Introduction to Deep Learning

22. Encoder-Decoder, Seq2seq

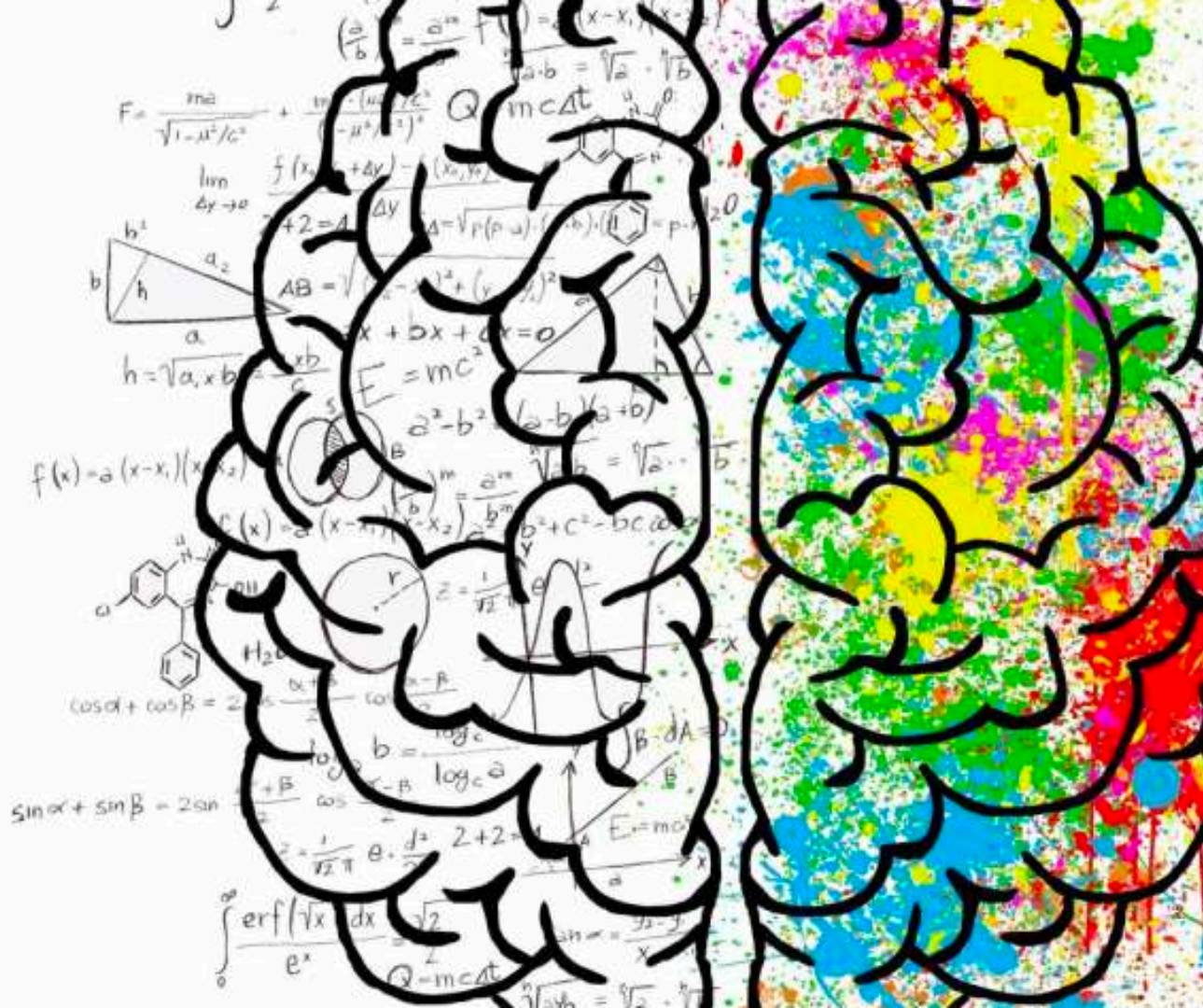
STAT 157, Spring 2019, UC Berkeley

Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157

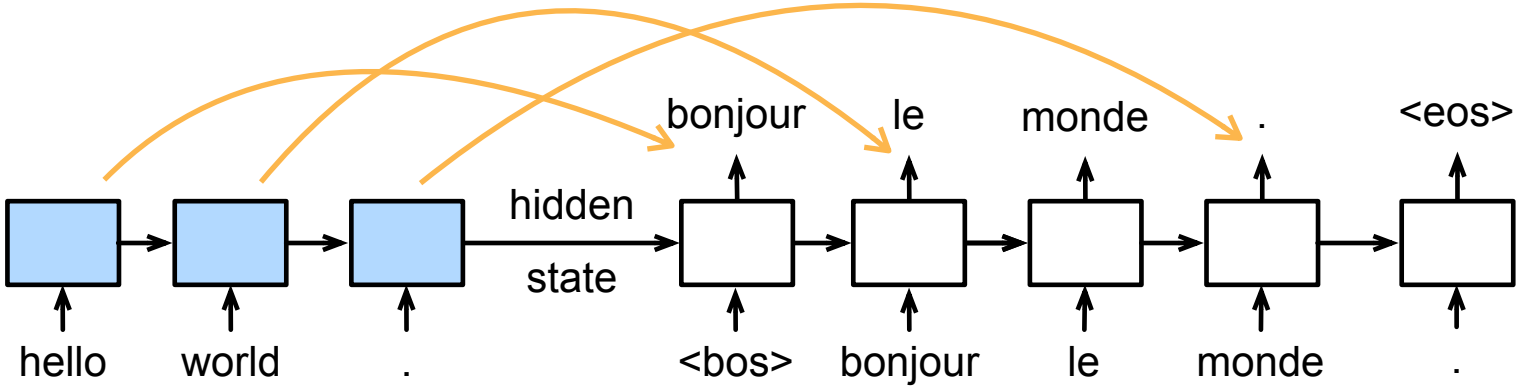
aws

Attention



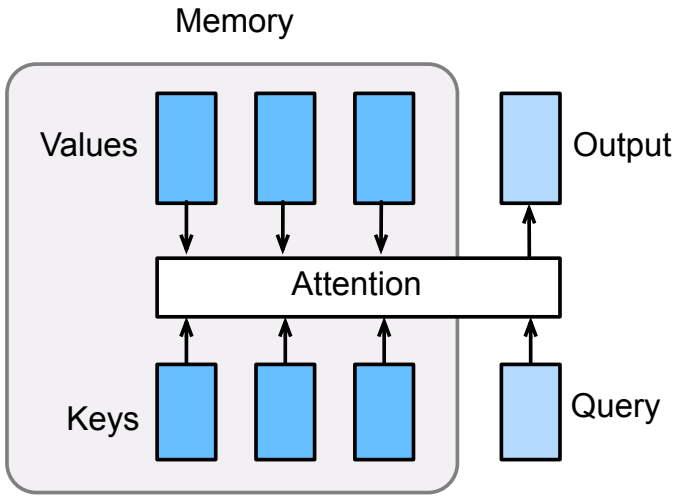
Motivation

- Each generated token might be related to different source tokens



Attention Layer

- An attention layer explicitly select related information
  - Its memory consists of key-value pairs
  - The output will be close to the values whose keys are similar to the query



Attention Layer

- Assume query $\mathbf{q} \in \mathbb{R}^{d_q}$ and the memory $(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_n, \mathbf{v}_n)$ with $\mathbf{k}_i \in \mathbb{R}^{d_k}$ $\mathbf{v}_i \in \mathbb{R}^{d_v}$
- Compute n scores $a_1, \ldots, a_n$ with $a_i = \alpha(\mathbf{q}, \mathbf{k}_i)$
- Use softmax to obtain the attention weights

$$b_1, \ldots, b_n = \text{softmax}(a_1, \ldots, a_n)$$

