

ST449 Artificial Intelligence and Deep Learning

Lecture 5

Sequence modeling



Milan Vojnovic

<https://github.com/lse-st449/lectures>

Topics of this lecture

- Recurrent neural networks
- Exploding and vanishing gradients
- Vector to sequence models
- Bidirectional recurrent neural networks
- Long short-term memory networks
- Gated recurrent units
- Sequence transduction models
- Encoder-decoder architecture
 - w and w/o alignment
- Transformer (attention) networks

Recurrent neural networks

Recurrent neural networks overview

- **Recurrent neural networks (RNNs)**: a family of neural networks for sequential data
 - Specialized for processing a sequence of vectors $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(T)}$
- Most RNNs can process sequences of **variable length**
- Key idea: **parameter sharing**
 - Parameter sharing across different parts of the model
- Similarities to convolutional neural networks:
 - Convolutional neural networks also use parameter sharing
 - Use of convolution for 1-dimensional temporal sequences

Unfolding computational graphs

- **Computational graph**: defines application of a set of computations
 - Mapping of inputs to outputs
 - Mapping parameters to a loss function value
- Consider a **non-linear dynamical system**, defined by the recursive equation:

$$\mathbf{h}^{(t+1)} = f_{\theta}(\mathbf{h}^{(t)})$$

where $\mathbf{h}^{(t)}$ is the state of the system, f_{θ} is a given function with θ parameter

- This is an **autonomous system** because the mapping f_{θ} does not depend on an independent variable (ex. time t)
- Unfolded equation:

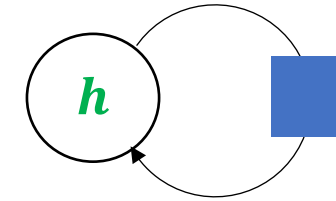
$$\mathbf{h}^{(1)} = f_{\theta}(\mathbf{h}^{(0)}), \mathbf{h}^{(2)} = f_{\theta} \circ f_{\theta}(\mathbf{h}^{(0)}), \dots, \mathbf{h}^{(t)} = f_{\theta} \circ \dots \circ f_{\theta}(\mathbf{h}^{(0)})$$

Unfolding computational graphs (cont'd)

- Recurrence equation:

$$\mathbf{h}^{(t+1)} = f_{\theta}(\mathbf{h}^{(t)})$$

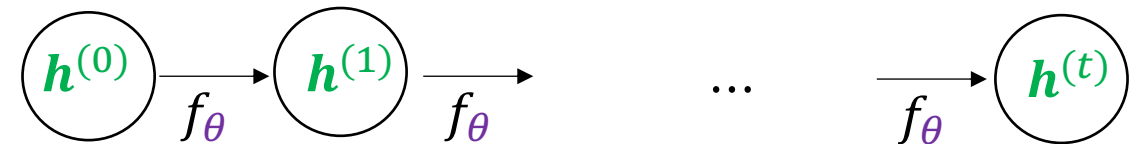
Recurrent graph



- Unfolded representation:

$$\mathbf{h}^{(t)} = f_{\theta} \circ \dots \circ f_{\theta}(\mathbf{h}^{(0)})$$

Unfolded graph



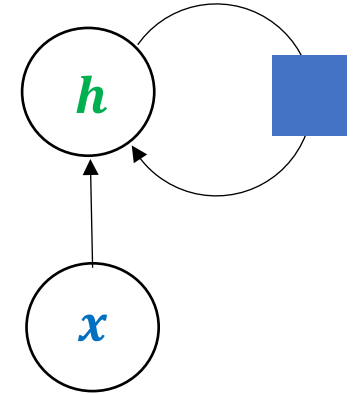
Unfolding computational graphs (cont'd)

- Non-autonomous dynamical system:

$$\mathbf{h}^{(t+1)} = f_{\theta}(\mathbf{h}^{(t)}, \mathbf{x}^{(t+1)})$$

Input sequence
“driving sequence”

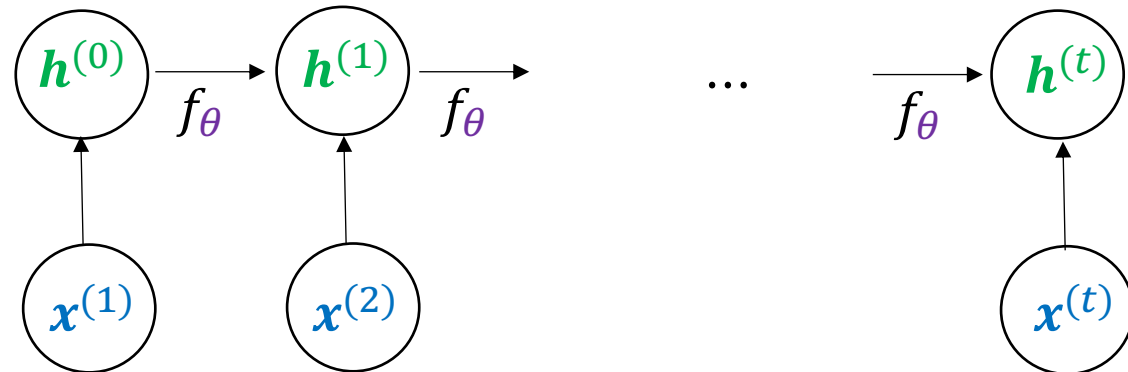
Recurrent graph



- Unfolded representation:

$$\mathbf{h}^{(t)} = f_{\theta}(f_{\theta}(\cdots f_{\theta}(\mathbf{h}^{(0)}, \mathbf{x}^{(1)}) \cdots, \mathbf{x}^{(t-1)}), \mathbf{x}^{(t)})$$

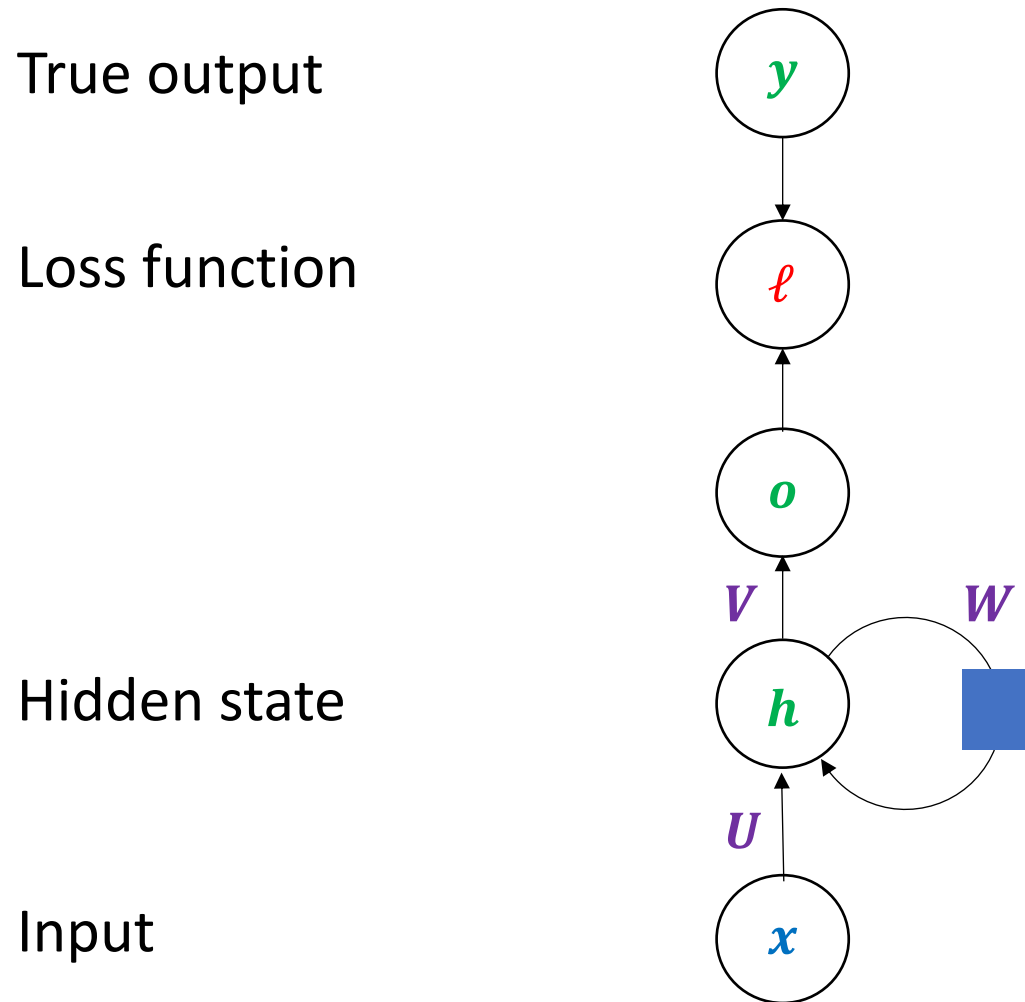
Unfolded graph



Unfolding computational graphs (cont'd)

- Recurrent graphs
 - Provide succinct representations
- Unfolded graphs
 - Provide explicit description of which computations to perform
 - Show paths along which information flows
 - Forward in time: for computing outputs and loss function
 - Backward in time: for computing gradients

Common RNN architecture: A



$$\hat{y}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

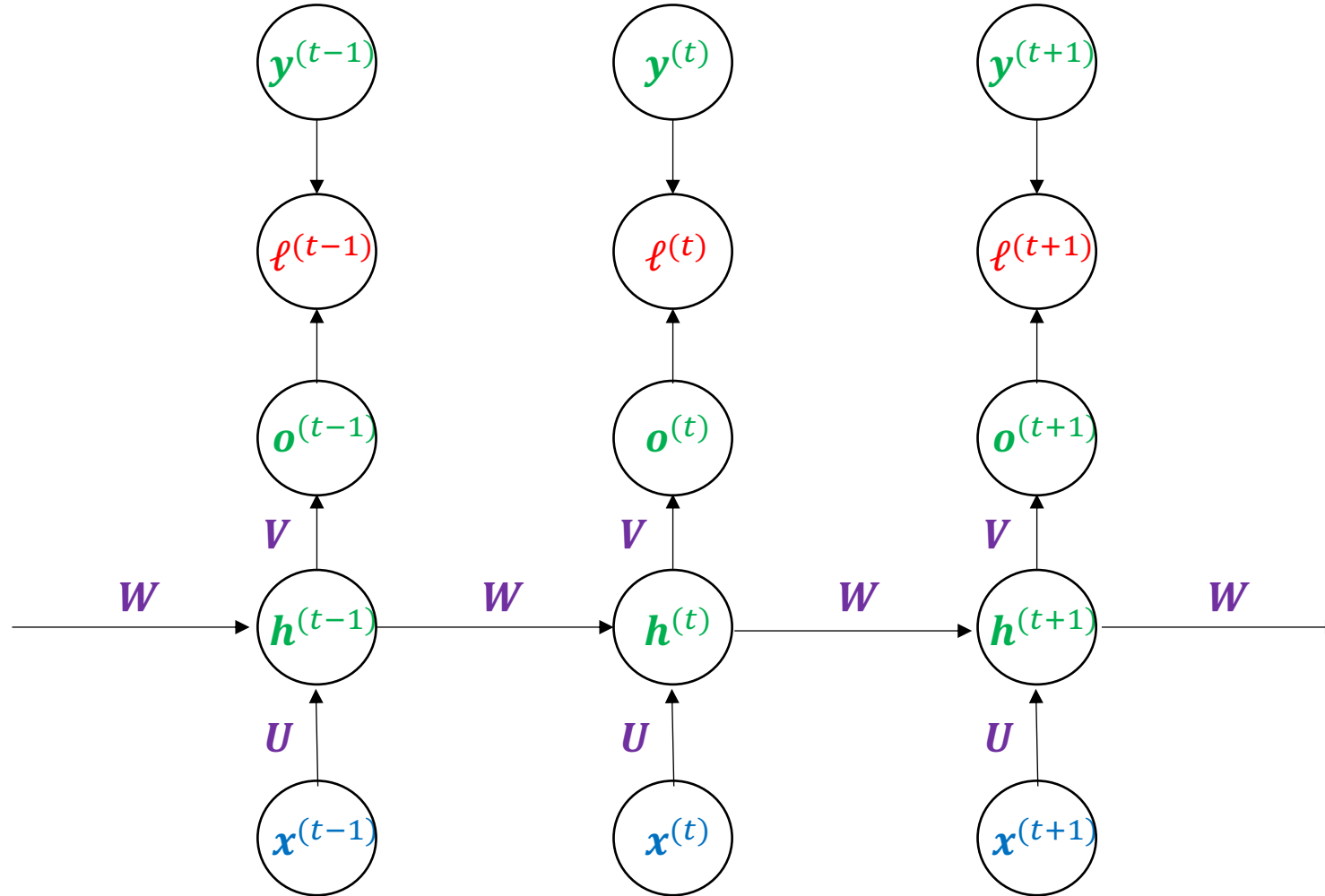
$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$

