Stanford University

CS 329P : Practical Machine Learning (2021 Fall)

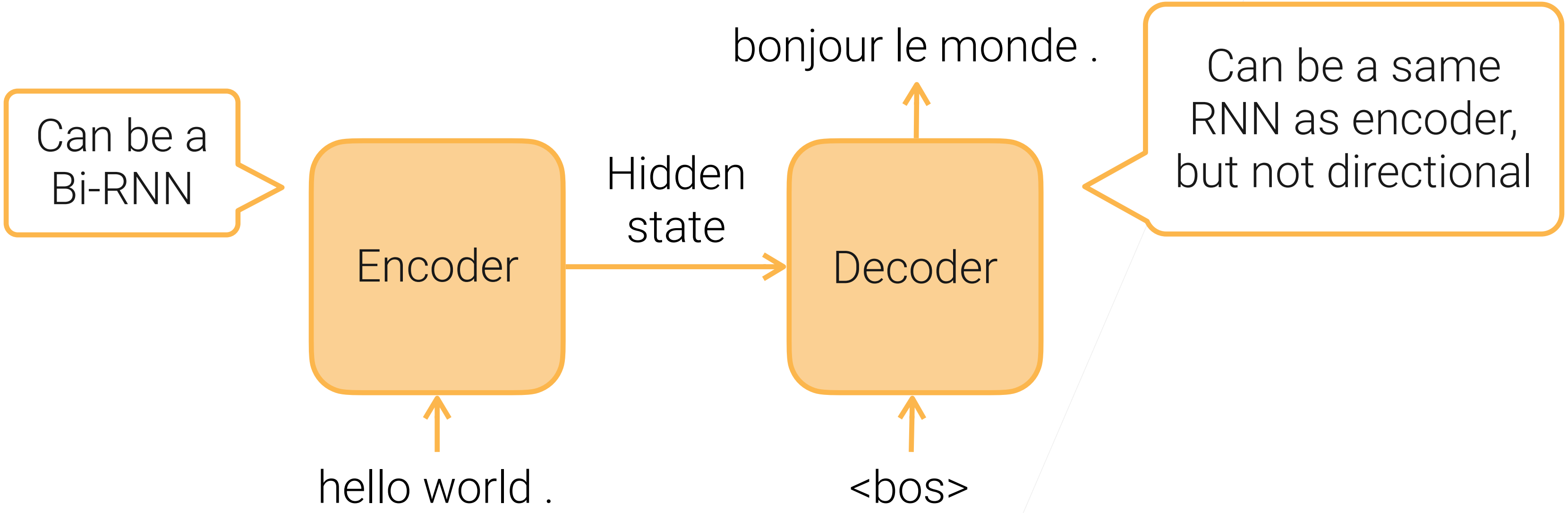
Attention

Qingqing Huang, Mu Li, Alex Smola

https://c.d2l.ai/stanford-cs329p

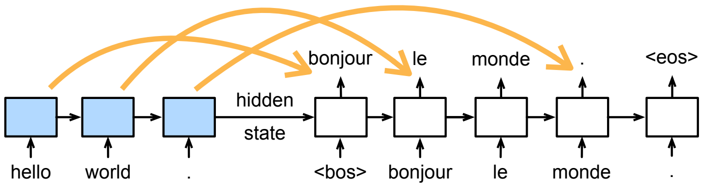
Encode-Decoder Architecture

- Break a NN into two parts: encoder and decoder
  - Especially when output more than a label
  - E.g. Sequence to sequence in machine translation:



Attention

- For RNN at time t, past info is in $\mathbf{h}_{t-1}$
  - Not directly use $\mathbf{h}_{t-2}, \ldots, \mathbf{h}_1$



- Attention uses $\alpha_1 \mathbf{h}_1 + \ldots + \alpha_{t-1} \mathbf{h}_{t-1}$, where $\alpha = \mathrm{softmax}(\mathbf{a})$
  - a_i is a learned attention score for the relation between $\mathbf{h}_i$ and $\mathbf{x}_t$
  - Scaled dot-product attention: $a_i = \langle \mathbf{h_i}, \mathbf{x_t} \rangle / \sqrt{d}$, d is the vector length
  - Additive attention: $a_i = \mathbf{v}^T \tanh(\mathbf{W}_h \mathbf{h_i} + \mathbf{W}_x \mathbf{x}_t)$, with learnable parameters $\mathbf{v}, \mathbf{W}_h, \mathbf{W}_x$
    - Can handle different $\mathbf{h_i}, \mathbf{x}_t$ with different length

