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

accumulation of errors are internal problem.

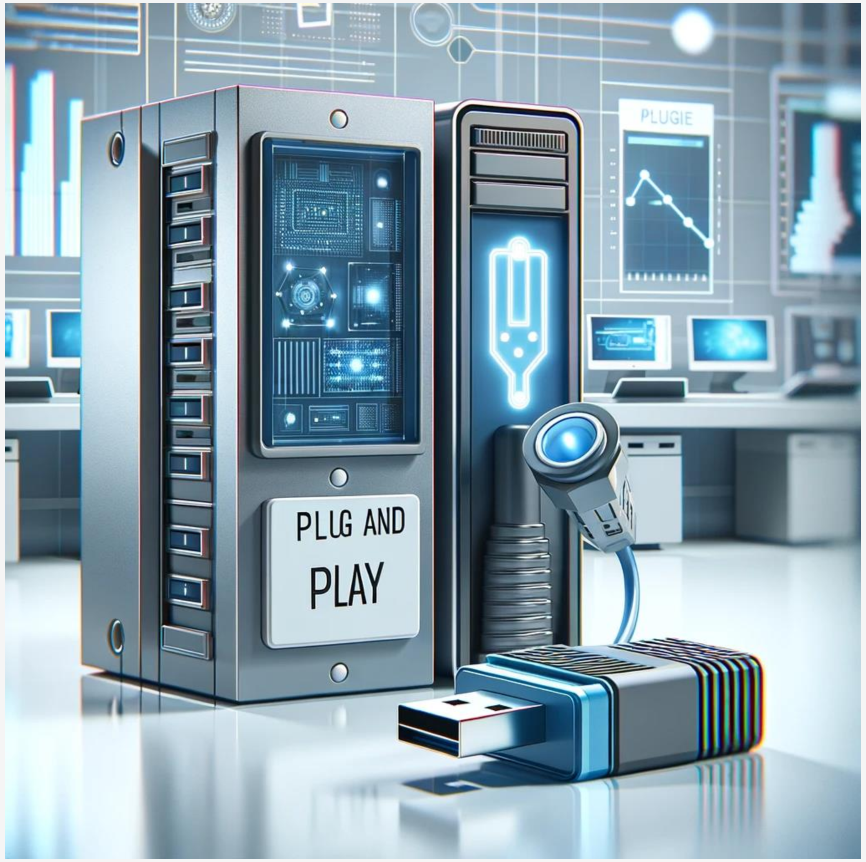
- Map directly from input to output
- No feature engineering
- Trained via backpropagation
  - Data and compute expensive

What are some advantages of end-to-end

- Better performance
- Simpler pipeline

no need of feature engineering.

- 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

expensive.

(accumulation of errors

very easy to overfit

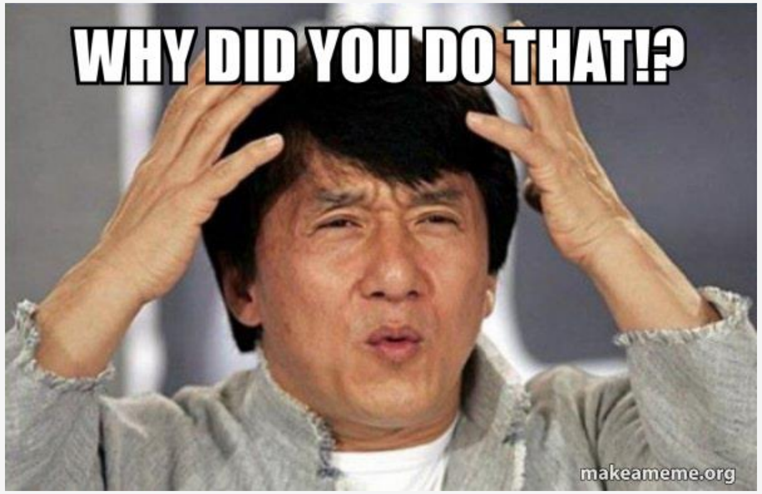
human can understand what's the factor.

Challenges

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

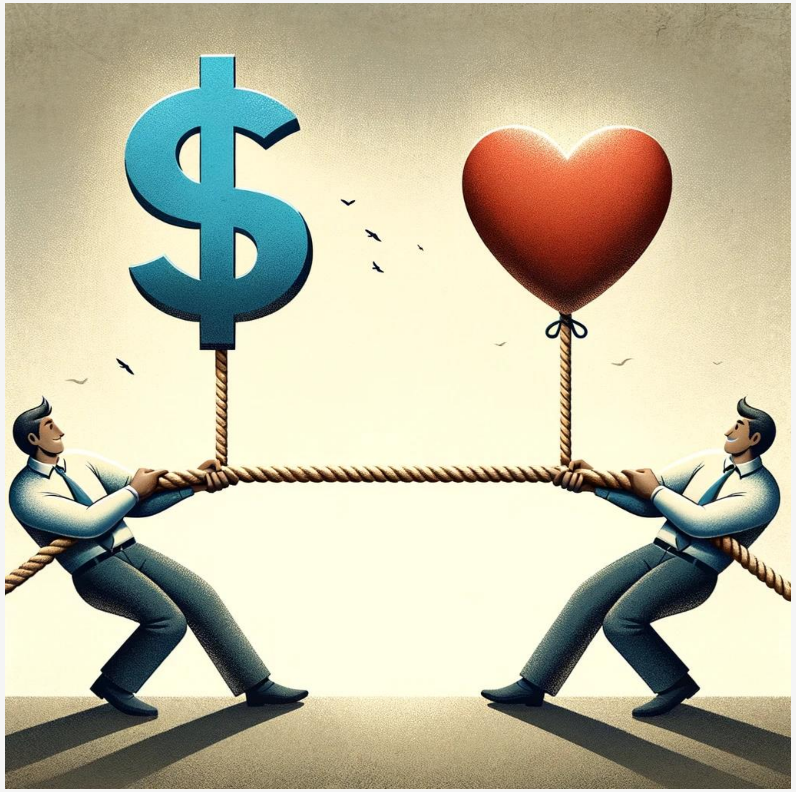
- Provide a faithful explanation of the model behavior

- 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

also

- 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?

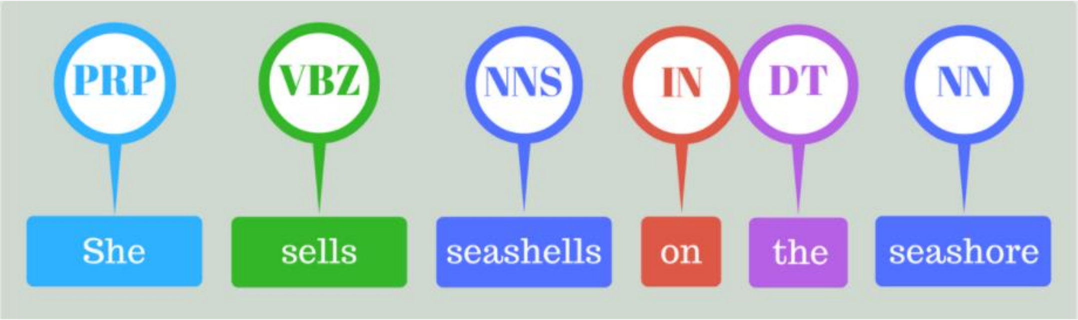
both

Encoder-Decoder Models

Input and output in NLP tasks

- What is the input and output of the following tasks
  - Sentiment analysis bunch of documents → (multiple classes)
  - Automated fact checking
  - Clustering documents based on topic
  - Machine translation (unlimited outputs (all possible combinations))
- How many possible outputs does each of those tasks have?

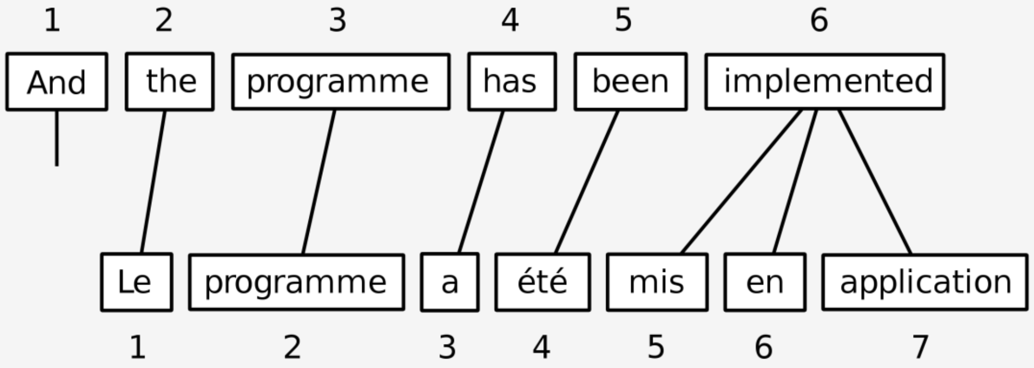
Sequence labeling vs sequence-to-sequence



- Consider the following two tasks
- What is similar between them?

They're both sequential

- What is different?