- The output is a weighted sum of values

$$\mathbf{o} = \sum_{i=1}^{n} b_i \mathbf{v}_i$$

Vary α to get different attention layers

Dot Product Attention

- Assume the query has the same length as values $\mathbf{q}, \mathbf{k}_i \in \mathbb{R}^d$

$$\alpha(\mathbf{q}, \mathbf{k}) = \langle \mathbf{q}, \mathbf{k} \rangle / \sqrt{d}$$

- Vectorized version
  - *m* queries $\mathbf{Q} \in \mathbb{R}^{m \times d}$ and *n* keys $\mathbf{K} \in \mathbb{R}^{n \times d}$

$$\alpha(\mathbf{Q}, \mathbf{K}) = \mathbf{Q}\mathbf{K}^T / \sqrt{d}$$

Multilayer Perception Attention

- Learnable parameters $\mathbf{W}_k \in \mathbb{R}^{h \times d_k}, \mathbf{W}_q \in \mathbb{R}^{h \times d_q}$, and $\mathbf{v} \in \mathbb{R}^h$

$$\alpha(\mathbf{k}, \mathbf{q}) = \mathbf{v}^T \tanh(\mathbf{W}_k \mathbf{k} + \mathbf{W}_q \mathbf{q})$$

- Equals to concatenate the key and query, and then feed into a single hidden-layer perception with hidden size h and output size 1

