

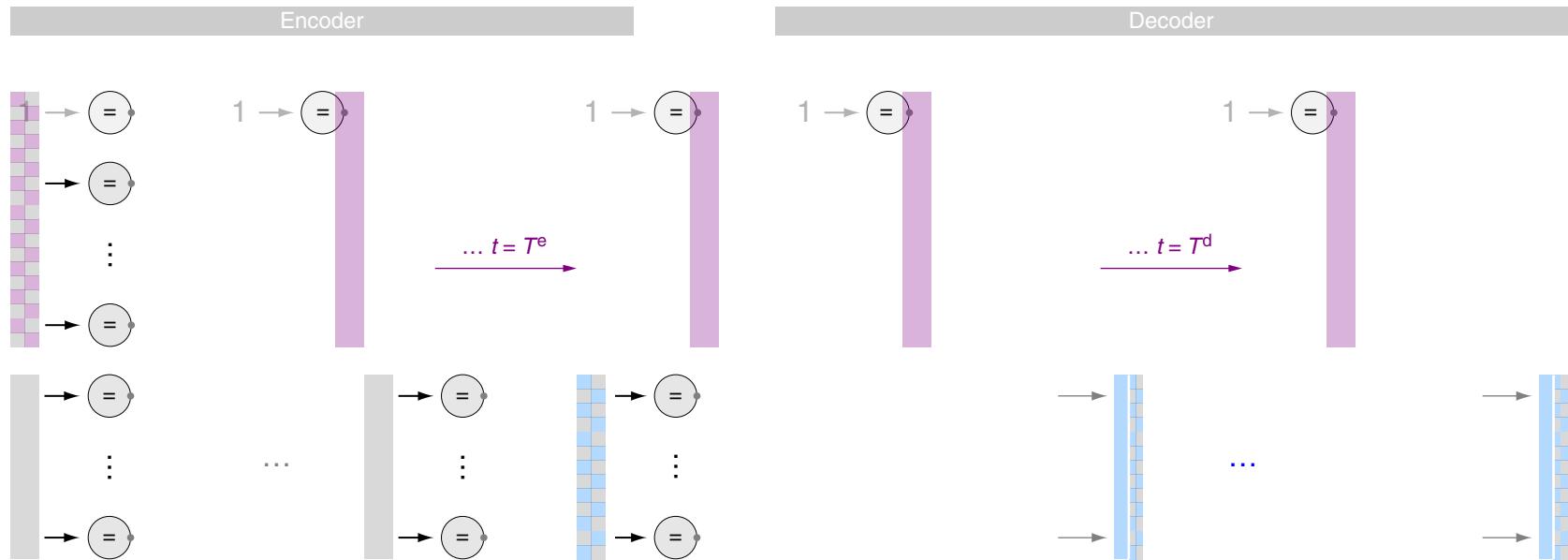
IX. Deep Learning

- ❑ Elements of Deep Learning
- ❑ Convolutional Neural Networks
- ❑ Autoencoder Networks
- ❑ Recurrent Neural Networks
- ❑ Long-Term Dependencies
- ❑ RNNs for Machine Translation
- ❑ Attention Mechanism
- ❑ Self Attention and Transformers
- ❑ Transformer Language Models
- ❑ Pretraining

Recurrent Neural Networks

Notation I (color scheme)

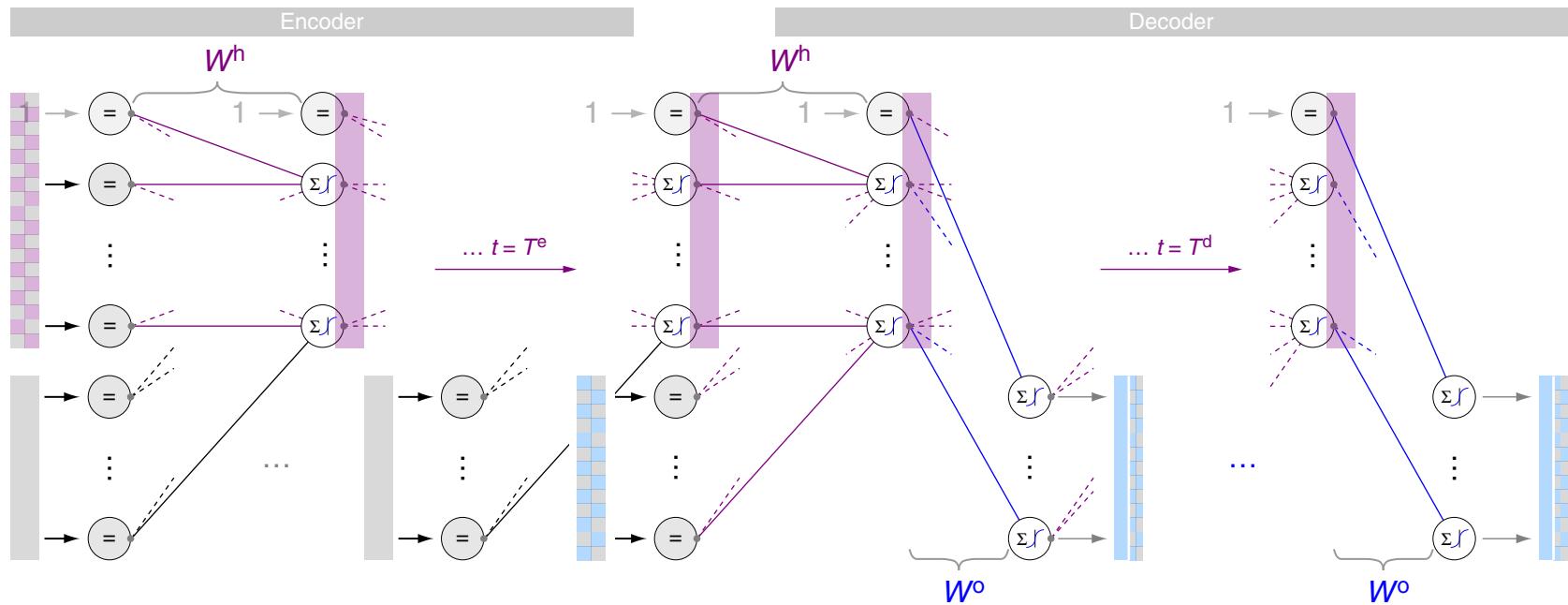
[notation: color scheme, computational graph, language model]



Recurrent Neural Networks

Notation I (color scheme)

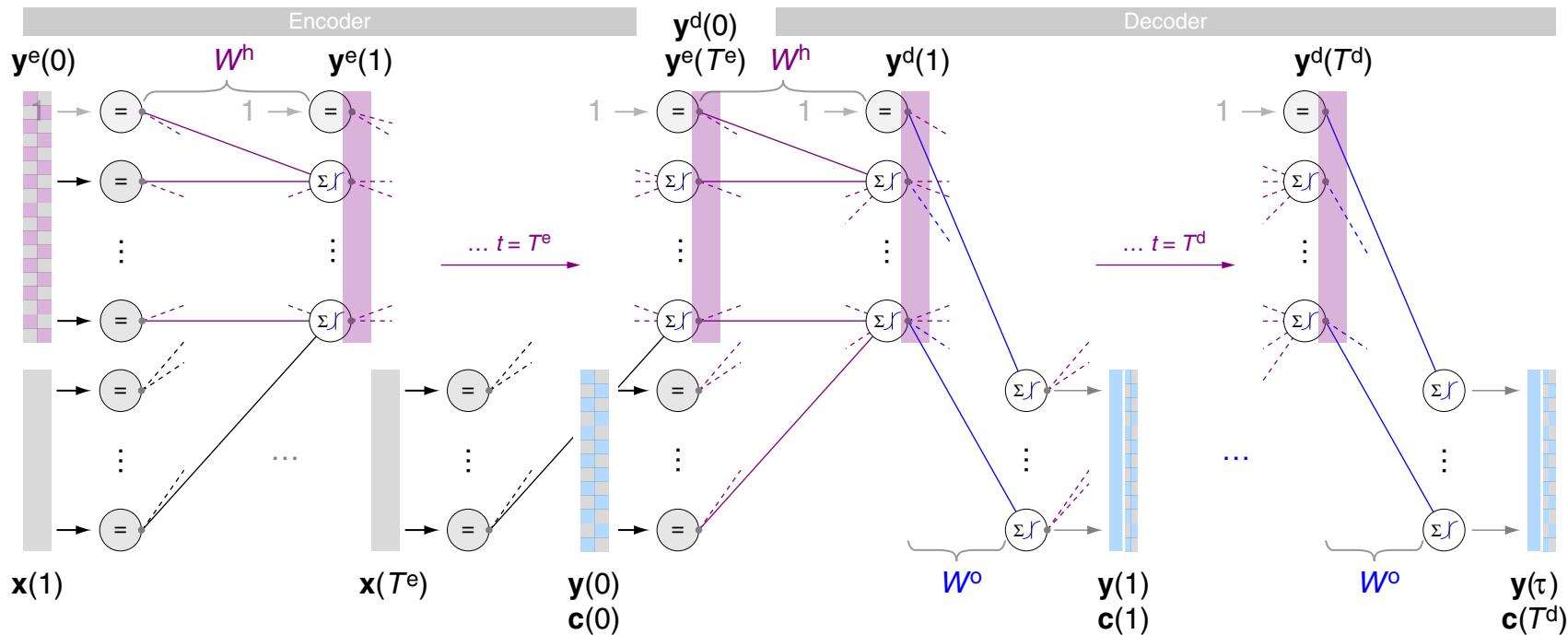
[notation: color scheme, computational graph, language model]



Recurrent Neural Networks

Notation I (color scheme)

[notation: color scheme, computational graph, language model]



$x(t)$ input
in_word

$y^h(t)$ hidden
 $y^e(t)$ hidden encoder
 $y^d(t)$ hidden decoder

predefined
hidden

$y^a(t)$ attention

$y(t)$ output
out_word (test)

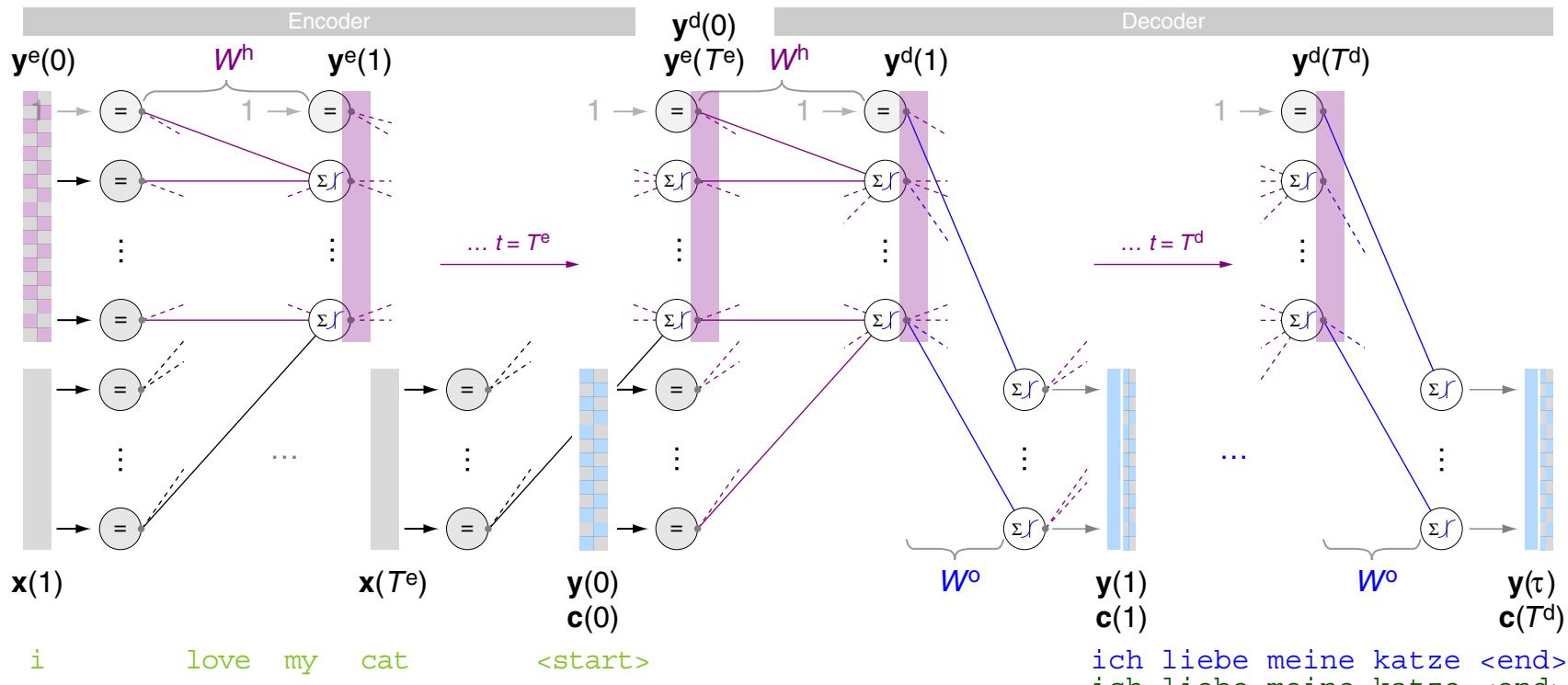
$c(t)$ target
out_word (training)

$y(t)$ output or $c(t)$ target
(depends on training or test phase)

Recurrent Neural Networks

Notation I (color scheme)

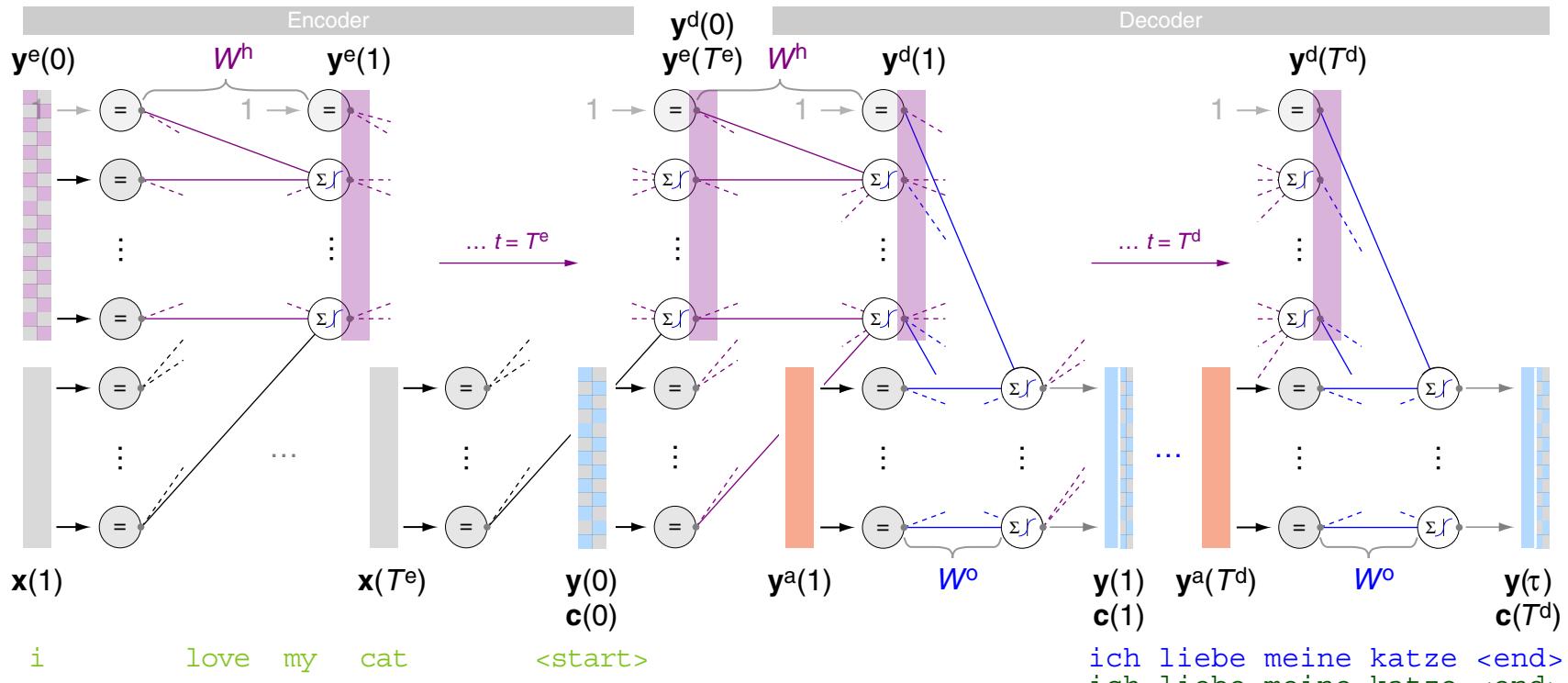
[notation: color scheme, computational graph, language model]



Recurrent Neural Networks

Notation I (color scheme)

[notation: color scheme, computational graph, language model]



Remarks:

- A hidden vector is the result of an intermediate computation in a multilayer network. If sequences are processed, hidden vectors may be distinguished as hidden *encoder* vectors (which consider the input at a certain time step) and hidden *decoder* vectors (which generate the output at a certain time step).
- A predefined hidden vector is used to for initialization purposes for the first hidden layer in a sequence-processing multilayer network. Typically, it is a constant vector of zeroes.
- Attention vectors combine the information in hidden vectors in a way that is specific to a certain time step. Keyword: vanishing gradient problem
- A target vector (or target vector sequence) encodes the desired output. Keywords: supervised learning, ground truth

Recurrent Neural Networks

Types of Learning Tasks [Recap]

(S1) **sequence → class**

sentence → {⊕, ⊖}

i love my cat → ⊕

(S2) **class → sequence**

{⊕, ⊖} → **sentence**

⊕ → i love my cat

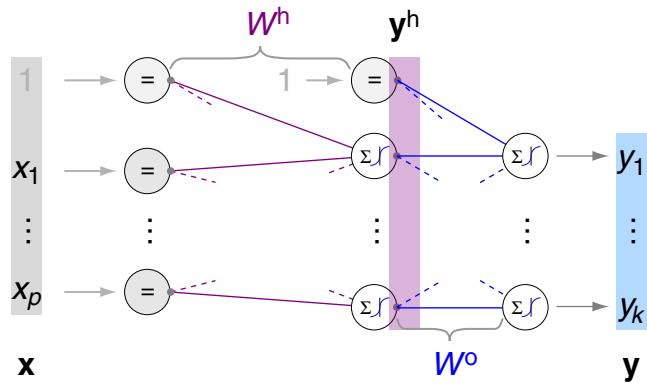
(S3) **sequence → sequence**

English sentence → **German sentence**

i love my cat → ich liebe meine katze

Recurrent Neural Networks

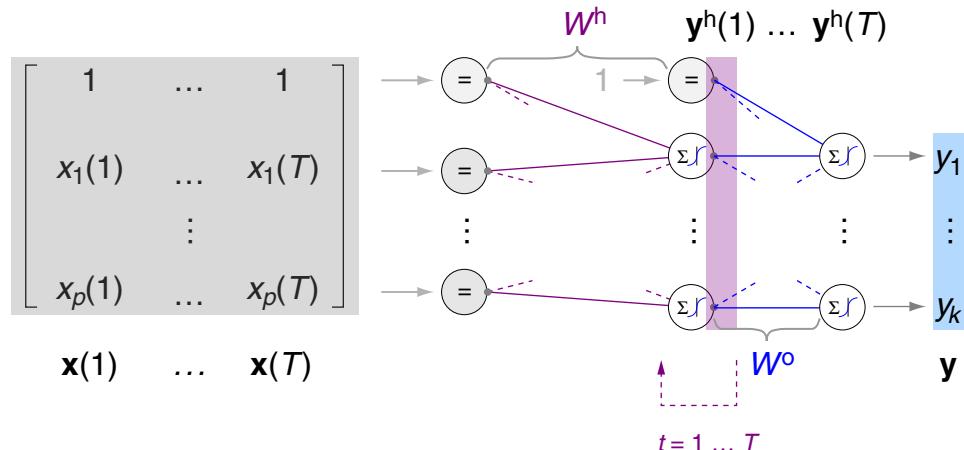
RNN Sequence Encoding



- One p -dimensional input vector \mathbf{x} .
- One hidden layer (general: $d-1$ hidden layers, i.e., d active layers).
- One k -dimensional output vector $\mathbf{y}(\mathbf{x})$.

