Recurrent Neural Networks (RNNs)

Deep Learning for NLP: Lecture 7

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München
poerner@cis.uni-muenchen.de

December 16, 2020

Outline

1. What are RNNs?
2. Vanilla RNN
3. Training RNNs
   - Backpropagation through time
   - The vanishing gradients problem
4. Gated RNNs
   - LSTM
   - GRU
   - Comparison
5. RNN applications
6. Extensions: Multi-layer RNNs, bidirectionality
7. RNNs with attention

Outline

1. What are RNNs?
2. Vanilla RNN
3. Training RNNs
   - Backpropagation through time
   - The vanishing gradients problem
4. Gated RNNs
   - LSTM
   - GRU
   - Comparison
5. RNN applications
6. Extensions: Multi-layer RNNs, bidirectionality
7. RNNs with attention

What are RNNs?

- Assumption: Text is written sequentially, so our model should read it sequentially
- "RNN": class of Neural Network architectures that process text sequentially (left-to-right or right-to-left)
- Generally speaking:
  - Internal "state" $\mathbf{h}$
  - RNN consumes one input $\mathbf{x}^{(j)}$ per time step j
  - Update function: $\mathbf{h}^{(j)} = f(\mathbf{x}^{(j)}, \mathbf{h}^{(j-1)}; \theta)$
    - ★ where parameters θ are shared across all time steps

Outline

1. What are RNNs?
2. Vanilla RNN
3. Training RNNs
   - Backpropagation through time
   - The vanishing gradients problem
4. Gated RNNs
   - LSTM
   - GRU
   - Comparison
5. RNN applications
6. Extensions: Multi-layer RNNs, bidirectionality
7. RNNs with attention

Vanilla RNN

- Let $\mathbf{x}^{(1)} \dots \mathbf{x}^{(J)}$, with $\mathbf{x}^{(j)} \in \mathbb{R}^{d'}$ be our input (e.g., a sequence of d'-dimensional word embeddings)
- Let $\mathbf{h}^{(0)} = \{0\}^d$ be our initial state
- Let $\theta = \{\mathbf{W} \in \mathbb{R}^{d \times d'}, \mathbf{V} \in \mathbb{R}^{d \times d}, \mathbf{b} \in \mathbb{R}^d\}$ be our parameters
- Then the j'th update is defined as:

$$\mathbf{h}^{(j)} = \tanh(\mathbf{W}\mathbf{x}^{(j)} + \mathbf{V}\mathbf{h}^{(j-1)} + \mathbf{b})$$

Vanilla RNN

- Sentence: "the cat sat"
- $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}) = (\mathbf{x}^{(\text{the})}, \mathbf{x}^{(\text{cat})}, \mathbf{x}^{(\text{sat})})$

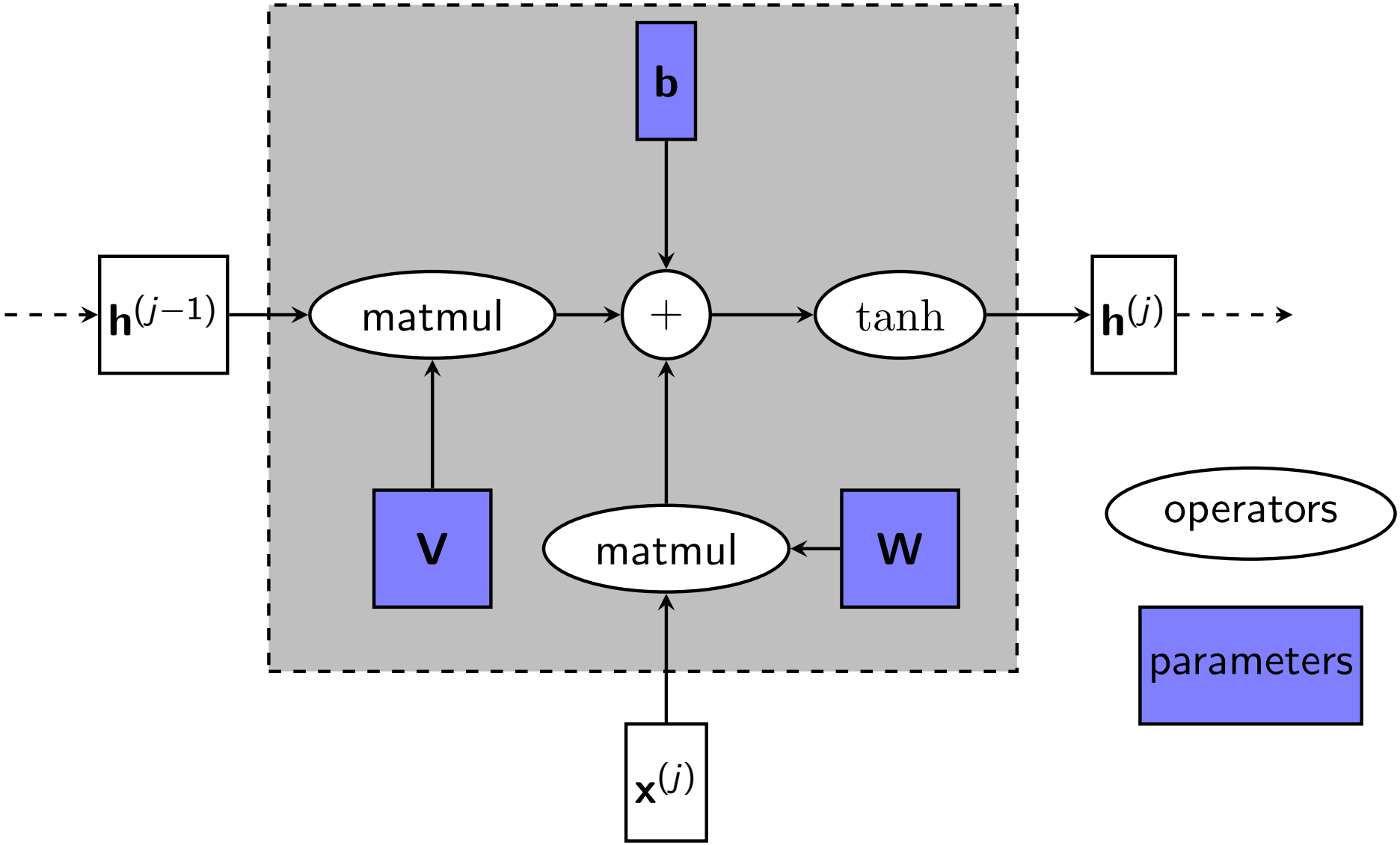
$$\mathbf{h}^{(1)} = \tanh(\mathbf{W}\mathbf{x}^{(\text{the})} + \mathbf{V}\mathbf{h}^{(0)} + \mathbf{b}) = \tanh(\mathbf{W}\mathbf{x}^{(\text{the})} + \mathbf{b})$$

$$\mathbf{h}^{(2)} = \tanh(\mathbf{W}\mathbf{x}^{(\text{cat})} + \mathbf{V}\mathbf{h}^{(1)} + \mathbf{b}) = \tanh(\mathbf{W}\mathbf{x}^{(\text{cat})} + \mathbf{V}\tanh(\mathbf{W}\mathbf{x}^{(\text{the})} + \mathbf{b}) + \mathbf{b})$$

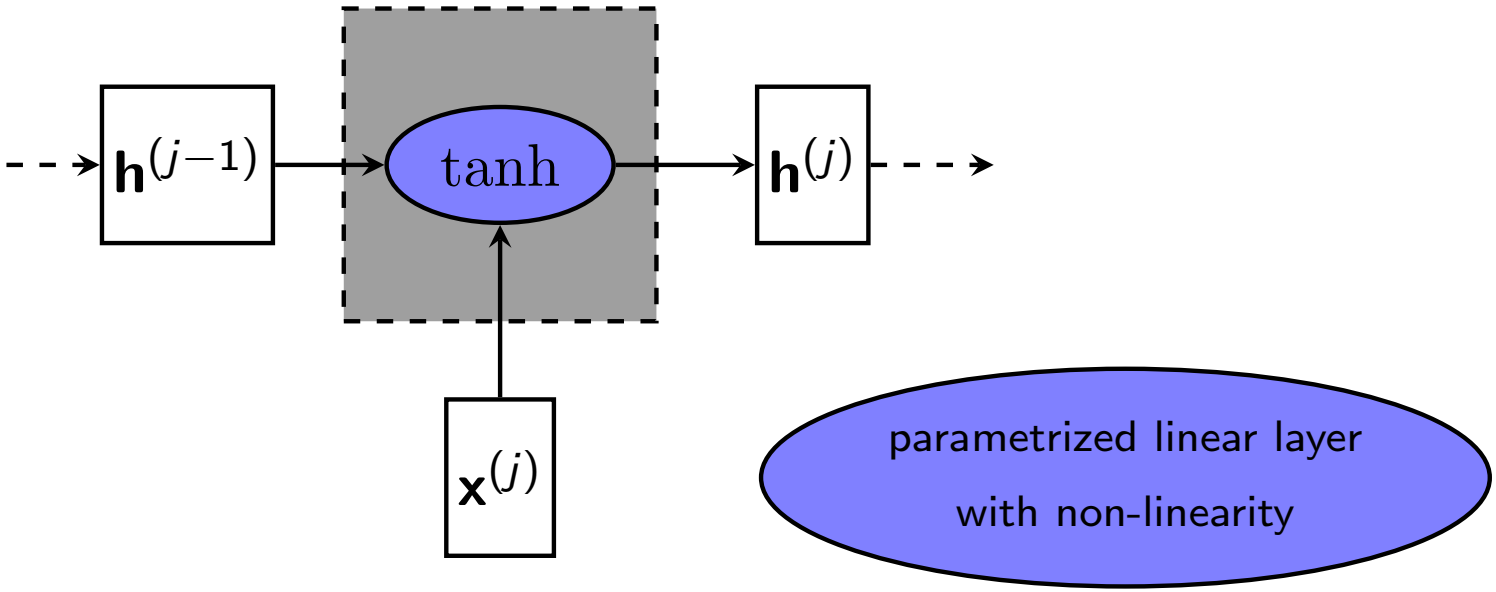
$$\mathbf{h}^{(3)} = \tanh(\mathbf{W}\mathbf{x}^{(\text{sat})} + \mathbf{V}\mathbf{h}^{(2)} + \mathbf{b}) = \ldots$$

