Sequence Labeling for Part of Speech and Named Entities

Part of Speech Tagging

Edited from Dan Jurafsky's book website:
https://web.stanford.edu/~jurafsky/slp3/

Parts of Speech

From the earliest linguistic traditions (Yaska and Panini 5th C. BCE, Aristotle 4th C. BCE), the idea that words can be classified into grammatical categories

- part of speech, word classes, POS, POS tags

8 parts of speech attributed to Dionysius Thrax of Alexandria (c. 1st C. BCE):

- noun, verb, pronoun, preposition, adverb, conjunction, participle, article
- These categories are relevant for NLP today.

Two classes of words: Open vs. Closed

Closed class words

- Relatively fixed membership
- Usually **function** words: short, frequent words with grammatical function
  - determiners: ***a, an, the***
  - pronouns: ***she, he, I***
  - prepositions: ***on, under, over, near, by, …***

Open class words

- Usually **content** words: Nouns, Verbs, Adjectives, Adverbs
  - Plus interjections: **oh, ouch, uh-huh, yes, hello**
- New nouns and verbs like *iPhone* or *to fax*

Open class ("content") words

Nouns

Proper

Janet
Italy

Common

cat, cats
mango

Verbs

Main

eat
went

Auxiliary

can
had

Adjectives *old green tasty*

Adverbs *slowly yesterday*

Numbers

122,312
one

Interjections *Ow hello*

… more

Closed class ("function")

Determiners *the some*

Conjunctions *and or*

Pronouns *they its*

Prepositions *to with*

Particles *off up*

… more

Part-of-Speech Tagging

Assigning a part-of-speech to each word in a text.

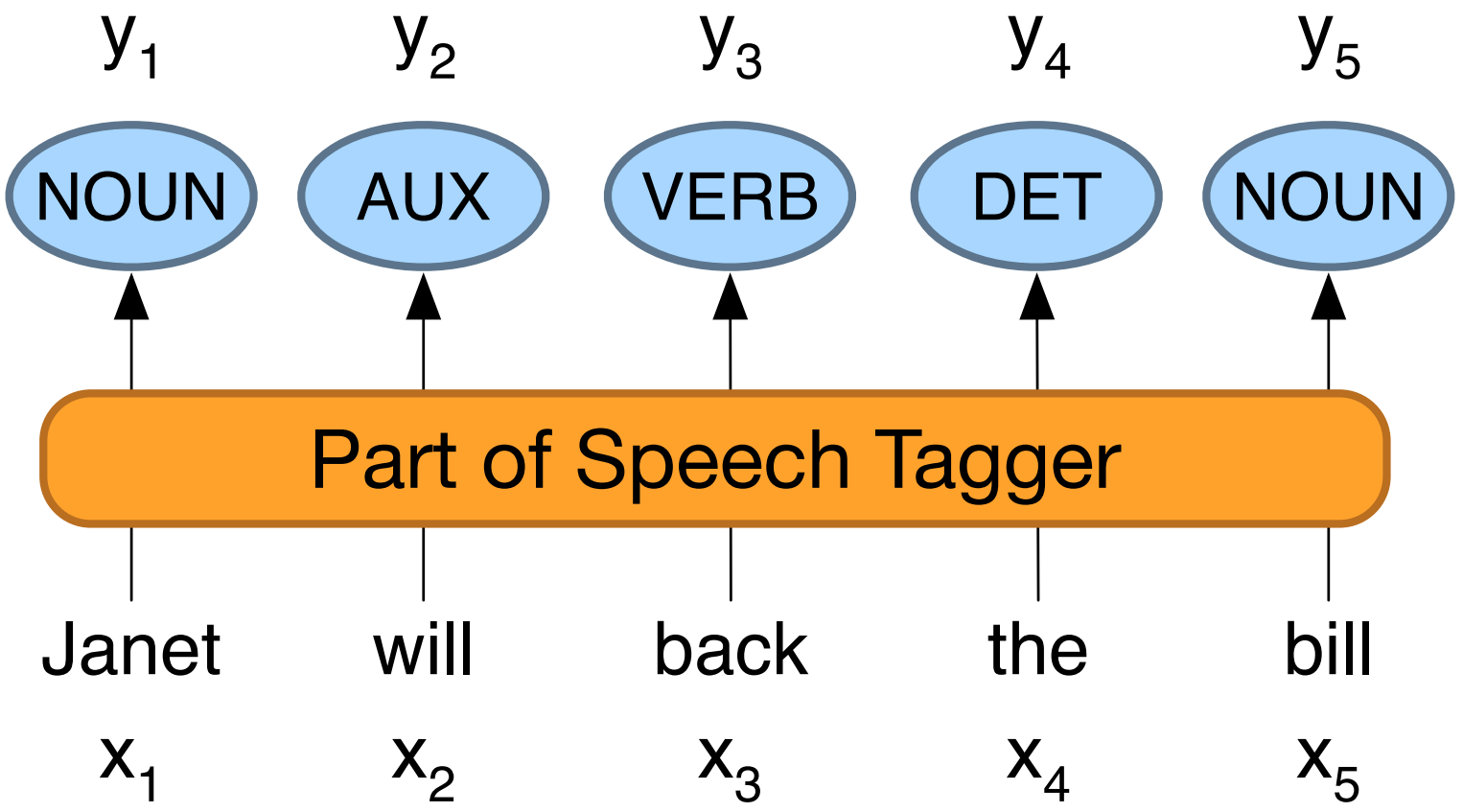
Words often have more than one POS.

book:

- VERB: (***Book*** *that flight*)
- NOUN: (*Hand me that* ***book***).

Part-of-Speech Tagging

Map from sequence $x_1,\ldots,x_n$ of words to $y_1,\ldots,y_n$ of POS tags



"Universal Dependencies" Tagset

Nivre et al. 2016

	Tag	Description	Example
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red*, *young*, *awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very*, *slowly*, *home*, *yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm*, *cat*, *mango*, *beauty*
	VERB	words for actions and processes	*draw*, *provide*, *go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina*, *IBM*, *Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh*, *um*, *yes*, *hello*
Closed Class Words	**ADP**	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and*, *or*, *but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*that*, *which*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

Sample "Tagged" English sentences

There/PRO were/VERB 70/NUM children/NOUN there/ADV ./PUNC

Preliminary/ADJ findings/NOUN were/AUX reported/VERB in/ADP today/NOUN ’s/PART New/PROPN England/PROPN Journal/PROPN of/ADP Medicine/PROPN

Why Part of Speech Tagging?

- Can be useful for other NLP tasks
  - Parsing: POS tagging can improve syntactic parsing
  - MT: reordering of adjectives and nouns (say from Spanish to English)
  - Sentiment or affective tasks: may want to distinguish adjectives or other POS
  - Text-to-speech (how do we pronounce “lead” or "object"?)
- Or linguistic or language-analytic computational tasks
  - Need to control for POS when studying linguistic change like creation of new words, or meaning shift
  - Or control for POS in measuring meaning similarity or difference

How difficult is POS tagging in English?

Roughly 15% of word types are ambiguous

- Hence 85% of word types are unambiguous
- *Janet* is always PROPN, *hesitantly* is always ADV

But those 15% tend to be very common.

So ~60% of word tokens are ambiguous

E.g., *back*

earnings growth took a back/ADJ seat
a small building in the back/NOUN
a clear majority of senators back/VERB the bill
enable the country to buy back/PART debt
I was twenty-one back/ADV then

POS tagging performance in English

How many tags are correct? (Tag accuracy)

- About 97%
  - Hasn't changed in the last 10+ years
  - HMMs, CRFs, BERT perform similarly .
  - Human accuracy about the same

But baseline is 92%!

- Baseline is performance of stupidest possible method
  - "Most frequent class baseline" is an important baseline for many tasks
    - Tag every word with its most frequent tag
    - (and tag unknown words as nouns)
- Partly easy because
  - Many words are unambiguous

Sources of information for POS tagging

Janet will back the bill
AUX/NOUN/VERB? NOUN/VERB?

Prior probabilities of word/tag
- "will" is usually an AUX

Identity of neighboring words
- "the" means the next word is probably not a verb

Morphology and wordshape:
- Prefixes — unable: un- → ADJ
- Suffixes — importantly: -ly → ADJ
- Capitalization — Janet: CAP → PROPN

Standard algorithms for POS tagging

Supervised Machine Learning Algorithms:

- Hidden Markov Models
- Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Large Language Models (like BERT), finetuned

All required a hand-labeled training set, all about equal performance (97% on English)

All make use of information sources we discussed

- Via human created features: HMMs and CRFs
- Via representation learning: Neural LMs

Sequence Labeling for Part of Speech and Named Entities

Part of Speech Tagging

Sequence Labeling for Part of Speech and Named Entities

Named Entity Recognition (NER)

Named Entities

- **Named entity**, in its core usage, means anything that can be referred to with a proper name. Most common 4 tags:
  - PER (Person): “Marie Curie”
  - LOC (Location): “New York City”
  - ORG (Organization): “Stanford University”
  - GPE (Geo-Political Entity): "Boulder, Colorado"
- Often multi-word phrases
- But the term is also extended to things that aren't entities:
  - dates, times, prices

Named Entity tagging

The task of named entity recognition (NER):

- find spans of text that constitute proper names
- tag the type of the entity.

NER output

Citing high fuel prices, [ORG **United Airlines**] said [TIME **Friday**] it has increased fares by [MONEY **$6**] per round trip on flights to some cities also served by lower-cost carriers. [ORG **American Airlines**], a unit of [ORG **AMR Corp.**], immediately matched the move, spokesman [PER **Tim Wagner**] said. [ORG **United**], a unit of [ORG **UAL Corp.**], said the increase took effect [TIME **Thursday**] and applies to most routes where it competes against discount carriers, such as [LOC **Chicago**] to [LOC **Dallas**] and [LOC **Denver**] to [LOC **San Francisco**].

Why NER?

Sentiment analysis: consumer's sentiment toward a particular company or person?

Question Answering: answer questions about an entity?

Information Extraction: Extracting facts about entities from text.

Why NER is hard

1) Segmentation
   - In POS tagging, no segmentation problem since each word gets one tag.
   - In NER we have to find and segment the entities!
2) Type ambiguity

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.

BIO Tagging

How can we turn this structured problem into a sequence problem like POS tagging, with one label per word?

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

BIO Tagging

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

Now we have one tag per token!!!

BIO Tagging

B: token that *begins* a span

I: tokens *inside* a span

O: tokens outside of any span

# of tags (where n is #entity types):

1 O tag,

n B tags,

n I tags

total of *2n+1*

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

BIO Tagging variants: IO and BIOES

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

Standard algorithms for NER

Supervised Machine Learning given a human-labeled training set of text annotated with tags

- Hidden Markov Models
- Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Large Language Models (like BERT), finetuned

Sequence Labeling for Part of Speech and Named Entities

Named Entity Recognition (NER)

Recurrent Neural Networks

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Network

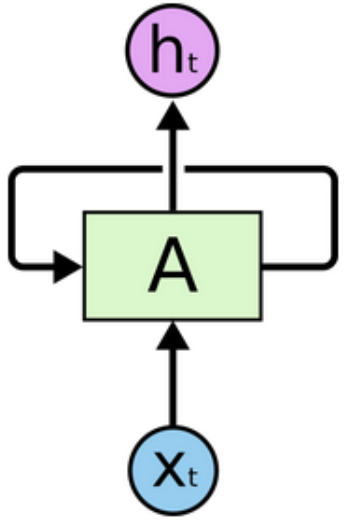
The way we process and understand language depends on context

- When we read, we don’t begin again from scratch to understand each word in a sentence
- We use our understanding of previous words in the sentence to understand the current word
- Traditional NN doesn’t take context into account
- RNN addresses this issue

Recurrent Neural Network

Networks with loops

Allows information to persist



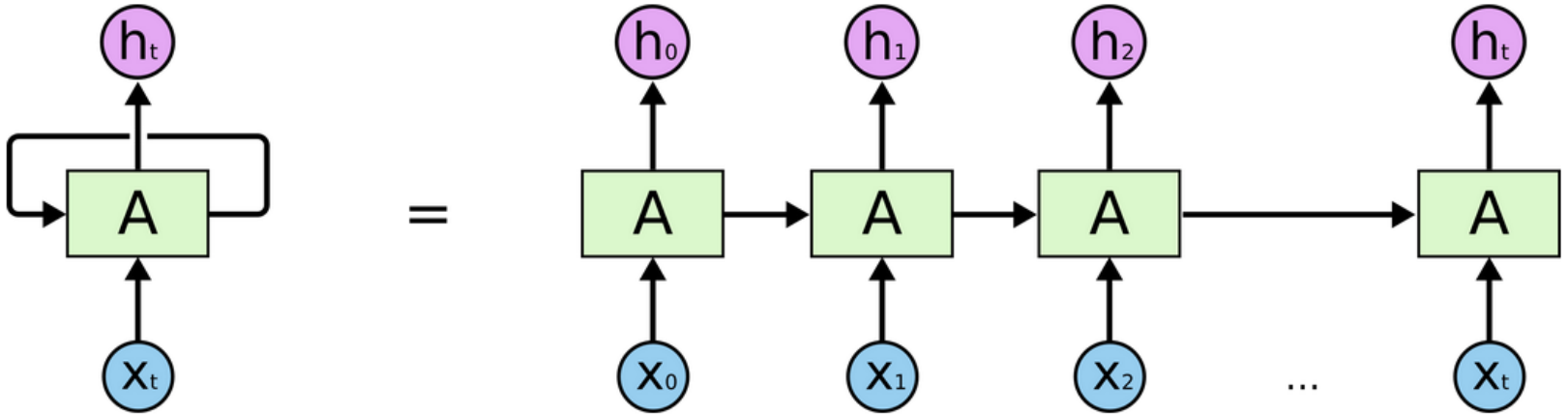
Network A
Input x_t
Output h_t

Recurrent Neural Network

Allows information to persist

Copies of the same network, each passing a message to a successor

C



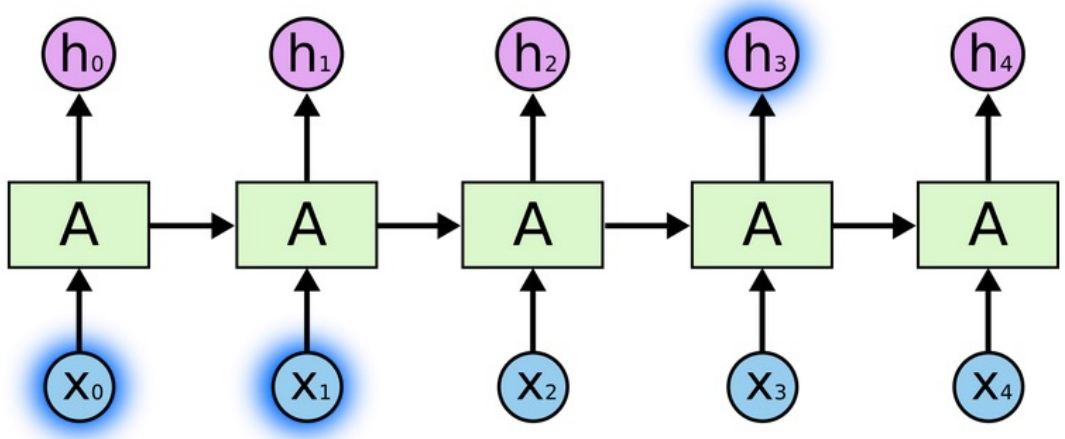
Recurrent Neural Network

Used in speech recognition, language modeling, translation, image captioning, …

Long-term Dependencies

RNN might be able to connect previous information to the present task

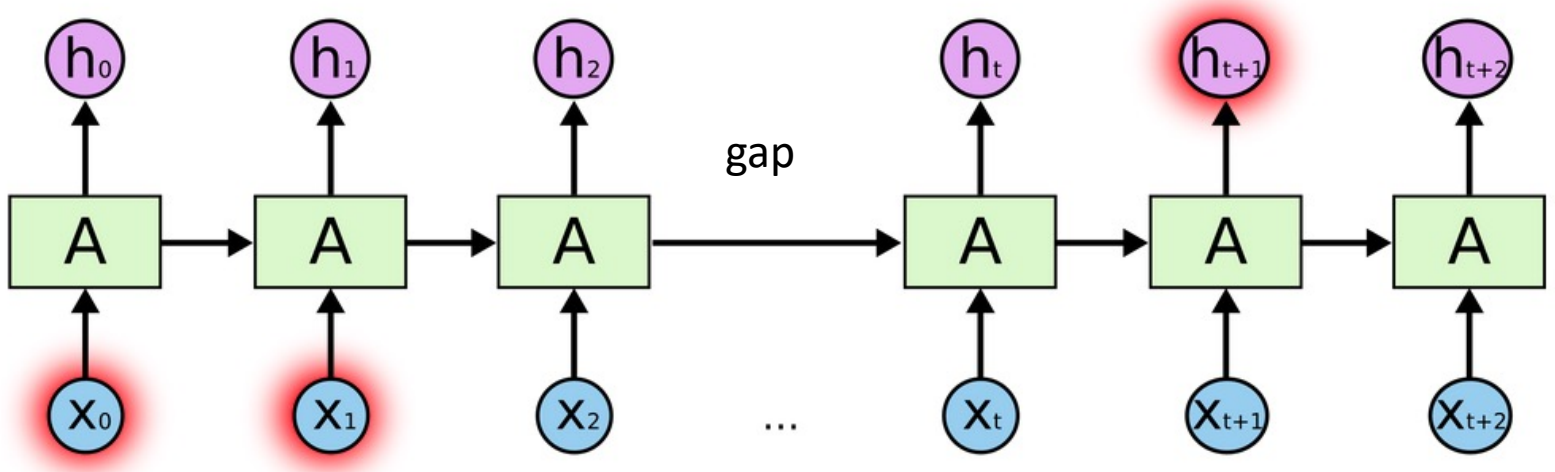
- When the gap between the relevant information and the place it is needed is small
- E.g., predict the last word in ”the clouds are in the XXX”



