# Revision week
# Neural NLP – from embeddings to LLMs

Venelin Kovatchev

Lecturer in Computer Science

v.o.kovatchev@bham.ac.uk

# Outline

- Word embeddings
- Compositionality and end-to-end networks
- Encoder-decoder and attention
- Transformers
- Transfer learning and types of transformers
- LLMs and RLHF finetuning

# Vector representations and word embeddings

# The distributional hypothesis and word embeddings

- The distributional hypothesis in semantics
  - What does it state?
  - How does it affect NLP?
- Word embeddings
  - Encoding words as “semantic” vectors

# Count based vector representations

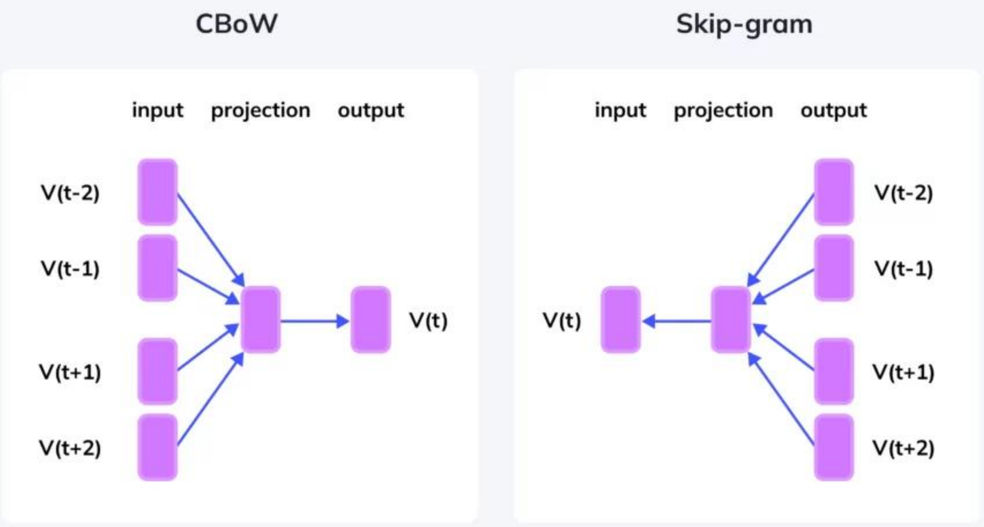
- Obtain a large corpus in the language/domain of interest
- Define a context of co-occurrence
  - What contexts can you think of?
- Count and fill in a co-occurrence matrix
- Apply transformations (which?)

# Word2Vec

- Learning embeddings directly from text

- A simple neural architecture

- Two algorithms
  - What is the difference between them?



# Negative Sampling

- Global objective: maximize the log probability of the dataset of size T with a context size c

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

- Calculate the probability of every word given context (or the other way around)

$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{W}\exp\left({v'_w}^T v_{w_I}\right)}$$

- Training with a softmax over the whole vocabulary is expensive

- Convert the task into "classifying the correct objective" using a logistic

$$P(+|w,c) = \sigma(\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

# Calculating similarity

- How do we calculate similarity between vectors?

- Dot product

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- Cosine

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

# Post-training word embeddings

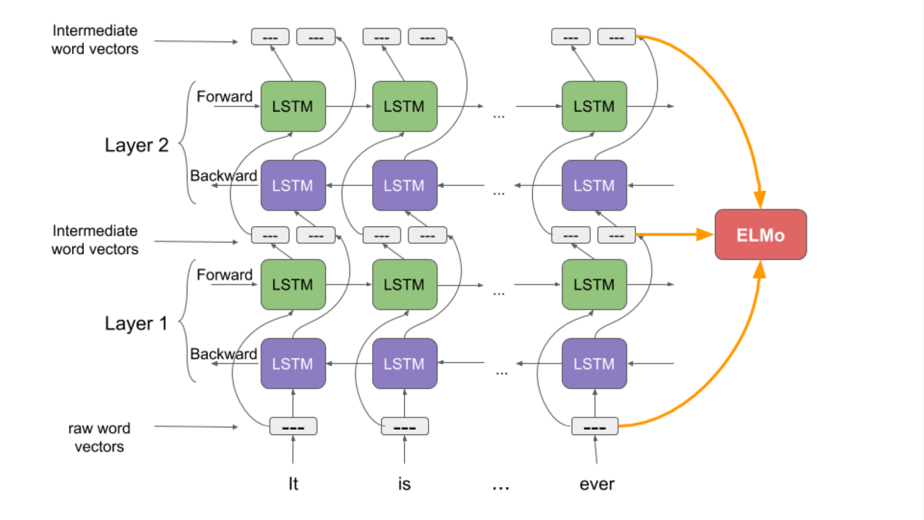
- We train embeddings using a logistic classification (with negative sampling)
- What happens with the logistic after the training?
- Can you think of another algorithm that uses similar approach?

# ELMO

- What is the most important difference between ELMO and word2vec?

- What makes ELMO representations "deep"?

- What is the training objective behind ELMO?



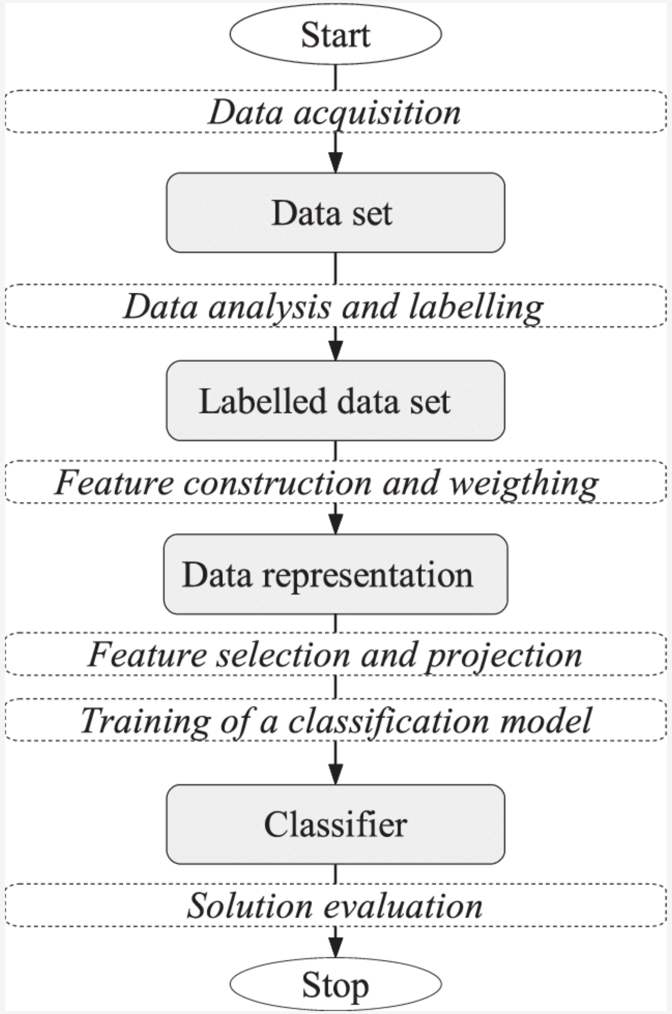
# Compositionality and End-to-end

# Feature engineering

- Analyze the problem, the input, and the desired outcome
- Explore existing resources and processing techniques
- Select the most relevant features and feature-extraction methods
- Empirically test what works best

# Text classification using features

- Step by step process
- Involves active human engagement
  - Feature selection and extraction
- Data is fed into a classifier (Logistic, NB, SVM)
- Iteratively improve feature selection and model (hyper) parameters



# Why not go end to end?

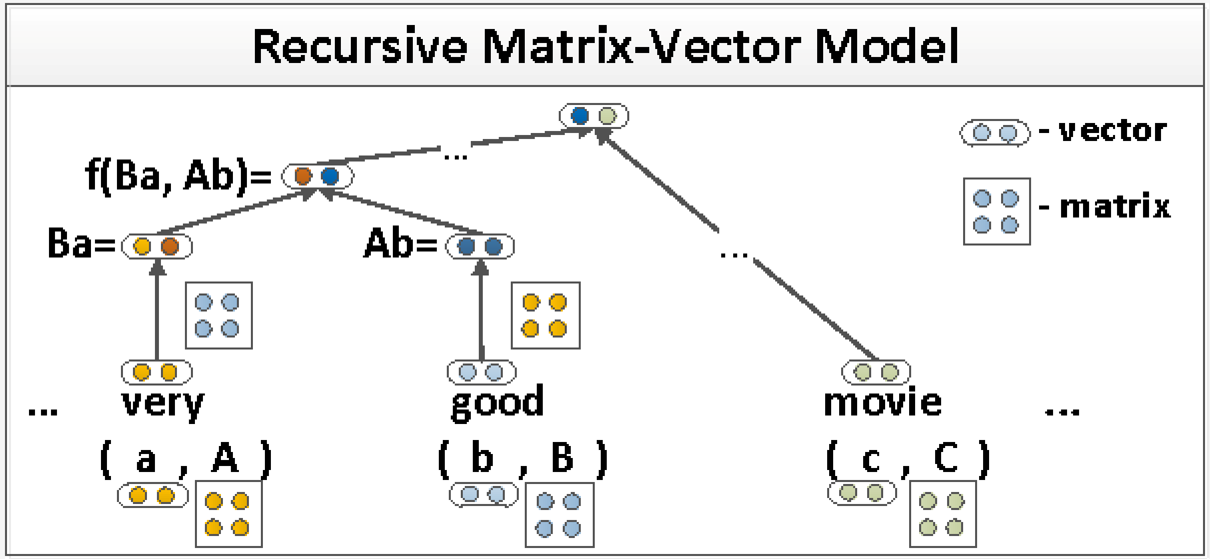
- Do we need full pipelines?
- Embeddings make it possible to “feed” text directly into models
- Is it possible to go fully end-to-end and eliminate
  - Accumulation of errors
  - Human labor and supervision
  - (In)compatibility issues between elements?

# Embeddings and the problem of compositionality

- Embeddings represent individual words
- NLP deals with processing texts
- How to go from word representations to text representations?