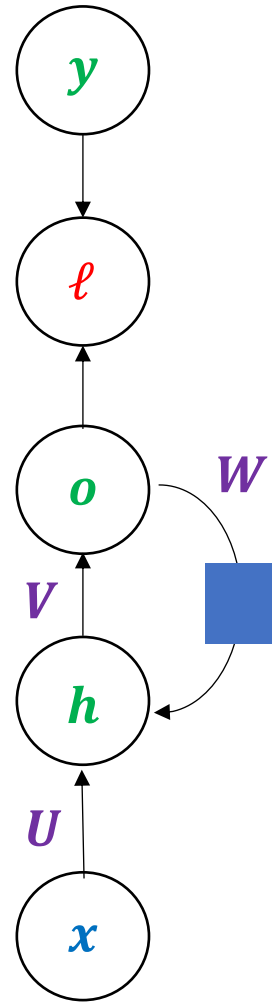
$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$

- Output: each time step
- Recurrent connections: between hidden units at successive time steps

Common RNN architecture: A (unfolded)



Common RNN architecture: B



$$\hat{y}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

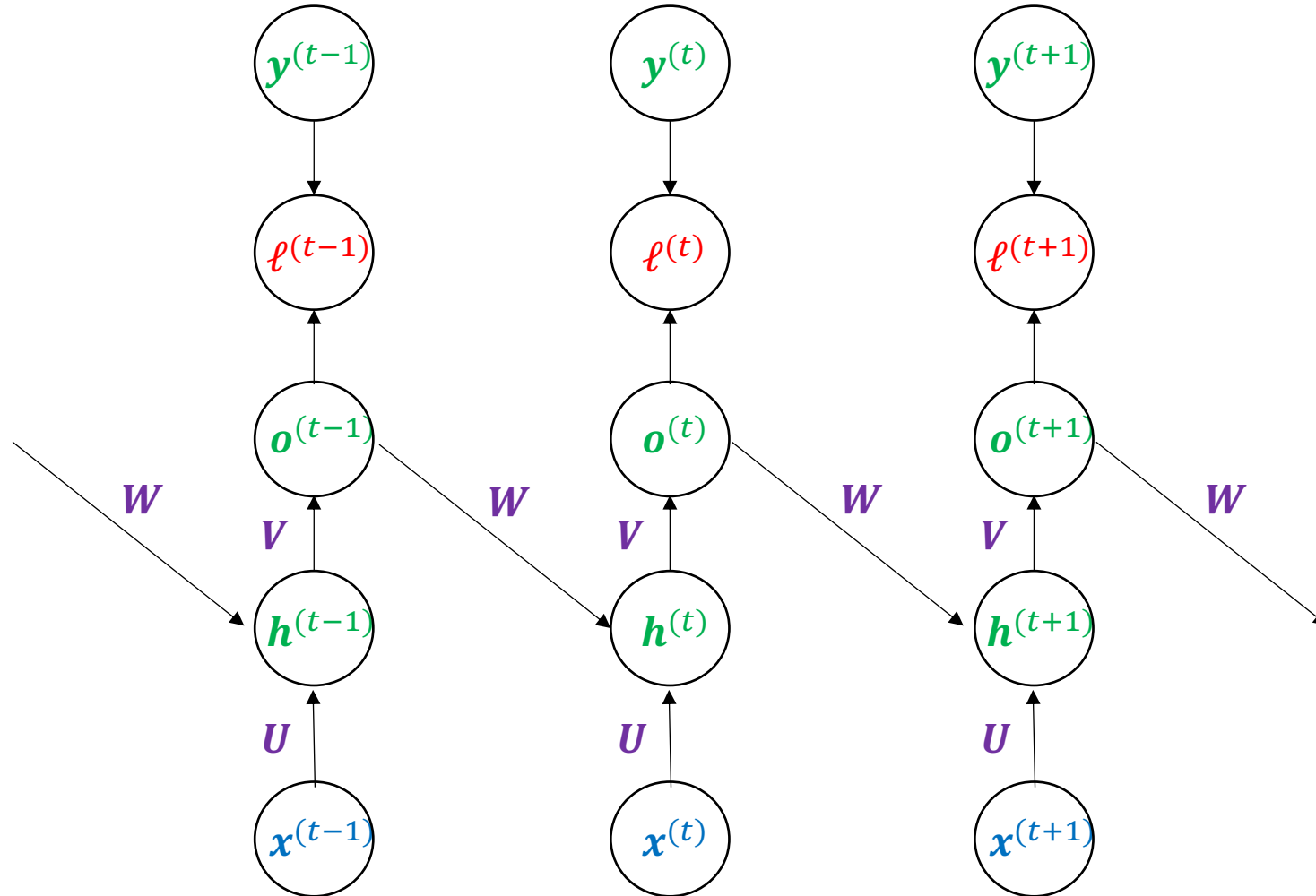
$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$

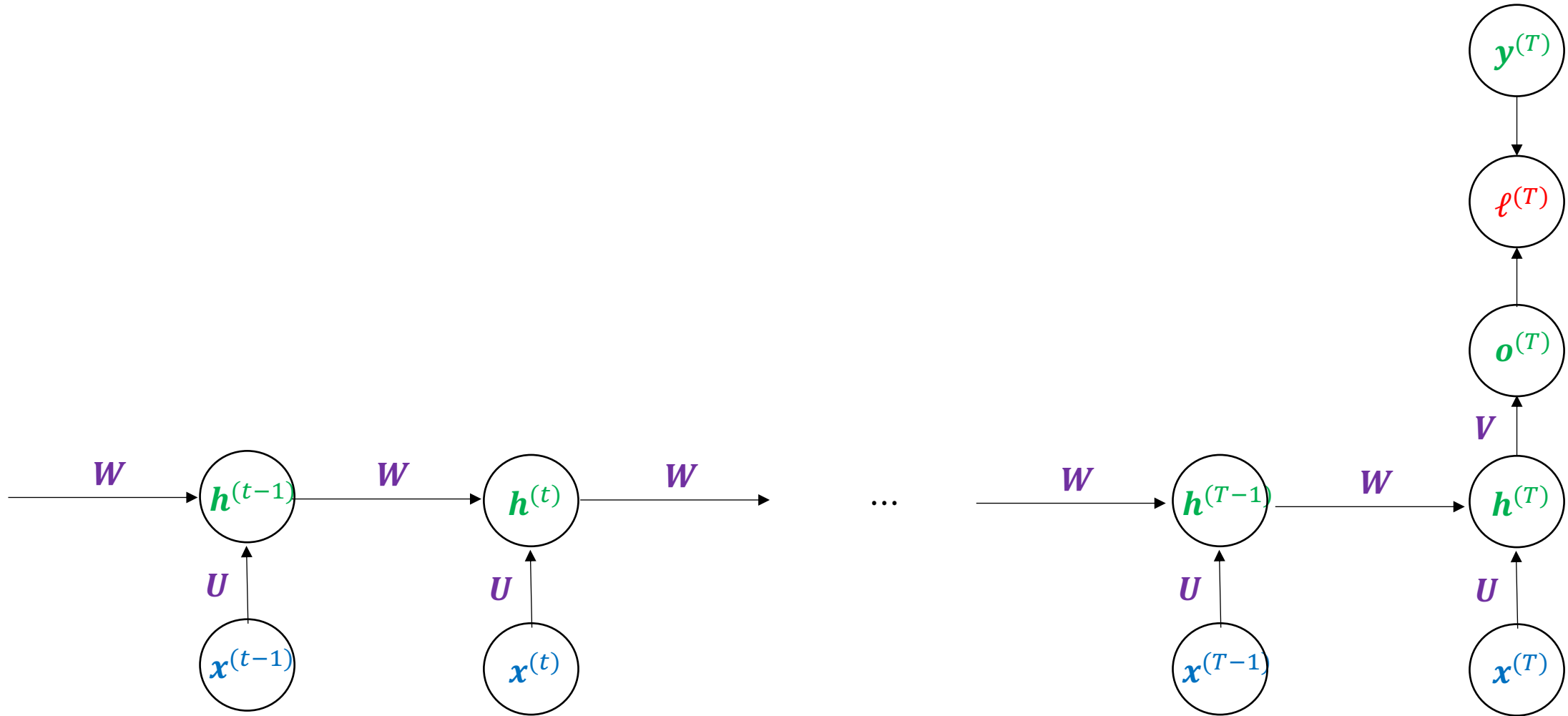
$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$

- Output: each time step
- Recurrent connections: from output at a time step to the hidden unit at next time step

Common RNN architecture: B (unfolded)



Common RNN architectures: C



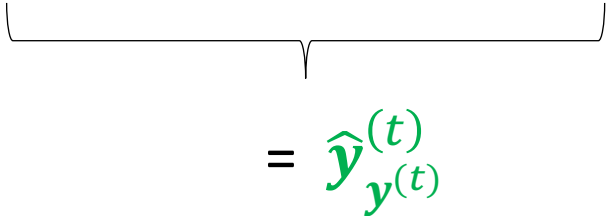
- Output: **after** an input sequence is read
- Recurrent connections: **between hidden units**

Loss function: negative log-likelihood

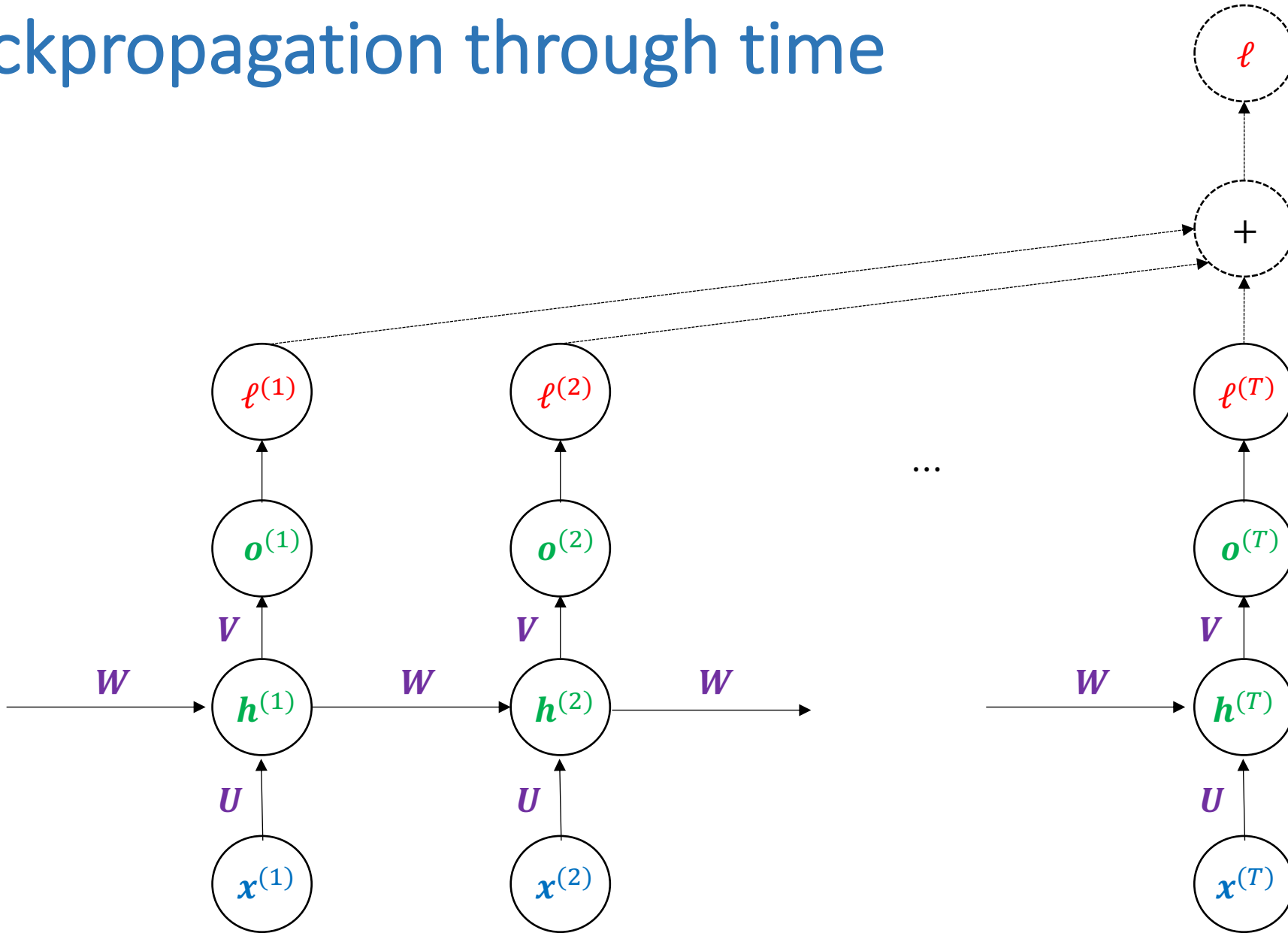
- Negative log-likelihood function:

$$\ell((\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}), (\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(T)})) = \sum_{t=1}^T \ell^{(t)}$$

where

$$\ell^{(t)} = -\log p_{\theta}(\mathbf{y}^{(t)} \mid \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)})$$

$$= \hat{\mathbf{y}}_{\mathbf{y}^{(t)}}^{(t)}$$

Backpropagation through time



Backpropagation through time (cont'd)

