Department of
Linguistics and Translation
香港城市大學
City University of Hong Kong

Computational Linguistics
LT3233

Jixing Li

Lecture 12: Recurrent Neural Network

Slides adapted from Chris Manning

Lecture plan

- Recap: Language model with FFNN
- RNN
- Short break (15 mins)
- Hands-on exercises

Language model

Language Model (LM): A system that predicts the next word

Goal: Compute the probability of a sentence or sequence of words

P(*a pile of shaving cream***)** **or** **P(***cream|a pile of shaving***)**

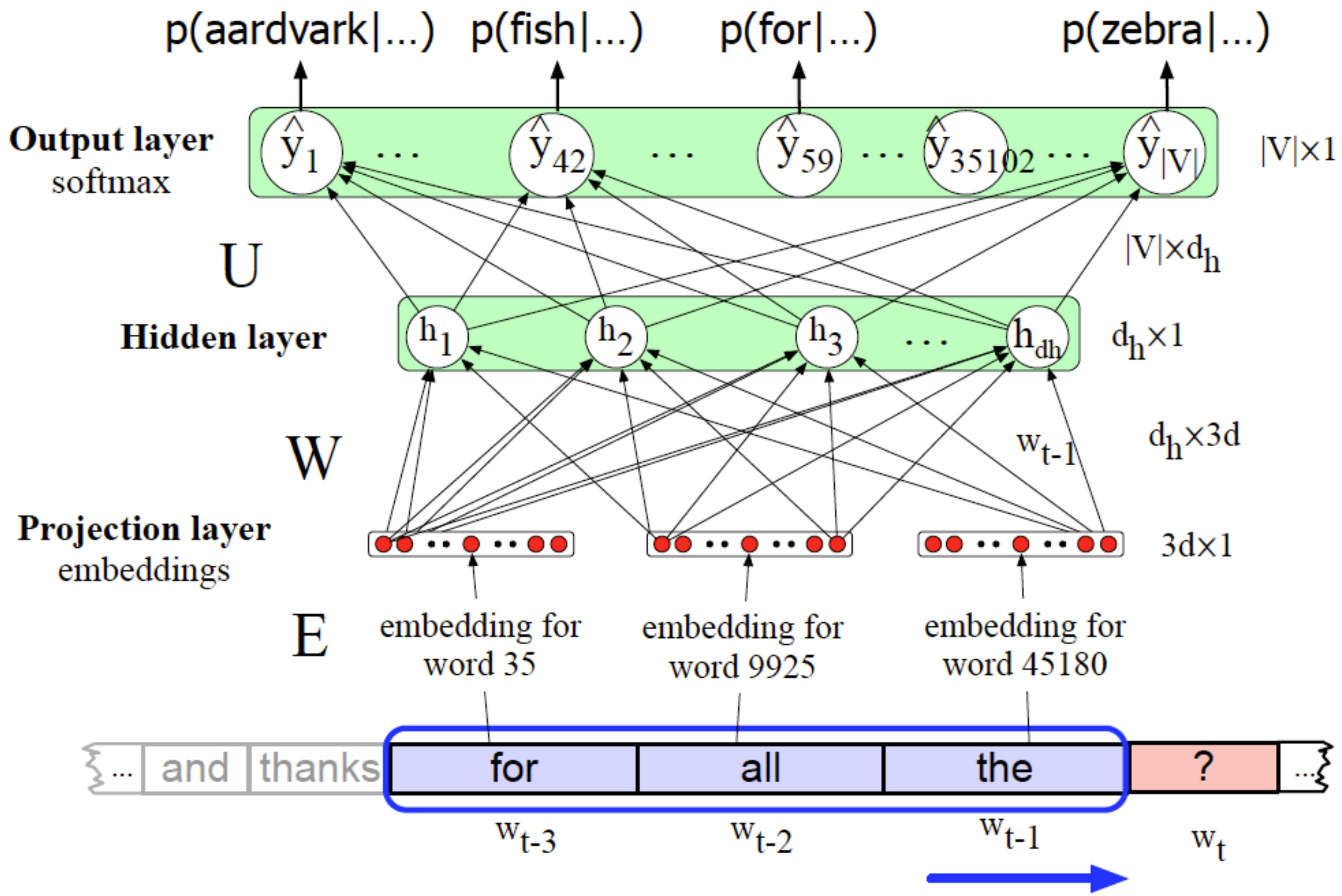
$P(W) = P(w_1, w_2, w_3, \ldots w_n)$ $P(w_n | w_1, w_2, w_3, \ldots w_{n-1})$

N-gram language model:

$$P(w_i | w_{i-1}) = \frac{\mathrm{Count}(w_{i-1}, w_i)}{\mathrm{Count}(w_{i-1})}$$

