Deep Learning for Language, Part 1: Recurrent Neural Networks

Robin Jia
USC CSCI 467, Fall 2023
October 3, 2023

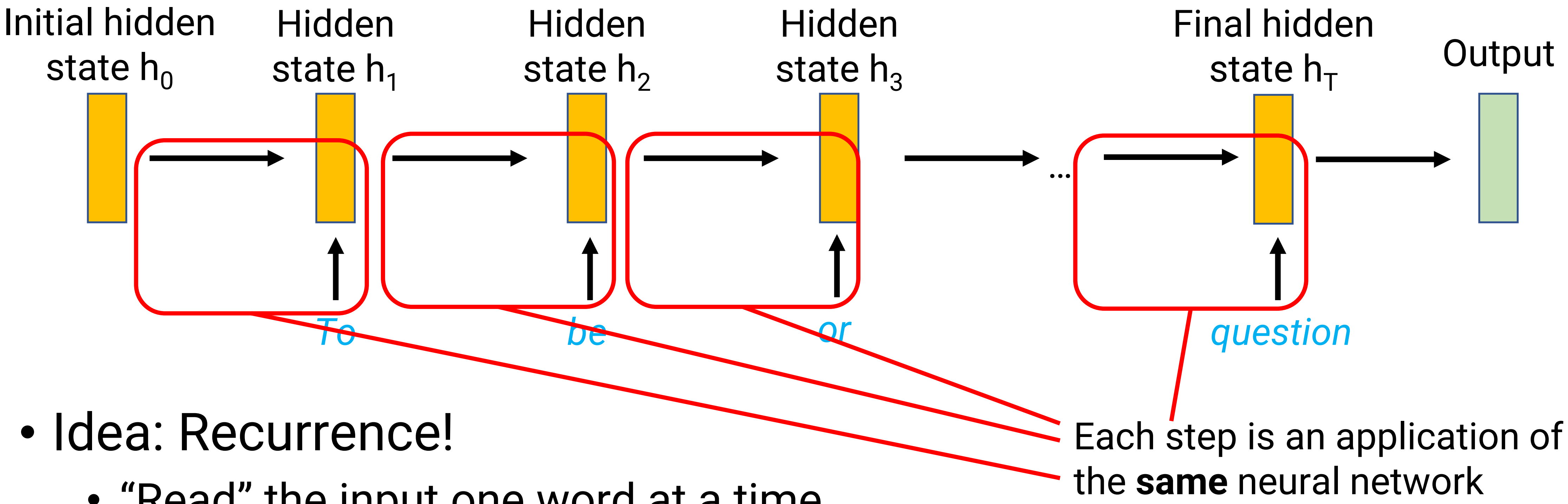
Outline

- Recurrent Neural Networks (RNNs) for sequential data
- Language modeling and Long-range dependencies
- Vanishing gradients and Gated RNNs

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?

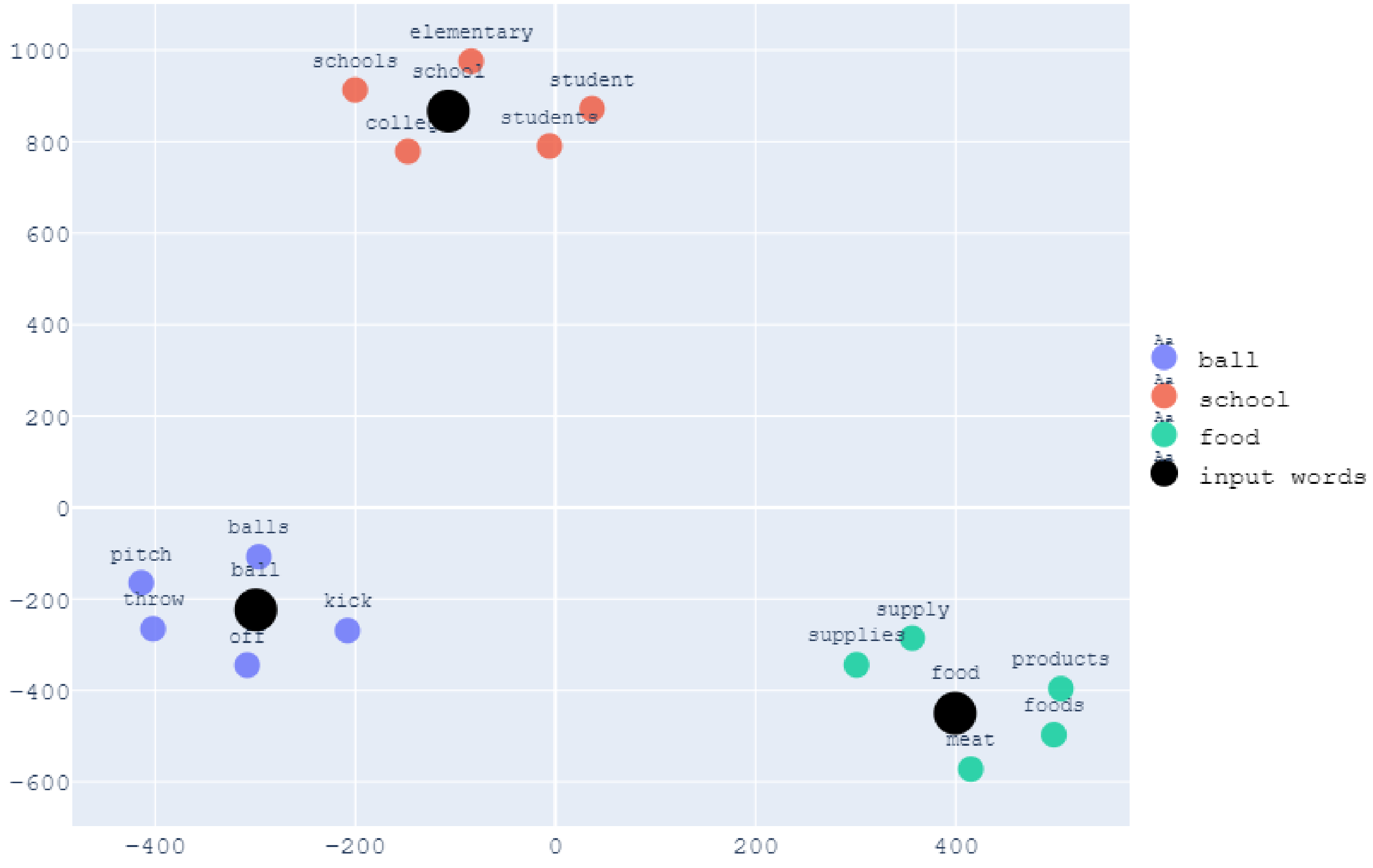
Recurrent Neural Networks (RNNs)



