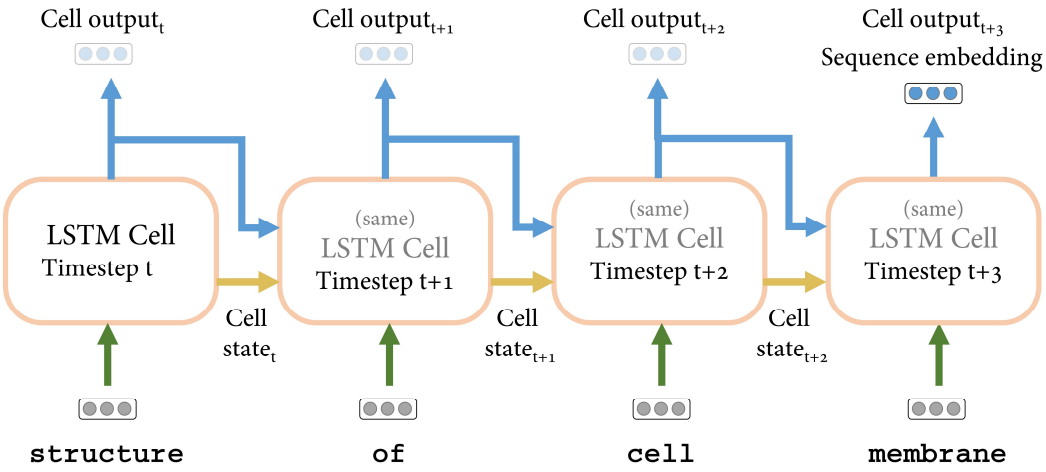
بنام خداوند جان و خرد



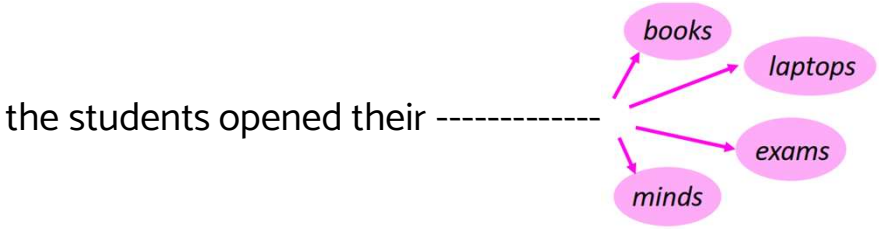
RNNs and Language Models

Mohammad Taher Pilehvar

TeIAS | Tehran Institute for Advanced Studies

Language Modeling

- Language Modeling is the task of predicting what word comes next



More formally: given a sequence of words $x_1, x_2, \ldots, x_t$, compute the probability distribution of the next word x_{t+1}:

$$P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where x_{t+1} can be any word in the vocabulary $V = \{w_1, \ldots, w_{|V|}\}$

Slide from Stanford's CS224N

Language Modeling

- You can also think of Language Model as a system that assigns a probability to a piece of text
- For example, if we have some text $x_1, x_2, \ldots, x_t$, then the probability of this text (according to the Language Model) is:

$$P(x_1, \ldots, x_T) = P(x_1) \times P(x_2|x_1) \times \cdots \times P(x_T|x_{T-1}, \ldots, x_1)$$

$$= \prod_{t=1}^{T} \underbrace{P(x_t|x_{t-1}, \ldots, x_1)}$$

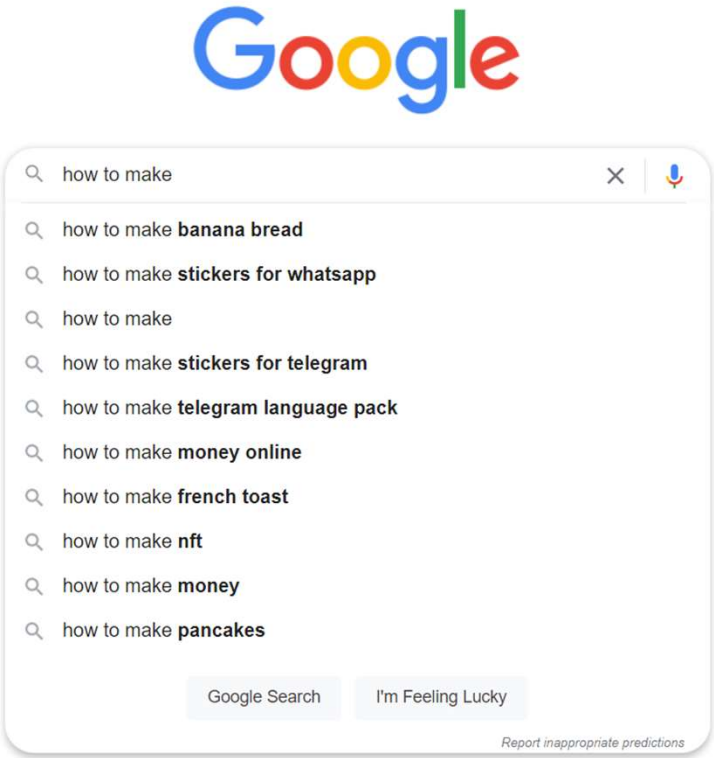
This is what our LM provides

Slide from Stanford's CS224N

You use LMs on the daily basis!

Slide from Stanford's CS224N

You use LMs on the daily basis!



n-gram Language Model

- Question: How to learn a Language Model?
- Answer (pre- Deep Learning): learn an n-gram language model!

- Definition: An *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - four-grams: “the students opened their”

- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word

Slide from Stanford’s CS224N

n-gram Language Model

- First we make a Markov assumption: $x^{(t+1)}$ depends only on the preceding n-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \quad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

- Question: How do we get these n-gram and (n-1)-gram probabilities?
- Answer: By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

Slide from Stanford's CS224N

n-gram Language Model: example

- Suppose we are learning a 4-gram Language Model.

condition on this

discard

~~as the proctor started the clock, the~~ students opened their _____

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

- For example, suppose that in the corpus:
  - "students opened their" occurred 1000 times
  - "students opened their books" occurred 400 times

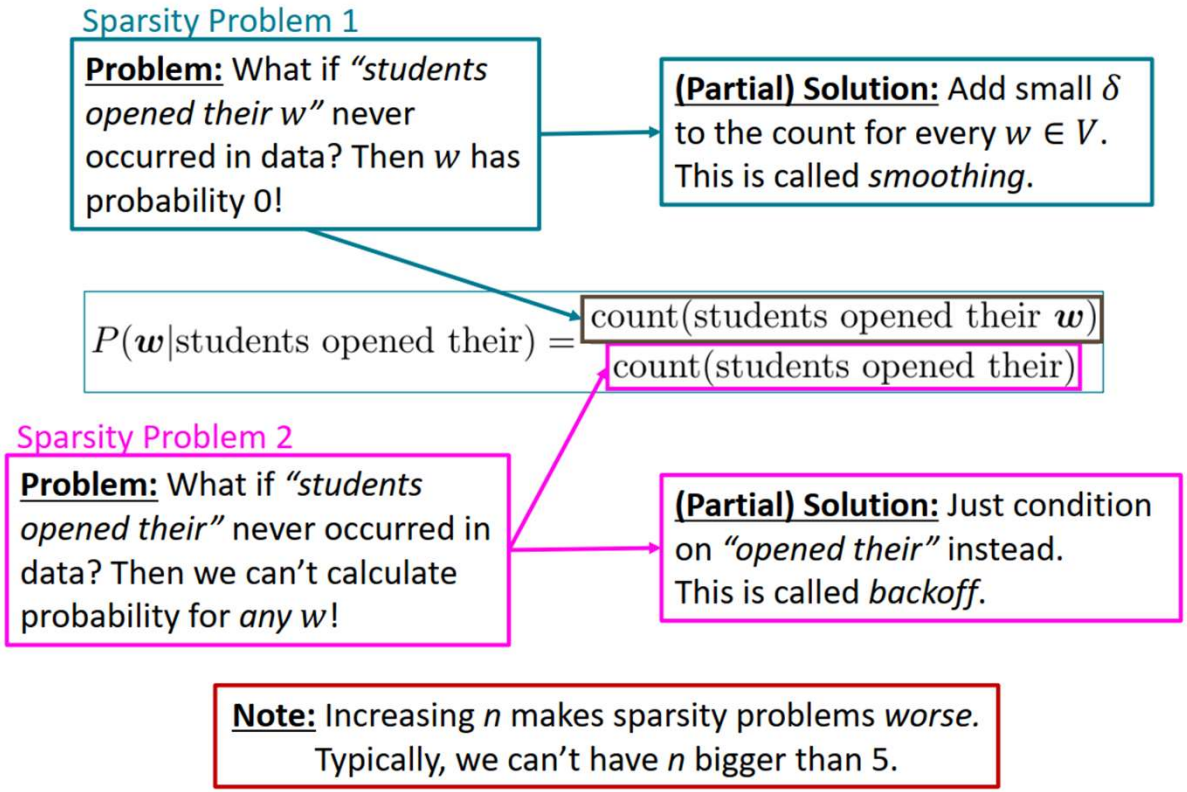
    P(books | students opened their) = 0.4
  - "students opened their exams" occurred 100 times

    P(exams | students opened their) = 0.1

Was it good that we discarded the context of "proctor"?

Slide from Stanford's CS224N

Sparsity Problems with n-gram Language Models



Slide from Stanford's CS224N

Sparsity Problems with n-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count(students opened their } \boldsymbol{w})}{\text{count(students opened their)}}$$

Increasing *n* or increasing corpus increases model size!

Slide from Stanford's CS224N

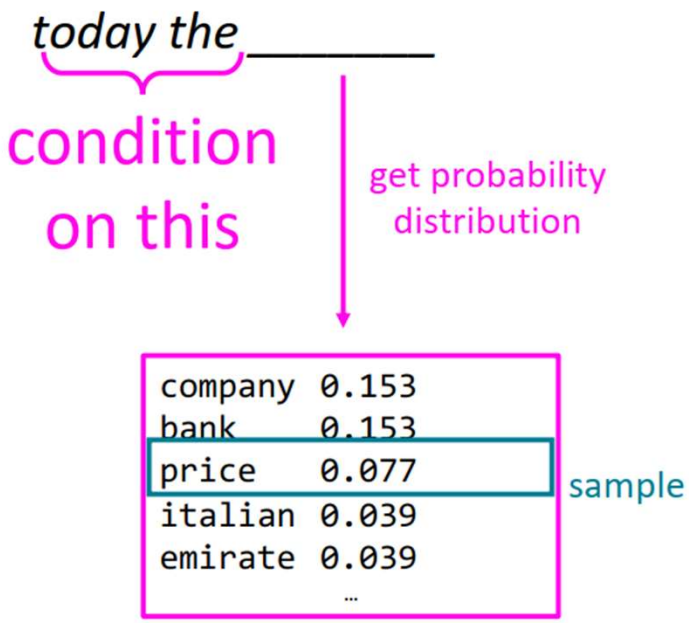
n-gram LMs in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Slide from Stanford's CS224N

Generating text with a n-gram Language Model

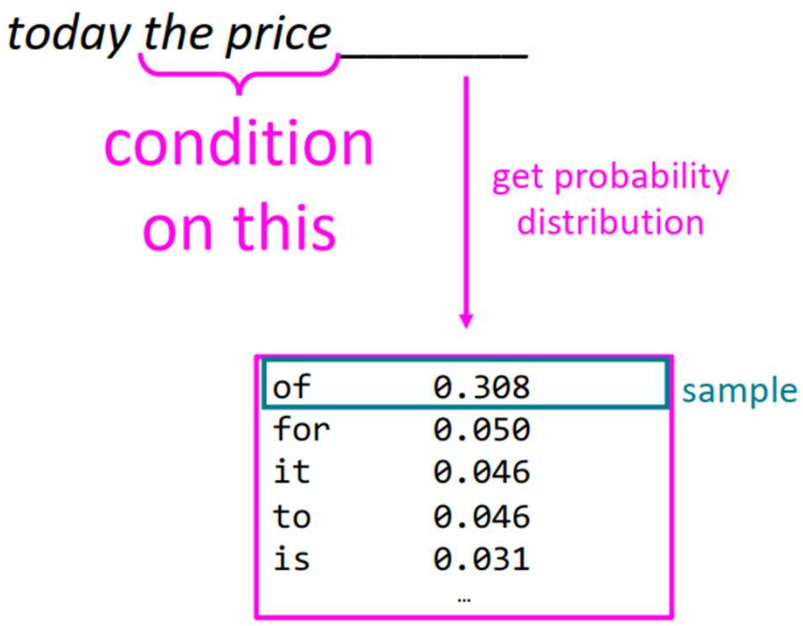
You can also use a Language Model to generate text



Slide from Stanford's CS224N

Generating text with a n-gram Language Model

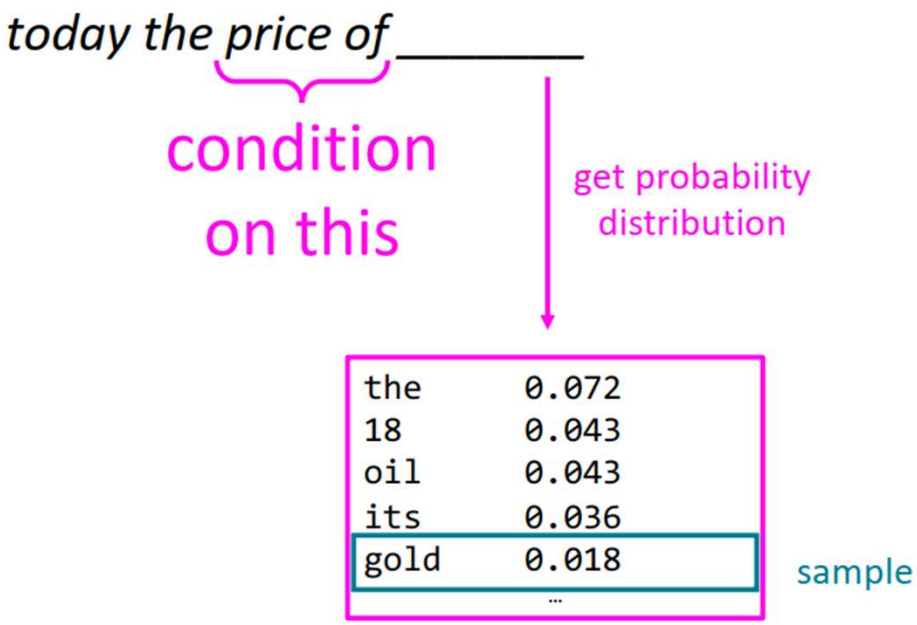
You can also use a Language Model to generate text



Slide from Stanford's CS224N

Generating text with a n-gram Language Model

You can also use a Language Model to generate text



Slide from Stanford's CS224N

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

...but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing *n* worsens sparsity problem, and increases model size...

Slide from Stanford's CS224N

Neural Networks

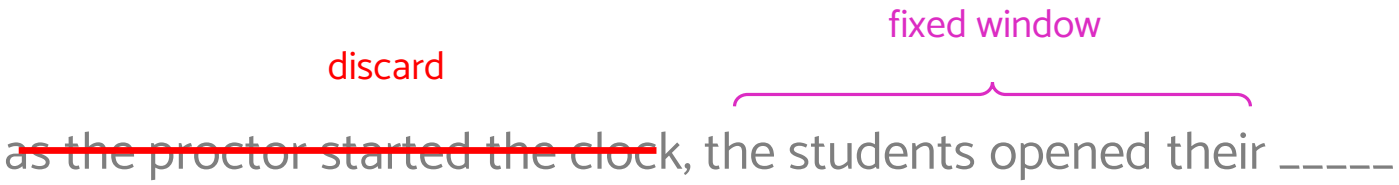
How to build a neural Language Model?

- Recall the Language Modeling task:
  - Input: sequence of words $x_1, x_2, \ldots, x_t$
  - Output: prob distribution of the next word $P(x_{t+1}|x_t, \ldots, x_1)$

- How about a window-based neural model?

Slide from Stanford's CS224N

A fixed-window neural Language Model



Slide from Stanford's CS224N

A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

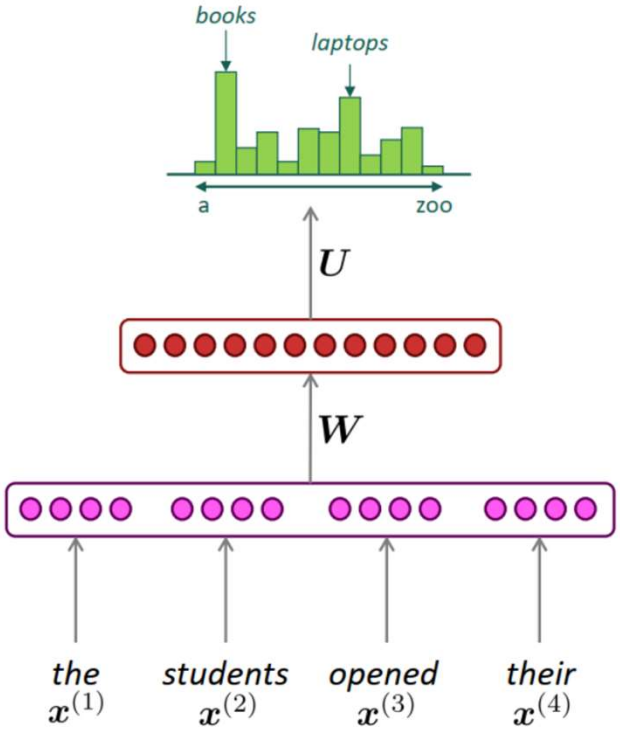
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Slide from Stanford's CS224N

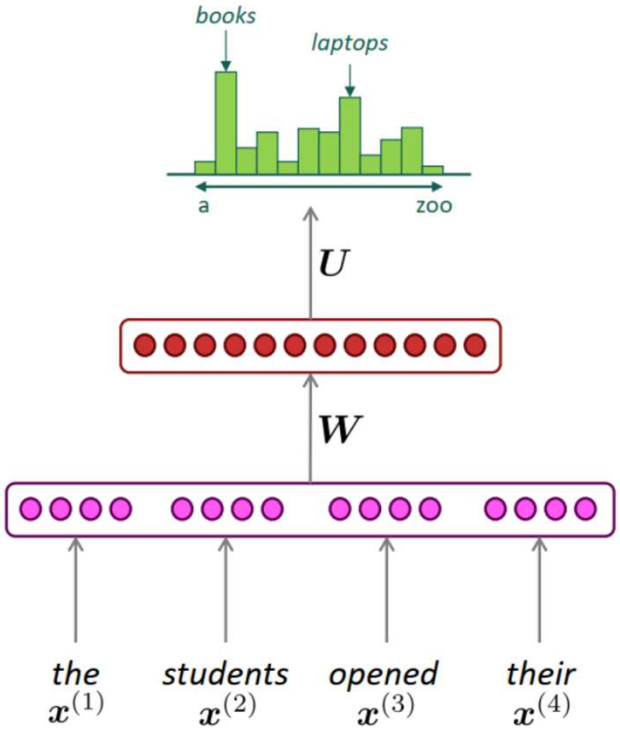
A fixed-window neural Language Model

Improvements over n-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x(0)$ and $x(1)$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.



Slide from Stanford's CS224N

Understanding RNNs

- In Dense and ConvNet, each input is processed independently, with no state kept in between inputs:
  - To process a sequence or a temporal series of data points, you have to show the entire sequence to the network at once: turn it into a single data point.

- In contrast, human processes a text word by word, while keeping memories of what came before.
  - Biological intelligence processes information incrementally while maintaining an internal model of what it's processing, built from past information and constantly updated as new information comes in.

Understanding RNNs

- RNN adopts the same principle, albeit in an extremely simplified version:
  - It processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far.

- The state of the RNN is reset between processing two different, independent sequences.
  - We can still consider one sequence a single data point: a single input to the network.

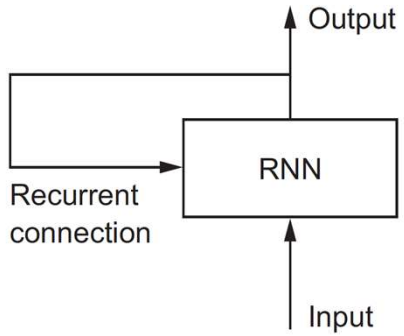


Figure 6.9 A recurrent network: a network with a loop

Understanding RNNs

To make it clear, let's implement the forward pass of a toy RNN in Numpy.

