Language Technologies Institute

Carnegie Mellon University

Algorithms for NLP

CS 11-711 · Fall 2020

Lecture 6: Neural language modeling & contextualized word representations

Emma Strubell

Announcements

- Recitation this Friday is a P1 Q&A with Han, with two optional quiz questions to work on even if you don’t have P1 questions, for participation credit.
- Some more readings posted for today’s lecture.

Recap

Language models

- Goal: compute the probability of a sentence (or sequence of words):

$$P(\mathbf{w}) = P(w_1, w_2, w_3, ..., w_n)$$

- Related task: probability of the next word:

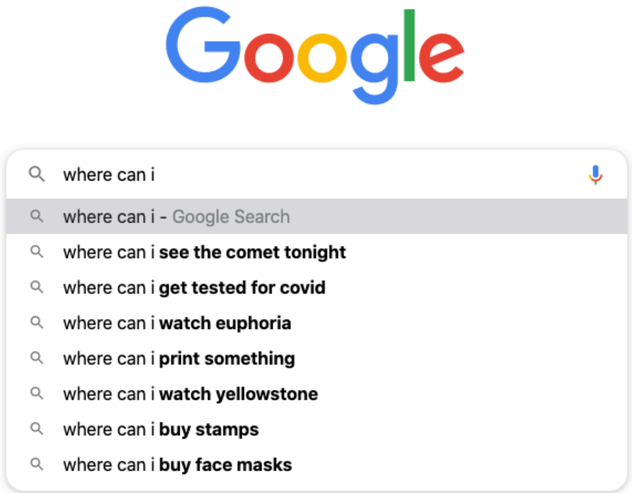
$$P(w_5 \mid w_4, w_3, w_2, w_1)$$

- A model that computes either of these is called a **language model** (or **LM**).

Recap

Why language models?

- Machine translation:
  P(*high winds tonight*) > P(*large winds tonight*)
- Spelling correction:
  P(*five minutes late*) > P(*five minuets late*)
- Speech recognition:
  P(*I saw a van*) > P(*eyes awe of an*)



Recap

Why language models?

- Machine translation:
  P(*high winds tonight*) > P(*large winds tonight*)
- Spelling correction:
  P(*five minutes late*) > P(*five minuets late*)
- Speech recognition:
  P(*I saw a van*) > P(*eyes awe of an*)
- Since ~2018, state-of-the-art dense vector representations of text.

Recap

n-gram language model challenges

I'll have a _____

I'd like to order the _____

Give me the _____

I'll take one _____

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

Recap

n-gram language model challenges

- Bias/variance trade-off

I'll have a _____ Give me the _____

I'd like to order the _____ I'll take one _____

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

Recap

n-gram language model challenges

- Bias/variance trade-off
- Generalization

I'll have a _____ Give me the _____

I'd like to order the _____ I'll take one _____

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

Recap

n-gram language model challenges

- Bias/variance trade-off
- Generalization

  I'll have a _____ Give me the _____

  I'd like to order the _____ I'll take one _____
- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

Recap

n-gram language model challenges

- Bias/variance trade-off
- Generalization

I'll have a _____ Give me the _____

I'd like to order the _____ I'll take one _____

- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**.

Recap

n-gram language model challenges

- Bias/variance trade-off
- Generalization

  I'll have a _____ Give me the _____

  I'd like to order the _____ I'll take one _____
- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**.
  - Better empirical performance.

Recap

n-gram language model challenges

- Bias/variance trade-off
- Generalization

  I'll have a _____ Give me the _____

  I'd like to order the _____ I'll take one _____
- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**.
  - Better empirical performance.
  - Provide state-of-the-art dense word representations.

Recap

n-gram language model challenges

- Bias/variance trade-off
- Generalization

  I'll have a _____ Give me the _____

  I'd like to order the _____ I'll take one _____
- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**.
  - Better empirical performance.
  - Provide state-of-the-art dense word representations.
  - But: a lot more computation. Slower to train and infer.

Neural language models

…	breathless	runners	approached	the
	w_{i-4}	w_{i-3}	w_{i-2}	w_{i-1}

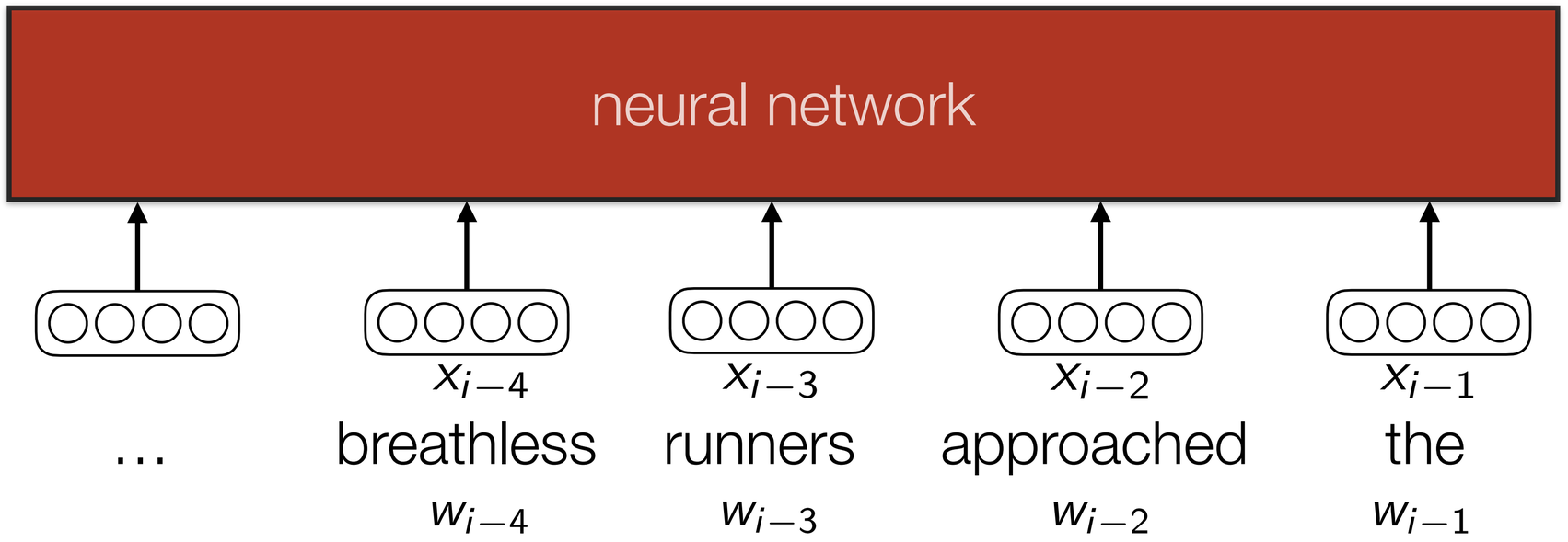
Neural language models

- Don’t count, predict!

…	breathless	runners	approached	the
	w_{i-4}	w_{i-3}	w_{i-2}	w_{i-1}

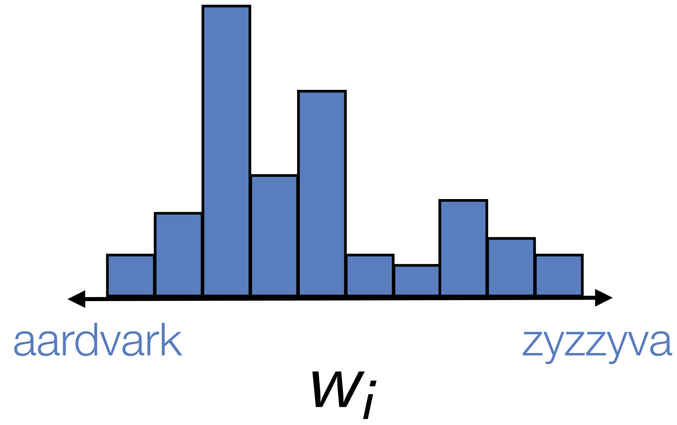
Neural language models

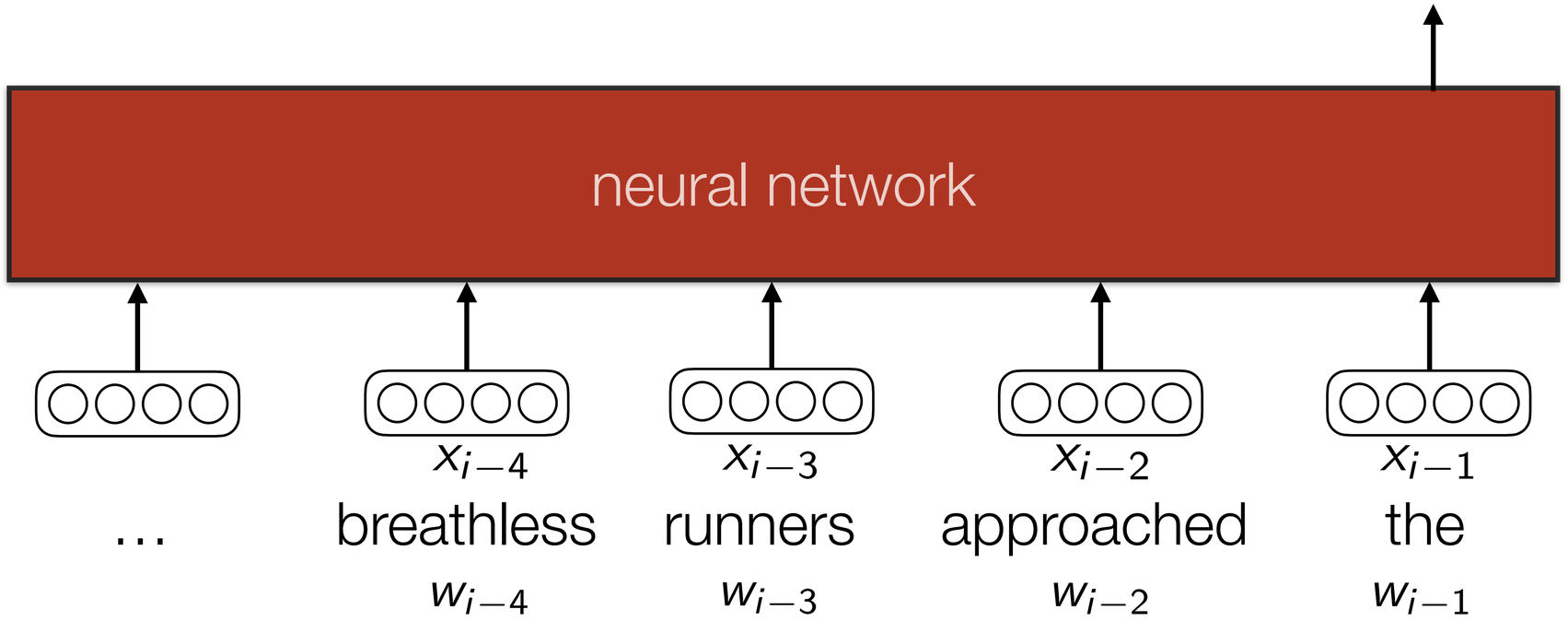
- Don’t count, predict!
- Input: word embeddings x_1, x_2, …, x_N



Neural language models

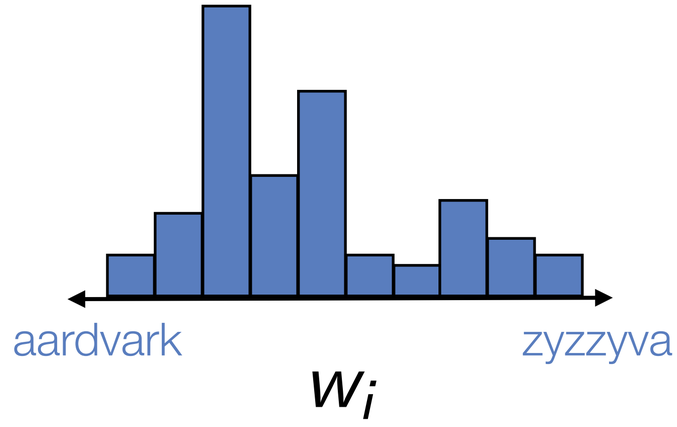
- Don’t count, predict!
- Input: word embeddings $x_1, x_2, \ldots, x_N$
- Output: $P(w_i \mid w_{i-1}, \ldots, w_2, w_1)$

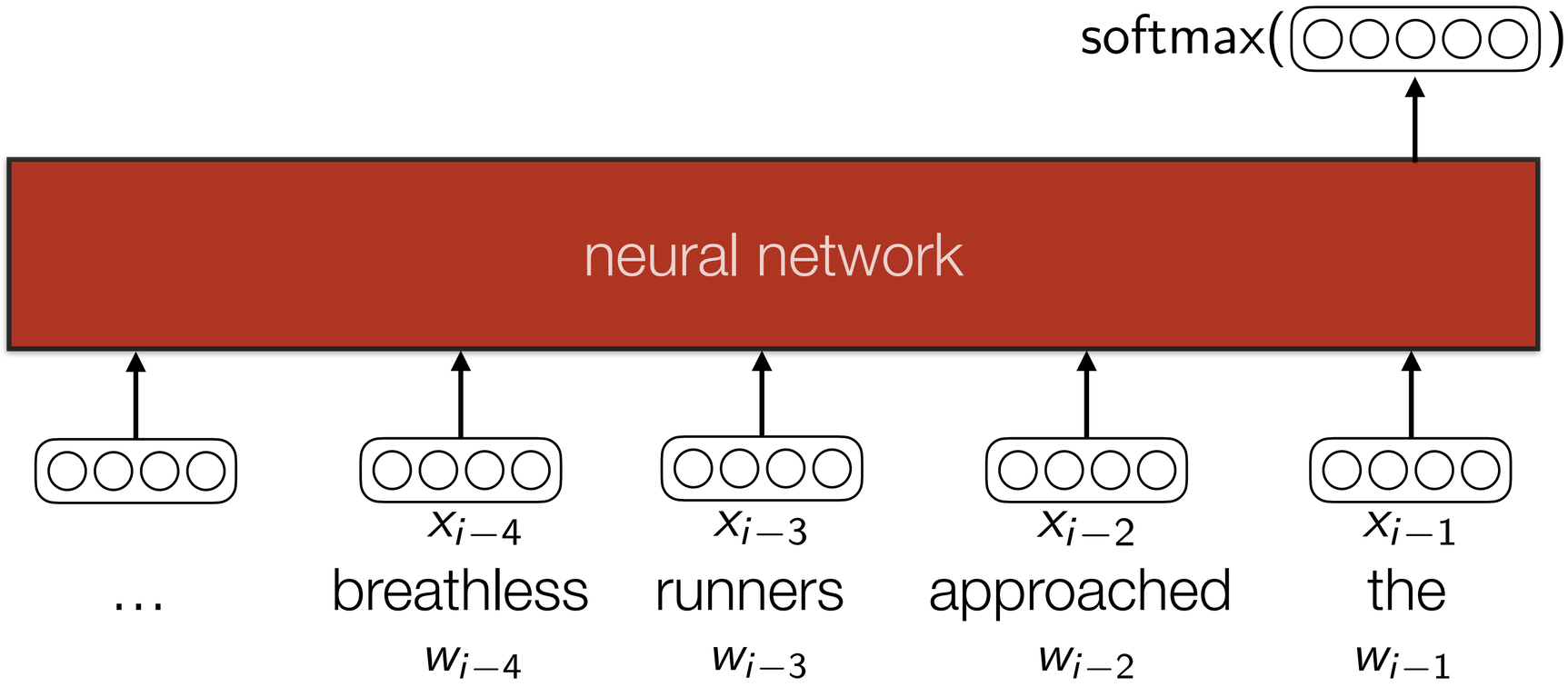




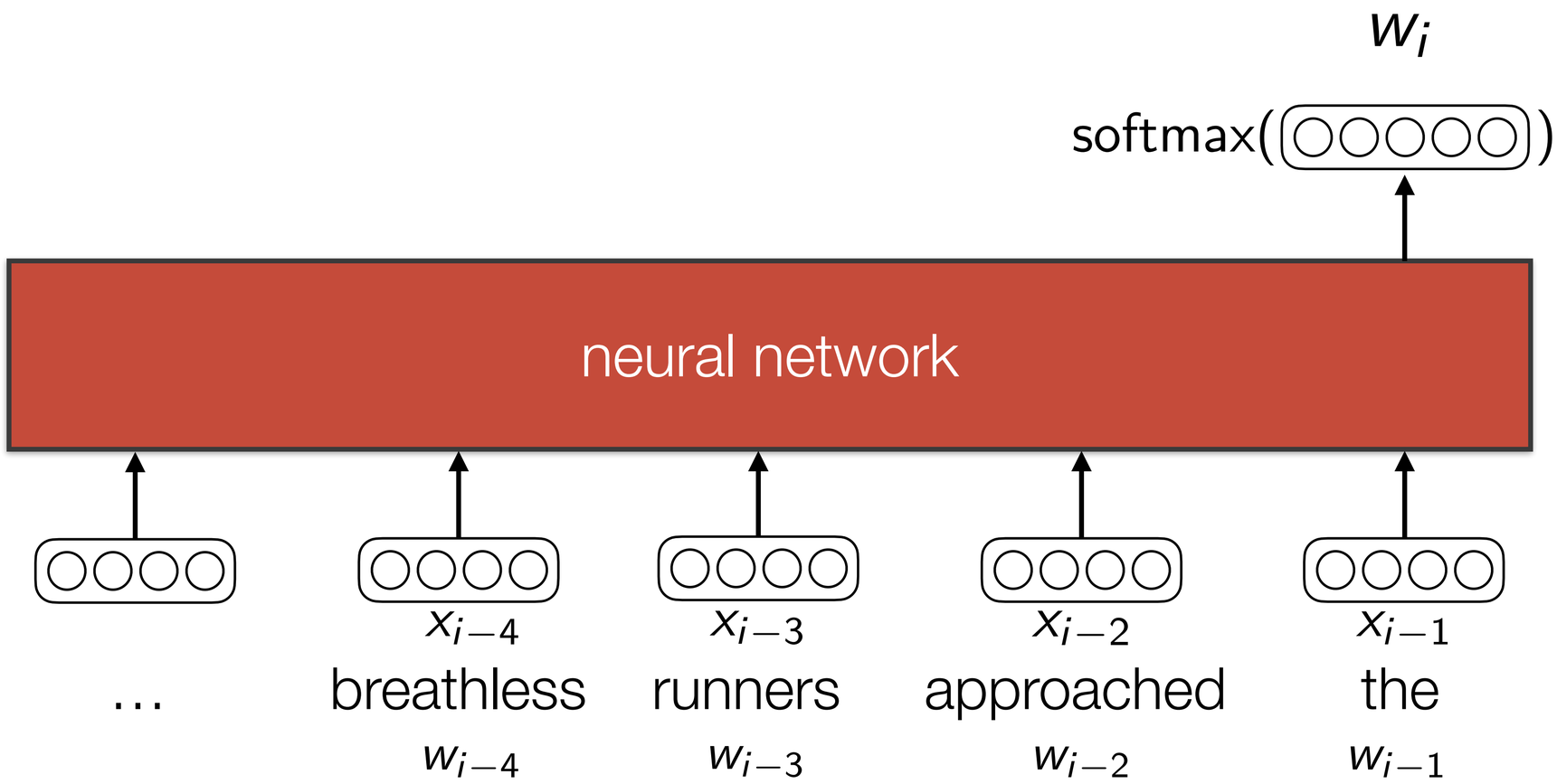
Neural language models

- Don’t count, predict!
- Input: word embeddings $x_1, x_2, \ldots, x_N$
- Output: $P(w_i \mid w_{i-1}, \ldots, w_2, w_1)$



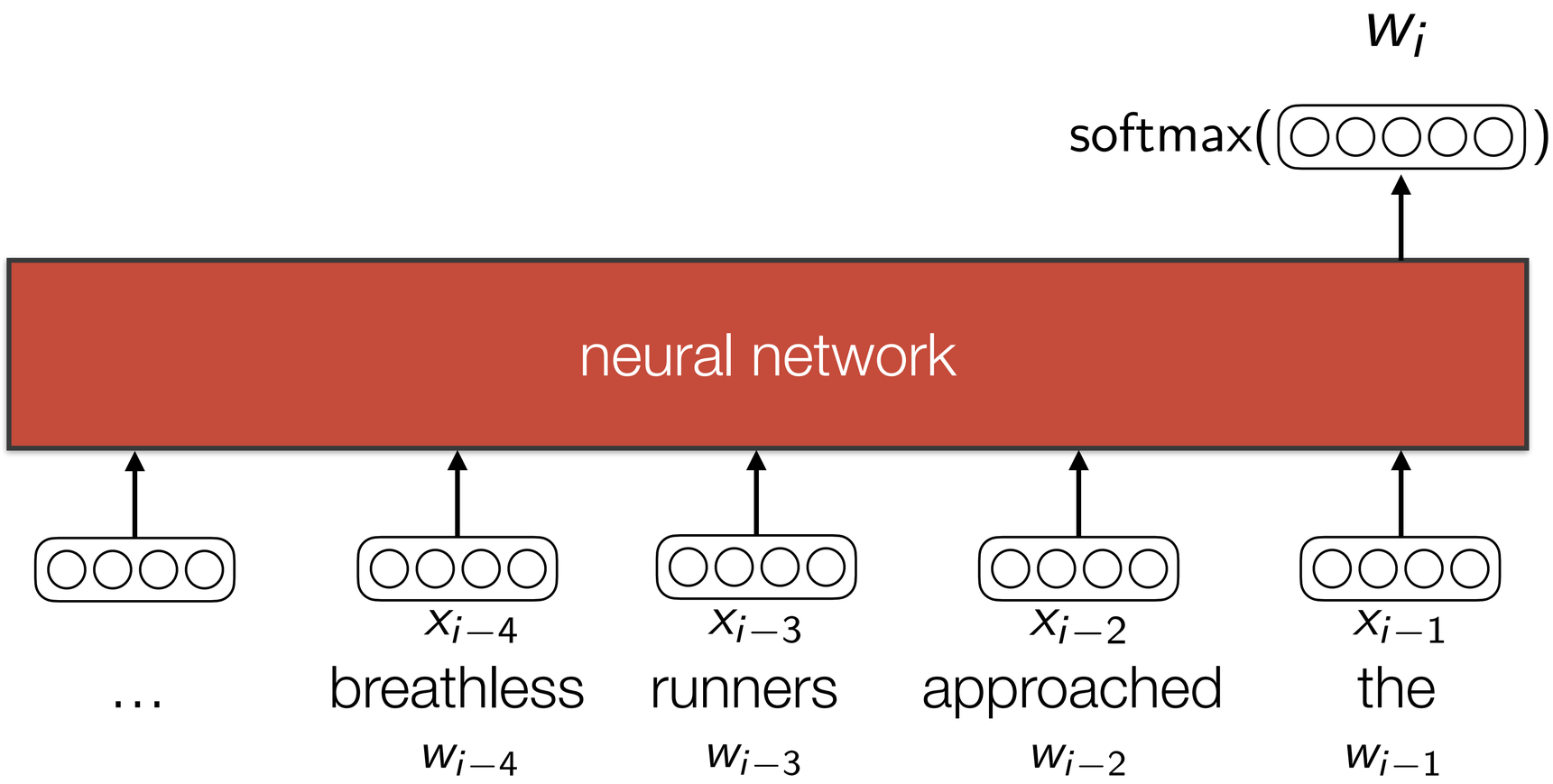


A feed-forward neural language model

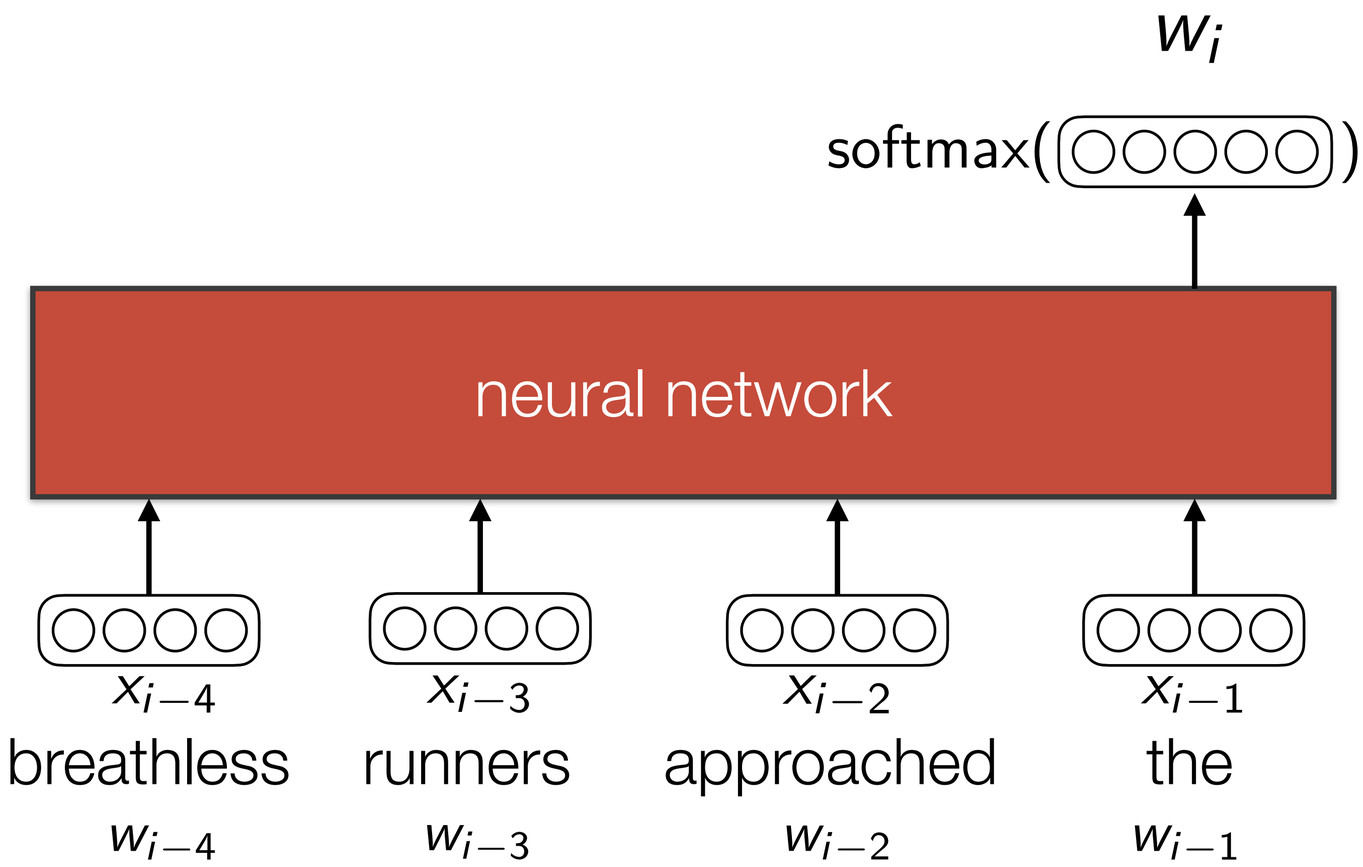


A feed-forward neural language model

- Fixed-size input (here: 4)

