Advanced NLP Assignment 4

12112627 李乐平

Q1. TinyBERT trains a smaller BERT model with knowledge distillation.

1. Illustrate how knowledge distillation works in TinyBERT.

Solution: TinyBERT uses a Transformer distillation method that fits different representations from BERT layers, such as the embedding layer, the hidden states, the attention matrices, and the prediction layer. TinyBERT also uses a two-stage learning framework that performs Transformer distillation at both the pre-training and task-specific learning stages. This ensures that TinyBERT can capture the general-domain as well as the task-specific knowledge in BERT. The general distillation stage uses the original BERT without fine-tuning as the teacher and a large-scale text corpus as the training data. The task-specific distillation stage uses the fine-tuned BERT as the teacher and an augmented task-specific dataset as the training data. The data augmentation procedure replaces some words in the original data with similar words predicted by BERT or GloVe embeddings.

2. Point out the file name and line number(s) of the code which determines the layers in the teacher model in Transformer distillation, i.e. $g(m)$function in Equation (6).

Solution: The script of general_distill.py from line 413 to line 433 performs the layer mapping.

```
student_atts, student_reps = student_model(input_ids, segment_ids, input_mask)
teacher_reps, teacher_atts, _ = teacher_model(input_ids, segment_ids, input_mask)
teacher_reps = [teacher_rep.detach() for teacher_rep in teacher_reps]  # speedup 1.5x
teacher_atts = [teacher_att.detach() for teacher_att in teacher_atts]

teacher_layer_num = len(teacher_atts)
student_layer_num = len(student_atts)
assert teacher_layer_num % student_layer_num == 0
layers_per_block = int(teacher_layer_num / student_layer_num)
new_teacher_atts = [teacher_atts[i * layers_per_block + layers_per_block - 1]
                    for i in range(student_layer_num)]

for student_att, teacher_att in zip(student_atts, new_teacher_atts):
    student_att = torch.where(student_att <= -1e2, torch.zeros_like(student_att).to(device),
                              student_att)
    teacher_att = torch.where(teacher_att <= -1e2, torch.zeros_like(teacher_att).to(device),
                              teacher_att)
    att_loss += loss_mse(student_att, teacher_att)

new_teacher_reps = [teacher_reps[i * layers_per_block] for i in range(student_layer_num + 1)]
new_student_reps = student_reps

for student_rep, teacher_rep in zip(new_student_reps, new_teacher_reps):
    rep_loss += loss_mse(student_rep, teacher_rep)
```

As well as the line 941 to line 959 of task_distill.py.

```
if not args.pred_distill:
    teacher_layer_num = len(teacher_atts)
    student_layer_num = len(student_atts)
    assert teacher_layer_num % student_layer_num == 0
    layers_per_block = int(teacher_layer_num / student_layer_num)
    new_teacher_atts = [teacher_atts[i * layers_per_block + layers_per_block - 1]
                        for i in range(student_layer_num)]

    for student_att, teacher_att in zip(student_atts, new_teacher_atts):
        student_att = torch.where(student_att <= -1e2, torch.zeros_like(student_att).to(device),
                                  student_att)
        teacher_att = torch.where(teacher_att <= -1e2, torch.zeros_like(teacher_att).to(device),
                                  teacher_att)

        tmp_loss = loss_mse(student_att, teacher_att)
        att_loss += tmp_loss

    new_teacher_reps = [teacher_reps[i * layers_per_block] for i in range(student_layer_num + 1)]
    new_student_reps = student_reps
```

Q2. BertViz is an interactive tool for visualizing attention in Transformer language models. Following the examples in the github repo and plot the model view (attention-all) of the Flan-T5-small model with the model input and output texts from three different tasks (you can decide the texts as you like).

