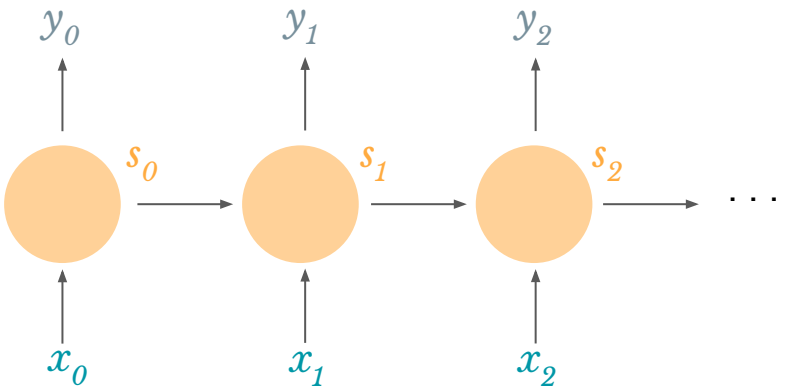
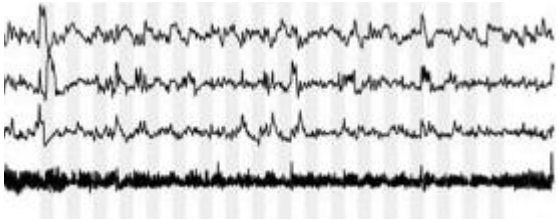
Sequence Modeling with Neural Networks

Harini Suresh

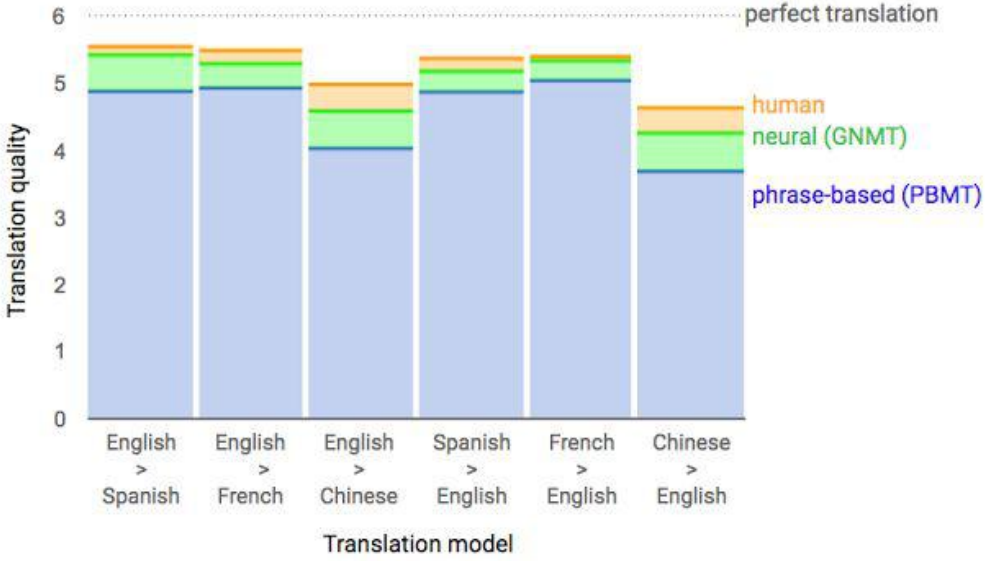


What is a sequence?

- “This morning I took the dog for a walk.” sentence
-  medical signals
-  speech waveform

Successes of deep models

Machine translation



Question Answering

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.

Super Bowl 50 decided the NFL champion for what season?
Ground Truth Answers: 2015 | the 2015 season | 2015
Prediction: 2015

Left:
https://research.googleblog.com/2016/09/a-neural-network-for-machine.html
Right:
https://rajpurkar.github.io/SQuAD-explorer/

Successes of deep models

a sequence modeling problem:
predict the next word

a sequence modeling problem

“This morning I took the dog for a walk.”

a sequence modeling problem

"This morning I took the dog for a walk."

given these words

a sequence modeling problem

"This morning I took the dog for a walk."

given these words

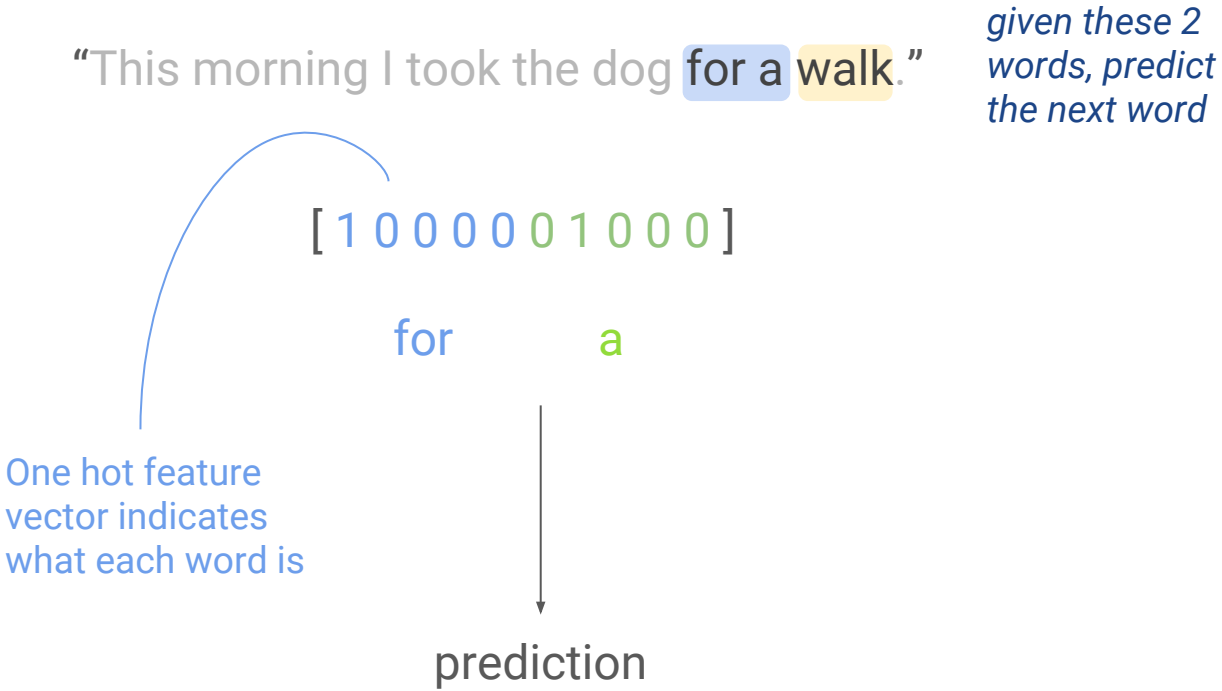
predict what comes next?

idea: use a fixed window

"This morning I took the dog for a walk."

given these 2 words, predict the next word

idea: use a fixed window

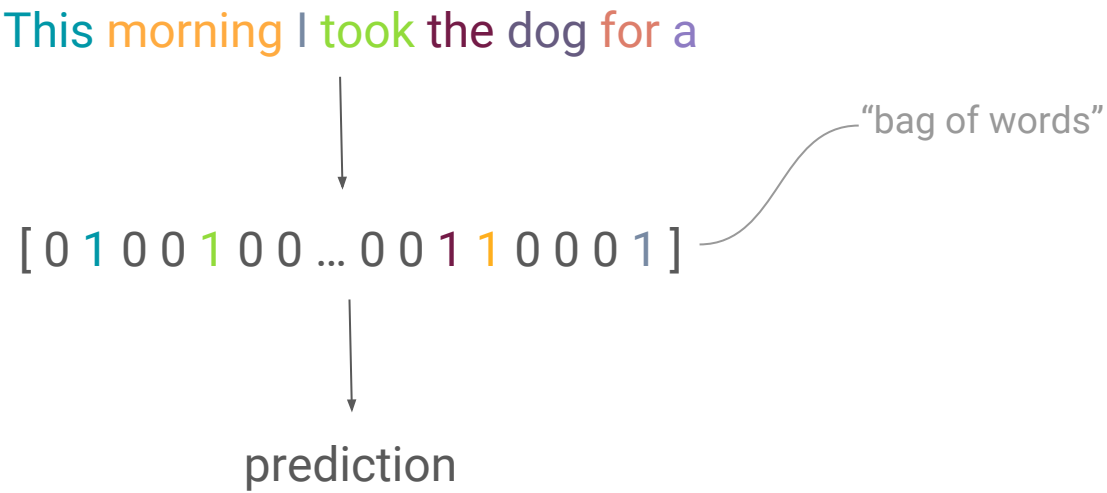


problem: we can't model long-term dependencies

"In France, I had a great time and I learnt some of the _____ language."

We need information from the far past and future to accurately guess the correct word.

idea: use entire sequence, as a set of counts



problem: counts don’t preserve order

problem: counts don’t preserve order

“The food was good, not bad at all.”

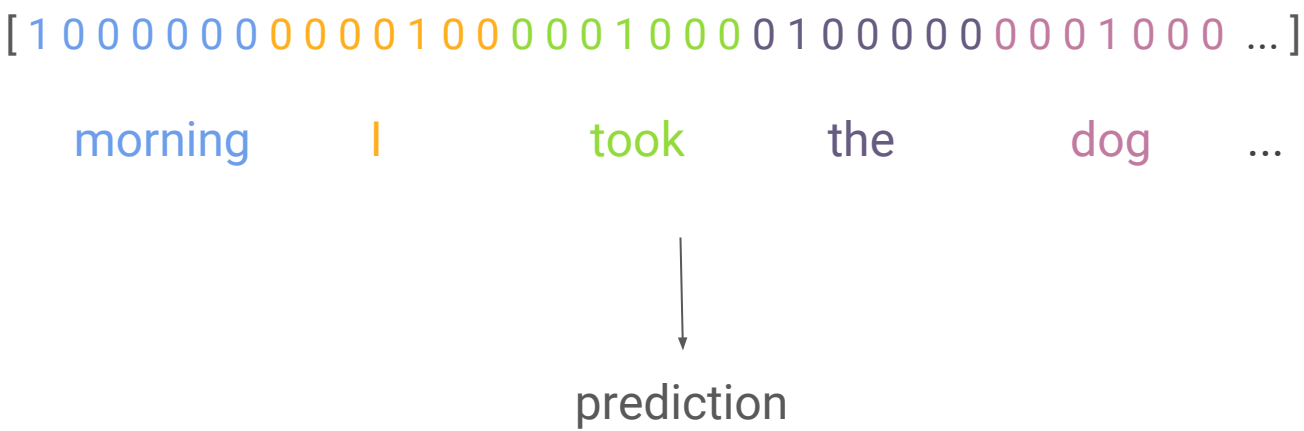
vs

“The food was bad, not good at all.”

idea: use a really big fixed window

"This morning I took the dog for a walk."

given these 7 words, predict the next word



problem: no parameter sharing

this morning

[1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 ...]

each of these inputs has a *separate parameter*

problem: no parameter sharing

this morning

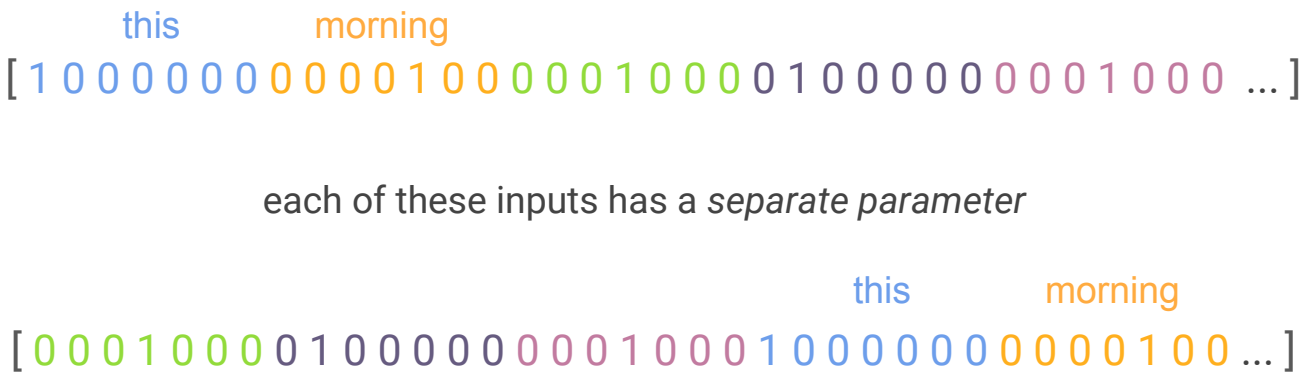
[1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 ...]

each of these inputs has a *separate parameter*

this morning

[0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 ...]

problem: no parameter sharing



things we learn about the sequence **won't transfer** if they appear **at different points** in the sequence.

to model sequences, we need:

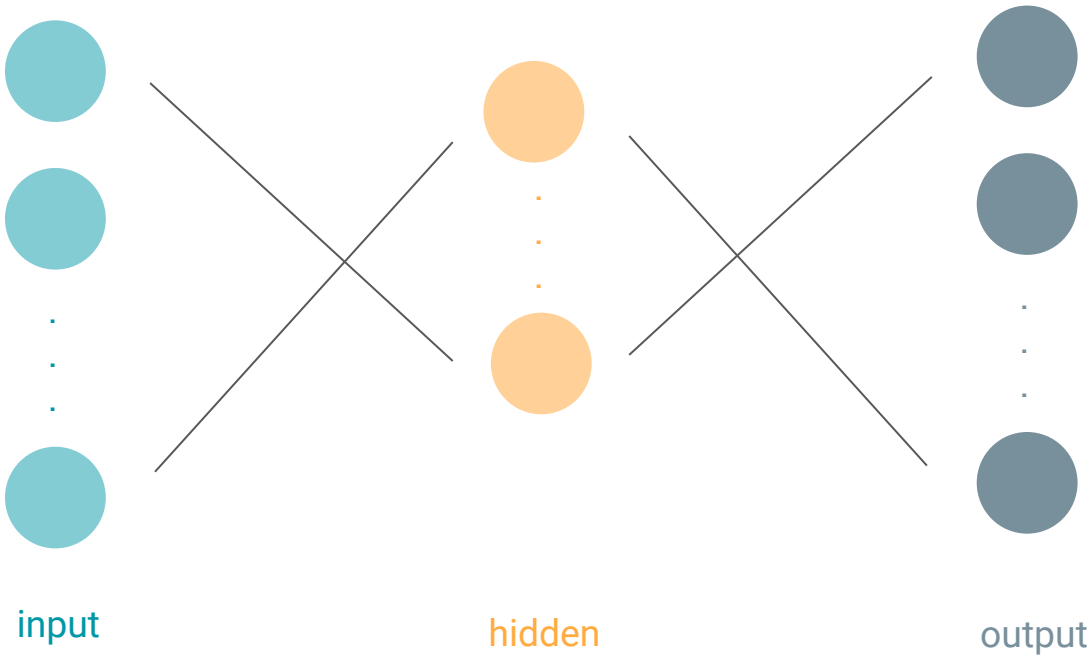
1. to deal with variable-length sequences
2. to maintain sequence order
3. to keep track of long-term dependencies
4. to share parameters across the sequence

to model sequences, we need:

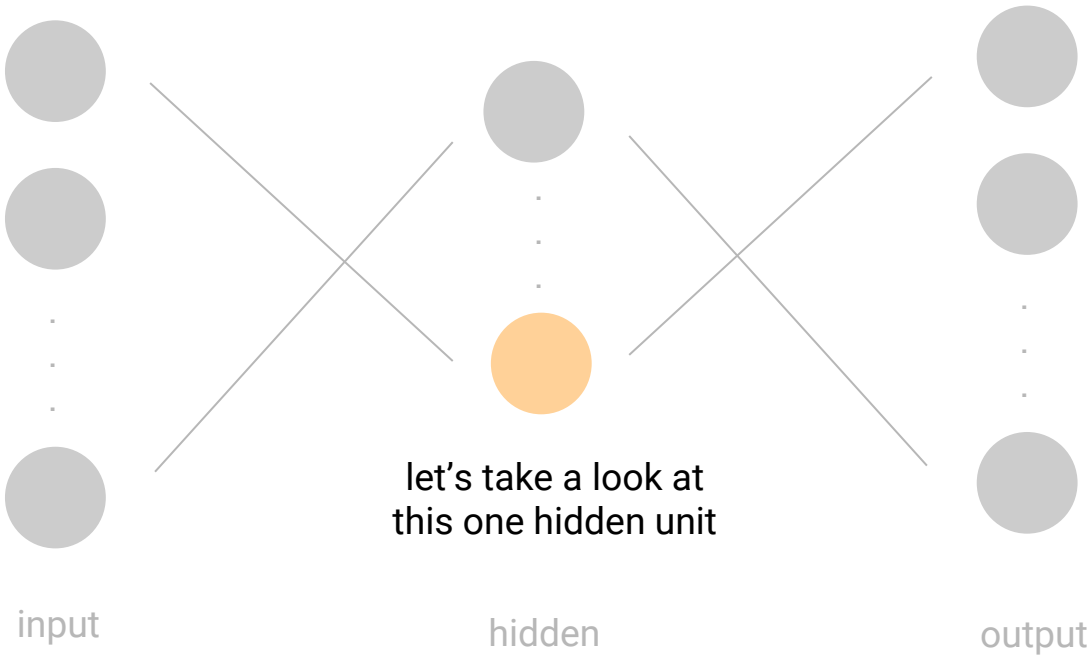
1. to deal with variable-length sequences
2. to maintain sequence order
3. to keep track of long-term dependencies
4. to share parameters across the sequence

let's turn to **recurrent neural networks**.

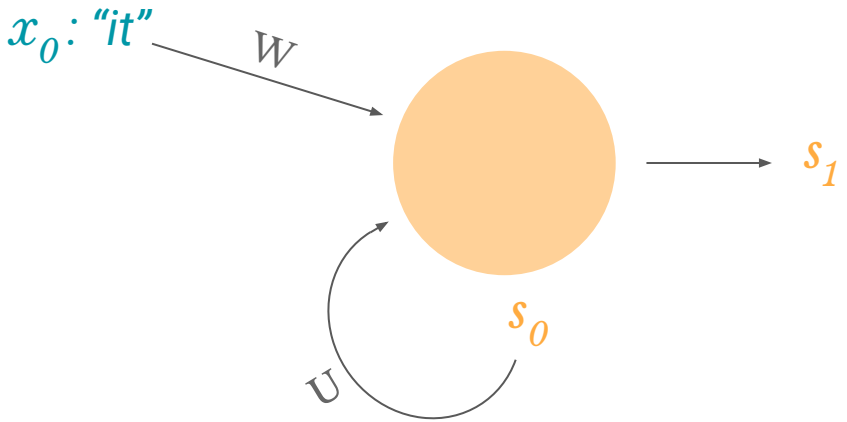
example network:



example network:



RNNS **remember** their previous state:

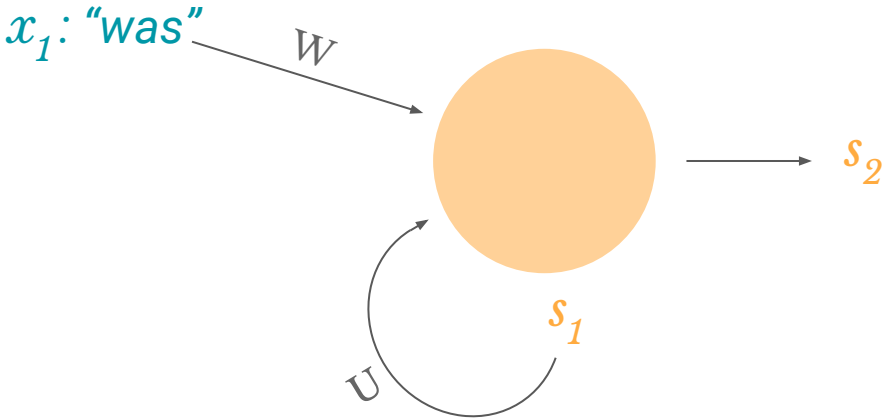


x_0 : vector representing first word
s_0 : cell state at $t = 0$ (some initialization)
s_1 : cell state at $t = 1$

$$s_1 = tanh(Wx_0 + Us_0)$$

$W,\ U$: weight matrices

RNNS **remember** their previous state:

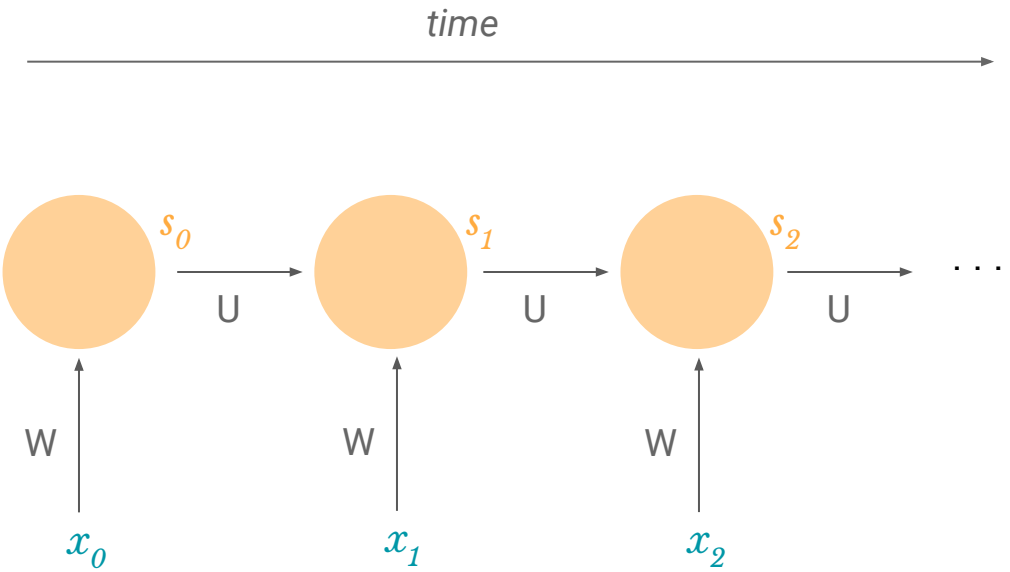


x_1 : vector representing second word
s_1 : cell state at $t = 1$
s_2 : cell state at $t = 2$

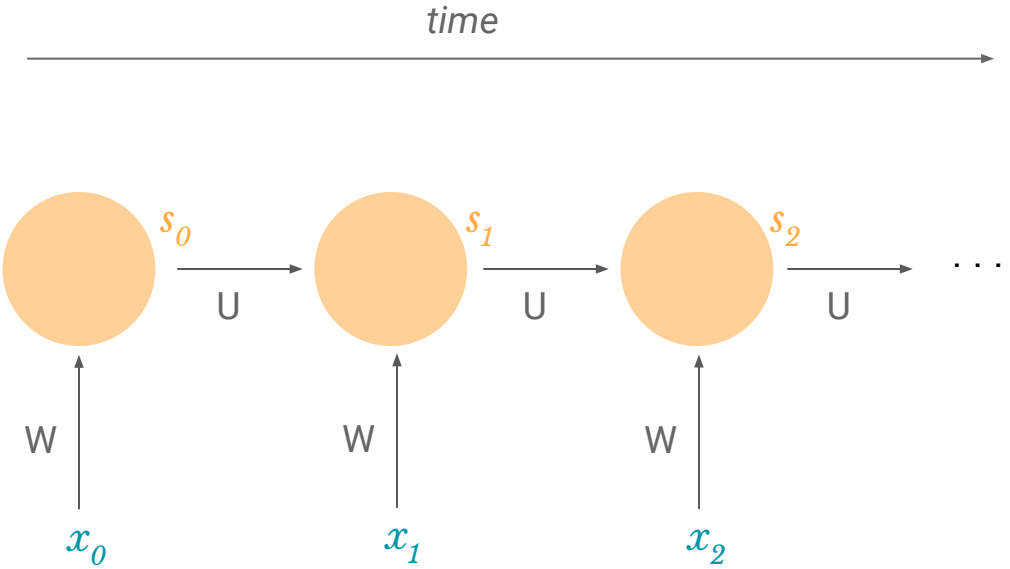
$$s_2 = tanh(Wx_1 + Us_1)$$

$W,\ U$: weight matrices

“unfolding” the RNN across time:

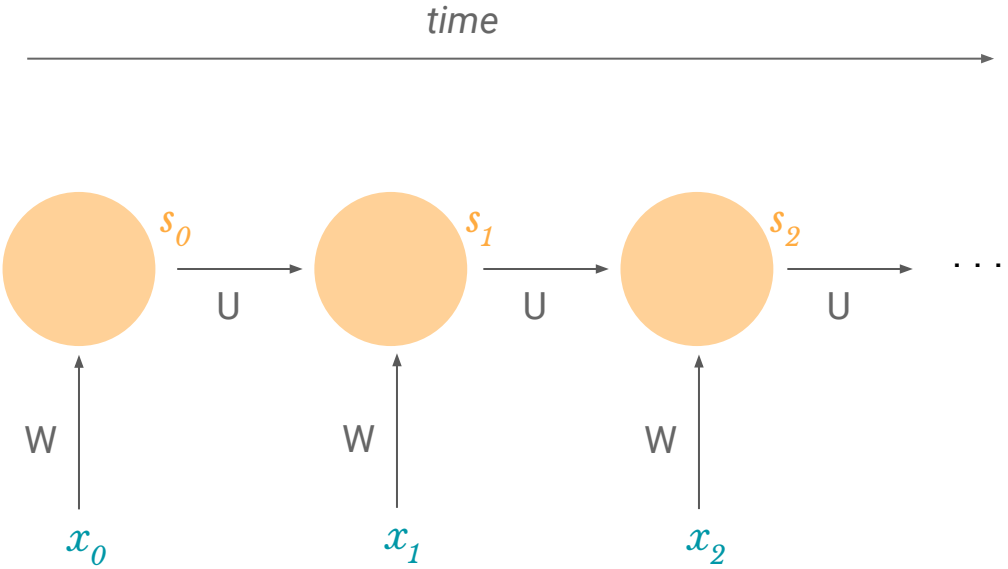


“unfolding” the RNN across time:



notice that we use the same parameters, W and U

“unfolding” the RNN across time:



s_n can contain information from all past timesteps

how do we **train** an RNN?

how do we **train** an RNN?

backpropagation!