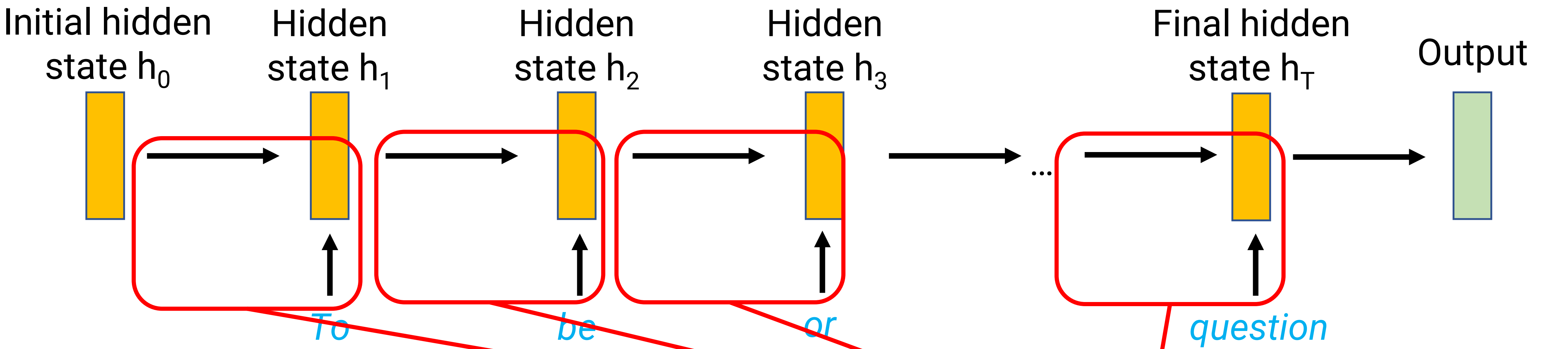
- Idea: Recurrence!
  - "Read" the input one word at a time
  - At each step, update the hidden state of the network
  - **Model parameters to do this update are same for each step**

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!



A "Vanilla"/"Elman" RNN

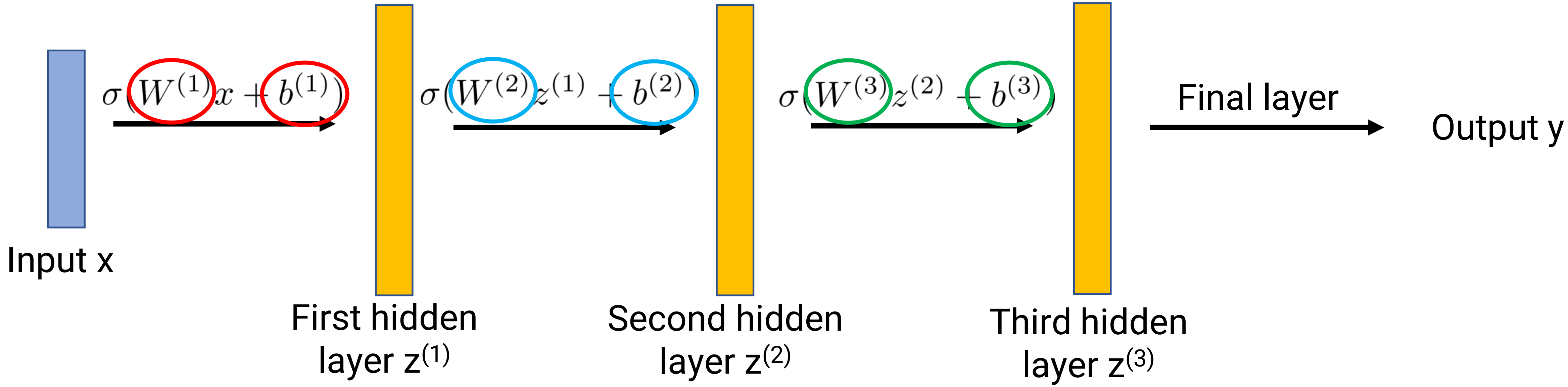


Same W's & b for each timestep

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

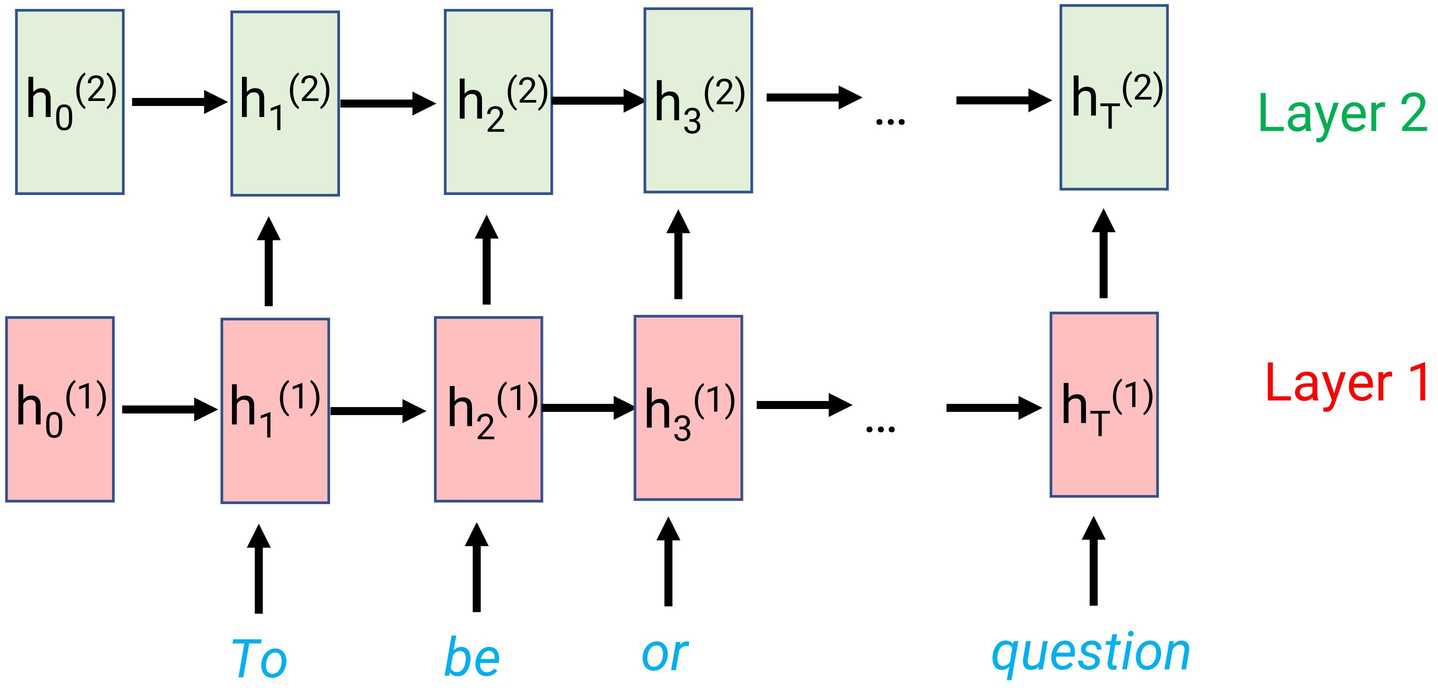
- At each timestep *t*, run a neural network that takes in 2 inputs (or 1 big input, by concatenation)
  - Previous hidden state h_{t-1}
  - Vector for current word x_t
- Learn linear function of both inputs, add bias, apply non-linearity
- Parameters: Recurrence params (W_h, W_x, b), initial hidden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

