

Discrete and Continuous Language Models

Ashish Vaswani

University of Southern California
Information Sciences Institute



Statistical Machine Translation (SMT)

source

target



Statistical Machine Translation (SMT)

source

target



la kato sidis sur mato

the cat sat on a mat

Statistical Machine Translation (SMT)

source

वह बिल्ली चटाई पर बैठी

la kato sidis sur mato

target



the cat sat on a mat

the cat sat on a mat

Statistical Machine Translation (SMT)

source

set a as the sigmoid of y

वह बिल्ली चटाई पर बैठी

la kato sidis sur mato



target

$$a = \frac{1}{1 + e^{-y}}$$

the cat sat on a mat

the cat sat on a mat

Statistical Machine Translation (SMT)

source

$$a = \frac{1}{1 + e^{-y}}$$

वह बिल्ली चटाई पर बैठी

la kato sidis sur mato



target

set a as the sigmoid of y

the cat sat on a mat

the cat sat on a mat

Statistical Machine Translation (SMT)

source

loop over all people

$$a = \frac{1}{1 + e^{-y}}$$

वह बिल्ली चटाई पर बैठी

la kato sidis sur mato

target

for (int i=0; i>personlist.size(); i++)

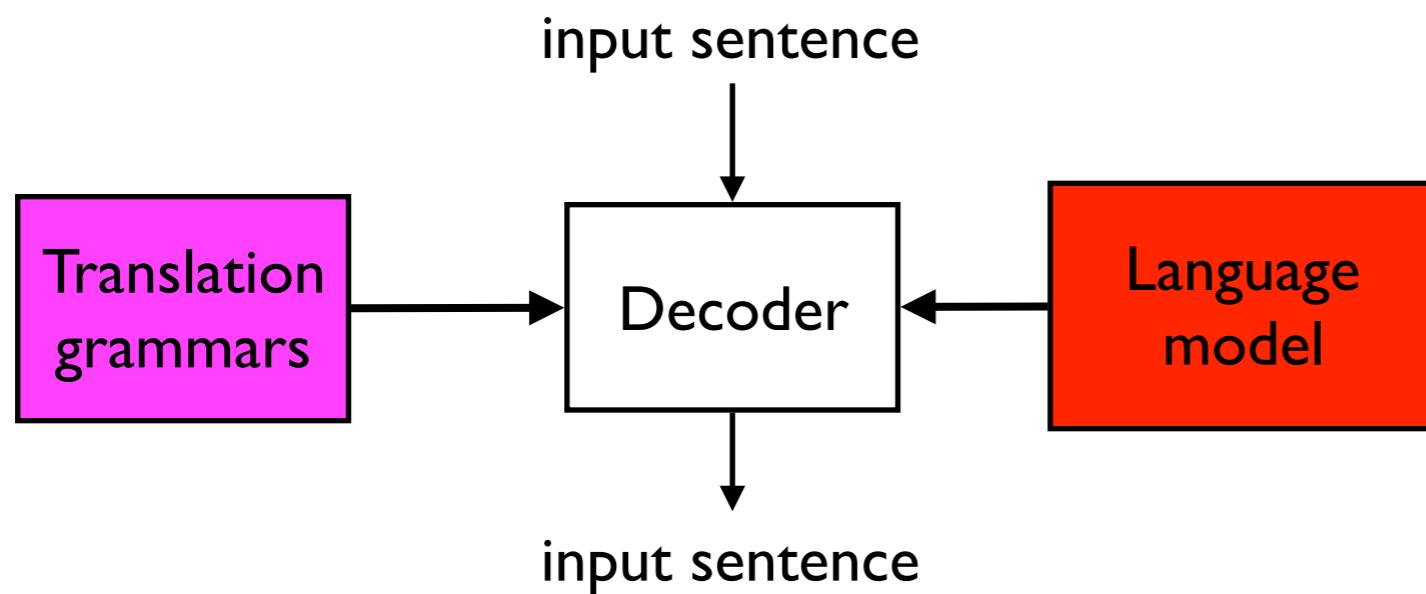
set a as the sigmoid of y

the cat sat on a mat

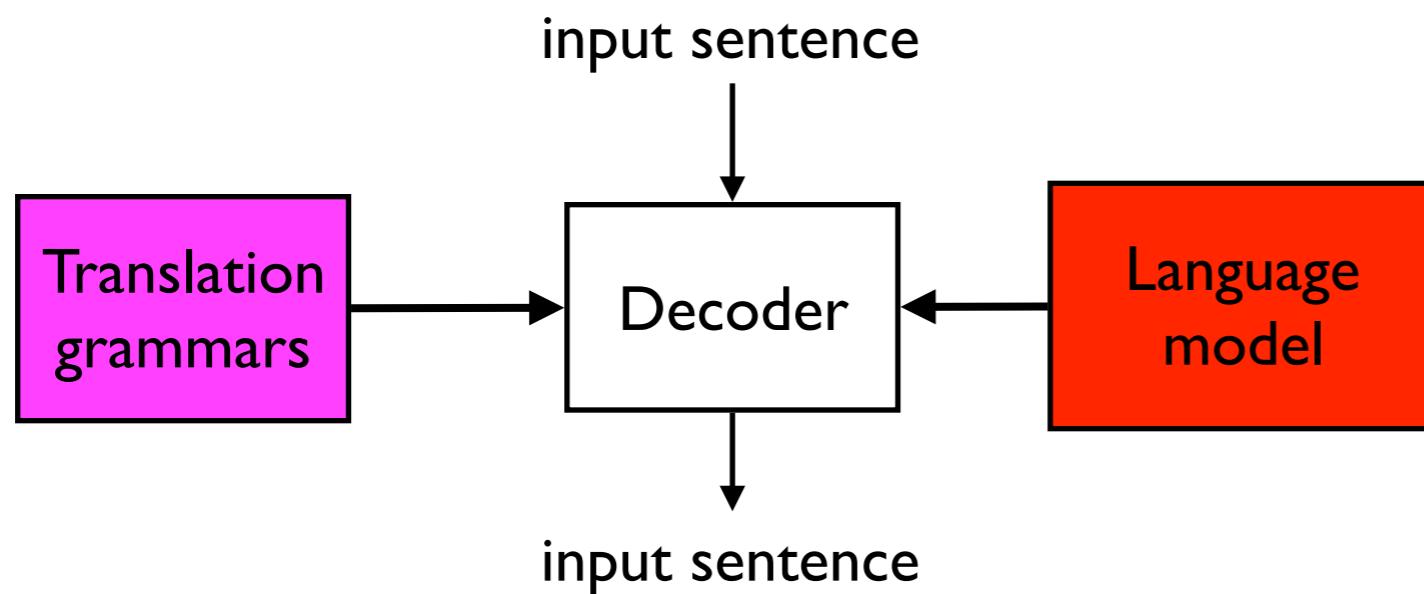
the cat sat on a mat



Language models in Machine Translation



Language models in Machine Translation

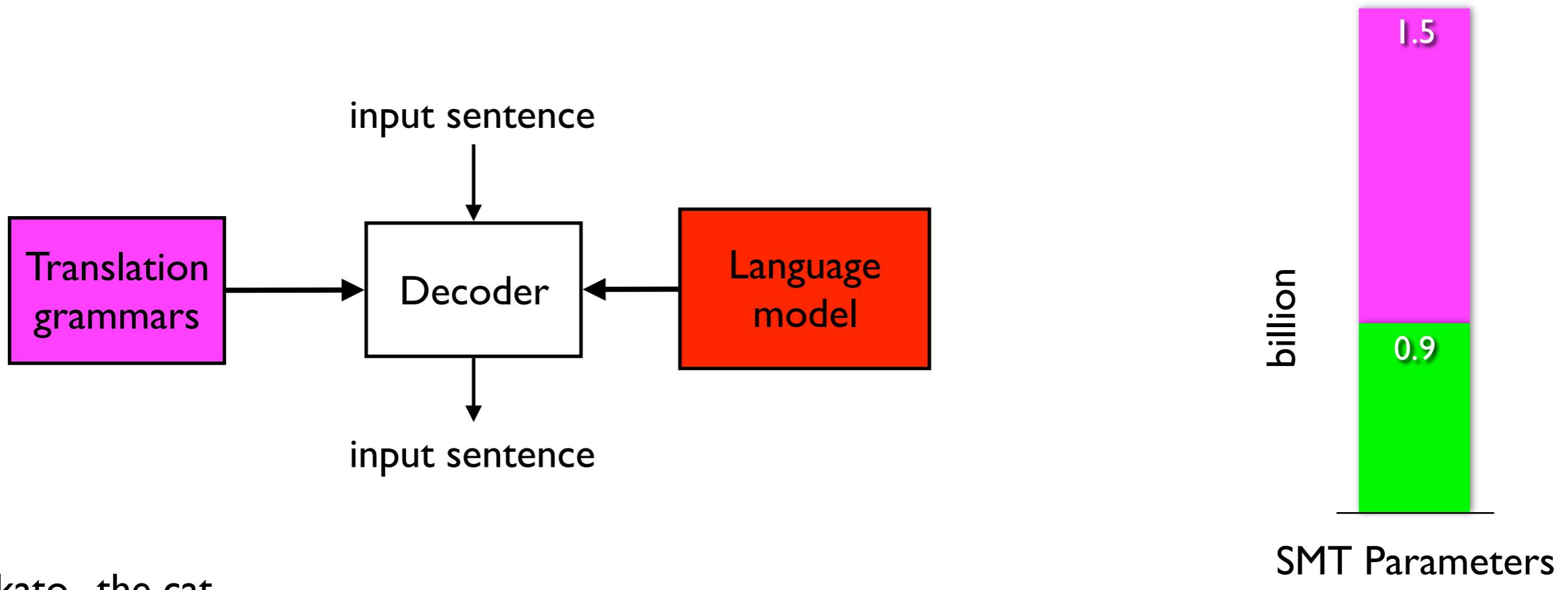


la kato , the cat

चटाई पर बैठी , sat on a mat

sigmoid of y , $\frac{1}{1 + e^{-y}}$

Language models in Machine Translation



la kato , the cat

चटाई पर बैठी , sat on a mat

$$\text{sigmoid of } y, \frac{1}{1 + e^{-y}}$$

Applications of Language Models

Natural Language Applications

- Machine Translation
- Speech Recognition
- Spelling Correction
- Document Summarization
- Sentence Completion

Programming Languages

- Code Completion
- Retrieval of code from natural language
- Retrieval of natural language from code snippets
- Identifying buggy code

Outline

What are Language Models ?

- Definition
- Markovization

Outline

What are Language Models ?

- Definition
- Markovization

Discrete Language Models

- Estimating Probabilities
- Maximum Likelihood Estimation
- Overfitting
- Smoothing

Outline

What are Language Models ?

- Definition
- Markovization

Discrete Language Models

- Estimating Probabilities
- Maximum Likelihood Estimation
- Overfitting
- Smoothing

Continuous Language Models

- Feed Forward Neural Network LM
- Recurrent Neural Network LM
 - A. Vanilla RNN
 - B. Long Short Term Memory RNN
- Visualizing LSTMs

What are Language
Models ?

Language Models

the cat sat on the mat

Language Models

$P(\text{the cat sat on the mat})$

Chain Rule

$P(\text{the cat sat on the mat})$

$$= P(\text{the}) P(\text{cat} \mid \text{the}) P(\text{sat} \mid \text{the cat}) P(\text{on} \mid \text{the cat sat}) \\ P(\text{the} \mid \text{the cat sat on}) P(\text{mat} \mid \text{the cat sat on the})$$

Chain Rule

$P(\text{the cat sat on the mat})$

$$= P(\text{the}) P(\text{cat} \mid \text{the}) P(\text{sat} \mid \text{the cat}) P(\text{on} \mid \text{the cat sat}) \\ P(\text{the} \mid \text{the cat sat on}) P(\text{mat} \mid \text{the cat sat on the})$$

$$P(w_1, w_2, \dots, w_n) = \prod_i P(w_i \mid w_1, w_2, \dots, w_{i-1})$$

n-gram Language Model



Andrei Markov

$$\begin{aligned} & P(\text{the cat sat on the mat}) \\ = & P(\text{the}) P(\text{cat} \mid \text{the}) P(\text{sat} \mid \text{cat}) P(\text{on} \mid \text{sat}) \\ & P(\text{the} \mid \text{on}) P(\text{mat} \mid \text{the}) \end{aligned}$$

Bigram Language Model



Andrei Markov

$$\begin{aligned} & P(\text{the cat sat on the mat}) \\ = & P(\text{the}) P(\text{cat} \mid \text{the}) P(\text{sat} \mid \text{cat}) P(\text{on} \mid \text{sat}) \\ & P(\text{the} \mid \text{on}) P(\text{mat} \mid \text{the}) \end{aligned}$$

Bigram Language Model



Andrei Markov

$$\begin{aligned} & P(\text{the cat sat on the mat}) \\ = & P(\text{the}) P(\text{cat} \mid \text{the}) P(\text{sat} \mid \text{cat}) P(\text{on} \mid \text{sat}) \\ & P(\text{the} \mid \text{on}) P(\text{mat} \mid \text{the}) \end{aligned}$$

Bigram Language Model



Andrei Markov

$P(\text{the cat sat on the mat})$

$$= P(\text{the}) P(\text{cat} | \text{the}) P(\text{sat} | \text{cat}) P(\text{on} | \text{sat}) \\ P(\text{the} | \text{on}) P(\text{mat} | \text{the})$$

Trigram Language Model



$$\begin{aligned} & P(\text{the cat sat on the mat}) \\ = & P(\text{the}) P(\text{cat} \mid \text{the}) P(\text{sat} \mid \text{the cat}) P(\text{on} \mid \text{cat sat}) \\ & P(\text{the} \mid \text{sat on}) P(\text{mat} \mid \text{on the}) \end{aligned}$$

Andrei Markov

Trigram Language Model



$$\begin{aligned} & P(\text{the cat sat on the mat}) \\ = & P(\text{the}) P(\text{cat} \mid \text{the}) P(\text{sat} \mid \text{the cat}) P(\text{on} \mid \text{cat sat}) \\ & P(\text{the} \mid \text{sat on}) P(\text{mat} \mid \text{on the}) \end{aligned}$$

Andrei Markov

$$P(w_1, w_2, \dots, w_n) \approx \prod_i P(w_i \mid w_{i-k+1}, w_2, \dots, w_{i-1})$$

Discrete Space Language Models

Estimating Probabilities

$P(\text{the cat sat on the mat})$

$$= P(\text{the}) P(\text{cat} | \text{the}) P(\text{sat} | \text{cat}) P(\text{on} | \text{sat}) \\ P(\text{the} | \text{on}) P(\text{mat} | \text{the})$$

$$P(\text{on} | \text{sat}) = \frac{\text{count}(\text{sat on})}{\text{count}(\text{sat})}$$

Estimating Probabilities

$P(\text{the cat sat on the mat})$

$$= P(\text{the}) P(\text{cat} | \text{the}) P(\text{sat} | \text{cat}) P(\text{on} | \text{sat}) \\ P(\text{the} | \text{on}) P(\text{mat} | \text{the})$$

$$P(\text{on} | \text{sat}) = \frac{c(\text{sat on})}{c(\text{sat})}$$

Estimating Probabilities

$$P(on \mid sat) = \frac{c(sat \text{ on})}{c(sat)}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \mid w_i)}{c(w_i)}$$

Estimating Probabilities

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \mid w_i)}{c(w_i)}$$

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

Estimating Probabilities

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \mid w_i)}{c(w_i)}$$

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(cat \mid the) = \frac{3}{6} = 0.5$$

Estimating Probabilities

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \mid w_i)}{c(w_i)}$$

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(cat \mid the) = \frac{3}{6} = 0.5$$

$$P(dog \mid the) = \frac{1}{6} = 0.167$$

Estimating Probabilities

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \mid w_i)}{c(w_i)}$$

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(cat \mid the) = \frac{3}{6} = 0.5$$

$$P(dog \mid the) = \frac{1}{6} = 0.167$$

$$P(mat \mid the) = \frac{1}{6} = 0.167$$

Estimating Probabilities

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \mid w_i)}{c(w_i)}$$

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(cat \mid the) = \frac{3}{6} = 0.5$$

$$P(dog \mid the) = \frac{1}{6} = 0.167$$

$$P(mat \mid the) = \frac{1}{6} = 0.167$$

$$P(the \mid on) = \frac{2}{2} = 1.0$$

Maximum Likelihood Estimation

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \mid w_i)}{c(w_i)}$$

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(cat \mid the) = \frac{3}{6} = 0.5$$

$$P(dog \mid the) = \frac{1}{6} = 0.167$$

$$P(mat \mid the) = \frac{1}{6} = 0.167$$

$$P(the \mid on) = \frac{2}{2} = 1.0$$

Maximum Likelihood Estimation (MLE)

$P(\langle s \rangle \text{ the cat sat on the mat } \langle /s \rangle) =$

Maximum Likelihood Estimation (MLE)

$P(< s > \text{ the cat sat on the mat } </s>) = P(\text{the} | < s >) \times$

Maximum Likelihood Estimation (MLE)

$$P(\text{<s> the cat sat on the mat </s>}) = P(\text{the} | \text{<s>}) \times \\ P(\text{cat} | \text{the}) \times$$

Maximum Likelihood Estimation (MLE)

$$P(\langle s \rangle \text{ the } \text{cat sat} \text{ on the mat } \langle /s \rangle) = P(\text{the} | \langle s \rangle) \times \\ P(\text{cat} | \text{the}) \times \\ P(\text{sat} | \text{cat}) \times$$

Maximum Likelihood Estimation (MLE)

$$P(\langle s \rangle \text{ the cat sat on the mat } \langle /s \rangle) = P(\text{the} | \langle s \rangle) \times \\ P(\text{cat} | \text{the}) \times \\ P(\text{sat} | \text{cat}) \times \\ P(\text{on} | \text{sat}) \times$$

Maximum Likelihood Estimation (MLE)

$$P(\langle s \rangle \text{ the cat sat on the mat } \langle /s \rangle) = P(\text{the} | \langle s \rangle) \times \\ P(\text{cat} | \text{the}) \times \\ P(\text{sat} | \text{cat}) \times \\ P(\text{on} | \text{sat}) \times \\ P(\text{the} | \text{on})$$

Maximum Likelihood Estimation (MLE)

$$P(\langle s \rangle \text{ the cat sat on the mat } \langle /s \rangle) = P(\text{the} | \langle s \rangle) \times \\ P(\text{cat} | \text{the}) \times \\ P(\text{sat} | \text{cat}) \times \\ P(\text{on} | \text{sat}) \times \\ P(\text{the} | \text{on}) \times \\ P(\text{mat} | \text{the}) \times$$

Maximum Likelihood Estimation (MLE)

$$P(\langle s \rangle \text{ the cat sat on the mat } \langle /s \rangle) = P(\text{the} | \langle s \rangle) \times \\ P(\text{cat} | \text{the}) \times \\ P(\text{sat} | \text{cat}) \times \\ P(\text{on} | \text{sat}) \times \\ P(\text{the} | \text{on}) \times \\ P(\text{mat} | \text{the}) \times \\ P(\langle /s \rangle | \text{mat})$$

Maximum Likelihood Estimation (MLE)

$$\begin{aligned} P(< s > \text{ the cat sat on the mat } </s>) &= P(\text{the} | < s >) \times \\ &\quad P(\text{cat} | \text{the}) \times \\ &\quad P(\text{sat} | \text{cat}) \times \\ &\quad P(\text{on} | \text{sat}) \times \\ &\quad P(\text{the} | \text{on}) \times \\ &\quad P(\text{mat} | \text{the}) \times \\ &\quad P(< /s > | \text{mat}) \\ &= 0.084 \end{aligned}$$

MLE with higher order n-grams ?

$$\begin{aligned} P(<\text{s}> <\text{s}> \text{ the cat sat on the mat } </\text{s}>) &= P(\text{the} | <\text{s}><\text{s}>) \times \\ &\quad P(\text{cat} | <\text{s}> \text{the}) \times \\ &\quad P(\text{sat} | \text{the cat}) \times \\ &\quad \ddots \\ &\quad P(</\text{s}> | \text{the mat}) \\ &= 0.33 \end{aligned}$$

MLE with higher order n-grams ?

If you increase the n-gram order, you will improve your training data probability with MLE

MLE with higher order n-grams ?

If you increase the n-gram order, you will improve your training data probability with MLE

$$P(\text{DATA})_{n\text{-gram}} \geq P(\text{DATA})_{(n-1)\text{-gram}}$$

MLE with higher order n-grams ?

If you increase the n-gram order, you will improve your training data probability with MLE

$$P(\text{DATA})_{n\text{-gram}} \geq P(\text{DATA})_{(n-1)\text{-gram}}$$

Log Sum Inequality

$$a_i \log \frac{a_i}{b_i} \geq \left(\sum_{i=1}^n a_i \right) \log \frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i}$$

Comparing Language Models

Data Splits

- Divide your data into train, development, and test.
- Train on the training set.
- Adjust hyperparameters on the dev set (length of n-grams).
- Test on the test set.

Extrinsic Evaluation

- Imagine you have two models A and B.
- Use them on an end-to-end task: Machine Translation, speech recognition.
- Evaluate the accuracy on the task.
- Improve language model.
- This process can take a while.

Intrinsic Evaluation

Log Likelihood

Trigram Model:

$$\sum_{i=1}^{\text{number of tokens}} \log P(w_i | w_{i-2}, w_{i-1})$$

Bigram Model:

$$\sum_{i=1}^{\text{number of tokens}} \log P(w_i | w_{i-1})$$

Higher is better

Intrinsic Evaluation

Cross Entropy

Trigram Model:

$$-\frac{1}{\text{number of tokens}} \sum_{i=1}^{\text{number of tokens}} \log P(w_i | w_{i-2}, w_{i-1})$$

Bigram Model:

$$-\frac{1}{\text{number of tokens}} \sum_{i=1}^{\text{number of tokens}} \log P(w_i | w_{i-1})$$

Lower is better

Intrinsic Evaluation

Perplexity

Trigram Model:

$$2^{-\frac{1}{number \ of \ tokens} \sum_{i=1}^{number \ of \ tokens} \log P(w_i | w_{i-2}, w_{i-1})}$$

Bigram Model:

$$2^{-\frac{1}{number \ of \ tokens} \sum_{i=1}^{number \ of \ tokens} \log P(w_i | w_{i-1})}$$

Lower is better

MLE and Overfitting

MLE recap

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \mid w_i)}{c(w_i)}$$

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(cat \mid the) = \frac{3}{6} = 0.5$$

$$P(dog \mid the) = \frac{1}{6} = 0.167$$

$$P(mat \mid the) = \frac{1}{6} = 0.167$$

$$P(the \mid on) = \frac{2}{2} = 1.0$$

MLE recap

$$P(w_i \mid w_{i-1})_{MLE} = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(cat \mid the)_{MLE} = \frac{3}{6} = 0.5$$

$$P(dog \mid the)_{MLE} = \frac{1}{6} = 0.167$$

$$P(mat \mid the)_{MLE} = \frac{1}{6} = 0.167$$

$$P(the \mid on)_{MLE} = \frac{2}{2} = 1.0$$

Overfitting

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

- You can achieve very low perplexity on training.
- But test data can be different from training.

Overfitting

TEST

< s > the dog caught the cat < /s >

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

- You can achieve very low perplexity on training.
- But test data can be different from training.

Overfitting

TEST

< s > the dog caught the cat < /s >

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(\text{caught} \mid \text{cat})_{MLE} = \frac{1}{3}$$

$$P(\text{caught} \mid \text{dog})_{MLE} = 0$$

Overfitting

TEST

< s > the dog caught the cat < /s >

Test probability is 0!

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(\text{caught} \mid \text{cat})_{MLE} = \frac{1}{3}$$

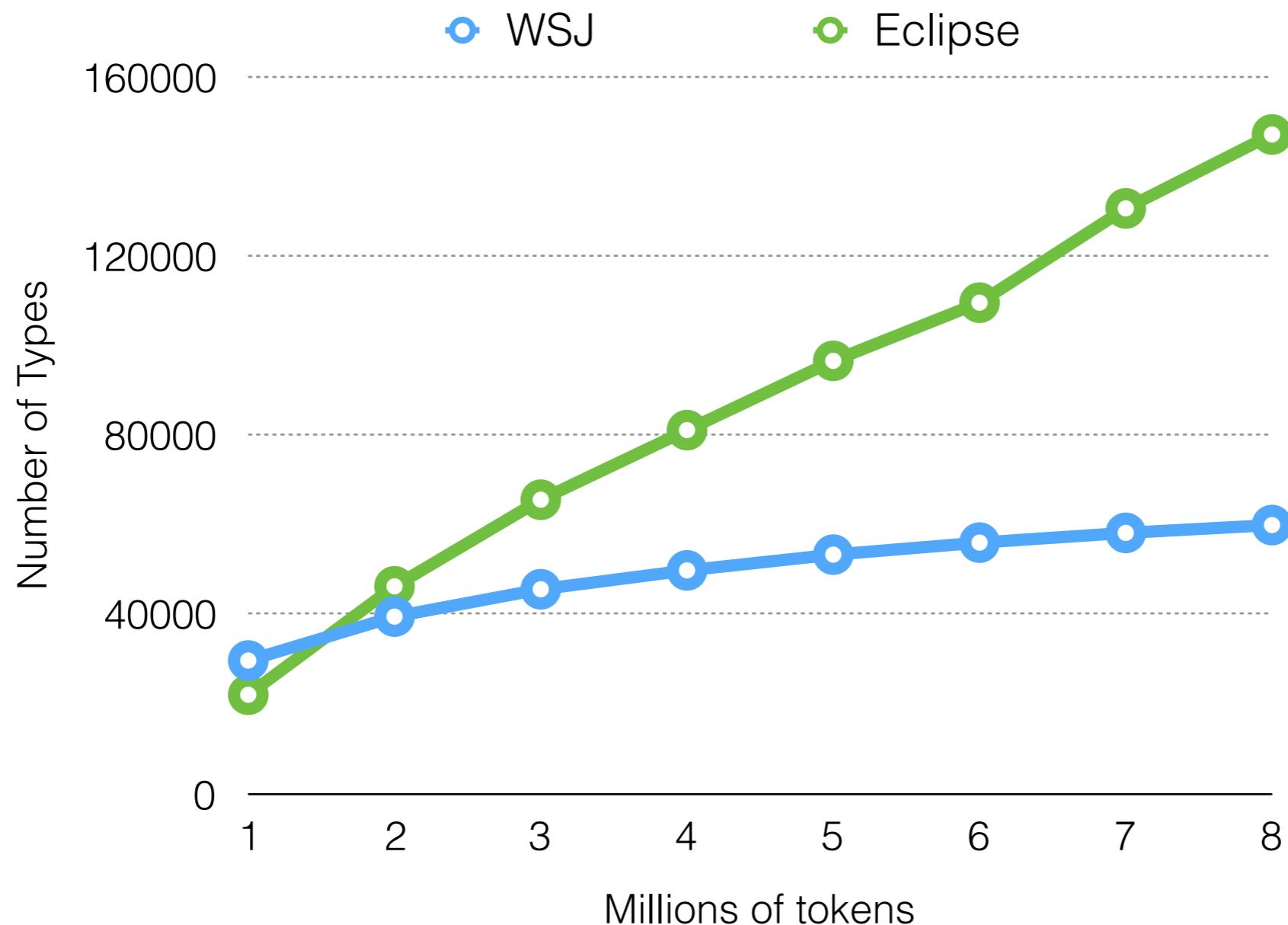
$$P(\text{caught} \mid \text{dog})_{MLE} = 0$$

Generalization

We need to train models that generalize well.

- Train n-gram models where 'n' is small
- Train n-gram models on large collections of data.

Tokens Vs Types



n-gram Language Models

l-gram

sharon
talks
bush
iPhone
mercurial

...

Number of unique l-grams in Language: 60×10^3

n-gram Language Models

2-gram

sharon	x	sharon
talks		talks
bush	x	bush
iPhone		iPhone
mercurial		mercurial
...		...

Number of unique 2-grams in Language:

3.6×10^9

n-gram Language Models

3-gram

sharon		sharon		sharon
talks		talks		talks
bush	×	bush	×	bush
iPhone		iPhone		iPhone
mercurial		mercurial		mercurial
...	

Number of unique 3-grams in Language:

2.16×10^{14}

How about Google n-grams ?

All Our N-gram are Belong to You

Posted: Thursday, August 03, 2006



Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word [n-gram models](#) for a variety of R&D projects, such as [statistical machine translation](#), speech recognition, [spelling correction](#), entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing [infrastructure](#) to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of one trillion words from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together. That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Watch for an announcement at the Linguistics Data Consortium ([LDC](#)), who will be distributing it soon, and then order your set of 6 DVDs. And [let us hear from you](#) - we're excited to hear what you will do with the data, and we're always interested in feedback about this dataset, or other potential datasets that might be useful for the research community.

Update (22 Sept. 2006): The LDC now has the [data available](#) in their catalog. The counts are as follows:

Research at Google
google.com/+ResearchatGoogle
vx, CS+x
G+ Follow +1
+ 1,124,360

Labels

Archive

Feed

Number of unique 3-grams:

2.16×10^{14}

How about Google n-grams ?

All Our N-gram are Belong to You

Posted: Thursday, August 03, 2006

G+ 67



Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word [n-gram models](#) for a variety of R&D projects, such as [statistical machine translation](#), speech recognition, [spelling correction](#), entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing [infrastructure](#) to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of one trillion words from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together. That's why we decided to share this enormous dataset with everyone. We processed [1,024,908,267,229 words](#) of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Watch for an announcement at the Linguistics Data Consortium ([LDC](#)), who will be distributing it soon, and then order your set of 6 DVDs. And [let us hear from you](#) - we're excited to hear what you will do with the data, and we're always interested in feedback about this dataset, or other potential datasets that might be useful for the research community.

Update (22 Sept. 2006): The LDC now has the [data available](#) in their catalog. The counts are as follows:

Research at Google
google.com/+ResearchatGoogle
vx, CS+x

G+ Follow +1
+ 1,124,360

Labels

Archive

Feed

Number of unique 3-grams:

2.16×10^{14}

How about Google n-grams ?

$$\frac{\text{Running text in Google n-grams}}{\text{Number of unique 3-grams in Language:}} = \frac{1 \times 10^{12}}{2.16 \times 10^{14}} = 0.0046$$

The Web ?

$$\frac{\text{Number of words in the indexed web}}{\text{Number of unique 3-grams in Language:}} = \frac{4.4927 \times 10^{14}}{2.16 \times 10^{14}} = 2.07$$

The Web ?

$$\frac{\text{Number of words in the indexed web}}{\text{Number of unique 3-grams in Code:}} = \frac{4.4927 \times 10^{14}}{3.375 \times 10^{15}} = 0.133$$

Better Solution: Smoothing

- Just collecting a lot of data is not enough.
- Smoothing methods differ in how they move probability from seen n-grams to unseen n-grams.
- Neural network language models have a more elegant solution for smoothing.

Better Solution: Smoothing

TEST

< s > the dog caught the cat < /s >

Test probability is 0!

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(\text{caught} \mid \text{cat})_{MLE} = \frac{1}{3}$$

$$P(\text{caught} \mid \text{dog})_{MLE} = 0$$

Better Solution: Smoothing

TEST

< s > the dog caught the cat < /s >

Test probability is > 0

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(\text{caught} \mid \text{cat})_{\text{smooth}} = \frac{1 - x}{3}$$

$$P(\text{caught} \mid \text{dog})_{\text{smooth}} = \frac{x}{3}$$

Data Preprocessing

- Split data into train/dev/test
- Get word frequencies on train and keep top V most frequent words
- Replace the remaining words with <unk>. This accounts for Out Of Vocabulary (OOV) words at test time.

Add One (Laplace) Smoothing

- Pretend that we saw every word once more than it did
- Add one to all all n-gram counts, seen or unseen

MLE estimate of probabilities:

$$P(w_i \mid w_{i-1})_{MLE} = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add One (Laplace) Smoothing

- Pretend that we saw every word once more than it did
- Add one to all all n-gram counts, seen or unseen

MLE estimate of probabilities:

$$P(w_i \mid w_{i-1})_{MLE} = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate of probabilities:

$$P(w_i \mid w_{i-1})_{Add-1} = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Add One (Laplace) Smoothing

TEST

< s > the dog caught the cat < /s >

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(\text{caught} \mid \text{cat})_{MLE} = \frac{1}{3}$$

$$P(\text{caught} \mid \text{dog})_{MLE} = 0$$

Add One (Laplace) Smoothing

TEST

< s > the dog caught the cat < /s >

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(\text{caught} \mid \text{cat})_{MLE} = \frac{1}{3}$$

$$P(\text{caught} \mid \text{dog})_{Add-1} = \frac{c(\text{dog}, \text{caught}) + 1}{c(\text{dog}) + |\mathcal{V}|} = \frac{0 + 1}{1 + 8} = 0.11 \quad P(\text{caught} \mid \text{dog})_{MLE} = 0$$

Add One (Laplace) Smoothing

TEST

< s > the dog caught the cat < /s >

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(\text{caught} \mid \text{cat})_{\text{Add-1}} = \frac{c(\text{cat}, \text{caught}) + 1}{c(\text{cat}) + |\nu|} = \frac{1 + 1}{3 + 8} = 0.18$$

$$P(\text{caught} \mid \text{cat})_{\text{MLE}} = \frac{1}{3}$$

$$P(\text{caught} \mid \text{dog})_{\text{Add-1}} = \frac{c(\text{dog}, \text{caught}) + 1}{c(\text{dog}) + |\nu|} = \frac{0 + 1}{1 + 8} = 0.11$$

$$P(\text{caught} \mid \text{dog})_{\text{MLE}} = 0$$

Add One (Laplace) Smoothing

TEST

< s > the dog caught the cat < /s >

TRAIN

< s > the cat sat on the mat < /s >

< s > the dog sat on the cat < /s >

< s > the cat caught the mouse < /s >

$$P(\text{caught} \mid \text{cat})_{\text{Add-1}} = \frac{c(\text{cat}, \text{caught}) + 1}{c(\text{cat}) + |\nu|} = \frac{1 + 1}{3 + 8} = 0.18 \quad \downarrow \quad P(\text{caught} \mid \text{cat})_{\text{MLE}} = \frac{1}{3}$$

$$P(\text{caught} \mid \text{dog})_{\text{Add-1}} = \frac{c(\text{dog}, \text{caught}) + 1}{c(\text{dog}) + |\nu|} = \frac{0 + 1}{1 + 8} = 0.11 \quad \uparrow \quad P(\text{caught} \mid \text{dog})_{\text{MLE}} = 0$$

Add One (Laplace) Smoothing

- Add-1 smoothing can smooth excessively.
- Not used for language modeling.

Good Turing Smoothing

- Move probability mass from n-grams that occur $c + 1$ times to those that occur c times.
- Move probability from n-grams occurring once to unseen n-grams

Let n_c be the number of n-grams seen c times. (Frequency of Frequency)

For every n-gram with count c , new count $c^* = (c + 1) \frac{n_{c+1}}{n_c}$

Good Turing Smoothing (Josh Goodman)

You're fishing in a lake and there are 8 species of fish

- carp, perch, whitefish, trout, salmon, eel, catfish, bass

Good Turing Smoothing

(Josh Goodman)

You're fishing in a lake and there are 8 species of fish

- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish

- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

Good Turing Smoothing

(Josh Goodman)

You're fishing in a lake and there are 8 species of fish

- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish

- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout ?

Good Turing Smoothing

(Josh Goodman)

You're fishing in a lake and there are 8 species of fish

- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish

- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout ? $\frac{1}{18}$

Good Turing Smoothing

(Josh Goodman)

You're fishing in a lake and there are 8 species of fish

- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish

- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout ? $\frac{1}{18}$

How likely is the next species new ?

Good Turing Smoothing

(Josh Goodman)

You're fishing in a lake and there are 8 species of fish

- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish

- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout ? $\frac{1}{18}$

How likely is the next species new ? $\frac{n_1}{N} = \frac{3}{18}$

Good Turing Smoothing

(Josh Goodman)

You're fishing in a lake and there are 8 species of fish

- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish

- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout ? $\frac{1}{18}$

How likely is the next species new ? $\frac{n_1}{N} = \frac{3}{18}$

In that case, what is the probability of the next species being trout ?

Good Turing Smoothing

(Josh Goodman)

You're fishing in a lake and there are 8 species of fish

- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish

- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout ? $\frac{1}{18}$

How likely is the next species new ? $\frac{n_1}{N} = \frac{3}{18}$

In that case, what is the probability of the next species being trout ? $< \frac{1}{18}$

Good Turing Smoothing (Josh Goodman)

$$P *_{GT} (\text{things with zero frequency}) = \frac{n_1}{18} = \frac{3}{18} \quad , \quad c^* = (c + 1) \frac{n_{c+1}}{n_c}$$

Probability of Unseen (bass or catfish) Probability seen once (bass or catfish)

Good Turing Smoothing (Josh Goodman)

$$P *_{GT} (\text{things with zero frequency}) = \frac{n_1}{18} = \frac{3}{18} \quad , \quad c^* = (c + 1) \frac{n_{c+1}}{n_c}$$

Probability of Unseen (bass or catfish) Probability seen once (bass or catfish)

$$P_{MLE}(\text{bass}) = \frac{0}{18}$$

$$P *_{GT} (\text{bass}) = \frac{3}{18}$$

Good Turing Smoothing

(Josh Goodman)

$$P *_{GT} (\text{things with zero frequency}) = \frac{n_1}{18} = \frac{3}{18} \quad , \quad c^* = (c + 1) \frac{n_{c+1}}{n_c}$$

Probability of Unseen (bass or catfish) Probability seen once (bass or catfish)

$$P_{MLE}(\text{bass}) = \frac{0}{18}$$

$$P_{MLE}(\text{trout}) = \frac{1}{18}$$

$$c^*(\text{trout}) = (1 + 1) \frac{1}{3} = \frac{2}{3}$$

$$P *_{GT} (\text{bass}) = \frac{3}{18}$$

$$P_{GT} * (\text{trout}) = \frac{\frac{2}{3}}{18} = \frac{1}{27}$$

Backoff and interpolation

If the context is less frequent, then use shorter contexts.

Backoff:

- If n-gram is frequent, use it.
- else, use lower order n-gram.

Interpolation

- Mix all n-gram contexts.
- Works better in practice than backoff.

Interpolation

Simple interpolation:

$$P_{\text{interp}}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i)$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$

Interpolation

Simple interpolation:

$$P_{\text{interp}}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i)$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$

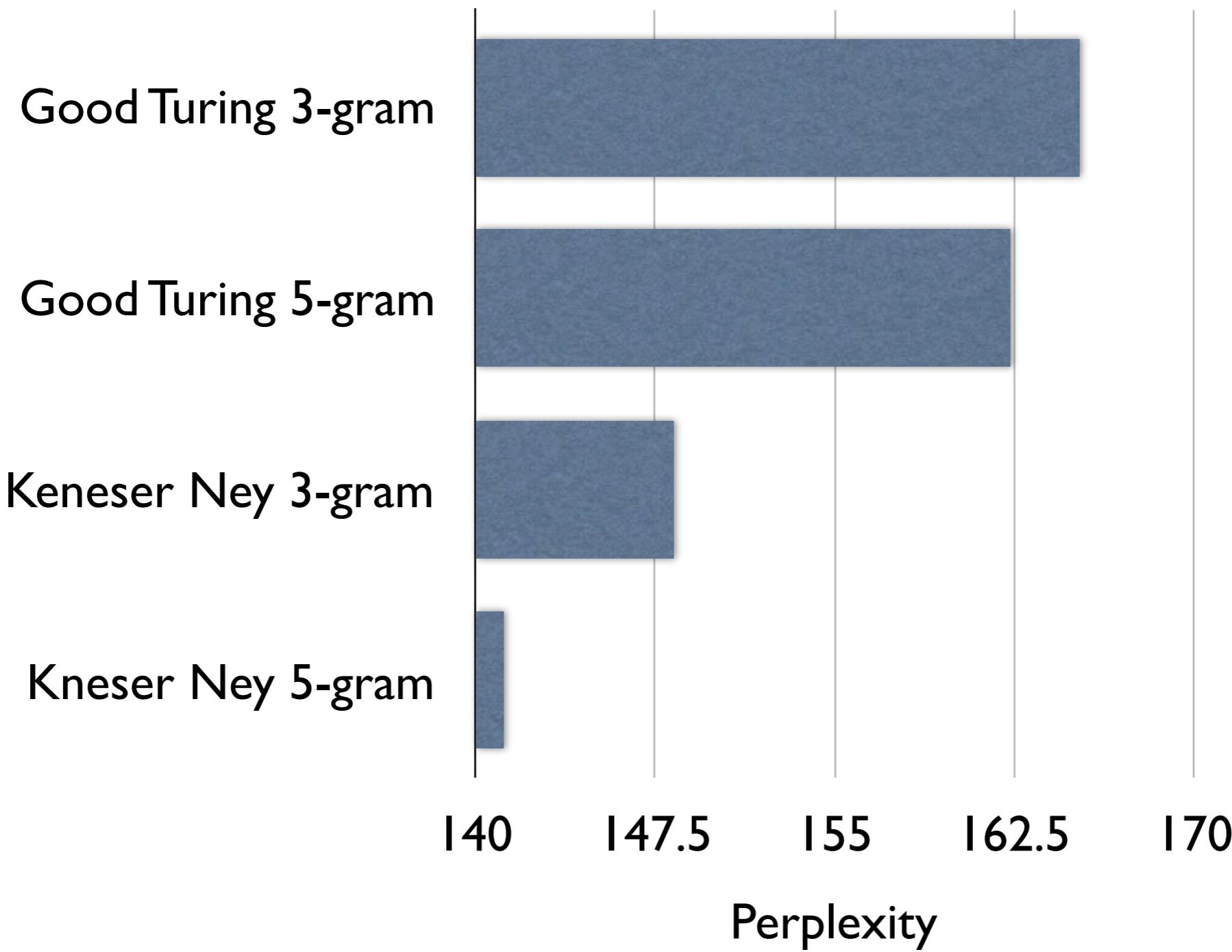
Conditioning interpolation weights on context:

$$\begin{aligned} P_{\text{interp}}(w_i \mid w_{i-2}, w_{i-1}) &= \lambda_1(w_{i-2}, w_{i-1}) P(w_i \mid w_{i-2}, w_{i-1}) \\ &\quad + \lambda_2(w_{i-2}, w_{i-1}) P(w_i \mid w_{i-1}) + \lambda_3(w_{i-2}, w_{i-1}) P(w_i) \end{aligned}$$

Interpolation

- Jeninek Mercer
- Absolute discounting
- Modified Kneser Ney

Performance on the Penn Treebank Dataset



Cache Language Models

- Work well for code.
- Intuition is that recently used words (variables) are likely to appear.

On the “Naturalness” of Buggy Code. (Ray et al., 2015)

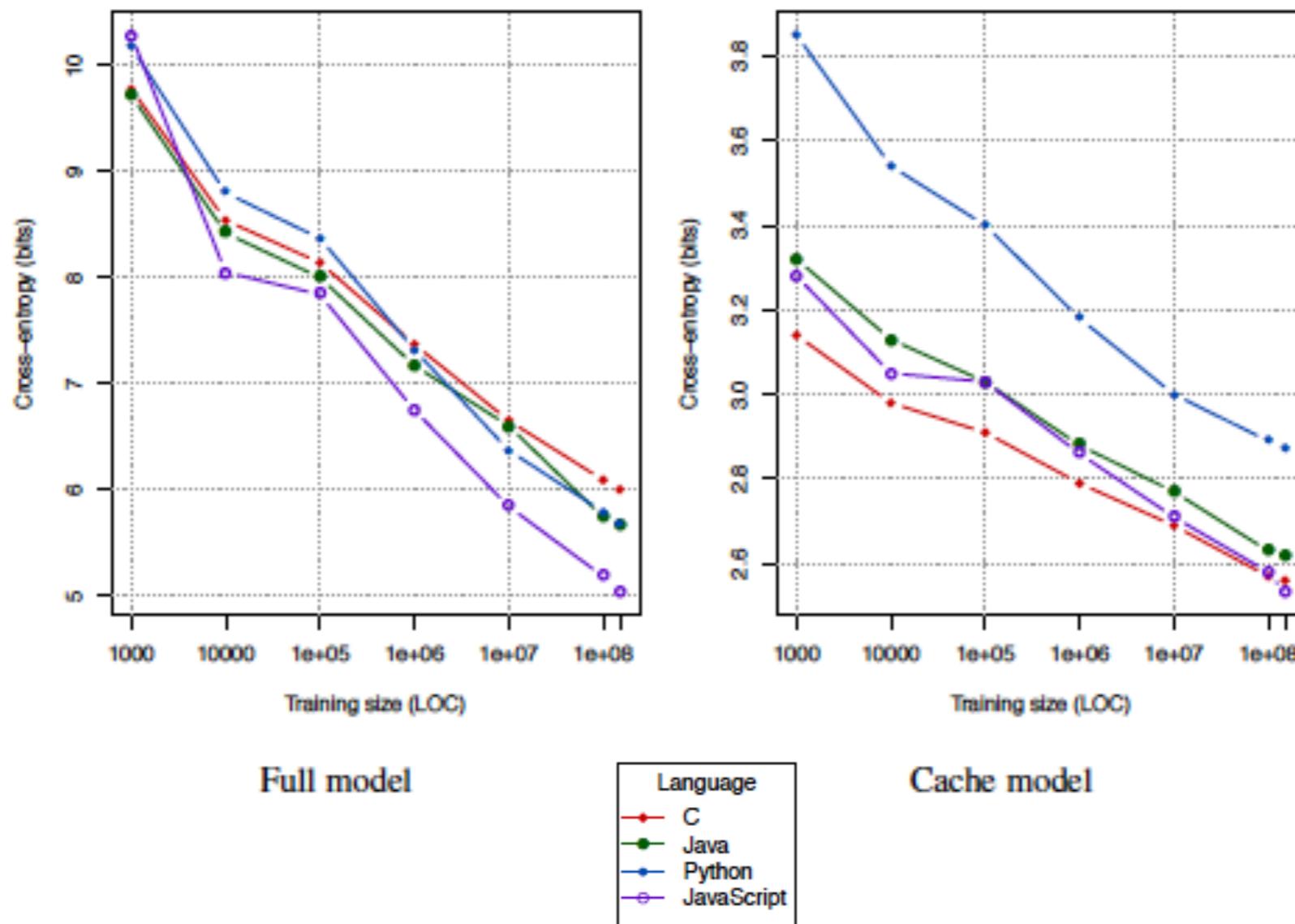
Cache Language Models

- Work well for code.
- Intuition is that recently used words (variables) are likely to appear.

$$P_{CACHE}(w_i \mid \text{history}) = \lambda P(w_i \mid w_{i-2}, w_{i-1}) + (1 - \lambda) \frac{c(w \in \text{history})}{|\text{history}|}$$

On the “Naturalness” of Buggy Code. (Ray et al., 2015)

Cache Language Models



Stefan Fiott, 2015 (Masters Thesis)

General approach for training n-gram models

- Collect n-gram counts
- Smooth for unseen n-grams
- Large number of parameters
- Fast probability computation

So Far...