- Start with nodes corresponding to loss functions $\ell^{(t)}$:

$$J_{\ell^{(t)}}(\ell) = 1$$

- For nodes corresponding to output functions $\mathbf{o}^{(t)}$:

$$J_{\mathbf{o}^{(t)}}(\ell) = J_{\ell^{(t)}}(\ell) J_{\mathbf{o}^{(t)}}(\ell^{(t)}) = \hat{\mathbf{y}}^{(t)} - \mathbf{e}_{\mathbf{y}^{(t)}}$$

\mathbf{e}_i := vector with the i-th element equal to 1 and other elements 0

- For nodes corresponding to hidden state $\mathbf{h}^{(t)}$:

$$J_{\mathbf{h}^{(T)}}(\ell) = \mathbf{V}^\top J_{\mathbf{o}^{(T)}}(\ell)$$

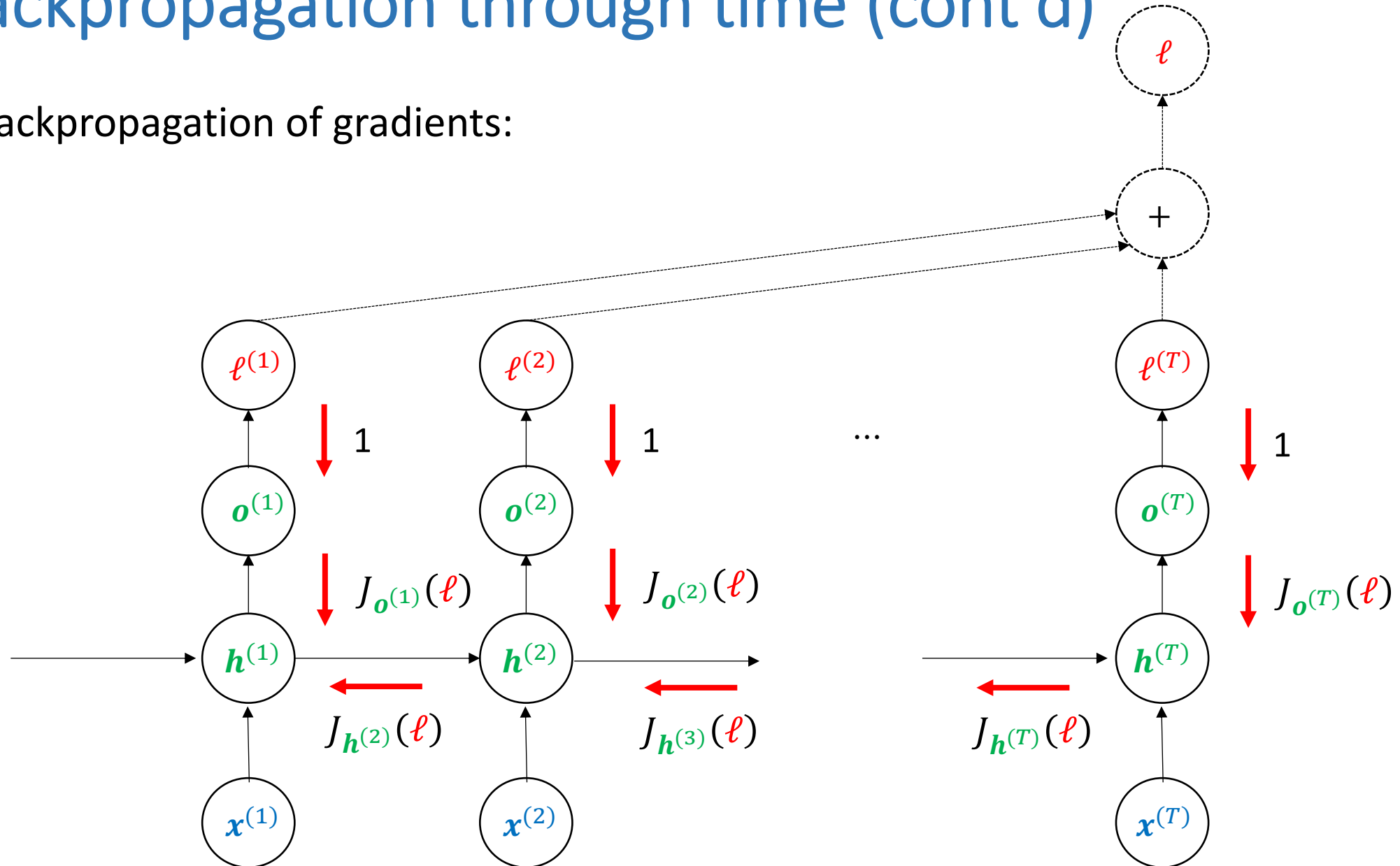
$$J_{\mathbf{h}^{(t)}}(\ell) = J_{\mathbf{h}^{(t)}}(\mathbf{h}^{(t+1)}) J_{\mathbf{h}^{(t+1)}}(\ell) + J_{\mathbf{h}^{(t)}}(\mathbf{o}^{(t)}) J_{\mathbf{o}^{(t)}}(\ell) \text{ for } 1 \leq t < T$$

$$\underbrace{J_{\mathbf{h}^{(t)}}(\mathbf{h}^{(t+1)}) J_{\mathbf{h}^{(t+1)}}(\ell)}_{\mathbf{D}_{\mathbf{h}^{(t+1)}} \mathbf{W} J_{\mathbf{h}^{(t+1)}}(\ell)} + \underbrace{J_{\mathbf{h}^{(t)}}(\mathbf{o}^{(t)}) J_{\mathbf{o}^{(t)}}(\ell)}_{\mathbf{V}^\top}$$

$$\mathbf{D}_{\mathbf{h}^{(t+1)}} = \text{diag}\left(1 - \left(h_1^{(t+1)}\right)^2, \dots, 1 - \left(h_n^{(t+1)}\right)^2\right)$$

Backpropagation through time (cont'd)

- Backpropagation of gradients:



Backpropagation through time (cont'd)

- Gradients with respect to parameters:

$$J_{\mathbf{c}}(\ell) = \sum_{t=1}^T J_{\mathbf{c}}(\mathbf{o}^{(t)})^\top J_{\mathbf{o}^{(t)}}(\ell) = \sum_{t=1}^T J_{\mathbf{o}^{(t)}}(\ell)$$

$$J_{\mathbf{b}}(\ell) = \sum_{t=1}^T J_{\mathbf{b}}(\mathbf{h}^{(t)}) J_{\mathbf{h}^{(t)}}(\ell) = \sum_{t=1}^T \mathbf{D}_{\mathbf{h}^{(t)}} J_{\mathbf{h}^{(t)}}(\ell)$$

$$J_{\mathbf{V}}(\ell) = \sum_{t=1}^T J_{\mathbf{o}^{(t)}}(\ell) J_{\mathbf{V}}(\mathbf{o}^{(t)}) = \sum_{t=1}^T J_{\mathbf{o}^{(t)}}(\ell) (\mathbf{h}^{(t)})^\top$$

Exercise

- Show that

$$J_{\mathbf{o}^{(t)}}(\ell) = \hat{\mathbf{y}}^{(t)} - \mathbf{e}_{y^{(t)}}$$

- Show that

$$J_{\mathbf{h}^{(t)}}(\mathbf{h}^{(t+1)}) J_{\mathbf{h}^{(t+1)}}(\ell) = \mathbf{D}_{\mathbf{h}^{(t+1)}} \mathbf{W} J_{\mathbf{h}^{(t+1)}}(\ell)$$

Exercise solution

$$\bullet \frac{\partial \ell}{\partial \mathbf{o}_i^{(t)}} = \frac{\partial \ell}{\partial \ell^{(t)}} \frac{\partial \ell^{(t)}}{\partial \mathbf{o}_i^{(t)}} = \frac{\partial \ell^{(t)}}{\partial \mathbf{o}_i^{(t)}}$$

$$\bullet \frac{\partial \ell^{(t)}}{\partial \mathbf{o}_i^{(t)}} = \frac{\partial}{\partial \mathbf{o}_i^{(t)}} \left(-\log \left(\frac{e^{\mathbf{o}_{y^{(t)}}^{(t)}}}{\sum_s e^{\mathbf{o}_s^{(t)}}} \right) \right) = -1_{y^{(t)}=i} + \frac{e^{\mathbf{o}_i^{(t)}}}{\sum_s e^{\mathbf{o}_s^{(t)}}} = -1_{y^{(t)}=i} + \hat{\mathbf{y}}_i^{(t)}$$

$$\Rightarrow \frac{\partial \ell}{\partial \mathbf{o}_i^{(t)}} = \hat{\mathbf{y}}_i^{(t)} - 1_{y^{(t)}=i}$$

or, equivalently,

$$J_{\mathbf{o}^{(t)}}(\ell) = \hat{\mathbf{y}}^{(t)} - \mathbf{e}_{y^{(t)}}$$

Exercise solution (cont'd)

- Basic facts: $y = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and $\tanh'(x) = 1 - y^2$

- $$J_{\mathbf{h}^{(t)}}(\mathbf{h}^{(t+1)})_{i,j} = \frac{\partial \mathbf{h}_i^{(t+1)}}{\partial \mathbf{h}_j^{(t)}} = \frac{\partial}{\partial \mathbf{h}_j^{(t)}} \tanh(\mathbf{a}_i^{(t+1)}) = \tanh'(\mathbf{a}_i^{(t+1)}) \mathbf{W}_{i,j} = \left(1 - \left(\mathbf{h}_i^{(t+1)}\right)^2\right) \mathbf{W}_{i,j}$$

where $\mathbf{a}^{(t+1)} = \mathbf{W}\mathbf{h}^{(t)} + \mathbf{U}\mathbf{x}^{(t+1)} + \mathbf{b}$

- $$\begin{aligned} & \left(J_{\mathbf{h}^{(t)}}(\mathbf{h}^{(t+1)}) J_{\mathbf{h}^{(t+1)}}(\ell) \right)_i = \sum_j J_{\mathbf{h}^{(t)}}(\mathbf{h}^{(t+1)})_{i,j} J_{\mathbf{h}^{(t+1)}}(\ell)_j \\ &= \sum_j \frac{\partial \mathbf{h}_i^{(t+1)}}{\partial \mathbf{h}_j^{(t)}} J_{\mathbf{h}^{(t+1)}}(\ell)_j \\ &= \sum_j \left(1 - \left(\mathbf{h}_i^{(t+1)} \right)^2 \right) \mathbf{W}_{i,j} J_{\mathbf{h}^{(t+1)}}(\ell)_j \\ &= \sum_j \left(\mathbf{D}_{\mathbf{h}^{(t+1)}} \mathbf{W} \right)_{i,j} J_{\mathbf{h}^{(t+1)}}(\ell)_j \\ &\Rightarrow J_{\mathbf{h}^{(t)}}(\mathbf{h}^{(t+1)}) J_{\mathbf{h}^{(t+1)}}(\ell) = \mathbf{D}_{\mathbf{h}^{(t+1)}} \mathbf{W} J_{\mathbf{h}^{(t+1)}}(\ell) \end{aligned}$$

Exploding and vanishing gradients

- RNNs are known to have an issue with exploding and vanishing gradients
 - **Exploding gradients**: increase of the norm of the gradient during training
 - **Vanishing gradients**: vanishing long term contributions

- We explain for an RNN with update equations:

$$\mathbf{h}^{(t)} = \mathbf{W}a(\mathbf{h}^{(t-1)}) + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$

and the loss function

$$\ell = \sum_{t=1}^T \ell^{(t)}$$

- Note: we can consider $\mathbf{s}^{(t)} = a(\mathbf{W}\mathbf{s}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b})$ by using $\mathbf{s}^{(t)} := a(\mathbf{h}^{(t)})$

Exploding and vanishing gradients (cont'd)

- Let θ denote any of the model parameters W, U, b
- The Jacobian of the loss function is given by

$$J_{\theta}(\ell) = \sum_{t=1}^T J_{\theta}(\ell^{(t)})$$

and

$$J_{\theta}(\ell^{(t)}) = J_{\mathbf{h}^{(t)}}(\ell^{(t)}) J_{\theta}(\mathbf{h}^{(t)}) = J_{\mathbf{h}^{(t)}}(\ell^{(t)}) \underbrace{\sum_{s=1}^t J_{\mathbf{h}^{(s)}}(\mathbf{h}^{(t)}) J_{\theta}(\mathbf{h}^{(s)})}_{\text{contributions:}}$$

