CS695-002 Special Topics in NLP

Language Modeling, Smoothing, and Recurrent Neural Networks

Antonis Anastasopoulos

https://cs.gmu.edu/~antonis/course/cs695-fall20/

Slides are taken from Graham Neubig's CMU NN4NLP course

Are These Sentences OK?

- Jane went to the store.
- store to Jane went the.
- Jane went store.
- Jane goed to the store.
- The store went to Jane.
- The food truck went to Jane.

Calculating the Probability of a Sentence

Jane went to the store .

$$P(X) = \prod_{i=1}^{n} P(x_i)$$

Calculating the Probability of a Sentence

Jane went to the store .

$$P(X) = \prod_{i=1}^{n} \underbrace{P(x_i)}_{\text{Unigram}}$$

Calculating the Probability of a Sentence

Jane went to the store .

$$P(X) = \prod_{i=1}^{n} \underbrace{P(x_i)}_{\text{Unigram}}$$

$$P(\text{Jane went to the store}) = P(Jane) \times P(went) \times P(to) \times P(the) \times P(store) \times P(\,.\,)\,.$$

Calculating the Probability of a Sentence

Jane went to the store .

$$P(X) = \prod_{i=1}^{n} \underbrace{P(x_i)}_{\text{Unigram}}$$

$$P(\text{Jane went to the store}) = P(Jane) \times P(went) \times P(to) \times P(the) \times P(store) \times P(\,.\,)\,.$$

But word order and context matters!

Calculating the Probability of a Sentence

$$P(X) = \prod_{i=1}^{I} P(\underbrace{x_i}_{\text{Next Word}} \mid \underbrace{x_1, \ldots, x_{i-1}}_{\text{Context}})$$

Calculating the Probability of a Sentence

$$P(X) = \prod_{i=1}^{I} P(\underbrace{x_i}_{\text{Next Word}} \mid \underbrace{x_1, \ldots, x_{i-1}}_{\text{Context}})$$

$$\begin{aligned}
P(\text{Jane went to the store}) = {} & P(Jane \mid <s>) \times P(went \mid Jane) \times \\
& P(to \mid went) \times P(the \mid to) \times \\
& P(store \mid the) \times P(. \mid store) \\
& P(</s> \mid .)
\end{aligned}$$

Calculating the Probability of a Sentence

$$P(X) = \prod_{i=1}^{I} P(\underbrace{x_i}_{\text{Next Word}} \mid \underbrace{x_1, \ldots, x_{i-1}}_{\text{Context}})$$

The big problem: How do we predict

$$P(x_i \mid x_1, \ldots, x_{i-1})$$

?!?!

Count-based Language Models

Count-based Language Models

- Count up the frequency and divide:

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

Count-based Language Models

- Count up the frequency and divide:

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

Corpus:

The cat sat on the mat .
A dog chased the cat .
A mouse ate some cheese .
The mouse ran under a mat .

Count-based Language Models

- Count up the frequency and divide:

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

Corpus:

The cat sat on the mat .	A mouse ate some cheese .
A dog chased the cat .	The mouse ran under a mat .

$$p(chased \mid dog) = ? \quad p(cat \mid the) = ? \quad p(the \mid <s>) = ?$$

Count-based Language Models

- Count up the frequency and divide:

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

Corpus:

The cat sat on the mat .
A dog chased the cat .
A mouse ate some cheese .
The mouse ran under a mat .

$$p(chased \mid dog) = \frac{1}{1} = 1 \quad p(cat \mid the) = \frac{1}{4} = 0.25 \quad p(the \mid <s>) = 0.5$$

Count-based Language Models

- Count up the frequency and divide:

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

Corpus:

The cat sat on the mat .	A mouse ate some cheese .
A dog chased the cat .	The mouse ran under a mat .

$p($A cat chased the mouse .$) = ?$

Count-based Language Models

Corpus:

The cat sat on the mat .
A dog chased the cat .
A mouse ate some cheese .
The mouse ran under a mat .

p(A cat chased the mouse .) =

$$p(<s>|A) \times$$
$$p(cat\,|\,a) \times$$
$$p(chased\,|\,cat) \times$$
$$p(the\,|\,chased) \times$$
$$p(mouse\,|\,the) \times$$
$$p(\,.\,|\,mouse)$$

Count-based Language Models

Corpus:

The cat sat on the mat .

A dog chased the cat .

A mouse ate some cheese .

The mouse ran under a mat .

$p($A cat chased the mouse .$) =$

$$p(<s> | A) \times$$
$$p(cat | a) \times$$
$$p(chased | cat) \times$$
$$p(the | chased) \times$$
$$p(mouse | the) \times$$
$$p(. | mouse)$$

Count-based Language Models

- Count up the frequency and divide:

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

- Add smoothing to deal with zero counts:

$$p(x_i | x_{i-n+1:i-1}) = \frac{c(x_{i-n+1:i}) + \alpha}{c(x_{i-n+1:i-1}) + \alpha |V|}$$

Count-based Language Models

Corpus:

The cat sat on the mat . A mouse ate some cheese .
A dog chased the cat . The mouse ran under a mat .

$$|V| = |\{the, a, cat, sat, \dots\}| = 15 \qquad \alpha = 1$$

Count-based Language Models

Corpus:

The cat sat on the mat .
A dog chased the cat .
A mouse ate some cheese .
The mouse ran under a mat .

$|V| = |\{the, a, cat, sat, \ldots\}| = 15 \qquad \alpha = 1$

$p($A cat chased the mouse .$) =$

$$p(<s> \| A) \times$$
$$p(cat \mid a) \times$$
$$p(chased \mid cat) \times$$
$$p(the \mid chased) \times$$
$$p(mouse \mid the) \times$$
$$p(\,.\mid mouse)$$

Count-based Language Models

- Count up the frequency and divide:

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

- Add smoothing to deal with zero counts:

$$P(x_i \mid x_{i-n+1:i-1}) = \frac{c(x_{i-n+1:i}) + \alpha}{c(x_{i-n+1:i-1}) + \alpha |V|}$$

- Another way to smooth: skip some words

$$\begin{aligned} P(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) =& \lambda P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) \\ &+ (1-\lambda) P(x_i \mid x_{1-n+2}, \ldots, x_{i-1}) \end{aligned}$$

Evaluation

- **Log-likelihood:**

$$LL(\mathcal{E}_{test}) = \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

Evaluation

- **Log-likelihood:**

$$LL(\mathcal{E}_{test}) = \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

Evaluation

- **Log-likelihood:**

$$LL(\mathcal{E}_{test}) = \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word (Cross) Entropy:**

$$H(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} - \log_2 P(E)$$

Evaluation

- **Log-likelihood:**

$$LL(\mathcal{E}_{test}) = \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word (Cross) Entropy:**

$$H(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} -\log_2 P(E)$$

- **Perplexity:**

$$ppl(\mathcal{E}_{test}) = 2^{H(\mathcal{E}_{test})} = e^{-WLL(\mathcal{E}_{test})}$$

Evaluation

What does “My LM achieves a perplexity of 23” mean?

https://sjmielke.com/comparing-perplexities.htm

What Can we Do w/ LMs?

- Score sentences:

Jane went to the store . → high
store to Jane went the . → low

(same as calculating loss for training)

What Can we Do w/ LMs?

- Score sentences:

Jane went to the store . → high
store to Jane went the . → low

(same as calculating loss for training)

- Generate sentences:

while didn't choose end-of-sentence symbol:
  calculate probability
  sample a new word from the probability distribution

Problems and Solutions?

- Cannot share strength among **similar words**

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

→ solution: class based language models

Problems and Solutions?

- Cannot share strength among **similar words**

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

→ solution: class based language models

- Cannot condition on context with **intervening words**

Dr. Jane Smith	Dr. Gertrude Smith

→ solution: skip-gram language models

Problems and Solutions?

- Cannot share strength among **similar words**

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

→ solution: class based language models

- Cannot condition on context with **intervening words**

Dr. Jane Smith   Dr. Gertrude Smith

→ solution: skip-gram language models

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet

for programming class he wanted to buy his own computer

→ solution: cache, trigger, topic, syntactic models, etc.

An Alternative:
Featurized Log-Linear Models

An Alternative: Featurized Models

- Calculate features of the context
- Based on the features, calculate probabilities
- Optimize feature weights using gradient descent, etc.

Example:

Previous words: “giving a"

Example:

Previous words: “giving a"

a
the
talk
gift
hat
…

Words we’re predicting

Example:

Previous words: “giving a"

a	
the	
talk	$b = \begin{pmatrix} 3.0 \\ 2.5 \\ -0.2 \\ 0.1 \\ 1.2 \\ \ldots \end{pmatrix}$
gift	
hat	
…	
Words we’re predicting	How likely are they?

Example:

Previous words: "giving a"

a the talk gift hat …	$b=\begin{pmatrix}3.0\\2.5\\-0.2\\0.1\\1.2\\\ldots\end{pmatrix}$	$w_{1,a}=\begin{pmatrix}-6.0\\-5.1\\0.2\\0.1\\0.5\\\ldots\end{pmatrix}$
Words we're predicting	How likely are they?	How likely are they given prev. word is "a"?

Example:

Previous words: “giving a"

$$\begin{array}{c} \text{a} \\ \text{the} \\ \text{talk} \\ \text{gift} \\ \text{hat} \\ \dots \end{array} \qquad b=\begin{pmatrix} 3.0 \\ 2.5 \\ -0.2 \\ 0.1 \\ 1.2 \\ \dots \end{pmatrix} \quad w_{1,a}=\begin{pmatrix} -6.0 \\ -5.1 \\ 0.2 \\ 0.1 \\ 0.5 \\ \dots \end{pmatrix} \quad w_{2,giving}=\begin{pmatrix} -0.2 \\ -0.3 \\ 1.0 \\ 2.0 \\ -1.2 \\ \dots \end{pmatrix}$$

Words we’re predicting

How likely are they?

How likely are they given prev. word is “a”?

How likely are they given 2nd prev. word is “giving”?

Example:

Previous words: “giving a"

$$
\begin{array}{c} a \\ the \\ talk \\ gift \\ hat \\ \ldots \end{array}
\quad
b=\begin{pmatrix} 3.0 \\ 2.5 \\ -0.2 \\ 0.1 \\ 1.2 \\ \ldots \end{pmatrix}
\quad
w_{1,a}=\begin{pmatrix} -6.0 \\ -5.1 \\ 0.2 \\ 0.1 \\ 0.5 \\ \ldots \end{pmatrix}
\quad
w_{2,giving}=\begin{pmatrix} -0.2 \\ -0.3 \\ 1.0 \\ 2.0 \\ -1.2 \\ \ldots \end{pmatrix}
\quad
s=\begin{pmatrix} -3.2 \\ -2.9 \\ 1.0 \\ 2.2 \\ 0.6 \\ \ldots \end{pmatrix}
$$

Words we’re predicting	How likely are they?	How likely are they given prev. word is “a”?	How likely are they given 2nd prev. word is “giving”?	Total score

Softmax

- Convert scores into probabilities by taking the exponent and normalizing (softmax)

Softmax

- Convert scores into probabilities by taking the exponent and normalizing (softmax)

$$P(x_i \mid x_{i-n+1}^{i-1}) = \frac{e^{s(x_i \mid x_{i-n+1}^{i-1})}}{\sum_{\tilde{x}_i} e^{s(\tilde{x}_i \mid x_{i-n+1}^{i-1})}}$$

Softmax

- Convert scores into probabilities by taking the exponent and normalizing (softmax)

$$P(x_i \mid x_{i-n+1}^{i-1}) = \frac{e^{s(x_i \mid x_{i-n+1}^{i-1})}}{\sum_{\tilde{x}_i} e^{s(\tilde{x}_i \mid x_{i-n+1}^{i-1})}}$$

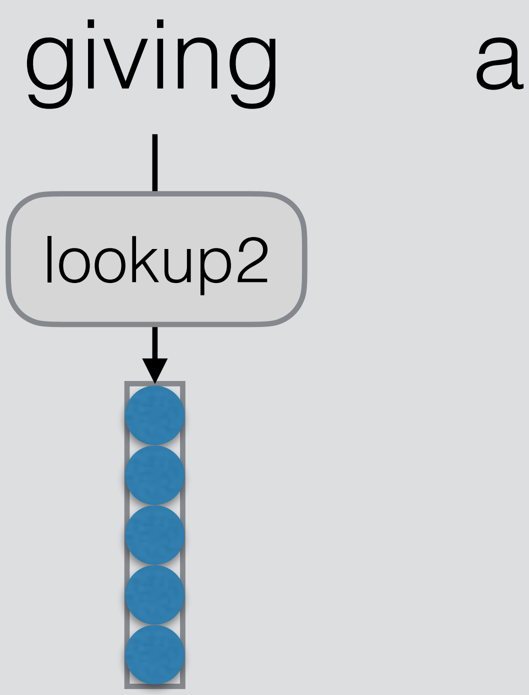
$$\text{s}=\begin{pmatrix} -3.2 \\ -2.9 \\ 1.0 \\ 2.2 \\ 0.6 \\ \dots \end{pmatrix} \longrightarrow \text{p}=\begin{pmatrix} 0.002 \\ 0.003 \\ 0.329 \\ 0.444 \\ 0.090 \\ \dots \end{pmatrix}$$

A Computation Graph View

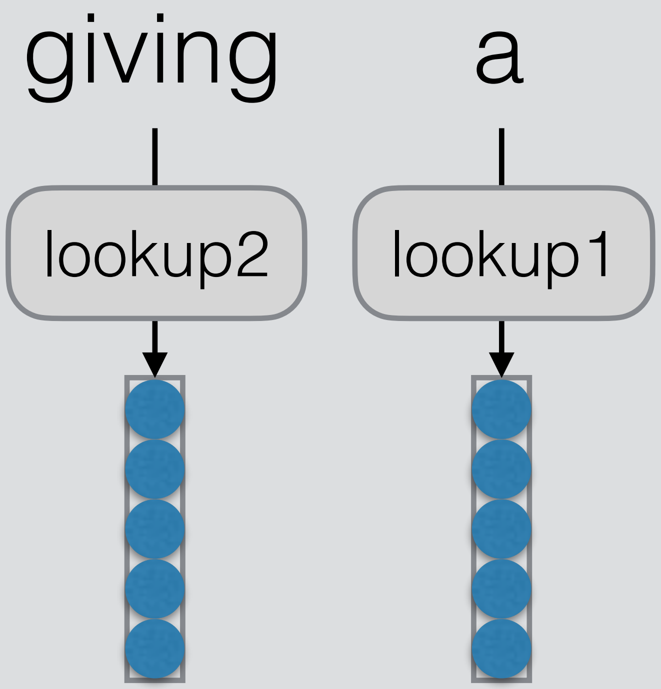
giving a

Each vector is size of output vocabulary

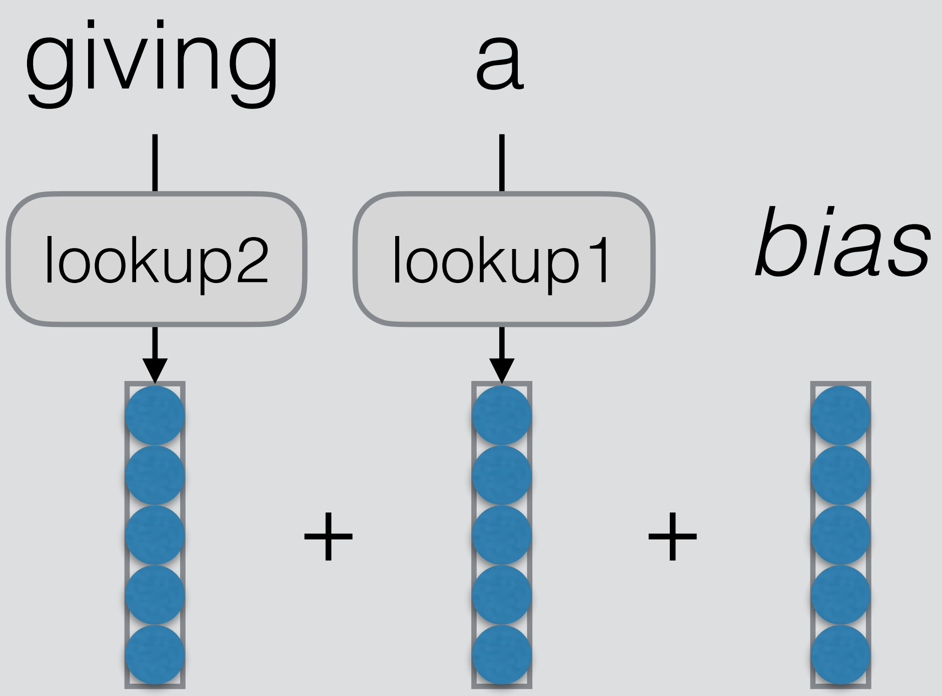
A Computation Graph View



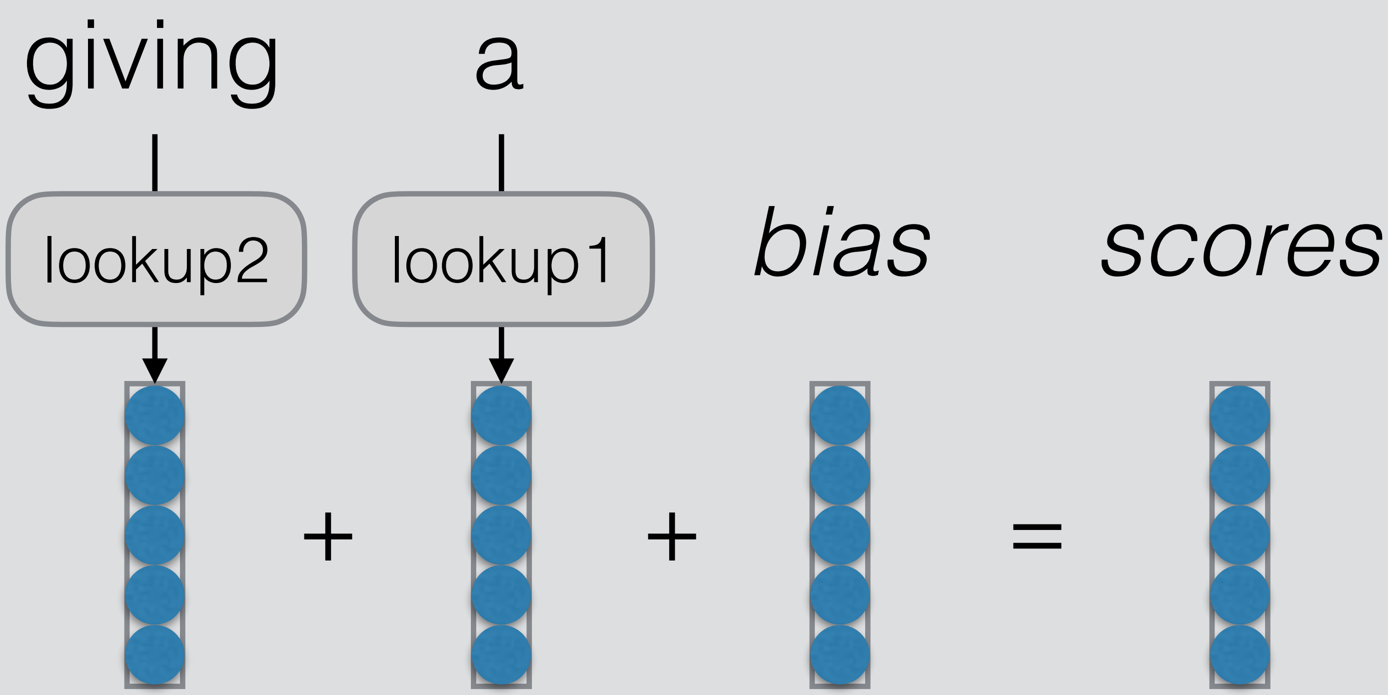
A Computation Graph View



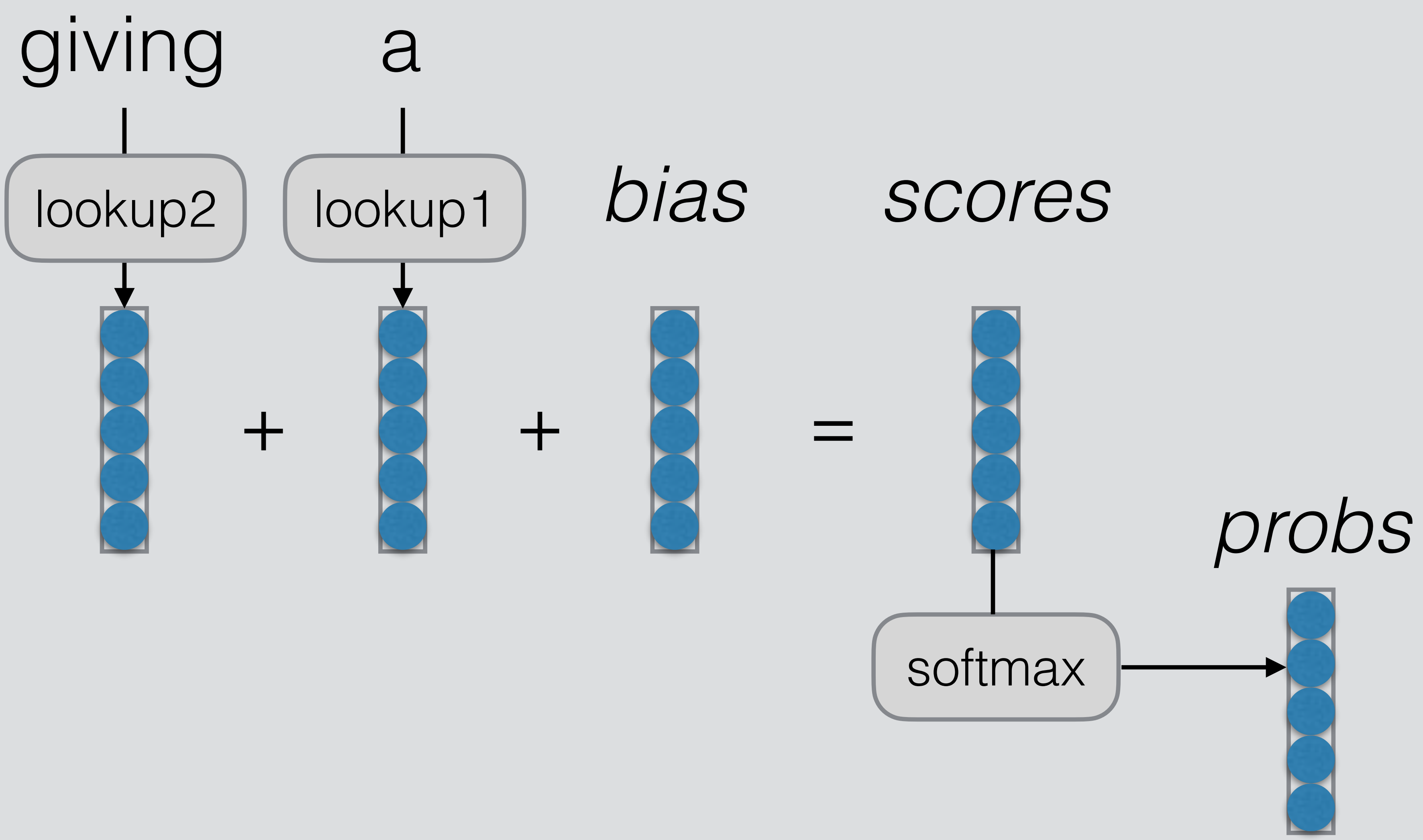
A Computation Graph View



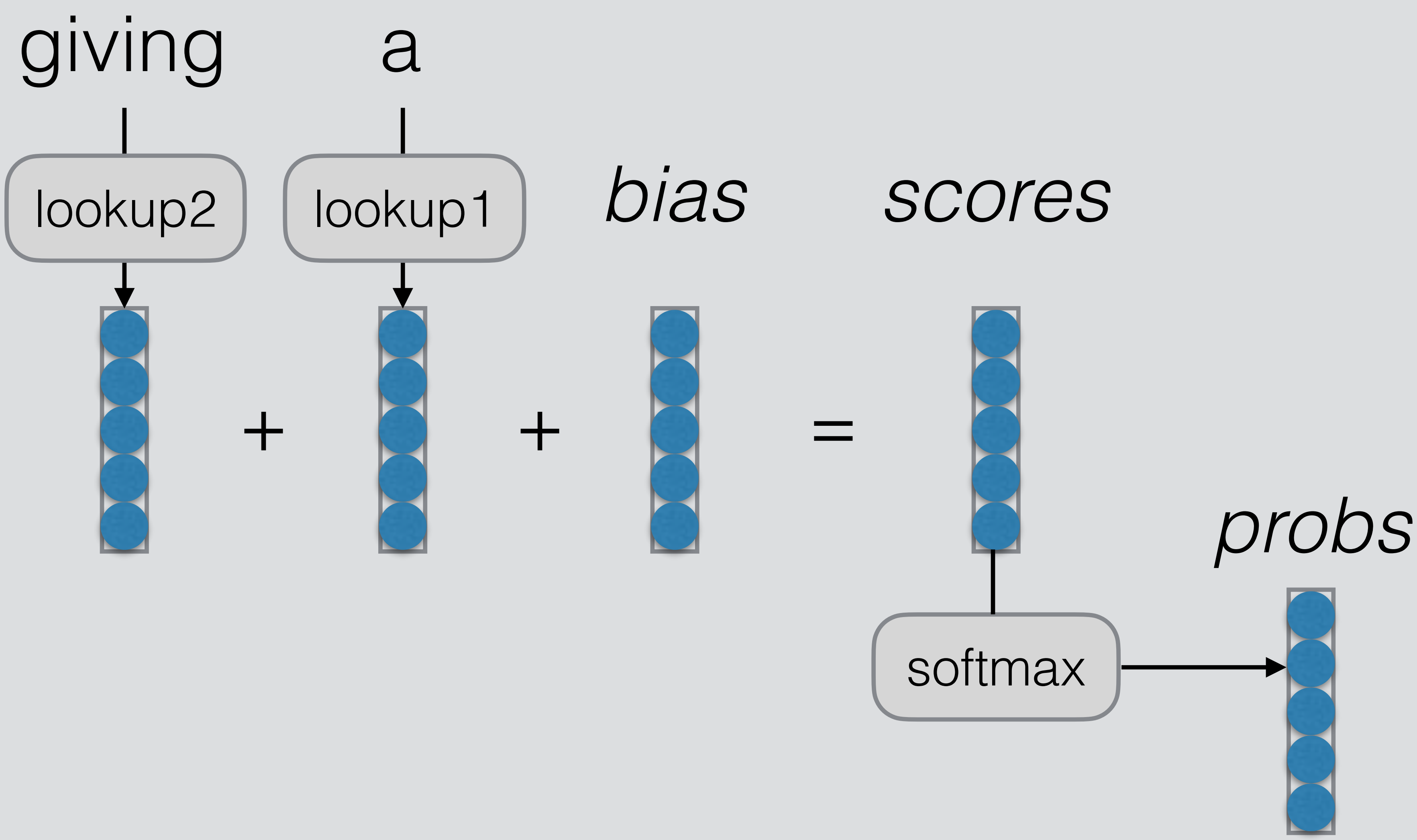
A Computation Graph View



A Computation Graph View



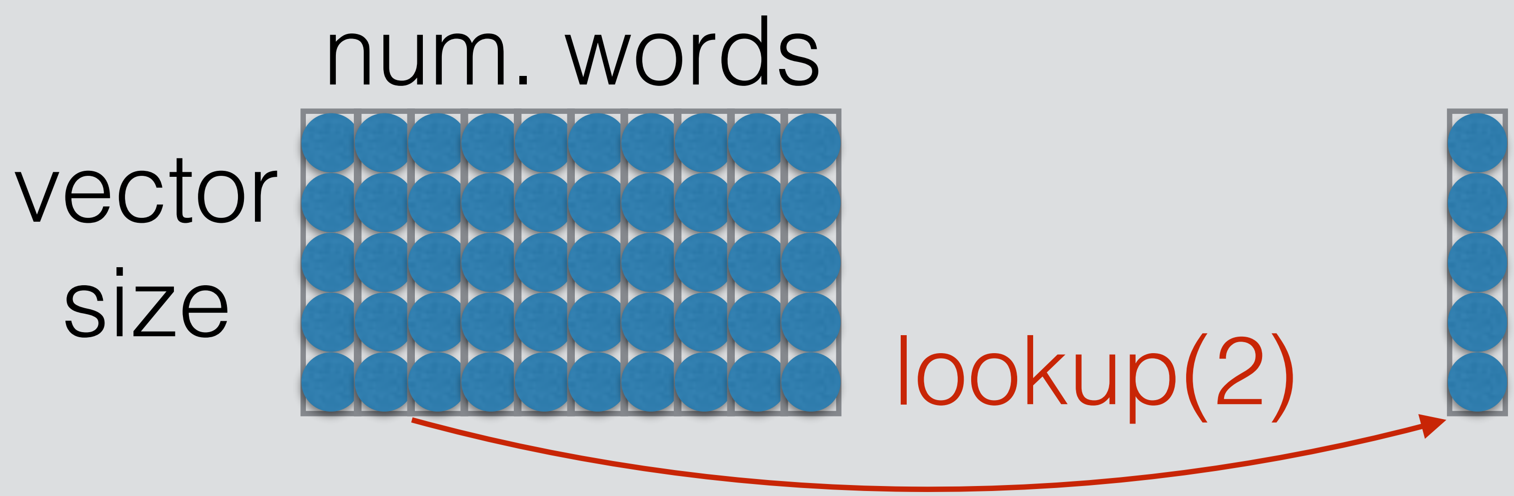
A Computation Graph View



Each vector is size of output vocabulary

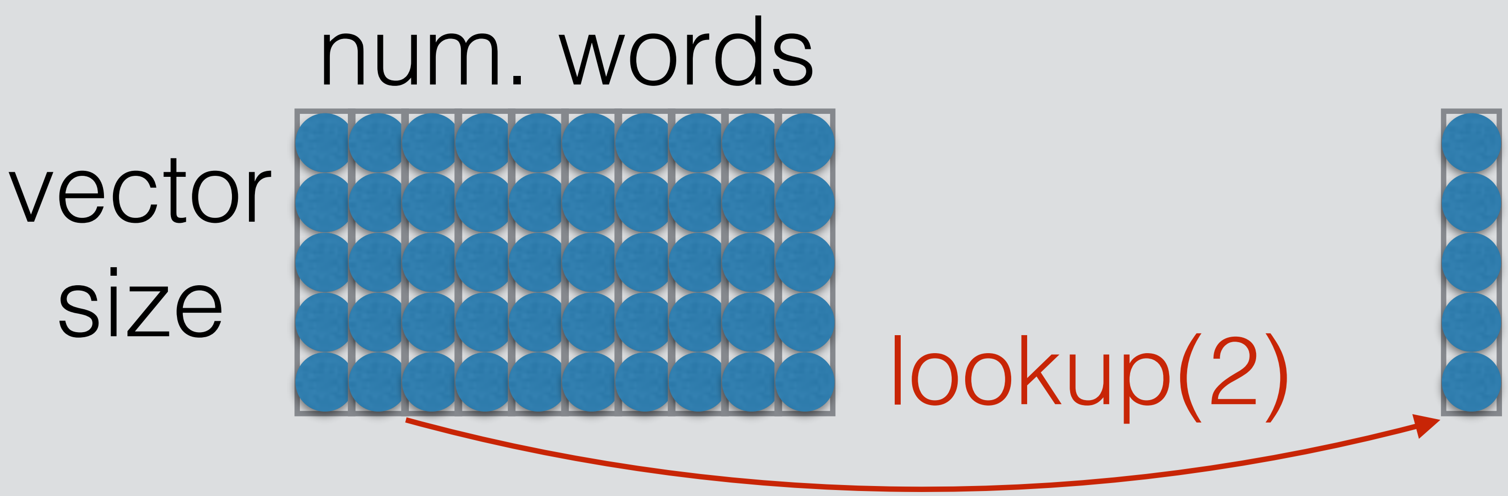
A Note: "Lookup"

- Lookup can be viewed as "grabbing" a single vector from a big matrix of word embeddings

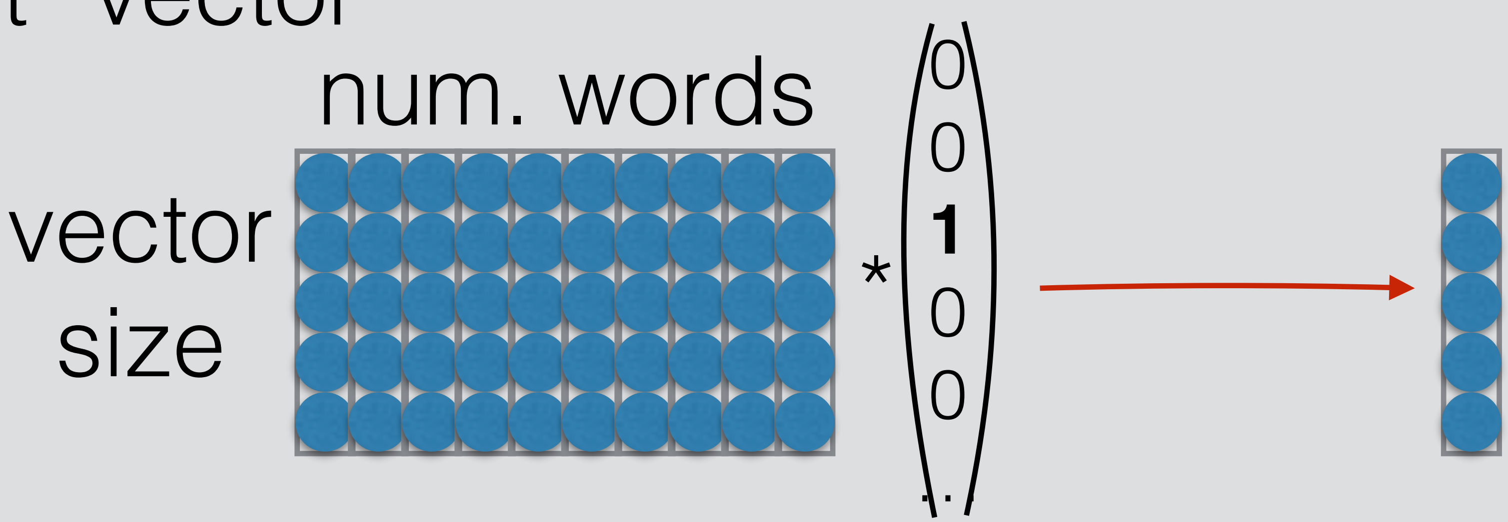


A Note: "Lookup"

- Lookup can be viewed as "grabbing" a single vector from a big matrix of word embeddings

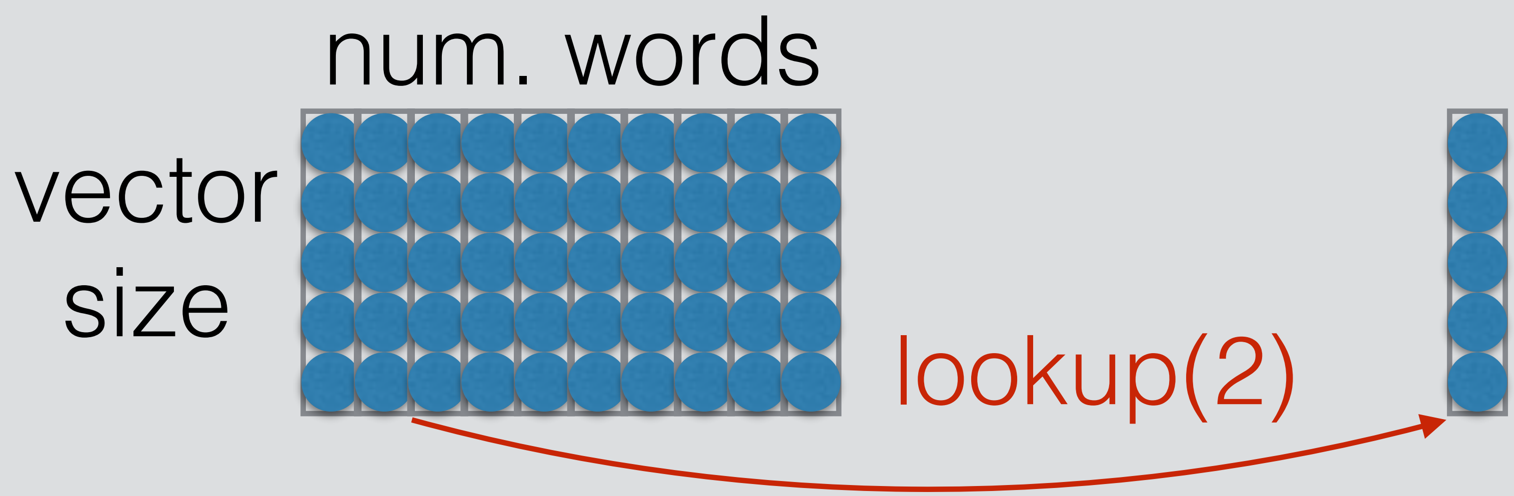


- Similarly, can be viewed as multiplying by a "one-hot" vector

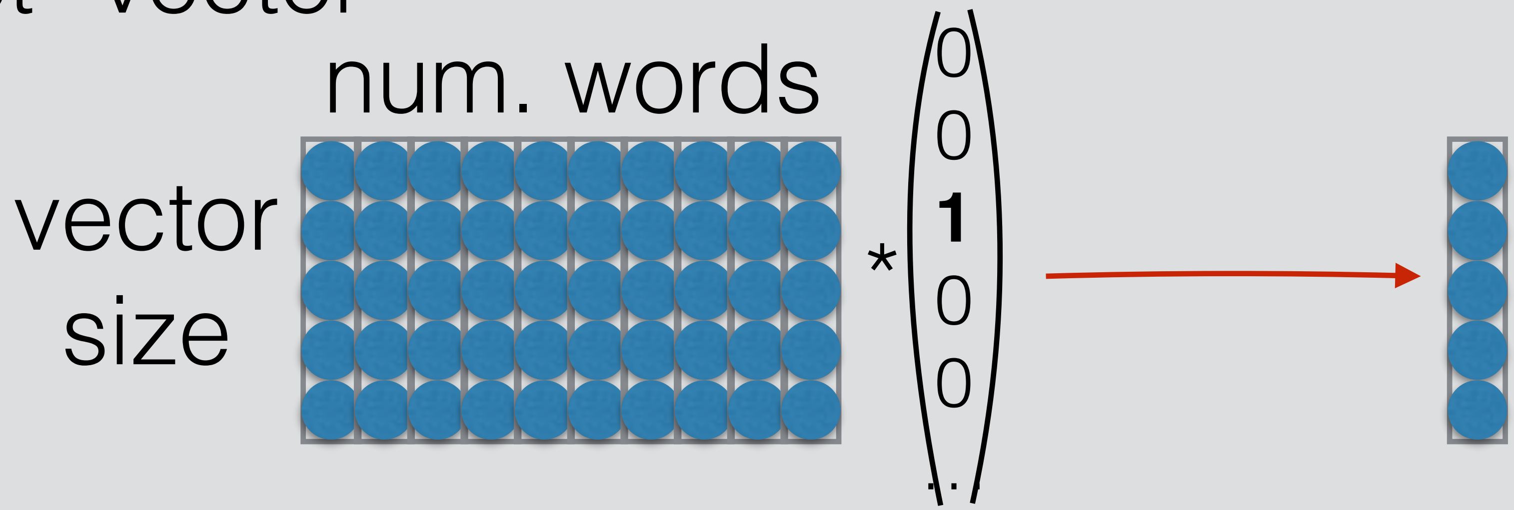


A Note: "Lookup"

- Lookup can be viewed as "grabbing" a single vector from a big matrix of word embeddings



- Similarly, can be viewed as multiplying by a "one-hot" vector



- Former tends to be faster

Training a Model

- **Reminder:** to train, we calculate a "loss function" (a measure of how bad our predictions are), and move the parameters to reduce the loss
- The most common loss function for probabilistic models is "negative log likelihood"

Training a Model

- **Reminder:** to train, we calculate a "loss function" (a measure of how bad our predictions are), and move the parameters to reduce the loss
- The most common loss function for probabilistic models is "negative log likelihood"

If element 3 (or zero-indexed, 2) is the correct answer:

$$p = \begin{pmatrix} 0.002 \\ 0.003 \\ \boxed{0.329} \\ 0.444 \\ 0.090 \end{pmatrix} \qquad 0.329 \xrightarrow{-\log} 1.112$$

$\ldots$

Parameter Update

- Back propagation allows us to calculate the derivative of the loss with respect to the parameters

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}}$$

Parameter Update

- Back propagation allows us to calculate the derivative of the loss with respect to the parameters

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}}$$

- Simple stochastic gradient descent optimizes parameters according to the following rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \ell}{\partial \boldsymbol{\theta}}$$

Choosing a Vocabulary

Unknown Words

Unknown Words

- Necessity for UNK words

Unknown Words

- Necessity for UNK words
  - We won’t have all the words in the world in training data

Unknown Words

- Necessity for UNK words
  - We won’t have all the words in the world in training data
  - Larger vocabularies require more memory and computation time

Unknown Words

- Necessity for UNK words
  - We won’t have all the words in the world in training data
  - Larger vocabularies require more memory and computation time
- Common ways:
  - Frequency threshold (usually UNK <= 1)

Unknown Words

- Necessity for UNK words
  - We won’t have all the words in the world in training data
  - Larger vocabularies require more memory and computation time
- Common ways:
  - Frequency threshold (usually UNK <= 1)
  - Rank threshold

Unknown Words

A very large number of published documents contain text only. They often look boring, and they are often written in obscure language, using mile-long sentences and cryptic technical terms, using one font only, perhaps even without headings. Such style, or lack of style, might be the one you are strongly expected to follow when writing eg scientific or technical reports, legal documents, or administrative papers. It is natural to think that such documents would benefit from a few illustrative images. (However, just adding illustration might be rather useless, if the text remains obscure and unstructured.)

