Text-based Emotion Recognition across BERT Family and LSTM

NLP class project

Andy Cherney, Abbeer Wani, Kexin Shang

Our reference paper:
Comparative Analyses of BERT, RoBERTa, DistilBERT, and XLNet for Text-Based Emotion Recognition

This paper analyzes the efficacy of BERT, RoBERTa, DistilBERT, and XLNet pre-trained transformer models in recognizing emotions from texts. The paper undertakes this by analyzing each candidate model's output compared with the remaining candidate models. The implemented models are fine-tuned on the ISEAR data to distinguish emotions into anger, disgust, sadness, fear, joy, shame, and guilt.

A. F. Adoma, N. -M. Henry and W. Chen, "Comparative Analyses of Bert, Roberta, Distilbert, and Xlnet for Text-Based Emotion Recognition," *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, Chengdu, China, 2020, pp. 117-121, doi: 10.1109/ICCWAMTIP51612.2020.9317379.

Data: ISEAR

- Publicly available dataset constructed through cross-culture questionnaire studies in 37 countries.
- It contains 7666 sentences classified into seven distinct emotion labels:
    - Joy (0)
    - Anger (1)
    - Sadness (2)
    - Shame (3)
    - Guilt (4)
    - Surprise (5)
    - Fear (6)

Table 1 Data Distribution of the ISEAR Dataset

Emotion Labels	Quantity
Anger	1096
Disgust	1096
Sadness	1096
Shame	1096
Fear	1095
Joy	1094
Guilt	1093
Total	7666

Hyperparameters

- Batch size: 16
- Epoch: 10
- Learning rate: 4e-5
- Padding length: 200 (but changed to max_length in our practice)
- Optimizer: Adam
- Loss function: sparse_categorical_crossentropy (equivalent to torch.nn.functional.cross_entropy)

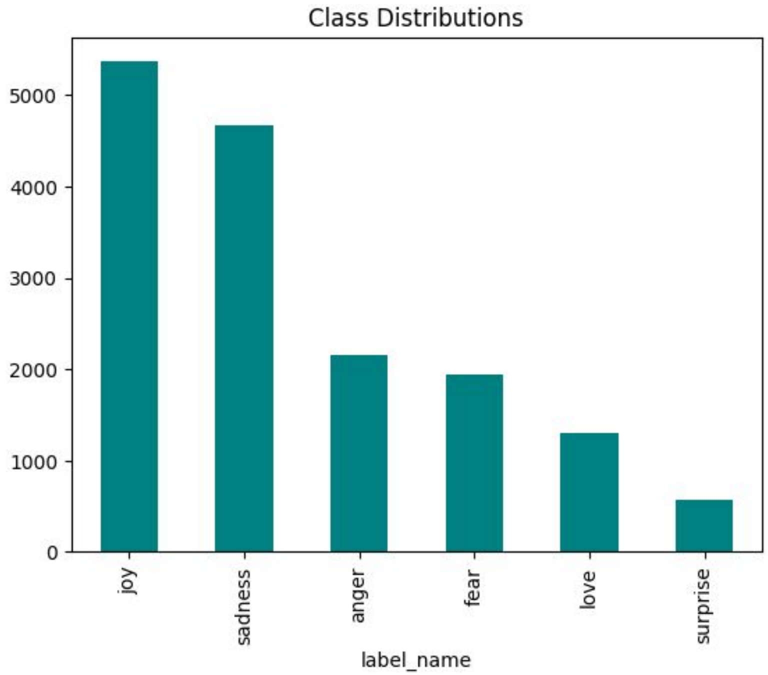
Our data:

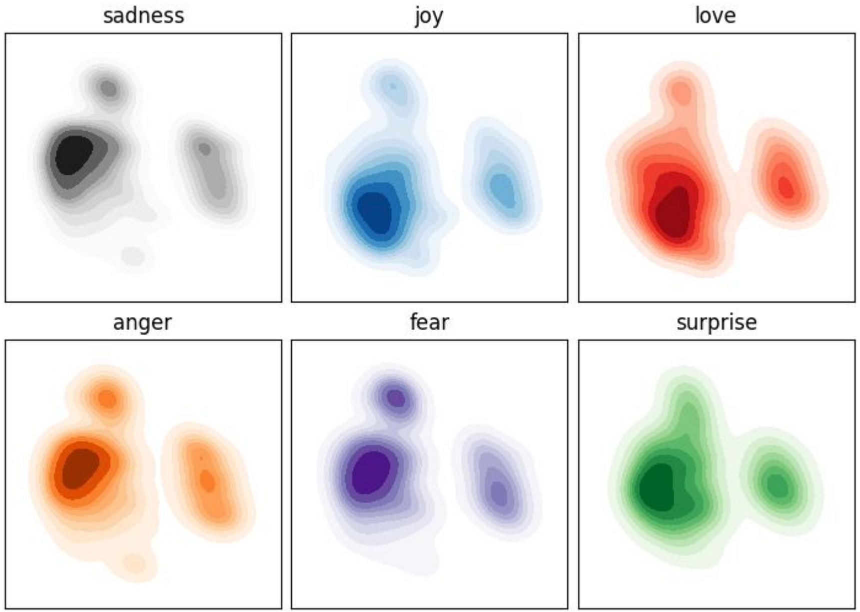
- Set of 20,000 unique English tweets gathered on the Huggingface repository
- Very similar in structure to the ISEAR dataset but has slightly different labels:
    - Sadness (0)
    - Joy (1)
    - Love (2)
    - Anger (3)
    - Fear (4)
    - Surprise (5)

text string · *lengths* 7 300	label class label 6 classes
i didnt feel humiliated	0 sadness
i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake	0 sadness
im grabbing a minute to post i feel greedy wrong	3 anger
i am ever feeling nostalgic about the fireplace i will know that it is still on the property	2 love
i am feeling grouchy	3 anger
ive been feeling a little burdened lately wasnt sure why that was	0 sadness
ive been taking or milligrams or times recommended amount and ive fallen asleep a lot faster but i also feel like so funny	5 surprise
i feel as confused about life as a teenager or as jaded as a year old man	4 fear

‹ Previous 1 2 3 ... 160 Next ›

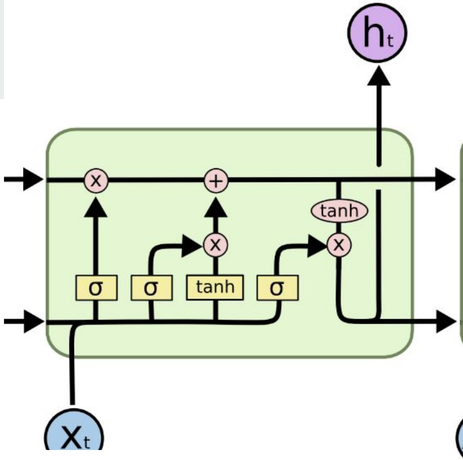
Data Exploration





The Baseline Model: LSTM

LSTM



Long Short-Term Memory Units (LSTM) are a special type of RNN that further improved Gated Recurrent Units (GRUs).

With 4 carefully regulated "gates", LSTM has the ability to remove or add information to the cell state and thus avoid the **long-term dependency problem**.

However, as a previous generation architecture, **LSTM doesn't have attention mechanism as BERT family do.** In this project, we added another experience on LSTM as our baseline.

Implementation

1) Make vocab and dataloader
   Max_padding = 117

2) Train a embedding word vectors use our train data based on "glove-wiki-gigaword-50"

```
LSTM_wvs.shape, LSTM_wvs
```

```
[8]  ✓ 0.0s
```

```
(torch.Size([3120, 50]),
 tensor([[ 1.9269,  1.4873,  0.9007,  ...,  1.3835, -1.2024,  0.7078],
        [ 2.2181,  0.5232,  0.3466,  ...,  0.5069, -0.4752, -0.4920],
        [ 0.1189,  0.1525, -0.0821,  ..., -0.5751, -0.2667,  0.9212],
        ...,
        [ 0.8106, -0.5419,  0.1878,  ..., -0.9614,  0.0169, -1.0266],
        [ 0.4760,  0.8361, -1.1790,  ..., -0.7668, -0.4678, -0.1378],
        [ 0.3964,  1.2262, -0.6604,  ..., -0.3464, -0.6066, -0.4939]]))
```