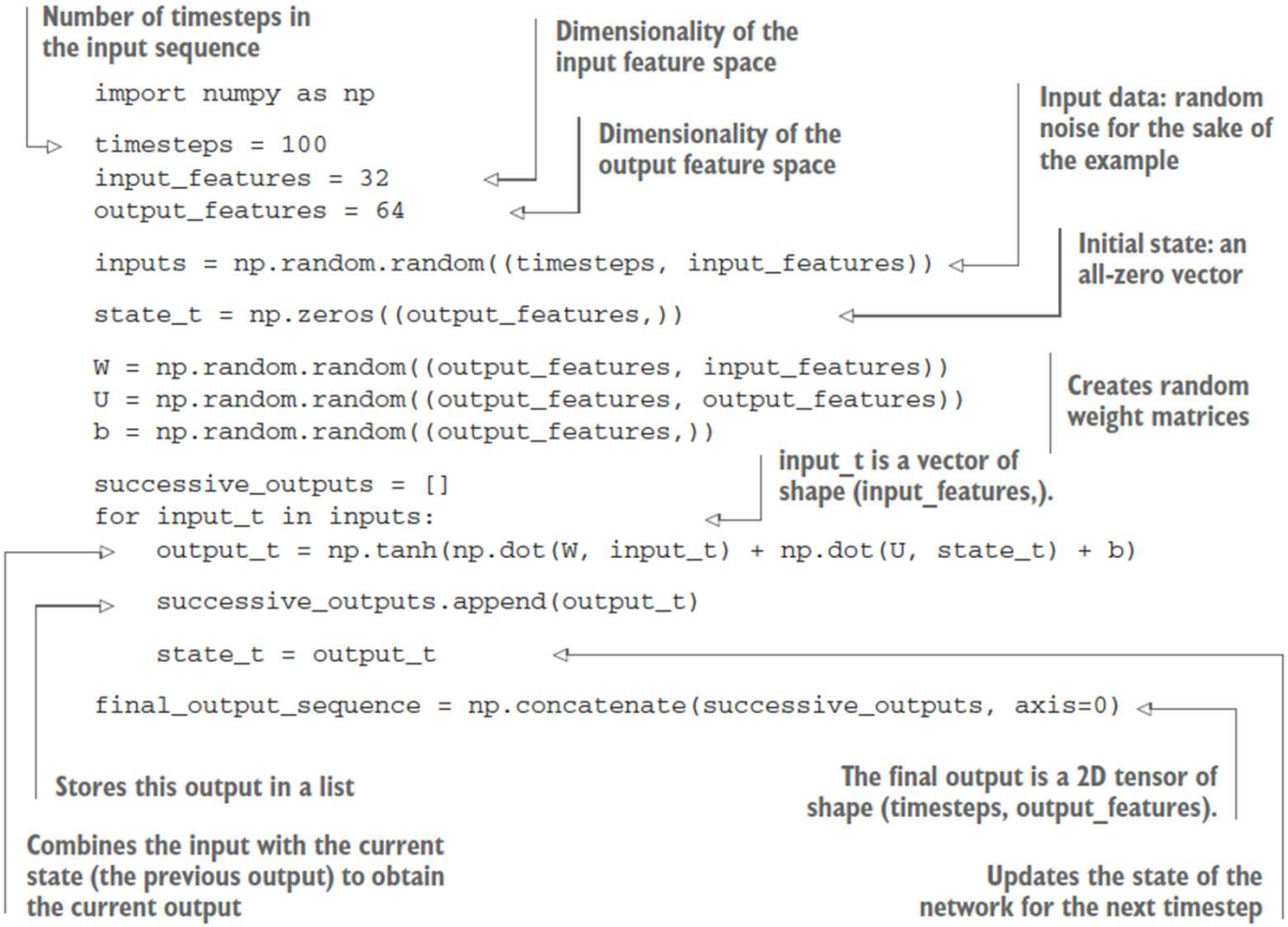
```
state_t = 0                          <-- The state at t
for input_t in input_sequence:       <-- Iterates over sequence elements
    output_t = f(input_t, state_t)
    state_t = output_t               <-- The previous output becomes the state for the next iteration.
```

Understanding RNNs

- You can even flesh out the function **`f`**: the transformation of the input and state into an output will be parameterized by two matrices, **`W`** and **`U`**, and a bias vector.

```
state_t = 0
for input_t in input_sequence:
    output_t = activation(dot(W, input_t) + dot(U, state_t) + b)
    state_t = output_t
```

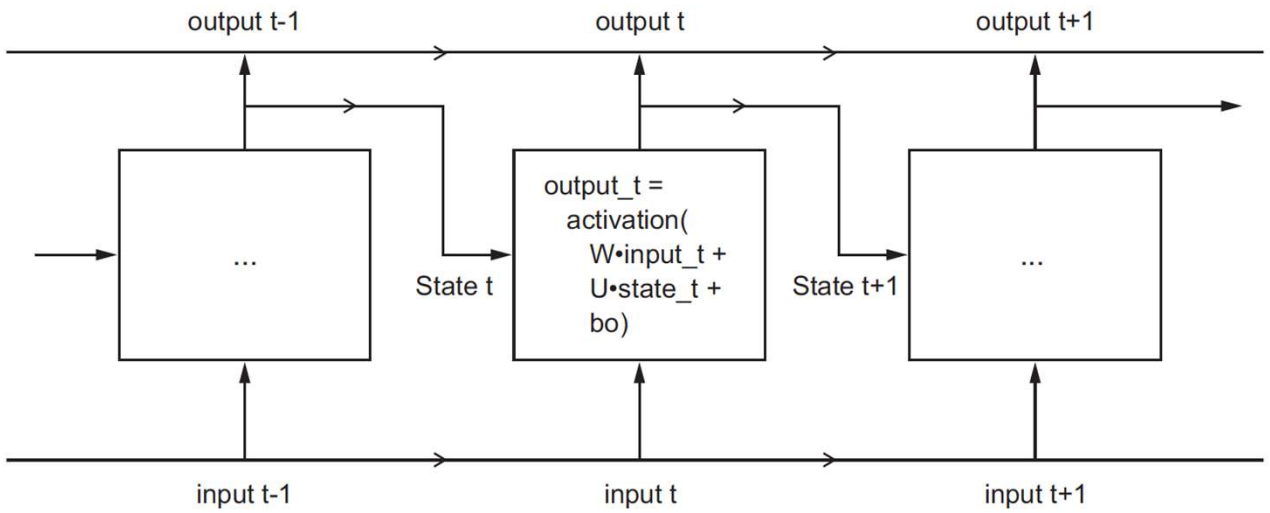
Understanding RNNs



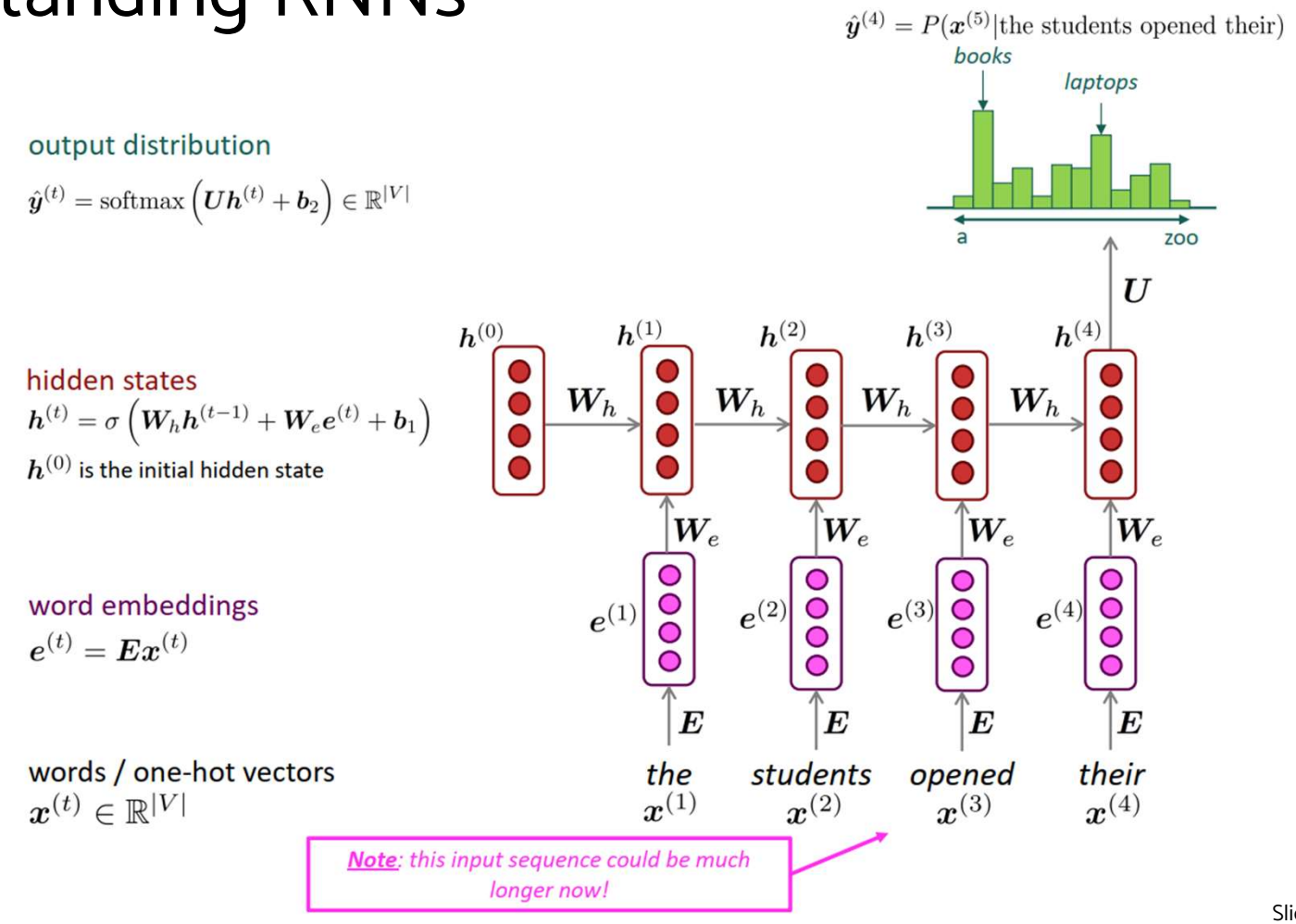
Understanding RNNs

- RNNs are characterized by their step function, such as the following function in this case:

```
output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
```



Understanding RNNs



Slide from Stanford's CS224N

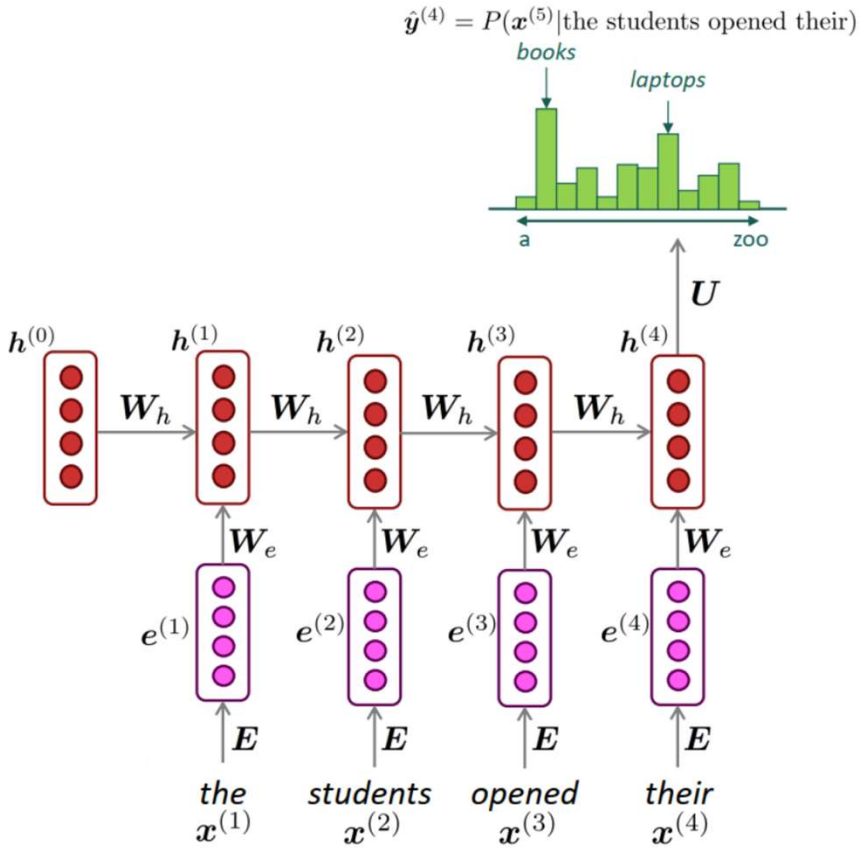
RNN Language Models

RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input context
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back



Slide from Stanford's CS224N

Training an RNN LM

- Get a big corpus of text which is a sequence of words $x^1, \ldots, x^T$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t.
  - i.e. predict probability dist of every word, given words so far
- Loss function on step t is **cross-entropy** between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

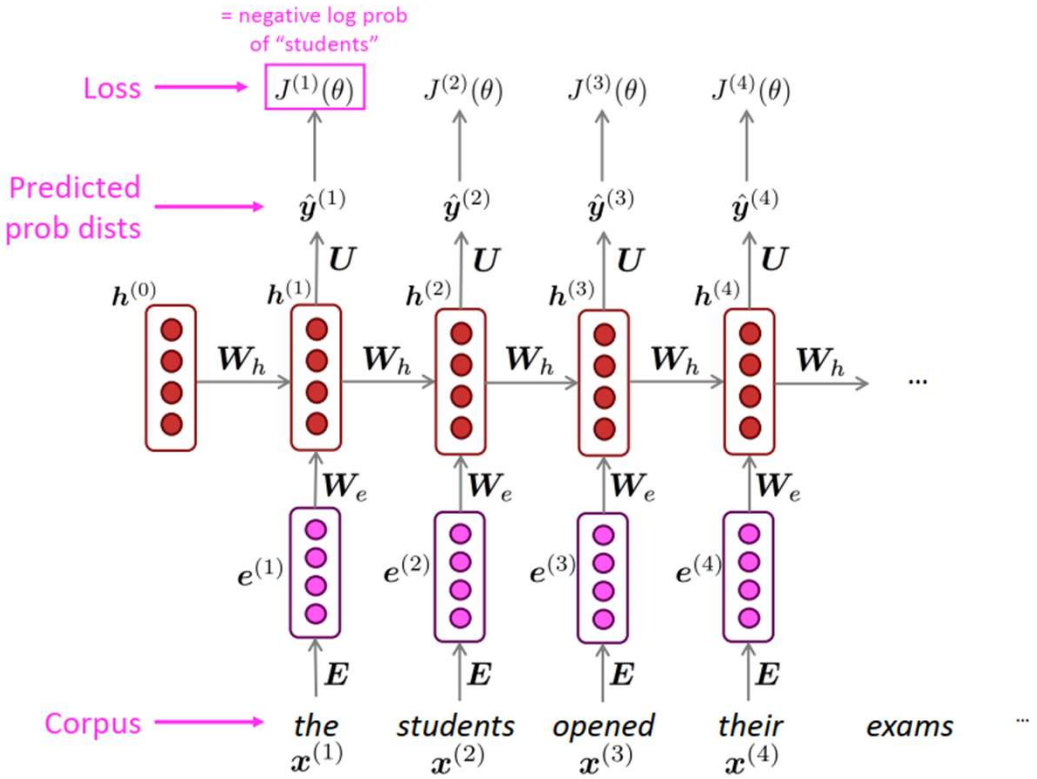
$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

