GROUP 28

Laksh Jain – 23110185 | Tanish Yelgoe – 23110328

Github: [Jain-Laksh/CS203-Lab-07]

LAB-07 REPORT

1] Loading the initial dataset into train, test and validation

```
#Load the dataset into train,test and validation
train = pd.read_csv('train.tsv', sep='\t',header=None)
test = pd.read_csv('test.tsv', sep='\t',header=None)

#Divide the train set into train and validation set
train, validation = train_test_split(train, test_size=0.2, random_state=42)
```

2] Initializing the models

A] MLP for Bag of Words (input dimension = 10000)

```
mlp1 = nn.Sequential(
    nn.Linear(10000, 512),
    nn.ReLU(),
    nn.Dropout(0.3),

    nn.Linear(512, 256),
    nn.ReLU(),
    nn.Dropout(0.3),

    nn.Linear(256, 128),
    nn.ReLU(),
    nn.Dropout(0.3),

    nn.Linear(128, 64),
    nn.ReLU(),
    nn.Dropout(0.3),

    nn.Linear(64, 2)  # No activation for final layer (assumes logits for
classification)
)

print(mlp1)
```

```
Sequential(
  (0): Linear(in_features=10000, out_features=512, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.3, inplace=False)
  (3): Linear(in_features=512, out_features=256, bias=True)
  (4): ReLU()
  (5): Dropout(p=0.3, inplace=False)
  (6): Linear(in_features=256, out_features=128, bias=True)
  (7): ReLU()
  (8): Dropout(p=0.3, inplace=False)
  (9): Linear(in_features=128, out_features=64, bias=True)
  (10): ReLU()
  (11): Dropout(p=0.3, inplace=False)
  (12): Linear(in_features=64, out_features=2, bias=True)
)
```

Printing the number of trainable parameters:

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

# Print the number of trainable parameters
print(f"Total trainable parameters: {count_parameters(mlp1)}")
```

Total trainable parameters: 5293122

B] MLP for training using the embeddings (input dimension = embeddings dimension = 762)

```
mlp2 = nn.Sequential(
    nn.Linear(768, 512),nn.ReLU(),nn.Dropout(0.3), # Model input dimension is
768 to match embedings dimension
    nn.Linear(512, 256),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(256, 128),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(128, 64),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(64, 2)  # No activation for final layer (assumes logits for
classification)
)
```

3A] Converting sentences into bag or words vectors (10000 features) using CountVectorizer:

```
vectorizer = CountVectorizer(max_features=10000)
X_train = vectorizer.fit_transform(train_corpus)
X_val = vectorizer.transform(validation_corpus)
```

```
vectorizer.get_feature_names_out().size
```

10000

3B] Converting sentences into embedding vectors

```
def get_bert_embeddings(text_corpus, tokenizer, model, batch_size=32,
strategy="mean"):
    embeddings_list = []

    for i in range(0, len(text_corpus), batch_size):
        batch = text_corpus[i : i + batch_size]

        inputs = tokenizer(batch, padding=True, truncation=True,
return_tensors="pt").to(device)

        with torch.no_grad():
            outputs = model(**inputs)

        embeddings = outputs.last_hidden_state  # Shape: (batch_size, seq_len,
768)

        if strategy == "mean":
            batch_embeddings = torch.mean(embeddings, dim=1)  # Shape:
(batch_size, 768)
            embeddings_list.append(batch_embeddings)

