# Homework 2: Sarcasm Detection (Part 2)

**CS 490**: Natural Language Processing · **Spring 2026**

**Due on**: 02/27/2026 @ 11:59 PM (AoE)

## 1 Overview

In Part 1, you implemented a Multi-Layer Perceptron (MLP) with static embeddings (word2vec) to detect sarcasm. In this assignment (Part 2), you will upgrade your pipeline to use **transformers**, specifically fine-tuning a pre-trained **BERT** model.

The goal remains to classify headlines as either *sarcastic* (1) or *not sarcastic* (0). You are provided with skeleton code `hw2.py`. You must implement the functionality for all blocks marked with `TODO`. **Your model must achieve higher than 80% accuracy on a hidden test set to receive credit.**

In addition to the implementation, you are required to submit a **typed** report containing your analysis and answers to the conceptual questions. Make sure there is **only one question per page** — this includes any **sub-questions**. If you **do not separate questions onto their own pages**, points will be deducted.

## 2 Dataset

You are provided with two files: `train.jsonl` and `valid.jsonl`. Each line in these files is a valid JSON object containing:

- `"headline"`: The text of the news headline (string).
- `"is_sarcastic"`: The label (int), where 1 is sarcastic and 0 is not.

## 3 Allowed Packages

You are required to use Python version `3.10.x` for this assignment. Only the following packages (and their dependencies) are allowed:

- `nltk`
- `numpy`
- `tqdm`
- `torch==2.6.0`
- `gensim==4.4.0`
- `transformers==4.56.2`

# 4 Programming Section (50 pts)

## Task 1: Data Loading (0 pts)

**Function:** `get_data(path: str) -> List[Dict]`

This is the same function from Homework 1. The function should read a `.jsonl` file line-by-line and return a list of dictionaries.

## Task 2: The Dataset Class (10 pts)

**Class:** `SarcasmDataset`

Unlike the manual feature engineering in Part 1, BERT requires specific input formatting (input IDs and an attention mask). You must implement a class that extends the PyTorch `Dataset` class.

**Method:** `__getitem__(self, index)`

1. Retrieve the text and label for the example with the given `index`.
2. Use the provided tokenizer (passed in `__init__`) to encode the text using the `encode_plus` function.
3. You **must** enable padding and truncation to the `max_length` on the **right** side. More precisely, if the input length is less than `max_length`, you must add padding tokens on the right side of the sequence. If the input length is greater than `max_length`, you must remove tokens from the right side of the sequence until it has length exactly equal to `max_length`.
4. Return a dictionary containing three key-value entries. The keys are the strings `'input_ids'`, `'attention_mask'`, and `'label'`. Each value is a **flattened** PyTorch tensor which is described as follows:

   `input_ids`
   (dtype: `int`, size: $(max\_length,)$)
   The sequence of token indices for the input text.

   `attention_mask`
   (dtype: `int`, size: $(max\_length,)$)
   Binary mask indicating non-padding (`1`) vs. padding (`0`) tokens.

   `label`
   (dtype: `int`, size: $(1,)$)
   The target class index for the classification task.

## Task 3: Model Architecture (10 pts)

**Class:** `SarcasmBERT`

Implement a class that wraps a pre-trained BERT model for binary classification.

### 3.1 Initialization

**Method:** `__init__(self)`

1. Load the pre-trained `bert-base-uncased` model using `BertModel.from_pretrained`.
2. Initialize a linear classification layer that maps the BERT hidden size to the number of classes.

### 3.2 Forward Pass

**Method:** `forward(self, input_ids, attention_mask)`

1. Pass the `input_ids` and `attention_mask` through the BERT model.
2. Extract the representation of the `[CLS]` token from the last hidden state.
3. Pass this `[CLS]` vector through your linear classifier to return logits.

## Task 4: Training Loop (15 pts)

**Function:** `train_loop(model, dataloader, device, lr, epochs)`
Implement the training loop for fine-tuning:

1. Use `torch.optim.AdamW` as the optimizer.
2. Move all batch tensors (`input_ids`, `mask`, `labels`) to the specified `device`.
3. Compute the cross-entropy loss and perform backpropagation.

## Task 5: Experimental Analysis (15 pts)

Compare the performance of your **Part 1** model (using all three different embeddings) against your **Part 2** model. You may want to create a copy of your `Homework 2` code to use as a sandbox for this analysis.

1. **Preparation** Ensure your `Homework 1` implementation is modified to support **batch updates** before starting.
2. **Loss Curve Plotting** Generate a plot showing the *loss curve* for each gradient update step for these models. You may display the results either as four curves overlaid on a single set of axes or as four separate plots arranged side-by-side. In either format, ensure that all four models are represented. **If you choose the side-by-side format, all plots must use identical x- and y-axis limits and scaling so that the curves are directly comparable.**
   - **Fixed Constraints:** To ensure a fair comparison, the following must be identical:
     - The order of training data (shuffle seeds)
     - The batch size
   - **Variables:** You are permitted to use different learning rates ($\eta$) and a different number of epochs for each model.
3. **Result Analysis** Provide explanations of your observations regarding the loss curve comparisons. What do the convergence rates or fluctuations tell you about the two models?
4. **Conceptual Understanding** Briefly explain the architectural advantages of **BERT** over **word2vec** embedding specifically in the context of *sarcasm detection.*

# 5 Conceptual Questions (50 pts)

Answer the following questions in your report. Keep your answers concise.

### Q1: Contextual Embeddings (10 pts)

In Homework 1, we used word2vec, which is a *static* embedding (i.e., each word is mapped to a single vector, regardless of the other words in the surrounding context), BERT on the other side produces *contextual* embeddings (i.e., the embedding of each word can vary depending on the surrounding context).

Explain how the word "bank" has two different meanings in the following sentences: "I went to the bank to deposit money" and "I sat on the river bank." Then describe how word2vec and BERT would represent the word "bank" differently in each sentence. For each sentence, give one example word that would likely be closest in embedding space to "bank" when using word2vec, and one example word that would likely be closest to "bank" when using BERT.

### Q2: The [CLS] Token (15 pts)

In Task 3, you extracted the embedding of the `[CLS]` token to perform classification.

1. What is the purpose of the `[CLS]` token in BERT's pre-training objective?
2. Why do we typically use this specific token for sentence-level classification tasks instead of averaging all token embeddings (like we did with word2vec)?

### Q3: Tokenization (10 pts)

BERT uses `WordPiece` tokenization, which breaks words into subwords (e.g., "playing" → "play" + "##ing").
Describe one major advantage of subword tokenization over the whole-word tokenization approach used in Homework 1, specifically regarding vocabulary size and unknown words (`<UNK>`).

### Q4: Fine-Tuning vs. Feature Extraction (15 pts)

In this assignment, we are fine-tuning the entire BERT model (updating all weights).
Alternatively, we could have "frozen" the BERT weights and only trained the final linear layer.

1. Which approach (full fine-tuning vs fine-tuning only the last linear layer) typically yields higher accuracy for specific downstream tasks?
2. Which approach is computationally cheaper during the training phase? Explain why.

# 6 Evaluation and Submission

- **Code:** Submit your `hw2.py` as well as `checkpoint.pt` on Gradescope under `Homework 2 - Programming`. You will need to use GitHub for this submission. For your reference: GitLFS and Submission Guide
- **Report:** Submit a typed PDF containing your analysis and conceptual answers under `Homework 2 - Conceptual`.
- **Runtime:** Ensure your training converges **within 40 minutes**. Submissions exceeding this limit will receive zero points.