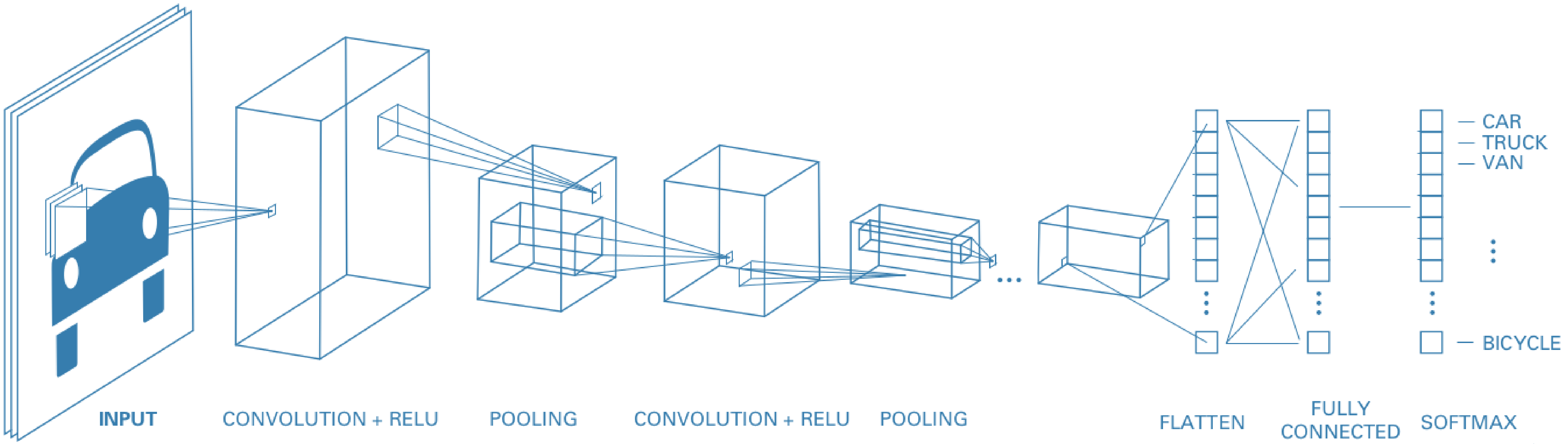
Deep Learning for Language: Recurrent Neural Networks, Attention

Robin Jia
USC CSCI 467, Spring 2023
February 28, 2023

Outline

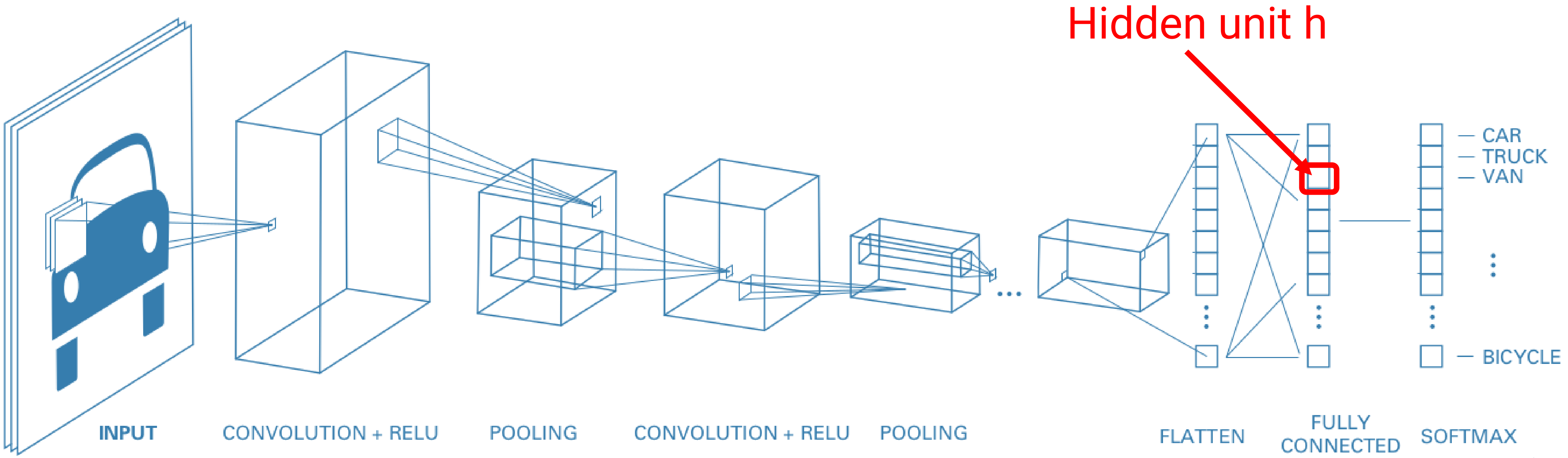
- Loose ends
  - How does backprop learn features?
  - Visualizing CNN features
- Recurrent Neural Networks for sequential data
- Sequence-to-sequence and Attention

How does backprop learn features?



- Every convolution & fully connected layer has (many) parameters
  - Convolutional: Kernel with #outChannels x (#inChannels x K x K + 1) params
  - Fully connected: #outDimensions x (#inDimensions + 1) params
- These all have to get learned by backprop + gradient descent on the loss

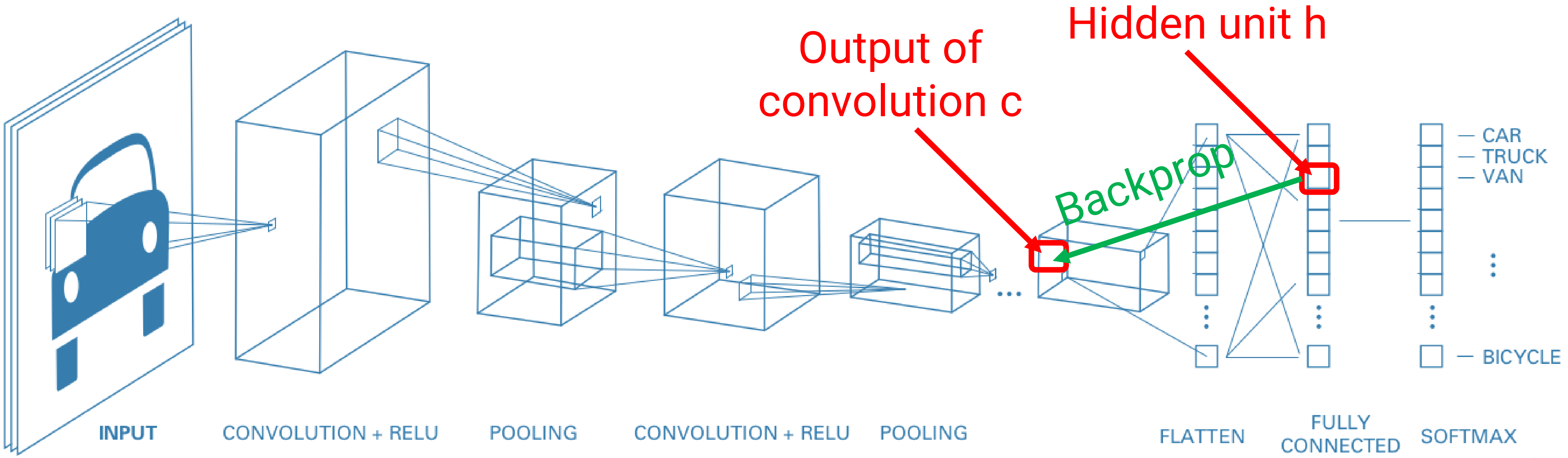
How does backprop learn features?



