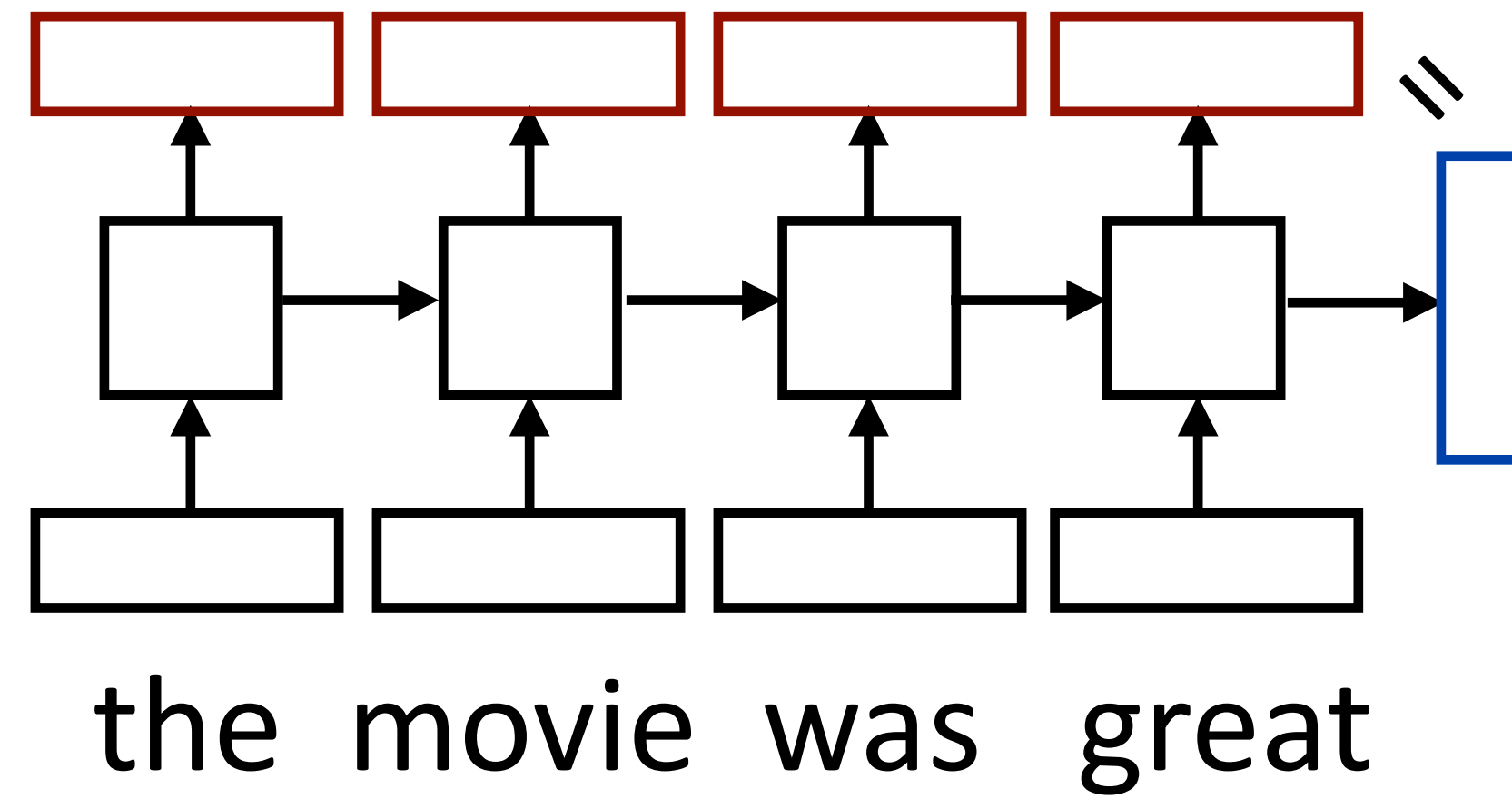# Using RNNs

What do RNNs produce?


the movie was great

▸ Encoding of the sentence (final h_i/c_i) — can pass this a decoder or make a classification decision about the sentence

▸ Encoding of each word (each h_i) — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)

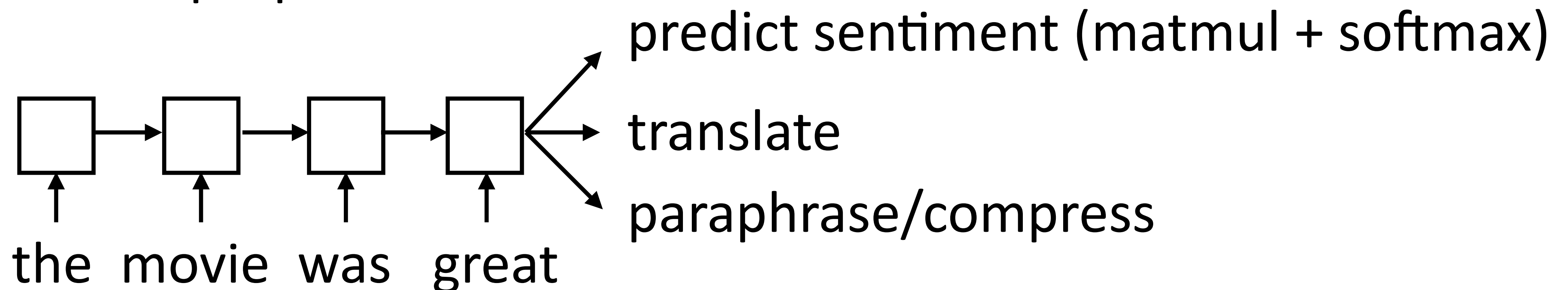▸ RNN can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors

RNN Uses

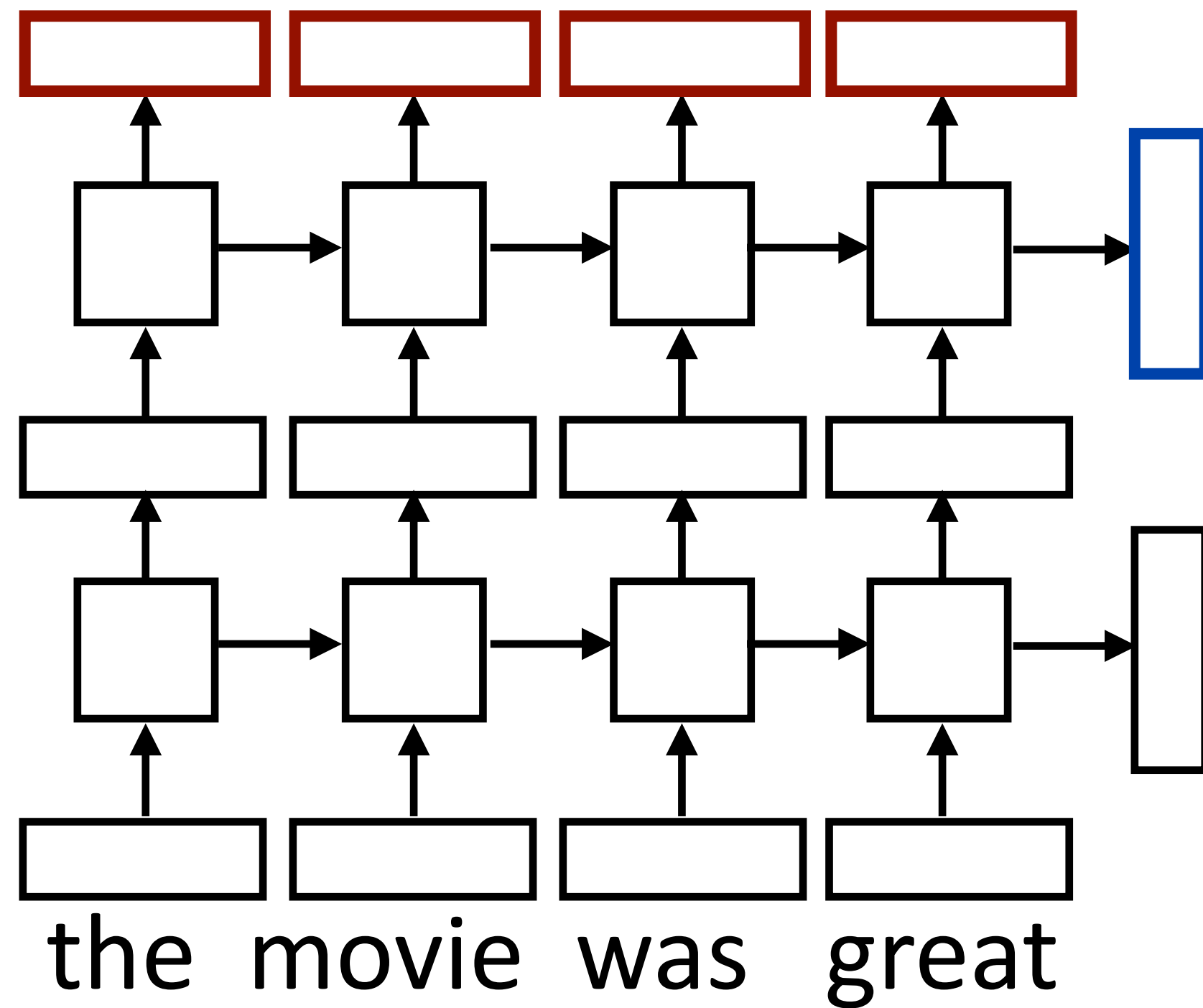▸ Transducer: make some prediction for each element in a sequence

DT   NN   VBD   JJ
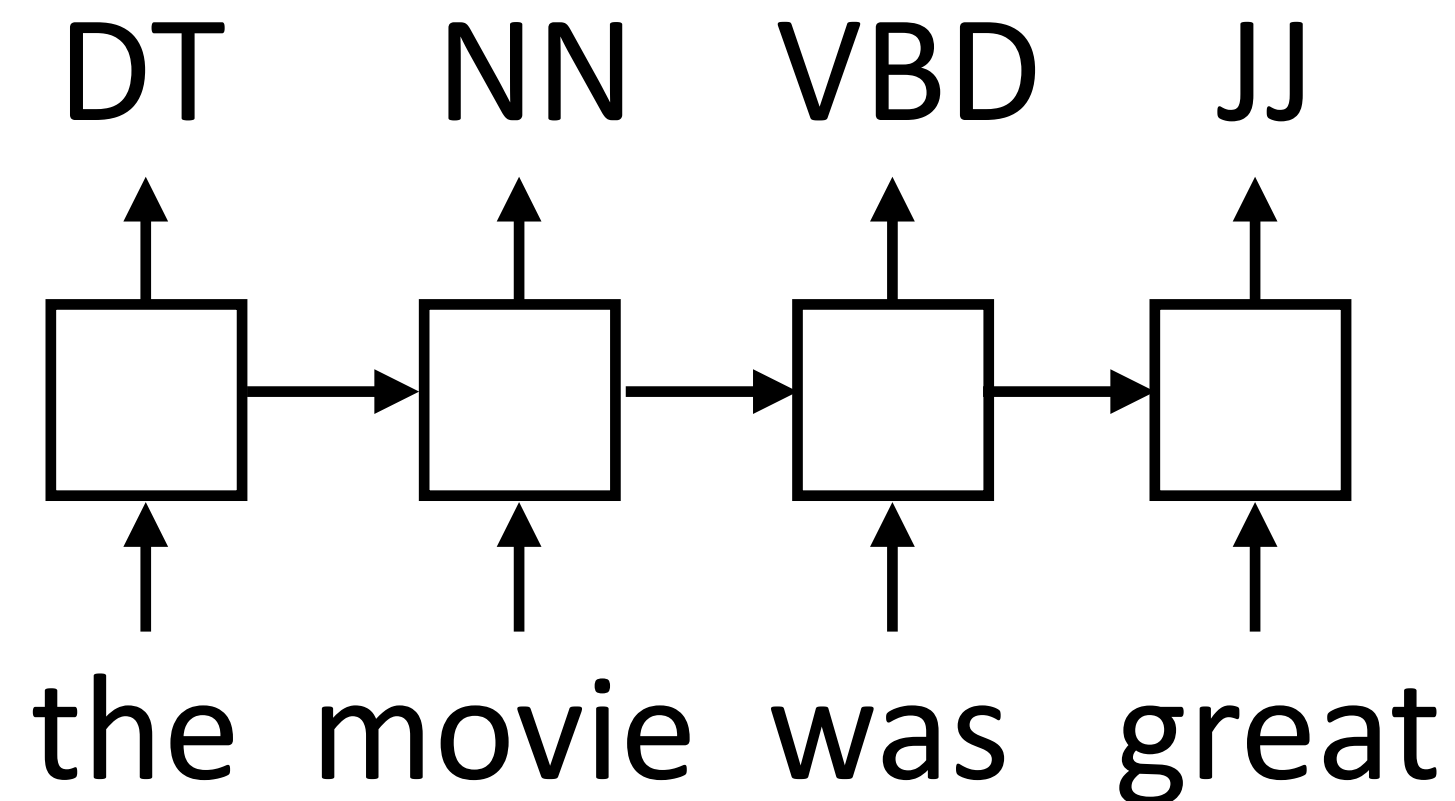
output **y** = score for each tag, then softmax

the movie was great

▸ Acceptor/encoder: encode a sequence into a fixed-sized vector and use that for some purpose

predict sentiment (matmul + softmax)