- By the chain rule

$$J_{\mathbf{h}^{(s)}}(\mathbf{h}^{(t)}) = \prod_{r=s+1}^t J_{\mathbf{h}^{(r-1)}}(\mathbf{h}^{(r)}) = \prod_{r=s+1}^t W \text{diag}(a'(\mathbf{h}^{(r-1)}))$$

contributions: long term $s \ll t$
short term $s \approx t$

Special case: a identity mapping

- For a being the identity mapping, we have

$$J_{\mathbf{h}^{(s)}}(\mathbf{h}^{(t)}) = \mathbf{W}^{t-s}$$

- If the spectral radius $\rho(\mathbf{W})$ of \mathbf{W} is such that

$$\rho(\mathbf{W}) < 1$$

then the long term components vanish

- Condition $\rho(\mathbf{W}) > 1$ is necessary for exploding gradients

General case

- Assume that a is such that $\|\text{diag}(a'(\mathbf{h}^{(s)}))\| \leq \gamma$
- Then, we have

$$\|J_{\mathbf{h}^{(r)}}(\mathbf{h}^{(r-1)})\| \leq \|\mathbf{W}\| \|\text{diag}(a'(\mathbf{h}^{(r)}))\| \leq \rho(\mathbf{W})\gamma$$

and, thus,

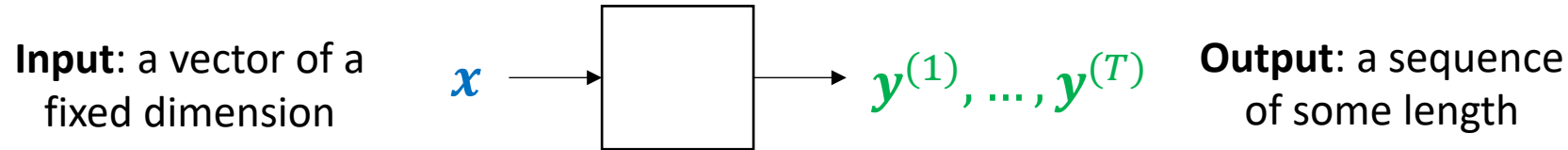
$$\|J_{\mathbf{h}^{(s)}}(\mathbf{h}^{(t)})\| \leq (\rho(\mathbf{W})\gamma)^{t-s}$$

- If $\rho(\mathbf{W}) < 1/\gamma$, then vanishing gradients occur

Strategies for multiple time scales

- Adding skip connections through time
- Leaky units
- Removing connections

Vector to sequence RNN models

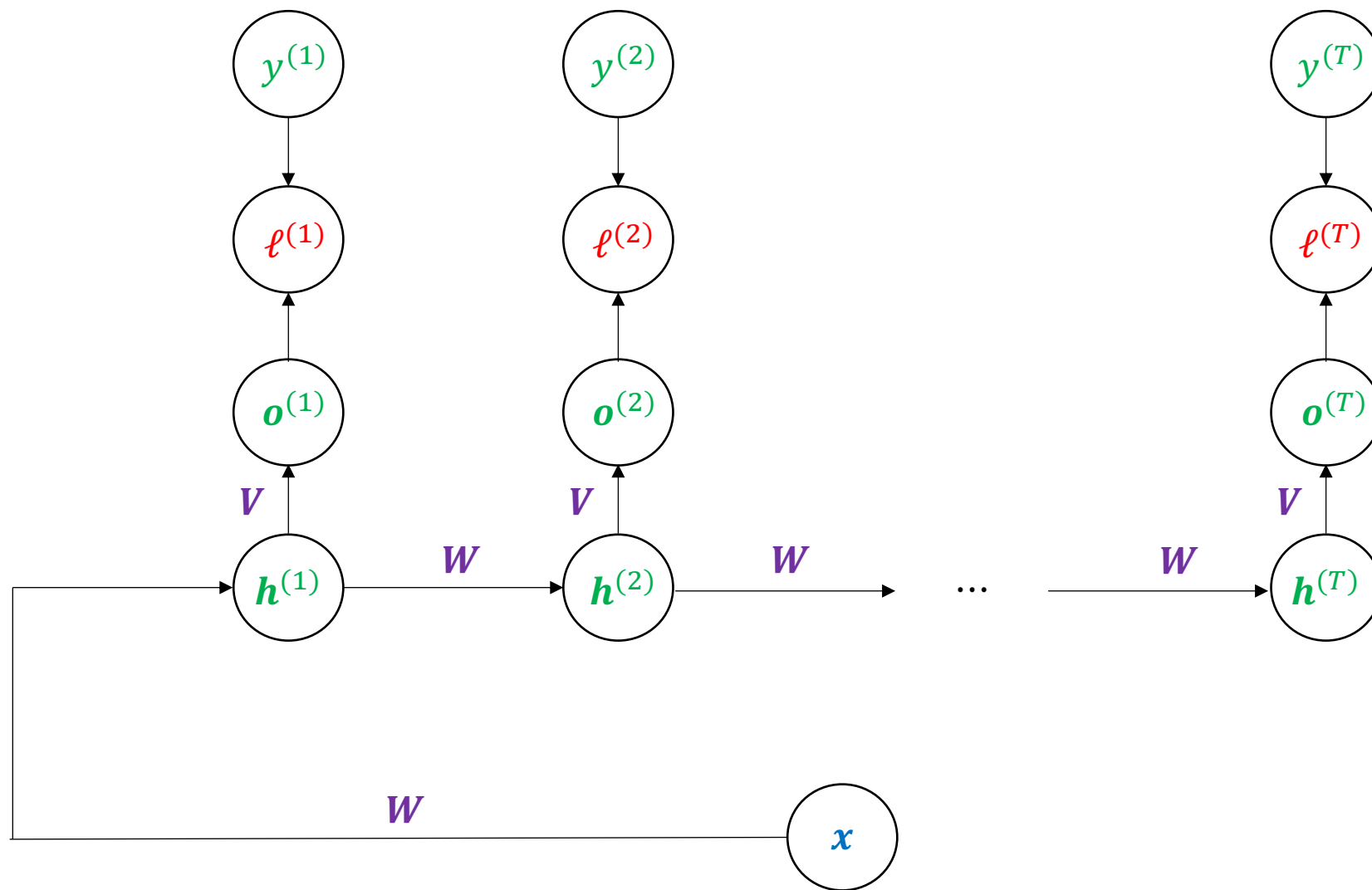


- Example applications:
 - Image captioning
 - A module for sequence-to-sequence models
- Common approaches based on using a standard RNN:
 1. Initialize hidden state $h^{(0)}$ to input vector x
 2. Feed input vector x as input at each time step
 3. Combine both

