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

Lecture 8/6: Attention

Next: Multi Attention

- Transformers Architect
- use Attention instead of Convolutions, RNC

Attention

"Attention is ALL you need" (2017)

courses.d2l.ai/berkeley-stat-157/inde

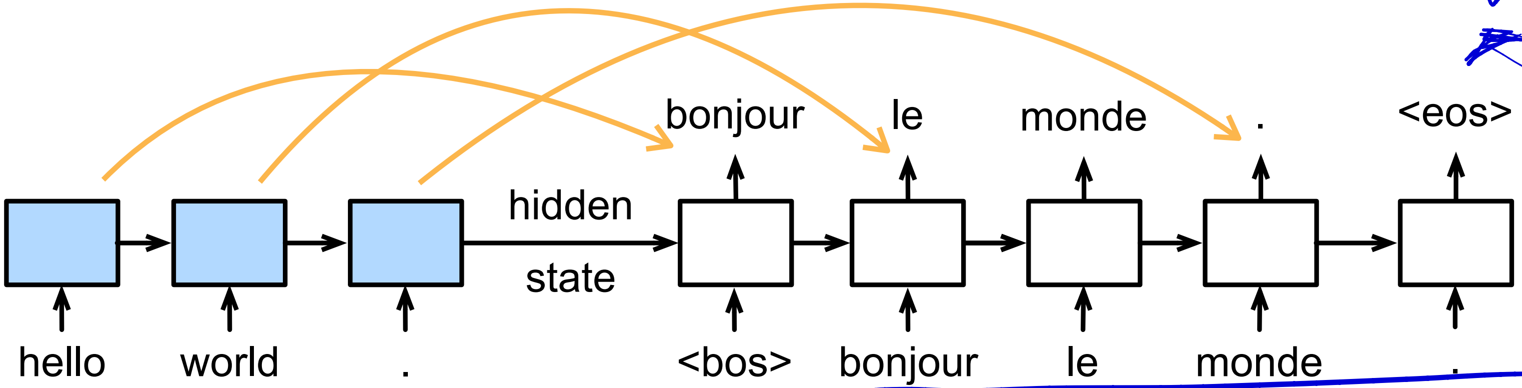
Motivation

Attention = a new layer in NN

from RNN parts

- Each generated token might be related to different source tokens

k? q? v?



- setup(q, k, v, α, h ...)
- math formulas
- code (d2L)

- how these play a role inside RNN (NN for sequences) (2017)

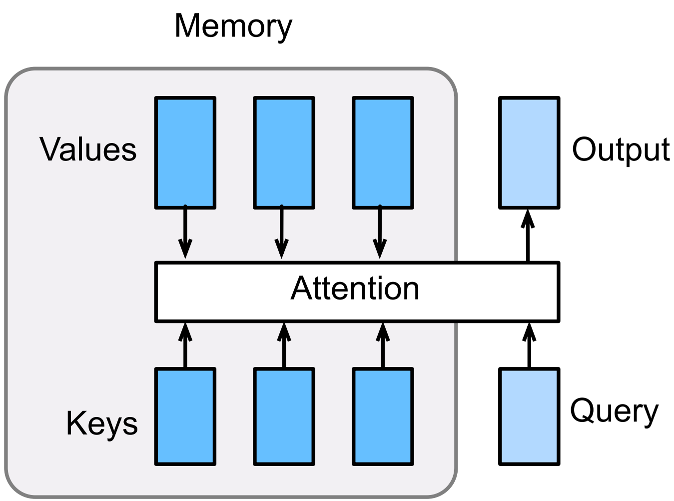
aws

Attention Layer

; q = queries ; k = keys ; v = values $\alpha = \mathrm{sim}(k, q)$ "kernel"

d = dim ; H = hidden ; h = # attention heads (multi head attention)

- 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



current step, query

$$\mathrm{Attention}(\boxed{q}, D^{\{k,v\}}) = \sum_{i \in D} \alpha(q, \boxed{k_i})\, v_i \overset{\text{"Aggregate"}}{=\!=} \text{weighted\_avg}(v_i) \text{ with weights } \alpha(q, k_i)$$

prev steps

Flashback: $\alpha(u,v)$ = similarity kernel(x_u, x_v)

weighted avg by similarity : HW5

$$\text{pred}(\underset{\text{test}}{z}, \underset{\text{label}}{Y}) = \frac{\sum_{i=1}^{N} \alpha(z, x_i) \cdot Y_i}{\sum \alpha(z, x_i)} = \text{weighted avg}(Y_i) \text{ by weights } \text{sim}(z, x_i)$$

$\alpha(z, x_i)$ = high $\Rightarrow$ x_i is important for z

$\alpha(z, x_j)$ = low $\Rightarrow$ x_j not important for z

pred(z) should pay ATTENTION to x_i, but not much to x_j

RNN Attention propreties

- $\sum_i \alpha(q, k_i) = 1$ (distribution) norm $\alpha(q, k_i) = \frac{\alpha(q, k_i)}{\sum_j \alpha(q, k_j)}$
- $\alpha(q, k_i) \geq 0$ $\Rightarrow$ use softmax instead of norm

$$\underbrace{\alpha(q, k_i)}_{\text{output}} = \frac{\exp(\text{linear score}(q, k_i))}{\sum_j \exp(\text{linear-score}(q, k_j))}$$