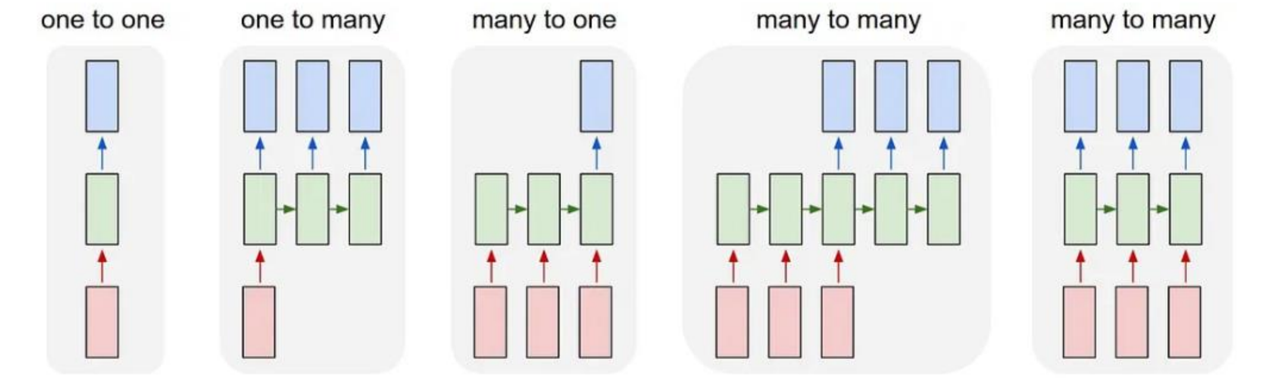
- How would you approach each of them?

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
    many-to one
  - Machine translation
    many to many
  - Image captioning
    one to many
  - Sequence labeling



The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy

Encoder decoder

- We use a model family called encoder-decoder

(not just about sequence-to-sequence)

- Simple idea
  - Encoder "represents" the source (e.g., English)
  - Decoder "generates" the target (e.g., German)

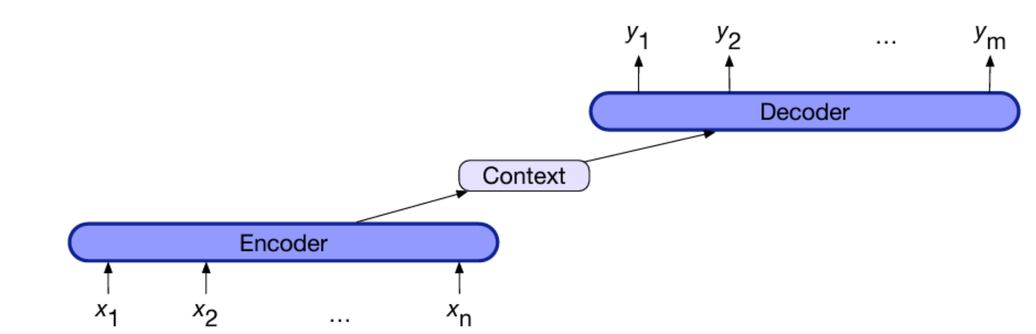


Fig 9.16

- Can you suggest tasks that can use encoder-decoder?

Machine Translation, summarisation, question answering ...

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?

RNN for both. (potentially)

- 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]

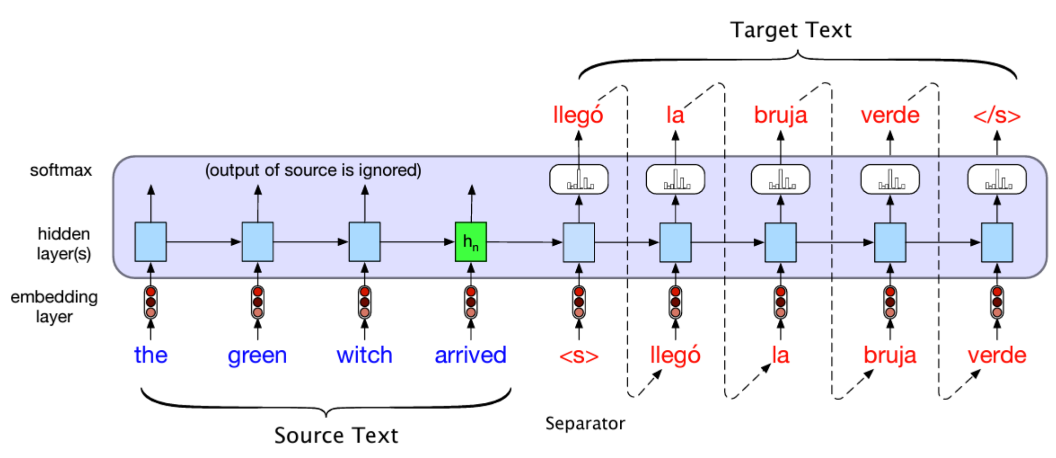
- The hidden state at SEP will contain all the information about the first sentence

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?

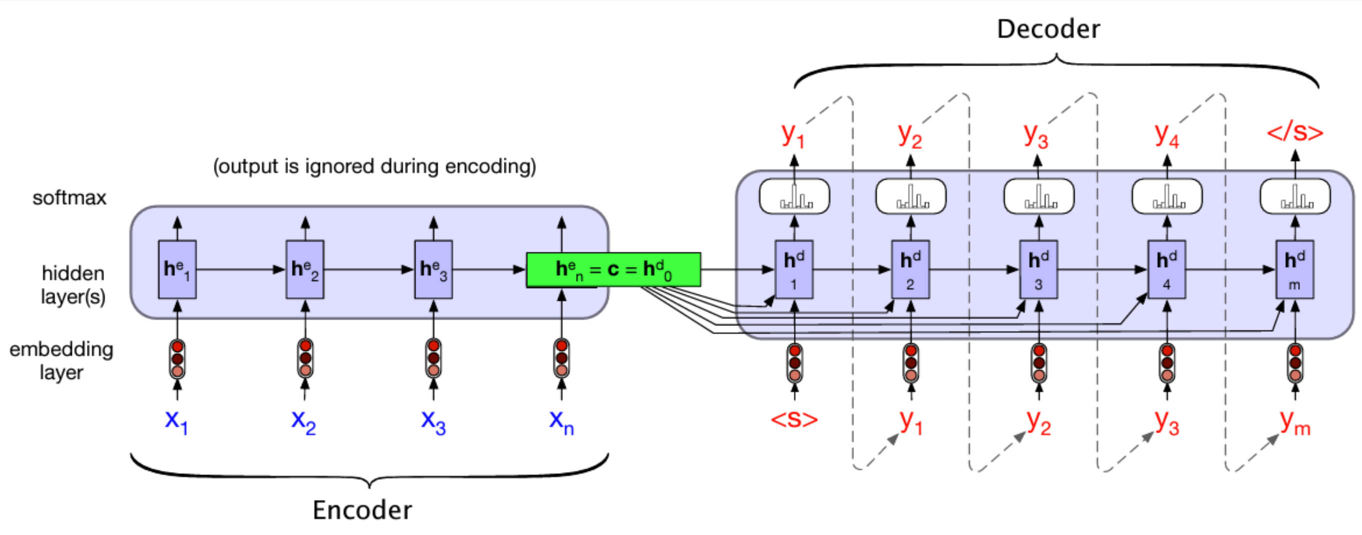
This model might forget the input (no long-term dependencies)

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

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

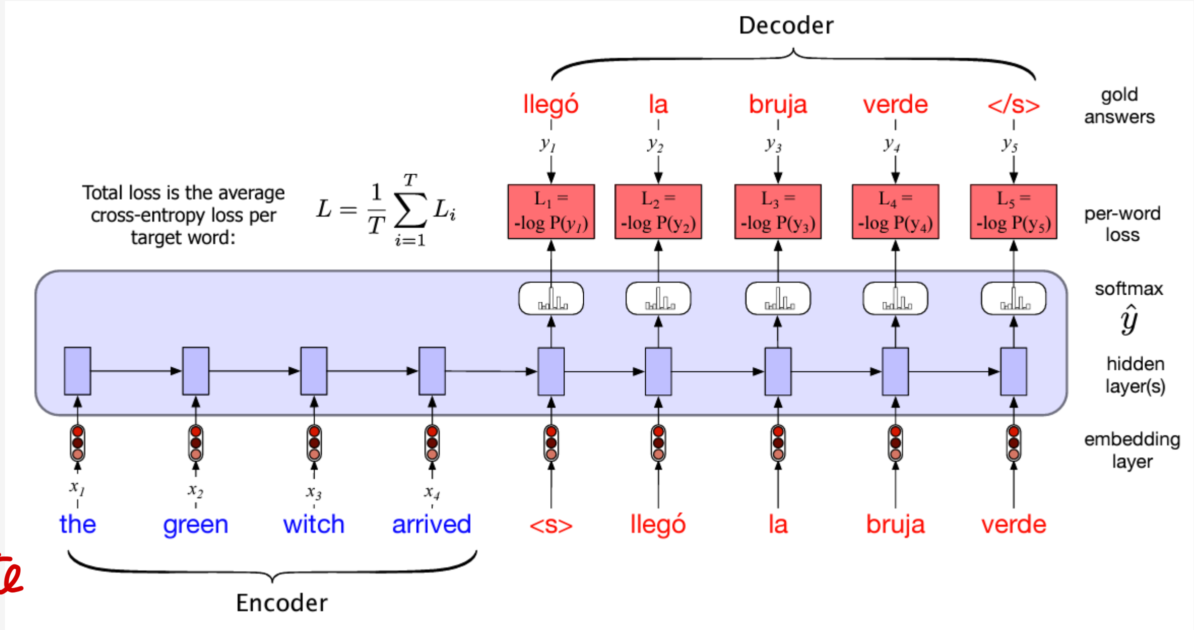
- Why is there a "y" at the calculation of the hidden state h_t^d?

