

Task 1 - Transformer from Scratch

Introduction

The goal of this task is to implement a Transformer model from scratch for language modeling using the Shakespearean dataset. The model should be trained to generate text in Shakespeare's style.

Dataset Pre-Preprocessing

The preprocessing steps for the Shakespeare dataset involve several key stages to prepare the raw text for effective model training.

First, the text data is loaded from files, and each line is cleaned by stripping whitespace and ensuring non-empty lines. The text is then tokenized, where each sentence is split into individual words or tokens. Special tokens like <PAD>, <START>, <STOP>, and <UNK> are introduced to handle padding, sentence boundaries, and unknown words. A vocabulary is built using the Counter class to count token frequencies, and the tokens are mapped to unique integer IDs for model compatibility.

This mapping helps convert textual data into numerical format suitable for neural networks. After tokenization, sentences are padded to a fixed length (MAX_LEN) to ensure uniformity across batches. Padding tokens are used where necessary, and the model is designed to ignore these during loss calculation.

Additionally, a causal mask is created to prevent the model from accessing future tokens during training, maintaining the autoregressive property required for language modeling. Finally, the processed data is split into training and validation sets, ensuring that the model can be evaluated effectively on unseen data.

Models

The model architecture is based on the Transformer framework, designed specifically for language modeling tasks like generating Shakespearean text.

It begins with an embedding layer that converts input tokens into dense vector representations, enriched with positional embeddings to capture the sequence order since Transformers lack inherent sequence modeling capabilities. The core of the model consists of multiple Transformer blocks—in this case, two—each containing a multi-head self-attention mechanism that enables the model to focus on different parts of the input simultaneously, followed by a feedforward

neural network (FFN) with ReLU activation. Layer normalization and residual connections are incorporated to stabilize training and mitigate issues like vanishing gradients

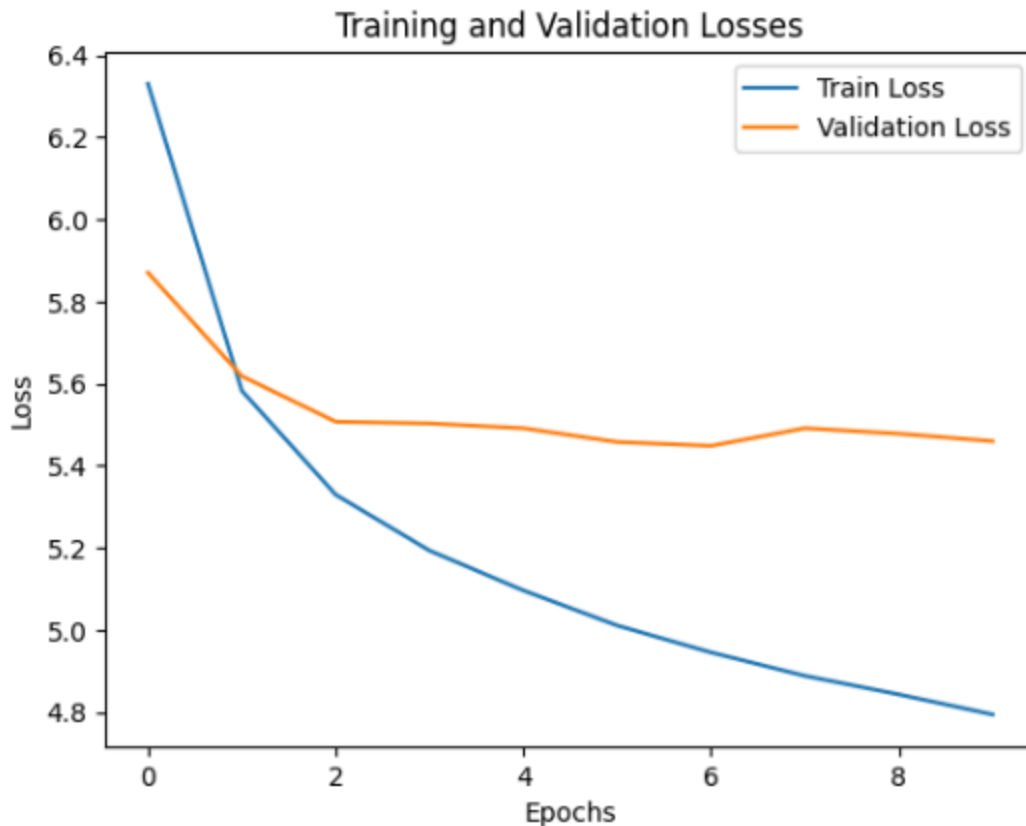
. The output of the final Transformer block is passed through a fully connected layer that projects it back to the vocabulary size, producing logits for each token, which are then used to calculate the loss during training.