Attention Layer

queries Keys

- Assume query $\boxed{\mathbf{q}} \in \mathbb{R}^{d_q}$ and the memory $(\boxed{\mathbf{k}_1}, \boxed{\mathbf{v}_1}), \ldots, (\boxed{\mathbf{k}_n}, \boxed{\mathbf{v}_n})$
  with $\mathbf{k}_i \in \mathbb{R}^{d_k}$ $\mathbf{v}_i \in \mathbb{R}^{d_v}$

values

- Compute n scores $a_1, \ldots, a_n$ with $a_i = \alpha(\mathbf{q}, \mathbf{k}_i)$
- Use softmax to obtain the attention weights

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

- 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

aws

Dot Product Attention

linear similarity = "linear kernel"

score $(q, k) = \langle q \cdot k \rangle$

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

SCALE DOT PRODUCT

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

scale by length

comp.

irrelevant

(normalization, scaling)

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

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

matters

Euclidian? $\alpha(q_i, k) = -\frac{1}{2}\|q-k\|^2 = \boxed{q_i^T k} - \frac{1}{2}\|k_i\|^2 - \frac{1}{2}\|q\|^2$

Softmax (Euclid) $= \underline{\exp(-\frac{1}{2}\|q-k\|^2)}$

normalized

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

- Attention "Pooling" as similarity computation
  - $\alpha(q,k) = \langle q \cdot k \rangle / \sqrt{d}$ linear → important for parallel computation
  - $\alpha(q,k) = \exp\left(-\frac{1}{2}\|q-k\|^2\right)$ Gaussian
  - $\alpha(q,k) = \exp\left(-\frac{1}{2}\frac{\|q-k\|^2}{2\sigma^2}\right)$ Gaussian scaled for width / adapted
  - $\alpha(q,k) = \begin{cases} 1 & \text{if } \|q-k\| \leq \text{threshold} \\ 0 & \text{if not} \end{cases}$ "Boxcar"
  - $\alpha(q,k) = \max\{0, 1-\|q-k\|\}$
- Attn score $= \sum_i \boxed{V_i} \cdot \underbrace{\alpha(q,k)}_{\text{normalized}}$ → values

---

- Masked Softmax: sequences of different lengths.

$$\underset{\text{query}}{q^T} \cdot \underset{\text{key}}{k} \longrightarrow q^T \cdot \underbrace{\boxed{M}}_{\text{masked matrix}} \cdot k$$

masked matrix (alligns dimensions q, k)

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 $\mathbf{v} \in \mathbb{R}^h$

values

queries

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

keys

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

Bahdanau : important

$$\text{RNN enc-dec} \xrightarrow{\text{step}} \text{Transformer}$$

$\boxed{\text{Bahdanau}}$

t = current step

$$c_t = \sum_{s=1}^{T} \alpha_{t,s} \cdot h_s$$

$\alpha_{t,s} = \text{sim}(t, k_s)$

$k_s = v_s = h_s$

s = index prev hidden rec step

Code...

= $\underline{\text{key } k_s}$ = val v_s = h_s = Encoder hidden state at step t

= "how useful is $\boxed{enc(x_s)}$"

s token in input sequence x

dec hidden state at step t

= query q_t = step t hidden state of decoder

$$\text{score}(s_t, h_s) = V_t \cdot \tanh(W_s \cdot h_s + W_t s_t)$$

aws

similarity

$$\alpha(t, s) = \text{softmax}(\text{score}(s_t, h_s))$$

At decoder step/state t: What encoder steps h_s are most relevant? → from past sequence (x)

⇒ give those steps high weights $\alpha(t, s)$

Seq2seq with Attention

Multihead Attention

- Multiple Attn model = scoring at same time
  $H_1\ H_2\ H_3 \ldots H_h$ — h = # heads (attn score)

- Computation simplified, parallel ⇒ larger scale with GPU

- q, k, v = linear function of input ⇒ much faster

- replace REC, CONV layers with particular attn
  - no REC ⇒ diff sequence mechanic? — add positioning encoding
    step 1, 2, ... in sequence
  - more general / can be repurposed easily — not common in RNN but possible

- score = $\boxed{\dfrac{QK^T}{\sqrt{d}} \cdot V}$ (values) → dot prod easy to compute ; $\alpha = \text{softmax}(\text{score})$