- Training example $(x^{(1)}, y^{(1)})$: $\partial(\text{Loss})/\partial(h) > 0$
  - Means that making h **smaller** leads to lower loss
- Training example $(x^{(2)}, y^{(2)})$: $\partial(\text{Loss})/\partial(h) < 0$
  - Means that making h **larger** leads to lower loss

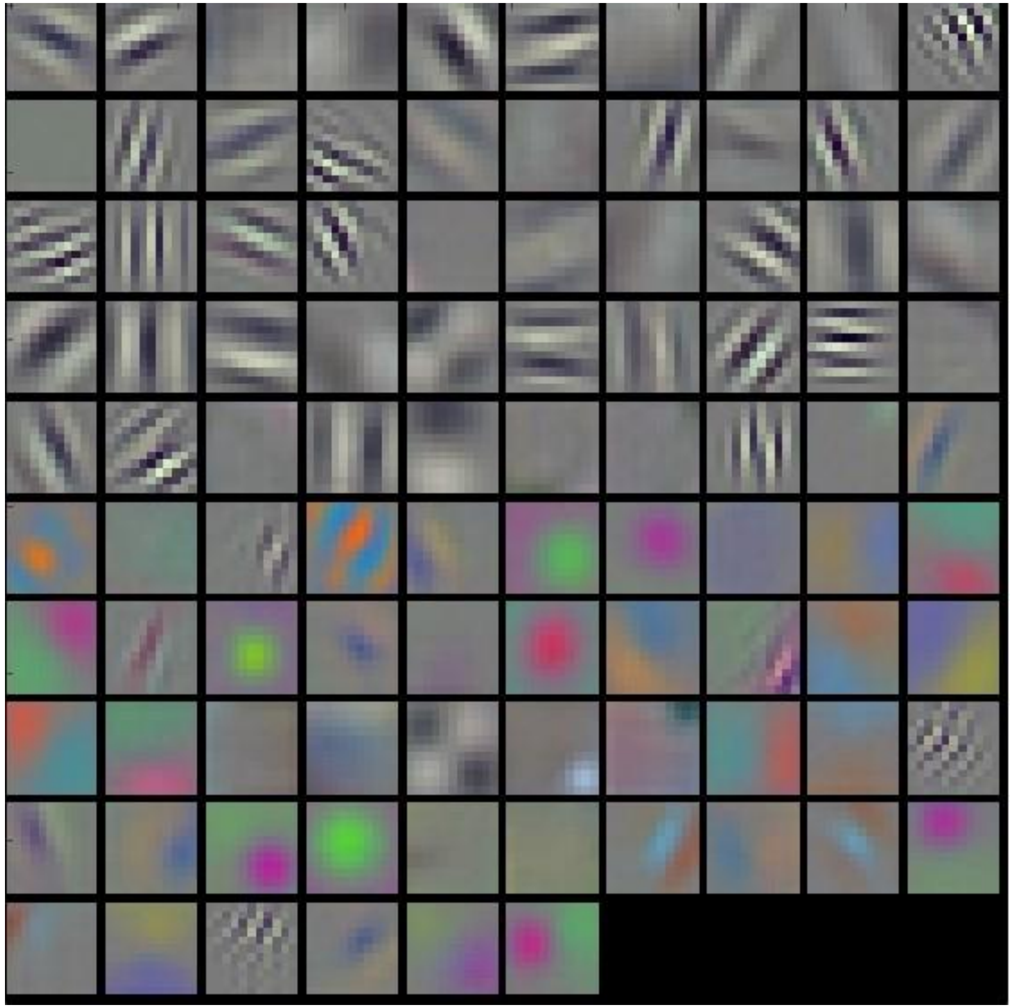
- h is output of "classifier"
- Gradient tunes classifier parameters to make output larger on some examples, smaller on others

How does backprop learn features?



- Backpropagation: Does making c bigger change h in good or bad way?
- Sum up these considerations over all hidden units that depend on c
- Train convolutional kernel parameters so that value of c leads to [values of h's that lead to good outputs]
- And so on for earlier layers…

What features do CNNs learn?



- Kernels of AlexNet first layer
  - Each one is 3 (for RGB) x 11 x 11
- What is learned?
  - Edge detectors in different directions and widths
  - Patches of various colors

What features do CNNs learn?

Faces

Dogs (eyes?)

Red ornaments/
flowers

Text (years?)

Houses

Lens flare?

Each Row: Images that activate a different neuron in 5th POOL layer of AlexNet

Outline

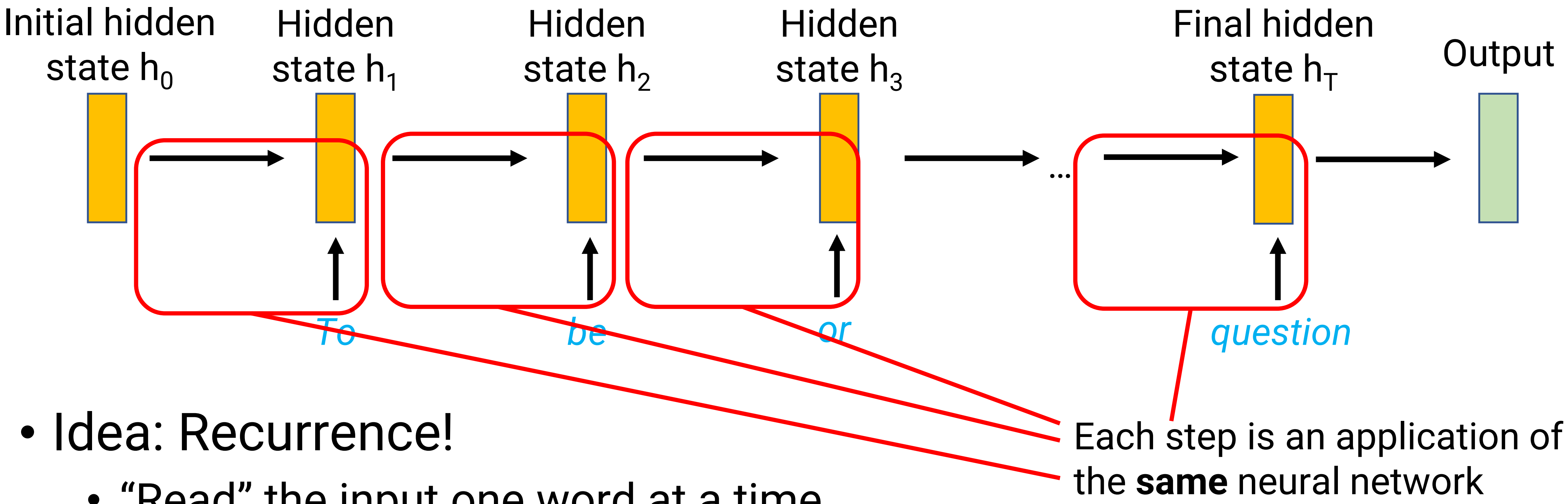
- Loose ends
  - How does backprop learn features?
  - Visualizing CNN features
- Recurrent Neural Networks for sequential data
- Sequence-to-sequence and Attention

Note: Often there are many similar ways to achieve similar results
No one way of modeling is "correct"
I want you to remember the modeling ideas/concepts

Handling textual data

- Images: We assume inputs are fixed dimensional
  - Can crop/rescale as needed
- Text: Inputs are naturally variable-sized!
  - Example 1: *Amazing!*
  - Example 2: *There are many issues with this movie, such as…*
- Challenge: How can we use the **same** set of model parameters to handle inputs of any size?