Model Training

The hyperparameters selected for this model include an embedding dimension of 256, allowing for rich token representations, and two Transformer layers to balance model complexity and performance. The model uses four attention heads to capture diverse contextual information from different parts of the sequence. A dropout rate of 0.2 is applied as a regularization technique to prevent overfitting, while a learning rate of 0.001, along with a learning rate scheduler, helps optimize the training process effectively. The batch size is set to 32 to ensure efficient gradient computation, and the maximum sequence length is limited to 128 tokens to manage memory usage while preserving contextual information. The optimizer used is Adam with weight decay for regularization, and gradient clipping is applied with a maximum norm of 1.0 to prevent exploding gradients. This combination of architecture and hyperparameters aims to achieve a strong balance between learning capacity, generalization, and computational efficiency

Loss Plots

```
/
Epoch 1: Train Loss = 6.3305, Val Loss = 5.8703
Sample text: <START> ANGELO Because defending Red play affect nimbly salt Call ELBOW brook cogitation Night Scorns Honest Accursed up-roused block congealed Oxfordshire
Epoch 2: Train Loss = 5.5827, Val Loss = 5.6182
Sample text: <START> ISABELLA tetchy seld-shown conflicts naught achieved amain courted affray marriage-day MAMILLIUS Translate crushing Bred notable remorse sort afore beware 'True
Epoch 3: Train Loss = 5.3293, Val Loss = 5.5071
Sample text: <START> ANGELO goddesses hourly Grecian winners daring-hardy beetle combating statute surmise o'erthrown degrees peruse Vitruvius galled levy 'havior Avaunt birds foreru
n
Epoch 4: Train Loss = 5.1930, Val Loss = 5.5030
Sample text: <START> ISABELLA crime shallow Crowd distressed clears thirty piles wept drive thirty gentry richly opposers 'Shall appears unking pity neuter severely
Epoch 5: Train Loss = 5.0959, Val Loss = 5.4912
Sample text: <START> ISABELLA intestate bait instruments honours bites graft'st Abraham shrub Shaw cowardice excellence neglected pardoning flag Unreasonable kindred covets nineteen
helm
Epoch 6: Train Loss = 5.0104, Val Loss = 5.4576
Sample text: <START> LUCIO sacrifices besides sweating convert smiled Prepare dolours unjustly popular proceed Rue likeness ensign tamed eagle truncheon inclining surfeit such
Epoch 7: Train Loss = 4.9448, Val Loss = 5.4481
Sample text: <START> ANGELO multitudes 'True knots yeoman slaughter-man intellect middle Would permitted dallies victory Unvenerable serpigo Call't Ratcliff charged quarry towns R
Epoch 8: Train Loss = 4.8878, Val Loss = 5.4913
Sample text: <START> ISABELLA Noting choked thundering titleless breathing commons smoke dies weightier abhorred hoped-for resting mutiny underprop vexed fitly length Attend courtsh
ip
Epoch 9: Train Loss = 4.8425, Val Loss = 5.4779
Sample text: <START> DUKE images seemeth tried haught new-appearing eternal damned lean hairs lane Withhold aches cheering defy Cuts lay estates sufficiently trusty
Epoch 10: Train Loss = 4.7937, Val Loss = 5.4597
Sample text: <START> ISABELLA Wisely Unpeopled 'good opposer receives soon-believing lodges cape sit pensive sworn caterpillars beguiled 'They rheums conduct butler barr'st soften
Model saved as 'simplified_transformer_lm.pth'.
```



Training perplexity: 171.9300
Validation perplexity: 237.4536

Task 2 - Claim Normalization

Introduction

Claim Normalization is a novel task that involves transforming complex and noisy social media posts into clear and structured claims, referred to as normalized claims.