(through time)

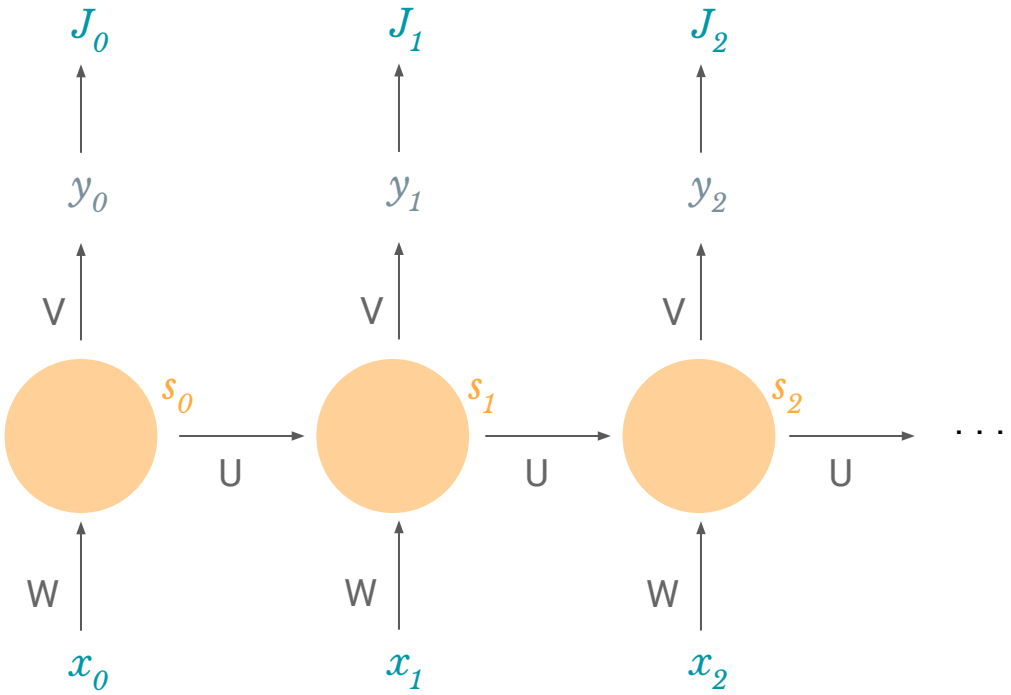
remember: **backpropagation**

1. take the derivative (gradient) of the loss with respect to each parameter

2. shift parameters in the opposite direction in order to minimize loss

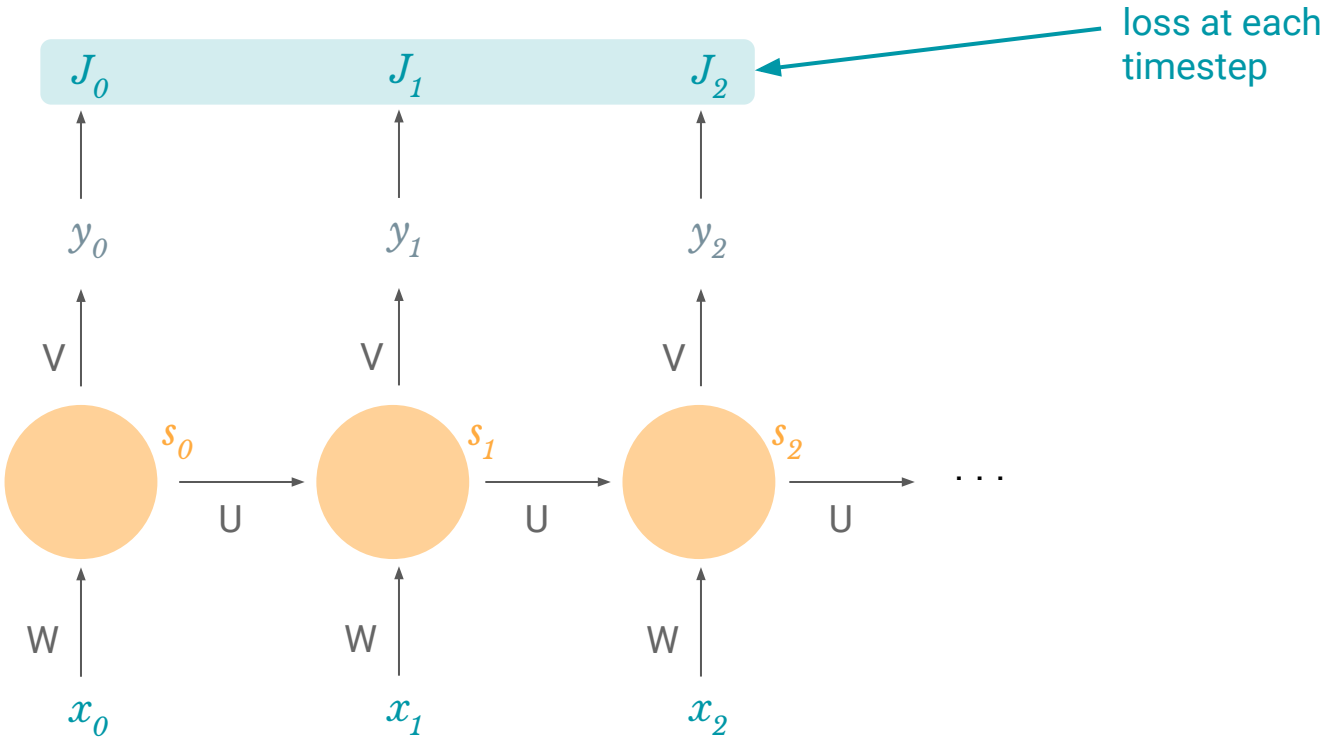
we have a **loss at each timestep**:

(since we're making a prediction at each timestep)



we have a **loss at each timestep**:

(since we're making a prediction at each timestep)



we **sum the losses** across time:

loss at time $t = J_t(\Theta)$

Θ = our parameters, like weights

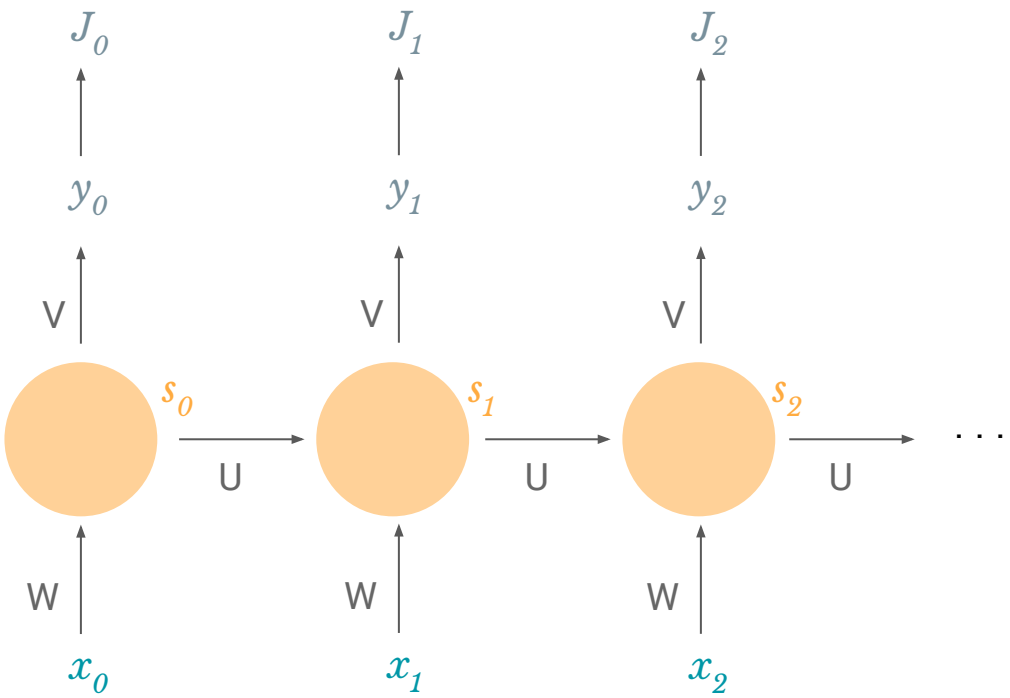
total loss = $J(\Theta) = \Sigma_t J_t(\Theta)$

what are our **gradients**?

we sum gradients across time for each parameter *P*:

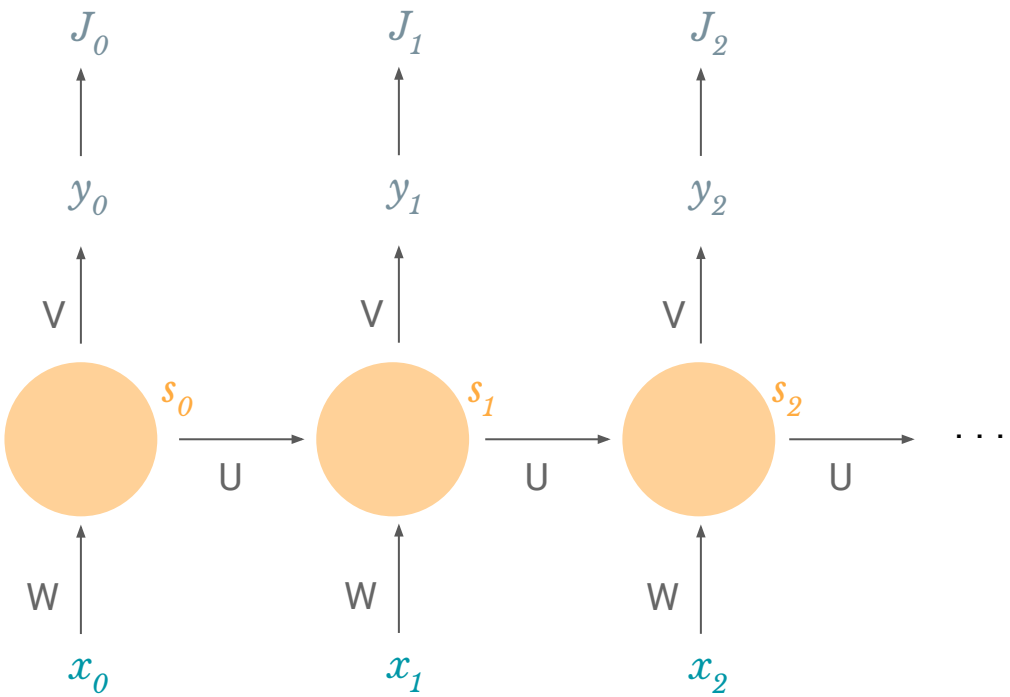
$$\frac{\partial J}{\partial P} = \sum_t \frac{\partial J_t}{\partial P}$$

let's try it out for *W* with the **chain rule**:



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

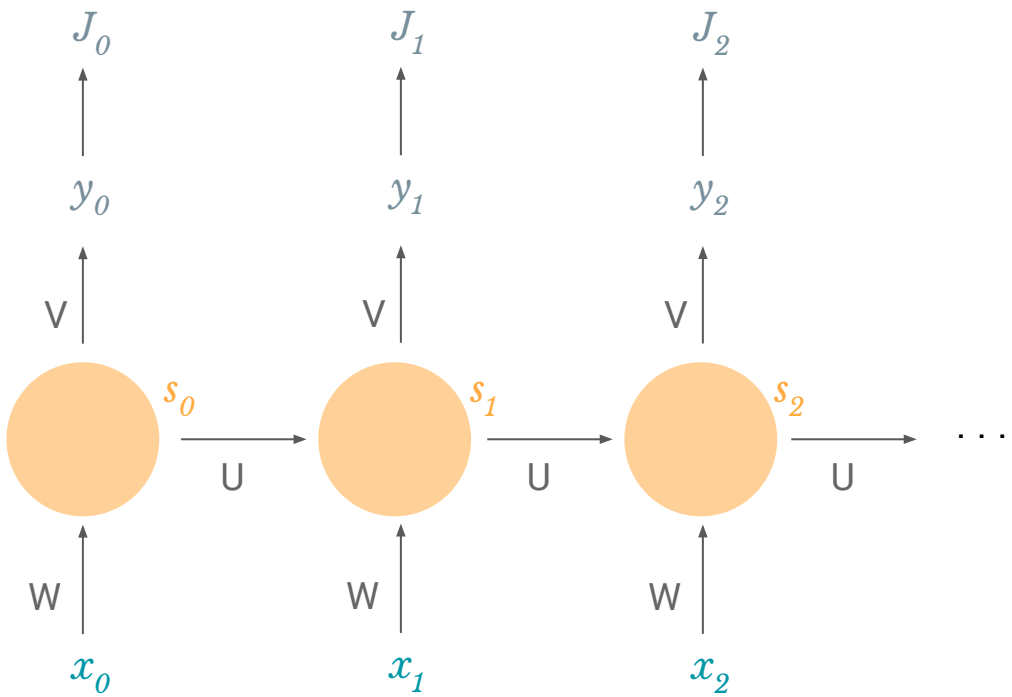
let's try it out for *W* with the **chain rule**:



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep *t*:

let's try it out for *W* with the **chain rule**:

