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/

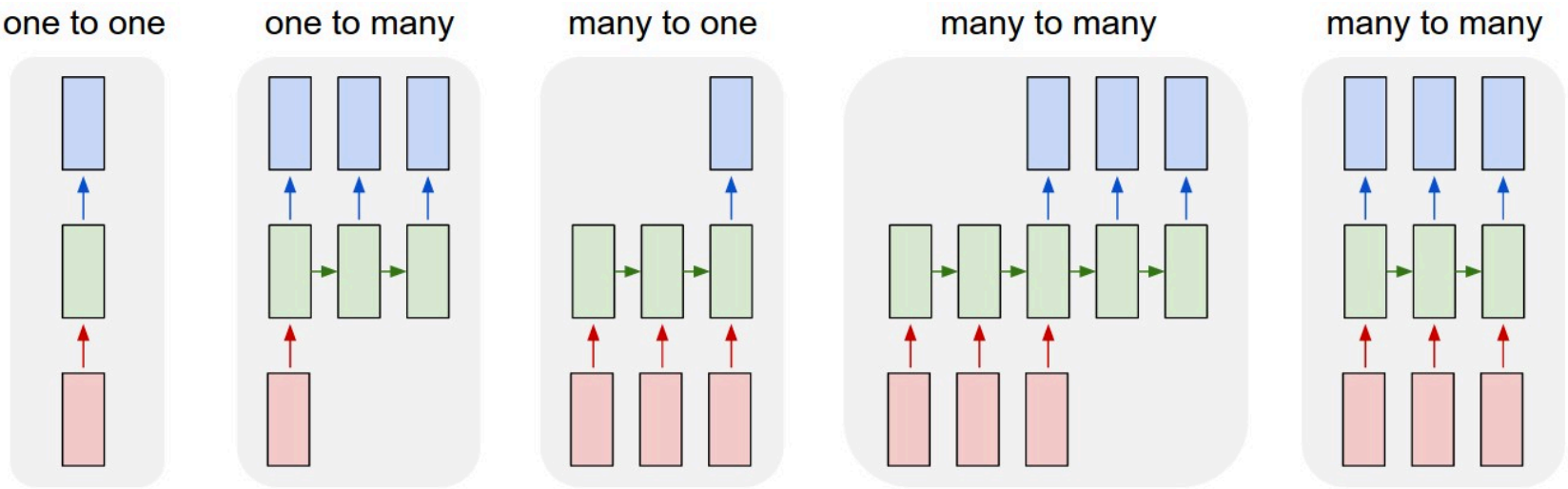
Attention Is All You Need, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

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

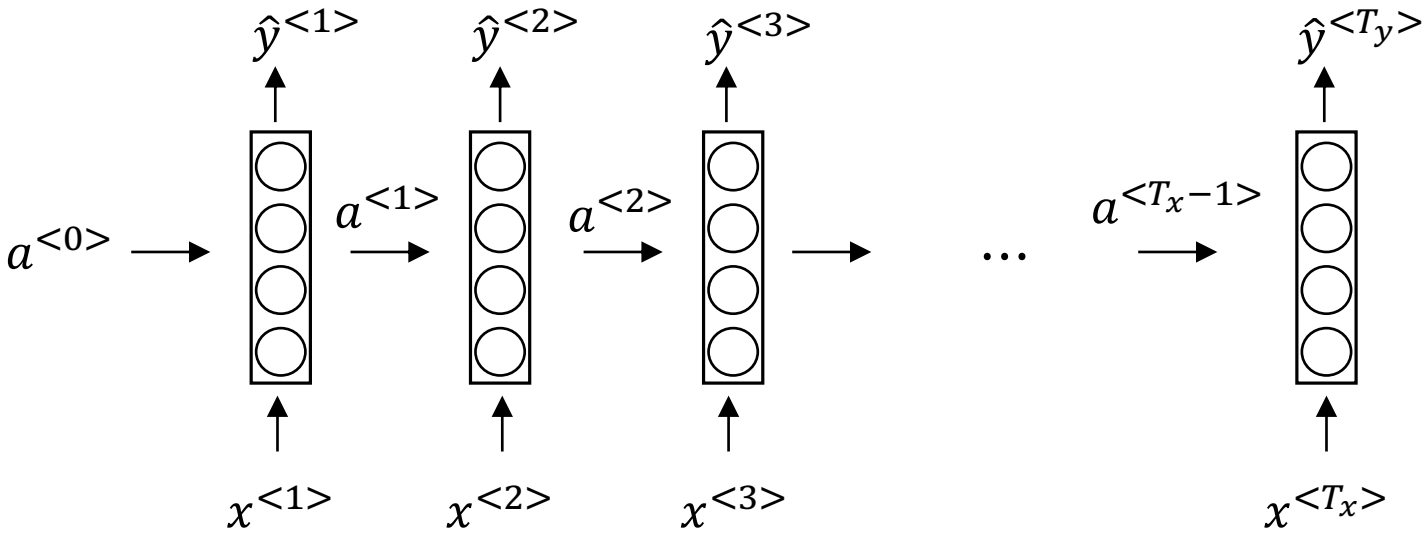
https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html

