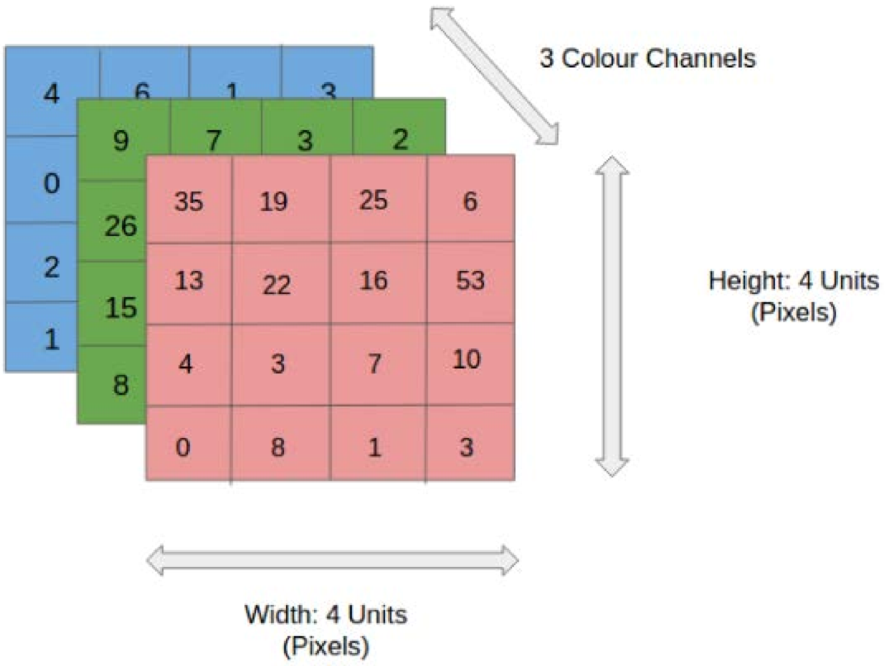
CS60050 Machine learning

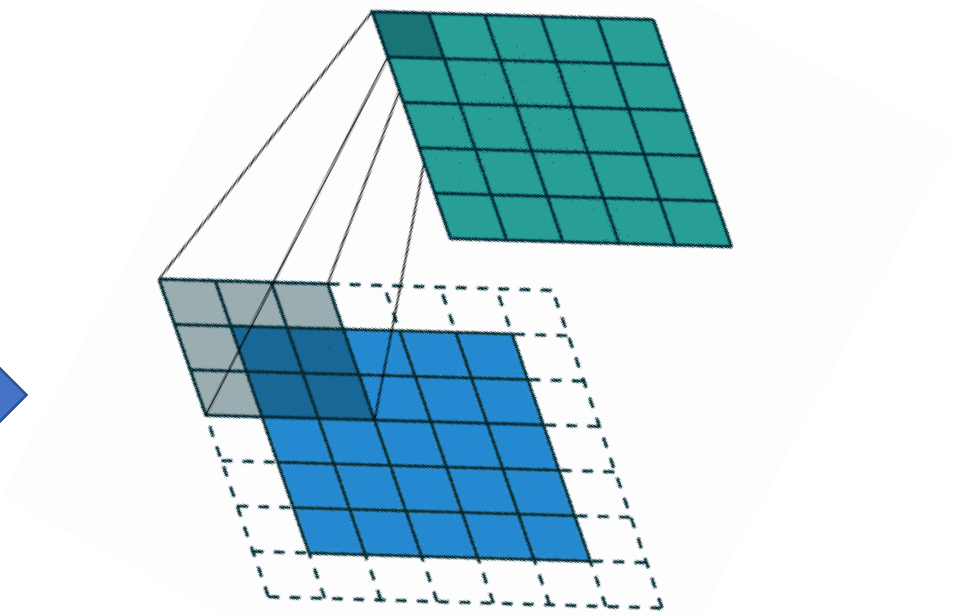
Neural Network Architectures

Sudeshna Sarkar

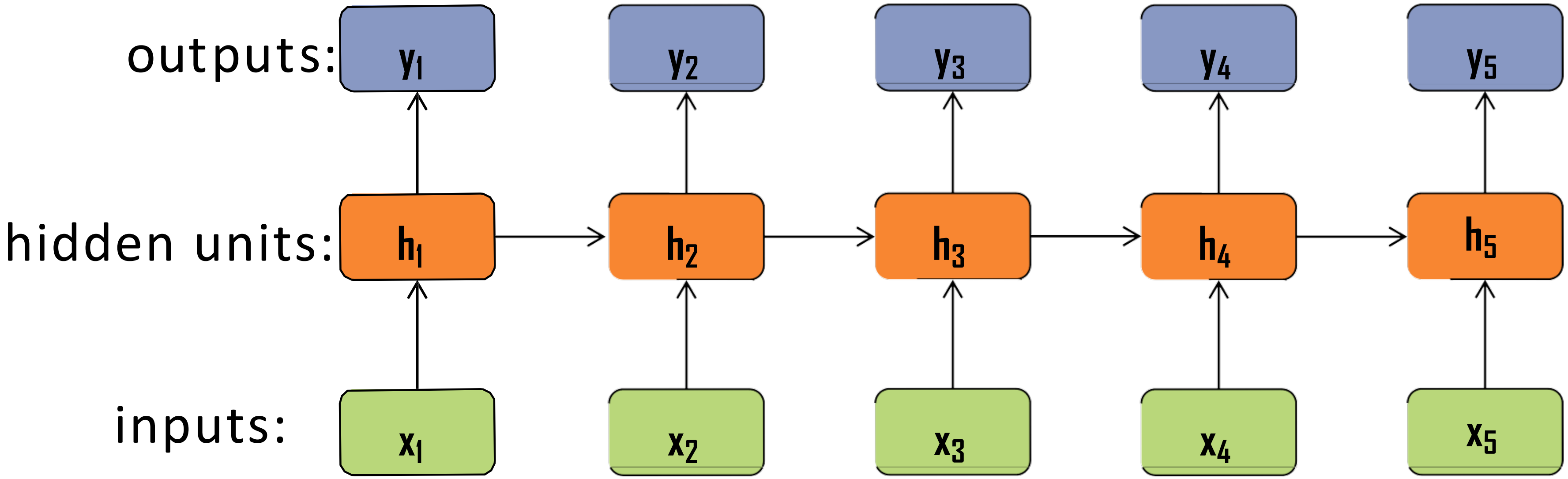
CNN and Convolution



Kernel Size: 3x3x1
Stride: 1
Zero-Padding



Recurrent Neural Networks (RNNs)



Definition of the RNN:

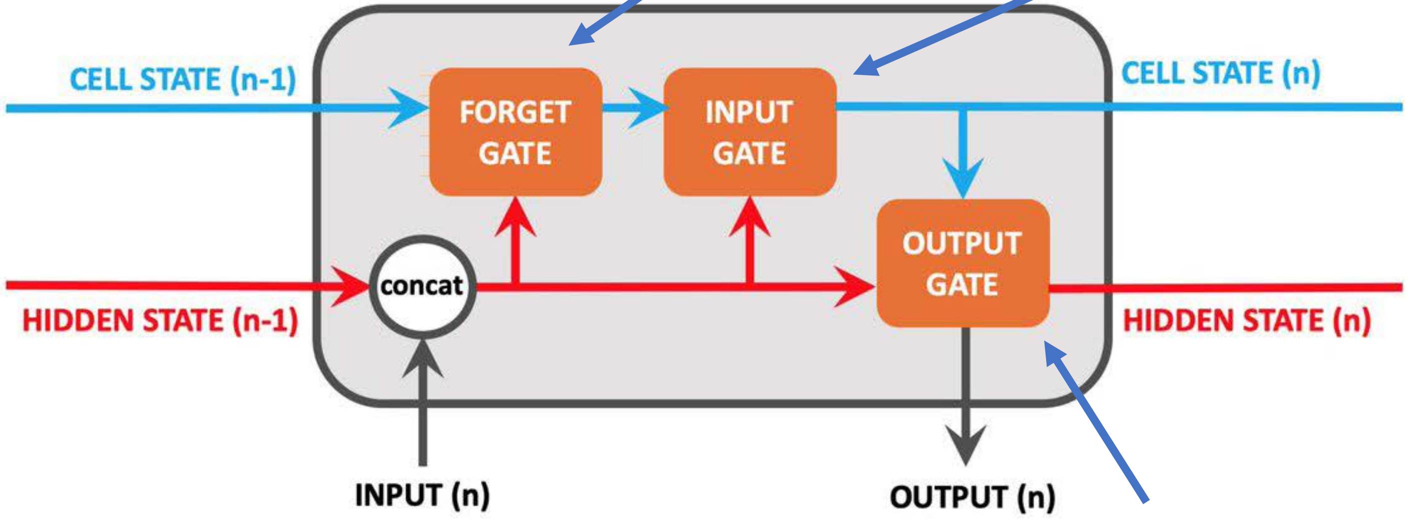
$h_t = g\,(W_{xh}\, x_t + W_{hh}\, h_{t-1} + b_h)$ $y_t = W_{hy}$
$h_t + b_y$

RNN 2.0: Long short-term Memory (LSTM)

- Hidden State: holds previous information (Short-term memory)
- Cell State: memory of the network (Long-term memory)

What should be forget in cell state

How much new data should be taken



How much does memory affect output

Sequence to Sequence Model

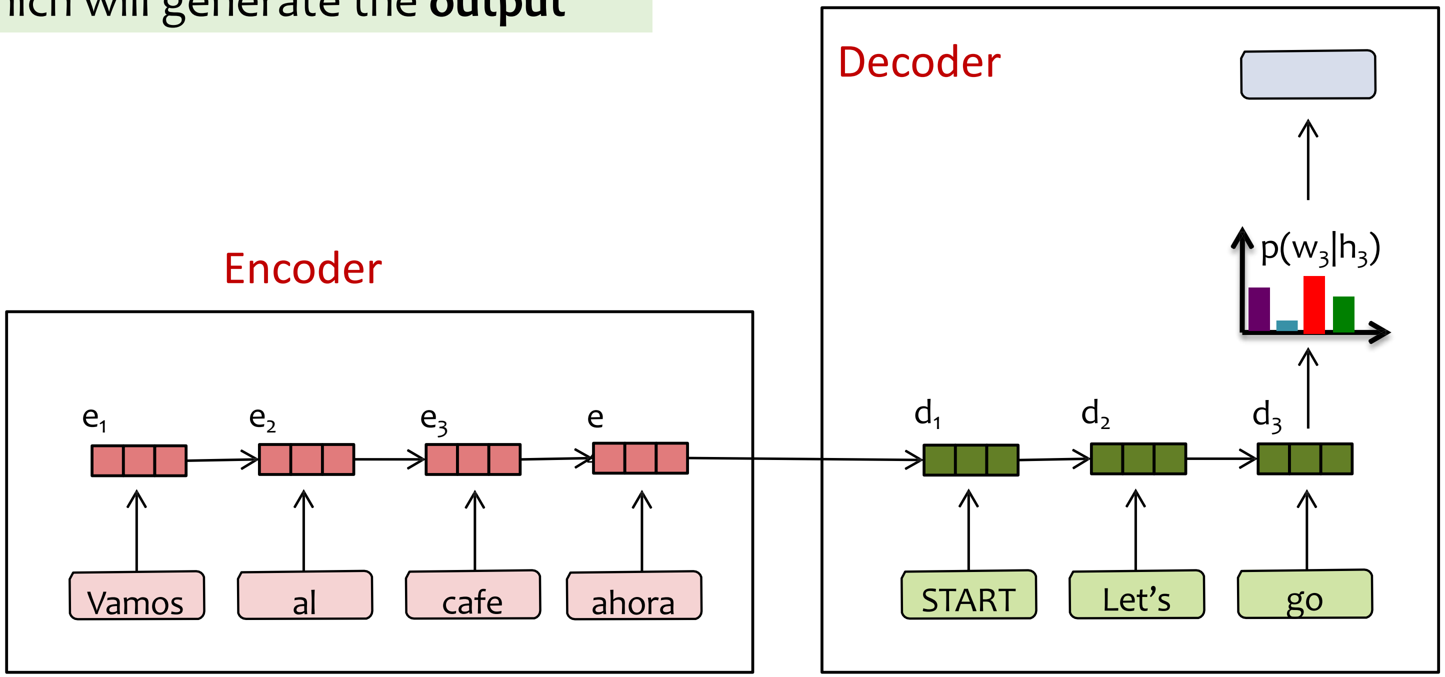
Suppose you want generate a sequence conditioned on another input

Key Idea:

1. Use an **encoder** model to generate a vector representation of the **input**
2. Feed the output of the encoder to a **decoder** which will generate the **output**

