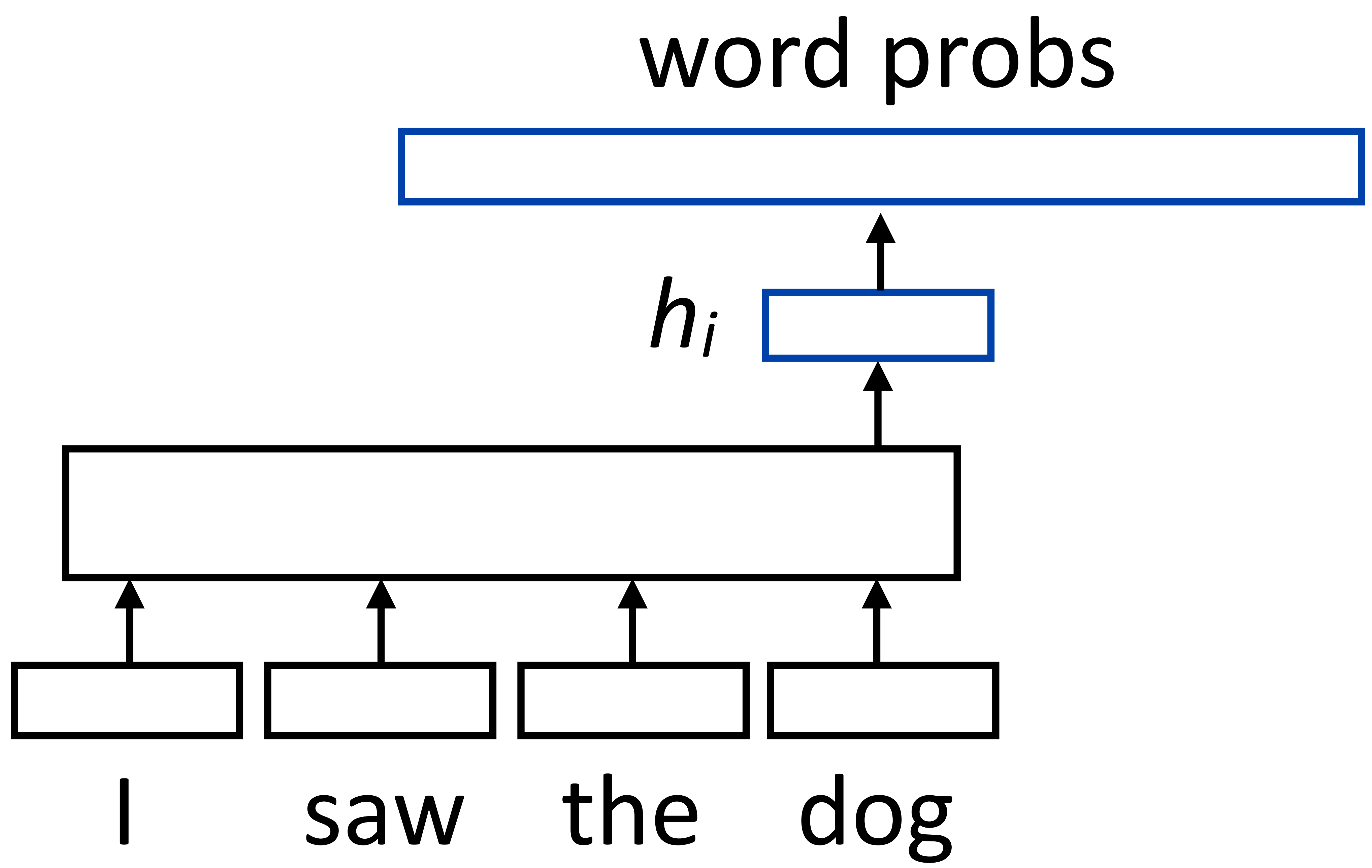
Transformer Language Modeling



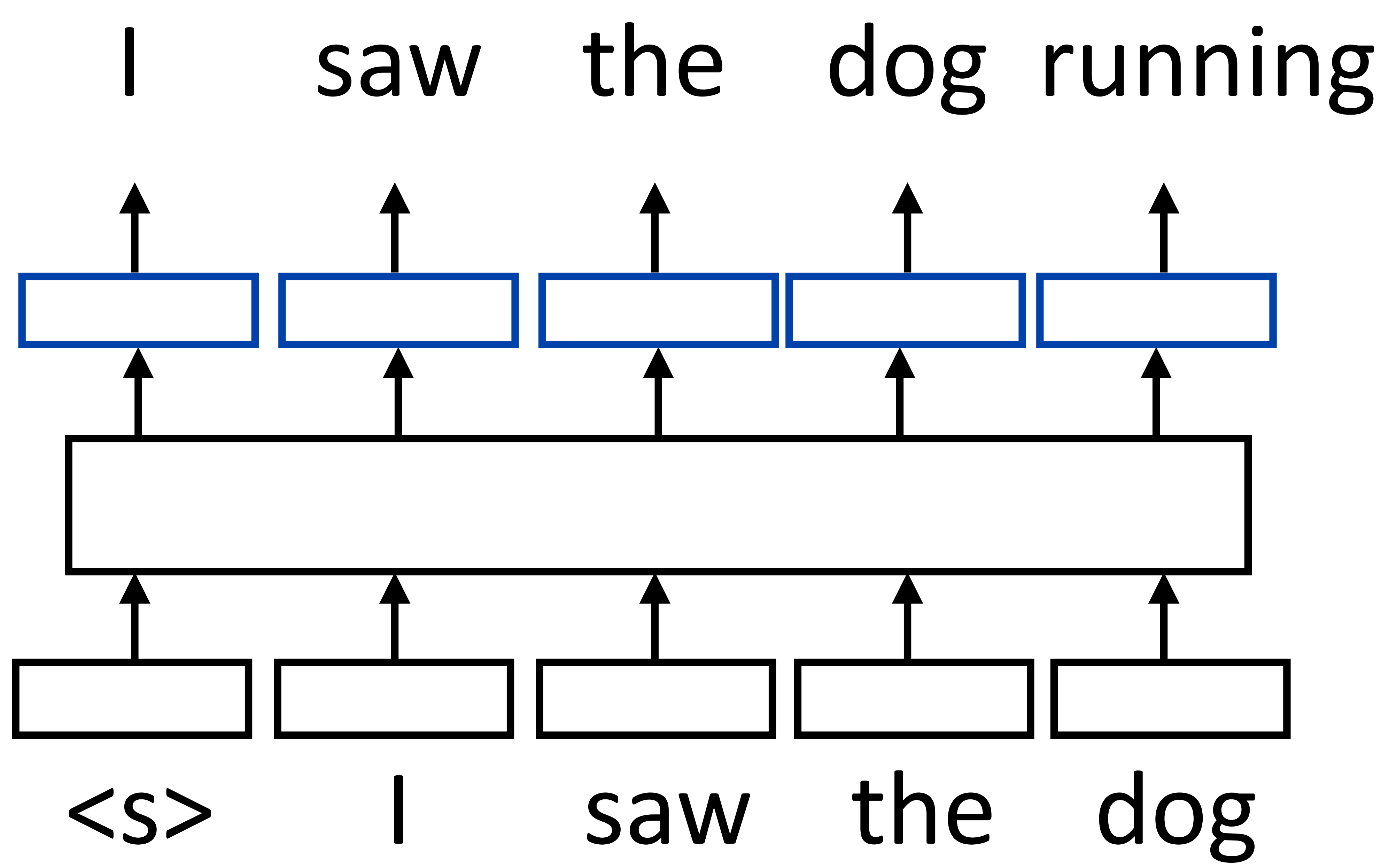
$$P(w|\text{context}) = \frac{\exp(\mathbf{w} \cdot \mathbf{h_i})}{\sum_{w'} \exp(\mathbf{w'} \cdot \mathbf{h_i})}$$