    return torch.cat(embeddings_list, dim=0)  # Shape: (num_samples, 768)
```

Here, we take the output of the last hidden layer of the bert-uncased model. Since the output dimension depends on the sequence length, we take the mean of the vectors across the sequence length dimension to get a final vector with 762 feature.

4] Hyperparameters for the training for both cases and checkpointing

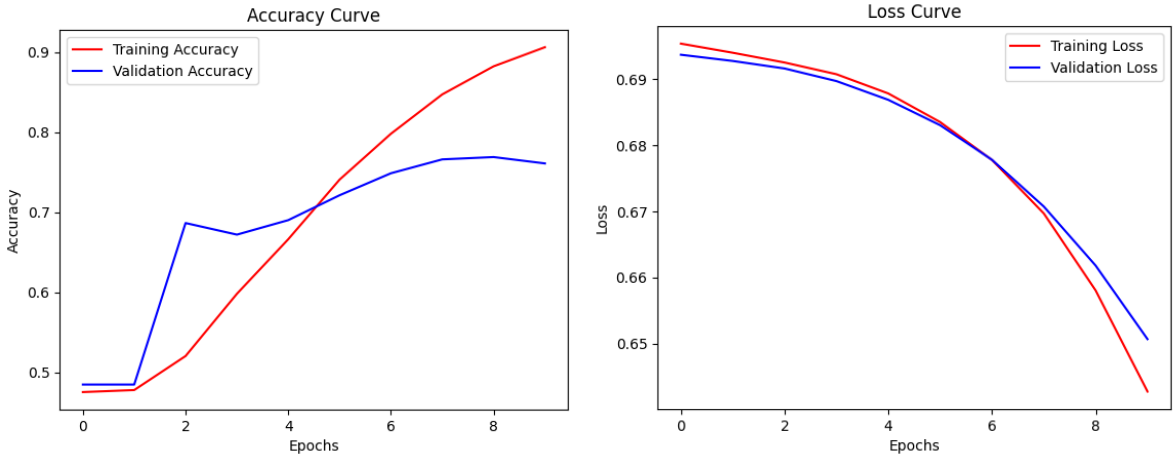
```
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(mlp.parameters(), lr=0.001)
n_epochs = 10
```

```
# Save the best model based on validation accuracy
    if val_accuracy > best_val_acc:
        best_val_acc = val_accuracy
        torch.save(mlp2.state_dict(), best_model_path)
        print("Best model updated.")
```

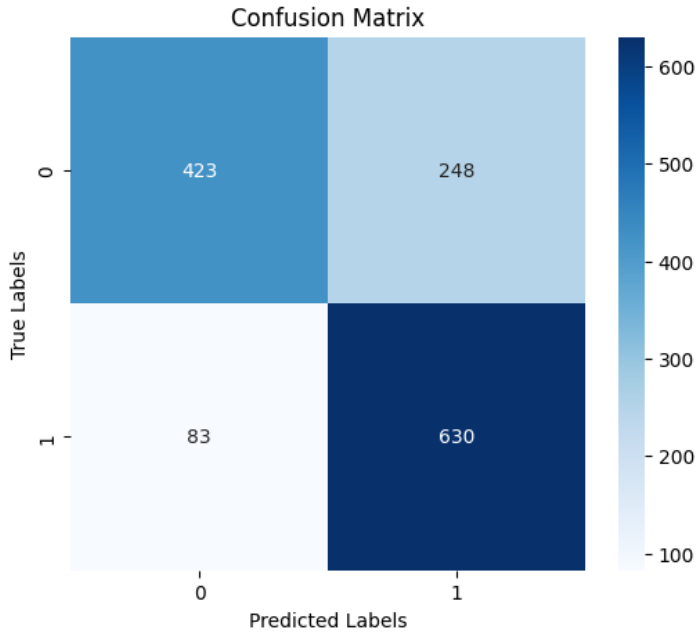
We save only the best model based on the validation accuracy. This ensures we don't consume much space. Also, we do not save the entire model, but only the model weights. When loading the checkpoints, we first initialize a model with same architecture and then load the saved weights. These measures are taken keeping in mind the space constraints. We can take further more measures such as saving in a zip file format, or other formats, and lower the precision so that less space is used.

5A] Final evaluation metrics for Bag of Words