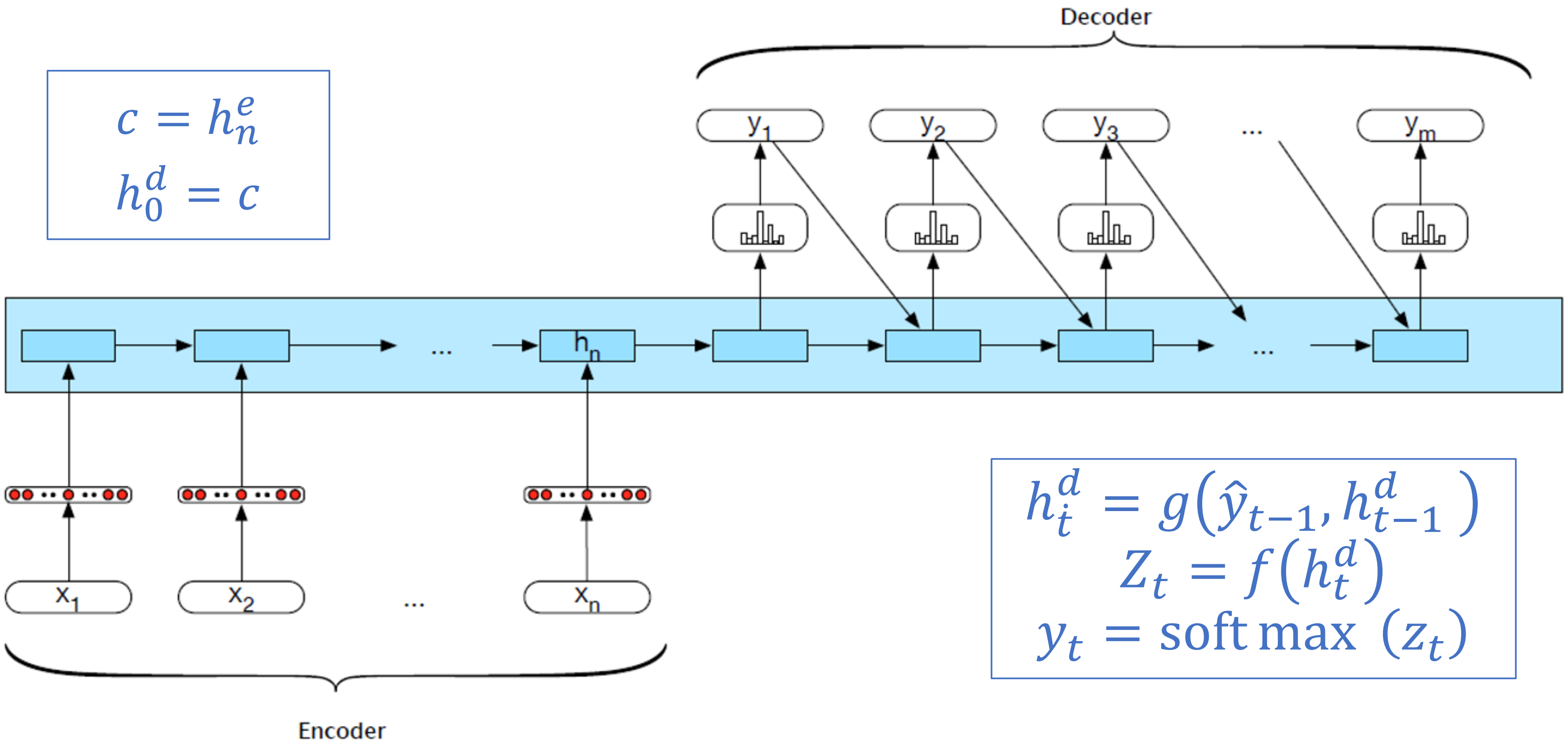
Applications:

- translation: Spanish to English
- summarization: article to summary
- speech recognition: speech signal to transcription



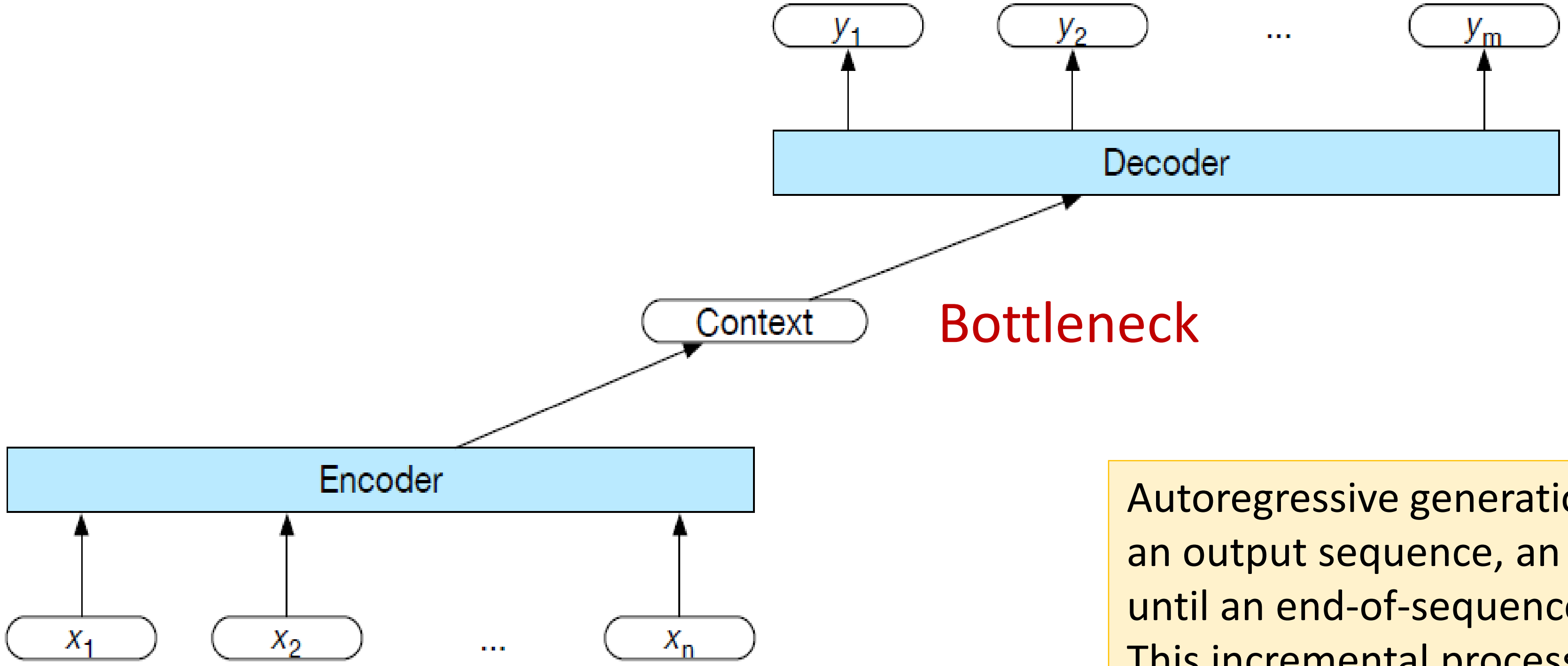
Encoder-decoder networks

$$c = h_n^e$$
$$h_0^d = c$$



$$h_t^d = g\left(\hat{y}_{t-1}, h_{t-1}^d\right)$$
$$Z_t = f\left(h_t^d\right)$$
$$y_t = \text{soft max}\ (z_t)$$

General Encoder Decoder Model

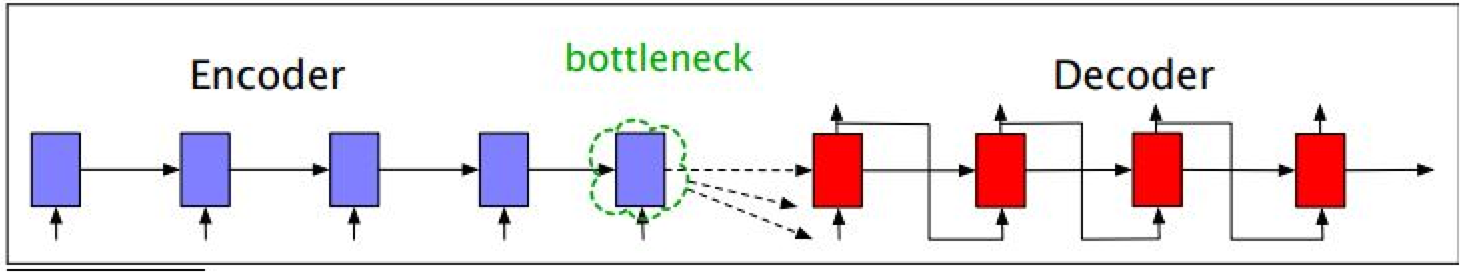


Bottleneck

Autoregressive generation is used to produce an output sequence, an element at a time, until an end-of-sequence marker is generated. This incremental process is guided by the context provided by the encoder as well as any items generated for earlier states by the decoder.

RNNs, LSTMs, GRUs, CNN, Transformers

Bottleneck



Weaknesses of the context vector:

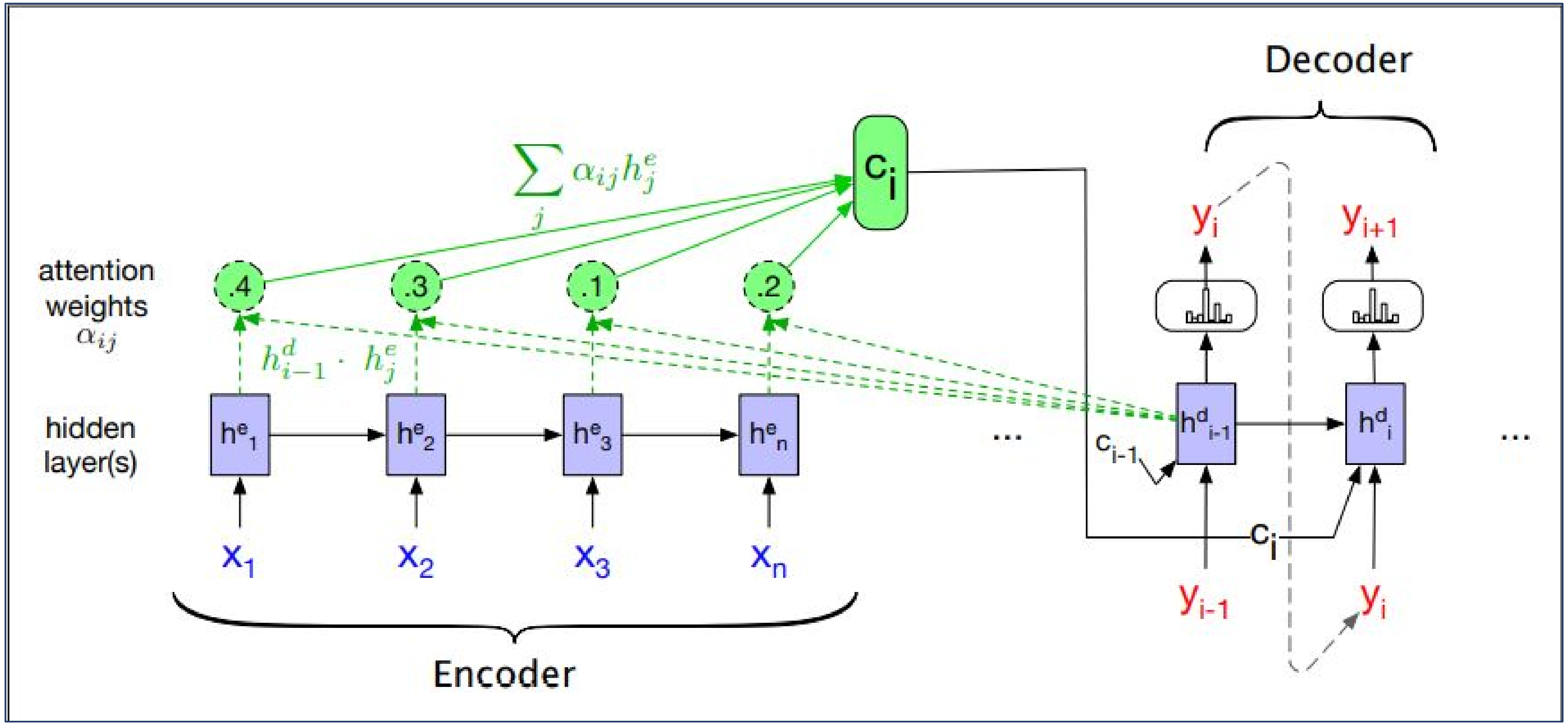
- Only directly available at the beginning of the process and its influence wanes as the output sequence is generated
- Context vector is a function (e.g. last, average, max, concatenation) of the hidden states of the encoder. This approach loses useful information about each of the individual encoder states