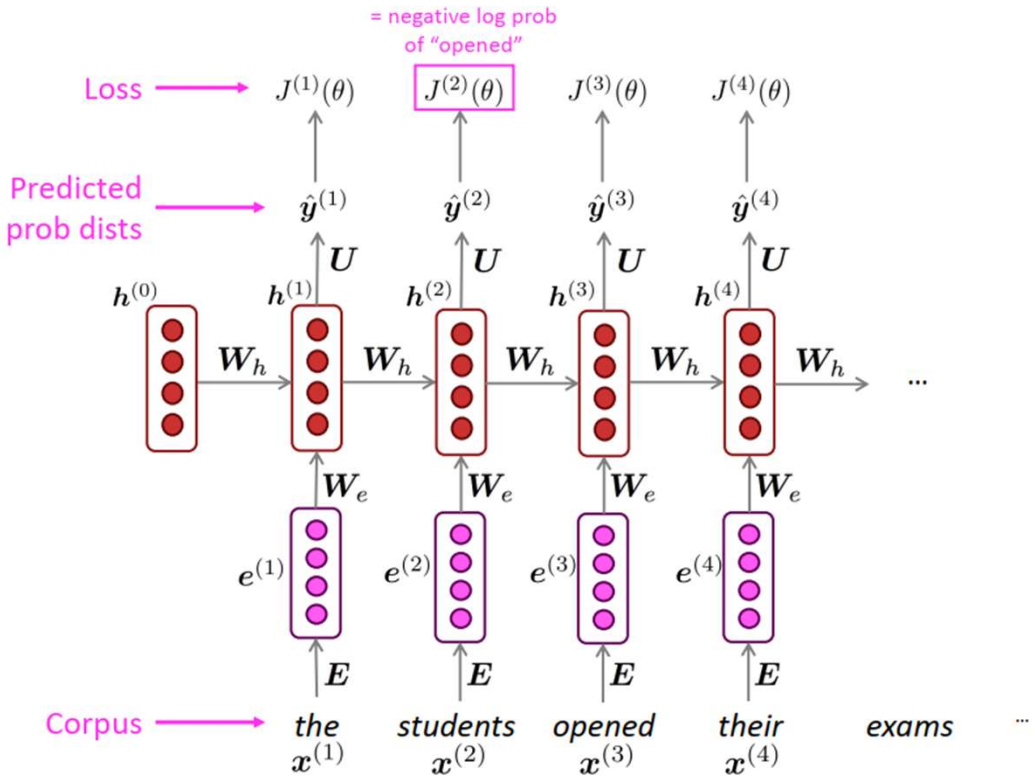
Slide from Stanford's CS224N

Training an RNN LM



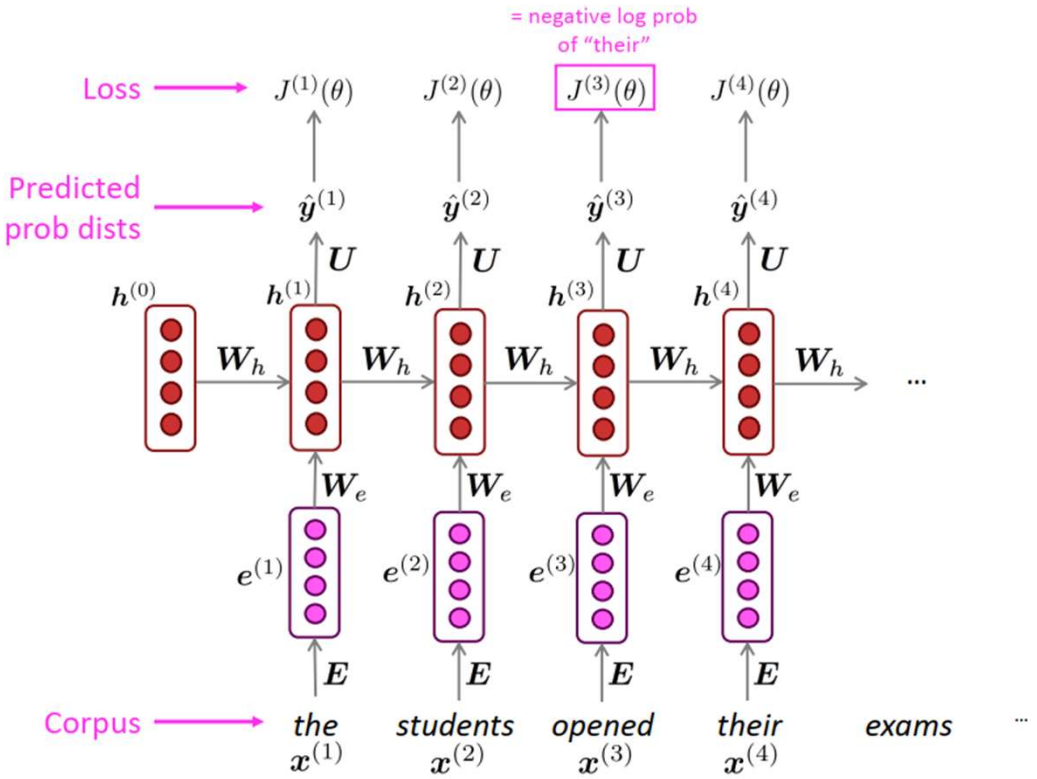
Slide from Stanford's CS224N

Training an RNN LM



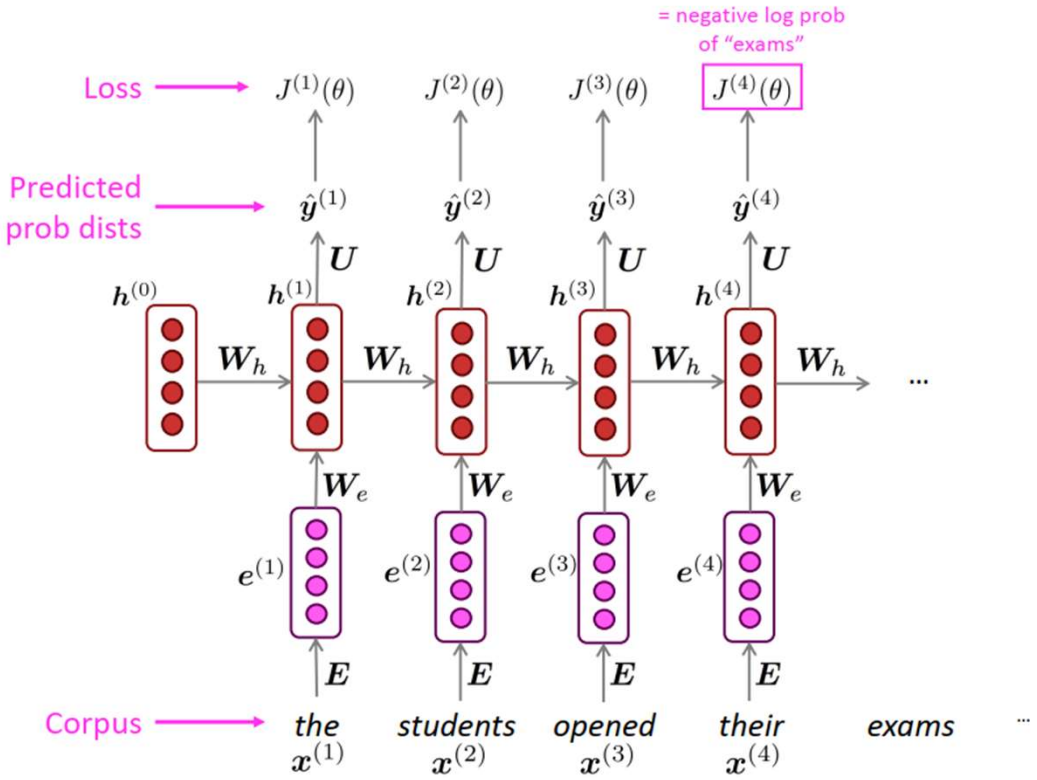
Slide from Stanford's CS224N

Training an RNN LM



Slide from Stanford's CS224N

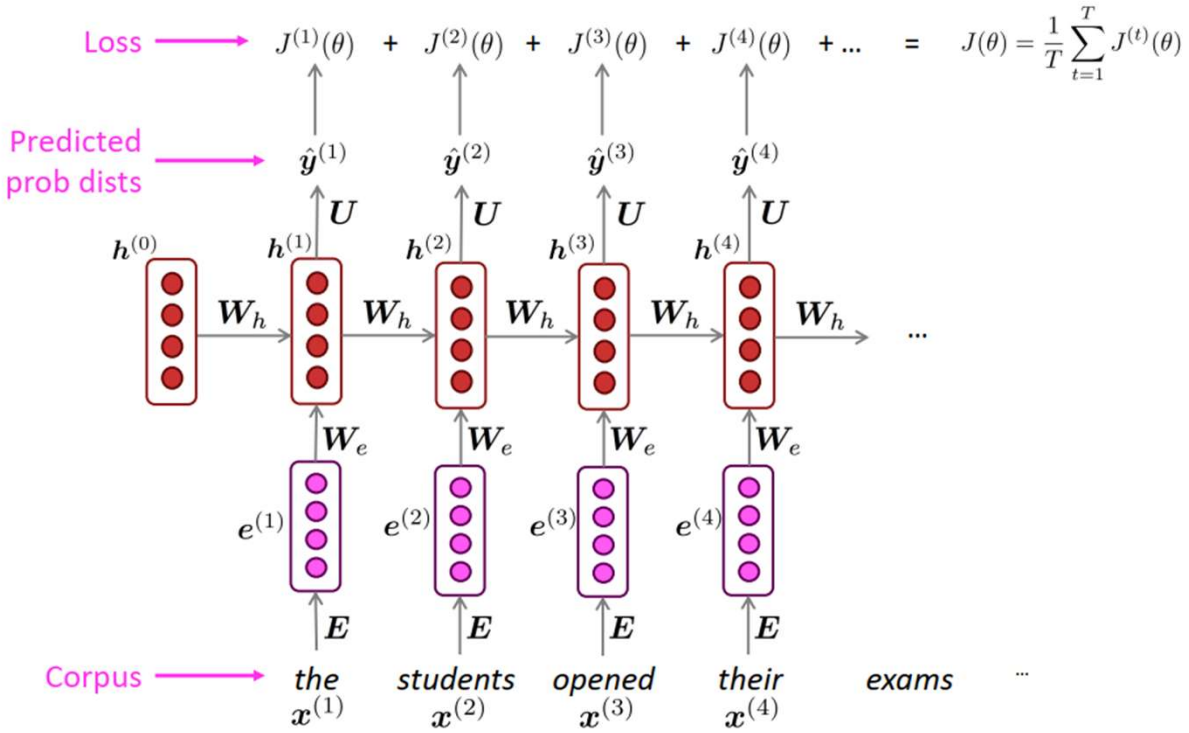
Training an RNN LM



Slide from Stanford's CS224N

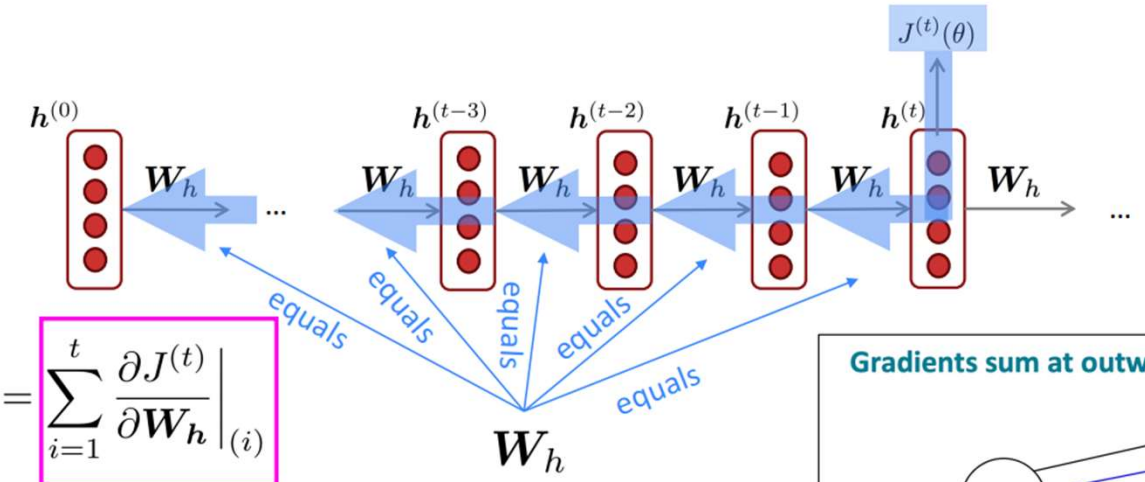
Training an RNN LM

Teacher Forcing



Slide from Stanford's CS224N

Backpropagation for RNNs

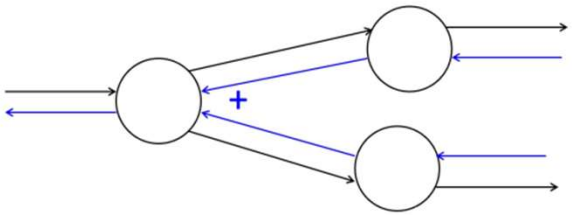


$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\Big|_{(i)}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps *i=t,...,0*, summing gradients as you go. This algorithm is called **"backpropagation through time"** [Werbos, P.G., 1988, *Neural Networks* **1**, and others]

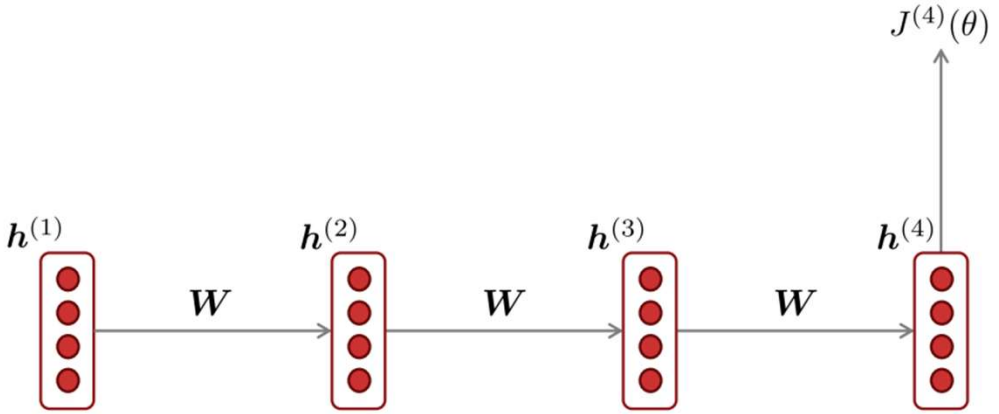
Gradients sum at outward branches



$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}$$

Slide from Stanford's CS224N

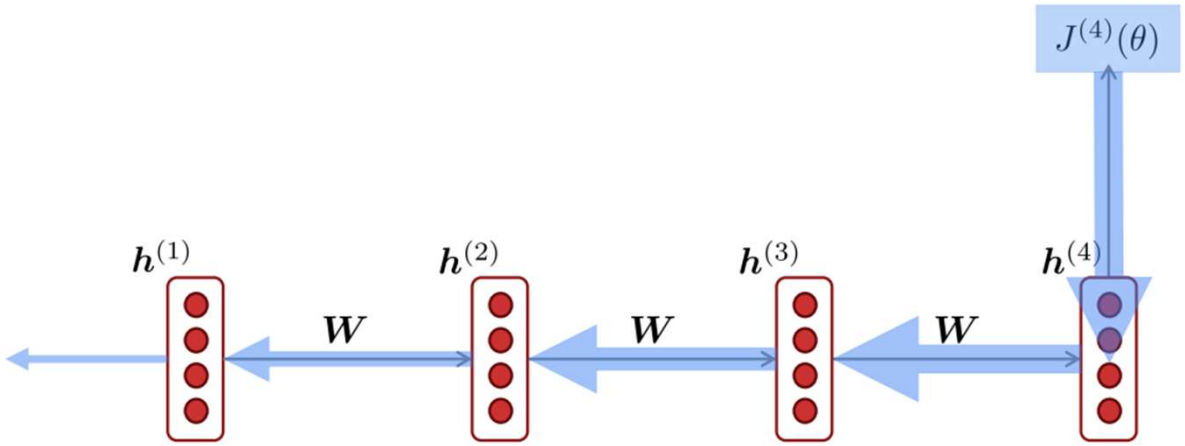
Backpropagation for RNNs



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

Slide from Stanford's CS224N

Backpropagation for RNNs



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Slide from Stanford's CS224N

Vanishing gradient proof sketch (linear case)

- Recall: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$
- What if σ were the identity function, $\sigma(x) = x$?

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \boldsymbol{W}_h \qquad \text{(chain rule)}$$

$$= \boldsymbol{I}\, \boldsymbol{W}_h = \boldsymbol{W}_h$$

- Consider the gradient of the loss $J^{(i)}(\theta)$ on step i, with respect to the hidden state $\boldsymbol{h}^{(j)}$ on some previous step j. *Let* $\ell = i - j$

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \qquad \text{(chain rule)}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \boldsymbol{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^{\ell} \qquad \text{(value of } \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\text{)}$$

If W_h is "small", then this term gets exponentially problematic as ℓ becomes large

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf (and supplemental materials), at http://proceedings.mlr.press/v28/pascanu13-supp.pdf

Vanishing gradient proof sketch (linear case)

- What's wrong with $\boldsymbol{W}_h^\ell$?
- Consider if the eigenvalues of $\boldsymbol{W}_h$ are all less than 1: (sufficient but not necessary)

$$\lambda_1, \lambda_2, \ldots, \lambda_n < 1$$
$$\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n \text{ (eigenvectors)}$$

- We can write $\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^\ell$ using the eigenvectors of $\boldsymbol{W}_h$ as a basis:

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^\ell = \sum_{i=1}^{n} c_i \lambda_i^\ell \boldsymbol{q}_i \approx \mathbf{0} \text{ (for large } \ell)$$

Approaches 0 as ℓ grows, so gradient vanishes

- What about nonlinear activations σ (i.e., what we use?)
  - Pretty much the same thing, except the proof requires $\lambda_i < \gamma$ for some γ dependent on dimensionality and σ

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf (and supplemental materials), at http://proceedings.mlr.press/v28/pascanu13-supp.pdf

Understanding LSTM and GRU layers

SimpleRNN is generally too simplistic to be of real use.

- Although it should theoretically be able to retain at time `t` information about inputs seen many timesteps before, in practice, such long-term dependencies are impossible to learn.

Vanishing gradient problem

Vanishing gradient problem

An effect that is similar to what is observed with non-recurrent networks (feedforward networks) that are many layers deep: as you keep adding layers to a network, the network eventually becomes untrainable.

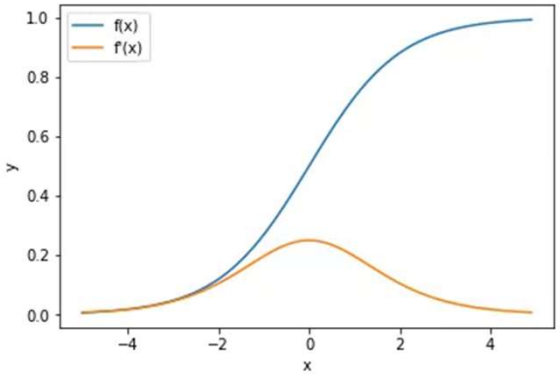
Vanishing gradients

Arises due to the nature of the back-propagation optimization

- The weight and bias values in the various layers within a neural network are updated each optimization iteration by stepping in the direction of the *gradient* of the weight/bias values with respect to the loss function.

Problematic for deep feedforward networks

- Particularly with sigmoid activation functions.
- When the sigmoid function value is either too high or too low, the derivative (orange line) becomes very small i.e. << 1.



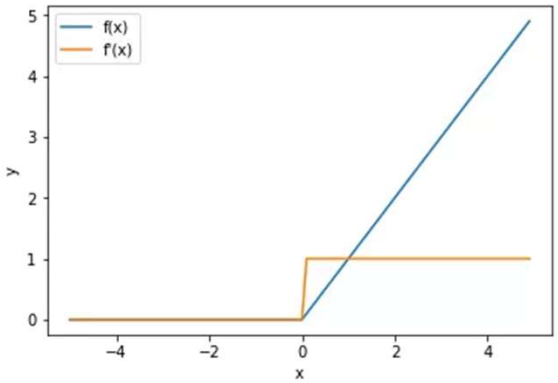
Read more here: https://adventuresinmachinelearning.com/vanishing-gradient-problem-tensorflow/

Vanishing gradients

ReLU activation can partly solve the problem

No degradation of the error signal

But, certain weights can be cancelled out whenever there is a negative input into a given neuron



Vanishing gradients

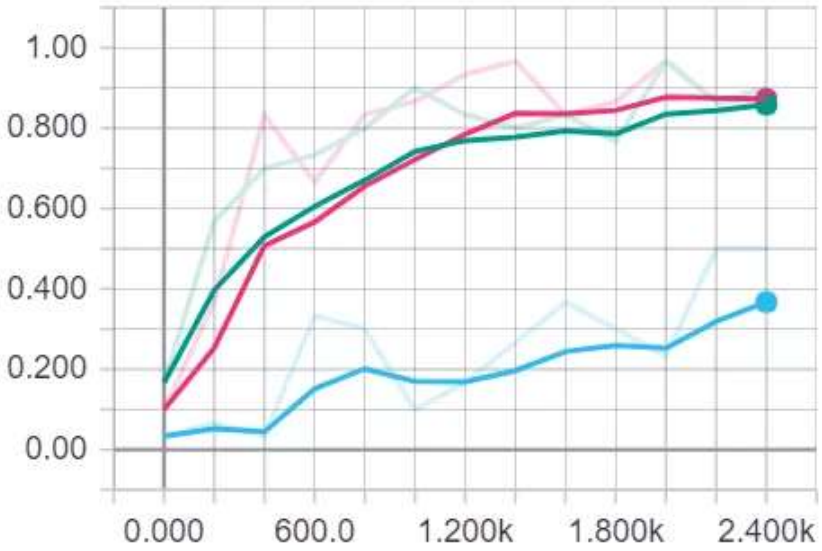
Leaky ReLU

$$f(x) = \max(\alpha x, x)$$

The good thing about the Leaky ReLU activation function is that the derivative when x is below zero is alpha – i.e. it is a small but no longer 0.

Vanishing gradients (in practice)

MNIST using a 7-layer densely connected network

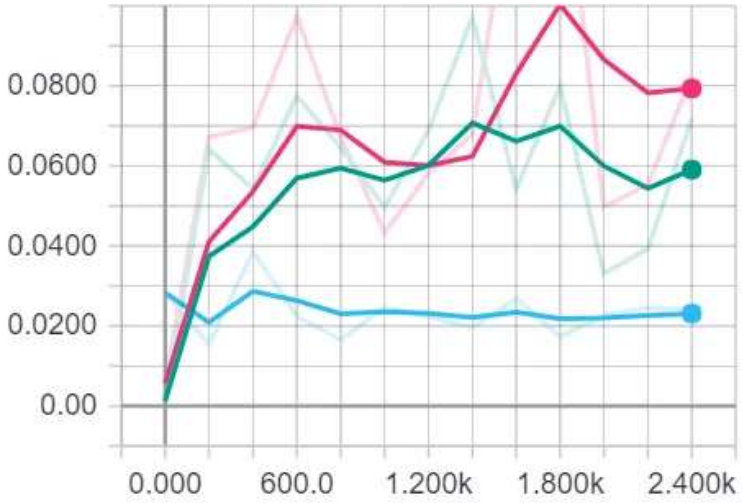


Accuracy of the three activation scenarios
sigmoid (blue),
ReLU (red),
Leaky ReLU (green)

Vanishing gradients (in practice)

MNIST using a 7-layer densely connected network

Mean absolute gradient logs during training

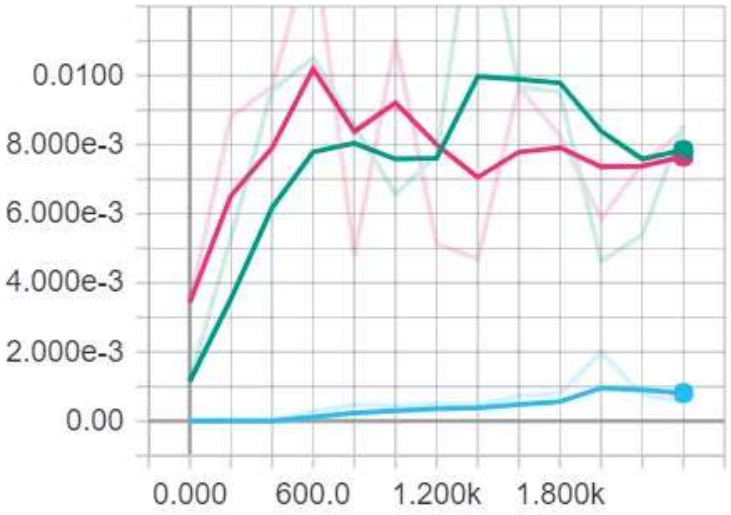


Mean absolute gradients
Output layer (6th layer)
sigmoid (blue),
ReLU (red),
Leaky ReLU (green)

Vanishing gradients (in practice)

MNIST using a 7-layer densely connected network

Mean absolute gradient logs during training



Mean absolute gradients
1st layer
sigmoid (blue),
ReLU (red),
Leaky ReLU (green)

