Encoder-Decoder Models
Attention. “The” transformer model.

Venelin Kovatchev

Lecturer in Computer Science

v.o.kovatchev@bham.ac.uk

Outline

- Quick recap
- Encoder-decoder networks
- Attention
- Original transformer

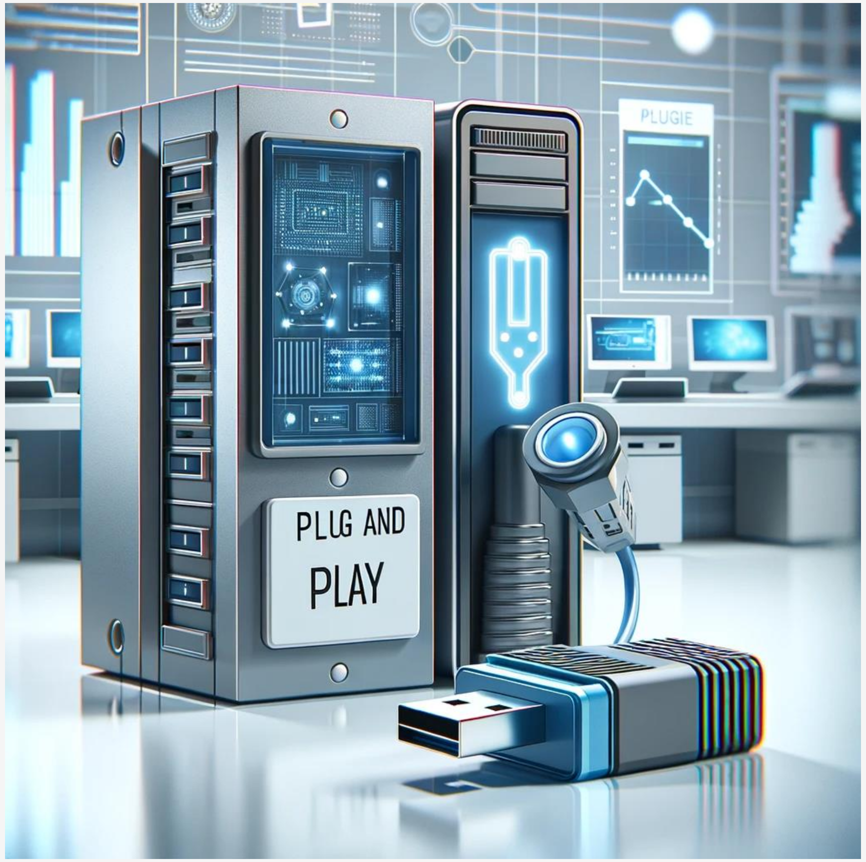
Quick recap:
End-to-end neural networks

What are end-to-end models

- Task specific models
- Map directly from input to output
- No feature engineering
- Trained via backpropagation
  - Data and compute expensive

What are some advantages of end-to-end

- Better performance
- Simpler pipeline
- Changing the problem formulation
  - The task is defined by the data and the metrics
- Making NLP more accessible
  - Plug and play



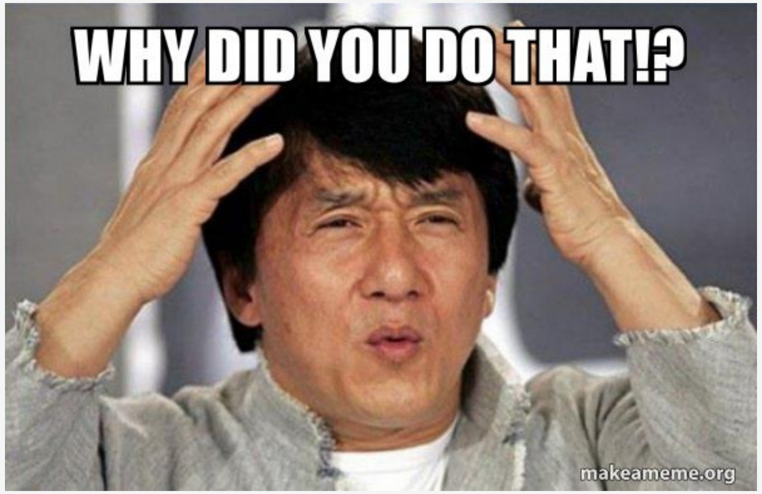
Challenges with going end to end

- My take on key challenges
  - Computational and data cost
  - Dependency on data and task formulation
  - Explainability and Interpretability
  - Bias, guarantees, and robustness

Explainability and Interpretability

- Interpreting feature-based models
  - Feature values (“v1agra”) + weights = prediction (“spam”)

- Interpreting end-to-end neural networks
  - Feature values (300d dense vector)
  - weights (input, forget, output gates)
  - different types of nonlinearity

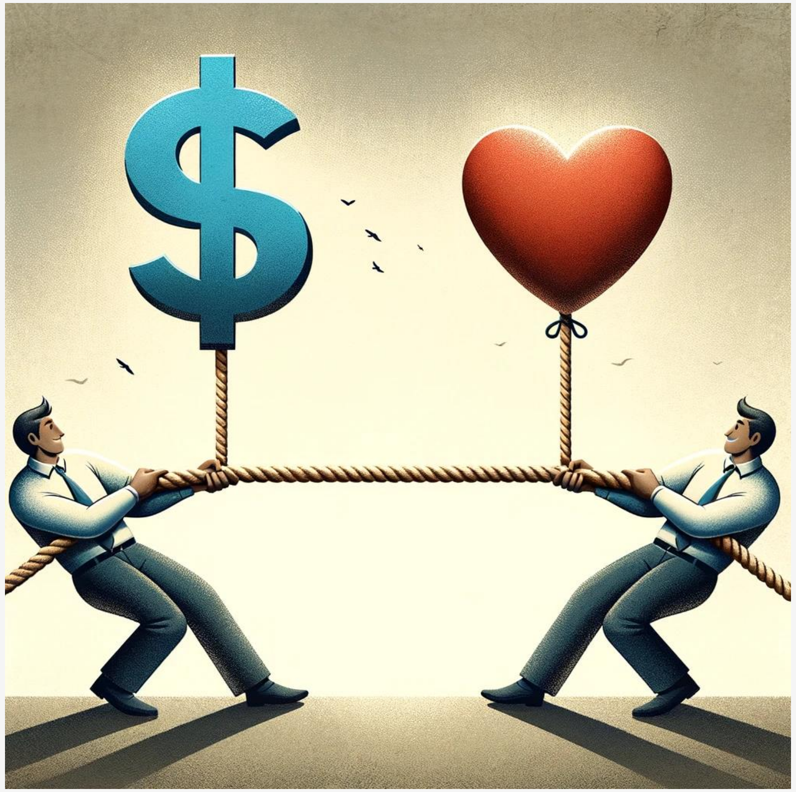


Explainability and Interpretability

- Provide a (valid) justification for the model behavior
- Provide a faithful explanation of the model behavior
- Provide an explanation that is useful for a human
  - To assess the model
  - To learn how to perform the task
  - In a Human-Computer collaboration

Bias, Guarantees, and Robustness

- An end-to-end neural network finds the (mathematically) optimal solution to a formally defined problem
- Sometimes the optimal solution can lead to undesired behavior
  - bias with respect to race, gender, religion, sexual orientation
  - “shortcuts” to solving tasks
- How do we guarantee the model is consistent and bias-free?
  - Evaluation and algorithmic fairness
- How do we know if the algorithm is safe from adversarial attacks?