Standard n-
gram

Model Size



Training Time



Query Time



Continuous Space Language Models

Feed Forward Neural Network Language Models

Motivation for Neural Network Language Models

TEST

< s > ibm was bought < /s >

$$c(ibm \text{ was bought}) = 0$$

TRAIN

< s > apple was sold < /s >

Motivation for Neural Network Language Models

TEST

< s > ibm was bought < /s >

$$c(ibm \text{ was bought}) = 0$$

TRAIN

< s > apple was sold < /s >

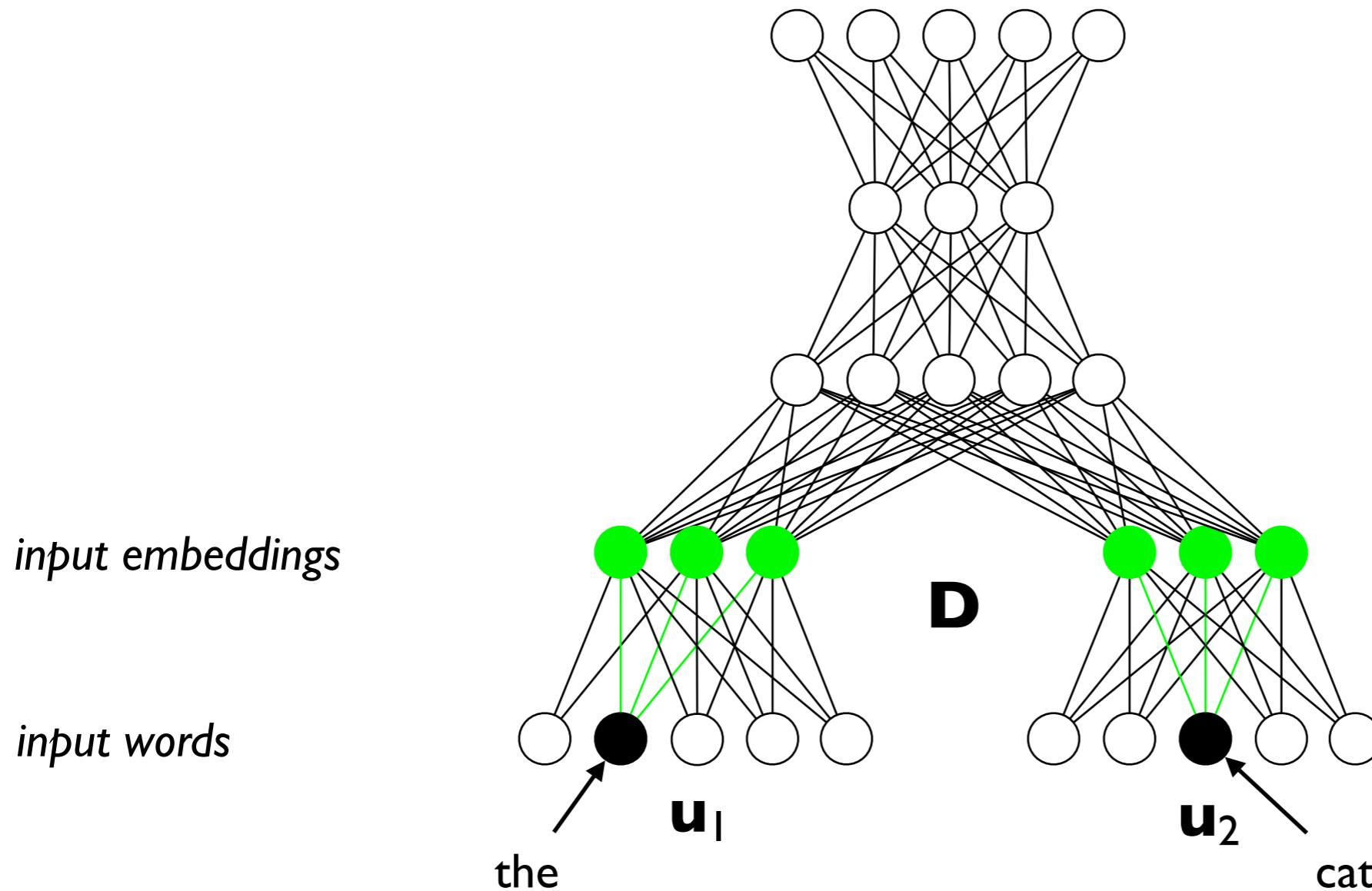
bought \approx *sold*

ibm \approx *apple*

Neural Probabilistic Language Models

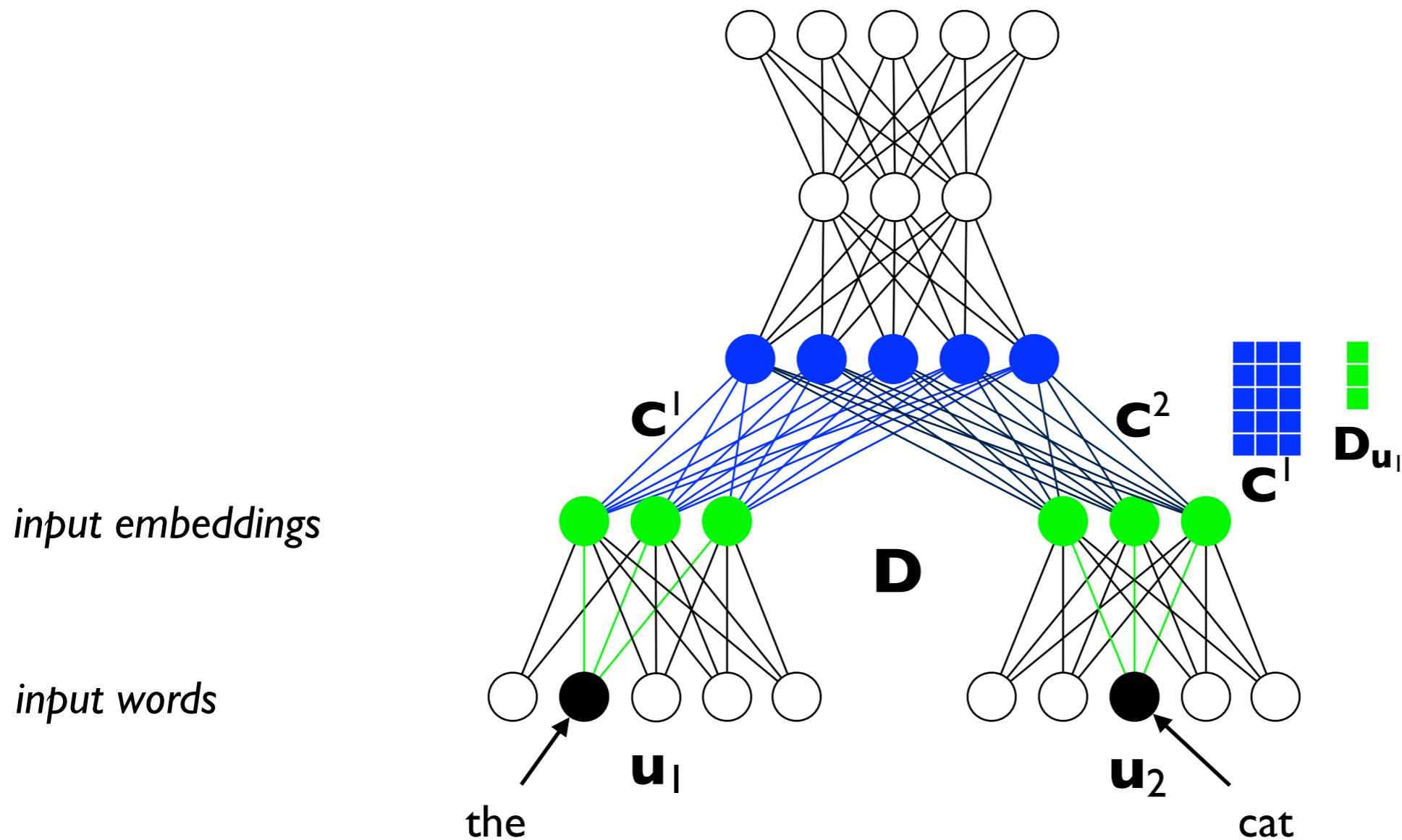
Bengio et al., 2003

$P(\text{sat} \mid \text{the cat})$



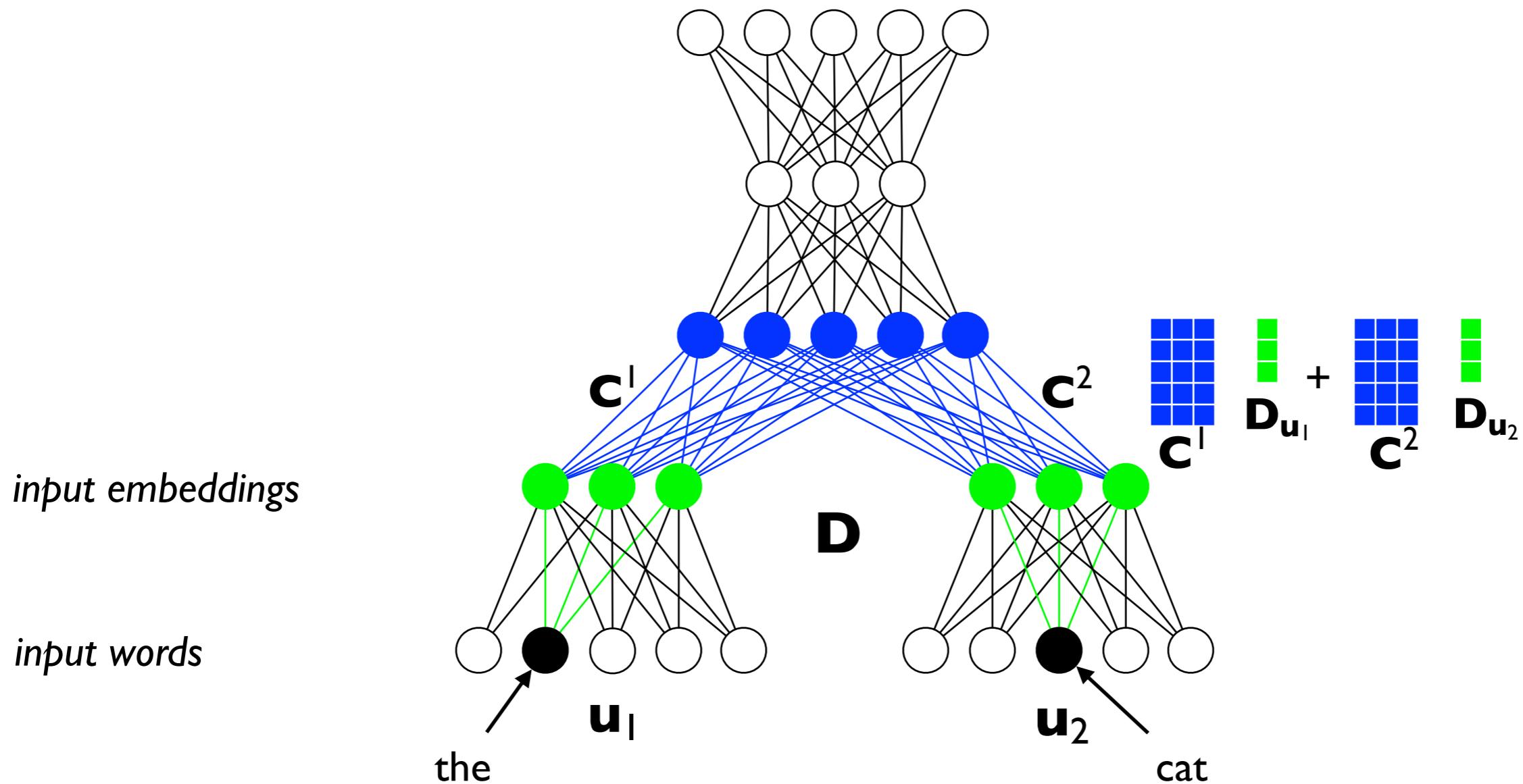
Neural Probabilistic Language Models

$P(\text{sat} \mid \text{the cat})$



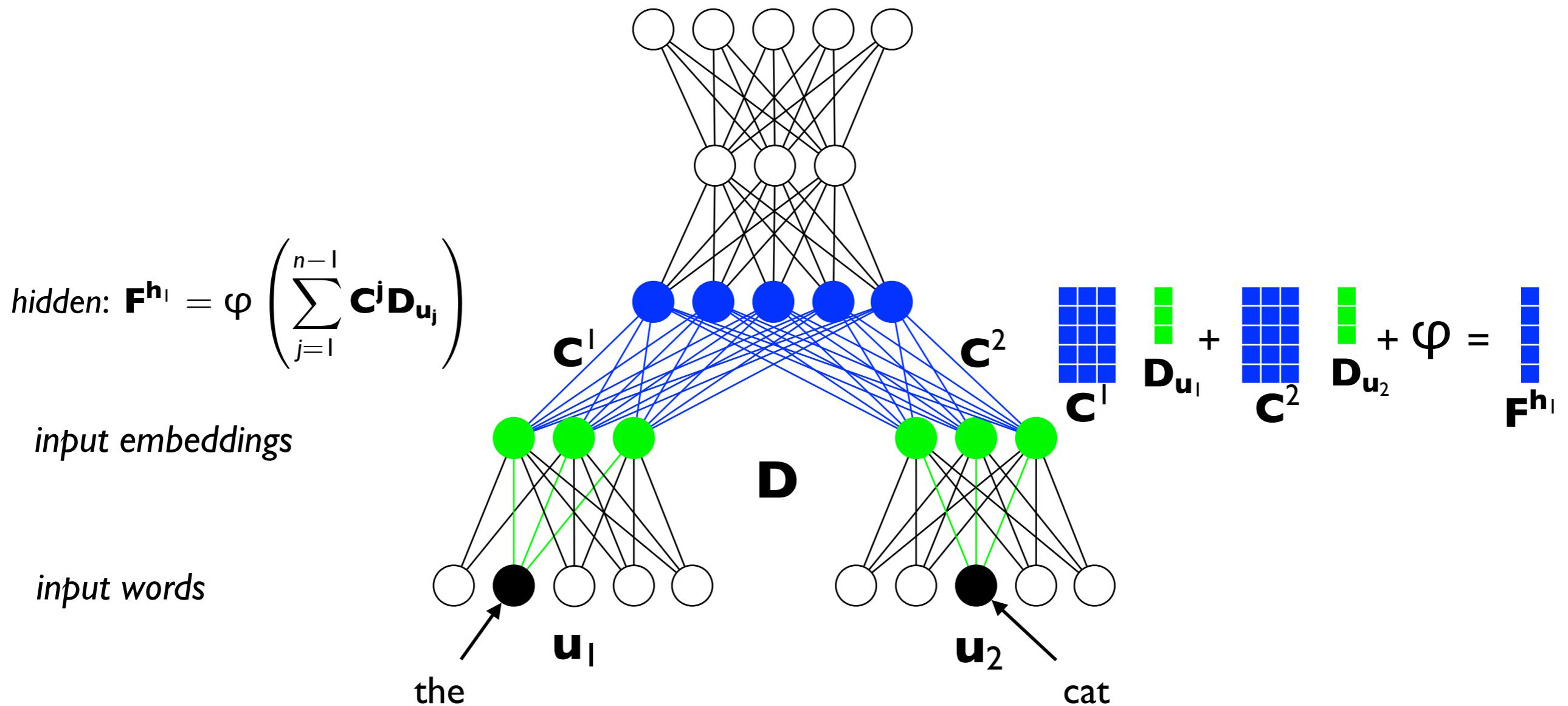
Neural Probabilistic Language Models

$P(\text{sat} \mid \text{the cat})$



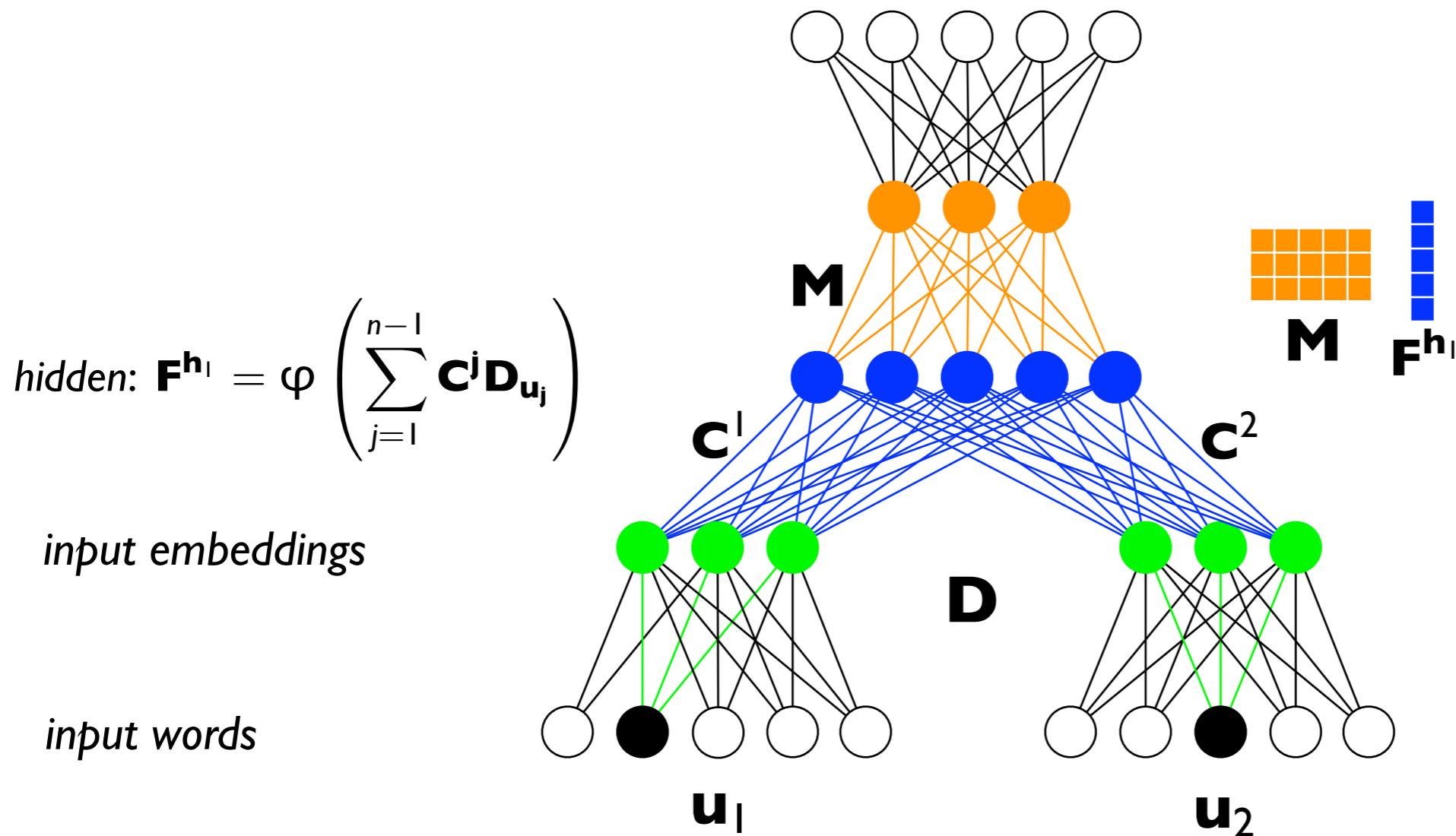
Neural Probabilistic Language Models

$P(\text{sat} \mid \text{the cat})$



Neural Probabilistic Language Models

$P(\text{sat} \mid \text{the cat})$



Neural Probabilistic Language Models

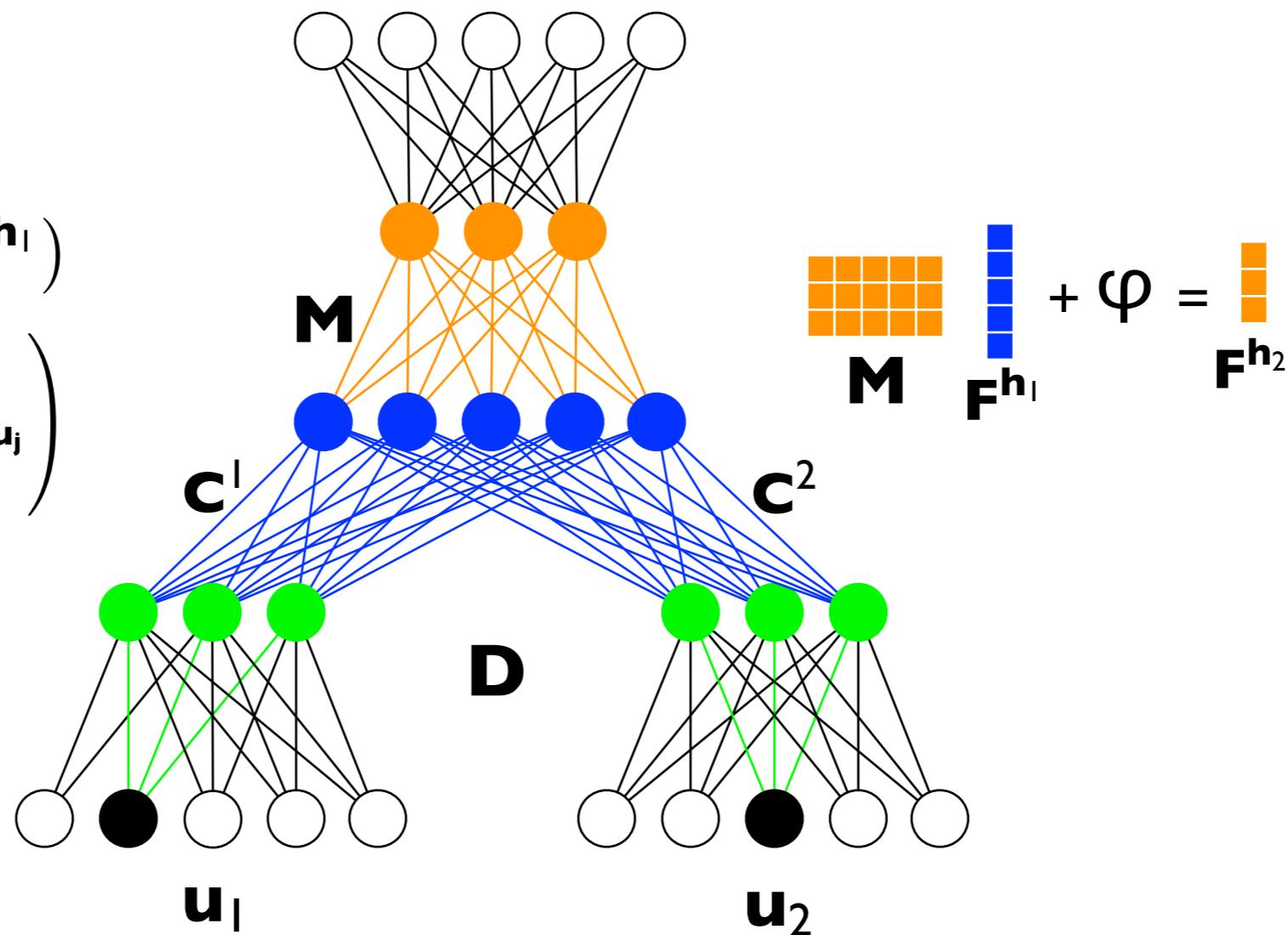
$P(\text{sat} \mid \text{the cat})$

$$\text{hidden: } \mathbf{F}^{\mathbf{h}_2} = \varphi (\mathbf{M} \mathbf{F}^{\mathbf{h}_1})$$

$$\text{hidden: } \mathbf{F}^{\mathbf{h}_1} = \varphi \left(\sum_{j=1}^{n-1} \mathbf{C}^j \mathbf{D}_{\mathbf{u}_j} \right)$$

input embeddings

input words



Neural Probabilistic Language Models

$P(\text{sat} \mid \text{the cat})$

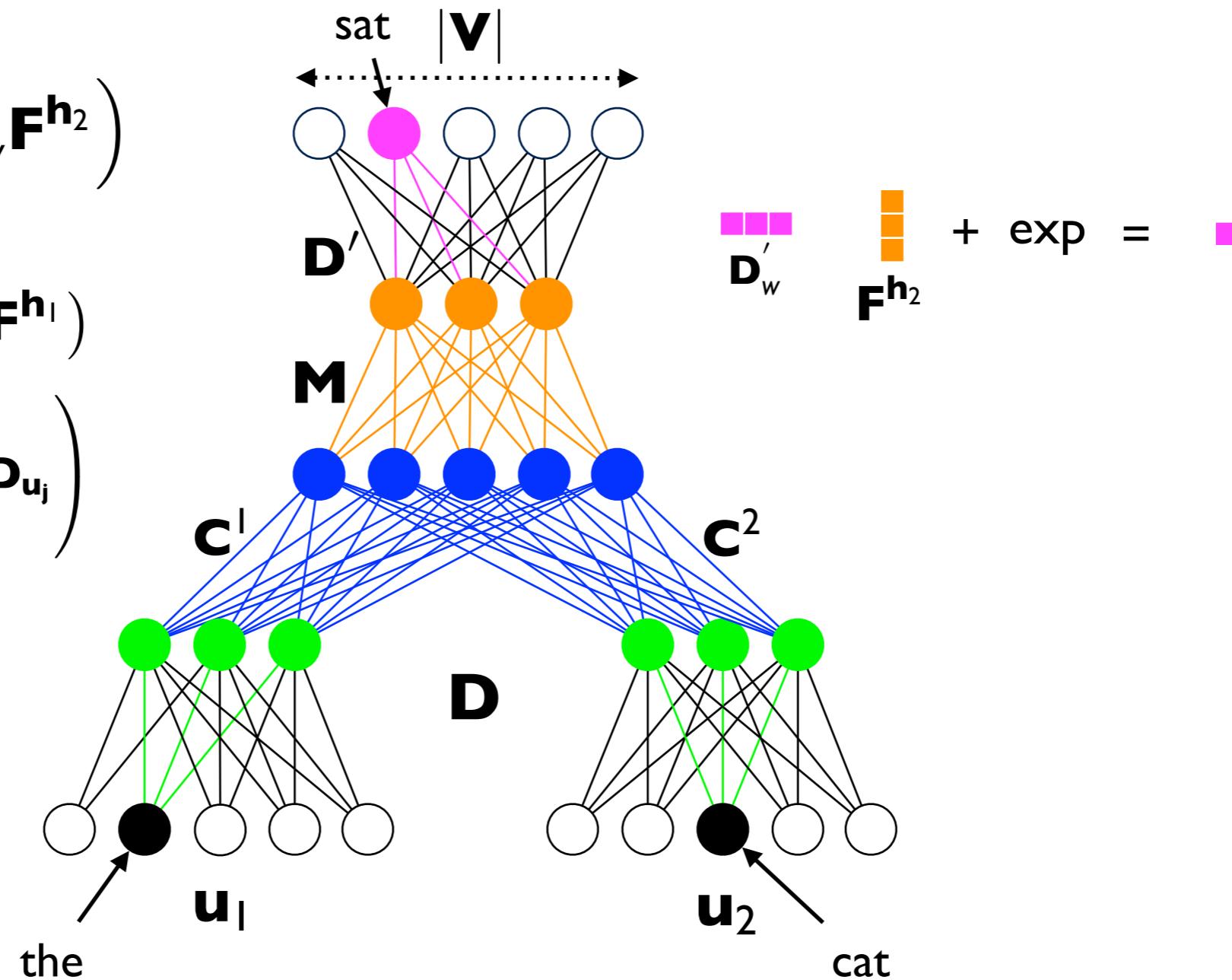
$$p(w \mid \mathbf{u}) = \exp \left(\mathbf{D}'_w \mathbf{F}^{\mathbf{h}_2} \right)$$

$$\text{hidden: } \mathbf{F}^{\mathbf{h}_2} = \varphi (\mathbf{M} \mathbf{F}^{\mathbf{h}_1})$$

$$\text{hidden: } \mathbf{F}^{\mathbf{h}_1} = \varphi \left(\sum_{j=1}^{n-1} \mathbf{C}^j \mathbf{D}_{\mathbf{u}_j} \right)$$

input embeddings

input words



Neural Probabilistic Language Models

$P(\text{sat} \mid \text{the cat})$

normalization : $Z(\mathbf{u}) = \sum_{w' \in V} p(w' \mid \mathbf{u})$

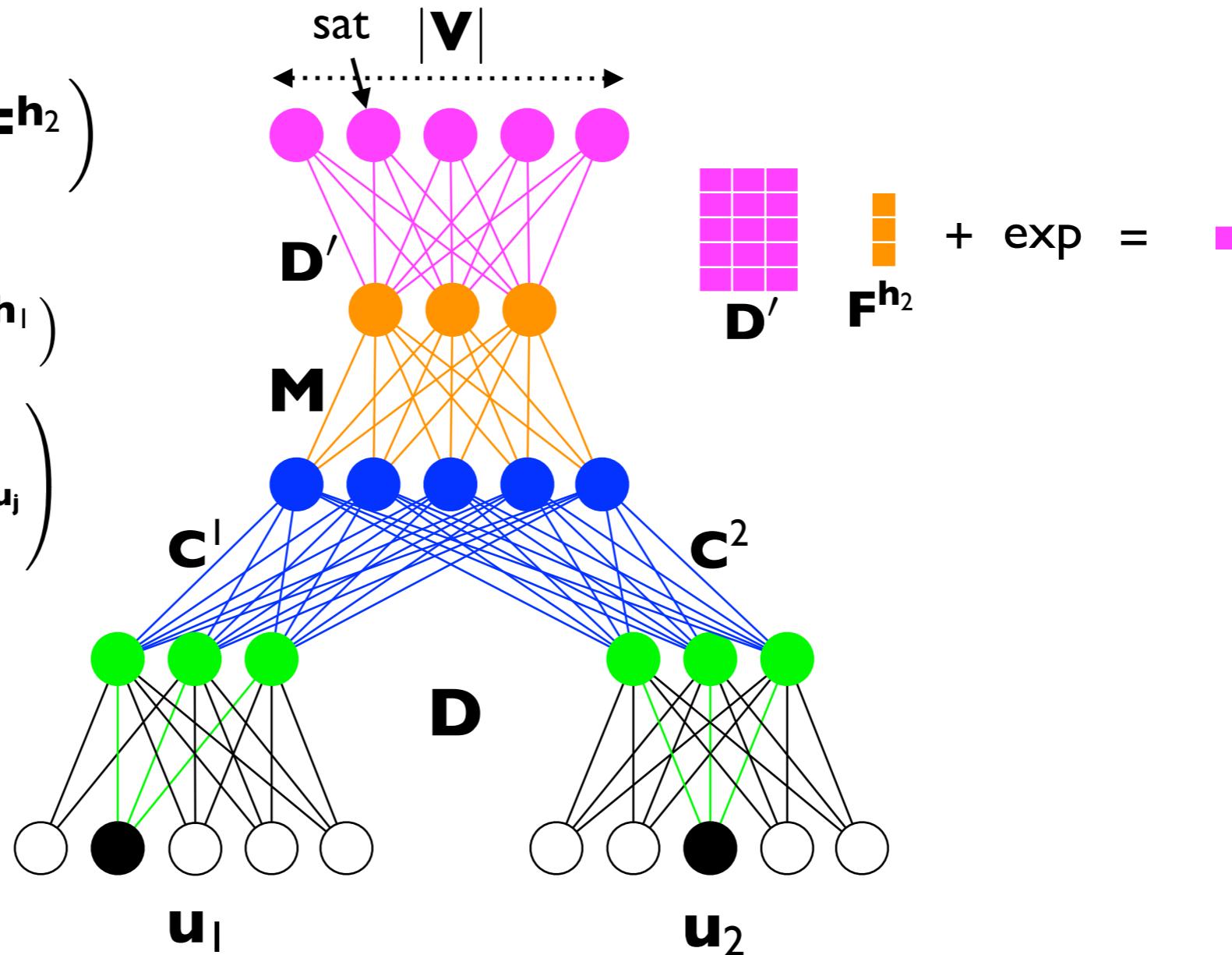
$$p(w \mid \mathbf{u}) = \exp \left(\mathbf{D}'_w \mathbf{F}^{\mathbf{h}_2} \right)$$

$$\text{hidden: } \mathbf{F}^{\mathbf{h}_2} = \varphi (\mathbf{M} \mathbf{F}^{\mathbf{h}_1})$$

$$\text{hidden: } \mathbf{F}^{\mathbf{h}_1} = \varphi \left(\sum_{j=1}^{n-1} \mathbf{C}^j \mathbf{D}_{\mathbf{u}_j} \right)$$

input embeddings

input words



Neural Probabilistic Language Models

$P(\text{sat} \mid \text{the cat})$

normalization : $Z(\mathbf{u}) = \sum_{w' \in V} p(w' \mid \mathbf{u})$

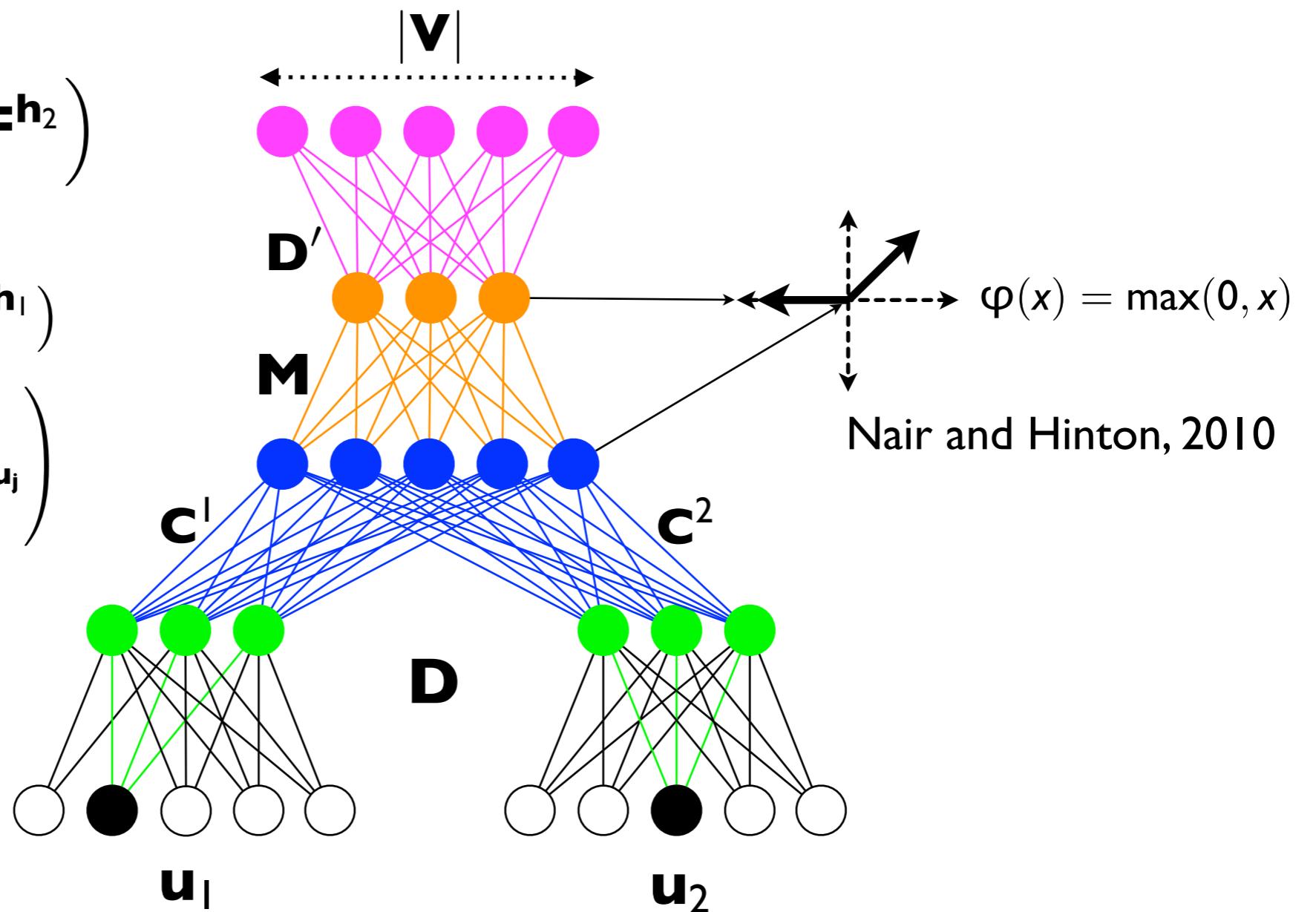
$$p(w \mid \mathbf{u}) = \exp \left(\mathbf{D}'_w \mathbf{F}^{\mathbf{h}_2} \right)$$

$$\text{hidden: } \mathbf{F}^{\mathbf{h}_2} = \varphi (\mathbf{M} \mathbf{F}^{\mathbf{h}_1})$$

$$\text{hidden: } \mathbf{F}^{\mathbf{h}_1} = \varphi \left(\sum_{j=1}^{n-1} \mathbf{C}^j \mathbf{D}_{\mathbf{u}_j} \right)$$

input embeddings

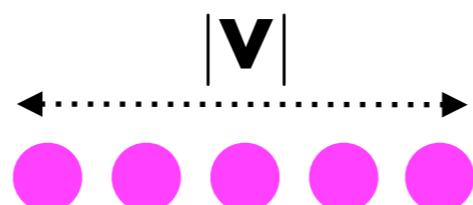
input words



Neural Probabilistic Language Models

$$\text{normalization : } Z(\mathbf{u}) = \sum_{w' \in V} p(w' | \mathbf{u})$$

$$p(w | \mathbf{u}) = \exp \left(\mathbf{D}_w' \mathbf{F}^{\mathbf{h}_2} \right)$$



Neural Probabilistic Language Models

$$P(w \mid \mathbf{u}) = \frac{\exp \left(\mathbf{D}_w' \mathbf{F}^{\mathbf{h}_2} \right)}{\mathbf{z}(\mathbf{u})}$$

Maximum Likelihood Training

$$\theta_{ML} = \arg \max_{\theta} P(w | \mathbf{u})$$

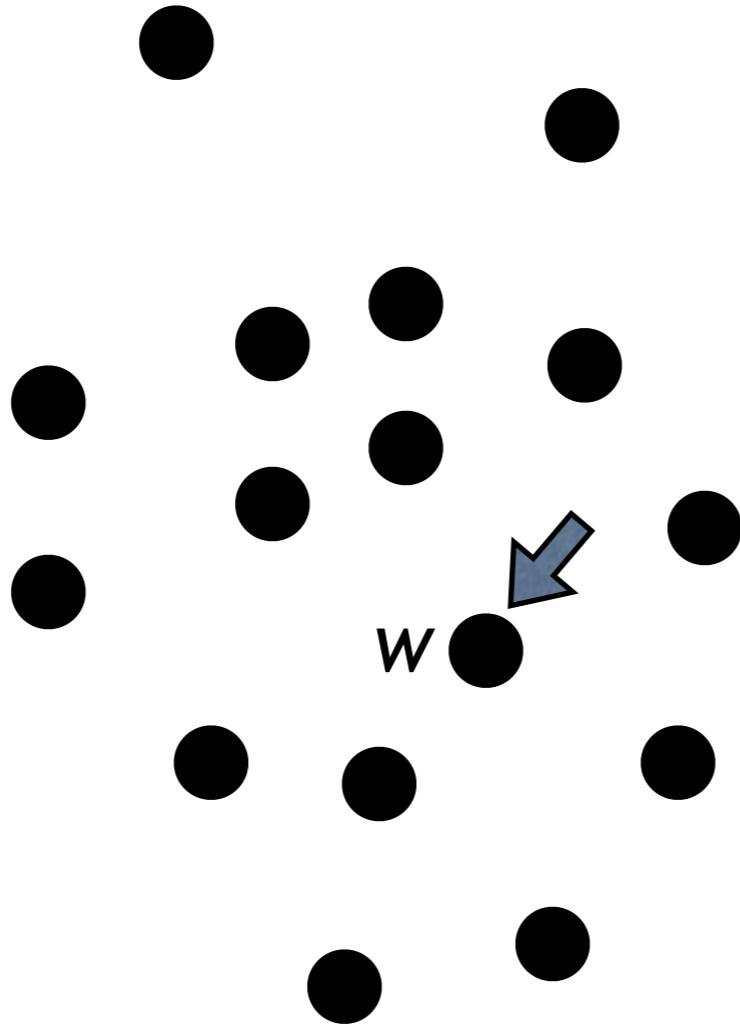
Maximum Likelihood Training

$$\theta_{ML} = \arg \max_{\theta} \log P(w | u)$$

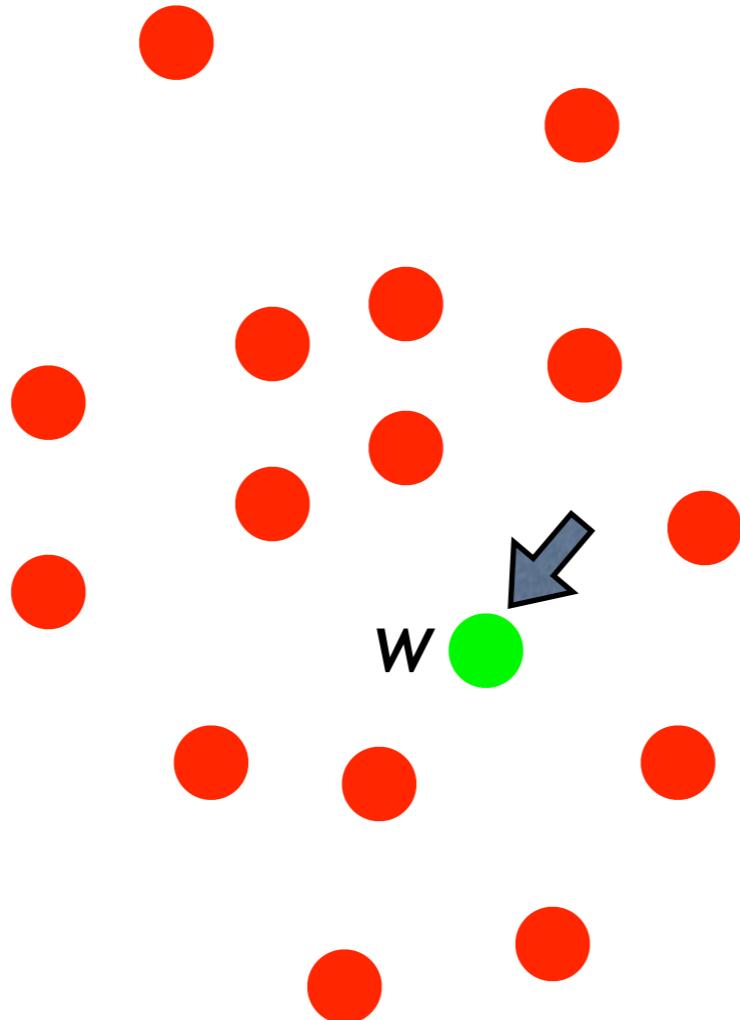
Maximum Likelihood Training

$$\theta_{ML} = \arg \max_{\theta} \log \frac{\exp \left(\mathbf{D}_w' \mathbf{F}^{\mathbf{h}_2} \right)}{\mathbf{z}(\mathbf{u})}$$

Maximum Likelihood Training



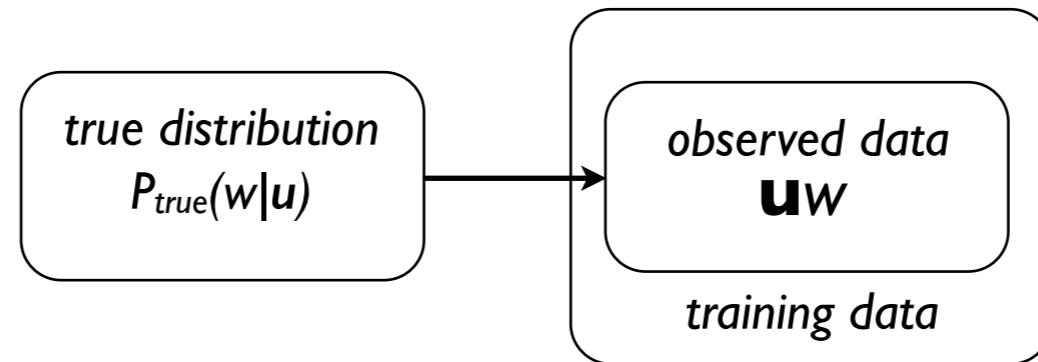
Maximum Likelihood Training



Maximum Likelihood Training

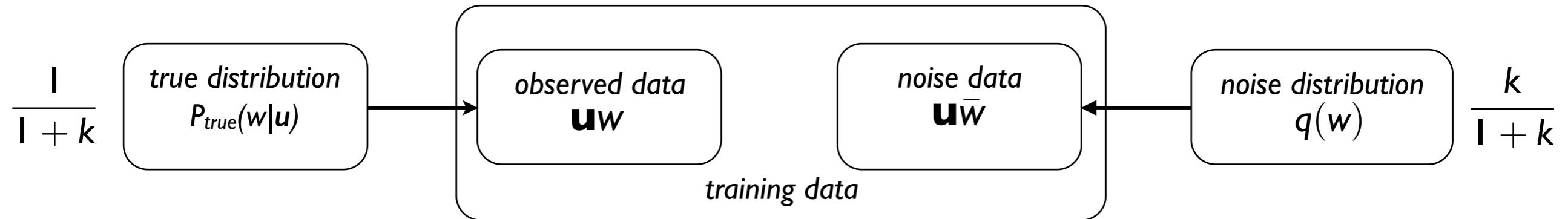
- Perform stochastic gradient descent :
 - Compute $P(w|u)$ using Forward Propagation
 - Compute gradient with Backward Propagation
- Very slow training and decoding times

Maximum Likelihood Training



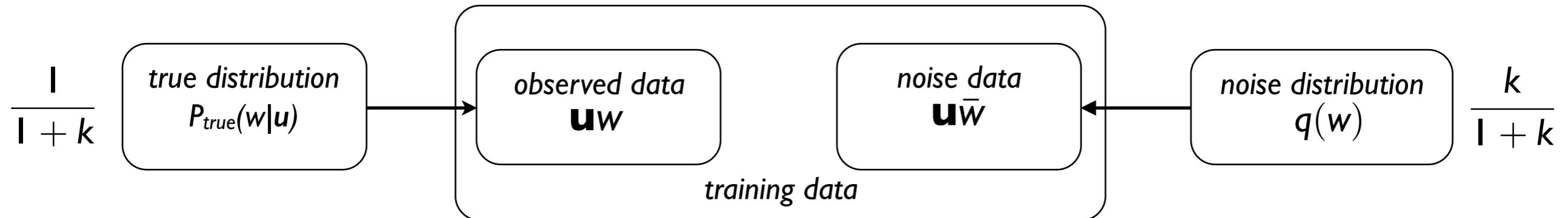
The cat sat

Noise Contrastive Estimation



The cat sat

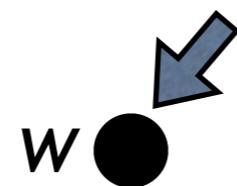
Noise Contrastive Estimation



The cat sat

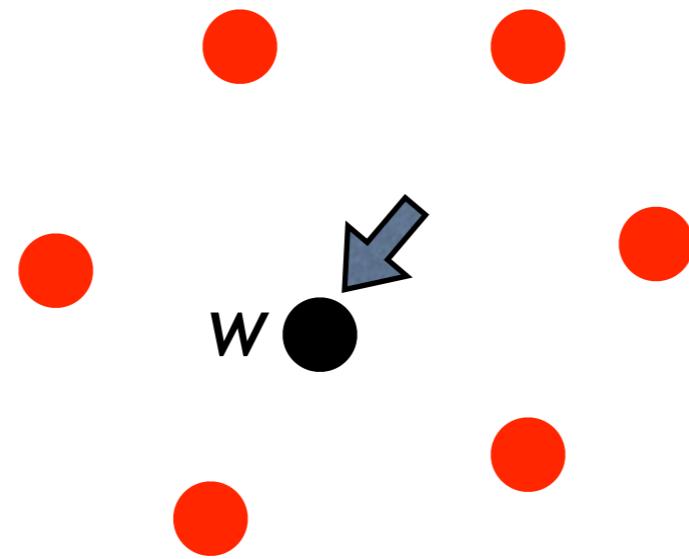
The cat pig
The cat hat
The cat on
...

Noise Contrastive Estimation



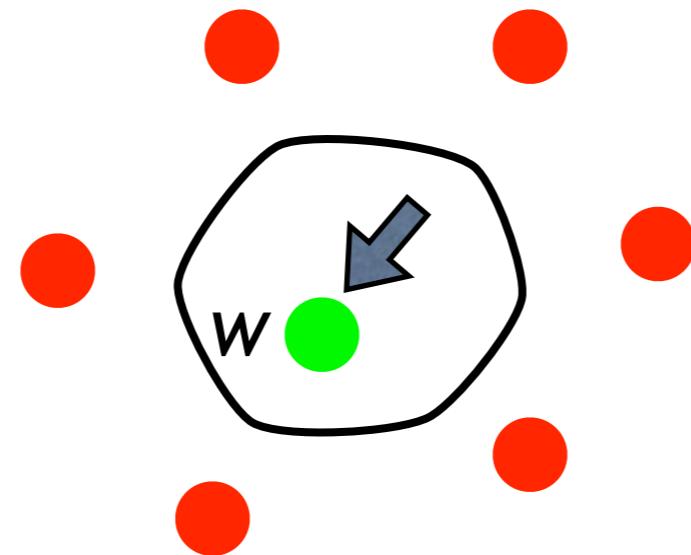
Noise Contrastive Estimation

k noise samples

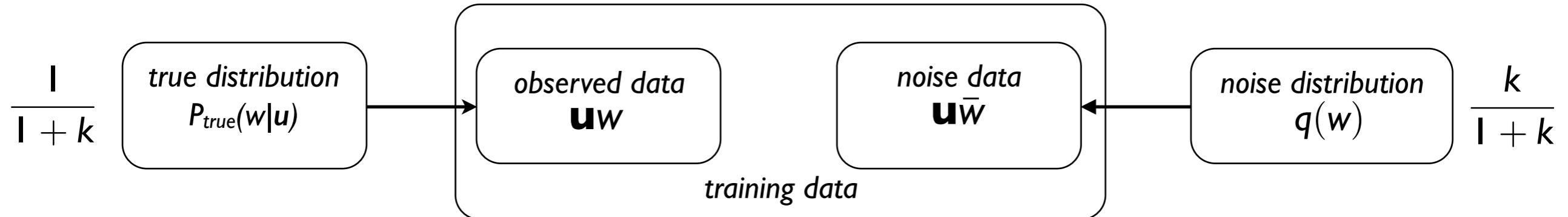


Noise Contrastive Estimation

k noise samples



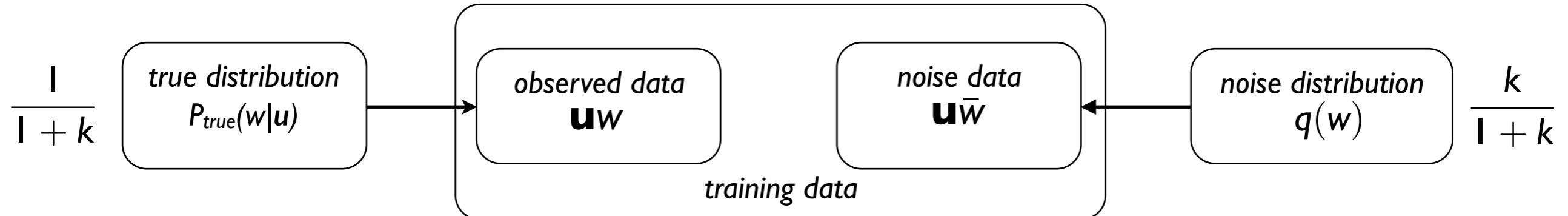
Noise Contrastive Estimation



The cat sat

The cat pig
The cat hat
The cat on
...

Noise Contrastive Estimation

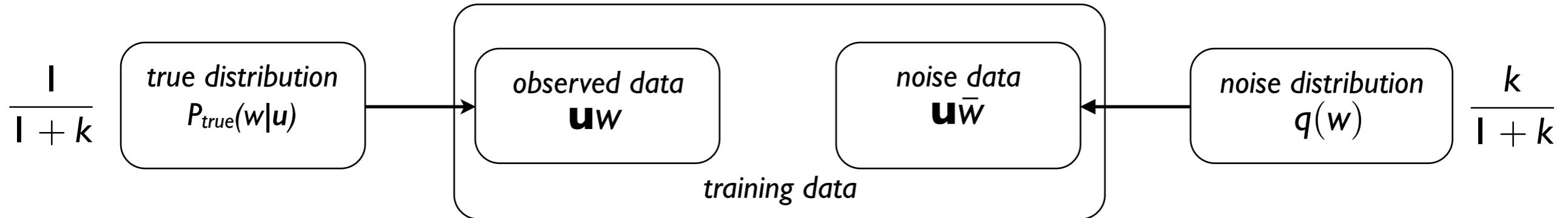


The cat sat

The cat pig
The cat hat
The cat on
...

$P(w \text{ is true} | uw)$

Noise Contrastive Estimation

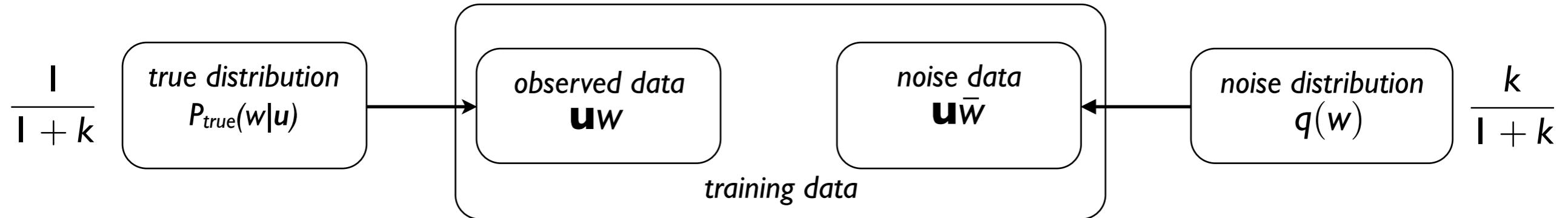


The cat sat

The cat pig
The cat hat
The cat on
...

$$P(w \text{ is true} \mid \mathbf{u}w) = \frac{P(w \mid \mathbf{u})}{P(w \mid \mathbf{u}) + kq(w)}$$

Noise Contrastive Estimation



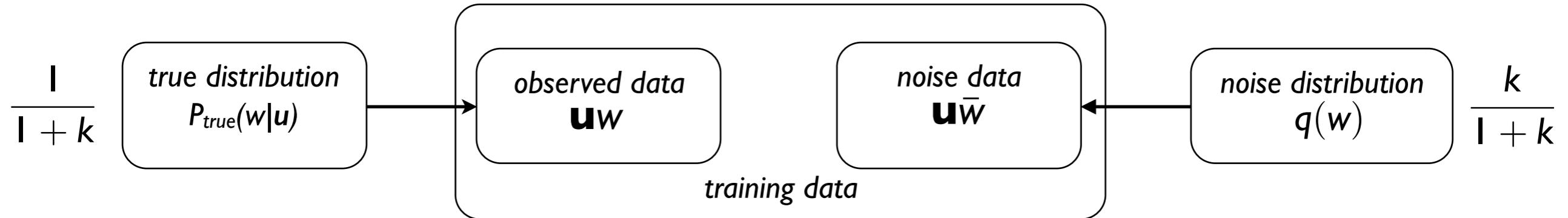
The cat sat

$$P(w \text{ is true} \mid \mathbf{u}_w) = \frac{P(w \mid \mathbf{u})}{P(w \mid \mathbf{u}) + kq(w)}$$

The cat pig
The cat hat
The cat on
...

$$P(\bar{w}_j \text{ is noise} \mid \mathbf{u}_{\bar{w}_j})$$

Noise Contrastive Estimation



The cat sat

$$P(w \text{ is true} \mid \mathbf{u}w)$$

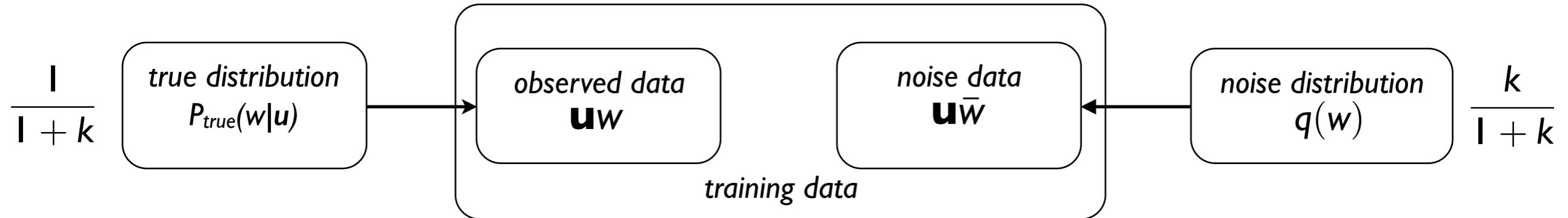
$$\frac{P(w \mid \mathbf{u})}{P(w \mid \mathbf{u}) + kq(w)}$$

The cat pig
The cat hat
The cat on
...

$$P(\bar{w}_j \text{ is noise} \mid \mathbf{u}\bar{w}_j)$$

$$\frac{q(\bar{w}_j)}{P(\bar{w}_j \mid \mathbf{u}) + kq(w)}$$

Noise Contrastive Estimation



The cat sat

$$\frac{P(C = 0 \mid \mathbf{u}w)}{P(w \mid \mathbf{u}) + kq(w)}$$

The cat pig
The cat hat
The cat on
...

$$\frac{P(C = 1 \mid \mathbf{u}\bar{w}_j)}{P(\bar{w}_j \mid \mathbf{u}) + kq(w)}$$

Noise Contrastive Estimation

$$L = \log P(C = 0 \mid \mathbf{u}_w) + \sum_{j=1}^k \log P(C = 1 \mid \mathbf{u}_{\bar{w}_j})$$

$$= \log \frac{P(w \mid \mathbf{u})}{P(w \mid \mathbf{u}) + kq(w)} + \sum_{j=1}^k \log \frac{q(\bar{w}_j)}{P(\bar{w}_j \mid \mathbf{u}) + kq(w)}$$

Noise Contrastive Estimation

Gutmann and Hyvarinen, 2010, Mnih and Teh, 2012

For each training example (u, w) :

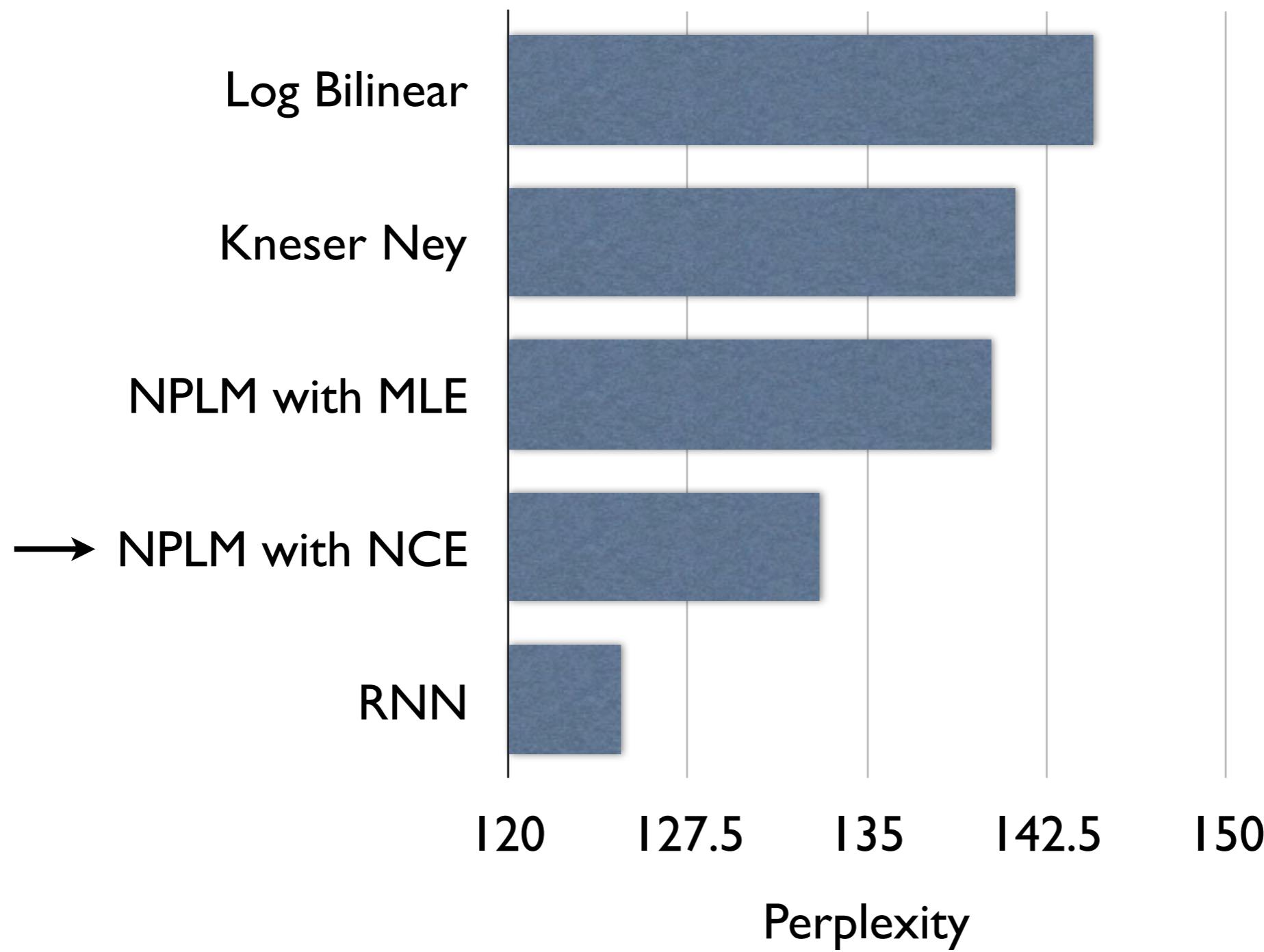
generate k noise samples

train model to classify real examples and noise samples

Advantages of NCE

- Much Faster training time.
- You can significantly speed up test time by encouraging the model to have a normalization constant of 1.