From SimpleRNN to LSTM

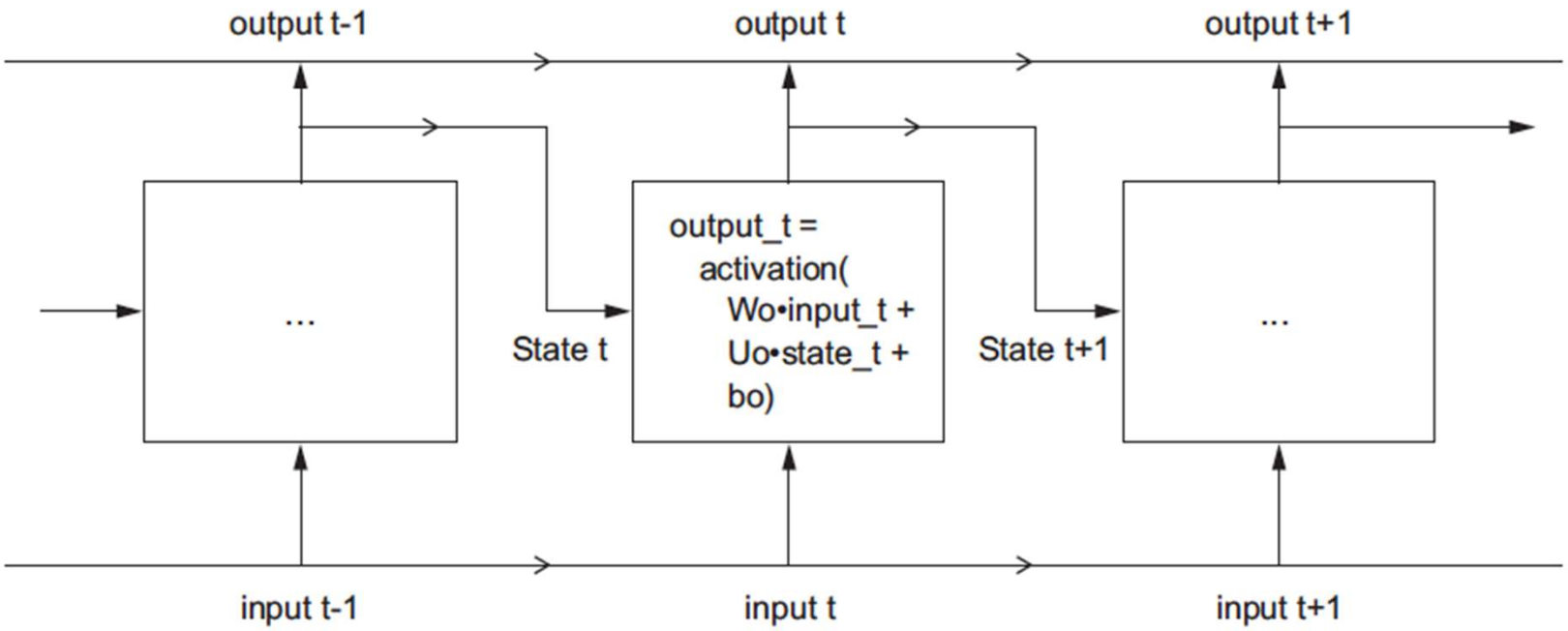


Figure 6.13 The starting point of an `LSTM` layer: a `SimpleRNN`

From SimpleRNN to LSTM

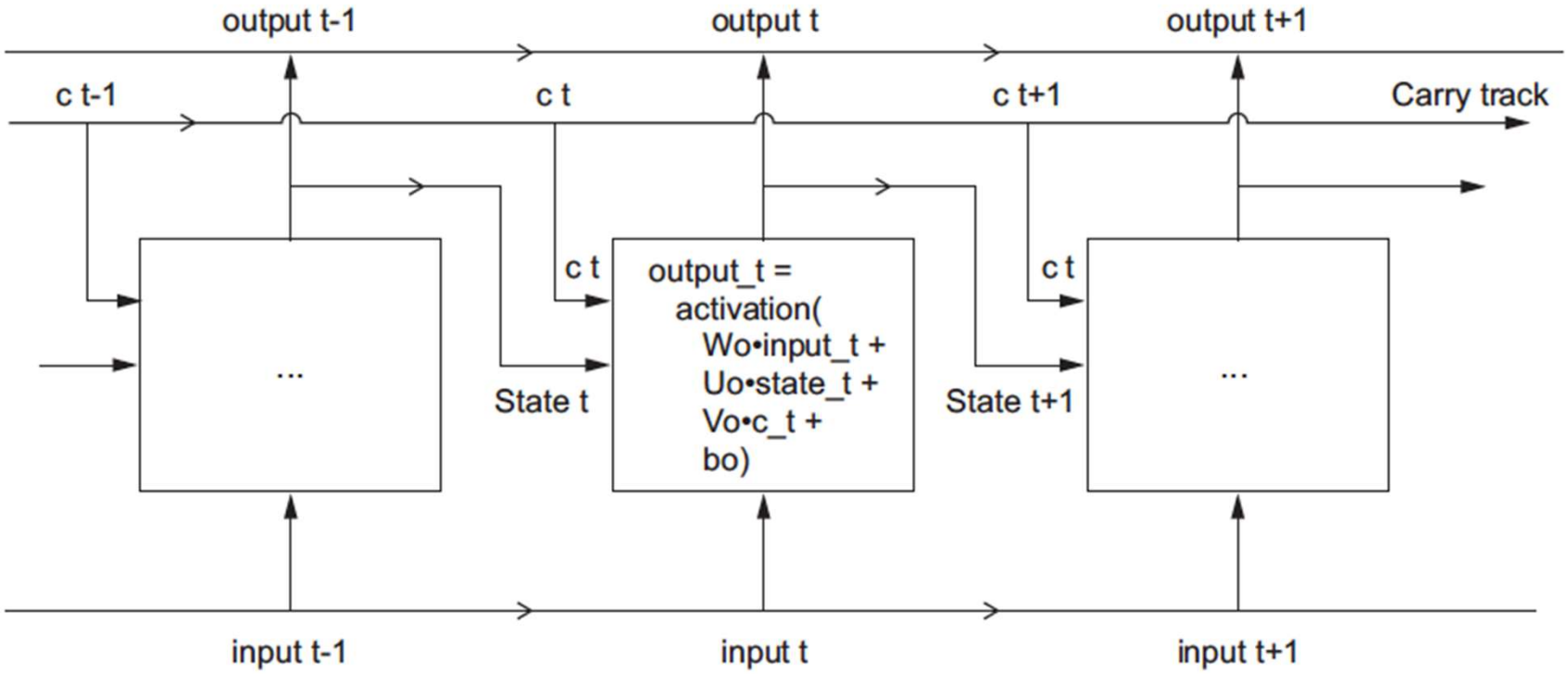


Figure 6.14 Going from a `SimpleRNN` to an `LSTM`: adding a carry track

From SimpleRNN to LSTM

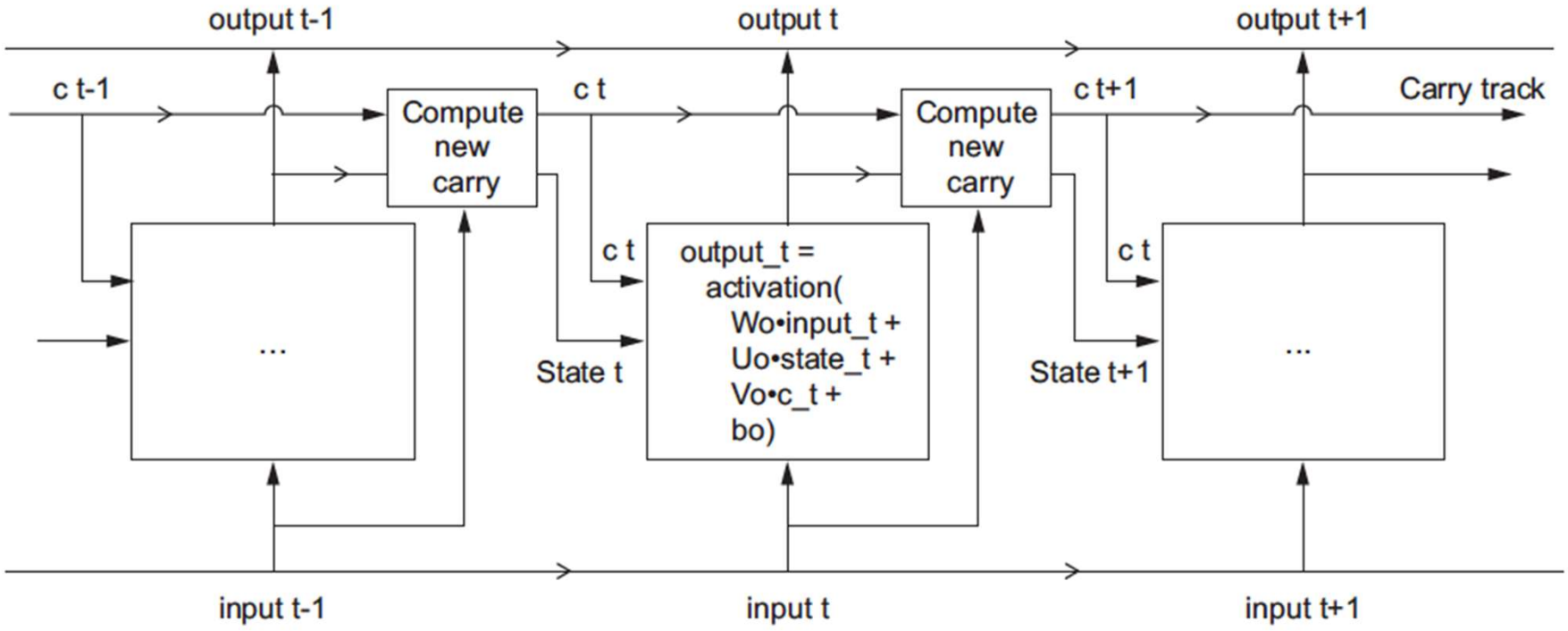


Figure 6.15 Anatomy of an LSTM

From SimpleRNN to LSTM

Listing 6.25 Pseudocode details of the LSTM architecture (1/2)

```
output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + dot(C_t, Vo) + bo)

i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
```

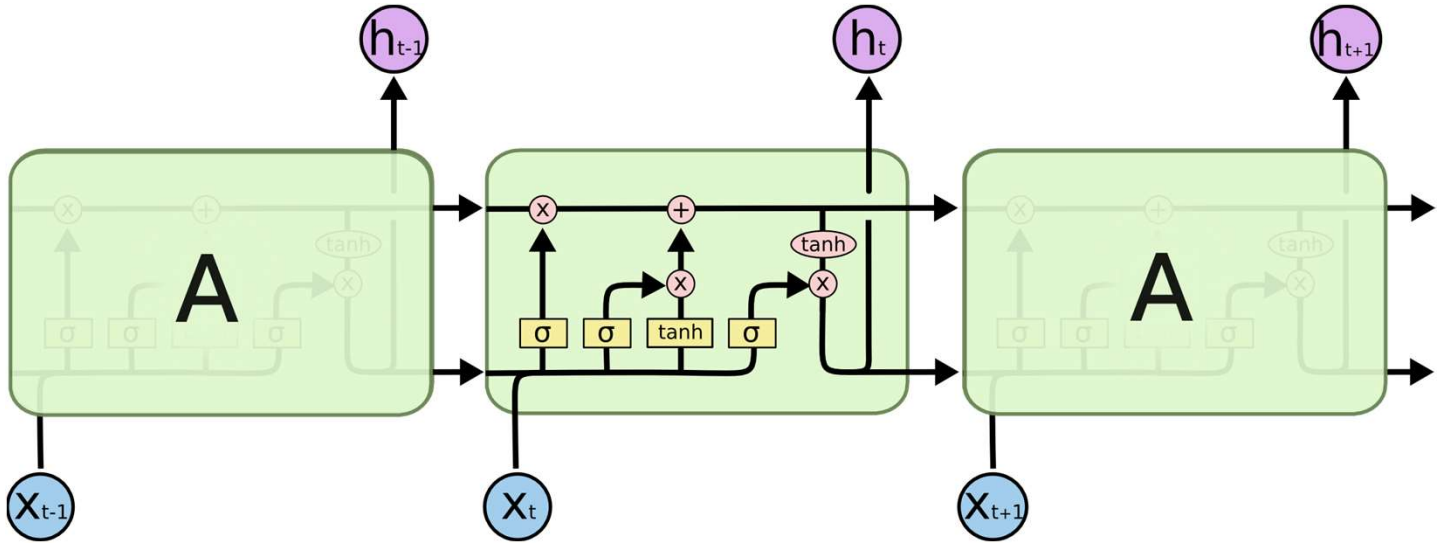
You obtain the new carry state (the next `c_t`) by combining `i_t`, `f_t`, and `k_t`.

Listing 6.26 Pseudocode details of the LSTM architecture (2/2)

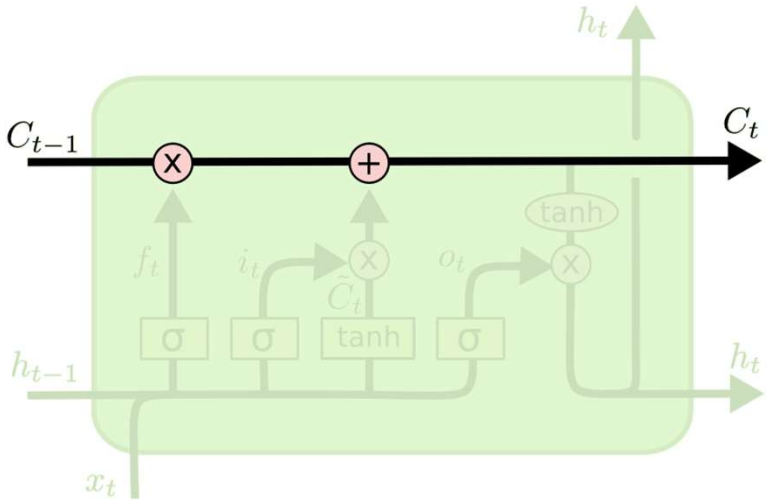
```
c_t+1 = i_t * k_t + c_t * f_t
```

LSTMs: a deeper look

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



LSTMs



Cell state

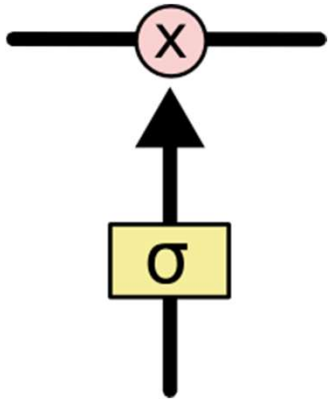
- kind of like a conveyor belt

It runs straight down the entire chain,
with only some minor linear interactions.

It's very easy for information to just flow along it unchanged.

LSTMs

Gates



A way to optionally let information through.

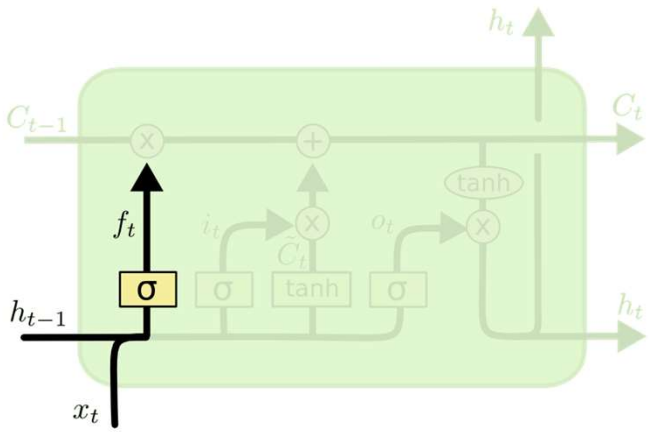
They are composed of a sigmoid neural net layer and a pointwise multiplication operation.

- A value of zero means “let nothing through,” while a value of one means “let everything through!”

LSTMs

Forget gate layer

What information we're going to throw away from the cell state?



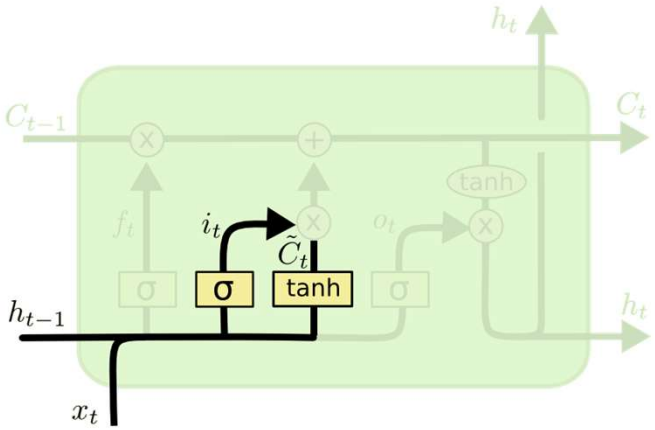
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTMs

What new information we're going to store in the cell state?

Two parts:

1. A sigmoid layer called the *input gate layer* decides which values we'll update.
2. A tanh layer creates a vector of new candidate values, that could be added to the state.

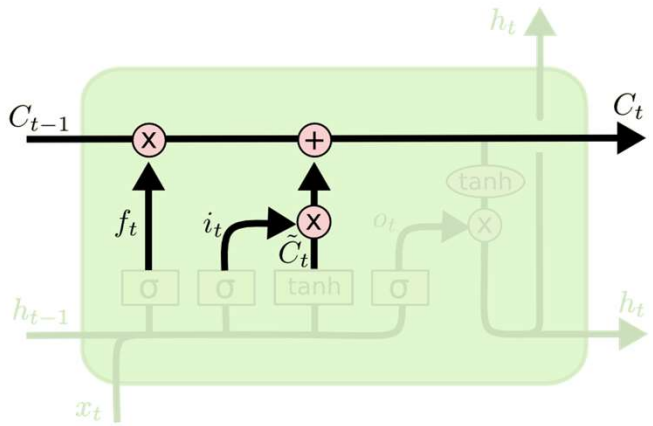


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

LSTMs

It's now time to update the old cell state C_{t-1} to C_t

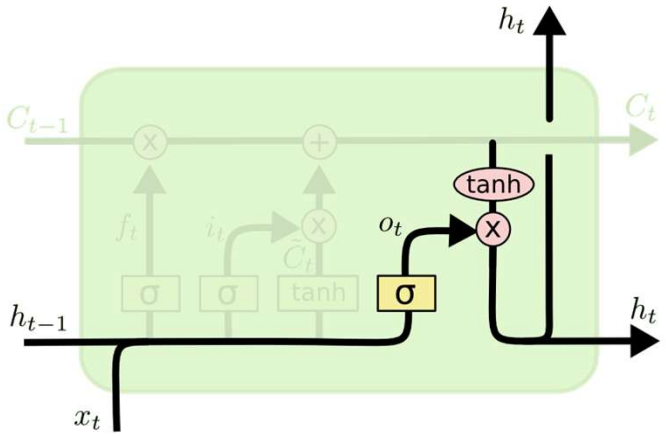


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTMs

Finally, we need to decide what we're going to output. This output will be based on our cell state but will be a filtered version.

1. A sigmoid layer to decide what parts of the cell state to output.
2. Put the cell state through tanh (to push the values to be between –1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

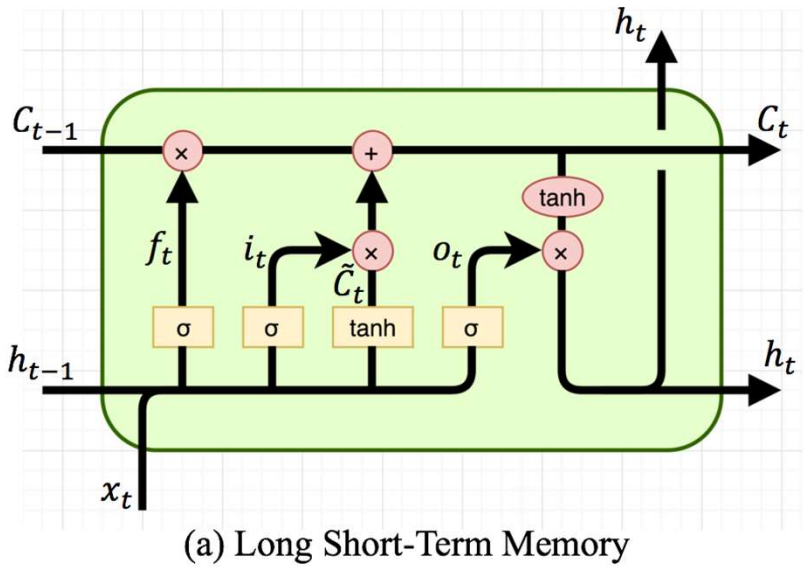


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

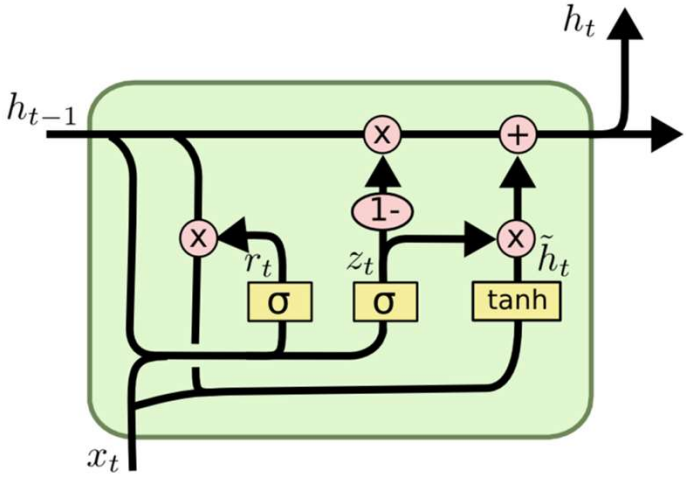
$$h_t = o_t * \tanh\left(C_t\right)$$

GRU: Gated Recurrent Units

Kyunghyun Cho et al (2014)



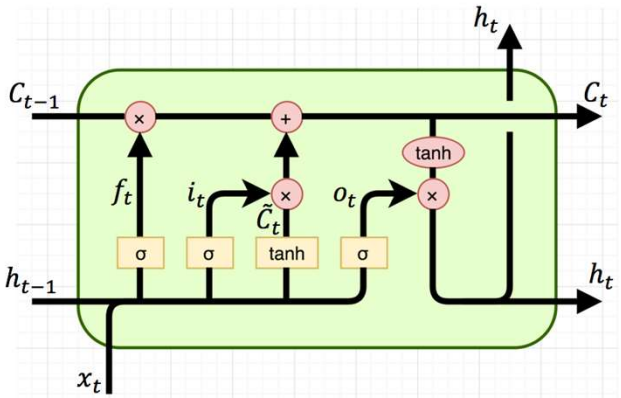
(a) Long Short-Term Memory



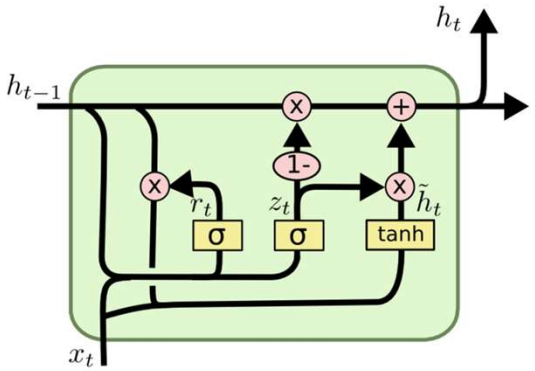
(b) Gated Recurrent Unit

https://isaacchanghau.github.io/post/lstm-gru-formula/

GRU: Gated Recurrent Units



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

$$
\begin{aligned}
i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\
f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\
o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\
\tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right) \\
C_t &= \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\
h_t &= \tanh(C_t) * o_t
\end{aligned}
$$

$$
\begin{aligned}
z_t &= \sigma\left(x_t U^z + h_{t-1} W^z\right) \\
r_t &= \sigma\left(x_t U^r + h_{t-1} W^r\right) \\
\tilde{h}_t &= \tanh\left(x_t U^h + (r_t * h_{t-1}) W^h\right) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

Evaluating language models

- The standard evaluation metric for Language Models is **perplexity**

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss :

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} - \log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

