CS/DS 552: Class 14

Jacob Whitehill

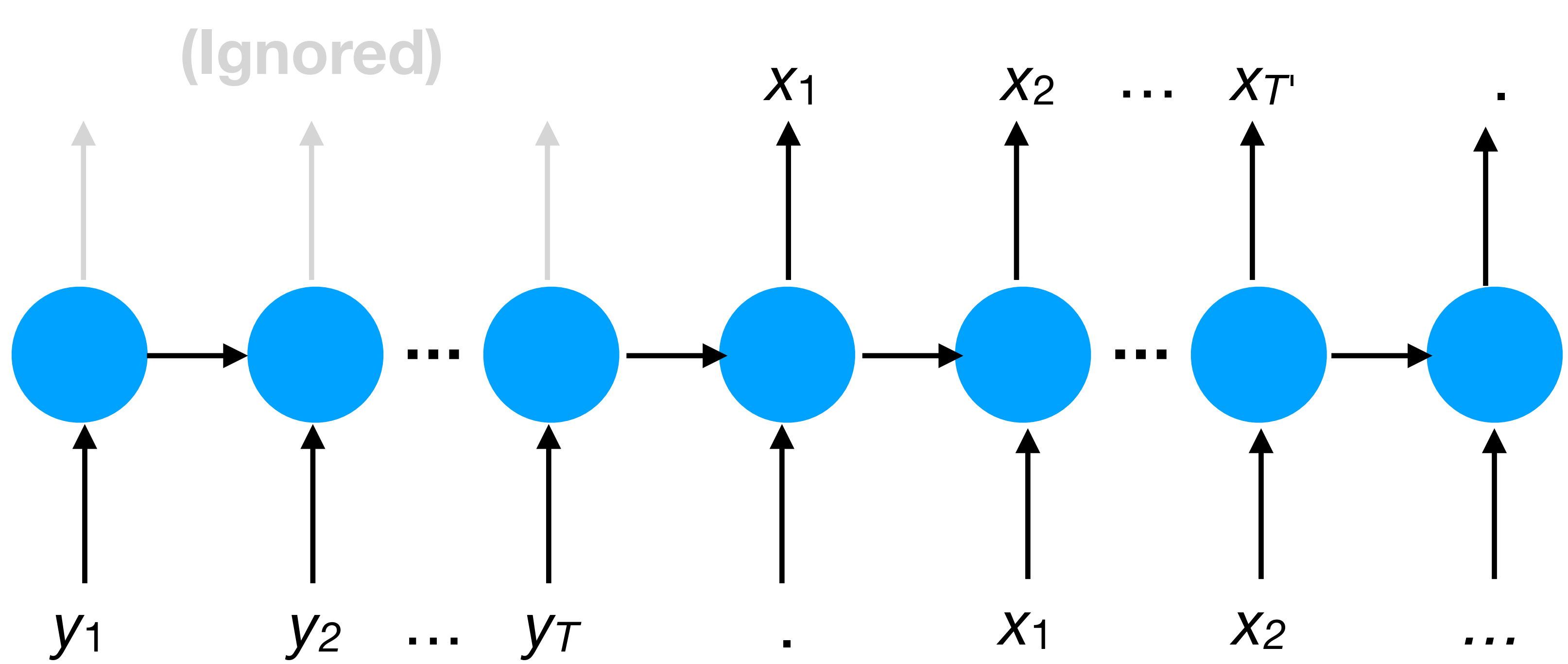
RNNs for machine translation

Machine translation

- Suppose we want to translate from one language to another.
- Language 1 vocabulary: { a, b, . }.
- Language 2 vocabulary: { u, v, w, . }.
- We add to both vocabularies a “.” symbol that means **end-of-sentence** (EOS).

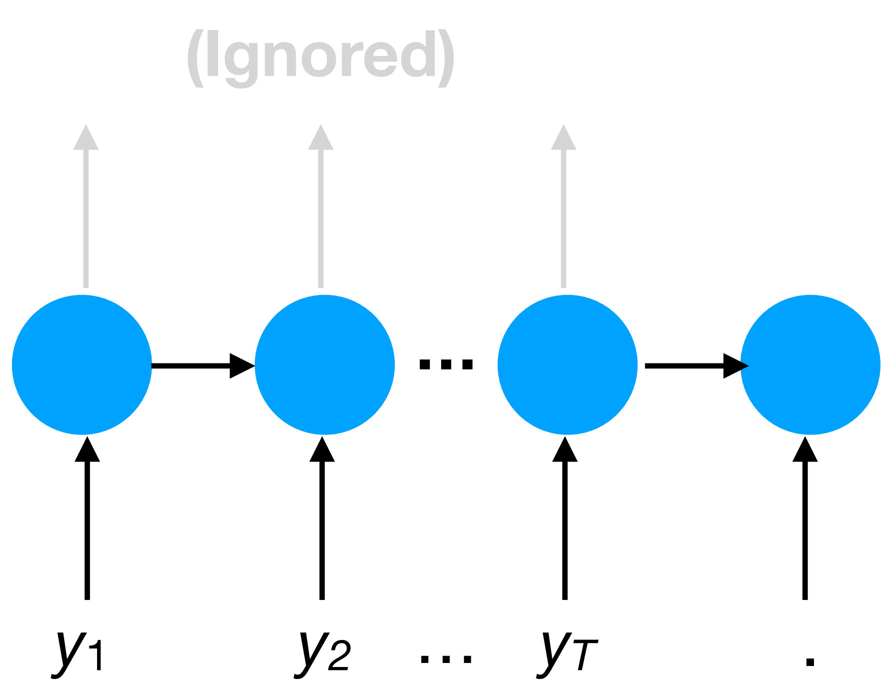
RNNs for machine translation

- We can construct an RNN to translate from the source language to the target language.
- Note that the length T of the input sentence is generally not equal to the length T' of the output sentence; hence, we cannot simply output one x_t for each y_t.



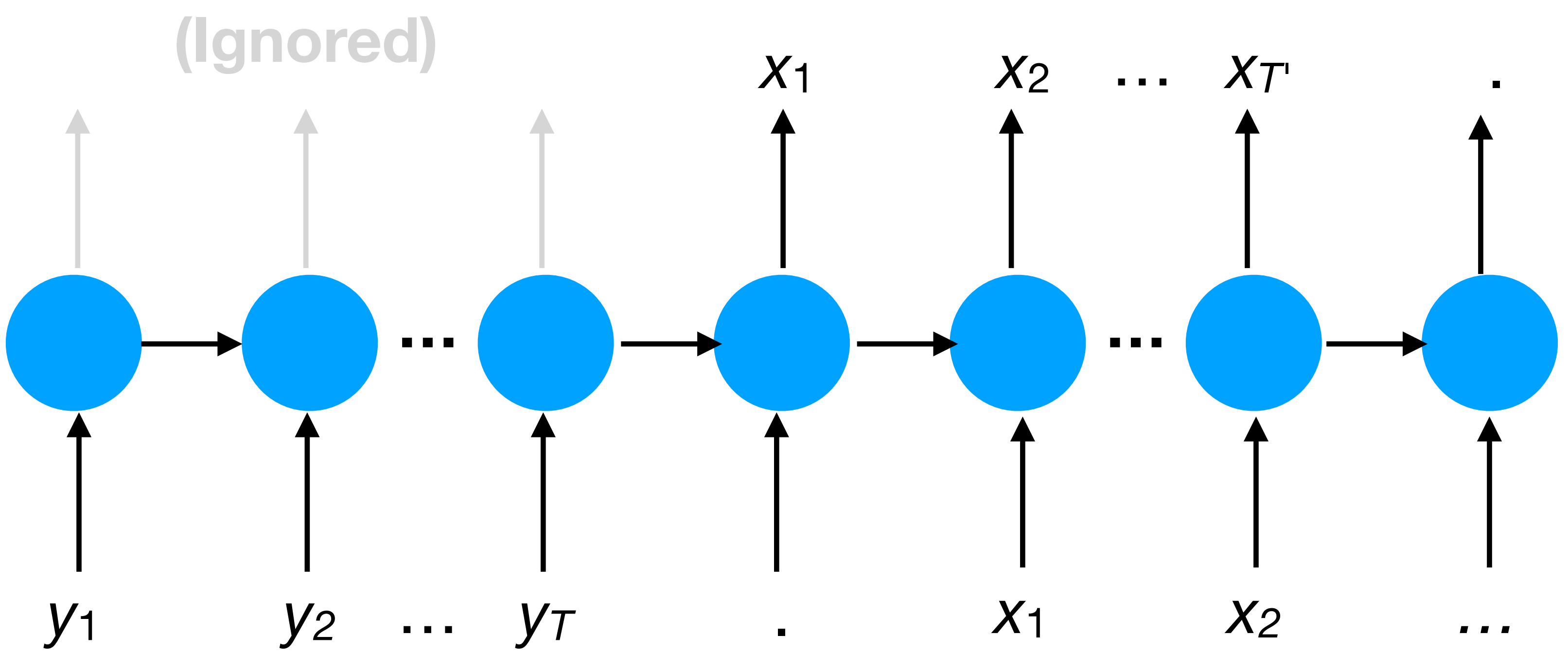
RNNs for machine translation

- We can construct an RNN to translate from the source language to the target language:
  1. We first input the T words of the input sentence as y_1, …, y_T, followed by .



RNNs for machine translation

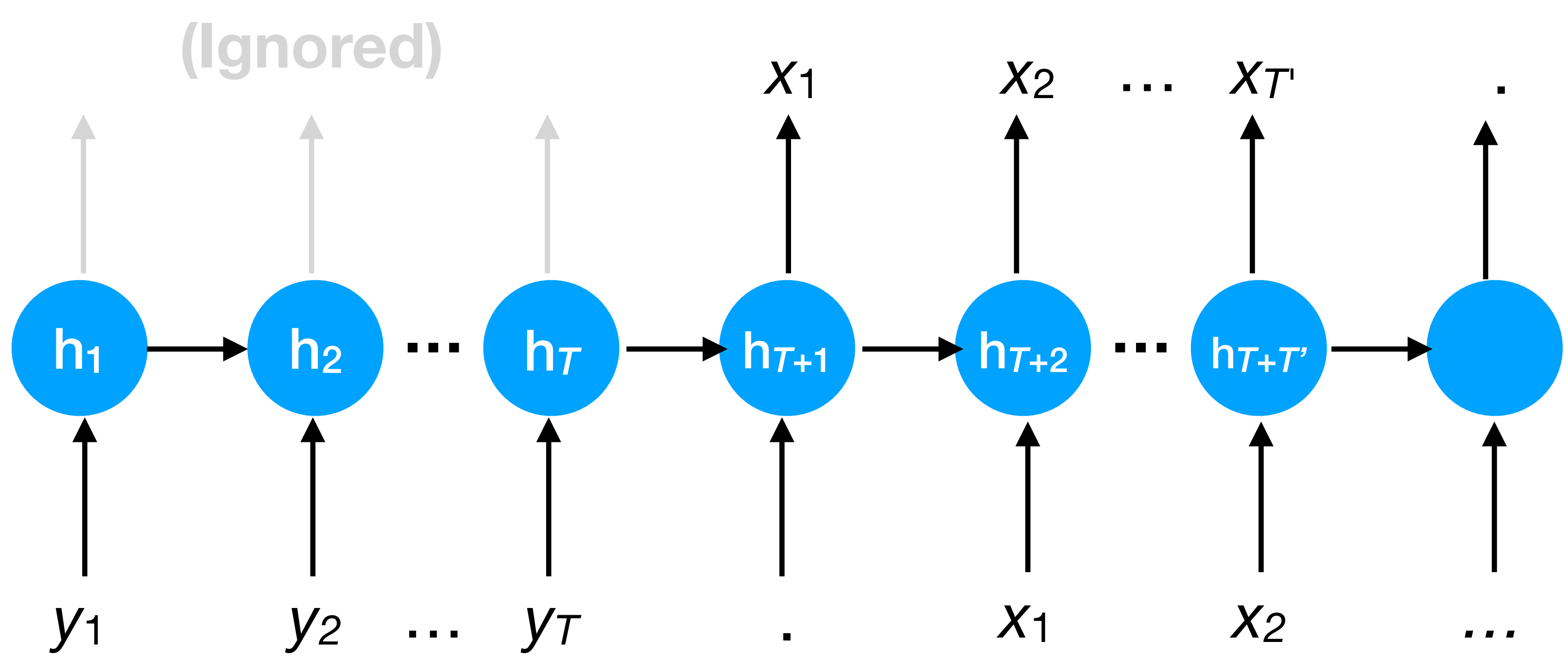
- We can construct an RNN to translate from the source language to the target language:
  1. We first input the T words of the input sentence as y_1, …, y_T, followed by .
  2. We then obtain the T' words of the output sentence autoregressively as x_1, …, $x_{T'}$, until the model outputs .



RNNs for machine translation

- The RNN's hidden state $\mathbf{h}_t$ captures both the **meaning of the input y** and the **summary of the output up to time *t***.

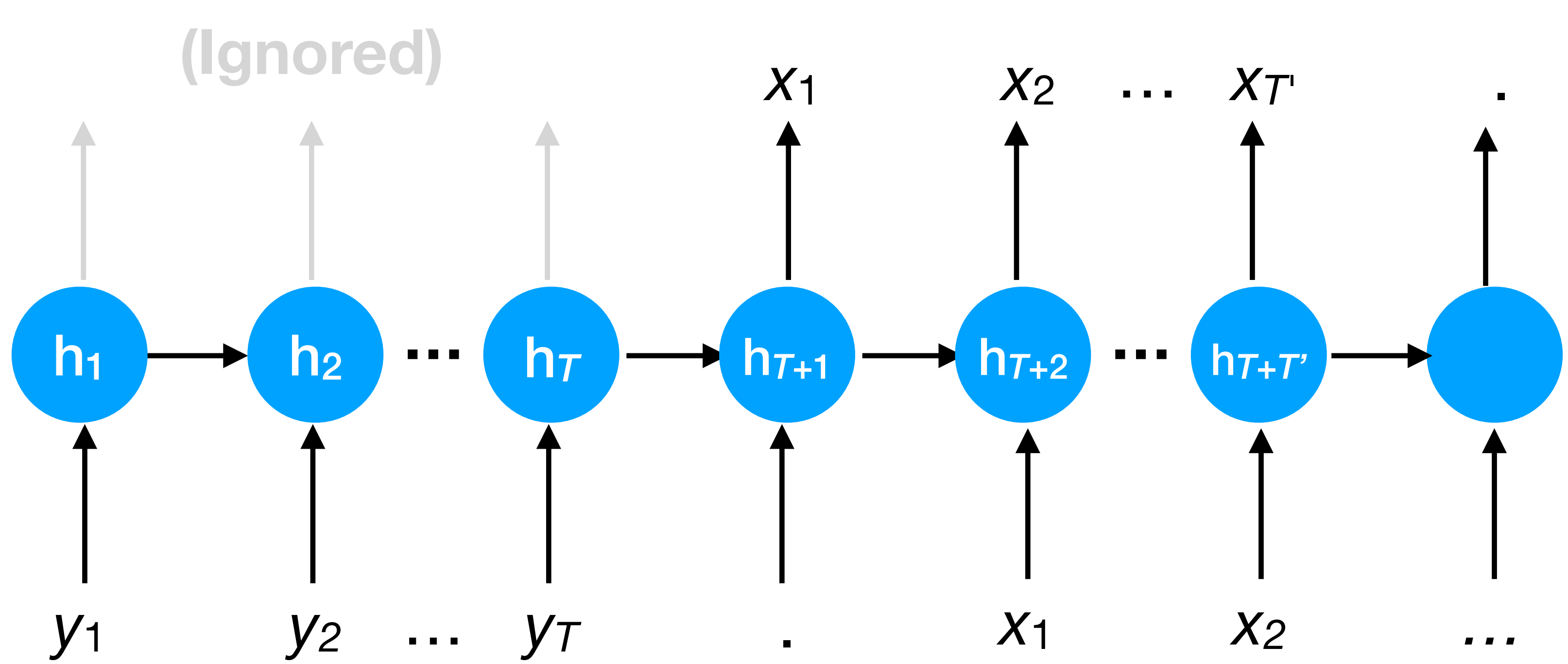
- $\mathbf{h}_t$ effectively "compresses" the variable-length history into a fixed-length representation (i.e., $O(1)$ space).



RNNs for machine translation

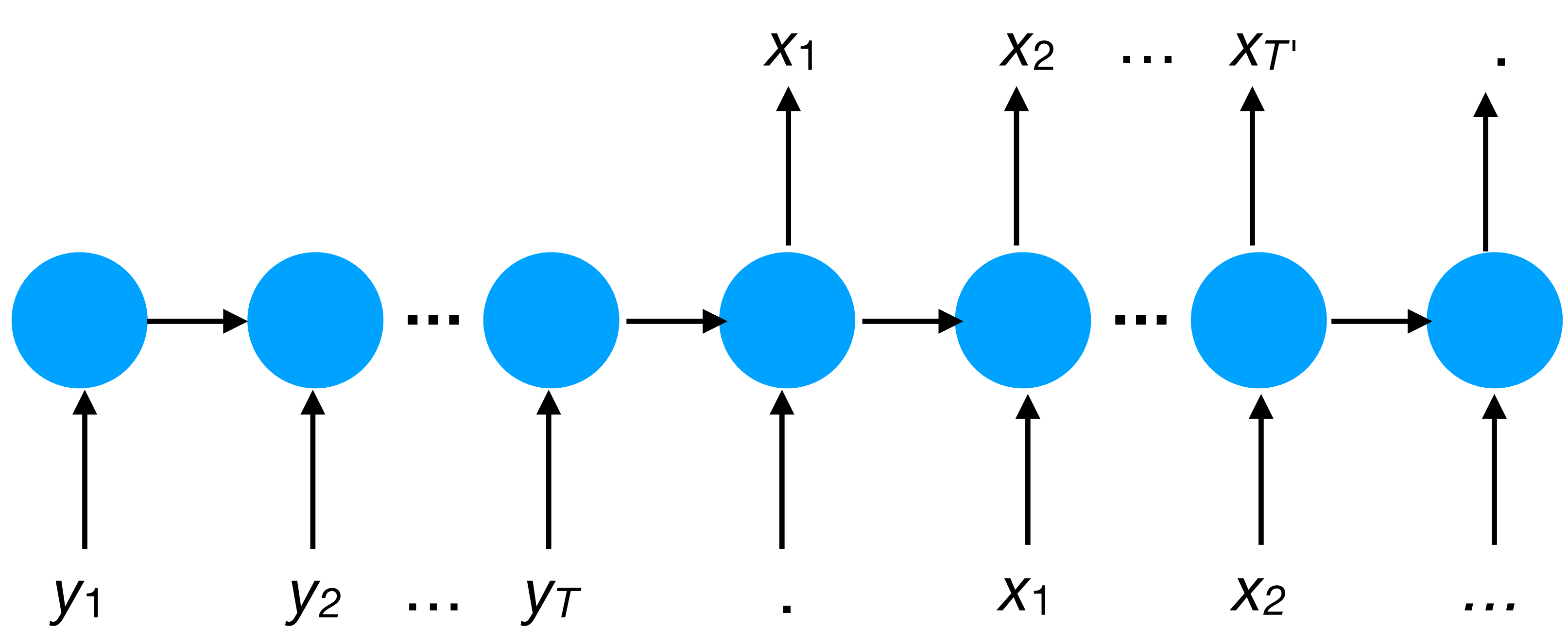
- This approach can work (and is the idea behind LLMs) but typically requires a very large model to be successful.

- Instead, it can be beneficial to break the machine translation problem into subtasks: “encoding” and “decoding”.