Unknown Words

a very large number of published documents contain text only . they often look boring , and they are often written in obscure language , using mile-long sentences and cryptic technical terms , using one font only , perhaps even without headings . such style, or lack of style, might be the one you are strongly expected to follow when writing eg scientific or technical reports , legal documents , or administrative papers . it is natural to think that such documents would benefit from a few illustrative images . (however , just adding illustration might be rather useless , if the text remains obscure and unstructured .)

truecase + tokenize

Unknown Words

a very large number of published documents contain text only . they often look boring , and they are often written in obscure language , using mile-long sentences and cryptic technical terms , using one font only , perhaps even without headings . such style, or lack of style, might be the one you are strongly expected to follow when writing eg scientific or technical reports , legal documents , or administrative papers . it is natural to think that such documents would benefit from a few illustrative images . (however , just adding illustration might be rather useless , if the text remains obscure and unstructured .)

Find rare words (e.g. with freq<2)

Unknown Words

a very large number of published documents contain text only . they often look boring , and they are often written in obscure language , using UNK sentences and cryptic technical terms , using one font only , perhaps even without headings . such style, or lack of style, might be the one you are strongly expected to follow when writing eg scientific or technical reports , legal documents , or UNK papers . it is natural to think that such documents would benefit from a few illustrative images . (however , just adding UNK might be rather useless , if the text remains obscure and UNK .)

Substitute with UNK

Evaluation and Vocabulary

- **Important:** the vocabulary must be the same over models you compare

Evaluation and Vocabulary

- **Important:** the vocabulary must be the same over models you compare
- Or more accurately, all models must be able to generate the test set (it’s OK if they can generate *more* than the test set, but not less)
  - e.g. Comparing a character-based model to a word-based model is fair, but not vice-versa

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

- Cannot condition on context with **intervening words**

Dr. Jane Smith   Dr. Gertrude Smith

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet

for programming class he wanted to buy his own computer

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

→ not solved yet 😞

- Cannot condition on context with **intervening words**

Dr. Jane Smith   Dr. Gertrude Smith

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet

for programming class he wanted to buy his own computer

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car she purchased a car	she bought a bicycle she purchased a bicycle

→ not solved yet 😞

- Cannot condition on context with **intervening words**

Dr. Jane Smith	Dr. Gertrude Smith

→ solved! 😀

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet

for programming class he wanted to buy his own computer

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car she purchased a car	she bought a bicycle she purchased a bicycle

→ not solved yet 😞

- Cannot condition on context with **intervening words**

Dr. Jane Smith	Dr. Gertrude Smith

→ solved! 😀

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet
for programming class he wanted to buy his own computer

→ not solved yet 😞

Break
Beyond Linear Models

Linear Models can't Learn Feature Combinations

Linear Models can't Learn Feature Combinations

farmers eat steak → **high**

Linear Models can't Learn Feature Combinations

farmers eat steak → **high**

farmers eat hay → **low**

Linear Models can't Learn Feature Combinations

farmers eat steak → **high**

farmers eat hay → **low**

cows eat steak → **low**

Linear Models can't Learn Feature Combinations

farmers eat steak → **high**

farmers eat hay → **low**

cows eat steak → **low**

cows eat hay → **high**

Linear Models can't Learn Feature Combinations

farmers eat steak → **high**

farmers eat hay → **low**

cows eat steak → **low**

cows eat hay → **high**

- These can't be expressed by linear features

Linear Models can't Learn Feature Combinations

farmers eat steak → **high**
farmers eat hay → **low**

cows eat steak → **low**
cows eat hay → **high**

- These can't be expressed by linear features
- What can we do?
  - Remember combinations as features (individual scores for "farmers eat", "cows eat")

Linear Models can't Learn Feature Combinations

farmers eat steak → **high**

farmers eat hay → **low**

cows eat steak → **low**

cows eat hay → **high**

- These can't be expressed by linear features
- What can we do?
  - Remember combinations as features (individual scores for "farmers eat", "cows eat")
    → Feature space explosion!

Linear Models can't Learn Feature Combinations

farmers eat steak → **high**

farmers eat hay → **low**

cows eat steak → **low**

cows eat hay → **high**

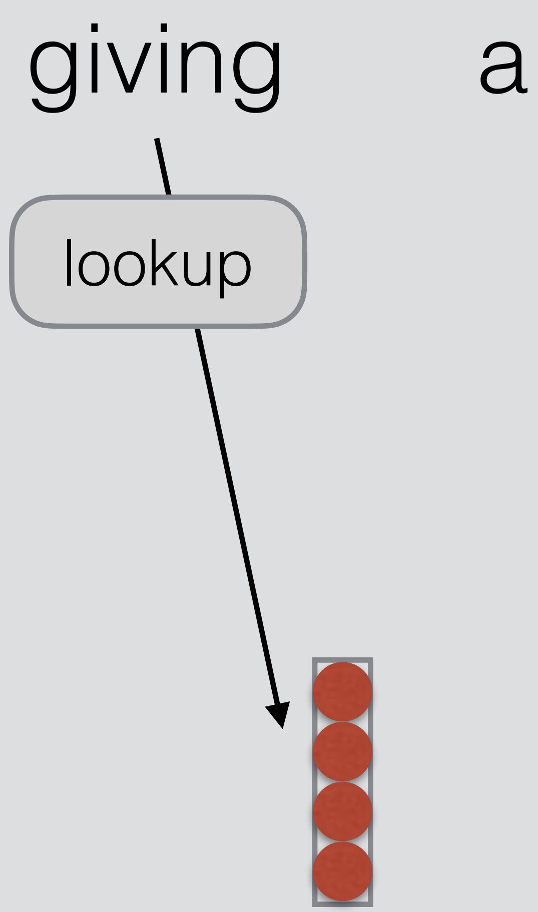
- These can't be expressed by linear features
- What can we do?
  - Remember combinations as features (individual scores for "farmers eat", "cows eat")
    → Feature space explosion!
  - Neural nets

Neural Language Models

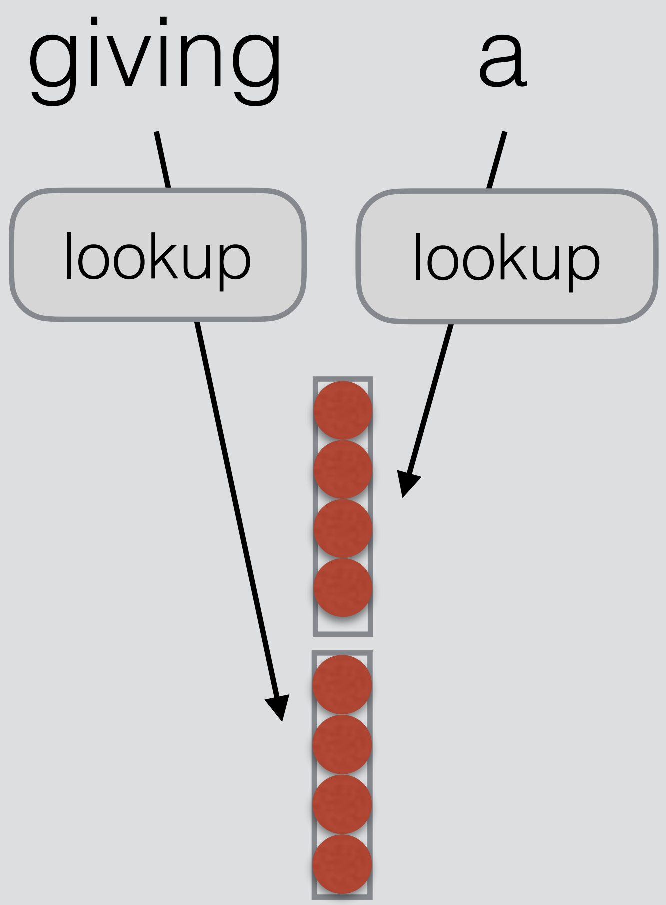
- (See Bengio et al. 2004)

Neural Language Models

- (See Bengio et al. 2004)



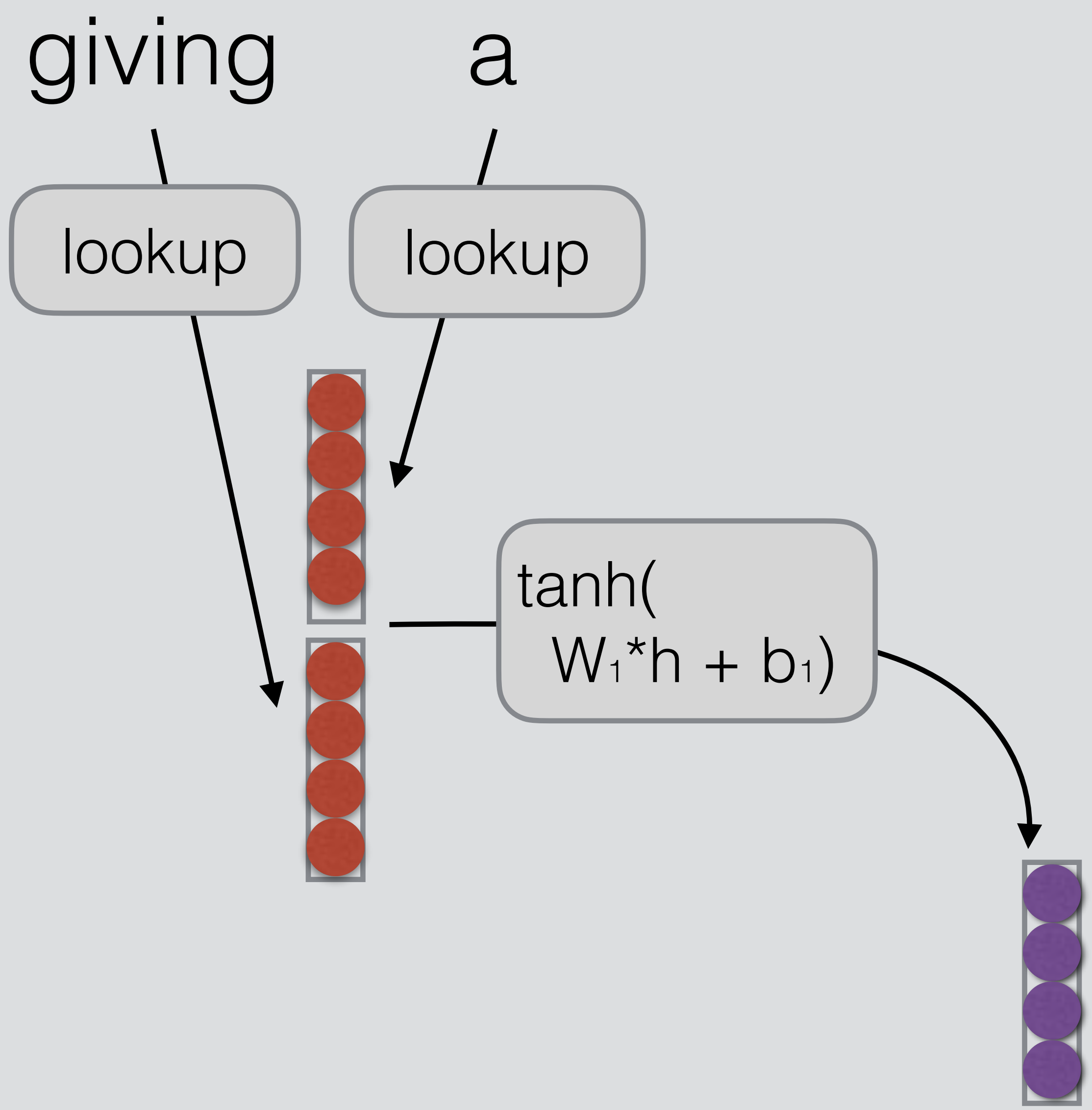
Neural Language Models



- (See Bengio et al. 2004)

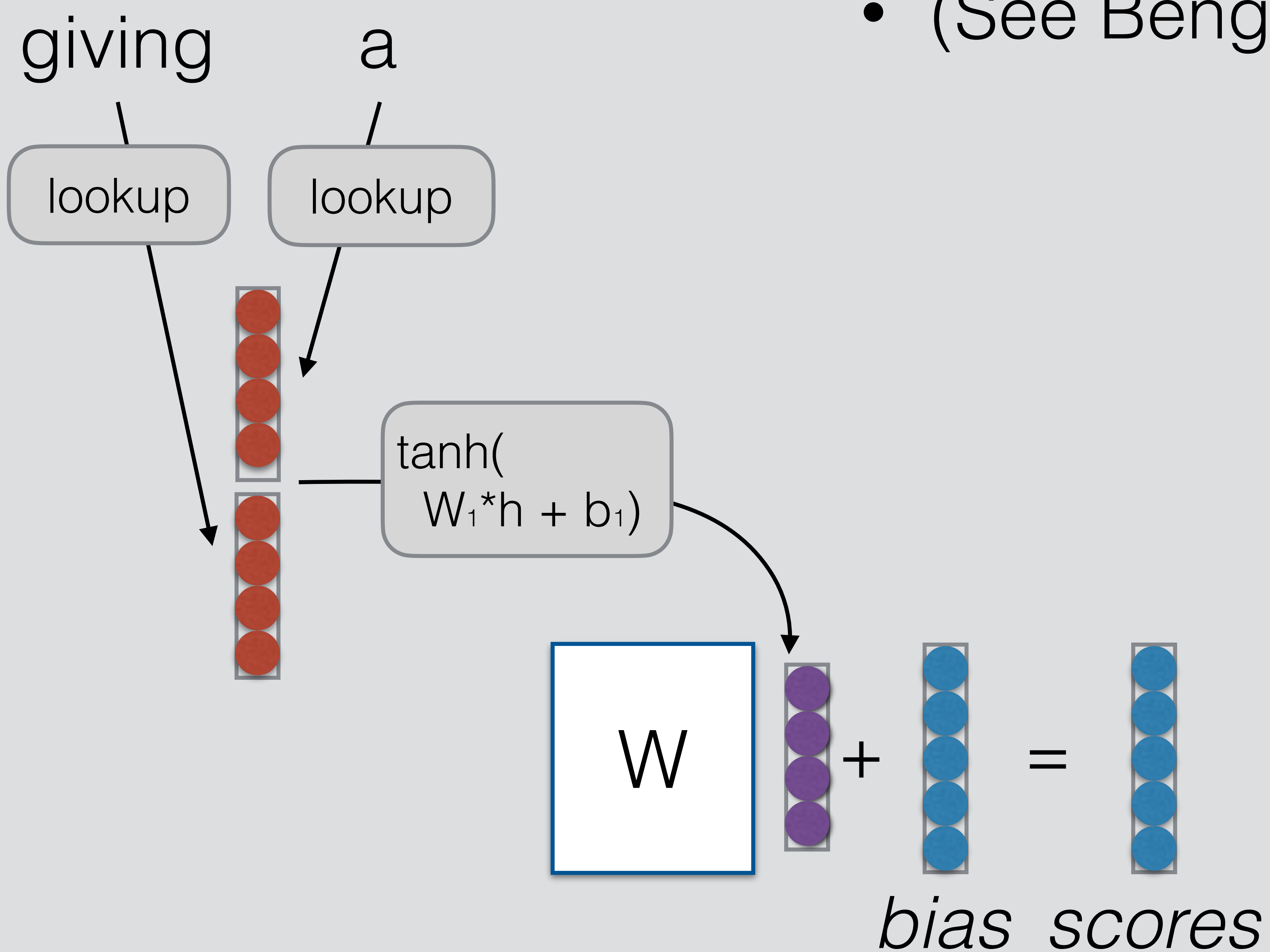
Neural Language Models

- (See Bengio et al. 2004)



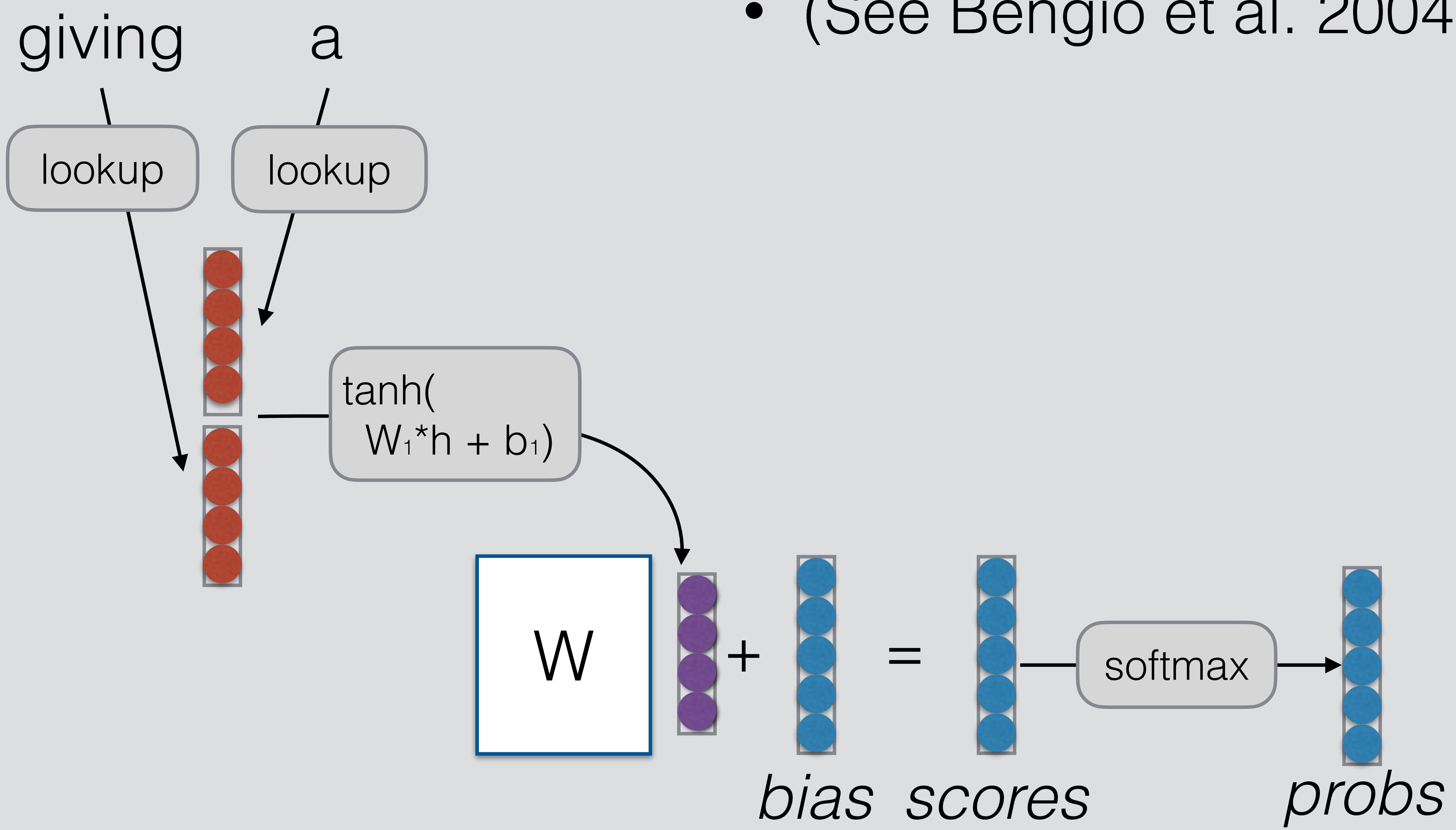
Neural Language Models

- (See Bengio et al. 2004)

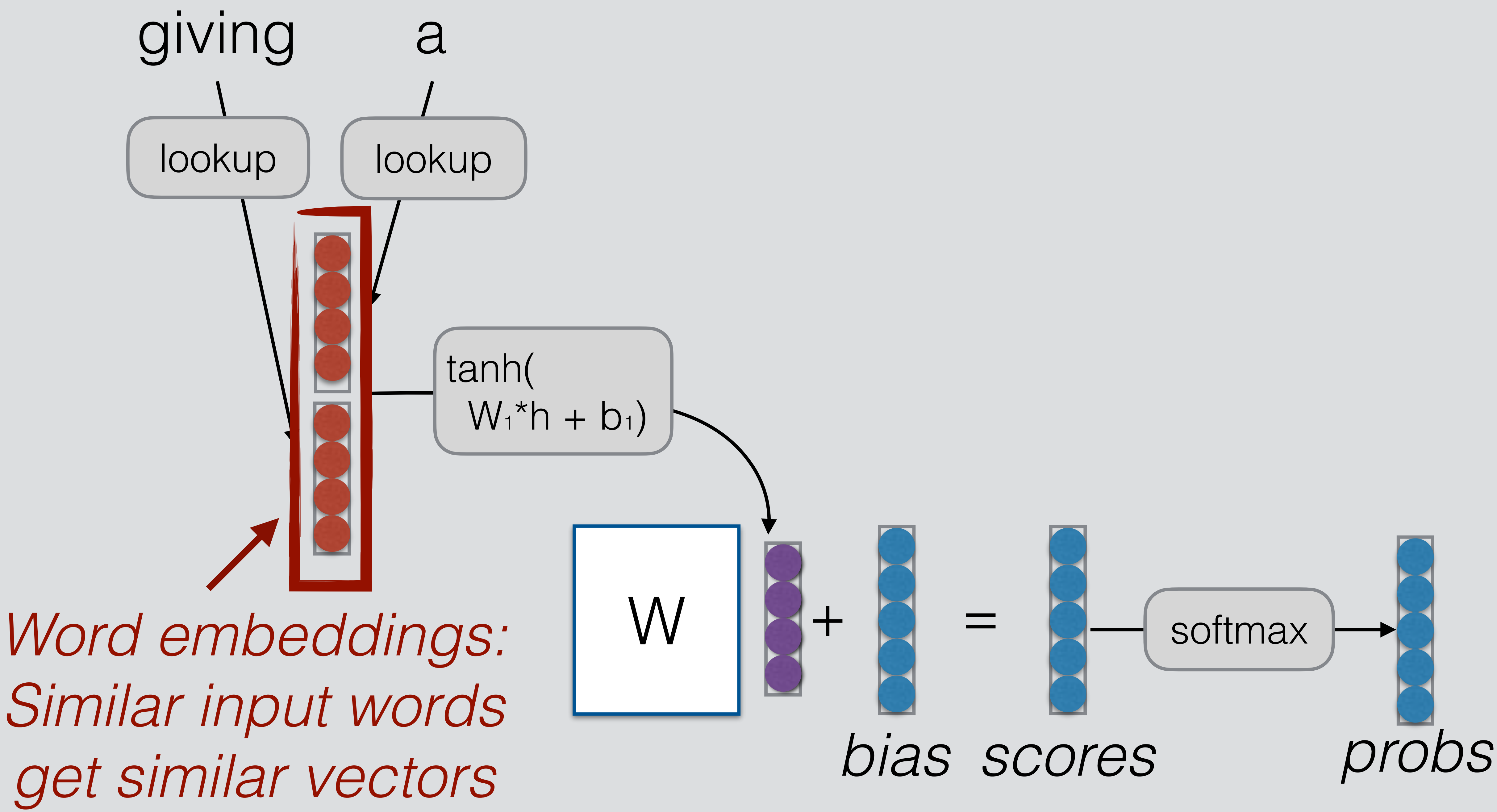


Neural Language Models

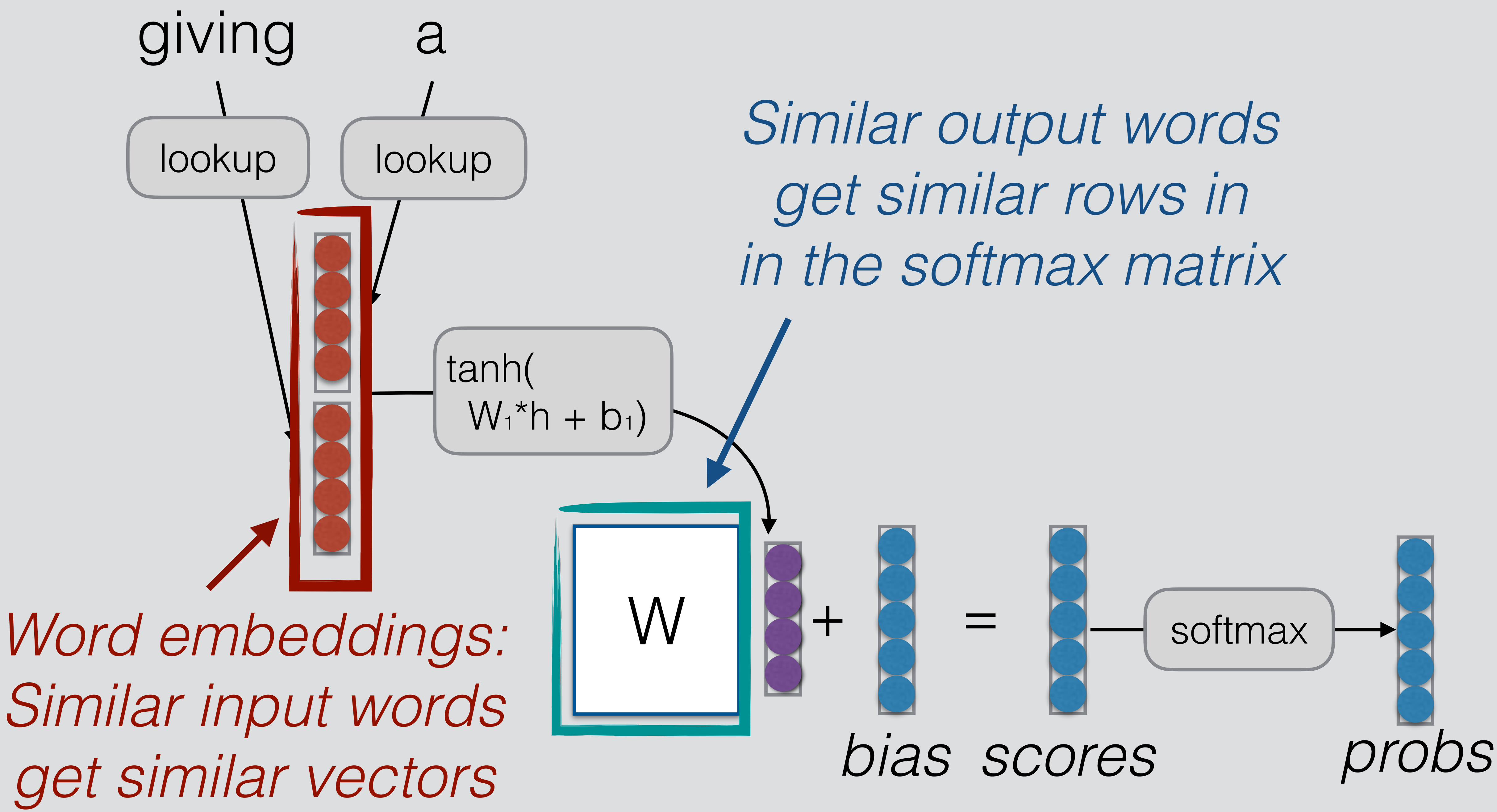
- (See Bengio et al. 2004)



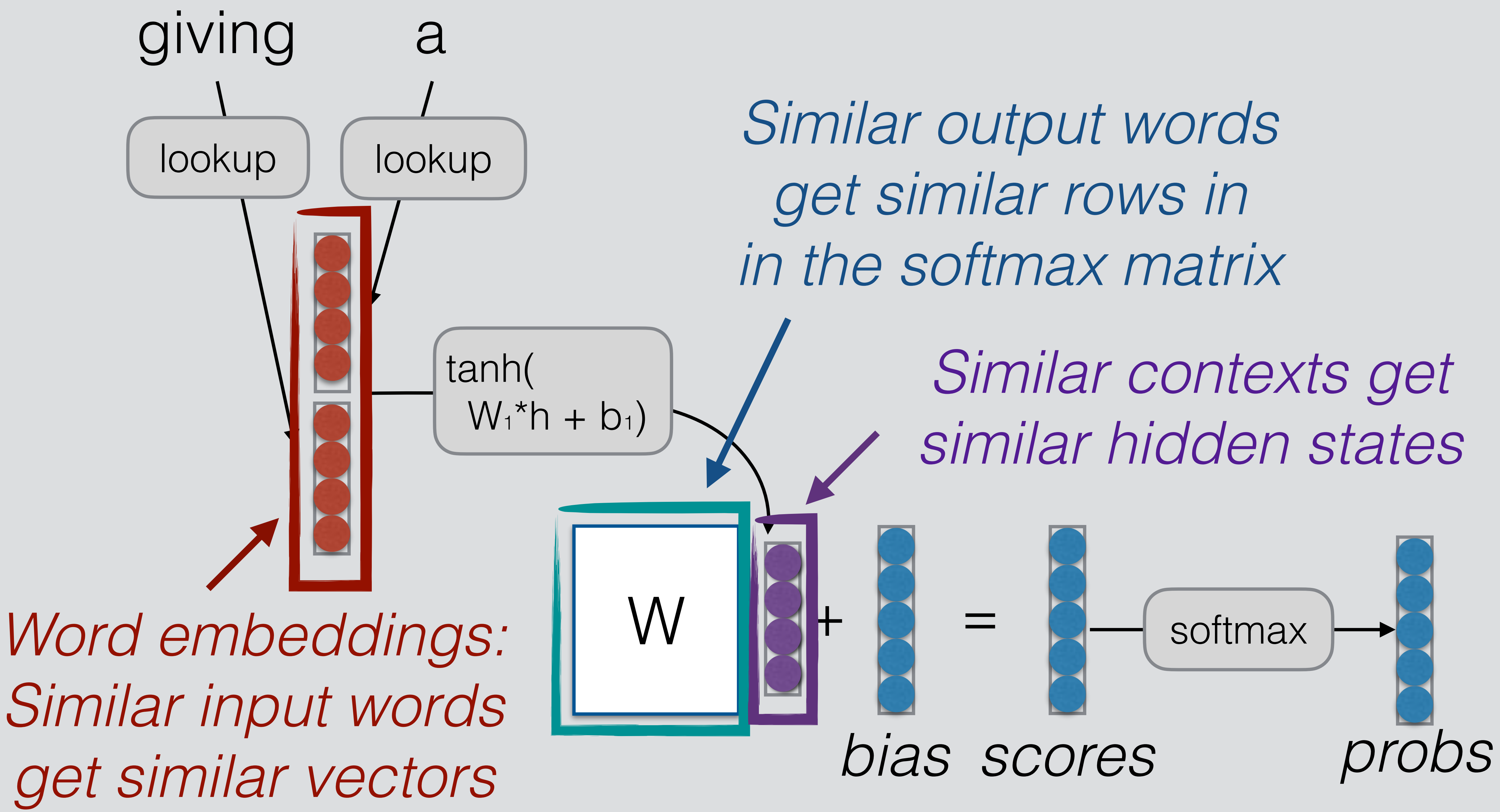
Where is Strength Shared?



Where is Strength Shared?



Where is Strength Shared?