Implementation

3) Define LSTM module from nn.Module

   Bidirectional = True
   Lstm_layers = 2
   Output_dim = 6
   Linear_pre layer's input dim = 2*hidden_dim*seq_len(117)
   Gradient clipping turned on

2) Fine-tune with a self-defined trainer function
   early stop turned on

3) Evaluate:
   Use argmax for multi-label classification prediction

Results

Accuracy: 64%

Training time: 30 sec

	precision	recall	f1-score	support
Sadness	0.499006	0.919414	0.646907	273.000
Joy	0.806061	0.760000	0.782353	350.000
Love	1.000000	0.123288	0.219512	73.000
Anger	0.833333	0.384615	0.526316	156.000
Fear	0.666667	0.452174	0.538860	115.000
Surprise	1.000000	0.242424	0.390244	33.000
Accuracy	0.646000	0.646000	0.646000	0.646
Macor Avg	0.800844	0.480319	0.517365	1000.000
Wt Avg	0.731017	0.646000	0.623406	1000.000

```
Start epoch 9
Total epoch loss: 12773.06 (Avg. 22.527446)
Total validation loss: 1113.70 (Avg. 1.964203)

Start epoch 10
Total epoch loss: 13908.61 (Avg. 22.077156)
Total validation loss: 1046.93 (Avg. 1.661801)

Start epoch 11
Total epoch loss: 14947.98 (Avg. 21.569961)
Total validation loss: 953.59 (Avg. 1.376029)

Start epoch 12
Total epoch loss: 15925.78 (Avg. 21.065850)
Total validation loss: 954.83 (Avg. 1.263006)
No improvement! Early stopping
```

The BERT Family and XLNET

BERT

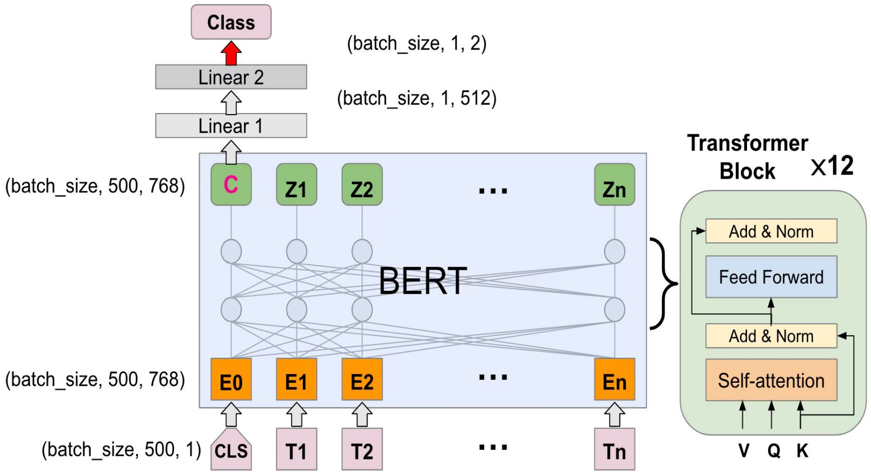
Input Embeddings: Word, position, and token type embeddings combined, normalized, and regularized.

Transformer Encoder: A stack of 12 transformer blocks, each with self-attention and feed-forward sub-layers (12 transformer layers, 768-hidden layers, 12 attention heads)

Pooling Layer: Applies a dense layer and Tanh activation to the [CLS] token.

Dropout: Adds regularization before the final classification layer.

Classification Layer: Projects the pooled output to the desired number of classes.



