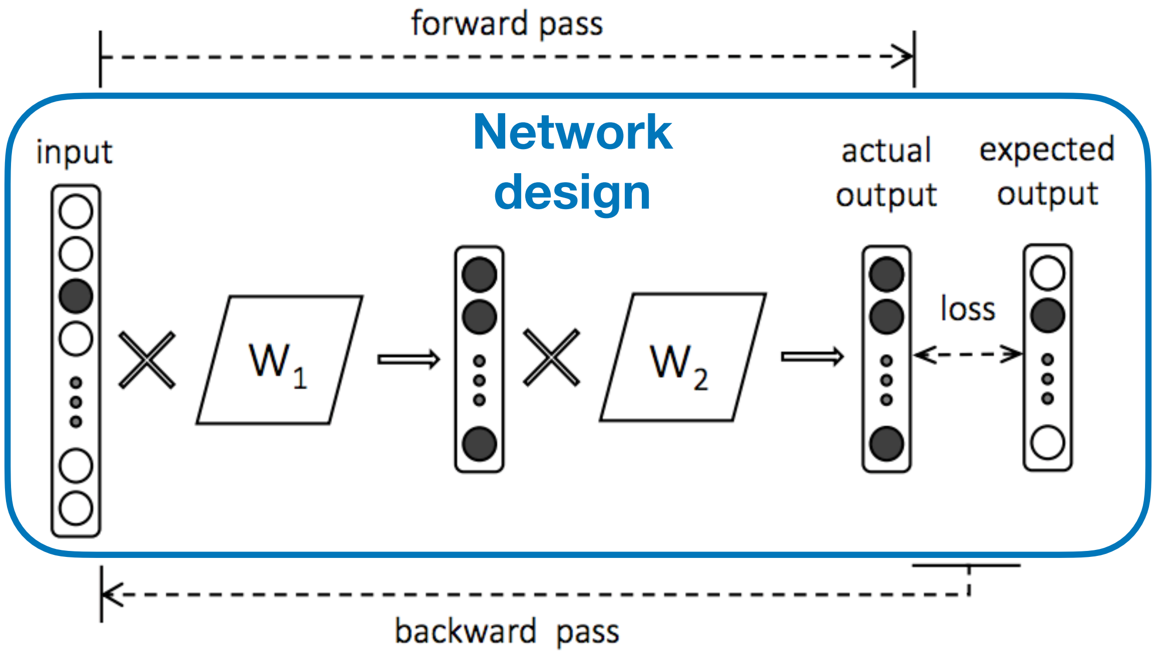
Advanced Topics in Deep Learning: Recurrent Neural Networks

Anders Søgaard

Previous Lectures

Forward propagation

Input representation



Error estimation

Parameter updates

Figure: Mitra and Craswell (2019)

Week	Lecturer	Subject	Literature	Assignment
1	Stefan	Introduction to Neural Networks.	d2l 2.1-2.5, 2.7, 11.5.1, **slides**	
2	Stefan	CNNs; FCNs; U-Nets. Data augmentation; invariance; regularization e.g. dropout	d2l 6, 7, **slides**	Assignment 1 (May 10)
3	Anders/Phillip	RNNs (May 9) and Transformers (May 11)	Transformers: Vaswani et al. (2017)	Assignment 2 (May 17)
4	Phillip/Anders	May 16-18: Representation Learning	Autoencoders (1/2): tba Self-supervised learning: blog post Contrastive learning: Dor et al. (2018)	
5	Anders	May 23-25: Adversarial Learning	Autoencoders (2/2): Chandar et al. (2011) GANs: Lample et al. (2018) Applications: Goodfellow et al. (2015)	Assignment 3 *[MC on Representation Learning/1p Report on Adversarial Learning]* (May 31)
6	Anders	May 30-June 1: Interpretability	Literature: Søgaard (2022)	
7	Anders	June 8: Interpretability (Note: June 6 off)	Literature: Feng and Boyd-Graber (2018) ; Jiang and Senge (2021)	
8	Anders	June 13-15: Best Practices	Literature: Dodge et al. (2019) and Raji et al. (2021)	Assignment 4 *[MC on Interpretability; 1p Report on Best Practices]* (June 21)