```
FINAL EVALUATION METRICS FOR BOW
Train Loss: 0.6427776217460632 | Train Accuracy : 0.9058887283236994
Validation Loss: 0.6506825089454651 | Validation Accuracy : 0.7608381502890174
```

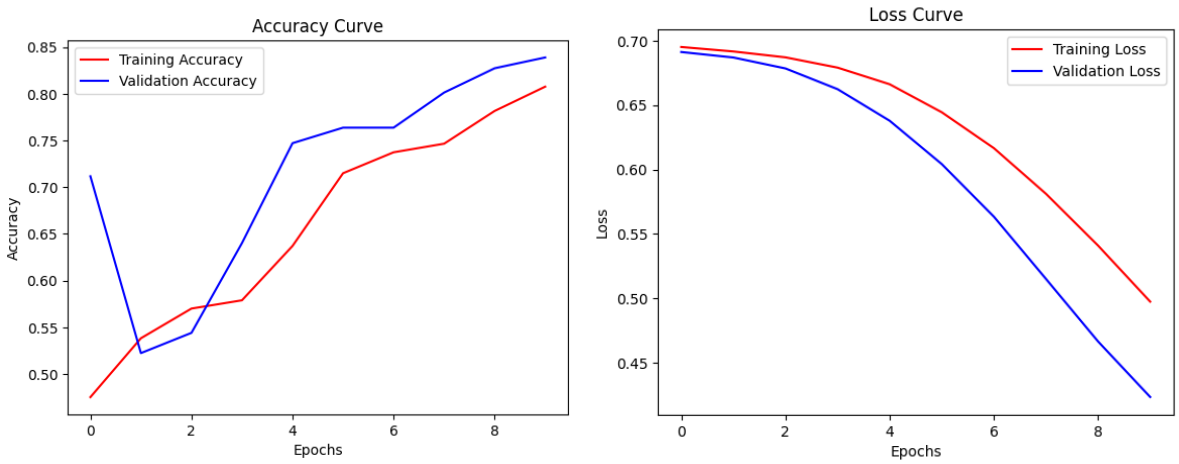


Confusion matrix on validation dataset

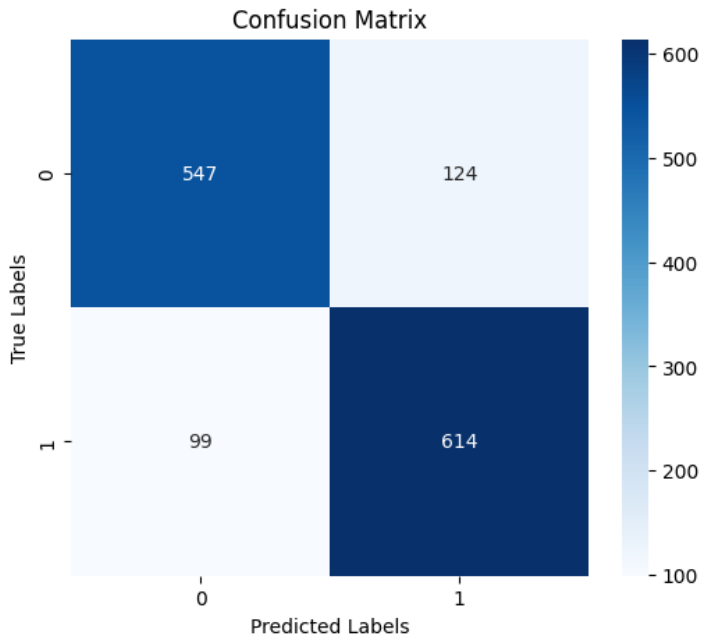


5B] Final evaluation metrics for Embeddings

```
FINAL EVALUATION METRICS FOR EMBEDDINGS
Train Loss: 0.49745723605155945 | Train Accuracy : 0.8076228323699421
Validation Loss: 0.4234275221824646 | Validation Accuracy : 0.8388728323699421
```



Confusion matrix on validation dataset



6] Load the IMDB Dataset

```
imdb_train_data, imdb_test_data = train_test_split(imdb_data, test_size=0.2,
random_state=42)
imdb_train_data, imdb_val_data = train_test_split(imdb_train_data,
test_size=0.2, random_state=42)
```

7] Load the models from the best checkpoints saved to resume training

A] Bag of Words