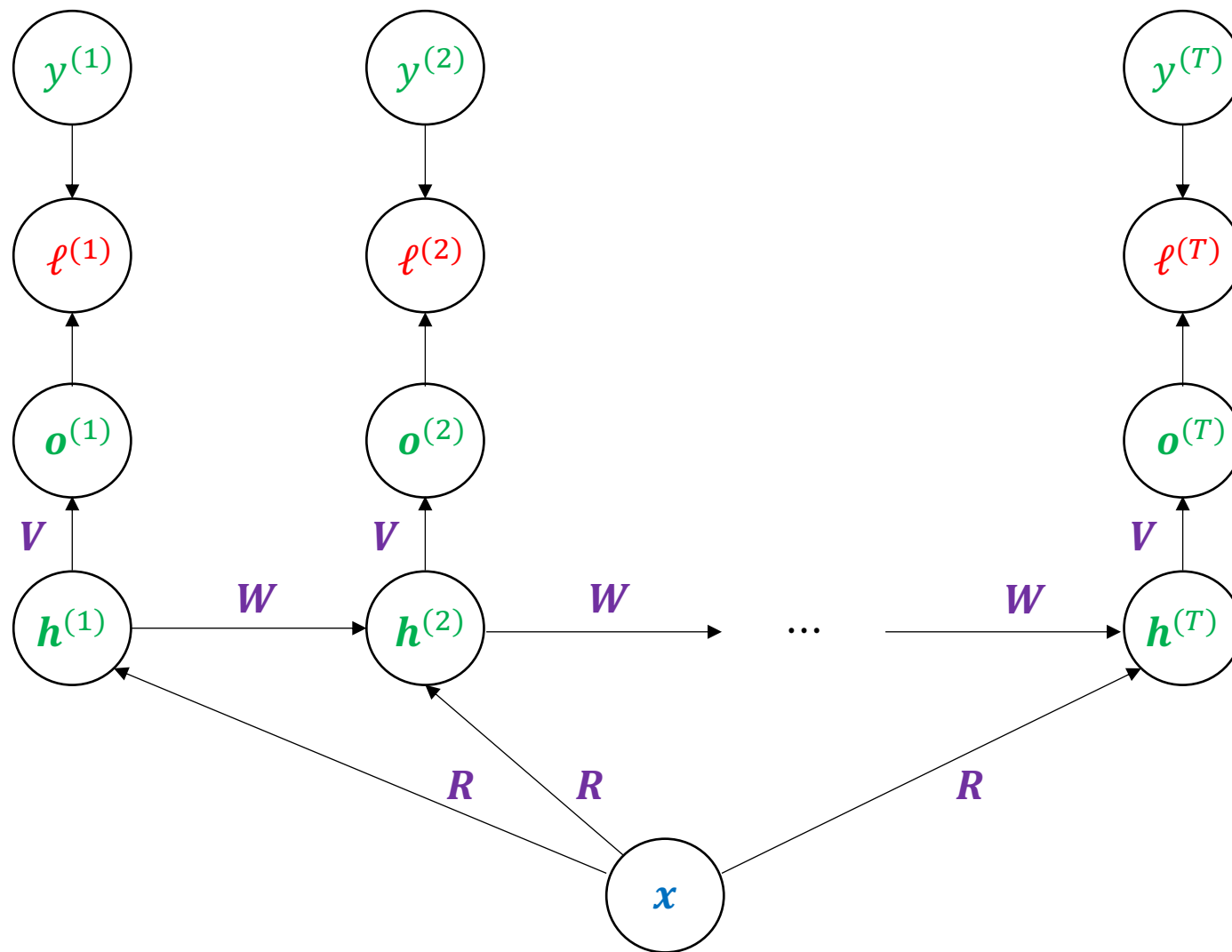


A bird flies
over the water

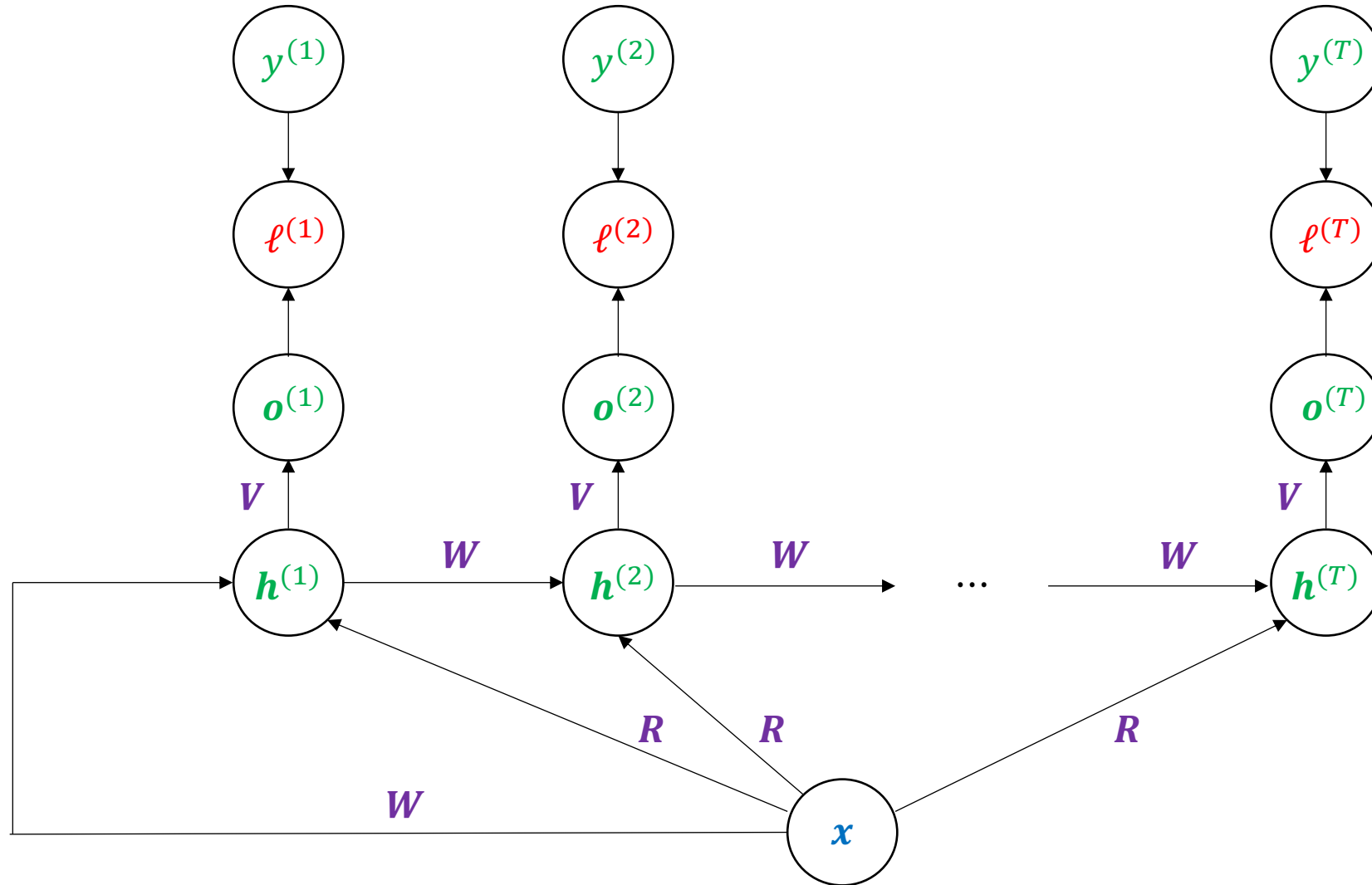
Approach 1



Approach 2



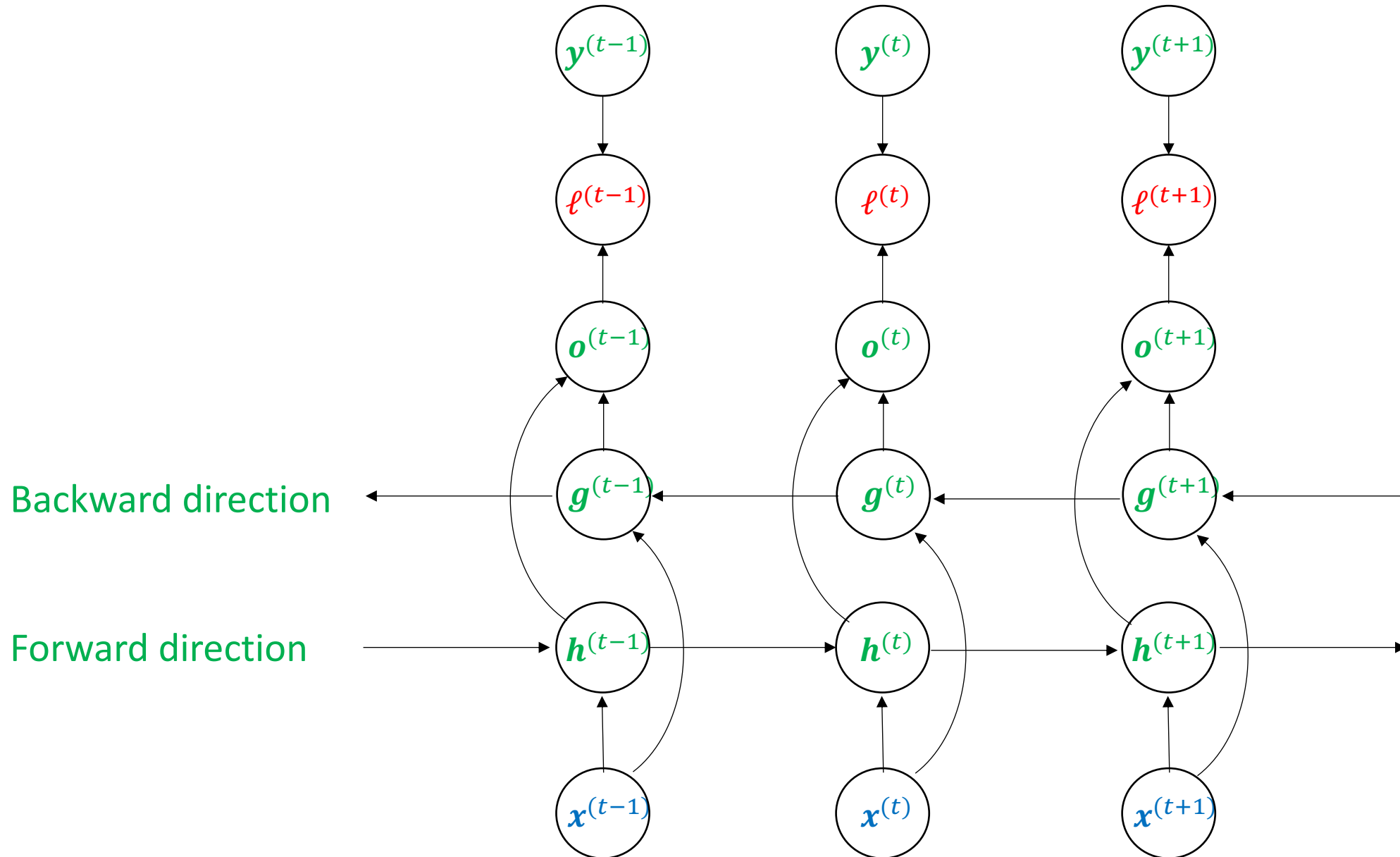
Approach 3



Bidirectional RNNs

- **Bidirectional RNNs**: output predictions depend on the whole input sequence
- Example use cases:
 - **Speech recognition**: interpretation of a phoneme may depend on the next few phonemes (co-articulation) or even next few words (linguistic dependencies)
 - **Handwriting recognition**
 - **Bioinformatics**
- Introduced by Schuster and Paliwan 1997
- Key idea: **combine two RNNs**
 - One having connections forward in time
 - Other having connections backward in time

Common bidirectional RNN architecture



Gated RNNs

Gated RNNs: key ideas

- Create paths through time along which derivatives of parameters neither vanish nor explode
- Connection weights may change at each time step

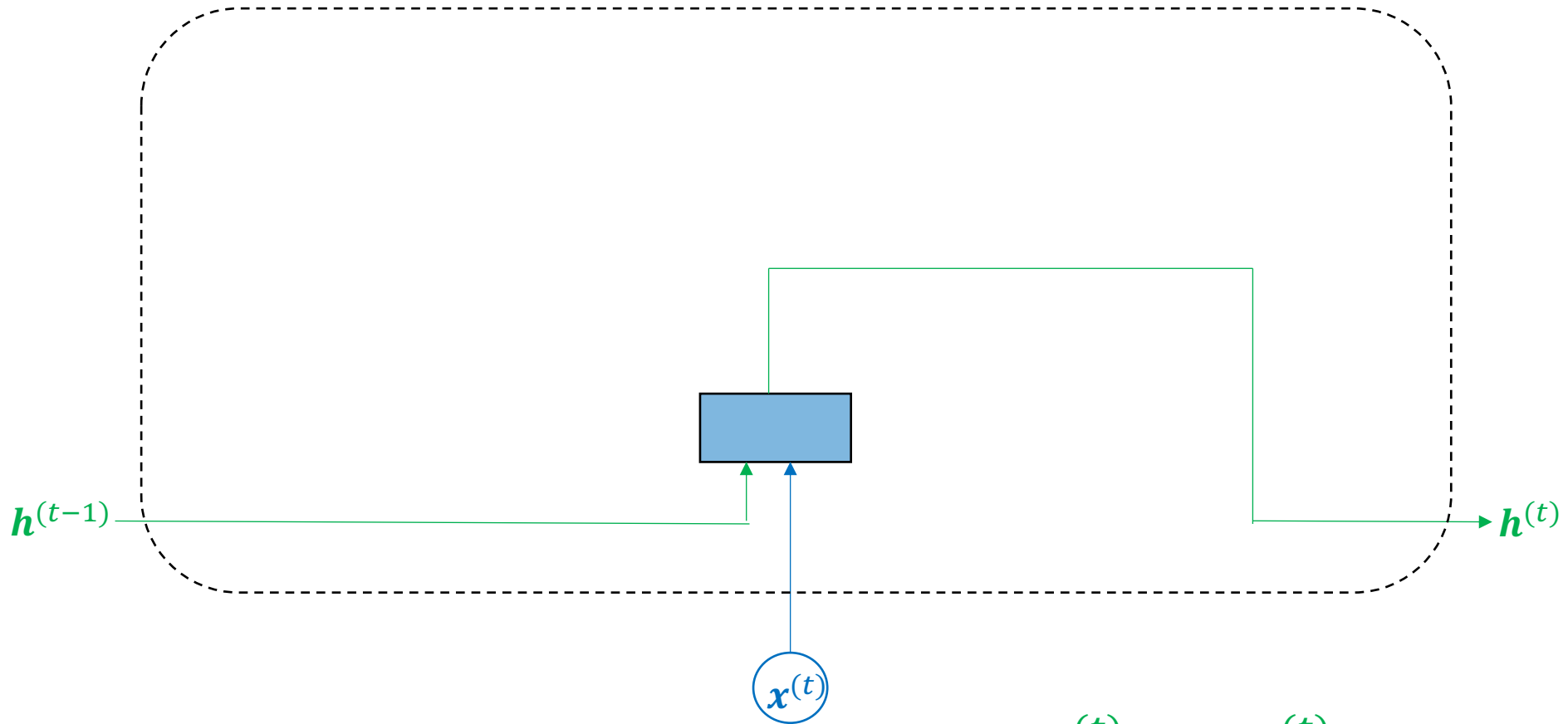
Long Short-Term Memory (LSTM)

- **Key idea:** self-loops creating paths where the gradient can flow for long durations
 - Weights of the self-loop are gated, i.e. controlled by another hidden unit
 - This allows to change the timescale of integration
- Introduced by Hochreiter and Schmidhuber (1997), self-loops weights conditioned on the context Gers et al (2000)
- Shown to learn long-term dependencies more easily than simple RNNs

LSTM applications

- Unconstrained handwriting recognition
- Speech recognition
- Handwriting generation
- Machine translation
- Image captioning
- Parsing