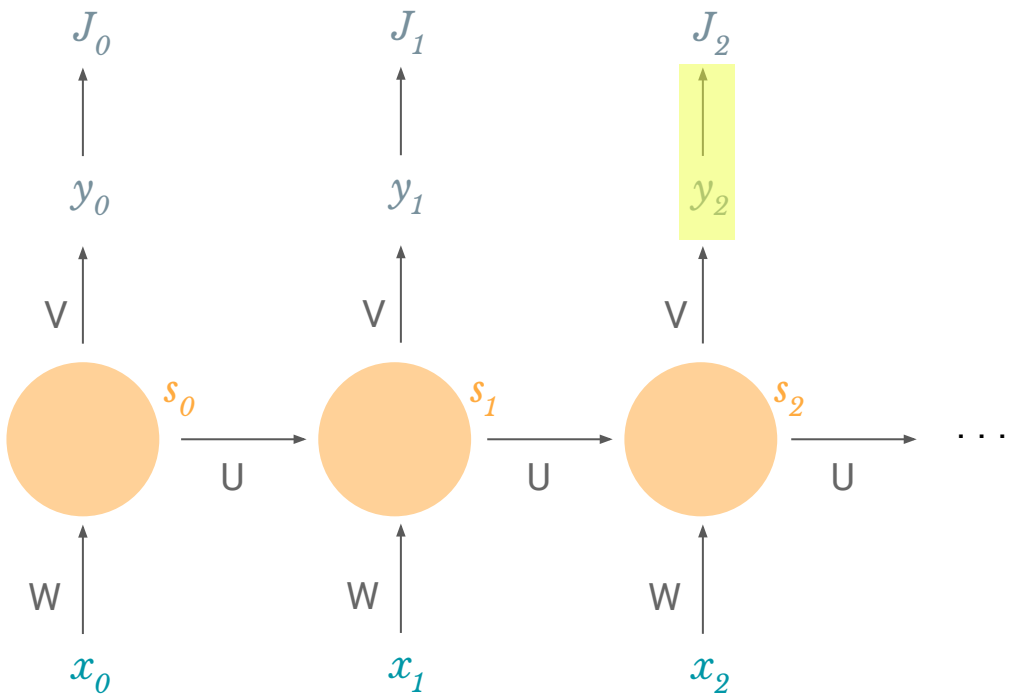


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep *t*:

$$\frac{\partial J_2}{\partial W}$$

let's try it out for *W* with the **chain rule**:

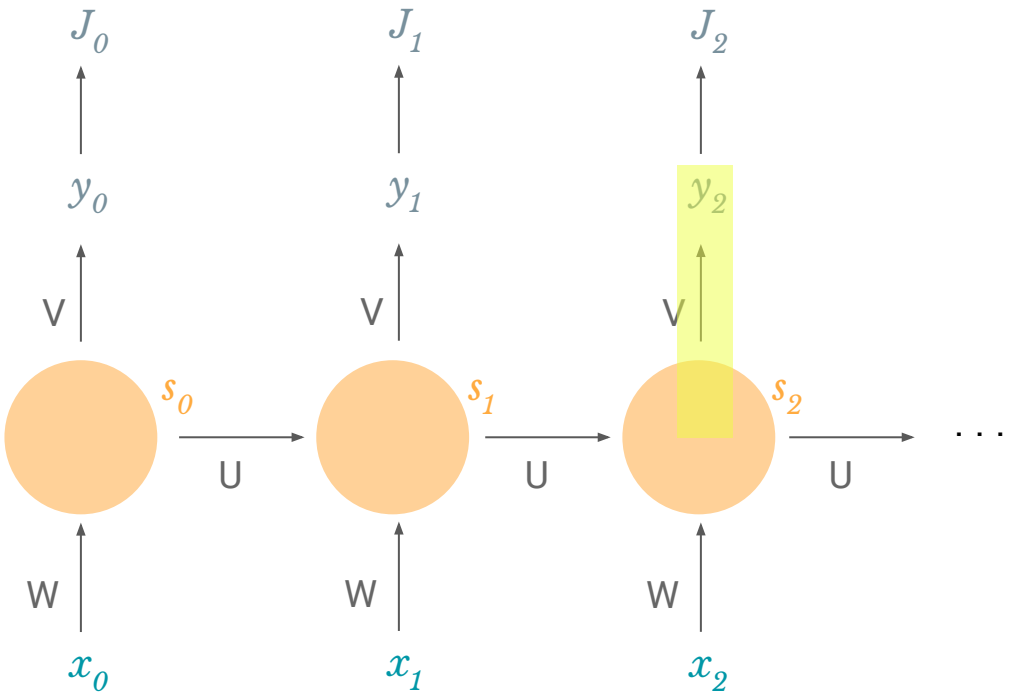


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep *t*:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}$$

let's try it out for *W* with the **chain rule**:

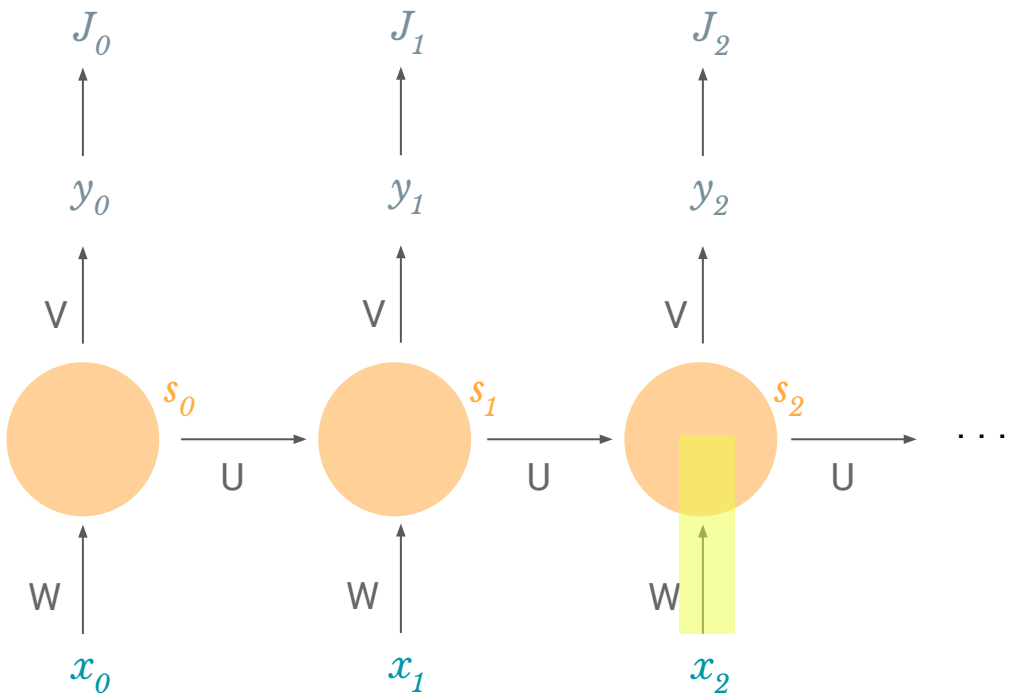


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep *t*:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}$$

let's try it out for *W* with the **chain rule**:

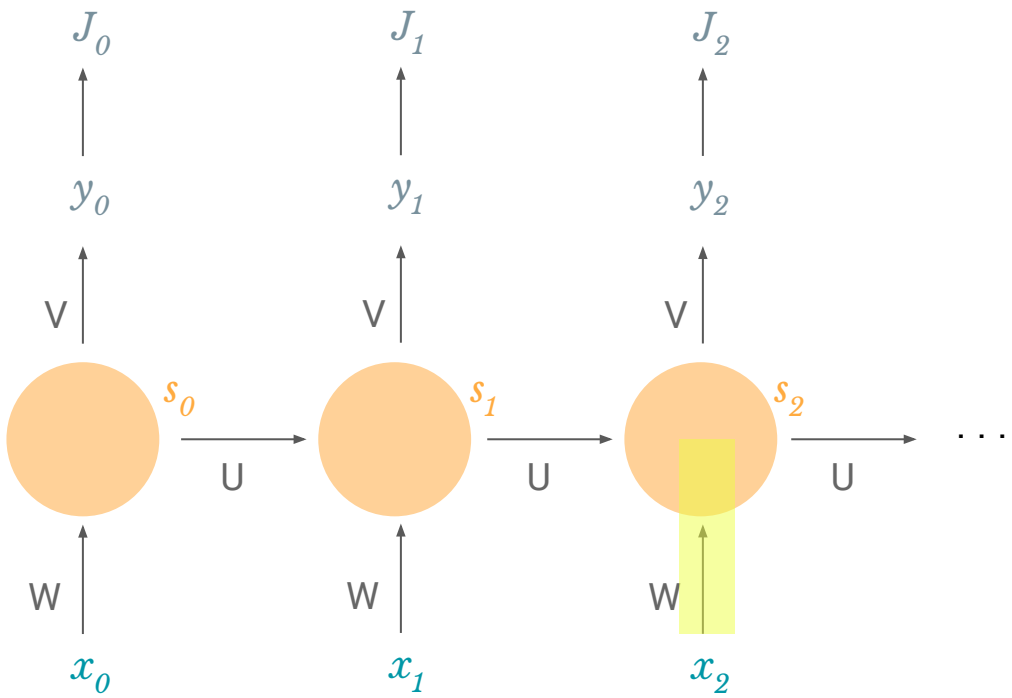


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep *t*:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$

let's try it out for *W* with the **chain rule**:



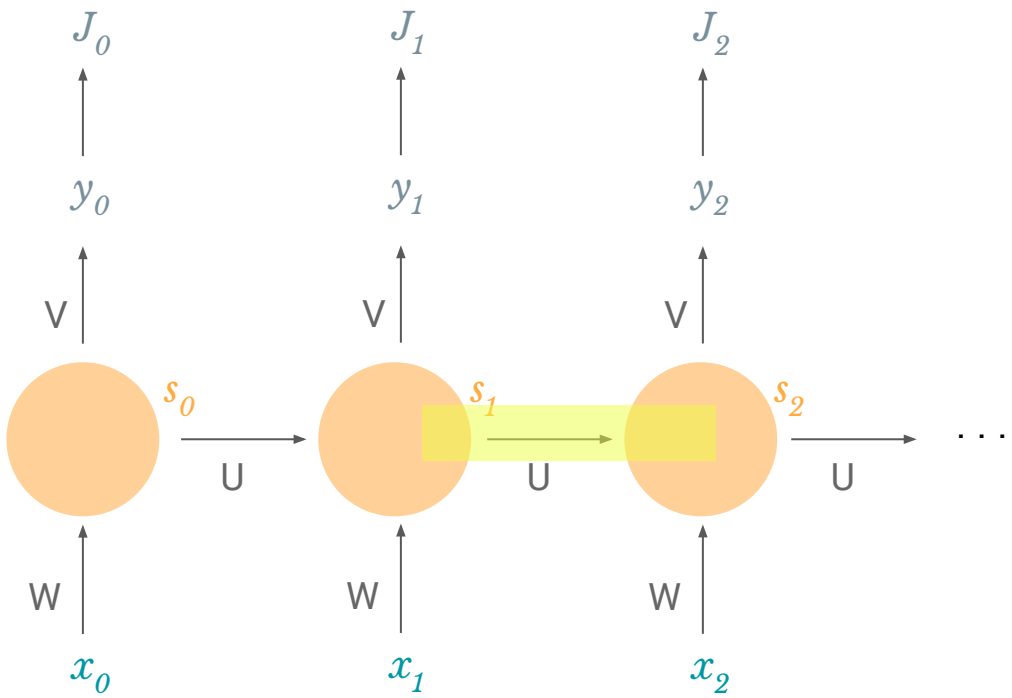
$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep *t*:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$

but wait…

let's try it out for *W* with the **chain rule**:



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

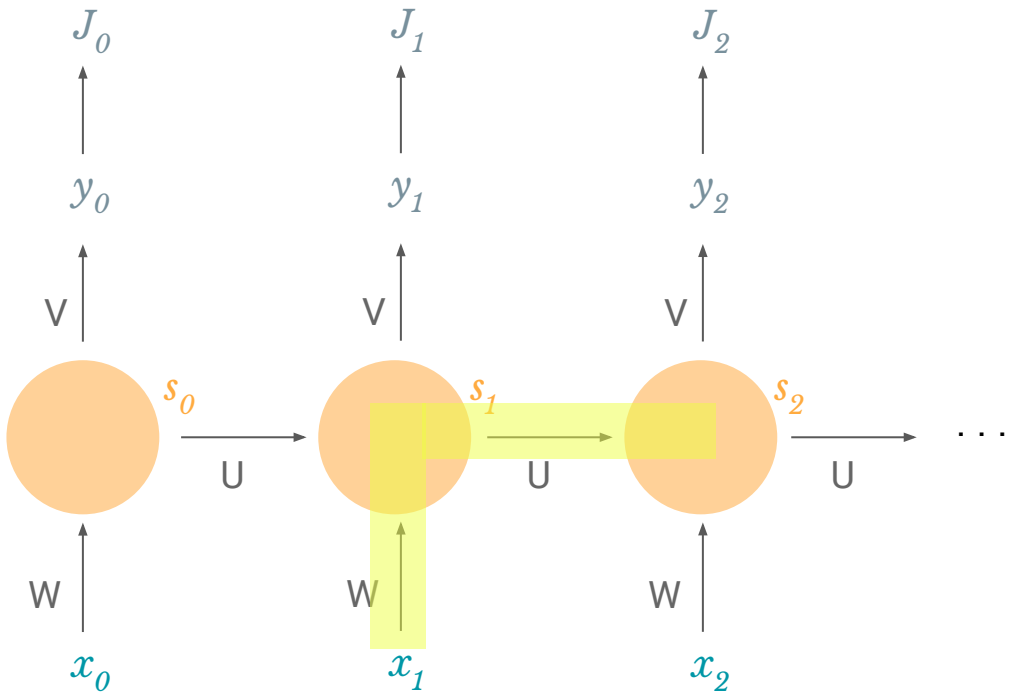
so let's take a single timestep *t*:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$

but wait...