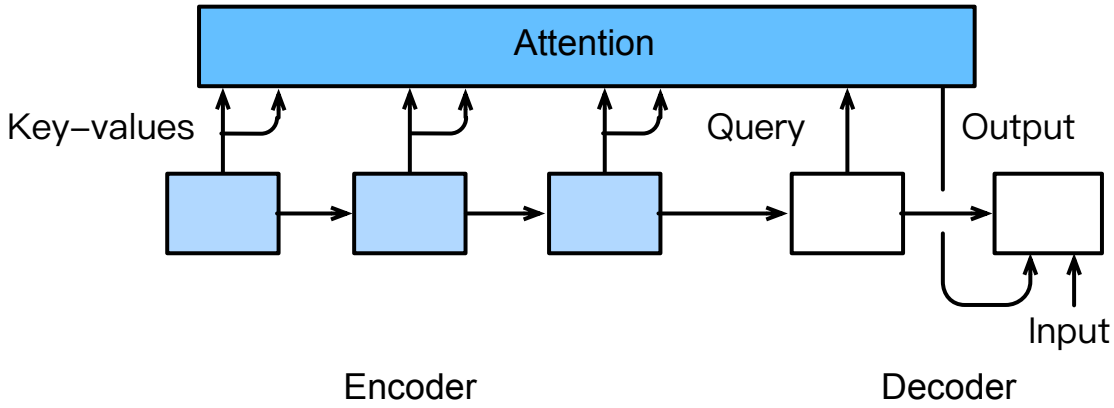
Code…

courses.d2l.ai/berkeley-stat-157/index.html

Seq2seq with Attention

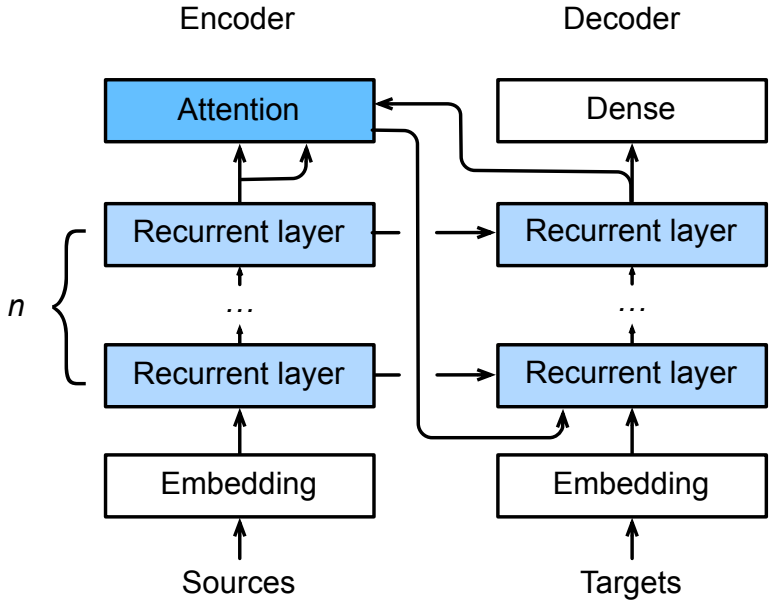
Model Architecture

- Add an additional attention layer to use encoder's outputs as memory
- The attention output is used as the decoder's input



Encoder/Decoder Details

- The output of the last recurrent layer in the encoder is used
- The attention output is then concatenated with the embedding output to feed into the first recurrent layer in the decoder



Code…

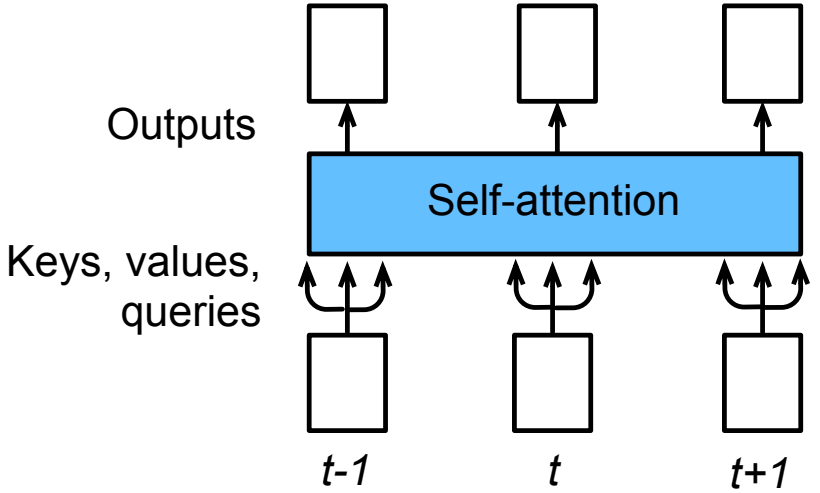
courses.d2l.ai/berkeley-stat-157/index.html