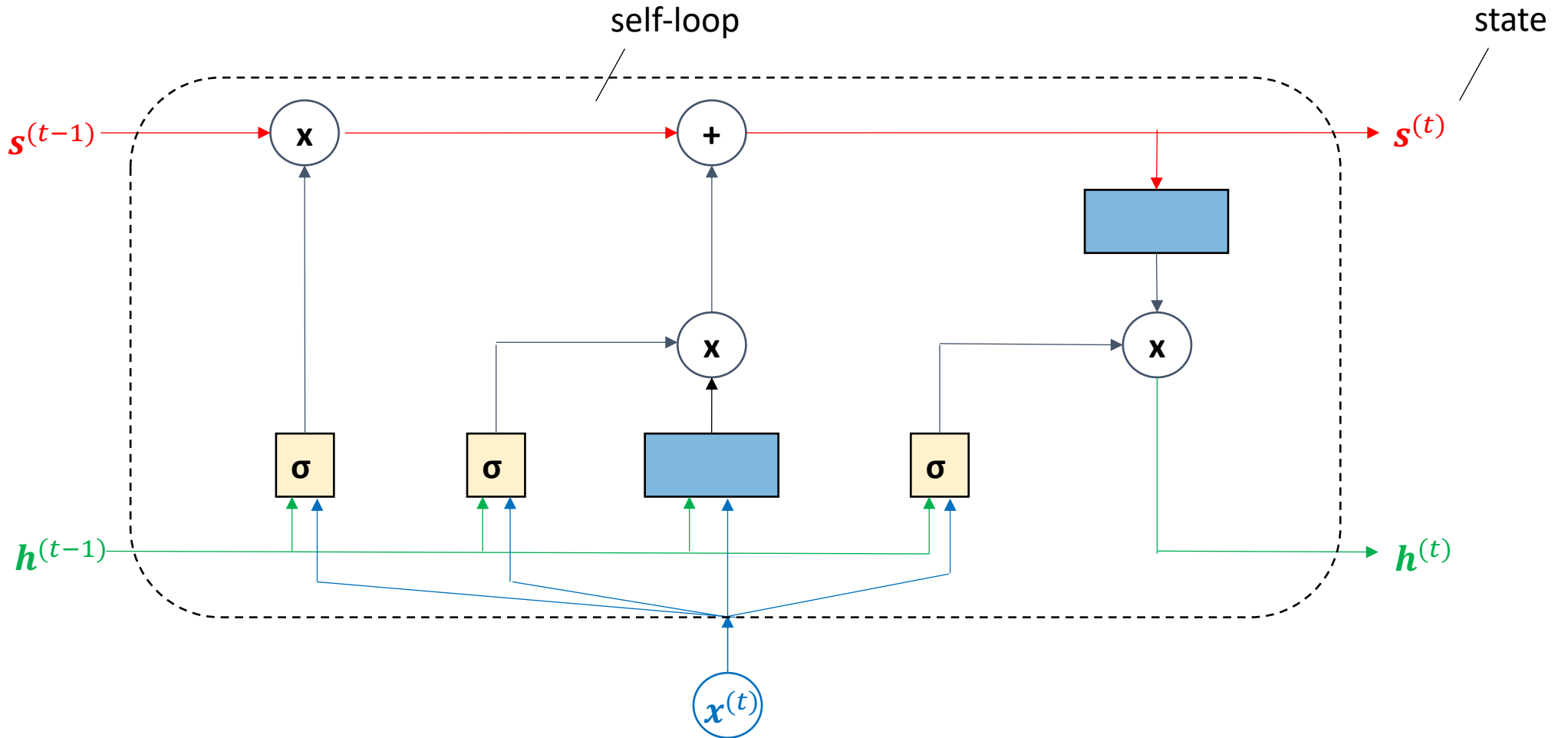
Basic RNN cell



$$h^{(t)} = a(a^{(t)})$$

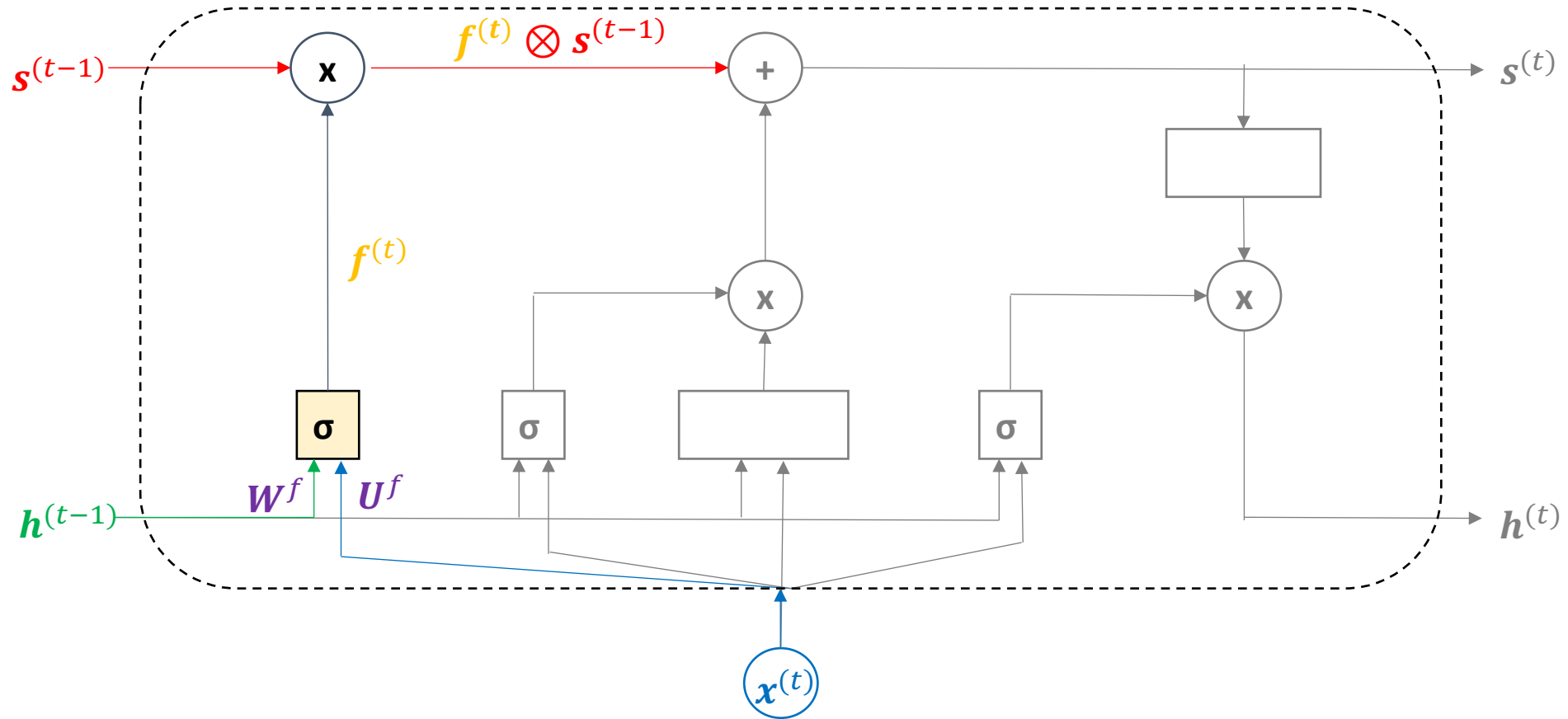
$$a^{(t)} = \mathbf{W}h^{(t-1)} + \mathbf{U}x^{(t)} + \mathbf{b}$$

LSTM cell



σ gating units control the flow of information (σ denotes sigmoid function)

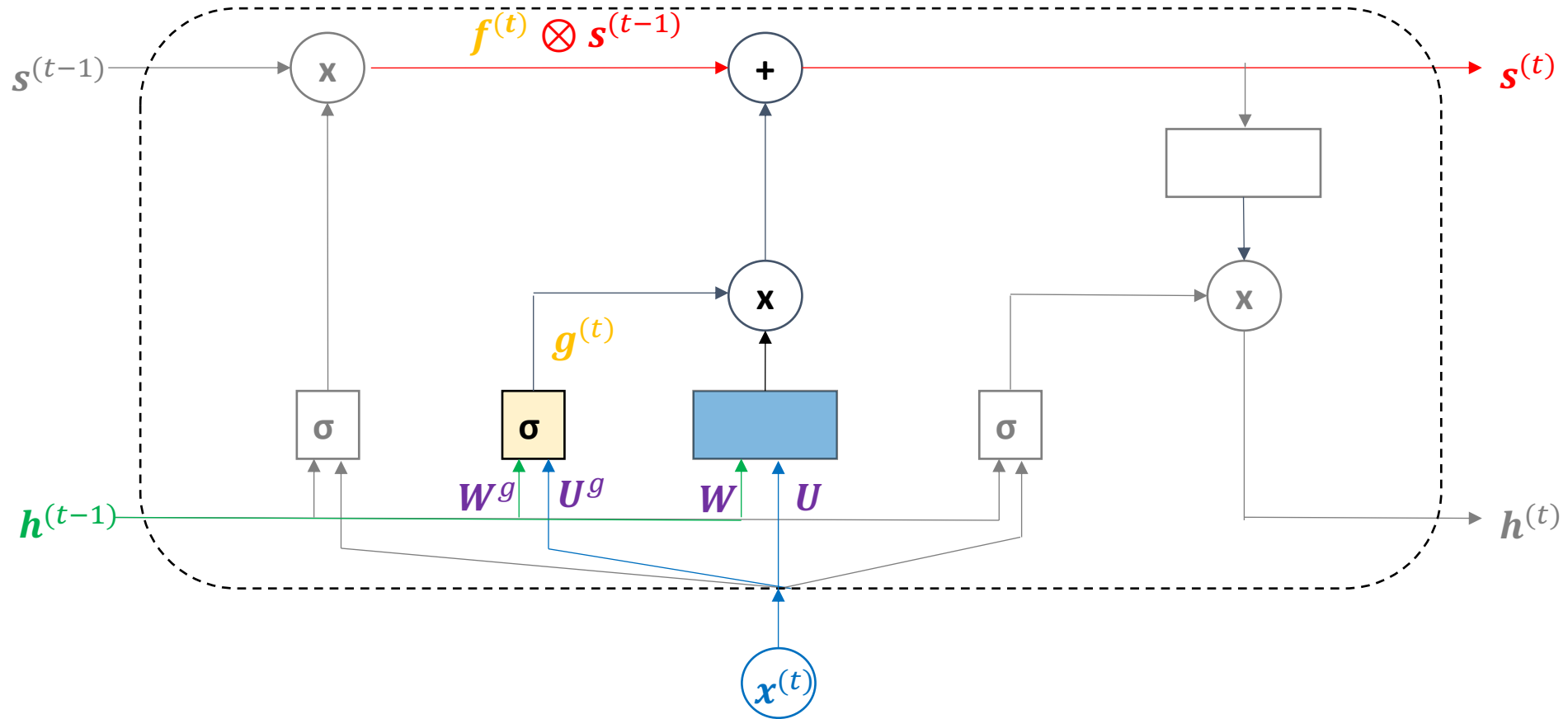
Forget gate



- $f^{(t)} = \sigma(U^f x^{(t)} + W^f h^{(t-1)} + b^f)$

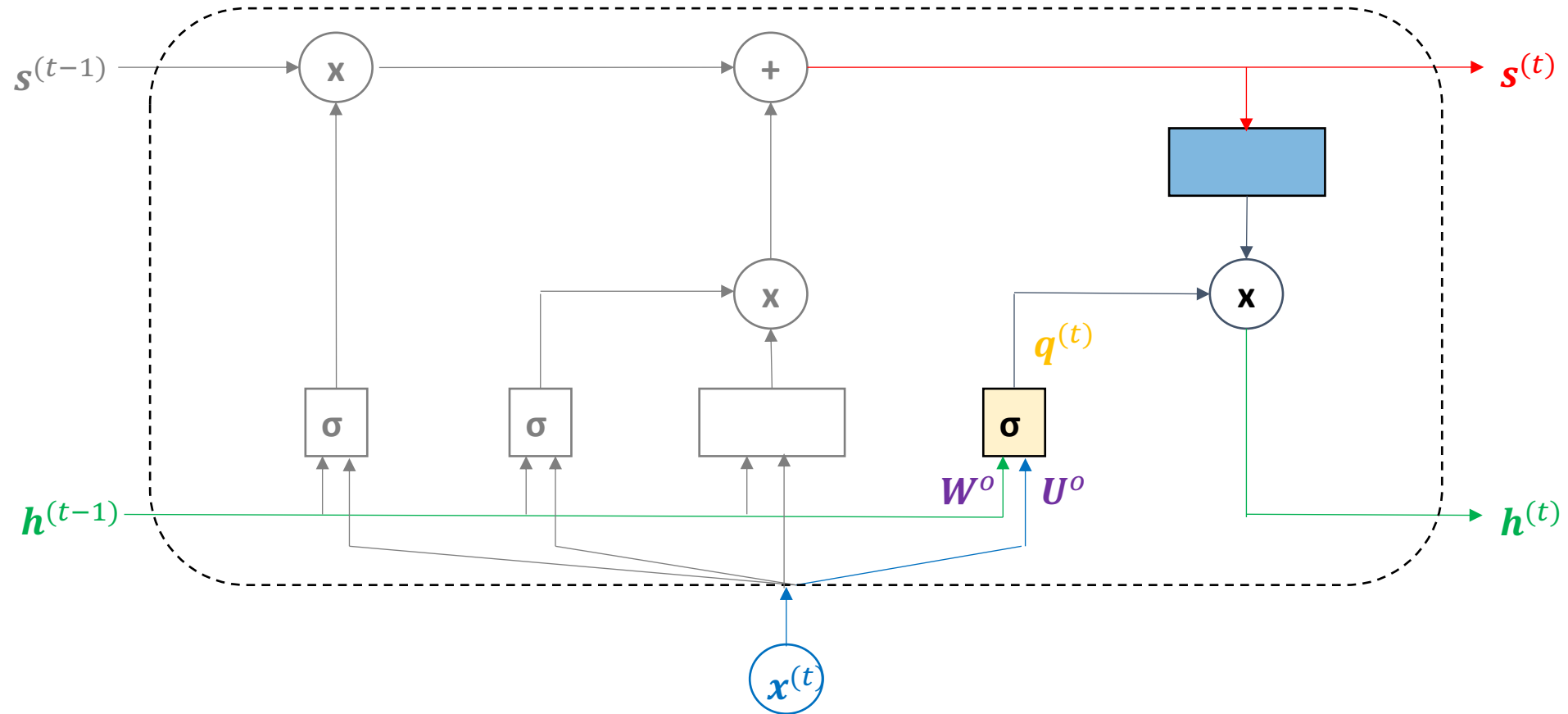
- Def. for two vectors x and y : $x \otimes y = (x_1 y_1, \dots, x_n y_n)^\top$

Input gate



- $g^{(t)} = \sigma(U^g x^{(t)} + W^g h^{(t-1)} + b^g)$
- $s^{(t)} = f^{(t)} \otimes s^{(t-1)} + g^{(t)} \otimes a(Ux^{(t)} + Wh^{(t-1)} + b)$

Output gate



- $q^{(t)} = \sigma(U^o x^{(t)} + W^o h^{(t-1)} + b^o)$
- $h^{(t)} = a(s^{(t)}) \otimes q^{(t)}$

Gated recurrent unit (GRU) [Cho et al, 2014]

- GRU cell update equations:

$$\mathbf{h}^{(t)} = \mathbf{u}^{(t)} \otimes \mathbf{h}^{(t-1)} + (1 - \mathbf{u}^{(t)}) \otimes a(\mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}(\mathbf{r}^{(t)} \otimes \mathbf{h}^{(t-1)}) + \mathbf{b})$$

$$\mathbf{u}^{(t)} = \sigma(\mathbf{U}^u \mathbf{x}^{(t)} + \mathbf{W}^u \mathbf{h}^{(t-1)} + \mathbf{b}^u)$$

