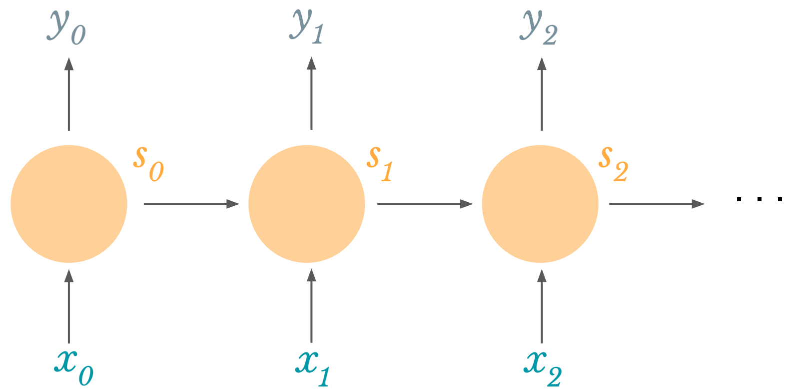
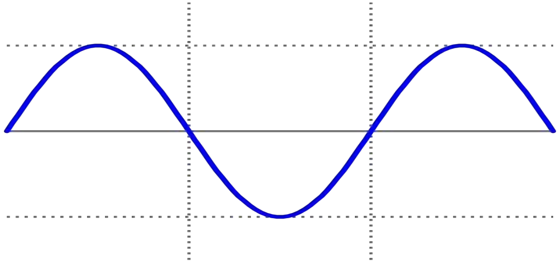
Sequence Modeling with Neural Networks

Harini Suresh



What is a sequence?

- “I took the dog for a walk this morning.”
- 
- 

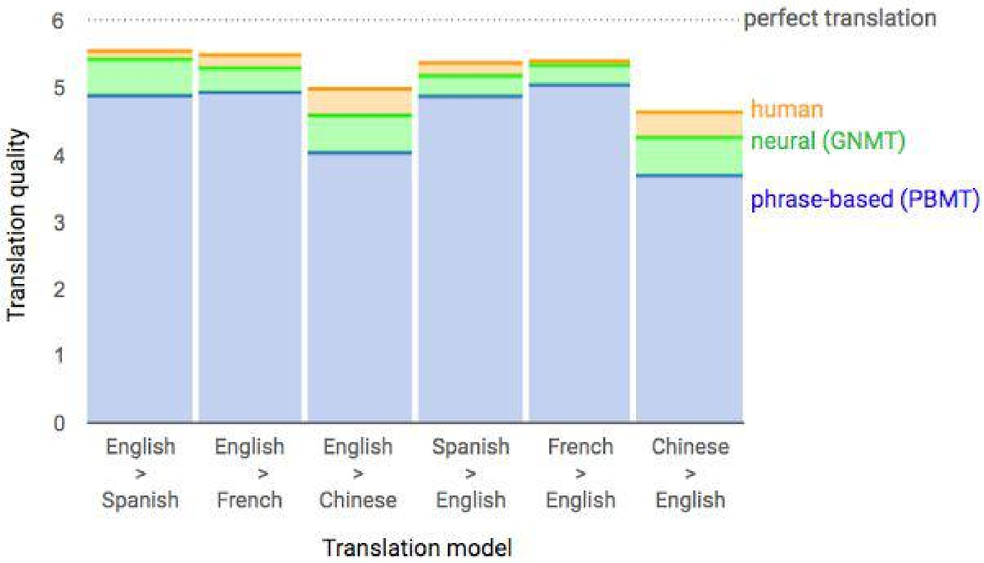
sentence

function

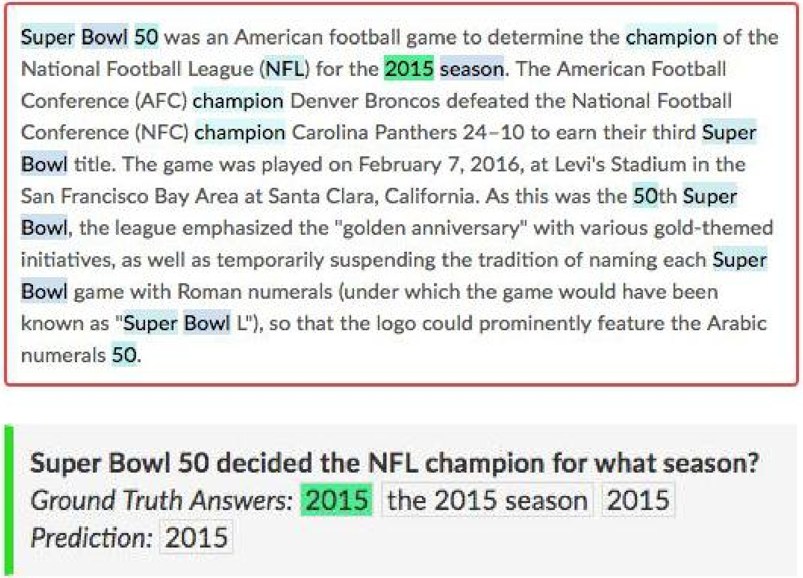
speech waveform

Successes of deep models

Machine translation



Question Answering



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

how do we model sequences?

idea: represent a sequence as a *bag of words*

"I dislike rain."

↓

[0 1 0 1 0 0 0 1]

↓

prediction

problem: bag of words does not preserve order

problem: bag of words does not preserve order

"The food was good, not bad at all."

vs

"The food was bad, not good at all."

idea: maintain an ordering within feature vector

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

On Monday it was snowing

prediction

One hot feature vector indicates what each word is

problem: hard to deal with different word orders

"On Monday, it was snowing."

vs

"It was snowing on Monday."

problem: hard to deal with different word orders

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

On Monday it was snowing

vs

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

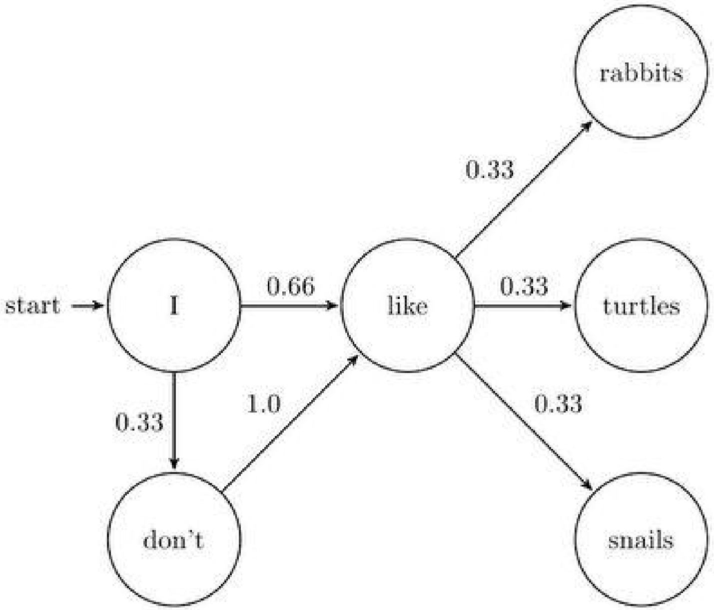
It was snowing on Monday

problem: hard to deal with different word orders

"On Monday it was snowing."
vs
"It was snowing on Monday."

We would have to relearn the rules of language at each point in the sentence.

idea: markov models



problem: we can't model long-term dependencies

markov assumption: each state depends only on the last state.

problem: we can't model long-term dependencies

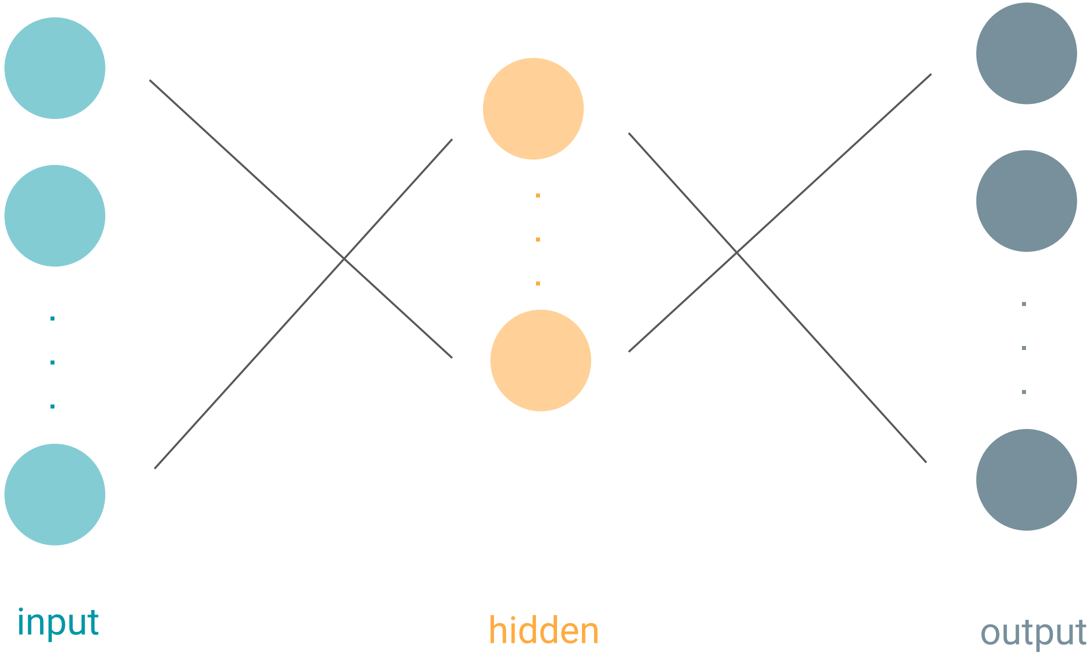
"In France, I had a great time and I learnt some of the _____ language."

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

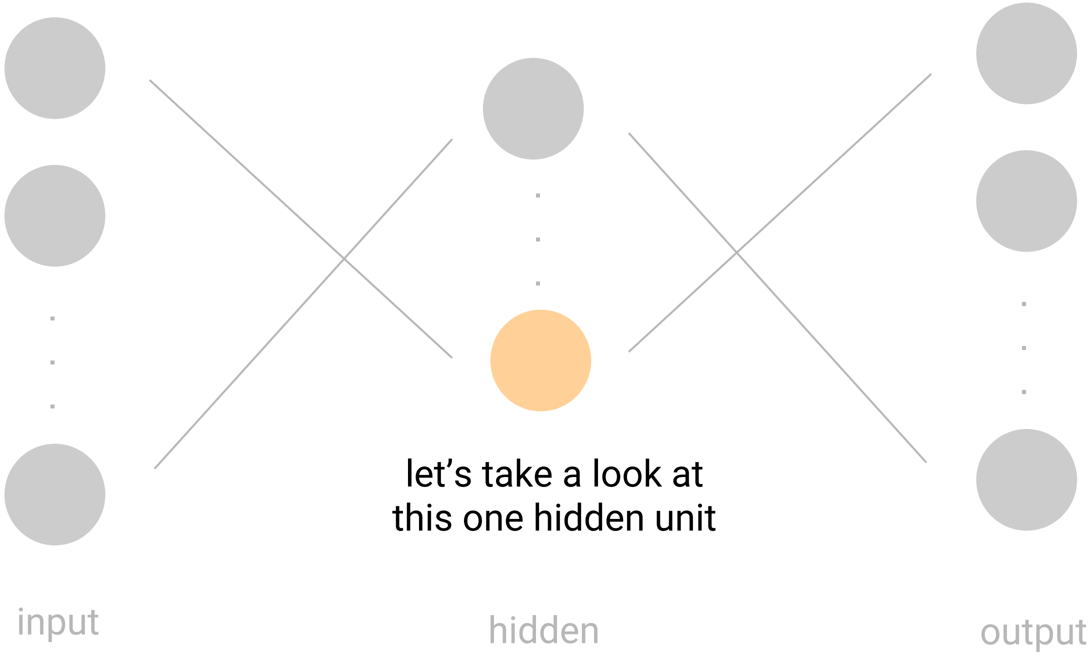
let's turn to **recurrent neural networks**! (RNNs)