Recurrent Neural Networks

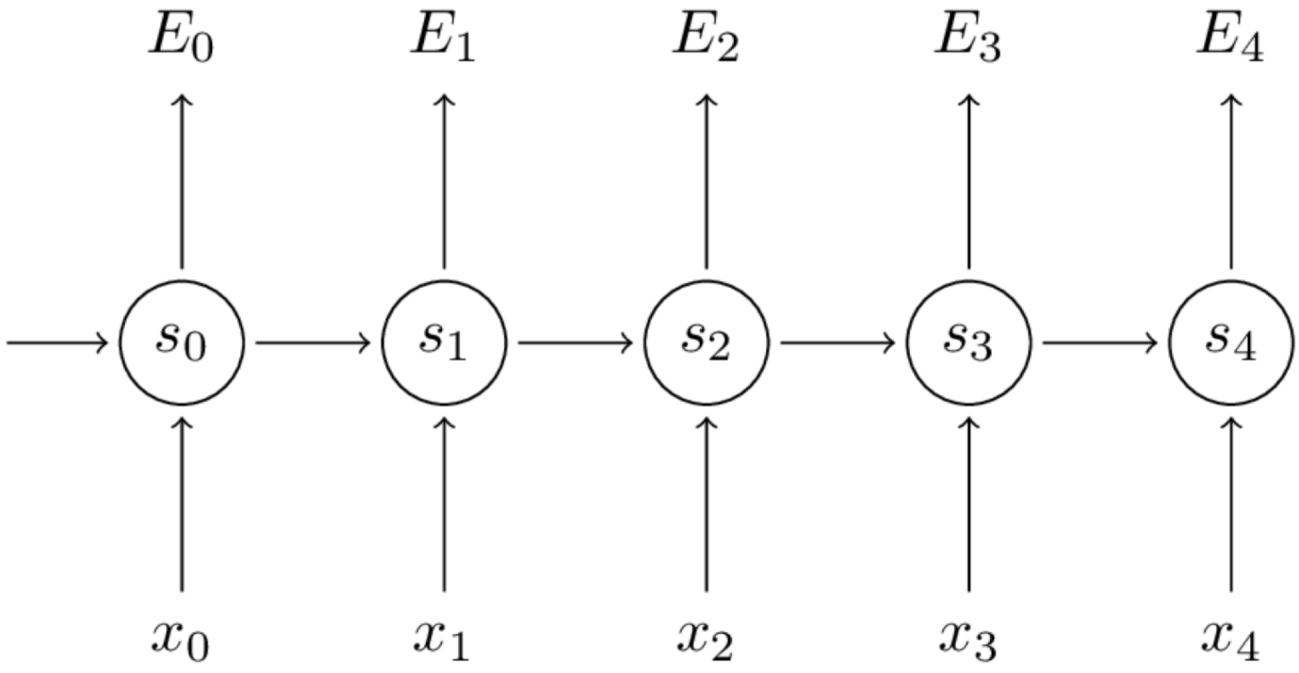
- **Sequences**. Dense Neural Networks, and CNNs only accept a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output (e.g. probabilities of different classes).
- RNN's can operate over Sequences in the input, the output, or in the most general case both.



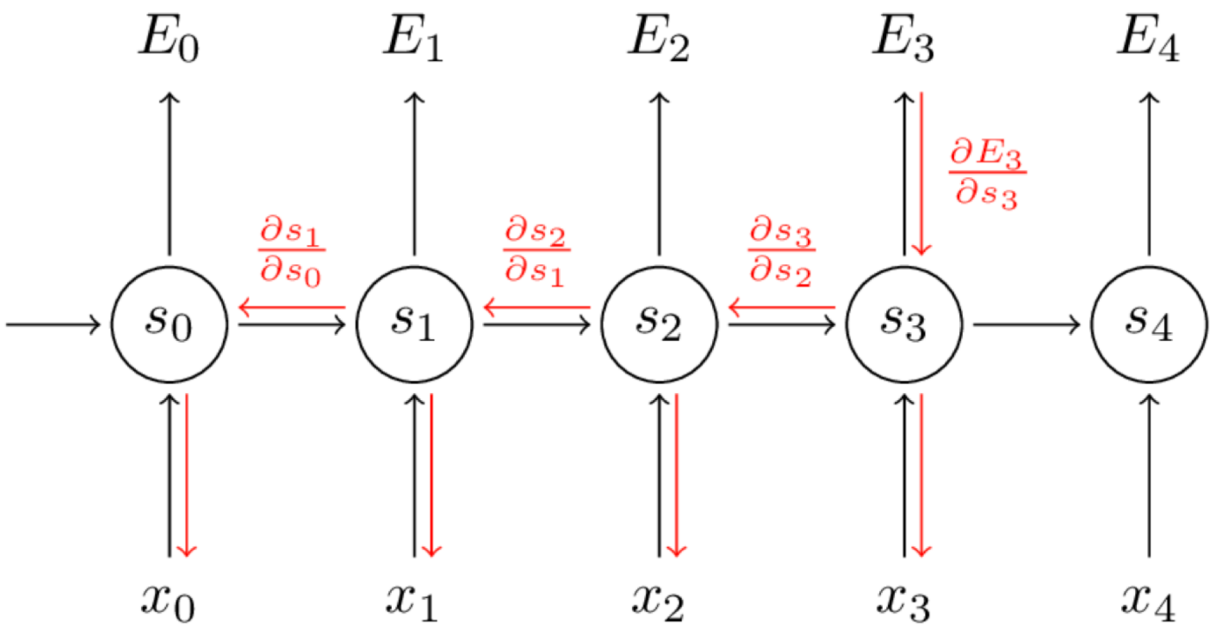
RNN's combine current input with prior state



Forward Prop



Back Prop


E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

RNN forward propagation computation

- **RNN's** accept an input vector x and give you an output vector y.
- Output vector's contents are influenced not only by the input you just fed in, but also on the entire history of inputs you've fed in in the past.
- The RNN class has some internal state that it gets to update every time step is called. In the simplest case this state consists of a single *hidden* vector h.
- 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 and one is based on the current input.
- We can also write the hidden state update as $h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$, where tanh is applied elementwise.

```
# 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
```