Data Preprocessing

The preprocessing pipeline was designed to clean and normalize text data from social media posts and their corresponding claims. The goal was to prepare the data for training by ensuring consistency in formatting, removing noise, and expanding informal language.

1. Contraction Expansion

Contractions such as "don't" and "it's" were expanded into their full forms ("do not", "it is") using the `contractions` Python library. This step helps improve the

quality of tokenization and downstream language processing.

2. **Abbreviation Expansion**

A custom abbreviation mapping was applied to replace commonly used abbreviations with their full forms. For example:

- "Gov." → "Governor"
- "VP" → "Vice President"
- "U.S." → "United States"

3. This ensures clarity and semantic completeness in the text.

4. **Text Cleaning**

To reduce noise, the following cleaning operations were performed:

- Removal of URLs (e.g., `https://example.com`)
- Removal of special characters, keeping only alphanumeric characters and basic punctuation (. , ! ?)
- Elimination of excessive whitespace

5. **Lowercasing**

All text was converted to lowercase to ensure case consistency and reduce vocabulary size.

6. **DataFrame Processing**

The above preprocessing steps were applied to both the `Social Media Post` and `Normalized Claim` columns. The processed columns were renamed to `input_text` and `target_text`, respectively. Rows with missing values were dropped.

7. **Dataset Preparation**

The cleaned datasets were saved as CSV files:

- `train.csv`
- `val.csv`
- `Test.csv`

Model 1 - BART

BART Architecture

The task of claim normalization was framed as a **sequence-to-sequence (seq2seq)** problem, where a noisy or informal social media post is translated into a normalized claim. For this, we used **BART**—a pre-trained Transformer model provided by the Hugging Face transformers library.

Model Used facebook/bart-base

- BART combines the strengths of BERT (bidirectional encoder) and GPT (autoregressive decoder).
- It is particularly well-suited for generation tasks like summarization, translation, and text normalization.

Tokenizer

- BartTokenizer was used to tokenize the input_text (social media post) and target_text (normalized claim).
- Texts were padded and truncated to fixed lengths for model input:
 - **Input sequence max length:** 512 tokens
 - **Target sequence max length:** 128 tokens

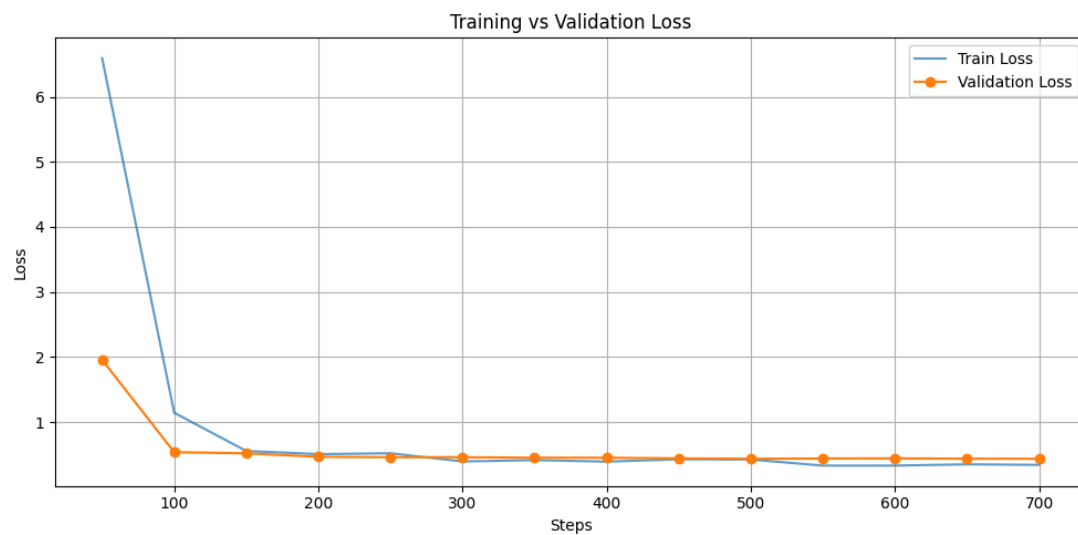
Hyperparameters

```
training_args = TrainingArguments(  
    output_dir="./bart-normalizer",  
    evaluation_strategy="steps",  
    learning_rate=5e-5,  
    per_device_train_batch_size=8,  
    per_device_eval_batch_size=8,  
    num_train_epochs=3,  
    weight_decay=0.01,  
    save_strategy="epoch",  
    logging_dir="./logs",  
    logging_steps=50,  
    save_total_limit=1,  
    fp16=torch.cuda.is_available(),  
    report_to = "none"  
)
```

