

# Deep Learning Trends for NLP (2019)

Jay Urbain, PhD

References:

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

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

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

Attention Is All You Need, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

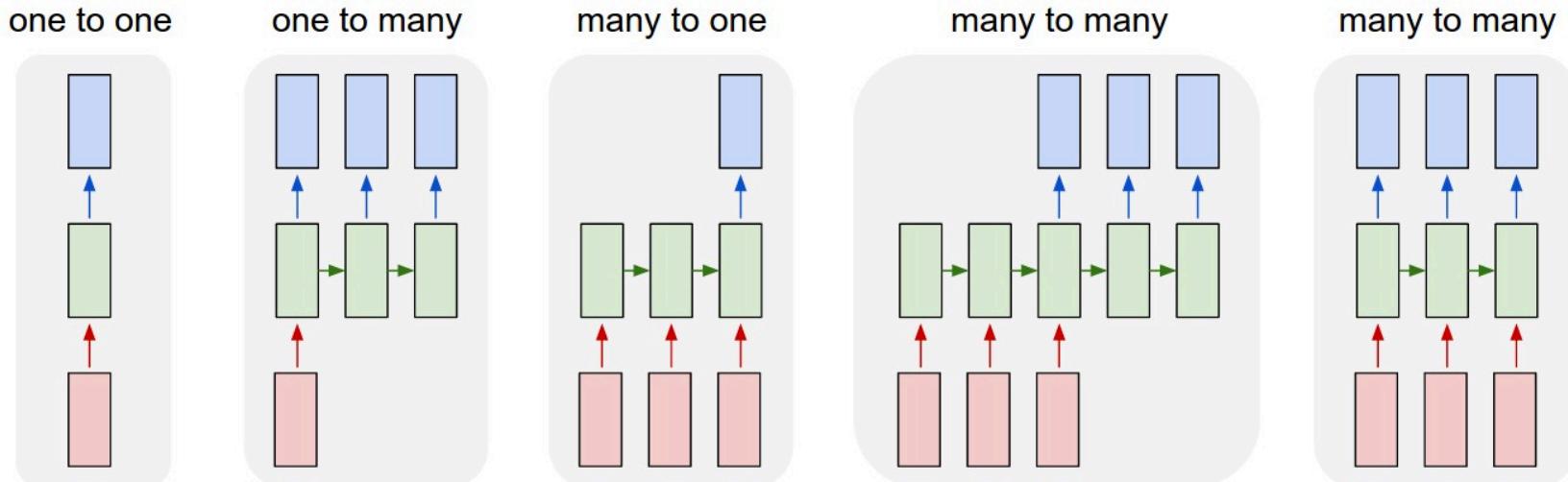
[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding \(2018\)](#) Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

<https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html>

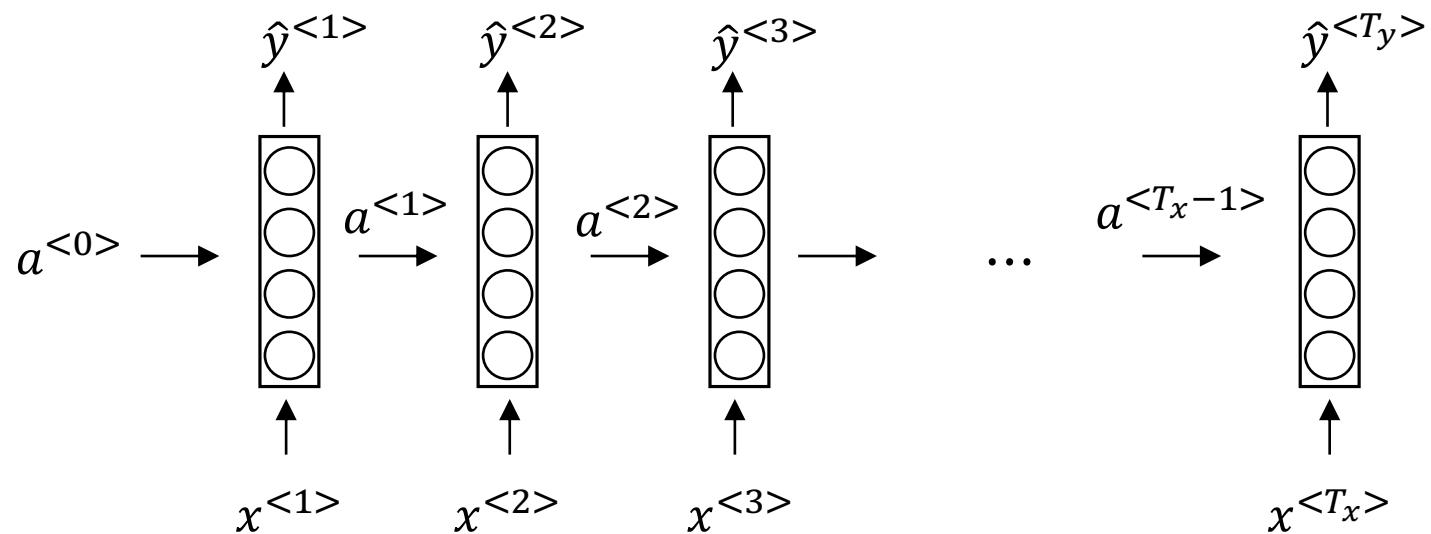
<https://indico.io/blog/sequence-modeling-neural-networks-part2-attention-models/>

# Recurrent Neural Networks

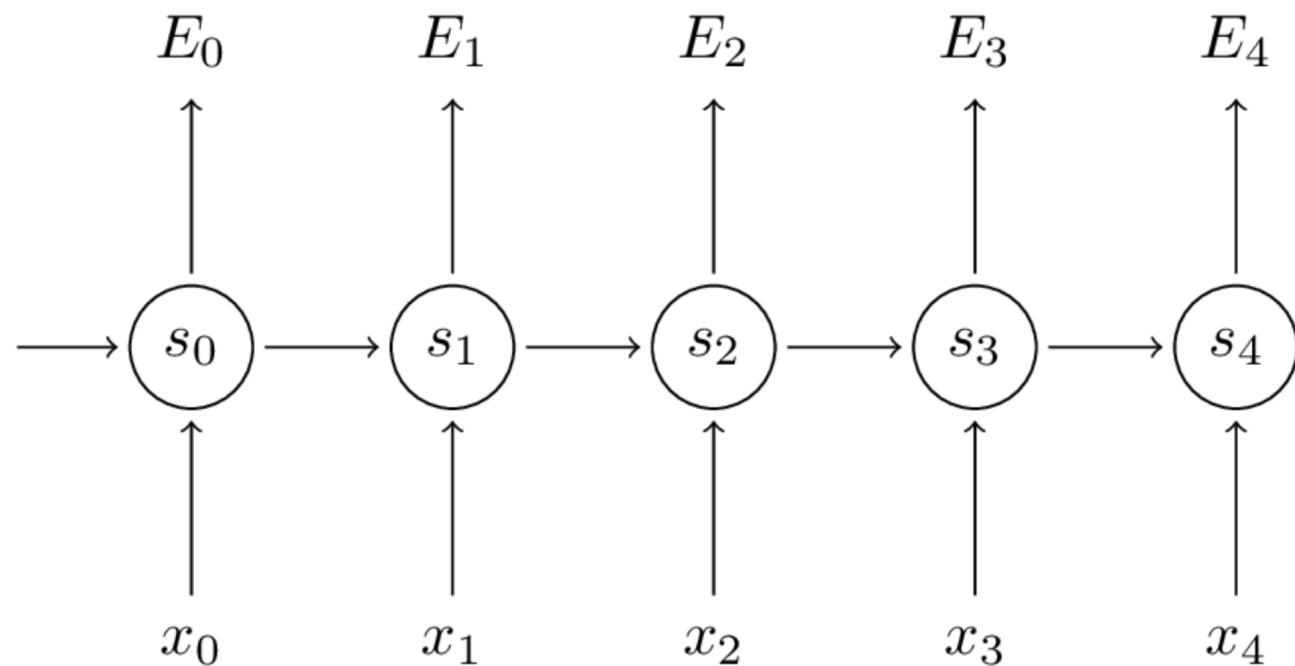
- **Sequences.** Dense Neural Networks, and CNNs only accept a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output (e.g. probabilities of different classes).
- RNN's can operate over Sequences in the input, the output, or in the most general case both.