Recurrent Neural Networks

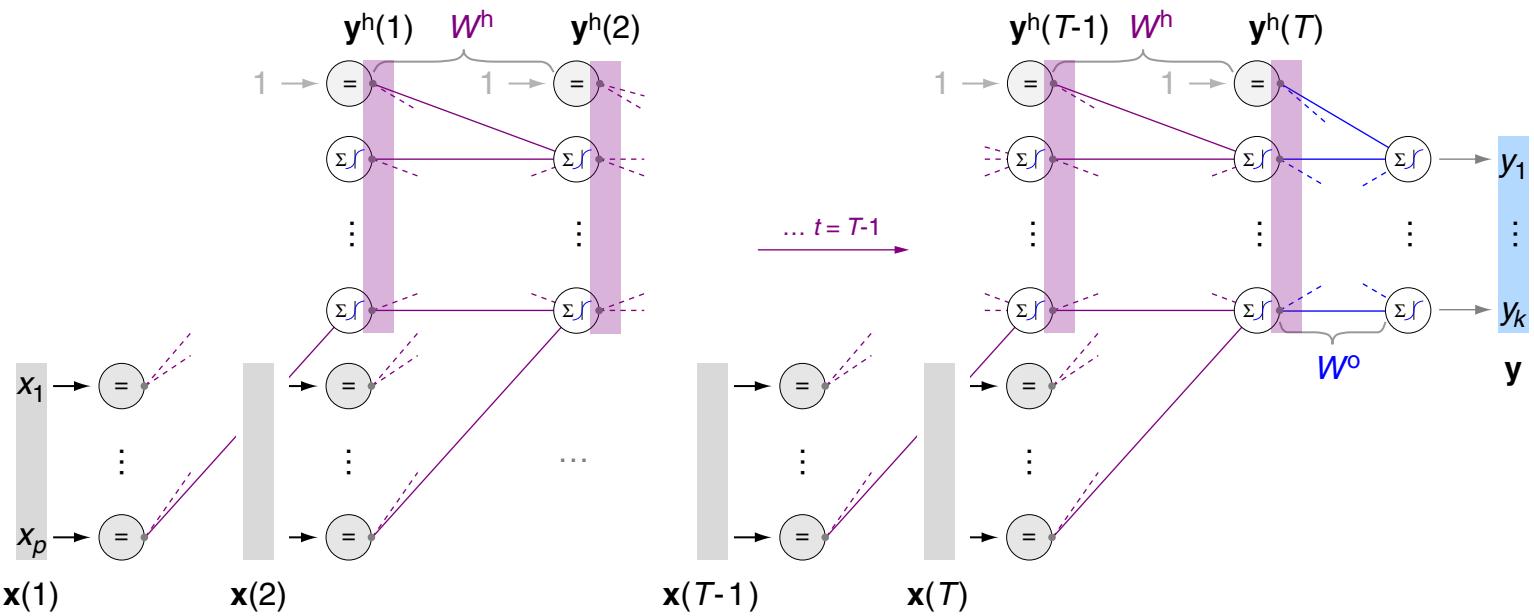
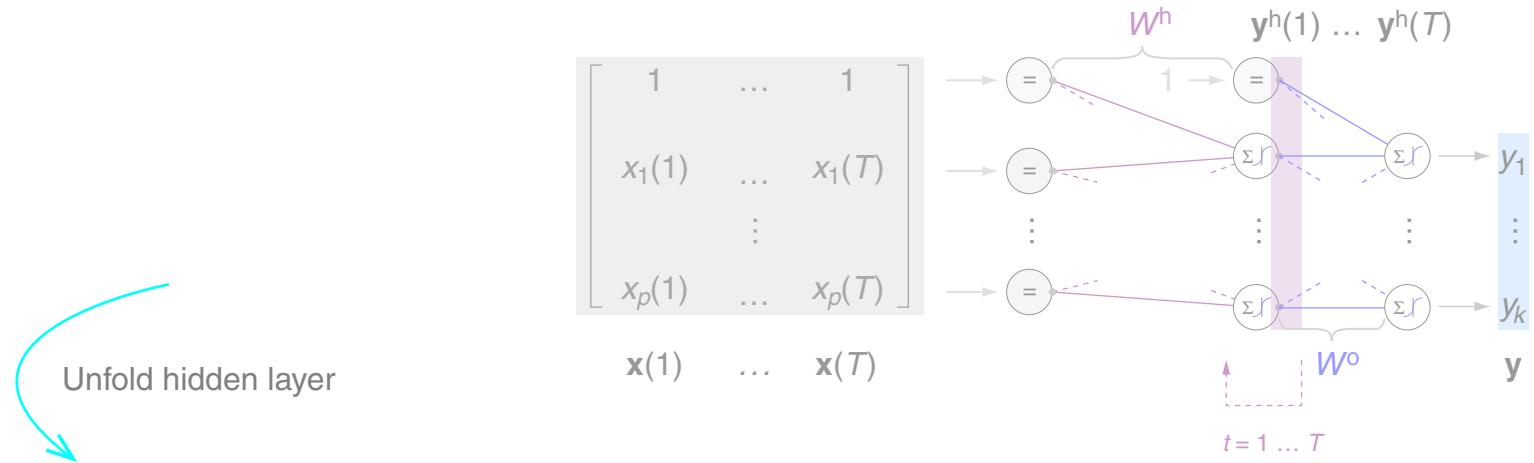
RNN Sequence Encoding (continued)



- Sequence of p -dimensional input vectors $[\mathbf{x}(1), \dots, \mathbf{x}(T)]$.
- One hidden layer that is recurrently updated.
- One k -dimensional output vector $\mathbf{y}([\mathbf{x}(1), \dots, \mathbf{x}(T)])$ or \mathbf{y} .

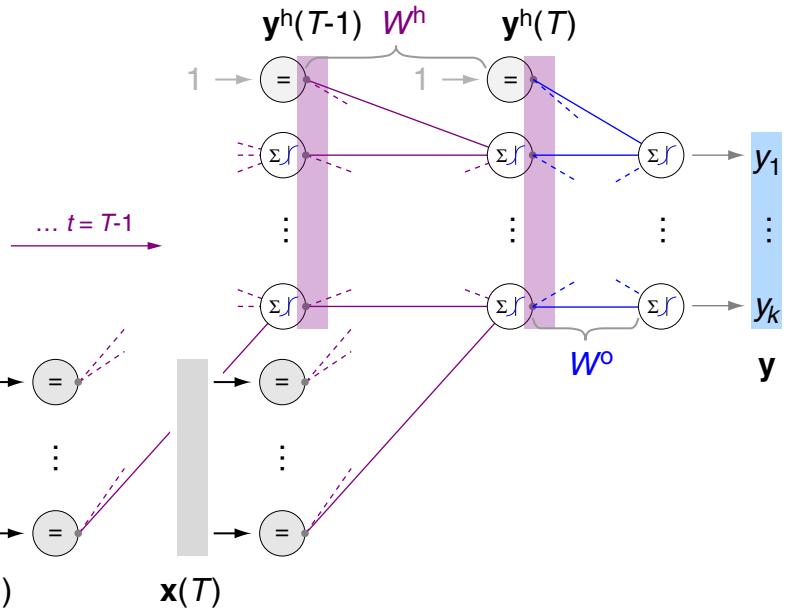
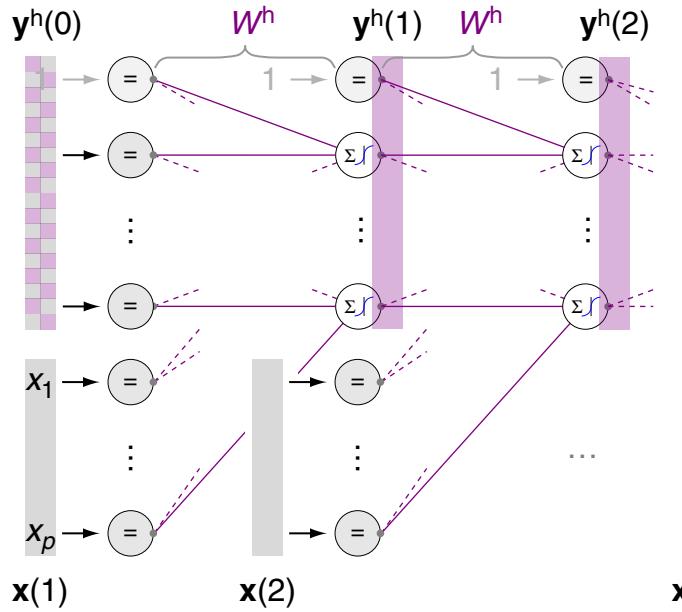
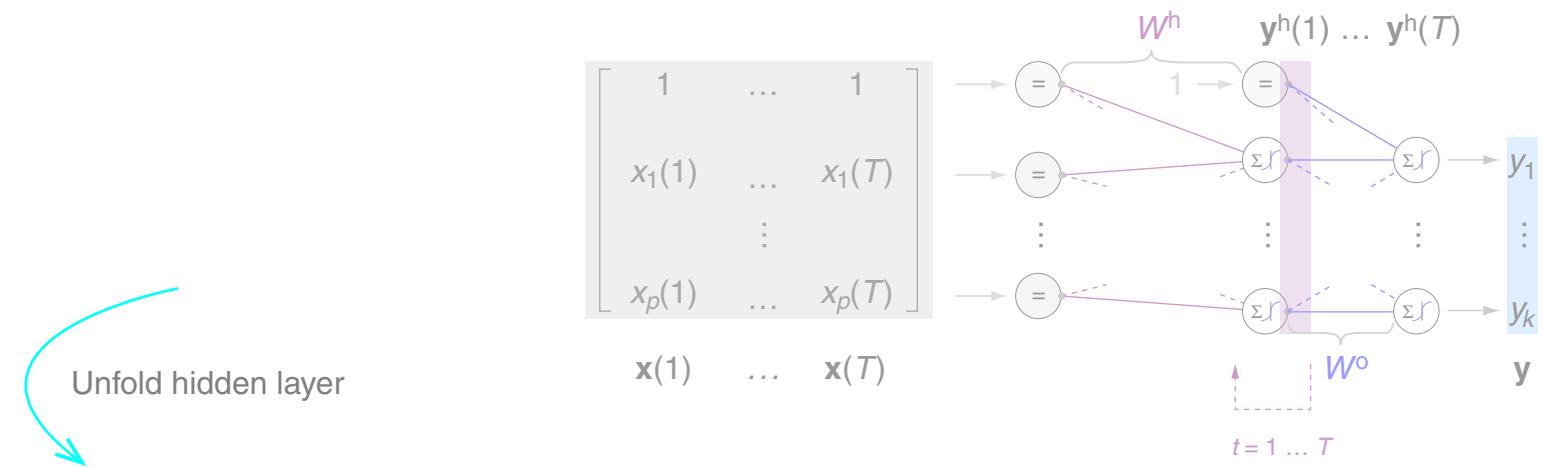
Recurrent Neural Networks

RNN Sequence Encoding (continued)



Recurrent Neural Networks

RNN Sequence Encoding (continued)



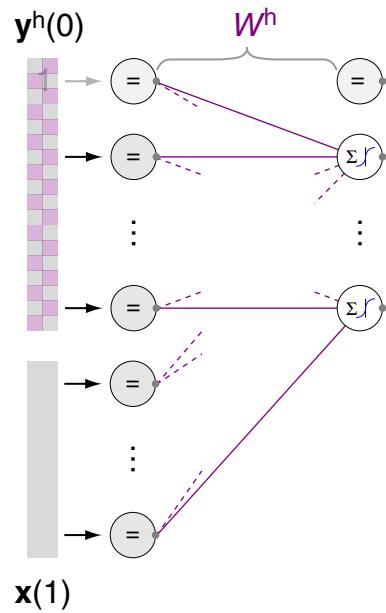
Remarks:

- An input sequence is written in brackets, $[x(1), \dots, x(T)]$, where $x(t), t = 1, \dots, T$, denotes the input vector at time step t .
- The words in the input sequence are usually one-hot-encoded, i.e., by a p -dimensional input vector with a “1” whose position indicates the word, and zeros elsewhere.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [\[encoding overview\]](#)

t: 0 → 1

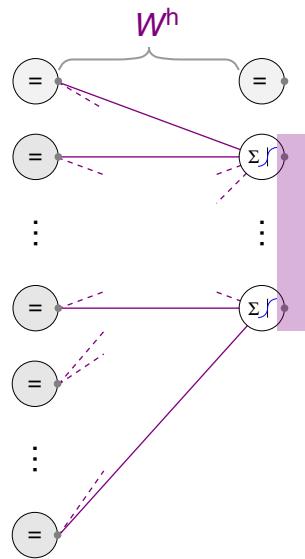


Input encoding over t .

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

$t: 0 \rightarrow 1$

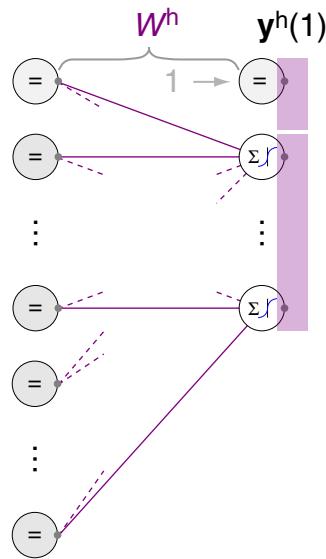


Input encoding over t .

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1

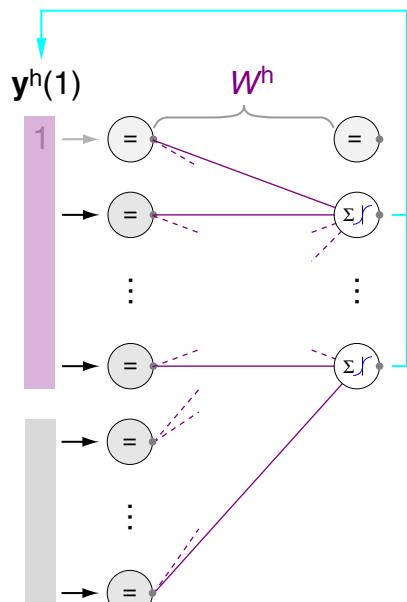


Input encoding over t .

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1

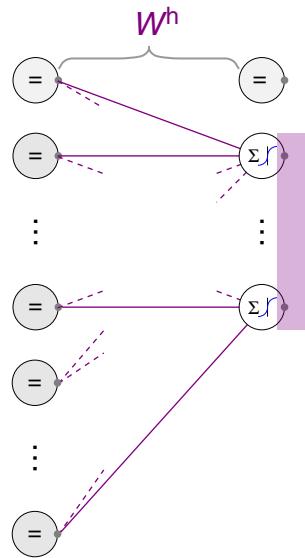


Input encoding over t .

Recurrent Neural Networks

RNN Sequence Encoding (continued) [\[encoding overview\]](#)

t: 1 → 2

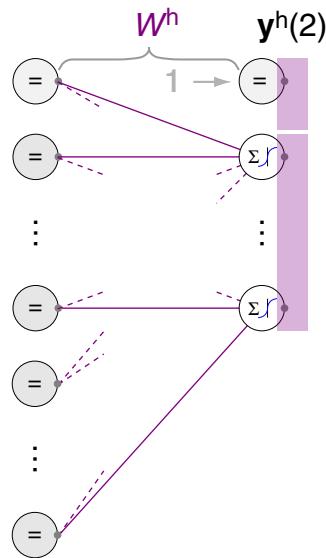


Input encoding over t .

Recurrent Neural Networks

RNN Sequence Encoding (continued) [\[encoding overview\]](#)

t: 1 → 2

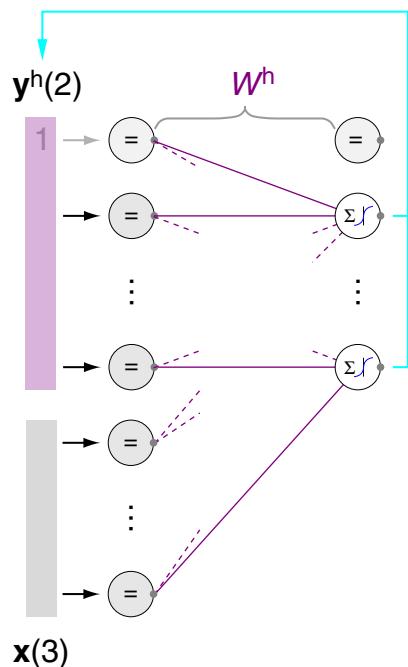


Input encoding over t .

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

$t: 1 \rightarrow 2$

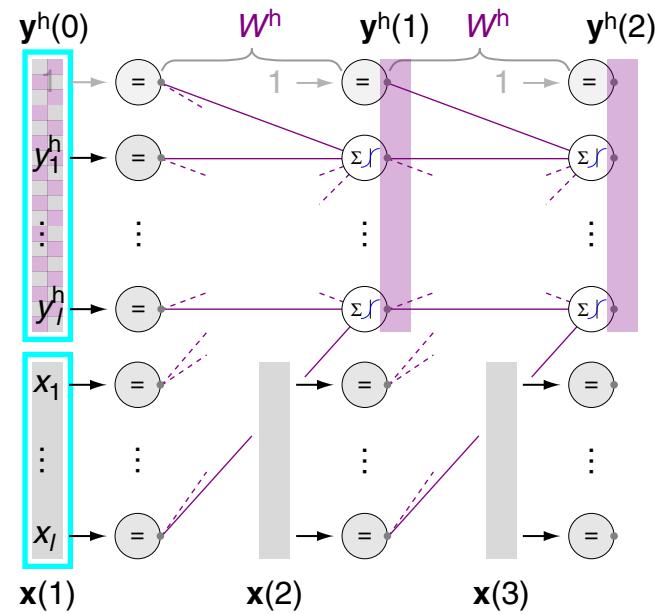
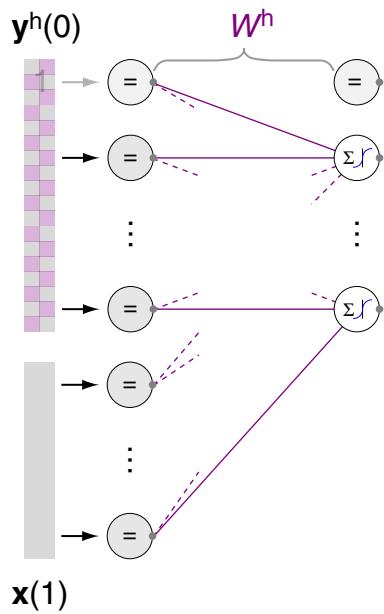


Input encoding over t .

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1



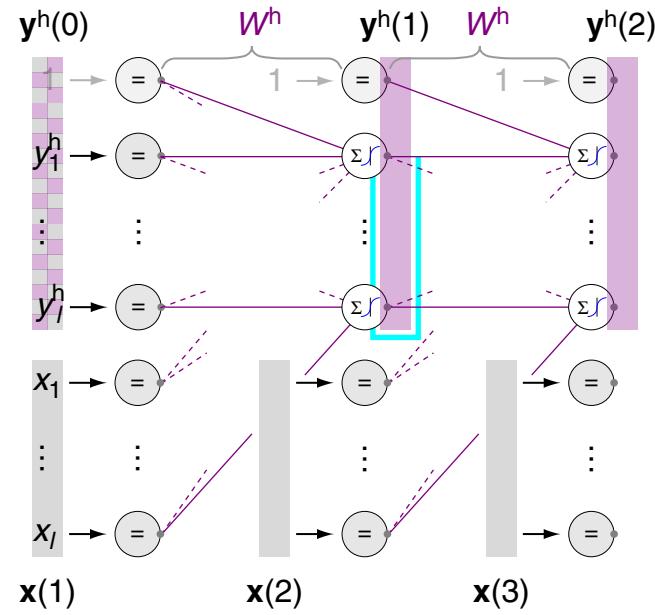
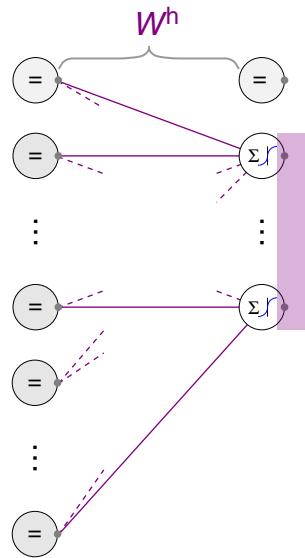
Input encoding over t .

The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1



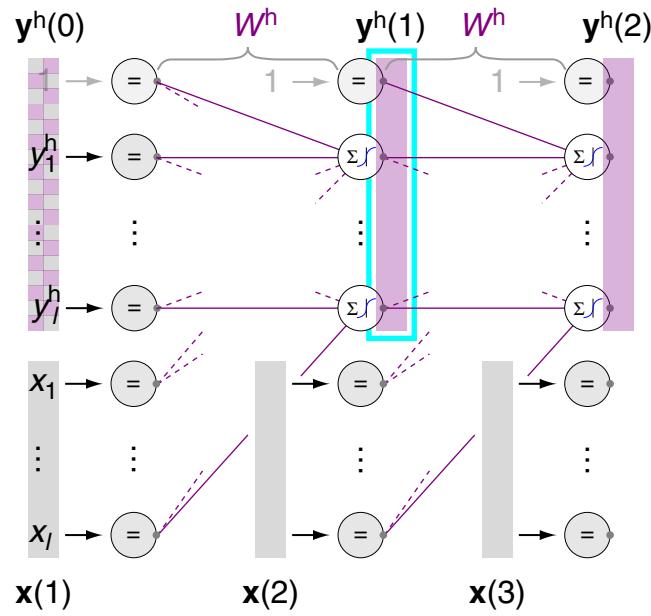
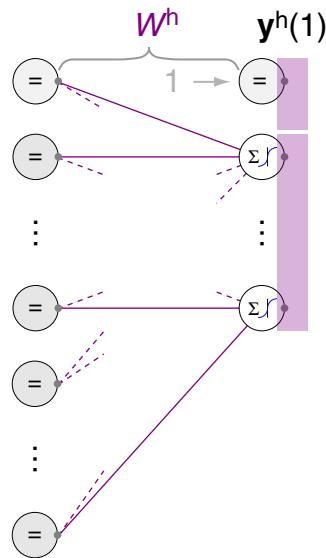
Input encoding over t .

The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: $0 \rightarrow 1$



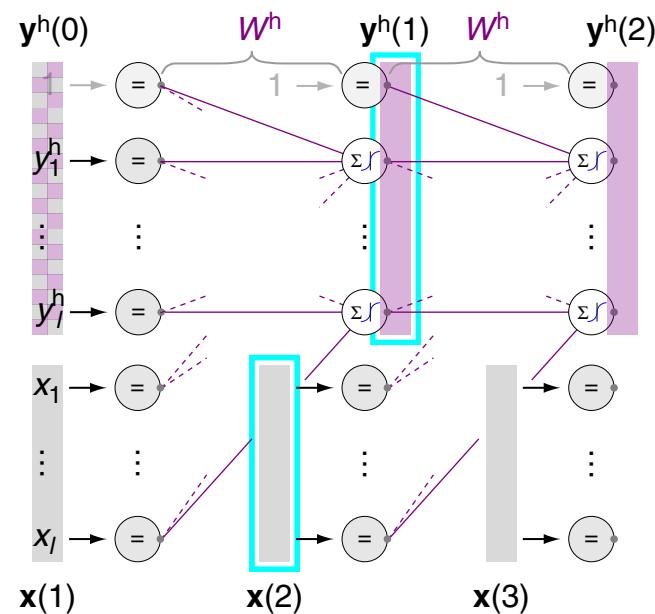
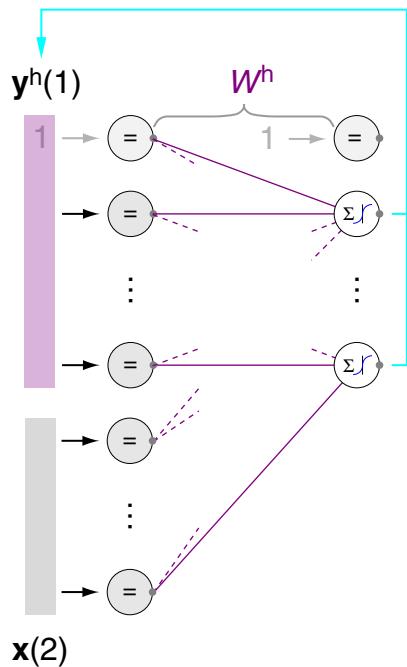
Input encoding over t .