What networks do we know so far

- Feed forward networks (FFN)
- Recurrent neural networks (RNN) (+ LSTM, GRU)
- Convolutional neural networks (CNN)

- Pop quiz: are these networks for supervised or unsupervised NLP?

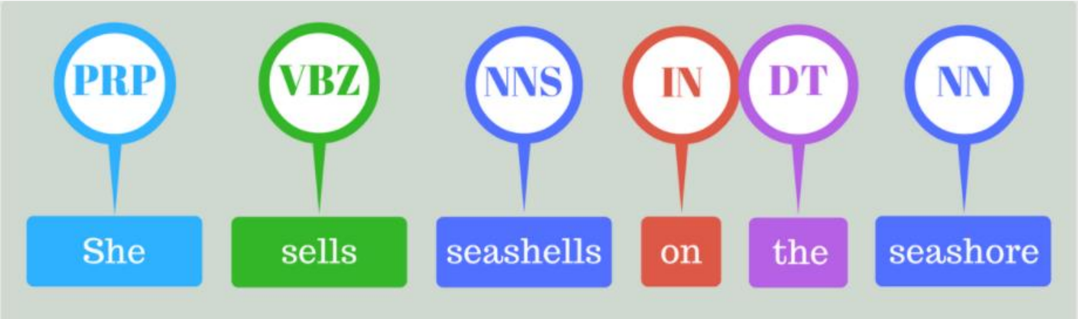
Encoder-Decoder Models

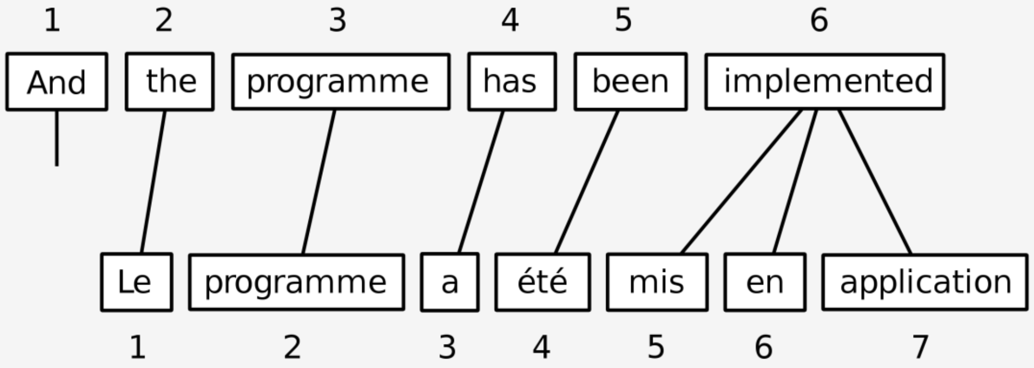
Input and output in NLP tasks

- What is the input and output of the following tasks
  - Sentiment analysis
  - Automated fact checking
  - Clustering documents based on topic
  - Machine translation
- How many possible outputs does each of those tasks have?

Sequence labeling vs sequence-to-sequence

- Consider the following two tasks
- What is similar between them?
- What is different?
- How would you approach each of them?



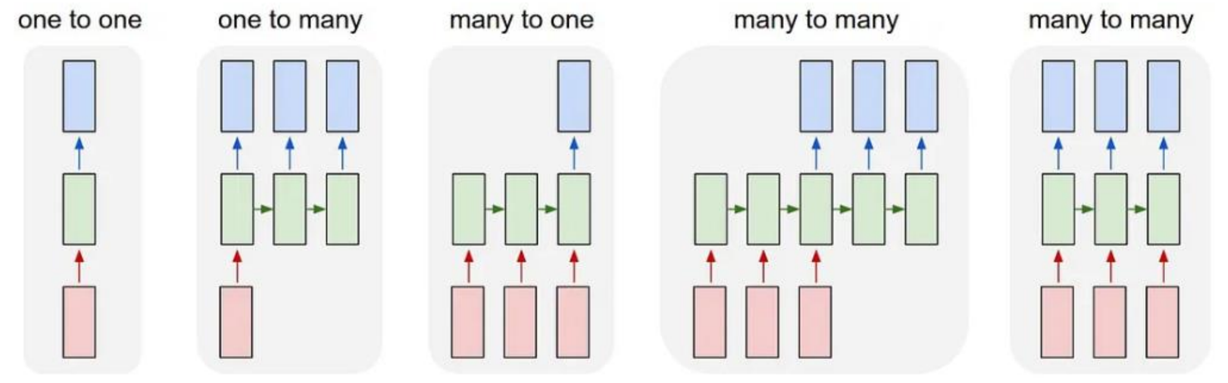


Key differences

- Same length vs different length
- One-to-one alignment vs no one-to-one alignment
- Local dependencies vs long-distance dependencies
    - Within the output
    - Between the input and the output

Different task formulations

- Which of the following images corresponds to:
  - FFN
  - RNN for text classification
  - Machine translation
  - Image captioning
  - Sequence labeling



The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy

Encoder decoder

- We use a model family called encoder-decoder
- Simple idea
  - Encoder "represents" the source (e.g., English)
  - Decoder "generates" the target (e.g., German)
- Can you suggest tasks that can use encoder-decoder?

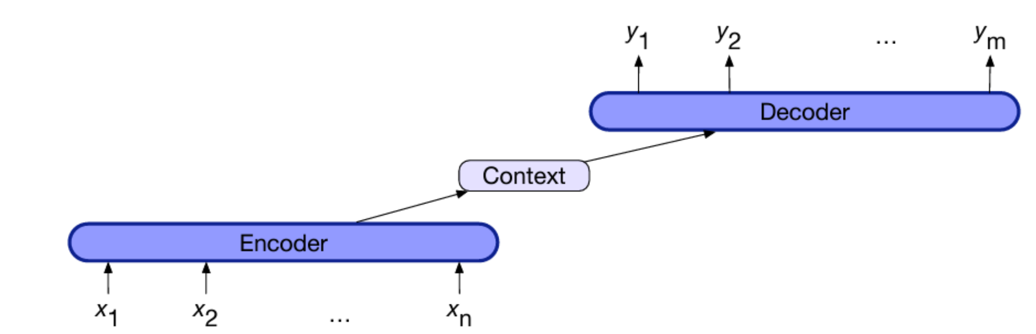


Fig 9.16

Usage of encoder decoder

- General usage of encoder decoder
  - Mapping between data of different format, size, and structure
  - Encoder-decoder vs sequence-to-sequence
- Examples for tasks that can use encoder-decoder:
  - Machine translation
  - Text summarization
  - Question answering
  - Image captioning

How do we implement encoder-decoder?

- Can you propose a way to implement enc-dec model with what we know so far?
- How do we encode the input?
- How do we decode the output?