This Lecture

- Word Embeddings (Recap)
- Recurrent Neural Networks
- RNNs for Embeddings

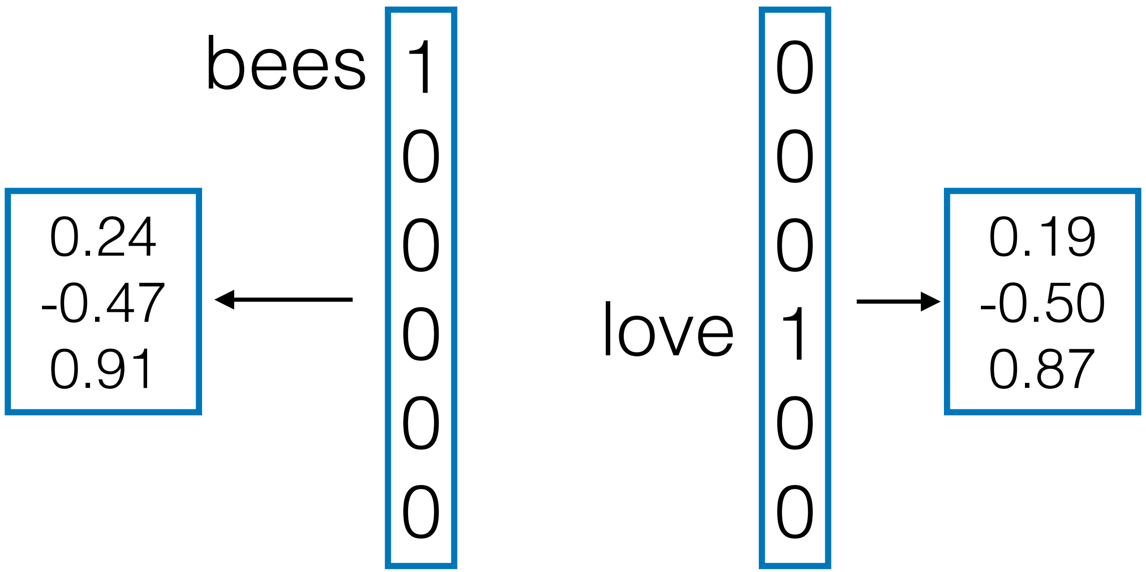
Word Embeddings

Representing Text

- One-hot vector over a vocabulary of V word types $w \in \mathcal{R}^{|V|}$
- Continuous dense vector with E dimensions (word embedding)

$$x \in \mathcal{R}^{|E|}$$

- One-hot -to- word embedding is a lookup function

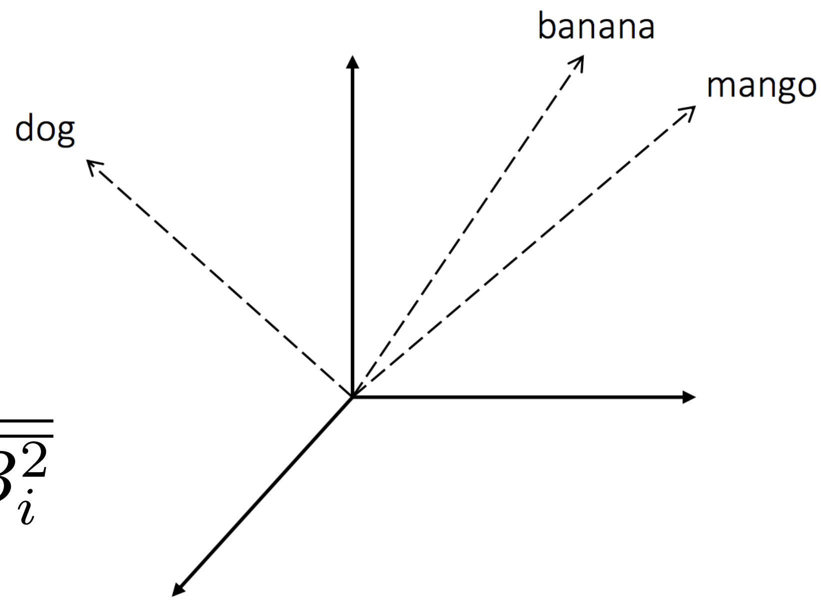


Why learn Embeddings?

- Overcome sparsity of one-hot vectors

- Cosine similarity will tell us the **angle formed between vectors A and B,** where A and B are representations of the different terms

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$



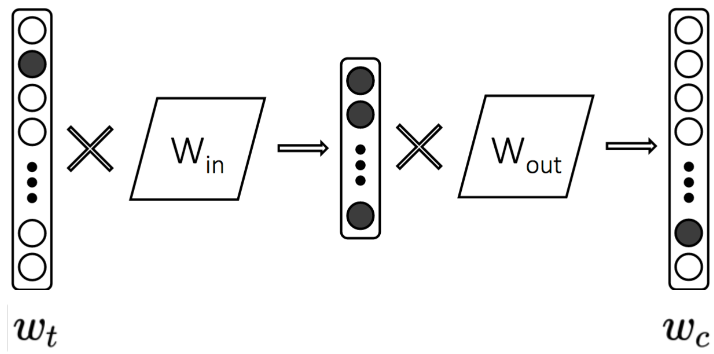
Type-based Skip-gram Embeddings

- Skip-gram embeddings are learned in the context of **predicting the surrounding words** $w_{i-|C|}$, $w_{i+|C|}$, given the target word w_t

$$\mathcal{L} = \sum_{i=1}^{T} \sum_{c \in \mathcal{C}_t} \log \mathrm{p}(w_c | w_t)$$