Encoder Decoder Attention

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

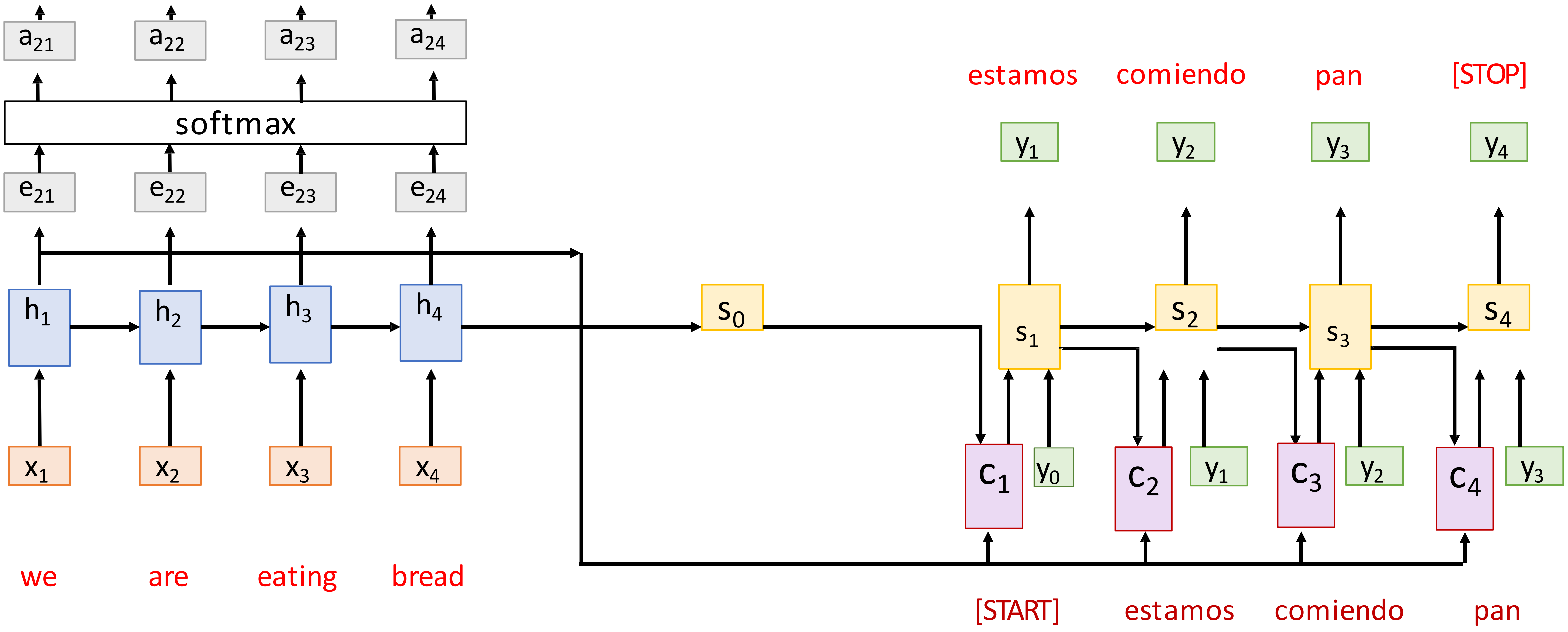
$$\alpha_{i,j} = softmax(score(h_{i-1}^d, h_j^e)\forall j \in e)$$

$$c_i = \sum_j \alpha_{i,j} h_j^e$$

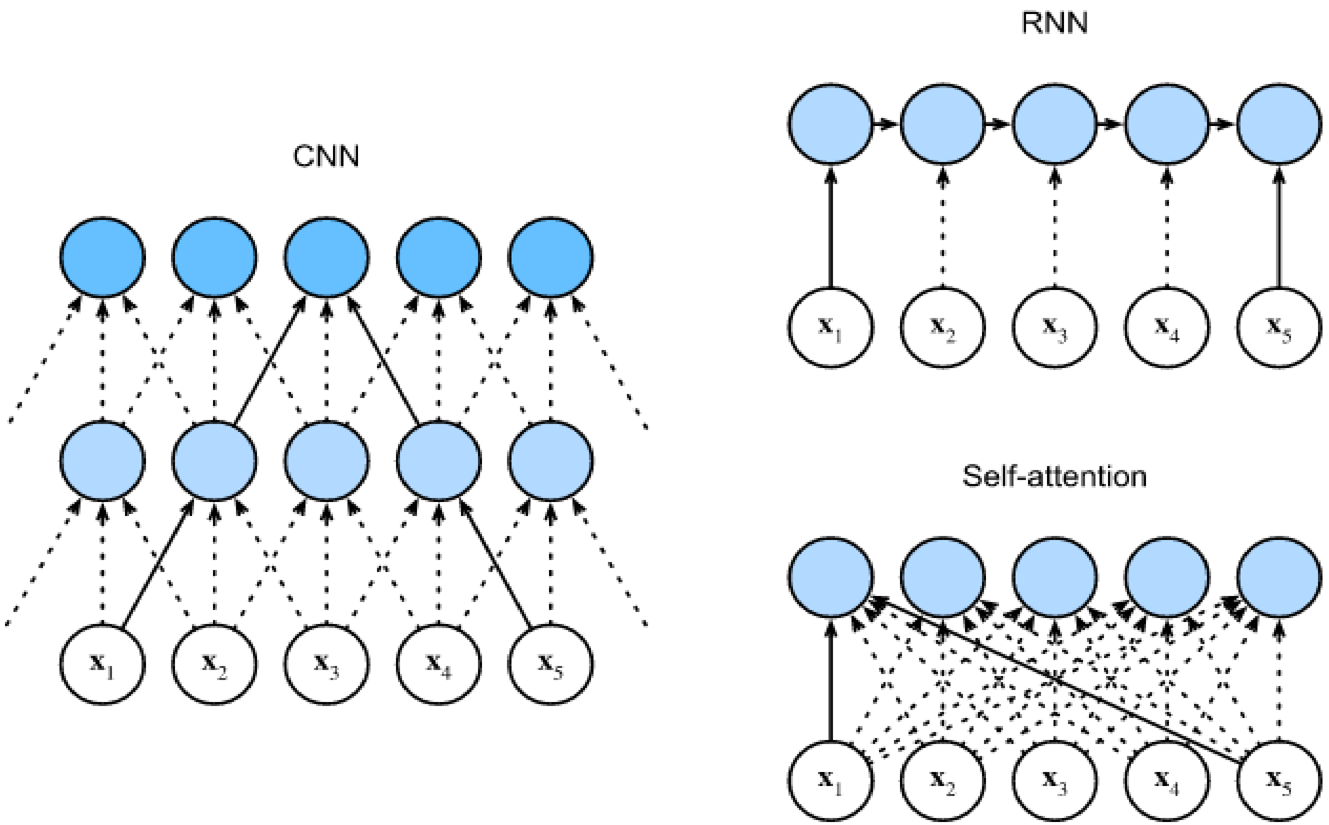


$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

Encoder Decoder Attention

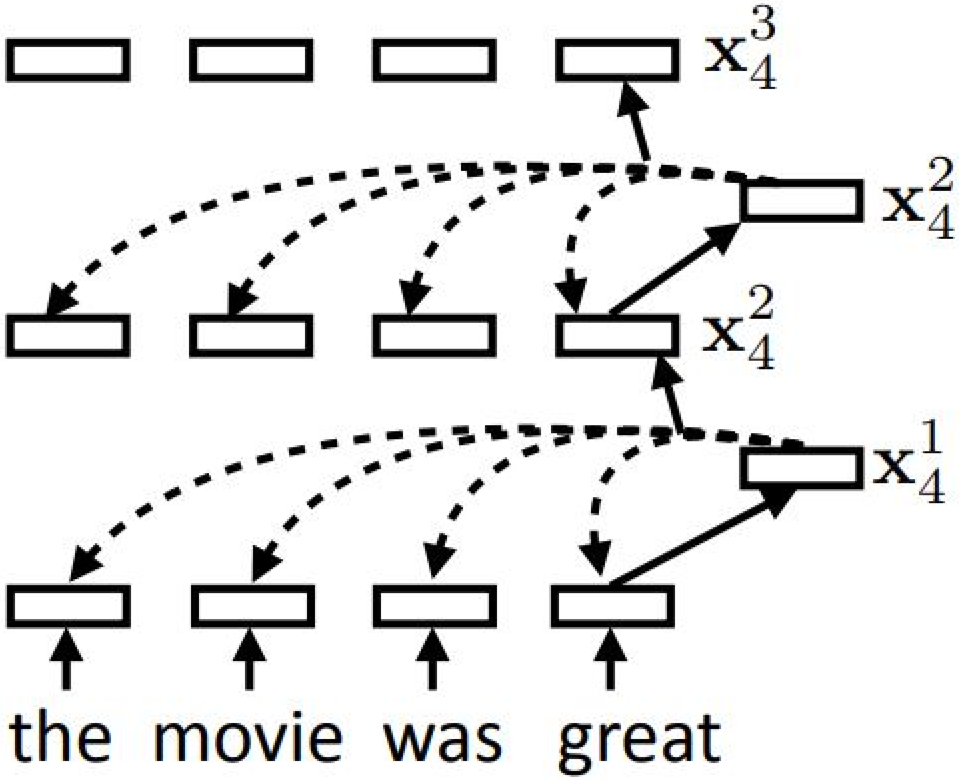


Comparing CNNs, RNNs, and Self-Attention



Self-attention

- Each word is a query to form attention over all tokens
- This generates a context-dependent representation of each token: a weighted sum of all tokens
- The attention weights dynamically mix how much is taken from each token
- Can run this process iteratively, at each step computing self-attention on the output of the previous level



Self-attention

k : level number

X : input vectors

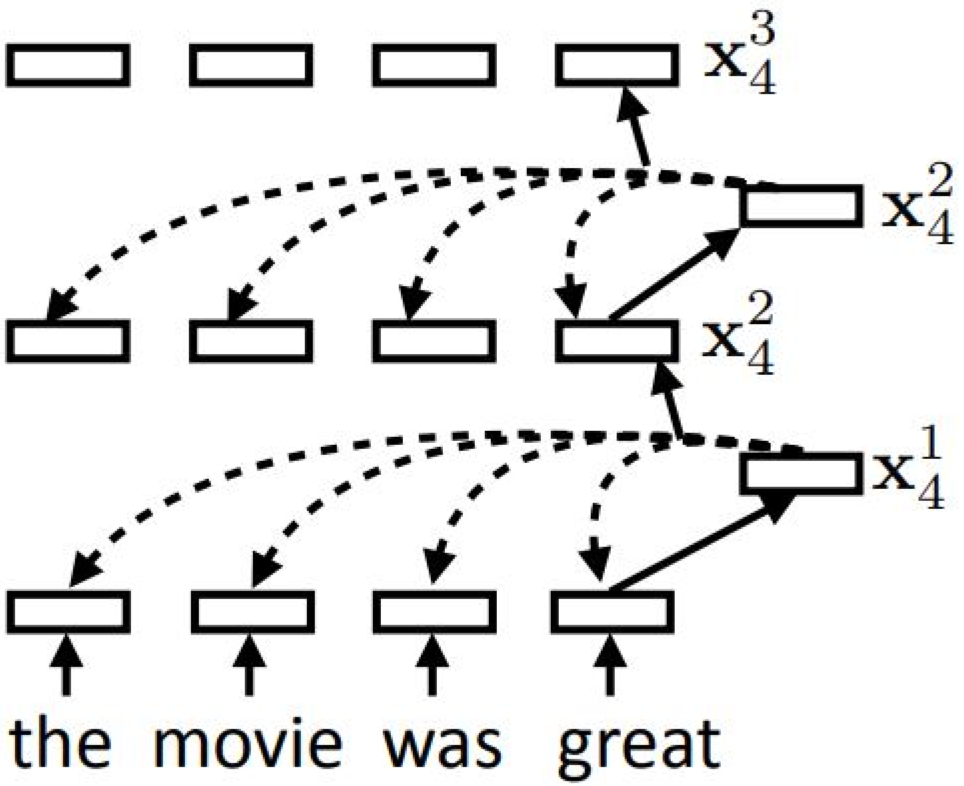
$X = \mathbf{x}_1, \dots, \mathbf{x}_n$

$\mathbf{x}_i^1 = \mathbf{x}_i$

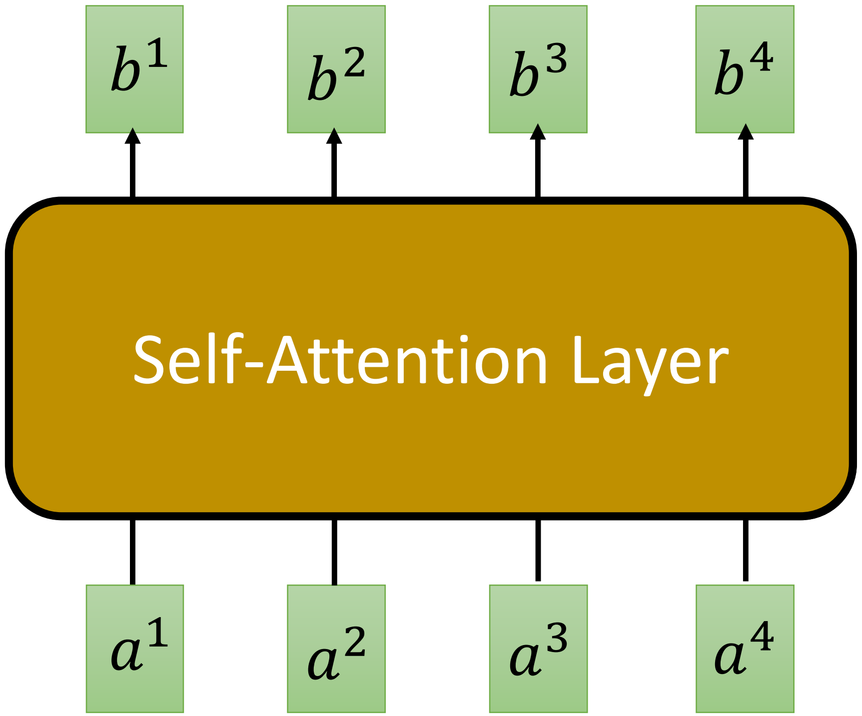
$\bar{\alpha}_{i,j}^k = \mathbf{x}_i^{k-1} \cdot \mathbf{x}_j^{k-1}$