Single RNN as encoder decoder

- Let’s consider a single RNN for the task

- Add a separator between the two texts:

  - [sentence] [in] [English] [SEP] [sentence] [in] [German]

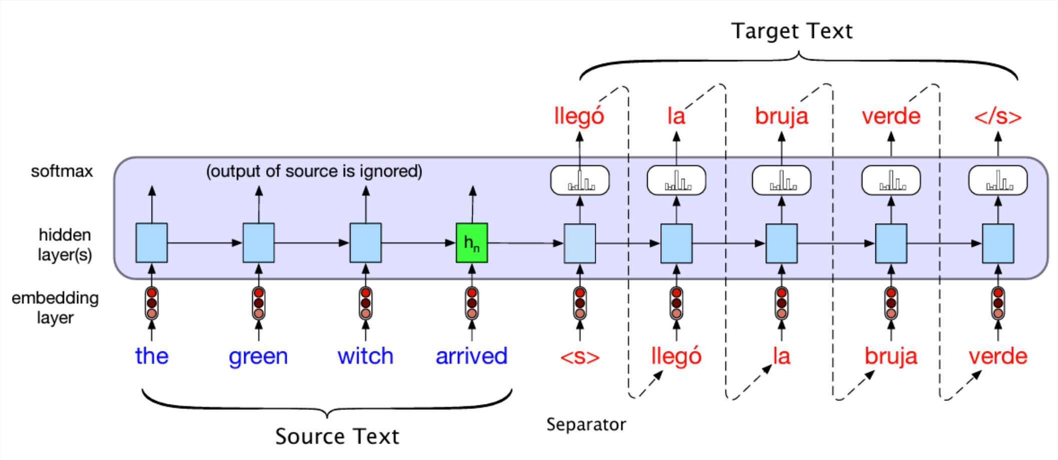
- The hidden state at SEP will contain all the information about the first sentence

Conditional generation

- How does a traditional language model generate text?
- How does an encoder-decoder RNN generate text?
- Does that concept look familiar?

A single RNN as encoder-decoder

- Consider using the following model
  - We use “English” as a “prompt”
  - Hidden state at <s> “encodes” the text
  - We generate Spanish step by step from x and h



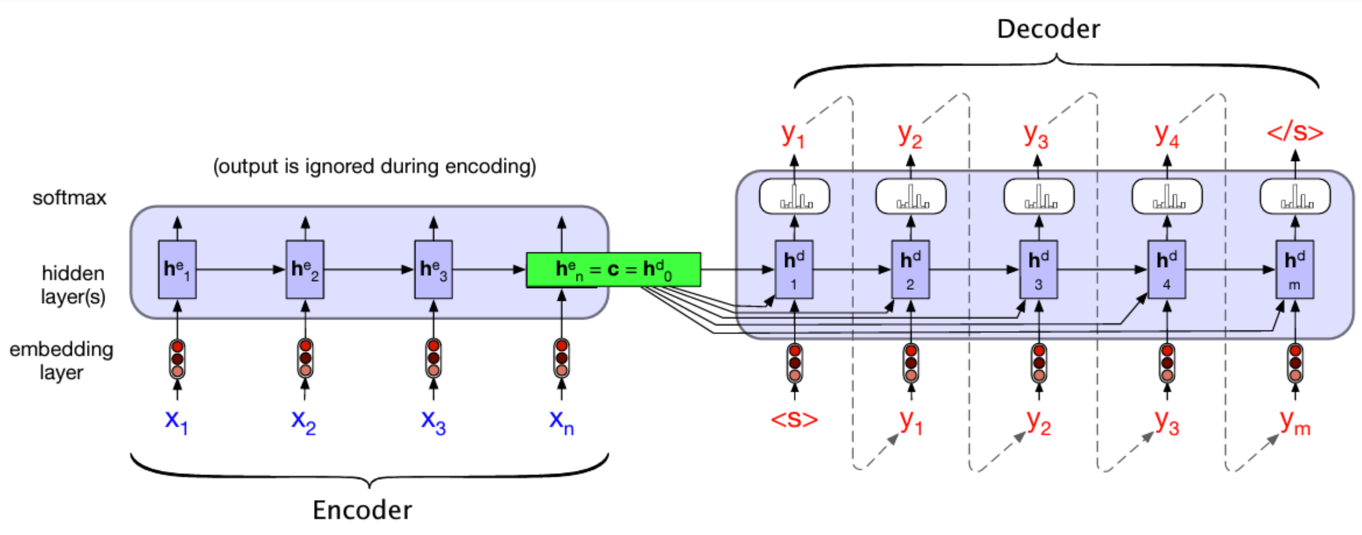
- What would be some problems with this model?
  - What if the task was text captioning?

Using separate RNNs for encoder and decoder

- Train two models
- Pass the context at every step

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

- Can you point a potential problem?
  - What could improve this architecture?



- What is the purpose of the encoder?
  - Should it be able to generate?

Formal representation of RNN based decoder

- The context is the last h of the encoder
- The hidden stage at step 0 is just the context
- For every step after 0, we use both h and c
- We use the hidden state to predict y at time t
- Why is there a "y" at the calculation of the hidden state h_t^d?

$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}
$$

use the value from the previous state

3 hidden state:

- context
- word embeddings
- previous state.

Training encoder-decoder models

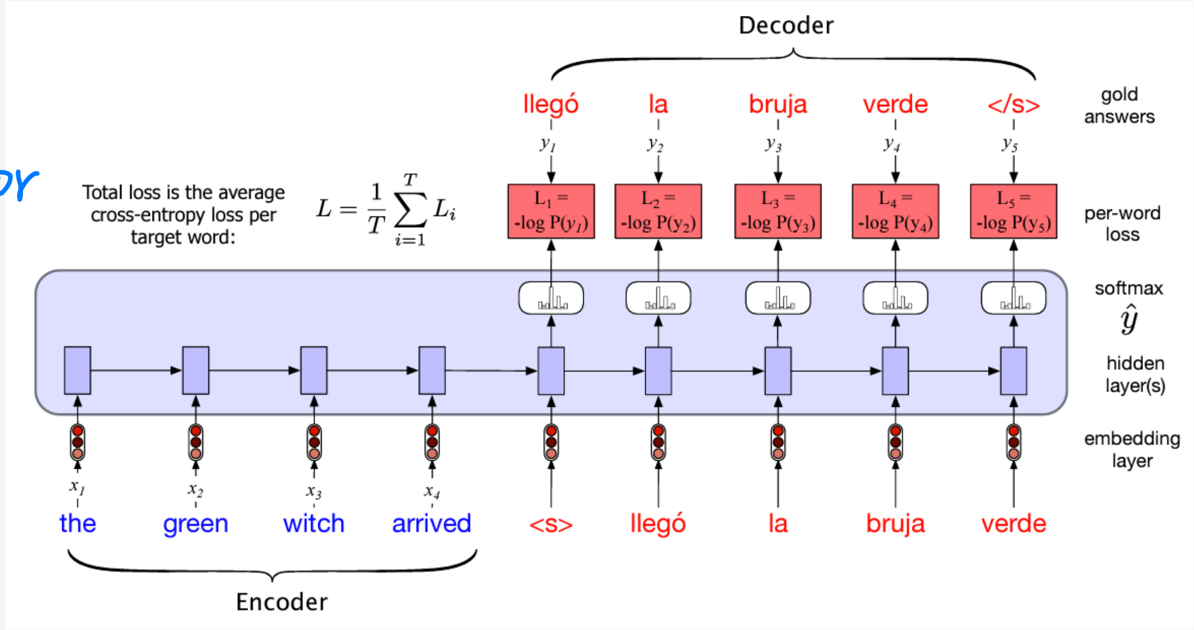
incorrect.

correct

- Models are trained end-to-end
- Encoder is trained through hidden layers

if you get a word, calculate the error and mitigate them

- Decoder is trained through teacher forcing
  - Remember "teacher forcing"?