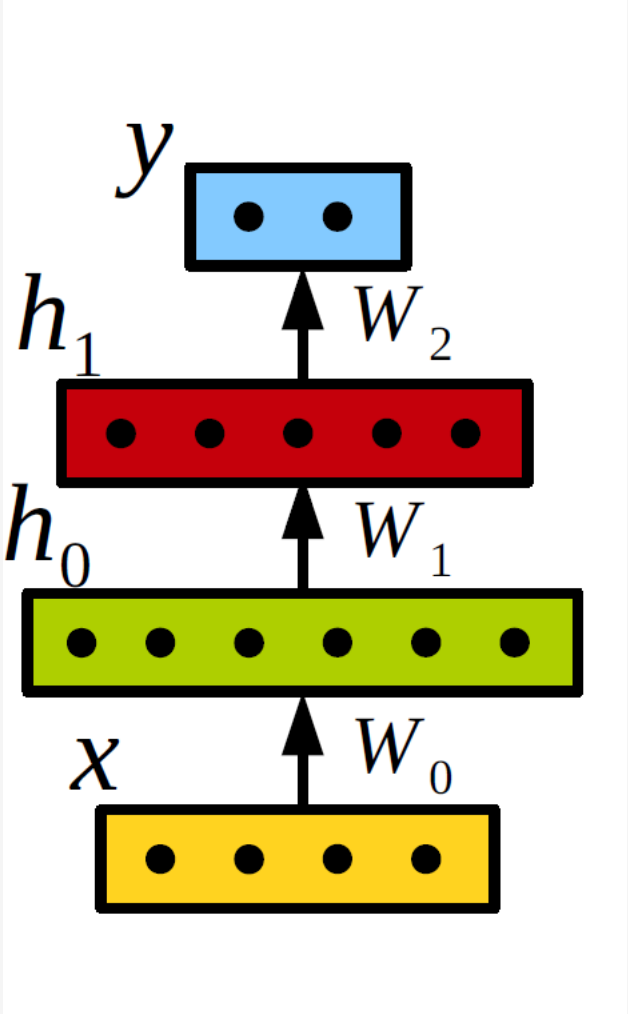
# Composing word meaning

- Vector operations
  - Vector addition
  - Pointwise vector multiplication
  - Vector concatenation
  - Complex matrix-vector operations
  - Which of these operations consider text structure?




- Let a neural network do the compositionality

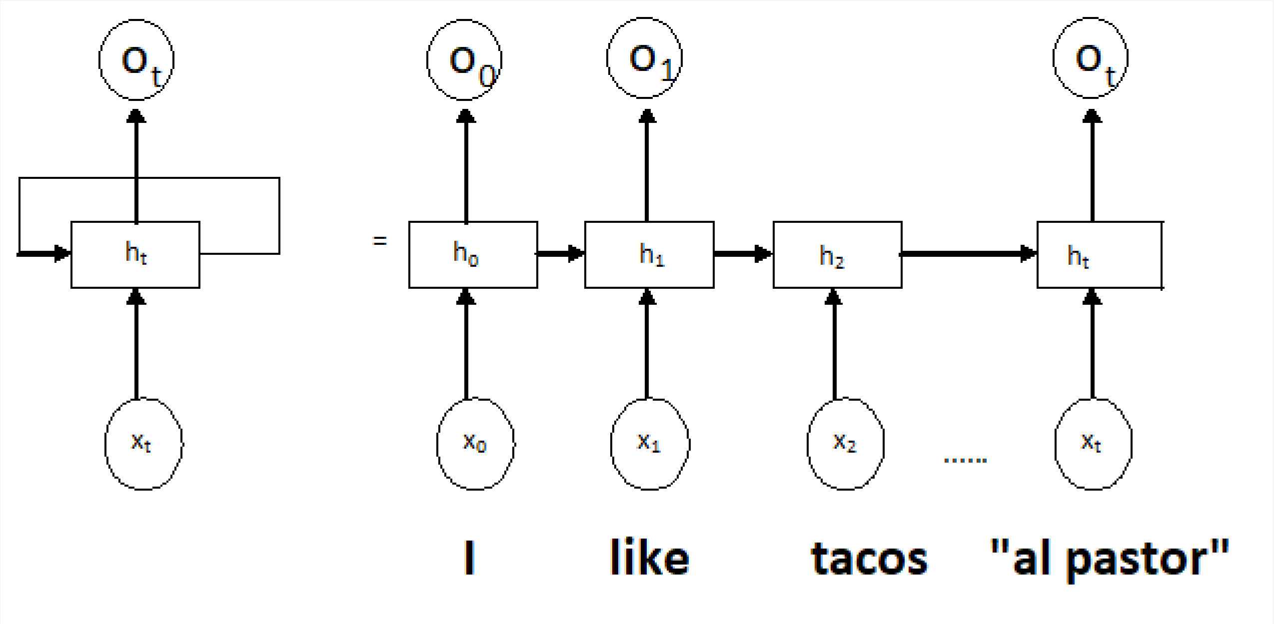
# Multi-layer perceptron




- $y = \text{softmax}(h_1 \cdot W_2 + b_2)$
- $h_1 = f(h_0 \cdot W_1 + b_1)$
- $h_0 = f(x \cdot W_0 + b_0)$
- Non-linear functions f:
  - Sigmoid: $\sigma(x) = \frac{1}{\exp(-x)}$
  - Hyperbolic: $\tanh(x) = \frac{1 - \exp(-2x)}{1+ \exp(-2x)}$
  - ReLU: $\text{rect}(x) = \max(0, x)$

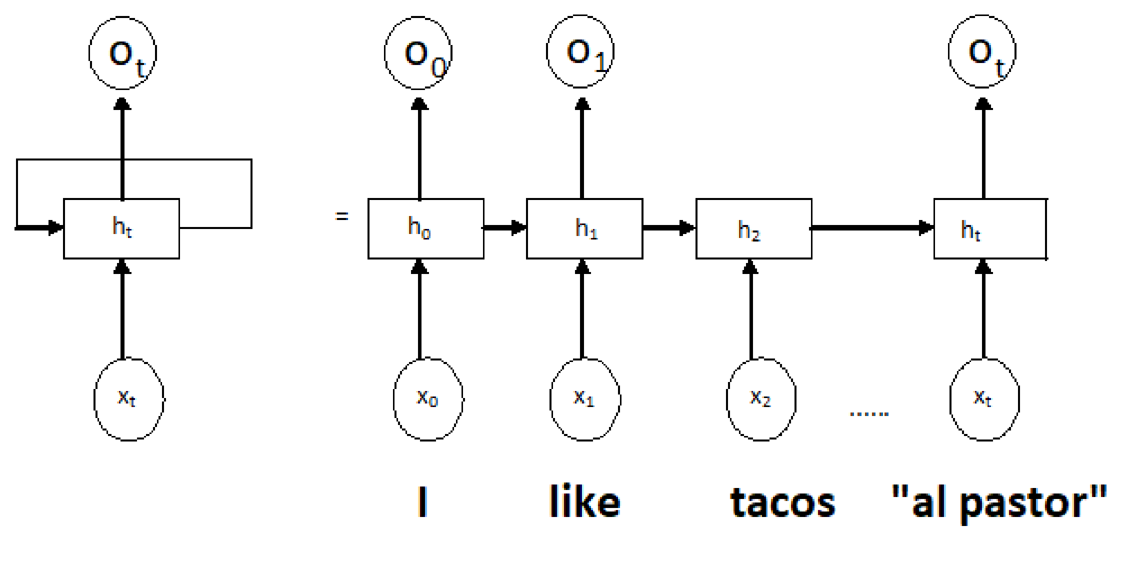
# Recurrent neural networks

- How to represent text of varying length?
- Copy the network for each word
- At each timestep t, input is $X_t$ and $h_{(t-1)}$
- I + like + tacos + "al pastor"
- Left to right, combining words one at a time
- Sequence classification
- Sequence to sequence



# RNNs (formally)

- At each timestep
  - Input $x_t$, and previous hidden state $h_{(t-1)}$
  - Three sets of weights:
    - input ($W_x$), hidden ($W_h$) , and output ($W_o$)
  - $h_t = f_h(W_x x_t + W_h h_{(t-1)})$
  - $O_t = f_o(W_o h_t)$
  - $J_t = f(O_t, y_t)$
- Pop-quiz: What are we predicting at O?



# Neural language modeling with RNNs

- More formally:
  - $h_t = \tanh(W_x x_t + W_h h_{(t-1)})$
  - $\hat{y}_t = \text{softmax}(W_y h_t)$
  - $J_t = -\log \hat{y}_{t,\text{ correct}}$ (- log prob the word at t+1)
- $J_{sent} = \frac{1}{T}\sum_{t=1}^{T} -log\ \hat{y}_{t,\text{ correct}}$
- Pop quiz: What is "weight tying" in RNNs?

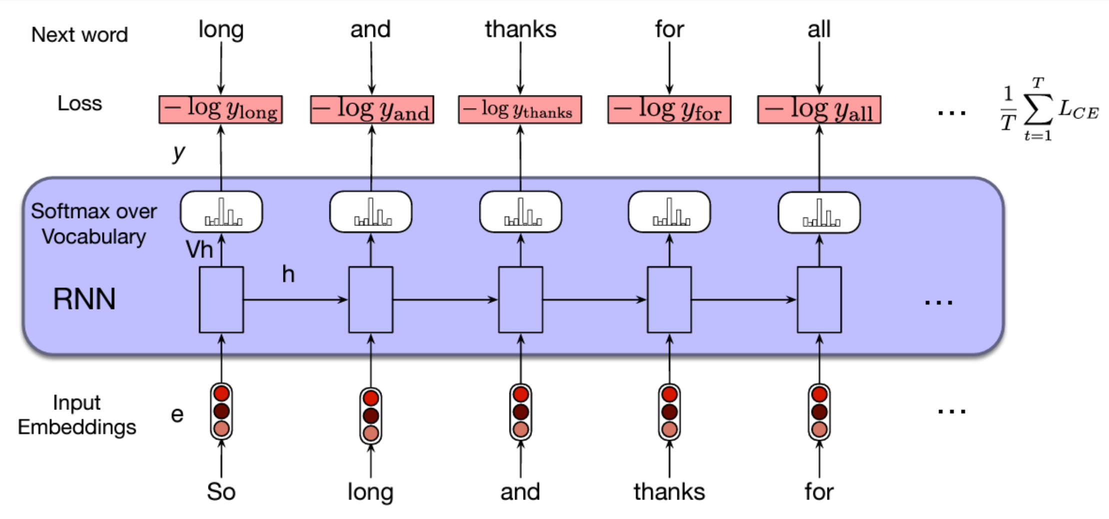



Figure 9.6 from SLP, chapter 9

# Stacked and bidirectional RNNs

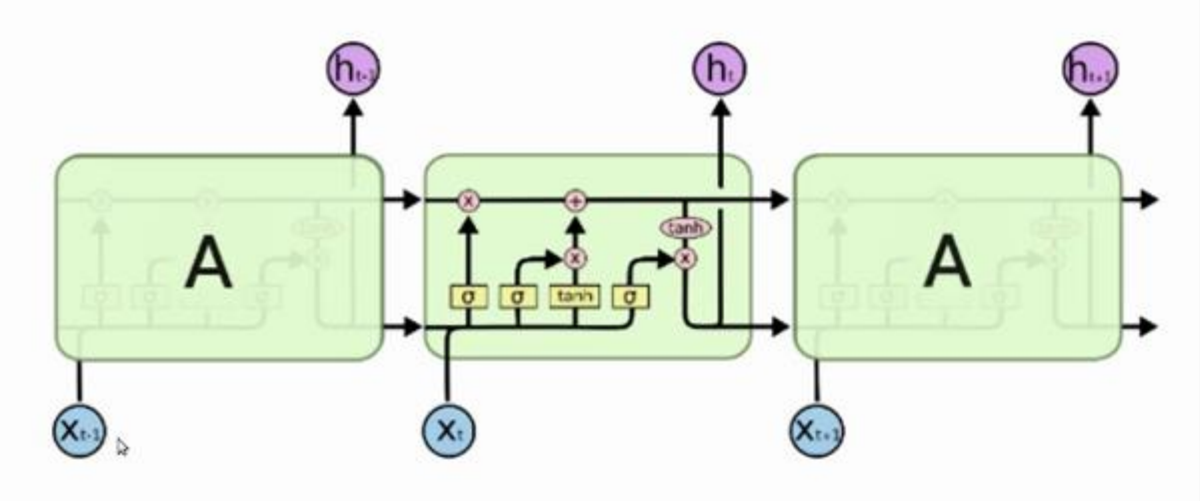
- The basic RNN is a powerful tool
- We can go deeper
- Stacking RNNs
- Bidirectional RNNs