- Question: Which word(s) does $\mathbf{h}^{(1)}$ depend on?
  - "the"
- Question: Which word(s) does $\mathbf{h}^{(2)}$ depend on?
  - "the cat"
- Question: Which word(s) does $\mathbf{h}^{(3)}$ depend on?
  - "the cat sat"

Vanilla RNN cell



Vanilla RNN cell (simplified)



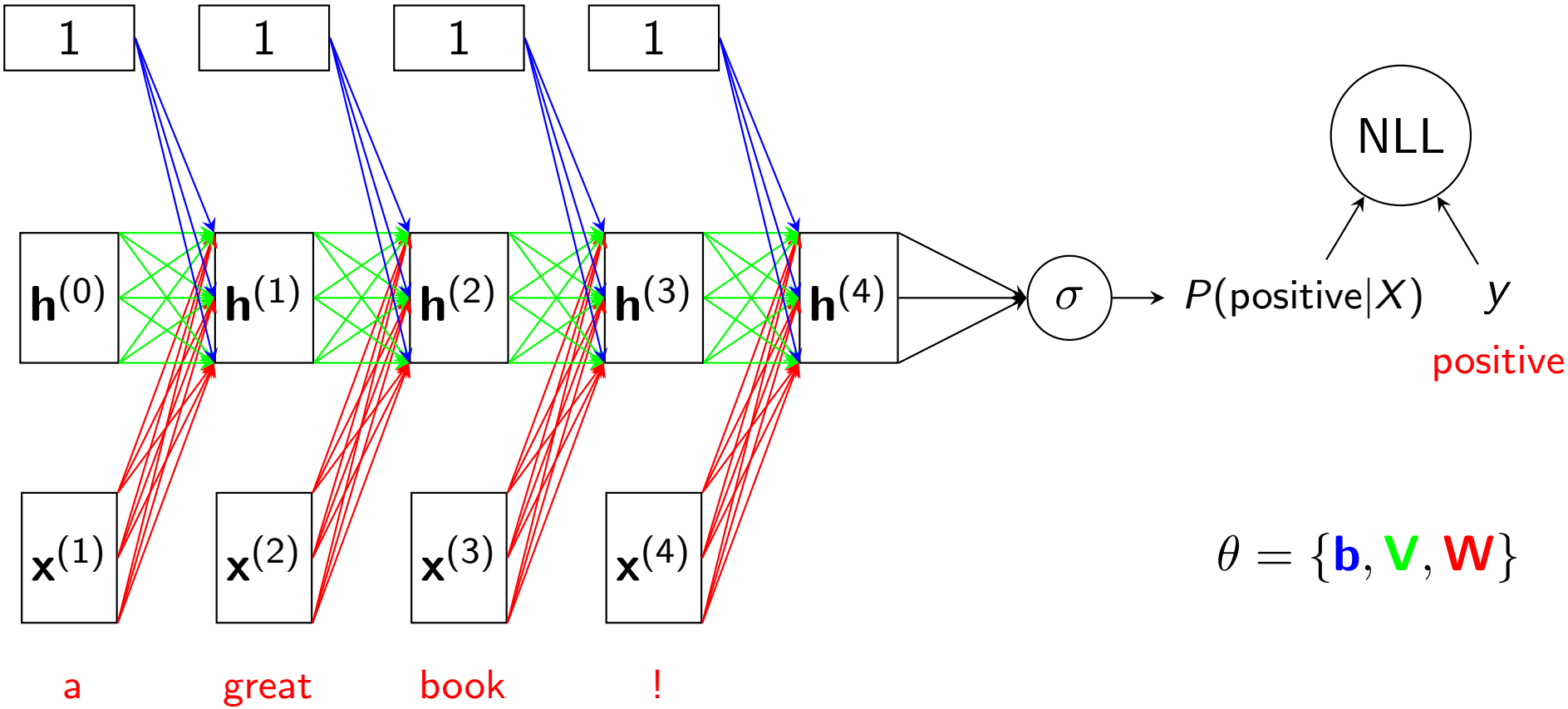
Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1. What are RNNs?
2. Vanilla RNN
3. Training RNNs
   - Backpropagation through time
   - The vanishing gradients problem
4. Gated RNNs
   - LSTM
   - GRU
   - Comparison
5. RNN applications
6. Extensions: Multi-layer RNNs, bidirectionality
7. RNNs with attention

Example: Binary sentiment classifier



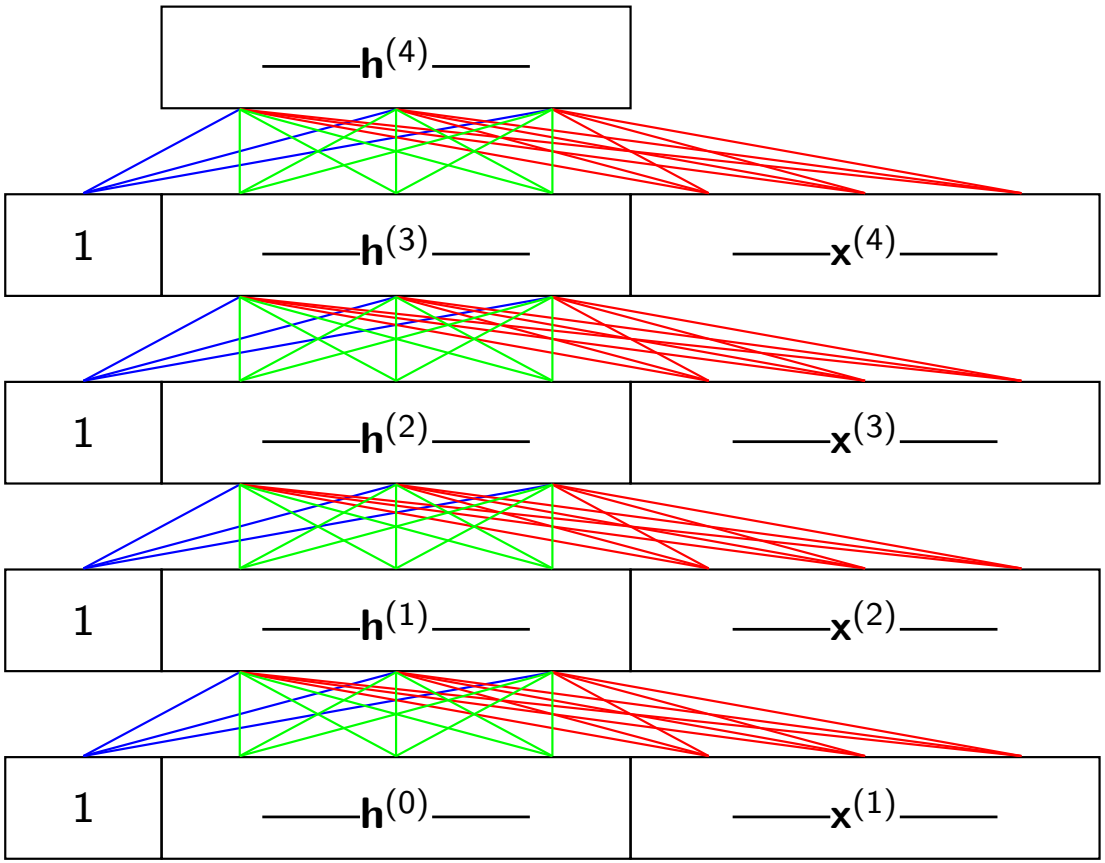
$$\theta = \{\mathbf{b}, \mathbf{V}, \mathbf{W}\}$$

(Non-linearity omitted for readability.)