Simple FFNN language model

Predicting next word w_t given prior words w_{t-1}, w_{t-2}, w_{t-3}, ... using **sliding windows (of fixed length)**



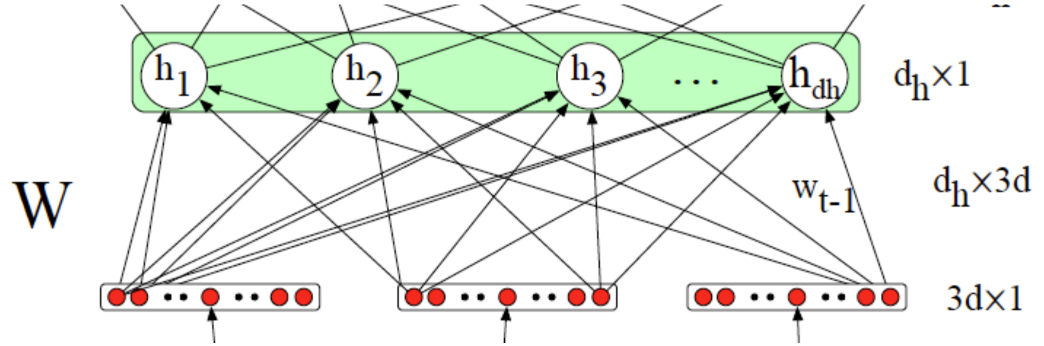
FFNN LMs

Improvements over n-gram LMs:

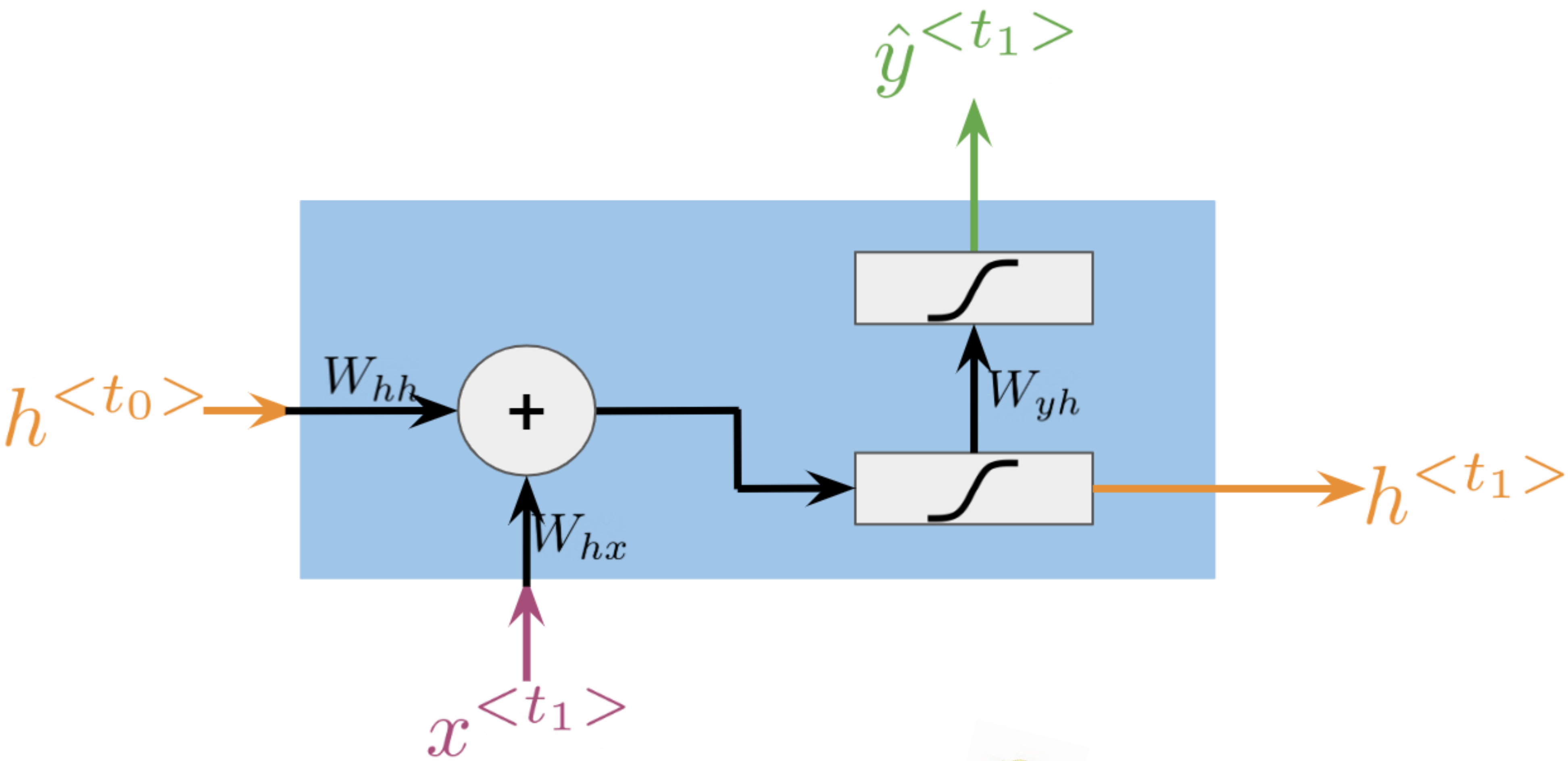
- No sparsity problem
- Don't need to store all observed n-grams
- Embeddings can generalize and predict unseen words