Better perplexity



NNLM on the Android Dataset

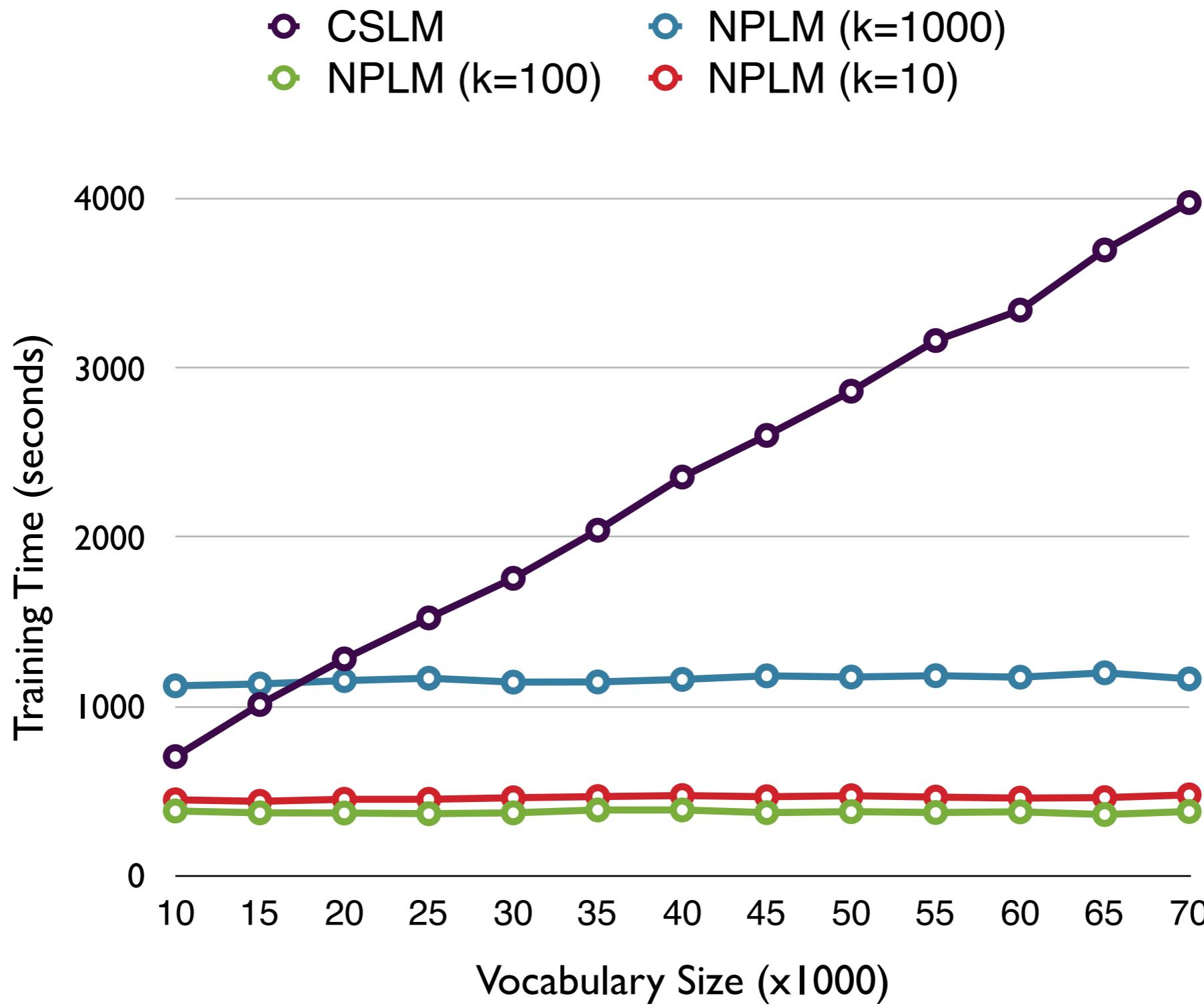
- Dataset of Android Apps
- 11 million tokens
- 90/10 split for Training set and Development
- 50k Vocabulary
- Cross entropy of 2.95 on Dev.

Results from Saheel Godane

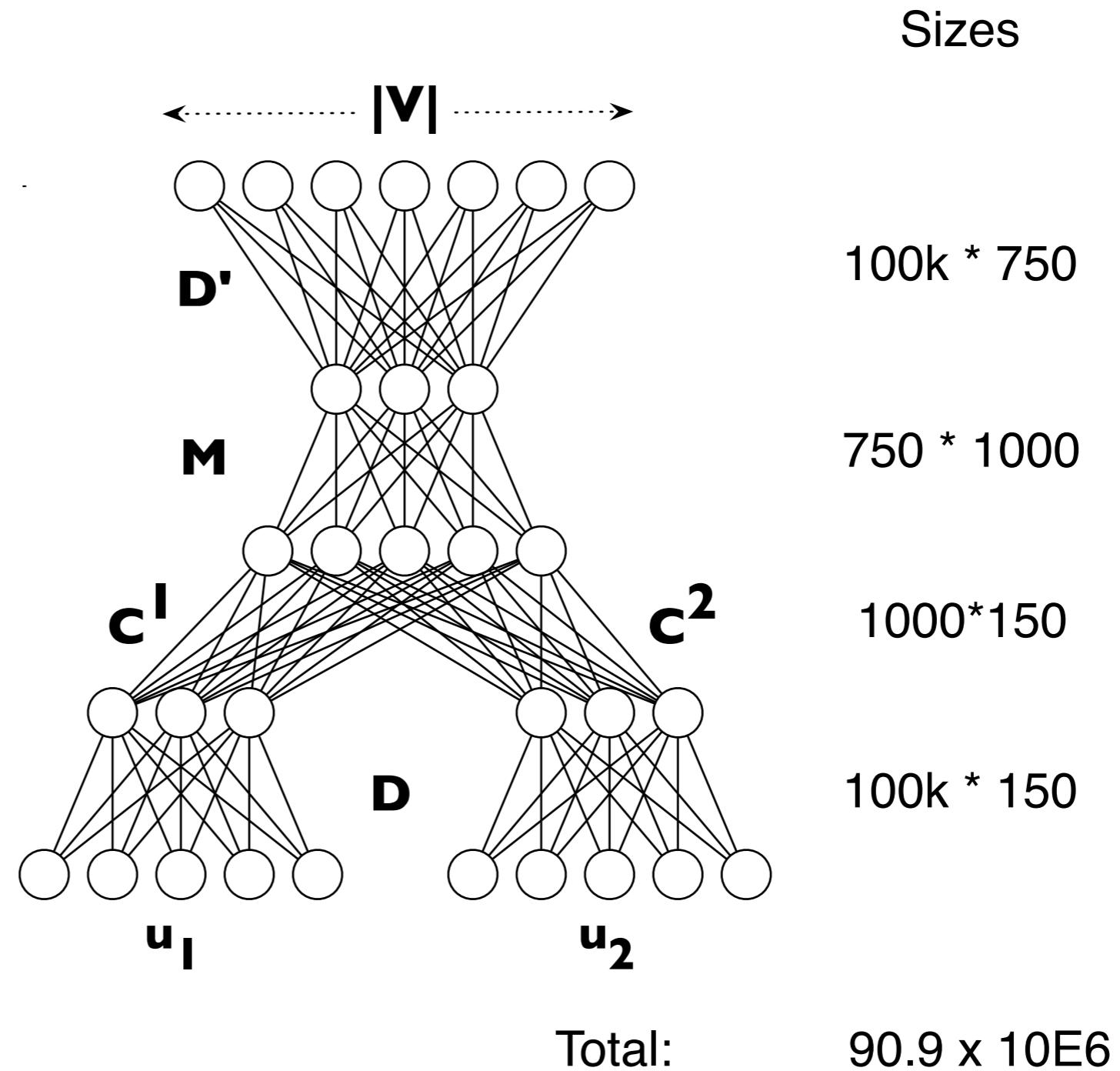
So Far...

	Standard n-gram	NPLM MLE	NPLM NCE
Model Size			
Training Time			
Query Time			

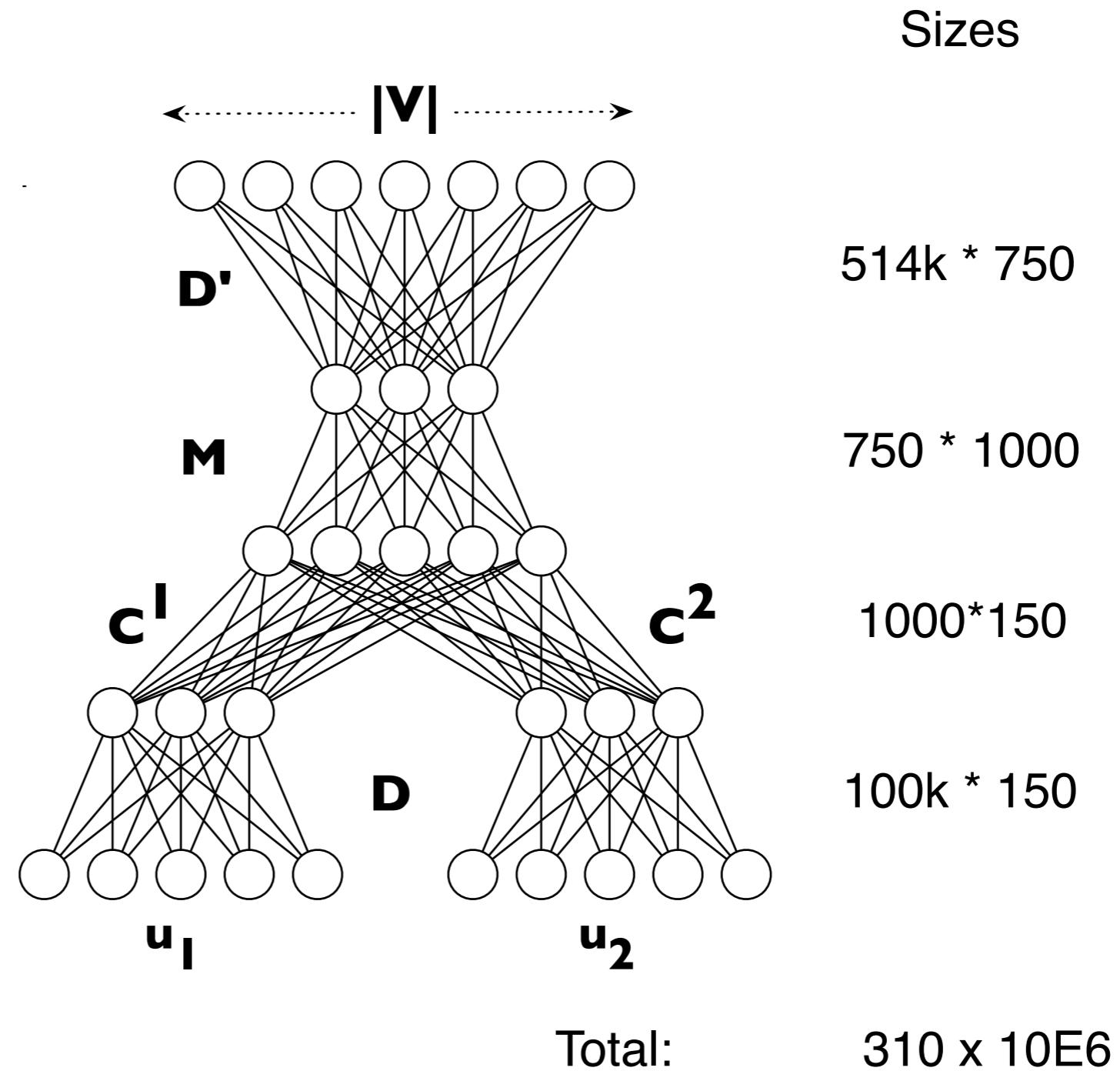
Faster training times



Neural Network Architecture



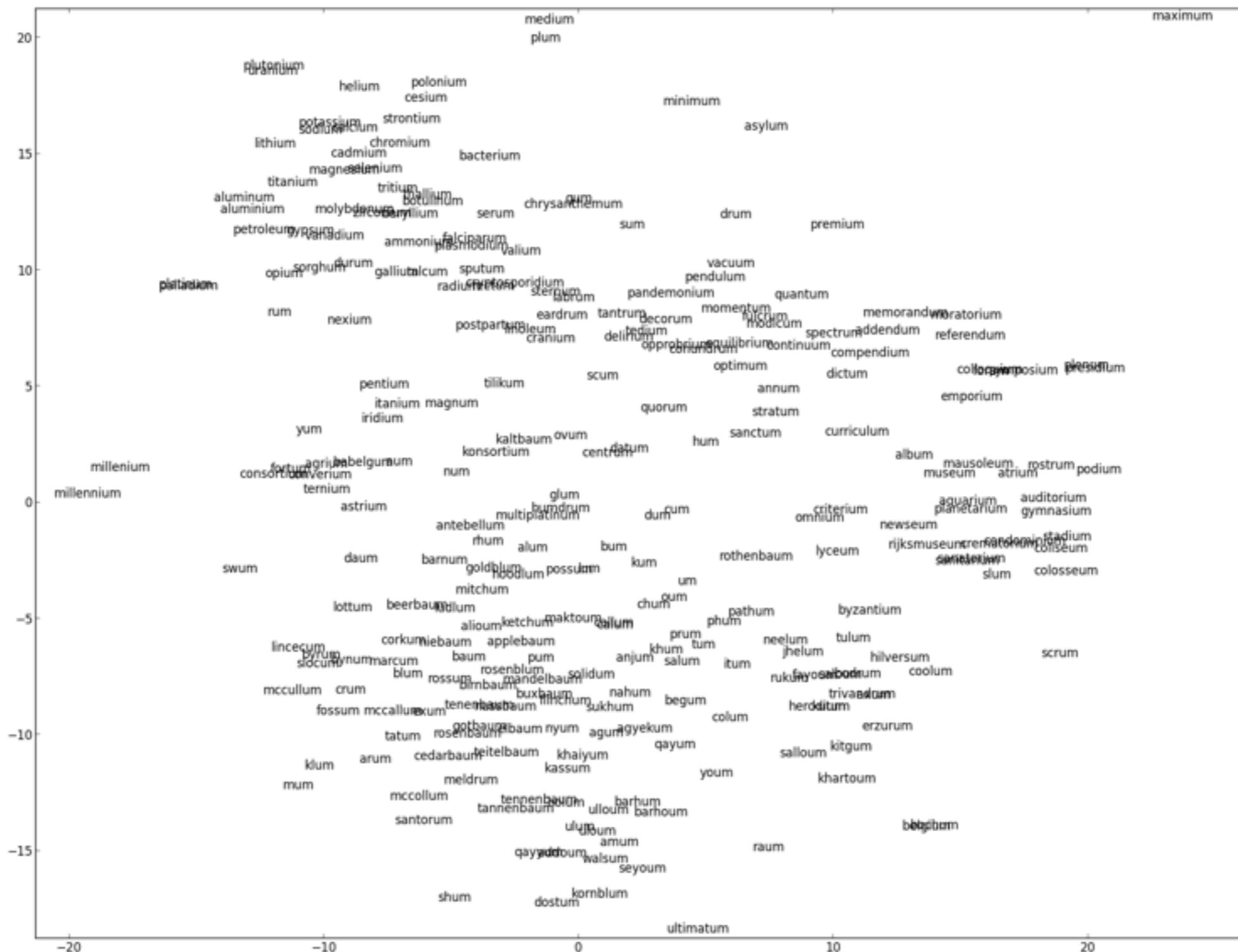
Neural Network Architecture



Nearest Neighbors

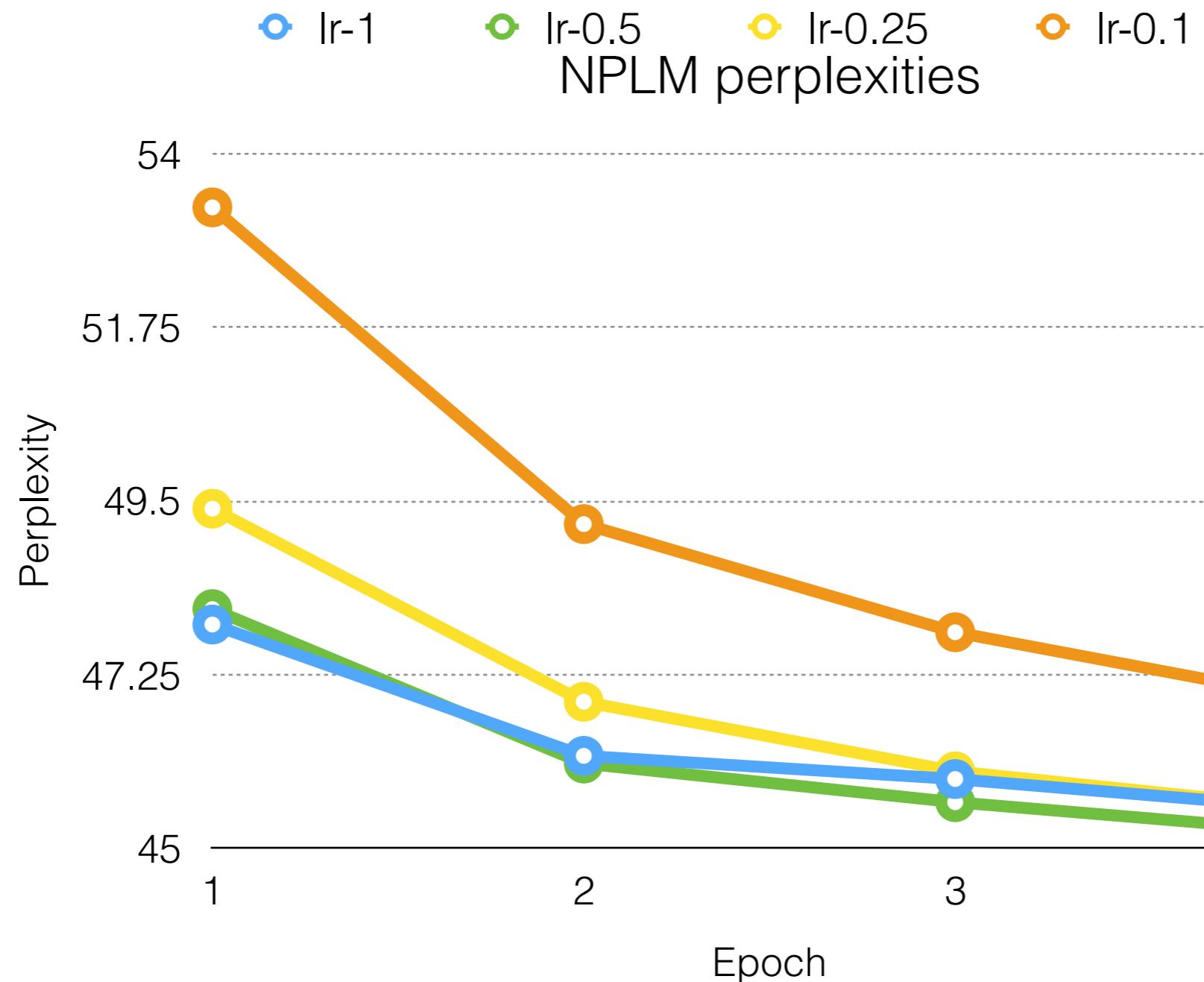
doctor	hospital	apple	bought
physician	medical	ibm	purchased
dentist	clinic	intel	sold
pharmacist	hospitals	seagate	procured
psychologist	hospice	hp	scooped
neurosurgeon	mortuary	netflix	cashed
veterinarian	sanatorium	kmart	reaped
physiotherapist	orphanage	tivo	fetched

2-dim projection



Slide from David Chiang

Perplexity is robust to learning rate

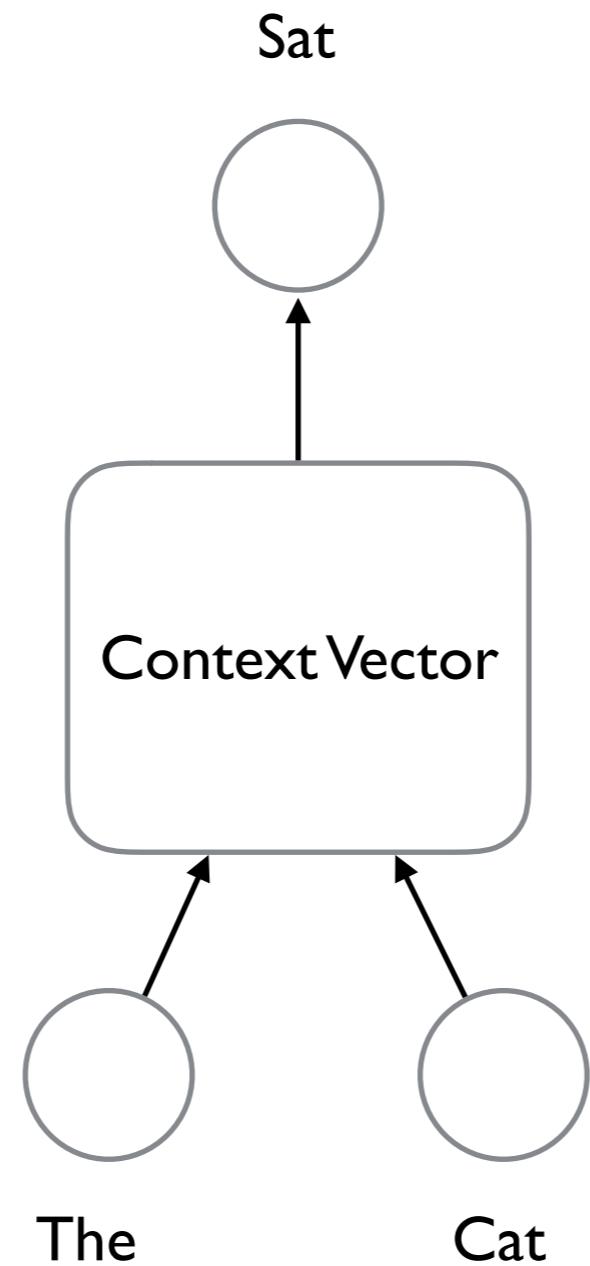


Other approaches

- Class based softmax
- Hierarchical Softmax
- Automatically discovering the right size of the network
(Murray and Chiang, 2015)
- NCE has been used in modeling Code and Language
(Allamanis et al., 2015)

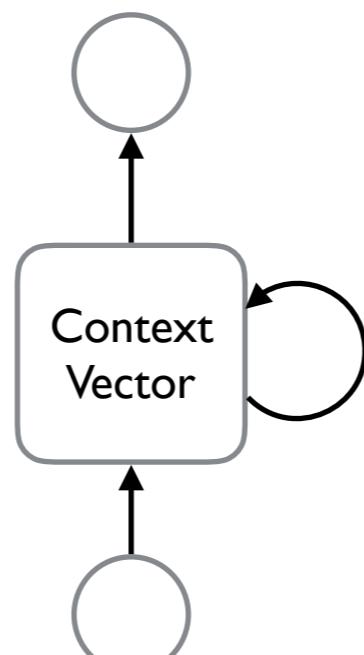
Recurrent Neural Network Language Models