# LSTM

- Popular variation of RNN that address native limitations
- Same recurrent concept (copy the network)
- Three different “gates”:
  - Forget gate
  - Input gate
  - Output gate
- Gates manipulate the flow of information through time
  - Addressing conflicting objectives

# Understanding the gates

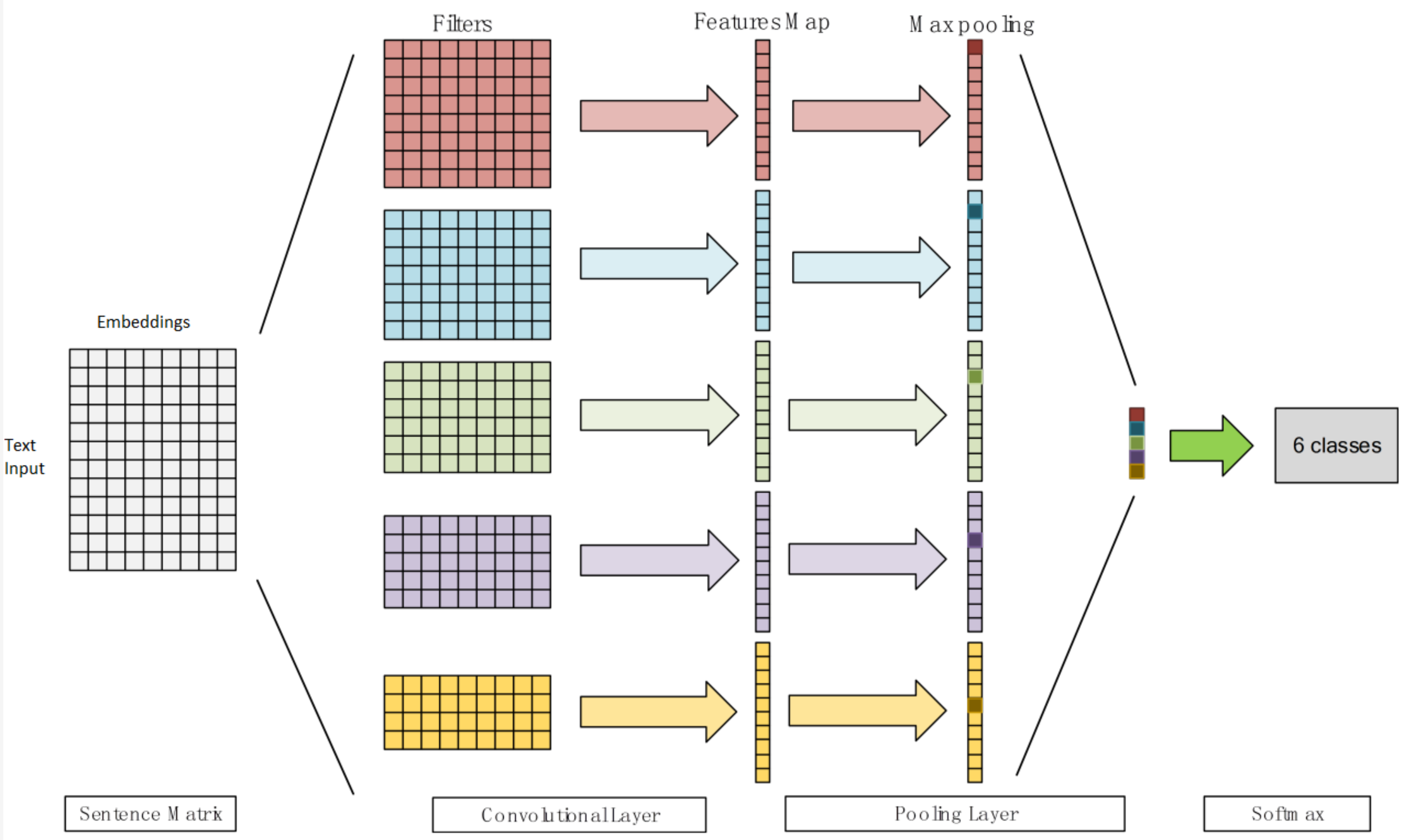
- All gates have the same format:
  - A feedforward layer followed by a sigmoid
- The gates are filtering out information at a certain part of the network
- Each gate has two important aspects:
  - How to calculate the filter
  - What is the input that the filter is applied to



# Convolutional neural networks (CNN)

- Another popular architecture for NLP
  - Deals with text of various size
- Inspired by computer vision success
- Applying filters of different size to the input (2,3,4) + pooling
  - Analogous to n-grams

# Convolutional neural networks (CNN)



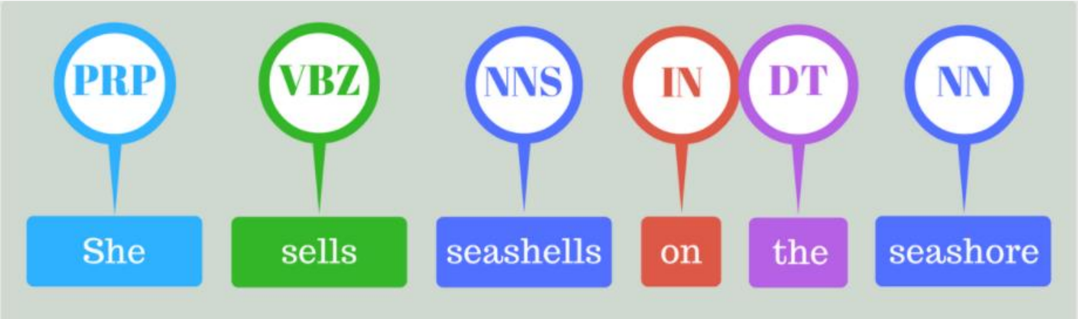
# Compositionality

- How each network “composes” meaning
  - Feed-forward network
    - Linear combination of words (no order) + activation function
  - RNN/LSTM
    - Recursively, word by word (linear order)
  - CNN
    - Locally, similar to n-gram (proximity window, limited order importance)

# Encoder-decoder and attention

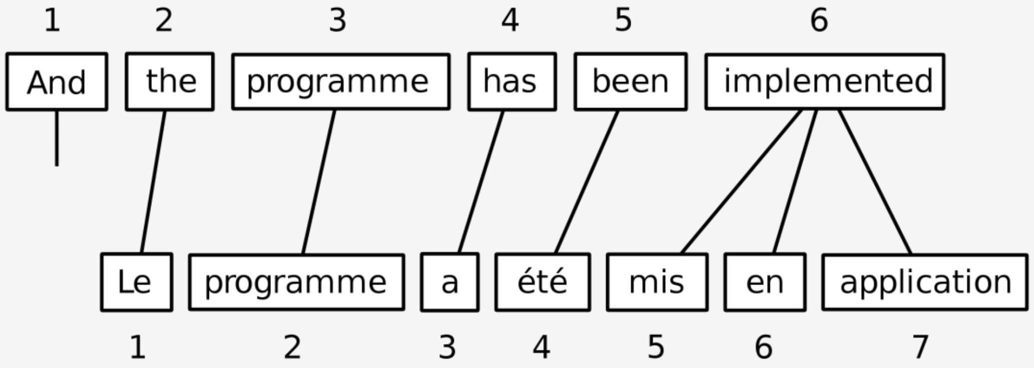
# Sequence labeling vs sequence-to-sequence problems