Remaining problems:

- Fixed window is **too small**
- Enlarging window enlarges W

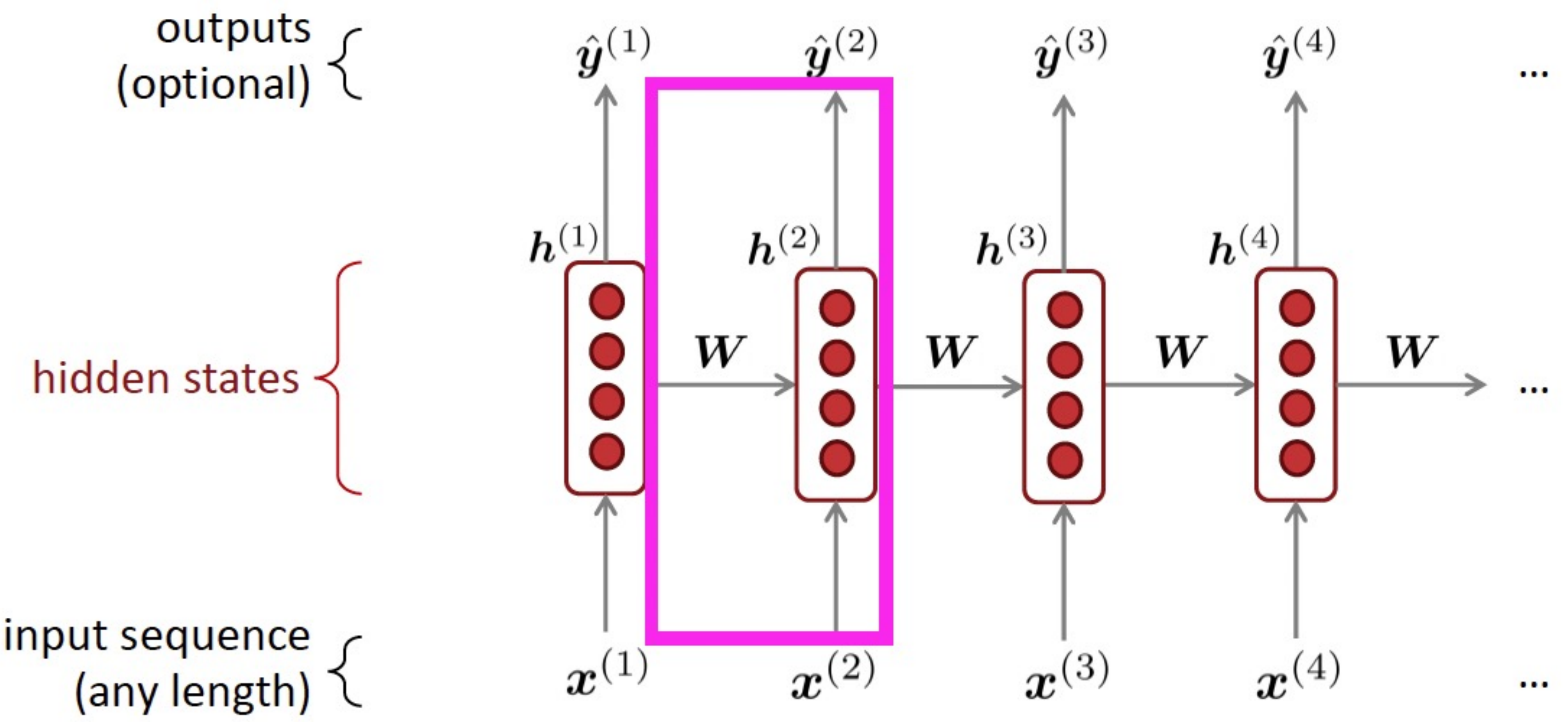


Recurrent Neural Networks (RNN)