What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car she purchased a car	she bought a bicycle she purchased a bicycle

→ solved, and similar contexts as well! 😀

- Cannot condition on context with **intervening words**

Dr. Jane Smith   Dr. Gertrude Smith

→ solved! 😀

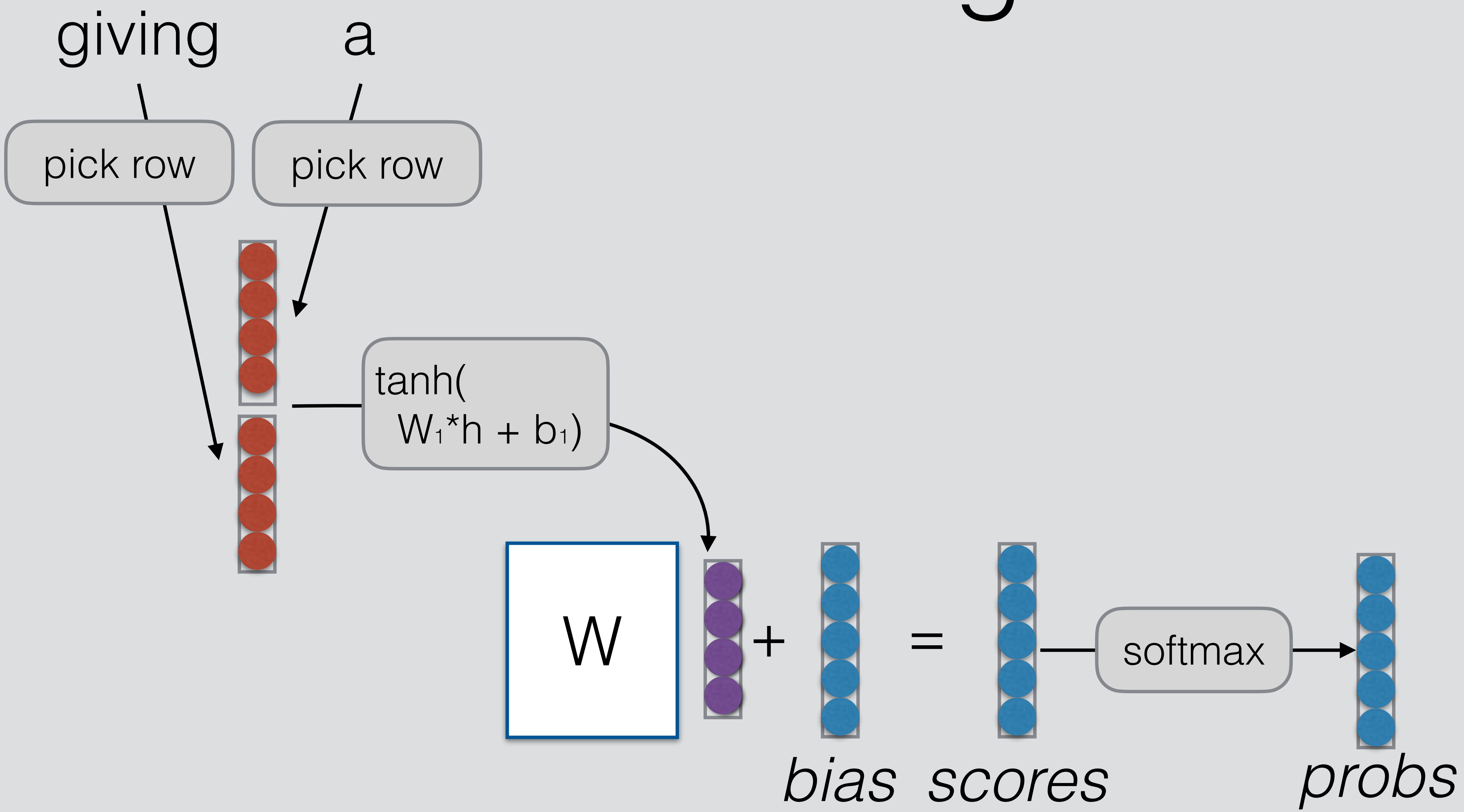
- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet

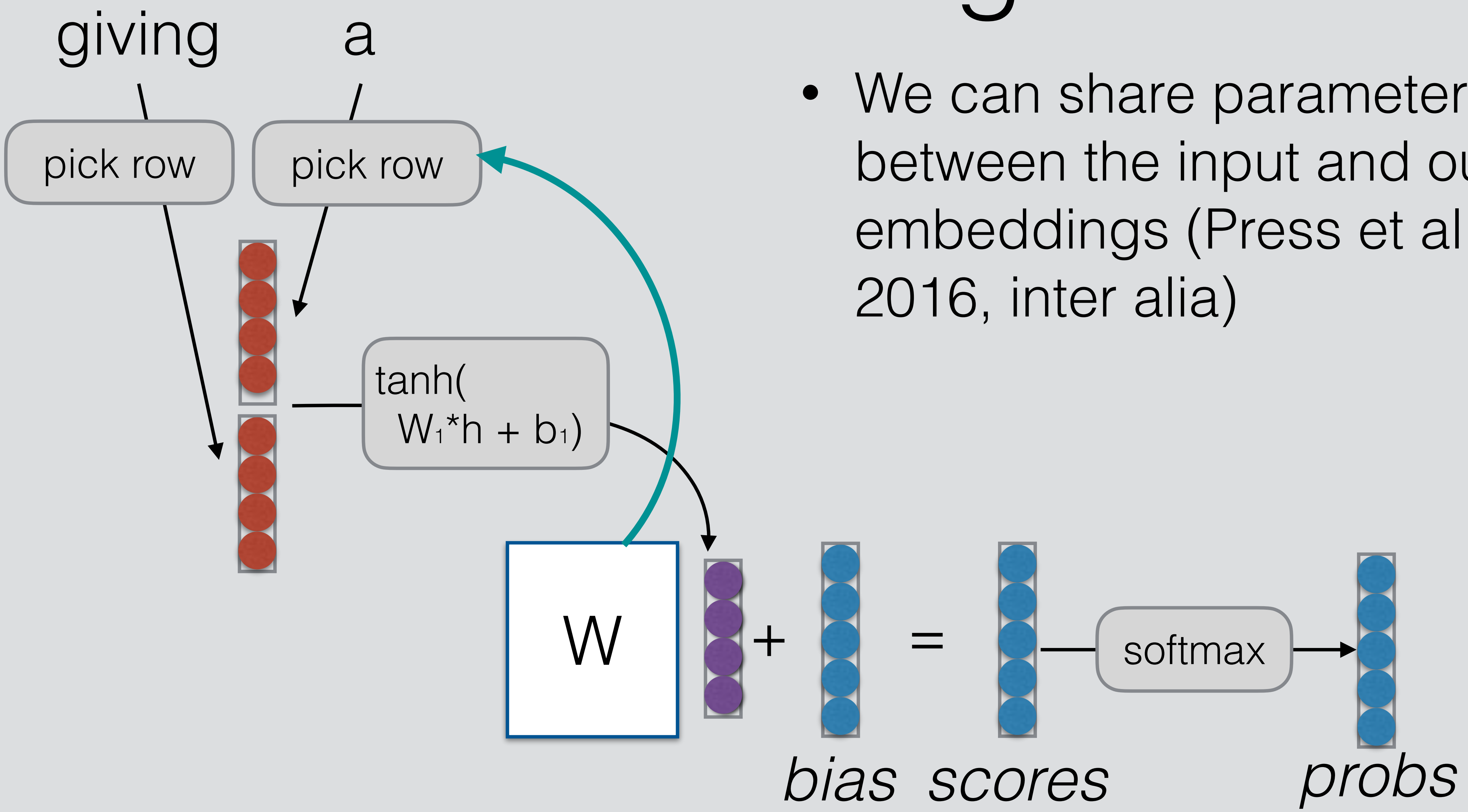
for programming class he wanted to buy his own computer

→ not solved yet 😞

Tying Input/Output Embeddings



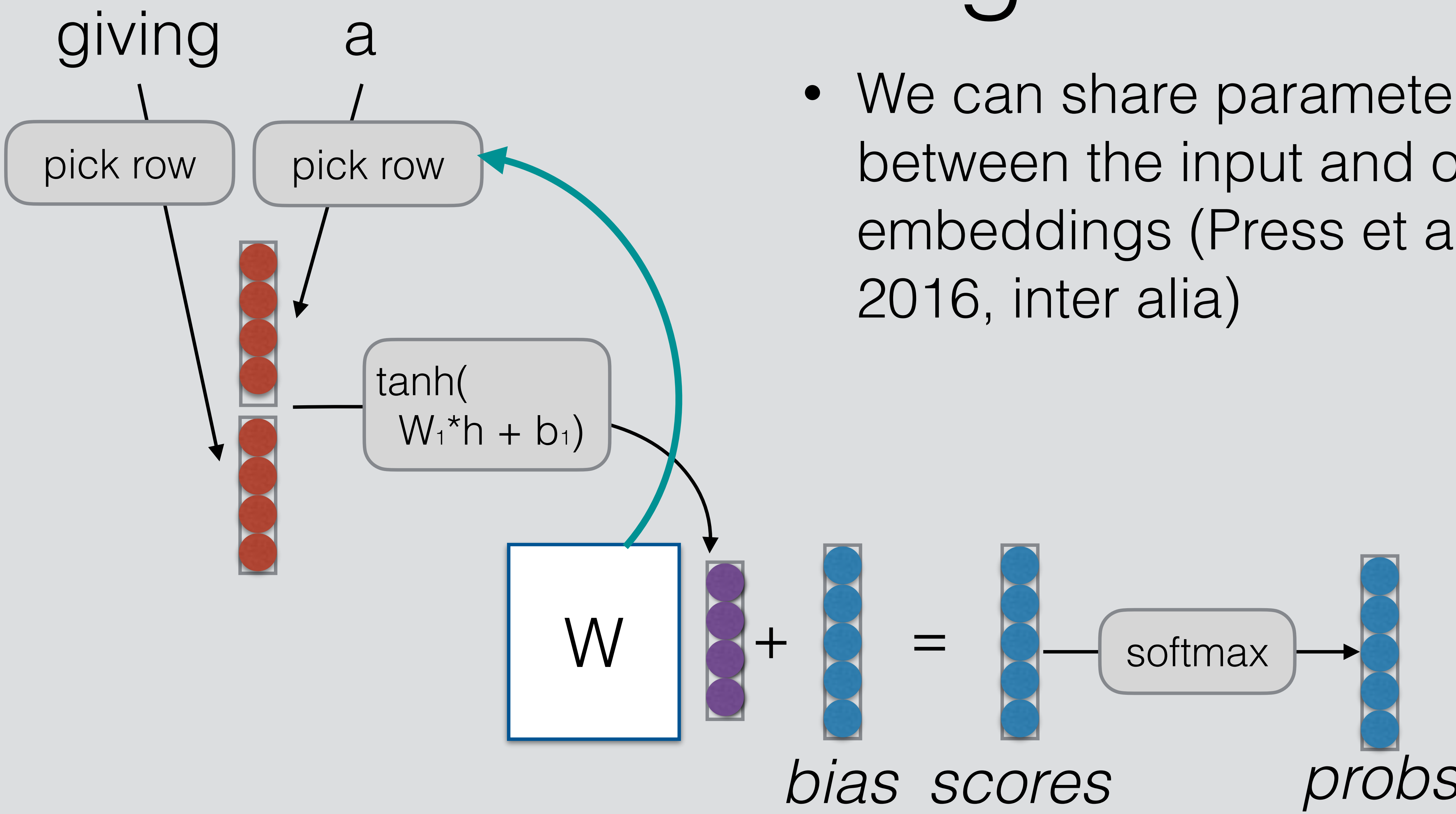
Tying Input/Output Embeddings



- We can share parameters between the input and output embeddings (Press et al. 2016, inter alia)

Tying Input/Output Embeddings

- We can share parameters between the input and output embeddings (Press et al. 2016, inter alia)



Want to try? Delete the input embeddings, and instead pick a row from the softmax matrix.

Training Tricks

Shuffling the Training Data

- Stochastic gradient methods update the parameters a little bit at a time

Shuffling the Training Data

- Stochastic gradient methods update the parameters a little bit at a time
    - What if we have the sentence “I love this sentence so much!” at the end of the training data 50 times?

Shuffling the Training Data

- Stochastic gradient methods update the parameters a little bit at a time
    - What if we have the sentence “I love this sentence so much!” at the end of the training data 50 times?
- To train correctly, we should randomly shuffle the order at each time step

Other Optimization Options

- **SGD with Momentum:** Remember gradients from past time steps to prevent sudden changes

- **Adagrad:** Adapt the learning rate to reduce learning rate for frequently updated parameters (as measured by the variance of the gradient)

- **Adam:** Like Adagrad, but keeps a running average of momentum and gradient variance

- **Many others:** RMSProp, Adadelta, etc.
(See Ruder 2016 reference for more details)

Early Stopping, Learning Rate Decay

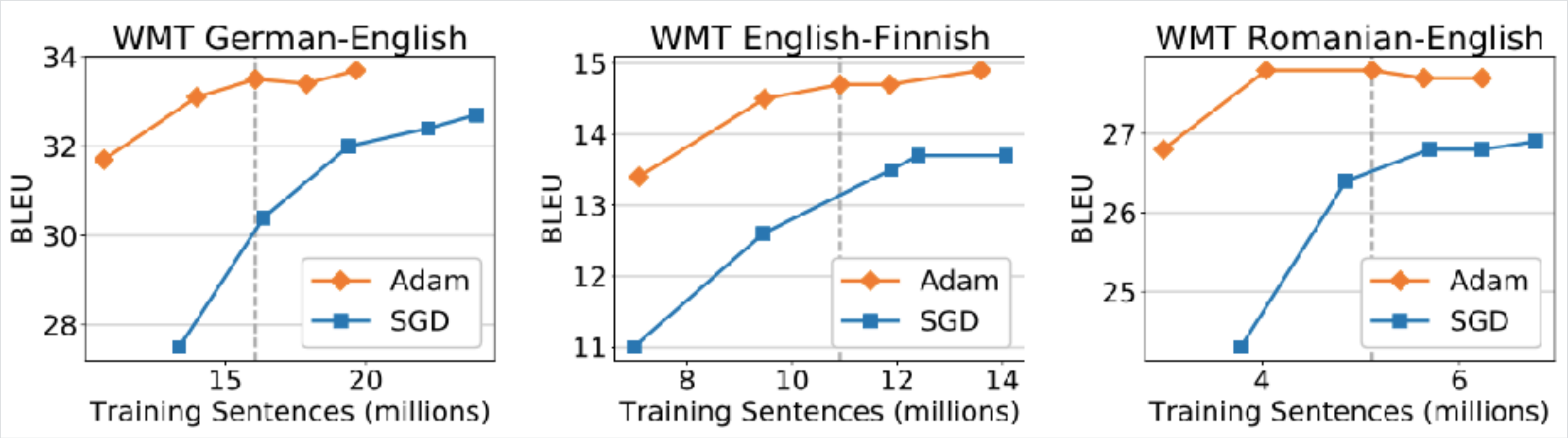
- Neural nets have tons of parameters: we want to prevent them from over-fitting

- We can do this by monitoring our performance on held-out development data and stopping training when it starts to get worse

- It also sometimes helps to reduce the learning rate and continue training

Which One to Use?

- Adam is usually fast to converge and stable
- But simple SGD tends to do very will in terms of generalization (Wilson et al. 2017)
- You should use learning rate decay, (e.g. on Machine translation results by Denkowski & Neubig 2017)



Dropout

(Srivastava+ 14)

Dropout

(Srivastava+ 14)

- Neural nets have lots of parameters, and are prone to overfitting

Dropout
(Srivastava+ 14)

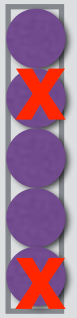
- Neural nets have lots of parameters, and are prone to overfitting

- Dropout: randomly zero-out nodes in the hidden layer with probability *p* at **training time only**

Dropout

(Srivastava+ 14)

- Neural nets have lots of parameters, and are prone to overfitting
- Dropout: randomly zero-out nodes in the hidden layer with probability *p* at **training time only**

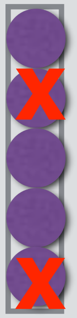


Dropout

(Srivastava+ 14)

- Neural nets have lots of parameters, and are prone to overfitting

- Dropout: randomly zero-out nodes in the hidden layer with probability *p* at **training time only**



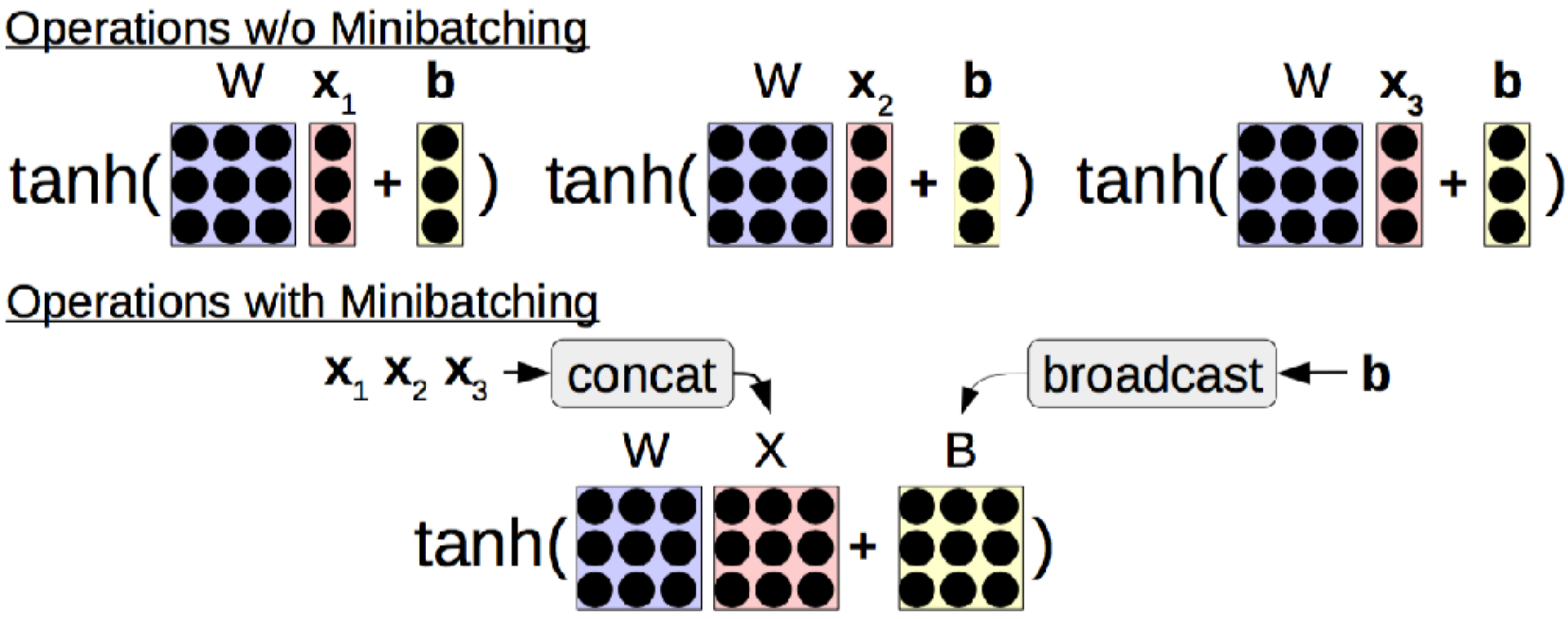
- Because the number of nodes at training/test is different, scaling is necessary:
    - **Standard dropout:** scale by *p* at test time
    - **Inverted dropout:** scale by 1/(1-*p*) at training time
- An alternative: **DropConnect** (Wan+ 2013) instead zeros out weights in the NN

Efficiency Tricks: Operation Batching

Efficiency Tricks: Mini-batching

- On modern hardware 10 operations of size 1 is **much slower than** 1 operation of size 10
- Minibatching combines together smaller operations into one big one

Minibatching



Manual Mini-batching

Manual Mini-batching

- Group together similar operations (e.g. loss calculations for a single word) and execute them all together

Manual Mini-batching

- Group together similar operations (e.g. loss calculations for a single word) and execute them all together
  - In the case of a feed-forward language model, each word prediction in a sentence can be batched

Manual Mini-batching

- Group together similar operations (e.g. loss calculations for a single word) and execute them all together
  - In the case of a feed-forward language model, each word prediction in a sentence can be batched
  - For recurrent neural nets, etc., more complicated

Manual Mini-batching

- Group together similar operations (e.g. loss calculations for a single word) and execute them all together
  - In the case of a feed-forward language model, each word prediction in a sentence can be batched
  - For recurrent neural nets, etc., more complicated
- How this works depends on toolkit
  - Most toolkits require you to add an extra dimension representing the batch size

Manual Mini-batching

- Group together similar operations (e.g. loss calculations for a single word) and execute them all together
  - In the case of a feed-forward language model, each word prediction in a sentence can be batched
  - For recurrent neural nets, etc., more complicated
- How this works depends on toolkit
  - Most toolkits require you to add an extra dimension representing the batch size
  - DyNet has special minibatch operations for lookup and loss functions, everything else automatic
  - In PyTorch (almost) all operations already automatically support batches

Mini-batched Code Example

```
# in_words is a tuple (word_1, word_2)
# out_label is an output label
word_1 = E[in_words[0]]
word_2 = E[in_words[1]]
scores_sym = W*dy.concatenate([word_1, word_2])+b
loss_sym = dy.pickneglogsoftmax(scores_sym, out_label)
```

(a) Non-minibatched classification.

```
# in_words is a list [(word_{1,1}, word_{1,2}), (word_{2,1}, word_{2,2}), ...]
# out_labels is a list of output labels [label_1, label_2, ...]
word_1_batch = dy.lookup_batch(E, [x[0] for x in in_words])
word_2_batch = dy.lookup_batch(E, [x[1] for x in in_words])
scores_sym = W*dy.concatenate([word_1_batch, word_2_batch])+b
loss_sym = dy.sum_batches( dy.pickneglogsoftmax_batch(scores_sym, out_labels) )
```

(b) Minibatched classification.

A Case Study:
Regularizing and Optimizing LSTM Language Models (Merity et al. 2017)

Regularizing and Optimizing LSTM Language Models (Merity et al. 2017)

- Uses LSTMs as a backbone (discussed later)
- A number of tricks to improve stability and prevent overfitting:
  - DropConnect regularization
  - SGD w/ averaging triggered when model is close to convergence
  - Dropout on recurrent connections and embeddings
  - Weight tying
  - Independently tuned embedding and hidden layer sizes
  - Regularization of activations of the network
- Strong baseline for language modeling, SOTA at the time (without special model, just training methods)

Break
Next: Recurrent Neural Networks

NLP and Sequential Data

- NLP is full of sequential data
  - Words in sentences
  - Characters in words
  - Sentences in discourse
  - …

Long-distance Dependencies in Language

- Agreement in number, gender, etc.

He does not have very much confidence in **himself**.
She does not have very much confidence in **herself**.

Long-distance Dependencies in Language

- Agreement in number, gender, etc.

He does not have very much confidence in **himself**.
She does not have very much confidence in **herself**.

- Selectional preference

The **reign** has lasted as long as the life of the **queen**.
The **rain** has lasted as long as the life of the **clouds**.

Can be Complicated!

- What is the referent of “it”?

The trophy would not fit in the brown suitcase because it was too **big**.

Can be Complicated!

- What is the referent of “it”?

The trophy would not fit in the brown suitcase because it was too **big**.

Trophy

Can be Complicated!

- What is the referent of “it”?

The trophy would not fit in the brown suitcase because it was too **big**.

Trophy

The trophy would not fit in the brown suitcase because it was too **small**.

Can be Complicated!

- What is the referent of “it”?

The trophy would not fit in the brown suitcase because it was too **big**.

Trophy

The trophy would not fit in the brown suitcase because it was too **small**.

Suitcase

(from Winograd Schema Challenge:
http://commonsensereasoning.org/winograd.html)

Recurrent Neural Networks

(Elman 1990)

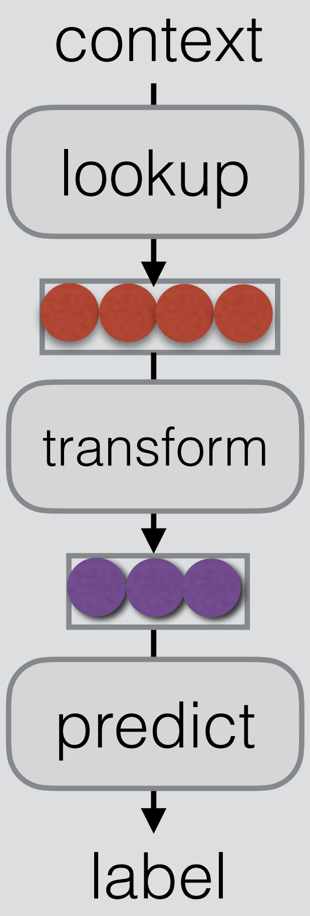
- Tools to “remember” information

Recurrent Neural Networks

(Elman 1990)

- Tools to “remember” information

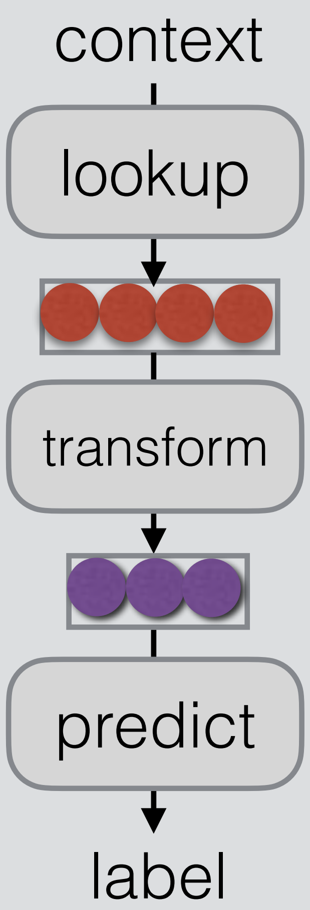
Feed-forward NN



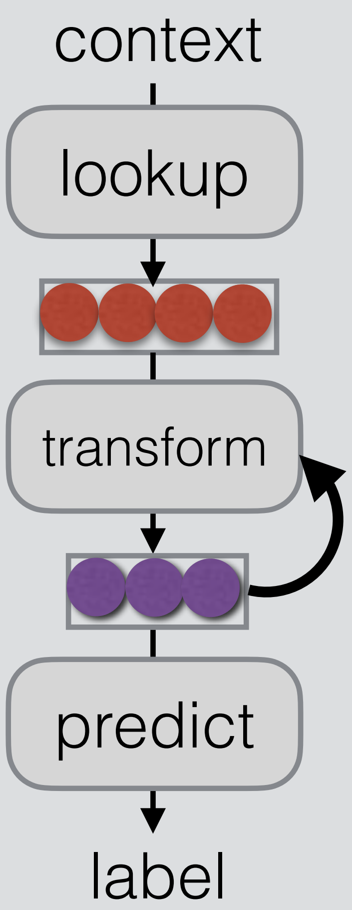
Recurrent Neural Networks
(Elman 1990)

- Tools to "remember" information

Feed-forward NN

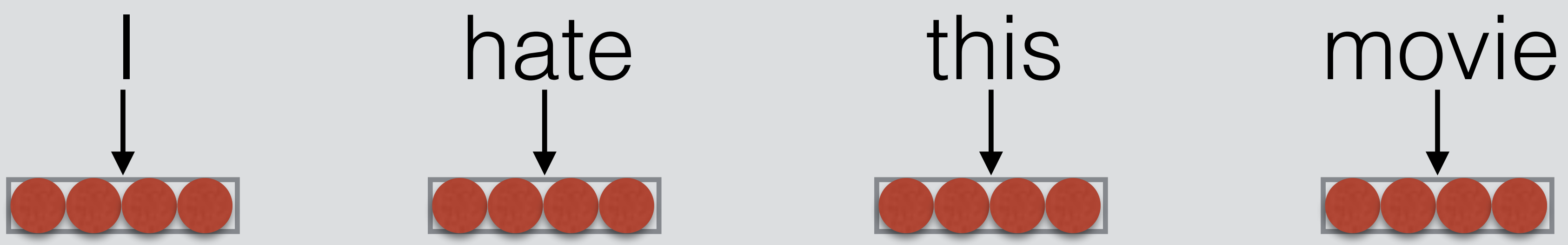


Recurrent NN



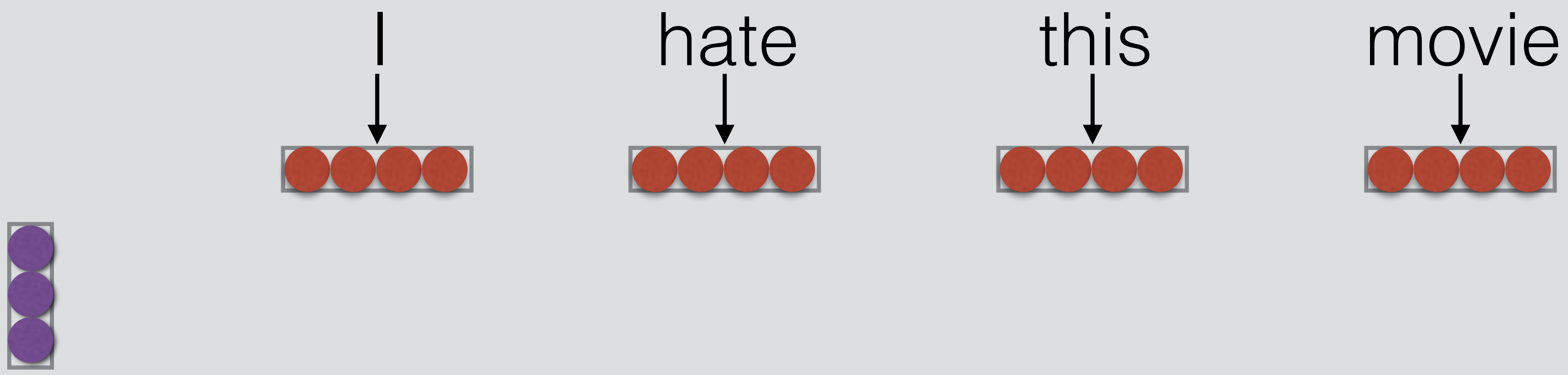
Unrolling in Time

- What does processing a sequence look like?



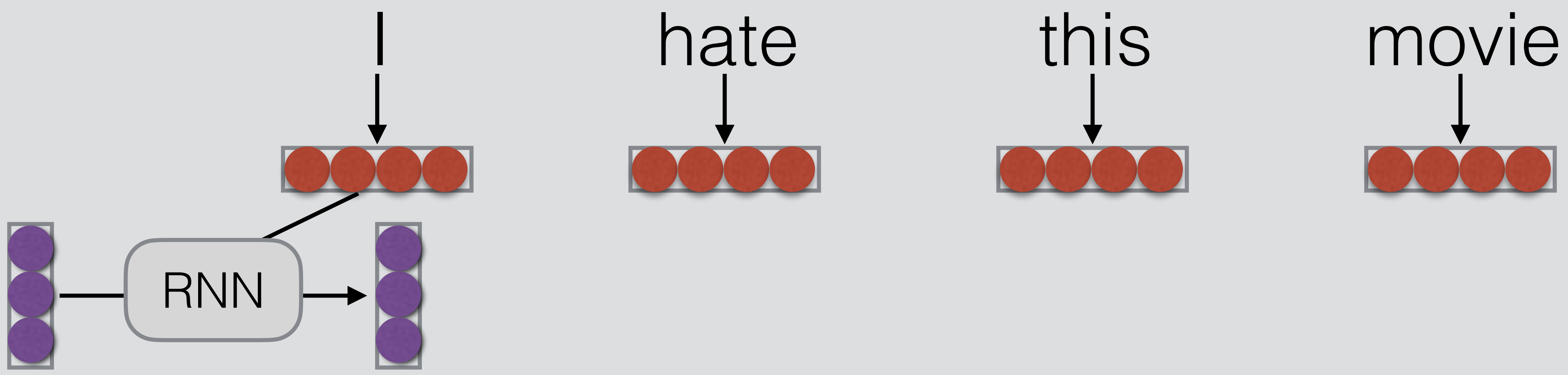
Unrolling in Time

- What does processing a sequence look like?



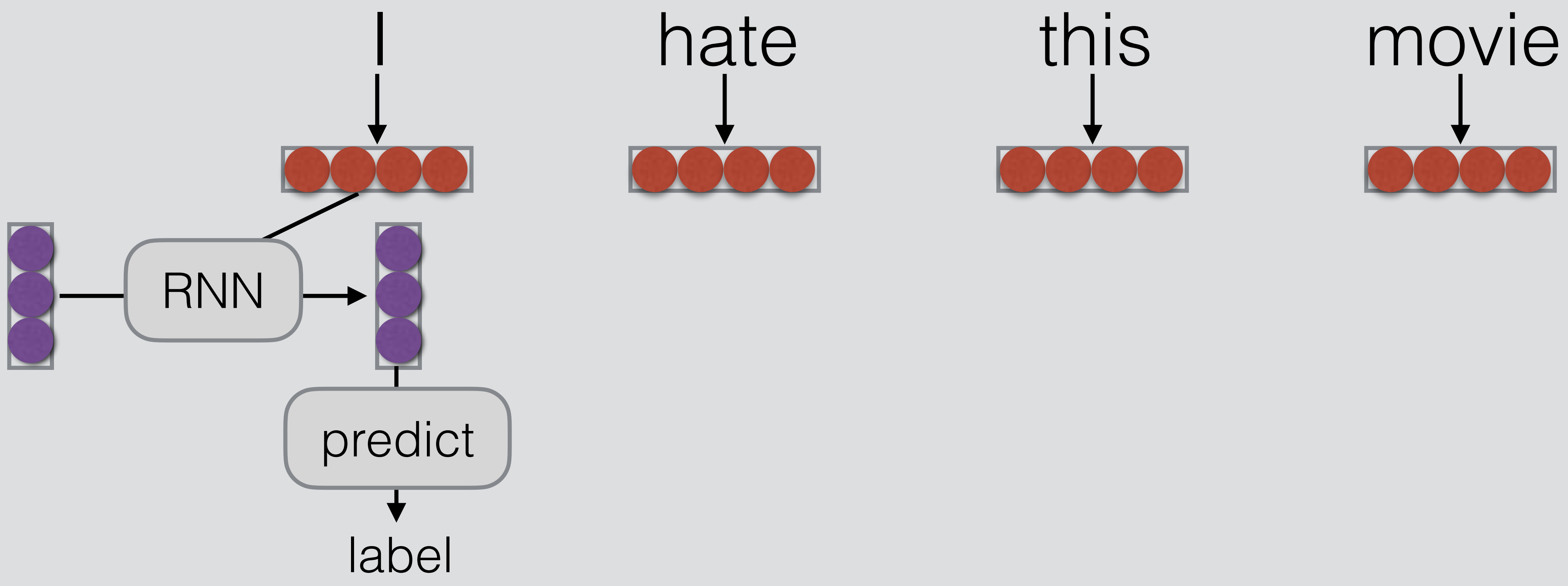
Unrolling in Time

- What does processing a sequence look like?



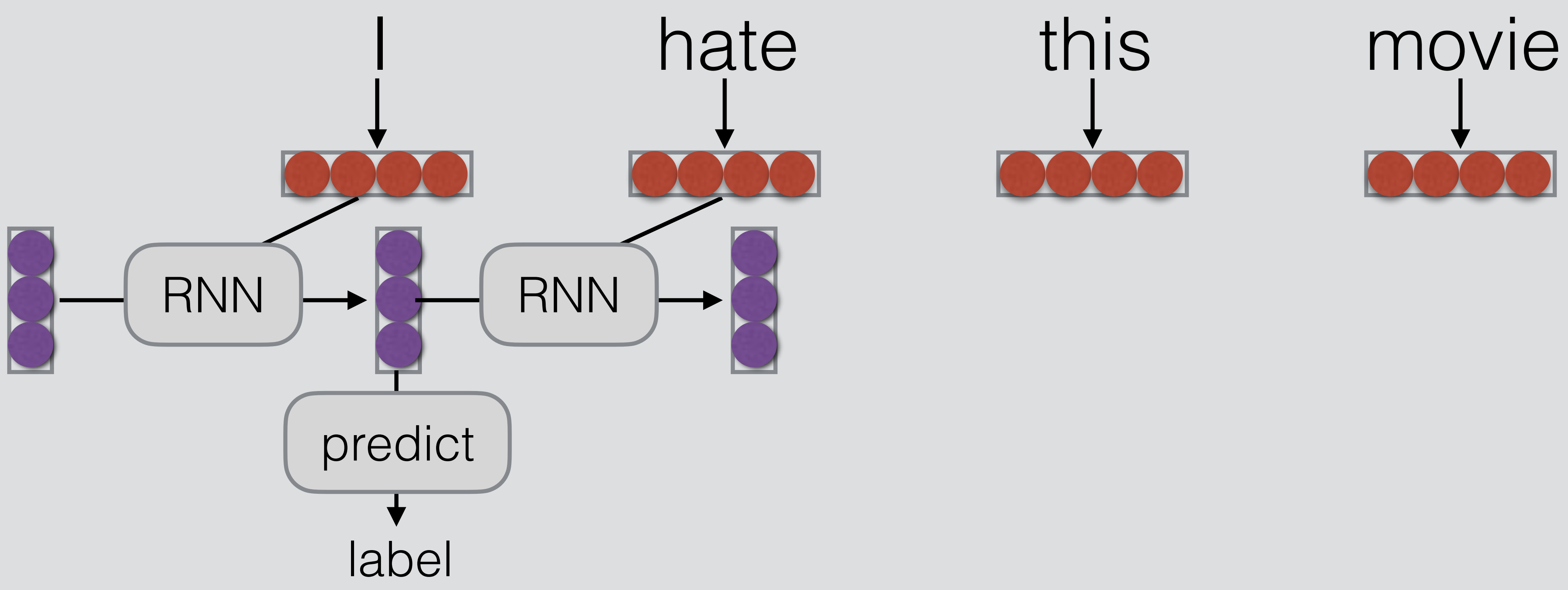
Unrolling in Time

- What does processing a sequence look like?