- 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

# 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?

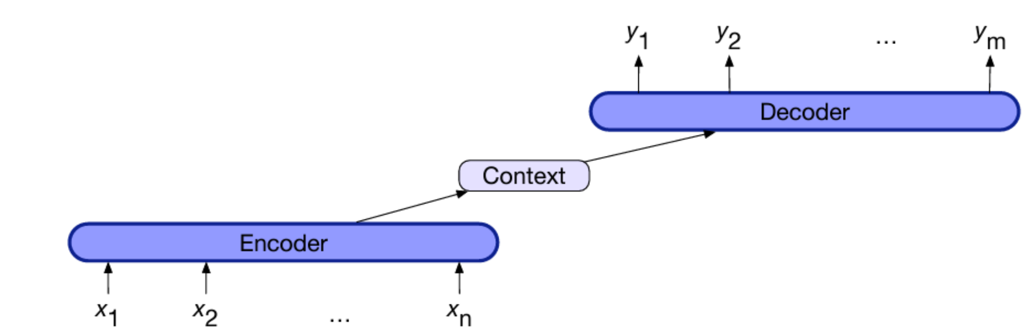


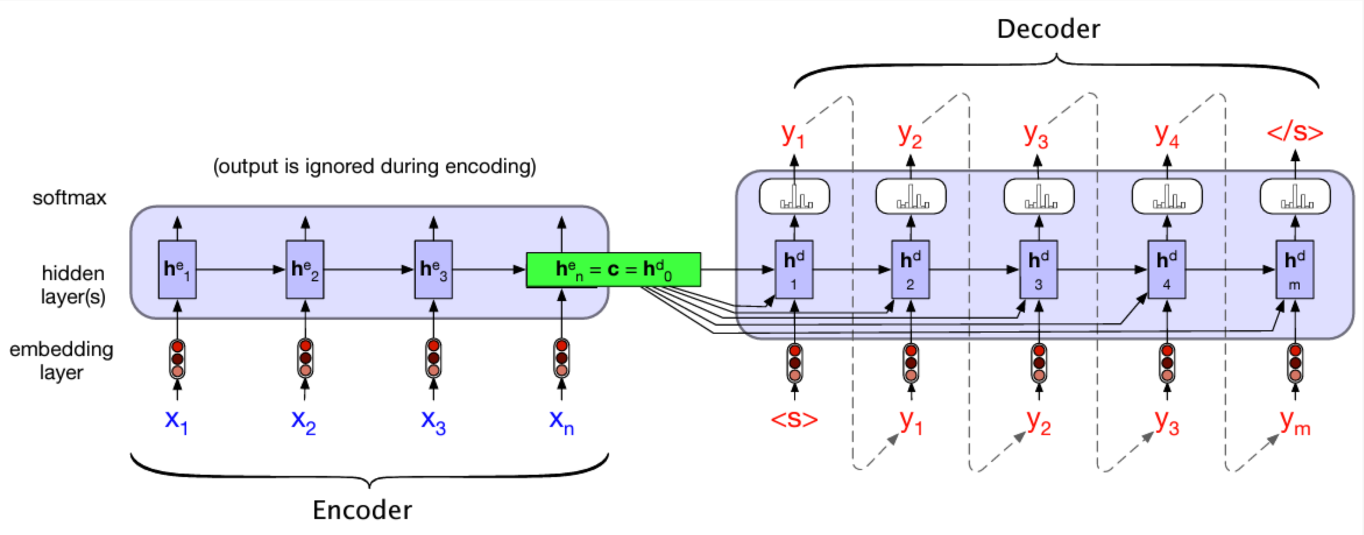

Fig 9.16

# Using separate RNNs for encoder and decoder

- Train two models
- Pass the context at every step

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

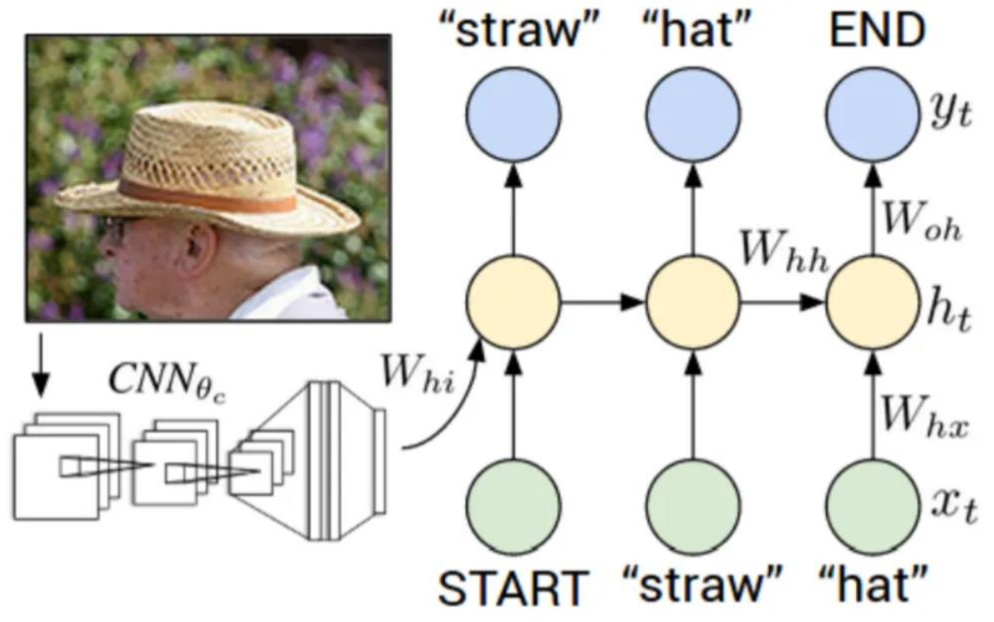
- Can you point a potential problem?
  - What could improve this architecture?




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

# Encoder decoder across modalities: image captioning

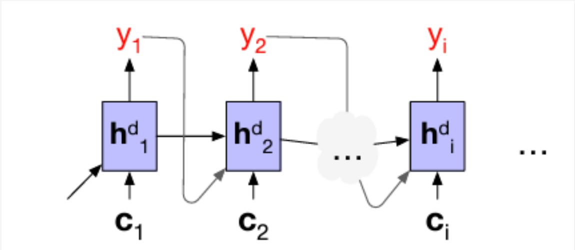
- The encoder and decoder “talk” via the context
- They don’t have to be the same type of model
- The modalities don’t have to match
    - Speech to text
    - Image to text




Deep Visual-Semantic Alignments for Generating Image Descriptions

# Attention – basic implementation

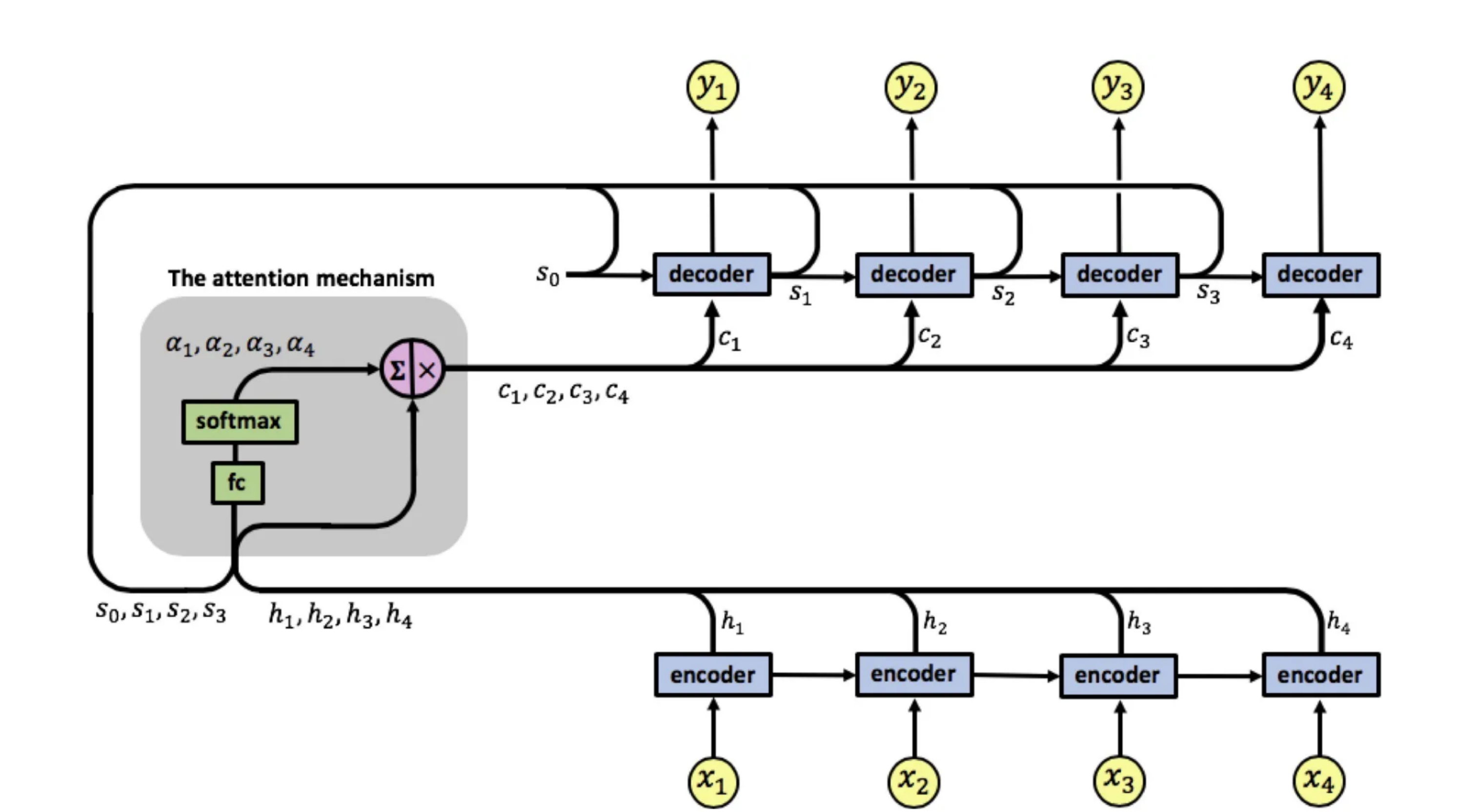
- Intuition: each token in the target should use a "personalized" context
  - Should have access to all hidden states in the encoder
  - The context should have a fixed (vector) length
- Weighted sum of all encoder hidden states
  - Calculated separately at each decoder step
  - Using the hidden state 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