1. to maintain word order
2. to share parameters across the sequence
3. to keep track of long-term dependencies

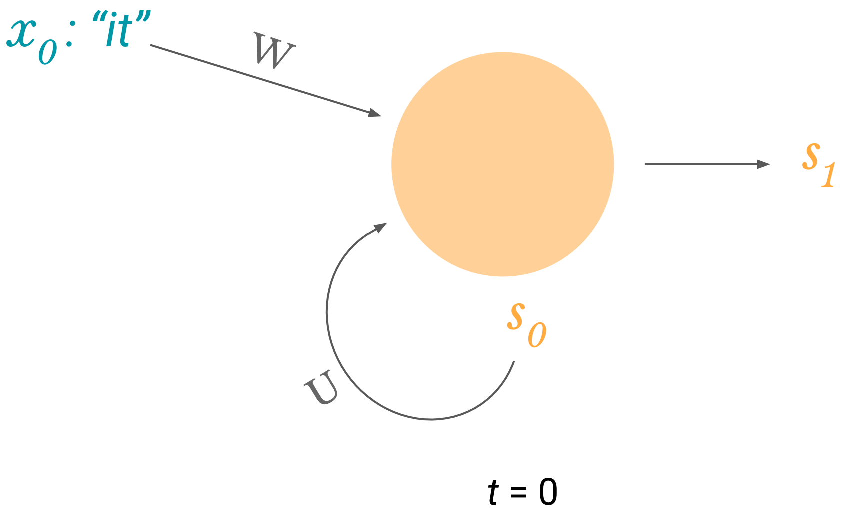
example network:



example network:



RNNS **remember** their previous state:

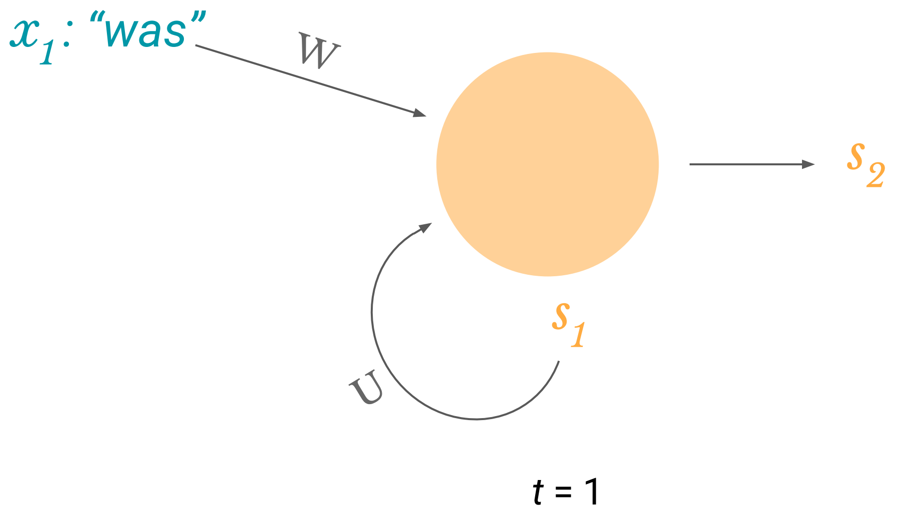


x_0 : vector representing first word
s_0 : cell state at $t = 0$ (some initialization)
s_1 : cell state at $t = 1$

$$s_1 = tanh(W x_0 + U s_0)$$

$W,\ U$: weight matrices

RNNS **remember** their previous state:

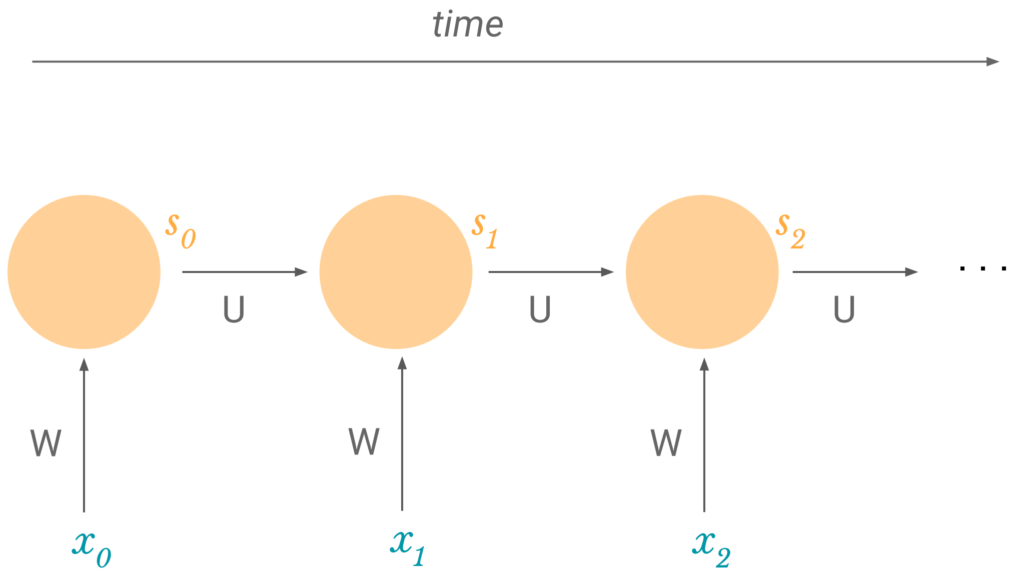


x_1 : vector representing second word
s_1 : cell state at $t = 1$
s_2 : cell state at $t = 2$

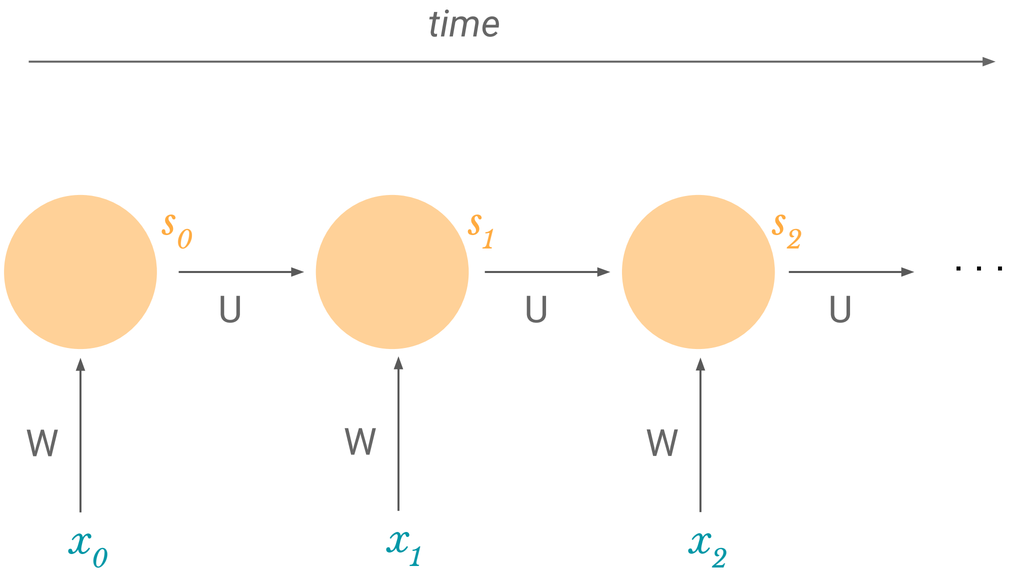
$$s_2 = tanh(Wx_1 + Us_1)$$

$W,\ U$: weight matrices

“unfolding” the RNN across time:

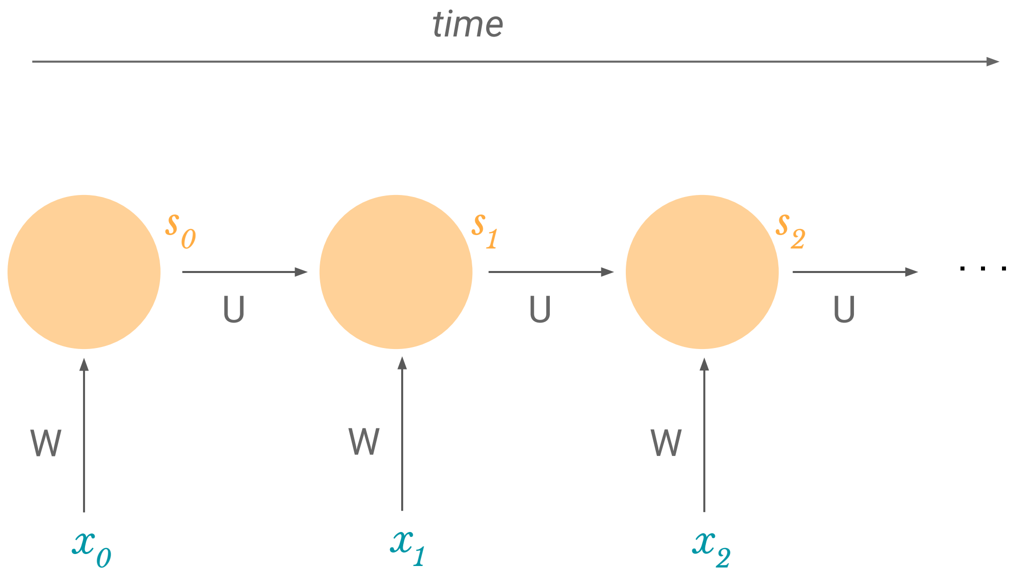


"unfolding" the RNN across time:



notice that W and U stay the same!

“unfolding” the RNN across time:



s_n can contain information from all past timesteps

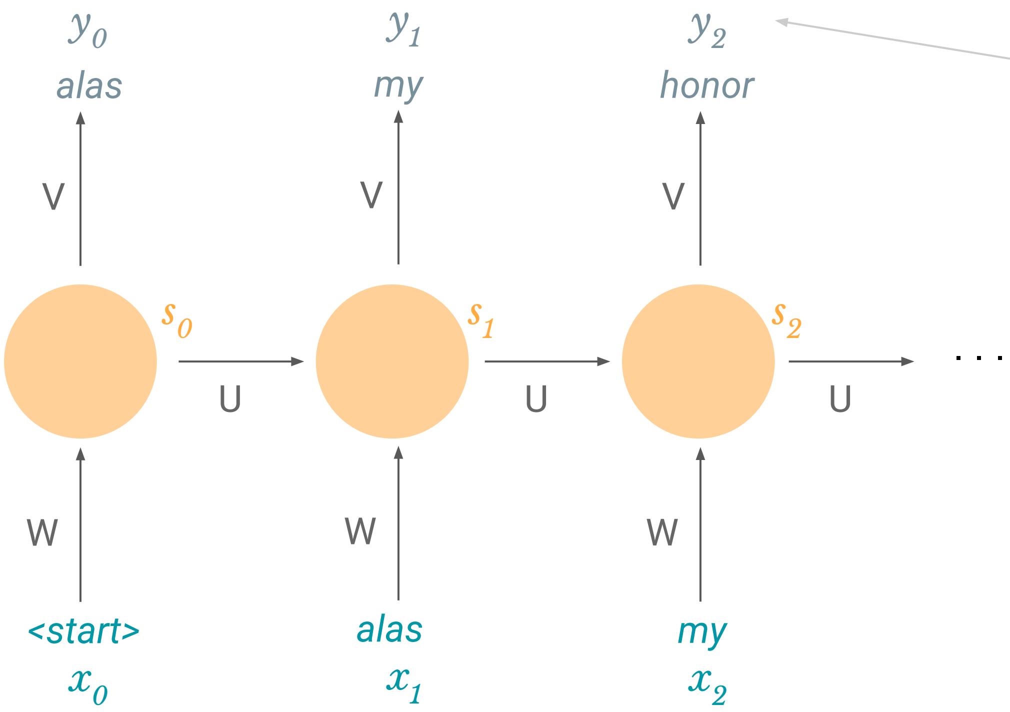
possible task: language model

all the works of shakespeare

language model

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

possible task: language model



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

possible task: language model

King James Bible,
Structure and Interpretation of Computer Programs → *language model* → 37:29 The righteous shall inherit the land, and leave it for an inheritance unto the children of Gad according to the number of steps that is linear in *b*.

hath it not been for the singular taste of old Unix, "new Unix" would not exist.