$$s_2 = tanh(U s_1 + W x_2)$$

let's try it out for *W* with the **chain rule**:



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep *t*:

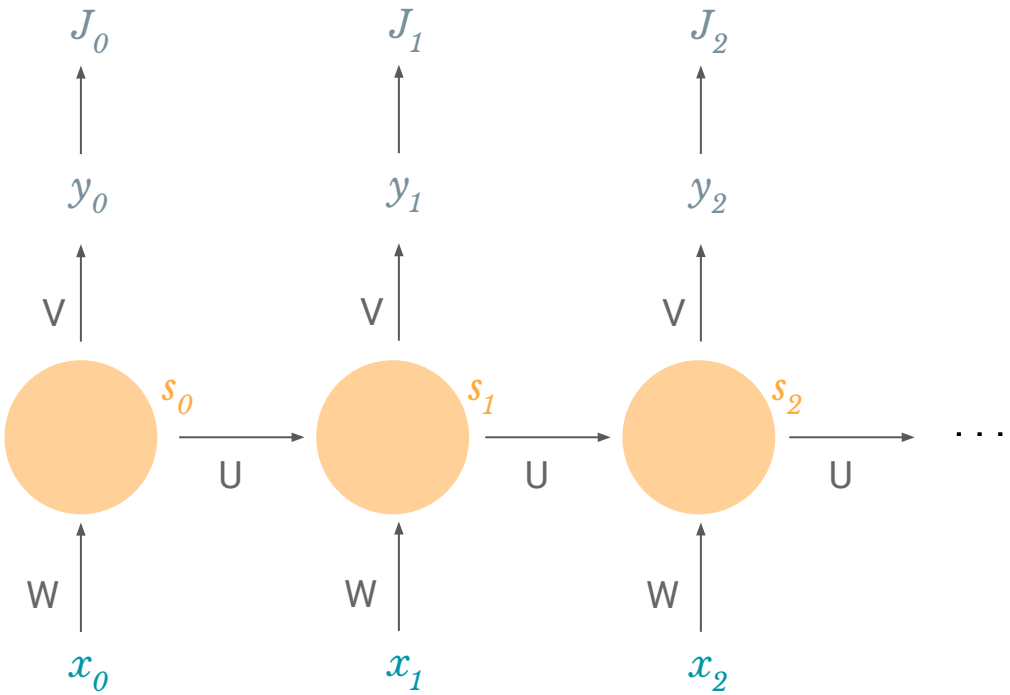
$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$

but wait...

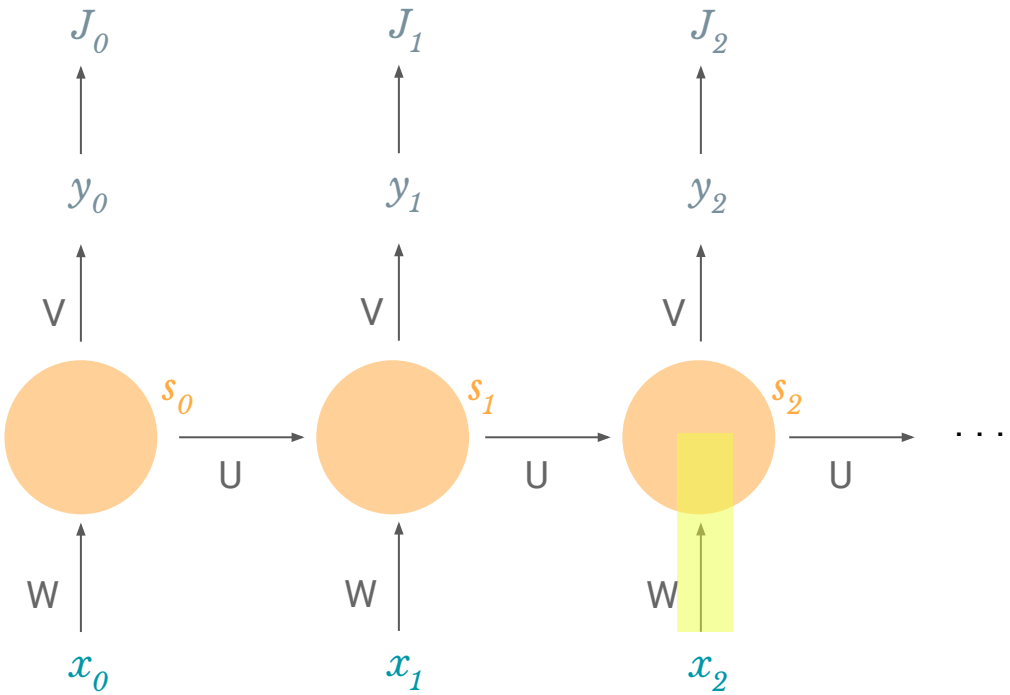
$$s_2 = tanh(U s_1 + W x_2)$$

s_1 also depends on *W* so we can't just treat $\frac{\partial s_2}{\partial W}$ as a constant!

how does s_2 depend on W?

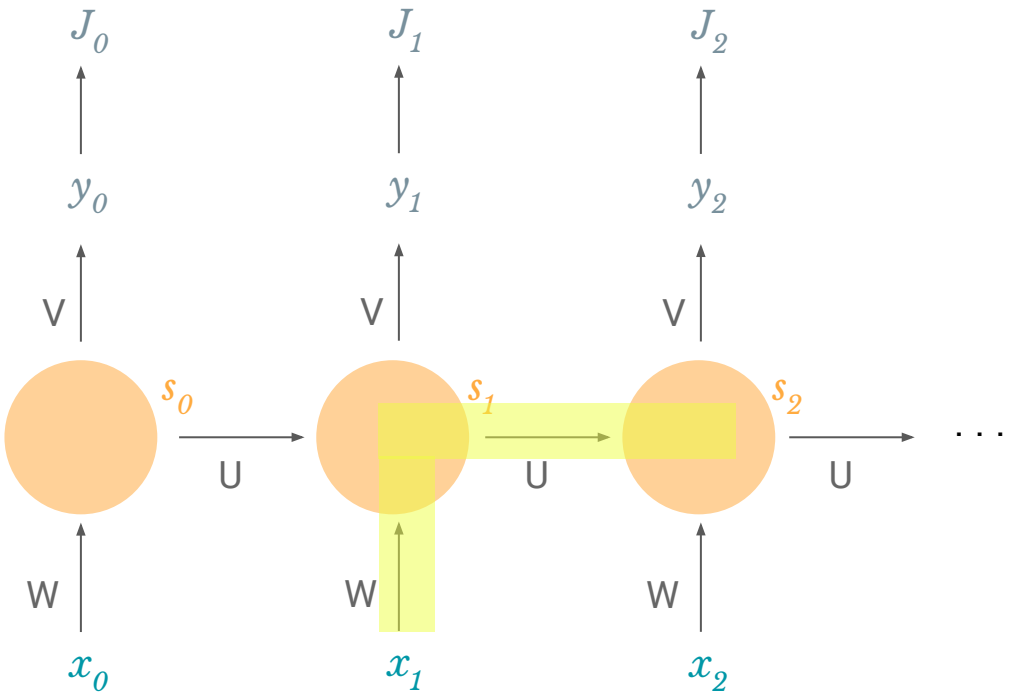


how does s_2 depend on W?



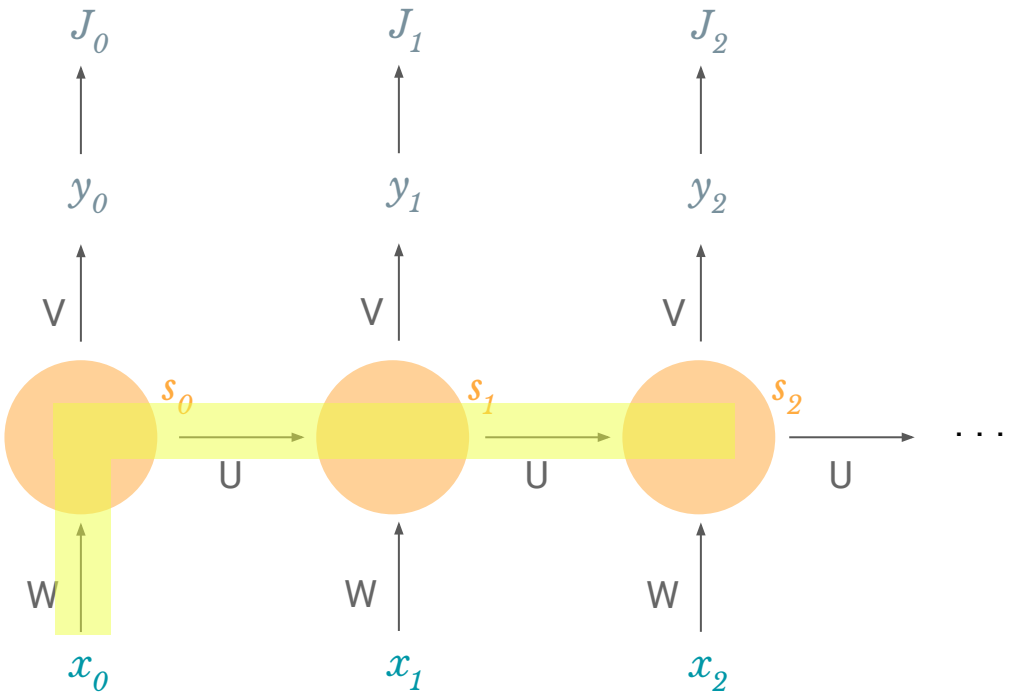
$$\frac{\partial s_2}{\partial W}$$

how does s_2 depend on W?



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W}$$

how does s_2 depend on W?



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial s_2}{\partial s_0}\frac{\partial s_0}{\partial W}$$

backpropagation through time:

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \underbrace{\frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}}$$

Contributions of *W* in previous timesteps to the error at timestep *t*

backpropagation through time:

$$\frac{\partial J_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial J_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} \underbrace{\frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}}$$

Contributions of *W* in previous timesteps to the error at timestep *t*

why are RNNs **hard to train**?

problem: vanishing gradient

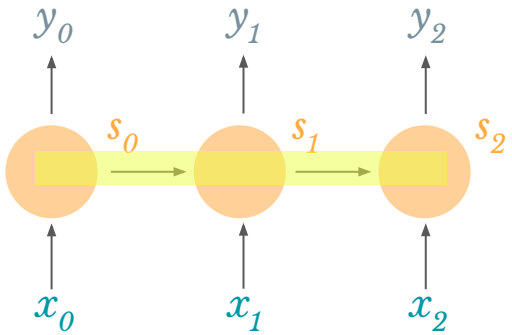
$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}$$

problem: vanishing gradient

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}$$

problem: vanishing gradient

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}$$

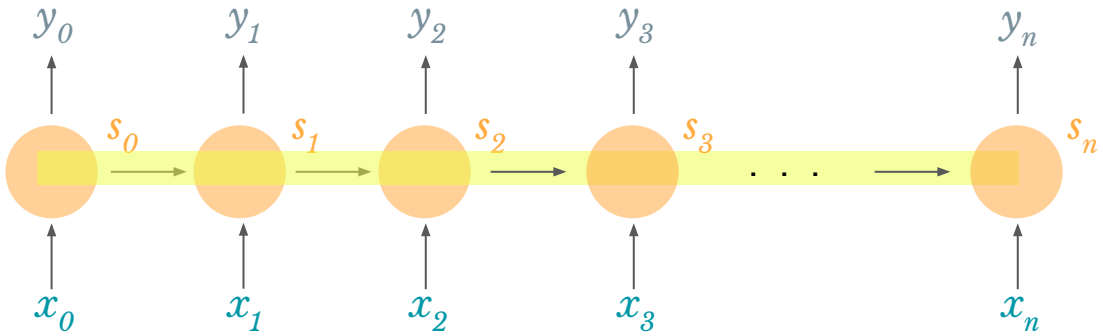


at $k = 0$: $\frac{\partial s_2}{\partial s_0} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$

problem: vanishing gradient

$$\frac{\partial J_n}{\partial W} = \sum_{k=0}^{n} \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \frac{\partial s_n}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

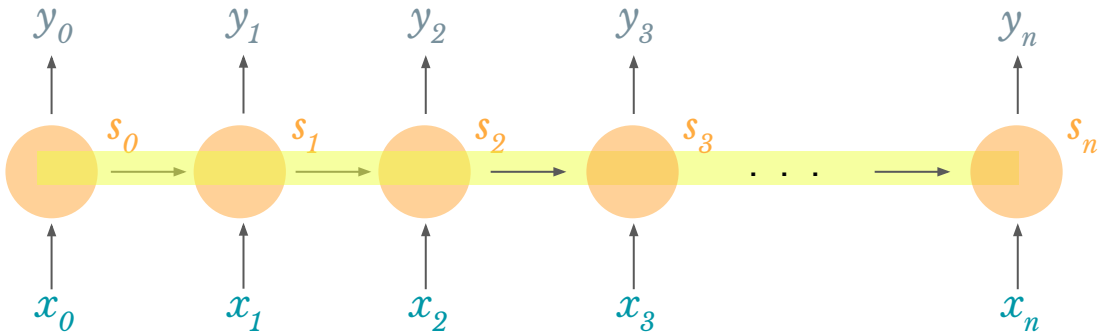


problem: vanishing gradient

$$\frac{\partial J_n}{\partial W} = \sum_{k=0}^{n} \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \frac{\partial s_n}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

as the gap between timesteps gets bigger, this product gets longer and longer!



problem: vanishing gradient

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

problem: vanishing gradient

what are each of these terms? →

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

problem: vanishing gradient

what are each of these terms?

