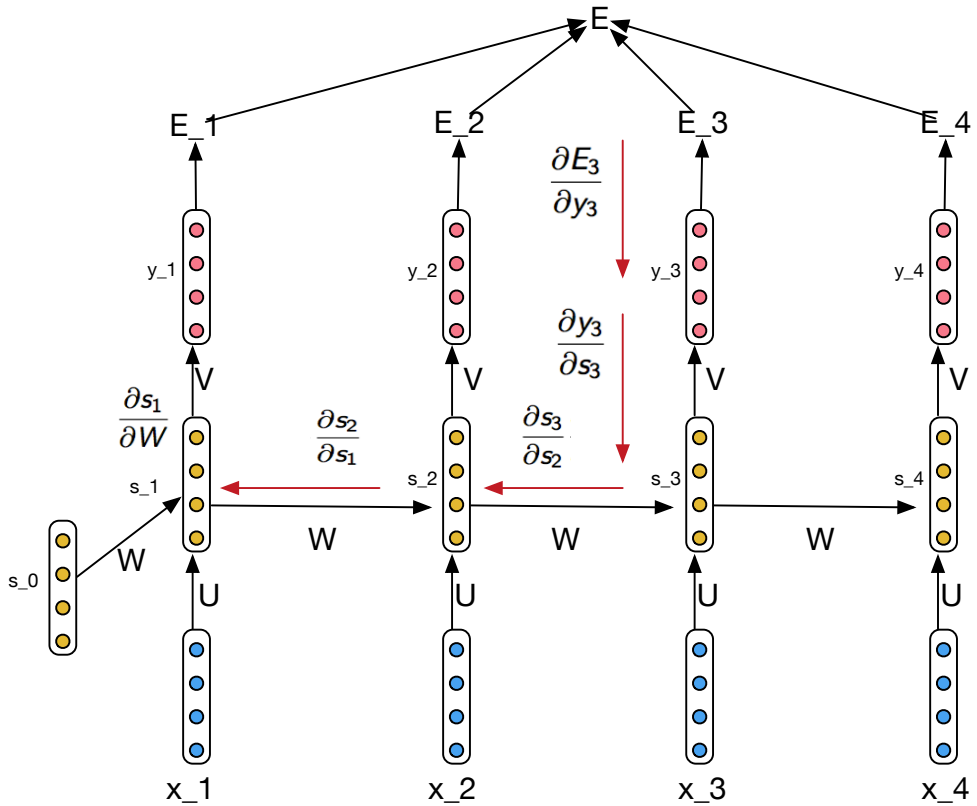
Backpropagation through time

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}$$



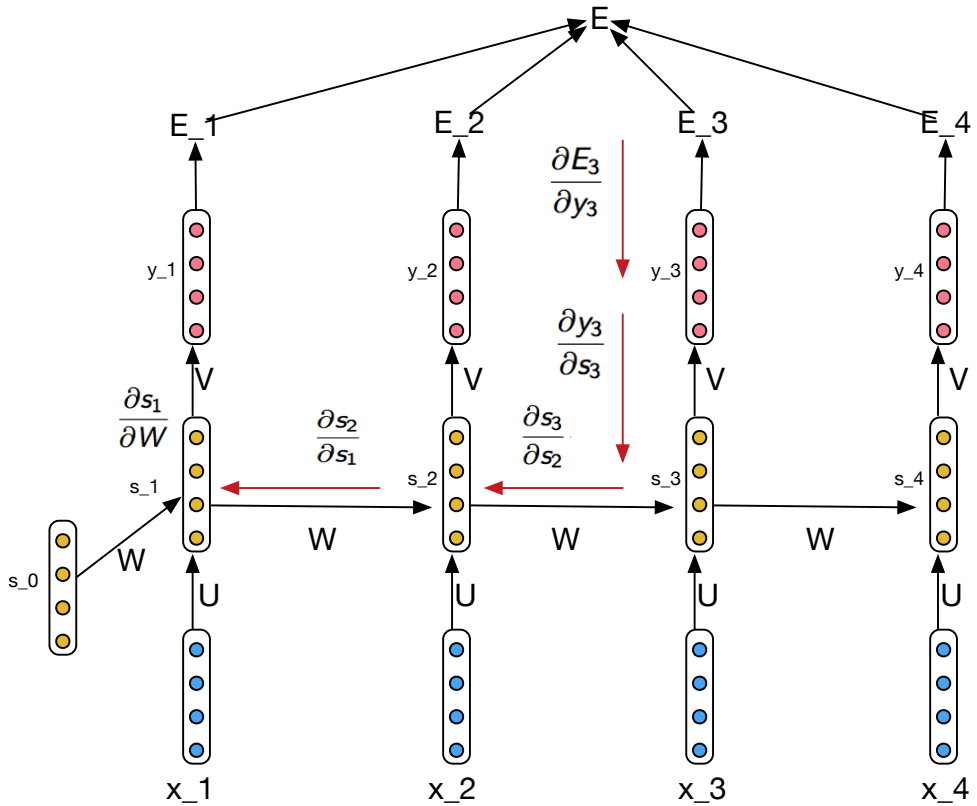
Backpropagation through time

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}$$



Backpropagation through time

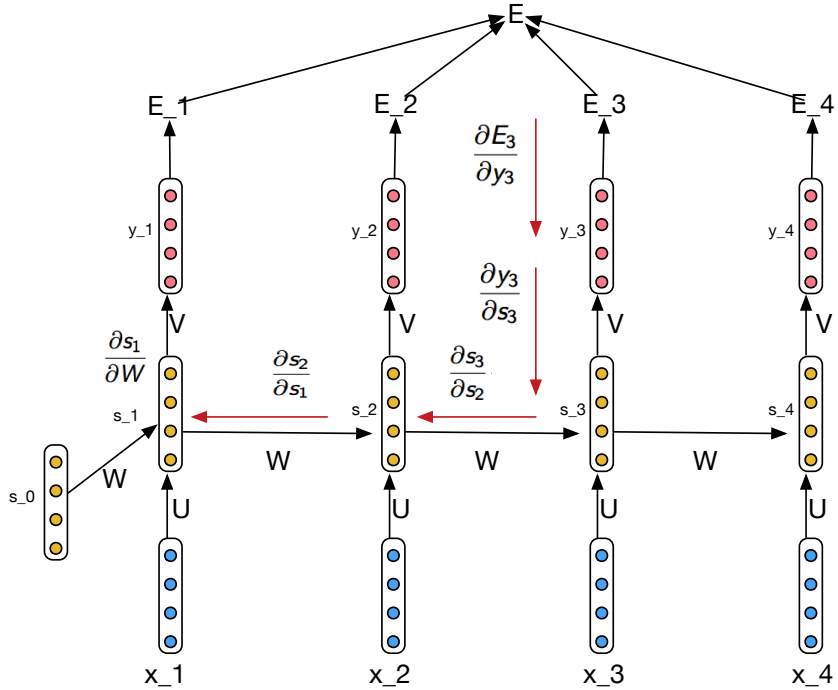
$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}$$



RNN learning issues

Vanishing (exploding) gradient for long dependencies.

- Exploding gradient solution:
  - Gradient clipping – clipping the norm of the gradient if exceeds a certain threshold.
- Vanishing gradient solution:
  - Long short-term memory networks (LSTM) [Hochreiter and Schmidhuber 1997]
  - Gated recurrent unit networks (GRU) [Cho et al. 2014]



Outline

Word Embeddings Recap

Recurrent Neural Networks

Encoder Decoder Approach

Motivations

- What if we want an output of variable length with respect to the input?
- Can we use RNN for tasks where we have a sequence as input and we want another sequence as output?