$\alpha_i^k = softmax(\bar{\alpha}_{i,1}^k, \dots, \bar{\alpha}_{i,n}^k)$

$$x_i^k = \sum_{i=1}^{n} \alpha_{i,j}^k \mathbf{x}_j^{k-1}$$



Self-Attention



b^i is obtained based on the whole input sequence.
b^1, b^2, b^3, b^4 can be computed in parallel.

q: query (to match others)

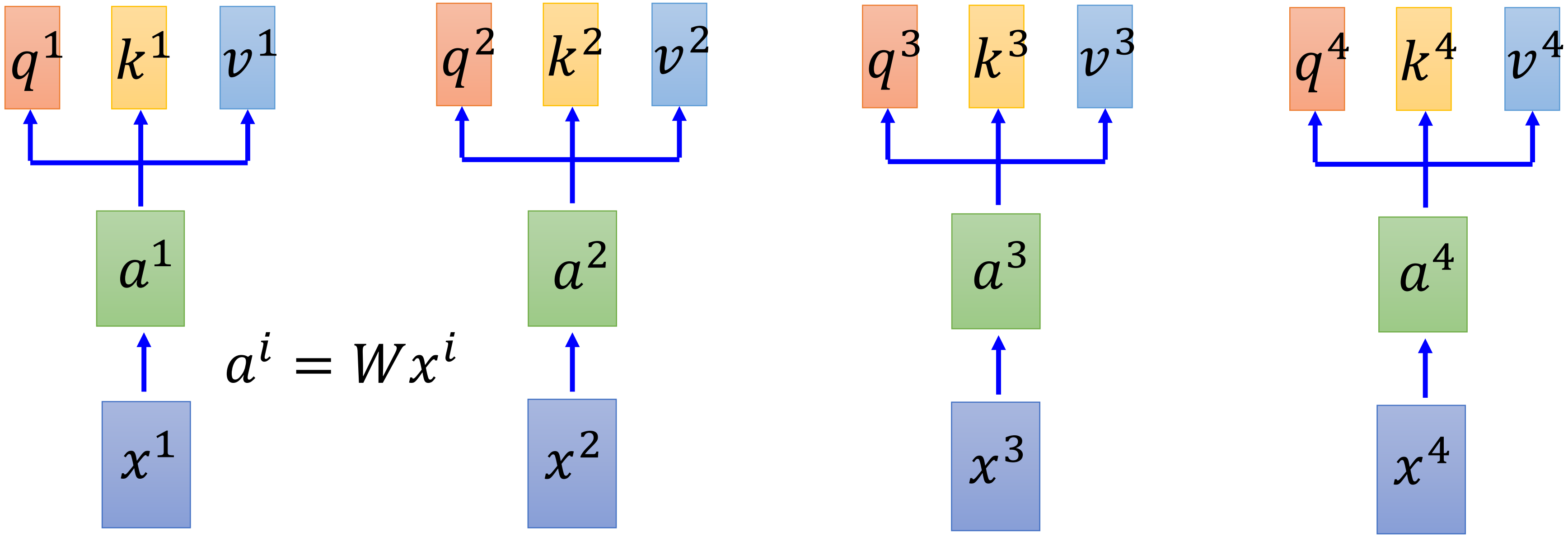
$$q^i = W^q a^i$$

k: key (to be matched)

