Chapter ML:IX

IX. Deep Learning

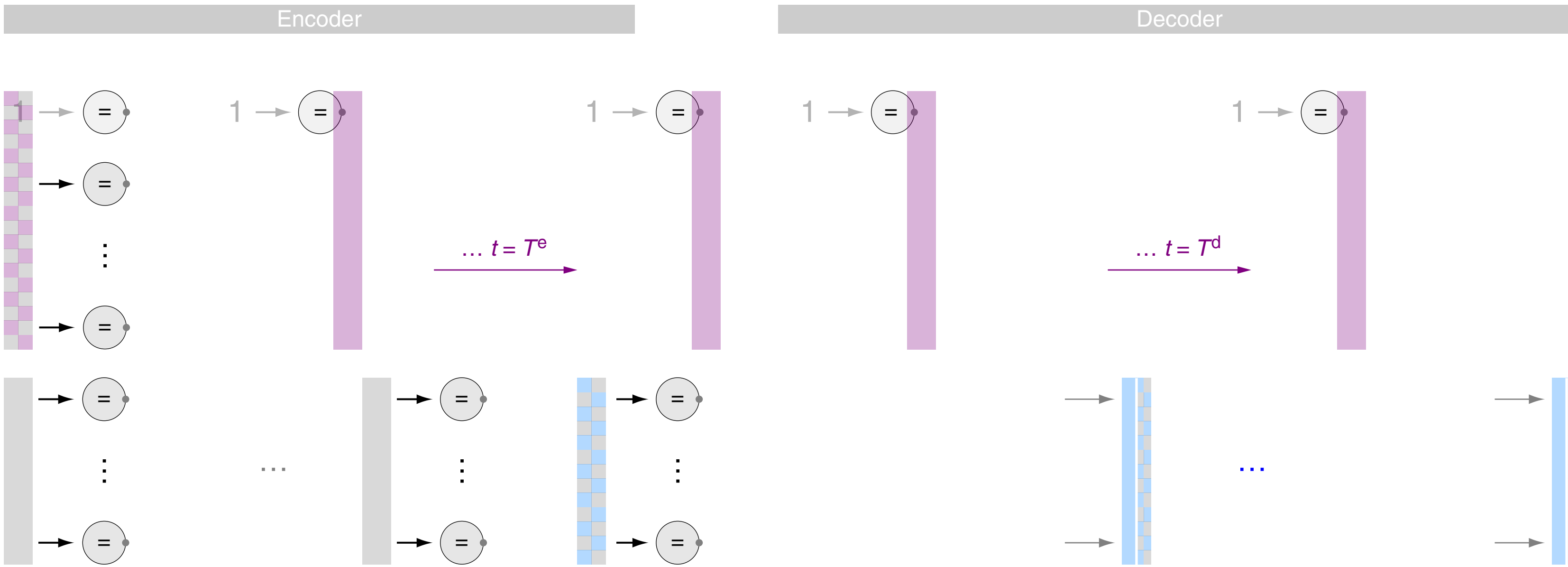
- Elements of Deep Learning
- Convolutional Neural Networks
- Autoencoder Networks

- Recurrent Neural Networks
- Long-Term Dependencies
- RNNs for Machine Translation
- Attention Mechanism
- Transformer

- Transformer Language Models
- Pretraining and Finetuning

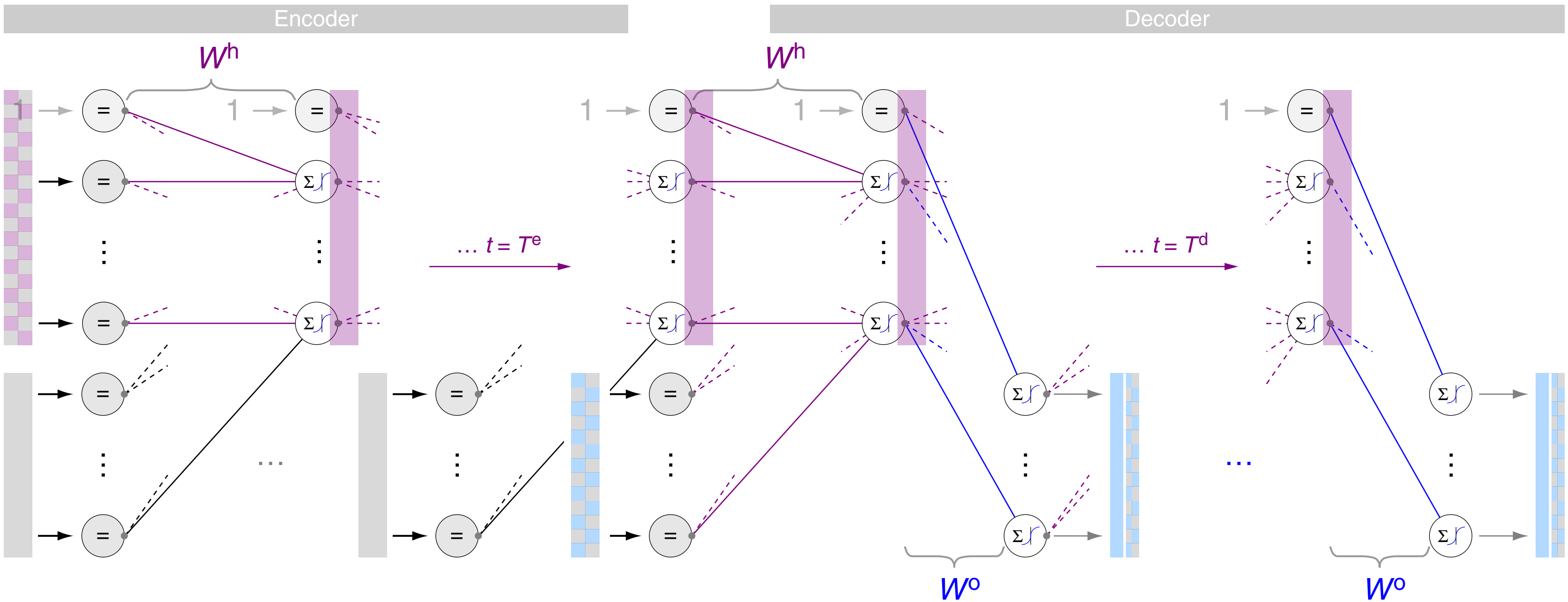
Recurrent Neural Networks

Notation I (MLP matrices)   [notation: MLP matrices, computational graph, language model]



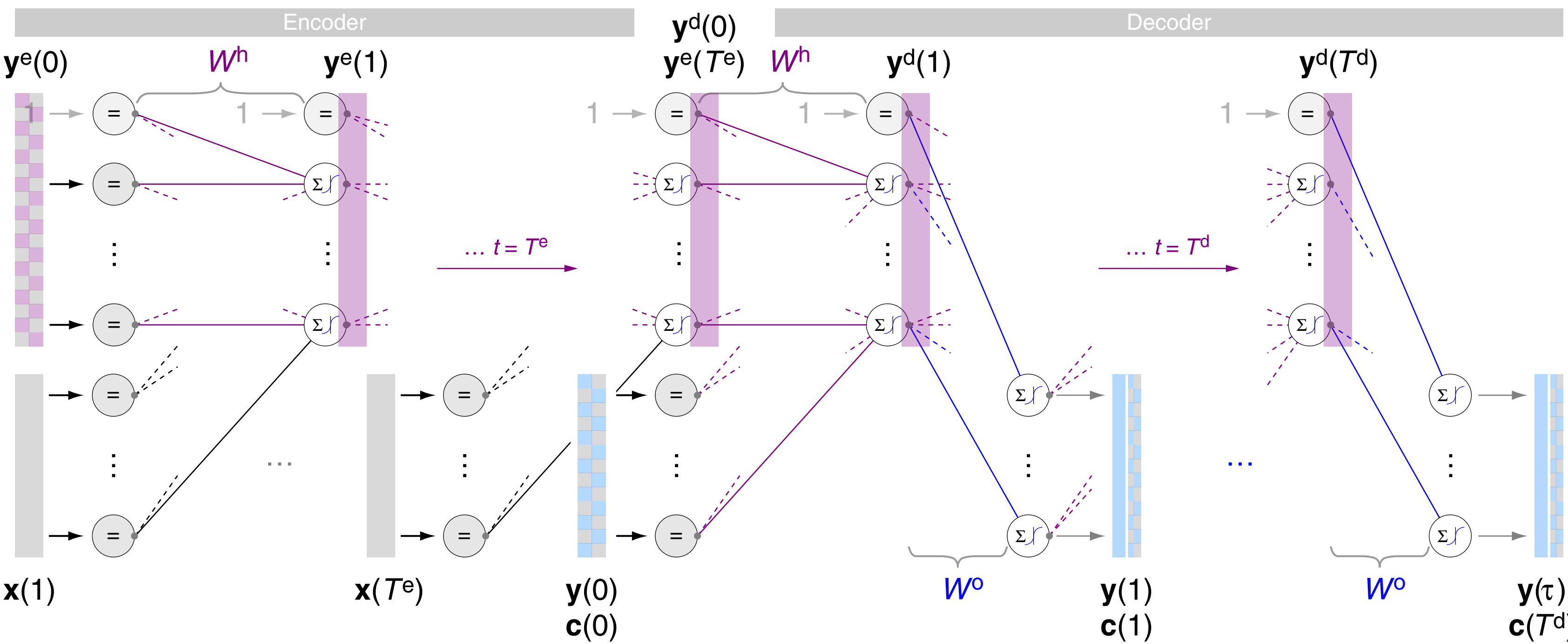
Recurrent Neural Networks

Notation I (MLP matrices) [notation: MLP matrices, computational graph, language model]



Recurrent Neural Networks

Notation I (MLP matrices) [notation: MLP matrices, computational graph, language model]



$\mathbf{x}(t)$ input
in_word

$\mathbf{y}^{\mathrm{h}}(t)$ hidden
$\mathbf{y}^{\mathrm{e}}(t)$ hidden encoder
$\mathbf{y}^{\mathrm{d}}(t)$ hidden decoder

predefined hidden

$\mathbf{y}^{\mathrm{a}}(t)$ attention

$\mathbf{y}(t)$ output
out_word (test)

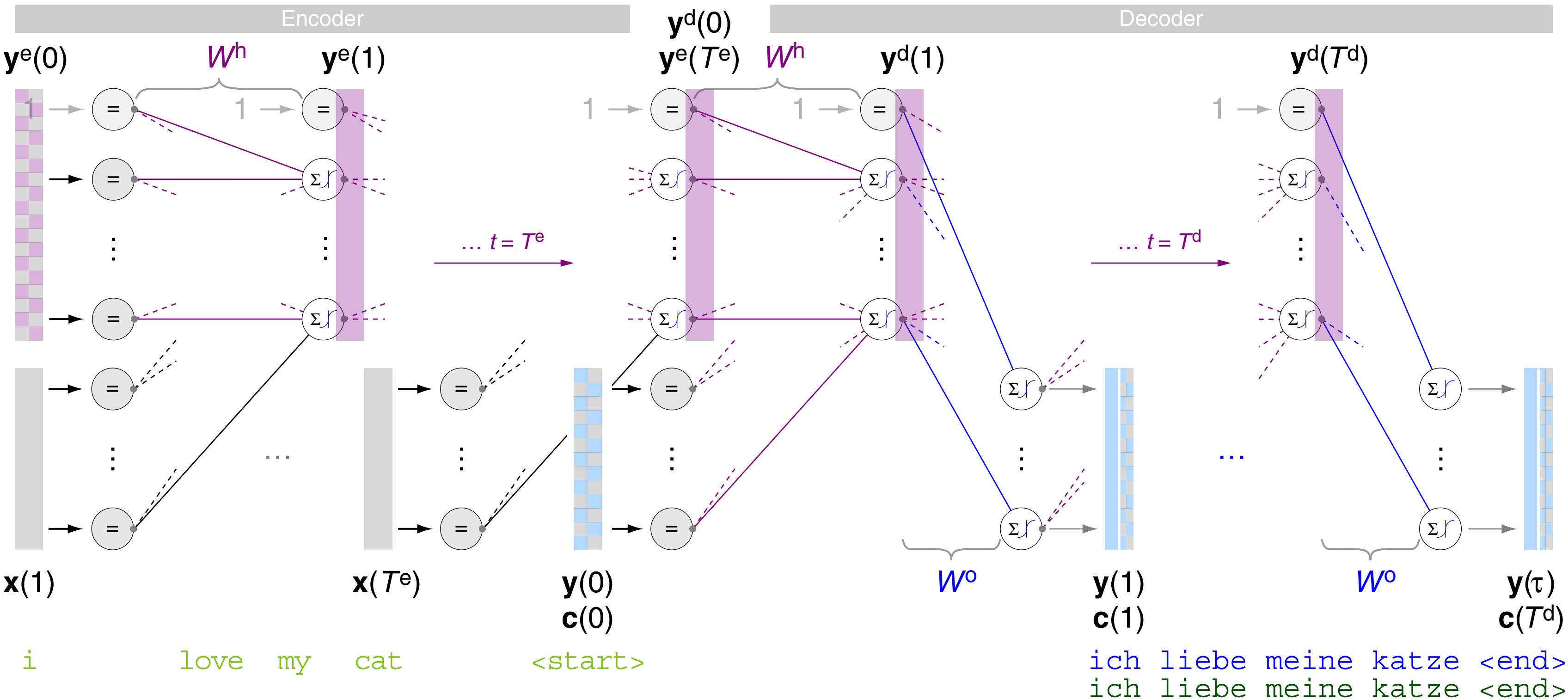
$\mathbf{c}(t)$ target
out_word (training)

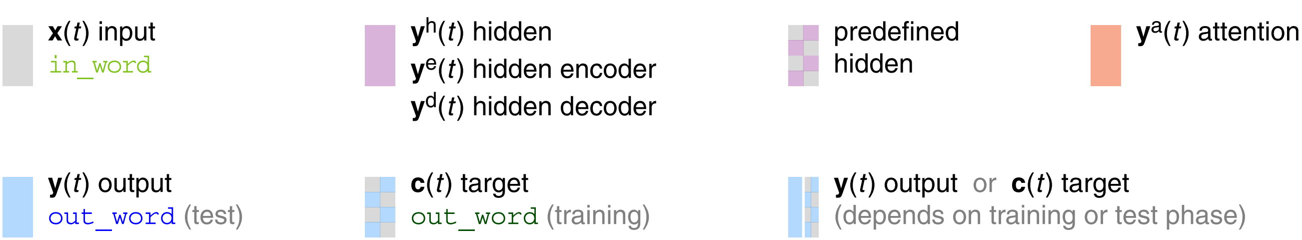
$\mathbf{y}(t)$ output or $\mathbf{c}(t)$ target
(depends on training or test phase)

Recurrent Neural Networks

Notation I (MLP matrices)

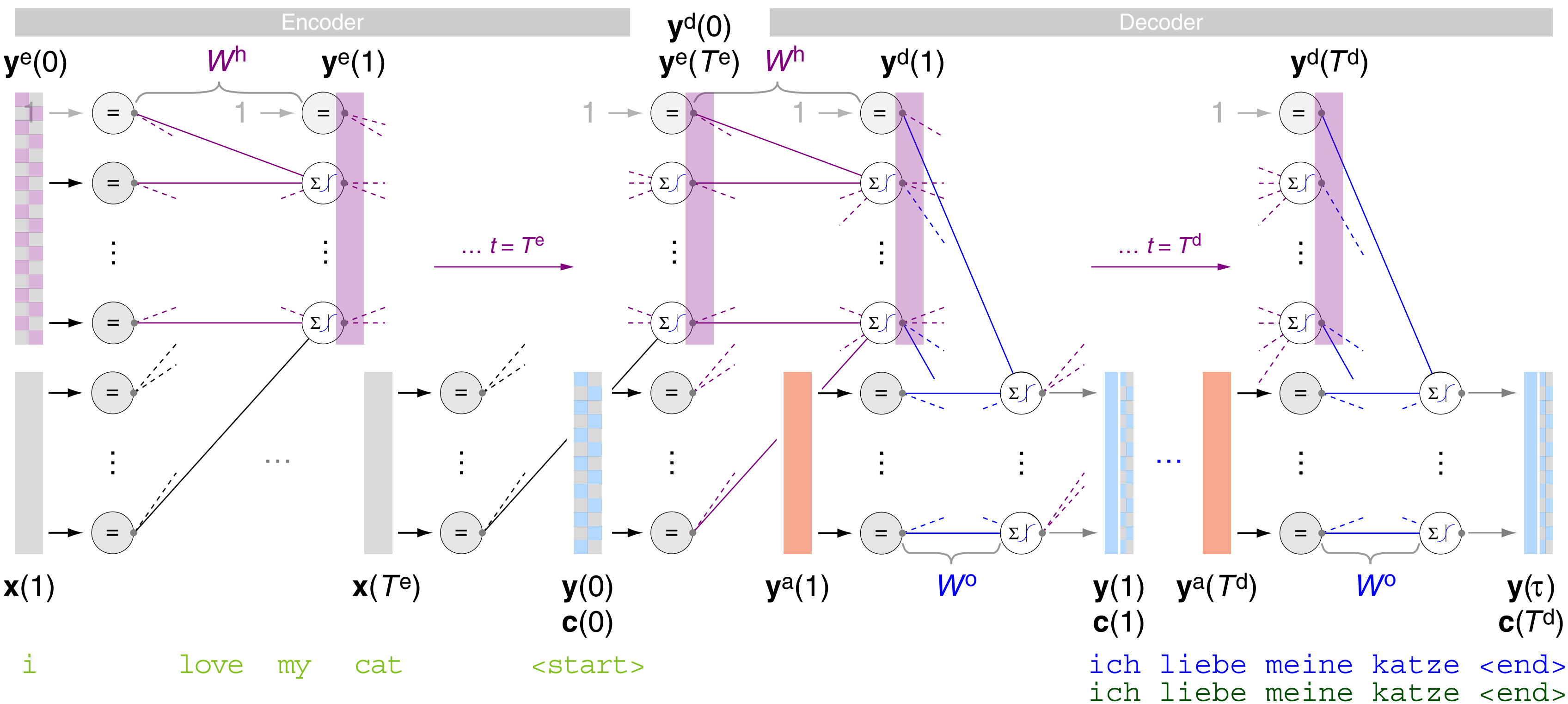
[notation: MLP matrices, computational graph, language model]





Recurrent Neural Networks

Notation I (MLP matrices) [notation: MLP matrices, computational graph, language model]



$\mathbf{x}(t)$ input
in_word

$\mathbf{y}^{h}(t)$ hidden
$\mathbf{y}^{e}(t)$ hidden encoder
$\mathbf{y}^{d}(t)$ hidden decoder

predefined hidden

$\mathbf{y}^{a}(t)$ attention

$\mathbf{y}(t)$ output
out_word (test)

$\mathbf{c}(t)$ target
out_word (training)

$\mathbf{y}(t)$ output or $\mathbf{c}(t)$ target
(depends on training or test phase)

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

`i love my cat` $\rightarrow \oplus$

(S2) class → sequence

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

$\oplus \rightarrow$ `i love my cat`

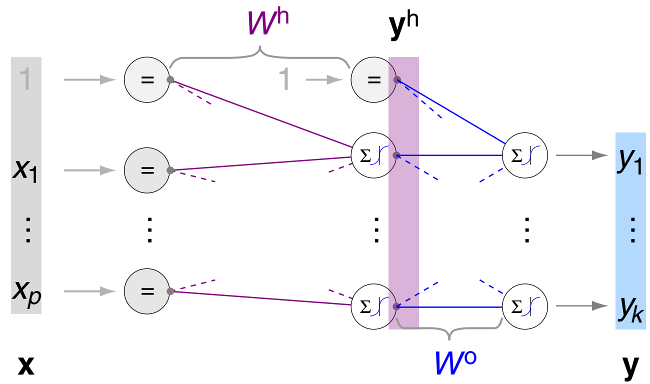
(S3) sequence → sequence

English sentence → German sentence

`i love my cat` $\rightarrow$ `ich liebe meine katze`

Recurrent Neural Networks

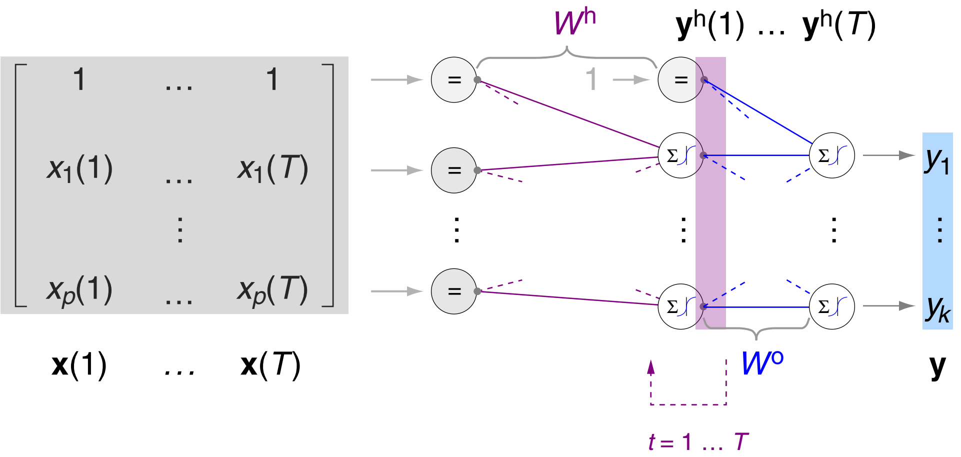
RNN Sequence Encoding



- 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

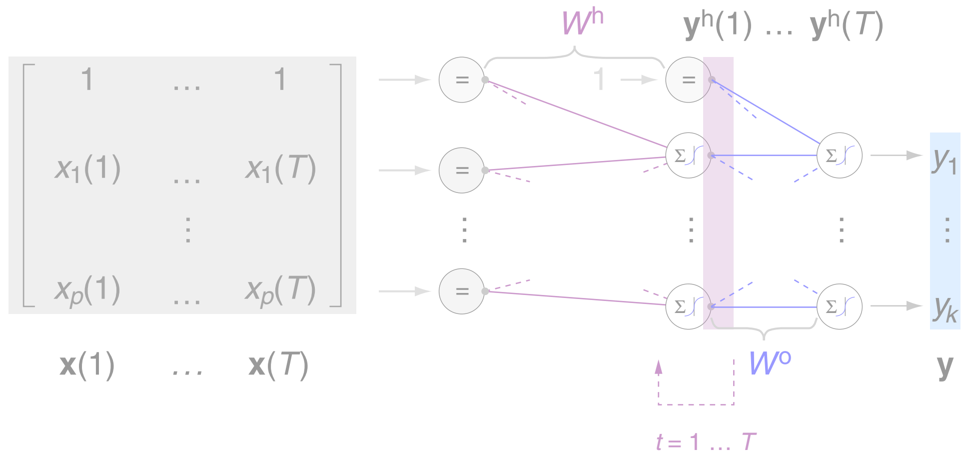
RNN Sequence Encoding (continued)



