CS203 Assignment 7

Nikhil Goyal 23110218
Aayush Bundel 23110005

Python codes and related material for the assignment are available here.

1. Dataset Preparation (10%)

- Load the training dataset and test data (Dataset 1). [UPDATED DATASET]
- Also, the IMDB dataset (Dataset 2) can be used for continual learning.
- Use 20% of the training dataset as the validation set.

The datasets were loaded and read into Pandas dataframes with columns named “sentence” and “label.” The labels for IMDB dataset were changed to match those for Dataset 1

2. Construct a **Multi-Layer Perceptron (MLP) model**. **(20%)**

- The parameter should be with:
  - hidden_sizes=[512, 256, 128, 64]
  - Output should have two labels.
  - With the following architecture:

```
Model Summary:
================================================
Input Size: 10000
Layer 1: Linear(10000, 512)
  Parameters: 5,120,512
  Activation: ReLU + Dropout(0.3)
Layer 2: Linear(512, 256)
  Parameters: 131,328
  Activation: ReLU + Dropout(0.3)
Layer 3: Linear(256, 128)
  Parameters: 32,896
  Activation: ReLU + Dropout(0.3)
Layer 4: Linear(128, 64)
  Parameters: 8,256
  Activation: ReLU + Dropout(0.3)
Layer 5: Linear(64, 2)
```

- Count the number of trainable parameters in the model using the automated function.

The model was defined under the MLPModel class with the following code snippet:

```python
class MLPModel(nn.Module):
    def __init__(self, input_size = 10000, hidden_sizes = [512, 256, 128, 64], output_size = 2, dropout_rate = 0.3):
        super(MLPModel, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_size, hidden_sizes[0]),
            nn.ReLU(),
            nn.Dropout(dropout_rate),

            nn.Linear(hidden_sizes[0], hidden_sizes[1]),
            nn.ReLU(),
            nn.Dropout(dropout_rate),

            nn.Linear(hidden_sizes[1], hidden_sizes[2]),
            nn.ReLU(),
            nn.Dropout(dropout_rate),

            nn.Linear(hidden_sizes[2], hidden_sizes[3]),
            nn.ReLU(),
            nn.Dropout(dropout_rate),

            nn.Linear(hidden_sizes[3], output_size)
        )

    def forward(self, x):
        return self.layers(x)
```

The model summary and total number of trainable parameters can be seen below:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 512]       5,120,512
              ReLU-2               [-1, 1, 512]               0
           Dropout-3               [-1, 1, 512]               0
            Linear-4               [-1, 1, 256]         131,328
              ReLU-5               [-1, 1, 256]               0
           Dropout-6               [-1, 1, 256]               0
            Linear-7               [-1, 1, 128]          32,896
              ReLU-8               [-1, 1, 128]               0
           Dropout-9               [-1, 1, 128]               0
           Linear-10                [-1, 1, 64]           8,256
             ReLU-11                [-1, 1, 64]               0
          Dropout-12                [-1, 1, 64]               0
           Linear-13                 [-1, 1, 2]             130
================================================================
Total params: 5,293,122
Trainable params: 5,293,122
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.04
Forward/backward pass size (MB): 0.02
Params size (MB): 20.19
Estimated Total Size (MB): 20.25
----------------------------------------------------------------
```

```
[ ] print(f"The total number of parameters in the model is {sum(p.numel() for p in model.parameters())}")

The total number of parameters in the model is 5293122
```

3. Implement case 1: Bag-of-words (20%)

- Implement the bag-of-words (max_features=10000).
- Hint: from sklearn.feature_extraction.text import CountVectorizer

```
vectorizer = CountVectorizer(max_features = 10000)

# Fit and transform the training dataset
X_train_bow = vectorizer.fit_transform(train_data["sentence"])
X_val_bow = vectorizer.transform(val_data["sentence"])
X_test_bow = vectorizer.transform(test_df["sentence"])

# Get the vocabulary (mapping of words to index positions)
vocab = vectorizer.get_feature_names_out()

# Print some details
print("Vocabulary Size:", len(vocab))
print("Sample Bag-of-Words Representation (Training Set):\n", X_train_bow.toarray())
```

These numpy arrays were then converted into Torch Tensors and then the obtained Tensors were used to create DataLoaders, which were used to train the pre-defined architecture.

4. Implement case 2: Construct a function to use embeddings on the same model. (20%)

- Use the model: meta-llama/Llama-3.1-8B or use bert-base-uncased if facing issues with the GPU constraints.
  TIPS:
  You can use the distilled version, gather embeddings for 200 samples, and even reduce the precision to deal with computing issues!
- Hints:

```
self.tokenizer = AutoTokenizer.from_pretrained(model_name)
self.model = AutoModel.from_pretrained(model_name).to(device)
self.embedding_size = self.model.config.hidden_size
self.model_loaded = True
```

USE BOTH CASES IN PARALLEL TO EACH OTHER (ONE WITH BOW AND ANOTHER WITH EMBEDDINGS). NOT ON TOP OF EACH OTHER.