equivalent to

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

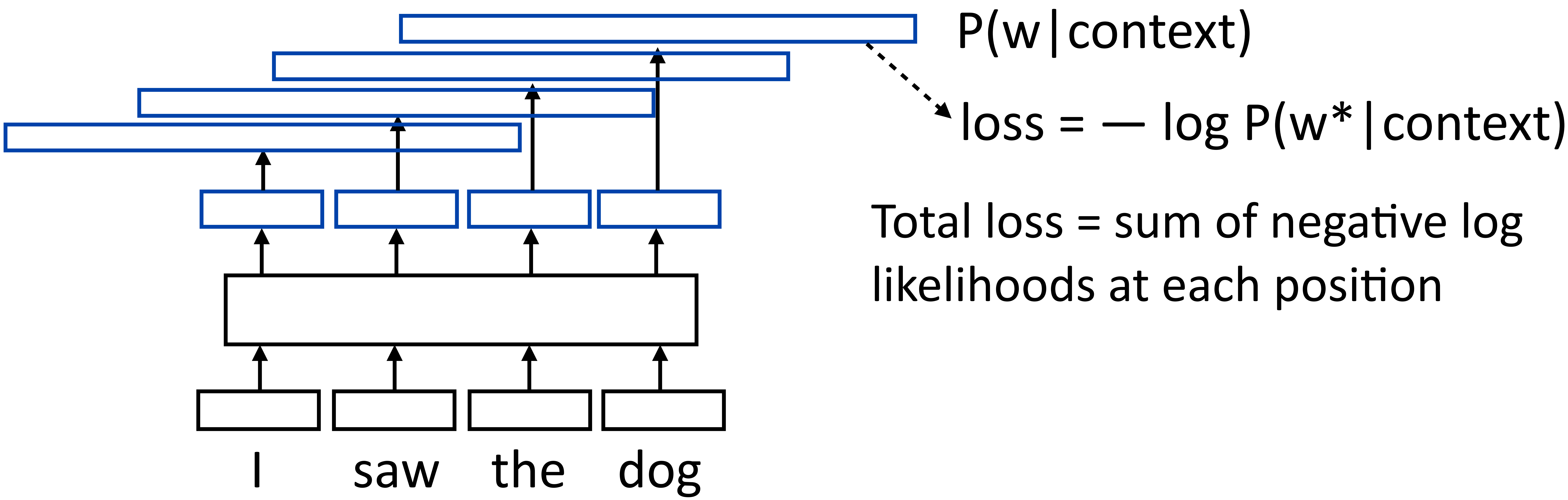
- h_i is the embedding of *dog* produced by the Transformer
- W is a (vocab size) x (hidden size) matrix; linear layer in PyTorch (rows are word embeddings)