Transformer

courses.d2l.ai/berkeley-stat-157/index.html

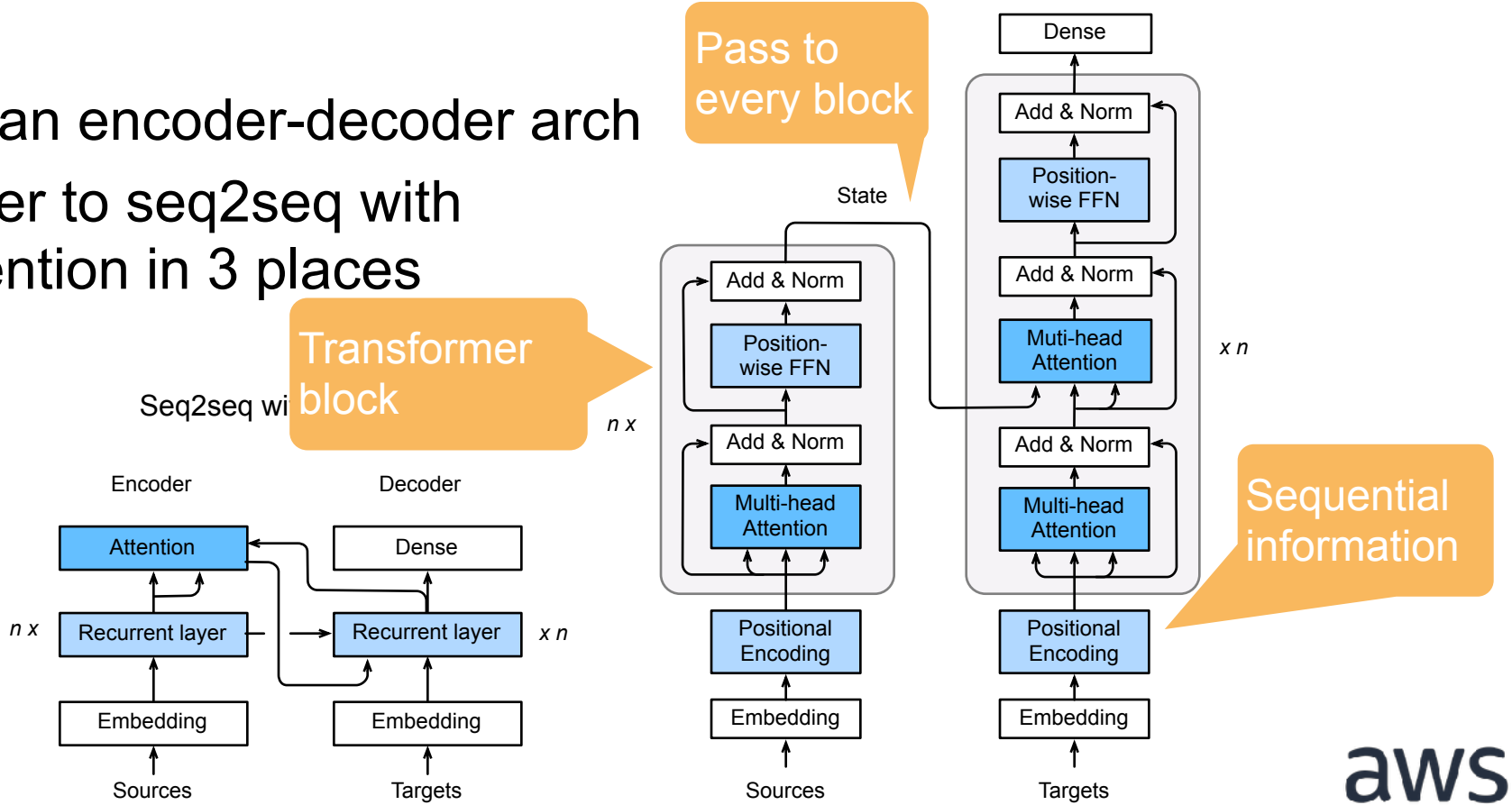
Self-attention

- To generate *n* outputs with *n* inputs, we can copy each input into a key, a value and a query
- No sequential information is preserved
- Run in parallel



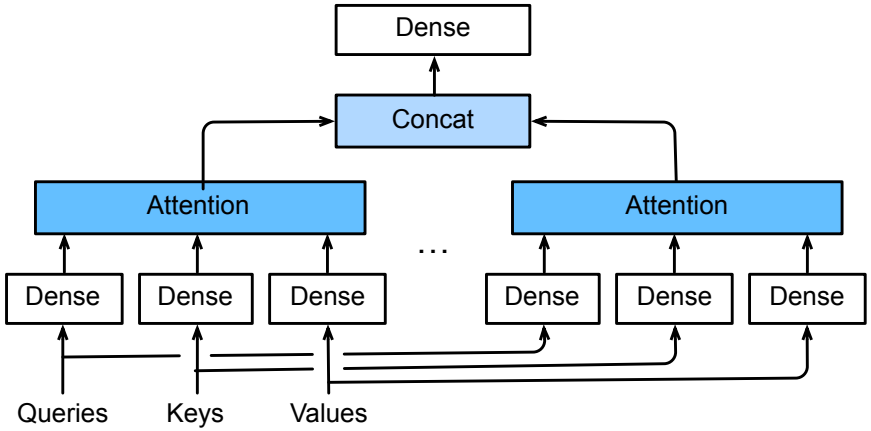
Transformer Architecture

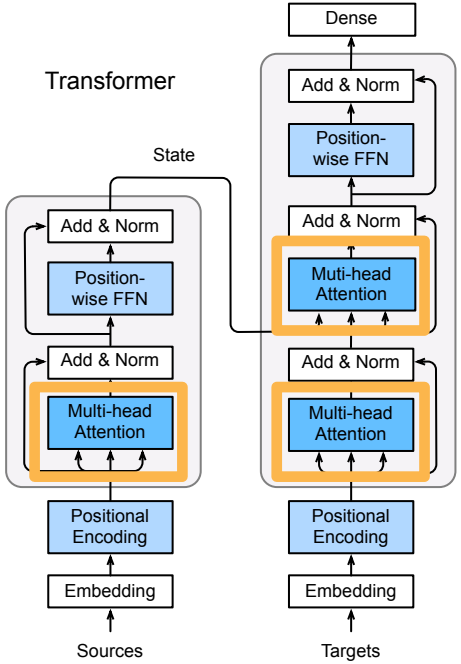
- It's an encoder-decoder arch
- Differ to seq2seq with attention in 3 places



courses.d2l.ai/berkeley-stat-157/index.html

Multi-head Attention





$$\mathbf{W}_q^{(i)} \in \mathbb{R}^{p_q \times d_q}, \mathbf{W}_k^{(i)} \in \mathbb{R}^{p_k \times d_k}, \text{ and } \mathbf{W}_v^{(i)} \in \mathbb{R}^{p_v \times d_v}$$