$$k^i = W^k a^i$$

v: information to be extracted

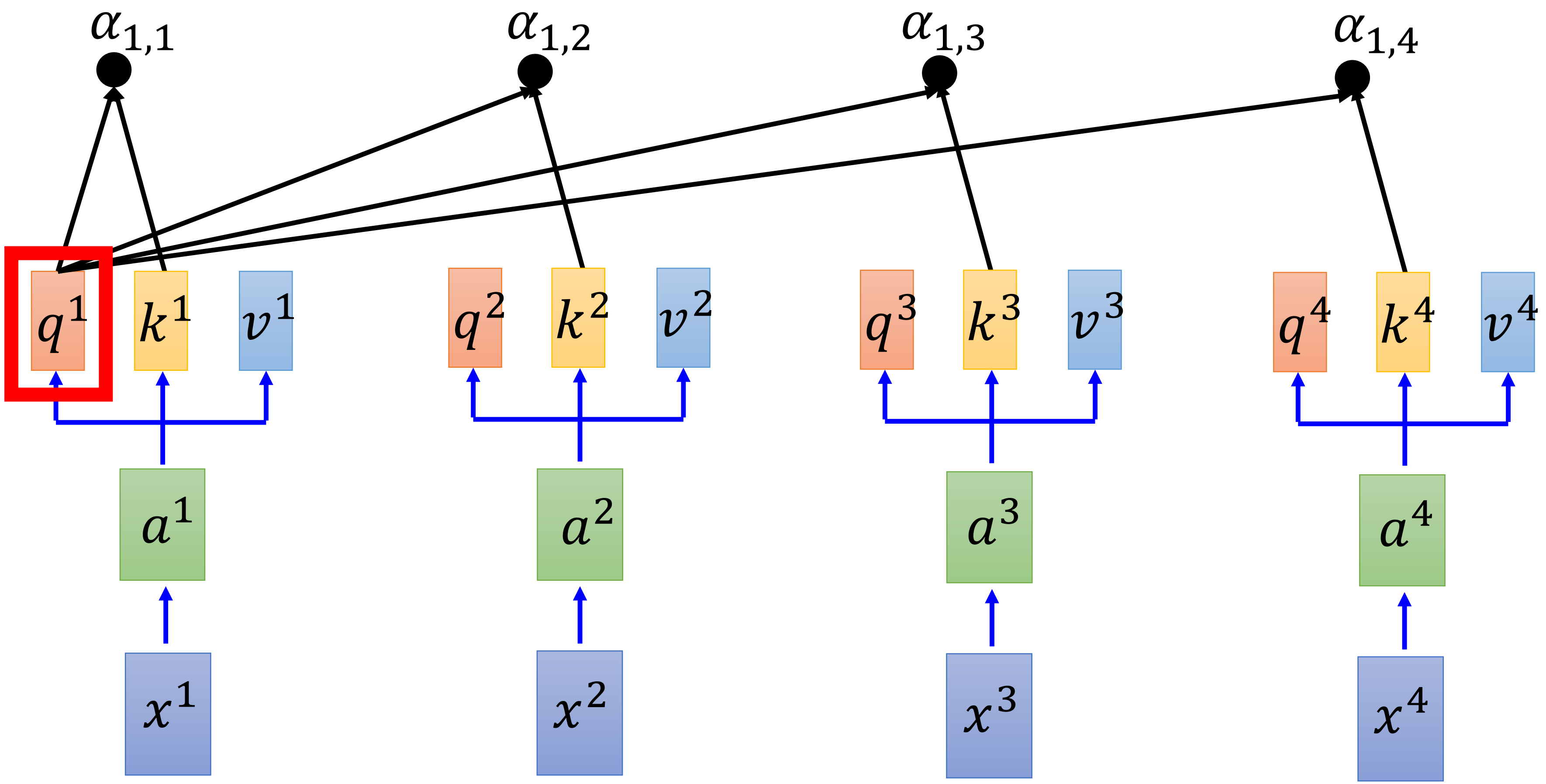
$$v^i = W^v a^i$$



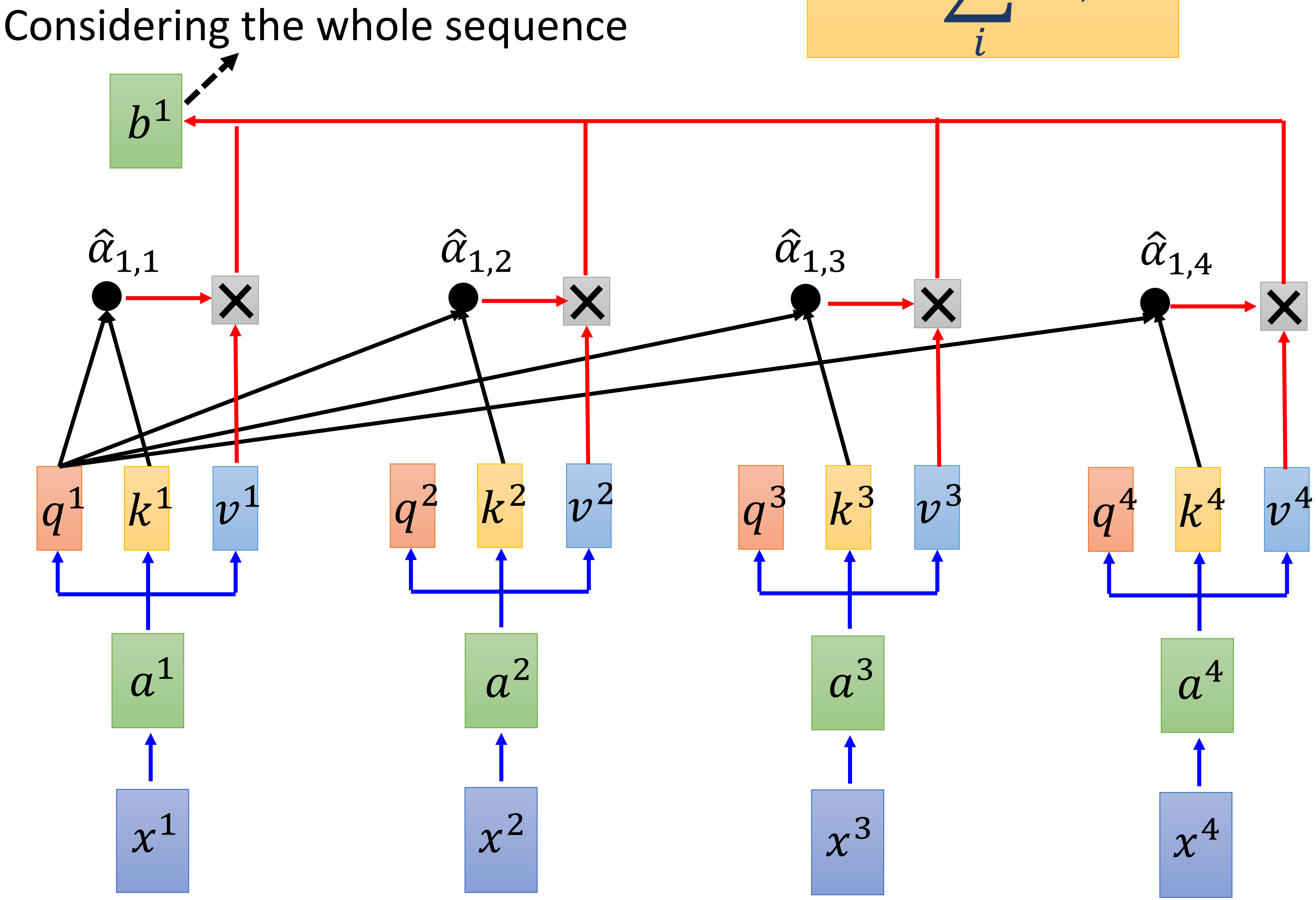
Scaled Dot-Product Attention

d is the dim of q and k

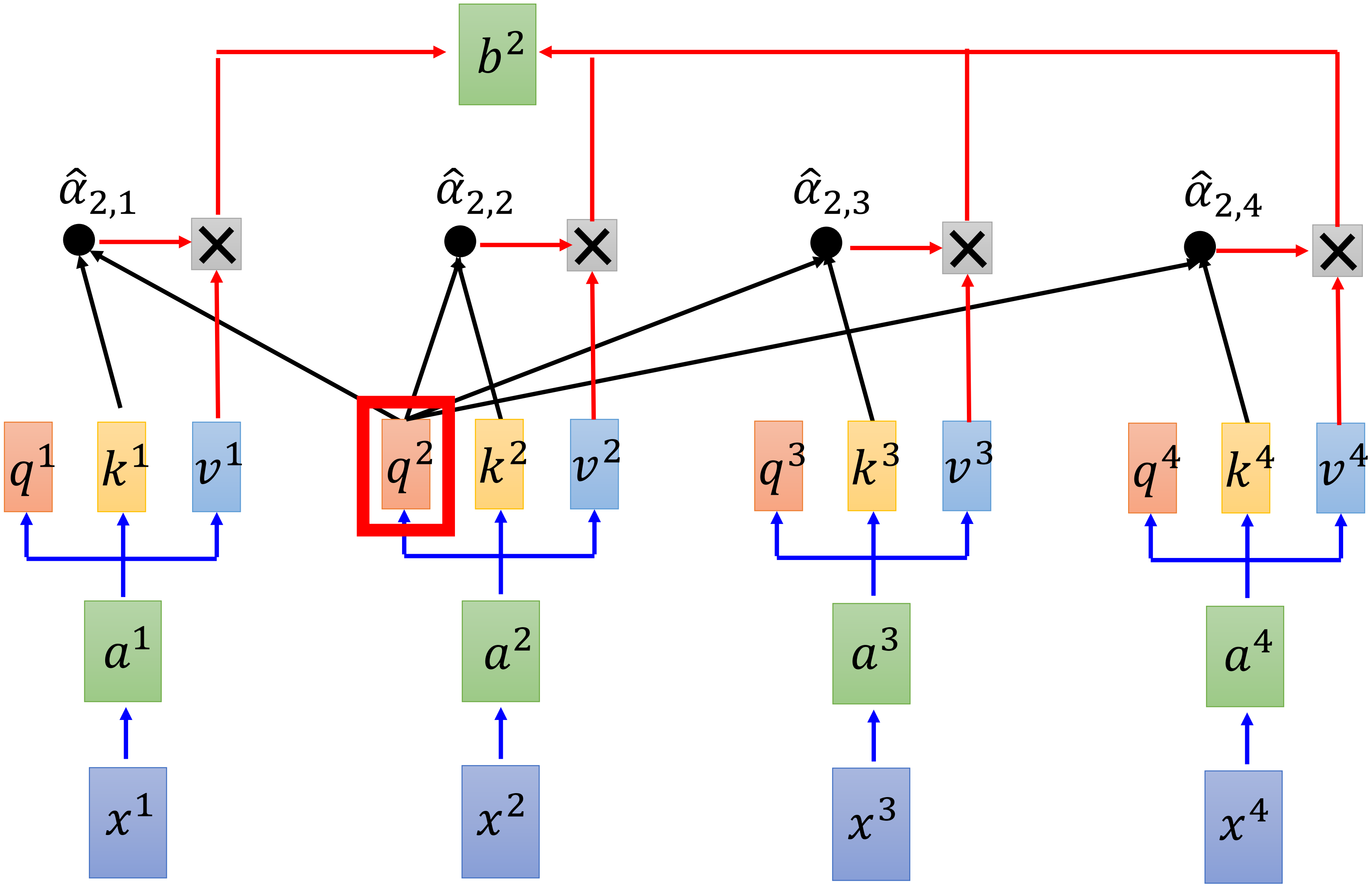
$$\alpha_{1,i} = \underbrace{q^1 \cdot k^i}_{\text{dot product}} / \sqrt{d}$$



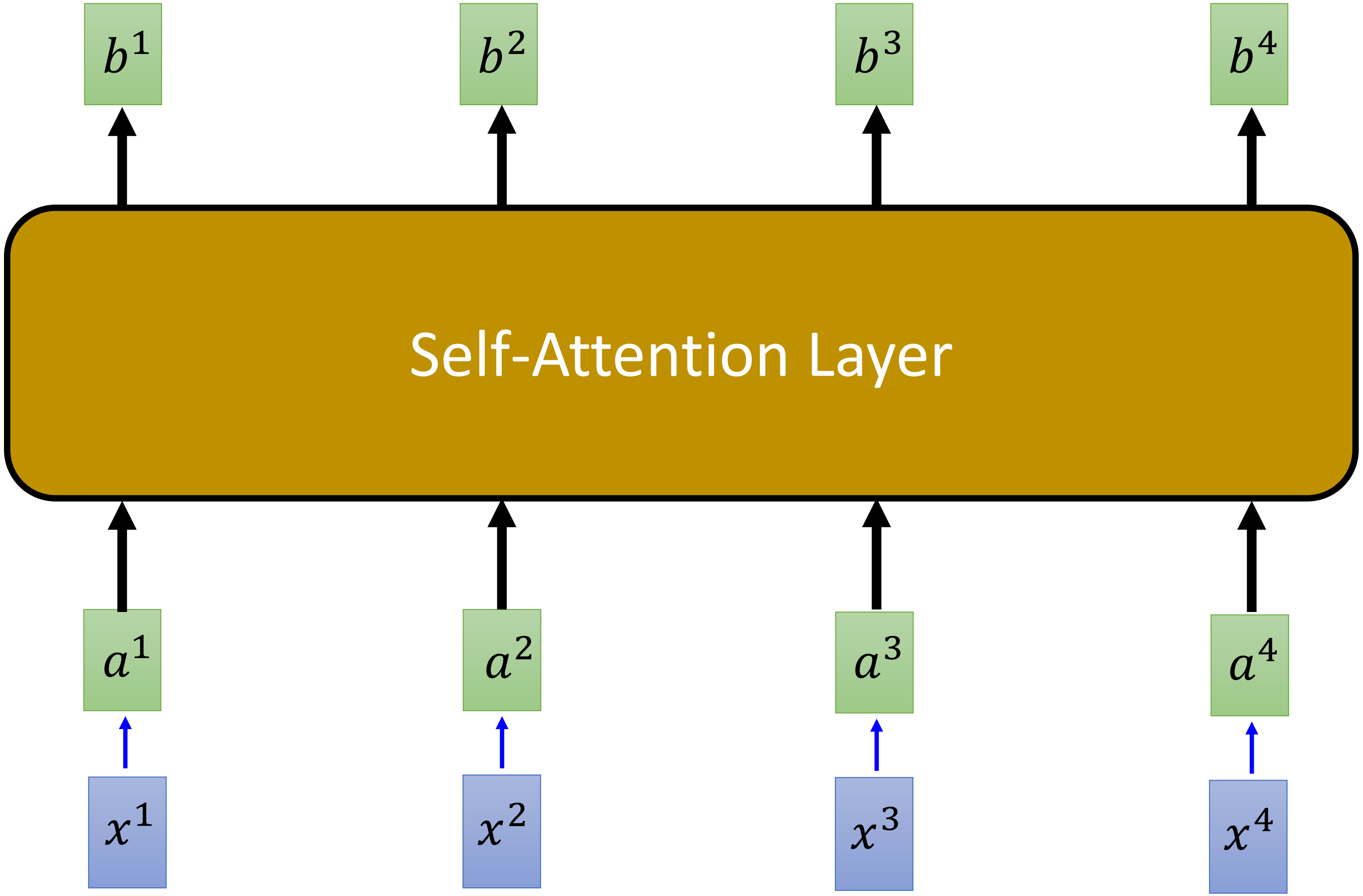
$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$



$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$

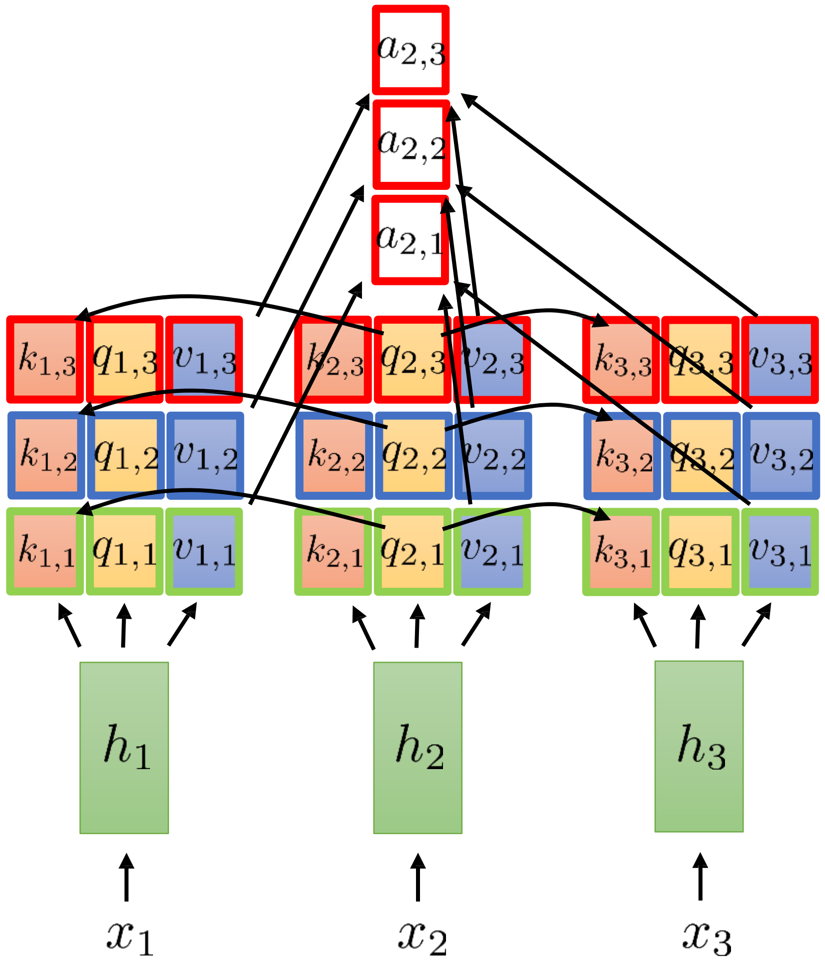


b^1, b^2, b^3, b^4 can be computed in parallel.



Multi-head attention

Idea: have multiple keys, queries, and values for every time step!



full attention vector formed by concatenation:

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

compute weights **independently** for each head

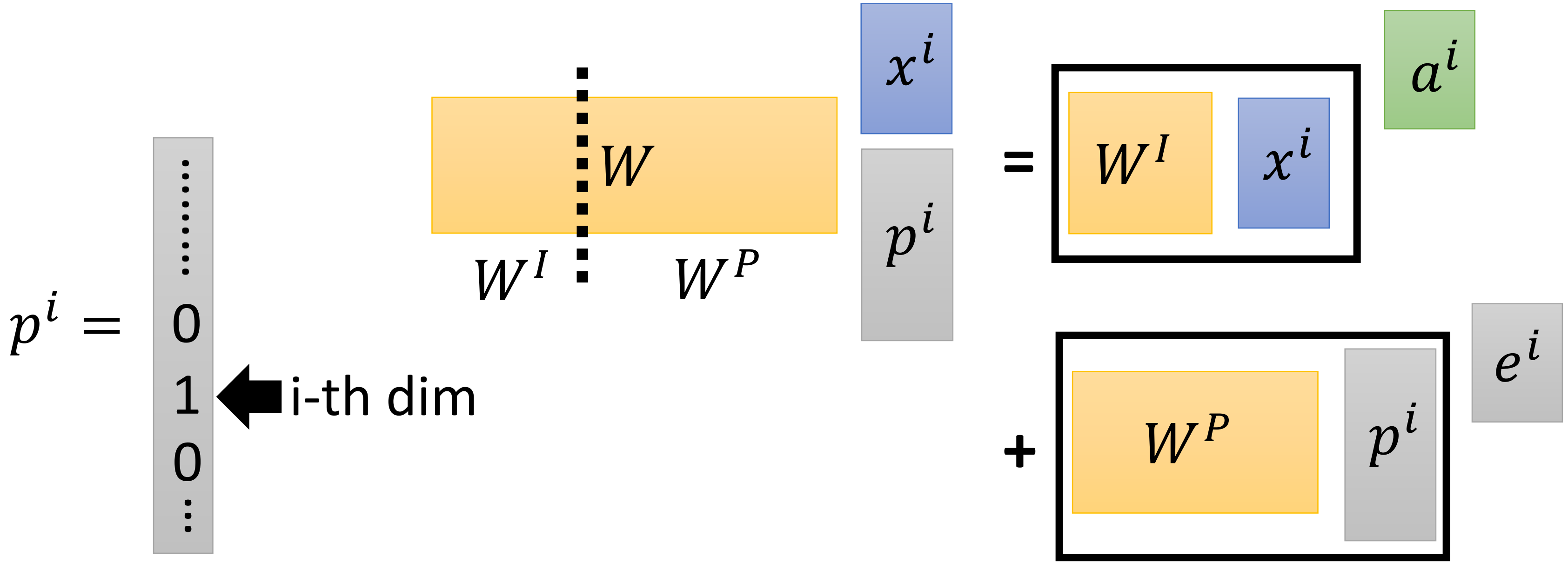
$$e_{l,t,i} = q_{l,i} \cdot k_{l,i}$$

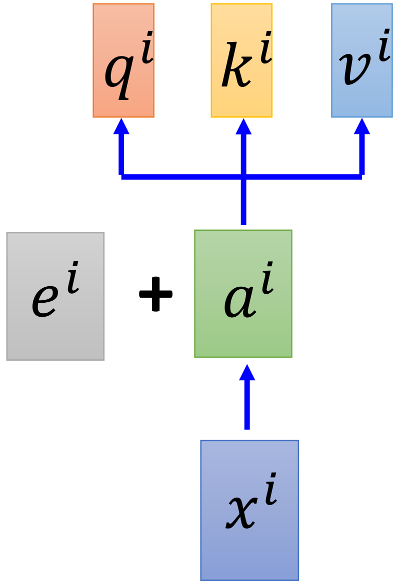
$$\alpha_{l,t,i} = \exp(e_{l,t,i}) / \sum_{t'} \exp(e_{l,t',i})$$