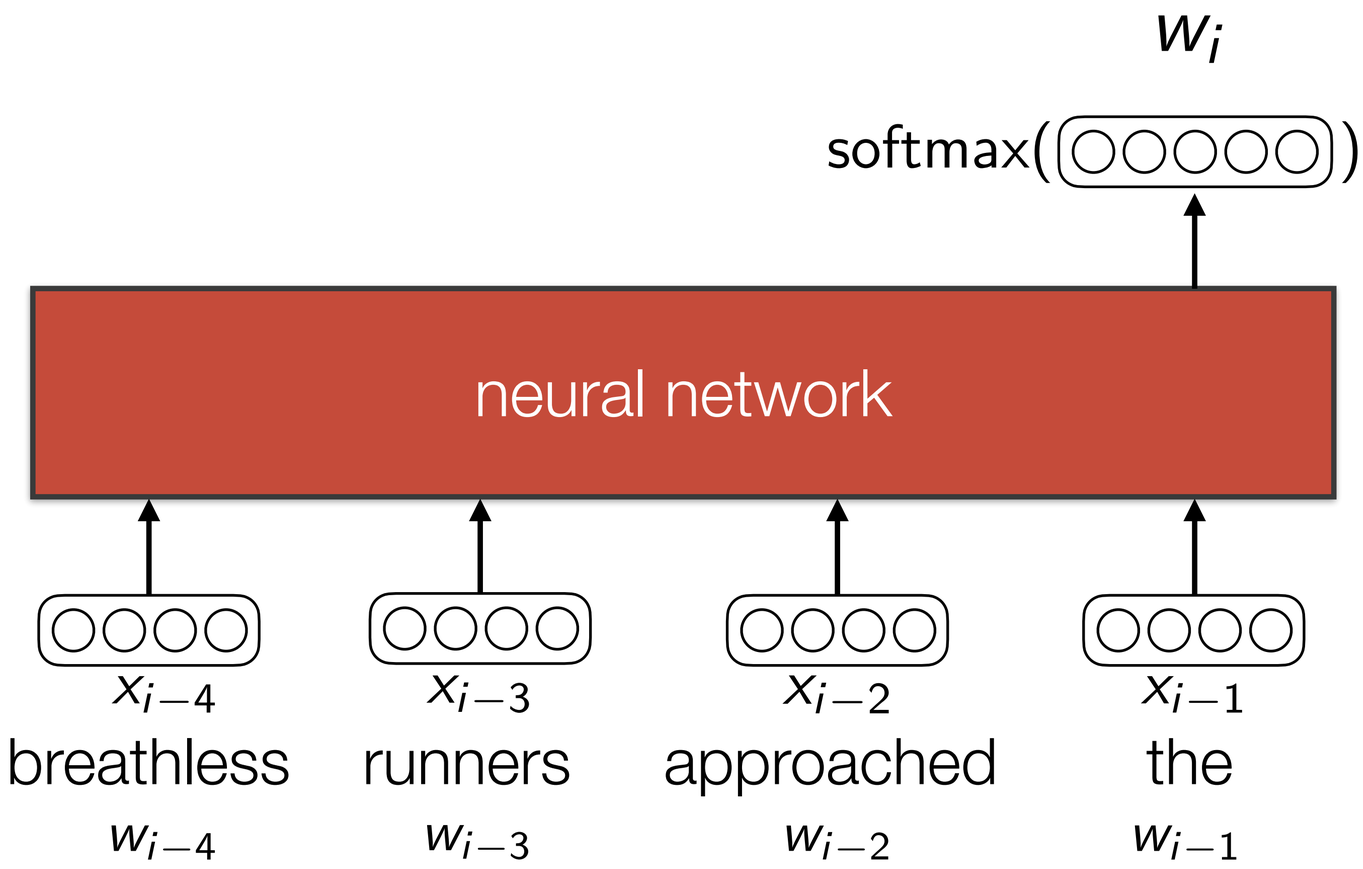


A feed-forward neural language model



A feed-forward neural language model

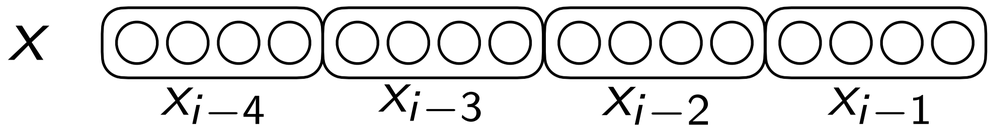
- Concatenate *k* word embeddings: $\mathbf{x} = [x_{i-4}; x_{i-3}; x_{i-2}; x_{i-1}]$



A feed-forward neural language model

- Concatenate *k* word embeddings: $\mathbf{x} = [x_{i-4}; x_{i-3}; x_{i-2}; x_{i-1}]$

dims:



$[4d_{word}]$

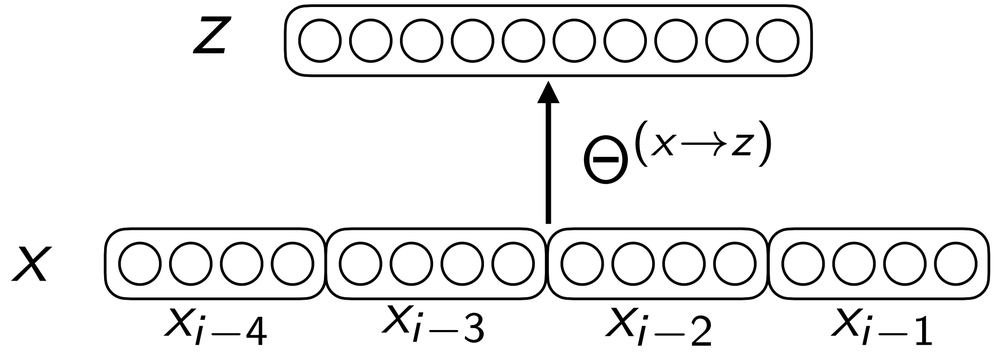
breathless runners approached the

w_{i-4} w_{i-3} w_{i-2} w_{i-1}

A feed-forward neural language model

- Concatenate *k* word embeddings: $\mathbf{x} = [x_{i-4}; x_{i-3}; x_{i-2}; x_{i-1}]$
- Hidden layer: $f(\Theta^{(x \to z)} x)$

dims:



$[d_z]$

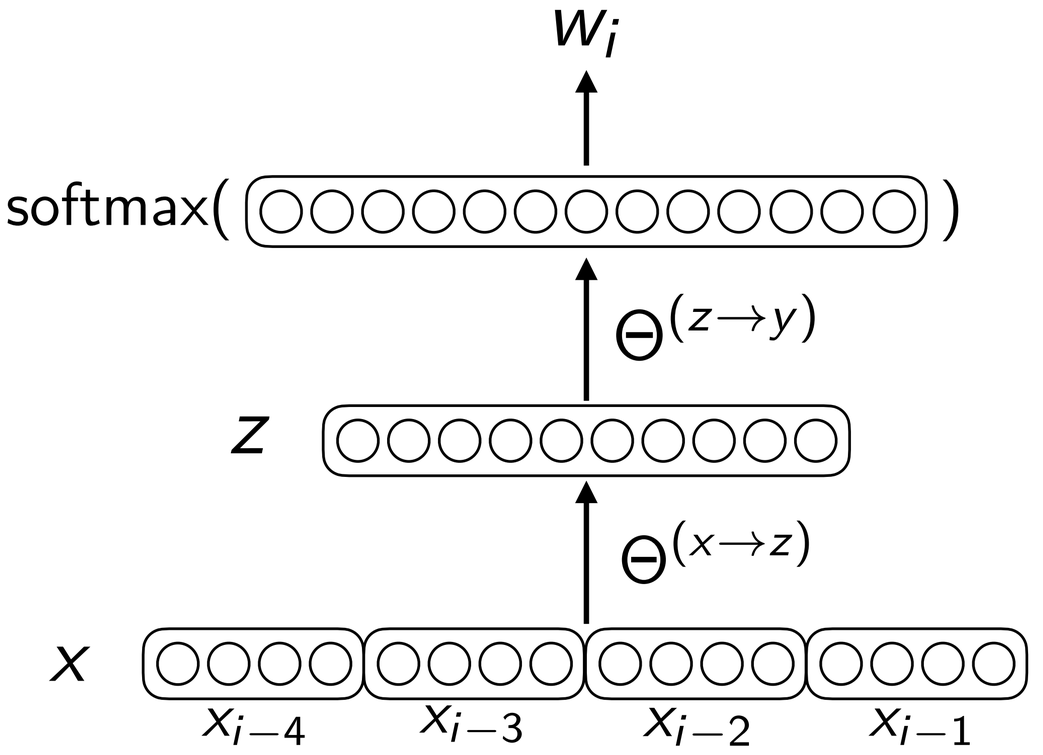
$[d_z \times 4d_{word}]$

$[4d_{word}]$

breathless w_{i-4} runners w_{i-3} approached w_{i-2} the w_{i-1}

A feed-forward neural language model

- Concatenate *k* word embeddings: $\mathbf{x} = [x_{i-4}; x_{i-3}; x_{i-2}; x_{i-1}]$
- Hidden layer: $f(\Theta^{(x \to z)} x)$
- Output layer: $\text{SoftMax}(\Theta^{(z \to y)} z)$



dims:

$[V]$

$[V \times d_z]$

$[d_z]$

$[d_z \times 4d_{word}]$

$[4d_{word}]$

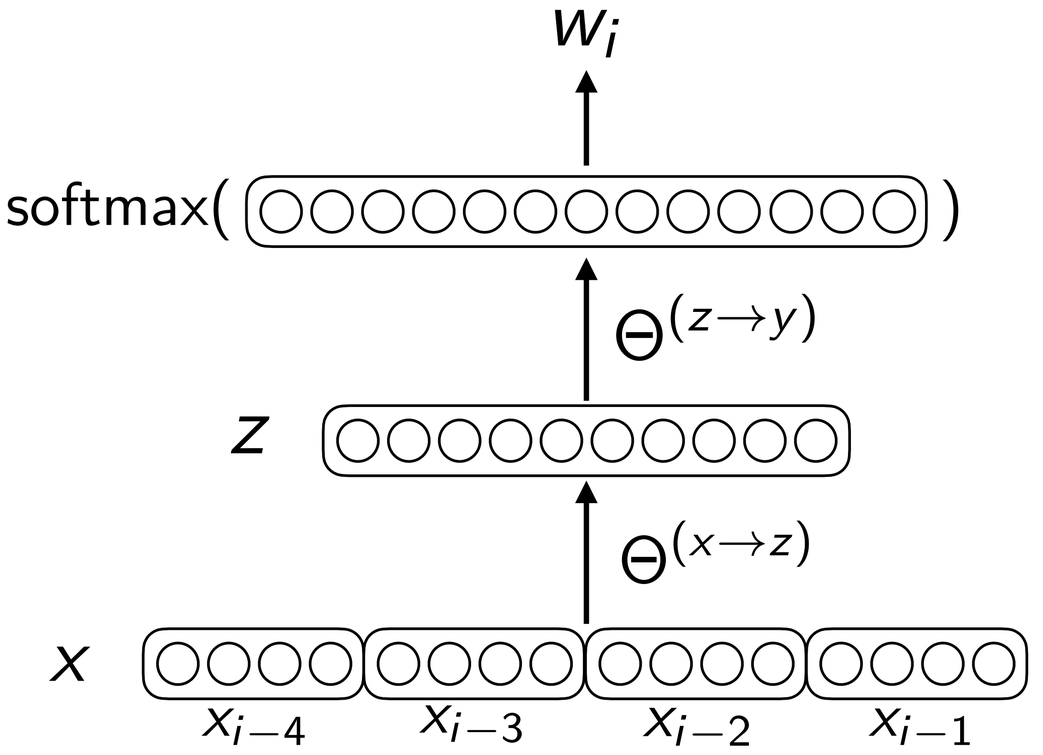
breathless runners approached the

w_{i-4} w_{i-3} w_{i-2} w_{i-1}

A feed-forward neural language model

- Concatenate *k* word embeddings: $\mathbf{x} = [x_{i-4}; x_{i-3}; x_{i-2}; x_{i-1}]$
- Hidden layer: $f(\Theta^{(x \to z)} x)$
- Output layer: $\text{SoftMax}(\Theta^{(z \to y)} z)$

$$P(w_i = j \mid z) = \frac{\exp(\theta_j^{(z \to y)} \cdot z)}{\sum_{j' \in V} \exp(\theta_{j'}^{(z \to y)} \cdot z)}$$



dims:

$[V]$

$[V \times d_z]$

$[d_z]$

$[d_z \times 4d_{word}]$

$[4d_{word}]$

breathless runners approached the

w_{i-4} w_{i-3} w_{i-2} w_{i-1}

A feed-forward neural language model

A feed-forward neural language model

- How does this compare to n-gram models from last class?

Improvements:

Remaining challenges:

A feed-forward neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)

Remaining challenges:

A feed-forward neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)

Remaining challenges:

A feed-forward neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Some sharing of representations across words

Remaining challenges:

A feed-forward neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Some sharing of representations across words

Remaining challenges:

- Still not enough context: larger context grows $\Theta^{(x \rightarrow z)}$ linear in *n.*

A feed-forward neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Some sharing of representations across words

Remaining challenges:

- Still not enough context: larger context grows $\Theta^{(x \to z)}$ linear in *n.*
- Each x_i uses different rows of $\Theta^{(x \to z)}$. Weights not shared across words.

A recurrent neural network language model

A recurrent neural network language model

- Solution: a **recurrent neural network (RNN)**

A recurrent neural network language model

- Solution: a **recurrent neural network (RNN)**
  - Maintain a context vector, *h.* At each timestep (w_j), compose the context with the current word x_j to create a new context for the next timestep:

$$h_j = RNN(x_j, h_{j-1})$$

A recurrent neural network language model

- Solution: a **recurrent neural network (RNN)**
  - Maintain a context vector, *h.* At each timestep (w_j), compose the context with the current word x_j to create a new context for the next timestep:

$$h_j = RNN(x_j, h_{j-1})$$

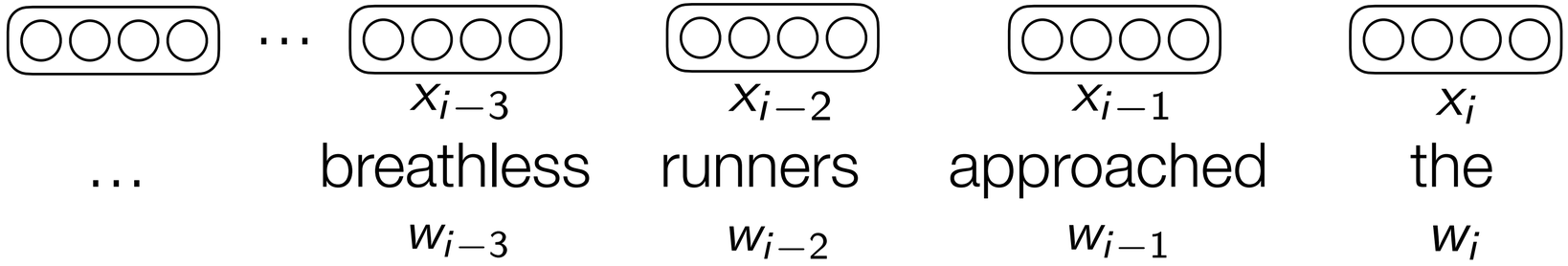
  - *Recurrent* because we repeatedly apply the same function $RNN(\cdot)$ each time.

A recurrent neural network language model

- Solution: a **recurrent neural network (RNN)**
  - Maintain a context vector, *h.* At each timestep (w_j), compose the context with the current word x_j to create a new context for the next timestep:

$$h_j = RNN(x_j, h_{j-1})$$

  - *Recurrent* because we repeatedly apply the same function $RNN(\cdot)$ each time.

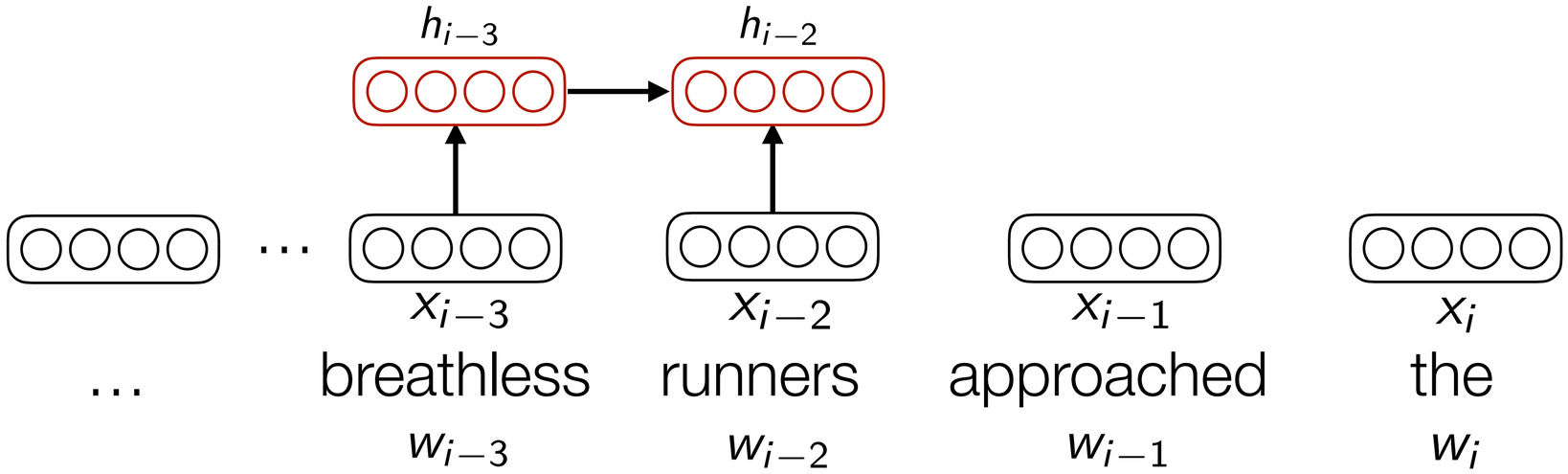


A recurrent neural network language model

- Solution: a **recurrent neural network (RNN)**
  - Maintain a context vector, *h.* At each timestep (w_j), compose the context with the current word x_j to create a new context for the next timestep:

$$h_j = RNN(x_j, h_{j-1})$$

  - *Recurrent* because we repeatedly apply the same function $RNN(\cdot)$ each time.

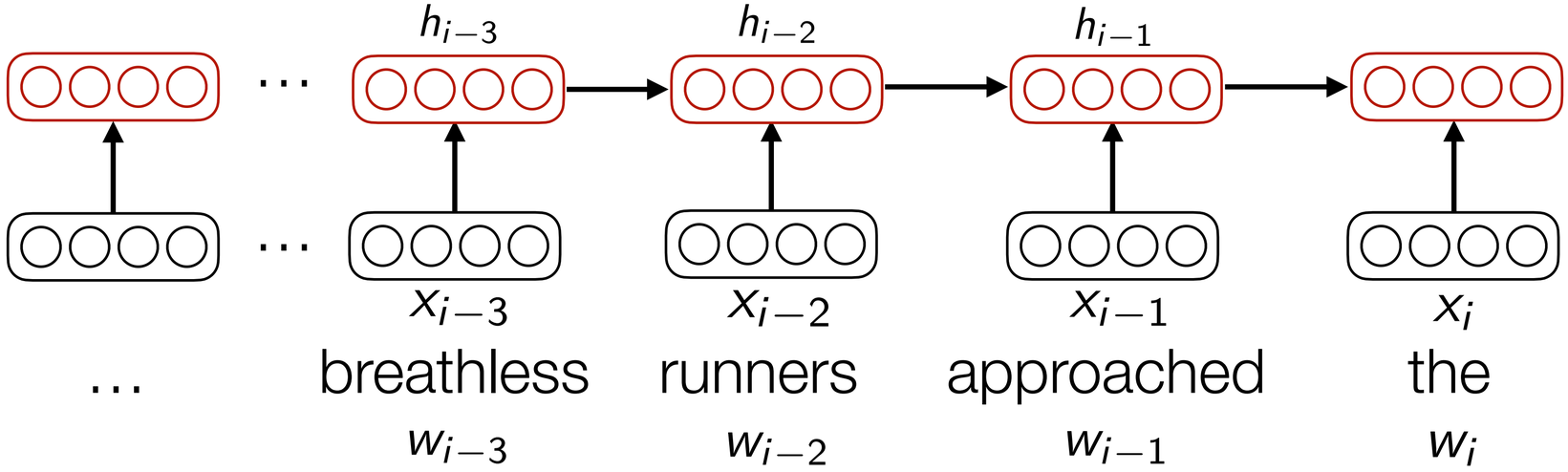


A recurrent neural network language model

- Solution: a **recurrent neural network (RNN)**
  - Maintain a context vector, *h.* At each timestep (w_j), compose the context with the current word x_j to create a new context for the next timestep:

$$h_j = RNN(x_j, h_{j-1})$$

  - *Recurrent* because we repeatedly apply the same function $RNN(\cdot)$ each time.