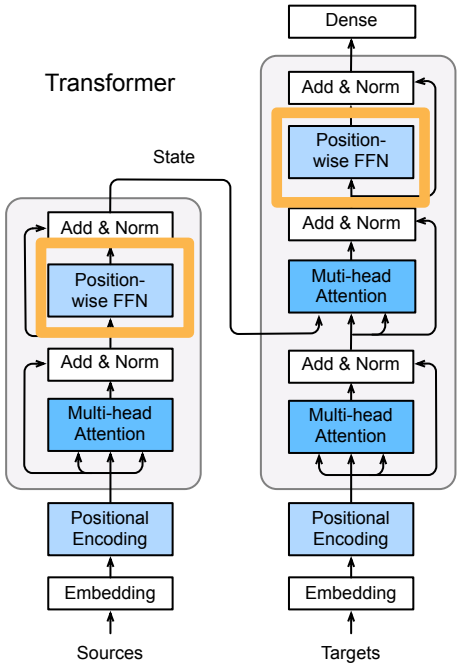
$$\mathbf{o}^{(i)} = \text{attention}(\mathbf{W}_q^{(i)}\mathbf{q}, \mathbf{W}_k^{(i)}\mathbf{k}, \mathbf{W}_v^{(i)}\mathbf{v})$$

for $i = 1, \ldots, h$

$$\mathbf{o} = \mathbf{W}_o \begin{bmatrix} \mathbf{o}^{(1)} \\ \vdots \\ \mathbf{o}^{(h)} \end{bmatrix}$$

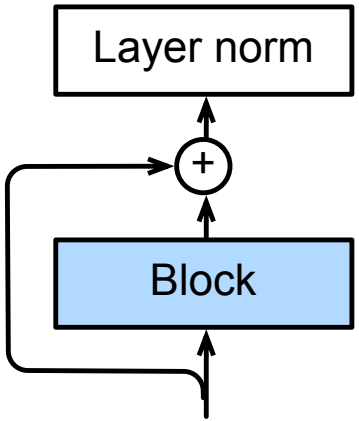
Position-wise Feed-Forward Networks

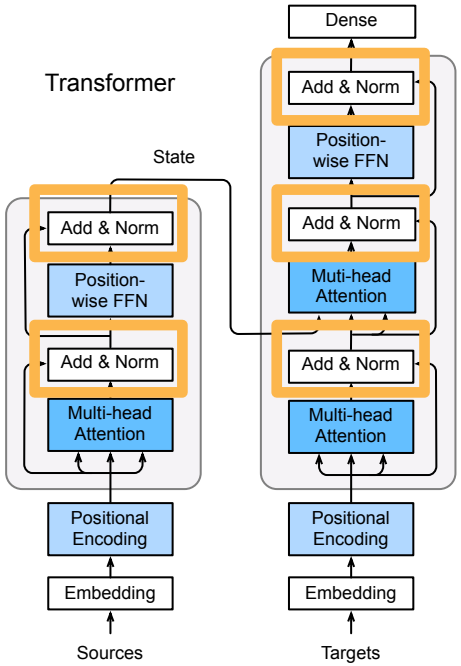
- Reshape input (batch, seq len, fea size) into (batch * seq len, fea size)
- Apply a two layer MLP
- Reshape back into 3-D
- Equals to apply two (1,1) conv layers



Add and Norm

- Layer norm is similar to batch norm
- But the mean and variances are calculated along the last dimension
- `X.mean(axis=-1)` instead of the first batch dimension in batch norm `X.mean(axis=0)`



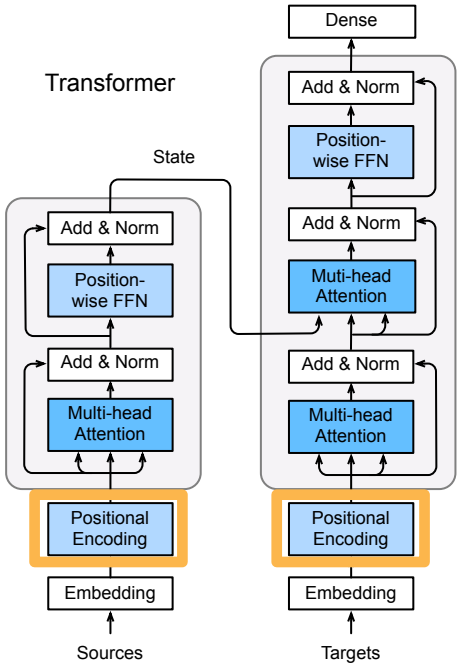


Positional Encoding

- Assume embedding output $X \in \mathbb{R}^{l \times d}$ with shape (seq len, embed dim)
- Create $P \in \mathbb{R}^{l \times d}$ with

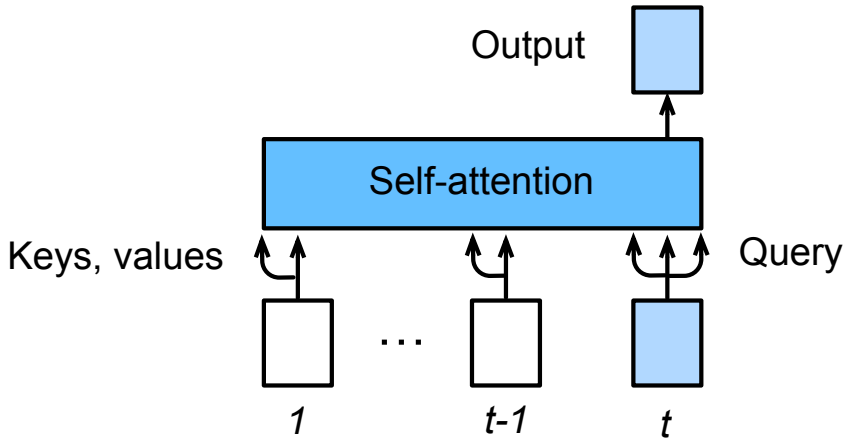
$$P_{i,2j} = \sin(i/10000^{2j/d})$$
$$P_{i,2j+1} = \cos(i/10000^{2jd})$$