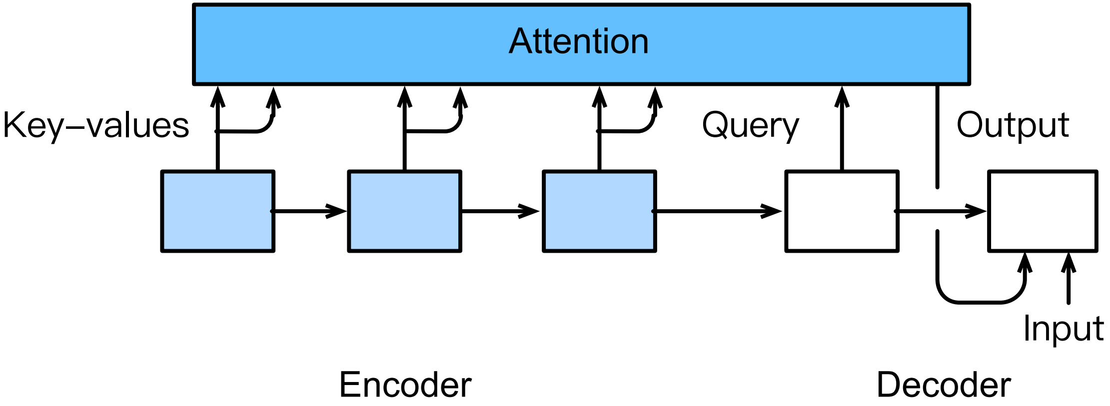
- current decoder demand ← $\boxed{Q = X \cdot W_q}$ (queries); $\boxed{K = X \cdot W_k}$ (keys) → prev enc() of x to match with q; $V = X \cdot W_v$ (values)

learn W_q, W_k, W_v, W_o (output)

$$\text{Multihead}(Q, K, V) = \text{Concat}(H_1, H_2 \ldots H_h) \cdot W_o$$

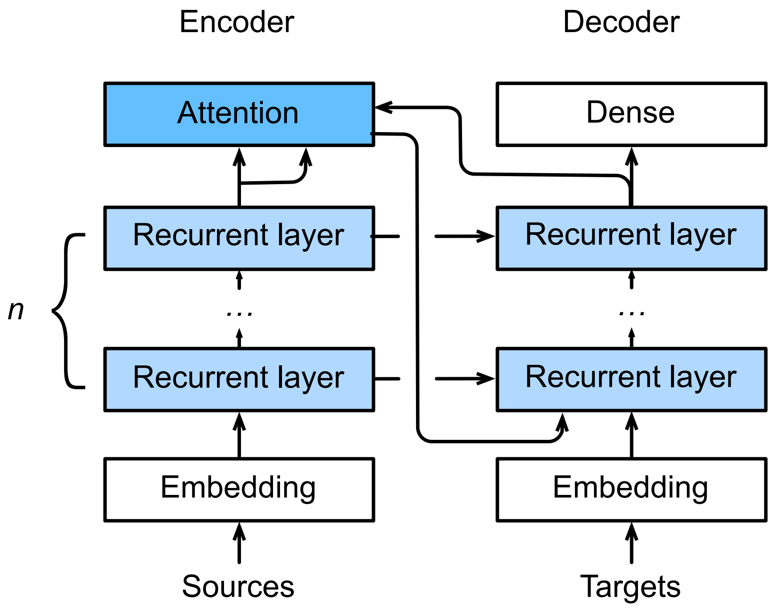
Model Architecture

- 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



next time: transformers

due Fri 8/1 ← HW6 demo part 2.

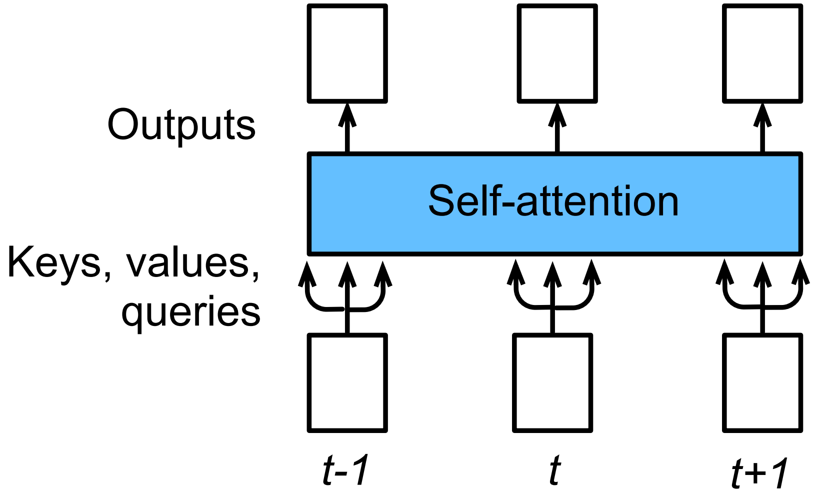
Code...

projects due 8/10, keep updating chat with TA.