Long-term Dependencies

RNN might be able to connect previous information to the present task

- But there are cases where we need more context – it is possible that the gap between the relevant information and the point where it's needed to become very large



Long-term Dependencies

RNN might be able to connect previous information to the present task

- As the gap grows, RNNs become unable to learn to connect the information
- E.g., “I grew up in France. I learn to cycle when I was very young but only learned to swim as an adult. I also love to cook and bake. I can make a mean cake. Since I grew up there, I also speak fluent XXX”

Long-term Dependencies

Vanishing gradient problem

Impossible for the model to learn correlation between temporally distant events

"On the difficulty of training RNNs", Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, ICML, 2013

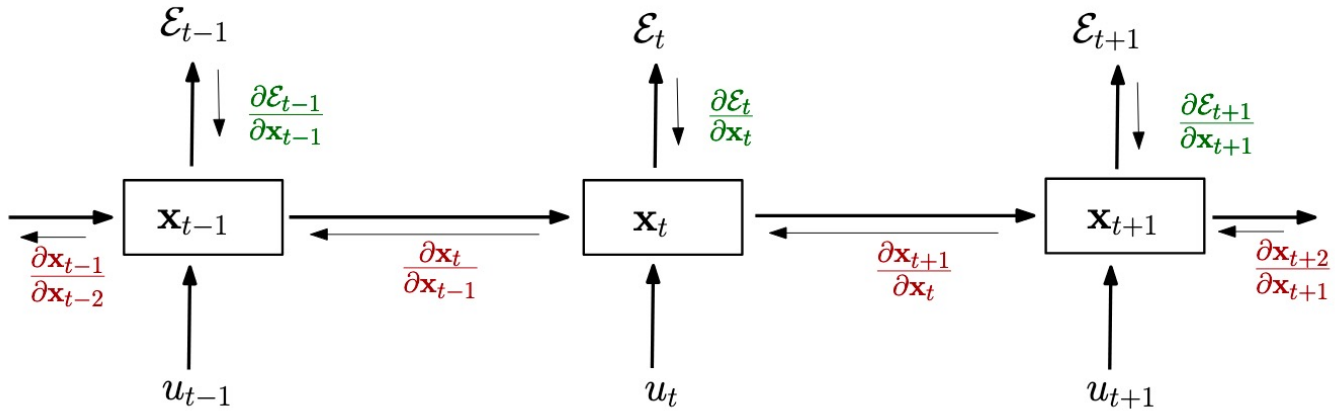
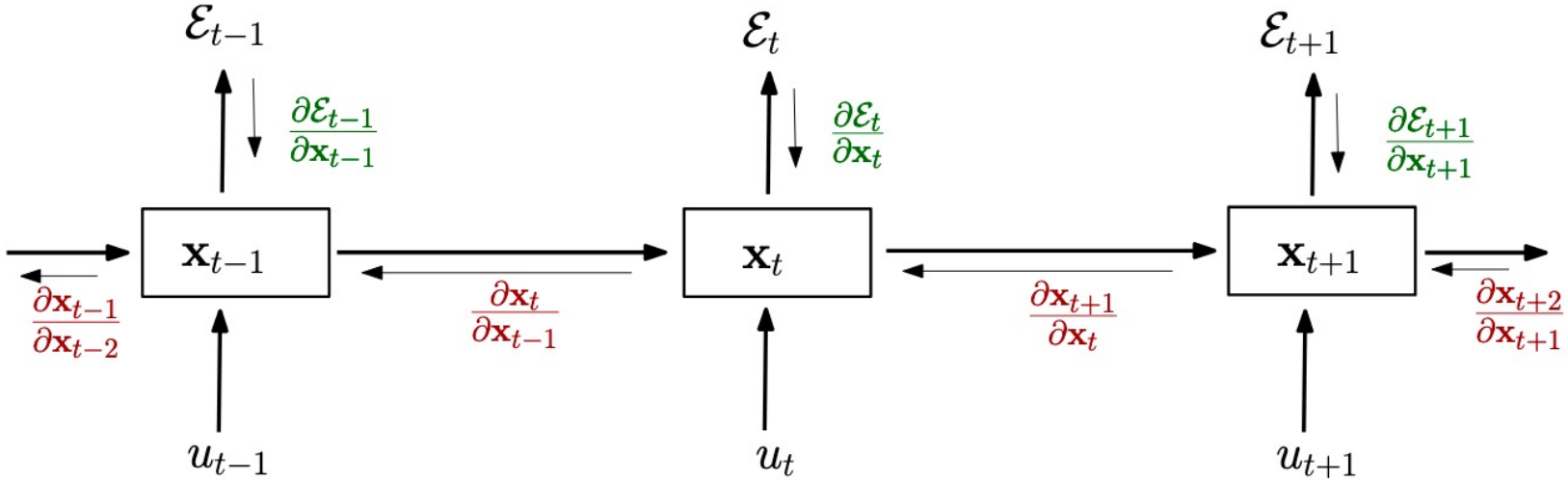


Figure 2. Unrolling recurrent neural networks in time by creating a copy of the model for each time step. We denote by $\mathbf{x}_t$ the hidden state of the network at time t, by $\mathbf{u}_t$ the input of the network at time t and by $\mathcal{E}_t$ the error obtained from the output at time t.

Long-term Dependencies



$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial \mathcal{E}_t}{\partial \theta}$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right)$$

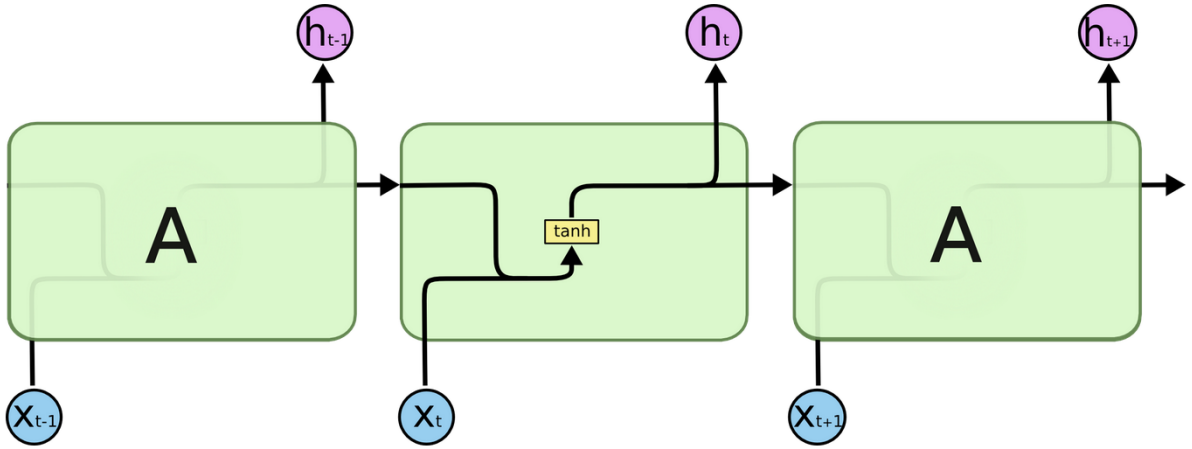
$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}}$$: Chain rule, this product can become really small

See this tutorial for more: https://arxiv.org/pdf/1610.02583.pdf

LSTM Networks

Long Short Term Memory networks

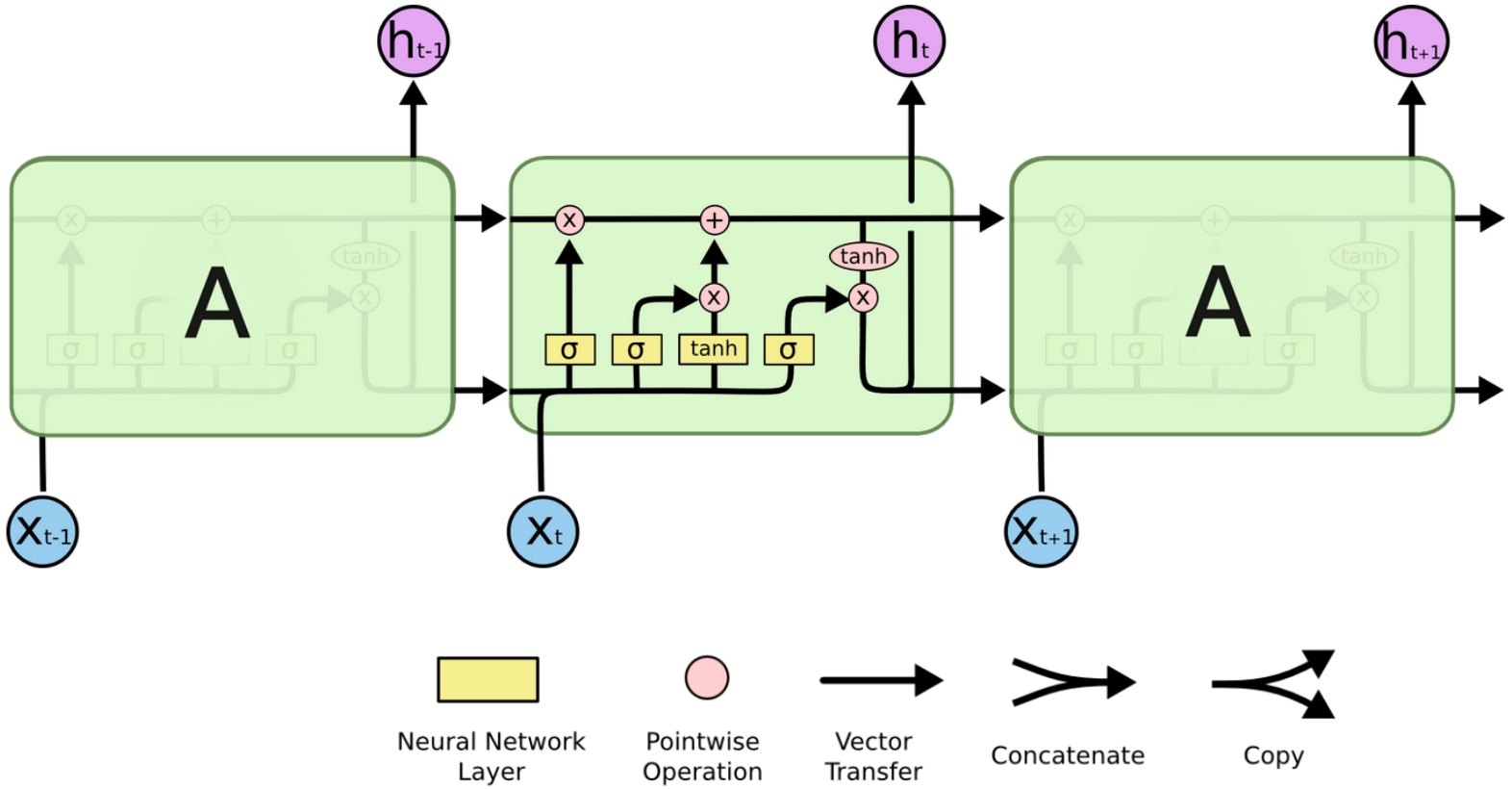
- Explicitly designed to avoid the long term dependency problem
- E.g., RNN with a single tanh layer



$$h_t = \tanh(W[h_{t-1}, x_t] + b)$$

LSTM Networks

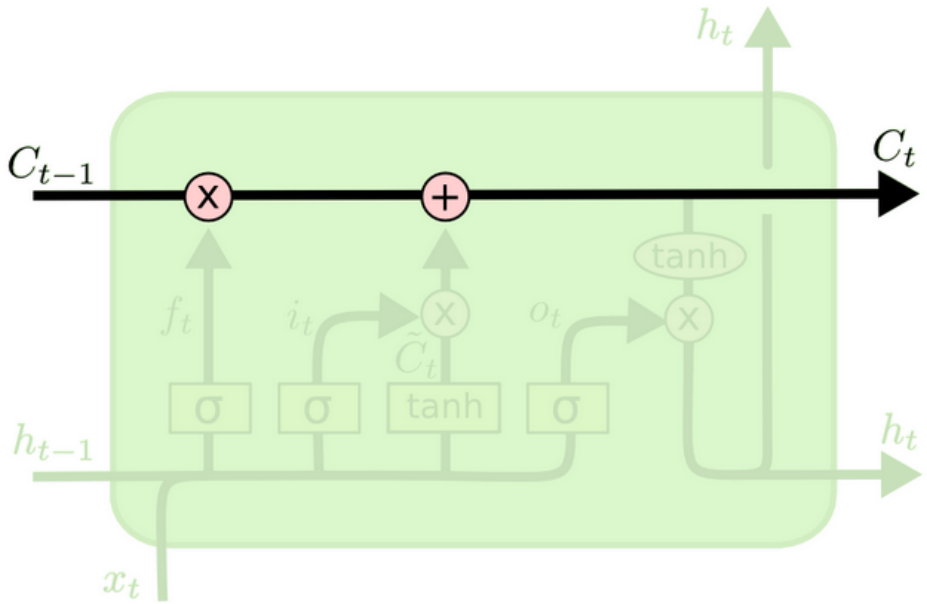
Four neural network layers



LSTM Networks

Key:

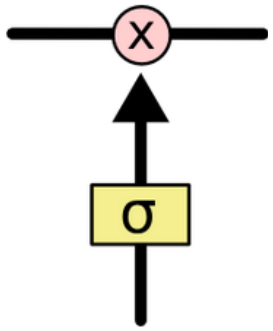
- Cell state: like a conveyor belt, runs straight down the entire chain, information flows along it unchanged



LSTM Networks

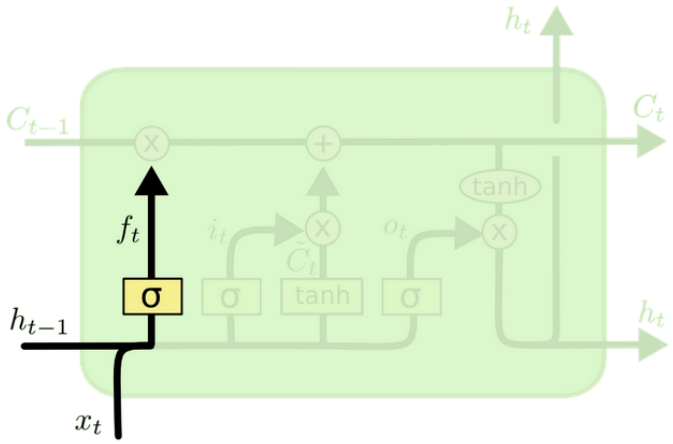
Key:

- Cell state: like a conveyor belt, runs straight down the entire chain, information flows along it unchanged
- Gates: a way to optionally let information through
  - E.g., sigmoid layer and pointwise multiplication information
  - The sigmoid outputs numbers between zero and one, describing how much of each component should be let through: 0 means lets nothing through, 1 means let everything through



LSTM Networks Step-by-Step

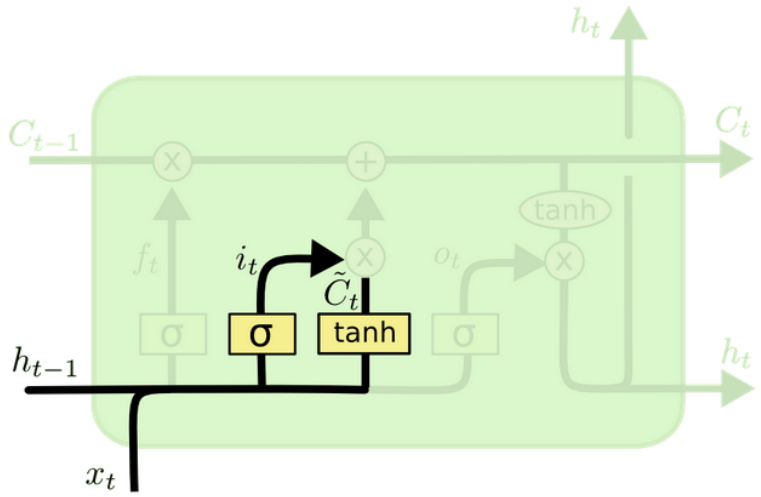
1. Decide what information to **throw away** from the cell state
   - Forget gate layer
   - E.g., in language modeling, the cell state may contain "gender" of the present subject; so when we see a new subject, we may want to "forget" the gender of the old subject



