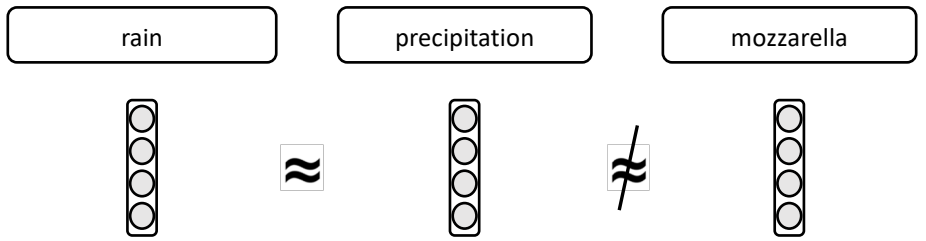
Deep Learning - 2019

Deep Learning for Text

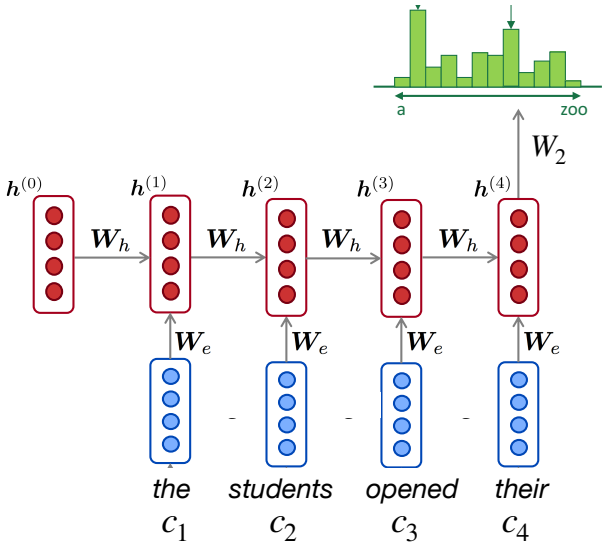
Prof. Avishek Anand

What we learnt before…

- Word embeddings
  - Auto-encoder style architecture to learn dense representations for words



- RNNs
  - Sequence modelling by weight sharing across time steps



Tasks we Focus on Today: Language Models

- Given a piece of text, we want to find out which language does it generated from
  - English, German, French,…
  - Donald Trump, Shakespeare,..
  - Finding better word order
- Greatly useful in NLP, Information Retrieval, Learning from text
- From what are the LM created ?
  - Large bunch of documents where the source is known
    - Cheap and present abundantly
- What are language models ?
  - In essence probability distributions

We assume the number of hidden layers of the LSTM unit is 2 and the number of hidden units is 16.
The output shape returned by the encoder after performing forward calculation on the input is (number of time steps, batch size, number of hidden units).
The shape of the multi-layer hidden state of the gated recurrent unit in the final time step is (number of hidden layers, batch size, number of hidden units).
For the gated recurrent unit, the state list contains only one element, which is the hidden state.
If long short-term memory is used, the state list will also contain another element, which is the memory cell.

P(hot summer the has been)

P(hot the summer has been)

P(the Summer has been hot)

Tasks we Focus on Today – Machine Translation

- Translation is $40 Billion industry
- Social/Economic/Military uses
- Google Translate (2016): 500 Million users, 100 Billion words/Day
- Lots of monolingual data available

- Fundamental NLP task!
- Essentially Language Models with conditioning

Goals of the Language Model

- **Goal:** compute the probability of a sentence or sequence of words:
  - $P(W) = P(w_1,w_2,w_3,w_4,w_5...w_n)$
- **Related task:** probability of an upcoming word: $P(w_5|w_1,w_2,w_3,w_4)$
- A model that computes either of these:
  $P(W)$ or $P(w_n|w_1,w_2...w_{n-1})$ is called a language model or LM

Statistical Language Models

- Learning language models from large text corpora
  - Given a large text corpus – say all sentences in Wikipedia

- Count frequency of occurrence of words – unigrams, bigrams, tri-grams ,…

$$p(w_1) = \frac{\text{count}(w_1)}{\sum \text{count}(w_i)} \qquad p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \qquad p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

- Why is more context better ?
  - Tri-gram > Bi-gram > Unigram

P(hot summer the has been)

P(hot the summer has been)

P(the summer has been hot)

- Typically have a Markovian assumption

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i|w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P\left(w_i|w_{i-(n-1)}, \ldots, w_{i-1}\right)$$

Problem of Sparsity

What comes next ? students opened their X

Sparsity Problem 1

Problem: What if *"students opened their* w_j*"* never occurred in data? Then w_j has probability 0!

(Partial) Solution: Add small δ to count for every $w_j \in V$. This is called *smoothing*.

$$p(w_j \mid \text{students opened their}) = \frac{\text{count(students opened their } w_j)}{\text{count(students opened their)}}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* w_j!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Problems with Classical LM

- Sparsity
  - Need lot of data to estimate long-length n-grams
  - Assume |V| = 10^5. What is the number of contexts for 4-grams ?