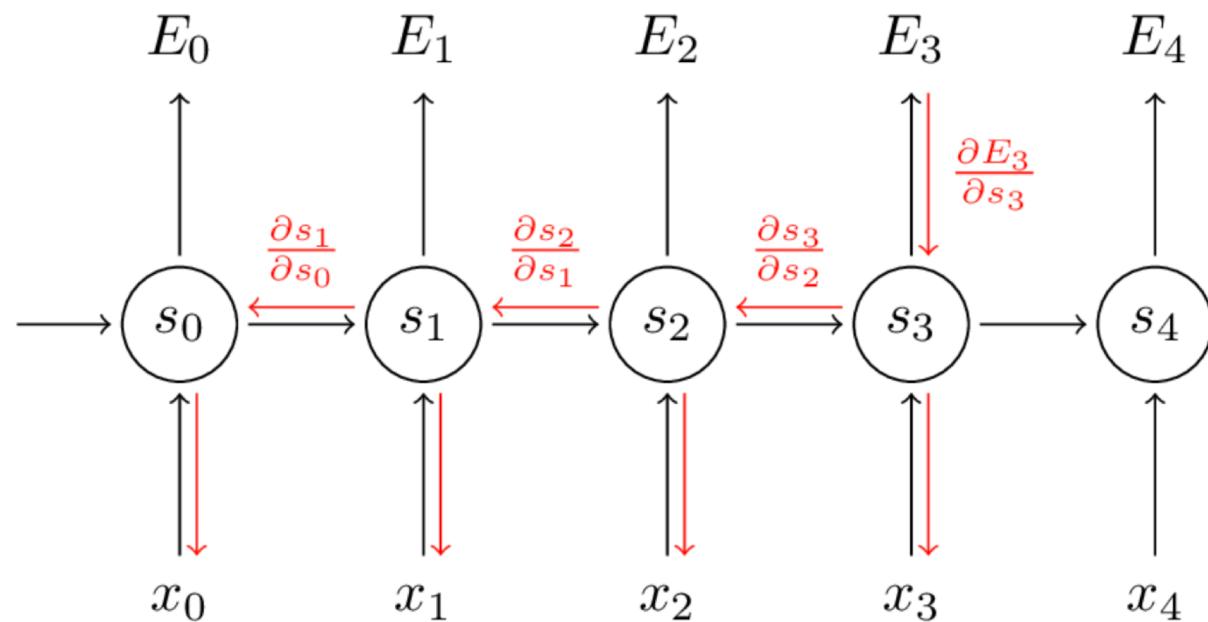
RNN's combine current input with prior state



# Forward Prop



# Back Prop



# RNN forward propagation computation

- RNN's accept an input vector  $x$  and give you an output vector  $y$ .
- The output vector's contents are influenced by the input fed in, and the entire history of inputs you've fed in in the past.
- The RNN class has some internal state  $h$  that it gets to update every time step is called.
- This RNN's parameters are the three matrices  $W_{hh}$ ,  $W_{xh}$ ,  $W_{hy}$ .
- The hidden state **self.h** is initialized with the zero vector.
- There are two terms inside of the tanh:
  - one is based on the previous hidden state
  - one is based on the current input.

```
# forward pass
class RNN:
    # ...
    def step(self, x):
        # update the hidden state
        self.h =
            np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
        # compute the output vector
        y = np.dot(self.W_hy, self.h)
        return y

rnn = RNN()
# x is an input vector, y is the RNN's output vector
y = rnn.step(x)
```

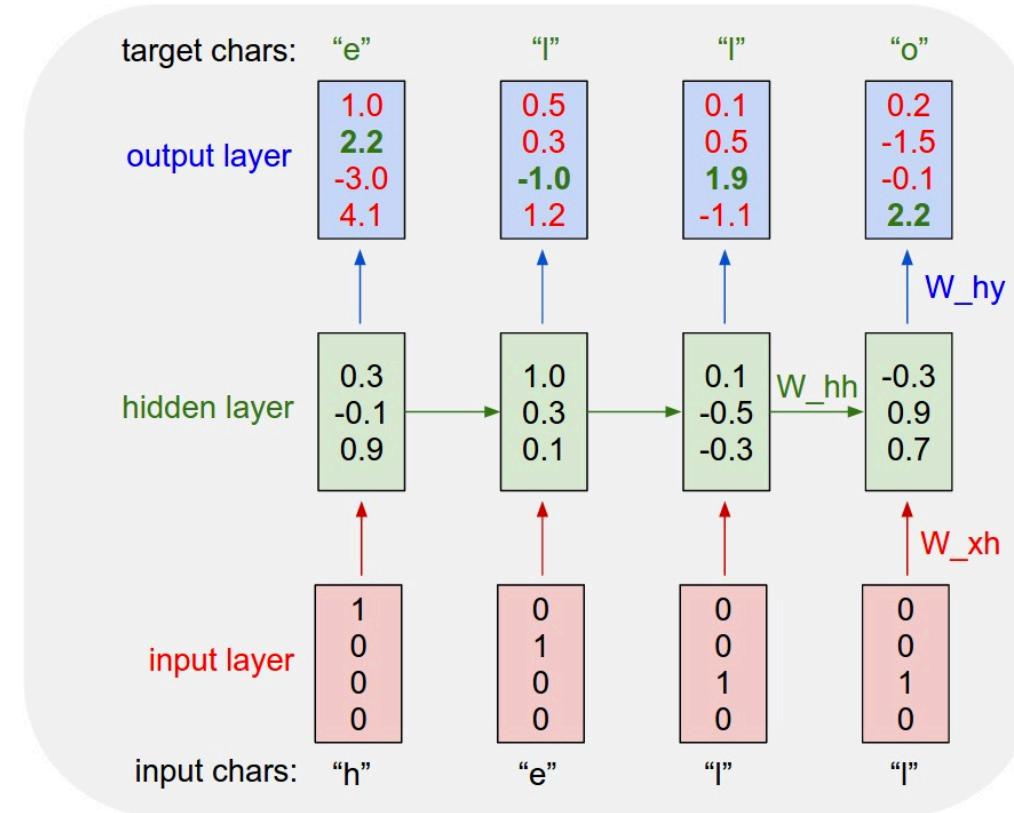
# Building deeper layers

- RNNs are neural networks and everything works monotonically better (if done right) if you start stacking models up like pancakes.
- For instance, we can form a 2-layer recurrent network as follows:

```
y1 = rnn1.step(x) y = rnn2.step(y1)
```

# Character-Level Language Models

- Give the RNN a huge chunk of text and ask it to *model the probability distribution* of the next character in the sequence given a sequence of previous characters.
- This will then allow the RNN to generate new text one character at a time.
- The probability of “e” should be likely given the context of “h”, 2. “l” should be likely in the context of “he”, etc.
- Each character is encoded into a vector using 1-hot-encoding encoding.
- Then observe a sequence of 4-dimensional output vectors (one dimension per character), which we interpret as the confidence the RNN currently assigns to each character coming next in the sequence.



# Character-Level Language Models

- At **test time**, feed a character into the RNN and get a distribution over what characters are likely to come next.
- Sample from this distribution, and feed it right back in to get the next letter.
- Repeat this process and you're sampling text! Lets now train an RNN on different datasets and see what happens.