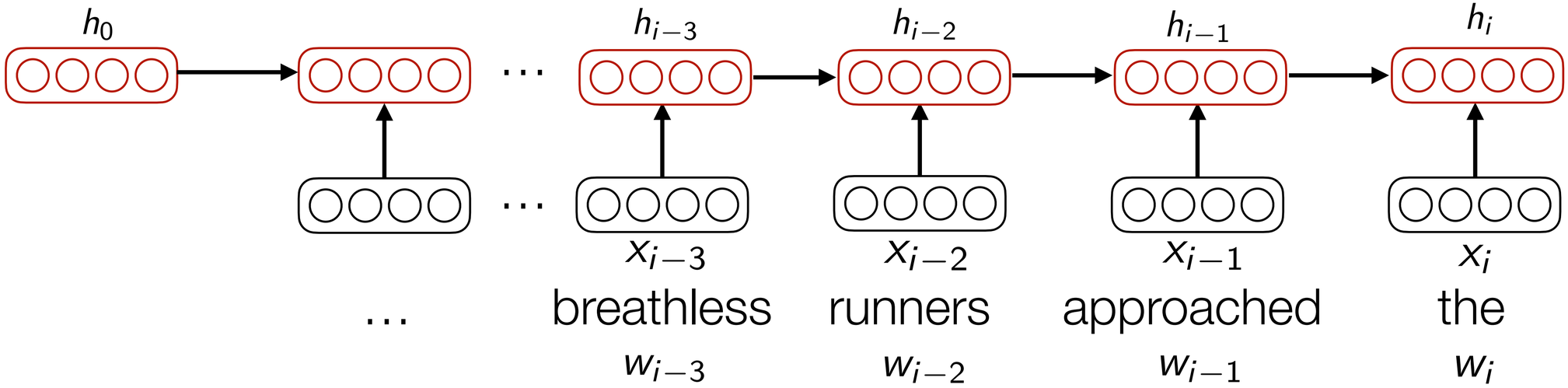


A recurrent neural network language model

- Solution: a **recurrent neural network (RNN)**
  - Maintain a context vector, *h.* At each timestep (w_j), compose the context with the current word x_j to create a new context for the next timestep:

$$h_j = RNN(x_j, h_{j-1})$$

  - *Recurrent* because we repeatedly apply the same function $RNN(\cdot)$ each time.

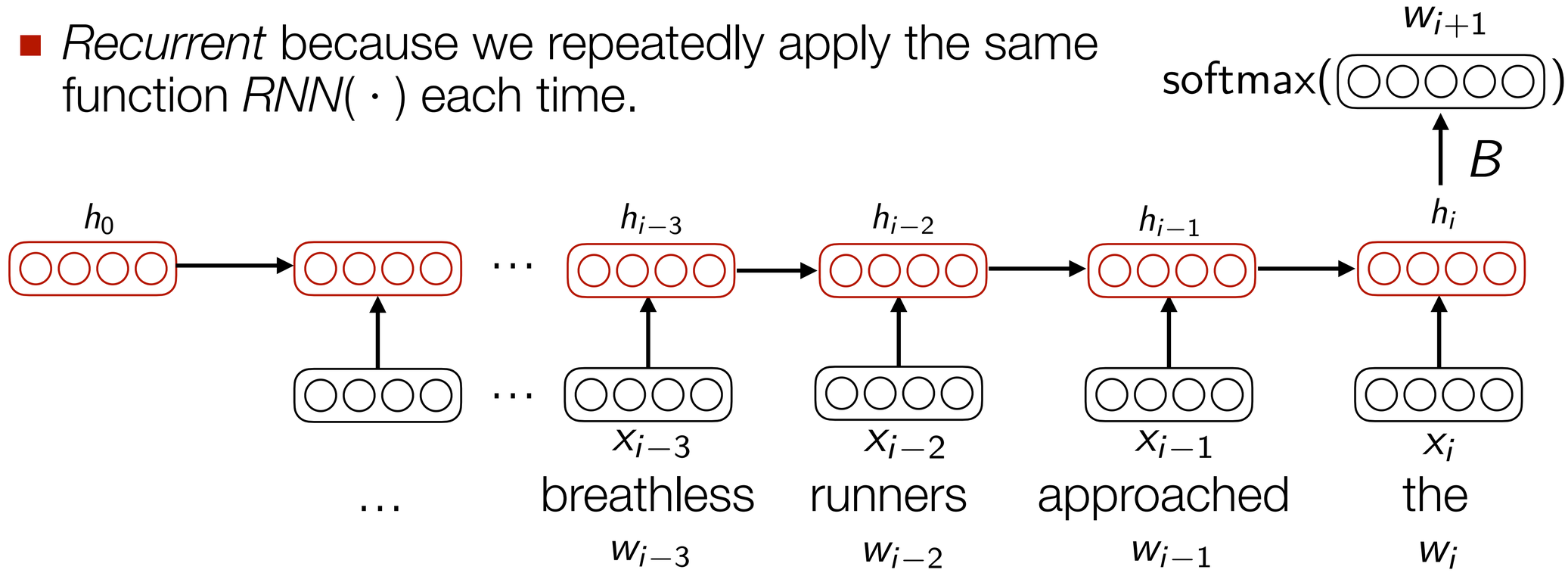


A recurrent neural network language model

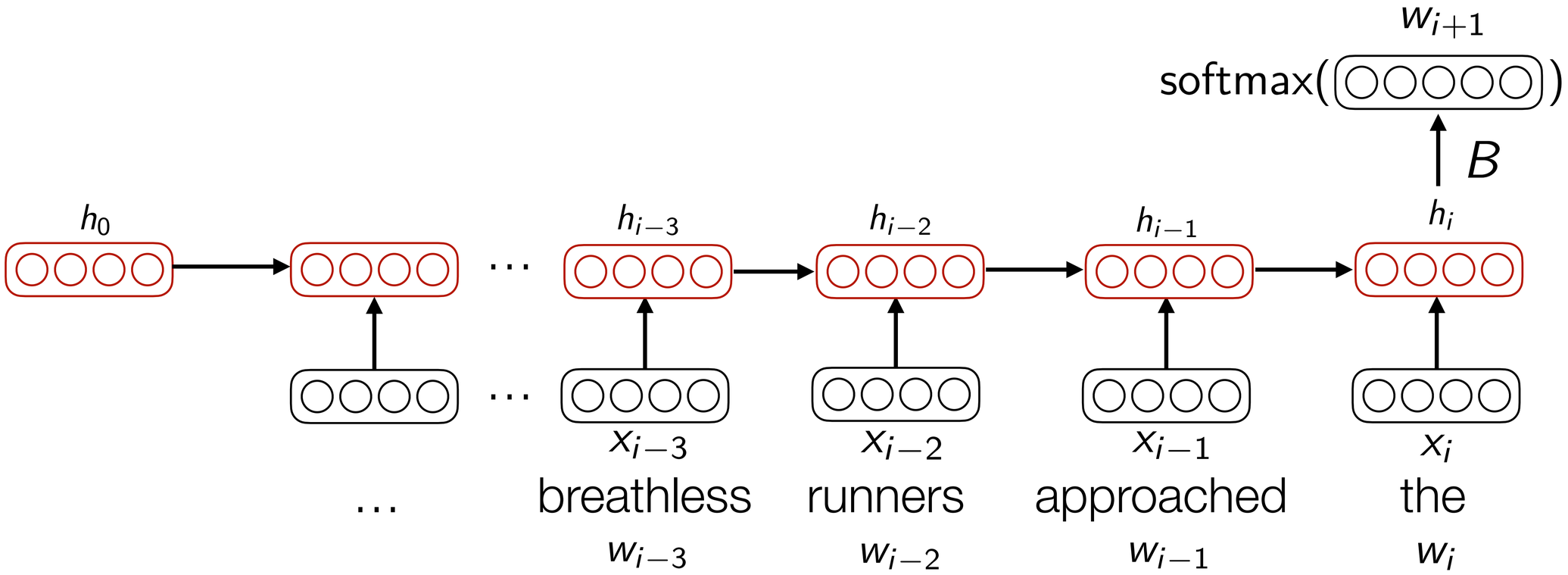
- Solution: a **recurrent neural network (RNN)**
  - Maintain a context vector, *h.* At each timestep (w_j), compose the context with the current word x_j to create a new context for the next timestep:

$$h_j = RNN(x_j, h_{j-1})$$

  - *Recurrent* because we repeatedly apply the same function $RNN(\cdot)$ each time.

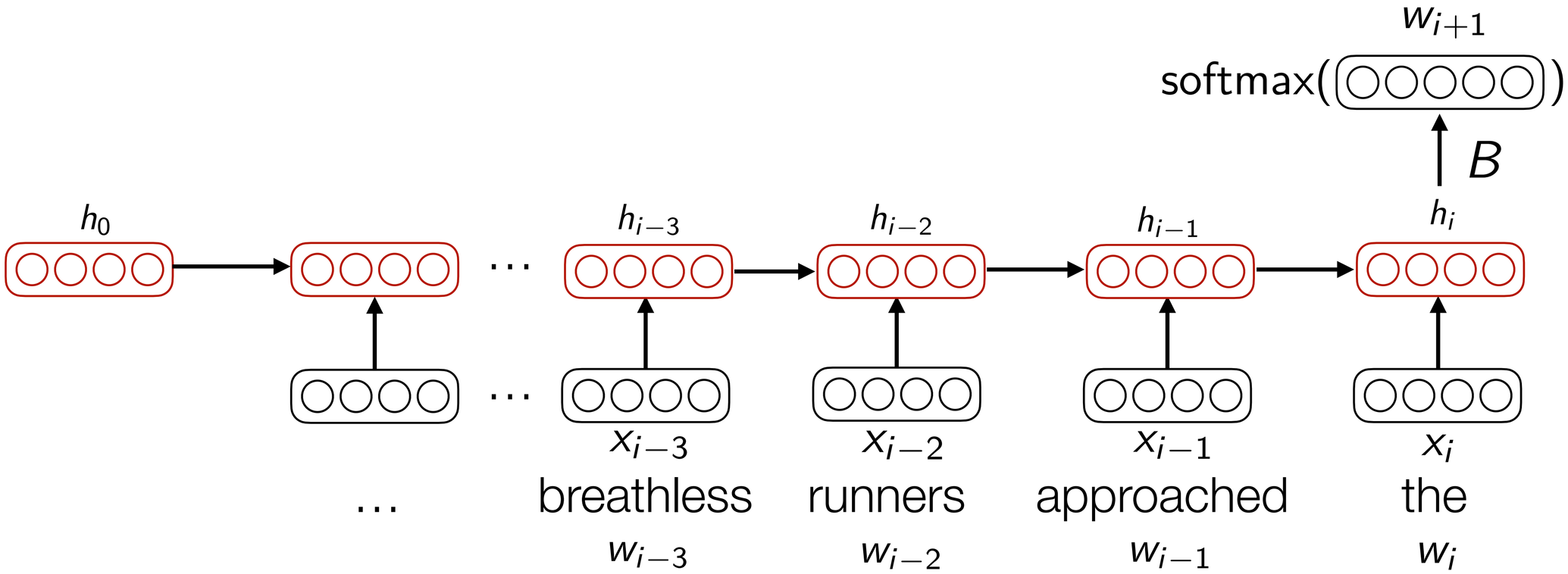


A recurrent neural network language model



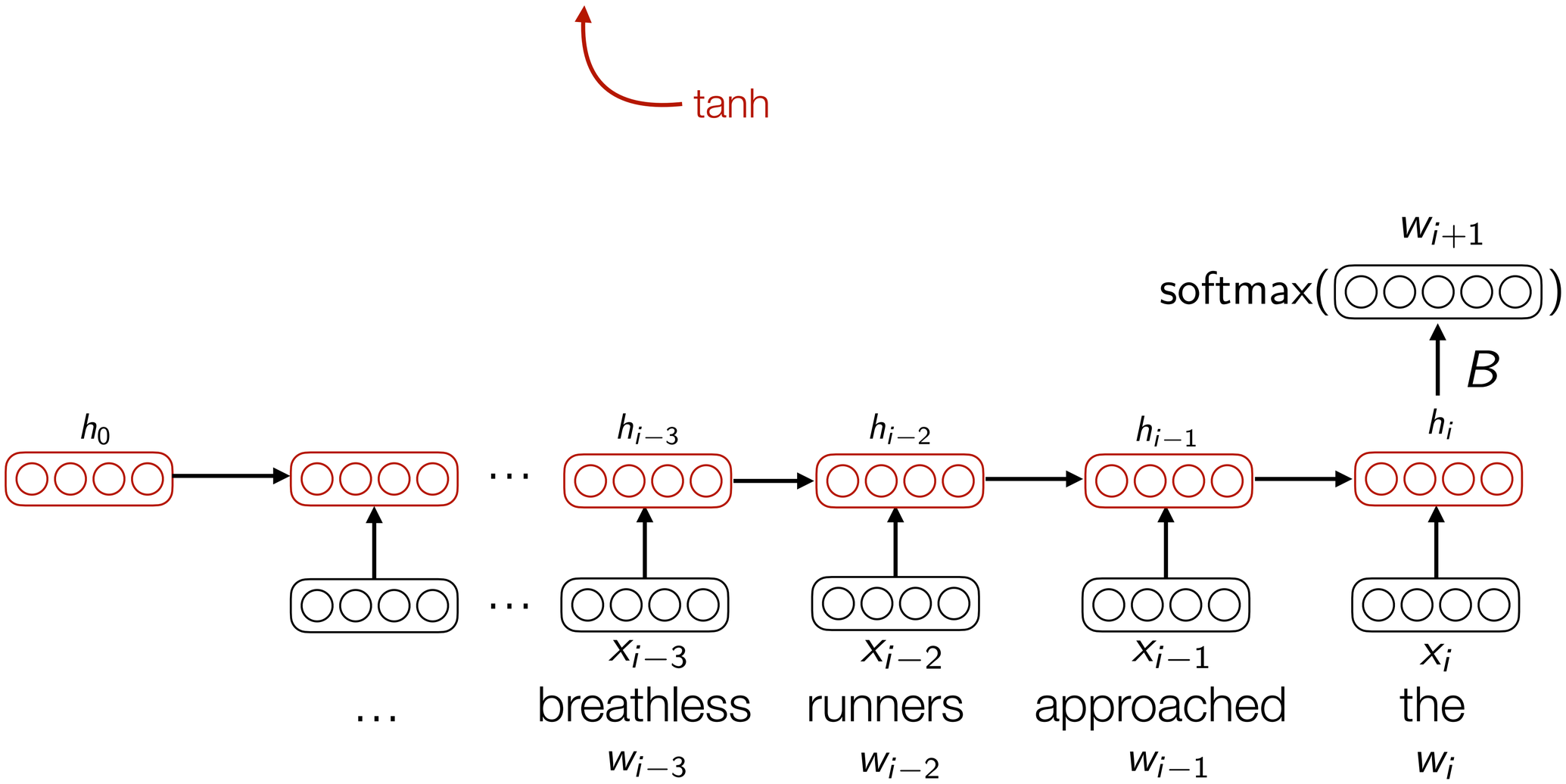
A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$



A recurrent neural network language model

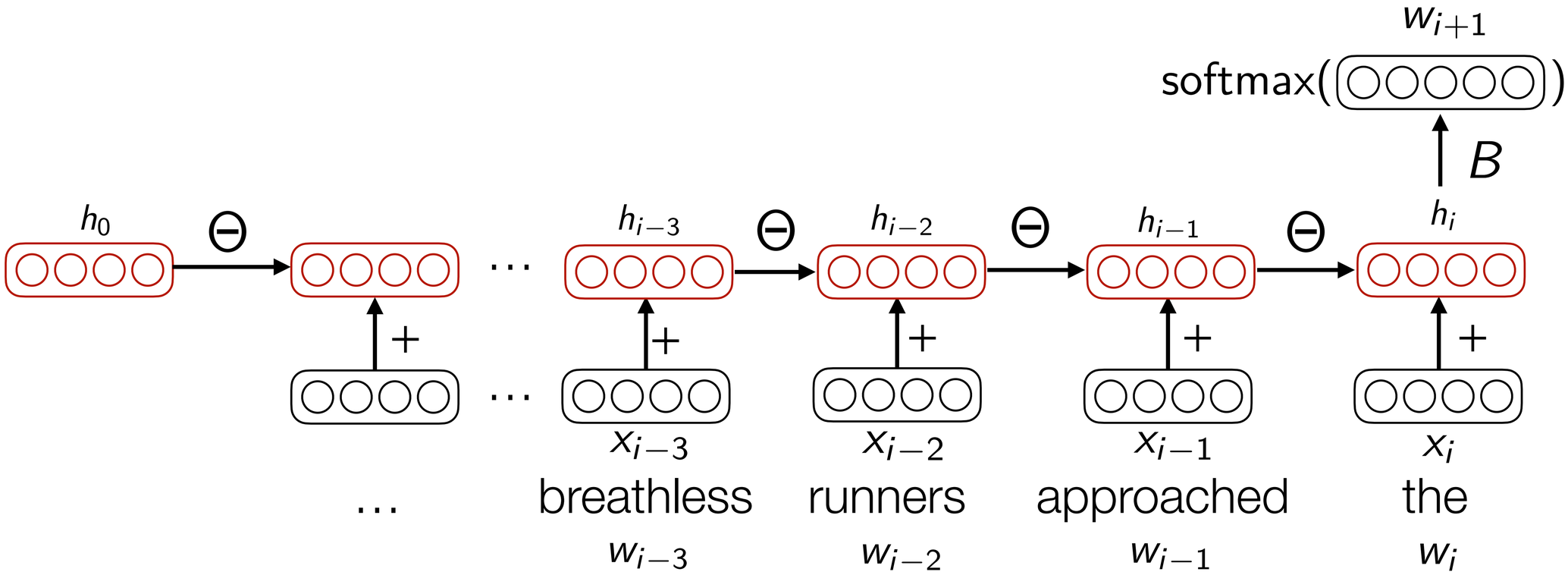
- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$



A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$

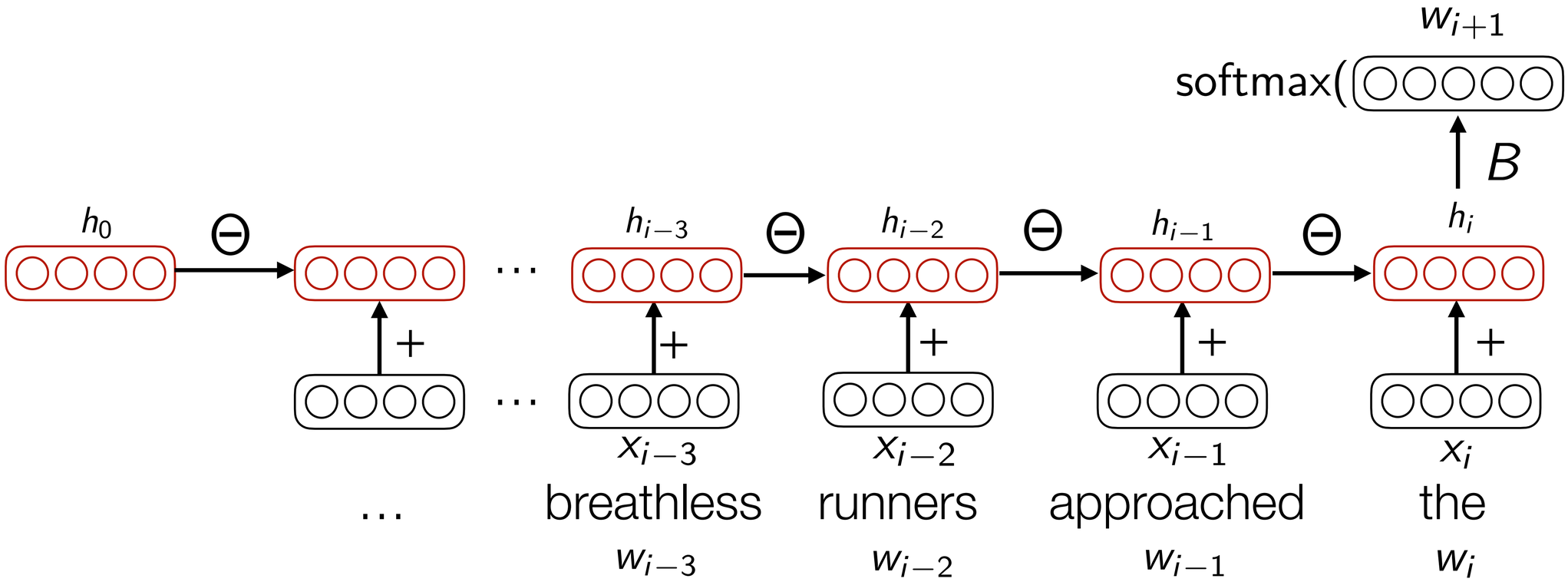
tanh



A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$

tanh

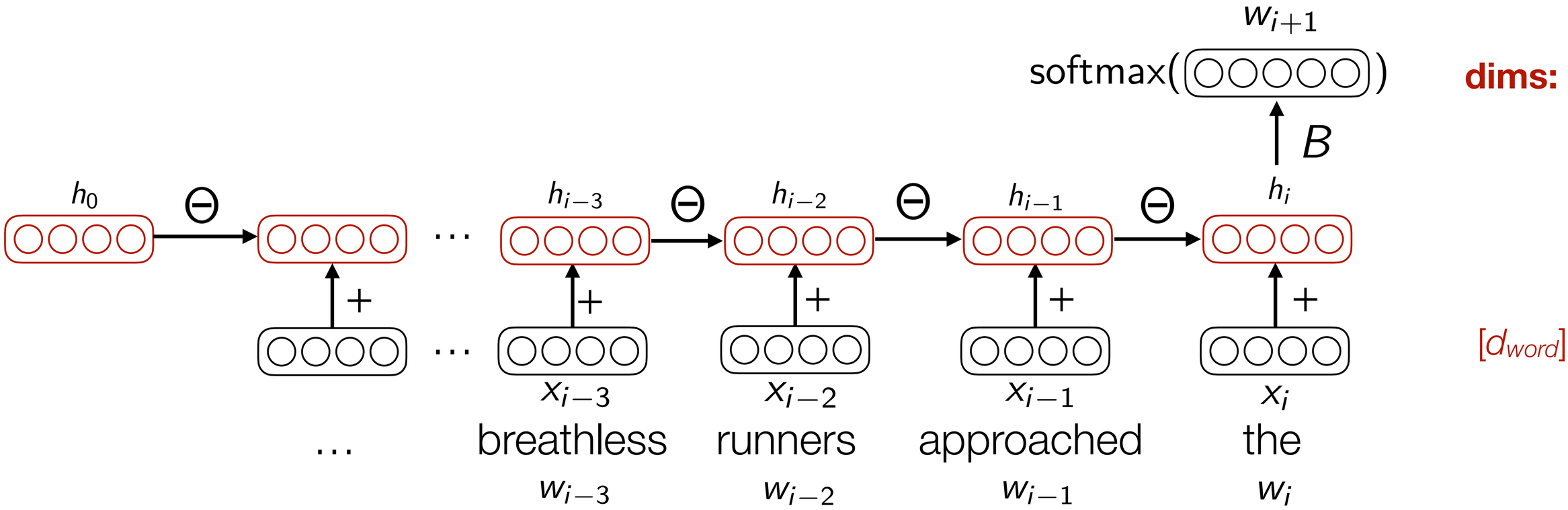


dims:

A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$

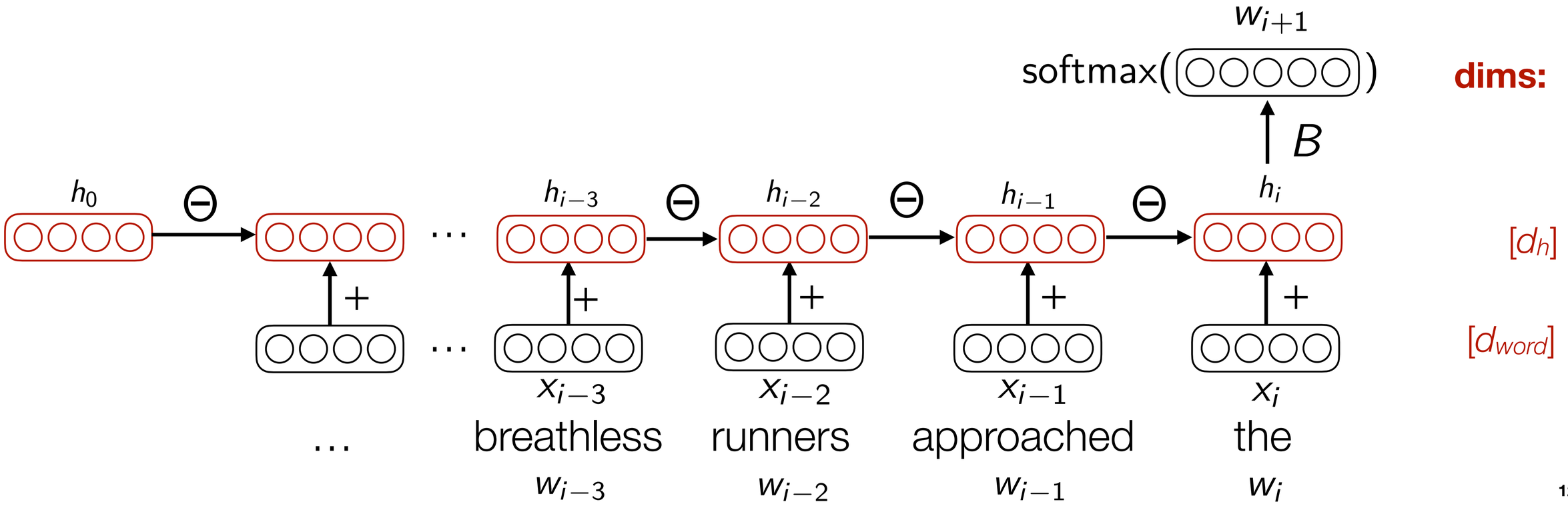
tanh



A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$

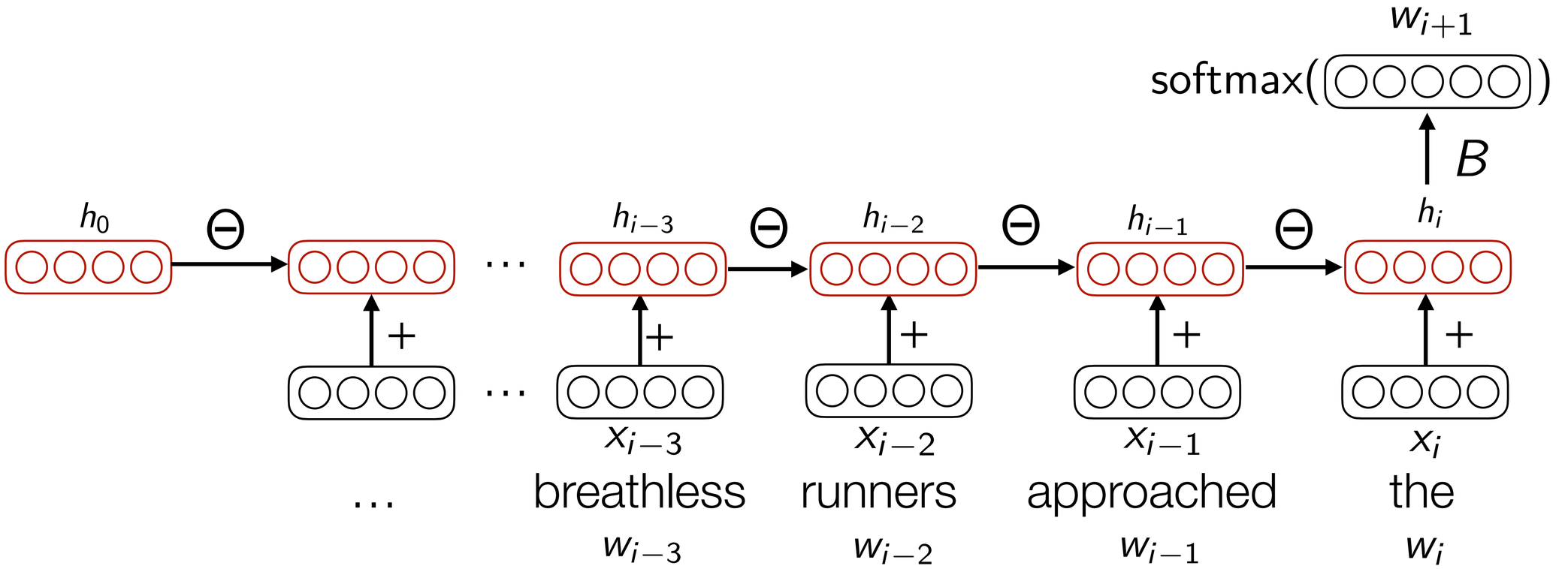
tanh



A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$

tanh



dims:

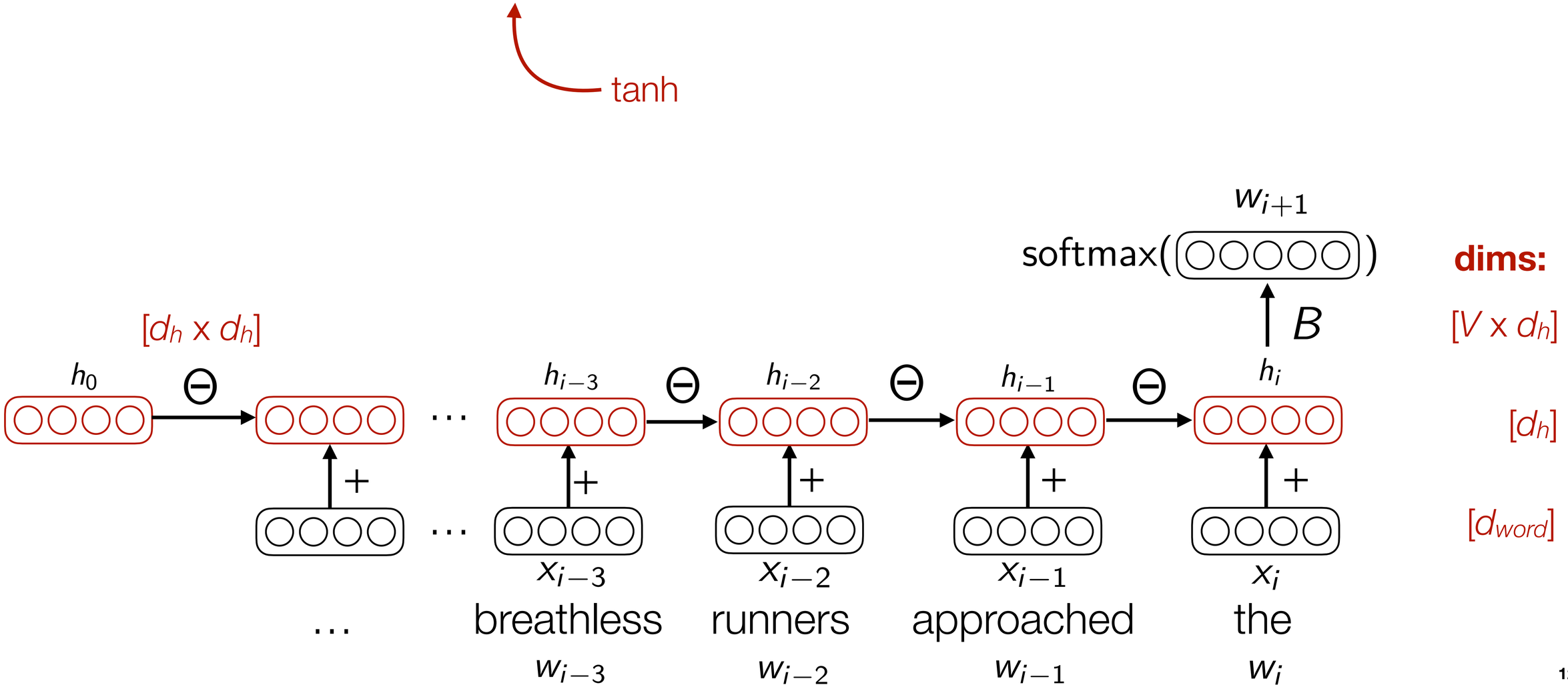
$[V \times d_h]$

$[d_h]$

$[d_{word}]$

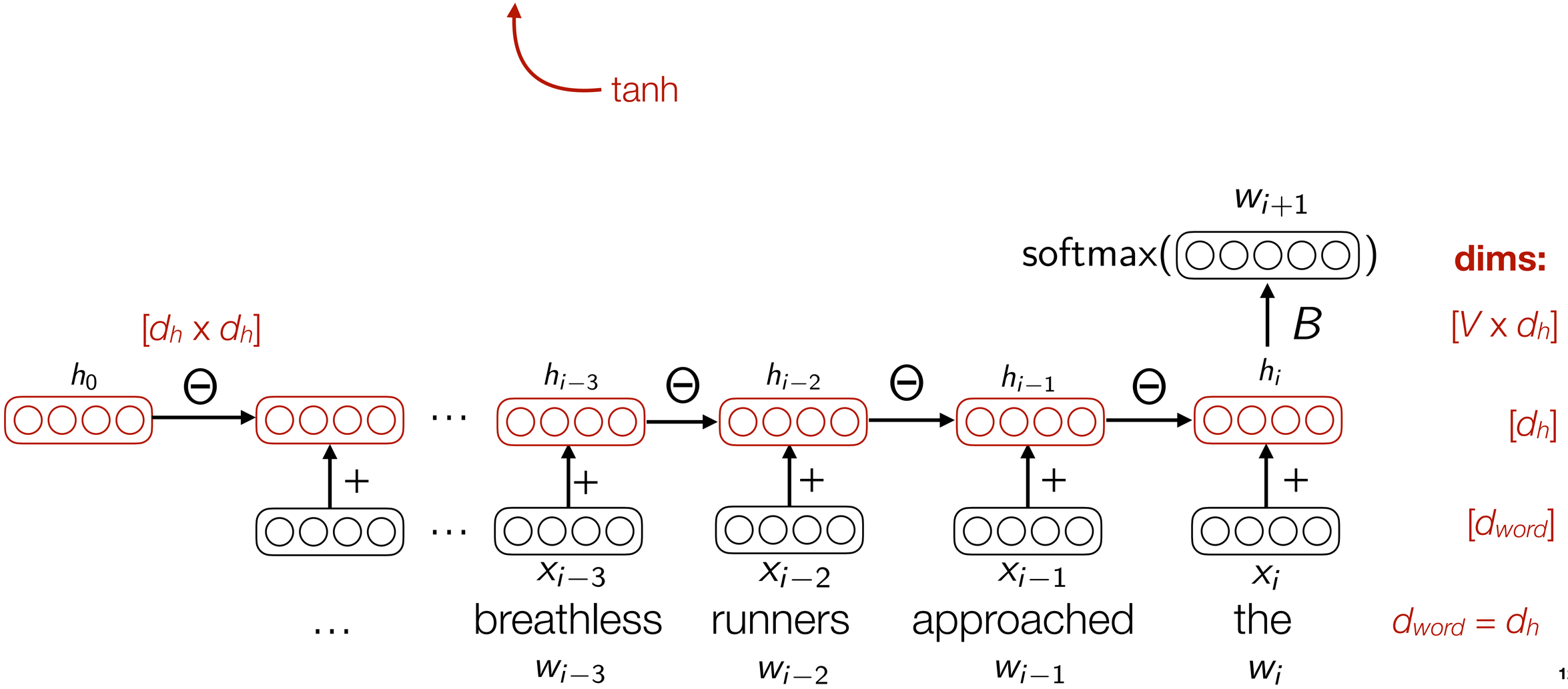
A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$



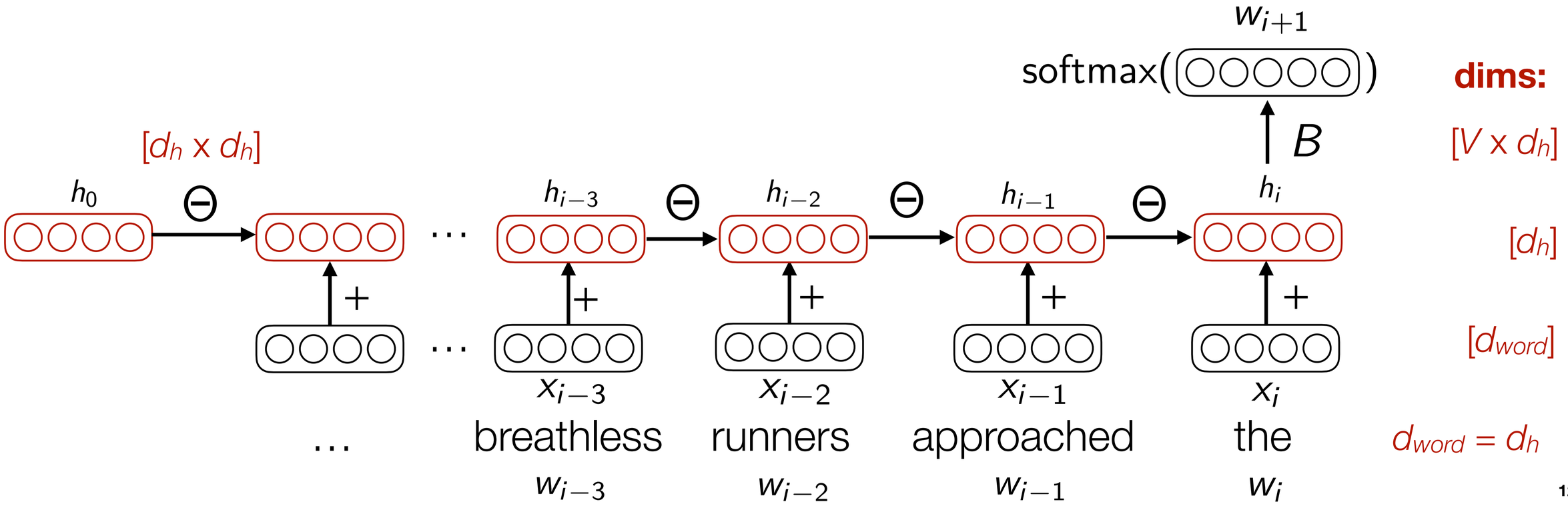
A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$

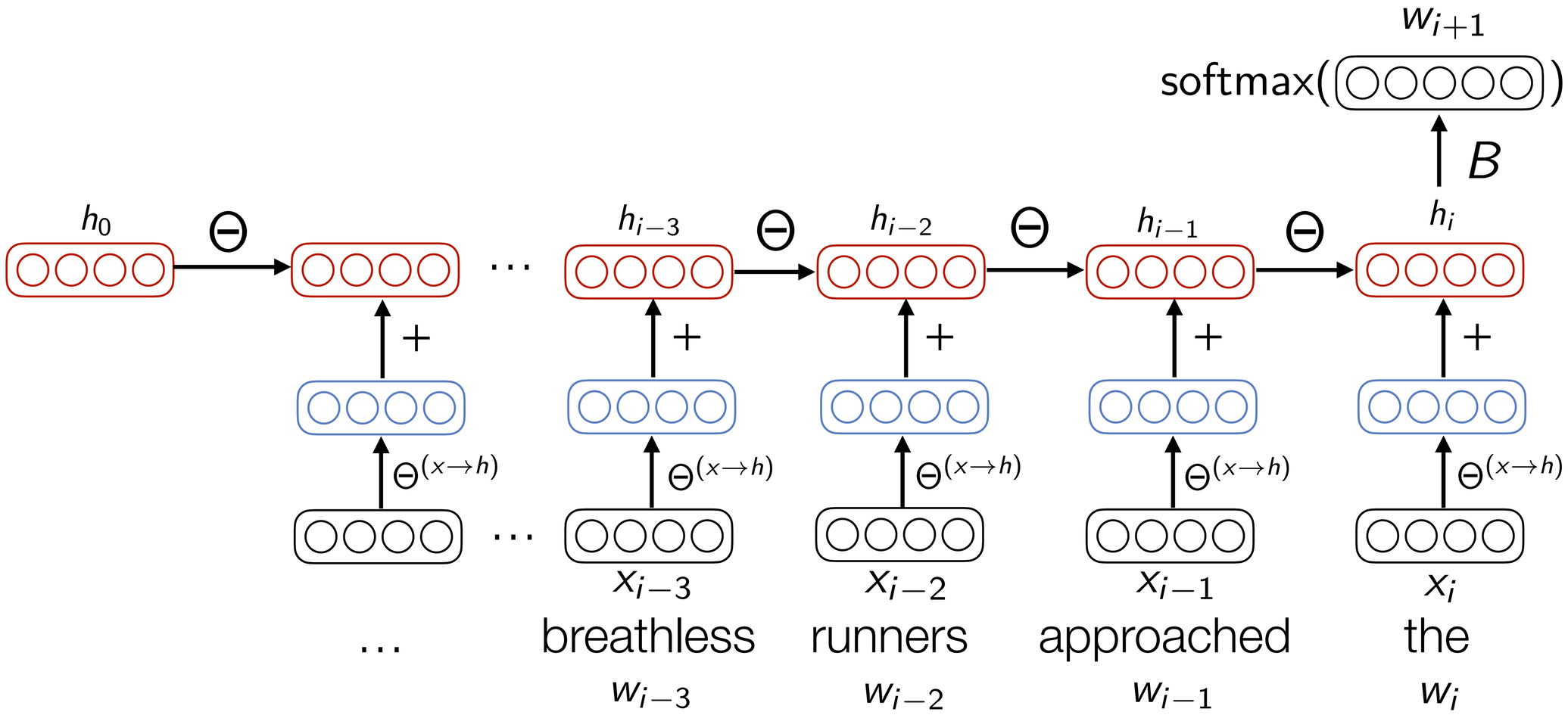


A recurrent neural network language model

- **Elman network**: $RNN(\cdot) = f(\Theta h_{i-1} + x_i)$
- If $d_{word} = d_h$, can tie input (X) and output (B) word embeddings.

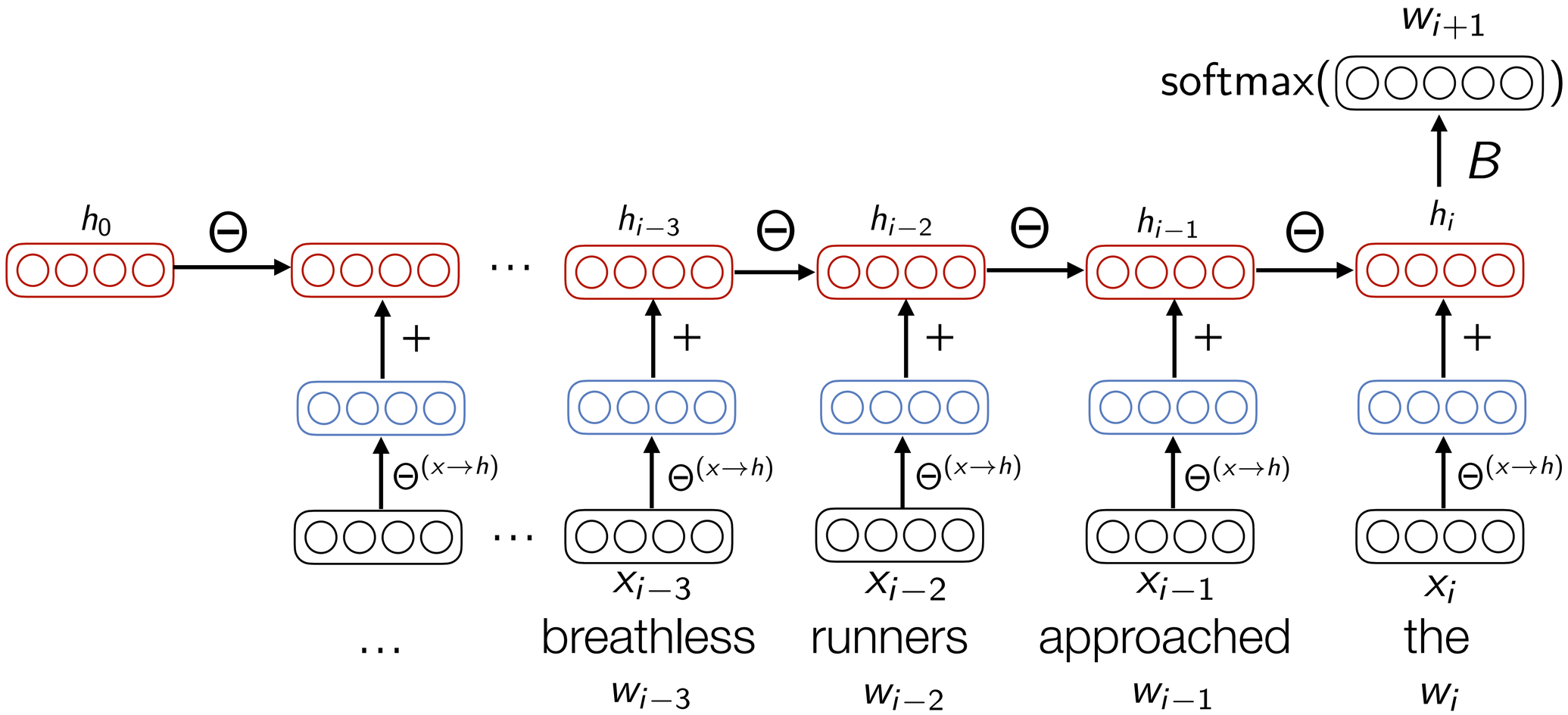


A recurrent neural network language model



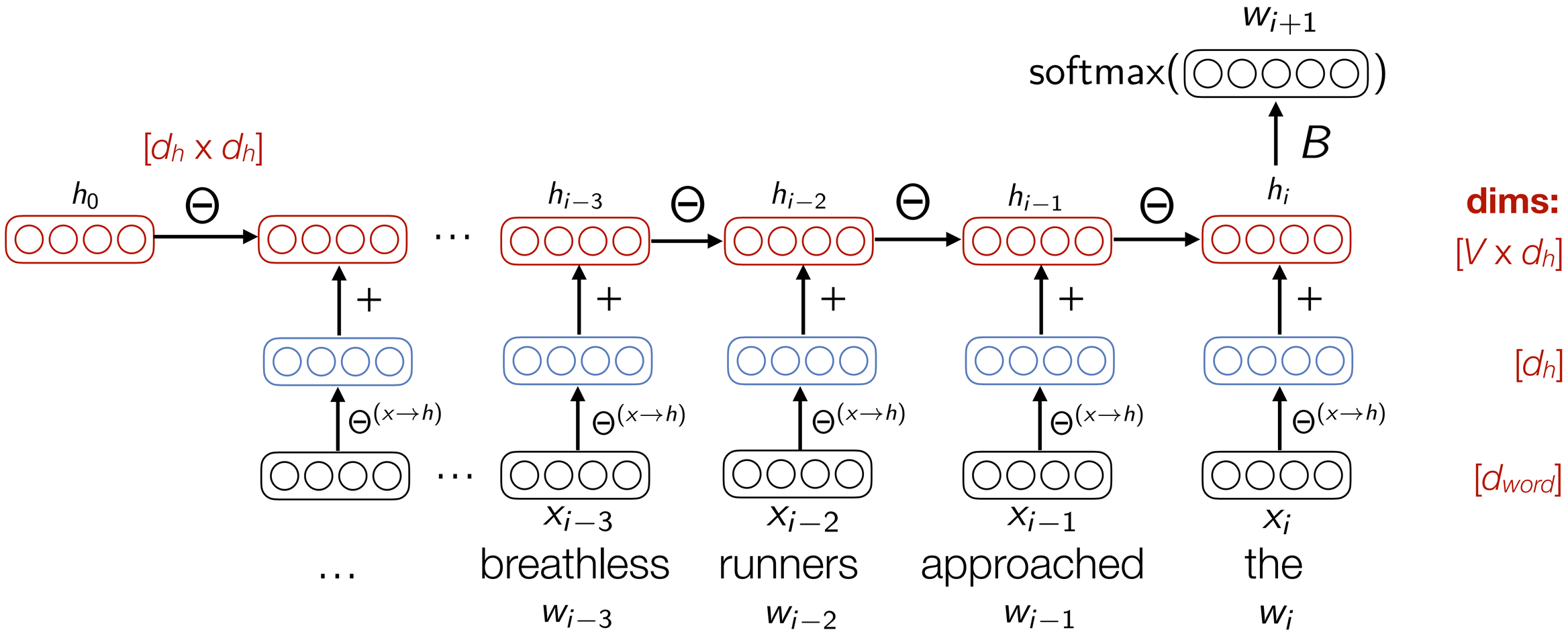
A recurrent neural network language model