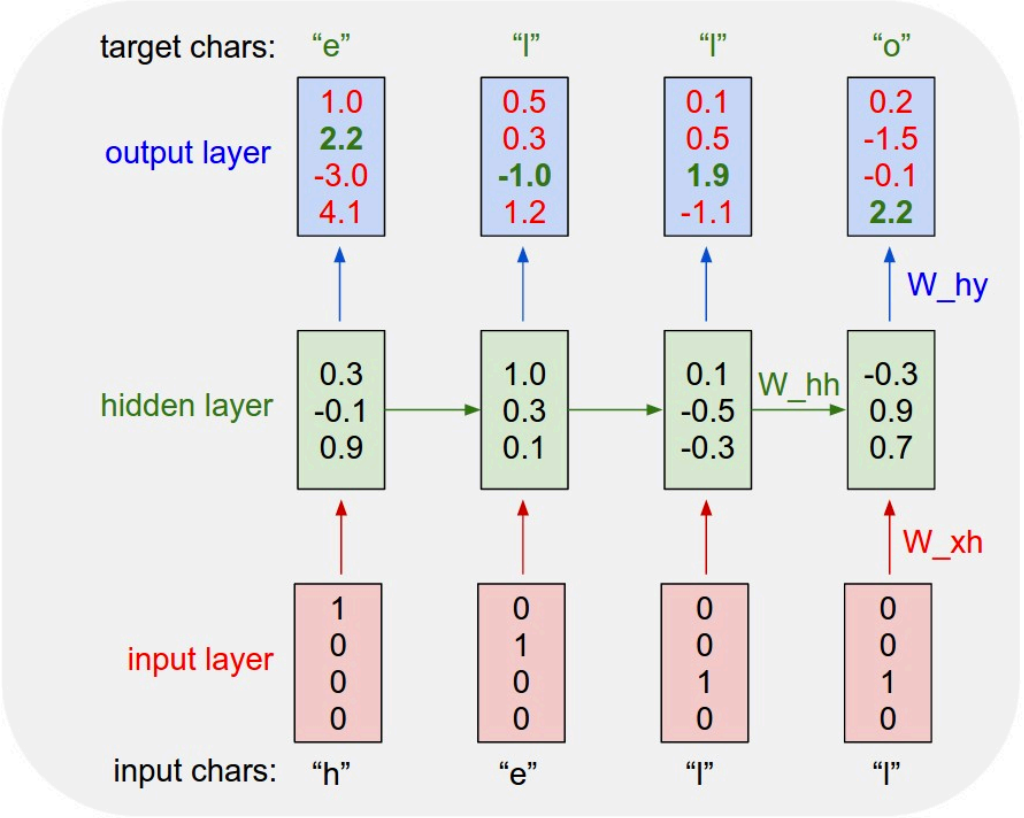
Building deeper layers

- 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.



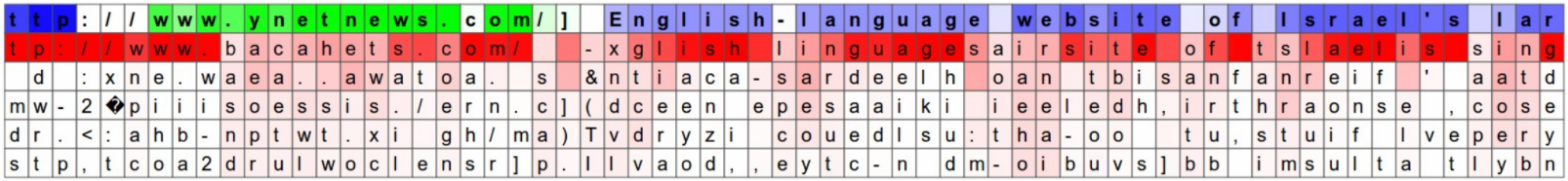
Character-Level Language Models

- 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

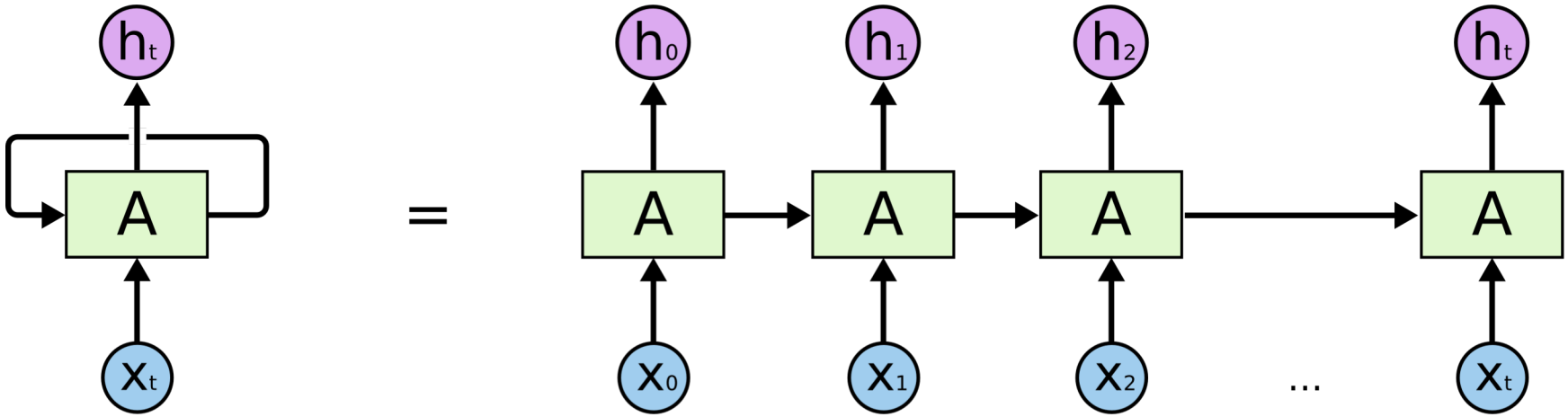
- PANDARUS: Alas, I think he shall be come approached and the day When little srain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.
- Second Senator: They are away this miseries, produced upon my soul, Breaking and strongly should be buried, 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.