translate

paraphrase/compress

the movie was great

Multilayer Bidirectional RNN

the movie was great

the movie was great
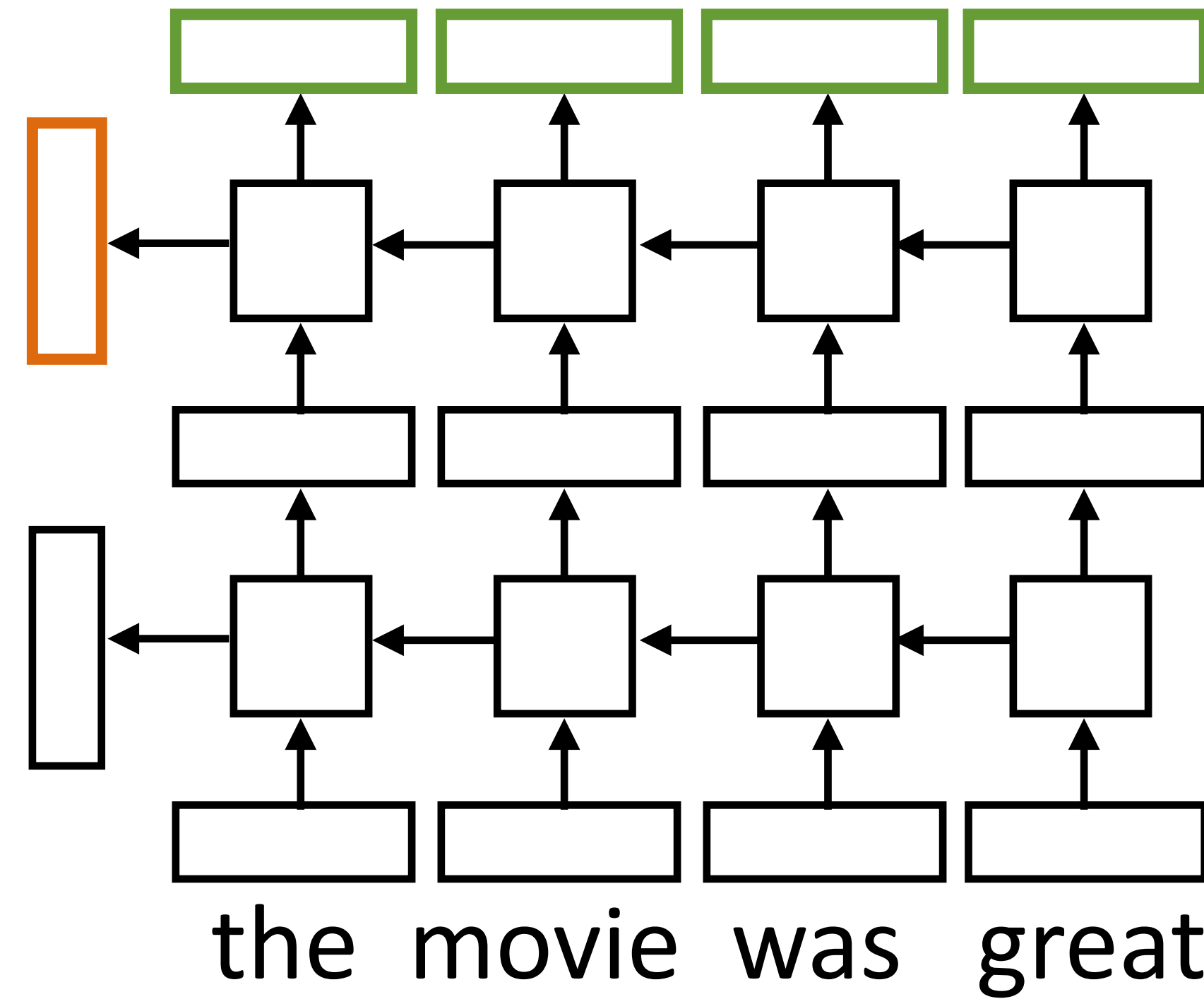
▸ Sentence classification based on concatenation of both final outputs
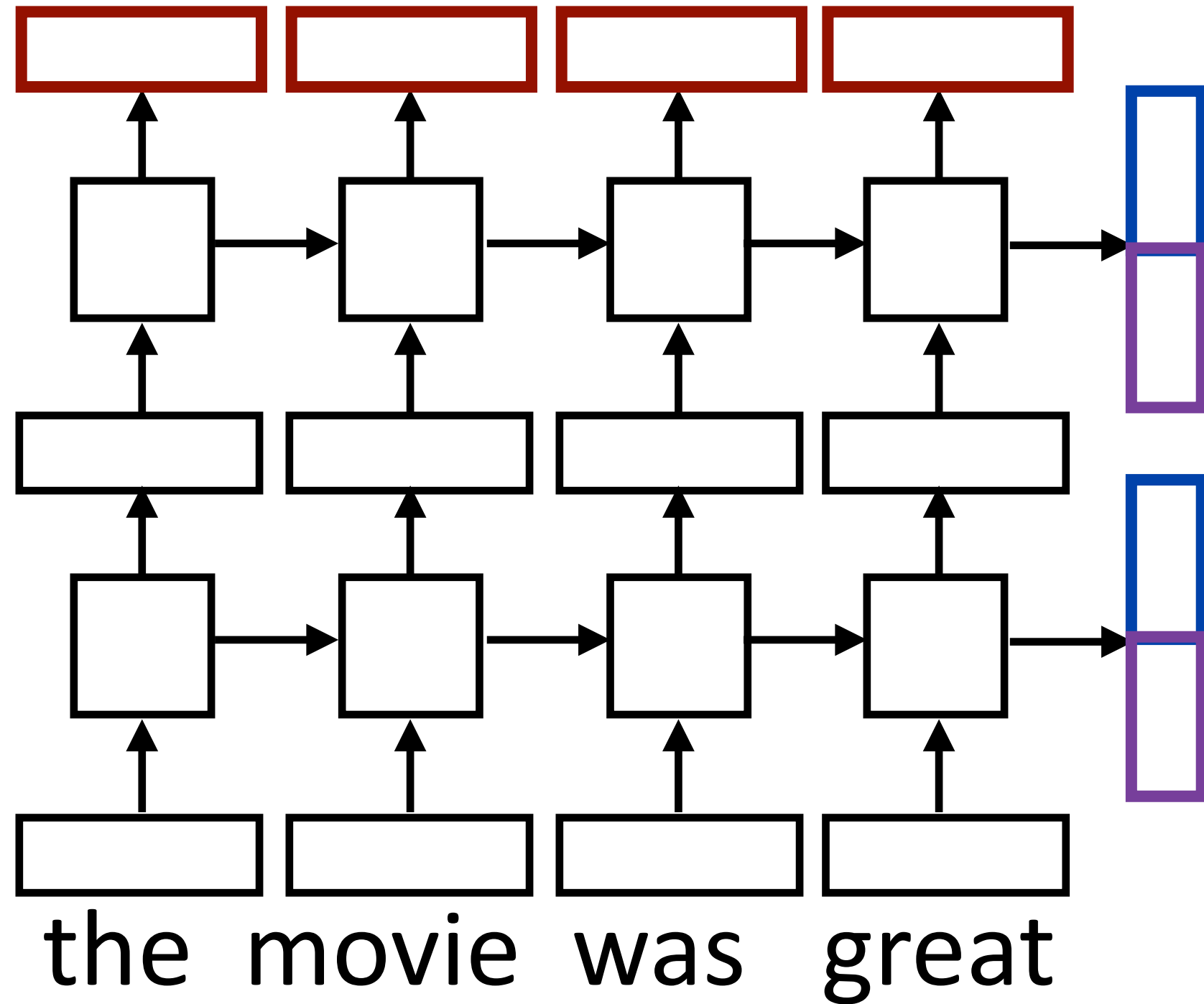
▸ Token classification based on concatenation of both directions' token representations
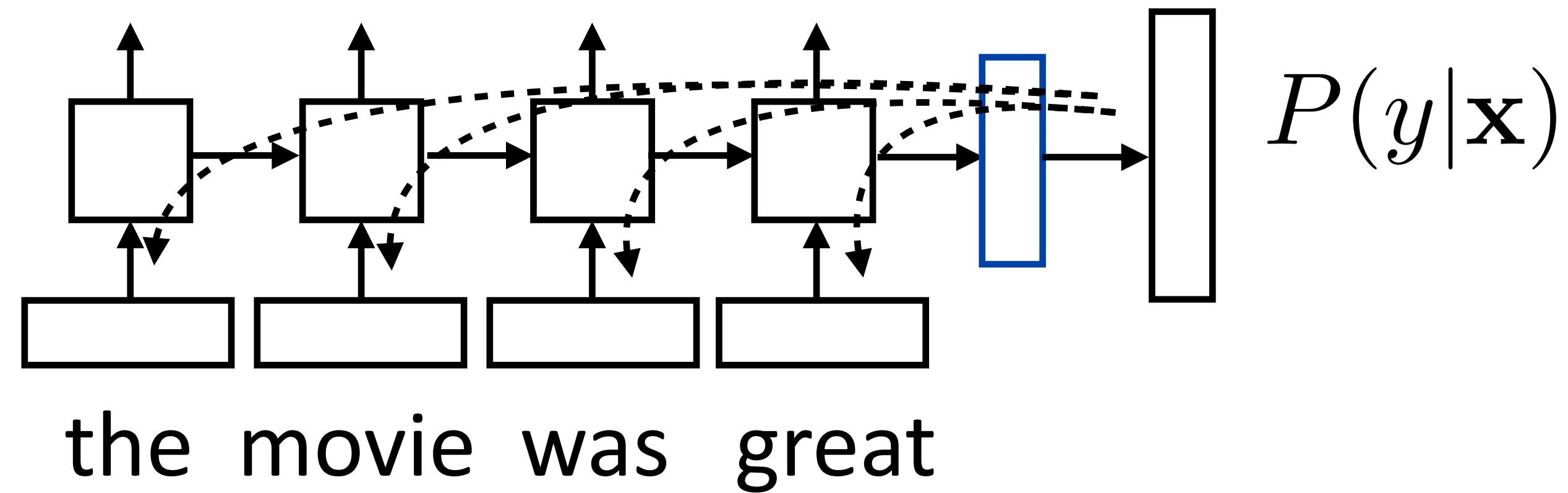
What do LSTMs return in PyTorch?