Transformer

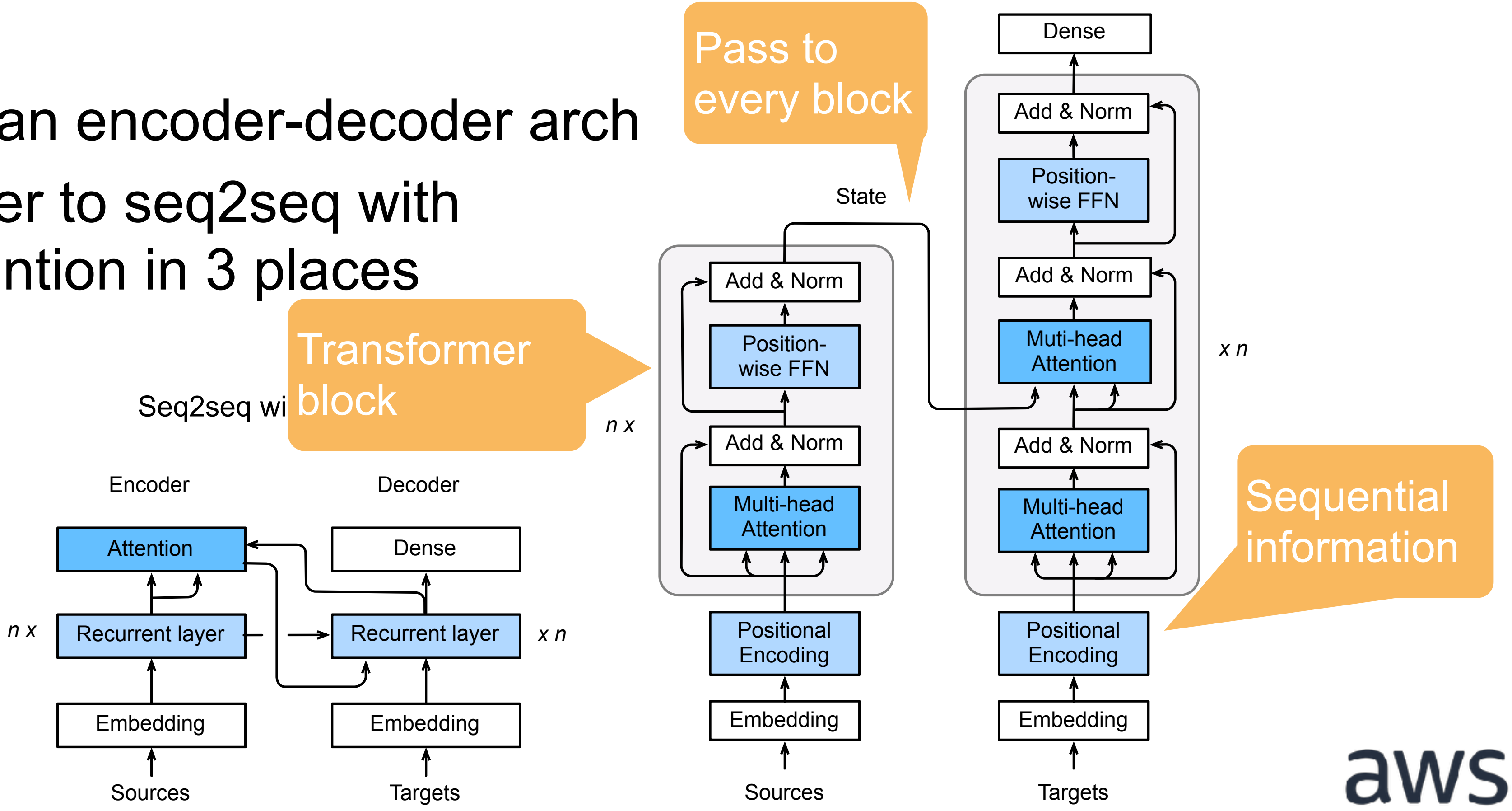
Self-attention

- 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



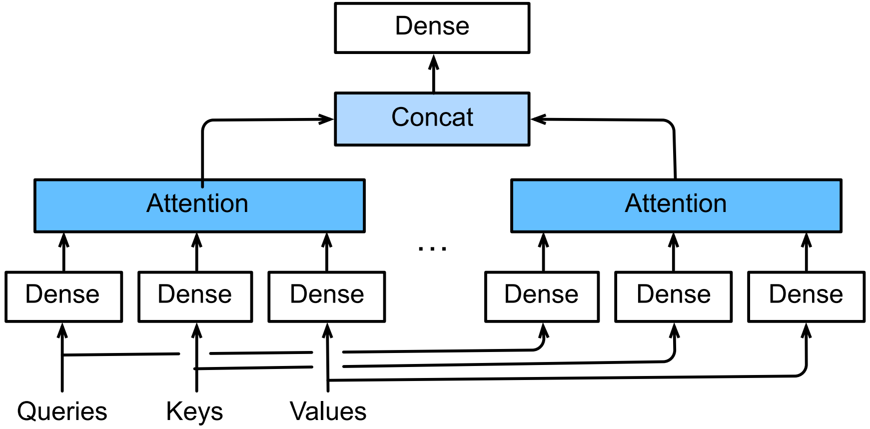
Transformer Architecture

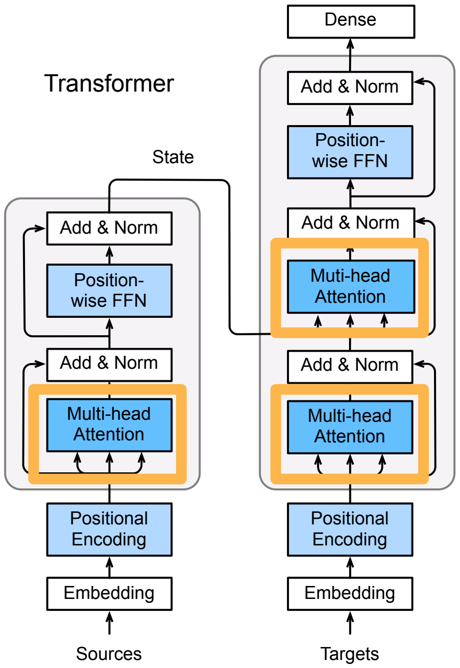
- It's an encoder-decoder arch
- Differ to seq2seq with attention in 3 places



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

Multi-head Attention





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

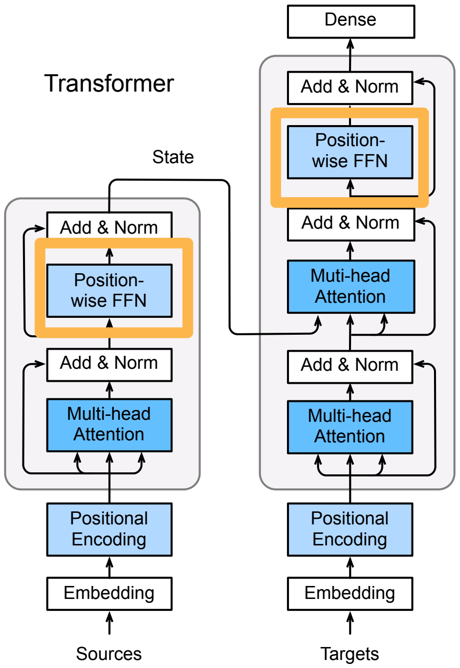