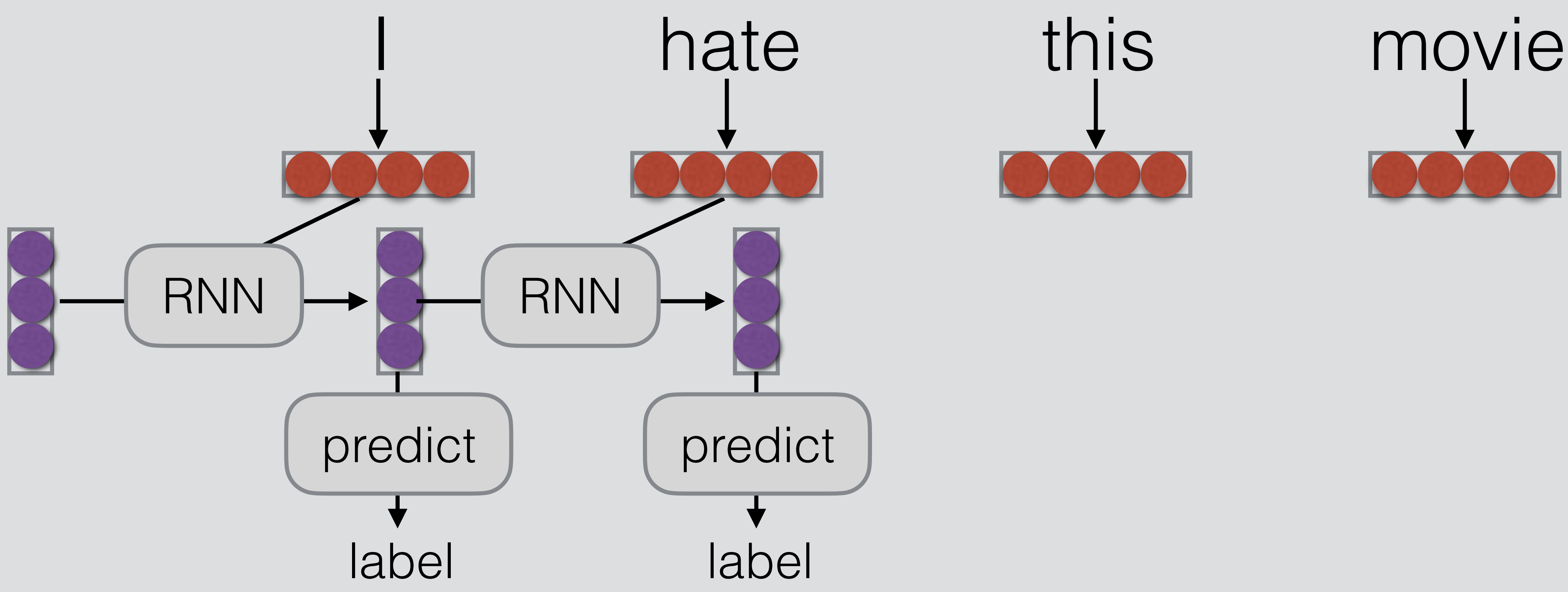
Unrolling in Time

- What does processing a sequence look like?



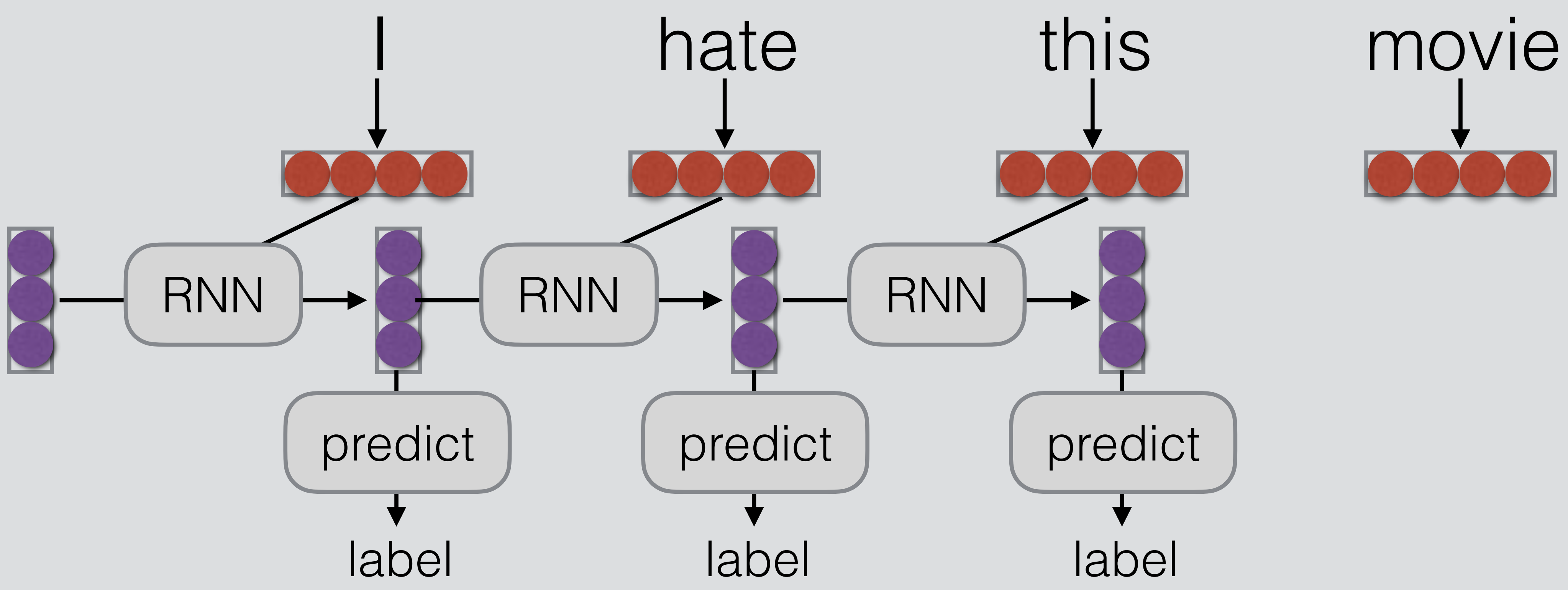
Unrolling in Time

- What does processing a sequence look like?



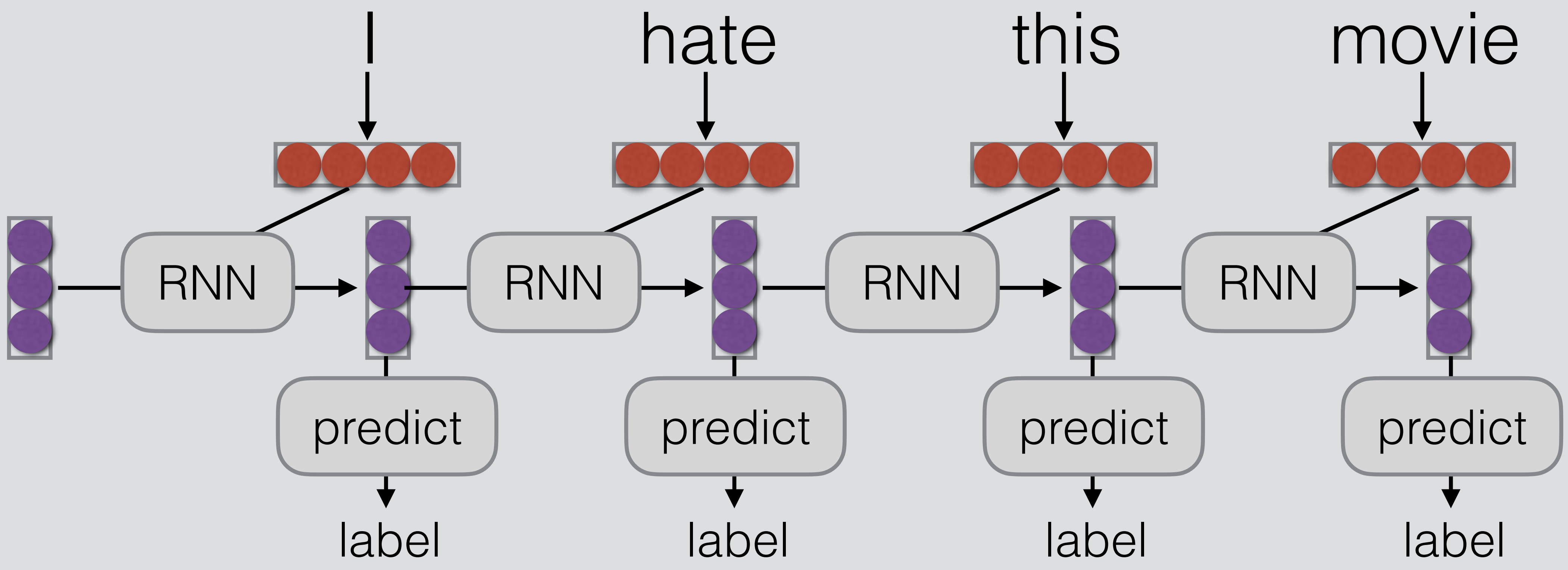
Unrolling in Time

- What does processing a sequence look like?

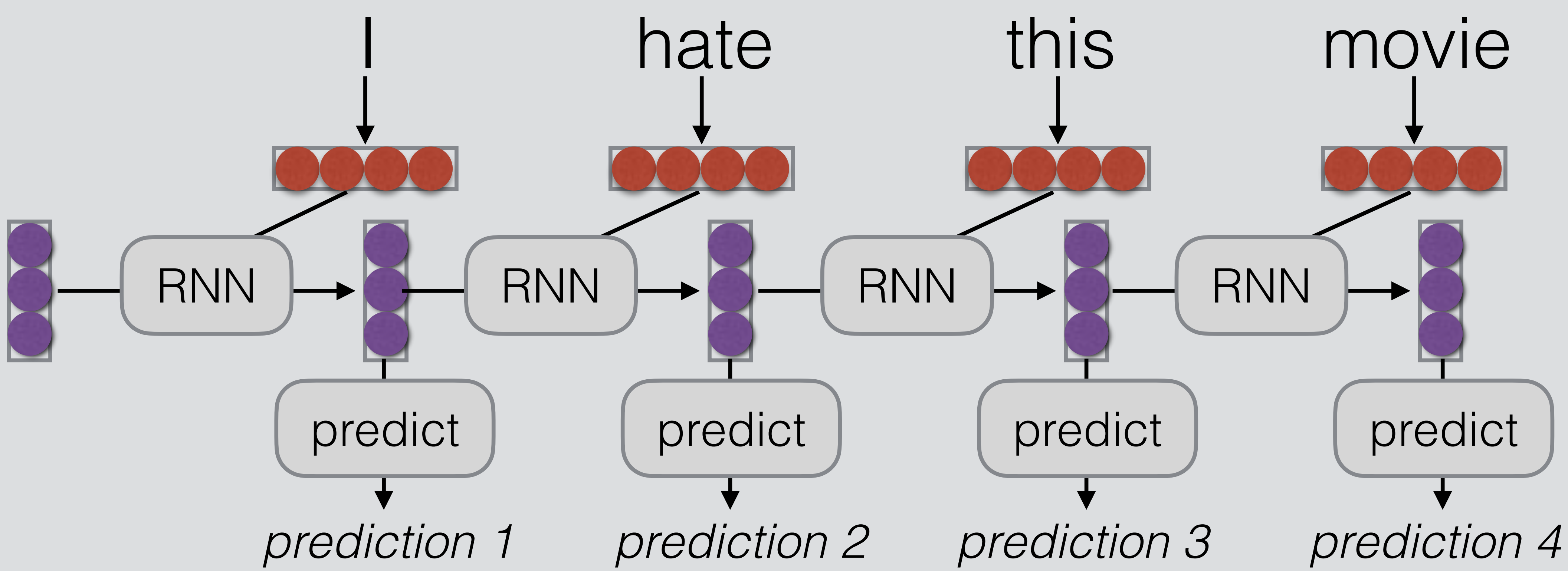


Unrolling in Time

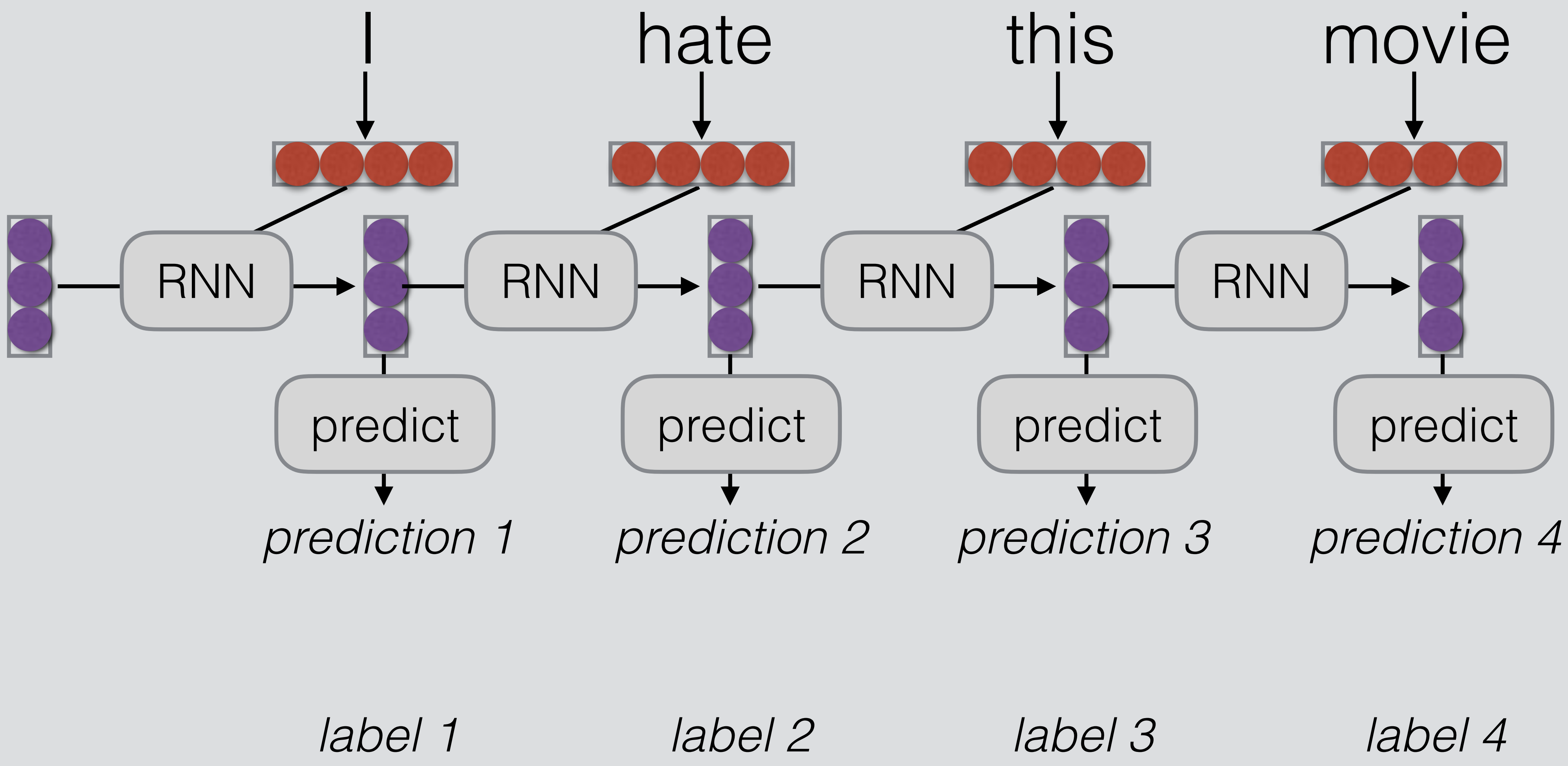
- What does processing a sequence look like?



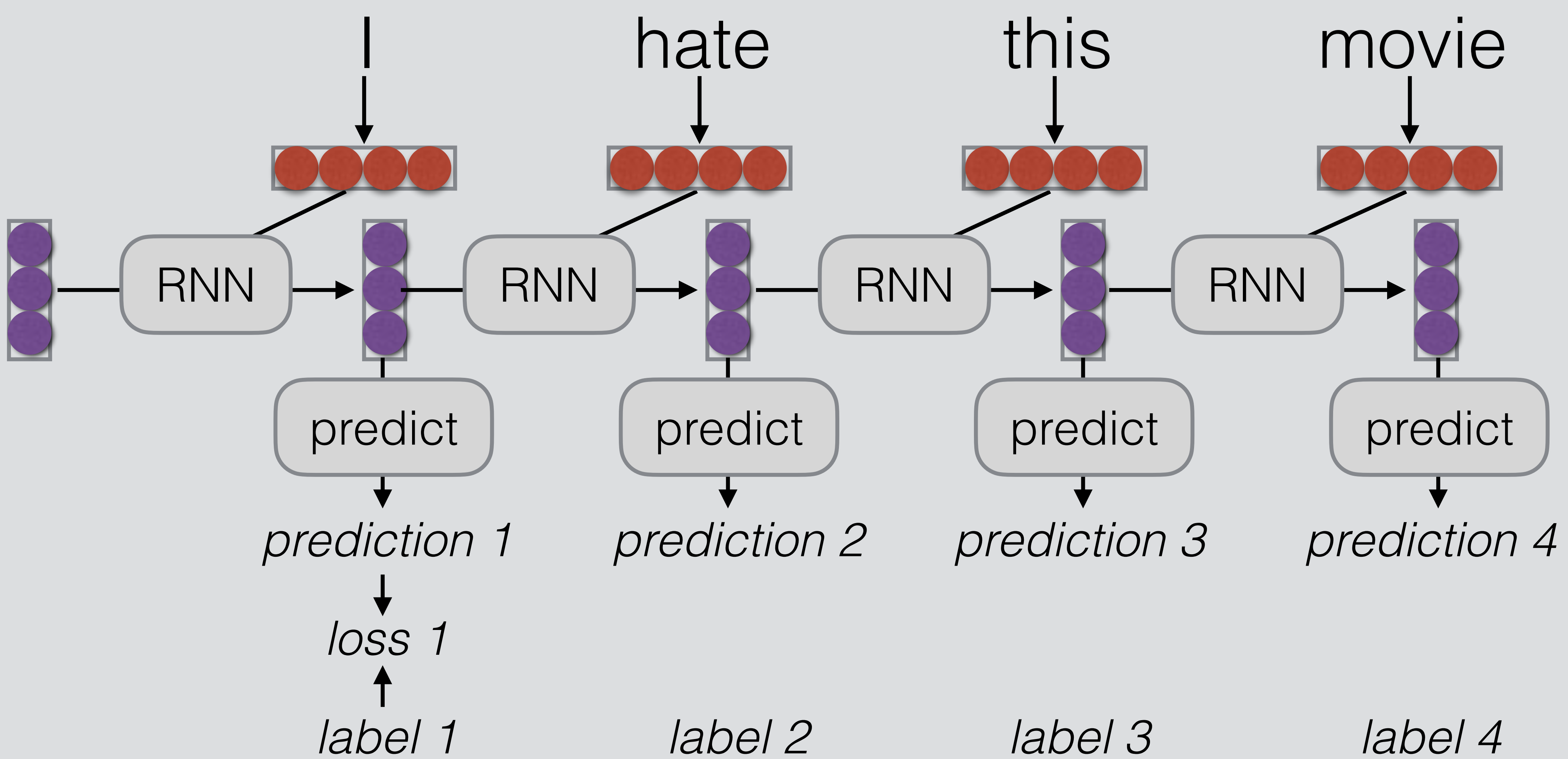
Training RNNs



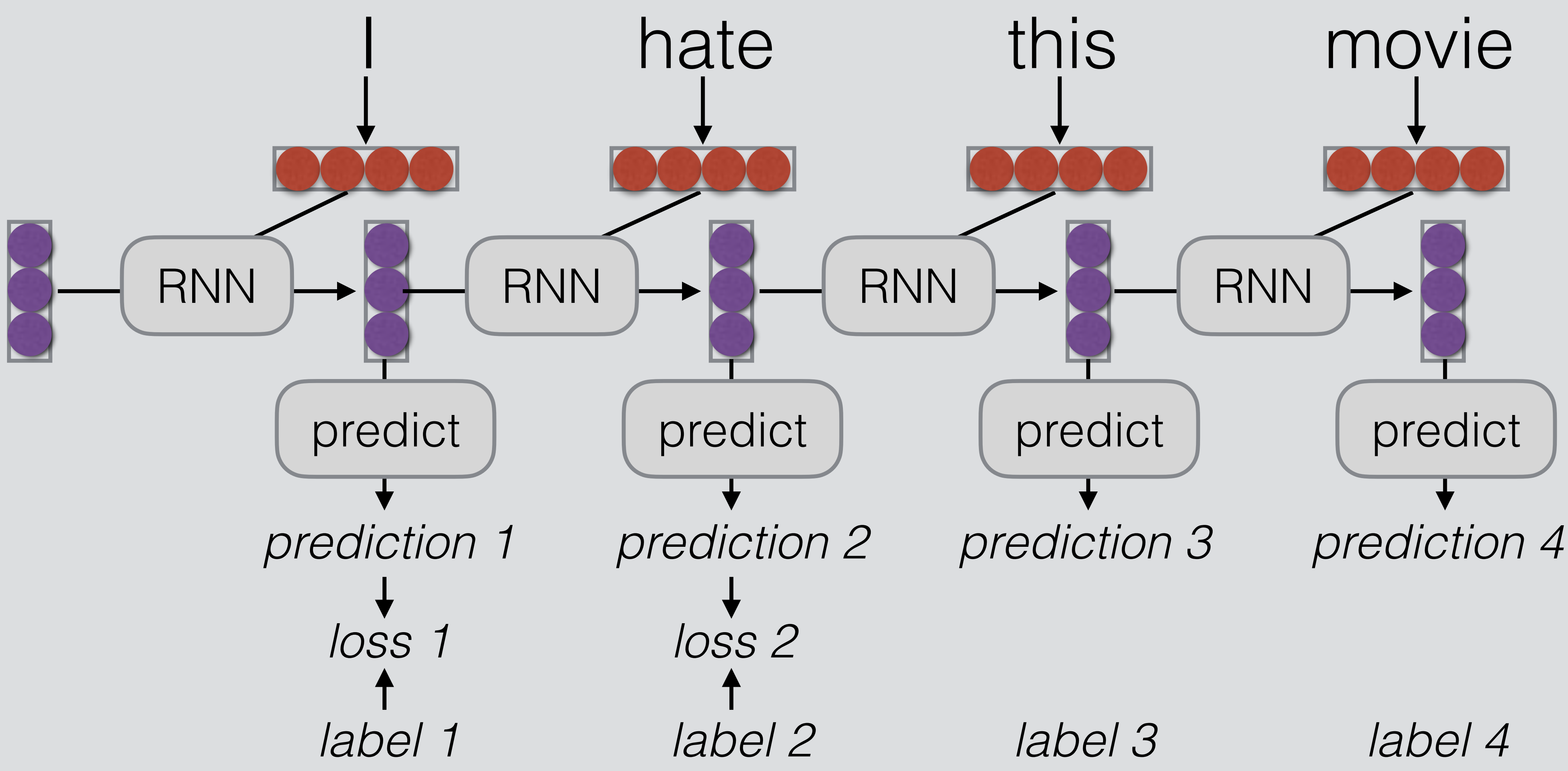
Training RNNs



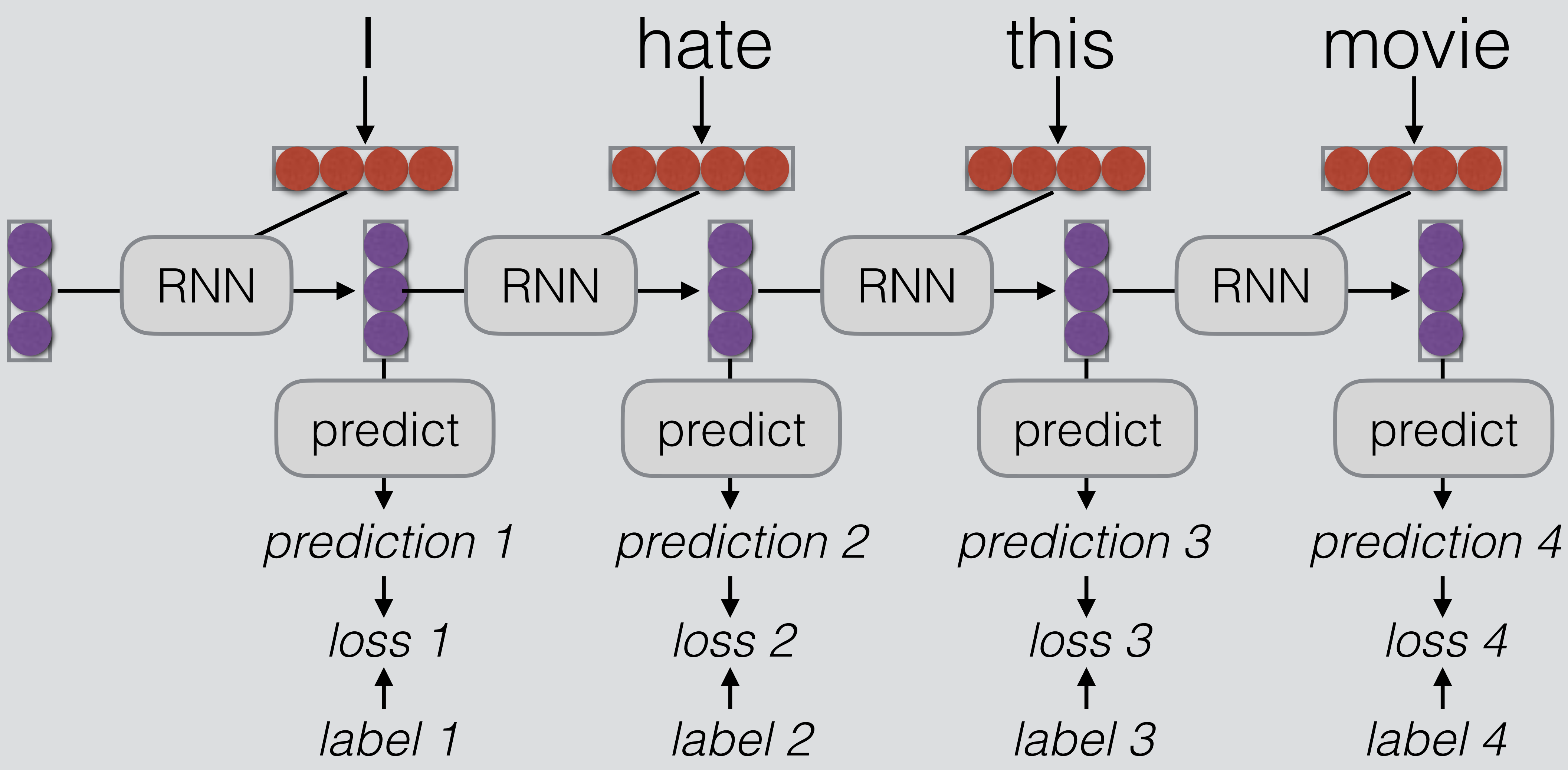
Training RNNs



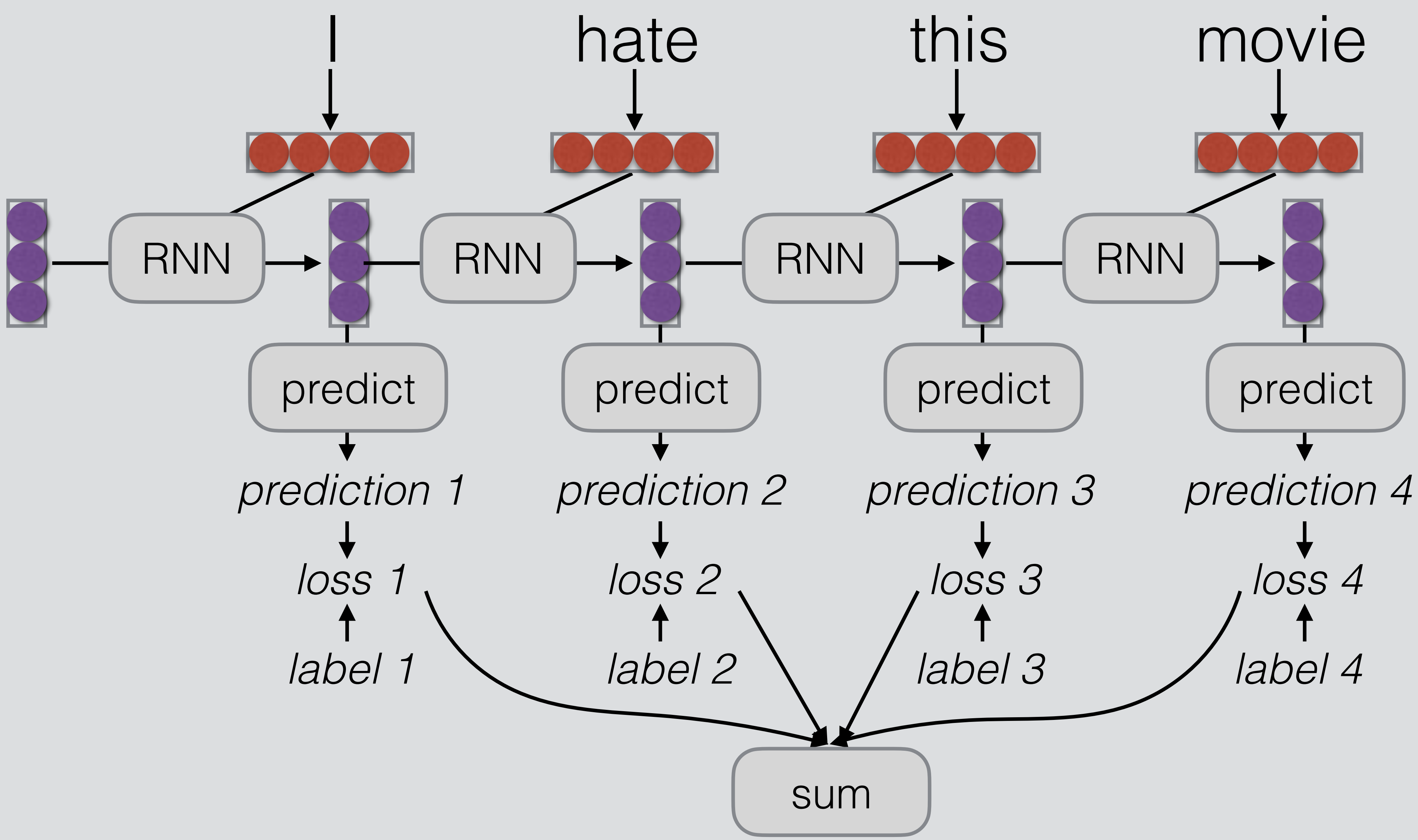
Training RNNs



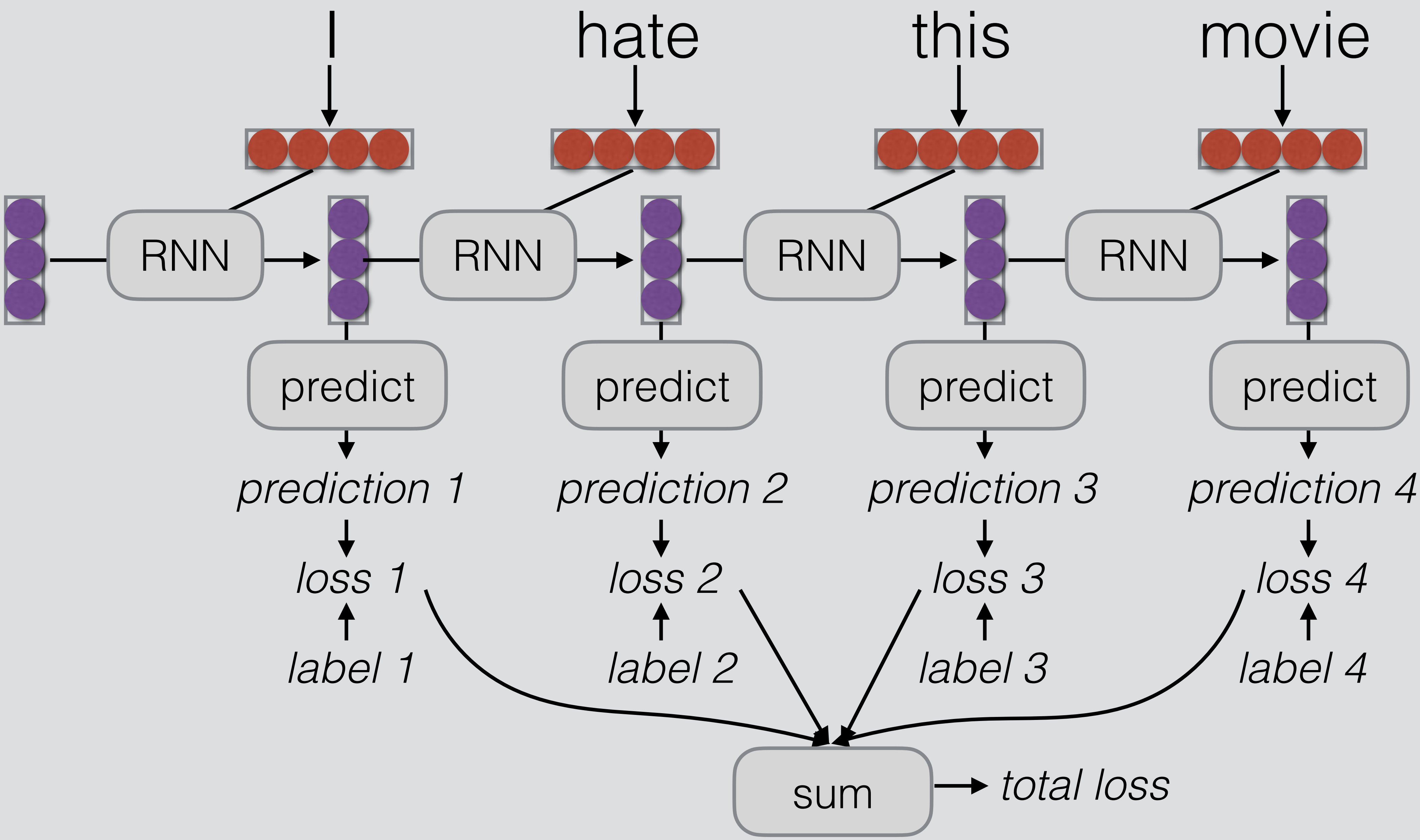
Training RNNs



Training RNNs



Training RNNs

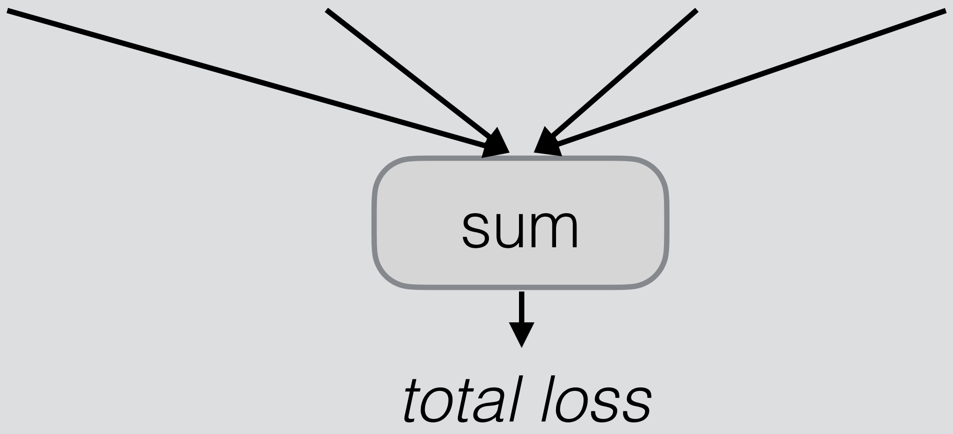


RNN Training

- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop

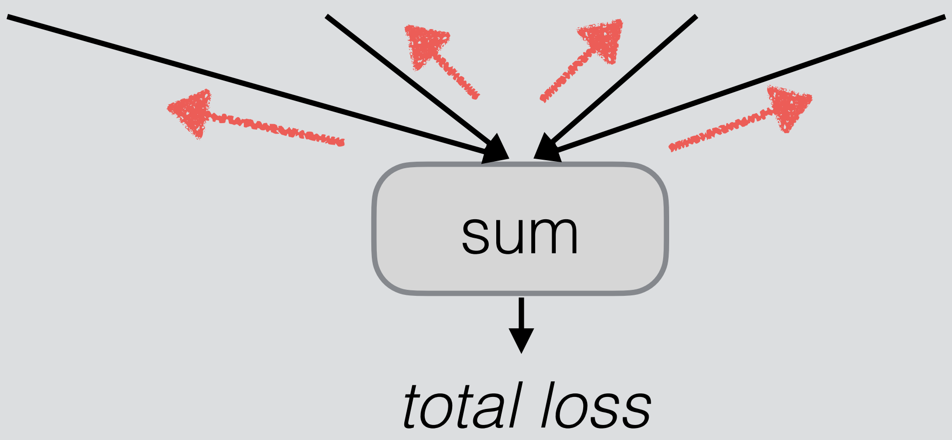
RNN Training

- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop



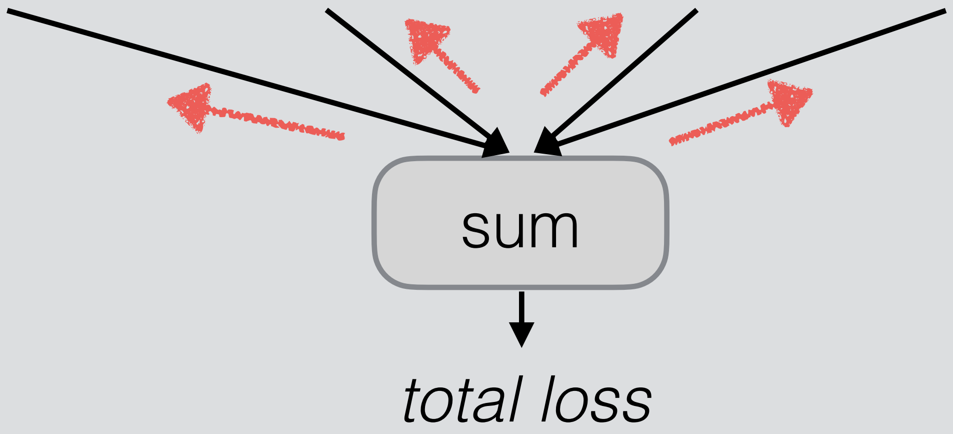
RNN Training

- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop



RNN Training

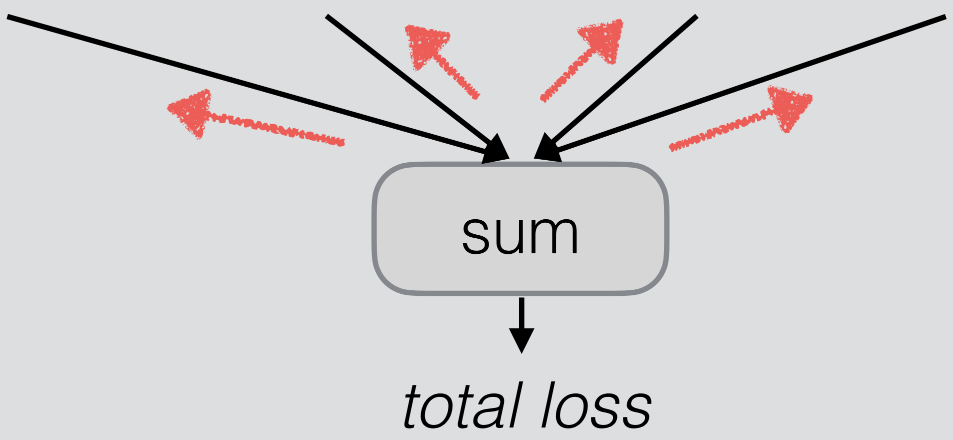
- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop



- Parameters are tied across time, derivatives are aggregated across all time steps

RNN Training

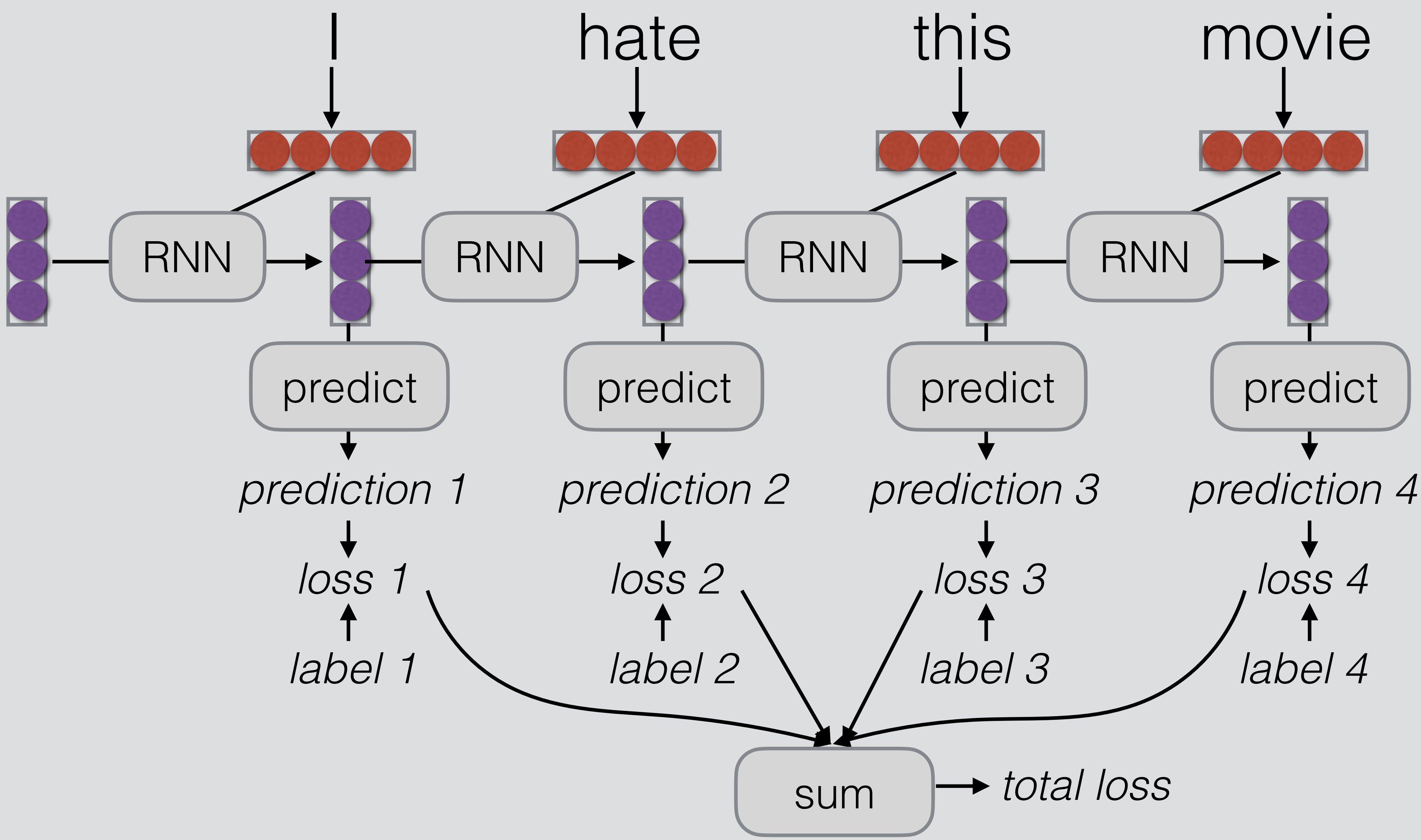
- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop



- Parameters are tied across time, derivatives are aggregated across all time steps

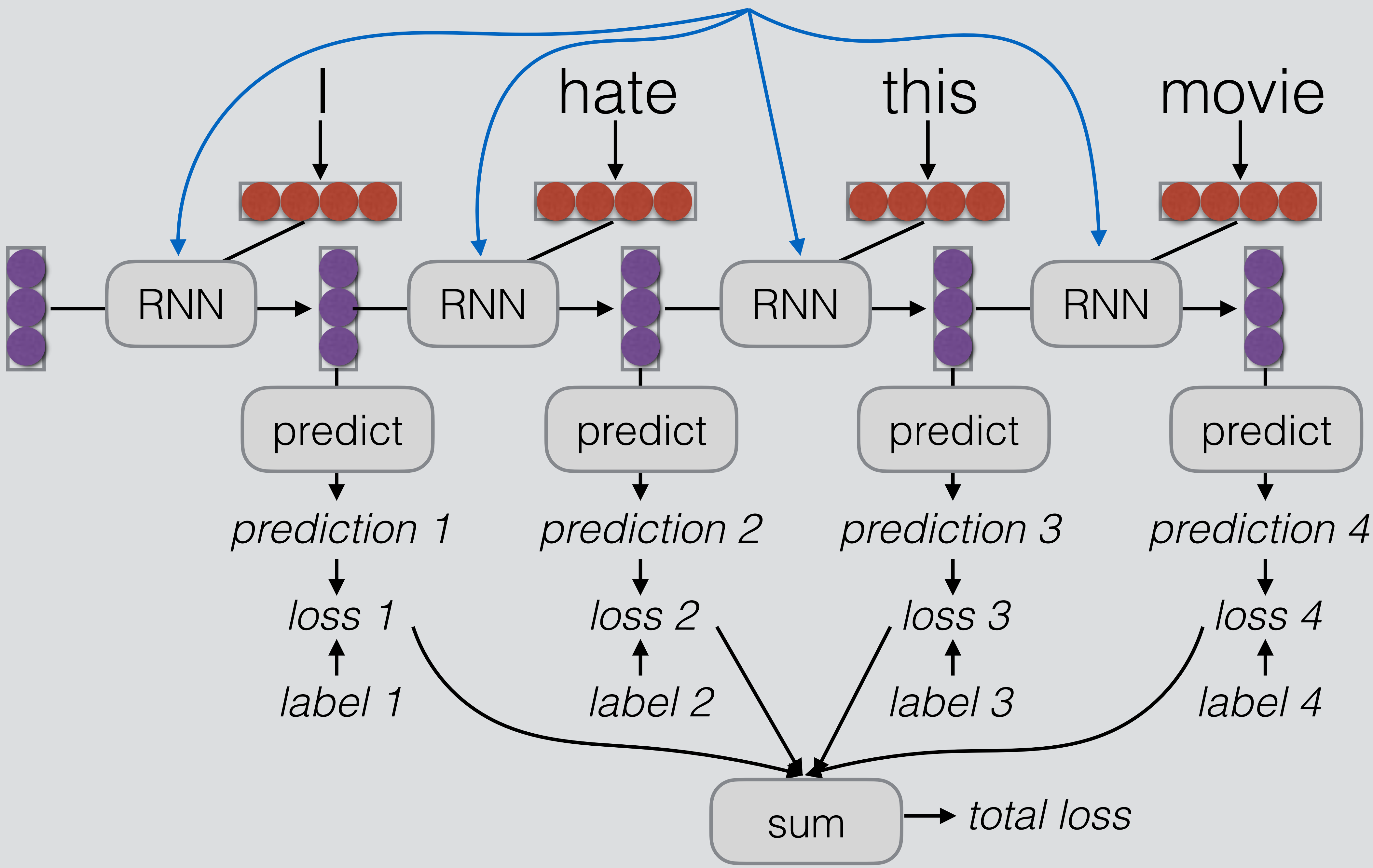
- This is historically called “backpropagation through time” (BPTT)

Parameter Tying



Parameter Tying

Parameters are shared! Derivatives are accumulated.

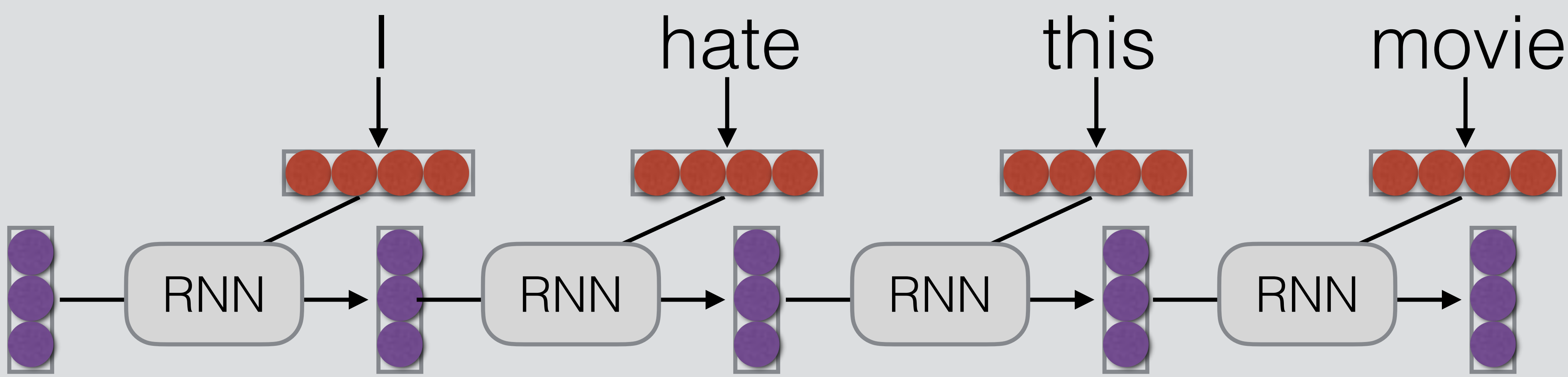


Applications of RNNs

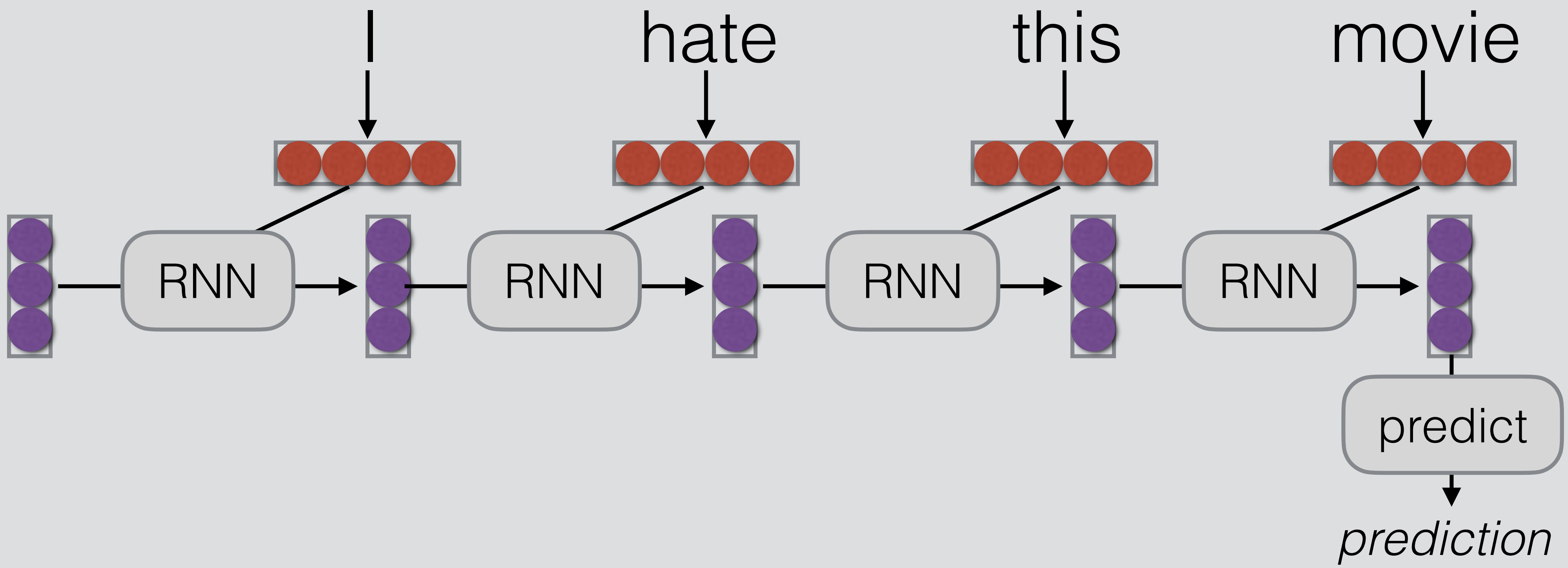
What Can RNNs Do?

- Represent a sentence
  - Read whole sentence, make a prediction
- Represent a context within a sentence
  - Read context up until that point

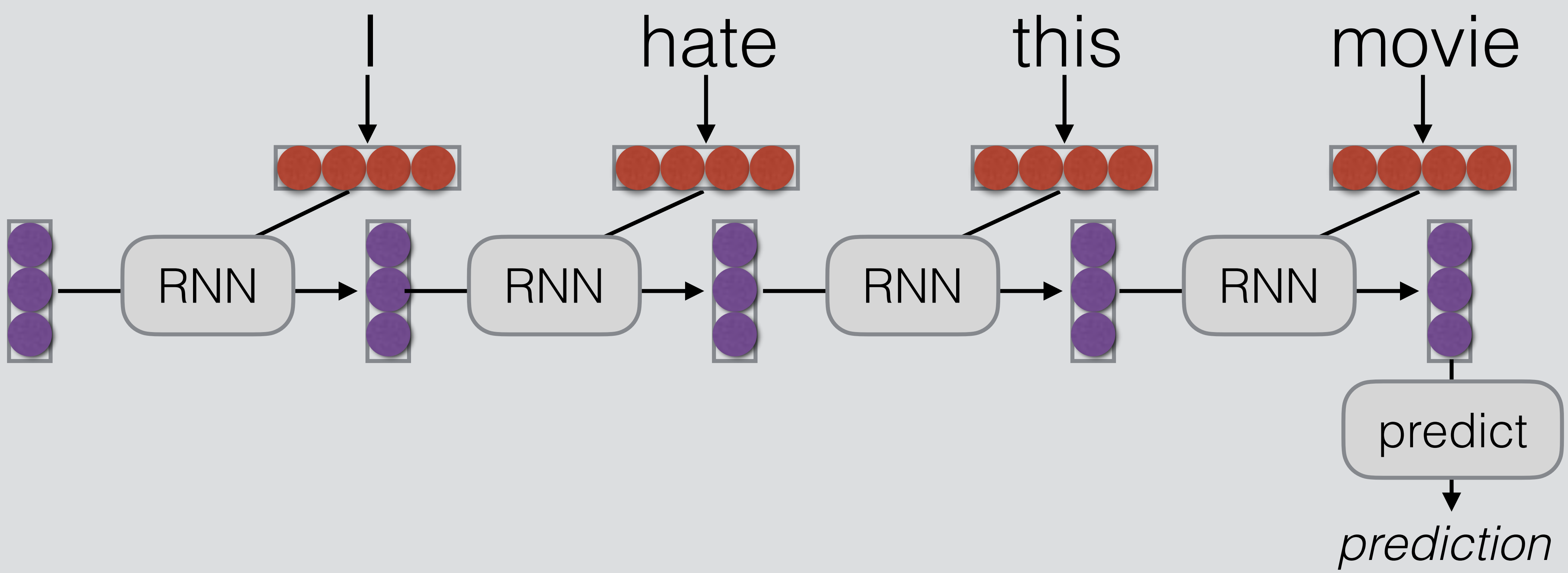
Representing Sentences



Representing Sentences

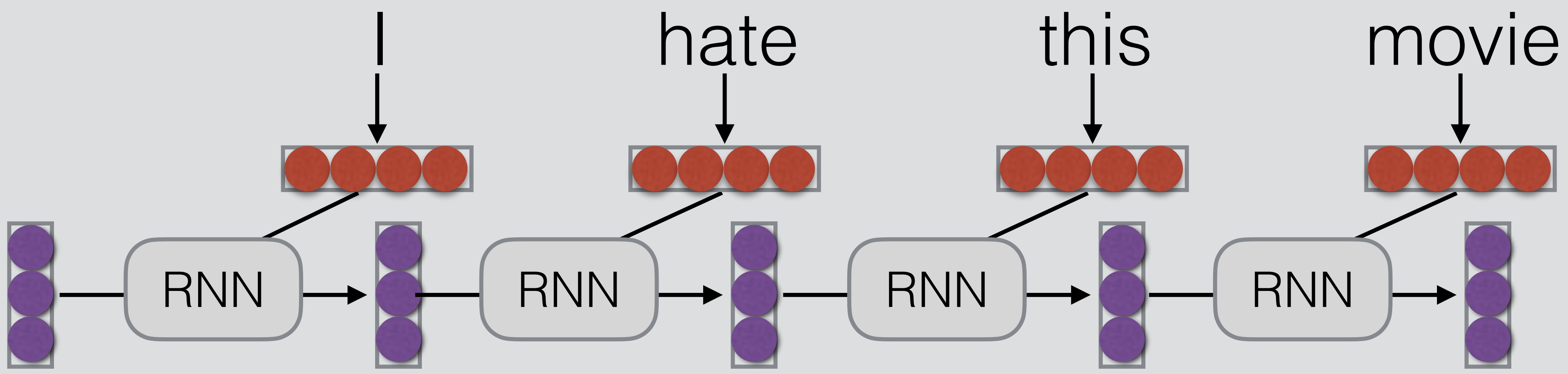


Representing Sentences

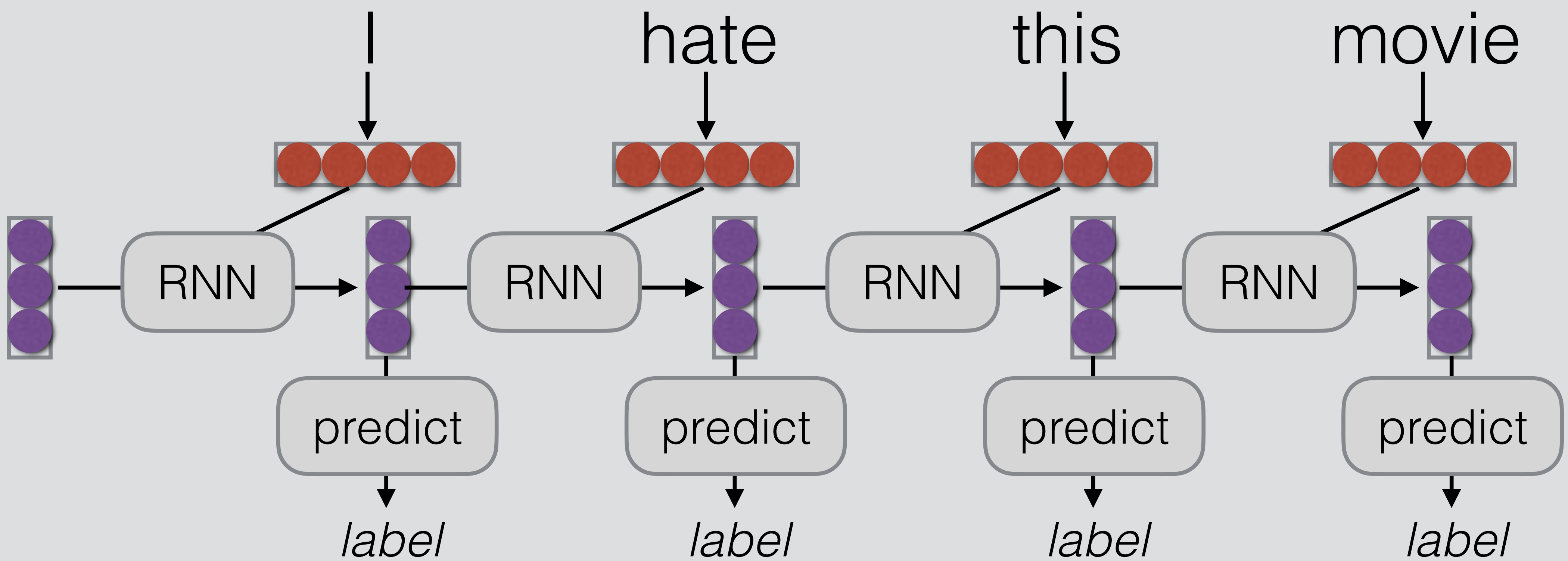


- Sentence classification
- Conditioned generation
- Retrieval

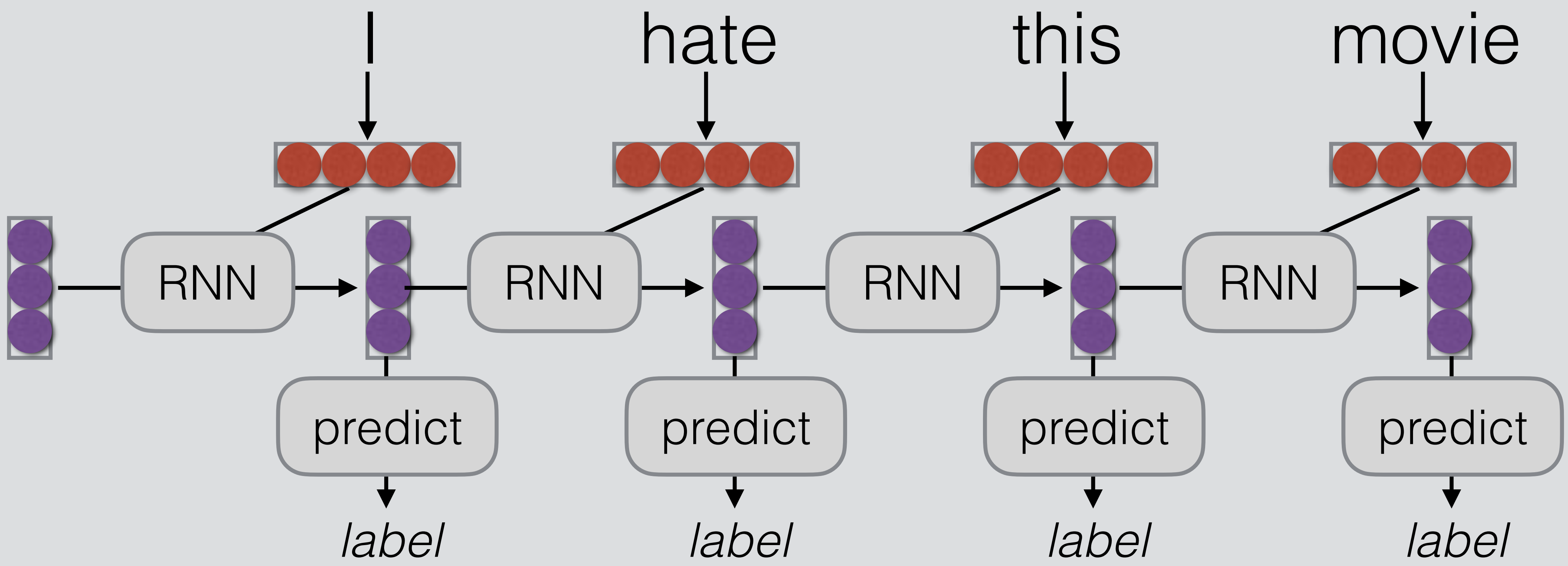
Representing Contexts



Representing Contexts

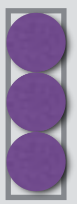


Representing Contexts

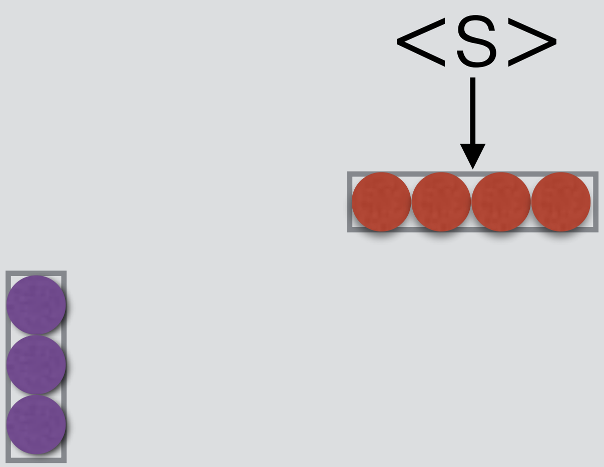


- Tagging
- Language Modeling
- Calculating Representations for Parsing, etc.