Backpropagation through time

- RNNs are trained via "backpropagation through time"
- To understand how this works, imagine the RNN as a Feed-Forward Net (FFN), whose depth is equal to the sentence length
- For now, let's pretend that every time step (every layer) has its own dummy parameters $\theta^{(j)}$, which are identical copies of θ

Vanilla RNN as Feed-Forward Net



$$\theta^{(4)} = \{\mathbf{b}^{(4)}, \mathbf{V}^{(4)}, \mathbf{W}^{(4)}\}$$

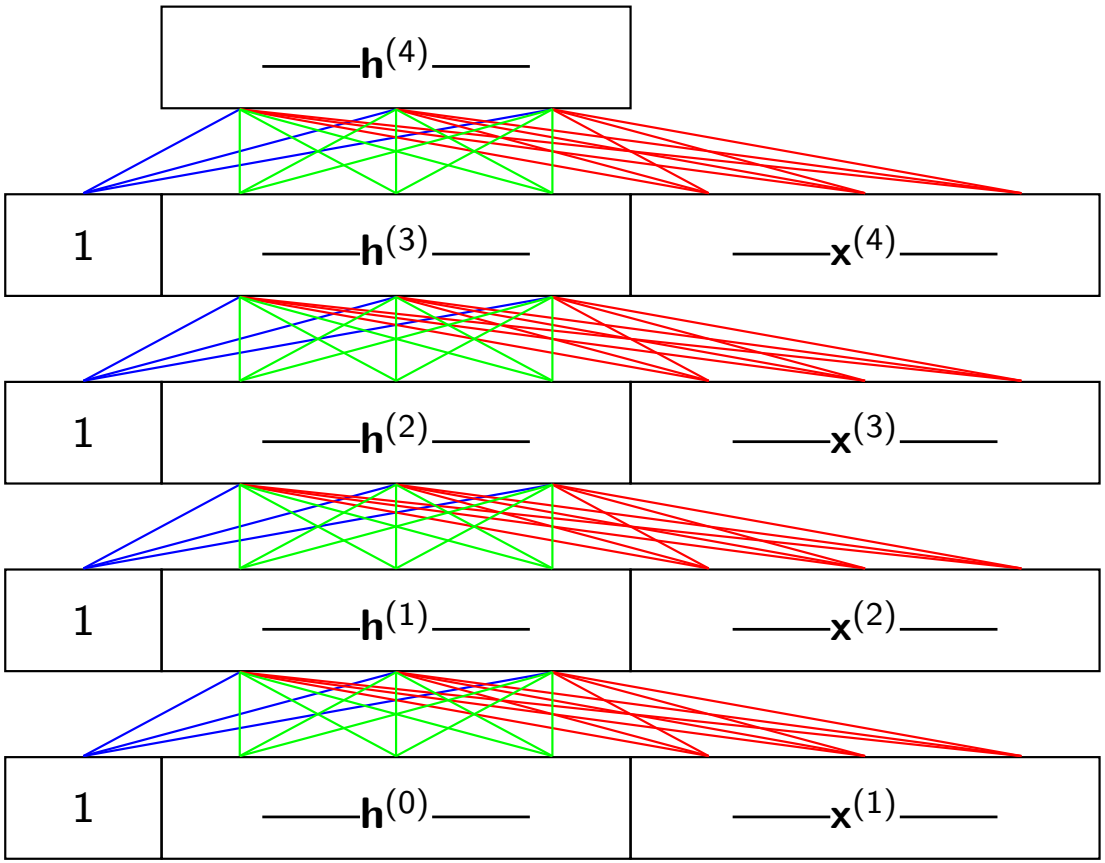
$$\theta^{(3)} = \{\mathbf{b}^{(3)}, \mathbf{V}^{(3)}, \mathbf{W}^{(3)}\}$$

$$\theta^{(2)} = \{\mathbf{b}^{(2)}, \mathbf{V}^{(2)}, \mathbf{W}^{(2)}\}$$

$$\theta^{(1)} = \{\mathbf{b}^{(1)}, \mathbf{V}^{(1)}, \mathbf{W}^{(1)}\}$$

Backpropagating through time

- Assume that we have calculated derivatives of loss L w.r.t. last state: $\frac{\partial L}{\partial \mathbf{h}^{(J)}}$
- Then we can apply the normal backpropagation algorithm for FFNs to calculate $\nabla_{\mathbf{W}^{(i)}} L, \nabla_{\mathbf{V}^{(i)}} L, \nabla_{\mathbf{b}^{(i)}} L$ for all dummy parameters.



$\{\nabla_{\mathbf{b}^{(4)}} L, \nabla_{\mathbf{V}^{(4)}} L, \nabla_{\mathbf{W}^{(4)}} L\}$

$\{\nabla_{\mathbf{b}^{(3)}} L, \nabla_{\mathbf{V}^{(3)}} L, \nabla_{\mathbf{W}^{(3)}} L\}$

$\{\nabla_{\mathbf{b}^{(2)}} L, \nabla_{\mathbf{V}^{(2)}} L, \nabla_{\mathbf{W}^{(2)}} L\}$

$\{\nabla_{\mathbf{b}^{(1)}} L, \nabla_{\mathbf{V}^{(1)}} L, \nabla_{\mathbf{W}^{(1)}} L\}$

Backpropagating through time

- But of course, there is only one set of parameters.
- So the actual gradients are:

$$\nabla_{\mathbf{W}} L = \sum_{j=1}^{J} \nabla_{\mathbf{W}^{(j)}} L$$

$$\nabla_{\mathbf{V}} L = \sum_{j=1}^{J} \nabla_{\mathbf{V}^{(j)}} L$$

$$\nabla_{\mathbf{b}} L = \sum_{j=1}^{J} \nabla_{\mathbf{b}^{(j)}} L$$

- (In reality, we are of course using pytorch to calculate the gradients.)

Vanishing gradients

- On long inputs, Vanilla RNNs suffer from "vanishing" gradients
- Vanishing gradients mean that the impact that an input has on the gradient of the loss becomes smaller when it is further away from the loss in the computation graph.

Vanishing gradients

- We assume that we have backpropagated the partial derivatives of the loss to $\mathbf{h}^{(J)}$:

$$\frac{\partial L}{\partial \mathbf{h}^{(J)}}$$

- Backpropagate to $\mathbf{h}^{(j)}$ via chain rule:

$$\frac{\partial L}{\partial \mathbf{h}^{(j)}} = \Big[\prod_{j'=j+1}^{J} \Big(\frac{\partial \mathbf{h}^{(j')}}{\partial \mathbf{h}^{(j'-1)}}\Big)^T\Big] \frac{\partial L}{\partial \mathbf{h}^{(J)}}$$

$$= \Big[\prod_{j'=j+1}^{J} \Big(\frac{\partial \mathbf{V}\mathbf{h}^{(j'-1)}}{\partial \mathbf{h}^{(j'-1)}}\Big)^T \frac{\partial \tanh(\mathbf{V}\mathbf{h}^{(j'-1)} + \ldots)}{\partial \mathbf{V}\mathbf{h}^{(j'-1)}}\Big] \frac{\partial L}{\partial \mathbf{h}^{(J)}}$$

Vanishing gradients

$$\frac{\partial L}{\partial \mathbf{h}^{(j)}} = \Big[\prod_{j'=j+1}^{J} \Big(\frac{\partial \mathbf{V}\mathbf{h}^{(j'-1)}}{\partial \mathbf{h}^{(j'-1)}} \Big)^T \frac{\partial \tanh(\mathbf{V}\mathbf{h}^{(j'-1)} + \ldots)}{\partial \mathbf{V}\mathbf{h}^{(j'-1)}} \Big] \frac{\partial L}{\partial \mathbf{h}^{(J)}}$$

- What happens to $\frac{\partial L}{\partial \mathbf{h}^{(j)}}$ when the distance $J - j$ grows?
  - Remember that $\tanh$ is applied elementwise, and that the derivative of $\tanh$ is between 0 an 1. So the red Jacobian matrix is a diagonal matrix with entries between 0 and 1. The product of many such matrices approaches zero.
  - Furthermore, the blue Jacobian matrix is just $\mathbf{V}$. When initialized with small enough values, $\prod \mathbf{V}$ will approach zero as well.
  - As a result, $\frac{\partial L}{\partial \mathbf{h}^{(j)}}$ approaches zero ("vanishes")
- What does this mean?
  - Since the "dummy parameter gradients" of step j, $\nabla_{\theta^{(j)}} L$ are upstream from $\frac{\partial L}{\partial \mathbf{h}^{(j)}}$, they approach zero too, i.e., their effect on the "dummy gradient sum" is negligible.
  - This means that if the words that your RNN should be paying attention to are far from the loss, the network will not (or slowly) adjust its weights to those words

Exploding gradients

- So why don't we just use a nonlinearity with a derivative larger than 1, or initialize $\mathbf{V}$ differently?
  - $||\frac{\partial L}{\partial \mathbf{h}^{(j)}}||$ would become very large ("explode"). This is even worse than vanishing gradients, because it leads to non-convergence of gradient descent.
  - So vanishing gradients is the lesser of two evils.

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1. What are RNNs?
2. Vanilla RNN
3. Training RNNs
   - Backpropagation through time
   - The vanishing gradients problem
4. Gated RNNs
   - LSTM
   - GRU
   - Comparison
5. RNN applications
6. Extensions: Multi-layer RNNs, bidirectionality
7. RNNs with attention

Long-Short Term Memory Network (LSTM)

- Proposed in Hochreiter and Schmidhuber, 1997
- Became popular around 2010 for handwriting recognition, speech recognition, and many NLP problems
- Addresses vanishing gradients by changing the architecture of the RNN cell