# Sample character generation from tiny Shakespear collection -

- PANDARUS: Alas, I think he shall be come approached and the day When little strain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.
- Second Senator: They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.
- DUKE VINCENTIO: Well, your wit is in the care of side and that.
- Second Lord: They would be ruled after this chamber, and my fair nues begun out of the fact, to be conveyed, Whose noble souls I'll have the heart of the wars.
- Clown: Come, sir, I will make did behold your worship.
- VIOLA: I'll drink it.

notebooks/nlp/CharacterRNN

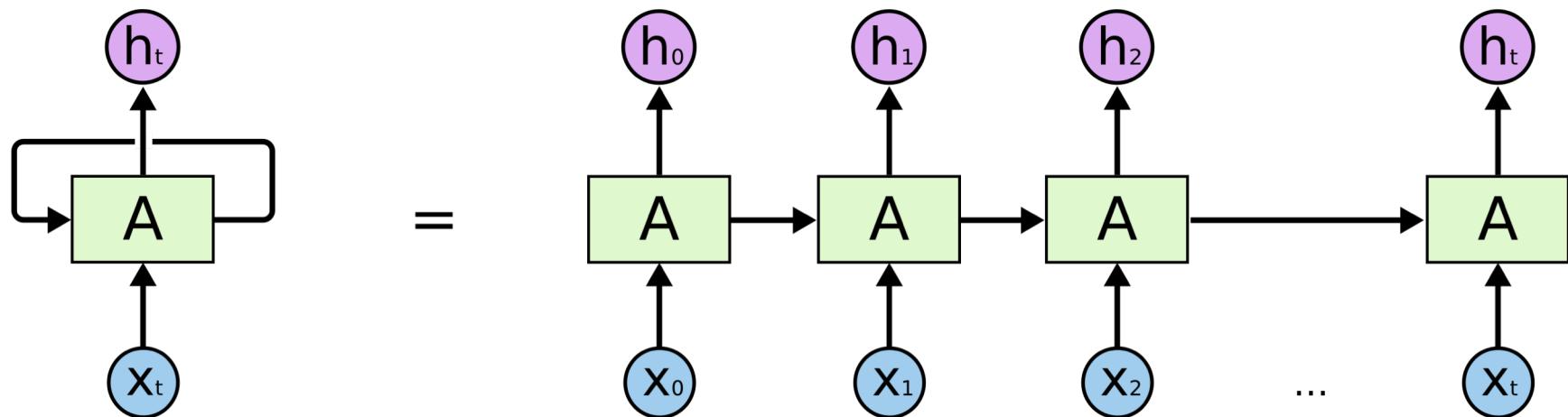
# Visualization of top 5 under each character in sequence

The visualization shows a grid where each row represents a character position in the sequence, and each column represents a candidate character. The colors indicate the probability or frequency of each candidate at that position. The sequence itself is visible along the top and left edges of the grid.

t	t	p	:	/	w	w	w	.y	n	e	t	n	w	s.	c	o	m/	]	E	n	g	l	i	sh	-	l	a	ng	u	g	e		w	e	b	s	t	e		o	f	I	s	r	a	e	l	'	s	l	a	r							
t	p	:	/	/	w	w	w	.b	a	c	a	h	e	t	s.	c	o	m/	]	E	n	g	l	i	sh	-	x	g	l	l	u	g	u	g		w	e	b	s	t	e		o	f	I	s	r	a	e	l	'	s	l	a	r				
d	:x	n	e.	w	a	e	a..	a	w	a	t	o	a.	s	&	n	t	i	a	c	a	-	s	a	r	d	e	l	h	o	a	n	t	b	i	s	a	n	f	a	n	r	e	i	f	'	a	a	t	d									
m	w	-	2	◆	p	i	i	i	o	s	e	s	s	i	s	.	/	e	r	n.	c	]	(	d	c	e	e	n	e	p	e	s	a	a	i	k	i	i	e	e	l	e	d	h,	i	r	t	h	r	a	o	n	s	e	,	c	o	s	e
d	r.	<	:	a	h	b-	n	p	t	w	t.	x	i	g	h	/	m	)	T	v	d	r	y	z	i	c	o	u	e	d	l	s	u	:	t	h	-	o	o	tu,	st	uif	l	v	e	p	er	y											
s	t	p,	t	c	o	a	z	d	r	u	l	w	o	c	l	e	n	s	]	p.	l	l	v	a	o	d,	,,	e	y	t	c-	n	d	m-	o	i	b	u	v	s]	b	b	i	m	s	u	l	t	a	t	l	y	b						

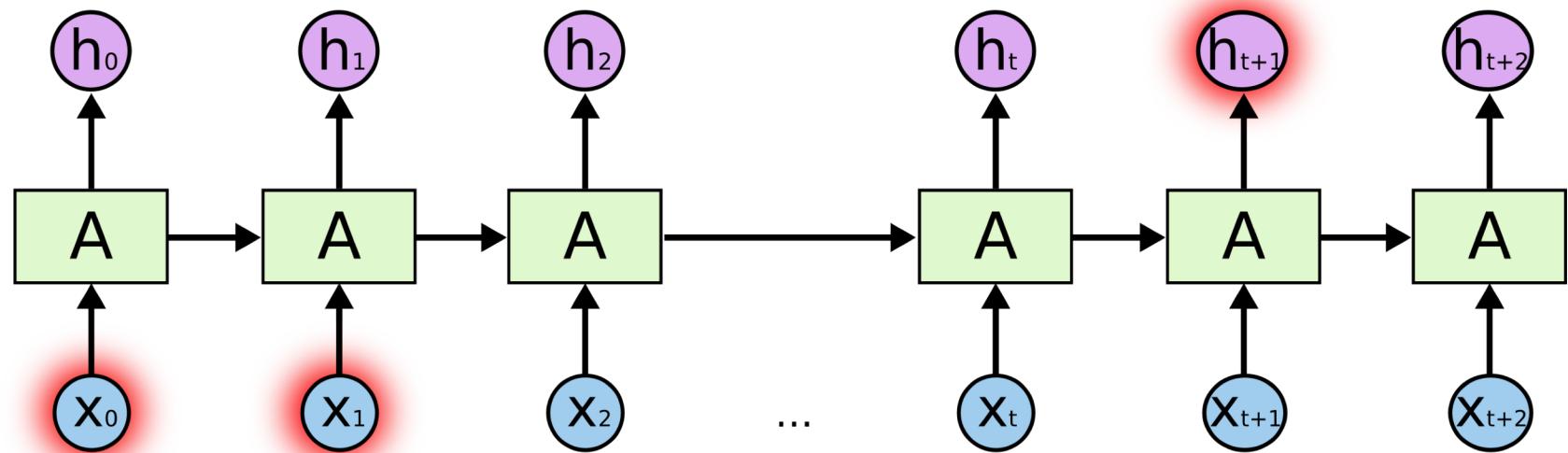
# RNNs

- One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task.