the movie was great

- ▸ Hidden/cell states are a 2-tuple, tensors of size [num_layers * num_directions, batch size, dimensionality]
  - ▸ 2x1xdim here

- ▸ Outputs are a single tensor of size [seq_len, batch size, num_directions * hidden_size]
  - ▸ 4x1xdim here

Training RNNs

$$P(y|\mathbf{x})$$

the movie was great
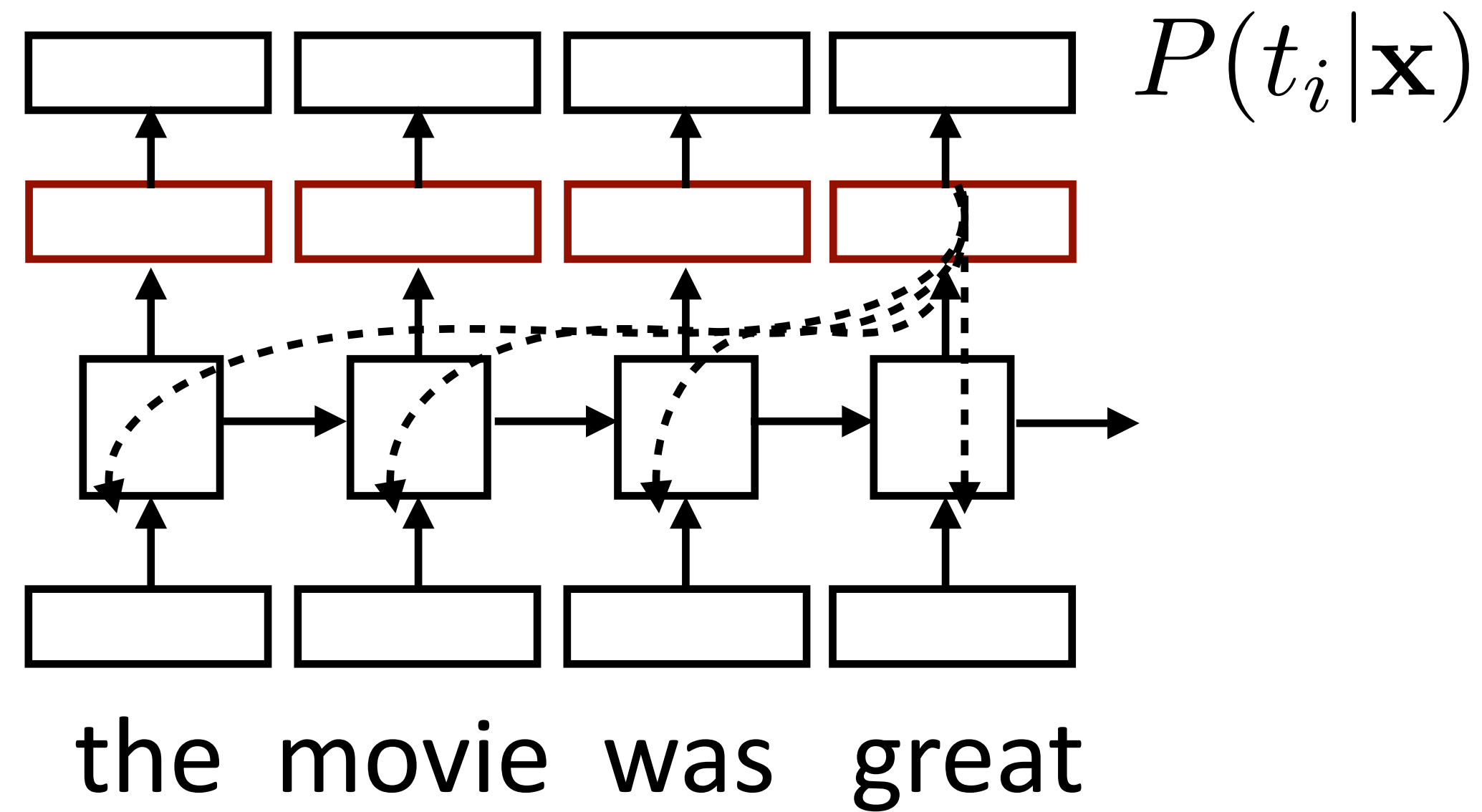
▸ Loss = negative log likelihood of probability of gold label (or use SVM or other loss)