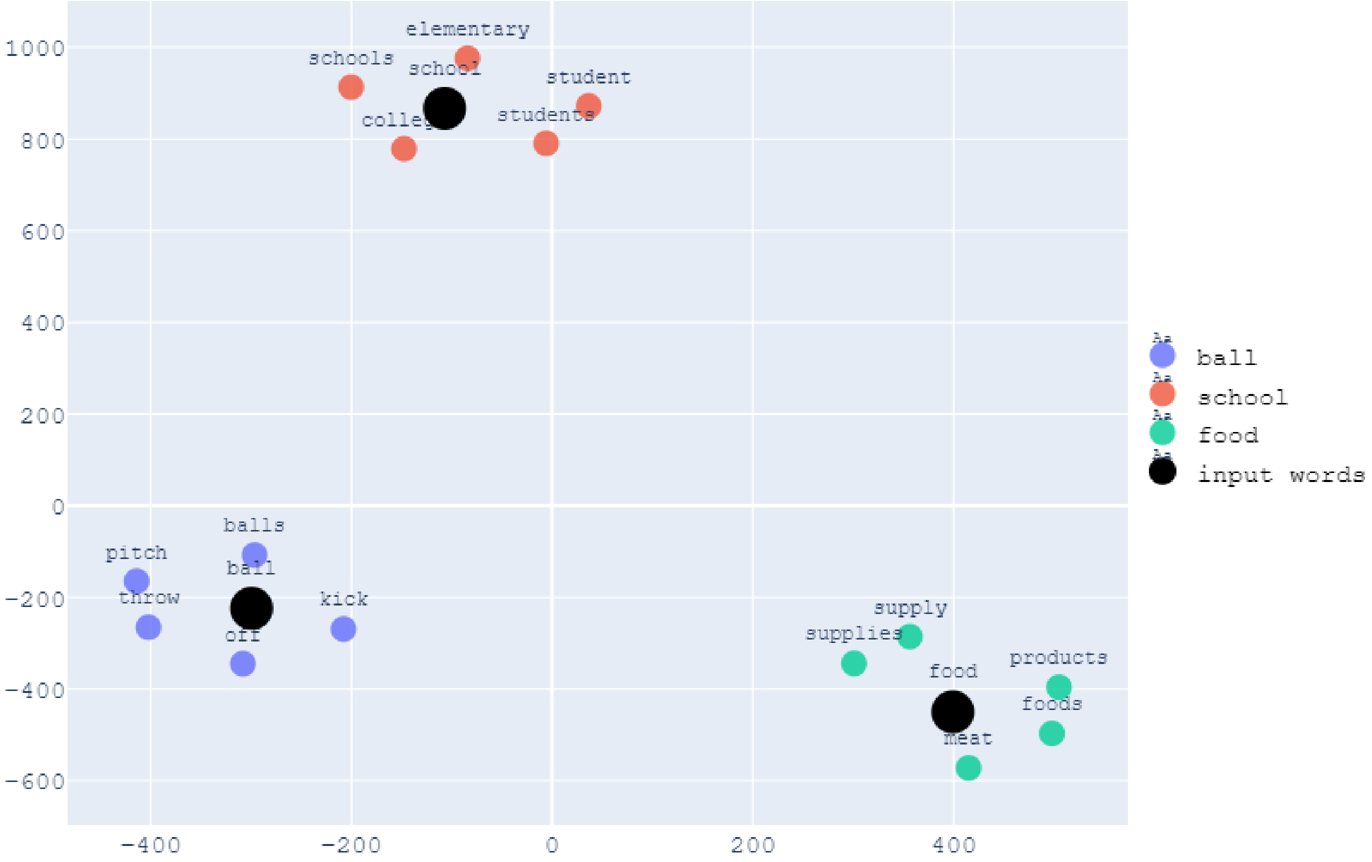
Recurrent Neural Networks (RNNs)



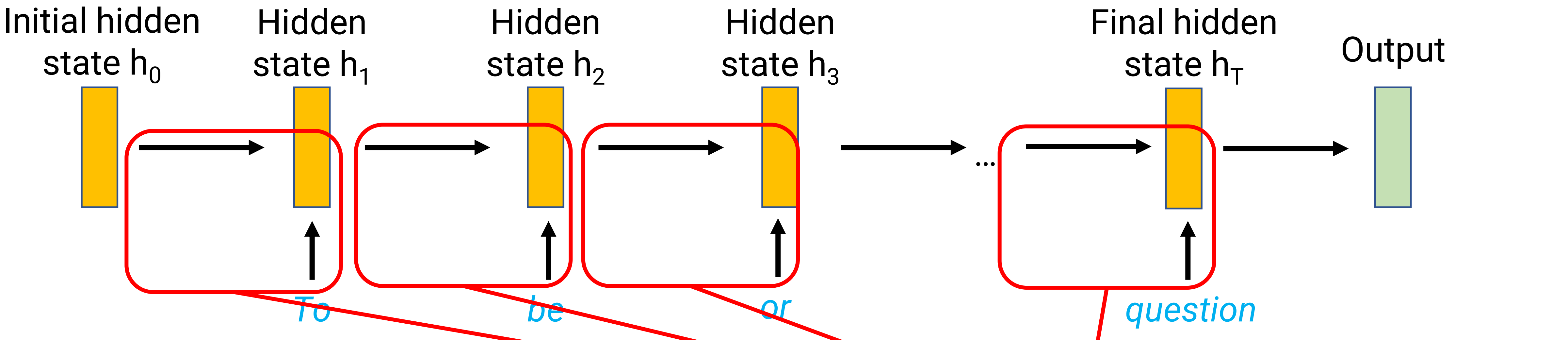
- Idea: Recurrence!
  - "Read" the input one word at a time
  - At each step, update the hidden state of the network
  - **Model parameters to do this update are same for each step**

Word Embeddings

- How do we “feed” the next word to the RNN?
- Want to learn a vector that represents each word
  - For each word *w* in vocabulary *V*, have vector v_w of size *d*
  - |V| * d parameters needed
- Intuition: Similar words get similar vectors
  - More on learning word vectors later in the class!



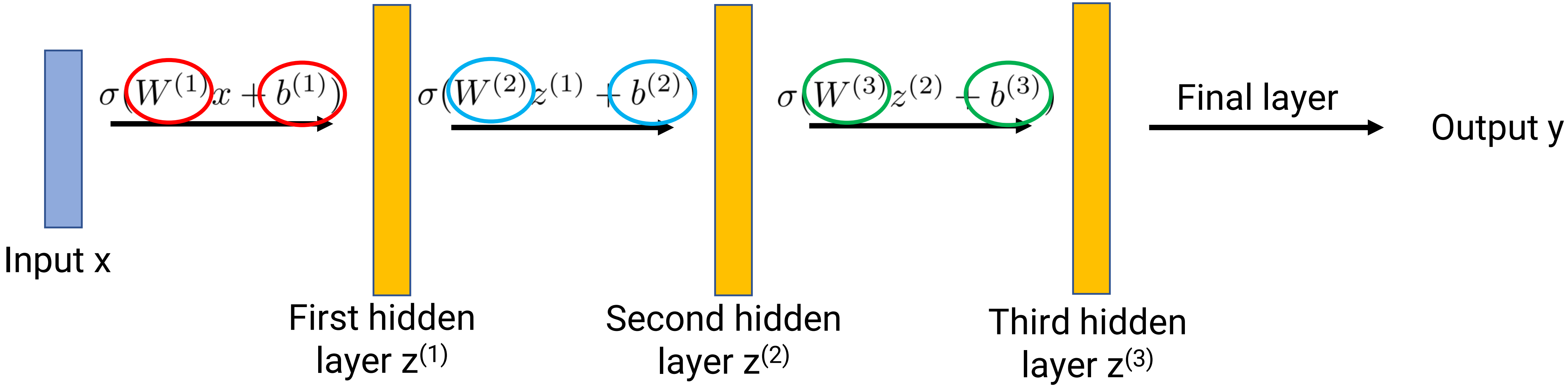
One RNN variant



$$h_t = \sigma\left(\underbrace{W_h h_{t-1}}_{\text{Linear function of prev. hidden state}} + \underbrace{W_x x_t}_{\text{Linear function of current word vector}} + b\right)$$