Recurrence vs. Depth



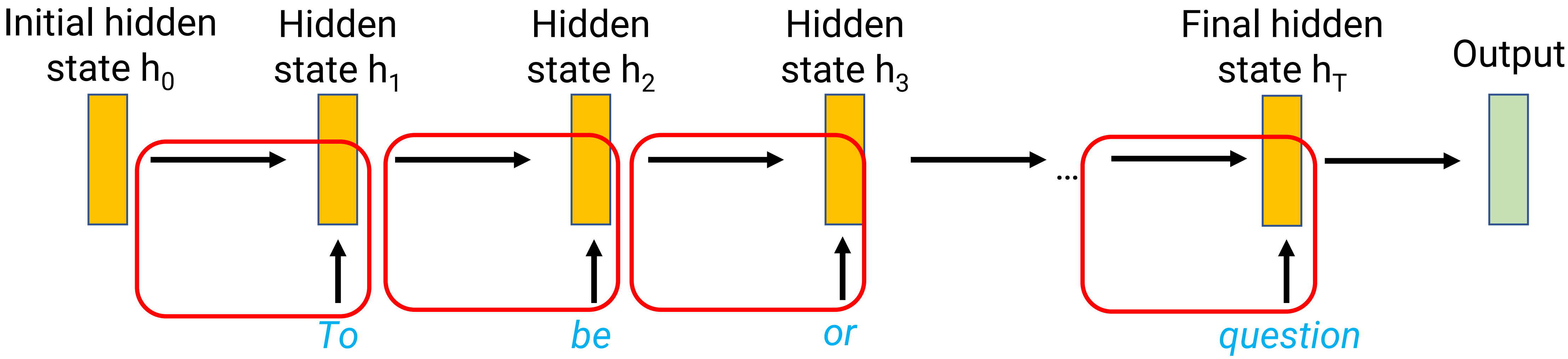
- Deep networks (i.e., adding more layers)
  - Computation graph becomes longer
  - Number of parameters also grows; each step uses new parameters
- Recurrent neural networks
  - Computation graph becomes longer
  - Number of parameters **fixed**; each step uses **same parameters**

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

Training an RNN

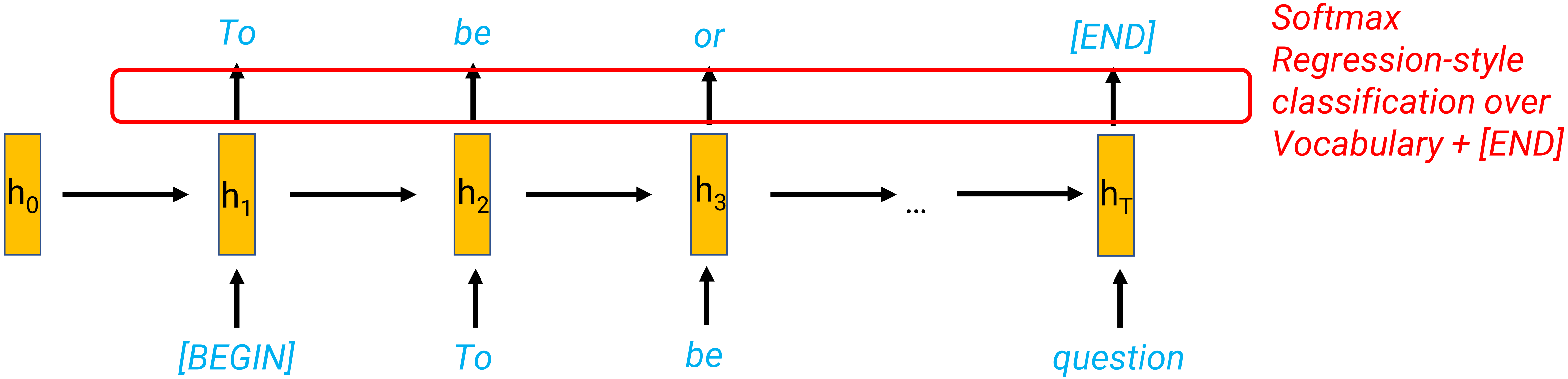


- Same recipe: Backpropagation to compute gradients + gradient descent
- Must backpropagate through whole computation graph
  - “Backpropagation through time”
  - Same weights for recurrence used at every time step; total change to weights is added up for each timestep