$$a_{l,i} = \sum_t \alpha_{l,t,i} v_{t,i}$$

around **8** heads seems to work pretty well for big models

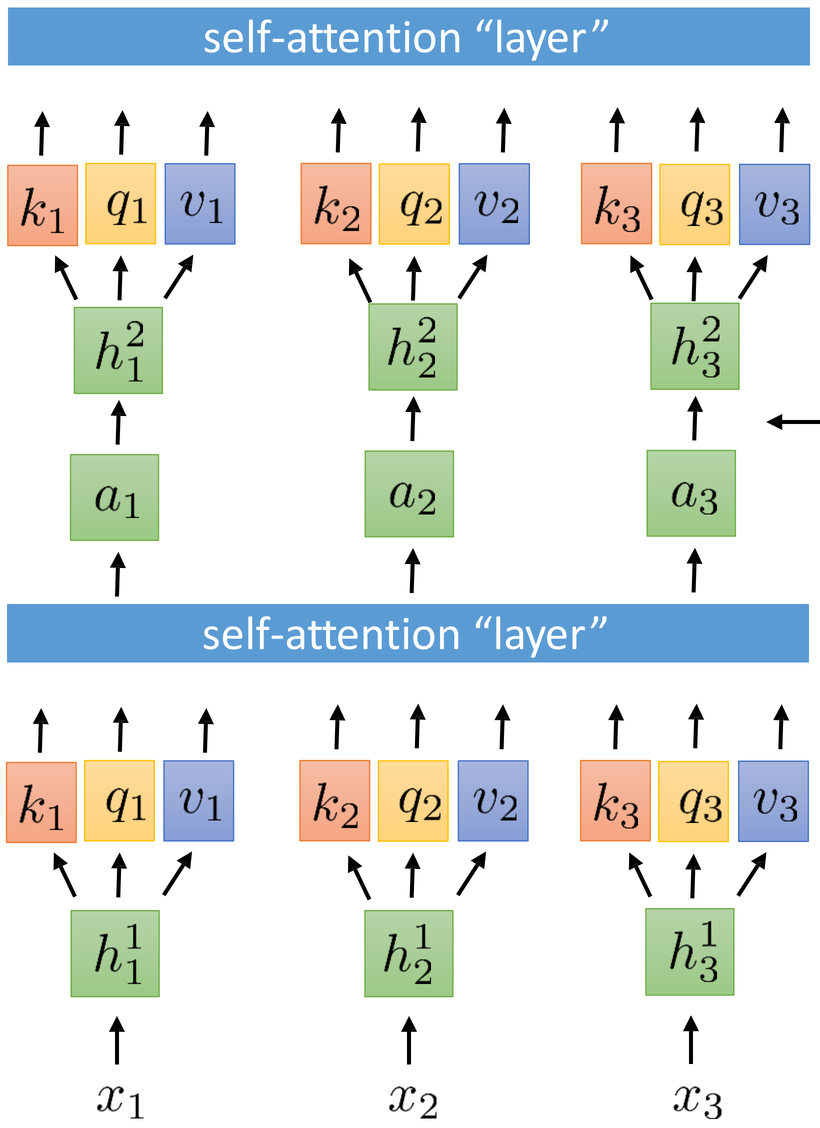
Positional Encoding





- Each position has a unique positional vector e^i (not learned from data)
- each x^i appends a one-hot vector p^i orr add them

Alternating self-attention & nonlinearity



some non-linear (learned) function
e.g., $h_t^\ell = \sigma(W^\ell a_t^\ell + b^\ell)$

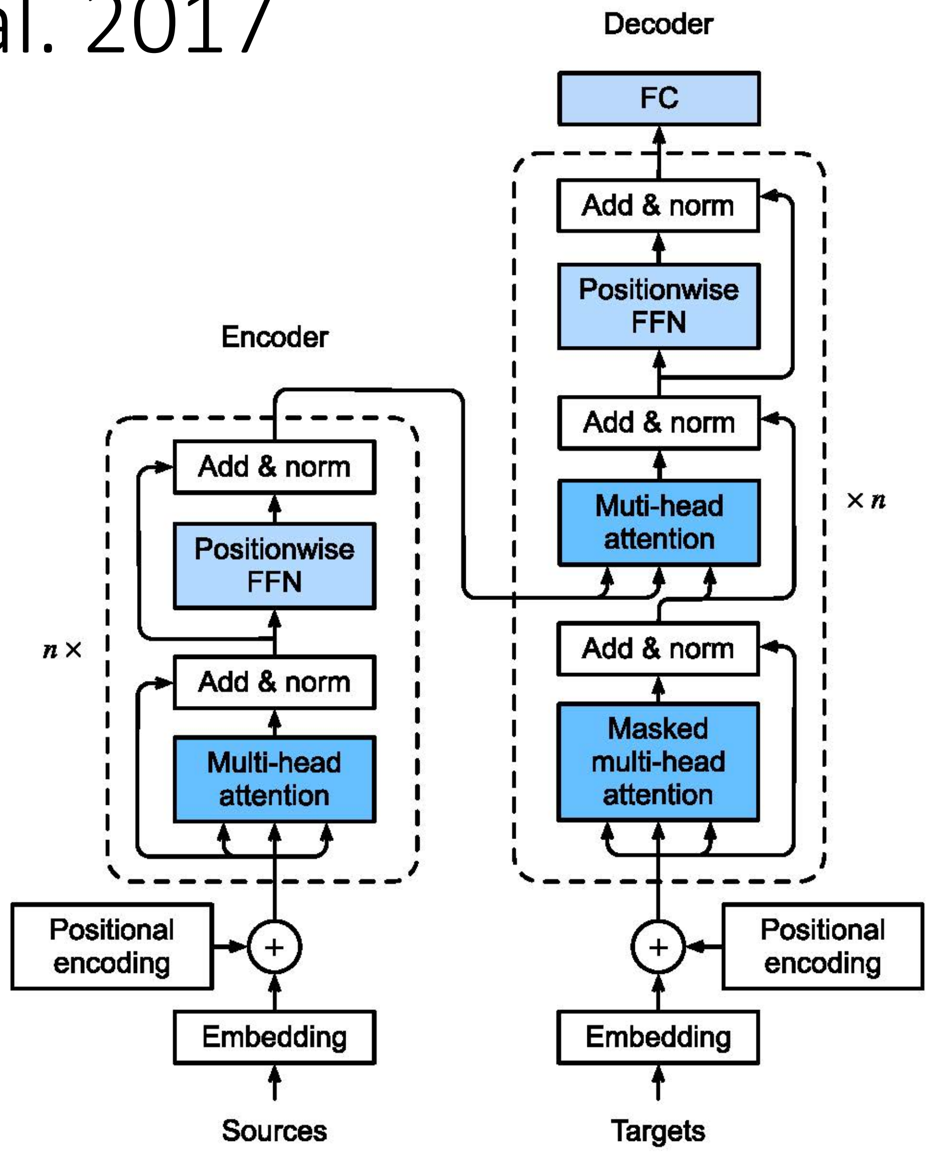
just a neural net applied at every position after every self-attention layer!

Sometimes referred to as "position-wise feedforward network"

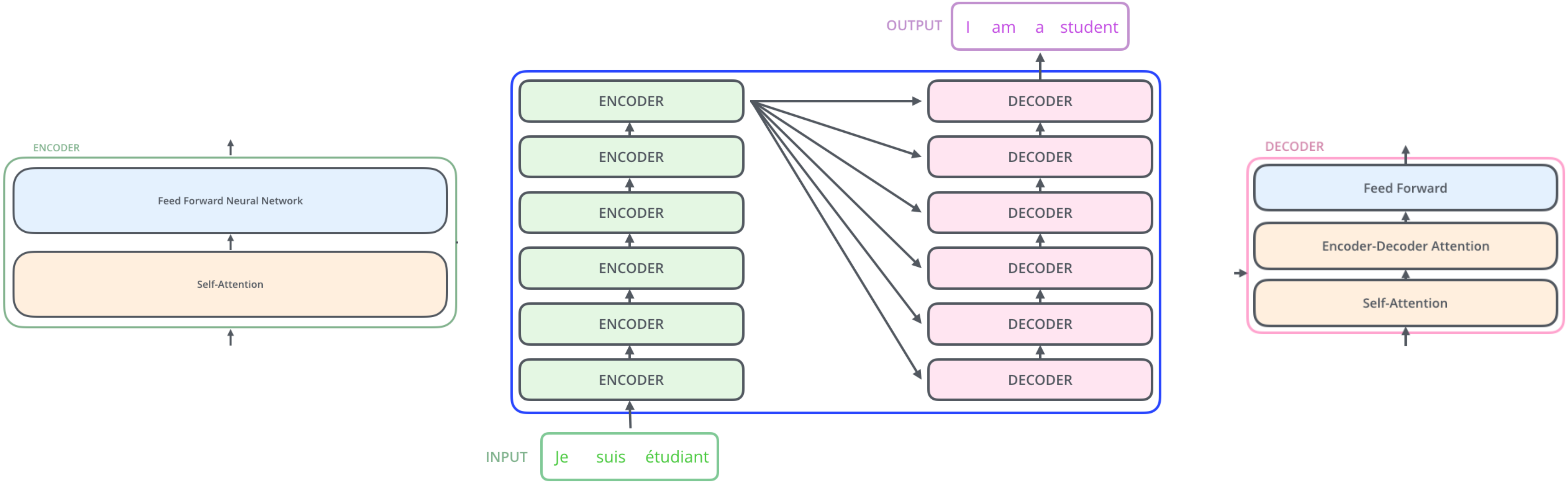
We'll describe some specific commonly used choices shortly

Transformer Model Vaswani et al. 2017

- Transformers map sequences of input vectors $(x_1, x_2, \dots, x_n)$ to sequences of output vectors $(y_1, y_2, \dots, y_m)$.
- Made up of stacks of Transformer blocks.
  - combine linear layers, feedforward networks, and self-attention layers.
- **Self-attention** allows a network to directly extract and use information from arbitrarily large contexts

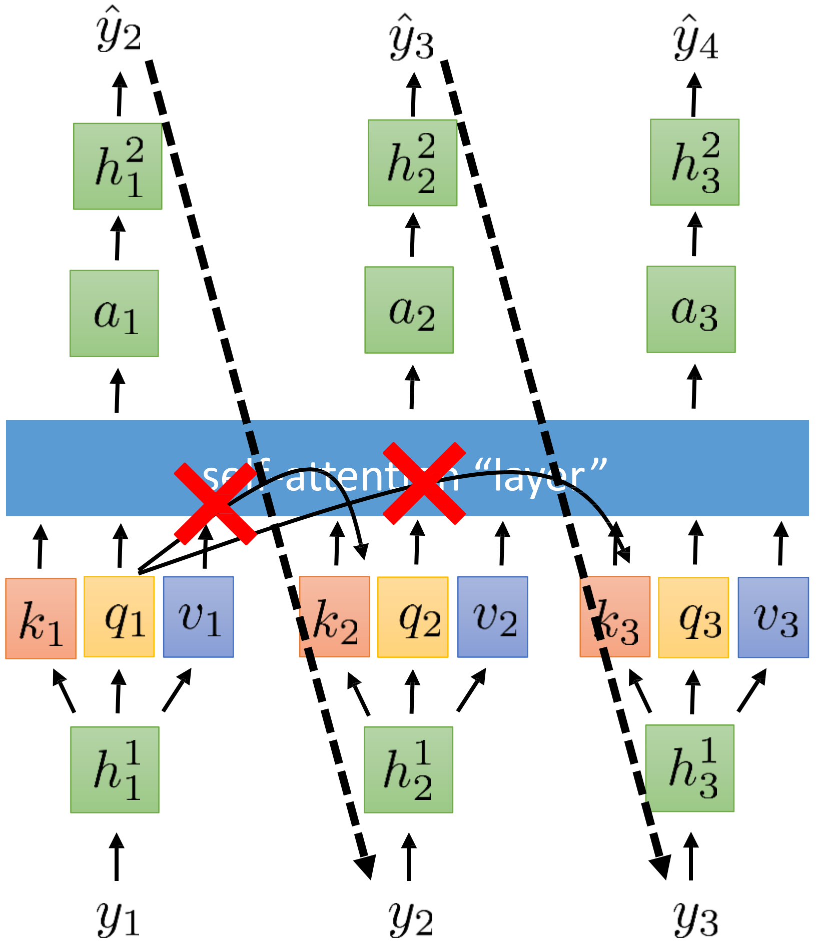


Transformer: Structure



Masked attention

A **crude** self-attention "language model":