- If we don't want to tie X with B, can also add another projection matrix $\Theta^{(x \to h)}$



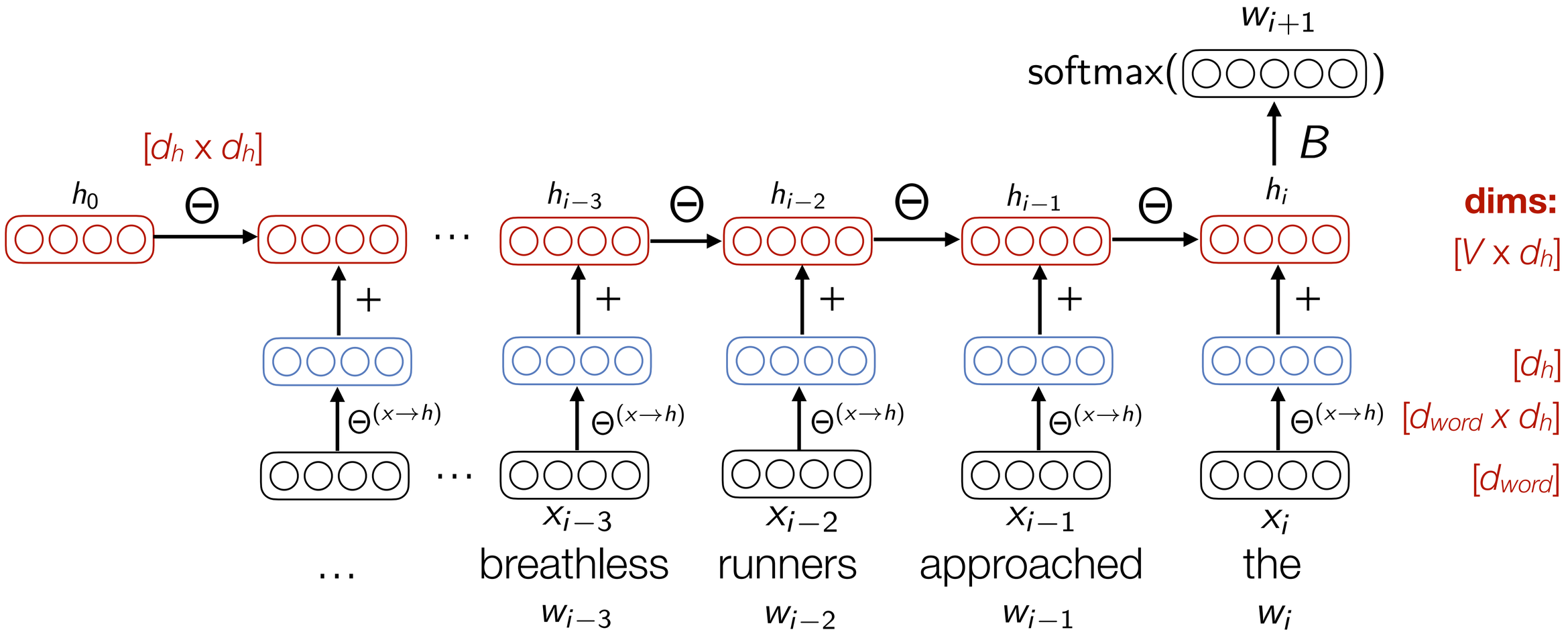
A recurrent neural network language model

- If we don't want to tie X with B, can also add another projection matrix $\Theta^{(x \to h)}$

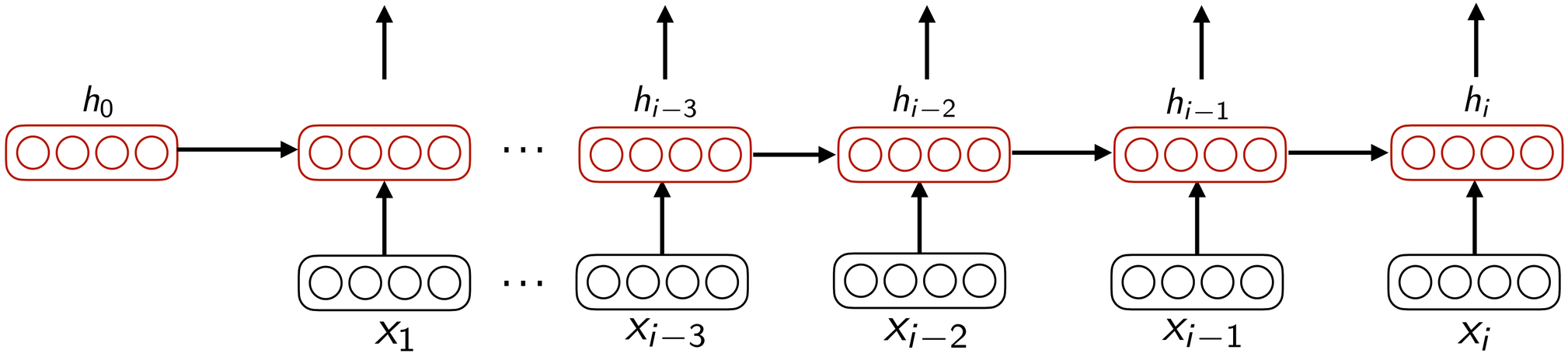


A recurrent neural network language model

- If we don't want to tie X with B, can also add another projection matrix $\Theta^{(x \to h)}$

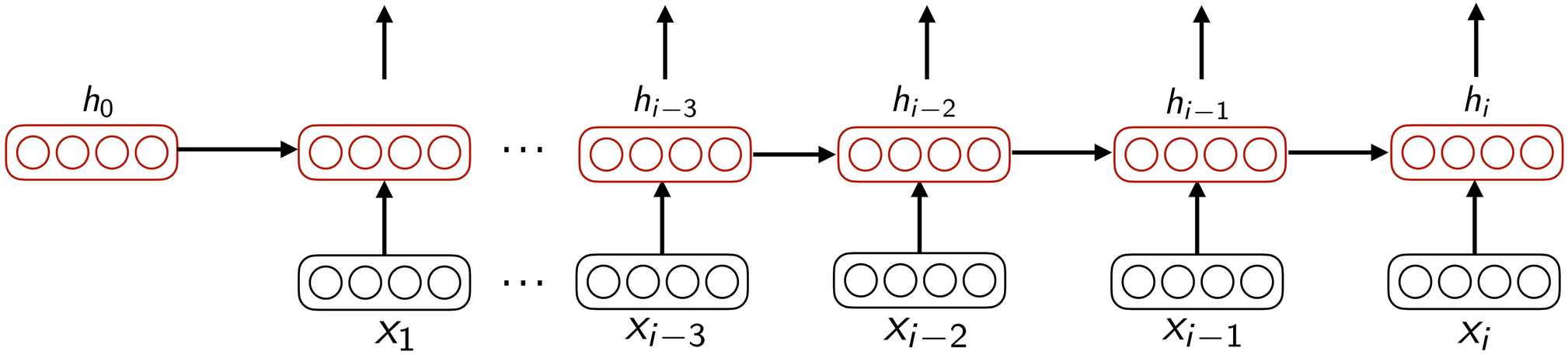


Learning RNNs



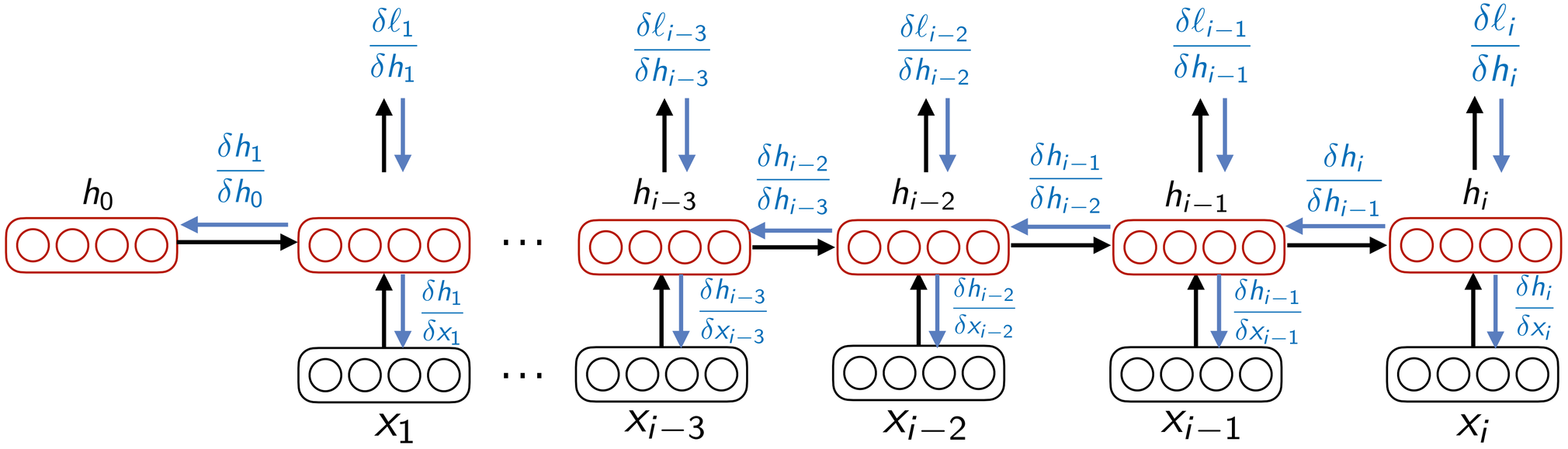
Learning RNNs

- Like feed-forward neural networks, RNNs can be trained using backpropagation and stochastic gradient descent.



Learning RNNs

- Like feed-forward neural networks, RNNs can be trained using backpropagation and stochastic gradient descent.
- **Backpropagation through time**: In addition to each layer, now gradients must be computed with respect to each *timestep*. “Unroll” the RNN and treat the parameters at each timestep as if it were another layer; apply the chain rule.



A recurrent neural language model

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

Remaining challenges:

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)

Remaining challenges:

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)

Remaining challenges:

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Sharing of representations across words

Remaining challenges:

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Sharing of representations across words
- Models long context

Remaining challenges:

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Sharing of representations across words
- Models long context

Remaining challenges:

- Softmax over large vocabulary

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Sharing of representations across words
- Models long context

Remaining challenges:

- Softmax over large vocabulary
- High variance / overfitting

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Sharing of representations across words
- Models long context

Remaining challenges:

- Softmax over large vocabulary
- High variance / overfitting
- Exploding and vanishing gradients

A recurrent neural language model

- How does this compare to n-gram models from last class?

Improvements:

- Model size: O(V), not O(V^n)
- Sparsity (lack thereof)
- Sharing of representations across words
- Models long context

Remaining challenges:

- Softmax over large vocabulary
- High variance / overfitting
- Exploding and vanishing gradients

But what about that huge softmax over V?

But what about that huge softmax over *V*?

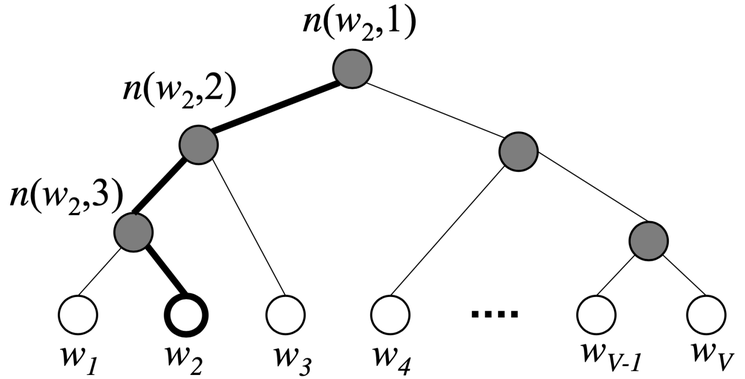
- Same problem as word embeddings: don’t want to score wrt entire vocab!

But what about that huge softmax over *V*?

- Same problem as word embeddings: don’t want to score wrt entire vocab!
- Solutions:

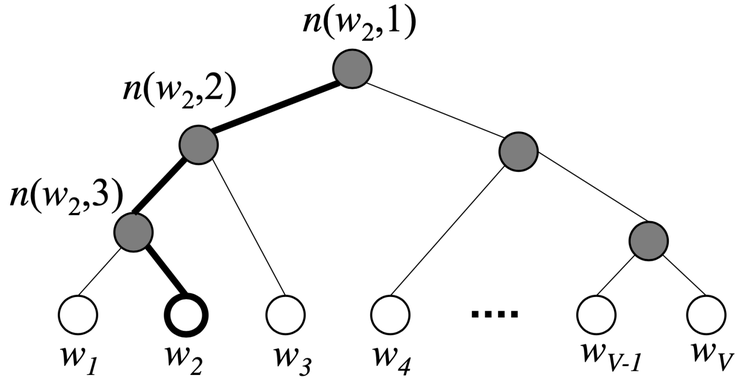
But what about that huge softmax over *V*?

- Same problem as word embeddings: don't want to score wrt entire vocab!
- Solutions:
  - Hierarchical softmax



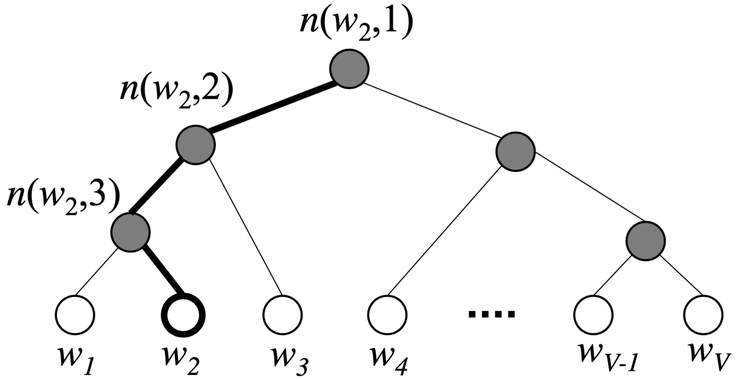
But what about that huge softmax over *V*?

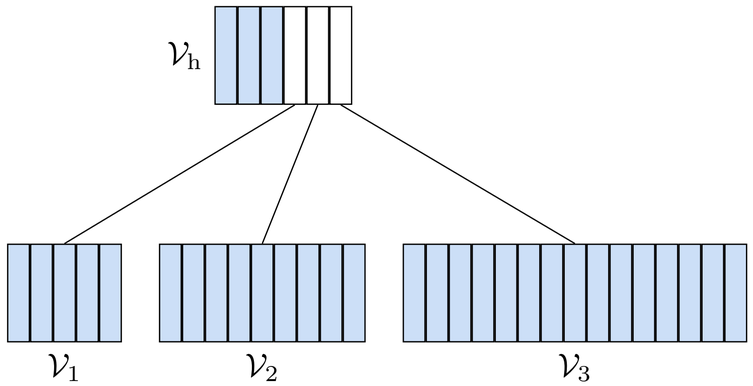
- Same problem as word embeddings: don't want to score wrt entire vocab!
- Solutions:
  - Hierarchical softmax
  - Noise-contrastive estimation (NCE)



But what about that huge softmax over *V*?

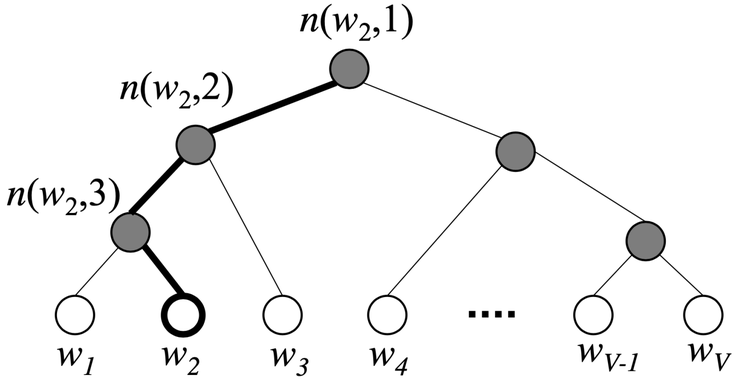
- Same problem as word embeddings: don't want to score wrt entire vocab!
- Solutions:
  - Hierarchical softmax
  - Noise-contrastive estimation (NCE)
  - Adaptive softmax [Grave et al. 2017]

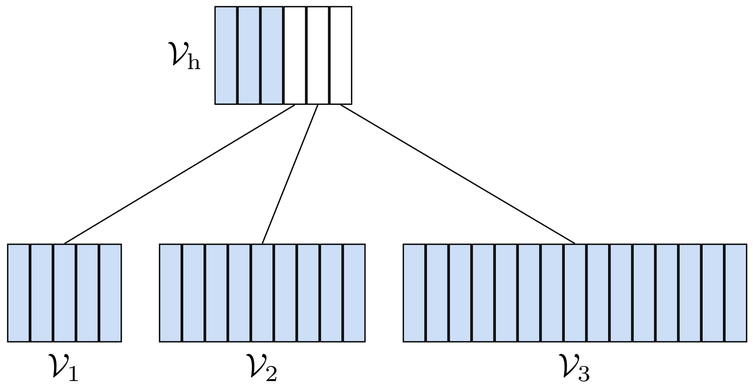




But what about that huge softmax over *V*?

- Same problem as word embeddings: don't want to score wrt entire vocab!
- Solutions:
  - Hierarchical softmax
  - Noise-contrastive estimation (NCE)
  - Adaptive softmax [Grave et al. 2017]





	ppl	training time
full softmax	144	83 min
sampling	166	41 min
HSM (freq)	166	34 min
HSM (sim)	155	41 min
D-softmax	195	53 min
D-softmax [*]	147	54 min
Ours	147	30 min

Regularization in RNNs

Figure: Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR, 2014.

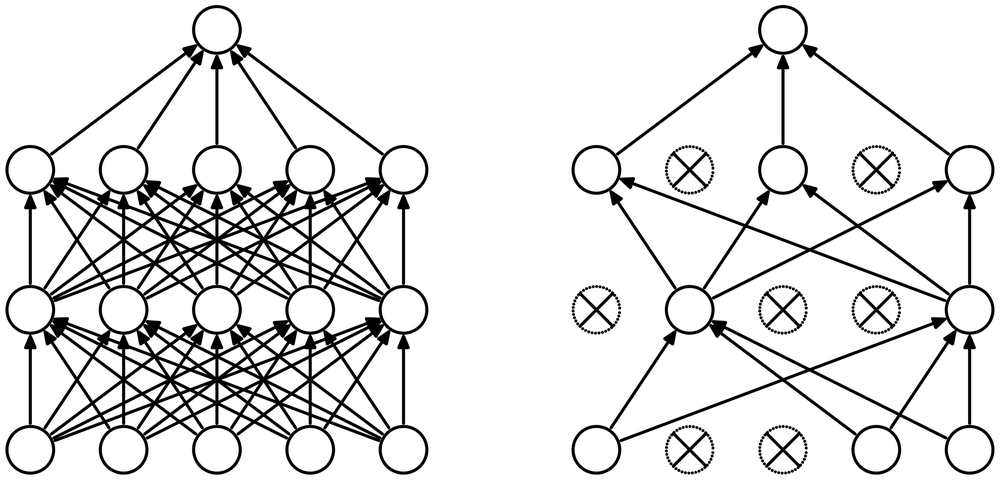
Regularization in RNNs

- Can use ℓ_1 and ℓ_2 regularization in RNNs (and feed-forward networks).

Figure: Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR, 2014.

Regularization in RNNs

- Can use ℓ_1 and ℓ_2 regularization in RNNs (and feed-forward networks).
- More common: **dropout** regularization:



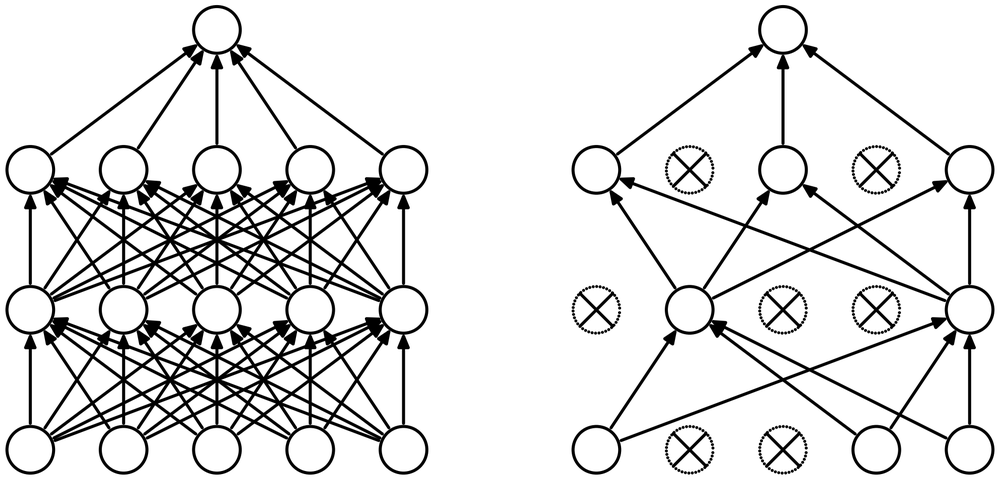
(a) Standard Neural Net

(b) After applying dropout.

Figure: Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR, 2014.

Regularization in RNNs

- Can use ℓ_1 and ℓ_2 regularization in RNNs (and feed-forward networks).
- More common: **dropout** regularization:
  - During training, set each activation/neuron to 0 with probability *p*.



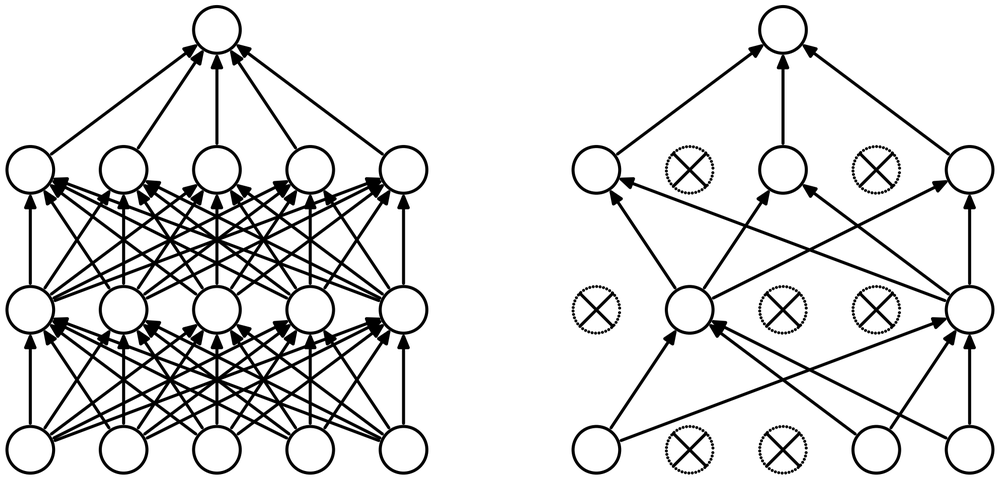
(a) Standard Neural Net

(b) After applying dropout.