BERT Inputs

- This is a transformer model, which requires tokens to be encoded as numerical vectors
    - Produces a static attention mask and input_ids (the embeddings)
    - Model randomly masks 15% of the words in the input then runs the entire masked sentence through every epoch
- APIs used: BertForSequenceClassification, BertTokenizer
- Tokenization style:

```
'[CLS] im feeling rather rotten so im not very ambitious righ
now [SEP] [PAD] [PAD]..'
```

BERT Implementation

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=6, bias=True)
)
```

AdamW optimizer with LR Scheduler to adjust learning rate

```python
# the folowing params are the same as in our reference paper
batch_size = 16
num_epochs = 10
num_training_steps = num_epochs * len(BERT_train_dataloader)
total_steps = len(BERT_train_dataloader)

optimizer = Adam(BERT_model.parameters(), lr=4e-5) #Adam optimizer
lr_scheduler = get_scheduler(name="linear", optimizer=optimizer, num_warmup_steps=0, num_training_steps=num_training_steps)
```

Accelerator in training loop to speed up backprop

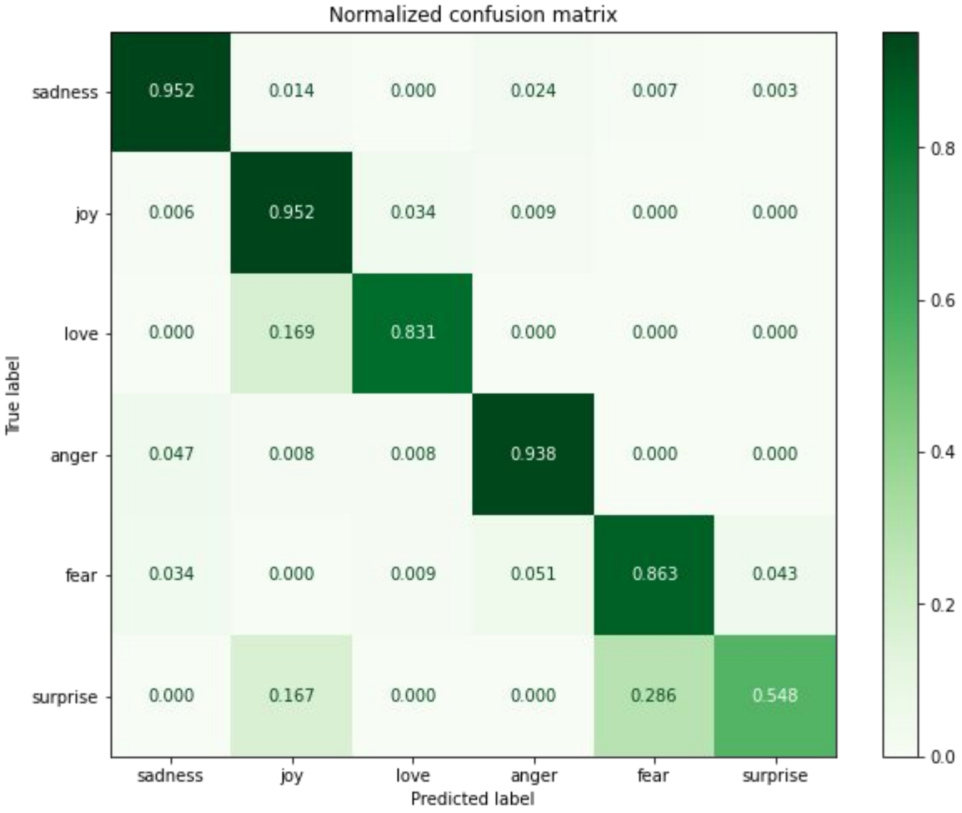
```python
progress_bar = tqdm(range(num_training_steps))

BERT_model.train()
total_loss = 0
accelerator = Accelerator()

for epoch in range(num_epochs):

    print('-' * 20)
    print(f'Begin epoch {epoch+1}')
    print('-' * 20)

    for step, batch in enumerate(BERT_train_dataloader): # 63 + 1
```

BERT Training Performance

