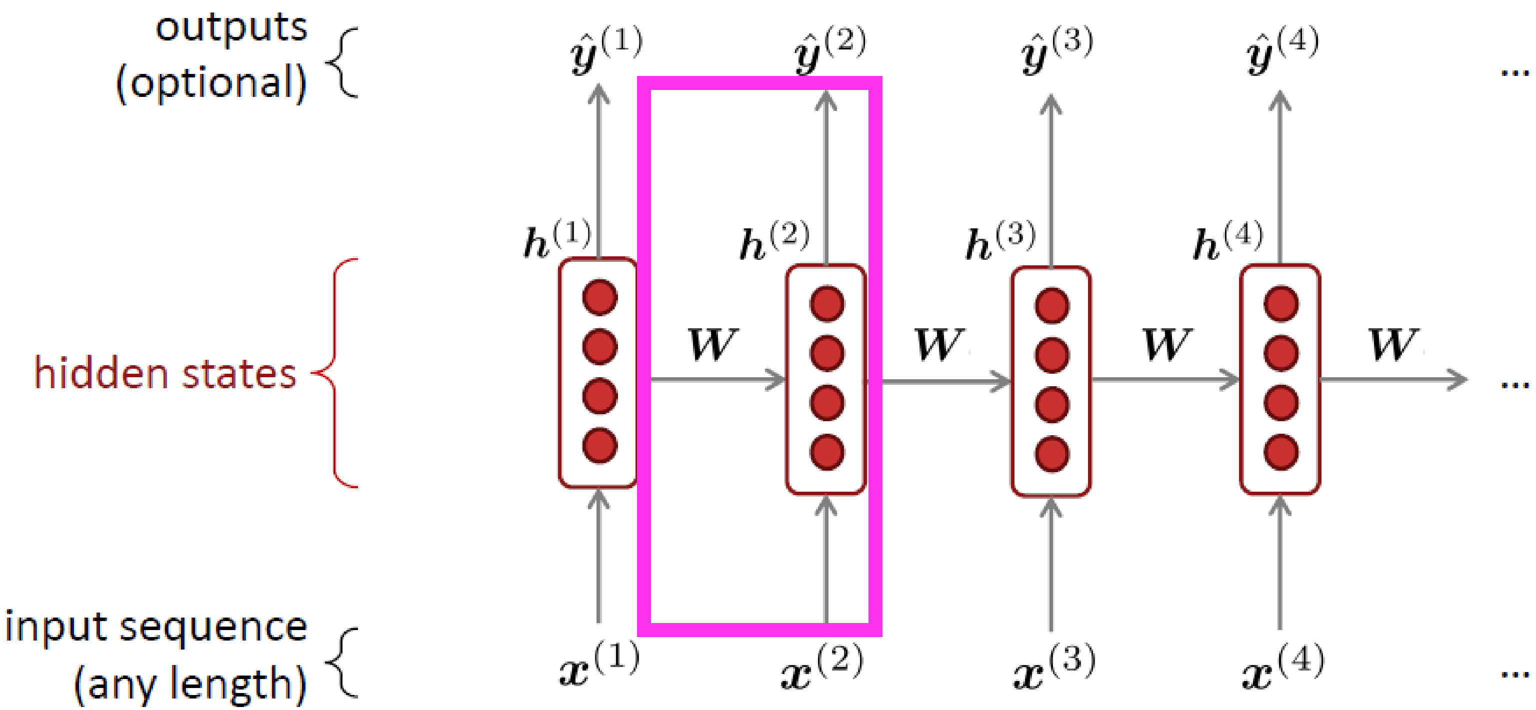
Recurrent Neural Networks (RNN)