Visualization of top 5 under each character in sequence



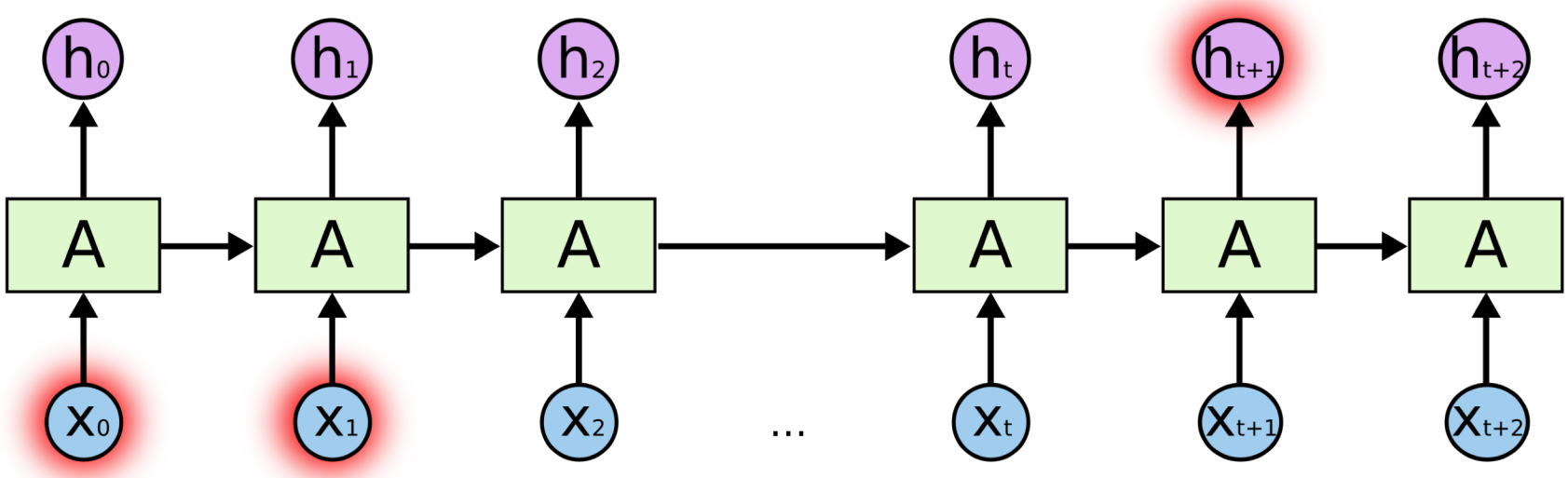
RNNs

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

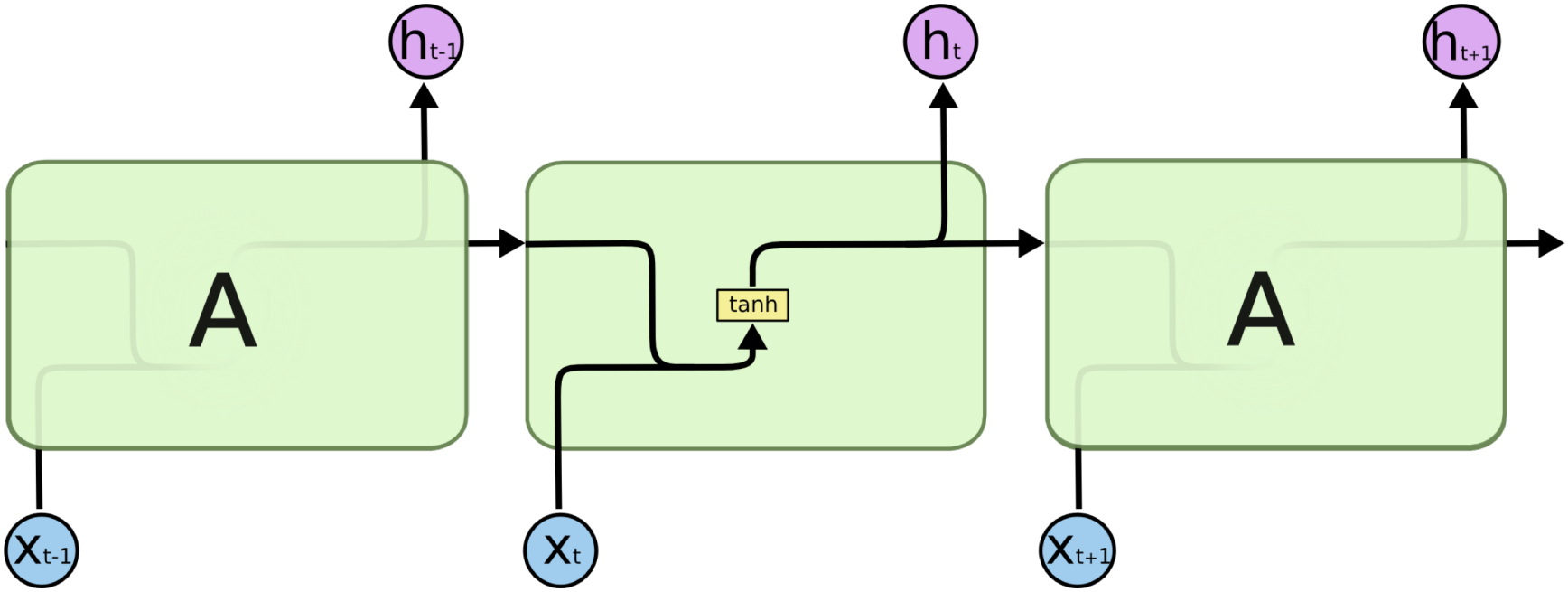


Problem with long-term dependencies

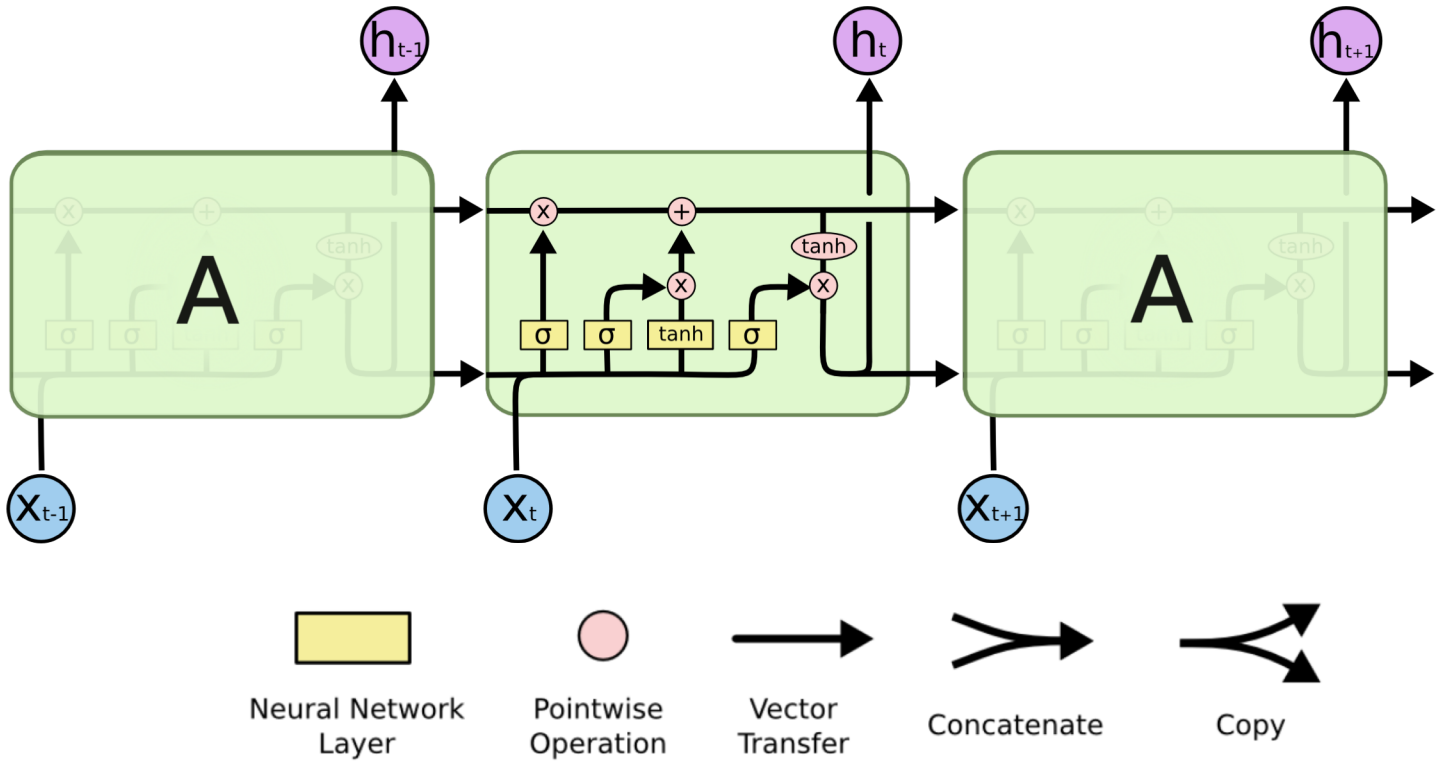
- May need more context.
- Vanishing gradient problem.



The repeating module in a standard RNN contains a single layer