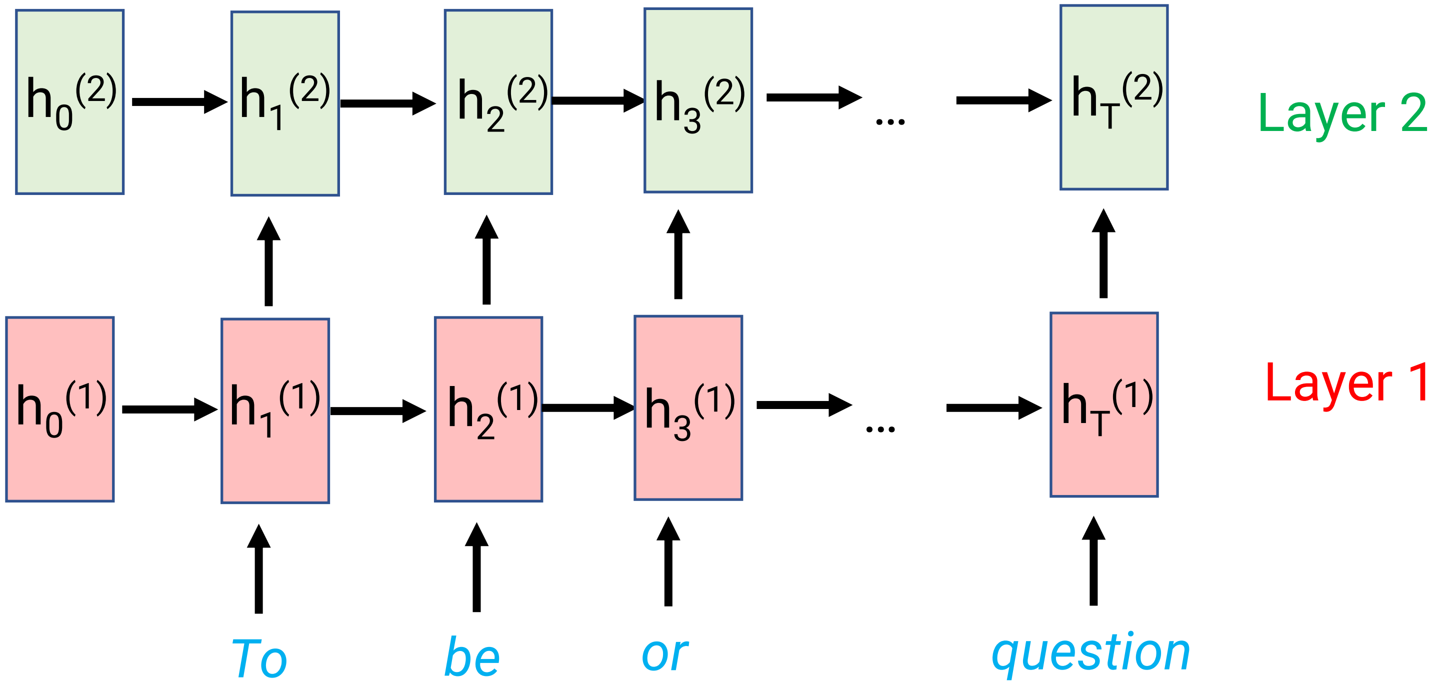
- At each timestep t, run a neural network that takes in 2 inputs (or 1 big input, by concatenation)
  - Previous hidden state h_{t-1}
  - Vector for current word x_t
- Learn linear function of both inputs, add bias, apply non-linearity
- Parameters: Recurrence params (W_h, W_x, b), initial hidden state h_0, word vectors

Recurrence vs. Depth



- Deep networks (i.e., adding more layers)
  - Computation graph becomes longer
  - Number of parameters also grows; each step uses new parameters
- Recurrent neural networks
  - Computation graph becomes longer
  - Number of parameters **fixed**; each step uses **same parameters**

Recurrence and Depth



- You can have multiple layers of recurrence too!
  - Left-to-right axis ("time dimension"): Length is size of input, same parameters in each step
  - Top-to-bottom axis ("depth dimension"): Length is depth of network, different parameters in each row

Announcements

- HW2 due this Thursday
  - Pytorch not reproducible across different hardware
  - Still used in assignment as it is very widely used for deep learning
  - Ultimately we will grade by reading your code, not by checking if your numbers in the write-up are “correct”
- Proposals should be returned with feedback by Thursday
- Tuesday, March 7: Discussion of Midterm Report due March 23
- Section canceled March 10
  - We will stop doing HW review sections, as they seem less popular
  - Please still come to OH if you want clarifications on old HW problems

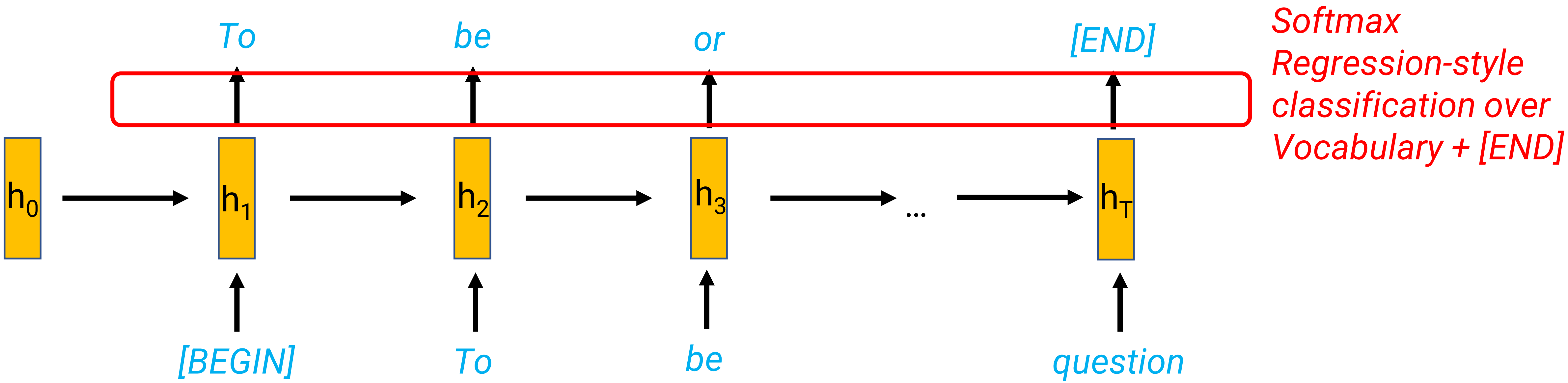
Outline

- Loose ends
  - How does backprop learn features?
  - Visualizing CNN features (cat neuron?)
- Recurrent Neural Networks for sequential data
- Sequence-to-sequence and Attention

How to use RNNs?

- Language modeling/text generation (“Decoder only”)
- Text classification (“Encoder only”)
- Sequence-to-sequence (“Encoder-decoder”)

Language Modeling (“Decoder only”)



- At each step, predict the next word given current hidden state
  - Essentially a softmax regression “head”—takes in hidden state, outputs distribution over Vocabulary + [END]
- Start with special *[BEGIN]* token (so the first word model generates is first real word)
- One step’s output becomes next step’s input (“autoregressive”)
- To mark end of sequence, model should predict the *[END]* token
- Called a “Decoder” because it looks at the hidden state and “decodes” the next word

Long-Range Dependencies

- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model **long-range dependencies**

The keys to the cabinet ___ (on the table)

plural singular

Long-Range Dependencies

- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model **long-range dependencies**

*The keys to the cabinet **are** (on the table)*

plural singular

Long-Range Dependencies

- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model **long-range dependencies**