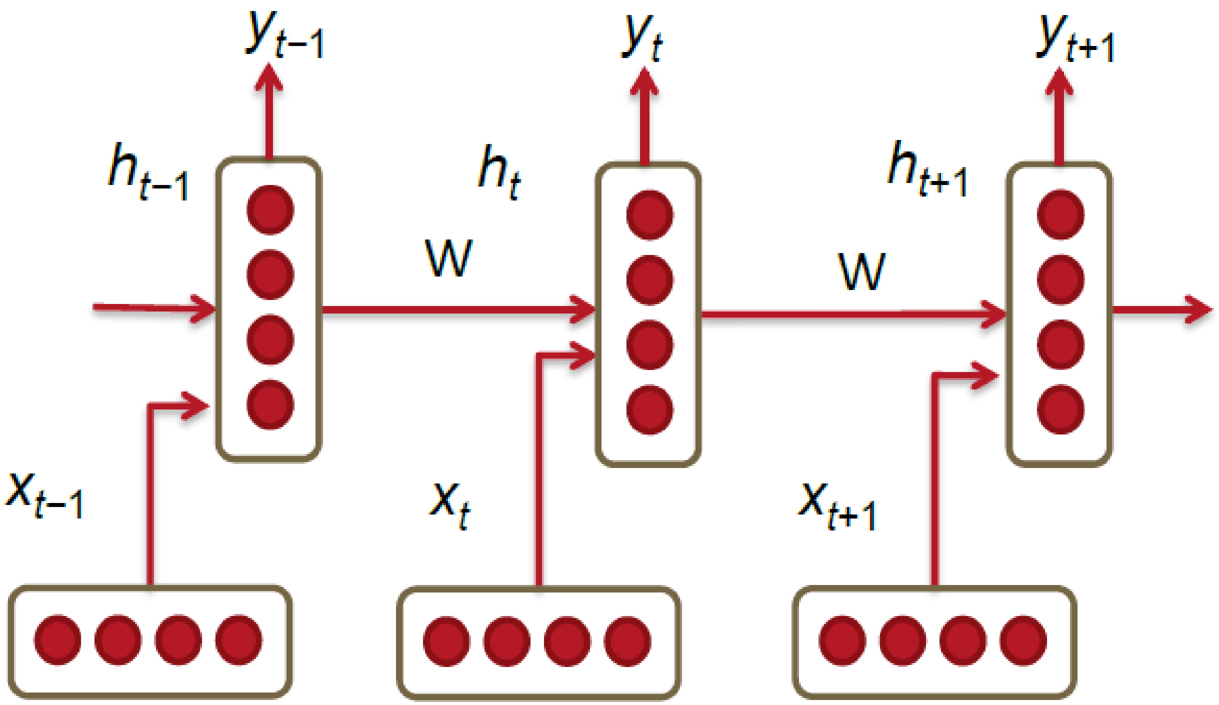
A family of neural architectures

Core idea: Apply the same weights *W repeatedly*



A Recurrent Neural Network (RNN). Three time-steps are shown.

Unlike the conventional translation models, where only a finite window of previous words would be considered for conditioning the language model, **Recurrent Neural Networks (RNN) are capable of conditioning the model on all previous words in the corpus.**