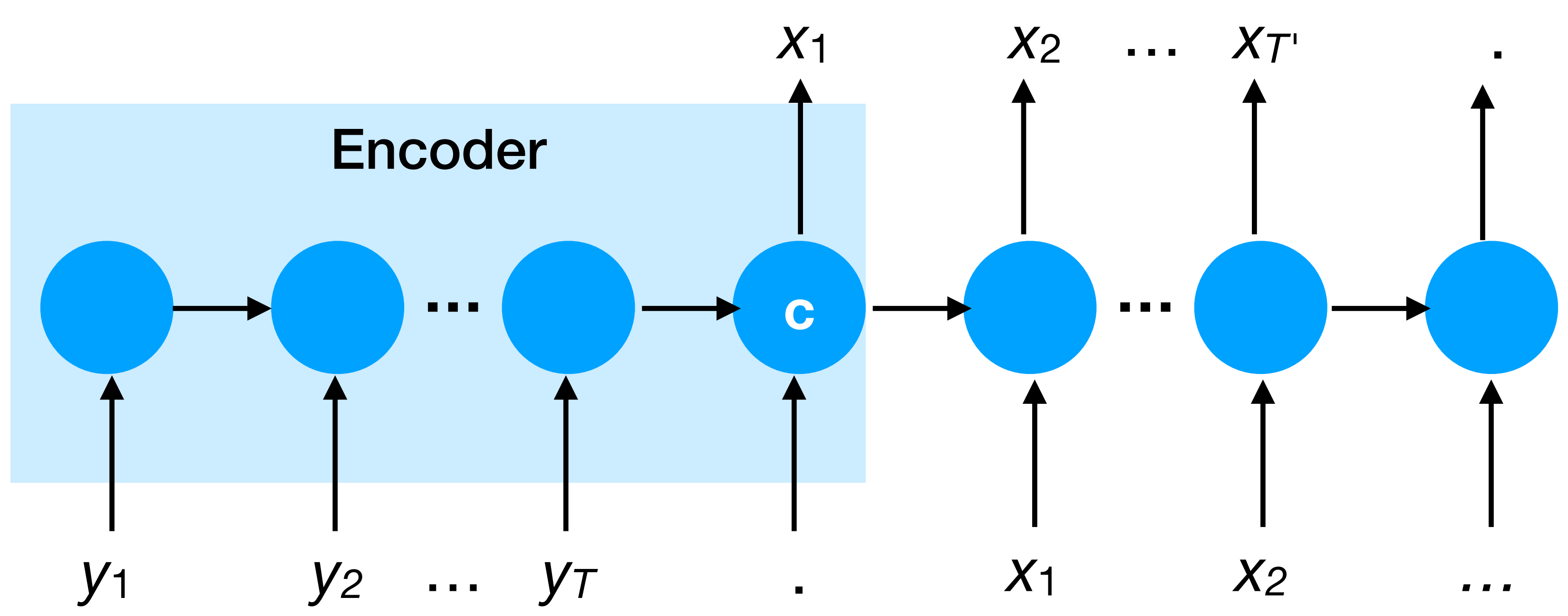
Encoder-Decoder models

- We construct a sequence-to-sequence model consisting of an **encoder** RNN and a **decoder** RNN:



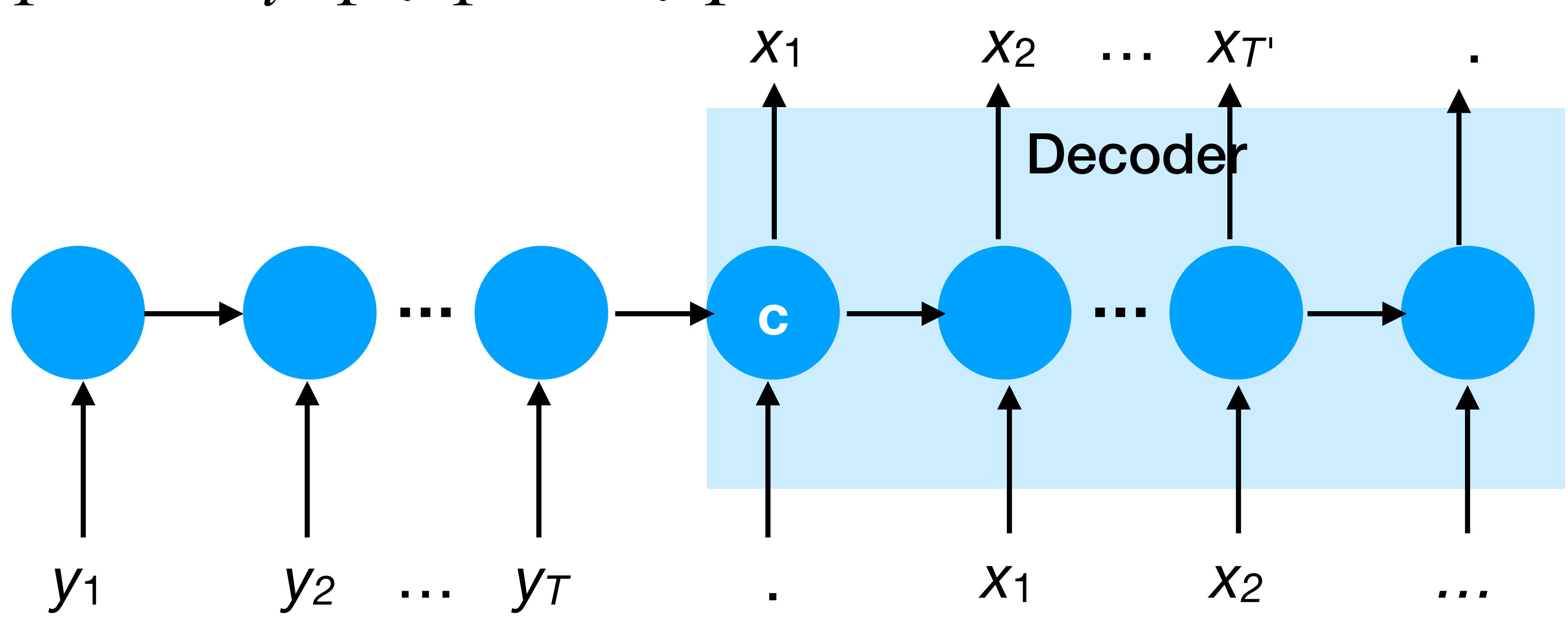
Encoder-Decoder models

- The encoder ingests the input sequence $y_1, \ldots, y_T$ and produces a context vector **c** that captures **y**'s meaning.



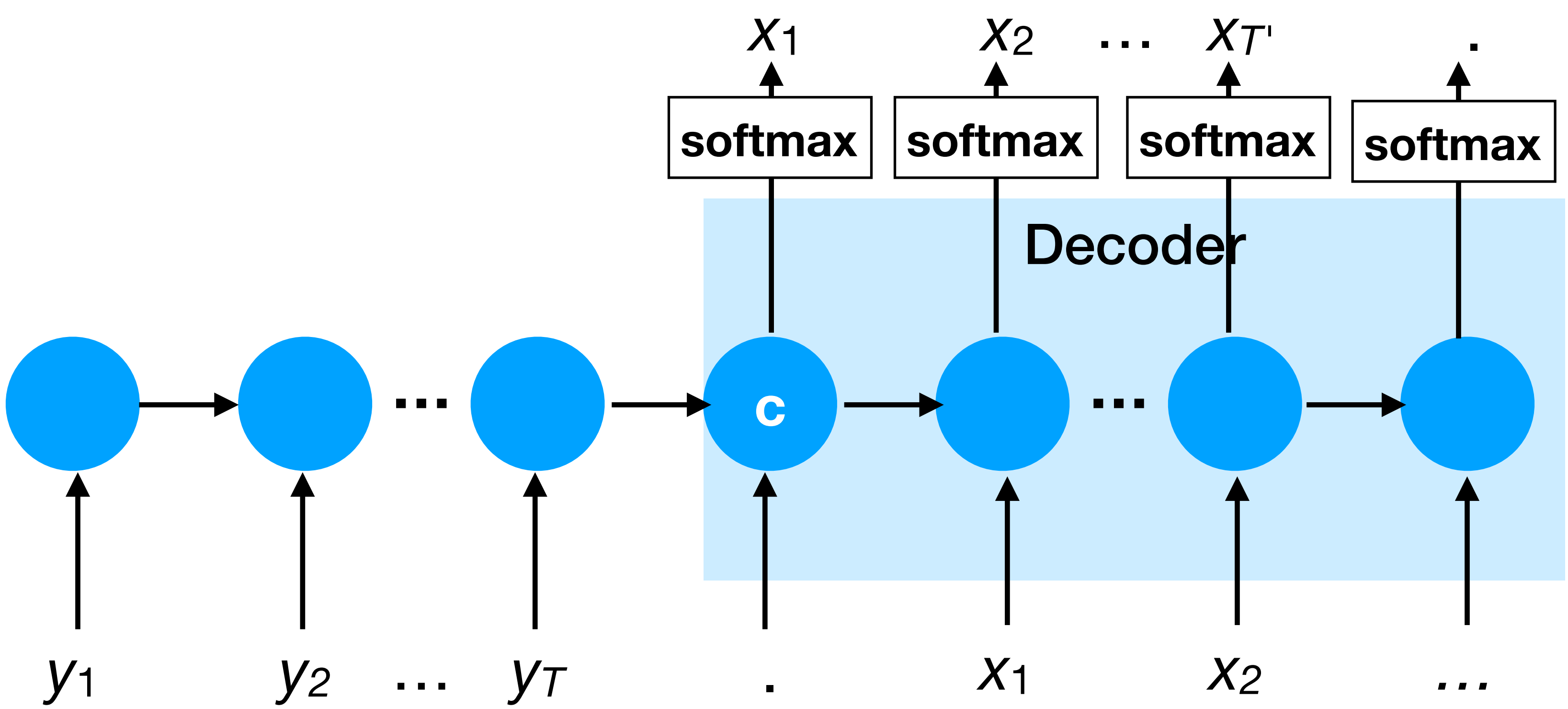
Encoder-Decoder models

- The encoder ingests the input sequence $y_1, \ldots, y_T$ and produces a context vector **c** that captures **y**'s meaning.
- The decoder uses the context vector to estimate $P(x_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$ at each timestep t.



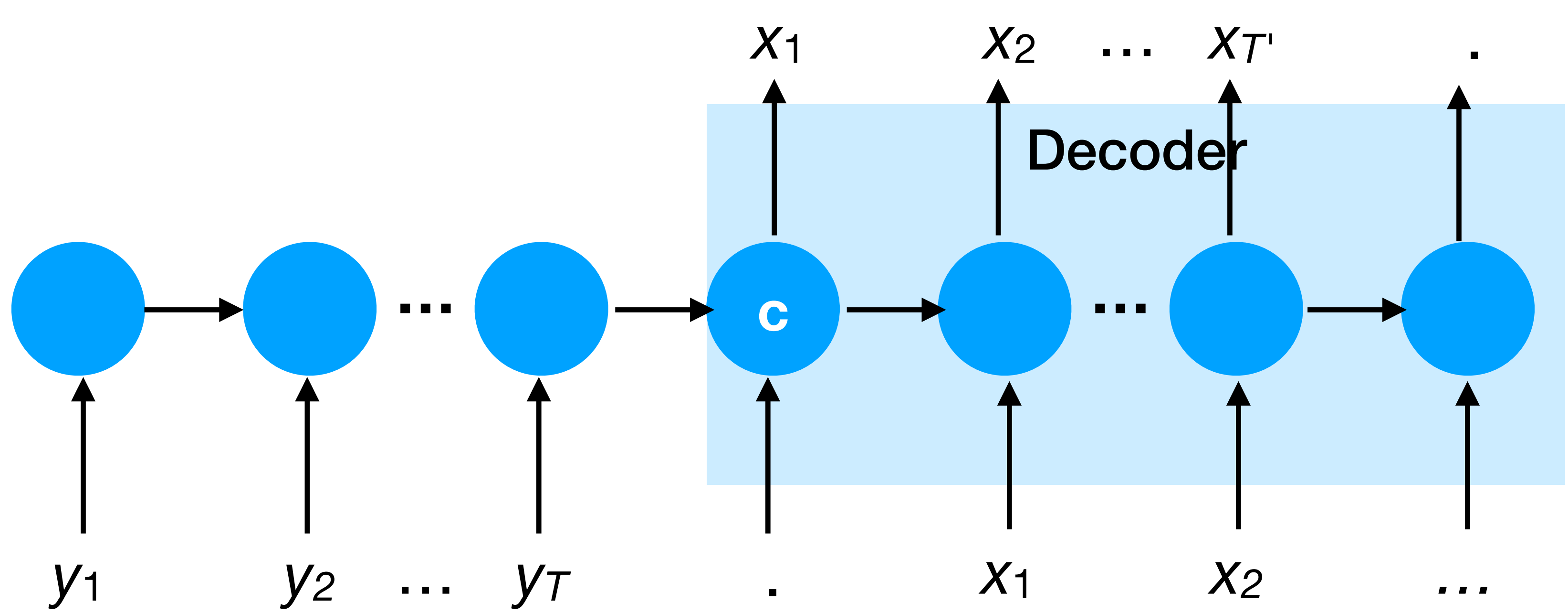
Encoder-Decoder models

- Since each x_t belongs to a finite set, we can compute $P(x_t \,|\, x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$ using softmax.



Encoder-Decoder models

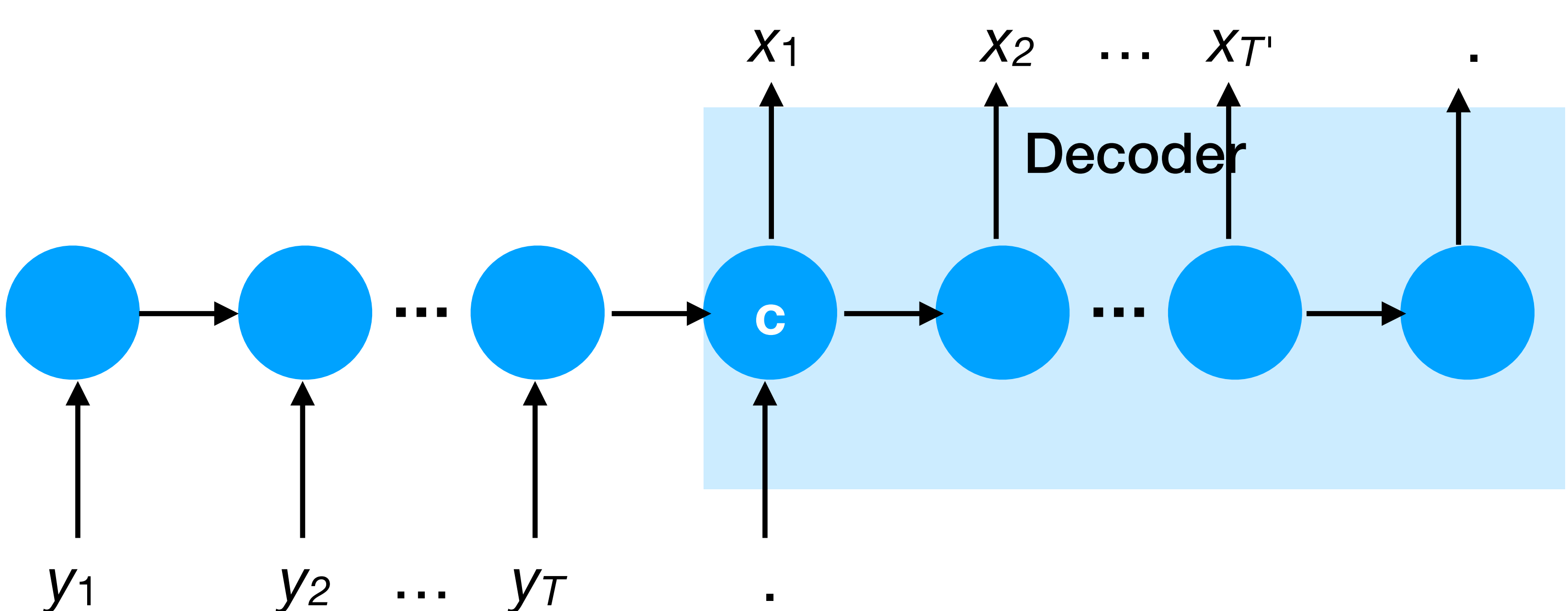
- We can then autoregressively sample each $x_t \sim P(x_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$ to generate the sentence.



Encoder-decoder model: training

- We train an encoder-decoder model to maximize the log-probability of producing the correct translation:

$$\log P(x_1, \ldots, x_{T'} \mid y_1, \ldots, y_T) = \log \prod_{t=1}^{T'} P(x_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$$
$$= \sum_{t=1}^{T'} \log P(x_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$$



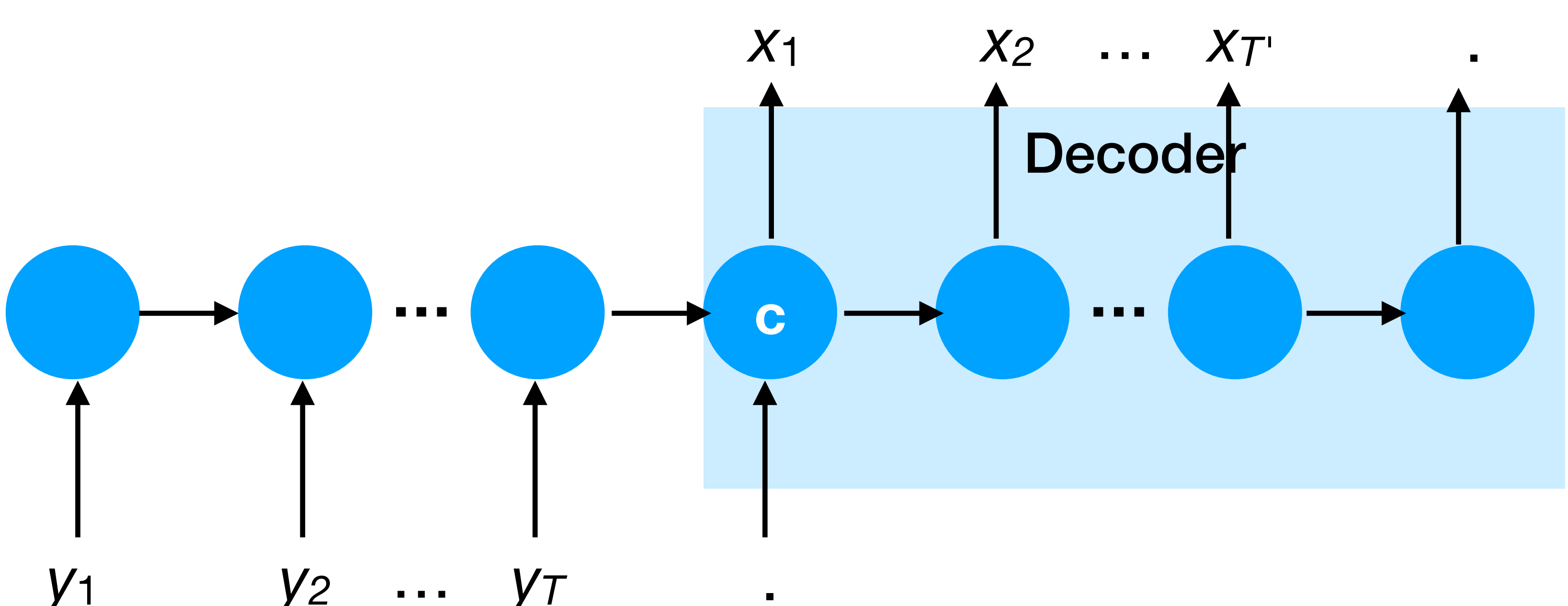
Encoder-decoder model: training

- We train an encoder-decoder model to maximize the log-probability of producing the correct translation:

$$\log P(x_1, \ldots, x_{T'} \mid y_1, \ldots, y_T) = \log \prod_{t=1}^{T'} P(x_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$$

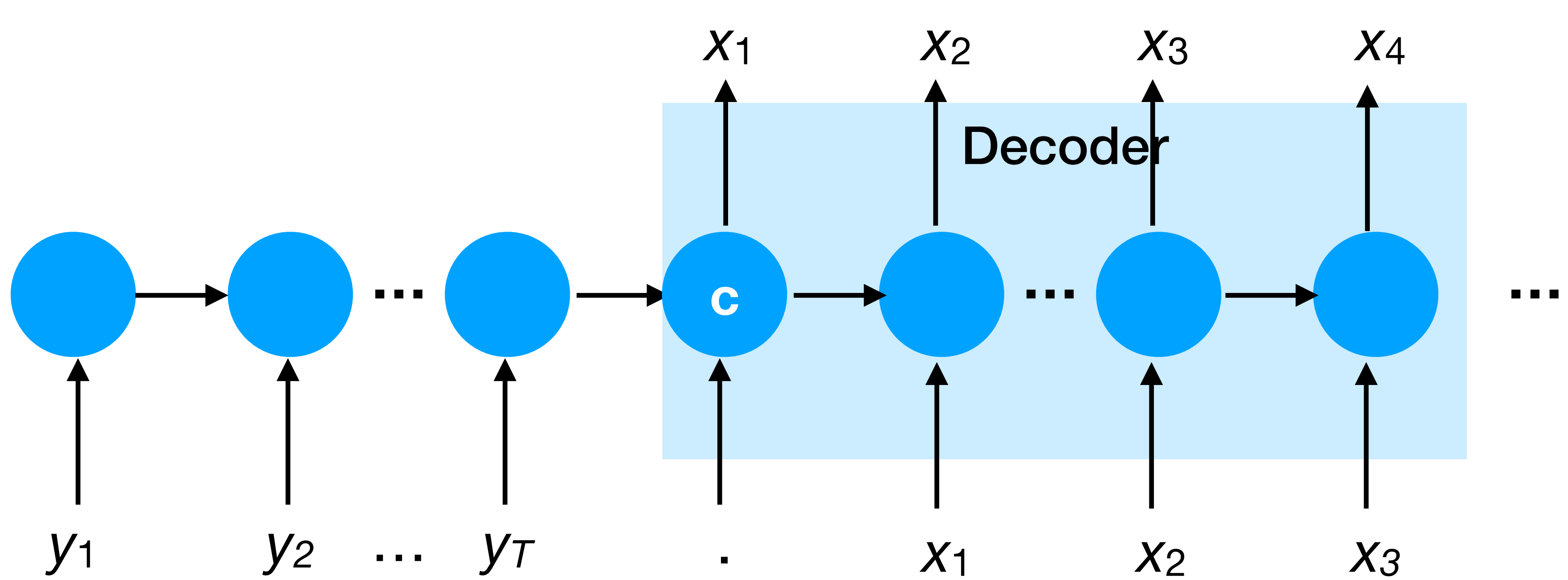
$$= \sum_{t=1}^{T'} \log P(x_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$$

Minimize CE loss to maximize log-likelihood



Teacher forcing

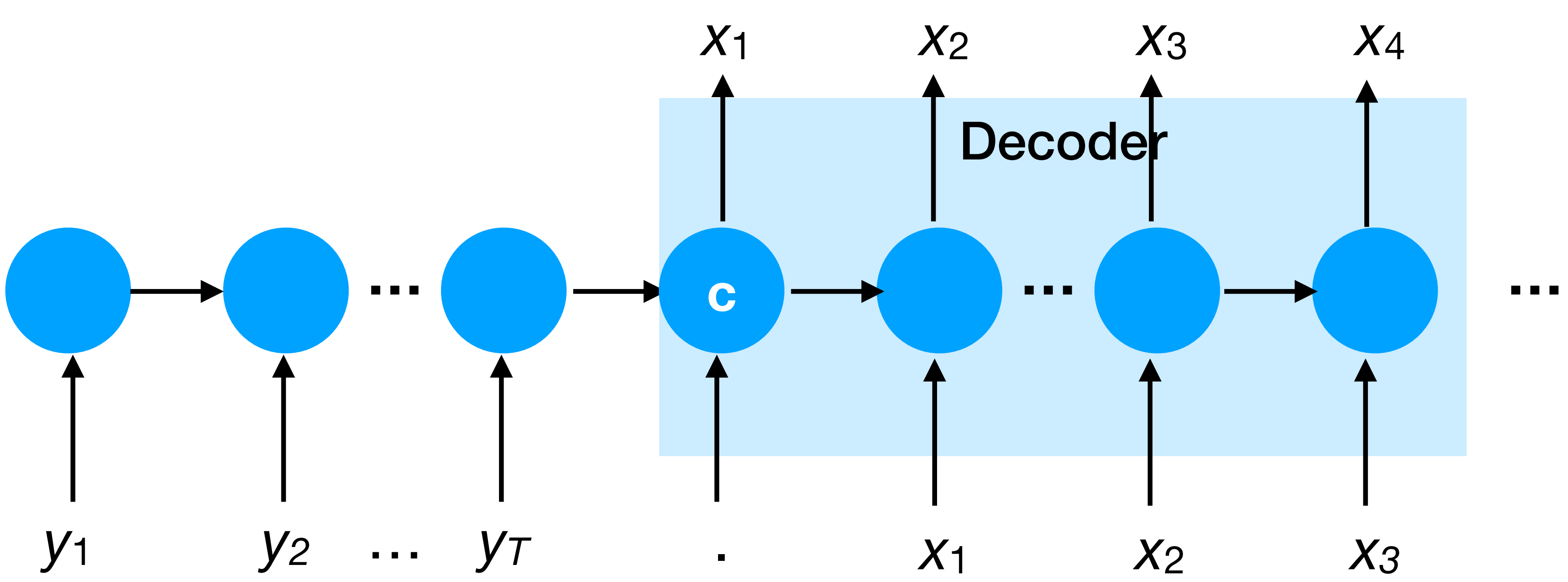
- Recall that the RNN auto-regressively feeds the predicted word x_t back to the decoder at timestep t+1.