Training Transformer LMs



- 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 Transformer LMs



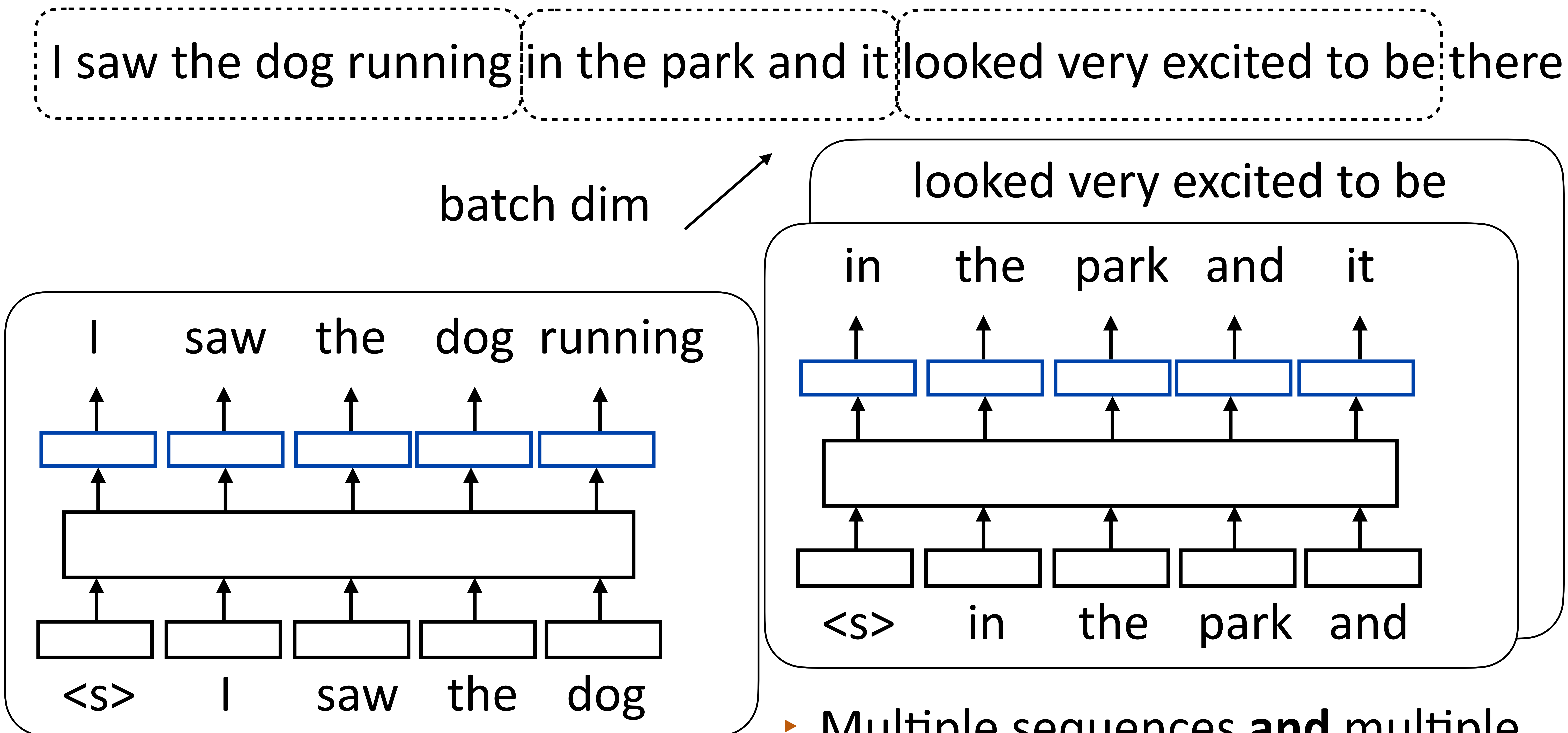
Total loss = sum of negative log likelihoods at each position

```
loss_fcn = nn.NLLLoss()
loss += loss_fcn(log_probs, ex.output_tensor)
```

[seq len, num output classes] [seq len]

- Batching is a little tricky with NLLLoss: need to collase [batch, seq len, num classes] to [batch * seq len, num classes]. You do not need to batch