Motivations

- What if we want an output of variable length with respect to the input?
- Can we use RNN for tasks where we have a sequence as input and we want another sequence as output?
- machine translation: `Mary eats apples. ->`

Motivations

- What if we want an output of variable length with respect to the input?
- Can we use RNN for tasks where we have a sequence as input and we want another sequence as output?
- machine translation: `Mary eats apples. ->` `Marie mange des pommes.`
- question answering: `Tim is playing in his room.||Where is Tim? ->`

Motivations

- What if we want an output of variable length with respect to the input?
- Can we use RNN for tasks where we have a sequence as input and we want another sequence as output?
- machine translation: `Mary eats apples.` -> `Marie mange des pommes.`
- question answering: `Tim is playing in his room.||Where is Tim?` -> `Tim is in his room.`

Motivations

- What if we want an output of variable length with respect to the input?
- Can we use RNN for tasks where we have a sequence as input and we want another sequence as output?
- machine translation: `Mary eats apples.` -> `Marie mange des pommes.`
- question answering: `Tim is playing in his room.||Where is Tim?` -> `Tim is in his room.`

Encoder decoder a.k.a. sequence to(2) sequence learning.
[Sutskever et al. 2014]

Sequence to sequence learning - Encoding

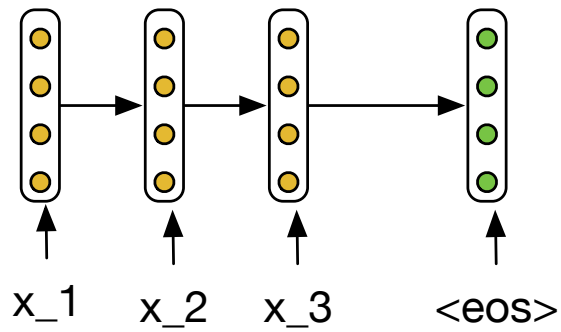
$$p(z_1, ..., z_{T'} | x_1, ..., x_T)$$

Sequence to sequence learning - Encoding

$$p(z_1, ..., z_{T'} | x_1, ..., x_T)$$

- Encode the entire input sequence in a single vector

$$s_t = f(U \cdot x_t + W \cdot s_{t-1})$$

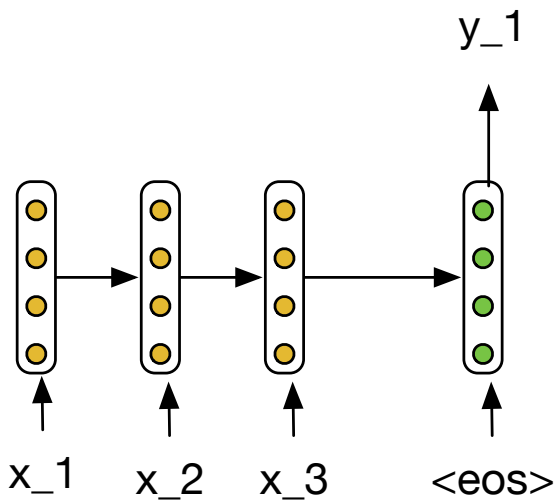


Sequence to sequence learning - Decoding

$$p(z_1, ..., z_{T'} | x_1, ..., x_T)$$

- proceed like in a standard RNN starting from the encoding vector

$$y_1 = softmax(V \cdot s_{T_e})$$



Sequence to sequence learning - Decoding

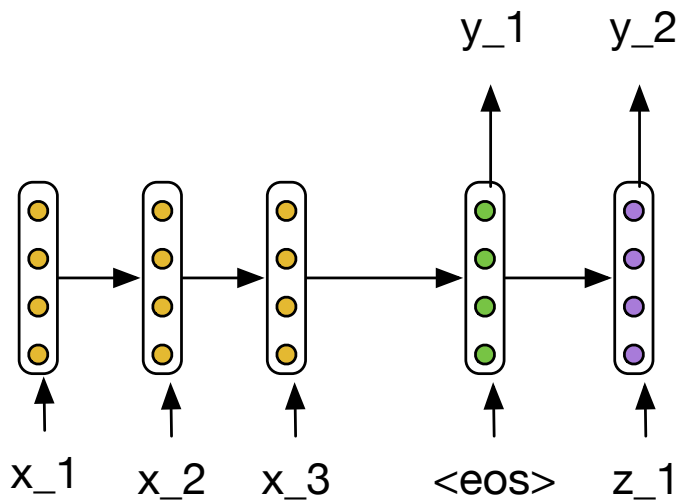
$$p(z_1, ..., z_{T'} | x_1, ..., x_T)$$

- proceed like in a standard RNN starting from the encoding vector

$$y_1 = softmax(V \cdot s_{T_e})$$

- and then continue decoding the second word in the sequence with

$$s_1^d = f(U \cdot z_1 + W \cdot s_{T_e}), \quad y_2 = softmax(V \cdot s_1^s)$$



Sequence to sequence learning - Decoding

$$p(z_1, ..., z_{T'} | x_1, ..., x_T)$$

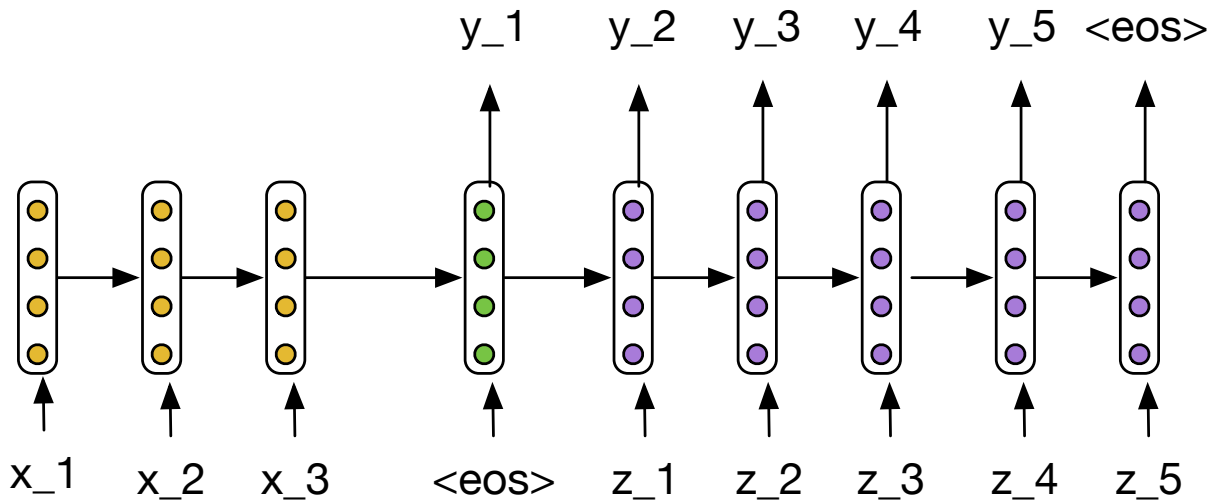
- Begin to decode the output sequence conditioning on this vector