Code

- Scaled dot-product attention:

```
# Shape of `queries`: (batch_size, #queries, d)
# Shape of `keys`: (batch_size, #keys, d)
# Shape of `values`: (batch_size, #values, d)
def dot_product_attention(queries, keys, values):
    d = queries.shape[-1]
    scores = torch.bmm(queries, keys.transpose(1, 2)) / math.sqrt(d)
    return torch.bmm(F.softmax(scores, dim=-1), values)
```

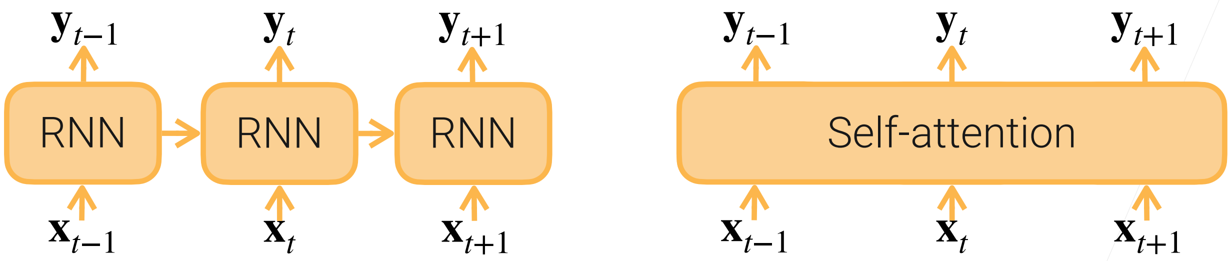
- The arguments to bmm: $\mathbf{X} \in \mathbb{R}^{n \times a \times b}$ and $\mathbf{Y} \in \mathbb{R}^{n \times b \times c}$
  - Return $\mathbf{O} \in \mathbb{R}^{n \times a \times c}$ with $\mathbf{O}_i = \mathbf{X}_i \mathbf{Y}_i$, for $i = 1, \ldots, n$

Full code: http://d2l.ai/chapter_attention-mechanisms/attention-scoring-functions.html

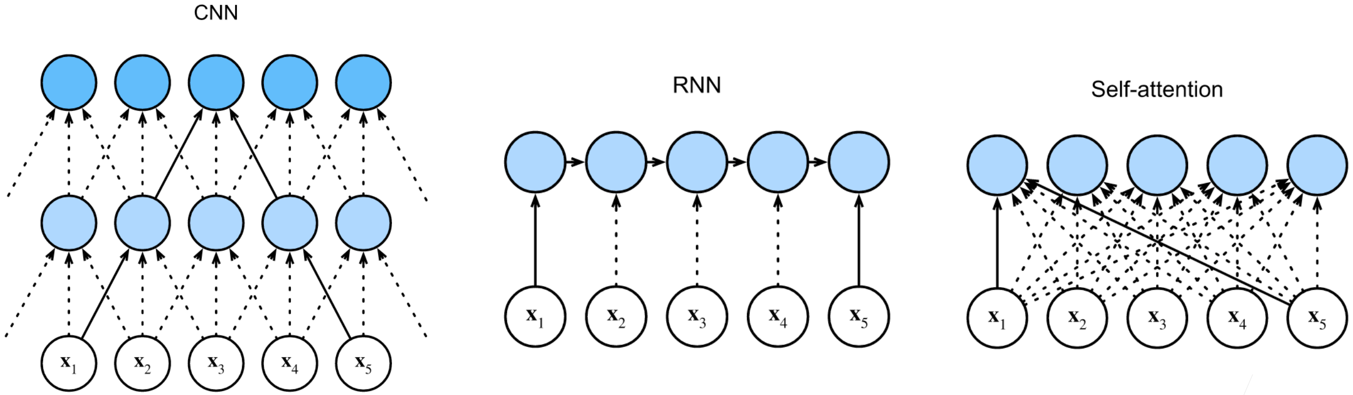
Self-attention

- Attention: given query $\mathbf{q} \in \mathbb{R}^q$, key-value pairs $(\mathbf{k}_i \in \mathbb{R}^k, \mathbf{v}_i \in \mathbb{R}^v)$, attention outputs $\sum_{i=1} \alpha_i \mathbf{v}_i \in \mathbb{R}^v$ with $\boldsymbol{\alpha} = \mathrm{softmax}(\mathbf{a})$, where $a_i = \mathrm{score}(\mathbf{q}, \mathbf{k}_i)$