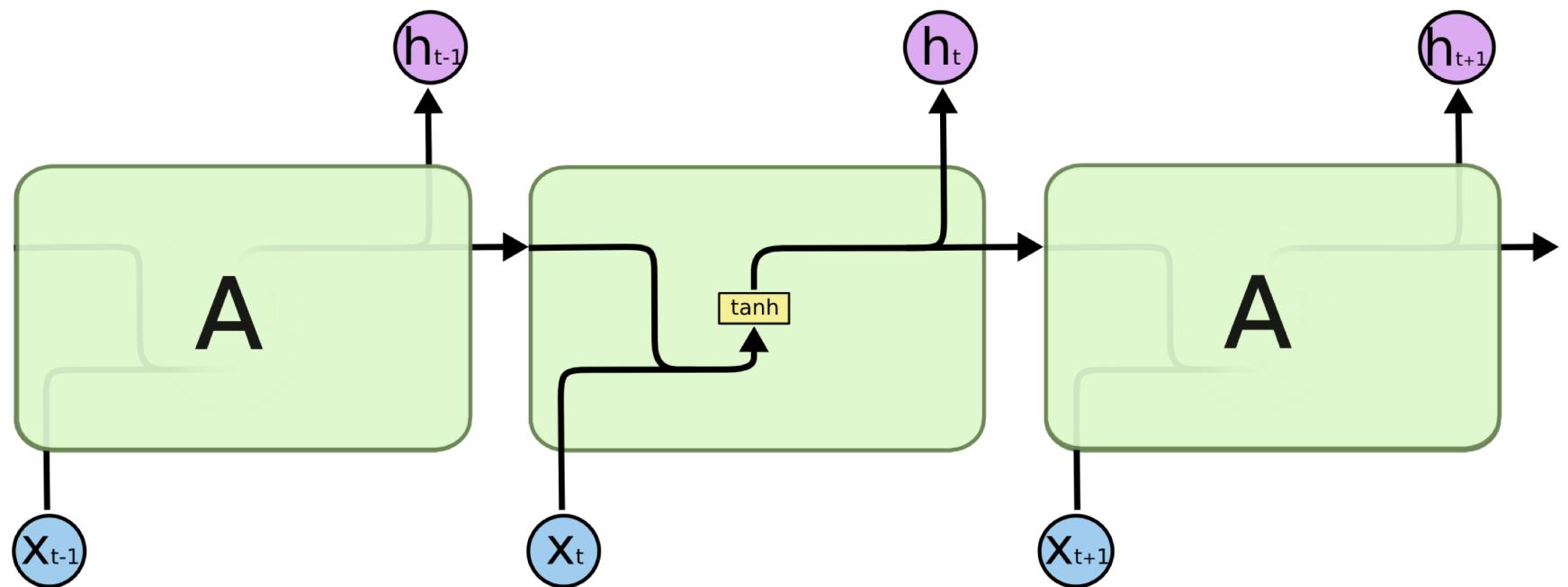


# Problem with long-term dependencies

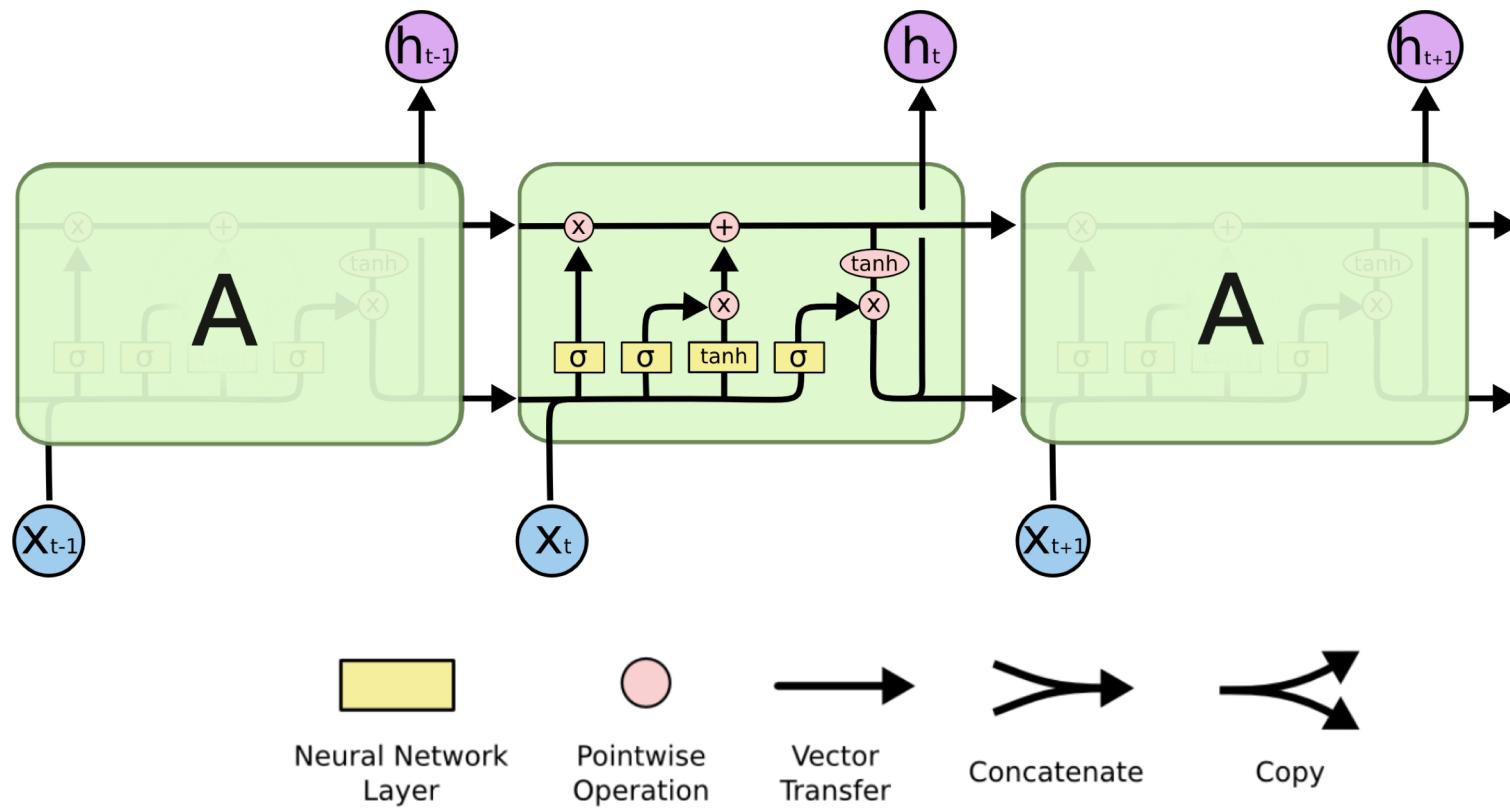
- May need more context.
- Vanishing gradient problem.



The repeating module in a standard RNN contains a single layer

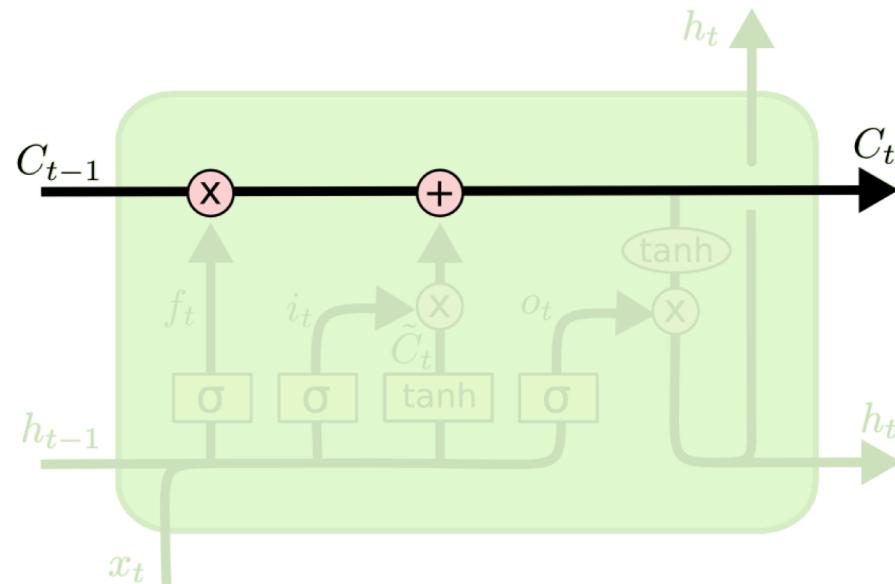


The repeating module in an LSTM contains four interacting layers



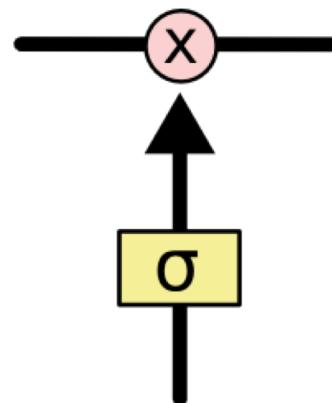
# The Core Idea Behind LSTMs

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The cell state runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