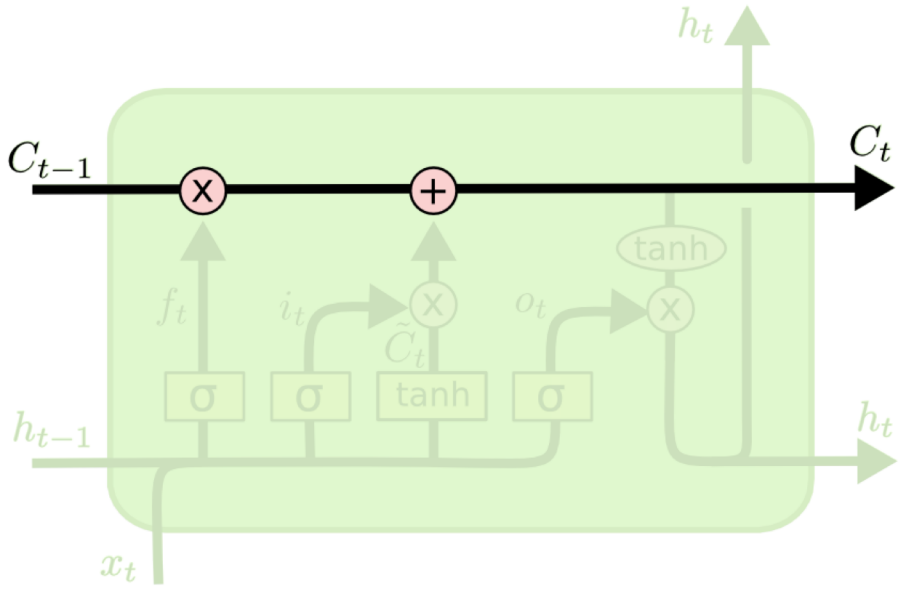


The repeating module in an LSTM contains four interacting layers



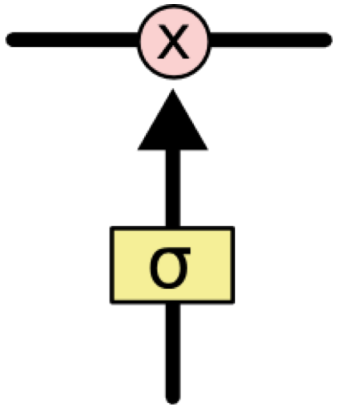
The Core Idea Behind LSTMs

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The cell state is 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.



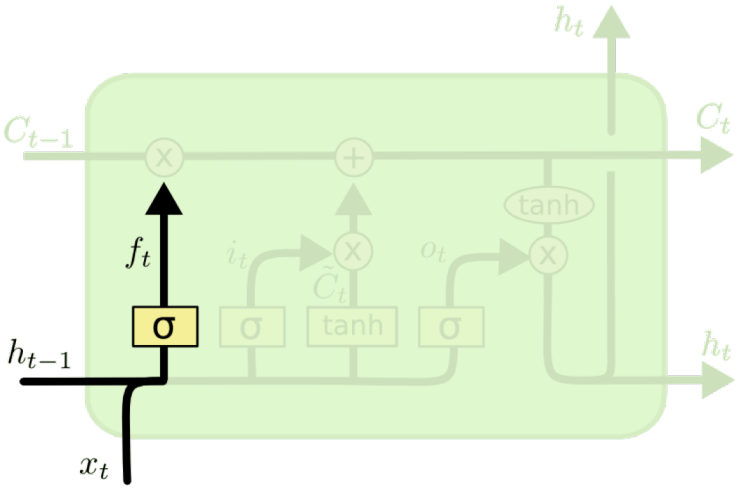
Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