$$\mathbf{o}^{(i)} = \text{attention}(\mathbf{W}_q^{(i)}\mathbf{q}, \mathbf{W}_k^{(i)}\mathbf{k}, \mathbf{W}_v^{(i)}\mathbf{v})$$

for $i = 1, \ldots, h$

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

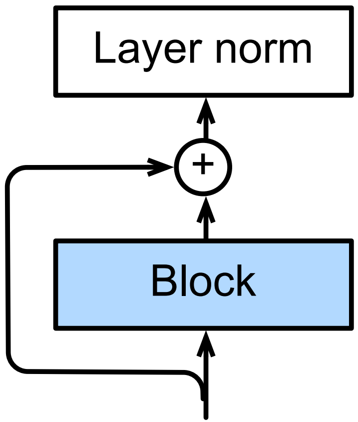
Position-wise Feed-Forward Networks

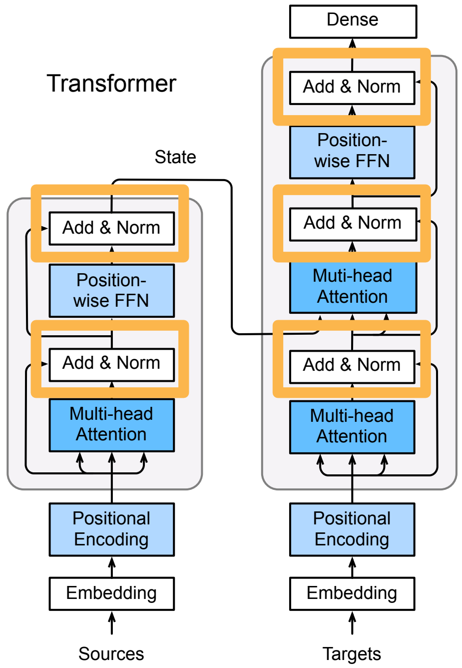
- Reshape input (batch, seq len, fea size) into (batch * seq len, fea size)
- Apply a two layer MLP
- Reshape back into 3-D
- Equals to apply two (1,1) conv layers