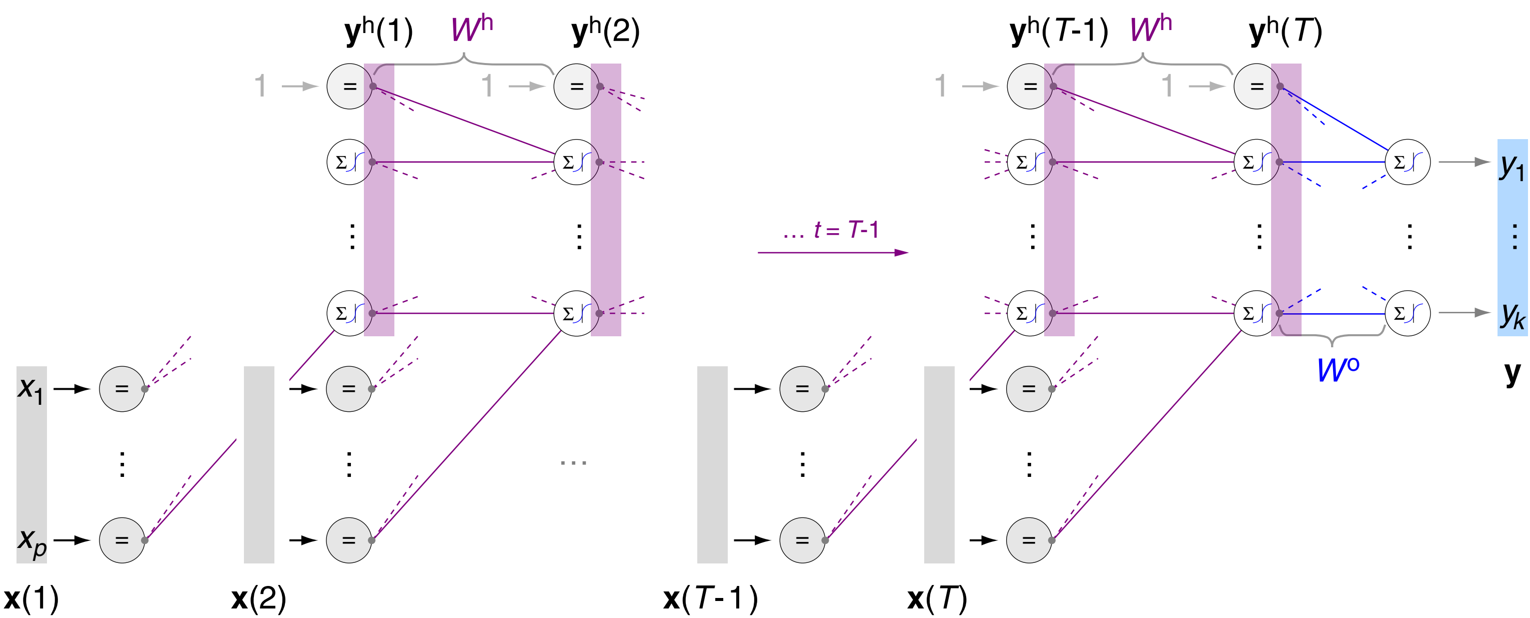
- Sequence of p-dimensional input vectors $[\mathbf{x}(1), \ldots, \mathbf{x}(T)]$.
- One hidden layer that is recurrently updated.
- One k-dimensional output vector $\mathbf{y}([\mathbf{x}(1), \ldots, \mathbf{x}(T)])$ or $\mathbf{y}$.

Recurrent Neural Networks

RNN Sequence Encoding (continued)

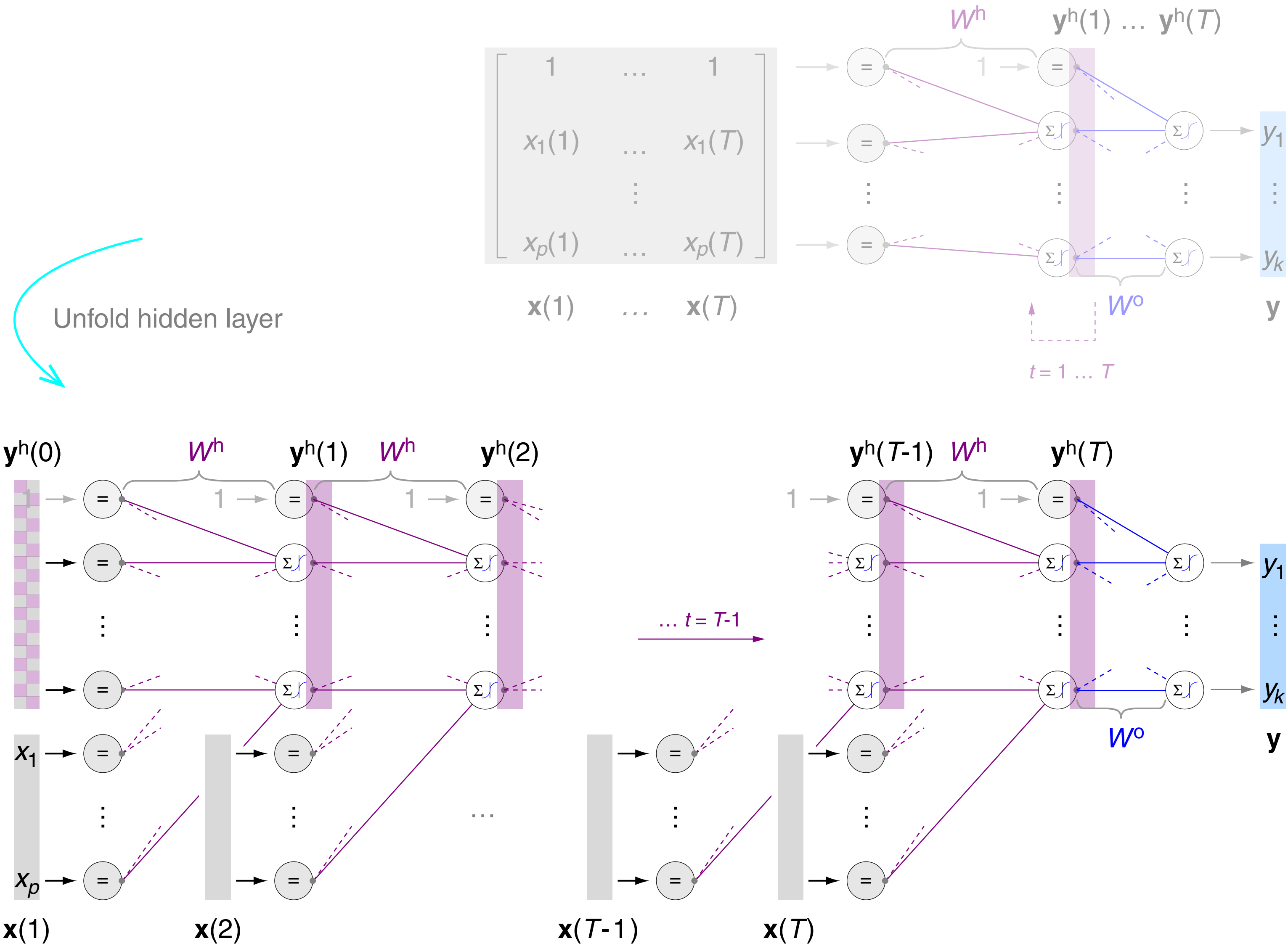


Unfold hidden layer



Recurrent Neural Networks

RNN Sequence Encoding (continued)

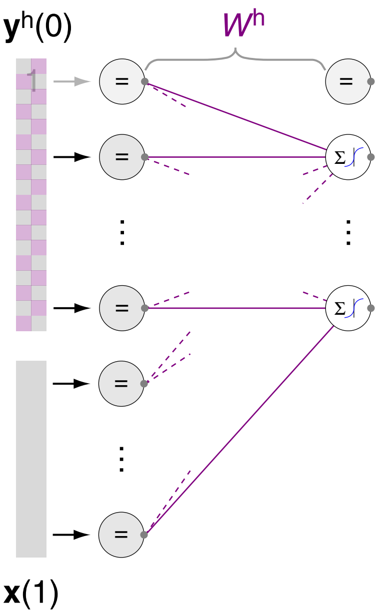


Remarks:

- An input sequence is written in brackets, $[\mathbf{x}(1), \ldots, \mathbf{x}(T)]$, where $\mathbf{x}(t), t = 1, \ldots, T$, 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.

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1

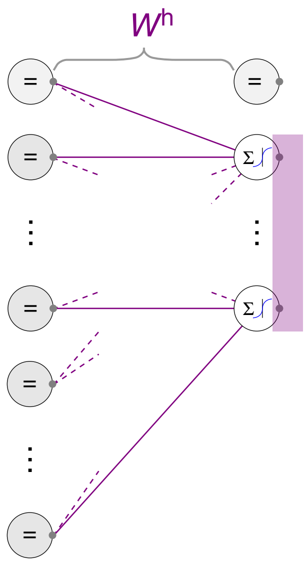


Input encoding over t.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1

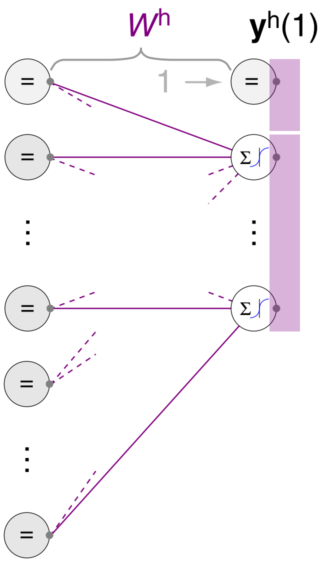


Input encoding over t.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1

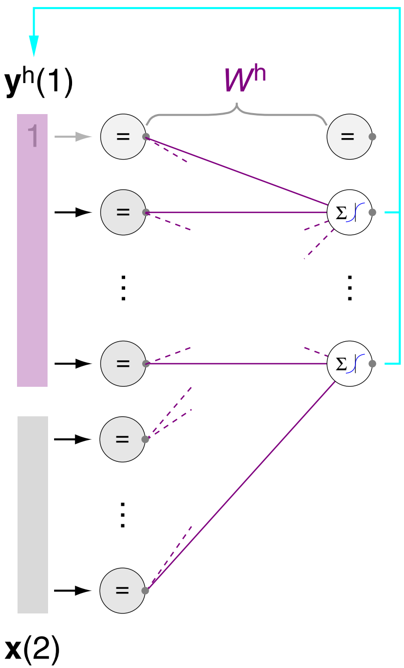


Input encoding over t.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1

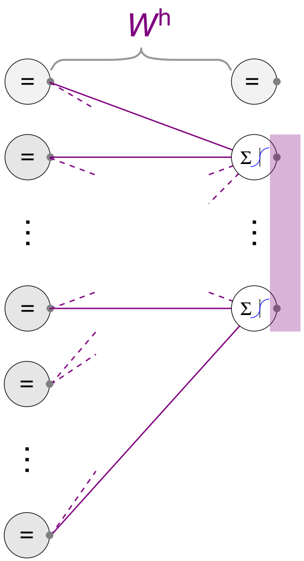


Input encoding over t.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 1 → 2

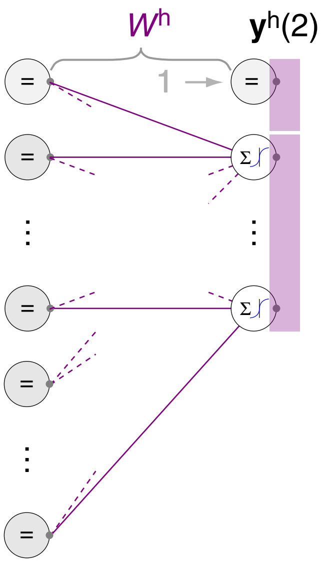


Input encoding over t.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 1 → 2

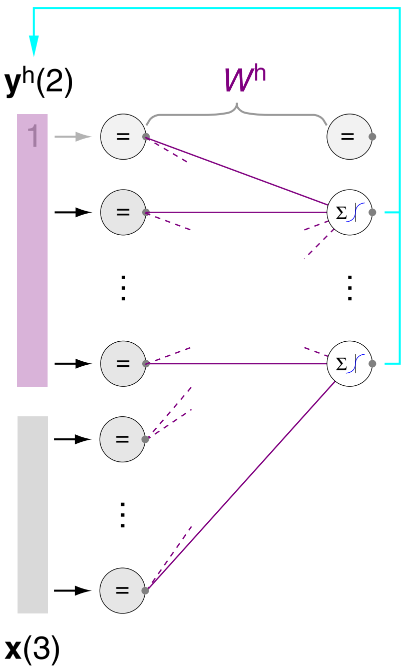


Input encoding over t.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 1 → 2

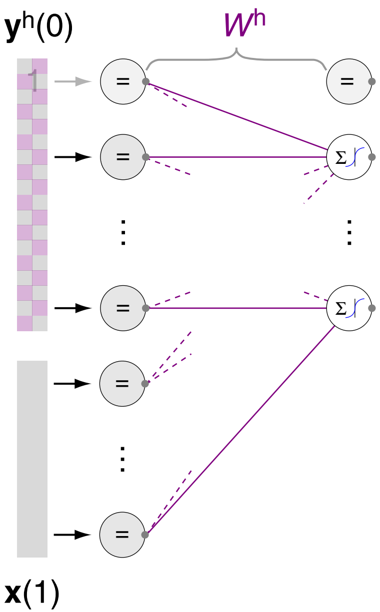


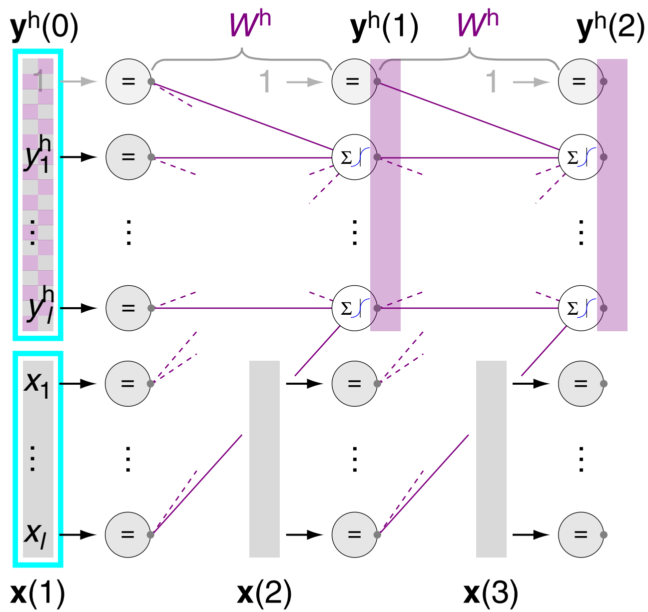
Input encoding over t.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1





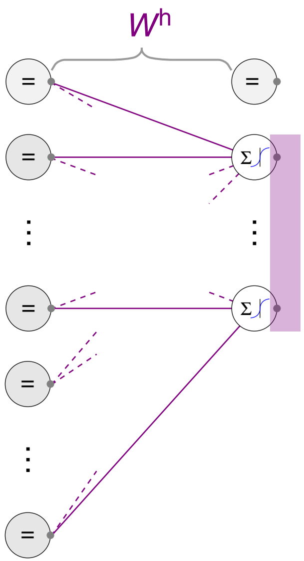
Input encoding over t.

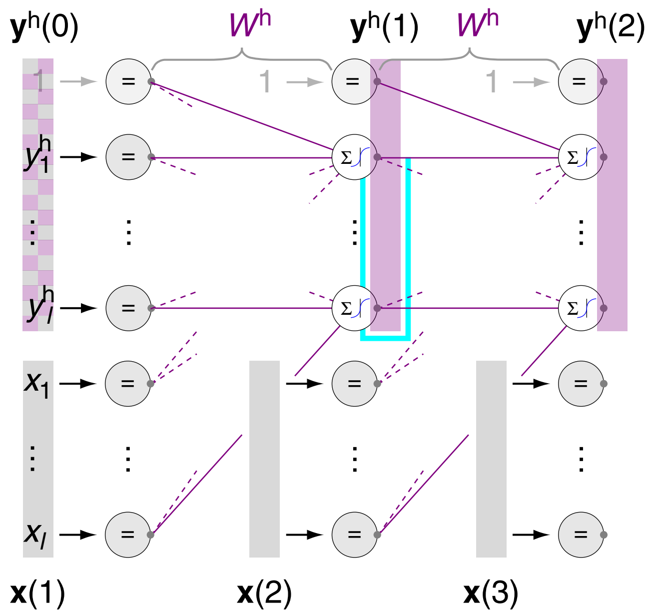
The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1





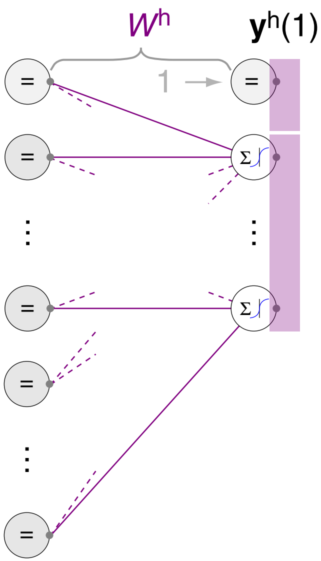
Input encoding over t.

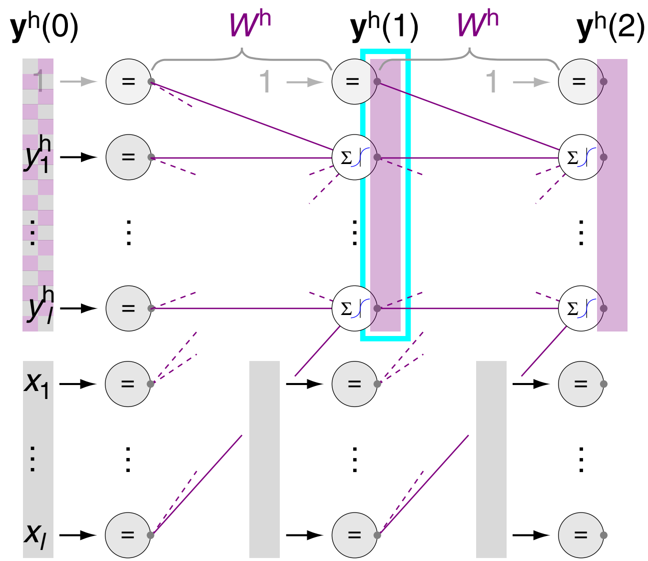
The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 0 → 1





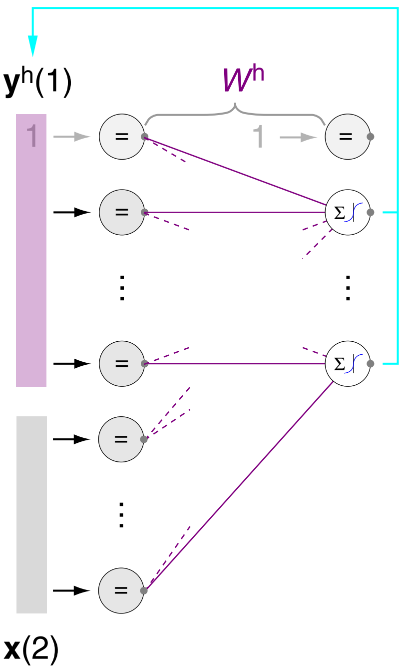
Input encoding over t.