```
Load the best checkpoint
model_loaded_bow = nn.Sequential(
    nn.Linear(10000, 512),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(512, 256),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(256, 128),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(128, 64),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(64, 2)  # No activation for final layer (assumes logits for
classification)
)
model_loaded_bow.load_state_dict(torch.load("best_model_bow.pth"))
```

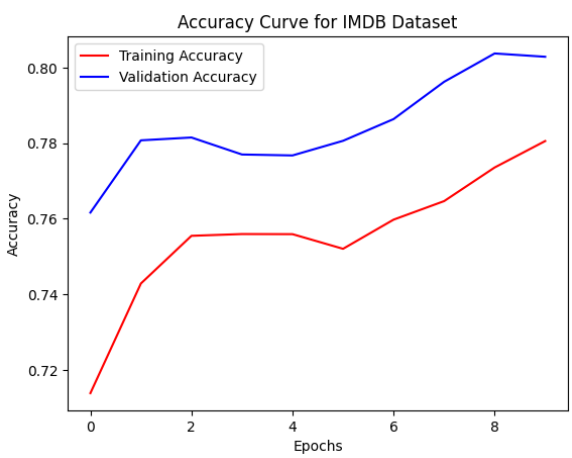
B] Embeddings

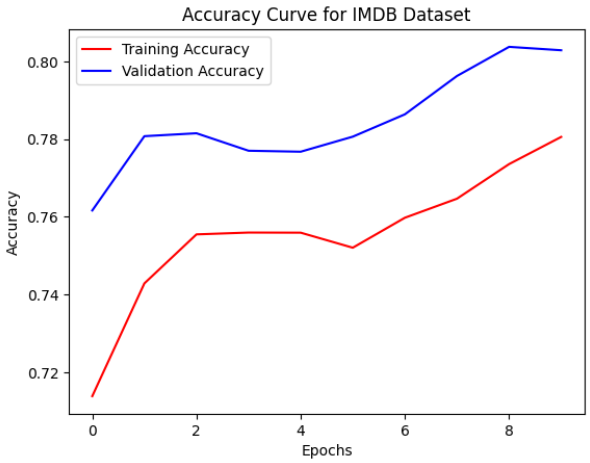
```
Load the best checkpoint
model_loaded_embeddings = nn.Sequential(
    nn.Linear(768, 512),nn.ReLU(),nn.Dropout(0.3), # input dimensions =
embeddings dimension = 768
    nn.Linear(512, 256),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(256, 128),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(128, 64),nn.ReLU(),nn.Dropout(0.3),
    nn.Linear(64, 2)  # No activation for final layer (assumes logits for
classification)
)
model_loaded_embeddings.load_state_dict(torch.load("best_model_embeddings.pth"
))
```

8] Final Evaluation metrics for:

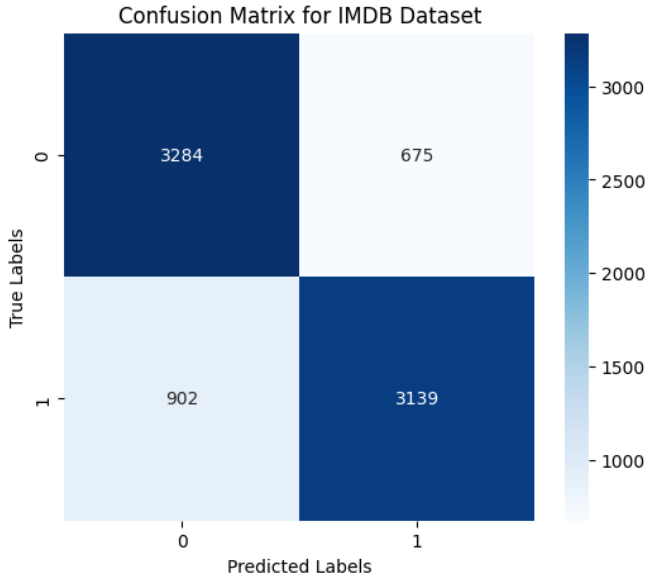
A] Resume training for bag of words on IMDB dataset