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)`



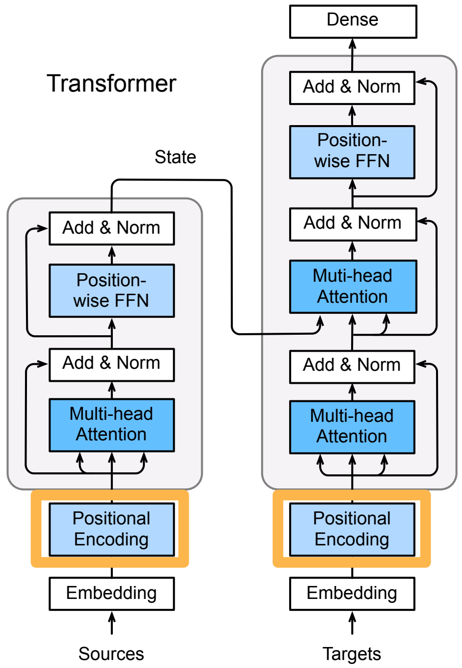


Positional Encoding

- 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

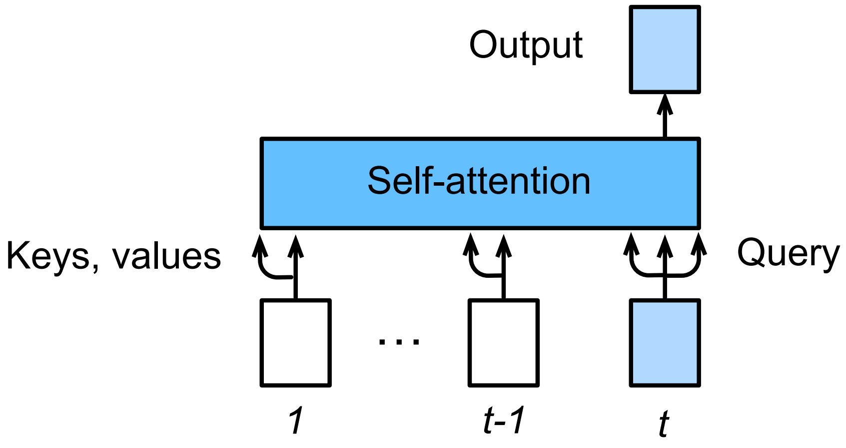
$$P_{i,2j} = \sin(i/10000^{2j/d})$$
$$P_{i,2j+1} = \cos(i/10000^{2jd})$$