When on Training, there is a text

but, when on generating, no text known so use the output.

Training encoder-decoder models

- Models are trained end-to-end
- Encoder is trained through hidden layers
- Decoder is trained through teacher forcing
  - Remember “teacher forcing”?



In the training, the input of the next state will use true output (token) even if the previous state results in wrong output.

Teacher forcing. Might cause Explosure bias

Why are encoder-decoder models important?

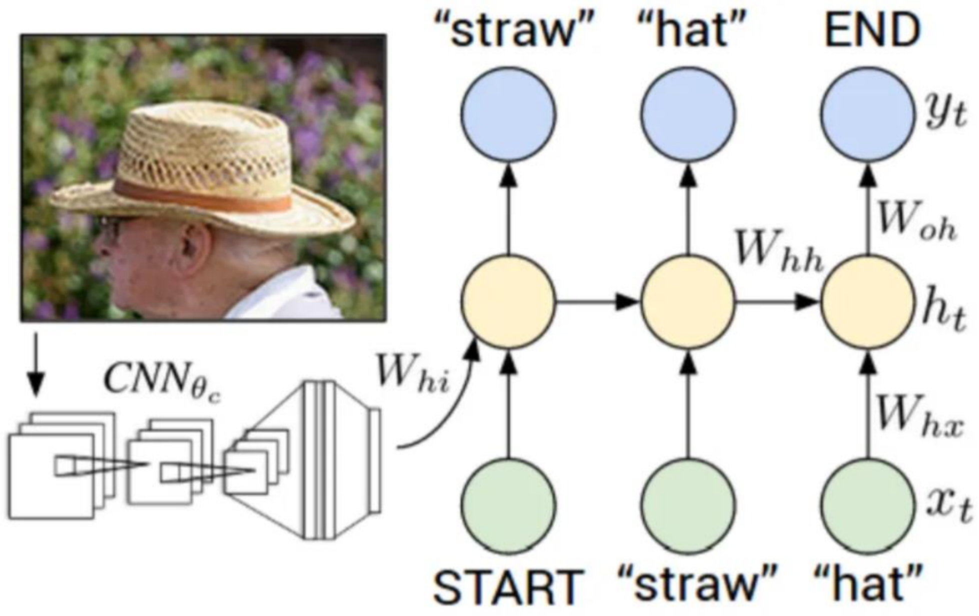
- Instant improvement on machine translation
  - Google Translate switching to NMT
- Key concepts reused (and giving raise to) Attention and Transformers
- Bridging the gap between modalities

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

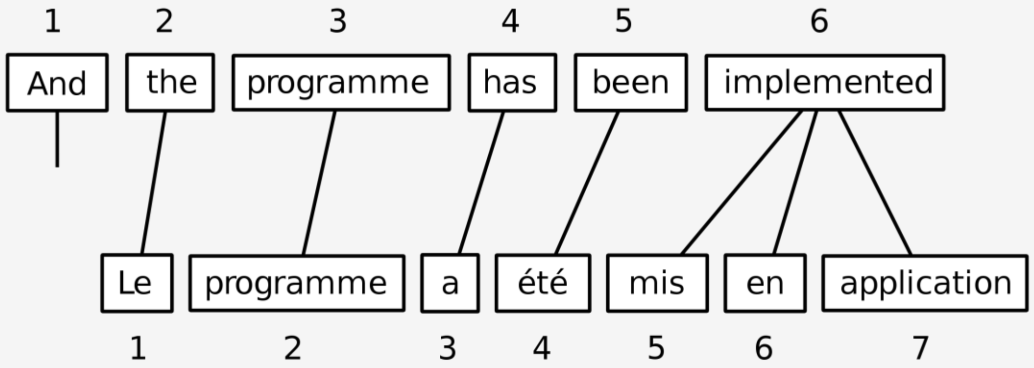


Deep Visual-Semantic Alignments for Generating Image Descriptions

Attention

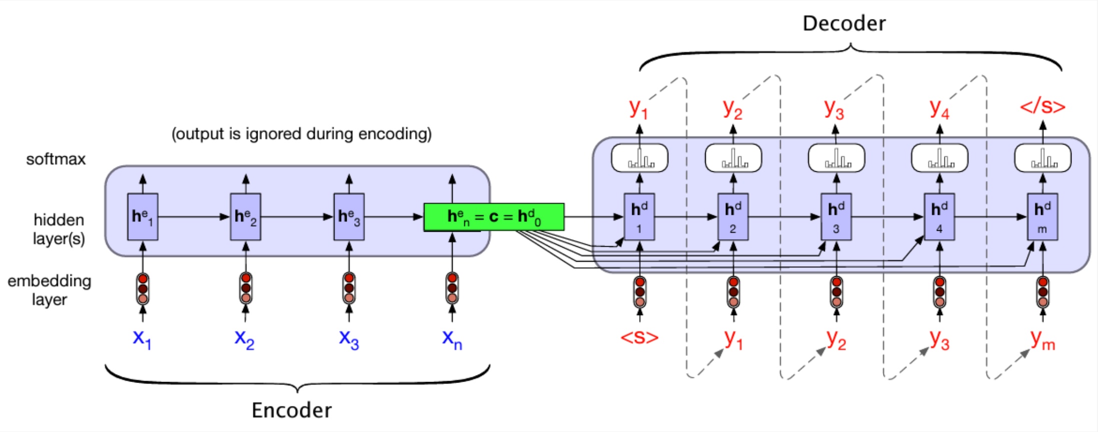
A bottleneck of RNNs

- 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

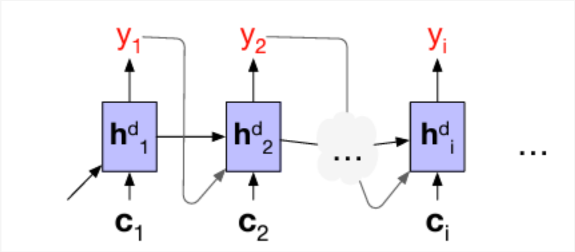
- Intuition: each token in the target should use a “personalized” context

- 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:

$$\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \;=\; \mathbf{h}^d_{i-1} \cdot \mathbf{h}^e_j$$

- Weight vector:

$$\begin{aligned}\alpha_{ij} &= \mathrm{softmax}(\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)) \\ &= \frac{\exp(\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j))}{\sum_k \exp(\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))}\end{aligned}$$

- Personalized context:

$$\mathbf{c}_i \;=\; \sum_j \alpha_{ij}\, \mathbf{h}^e_j$$

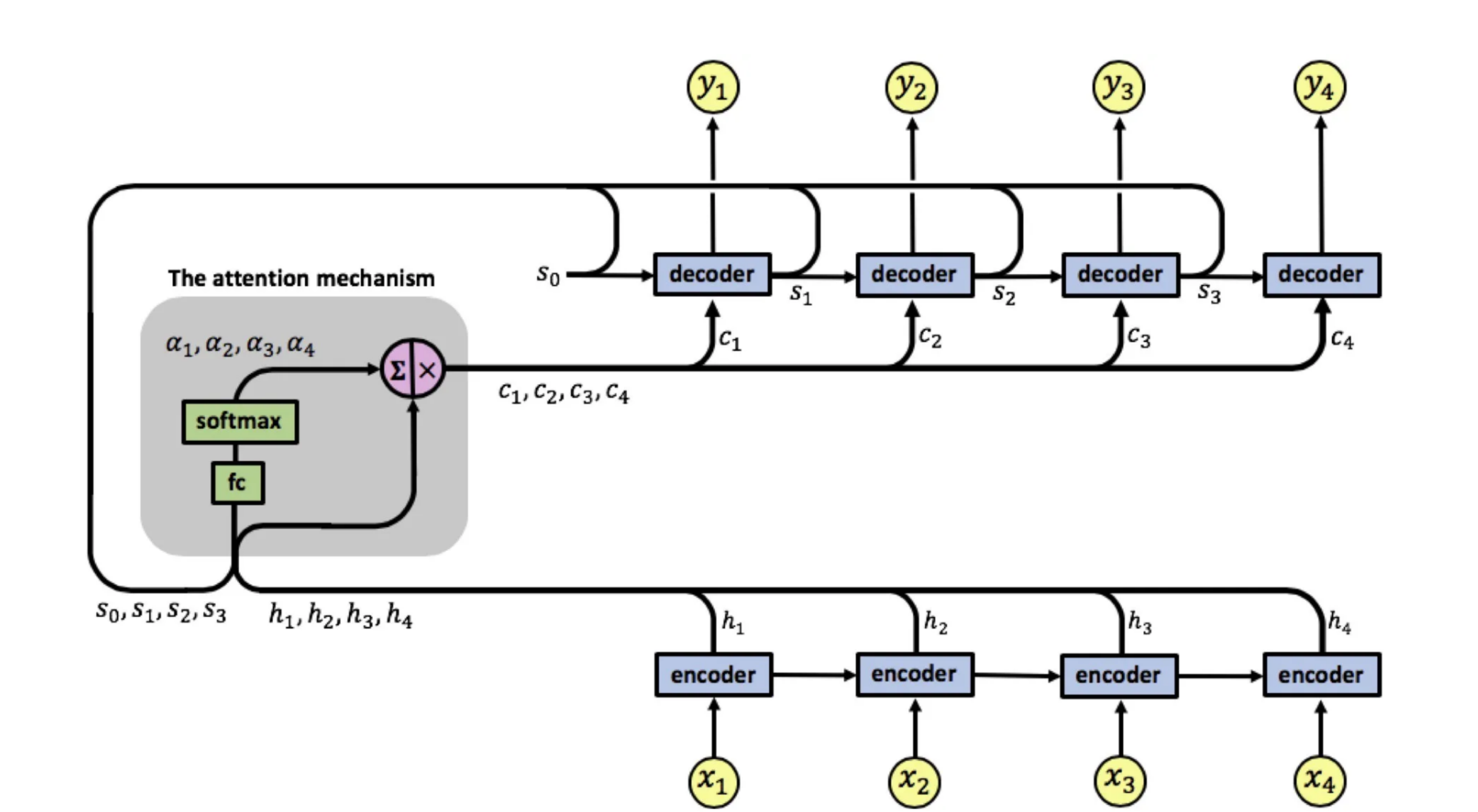
- More complex scoring functions:

$$\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \;=\; \mathbf{h}^d_{t-1} \mathbf{W}_s \mathbf{h}^e_j$$