$$p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

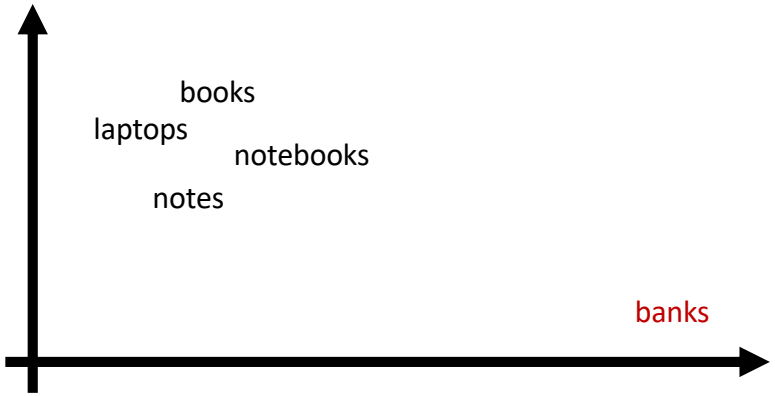
Most of them are not seen

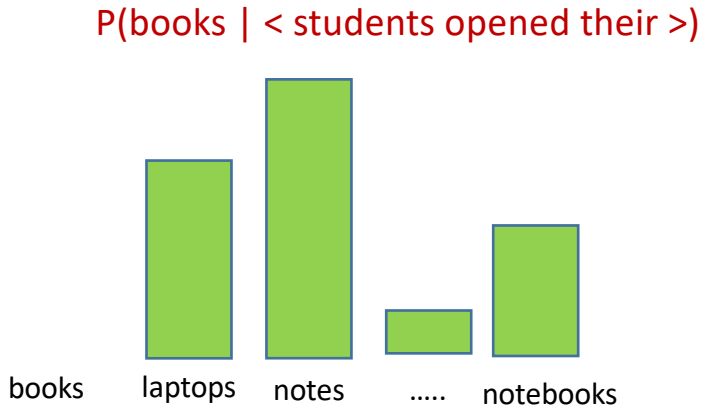
- Insufficient context
  - Markovian assumption imposes a computational limit on the context

- What is needed ?
  - We could estimate the probabilities for n-grams better by exploiting word similarity
  - Why not let a Neural Network estimate these probabilities automatically ?

Neural Language Model

- How does is solve the sparsity problem ?
  - Words as dense distributed vectors so there can be sharing of statistical weight between similar words
  - Doing just this solves the sparseness problem of conventional n-gram models
    - **students opened their** {books, laptops, notes, notebooks, ….}
    - Even if notes is not observed or P(books | < **students opened their** >) = 0





RNN Neural Model

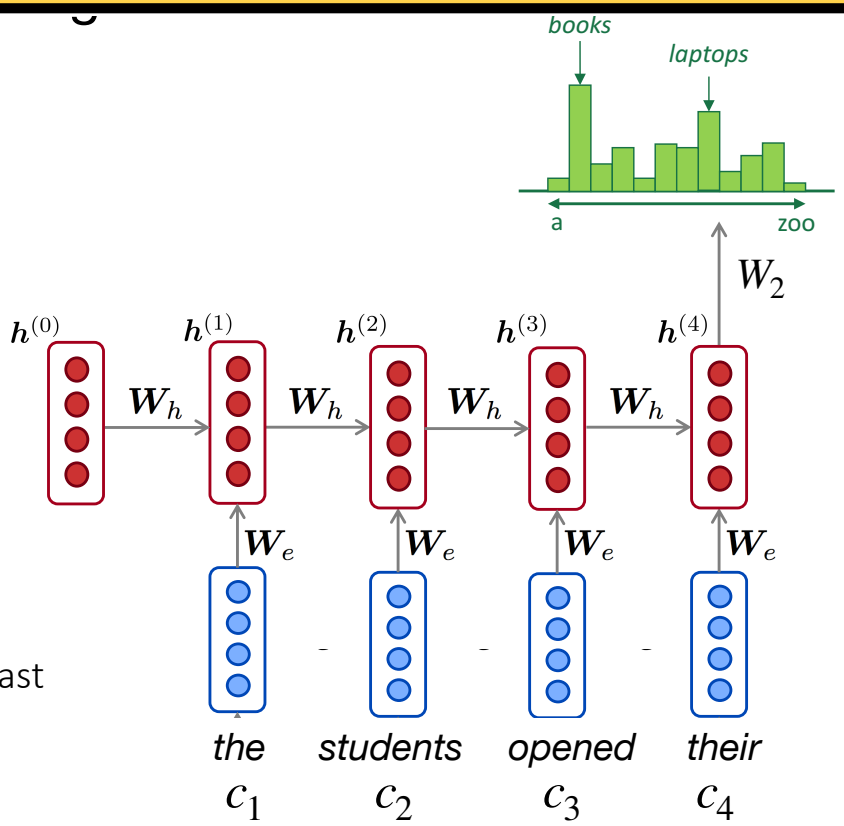
word embeddings

$$c_1, c_2, c_3, c_4$$

output distribution

$$\hat{y} = \text{softmax}(W_2 h^{(t)} + b_2)$$

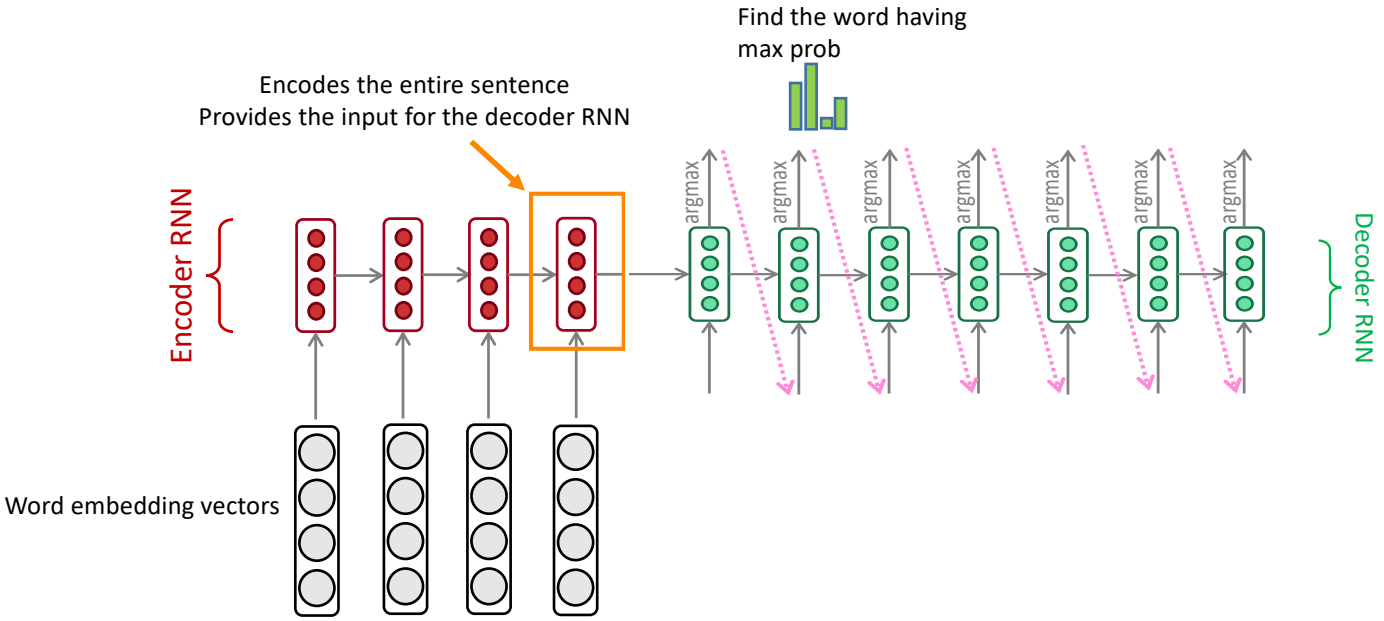
- What is the training data ?
  - Training data generation: Mask the last word (predict the last word)
- What is the loss function ?
- What are the parameters to be learnt ?