▸ Backpropagate through entire network, RNN parameters get a gradient update from each timestep

▸ Example: sentiment analysis

Training RNNs

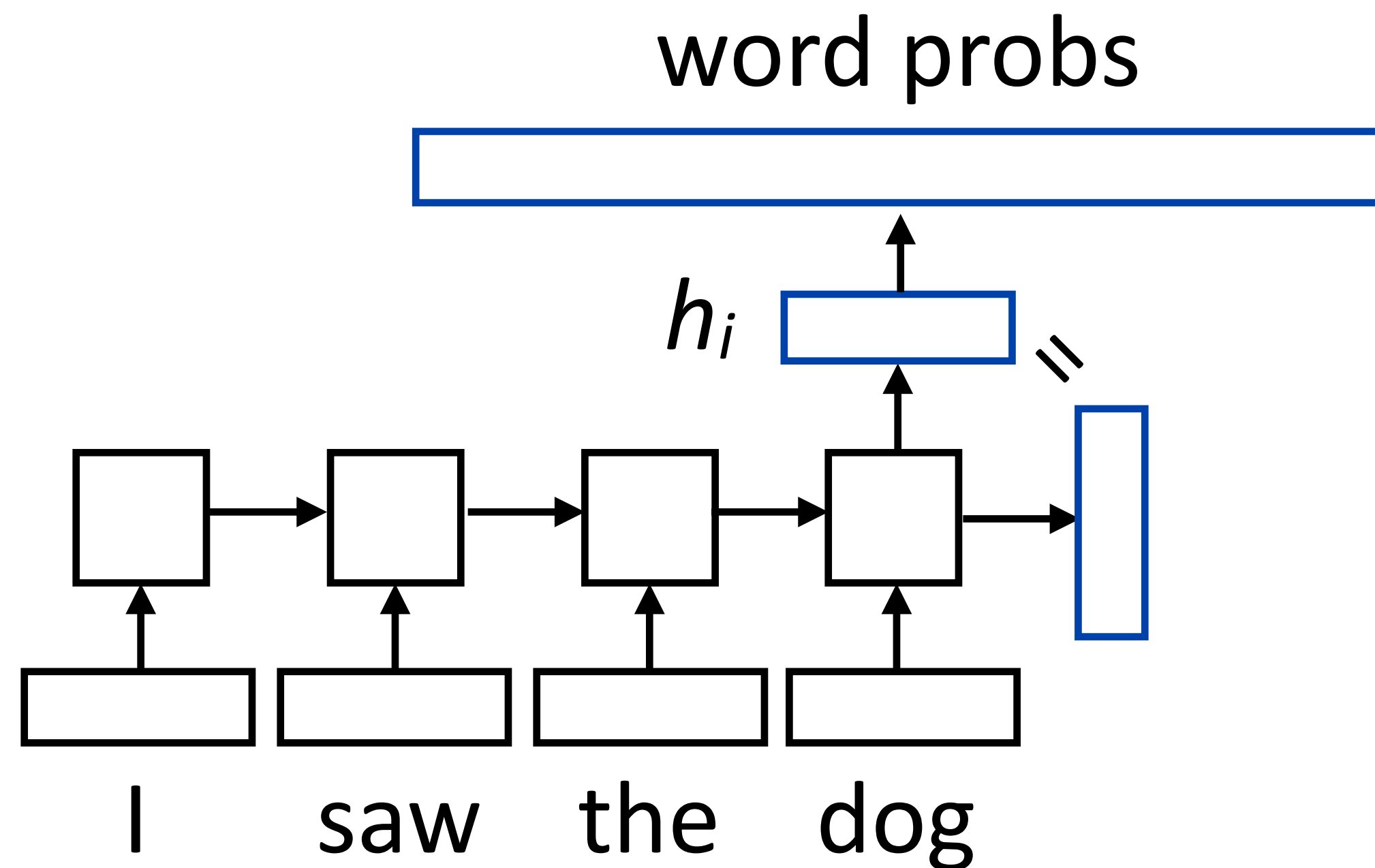$$P(t_i | \mathbf{x})$$

the movie was great

- ▸ Loss = negative log likelihood of probability of gold predictions, summed over the tags

- ▸ Loss terms filter back through network

- ▸ Example: language modeling (predict next word given context)

RNN Language Modeling

RNN Language Modeling

word probs

$$P(w|\text{context}) = \frac{\exp(\mathbf{w} \cdot \mathbf{h_i})}{\sum_{w'} \exp(\mathbf{w'} \cdot \mathbf{h_i})}$$

h_i

equivalent to

I   saw   the   dog
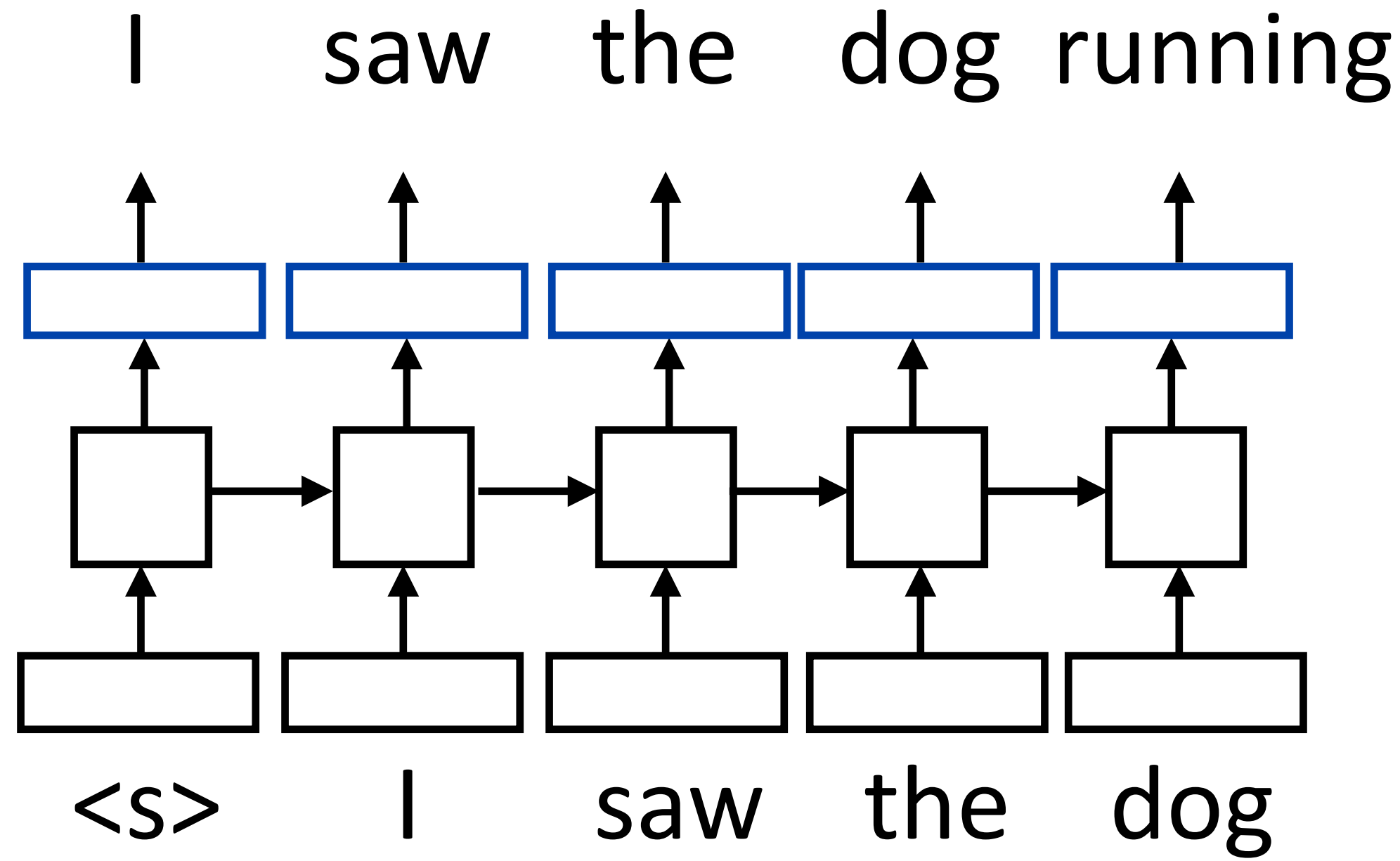
$$P(w|\text{context}) = \text{softmax}(W\mathbf{h}_i)$$