Why are encoder-decoder models important?

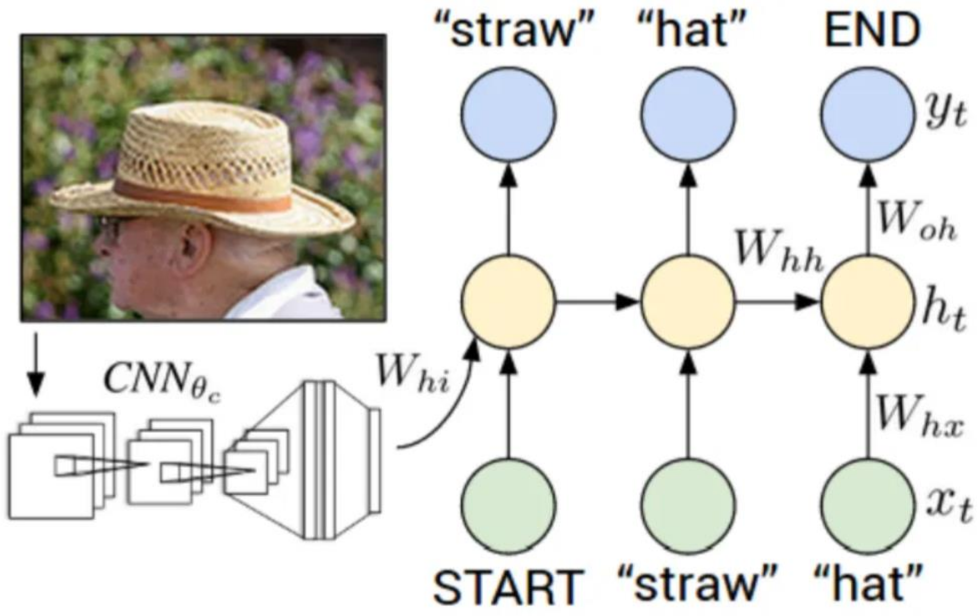
- Instant improvement on machine translation
  - Google Translate switching to NMT
- Key concepts reused (and giving raise to) Attention and Transformers
- Bridging the gap between modalities

Encoder decoder across modalities: image captioning

- The encoder and decoder “talk” via the context

- They don’t have to be the same type of model

- The modalities don’t have to match
    - Speech to text
    - Image to text

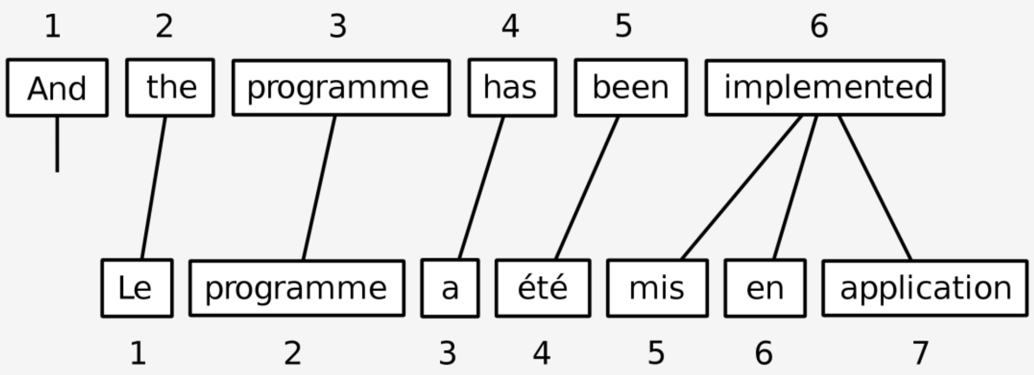


Deep Visual-Semantic Alignments for Generating Image Descriptions

Attention

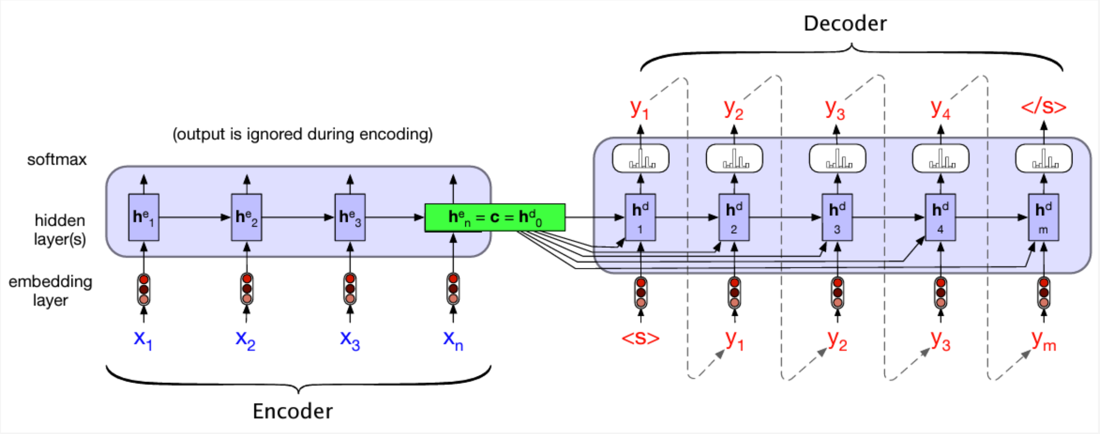
A bottleneck of RNNs

- Consider the problem of MT and an encoder-decoder solution



- The "context" is the information from the input that we need to generate the target

- To generate word y_t, we use the prior information for $y_1 – y_{(t-1)}$ and the same c



- Should c be the same for every word?

Attention – intuition and restrictions

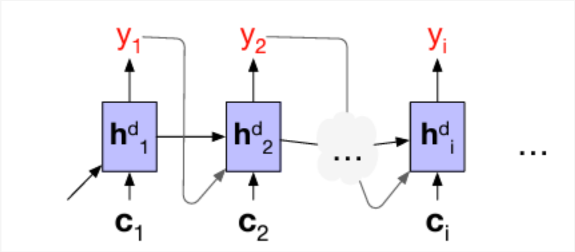
- Intuition: each token in the target should use a “personalized” context

- Access all the hidden states in the encoder
  - Still needs to have a fixed length, regardless of variable input length

- Any ideas how we can do that?