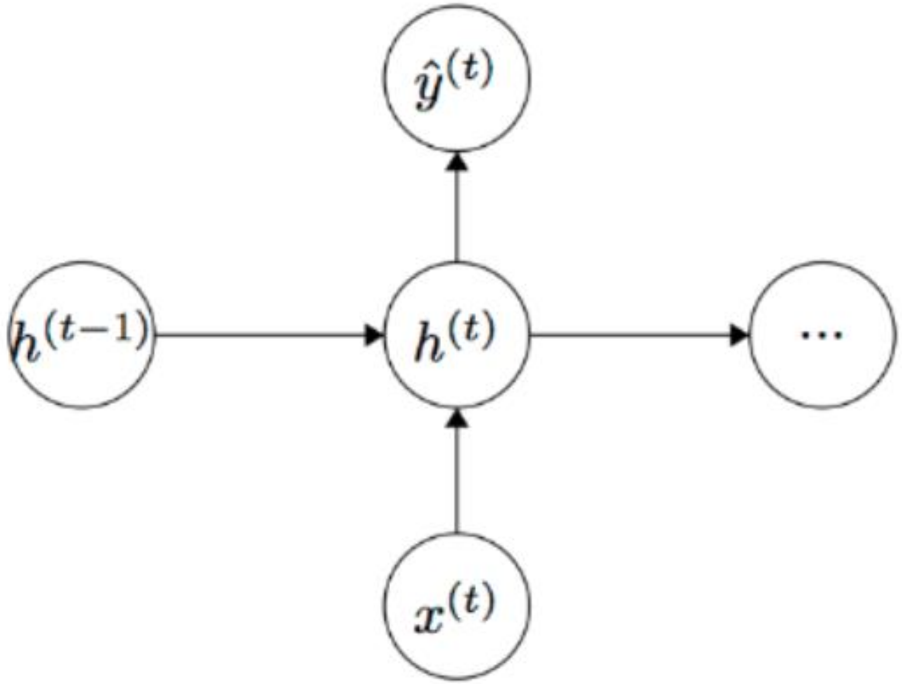


RNN architecture where each vertical rectangular box is a hidden layer at a time-step, t. Each such layer holds a number of neurons, each of which performs a linear matrix operation on its inputs followed by a non-linear operation (e.g. tanh()). At each time-step, there are two inputs to the hidden layer: the output of the previous layer h_{t-1}, and the input at that timestep x_t. The former input is multiplied by a weight matrix $W^{(hh)}$ and the latter by a weight matrix W(hx) to produce output features h_t, which are multiplied with a weight matrix W(S) and run through a softmax over the vocabulary to obtain a prediction output ˆ y of the next word