In [3]:
```
# !python -m pip install --user bertviz
```

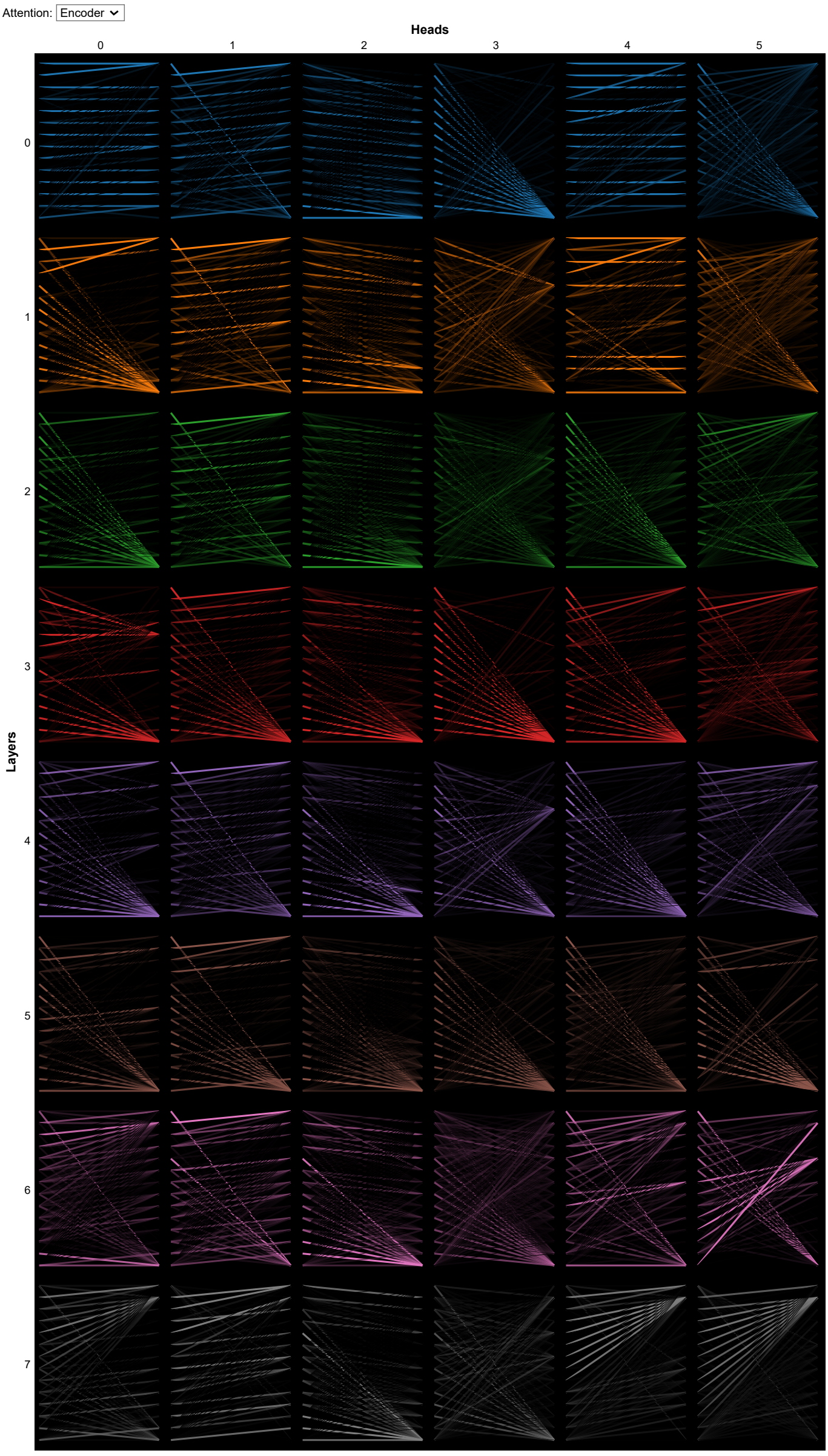
In [1]:
```
from transformers import T5Tokenizer, T5ForConditionalGeneration, utils
from bertviz import model_view, head_view
utils.logging.set_verbosity_error()  # Suppress standard warnings

model_name = "./Q2_flan_t5_small_ckpt"

model = T5ForConditionalGeneration.from_pretrained(model_name, output_attentions=True)
tokenizer = T5Tokenizer.from_pretrained(model_name)
```

```
C:\Users\Wandering Troubadour\AppData\Roaming\Python\Python310\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthedocs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
prefix = "Translate English to German: "
input_text = "She sees the small elephant."


outputs = model.generate(tokenizer.encode(prefix + input_text, return_tensors="pt"))
output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

encoder_input_ids = tokenizer(prefix + input_text, return_tensors="pt", add_special_tokens=True).input_ids

with tokenizer.as_target_tokenizer():
    decoder_input_ids = tokenizer(output_text, return_tensors="pt", add_special_tokens=True).input_ids

outputs = model(input_ids=encoder_input_ids, decoder_input_ids=decoder_input_ids)

encoder_text = tokenizer.convert_ids_to_tokens(encoder_input_ids[0])
decoder_text = tokenizer.convert_ids_to_tokens(decoder_input_ids[0])

model_view(
    encoder_attention=outputs.encoder_attentions,
    decoder_attention=outputs.decoder_attentions,
    cross_attention=outputs.cross_attentions,
    encoder_tokens= encoder_text,
    decoder_tokens=decoder_text
)
```