- 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

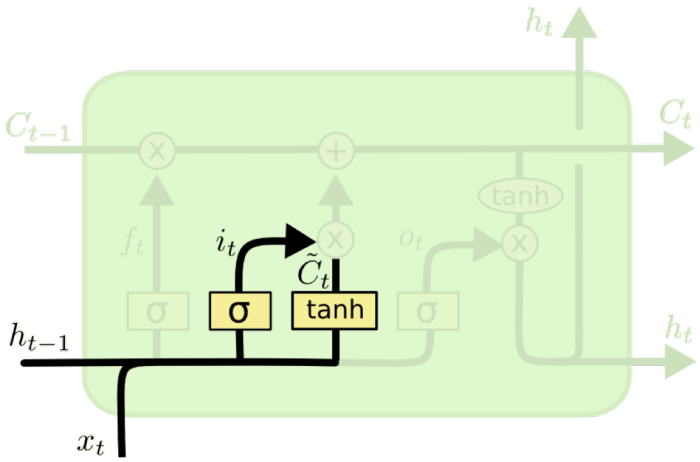
- What information we're going to throw away from the cell state.
- Looks at h_t-1 and x_t, and outputs a number between 0 and 1 for each number in the cell state C_t-1.
- A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



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

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 t

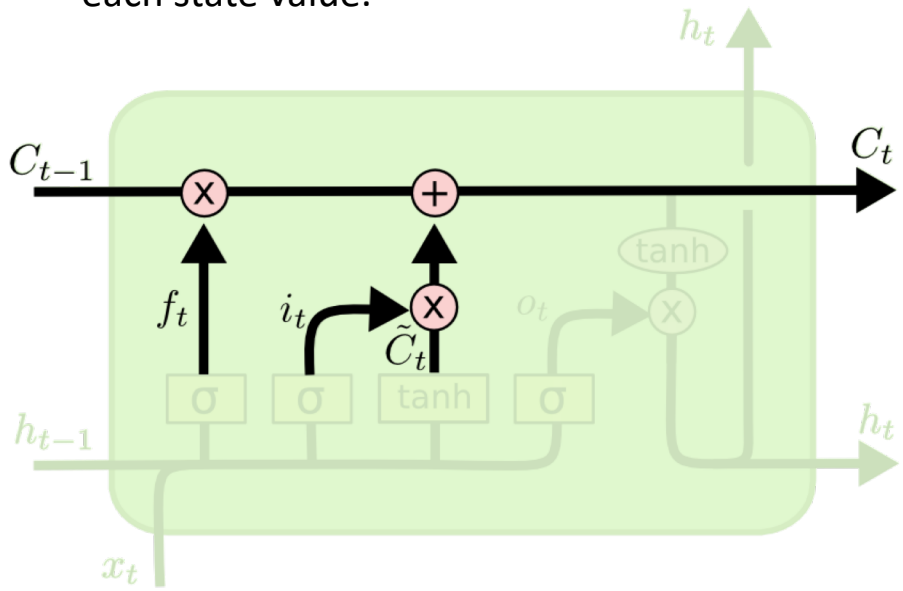


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Update the cell state

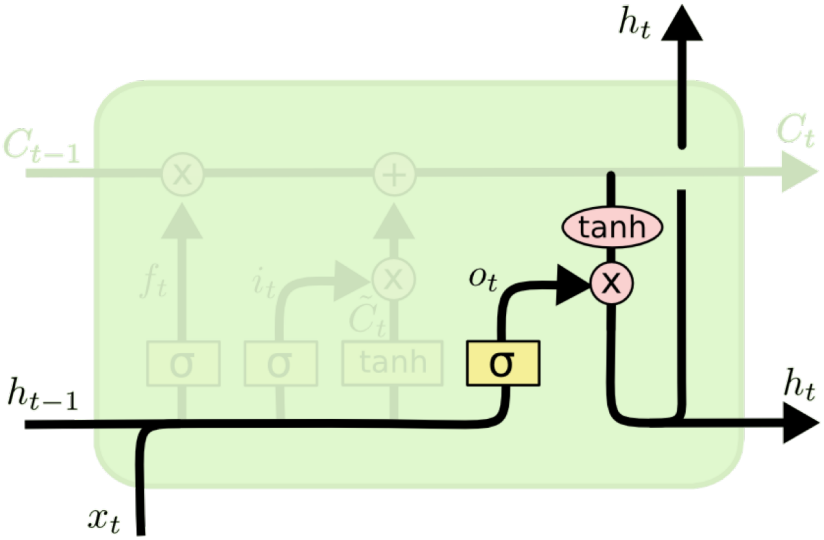
- Update the old cell state, C_t-1, into the new cell state C_t.
- Multiply the old state by f_t, forgetting the things we decided to forget earlier.
- Then add it to $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.



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

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.



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

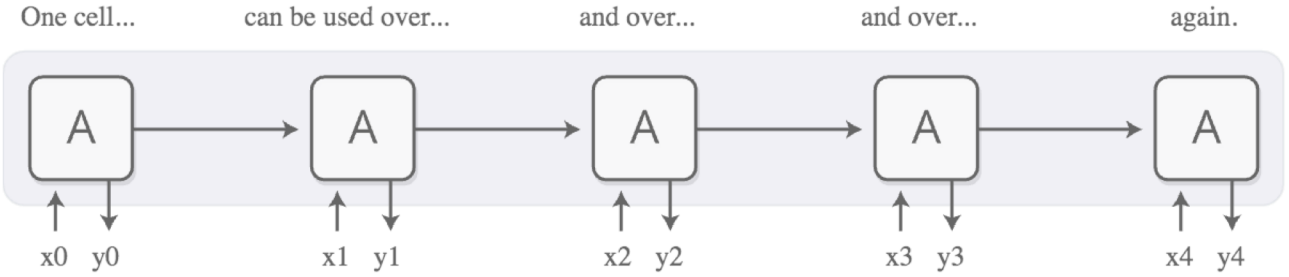
$$h_t = o_t * \tanh\left(C_t\right)$$

Summary LSTMs

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

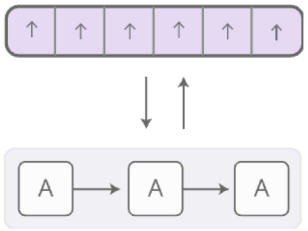
Attention and Augmented Memory Networks

- 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.