The inputs and outputs to a neuron of a RNN

$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]})$$

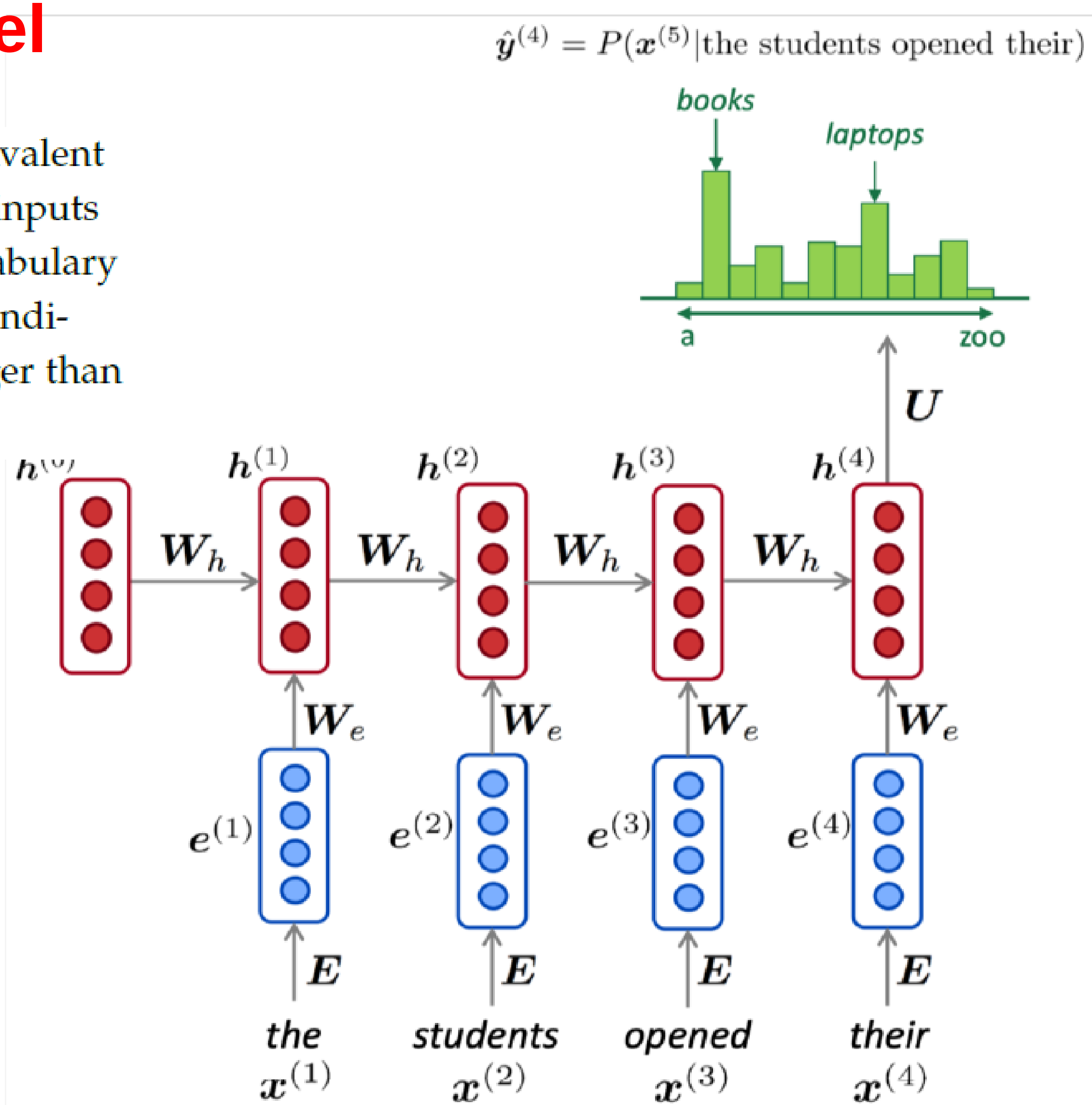
$$\hat{y}_t = softmax(W^{(S)}h_t)$$



What is interesting here is that the same weights $W^{(hh)}$ and $W^{(hx)}$ are applied repeatedly at each timestep. Thus, the number of parameters the model has to learn is less, and most importantly, is independent of the length of the input sequence - thus defeating the curse of dimensionality!

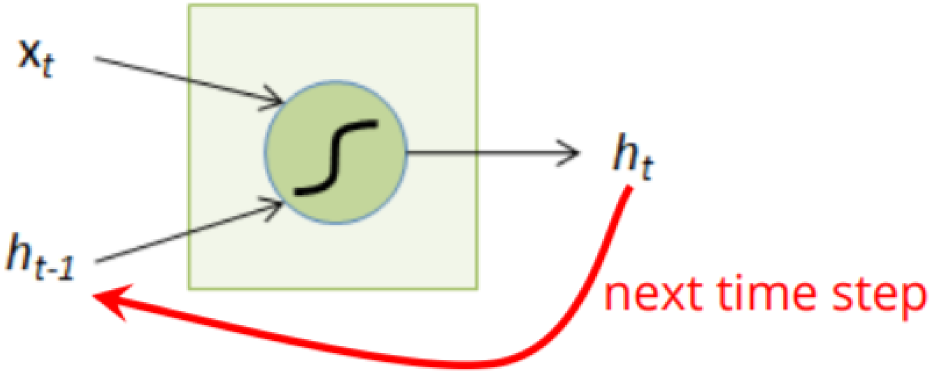
An RNN Language Model

The notation in this image is slightly different: here, the equivalent of $W^{(hh)}$ is W_h, $W^{(hx)}$ is W_e, and $W^{(S)}$ is U. E converts word inputs $x^{(t)}$ to word embeddings $e^{(t)}$. The final softmax over the vocabulary shows us the probability of various options for token $x^{(5)}$, conditioned on all previous tokens. The input could be much longer than 4-5 tokens.