```
--------------------
Begin epoch 10
--------------------
Total loss:  379.95814476534724
[0.000%] Step 0/63.                Average Loss: 0.042143
Total loss:  381.5416242759675
[15.873%] Step 10/63.                Average Loss: 0.041580
Total loss:  382.22154187969863
[31.746%] Step 20/63.                Average Loss: 0.040941
Total loss:  382.7556634731591
[47.619%] Step 30/63.                Average Loss: 0.040307
Total loss:  383.2070836648345
[63.492%] Step 40/63.                Average Loss: 0.039686
Total loss:  383.7115953452885
[79.365%] Step 50/63.                Average Loss: 0.039090
Total loss:  384.06783994473517
[95.238%] Step 60/63.                Average Loss: 0.038499
--------------------
End of epoch 10
--------------------
Total epoch loss: 384.12         (Average Loss: 0.038412)
```

Training time: 2 hours
Evaluation time: 45 minutes

BERT Model Results



91.4% accuracy

	precision	recall	f1-score	support
sadness	0.9516	0.9582	0.9549	287
joy	0.9517	0.9331	0.9423	359
love	0.8310	0.8082	0.8194	73
anger	0.9380	0.8832	0.9098	137
fear	0.8632	0.8783	0.8707	115
surprise	0.5476	0.7931	0.6479	29
accuracy			0.9140	1000
macro avg	0.8472	0.8757	0.8575	1000
weighted avg	0.9191	0.9140	0.9157	1000

RoBERTa

12 transformer layers, 768-hidden layers, 12 attention heads

Tokenization style:

`'<s>im feeling rather rotten so im not very ambitious right now</s><pad><pad>..'`

API I used: RobertaTokenizer, RobertaForSequenceClassification from transformers

Features of RoBERTa – Pre-training with dynamic masking

In contrast to the static masking used in the original BERT model, which masks the same tokens at every epoch of pre-training, dynamic masking involves **randomly masking** different tokens at different points during pre-training.

This encourages the model to learn more robustly

Implementation

1) Use “mps” to utilize the GPU on M3 macbook

```
device = "mps" if torch.backends.mps.is_built() else torch.device("cpu")
model.to(device)
```

2) Use lr_scheduler to adjust learning rate dynamically

```
optimizer = AdamW(model.parameters(), lr=4e-5) #Adam optimizer with weight decay

lr_scheduler = get_scheduler(
    name="linear", optimizer=optimizer, num_warmup_steps=0, num_training_steps=num_training_steps)
```

3) Use Accelerator package to accelerate the backpropagation on loss function

```
for step, batch in enumerate(RoBERTa_train_dataloader): # 63 + 1

    optimizer.zero_grad()
    batch = {k: v.to(device) for k, v in batch.items()}
    outputs = model(**batch) # equivalent to model.forward(**batch)

    loss = torch.nn.functional.cross_entropy(
        target=batch['labels'], input=outputs.logits, reduction='sum')
    accelerator.backward(loss)
    optimizer.step()
    lr_scheduler.step()

    total_loss += loss.cpu().detach().item()
```

4) Get metrics from *sklearn.metrics import classification_report*

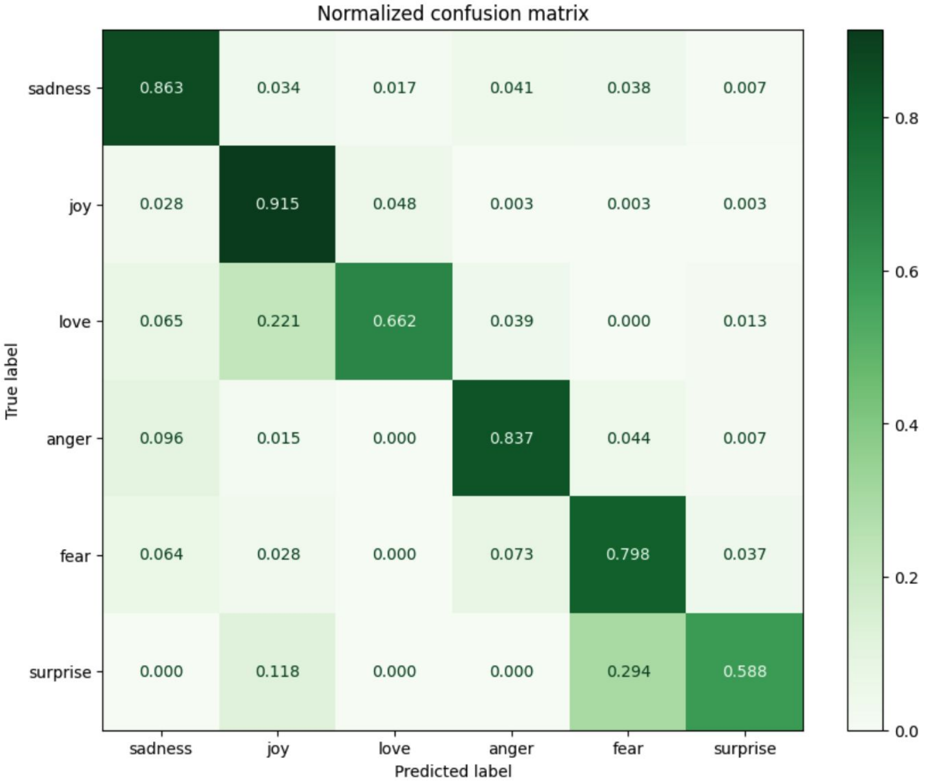
Results

Accuracy: 86%

Training time: 2 hours
Validation time: 8 mins

	precision	recall	f1-score	support
Sadness	0.901754	0.895470	0.898601	287.00
Joy	0.898876	0.891365	0.895105	359.00
Love	0.714286	0.684932	0.699301	73.00
Anger	0.805556	0.846715	0.825623	137.00
Fear	0.837607	0.852174	0.844828	115.00
Surprise	0.678571	0.655172	0.666667	29.00
Accuracy	0.860000	0.860000	0.860000	0.86
Macor Avg	0.806108	0.804305	0.805021	1000.00
Wt Avg	0.860007	0.860000	0.859889	1000.00