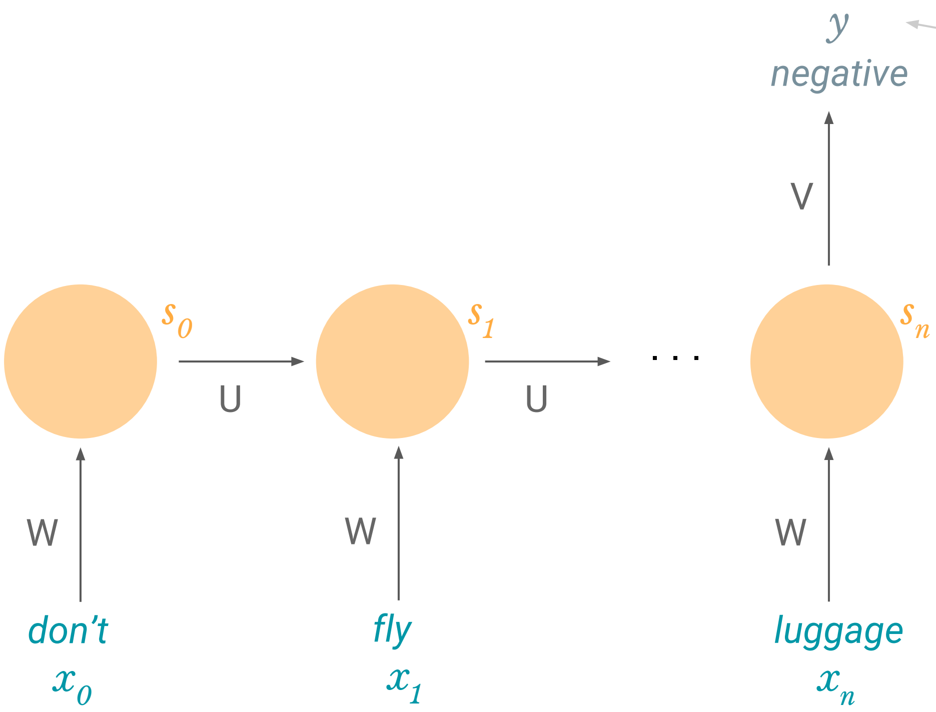
http://kingjamesprogramming.tumblr.com/

possible task: classification (i.e. sentiment)

:(

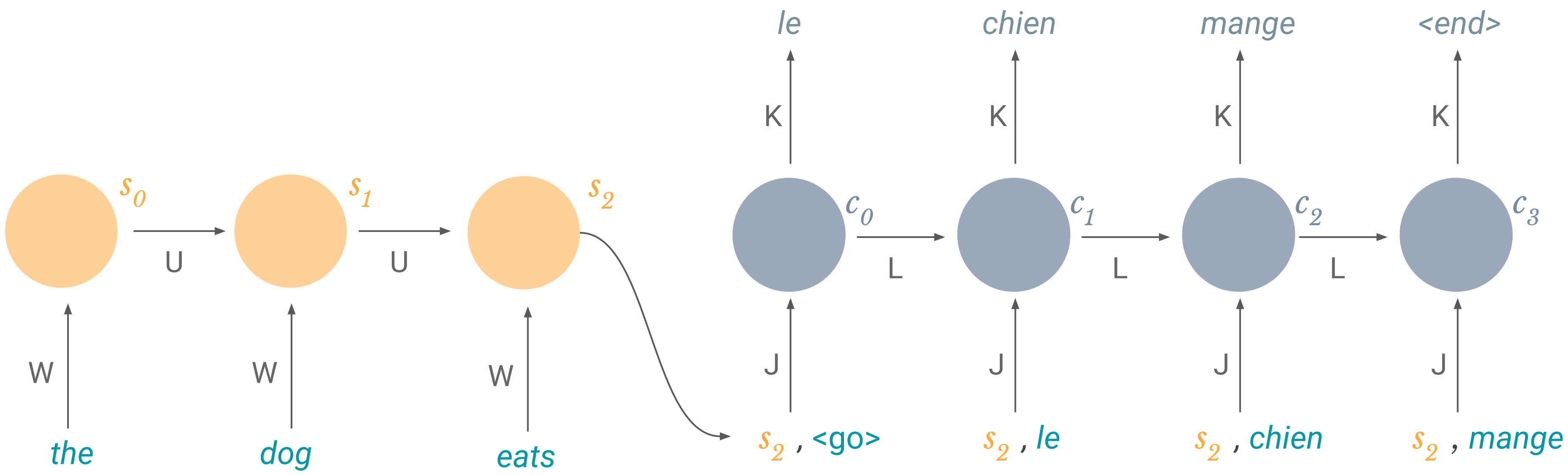
:)

possible task: classification (i.e. sentiment)



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

possible task: machine translation



how do we **train** an RNN?

how do we **train** an RNN?

backpropagation!

(through time)

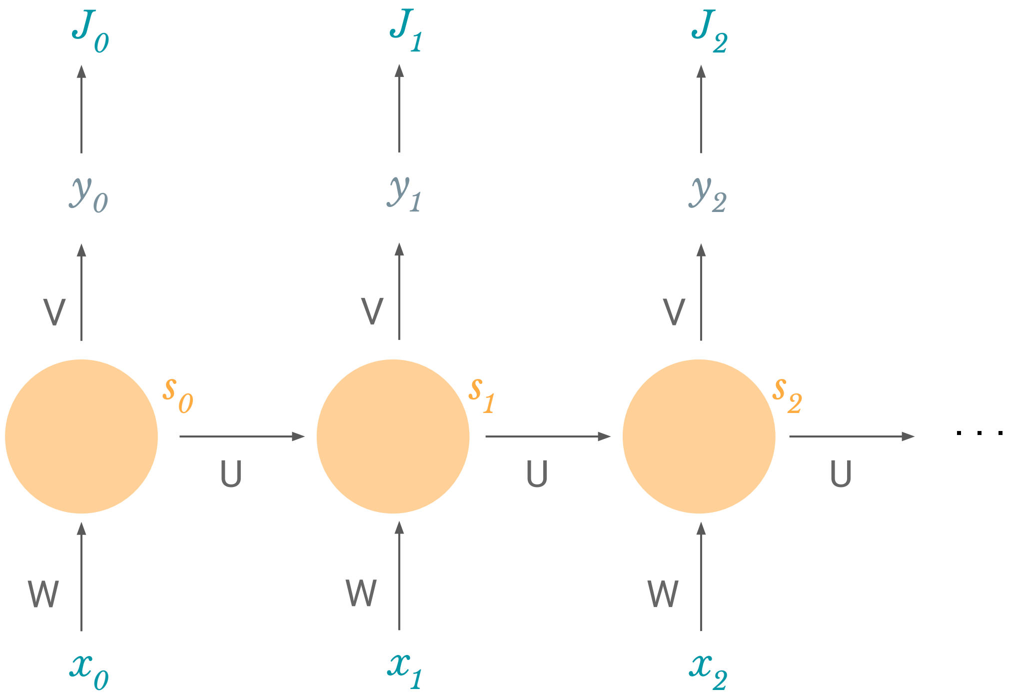
remember: **backpropagation**

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

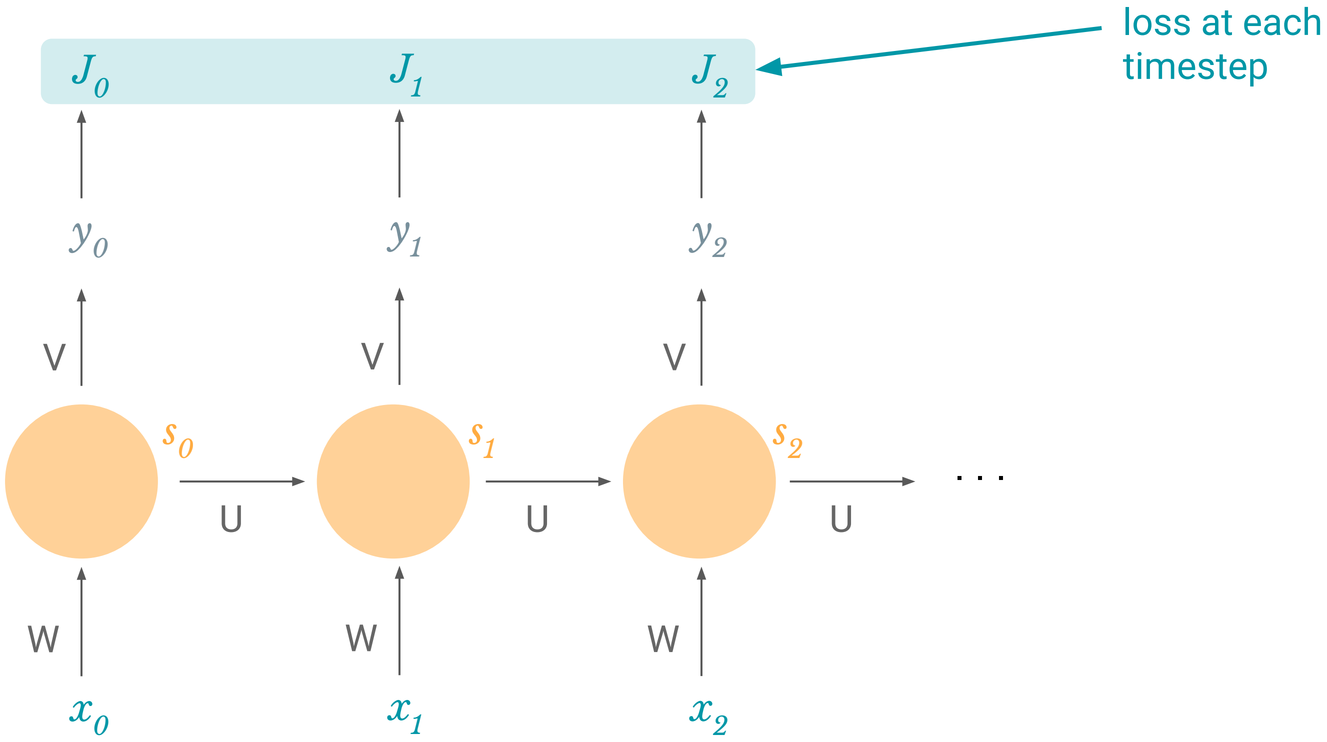
we have a **loss at each timestep**:

(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 = $J_t(\boldsymbol{\Theta})$

$\boldsymbol{\Theta}$ = our parameters, like weights

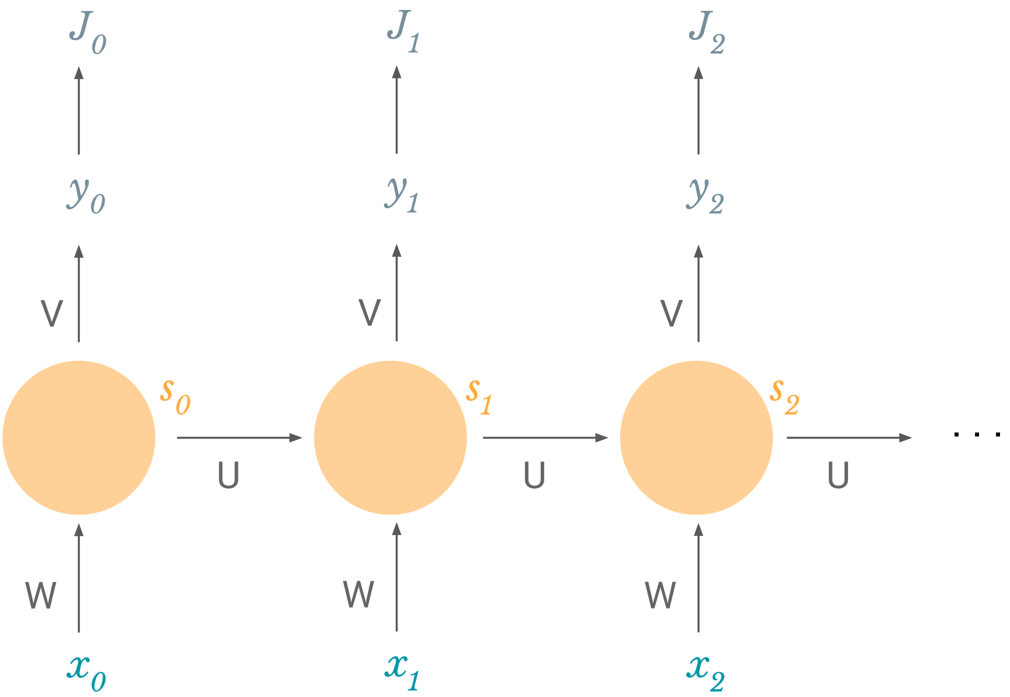
total loss = $J(\boldsymbol{\Theta}) = \Sigma_t \, J_t(\boldsymbol{\Theta})$

what are our **gradients**?