Attention – basic implementation

- Weighted sum of all encoder hidden states
  - Calculated separately at each decoder step
  - Using the hidden state at (t-1)



- Dot product attention
  - Calculate the similarity between $h_{(t-1)}$ and each encoder state h^e
  - Use the similarity scores to calculate the weighted sum

Dot product attention (formally)

- Scoring function: $$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_j^e$$

- Weight vector: $$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e)) \\ &= \frac{\exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e)}{\sum_k \exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_k^e))} \end{aligned}$$

we don't use softmax

- Personalized context: $$\mathbf{c}_i = \sum_j \alpha_{ij} \, \mathbf{h}_j^e$$

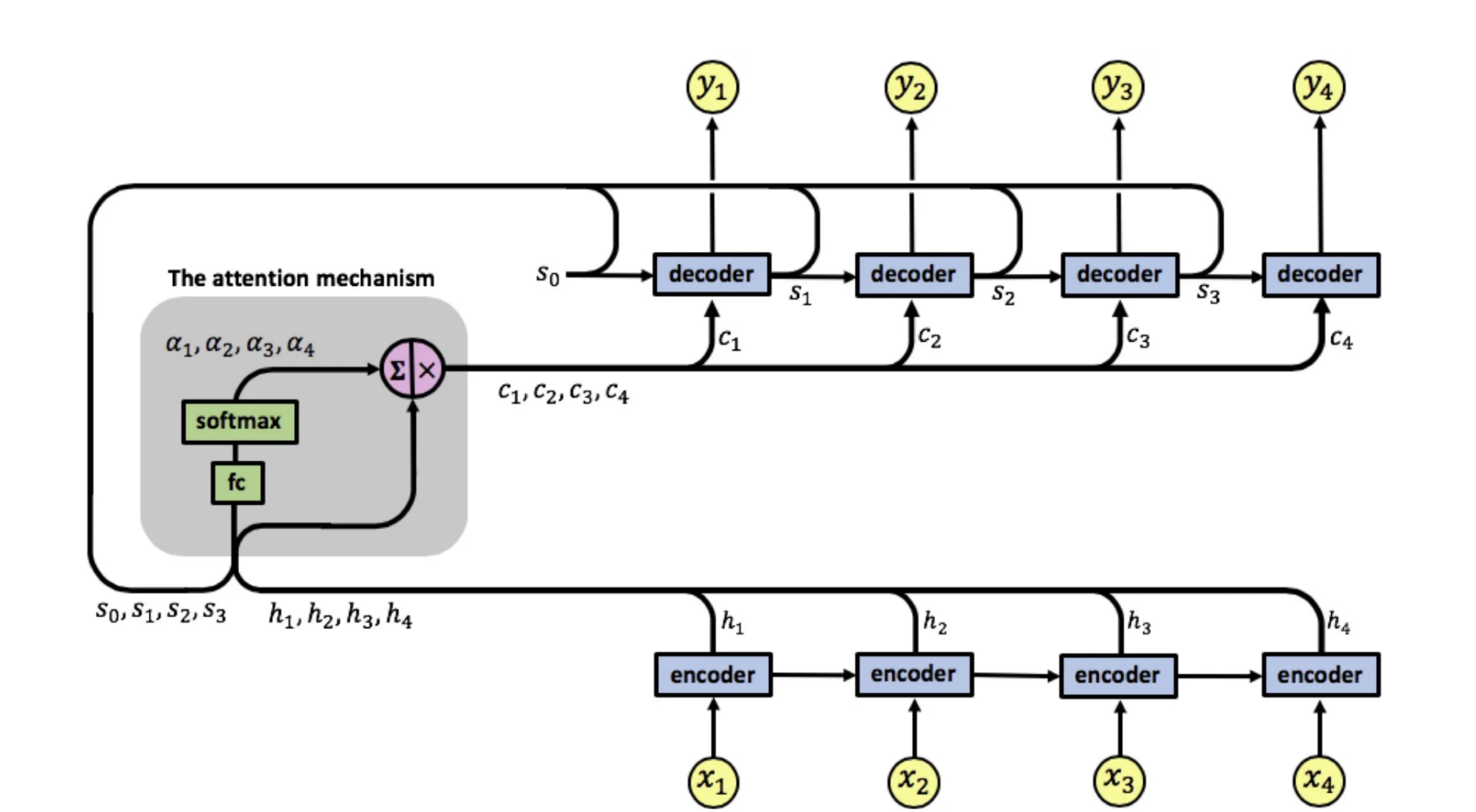
- More complex scoring functions:

$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{t-1}^d \mathbf{W}_s \mathbf{h}_j^e$$

Visualization of RNN with attention

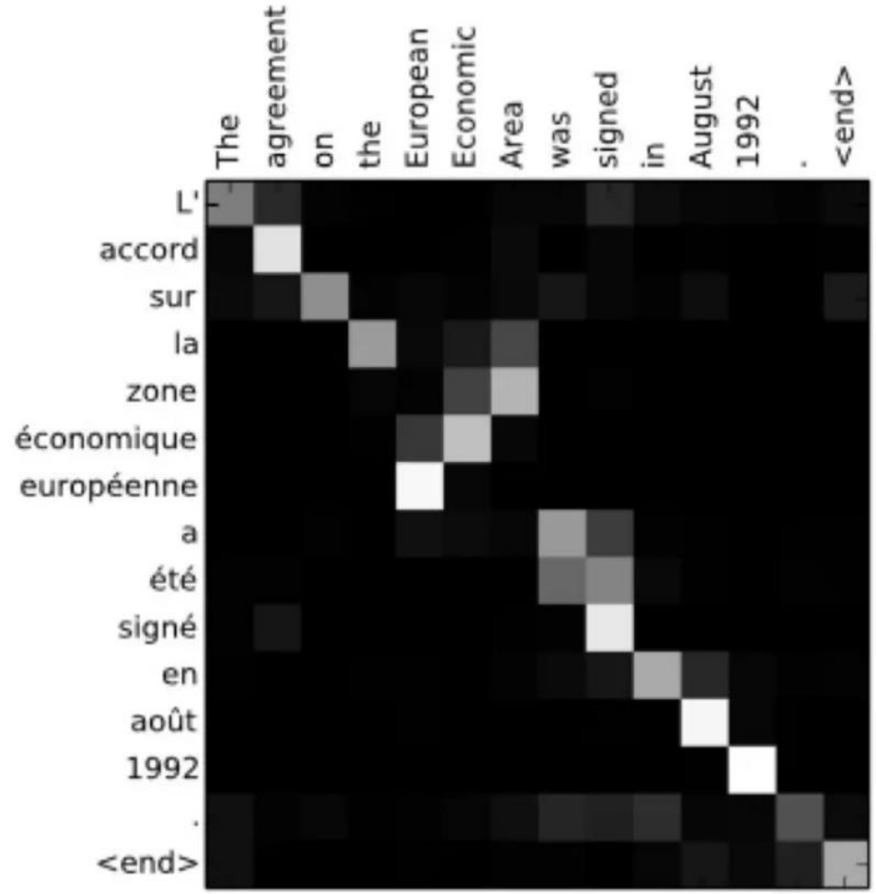
- RNN with attention

- Attention is learned via a simple FFN



Visualizing attention

- Linear weights are interpretable
- We can see which word is more important
- Can we use attention for explainability?