Feed Forward NNLM



Recurrent NNLMs

output

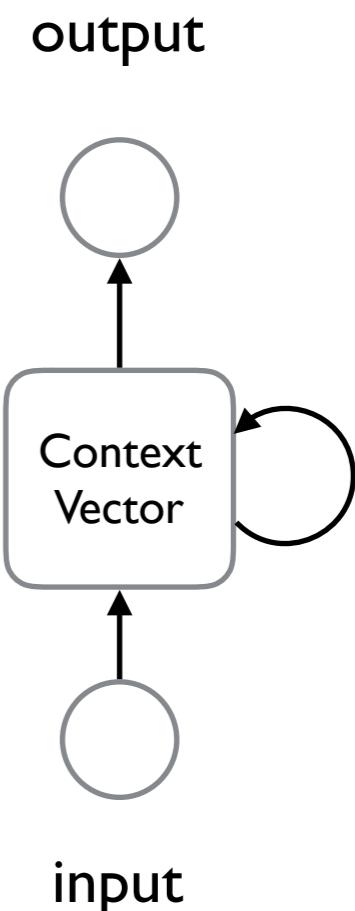


input

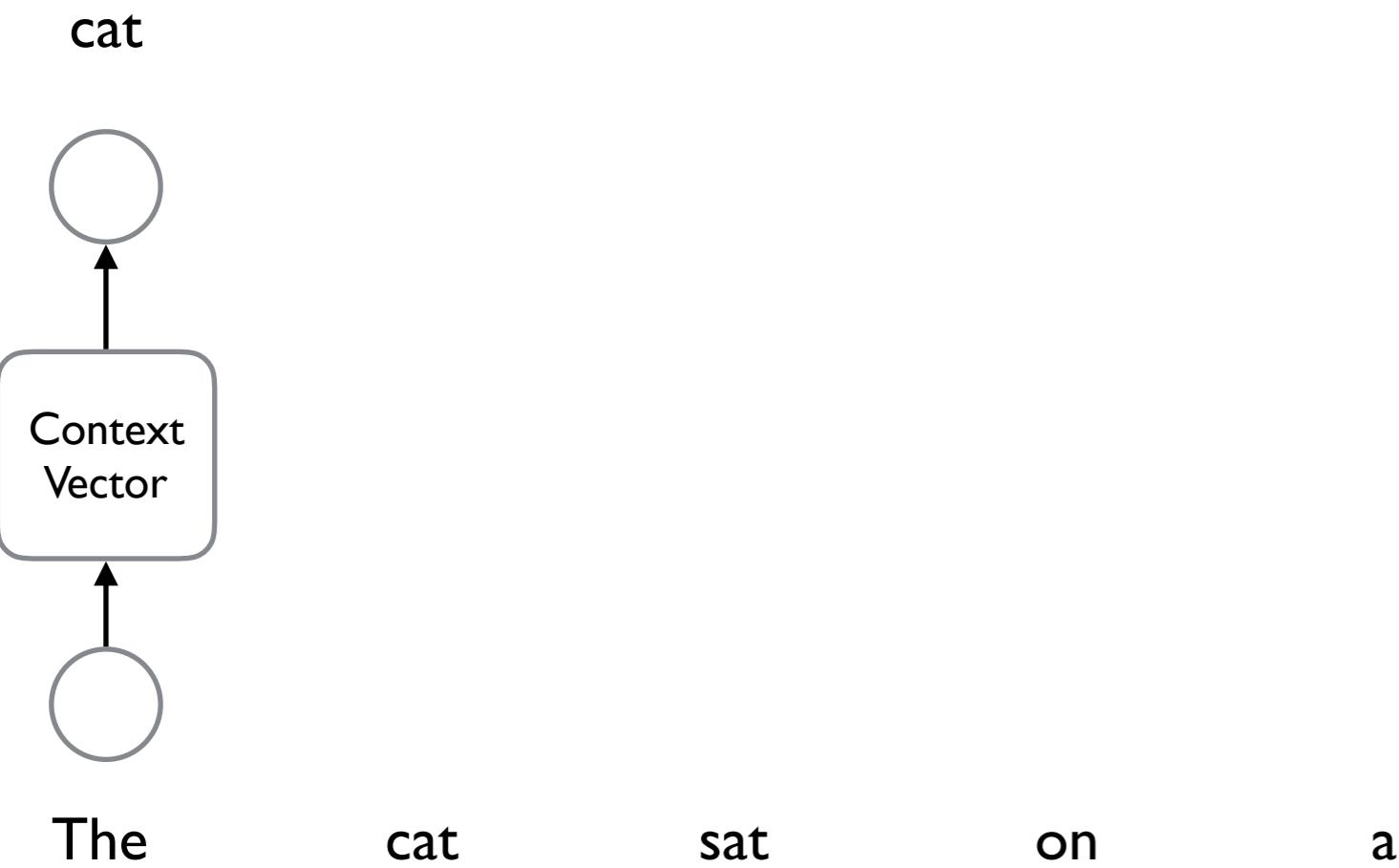
Recurrent NNLMs

The cat sat on a mat

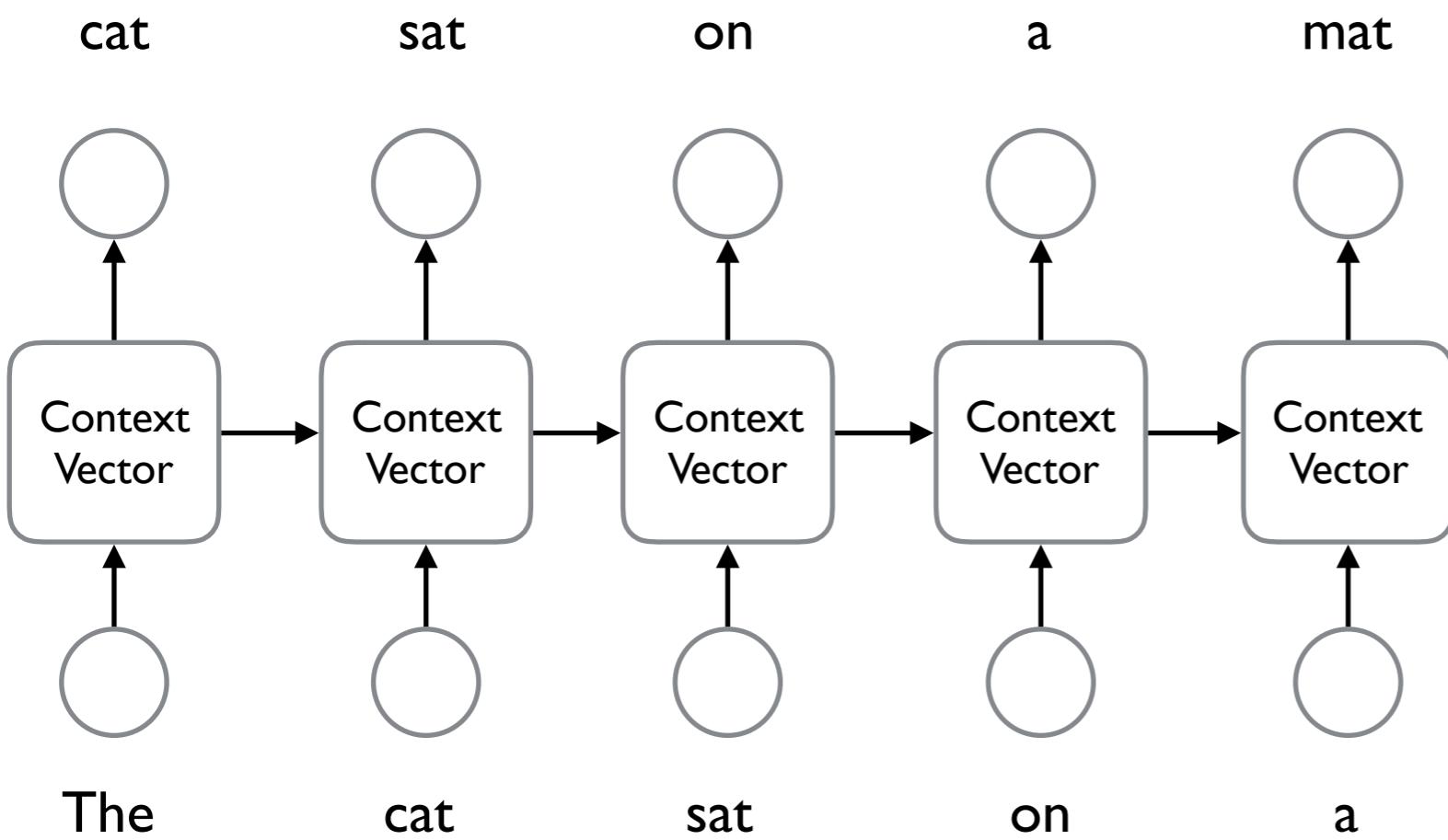
∩



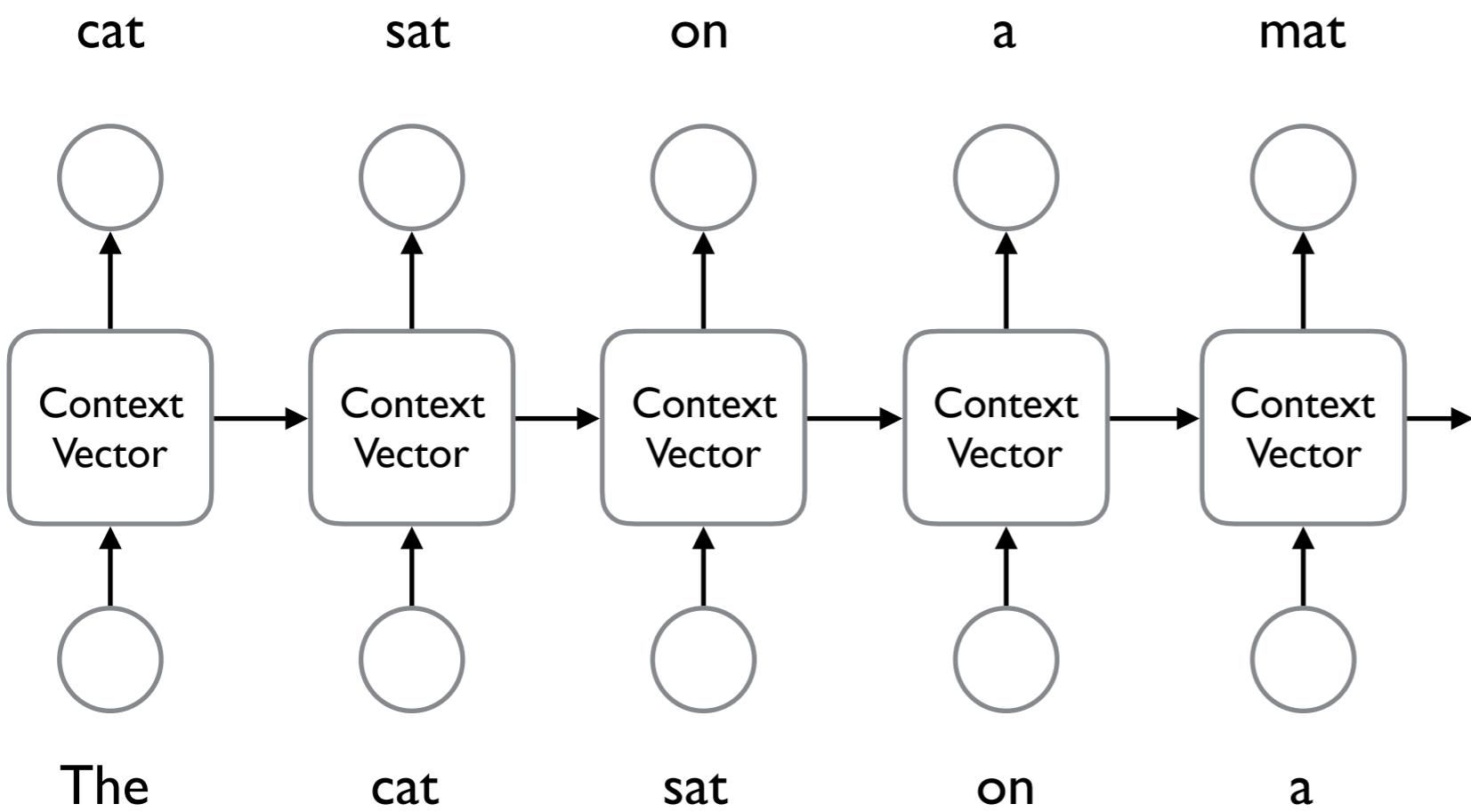
Recurrent NNLMs



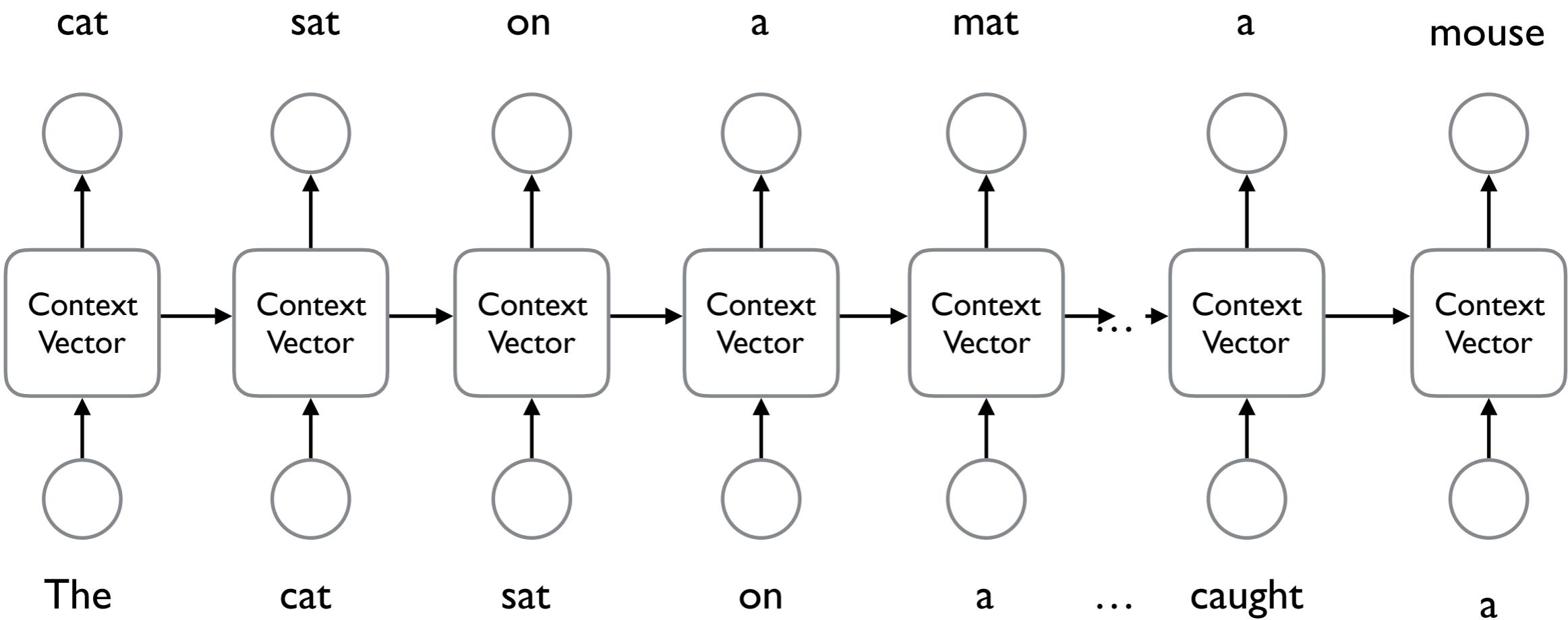
Recurrent NNLMs



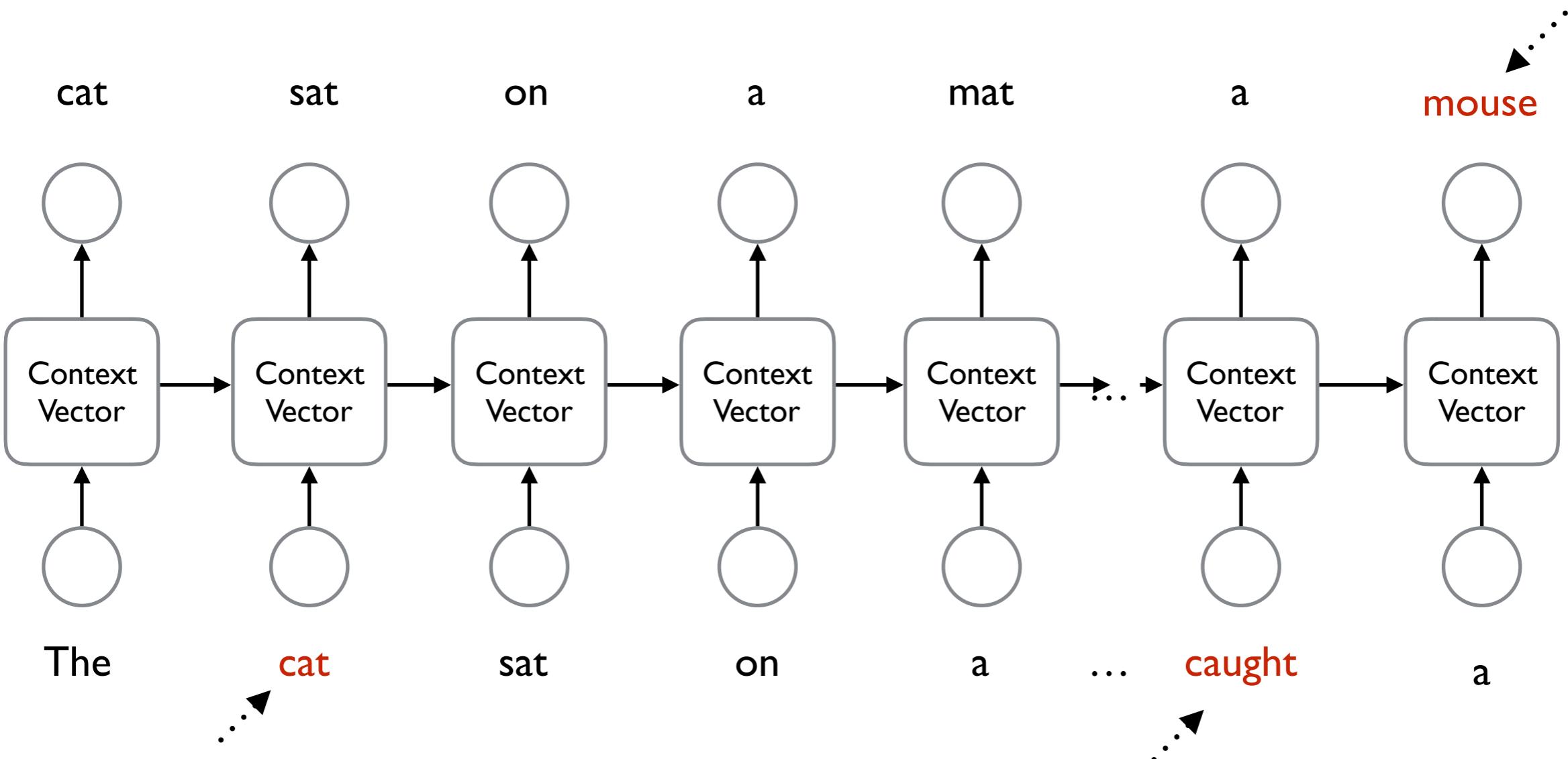
Recurrent NNLMs



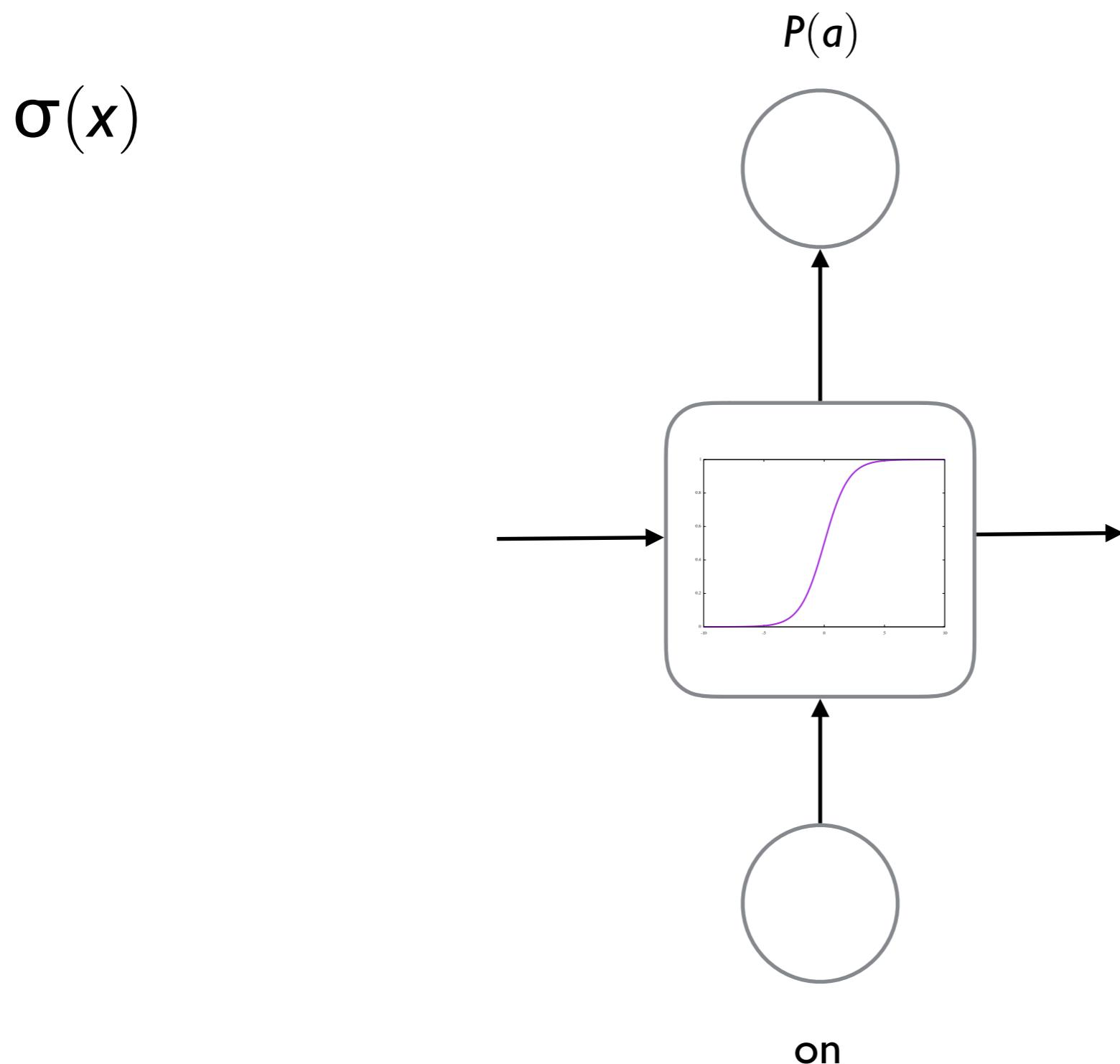
Recurrent NNLMs



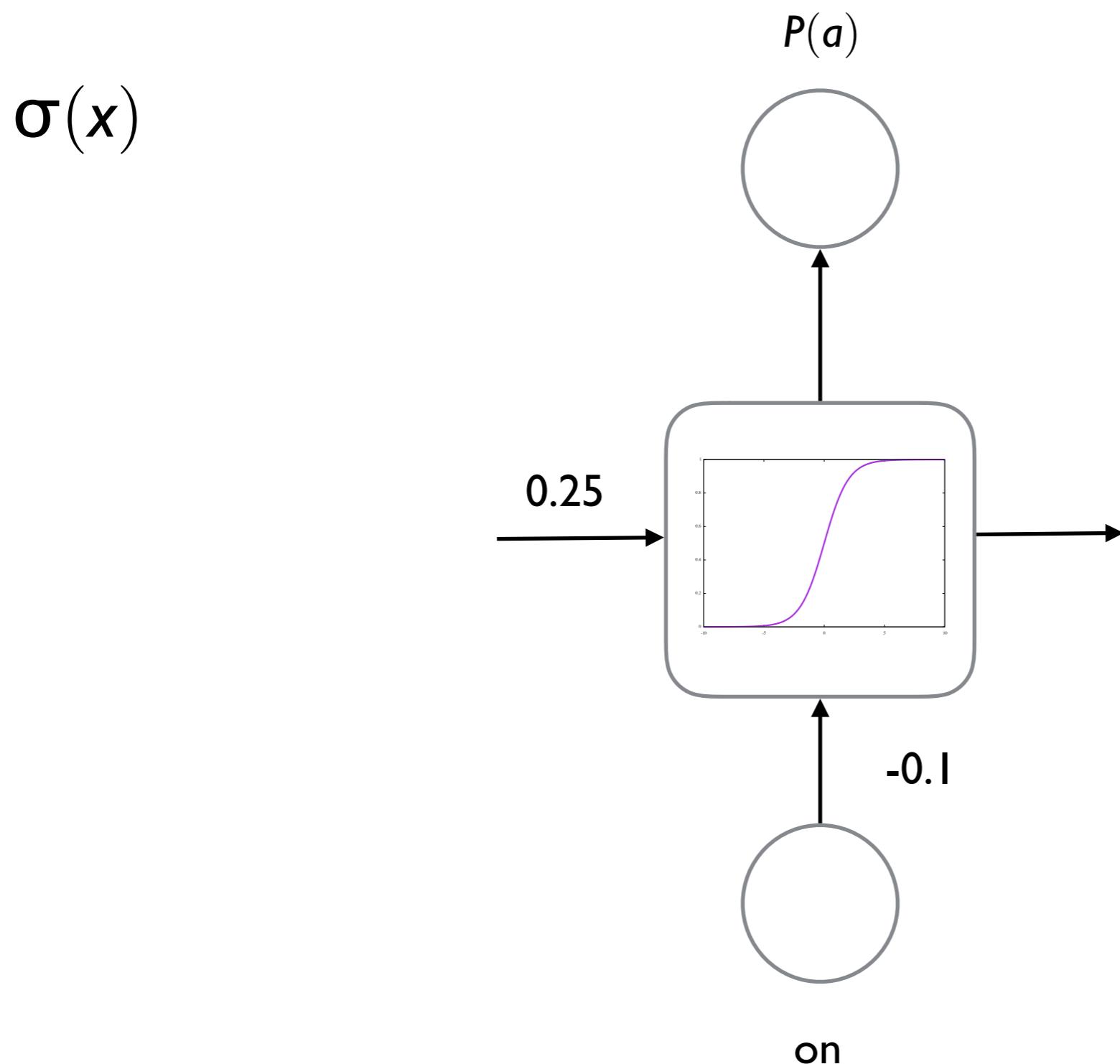
Recurrent NNLMs



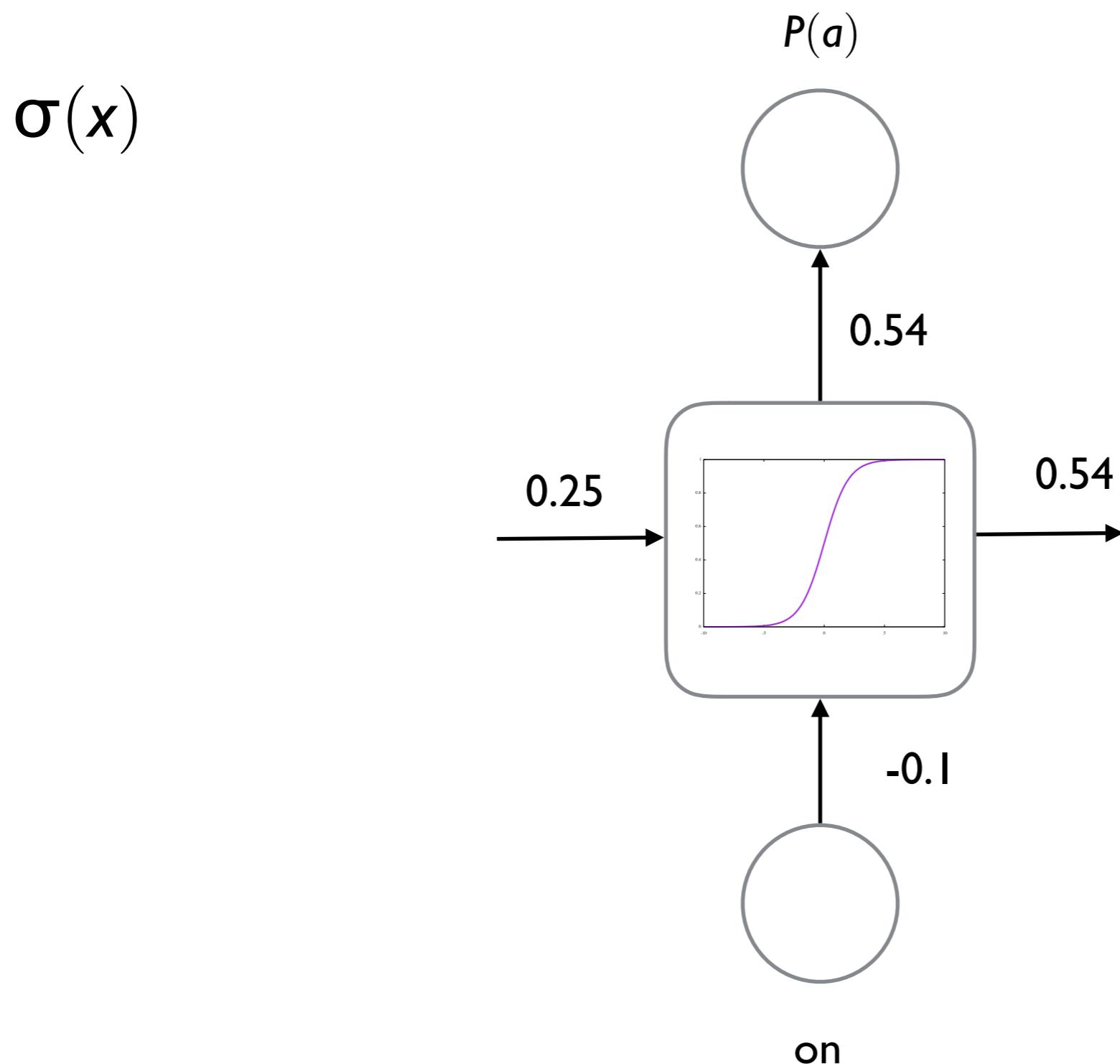
Recurrent NNLMs



Recurrent NNLMs



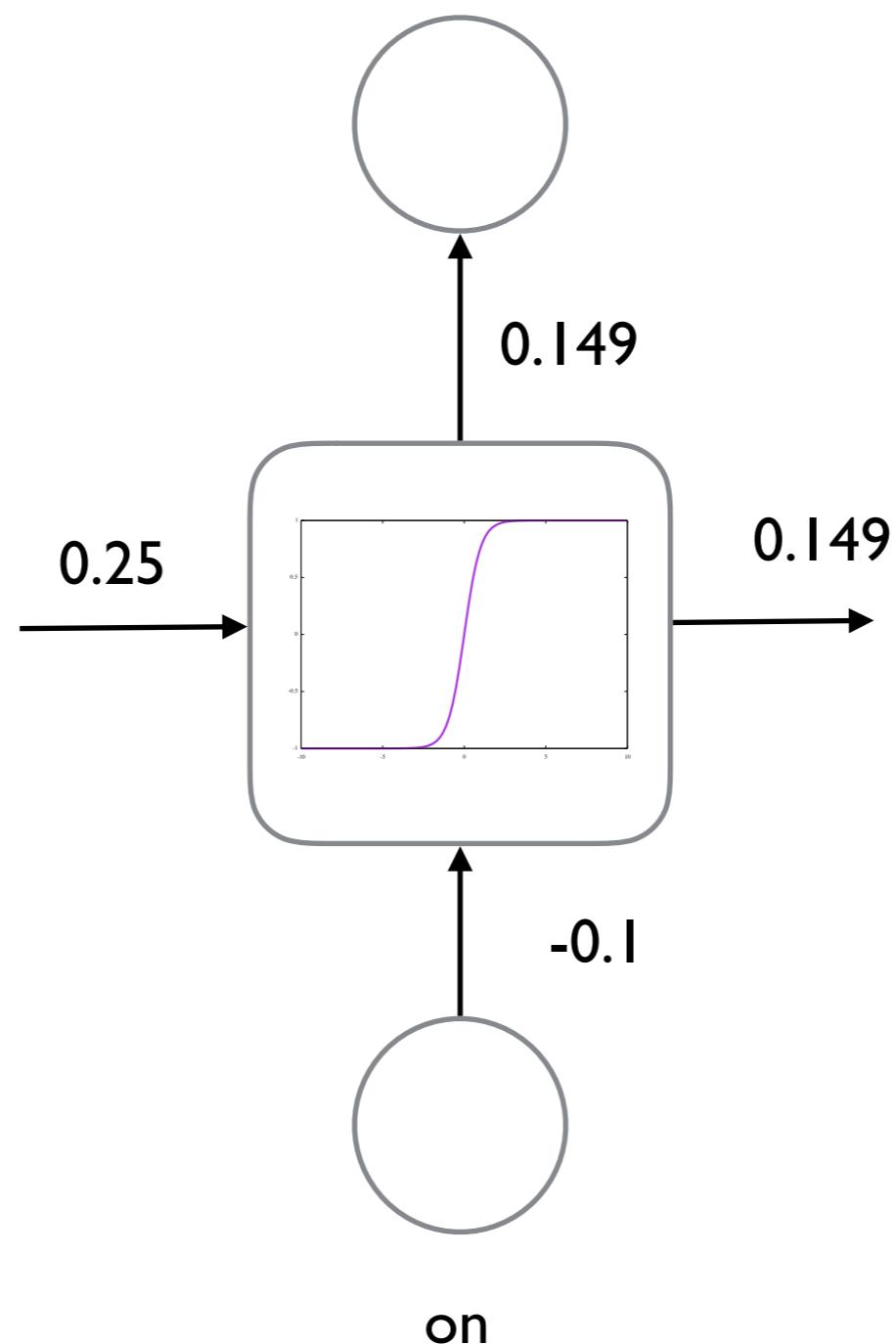
Recurrent NNLMs



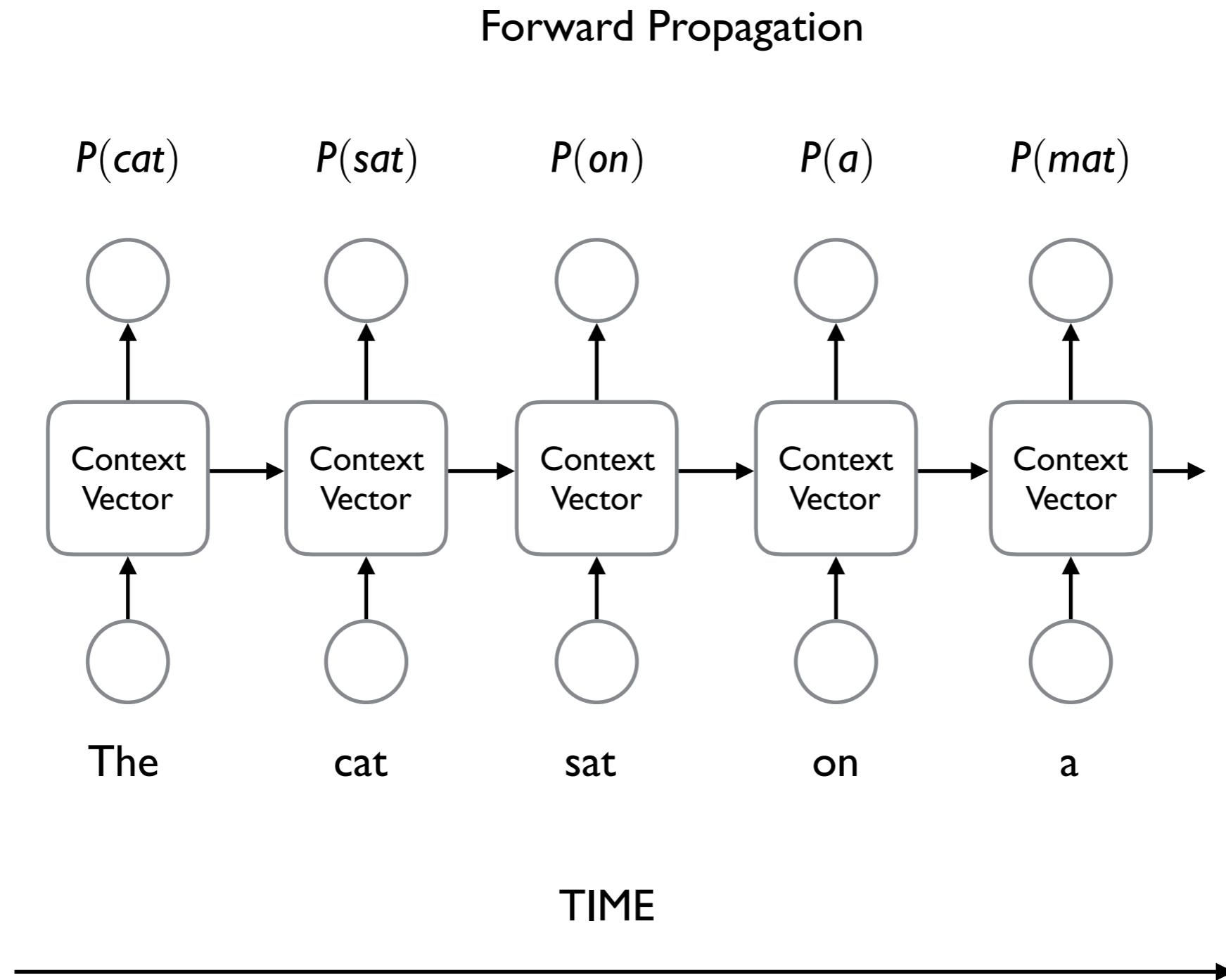
Recurrent NNLMs

$$P(a|\text{context}) = 0.9$$

$\tanh(x)$

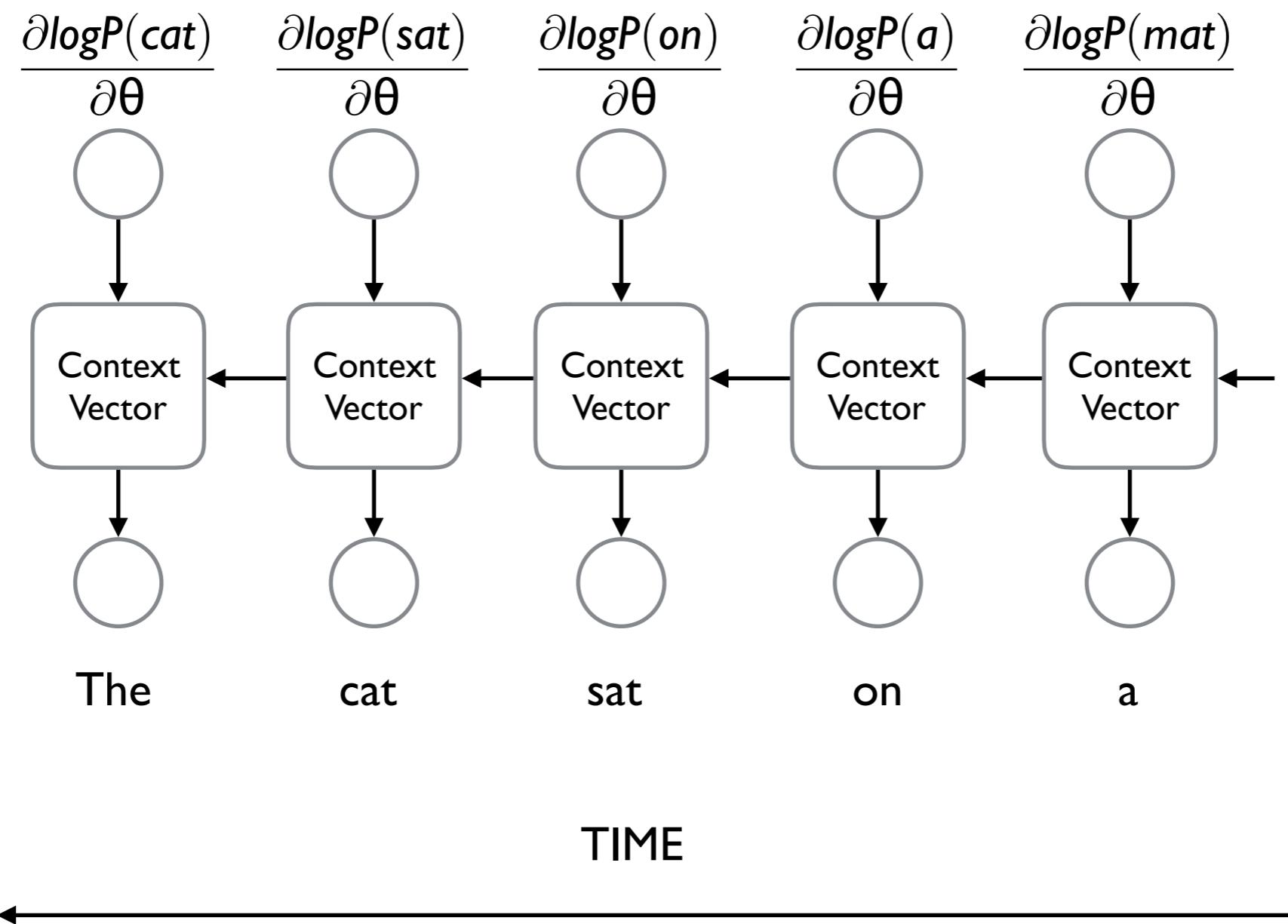


Training Recurrent NNLMs



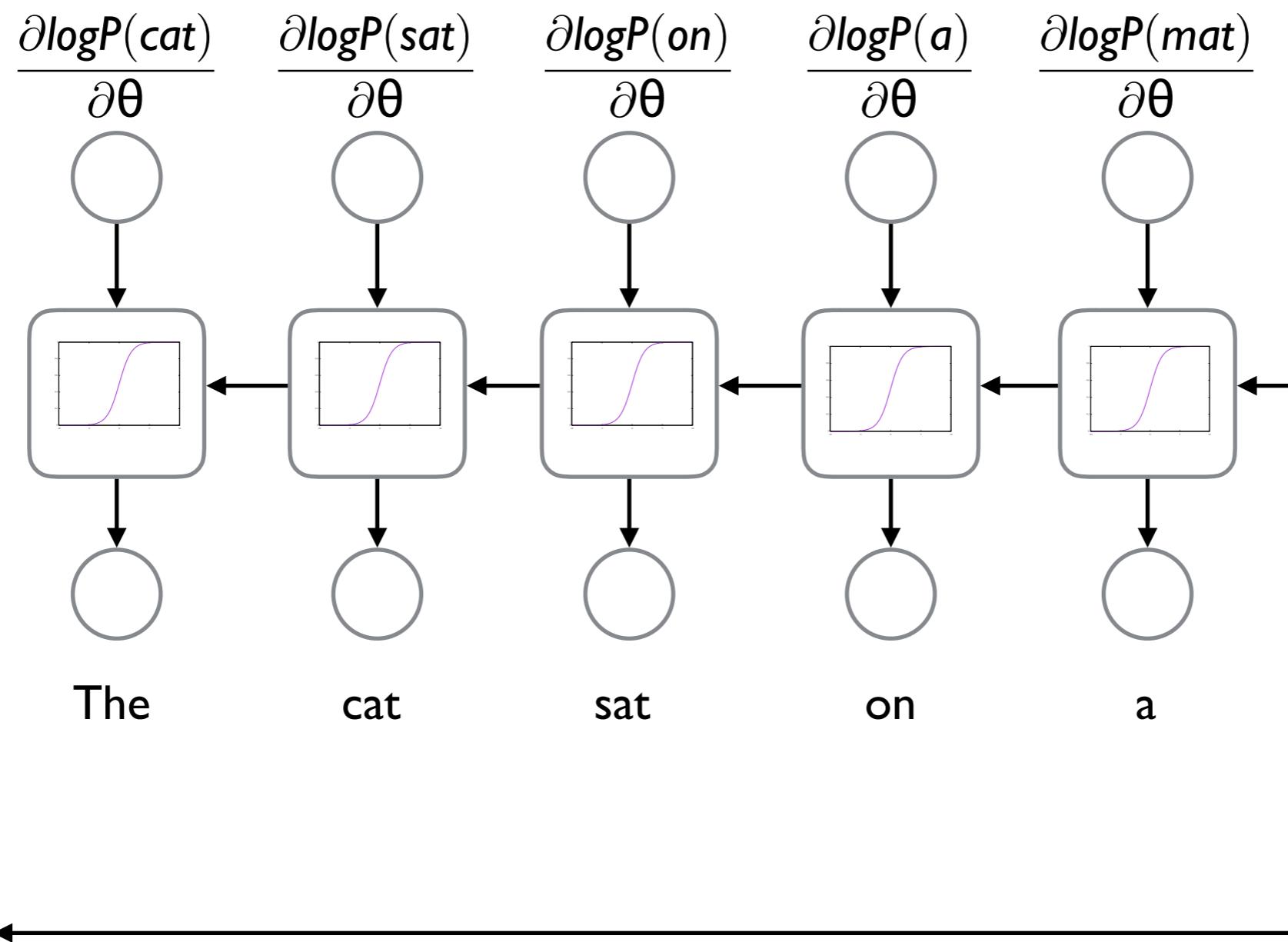
Training Recurrent NNLMs

Back Propagation through time



Vanishing gradients

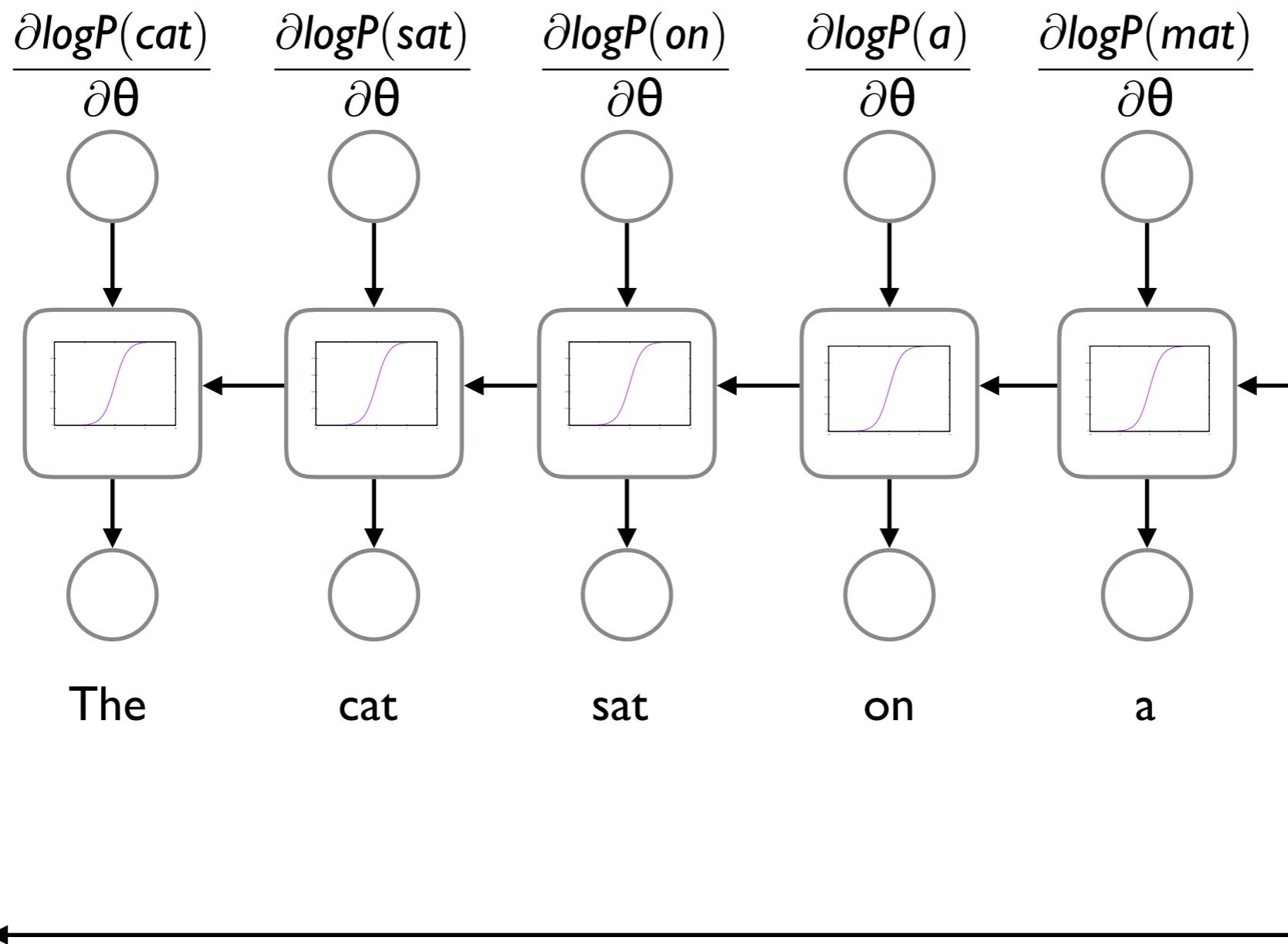
Back Propagation through time



Vanishing gradients

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

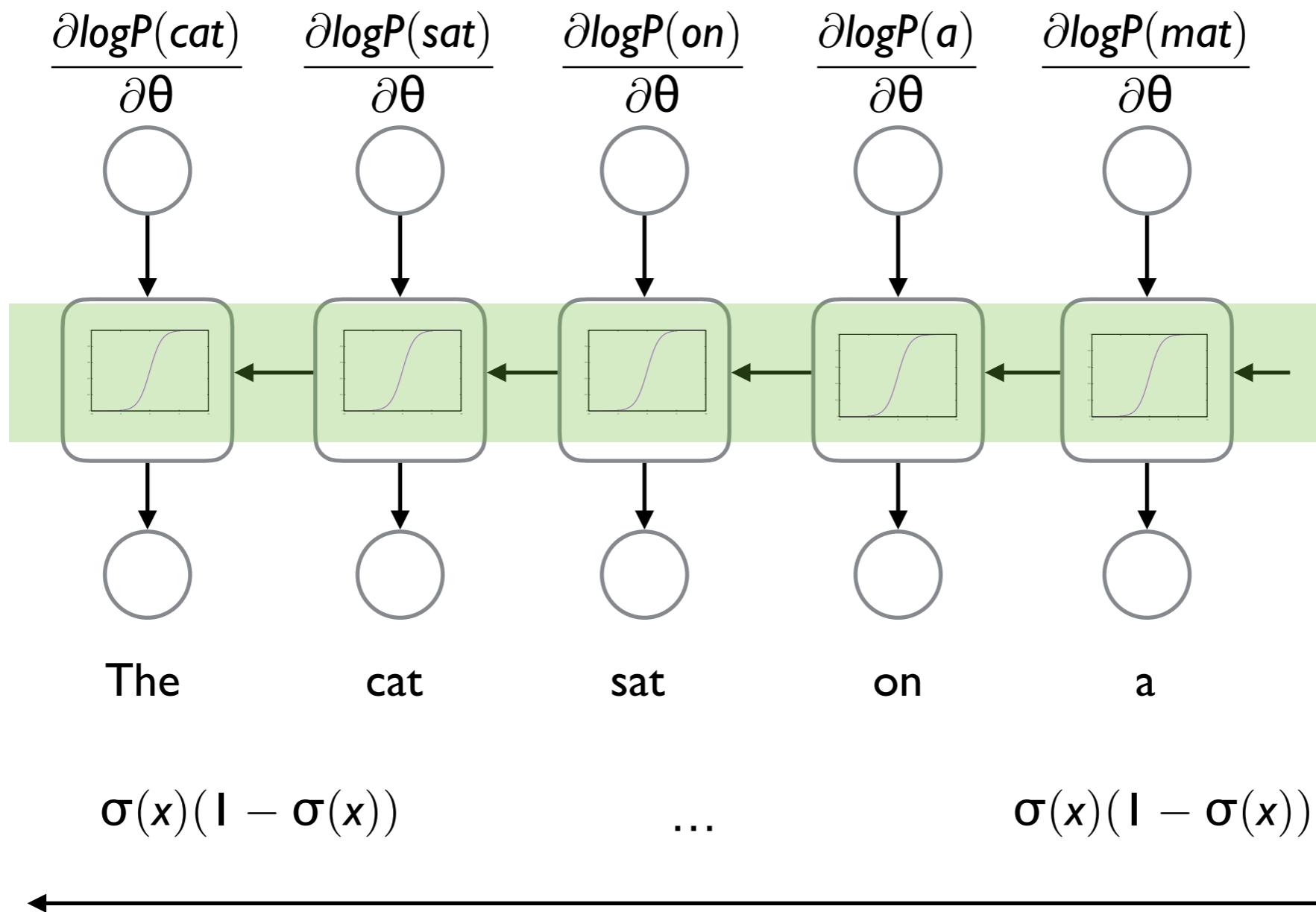
Back Propagation through time



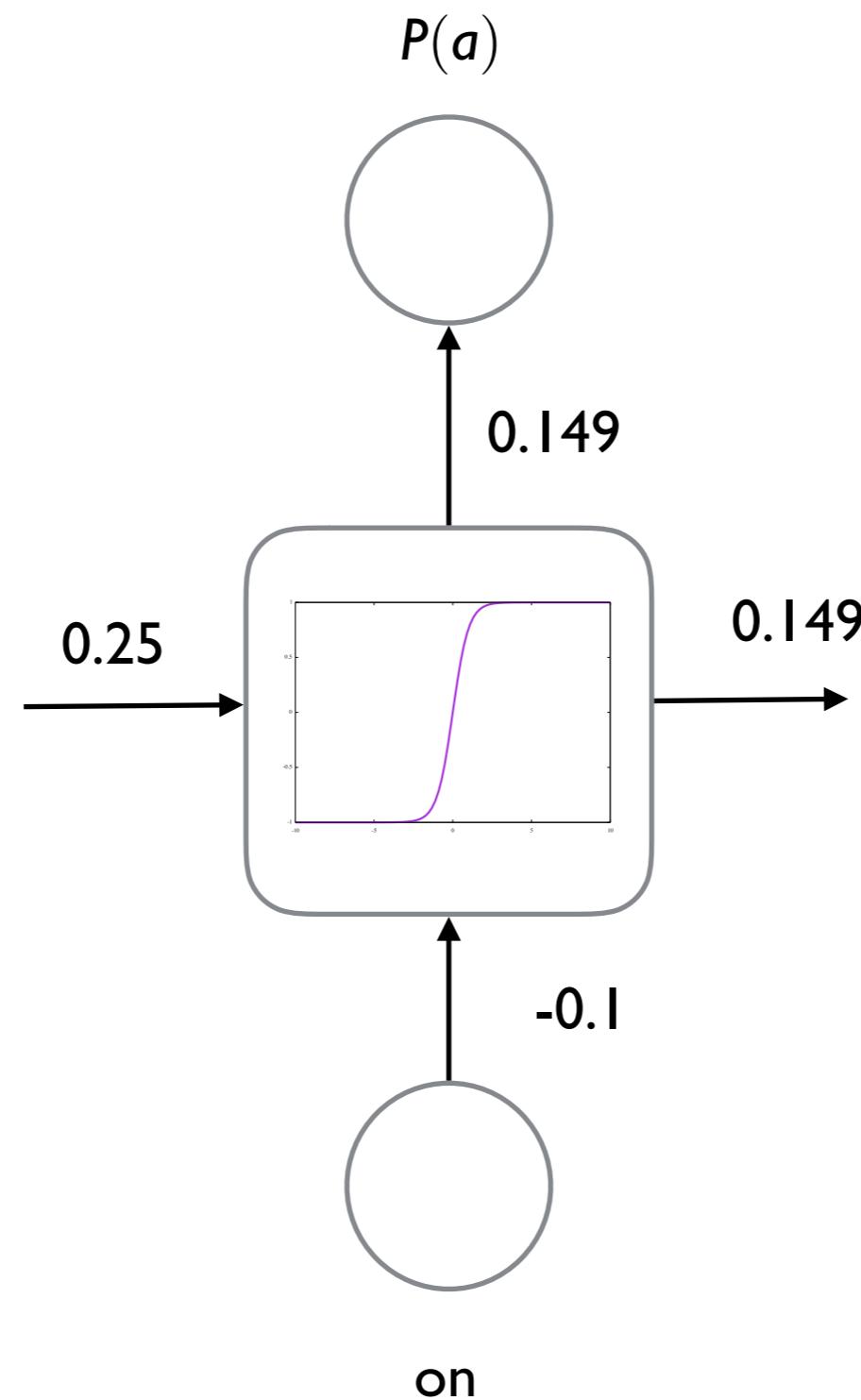
Vanishing gradients

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

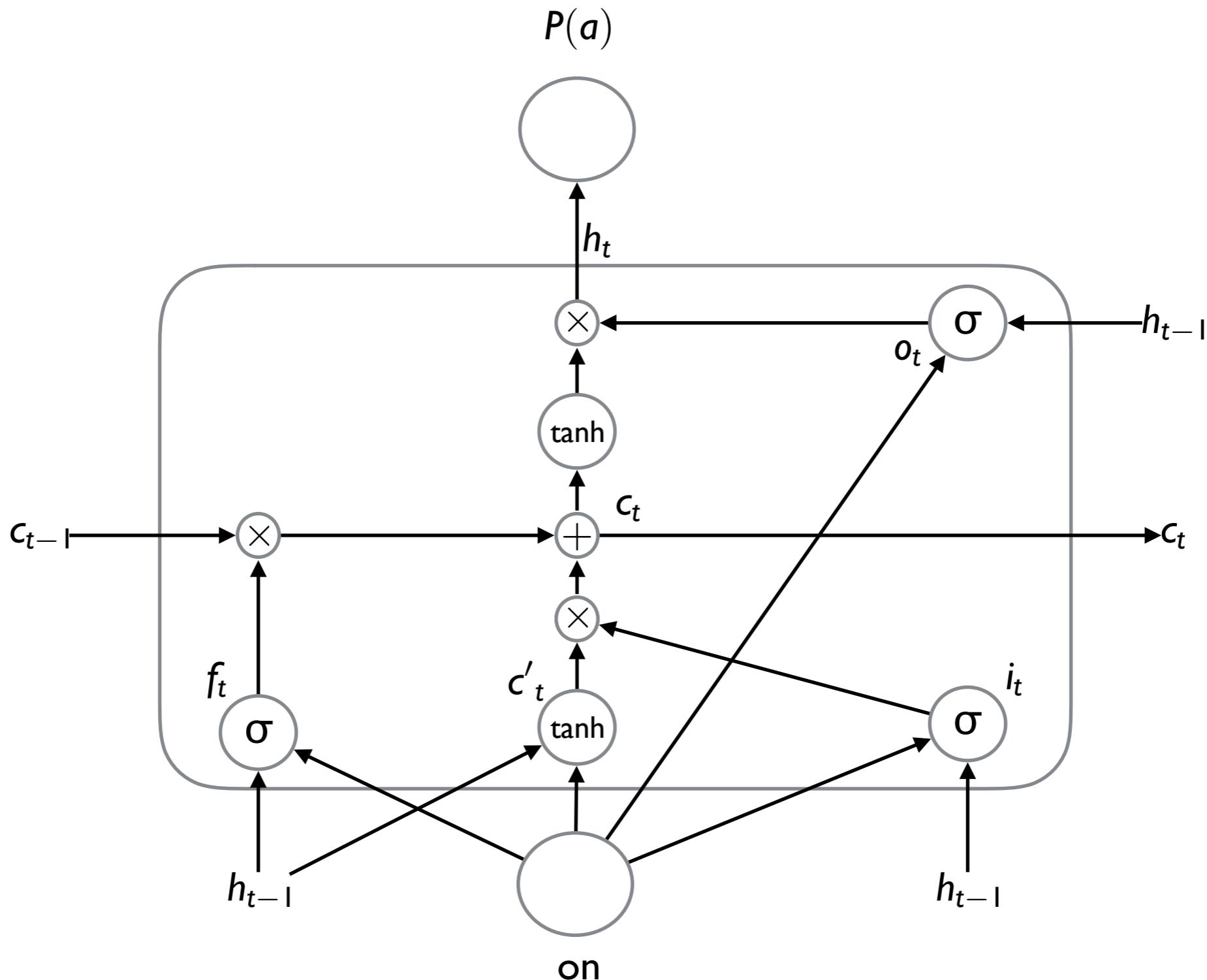
Back Propagation through time



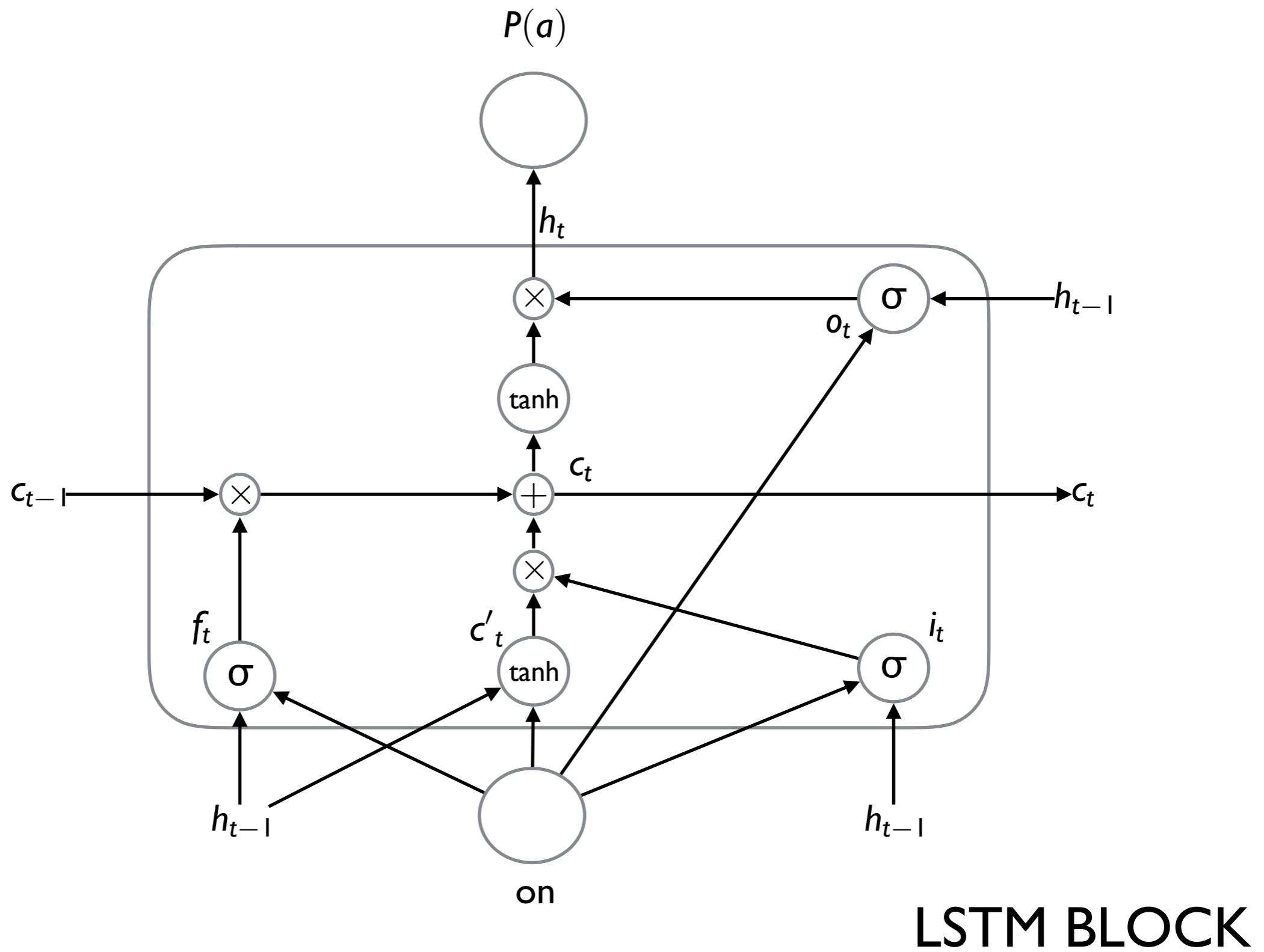
Solution: Long Short Term Memory



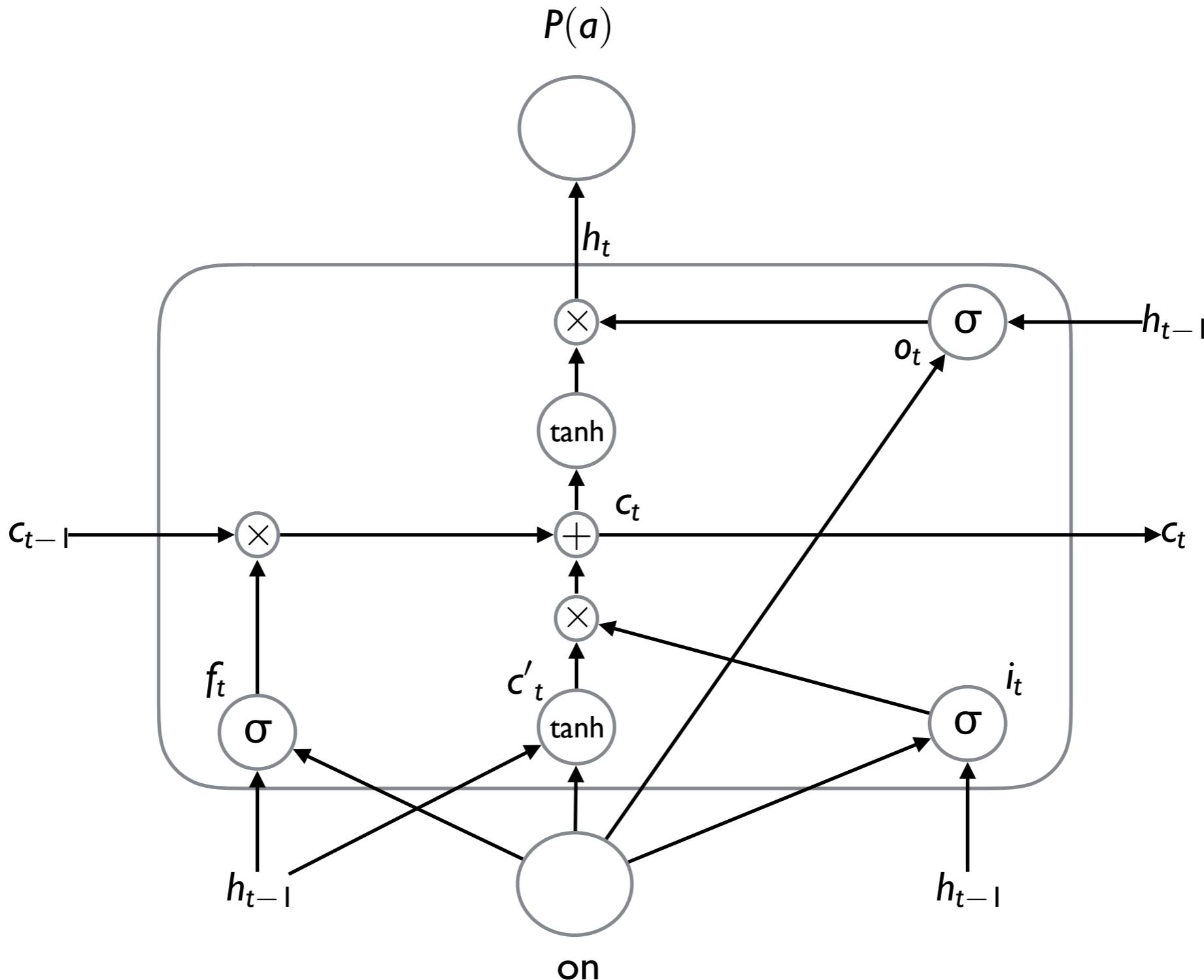
Solution: Long Short Term Memory



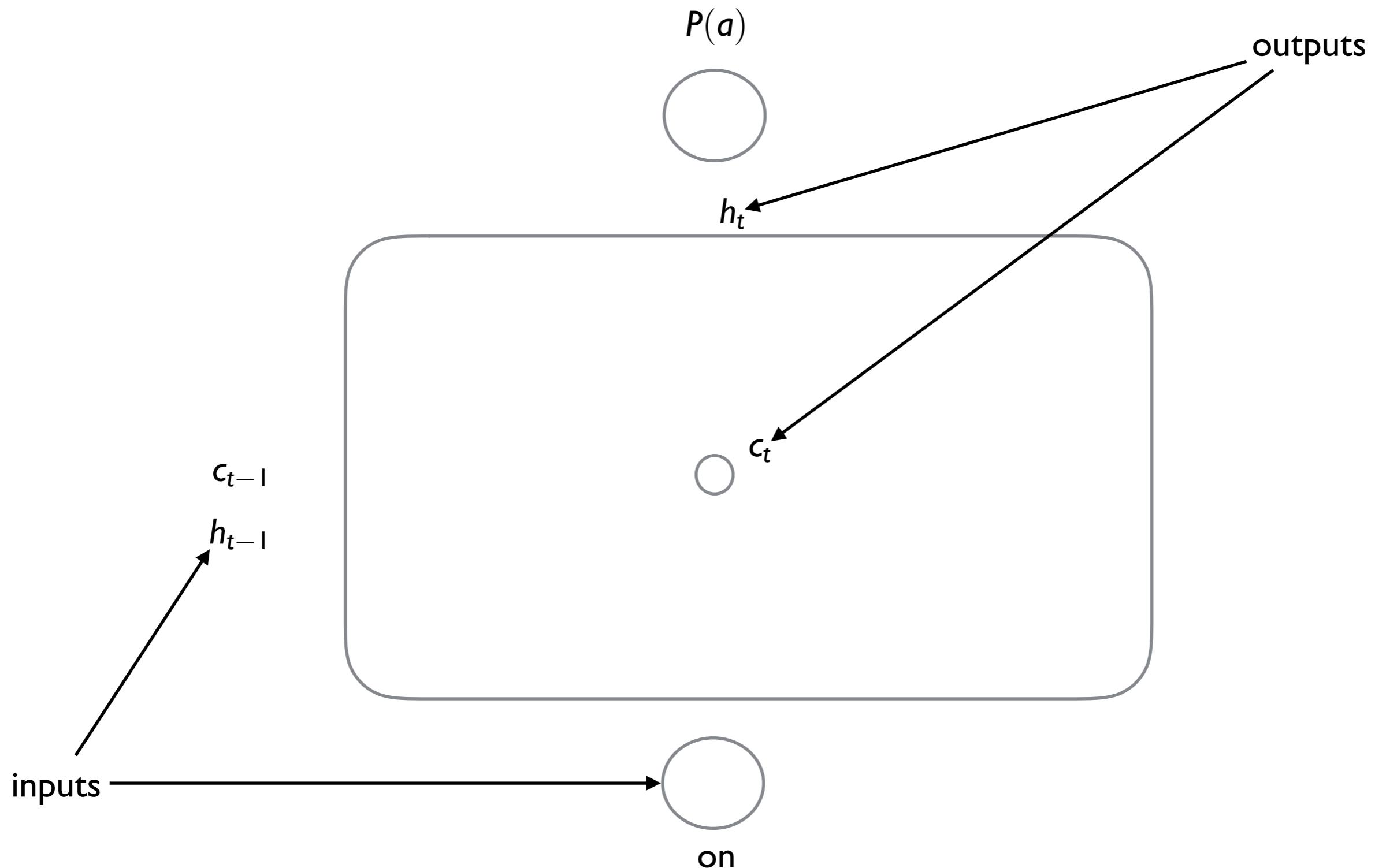
Solution: Long Short Term Memory



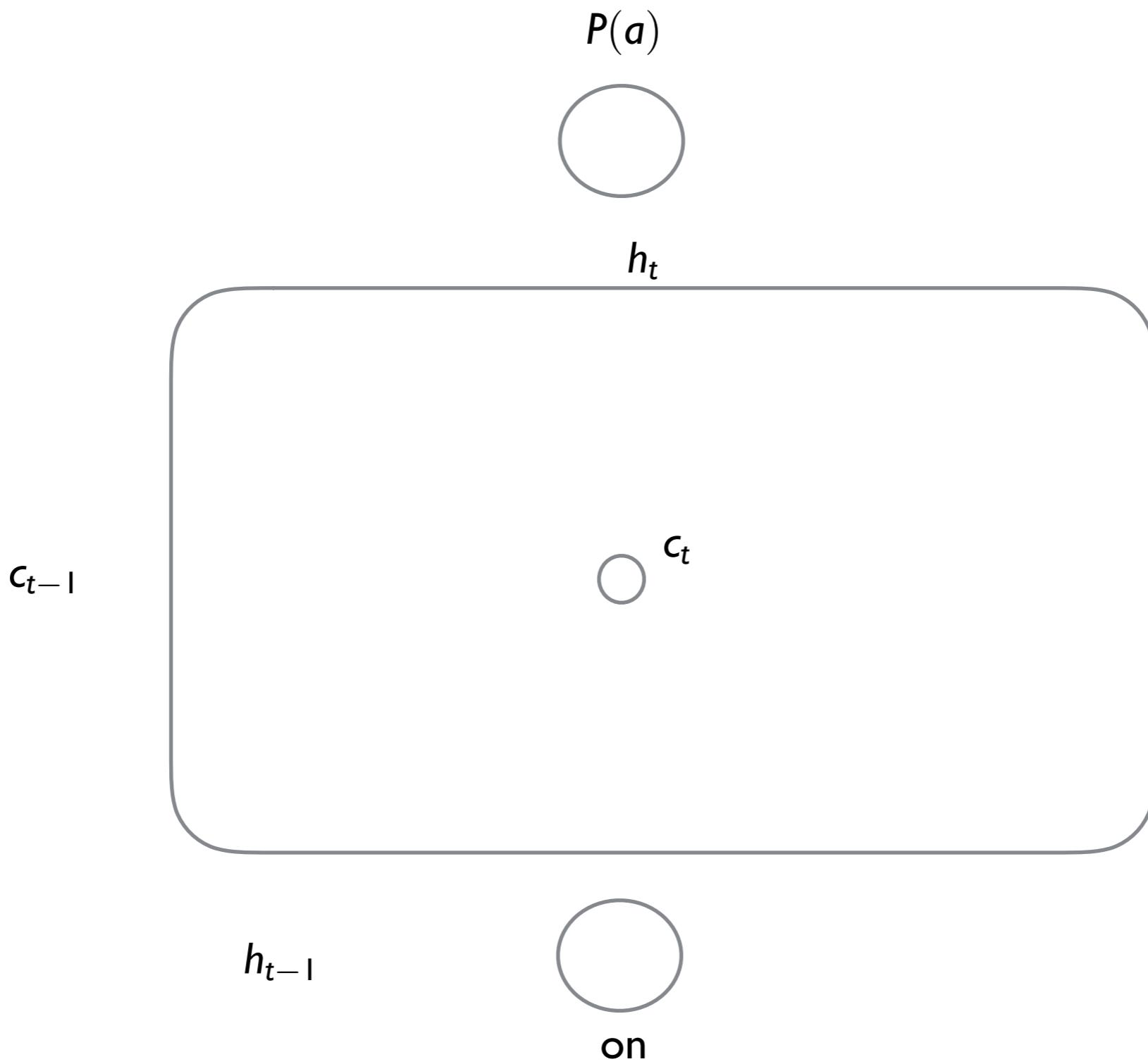