Teacher forcing

- However, **at training time**, we can use an alternative strategy since the correct output words are known.

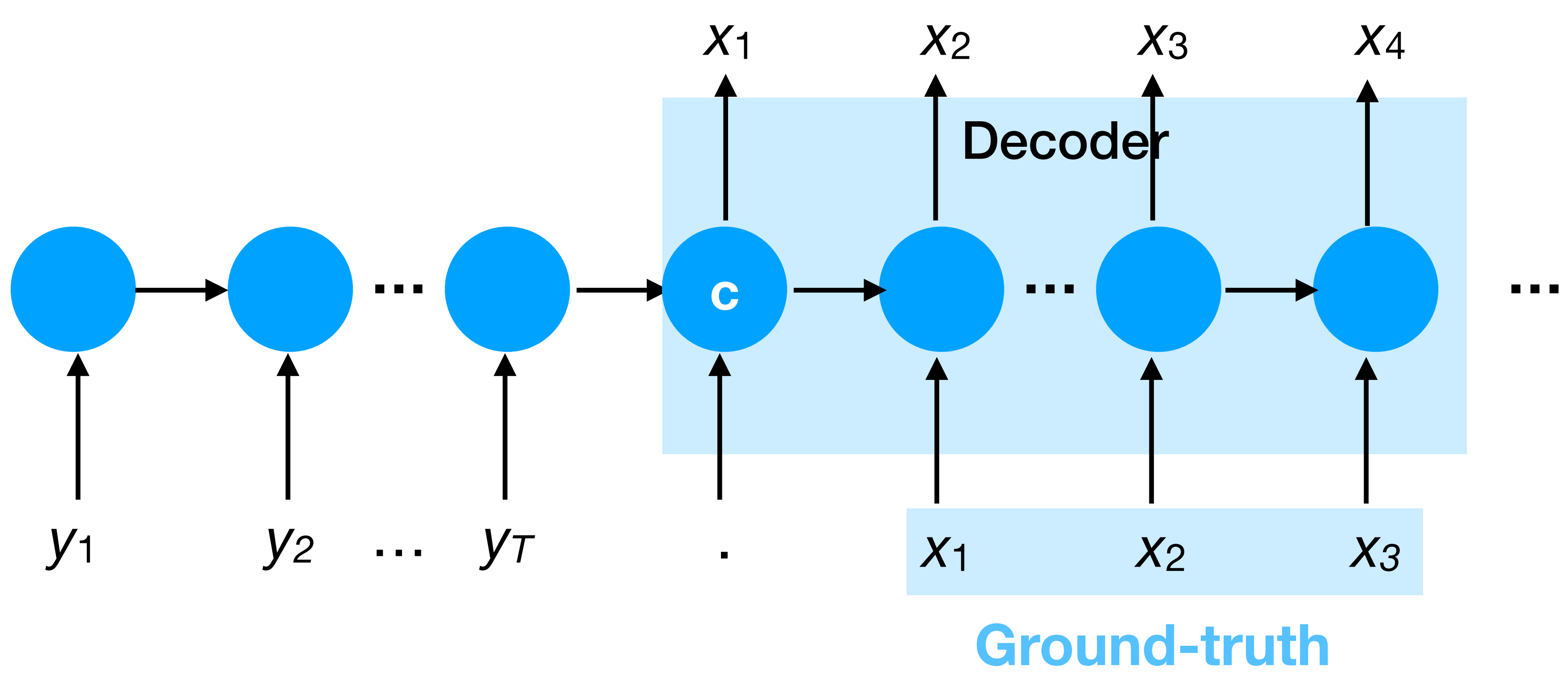
- Instead of feeding the decoder the NN's *predictions*,



Teacher forcing

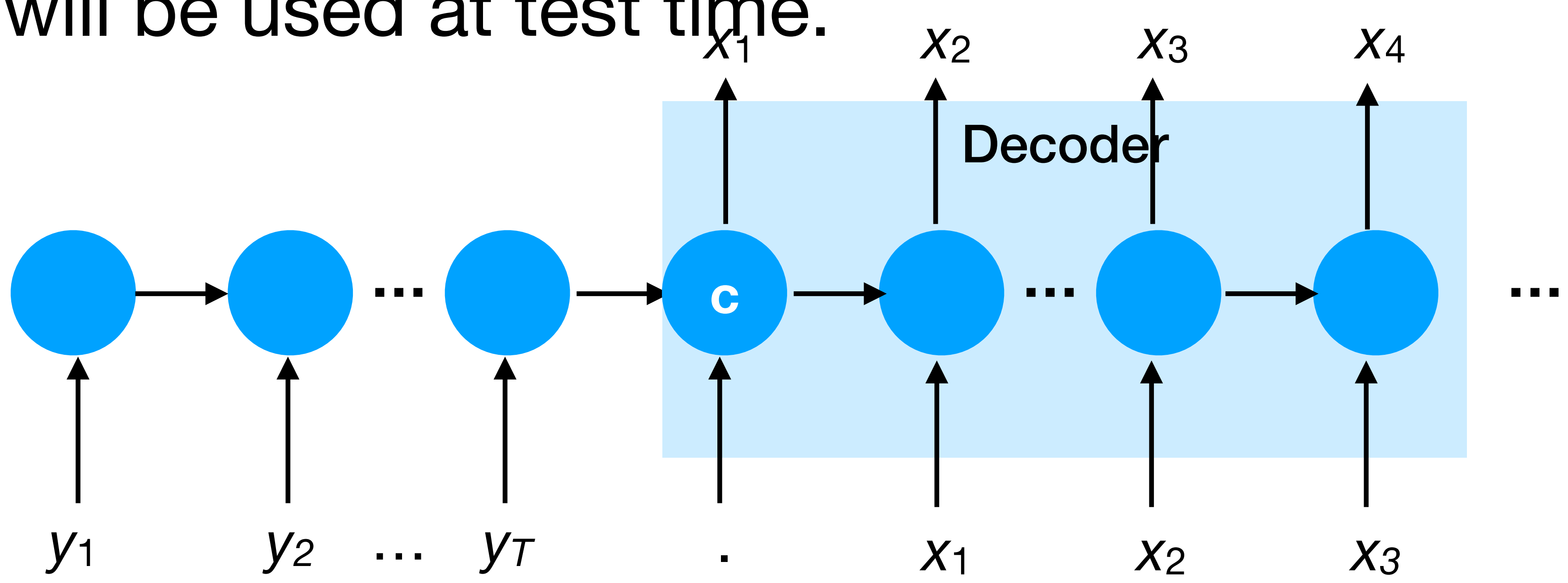
- However, **at training time**, we can use an alternative strategy since the correct output words are known.

- Instead of feeding the decoder the NN's *predictions*, we can feed the *ground-truth* values at each timestep.



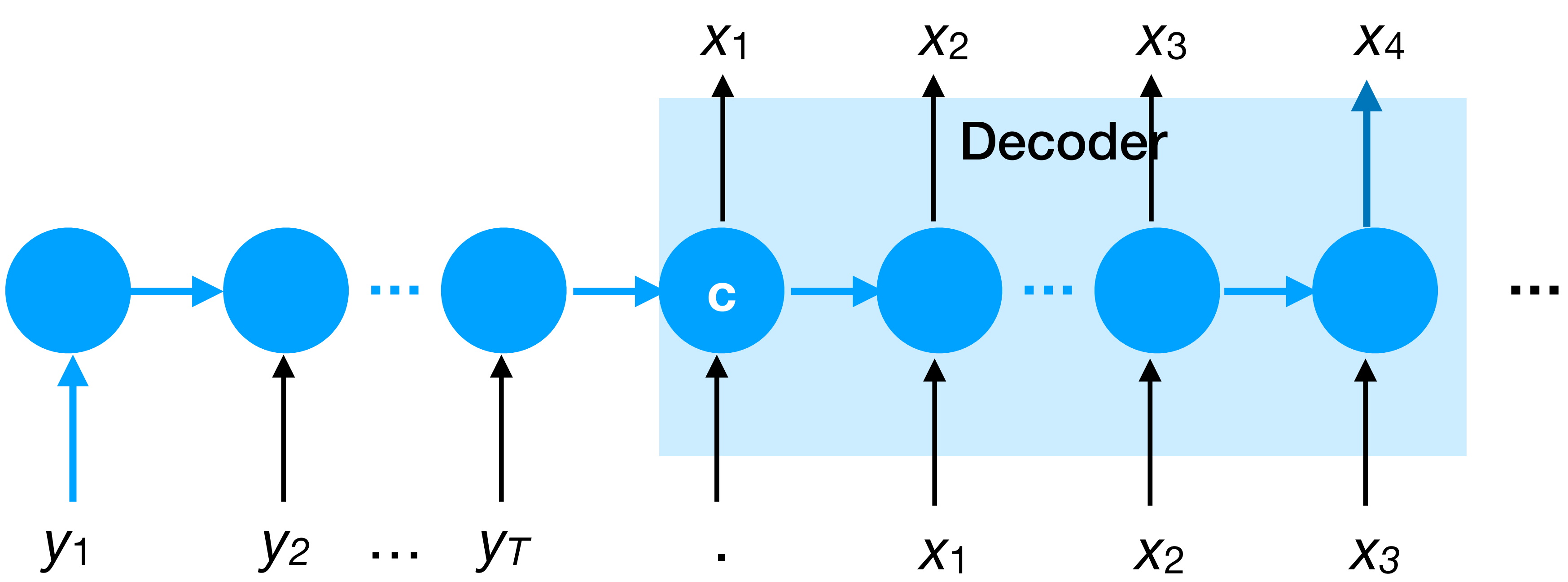
Teacher forcing

- This is called **teacher forcing**.
- Why helpful: we are feeding the NN the correct inputs rather than noisy ones.
- Why harmful: we are not training the NN consistently with how it will be used at test time.



Teacher forcing

- Unfortunately, teacher forcing does not increase parallelism since the span is still $O(T+T')$ due to the hidden state dependency.

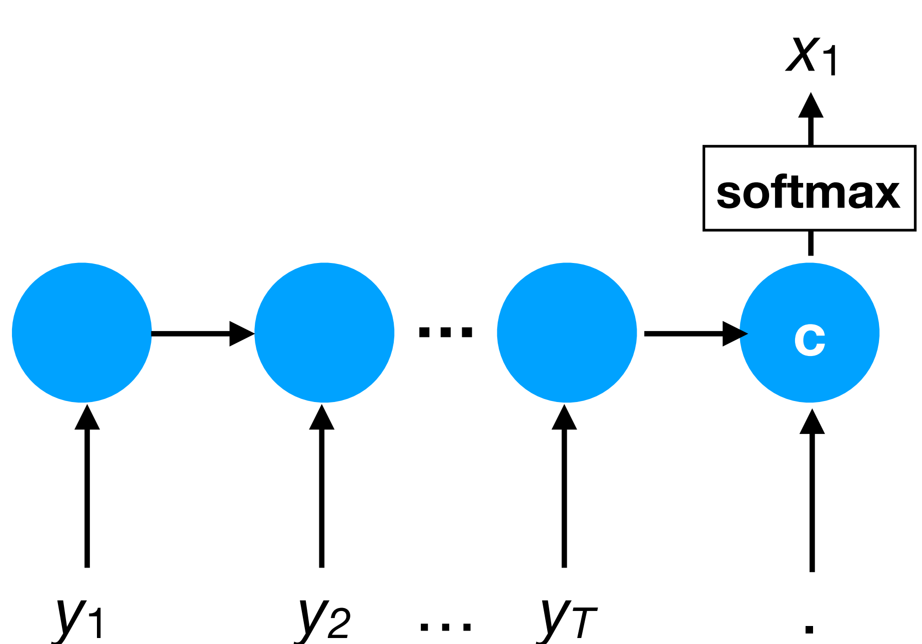


Encoder-decoder model: testing/inference

Sampling each x_t

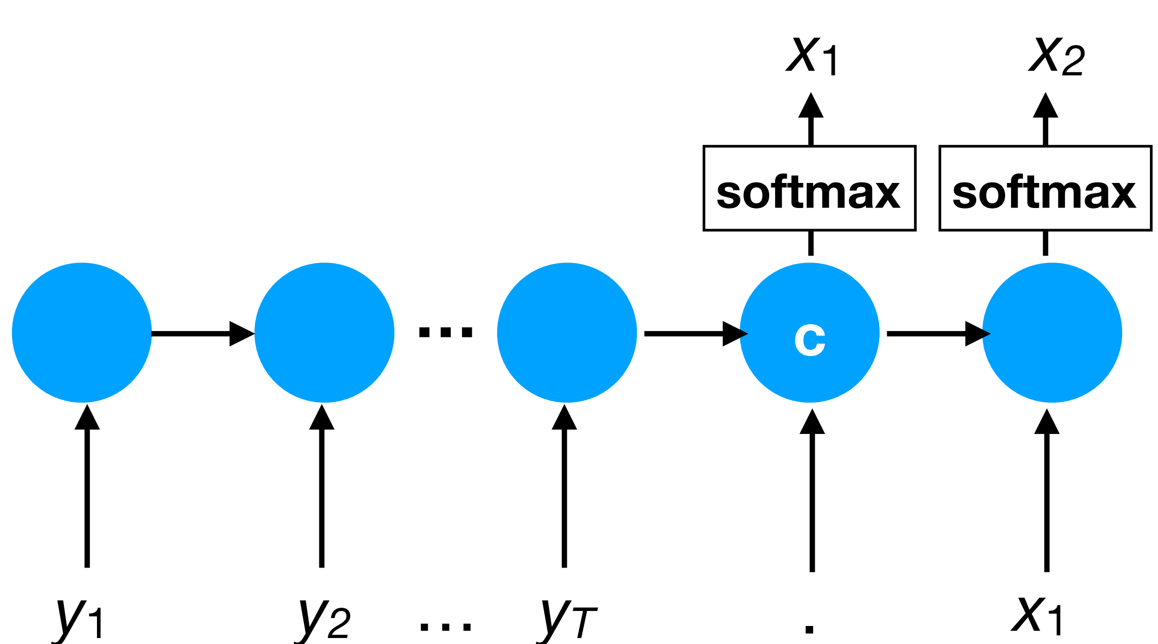
- Given each $P(x_t \,|\, x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$ estimated by the RNN, we can autoregressively sample a sentence **x**.

- Each such distribution is computed via softmax.



Sampling each x_t

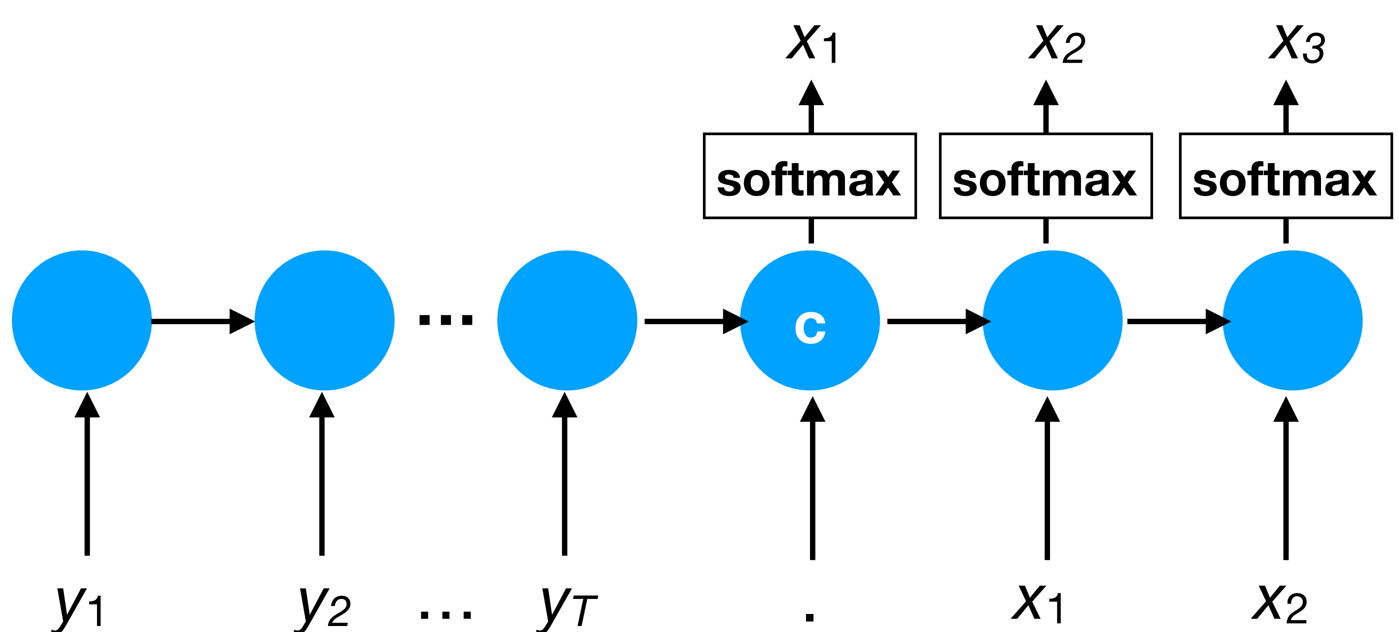
- Given each $P(x_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$ estimated by the RNN, we can autoregressively sample a sentence **x**.
- Each such distribution is computed via softmax.



Sampling each x_t

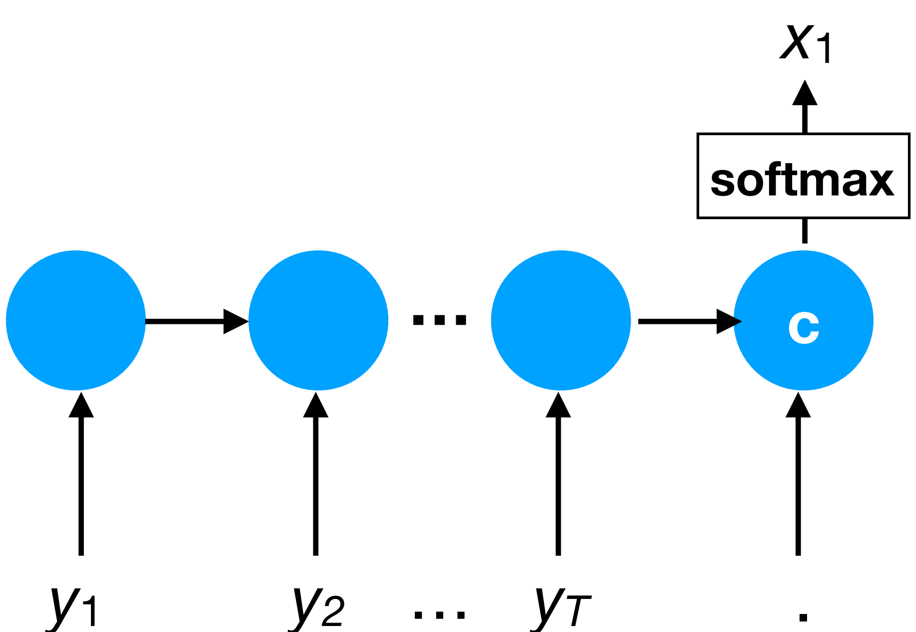
- Given each $P(x_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_T)$ estimated by the RNN, we can autoregressively sample a sentence **x**.