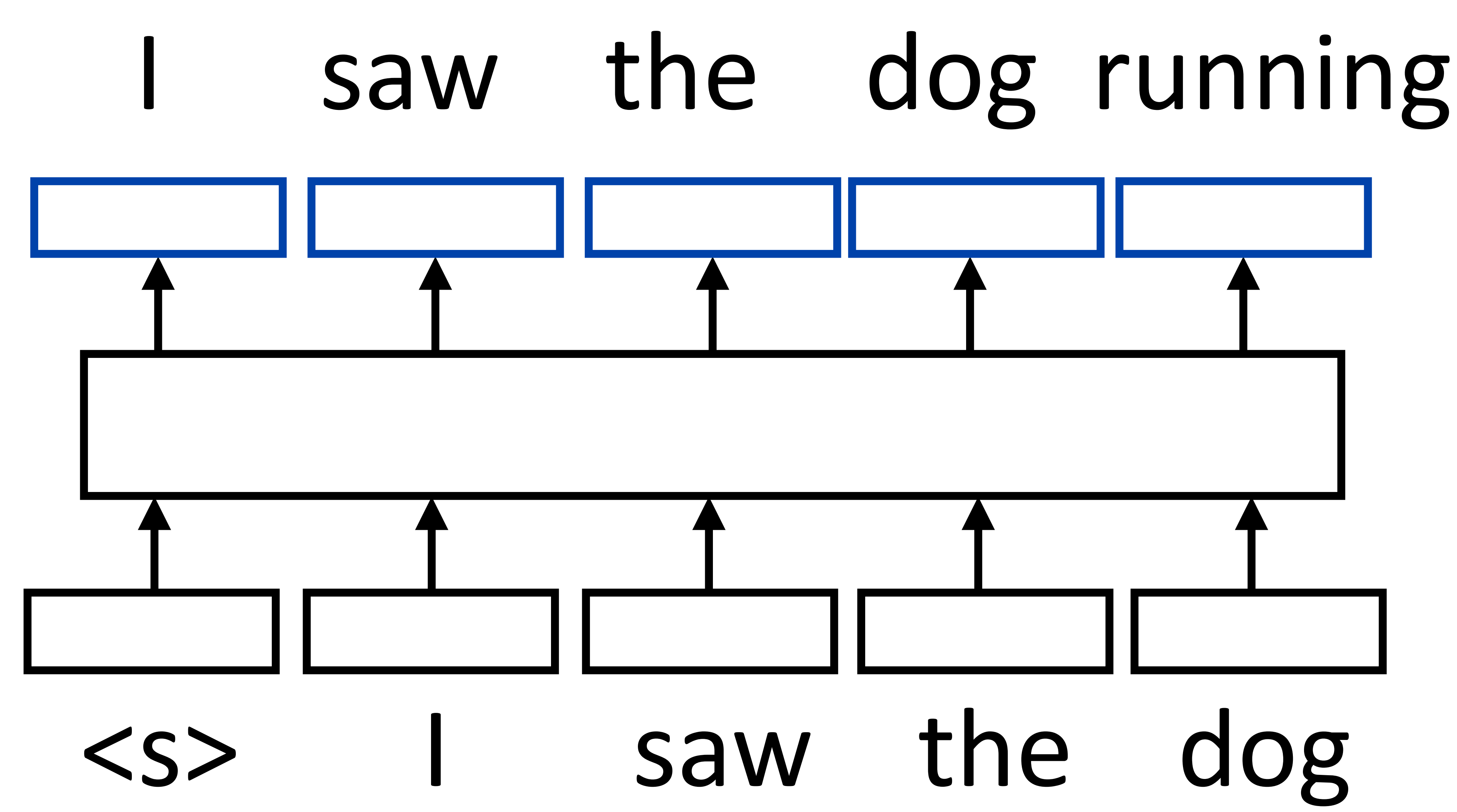
Batched LM Training



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

A Small Problem with Transformer LMs

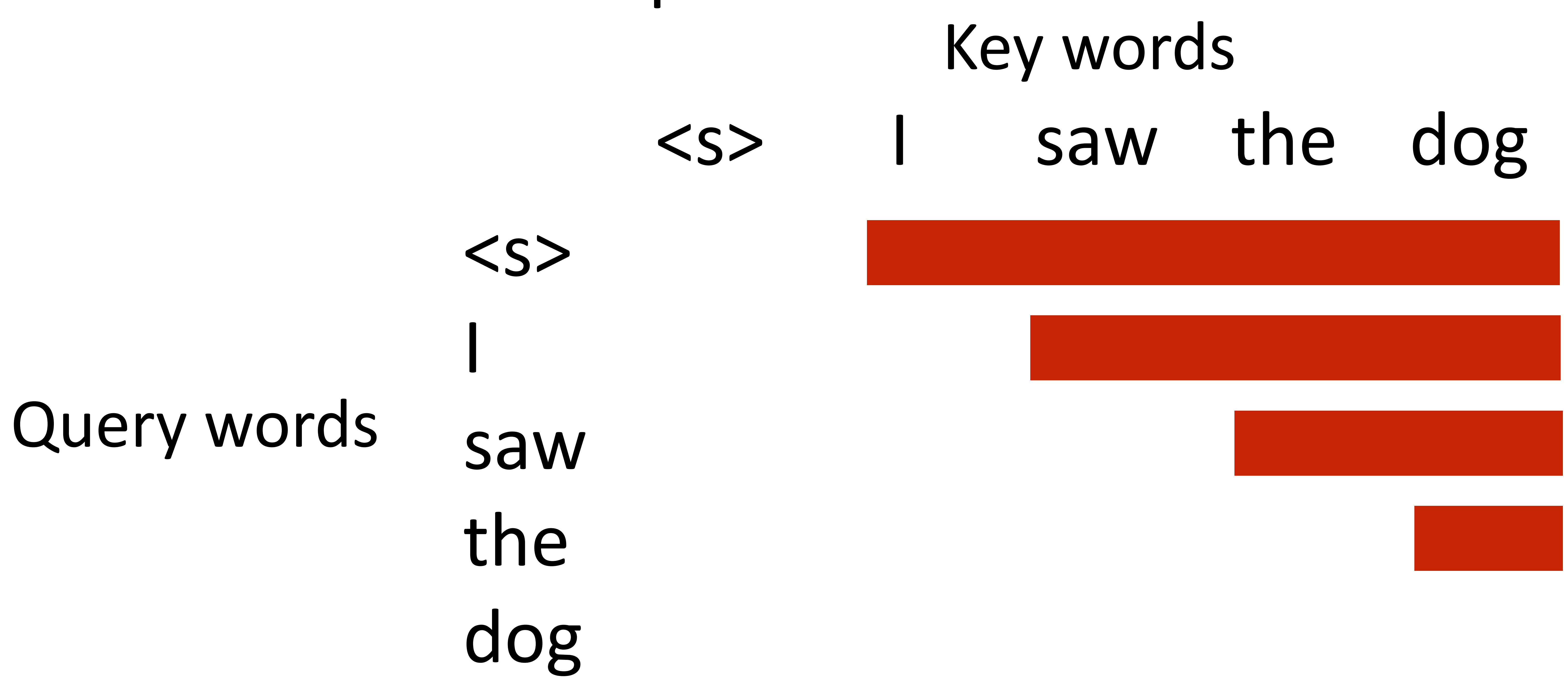
- This Transformer LM as we've described it will *easily* achieve perfect accuracy. Why?



- With standard self-attention: "I" attends to "saw" and the model is "cheating". How do we ensure that this doesn't happen?

Attention Masking

- What do we want to prohibit?



- This is called a **causal** mask (also, causal self-attention / causal Transformers). Only things in the "past" can influence the "present"

Implementation in PyTorch

- nn.TransformerEncoder can be built out of nn.TransformerEncoderLayers, can accept an input and a mask for language modeling:

```
# Inside the module; need to fill in size parameters
layers = nn.TransformerEncoderLayer([...])
transformer_encoder = nn.TransformerEncoder(encoder_layers, num_layers=[...])
[. . .]
# Inside forward(): puts negative infinities in the red part
mask = torch.triu(torch.ones(len, len) * float('-inf'), diagonal=1)
output = transformer_encoder(input, mask=mask)
```