Training

[738/738 06:17, Epoch 3/3]

Step	Training Loss	Validation Loss
50	6.592400	1.958104
100	1.144200	0.538470
150	0.556500	0.520429
200	0.508200	0.466945
250	0.521300	0.460375
300	0.395900	0.460981
350	0.416400	0.453082
400	0.393100	0.452555
450	0.428900	0.443097
500	0.424900	0.436796
550	0.332200	0.441173
600	0.332400	0.443042
650	0.351700	0.438587
700	0.343300	0.437363



Evaluation

```
{'ROUGE-L': 0.348, 'BLEU-4': 0.2552, 'BERTScore (F1)': 0.8835}
```

Model 2 - T5

T5 Architecture

To perform claim normalization as a **text-to-text generation task**, we also utilized the **T5 model**. T5 treats every NLP task—classification, translation, summarization, etc.—as a text generation problem, making it a flexible choice for normalization tasks.

Model Used: t5-base

- T5 uses an encoder-decoder architecture.
- It is trained on a wide range of tasks under a unified framework, making it highly generalizable.

Tokenizer:

- T5Tokenizer was used to tokenize the `input_text` (social media post) and `target_text` (normalized claim).
- Both inputs and outputs were padded and truncated to fixed maximum lengths:
 - **Input sequence max length:** 512 tokens
 - **Target sequence max length:** 128 tokens

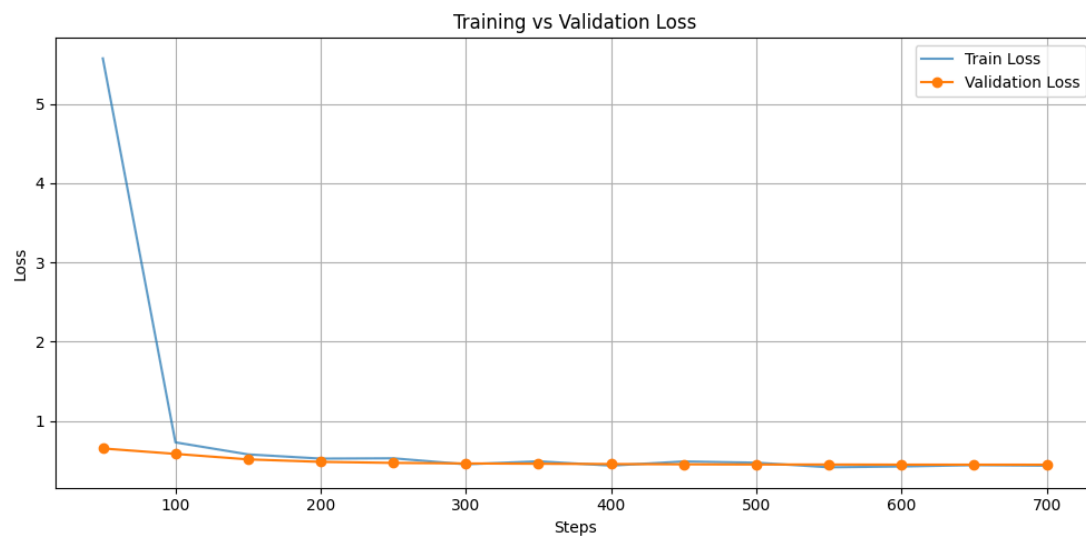
Hyperparameters

```
training_args = TrainingArguments(  
    output_dir="./t5-normalizer",  
    evaluation_strategy="steps",  
    learning_rate=5e-5,  
    per_device_train_batch_size=8,  
    per_device_eval_batch_size=8,  
    num_train_epochs=3,  
    weight_decay=0.01,  
    save_strategy="epoch",  
    logging_dir="./logs",  
    logging_steps=50,  
    save_total_limit=1,  
    fp16=torch.cuda.is_available(),  
    report_to = "none"  
)
```