Attention is all you need
The original Transformer

Training vs Finetuning

- Simple end-to-end models are trained for a single task

- Word embeddings can be reused, compositionality is learned

- Transfer learning has limited capabilities
  - From similarity to inference
  - From emotion to sentiment

Need for powerful transfer learning models

- Generic representation framework
  - Represent (contextual) word meaning
  - Represent interactions between words
  - Capture different types of meaning and interactions
- Easy to adapt to new tasks with minimal adjustment
- Looks familiar?
  - Many of the problems and RQs remain the same, just the context changes

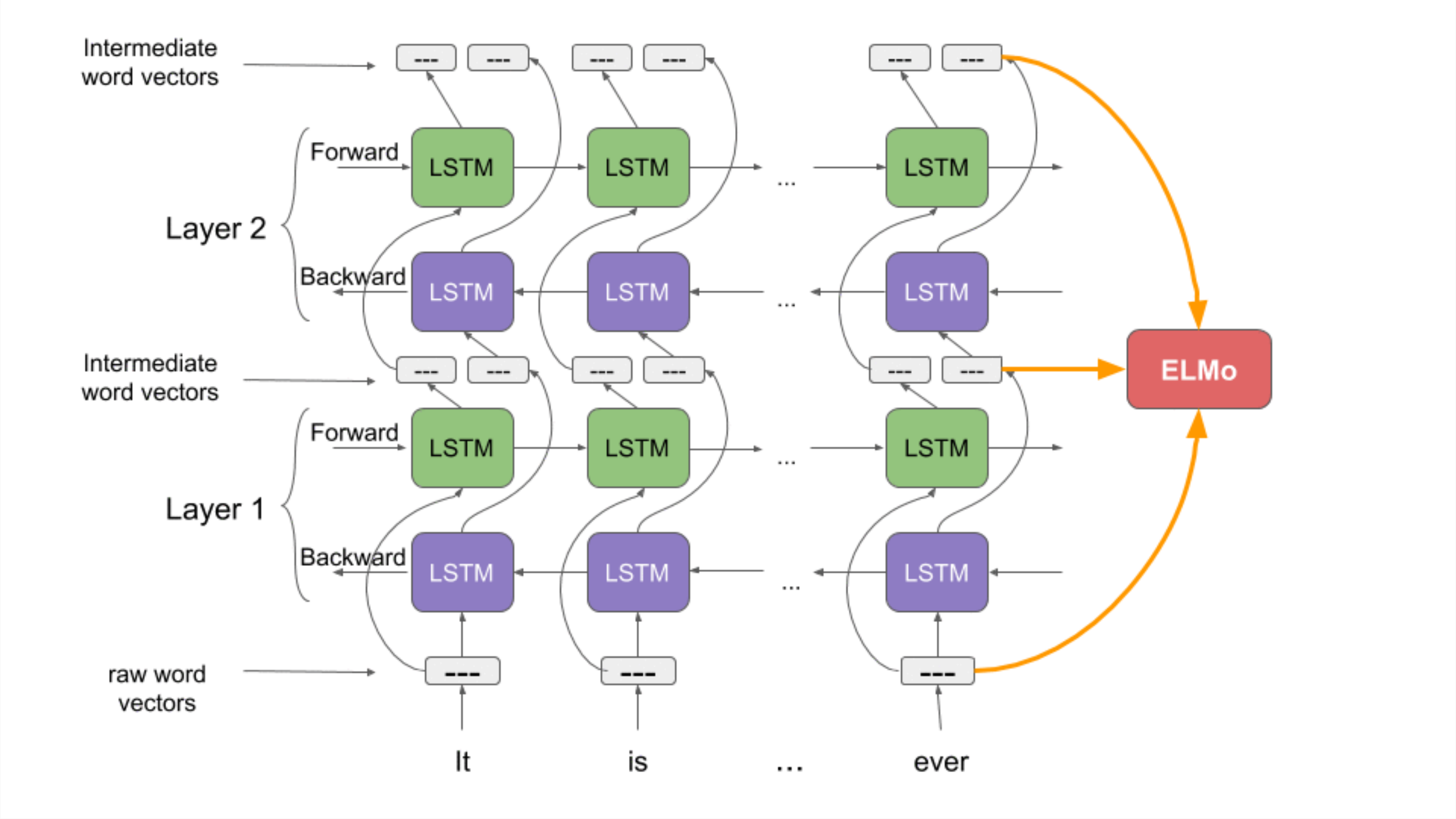
Look back at ELMO

- ELMO embeddings meet most of those expectations
  - In-context meaning
  - Interactions between words
  - Deep representation capturing different relations
  - Task specific weight learning

- Pop quiz: how did ELMO embeddings work?

Elmo architecture

- How can we improve over that? using Attention

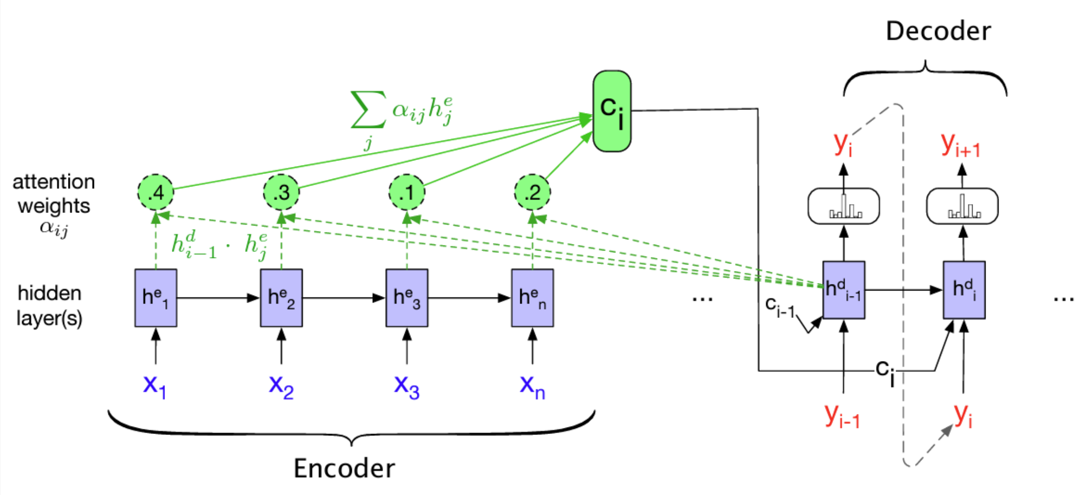


Self attention

- Attention works better than RNN/LSTM for encoder-decoder models

- Can we use attention for a standalone network?

