Deep Learning 2024 Assignment 3

This is a **group assignment**, and its deadline is **Friday, Januar 10, 2025, 22:00**. You must submit your solution electronically via the Absalon home page.

1 Segment Anything

In this assignment, you will experiment with the Segment Anything Model (SAM) [1] on the x-ray dataset for lung segmentation. Your goal is to check whether the claim of the ability to segment *"anything"* also extends to medical x-ray images.

Start by reading the *Segment Anything* paper and familiarize yourself with how the model was trained and evaluated. Your tasks are:

1.1. **Discussion:** Based on the training of the model and the evaluation results presented in the paper, do you expect SAM to perform well on medical x-ray segmentation tasks?

1.2. **Experiment:** Use the interactive web interface at `https://segment-anything.com/demo`. Use the three provided lung x-ray images and try to segment the lungs interactively using:

- Point prompting
- Bounding box prompting
- Automatic mode

Analyze the qualitative results, discuss the pros and cons of each method, and evaluate SAM's general suitability for the given task.

2 Prompt Engineering in Python

To implement SAM with code, consult the example notebook `https://github.com/facebookresearch/segment-anything/blob/main/notebooks/predictor_example.ipynb`. You are provided with `sam_lung_xrays.ipynb`. Your tasks are:

2.1. Implement the missing parts for loading and transforming the data for the SAM predictor object (indicated by `TODO` comments). Select an appropriate prompt to segment the lungs in a single example image. Visualize your approach by overlaying the example image with your prompt (e.g., points or bounding boxes) and the resulting segmentation mask.

2.2. Evaluate your prompting method on the validation split of the dataset. Report the average and standard deviation of the F1-score. Analyze the results qualitatively on two selected example images.

3 Fine-tuning Language Model

In this exercise, we will fine-tune a language model for a text classification task. We will use a transformer architecture and analyze the difference in attention maps between a fine-tuned and a pre-trained (not fine-tuned) model. In the next exercise, we will use few-shot prompting for the same task and compare the performances. You will use PyTorch and the Hugging Face Transformers library (docs) to implement and analyze these tasks. Remember to enable the GPU on Colab.

We will use the following datasets and models:

Dataset: Use a small classification dataset, such as the AG News (topic classification) **or** IMDb (sentiment analysis) dataset. Ensure the dataset is downloaded and preprocessed into training and testing splits.

Models: Pre-trained models: *bert-base-uncased* and *gpt2*

Here is a small code snippet to load the data and set up the model:

```python
from transformers import BertForSequenceClassification, Trainer,
    TrainingArguments
from datasets import load_dataset

# Load dataset
dataset = load_dataset('ag_news')

# Load model and tokenizer
model = BertForSequenceClassification.from_pretrained("bert-base-
    uncased", num_labels=4)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Preprocess data
def preprocess_function(examples):
    return tokenizer(examples['text'], truncation=True, padding=True)

encoded_dataset = dataset.map(preprocess_function, batched=True)

# Training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    num_train_epochs=3
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=encoded_dataset["train"],
    eval_dataset=encoded_dataset["test"]
)

trainer.train()
```

Listing 1: Fine-tuning BERT for Text Classification

Your tasks are:

3.1. Give a brief description of the dataset including its purpose, structure, and key characteristics.

3.2. Provide a brief dataset analysis with summary statistics (e.g., number of samples per class) and representative examples to illustrate the data.

3.3. Fine-tune the BERT Model. Fine-tune *bert-base-uncased* on your dataset using the Trainer API or a custom training loop. Save the fine-tuned model for later use.

3.4. Write a script to extract and visualize attention maps from the last attention layer of the pre-trained (not fine-tuned) model and fine-tuned model. *Hint: Use the outputs.attentions from the model's forward pass and the code snippet provided below.*

3.5. Select a few example sentences and compare the attention maps between the two models. Make visualizations of the attention maps either as screenshots or plots showcasing the attention maps for both the pre-trained and fine-tuned models, highlighting notable differences and observations.