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

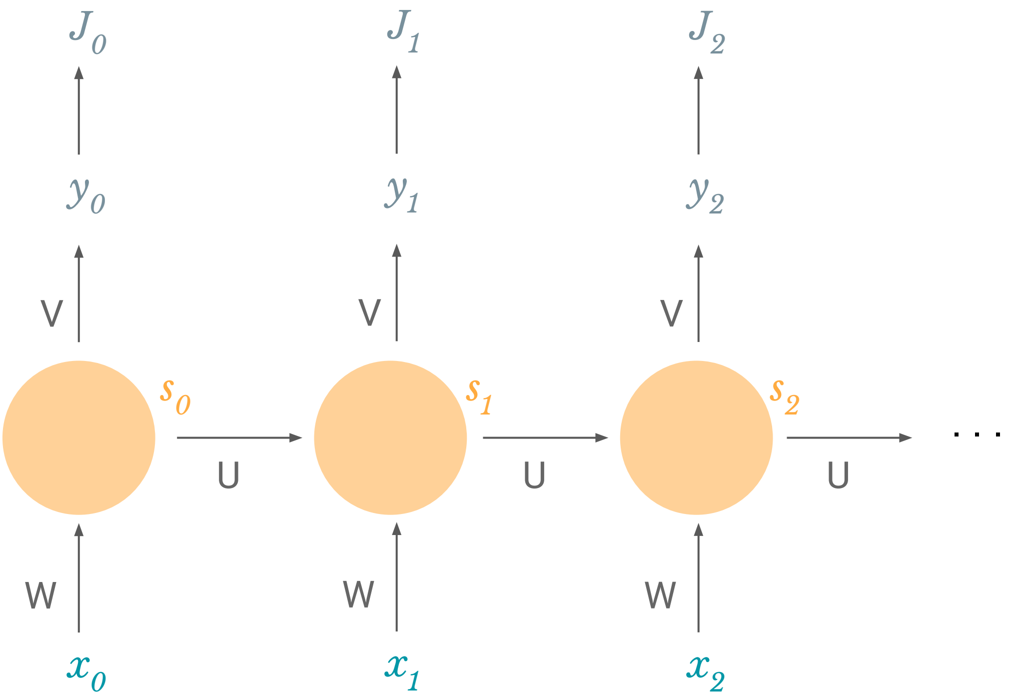
$$\frac{\partial J}{\partial P} = \sum_t \frac{\partial J_t}{\partial P}$$

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



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

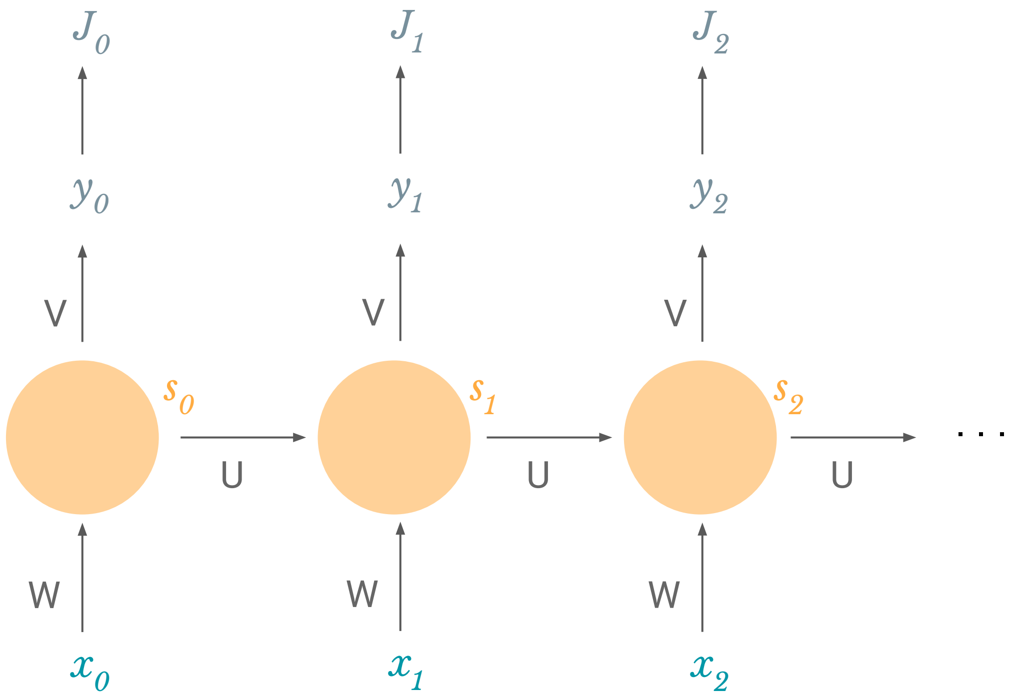
let's try it out for *W* with the **chain rule**:



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

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

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

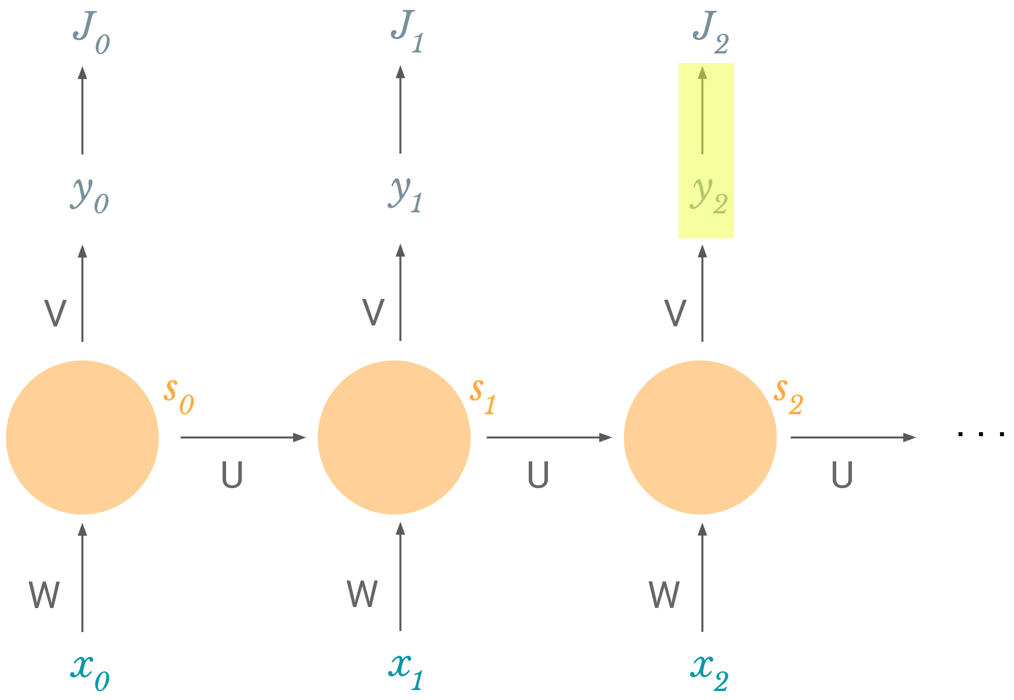


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

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**:

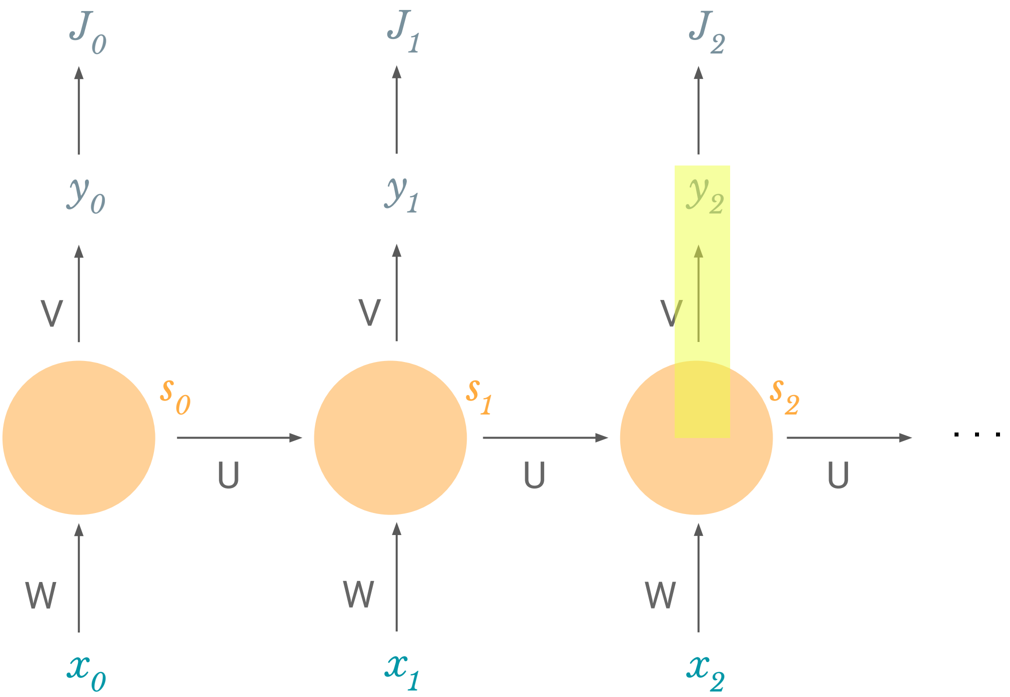


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

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**:

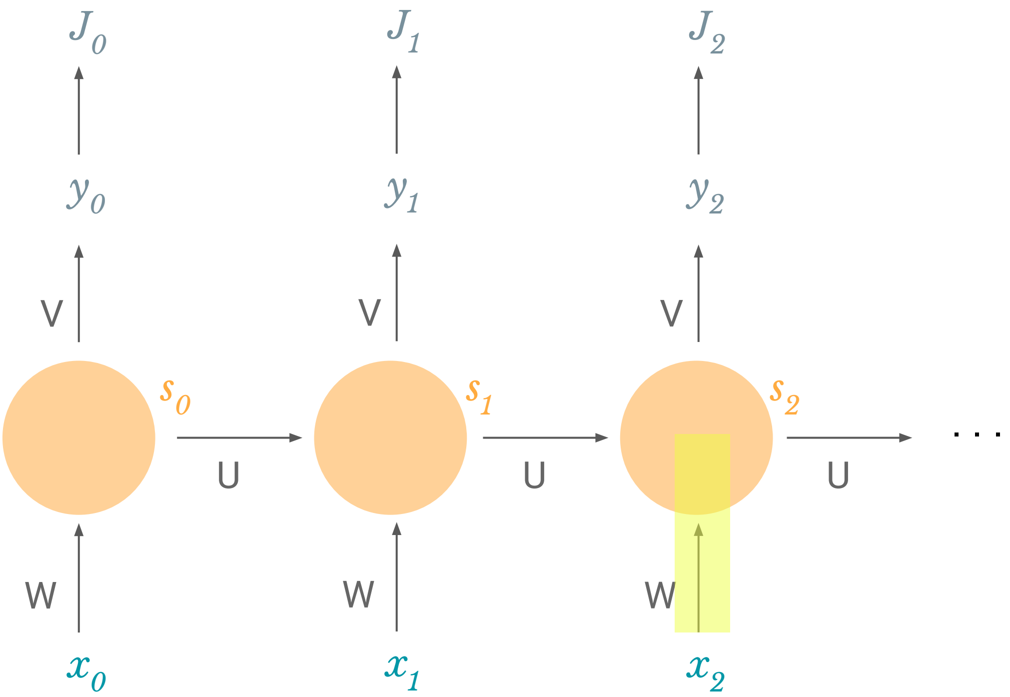


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

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**:

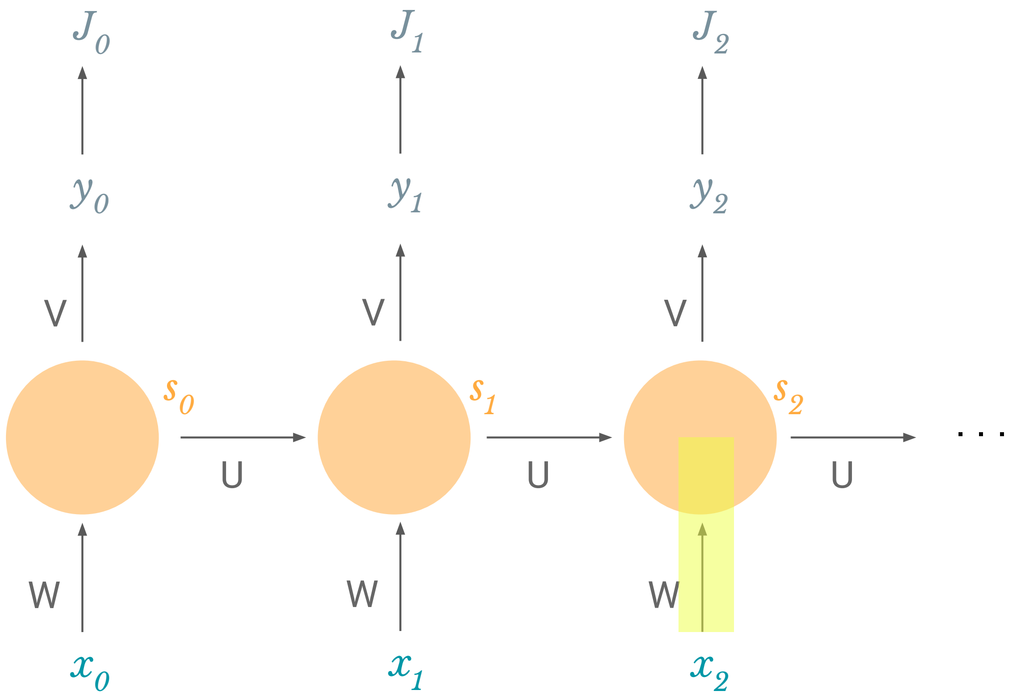


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

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**:



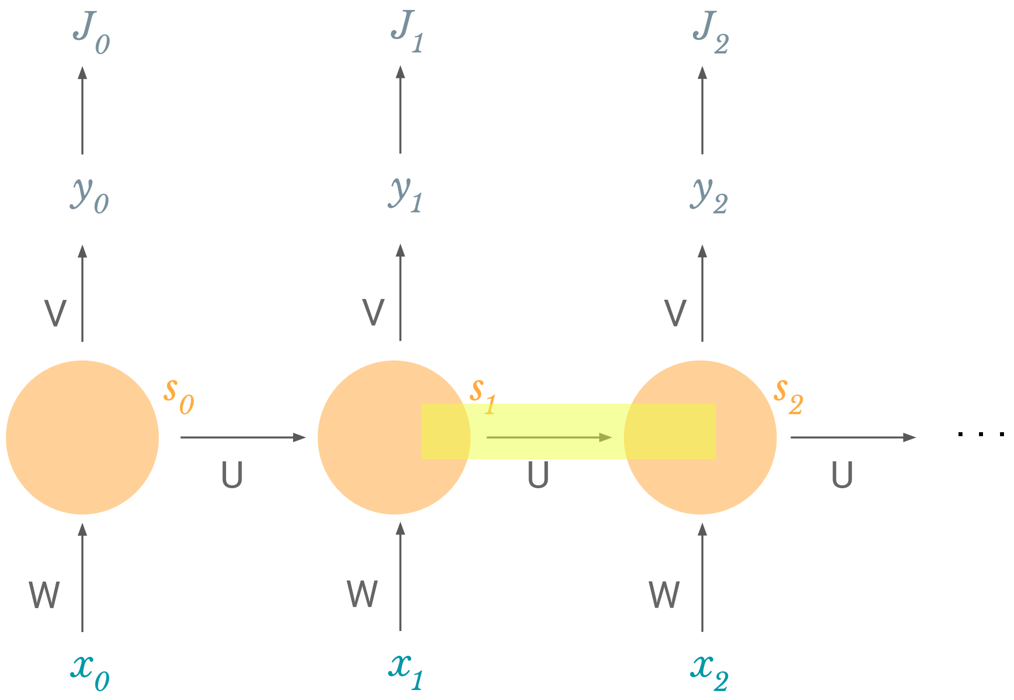
$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

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**:



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

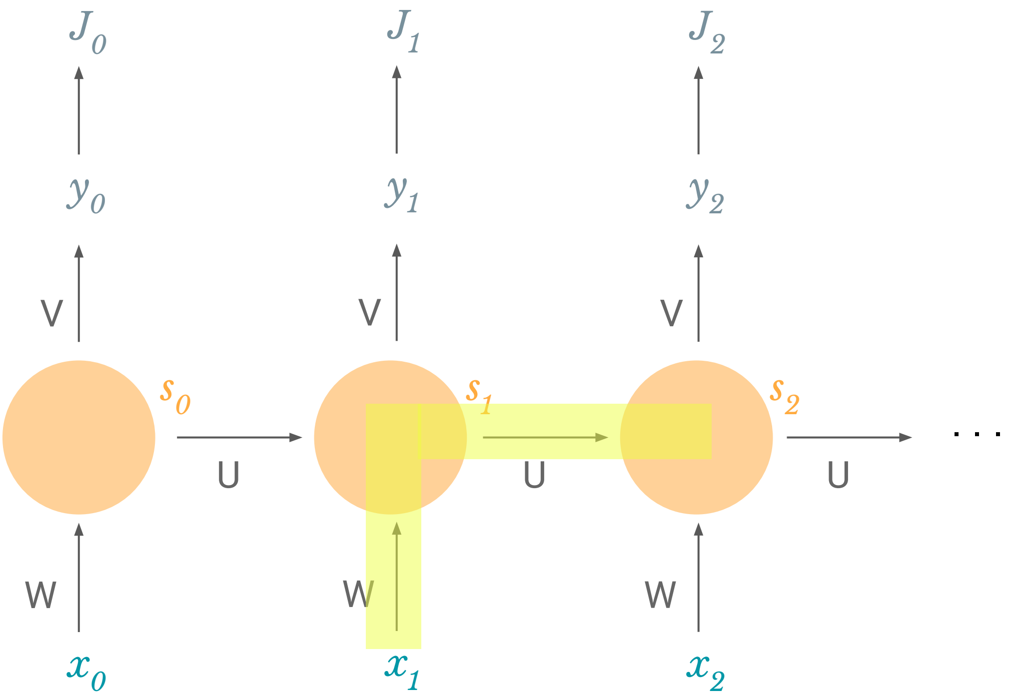
so let's take a single timestep *t*:

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

but wait...

$$s_2 = tanh(Us_1 + Wx_2)$$

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*:

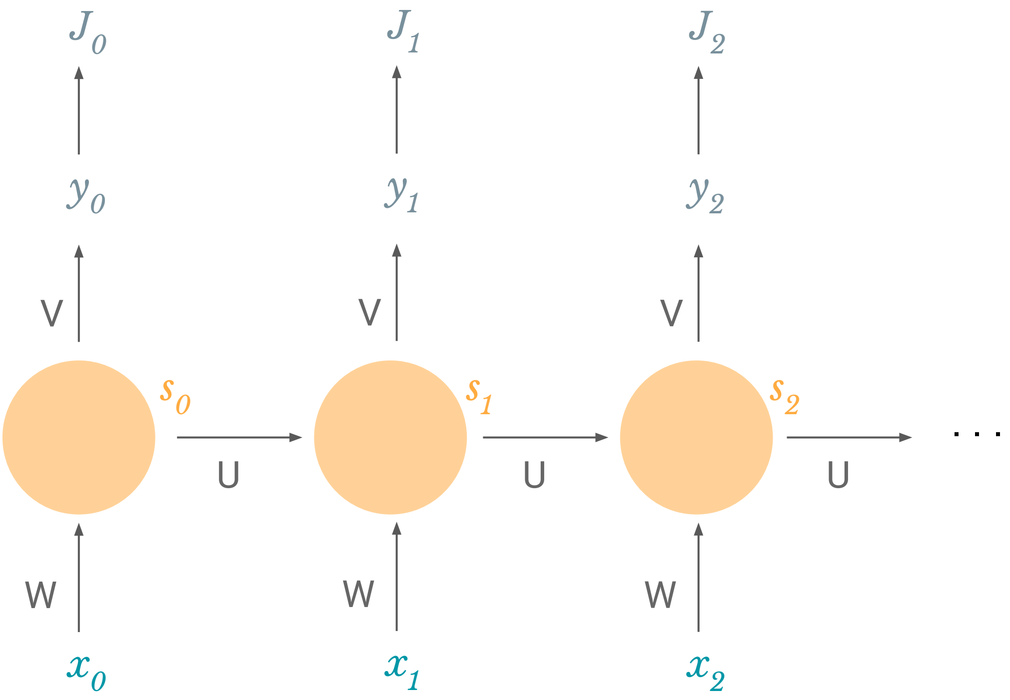
$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$

but wait…

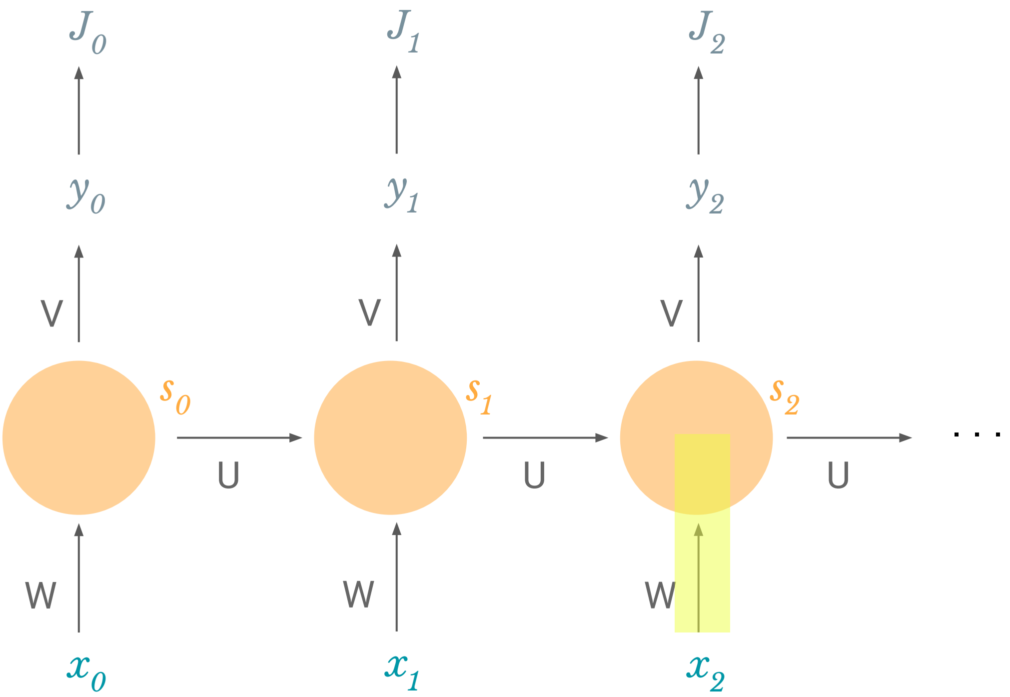
$$s_2 = tanh(Us_1 + Wx_2)$$

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?