The keys to the cabinet by the door ***are*** *(on the table)*

Long-Range Dependencies

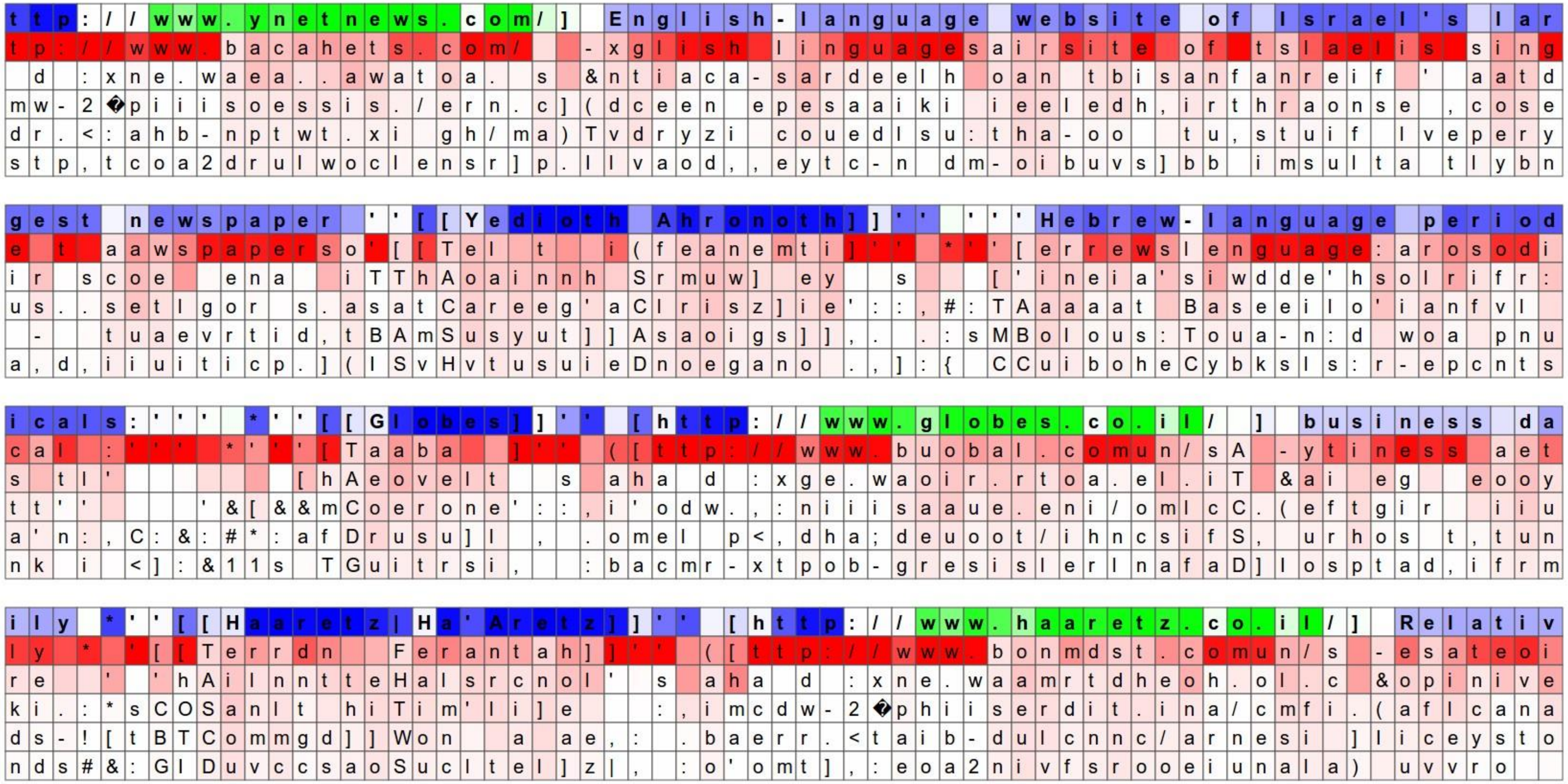
- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model **long-range dependencies**

*The keys to the cabinet by the door on the left **are** (on the table)*

Advanced RNNs

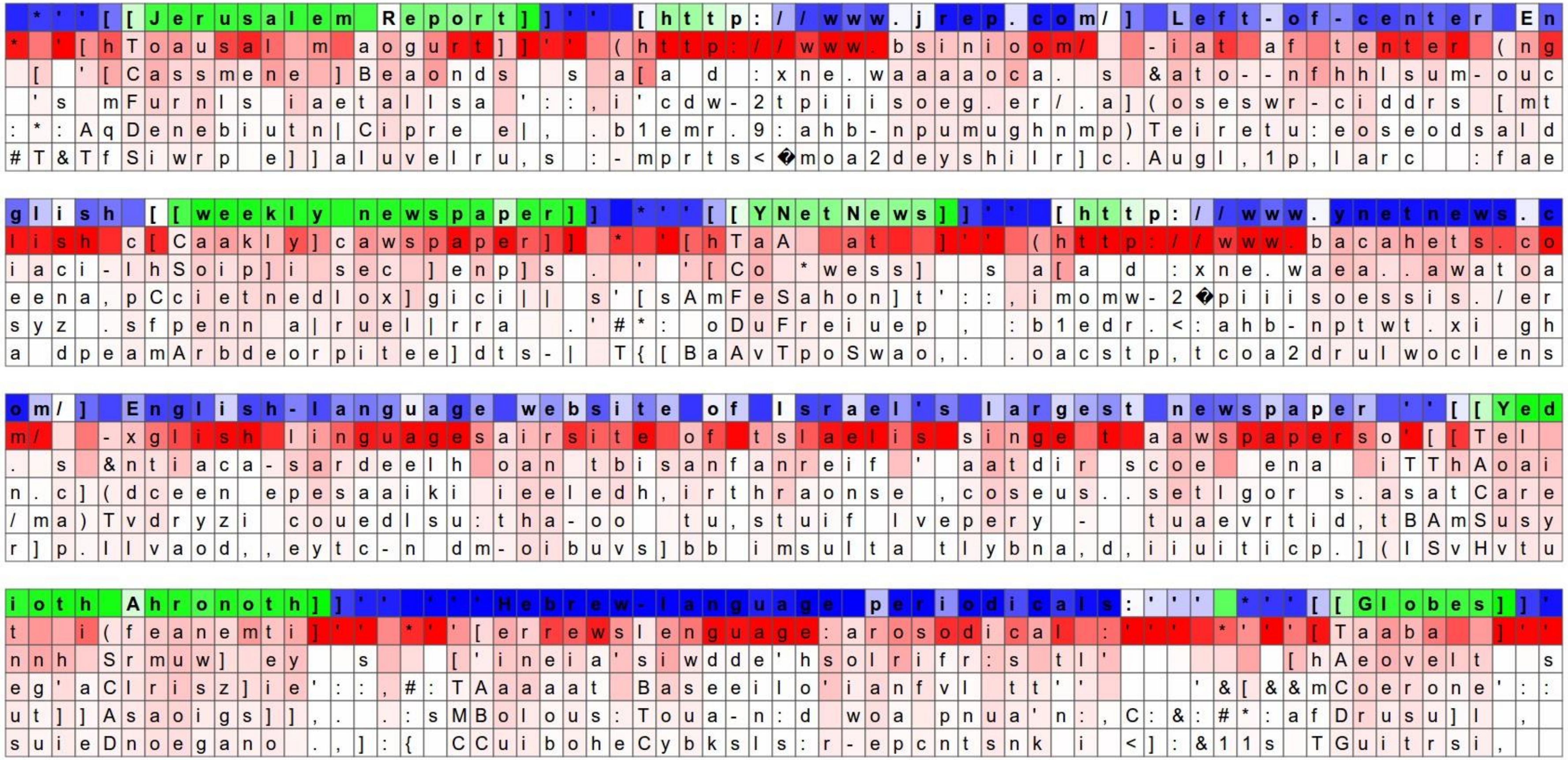
- “Gated” RNNs (GRUs, LSTMs)
    - Better at holding on to long-range state
    - These are usually preferable to the RNN variant I showed today
    - They work the same way, but the recurrence relationship between previous hidden state and next hidden state is more complicated…

What do RNNs learn?