Visualization of RNN with attention

- RNN with attention

- Attention is learned via a simple FFN

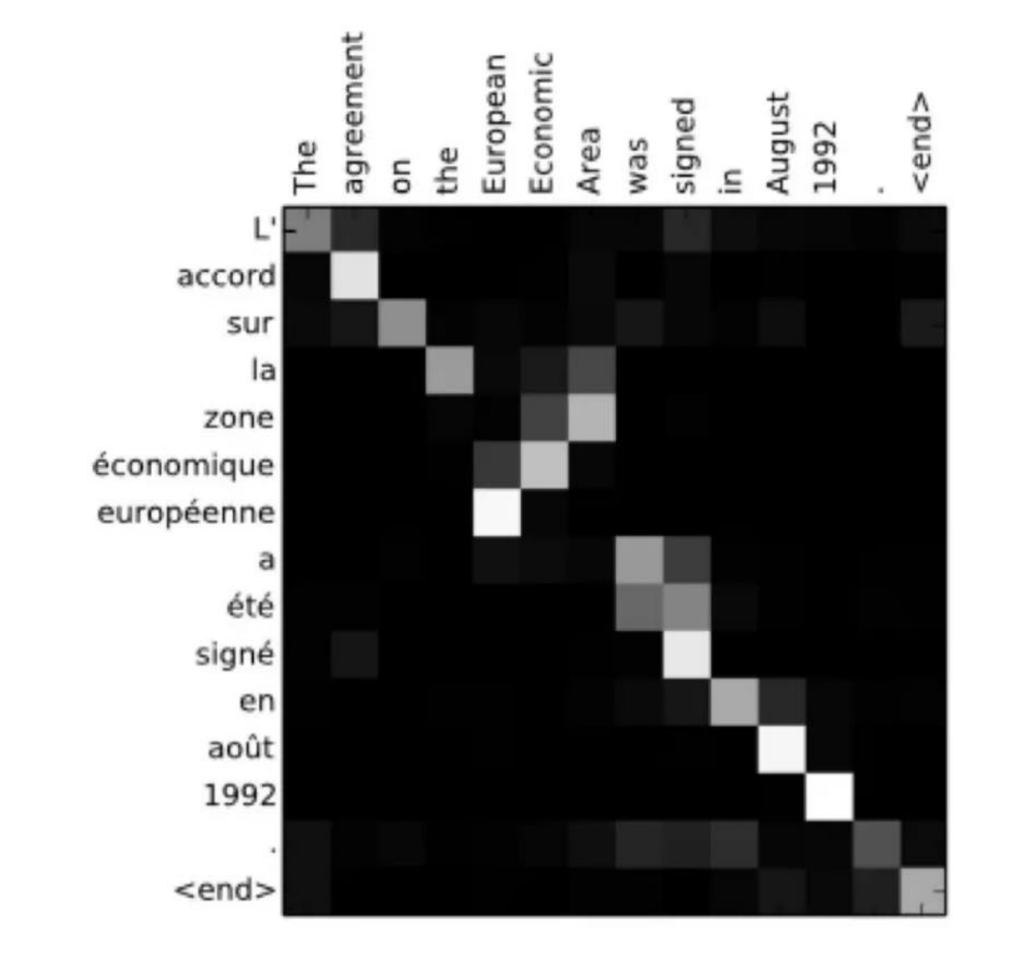


Visualizing attention

- 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

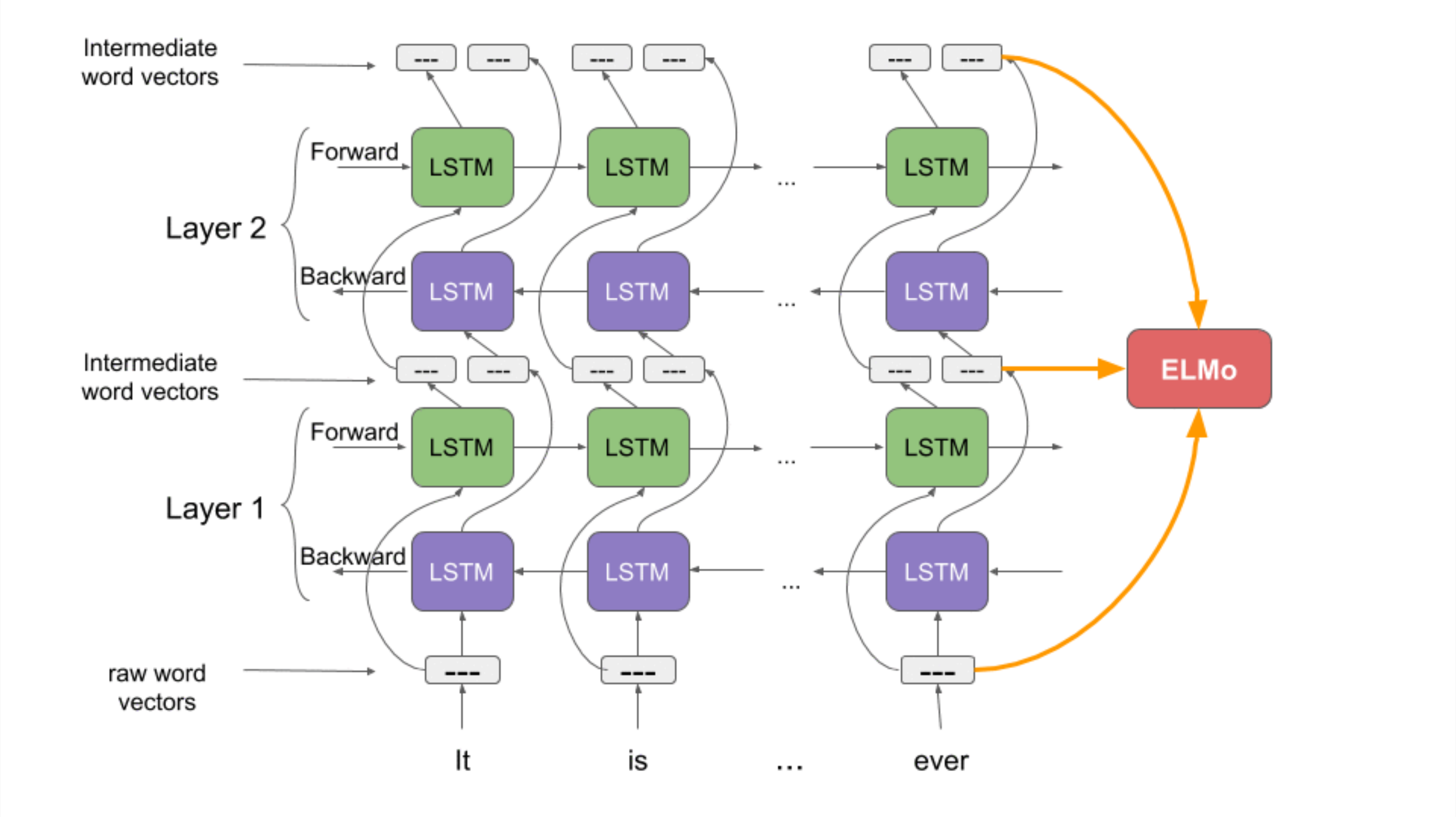
Look back at ELMO – BiLSTM

- 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

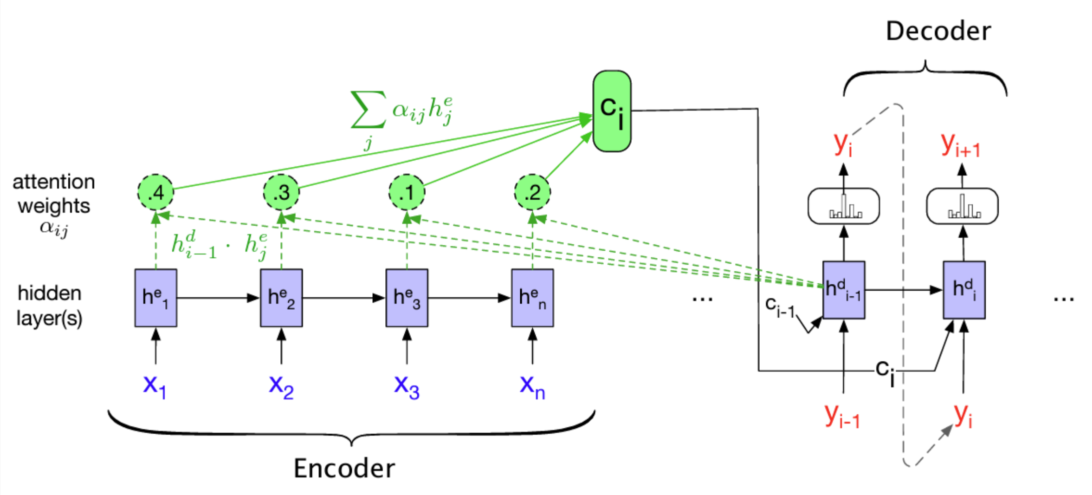
- How can we improve over that? attach attention!



Self attention

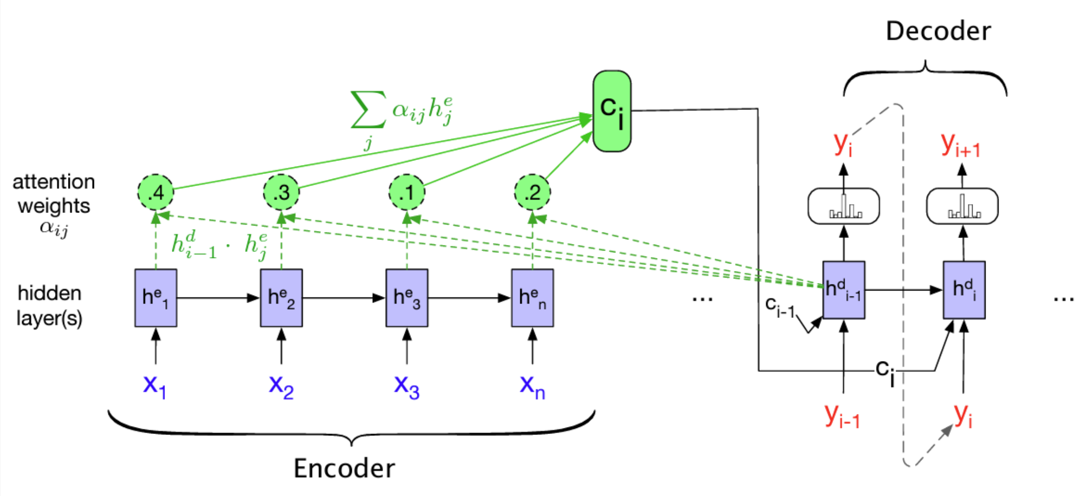
- Attention works better than RNN/LSTM for encoder-decoder models

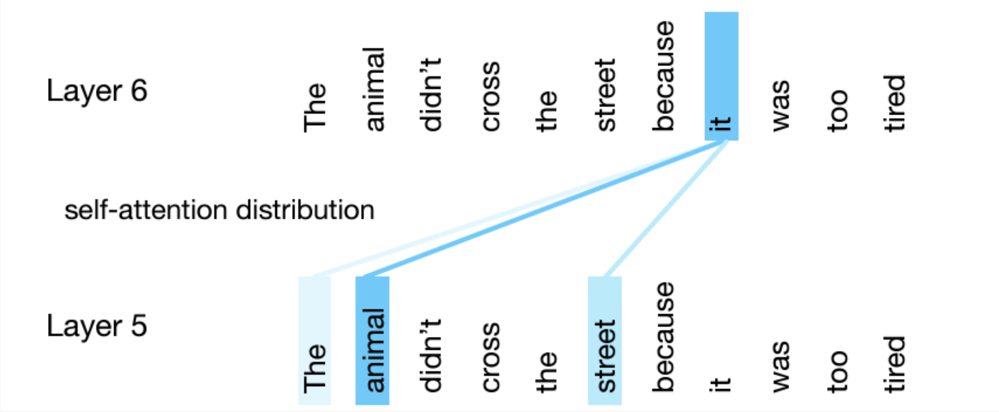
- Can we use attention for a standalone network?



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





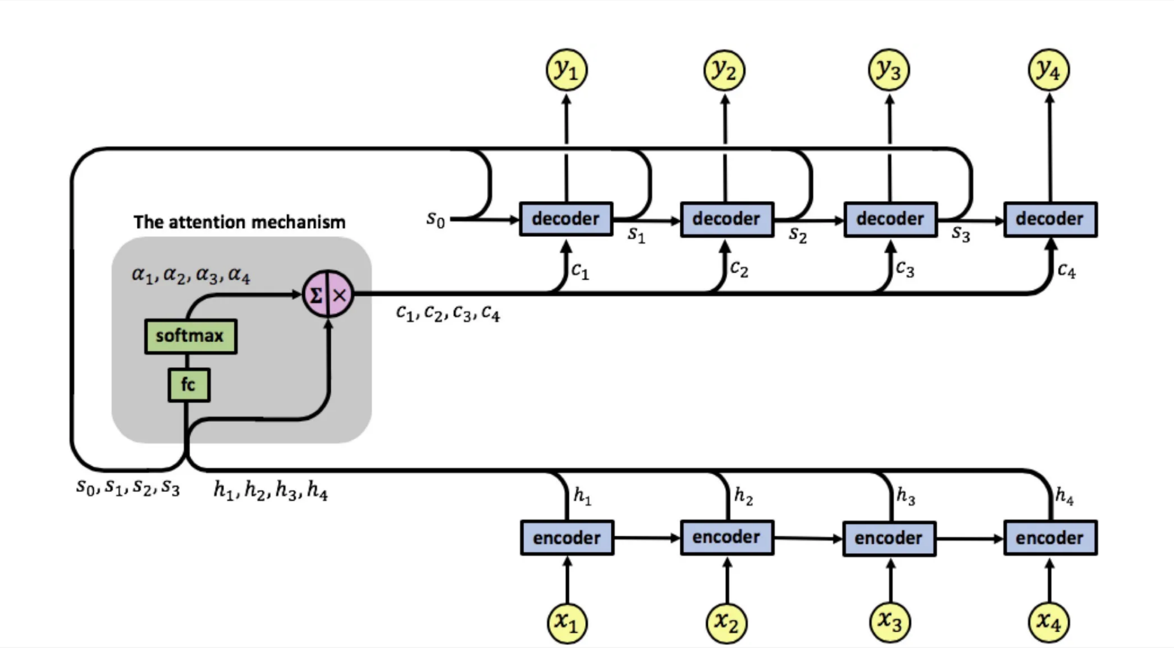
Causal attention vs bidirectional attention

= Masked attention

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

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

casual attention: not using the weight from the future



bidirectional attention: all the weights.

Causal self attention

following the real case. (when generating data)

- 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

representation

computational problem, if you use RNN, you can't calculate representation unless you calculate the previous state.

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

Pop quiz

- Can a transformer model process infinite input?

No, needs to be fix length

- Can an RNN be (natively) parallelized?