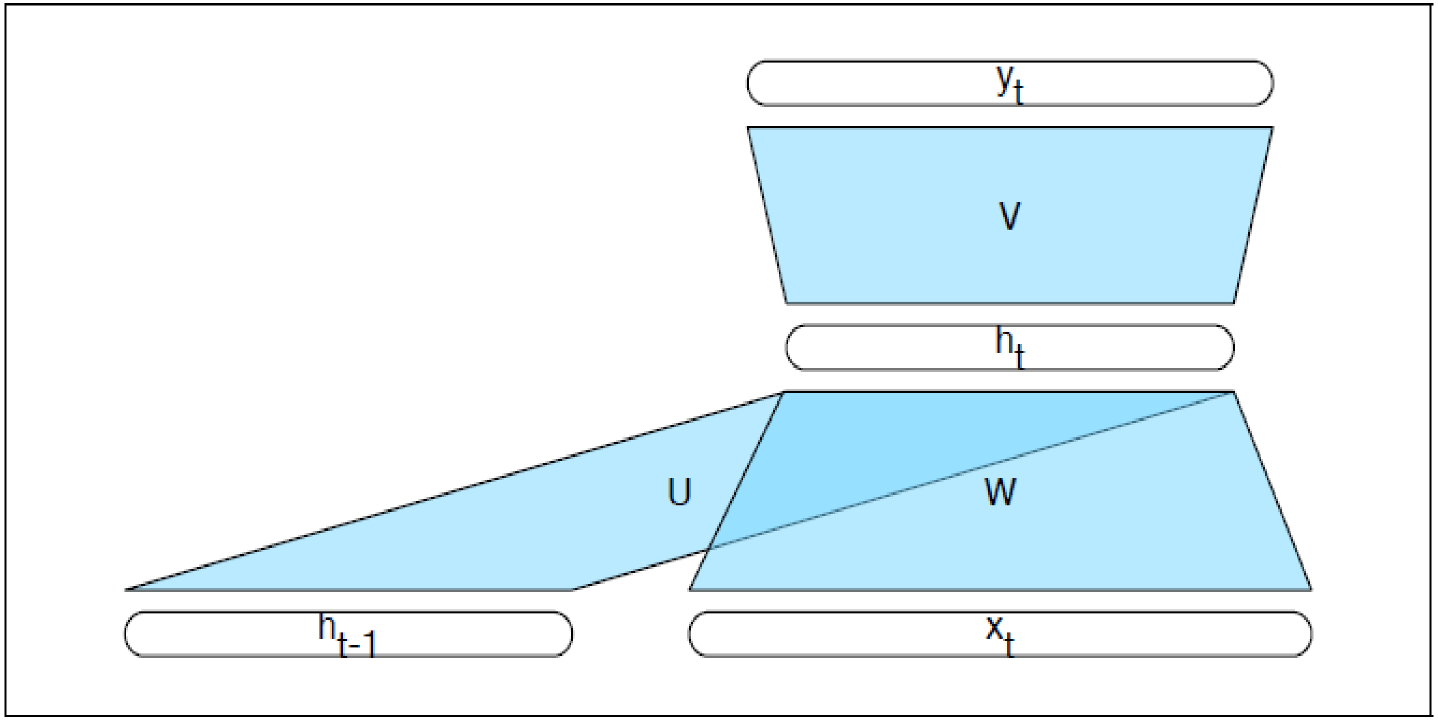
The Recurrent Neuron

- x_t: Input at time t
- h_{t-1}: State at time t-1



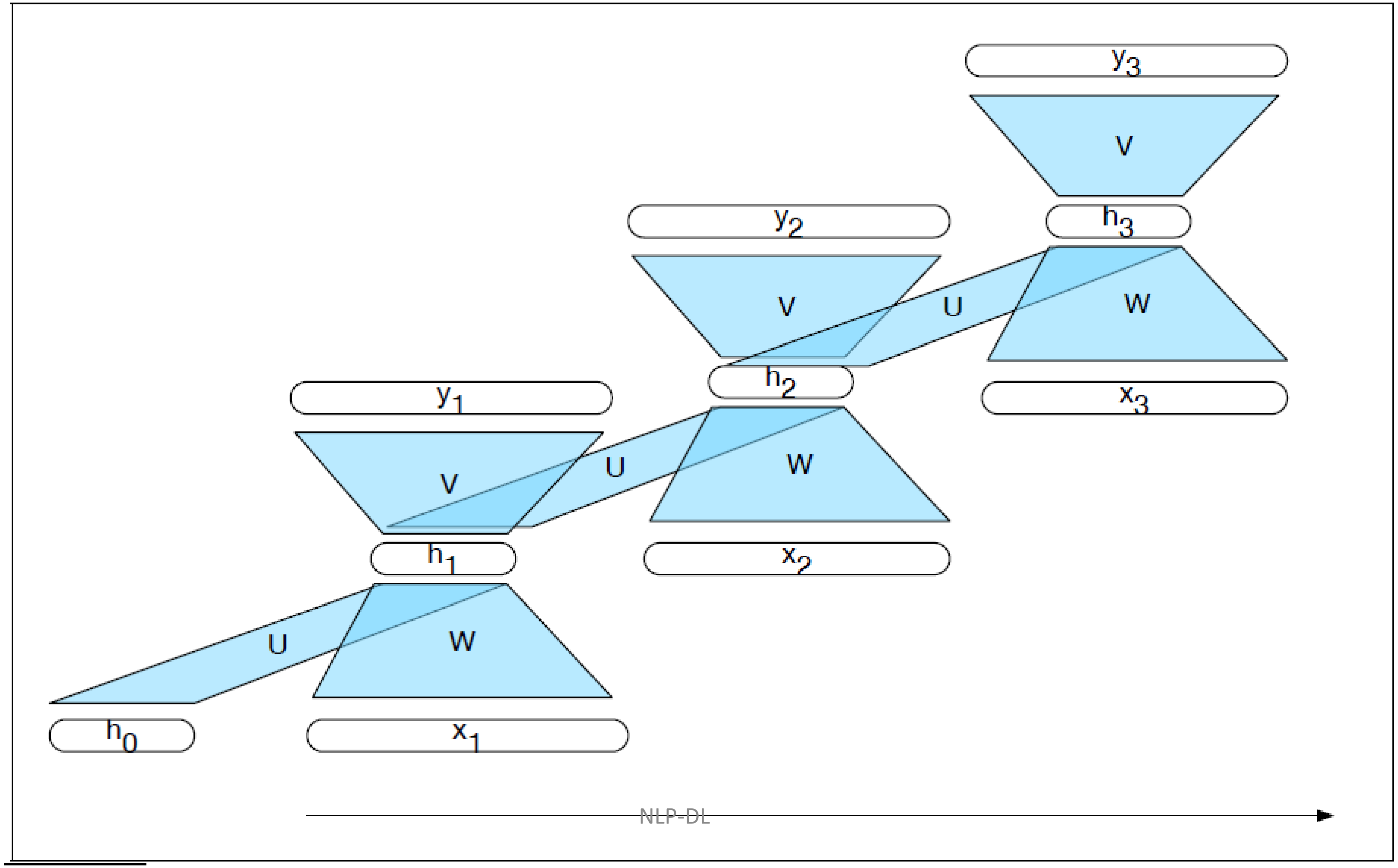
$$h_t = f(W_h h_{t-1} + W_x x_t)$$

Simple recurrent neural network illustrated as a feedforward network.



A simple recurrent neural network shown unrolled in time.

Network layers are copied for each time step, while the weights U, V and W are shared in common across all time steps.