The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1



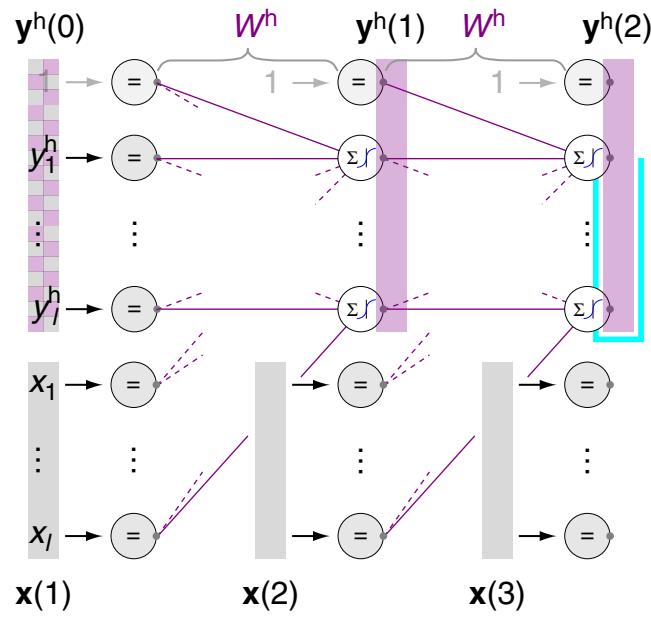
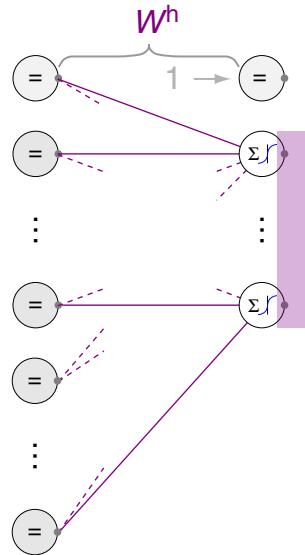
Input encoding over t .

The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: $1 \rightarrow 2$



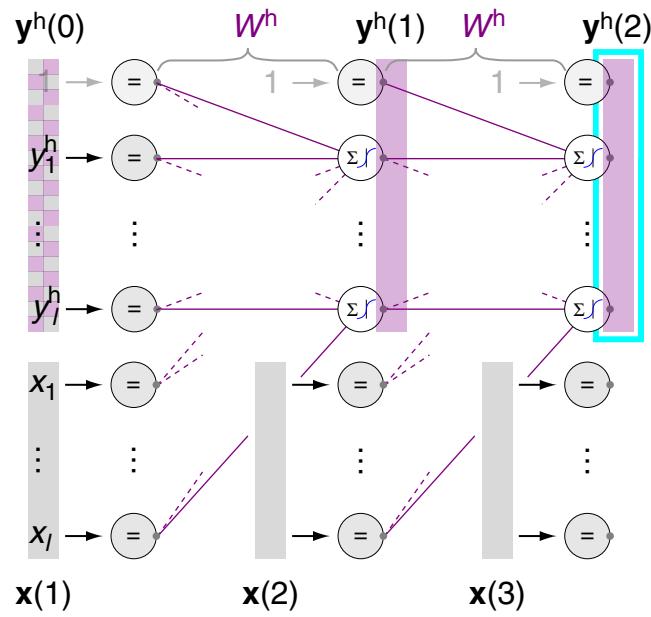
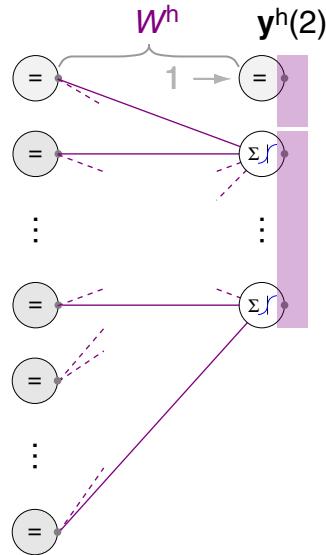
Input encoding over t .

The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: $1 \rightarrow 2$



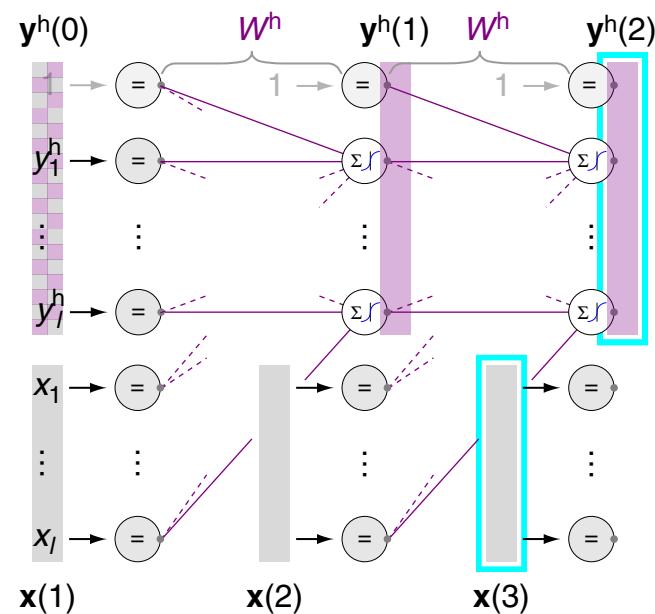
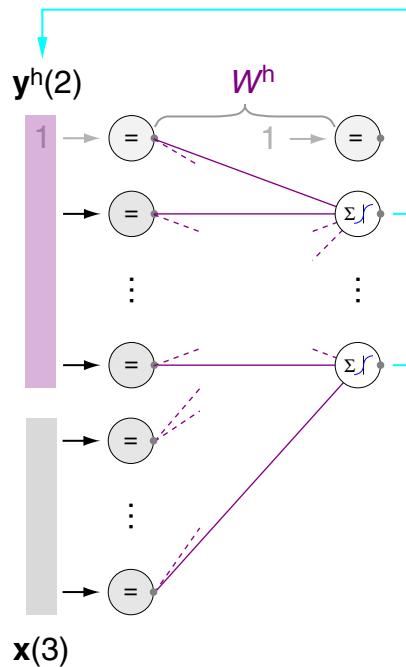
Input encoding over t .

The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 1 → 2



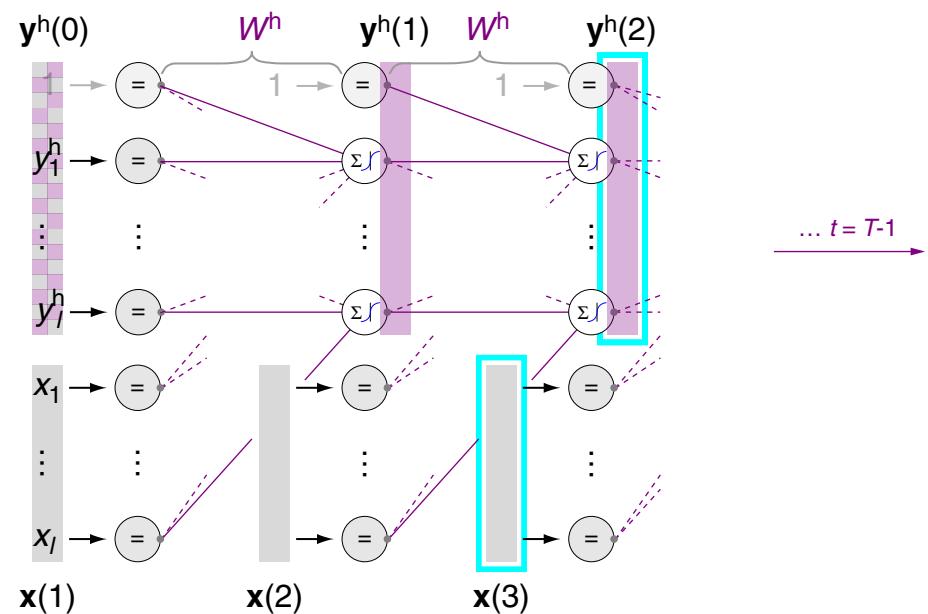
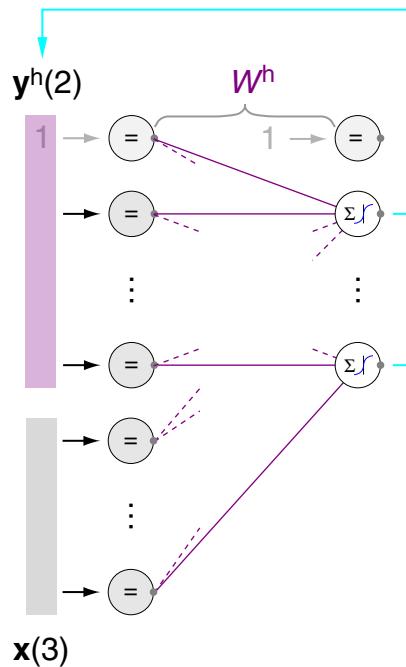
Input encoding over t .

The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

$t: 1 \rightarrow 2$



Input encoding over t .

The hidden layer at subsequent time steps.

Recurrent Neural Networks

(S1) Sequence-to-Class: Sentiment Classification

- I love my cat. $\rightarrow \oplus$
- Cats and dogs lap water. $\rightarrow \oplus$
- It is raining cats and dogs. $\rightarrow \ominus$
- Cats and dogs are not allowed. $\rightarrow \ominus$
- Cats and dogs have always been natural enemies. $\rightarrow \ominus$

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water)

Recurrent Neural Networks

(S1) Sequence-to-Class: Sentiment Classification (continued)

- I love my cat. $\rightarrow \oplus$
- Cats and dogs lap water. $\rightarrow \oplus$
- It is raining cats and dogs. $\rightarrow \ominus$
- Cats and dogs are not allowed. $\rightarrow \ominus$
- Cats and dogs have always been natural enemies. $\rightarrow \ominus$

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water)

Input:
$$[\mathbf{x}(1), \dots, \mathbf{x}(4)] = \left[\begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix} \right]$$
$$\hat{=} [\text{word_11}, \text{word_15}, \text{word_16}, \text{word_6}]$$
$$\hat{=} \text{I love my cat}$$

Recurrent Neural Networks

(S1) Sequence-to-Class: Sentiment Classification (continued)

- I love my cat. $\rightarrow \oplus$
- Cats and dogs lap water. $\rightarrow \oplus$
- It is raining cats and dogs. $\rightarrow \ominus$
- Cats and dogs are not allowed. $\rightarrow \ominus$
- Cats and dogs have always been natural enemies. $\rightarrow \ominus$

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water)

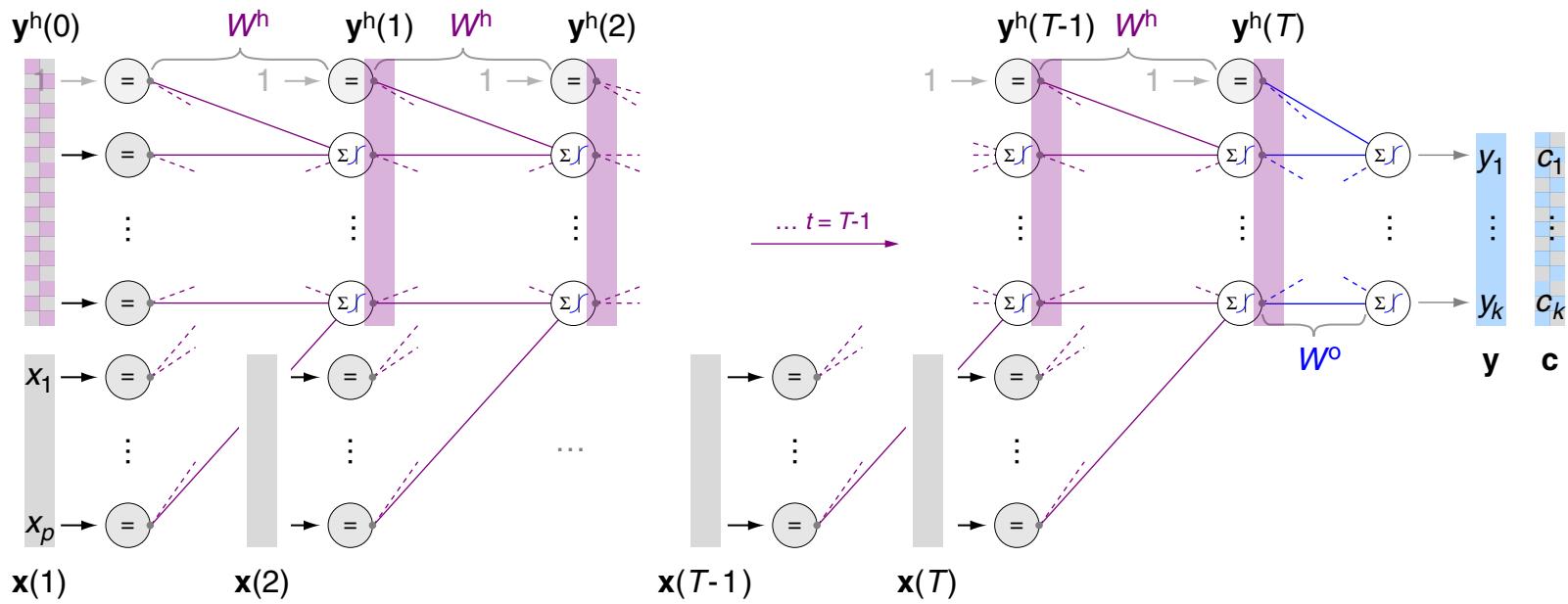
Input:
$$[\mathbf{x}(1), \dots, \mathbf{x}(4)] = \left[\begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix} \right]$$
$$\hat{=} [\text{word_11}, \text{word_15}, \text{word_16}, \text{word_6}]$$
$$\hat{=} \text{I love my cat}$$

Output:
$$\mathbf{y}([\mathbf{x}(1), \dots, \mathbf{x}(4)]) = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

Target:
$$\mathbf{c} = \begin{pmatrix} \oplus \\ \cdot \end{pmatrix}$$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs



Input:

$$[\mathbf{x}(1), \dots, \mathbf{x}(T)]$$

Hidden:

$$\mathbf{y}^h(t) = \sigma \left(W^h \begin{pmatrix} \mathbf{y}^h(t-1) \\ \mathbf{x}(t) \end{pmatrix} \right), t = 1, \dots, T$$

Target:

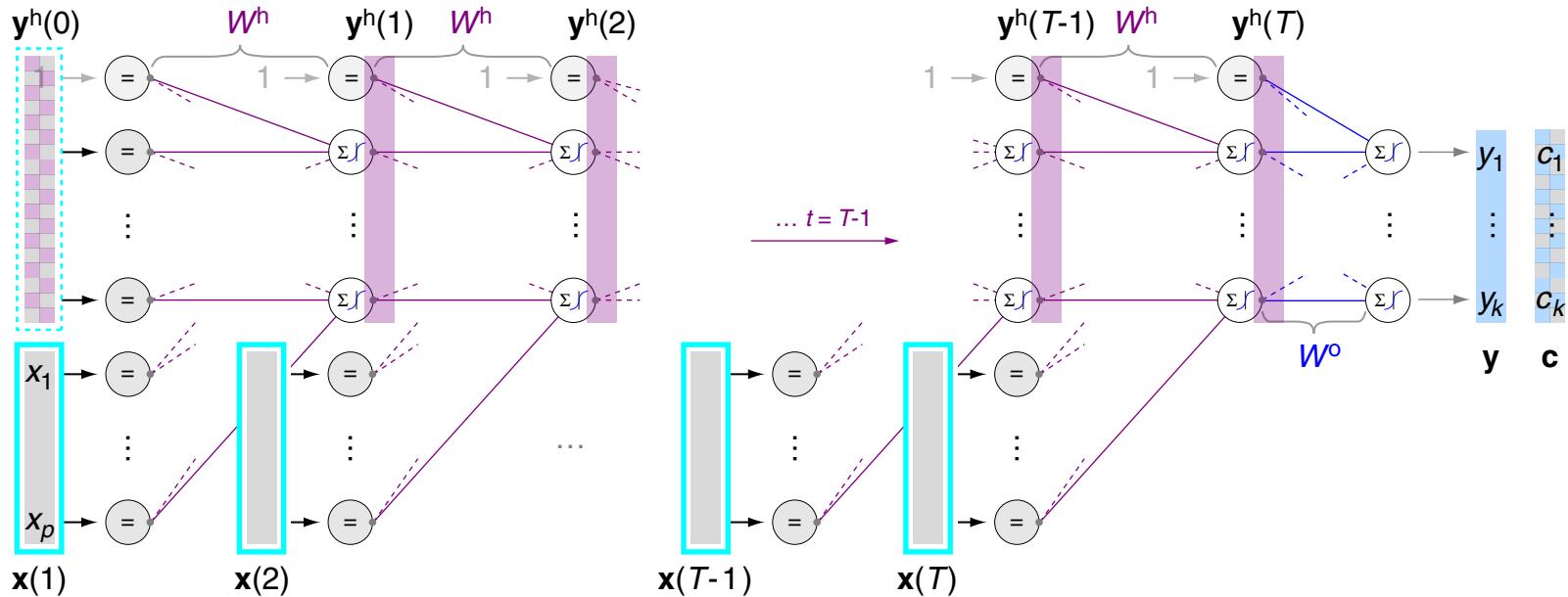
$$\mathbf{c}$$

Output:

$$\mathbf{y} = \sigma_1 (W^o \mathbf{y}^h(T))$$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs (continued)



Input:

$$[\mathbf{x}(1), \dots, \mathbf{x}(T)]$$

Output:

$$\mathbf{y} = \sigma_1(W^o \mathbf{y}^h(T))$$

Hidden:

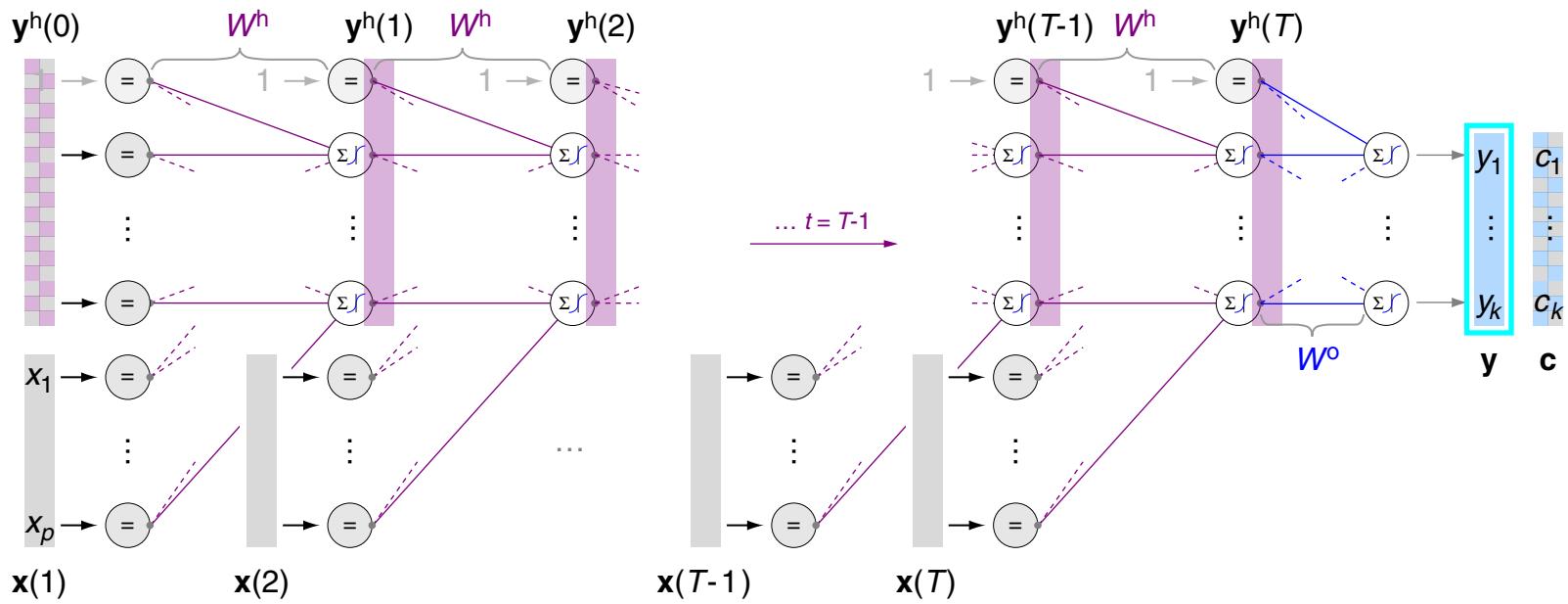
$$\mathbf{y}^h(t) = \sigma \left(W^h \begin{pmatrix} \mathbf{y}^h(t-1) \\ \mathbf{x}(t) \end{pmatrix} \right), t = 1, \dots, T$$

Target:

$$\mathbf{c}$$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs (continued)