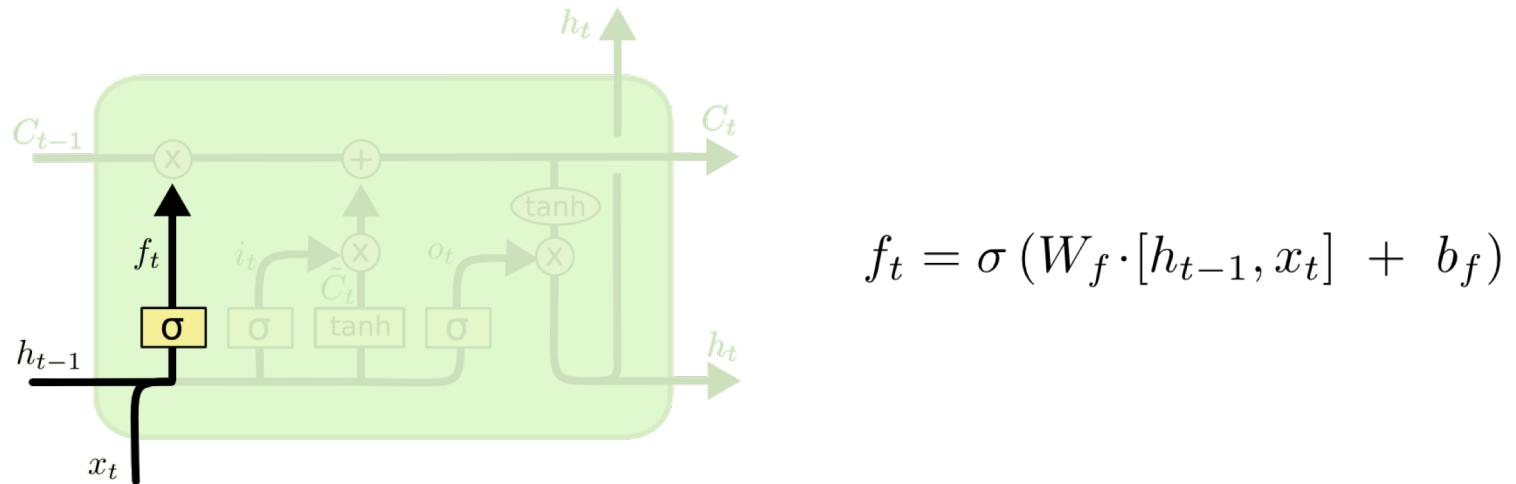
Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.
- A value of zero means “let nothing through,” while a value of one means “let everything through”
- An LSTM has three of these gates, to protect and control the cell state.



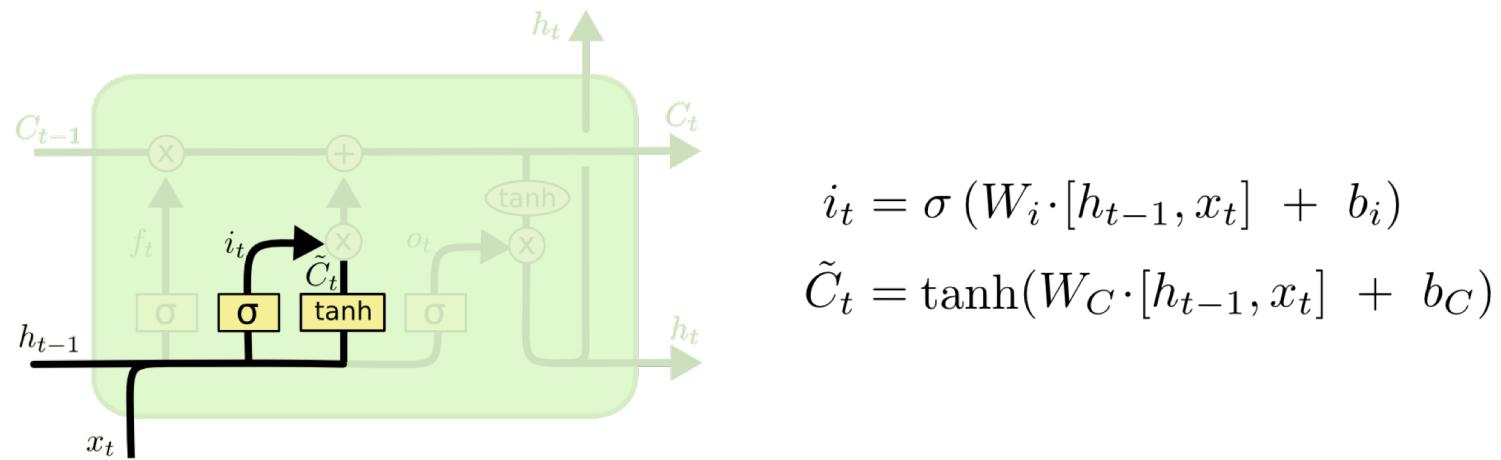
# Forget gate layer

- What information we're going to throw away from the cell state.
- Looks at  $h_{t-1}$  and  $x_t$ , and outputs a number between 0 and 1 for each number in the cell state  $C_{t-1}$ .
- A 1 represents “completely keep this” while a 0 represents “completely get rid of this.”



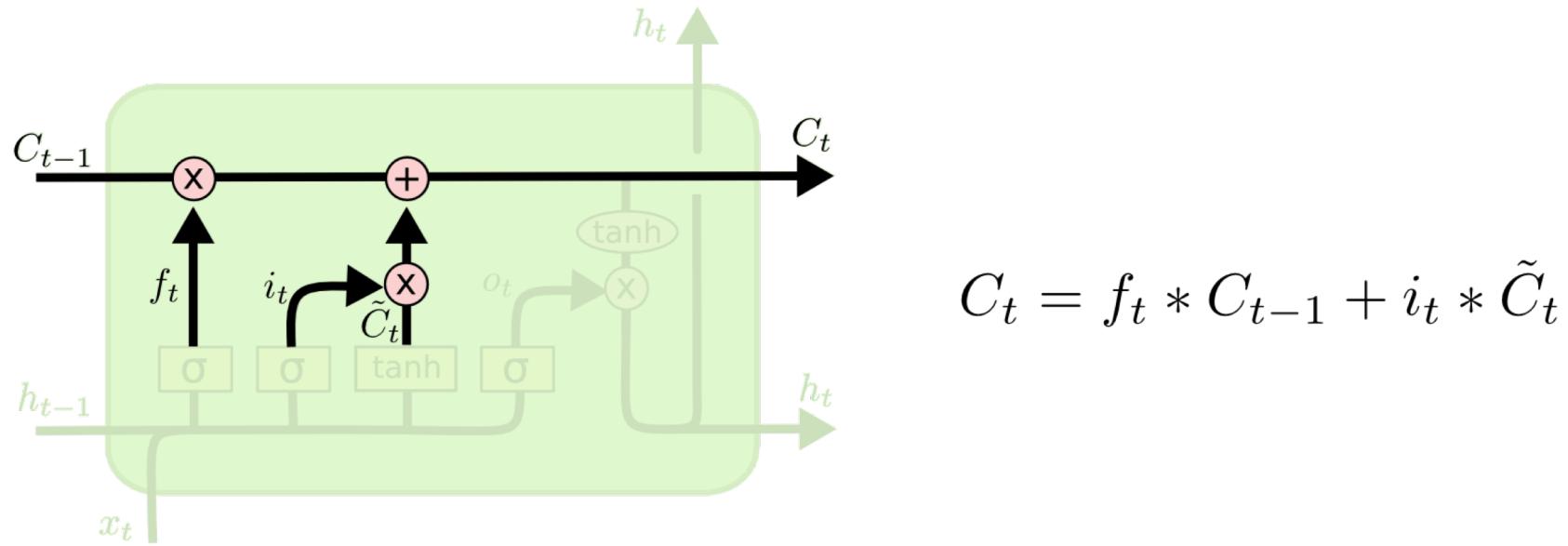
# What information to keep

- First, a sigmoid layer called the “input gate layer” decides which values we’ll update.
- Next, a tanh layer creates a vector of new candidate values,  $\tilde{C}_t$ , that could be added to  $C_t$ .