Over all surrounding windows in the corpus

Estimate probability of predicting each word in the surrounding window



$$\log \mathrm{p}(w_c | w_t) := \mathrm{softmax}(W_{out} W_{in} w_t)$$

- Softmax doesn't scale so you can use noise-contrastive estimation

$$\sum_{t=1}^{T} \left[\sum_{c \in \mathcal{C}_t} \ell(s(w_t, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t, n)) \right] \qquad s(w_t, w_c) = \mathbf{u}_{w_t}^{\top} \mathbf{v}_{w_c}$$

Mikolov et al. 2013; Figure Mitra and Craswell 2019

Type-based FastText Embeddings

- The main shortcoming of type-based embeddings is **how should we deal with out-of-vocabulary tokens**?

- FastText proposes to split words into all sequences of 3—6 characters and learn embeddings $\mathbf{z}_g$ for each sequence

`subwords`(**where**)

<wh	**<whe**	**<wher**	**<where**
whe	**wher**	**where**	**where>**
her	**here**	**here>**	
ere	**ere>**		
re>			

`FastText`

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c$$

`Type-based`

$$s(w_t, w_c) = \mathbf{u}_{w_t}^\top \mathbf{v}_{w_c}$$

Bojanowski et al. 2016

Recurrent Neural Networks

Recurrent Neural Networks

- The **order of the words** in a sentence is not random. The order often **conforms to rules of syntax and semantics**.

The girl saw the dog with a telescope

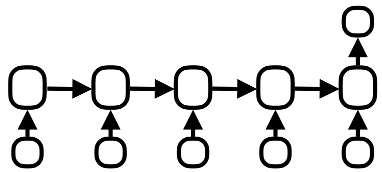
Semantic nonsense → **The telescope saw the girl with a dog**

The dog with the a girl telescope saw ← Syntactic gibberish

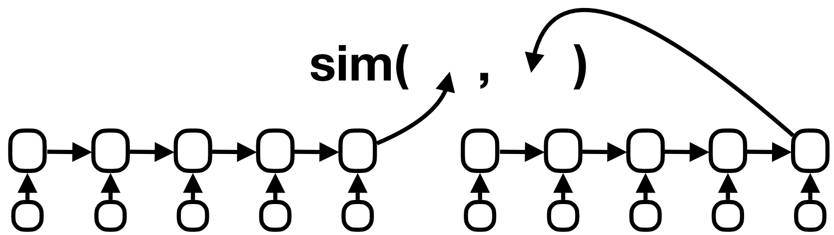
- An averaged embedding representation of a sentence **does not capture the difference in these sentences**.

Applications of RNNs

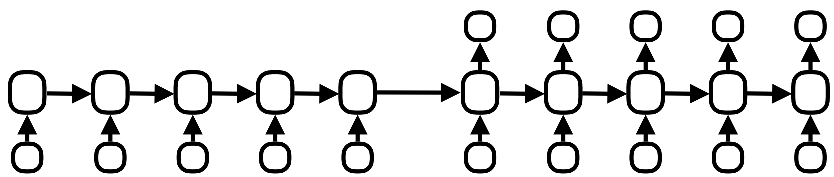
- **Classification**
  - Text Categorization



- **Sequence encoding**
  - Sentence similarity



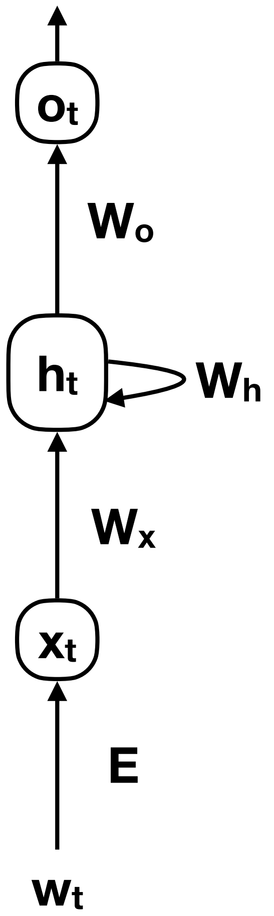
- **Autoregressive modelling**
  - Machine translation