- 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…

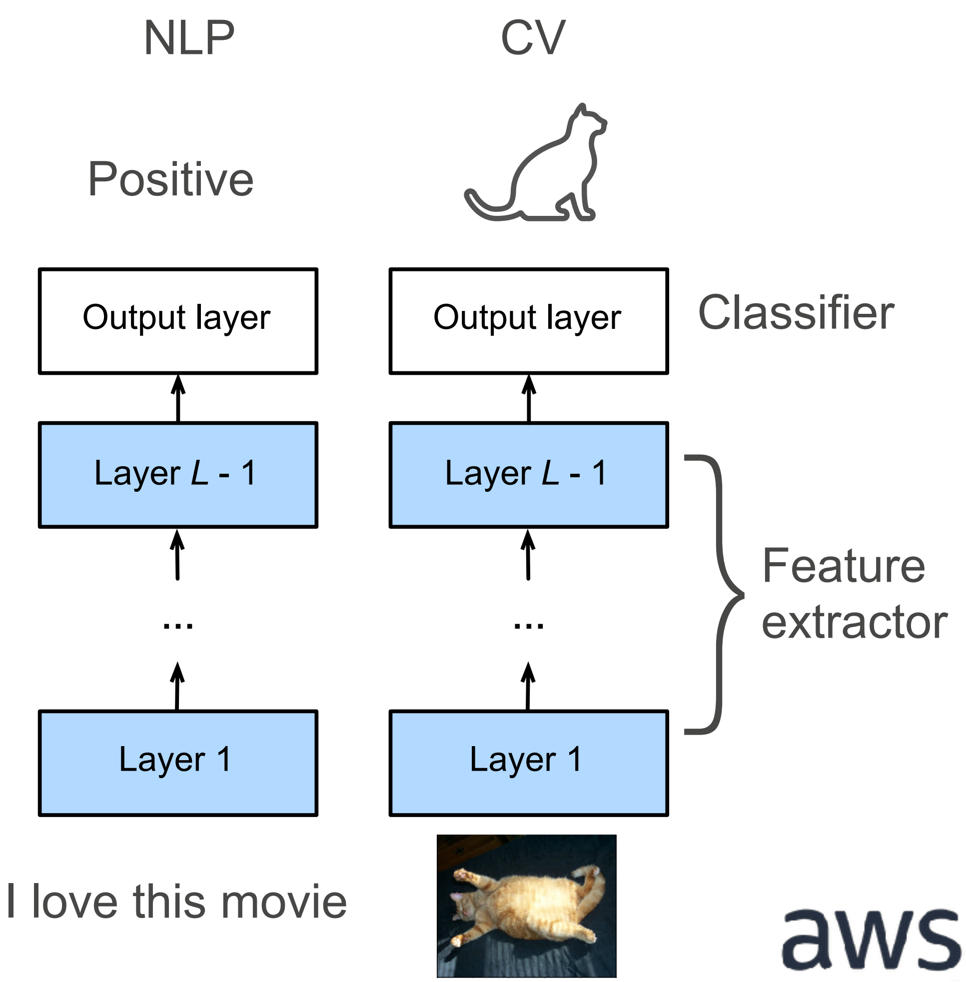
BERT

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

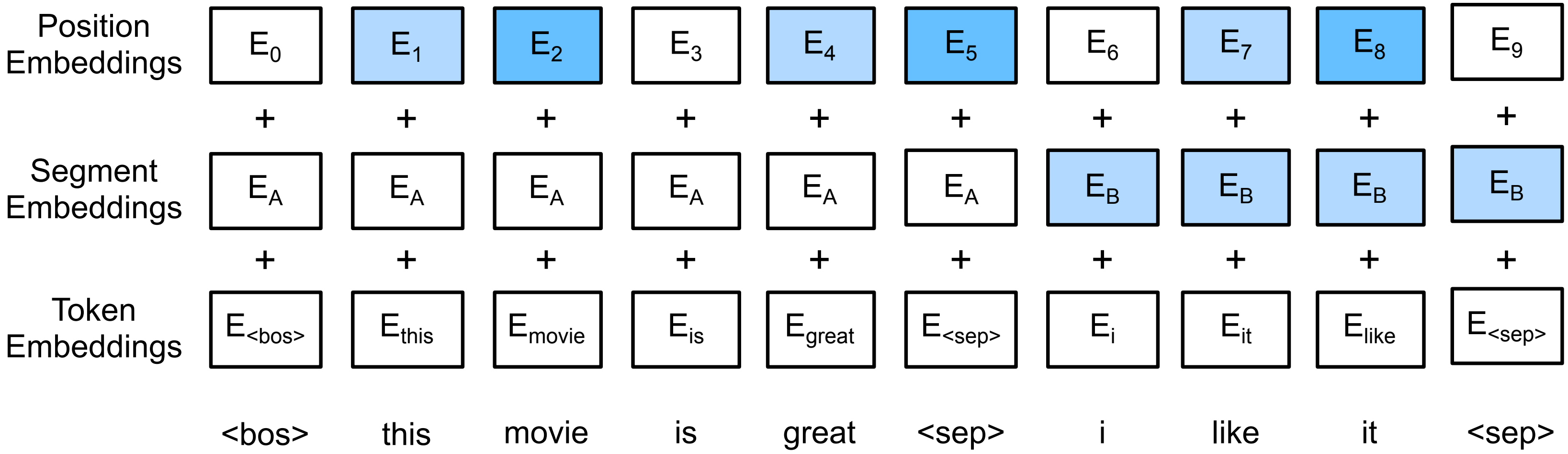


BERT Architecture

- 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

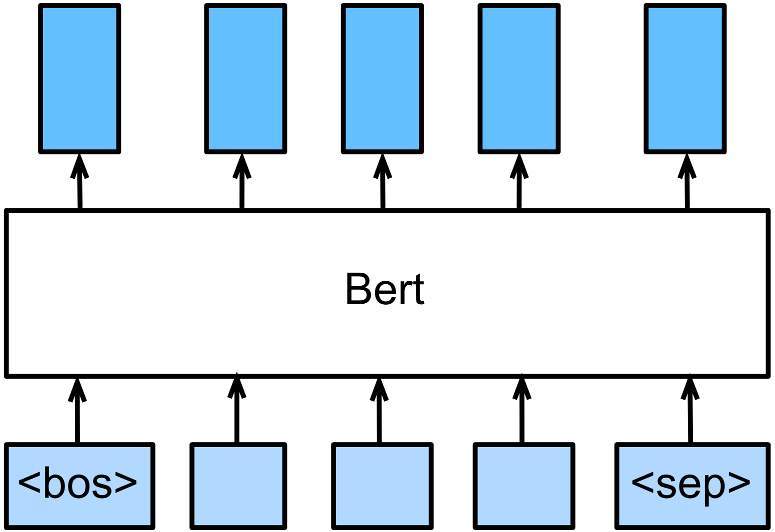
- Randomly mask (e.g. 15%) tokens in each sentence, predict these masked tokens
  - Transformer is bidirectional, which breaks the unidirectional limit of standard LM
- No mask token (<mask>) in fine turning tasks
  - 80% of the time, replace selected tokens with <mask>
  - 10% of the time, replace with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