Forward Propagation in LSTM Block



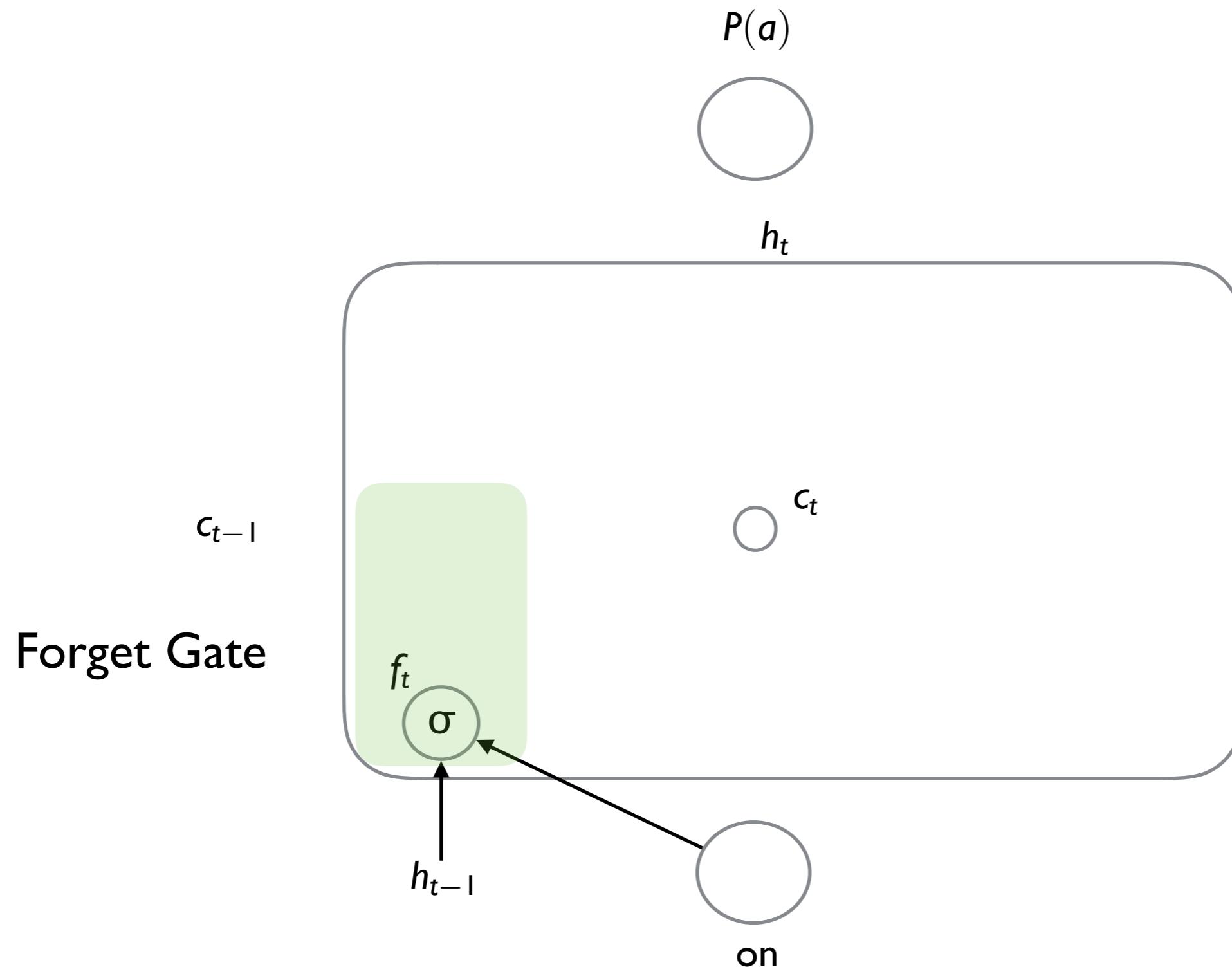
Forward Propagation in LSTM Block



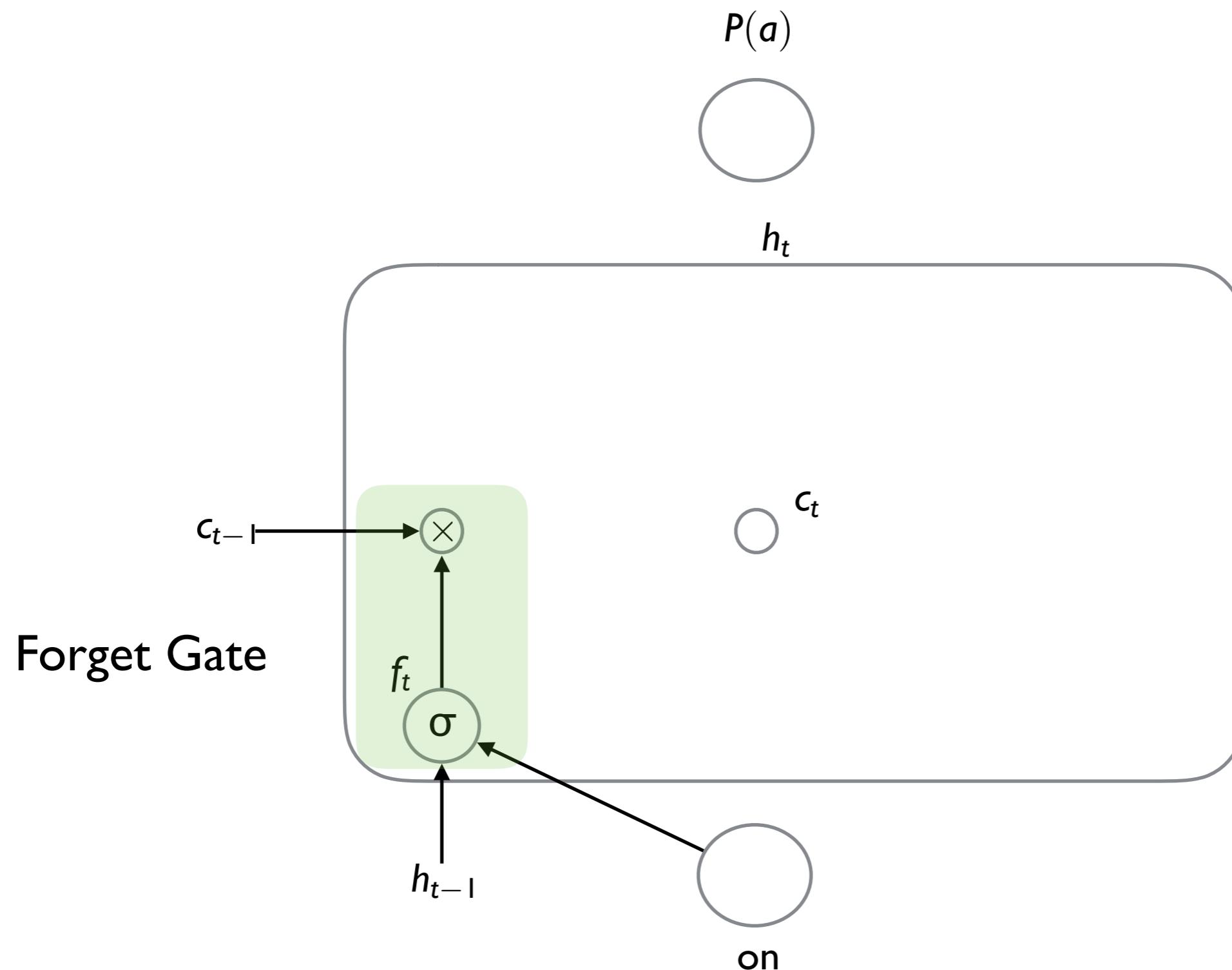
I. Forgetting memory



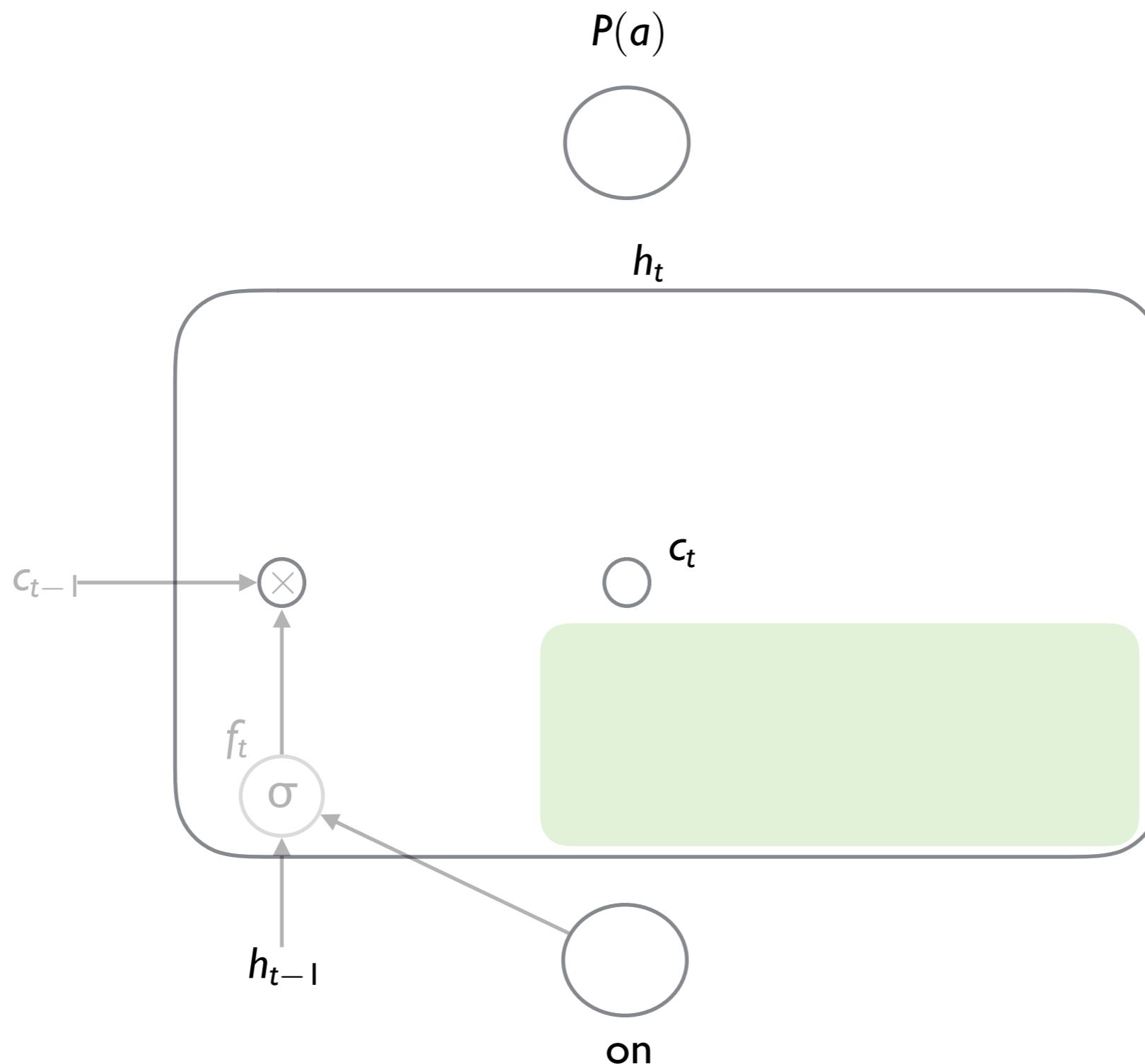
I. Forgetting memory



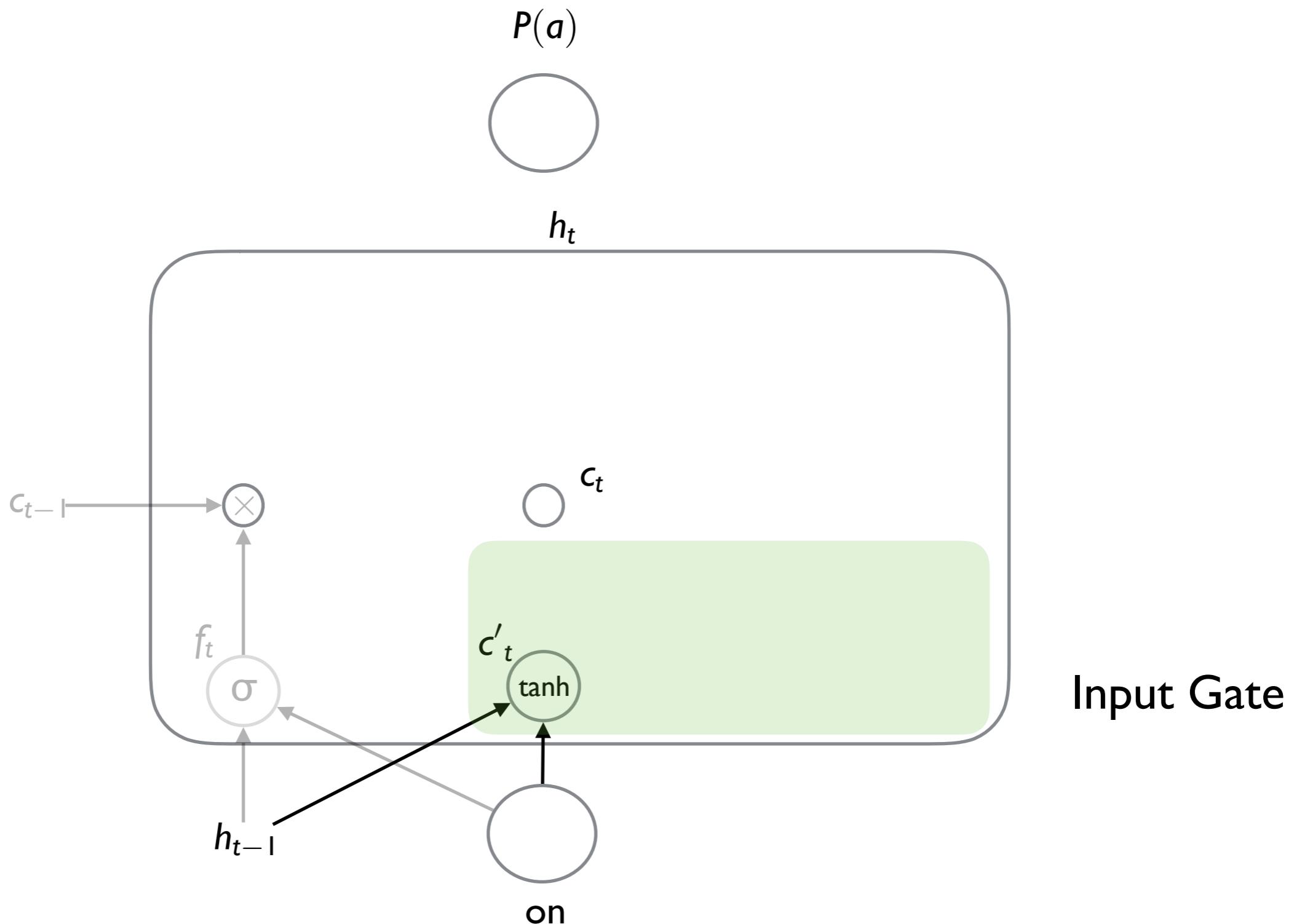
I. Forgetting memory



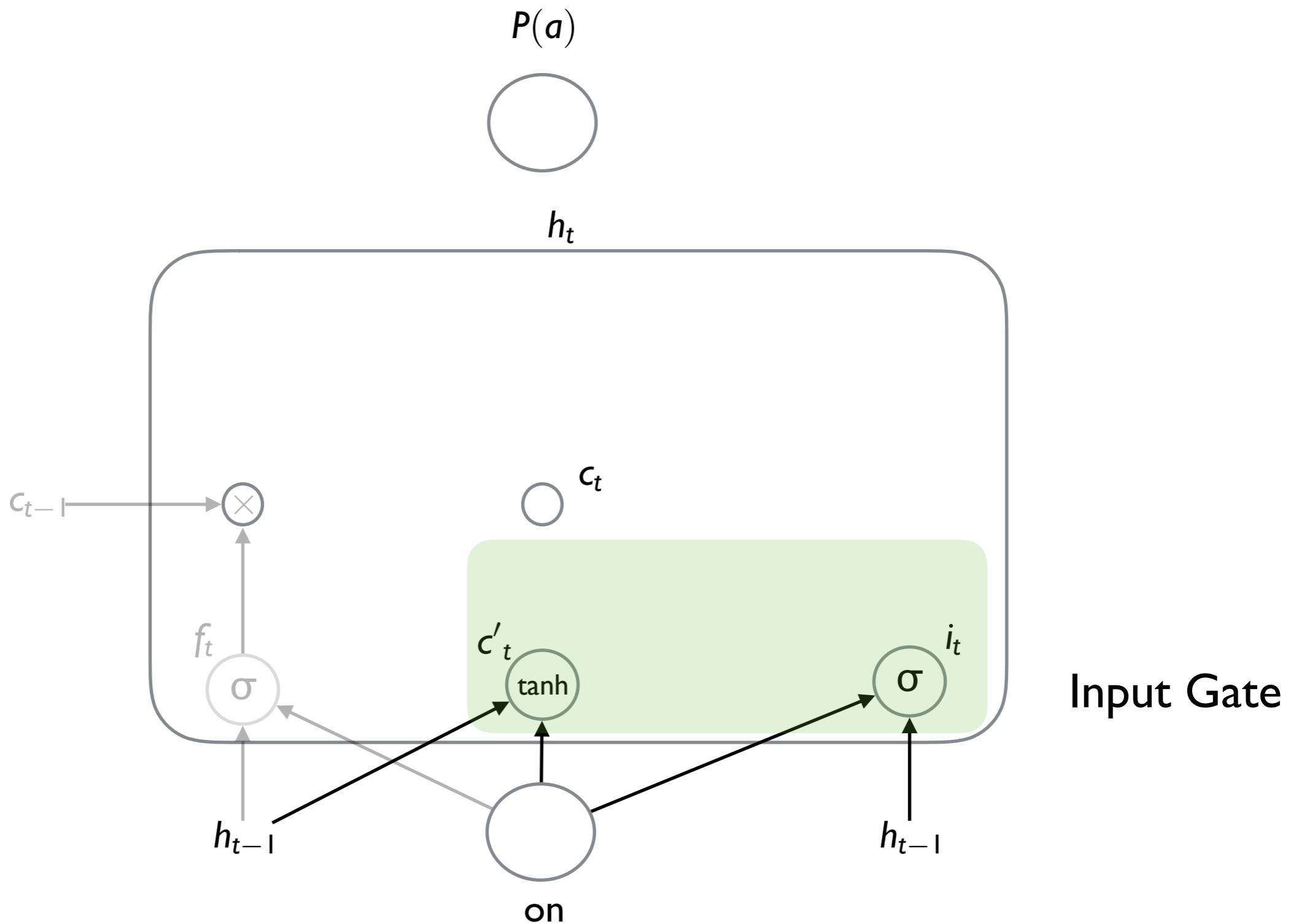
2.Adding new memories



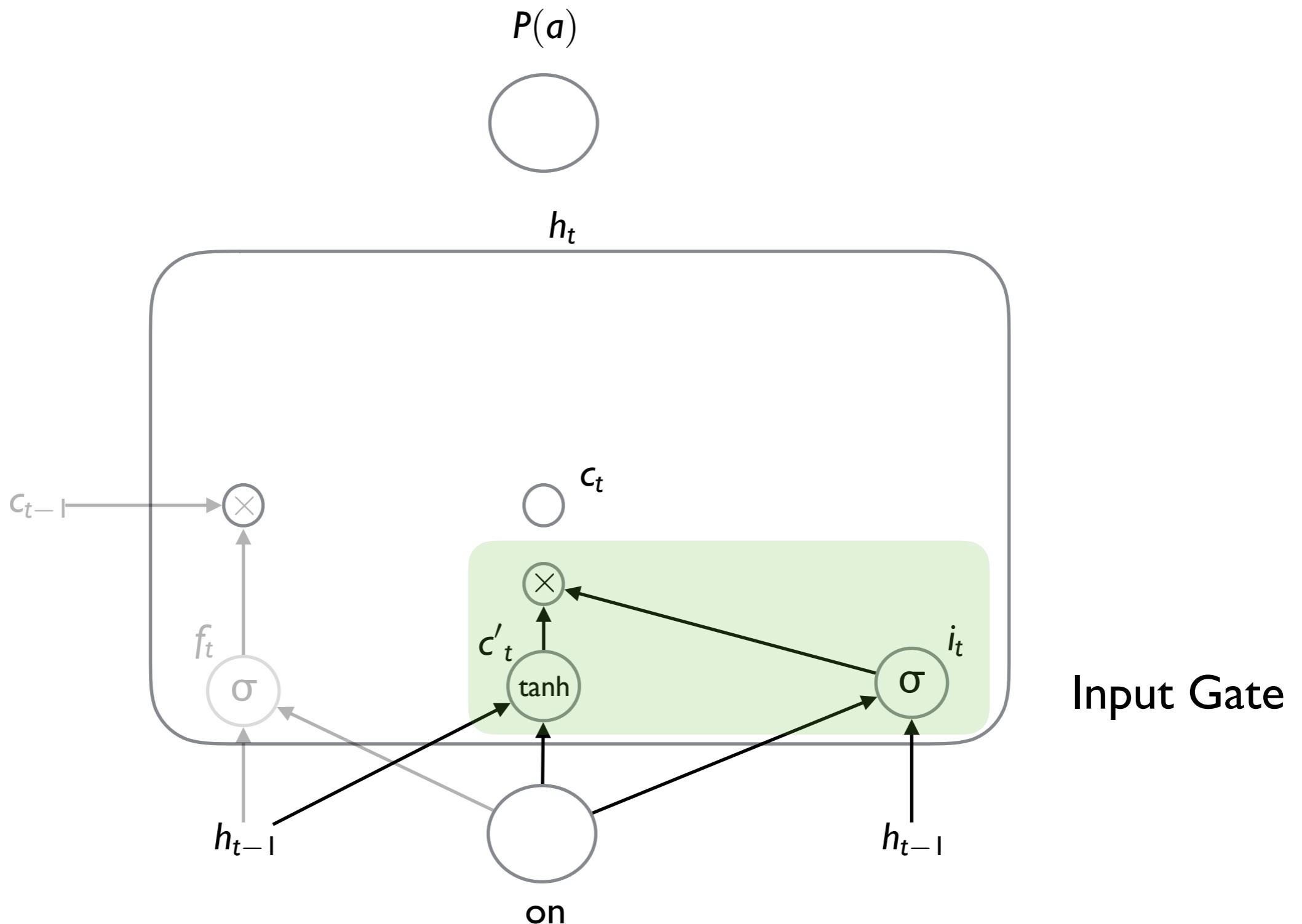
2. Adding new memories



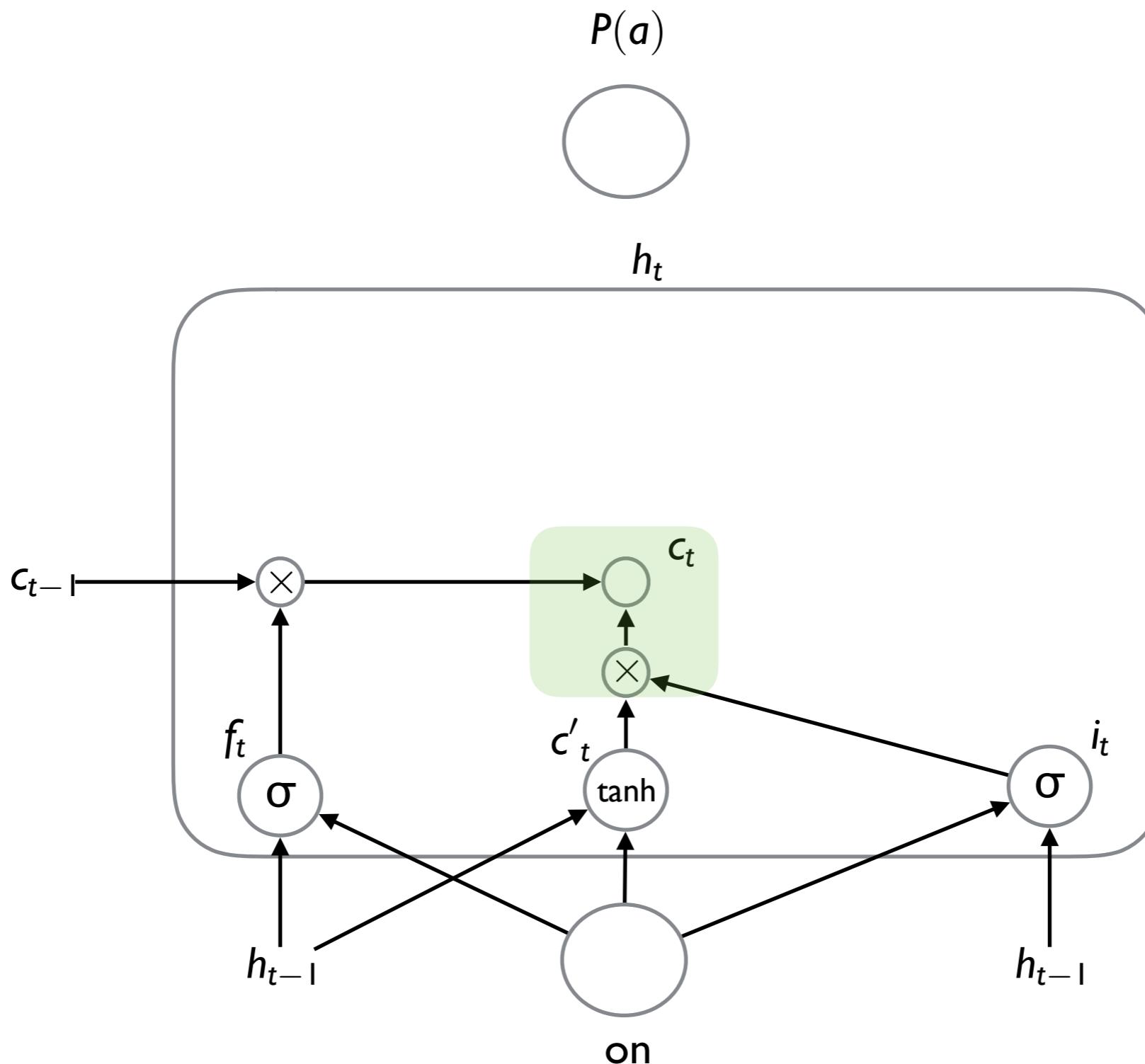
2. Adding new memories



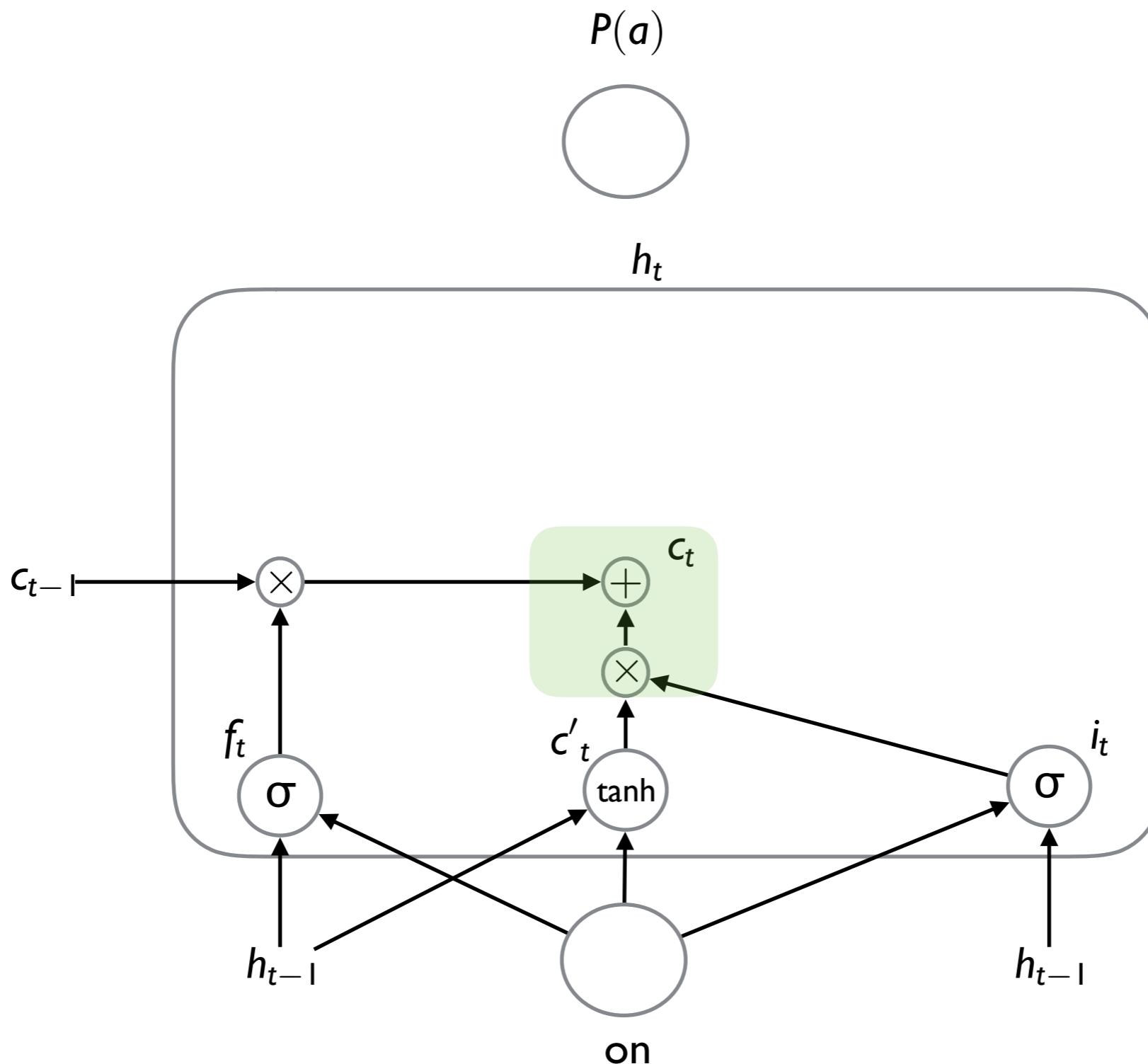
2. Adding new memories



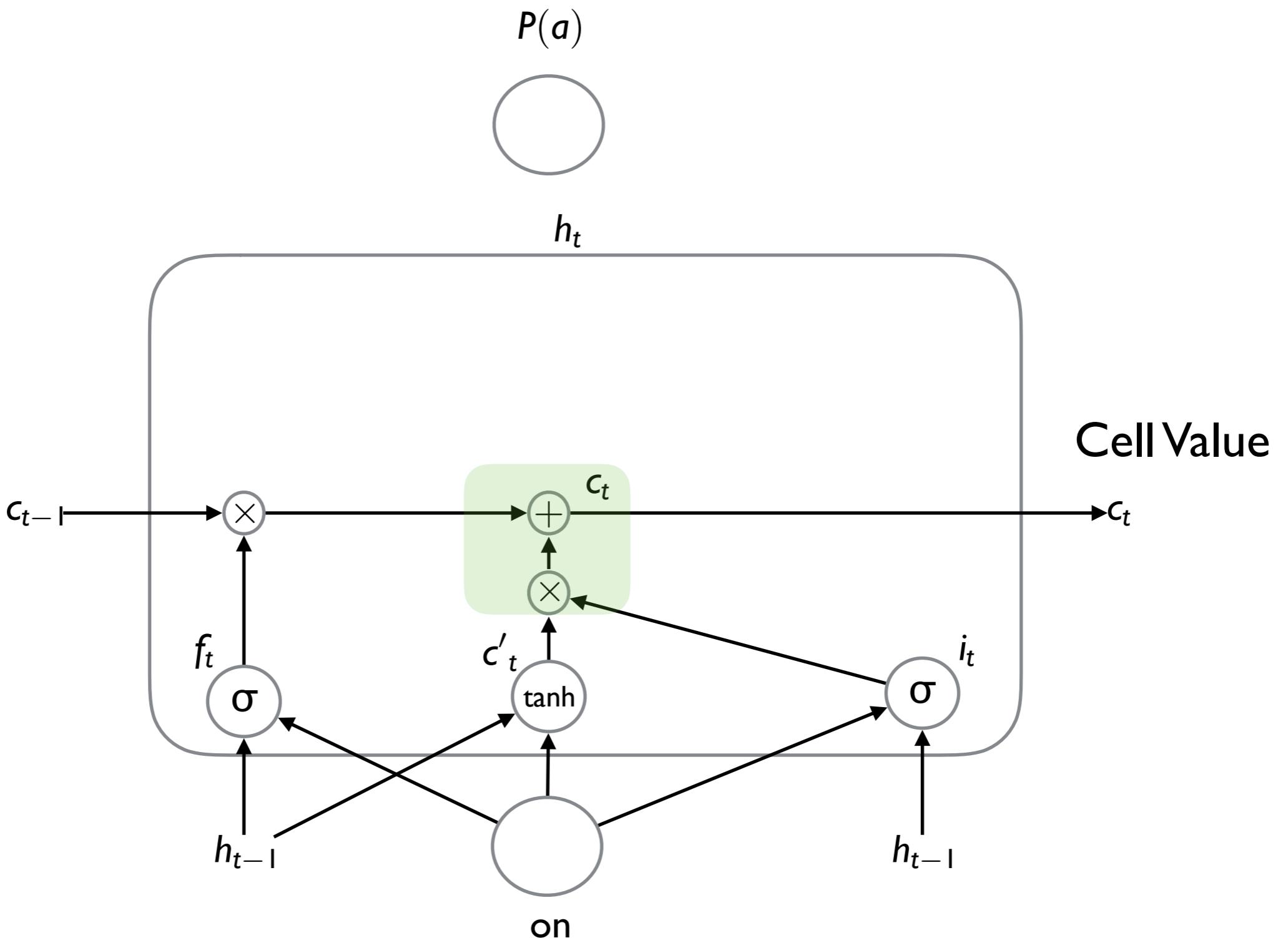
3. Updating the memory



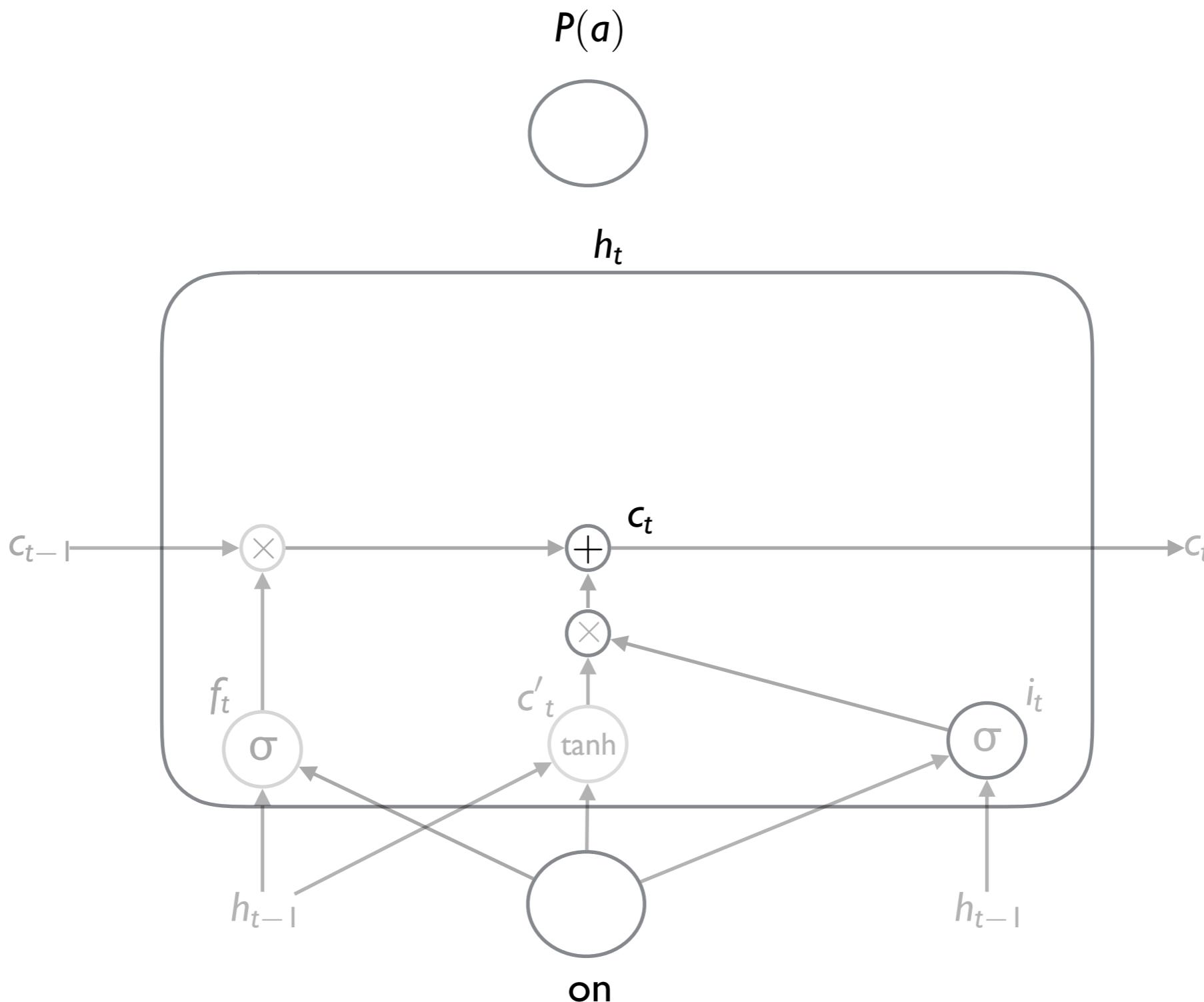
3. Updating the memory



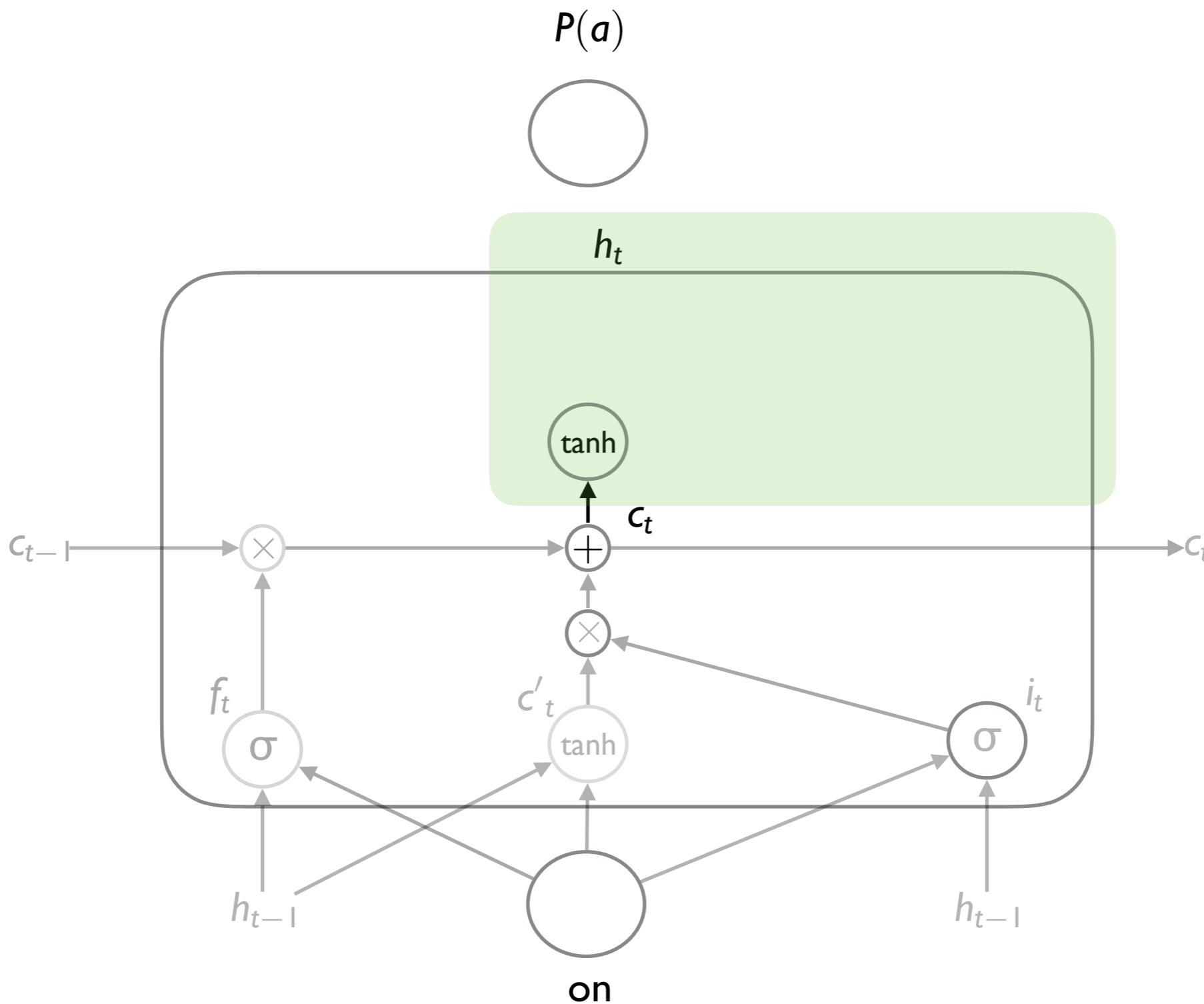
3. Updating the memory



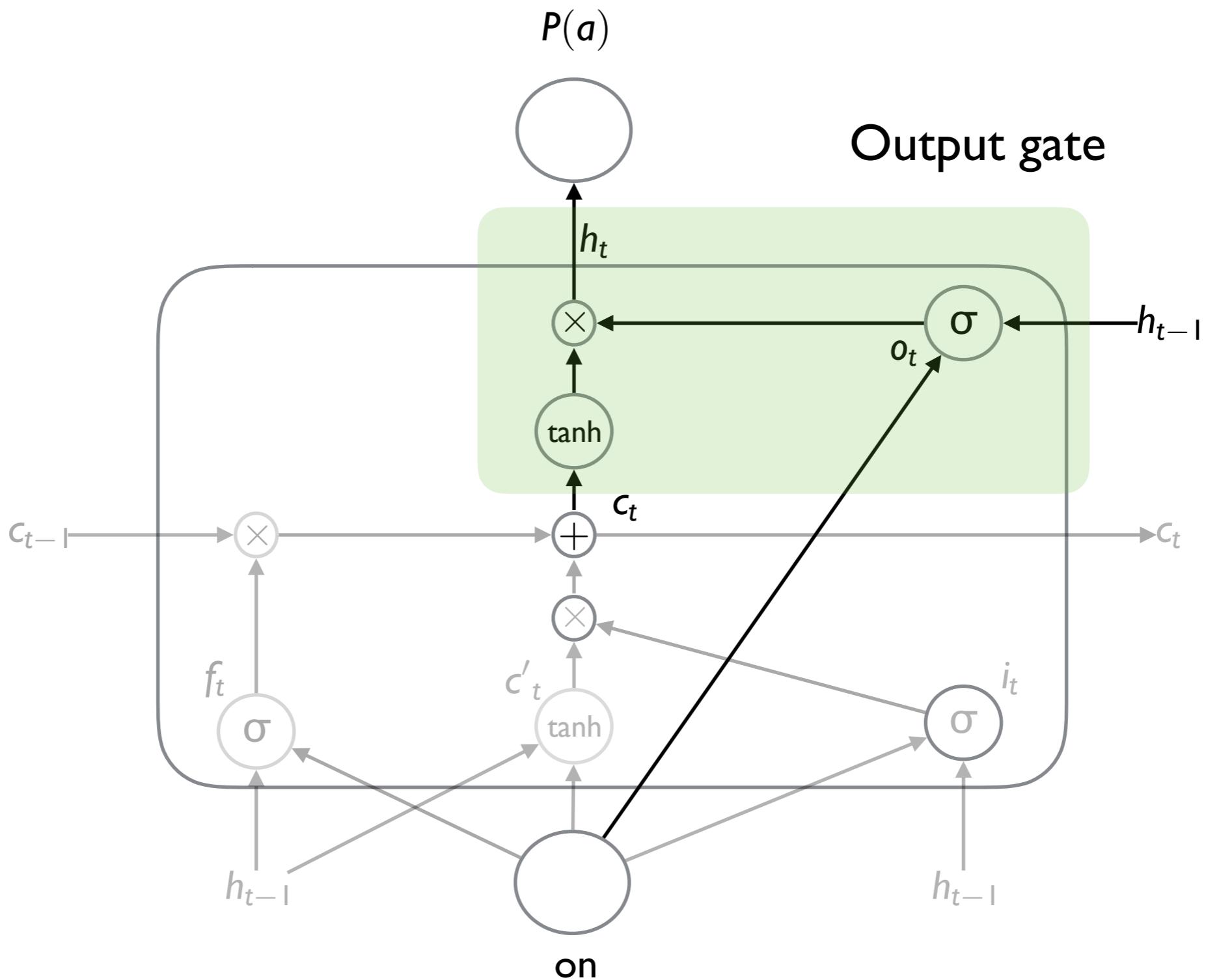
4. Calculating hidden state



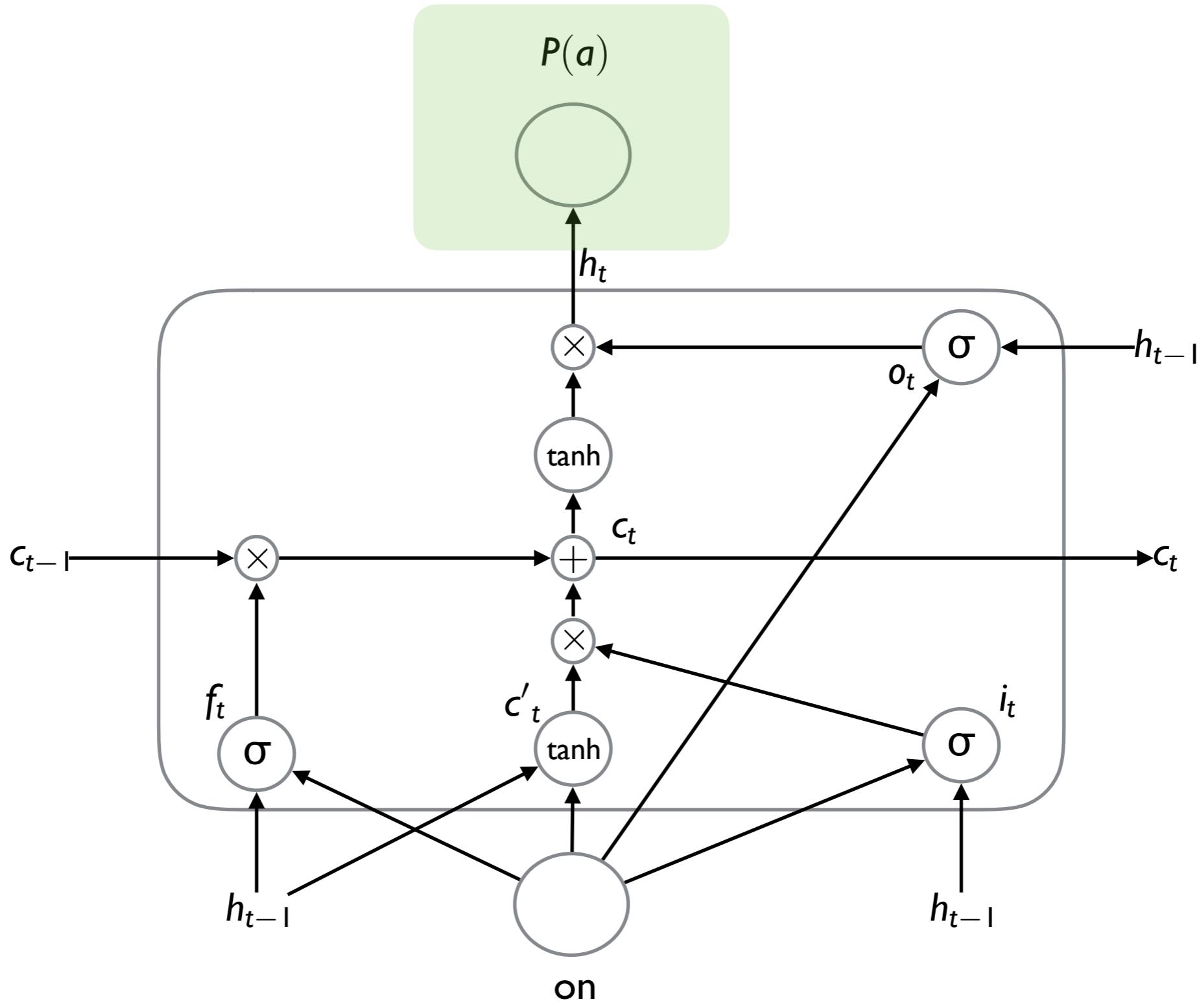
4. Calculating hidden state



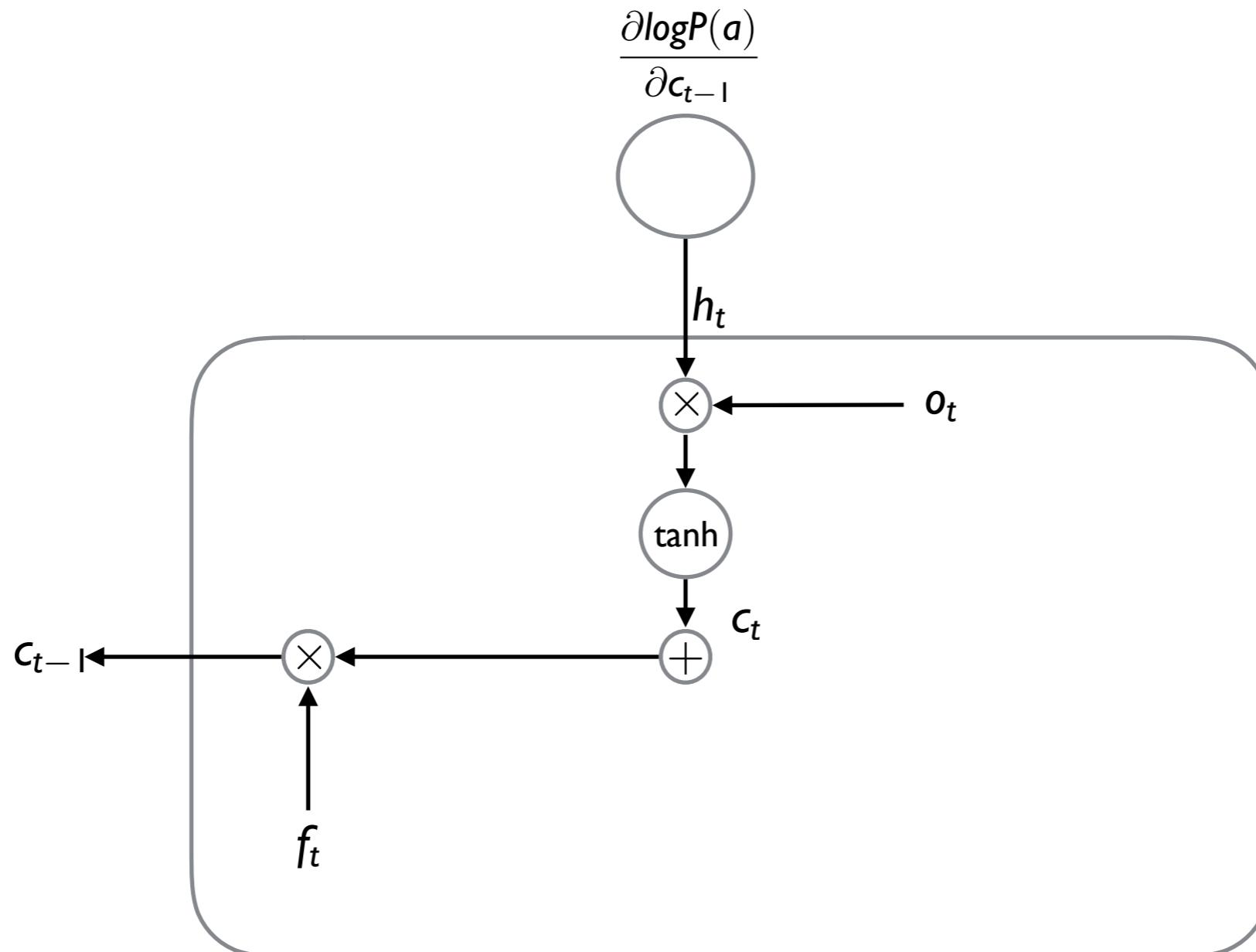
4. Calculating hidden state



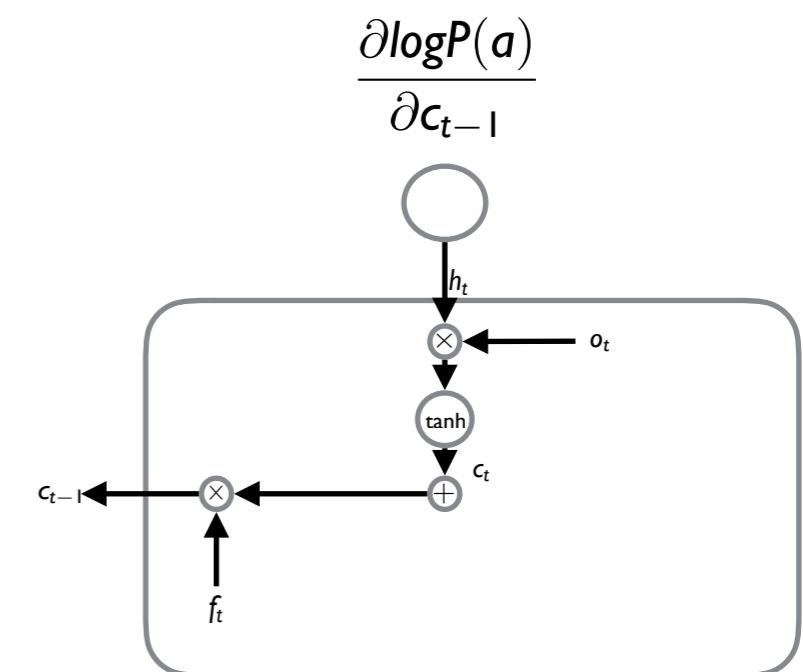
5. Computing probabilities



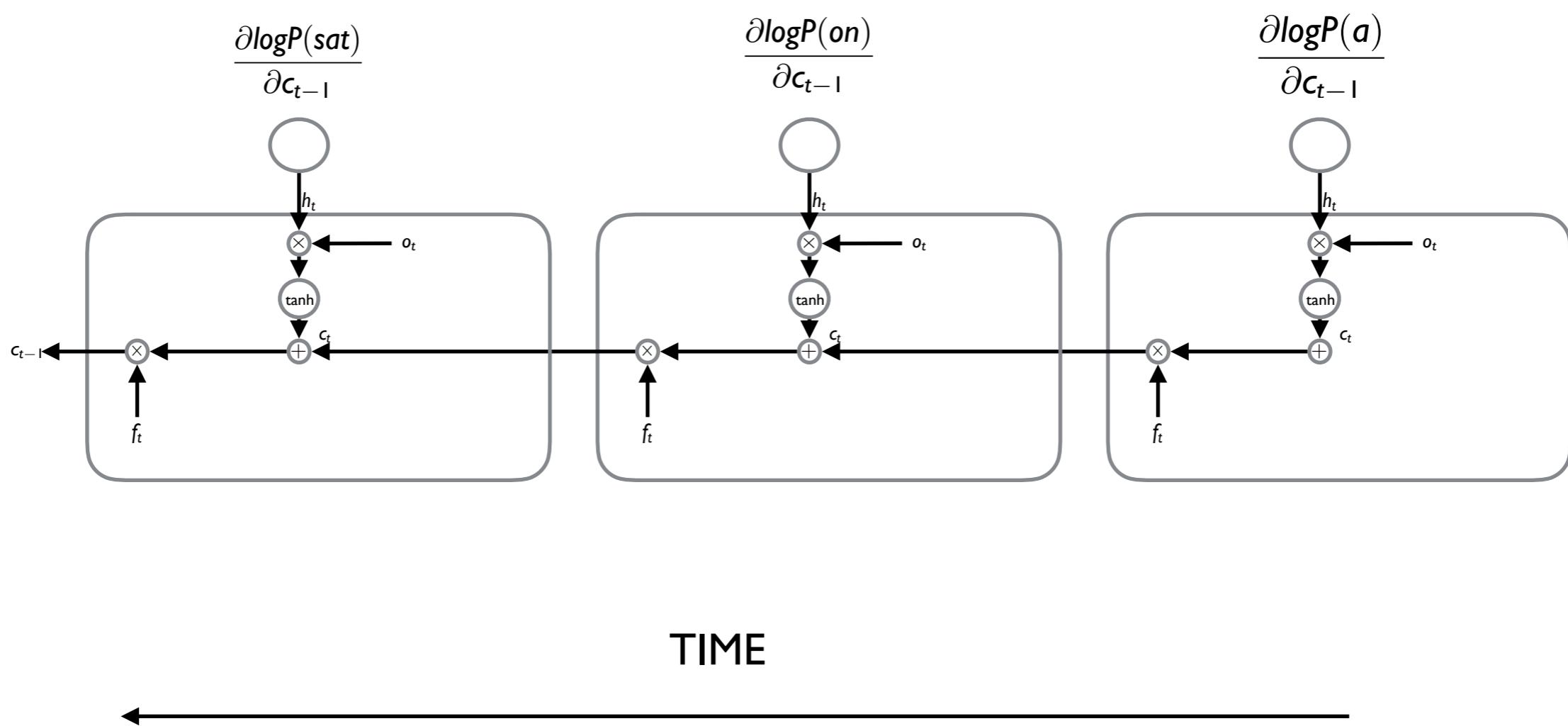
No more vanishing gradient



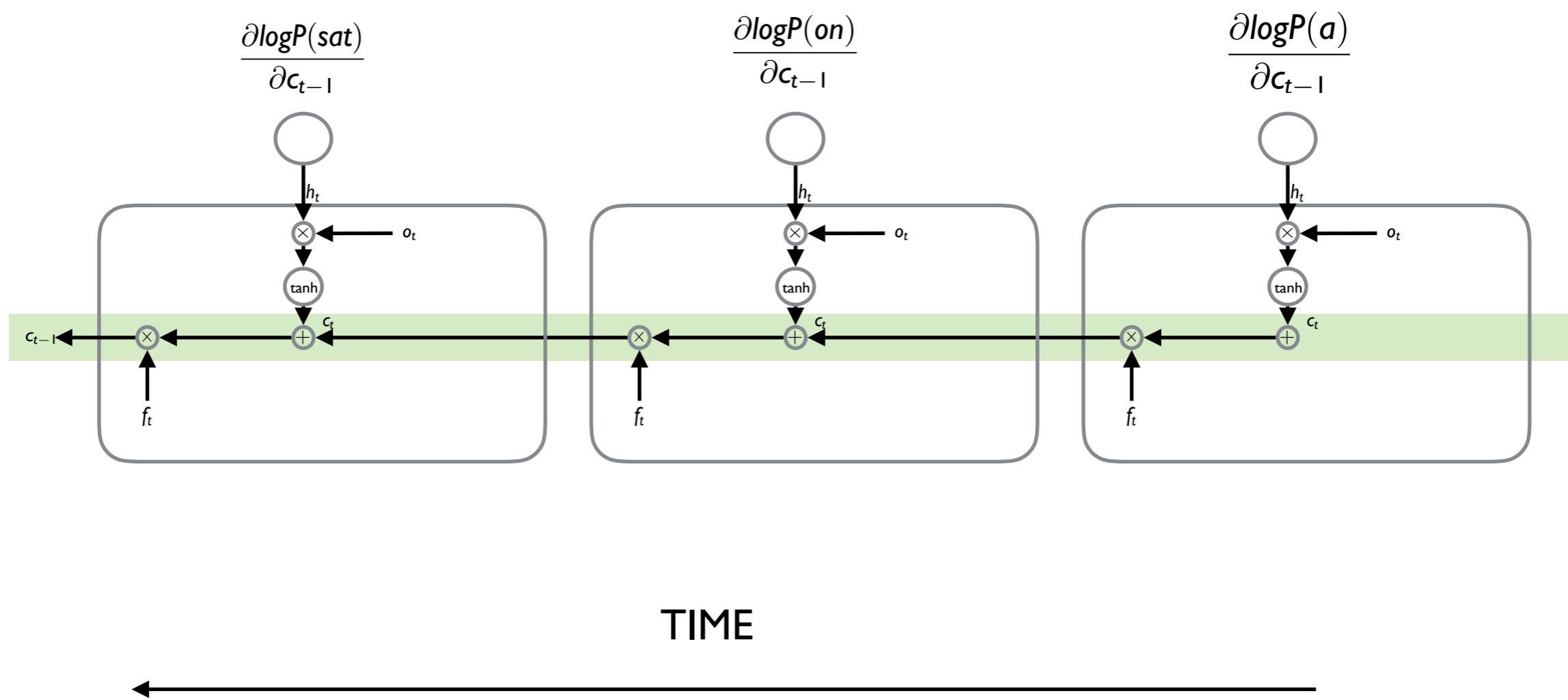
No more vanishing gradient



No more vanishing gradient

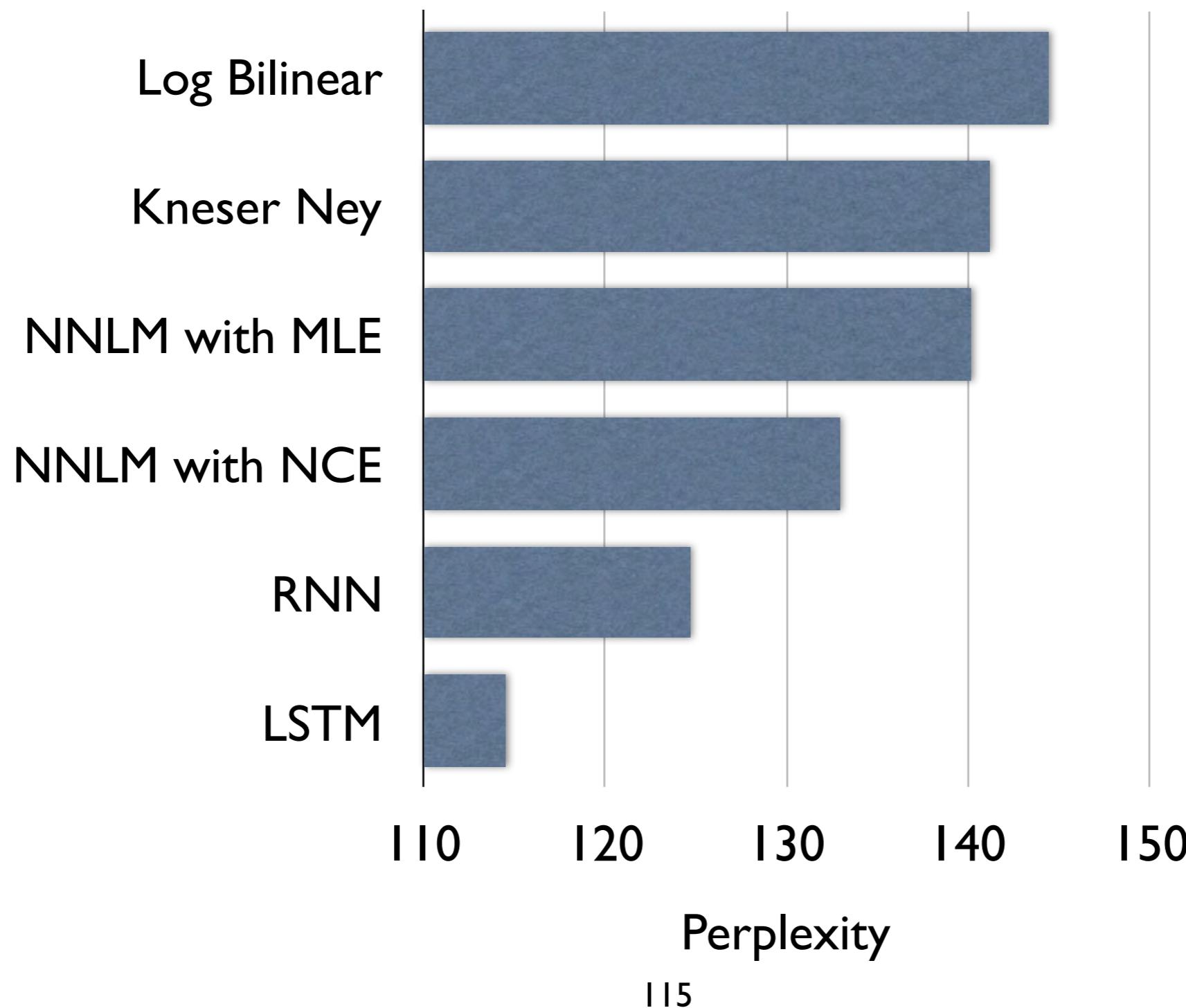


No more vanishing gradient



Improved Perplexity on Penn Treebank

Zaremba et al., 2014



LSTM Successes in NLP

- Language Modeling
- Neural Machine Translation
- Natural Language Parsing
- Tagging

Large Vocabulary LSTMs

- Language Modeling
- Neural Machine Translation
- Natural Language Parsing

Visualizing Simple LSTMs