Use word embeddings

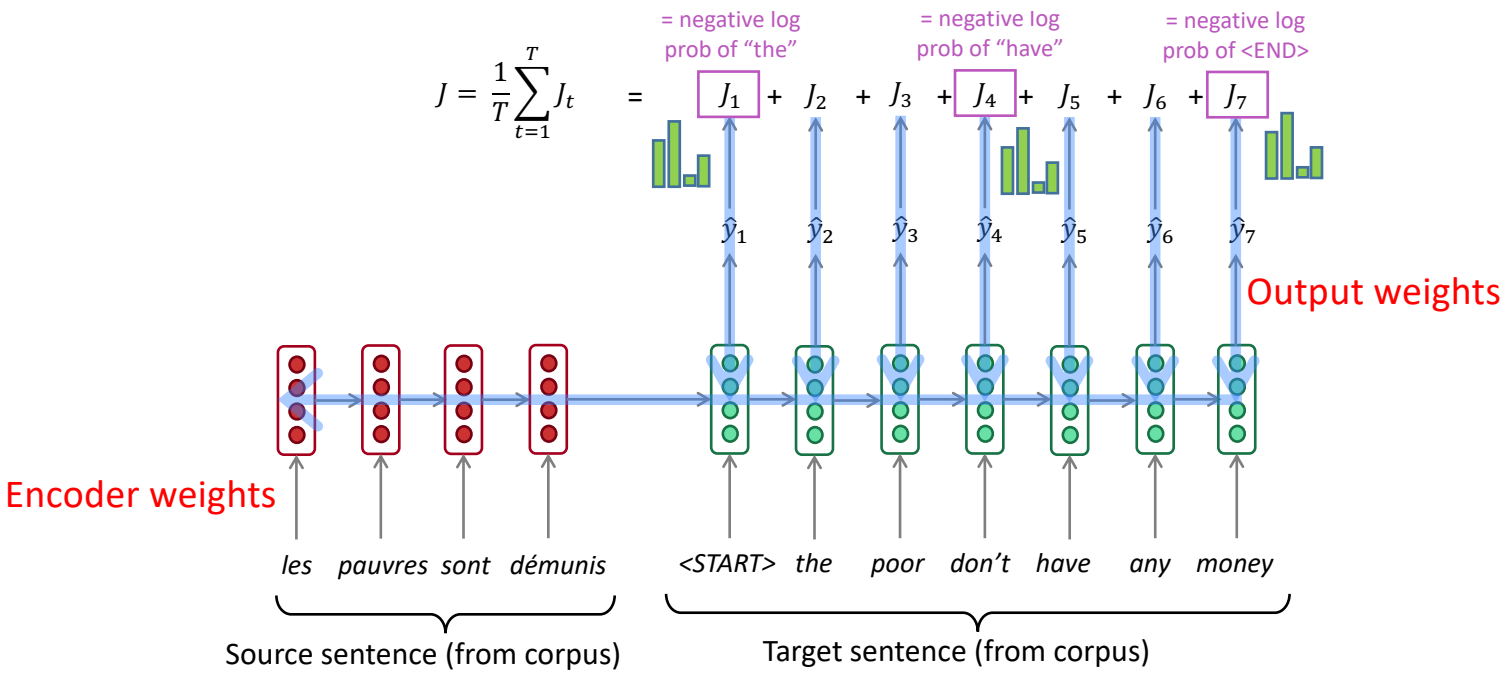
- Solves the problems due to Markovian assumption and sparsity

Neural Machine Translation – Simple Model



- Input as word embeddings
- Encoder: Use GRU/LSTM to encode the input sentence
- Decode the next word
  - Given previous hidden state
  - Previously generated word decode next word $P(x_t \mid x_{t-1}\, h_{t-1})$
  - find word with highest likelihood from the conditional prob. – argmax of the softmax distribution

Training a Neural MT



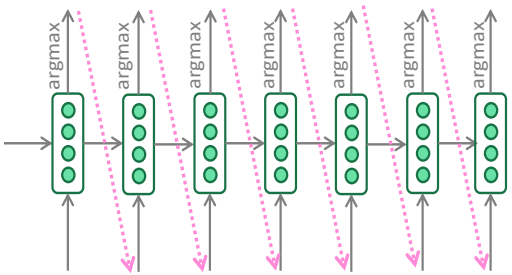
What are the parameters that are learnt ?