```
FINAL EVALUATION METRICS FOR BOW
Train Loss: 0.5579841732978821 | Train Accuracy : 0.7805625
Validation Loss: 0.5342373847961426 | Validation Accuracy : 0.802875
```



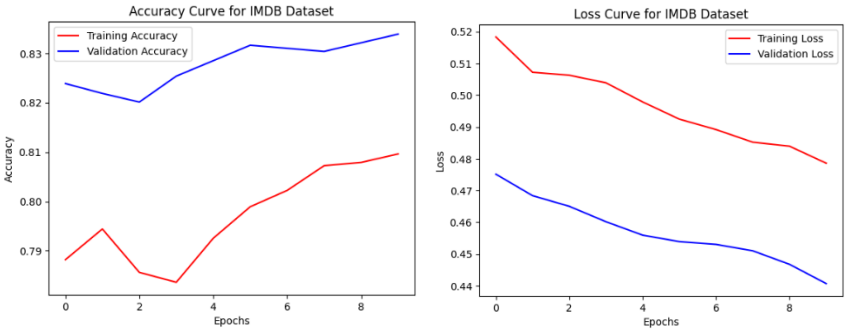


Confusion matrix on validation dataset of IMDB dataset

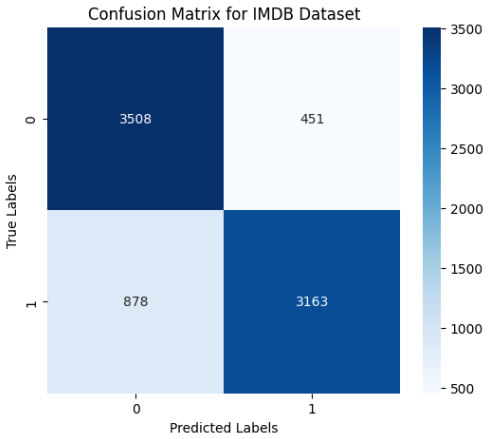


B] Resume training for embeddings on IMDB dataset