- Each such distribution is computed via softmax.



Temperature

- Sometimes we may want to increase/decrease the amount of noise in each x_t to yield *more likely* or *more diverse* outputs.

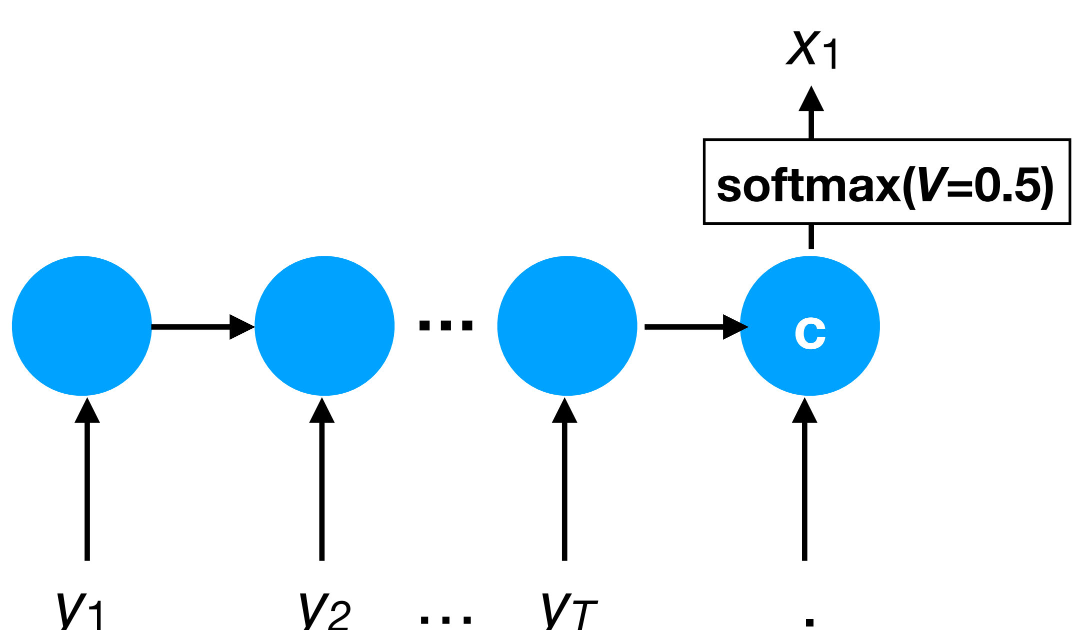


Temperature

- Sometimes we may want to increase/decrease the amount of noise in each x_t to yield *more likely* or *more diverse* outputs.

- We can achieve this by computing the softmax with a **temperature** V: $\mathrm{softmax}(\mathbf{z}, V)_k = \dfrac{\exp(z_k/V)}{\sum_{k'=1}^{K} \exp(z_{k'}/V)}$

- Higher temperature V leads to more uniform probabilities.



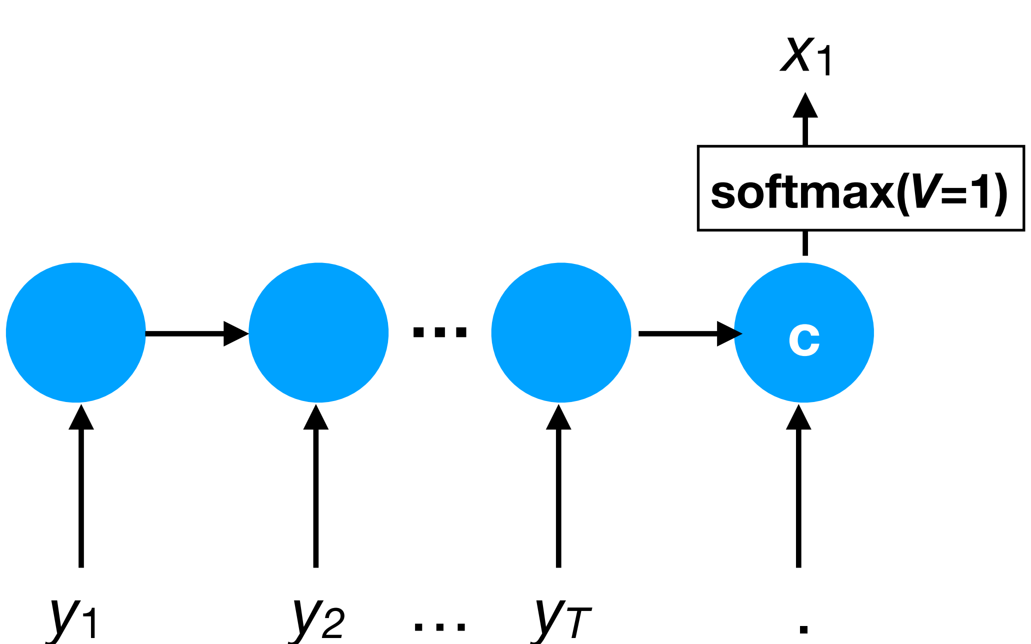
u	0.82
v	0.11
w	0.02
.	0.04

Temperature

- Sometimes we may want to increase/decrease the amount of noise in each x_t to yield *more likely* or *more diverse* outputs.

- We can achieve this by computing the softmax with a **temperature** V: $\text{softmax}(\mathbf{z}, V)_k = \dfrac{\exp(z_k/V)}{\sum_{k'=1}^{K} \exp(z_{k'}/V)}$

- Higher temperature V leads to more uniform probabilities.

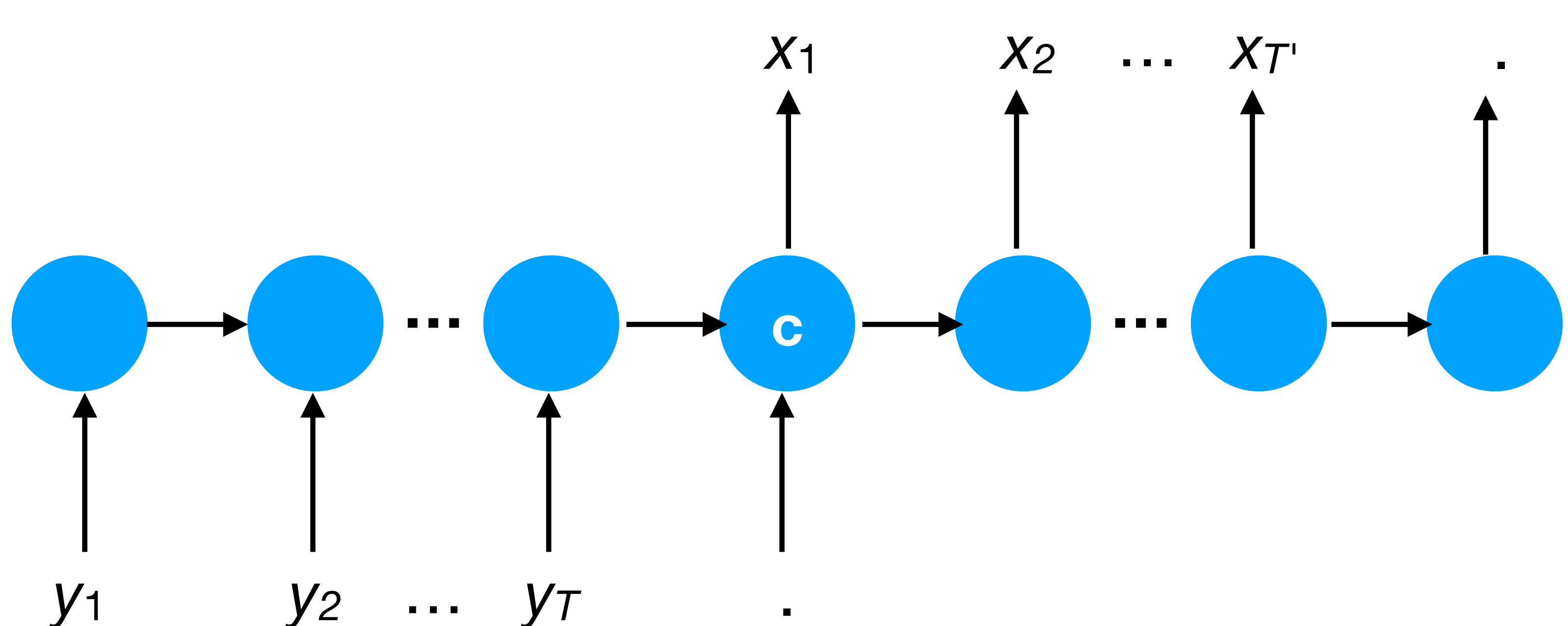


u	0.56
v	0.21
w	0.10
.	0.13

Inference: Finding the most likely translations

- We may wish to find the **most likely sentence** $x_1, \ldots, x_{T'}$ that corresponds to the input sentence $y_1, \ldots, y_T$.

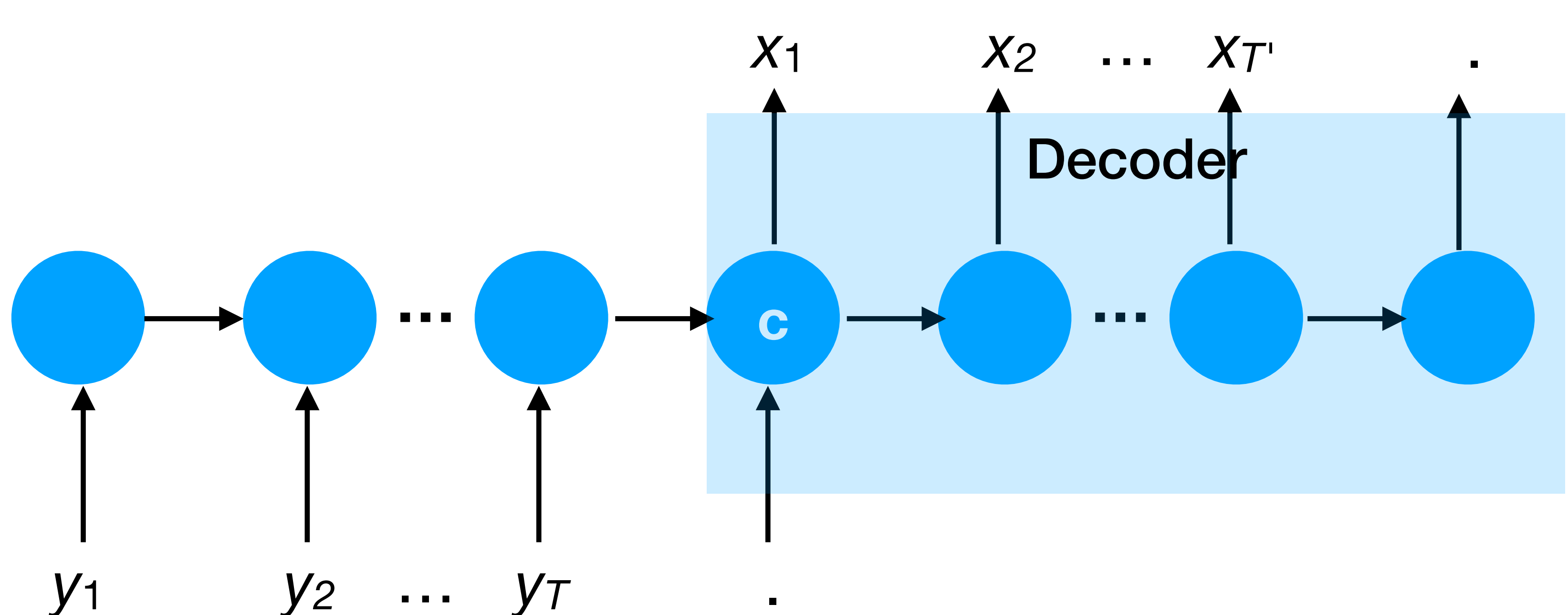
- How can we compute the likelihood $P(x_1, \ldots, x_{T'} \mid y_1, \ldots, y_T)$?



Inference: Finding the most likely translations

- Since $P(x_1, \ldots, x_{T'} \mid y_1, \ldots, y_T)$ factorizes over t, we can:
  1. Pass a sentence $y_1, \ldots, y_T$ into the RNN.
  2. Obtain the probability of each symbol x_t.
  3. Multiply the probabilities together:

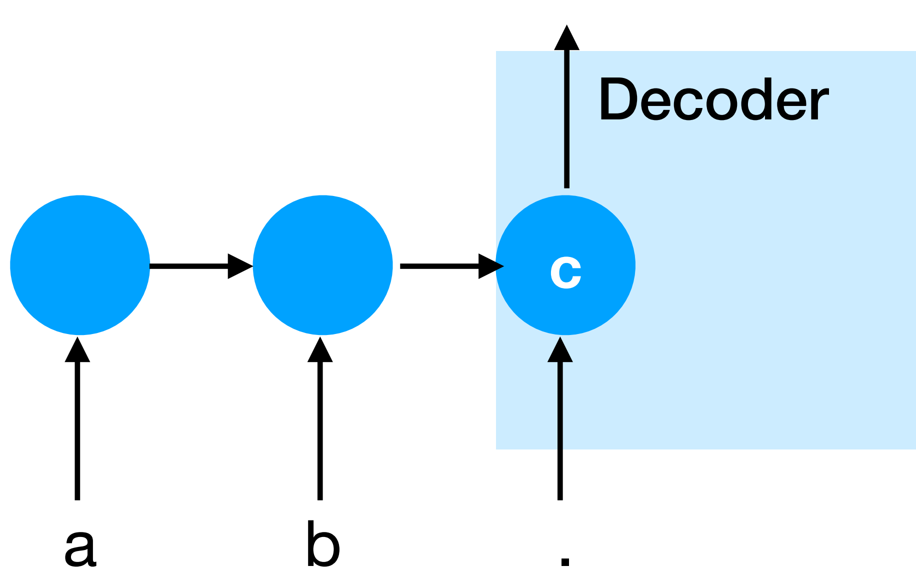
$$P(x_1, x_2, x_3 \mid y_1, y_2) = P(x_1 \mid y_1, y_2)\, P(x_2 \mid y_1, y_2, x_1)\, P(x_3 \mid y_1, y_2, x_1, x_2)$$



Inference: Finding the most likely translations

- Suppose y_1=a, y_2=b, y_3=. Then for x_1=w, x_2=v, we have:

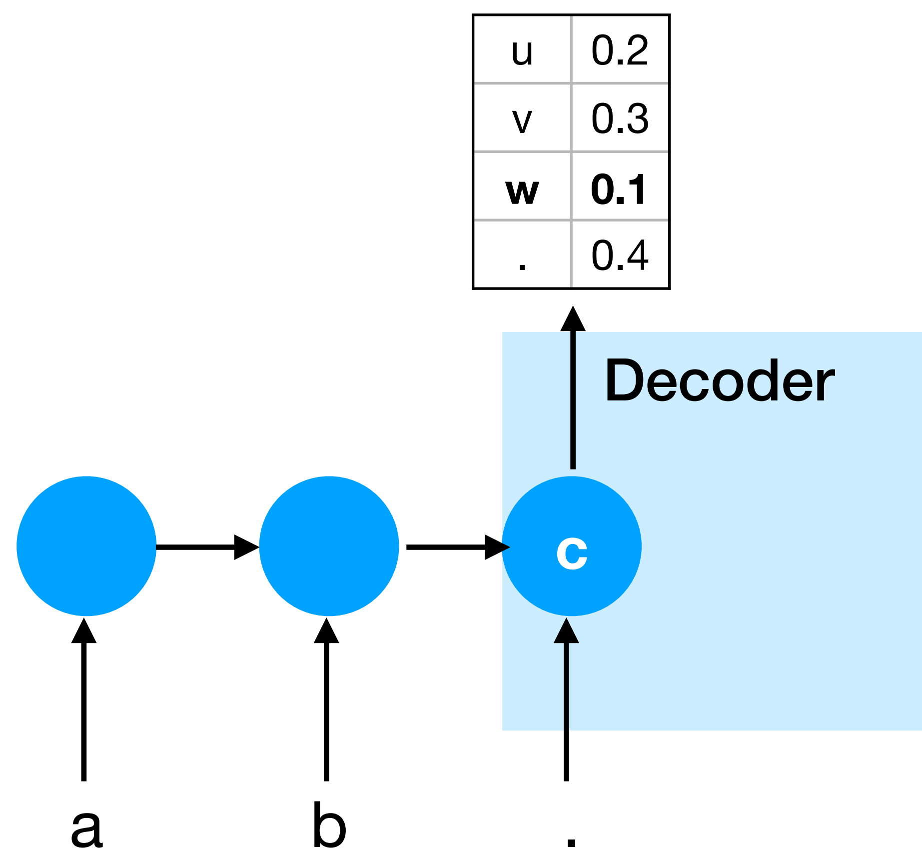
***P*(“w v” | “a b.”) =**



Inference: Finding the most likely translations

- Suppose y_1=a, y_2=b, y_3=. Then for x_1=w, x_2=v, we have:

***P*("w v" | "a b.") = 0.1**

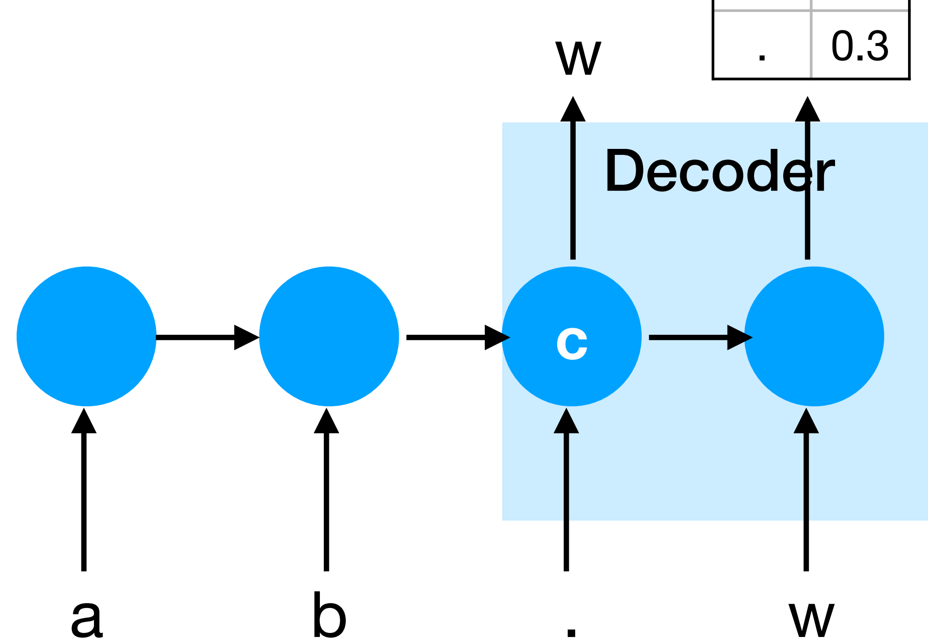


Inference: Finding the most likely translations

- Suppose y_1=a, y_2=b, y_3=. Then for x_1=w, x_2=v, we have:

***P*(“w v” | “a b.”) = 0.1 * 0.7**

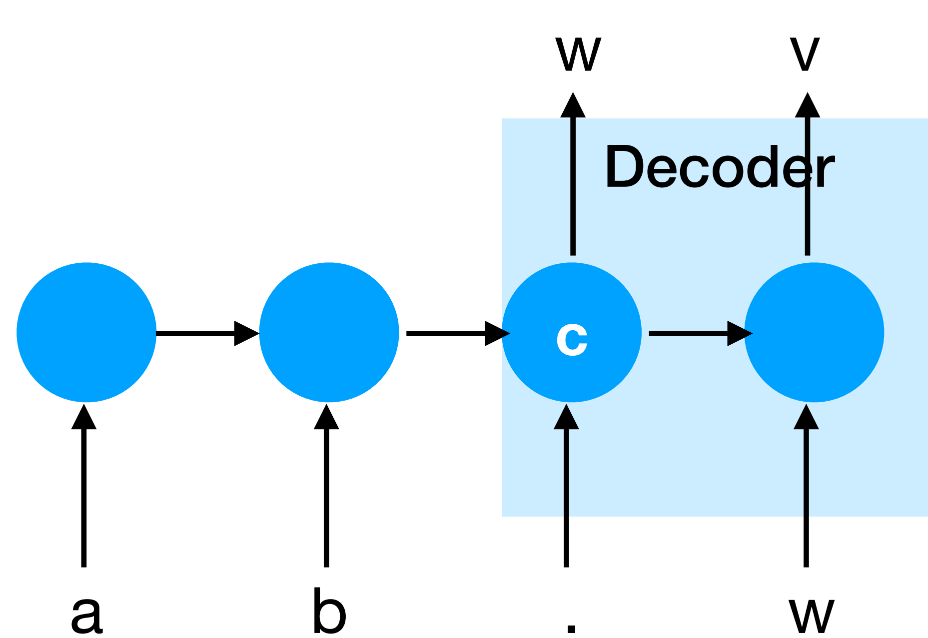
u	0.1
v	**0.7**
w	0
.	0.3



Inference: Finding the most likely translations

- Suppose y_1=a, y_2=b, y_3=. Then for x_1=w, x_2=v, we have:

P("w v" | "a b.") = 0.1 * 0.7 = 0.07



Inference: Finding the most likely translations

- Unfortunately, to search over all translations, there are exponentially (in T') many different probabilities $P(x_1, \ldots, x_{T'} \mid y_1, \ldots, y_T)$ we would need to compute.