Simple RNN: Definition

w: Explaining the Simple RNN model
t:

$$\mathbf{w_{t+1}} = \mathbf{softmax(o_t)}$$



Output vector

$$\mathbf{o_t = W_o h_t + b_o}$$

Hidden state

$$\mathbf{h_t = f(W_x x_t + W_h h_{t-1} + b_h)}$$

Embedded word

$$\mathbf{x_t = E[w_t]}$$

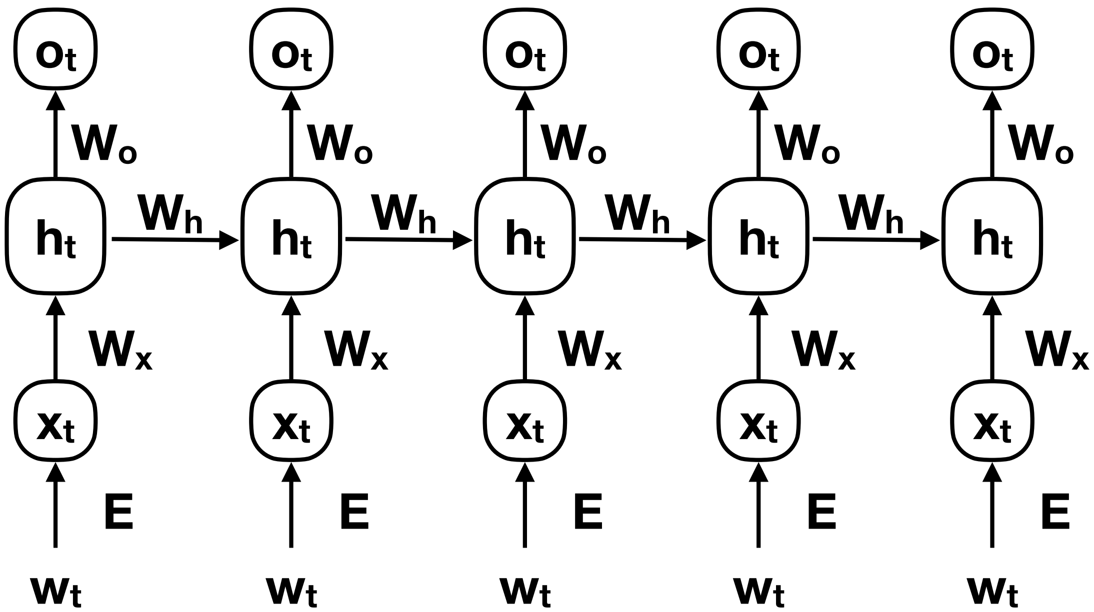
1-hot vector

$$\mathbf{w_t = [0\ 0\ 0\ 0\ 1]}$$

Elman. (1990). "Finding Structure in Time"

Unrolled Simple RNN

- This is a feed-forward neural network that has depth-in-time.



```python
class SingleRNN(nn.Module):
    def __init__(self, n_inputs, n_neurons):
        super(SingleRNN, self).__init__()

        self.Wx = torch.randn(n_inputs, n_neurons) # 4 X 1
        self.Wy = torch.randn(n_neurons, n_neurons) # 1 X 1

        self.b = torch.zeros(1, n_neurons) # 1 X 4

    def forward(self, X0, X1):
        self.Y0 = torch.tanh(torch.mm(X0, self.Wx) + self.b) # 4 X 1

        self.Y1 = torch.tanh(torch.mm(self.Y0, self.Wy) +
                            torch.mm(X1, self.Wx) + self.b) # 4 X 1

        return self.Y0, self.Y1
```

Exercise: What is this?