```
FINAL EVALUATION METRICS FOR EMBEDDINGS
Train Loss: 0.4785939157009125 | Train Accuracy : 0.809625
Validation Loss: 0.44071438908576965 | Validation Accuracy : 0.833875
```



Confusion matrix on validation dataset of IMDB dataset



9] TensorBoard Integration and Logged Metrics

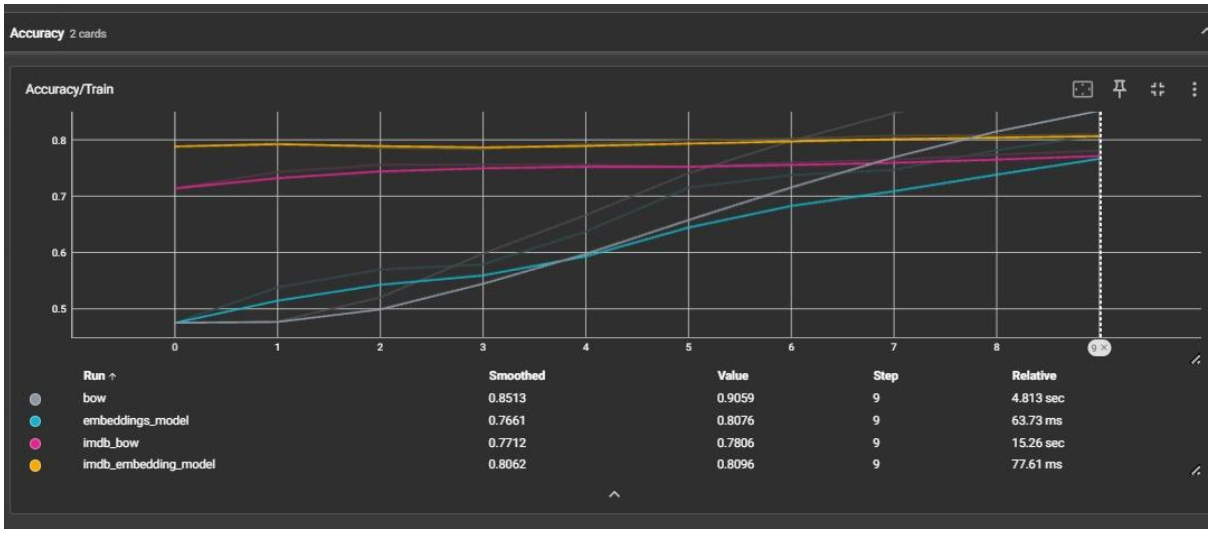
```
writer = SummaryWriter(log_dir="runs/bow")
 # Log to TensorBoard
    writer.add_scalar("Loss/Train", loss.item(), epoch)
    writer.add_scalar("Loss/Validation", loss_val.item(), epoch)
    writer.add_scalar("Accuracy/Train", train_accuracy, epoch)
    writer.add_scalar("Accuracy/Validation", val_accuracy, epoch)
writer.close()
```

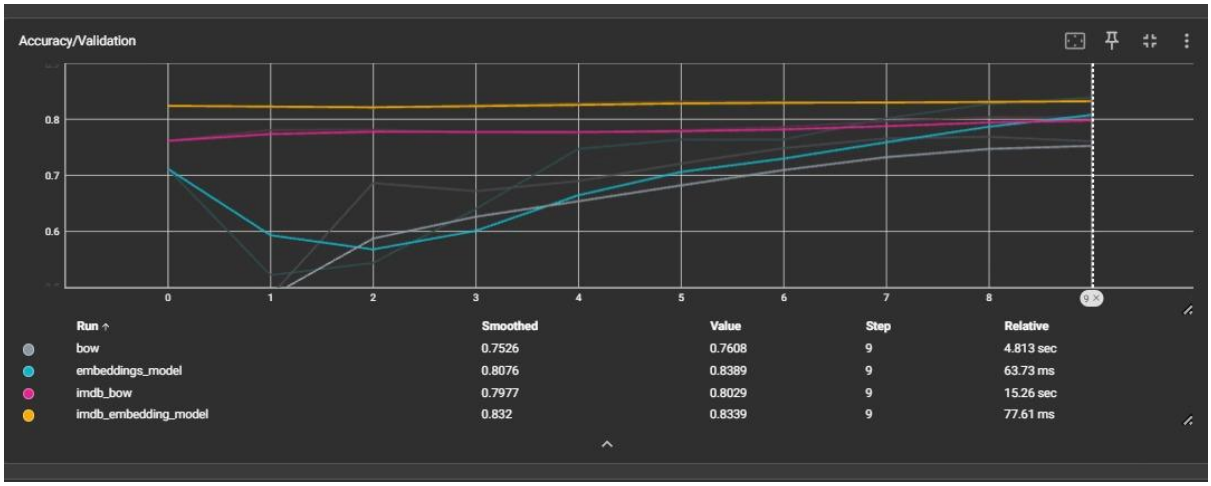
And similarly for the model based on embeddings

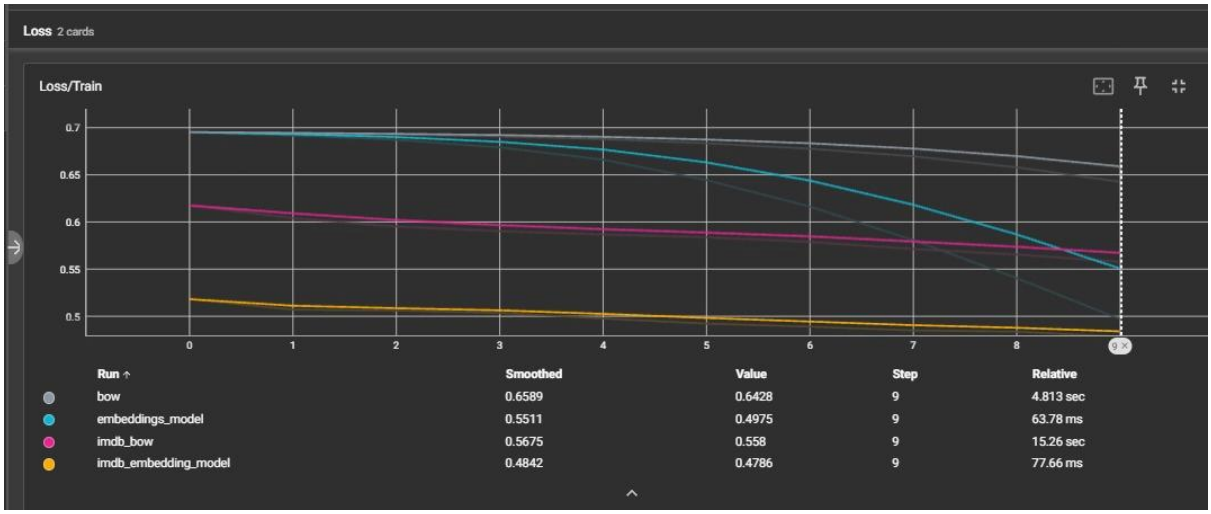
```