▸ *W* is a (vocab size) x (hidden size) matrix; linear layer in PyTorch (rows are word embeddings)

Training RNNLMs

I   saw   the   dog   running

<s>   I   saw   the   dog

▸ Input is a sequence of words, output is those words shifted by one,

▸ Allows us to train on predictions across several timesteps simultaneously (similar to batching but this is NOT what we refer to as batching)
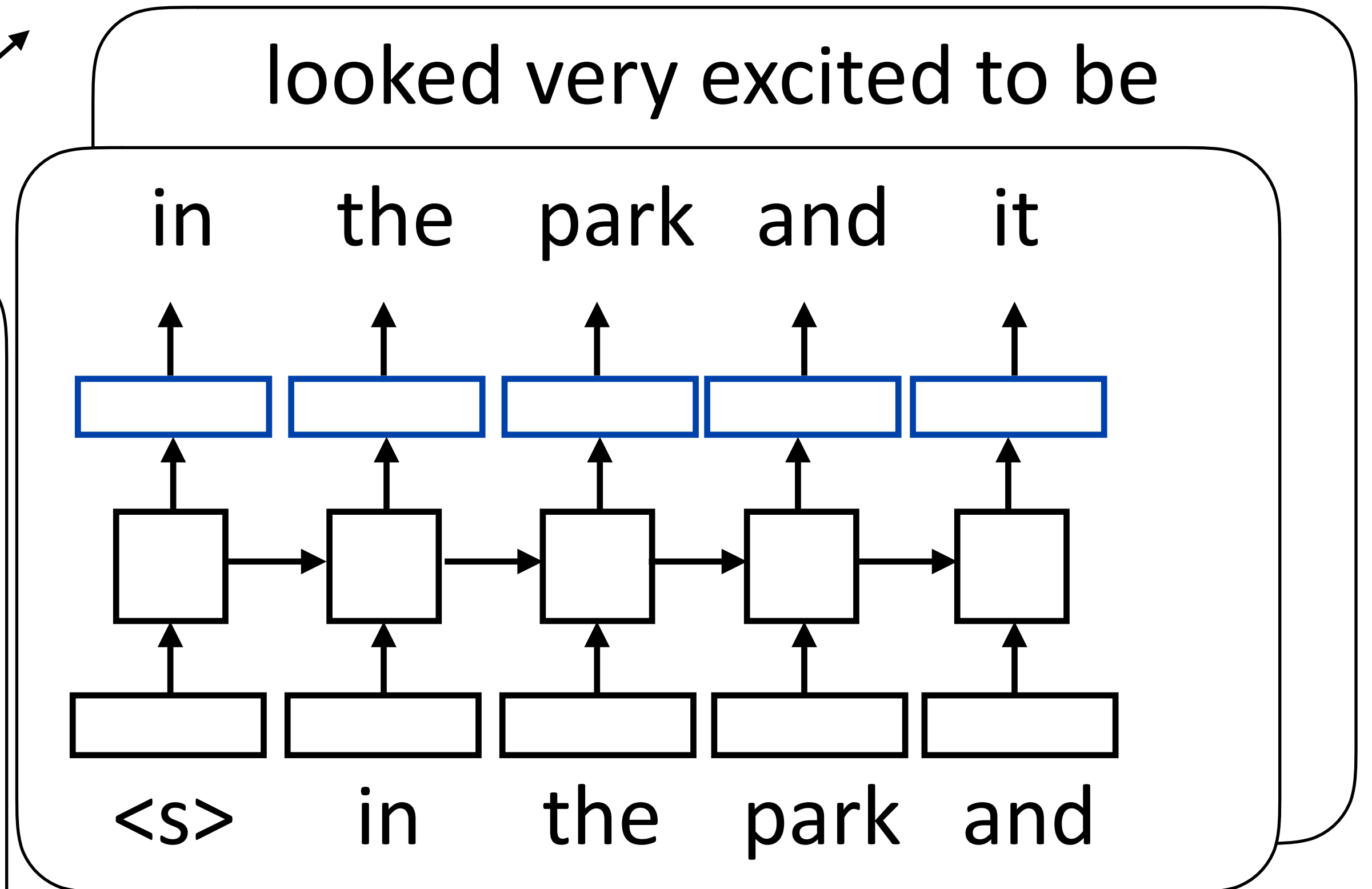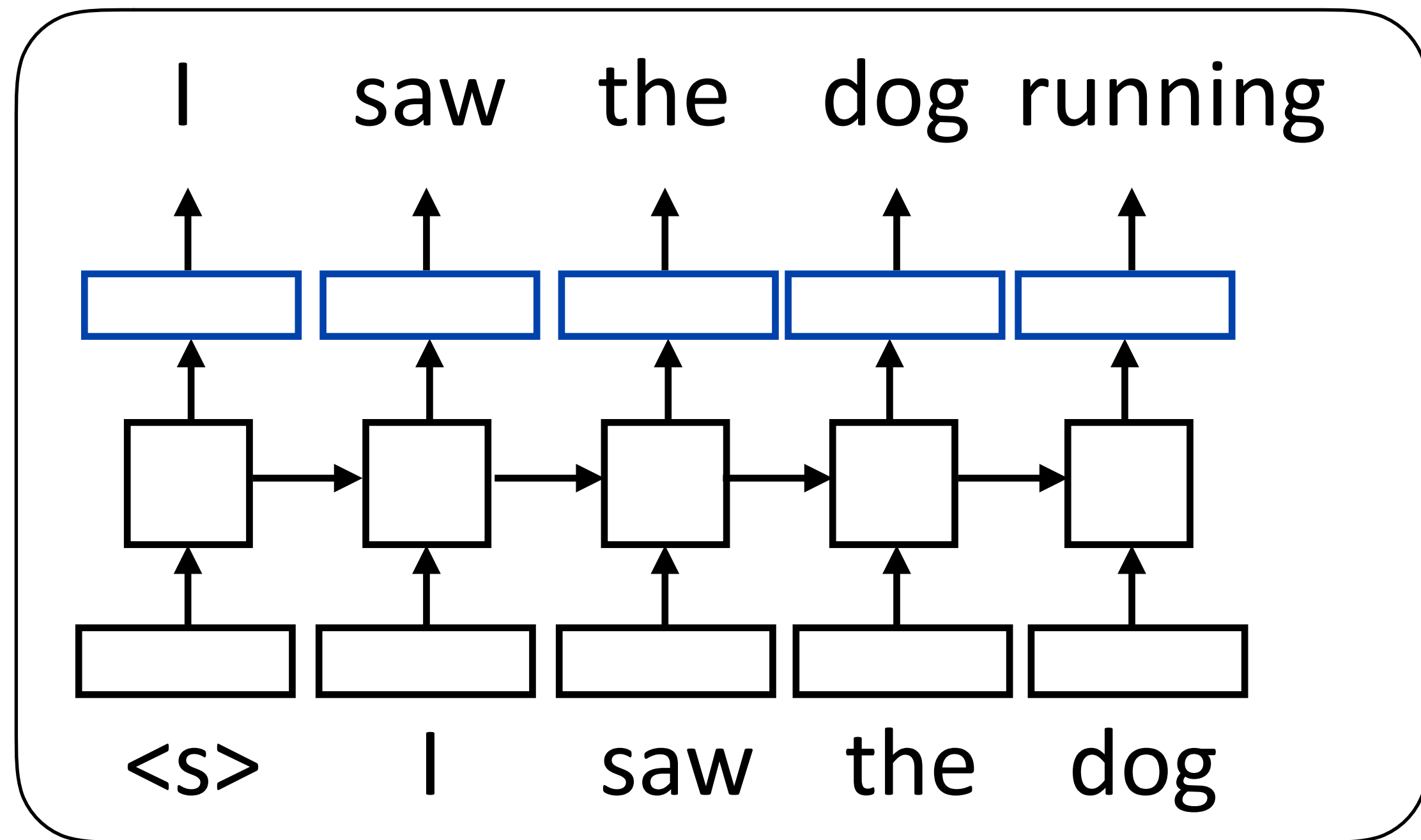
Training RNNLMs