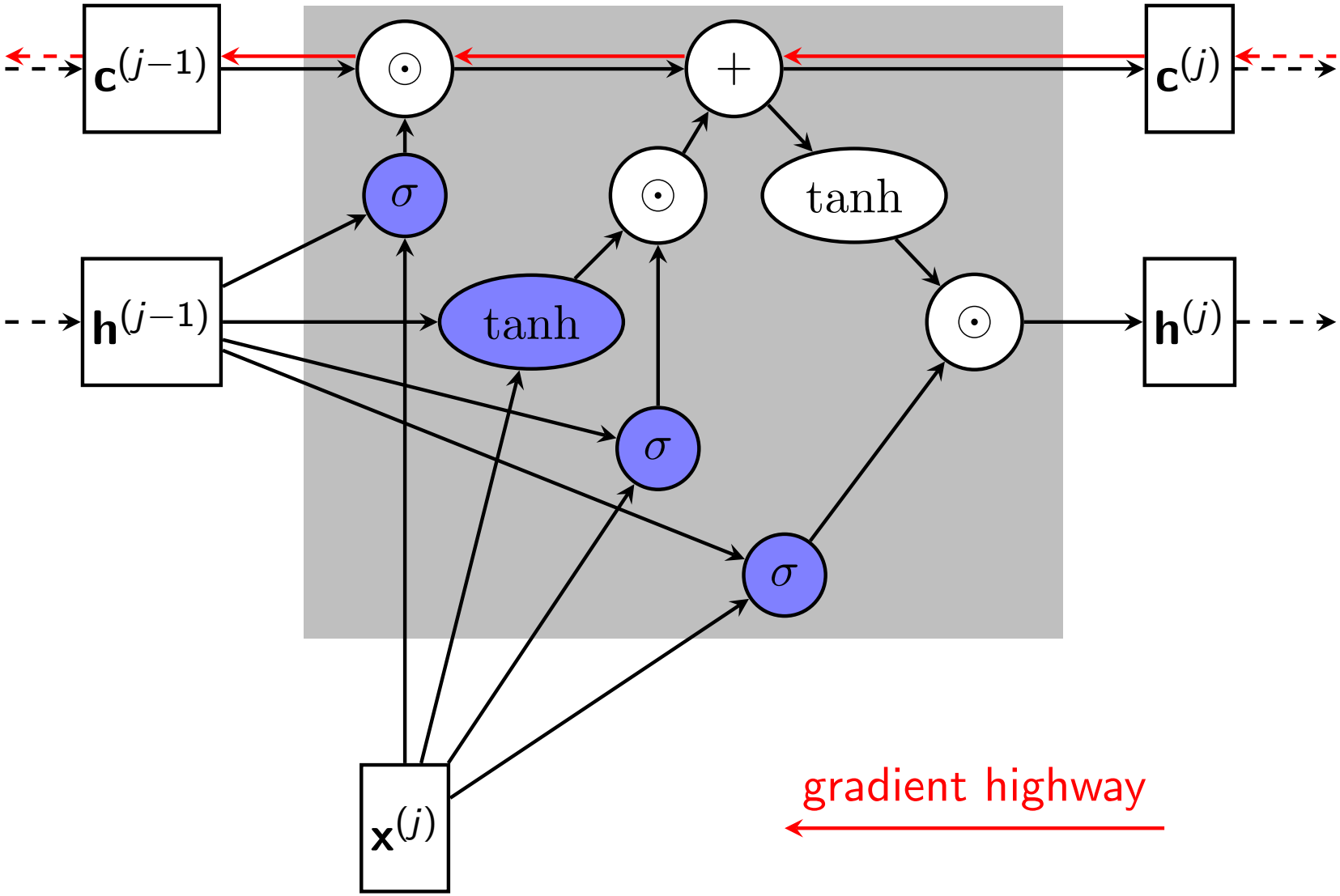
Long-Short Term Memory Network (LSTM)

- Two states: **h** ("short-term memory") and **c** ("long-term memory")
- Candidate state $\bar{\mathbf{h}} \in \mathbb{R}^d$ corresponds to **h** in the Vanilla RNN
- Interactions are mediated by "gates" $\in (0,1)^d$, which apply elementwise:
  - Forget gate **f** decides what information from **c** should be forgotten
  - Input gate **i** decides what information from $\bar{\mathbf{h}}$ should be added to **c**
  - Output gate **o** decides what information from **c** should be exposed to **h**
- Each gate and the candidate state have their own parameters $\theta^{(i)}, \theta^{(f)}, \theta^{(o)}, \theta^{(\bar{h})}$
- "Gradient highway" from $\mathbf{c}^{(j)}$ to $\mathbf{c}^{(j-1)}$, with no non-linearities or matrix multiplications

LSTM definition

$$
\begin{aligned}
\mathbf{h}^{(0)} &= \mathbf{c}^{(0)} = \{0\}^d \\
\mathbf{f}^{(j)} &= \sigma(\mathbf{W}^{(f)}\mathbf{x}^{(j)} + \mathbf{V}^{(f)}\mathbf{h}^{(j-1)} + \mathbf{b}^{(f)}) \\
\mathbf{i}^{(j)} &= \sigma(\mathbf{W}^{(i)}\mathbf{x}^{(j)} + \mathbf{V}^{(i)}\mathbf{h}^{(j-1)} + \mathbf{b}^{(i)}) \\
\mathbf{o}^{(j)} &= \sigma(\mathbf{W}^{(o)}\mathbf{x}^{(j)} + \mathbf{V}^{(o)}\mathbf{h}^{(j-1)} + \mathbf{b}^{(o)}) \\
\bar{\mathbf{h}}^{(j)} &= \tanh(\mathbf{W}^{(\bar{h})}\mathbf{x}^{(j)} + \mathbf{V}^{(\bar{h})}\mathbf{h}^{(j-1)} + \mathbf{b}^{(\bar{h})}) \\
\mathbf{c}^{(j)} &= \mathbf{f}^{(j)} \odot \mathbf{c}^{(j-1)} + \mathbf{i}^{(j)} \odot \bar{\mathbf{h}}^{(j)} \\
\mathbf{h}^{(j)} &= \mathbf{o}^{(j)} \odot \tanh(\mathbf{c}^{(j)})
\end{aligned}
$$

LSTM cell


$\mathbf{c}^{(j-1)}$
$\mathbf{c}^{(j)}$
$\mathbf{h}^{(j-1)}$
$\mathbf{h}^{(j)}$
$\mathbf{x}^{(j)}$
σ
σ
σ
tanh
tanh
gradient highway

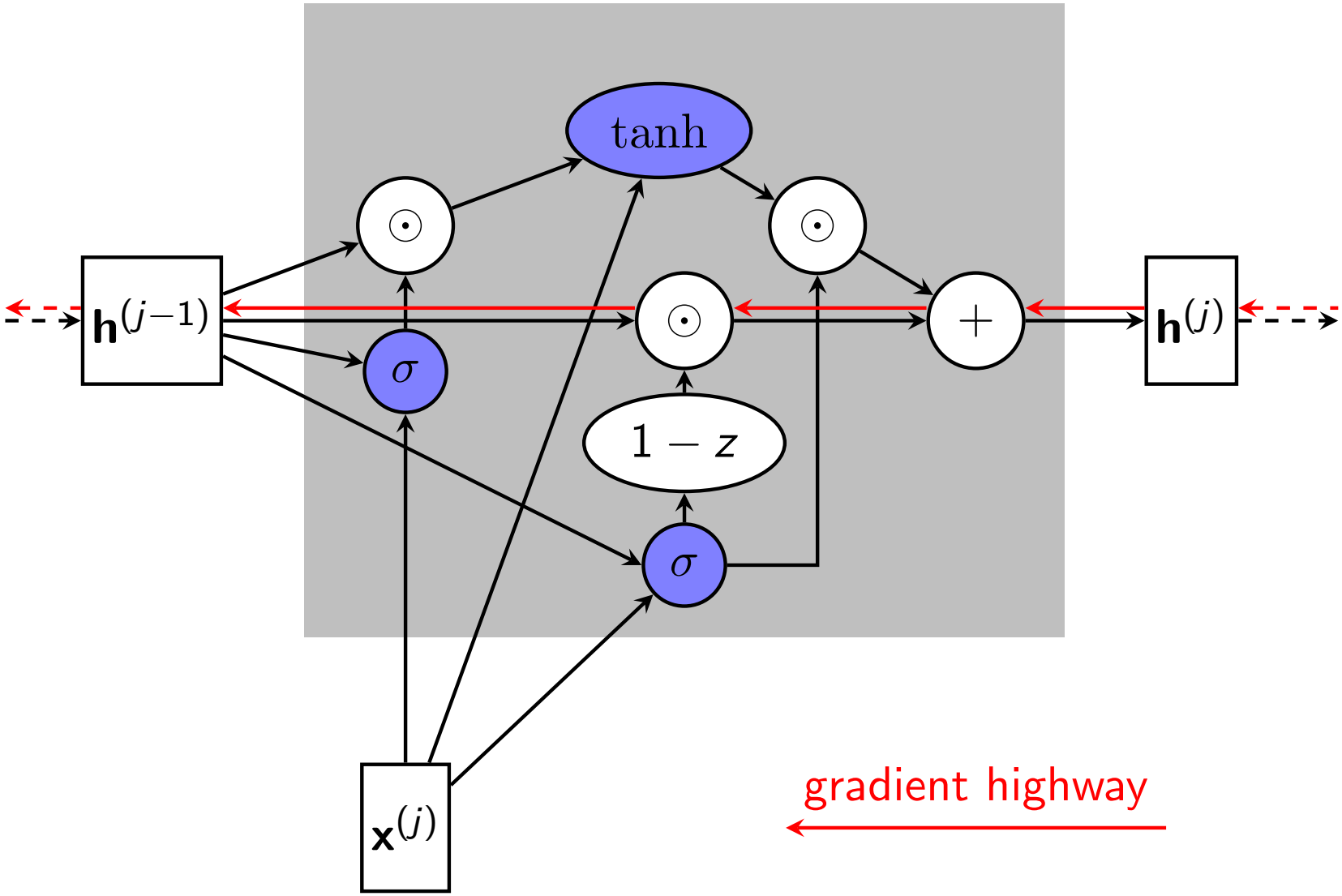
Gated Recurrent Unit (GRU)

- Proposed by Cho et al. (2014)
- Lightweight alternative to LSTM, with only one state and three sets of parameters
- State $\mathbf{h}$ is a dynamic "interpolation" of long and short term memory
- Reset gate $\mathbf{r} \in (0,1)^d$ controls what information passes from $\mathbf{h}$ to candidate state $\bar{\mathbf{h}}$
- Update gate $\mathbf{z} \in (0,1)^d$ interpolates between $\mathbf{h}$ and $\bar{\mathbf{h}}$
- Separate set of parameters $\theta^{(r)}, \theta^{(z)}, \theta^{(\bar{h})}$ for each gate and the candidate state.