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

$$\frac{\partial s_n}{\partial s_{n-1}} = W^T diag[f'(W_{s_{j-1}} + Ux_j)]$$

W = sampled from standard normal distribution = mostly < 1

f = tanh or sigmoid so **f'** < 1

problem: vanishing gradient

what are each of these terms?

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

$$\frac{\partial s_n}{\partial s_{n-1}} = W^T diag[f'(W_{s_{j-1}} + Ux_j)]$$

W = sampled from standard normal distribution = mostly < 1

f = tanh or sigmoid so **f'** < 1

we're multiplying a lot of **small numbers** together.

we're multiplying a lot of **small numbers** together.

so what?

errors due to further back timesteps have increasingly **smaller gradients.**

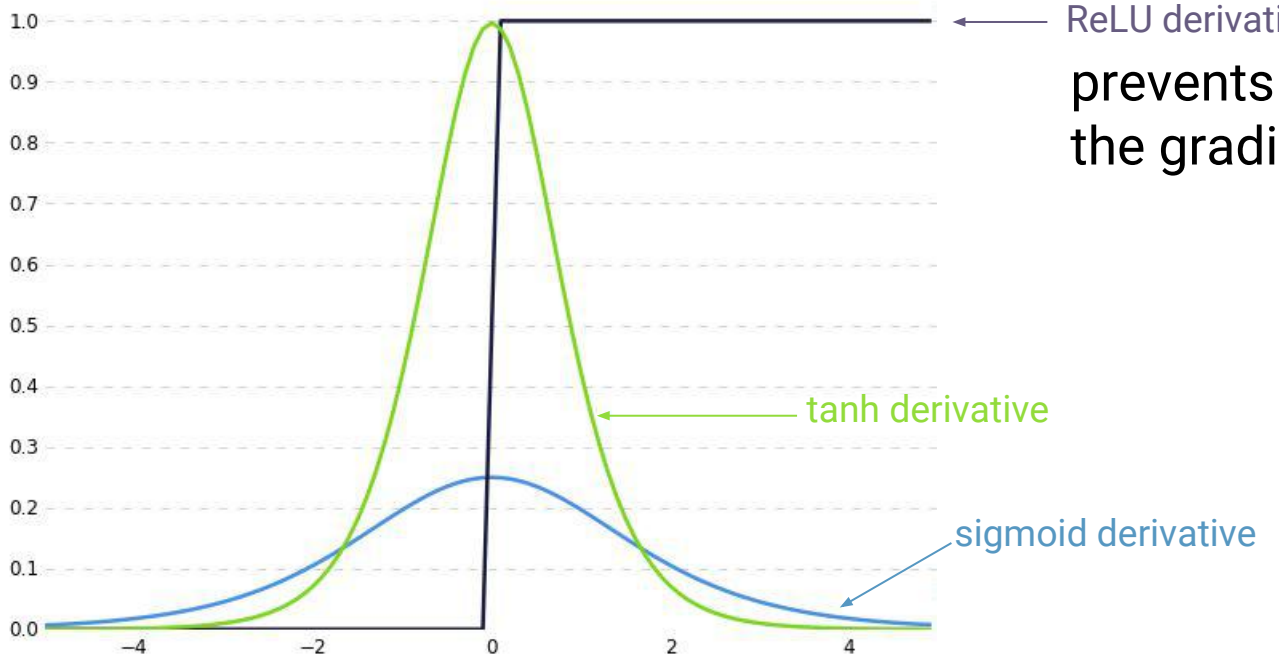
so what?

parameters become biased to **capture shorter-term** dependencies.

“In France, I had a great time and I learnt some of the _____ language.”

our parameters are not trained to capture long-term dependencies, so the word we predict will mostly depend on the previous few words, not much earlier ones

solution #1: **activation functions**



prevents f’ from shrinking the gradients

solution #2: **initialization**

weights initialized to identity matrix

biases initialized to zeros

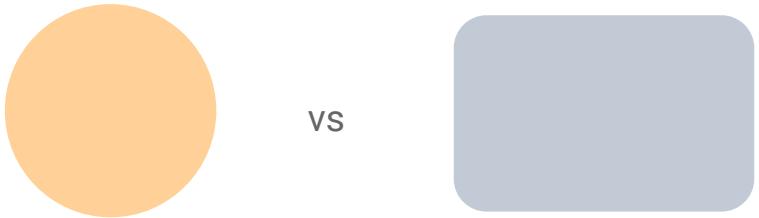
$$\longrightarrow \quad I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

prevents W from shrinking the gradients

a different type of solution:
more complex cells

solution #3: **gated cells**

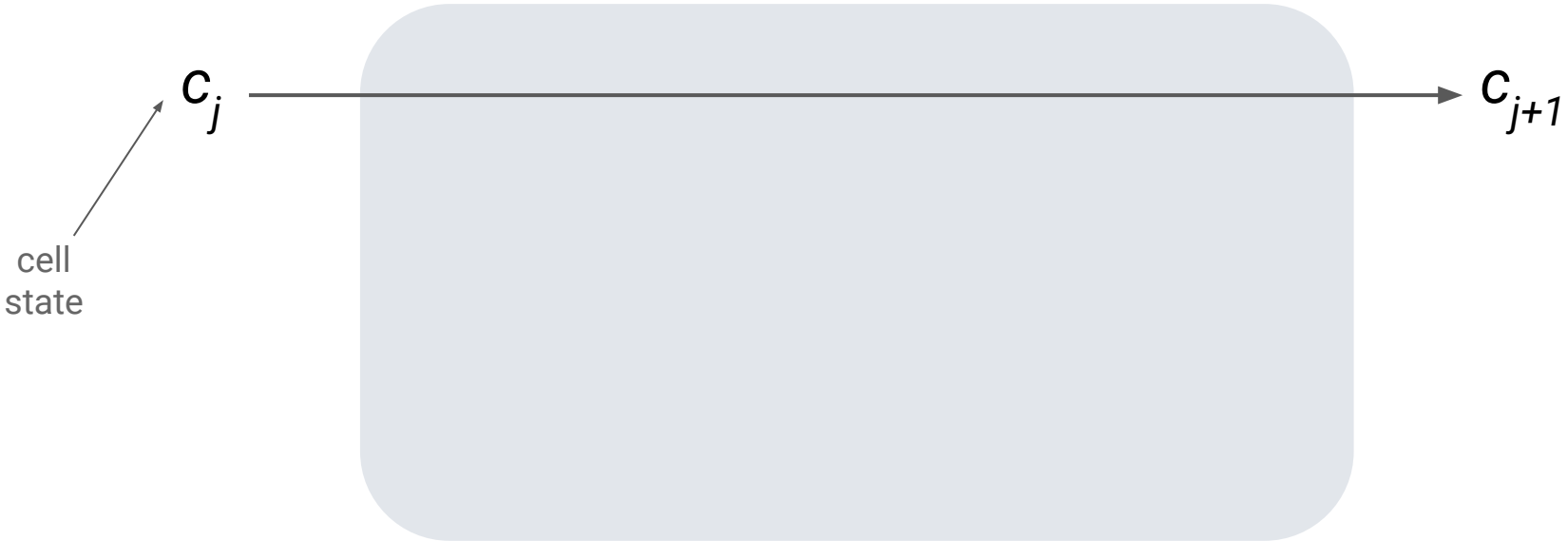
rather each node being just a simple RNN cell, make each node a more **complex unit with gates** controlling what information is passed through.



RNN vs LSTM, GRU, etc

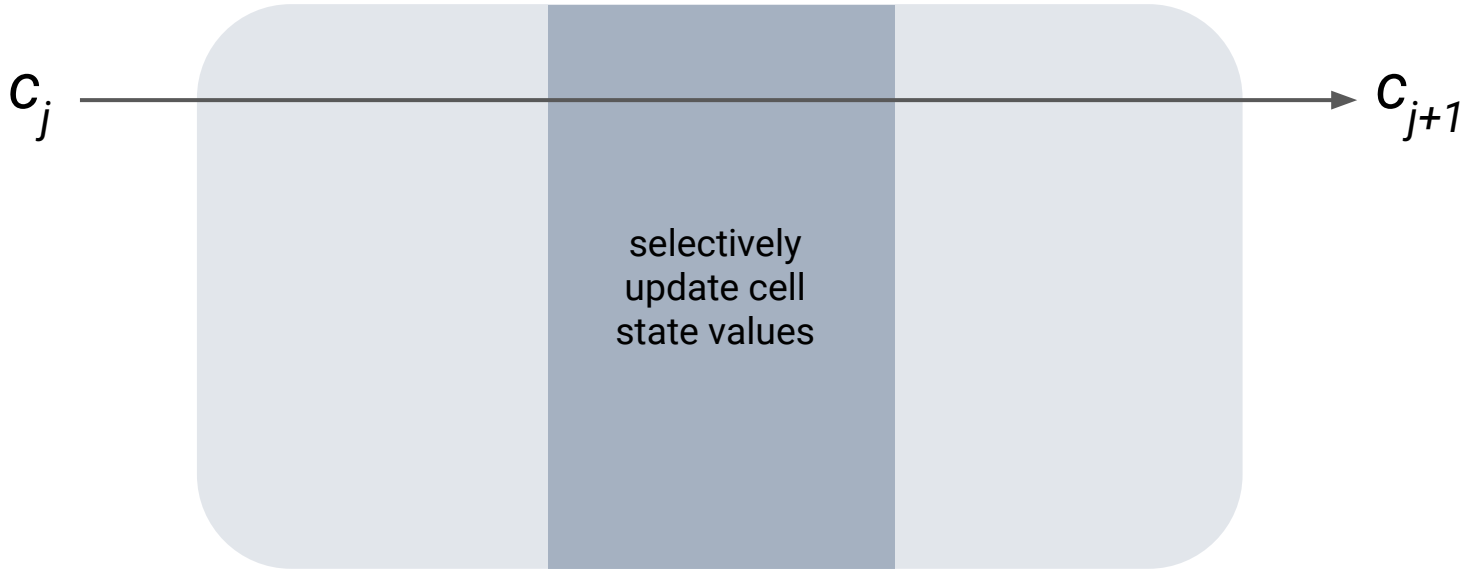
Long short term memory cells are able to keep track of information throughout many timesteps.