Outline

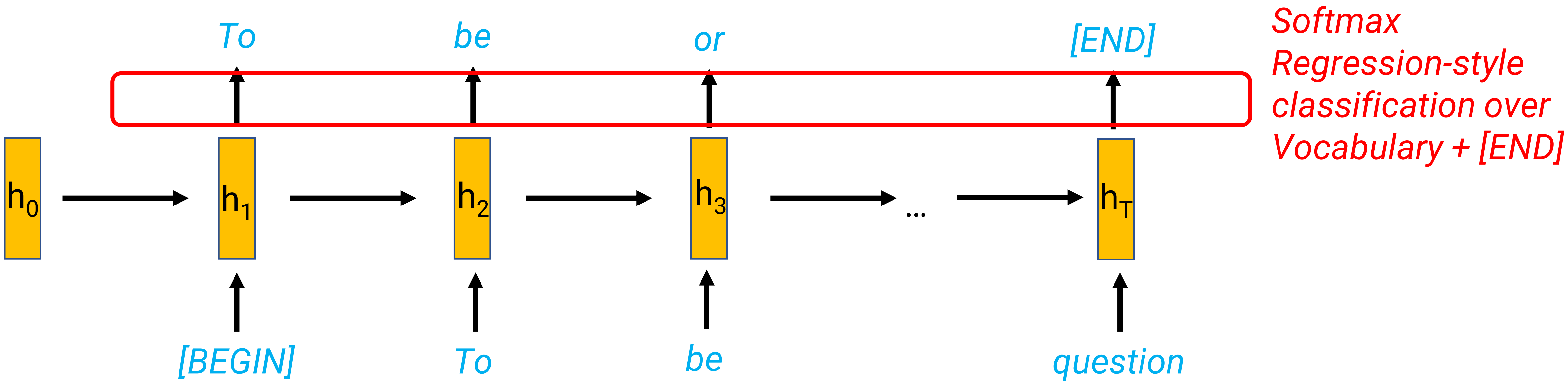
- Recurrent Neural Networks (RNNs) for sequential data
- Language modeling and Long-range dependencies
- Vanishing gradients and Gated RNNs

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

Language Modeling ("Decoder only")



- Training a language model
  - Input sequence is a real human-written document
  - For each word, compute classification loss (like softmax regression) for model, using the actual human-written next word as the correct "label"
  - Sum up loss for whole document, then backpropagate & update parameters with gradient descent

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**

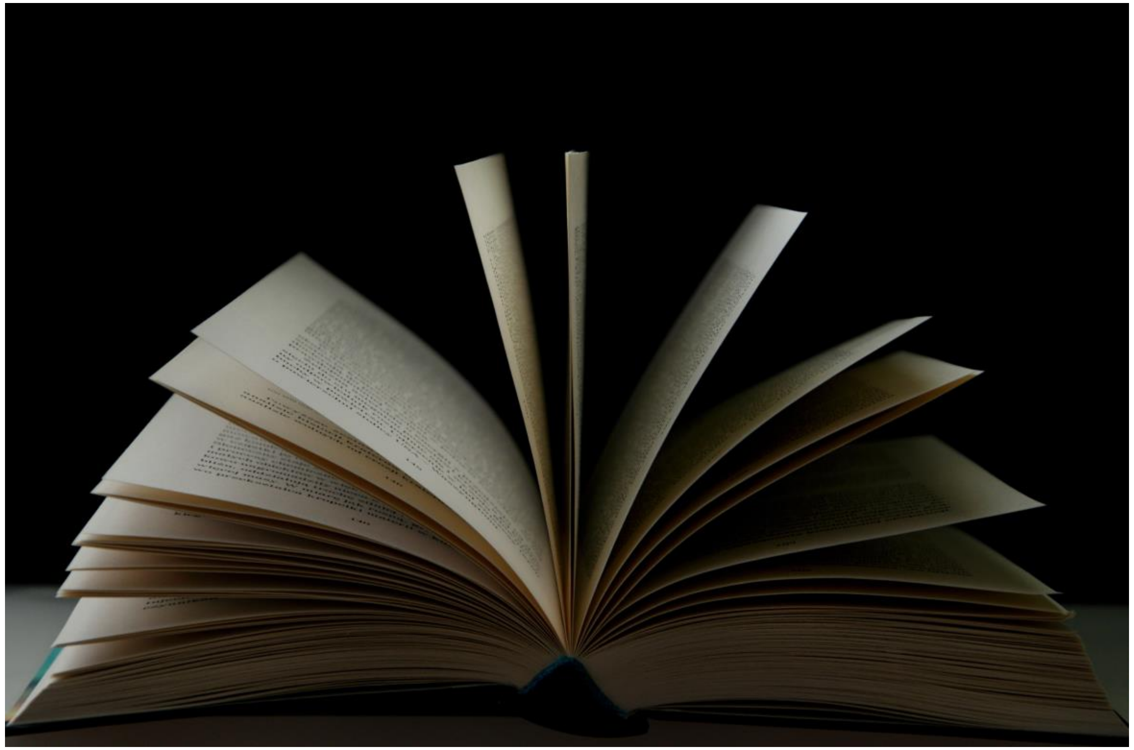
The keys to the cabinet by the door on the left ***are*** *(on the table)*

Long-Range Dependencies

"I voted for Nader because he was most aligned with my values," she said.

- "Coreference": When two words refer to the same underlying person/place/thing
  - Pronouns typically ***corefer*** to an ***antecedent*** (something mentioned earlier in the text)
- Coreference relationships can even span multiple sentences

Even longer-range dependencies



- Imagine trying to generate a novel…
  - Same set of characters
  - Characters have to behave in consistent ways
  - Sensible ordering of events

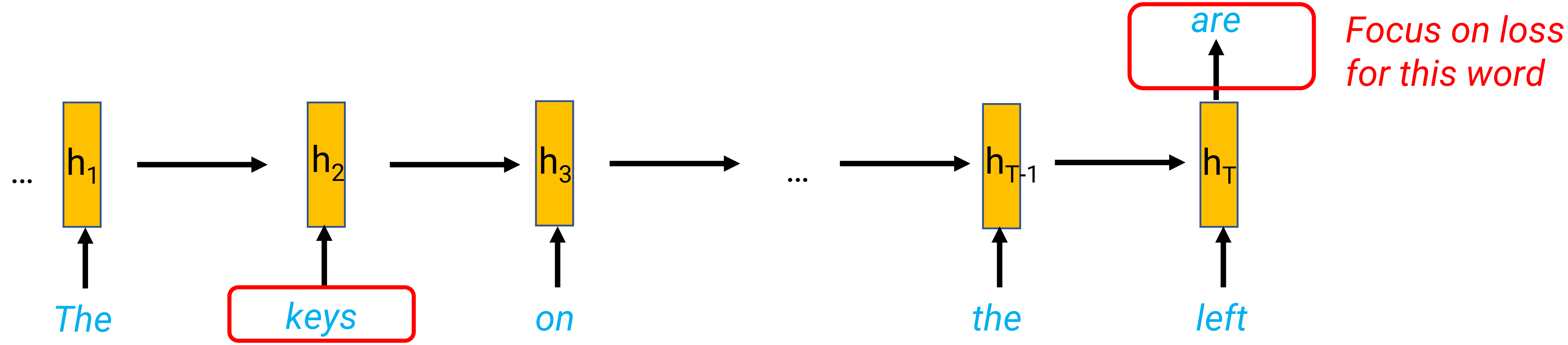
Announcements

- HW2 due this Thursday
- Thursday class: A bit more on RNNs + first half review
- Section Friday: Midterm Review (practice exam + questions)
- Midterm exam next Tuesday, October 10
  - In-class, 80 minutes, one double-sided 8.5x11 sheet of notes
  - Practice exam posted
  - Room assignments (also on Piazza)
    - Last name A-O: LVL 17 (this room)
    - Last name P-Z: THH 116