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM Networks Step-by-Step

2. Decide what **new** information to store in the cell state
   - Input gate layer decides which values to update
   - A layer creates a vector of new candidate values that could be added
   - E.g., we'd want to add the gender of the new subject into the cell state to replace the old one we're forgetting

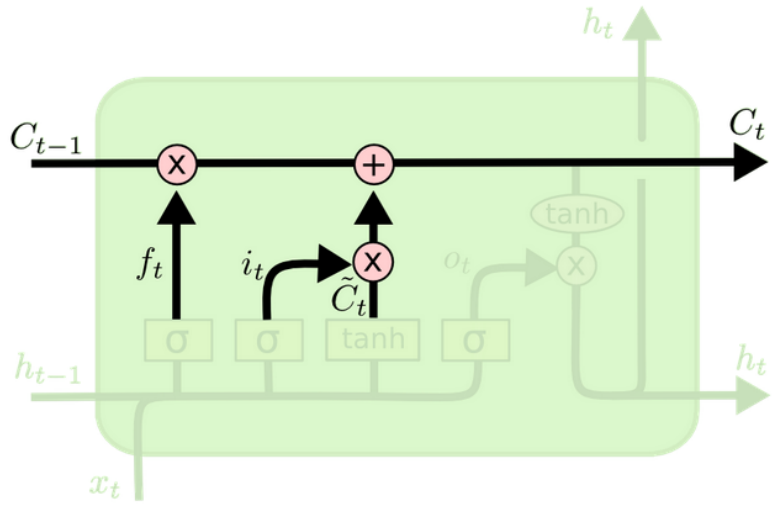


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM Networks Step-by-Step

3. Do the update
   - Multiply old state by f_t (to forget things)
   - Add $i_t * \tilde{C}_t$ (the new candidate values, scaled by how much we want to update each state value)
   - E.g., drop info about old subject's gender, add new info about gender



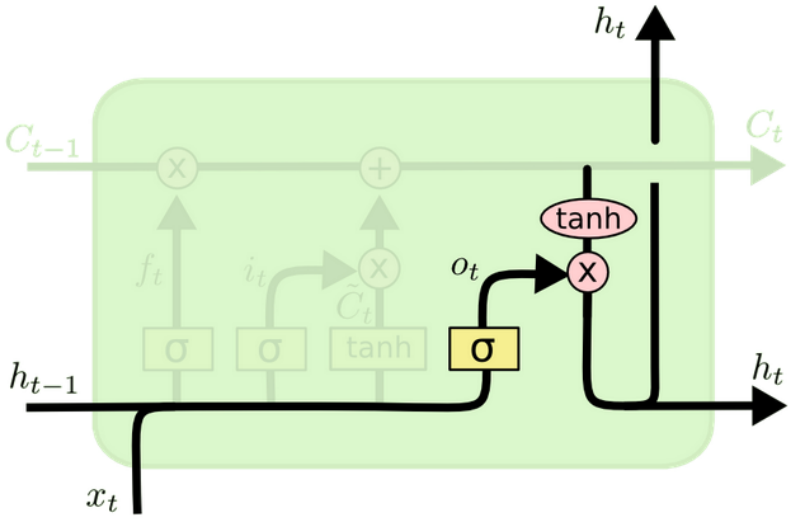
new

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

forget

LSTM Networks Step-by-Step

4. Decide the output based on cell state, but a filtered version
   - A layer decides what parts of the cell state we're outputting
   - Another layer pushes the cell state to be between certain numbers
   - E.g., for the language model, since it just saw a new subject, it might want to output information relevant to a verb such as whether the subject is SG/PL

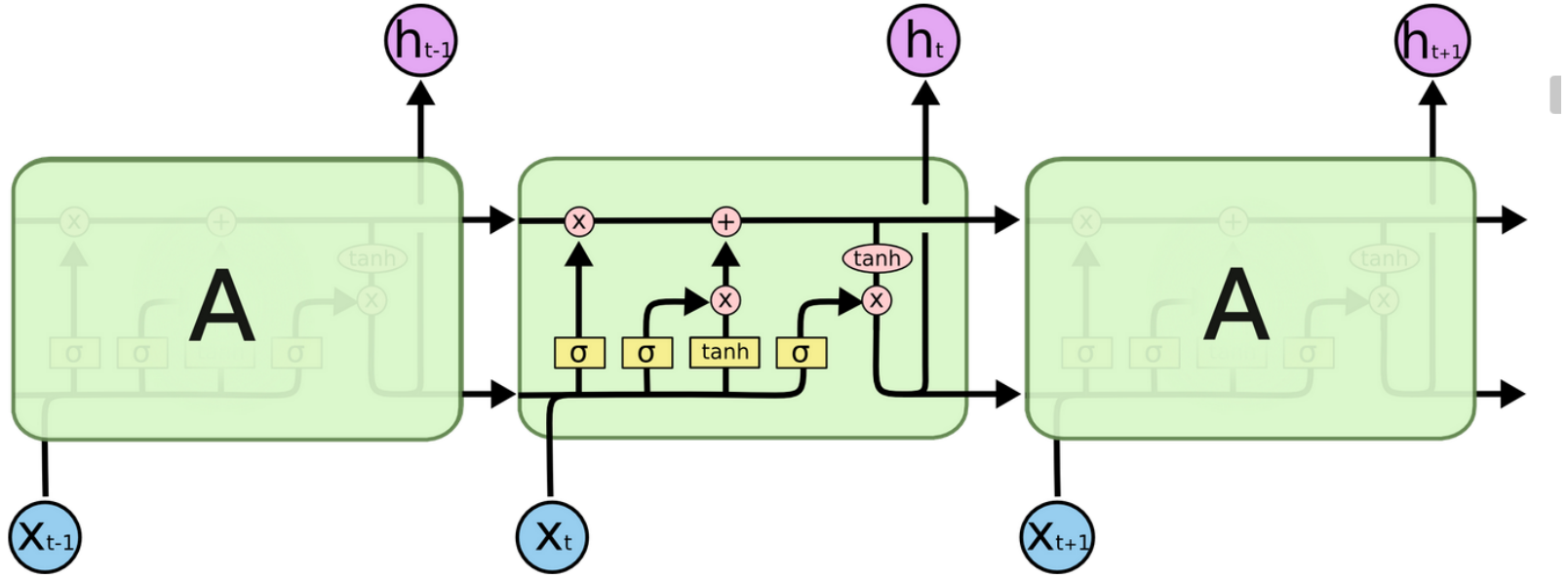


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM Networks

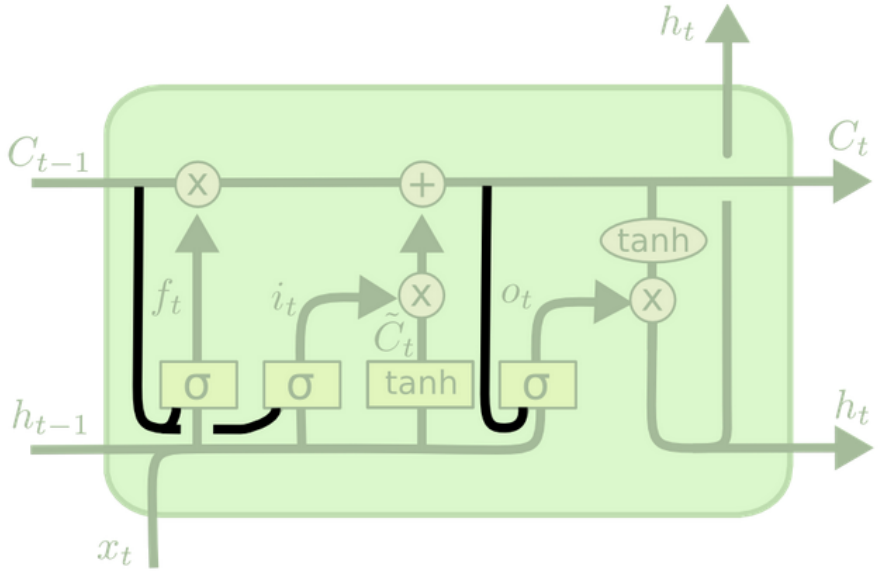
(1) forget, (2) add, (3) update, (4) output



LSTM Networks

A lot of variants:

Add “peephole connections” – letting gate layers look at the cell state (Gers & Schmidhuber, 2000)

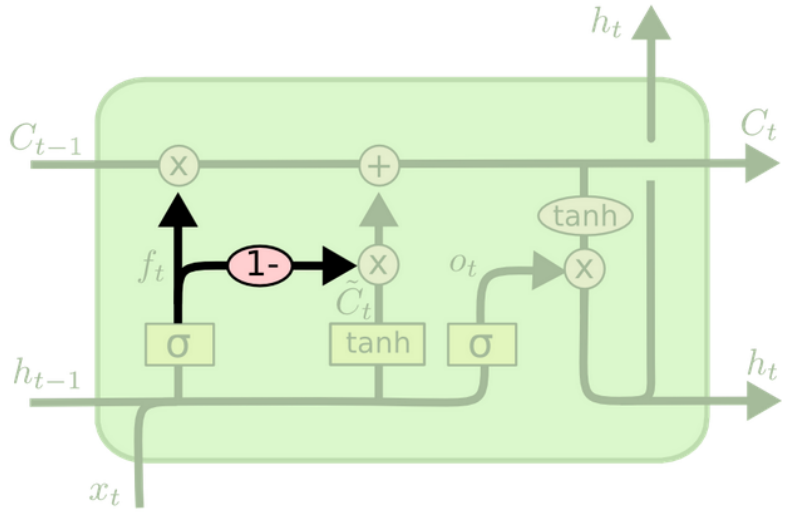


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

LSTM Networks

Use coupled forget and input gates:

- Decide what to forget and what to add jointly e.g., we're only forgetting something when we're going to input something in its place – or we only input new values to th

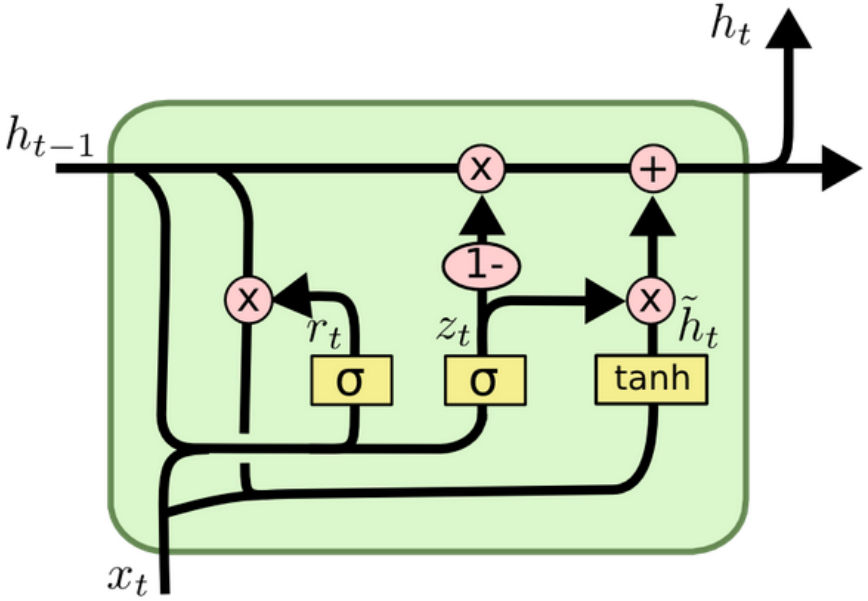


$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

LSTM Networks

Gated Recurrent Unit (GRU)

- Combines forget and input gates into a single “update gate”
- Merges the cell state and hidden state
- Resulting model is simpler than standard LSTM



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Sequence to Sequence Network

From: John Hewitt, Reno Kriz, and Graham Neubig's Neural Machine Translation and Sequence-to-sequence Models: A Tutorial

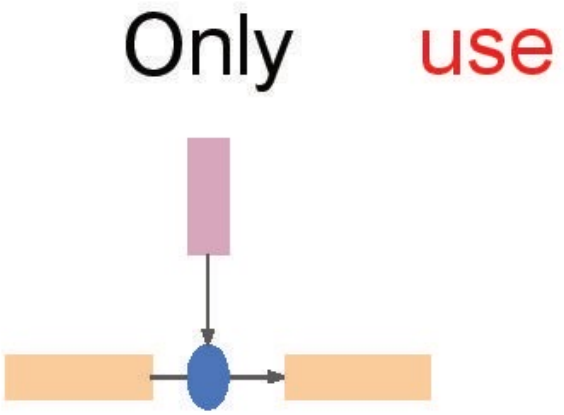
https://arxiv.org/pdf/1703.01619.pdf

RNN for language modeling

Only

The memory vector, or “state” .

RNN for language modeling

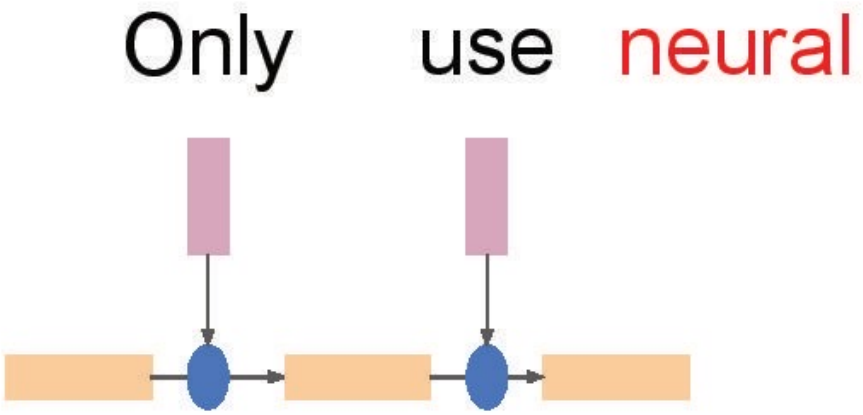


The memory vector, or “state” .

The “word vector” representation of the word.

The RNN function, which combines the word vector and the previous state to create a new state.

RNN for language modeling

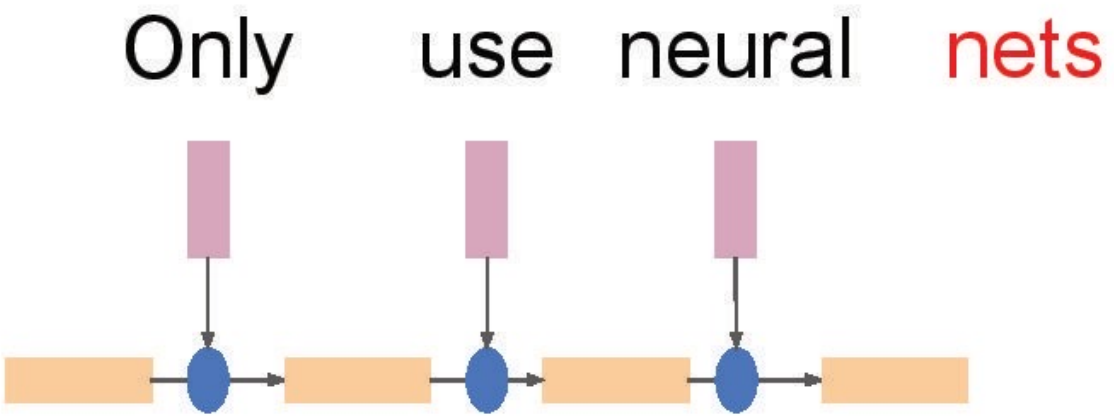


The memory vector, or “state” .

The “word vector” representation of the word.

The RNN function, which combines the word vector and the previous state to create a new state.

RNN for language modeling

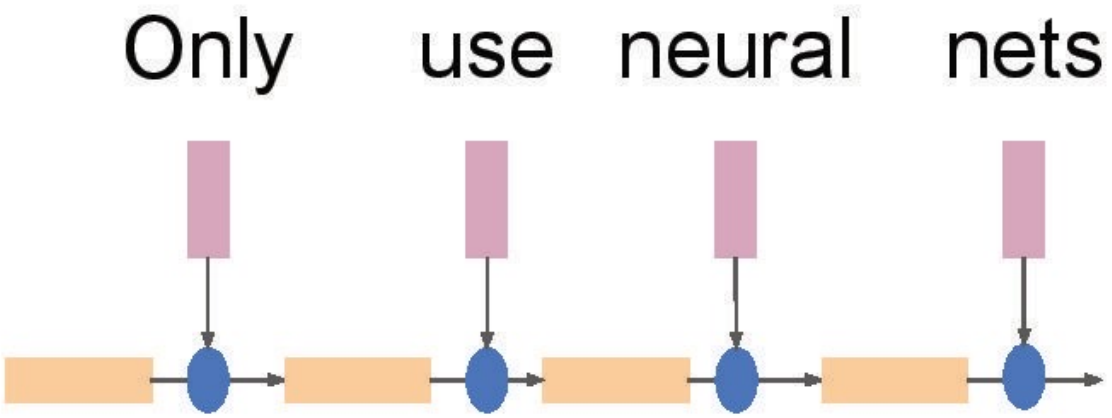


The memory vector, or “state” .

The “word vector” representation of the word.