Training

[738/738 12:36, Epoch 3/3]

Step	Training Loss	Validation Loss
50	5.572600	0.653430
100	0.730000	0.584916
150	0.578600	0.515589
200	0.525600	0.484412
250	0.529400	0.471402
300	0.453500	0.464486
350	0.491500	0.460638
400	0.436900	0.456874
450	0.489400	0.453403
500	0.473800	0.451081
550	0.415800	0.450777
600	0.426100	0.449342
650	0.443000	0.448930
700	0.436100	0.448364



Evaluation

```
{ 'ROUGE-L': 0.3243, 'BLEU-4': 0.2417, 'BERTScore (F1)': 0.877 }
```


Comparative Analysis

Metric	BART	T5
ROUGE-L	0.348	0.3243
BLEU-4	0.2552	0.2417
BERTScore (F1)	0.8835	0.877

- **ROUGE-L:** BART achieved a higher ROUGE-L score, indicating better overlap with the reference text in terms of longest common subsequences.
- **BLEU-4:** BART also outperformed T5 on BLEU-4, reflecting better precision in n-gram generation.
- **BERTScore:** While both models performed closely, BART had a slight edge in semantic similarity, as captured by BERTScore (F1).

Runtime and Resource Utilization

Aspect	BART	T5
Training Runtime	~6 minutes 17 seconds	~12 minutes 36 seconds
Model Size	~139M parameters	~220M parameters
Memory Footprint	Lower	Higher

- T5 required more memory and compute, which influenced runtime and scalability.
- **BART** trained nearly **twice as fast** as T5 under the same hardware conditions and batch sizes.

Resource Constraints and Model Selection

Due to limited 19 GB GPU compute on kaggle and the need for efficient experimentation, **BART was selected** for its:

- Faster training time
- Lower memory requirements
- Better overall performance across all metrics

Task 3 - Multimodal Sarcasm Explanation (MuSE)