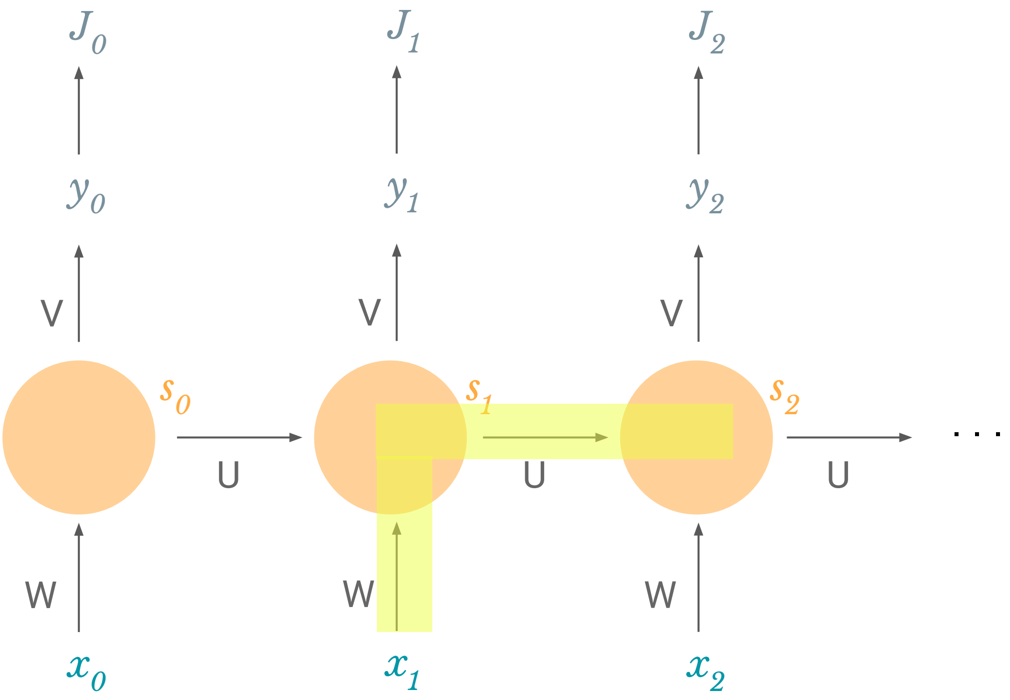


how does s_2 depend on W?



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

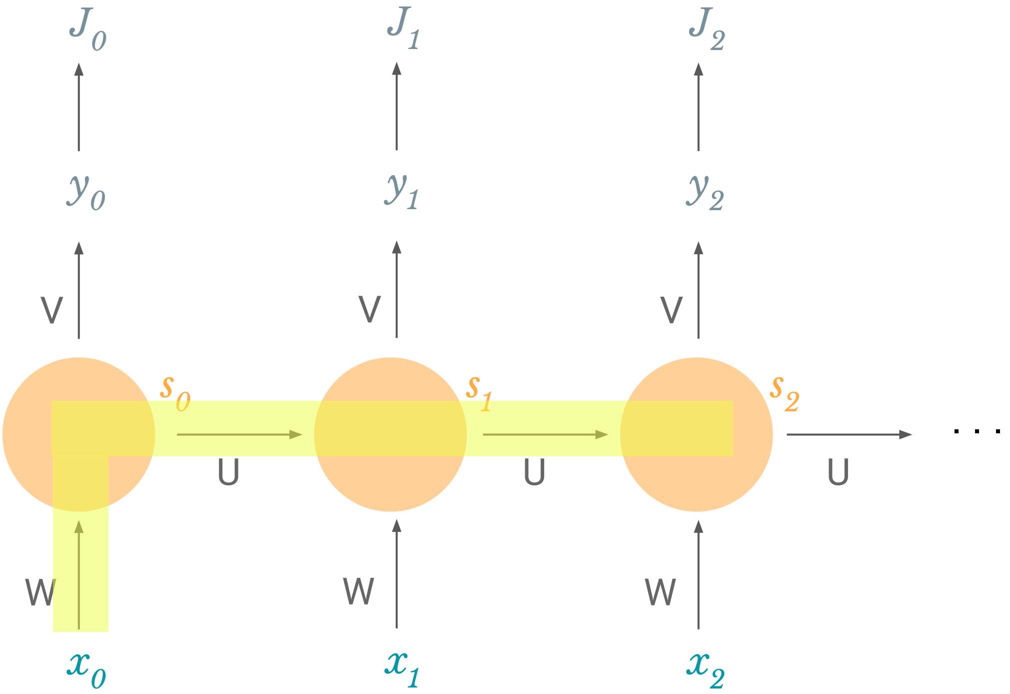
how does s_2 depend on W?



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

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

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:

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \underbrace{\frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}}_{\text{Contributions of } W \text{ in previous timesteps to the error at timestep } t}$$

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

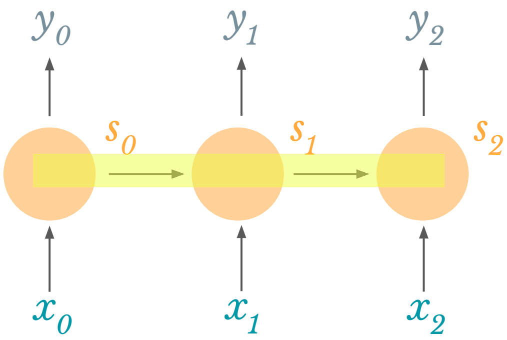
$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}$$

problem: vanishing gradient

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

problem: vanishing gradient

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

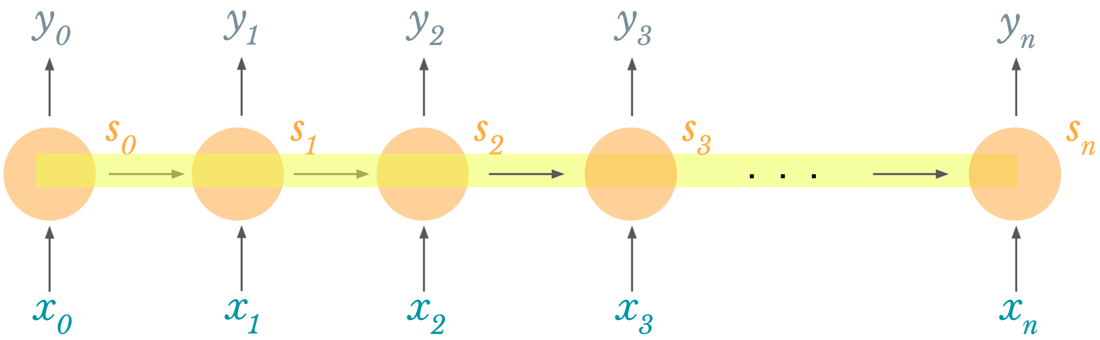


at k = 0: $$\boxed{\frac{\partial s_2}{\partial s_0} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}}$$

problem: vanishing gradient

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

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

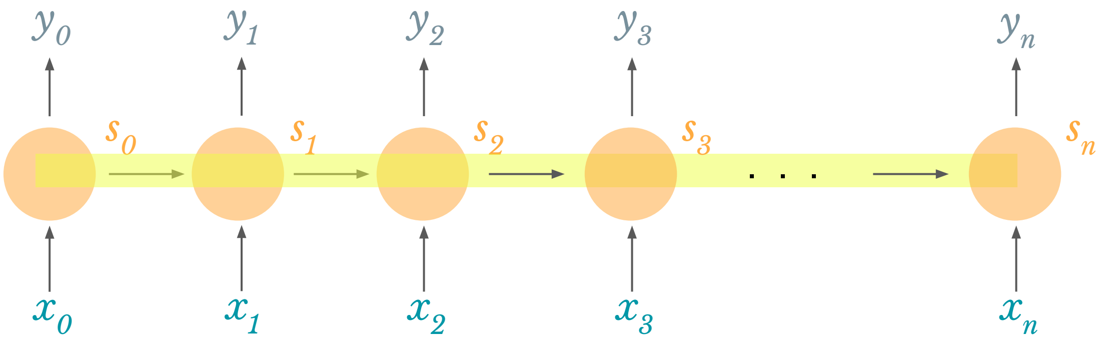


problem: vanishing gradient

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

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

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.**

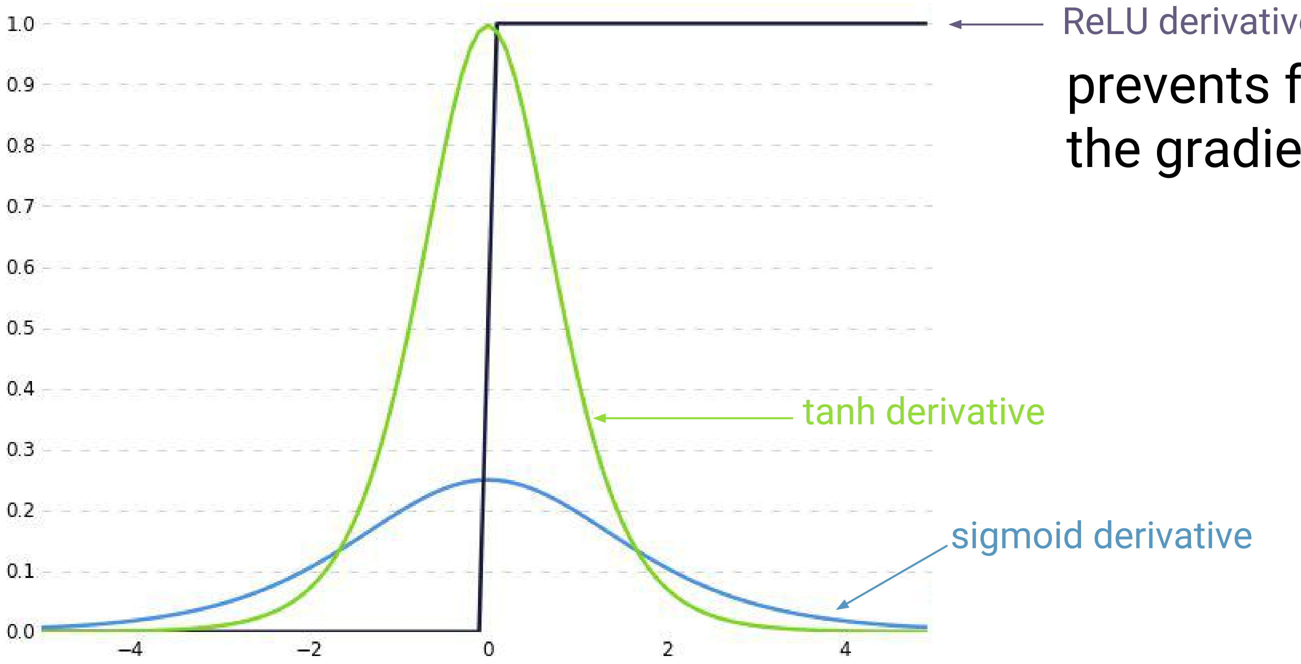
so what?