$$y_1 = softmax(V \cdot s_{T_e})$$

- continue decoding the second word in the sequence with

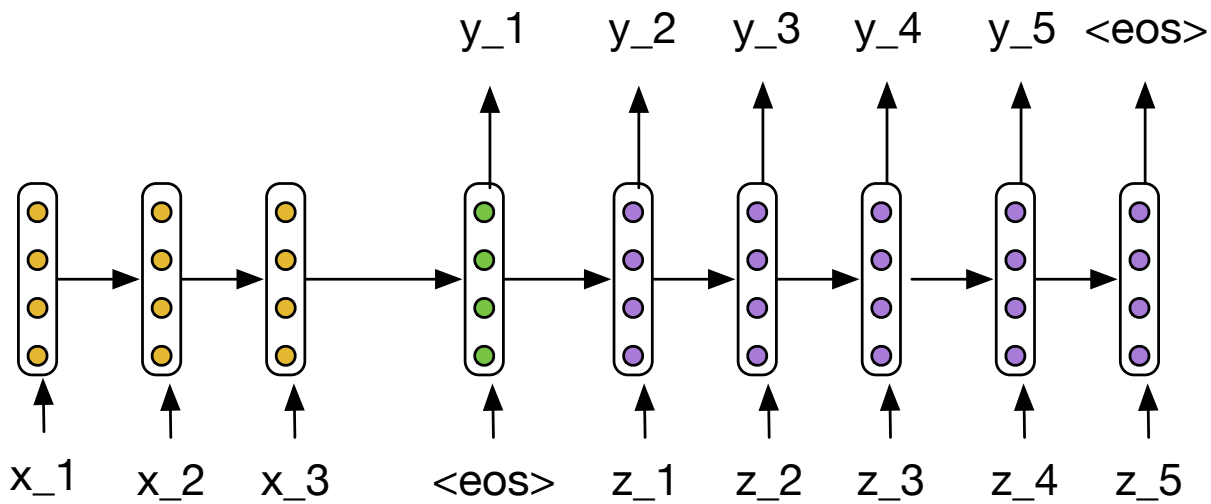
$$s_1^d = f(U \cdot z_1 + W \cdot s_{T_e}), \quad y_2 = softmax(V \cdot s_1^d)$$

- keep decoding until an <eos> token is predicted.



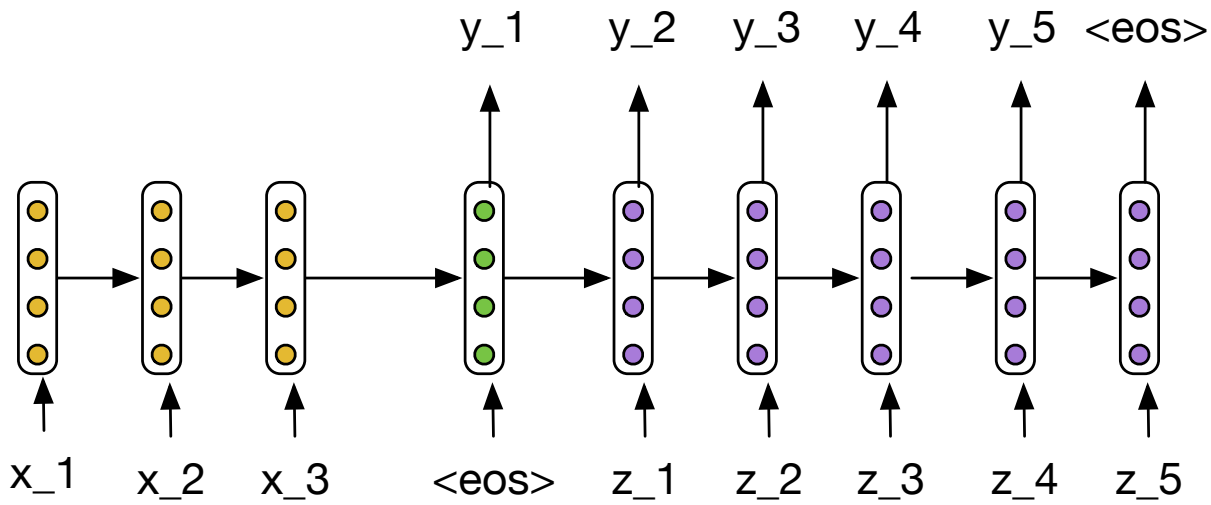
Encoder decoder

$$p(z_1, \ldots, z_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(z_t | s_{T_e}, z_1, \ldots, z_{t-1})$$



Conclusions

- Vanishing gradient is even more problematic in this setting.
- LSTMs or GRUs are crucial to capture long term dependencies.
- Very appealing in task like machine translation.



Lecture recap

- Word embeddings recap
- Neural networks recap
- Recurrent neural networks
- Sequence to sequence learning with neural networks

That's it!

Thanks for your attention!
Questions?