```python
class EmbeddingExtractor:
    def __init__(self, model_name = "google-bert/bert-base-uncased", device = "cuda"):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModel.from_pretrained(model_name).to(device)
        self.embedding_size = self.model.config.hidden_size
        self.device = device
        self.model_loaded = True

    def get_embeddings_batch(self, texts, batch_size=16):
        all_embeddings = []
        # Process the texts in batches
        for i in range(0, len(texts), batch_size):
            batch_texts = texts[i : i + batch_size]
            inputs = self.tokenizer(
                batch_texts,
                return_tensors="pt",
                padding=True,
                truncation=True,
                max_length=512
            )

            inputs = {key: val.to(self.device) for key, val in inputs.items()}
            with torch.no_grad():
                outputs = self.model(**inputs)

            batch_embeddings = outputs.last_hidden_state.mean(dim=1)
            all_embeddings.append(batch_embeddings.cpu().numpy())
        return np.concatenate(all_embeddings, axis=0)
```

5. Train the model with 10 epochs and create the best-performing model (checkpoint.pt) on the Dataset 1. (10%)

- Get the validation accuracy.

6. Use the checkpoint from before and train on the IMDB dataset (Dataset 2). (10%)

- Use the following parameters:
  criterion = nn.CrossEntropyLoss()
  optimizer = optim.Adam(model.parameters(), lr=0.0001) # Smaller learning rate

We used a smaller learning rate for continual training on IMDB(compared to the learning rate that we used for training on Dataset 1)

8. Compute the validation loss and accuracy on the validation set of the Dataset 1 and IMDB dataset. (10%)

Validation Accuracy →

Dataset 1: 87.86%

IMDB: 80.77%

Validation Loss →

Dataset 1: 0.3014

IMDB:

9. Submission Requirements

- **Python code** for training, testing, and evaluation.
- **Screenshots of the following** displaying:
  - Model architecture.
  - Hyperparameters.
  - Logged metrics.
  - Final evaluation results.
  - Confusion matrix visualization.
  - Training and validation loss curves.