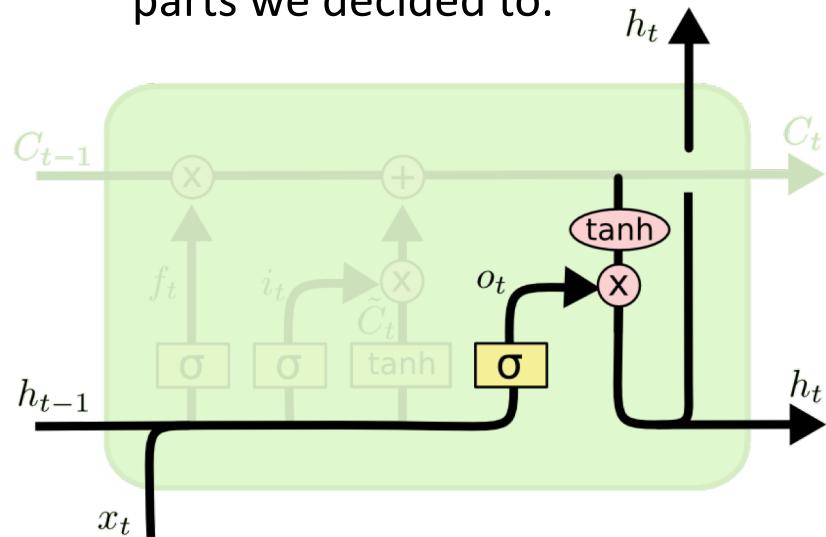
# Update the cell state

- Update the old cell state,  $C_{t-1}$ , into the new cell state  $C_t$ .
- Multiply the old state by  $f_t$ , forgetting the things we decided to forget earlier.
- Then add it to  $i_t * \tilde{C}_t$ . This is the new candidate values, scaled by how much we decided to update each state value.



# What to output

- Output will be based on our cell state.
- First, we run a sigmoid layer which decides what parts of the cell state we're going to output.
- Then, 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.



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

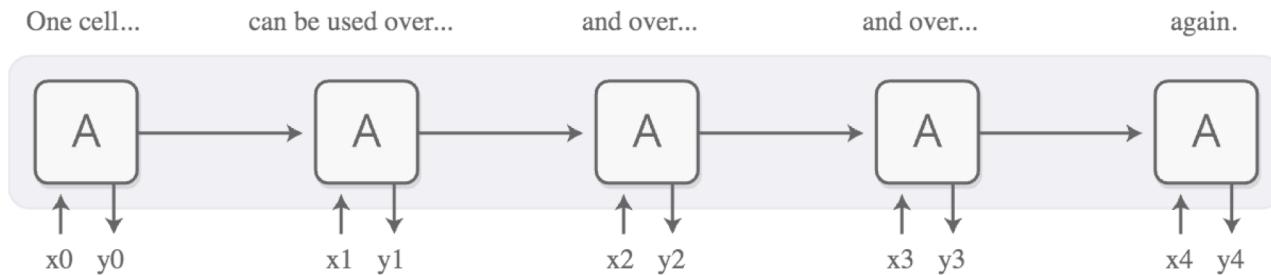
$$h_t = o_t * \tanh (C_t)$$

# Summary LSTMs

- Essentially all of the remarkable results people are achieving with RNNs are achieved using LSTMs.
- They really work a lot better for most tasks

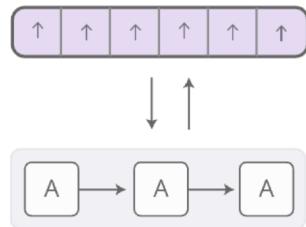
# Attention and Augmented Memory Networks

- RNNs are one of the staples of deep learning, allowing neural networks to work with sequences of data like text, audio and video.
- Convert a sequence down into a high-level understanding, to annotate sequences, and even to generate new sequences from scratch.
- LSTMs can be used on longer sequences.

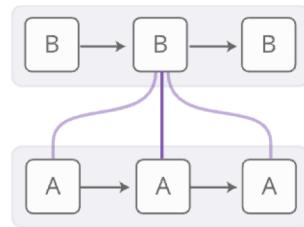


# Future directions for RNNs

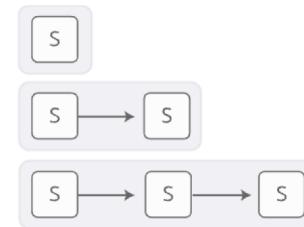
- RNNs have become very widespread in the last few years.
- As this has happened, there's been a growing number of attempts to augment RNNs with new properties.
- Four directions stand out:



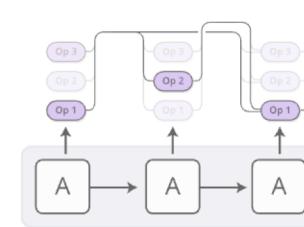
**Neural Turing  
Machines**  
have external memory  
that they can read and  
write to.



**Attentional  
Interfaces**  
allow RNNs to focus on  
parts of their input.



**Adaptive  
Computation Time**  
allows for varying  
amounts of computation  
per step.

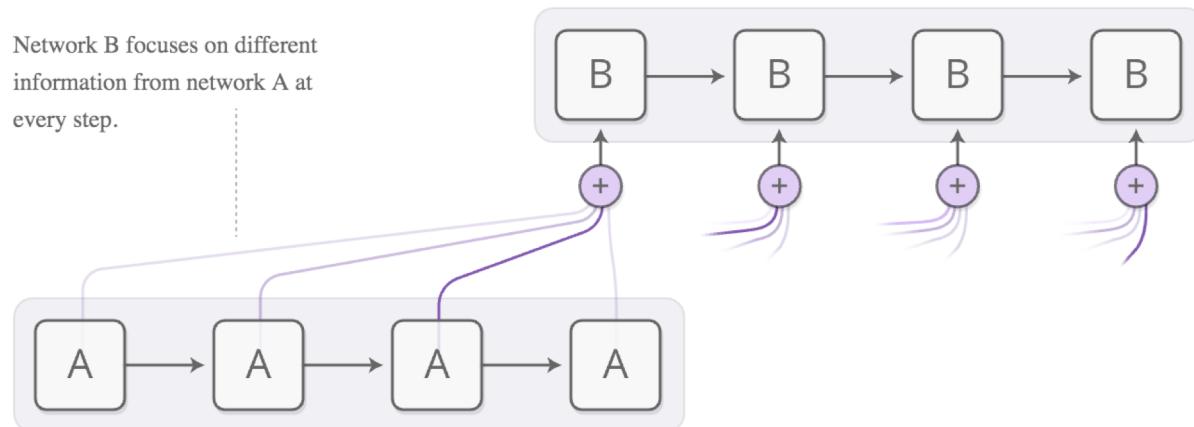


**Neural  
Programmers**  
can call functions,  
building programs as  
they run.

All rely on the same underlying trick—something called attention—to work.