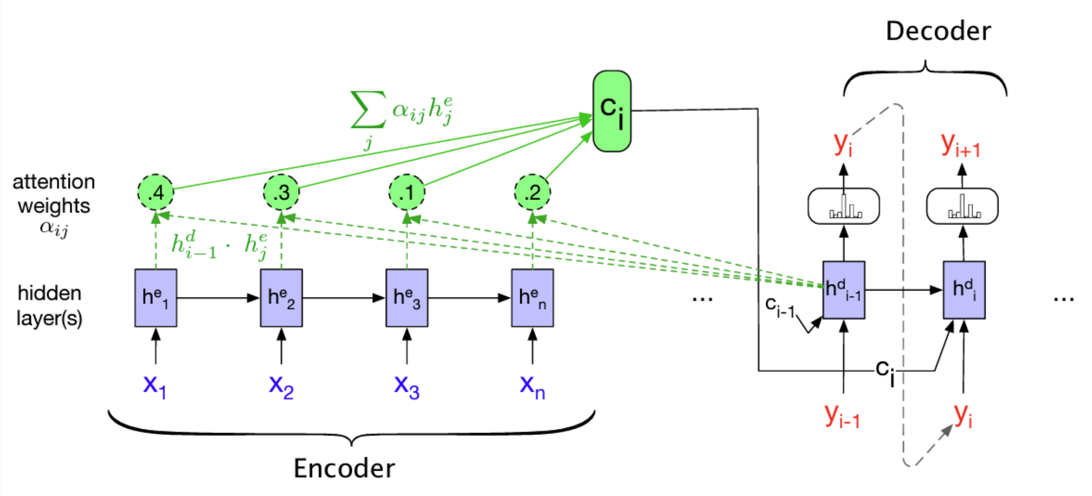
→ use self attention

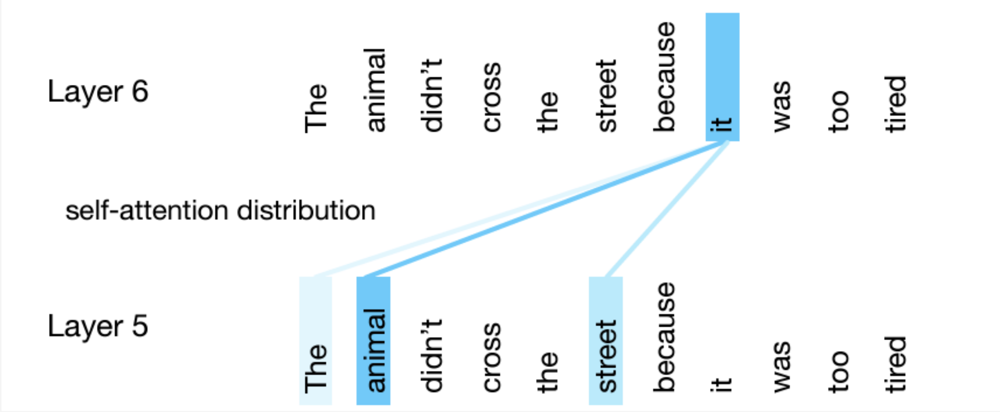


Self attention (2)

- Self attention is a key concept in building transformers

- It applies the same approach as attention in encoder-decoder, but on itself





Causal attention vs bidirectional attention

↙ more used for classification.

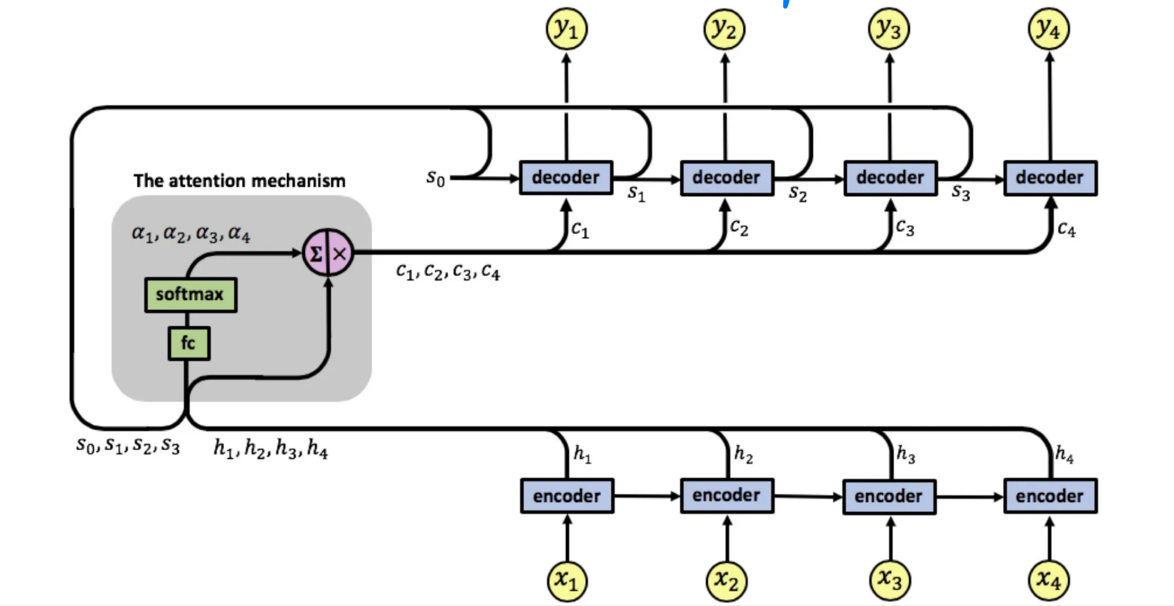
- In encoder-decoder attention, the attention is the weighted sum of all hidden states of the encoder

RNN can't handle long sequence (they do, but compare to LSTM)

- Which hidden states do we use in self attention?
  - Why?

· bidirectional attention cannot look into the future

→ time efficiency is not a problem



Causal self attention

- Causal self-attention is used in models like GPT

- Two key properties
  - Only calculated using words in one direction (left for european languages)
  - Each representation at a layer L is calculated independently of the others

- How does this compare to RNNs and LSTMs?
  - Why are these two properties important?

RNN has to compute hidden state sequentially.

Pop quiz

- Can a transformer model process infinite input?

No

- Can an RNN be (natively) parallelized?

No

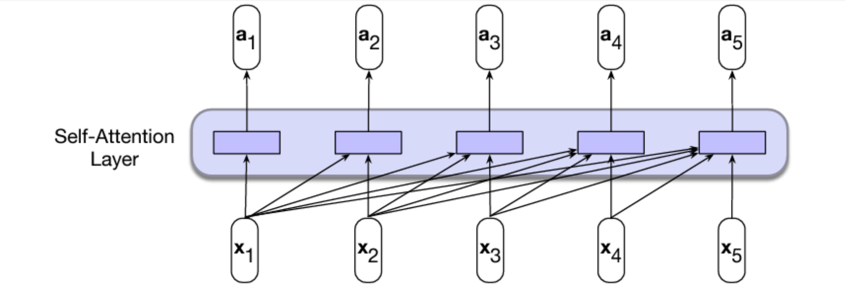
Causal self attention (intuition)

- Similar to RNNs, we have a 1:1 input-output mapping
- Same basic approach as original attention
  - Dot product + softmax + weighted sum

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \;\forall j \leq i \end{aligned}$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