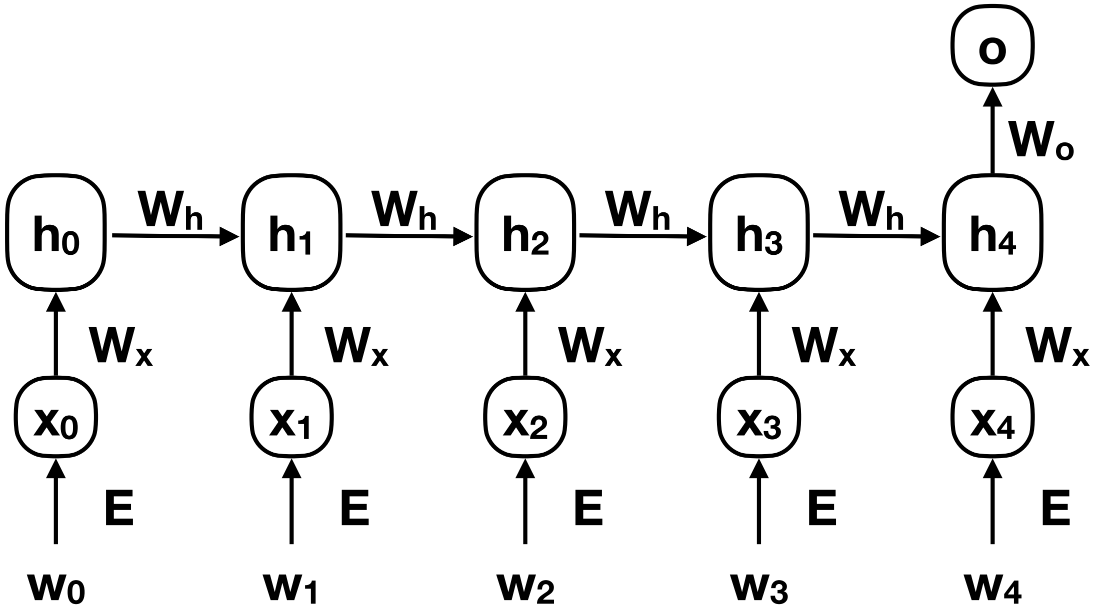
Padding data in RNNs

X

The	small	car		
The	car			
The	car	with	red	wheels
My	favourite	car		

Padded X

The	small	car	<p>	<p>
The	car	<p>	<p>	<p>
The	car	with	red	wheels
My	favourite	car	<p>	<p>



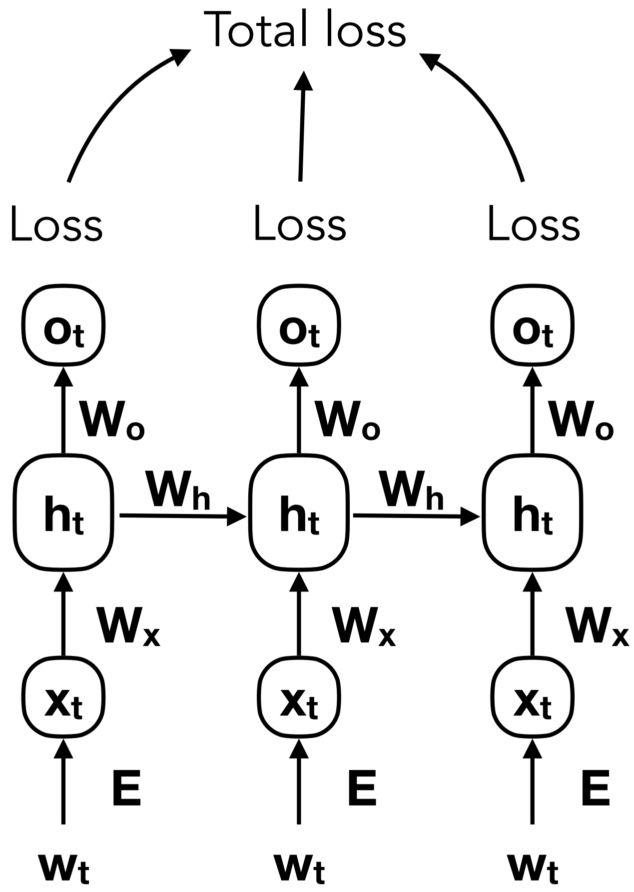
Loss masking in RNNs

Padded X

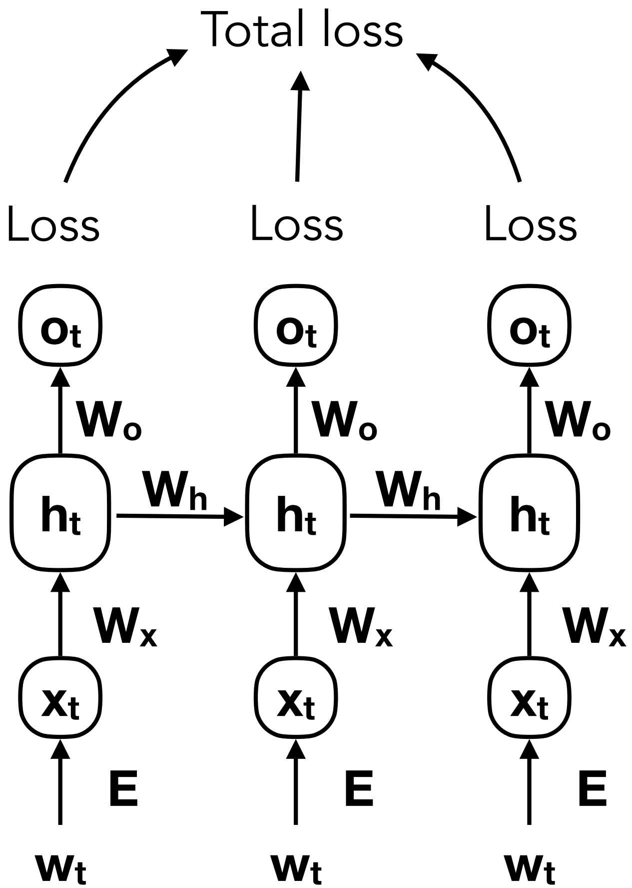
The	small	car	<p>	<p>
The	car	<p>	<p>	<p>
The	car	with	red	wheels
My	favourite	car	<p>	<p>