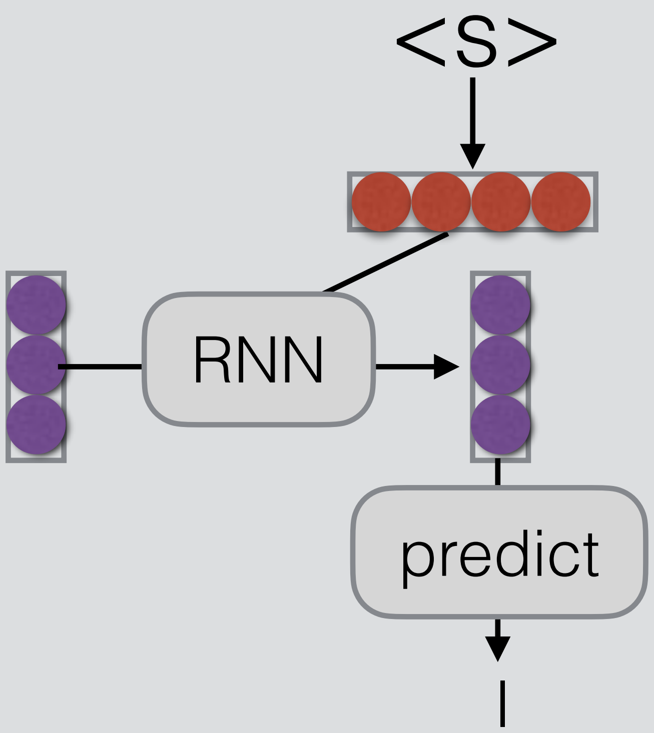
e.g. Language Modeling



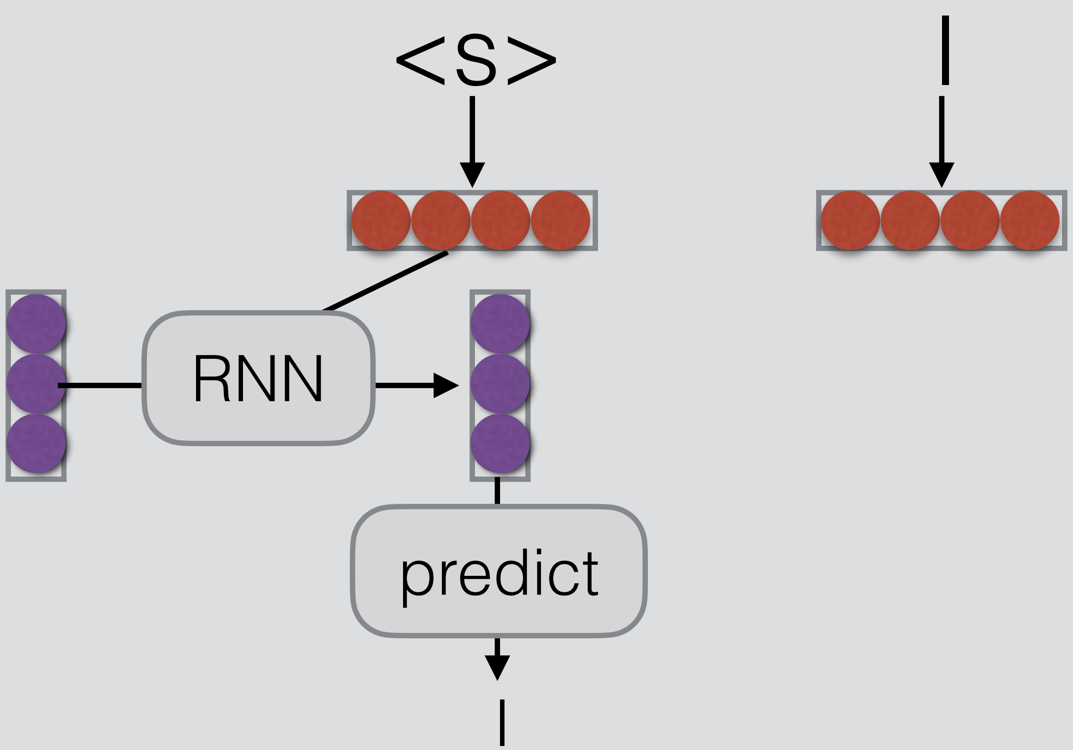
e.g. Language Modeling



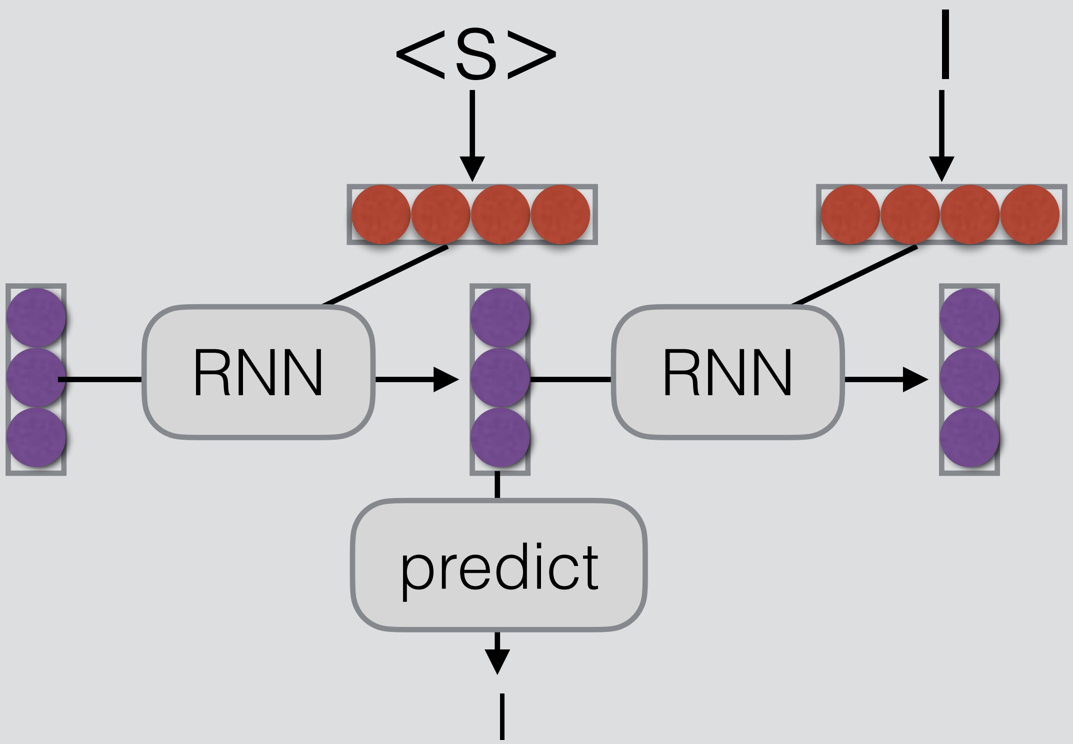
e.g. Language Modeling



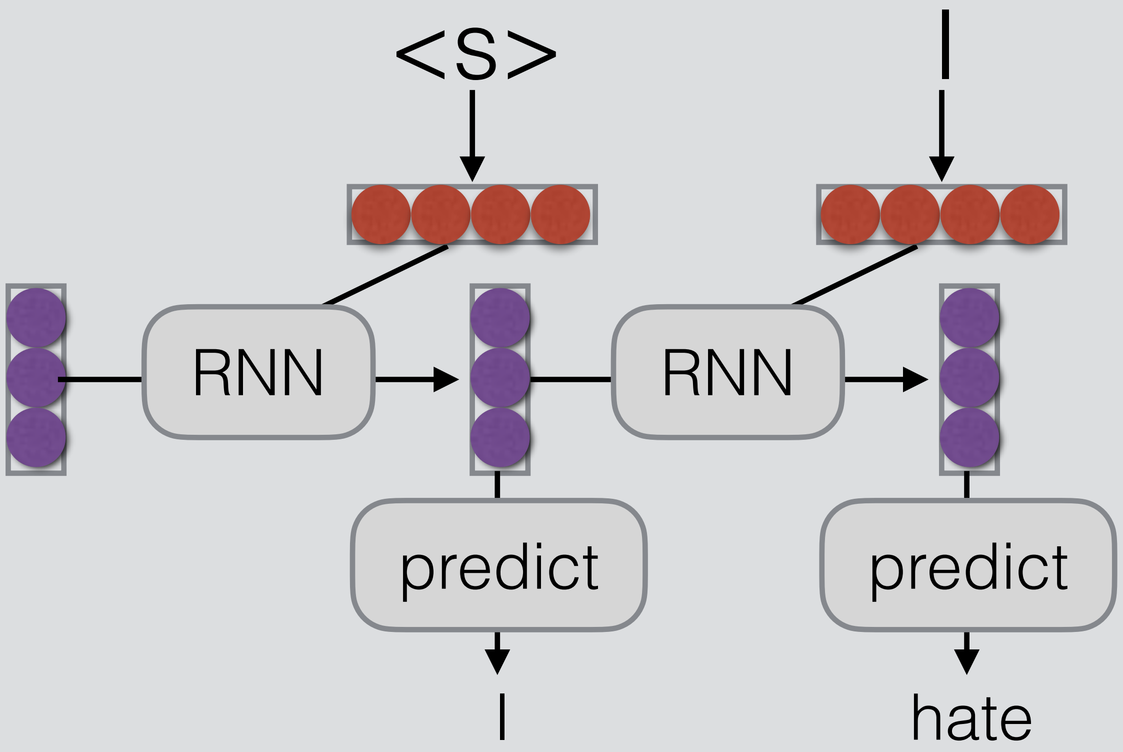
e.g. Language Modeling



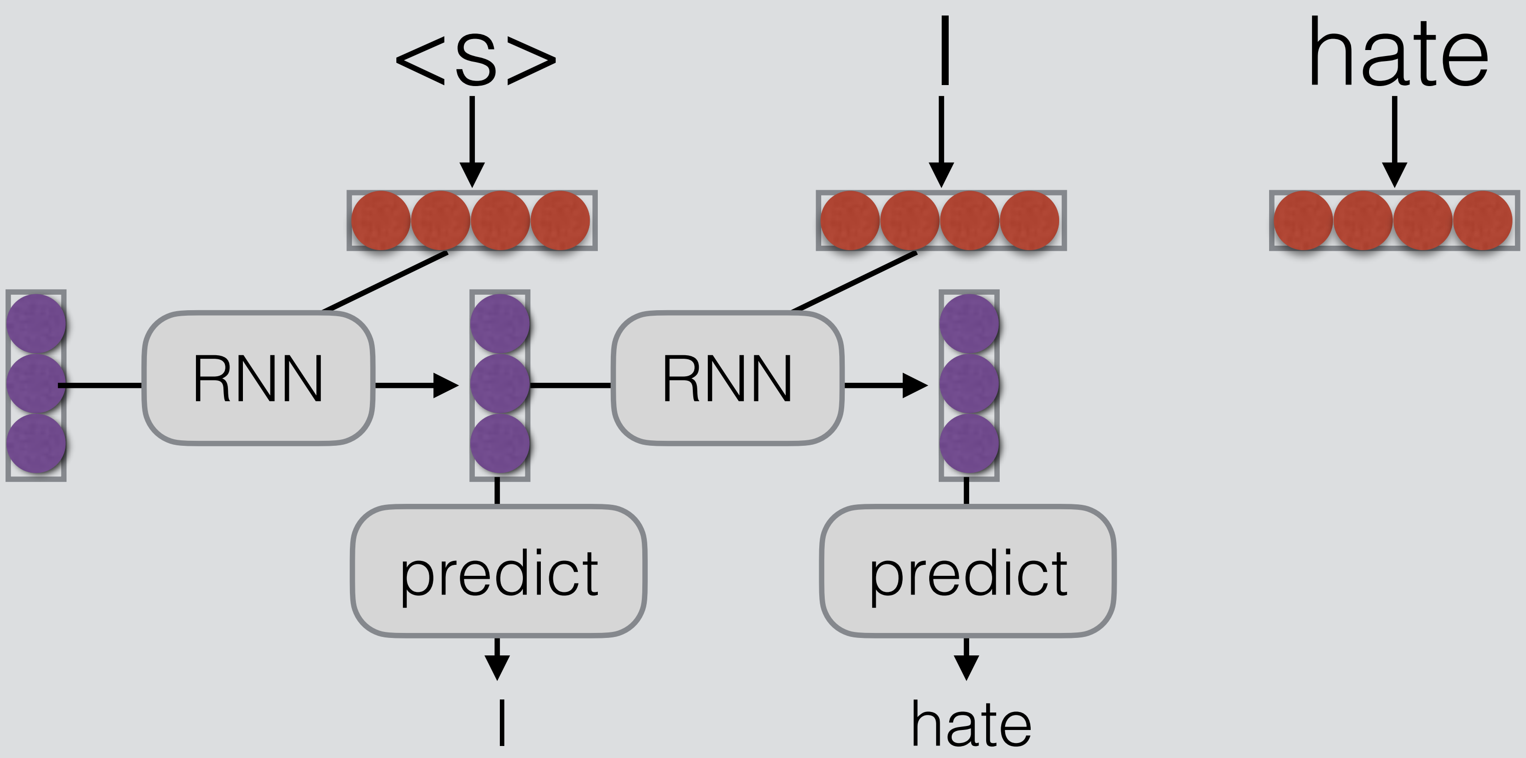
e.g. Language Modeling



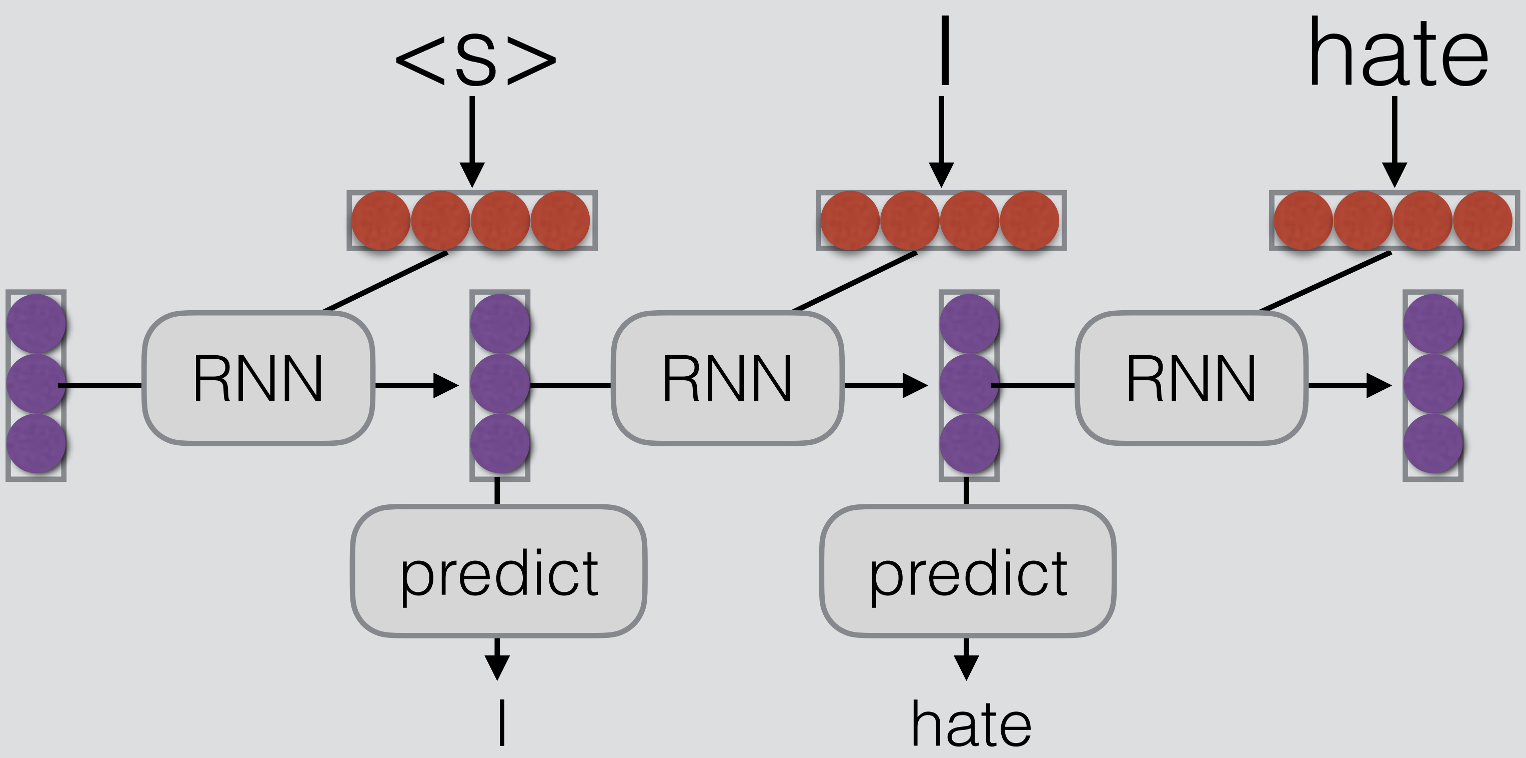
e.g. Language Modeling



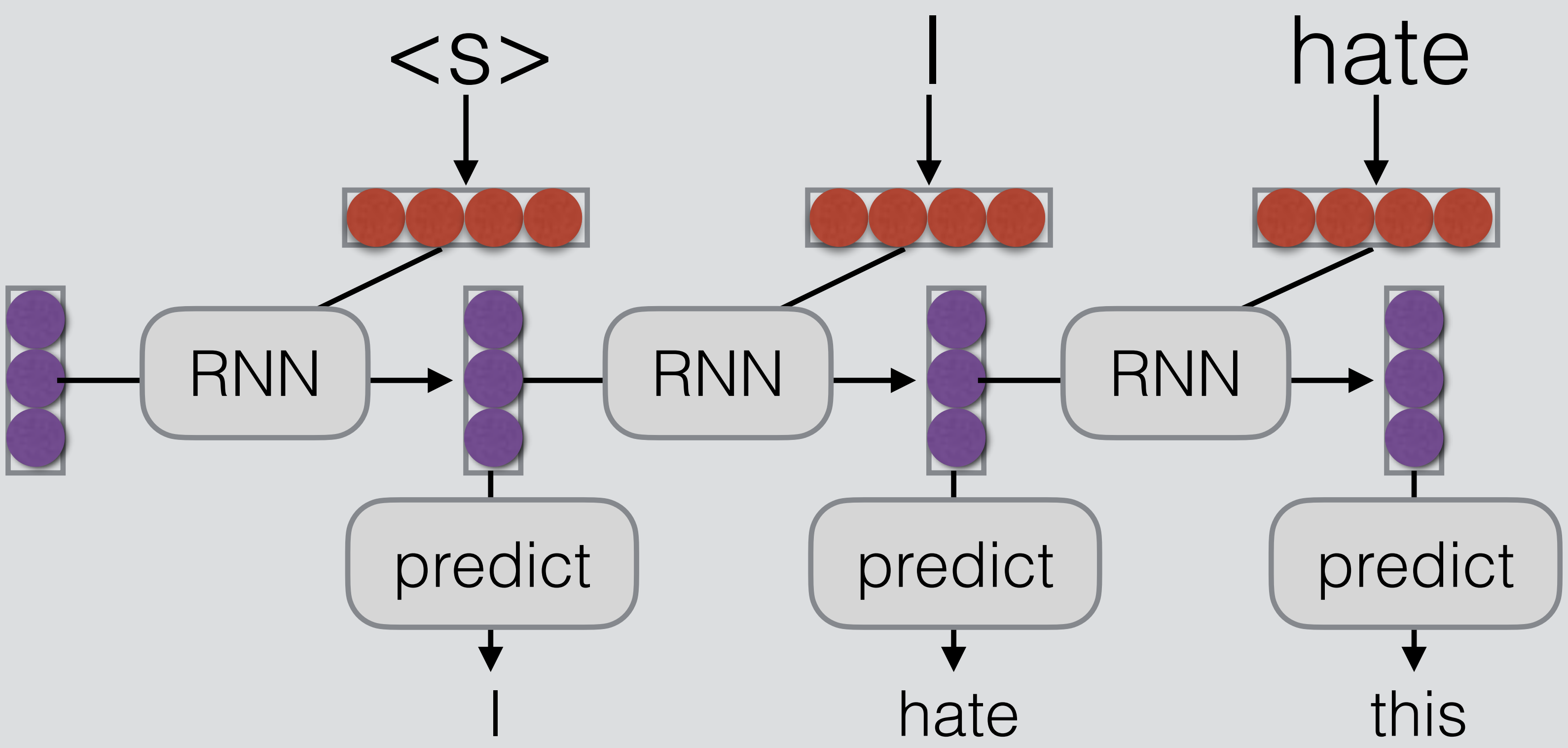
e.g. Language Modeling



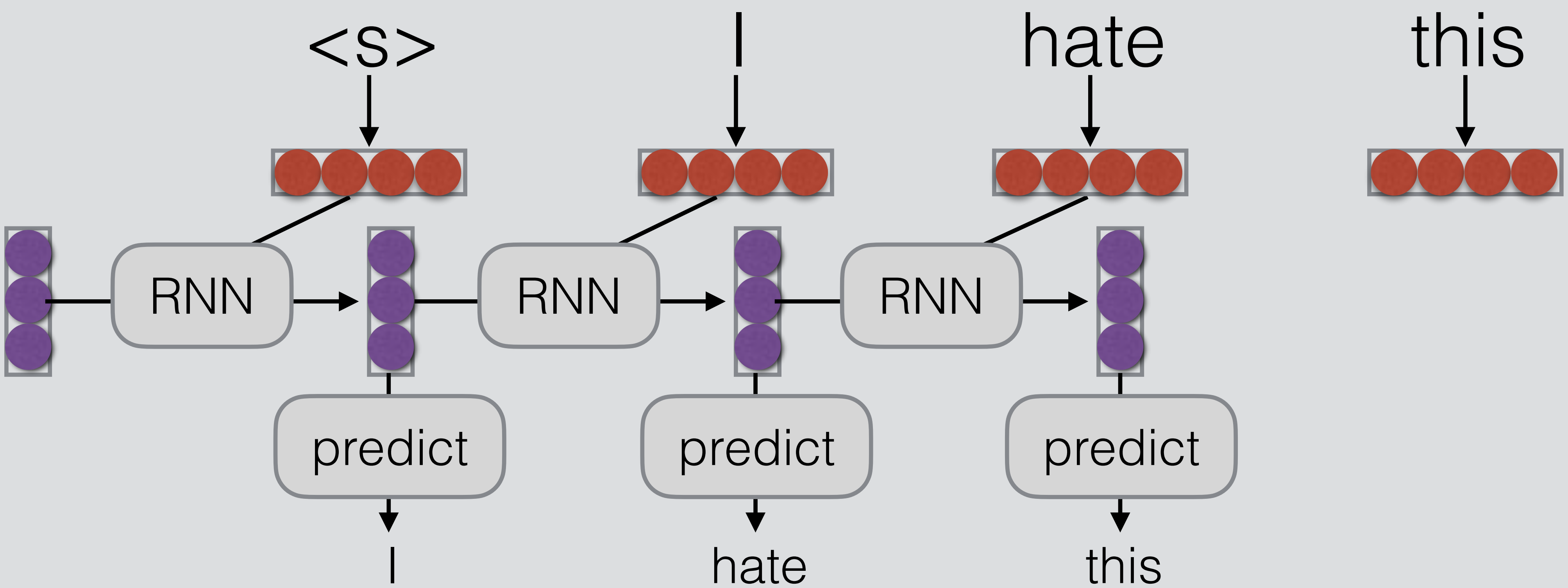
e.g. Language Modeling



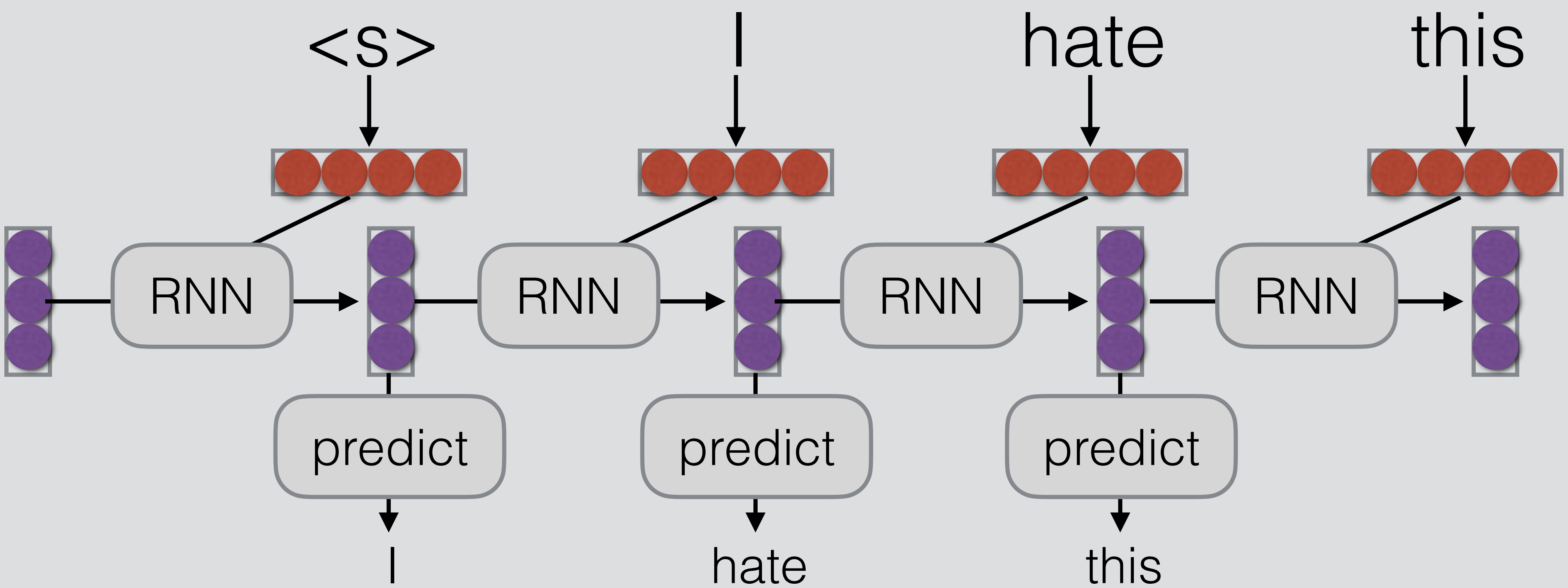
e.g. Language Modeling



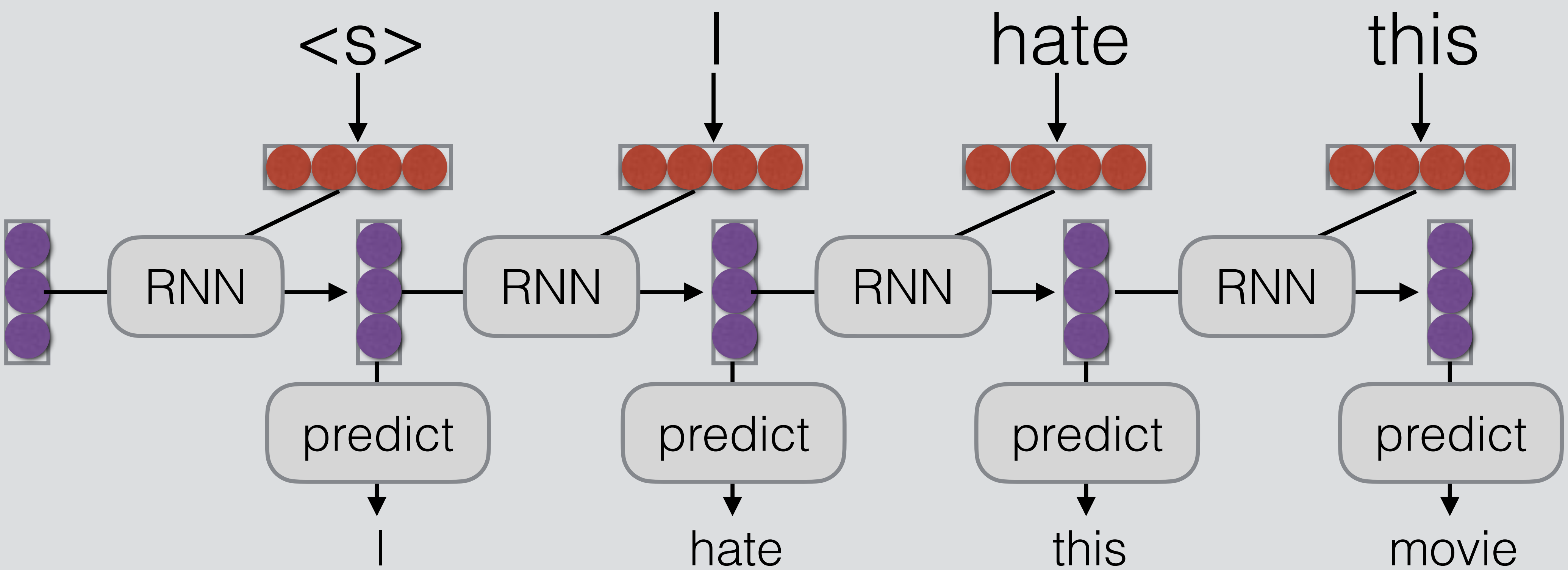
e.g. Language Modeling



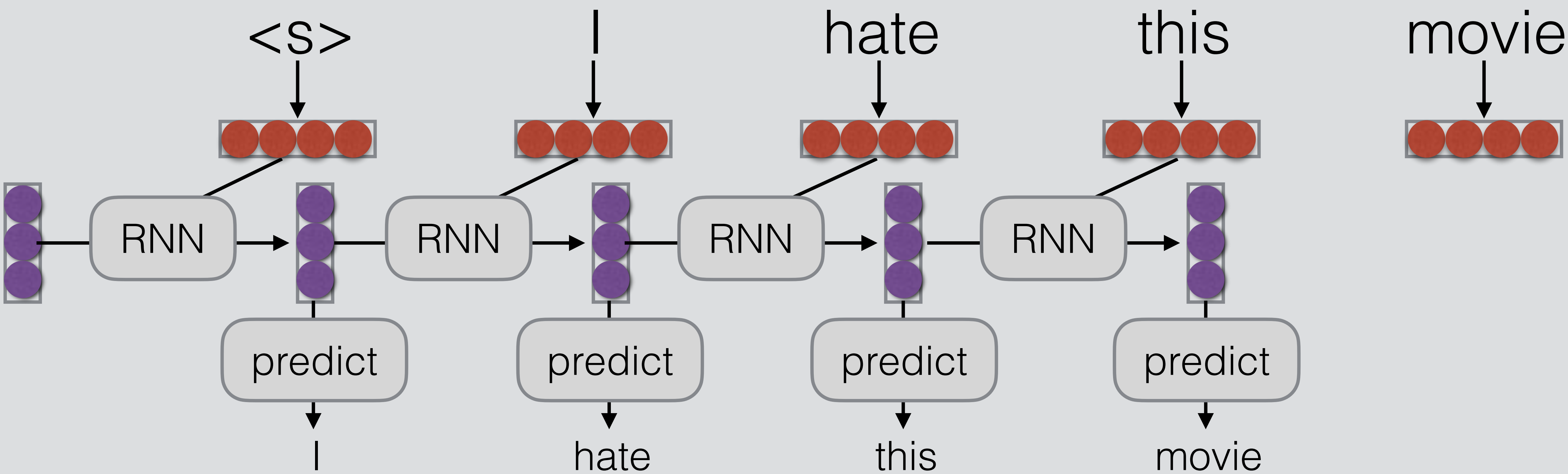
e.g. Language Modeling



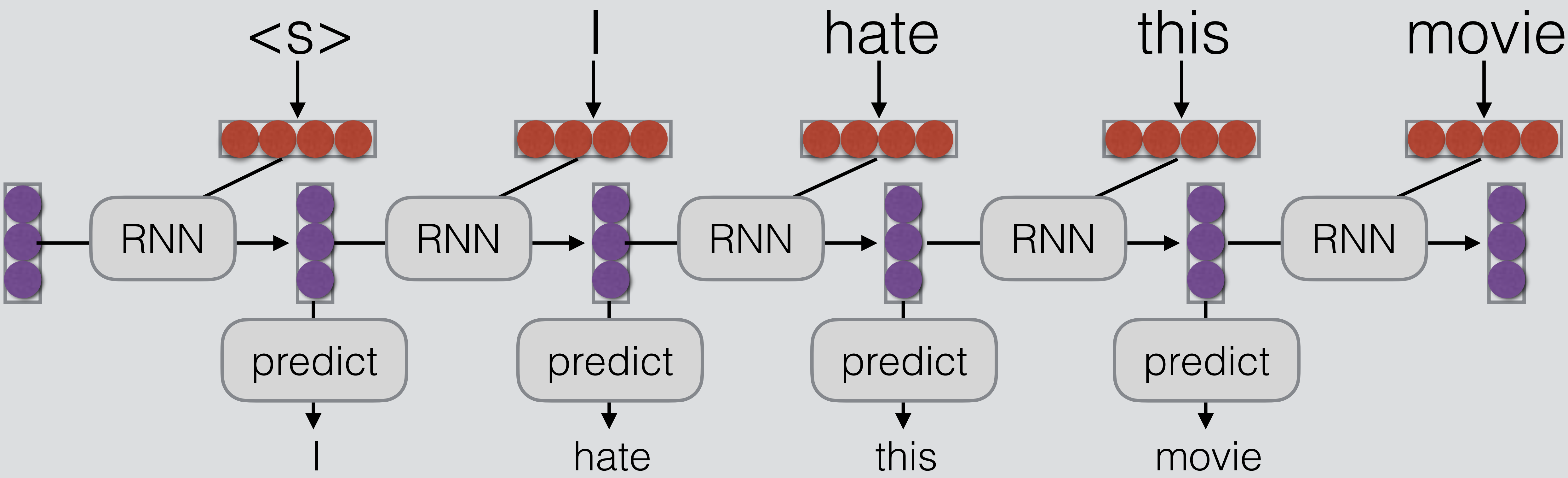
e.g. Language Modeling



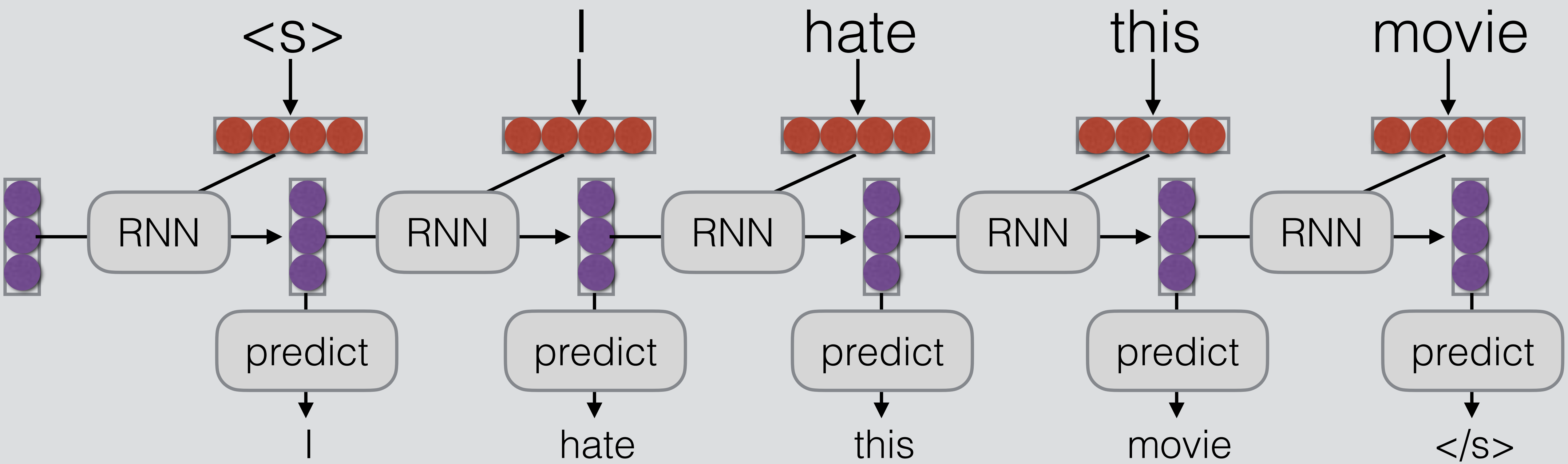
e.g. Language Modeling



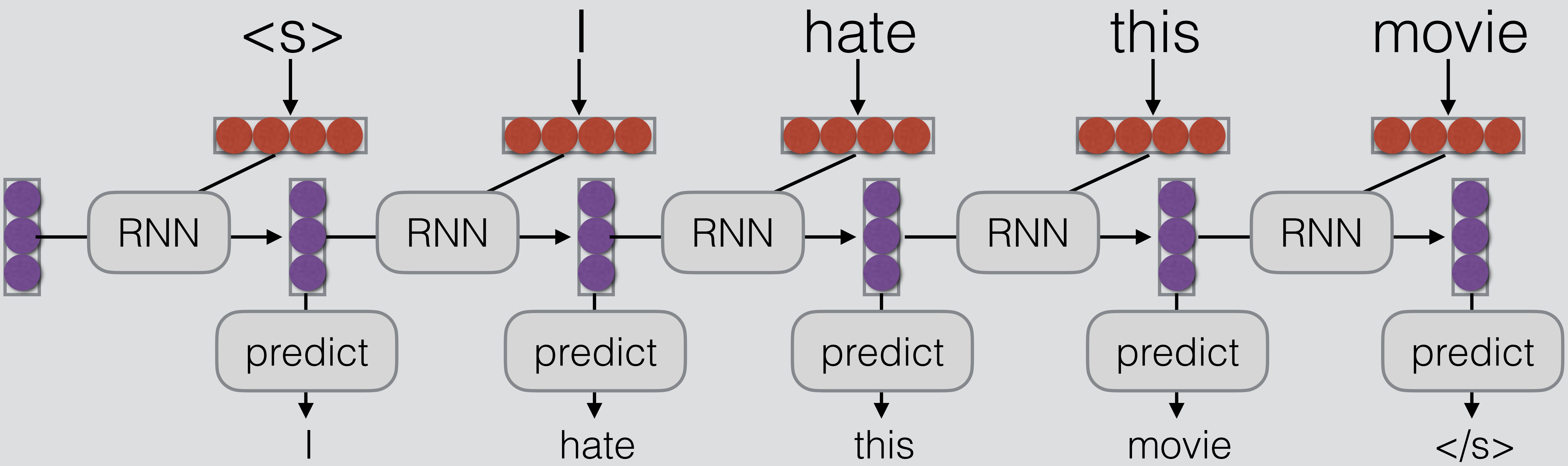
e.g. Language Modeling



e.g. Language Modeling



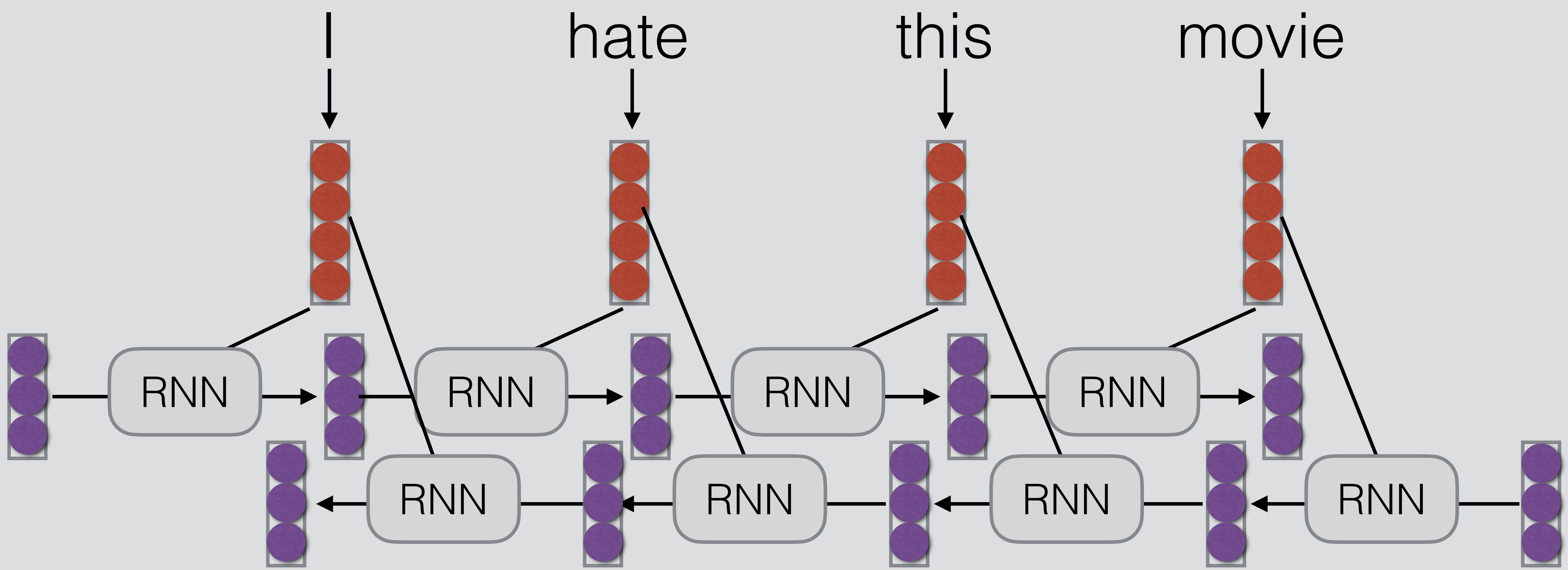
e.g. Language Modeling



- Language modeling is like a tagging task, where each tag is the next word!