The RNN function, which combines the word vector and the previous state to create a new state.

RNN for language modeling



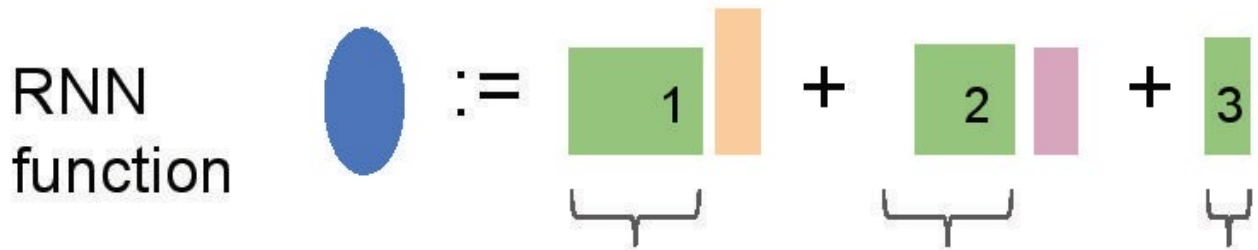
The memory vector, or “state” .

The “word vector” representation of the word.

The RNN function, which combines the word vector and the previous state to create a new state.

How does the RNN function work?

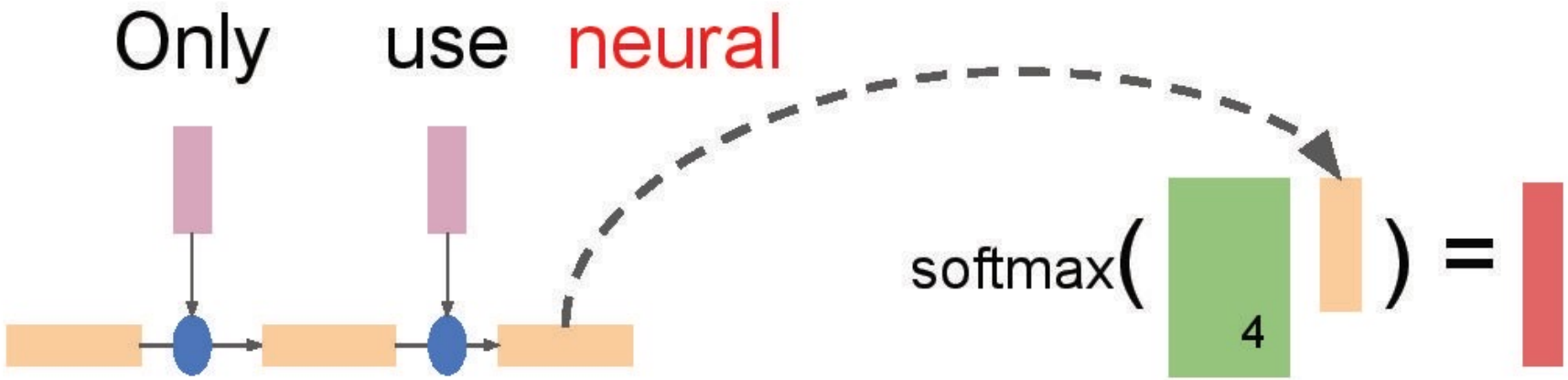
The RNN function takes the current RNN state and a word vector and produces a subsequent RNN state that "encodes" the sentence *so far*.



Learned weights representing how to combine past information (the RNN memory) and current information (the new word vector.)

How does the prediction function work?

We've seen how RNNs "encode" word sequences. But how do they produce probability distributions over a vocabulary?

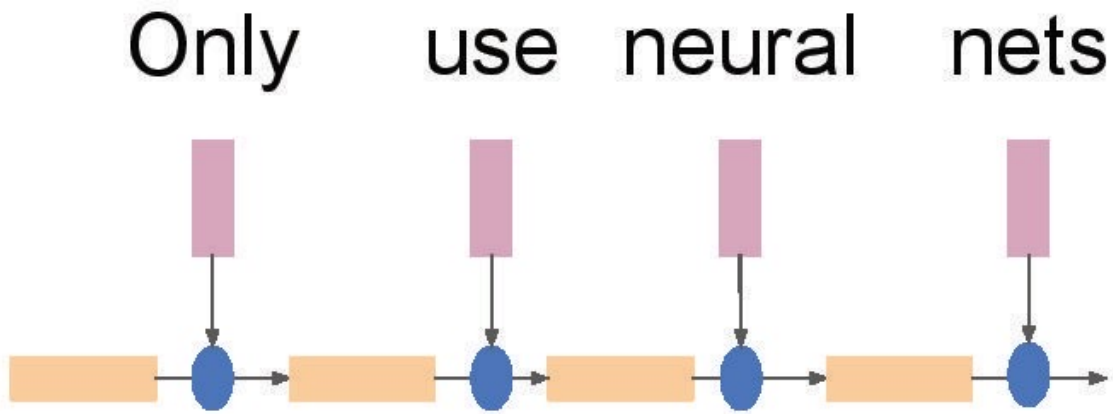


A probability distribution over the vocab, constructed from the RNN memory and 1 last transformation (in green.) The softmax function turns "scores" into a probability distribution.

Want to predict things other than the next word?

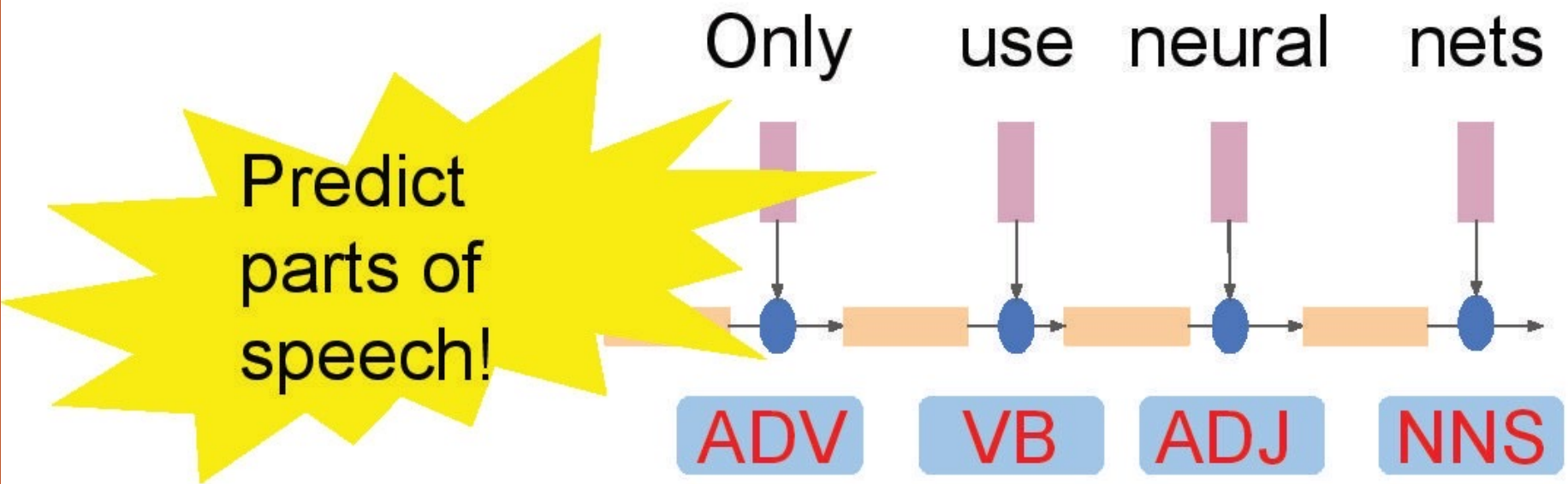
The model architecture (read: “design”) we’ve seen so far is frequently used in tasks other than language modeling, because modeling sequential information is useful in language, apparently.

Here’s our RNN encoder, representing the sentence.



Want to predict things other than the next word?

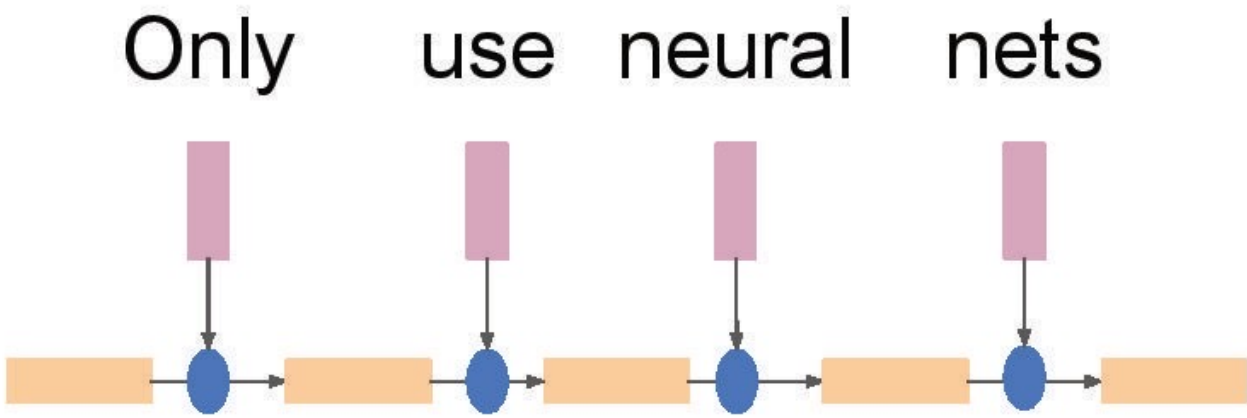
The model architecture (read: “design”) we’ve seen so far is frequently used in tasks other than language modeling, because modeling sequential information is useful in language, apparently.



Seq2Seq

General idea: build a representation

The method of building the representation is called an Encoder and is frequently an RNN.

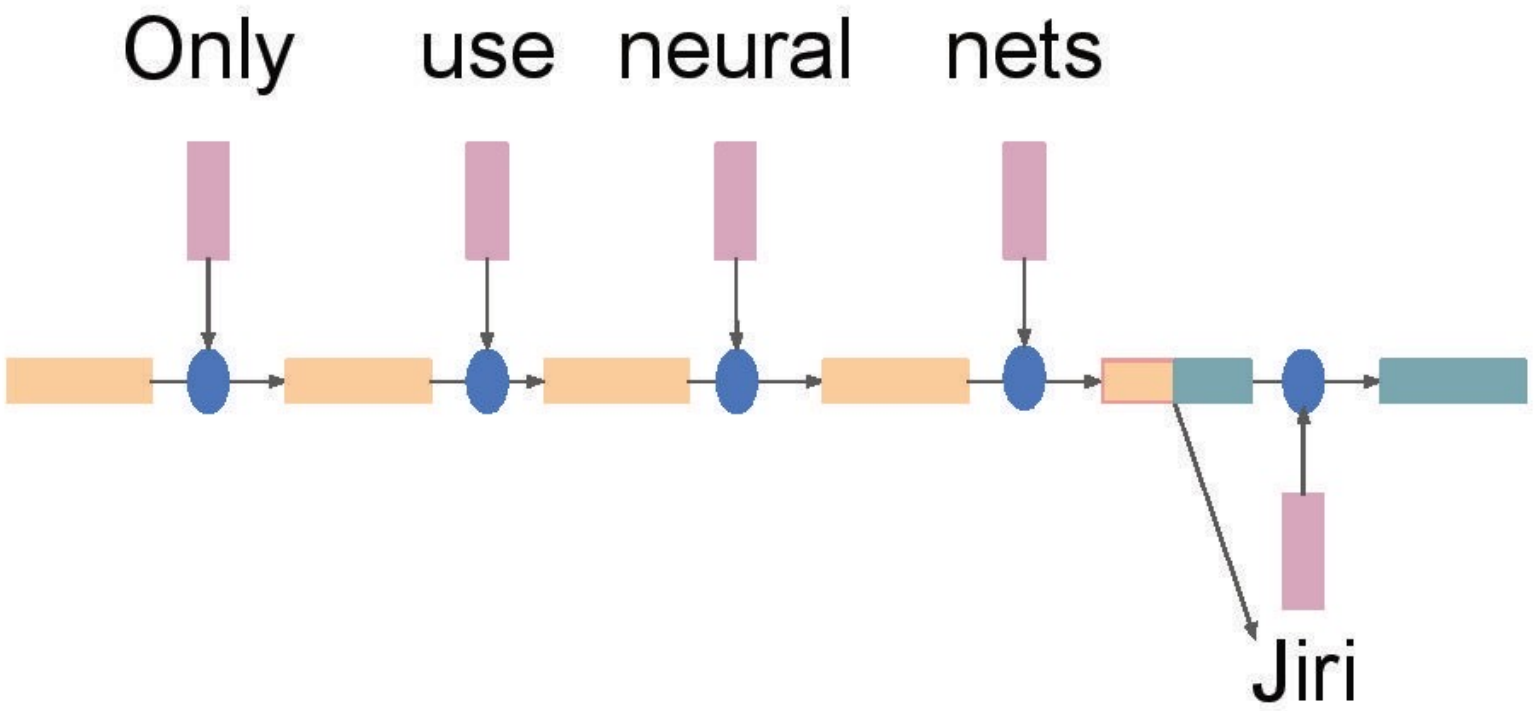


Each memory vector in the encoder attempts to represent the sentence so far, but mostly represents the word most recently input.

Seq2Seq

General idea: generate the output one token at a time

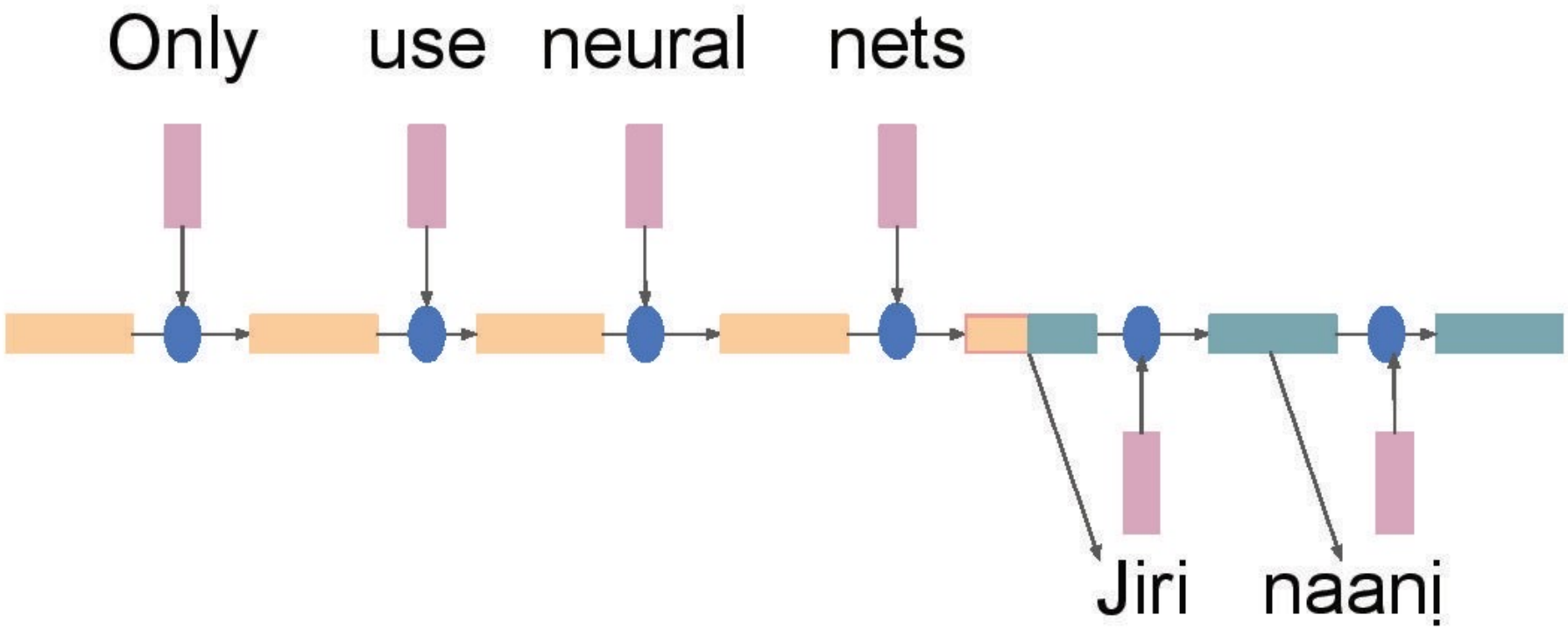
The model that takes the encoded representation and generates the output is called the Decoder, and, errrr, is also generally an RNN.