GRU definition

$$
\begin{aligned}
\mathbf{h}^{(0)} &= \{0\}^d \\
\mathbf{r}^{(j)} &= \sigma(\mathbf{W}^{(r)}\mathbf{x}^{(j)} + \mathbf{V}^{(r)}\mathbf{h}^{(j-1)} + \mathbf{b}^{(r)}) \\
\mathbf{z}^{(j)} &= \sigma(\mathbf{W}^{(z)}\mathbf{x}^{(j)} + \mathbf{V}^{(z)}\mathbf{h}^{(j-1)} + \mathbf{b}^{(z)}) \\
\bar{\mathbf{h}}^{(j)} &= \tanh(\mathbf{W}^{(\bar{h})}\mathbf{x}^{(j)} + \mathbf{V}^{(\bar{h})}(\mathbf{r}^{(j)} \odot \mathbf{h}^{(j-1)}) + \mathbf{b}^{(\bar{h})}) \\
\mathbf{h}^{(j)} &= (1 - \mathbf{z}^{(j)}) \odot \mathbf{h}^{(j-1)} + \mathbf{z}^{(j)} \odot \bar{\mathbf{h}}^{(j)}
\end{aligned}
$$

GRU cell


tanh
$\mathbf{h}^{(j-1)}$
σ
$1-z$
σ
+
$\mathbf{h}^{(j)}$
$\mathbf{x}^{(j)}$
gradient highway

Vanilla vs. GRU vs. LSTM

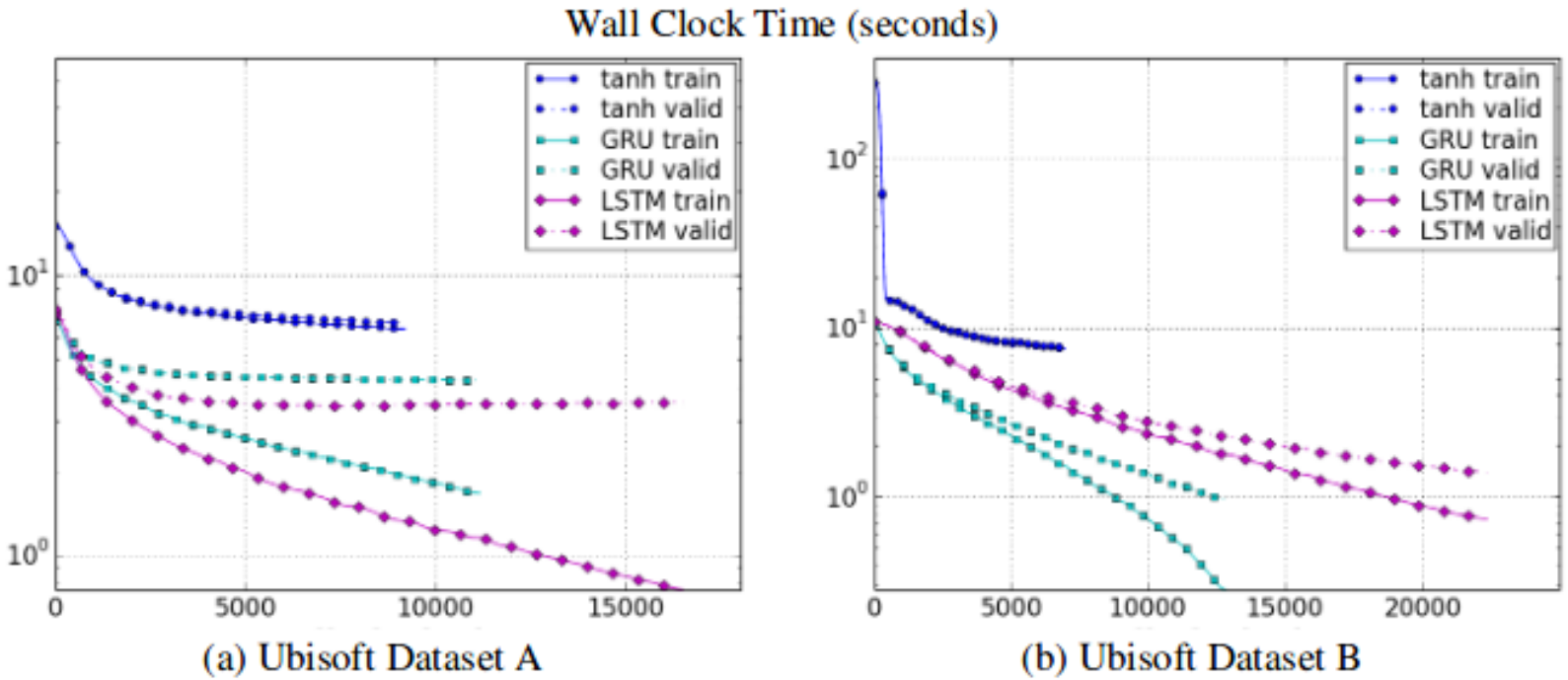


Figure from Chung et al. 2014: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

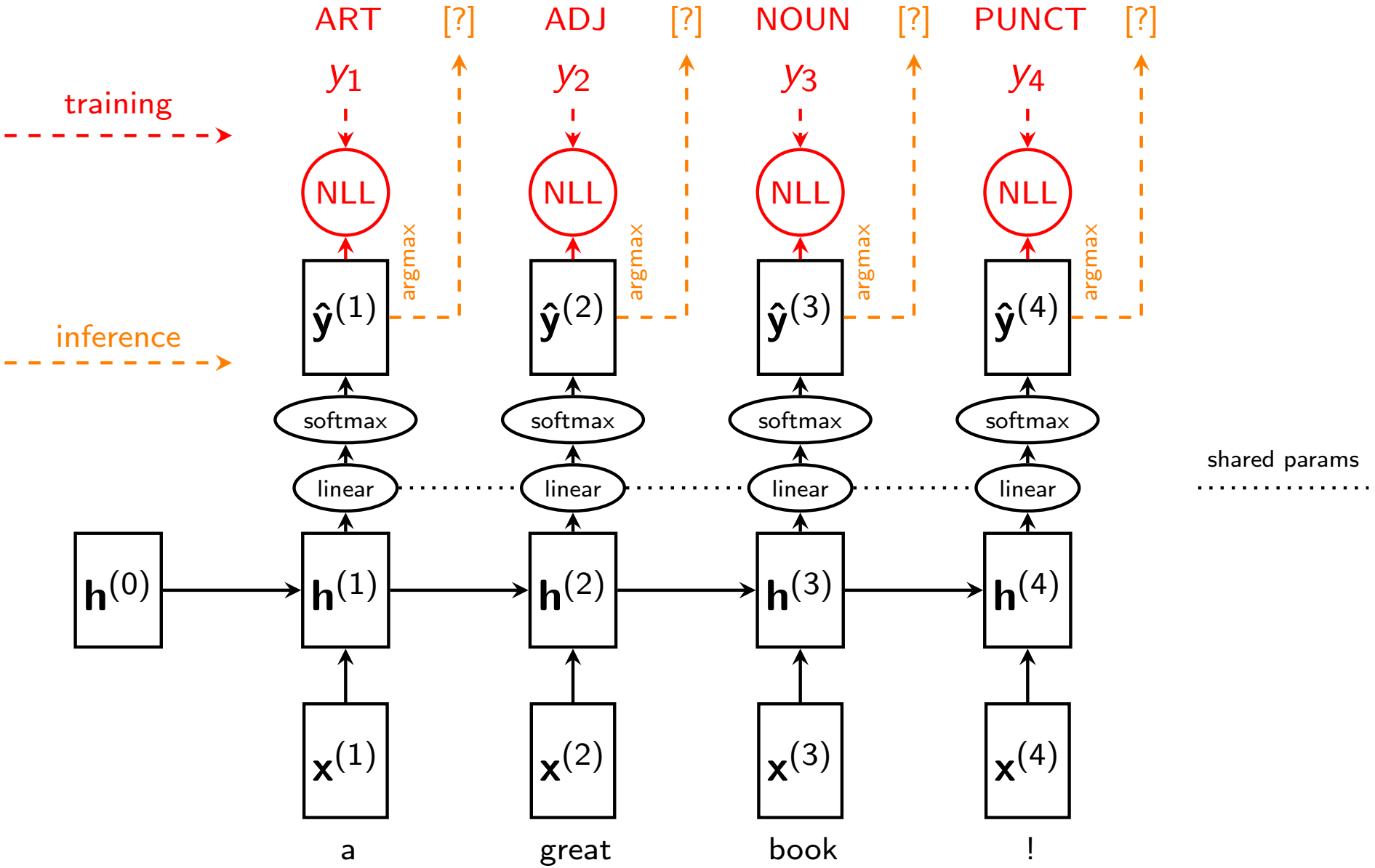
Questions?

Take a moment to write down any questions you have for the QA session!

