7ervudnat

November 28, 2024

-01-24 (.   1132249576)

:

tensorflow 2.

.   (VQA)

. Relational reasoning –   CLEVR.

GitHub   .

Tensorflow

```
[1]: #      Tensorflow 2
     # pip install tensorflow / conda install tensorflow

     #                  tensorflow
     import tensorflow as tf
     print(f"TensorFlow version: {tf.__version__}")
```

```
TensorFlow version: 2.17.0
```

```
[3]: #
     import tensorflow as tf
     from tensorflow.keras import layers
     import json
     import random
     import numpy as np
     from tensorflow.keras.preprocessing.text import Tokenizer
     from tensorflow.keras.preprocessing.sequence import pad_sequences
     import matplotlib.pyplot as plt

     #                  CLEVR,                   (              )
     path_to_dataset = "C:
      ↪\\Users\\4eka0\\Downloads\\CLEVR\\questions\\CLEVR_train_questions.json"

     #       JSON-
     with open(path_to_dataset, 'r') as f:
         clevr_data = json.load(f)
```

```
questions = clevr_data['questions']
print(f"               : {len(questions)}")
```

```
              : 699989
```

[7]:

```
#          1000
random.shuffle(questions)
train_data = questions[:1000]

#
all_questions = [q['question'] for q in train_data]
all_answers = [q['answer'] for q in train_data]

#
unique_answers = list(set(all_answers))
answer_to_index = {ans: i for i, ans in enumerate(unique_answers)}
indexed_answers = [answer_to_index[ans] for ans in all_answers]

#
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_questions)
sequences = tokenizer.texts_to_sequences(all_questions)
word_index = tokenizer.word_index

#
max_len = max(len(seq) for seq in sequences)
padded_sequences = pad_sequences(sequences, maxlen=max_len)

#                numpy
y_train = np.array(indexed_answers)
```

[21]:

```
def build_transformer_model(vocab_size, max_len, num_classes):
    inputs = layers.Input(shape=(max_len,))
    embedding = layers.Embedding(input_dim=vocab_size, output_dim=128)(inputs)

    #
    x = layers.MultiHeadAttention(num_heads=4, key_dim=128)(embedding,␣
 ↪embedding)
    x = layers.LayerNormalization()(x + embedding)
    x = layers.GlobalAveragePooling1D()(x)

    #
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    model = tf.keras.Model(inputs, outputs)
    return model

#
model = build_transformer_model(vocab_size=len(word_index) + 1,␣
 ↪max_len=max_len, num_classes=len(unique_answers))

#
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

#
model.fit(padded_sequences, y_train, epochs=30, batch_size=32)
```

```
Epoch 1/30
32/32                5s 41ms/step -
accuracy: 0.1730 - loss: 2.7975
Epoch 2/30
32/32                1s 40ms/step -
accuracy: 0.2859 - loss: 2.2755
Epoch 3/30
32/32                1s 40ms/step -
accuracy: 0.3489 - loss: 1.6901
Epoch 4/30
32/32                1s 41ms/step -
accuracy: 0.4270 - loss: 1.4958
Epoch 5/30
32/32                1s 41ms/step -
accuracy: 0.4424 - loss: 1.4335
Epoch 6/30
32/32                1s 41ms/step -
accuracy: 0.4442 - loss: 1.2815
Epoch 7/30
32/32                1s 42ms/step -
accuracy: 0.4726 - loss: 1.1871
Epoch 8/30
32/32                1s 41ms/step -
accuracy: 0.4827 - loss: 1.1892
Epoch 9/30
32/32                1s 42ms/step -
accuracy: 0.5237 - loss: 1.1319
Epoch 10/30
32/32                1s 43ms/step -
accuracy: 0.5526 - loss: 1.0643
Epoch 11/30
32/32                1s 43ms/step -
accuracy: 0.5328 - loss: 1.0515
```

```
Epoch 12/30
32/32             1s 42ms/step -
accuracy: 0.5173 - loss: 1.0858
Epoch 13/30
32/32             1s 39ms/step -
accuracy: 0.5321 - loss: 1.0480
Epoch 14/30
32/32             1s 39ms/step -
accuracy: 0.5209 - loss: 1.0134
Epoch 15/30
32/32             1s 40ms/step -
accuracy: 0.5758 - loss: 0.9567
Epoch 16/30
32/32             1s 41ms/step -
accuracy: 0.5403 - loss: 1.0148
Epoch 17/30
32/32             1s 41ms/step -
accuracy: 0.5621 - loss: 0.9747
Epoch 18/30
32/32             1s 43ms/step -
accuracy: 0.5667 - loss: 0.9799
Epoch 19/30
32/32             1s 41ms/step -
accuracy: 0.5639 - loss: 0.9457
Epoch 20/30
32/32             1s 43ms/step -
accuracy: 0.5759 - loss: 0.9371
Epoch 21/30
32/32             1s 42ms/step -
accuracy: 0.6100 - loss: 0.8819
Epoch 22/30
32/32             1s 43ms/step -
accuracy: 0.6172 - loss: 0.8796
Epoch 23/30
32/32             1s 41ms/step -
accuracy: 0.5827 - loss: 0.9088
Epoch 24/30
32/32             1s 42ms/step -
accuracy: 0.6032 - loss: 0.8633
Epoch 25/30
32/32             1s 39ms/step -
accuracy: 0.6021 - loss: 0.8936
Epoch 26/30
32/32             1s 41ms/step -
accuracy: 0.6062 - loss: 0.8831
Epoch 27/30
32/32             1s 41ms/step -
accuracy: 0.6513 - loss: 0.7932
```

```
Epoch 28/30
32/32              1s 43ms/step -
accuracy: 0.6416 - loss: 0.7918
Epoch 29/30
32/32              1s 42ms/step -
accuracy: 0.6439 - loss: 0.8188
Epoch 30/30
32/32              1s 41ms/step -
accuracy: 0.6316 - loss: 0.8079
```