- Self attention layer: same data for query, key, and value, outputs $\mathbf{y}_t = \sum_i \alpha_i^t \mathbf{x}_i$ for query $\mathbf{x}_t$ with $a_i^t = \mathrm{score}(\mathbf{x}_i, \mathbf{x}_t)$



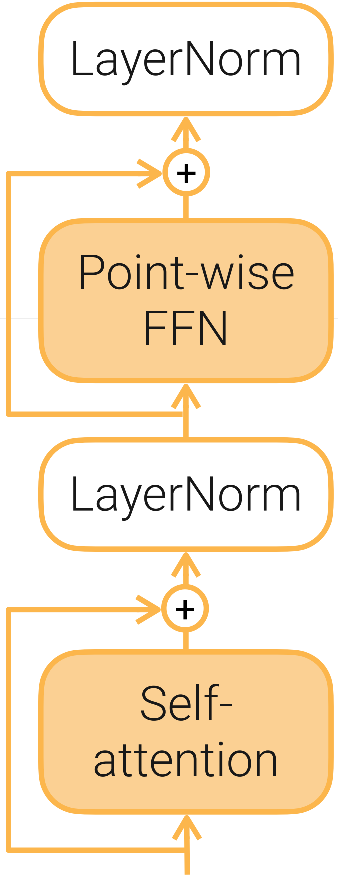
Compare self-attention with CNN and RNN



	CNN	RNN	Self-attention
Computation cost	O(knd^2)	O(nd^2)	O(n^2d)
Parallelization	O(n)	O(1)	O(n)
Max path length	O(n/k)	O(n)	O(1)

Transformers

- Transformer is an encoder-decoder model contains repeated transformer blocks
- Transformer block:
  - Multi-head self-attention to aggregate inputs weighted by element relations
  - Point-wise FFN uses a MLP to transform each output value, and share weights among values
  - LayerNorm and residual connections to make training easy



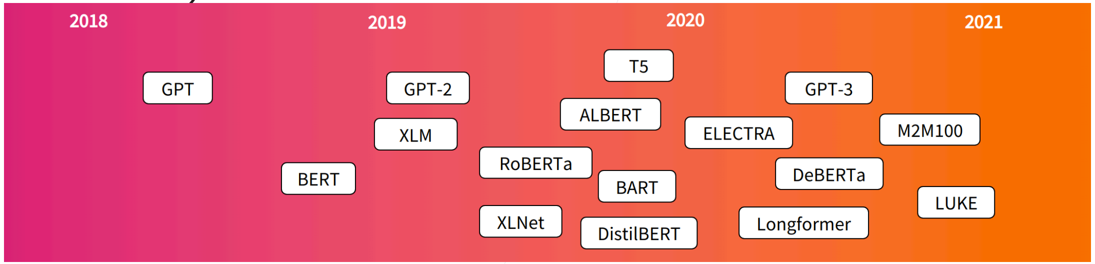
Code for Transformer Block

```
def multi_head_attetion(queries, keys, values, n_head):
    outputs = []
    for i in range(n_head):
        # W_q, W_k, W_v are linear layers
        outputs.append(dot_product_attention(
            W_q(queries), W_k(keys), W_v(values)))
    return W_o(torch.cat(outputs, dim=-1)) # W_o is a linear layer

ffn = nn.Sequential(nn.Linear(num_inputs, num_hiddens), nn.ReLU(),
                    nn.Linear(num_hiddens, num_outputs))

# shape of input `X`: (batch_size, seq_len, d)
def transformer_block(X):
    Y = nn.LayerNorm()(multi_head_attetion(X, X, X) + X)
    return nn.LayerNorm()(ffn(Y) + Y)
```

Full code: http://d2l.ai/chapter_attention-mechanisms/transformer.html

BERT and GPT

- BERT: big transformers with only the encoder
  - Good at encoding texts
- GPT: big transformers with only the decoder
  - Good at generate texts
- Modified the task to train on large-scale unlabelled corpus by self-training (more details later)
- Many variants now



https://huggingface.co/

Transformers in Vision

- A rising interest to apply Transformer beyond NLP
- ViT: extract a sequence of 16x16 batches from an image to input to a standard Transformer decode
- Transformers need more images compared to CNNs:
  - Convolution: leverage locality and translation invariance
  - Attention: learns general element relations in a sequence

Summary

- Attention: aggregate elements in a sequence based on element relations
- Transformers: an encode-decode model with stacked self-attention and MLP
  - Popularized by self-training models GPT and BERT
  - Becoming as another important NN architecture beyond MLP/CNN/RNN