Evaluation Criteria

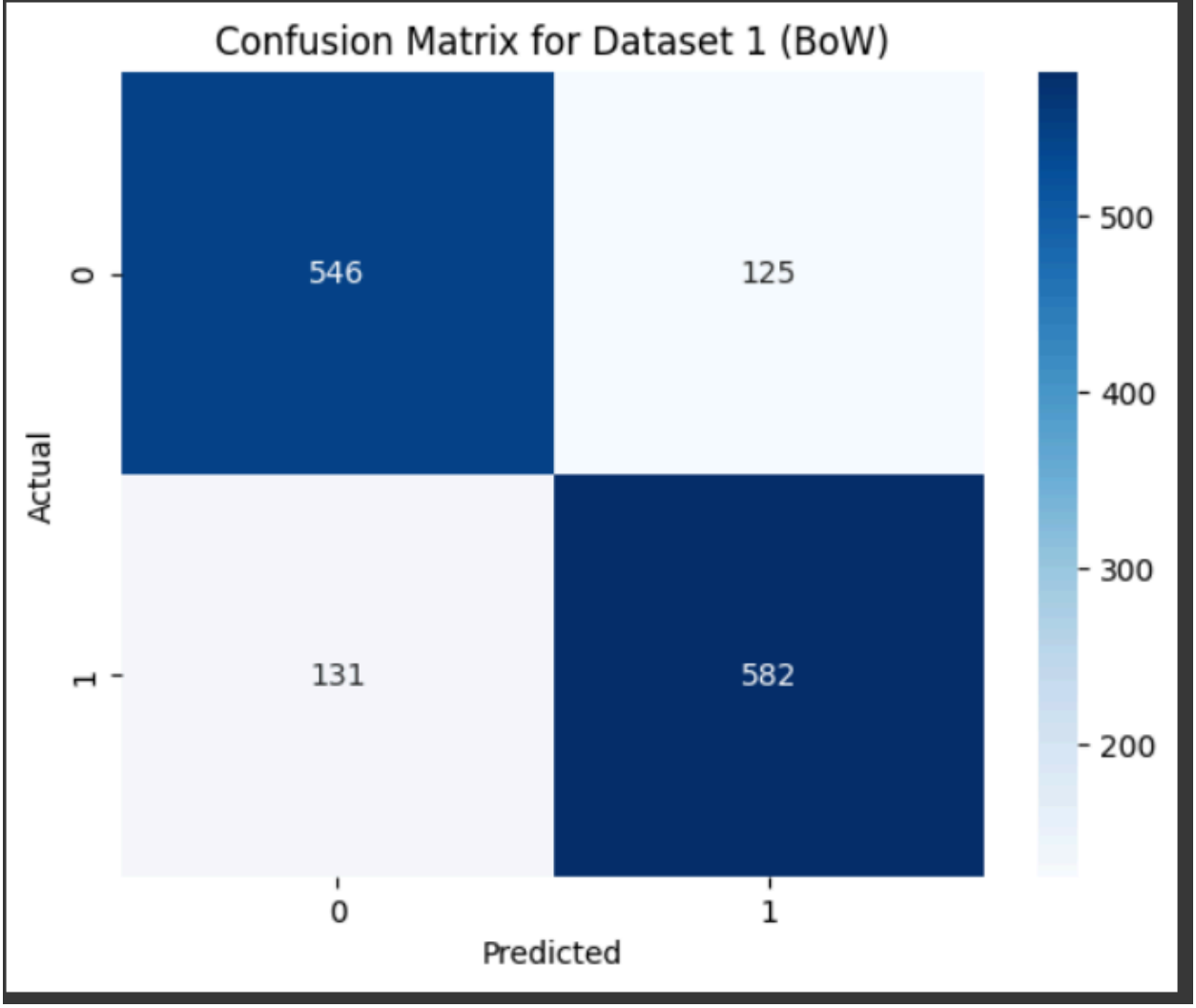
- Implement resume training from checkpoint → done!
- Add model parameter logging → done!
- Implement checkpoint compression → done!
- Add TensorBoard integration → done!
- Model architecture. → done!

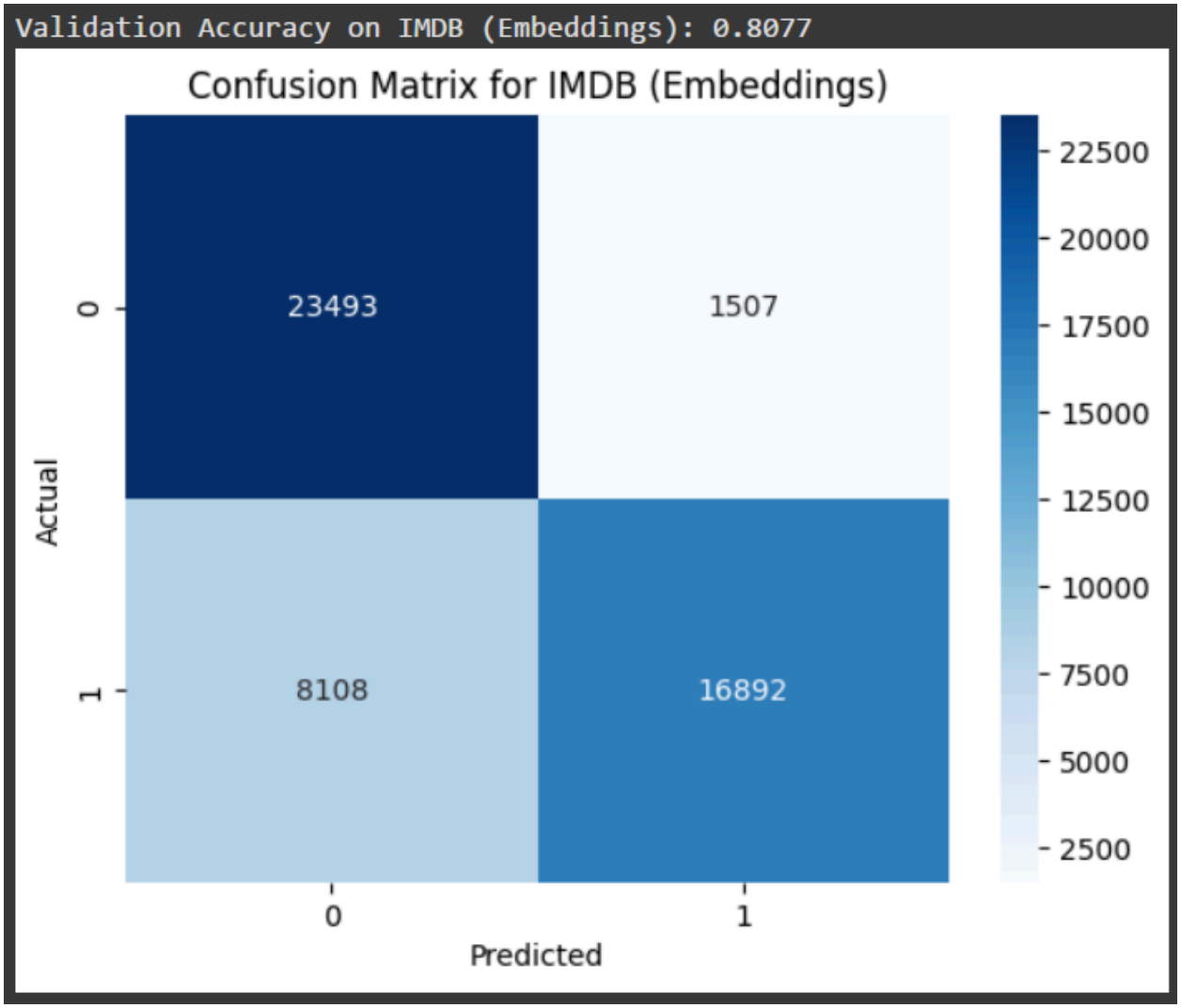
- Hyperparameters → done!
- Logged metrics. → done!
- Final evaluation results. → done!

We get an accuracy of ~81% on the IMDB dataset, and we achieve better performance if we use mean-based embeddings instead of using BoW (probably because of the fact that the BoW method does not consider the relative ordering in the sentence.).

- Confusion matrix visualization.

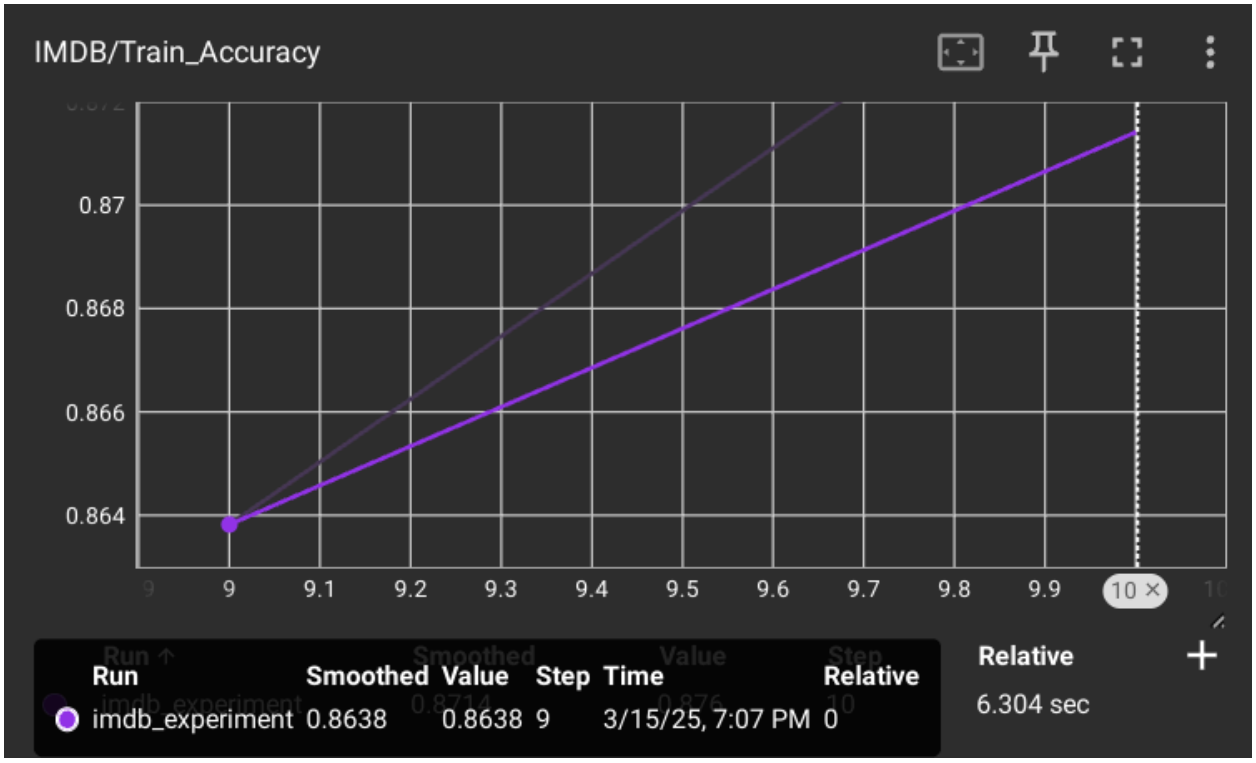
```
Validation Accuracy on Dataset 1 (BoW): 0.8150
```





- Training and validation loss curves.

These curves have been logged to TensorBoard, for example:


IMDB/Train_Accuracy
0.87
0.868
0.866
0.864
9
9.1
9.2
9.3
9.4
9.5
9.6
9.7
9.8
9.9
10
Run
Smoothed
Value
Step
Time
Relative
imdb_experiment
0.8638
0.8638
9
3/15/25, 7:07 PM
0
Relative
6.304 sec