# Visualization of RNN with attention

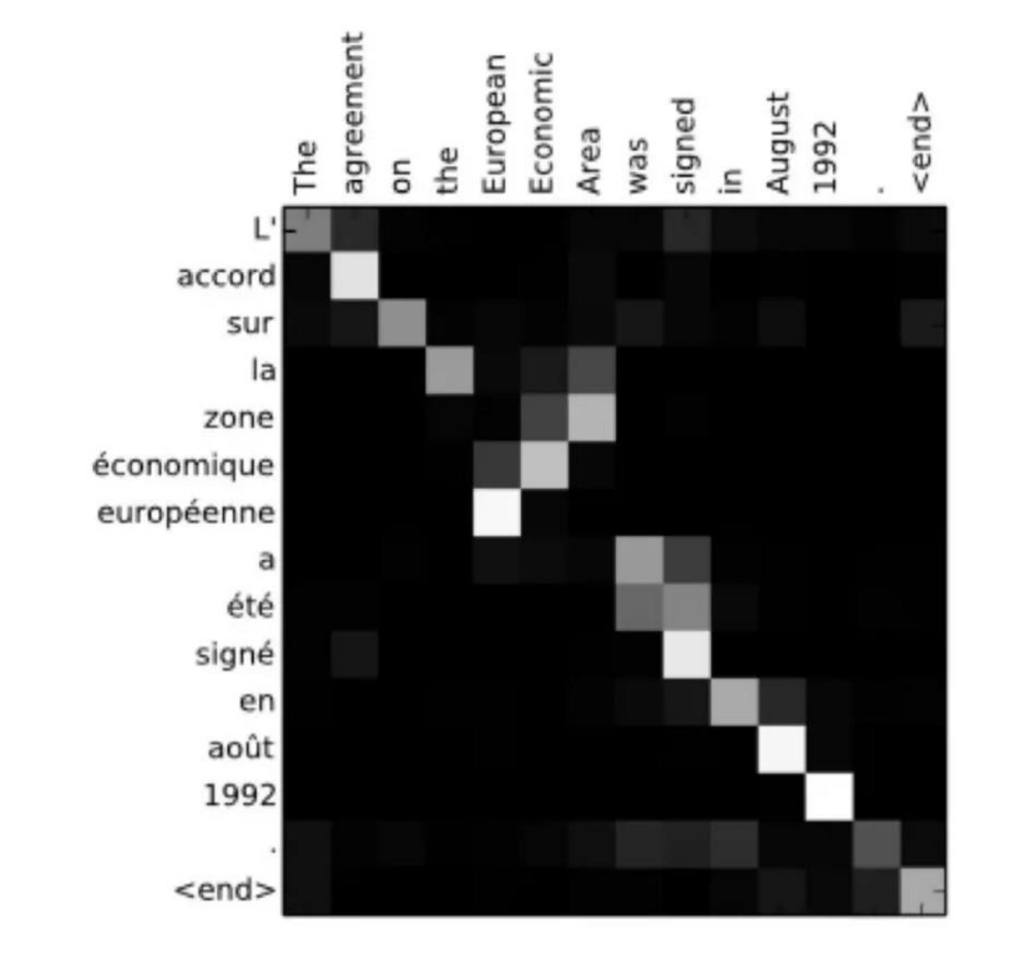
- RNN with attention

- Attention is learned via a simple FFN



# Visualizing attention

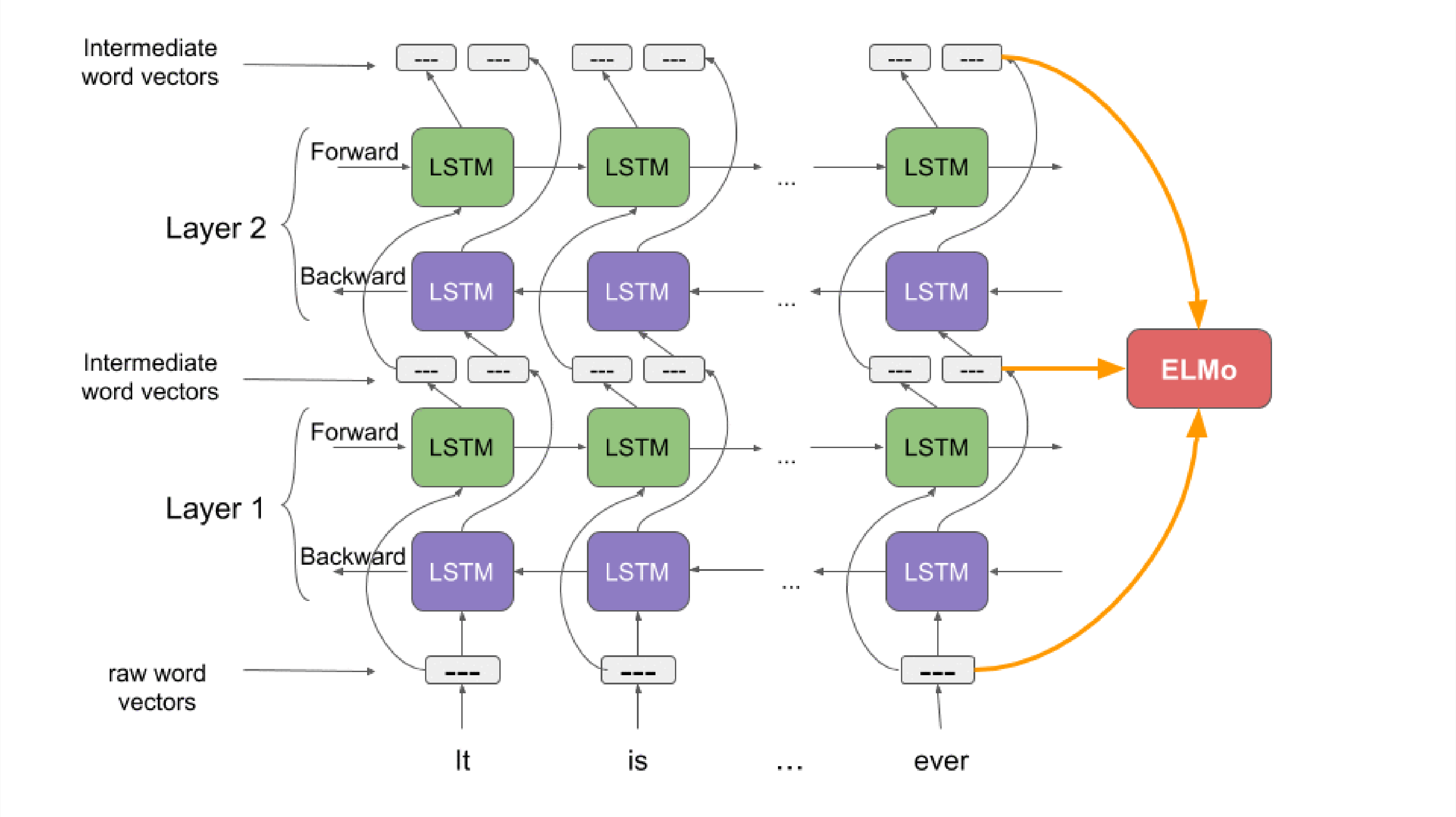
- Linear weights are interpretable
- We can see which word is more important
- Can we use attention for explainability?



# Transformers

# Elmo architecture

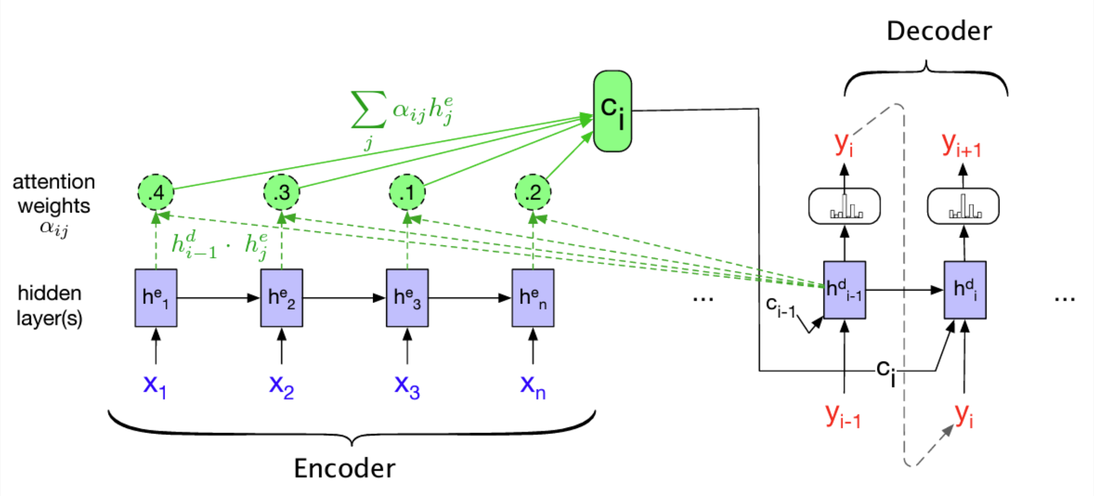
- How can we improve over that?



# Self attention

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

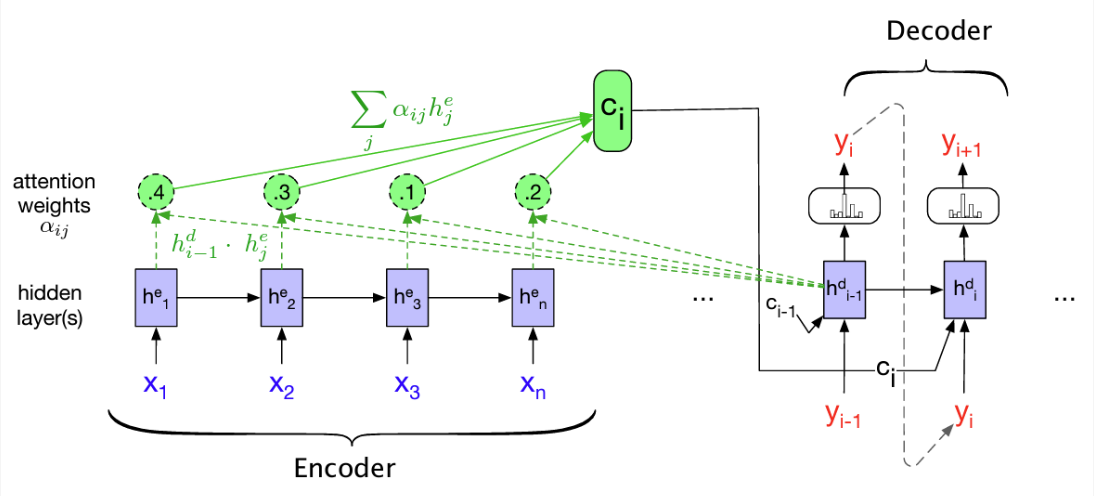
- Can we use attention for a standalone network?

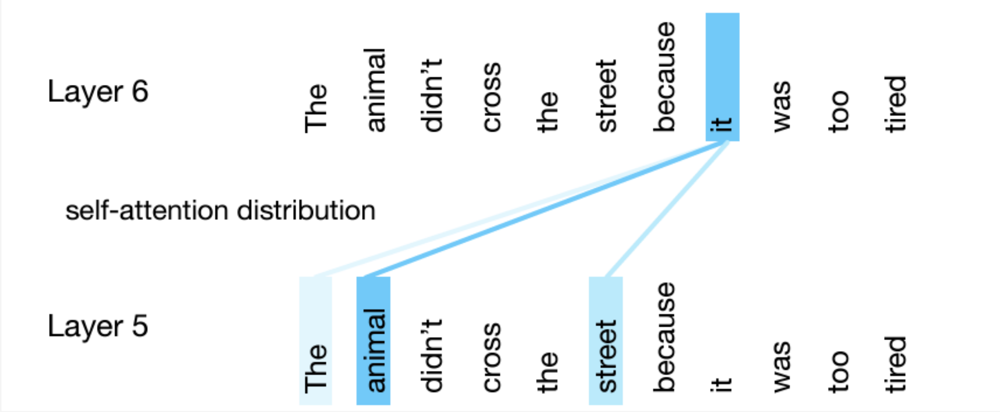


# Self attention (2)

- Self attention is a key concept in building transformers

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



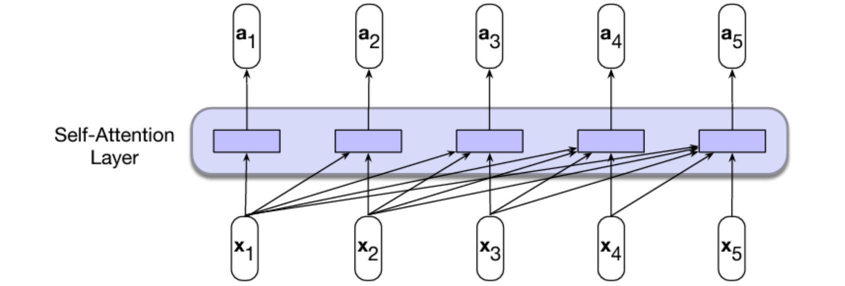
# Causal self attention (intuition)

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

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

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

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



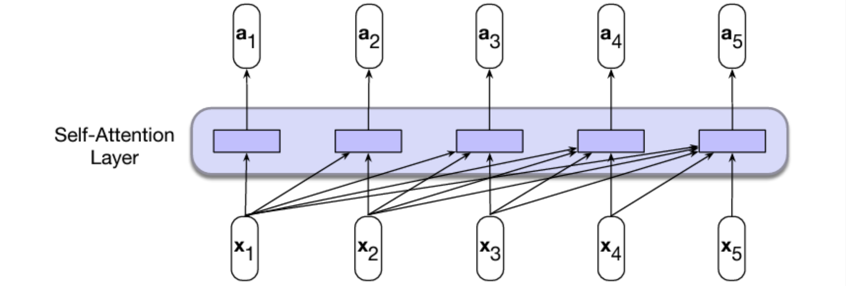

  - Which is the most similar token to $x_3$? What is the input to the first hidden layer?

# Decomposing input vectors

- We can use simple attention and it works
- 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

# How to model asymmetric compositionality in attention?

- Classical 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($x_i$, $x_j$)
  - “Scale”: for index i, calculate the hidden state $h_i$ as a weighted sum of $x_1$ … $x_i$: $h_i = \sum_{j \le i} \alpha_{ij} x_j$
- 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 $x_3$?