- Encoder and decoder weights,
- Output weights

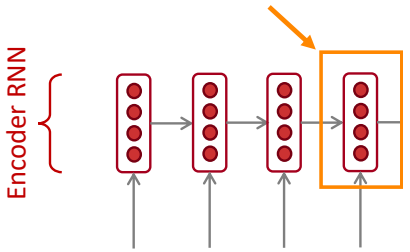
Limitations of RNN NMT

- Training problem: Training issues and architectural improvements
  - Same weights for encoding and de-coding, end-to-end training
  - Pre-trained vectors might not be good enough
  - Just looking at current state might be limiting

- Decoding problem: is chosing best candidate at each time step optimal ?
  - Problem of decoding – is argmax good enough



- Bottleneck problem : Encoding everything into one vector
  - Long sentences and distant contexts



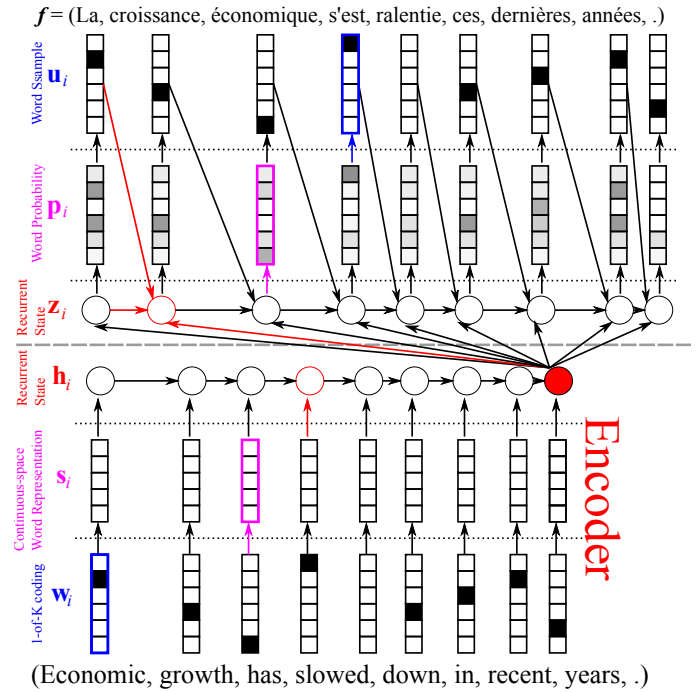
Overcoming Limitations of Training

- Three improvements
  - Current decoding is based on previous hidden state, prev. emitted word

1. Focus on the output of the encoder

2. Use different parameters for encoding and decoding

3. End to End training rather than using pre-trained embedding

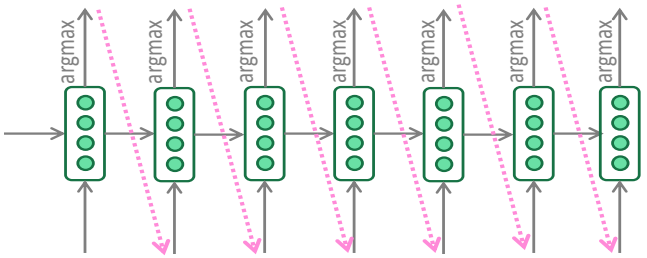


Decoding strategies

$$\arg\max \prod_{i=1}^{L} p\left(e_i | e_1, \ldots, e_{i-1}, f\right)$$

- Greedy approach – Choose the most probable word
  - Fast but inaccurate – repetitions, incoherent sentences

- Optimal approach – find all possible sentences
  - In greedy decoding, we cant go back to revise prev. decisions
  - Computationally expensive, prohibitive, DO NOT TRY..

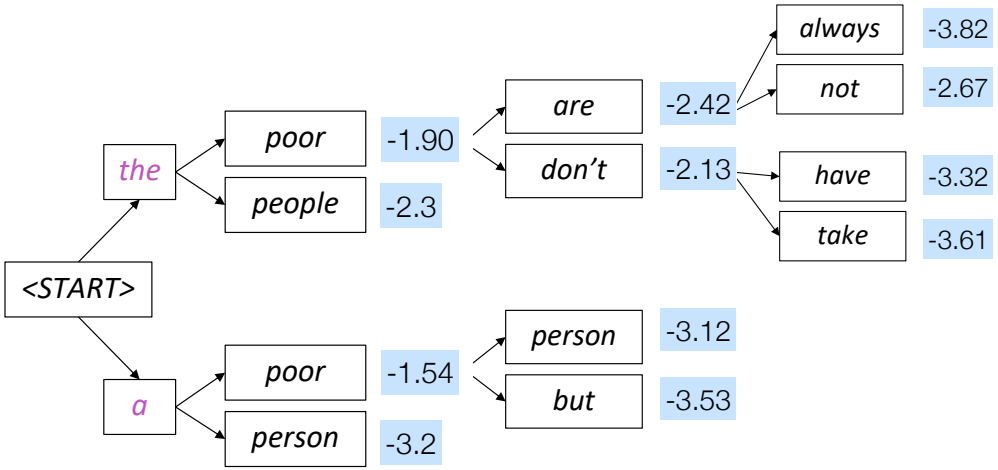


- Beam search
  - Explore several different hypothesis instead of a single one
  - Try to enumerate top-k most likely candidates at each time step
  - Best of both worlds – computationally tractable and works in practice
    - Usually beam size of 5-10 is good enough