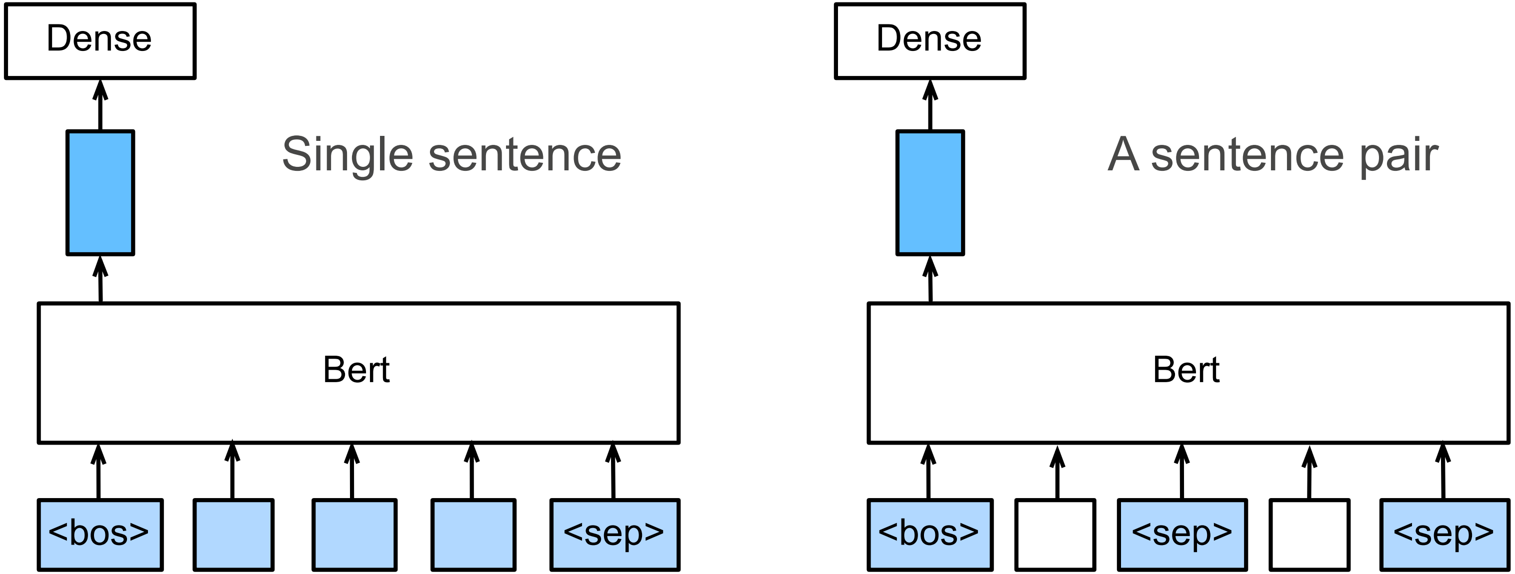
Bert for Fine Tuning

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



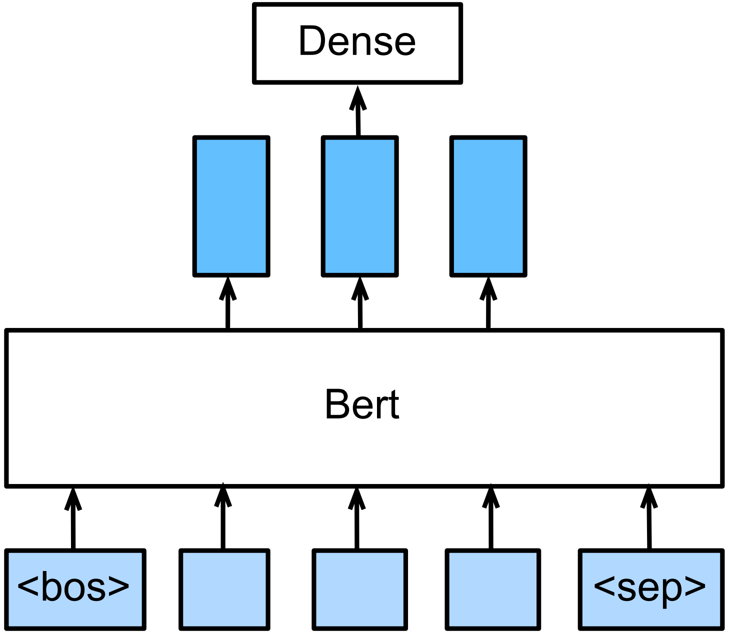
Sentences Classification

- 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

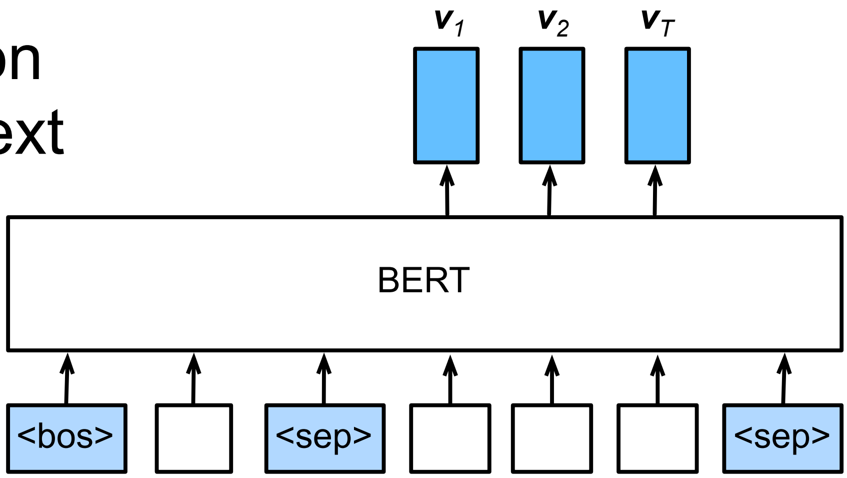


Question Answering

- 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…