Code snippet for extracting attention maps:

```python
from transformers import BertTokenizer, BertModel
import torch

model = BertModel.from_pretrained("bert-base-uncased",
    output_attentions=True)
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Example input
inputs = tokenizer("The movie was fantastic!", return_tensors="pt")

# Get attentions
outputs = model(**inputs)
attentions = outputs.attentions  # List of attention maps for each
    layer
```

Listing 2: Extracting Attention Maps with BERT

4 Few-shot Prompting Language Model

In this exercise, we use few-shot prompting to try to improve the performance on the text classification task. For our task, few-shot prompting could look something like this:

> Classify the sentiment of the following movie reviews:
> Review: The movie was fantastic, with stunning visuals and a great story.
> Sentiment: Positive
> Review: I found the film boring and overly long. Not worth watching.
> Sentiment: Negative
> Review: The acting was top-notch, but the plot had some flaws.
> Sentiment: Positive
> Review: [Test review here]
> Sentiment:

We will use a pre-trained GPT-2 model. Here is an incomplete code snippet to get started:

```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import torch

# TODO: Load the GPT-2 model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')
model.eval()

# TODO: Prepare the few-shot prompt with examples
few_shot_prompt =

# TODO: Define a function to get GPT-2 prediction for a new sentence
def get_gpt2_prediction(sentence):
    prompt = few_shot_prompt + f"Sentence: {sentence}\nSentiment:"
    inputs = tokenizer.encode(\_\_\_, return_tensors='pt')  # Complete
     the input
    outputs = model.generate(
        inputs,
        max_length=inputs.shape[1] + 10,
        do_sample=False,
        num_beams=5,
        early_stopping=True
        output_attentions=True
    )
    generated_text = tokenizer.decode(outputs[0])
    # TODO: Extract the sentiment prediction from the generated text
    predicted_sentiment = \_\_\_  # Complete this part
    return predicted_sentiment
```

Listing 3: Few-shot Prompting with GPT-2 (Complete the Missing Parts)

Your tasks are:

4.1. Implement few-shot prompting with GPT-2 for the classification task you used in the previous exercise. See Hints below.

4.2. Use GPT-2 to generate predictions for the test set based on the prompt. See Hints below.

4.3. Compare the classification accuracy between the GPT-2 few-shot prompting approach, the pre-trained (not fine-tuned) BERT model, and the fine-tuned BERT model. Report the accuracy, precision, recall, and F1-score for both models. Which model performs better? Discuss scenarios where one approach may outperform the other.

4.4. Discuss the trade-offs between fine-tuning and few-shot prompting in terms of computational resources, ease of implementation, and performance. Given GPT-2's generative nature, what are the potential limitations when applying it to classification tasks? Which approach would you recommend for a low-resource language? Why?

4.5. Provide and compare the attention map visualizations for the same example sentences as above (note that GPT-2's attention mechanisms might differ). See Hints below. How do attention patterns differ between BERT and GPT-2?

Hints:

- Include training data when designing the few-shot prompt. For example, the IMDb dataset could look like this:

Classify the sentiment of the following movie reviews:
Review: The movie was fantastic, with stunning visuals and a great story.
Sentiment: Positive
Review: I found the film boring and overly long. Not worth watching. Sentiment: Negative
Review: The acting was top-notch, but the plot had some flaws.
Sentiment: Positive
Review: [Test review here]
Sentiment:

- Tailor the few-shot examples to the specific task (binary sentiment for IMDb, multi-class classification for AG News). For AG News, explicitly specify the options (e.g., "Options: World, Sports, Business, Science/Technology") to guide GPT-2's output.
- Use post-processing to map outputs like positive, Positive., or variations back to clean labels.
- Ensure that the generation is controlled to prevent long or irrelevant outputs. Use parameters like *max_length* and *stop_token*.
- GPT-2 also provides attention outputs if you set *output_attentions=True*. You can extract and visualize them similarly to BERT.

References

[1] A. Kirillov *et al. Segment Anything.* 2023. URL: `https://arxiv.org/abs/2304.02643`.