- Output $X + P$



Predicting

- Predict at time *t*:
  - Inputs of previous times as keys and values
  - Input at time *t* as query, as well as key and value, to predict output



Code…

courses.d2l.ai/berkeley-stat-157/index.html

BERT

courses.d2l.ai/berkeley-stat-157/index.html

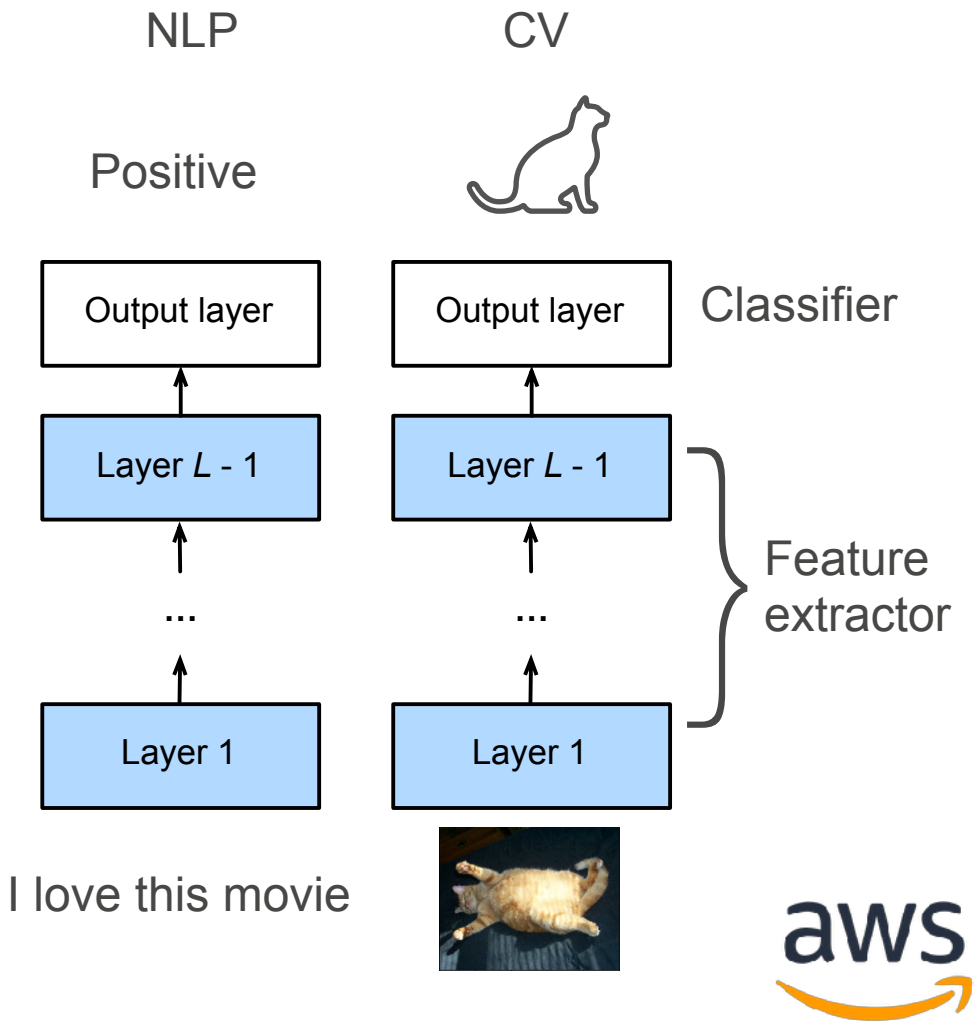
aws

Transfer Learning in NLP

- Use pre-trained models to extract word/sentence features for the new task
  - E.g. word2vec or language model
- Often don’t update the pre-trained models
- Need to construct a new model to capture the information needed for the new task
  - Word2vec ignores sequential information, language model only looks in a single direction

Motivation of BERT

- A fine-tuning based approach for NLP
- The pre-trained model capture sufficient data information
- Only need to add a simple output layer for a new task