```
--------------------
Begin epoch 10
--------------------
Total loss:  6154.420735940337
[0.000%] Step 0/63.             Average Loss: 0.682611
Total loss:  6170.62565331161
[15.873%] Step 10/63.             Average Loss: 0.672474
Total loss:  6181.252208486199
[31.746%] Step 20/63.             Average Loss: 0.662088
Total loss:  6197.837710484862
[47.619%] Step 30/63.             Average Loss: 0.652679
Total loss:  6205.694712385535
[63.492%] Step 40/63.             Average Loss: 0.642678
Total loss:  6214.259819447994
[79.365%] Step 50/63.             Average Loss: 0.633075
Total loss:  6222.346022441983
[95.238%] Step 60/63.             Average Loss: 0.623732
--------------------
End of epoch 10
```



DistilBERT

- **Input Layer:** The DistilBERT Model takes inputs: ids, and mask. These inputs are encoded representations of the input text obtained using a tokenizer.
- **pre-trained Layer:** The pre-trained “distilbert-base-uncased” weights are used to initialize the DistilBERT model. It processes the input tensors to obtain the hidden state output. The first token of the sequence is used to obtain a pooled representation of the input sequence from the hidden_state tensor.
- **Linear Layer pre-classifier:** Linear layer pre-classifier is responsible for extracting features from data by taking the output of the pre-trained layer and passing it to a fully connected layer.
- **Fully connected Layer:** The fully connected layer is a linear layer that takes the output of linear layer pre-classifier and maps it to the desired output dimensionality. In this case, the output dimensionality is six, corresponding to topic tagging.

Implementation

Tokenization

Transformer models like `DistilBERT` do not take raw strings as input; instead, they require the text to undergo tokenization and be encoded as numerical vectors. Tokenization involves breaking down text into smaller units, often word or subword pieces. We used WordPiece tokenizer, since it is used by `DistilBERT` model. The AutoTokenizer class to was used to load the tokenizer of the model by using the `from_pretrained()` method.

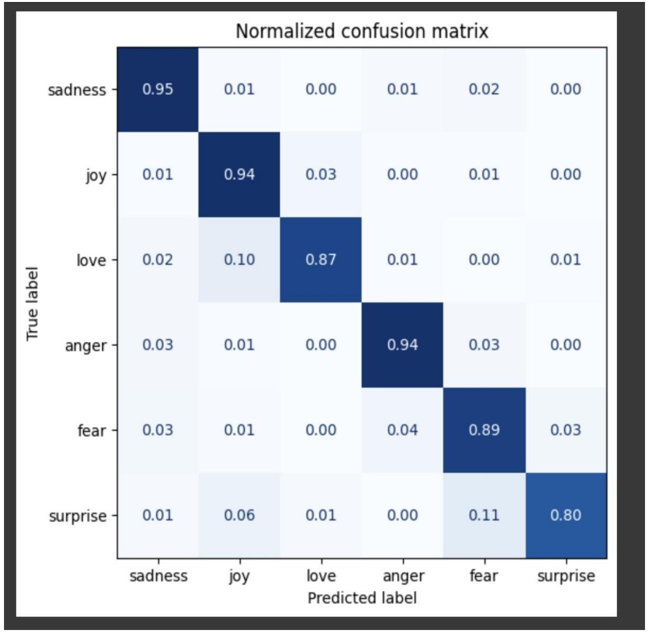
```python
def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True)