RNNs for sequence labeling

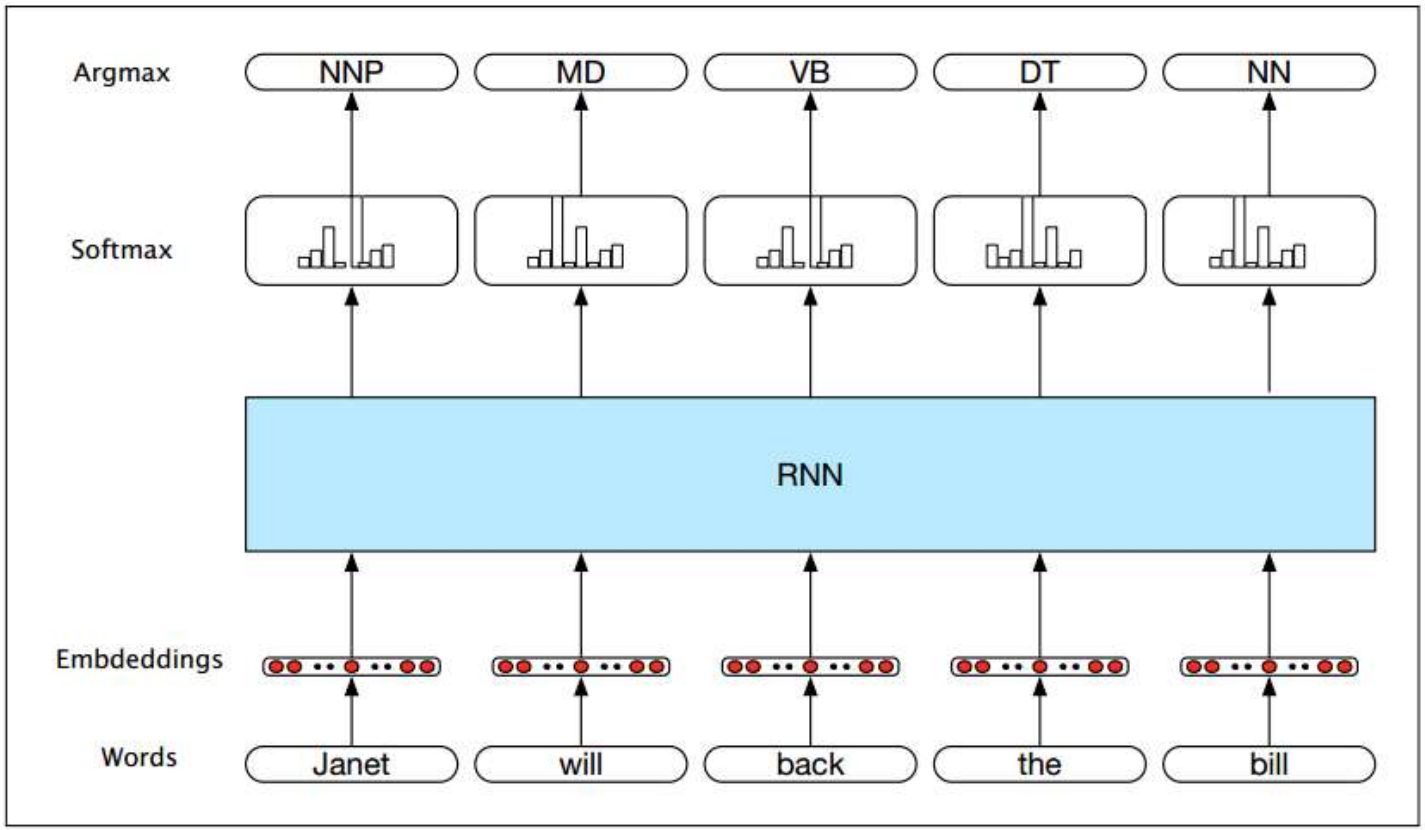


Image source

RNNs for language modeling -- training

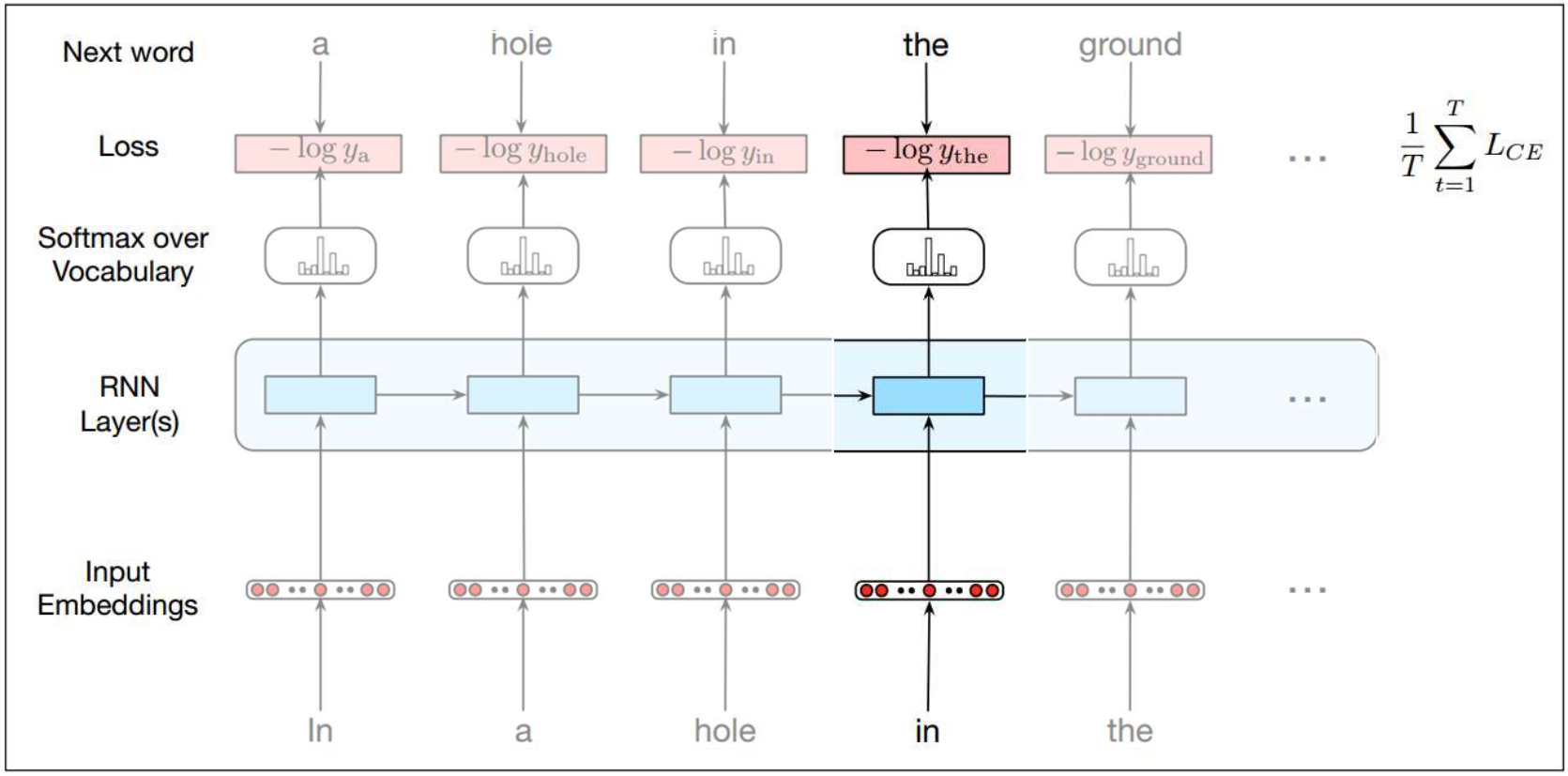


Image [source]

RNNs for language modeling -- autoregressive generation

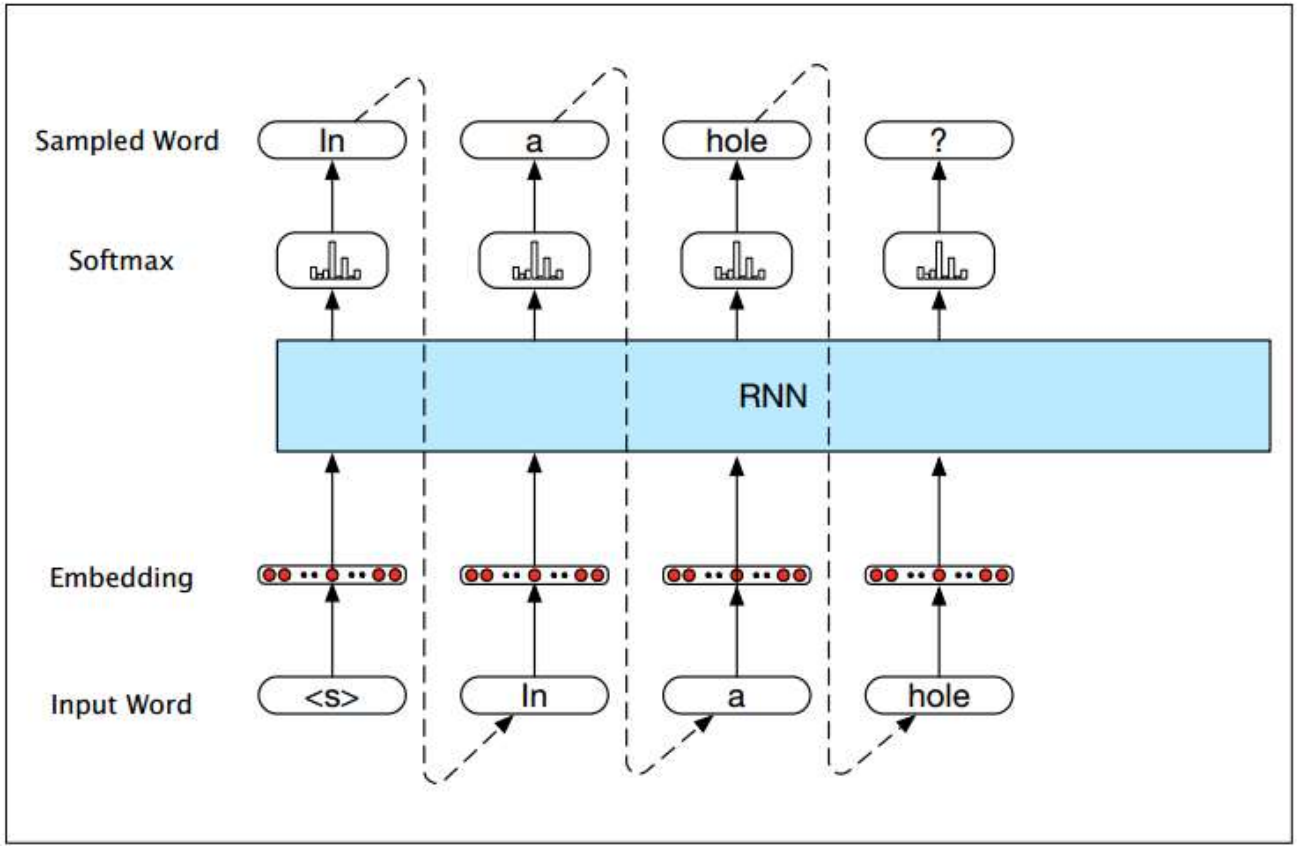
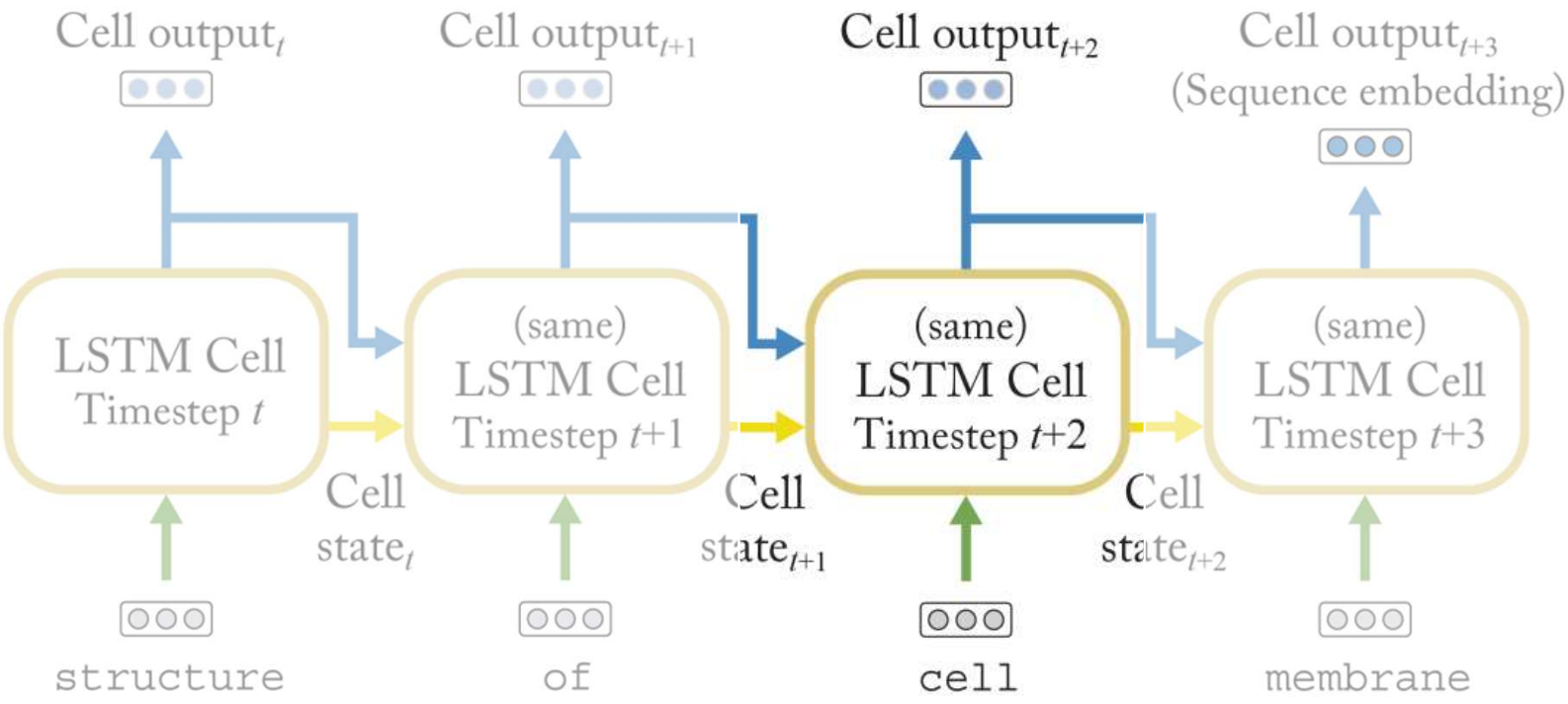


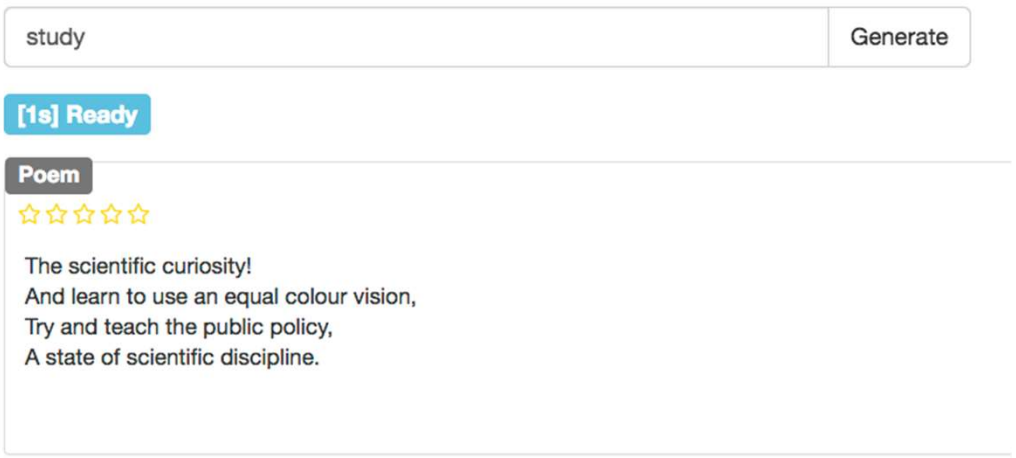
Image source

RNNs for language modeling



RNNs for language modeling

- Persian poetry!



روان را به دانش به کار آوری

به توران به خواب اندر آرد سری

http://52.24.230.241/poem/index.html

RNNs for language modeling

- Persian poetry!

نیست کو معلین
دامشت خانقاه به من من از در میخانه بست
در کار ما به در این خیال ما را ببرد
ز شاه ساهد می و زلف تو را تا برسد
تا به درد تو به میخانه در آب است
که با ز من آباد به در خسابات
به خاک دل بر ت

دید او را اندر آمد ای جوان
گفت می‌دانم امیران را به جان
در دم و زشتی که افزون‌تر بود
چون تو این گفت این جهان خالی بود
گر تو ما آن را کجا زید خداست
گفت حقست آن دم آن هدیه‌هاست
کیمیا بهتر از آن زاری شوی
قبله و قدرت از آن علت شوی
بازیی زین بندها قربان شود
گر نشاید خانه را پستی شود
تا بود کار دگر خندان کند
که پذیرای خدا پیدا کند
یا چو او گردان همی‌گویید هین
در پشیمانی در آید بر زمین

سواران گردنکشان دسته دید
خردمند را او بدان خسته دید
سکندر نگه کرد پس پهلوان
به بدخواه شد شاد و روشن روان
سپاه اندر آمد به پیش سوار
خردمند و شایسته‌ی کارزار
بفرمود تا بنده آگاه دید
چنین تا بر شاه ایران کشید
نهادند چیزی که پوشیده بود
جهان را درم داد و دینار بود
سران افسر از گوهر شاهوار
نخست آفرین کرد بر کردگار
چو بهرام بشنید گریان شدند

Moghaddam and Panahbehagh, 2018
https://www.darbare.com/Post/30084

RNNs for language modeling

- Wikipedia articles

```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be said to be directly in Cantonese
Communication, which followed a ceremony and set inspired prison, training. The
emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest of India with the conflict.
Copyright was the succession of independence in the slop of Syrian influence that
was a famous German movement based on a more popular servicious, non-doctrinal
and sexual power post. Many governments recognize the military housing of the
[[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those Socialism's rule,
was starting to signing a major tripad of aid exile.]]
```

http://karpathy.github.io

RNNs for language modeling

- Wikipedia articles

```
{ { cite journal | id=Cerling Nonforest Department|format=Newlymeslated|none } }
''www.e-complete''.

'''See also''': [[List of ethical consent processing]]

== See also ==
*[[Iender dome of the ED]]
*[[Anti-autism]]

===[[Religion|Religion]]===
*[[French Writings]]
*[[Maria]]
*[[Revelation]]
*[[Mount Agamul]]
```

http://karpathy.github.io

RNNs for language modeling

- XML

```
<page>
  <title>Antichrist</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestamp>2002-08-03T18:14:12Z</timestamp>
    <contributor>
      <username>Paris</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">#REDIRECT [[Christianity]]</text>
  </revision>
</page>
```

http://karpathy.github.io

RNNs for language modeling

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciy in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma **??**. Namely, by Lemma **??** we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition* **??** *(without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation* **??**. *Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (**??**). On the other hand, by Lemma **??** we see that

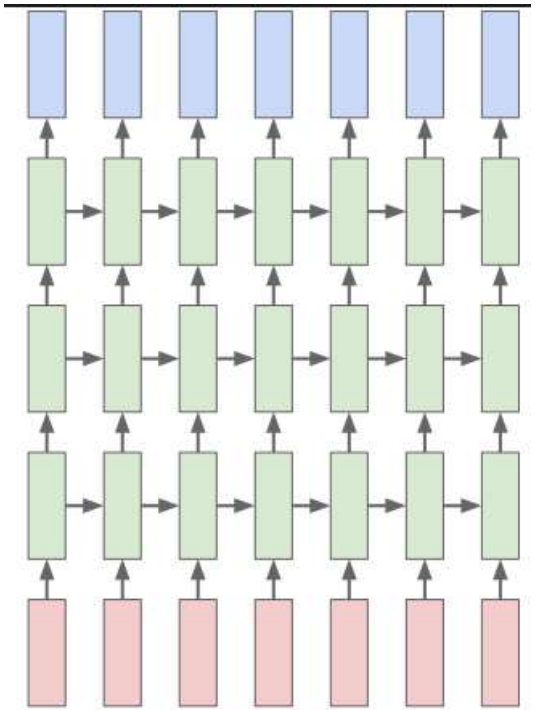
$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

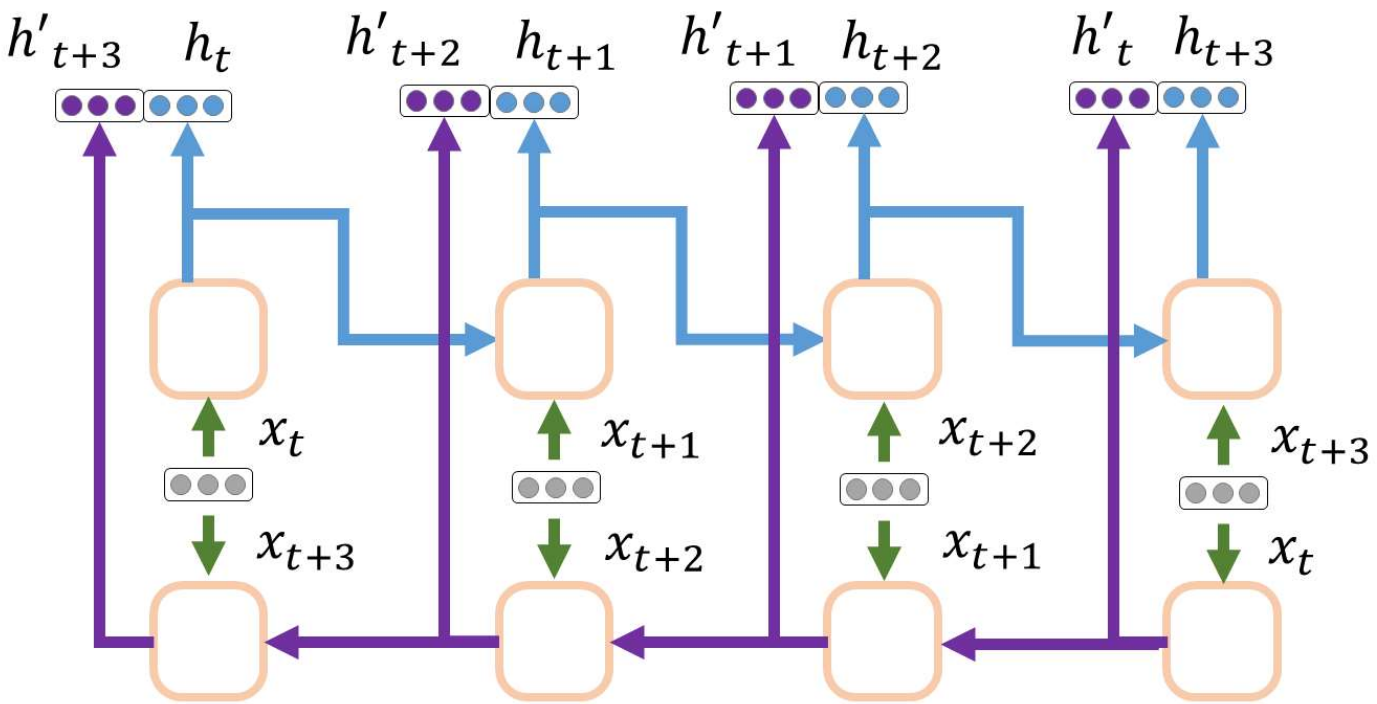
Stacking recurrent layers

- It is a generally a good idea to increase the capacity of your network until overfitting becomes your primary obstacle (assuming that you are already taking basic steps to mitigate overfitting, such as using dropout).