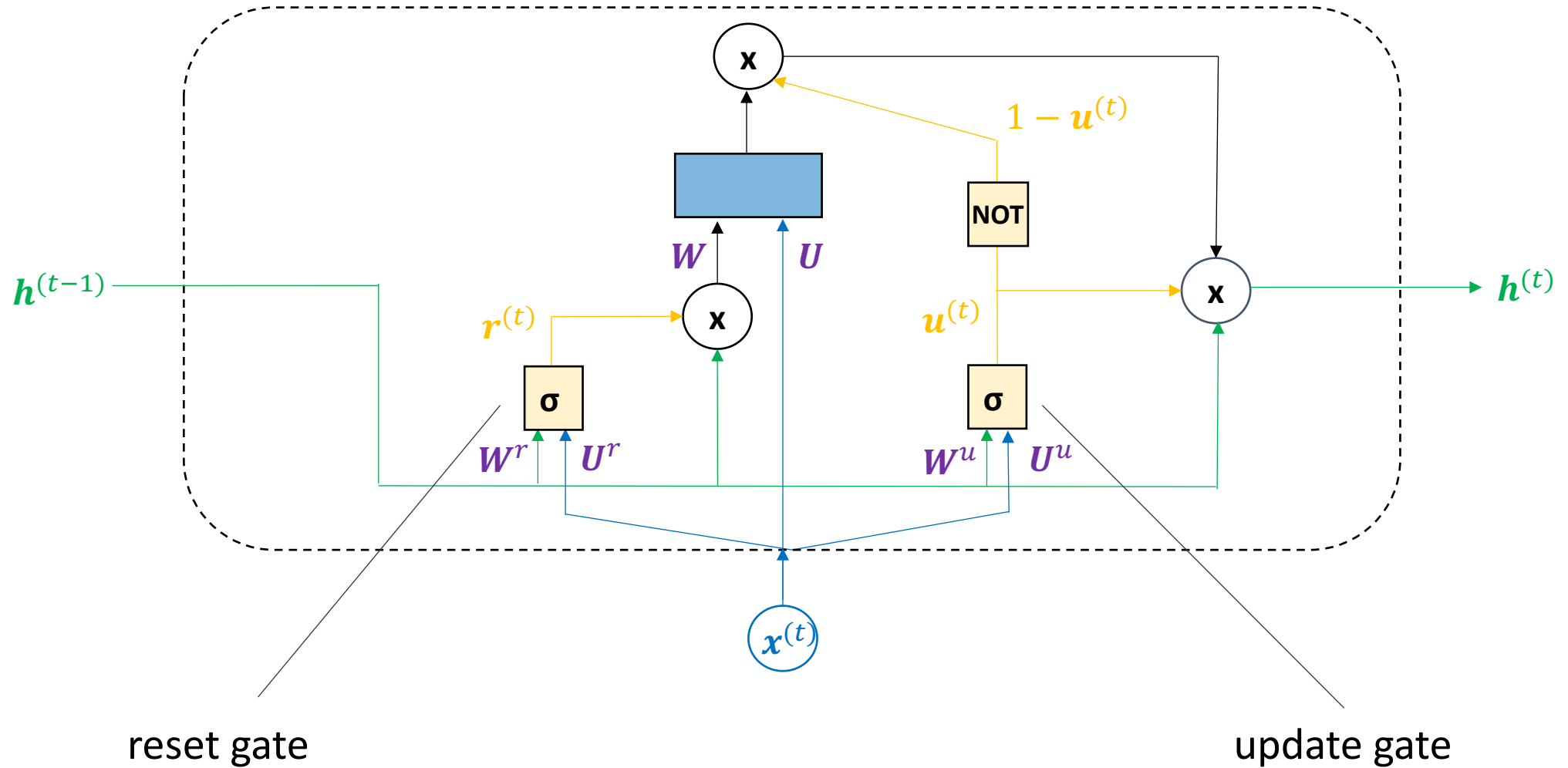
update gate

$$\mathbf{r}^{(t)} = \sigma(\mathbf{U}^r \mathbf{x}^{(t)} + \mathbf{W}^r \mathbf{h}^{(t-1)} + \mathbf{b}^r)$$

reset gate

- Each hidden unit has separate update and reset gates
 - Short-term dependencies captured by frequently active reset gate
 - Long-term dependencies captured by frequently active update gate

GRU network architecture



Sequence transduction models

Sequence transduction models

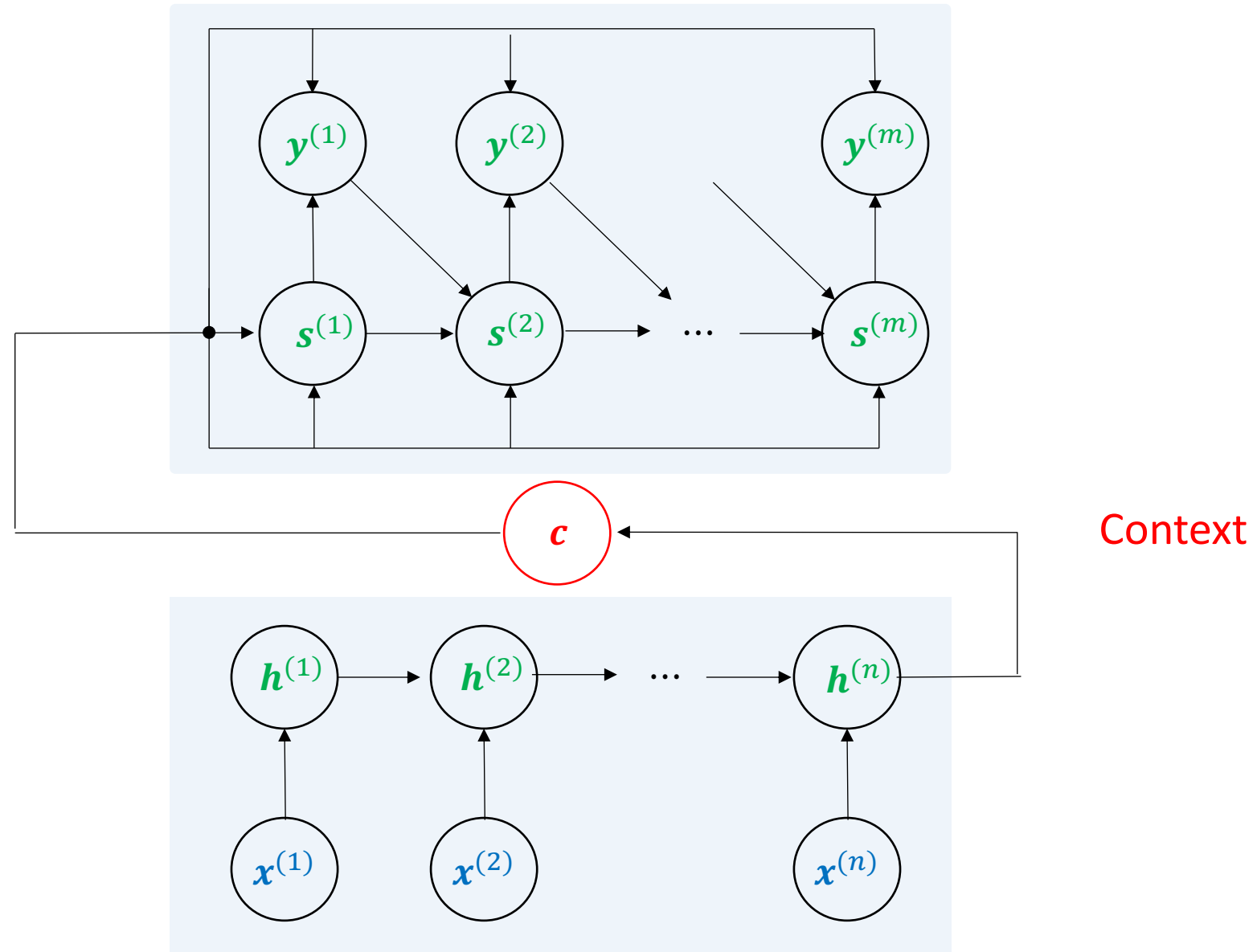
- **Sequence transduction**: mapping an input sequence to an output sequence
 - Input and output sequence are not necessarily of the same length
- Applications:
 - Machine translation, e.g. English to German, Chinese to English
 - Speech recognition
 - Question answering
- Common approach: **encoder-decoder architecture**
 - Also referred to as a **sequence-to-sequence architecture**

Encoder-decoder architecture

- Encoder-decoder architecture:
 - Encoder [reader | input] RNN
 - Decoder [writer | output] RNN connected with a context
- Encoder RNN: processes the input sequence and outputs a context representation
 - Context is usually a simple function of final hidden state
- Decoder RNN: maps a fixed-length context vector to an output sequence
- Example: neural machine translation
 - Encoder encodes a source sentence into a fixed-length vector from which decoder generates a translation

Standard encoder-decoder architecture

Decoder



Encoder

Jointly learning to align and translate [Bahdanau et al 2015]

- Using a fixed-length context vector may be a bottleneck
 - Potential difficulty for long sequences
 - Especially for sequences longer than observed in the training data
- Proposed solution: use an **attention mechanism** for soft-selection of parts of input sequence that are relevant for predicting a target output element
 - Alleviates having to form these parts as a hard segment explicitly

Alignment model

- Context vector for output at position i : $\mathbf{c}^{(i)} = \sum_{j=1}^n \alpha_{i,j} \mathbf{h}^{(j)}$ where

$$\alpha_{i,j} = \frac{e^{e_{i,j}}}{\sum_{k=1}^n e^{e_{i,k}}}$$

are the attention weights with

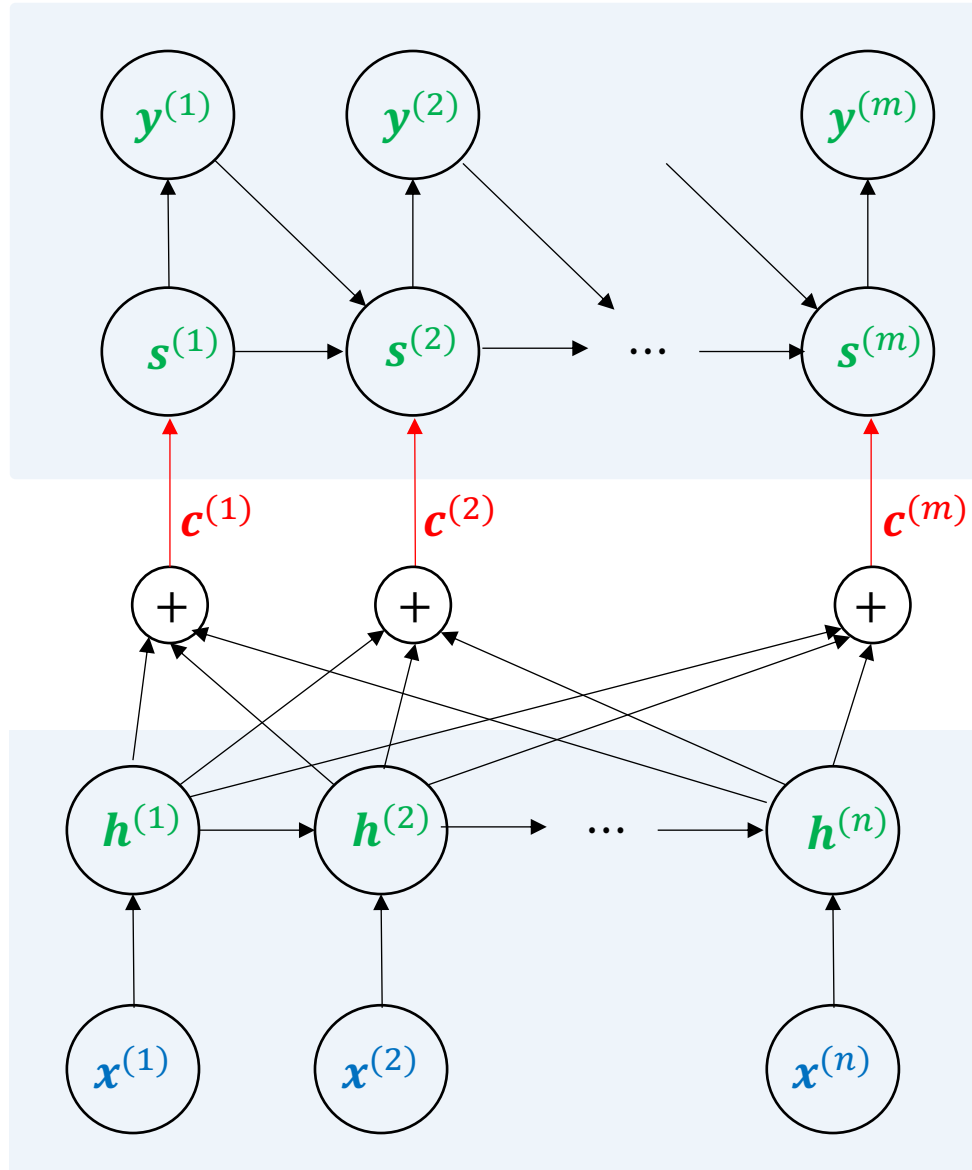
$$e_{i,j} = f_{\theta}(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

quantifying how well the input at position j and output at position i match

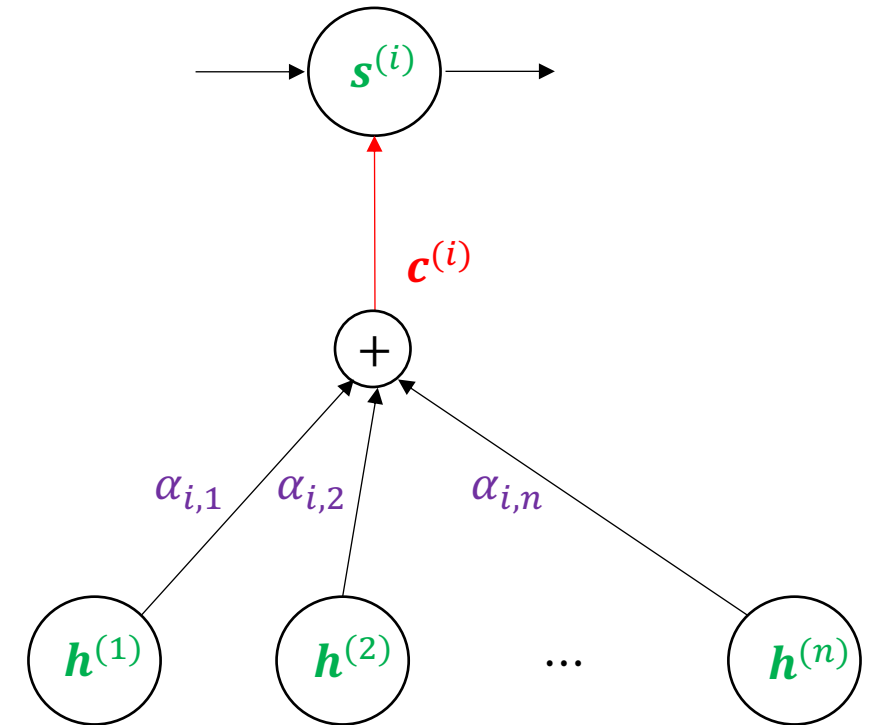
- Here f_{θ} an alignment function, parametrized as a feedforward neural network which is jointly trained with all other components of the network