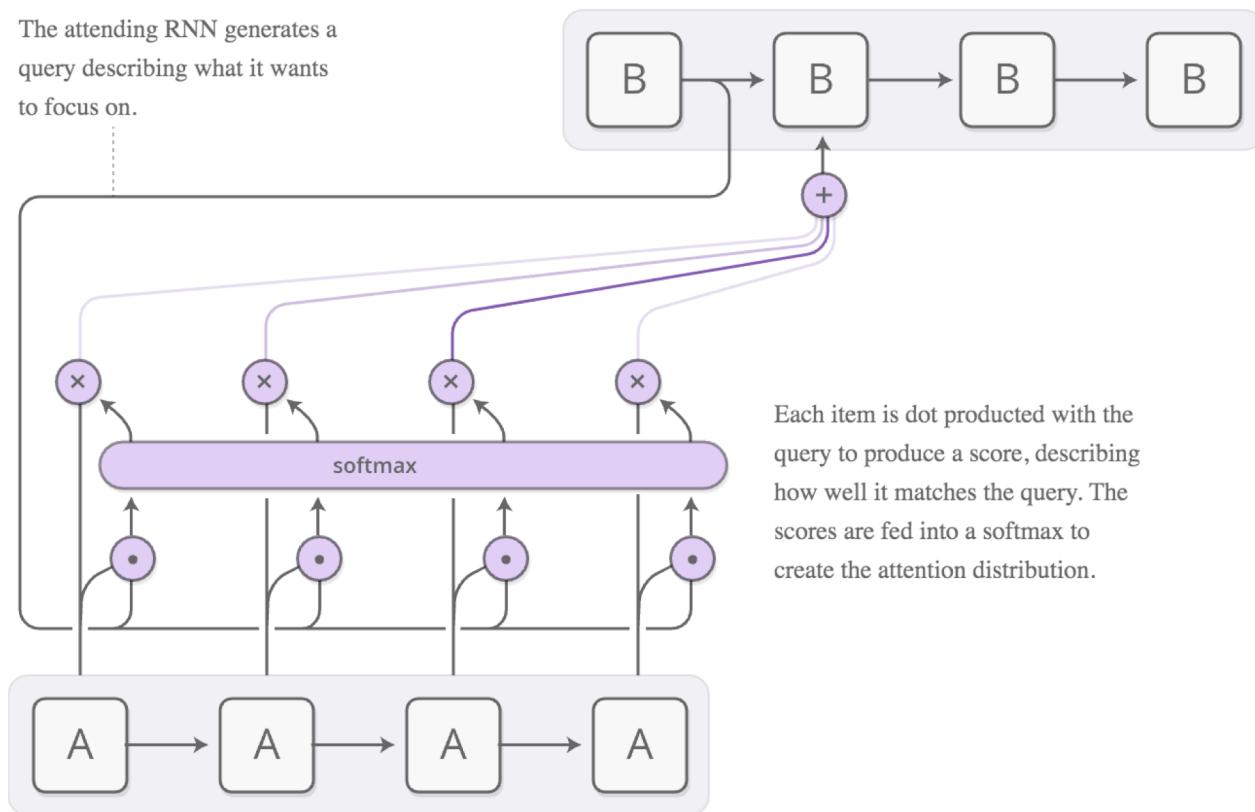
# Attentional Interfaces

- When you read or transcribe a paragraph, there are parts of the paragraph that you “attend” to.
- NNs can achieve this same behavior using *attention*, focusing on part of a subset of the information they’re given.
- E.g., an RNN can attend over the output of another RNN. At every time step, it focuses on different positions in the other RNN.

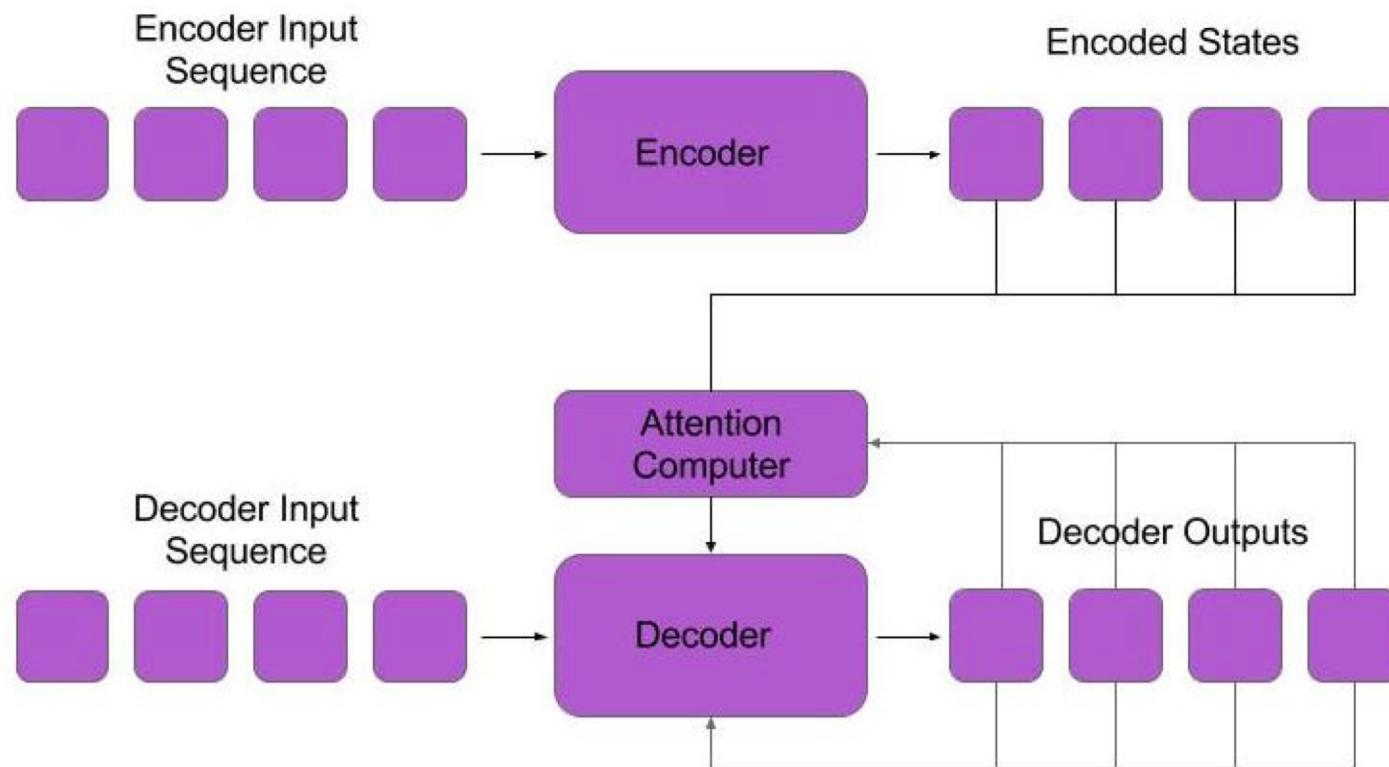


# Attending RNN

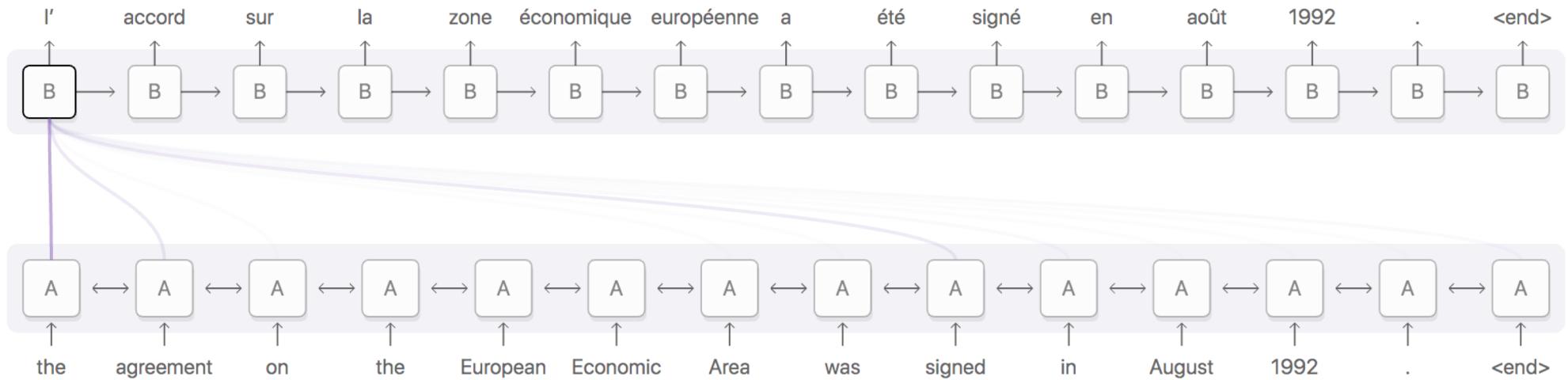
The attending RNN generates a query describing what it wants to focus on.