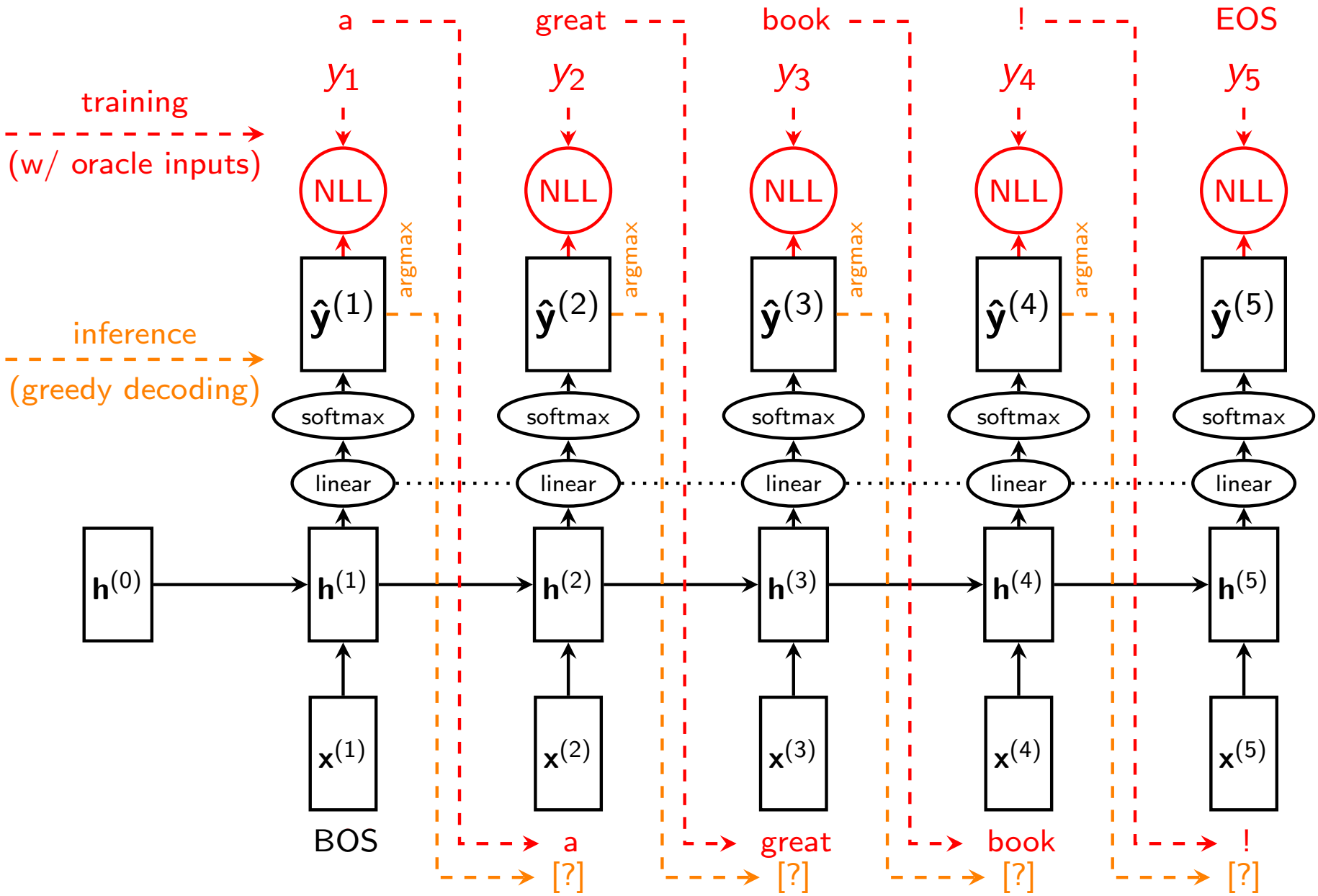
Outline

1. What are RNNs?
2. Vanilla RNN
3. Training RNNs
   - Backpropagation through time
   - The vanishing gradients problem
4. Gated RNNs
   - LSTM
   - GRU
   - Comparison
5. RNN applications
6. Extensions: Multi-layer RNNs, bidirectionality
7. RNNs with attention

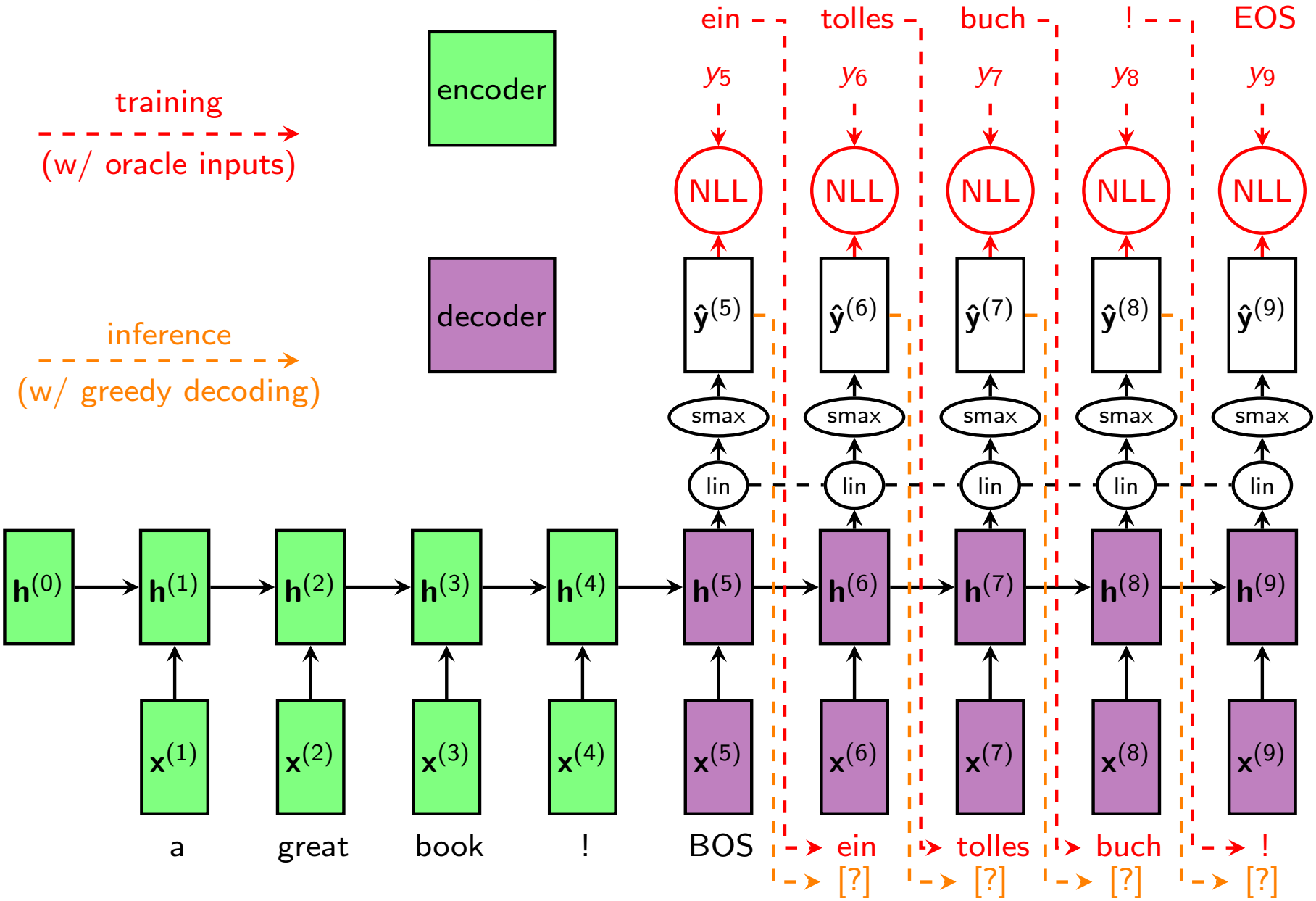
Tagging (example: Part-Of-Speech)



Autoregressive Language Modeling



Sequence-to-sequence (example: Machine Translation)



Questions?

