



Quiz Submissions - Quiz 5

WENXIN ZHANG (username: wz2164)

Attempt 1

Written: Oct 21, 2021 1:11 PM - Oct 22, 2021 3:19 PM

Submission View

Your quiz has been submitted successfully.

Question 1

0.5 / 1 point

Select true statements about Transformers:

- ☐ Transformer is a feed-forward neural network.
- ☐ Transformers have constant number of computation steps (think of nodes in computation graph / function calls) regardless of the sequence length.
- ☐ Positional embedding allows transformer to learn ordering across tokens within a sequence.
- ☐ Self-attention introduces recurrence in a transformer.

Question 2

1 / 1 point

Given an input sequence

$$\mathbf{x} = (x_1, \dots, x_T),$$

transformer model computes the output at time step t as

$$z_t = f(\mathbf{x}).$$

Is the following statement True or False? **Output at time step t depends on input values from all time steps including t itself.**

- ☐ True
- ☐ False

Question 3

1 / 1 point

Are there any parameters in the transformer model that are shared across time steps?

- ☐ True
- ☐ False

Question 4

1 / 1 point

Recall, that transformer consists of **Self-Attention**, and **Feed-Forward** layers. Fill in the gaps about parallel computation in terms of transformers.

While

___Feed-Forward___ layers computation can be processed asynchronously across time steps, this is not true for ___Self-Attention___ type of layers, because attention weights are computed using the ___Softmax___ function whose denominator depends on the inputs from all time steps.

Question 5

1 / 1 point

Consider the operation of computing query-key scores:

$$QK^T, \quad Q, K \in \mathbb{R}^{t,d},$$

where t denotes the sequence length and d denotes the number of components in key/query vectors.

What is the computational complexity of this operation?

☐

$$O(td^2)$$

☐

$$O(td)$$

☐

$$O(t^2d)$$

☐

$$O(t^2d^2)$$

Question 6

1 / 1 point

Consider the following vocabulary:

$$V = (v_1, v_2, v_3, v_4).$$

Lets assume we have a given masked sequence

$$\mathbf{x}_t$$

(masked at position t) and the transformer model produced this vector of scores at position t:

$$\mathbf{z}_t = (1.5, -10, 6, 0),$$

where score 1.5 corresponds to the word v_1 and so on.

Consider the following softmax transformation with temperature tau (we have used such temperature in the section about sampling with BERT):

$$\mathbf{p}_{t,i} = \frac{\exp(\mathbf{z}_{t,i}/\tau)}{\sum_j \exp(\mathbf{z}_{t,j}/\tau)}.$$

Now assume that the temperature tau is set to approach 0. We sample a token from distribution parameterized by \mathbf{p}_t .

What can we say about that sampled token?

☐ The sampled token is

v_4

☐ The sampled token is

v_2

☐ The sampled token is

v_1

☐ The sampled token is

v_3

☐ It is unclear what token was sampled.

Question 7

0.5 / 1 point

Masking operation in Masked Language Modeling is a very common way to add noise in the data. Here we consider another way of making data noisy named PLM: Permuted Language Modeling. In this case we randomly select several positions from the input sequence, and permute tokens across these positions (keeping other tokens intact!). Then, we train the transformer model to recover the original sequence.

Consider an example:

Input sequence: "New York University is a private university".

Assume we randomly selected tokens in positions 0, 2, 5, and randomly shuffled them (positions start from 0).

Thus, we get a noisy sequence: "University York private is a New university."

There will be separate loss factor for every selected position. Write down the following derivations:

1. Derive the loss factor $loss(position)$ for token in position 0.
2. Derive the loss for entire sequence in terms of loss factors $loss(position)$.

Hint: every loss factor should be written in terms of the conditional probabilities under the model

p_θ

loss(position=0):

$$L = -\log p_\theta(x_0) = -\log p_\theta(x_0 | x_{<0}) = -\log p_\theta(university)$$

;

loss(entire sequence) = loss(position=0) + loss(position=2) + loss(position=5)

$$L = -[\log p_\theta(University) + \log p_\theta(private | University, New) + \log p_\theta(New | U$$

▼ Hide Feedback

-0.5: probability should be conditioned on the entire noisy sequence

Question 8

1 / 1 point

You are given an attention map for a sequence "[CLS] Newton played as [MASK] during Super Bowl 50 . [SEP]".