P(w|context)

loss = — log P(w*|context)
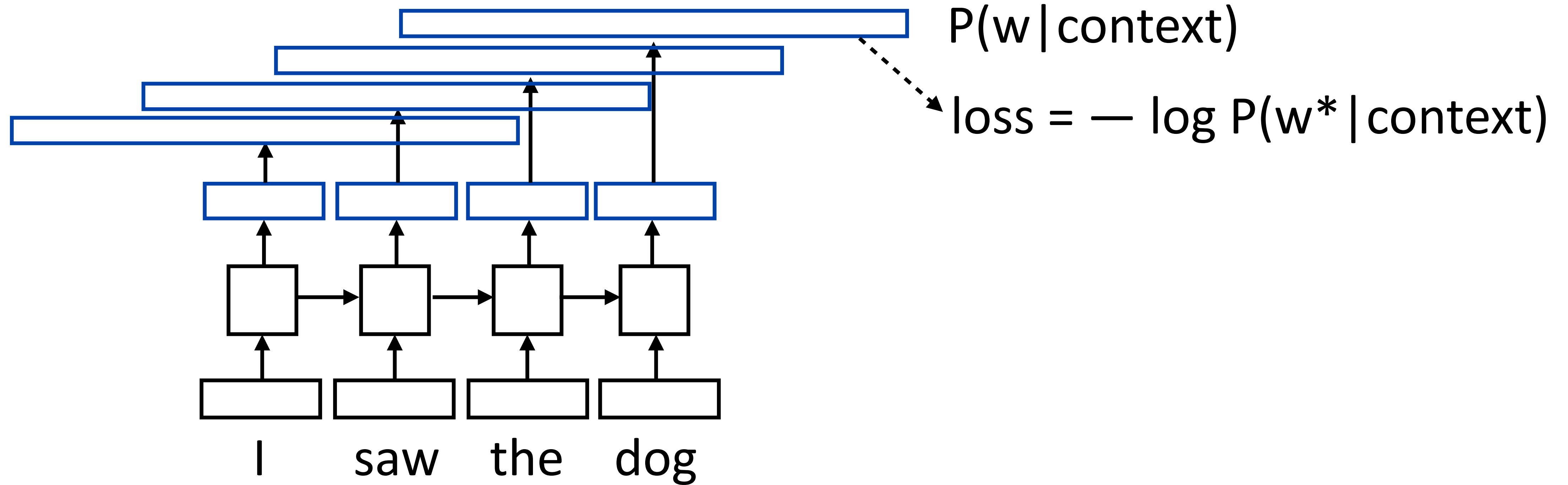
▸ Total loss = sum of negative log likelihoods at each position

▸ In PyTorch: simply add the losses together and call .backward()

Batched LM Training

I saw the dog running in the park and it looked very excited to be there

batch dim

looked very excited to be

in   the   park   and   it

I   saw   the   dog   running

<s>   in   the   park   and

<s>   I   saw   the   dog

▸ Multiple sequences **and** multiple timesteps per sequence

Batched LM Training

‣ torch.nn.LSTM / torch.nn.GRU: expect input in [seq len, batch, word dim] format, or in [batch, seq len, word dim] if you set batch_first = True