Decoding : Beam Search

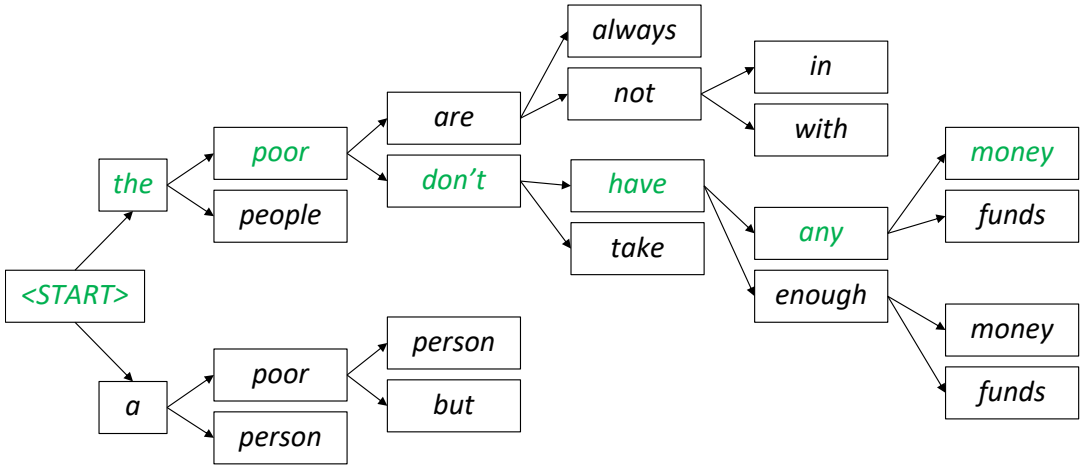
- Beam search of size k keeps track of the max log-likelihoods
- Only decodes by computing k candidates for each of the prev. top-k choices

- When do we stop ? Does is give us optimality ?



Beam size = 2

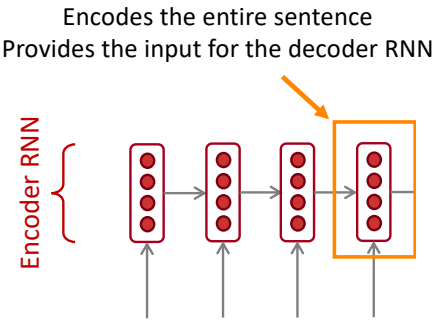
les pauvres sont démunis



Bottelneck Problem

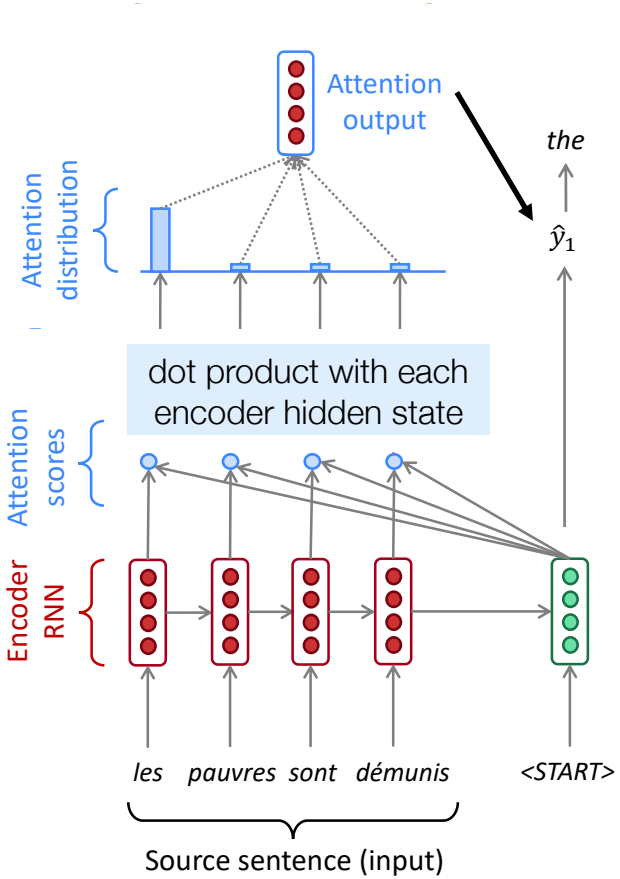
"You can't cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!"

Ray Mooney



- Given an already decoded output we should be able to refer back to the original statement
  - Partially solved by conditioning on the *encoding of the input sequence (bottleneck vector)*
- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
  - Conceptually similar to *alignments*

Attention Revisited

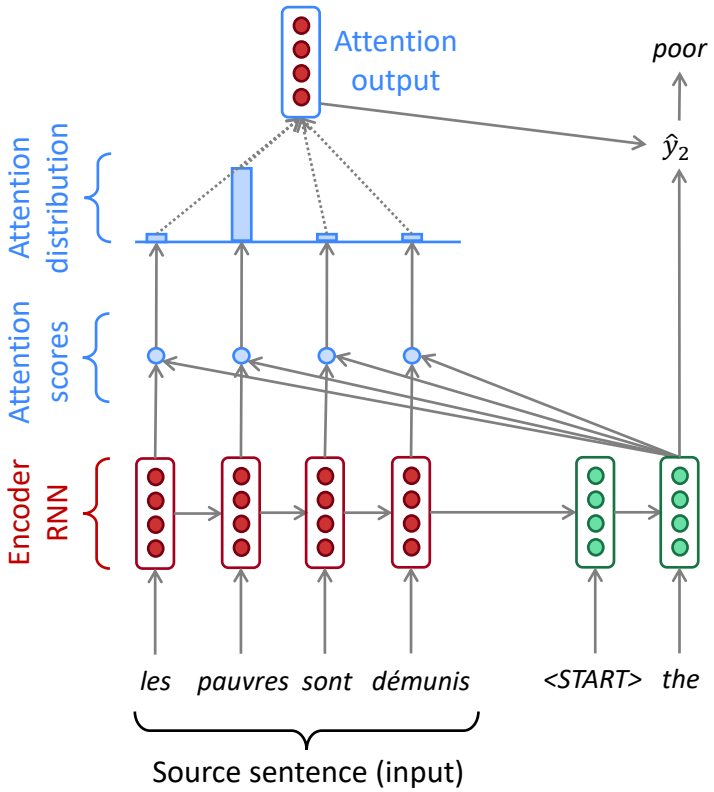


On this decoder timestep, we're mostly focusing on the first encoder hidden state (*"les"*)