Seq2Seq

General idea: generate the output one token at a time

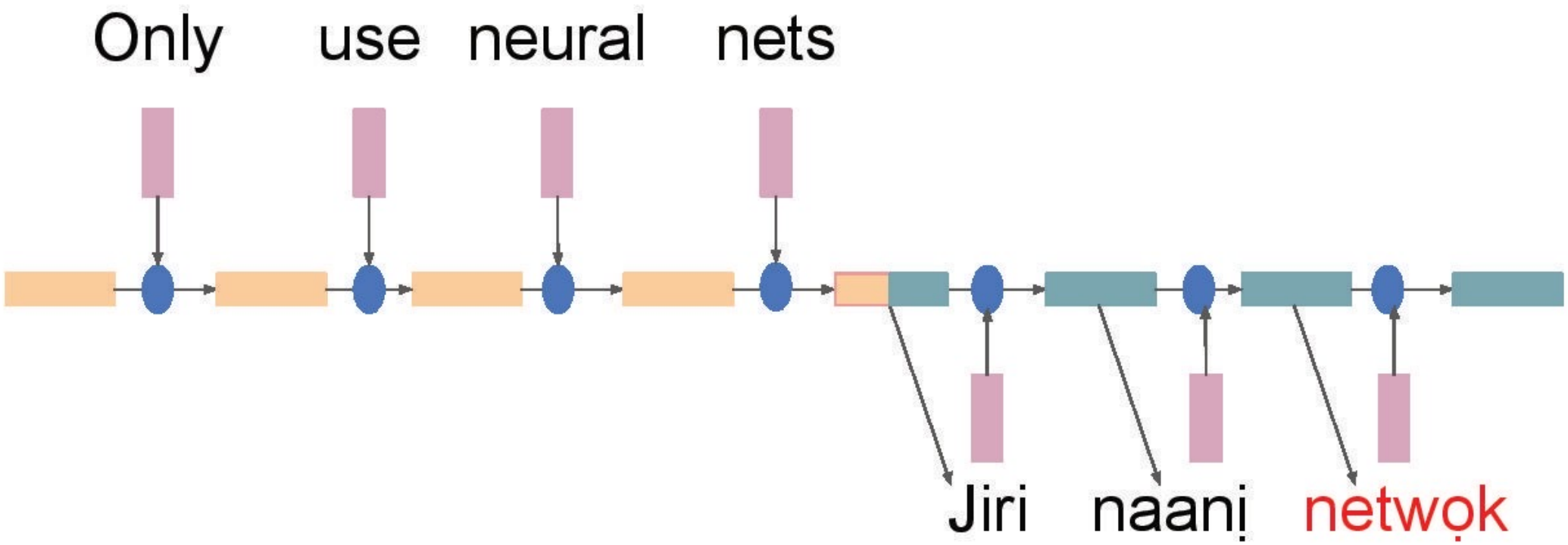
The model that takes the encoded representation and generates the output is called the Decoder, and, errrr, is also generally an RNN.



Seq2Seq

General idea: generate the output one token at a time

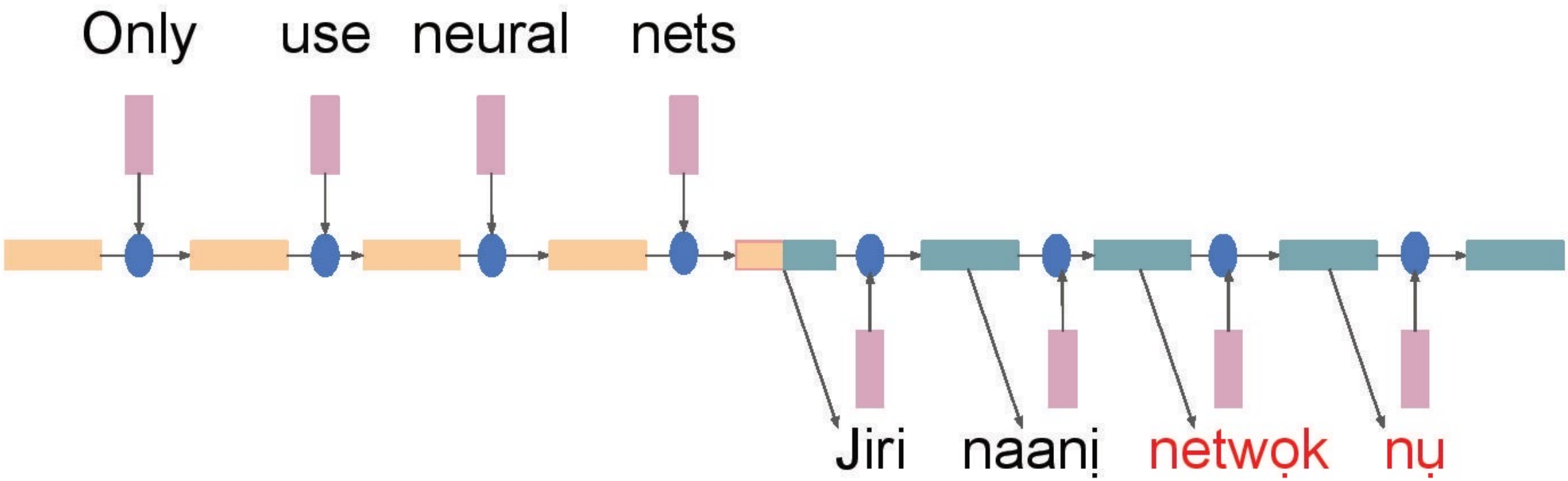
The model that takes the encoded representation and generates the output is called the Decoder, and, errrr, is also generally an RNN.



Seq2Seq

General idea: generate the output one token at a time

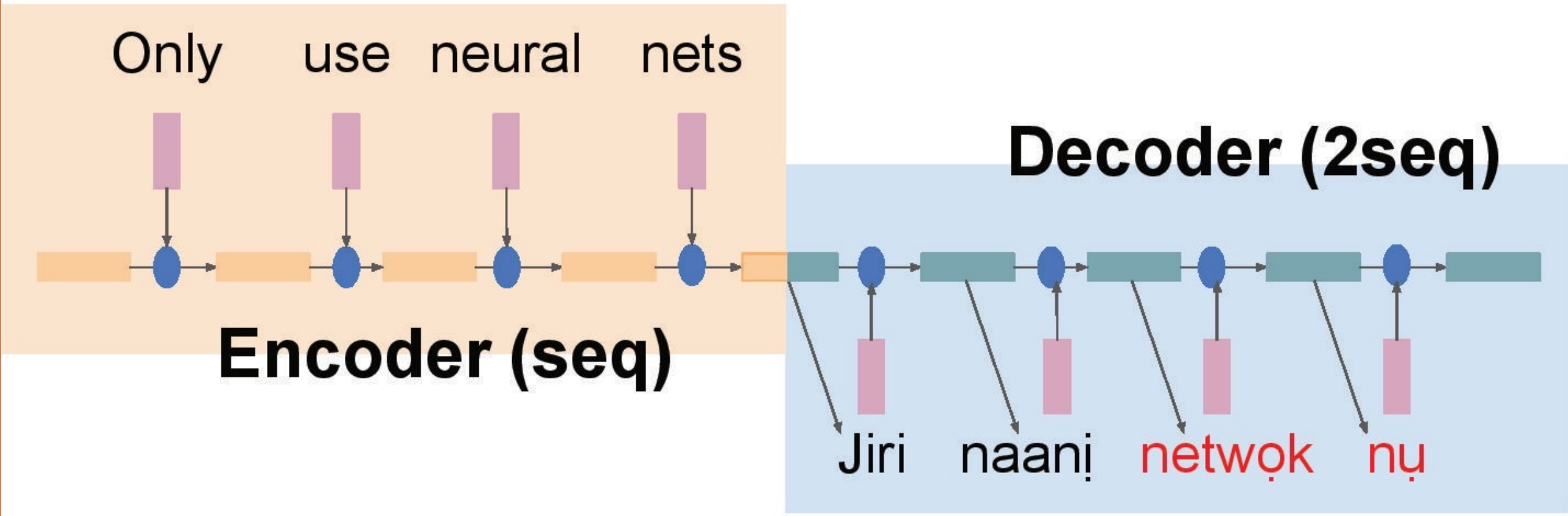
The model that takes the encoded representation and generates the output is called the Decoder, and, errrr, is also generally an RNN.



Seq2Seq

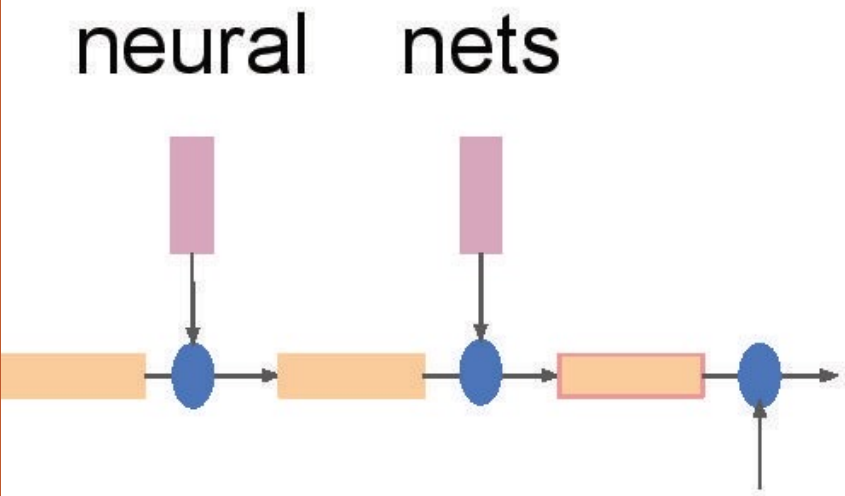
General idea: generate the output one token at a time

The model that takes the encoded representation and generates the output is called the Decoder, and, errrr, is also generally an RNN.



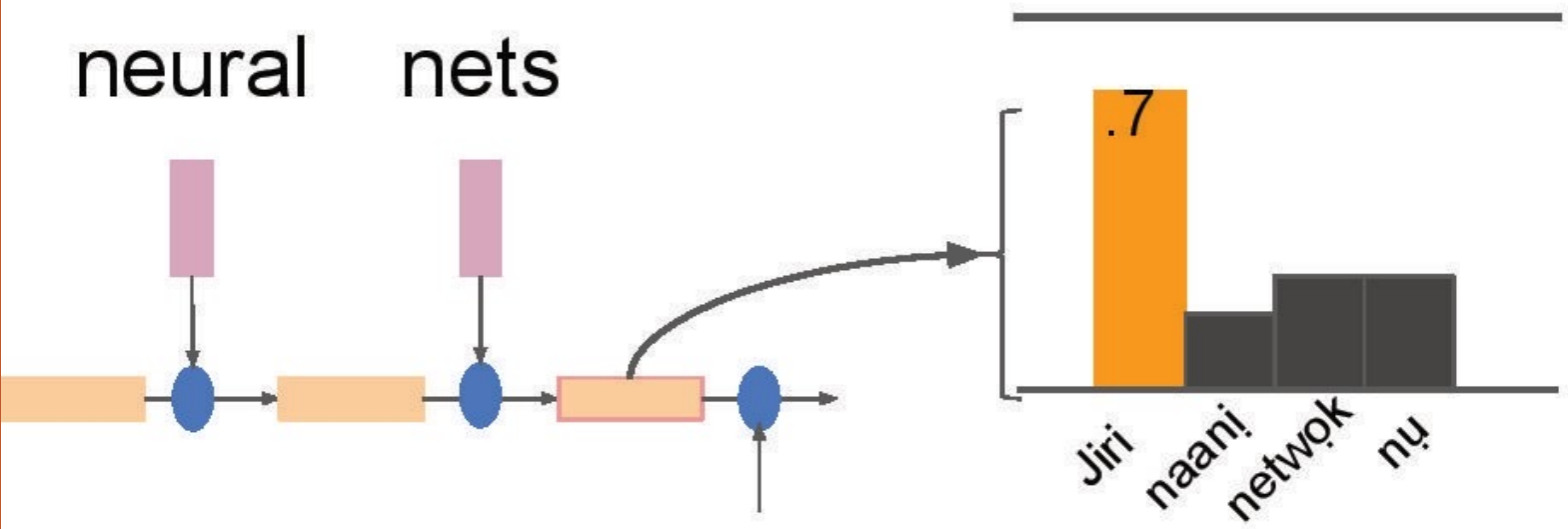
How is it trained?

In practice, training for a single sentence is done by “forcing” the decoder to generate gold sequences, and penalizing it for assigning the sequence a low probability. Losses for each token in the sequence are summed. Then, the summed loss is used to take a step in the right direction in all model parameters (including word embeddings!) (*stochastic gradient descent.)*



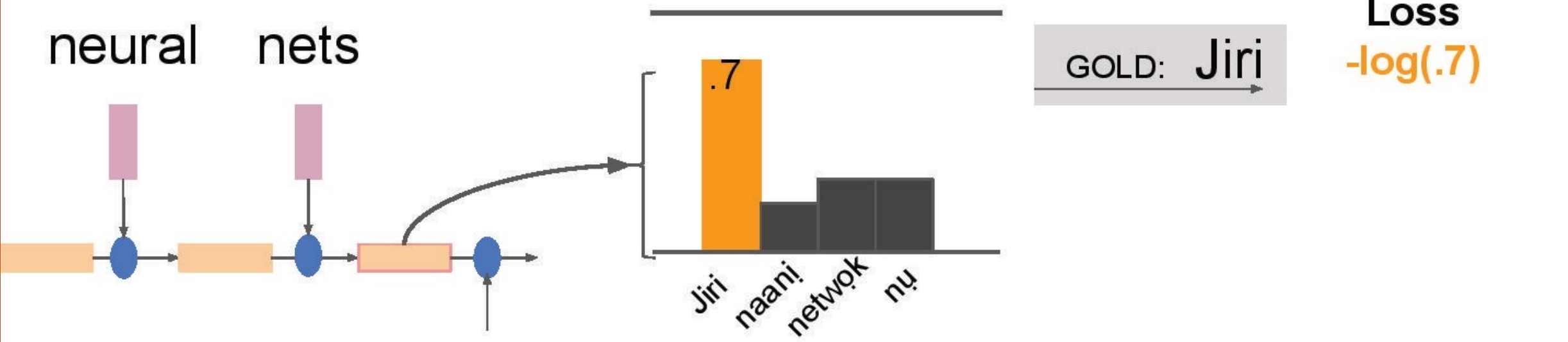
How is it trained?

In practice, training for a single sentence is done by “forcing” the decoder to generate gold sequences, and penalizing it for assigning the sequence a low probability. Losses for each token in the sequence are summed. Then, the summed loss is used to take a step in the right direction in all model parameters (including word embeddings!) (*stochastic gradient descent.)*



How is it formalized? How is it trained?

In practice, training for a single sentence is done by “forcing” the decoder to generate gold sequences, and penalizing it for assigning the sequence a low probability. Losses for each token in the sequence are summed. Then, the summed loss is used to take a step in the right direction in all model parameters (including word embeddings!) (*stochastic gradient descent.)*



Sentence-level training

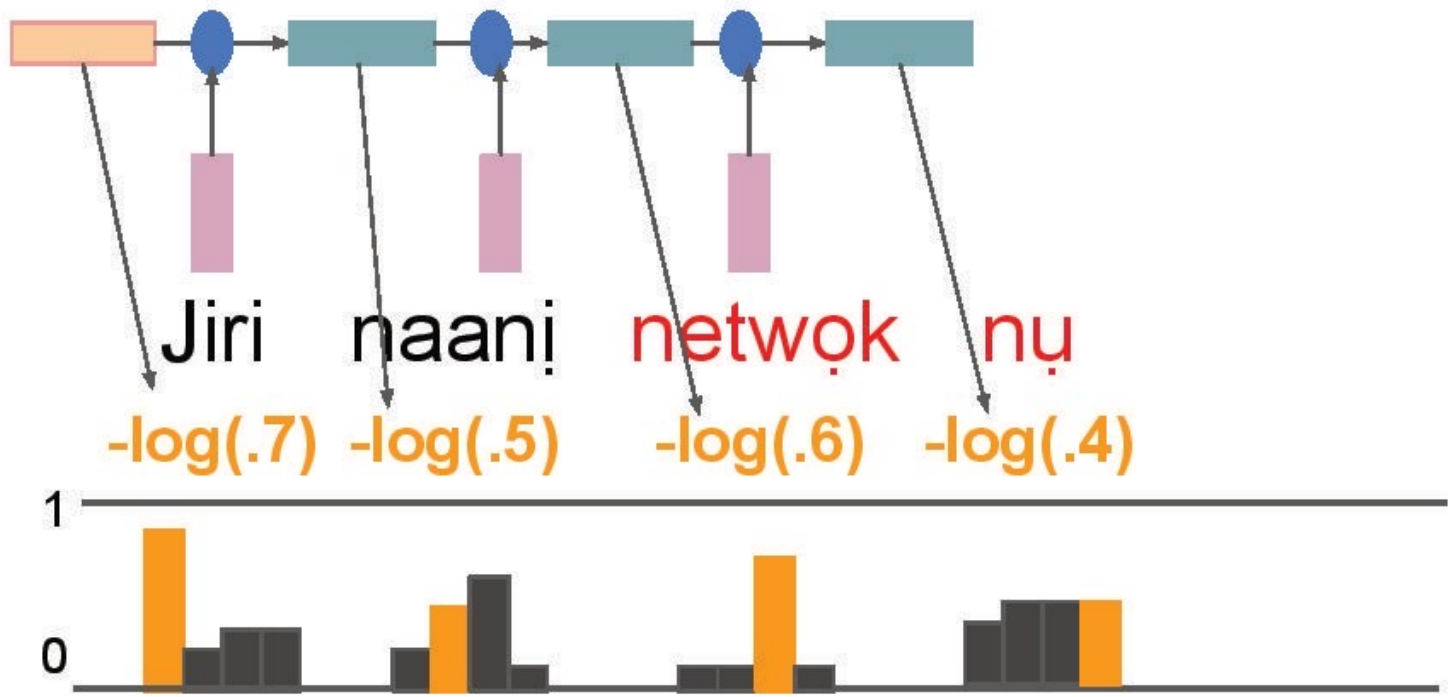
Almost all such networks are trained using **cross-entropy loss**. **At each step, the network produces a probability distribution over possible next tokens. This distribution is penalized from being different from the true distribution (e.g., a probability of 1 on the actual next token.)**