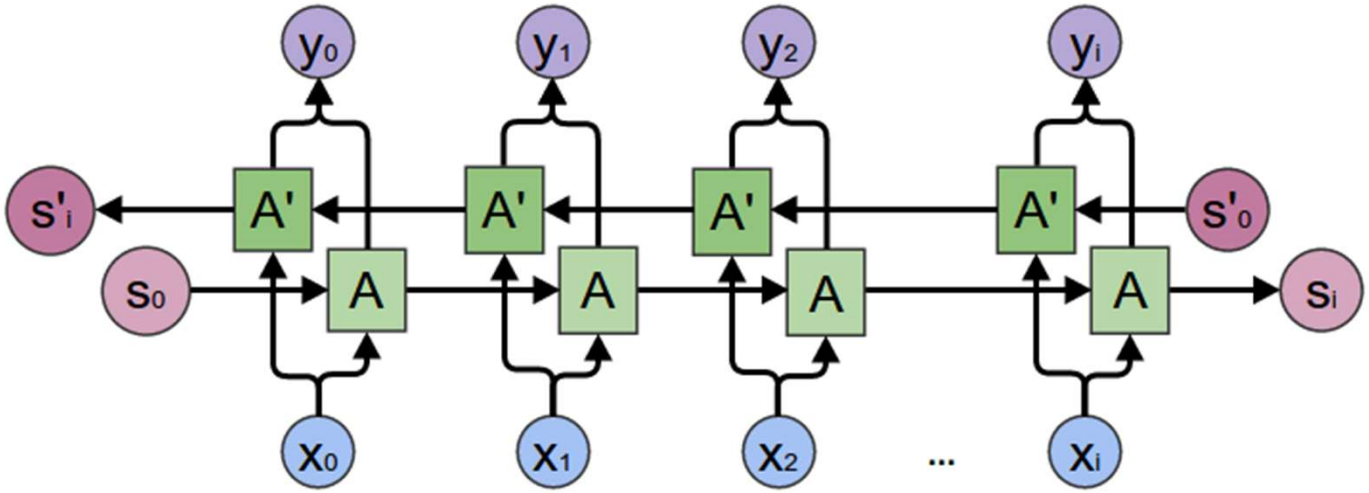
- Recurrent layer stacking is a classic way to build more powerful recurrent networks:
  - For instance, what currently powers the Google translate algorithm is a stack of seven large LSTM layers -- that's huge.



Bidirectional RNNs

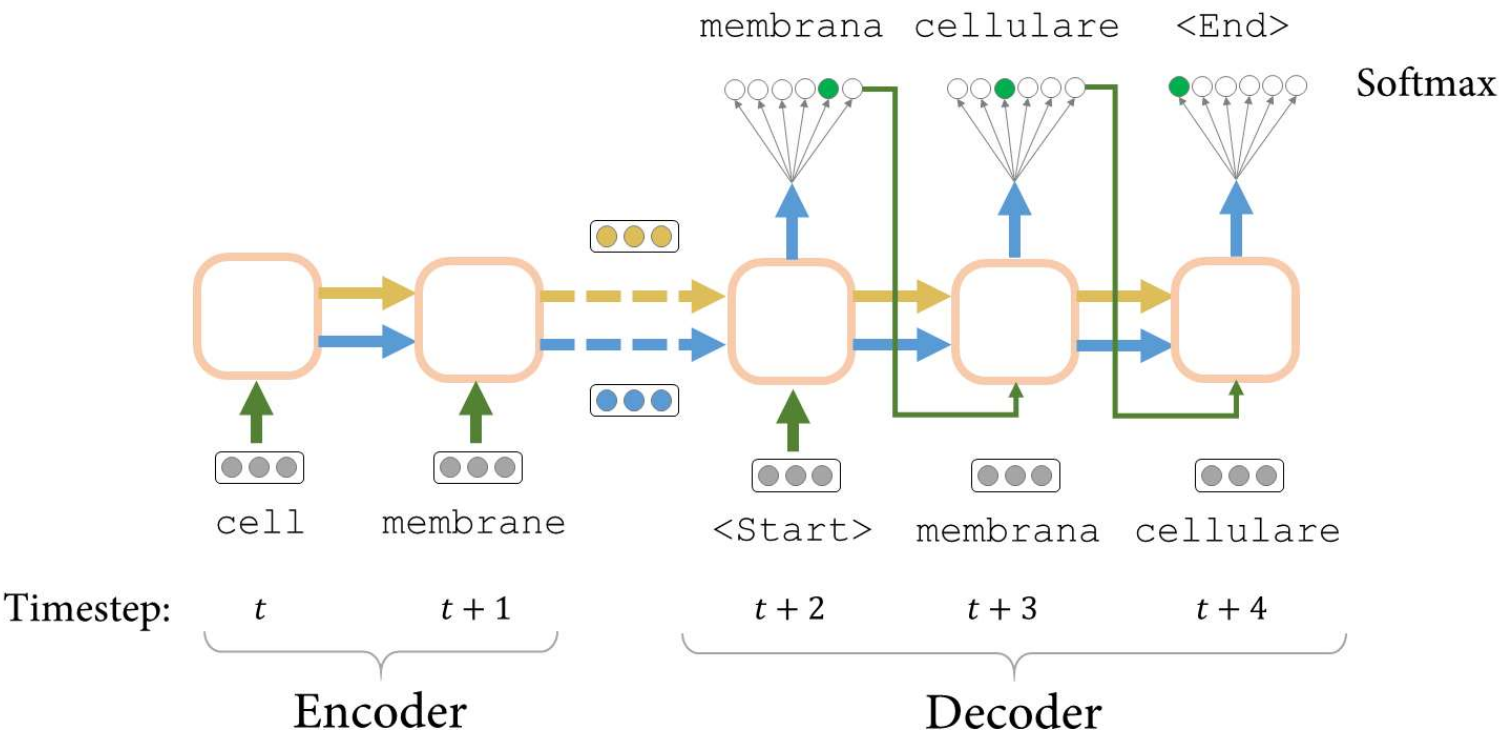


Bidirectional RNNs

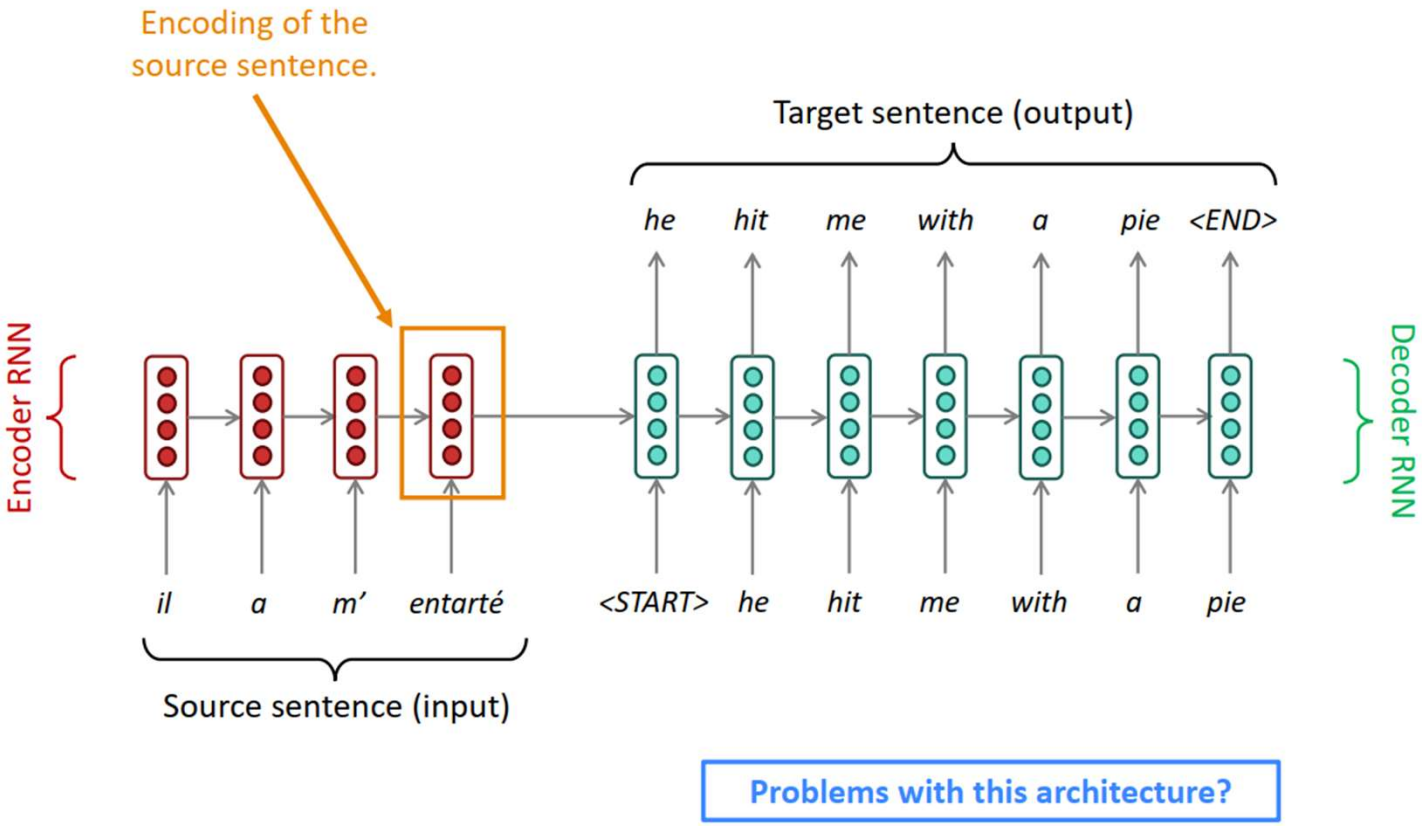


http://colah.github.io

Sequence transduction (Seq2seq)



Seq2seq: the bottleneck problem



Slide from Stanford's CS224N

Attention

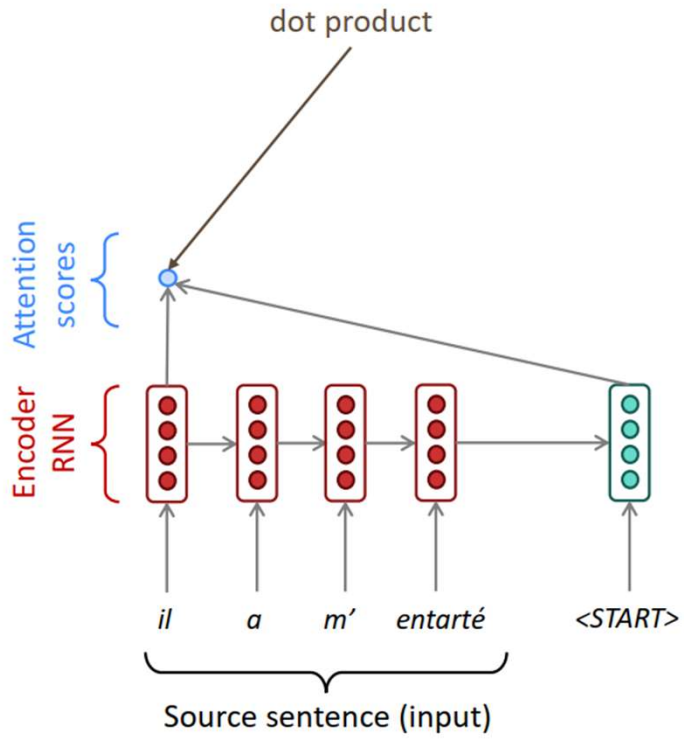
Attention provides a solution to the bottleneck problem.

Core idea:

- on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence

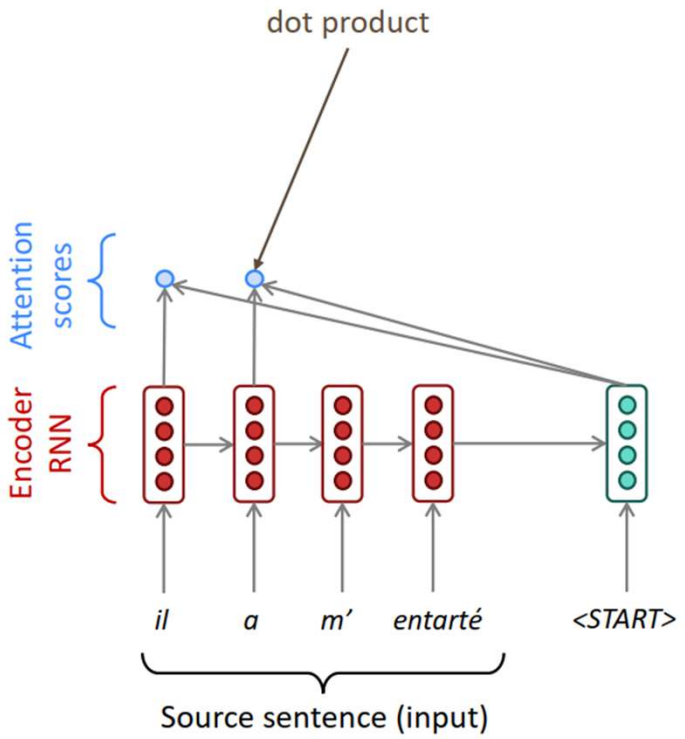
Slide from Stanford's CS224N

Seq2seq with attention



Slide from Stanford's CS224N

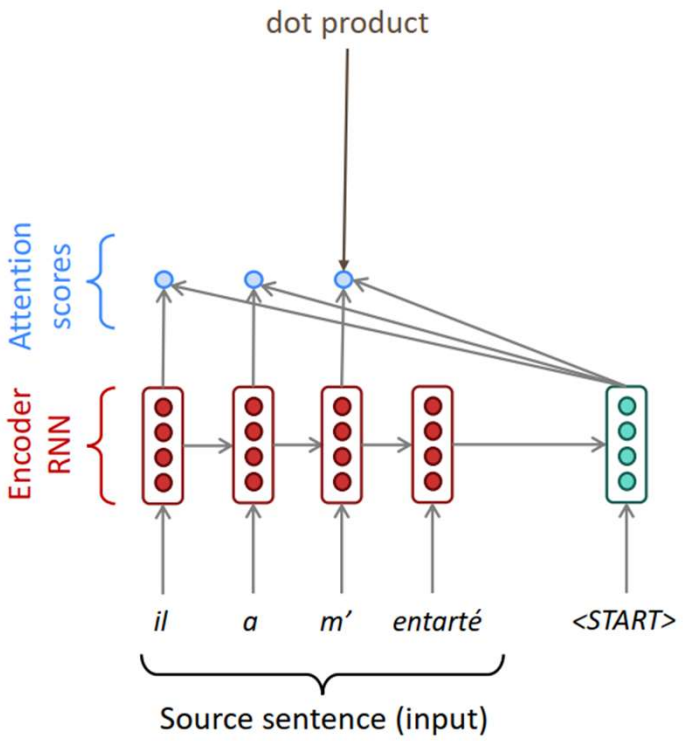
Seq2seq with attention



Decoder RNN

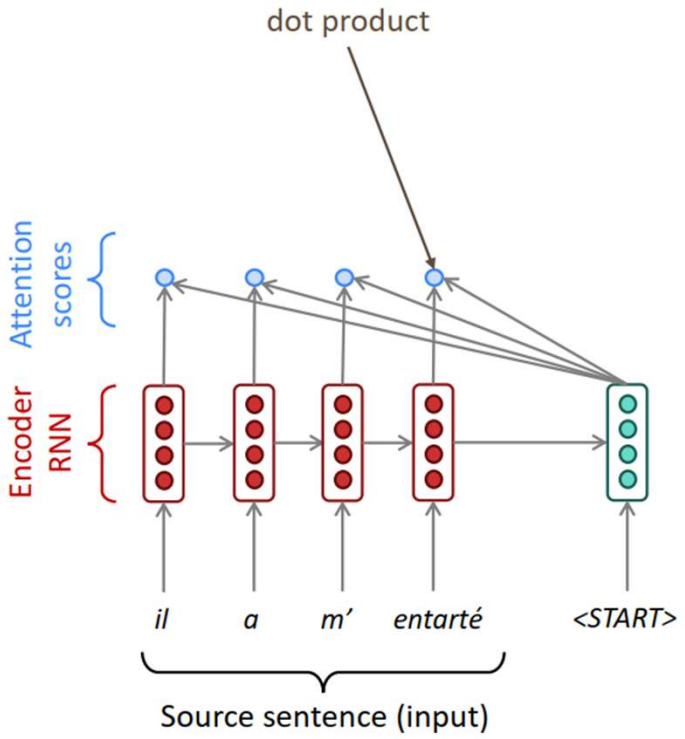
Slide from Stanford's CS224N

Seq2seq with attention



Slide from Stanford's CS224N

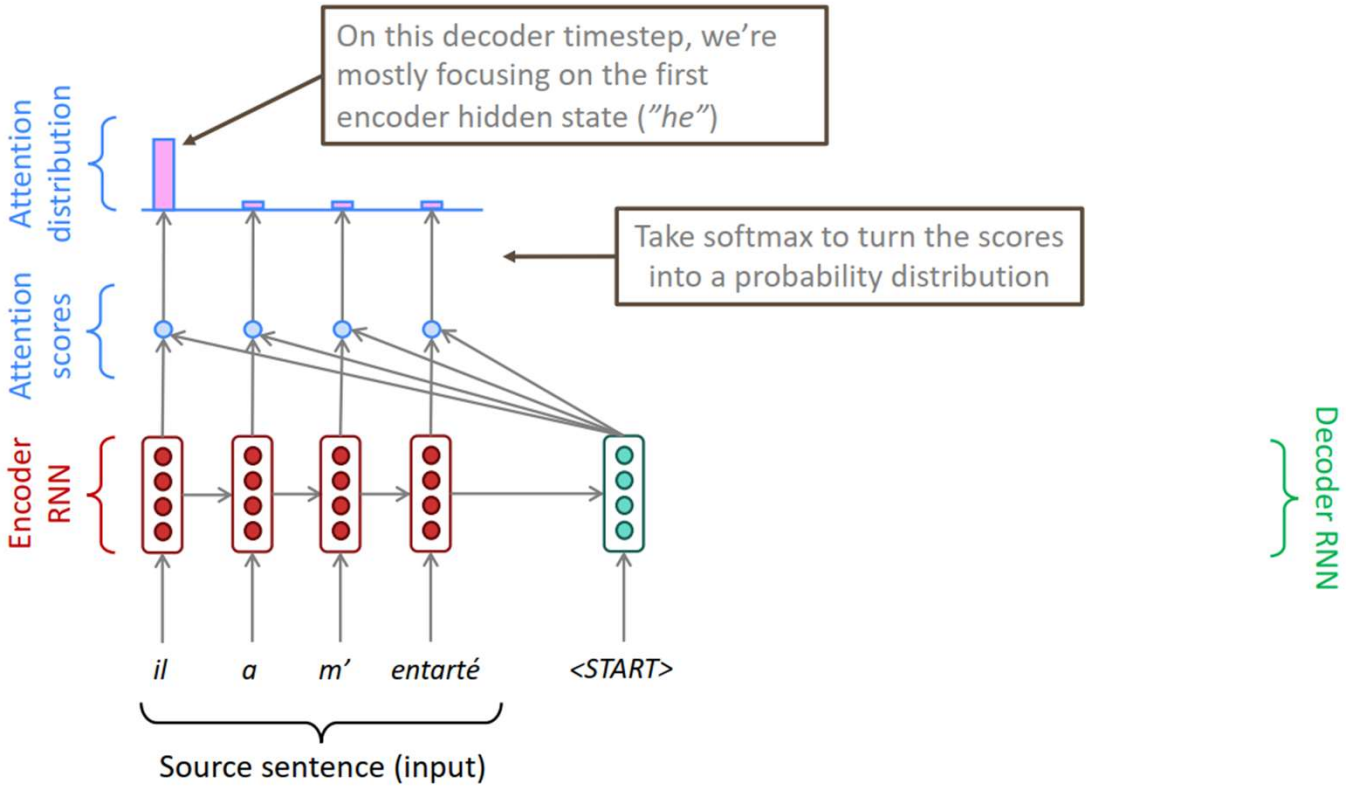
Seq2seq with attention



Decoder RNN

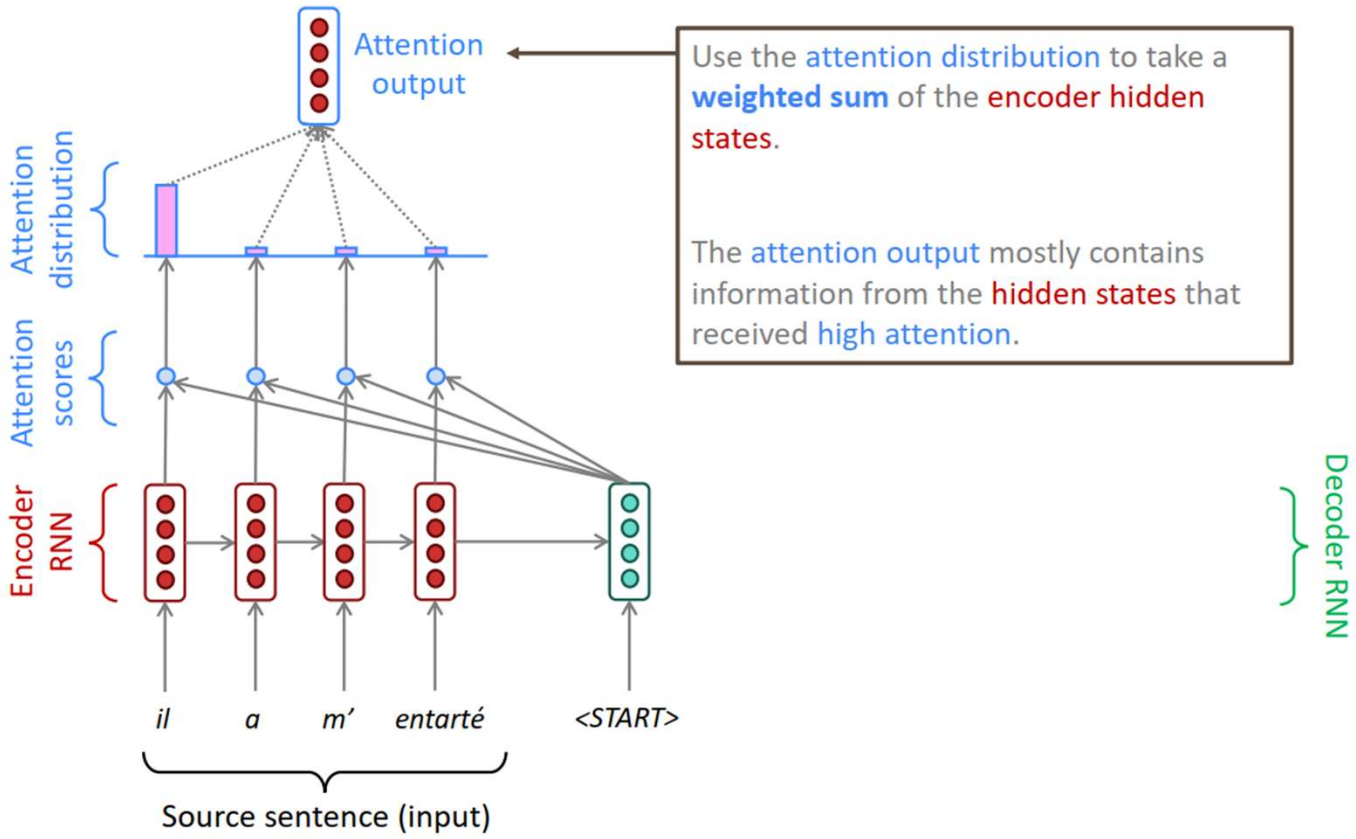
Slide from Stanford's CS224N

Seq2seq with attention



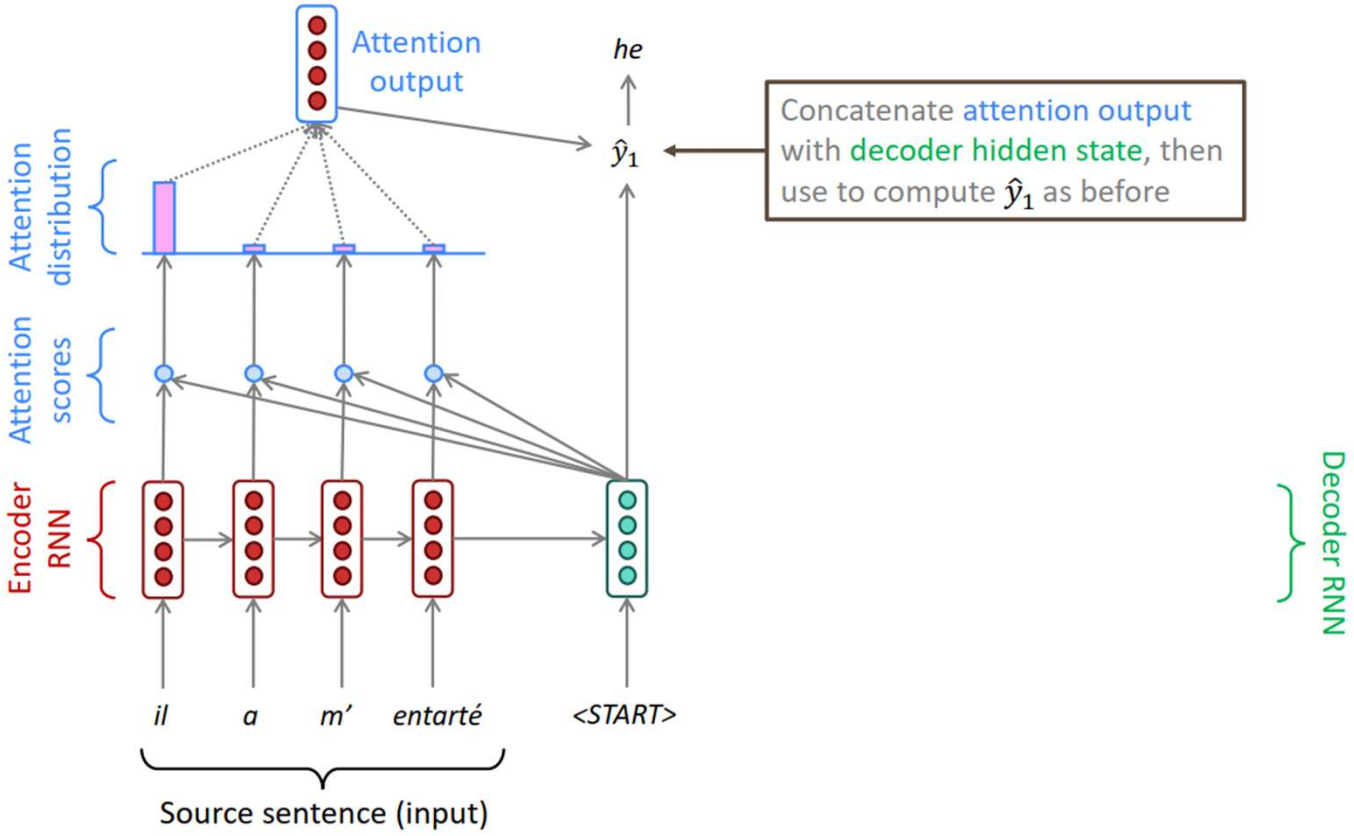
Slide from Stanford's CS224N

Seq2seq with attention



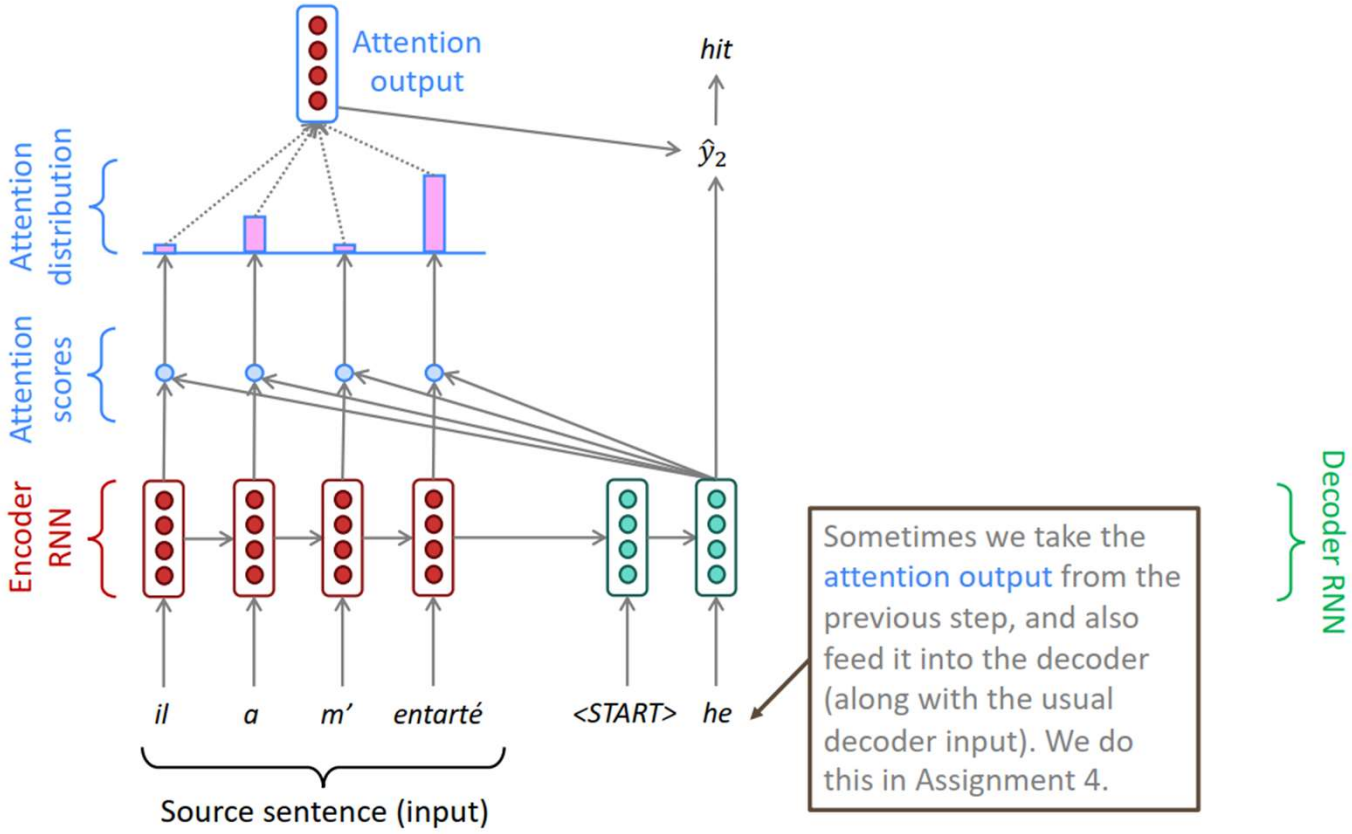
Slide from Stanford's CS224N

Seq2seq with attention



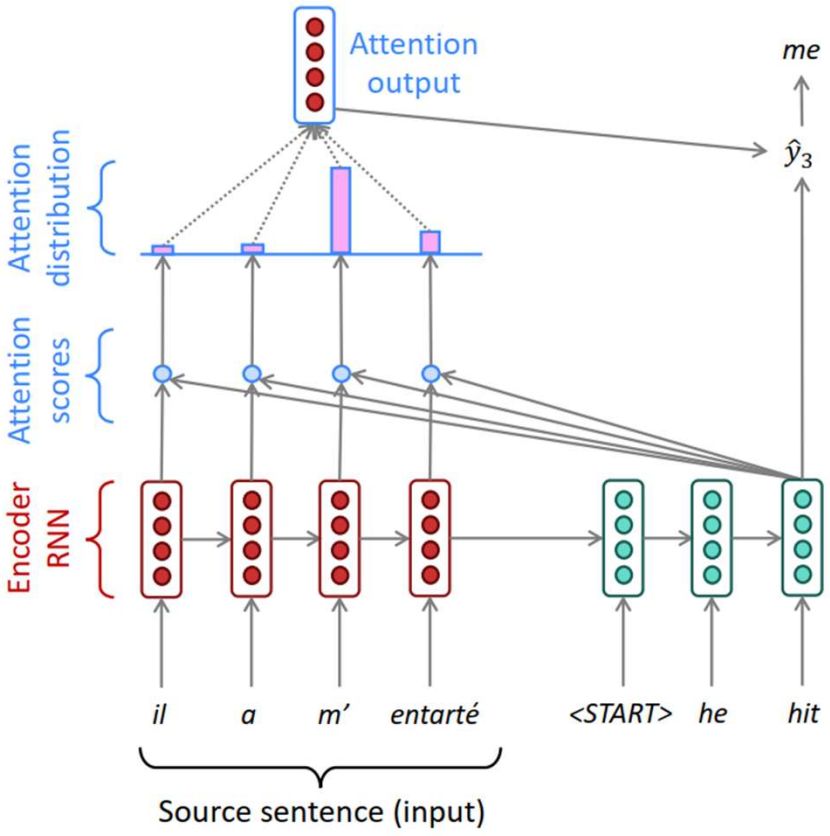
Slide from Stanford's CS224N

Seq2seq with attention



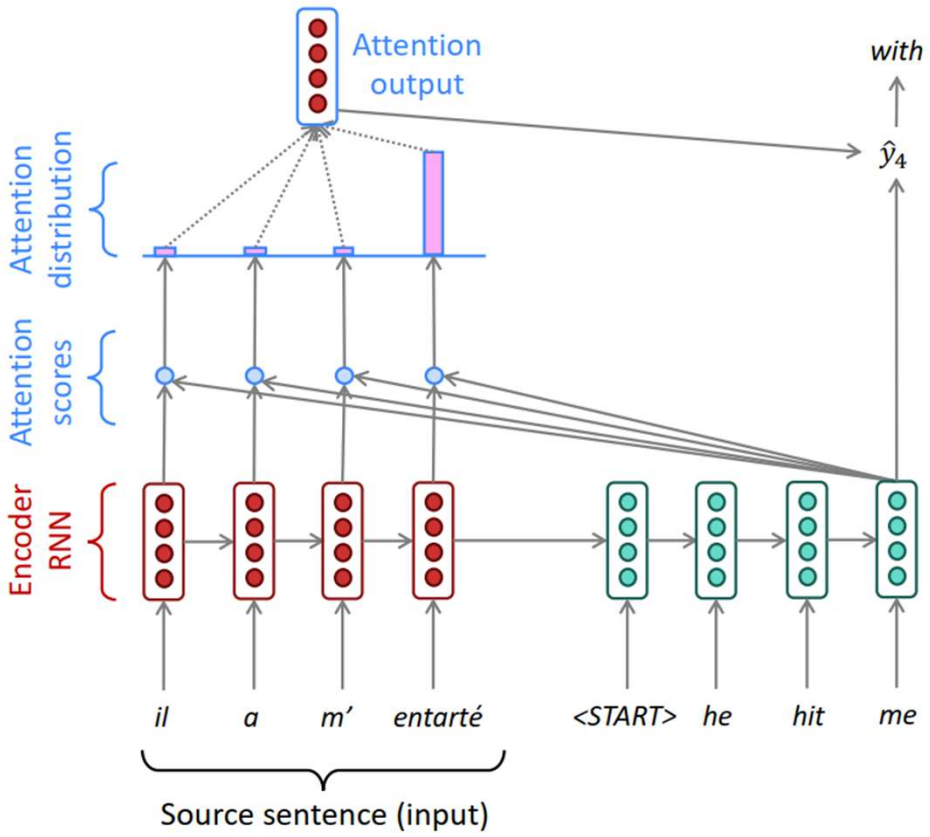
Slide from Stanford's CS224N

Seq2seq with attention



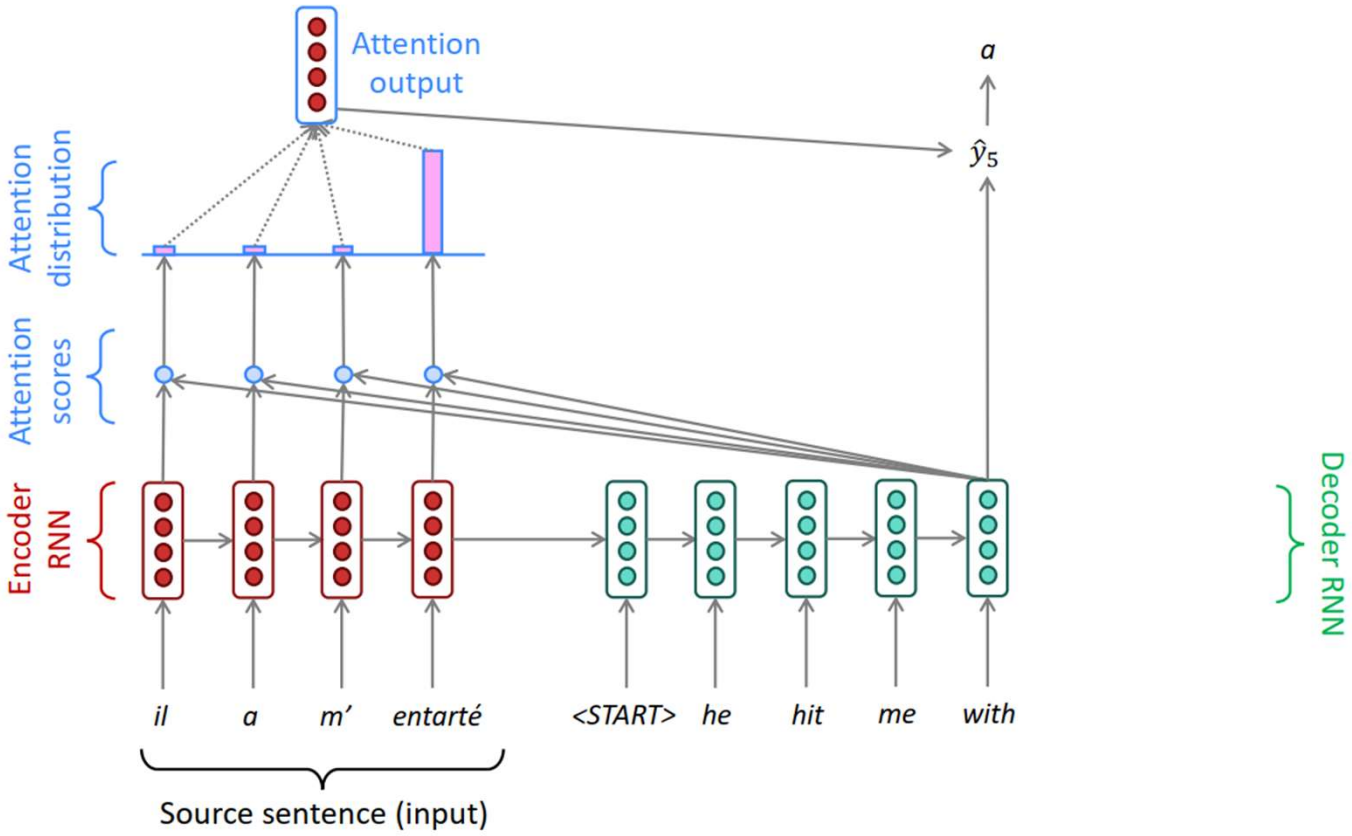
Slide from Stanford's CS224N

Seq2seq with attention



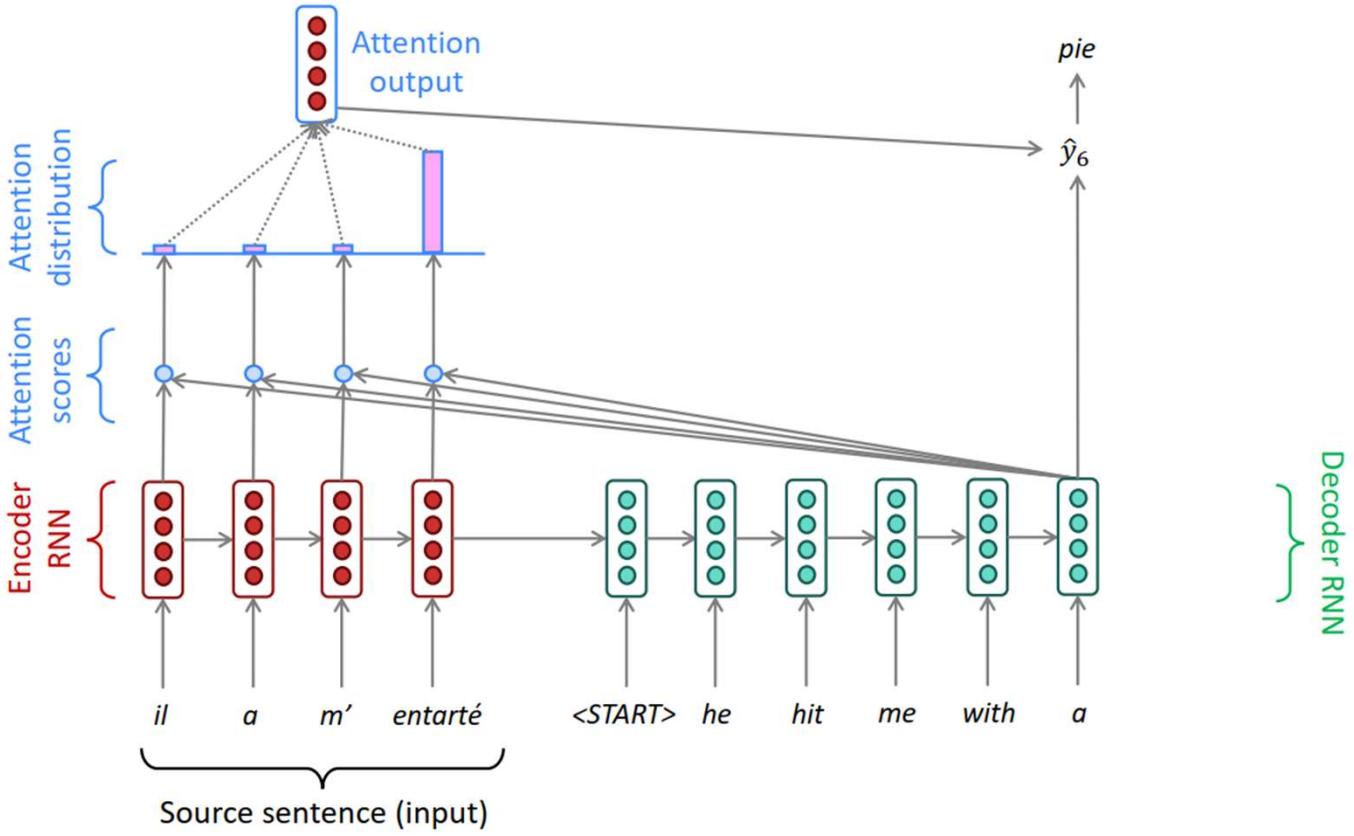
Slide from Stanford's CS224N

Seq2seq with attention



Slide from Stanford's CS224N

Seq2seq with attention



Slide from Stanford's CS224N

Attention in equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^h$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^h$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}^t$

$$\boldsymbol{a}^t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output with the decoder hidden state and proceed as in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^h$$