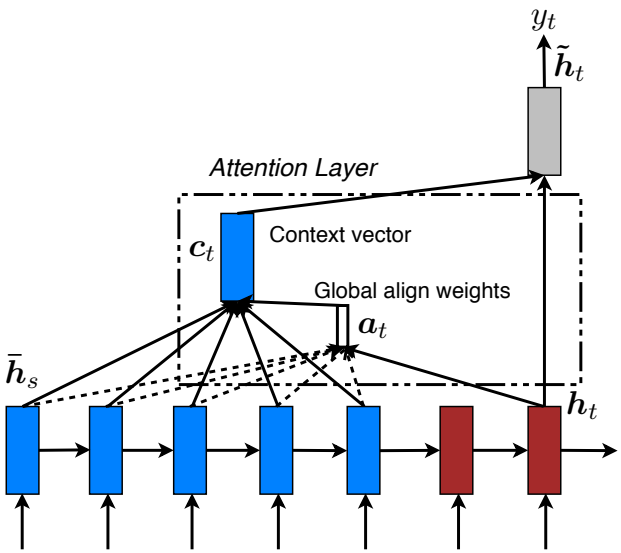
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information the hidden states that received high attention.

Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before



Attention Layer

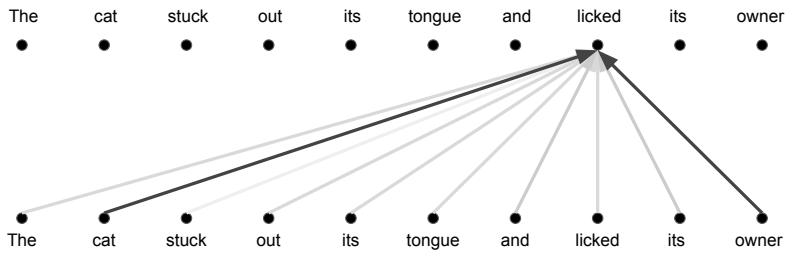


$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) \end{cases}$$

- Context vector is obtained as value (linear combination of all encoder states) using the decoder vector as key
- Use of attention is like memory access with
  - key as the current state and the
  - value to be retrieved as previous hidden states

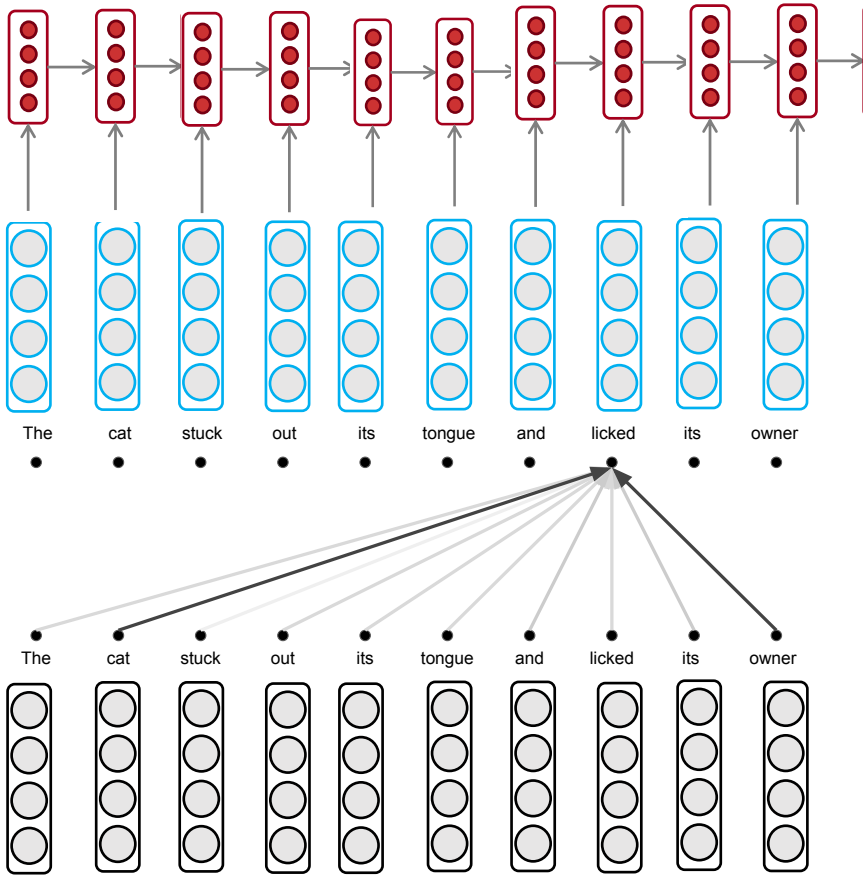
Self-Attention

- Utility of context-agnostic word vectors
  - "I am going to the bank", "The river bank can be dangerous", "X knows he can bank on Y"
    - The vector representation used as input to RNN cannot differentiate contexts
- Problems with RNNs: Sequential computation inhibits parallelization
  - What if we ignore recurrences altogether ?
- Self attention
  - Re-express representation in terms of the context the word occurs in



Incorporating context to word vectors

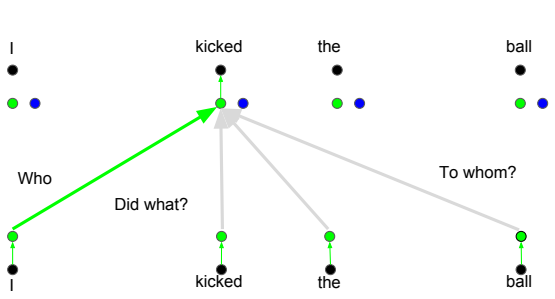
- We could re-express the word vectors as linear combinations of its neighboring vectors
  - Use attention-weights to incorporate context
  - Learn this end to end based on the task
- Intra-attention or self-attention
  - Construct Probability distribution of importance - Dot product and normalization with only the input
  - Re-express word vectors weighted by the probabilities as weights