At test time (when decoding), the **inputs** at steps 2 & 3 will be based on the output at step 1…

…which requires knowing the **input** at steps 2 & 3

Must allow self-attention into the **past**…

…but not into the **future**

Easy solution:

$$\cancel{e_{l,t} = q_l \cdot k_t}$$

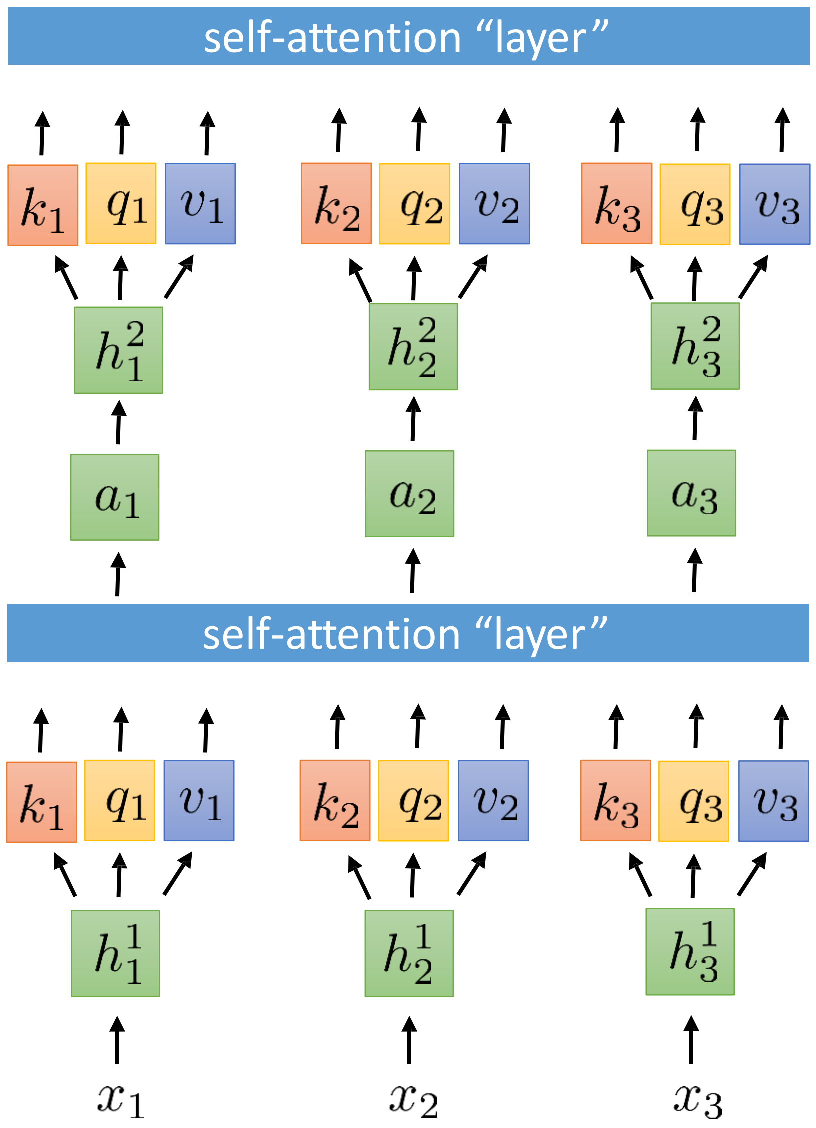
$$e_{l,t} = \begin{cases} q_l \cdot k_t & \text{if } l \geq t \\ -\infty & \text{otherwise} \end{cases}$$

in practice:

just replace $\exp(e_{l,t})$ with 0 if $l < t$

inside the softmax

Transformer summary



- Alternate self-attention “layers” with nonlinear position-wise feedforward networks (to get nonlinear transformations)
- Use positional encoding to make the model aware of relative positions of tokens
- Use multi-head attention
- Use masked attention if you want to use the model for decoding.

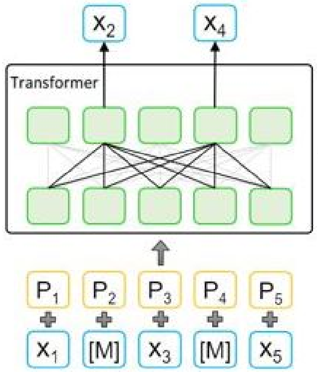
Self-supervised Models (Unsupervised Pretraining)

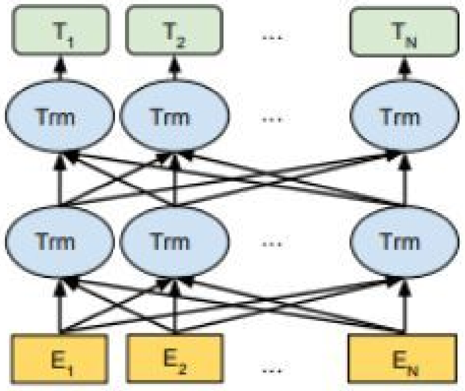
Incorporating context into word embeddings

a watershed idea in NLP

- BERT, 2018: Bidirectional Encoder Representations from Transformers (BERT, 2018)
- GPT

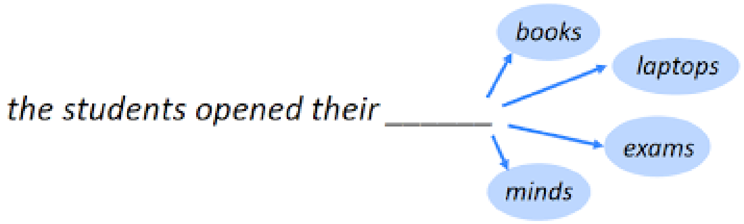
Led to significant improvements on virtually every NLP task.

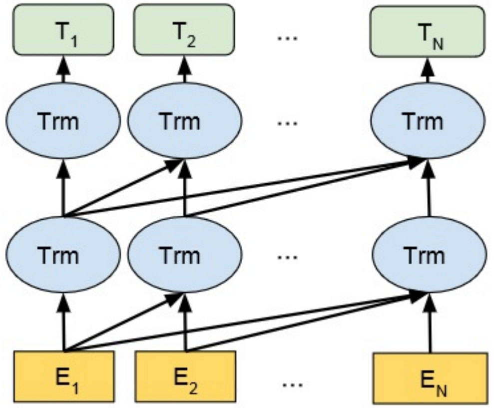




BERT Architecture

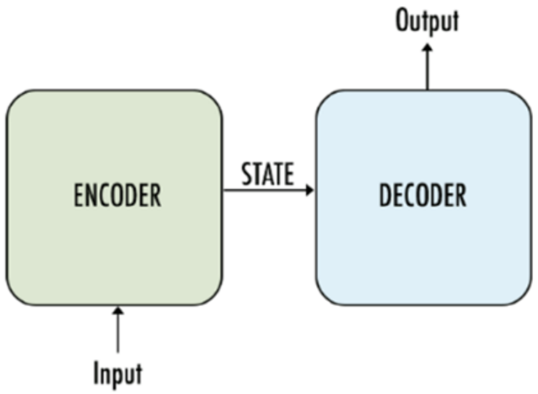
NLP
- Masked language modeling
- Predict the next word
- Next sentence prediction



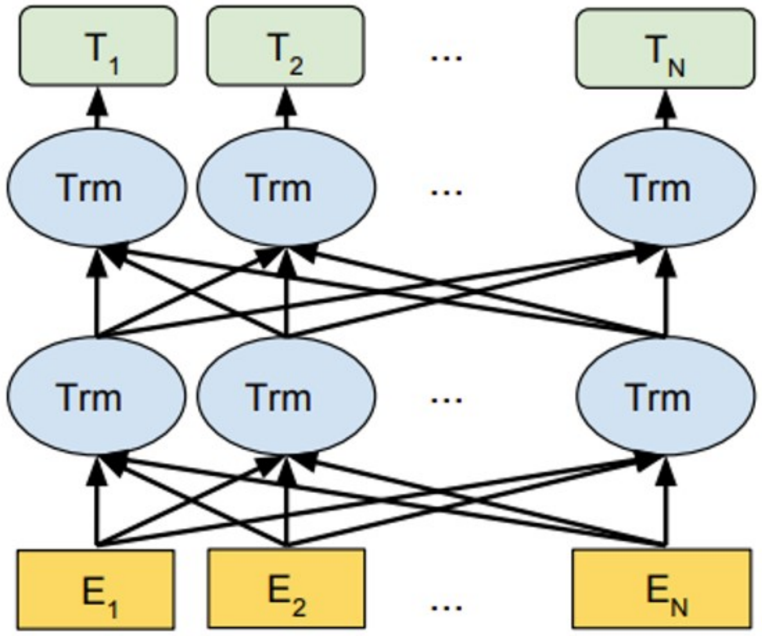


1. Build background knowledge
2. Approximate a form of common sense in AI systems.

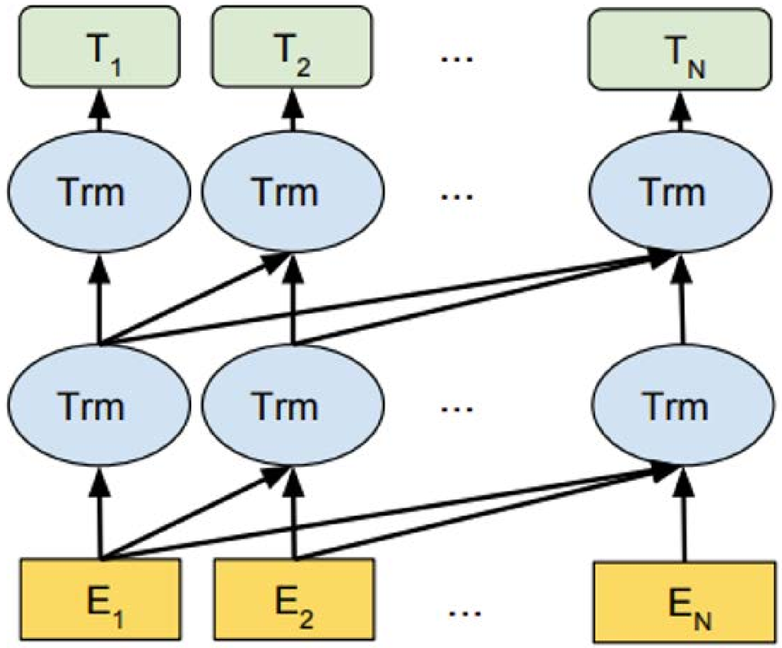
Self-attention encoder decoder



Parametric architectures for sentence denoising: Encoder

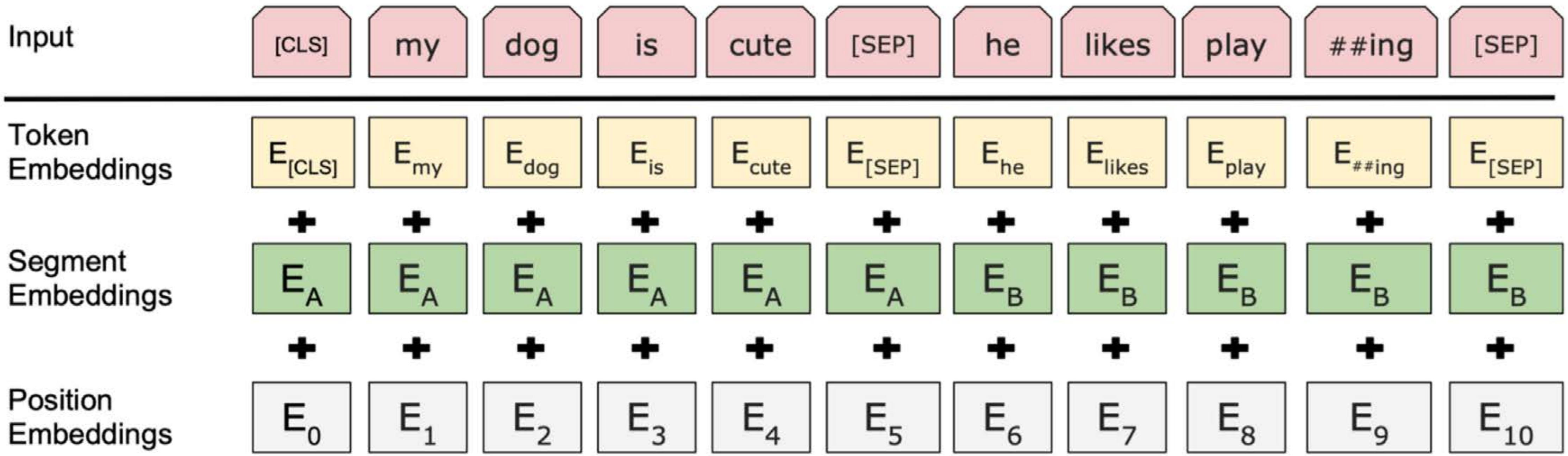


Parametric architectures for text completion: Decoder



BERT

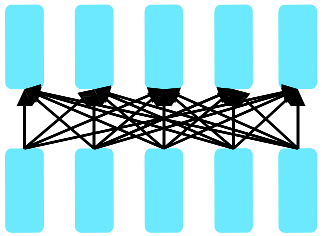
- multi-layer self-attention (Transformer)
- Input: a sentence or a pair of sentences with a separator and subword representation