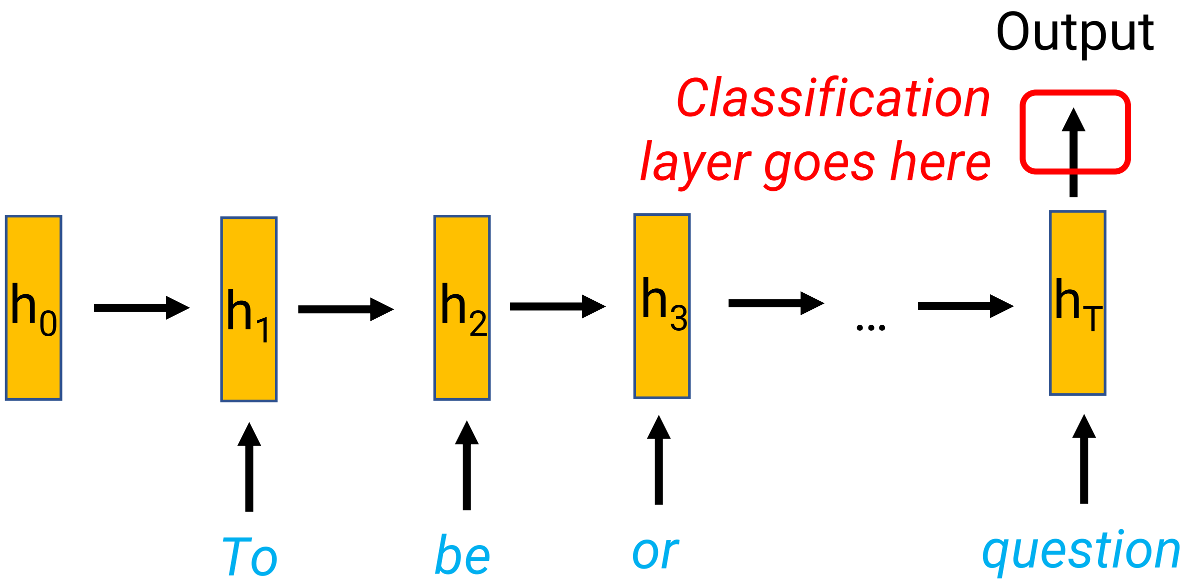
- Here: a character-level model (not word-level)
- Blue/Green: Low/high values of 1 neuron
- Below: Top-5 predictions for next character
- This neuron seems to detect whether we're inside a URL

What do RNNs learn?



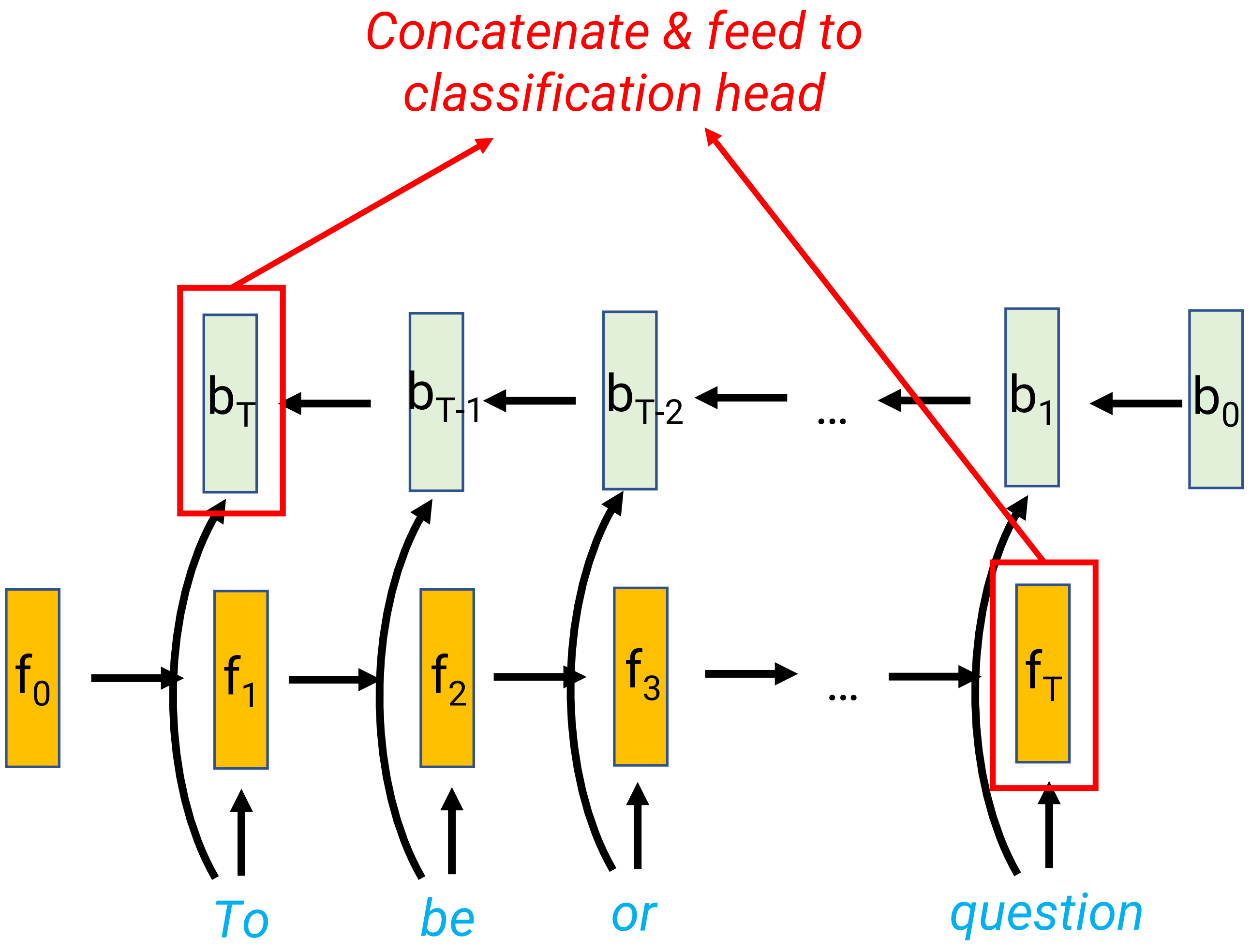
- Here: a character-level model (not word-level)
- Blue/Green: Low/high values of 1 neuron
- Below: Top-5 predictions for next character
- This neuron fires only inside a Markdown [[link]] (so it knows when to close the square brackets?)

Text classification ("Encoder only")