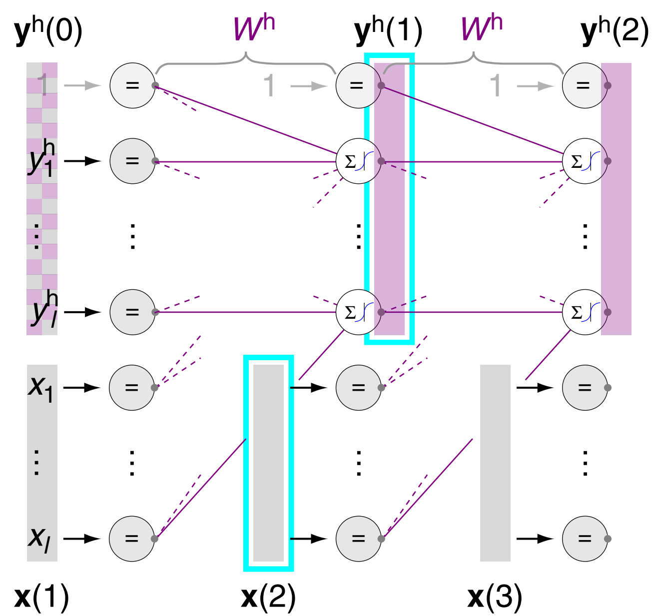
The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: $0 \rightarrow 1$





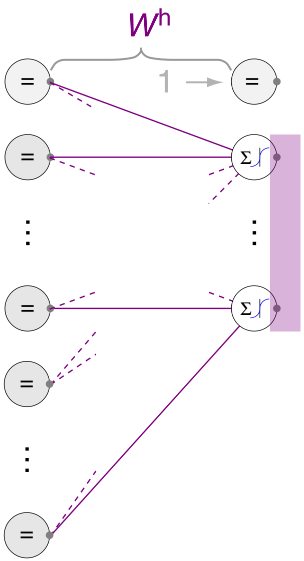
Input encoding over t.

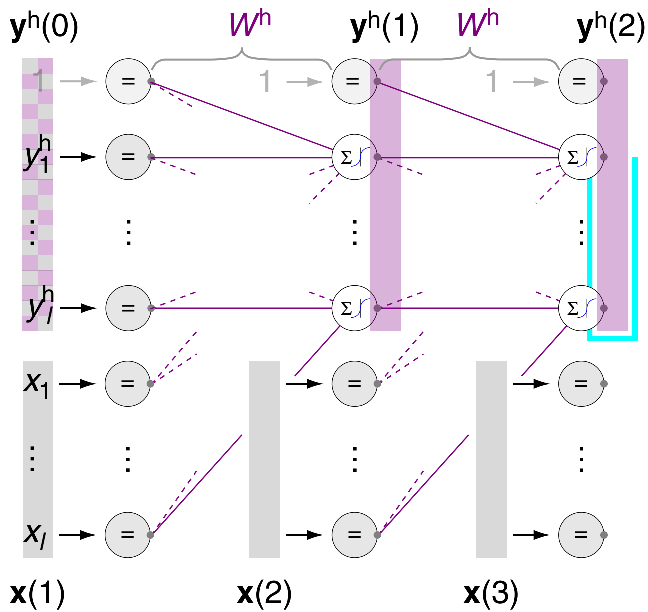
The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: $1 \rightarrow 2$





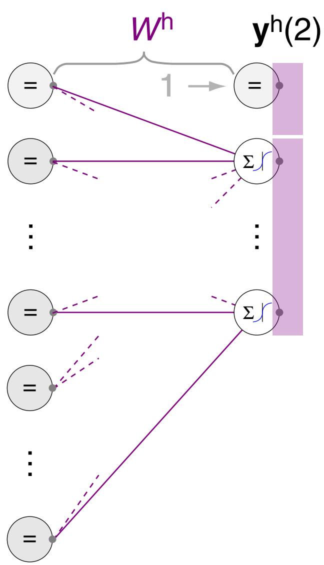
Input encoding over t.

The hidden layer at subsequent time steps.

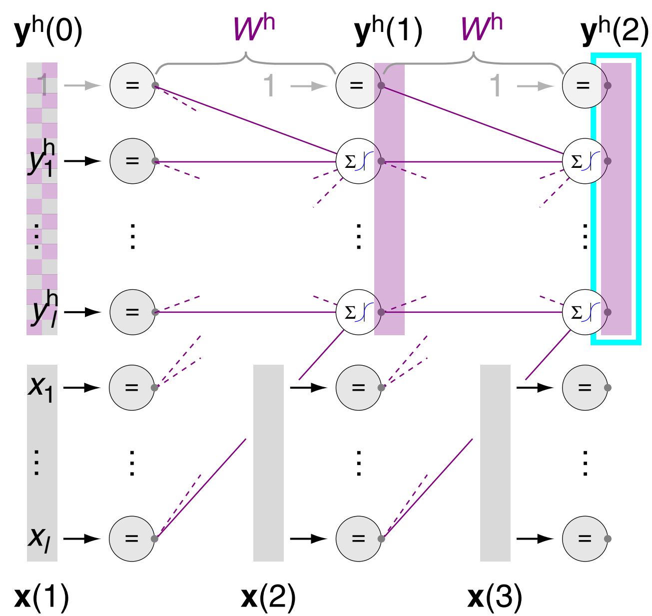
Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 1 → 2



Input encoding over t.

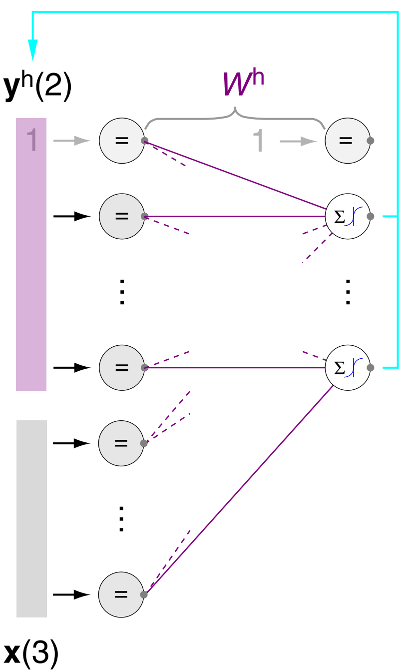


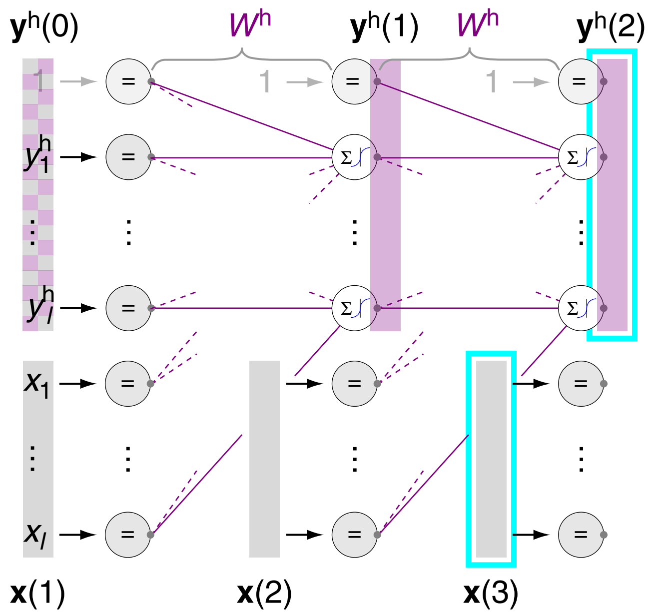
The hidden layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Encoding (continued) [encoding overview]

t: 1 → 2





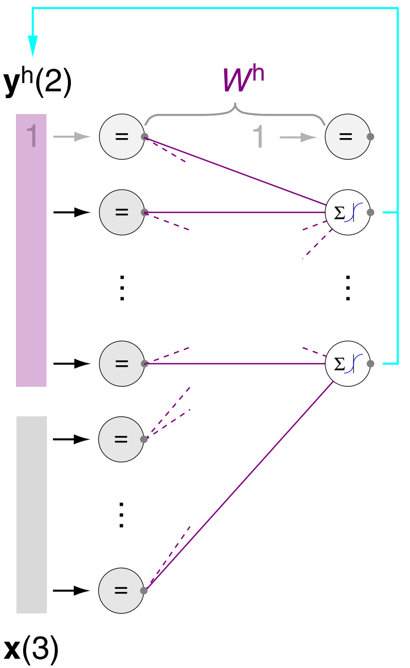
Input encoding over t.

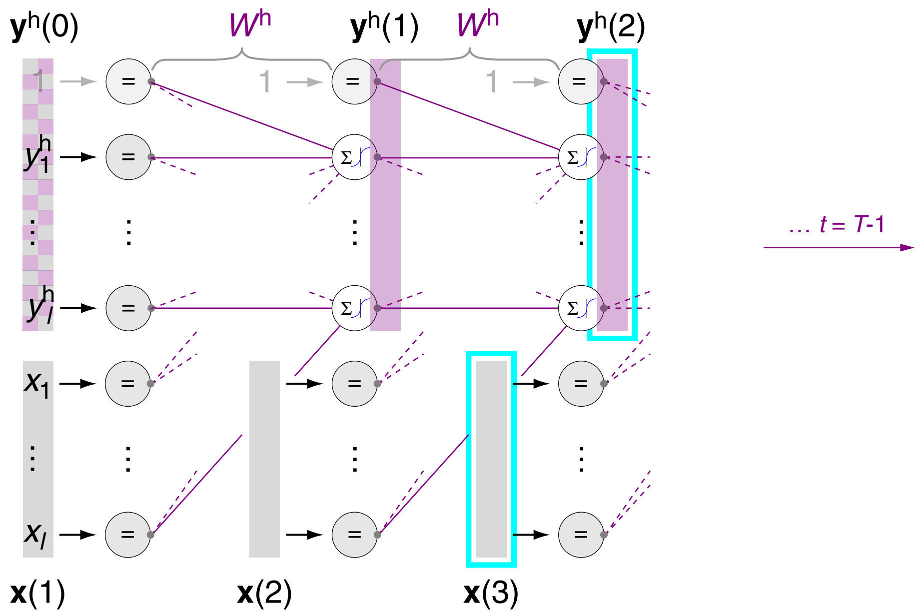
The hidden layer at subsequent time steps.

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

- 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:

$$[\mathbf{x}(1), \ldots, \mathbf{x}(4)] = \left[\begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix} \right]$$

$$\widehat{=} \; [\texttt{word\_11}, \texttt{word\_15}, \texttt{word\_16}, \texttt{word\_6}]$$

$\widehat{=}$ I love my cat

(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`)

Input:

$$[\mathbf{x}(1), \ldots, \mathbf{x}(4)] = \left[\begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix} \right]$$

$$\mathrel{\widehat{=}} [\texttt{word\_11}, \texttt{word\_15}, \texttt{word\_16}, \texttt{word\_6}]$$

$\mathrel{\widehat{=}}$ I love my cat

Output:

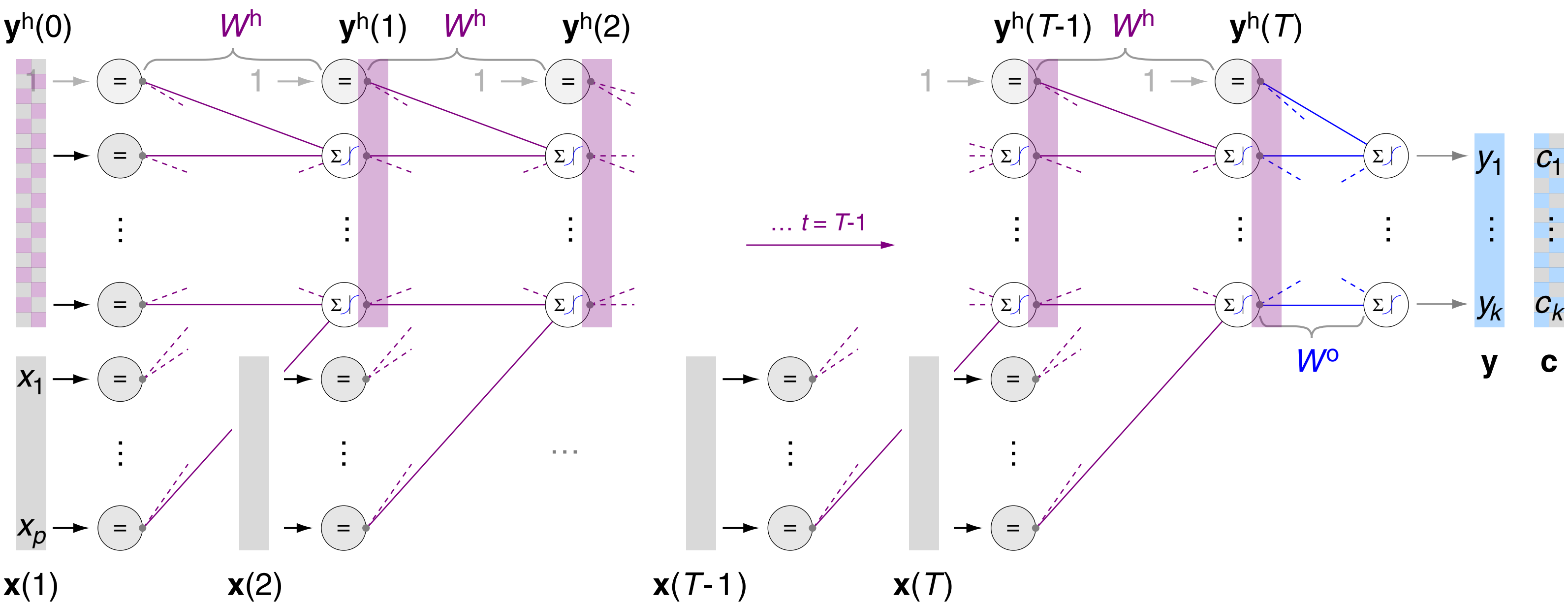
$$\mathbf{y}([\mathbf{x}(1), \ldots, \mathbf{x}(4)]) = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

Target:

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

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs



Input:

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

Hidden:

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

Target:

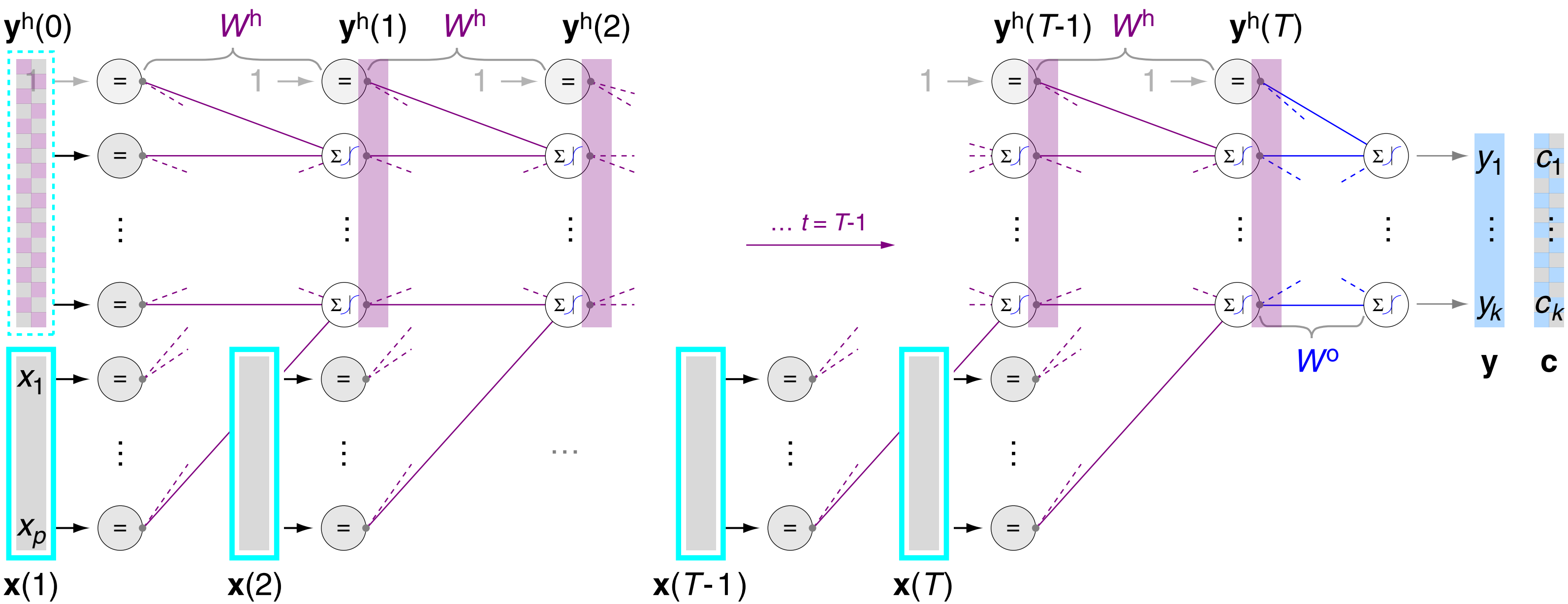
$\mathbf{c}$

Output:

$\mathbf{y} = \boldsymbol{\sigma}_{\Delta}\left(W^o\, \mathbf{y}^h(T)\right)$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs



Input:

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

Output:

$\mathbf{y} = \boldsymbol{\sigma}_{\Delta}\left(W^{o}\, \mathbf{y}^{h}(T)\right)$

Hidden:

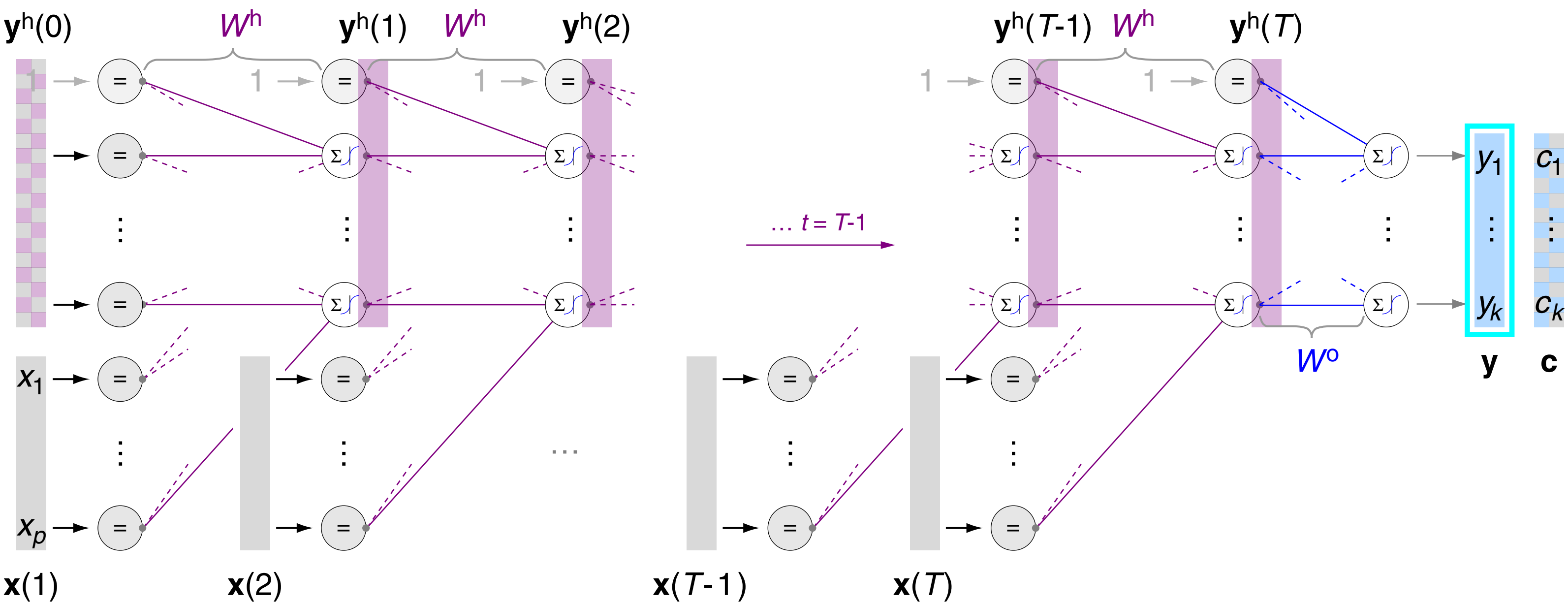
$\mathbf{y}^{h}(t) = \boldsymbol{\sigma}\left(W^{h}\begin{pmatrix}\mathbf{y}^{h}(t-1)\\ \mathbf{x}(t)\end{pmatrix}\right), t = 1, \ldots, T$

Target:

$\mathbf{c}$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs



Input:

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

Hidden:

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

Target:

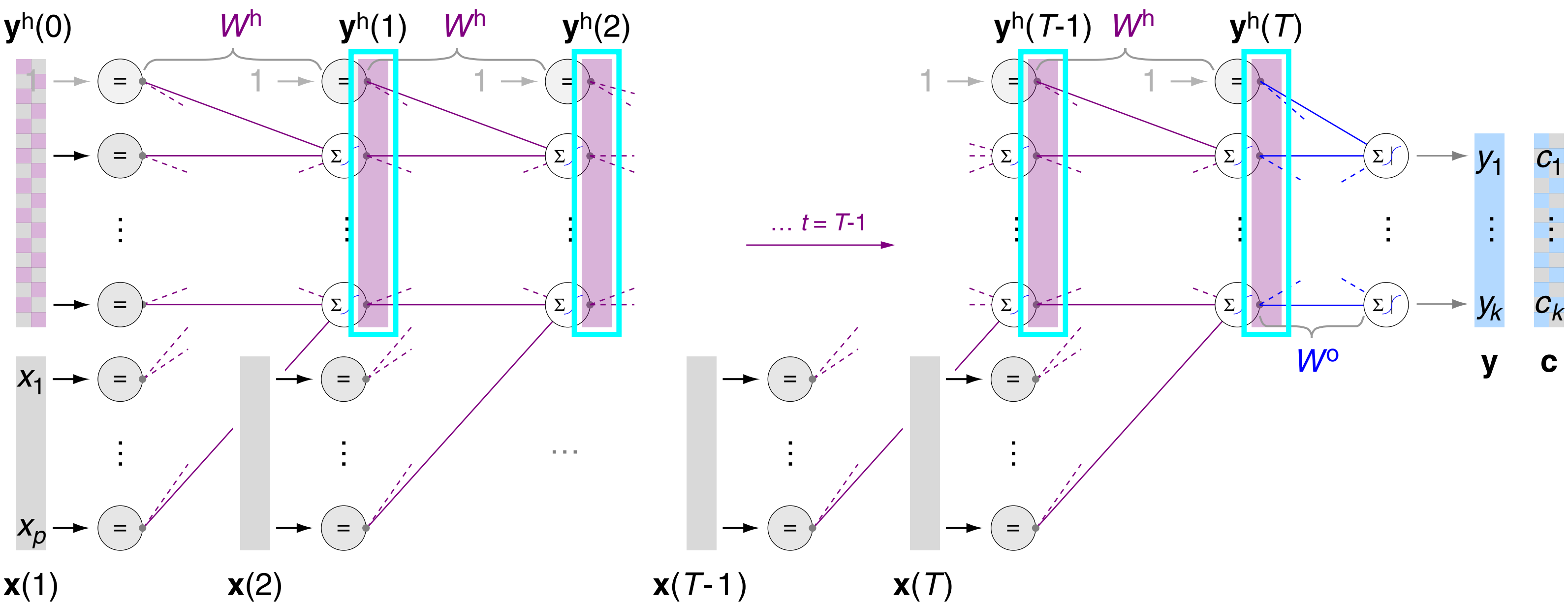
$\mathbf{c}$

Output:

$\mathbf{y} = \boldsymbol{\sigma}_{\Delta}\left(W^{\mathrm{o}}\,\mathbf{y}^{\mathrm{h}}(T)\right)$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs



Input:

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

Output:

$\mathbf{y} = \boldsymbol{\sigma}_{\triangle}\left(W^{\mathrm{o}}\, \mathbf{y}^{\mathrm{h}}(T)\right)$

Hidden:

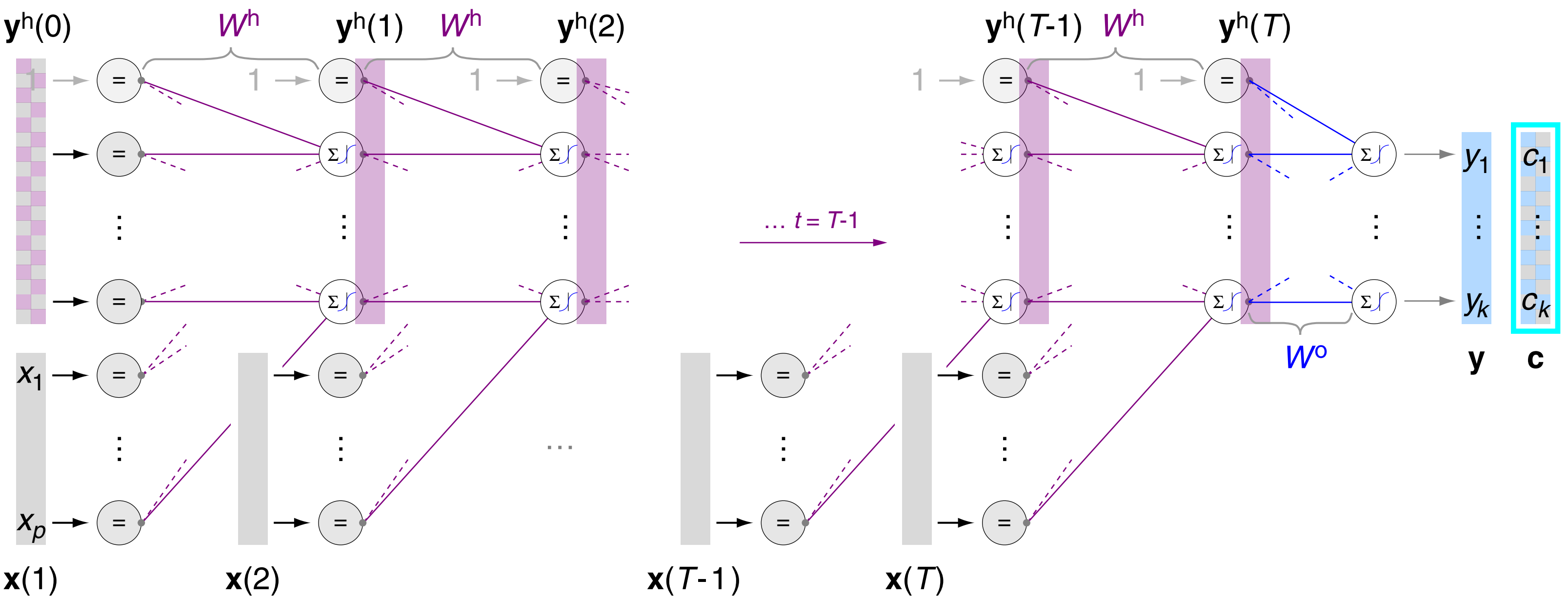
$$\mathbf{y}^{\mathrm{h}}(t) = \boldsymbol{\sigma}\left(W^{\mathrm{h}}\begin{pmatrix}\mathbf{y}^{\mathrm{h}}(t-1)\\ \mathbf{x}(t)\end{pmatrix}\right), t = 1, \ldots, T$$

Target:

$\mathbf{c}$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs



Input:

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

Hidden:

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

Target:

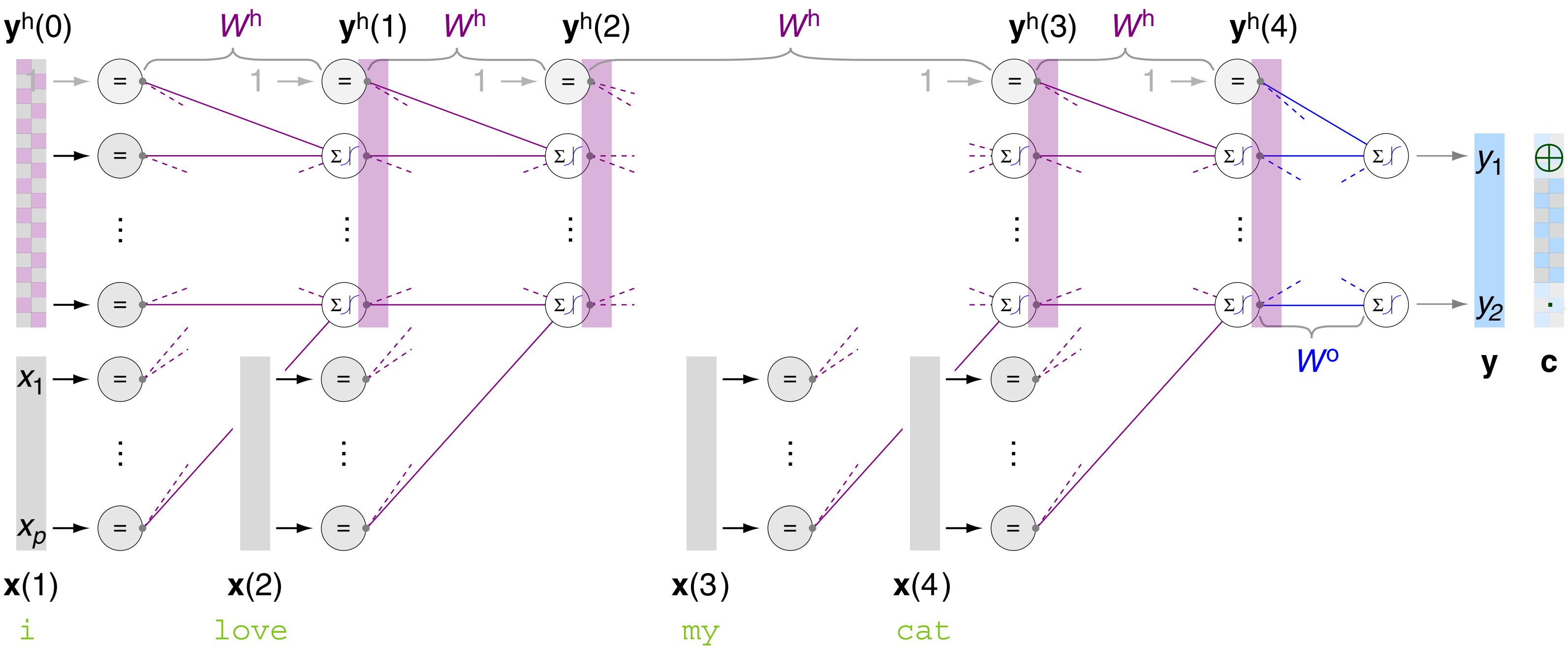
$\mathbf{c}$

Output:

$$\mathbf{y} = \boldsymbol{\sigma}_{\triangle}\left(W^{\mathsf{o}}\,\mathbf{y}^{\mathsf{h}}(T)\right)$$

Recurrent Neural Networks

(S1) Sequence-to-Class Mapping with RNNs



Input:

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

Output:

$$\mathbf{y} = \boldsymbol{\sigma}_{\Delta}\left(W^o\, \mathbf{y}^h(4)\right)$$

Hidden:

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

Target:

$$\mathbf{c}$$

Remarks:

- We denote $\mathbf{y}^{\mathsf{h}}(0)$ 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 $\mathbf{x}(t)$.

  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. $\boldsymbol{\sigma}_{\Delta}()$ denotes the softmax function. $\boldsymbol{\sigma}_{\Delta} : \mathbf{R}^k \to \Delta^{k-1}$, generalizes the logistic (sigmoid) function to k dimensions (to k exclusive classes), where $\boldsymbol{\sigma}_{\Delta}(\mathbf{z})|_i = e^{z_i} / \sum_{j=1}^{k} e^{z_j}$. [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]

The IGD Algorithm for Sequence-to-Class Tasks [$\text{IGD}_{\text{c2seq}}$]

Algorithm:	$\text{IGD}_{\text{seq2c}}$	Incremental Gradient Descent for RNNs at seq2class tasks.
Input:	D	Multiset of examples $([\mathbf{x}(1), \ldots, \mathbf{x}(T)], \mathbf{c})$ with $\mathbf{x}(t) \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weight matrices. (= hypothesis)

1. *initialize_random_weights*$(W^{\text{h}}, W^{\text{o}})$, $\mathbf{y}^{\text{h}}(0) = (0, \ldots, 0)^T$, $t_{\text{epoch}} = 0$
2. **REPEAT**
3. $t_{\text{epoch}} = t_{\text{epoch}} + 1$
4. FOREACH $([\mathbf{x}(1), \ldots, \mathbf{x}(T)], \mathbf{c}) \in D$ DO
5. Model function evaluation.
6. Calculation of residual vector.
7a.
7b. Calculation of derivative of the loss.
8. Parameter update $\widehat{=}$ one gradient step down.
9. ENDDO
10. **UNTIL**(*convergence*$(D, \mathbf{y}(\cdot), t_{\text{epoch}})$)
11. *return*$(W^{\text{h}}, W^{\text{o}})$

The IGD Algorithm for Sequence-to-Class Tasks [$\text{IGD}_{\text{c2seq}}$]

Algorithm:	$\text{IGD}_{\text{seq2c}}$	Incremental Gradient Descent for RNNs at seq2class tasks.
Input:	D	Multiset of examples $([\mathbf{x}(1), \ldots, \mathbf{x}(T)], \mathbf{c})$ with $\mathbf{x}(t) \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\mathsf{h}}, W^{\mathsf{o}}$	Weight matrices. (= hypothesis)

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

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

$\quad\quad$ ENDDO

$\quad\quad \mathbf{y} = \boldsymbol{\sigma}_{\triangle}\left(W^{\mathsf{o}}\,\mathbf{y}^{\mathsf{h}}(T)\right)$

6. $\quad\quad \boldsymbol{\delta} = \mathbf{c} - \mathbf{y}([\mathbf{x}(1), \ldots, \mathbf{x}(T)])$

7a. $\quad\quad \ell(\mathbf{w}) = l(\boldsymbol{\delta}) + \frac{\lambda}{n} R(\mathbf{w})$, $\nabla \ell(\mathbf{w}) = \text{autodiff}(\ell(), \mathbf{w})$ // backpropagation (7a+7b)

7b. $\quad\quad {}_{\triangle}W^{\mathsf{h}} = \eta \cdot \nabla^{\mathsf{h}} \ell(\mathbf{w})$, ${}_{\triangle}W^{\mathsf{o}} = \eta \cdot \nabla^{\mathsf{o}} \ell(\mathbf{w})$

8. $\quad\quad W^{\mathsf{h}} = W^{\mathsf{h}} + {}_{\triangle}W^{\mathsf{h}}$, $W^{\mathsf{o}} = W^{\mathsf{o}} + {}_{\triangle}W^{\mathsf{o}}$
9. $\quad$ ENDDO
10. **UNTIL**(*convergence*$(D, \mathbf{y}(\cdot), t_{\text{epoch}})$)
11. *return*$(W^{\mathsf{h}}, W^{\mathsf{o}})$

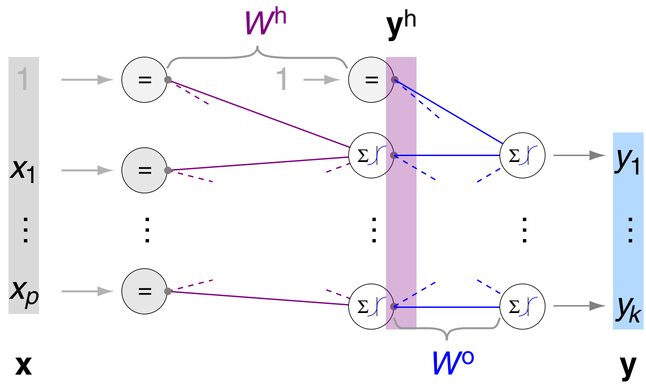
Recurrent Neural Networks

Types of Learning Tasks [Recap]

(S1)	sequence $\rightarrow$ class	sentence $\rightarrow \{\oplus, \ominus\}$ `i love my cat` $\rightarrow \oplus$
(S2)	class $\rightarrow$ sequence	$\{\oplus, \ominus\} \rightarrow$ sentence $\oplus \rightarrow$ `i love my cat`
(S3)	sequence $\rightarrow$ sequence	English sentence $\rightarrow$ German sentence `i love my cat` $\rightarrow$ `ich liebe meine katze`

Recurrent Neural Networks

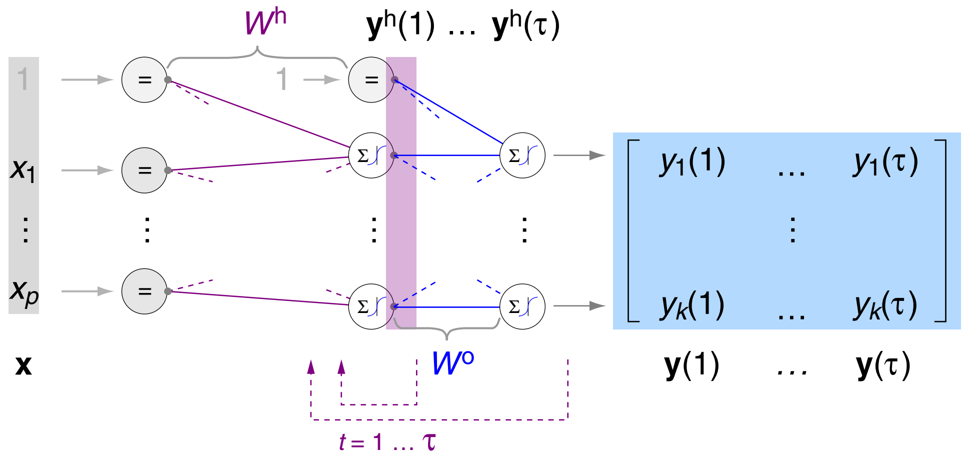
RNN Sequence Decoding



- 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

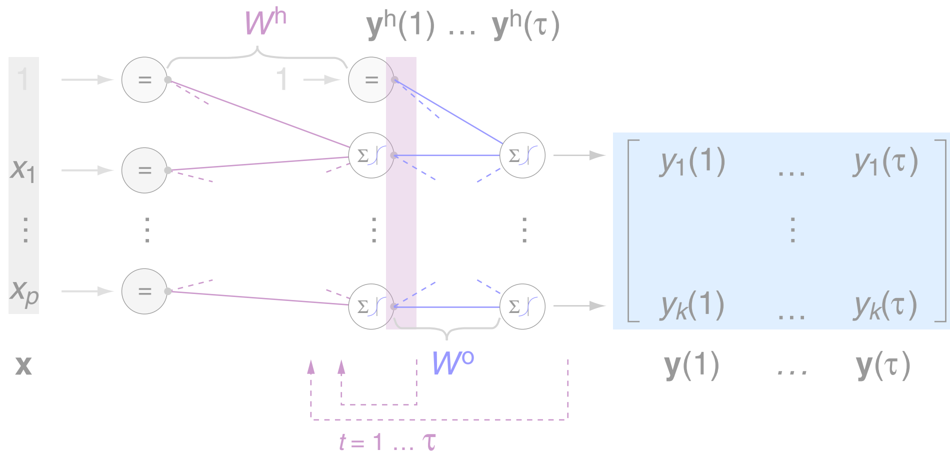
RNN Sequence Decoding (continued)



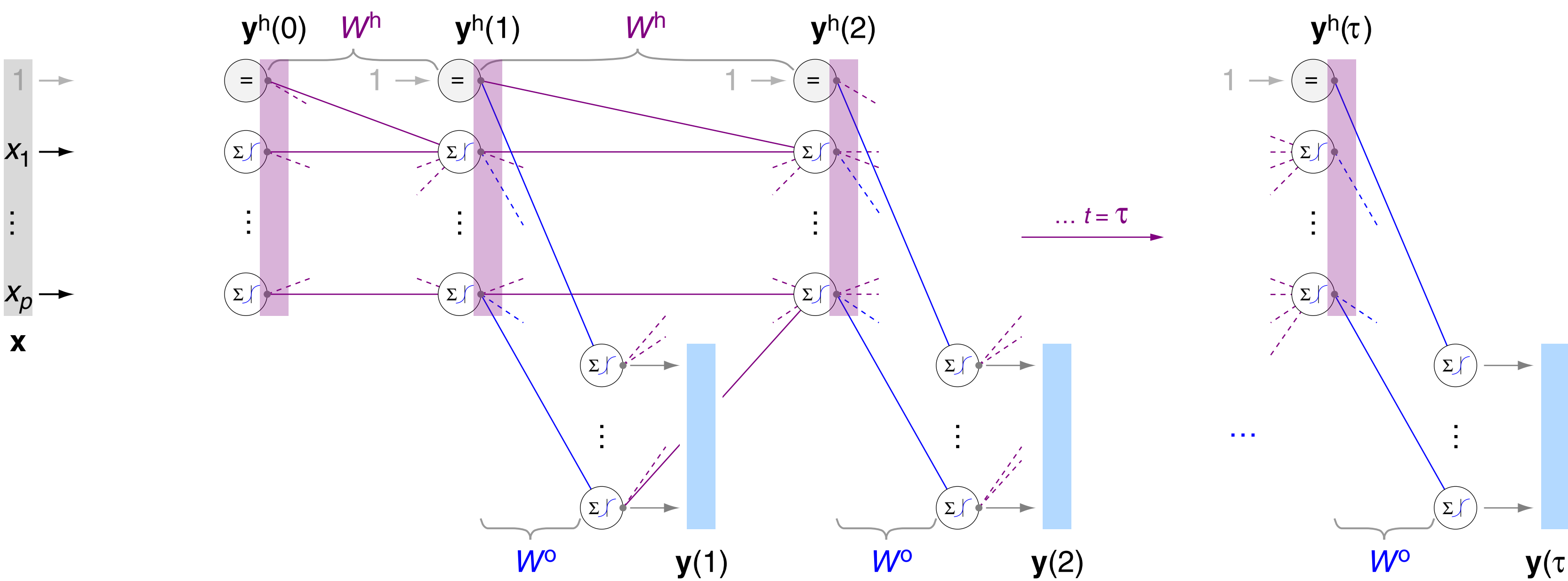
- One p-dimensional input vector $\mathbf{x}$.
- One hidden and one output layer, which are recurrently updated.
- Sequence of k-dimen. output vectors $[\mathbf{y}(\mathbf{x}, 1), \ldots, \mathbf{y}(\mathbf{x}, \tau)]$ or $[\mathbf{y}(1), \ldots, \mathbf{y}(\tau)]$.