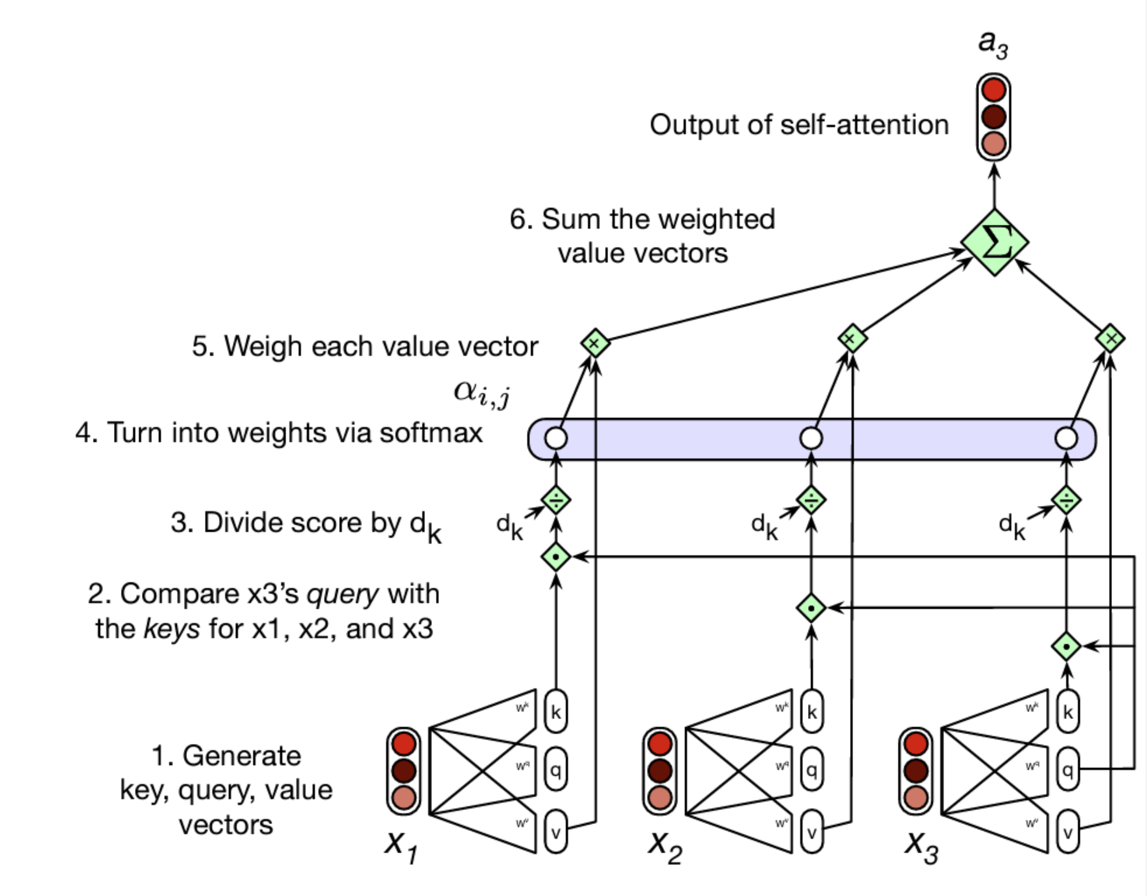
# The transformer self attention

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

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

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



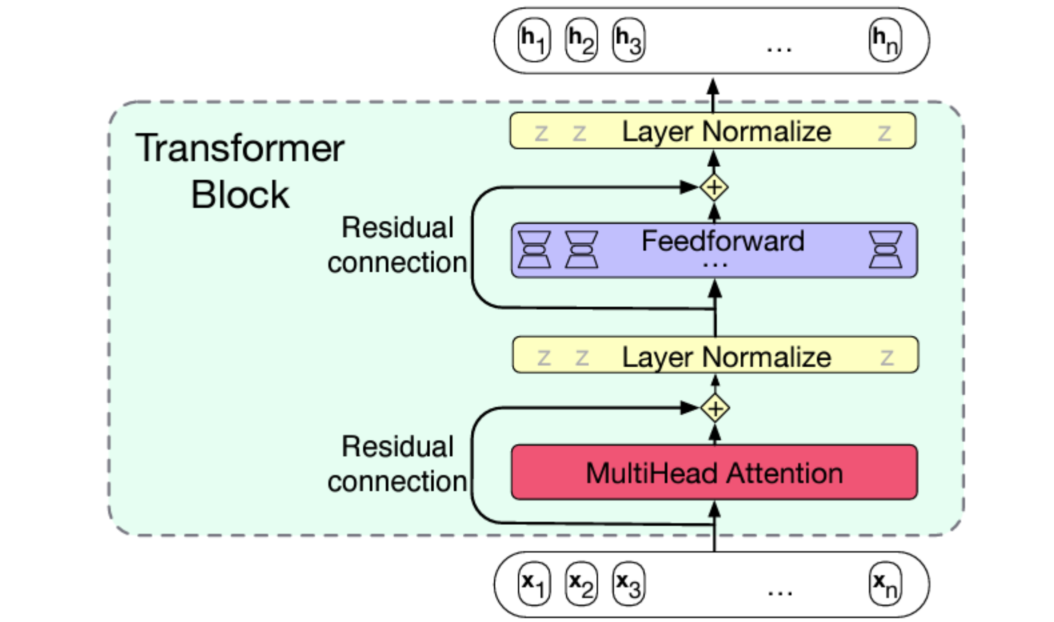
# Multiheaded self-attention

- 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
  - You can also think of multiheaded attention as "breaking" one big attention into specialized subsets
- 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
- Positional feedforward
  - Apply the same fully connected FFN to each x

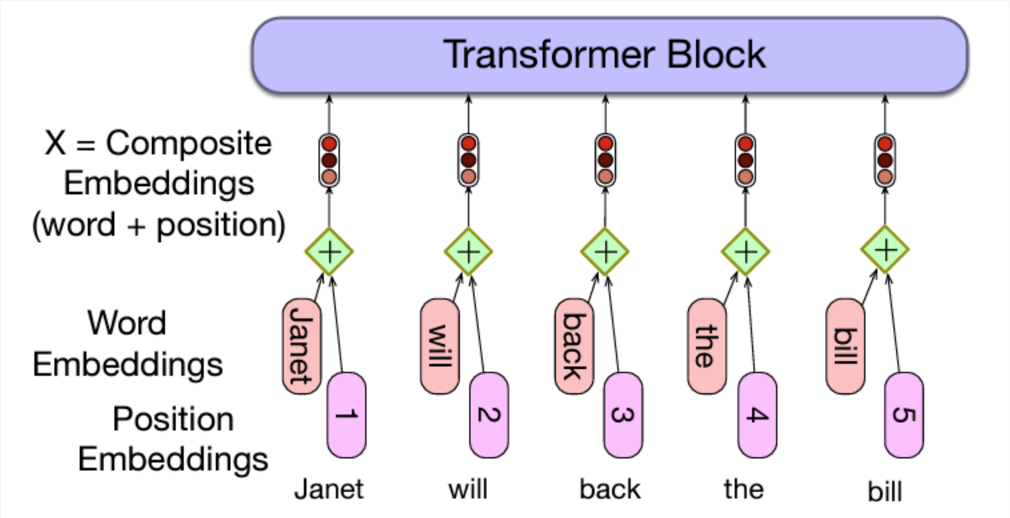


# The transformer block (formally)

- Simplified representation
  - O = LayerNorm(X + MultiHeadAttention(X) )
  - H = LayerNorm(O + FFN(O))
- You can change the order of operations in some implementations

# Encoding the Input. Positional Embeddings.

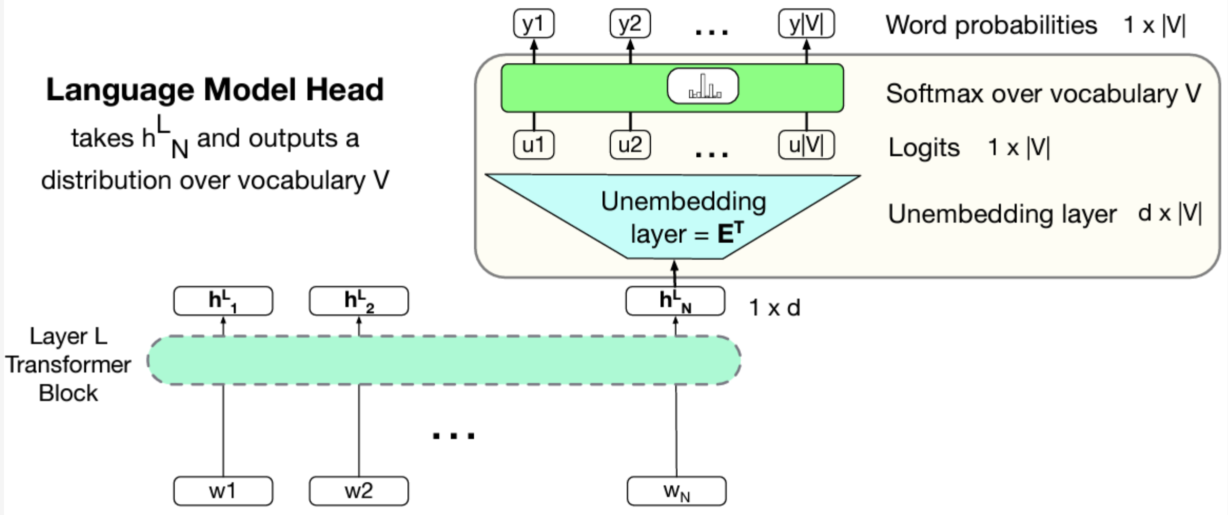
- Semantic embeddings
  - One-hot encoding maps to a row in a matrix
- Positional embeddings
  - One embedding for each position
  - Learnable; Same dimension as semantic
- Add semantic and positional embeddings




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

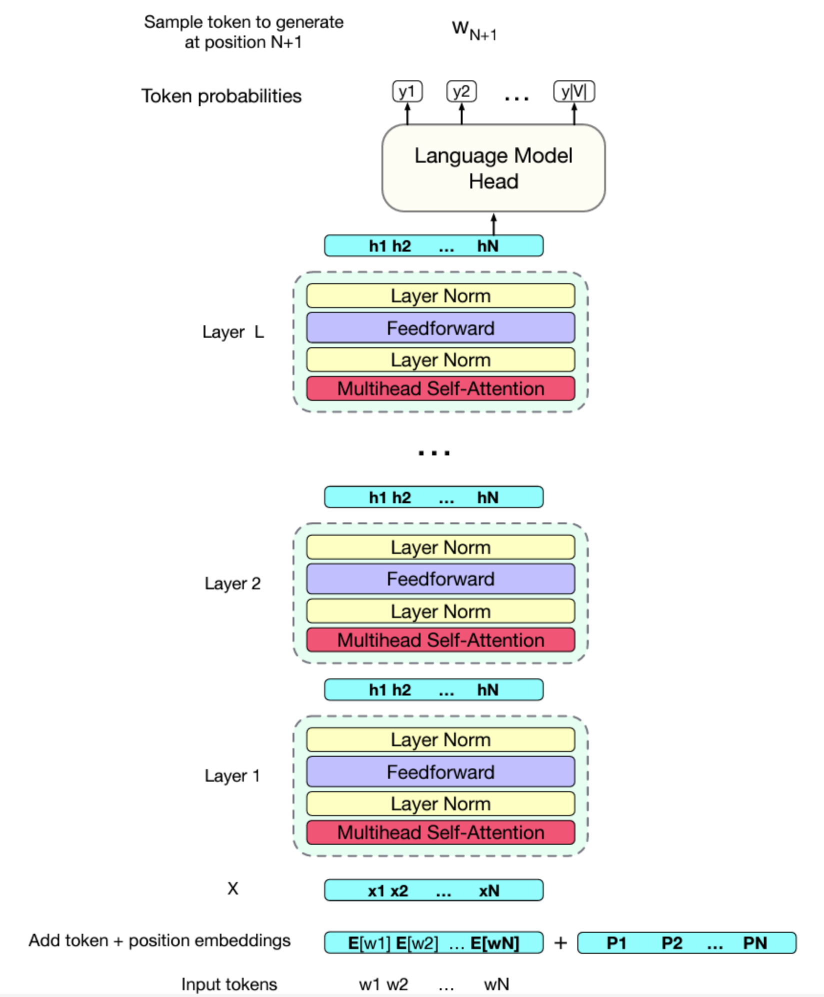
# Language modeling head

- 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?