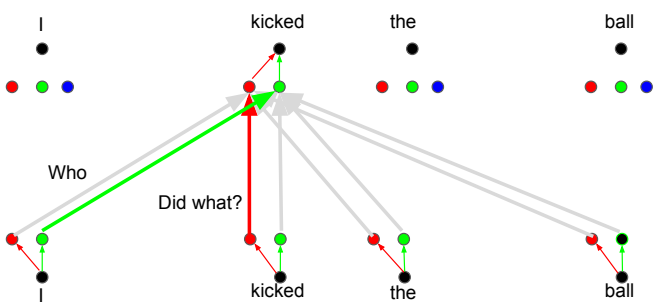
Multi-Head attention

- However all similarities do not encode the same aspects
  - Rather than having one notion of similarity, assume words are connected by different notions of similarity
  - Each notion of similarity is represented by a **attention head**
  - We do not explicitly specify what the similarity is, rather let the model learn this from data

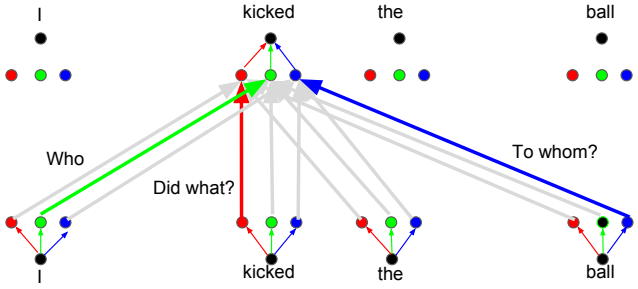
- Each attention head encodes a different type of similarity between words



Aspect: “who”



Aspect: “did what”

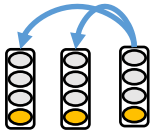


Aspect: “to whom”

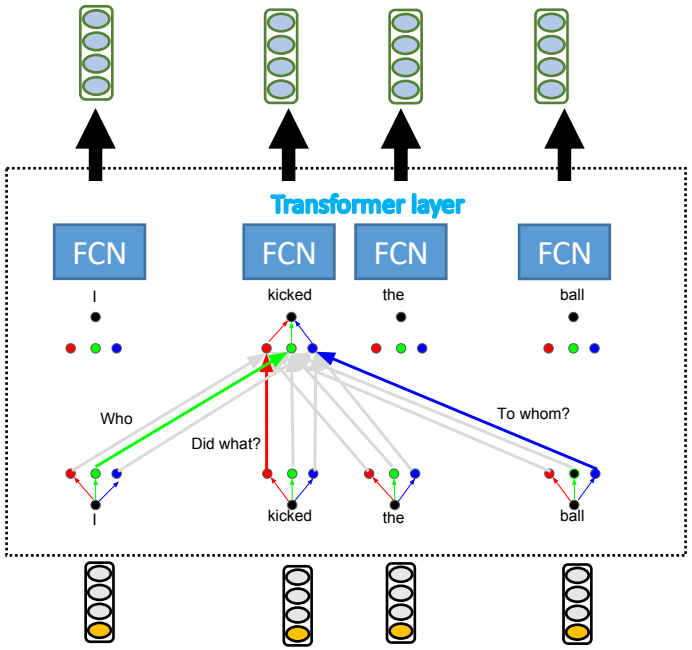
Transformers – Only Attention

- RNNs encode sequence information
  - Dependencies on the previous states makes it computationally expensive
  - Explicitly encode positional encoding

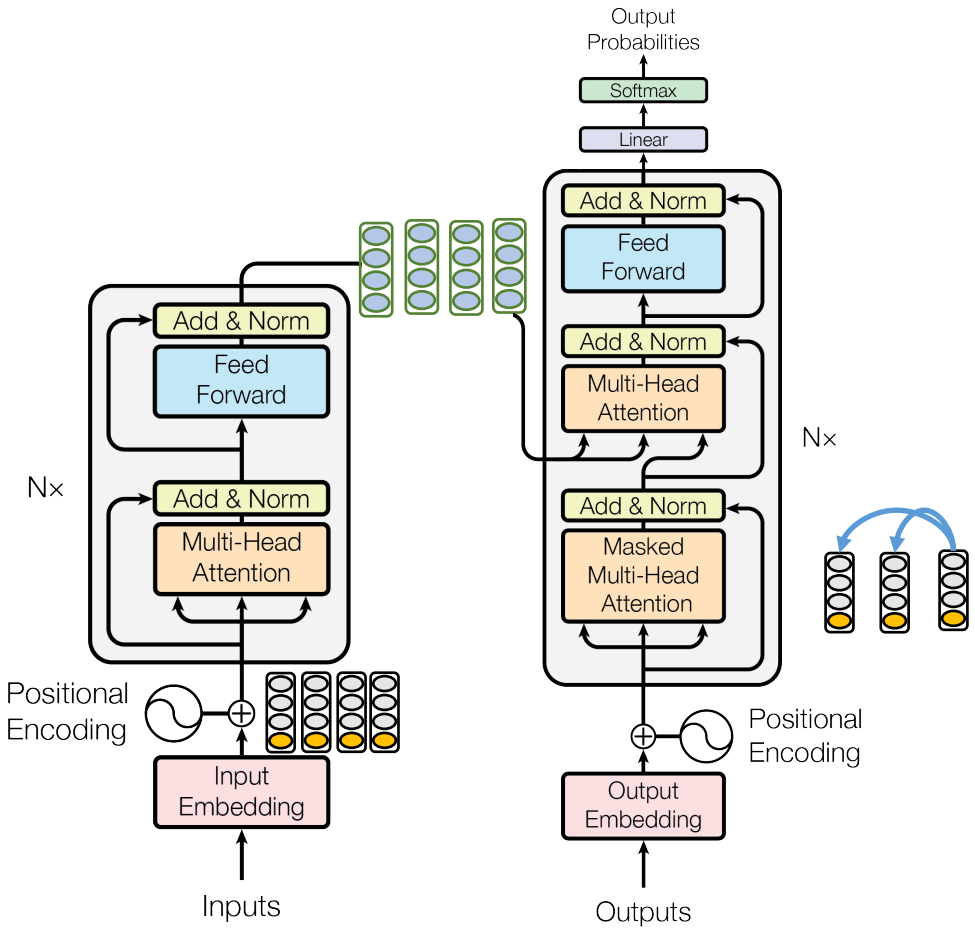
- Uses three types of attention
  - Self attention
  - Encoder decoder attention
  - Masked attention
    - Used in the decoder
    - Like self attention but only attends to the past tokens