writer = SummaryWriter(log_dir="runs/embeddings_model")
# Log to TensorBoard
    writer.add_scalar("Loss/Train", loss.item(), epoch)
    writer.add_scalar("Loss/Validation", loss_val.item(), epoch)
    writer.add_scalar("Accuracy/Train", train_accuracy, epoch)
    writer.add_scalar("Accuracy/Validation", val_accuracy, epoch)
writer.close()
```

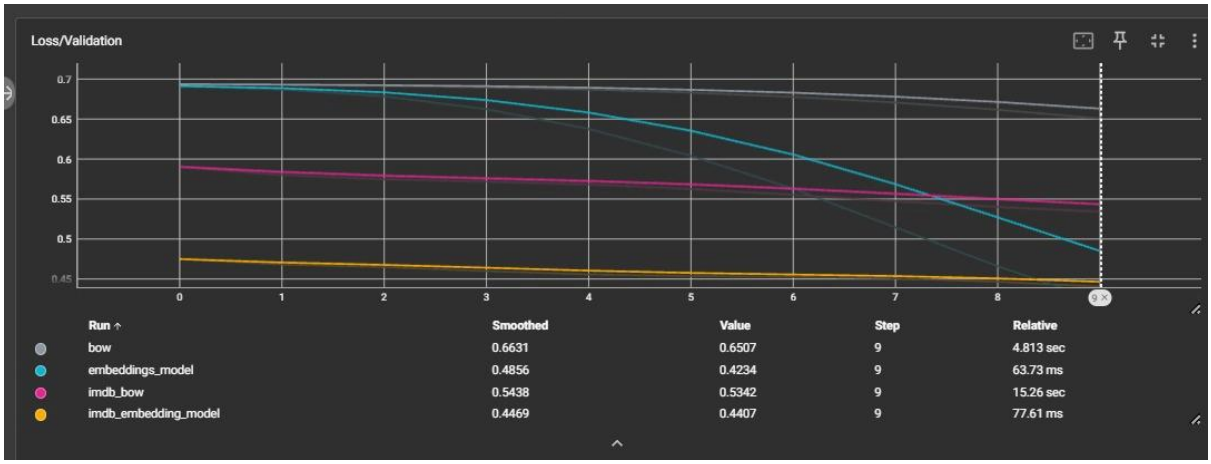
```
%reload_ext tensorboard
%load_ext tensorboard
%tensorboard --logdir runs
```

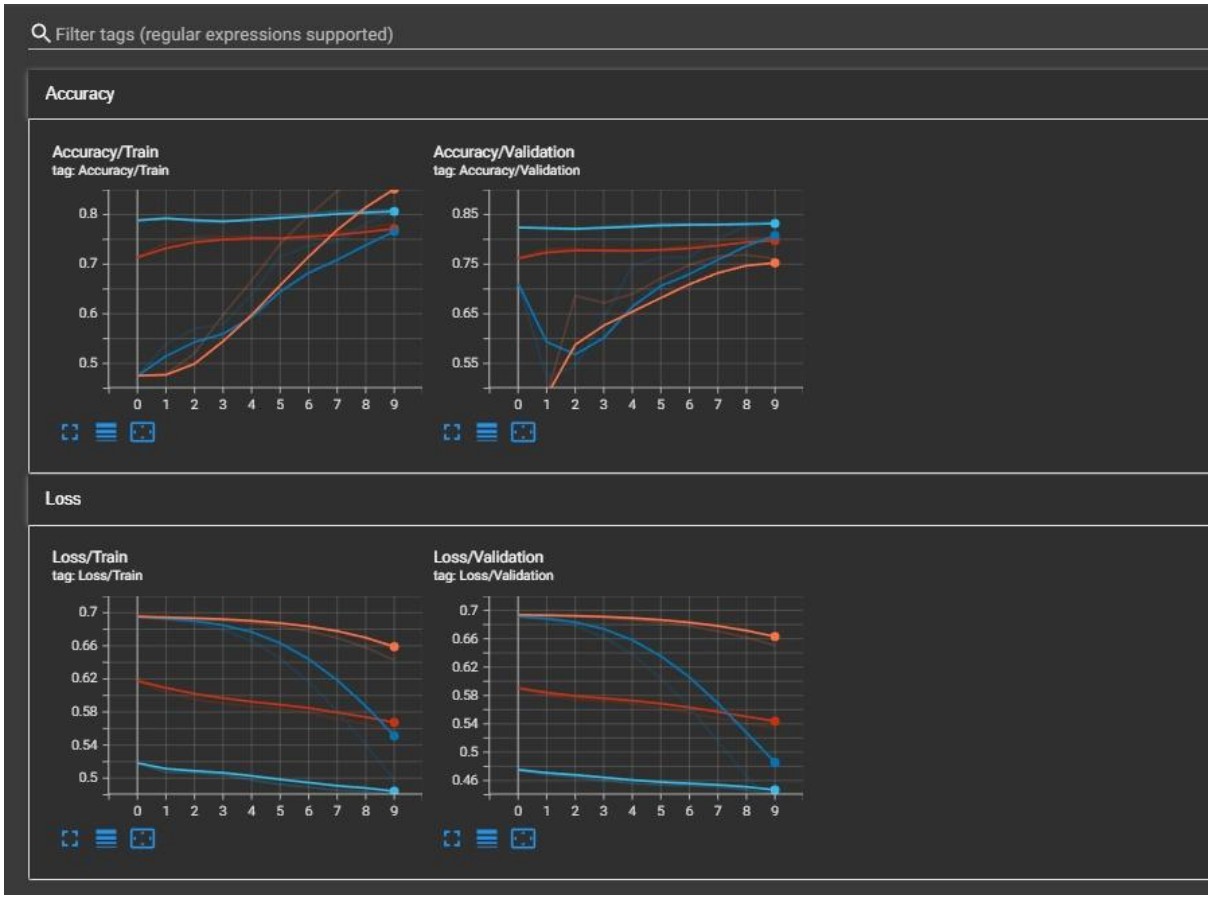
Logged Metrics:




Accuracy/Validation
0.8
0.7
0.6
0 1 2 3 4 5 6 7 8
Run ↑ Smoothed Value Step Relative
bow 0.7526 0.7608 9 4.813 sec
embeddings_model 0.8076 0.8389 9 63.73 ms
imdb_bow 0.7977 0.8029 9 15.26 sec
imdb_embedding_model 0.832 0.8339 9 77.61 ms


Loss 2 cards
Loss/Train
0.7
0.65
0.6
0.55
0.5
0 1 2 3 4 5 6 7 8