No, sequential

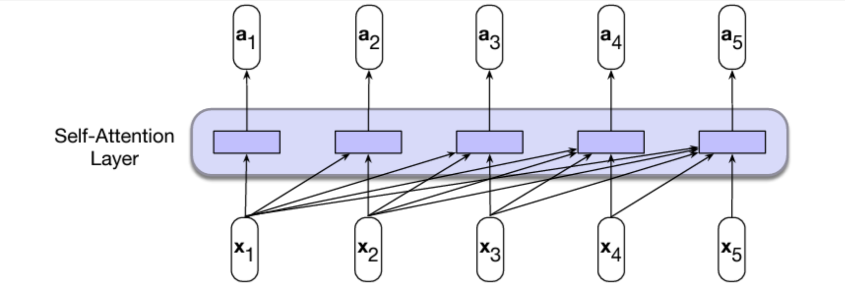
Causal self attention (intuition)

- 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?

x_3 itself

Decomposing input vectors

- We can use simple attention
- Transformers introduce query, key, value
- What are they, why do we need them and how do we use them?
  - The “dictionary” analogy
  - A semantic explanation, grounded in NLP

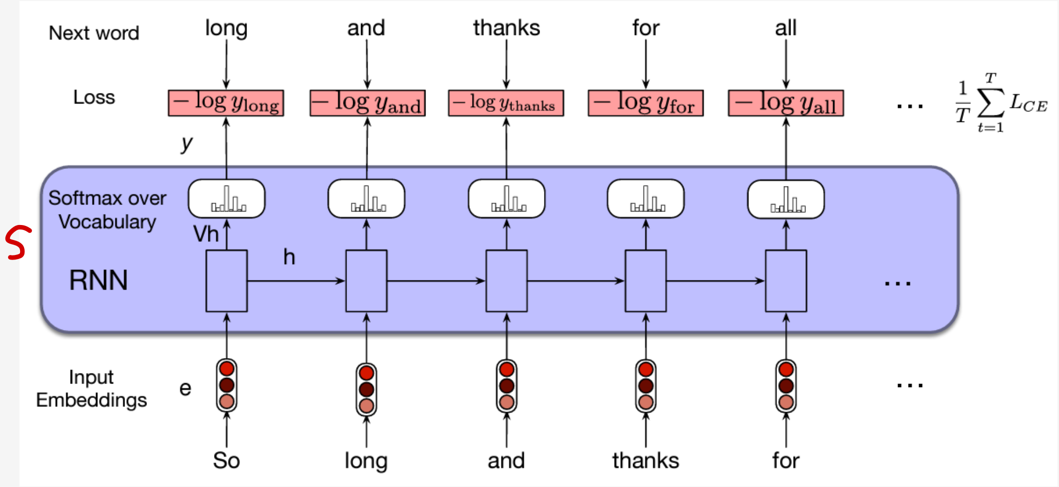
The LSTM and RNN

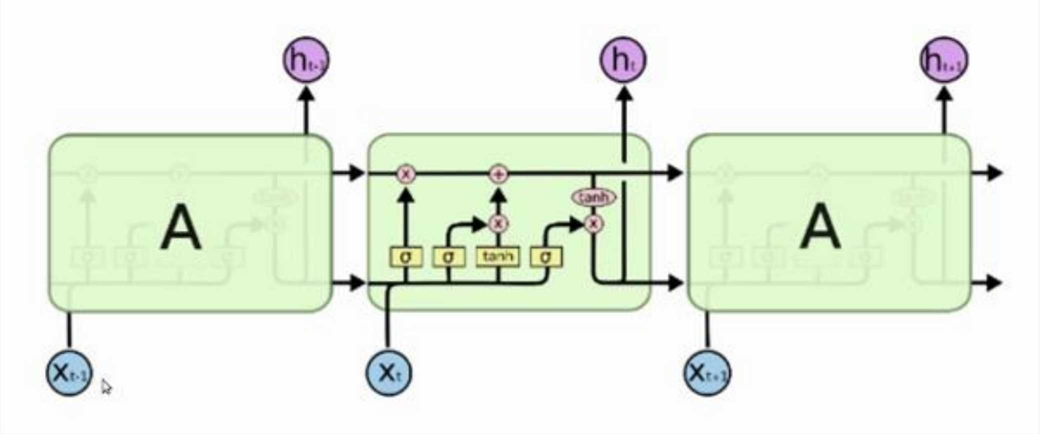
- What is the difference between LSTM and RNN?

long-term memory to handle
long-term dependencies

- Why do we need that "evolution"?

- Break a single hidden state into two + gates
    - Filtering and specialization





Compositionality of meaning

- Consider the following phrases
  - “A black dog”
  - “A house for my dog”
- What is the meaning of the dog in each phrase?
- Where is the dog in the second picture?

Compositionality of meaning (2)

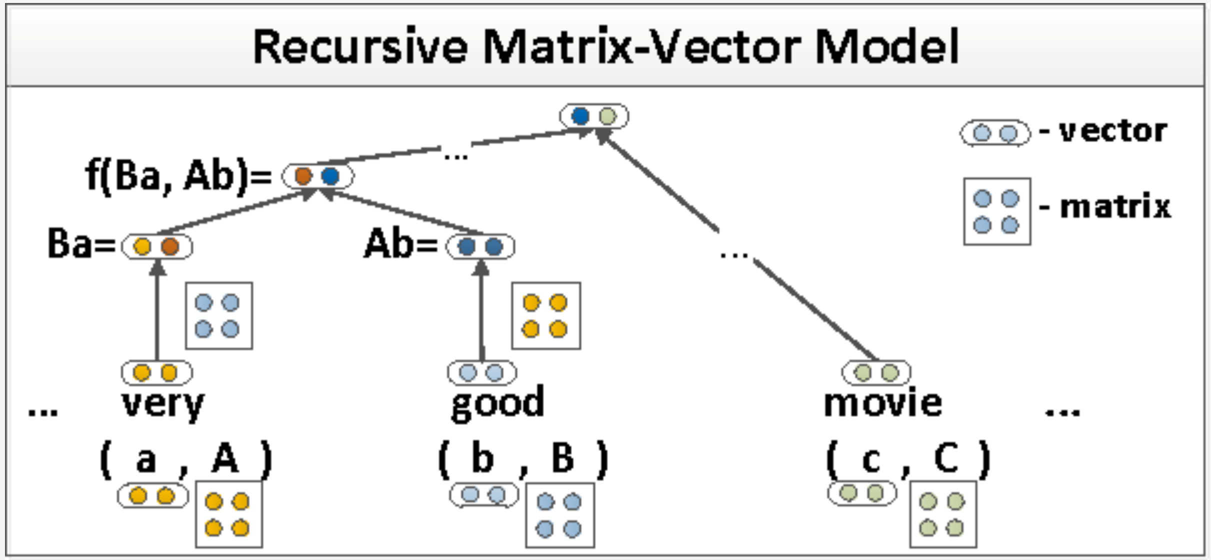
- What about “A house for my black dog”?
  - Does “dog” change the meaning of “house”? No
  - Does “house” change the meaning of “dog”? No
  - Does “dog” change the meaning of “black”? No
  - Does “black change the meaning of “dog”? Yes

- Meaning compositionality can be asymmetrical!

非对称.

Different aspects of meaning compositionality

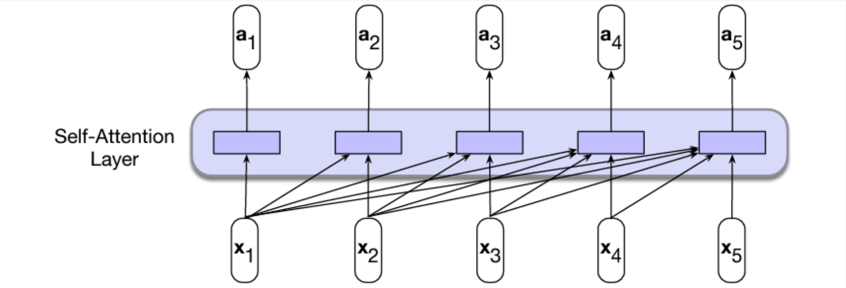
- Meaning compositionality is not a simple addition
- Words “behave” differently in different context
- Socher’s Vector-Matrix representation
  - Vector for the head, matrix for the complement
- Pop quiz: what would be the vector and what would be the matrix in “black dog”?



How to model asymmetric compositionality in attention?

- Self attention (that we have seen) has 1:1 correspondence

- Dot product attention is commutative
  - $a \cdot b = b \cdot a$
  - score(“black”, ”dog”) = score(“dog”, “black”)



- Pop quiz: would “black” have the same importance on “dog” as “dog” would have on “black”?

The query, key, value

- We project the input vector x to three vectors that serve different purpose: “query”, “key”, and “value”
- Two vector operations in the original attention:
  - “Score”: for indexes i and j, calculate how important is x_j for x_i: score(x_i, x_j) → query
  - “Scale”: for index i, calculate the hidden state h_i as a weighted sum of x_1 … x_i: $h_i = \sum_{j \le i} \alpha_{ij} x_j$ → key
- Each input vector x can three different roles
  - Argument 1 in score() [“dog” in score(“dog”, ”black”)] -> **query**
  - Argument 2 in score() [“dog” in score(“black”, “dog”)] -> **key**
  - The **value** used in scale to calculate the hidden state

Query, Key, Value (formally)

- We learn three different matrices (W^Q, W^K, W^V)
- Every input vector x_i 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: $$\mathbf{a}_i \; = \; \sum_{j \le i} \alpha_{ij} \mathbf{v}_j$$
- Pop quiz: which token will have the most impact on x_3?