[21]:
```
<keras.src.callbacks.history.History at 0x1c262c2afc0>
```

10

[23]:
```python
#
def test_model(model, tokenizer, questions, answers, num_samples=10):
    #
    random_questions = random.sample(list(zip(questions, answers)), num_samples)
    for i, (question, answer) in enumerate(random_questions):
        sequence = tokenizer.texts_to_sequences([question])
        padded = pad_sequences(sequence, maxlen=max_len)
        prediction = model.predict(padded)
        predicted_answer = unique_answers[np.argmax(prediction)]

        print(f"    {i + 1}: {question}")
        print(f"        : {answer}")
        print(f"           : {predicted_answer}")
        print("-" * 50)

#
test_model(model, tokenizer, all_questions, all_answers)
```

```
1/1              0s 234ms/step
    1: There is a green rubber object in front of the green thing to the left
of the small matte thing that is in front of the large metallic cylinder; how
big is it?
        : small
           : small
--------------------------------------------------
1/1              0s 36ms/step
    2: How many other things are the same material as the tiny red cube?
        : 1
           : 1
--------------------------------------------------
1/1              0s 37ms/step
    3: Is the number of large green matte spheres that are left of the small
brown thing less than the number of cubes right of the green rubber ball?
        : no
```

```
          : yes
--------------------------------------------------
1/1             0s 38ms/step
    4: What number of tiny brown things have the same shape as the green
object?
        : 0
          : 0
--------------------------------------------------
1/1             0s 39ms/step
    5: Are there fewer large purple balls that are in front of the ball than
purple spheres that are in front of the small yellow cube?
        : yes
          : yes
--------------------------------------------------
1/1             0s 38ms/step
    6: Is the number of big rubber things to the left of the big purple
metallic thing the same as the number of brown rubber things?
        : yes
          : yes
--------------------------------------------------
1/1             0s 38ms/step
    7: Is there any other thing that is the same material as the small cyan
thing?
        : yes
          : yes
--------------------------------------------------
1/1             0s 39ms/step
    8: Are the cube to the left of the large matte cube and the big purple
cylinder made of the same material?
        : no
          : no
--------------------------------------------------
1/1             0s 39ms/step
    9: Is the number of cyan metallic spheres that are behind the green
object less than the number of rubber things that are behind the brown matte
object?
        : yes
          : yes
--------------------------------------------------
1/1             0s 45ms/step
    10: What size is the green sphere that is made of the same material as
the blue thing?
        : small
          : metal
--------------------------------------------------
```

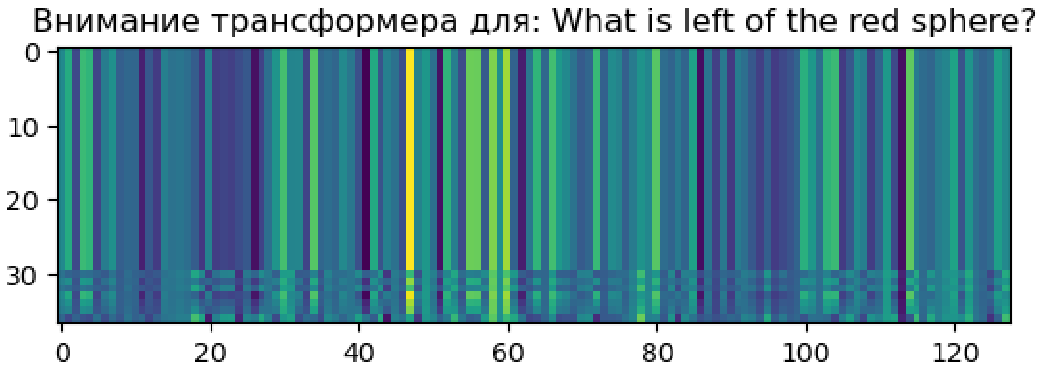
[25]:

```
def visualize_attention(model, sentence, tokenizer):
    sequence = tokenizer.texts_to_sequences([sentence])
    padded = pad_sequences(sequence, maxlen=max_len)

    #
    attention_layer = model.layers[2]  # MultiHeadAttention
    attention_output = tf.keras.Model(inputs=model.input,␣
 ↪outputs=attention_layer.output)
    attention_weights = attention_output(padded)

    plt.imshow(attention_weights[0].numpy(), cmap='viridis')
    plt.title(f"              : {sentence}")
    plt.show()

#
example_question = "What is left of the red sphere?"
visualize_attention(model, example_question, tokenizer)
```



GitHub

GitHub: https://github.com/SaniyaIslamova26/LPforAI/tree/main