- Heuristic: perform a greedy **beam search** to keep track of the top-B most likely translations $x_1, \ldots, x_{T'}$.

Beam search

Beam search

- Beam search is an efficient greedy heuristic that *approximately* optimizes:

$$\arg\max_{x_1,\ldots,x_{T'}} p(x_1, \ldots, x_{T'} \,|\, y_1, \ldots, y_T)$$

Beam search

1. At each output timestep *t*, keep track of top-*B* most likely translations, where *B* is the **beam width**:

$$\{(x_1, \ldots, x_t)^{(1)}, \ldots, (x_1, \ldots, x_t)^{(B)}\}$$

2. For each of our *B* candidates, we can compute:

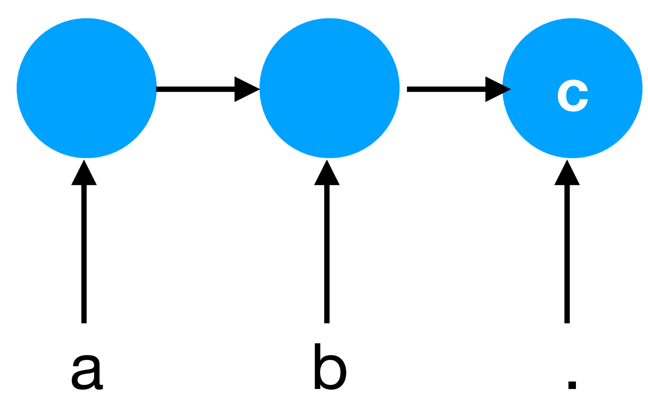
$$P(x_{t+1} \mid x_1, \ldots, x_t, y_1, \ldots, y_T)$$

3. If the output vocabulary has *K* words, then this results in *B***K* possible sequences of length *t*+1.

4. From these *B***K* choices, we select the top-*B* most likely translations of length *t*+1.

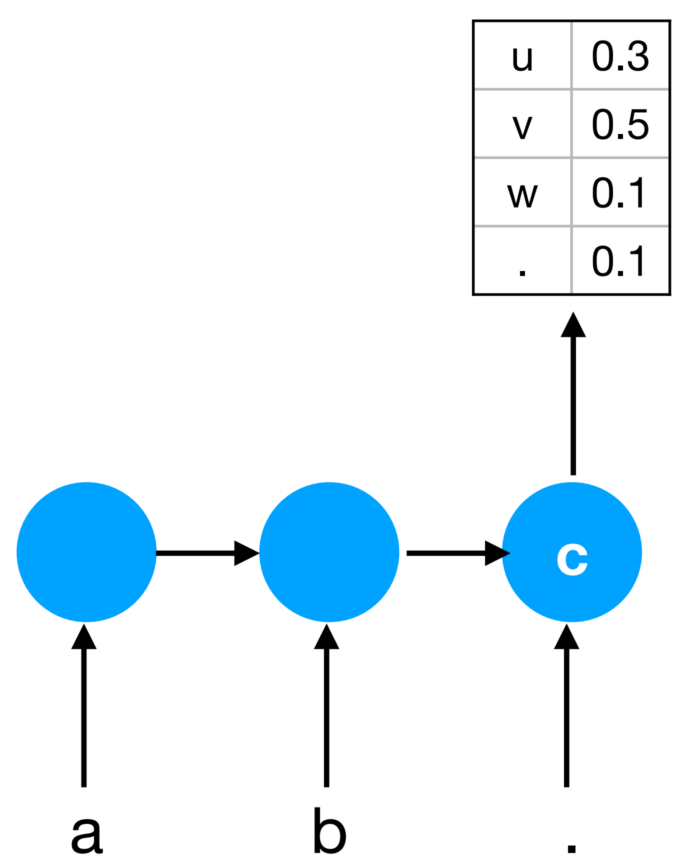
Example

- Let input vocabulary={a, b, .} and output vocabulary={u, v, w, .}.
- Let beam width *B*=2.



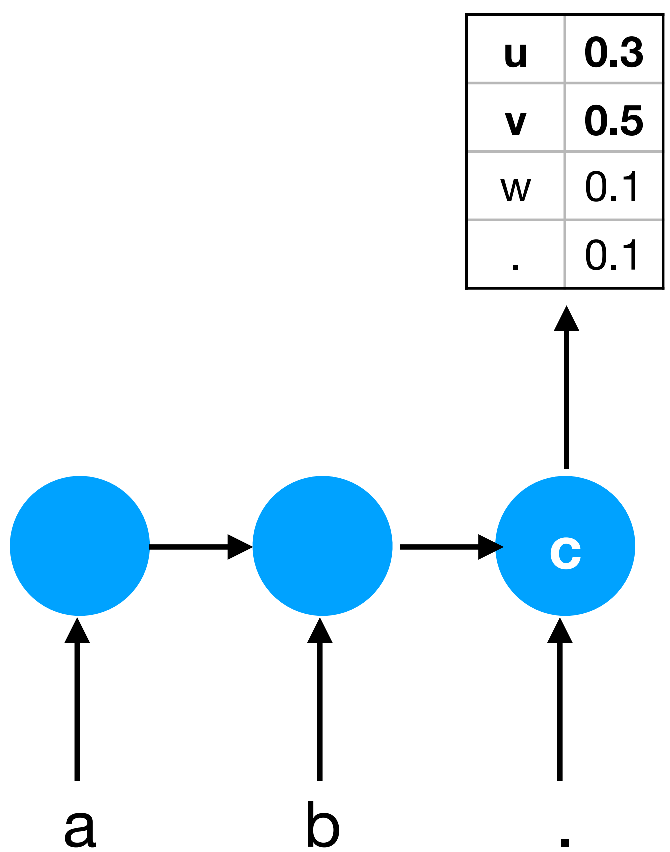
Example

- Beam at t=0: {}
- At t=1, pick top-B most likely possible symbols:



Example

- Beam at t=1: { $\overset{0.5}{(x_1=\text{v})}$, $\overset{0.3}{(x_1=\text{u})}$ }



Example

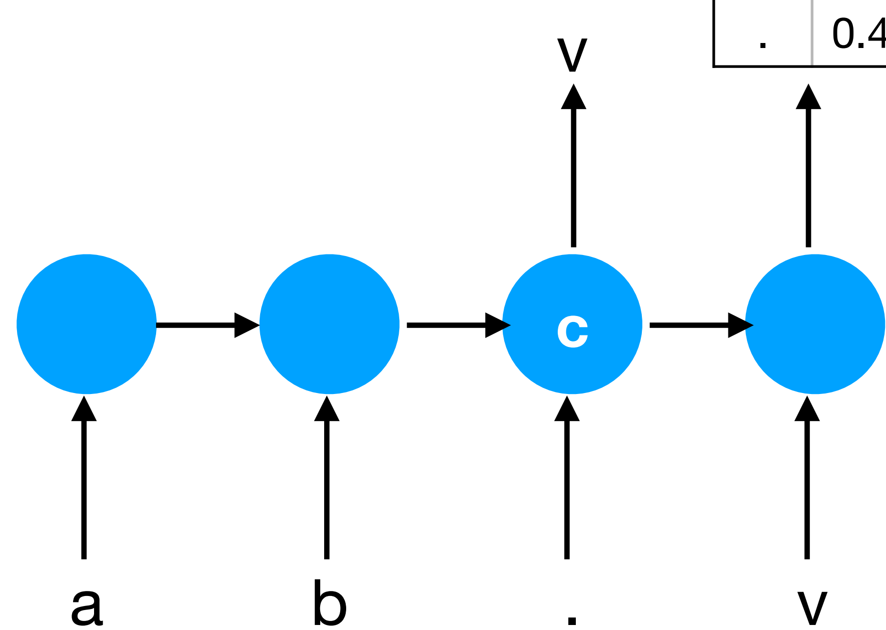
0.5 0.3

- Beam at t=1: { (**$\boldsymbol{x_1}$=v**), (x_1=u) }

- At t=2, compute $P(x_2 \mid x_1, y_1{=}\text{a}, y_2{=}\text{b})$ for each (x_1) in the beam:

$P(x_2 \mid x_1{=}\text{v}, y_1{=}\text{a}, y_2{=}\text{b})$

u	0.2
v	0.3
w	0.1
.	0.4



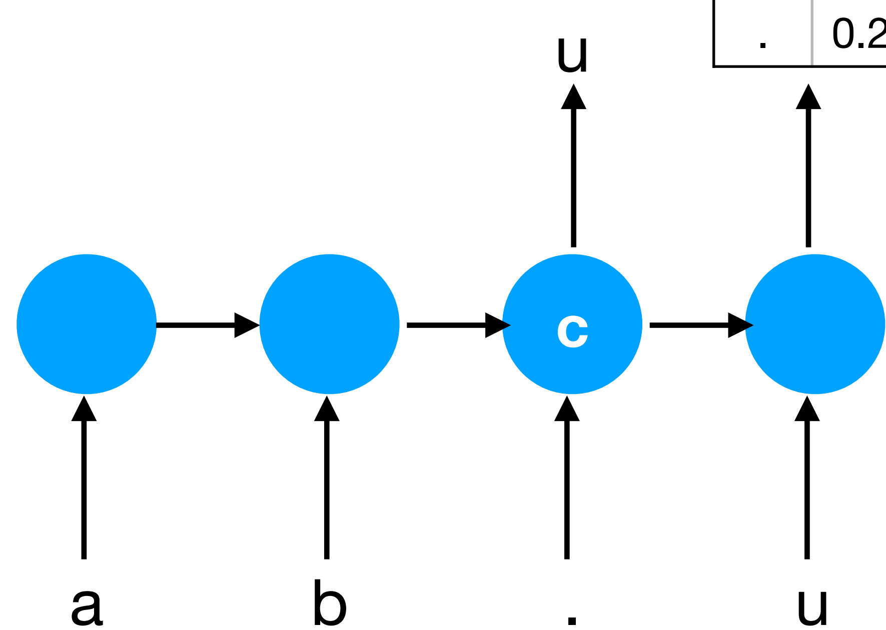
Example

- Beam at t=1: { $\overset{0.5}{(x_1\text{=v})}$, $\overset{0.3}{(\boldsymbol{x_1}\textbf{=u})}$ }

- At t=2, compute $P(x_2 \mid x_1, y_1\text{=a}, y_2\text{=b})$ for each (x_1) in the beam:

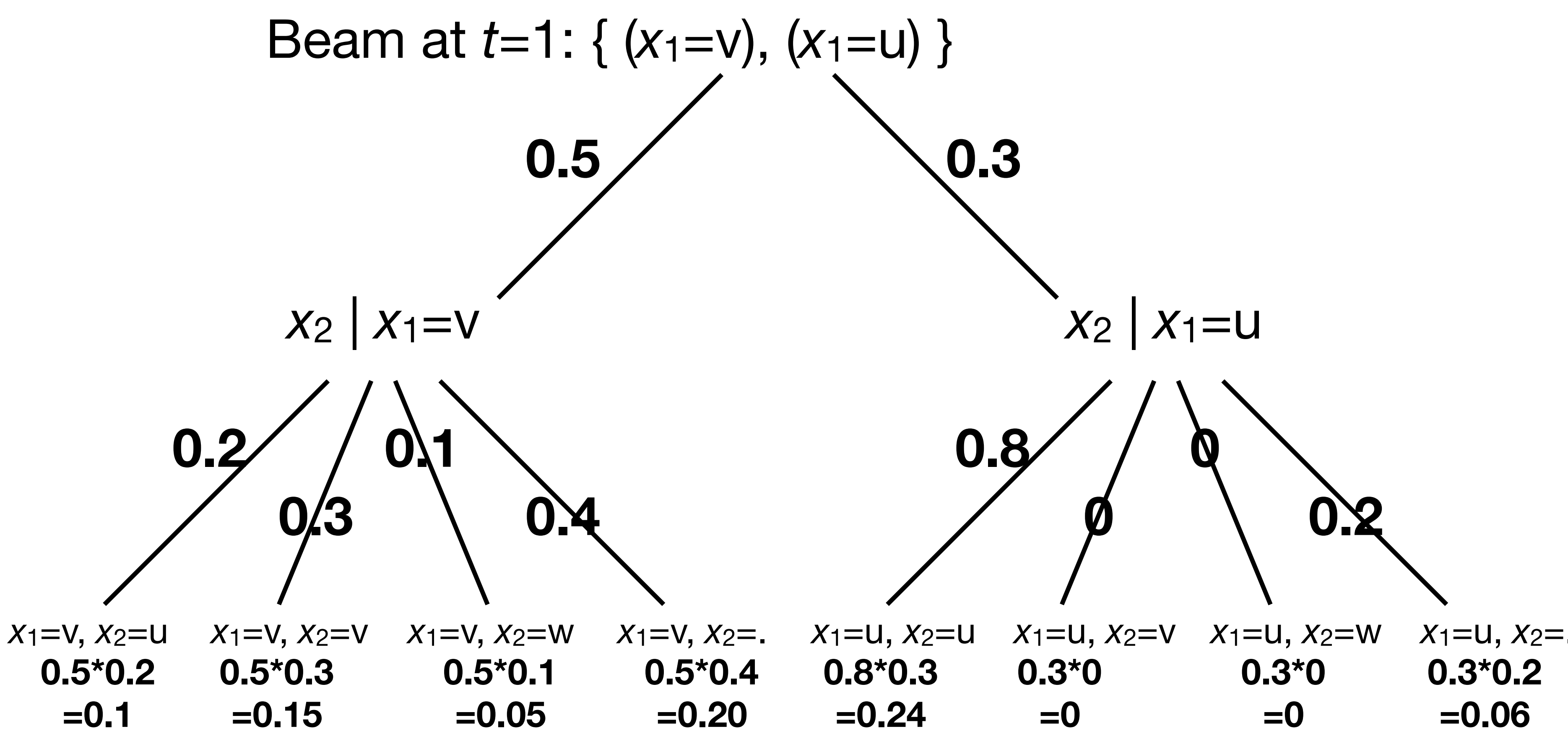
$P(x_2 \mid x_1\text{=u}, y_1\text{=a}, y_2\text{=b})$

u	0.8
v	0
w	0
.	0.2



Example

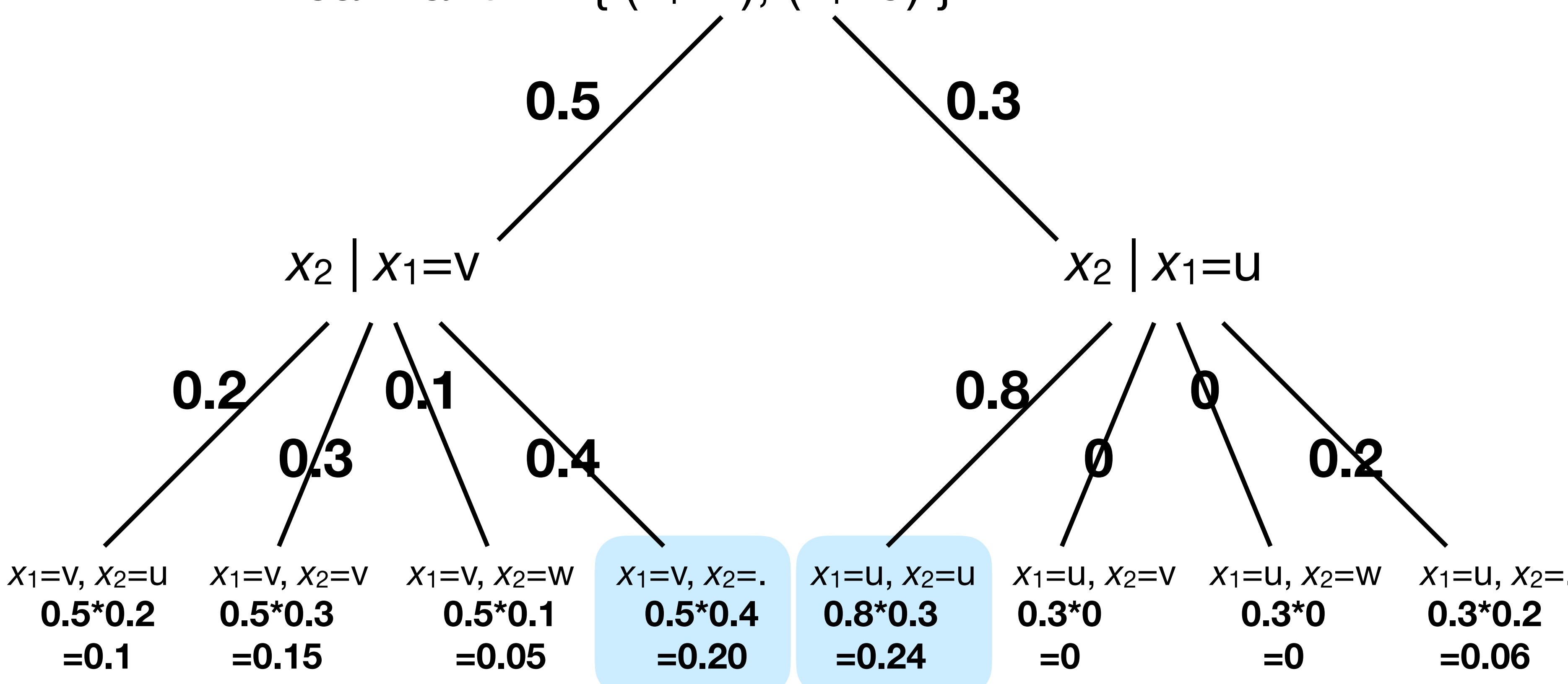
- This results in a total of B*K=2*4=8 possible sequences of length 2:



Example

- We then pick the top-*B* most likely sequences as our next beam.

Beam at t=1: { (x_1=v), (x_1=u) }



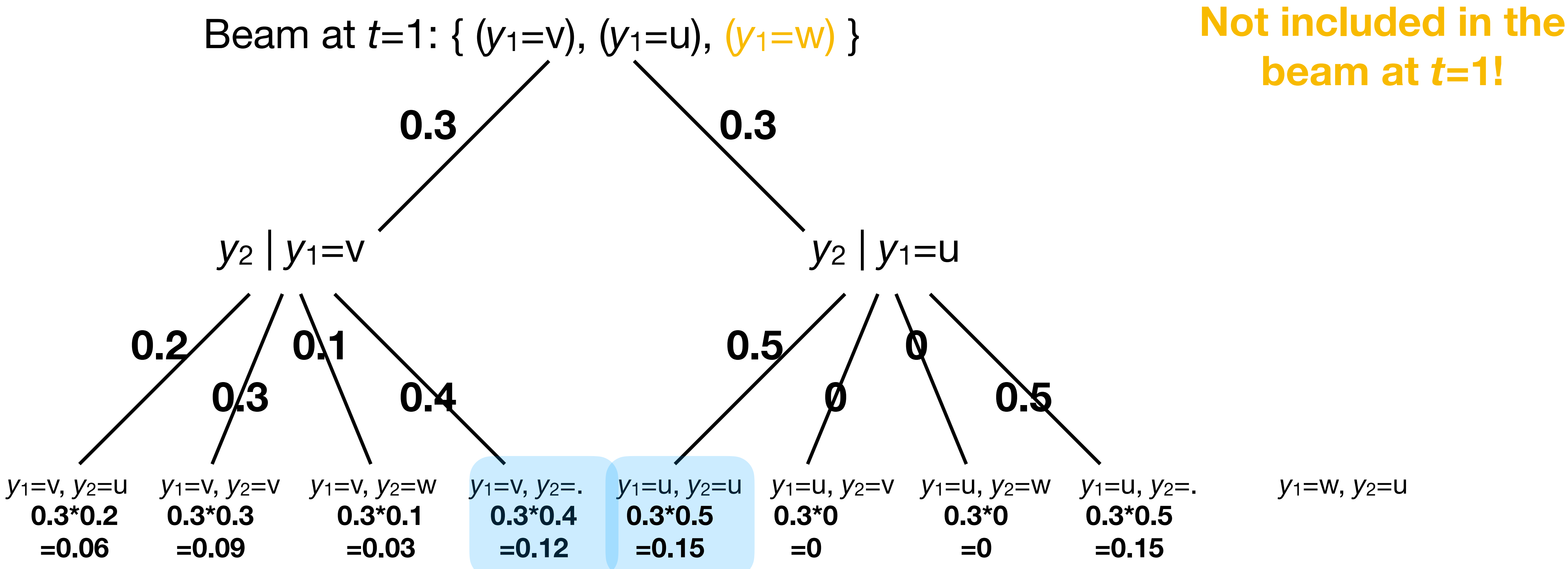
Beam at t=2: { (x_1=v, x_2=.), (x_1=u, x_2=u) }

Example

- We repeat this procedure until all beam hypotheses end with "." (end-of-sentence), or for a fixed maximum number of timesteps.