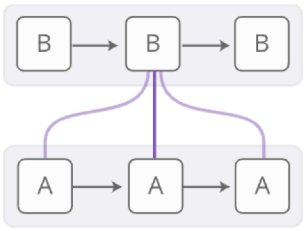


Future directions for RNNs

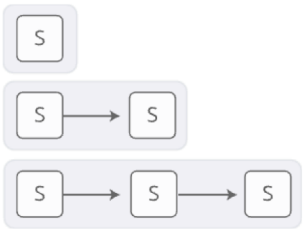
- RNNs have become very widespread in the last few years.
- As this has happened, we've seen a growing number of attempts to augment RNNs with new properties.
- Four directions stand out:



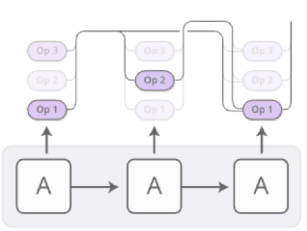
Neural Turing Machines
have external memory that they can read and write to.



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



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

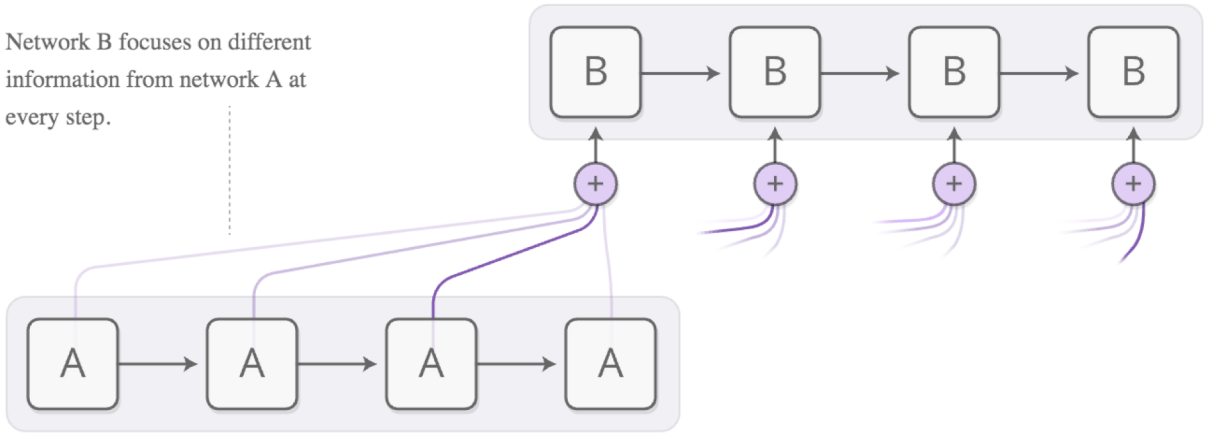


Neural Programmers
can call functions, building programs as they run.

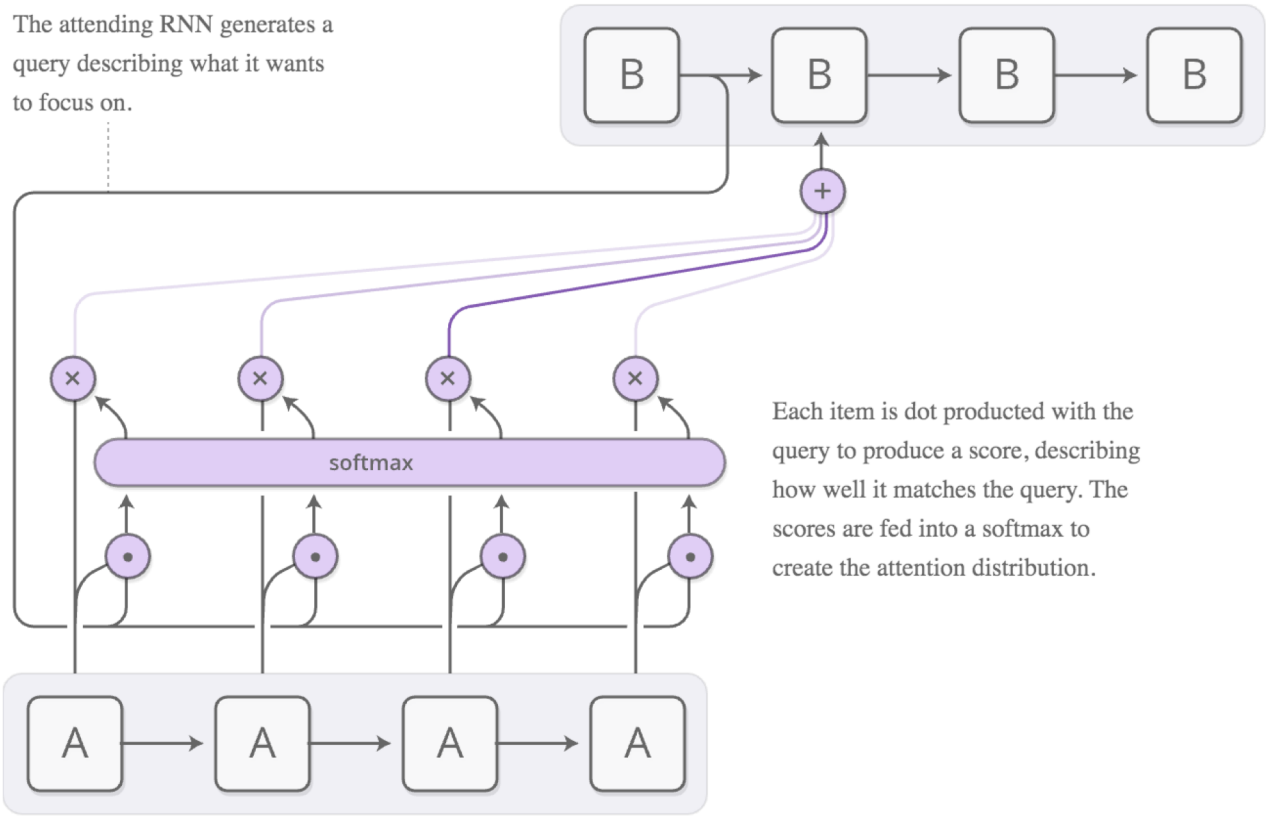
All rely on the same underlying trick—something called attention—to work.