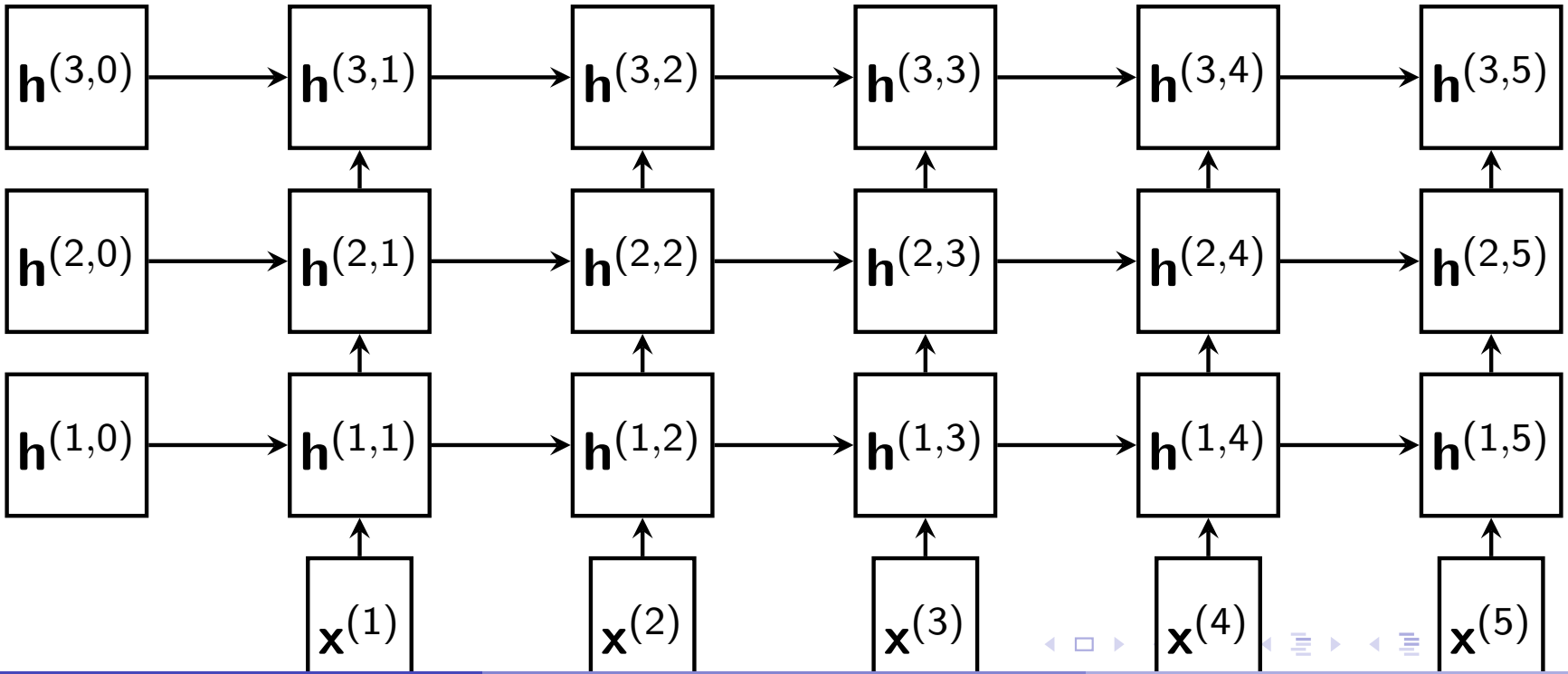
Take a moment to write down any questions you have for the QA session!

Outline

1. What are RNNs?
2. Vanilla RNN
3. Training RNNs
   - Backpropagation through time
   - The vanishing gradients problem
4. Gated RNNs
   - LSTM
   - GRU
   - Comparison
5. RNN applications
6. Extensions: Multi-layer RNNs, bidirectionality
7. RNNs with attention

Multi-Layer RNNs

- Stack of several RNNs (Vanilla RNNs, LSTMs, GRUs, etc.)
- Each RNN in the stack has its own parameters
- The input vectors of the l'th RNN are the hidden states of the $l-1$'th RNN
- The input vectors to the first RNN are the word embeddings, as usual
- We can output the hidden states of the last RNN, or a combination (concatenation, average, etc.) of the states of all RNNs.



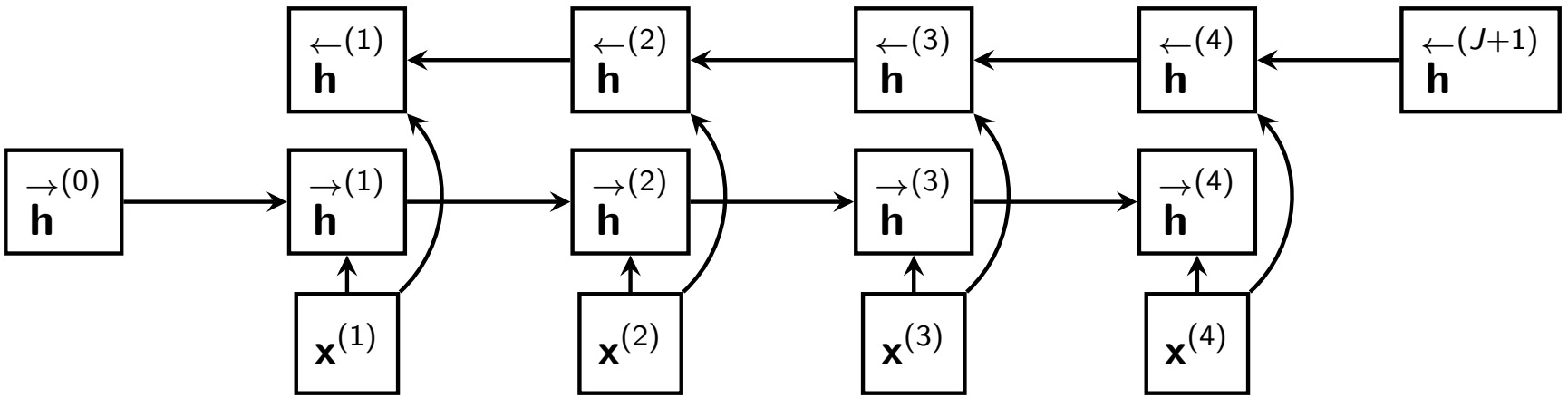
Bidirectional RNNs

- Two RNNs with separate parameters $\overrightarrow{\theta}, \overleftarrow{\theta}$
- Let $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(J)}$ be our input
- Let $\overrightarrow{\mathbf{h}}^{(0)} = \overleftarrow{\mathbf{h}}^{(J+1)} = \{0\}^d$ be our initial states.
- The forward RNN runs left-to-right over the input:

$$\overrightarrow{\mathbf{h}}^{(j)} = f(\mathbf{x}^{(j)}, \overrightarrow{\mathbf{h}}^{(j-1)}; \overrightarrow{\theta})$$

- The backward RNN runs right-to-left over the input:

$$\overleftarrow{\mathbf{h}}^{(j)} = f(\mathbf{x}^{(j)}, \overleftarrow{\mathbf{h}}^{(j+1)}; \overleftarrow{\theta})$$



Bidirectional RNNs

- The bidirectional RNN yields two sequences of hidden states:

$$(\overrightarrow{\mathbf{h}}^{(1)} \ldots \overrightarrow{\mathbf{h}}^{(J)}), (\overleftarrow{\mathbf{h}}^{(1)} \ldots \overleftarrow{\mathbf{h}}^{(J)})$$

- Question: If we are dealing with a sentence classification task, which states should we use to represent the sentence?
  - Concatenate $[\overrightarrow{\mathbf{h}}^{(J)}; \overleftarrow{\mathbf{h}}^{(1)}]$, because they have "seen" the entire sentence
- For tagging task, represent the j'th word as $[\overrightarrow{\mathbf{h}}^{(j)}; \overleftarrow{\mathbf{h}}^{(j)}]$
- Question: Can we use a bidirectional RNN for autoregressive language modeling?
  - No. In autoregressive language modeling, future inputs must be unknown to the model (since we want to learn to predict them).
  - We could train two separate autoregressive RNNs (one per direction), but we cannot combine their hidden states before making a prediction
- In sequence-to-sequence (e.g., Machine Translation), the encoder can be bidirectional, but the decoder cannot (same reason)

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1. What are RNNs?
2. Vanilla RNN
3. Training RNNs
   - Backpropagation through time
   - The vanishing gradients problem
4. Gated RNNs
   - LSTM
   - GRU
   - Comparison
5. RNN applications
6. Extensions: Multi-layer RNNs, bidirectionality
7. RNNs with attention

Limitations of RNNs

- At a given point in time j, the information about all past inputs $x^{(1)} \dots x^{(j)}$ is "crammed" into the state $\mathbf{h}^{(j)}$ (and $\mathbf{c}^{(j)}$ for an LSTM)
- So for long sequences, the state becomes a bottleneck
- For Machine Translation, this means that the source sentence is read exactly once, condensed into a single vector, and then the target sentence is produced.
- Question: Is this how you would translate a sentence?
  - A human translator might look at the source sentence multiple times, and translate it "bit by bit"
  - Attention is meant to mimick this process
- Proposed by Bahdanau et al. 2015, developed into stand-alone architecture ("Transformer") by Vaswani et al. 2017
- Currently the most popular architecture for NLP
- This week: Attention as a way to make RNNs better
- Next week: Transformers

Attention: The basic recipe

- **Ingredients:**

- One query vector: $\mathbf{q} \in \mathbb{R}^{d_q}$
- J key vectors: $\mathbf{K} \in \mathbb{R}^{J \times d_k}; (\mathbf{k}_1 \dots \mathbf{k}_J)$
- J value vectors: $\mathbf{V} \in \mathbb{R}^{J \times d_v}; (\mathbf{v}_1 \dots \mathbf{v}_J)$
- Scoring function $a : \mathbb{R}^{d_q} \times \mathbb{R}^{d_k} \rightarrow \mathbb{R}$
  - Maps a query-key pair to a scalar ("score")
  - a may be parametrized by parameters θ_a

Attention: The basic recipe

- **Step 1**: Apply a to $\mathbf{q}$ and all keys $\mathbf{k}_j$ to get scores (one per key):

$$\mathbf{e} = \begin{bmatrix} e_1 \\ \vdots \\ e_J \end{bmatrix} = \begin{bmatrix} a(\mathbf{q}, \mathbf{k}_1) \\ \vdots \\ a(\mathbf{q}, \mathbf{k}_J) \end{bmatrix}$$