A Simple 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

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

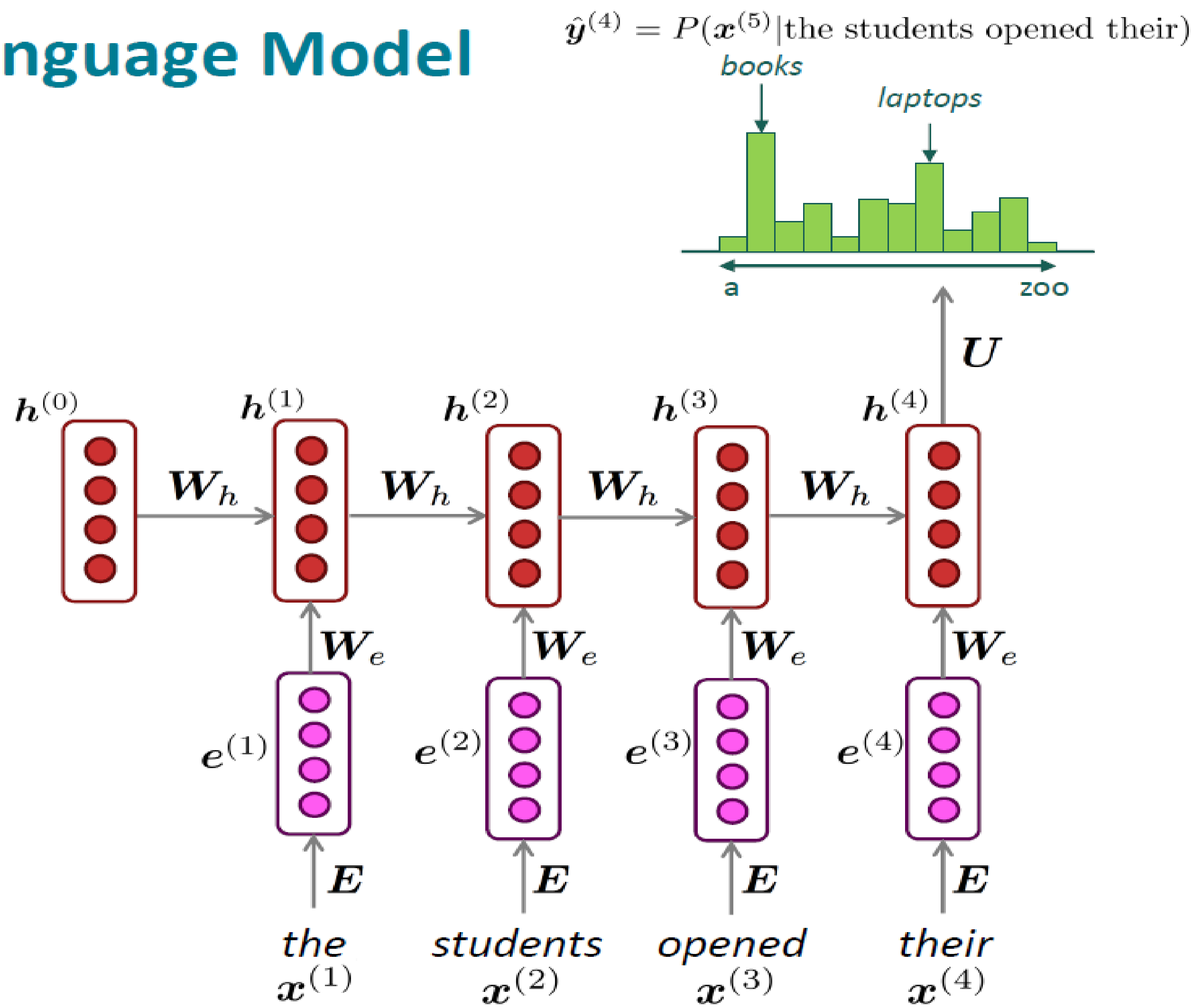
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