Input:

$$[\mathbf{x}(1), \dots, \mathbf{x}(T)]$$

Hidden:

$$\mathbf{y}^h(t) = \sigma \left(W^h \begin{pmatrix} \mathbf{y}^h(t-1) \\ \mathbf{x}(t) \end{pmatrix} \right), t = 1, \dots, T$$

Target:

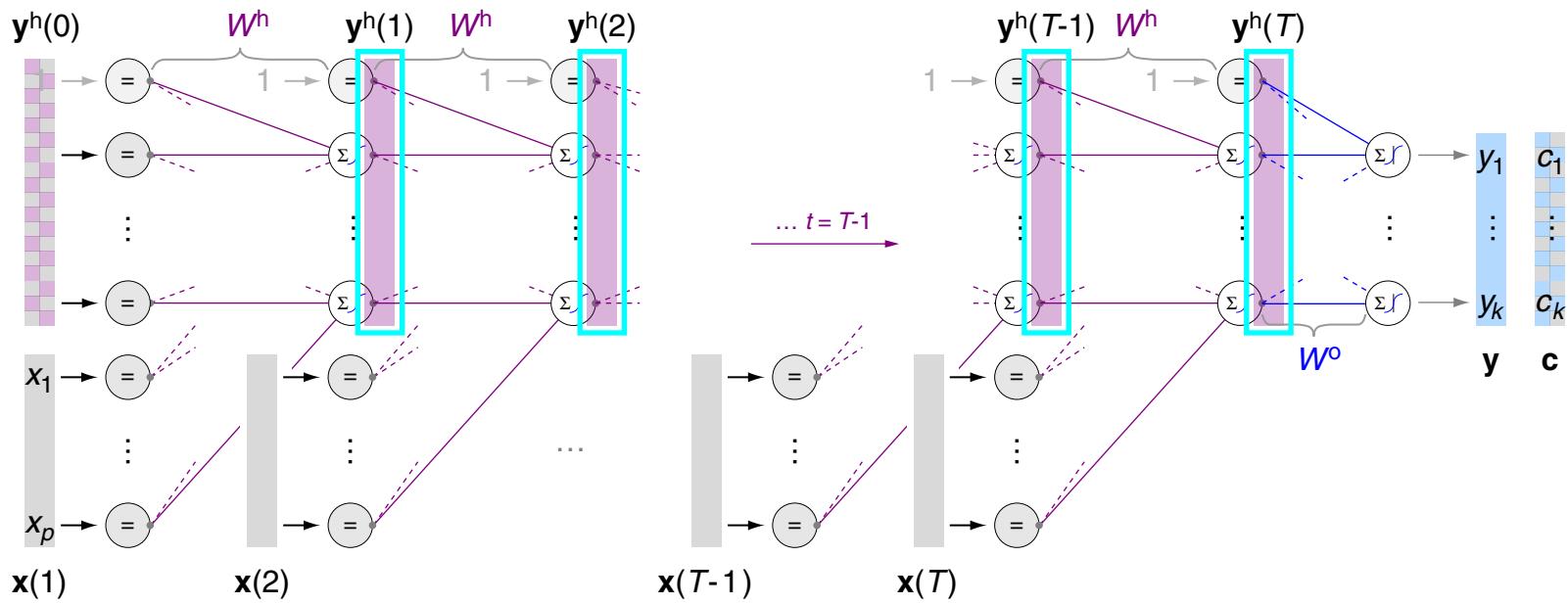
$$\mathbf{c}$$

Output:

$$\mathbf{y} = \sigma_1 (W^o \mathbf{y}^h(T))$$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs (continued)



Input:

$$[x(1), \dots, x(T)]$$

Hidden:

$$y^h(t) = \sigma \left(W^h \begin{pmatrix} y^h(t-1) \\ x(t) \end{pmatrix} \right), t = 1, \dots, T$$

Target:

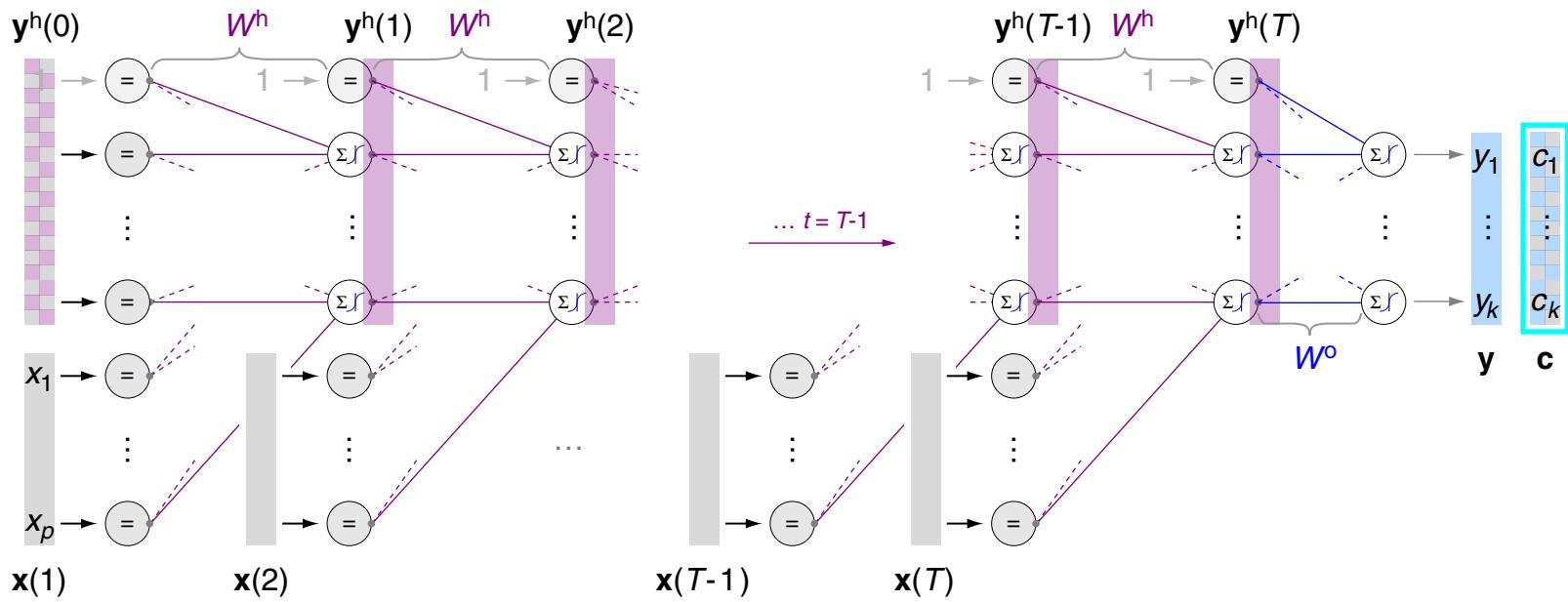
$$c$$

Output:

$$y = \sigma_1 (W^o y^h(T))$$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs (continued)



Input:

$$[\mathbf{x}(1), \dots, \mathbf{x}(T)]$$

Output:

$$\mathbf{y} = \sigma_1 (W^o \mathbf{y}^h(T))$$

Hidden:

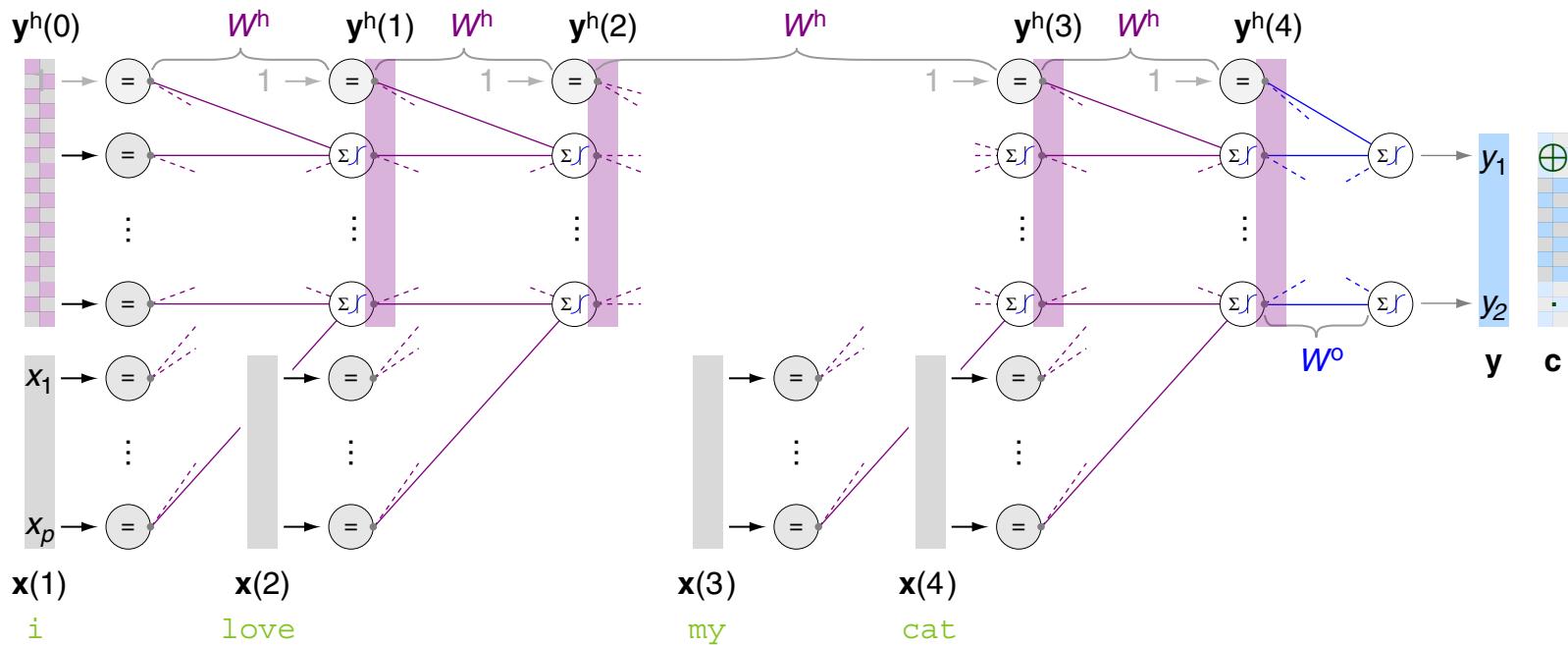
$$\mathbf{y}^h(t) = \sigma \left(W^h \begin{pmatrix} \mathbf{y}^h(t-1) \\ \mathbf{x}(t) \end{pmatrix} \right), t = 1, \dots, T$$

Target:

$$\mathbf{c}$$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs (continued)



Input:

$$[x(1), \dots, x(4)]$$

Output:

$$y = \sigma_1 (W^o y^h(4))$$

Hidden:

$$y^h(t) = \sigma \left(W^h \begin{pmatrix} y^h(t-1) \\ x(t) \end{pmatrix} + b \right), t = 1, \dots, 4$$

Target:

$$c$$

Remarks:

- We denote $y^h(0)$ not as input since this kind of predefined hidden vector does not contain any “actual knowledge”, but is usually initialized as vector of zeros.
- To keep the illustrations clear we use the bag-of-words model for representing (= embedding) the words as vectors $x(t)$.

In practice, however, one considers semantically stronger (language-model-based) embeddings, which also encode information about neighborhoods and occurrence probabilities. In this regard, either a previously computed embedding can be used, or the embedding can be learned along with the task, end-to-end.

- Recap. $\sigma_1()$ denotes the softmax function. $\sigma_1 : \mathbf{R}^k \rightarrow \Delta^{k-1}$, generalizes the logistic (sigmoid) function to k dimensions (to k exclusive classes), where $\sigma_1(\mathbf{z})|_i = e^{z_i} / \sum_{j=1}^k e^{z_j}$. [\[Wikipedia\]](#)
 $\Delta^{k-1} \subset \mathbf{R}^k$ denotes the standard $k-1$ -simplex, which contains all k -tuples with non-negative elements that sum to 1. [\[Wikipedia\]](#)

Recurrent Neural Networks

The IGD Algorithm for Sequence-to-Class Tasks [IGD_{c2seq}]

Algorithm: IGD_{seq2c} Incremental Gradient Descent for RNNs at seq2class tasks.

Input: D Multiset of examples $([\mathbf{x}(1), \dots, \mathbf{x}(T)], \mathbf{c})$ with $\mathbf{x}(t) \in \mathbf{R}^p$, $\mathbf{c} \in \{0, 1\}^k$.
 η Learning rate, a small positive constant.

Output: W^h, W^o Weight matrices. (= hypothesis)

1. *initialize_random_weights(W^h, W^o)*, $\mathbf{y}^h(0) = (0, \dots, 0)^T$, $t_{epoch} = 0$

2. **REPEAT**

3. $t_{epoch} = t_{epoch} + 1$

4. **FOREACH** $([\mathbf{x}(1), \dots, \mathbf{x}(T)], \mathbf{c}) \in D$ DO

5.



Model function evaluation.

6.



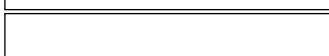
Calculation of residual vector.

7a.



Calculation of derivative of the loss.

7b.



Parameter update $\hat{=}$ one gradient step down.

8.

9. **ENDDO**

10. **UNTIL**(convergence($D, \mathbf{y}(\cdot), t_{epoch}$))

11. **return**(W^h, W^o)

Recurrent Neural Networks

The IGD Algorithm for Sequence-to-Class Tasks (continued) [IGD_{c2seq}]

Algorithm: IGD_{seq2c} Incremental Gradient Descent for RNNs at seq2class tasks.

Input: D Multiset of examples $([\mathbf{x}(1), \dots, \mathbf{x}(T)], \mathbf{c})$ with $\mathbf{x}(t) \in \mathbf{R}^p$, $\mathbf{c} \in \{0, 1\}^k$.
 η Learning rate, a small positive constant.

Output: W^h, W^o Weight matrices. (= hypothesis)

1. *initialize_random_weights(W^h, W^o)*, $\mathbf{y}^h(0) = (0, \dots, 0)^T$, $t_{epoch} = 0$
2. **REPEAT**
3. $t_{epoch} = t_{epoch} + 1$
4. **FOREACH** $([\mathbf{x}(1), \dots, \mathbf{x}(T)], \mathbf{c}) \in D$ **DO**
5. **FOR** $t = 1$ **TO** T **DO** // forward propagation

$$\mathbf{y}^h(t) = \sigma \left(W^h \begin{pmatrix} \mathbf{y}^h(t-1) \\ \mathbf{x}(t) \end{pmatrix} \right)$$

 ENDDO
 $\mathbf{y} = \sigma_1(W^o \mathbf{y}^h(T))$
6. $\delta = \mathbf{c} - \mathbf{y}([\mathbf{x}(1), \dots, \mathbf{x}(T)])$
- 7a. $\ell(\mathbf{w}) = l(\delta) + \frac{\lambda}{n} R(\mathbf{w})$, $\nabla \ell(\mathbf{w}) = \text{autodiff}(\ell(), \mathbf{w})$ // backpropagation (7a+7b)
- 7b. $\Delta W^h = \eta \cdot \nabla^h \ell(\mathbf{w})$, $\Delta W^o = \eta \cdot \nabla^o \ell(\mathbf{w})$
8. $W^h = W^h + \Delta W^h$, $W^o = W^o + \Delta W^o$
9. **ENDDO**
10. **UNTIL**(convergence($D, \mathbf{y}(\cdot), t_{epoch}$))
11. **return**(W^h, W^o)

Recurrent Neural Networks

Types of Learning Tasks [Recap]

(S1) sequence → class

sentence → $\{\oplus, \ominus\}$

i love my cat → \oplus

(S2) class → sequence

$\{\oplus, \ominus\} \rightarrow$ sentence

$\oplus \rightarrow$ i love my cat

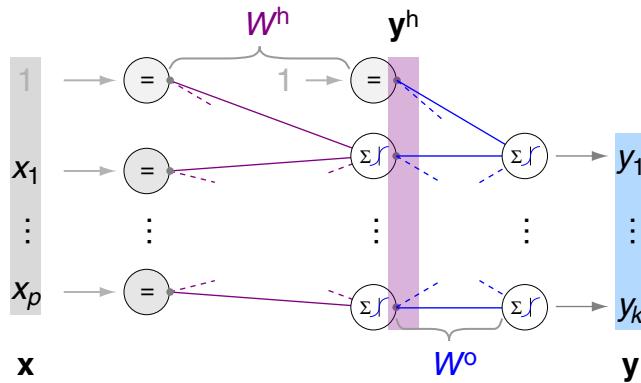
(S3) sequence → sequence

English sentence → German sentence

i love my cat → ich liebe meine katze

Recurrent Neural Networks

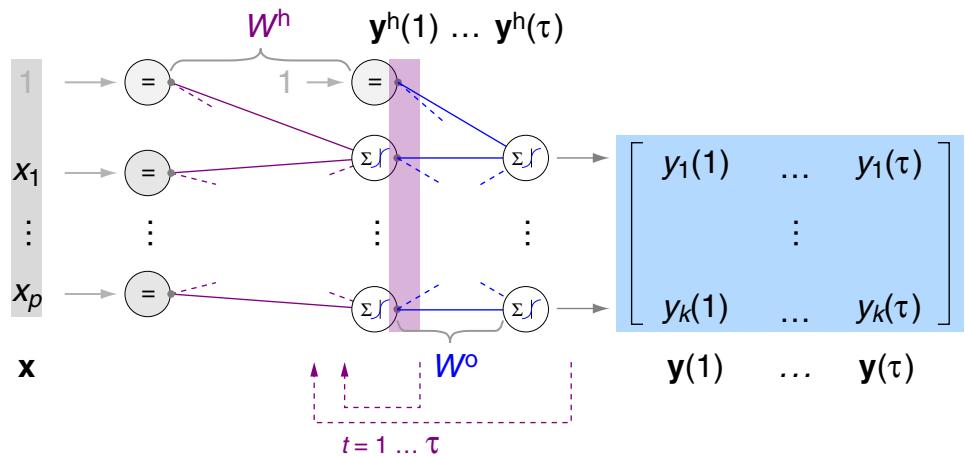
RNN Sequence Decoding



- One p -dimensional input vector \mathbf{x} .
- One hidden layer (general: $d-1$ hidden layers, i.e., d active layers).
- One k -dimensional output vector $\mathbf{y}(\mathbf{x})$.

Recurrent Neural Networks

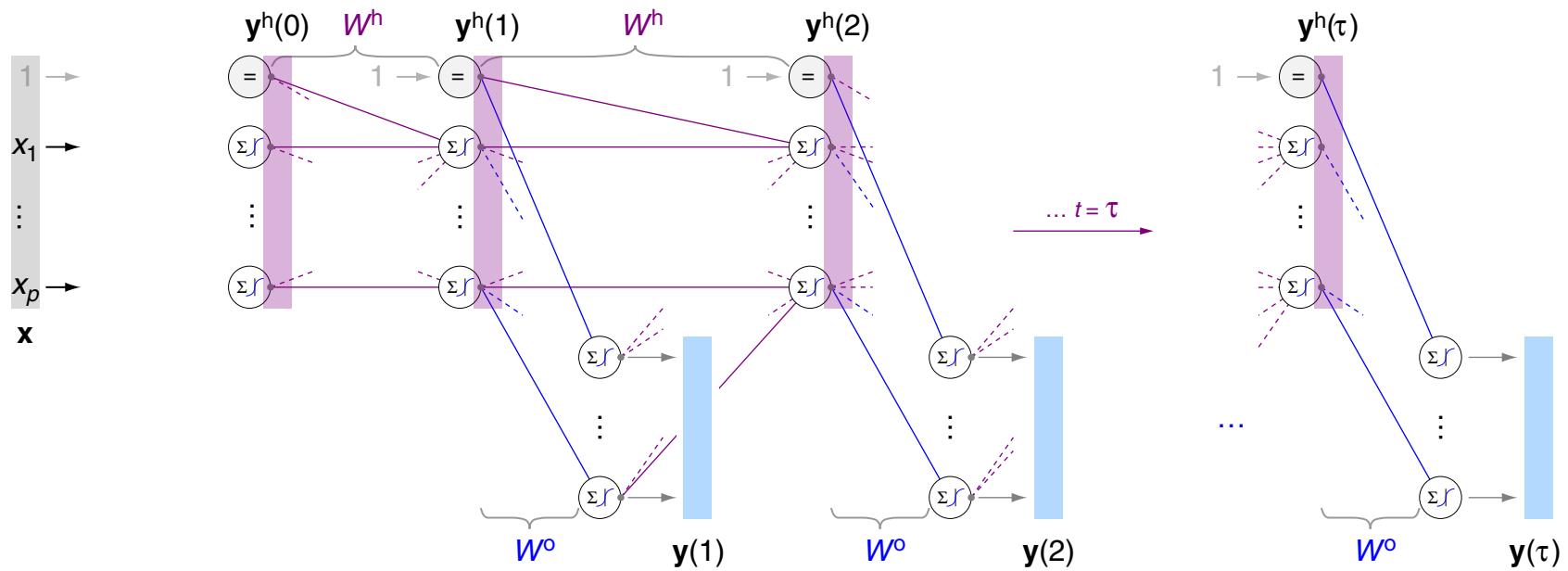
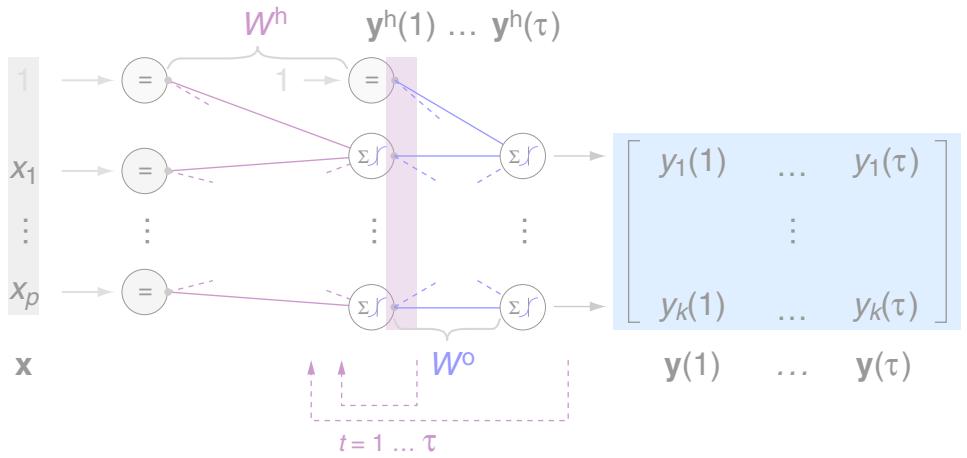
RNN Sequence Decoding (continued)



- One p -dimensional input vector \mathbf{x} .
- One hidden and one output layer, which are recurrently updated.
- Sequence of k -dimen. output vectors $[\mathbf{y}(\mathbf{x}, 1), \dots, \mathbf{y}(\mathbf{x}, \tau)]$ or $[\mathbf{y}(1), \dots, \mathbf{y}(\tau)]$.