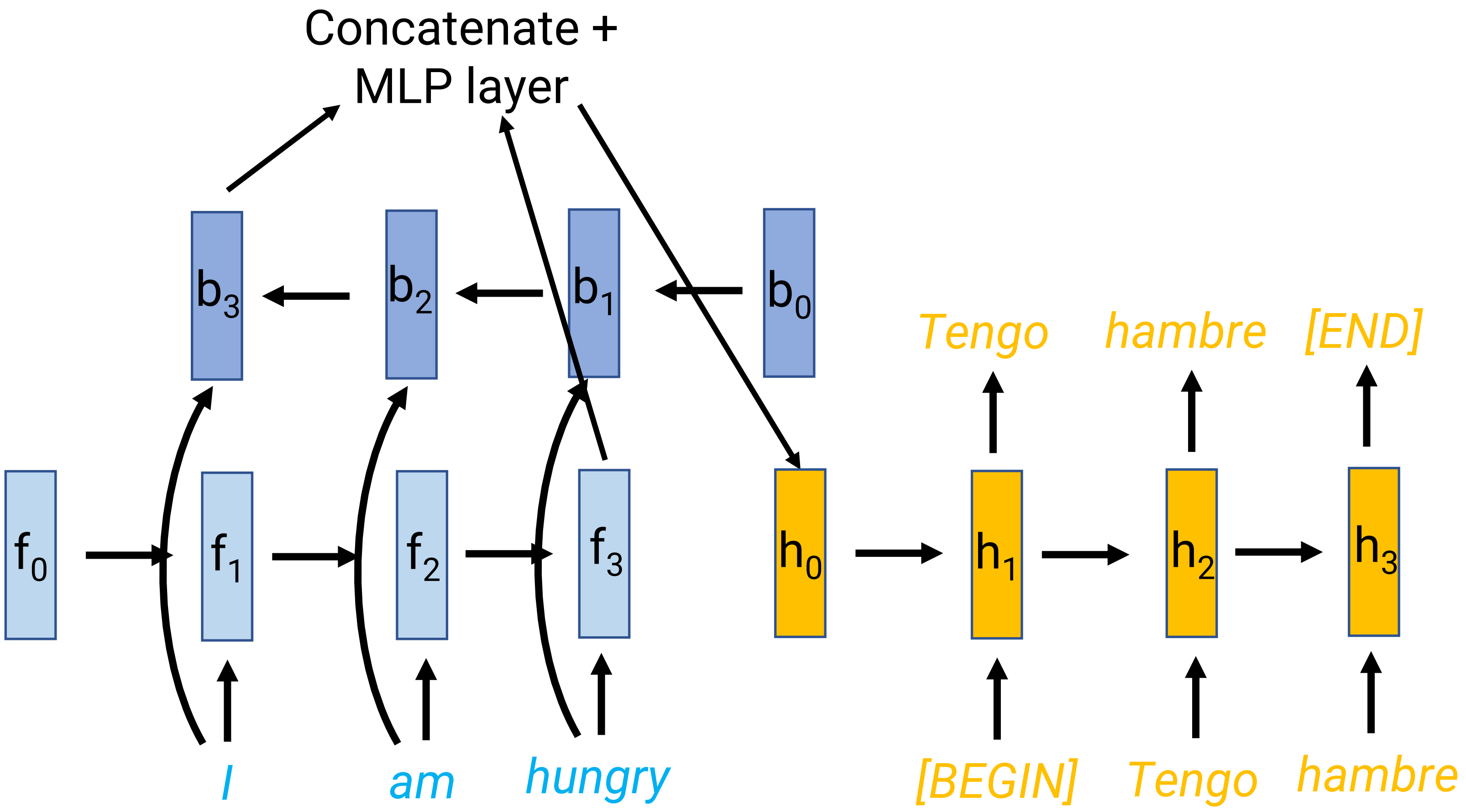
- First run an RNN over text
- Use the final hidden state as an "encoding" of the entire sequence
- Use this as features, train a classifier on top
- Downside: Later words processed better than early words (long range dependency issues)

Bi-directional encoders



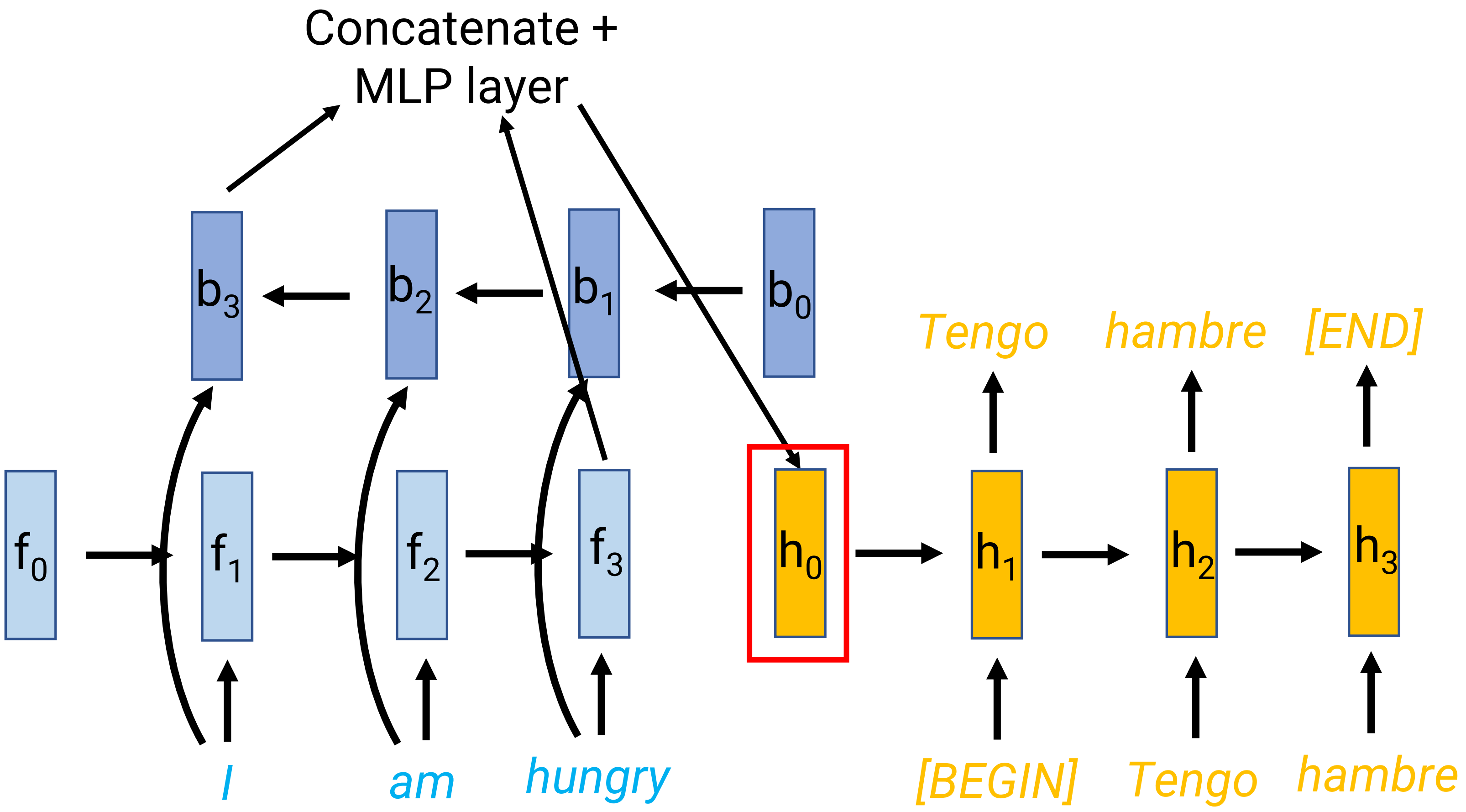
- Run one RNN left-to-right, and another one right-to-left
  - (I'll call forward-direction hidden states f_t, backward-direction hidden states b_t)
- Concatenate the 2 final hidden states as final representation
  - Note: This encoding is twice as large now—we've doubled the number of features passed to the final classifier

Sequence-to-sequence (“Encoder-decoder”)