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**

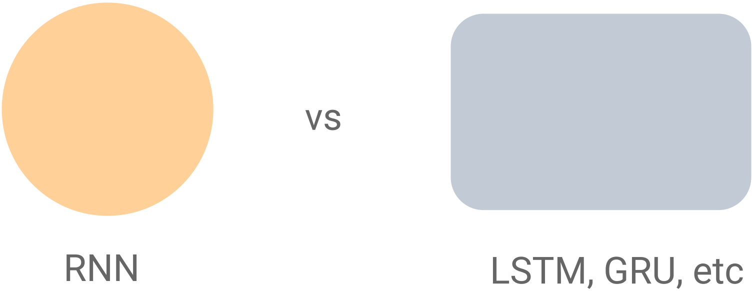
weights initialized to identity matrix
biases initialized to zeros

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

prevents W from shrinking the gradients

solution #3: **gated cells**

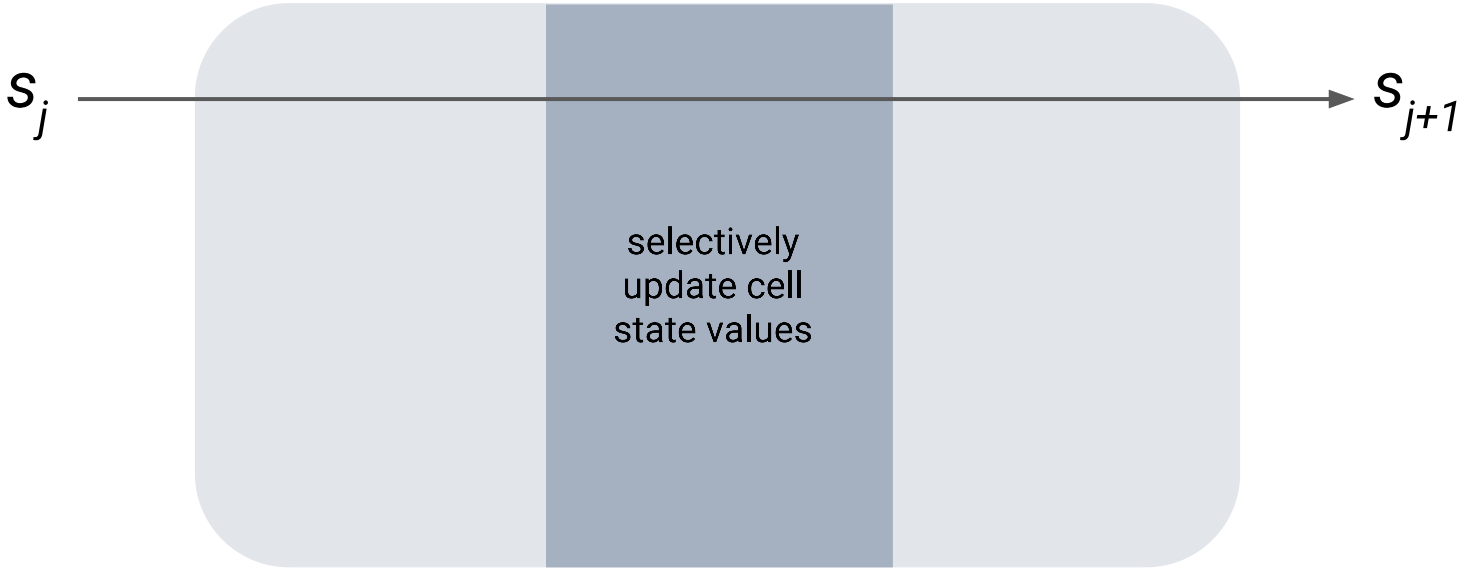
rather each node being just a simple RNN cell, make each node a more **complex unit with gates** controlling what information is passed through.



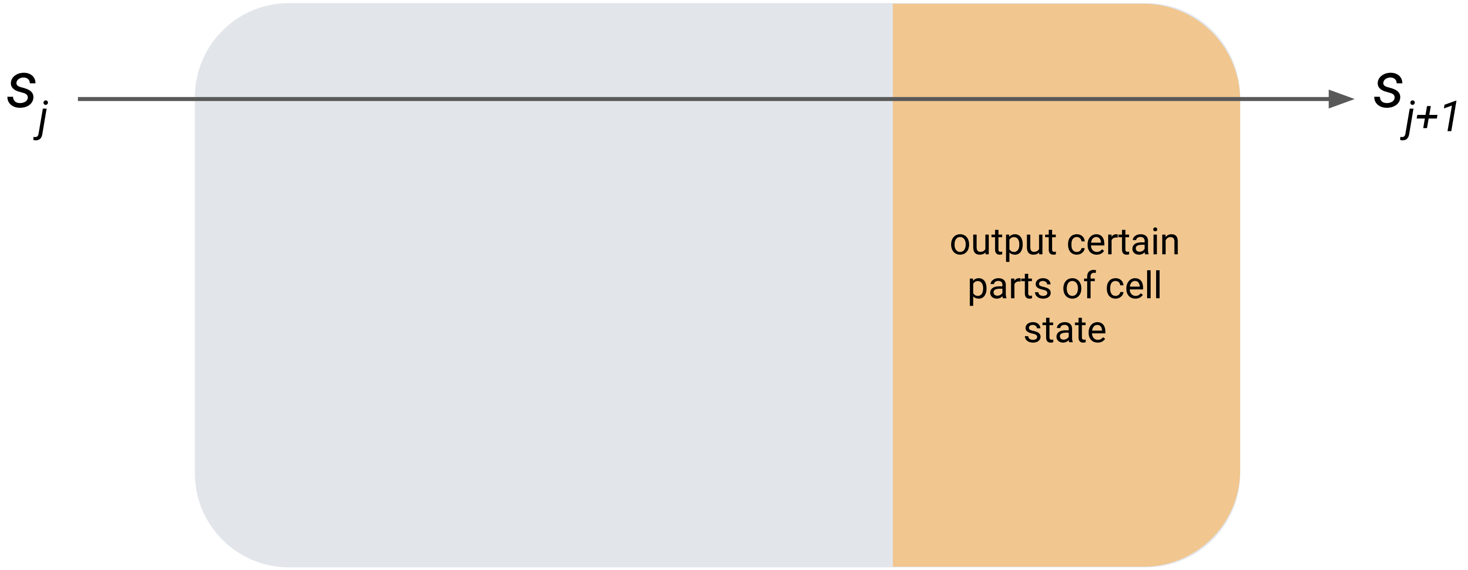
solution #3: more on **LSTMs**

solution #3: more on **LSTMs**

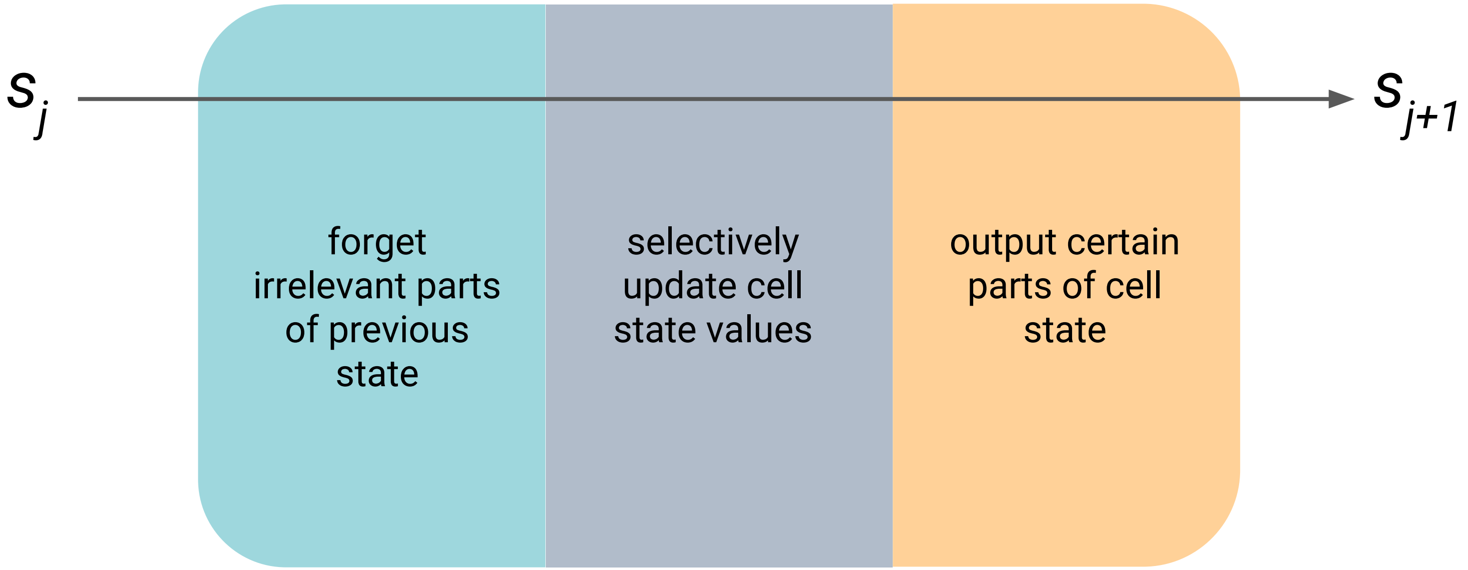
solution #3: more on **LSTMs**



solution #3: more on **LSTMs**



solution #3: more on **LSTMs**



why do LSTMs help?

1. forget gate allows information to pass through unchanged

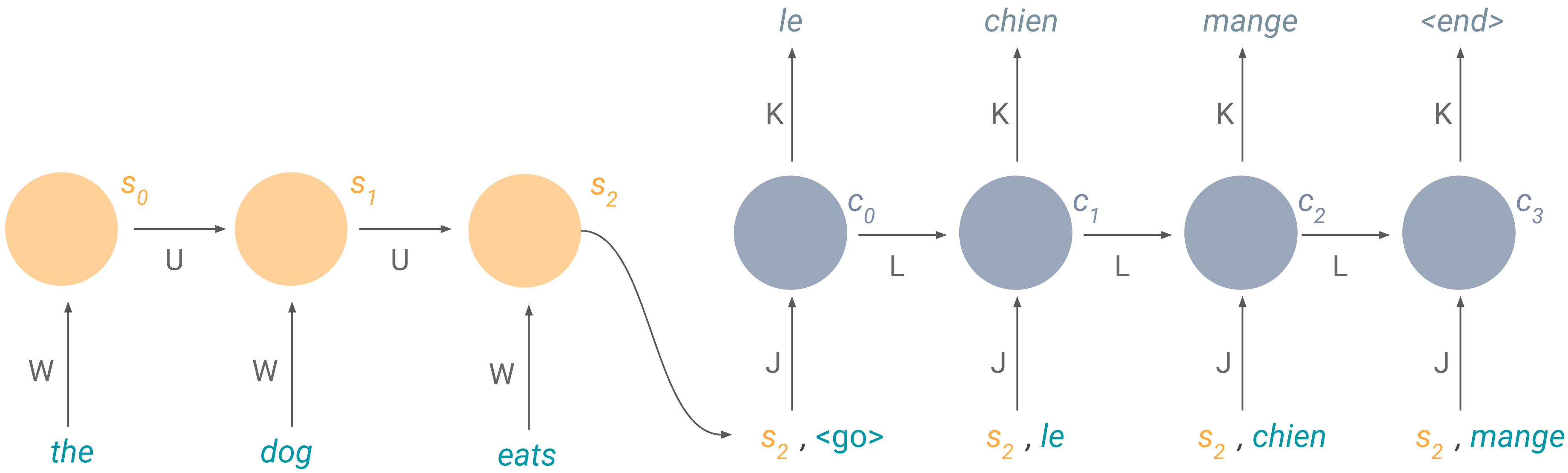
   → when taking the derivative, **f'** is 1 for what we want to keep!

2. s_j depends on s_{j-1} through addition!