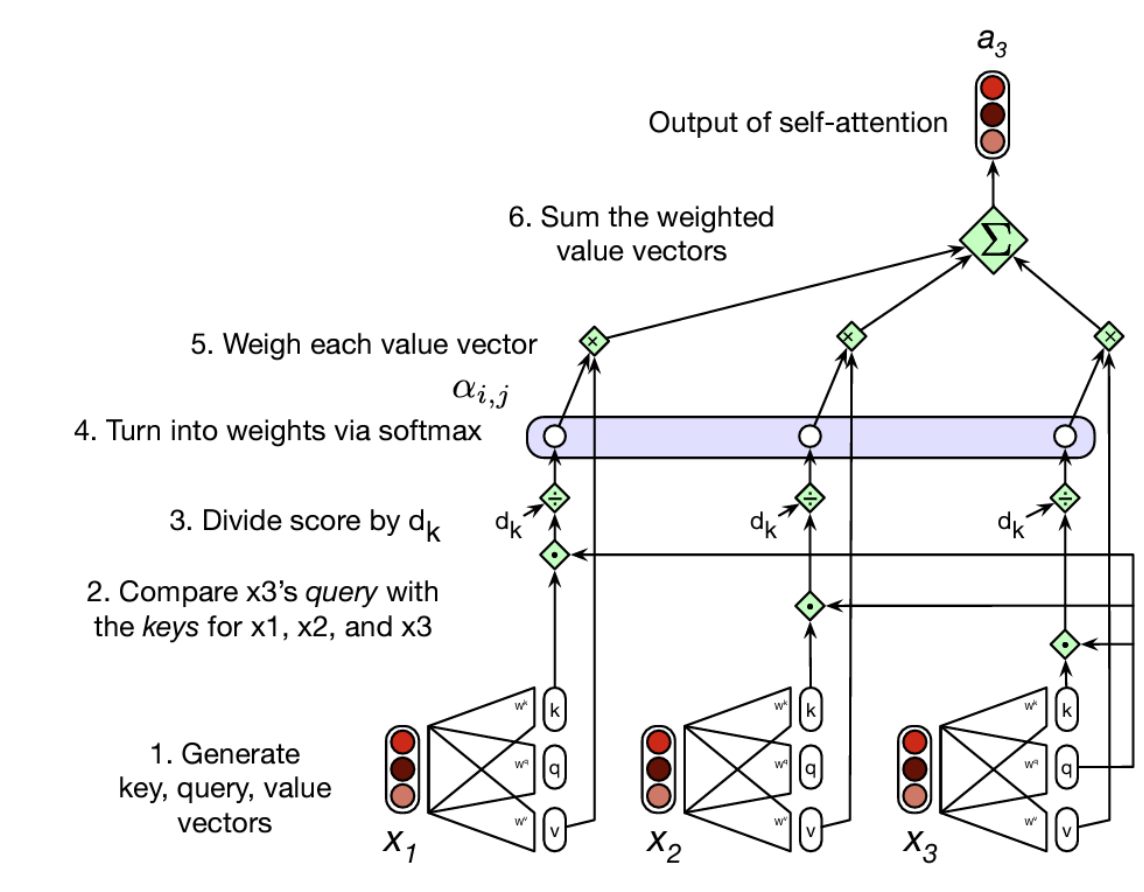
The transformer self attention

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

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

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



Parallelizing and masking the future

- Calculating hidden state h_t is independent of $h_{(t-1)}$
- We can compute all hidden states in a single operation
  - $Q = XW^Q$; $K = XW^K$; $V = XW^V$
  - $A = \text{SelfAttention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
- Can you see a problem for causal self attention?

N

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

↳ multiplying all the query key for the future, which we don't know

- Pop quiz: What is the complexity of the self-attention w.r.t. length of the input?

$O(n^2)$

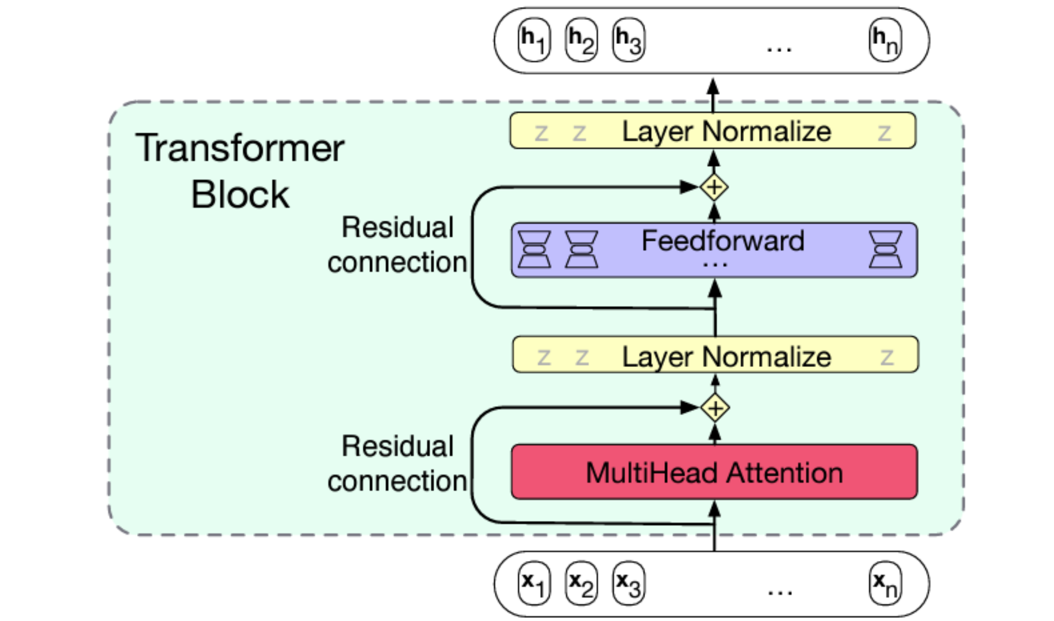
Multiheaded self-attention

- Instead of using a single self attention, we can use multiple
  - Each "head" has its own weights W^Q, W^K, W^V
  - The outputs of all heads are concatenated and projected to input dimensions
- Formally:

$$\begin{aligned}
\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q \,;\, \mathbf{K} = \mathbf{X}\mathbf{W}_i^K \,;\, \mathbf{V} = \mathbf{X}\mathbf{W}_i^V \\
\mathbf{head}_i = \mathrm{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\
\mathbf{A} = \mathrm{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O
\end{aligned}$$

The transformer block

- Residual connection
  - Copy the input of a layer to its output
- Layer normalize
  - Rescale each x vector to 0-mean with STD=1
- Feedforward
  - Apply the same fully connected FFN to each x



The transformer block (formally)

- Simplified representation
  - O = LayerNorm(X + MultiHeadAttention(X))

Normalisation

  - H = LayerNorm(O + FFN(O))

- You can change the order of operations in some implementaitons

Does transformer consider word order?

- Consider an autoregressive transformer

- Does it handle long-distance dependencies?

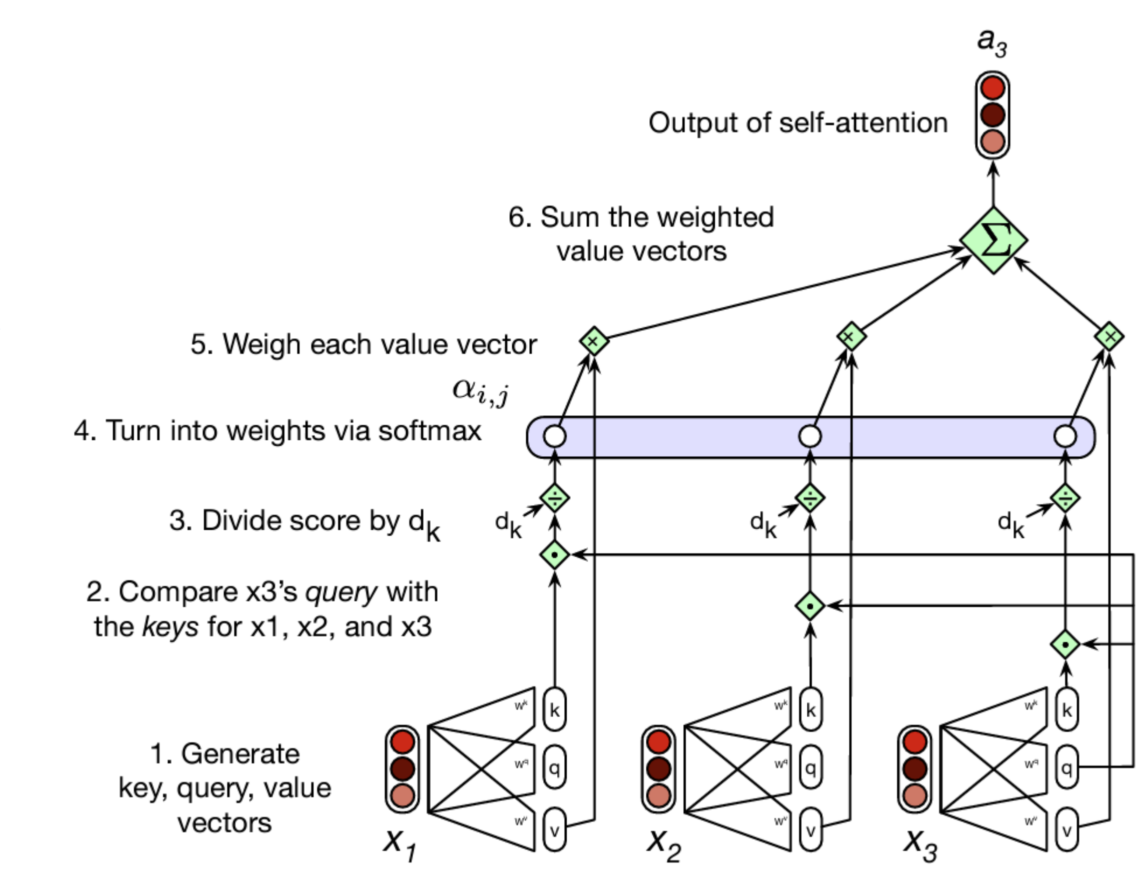
Yes, determine the way regardless of the word.