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



***Note**: this input sequence could be much longer now!*

RNN Language Models

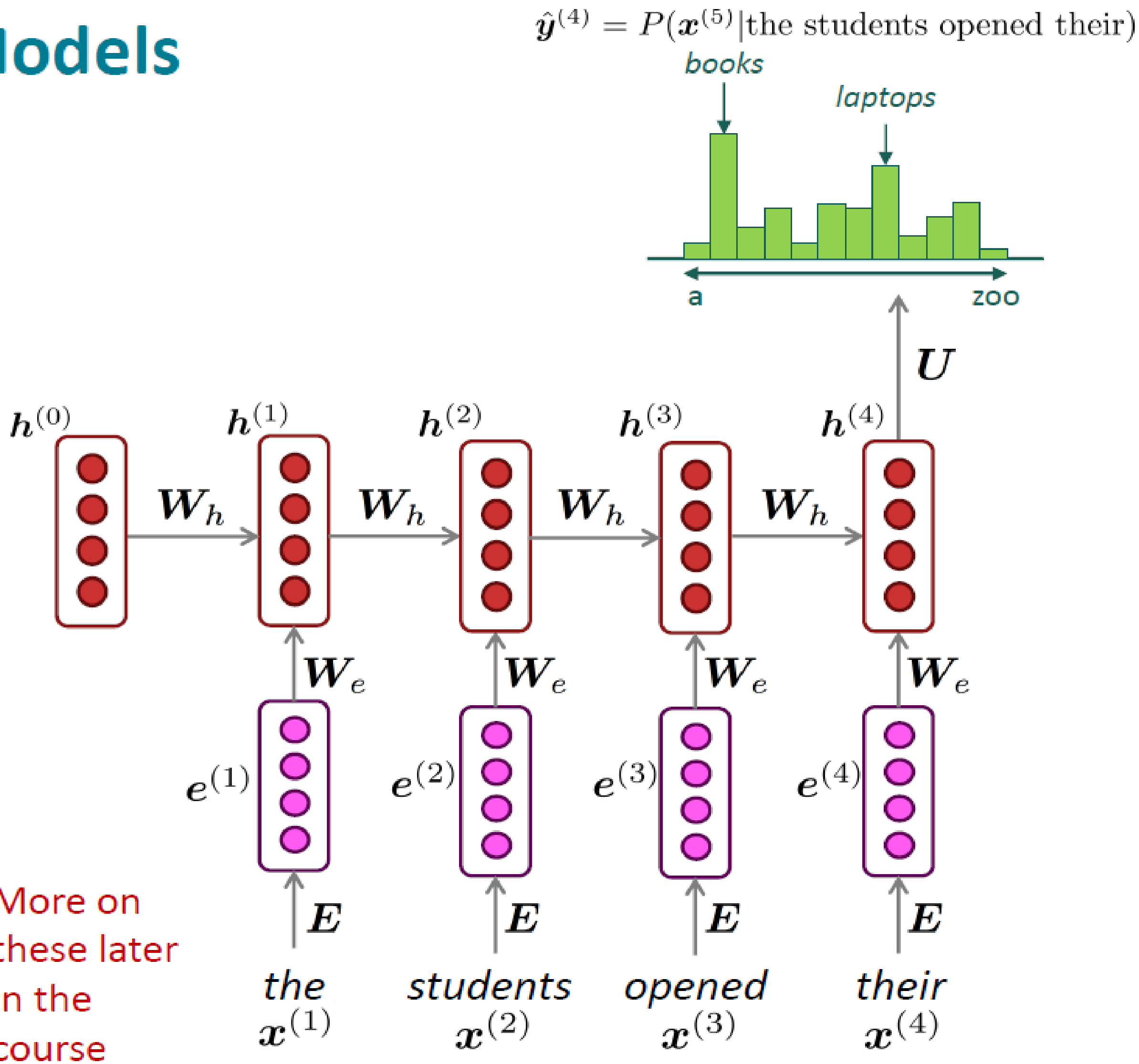
RNN **Advantages**:

- 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, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course



- **The amount of memory required to run a layer of RNN is proportional to the number of words in the corpus.**
- We can consider a sentence as a minibatch, and a sentence with k words would have k word vectors to be stored in memory.
- Also, the RNN must maintain two pairs of W, b matrices. As aforementioned, while the size of W could be very large, it does not scale with the size of the corpus (unlike the traditional language models).
- For a RNN with 1000 recurrent layers, the matrix would be 1000 X 1000 regardless of the corpus size.