Outline

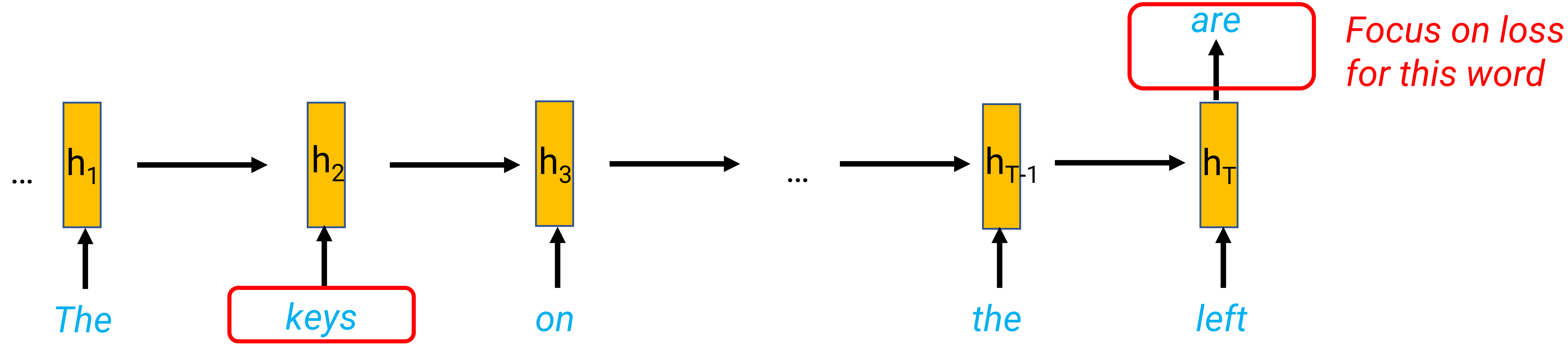
- Recurrent Neural Networks (RNNs) for sequential data
- Language modeling and Long-range dependencies
- Vanishing gradients and Gated RNNs

Backpropagation through time, revisited



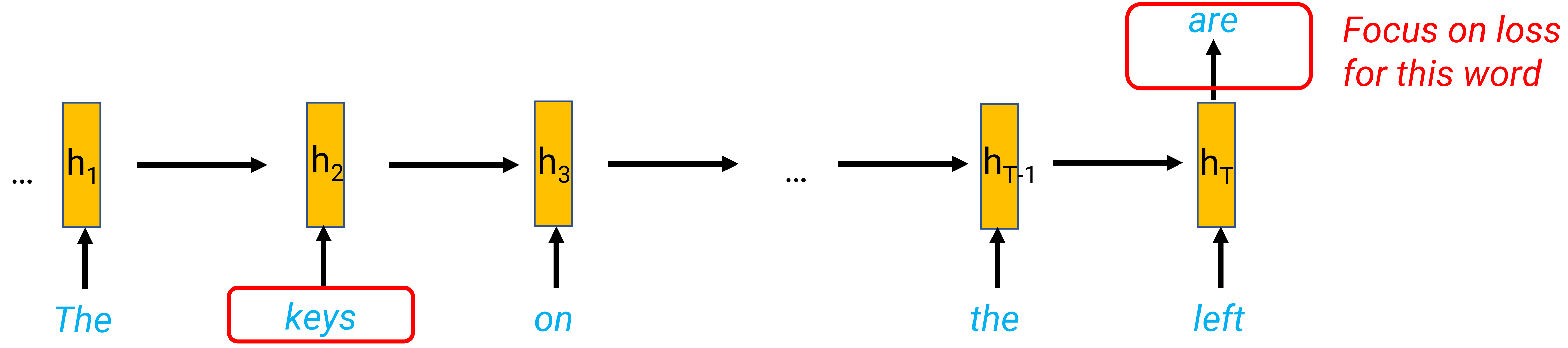
- Model needs to know that the correct word is *are* because of the word *keys*!
- Let's backpropagate the loss on generating *are* to the word vector parameters for *keys*
  - For simplicity, let's assume all the hidden states are just 1-dimensional
  - Step 1: Compute $\delta Loss/\delta(h_T)$
  - Step 2: Compute $\delta Loss/\delta(h_{T-1}) = \delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1})$
  - Step 3: Compute $\delta Loss/\delta(h_{T-2}) = \delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1}) * \delta(h_{T-1})/\delta(h_{T-2})$
  - …
  - Gradient through "*keys*" hidden state: $\delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1}) * \delta(h_{T-1})/\delta(h_{T-2}) * \ldots * \delta(h_3)/\delta(h_2)$
  - Gradient through "*keys*" word vector: $\delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1}) * \delta(h_{T-1})/\delta(h_{T-2}) * \ldots * \delta(h_3)/\delta(h_2) * \delta(h_2)/\delta(x_2)$

The Vanishing Gradient Problem



- Gradient through "*keys*" word vector: $\delta Loss/\delta(h_T)$ * $\delta(h_T)/\delta(h_{T-1})$ * $\delta(h_{T-1})/\delta(h_{T-2})$ * … * $\delta(h_3)/\delta(h_2)$ * $\delta(h_2)/\delta(x_2)$
  - What is each individual $\delta(h_t)/\delta(h_{t-1})$ term ?
  - Elman network: $h_t = \tanh\left(W_h h_{t-1} + W_x x_t + b\right),\quad \frac{\delta h_t}{\delta h_{t-1}} = \underbrace{\tanh'\left(W_h h_{t-1} + W_x x_t + b\right)}_{\text{Ignore for now}} \cdot \underbrace{W_h}_{\text{The same parameter over and over!}}$
  - After *t* timesteps, have a factor of $(W_h)^t$ (to the *t*-th power)!
  - If W_h << 1, this quickly becomes 0 ("vanishes")