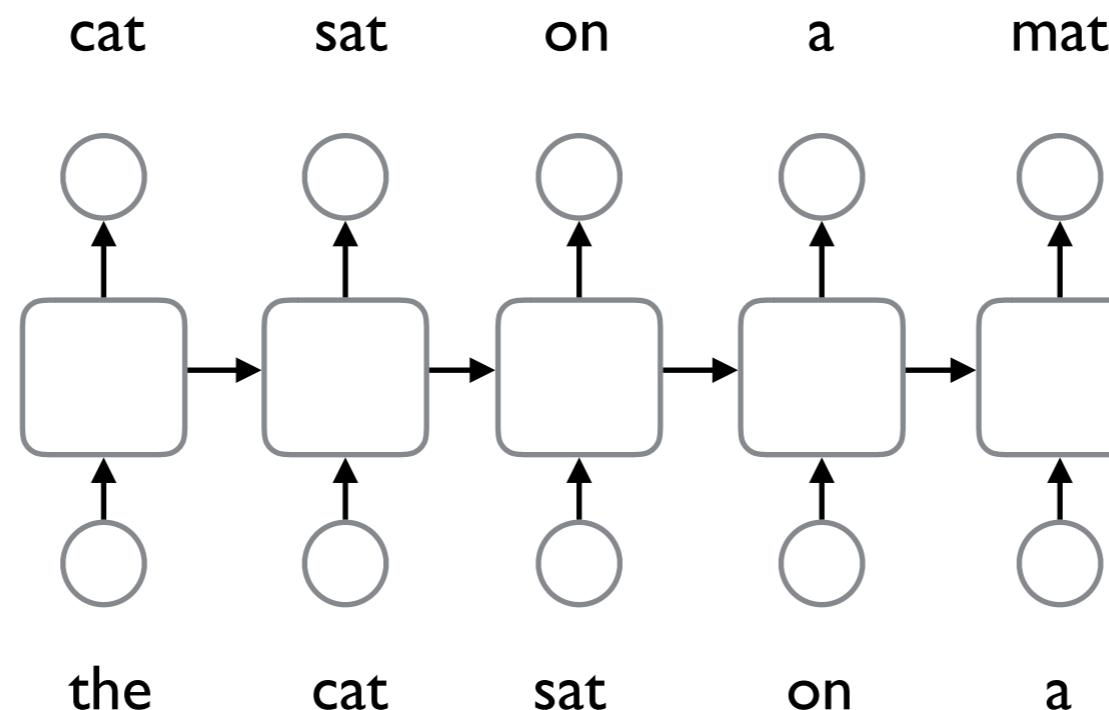
Joint work with Jon May

Encoder Decoder Framework for Sequence Sequence to Learning

Language Modeling:

Input: English

Output: English

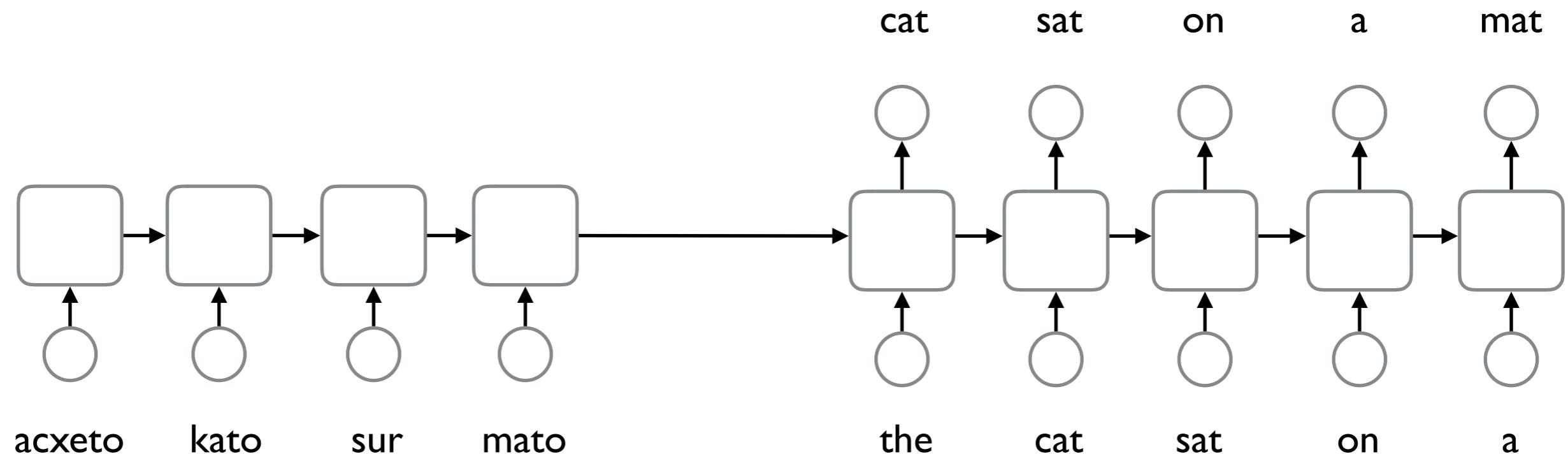


Encoder Decoder Framework for Sequence Sequence to Learning

Machine Translation

Input: Esperanto

Output: English

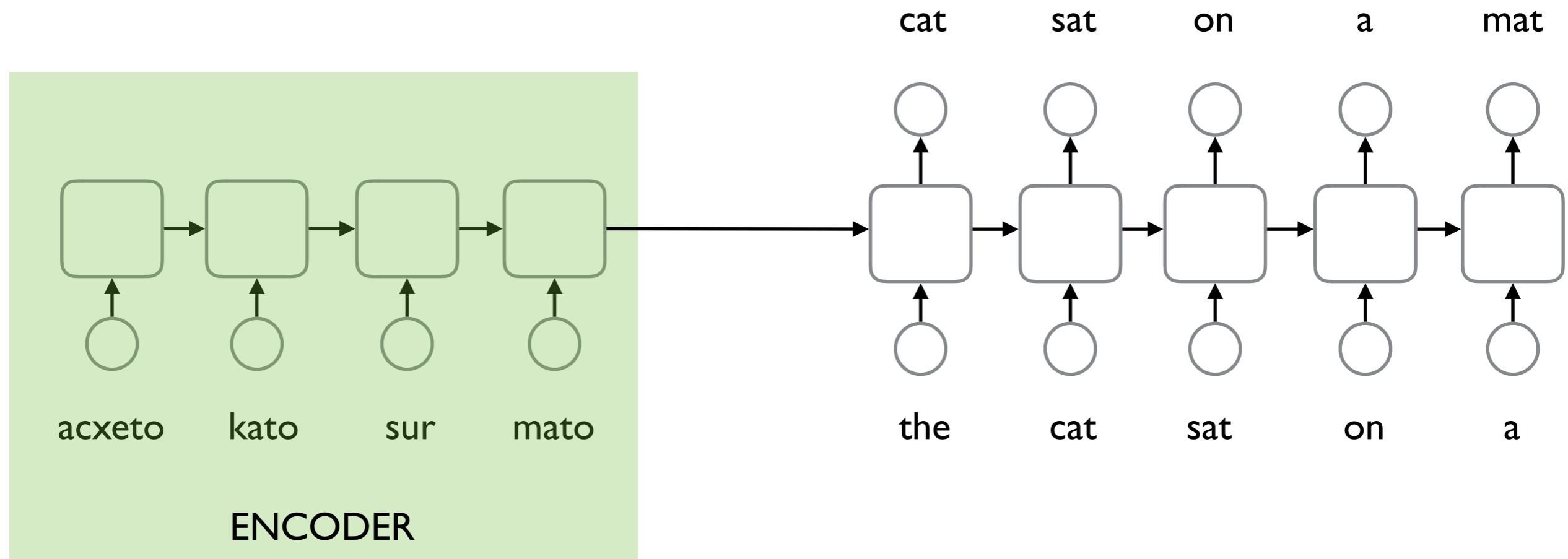


Encoder Decoder Framework for Sequence Sequence to Learning

Machine Translation

Input: Esperanto

Output: English

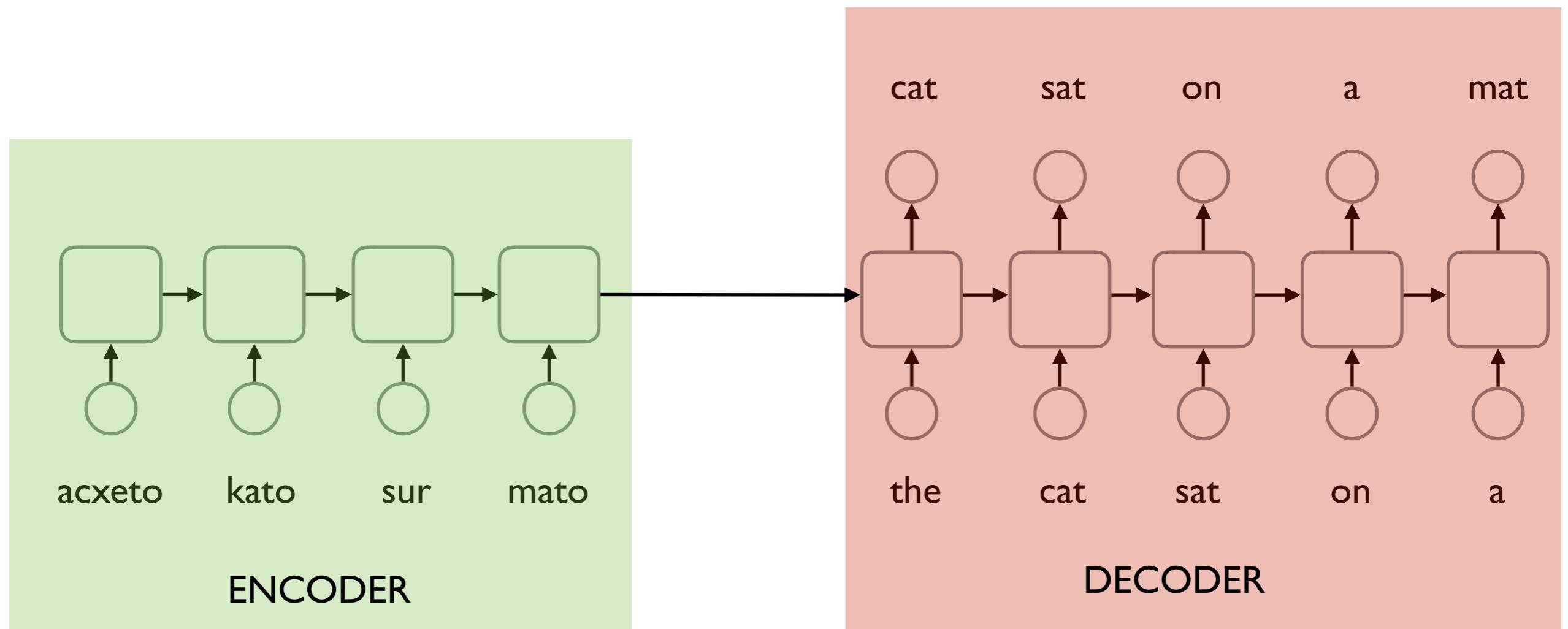


Encoder Decoder Framework for Sequence Sequence to Learning

Machine Translation

Input: Esperanto

Output: English

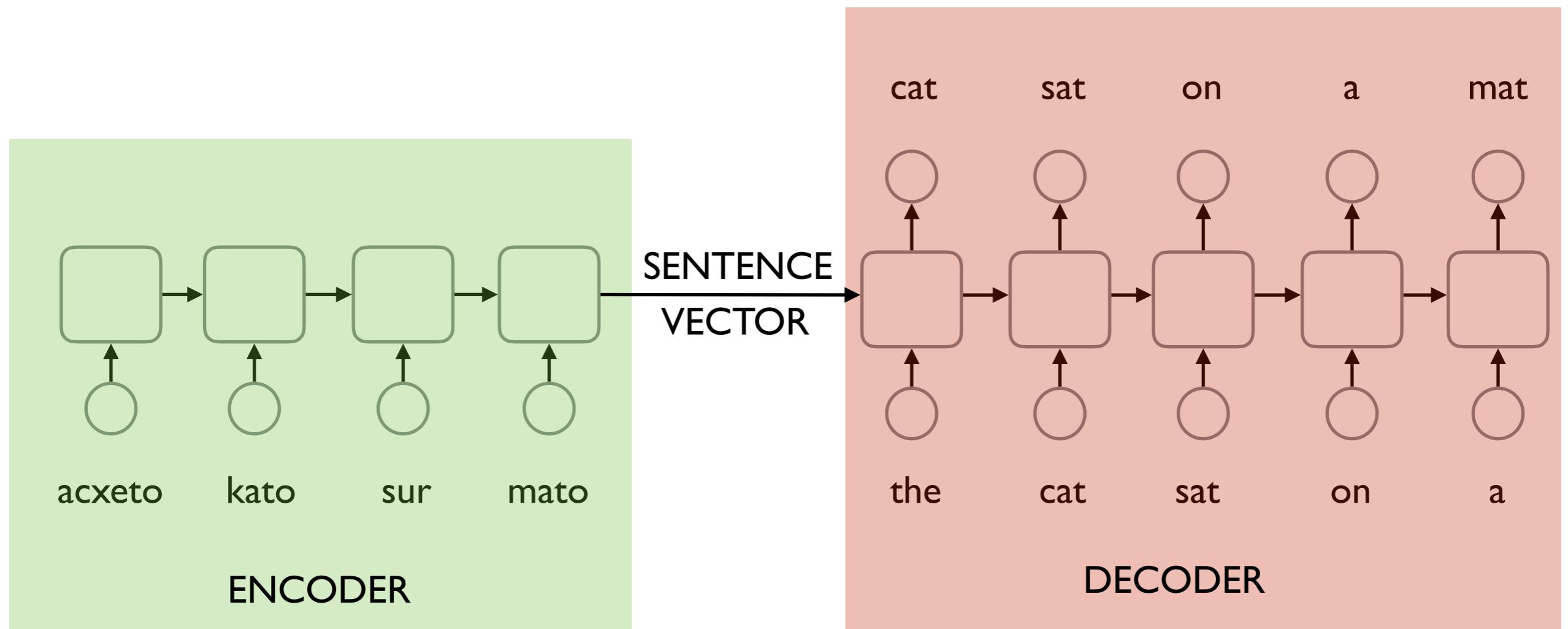


Encoder Decoder Framework for Sequence Sequence to Learning

Machine Translation

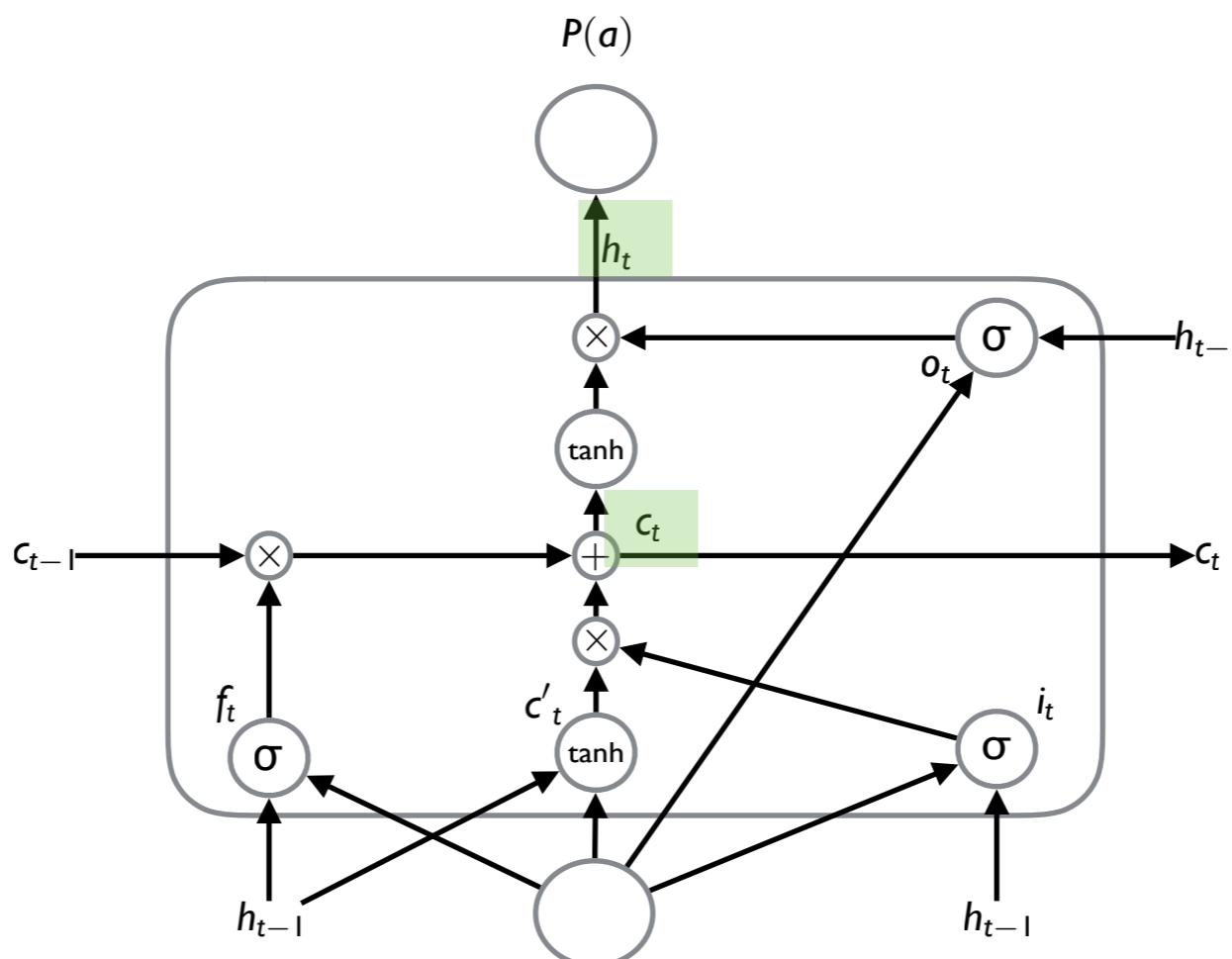
Input: Esperanto

Output: English

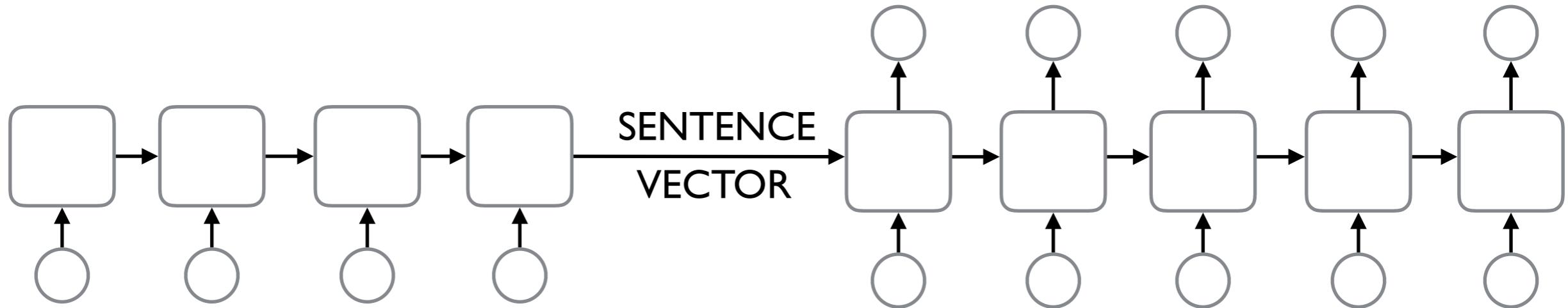


Visualization Approach

- Train on input/output sequences
- Look at LSTM internals for test input/output sequences



Counting a Symbol



Input:

a a a a a

a

a a a a a a a a a

Output:

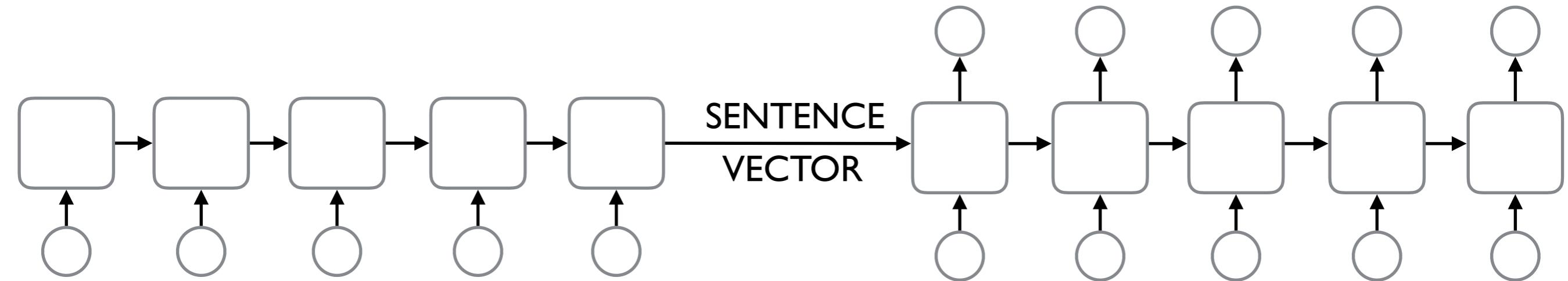
a a a a a

a

a a a a a a a a a

Input #(a) = Output #(a)

Counting a Symbol

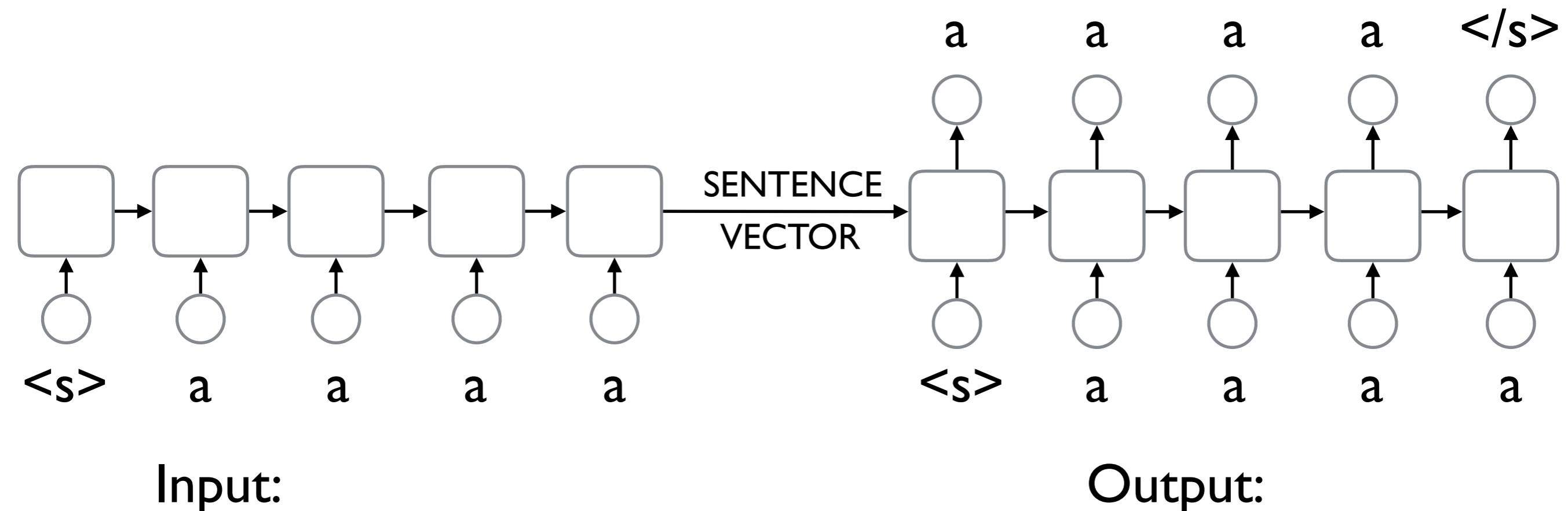


Input:
a a a a a

Output:
a a a a a

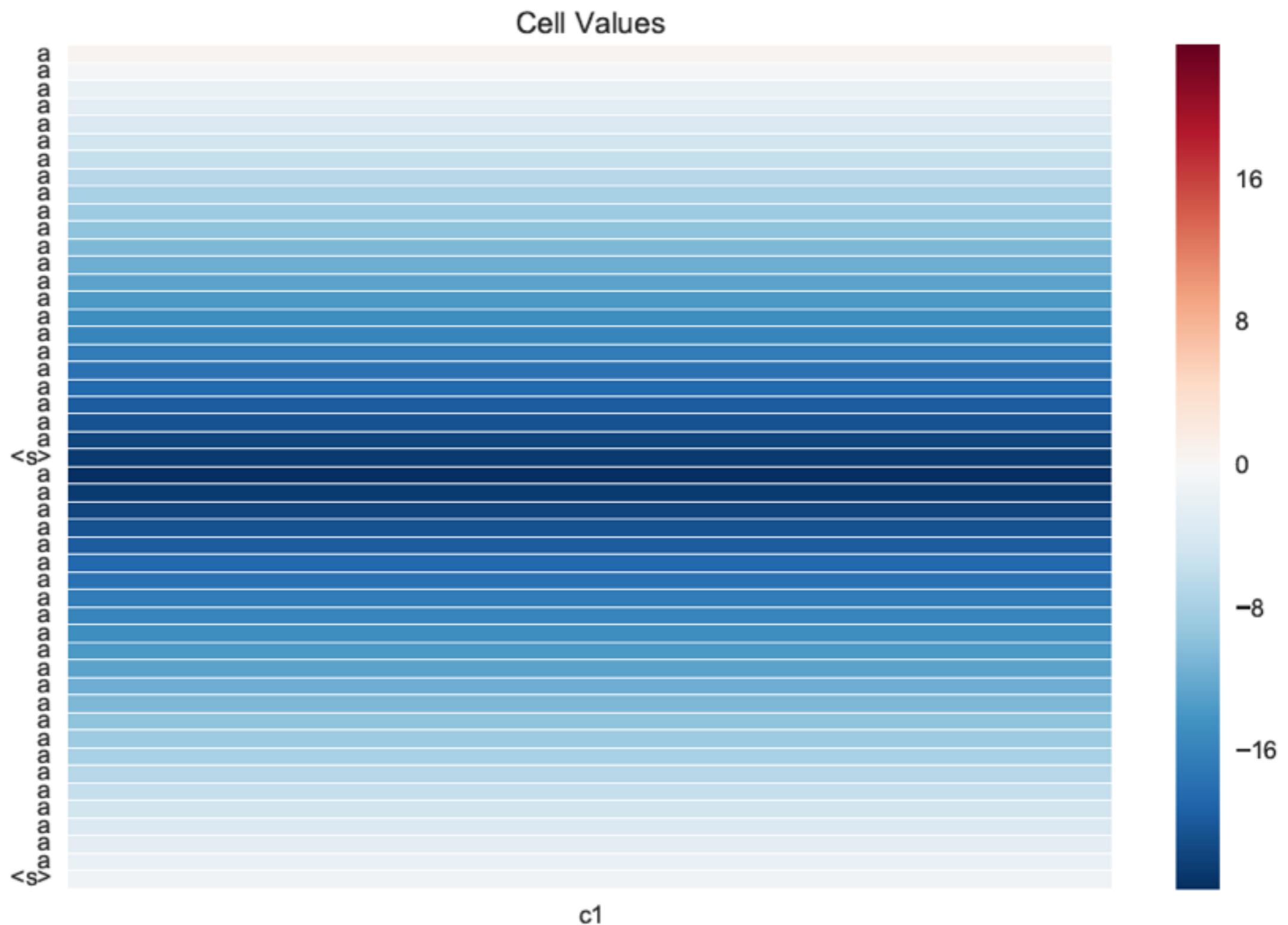
$$\text{Input } \#(a) = \text{Output } \#(a)$$

Counting a Symbol

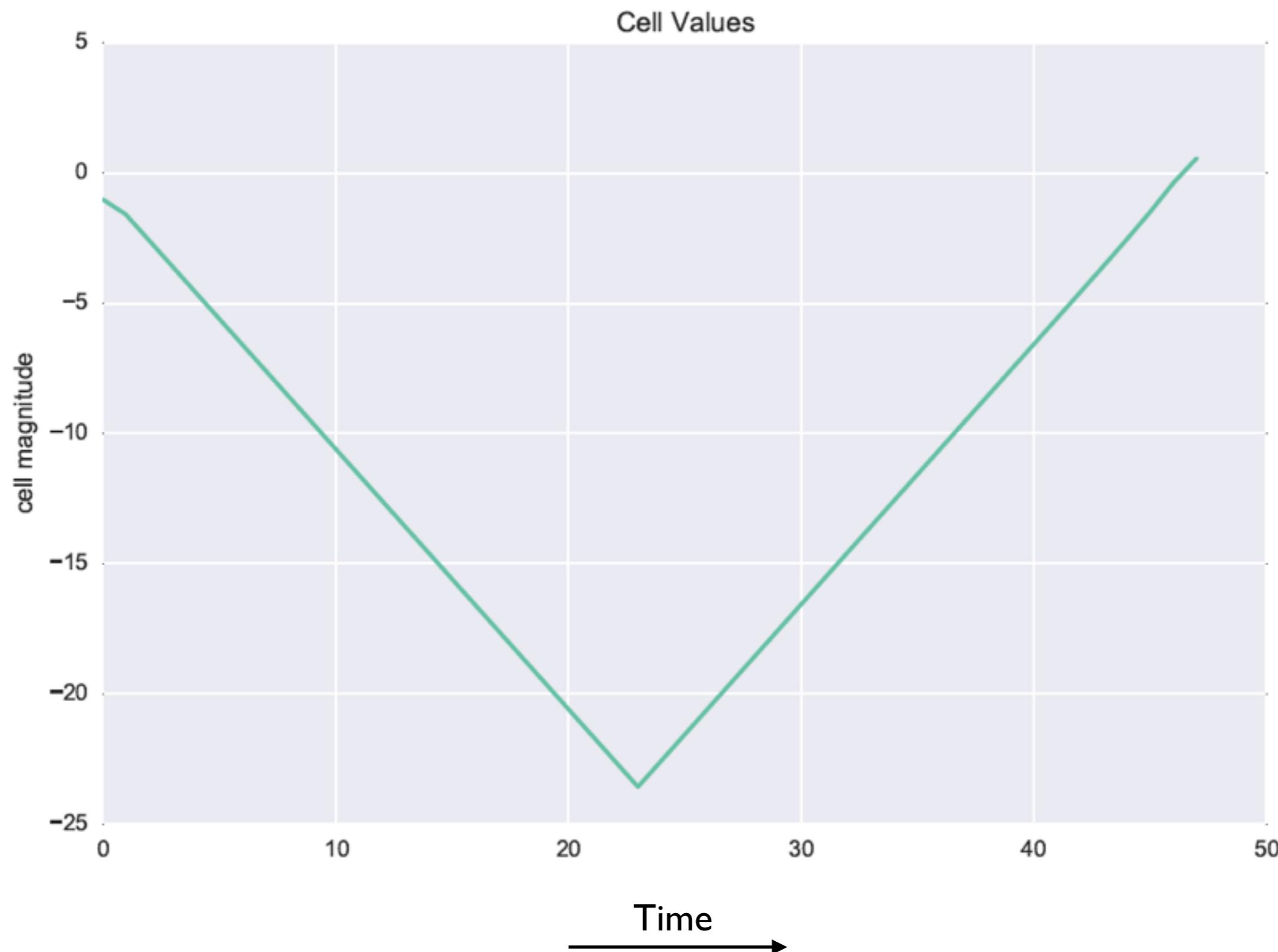


Input #(a) = Output #(a)

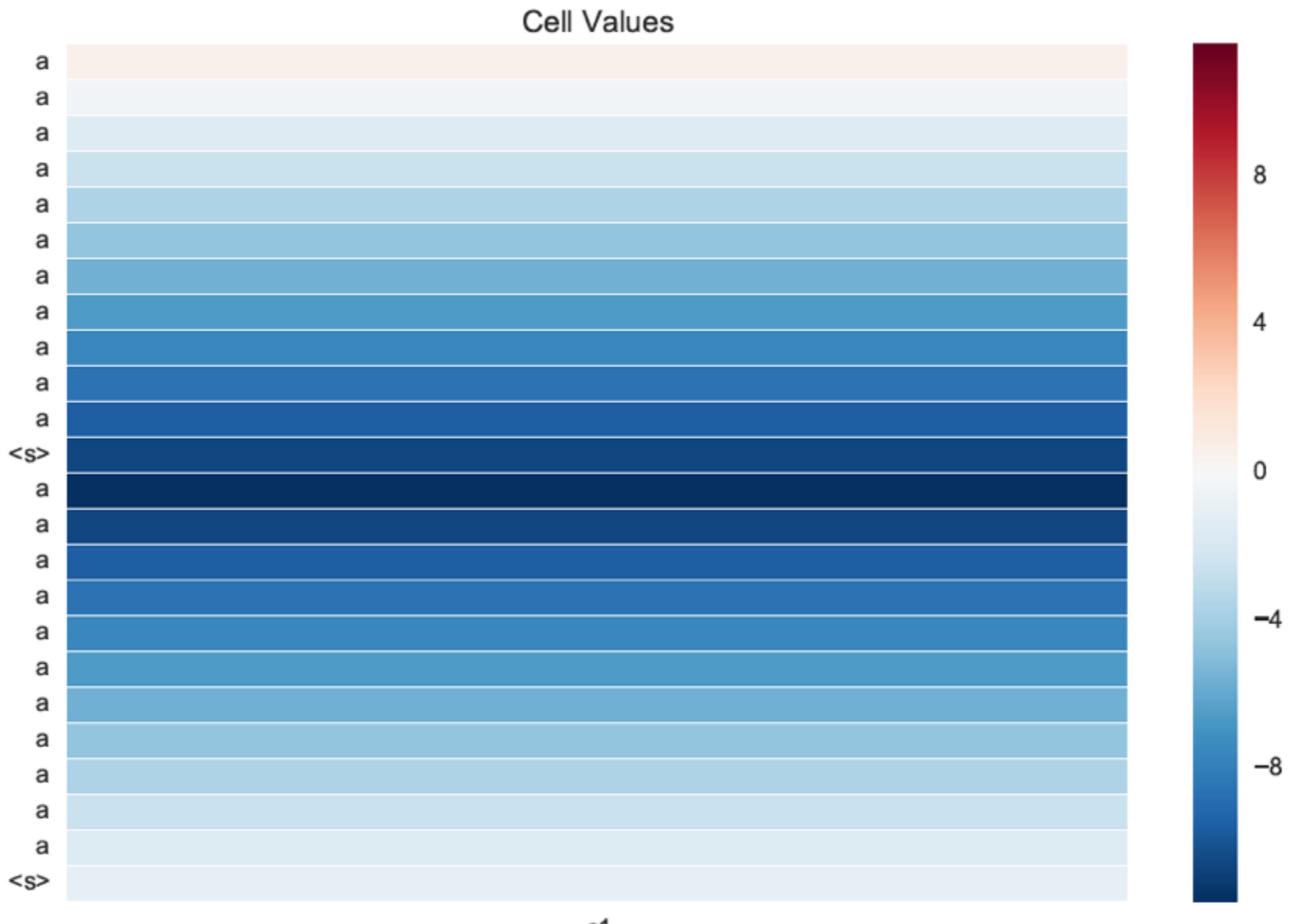
Counting:



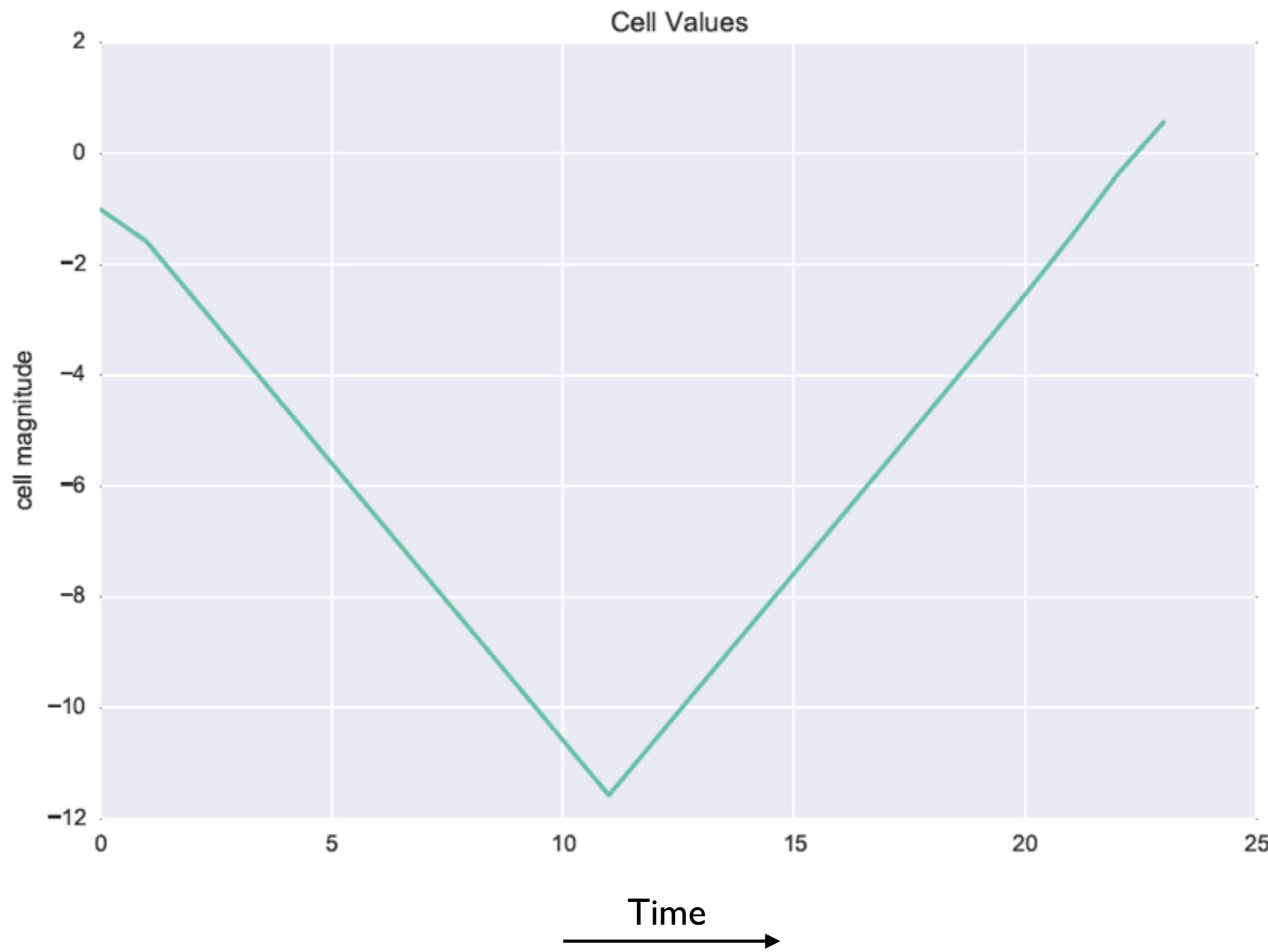
Counting:



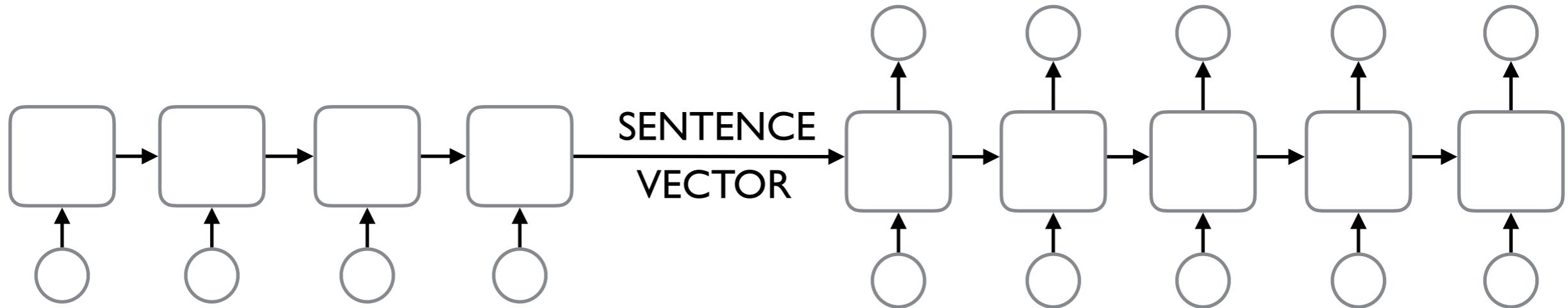
Counting:



Counting:



Counting a or b



Input:

a a a a a

b

a a a a a a a a a a

b b b b b b b b b b b b

Output:

a a a a a

b

a a a a a a a a a a

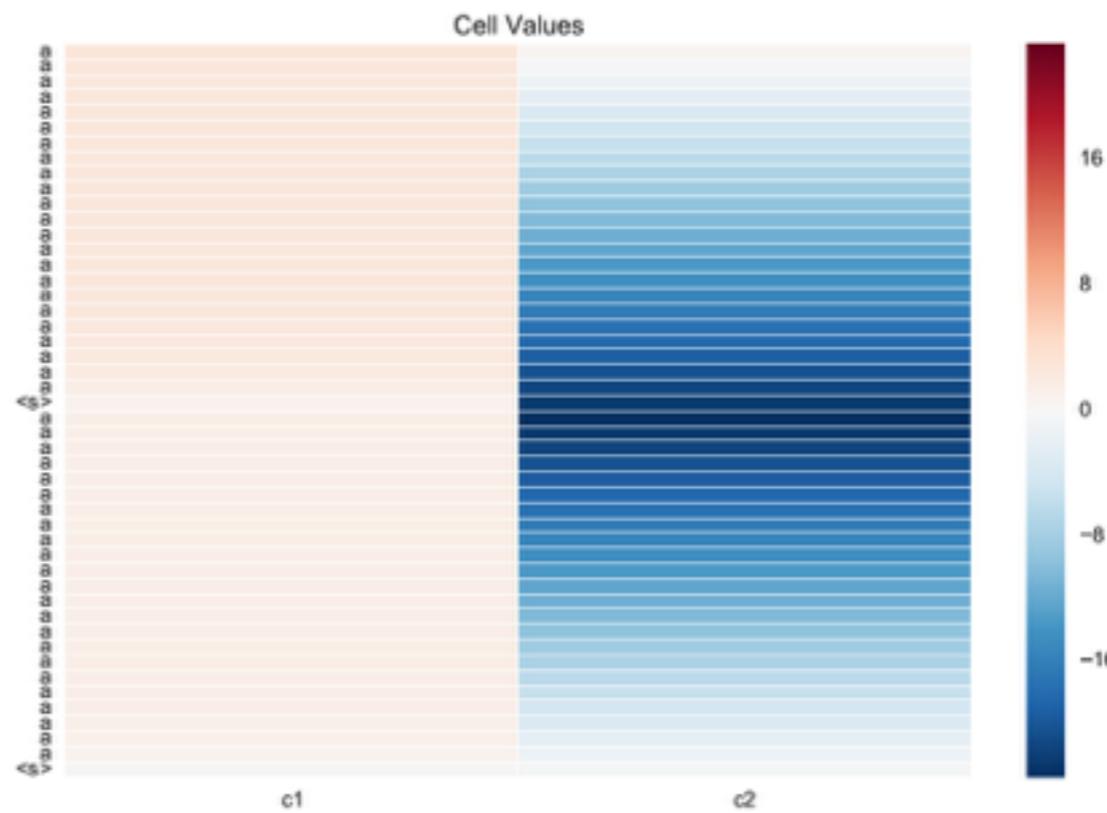
b b b b b b b b b b b b

Input #(a) = Output #(a)
OR

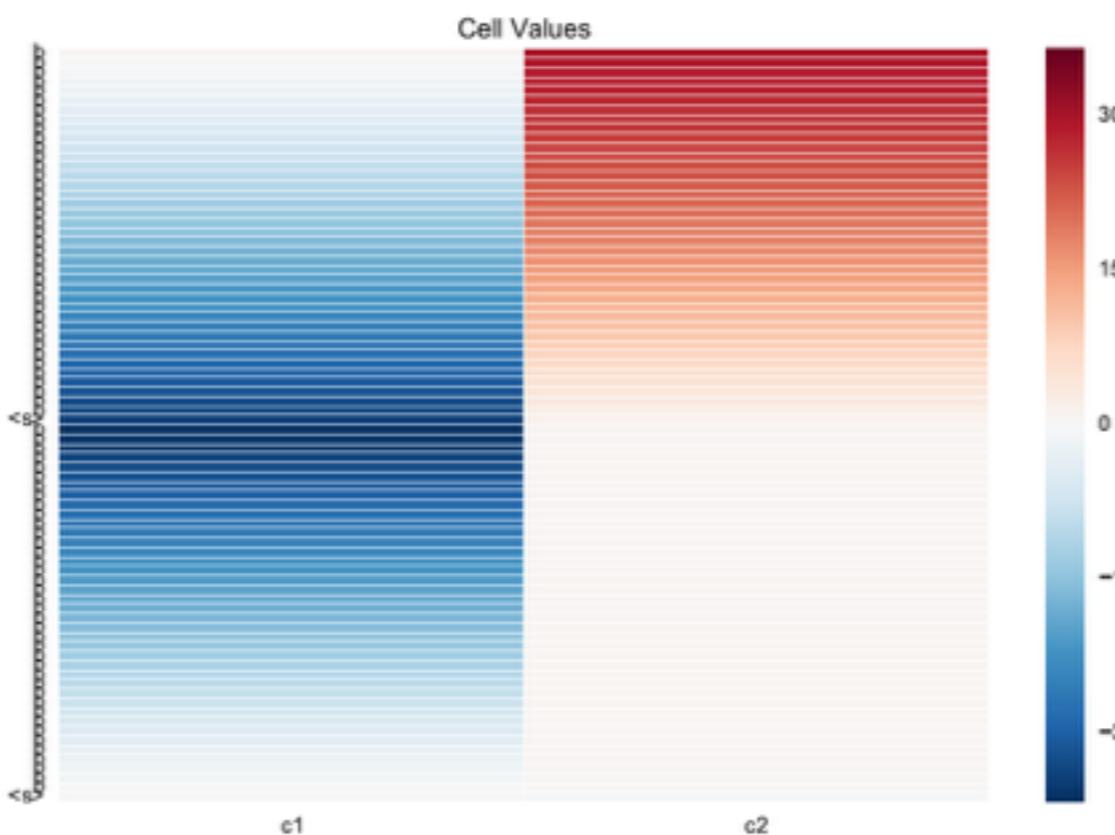
Input #(b) = Output #(b)