Figure: Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR, 2014.

Regularization in RNNs

- Can use ℓ_1 and ℓ_2 regularization in RNNs (and feed-forward networks).
- More common: **dropout** regularization:
  - During training, set each activation/neuron to 0 with probability *p.*
  - During inference, use all activations, but normalize by 1/*p*



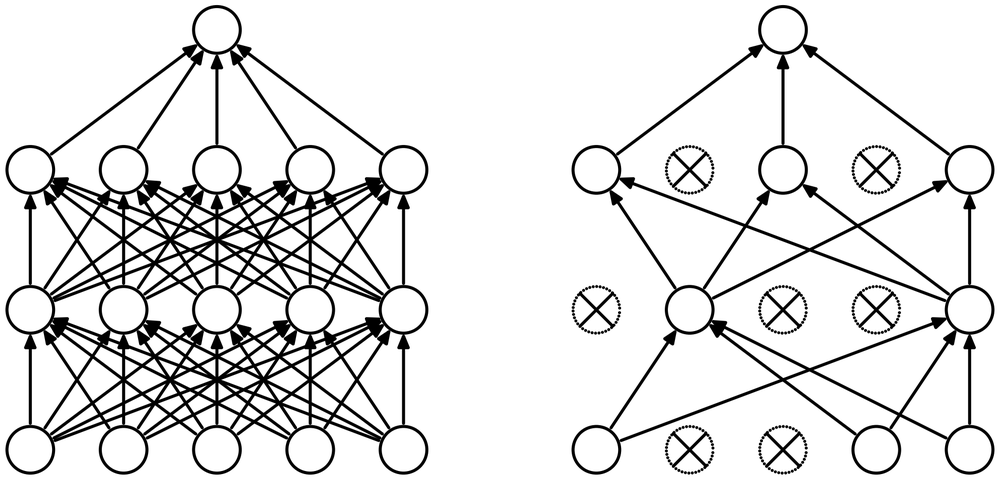
(a) Standard Neural Net

(b) After applying dropout.

Figure: Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR, 2014.

Regularization in RNNs

- Can use ℓ_1 and ℓ_2 regularization in RNNs (and feed-forward networks).
- More common: **dropout** regularization:
  - During training, set each activation/neuron to 0 with probability *p.*
  - During inference, use all activations, but normalize by 1/*p*
  - *p* = 0.1 – 0.5 in practice.



(a) Standard Neural Net

(b) After applying dropout.

Figure: Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR, 2014.

Problems learning RNNs

Figure: Goodfellow, Bengio and Courville. Deep Learning. MIT Press, 2016.

Problems learning RNNs

- In theory, should be able to propagate information over arbitrarily long contexts.

Figure: Goodfellow, Bengio and Courville. Deep Learning. MIT Press, 2016.

Problems learning RNNs

- In theory, should be able to propagate information over arbitrarily long contexts.
- In practice, RNNs suffer from **vanishing gradients** that decay to 0, or **exploding gradients** that increase towards infinity.

Figure: Goodfellow, Bengio and Courville. Deep Learning. MIT Press, 2016.

Problems learning RNNs

- In theory, should be able to propagate information over arbitrarily long contexts.
- In practice, RNNs suffer from **vanishing gradients** that decay to 0, or **exploding gradients** that increase towards infinity.
  - Exploding gradients mostly resolved by **gradient clipping**: thresholding gradient values, or rescaling them. Threshold/scale is a hyperparameter.

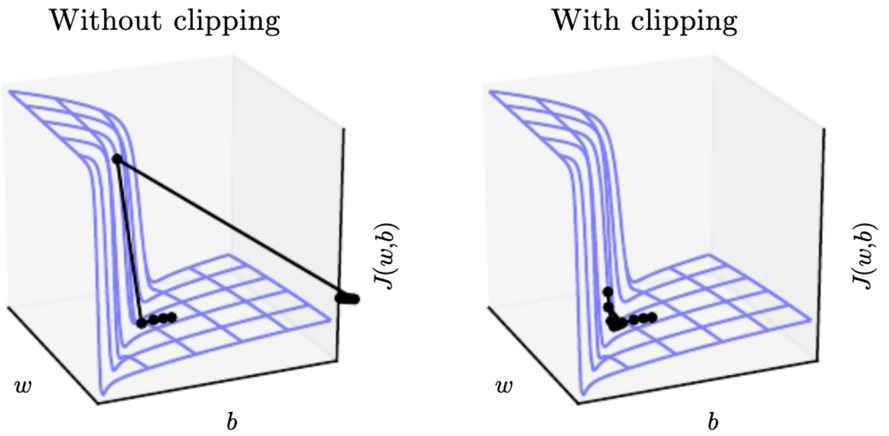


Figure: Goodfellow, Bengio and Courville. Deep Learning. MIT Press, 2016.

Problems learning RNNs

- In theory, should be able to propagate information over arbitrarily long contexts.
- In practice, RNNs suffer from **vanishing gradients** that decay to 0, or **exploding gradients** that increase towards infinity.
  - Exploding gradients mostly resolved by **gradient clipping**: thresholding gradient values, or rescaling them. Threshold/scale is a hyperparameter.
- Vanishing gradients mostly resolved by adding **gating** to the RNN composition function.

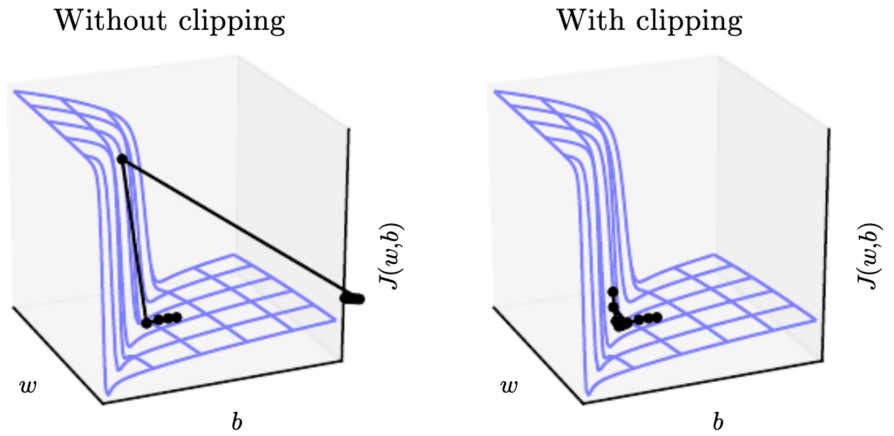


Figure: Goodfellow, Bengio and Courville. Deep Learning. MIT Press, 2016.

Long short term memory: LSTMs

Long short term memory: LSTMs

- Adds a *memory cell* c_m and three gates: input *i*, output *o*, and forget *f*

Long short term memory: LSTMs

- Adds a *memory cell* c_m and three gates: input i, output o, and forget f

- Gates are functions of the input and previous hidden state, in range [0, 1], applied element-wise w/ Hadamard $\odot$

$$f_{m+1} = \sigma(\Theta^{(h \to f)} h_m + \Theta^{(x \to f)} x_{m+1} + b_f)$$

$$i_{m+1} = \sigma(\Theta^{(h \to i)} h_m + \Theta^{(x \to i)} x_{m+1} + b_i)$$

$$o_{m+1} = \sigma(\Theta^{(h \to o)} h_m + \Theta^{(x \to o)} x_{m+1} + b_o)$$

Long short term memory: LSTMs

- Adds a *memory cell* c_m and three gates: input i, output o, and forget f

- Gates are functions of the input and previous hidden state, in range [0, 1], applied element-wise w/ Hadamard $\odot$

- Memory cell is updated as a gated sum of its previous value, and an update $\tilde{c}$, which is computed from input and previous hidden, and squashed.

$$f_{m+1} = \sigma(\Theta^{(h \to f)} h_m + \Theta^{(x \to f)} x_{m+1} + b_f)$$

$$i_{m+1} = \sigma(\Theta^{(h \to i)} h_m + \Theta^{(x \to i)} x_{m+1} + b_i)$$

$$o_{m+1} = \sigma(\Theta^{(h \to o)} h_m + \Theta^{(x \to o)} x_{m+1} + b_o)$$

$$c_{m+1} = f_{m+1} \odot c_m + i_{m+1} \odot \tilde{c}_{m+1}$$

$$\tilde{c}_{m+1} = \tanh(\Theta^{(h \to c)} h_m + \Theta^{(w \to c)} x_{m+1})$$

Long short term memory: LSTMs

- Adds a *memory cell* c_m and three gates: input i, output o, and forget f

$$f_{m+1} = \sigma(\Theta^{(h\to f)} h_m + \Theta^{(x\to f)} x_{m+1} + b_f)$$

- Gates are functions of the input and previous hidden state, in range [0, 1], applied element-wise w/ Hadamard $\odot$

$$i_{m+1} = \sigma(\Theta^{(h\to i)} h_m + \Theta^{(x\to i)} x_{m+1} + b_i)$$

$$o_{m+1} = \sigma(\Theta^{(h\to o)} h_m + \Theta^{(x\to o)} x_{m+1} + b_o)$$

- Memory cell is updated as a gated sum of its previous value, and an update $\tilde{c}$, which is computed from input and previous hidden, and squashed.

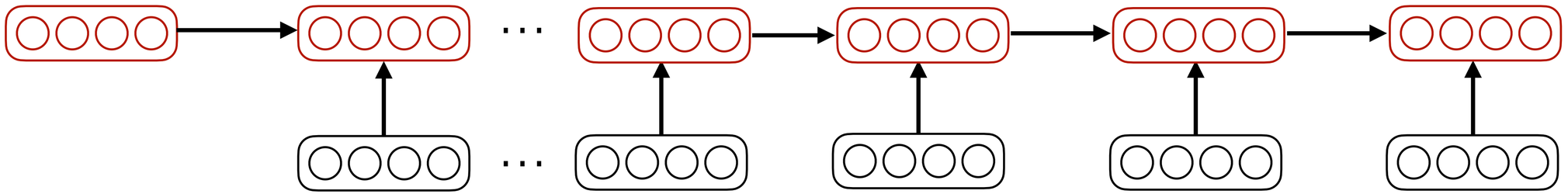
$$c_{m+1} = f_{m+1} \odot c_m + i_{m+1} \odot \tilde{c}_{m+1}$$

$$\tilde{c}_{m+1} = \tanh(\Theta^{(h\to c)} h_m + \Theta^{(w\to c)} x_{m+1})$$

- Next hidden state is a combination of output and memory cell.

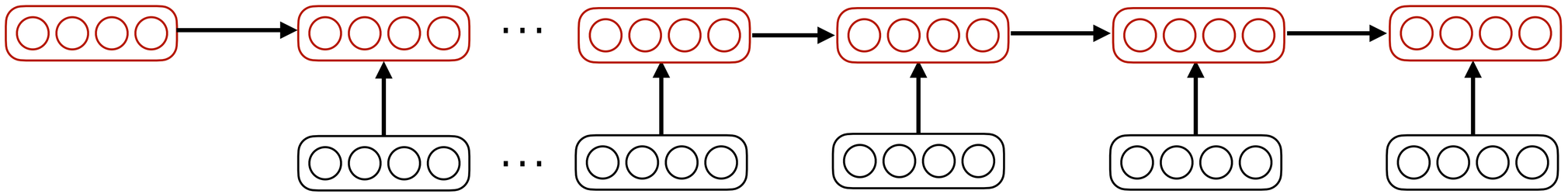
$$h_{m+1} = o_{m+1} \cdot \tanh(c_{m+1})$$

Stacking RNNs



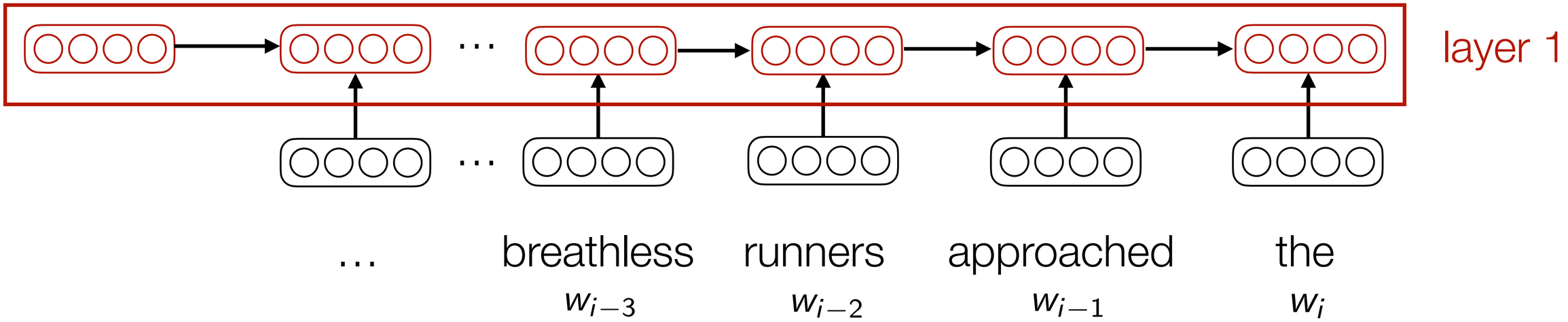
Stacking RNNs

- Just as we can add many hidden layers to feed-forward networks, can also add many RNN layers: **stacked RNNs**



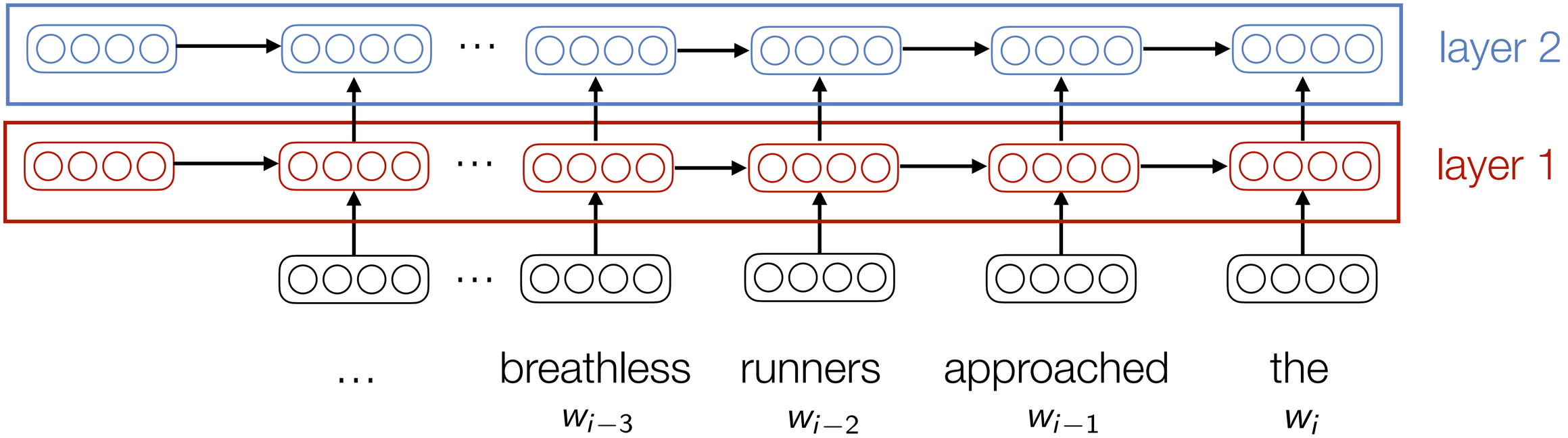
Stacking RNNs

- Just as we can add many hidden layers to feed-forward networks, can also add many RNN layers: **stacked RNNs**



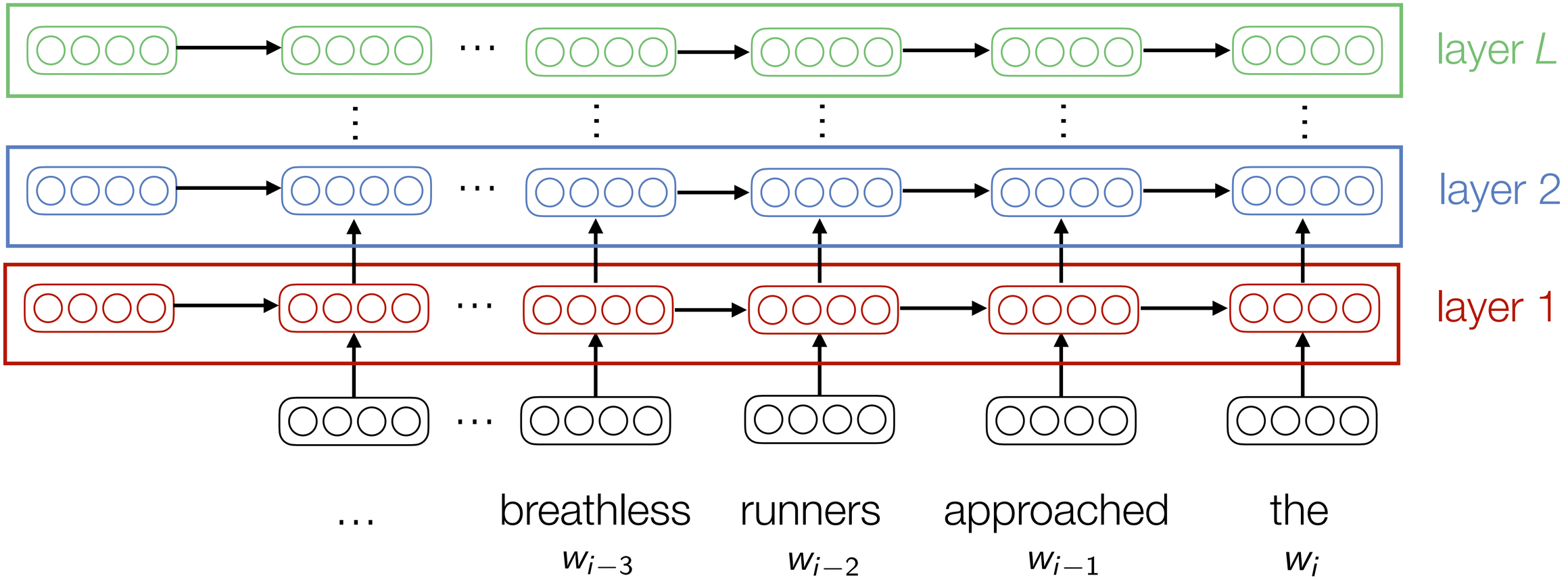
Stacking RNNs

- Just as we can add many hidden layers to feed-forward networks, can also add many RNN layers: **stacked RNNs**



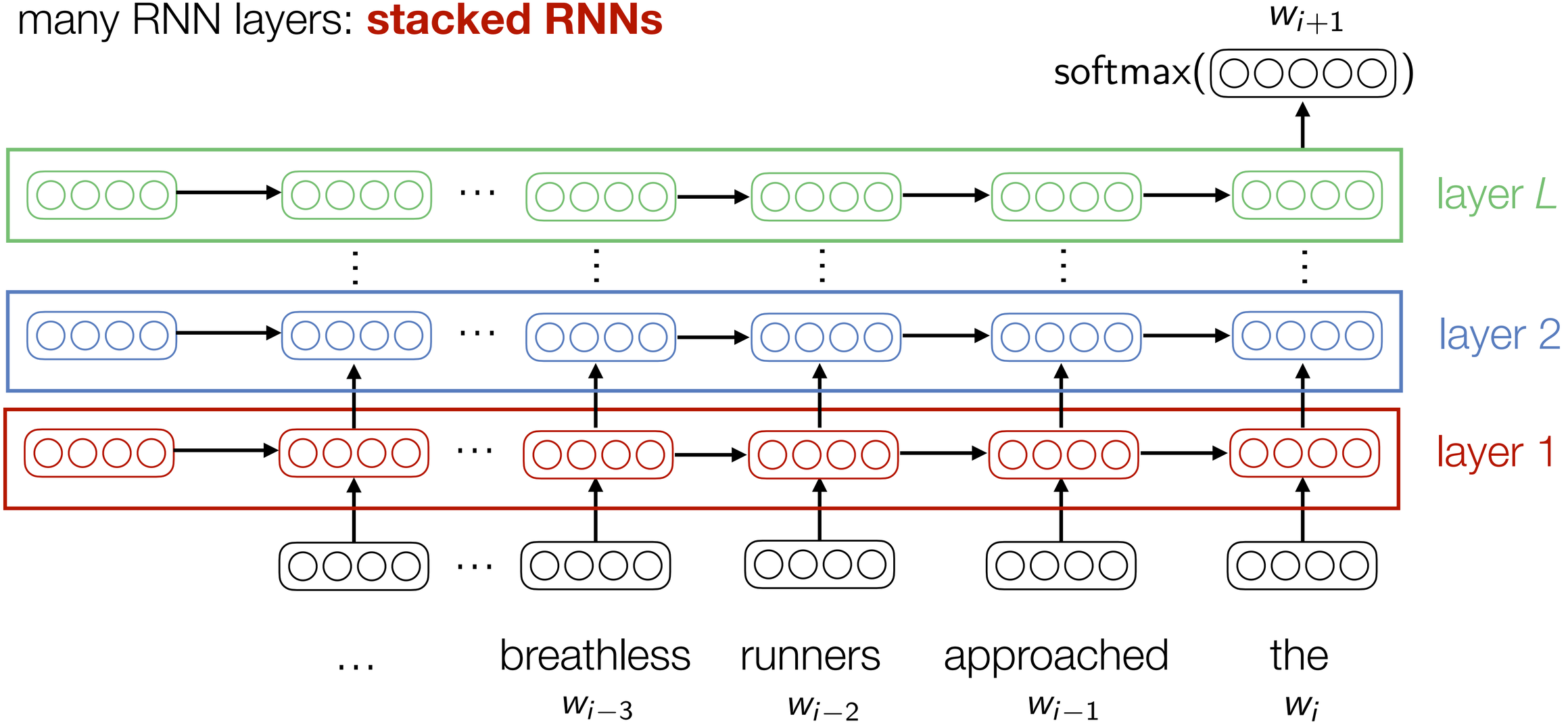
Stacking RNNs

- Just as we can add many hidden layers to feed-forward networks, can also add many RNN layers: **stacked RNNs**

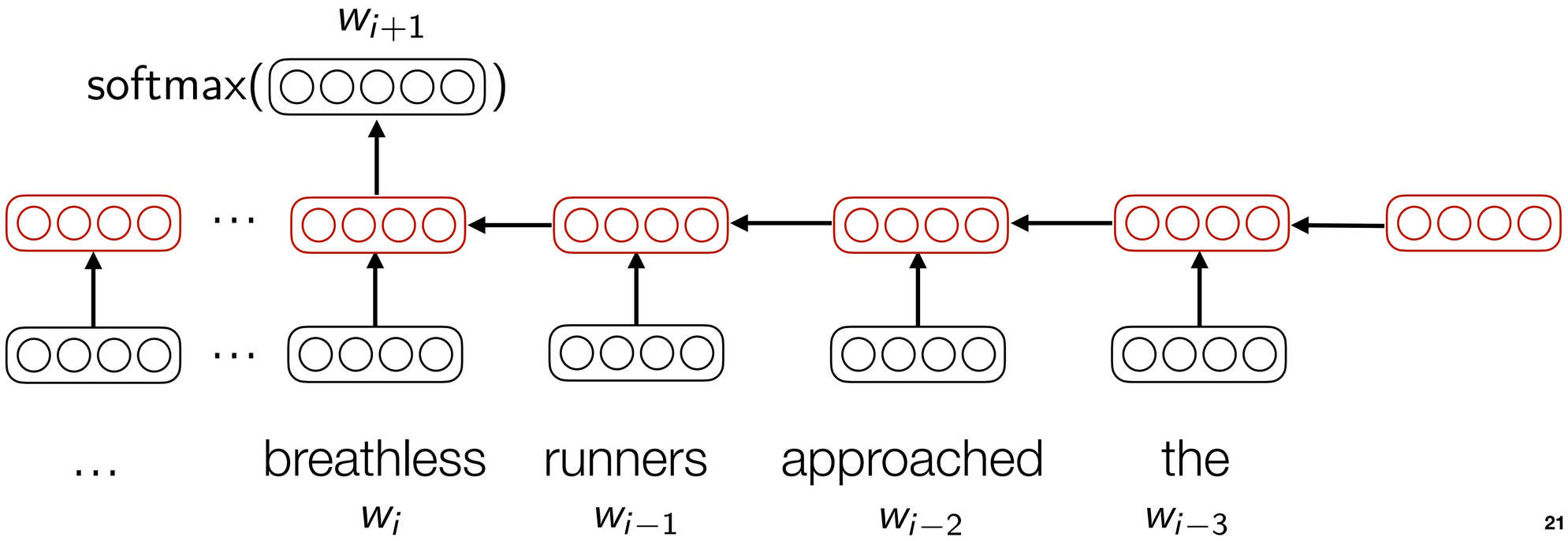


Stacking RNNs

- Just as we can add many hidden layers to feed-forward networks, can also add many RNN layers: **stacked RNNs**