The Vanishing Gradient Problem

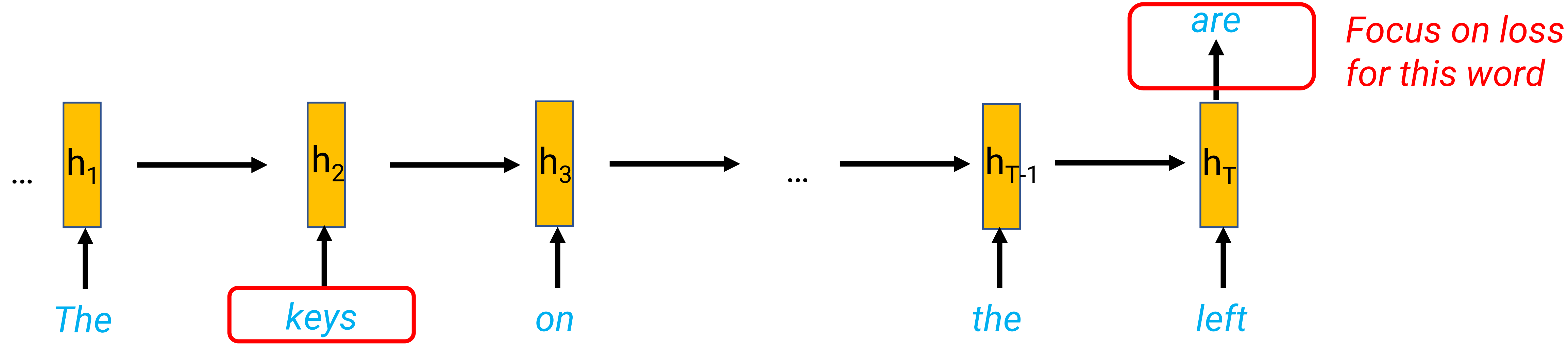


- Vanishing Gradients: Updates to one word/hidden state not influenced by loss on words many steps in the future
  - Illustrated only for 1-dimensional hidden states, but same thing happens when states are vectors/parameters are matrices
- Result: Hard for model to learn long-range dependencies!

Vanishing and Exploding

- Vanishing gradient occurs because
  - Gradient w.r.t. words t steps in the past has $(W_h)^t$
  - And when $W_h << 1$ (e.g., at initialization time)
- What if $W_h > 1$?
  - Gradients get bigger as you go backwards in time: Exploding gradients!
  - Vanishing gradients more usual, but explosion can happen too
- Quick fix: Gradient Clipping
  - If gradient is super large, “clip” it to some maximum amount
    - Rescale the total vector to some maximum norm
    - Clip each entry to be within some minimum/maximum value
- Outside of RNNs, vanishing/exploding gradients can happen whenever you have long computation graphs with lots of multiplications

Avoiding Vanishing Gradients



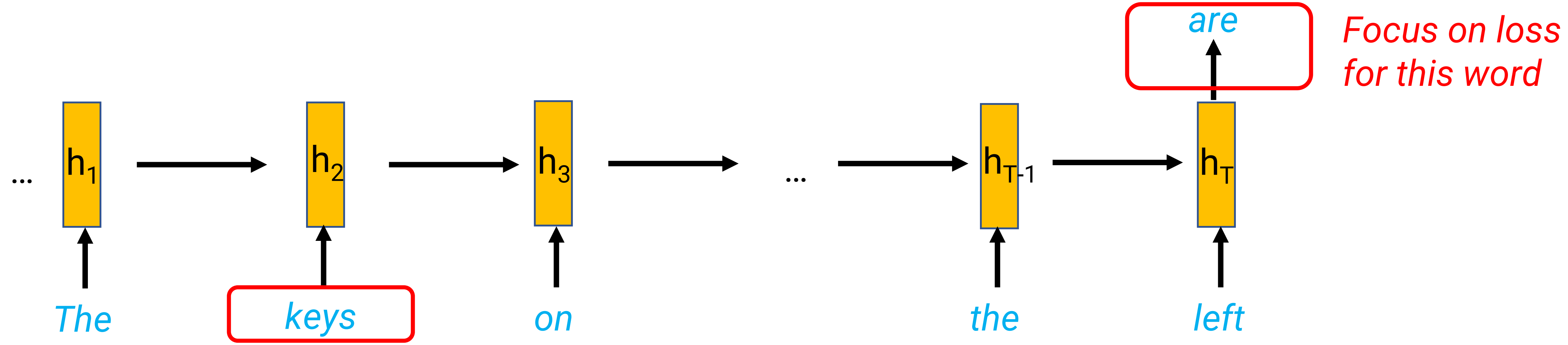
- Where did we go wrong?

$$h_t = \tanh\left(\underbrace{W_h h_{t-1}} + W_x x_t + b\right), \quad \frac{\delta h_t}{\delta h_{t-1}} = \tanh'\left(W_h h_{t-1} + W_x x_t + b\right) \underbrace{\cdot\, W_h}$$

Multiplicative relationship between previous state and next state

Leads to repeated multiplication by W_h

Avoiding Vanishing Gradients

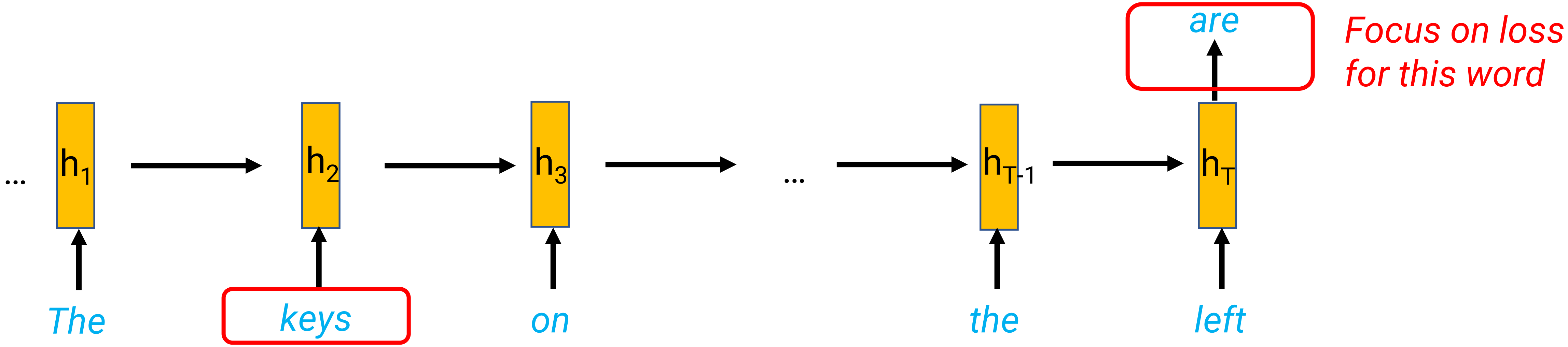


- Extreme idea: A purely additive relationship
  - Pro: No vanishing gradients
  - Pro: Old hidden state carried through to all future times
  - Con: May be good to "forget" irrelevant information about old states

$$h_t = h_{t-1} + \underbrace{g(h_{t-1}, x_t)}_{\text{Additive relationship}},$$

$$\frac{\delta h_t}{\delta h_{t-1}} = 1 + \underbrace{\frac{\delta}{\delta h_{t-1}} g(h_{t-1}, x_t)}_{\text{Gradients also add, not multiply}}$$