simple/vanilla/Elman RNN

RNN basic structure



RNN language model

output distribution

$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$

hidden states

$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$

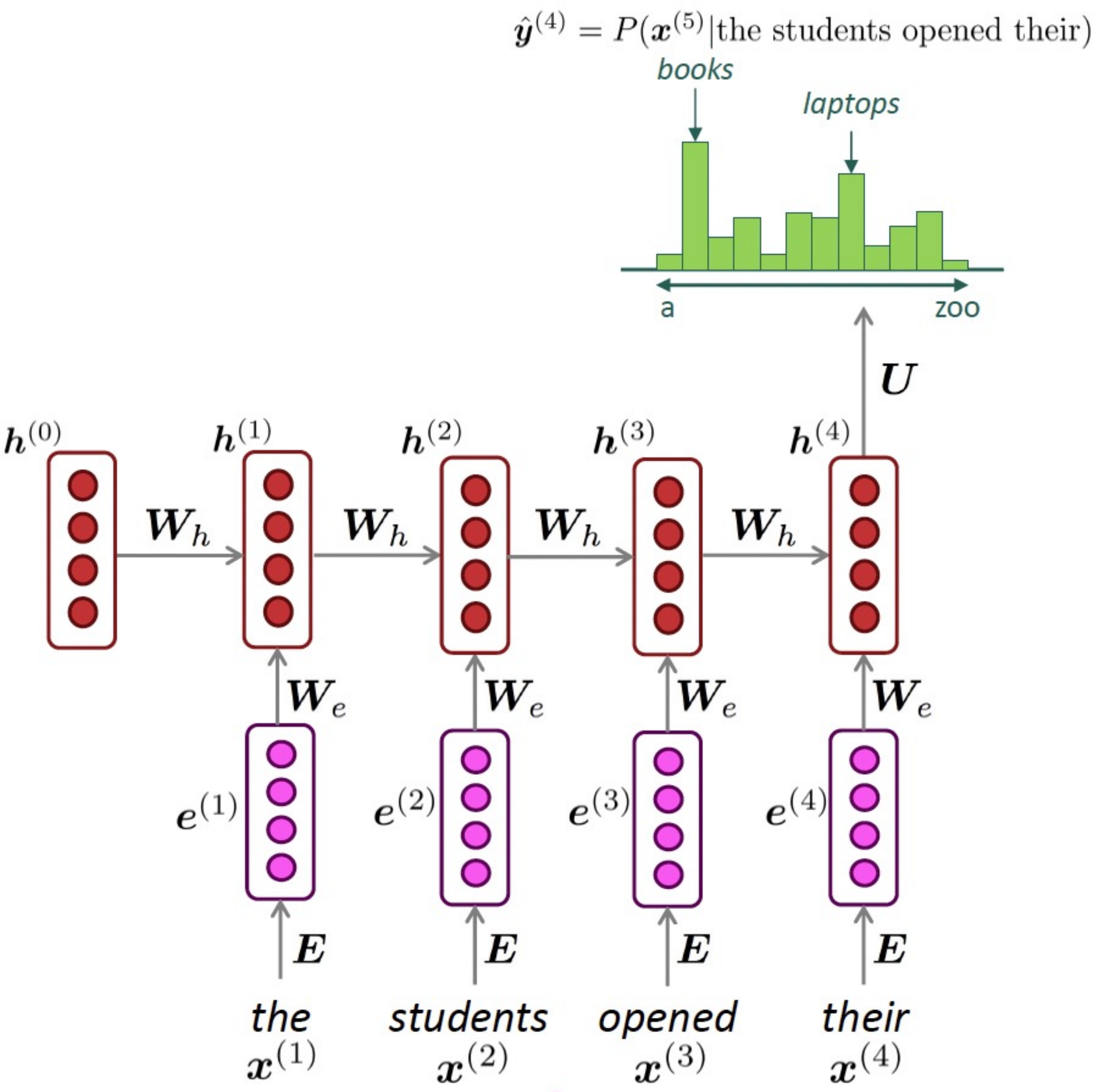
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$

words / one-hot vectors

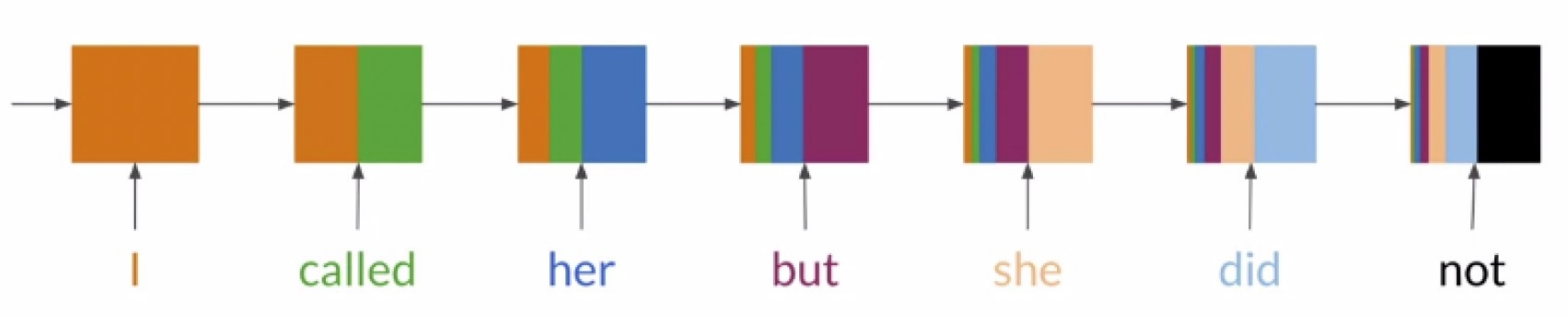
$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