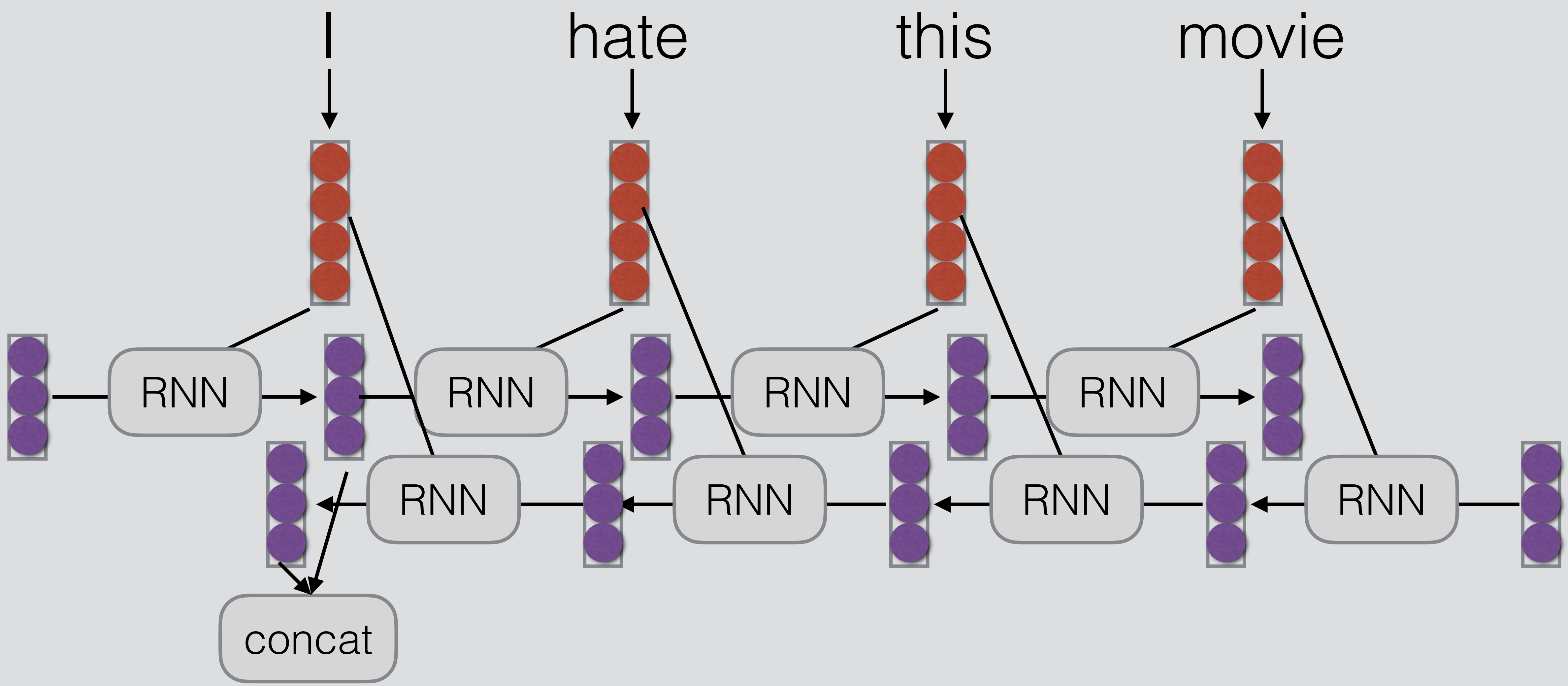
Bi-RNNs

- A simple extension, run the RNN in both directions



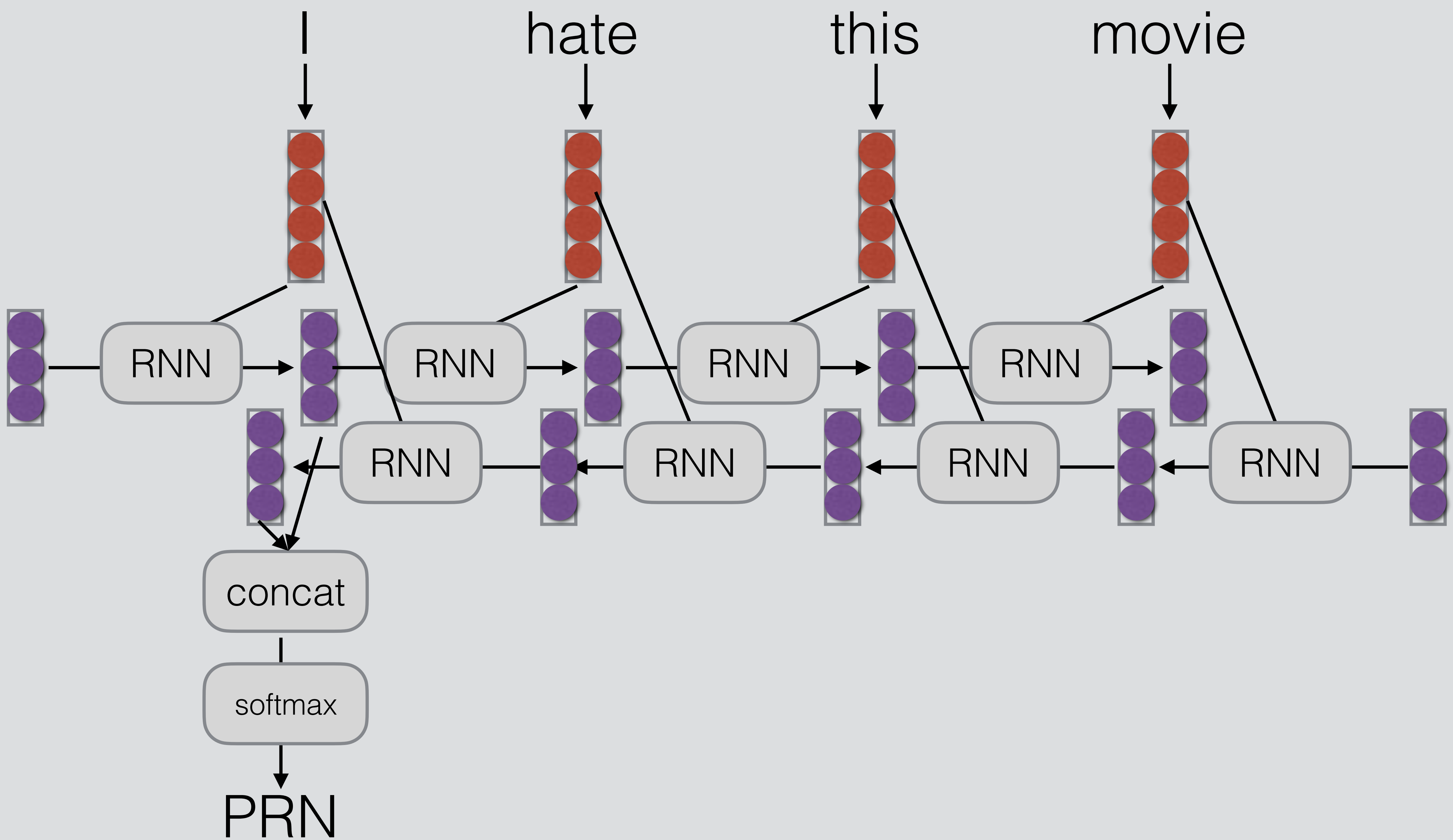
Bi-RNNs

- A simple extension, run the RNN in both directions



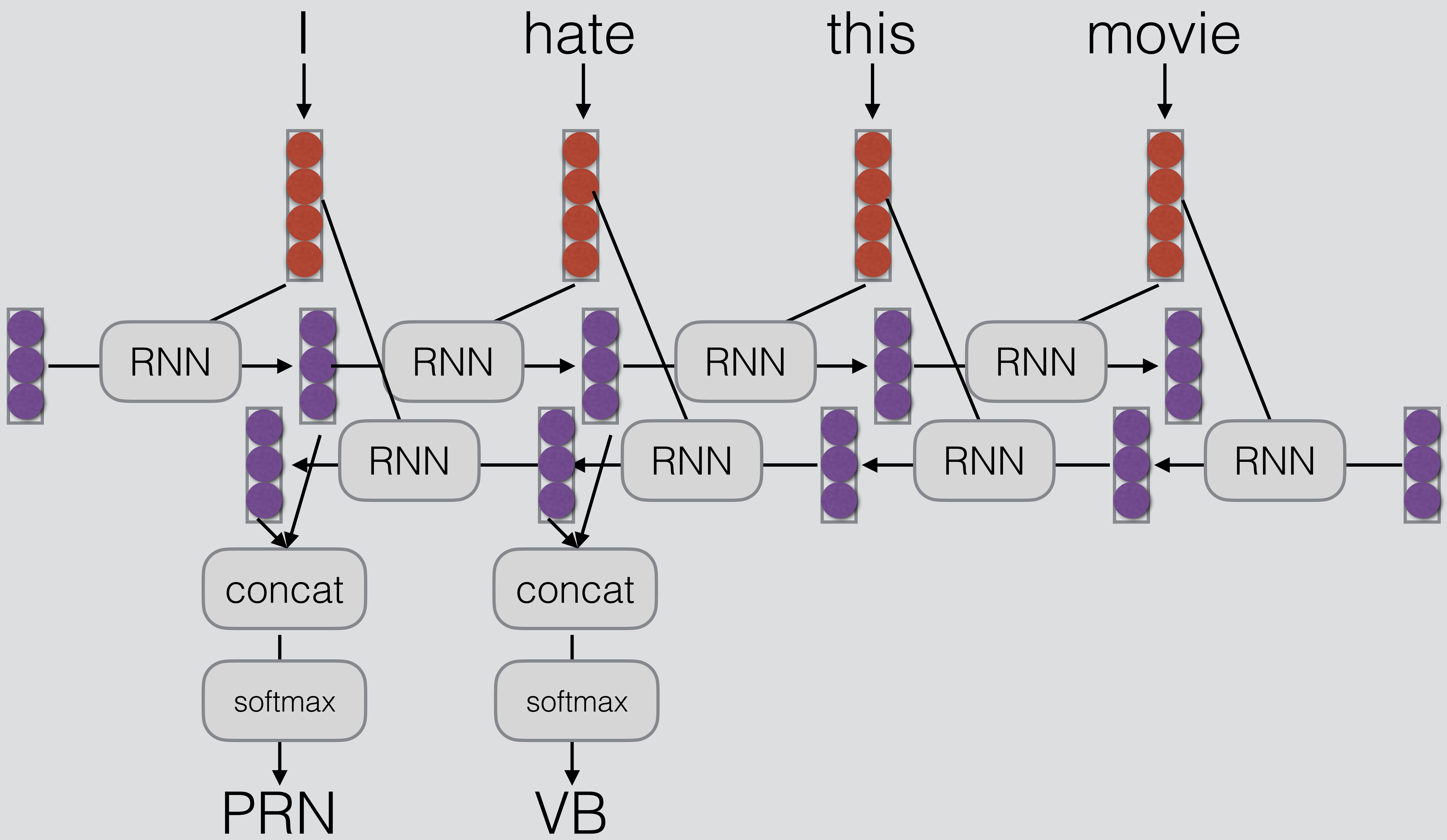
Bi-RNNs

- A simple extension, run the RNN in both directions



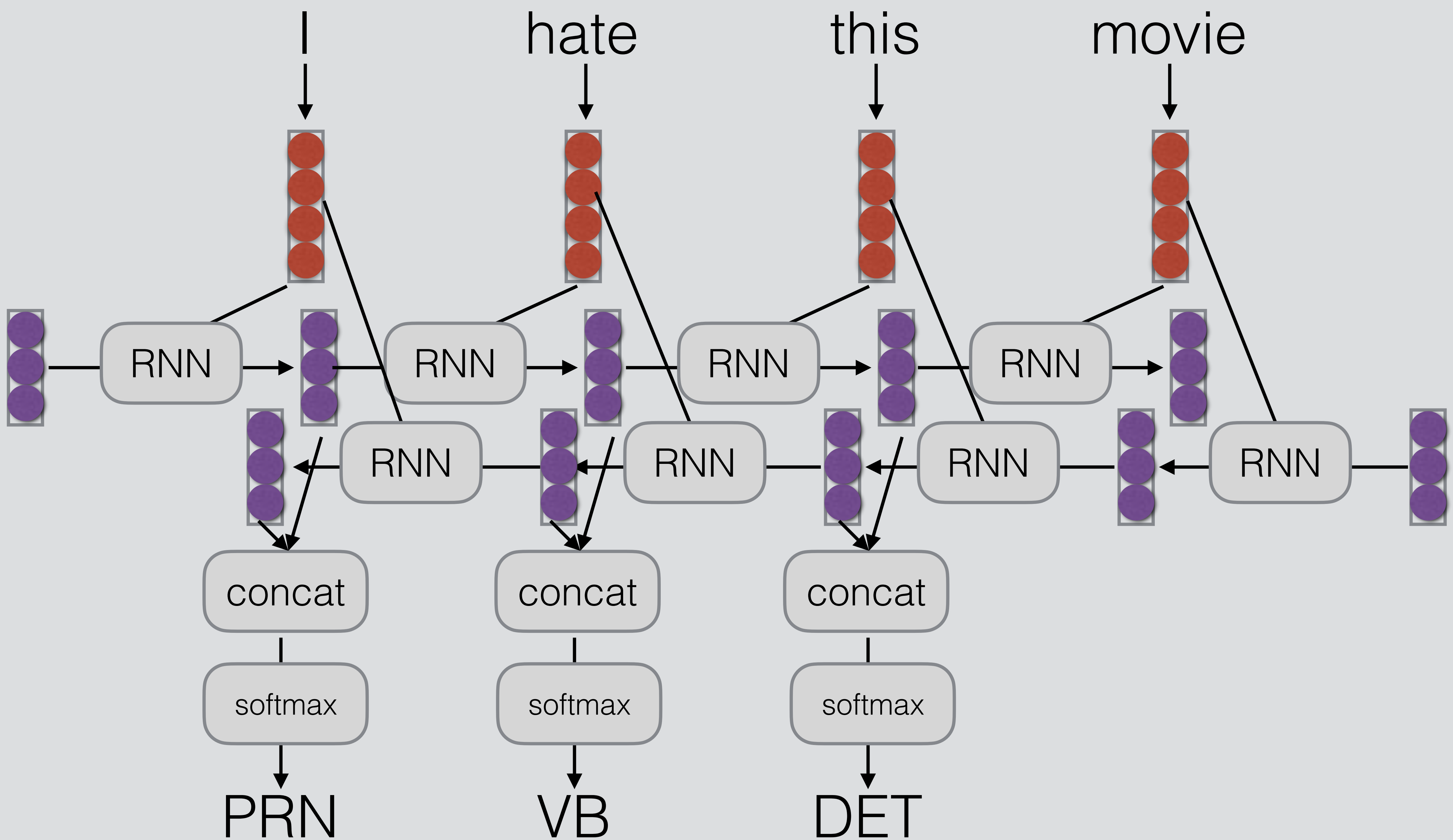
Bi-RNNs

- A simple extension, run the RNN in both directions



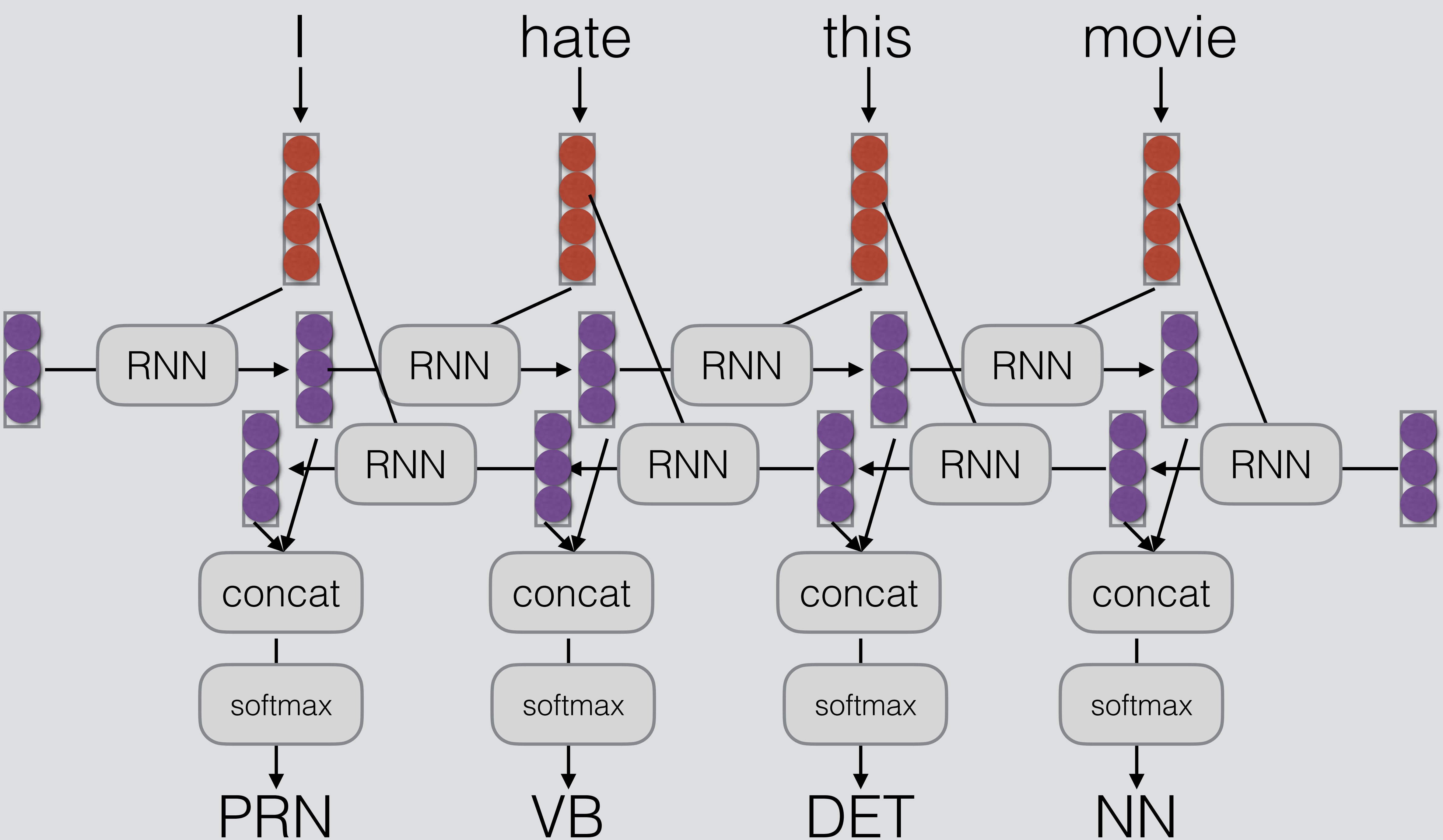
Bi-RNNs

- A simple extension, run the RNN in both directions



Bi-RNNs

- A simple extension, run the RNN in both directions

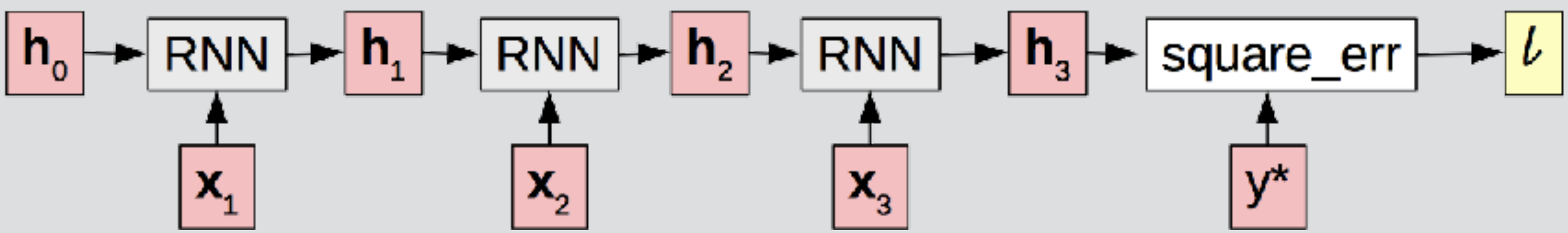


Vanishing Gradients

Vanishing Gradient

- Gradients decrease as they get pushed back

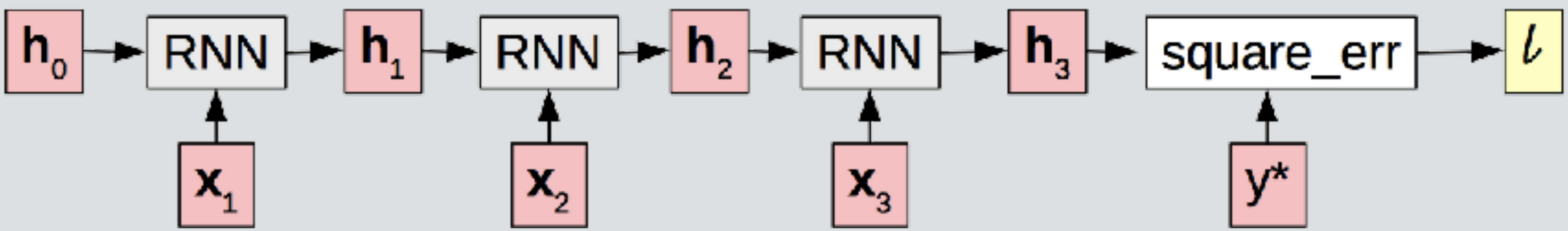
$$\frac{dl}{d_{h_0}} = \text{tiny} \quad \frac{dl}{d_{h_1}} = \text{small} \quad \frac{dl}{d_{h_2}} = \text{med.} \quad \frac{dl}{d_{h_3}} = \text{large}$$



Vanishing Gradient

- Gradients decrease as they get pushed back

$$\frac{dl}{d_{h_0}} = \text{tiny} \qquad \frac{dl}{d_{h_1}} = \text{small} \qquad \frac{dl}{d_{h_2}} = \text{med.} \qquad \frac{dl}{d_{h_3}} = \text{large}$$

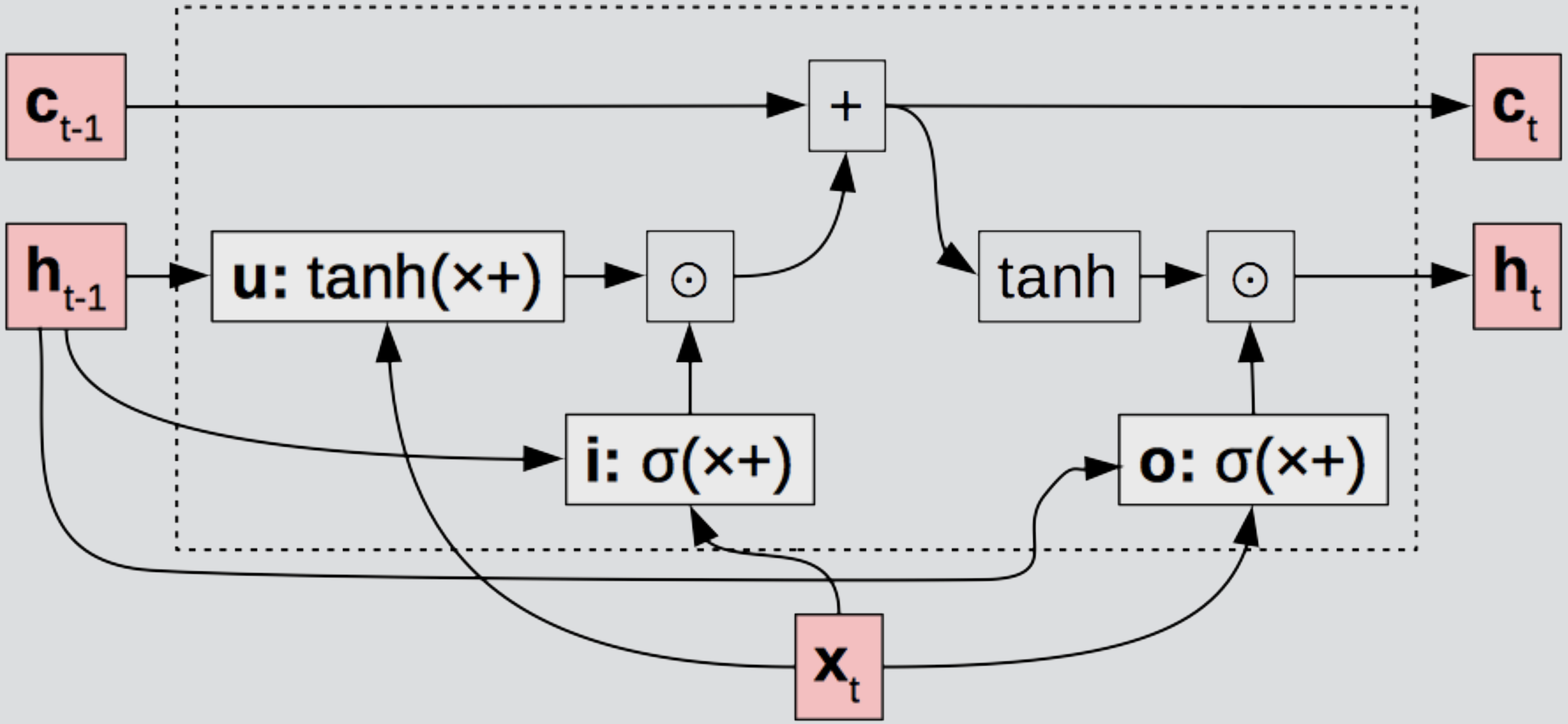


- Why? "Squashed" by non-linearities or small weights in matrices.

A Solution:
Long Short-term Memory
(Hochreiter and Schmidhuber 1997)

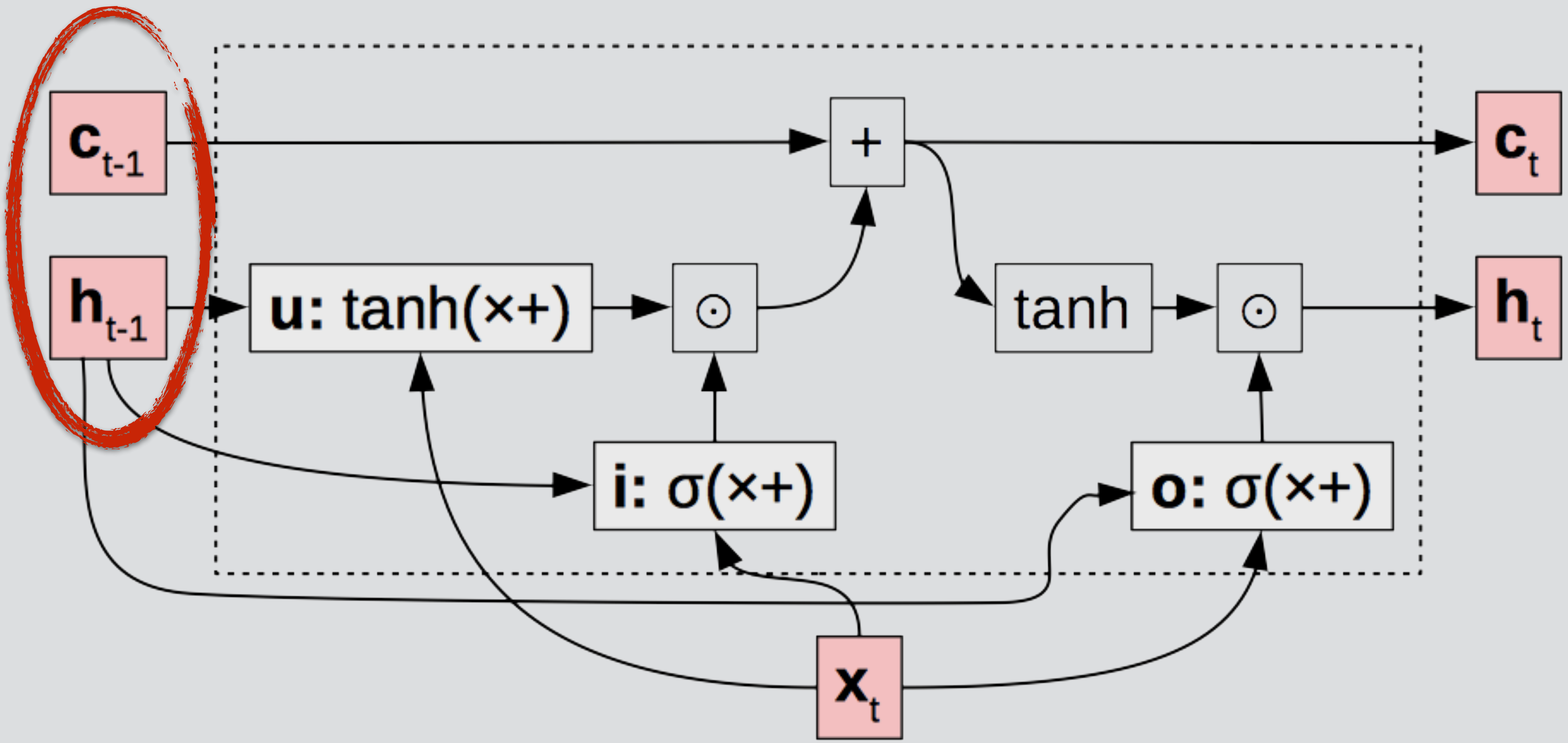
- **Basic idea:** make additive connections between time steps
- Addition does not modify the gradient, no vanishing
- Gates to control the information flow

LSTM Structure



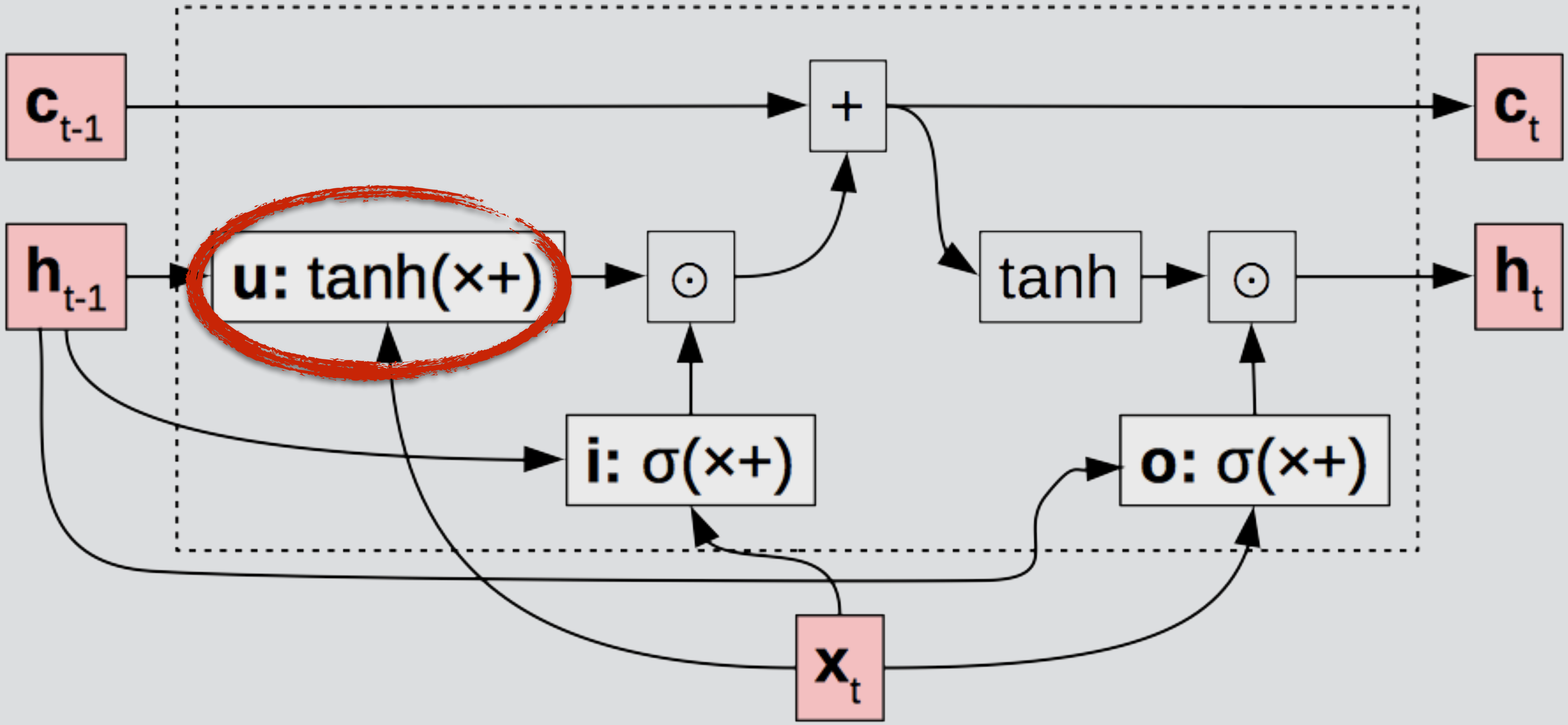
update **u**: what value do we try to add to the memory cell?
input **i**: how much of the update do we allow to go through?
output **o**: how much of the cell do we reflect in the next state?

LSTM Structure



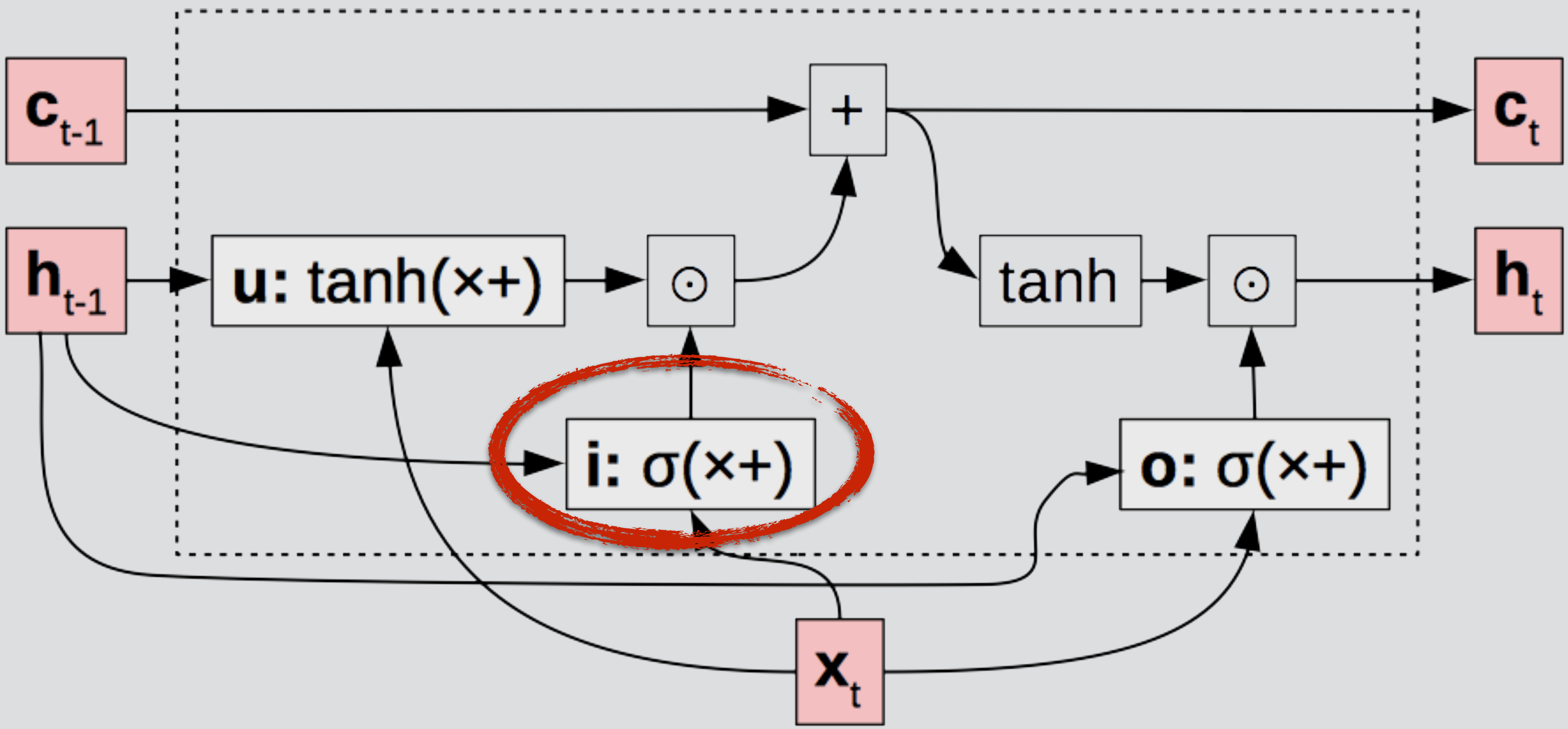
update **u**: what value do we try to add to the memory cell?
input **i**: how much of the update do we allow to go through?
output **o**: how much of the cell do we reflect in the next state?

LSTM Structure



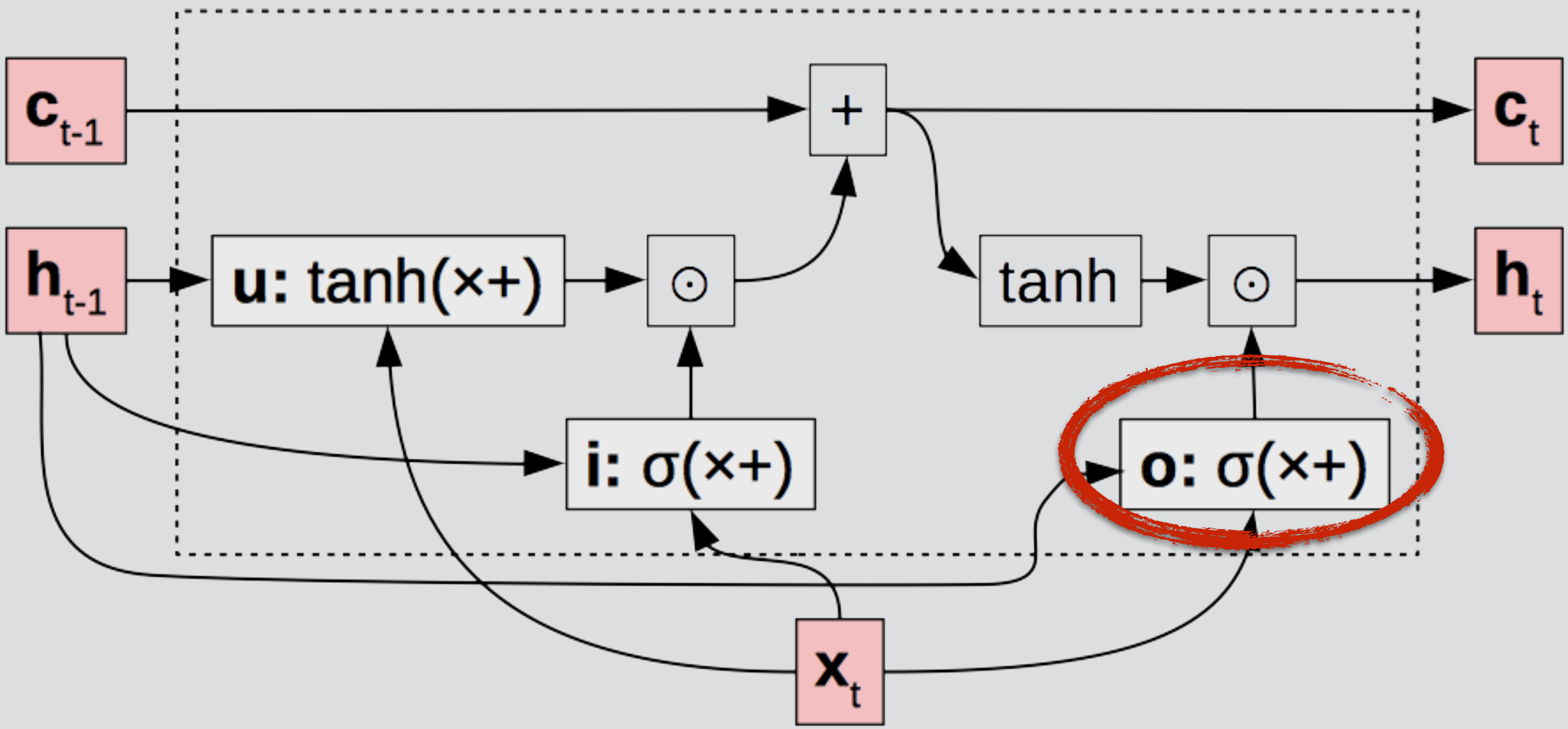
update **u**: what value do we try to add to the memory cell?
input **i**: how much of the update do we allow to go through?
output **o**: how much of the cell do we reflect in the next state?