Avoiding Vanishing Gradients



- Middle-ground: **Gated** recurrence relationship
  - Additive component makes gradients add, not multiply = less vanishing gradients
  - Forget gate allows for selectively “forgetting” some neurons within hidden state
  - When forget gate is all 1’s, becomes the purely additive model (no vanishing)

Elementwise multiplication

$$h_t = h_{t-1} \odot \underbrace{f(h_{t-1}, x_t)}_{\text{“forget gate” in [0, 1]}} + \underbrace{g(h_{t-1}, x_t)}_{\text{Additive relationship}}$$

Gated Recurrent Units (GRUs)

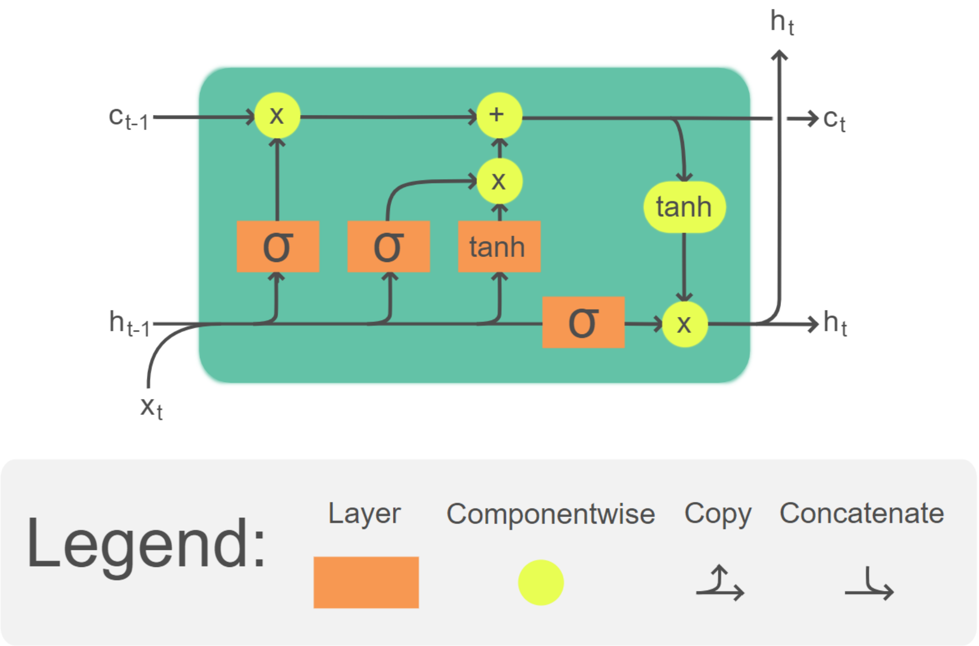
- One type of gated RNN
  - Here z is the “forget gate” vector
  - Where $z_i = 0$:
    - Forget this neuron
    - Allow updating its value
  - Where $z_i = 1$:
    - Don’t forget this neuron
    - Do not allow updating its value
- Parameters: *W*, *U*, plus parameters of *g*
  - (*g* has a slightly complicated form not shown, has its own parameters)

$$h_t = h_{t-1} \overbrace{\odot\, z}^{\text{“forget gate”}} + \overbrace{g(x_t, h_{t-1}) \odot (1 - z)}^{\text{Additive relationship}}$$

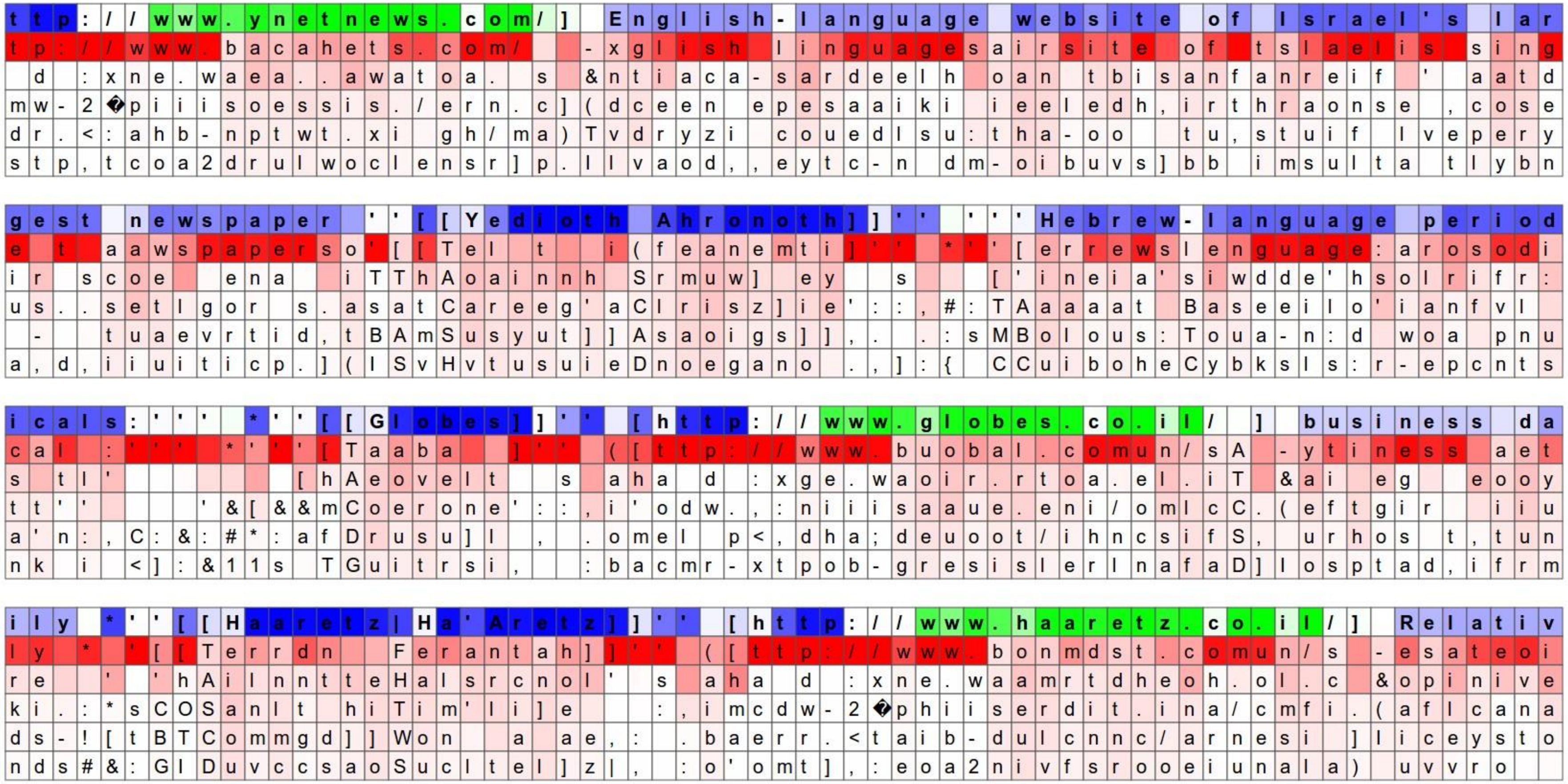
$$z = \underbrace{\sigma(W x_t + U h_{t-1})}_{\text{Sigmoid ensures gate is between 0 and 1}}$$