# A final transformer representation for LM

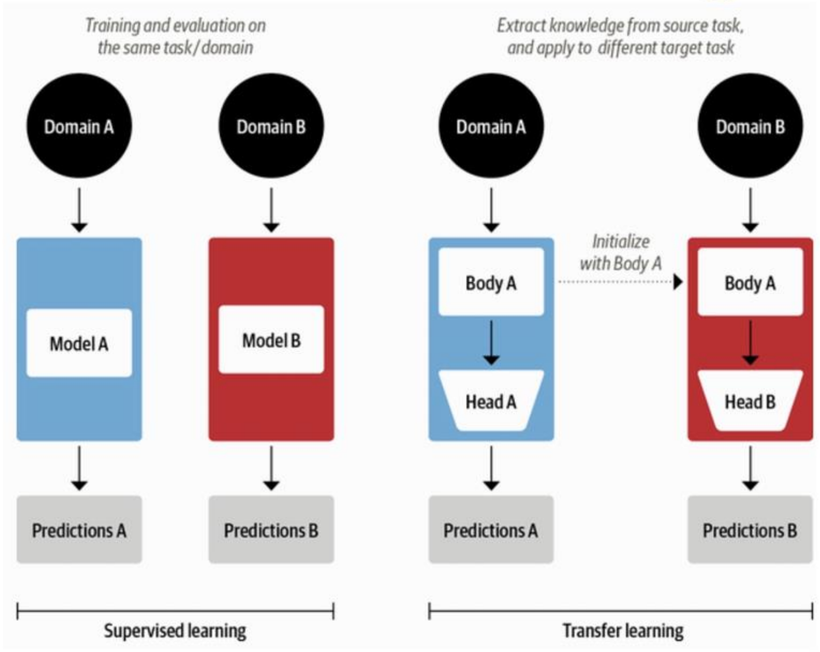
- Token + positional embedding
- Multiple stacked transformer blocks
- A classification head
  - Language modeling with weight tying and sampling



# Transfer Learning and Types of Transformers

# Supervised learning vs Transfer learning

- What are the goals and benefits of transfer learning?
- What are some potential issues or risks?
- Are there any other paradigms that you can think of in that area?




Source: Lewis Tunstall, Leandro von Werra, and Thomas Wolf (2022), Natural Language Processing with Transformers: Building Language Applications with Hugging Face, O'Reilly Media.

# The decoder transformer: GPT

- GPT1 combines different concepts we know so far
  - The standard transformer block
  - Neural Language Modeling
  - Transfer learning capabilities
- Pop quiz: What kind of attention does it use?
- Intuition:
  - Generative pre-training
  - Discriminative finetuning

# Finetuning GPT

- After pretraining, use the hidden state at last layer
- Add a last linear layer with m neurons (m = number of classes)
- Predict the target class:

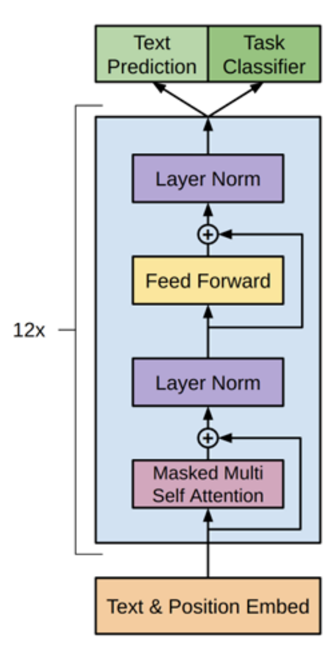
$$P(y|x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y).$$

- Maximize the probability of the correct labels (need labeled data)

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m).$$

- Combining both losses together

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

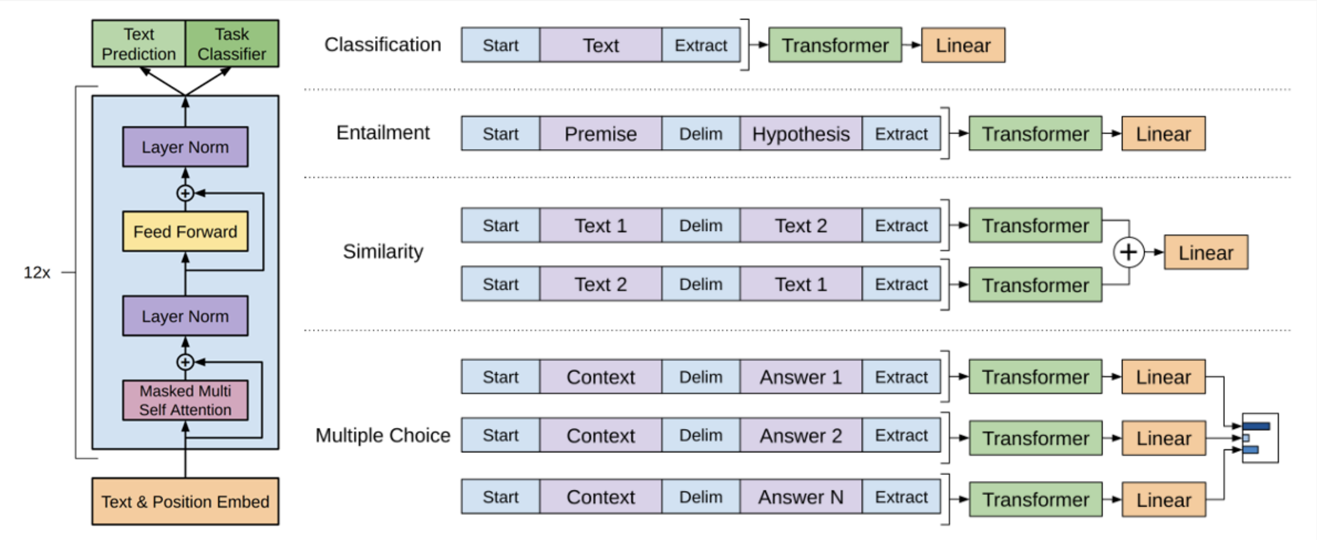


# Task specific input transformations

- Task reformulating

- Using special tokens (sep, start/end)

- Comparing separate "streams"



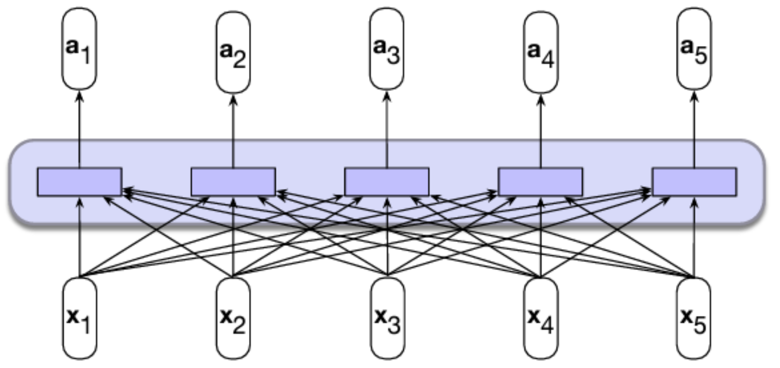

- Task design is a non-trivial task
  - Task formulation; Data format; Metrics and Evaluation

# The encoder transformer: BERT

- The original encoder-only transformer
- An English-only sub-word vocabulary consisting of 30,000 tokens
  - Most of the modern algorithms use subwords tokenizers and embeddings
- 768 hidden size
- 12 layers, 12 heads in each multi-head attention
- 100M parameters
- Trained on two tasks: Masked Language Modeling and Next Sentence Prediction

# The encoder transformer

- The encoder in “attention is all you need”
- Same architecture as the decoder
- Bi-directional self-attention
  - All key/query values, no masking
- Better for encoding source information



b) A bidirectional self-attention layer

# Masked language modeling objective

- Based on “cloze” tasks:
  - “Can I have a ____ of water, please?”
  - Does that remind you of something?
- Masked Language Modeling (MLM)
  - Randomly sample tokens from the text and perform alternations
  - Predict the original inputs for each position

# Next Sentence Prediction

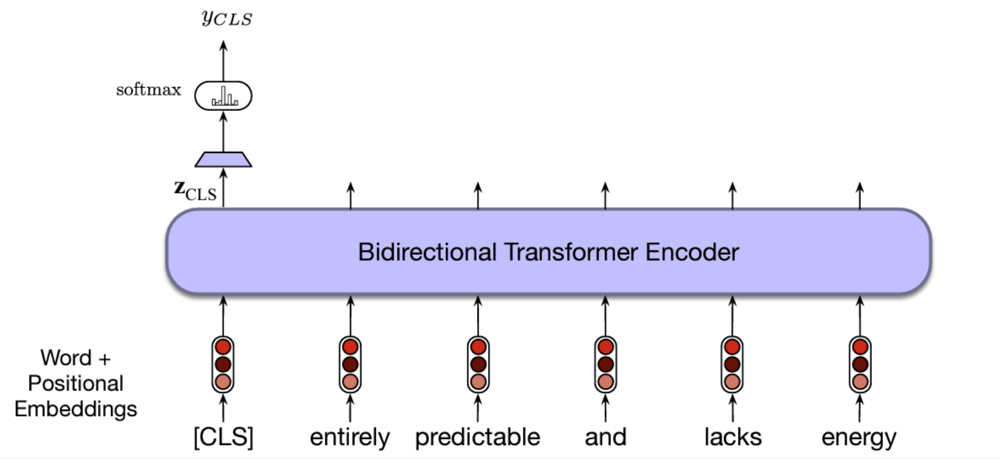
- MLM predicts relationships between words
- Transformers want to also process sentences
- Next sentence prediction task
    - Given two sentences, predict whether they are a pair of adjacent sentences

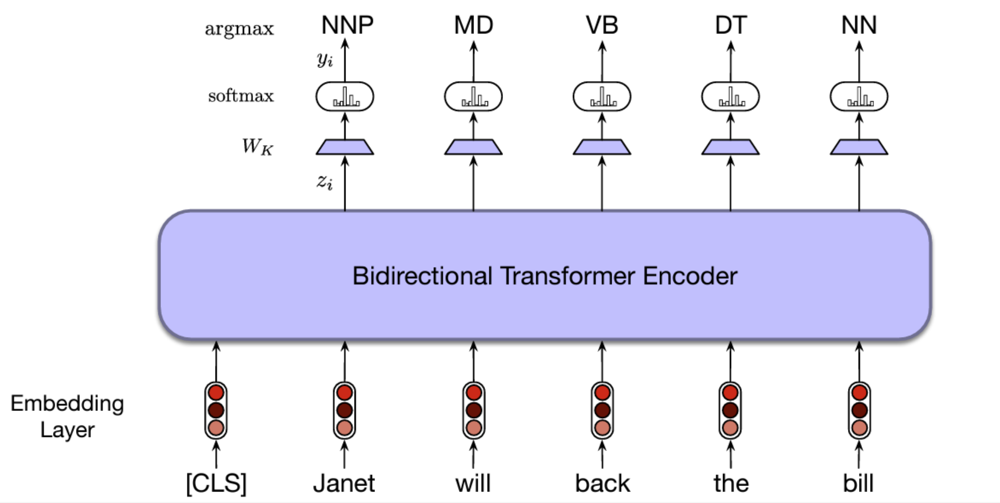
# Next sentence prediction. The CLS token.