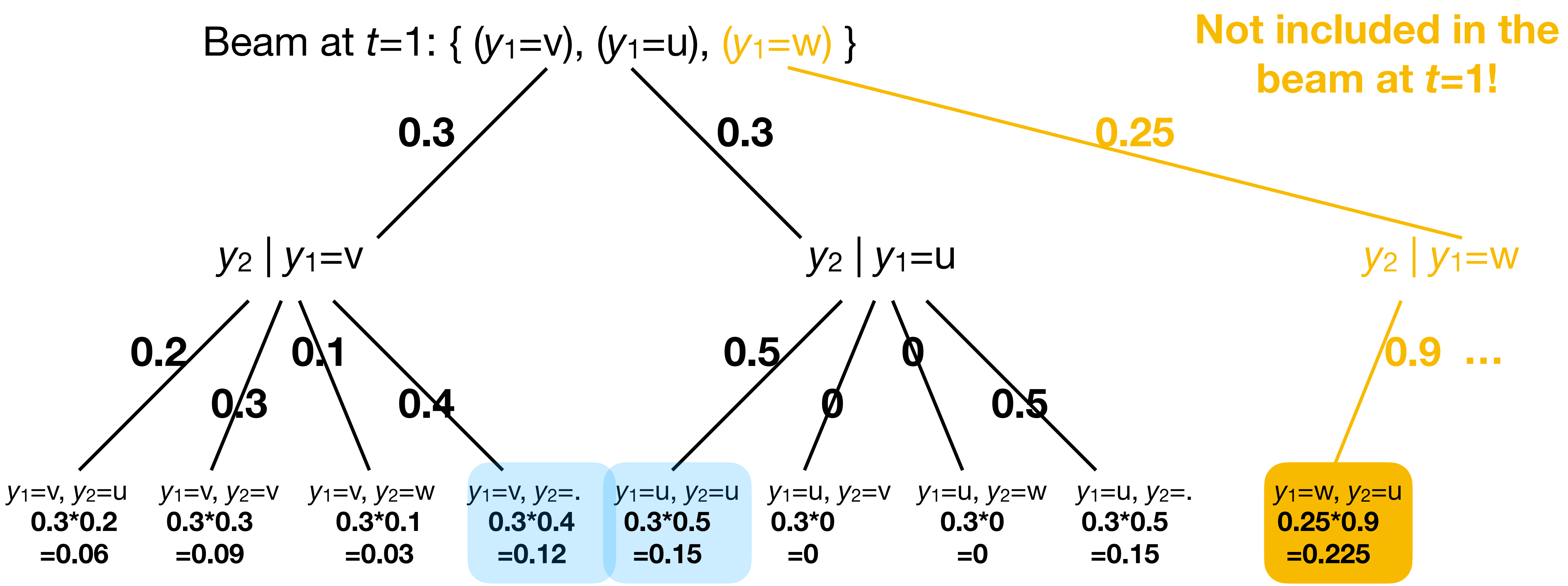
Exercise

- For the vocabulary { u, v, w, . }, can you devise a scenario over 2 timesteps where beamsearch does *not* give the optimal answer?



Exercise: Solution

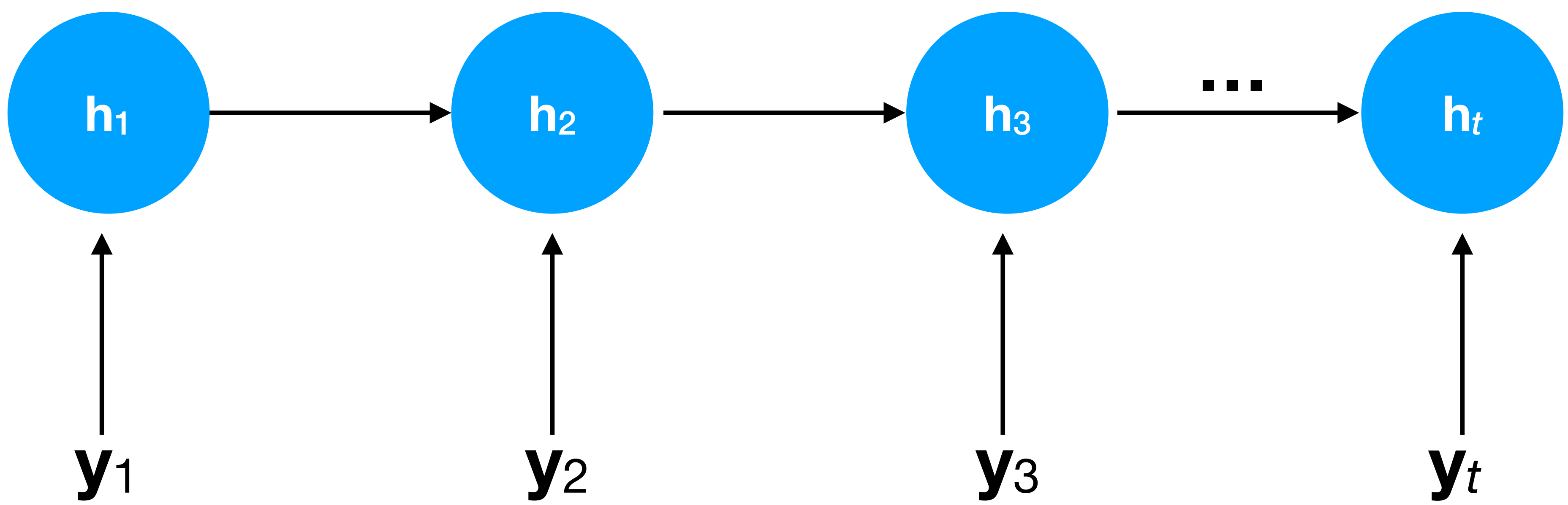
- For the vocabulary { u, v, w, . }, can you devise a scenario over 2 timesteps where beamsearch does *not* give the optimal answer?



Limitations of basic RNNs

- In practice, the hidden states { $\mathbf{h}_t$ } of basic RNNs have difficulty storing information long-term.

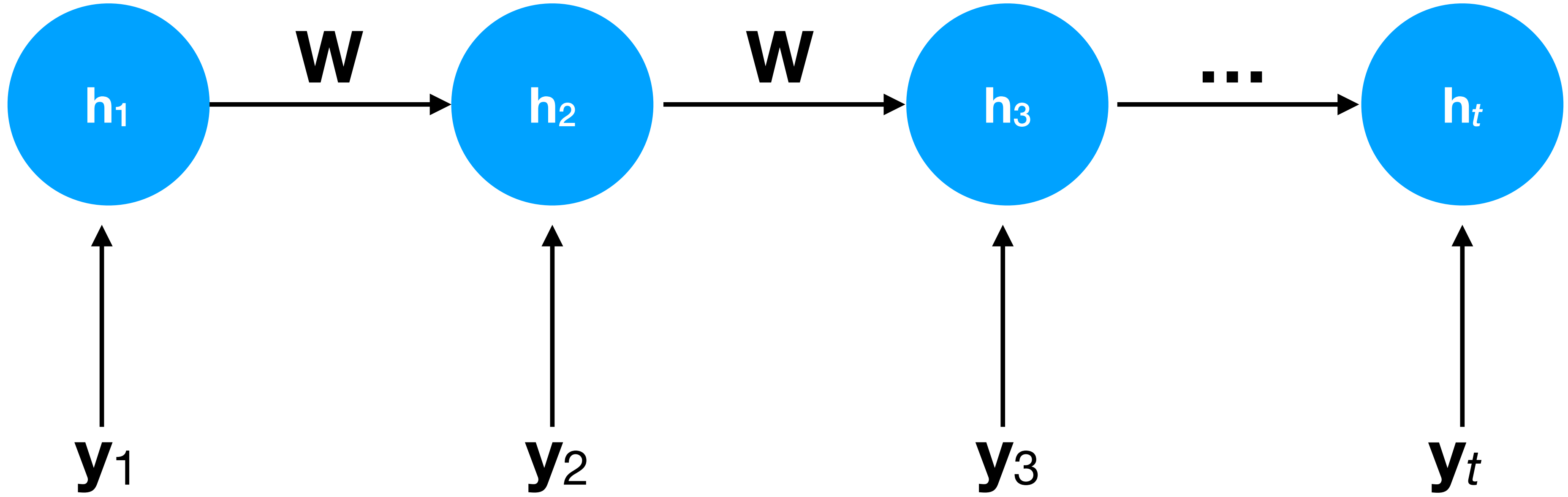
- Basic RNNs are also highly unstable to train.



Limitations of basic RNNs

- To see why, suppose we remove non-linearity and consider what happens when we repeatedly multiply $\mathbf{h}_t$ by $\mathbf{W}$.

$$\mathbf{h}_t = \mathbf{W}^t \mathbf{h}_1 + \ldots$$

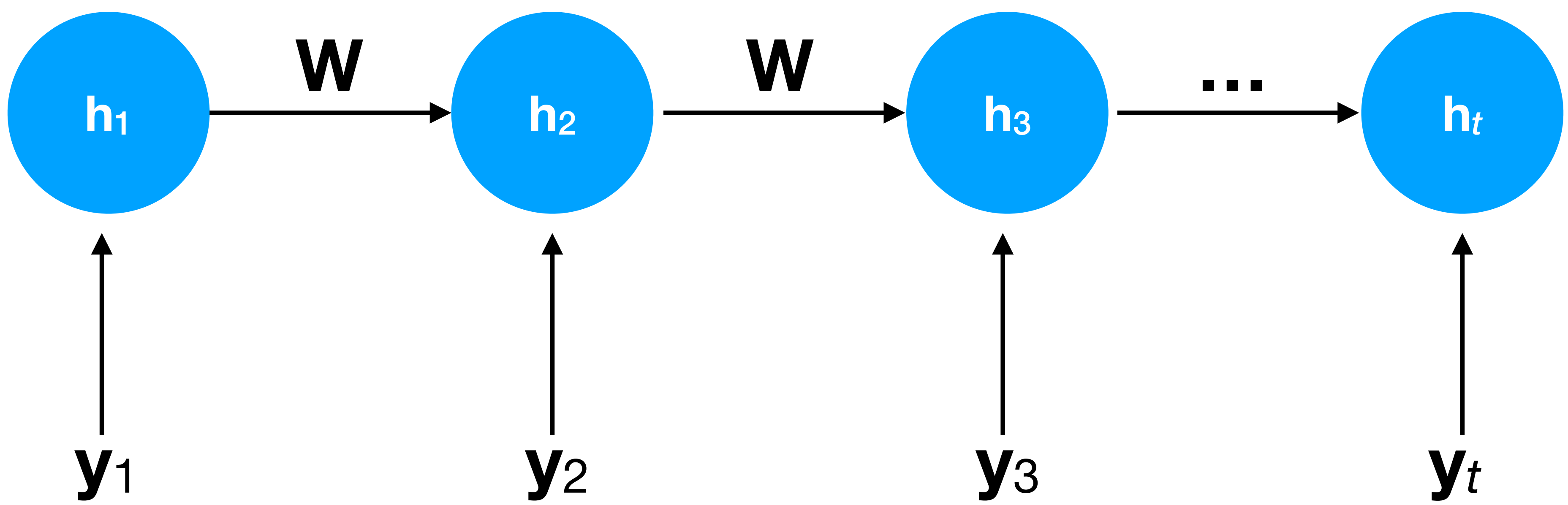


Limitations of basic RNNs

- To see why, suppose we remove non-linearity and consider what happens when we repeatedly multiply $\mathbf{h}_t$ by $\mathbf{W}$.

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}^t \mathbf{h}_1 + \dots \\ &= \mathbf{U}\mathbf{\Lambda}^t \mathbf{U}^\top \mathbf{h}_1 + \dots\end{aligned}$$

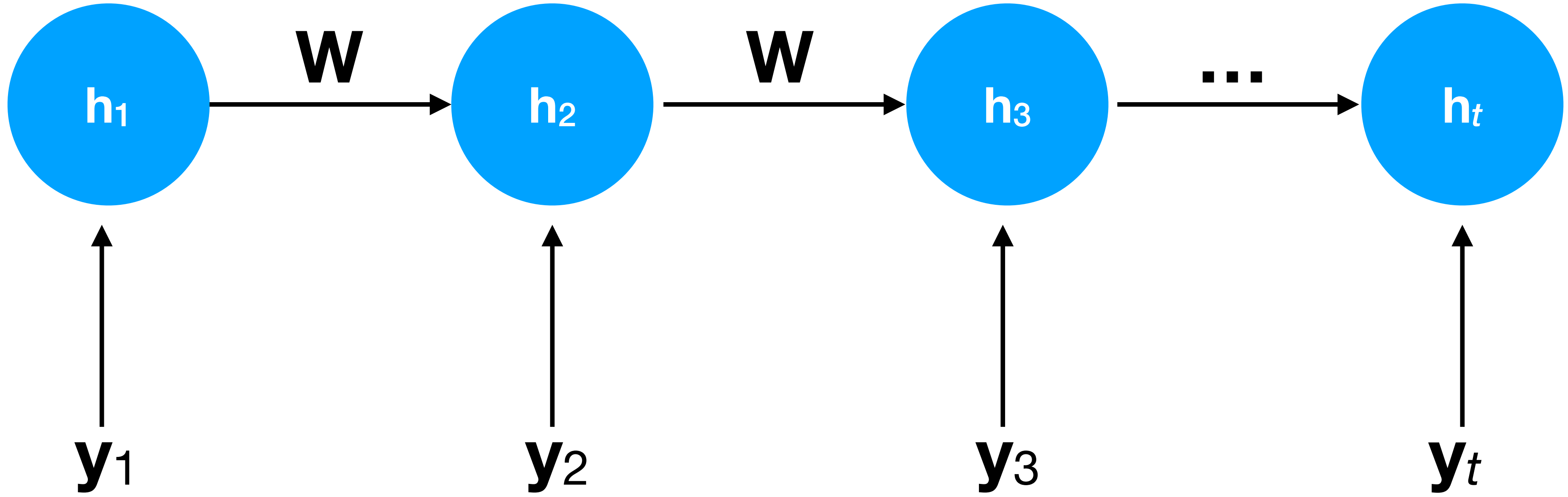
if **W** is diagonalizable



Limitations of basic RNNs

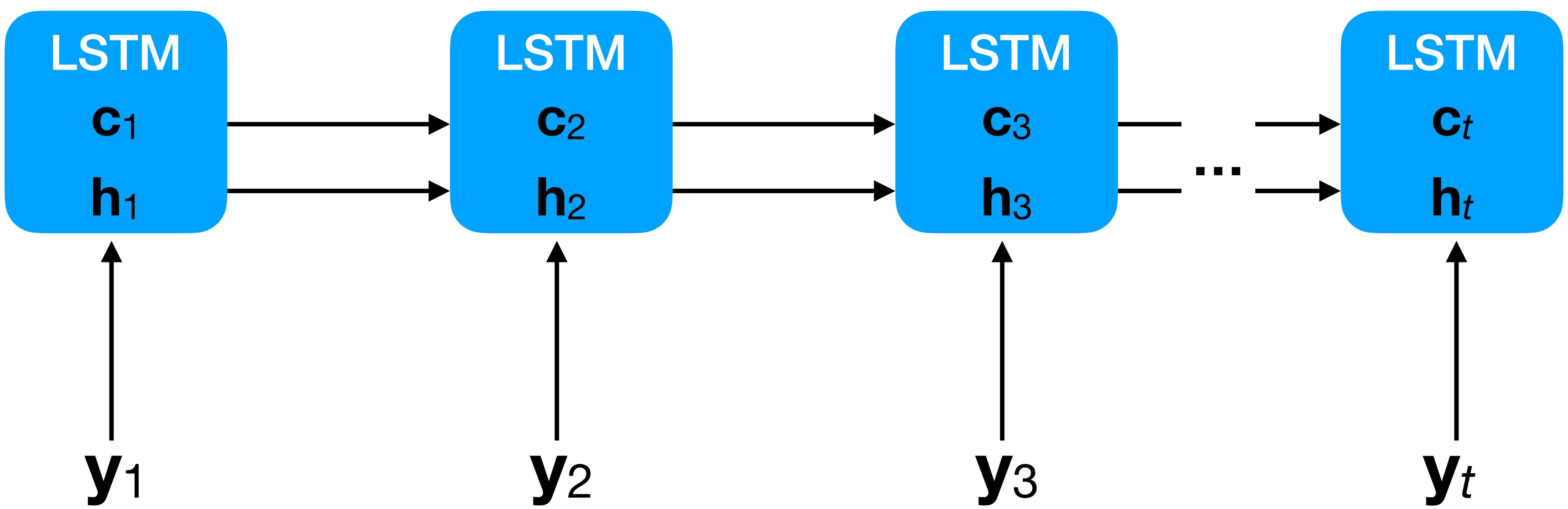
- Unless all **W**'s eigenvalues have magnitude ≈1, $\mathbf{h}_1$'s contribution to $\mathbf{h}_t$ will tend to explode to infinity or vanish to 0.

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}^t \mathbf{h}_1 + \ldots \\ &= \mathbf{U}\boldsymbol{\Lambda}^t \mathbf{U}^\top \mathbf{h}_1 + \ldots\end{aligned}$$



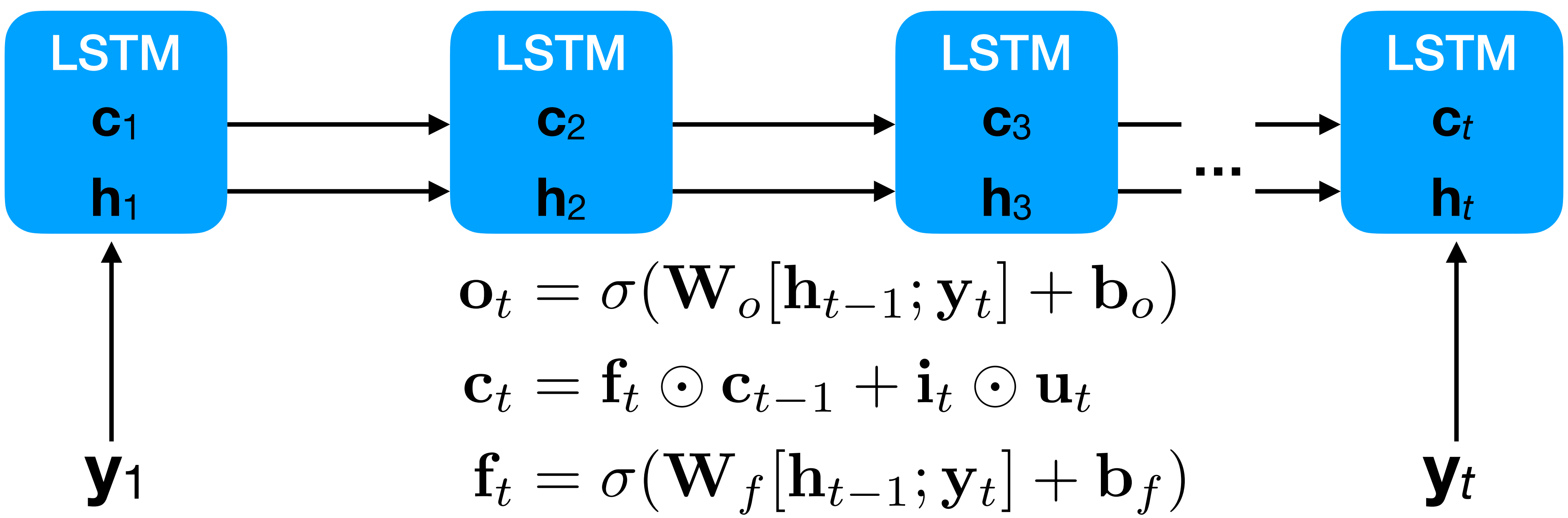
LSTM RNNs

- A **long short-term memory (LSTM) RNN** improves on this by making it easy to store information over long timespans.
- It contains both a hidden state $\mathbf{h}_t$ and a **cell state** $\mathbf{c}_t$, using the input $\mathbf{y}_t$.