Recurrent Neural Networks

RNN Sequence Decoding (continued)

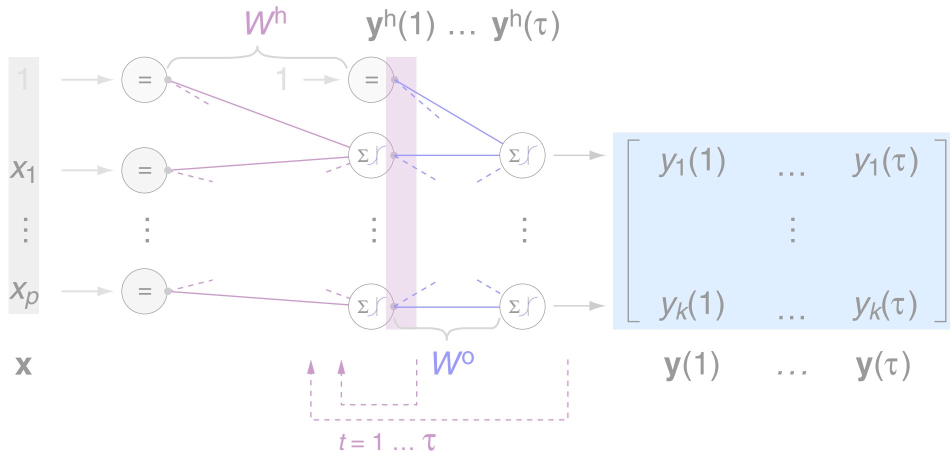


Unfold hidden and output layer

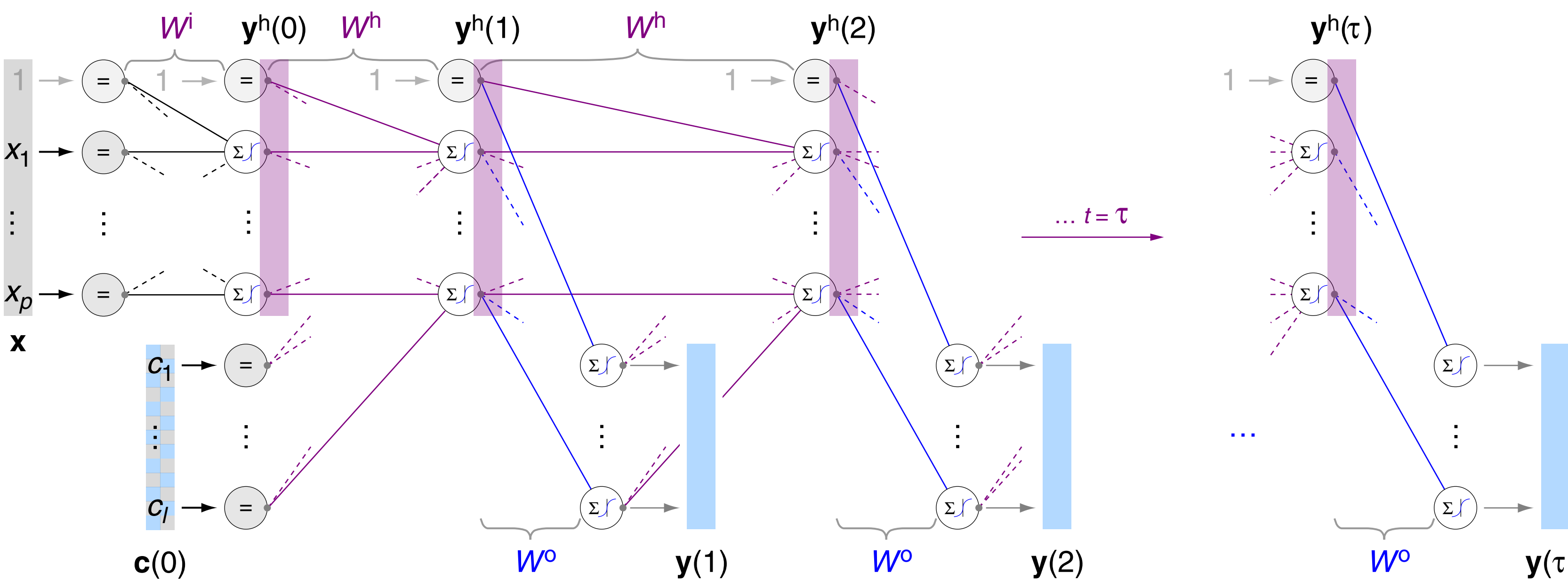


Recurrent Neural Networks

RNN Sequence Decoding (continued)



Unfold hidden and output layer



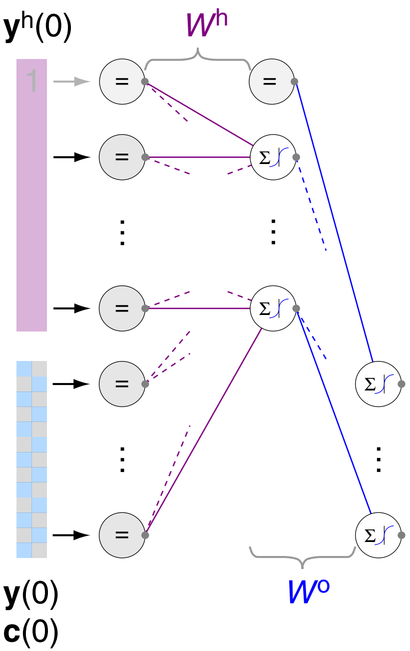
Remarks:

- An output sequence is written in brackets, $[\mathbf{y}(1), \ldots, \mathbf{y}(\tau)]$, where $\mathbf{y}(t), t = 1, \ldots, \tau$, 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 $\mathbf{y}^h$: $\mathbf{y}^{\mathsf{h}}(0) = \boldsymbol{\sigma}\left(W^{\mathsf{i}}\mathbf{x}\right)$.
- The parameter τ in $\mathbf{y}(\tau)$ is unknown. More specifically, the generation process terminates at that time step τ for which $\mathbf{y}(\tau) = (0, 0, \ldots, 0, 1)^T$ ($\widehat{=}$ $\mathtt{<end>}$).

  τ (the length of the computed output) does not have to be equal to T (the length of the target or ground truth).

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1

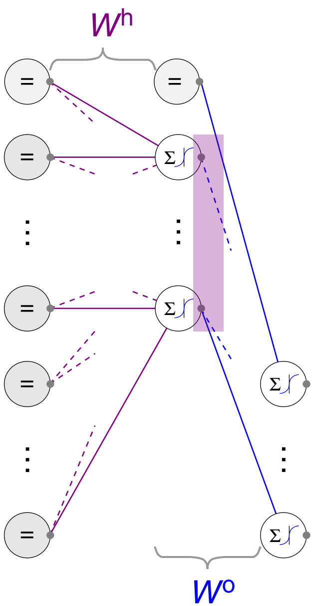


Output decoding over t.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1

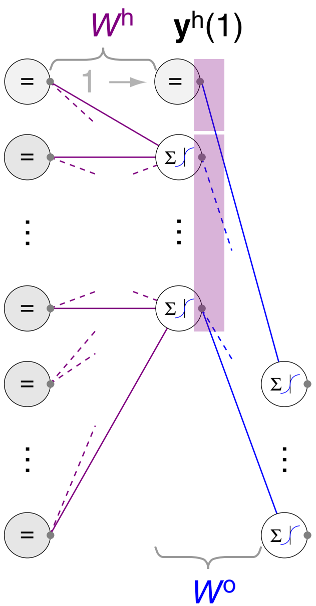


Output decoding over t.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $0 \rightarrow 1$

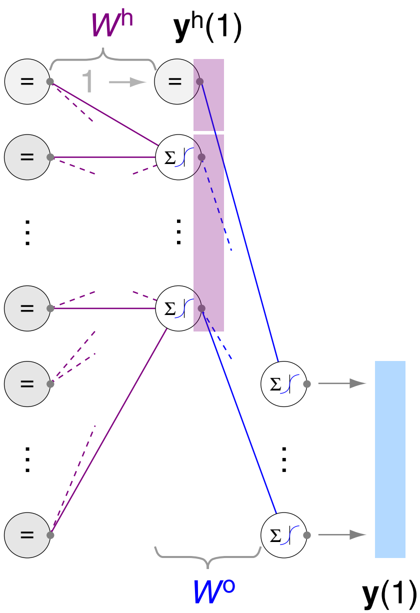


Output decoding over t.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 $\rightarrow$ 1

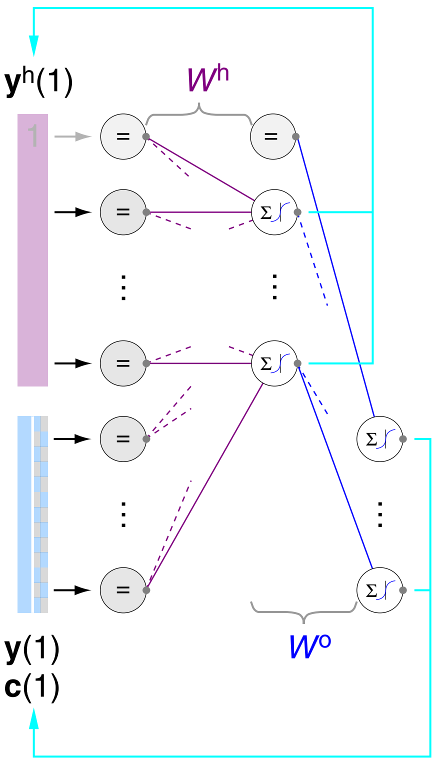


Output decoding over t.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1

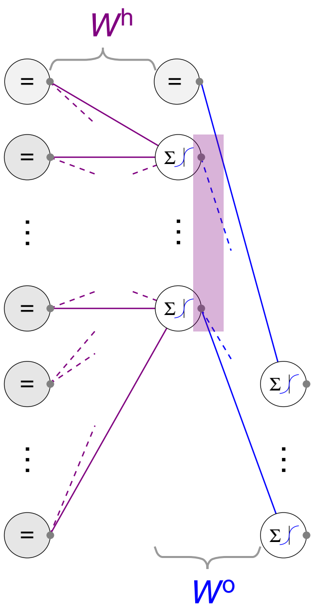


Output decoding over t.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2

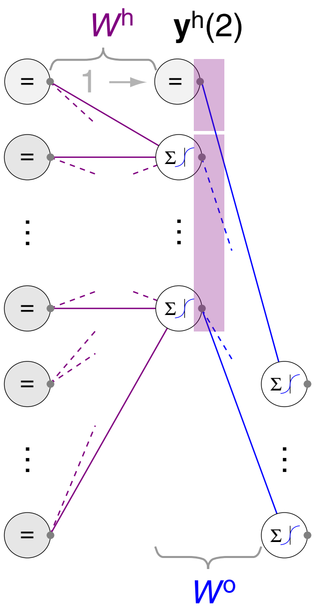


Output decoding over t.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2

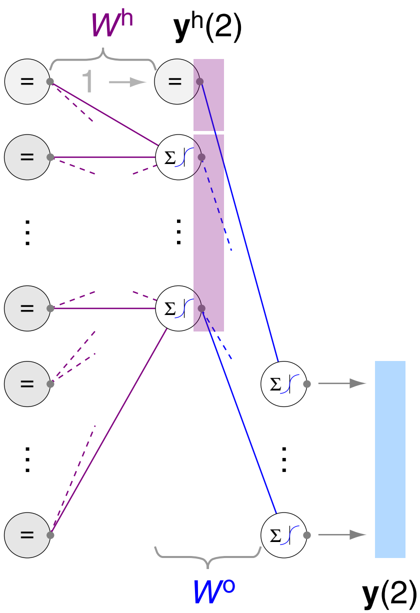


Output decoding over t.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2

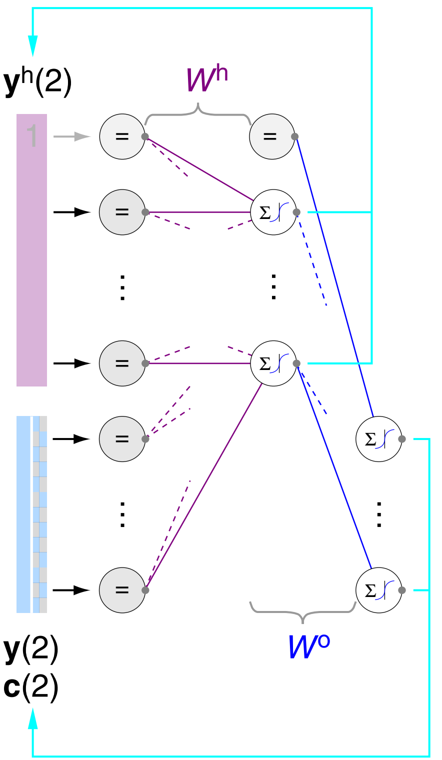


Output decoding over t.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2

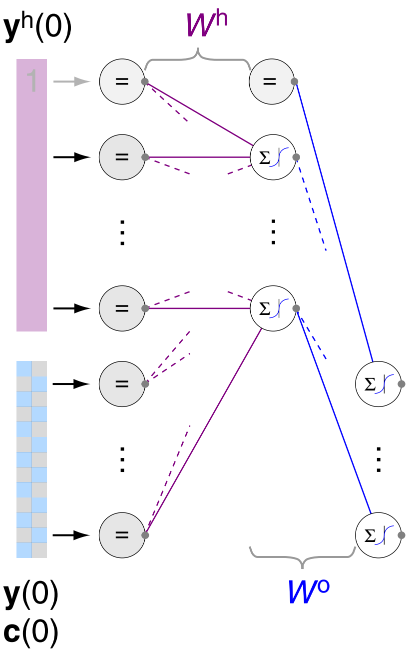


Output decoding over t.

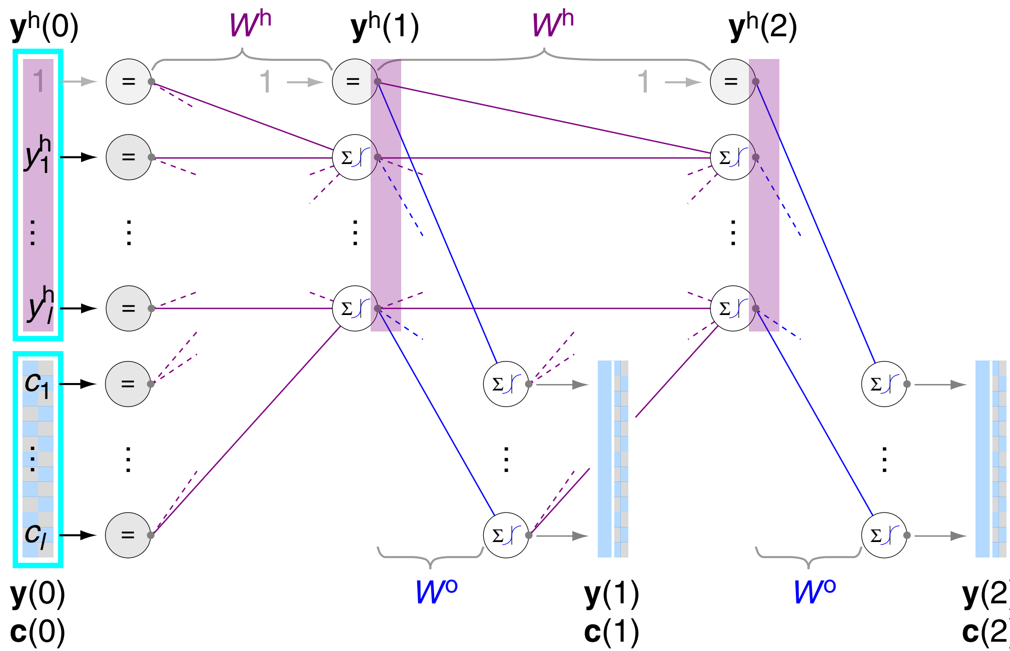
Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1



Output decoding over t.

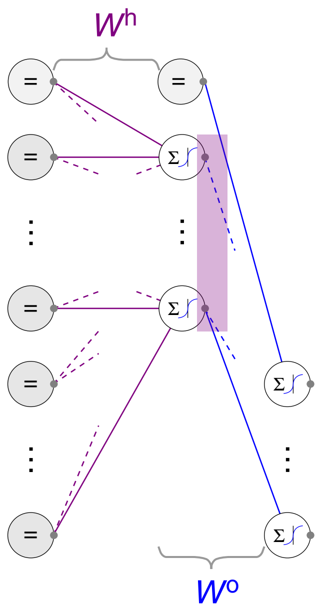


Hidden and output layer at subsequent time steps.

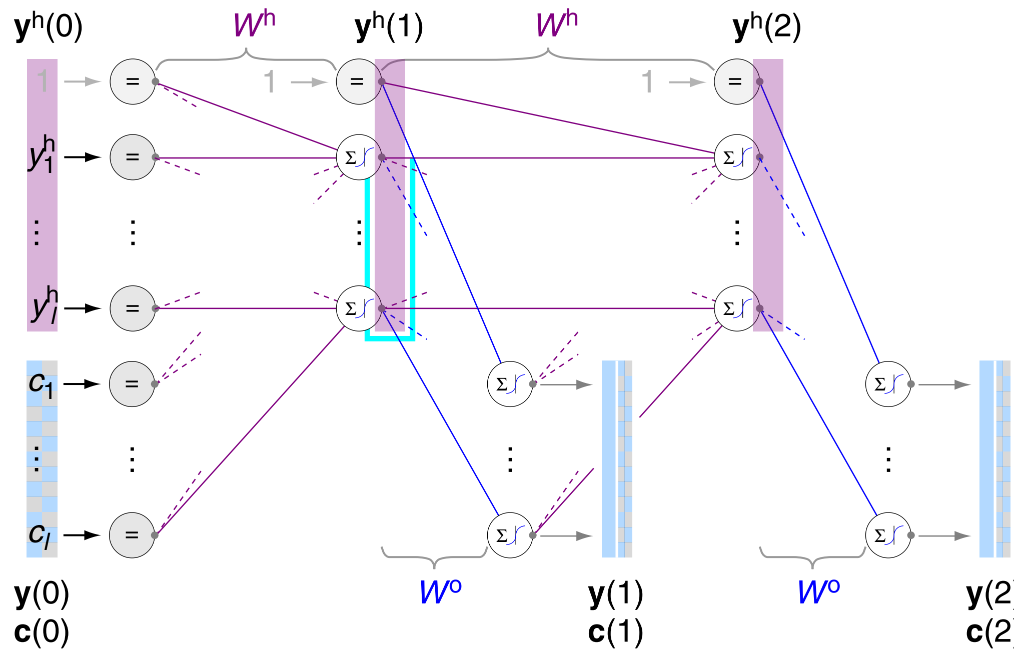
Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $0 \rightarrow 1$



Output decoding over t.

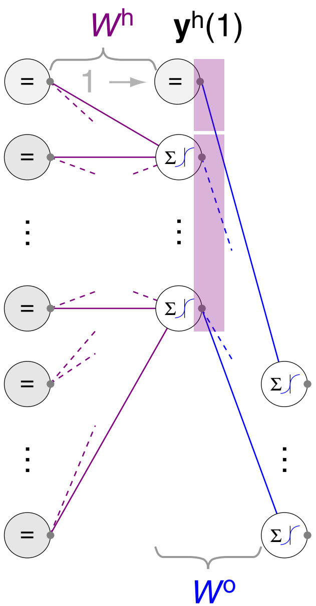


Hidden and output layer at subsequent time steps.

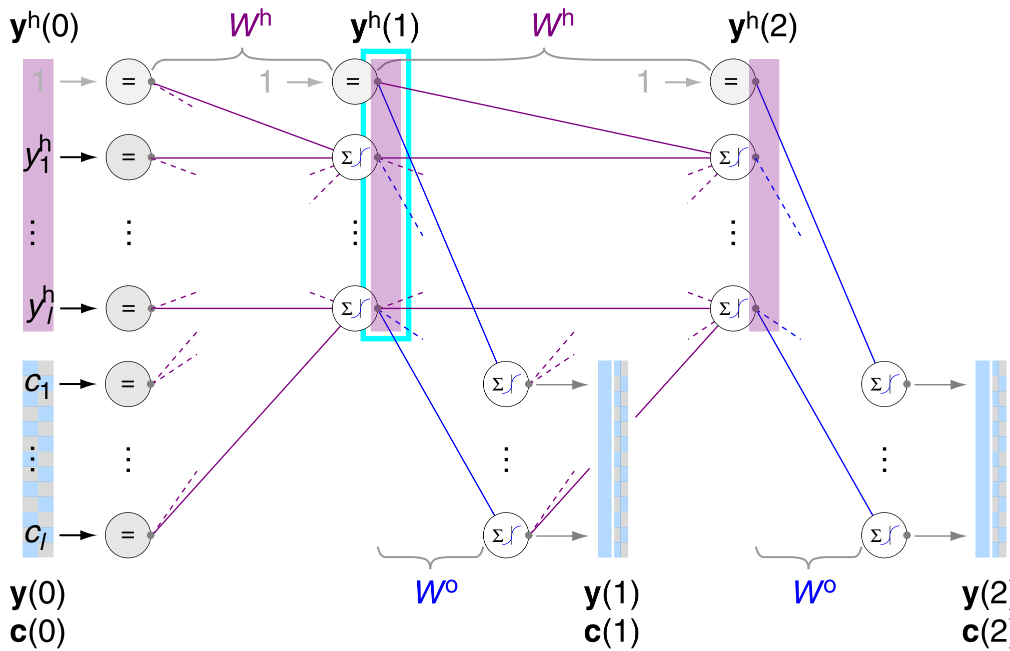
Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: $0 \rightarrow 1$



Output decoding over t.