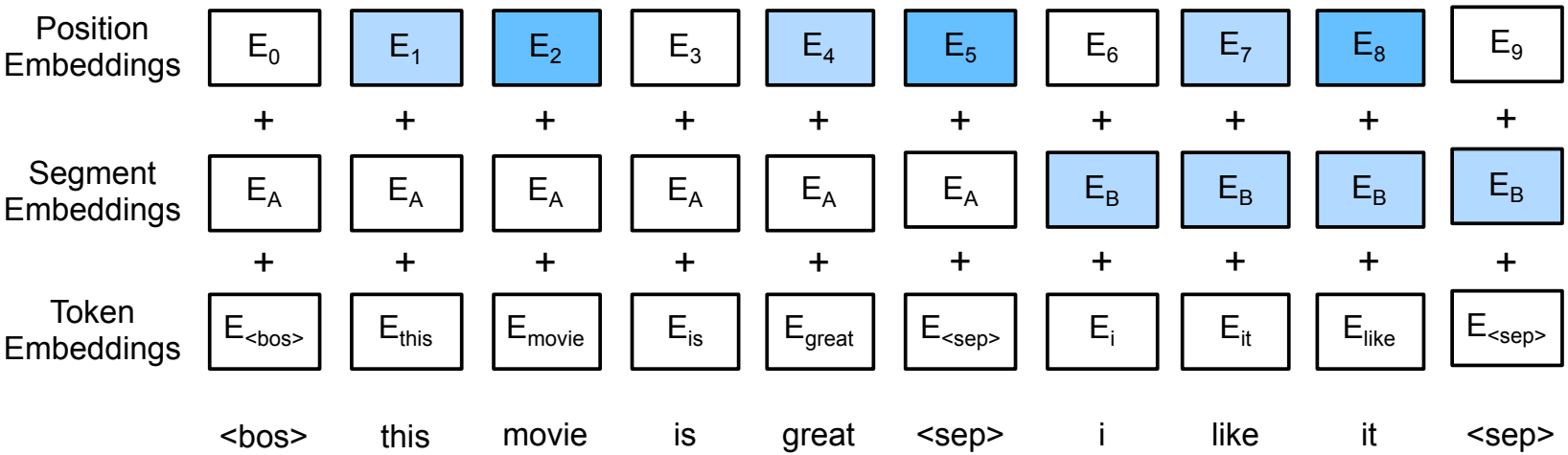


BERT Architecture

- A (big) Transformer encoder (without the decoder)
- Two variants:
  - Base: #blocks = 12, hidden size = 768, #heads = 12, #parameters = 110M
  - Large: #blocks = 24, hidden size = 1024, #heads = 16, #parameter = 340M
- Train on large-scale corpus (books and wikipedia) with > 3B words

Modification of inputs

- Each example is a pair of sentences
- Add an additional segment embedding



Pre-training Task 1: Masked Language Model

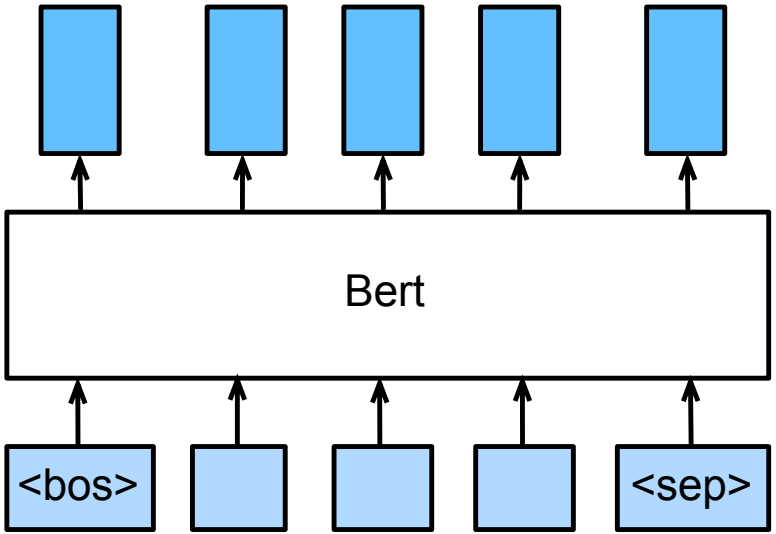
- Randomly mask (e.g. 15%) tokens in each sentence, predict these masked tokens
  - Transformer is bidirectional, which breaks the unidirectional limit of standard LM
- No mask token (<mask>) in fine turning tasks
  - 80% of the time, replace selected tokens with <mask>
  - 10% of the time, replace with randomly picked tokens
  - 10% of the time, keep the original tokens

Pre-training Task 2: Next Sentence Prediction

- 50% of time, choose a sequential sentence pair
  - <bos> this movie is great <sep> i like it <sep>
- 50% of time, choose a random sentence pair
  - <bos> this movie is great <sep> hello world <sep>
- Feed the Transformer output of <bos> into a dense layer to predict if it is a sequential pair