Counting a or b

Counting a

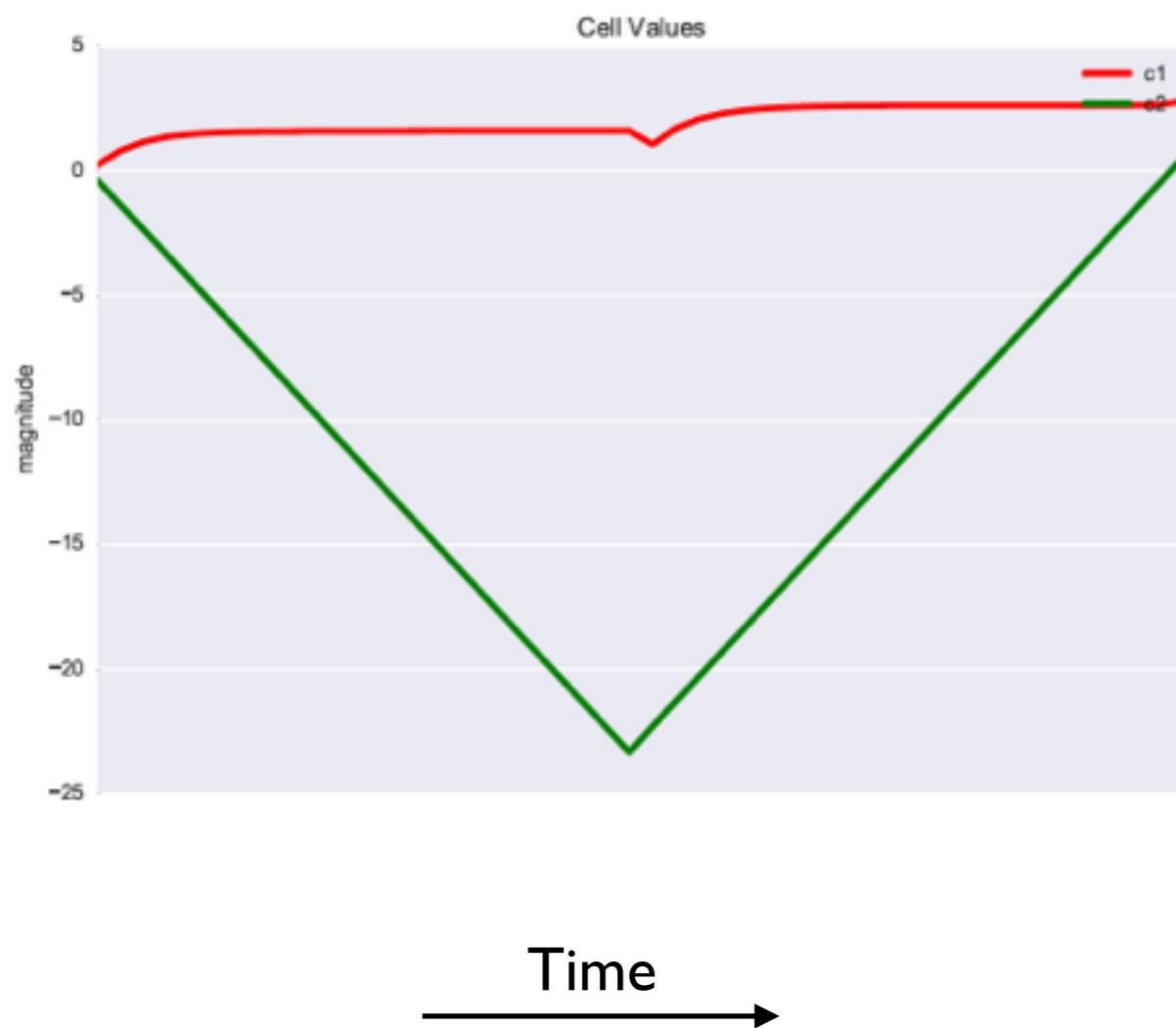


Counting b

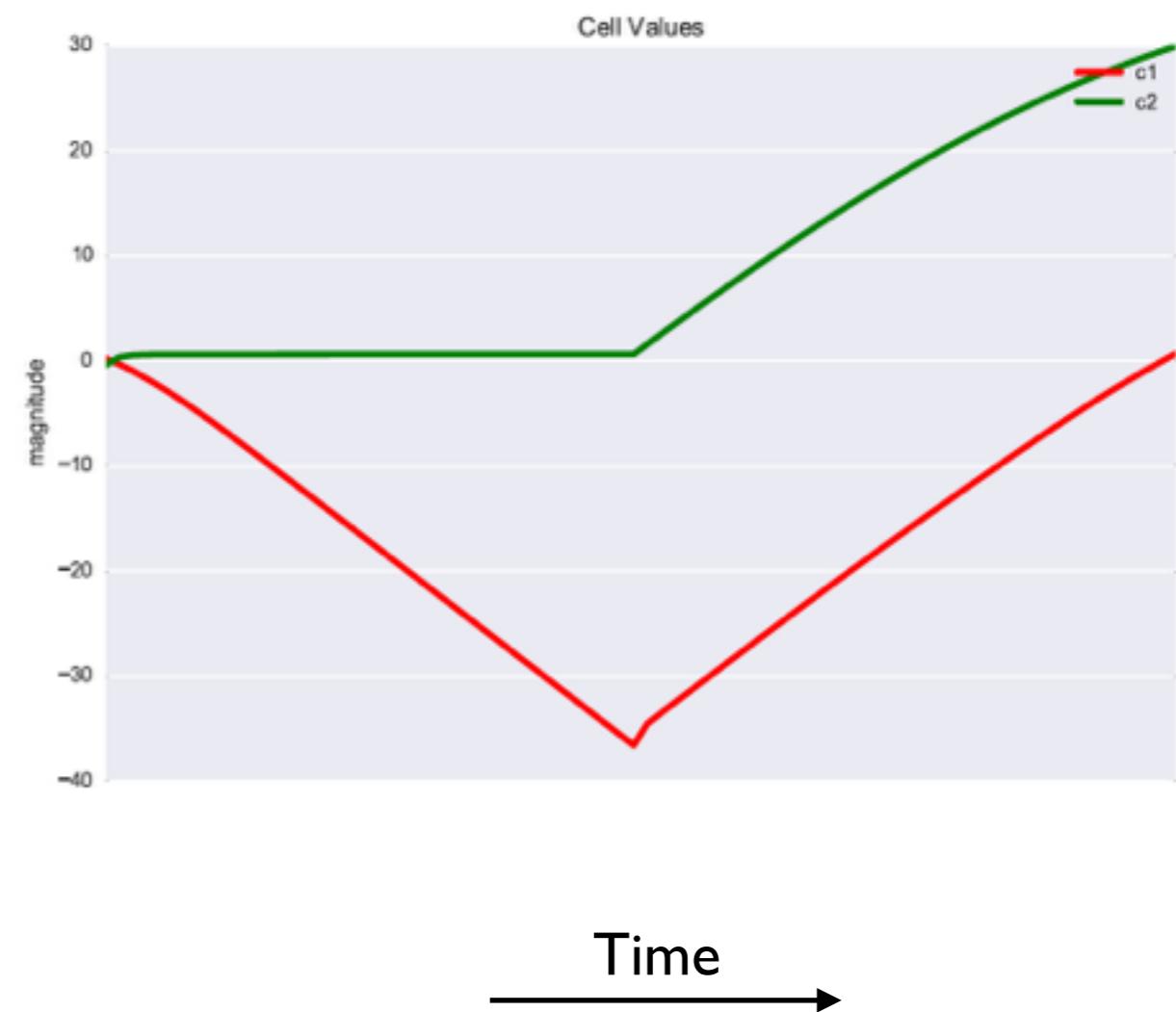


Counting a or b

Counting a

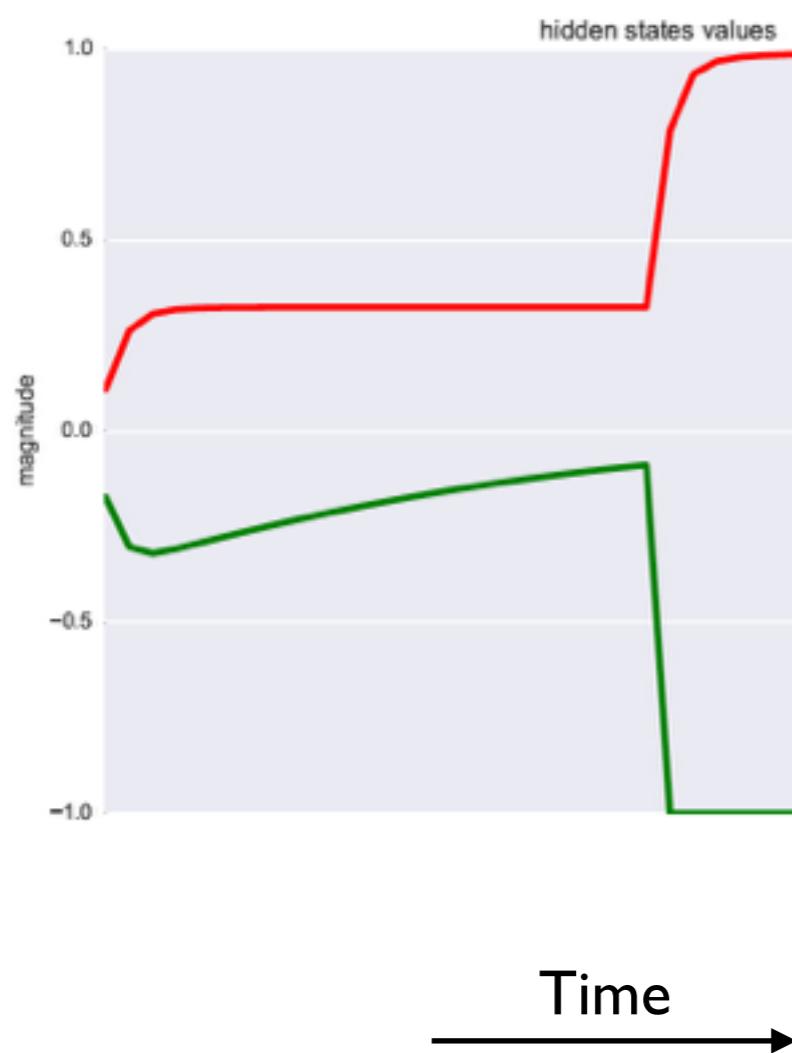


Counting b

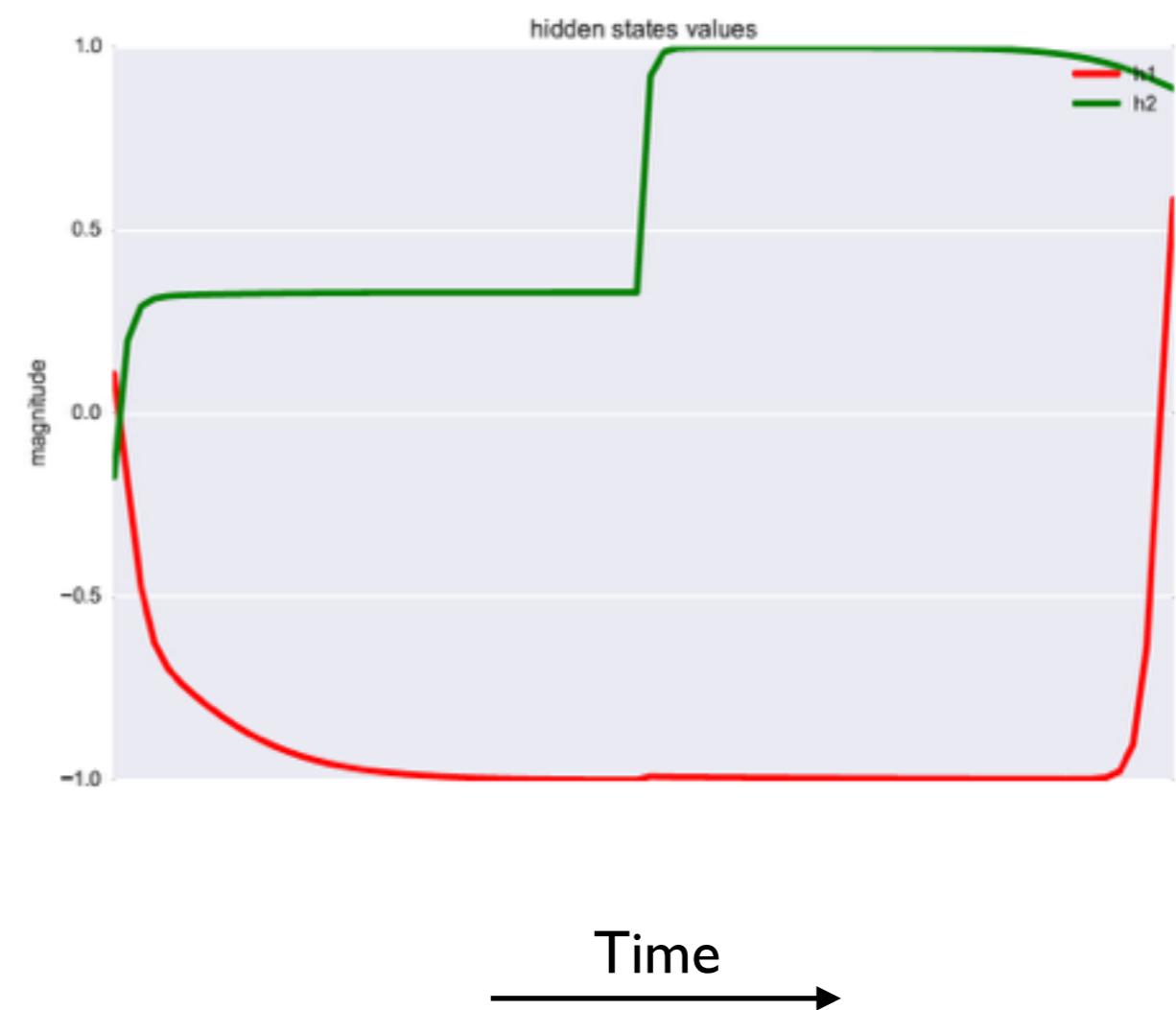


Counting a or b

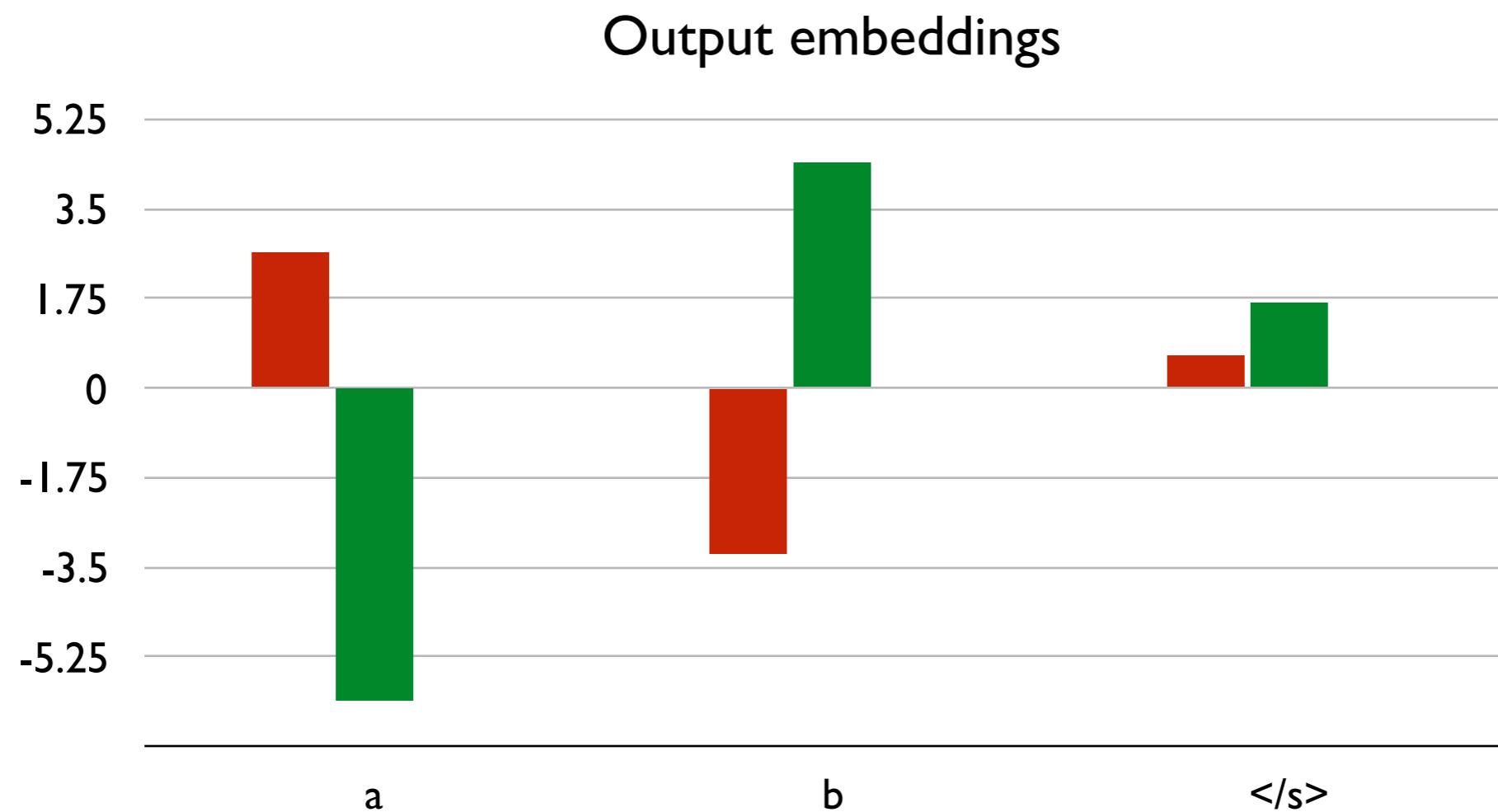
Counting a



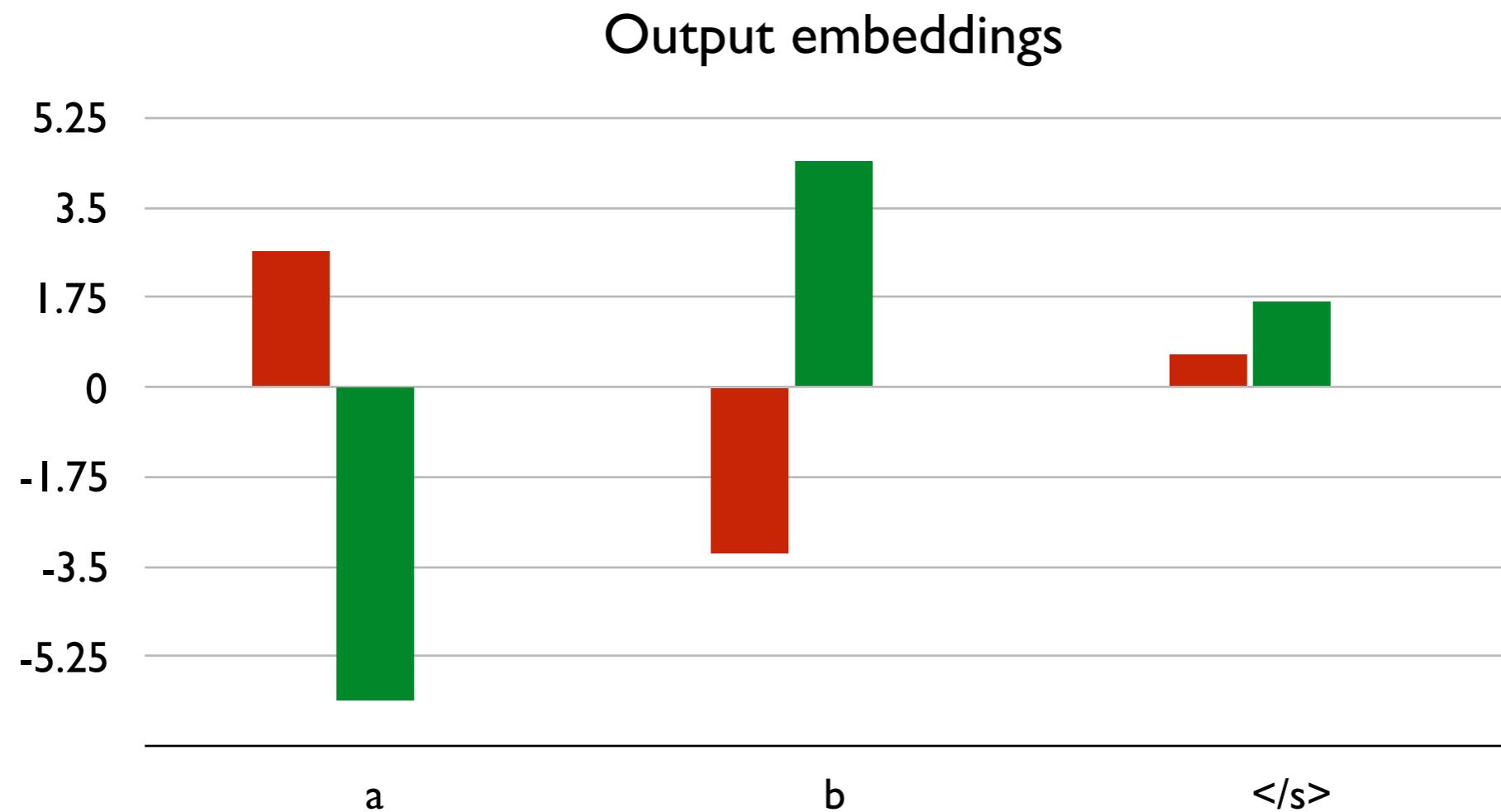
Counting b



Counting a or b



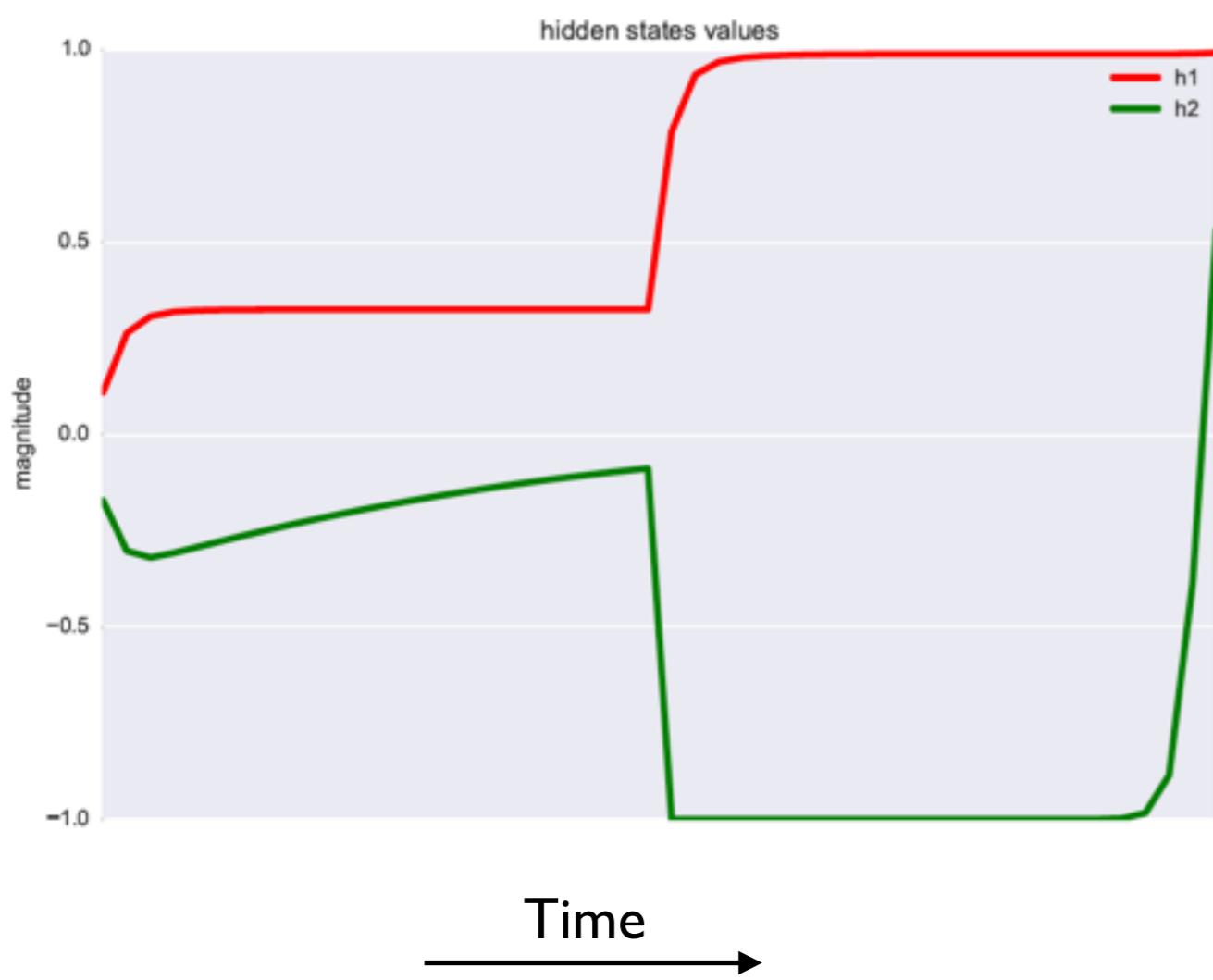
Counting a or b



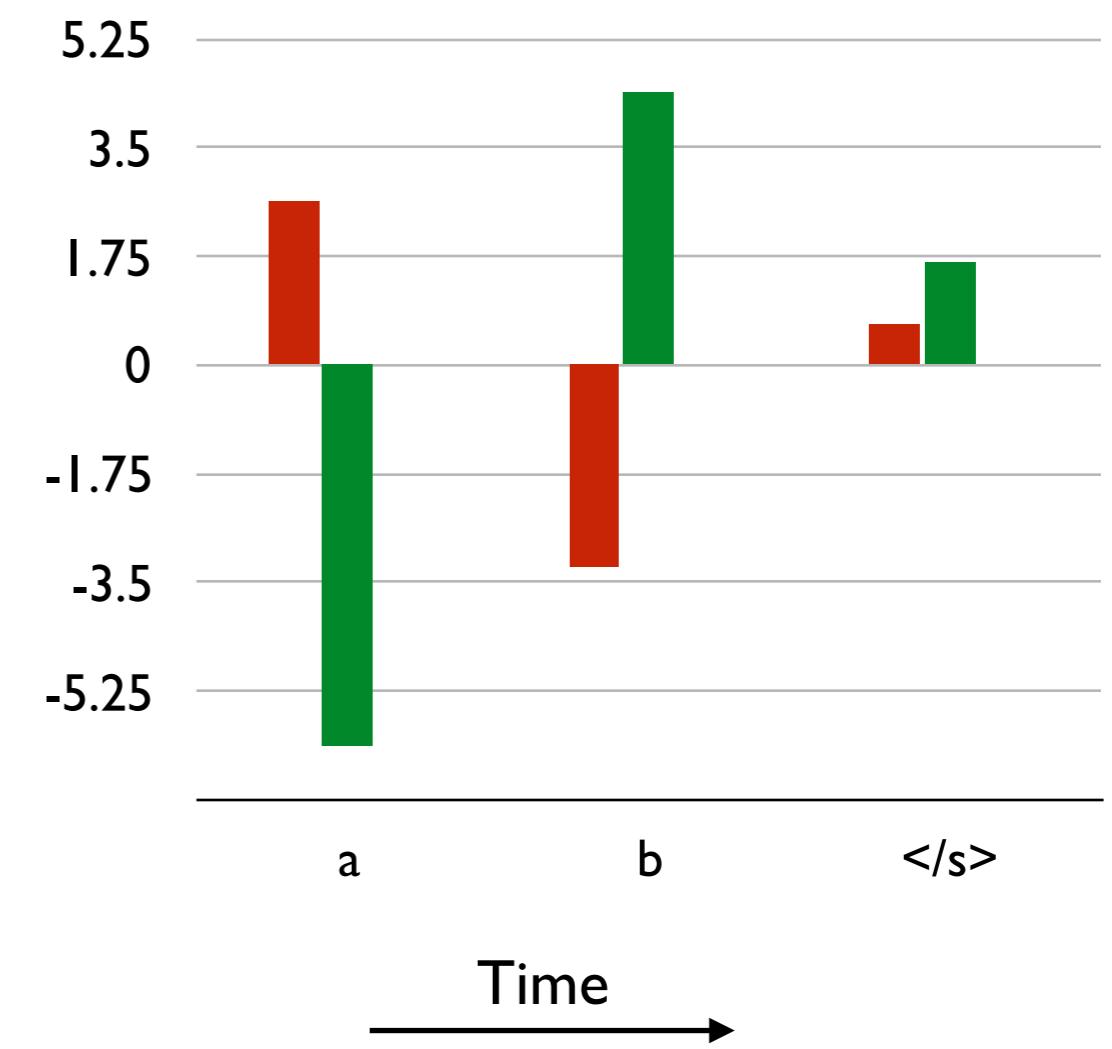
'a' and 'b' have flipped embeddings

Counting a or b

Counting a



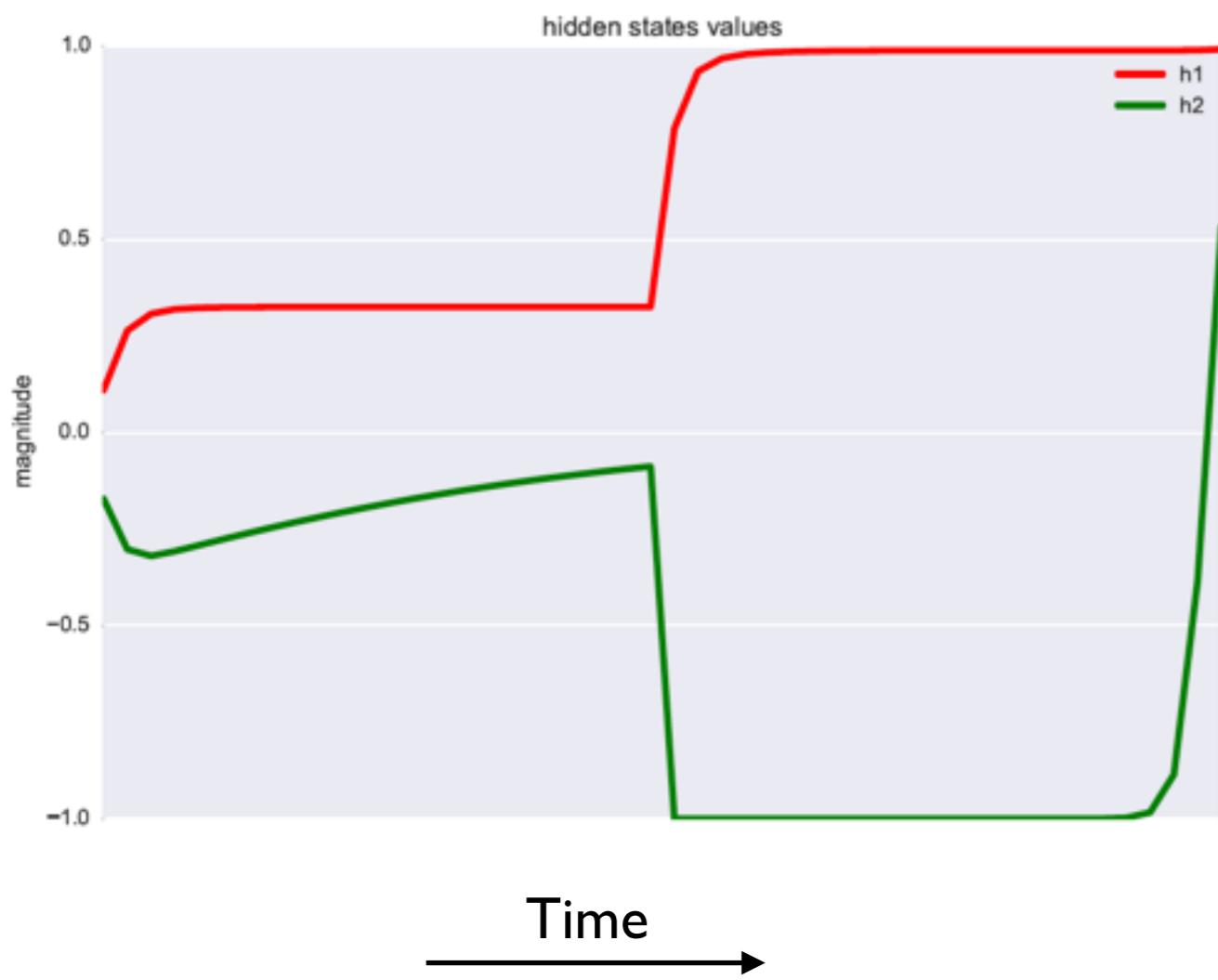
Output embeddings



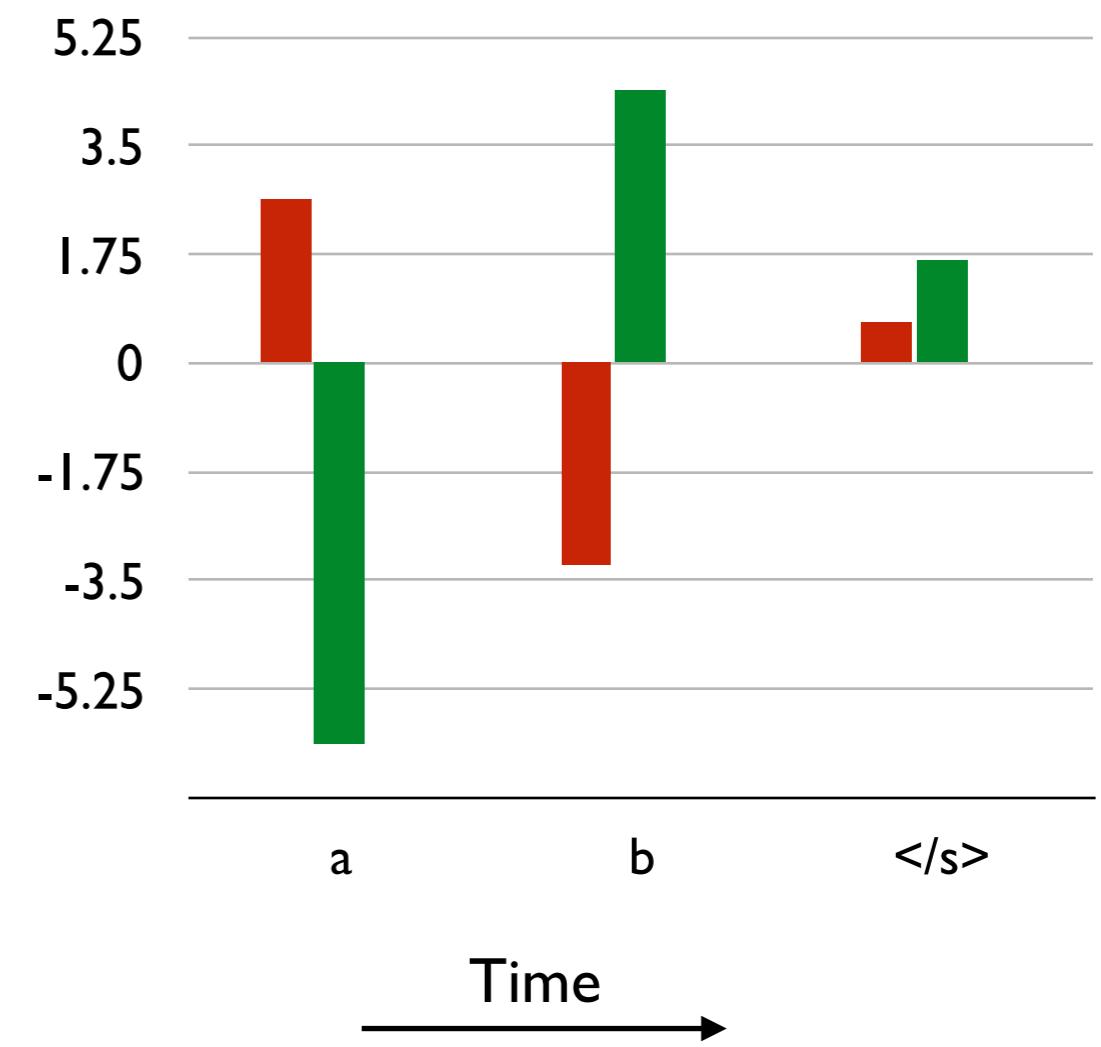
Counting a or b

$$p(\text{symbol}) \propto \text{embedding}(\text{symbol}) h^T$$

Counting a



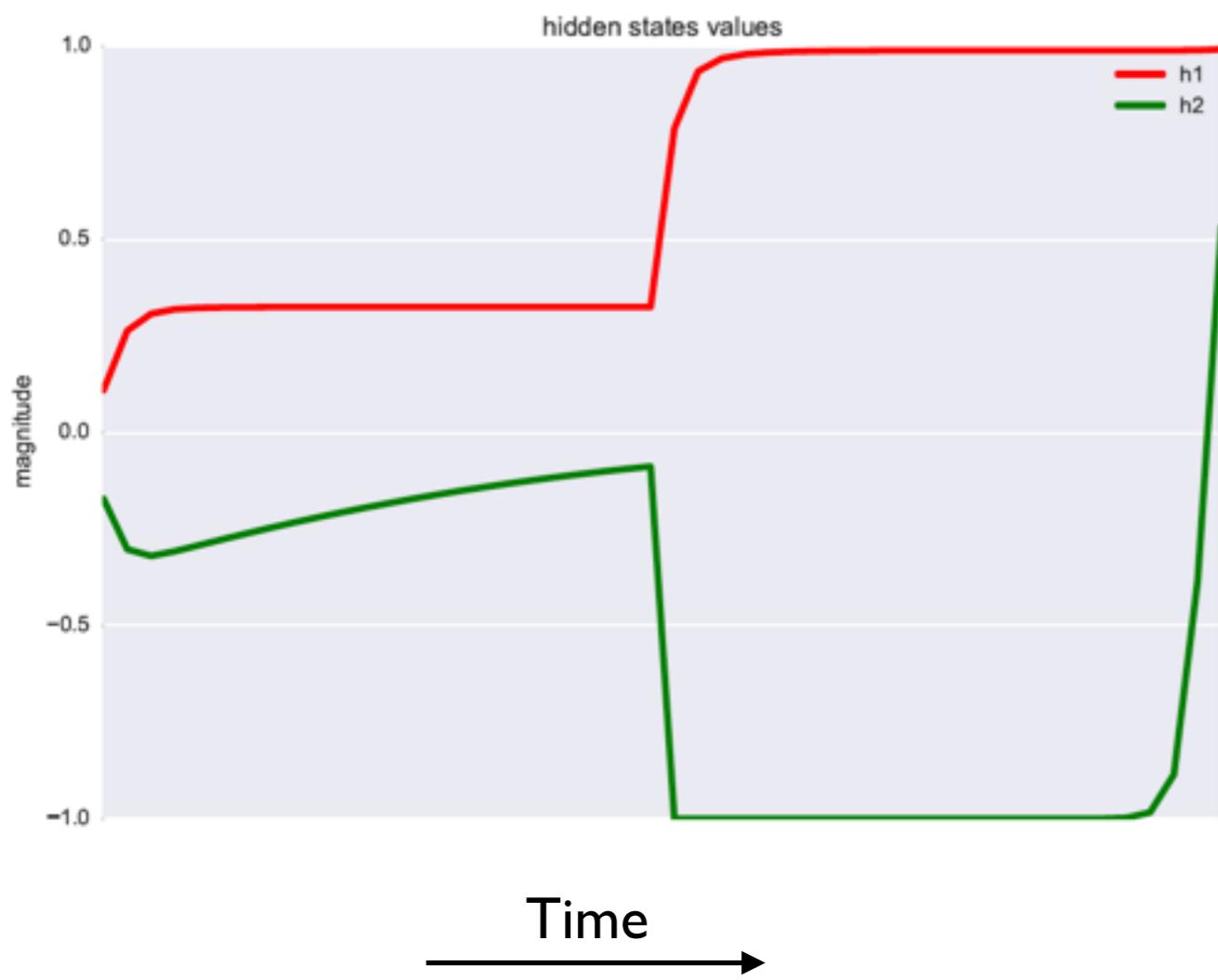
Output embeddings



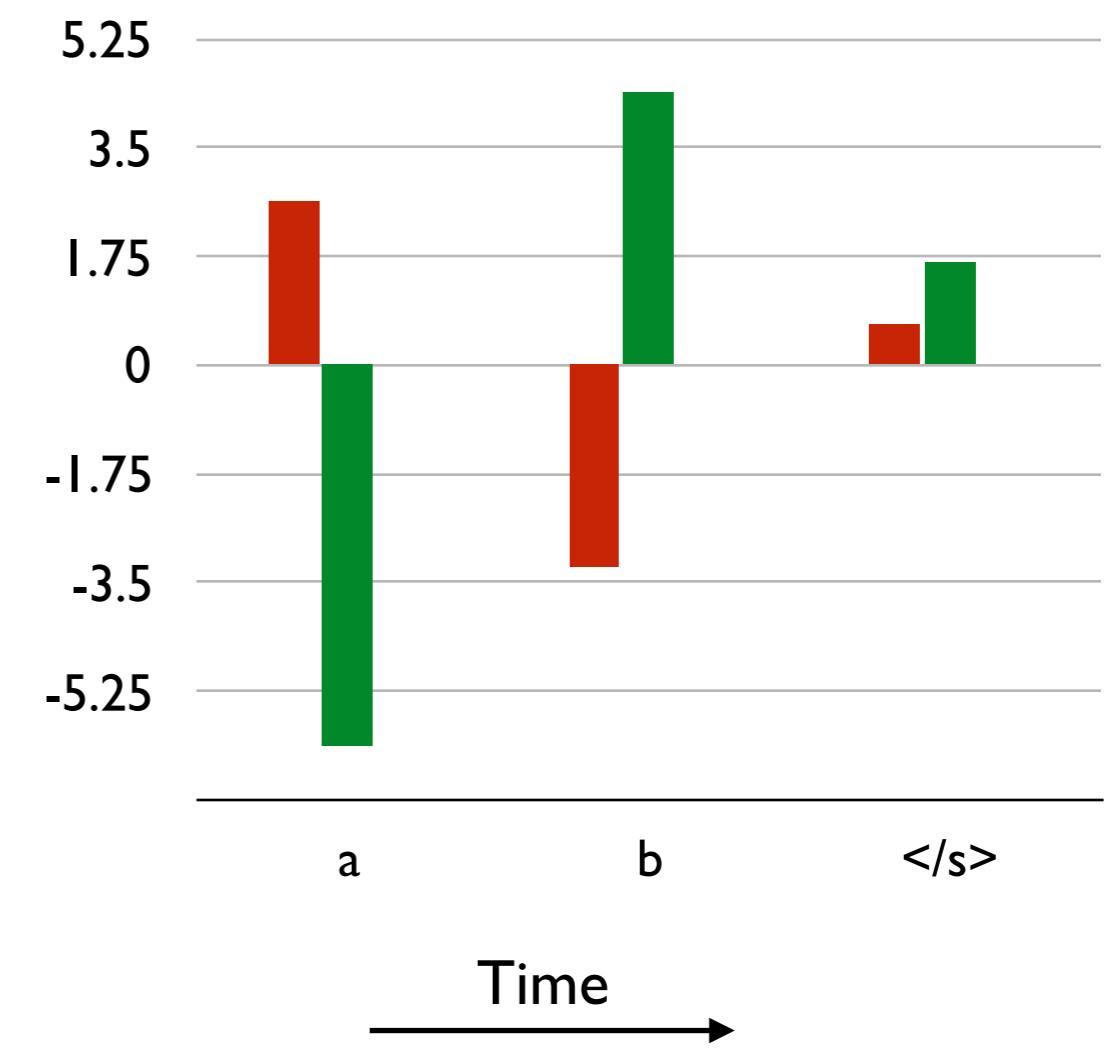
Counting a or b

Counting a

'a' region



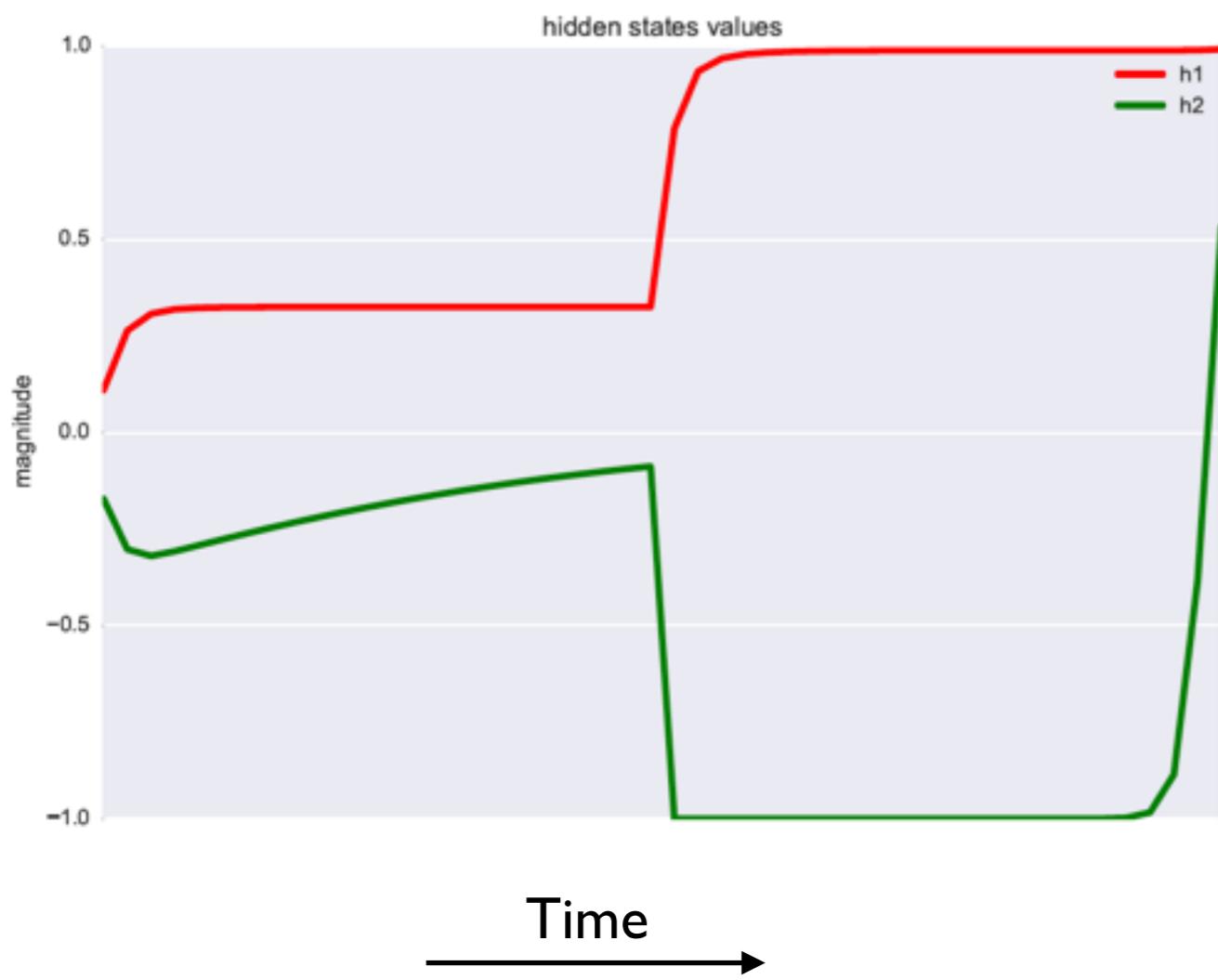
Output embeddings



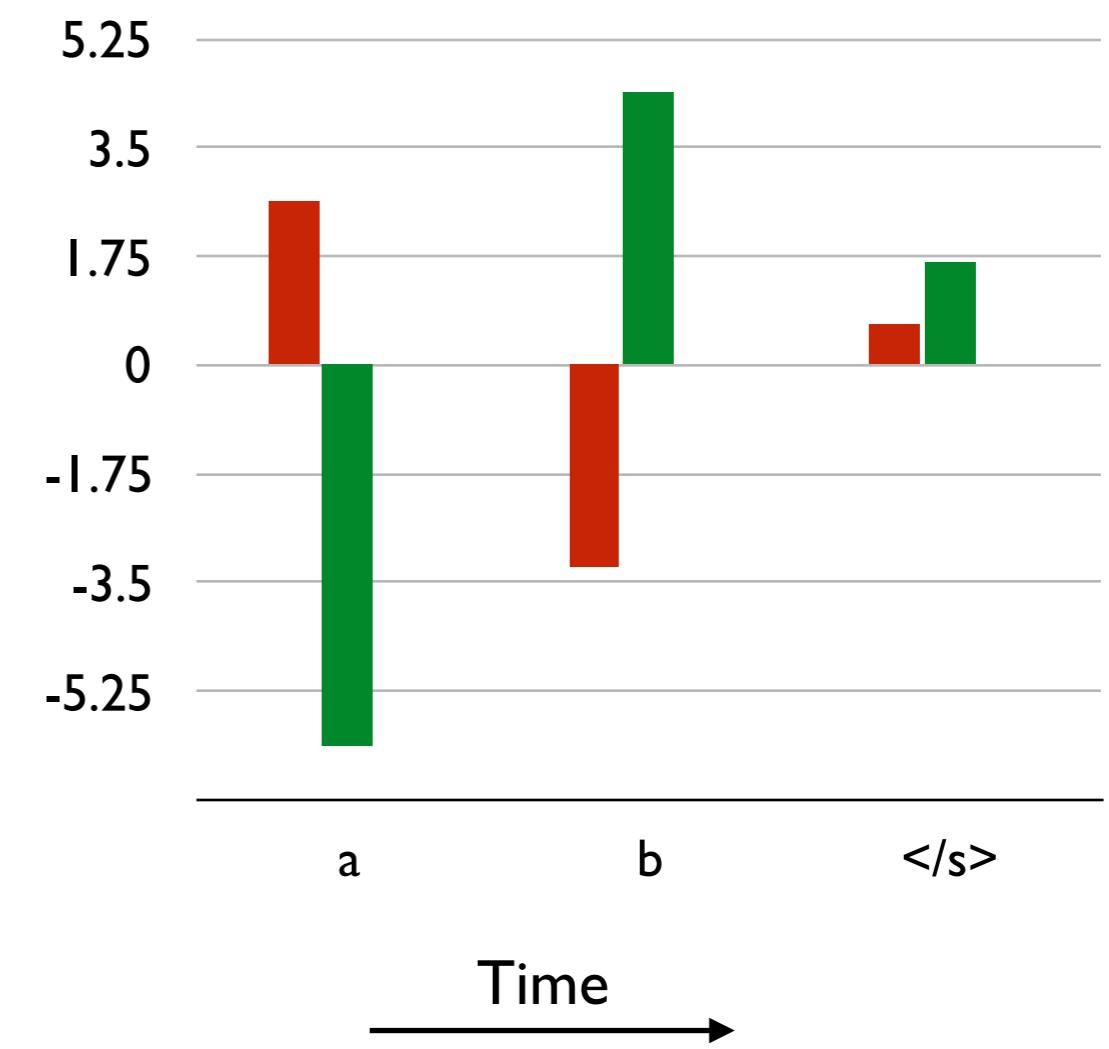
Counting a or b

Counting a

'a' region </s>

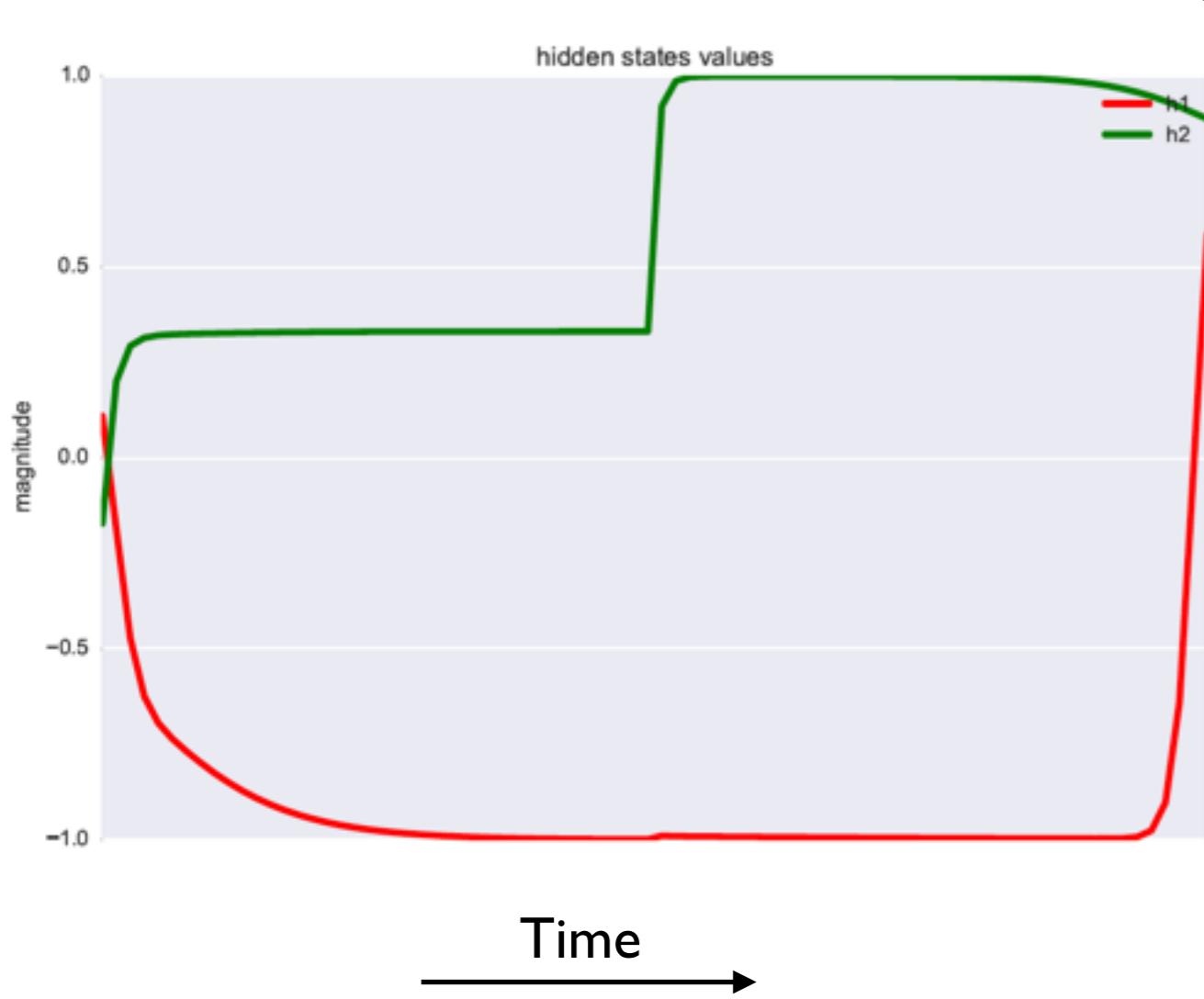


Output embeddings

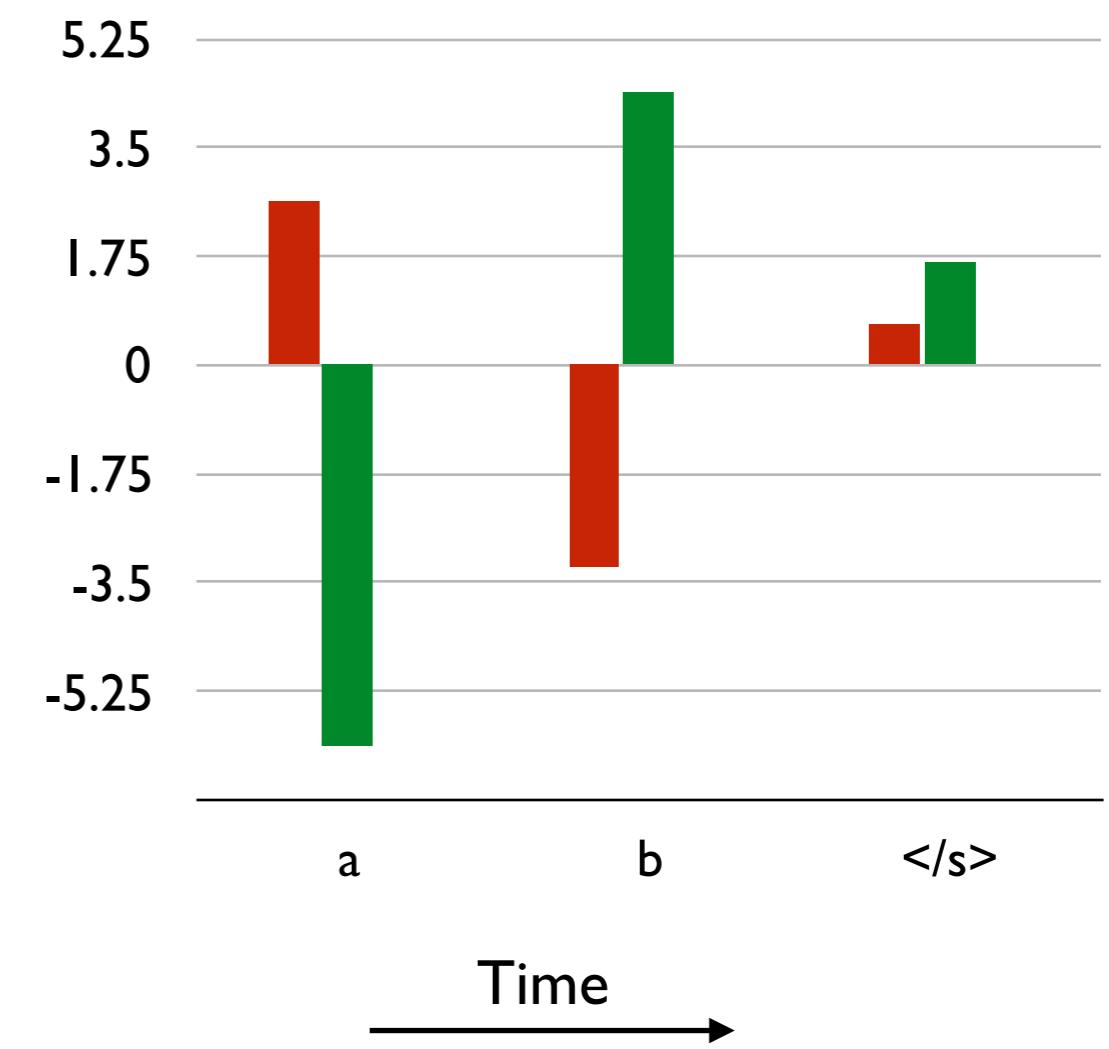


Counting a or b

Counting b



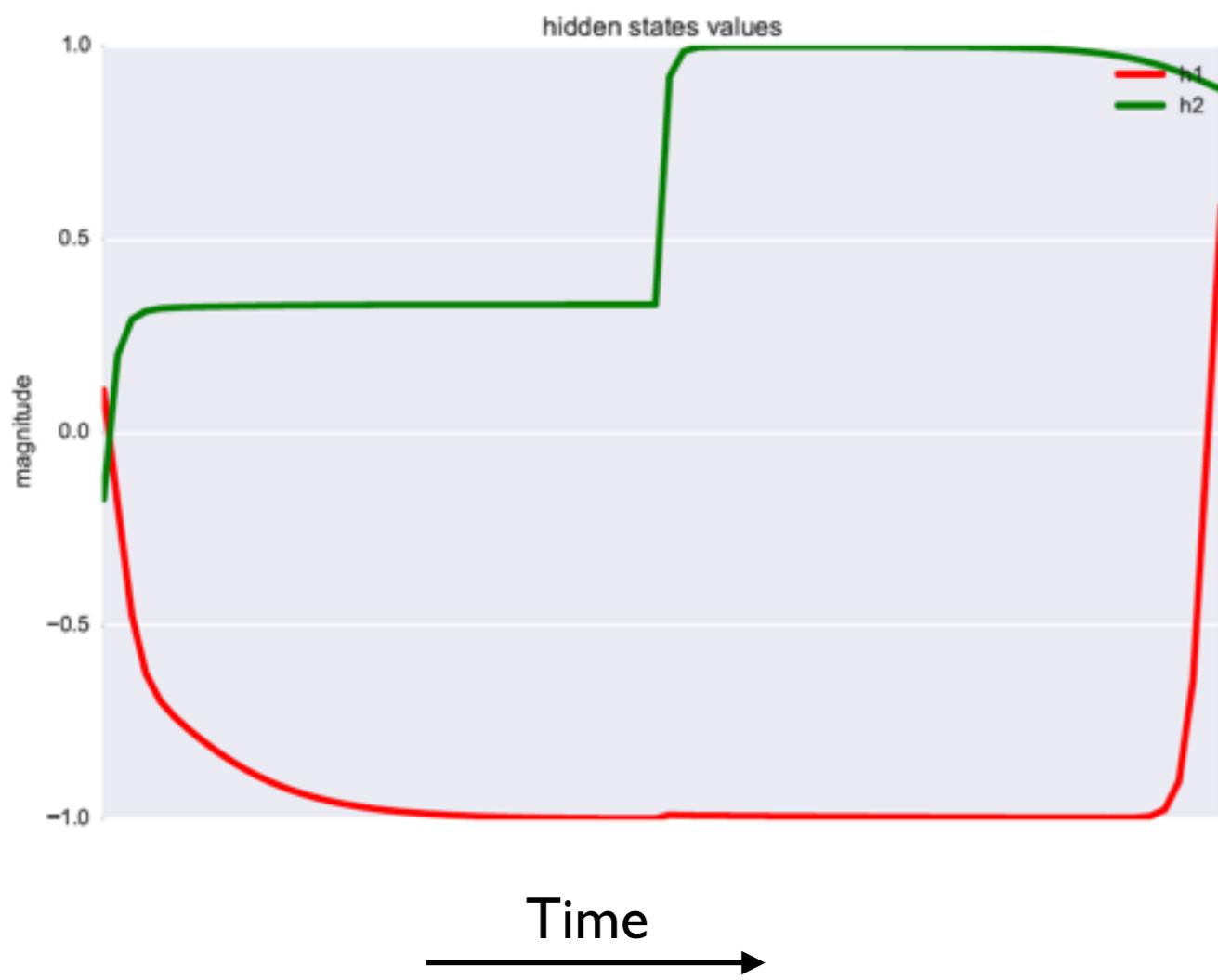
Output embeddings



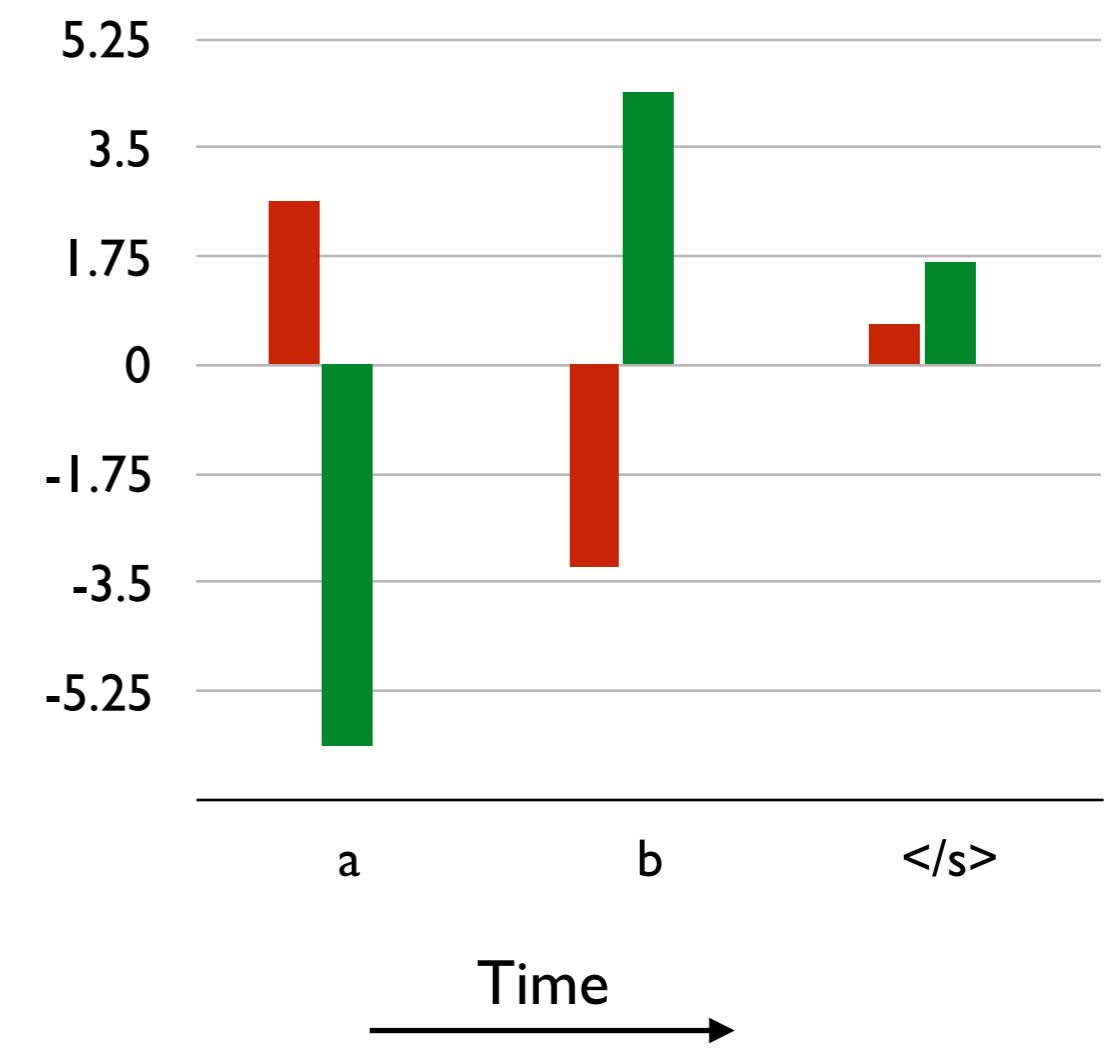
Counting a or b

Counting b

'b' region </s>



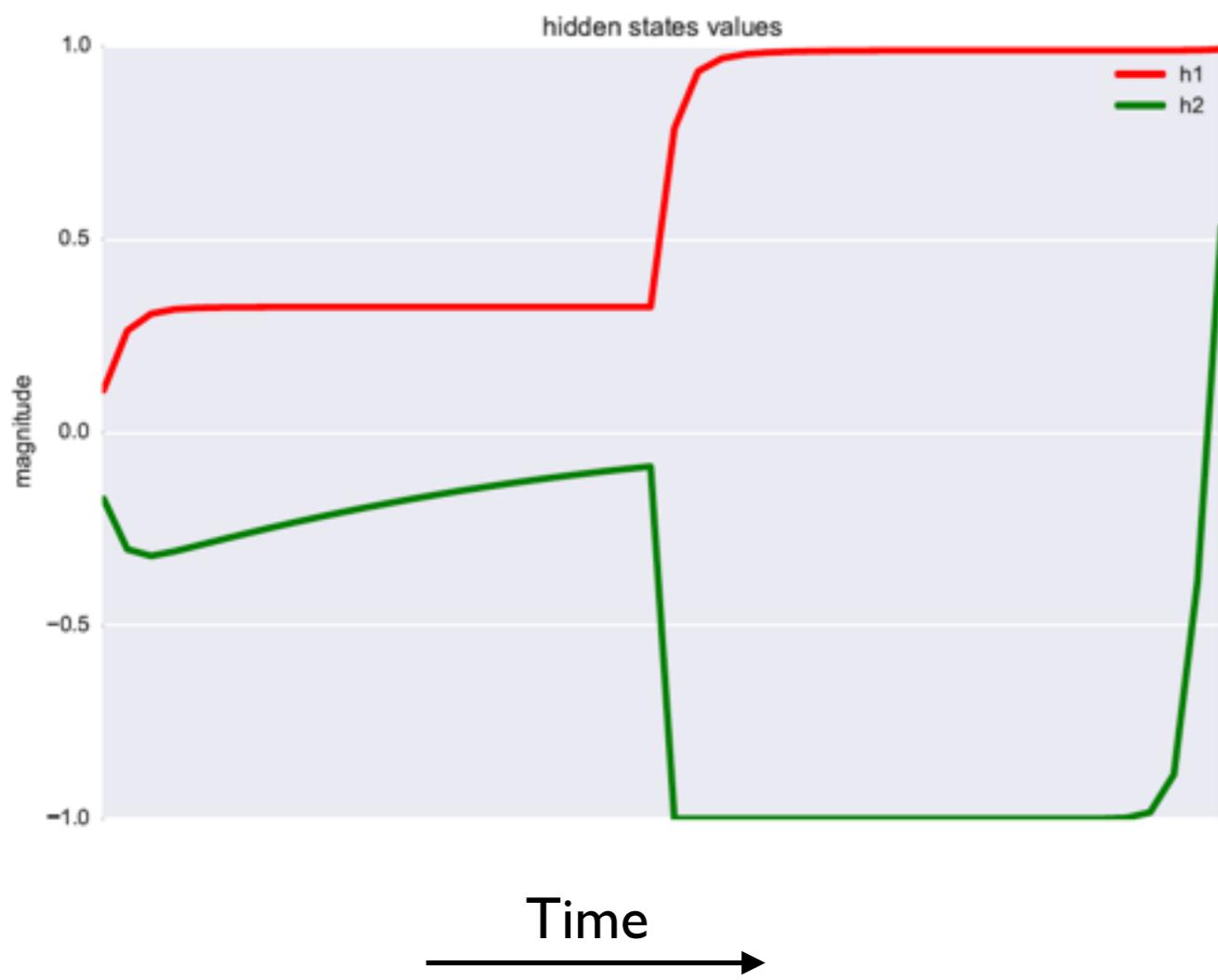
Output embeddings



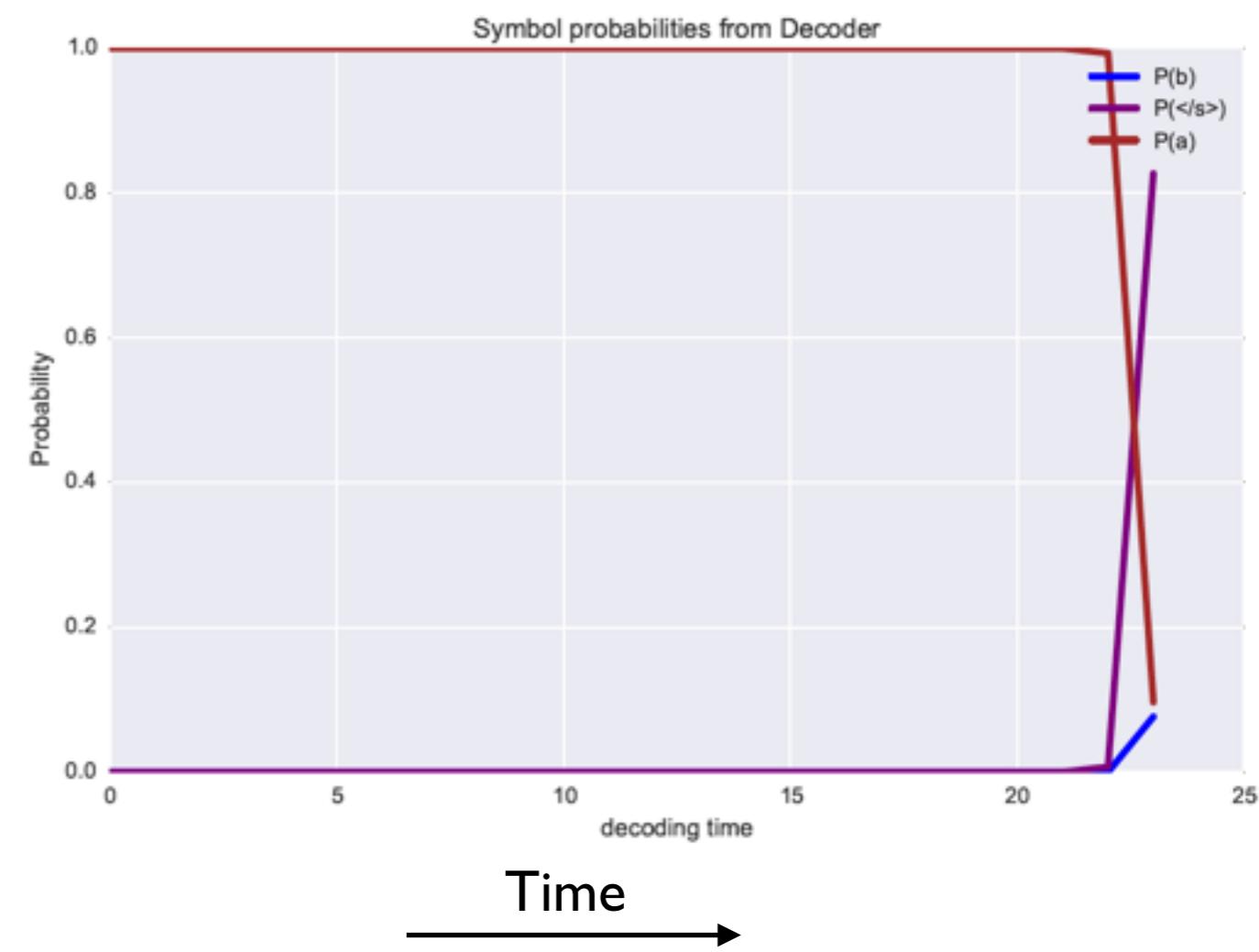
Counting a or b

Counting a

'a' region </s>



'a' region </s>



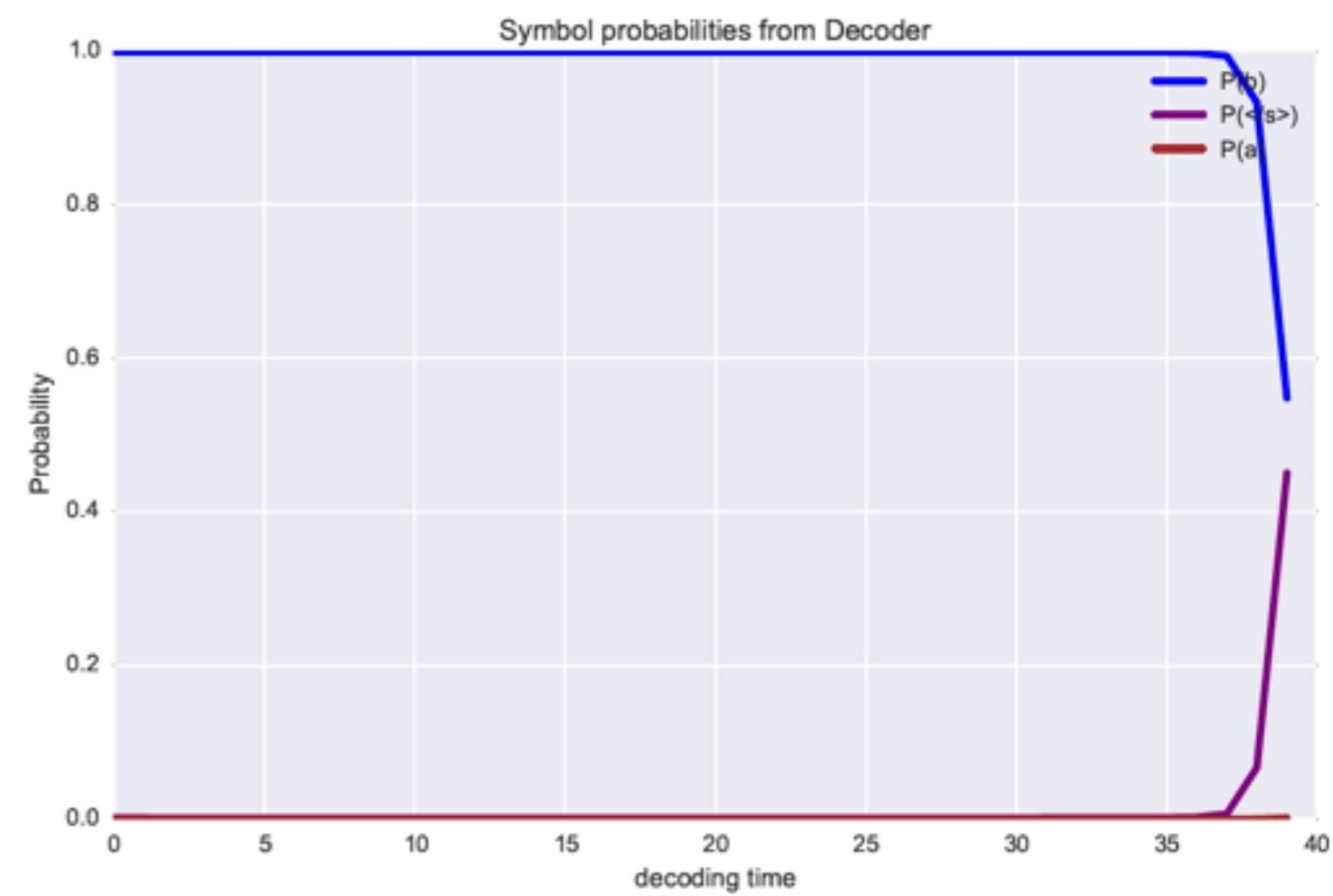
Counting a or b

Counting b

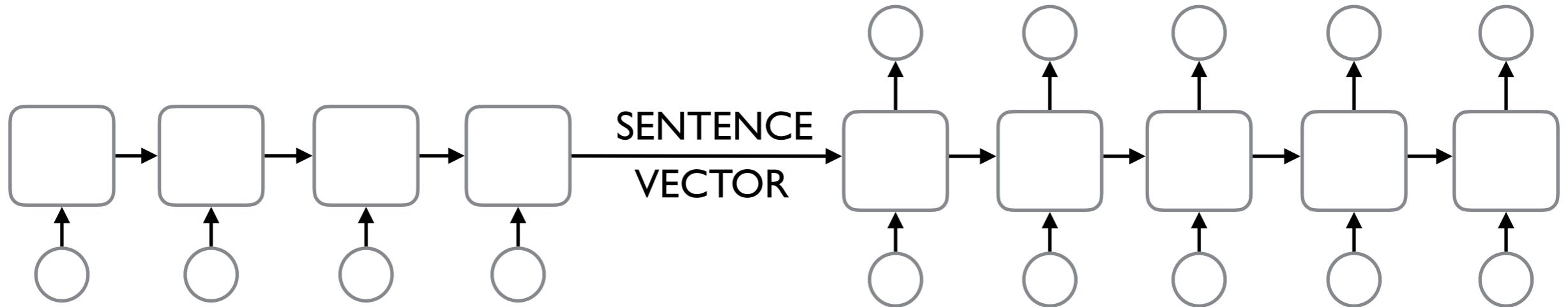
'b' region </s>



'b' region </s>



Counting $\log(a)$



Input:

a a a a a

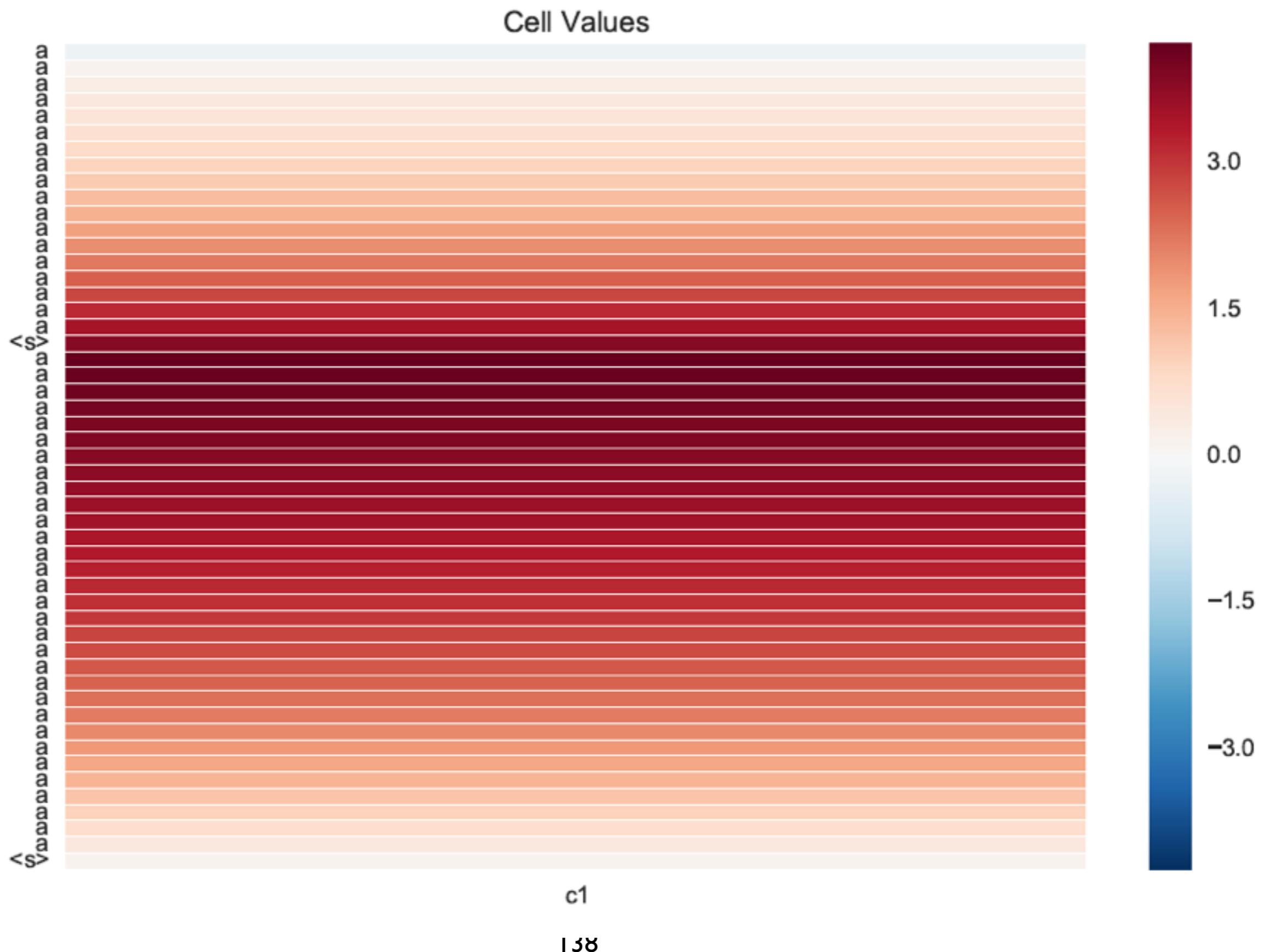
a a a a a a a a a

Output:

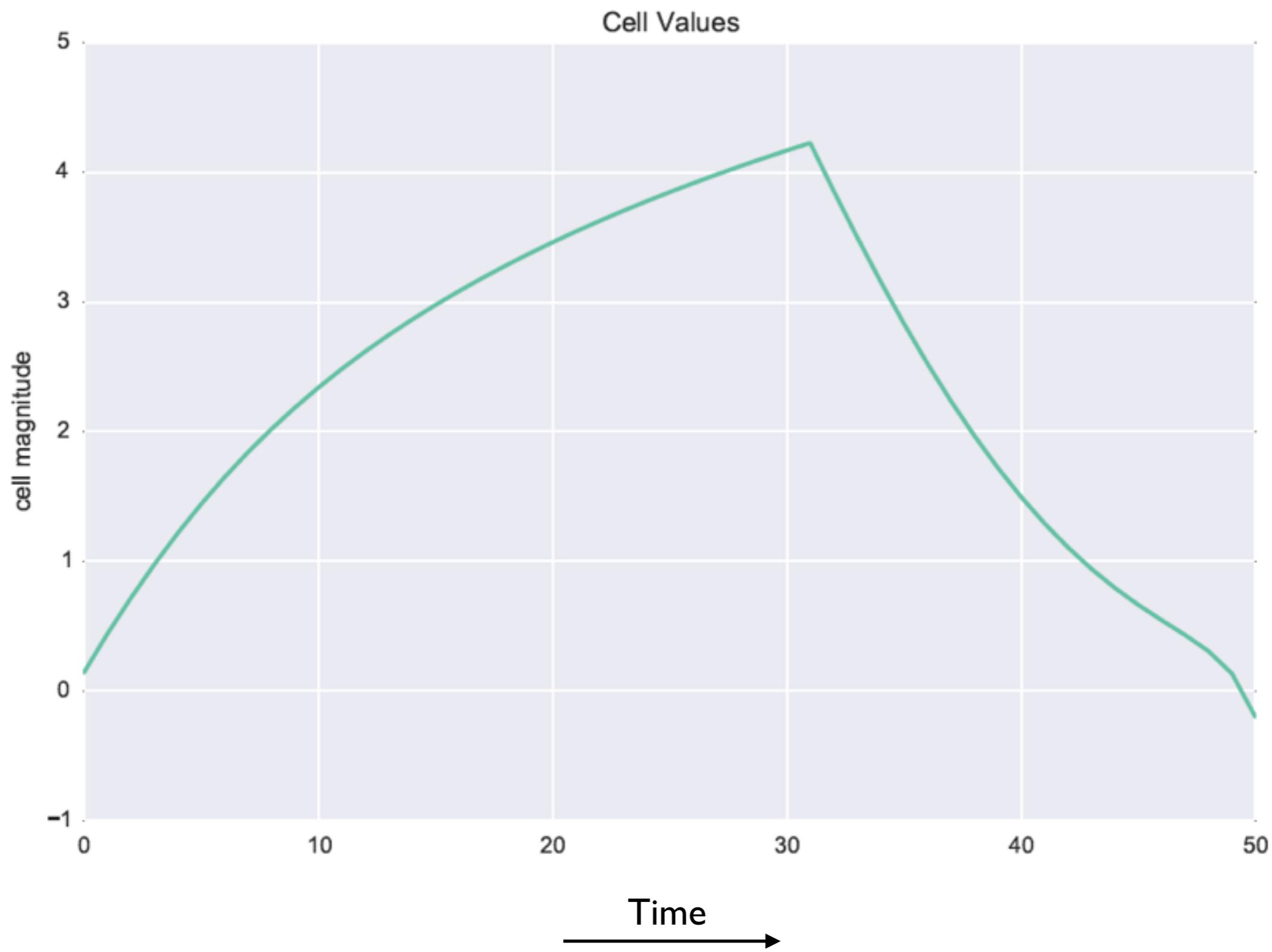
$5 * \log(5)$ number of a's

$5 * \log(9)$ number of a's

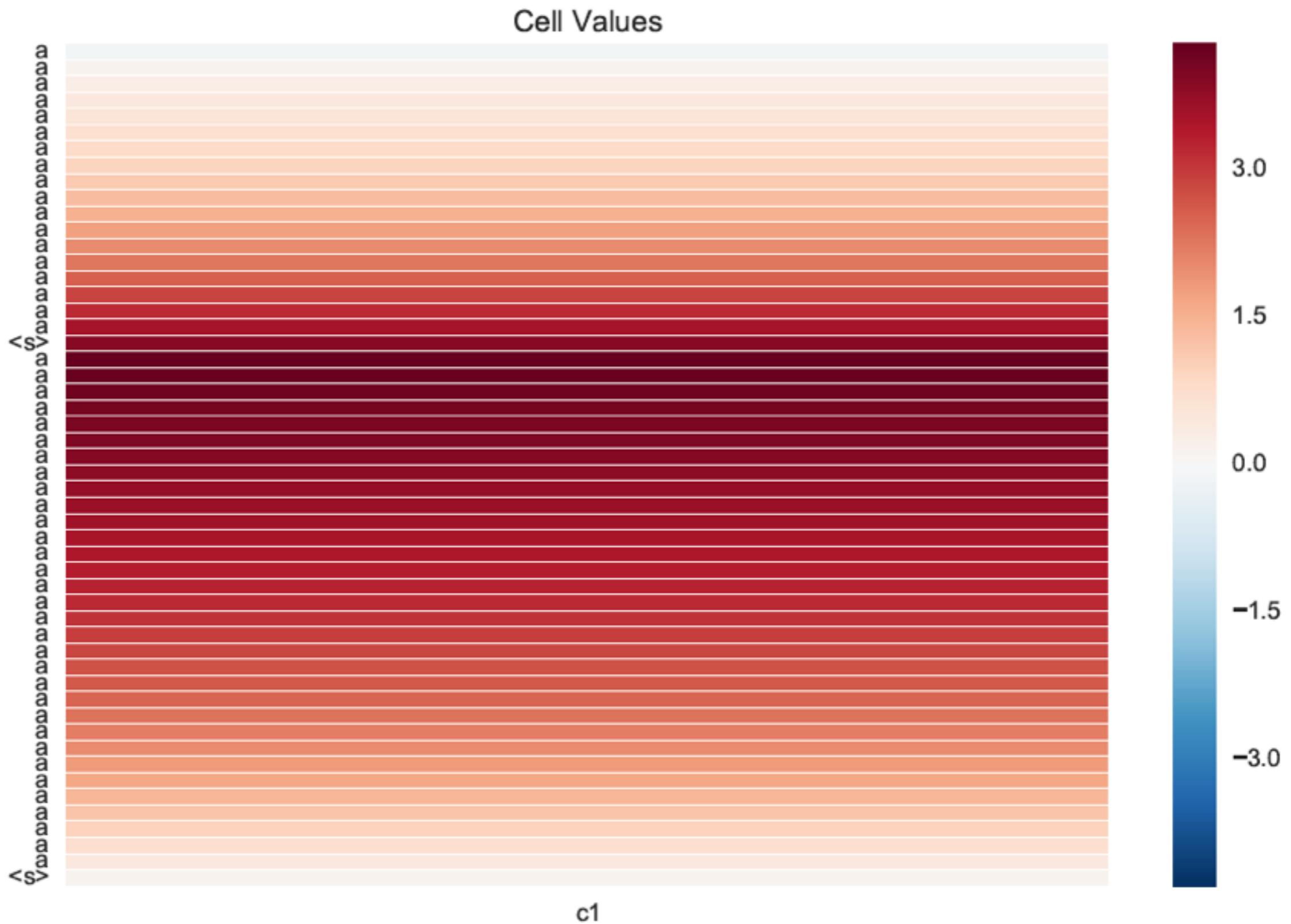
Counting log(a)



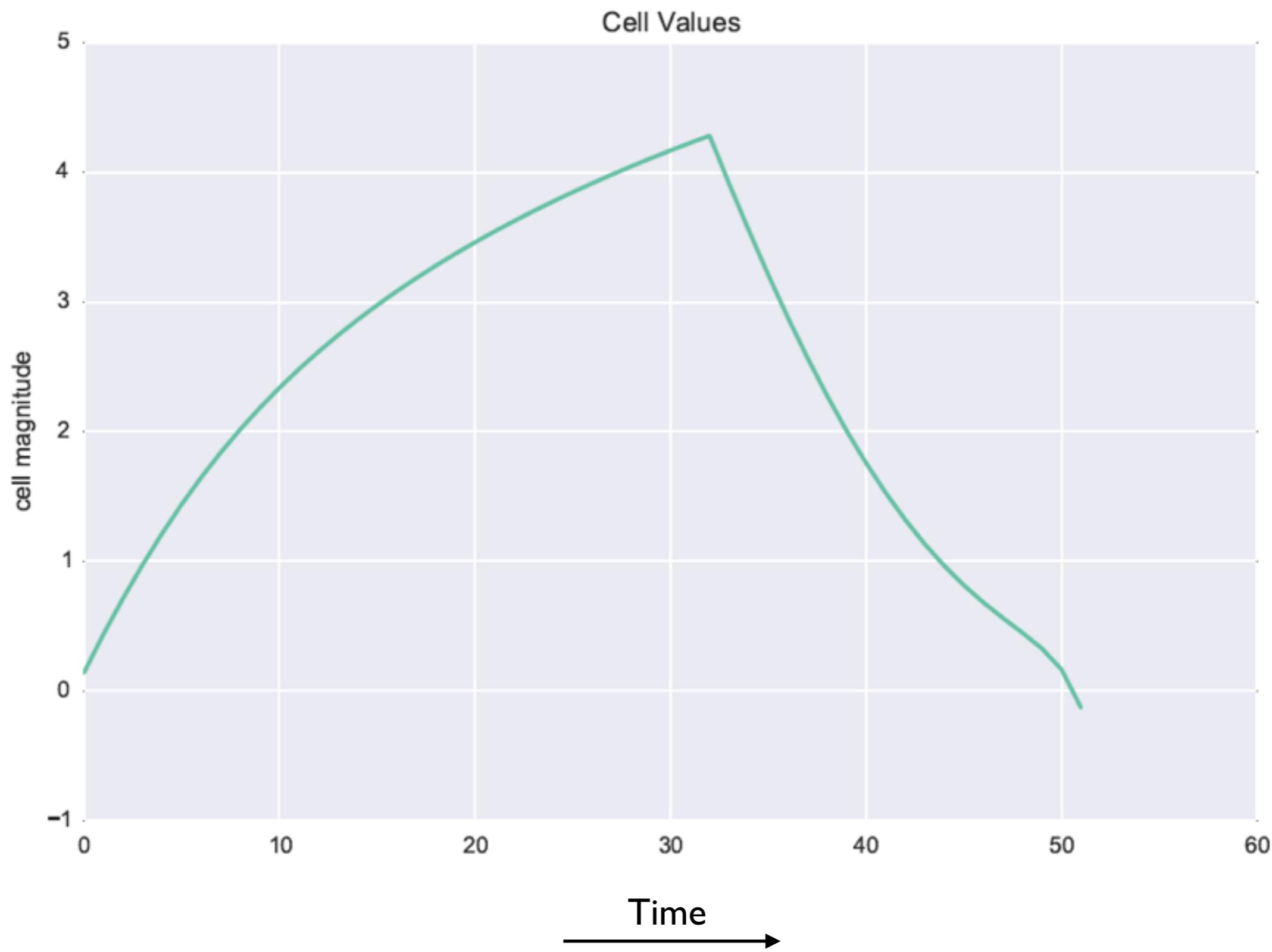
Counting $\log(a)$



Counting log(a)



Counting $\log(a)$



Downloadable Tools

Downloadable Tools

<http://www.speech.sri.com/projects/srilm/>

SRILM - The SRI Language Modeling Toolkit

SRILM is a toolkit for building and applying statistical language models (LMs), primarily for use in speech recognition, statistical tagging and segmentation, and machine translation. It has been under development in the [SRI Speech Technology and Research Laboratory](#) since 1995. The toolkit has also greatly benefitted from its use and enhancements during the [Johns Hopkins University/CLSP summer workshops](#) in 1995, 1996, 1997, and 2002 (see [history](#)).

These pages and the software itself assume that you know what statistical language modeling is. To learn about language modeling we recommend the textbooks

- [Speech and Language Processing](#) by Dan Jurafsky and Jim Martin (chapter 6 in the 1st edition, chapter 4 in the 2nd edition)
- [Foundations of Statistical Natural Language Processing](#) by Chris Manning and Hinrich Schütze (chapter 6).

Either book gives an excellent introduction to N-gram language modeling, which is the main type of LM supported by SRILM.

SRILM consists of the following components:

- A set of C++ class libraries implementing language models, supporting data structures and miscellaneous utility functions.
- A set of executable programs built on top of these libraries to perform standard tasks such as training LMs and testing them on data, tagging or segmenting text, etc.
- A collection of miscellaneous scripts facilitating minor related tasks.

SRILM runs on UNIX and Windows platforms.

SRILM has been used in a great variety of statistical modeling [applications](#).

Others have published [extensions](#) to SRILM that add new functionality.

Documentation

SRILM is still under development. The documentation in particular is work in progress. Best documented are the executable programs, scripts, and file formats, in the form of UNIX-style [manual pages](#). The libraries are documented mostly in the source code. An overview of what the software can do and its design philosophy can be found in the paper "SRILM - An Extensible Language Modeling Toolkit", in Proc. Intl. Conf. Spoken Language Processing, Denver, Colorado, September 2002 ([postscript](#), [PDF](#)). Links to other papers and tutorials, as well as frequently asked questions, are also given [here](#).

NEW A [recent paper](#) summarizes updates to SRILM since the 2002 paper.

Feed Forward Neural Language Model

<http://nlg.isi.edu/software/nplm/>

Neural Probabilistic Language Model Toolkit

NPLM is a toolkit for training and using feedforward neural language models (Bengio, 2003). It is fast even for large vocabularies (100k or more): a model can be trained on a billion words of data in about a week, and can be queried in about 40 μ s, which is usable inside a decoder for machine translation.

NPLM is written by [Ashish Vaswani](#), with contributions from [David Chiang](#) and [Victoria Fossum](#). It is distributed under the MIT open-source license.

- Latest stable version: [nplm-0.3.tar.gz](#)
- Paper: Decoding with large-scale neural language models improves translation. Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang, 2013. In *Proceedings of EMNLP*. [\[PDF\]](#)

Yandex: Training RNNs with NCE

<https://github.com/yandex/faster-rnnlm>

Faster RNNLM (HS/NCE) toolkit

In a nutshell, the goal of this project is to create an rnnlm implementation that can be trained on huge datasets (several billions of words) and very large vocabularies (several hundred thousands) and used in real-world ASR and MT problems. Besides, to achieve better results this implementation supports such praised setups as ReLU+DiagonalInitialization [1], GRU [2], NCE [3], and RMSProp [4].

How fast is it? Well, on One Billion Word Benchmark [8] and 3.3GHz CPU the program with standard parameters (sigmoid hidden layer of size 256 and hierarchical softmax) processes more than 250k words per second in 8 threads, i.e. 15 millions of words per minute. As a result an epoch takes less than one hour. Check [Experiments section](#) for more numbers and figures.

The distribution includes `./run_benchmark.sh` script to compare training speed on your machine among several implementations. The script downloads Penn Tree Bank corpus and trains four models: Mikolov's rnnlm with class-based softmax from rnnlm.org, Edrenkin's rnnlm with HS from Kaldi project, faster-rnnlm with hierarchical softmax, and faster-rnnlm with noise contrastive estimation. Note that while models with class-based softmax can achieve a little lower entropy than models hierarchical softmax, their training is infeasible for large vocabularies. On the other hand, NCE speed doesn't depend on the size of the vocabulary. What's more, models trained with NCE is comparable with class-based models in terms of resulting entropy.

Training LSTMs

<http://nlg.isi.edu/software/EUREKA.tar.gz>

NCE based training

https://github.com/isi-nlp/Zoph_RNN

GPU based training

Training LSTMs

<http://nlg.isi.edu/software/EUREKA.tar.gz>

NCE based training

https://github.com/isi-nlp/Zoph_RNN

GPU based training

RNNLM Toolkit

<http://rnnlm.org/>

Python Easy Neural Network Extruder

This is a library that tries to make creating neural networks as easy as possible by being as similar to Python/NumPy as possible. It borrows heavily from [Theano](#) and [CNN](#). If you find it useful, or want to help improve it, please let me (David Chiang) know!

You need to have NumPy (tested with 1.9).

- Having SciPy (tested with 0.14) helps.
- There is also experimental GPU support, which requires the development version of [libgpuarray](#).

The library is in pure Python and doesn't need to be built. Just make sure that the `penne` package directory (the one containing `__init__.py`) is in your Python path and import the package (note, don't `import * from` both `numpy` and `penne`, as they have many symbols in common):

[Basic examples](#) - very useful for quick introduction (training, evaluation, hyperparameter selection, simple n-best list rescoring, etc.) - 35MB

[Advanced examples](#) - includes large scale experiments with speech lattices (n-best list rescoring, ...) - 235MB, by Stefan Kombrink

Slides from my presentation at Google - [pdf](#)

RNNLM is now integrated into Kaldi toolkit! Check [this](#).

Penne

<https://bitbucket.org/ndnlp/penne>

Python Easy Neural Network Extruder

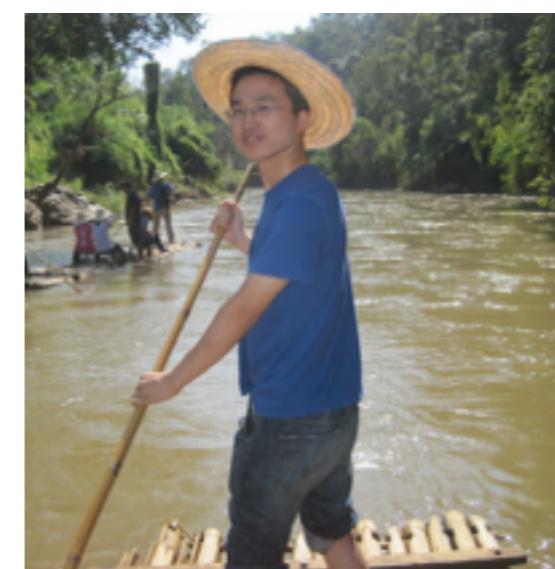
This is a library that tries to make creating neural networks as easy as possible by being as similar to Python/NumPy as possible. It borrows heavily from [Theano](#) and [CNN](#). If you find it useful, or want to help improve it, please let me (David Chiang) know!

You need to have NumPy (tested with 1.9).

- Having SciPy (tested with 0.14) helps.
- There is also experimental GPU support, which requires the development version of [libgpuarray](#).

The library is in pure Python and doesn't need to be built. Just make sure that the `penne` package directory (the one containing `__init__.py`) is in your Python path and import the package (note, don't `import * from` both `numpy` and `penne`, as they have many symbols in common):

Contributors and Collaborators



Thanks!