pprint.pprint(tokenize(emotions_dataset["train"][167:169]), compact=True)
```

```
{'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]],
 'input_ids': [[101, 1045, 2031, 1037, 3110, 2026, 3193, 3475, 2102, 2183, 2000,
                2022, 2200, 2759, 1998, 2008, 2015, 2986, 102],
               [101, 1045, 2215, 2000, 2907, 2023, 3110, 1997, 7135, 15180,
                1998, 4687, 5091, 102, 0, 0, 0, 0, 0]]}
```

```python
def get_hidden_states(batch):
    inputs = {k:v.to(device) for k,v in batch.items()
              if k in tokenizer.model_input_names}

    with torch.no_grad():
        last_hidden_state = model(**inputs).last_hidden_state
    return {"hidden_state": last_hidden_state[:,0].cpu().numpy()}
```

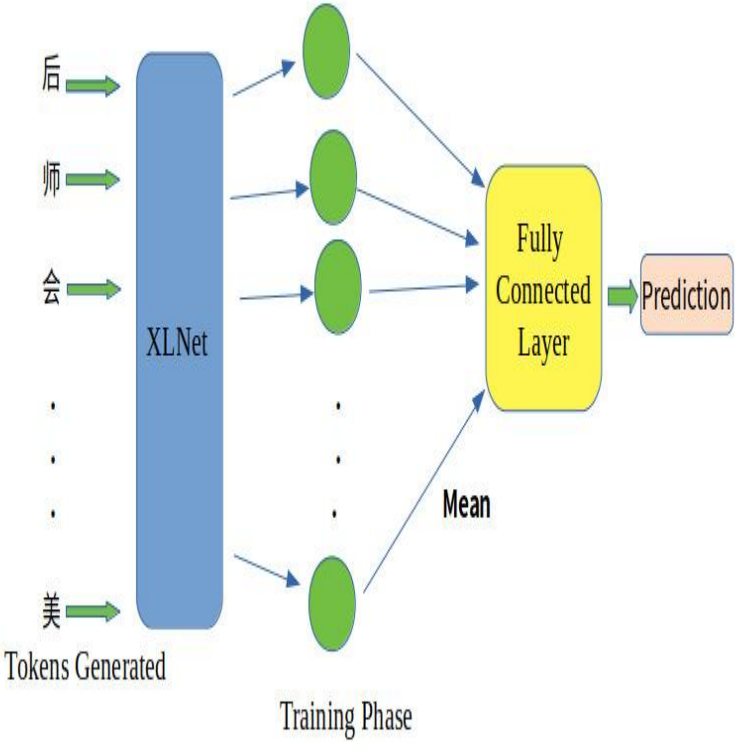
```python
encoded_emotions_ds.set_format("torch",
                               columns=["input_ids", "attention_mask", "label"])

hidden_emotions_ds = encoded_emotions_ds.map(get_hidden_states,
                                             batched=True)