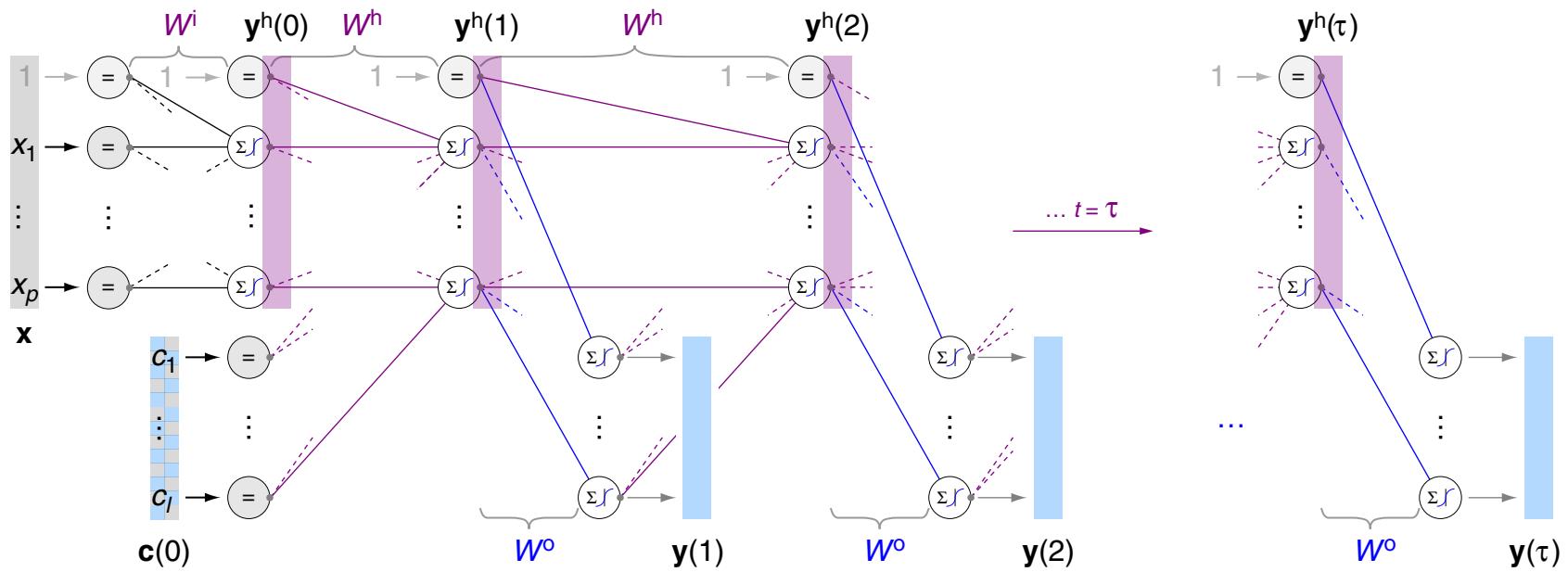
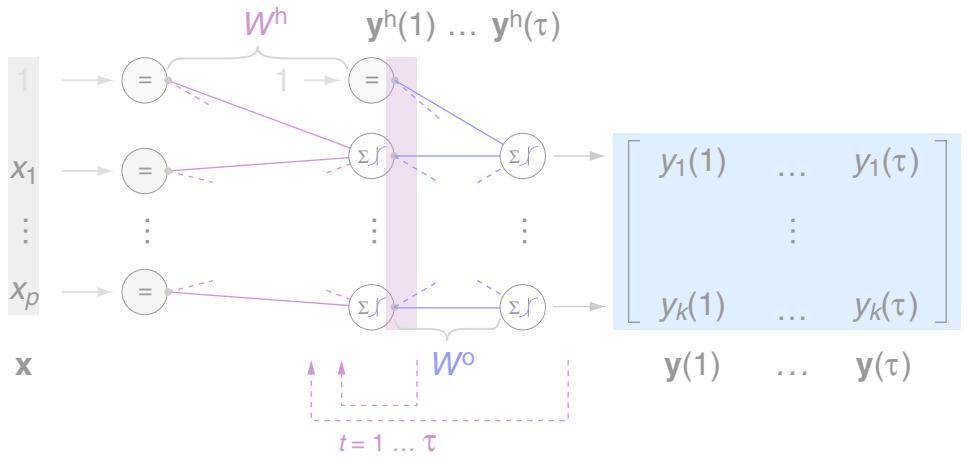
Recurrent Neural Networks

RNN Sequence Decoding (continued)



Recurrent Neural Networks

RNN Sequence Decoding (continued)



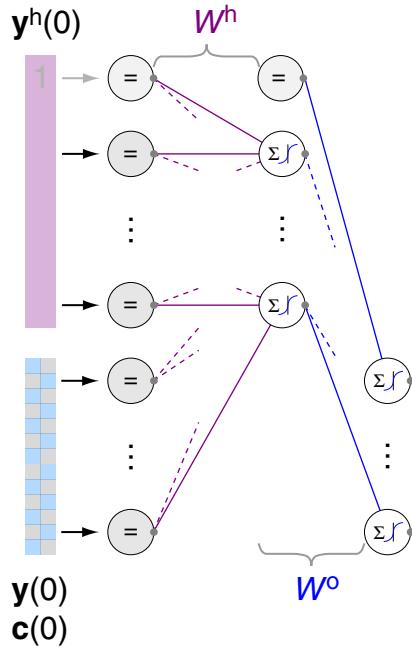
Remarks:

- An output sequence is written in brackets, $[\mathbf{y}(1), \dots, \mathbf{y}(\tau)]$, where $\mathbf{y}(t)$, $t = 1, \dots, \tau$, denotes the output vector at time step t .
- The words in the output sequence are usually one-hot-encoded, i.e., by a k -dimensional output vector with a “1” whose position indicates the word, and zeros elsewhere.
- If the input, \mathbf{x} , is clear from the context, we usually note $\mathbf{y}(\mathbf{x}, t)$ as $\mathbf{y}(t)$.
- The matrix W^i is necessary to embed the typically low-dimensional input vector \mathbf{x} regarding the high-dimensional hidden vectors \mathbf{y}^h : $\mathbf{y}^h(0) = \sigma(W^i \mathbf{x})$.
- The parameter τ in $\mathbf{y}(\tau)$ is unknown. More specifically, the generation process terminates at that time step τ for which $\mathbf{y}(\tau) = (0, 0, \dots, 0, 1)^T$ ($\hat{=}$ `<end>`).
 τ (the length of the computed output) does not have to be equal to T (the length of the target / ground truth).

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1

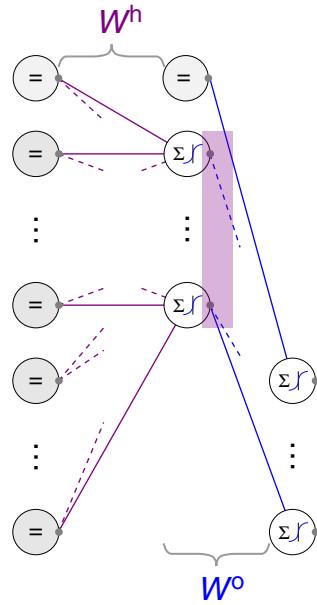


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

$t: 0 \rightarrow 1$

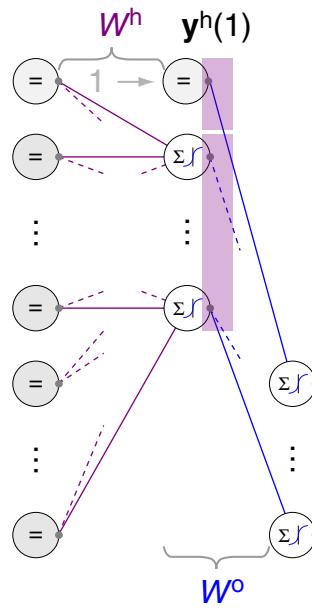


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1

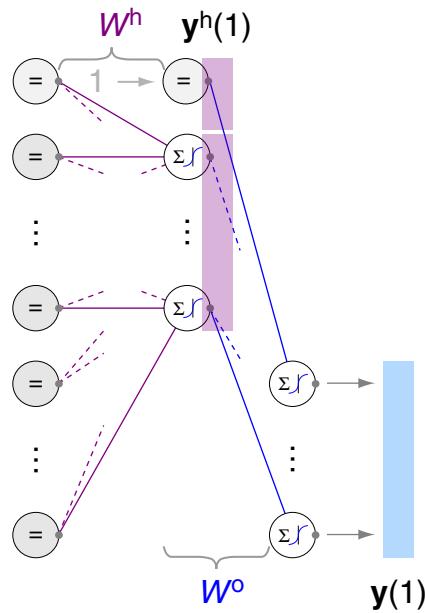


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1

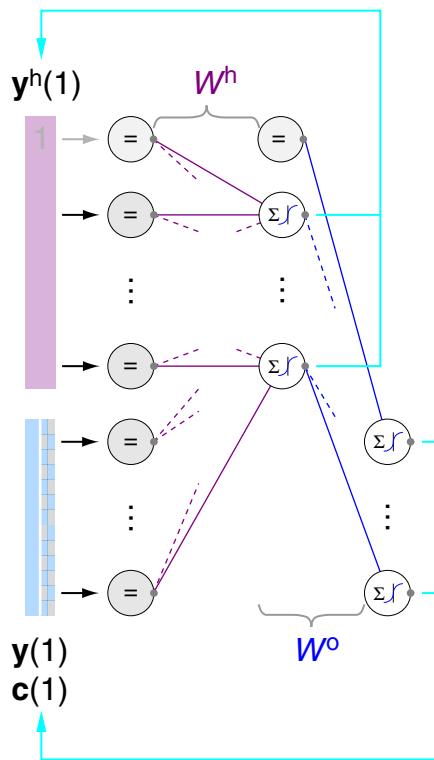


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1

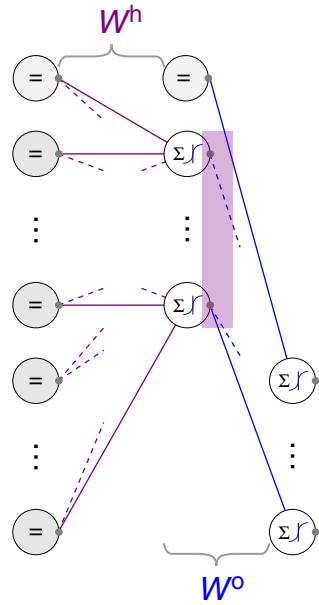


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $1 \rightarrow 2$

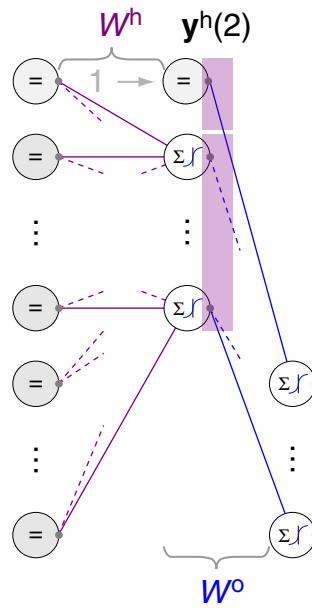


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $1 \rightarrow 2$

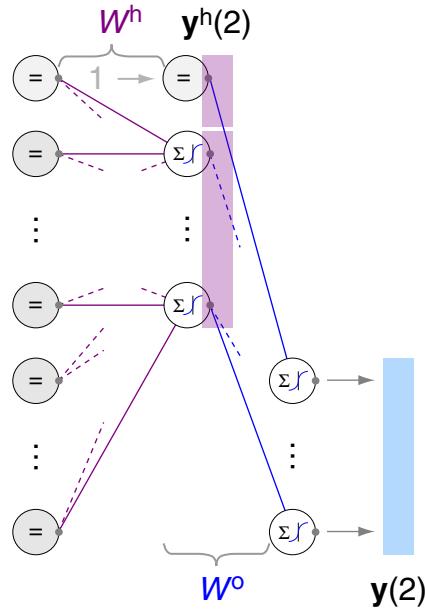


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2

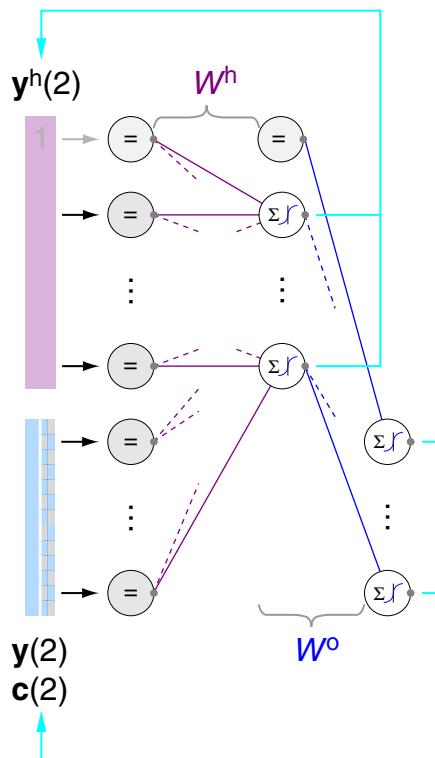


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2

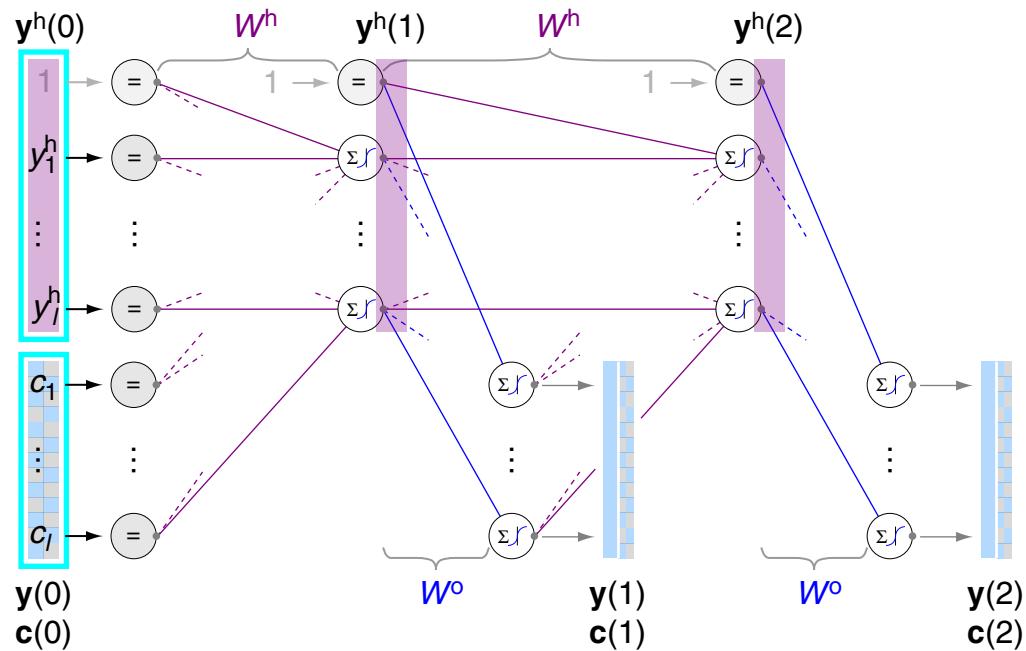
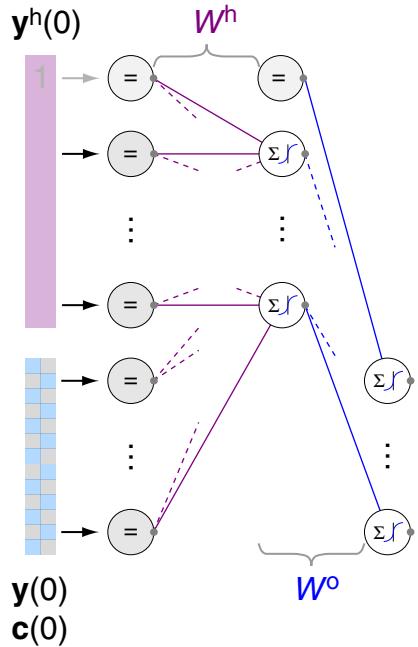


Output decoding over t .

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1



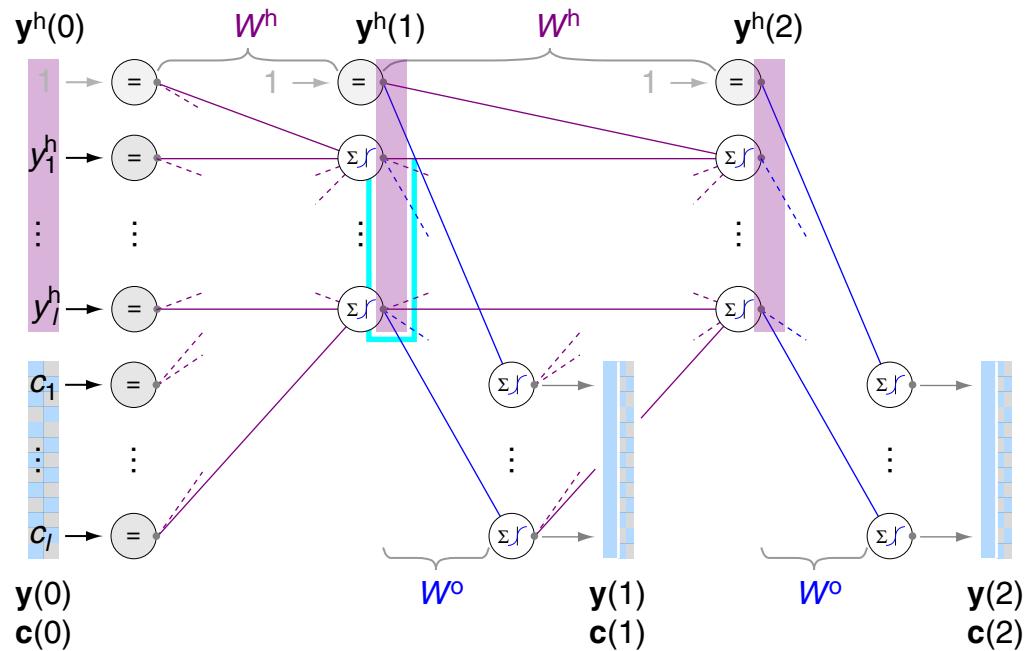
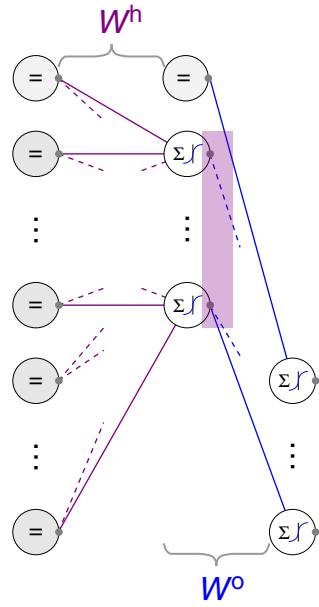
Output decoding over t .

Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $0 \rightarrow 1$



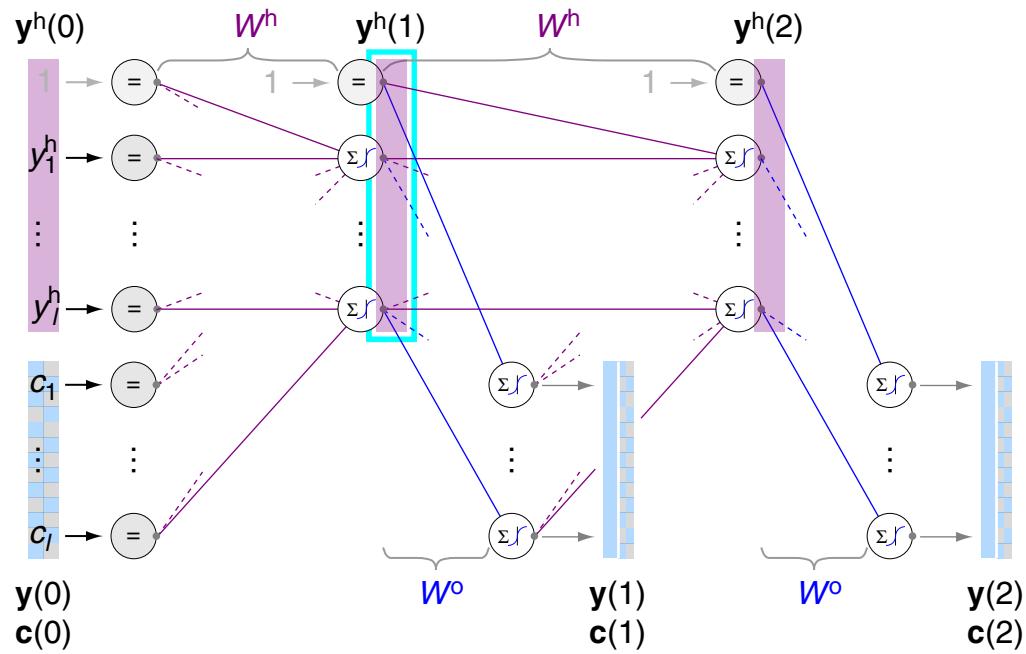
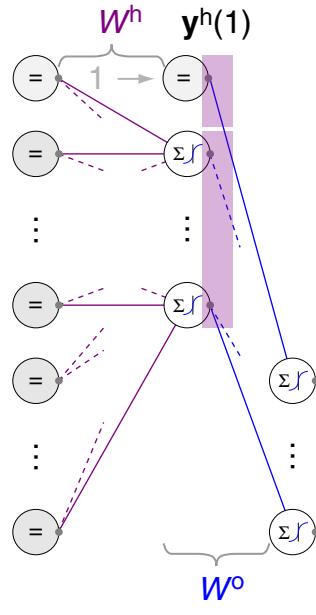
Output decoding over t .

Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $0 \rightarrow 1$



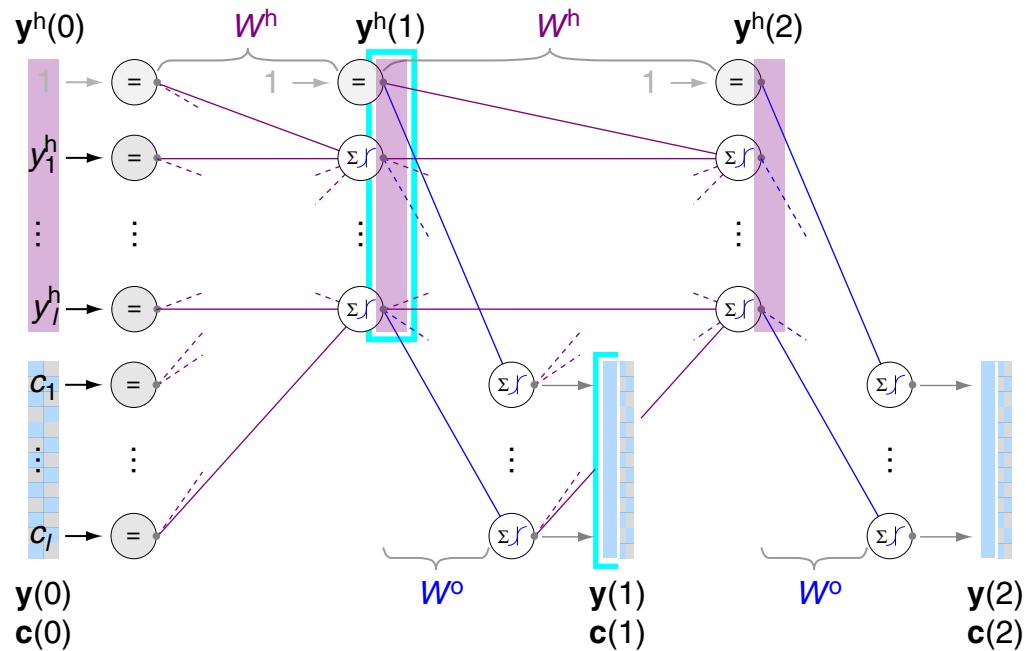
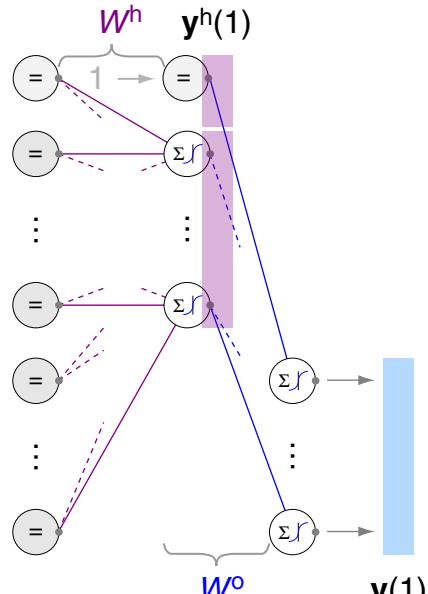
Output decoding over t .

Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $0 \rightarrow 1$



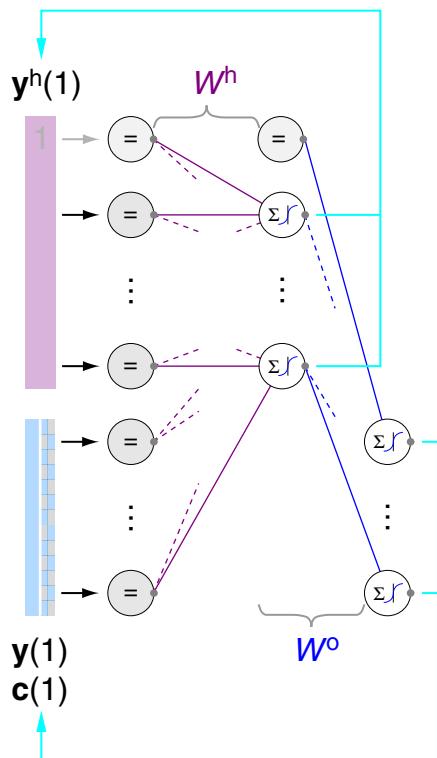
Output decoding over t .

Hidden and output layer at subsequent time steps.

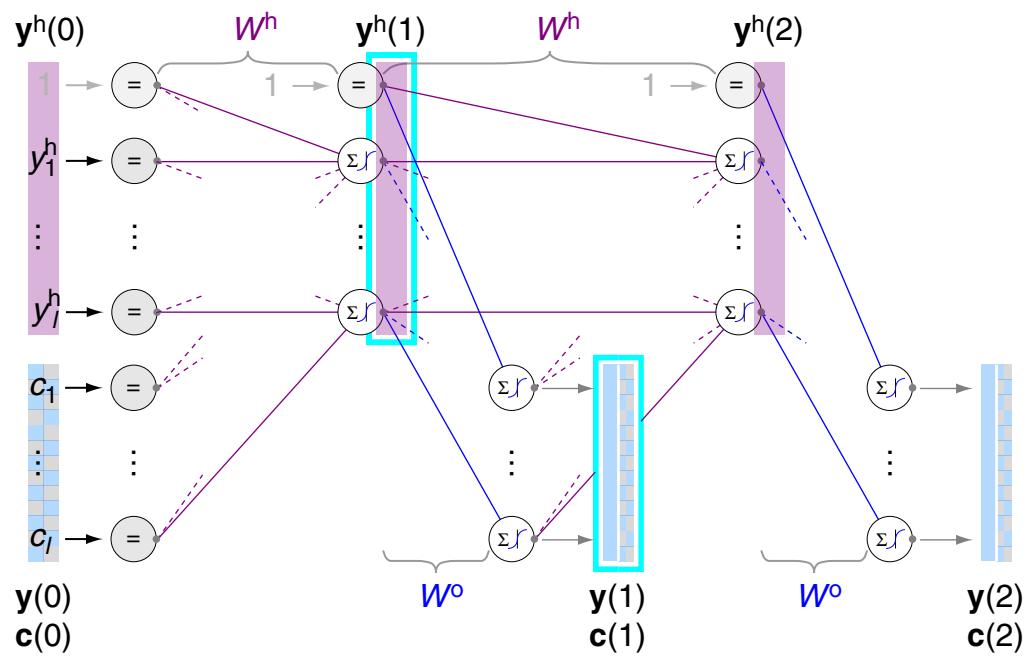
Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1



Output decoding over t .

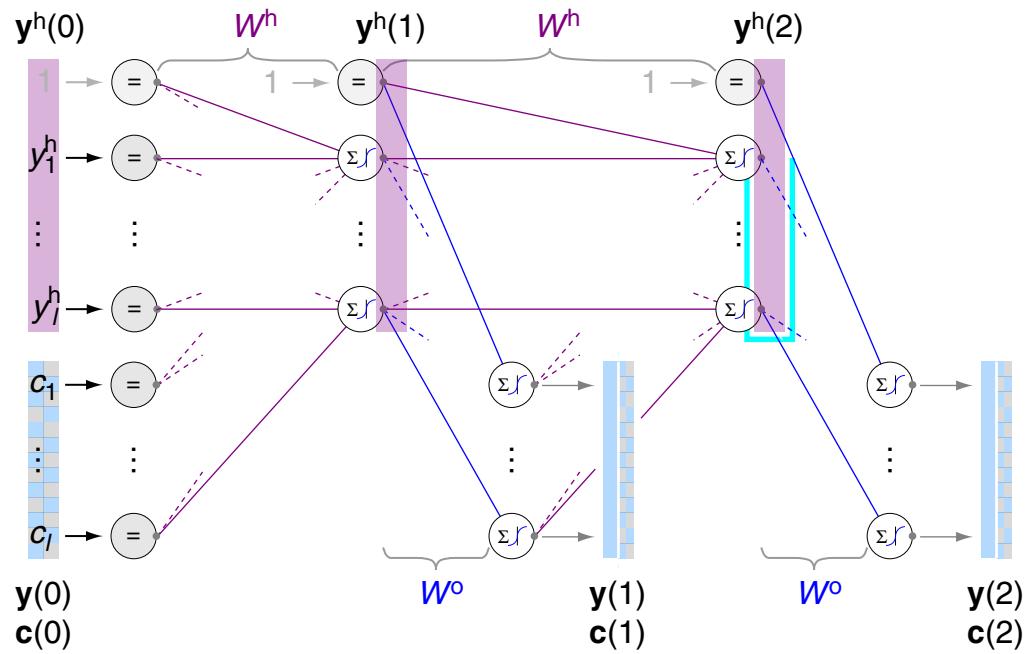
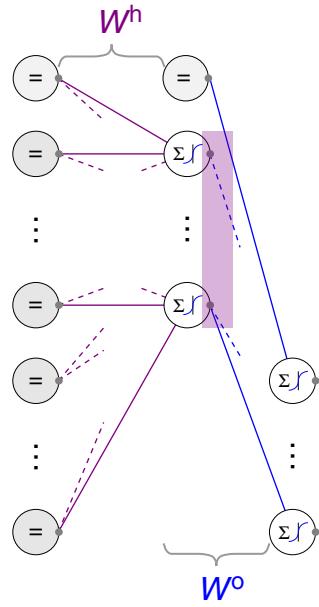


Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $1 \rightarrow 2$



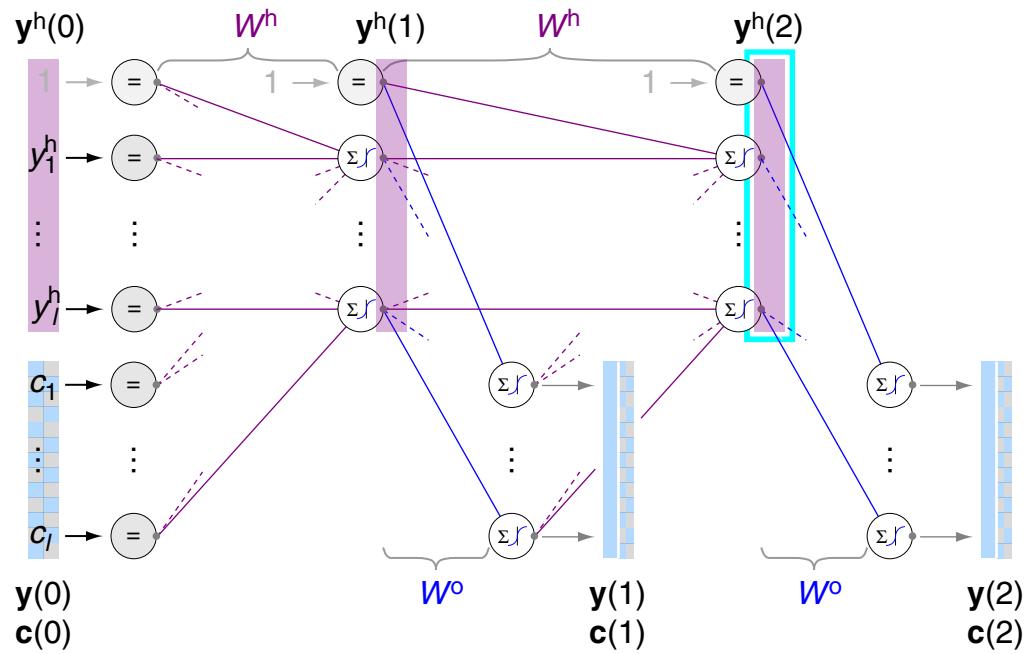
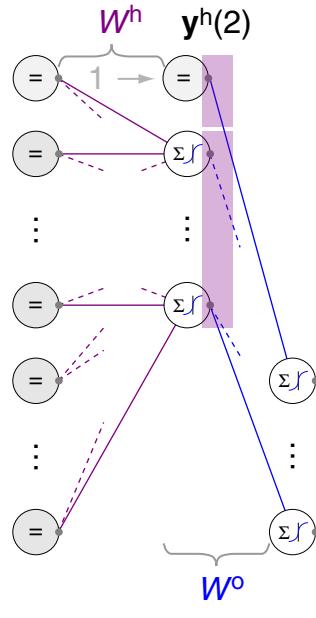
Output decoding over t .

Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $1 \rightarrow 2$



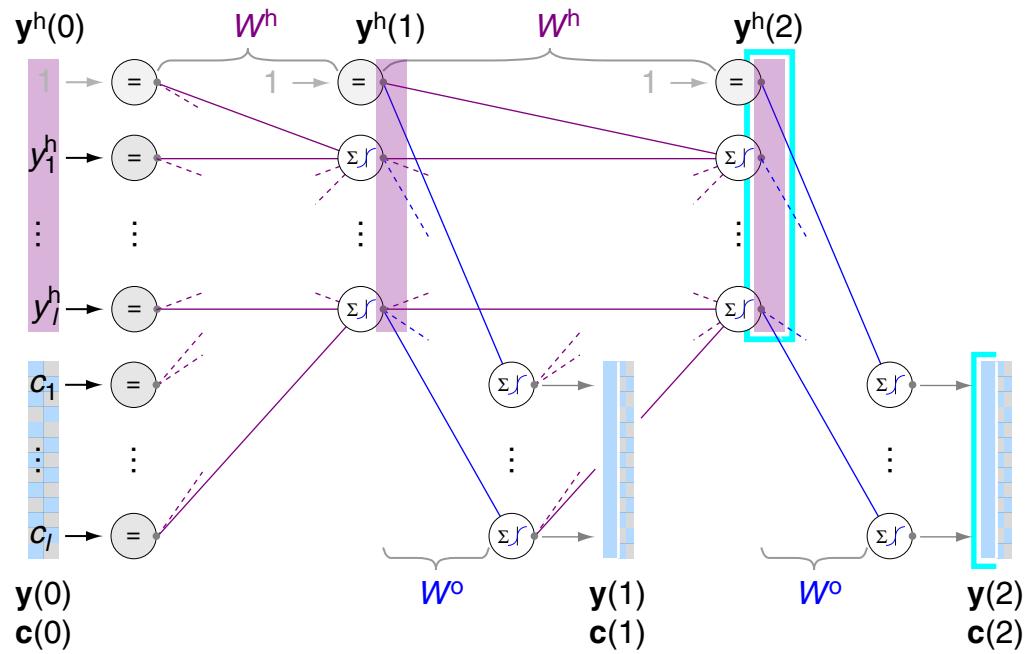
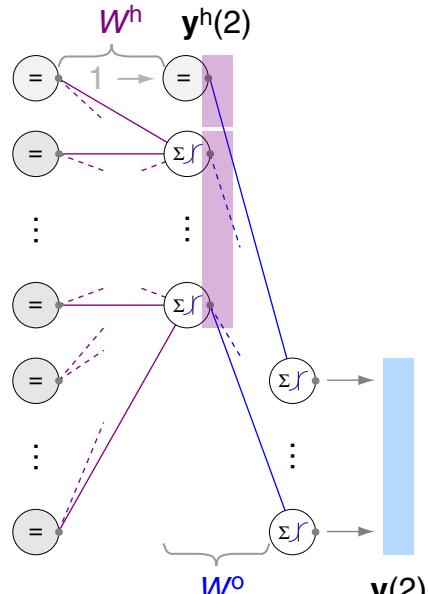
Output decoding over t .

Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $1 \rightarrow 2$



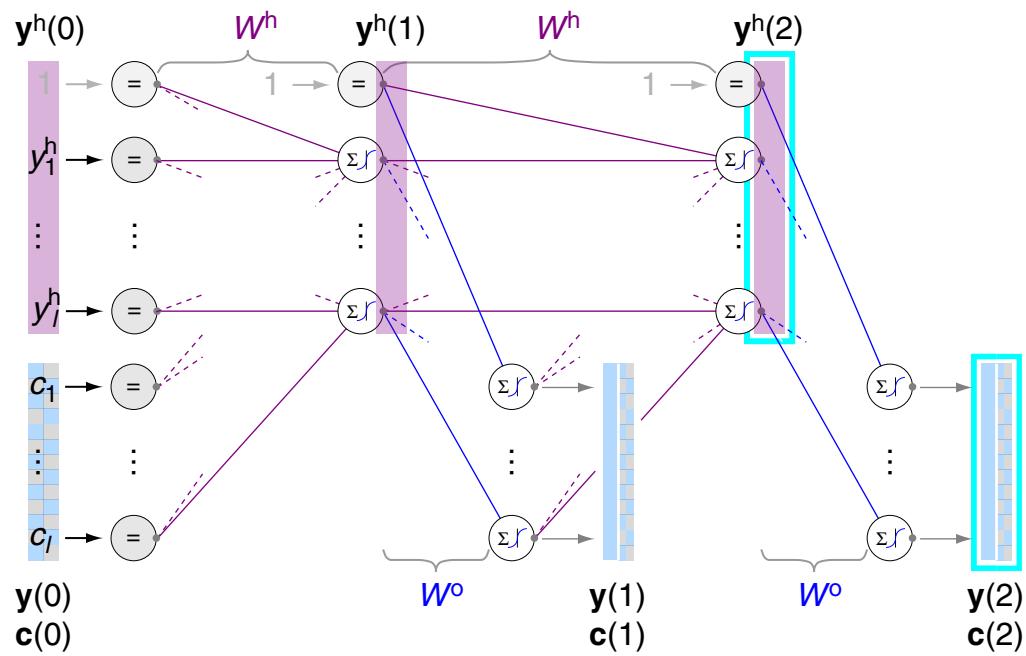
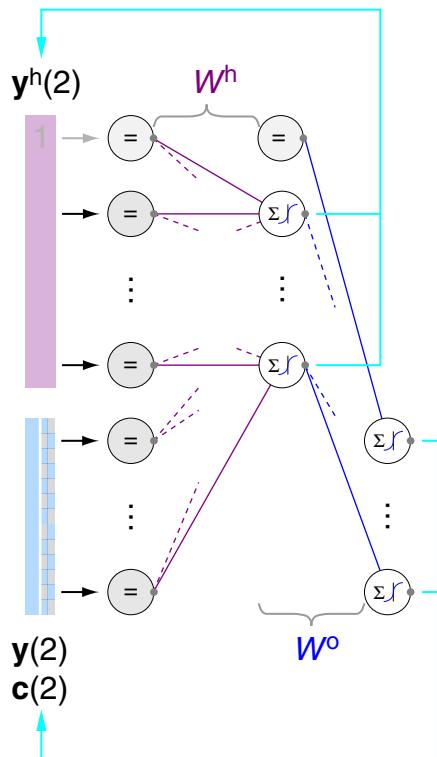
Output decoding over t .

Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2



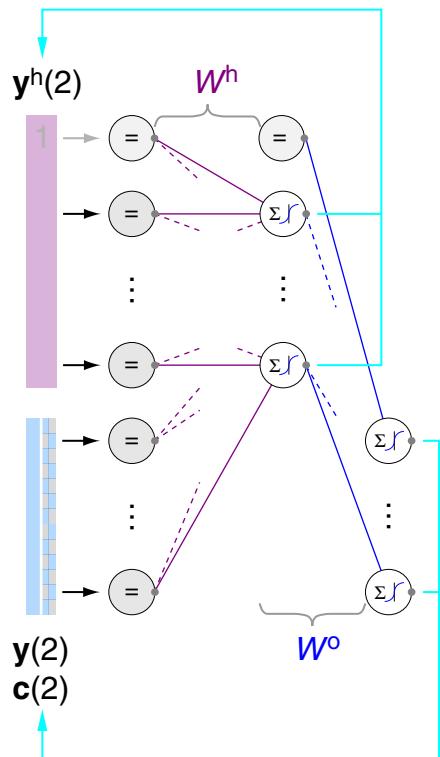
Output decoding over t .

Hidden and output layer at subsequent time steps.

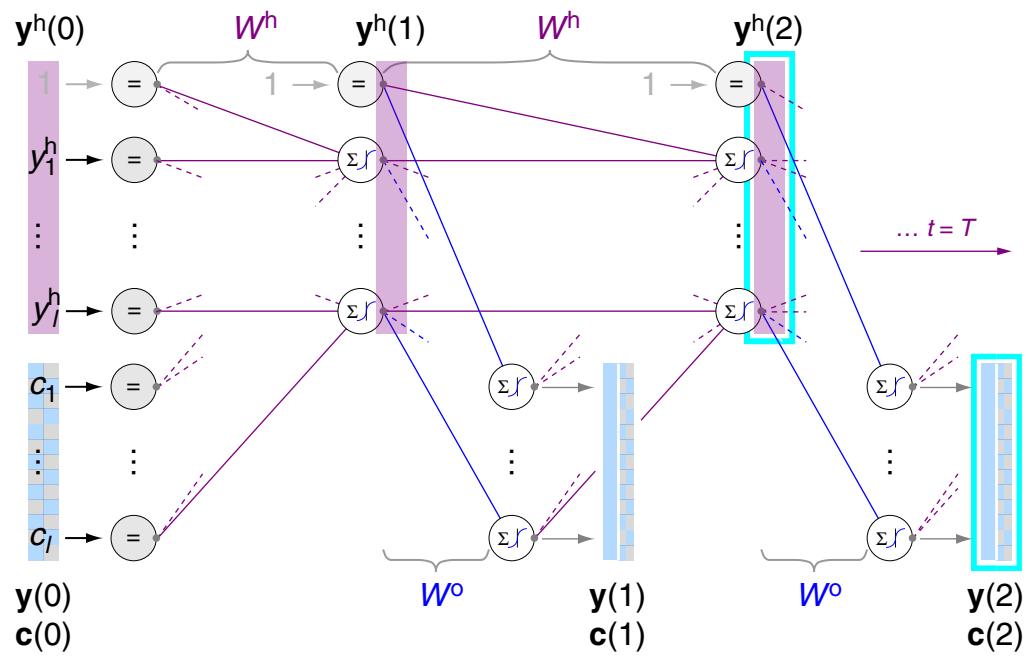
Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2



Output decoding over t .



Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation

- ⊕ → I love my cat.
- ⊕ → Cats and dogs lap water.
- ⊖ → It is raining cats and dogs.
- ⊖ → Cats and dogs are not allowed.
- ⊖ → Cats and dogs have always been natural enemies.

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water <start> <end>)

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation (continued)

- ⊕ → I love my cat.
- ⊕ → Cats and dogs lap water.
- ⊖ → It is raining cats and dogs.
- ⊖ → Cats and dogs are not allowed.
- ⊖ → Cats and dogs have always been natural enemies.

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water <start> <end>)

Input: $[[[[\mathbf{x}, \mathbf{y}(0)], \mathbf{y}(1)], \mathbf{y}(2)], \dots], \mathbf{y}(\tau-1)], \quad \mathbf{x} = \begin{pmatrix} \oplus \\ \cdot \end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \dots, \mathbf{y}(\tau)], \quad \mathbf{y}(0) \equiv \mathbf{c}(0) \hat{=} <\text{start}>, \quad \mathbf{y}(\tau) \hat{=} \mathbf{c}(5) \hat{=} <\text{end}>$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation (continued)

- ⊕ → I love my cat.
- ⊕ → Cats and dogs lap water.
- ⊖ → It is raining cats and dogs.
- ⊖ → Cats and dogs are not allowed.
- ⊖ → Cats and dogs have always been natural enemies.

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water <start> <end>)

Input: $[[[[\mathbf{x}, \mathbf{y}(0)], \mathbf{y}(1)], \mathbf{y}(2)], \dots], \mathbf{y}(\tau-1)], \mathbf{x} = \begin{pmatrix} \oplus \\ \cdot \end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \dots, \mathbf{y}(\tau)], \mathbf{y}(0) \equiv \mathbf{c}(0) \hat{=} \text{<start>}, \mathbf{y}(\tau) \hat{=} \mathbf{c}(5) \hat{=} \text{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation (continued)

- ⊕ → I love my cat.
- ⊕ → Cats and dogs lap water.
- ⊖ → It is raining cats and dogs.
- ⊖ → Cats and dogs are not allowed.
- ⊖ → Cats and dogs have always been natural enemies.

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water <start> <end>)

Input: $[[[[\mathbf{x}, \mathbf{y}(0)], \mathbf{y}(1)], \mathbf{y}(2)], \dots], \mathbf{y}(\tau-1)], \quad \mathbf{x} = \begin{pmatrix} \oplus \\ \cdot \end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \dots, \mathbf{y}(\tau)], \quad \mathbf{y}(0) \equiv \mathbf{c}(0) \hat{=} \text{<start>}, \quad \mathbf{y}(\tau) \hat{=} \mathbf{c}(5) \hat{=} \text{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation (continued)

- ⊕ → I love my cat.
- ⊕ → Cats and dogs lap water.
- ⊖ → It is raining cats and dogs.
- ⊖ → Cats and dogs are not allowed.
- ⊖ → Cats and dogs have always been natural enemies.

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water <start> <end>)

Input: $[[[[\mathbf{x}, \mathbf{y}(0)] , \mathbf{y}(1)] , \mathbf{y}(2)] , \dots] , \mathbf{y}(\tau-1)] , \quad \mathbf{x} = \begin{pmatrix} \oplus \\ \cdot \end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \dots, \mathbf{y}(\tau)], \quad \mathbf{y}(0) \equiv \mathbf{c}(0) \hat{=} \text{<start>}, \quad \mathbf{y}(\tau) \hat{=} \mathbf{c}(5) \hat{=} \text{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation (continued)

- ⊕ → I love my cat.
- ⊕ → Cats and dogs lap water.
- ⊖ → It is raining cats and dogs.
- ⊖ → Cats and dogs are not allowed.
- ⊖ → Cats and dogs have always been natural enemies.

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water <start> <end>)

Input: $[[[[\mathbf{x}, \mathbf{y}(0)] , \mathbf{y}(1)] , \mathbf{y}(2)] , \dots] , \mathbf{y}(\tau-1)] , \quad \mathbf{x} = \begin{pmatrix} \oplus \\ \cdot \end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \dots, \mathbf{y}(\tau)], \quad \mathbf{y}(0) \equiv \mathbf{c}(0) \hat{=} \text{<start>}, \quad \mathbf{y}(\tau) \hat{=} \mathbf{c}(5) \hat{=} \text{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation (continued)

- ⊕ → I love my cat.
- ⊕ → Cats and dogs lap water.
- ⊖ → It is raining cats and dogs.
- ⊖ → Cats and dogs are not allowed.
- ⊖ → Cats and dogs have always been natural enemies.

Vocabulary: (allowed always and are been cat cats dogs enemies have i is it lap love my natural not raining water <start> <end>)

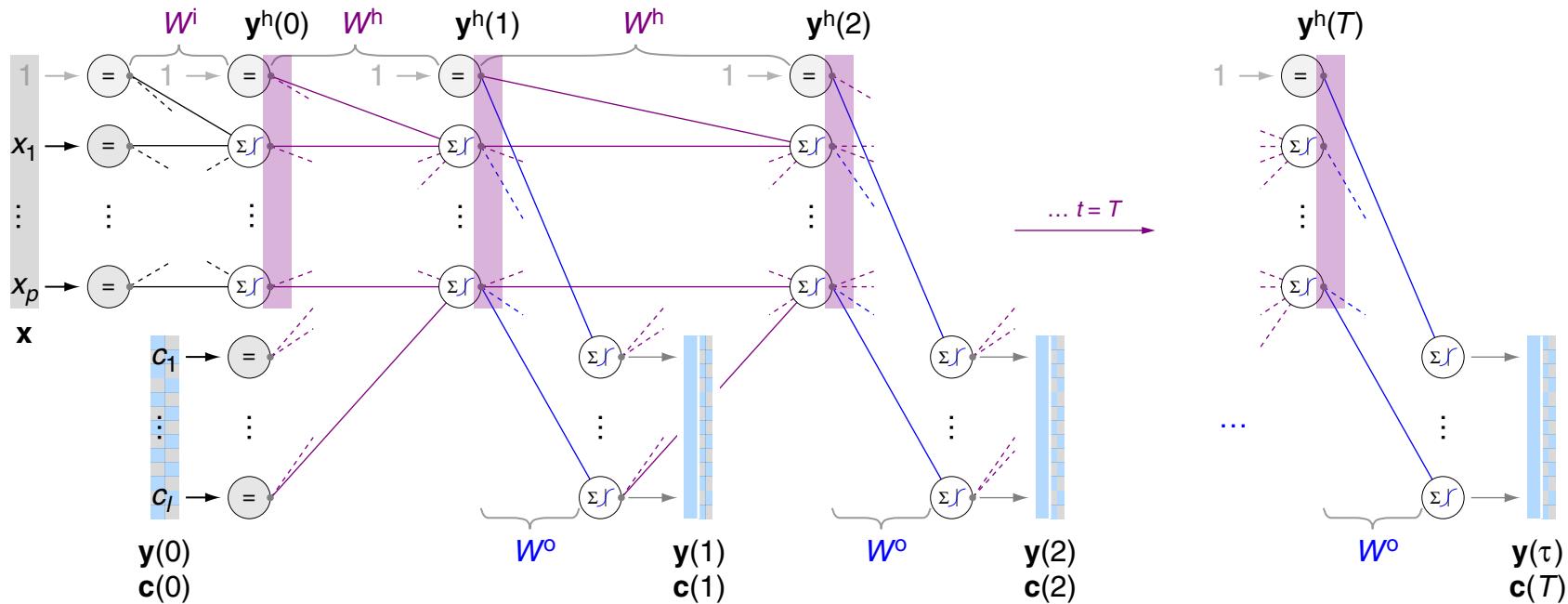
Input: $[[[[\mathbf{x}, \mathbf{y}(0)], \mathbf{y}(1)], \mathbf{y}(2)], \dots, \mathbf{y}(\tau-1)], \quad \mathbf{x} = \begin{pmatrix} \oplus \\ \cdot \end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \dots, \mathbf{y}(\tau)], \quad \mathbf{y}(0) \equiv \mathbf{c}(0) \hat{=} \text{<start>}, \quad \mathbf{y}(\tau) \equiv \mathbf{c}(5) \hat{=} \text{<end>}$

Target: $[\mathbf{c}(1), \dots, \mathbf{c}(5)] = \left[\begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 1 \end{pmatrix} \right]$
 $\hat{=} [\text{word_11}, \text{word_15}, \text{word_16}, \text{word_6}, \text{word_22}]$
 $\hat{=} \text{I love my cat}$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

$$x, [y(1), \dots, y(\tau-1)]$$

Output:

$$y(t) = \sigma_1(W^o y^h(t)), t = 1, \dots, \tau$$

Hidden:

$$y^h(0) = \sigma(W^i x)$$

$$y^h(t) = \sigma\left(W^h\left(y^h(t-1)\right)\right), t = 1, \dots, \tau$$

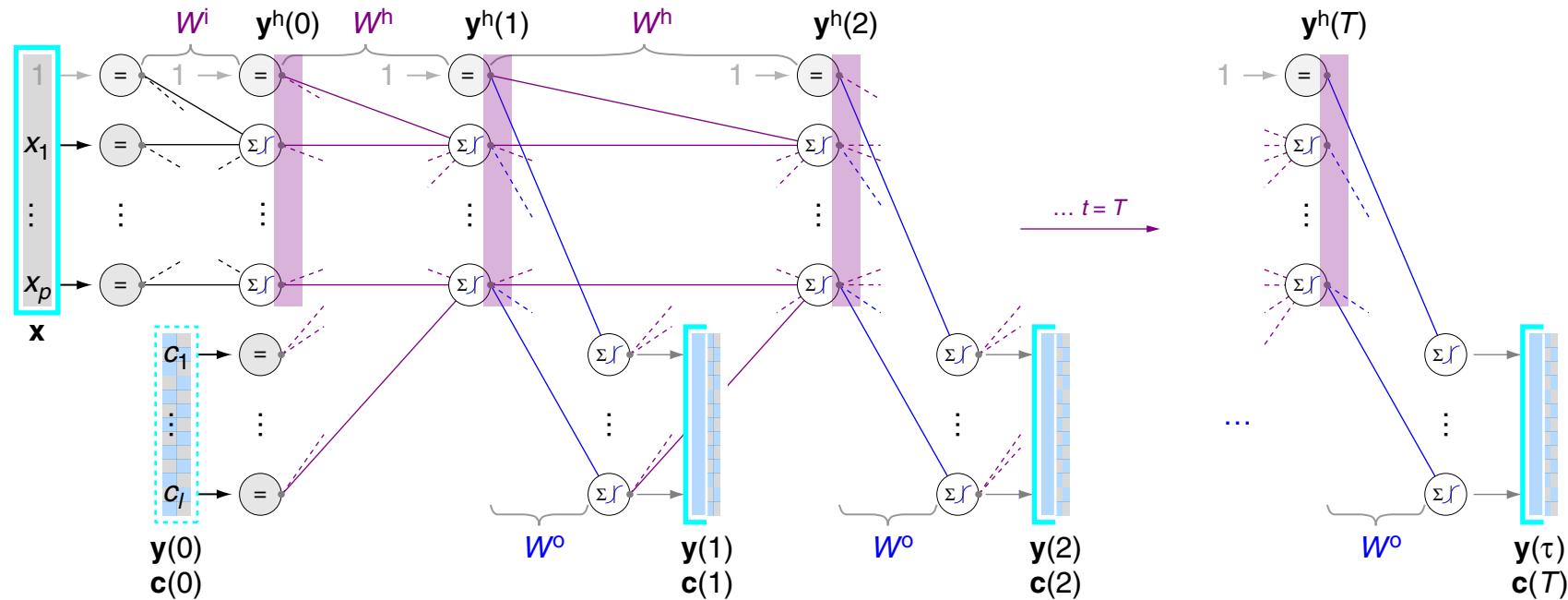
Target:

$$[c(1), \dots, c(T)]$$

$$c(T) \hat{=} \langle \text{end} \rangle$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs (continued)



Input:

$$x, [y(1), \dots, y(\tau-1)]$$

Output:

$$y(t) = \sigma_1(W^o y^h(t)), t = 1, \dots, \tau$$

Hidden:

$$y^h(0) = \sigma(W^i x)$$

$$y^h(t) = \sigma\left(W^h \begin{pmatrix} y^h(t-1) \\ y(t-1) \end{pmatrix}\right), t = 1, \dots, \tau$$

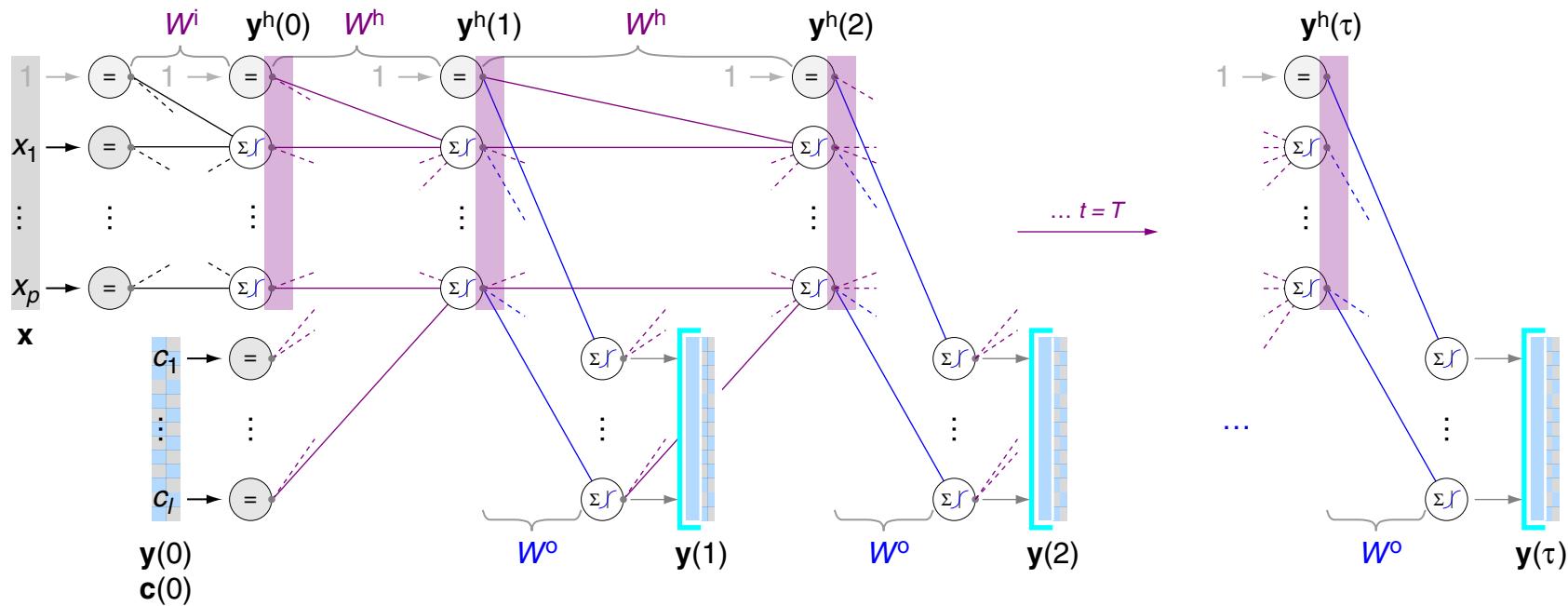
Target:

$$[c(1), \dots, c(T)]$$

$$c(T) \hat{=} \langle \text{end} \rangle$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs (continued)



Input:

$$x, [y(1), \dots, y(\tau-1)]$$

Output:

$$y(t) = \sigma_1 (W^o y^h(t)), t = 1, \dots, \tau$$

Hidden:

$$y^h(0) = \sigma (W^i x)$$

$$y^h(t) = \sigma \left(W^h \begin{pmatrix} y^h(t-1) \\ y(t-1) \end{pmatrix} \right), t = 1, \dots, \tau$$

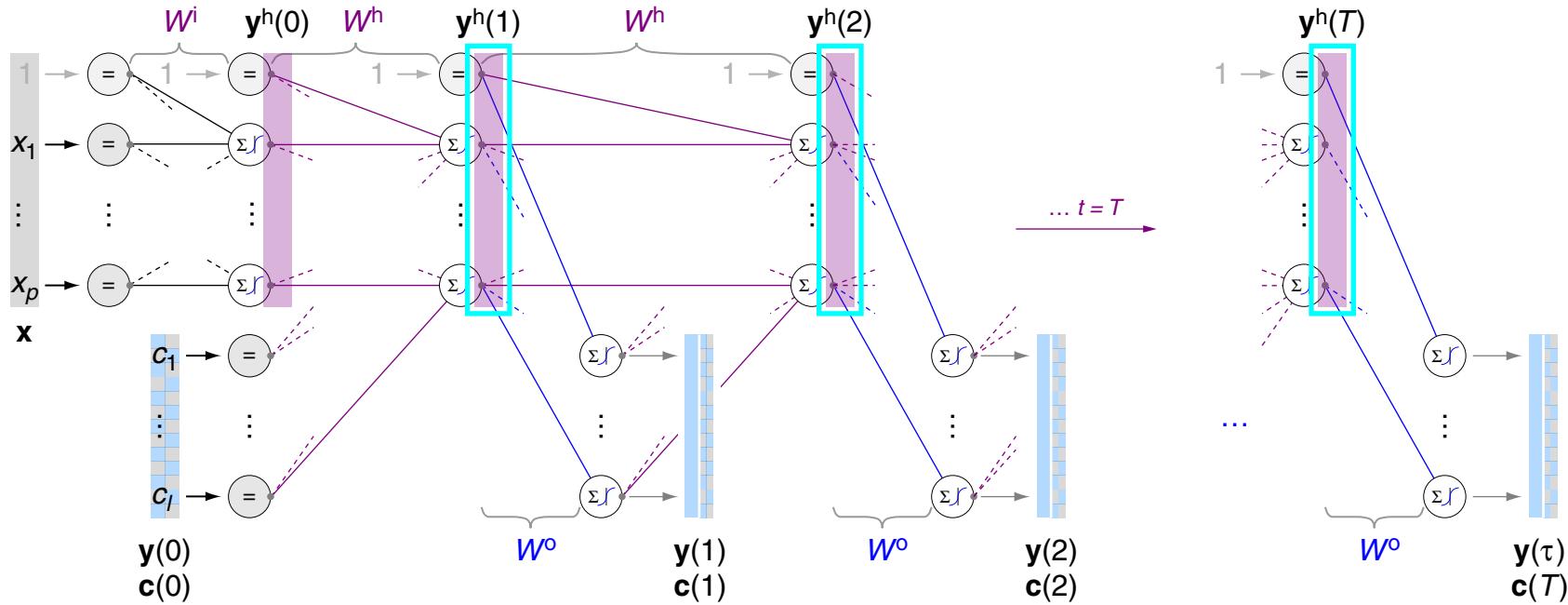
Target:

$$[c(1), \dots, c(\tau)]$$

$$c(\tau) \hat{=} \langle \text{end} \rangle$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs (continued)



Input:

$$x, [y(1), \dots, y(\tau-1)]$$

Output:

$$y(t) = \sigma_1 (W^o y^h(t)), t = 1, \dots, \tau$$

Hidden:

$$y^h(0) = \sigma \left(W^i x \right)$$

$$y^h(t) = \sigma \left(W^h \begin{pmatrix} y^h(t-1) \\ c(t-1) \end{pmatrix} \right), t = 1, \dots, T$$