Mask

1	1	1	0	0
1	1	0	0	0
1	1	1	1	1
1	1	1	0	0



Training Autoregressive RNNs

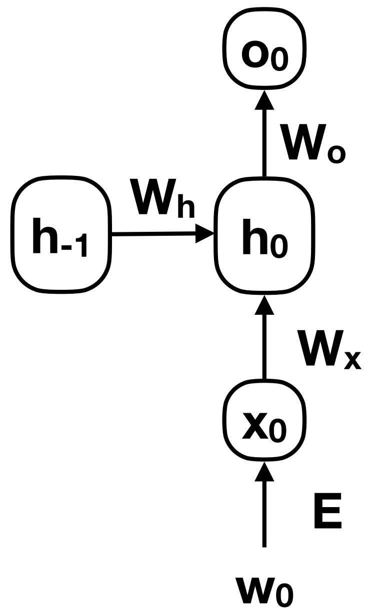


- Objective

$$-\sum_{i=1}^{n}\sum_{t=1}^{T}\log p(x_t^i|x_{<t}^i)$$

Simple RNN: Extra Notes

- What about the initial hidden state?

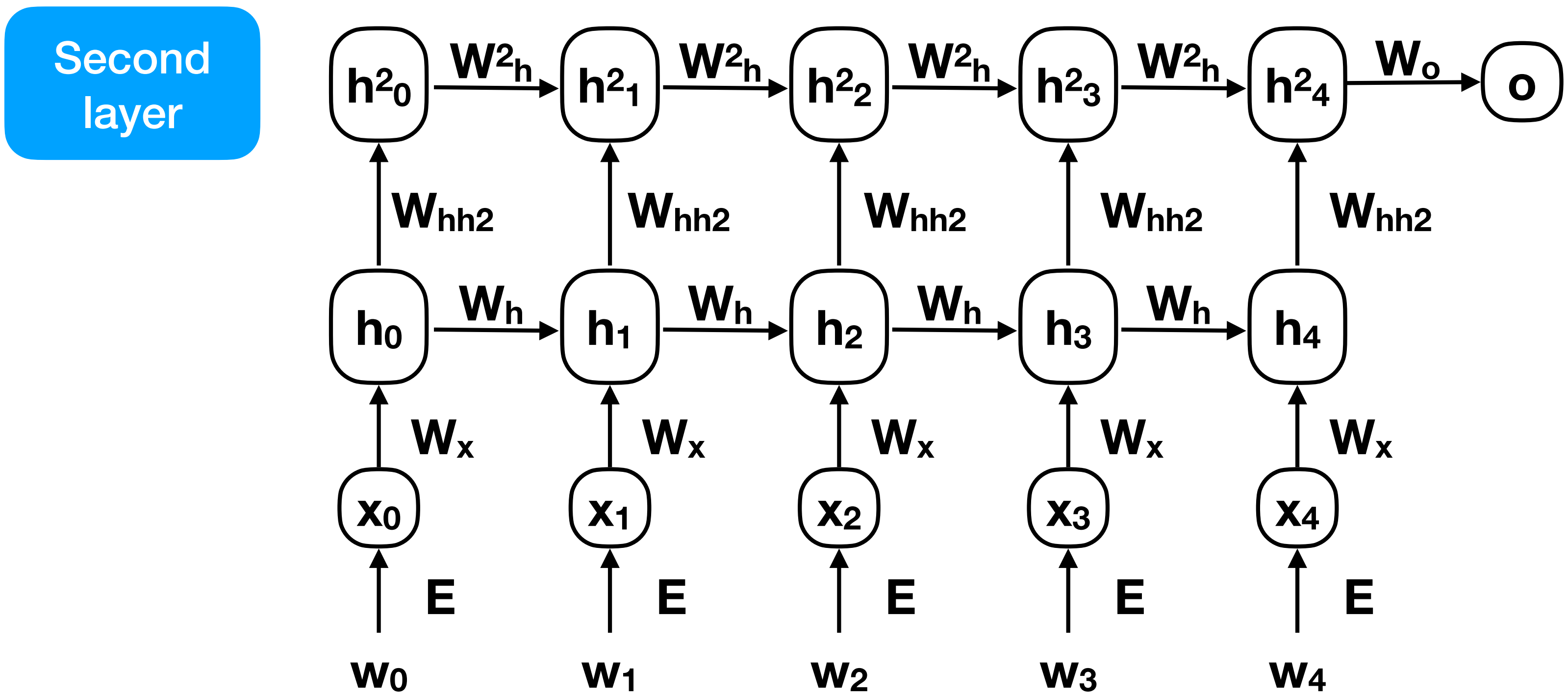


- Why are we reusing the transition vectors?