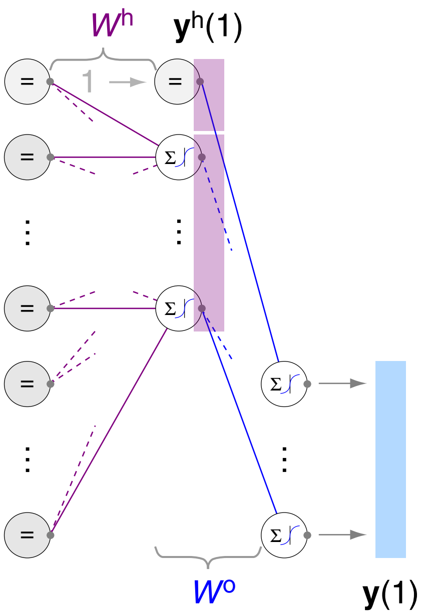


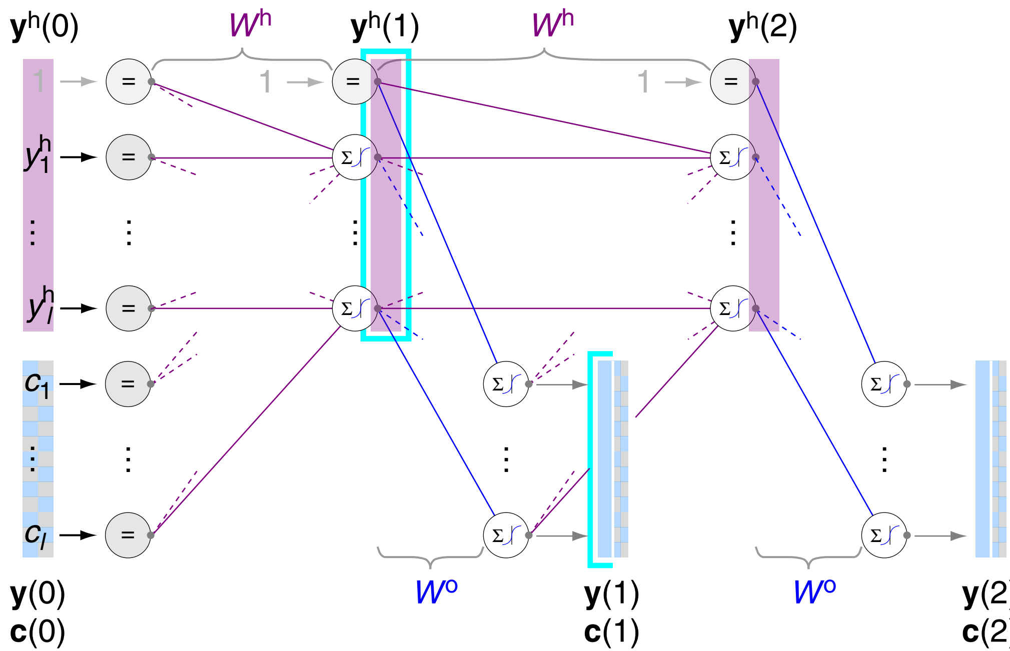
Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1





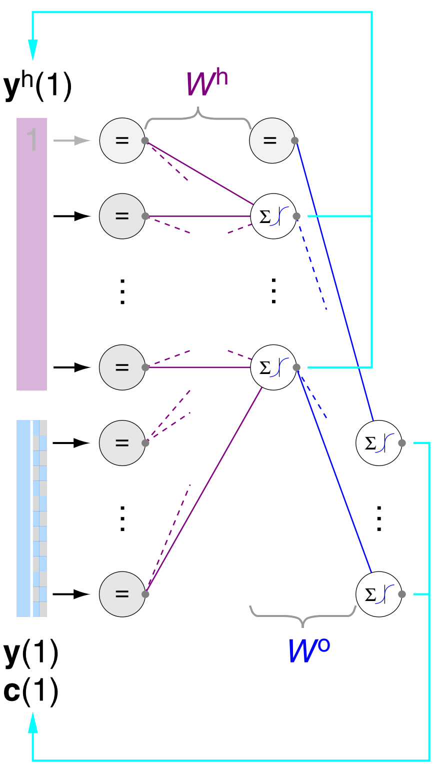
Output decoding over t.

Hidden and output layer at subsequent time steps.

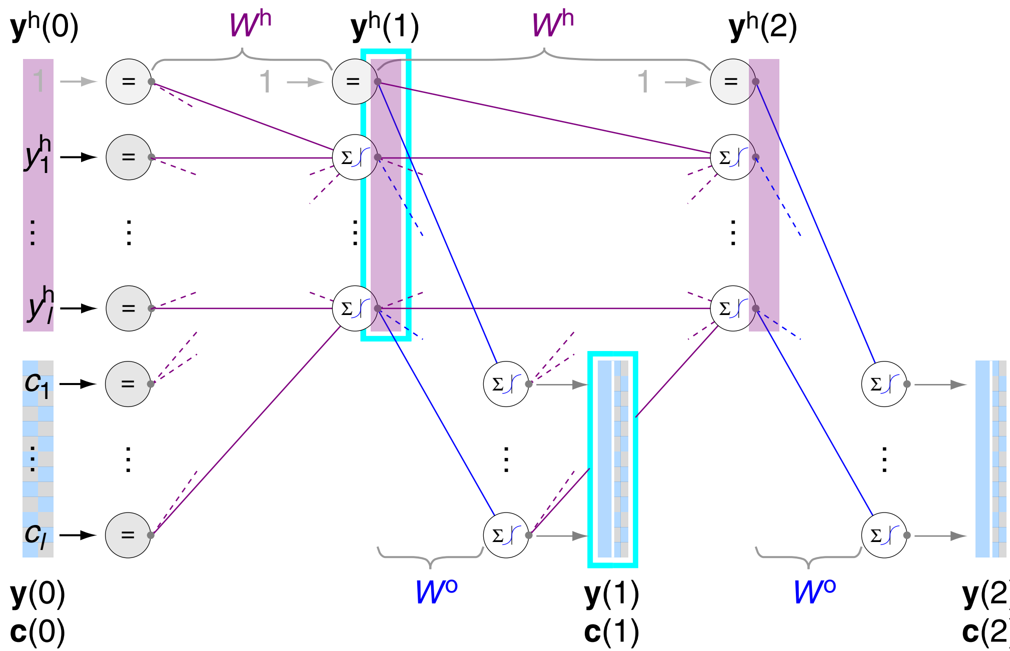
Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 0 → 1



Output decoding over t.

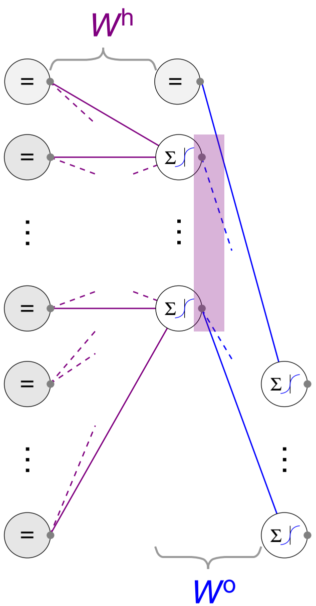


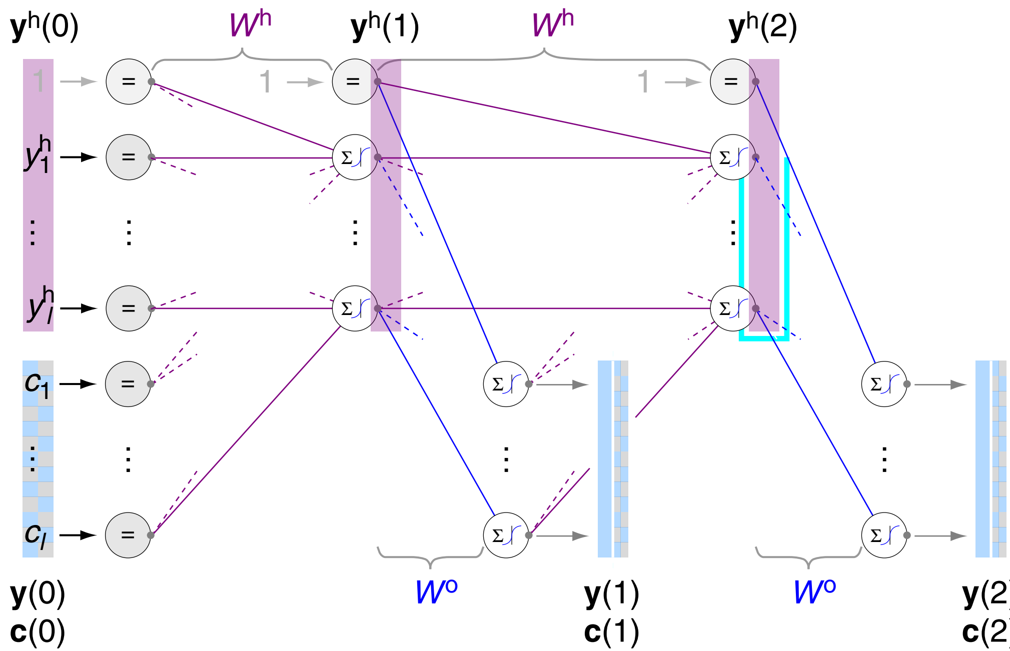
Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2





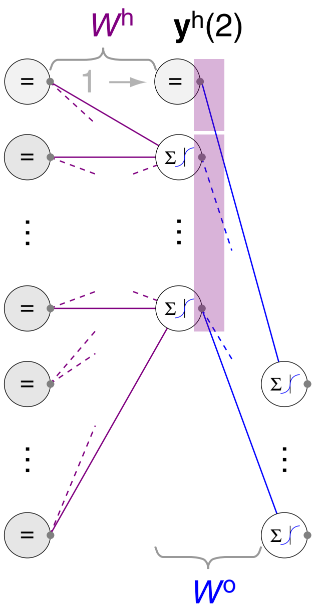
Output decoding over t.

Hidden and output layer at subsequent time steps.

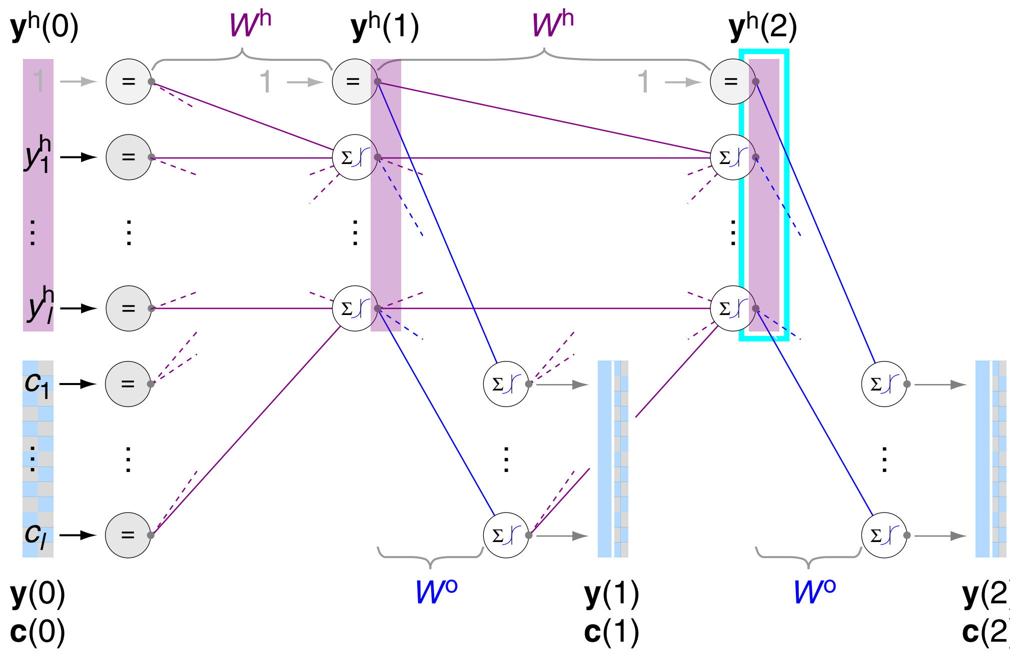
Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2



Output decoding over t.

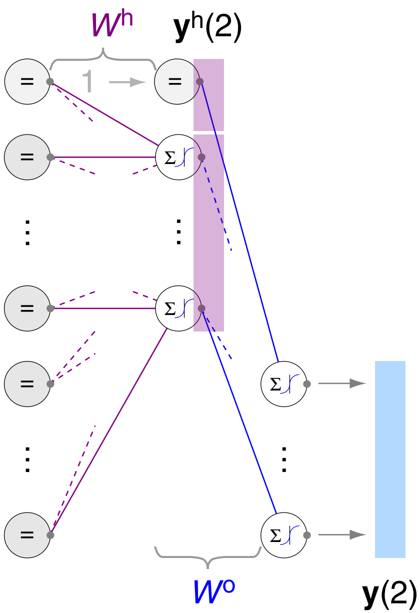


Hidden and output layer at subsequent time steps.

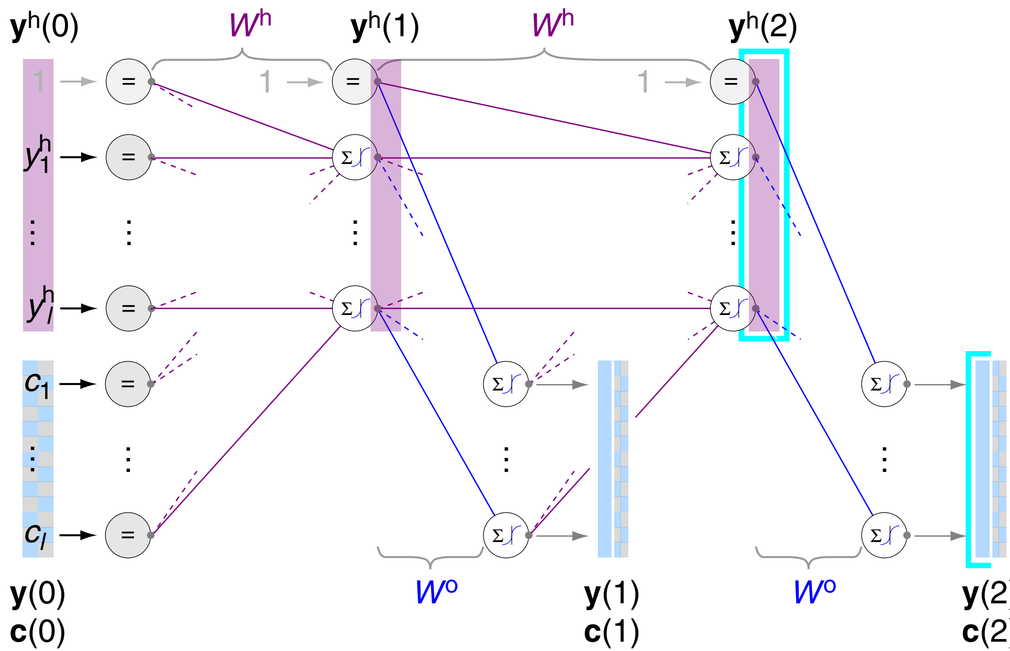
Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2



Output decoding over t.

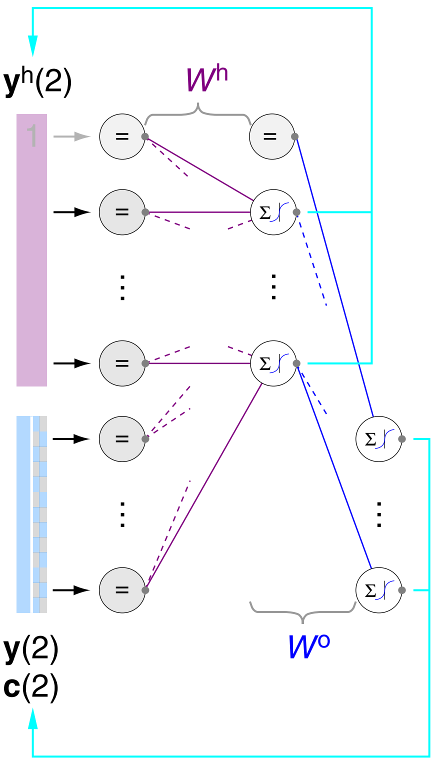


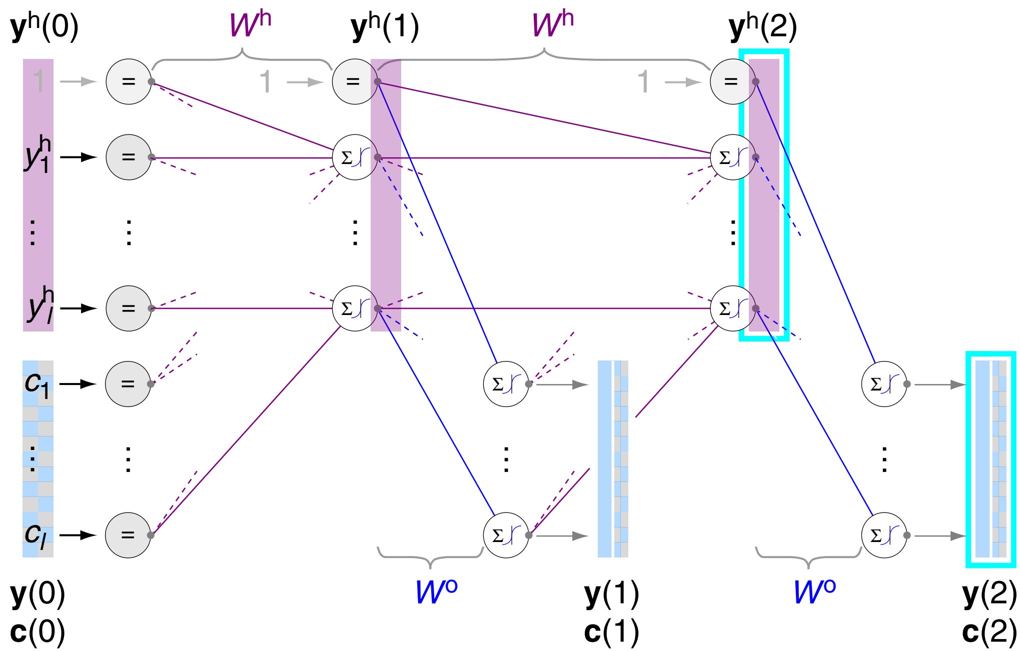
Hidden and output layer at subsequent time steps.

Recurrent Neural Networks

RNN Sequence Decoding (continued) [decoding overview]

t: 1 → 2





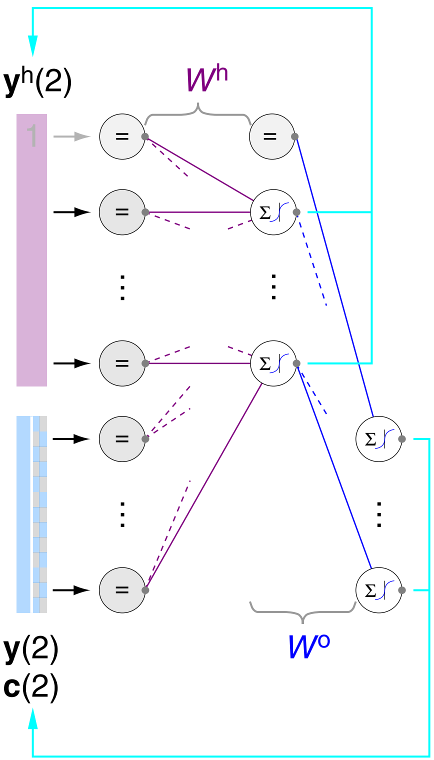
Output decoding over t.

Hidden and output layer at subsequent time steps.

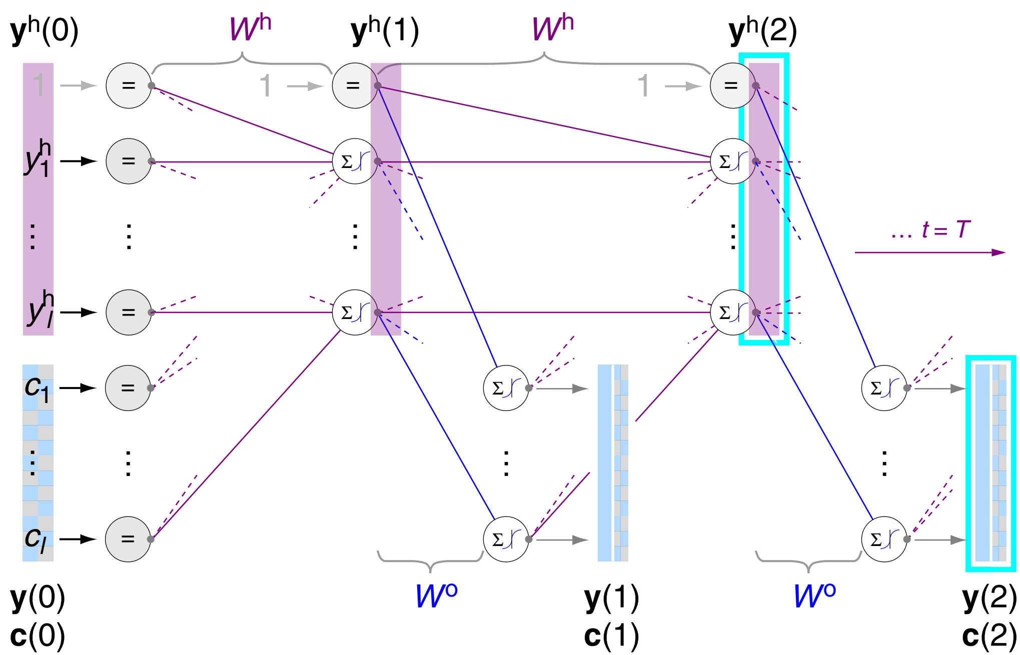
Recurrent Neural Networks

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

$\oplus \quad \rightarrow$ I love my cat.

$\oplus \quad \rightarrow$ Cats and dogs lap water.

$\ominus \quad \rightarrow$ It is raining cats and dogs.

$\ominus \quad \rightarrow$ Cats and dogs are not allowed.

$\ominus \quad \rightarrow$ 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

$\oplus \quad \rightarrow$ I love my cat.

$\oplus \quad \rightarrow$ Cats and dogs lap water.

$\ominus \quad \rightarrow$ It is raining cats and dogs.