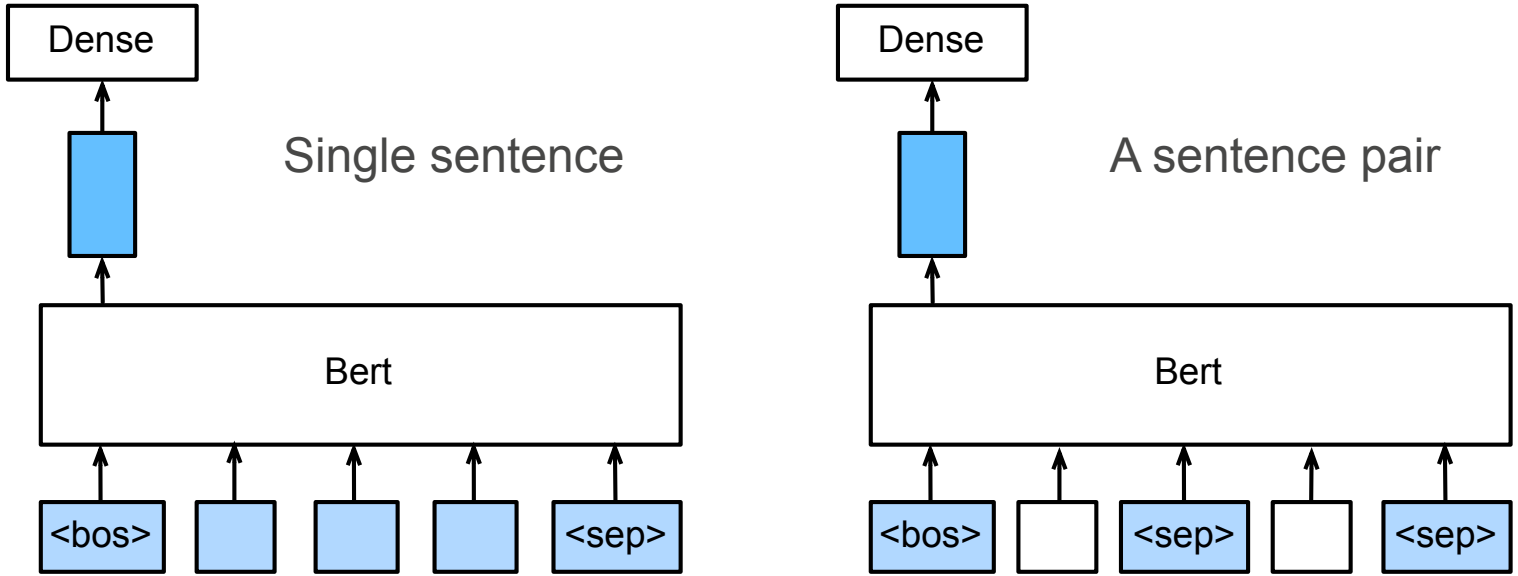
Bert for Fine Tuning

- Bert returns a feature vector for each token that captures the context information
- Different fine-tuning tasks use a different set of vectors



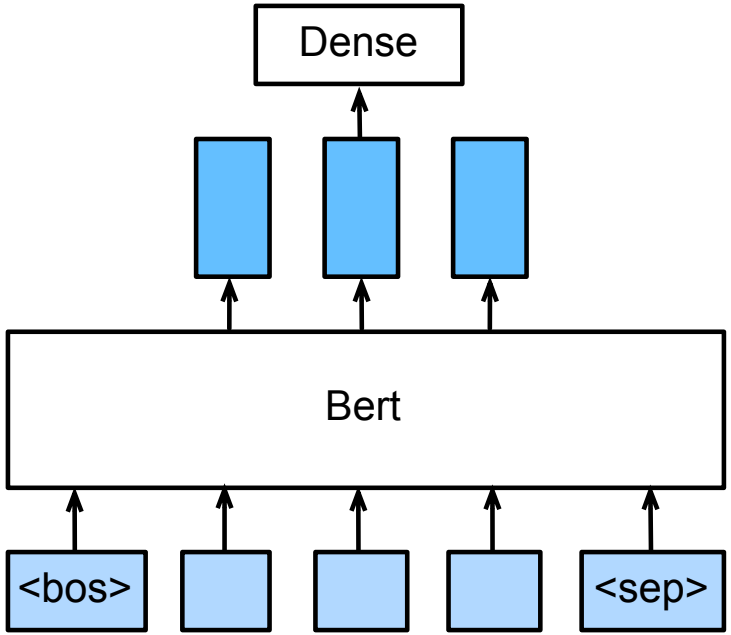
Sentences Classification

- Feed the <bos> token vector into a dense output layer



Named Entity Recognition

- Identify if a token is a named entity such as person, org, and locations…
- Feed each non-special token vector into a dense output layer

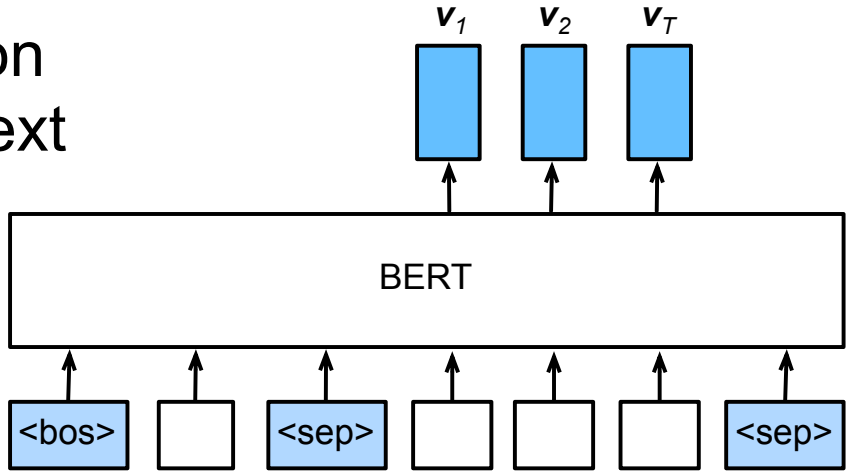


Question Answering

- Given a question and a description text, find the answer, which is a text segment in the description
- Given $\mathbf{p}_i$ the *i*-th token in the desperation, learn $\mathbf{s}$ so that

$$p_1, \ldots, p_T = \text{softmax}(\langle \mathbf{s}, \mathbf{v}_1 \rangle, \ldots, \langle \mathbf{s}, \mathbf{v}_T \rangle)$$

p_i is the probability *i*-th token is the segment start. Same for the end



Check GluonNLP for code…

courses.d2l.ai/berkeley-stat-157/index.html