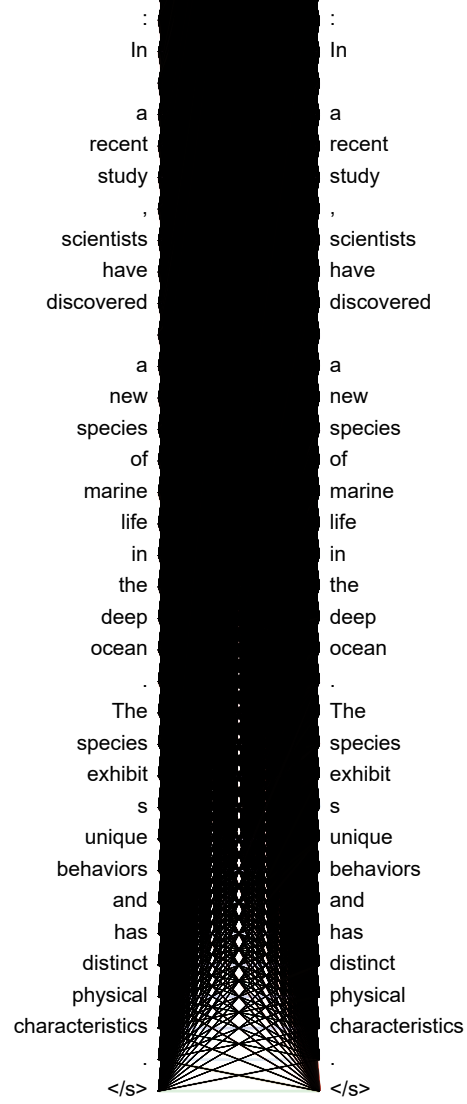
```
C:\Users\Wandering Troubadour\AppData\Roaming\Python\Python310\site-packages\transformers\generation\utils.py:1273: UserWarning: Using the model-agnostic default `max_length` (=20) to control the generation length. We recommend setting `max_new_tokens` to control the maximum length of the generation.
  warnings.warn(
C:\Users\Wandering Troubadour\AppData\Roaming\Python\Python310\site-packages\transformers\tokenization_utils_base.py:3856: UserWarning: `as_target_tokenizer` is deprecated and will be removed in v5 of Transformers. You can tokenize your labels by using the argument `text_target` of the regular `__call__` method (either in the same call as your input texts if you use the same keyword arguments, or in a separate call.
  warnings.warn(
```

Attention: Encoder



In [3]:
```
prefix = "Task: Given a piece of text or a document, generate a concise summary that captures the essential information."
input_text = """Context: In a recent study, scientists have discovered a new species of marine life in the deep ocean. The species exhibits unique behavior
"""
outputs = model.generate(tokenizer.encode(prefix + input_text, return_tensors="pt"))
output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(output_text)
encoder_input_ids = tokenizer(prefix + input_text, return_tensors="pt", add_special_tokens=True).input_ids

with tokenizer.as_target_tokenizer():
    decoder_input_ids = tokenizer(output_text, return_tensors="pt", add_special_tokens=True).input_ids

outputs = model(input_ids=encoder_input_ids, decoder_input_ids=decoder_input_ids)

encoder_text = tokenizer.convert_ids_to_tokens(encoder_input_ids[0])
decoder_text = tokenizer.convert_ids_to_tokens(decoder_input_ids[0])

head_view(
    encoder_attention=outputs.encoder_attentions,
    decoder_attention=outputs.decoder_attentions,
    cross_attention=outputs.cross_attentions,
    encoder_tokens= encoder_text,
    decoder_tokens=decoder_text
)
```

```
Scientists have discovered a new species of marine life in the deep ocean.
```