LSTM Structure



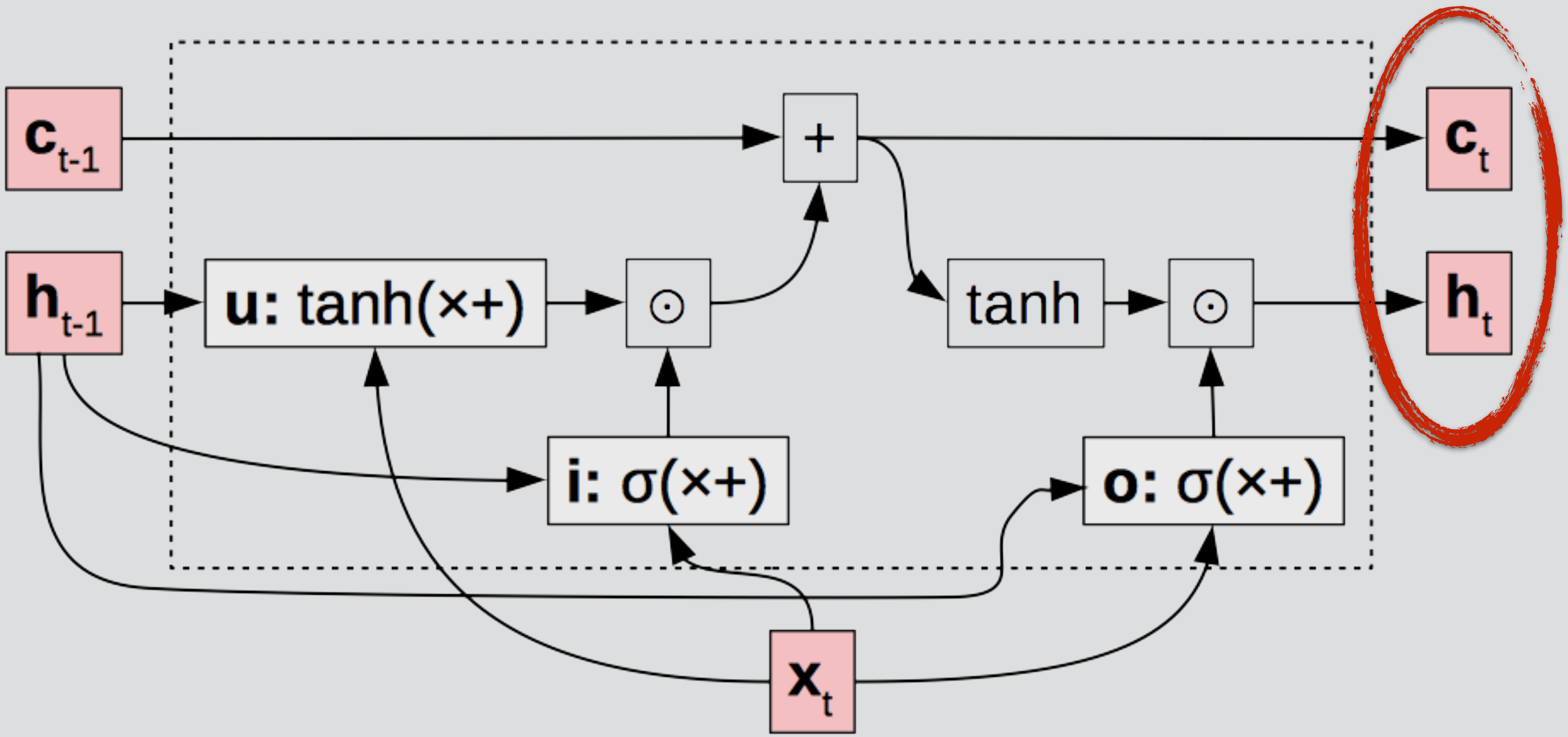
update **u**: what value do we try to add to the memory cell?
input **i**: how much of the update do we allow to go through?
output **o**: how much of the cell do we reflect in the next state?

LSTM Structure



update **u**: what value do we try to add to the memory cell?
input **i**: how much of the update do we allow to go through?
output **o**: how much of the cell do we reflect in the next state?

LSTM Structure

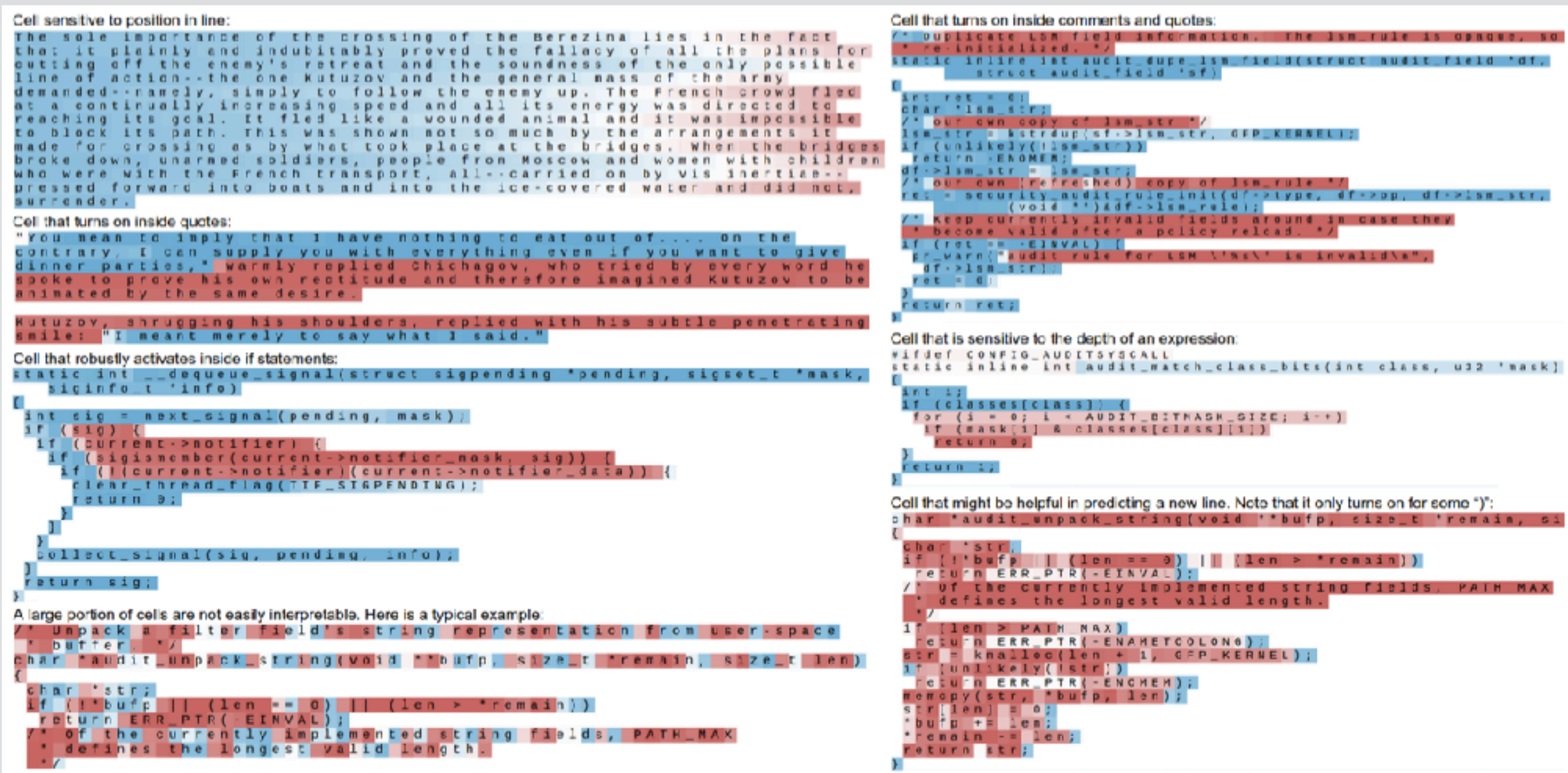


update **u**: what value do we try to add to the memory cell?
input **i**: how much of the update do we allow to go through?
output **o**: how much of the cell do we reflect in the next state?

What can LSTMs Learn? (1)

(Karpathy et al. 2015)

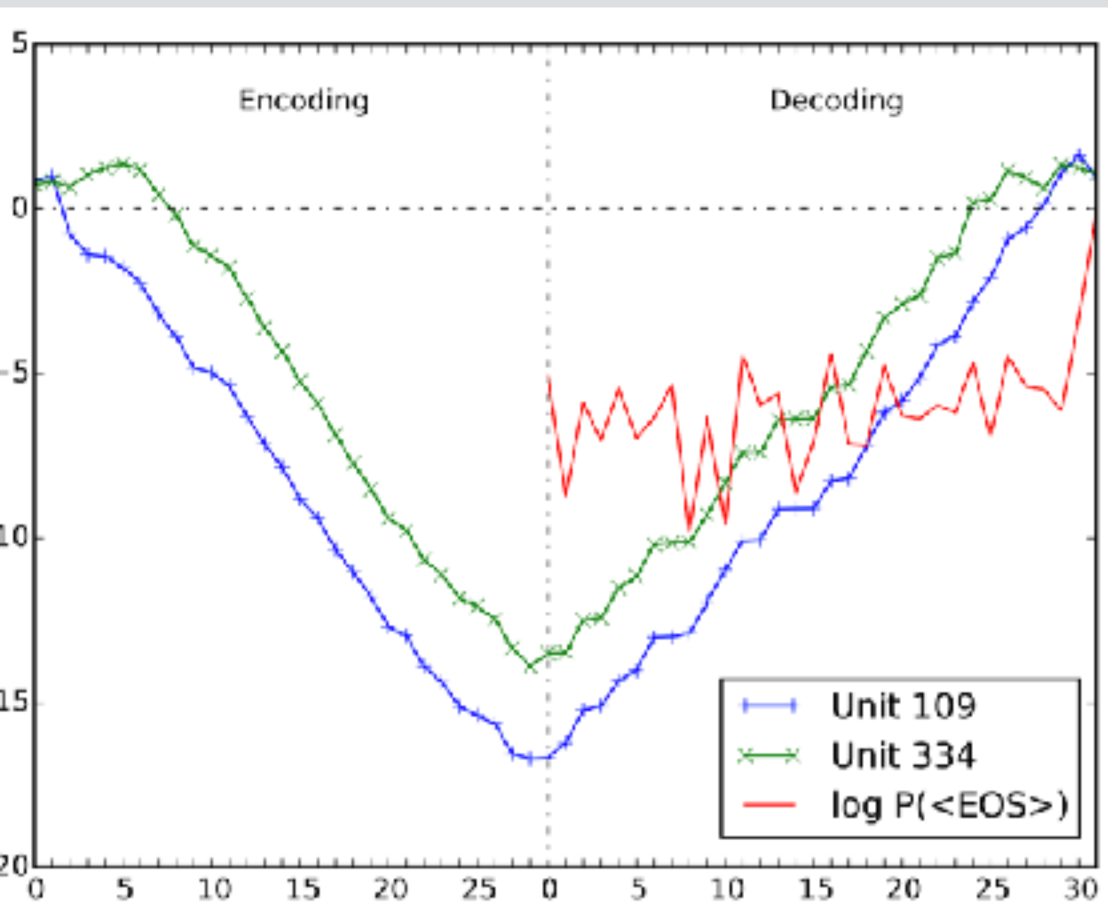
- Additive connections make single nodes surprisingly interpretable



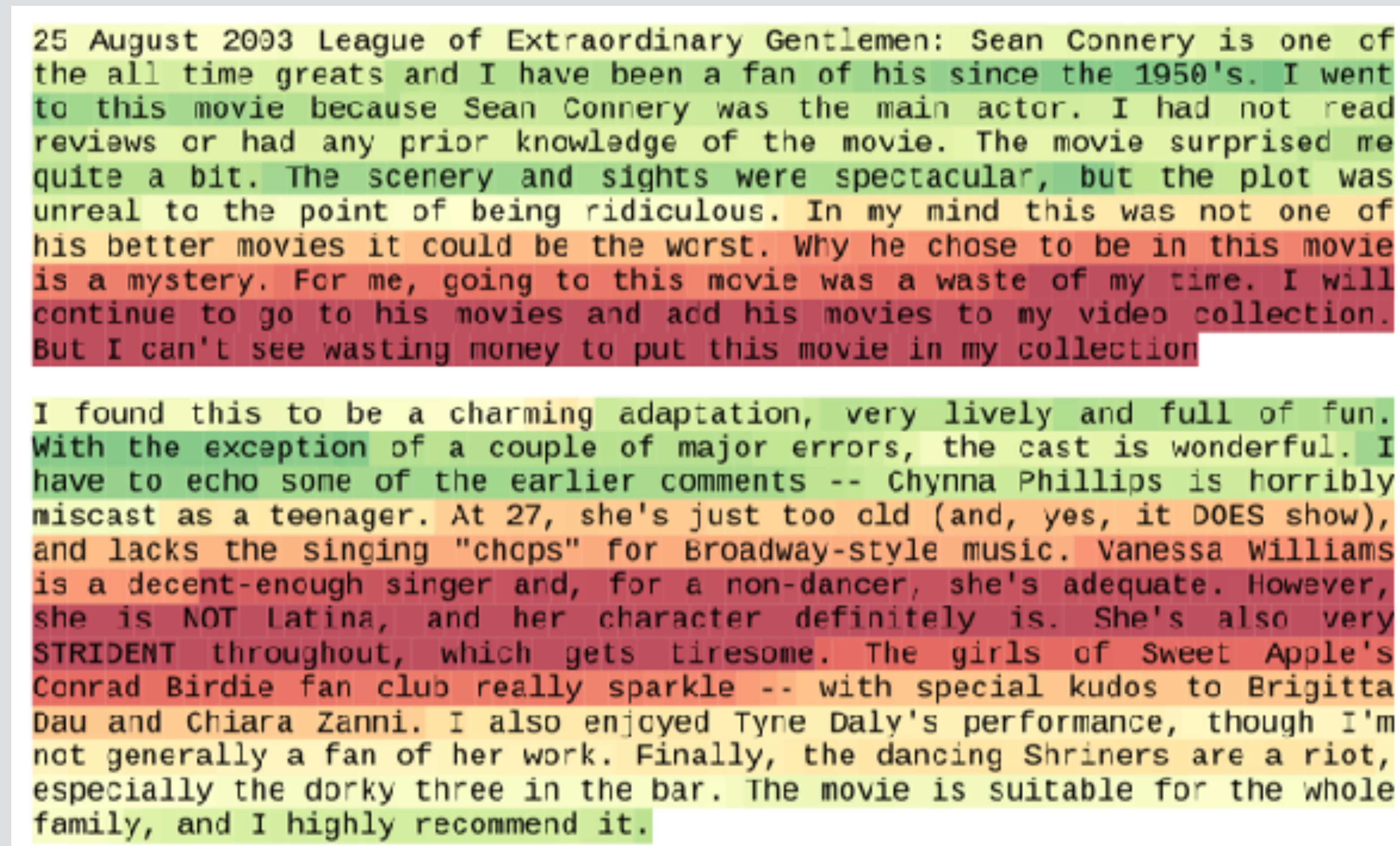
What can LSTMs Learn? (2)

(Shi et al. 2016, Radford et al. 2017)

Count length of sentence



Sentiment



Efficiency Tricks

Handling Mini-batching

- Mini-batching makes things much faster!

Handling Mini-batching

- Mini-batching makes things much faster!
- But mini-batching in RNNs is harder than in feed-forward networks

Handling Mini-batching

- Mini-batching makes things much faster!
- But mini-batching in RNNs is harder than in feed-forward networks
  - Each word depends on the previous word
  - Sequences are of various length

Mini-batching Method

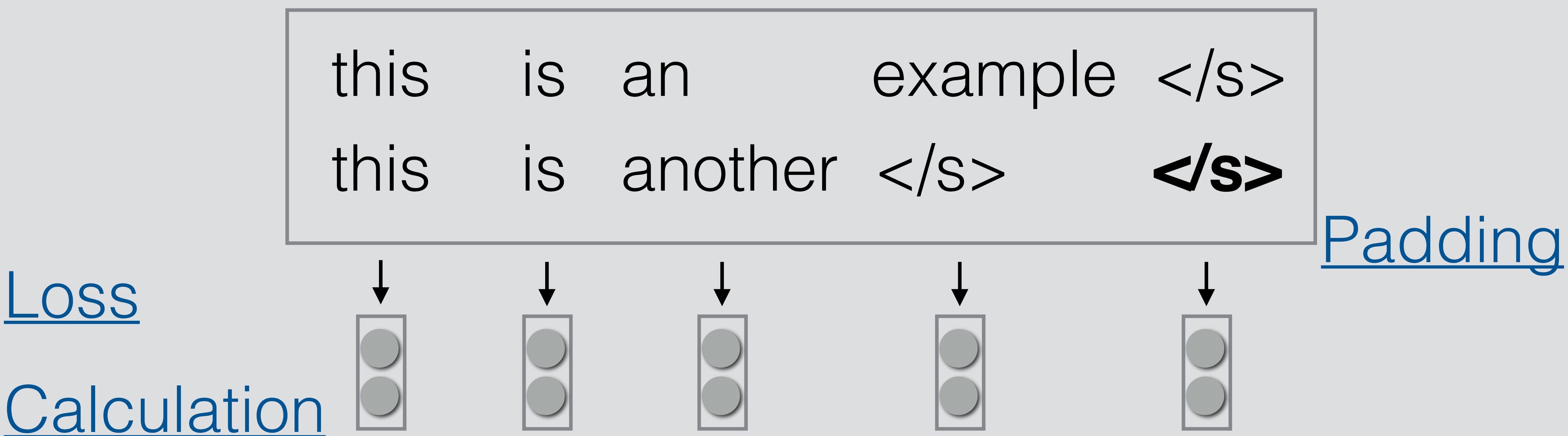
this	is	an	example	</s>
this	is	another	</s>	

Mini-batching Method

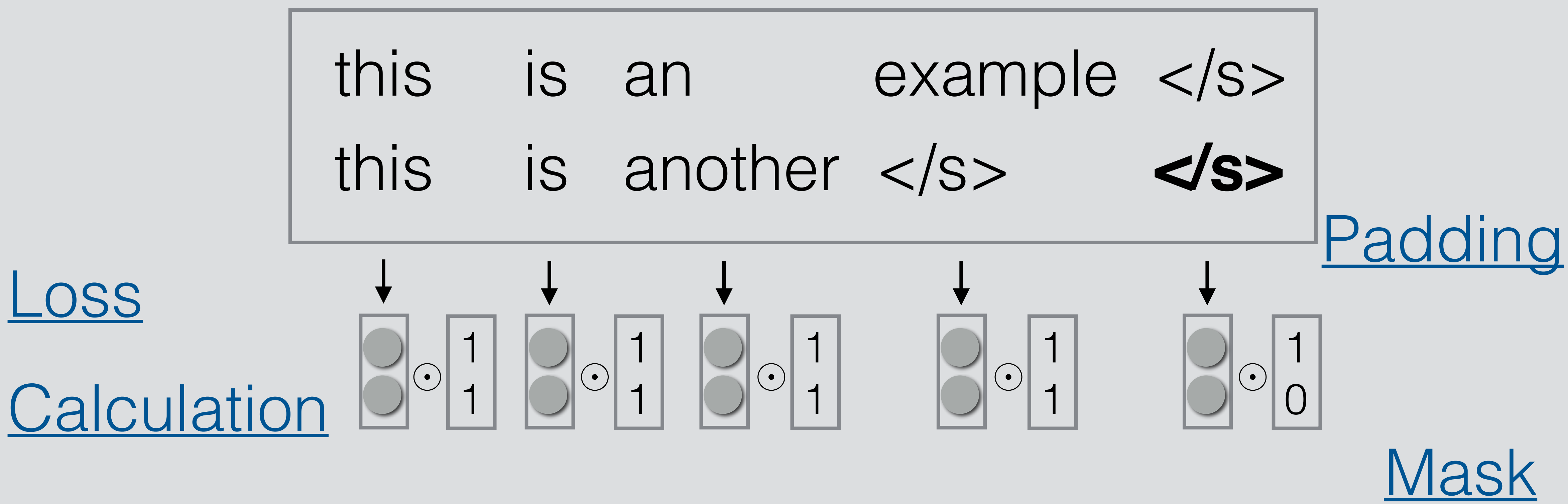
this	is	an	example	</s>
this	is	another	</s>	**</s>**

Padding

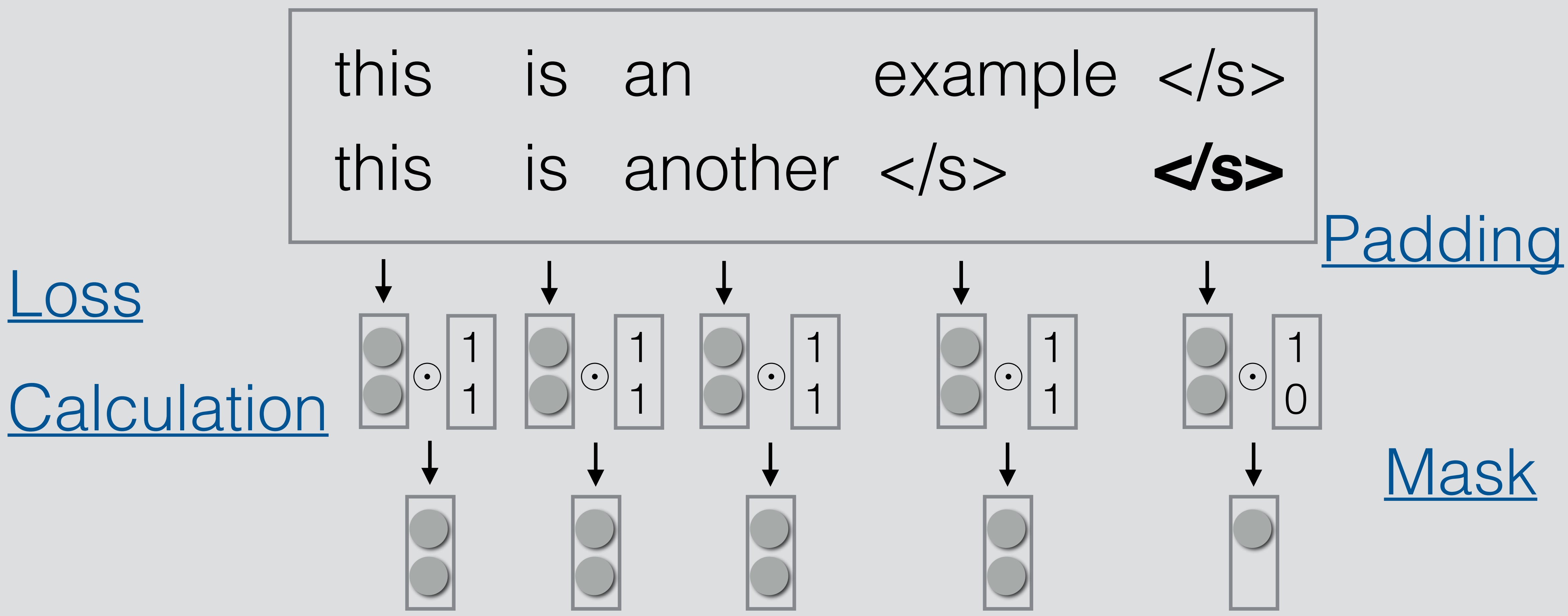
Mini-batching Method



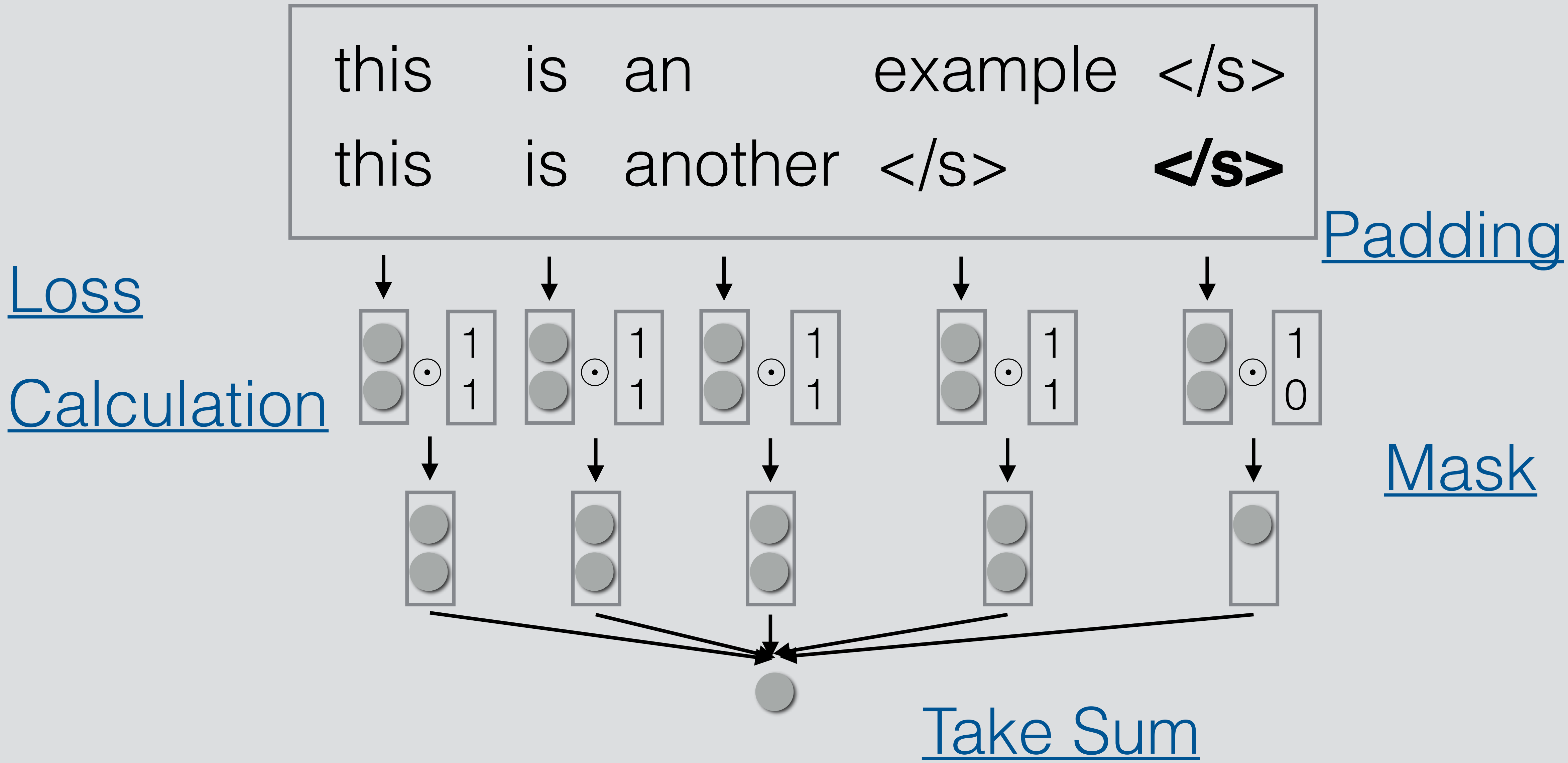
Mini-batching Method



Mini-batching Method



Mini-batching Method



Bucketing/Sorting

- If we use sentences of different lengths, too much padding and sorting can **result in decreased performance**

Bucketing/Sorting

- If we use sentences of different lengths, too much padding and sorting can **result in decreased performance**
- To remedy this: **sort sentences** so similarly-lengthed sentences are in the same batch

RNN Variants

RNN Variants

(Greffen et al. 2015)

- Gated Recurrent Units
  (GRU; Cho et al 2014)

RNN Variants
(Greffen et al. 2015)

- Gated Recurrent Units (GRU; Cho et al 2014)

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$h_t = \boxed{(1 - z_t)} \circ h_{t-1} + \boxed{z_t} \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

Additive or Non-linear

RNN Variants
(Greffen et al. 2015)

- Gated Recurrent Units
  (GRU; Cho et al 2014)

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = \boxed{(1 - z_t)} \circ h_{t-1} + \boxed{z_t} \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

Additive or Non-linear

- **Note:** GRUs cannot do things like simply count

RNN Variants
(Greffen et al. 2015)

- Gated Recurrent Units (GRU; Cho et al 2014)
- Many different types of architectures tested for LSTMs (Greffen et al. 2015)

NIG: No Input Gate: $\mathbf{i}^t = \mathbf{1}$
NFG: No Forget Gate: $\mathbf{f}^t = \mathbf{1}$
NOG: No Output Gate: $\mathbf{o}^t = \mathbf{1}$
NIAF: No Input Activation Function: $g(\mathbf{x}) = \mathbf{x}$
NOAF: No Output Activation Function: $h(\mathbf{x}) = \mathbf{x}$
CIFG: Coupled Input and Forget Gate: $\mathbf{f}^t = \mathbf{1} - \mathbf{i}^t$
NP: No Peepholes:

$$\begin{aligned}
\bar{\mathbf{i}}^t &= \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{b}_i \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{b}_f \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{b}_o
\end{aligned}$$

FGR: Full Gate Recurrence:

$$\begin{aligned}
\bar{\mathbf{i}}^t &= \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i \\
&\quad + \mathbf{R}_{ii}\mathbf{i}^{t-1} + \mathbf{R}_{fi}\mathbf{f}^{t-1} + \mathbf{R}_{oi}\mathbf{o}^{t-1} \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f \\
&\quad + \mathbf{R}_{if}\mathbf{i}^{t-1} + \mathbf{R}_{ff}\mathbf{f}^{t-1} + \mathbf{R}_{of}\mathbf{o}^{t-1} \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^{t-1} + \mathbf{b}_o \\
&\quad + \mathbf{R}_{io}\mathbf{i}^{t-1} + \mathbf{R}_{fo}\mathbf{f}^{t-1} + \mathbf{R}_{oo}\mathbf{o}^{t-1}
\end{aligned}$$

RNN Variants

(Greffen et al. 2015)

- Gated Recurrent Units (GRU; Cho et al 2014)
- Many different types of architectures tested for LSTMs (Greffen et al. 2015)
- **Conclusion:** basic LSTM quite good, other variants (e.g. coupled input/forget gates) reasonable

NIG: No Input Gate: $\mathbf{i}^t = \mathbf{1}$
NFG: No Forget Gate: $\mathbf{f}^t = \mathbf{1}$
NOG: No Output Gate: $\mathbf{o}^t = \mathbf{1}$
NIAF: No Input Activation Function: $g(\mathbf{x}) = \mathbf{x}$
NOAF: No Output Activation Function: $h(\mathbf{x}) = \mathbf{x}$
CIFG: Coupled Input and Forget Gate: $\mathbf{f}^t = \mathbf{1} - \mathbf{i}^t$
NP: No Peepholes:

$$
\begin{aligned}
\bar{\mathbf{i}}^t &= \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{b}_i \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{b}_f \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{b}_o
\end{aligned}
$$

FGR: Full Gate Recurrence:

$$
\begin{aligned}
\bar{\mathbf{i}}^t &= \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i \\
&\quad + \mathbf{R}_{ii}\mathbf{i}^{t-1} + \mathbf{R}_{fi}\mathbf{f}^{t-1} + \mathbf{R}_{oi}\mathbf{o}^{t-1} \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f \\
&\quad + \mathbf{R}_{if}\mathbf{i}^{t-1} + \mathbf{R}_{ff}\mathbf{f}^{t-1} + \mathbf{R}_{of}\mathbf{o}^{t-1} \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^{t-1} + \mathbf{b}_o \\
&\quad + \mathbf{R}_{io}\mathbf{i}^{t-1} + \mathbf{R}_{fo}\mathbf{f}^{t-1} + \mathbf{R}_{oo}\mathbf{o}^{t-1}
\end{aligned}
$$

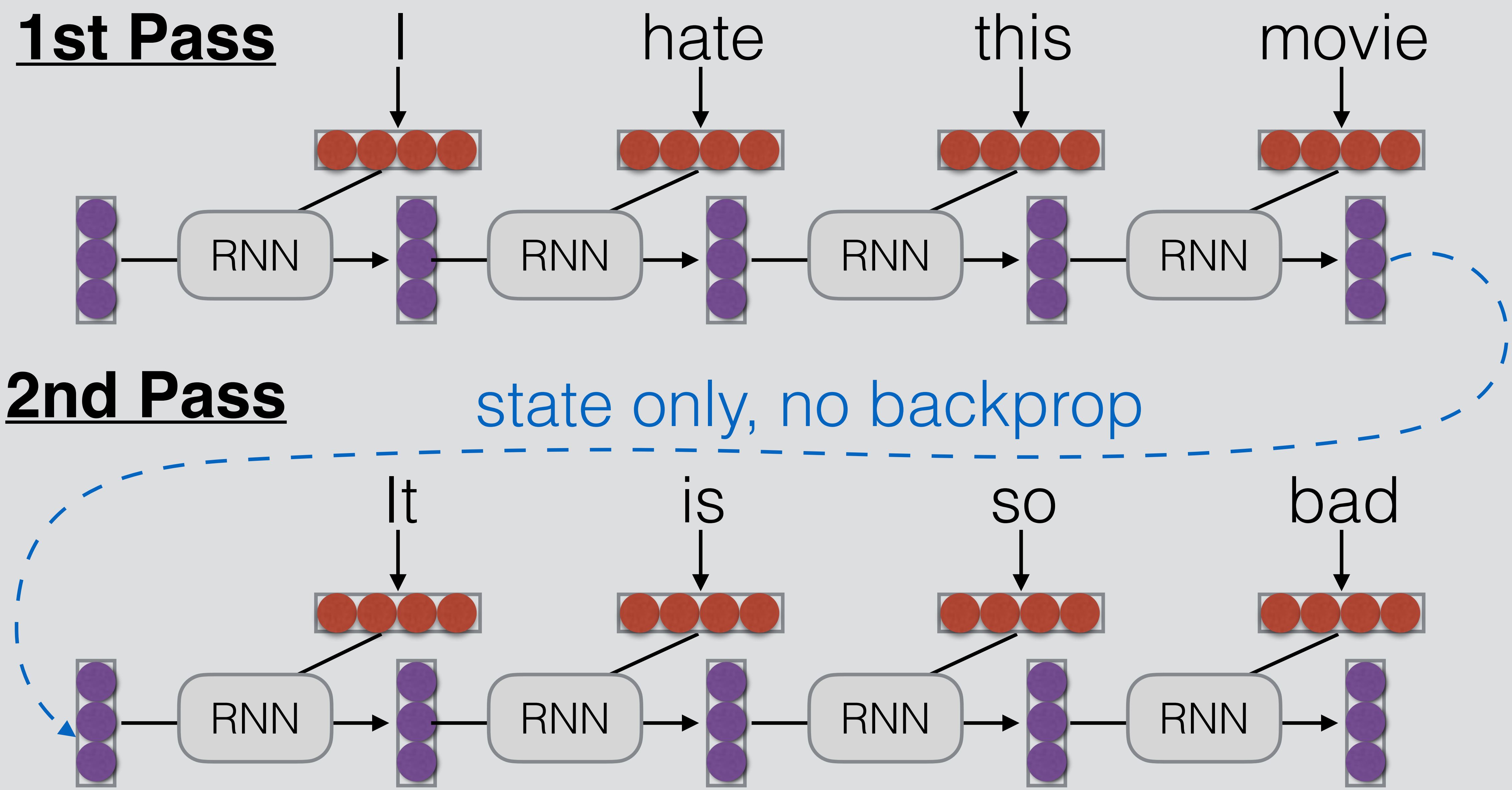
Handling Long Sequences

Handling Long Sequences

- Sometimes we would like to capture long-term dependencies over long sequences
- e.g. words in full documents
- However, this may not fit on (GPU) memory

Truncated BPTT

- Backprop over shorter segments, initialize w/ the state from the previous segment



Questions?
(see extra slides)

Simple Implementation of RNNs (in DyNet)

- Based on “*Builder” class (*=SimpleRNN/LSTM)
- Add parameters to model (once):

```
# LSTM (layers=1, input=64, hidden=128, model)
RNN = dy.SimpleRNNBuilder(1, 64, 128, model)
```

- Add parameters to CG and get initial state (per sentence):

```
s = RNN.initial_state()
```

- Update state and access (per input word/character):

```
s = s.add_input(x_t)
h_t = s.output()
```

RNNLM Example: Parameter Initialization

```
# Lookup parameters for word embeddings
WORDS_LOOKUP = model.add_lookup_parameters((nwords, 64))

# Word-level RNN (layers=1, input=64, hidden=128, model)
RNN = dy.SimpleRNNBuilder(1, 64, 128, model)

# Softmax weights/biases on top of RNN outputs
W_sm = model.add_parameters((nwords, 128))
b_sm = model.add_parameters(nwords)
```

RNNLM Example: Sentence Initialization

```
# Build the language model graph
def calc_lm_loss(wids):
    dy.renew_cg()

    # parameters -> expressions
    W_exp = dy.parameter(W_sm)
    b_exp = dy.parameter(b_sm)

    # add parameters to CG and get state
    f_init = RNN.initial_state()

    # get the word vectors for each word ID
    wembs = [WORDS_LOOKUP[wid] for wid in wids]

    # Start the rnn by inputting "<s>"
    s = f_init.add_input(wembs[-1])

                    ...
```

RNNLM Example:
Loss Calculation and State Update

…

```
# process each word ID and embedding
losses = []
for wid, we in zip(wids, wembs):

    # calculate and save the softmax loss
    score = W_exp * s.output() + b_exp
    loss = dy.pickneglogsoftmax(score, wid)
    losses.append(loss)

    # update the RNN state with the input
    s = s.add_input(we)

# return the sum of all losses
return dy.esum(losses)
```

Code Examples

`sentiment-rnn.py`