executed in parallel
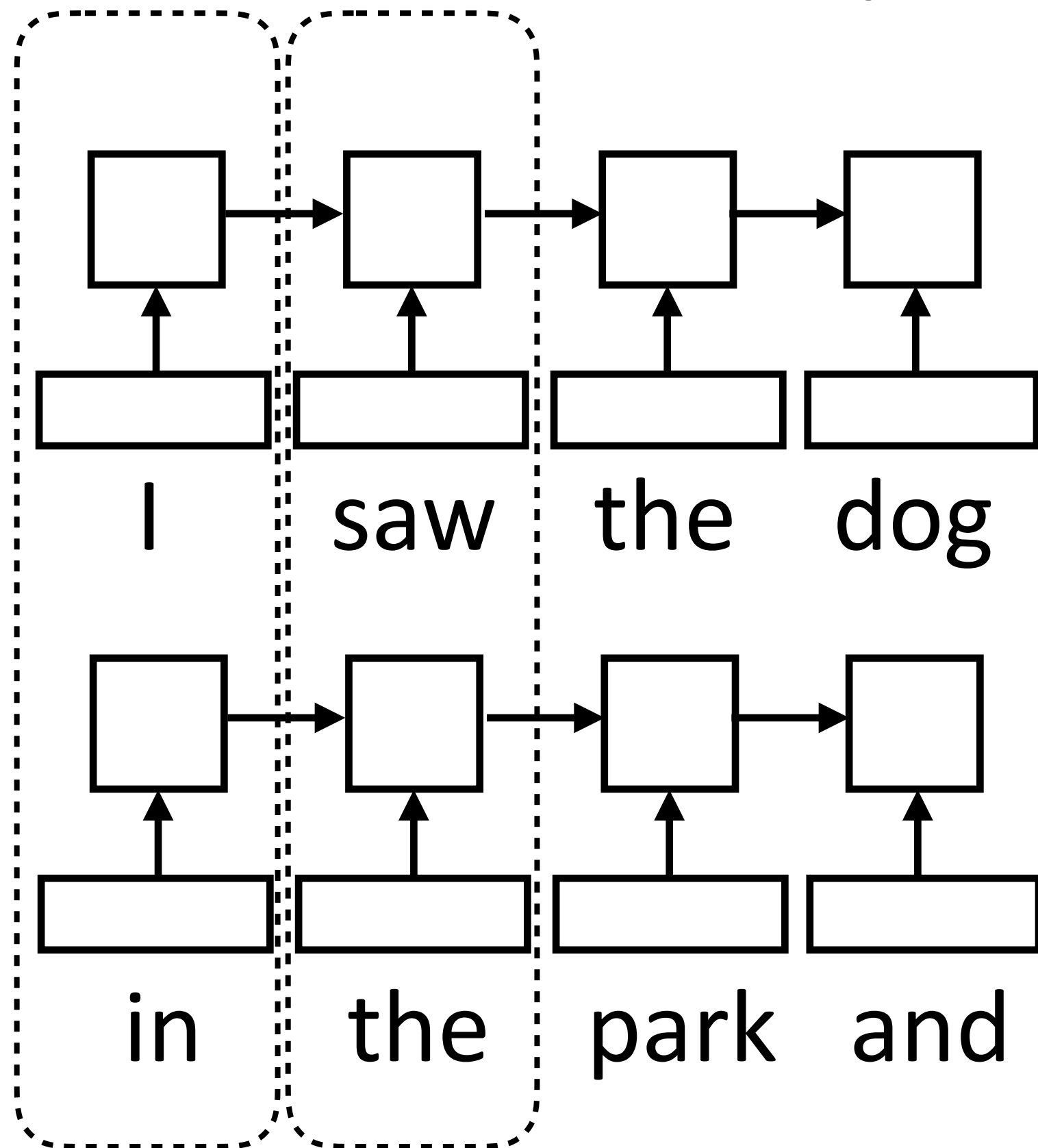
‣ Input: [4, 2, dim]

‣ Cannot parallelize across timesteps of RNN since output depends on previous timesteps, so using larger batches gives better parallelism

Other Implementation Details

‣ torch.nn.Embedding: maps sequence of word indices to vectors

  ‣ [126, 285] -> [[0.1, -0.07, 1.2],
                          [-2.3, 0.2, 1.4]]

  ‣ Moves from [sequence length] vector of indices -> [seq len, dim] tensor
    or [batch, sequence length] matrix -> [batch, seq len, dim tensor]

LM Evaluation

▶ Accuracy doesn't make sense — predicting the next word is generally impossible so accuracy values would be very low

▶ Evaluate LMs on the likelihood of held-out data (averaged to normalize for length)

$$\frac{1}{n} \sum_{i=1}^{n} \log P(w_i | w_1, \ldots, w_{i-1})$$

▶ Perplexity: exp(average negative log likelihood). Lower is better

   ▶ Suppose we have probs 1/4, 1/3, 1/4, 1/3 for 4 predictions

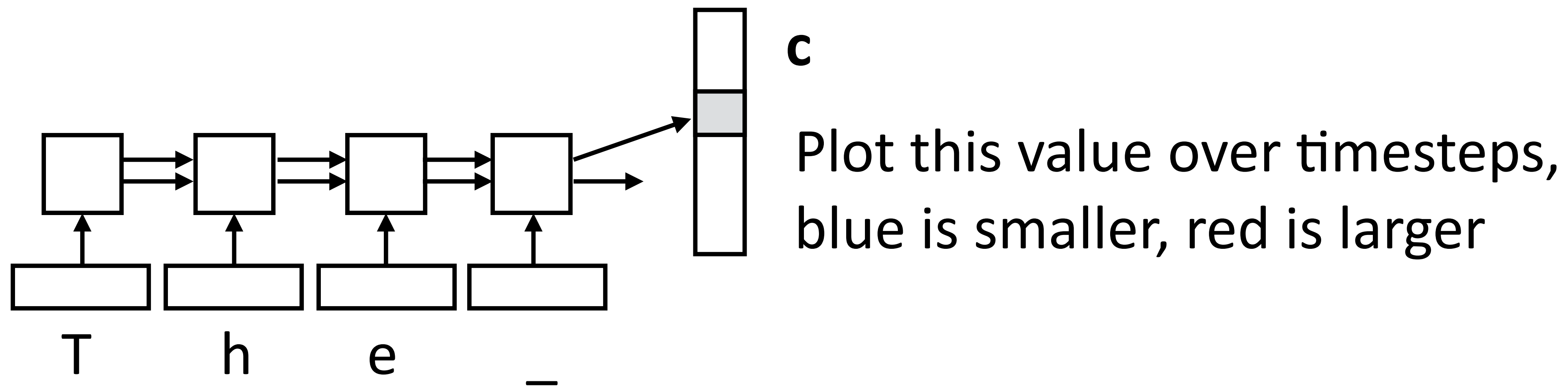   ▶ Avg NLL (base e) = 1.242   Perplexity = 3.464 <== geometric mean of denominators

Visualizing LSTM LMs

Visualizing LSTMs

▸ Train character LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

▸ Visualize activations of specific cells (components of **c**) to understand them

c

Plot this value over timesteps, blue is smaller, red is larger

T   h   e   _

Karpathy et al. (2015)

Visualizing LSTMs

▸ Train character LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

▸ Visualize activations of specific cells (components of **c**) to understand them

▸ Counter: know when to generate \n

Karpathy et al. (2015)

Visualizing LSTMs

▸ Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

▸ Visualize activations of specific cells to see what they track

▸ Binary switch: tells us if we're in a quote or not

Karpathy et al. (2015)

Visualizing LSTMs

▸ Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

▸ Visualize activations of specific cells to see what they track

▸ Stack: activation based on indentation

Karpathy et al. (2015)

Visualizing LSTMs

▸ Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

▸ Visualize activations of specific cells to see what they track

▸ Uninterpretable: probably doing double-duty, or only makes sense in the context of another activation

Karpathy et al. (2015)

State-of-the-art LMs

▶ Good LSTM LMs have ~27M params, 4-5 layers

▶ Kneser-Ney 5-gram model with cache: PPL = 125.7

▶ LSTM: PPL ~ 60-80 (depending on how much you optimize it)

▶ LSTM character-level: PPL ~1.5 (205 character vocab)

▶ Better language models using transformers (will discuss after MT)

Melis et al. (2017)