Long Short-Term Memory (LSTM)

- Another, more complicated gated RNN
- Commonly used in practice
- What's the idea?
  - Split the hidden state into normal hidden state h_t and "cell" state c_t
  - Cell state uses gated recurrence
  - Hidden state is gated function of cell state

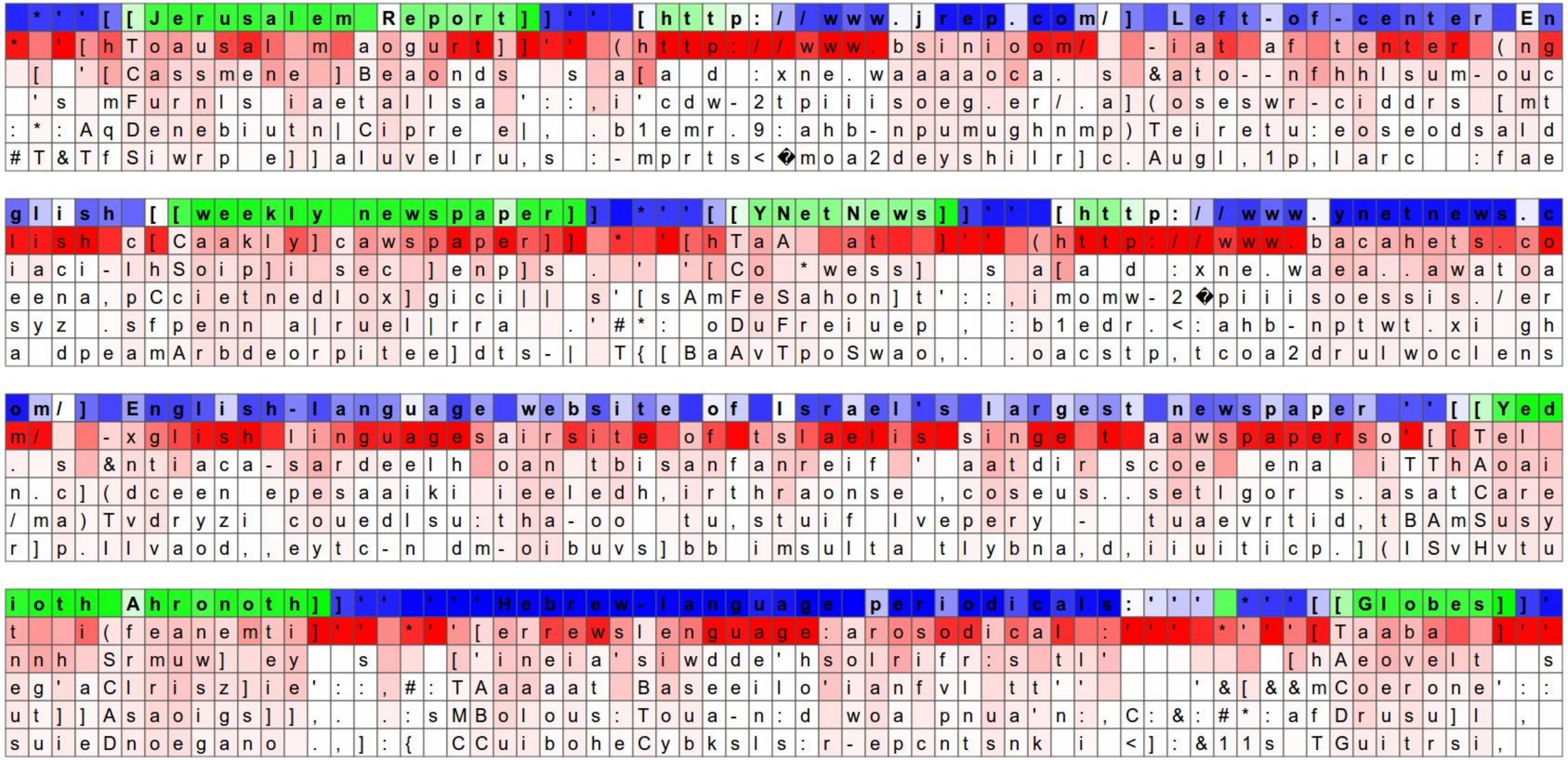


What do LSTMs learn?



- Here: a character-level LSTM (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 LSTMs learn?



- Here: a character-level LSTM (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?)

Conclusion

- Deep Learning for Language must deal with possibly long inputs
- RNNs handle arbitrarily long inputs with fixed number of parameters
- Need to handle long-range dependencies, but hard to learn due to vanishing gradients
- Gated RNNs (GRUs, LSTMs) can reduce vanishing gradient problems