hidden_emotions_ds["train"].column_names
```

Results



	precision	recall	f1-score	support
sadness	0.97	0.96	0.96	550
joy	0.96	0.94	0.95	704
love	0.86	0.87	0.86	178
anger	0.92	0.95	0.93	275
fear	0.83	0.89	0.86	212
surprise	0.90	0.75	0.82	81
accuracy			0.93	2000
macro avg	0.90	0.89	0.90	2000
weighted avg	0.93	0.93	0.93	2000

XLNet



	precision	recall	f1-score	support
0	0.62	0.71	0.66	550
1	0.72	0.77	0.74	704
2	0.49	0.40	0.44	178
3	0.60	0.49	0.54	275
4	0.55	0.50	0.53	212
5	0.49	0.32	0.39	81
accuracy			0.64	2000
macro avg	0.58	0.53	0.55	2000
weighted avg	0.63	0.64	0.63	2000

Paper Results

Table 2 Comparison of Precision, Recall And F1-scores of BERT, RoBERTa, DistilBERT, and XLNET on the ISEAR Dataset

Models	Anger			Disgust			Fear			Guilt			Joy			Sadness			Shame		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
BERT	0.56	0.57	0.57	0.71	0.63	0.67	0.74	0.76	0.75	0.65	0.69	0.67	0.84	0.91	0.88	0.76	0.8	0.78	0.63	0.57	0.6
RoBERTa	**0.67**	**0.59**	**0.62**	**0.76**	0.69	**0.73**	**0.8**	**0.81**	**0.8**	0.62	**0.76**	0.68	0.9	**0.96**	**0.93**	**0.77**	**0.81**	**0.79**	**0.69**	**0.62**	**0.65**
DistilBERT	0.52	0.57	0.55	0.69	0.65	0.67	0.7	0.76	0.73	0.63	0.6	0.61	0.88	0.81	0.85	0.76	0.8	0.78	0.54	0.51	0.52
XLNET	0.59	0.57	0.58	0.69	**0.73**	0.71	0.76	**0.81**	0.78	**0.7**	0.72	**0.71**	**0.91**	0.93	0.92	0.76	**0.81**	**0.79**	**0.69**	0.57	0.63

Our Results

Models	Sadness			Joy			Love			Anger			Fear			Surprise		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
BERT	0.95	**0.96**	0.95	0.95	0.93	0.94	0.83	0.81	0.82	**0.94**	0.88	0.91	**0.86**	0.88	**0.87**	0.55	**0.79**	0.65
RoBERTa	0.90	0.90	0.90	0.90	0.89	0.90	0.71	0.68	0.70	0.81	0.85	0.83	0.84	0.85	0.84	0.68	0.66	0.67
DistilBERT	**0.97**	**0.96**	**0.96**	**0.96**	**0.94**	**0.95**	0.86	**0.87**	**0.86**	0.92	**0.95**	**0.93**	0.83	**0.89**	0.86	0.90	0.75	**0.82**
XLNET	0.62	0.71	0.66	0.72	0.77	0.74	0.49	0.40	0.44	0.60	0.49	0.54	0.55	0.50	0.53	0.49	0.32	0.39
LSTM	0.50	0.92	0.65	0.81	0.76	0.78	**1.00**	0.12	0.22	0.83	0.38	0.53	0.67	0.45	0.54	**1.00**	0.24	0.39

Conclusion

1. From our reproduction, all models attained higher precision, recall and F1 score compared to the paper. This might because we used a more imbalanced dataset with fewer labels.

2. A notable finding/ confusion is that the best model shifts from RoBERTa to DistilBERT after applying the new data even if we used almost the same hyperparameters as in the paper.

3. Our baseline, LSTM, has the worst performance. This might have two potential reasons:

    a. LSTM is known for capturing long-range dependencies in **sequential text context**, but our assignment does not involve bulky context but only pieces of independent sentences. Therefore, LSTM lost its strength.

    b. The emergence of **attention mechanism** significantly boosted the performance of BERT family.

References

- Dataset: https://huggingface.co/datasets/dair-ai/emotion
- Reference Paper: https://ieeexplore.ieee.org/document/9317379
- https://huggingface.co/blog/bert-101
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://medium.com/@musawi.rahmat/distilbert-finetuned-emotion-368237455a96
- https://huggingface.co/docs/transformers/en/model_doc/xlnet
- https://huggingface.co/docs/transformers/en/model_doc/roberta
- https://www.comet.com/site/blog/roberta-a-modified-bert-model-for-nlp/

Thanks!