- **Step 2**: Turn $\mathbf{e}$ into probability distribution with the softmax function

$$\alpha_j = \frac{\exp(e_j)}{\sum_{j'=1}^{J} \exp(e_{j'})}$$

- **Step 3**: α-weighted sum over $\mathbf{V}$ yields one $\mathbb{R}^{d_v}$-dimensional output vector $\mathbf{o}$:

$$\mathbf{o} = \sum_{j=1}^{J} \alpha_j \mathbf{v}_j$$

Attention in Bahdanau et al., 2015

- Source sentence: $(\mathbf{x}_1 \dots \mathbf{x}_{T_x})$, encoded as hidden states $(\mathbf{h}_1 \dots \mathbf{h}_{T_x})$ by encoder RNN
- Decoder is defined as:

The hidden state s_i of the decoder given the annotations from the encoder is computed by

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i,$$

where

$$\begin{aligned}
\tilde{s}_i &= \tanh\left(WEy_{i-1} + U\left[r_i \circ s_{i-1}\right] + Cc_i\right) \\
z_i &= \sigma\left(W_z Ey_{i-1} + U_z s_{i-1} + C_z c_i\right) \\
r_i &= \sigma\left(W_r Ey_{i-1} + U_r s_{i-1} + C_r c_i\right)
\end{aligned}$$

From Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate

- Exercise: Recognize the architecture (hint: they don't use boldface)
  - This is a GRU
  - s_i are the states (which we called $\mathbf{h}^{(i)}$)
  - $E_{y_{i-1}}$ is the embedding of the word that was/should have been predicted at the previous time step (oracle or decoded input)
  - W_* and U_* are parameter matrices (which we called $\mathbf{W}^{(*)}$ and $\mathbf{V}^{(*)}$), and they drop the bias
  - C_* seem to be additional parameter matrices ... but where does the vector c_i come from?

Attention in Bahdanau et al., 2015

The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad (5)$$

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

where

$$e_{ij} = a(s_{i-1}, h_j)$$

From Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate

- Each equation corresponds to one step from our attention recipe. Exercise: Figure out which equation is which step.
  - The first equation is step 3. It defines the output vector c_i (**o** in our recipe) as an attention-weighted sum over encoder states. So the encoder states h_j are our value vectors ($\mathbf{v}_j$ in our recipe).
  - The second equation is step 2.
  - The third equation is step 1, which defines the raw scores.
    - s_{i-1} (previous decoder state) is the query vector (**q** in our recipe)
    - The encoder states h_j are also our key vectors (so $\mathbf{k}_j = \mathbf{v}_j$ in this case)

Attention in Bahdanau et al., 2015

- Scoring function is a Feed-Forward Net:

$$a(s_{i-1}, h_j) = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right),$$

Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate (appendix)

What does attention learn?

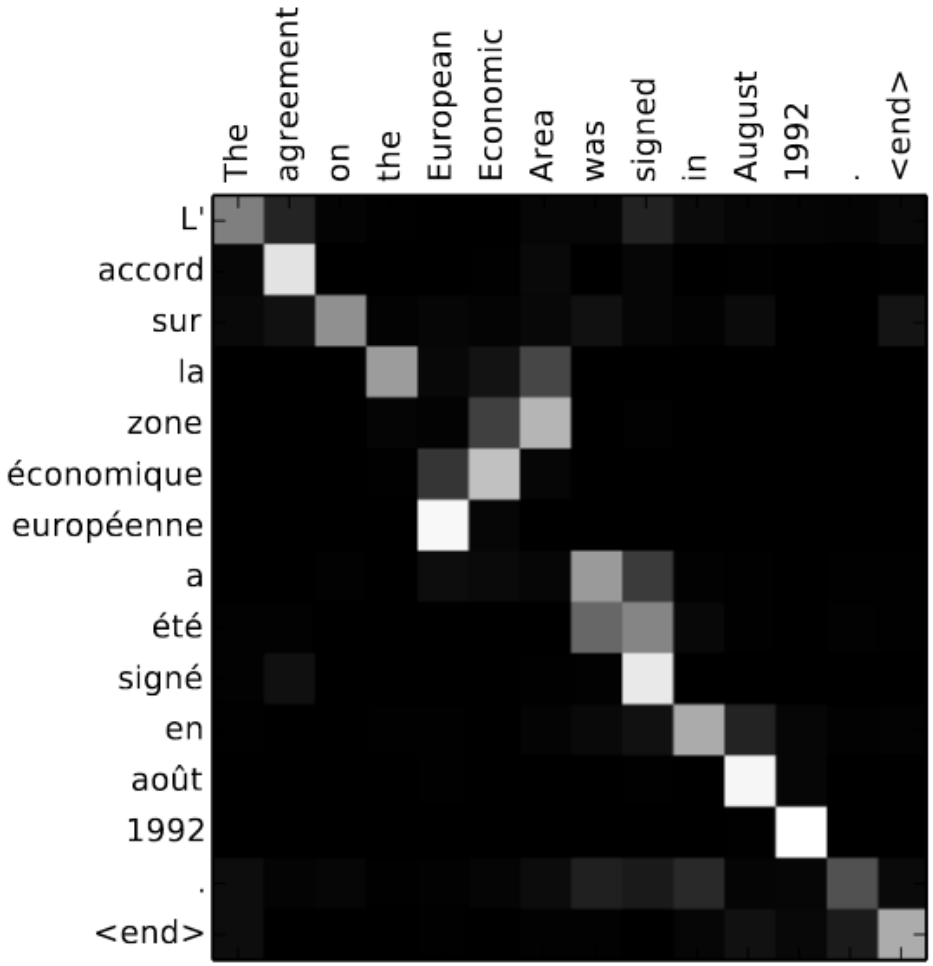


Figure from Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate.

Effect on translation quality for long sentences

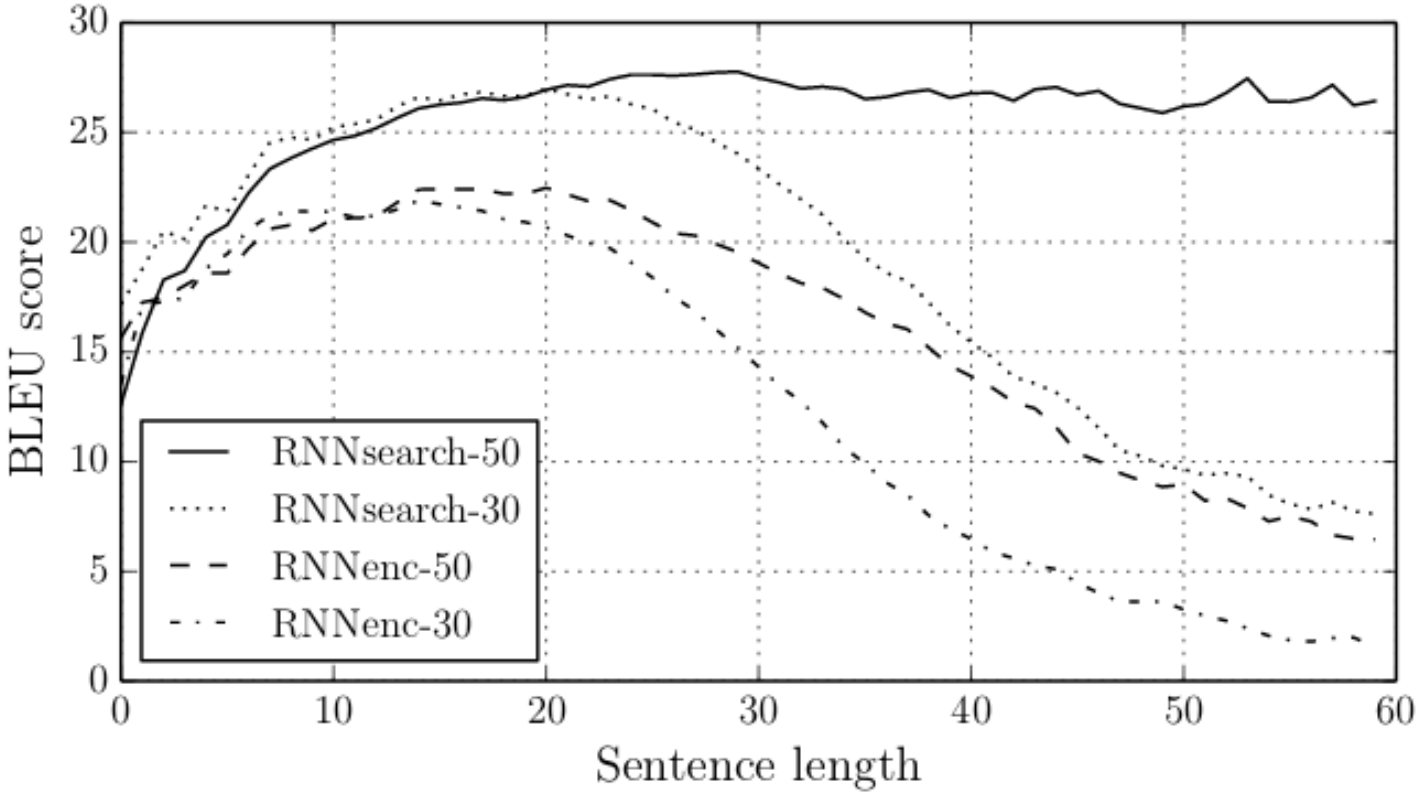


Figure from Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate.

- RNNsearch-30 and RNNsearch-50 are the attention models
- RNNenc-30 and RNNenc-50 are the baseline models without attention
- 30 and 50 are the state dimensionalities

- Important to note: The Bahdanau model is still an RNN, just with attention on top.
- Next week, we get rid of the RNN completely and introduce the attention-only Transformer architecture.

Questions?

Take a moment to write down any questions you have for the QA session!