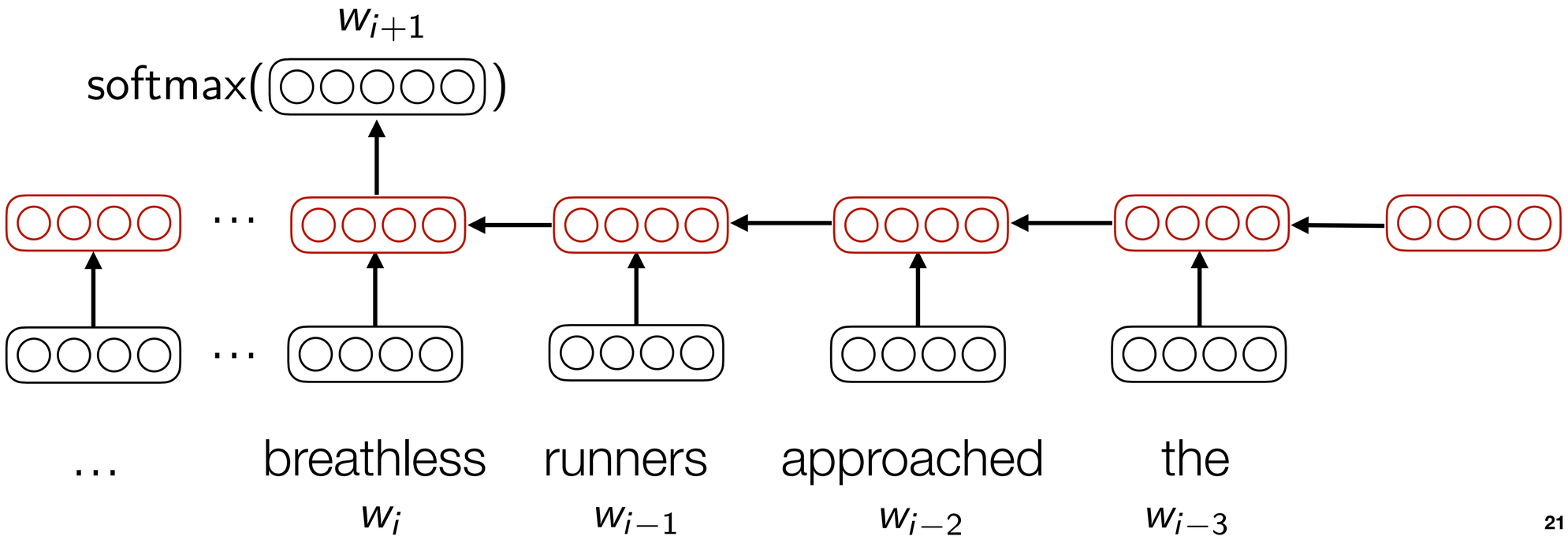


Bidirectional RNNs



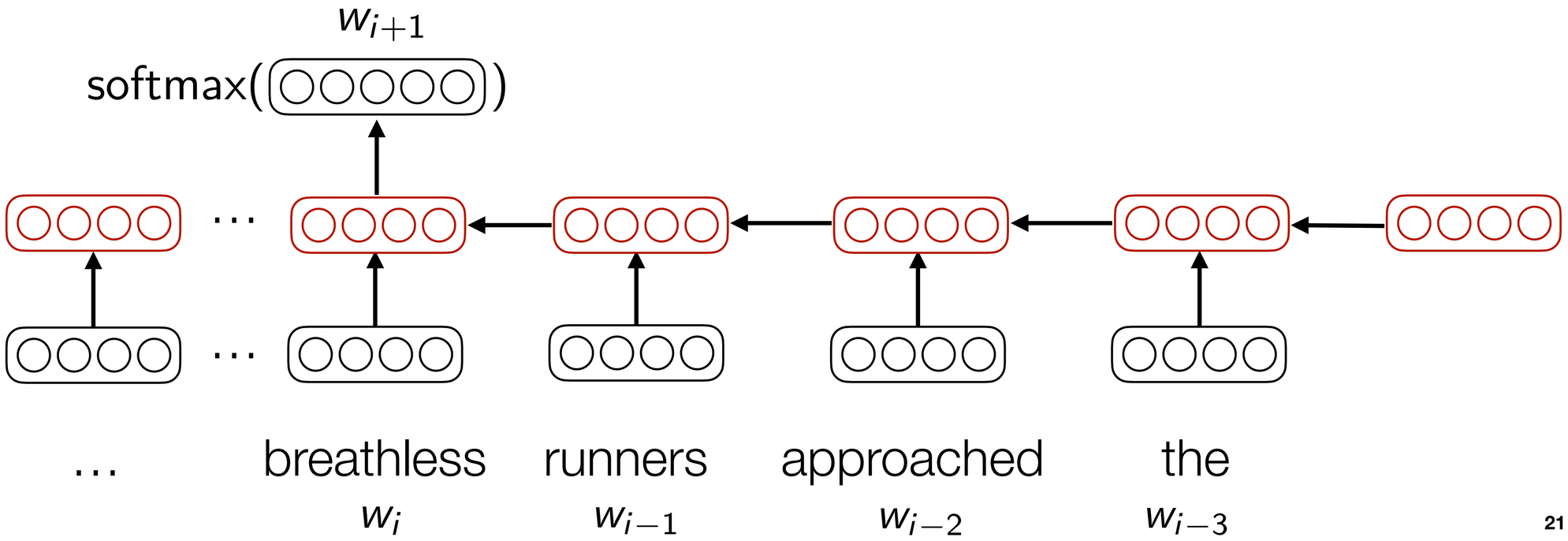
Bidirectional RNNs

- In language it's also often useful to model past *and future* context



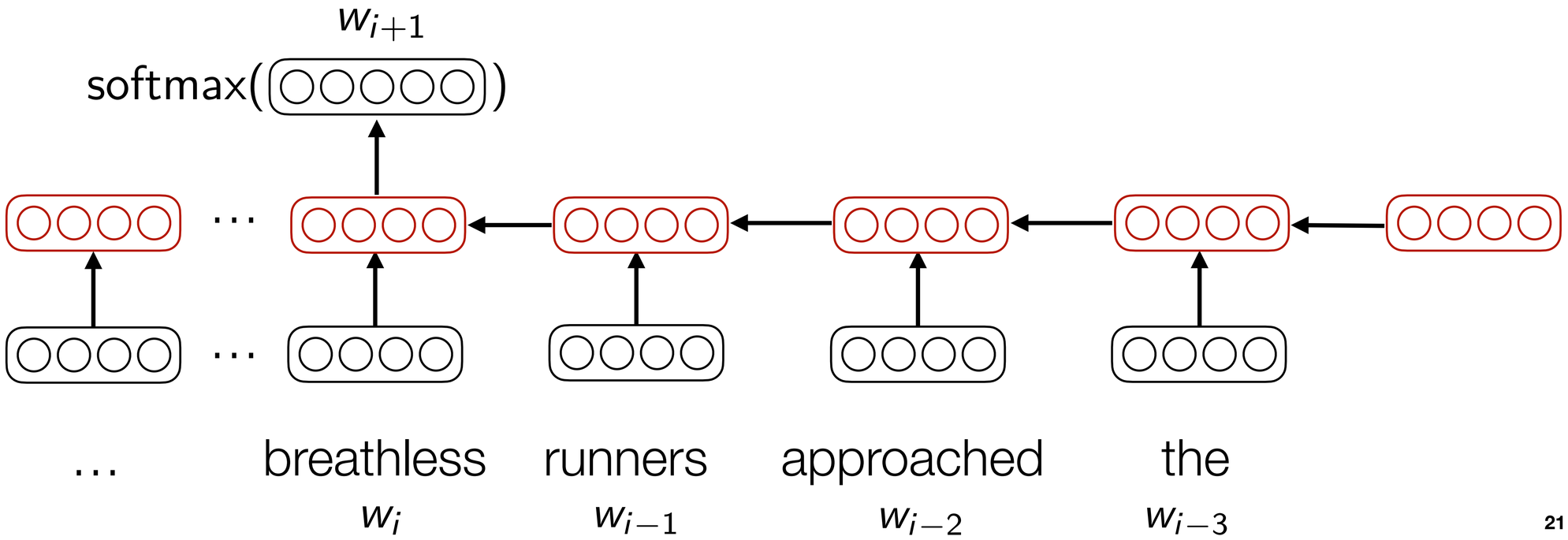
Bidirectional RNNs

- In language it's also often useful to model past *and future* context
  - Can also run an RNN in the backward direction (reverse reading order)

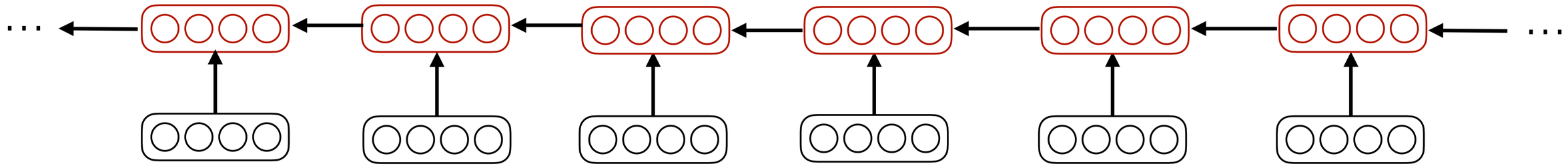


Bidirectional RNNs

- In language it's also often useful to model past *and future* context
  - Can also run an RNN in the backward direction (reverse reading order)
  - Combining both directions works best!

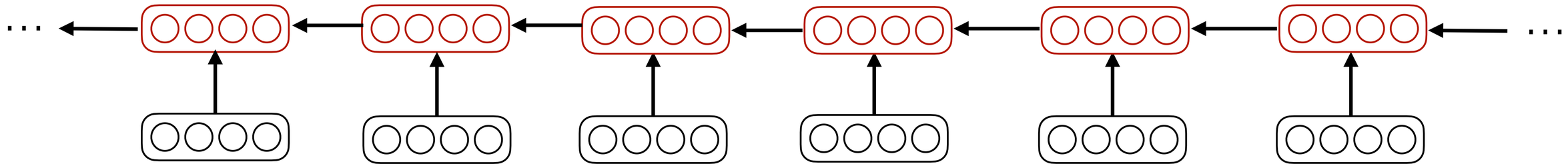


Bidirectional RNNs



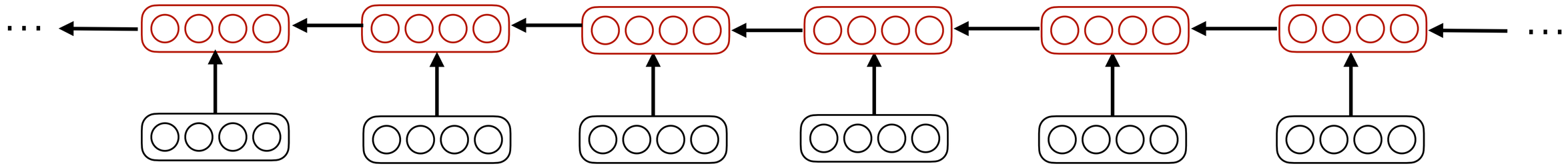
Bidirectional RNNs

- In language it's also often useful to model past *and future* context



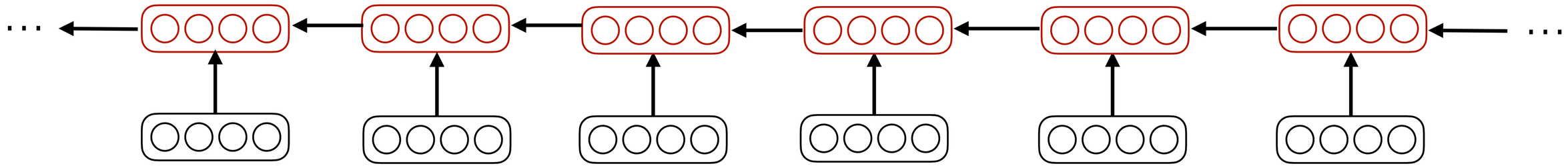
Bidirectional RNNs

- In language it's also often useful to model past *and future* context
  - Can also run an RNN in the backward direction (reverse reading order)



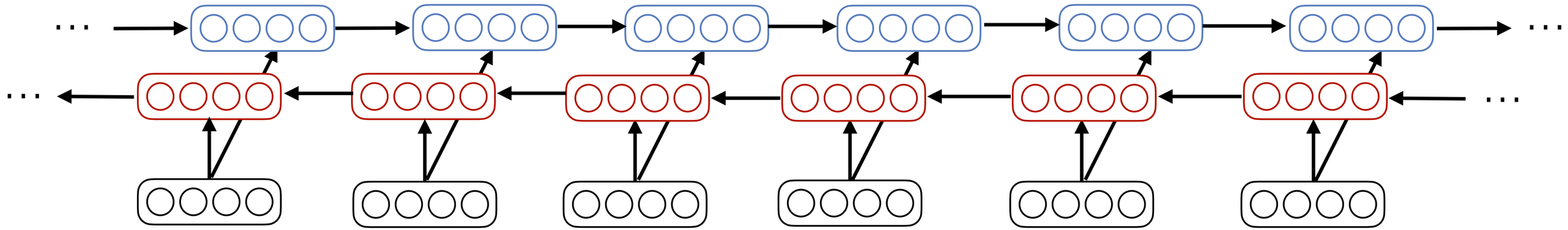
Bidirectional RNNs

- In language it's also often useful to model past *and future* context
  - Can also run an RNN in the backward direction (reverse reading order)
  - Combining both directions works best!



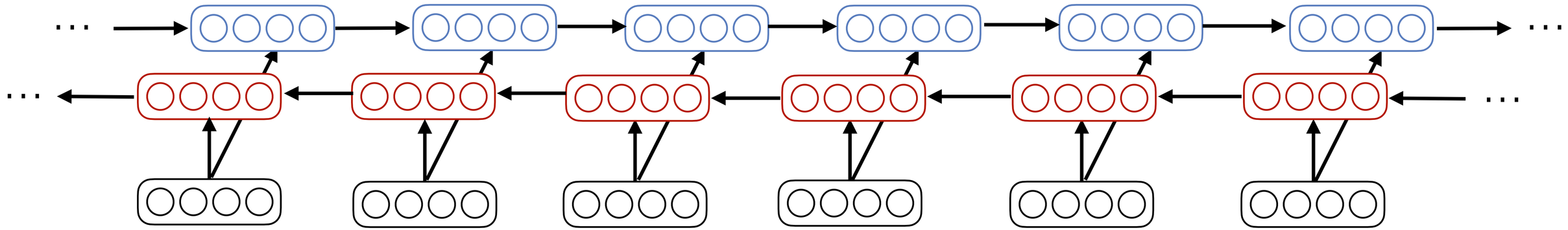
Bidirectional RNNs

- In language it's also often useful to model past *and future* context
  - Can also run an RNN in the backward direction (reverse reading order)
  - Combining both directions works best!



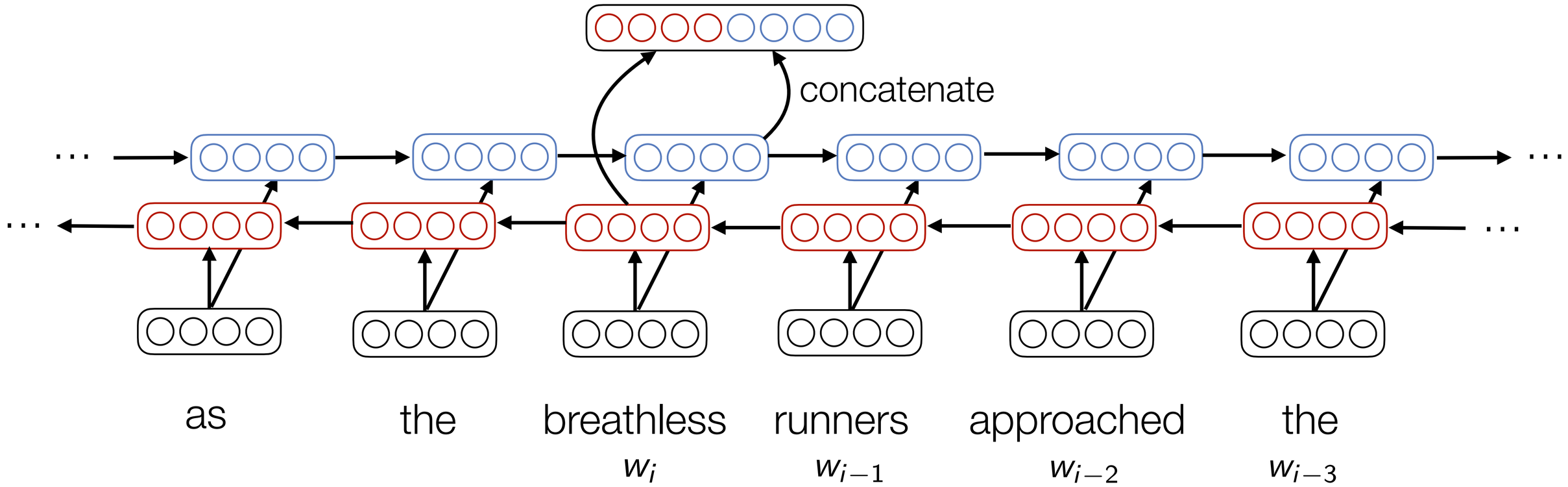
Bidirectional RNNs

- In language it's also often useful to model past *and future* context. Combining both directions works best!



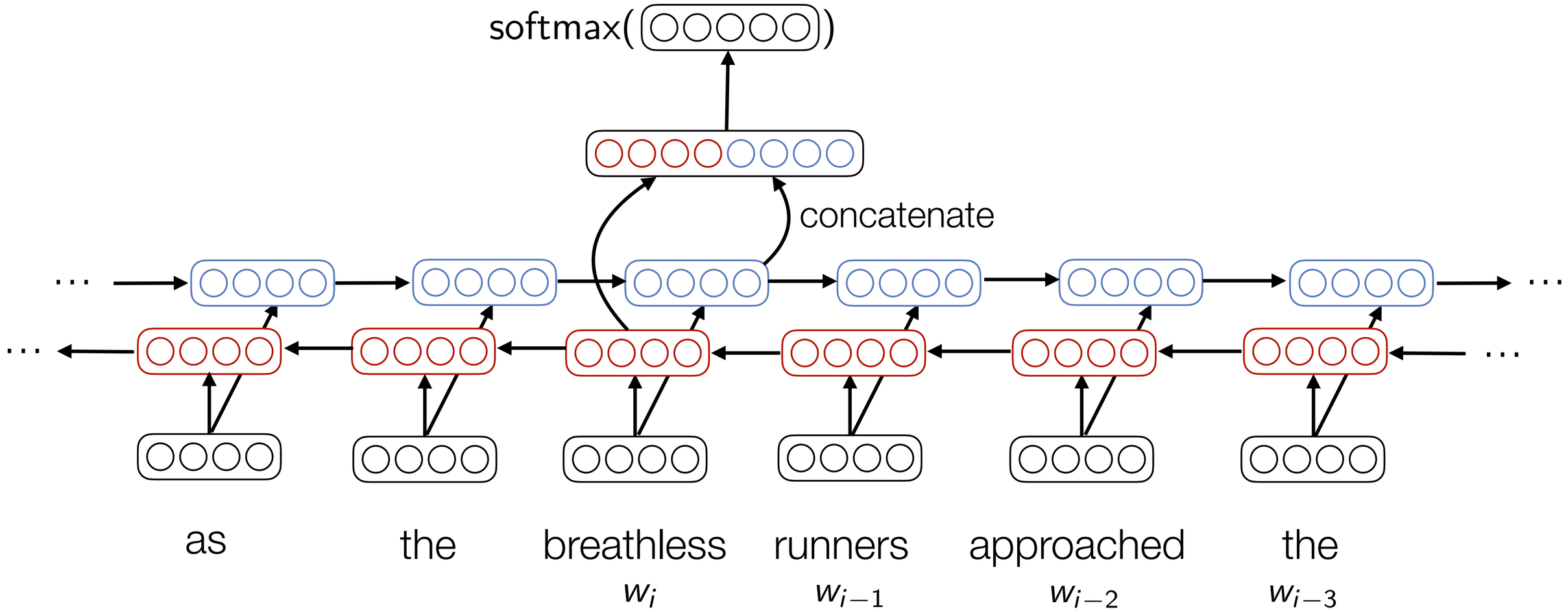
Bidirectional RNNs

- In language it's also often useful to model past *and future* context. Combining both directions works best!



Bidirectional RNNs

- In language it's also often useful to model past *and future* context. Combining both directions works best!



Dealing with unseen words

Dealing with unseen words

- As in n-gram language models, simplest way to deal with unseen words is to add an OOV token / word embedding.

Dealing with unseen words

- As in n-gram language models, simplest way to deal with unseen words is to add an OOV token / word embedding.
    - As an alternative to fixing the vocabulary, OOV embedding parameters can be learned using **word dropout**: replace entire words with OOV with probability p.

Dealing with unseen words

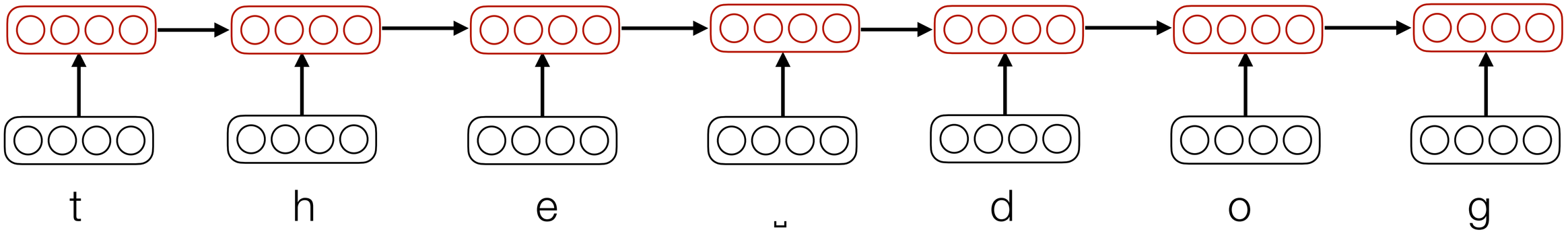
- As in n-gram language models, simplest way to deal with unseen words is to add an OOV token / word embedding.
    - As an alternative to fixing the vocabulary, OOV embedding parameters can be learned using **word dropout**: replace entire words with OOV with probability p.
- Even better, can we eliminate the OOV problem altogether?