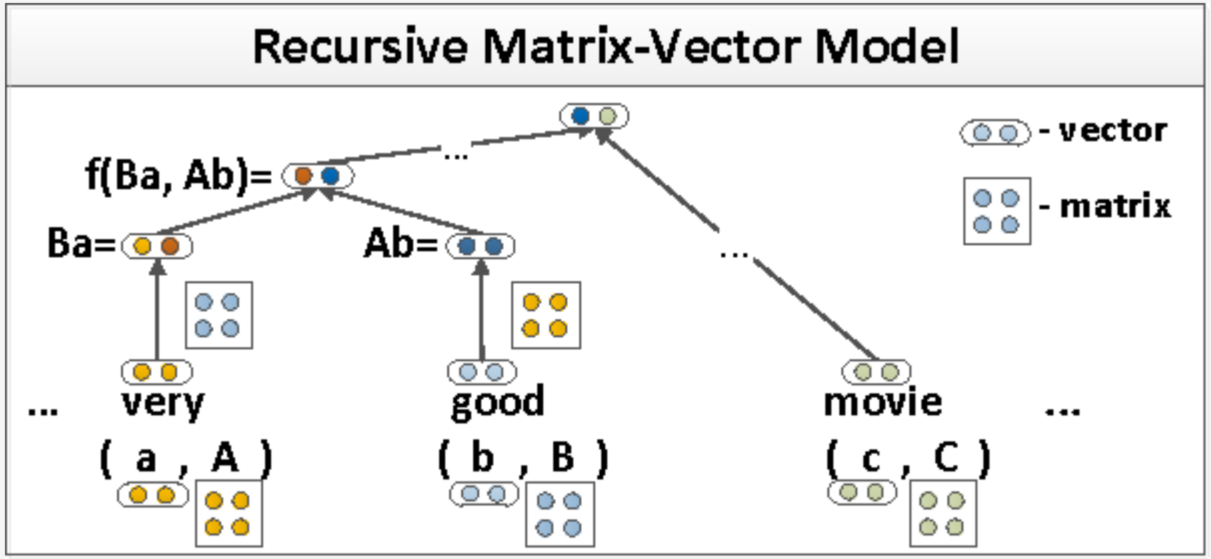


- Which is the most similar token to x_3? What is the input to the first hidden layer?

$\hookrightarrow$ X_3 itself

The query, key, value

- You can use a simple dot-product attention
- Transformers introduce the "query, key, value" concepts
- The key ideas:
  - Tokens have different roles
  - Meaning has different aspects
- Consider the following phrases:
  - "The dog is happy"
  - "The toy of the dog"



The query, key, value (2)

- Attention has two key computations
  - Calculate relative weight between vector x_i, and every other vector x_j : score(x_i, x_j)
  - Weighted sum of each vector x_j based on its relative importance to x_i : $$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$
- Three roles a token has in self attention
  - The query: the vector for x_i when x_i is the first element of score(), when we calculate a_i
  - The key: the vector for x_j when x_j is the second element of score(), when we calculate a_i
  - The value: the vector that we use for the scaling and the weighted sum
- Two vectors that define the interactions; One vector that contains the “substance”

The complex nature of meaning and meaning compositionality

- Simple semantic representations assume meaning is static and can be defined with a single function

- Modern algorithms (implicitly) account for complexity of meaning:
  - Vector-Matrix operators in early compositionality
  - Separation between long- and short-term memory (different gates)
  - Deep (contextual) representations
  - Query, Key, Value

Query, Key, Value (formally)

- We learn three different matrices (W^Q, W^K, W^V)
- Every input vector xi is projected to three different representations
  - $q_i = x_i W^Q$; $k_i = x_i W^K$; $v_i = x_i W^V$
- The new formula for score: $\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$
- The new formula for calculating weights: $\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$
- Pop quiz: which token will have the most impact on x_3?

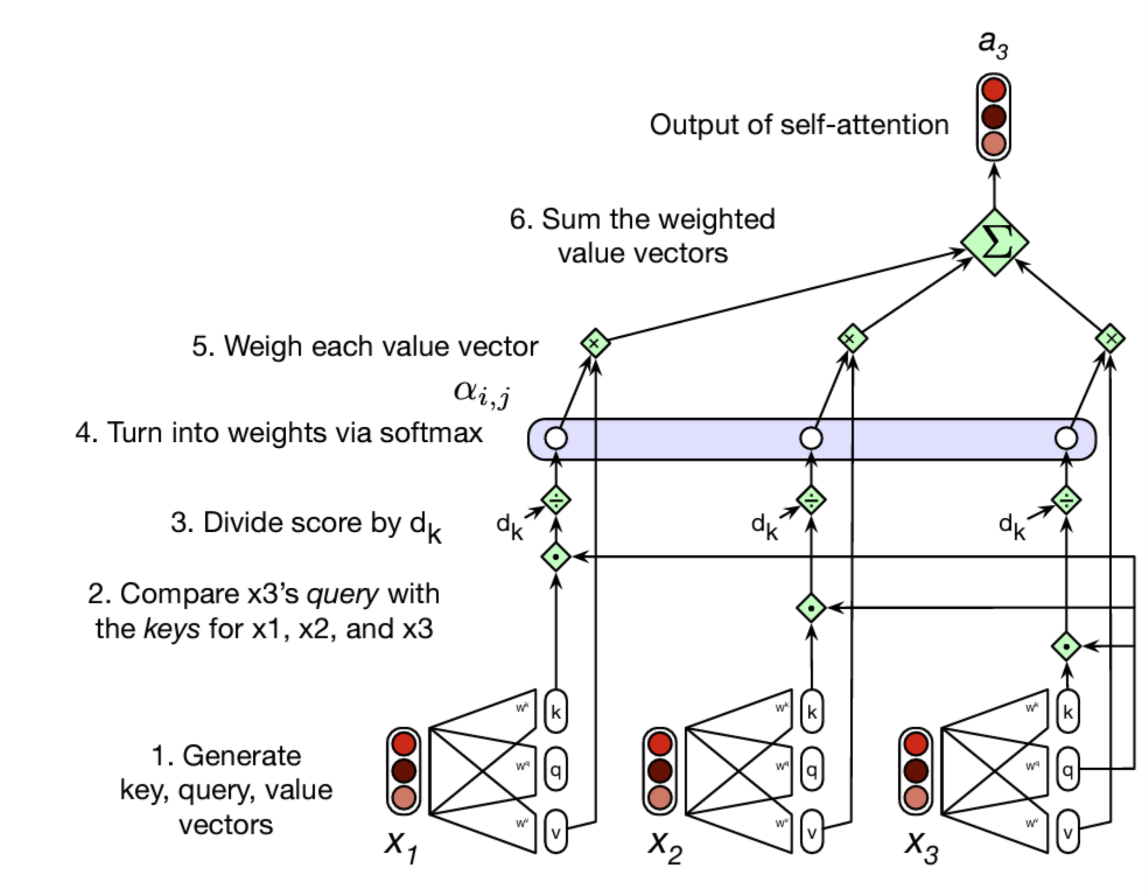
The transformer self attention

1. $\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}}; \mathbf{k}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{K}}; \mathbf{v}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{V}}$

2. and 3. $\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \dfrac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$

4. $\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i$

5. and 6. $\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$



Why are encoder-decoder models important?