$\ominus \quad \rightarrow$ Cats and dogs are not allowed.

$\ominus \quad \rightarrow$ 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: $[\,[\,[\,[\,[\mathbf{x},\ \mathbf{y}(0)\,],\ \mathbf{y}(1)],\ \mathbf{y}(2)],\ldots],\ \mathbf{y}(\tau-1)],\quad \mathbf{x} = \begin{pmatrix}\oplus\\ .\end{pmatrix}$

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

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation

$\oplus \quad \rightarrow$ I love my cat.

$\oplus \quad \rightarrow$ Cats and dogs lap water.

$\ominus \quad \rightarrow$ It is raining cats and dogs.

$\ominus \quad \rightarrow$ Cats and dogs are not allowed.

$\ominus \quad \rightarrow$ 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: $[\,[\,[\,[\,[\mathbf{x},\ \mathbf{y}(0)],\ \mathbf{y}(1)],\ \mathbf{y}(2)],\ldots],\ \mathbf{y}(\tau-1)],\quad \mathbf{x} = \begin{pmatrix}\oplus\\ .\end{pmatrix}$

Output: $[\mathbf{y}(1),\mathbf{y}(2),\mathbf{y}(3),\ldots,\mathbf{y}(\tau)],\quad \mathbf{y}(0)\equiv\mathbf{c}(0)\mathrel{\widehat{=}}\texttt{<start>},\quad \mathbf{y}(\tau)\mathrel{\widehat{=}}\mathbf{c}(5)\mathrel{\widehat{=}}\texttt{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation

$\oplus \quad \rightarrow$ I love my cat.

$\oplus \quad \rightarrow$ Cats and dogs lap water.

$\ominus \quad \rightarrow$ It is raining cats and dogs.

$\ominus \quad \rightarrow$ Cats and dogs are not allowed.

$\ominus \quad \rightarrow$ 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: $[\,[\,[\,[\,[\mathbf{x},\ \mathbf{y}(0)],\ \mathbf{y}(1)],\ \mathbf{y}(2)],\ldots],\ \mathbf{y}(\tau{-}1)],\qquad \mathbf{x} = \begin{pmatrix}\oplus \\ .\end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \ldots, \mathbf{y}(\tau)], \qquad \mathbf{y}(0) \equiv \mathbf{c}(0) \mathrel{\widehat{=}} \texttt{<start>}, \quad \mathbf{y}(\tau) \mathrel{\widehat{=}} \mathbf{c}(5) \mathrel{\widehat{=}} \texttt{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation

$\oplus \quad \rightarrow$ I love my cat.

$\oplus \quad \rightarrow$ Cats and dogs lap water.

$\ominus \quad \rightarrow$ It is raining cats and dogs.

$\ominus \quad \rightarrow$ Cats and dogs are not allowed.

$\ominus \quad \rightarrow$ 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: $[\,[\,[\,[\,[\mathbf{x},\ \mathbf{y}(0)],\ \mathbf{y}(1)],\ \mathbf{y}(2)], \ldots],\ \mathbf{y}(\tau{-}1)], \quad \mathbf{x} = \begin{pmatrix} \oplus \\ . \end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \ldots, \mathbf{y}(\tau)], \quad \mathbf{y}(0) \equiv \mathbf{c}(0) \mathrel{\widehat{=}} \texttt{<start>}, \quad \mathbf{y}(\tau) \mathrel{\widehat{=}} \mathbf{c}(5) \mathrel{\widehat{=}} \texttt{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation

$\oplus$ $\rightarrow$ I love my cat.

$\oplus$ $\rightarrow$ Cats and dogs lap water.

$\ominus$ $\rightarrow$ It is raining cats and dogs.

$\ominus$ $\rightarrow$ Cats and dogs are not allowed.

$\ominus$ $\rightarrow$ 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: $[\,[\,[\,[\,[\mathbf{x},\ \mathbf{y}(0)],\ \mathbf{y}(1)],\ \mathbf{y}(2)], \ldots],\ \mathbf{y}(\tau-1)],\qquad \mathbf{x} = \begin{pmatrix} \oplus \\ . \end{pmatrix}$

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \ldots, \mathbf{y}(\tau)],\qquad \mathbf{y}(0) \equiv \mathbf{c}(0) \mathrel{\widehat{=}} \texttt{<start>},\qquad \mathbf{y}(\tau) \mathrel{\widehat{=}} \mathbf{c}(5) \mathrel{\widehat{=}} \texttt{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence: Text Generation

$\oplus$ $\rightarrow$ I love my cat.

$\oplus$ $\rightarrow$ Cats and dogs lap water.

$\ominus$ $\rightarrow$ It is raining cats and dogs.

$\ominus$ $\rightarrow$ Cats and dogs are not allowed.

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

Output: $[\mathbf{y}(1), \mathbf{y}(2), \mathbf{y}(3), \ldots, \mathbf{y}(\tau)],\qquad \mathbf{y}(0) \equiv \mathbf{c}(0) \mathrel{\widehat{=}} \texttt{<start>},\qquad \mathbf{y}(\tau) \mathrel{\widehat{=}} \mathbf{c}(5) \mathrel{\widehat{=}} \texttt{<end>}$

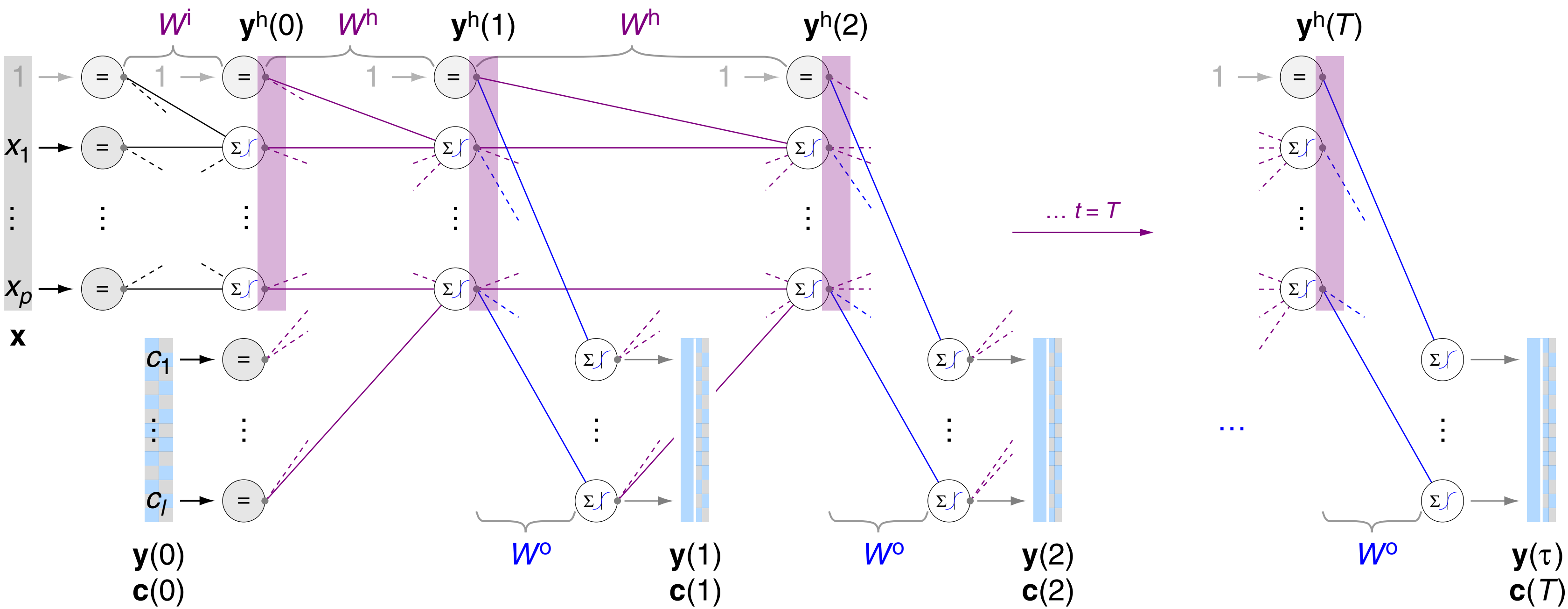
Target: $[\mathbf{c}(1), \ldots, \mathbf{c}(5)] = \left[\begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \end{pmatrix} \right]$

$\widehat{=}$ [`word_11, word_15, word_16, word_6, word_22`]

$\widehat{=}$ I love my cat

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

$\mathbf{x}$, $[\mathbf{y}(1), \ldots, \mathbf{y}(\tau{-}1)]$

Output:

$\mathbf{y}(t) = \boldsymbol{\sigma}_{\triangle}\left(W^{\mathrm{o}}\,\mathbf{y}^{\mathrm{h}}(t)\right), t = 1, \ldots, \tau$

Hidden:

$$\mathbf{y}^{\mathrm{h}}(0) = \boldsymbol{\sigma}\left(W^{\mathrm{i}}\,\mathbf{x}\right)$$

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

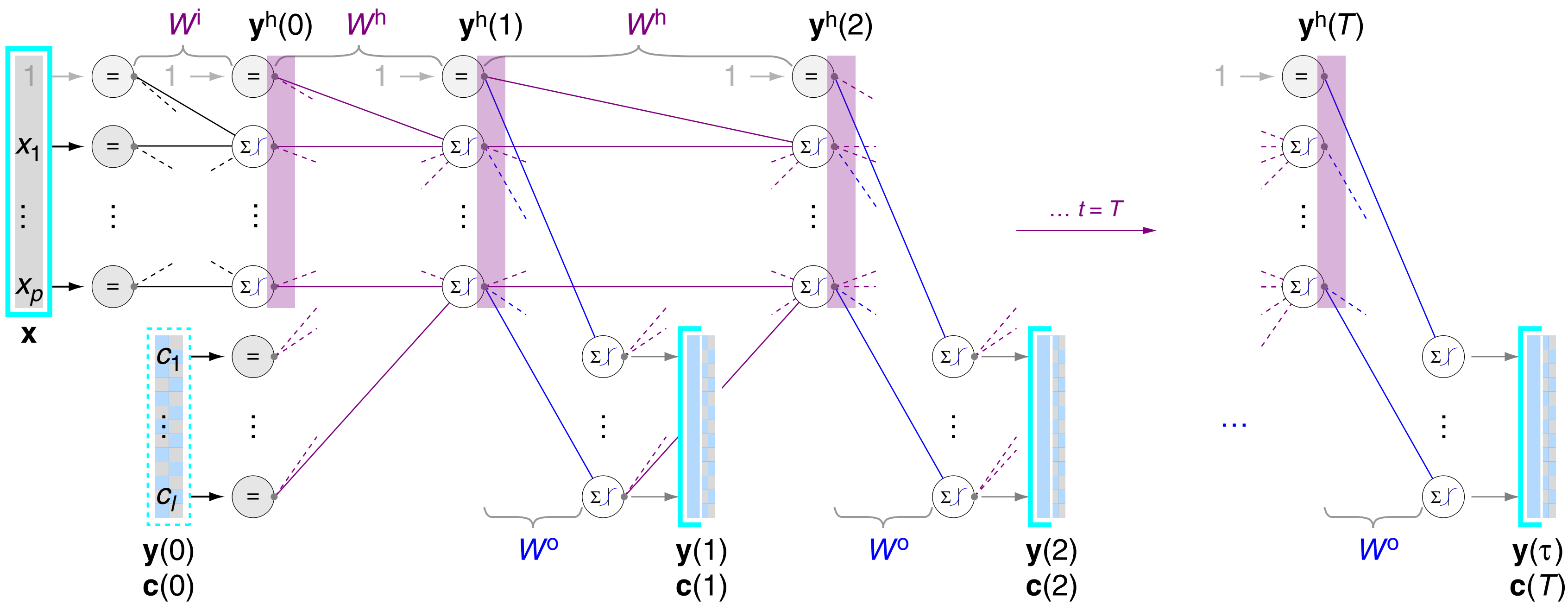
Target:

$[\mathbf{c}(1), \ldots, \mathbf{c}(T)]$

$\mathbf{c}(T) \mathrel{\hat{=}} \text{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

$\mathbf{x}$, $[\mathbf{y}(1), \ldots, \mathbf{y}(\tau-1)]$

Hidden:

$$\mathbf{y}^{\mathrm{h}}(0) = \boldsymbol{\sigma}\left(W^{\mathrm{i}}\,\mathbf{x}\right)$$

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

Target:

$[\mathbf{c}(1), \ldots, \mathbf{c}(T)]$

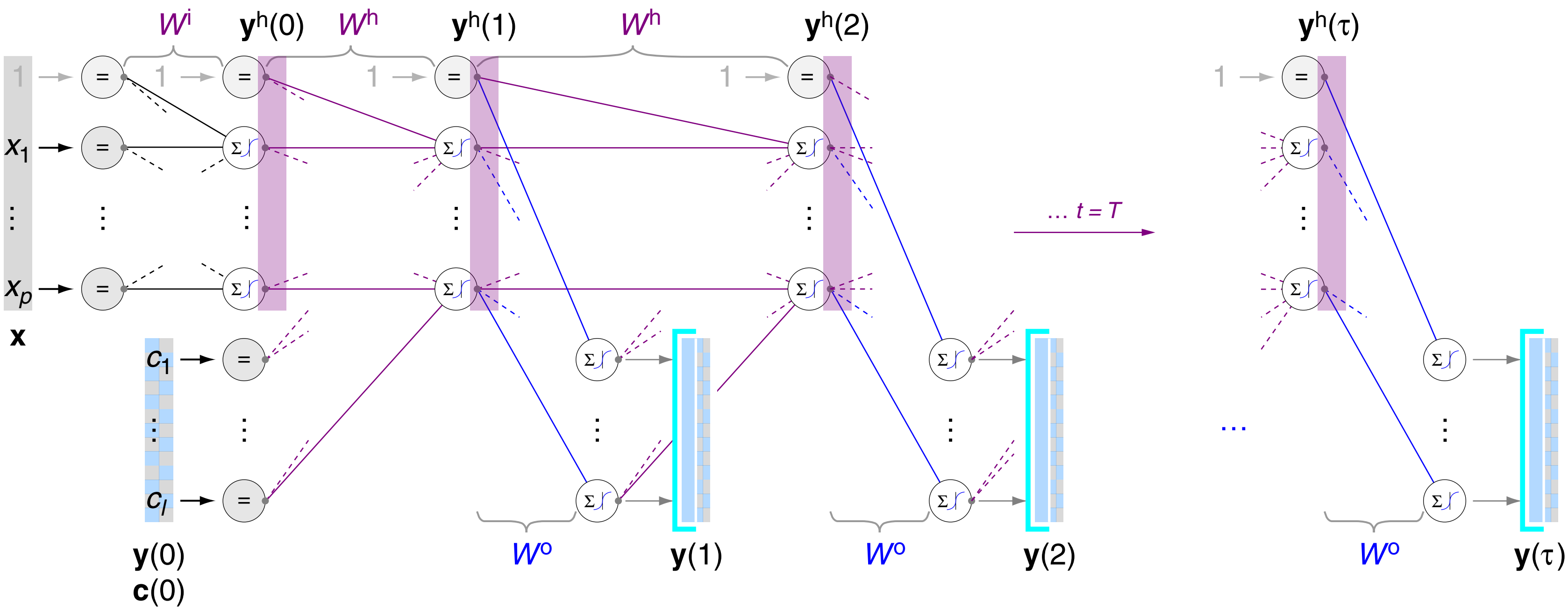
$\mathbf{c}(T) \mathrel{\widehat{=}} \text{<end>}$

Output:

$$\mathbf{y}(t) = \boldsymbol{\sigma}_{\Delta}\left(W^{\mathrm{o}}\,\mathbf{y}^{\mathrm{h}}(t)\right), t = 1, \ldots, \tau$$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

$\mathbf{x},\ [\mathbf{y}(1), \ldots, \mathbf{y}(\tau{-}1)]$

Hidden:

$\mathbf{y}^{\mathrm{h}}(0) = \boldsymbol{\sigma}\left(W^{\mathrm{i}}\,\mathbf{x}\right)$

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

Target:

$[\mathbf{c}(1), \ldots, \mathbf{c}(T)]$

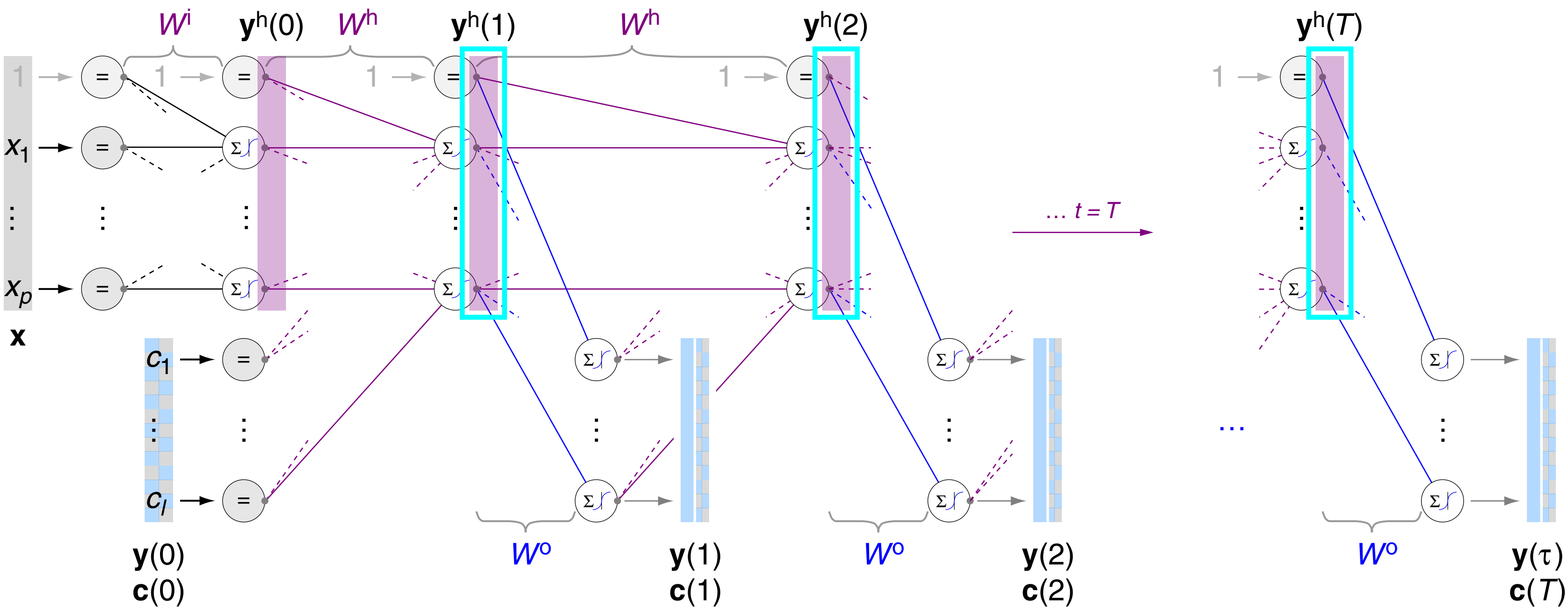
$\mathbf{c}(T) \mathrel{\widehat{=}}$ <end>

Output:

$\mathbf{y}(t) = \boldsymbol{\sigma}_{\Delta}\left(W^{\mathrm{o}}\,\mathbf{y}^{\mathrm{h}}(t)\right), t = 1, \ldots, \tau$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

$\mathbf{x}$, $[\mathbf{y}(1), \ldots, \mathbf{y}(\tau{-}1)]$

Output:

$\mathbf{y}(t) = \boldsymbol{\sigma}_\Delta\left(W^o\,\mathbf{y}^h(t)\right), t = 1, \ldots, \tau$

Hidden:

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

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

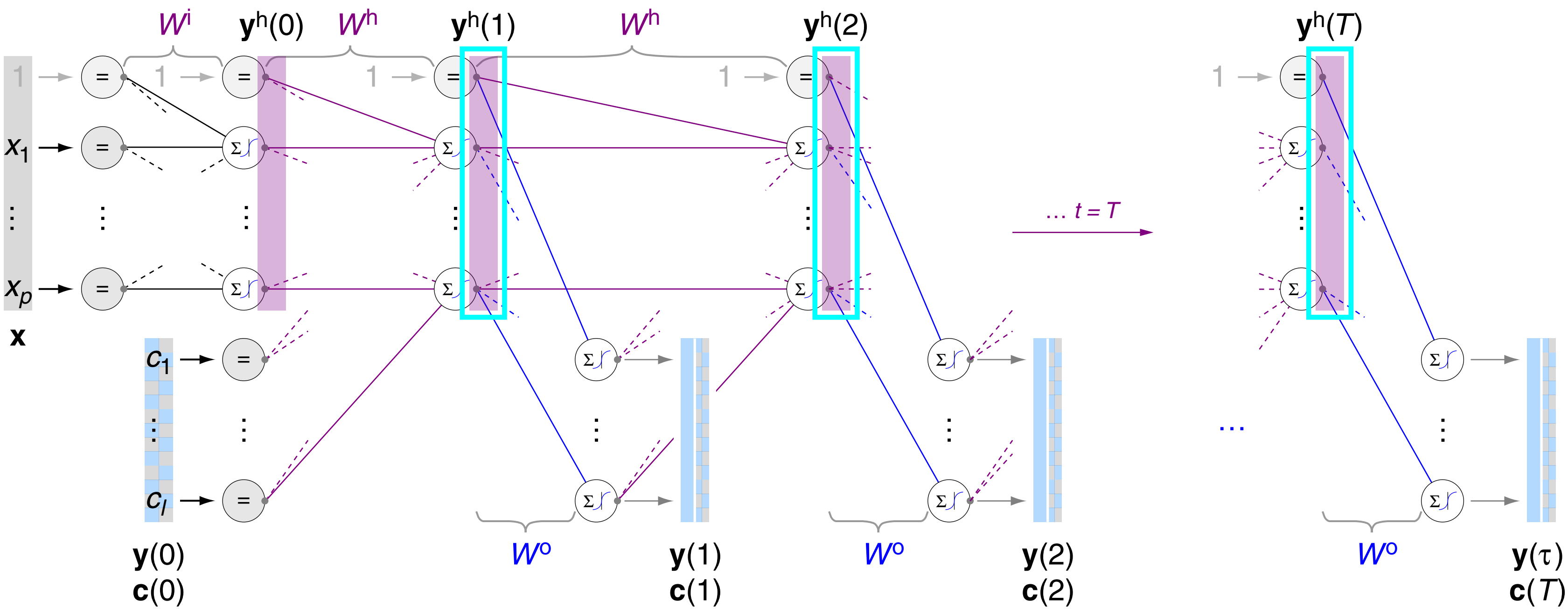
Target:

$[\mathbf{c}(1), \ldots, \mathbf{c}(T)]$

$\mathbf{c}(T) \mathrel{\widehat{=}}$ <end>

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

$\mathbf{x}$, $[\mathbf{y}(1), \ldots, \mathbf{y}(\tau{-}1)]$

Output:

$$\mathbf{y}(t) = \boldsymbol{\sigma}_\Delta\left(W^o\,\mathbf{y}^h(t)\right), t = 1, \ldots, \tau$$

Hidden:

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

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

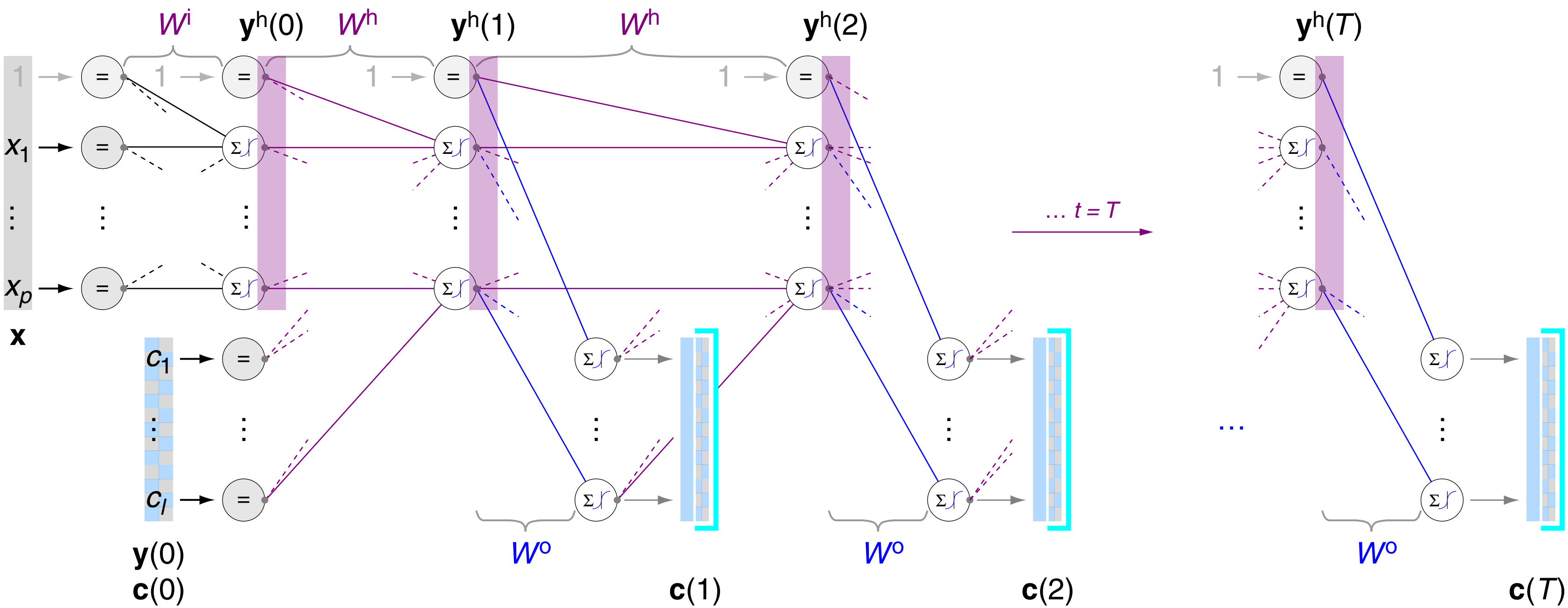
Target:

$[\mathbf{c}(1), \ldots, \mathbf{c}(T)]$

$\mathbf{c}(T) \widehat{=}$ <end>

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

$\mathbf{x}$, $[\mathbf{y}(1), \ldots, \mathbf{y}(\tau-1)]$

Output:

$\mathbf{y}(t) = \boldsymbol{\sigma}_{\Delta}\left(W^{\text{o}}\, \mathbf{y}^{\text{h}}(t)\right), t = 1, \ldots, \tau$

Hidden:

$$\mathbf{y}^{\text{h}}(0) = \boldsymbol{\sigma}\left(W^{\text{i}}\, \mathbf{x}\right)$$

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

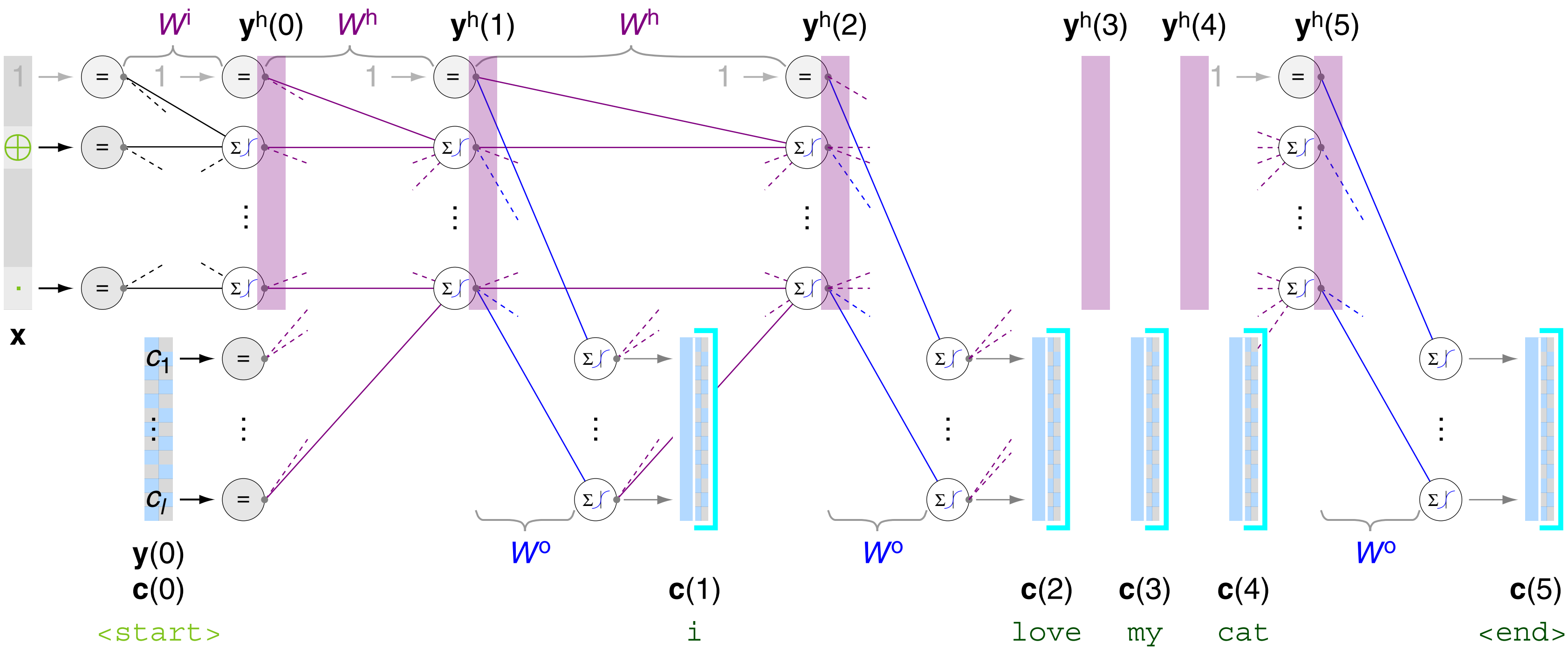
Target:

$[\mathbf{c}(1), \ldots, \mathbf{c}(T)]$

$\mathbf{c}(T) \mathrel{\widehat{=}}$ <end>

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

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

Output:

$$\mathbf{y}(t) = \boldsymbol{\sigma}_{\Delta}\left(W^{\mathsf{o}}\, \mathbf{y}^{\mathsf{h}}(t)\right), t = 1, \ldots, 5$$

Hidden:

$$\mathbf{y}^{\mathsf{h}}(0) = \boldsymbol{\sigma}\left(W^{\mathsf{i}}\, \mathbf{x}\right)$$

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

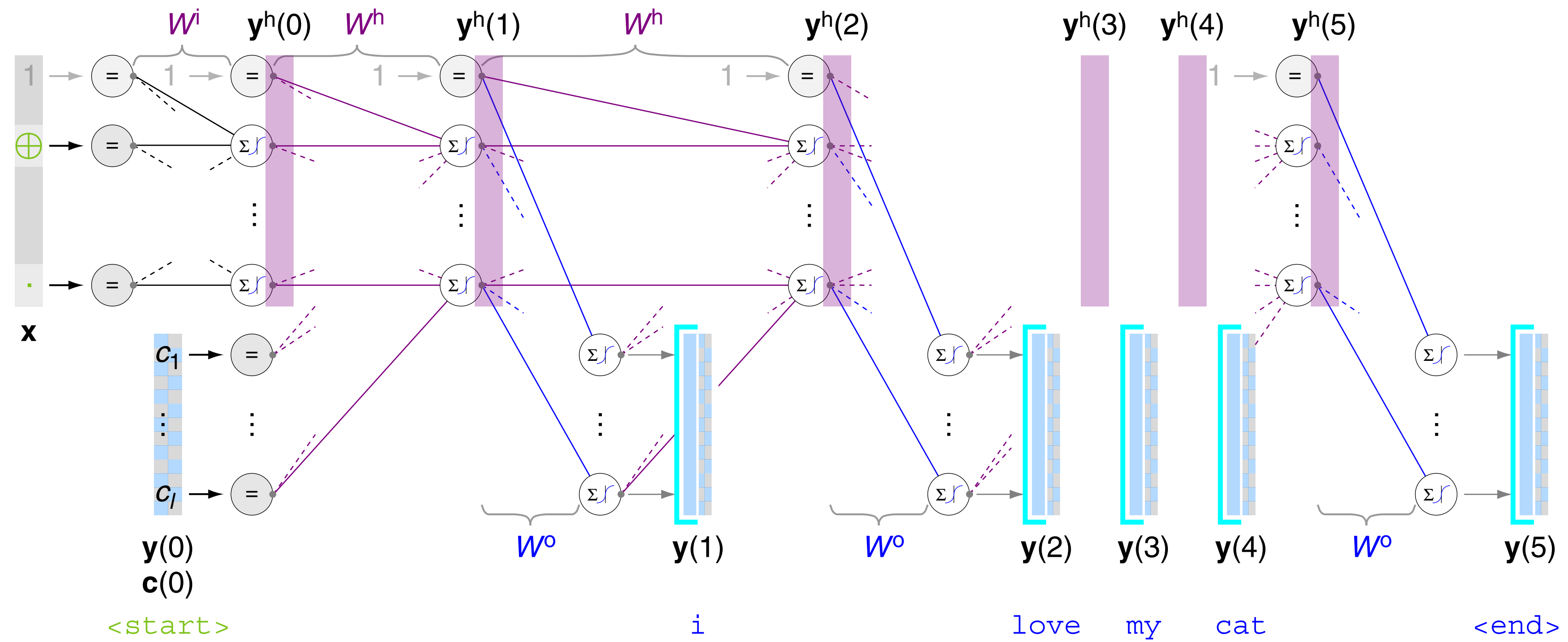
Target:

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

$\mathbf{c}(5) \mathrel{\widehat{=}} \texttt{<end>}$

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

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

Output:

$\mathbf{y}(t) = \boldsymbol{\sigma}_\Delta \left(W^{\mathrm{o}}\, \mathbf{y}^{\mathrm{h}}(t) \right), t = 1, \ldots, 5$

Hidden:

$\mathbf{y}^{\mathrm{h}}(0) = \boldsymbol{\sigma} \left(W^{\mathrm{i}}\, \mathbf{x} \right)$

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

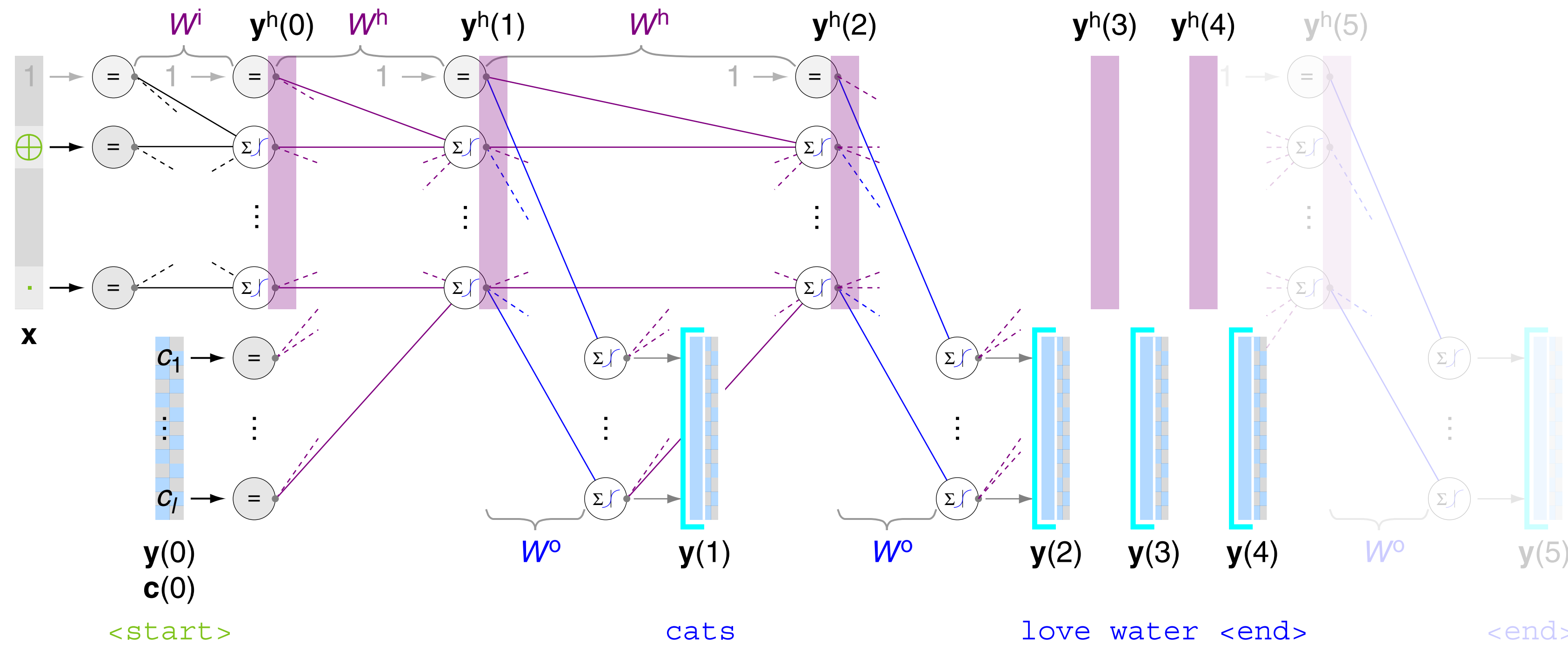
Target:

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

$\mathbf{c}(5) \mathrel{\widehat{=}}$ <end>

Recurrent Neural Networks

(S2) Class-to-Sequence Mapping with RNNs



Input:

$\mathbf{x}$, $[\mathbf{y}(1), \ldots, \mathbf{y}(3)]$

Output:

$\mathbf{y}(t) = \boldsymbol{\sigma}_\Delta\left(W^o\, \mathbf{y}^h(t)\right), t = 1, \ldots, 4$

Hidden:

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

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

Target:

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

$\mathbf{c}(5) \mathrel{\widehat{=}} \text{<end>}$

Remarks:

- We denote $\mathbf{y}(0)$ not as input since it is predefined and does not contain any "actual knowledge". In particular, $\mathbf{y}(0) \equiv \mathbf{c}(0) \mathrel{\widehat{=}} \texttt{<start>}$.

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

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

- At test time ("production mode") the calculation of $\mathbf{y}^h(t)$ has to consider the output $\mathbf{y}(t-1)$:

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

Recurrent Neural Networks

The IGD Algorithm for Class-to-Sequence Tasks [$\text{IGD}_{\text{seq2c}}$]

Algorithm:	$\text{IGD}_{\text{c2seq}}$	Incremental Gradient Descent for RNNs at class2seq tasks.
Input:	D	Multiset of examples $(\mathbf{x}, [\mathbf{c}(1), \ldots, \mathbf{c}(T)])$ with $\mathbf{x} \in \{0,1\}^p$, $\mathbf{c}(t) \in \mathbf{R}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\mathsf{i}}, W^{\mathsf{h}}, W^{\mathsf{o}}$	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 (x, [c(1), ..., c(T)]) ∈ D DO
5.                                   Model function evaluation.
6.                                   Calculation of residual vectors.
7a.
7b.                                  Calculation of derivative of the loss.
8.                                   Parameter update ≙ one gradient step down.
9.     ENDDO
10.  UNTIL(convergence(D, y(·), t_epoch))
11.  return(W^i, W^h, W^o)
```

Recurrent Neural Networks

The IGD Algorithm for Class-to-Sequence Tasks [$\text{IGD}_{\text{seq2c}}$]

Algorithm:	$\text{IGD}_{\text{c2seq}}$	Incremental Gradient Descent for RNNs at class2seq tasks.
Input:	D	Multiset of examples $(\mathbf{x}, [\mathbf{c}(1), \ldots, \mathbf{c}(T)])$ with $\mathbf{x} \in \{0,1\}^p$, $\mathbf{c}(t) \in \mathbf{R}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\mathsf{i}}, W^{\mathsf{h}}, W^{\mathsf{o}}$	Weights matrices. (= hypothesis)

1. *initialize_random_weights*$(W^{\mathsf{i}}, W^{\mathsf{h}}, W^{\mathsf{o}})$, $t_{\text{epoch}} = 0$
2. **REPEAT**
3. $\quad t_{\text{epoch}} = t_{\text{epoch}} + 1$
4. $\quad$ FOREACH $(\mathbf{x}, [\mathbf{c}(1), \ldots, \mathbf{c}(T)]) \in D$ DO
5. $\quad\quad \mathbf{y}^{\mathsf{h}}(0) = \boldsymbol{\sigma}\left(W^{\mathsf{i}}\,\mathbf{x}\right)$

$\quad\quad$ FOR $t = 1$ TO T DO // forward propagation

$$\mathbf{y}^{\mathsf{h}}(t) = \boldsymbol{\sigma}\left(W^{\mathsf{h}}\begin{pmatrix}\mathbf{y}^{\mathsf{h}}(t-1)\\ \mathbf{c}(t-1)\end{pmatrix}\right), \quad \mathbf{y}(t) = \boldsymbol{\sigma}_{\Delta}\left(W^{\mathsf{o}}\,\mathbf{y}^{\mathsf{h}}(t)\right)$$

$\quad\quad$ ENDDO

6. $\quad\quad [\boldsymbol{\delta}(1), \ldots, \boldsymbol{\delta}(T)] = [\mathbf{c}(1), \ldots, \mathbf{c}(T)] \ominus [\mathbf{y}(1), \ldots, \mathbf{y}(\tau)]$ // consider that T may $\neq \tau$

7a. $\quad\quad \ell(\mathbf{w}) = \sum_t l(\boldsymbol{\delta}(t)) + \frac{\lambda}{n} R(\mathbf{w})$, $\nabla\ell(\mathbf{w}) = \text{autodiff}(\ell(), \mathbf{w})$ // backprop.(7a+7b)

7b. $\quad\quad \Delta W^{\mathsf{i}} = \eta \cdot \nabla^{\mathsf{i}}\ell(\mathbf{w})$, $\Delta W^{\mathsf{h}} = \eta \cdot \nabla^{\mathsf{h}}\ell(\mathbf{w})$, $\Delta W^{\mathsf{o}} = \eta \cdot \nabla^{\mathsf{o}}\ell(\mathbf{w})$

8. $\quad\quad W^{\mathsf{i}} = W^{\mathsf{i}} + \Delta W^{\mathsf{i}}$, $W^{\mathsf{h}} = W^{\mathsf{h}} + \Delta W^{\mathsf{h}}$, $W^{\mathsf{o}} = W^{\mathsf{o}} + \Delta W^{\mathsf{o}}$
9. $\quad$ ENDDO
10. **UNTIL**(*convergence*$(D, \mathbf{y}(\cdot), t_{\text{epoch}})$)
11. *return*$(W^{\mathsf{i}}, W^{\mathsf{h}}, W^{\mathsf{o}})$

Remarks:

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