# Attention Computer



# Attention in language translation encoder-decoder

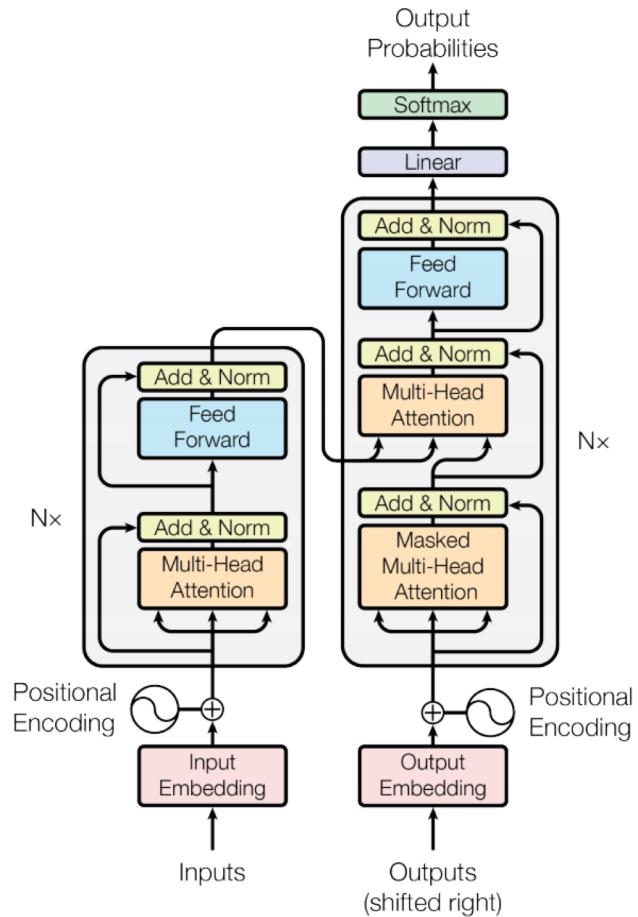


## Attention is all you need: Encoder-decoder Transformer model with attention, self-attention

- Propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.
- Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train.

Example:

<http://54.163.221.189:8080/>



# State-of-the-art: Pretraining

- One of the biggest challenges in NLP is the shortage of training data.
- Because NLP is a diversified field with many distinct tasks, most task-specific datasets contain only a few thousand or a few hundred thousand human-labeled training examples.
- Modern deep learning-based NLP models see benefits from much larger amounts of data, improving when trained on millions, or *billions*, of annotated training examples.
- To help close this gap in data, researchers have developed a variety of techniques for training general purpose language representation models using the enormous amount of unannotated text on the web.
- Known as *pre-training*.
- The pre-trained model can then be fine-tuned on small-data NLP tasks like question answering and sentiment analysis, resulting in substantial accuracy improvements compared to training on these datasets from scratch.

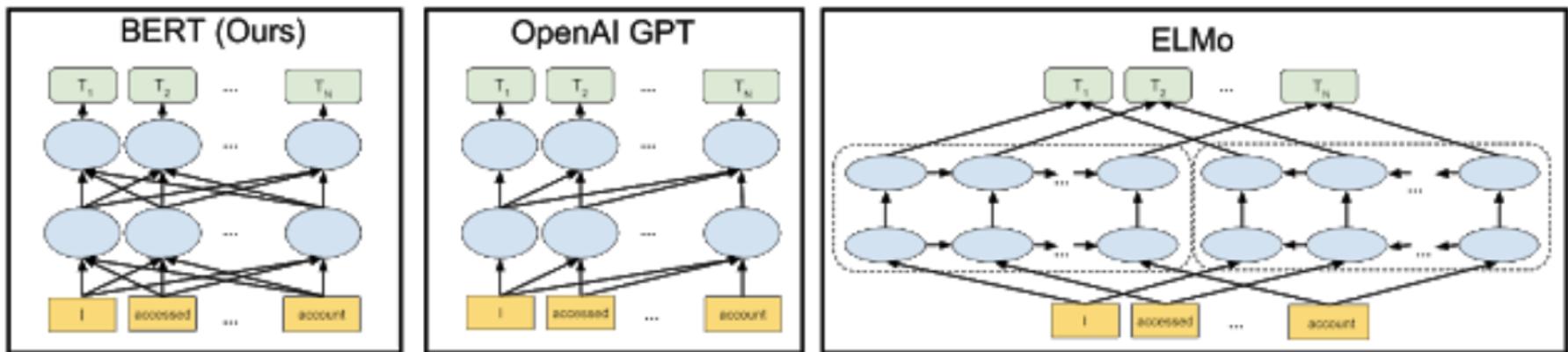
## BERT: State-of-the-Art Pre-training for Natural Language Processing

- BERT builds upon recent work in pre-training contextual representations — including Semi-supervised Sequence Learning, Generative Pre-Training, ELMo, and ULMFit.
- Unlike previous models, **BERT is the first deeply bidirectional, unsupervised language representation**, pre-trained using only a plain text corpus (in this case, Wikipedia).
- Pre-trained representations can either be *context-free* or *contextual*.
- *Contextual* representations can further be *unidirectional* or *bidirectional*.

## BERT: State-of-the-Art Pre-training for Natural Language Processing

- Context-free models such as **word2vec** or **GloVe** generate a single word embedding representation for each word in the vocabulary.
- Contextual models generate a representation of each word that is based on the other words in the sentence.
- For example, in the sentence “*I accessed the bank account*,” a unidirectional contextual model would represent “bank” based on “*I accessed the*” but not “account.”
- However, BERT represents “bank” using both its previous and next context — “*I accessed the ... account*” — starting from the very bottom of a deep neural network, making it deeply bidirectional.

# BERT



BERT is deeply bidirectional, OpenAI GPT is unidirectional, and ELMo is shallowly bidirectional.

# The Strength of Bidirectionality

- It is not possible to train bidirectional models by simply conditioning each word on its previous *and* next words.
- This would allow the word that's being predicted to indirectly "see itself" in a multi-layer model.
- To solve this problem, the authors masked out some of the words in the input and then condition each (masked) word bidirectionally to predict the masked words. For example:

**Input:** The man went to the [MASK]<sub>1</sub> . He bought a [MASK]<sub>2</sub> of milk .  
**Labels:** [MASK]<sub>1</sub> = store; [MASK]<sub>2</sub> = gallon

# Sentence Pretraining

- BERT also learns to model relationships between sentences by pre-training on a very simple task that can be generated from any text corpus.
- Given two sentences A and B, is B the actual next sentence that comes after A in the corpus, or just a random sentence? For example:

**Sentence A** = The man went to the store.

**Sentence B** = He bought a gallon of milk.

**Label** = IsNextSentence

**Sentence A** = The man went to the store.

**Sentence B** = Penguins are flightless.

**Label** = NotNextSentence

## BERT: State of the art

- Demonstrated state-of-the-art results on 11 NLP tasks, including the very competitive Stanford Question Answering Dataset (SQuAD v1.1).