Homework 1: Sarcasm Detection (Part 1)

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

Due on: 02/12/2026 @ 11:59 PM (AoE)

1 Overview

In this assignment, you will build a complete machine learning pipeline to detect sarcasm in news headlines. Your task involves implementing feature extraction techniques from scratch and constructing a flexible Multi-Layer Perceptron (MLP) using PyTorch.

The goal is to classify headlines as either *sarcastic* (1) or *not sarcastic* (0). You are provided with a boilerplate file `hw1.py`. You must implement the functionality for all blocks marked with `TODO` and make sure the code can execute successfully when using `python3 hw1.py`. **Your model must achieve higher than 70% accuracy on a hidden test set to receive credit.**

In addition to the implementation, you are required to conduct an analysis to find optimal hyperparameters. You must submit a **typed** report containing your plots and answers to the conceptual questions.

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 (as well as its dependencies) are allowed. If there is not a specified version, you can use the most up-to-date version of that package.

- `nltk`
- `numpy`
- `torch==2.6.0`
- `gensim==4.4.0`

4 Programming Section (55 pts)

Task 1: Data Loading (5 pts)

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

Implement a function that reads a `.jsonl` file line-by-line. It should parse each line as a JSON object and return a list of dictionaries. A usage example is provided in the function docstring.

Task 2: Feature Engineering (20 pts)

Class: `TextFeaturizer`

Neural networks require numerical input. You must transform raw text into numerical vectors using three different encoding strategies, described below.

2.1 Vocabulary Building (5 pts)

Method: `build_vocab(self, corpus: List[str])`

Iterate through the entire training corpus to construct the vocabulary.

1. Tokenize the text (a simple regex tokenizer is provided, you can also use `nltk` if you want).
2. Identify all unique words in the corpus.
3. Assign a unique integer index to each word starting from 1.
4. Reserve index 0 for the `<UNK>` (Unknown) token.

2.2 One-Hot Encoding (5 pts)

Method: `to_one_hot(self, text: str) -> np.ndarray`

Represent the text as a binary vector $v \in \{0,1\}^{|V|}$, where $|V|$ is the vocabulary size.

$$v_i = \begin{cases} 1 & \text{if word } i \text{ is present in the text} \\ 0 & \text{otherwise} \end{cases}$$

2.3 Bag-of-Words (BoW) (5 pts)

Method: `to_bow(self, text: str) -> np.ndarray`

Represent the text as a count vector $v \in \mathbb{R}^{|V|}$.

$$v_i = \text{count of word } i \text{ in the text}$$

2.4 Average Word2Vec (5 pts)

Method: `to_word2vec(self, text: str) -> np.ndarray`

Represent the text as the average of its constituent word embeddings. Given a sentence with N tokens $w_1, w_2, ..., w_N$, and a pre-trained embedding matrix E:

$$v = \frac{1}{N} \sum_{j=1}^{N} E(w_j)$$

Note: Pre-trained embeddings are loaded in the `__init__` method. You need to download the embedding from **here** and put it locally at your working directory. If a word is not found in the pre-trained dictionary, you may skip it. If no words in the sentence have embeddings, return a zero vector. You do not need to submit the downloaded word embedding to Gradescope.

Task 3: Model Architecture (10 pts)

Class: `SarcasmMLP`

Implement a flexible Feed-Forward Neural Network (MLP) using PyTorch.

3.1 Initialization (5 pts)

Method: `__init__(self, input_size, hidden_sizes, output_size)`
Your model must dynamically construct layers based on the `hidden_sizes` list.

- For example, if `hidden_sizes = [64, 32]`, the network structure should be:

$$\text{Input} \rightarrow \text{Linear}(D_{in}, 64) \rightarrow \text{ReLU} \rightarrow \text{Linear}(64, 32) \rightarrow \text{ReLU} \rightarrow \text{Linear}(32, D_{out})$$

- Use `nn.ModuleList` or `nn.Sequential` to store the layers.

3.2 Forward Pass (5 pts)

Method: `forward(self, x:  torch.Tensor)`
Pass the input tensor x through the defined layers and activation functions, returning the final raw logits (before Softmax).

Task 4: Training Loop (10 pts)

Function: `train_loop(model, X, y, lr, epochs)`
Implement the standard PyTorch training loop:

1. Convert numpy arrays X and y to PyTorch Tensors.
2. Use `CrossEntropyLoss` as the criterion.
3. Use `Adam` as the optimizer.
4. For each epoch:
    - Perform the forward pass.
    - Compute the loss.
    - Perform backpropagation (`backward`) and update weights (`step`).
    - Calculate and store the accuracy for that epoch.

Task 5: Experimental Analysis (10 pts)

You are required to identify and experiment with different hyperparameter values available in your implementation and plot the loss curves to compare approaches. In addition, also explain your findings in few sentences.

Requirements:

- Analyze **at least four** different hyperparameter configurations.
- Examples include: changing of the learning rate, number of layers, hidden layer dimensions, or comparing feature encoding methods (one-hot vs BoW vs word2vec).
- Generate plots showing *Training Loss vs. Epochs* for your comparisons.

5 Conceptual Questions (40 pts)

Answer the following questions in your report to demonstrate your understanding of the underlying concepts. Your answer should only consists a few short sentences.

Q1: Backpropagation (10 pts)

Consider a simple single-neuron model with input x, weight w, bias b, and identity activation function ($f(z) = z$). The prediction is $\hat{y} = w \cdot x + b$. We use the Squared Error loss function $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$.
Derive the partial derivative of the loss with respect to the weight w (i.e., $\frac{\partial L}{\partial w}$). Show your steps using the Chain Rule.

Q2: Loss Functions (10 pts)

In this assignment, we use **cross-entropy (CE)** loss rather than **Mean Squared Error (MSE)**.

1. Explain the probabilistic interpretation of cross-entropy loss in binary classification.
2. Why is MSE generally less suitable for classification tasks with sigmoid/softmax outputs compared to CE?

Q3: Feature Representation (15 pts)

Compare **One-Hot Encoding**, **Bag-of-Words (BoW)**, and **Word2Vec** based on the following criteria:

1. **Sparsity:** Which vectors are sparse and which are dense?
2. **Vocabulary Size:** How does the dimensionality of the feature vector scale with vocabulary size for each method?
3. **Semantic Meaning:** Which method captures semantic similarity between words (e.g., “dog” is close to “cat”)? Explain why.

Q4: Limitations (5 pts)

Briefly describe one major limitation of the **Average Word2Vec** approach for representing full sentences. (Hint: Consider the sentence “I like bananas but not apples” vs “I like apples but not bananas”).

6 Evaluation and Submission

- **Code:** Submit your `hw1.py` on Gradescope under `Homework 1 - Programming` with the optimal hyperparameter values. The performance will be checked against a hidden test split.
- **Report:** Submit a typed PDF containing your plots from Task 5 and answers to the Conceptual Questions under `Homework 1 - Conceptual`.
- **Runtime:** Ensure your training converges **within 40 minutes**. Submissions exceeding this limit will receive zero points.