  $\mathbf{W_h}$ $\mathbf{W_x}$ $\mathbf{W_o}$

Multi-layer RNN

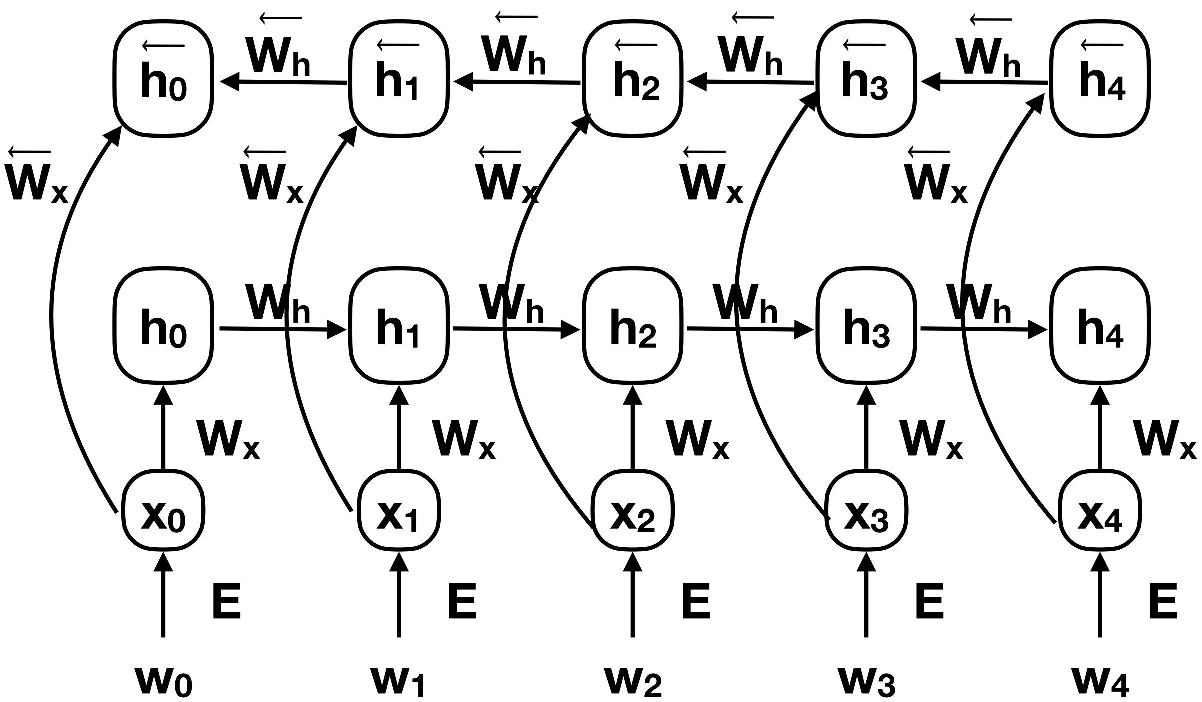
- Recurrent Neural Networks can also have layer-wise depth



$$\mathbf{h_t^2} = \mathbf{f}(W_{hh2}h_t + W_h^2 h_{t-1}^2 + b_h^2)$$

Bidirectional Networks

- Alternatively, we can encode the sequence in reverse.
- This “bidirectional” model provides a richer representation.