In [4]:
```
prefix = "Task: Answer questions based on a given context or passage."
input_text = """
Context: "The Eiffel Tower is a famous landmark in Paris, France. It was completed in 1889 and has become a symbol of the city."
Question: "When was the Eiffel Tower completed?"""

outputs = model.generate(tokenizer.encode(prefix + input_text, return_tensors="pt"))
output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(output_text)
encoder_input_ids = tokenizer(prefix + input_text, return_tensors="pt", add_special_tokens=True).input_ids

with tokenizer.as_target_tokenizer():
    decoder_input_ids = tokenizer(output_text, return_tensors="pt", add_special_tokens=True).input_ids

outputs = model(input_ids=encoder_input_ids, decoder_input_ids=decoder_input_ids)

encoder_text = tokenizer.convert_ids_to_tokens(encoder_input_ids[0])
decoder_text = tokenizer.convert_ids_to_tokens(decoder_input_ids[0])

head_view(
    encoder_attention=outputs.encoder_attentions,
    decoder_attention=outputs.decoder_attentions,
    cross_attention=outputs.cross_attentions,
    encoder_tokens= encoder_text,
    decoder_tokens=decoder_text
)
```

1889

Q3:

LangChain is a framework for developing applications powered by language models. **1.** Read the langchain documentation above. List five applications you can implement with the help of Langchain.

Solution: Here are five potential applications that can be implemented with LangChain:

1. **Chatbots**: LangChain can be used to develop chatbots that can understand natural language and provide context-aware responses to users.
2. **Content creation**: LangChain can be used to develop applications that can generate content such as articles, summaries, and reports based on user input.
3. **Language translation**: LangChain can be used to develop language translation applications that can translate text from one language to another.
4. **Search engines**: LangChain can be used to develop search engines that can understand natural language queries and provide relevant search results.
5. **Sentiment analysis**: LangChain can be used to develop applications that can analyze the sentiment of text data such as customer feedback, social media posts, and news articles.

2. Write the code to design one application you listed above. You can use open-sourced LLMs from huggingface platform.

In [44]:
```
# !pip install --user langchain
```

In [ ]:
```
import time
import logging
import requests
from typing import Optional, List, Dict, Mapping, Any

import langchain
from langchain.llms.base import LLM
from langchain.cache import InMemoryCache
from langchain.chains import ConversationChain
from langchain.schema import HumanMessage, SystemMessage


logging.basicConfig(level=logging.INFO)
langchain.llm_cache = InMemoryCache()

class ChatGLM(LLM):

    url = "http://172.18.30.162:11137/chat"

    @property
    def _llm_type(self) -> str:
        return "chatglm"

    def _construct_query(self, prompt: str) -> Dict:
        query = {
            "human_input": prompt
        }
        return query

    @classmethod
    def _post(cls, url: str,
        query: Dict) -> Any:
        _headers = {"Content_Type": "application/json"}
        with requests.session() as sess:
            resp = sess.post(url,
                json=query,
                headers=_headers,
                timeout=600)
        return resp

    def _call(self, prompt: str,
        stop: Optional[List[str]] = None) -> str:
        # construct query
        query = self._construct_query(prompt=prompt)

        # post
        resp = self._post(url=self.url,
            query=query)

        if resp.status_code == 200:
            resp_json = resp.json()
            predictions = resp_json["response"]
            return predictions
        else:
            return "请求模型"

    @property
    def _identifying_params(self) -> Mapping[str, Any]:
        """Get the identifying parameters.
        """
        _param_dict = {
            "url": self.url
        }
        return _param_dict

if __name__ == "__main__":
    llm = ChatGLM()
    conversation = ConversationChain(llm = llm)
    with open("system_prompt.txt", "r", encoding = "utf-8") as file:
        prefix = file.read()

    prompts = [SystemMessage(content = prefix)]

#     print("Try connecting...")

#     llm(prompts, stop=["you"])
#     conversation.run(prefix)

#     print("Successfully connected.")


    while True:
        human_input = input("你: ")
        begin_time = time.time() * 1000

#         prompts.append(HumanMessage(content = human_input))
        response = conversation.run(human_input)

        end_time = time.time() * 1000
        used_time = round(end_time - begin_time, 3)
        logging.info(f"Process time: {used_time}ms")

        print(f"AI: {response}")
```

```
你：你好

INFO:root:Process time: 5058.873ms

AI：你好！有什么我可以帮助你的吗？
你：你了解什么是自然语言处理吗？

INFO:root:Process time: 32277.917ms

AI：自然语言处理(Natural Language Processing,NLP)是一种人工智能领域，旨在使计算机理解和处理人类语言。它涉及到语音识别、机器翻译、文本分类、情感分析、信息提取和对话系统等任务。NLP技术广泛应用于许多领域，例如搜索引擎、社交媒体、在线客服、智能助手、机器翻译、金融、医疗和法律等。
```