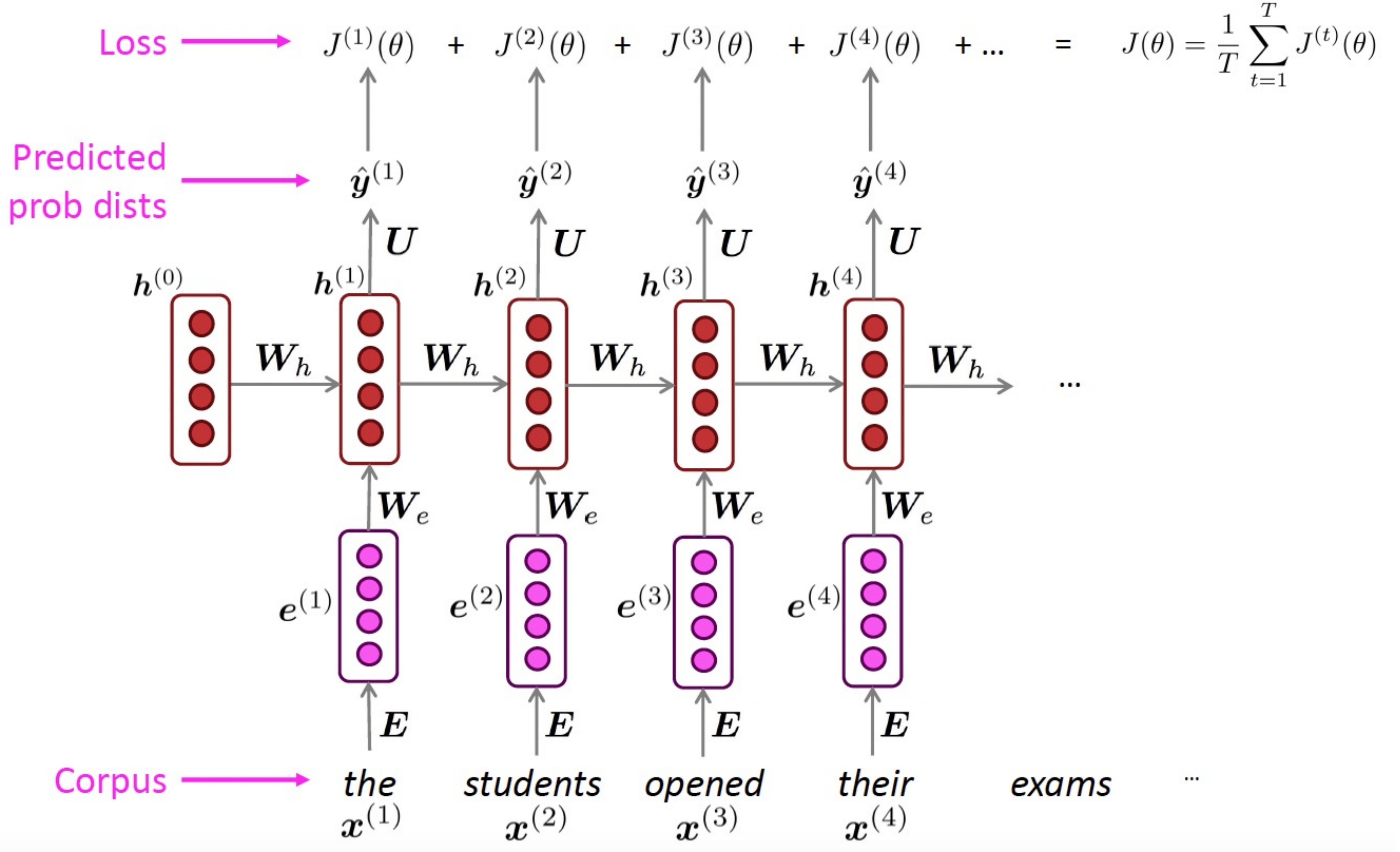
RNN language model

Advantages:

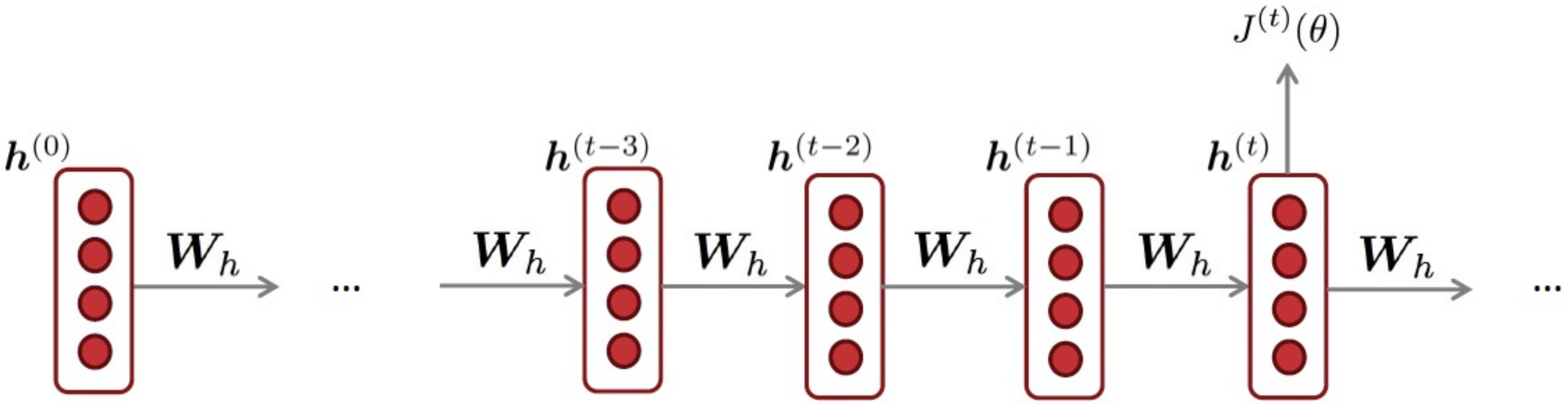
- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input context: Same weights applied on every timestep



Training an RNN LM



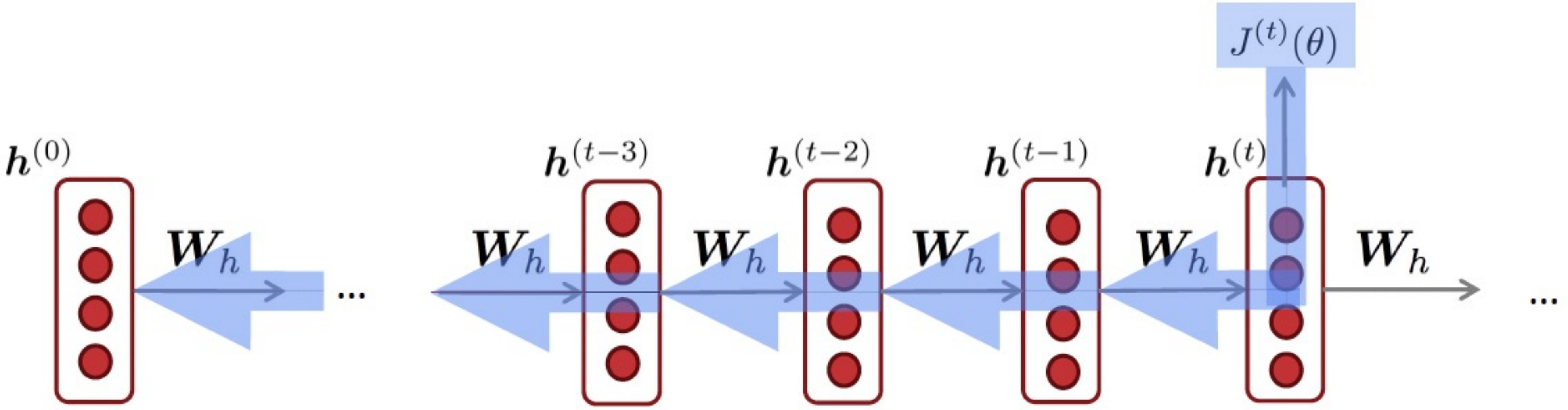
Backpropagation for RNNs



The derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix $\boldsymbol{W}_h$ is the sum of the gradient w.r.t. each time it appears