LSTM RNNs

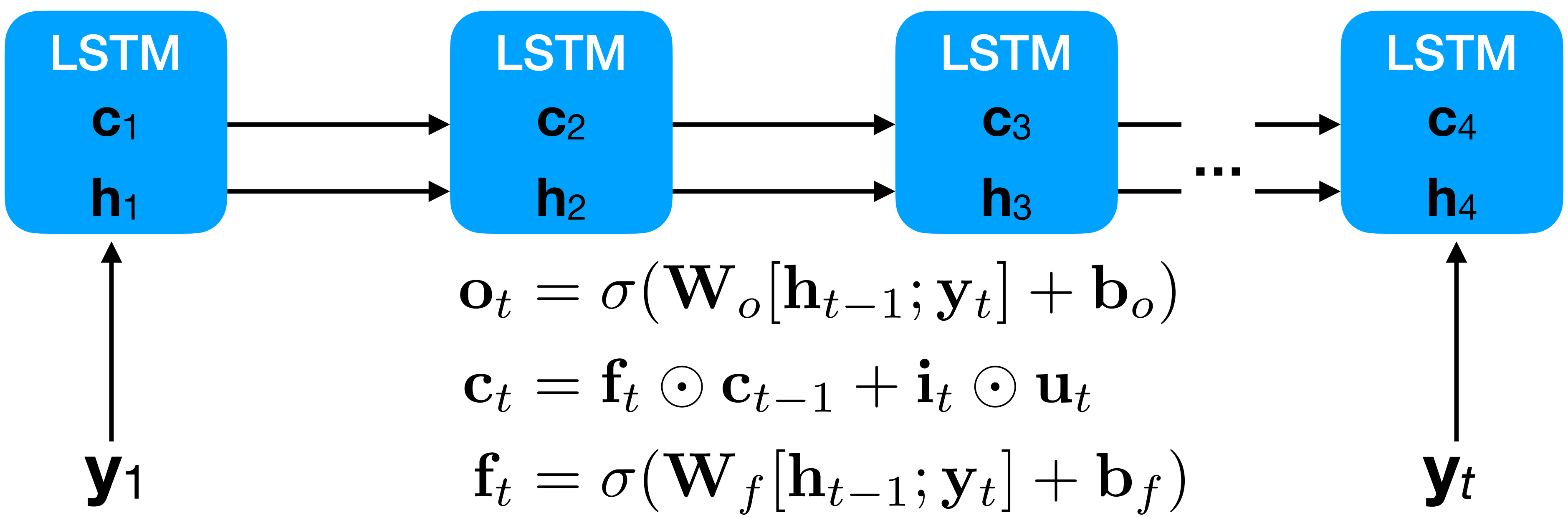
- **Input gates $\mathbf{i}_t$** control what parts of input $\mathbf{y}_t$ are allowed into $\mathbf{c}_t$.
- **Forgetting gates $\mathbf{f}_t$** control what parts of $\mathbf{c}_{t\text{-}1}$ are allowed into $\mathbf{c}_t$.
- **Output gates $\mathbf{o}_t$** control what parts of $\mathbf{c}_t$ are allowed into $\mathbf{h}_t$.



$$
\begin{aligned}
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1};\mathbf{y}_t] + \mathbf{b}_o) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{h}_{t-1};\mathbf{y}_t] + \mathbf{b}_f) \\
\mathbf{u}_t &= \tanh(\mathbf{W}_u[\mathbf{h}_{t-1};\mathbf{y}_t] + \mathbf{b}_u) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{h}_{t-1};\mathbf{y}_t] + \mathbf{b}_i) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

LSTM RNNs

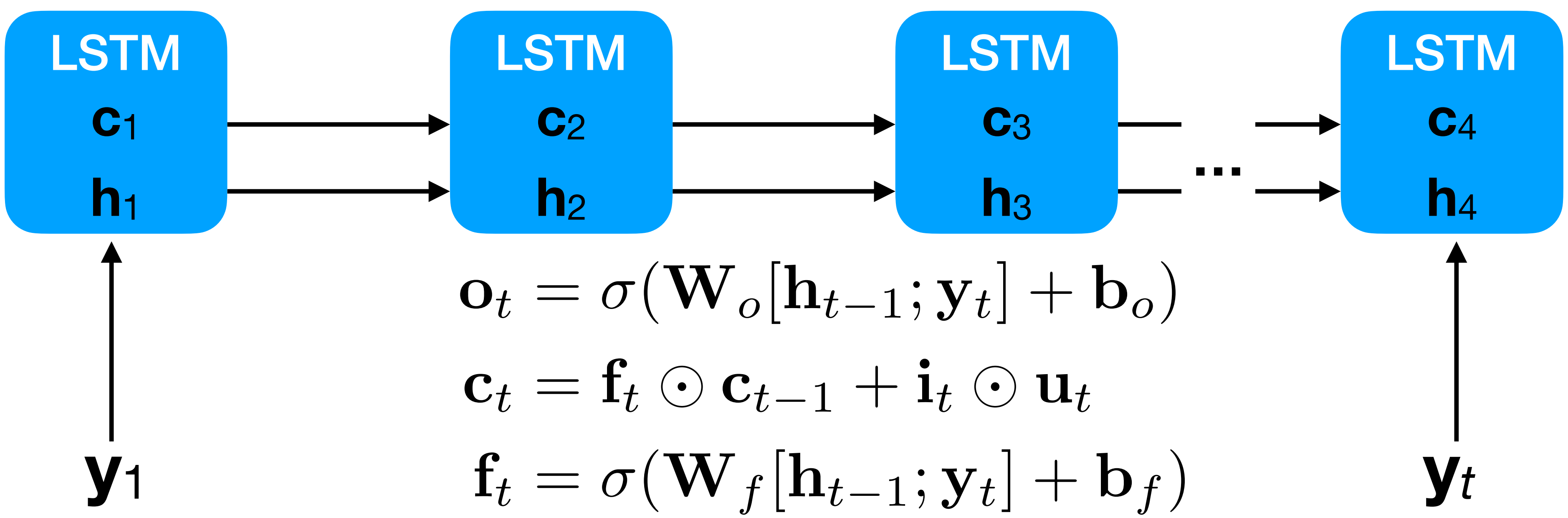
- If the $\mathbf{f}_t$=**1**, then $\mathbf{c}_t$ will maintain its value over time without exploding/vanishing.



$$
\begin{aligned}
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}; \mathbf{y}_t] + \mathbf{b}_o) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}; \mathbf{y}_t] + \mathbf{b}_f) \\
\mathbf{u}_t &= \tanh(\mathbf{W}_u[\mathbf{h}_{t-1}; \mathbf{y}_t] + \mathbf{b}_u) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}; \mathbf{y}_t] + \mathbf{b}_i) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

LSTM RNNs

- If the $\mathbf{f}_t$=**1**, then $\mathbf{c}_t$ will maintain its value over time without exploding/vanishing.
- Which components of $\mathbf{c}_t$ are “retrieved” from long-term storage depends on $\mathbf{y}_t$ and $\mathbf{h}_{t-1}$.



$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1};\mathbf{y}_t] + \mathbf{b}_o)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1};\mathbf{y}_t] + \mathbf{b}_f)$$

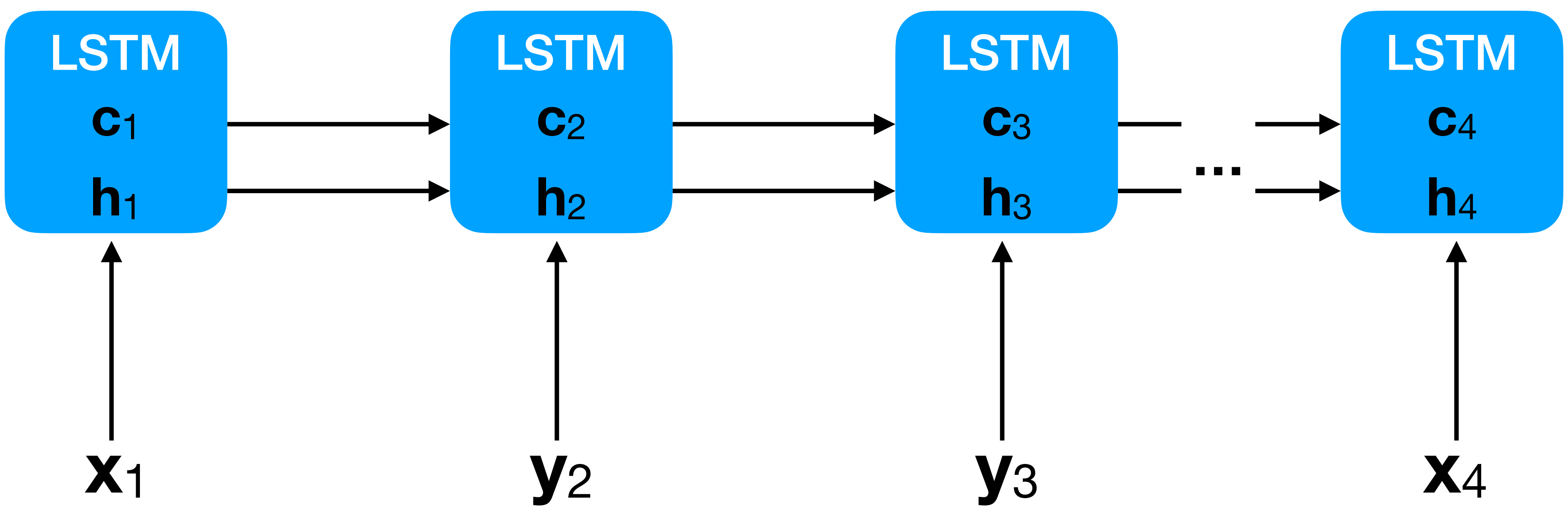
$$\mathbf{u}_t = \tanh(\mathbf{W}_u[\mathbf{h}_{t-1};\mathbf{y}_t] + \mathbf{b}_u)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1};\mathbf{y}_t] + \mathbf{b}_i)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

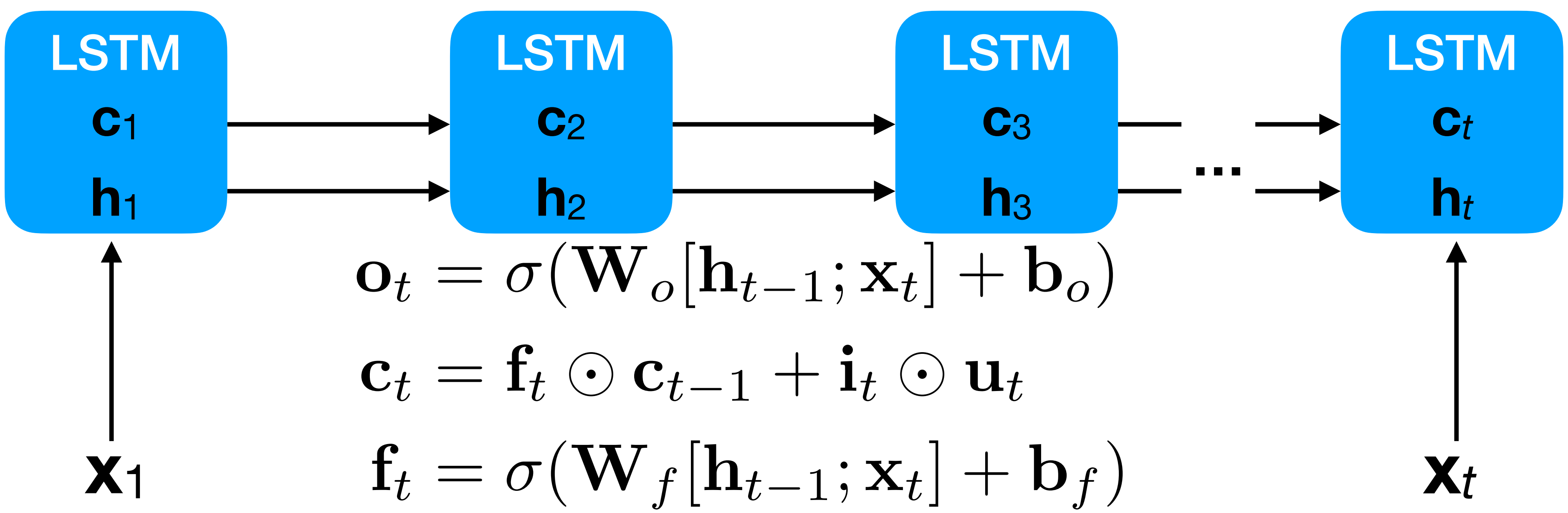
LSTM RNNs

- In practice, LSTMs are much easier to train than basic RNNs.
- The memory cell $\mathbf{c}_t$ selectively stores & forgets information from the input.
- It is still limited since it tries to summarize an entire history in one vector.



LSTM RNNs

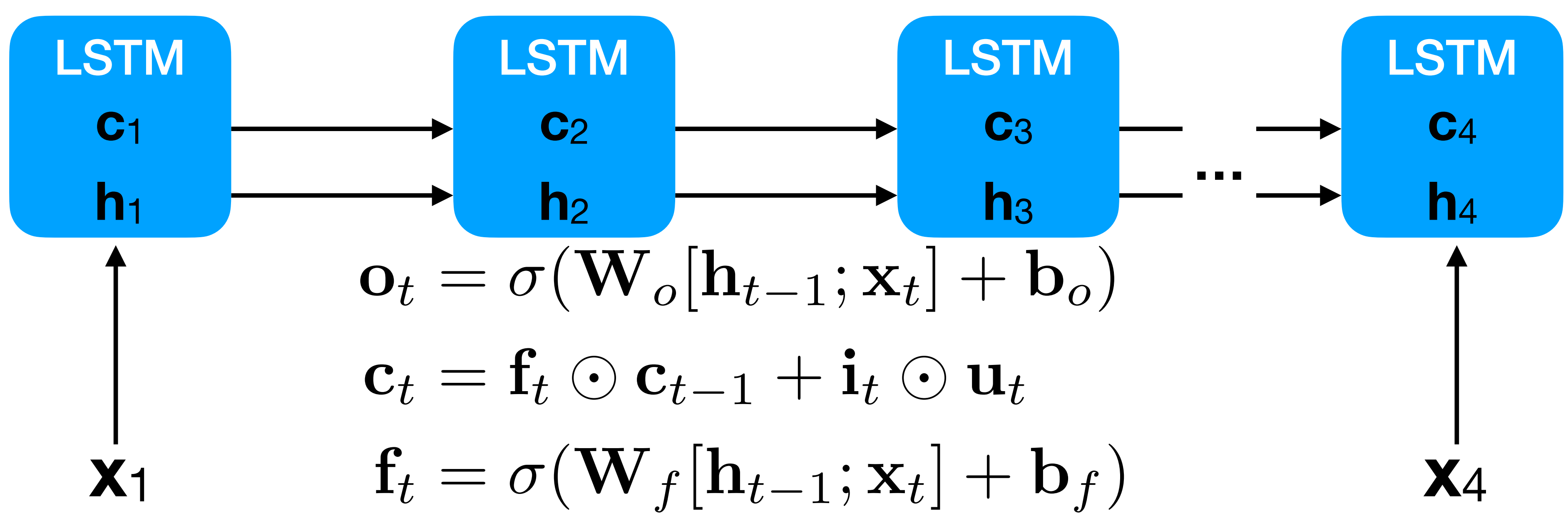
- **Input gates $\mathbf{i}_t$** control what parts of input $\mathbf{x}_t$ are allowed into $\mathbf{c}_t$.
- **Forgetting gates $\mathbf{f}_t$** control what parts of $\mathbf{c}_{t\text{-}1}$ are allowed into $\mathbf{c}_t$.
- **Output gates $\mathbf{o}_t$** control what parts of $\mathbf{c}_t$ are allowed into $\mathbf{h}_t$.



$$
\begin{aligned}
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_o) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_f) \\
\mathbf{u}_t &= \tanh(\mathbf{W}_u[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_u) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_i) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

LSTM RNNs

- If the $\mathbf{f}_t$=**1**, then $\mathbf{c}_t$ will maintain its value over time without exploding/vanishing.



$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_o)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_f)$$

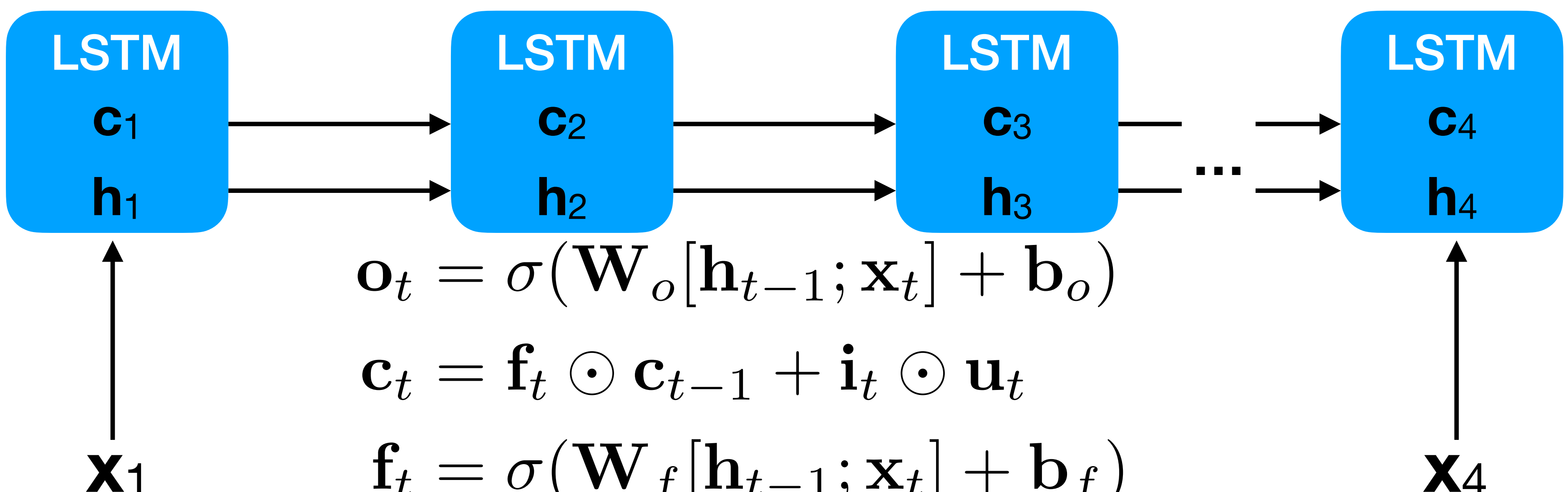
$$\mathbf{u}_t = \tanh(\mathbf{W}_u[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_u)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_i)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LSTM RNNs

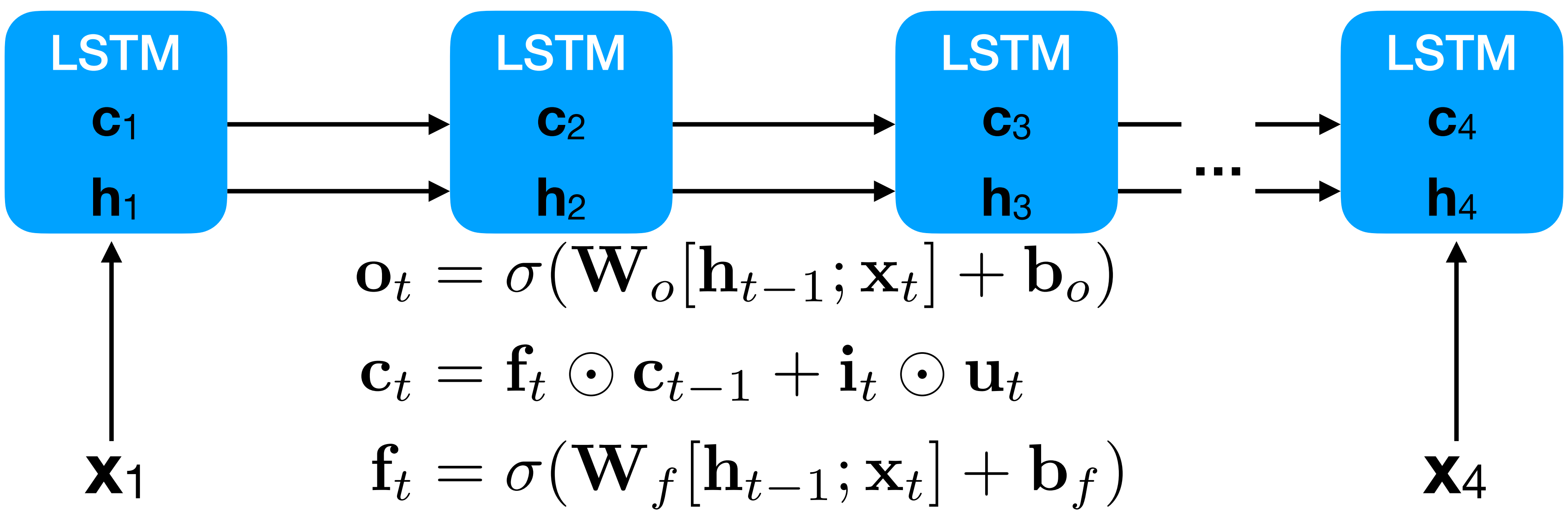
- If the $\mathbf{f}_t$=**1**, then $\mathbf{c}_t$ will maintain its value over time without exploding/vanishing.
- Which components of $\mathbf{c}_t$ are "retrieved" from long-term storage depends on $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$.



$$
\begin{aligned}
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_o) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_f) \\
\mathbf{u}_t &= \tanh(\mathbf{W}_u[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_u) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_i) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

LSTM RNNs

- In practice, LSTMs are much easier to train than basic RNNs.
- However, the memory cell $\mathbf{c}_t$ provides only limited storage since it summarizes an entire memory history into a single vector.



$$
\begin{aligned}
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_o) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_f) \\
\mathbf{u}_t &= \tanh(\mathbf{W}_u[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_u) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_i) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

Prefix-sums

Sum the numbers

- The simplest way to sum up *n* numbers is sequential (with $O(n)$ time):

```
total = 0
for i in range(n):
     total += nums[i]
```

total = 13

3	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Sum the numbers

- However, we can also parallelize this by computing sums in a tree:

Rule: parent = child1 + child2

3	1	0	2	4	1	0	2

Sum the numbers

- However, we can also parallelize this by computing sums in a tree:

Rule: parent = child1 + child2

4		2		5		2	
3	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Sum the numbers

- However, we can also parallelize this by computing sums in a tree:

Rule: parent = child1 + child2

6 7

4 2 5 2

3	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Sum the numbers

- However, we can also parallelize this by computing sums in a tree:

Rule: parent = child1 + child2

13

6 7

4 2 5 2

3	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Sum the numbers

- Now the computational cost is only $O(\log n)$.

13

6 7

4 2 5 2

3	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- What if we also want to compute all the prefix-sums?

p	3	4	4	6	10	11	11	13
x	3	1	0	2	4	1	0	2

Prefix-sums

- What if we also want to compute all the prefix-sums?