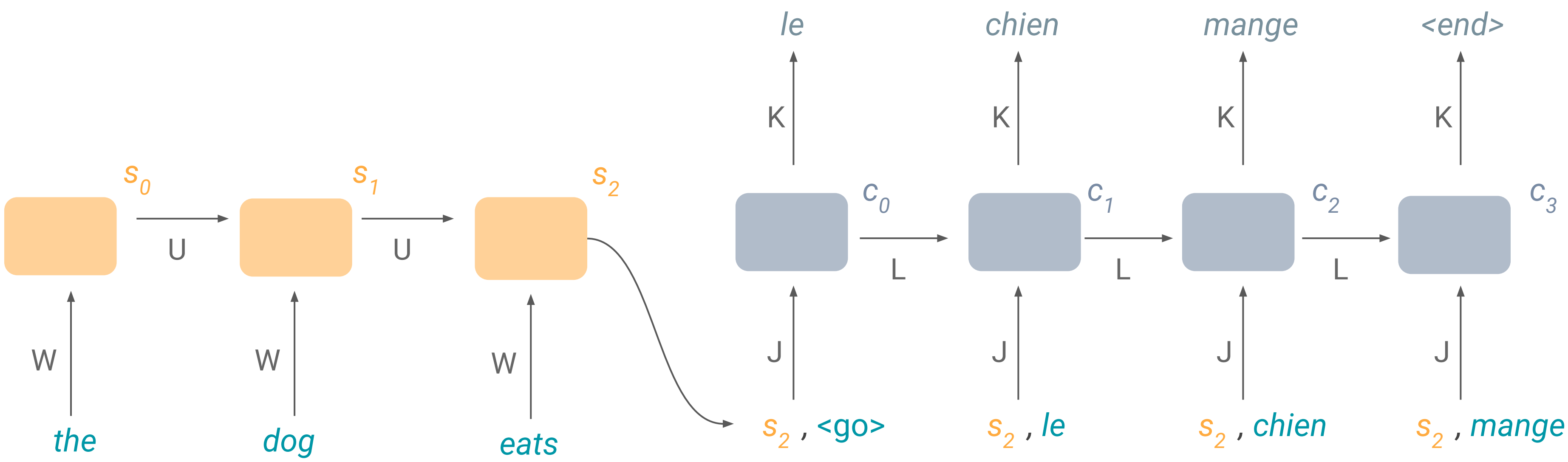
   → when taking the derivative, not lots of small **W** terms!

in practice: **machine translation**.

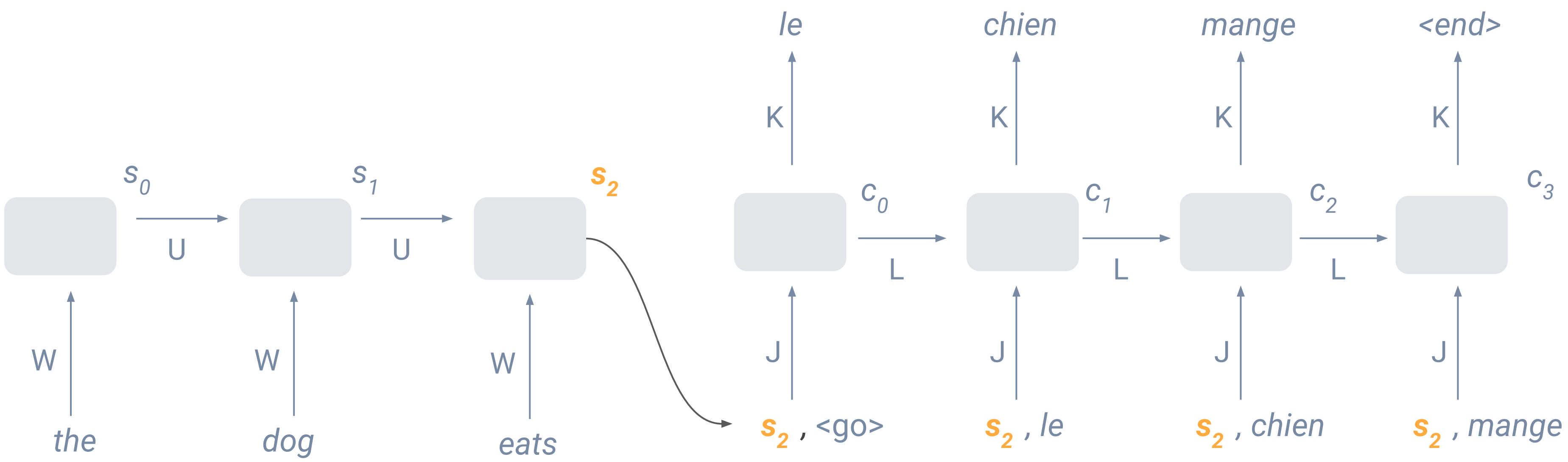
basic encoder-decoder model:



add LSTM cells:

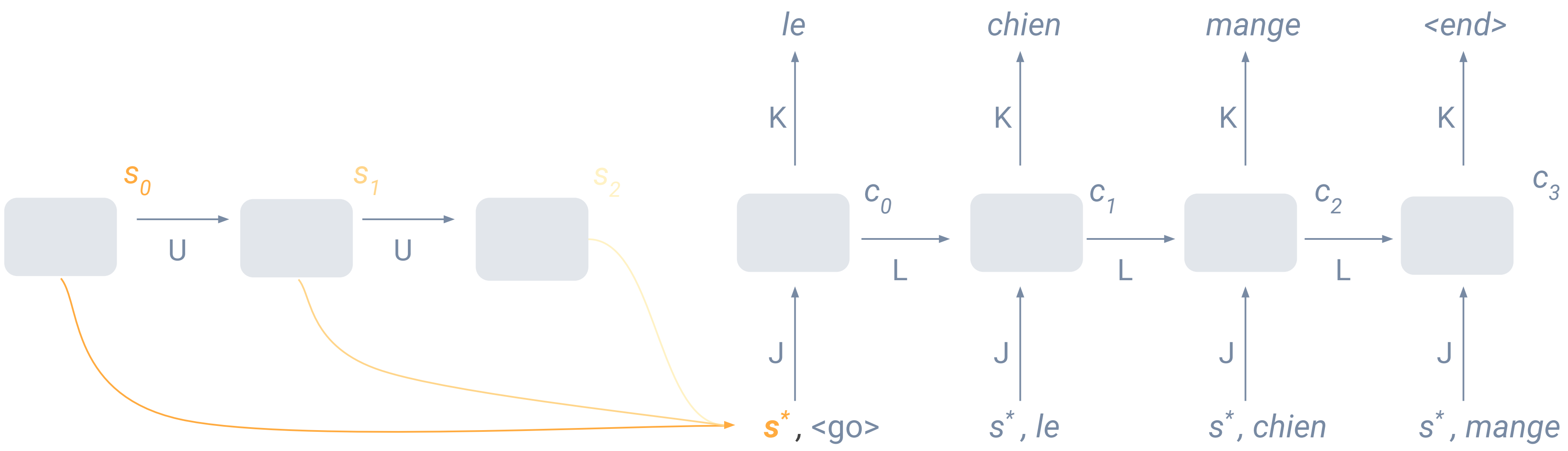


problem: a fixed-length encoding is limiting

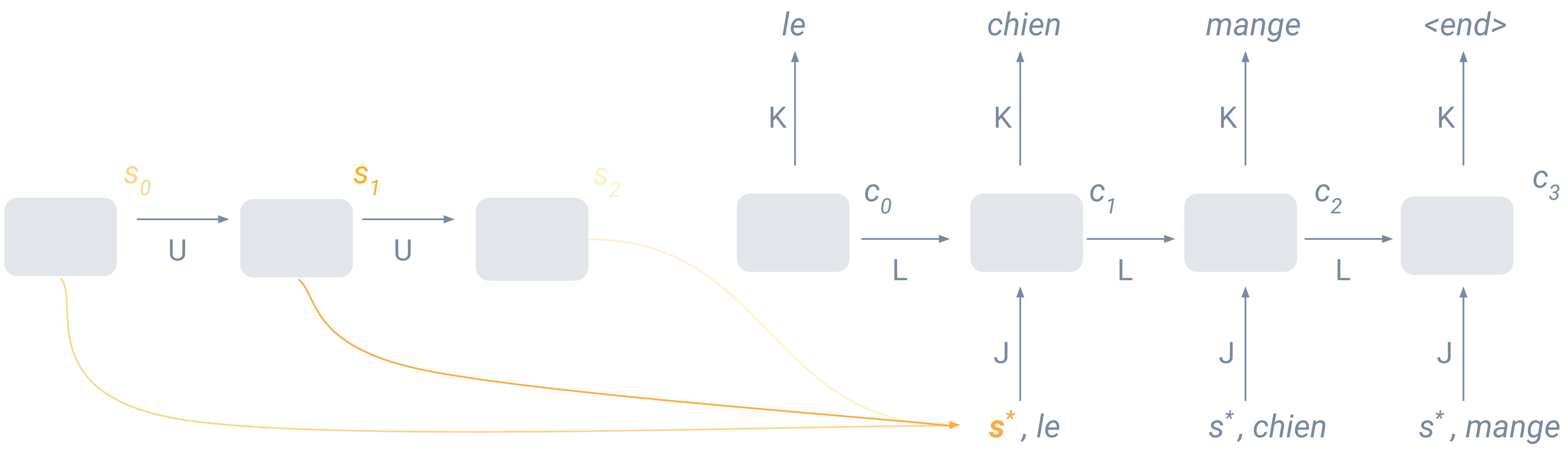


all the decoder knows about the input sentence is in one fixed length vector, s_2

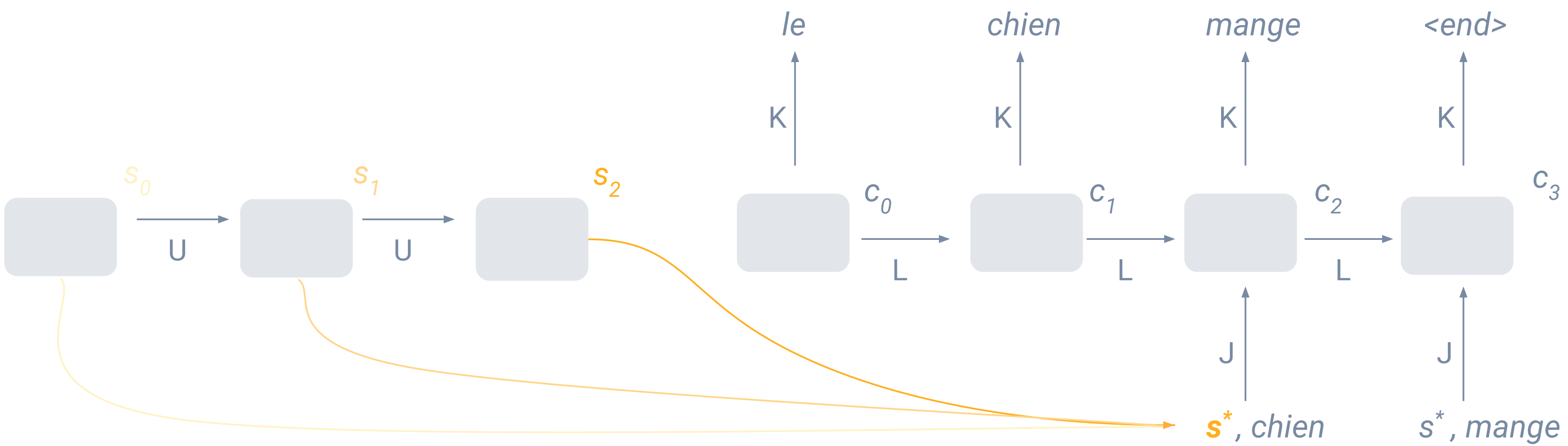
solution: attend over all encoder states



solution: attend over all encoder states



solution: attend over all encoder states



now we can model **sequences**!

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