Sentence-level training

Almost all such networks are trained using **cross-entropy loss**. **At each step, the network produces a probability distribution over possible next tokens. This distribution is penalized from being different from the true distribution (e.g., a probability of 1 on the actual next token.)**

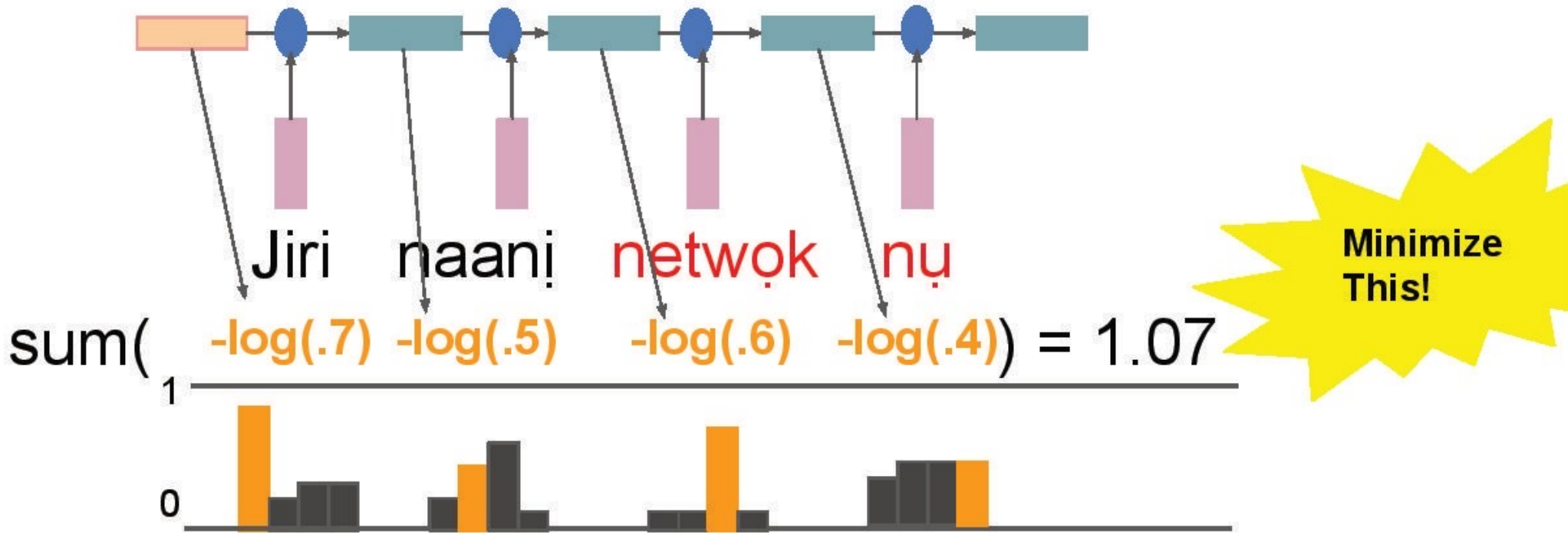
Sentence-level training

Almost all such networks are trained using **cross-entropy loss**. **At each step, the network produces a probability distribution over possible next tokens. This distribution is penalized from being different from the true distribution (e.g., a probability of 1 on the actual next token.)**



Sentence-level training

Almost all such networks are trained using **cross-entropy loss**. **At each step, the network produces a probability distribution over possible next tokens. This distribution is penalized from being different from the true distribution (e.g., a probability of 1 on the actual next token.)**



How is it formalized?

Let h_t be the RNN hidden state at timestep t:

How is it formalized?

Let h_t be the RNN hidden state at timestep t:

Let x_t be the input vector at timestep t: 

How is it formalized?

Let h_t be the RNN hidden state at timestep t:

Let x_t be the input vector at timestep t: 

The RNN equation posits 2 matrices and 1 vector as parameters:

$\mathbf{W}^{hx}$ integrates input vector information. 

How is it formalized?

Let h_t be the RNN hidden state at timestep t:

Let x_t be the input vector at timestep t: 

The RNN equation posits 2 matrices and 1 vector as parameters:

- $\mathbf{W}^{hx}$ integrates input vector information. 
- $\mathbf{W}^{hh}$ integrates information from the previous timestep. 

How is it formalized?

Let h_t be the RNN hidden state at timestep t:

Let x_t be the input vector at timestep t: 

The RNN equation posits 2 matrices and 1 vector as parameters:

$\mathbf{W}^{hx}$ integrates input vector information. 

$\mathbf{W}^{hh}$ integrates information from the previous timestep. 

$\mathbf{b}^{h}$ is a bias term. (What function does this perform?)

How is it formalized?

Let h_t be the RNN hidden state at timestep t:

Let x_t be the input vector at timestep t: 

The RNN equation posits 2 matrices and 1 vector as parameters:

- $\mathbf{W^{hx}}$ integrates input vector information. 
- $\mathbf{W^{hh}}$ integrates information from the previous timestep. 
- $\mathbf{b^h}$ is a bias term. (What function does this perform?)

The RNN equation is: $\mathbf{h_t = \tanh(W^{hx}x_t + W^{hh}h_{t-1} + b^h)}$

How is it formalized?

For prediction, we take the current hidden state, and use it as *features* in what is more or less a linear regression.

How is it formalized?

For prediction, we take the current hidden state, and use it as *features* in what is more or less a linear regression.

Let d_t be our decision (e.g., word, POS tag) at timestep t. Let D be the set of all possible decisions. Let $\mathbf{s}_{t-1}$ be the most recent *decoder* hidden state.

How is it formalized?

For prediction, we take the current hidden state, and use it as *features* in what is more or less a linear regression.

Let d_t be our decision (e.g., word, POS tag) at timestep t. Let D be the set of all possible decisions. Let $\mathbf{s}_{t-1}$ be the most recent *decoder* hidden state.

$$\mathbf{d}_t = \textit{argmax}_{d' \in D}\ \mathbf{p}(\ \mathbf{d'} \mid \mathbf{x}_{1:n}, \mathbf{d}_{1:t-1})$$

How is it formalized?

For prediction, we take the current hidden state, and use it as *features* in what is more or less a linear regression.

Let d_t be our decision (e.g., word, POS tag) at timestep t. Let D be the set of all possible decisions. Let $\mathbf{s}_{t-1}$ be the most recent *decoder* hidden state.

$$\mathbf{d}_t = \textit{argmax}_{d' \in D}\ \mathbf{p}(\ \mathbf{d'} \mid \mathbf{x}_{1:n}, \mathbf{d}_{1:t-1})$$

$$\mathbf{p}(\ * \mid \mathbf{x}_{1:n}, \mathbf{d}_{1:t-1}) = \mathbf{softmax}_D(\mathbf{W}^{Dh}\mathbf{s}_{t-1} + \mathbf{b}^D)$$

How is it formalized?

For prediction, we take the current hidden state, and use it as *features* in what is more or less a linear regression.

Let d_t be our decision (e.g., word, POS tag) at timestep t. Let D be the set of all possible decisions. Let $\mathbf{s}_{t-1}$ be the most recent *decoder* hidden state.

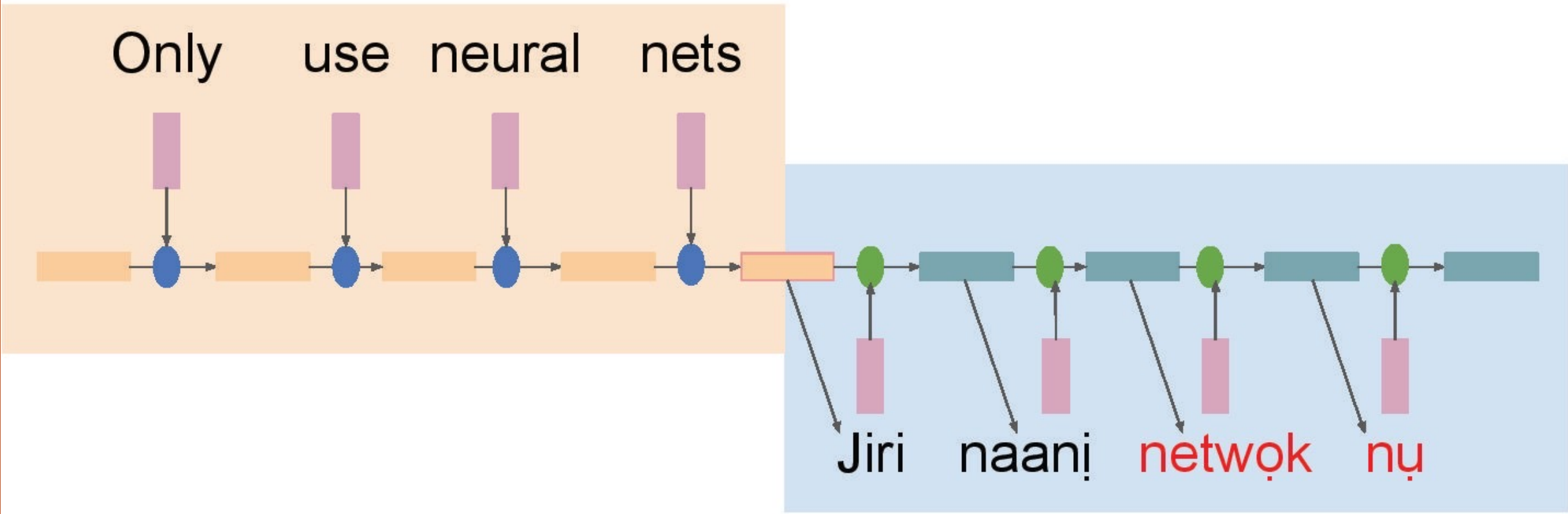
$$\mathbf{d}_t = \textit{argmax}_{d' \in D}\ \mathbf{p}(\ \mathbf{d'} \mid \mathbf{x}_{1:n}, \mathbf{d}_{1:t-1})$$

$$\mathbf{p}(\ * \mid \mathbf{x}_{1:n}, \mathbf{d}_{1:t-1}) = \mathbf{softmax}_D(\mathbf{W}^{Dh}\mathbf{s}_{t-1} + \mathbf{b}^D)$$

Note that $\mathbf{W}^{hD}\mathbf{s}_{t-1} + \mathbf{b}^D$ produces a *vector of scores.* The **softmax** function normalizes scores to a probability distribution by exponentiating each dimension, and normalizing by the sum. For some choice k of K, $\mathbf{p(k)} = \mathbf{e}^{\text{score(k)}} / \sum_{k' \in K} \mathbf{e}^{\textit{score(k')}}$

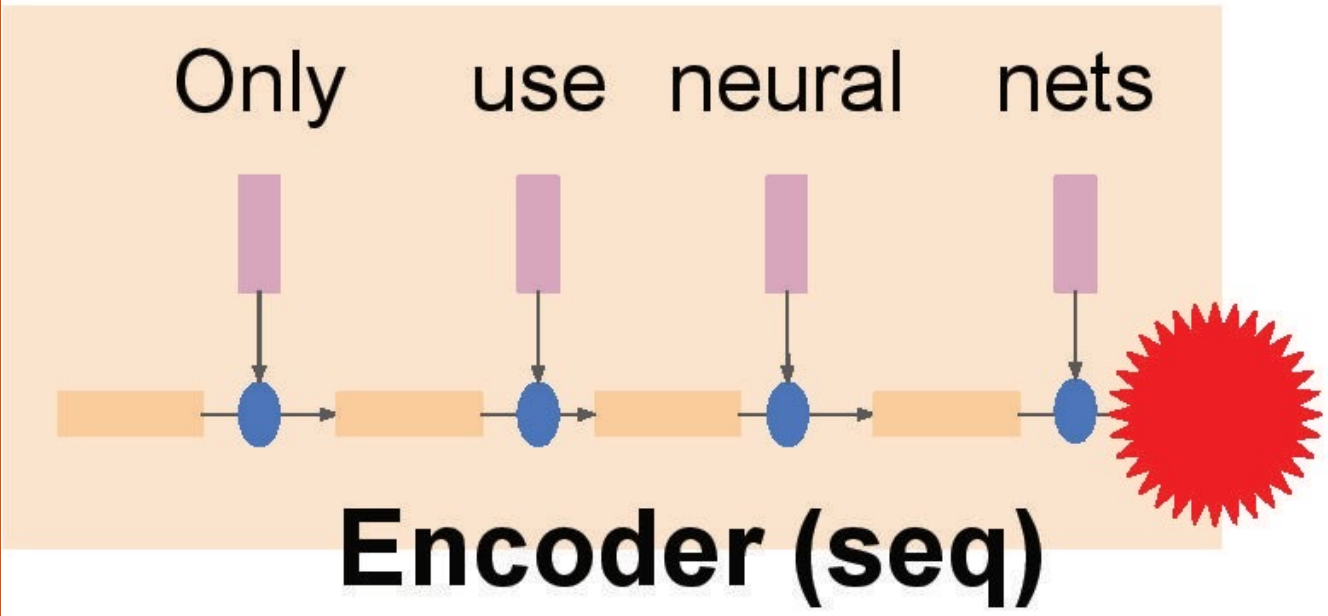
The information bottleneck and latent structure

Given the diagram below, what problem do you foresee when translating progressively longer sentences?



The information bottleneck and latent structure

We are trying to encode *variable-length* structure (e.g., variable-length sentences) in a fixed-length memory (e.g., only the 300 dimensions of your hidden state.)

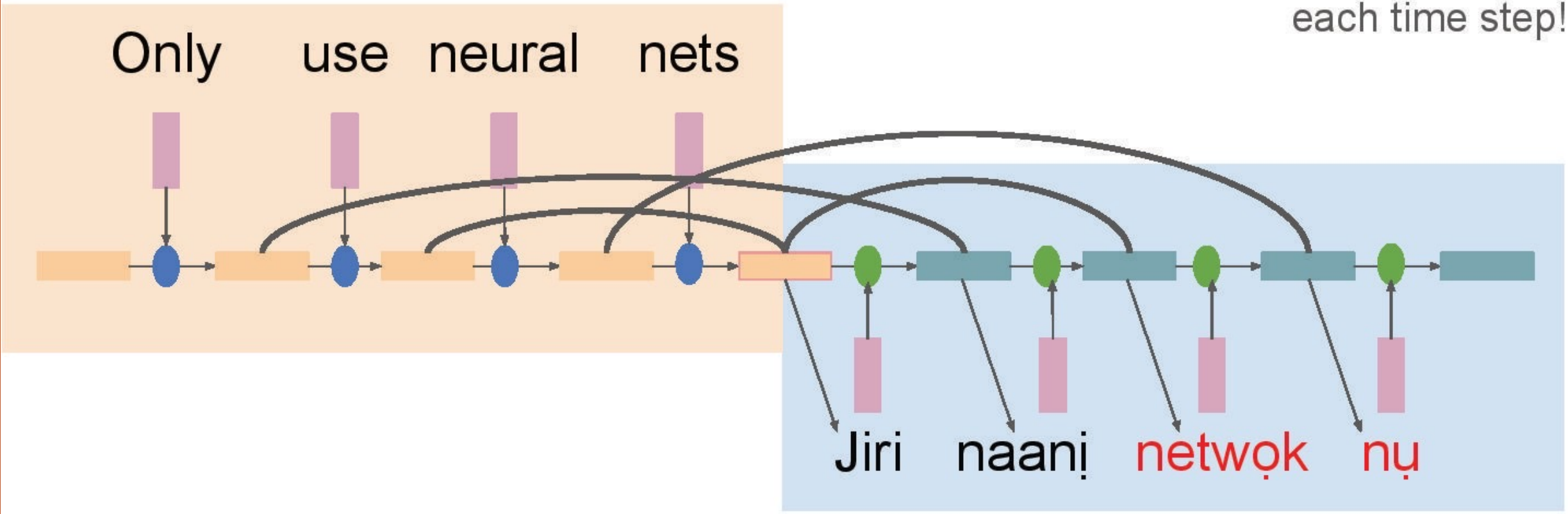


The last encoder hidden state is the bottleneck -- all information in the source sentence must pass through it to get to the decoder.

Finding a solution to this problem was the final advance that made neural MT competitive with previous approaches.