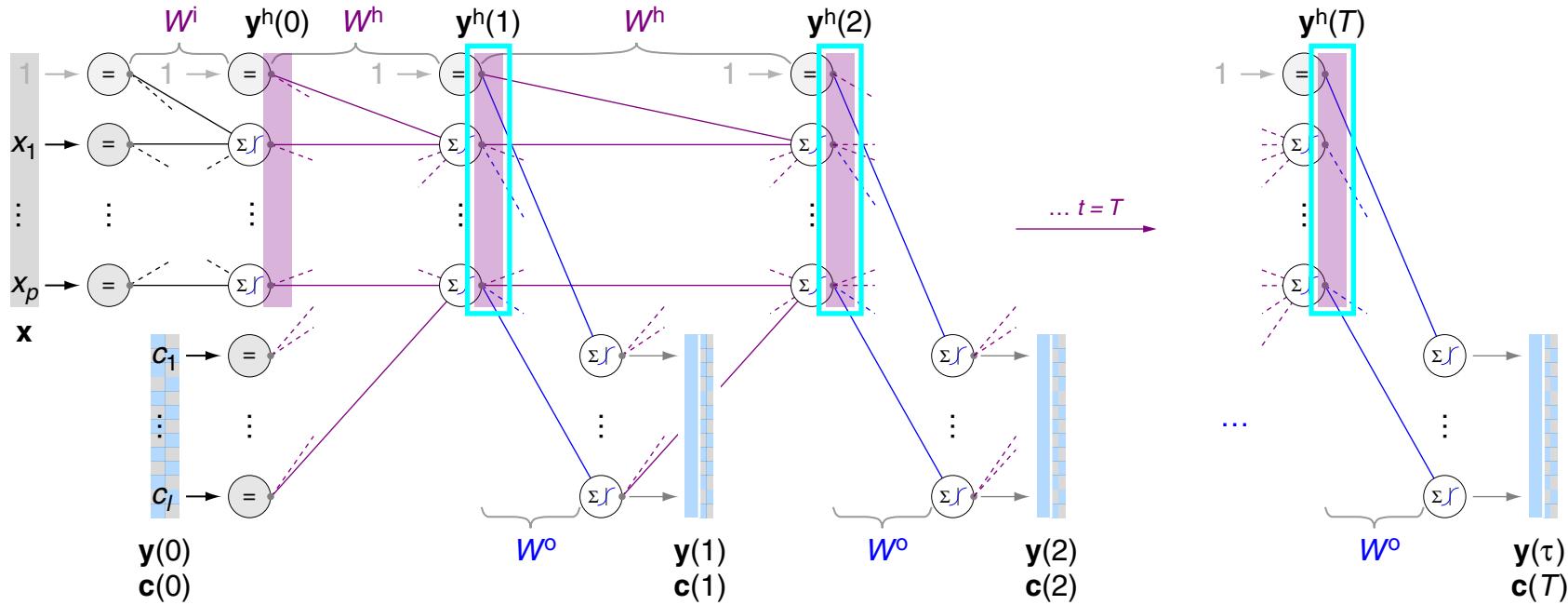
Target:

$$[c(1), \dots, c(T)]$$

$$c(T) \hat{=} \langle \text{end} \rangle$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs (continued)



Input:

$$x, [y(1), \dots, y(\tau-1)]$$

Output:

$$y(t) = \sigma_1 (W^o y^h(t)), t = 1, \dots, \tau$$

Hidden:

$$y^h(0) = \sigma (W^i x)$$

$$y^h(t) = \sigma \left(W^h \begin{pmatrix} y^h(t-1) \\ y(t-1) \end{pmatrix} \right), t = 1, \dots, \tau$$

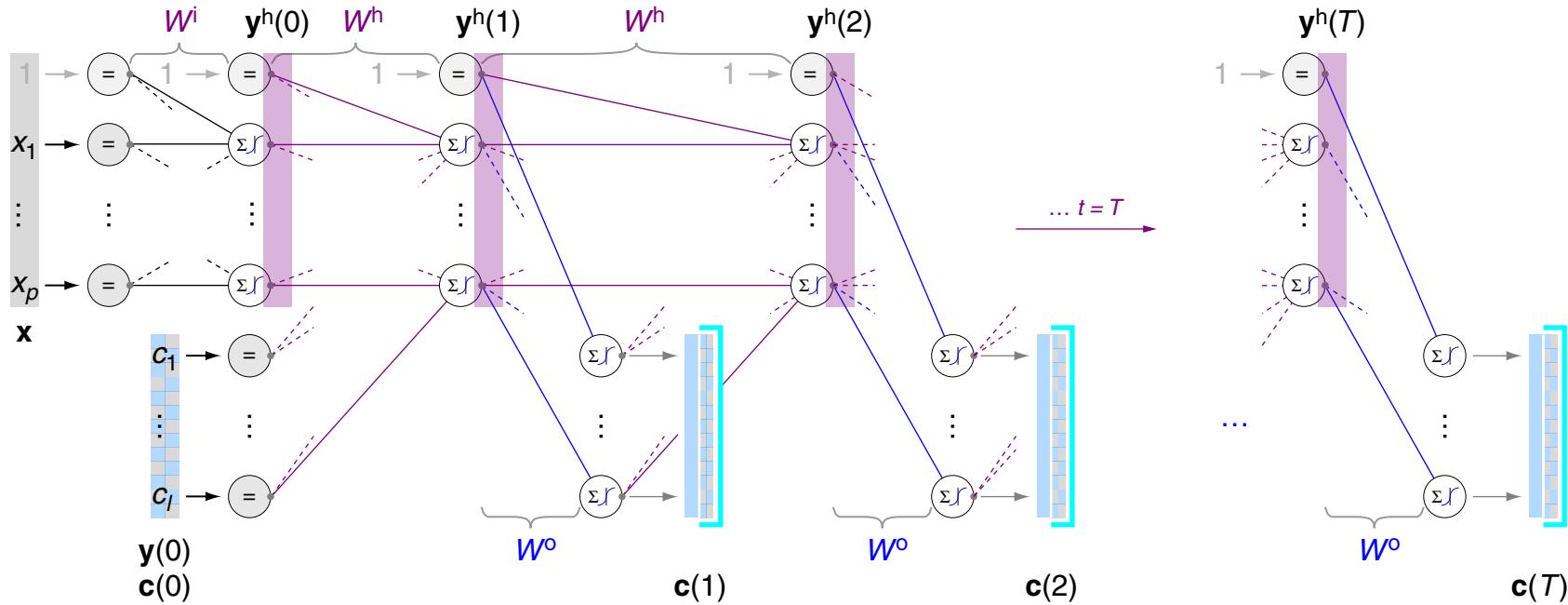
Target:

$$[c(1), \dots, c(T)]$$

$$c(T) \hat{\equiv} \langle \text{end} \rangle$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs (continued)



Input:

$$x, [y(1), \dots, y(\tau-1)]$$

Output:

$$y(t) = \sigma_1 (W^o y^h(t)), t = 1, \dots, \tau$$

Hidden:

$$y^h(0) = \sigma (W^i x)$$

$$y^h(t) = \sigma \left(W^h \begin{pmatrix} y^h(t-1) \\ y(t-1) \end{pmatrix} \right), t = 1, \dots, \tau$$

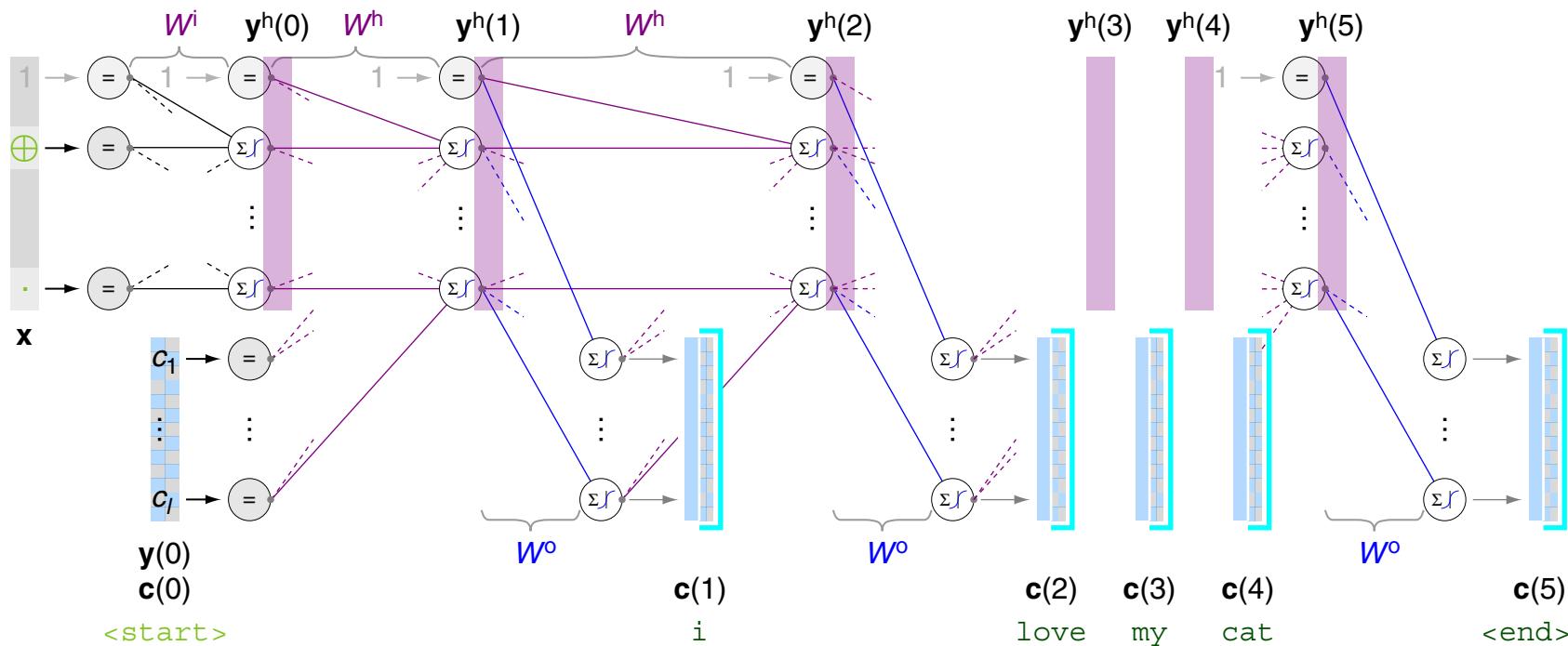
Target:

$$[c(1), \dots, c(T)]$$

$$c(T) \hat{=} \langle \text{end} \rangle$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs (continued)



Input:

$$x, [y(1), \dots, y(4)]$$

Output:

$$y(t) = \sigma_1(W^o y^h(t)), t = 1, \dots, 5$$

Hidden:

$$y^h(0) = \sigma(W^i x)$$

$$y^h(t) = \sigma\left(W^h \begin{pmatrix} y^h(t-1) \\ c(t-1) \end{pmatrix}\right), t = 1, \dots, 5$$

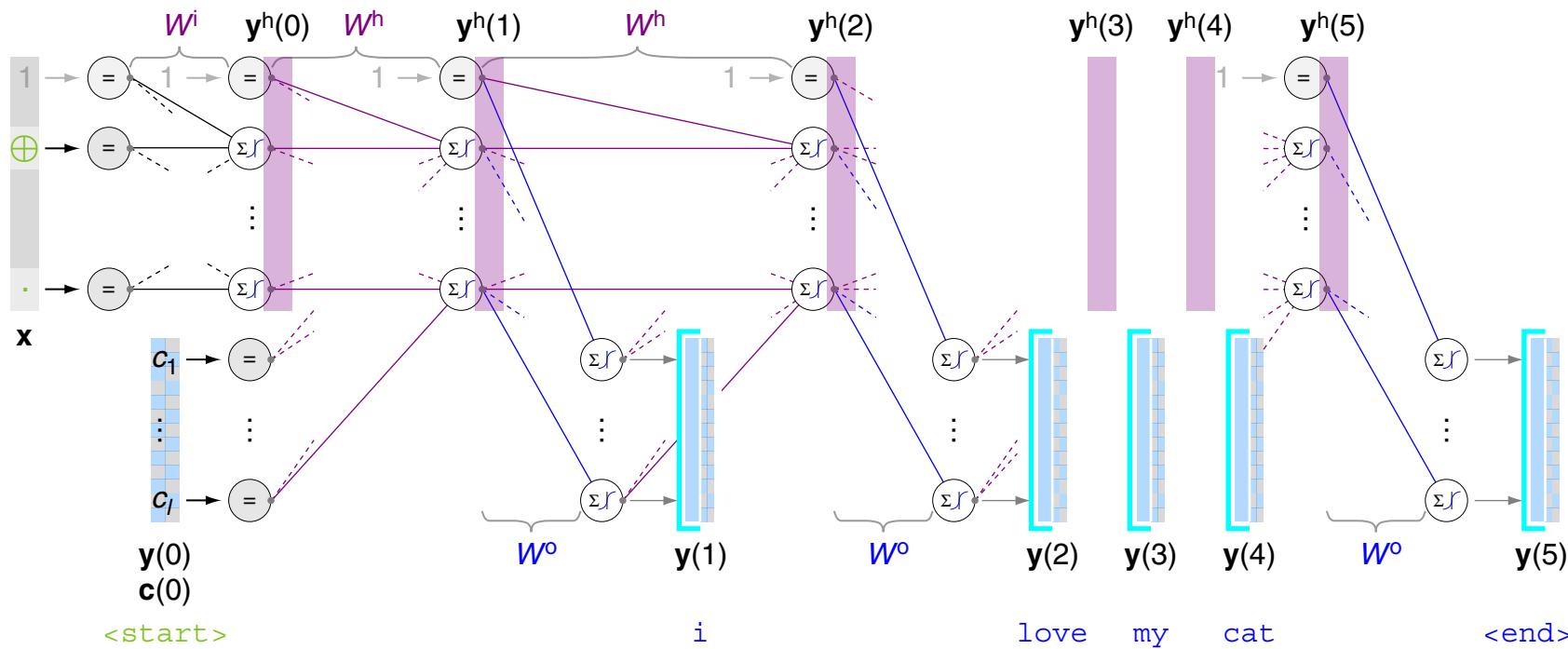
Target:

$$[c(1), \dots, c(5)]$$

$$c(5) \hat{=} <\text{end}>$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs (continued)



Input:

$$\mathbf{x}, [\mathbf{y}(1), \dots, \mathbf{y}(4)]$$

Output:

$$\mathbf{y}(t) = \sigma_1 (W^o \mathbf{y}^h(t)), t = 1, \dots, 5$$

Hidden:

$$\mathbf{y}^h(0) = \sigma (W^i \mathbf{x})$$

$$\mathbf{y}^h(t) = \sigma \left(W^h \begin{pmatrix} \mathbf{y}^h(t-1) \\ \mathbf{y}(t-1) \end{pmatrix} \right), t = 1, \dots, 5$$

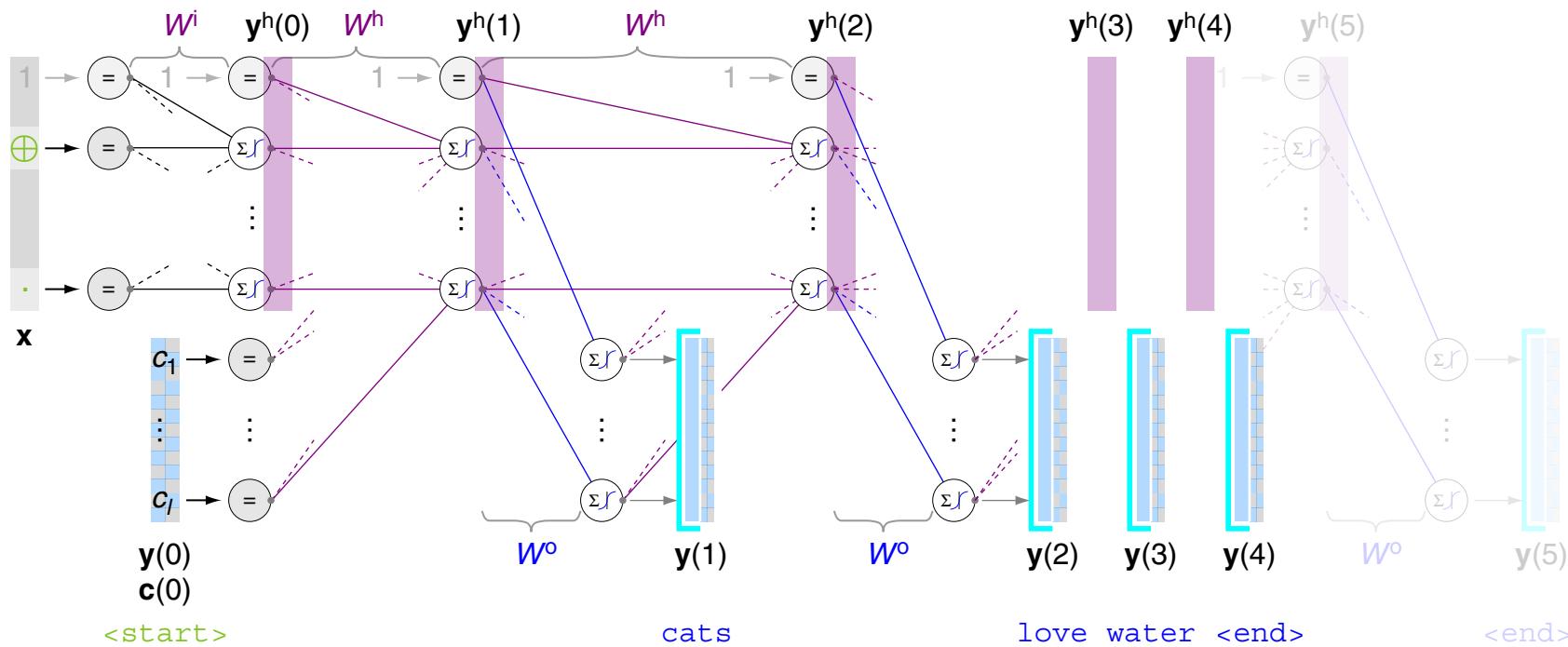
Target:

$$[\mathbf{c}(1), \dots, \mathbf{c}(5)]$$

$$\mathbf{c}(5) \hat{=} \text{}$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs (continued)



Input:

$$x, [y(1), \dots, y(3)]$$

Output:

$$y(t) = \sigma_1(W^o y^h(t)), t = 1, \dots, 4$$

Hidden:

$$y^h(0) = \sigma(W^i x)$$

$$y^h(t) = \sigma\left(W^h \begin{pmatrix} y^h(t-1) \\ y(t-1) \end{pmatrix}\right), t = 1, \dots, 4$$

Target:

$$[c(1), \dots, c(5)]$$

$$c(5) \hat{=} \langle \text{end} \rangle$$

Remarks:

- We denote $y(0)$ not as input since it is predefined and does not contain any “actual knowledge”. In particular, $y(0) \equiv c(0) \hat{=} \langle \text{start} \rangle$.

- At training time the calculation of $y^h(t)$ usually considers the ground truth $c(t-1)$:

$$y^h(t) = \sigma \left(\textcolor{violet}{W}^h \begin{pmatrix} y^h(t-1) \\ c(t-1) \end{pmatrix} \right)$$

- At test time (“production mode”) the calculation of $y^h(t)$ has to consider the output $y(t-1)$:

$$y^h(t) = \sigma \left(\textcolor{violet}{W}^h \begin{pmatrix} y^h(t-1) \\ y(t-1) \end{pmatrix} \right)$$

Recurrent Neural Networks

The IGD Algorithm for Class-to-Sequence Tasks [IGD_{seq2c}]

Algorithm: IGD_{c2seq} Incremental Gradient Descent for RNNs at class2seq tasks.

Input: D Multiset of examples ($\mathbf{x}, [\mathbf{c}(1), \dots, \mathbf{c}(T)]$) with $\mathbf{x} \in \{0, 1\}^p$, $\mathbf{c}(t) \in \mathbf{R}^k$.
 η Learning rate, a small positive constant.

Output: W^i, W^h, W^o Weights matrices. (= hypothesis)

1. *initialize_random_weights(W^i, W^h, W^o)*, $t_{epoch} = 0$
2. **REPEAT**
3. $t_{epoch} = t_{epoch} + 1$
4. **FOREACH** ($\mathbf{x}, [\mathbf{c}(1), \dots, \mathbf{c}(T)] \in D$) **DO**
5.

Model function evaluation.
6.

Calculation of residual vectors.
- 7a.

Calculation of derivative of the loss.
- 7b.
8.

Parameter update $\hat{=}$ one gradient step down.
9. **ENDDO**
10. **UNTIL**(convergence($D, \mathbf{y}(\cdot), t_{epoch}$))
11. **return**(W^i, W^h, W^o)

Recurrent Neural Networks

The IGD Algorithm for Class-to-Sequence Tasks (continued) [IGD_{seq2c}]

Algorithm: IGD_{c2seq} Incremental Gradient Descent for RNNs at class2seq tasks.

Input: D Multiset of examples ($\mathbf{x}, [\mathbf{c}(1), \dots, \mathbf{c}(T)]$) with $\mathbf{x} \in \{0, 1\}^p$, $\mathbf{c}(t) \in \mathbf{R}^k$.
 η Learning rate, a small positive constant.

Output: W^i, W^h, W^o Weights matrices. (= hypothesis)

1. *initialize_random_weights(W^i, W^h, W^o)*, $t_{epoch} = 0$
2. **REPEAT**
3. $t_{epoch} = t_{epoch} + 1$
4. **FOREACH** ($\mathbf{x}, [\mathbf{c}(1), \dots, \mathbf{c}(T)] \in D$) **DO**
5. $\mathbf{y}^h(0) = \sigma(W^i \mathbf{x})$
FOR $t = 1$ **TO** T **DO** // forward propagation
$$\mathbf{y}^h(t) = \sigma \left(W^h \begin{pmatrix} \mathbf{y}^h(t-1) \\ \mathbf{c}(t-1) \end{pmatrix} \right), \quad \mathbf{y}(t) = \sigma_1(W^o \mathbf{y}^h(t))$$
ENDDO
6. $[\delta(1), \dots, \delta(T)] = [\mathbf{c}(1), \dots, \mathbf{c}(T)] \ominus [\mathbf{y}(1), \dots, \mathbf{y}(\tau)]$ // consider that T may $\neq \tau$
- 7a. $\ell(\mathbf{w}) = \sum_t l(\delta(t)) + \frac{\lambda}{n} R(\mathbf{w}), \quad \nabla \ell(\mathbf{w}) = \text{autodiff}(\ell(), \mathbf{w})$ // backprop. (7a+7b)
- 7b. $\Delta W^i = \eta \cdot \nabla^i \ell(\mathbf{w}), \quad \Delta W^h = \eta \cdot \nabla^h \ell(\mathbf{w}), \quad \Delta W^o = \eta \cdot \nabla^o \ell(\mathbf{w})$
8. $W^i = W^i + \Delta W^i, \quad W^h = W^h + \Delta W^h, \quad W^o = W^o + \Delta W^o$
9. **ENDDO**
10. **UNTIL**(*convergence*($D, \mathbf{y}(\cdot), t_{epoch}$))
11. **return**(W^i, W^h, W^o)

Remarks:

- We use the operator $\gg\ominus\ll$ to compare the two sequences $[c(t)]$ and $[y(t)]$ of possibly different length. Note that the concrete semantics of $\gg\ominus\ll$ is left open here.