Deep Learning 2025 Assignment 3

This is a **group assignment**, and its deadline is **Friday, January 9, 2026, 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 Chain-of-Thought Reasoning

In this exercise, you will investigate how *chain-of-thought* (CoT) prompting affects the answers produced by an instruction-tuned language model on multiple-choice science questions. Rather than focusing on training, we will compare its predictions:

- without CoT (direct answer), and
- with CoT (three reasoning steps + answer).

Your goal is to analyze when CoT helps, when it hurts, and what kind of reasoning the model appears to perform. You may use any reasonably sized instruction-tuned model available in the Hugging Face `transformers` library, such as `google/flan-t5-small`, `google/flan-t5-base`, `declare-lab/flan-alpaca-base`, or similar. Make sure the model is instruction-tuned / chat-style (so it can follow the formatting prompt). Your tasks are:

4.1. **Load a model:** Choose a lightweight instruction-tuned model (see examples above) and load it using Hugging Face `transformers`. Clearly state in your report which model you used.

4.2. **Sample 200 questions from ScienceQA (text-only):** Use the `ScienceQA` dataset (text-only subset) and randomly select 200 multiple-choice questions that have exactly four answer options. Format each question so that the options correspond to letters A–D.

> **Example format**
> Question: The main purpose of a lock is to ...
> (A) open things easily (B) secure valuable items (C) decorate furniture (D) make noise

4.3. **Prompt without CoT (baseline):** For each question, construct a simple prompt that asks the model to directly output the final answer as a letter A–D and record the predicted letter for each question. For example:

```
Question: ...
Options:
(A) ...
(B) ...
(C) ...
(D) ...
Answer:
```

4.4. **Prompt with CoT:** For the same questions, use the following instruction so that the model outputs three reasoning steps followed by the final answer. Apply this prompt to each question and save:

- the three reasoning steps (or the first three, if more are given),
- the final predicted letter.

```
You are a careful reasoning assistant that answers multiple-choice
questions about science. First, think step by step. Then, give the
final answer. Format exactly as:
Reasoning: <step 1>
Reasoning: <step 2>
Reasoning: <step 3>
Answer: <final letter>
```

4.5. **Flip analysis:** Compare the baseline prediction (no CoT) with the CoT prediction for each question and categorize each example into one of four groups:

- **Flip-to-correct**: baseline incorrect, CoT correct.
- **Flip-to-incorrect**: baseline correct, CoT incorrect.
- **Correct–Correct**: baseline correct, CoT also correct.
- **Incorrect–Incorrect**: baseline incorrect, CoT also incorrect.

Compute and report the counts for each category, and overall accuracy without CoT vs. with CoT. Visualize the four category counts in a bar plot.

4.6. **Qualitative inspection:** For each of the four categories above, randomly sample up to 20 examples (or all, if fewer than 20). For these examples:

- Inspect the model's reasoning steps.
- Identify any patterns in the questions or topics: e.g., what kinds of questions tend to *flip to correct*?

- For **Flip-to-incorrect** cases: does the reasoning look plausible but misleading? nonsensical?
- For **Incorrect–Incorrect**: is the model genuinely reasoning (but wrong), or just explaining its wrong guess?
- For **Correct–Correct**: is the CoT adding useful justification, or does it seem redundant? In your opinion, is it worth generating CoT in such cases?

4.7. **Analysis & Discussion:** Summarize your findings in a short written analysis covering:

- Whether CoT improved overall performance for your chosen model.
- How often CoT helped (Flip-to-correct) vs. hurt (Flip-to-incorrect).
- Any systematic patterns you observed in the qualitative examples.
- Your thoughts on when CoT represents genuine reasoning vs. a post-hoc explanation.

Further reading can be found in [2].

References

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

[2] Samuel Lewis-Lim et al. "Analysing Chain of Thought Dynamics: Active Guidance or Unfaithful Post-hoc Rationalisation?" In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing* (2025), pp. 29826–29841.