The information bottleneck and latent structure

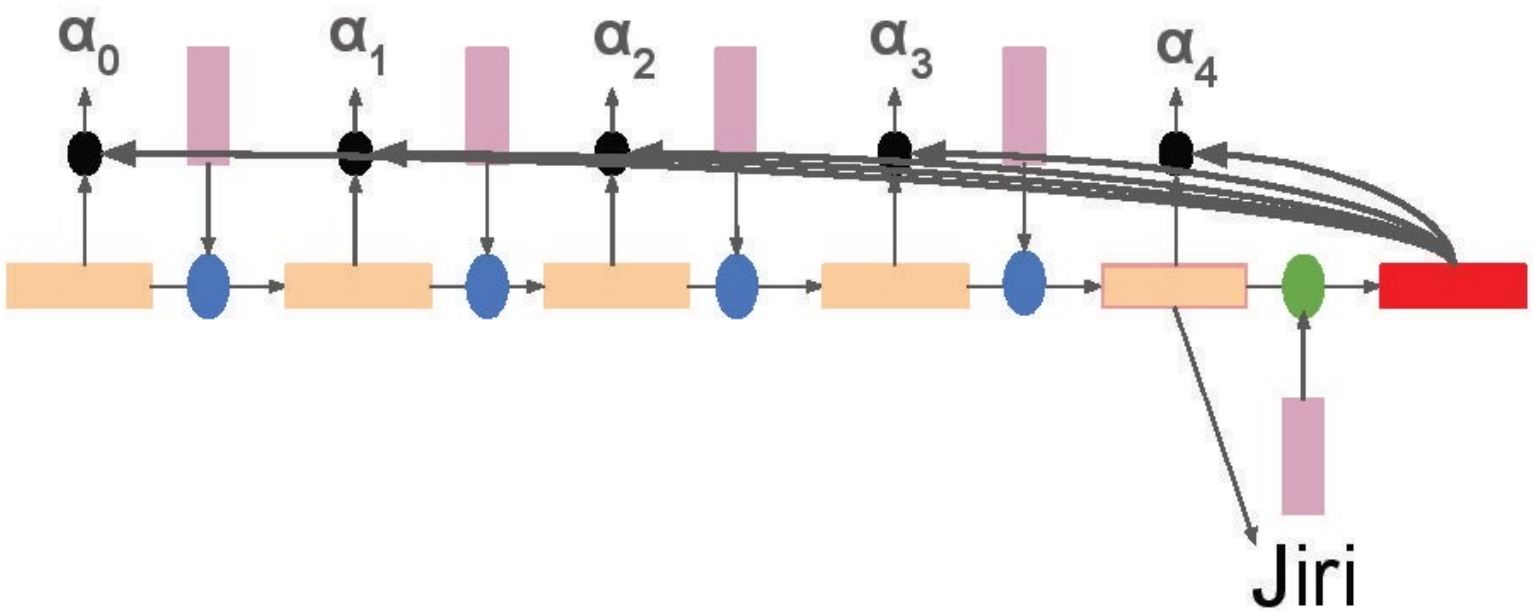
The key insight is related to the word alignment **(jointly learn and translate)** . We allow the decoder to look at *any* encoder state, and let it learn which are important at each time step!



Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

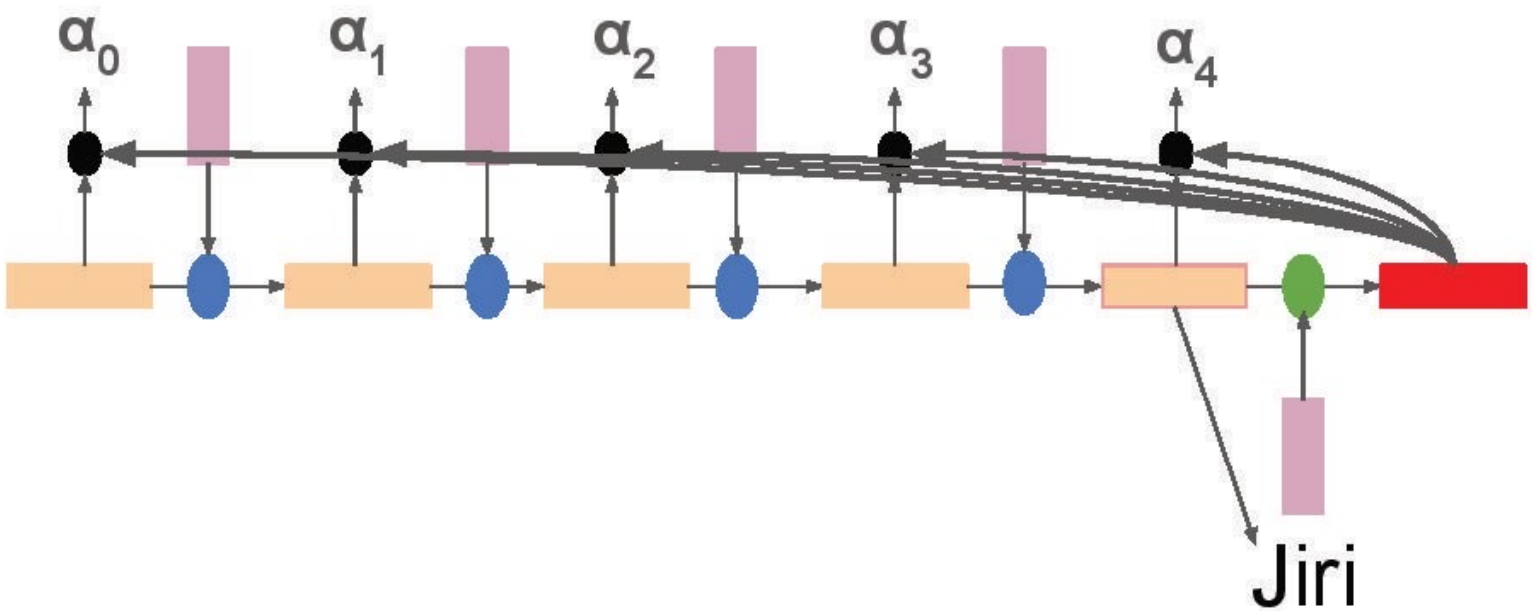
Step 1: Take the decoder state, and compute an *affinity* α_i with all encoder states.



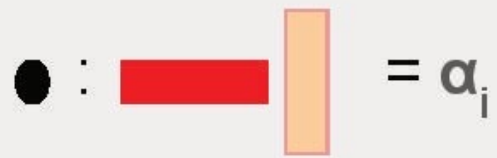
Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 1: Take the decoder state, and compute an *affinity* α_i with all encoder states.



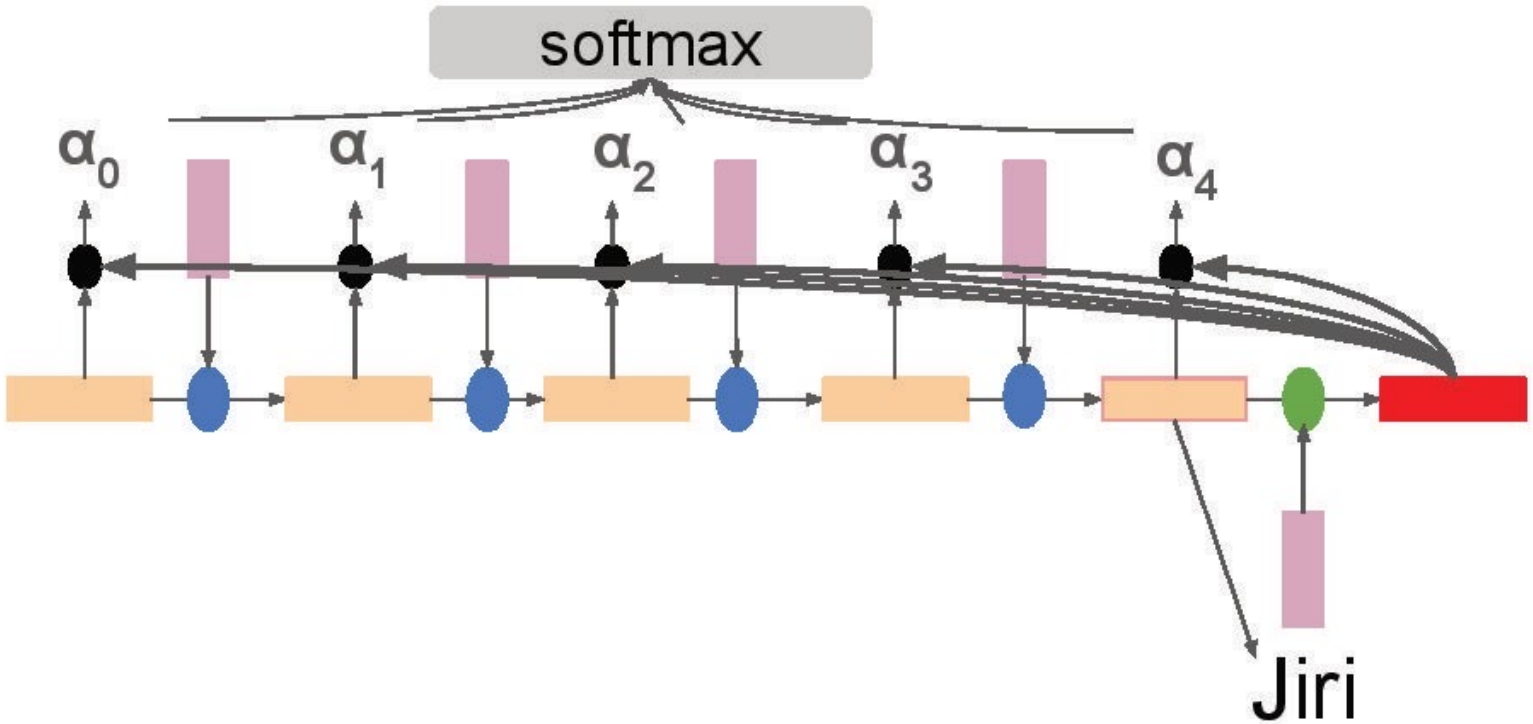
The affinity function, ● , is a dot product, or something similar.



Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

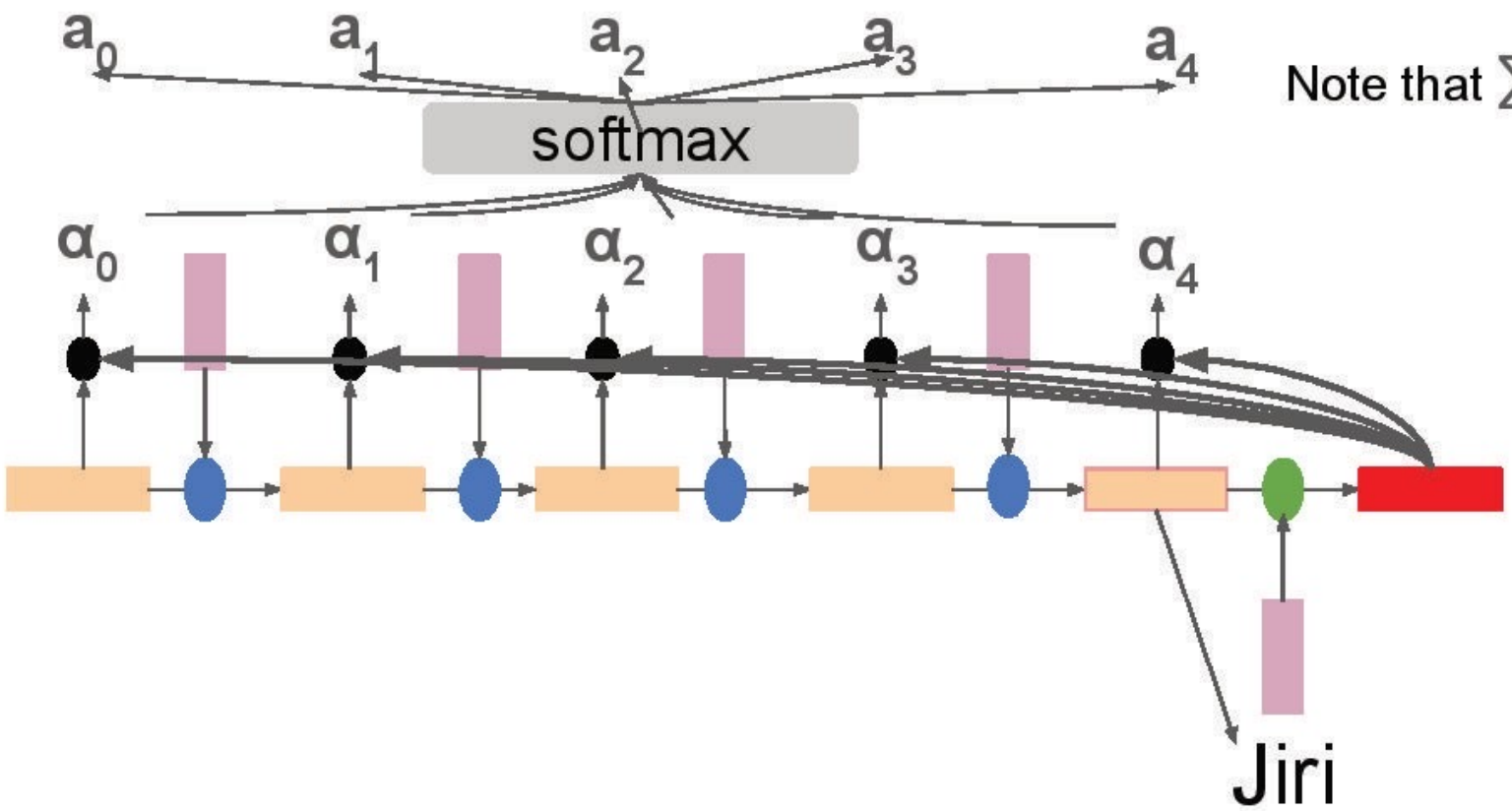
Step 2: Normalize the scores to sum to 1 by the softmax function.



Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 2: Normalize the scores to sum to 1 by the softmax function.

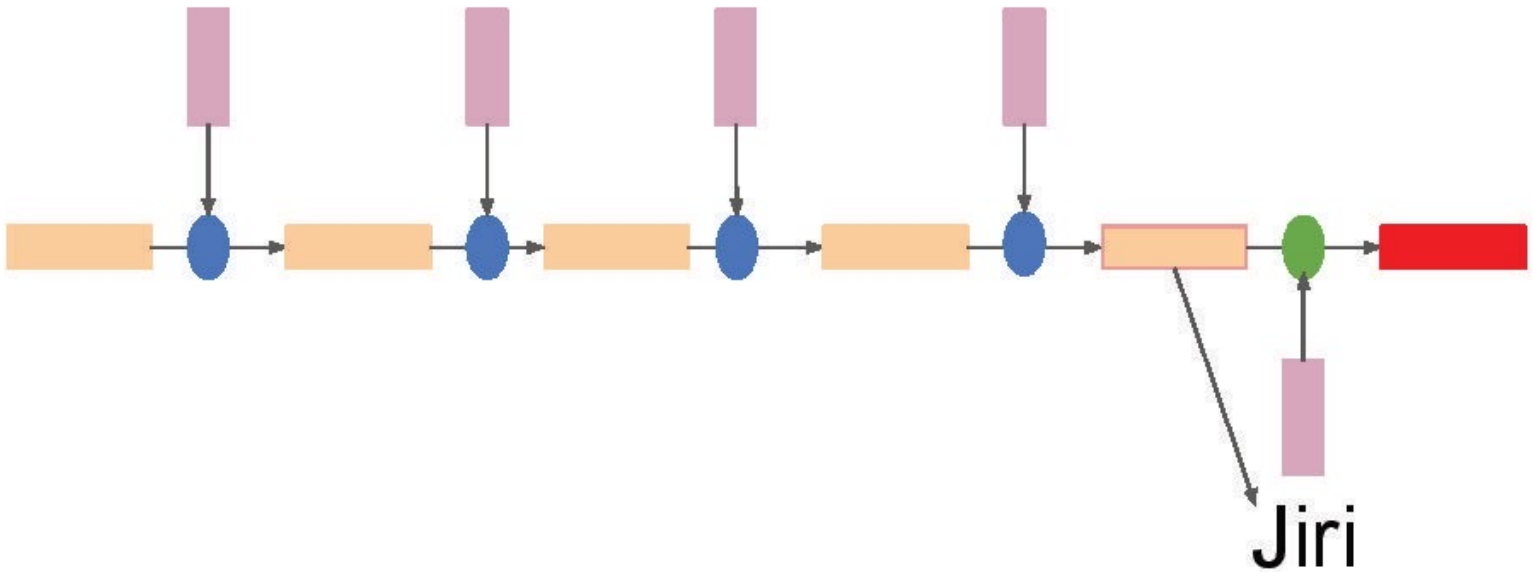


Note that $\sum_{i=1,2,3,4} a_i = 1$

Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 3: Average the encoder states, weighted by the **a** distribution.