solution #3: more on **LSTMs**

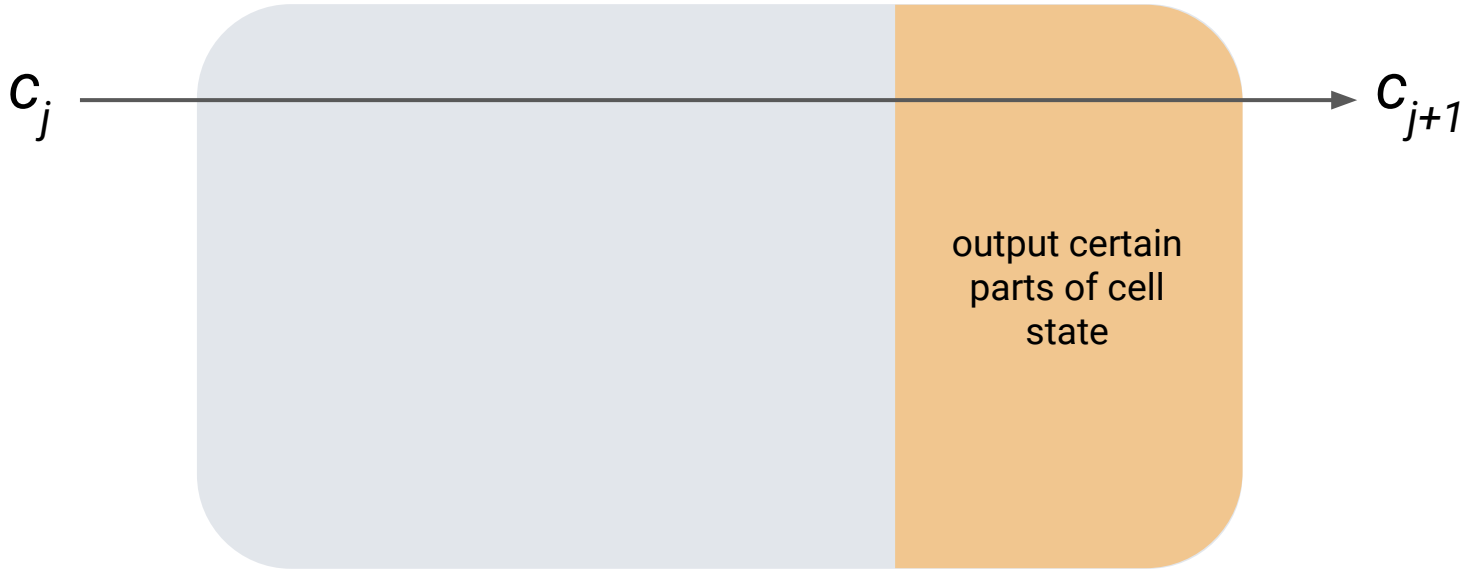


solution #3: more on **LSTMs**

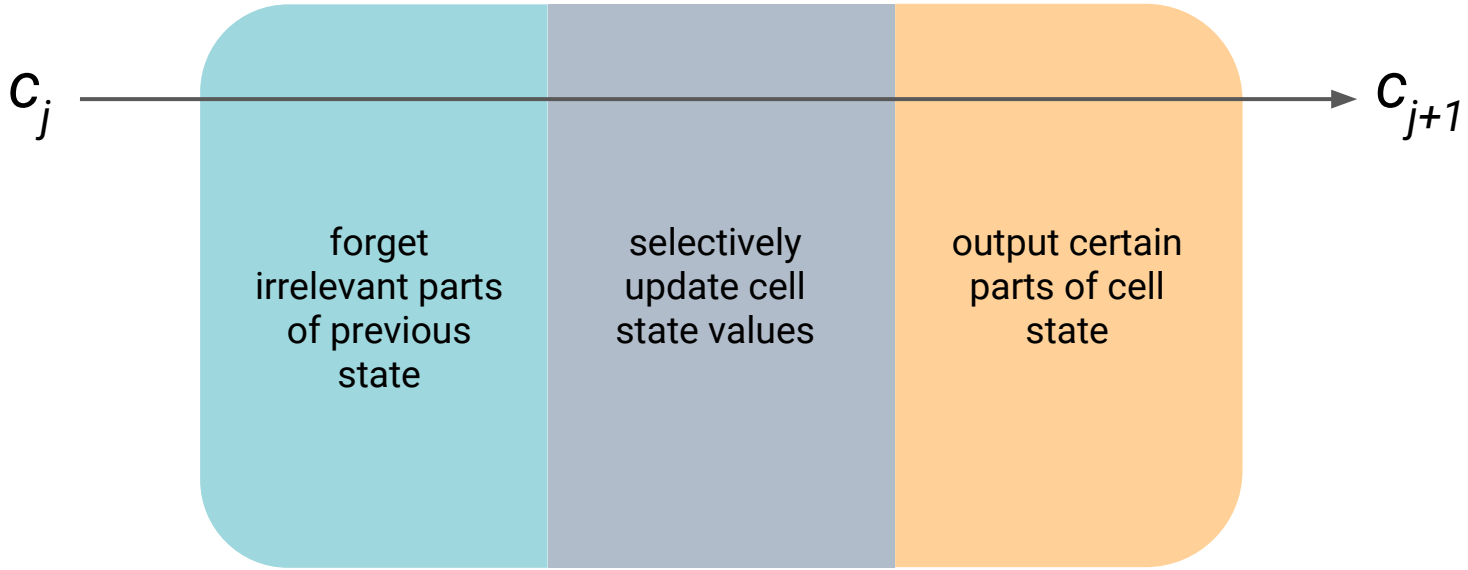
solution #3: more on **LSTMs**



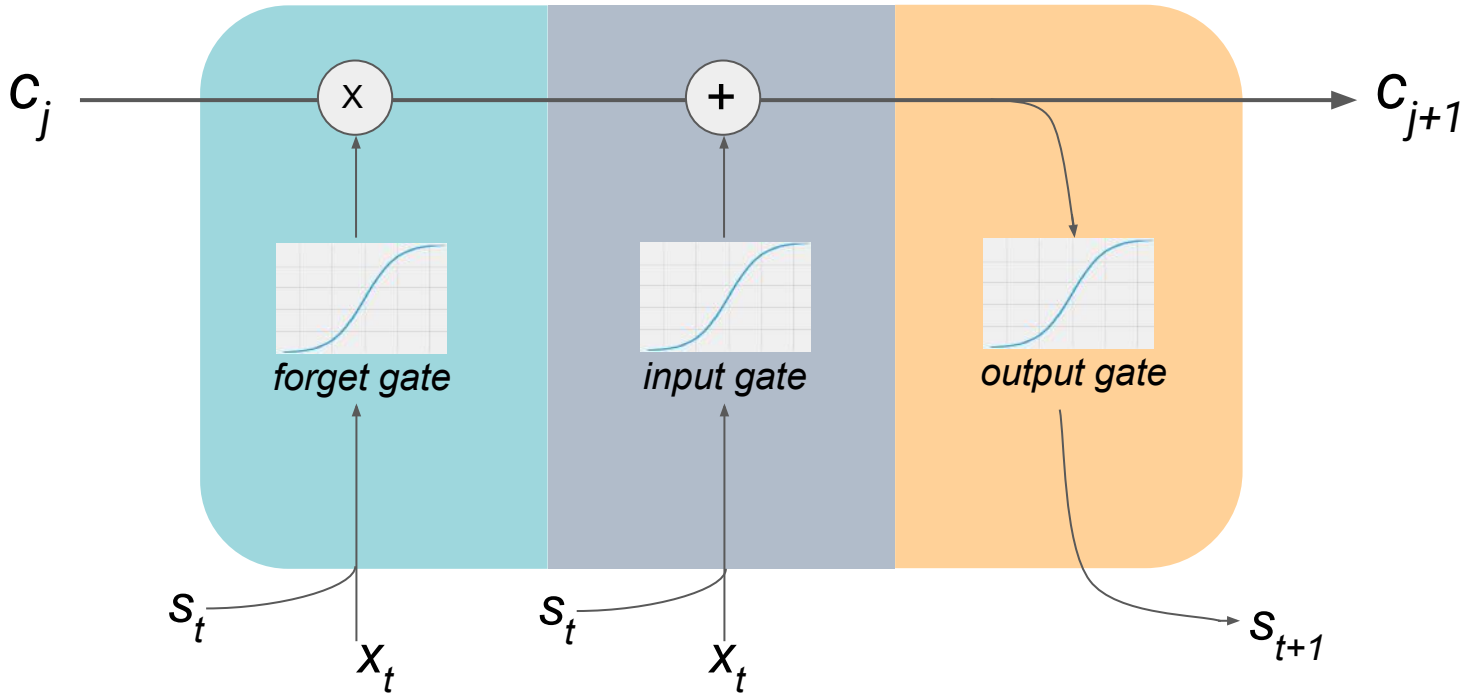
solution #3: more on **LSTMs**



solution #3: more on **LSTMs**



solution #3: more on **LSTMs**



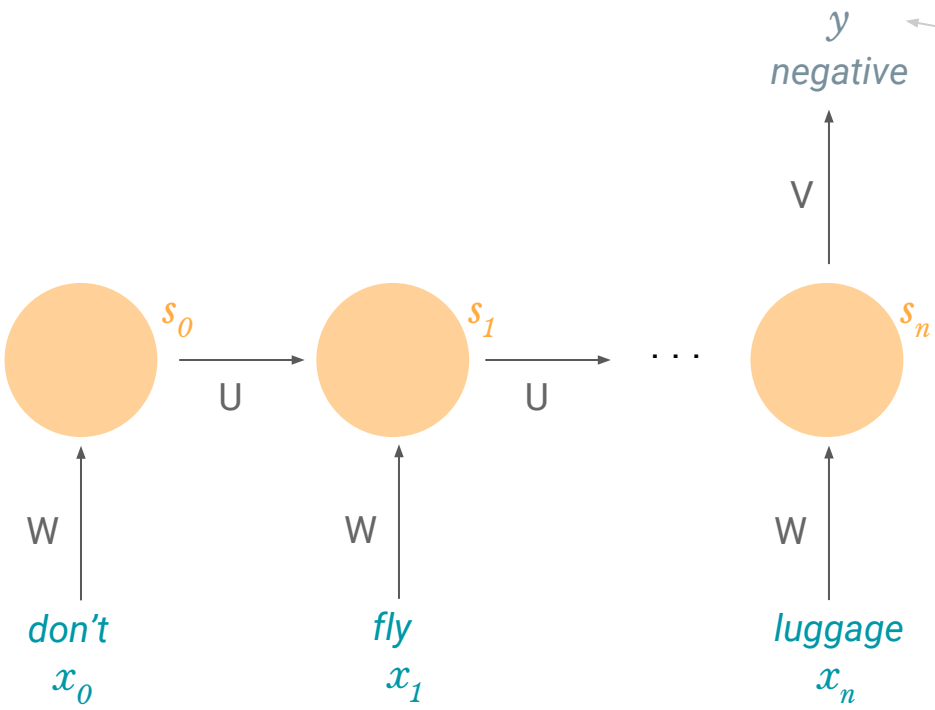
why do LSTMs help?

1. forget gate allows information to pass through unchanged
2. cell state is separate from what's outputted
3. s_j depends on s_{j-1} through addition!

   → derivatives don't expand into a long product!

possible task: classification (i.e. sentiment)

possible task: classification (i.e. sentiment)

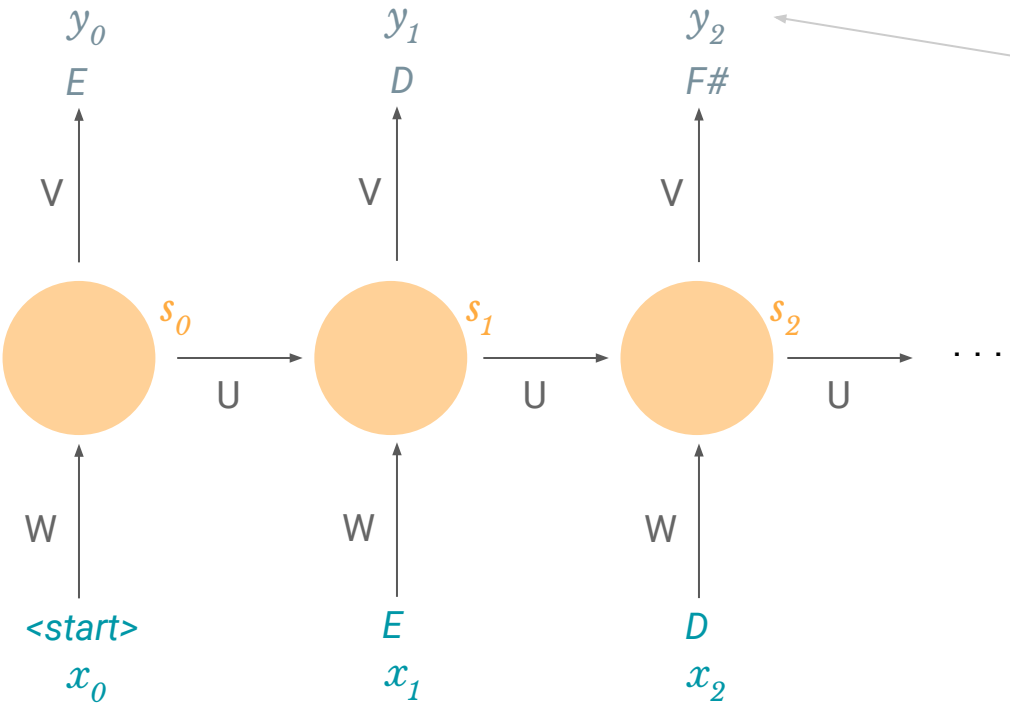


y is a probability distribution over possible classes (like positive, negative, neutral), aka a *softmax*

possible task: music generation

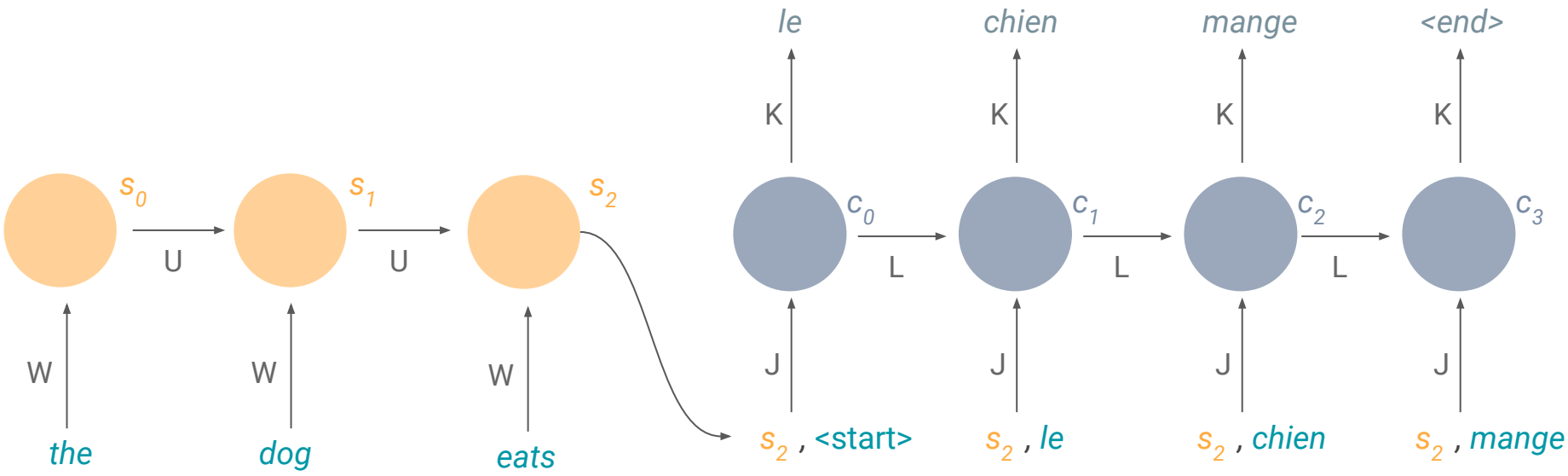
Music by: Francesco Marchesani, Computer Science Engineer, PoliMi