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} \right|_{(i)}$$

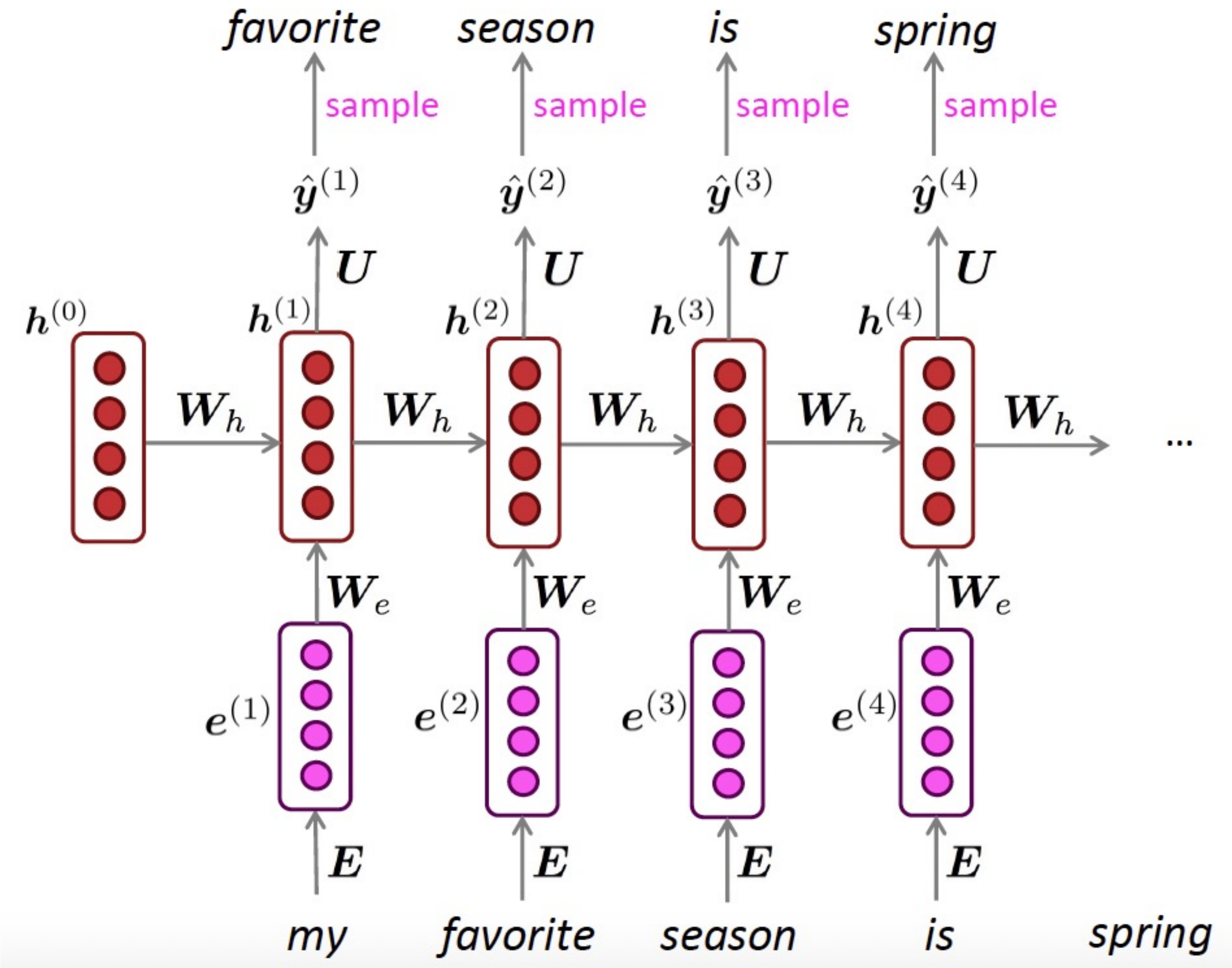
Backpropagation through time



Backpropagate over timesteps *i=t,…,0*, summing gradients as you go → **"backpropagation through time"**

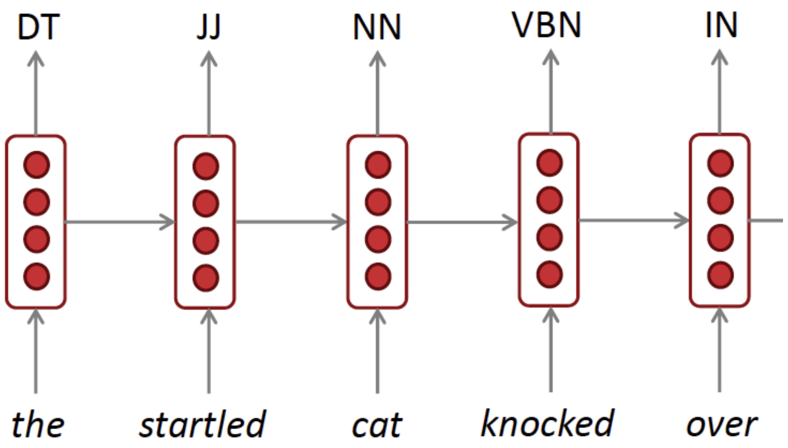
In practice, often "truncated" after ~20 timesteps for training efficiency reasons