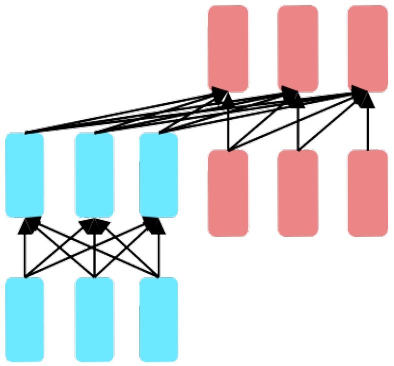
Large Language Models

A language model
- Models language
- Assigns probability to a sequence of words

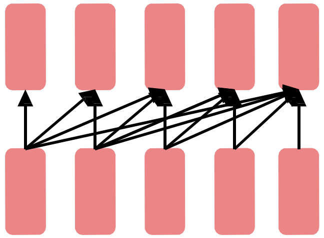
- Encoder only models: BERT, RoBERTa, Electra
- Decoder only models: GPT-n
- Encoder-decoder models: full encoder, autoregressive decoder : T5, BART



Encoders



Encoder-Decoders



Decoders

Trained by Self-supervised learning on a huge corpus.

Pre-training allows language models to learn robust task-agnostic features

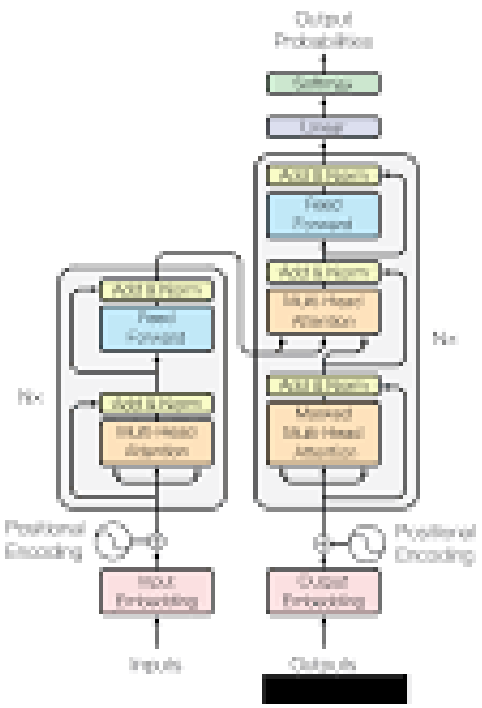
May be followed by
- Supervised fine-tuning for tasks
- Reinforcement learning with human feedback

The promise : One single model for many tasks

Three types of LLMs

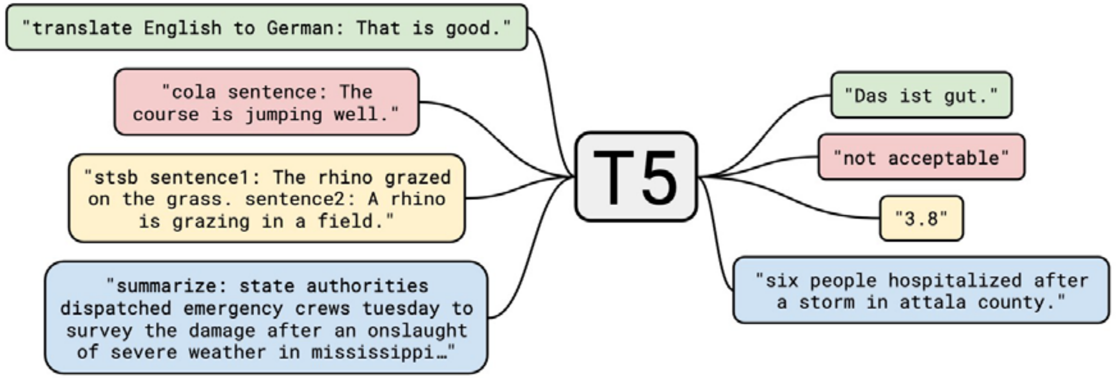
BERT

Encoder

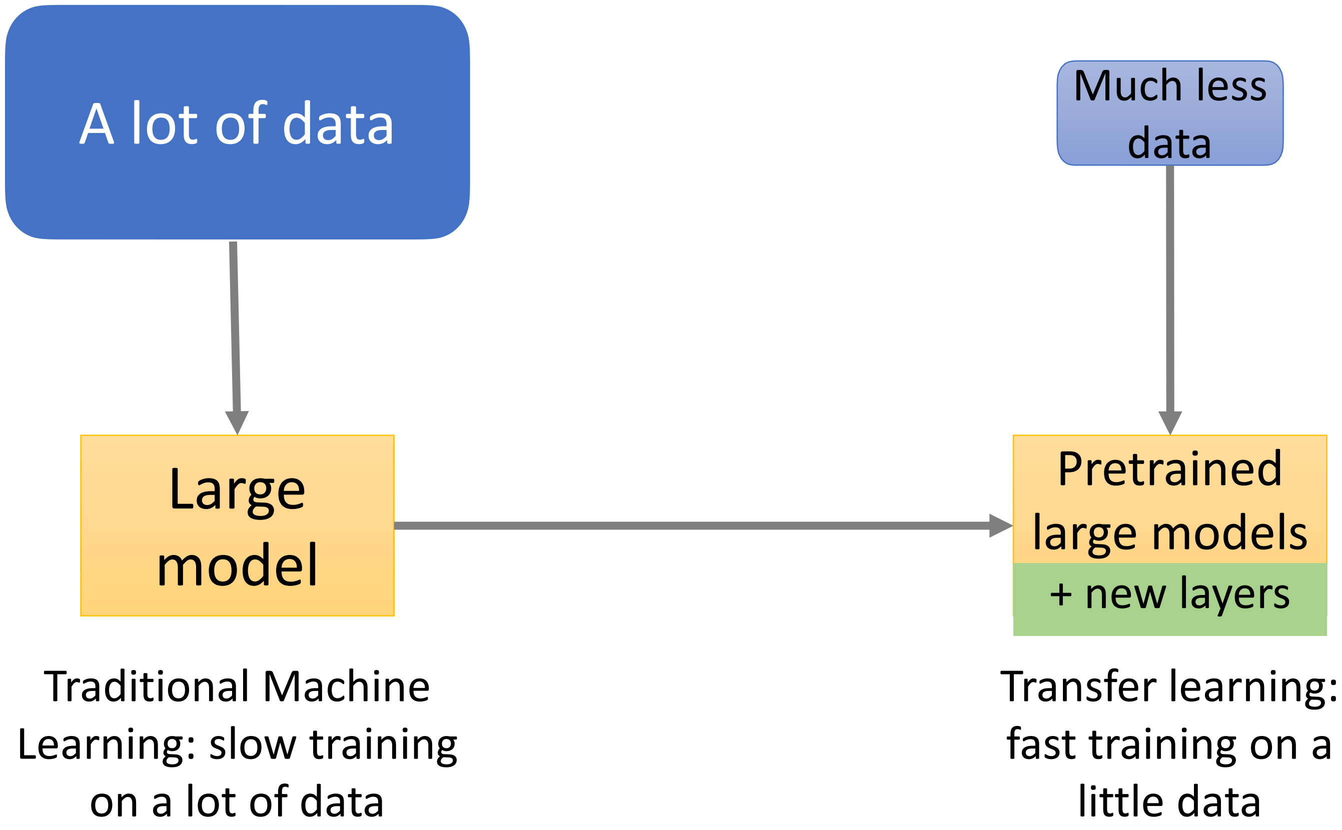


GPT

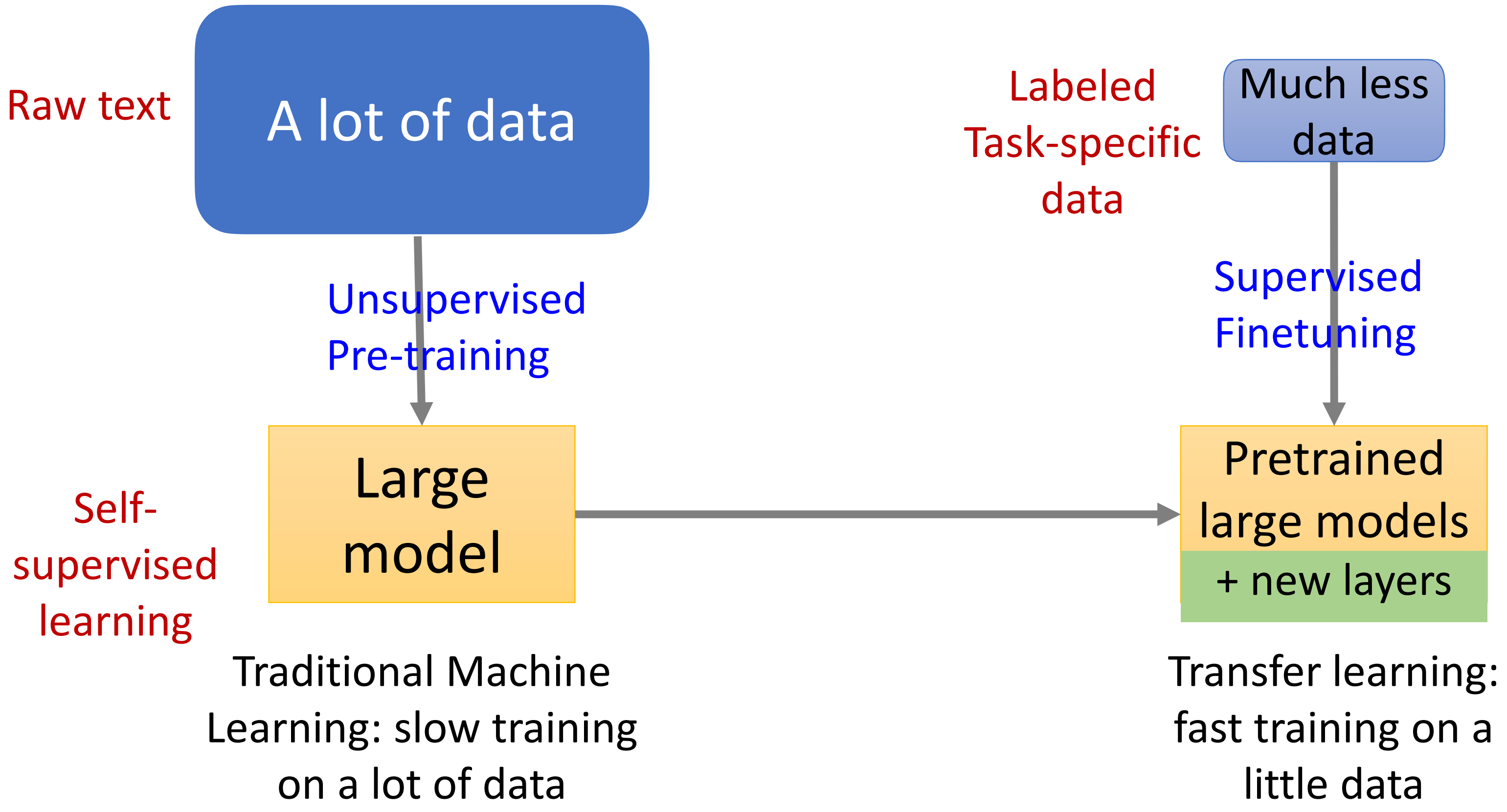
Decoder



Transfer Learning



Transfer Learning in NLP



How are LLMs applied in various tasks/domains?

- Previously
  - By adding task-specific layers on top of LLMs and fine-tuning them on labeled data of such tasks
  - Examples: Text classification, language translation, question-answering

- More recently
  - By prompt engineering/tuning, **without** changing LLM params
  - Design a prompt that elicits the desired output from the LLM
  - Example (English->French translation):

Translate the following English sentence [sentence input] *to French:* [model output]