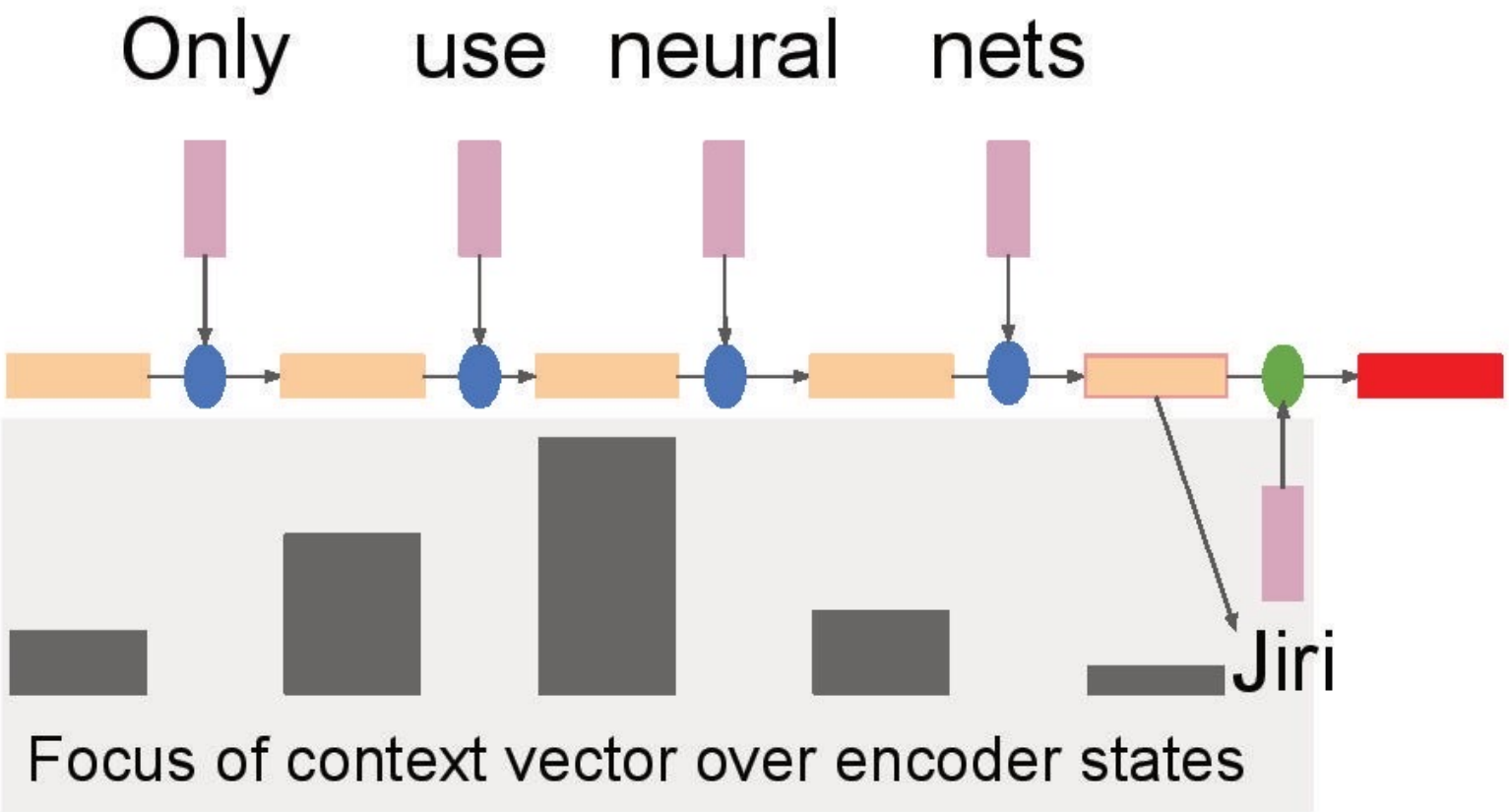
This weighted average

Is called the **context vector.**

Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 3: Average the encoder states, weighted by the **a** distribution.

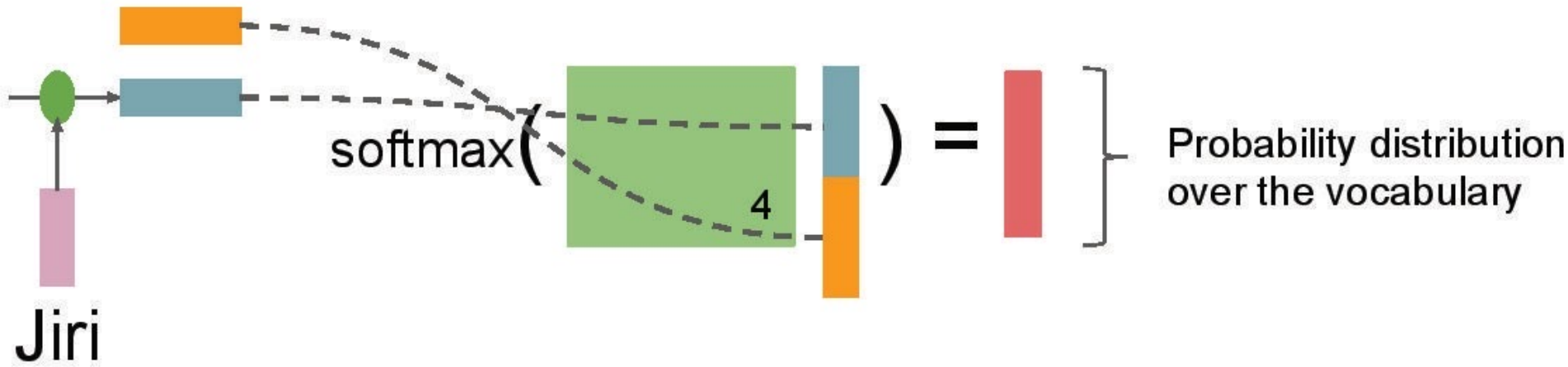


In this example, since “Jiri” means “use”, the attention will focus on the vectors around “use”.

Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 4: Use the context vector at prediction, concatenating it to the decoder state.



This vector has the current decoder information, , but also a focused summary of the encoder, .

Attention Formalization

Attention computes the **affinity between the decoder state and all encoder states**.There are many affinity computation methods, but they're all like a **dot product**.

Let there are *n* encoder states. The affinity between encoder state *i* and the decoder state is $\boldsymbol{\alpha}_i$. The encoder states are $\mathbf{h}_{1:n}$, and the decoder state is $\mathbf{s}_{t-1}$.

Let $\boldsymbol{\alpha}_i = f(\mathbf{h}_i, \mathbf{s}_{t-1}) = \mathbf{h}_i^\top \mathbf{s}_{t-1}$

Let weights $\mathbf{a} = \text{softmax}(\boldsymbol{\alpha})$.

Let the context $\mathbf{c} = \sum_{i=1:n} \mathbf{h}_i \mathbf{a}_i$. *(Note that this is a weighted average.)*

Attention Formalization

Attention is used at prediction as extra information in the final prediction.

Reminder, we let the context $\mathbf{c} = \sum_{i=1:n} \mathbf{h}_i \mathbf{a}_i$. *(Weighted average of encoder states.)*

Attention Formalization

Attention is used at prediction as extra information in the final prediction.

Reminder, we let the context $\mathbf{c} = \sum_{i=1:n} \mathbf{h}_i \mathbf{a}_i$. *(Weighted average of encoder states.)*

Let the notation [**s**;**c**] mean the concatenation of vectors **s** and **c**.

$$\mathbf{d}_t = \textit{argmax}_{d' \in D}\ p(\ d' \mid \mathbf{x}_{1:n}, \mathbf{d}_{1:t-1})$$

(same as before, without attention)

Attention Formalization

Glossing over this slide is totally reasonable. Feel free to check your phone, ping your Bitcoin investment, see if your The Boring Company® (Not a) Flamethrower has shipped.

Attention is used at prediction as extra information in the final prediction.

Reminder, we let the context $\mathbf{c} = \sum_{i=1:n} \mathbf{h}_i \mathbf{a}_i$. *(Weighted average of encoder states.)*

Let the notation [**s**;**c**] mean the concatenation of vectors **s** and **c**.

$$\mathbf{d}_t = \textit{argmax}_{d' \in D}\ p(\ d' \mid \mathbf{x}_{1:n}, \mathbf{d}_{1:t-1})$$

$$p(\ * \mid \mathbf{x}_{1:n}, \mathbf{d}_{1:t-1}) = \mathbf{softmax}_D(\mathbf{W}^{D(2h)}[\mathbf{s}_{t-1};\mathbf{c}]+\mathbf{b}^D)$$

So, the only difference is that the final prediction uses the context vector concatenated to the decoder state to make the prediction.

Empirical considerations

There are a lot of "hyperparameter" choices that can greatly affect the quality of your model. **In short, take parameters from papers/tutorials, and grid search (try many combinations of parameters) around them.**

RNN variants: LSTMs have a different (much better) recurrent equation.

Hidden state sizes: larger: more memory! Requires more data.

Embedding sizes: more representation power! Requires more data.

Learning rate: the step size you take in learning your parameters! Start this "large", and cut it in half when your training stops improving development set performance.

Empirical considerations

There are a lot of “hyperparameter” choices that can greatly affect the quality of your model. **In short, take parameters from papers/tutorials, and grid search (try many combinations of parameters) around them.**

Regularization: “dropout” prevents overfitting by making each node in your hidden state unavailable for an observation with a given probability. Try some values around .2 to .3.

Batch size: The number of observations to group together before performing a parameter update step. Larger batches: less fine-grained training, many more observations per minute, especially on GPU.

CASE STUDY:

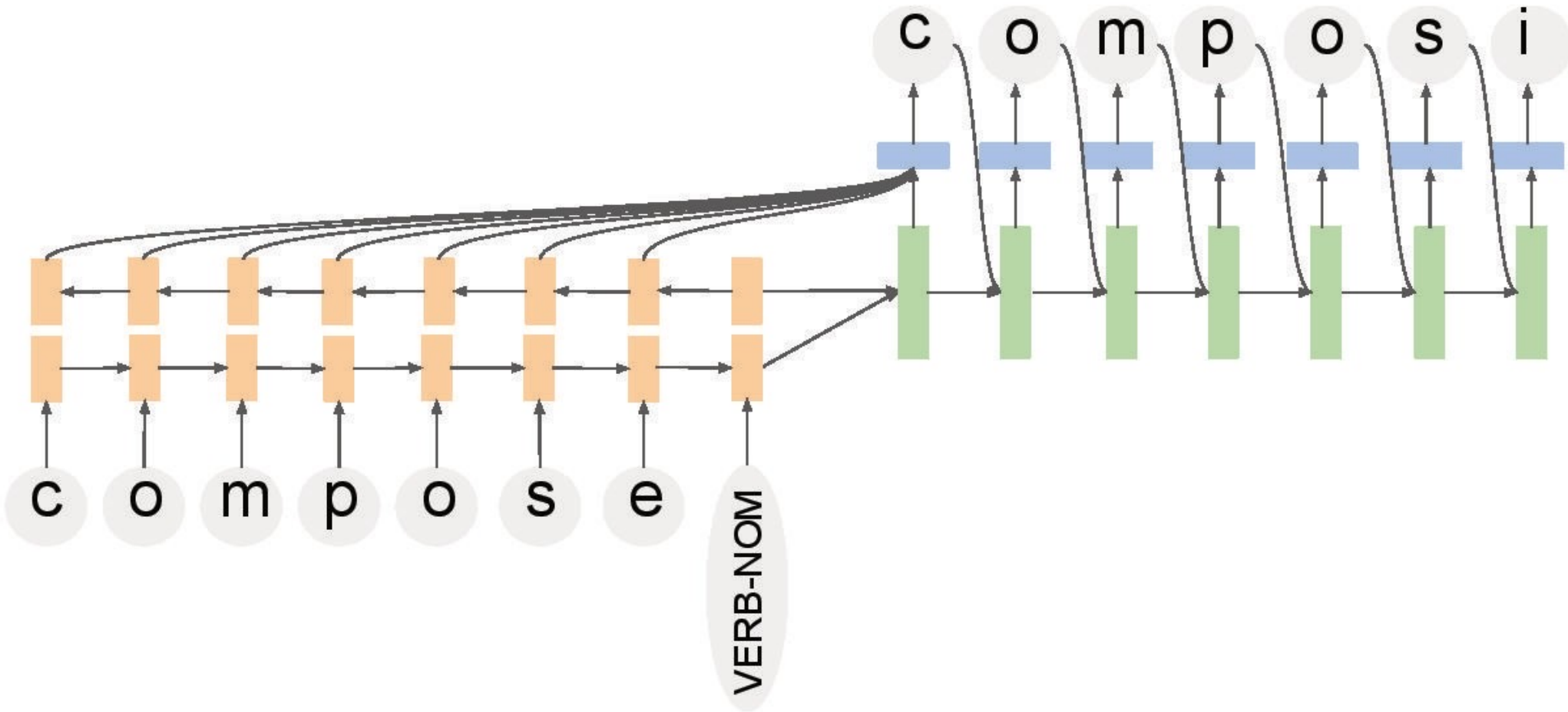
Derivational morphology

- Process of generating new words from existing words
  - Changes semantic meaning
  - Often a new part-of-speech

employ	V -> N, Agent	employer
employ	V -> N, Passive	employee
employ	V -> N, Result	employment
employ	V -> Adj, Potential	employable
employable	V -> Adj -> N, Stative	employability

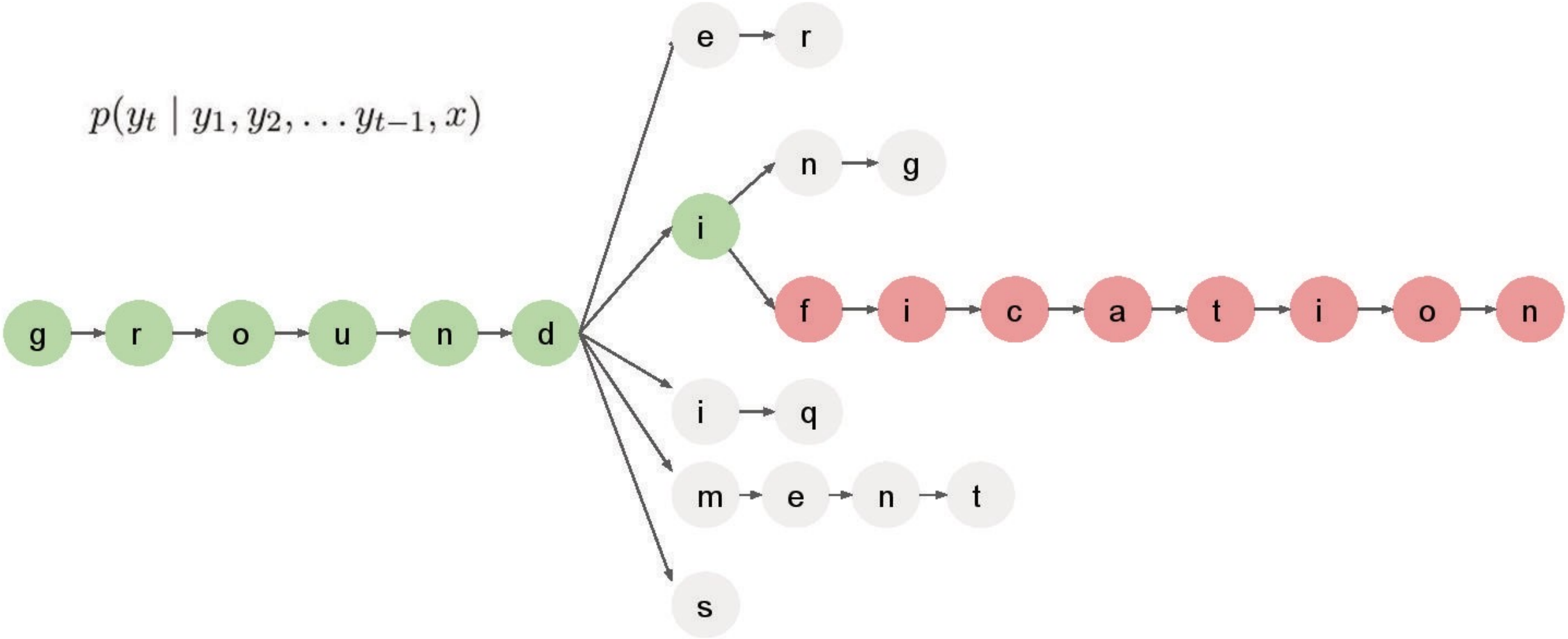
Encoder (seq) **Decoder (2seq)**

Derivational morphology



Derivational morphology: search

$$p(y_t \mid y_1, y_2, \ldots y_{t-1}, x)$$



Reference Sheet

The memory vector, or "state" . Color denotes whether encoder or decoder.

The "word vector" representation of the word.

The RNN function, which combines the word vector and the previous state to create a new state.

A learned parameter matrix

W^{hx} integrates input vector information.

W^{hh} integrates information from the previous timestep.

b^h is a bias term.

d_t is our decision at timestep t.

The RNN equation is:

$$h_t = \tanh(W^{hx}x_t + W^{hh}h_{t-1} + b^h)$$

$$d_t = \textit{argmax}_{d' \in D}\ p(\ d' \mid x_{1:n}, d_{1:t-1})$$

$$p(\ * \mid x_{1:n}, d_{1:t-1}) =$$

$$\text{softmax}_D(W^{Dh}s_{t-1} + b^D)$$