Dealing with unseen words

- As in n-gram language models, simplest way to deal with unseen words is to add an OOV token / word embedding.
    - As an alternative to fixing the vocabulary, OOV embedding parameters can be learned using **word dropout**: replace entire words with OOV with probability p.
- Even better, can we eliminate the OOV problem altogether?
- Yes, using character- or byte-level models

Dealing with unseen words

- As in n-gram language models, simplest way to deal with unseen words is to add an OOV token / word embedding.
  - As an alternative to fixing the vocabulary, OOV embedding parameters can be learned using **word dropout**: replace entire words with OOV with probability p.
- Even better, can we eliminate the OOV problem altogether?
- Yes, using character- or byte-level models

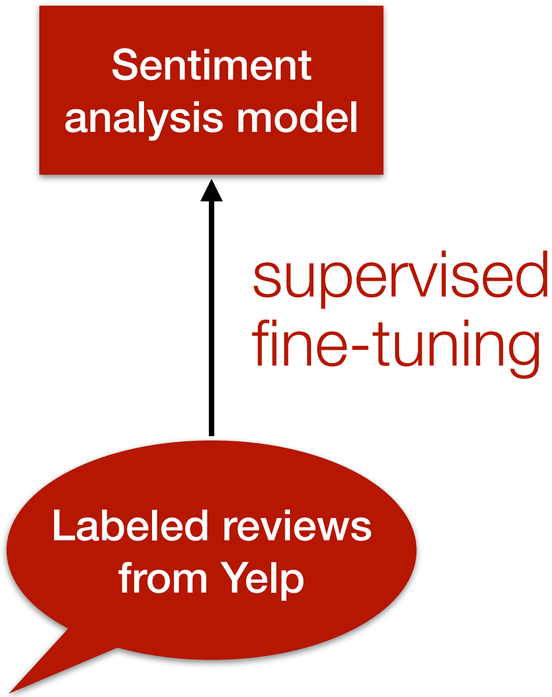


Transfer learning from language models

- Let’s say I want to train a model for sentiment analysis.

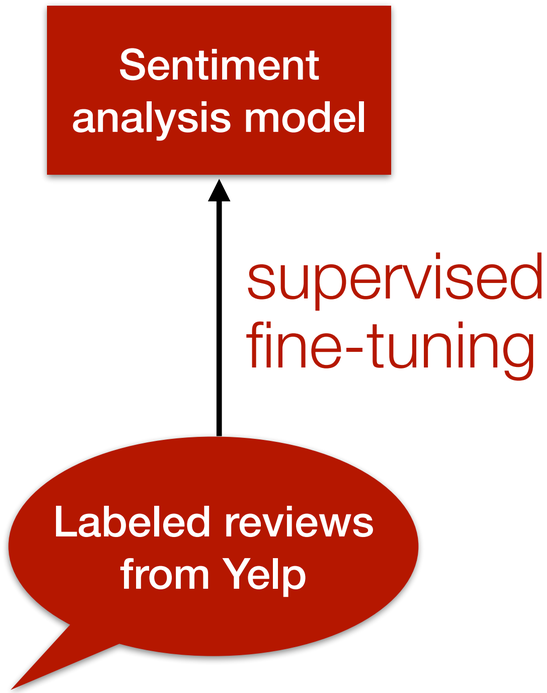
Transfer learning from language models

- Let's say I want to train a model for sentiment analysis.
- The old way: train the model using supervised learning on a corpus of labeled examples, e.g. Yelp reviews, IMDB movie reviews



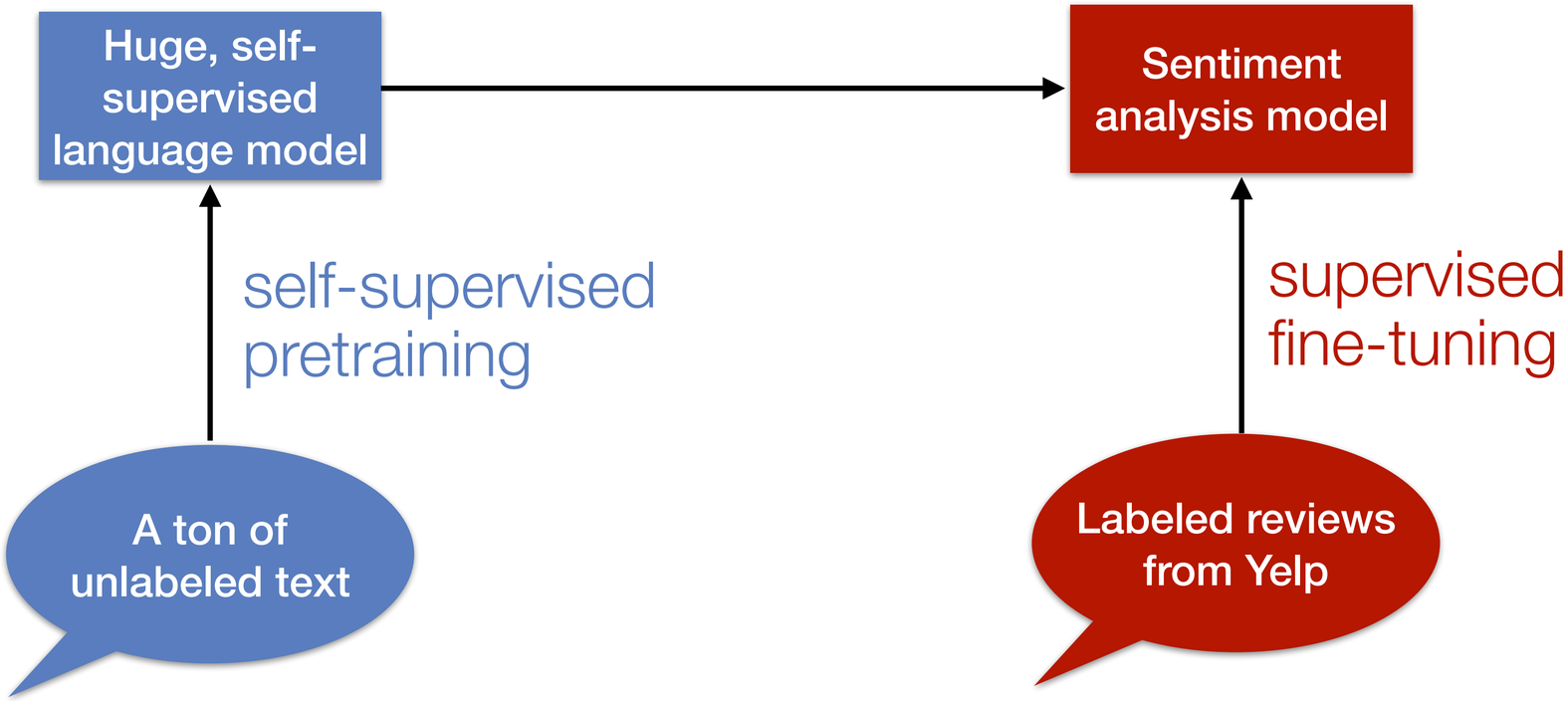
Transfer learning from language models

- Let's say I want to train a model for sentiment analysis.



Transfer learning from language models

- Let's say I want to train a model for sentiment analysis.
- The new way: **transfer learning** from a large language model trained using a self-supervised objective on a massive amount of unlabled text.



Language models as word representations

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

- Multiple senses entangled:

play = | -1.36 | -0.9 | 0.71 | -0.22 | 0.77 | -1.36 | -0.72 | 0.71 | 0.15 | ...

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

- Multiple senses entangled:

play = | -1.36 | -0.9 | 0.71 | -0.22 | 0.77 | -1.36 | -0.72 | 0.71 | 0.15 | ...

nearest neighbors:

playing	played	Play
game	plays	football
games	player	multiplayer

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

- Multiple senses entangled:

play = | -1.36 | -0.9 | 0.71 | -0.22 | 0.77 | -1.36 | -0.72 | 0.71 | 0.15 | ...

nearest neighbors:

playing	played	Play	verb
game	plays	football	
games	player	multiplayer	

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

- Multiple senses entangled:

play = | -1.36 | -0.9 | 0.71 | -0.22 | 0.77 | -1.36 | -0.72 | 0.71 | 0.15 | ...

nearest neighbors:

playing	played	Play	verb
game	plays	football	noun
games	player	multiplayer	

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

- Multiple senses entangled:

play = | -1.36 | -0.9 | 0.71 | -0.22 | 0.77 | -1.36 | -0.72 | 0.71 | 0.15 | ...

nearest neighbors:

playing	played	Play
game	plays	football
games	player	multiplayer

verb
noun
adjective

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

- Instead, want **contextualized** word embeddings: token representations that differ depending on the context.

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

- Instead, want **contextualized** word embeddings: token representations that differ depending on the context.
  - But, still want to leverage self-supervised training on massive data.

Language models as word representations

- A problem with (most) word2vec-style word embeddings: single representation per word type.

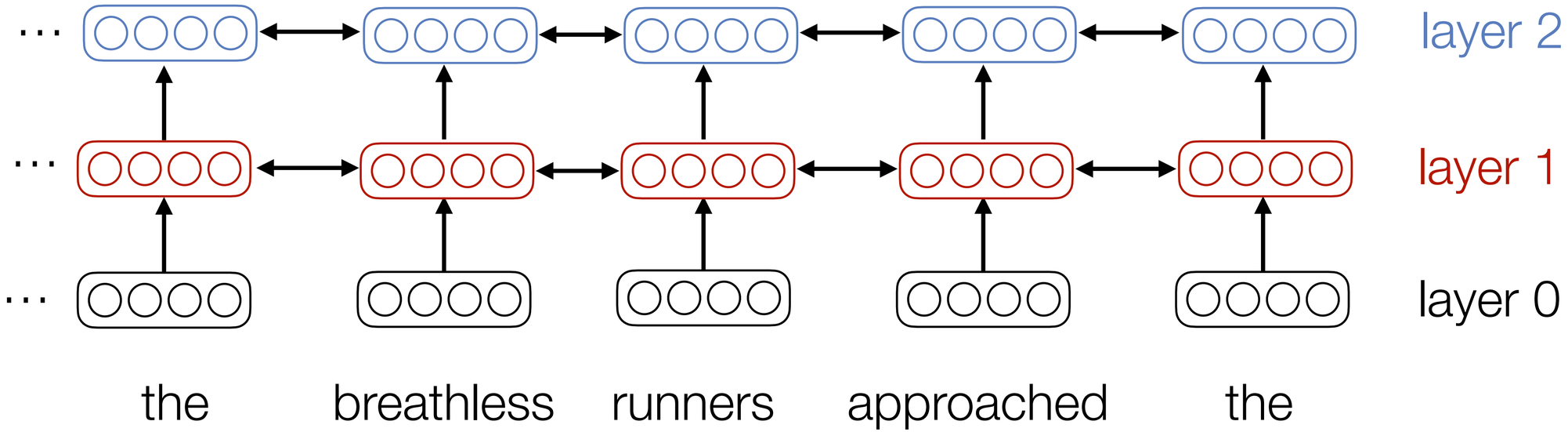
the new-look **play** area is due to be completed by early spring 2020

gerrymandered congressional districts favor representatives who **play** to the party base

the freshman then completed the three-point **play** for a 66-63 lead

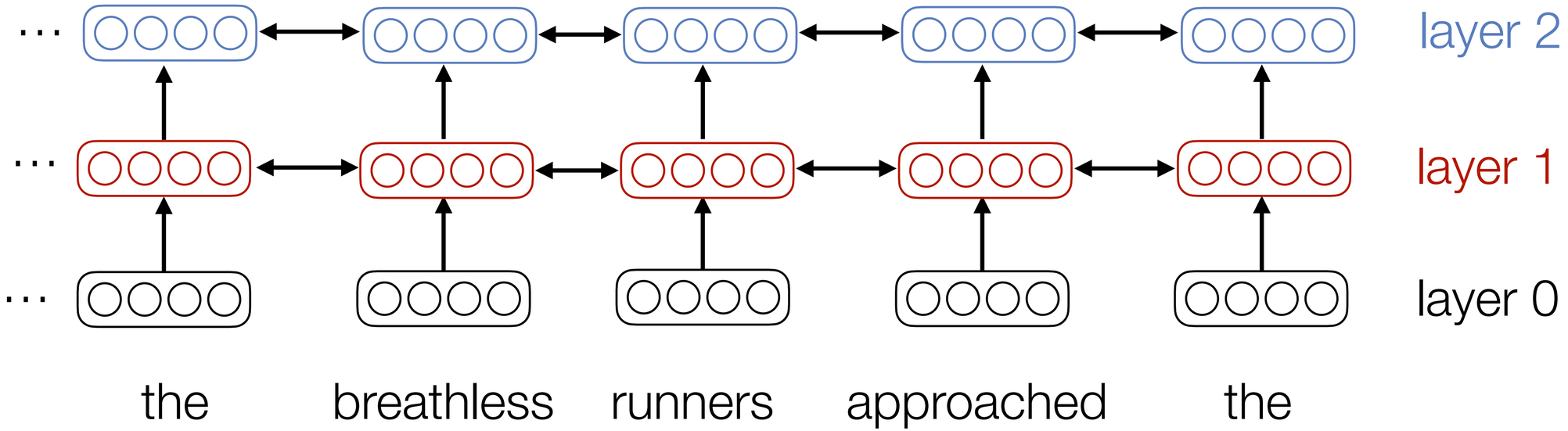
- Instead, want **contextualized** word embeddings: token representations that differ depending on the context.
  - But, still want to leverage self-supervised training on massive data.
  - Solution: use hidden representations from a neural language model.

ELMo: Deep contextualized word embeddings



Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

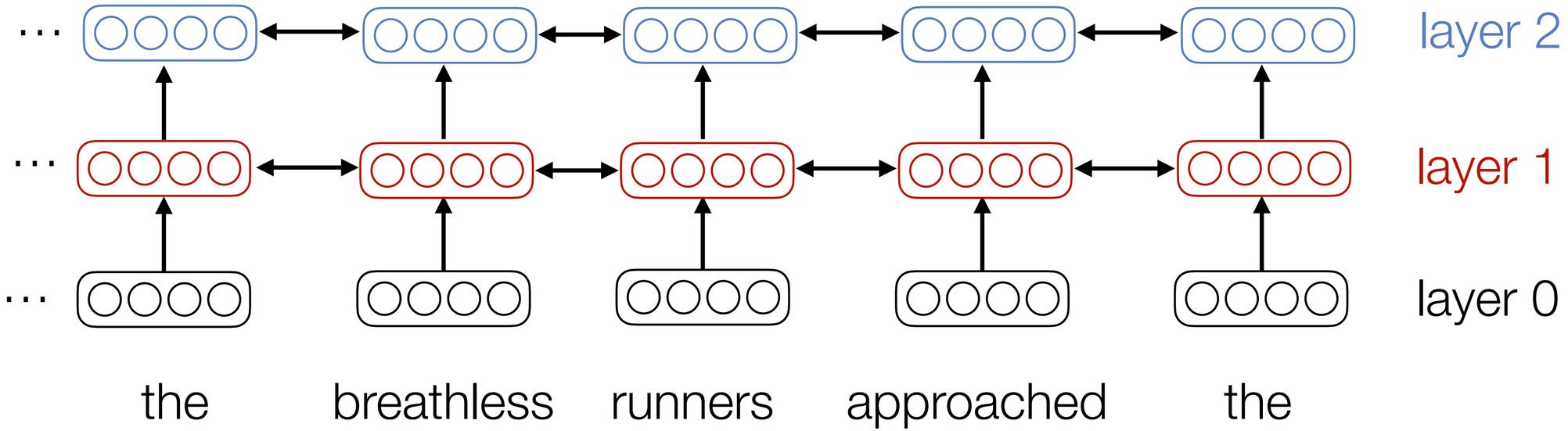


stacked bidirectional LSTM LM

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Key idea: context-dependent embedding for each word interpolates representations for that word from each layer

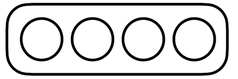


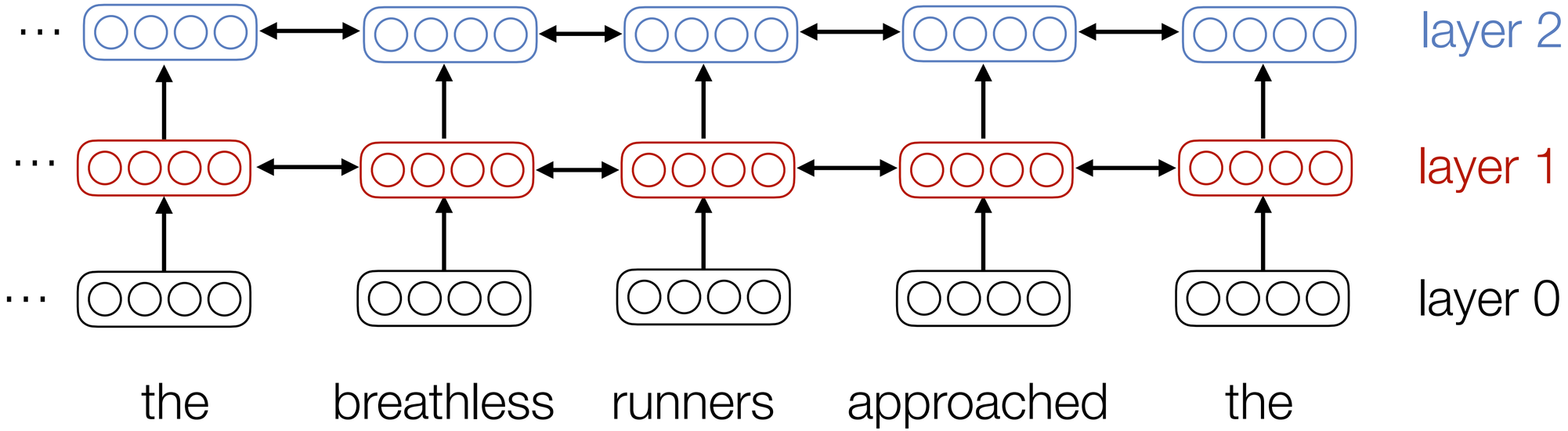
stacked bidirectional LSTM LM

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Key idea: context-dependent embedding for each word interpolates representations for that word from each layer

breathless = γ **(** $s_1 \cdot$  **+** $s_2 \cdot$  **+** $s_3 \cdot$  **)**



stacked bidirectional LSTM LM

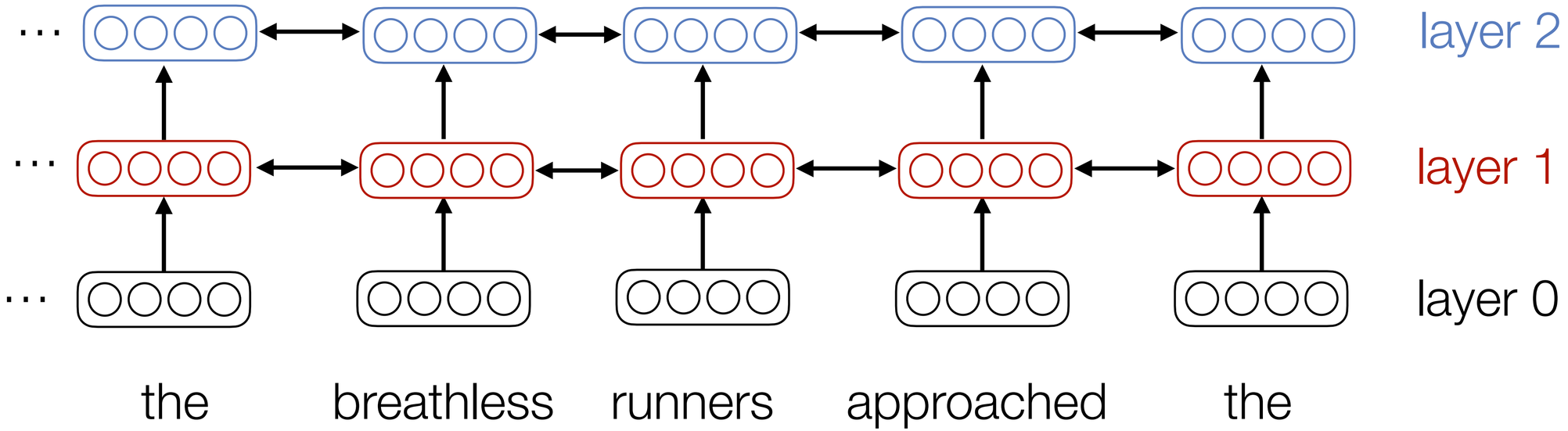
Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Key idea: context-dependent embedding for each word interpolates representations for that word from each layer

breathless = γ **(** $s_1 \cdot$ ◯◯◯◯ **+** $s_2 \cdot$ ◯◯◯◯ **+** $s_3 \cdot$ ◯◯◯◯ **)**

$$\sum_{j=1}^{L} s_j = 1$$



stacked bidirectional LSTM LM

Peters et al. Deep contextualized word representations. NAACL 2018.

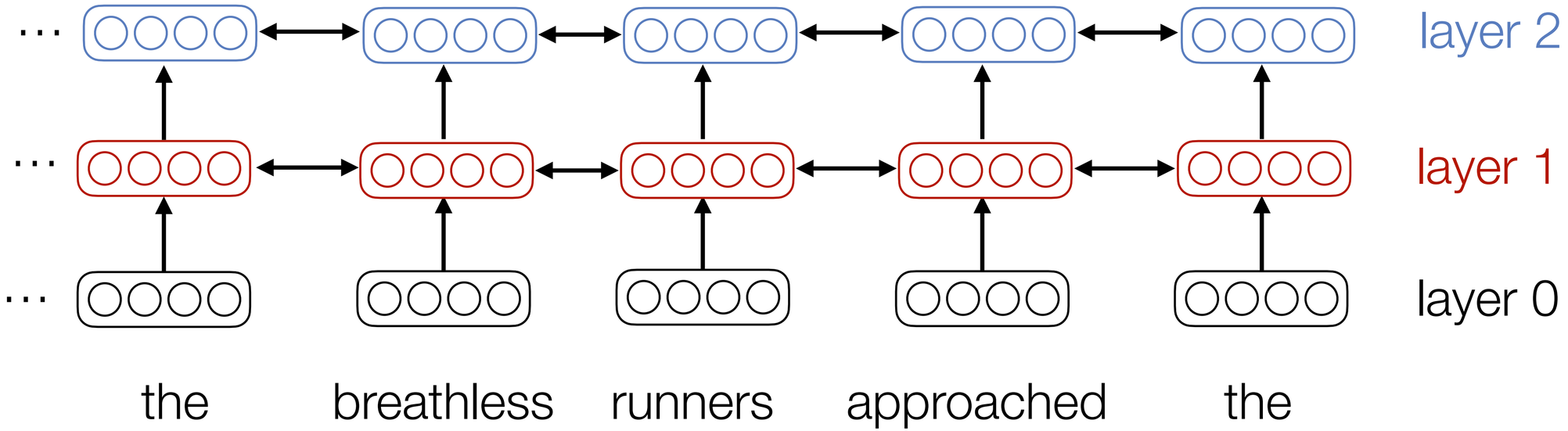
ELMo: Deep contextualized word embeddings

- Key idea: context-dependent embedding for each word interpolates representations for that word from each layer

breathless = γ **(** $s_1 \cdot$ ⬭ **+** $s_2 \cdot$ ⬭ **+** $s_3 \cdot$ ⬭ **)**

$$\sum_{j=1}^{L} s_j = 1$$

- Interpolation weights are task-specific (fine-tuned on supervised data.)



stacked bidirectional LSTM LM

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

question answering

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

	Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
question answering	SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
natural language inference	SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
	SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
	Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
	NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
	SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

	Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
question answering	SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
natural language inference	SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
semantic role labeling	SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
	Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
	NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
	SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

	Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
question answering	SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
natural language inference	SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
semantic role labeling	SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
coreference	Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
	NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
	SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

	Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
question answering	SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
natural language inference	SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
semantic role labeling	SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
coreference	Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
named entity recognition	NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
	SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

	Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
question answering	SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
natural language inference	SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
semantic role labeling	SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
coreference	Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
named entity recognition	NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
sentiment analysis	SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Peters et al. Deep contextualized word representations. NAACL 2018.

ELMo: Deep contextualized word embeddings

- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

	Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
question answering	SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
natural language inference	SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
semantic role labeling	SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
coreference	Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
named entity recognition	NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
sentiment analysis	SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Peters et al. Deep contextualized word representations. NAACL 2018.

Announcements

- Recitation this Friday is a P1 Q&A with Han, with two optional quiz questions to work on even if you don’t have P1 questions, for participation credit.
- Some more readings posted for today’s lecture.