- Next sentence prediction
  - 50 % true adjacent pairs
  - A special [CLS] token added at the beginning
  - A special [SEP] token added between texts
  - Special “sentence position” (first/second) are added to input
- When predicting the sentence relation, we use the CLS as an input to softmax

# Adapting other tasks to work with BERT

- How would we perform paraphrase identification?
  - What is the input/output/classification process?
- Performing other tasks:
  - Extractive QA
  - Sequence labeling



# The encoder-decoder model

- Encoder
  - Bi-directional attention can “see” all tokens
  - Follows the architecture we have seen last week
- Decoder
  - Causal attention
  - Additional Multiheaded Attention
    - Why do we need it?
    - What would be the Q, K, V used by it?

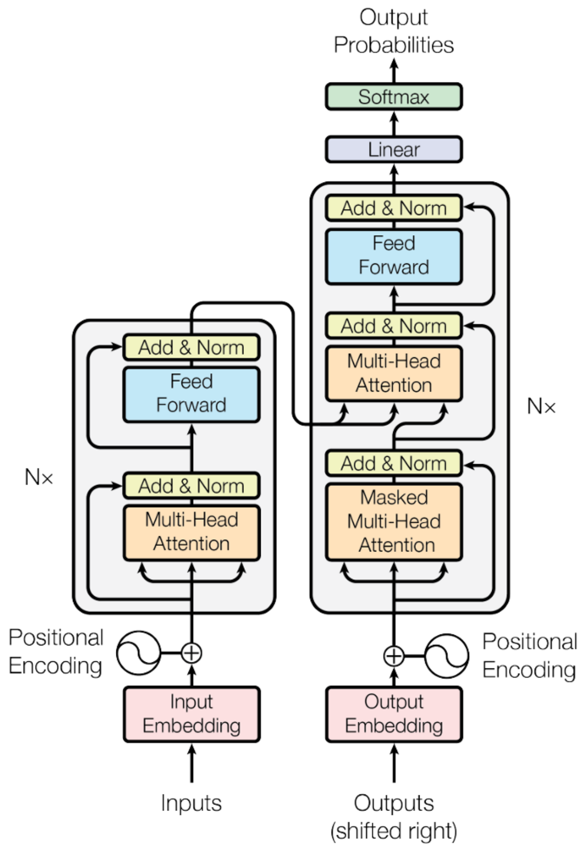
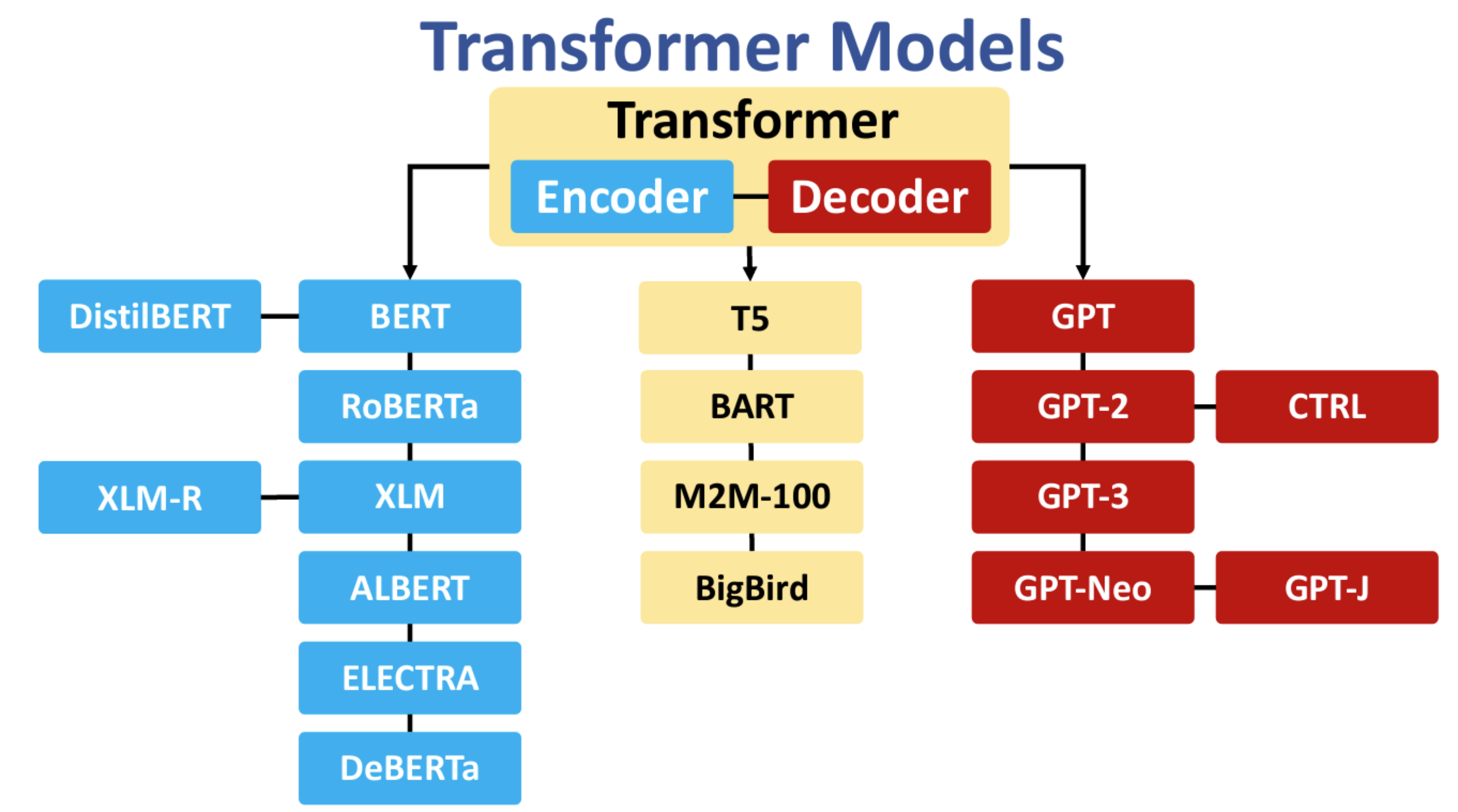



Figure 1: The Transformer - model architecture.

# The transformer family tree




Source: Lewis Tunstall, Leandro von Werra, and Thomas Wolf (2022), Natural Language Processing with Transformers: Building Language Applications with Hugging Face, O'Reilly Media.

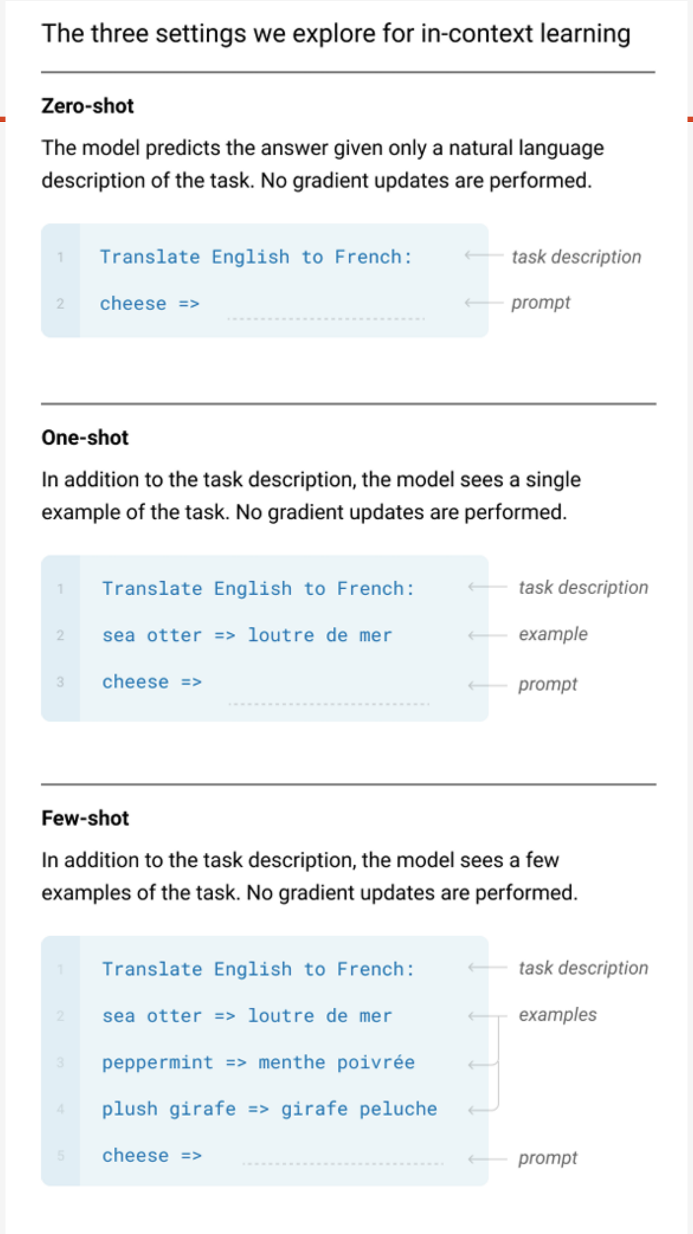
# In-context learning and RLHF finetuning

# In-context learning

- Taking transfer learning to the extreme
- Using the input to specify the task
  - “What is the sentiment of the following text: I like this movie, it’s the best in the Avengers series!”
  - “Do those sentences contradict each other: I bike to work every day. <SEP> I drive to work every day.”
- Emerging property
  - A by-product of scaling the model above a certain size

# Zero- One- and Few-shot learning

- Three different experimental conditions
- No gradient update or finetuning
- The only difference – number of examples



# InstructGPT – training models to follow instructions

- Scale is not everything
  - Hallucinations
  - Toxicity
  - Lack of helpfulness
- Improve the training (and evaluation) procedures
- “Align” models with their users

# The LM training objective

- Language modeling is not “following instructions”
- Language modeling does not take (individual) preferences
- Every training sequence is equally important
- Preference in output (in language modeling) depends on
    - The observed frequency in training data
    - The sampling strategy
- Few- and Zero-shot learning are an “emerging” side effect, not an intentionally defined goal

# The training process of InstructGPT