Generating text with an RNN LM

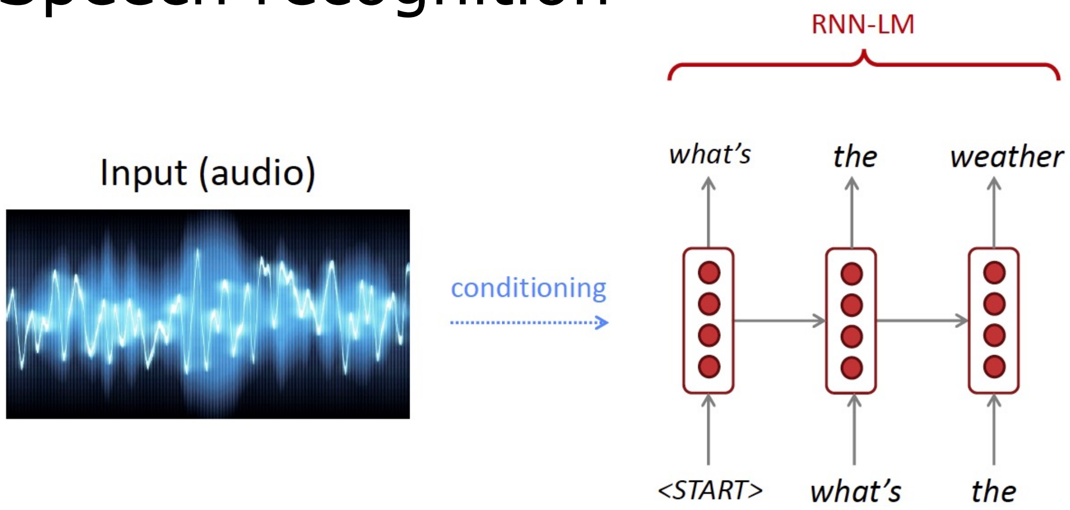


RNN Applications

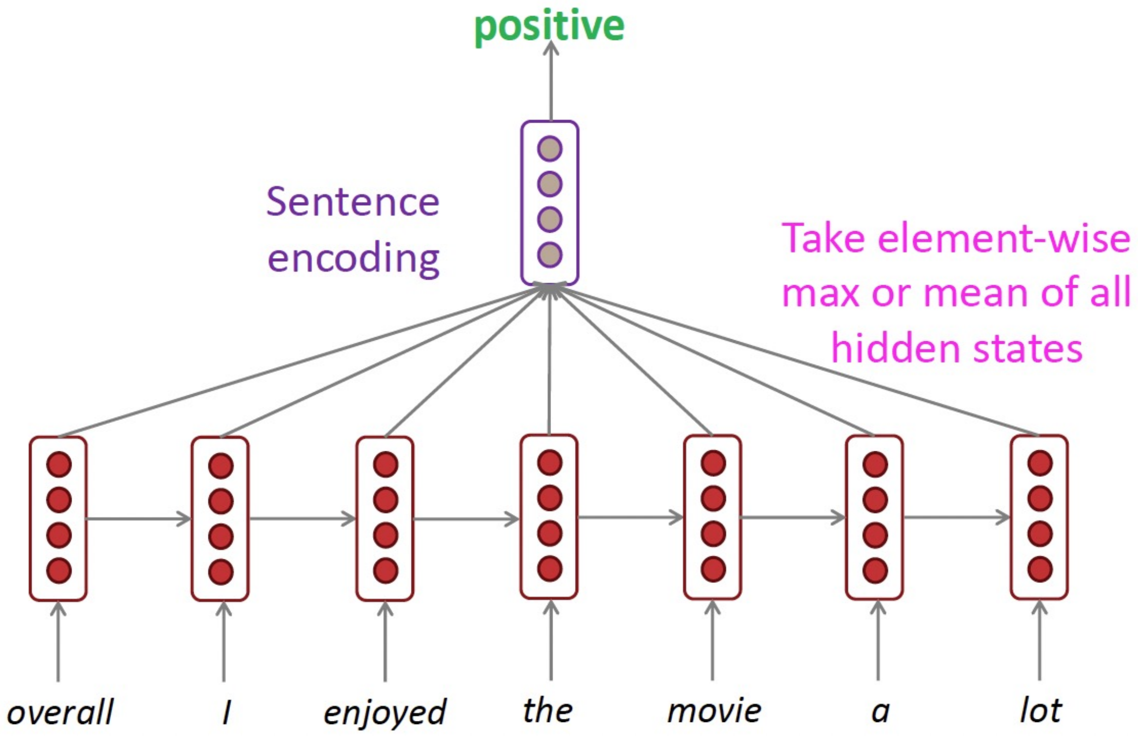
POS tagging



Speech recognition



Sentiment analysis



To do

- Optional reading: **SLP** Ch9
- Do HW9