Backward

$$\overleftarrow{\mathbf{h}}_t = f(\overleftarrow{\mathbf{h}}_{t-1}, \mathbf{x}_t)$$

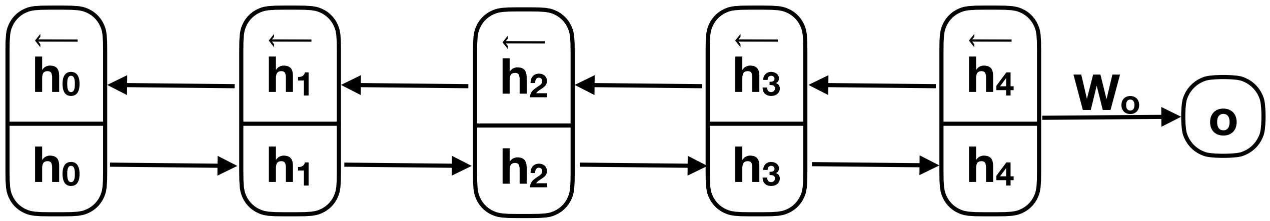
Forward

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

Schuster and Paliwal 1997

Bi-Directional Networks

- Now we can train our classifiers on a concatenation of the forward and backward representations of the sequence



$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \qquad \overleftarrow{\mathbf{h}}_t = f(\overleftarrow{\mathbf{h}_{t-1}}, \mathbf{x}_t)$$

$$\mathbf{h}_t = [\mathbf{h}_t \oplus \overleftarrow{\mathbf{h}}_t]$$

Schuster and Paliwal 1997

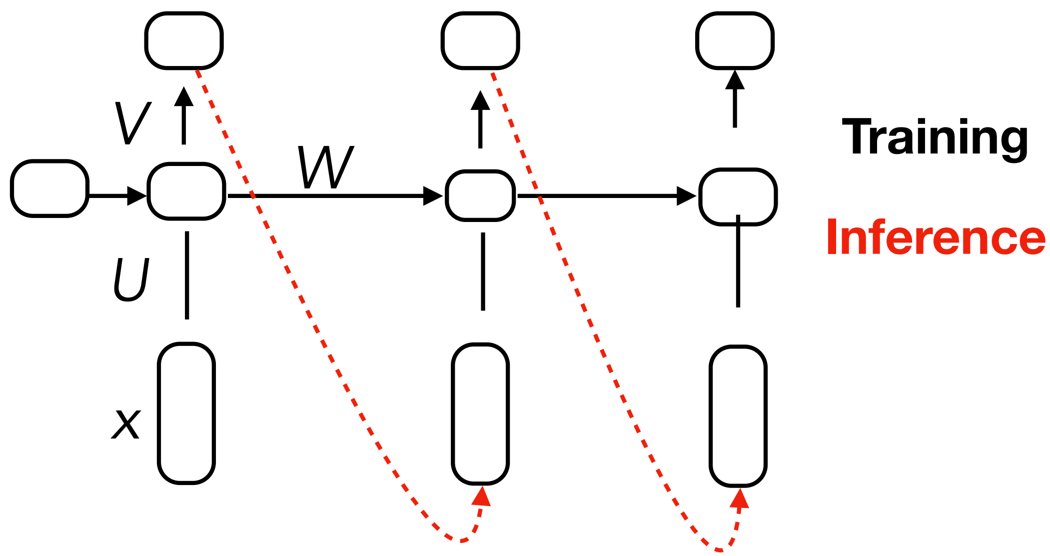
RNNs for Embeddings

Autoregressive Recurrent Neural Networks

- Elman-style model (Cognitive Science, 1990)
  - LSTM (Hochreiter and Schmidhuber, 1997)
  - Gated Recurrent Unit (Cho et al. 2014)

$$o_t = \mathrm{softmax}(V s_t)$$

$$s_t = f(U x_t + W s_{t-1})$$



- Create a **contextualized representation** of a word in the sequence of hidden states s_t

Contextual Embeddings from Language Models

- Represent each token using multi-layer bidirectional LSTMs.

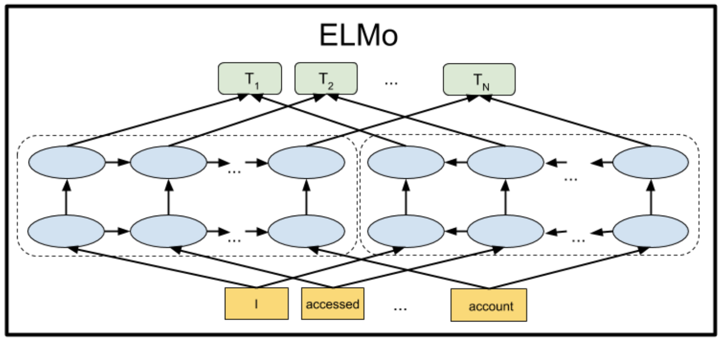
- Final representation is a **weighted sum of the representations** at the different layers in the model

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

Sum over each Layer

Softmax-normalized task weights

Feature vector for token *k* at layer *j*



Peters et al. 2018;
Figure: https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html