possible task: music generation

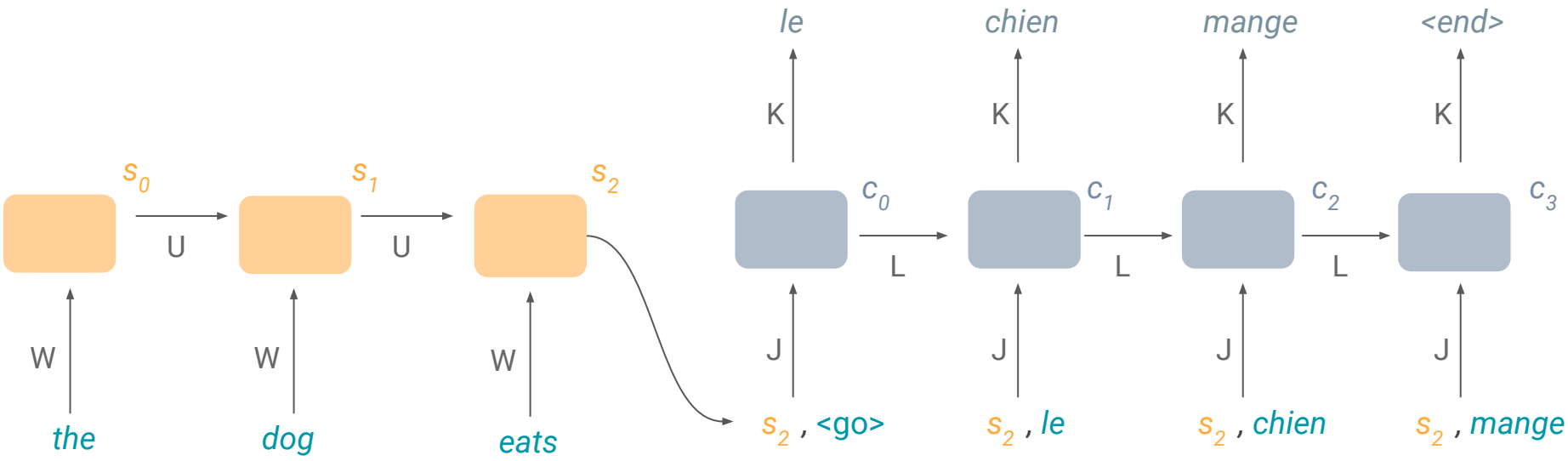


y_i is actually a probability distribution over possible next notes, aka a *softmax*

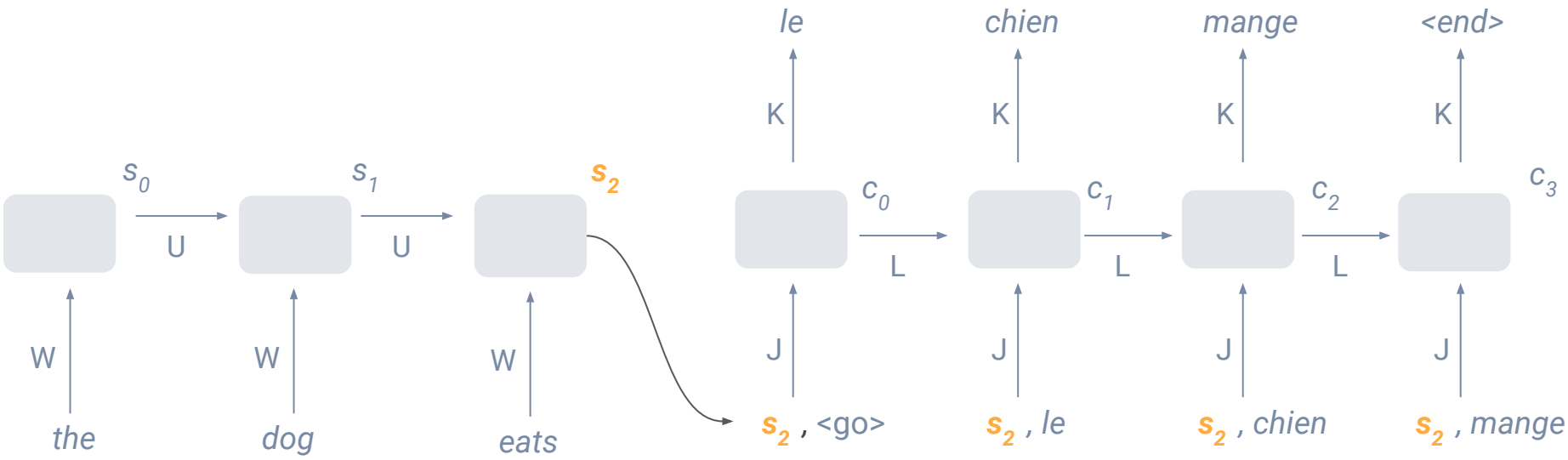
possible task: machine translation



possible task: machine translation

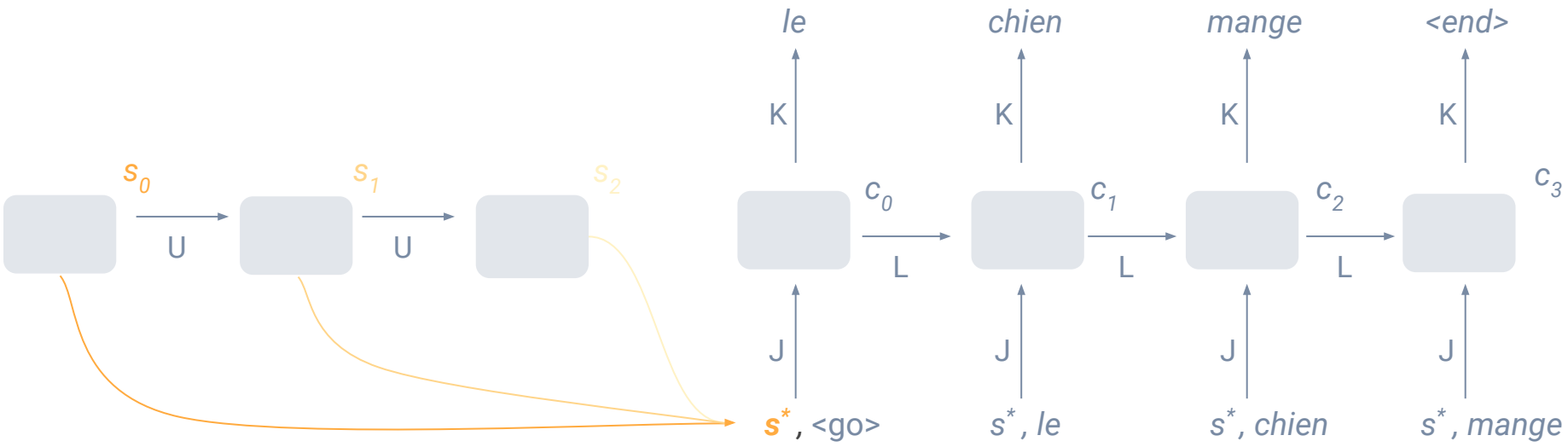


problem: a single encoding is limiting

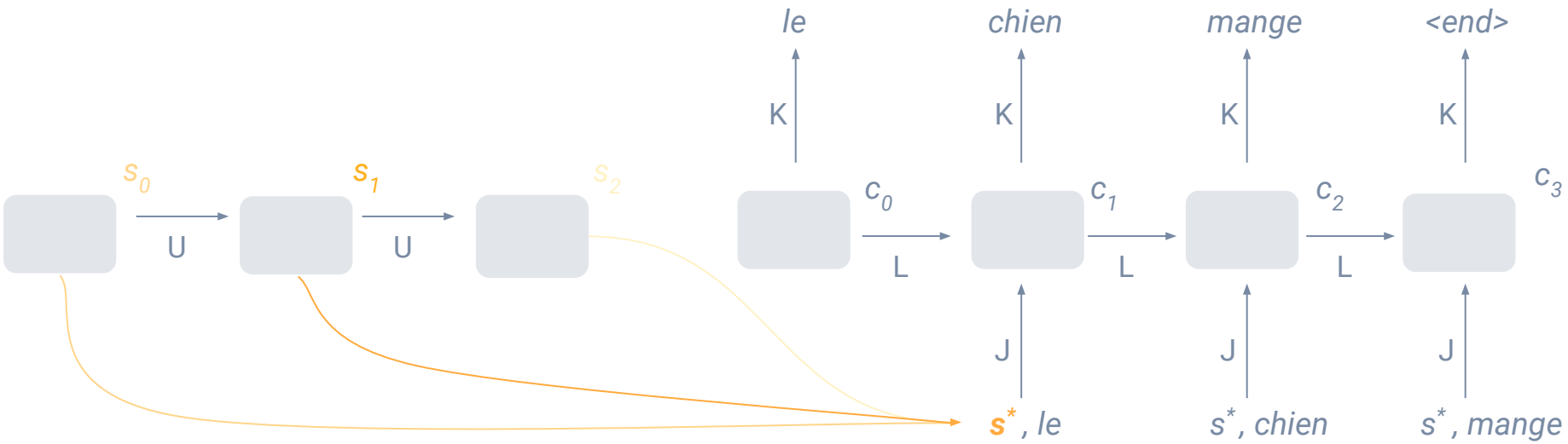


all the decoder knows about the input sentence is in one fixed length vector, s_2

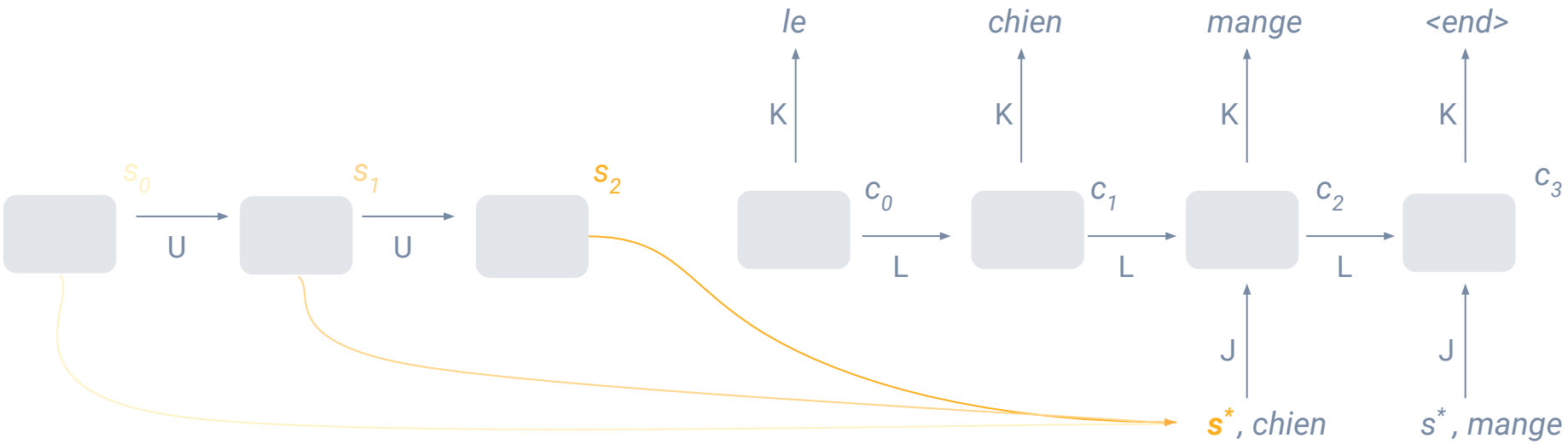
solution: attend over all encoder states



solution: attend over all encoder states



solution: attend over all encoder states



now we can model **sequences**!

- why recurrent neural networks?
- training them with backpropagation through time
- solving the vanishing gradient problem with activation functions, initialization, and gated cells (like LSTMs)
- building models for classification, music generation and machine translation
- using attention mechanisms

and there’s lots more to do!

- extending our models to timeseries + waveforms
- complex language models to generate long text or books
- language models to generate code
- controlling cars + robots
- predicting stock market trends
- summarizing books + articles
- handwriting generation
- multilingual translation models
- … many more!