Slide from Stanford's CS224N

Attention

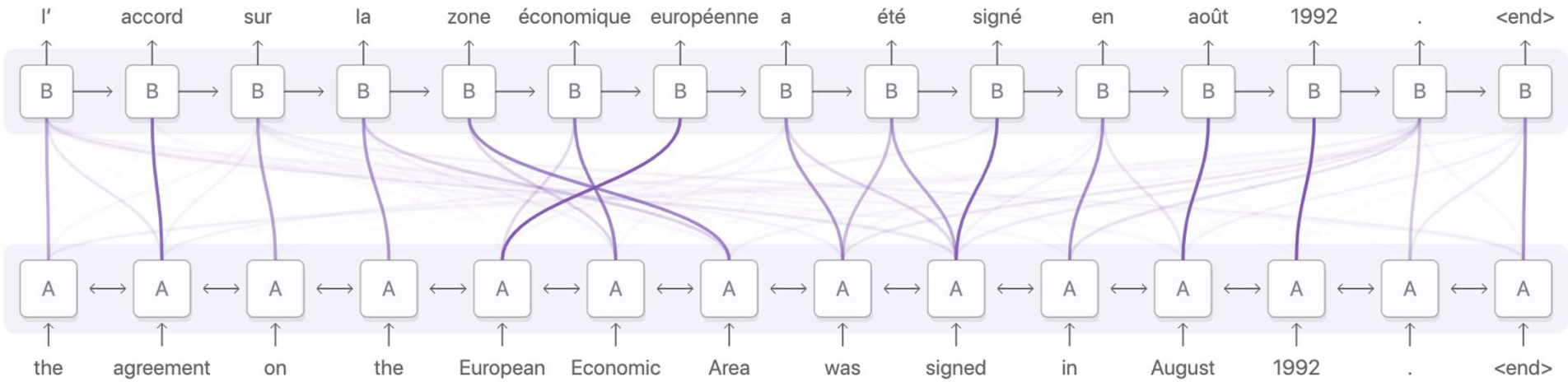
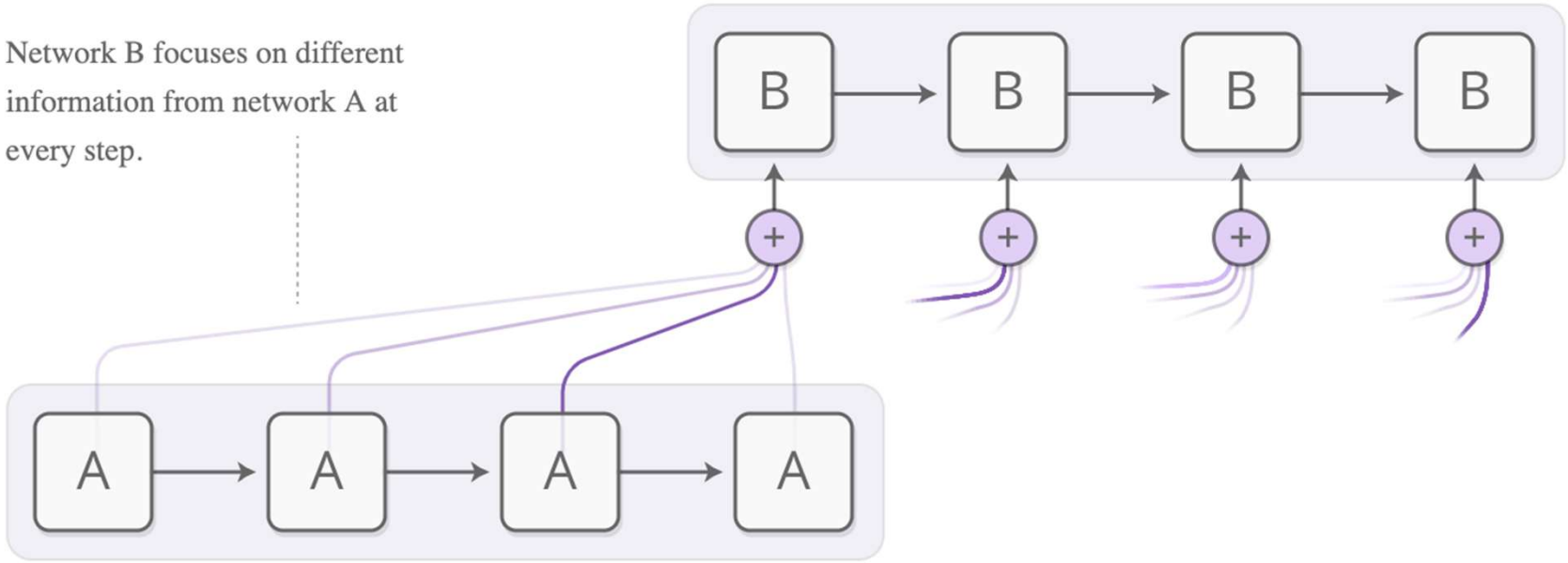


Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

https://distill.pub/2016/augmented-rnns/

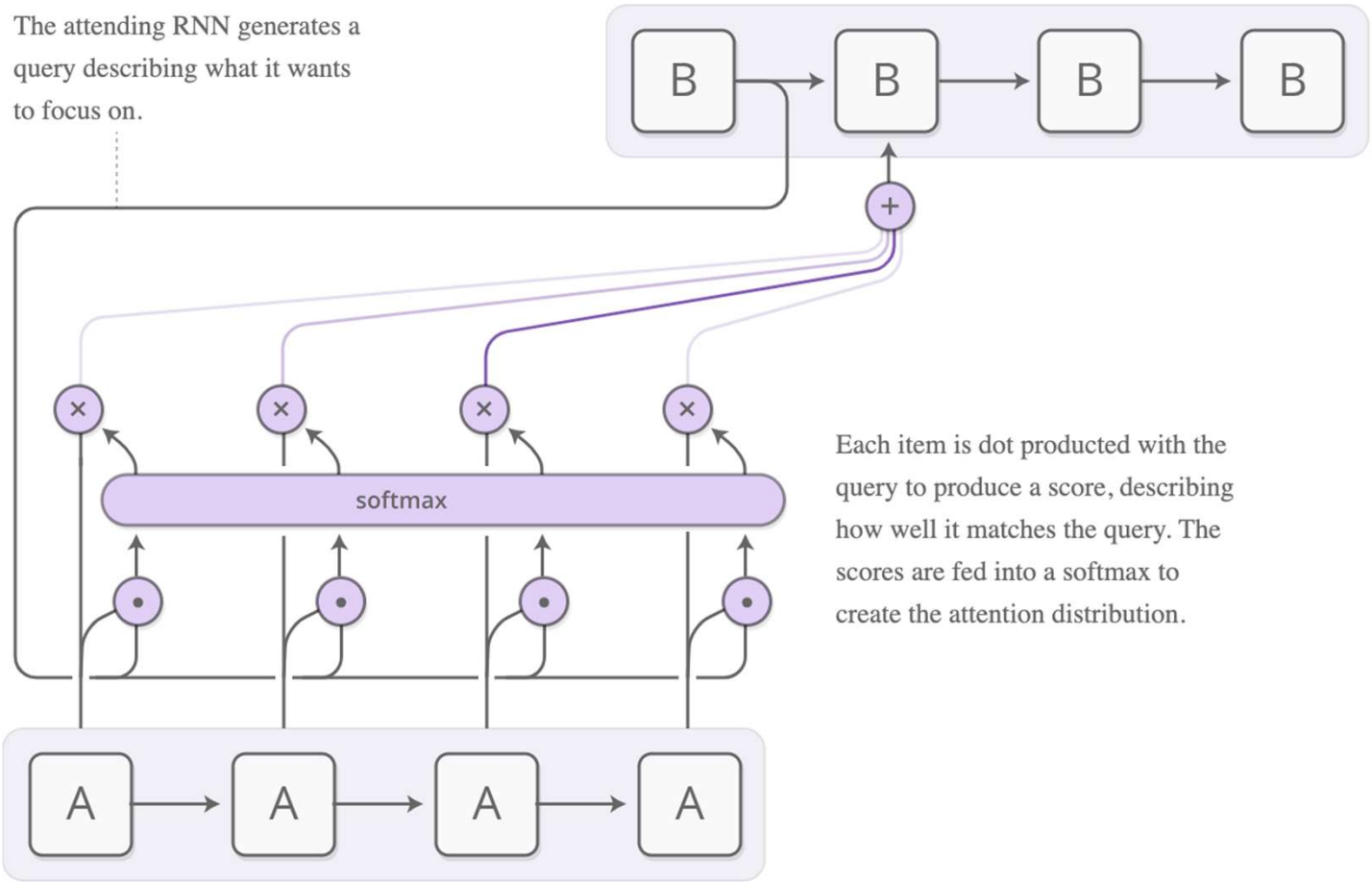
Attention

Idea: while generation focus on part of the information given

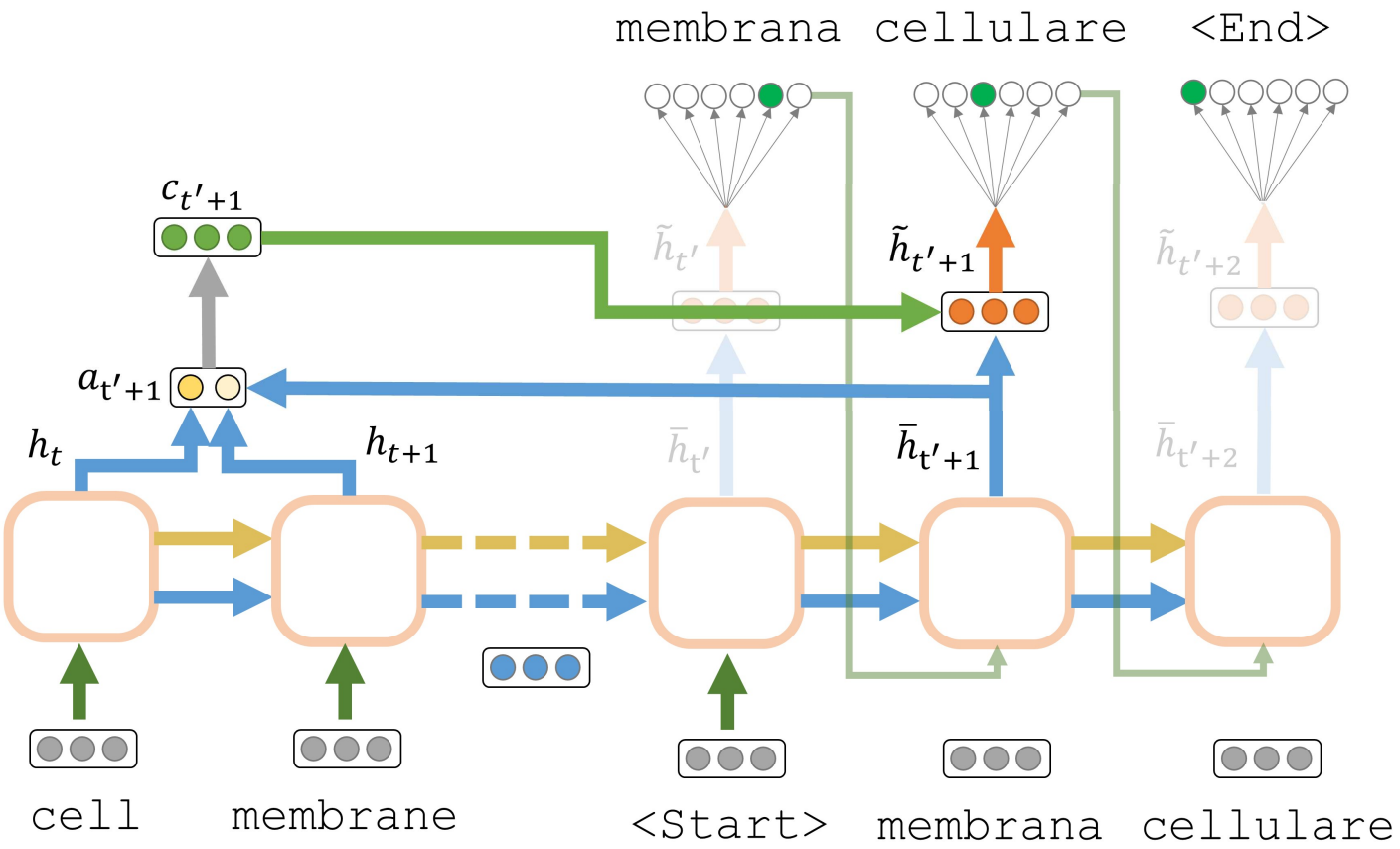


https://distill.pub/2016/augmented-rnns/

Attention


The attending RNN generates a query describing what it wants to focus on.
B
B
B
B
+
×
×
×
×
softmax
Each item is dot producted with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.
A
A
A
A

Attention


membrana
cellulare
<End>
$c_{t'+1}$
$\tilde{h}_{t'}$
$\tilde{h}_{t'+1}$
$\tilde{h}_{t'+2}$
$a_{t'+1}$
h_t
h_{t+1}
$\bar{h}_{t'}$
$\bar{h}_{t'+1}$
$\bar{h}_{t'+2}$
cell
membrane
<Start>
membrana
cellulare

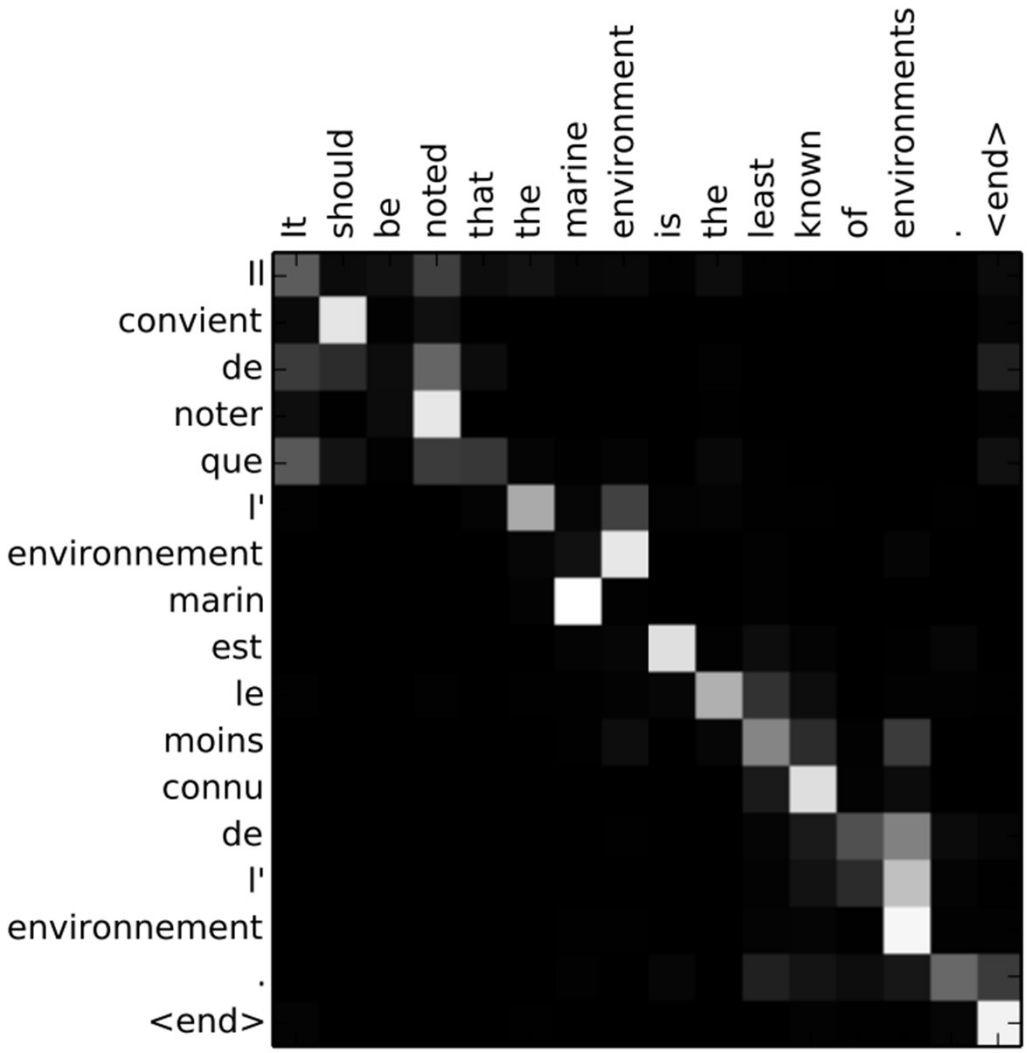
Attention: advantages

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides more "human-like" model of the MT process
  - You can look back at source sentence while translating, rather than needing to remember it all
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with the vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself

Slide from Stanford's CS224N

Attention

Inherent alignment



Dzmitry Bahdanau et al, 2014

Attention variants

- Basic dot-product attention: $e_i = s^T h_i \in \mathbb{R}$
  - Note: this assumes $d_1 = d_2$. This is the version we saw earlier.
- Multiplicative attention: $e_i = s^T W h_i \in \mathbb{R}$
  - Where $W \in \mathbb{R}^{d_1 \times d_2}$ is a weight matrix. Perhaps better called "bilinear attention"
- Reduced-rank multiplicative attention: $e_i = s^T (U^T V) h_i = (U_s)^T (V h_i)$
  - For low rank matrices $U \in R^{k \times d_2},\ V \in R^{k \times d_1},\ k \ll d_1, d_2$
- Additive attention: $e_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R}$
  - Where $W_1 \in \mathbb{R}^{d_3 \times d_1}$ and $W_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $v \in \mathbb{R}^{d_3}$ is a weight vector.
  - d_3 (the attention dimensionality) is a hyperparameter
  - "Additive" is a weird/bad name. It's really using a feed-forward neural net layer.

Slide from Stanford's CS224N

Attention is a general Deep Learning technique

- However: You can use attention in many architectures
  - (not just seq2seq) and many tasks (not just Seq2seq)

- More general definition of attention:
  - Given a set of vector **values**, and a vector **query**, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query *attends* to the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)

Slide from Stanford's CS224N

Attention is a general Deep Learning technique

Given a set of vector **values**, and a vector **query**, attention is a technique to compute a weighted sum of the values, dependent on the query

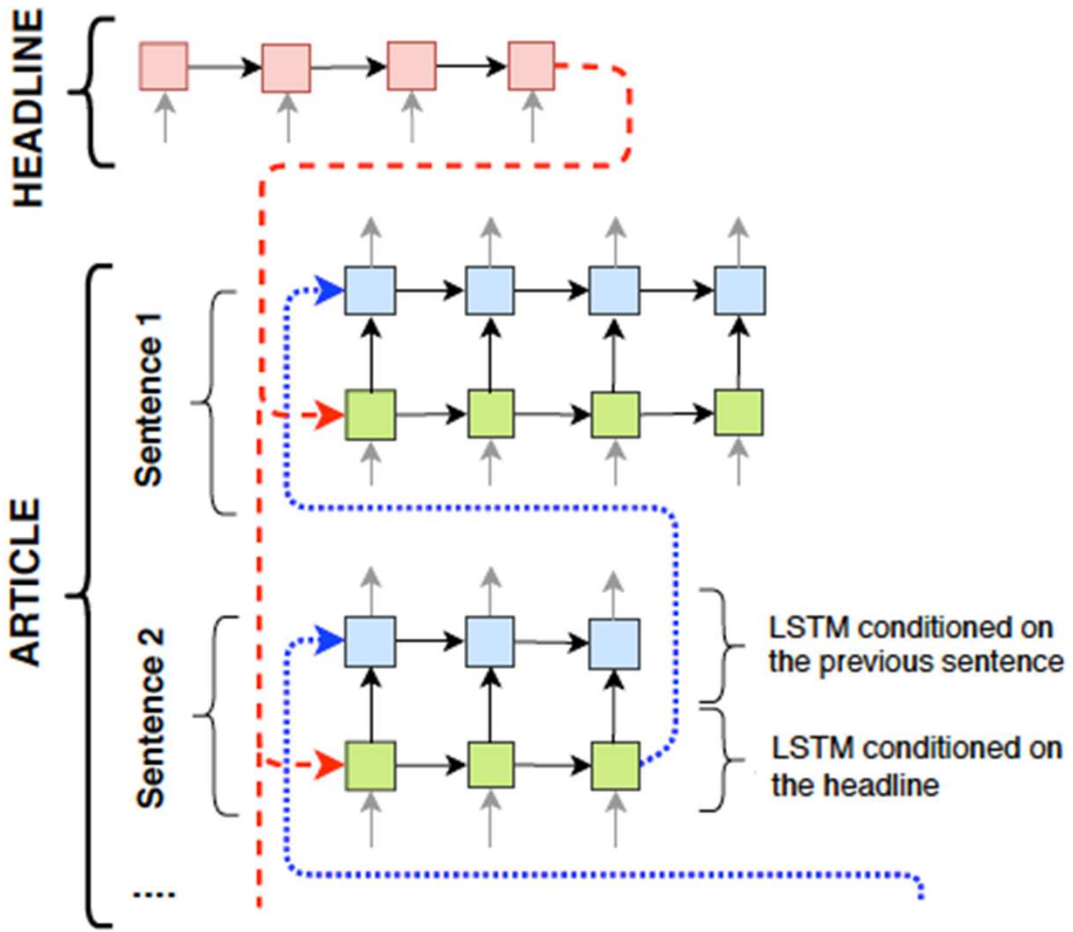
- Intuition:
  - The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.
  - Attention is a way to obtain a fixed-size representation of an arbitrary set of representations (the values), dependent on some other representation (the query).

- Upshot:
  - Attention has become the powerful, flexible, general way pointer and memory manipulation in all deep learning models. A new idea from after 2010! From NMT!

Slide from Stanford's CS224N

Attention

Veracity detection



Attention

A stop sign is on a road with a mountain in the background.

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

Show, attend, and tell, 2015.

Questions?

RNNs: sequence bucketing

Too much padding can result in reduced performance

Place sequences with similar lengths in the same batch

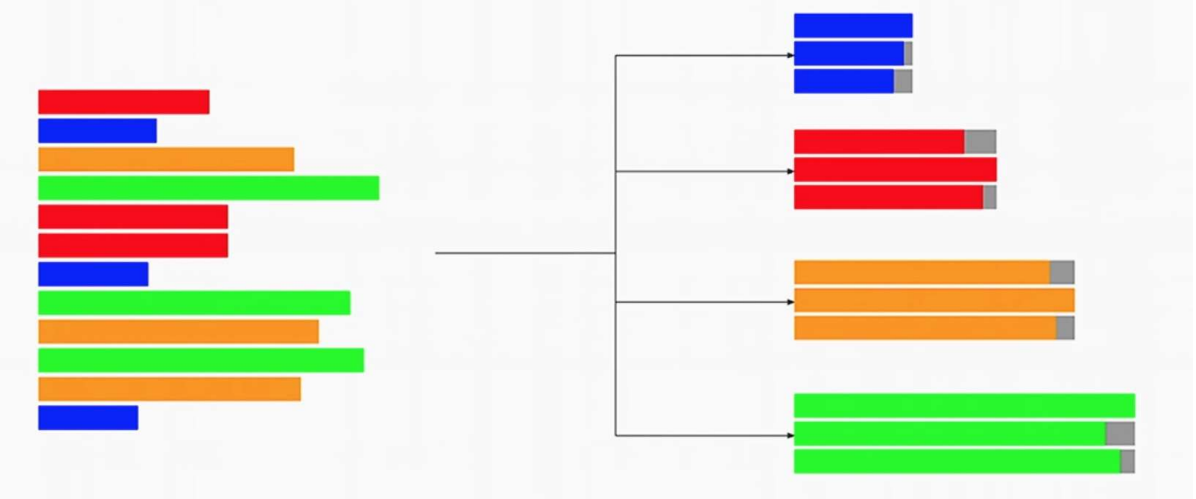


Image source