- A simple sequential algorithm is:

```
p[0] = x[0]
for i in range(1, n):
    p[i] = p[i-1] + x[i]
```

- Total work is $O(n)$ and span is $O(n)$.

p	3	4	4	6	10	11	11	13
x	3	1	0	2	4	1	0	2

Prefix-sums

- Can we parallelize this?

p	3	4	4	6	10	11	11	13
x	3	1	0	2	4	1	0	2

Prefix-sums

- Can we parallelize this? Yes, using a two-pass recursion.

p	3	4	4	6	10	11	11	13
x	3	1	0	2	4	1	0	2

Prefix-sums

- We first recursively compute sums pairwise.

13

6 7

4 2 5 2

p								
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- We first recursively compute sums pairwise.

- At this point, we can set `p[7]=13`.

13

6 7

4 2 5 2

p								**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- We then descend down the tree and fill in the entries of **p**.

13

6 7

4 2 5 2

p								**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- We then descend down the tree and fill in the entries of **p**.
- From 13's left child, we know the sum up to i=4 must be 6.

13

6 7

4 2 5 2

p								**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- We then descend down the tree and fill in the entries of **p**.
- From 13’s left child, we know the sum up to i=4 must be 6.
- Thus, `p[3]=6`.

13

6 7

4 2 5 2

p				**6**				**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- Recursing into 13’s right sub-tree (7)

13

6 **7**

4 2 5 2

p				**6**				**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- Recursing into 13’s right sub-tree (7), we know that the sum of `x[4:6]` was 5.

13

6 **7**

4 2 5 2

p				**6**				**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- Recursing into 13’s right sub-tree (7), we know that the sum of `x[4:6]` was 5.
- Thus, `p[6]=p[4]+5`.

13

6 **7**

4 2 5 2

p				**6**		**11**		**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- By similar logic, we can compute:

13

6 7

4 2 **5** 2

p				**6**	**10**	**11**		**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- By similar logic, we can compute:

13

6 7

4 2 5 **2**

p				6	10	11	11	13
x	3	1	0	2	4	1	0	2

Prefix-sums

- By similar logic, we can compute:

13

6 7

4 2 5 2

p	3	4	4	6	10	11	11	13
x	3	1	0	2	4	1	0	2

Prefix-sums

- Using this **parallel scan**, we reduce the span to $O(\log n)$.

13

6 7

4 2 5 2

p	**3**	**4**	**4**	**6**	**10**	**11**	**11**	**13**
x	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- Clearly, we can generalize this approach to the prefix-sum of *vectors* by performing (and parallelizing) the scan channel-wise, e.g.:

P	**3, -1**	**4, 3**	**4, 5**	**6, 3**	**10, 3**	**11, 6**	**11, 7**	**13, 8**
X	**3, -1**	**1, 4**	**0, 2**	**2, -2**	**4, 0**	**1, 3**	**0, 1**	**2, 1**

Prefix-sums

- We can also generalize it to the prefix-sums of *multiples* of the **X**, i.e., $p_i = \sum_{i'=1}^{i} (a \times x_i) = p_{i-1} + a \times x_i$.

- The example below is for *a*=2.

P	**6, -2**	**8, 6**	**8, 10**	**12, 6**	**12, 6**	**22, 12**	**22, 14**	**26, 16**
X	**3, -1**	**1, 4**	**0, 2**	**2, -2**	**4, 0**	**1, 3**	**0, 1**	**2, 1**

Prefix-sums

- We can further generalize this idea from addition (+) to **any binary associative operator $\oplus$**.

- $\oplus$ is associative iff $x \oplus (y \oplus z) = (x \oplus y) \oplus z \quad \forall x, y, z$

- Examples include scalar addition, scalar multiplication, matrix addition, matrix multiplication.

P	**0, -1**	**4, 3**	**4, 5**	**6, 3**	**6, 3**	**11, 6**	**11, 7**	**13, 8**
X	**3, -1**	**1, 4**	**0, 2**	**2, -2**	**4, 0**	**1, 3**	**0, 1**	**2, 1**

Prefix-sums

- What if we want to compute $p_i = b \times p_{i-1} + x_i$ for all *i*?

P								
X	3	1	0	2	4	1	0	2

Exercise

- What if we want to compute $p_i = b \times p_{i-1} + x_i$ for all *i*?
- Manually compute the answers for *b*=2:

P								
X	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Solution

- What if we want to compute $p_i = b \times p_{i-1} + x_i$ for all *i*?
- Manually compute the answers for *b*=2:

P	3	7	14	30	64	129	258	518
X	3	1	0	2	4	1	0	2

Prefix-sums

- Can we parallelize $p_i = b \times p_{i-1} + x_i$ for all *i*?
- We need an associative operator. Maybe $x \oplus y = bx + y$?
- Let's try it out…

P	**3**	**7**	**14**	**30**	**64**	**129**	**258**	**518**
X	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- Our operator: $x \oplus y = bx + y$
  - $3 \oplus 1 = b * 3 + 1 = 2 * 3 + 1 = 7$

P	**3**	**7**
X	**3**	**1**

Prefix-sums

- Our operator: $x \oplus y = bx + y$
  - $3 \oplus 1 = b * 3 + 1 = 2 * 3 + 1 = 7$
  - $7 \oplus 0 = 2 * 7 + 0 = 14$

P	**3**	**7**	**14**
X	**3**	**1**	**0**

Prefix-sums

- Our operator: $x \oplus y = bx + y$
  - $3 \oplus 1 = b * 3 + 1 = 2 * 3 + 1 = 7$
  - $7 \oplus 0 = 2 * 7 + 0 = 14$
- Looking good so far!

P	3	7	14	30	64	129	258	518
X	3	1	0	2	4	1	0	2

Prefix-sums

- Unfortunately, $x \oplus y = bx + y$ is not associative:
  - $1 \oplus 0 = 2 * 1 + 0 = 2$
  - $3 \oplus 2 = 2 * 3 + 2 = 8$
- Hence, $(3 \oplus 1) \oplus 0 \neq 3 \oplus (1 \oplus 0)$.

P	3	7	14	30	64	129	258	518
X	3	1	0	2	4	1	0	2

Prefix-sums

- Unfortunately, $x \oplus y = bx + y$ is not associative:
  - $1 \oplus 0 = 2 * 1 + 0 = 2$
  - $3 \oplus 2 = 2 * 3 + 2 = 8$ Needed to multiply by b^2!
- Hence, $(3 \oplus 1) \oplus 0 \neq 3 \oplus (1 \oplus 0)$.

P	3	7	14	30	64	129	258	518
X	3	1	0	2	4	1	0	2

Prefix-sums

- We need a way of storing the current power of *b*.
- Hence, we will use 2 numbers (u,v) to represent each step of the computation.
- Operator: $(u, v) \oplus (u', v') = (b^{v'}u + u', v + v')$
- We initialize $(u, v)=(x_i, 1)$ for each i.

P	**3**	**7**	**14**	**30**	**64**	**129**	**258**	**518**
X	**3**	**1**	**0**	**2**	**4**	**1**	**0**	**2**

Prefix-sums

- Operator: $(u, v) \oplus (u', v') = (b^{v'}u + u', v + v')$
  - $(3,1) \oplus (1,1) = (2^1 * 3 + 1,\ 2) = (7,2)$
  - $(7,2) \oplus (0,1) = (2^1 * 7 + 0,3) = (14,3)$

P	3	7	14	30	64	129	258	518
X	3	1	0	2	4	1	0	2

Prefix-sums

- Operator: $(u, v) \oplus (u', v') = (b^{v'}u + u', v + v')$
  - $(1,1) \oplus (0,1) = (2^1,2) = (2,2)$
  - $(3,1) \oplus (2,2) = (2^2 * 3 + 2,3) = (14,3)$
- This operator is associative.

P	3	7	14	30	64	129	258	518
X	3	1	0	2	4	1	0	2

Prefix-sums

- We can thus use a parallel scan to compute $p_i = b \times p_{i-1} + x_i$ with span $O(\log n)$.

(518,8)

(30,4) (38,4)

(7,2) (2,2) (9,2) (2,2)

P	3	7	14	30	64	129	258	518
X	3	1	0	2	4	1	0	2

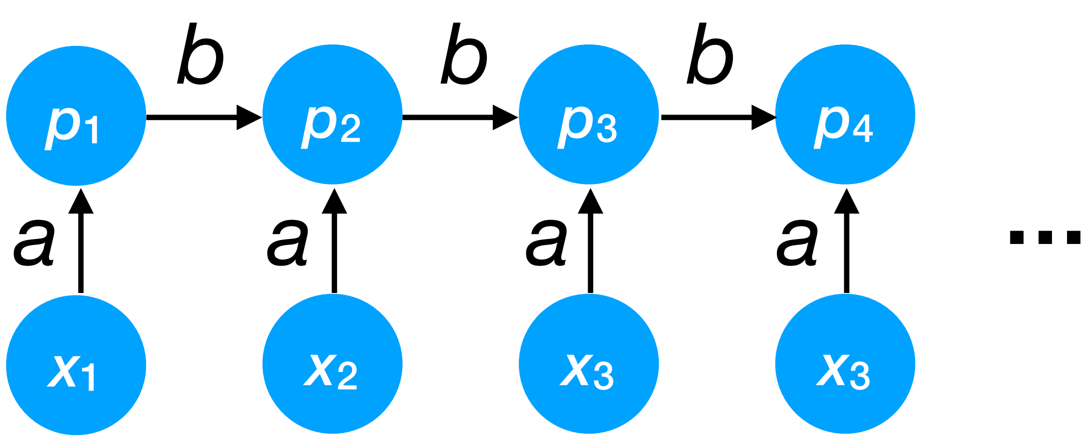
Prefix-sums

- Putting all the parts together, we can use a parallel scan to compute prefix-sums of the form $p_i = b \times p_{i-1} + ax_i$.

P	3	7	14	30	64	129	258	518
X	3	1	0	2	4	1	0	2

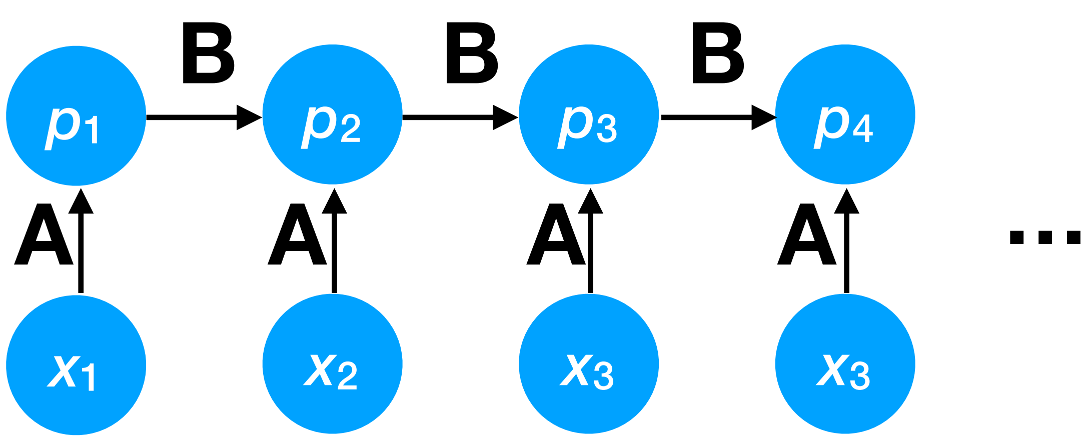
Prefix-sums

- Putting all the parts together, we can use a parallel scan to compute prefix-sums of the form $p_i = b \times p_{i-1} + ax_i$.



Prefix-sums

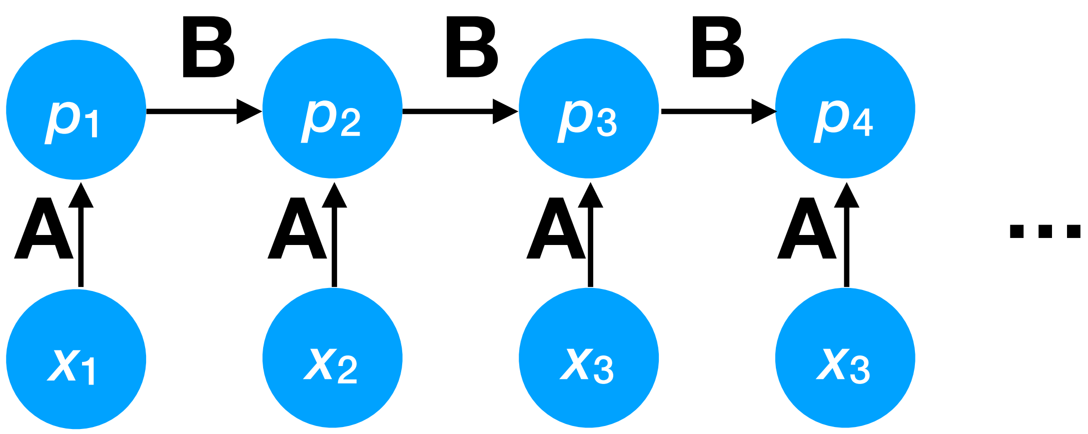
- Putting all the parts together, we can use a parallel scan to compute prefix-sums of the form $\mathbf{p}_i = \mathbf{B}\mathbf{p}_{i-1} + \mathbf{A}x_i$.



Prefix-sums

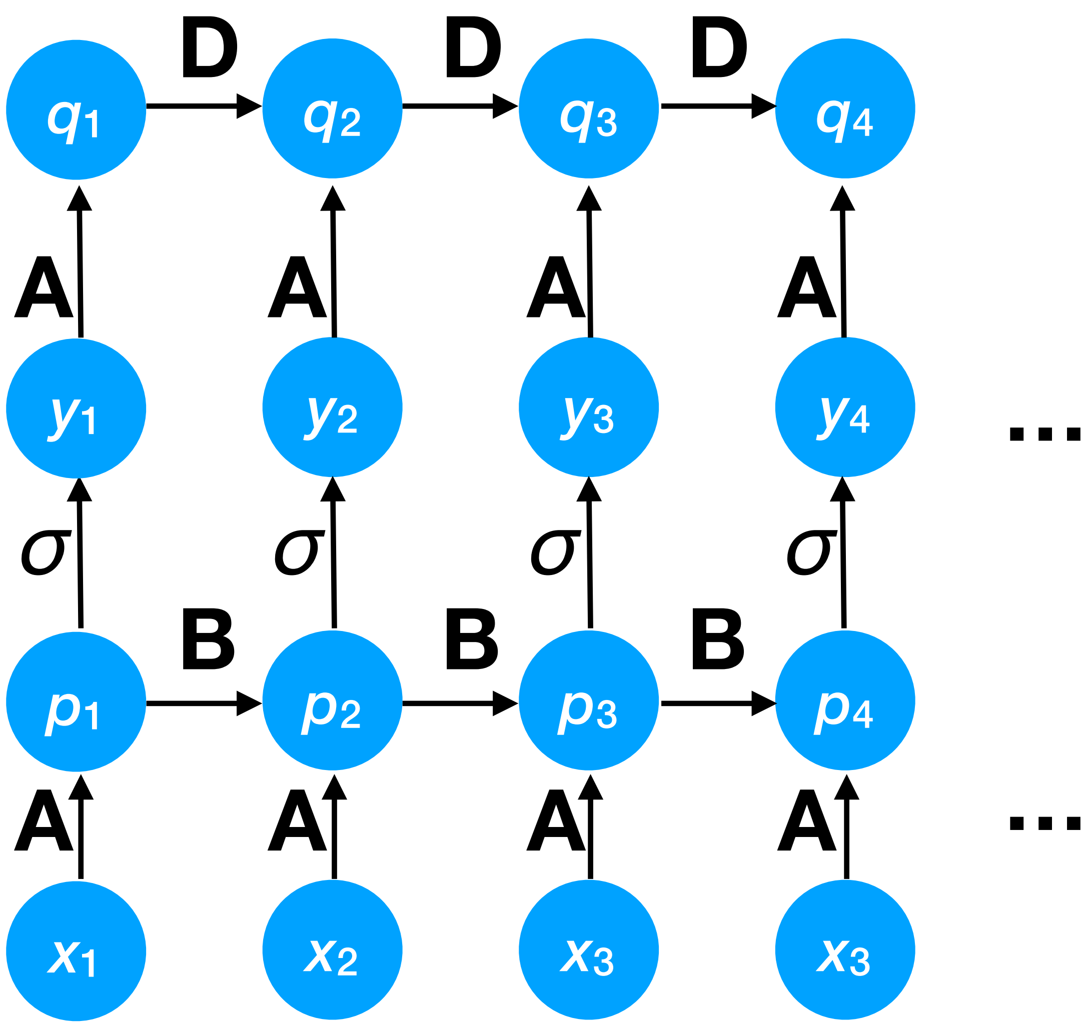
- Putting all the parts together, we can use a parallel scan to compute prefix-sums of the form $\mathbf{p}_i = \mathbf{B}\mathbf{p}_{i-1} + \mathbf{A}x_i$.

- This is equivalent to a linear RNN.



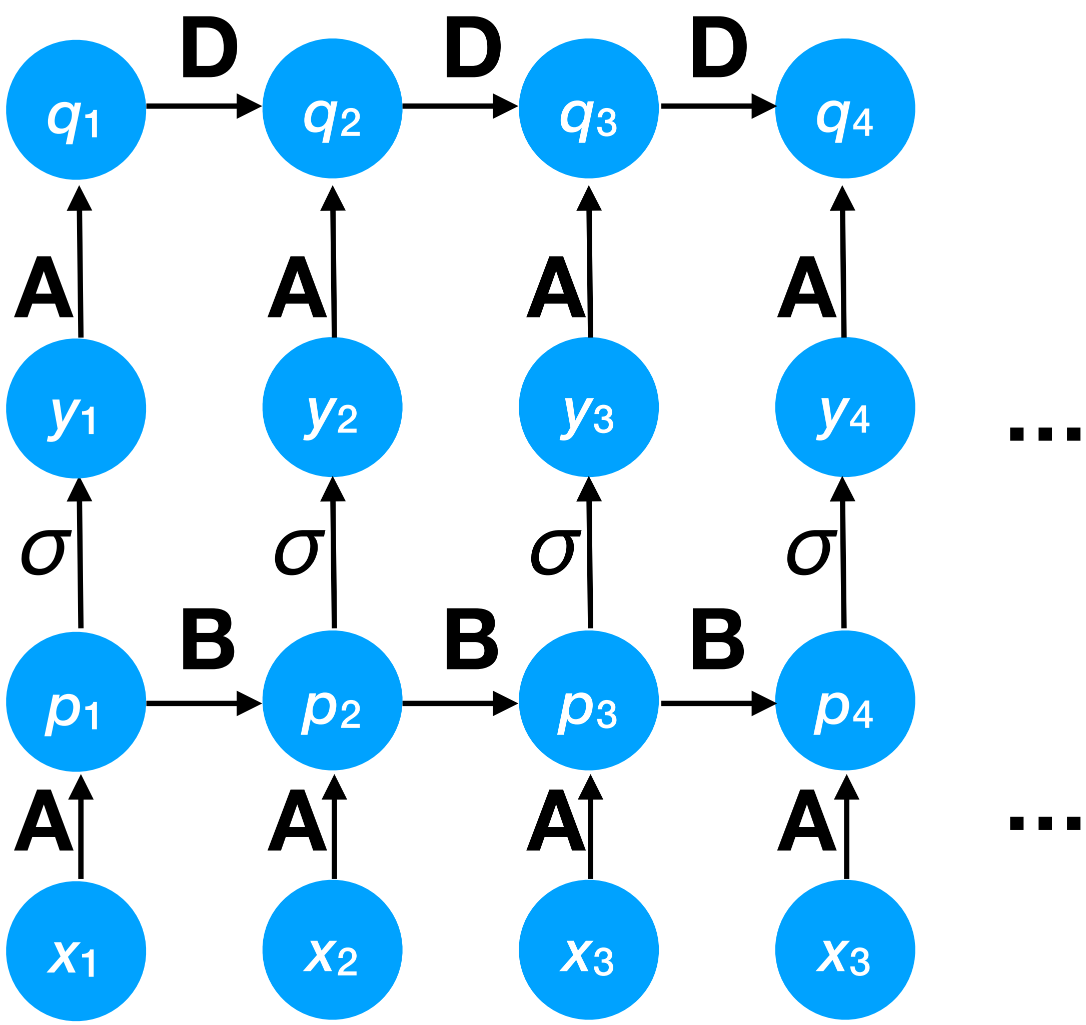
Prefix-sums

- While not sufficient by itself, it can be a powerful component in a larger NN with nonlinearities, e.g.:



Prefix-sums

- It is also one of the key ideas behind structured state space models (S3), e.g., Mamba.



Exercises

Exercise 1

- Suppose that, in a diffusion, we want to "merge" two faces $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ at some timestep t by computing $\mathbf{z}_t^{(1)}$ and $\mathbf{z}_t^{(2)}$, averaging to get $\mathbf{z}_t=(\mathbf{z}_t^{(1)}+\mathbf{z}_t^{(2)})/2$, and then sampling $\mathbf{x} \sim P(\mathbf{x} \mid \mathbf{z}_t)$?

- What is (slightly) wrong with this?

Exercise 2

- Consider the statement:
  - The fundamental goal of a VAE is to minimize reconstruction error while also keeping the KL divergence between $Q(\mathbf{z} \mid \mathbf{x})$ and $P(\mathbf{z})$ low.
- In which sense is this statement true?
- In which sense is it false?