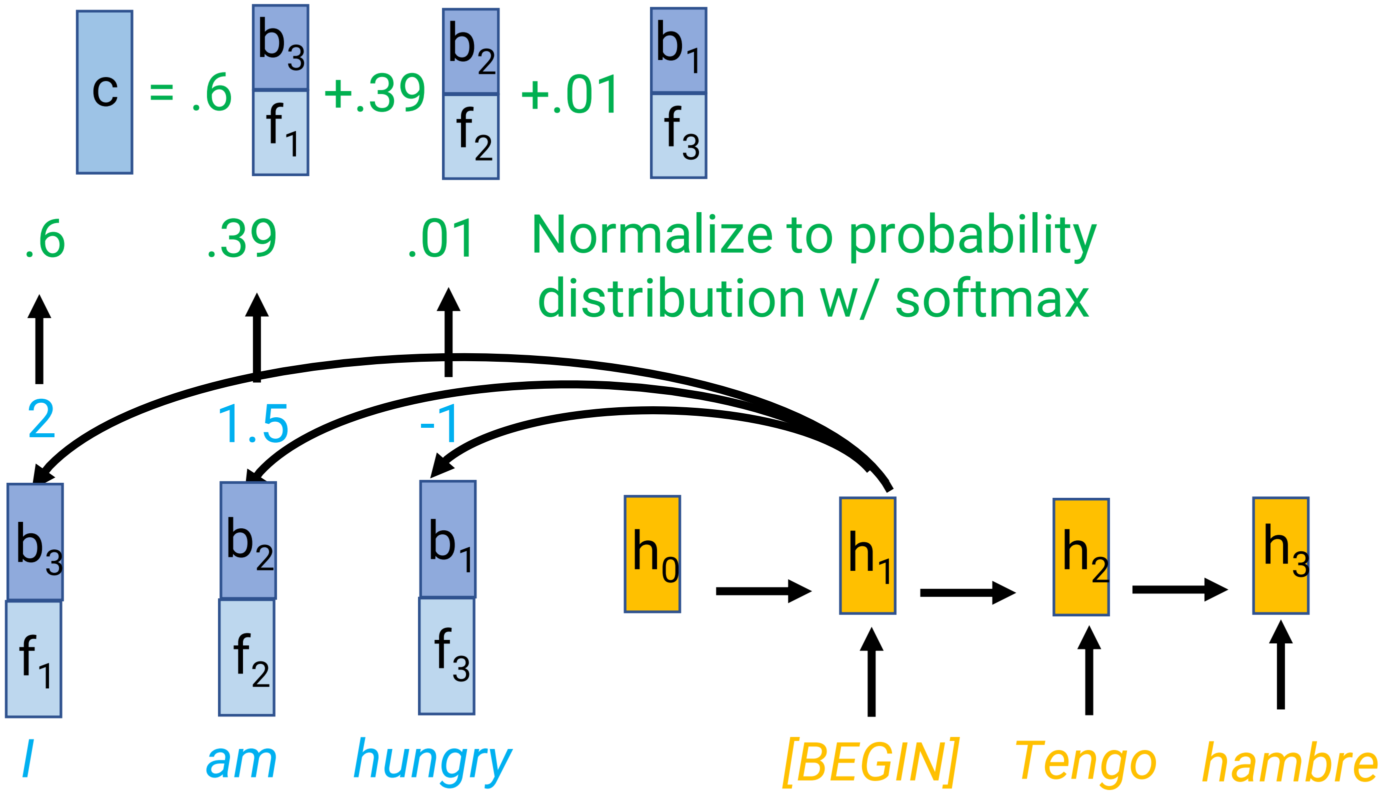
- Example: Machine Translation
  - Input = English text
  - Output = Spanish text
- Encoder: Process English sentence into vector
  - E.g. Bidirectional encoder + MLP layer to generate decoder’s initial state
- Decoder: Use vector as initial hidden state and start doing language modeling in Spanish
- Vector space acts as a “shared language”

What's missing? Alignment



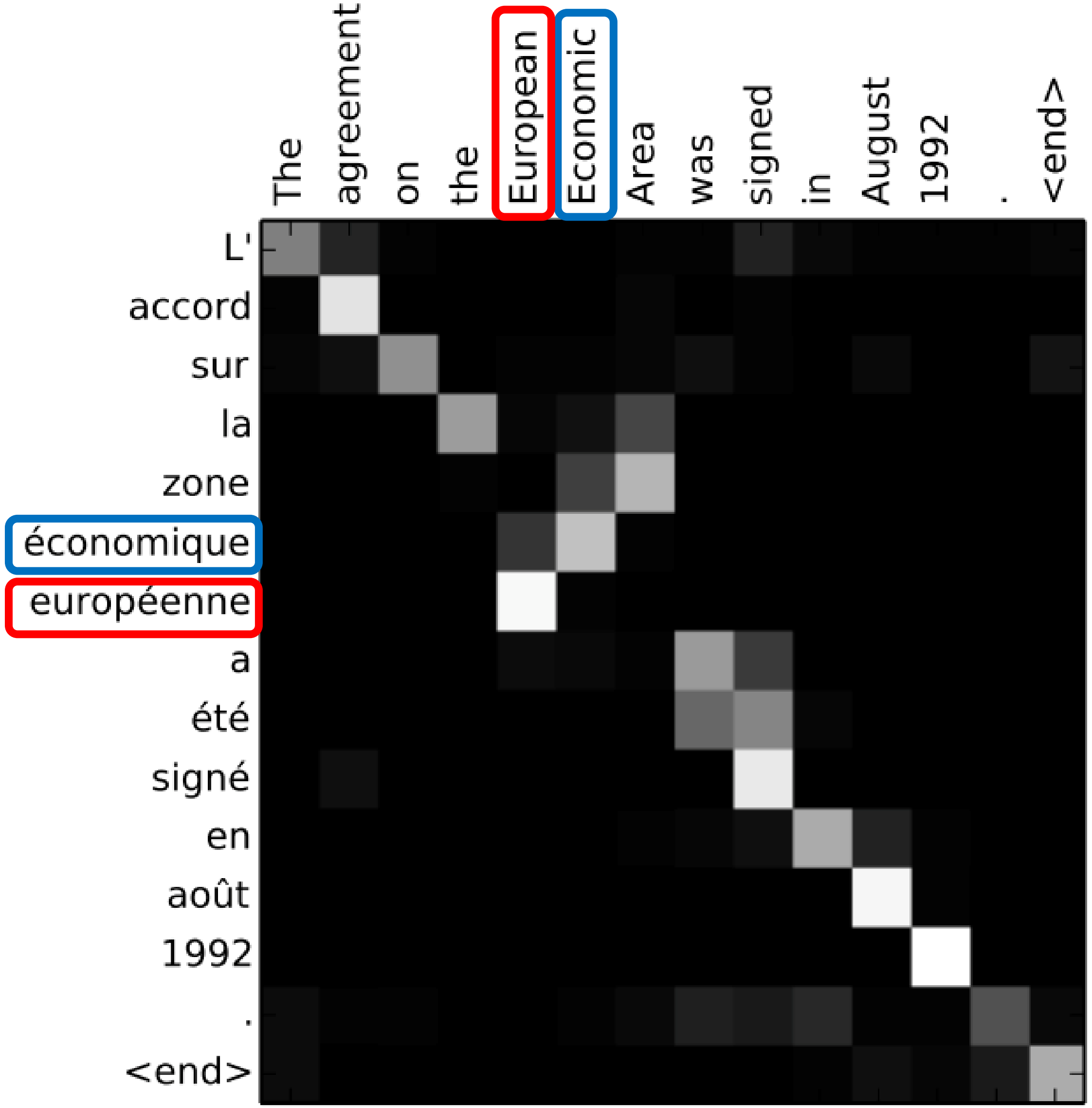
- Challenge: The single encoder output has to store information about the entire sentence in a single vector
- Would be much easier if we can "refer to our notes"
- Traditional MT: Alignment between input & output sentences
- Can we get a neural network to model alignments?

Attention



- Compute similarity between decoder hidden state and each encoder hidden state
    - E.g., dot product, if same size
- Normalize similarities to probability distribution with softmax
- "Context" vector *c* = weighted average of encoder states based on the probabilities
    - No new parameters (like ReLU/max pool)
    - Use *c* when computing decoder outputs or transitions
- Intuition
    - Step 1: Find similar input words
    - Step 2: Grab the encoder representation of those words
    - Step 3: Tell the decoder that this is relevant

Visualizing attention



- Source is English, Target is French
- Each row is a probability distribution over the English text
- Alignment makes sense, overcomes word order differences
  - When generating “économique” attend to “Economic”
  - When generating “européenne” attend to “European”

Conclusion

- Deep Learning for Language must deal with possibly long inputs
- RNNs handle arbitrarily long inputs with fixed number of parameters
- Challenges
  - Long range dependencies
  - Modeling alignments between input and output sequences
- Next time: Can Attention solve everything?