Attentional Interfaces

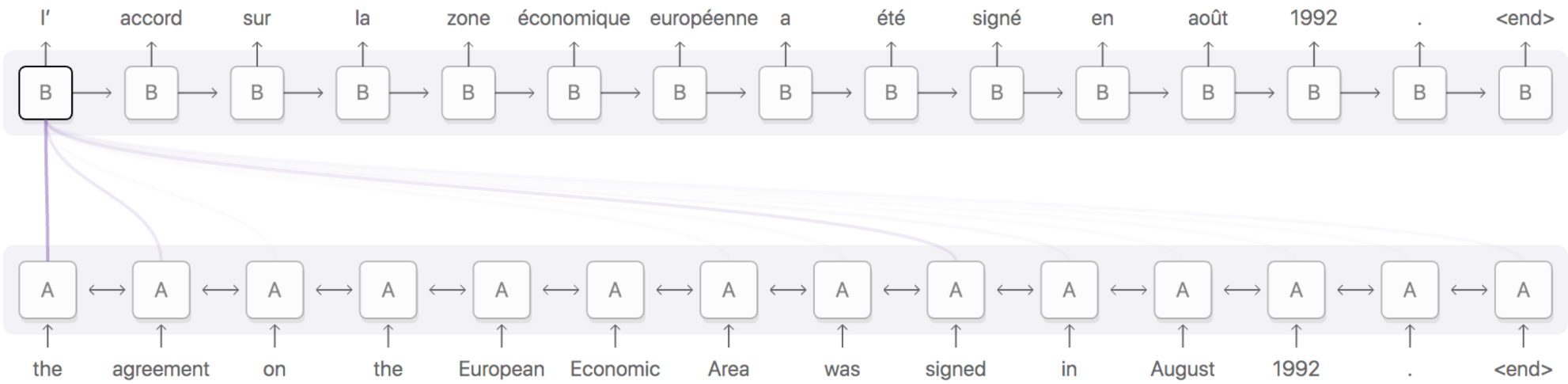
- When you read or transcribe a paragraph, there are parts of the paragraph that you ”attend” to.
- NNs can achieve this same behavior using *attention*, focusing on part of a subset of the information they’re given.
- E.g., an RNN can attend over the output of another RNN. At every time step, it focuses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



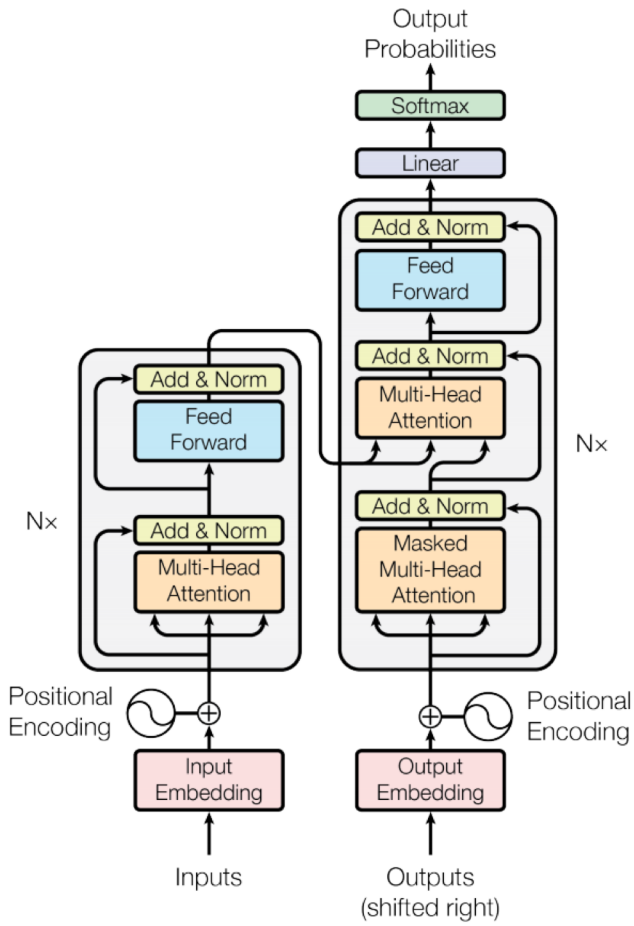
Attending RNN



Attention in language translation encoder-decoder



Attention is all you need: Encoder-decode
Transformer model with attention, self-at



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.

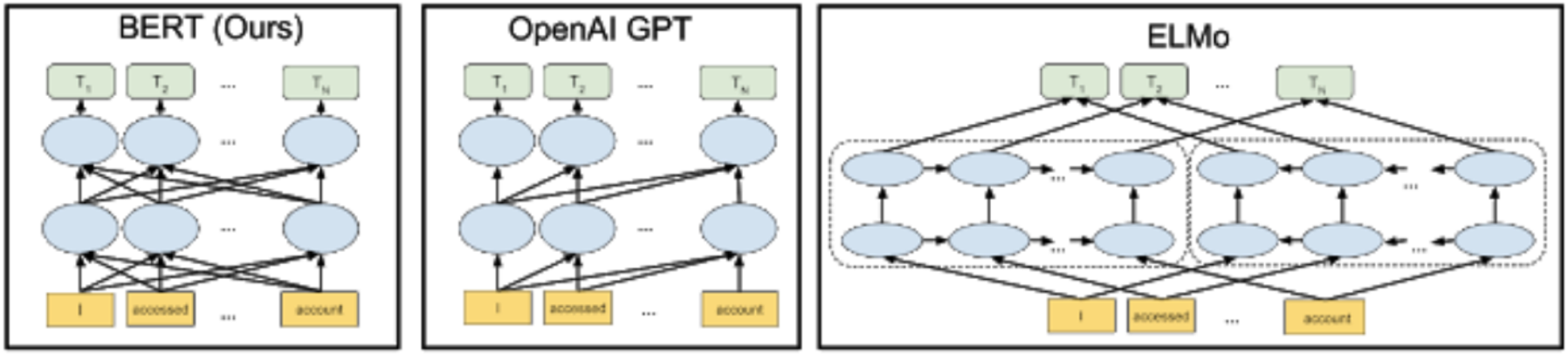
BERT: State-of-the-Art Pre-training for Natural Language Processing

- 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 $[MASK]_1$. He bought a $[MASK]_2$ of milk .
Labels: $[MASK]_1$ = store; $[MASK]_2$ = 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).