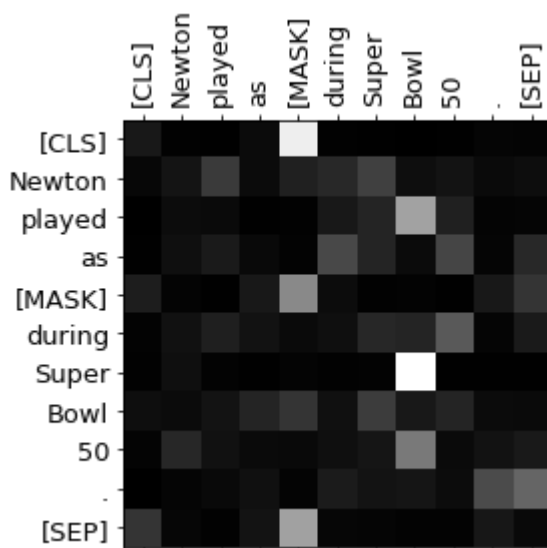
Let

$$a(t_k | t_q)$$

be an attention weight assigned by the self-attention module for **query** token t_q , and **key** token t_k .

Both t_q and t_k are some tokens from the given sequence.

Select true statements about the attention weights by investigating the attention map. Attention weight magnitude is depicted using amount of white. (more white -> higher weight)


☐

$$\operatorname{argmax}_{t_k} a(t_k | [\text{MASK}]) = [\text{MASK}]$$

☐

$$a(\text{Super} | \text{Bowl}) > a(\text{Bowl} | \text{Super})$$

☐

$$\sum_{t_k} a(t_k | [\text{SEP}]) < 1$$

Question 9

0 / 1 point

During the lab we implemented a Transformer Encoder model. In this question, we will try to convert it to the Transformer Decoder implementation. While pytorch features TransformerDecoderLayer class, we will try to implement it manually this time. In contrast to the Transformer Encoder, Decoder is an autoregressive model, i.e., in each self-attention layer, when calculating the attention scores for some query token x_t , model only considers preceding tokens x_{t-1} , x_{t-2} , ..., x_1 , ignoring tokens that follow x_t . In terms of implementation, such behavior is achieved by explicitly masking (i.e., setting

attention weights to zero) tokens on all time steps $t' > t$. We ask you to generate such a mask, that prevents transformer from considering future tokens when applying attention.

More information on masked attention: <https://jalammar.github.io/illustrated-gpt2/> (The illustrated masked attention section).

```
class Transformer(nn.Module):
    def __init__(self, vocab_size, max_len, pad_idx, dim=8, num_layers=4, nhead=2):
        super().__init__()
        self.token_embed = nn.Embedding(vocab_size, dim)
        self.position_embed = nn.Embedding(max_len, dim)
        encoder_layer = nn.TransformerEncoderLayer(d_model=dim, nhead=nhead,
dim_feedforward=64, dropout=0.0)
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.projection = nn.Linear(dim, vocab_size)
        self.pad_idx = pad_idx
        self.max_len = max_len

    def features(self, token_indices):
        pos = torch.arange(len(token_indices), device=token_indices.device).unsqueeze(1)
        x = self.token_embed(token_indices) + self.position_embed(pos)
        attn_mask = ~token_indices.ne(self.pad_idx).transpose(0, 1)
        mask =
        ___torch.triu(torch.ones(len(token_indices), len(token_indices)))___

        mask.masked_fill(mask==1, float('-inf')).to(token_indices.device)
        x = self.encoder(x, src_key_padding_mask=attn_mask, src_mask=mask)
        return x

    def forward(self, token_indices):
        x = self.features(token_indices) # shape: [seq_len x batch_size x hidden_size]
        x = self.projection(x) # shape: [seq_len x batch_size x vocab_size]
        return x
```

▶ View Feedback

Question 10

0.1 / 0.1 points

How long did you spend to complete this quiz (in hours)?

1.5H

Done