Run ↑ Smoothed Value Step Relative
bow 0.6589 0.6428 9 4.813 sec
embeddings_model 0.5511 0.4975 9 63.78 ms
imdb_bow 0.5675 0.558 9 15.26 sec
imdb_embedding_model 0.4842 0.4786 9 77.66 ms


Loss/Validation
0.7
0.65
0.6
0.55
0.5
0.45
0 1 2 3 4 5 6 7 8
Run ↑ Smoothed Value Step Relative
bow 0.6631 0.6507 9 4.813 sec
embeddings_model 0.4856 0.4234 9 63.73 ms
imdb_bow 0.5438 0.5342 9 15.26 sec
imdb_embedding_model 0.4469 0.4407 9 77.61 ms


Filter tags (regular expressions supported)
Accuracy
Accuracy/Train
tag: Accuracy/Train
0.8
0.7
0.6
0.5
0 1 2 3 4 5 6 7 8 9
Accuracy/Validation
tag: Accuracy/Validation
0.85
0.75
0.65
0.55
0 1 2 3 4 5 6 7 8 9
Loss
Loss/Train
tag: Loss/Train
0.7
0.66
0.62
0.58
0.54
0.5
0 1 2 3 4 5 6 7 8 9
Loss/Validation
tag: Loss/Validation
0.7
0.66
0.62
0.58
0.54
0.5
0.46
0 1 2 3 4 5 6 7 8 9