- Does it handle word order?

No

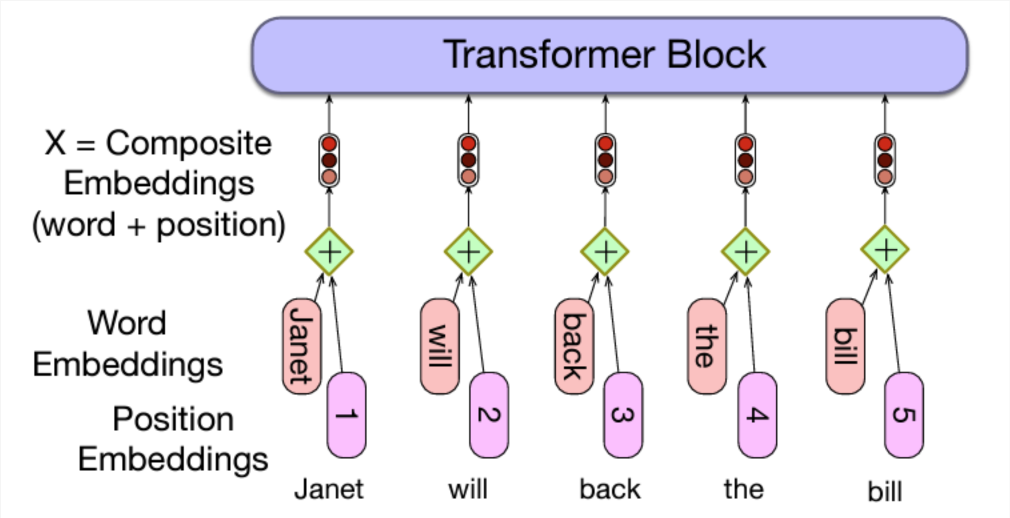
  - Does the position of x1 and x2 matter?

No, very similar with FFN. : To fix this, positional encoding



Encoding the Input. Positional Embeddings.

- Semantic embeddings
  - One-hot encoding maps to a row in a matrix
- Positional embeddings
  - One embedding for each position
  - Learnable; Same dimension as semantic
- Add semantic and positional embeddings



- Alternative techniques: use functions (sine/cosine); calculate relative positional embeddings

Classification layer: The “Head” of the model

- How did we train word2vec?

negative sampling, prepare a fake task

  - How did we reduce the computational cost?

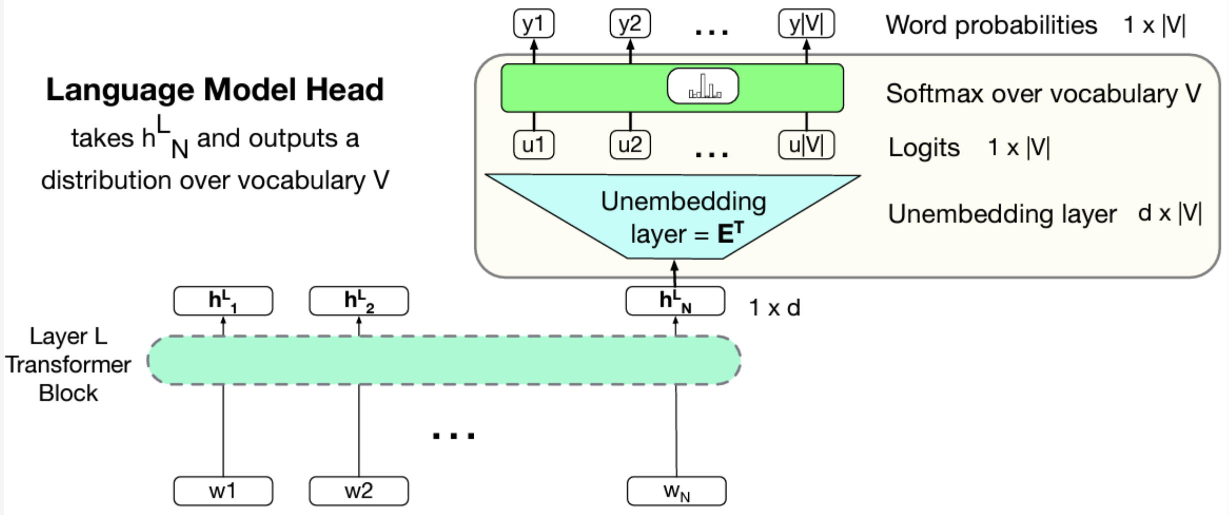
and it's the same.

- The concept of transfer learning
  - Train on one objective
  - Reuse the model on another task
  - We keep the stacked transformer blocks, change the “head”

Language modeling head

weight tying

- Language modeling
  - Efficient for learning representations
  - Self-supervised

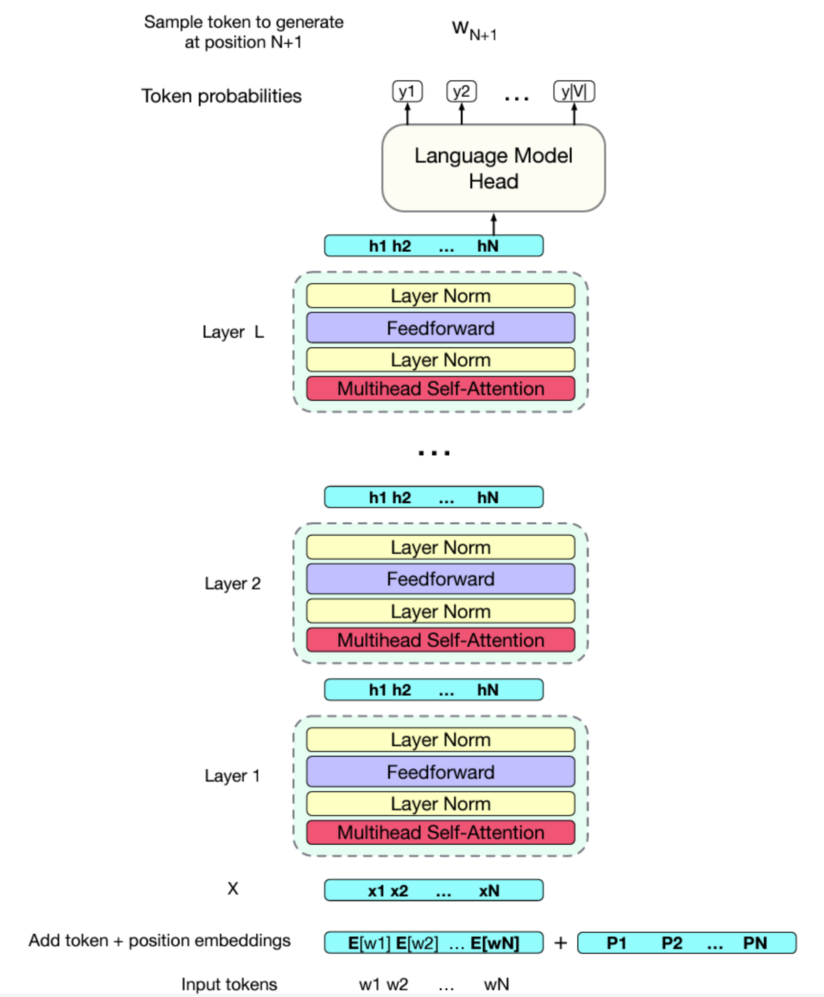


- Project h_N to vocabulary size
  - Do we know any computational tricks for that?
  - What would h^L_N look like?

↳ looking like the vector for the next word.

A final transformer representation for LM

- Token + positional embedding
- Multiple stacked transformer blocks
- A classification head
  - Language modeling with weight tying and sampling



Conclusions

Encoder decoder models

- Dealing with tasks where input and output are mismatched
  - Different length
  - No 1 to 1 alignment

- We use one model to encode the input (image, text in English)
- We use another model to generate text in target language

- Simple encoder decoder is based on RNNs/LSTMs

Attention

- RNNs have problems with long-distance dependencies
- Decoding from a single hidden state is restricted
- Attention uses all hidden states and compares current decoder state
    - Dot product attention

Transformers

- Self attention builds upon the attention from encoder-decoder
- Query, Key, Value project the input based on its function
- Multiheaded attention stacks multiple self attentions

Transfer learning

- The goal of transformers – learn contextual (and text) representations and reuse

- The head of the transformer determines the task

- Multiple problems can be framed as classification or generation