- In the decoder compute a distribution over words
  - Given the current token
  - Attention weights to all input tokens
  - Attention weights to all output tokens (already generated)



Transformers

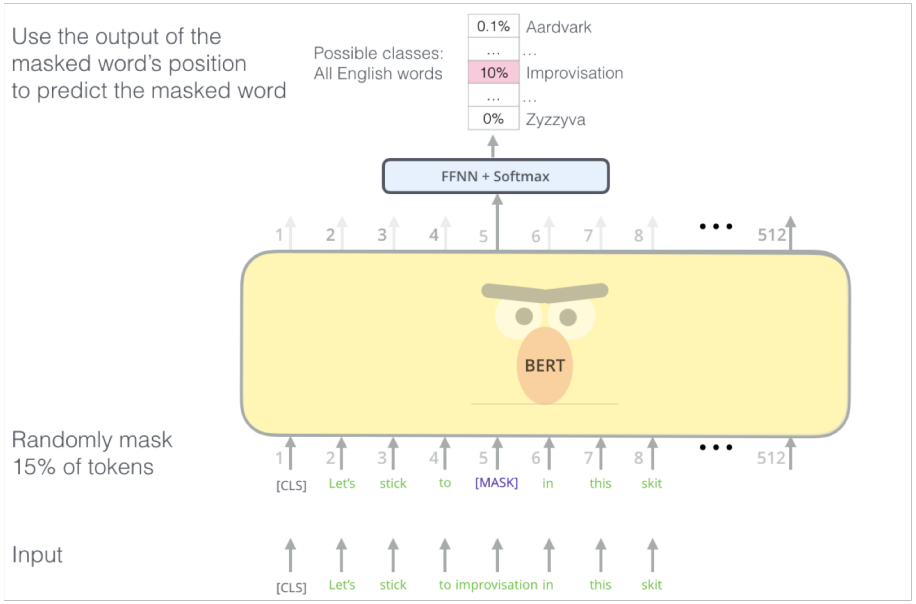


- Encoder decoder attention
  - Given current state **t** compute a representation by attending to all input representations
    - Solves all long distance context issues
- Masked attention
  - Useful in training, mask (zero out) future values
  - In RNNs this was not a problem
- Lots of other bells and whistles
  - Residual connections – We learnt this in lecture on CNNs
  - Layer Normalization – Similar idea to batch norm. (Optimization Tech.)
  - Scaling and Dropouts – Lecture on Regularization Tech.
  - Beam Search – this lecture
- Massive set of parameters
  - Currently the state-of-art in many NLP tasks (almost)

BERT – Different training data

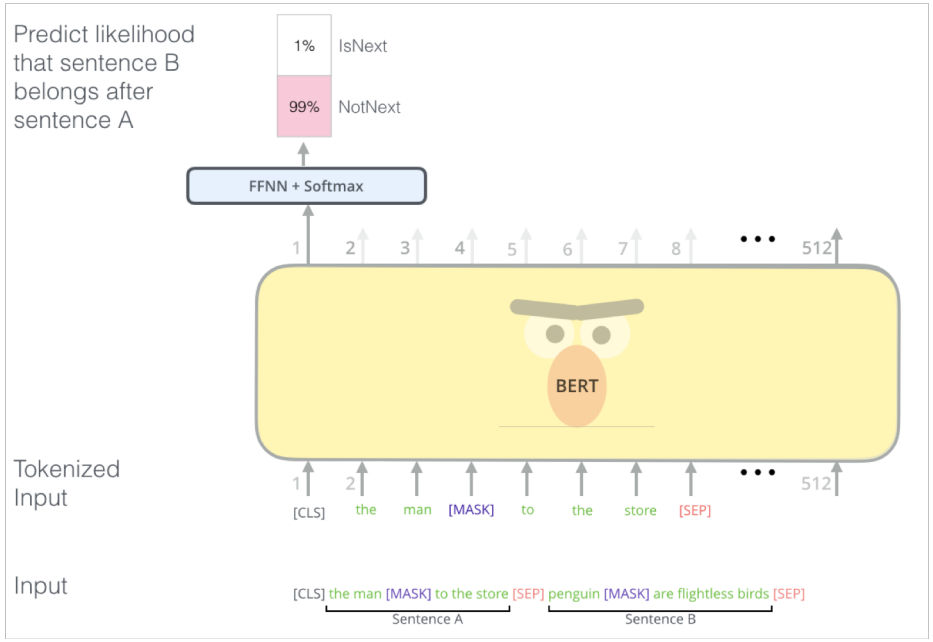
Masked word prediction

- Given a sentence with some words masked at random, can we predict them?
- Randomly select 15% of tokens to be replaced with “<MASK>”



Next sentence prediction

- Given two sentences, does the first follow the second? Teaches BERT about relationship between two sentences
- 50% of the time the actual next sentence, 50% random



References

- http://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
- http://jalammar.github.io/illustrated-transformer/

- Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. "A neural probabilistic language model." *Journal of machine learning research* 3, no. Feb (2003): 1137-1155.
- Neural Machine Translation: Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).
- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
- Transformers: Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Advances in neural information processing systems*, pp. 5998-6008. 2017.
- BERT: Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).