Encoder-decoder architecture with alignment

Decoder



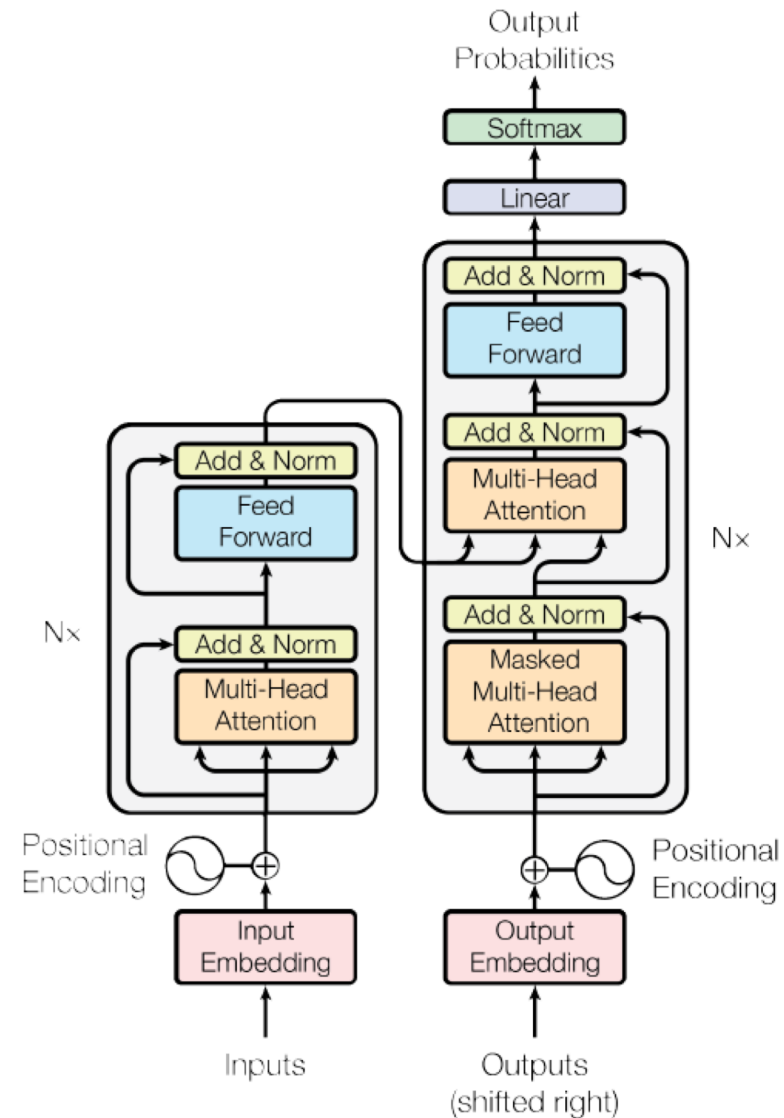
Encoder



Transformer architecture

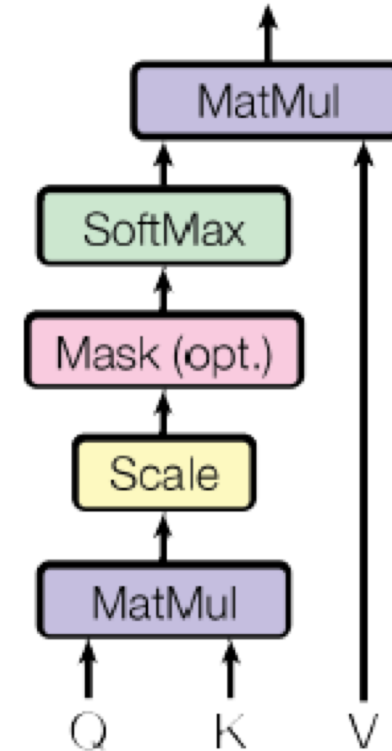
[Vaswani et al, 2017]

- An encoder-decoder architecture with
 - An attention mechanism
 - A stack of fully connected layers
- No recurrent neural networks
- Fast training: allows for parallelization



Attention module

- Attention based on **scalar dot-product**
- Input:
 - Q query vectors of dimension d_k
 - K key vectors of dimension d_k
 - V value vectors of dimension d_v
- Output:



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$1/\sqrt{d_k}$ normalization to control the skew of softmax

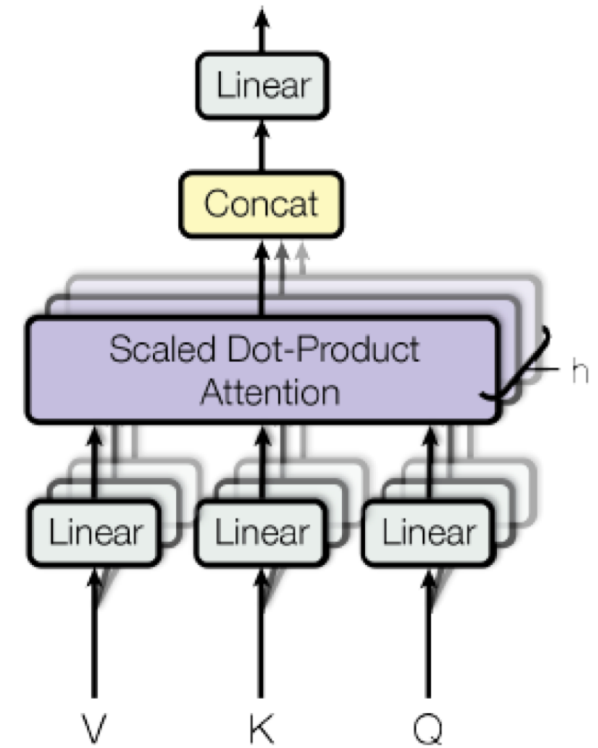
Attention module

- Multi-head attention

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}^O$$

$$\text{head}_i = \text{Attention}(\mathbf{Q} \mathbf{W}_i^Q, \mathbf{K} \mathbf{W}_i^K, \mathbf{V} \mathbf{W}_i^V)$$

where $\mathbf{W}_i^Q \in \mathbf{R}^{d \times d_k}$, $\mathbf{W}_i^K \in \mathbf{R}^{d \times d_k}$, $\mathbf{W}_i^V \in \mathbf{R}^{d \times d_v}$ are parameters



Use of attention modules

- Encoder-decoder attention layer
 - Queries come from the previous decoder layer, keys and values come from the output of the encoder
 - This mimics typical encoder-decoder attention mechanism in sequence to sequence models
- Encoder self-attention layer
 - Each position in the encoder can attend to **all positions** in the previous layer of the encoder
- Decoder self-attention layer
 - Each position in the decoder can attend to **all position in the decoder up to and including that position** (to ensure auto-regressive property)
 - Implemented by **masking** (setting input values to softmax to $-\infty$)

Feedforward network module

- Feedforward network module applied to each position separately and identically
- Two linear transformations with a ReLU activation in between:

$$\text{FFN}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

- Parameters are different for different layers
- Dimensions of input and outputs are 512
- The inner layer has dimensionality of 2048

References

- I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, Chapter 10: Sequence Modeling: Recurrent and Recursive Nets, The MIT Press, 2017
- A. Graves, Supervised Sequence Labelling with Recurrent Neural Networks, Springer 2012
- S. Hochreiter and J. Schmidhuber, Long Short-Term Memory, Neural Computation, 9(8), 1735-1780, 1997
- J. Chung, C. Gulchere, K. Cho, and Y. Bengio, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, ArXiv 2014
- I. Sutskever, O. Vinyals and Q. V. Le, Sequence to Sequence Learning with Neural Networks, NIPS 2014
- D. Bahdanu, K. Cho, and Y. Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015
- A. Vaswani et al, Attention Is All You Need, NIPS 2017

References (cont'd)

- Y. Bengio, P. Simard, and P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. on Neural Networks, 5(2), 157-166, 1994
- A. Vaswani et al, Attention is All you Need, NIPS 2017
- Cho et al, Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, EMNLP 2014
- M. Schuster and K. K. Paliwan, Bidirectional Recurrent Neural Networks, IEEE Trans. on Signal Processing, Vol 45, No 11, November 1997
- I. Sutskever, O. Vinyals, and Q. V. Le, Sequence to Sequence Learning with Neural Networks, NIPS 2014
- Wu et al, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016

References (cont'd)

Exploding and vanishing gradients

- Y. Bengio, P. Simard, and P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. on Neural Networks, 5(2), 157-166, 1994
- K. Doya, Bifurcations of recurrent neural networks in gradient descent learning, IEEE Trans. on Neural Networks, 1, 75-80, 1993
- R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, ICML 2013

References (cont'd)

Error metrics

- M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul, A Study of Translation Edit Rate with Targeted Human Annotation, Proceedings of Association for Machine Translation in the Americas, 2006
- K. Papineni, S. Roukos, T. Ward, and W.-Y. Zhu, BLEU: a Method for Automatic Evaluation of Machine Translation, Proc. of ACL 2002

Seminar exercises

- Text generation using an RNN

Appendix

Performance metrics

Word error rate (WER)

- A common metric of the performance of a speech recognition or a machine translation system
- Word error rate of a candidate text for given reference text:

$$\text{WER} = \frac{S+D+I}{N}$$

- N = number of words in the reference
- S = number of substitutions to transform the reference text to the candidate text
- D = number of deletions to transform the reference text to the candidate text
- C = number of correct words
- Note: $N = S + D + C$

Translation error rate (TER)

- An error metric for machine translation that measures the number of edits required to change an output into one of given references
- Introduced by [Snover et al 2006]
- <http://www.cs.umd.edu/~snover/tercom/>

BLEU (bilingual evaluation understudy)

- An error metric for automatic evaluation of machine translation [Papineni et al, 2002]
- Addresses the requirements:
 - Computation efficiency (to enable quick experimentation)
 - Language independency
 - High correlation with human evaluations (the closer a machine translation is to a professional human translation, the better is)
- Uses a numerical **translation closeness metric** and a corpus of good quality **reference human translations**
- Translation closeness metric
 - Inspired by word error rate metric used by the speech recognition community
 - Main idea: use a weighted average of variable length phrase matches against reference translations

Standard unigram precision

- **Standard unigram precision**: number of candidate translation words (unigrams) which occur in **any** reference translation divided by the total number words in the candidate translation
- Example:
 - **Candidate**: the the the the the the the.
 - **Reference 1**: The cat is on the mat.
 - **Reference 2**: There is a cat on the mat.
- Standard unigram precision = 1
 - Undesired: high precision for common words

Modified n-gram precision

- Modified unigram precision:
 - For each word in the candidate translation, count the maximum number of times the word occurs in a reference translation (referred to as maximum reference count)
 - Clip the count value for each candidate word by its maximum reference count
 - Modified unigram precision: sum of clipped counts for each distinct word in the candidate translation and divide by the total number of words in the candidate translation
- Example (cont'd): modified unigram precision = $2/7$

BLEU score

- Modified n-gram precision is defined analogously to modified unigram precision
- BLEU score of a candidate translation:

$$\text{BLEU} = \text{BP} \exp\left(\sum_{n=1}^N w_n \log(p_n)\right)$$

brevity penalty positive weights summing to 1 modified n-gram precision

where

$$\text{BP} = \exp\left(\min\left\{1 - \frac{r}{c}, 0\right\}\right)$$

Effective reference corpus length
= sum of best match lengths for each
candidate sentence in the corpus length of the candidate translation