Dataset and Pre-Preprocessing

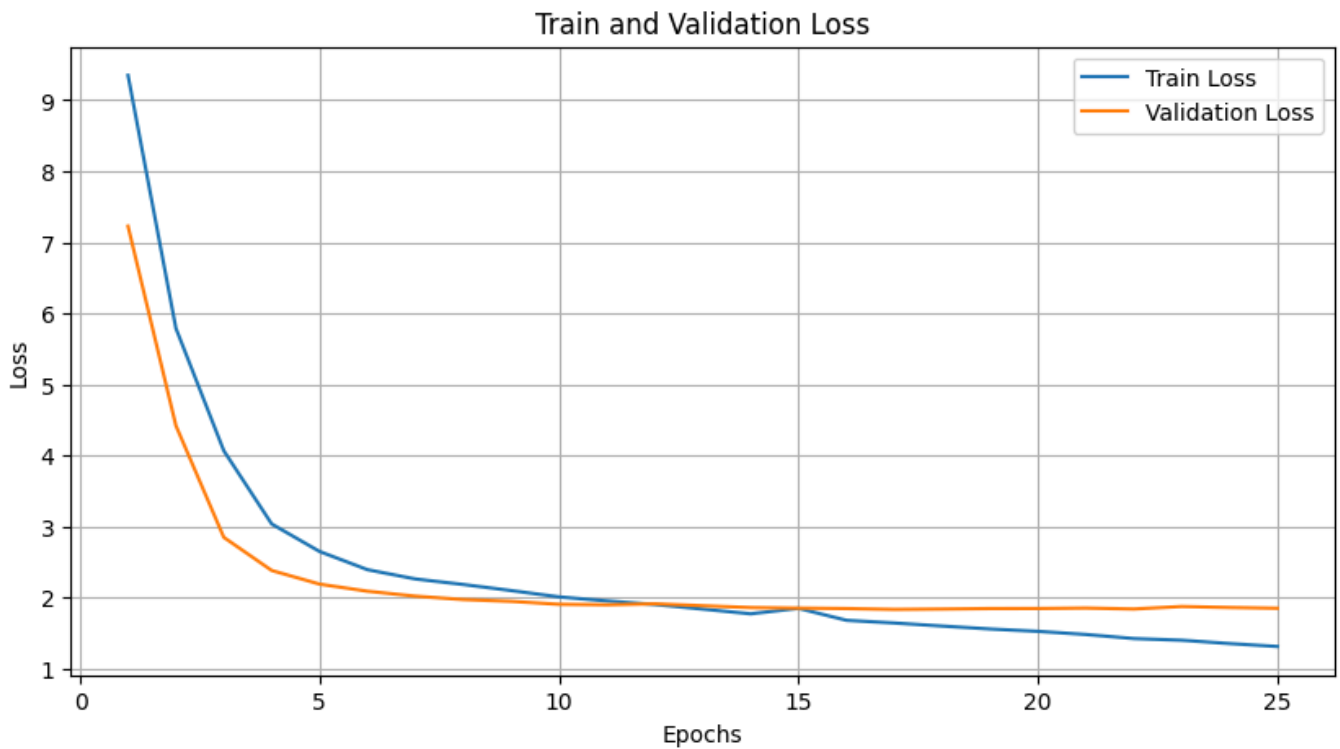
- **Multimodal Data:** Handles text, images, and image-derived text.
- **Input DataFrame:** Contains pid, text, explanation, and target_of_sarcasm.
- **Pickled Dictionaries:**
 - Image descriptions.
 - Object detection outputs.
- **Image Loading:** Images loaded via pid and converted to RGB.
- **Object Classes:** Object detection results concatenated into a text string.
- **Image Description:** Image-level descriptions retrieved.
- **Dataset Output:** Dictionary with:
 - Sarcastic text.
 - Target of sarcasm.
 - Detected object classes.
 - Image description.
 - Ground truth explanation.
 - Image.

Model architecture and hyperparameters used.

- **Multimodal Architecture:** Combines text and visual information.
- **Pretrained Backbones:**
 - **facebook/bart-base** for text encoding/decoding.
 - **google/vit-base-patch16-224** for image feature extraction.
- **Textual Stream:**
 - Encodes **text**, **target**, **objects**, and **description** using BART encoder.
 - Modality-type embeddings added.
- **Visual Stream:**
 - Images processed by ViT.
 - ViT embeddings projected to match BART's hidden size.
- **Self-Attention:**
 - Separate Transformer Encoders for visual and textual embeddings.
- **Cross-Attention:**
 - Bi-directional: visual-to-text and text-to-visual.
 - Normalized and residual-connected outputs.
- **Gated Fusion:**
 - Mean-pooled embeddings linearly transformed.
 - Gating vectors modulate modality contributions.

- Softmax-weighted sum of fusion variants.
- **Decoder:**
 - Fused vector passed to BART decoder for explanation generation.
 - Teacher forcing during training.
 - Beam search (4 beams) for inference.
- **Training Parameters:**
 - Dropout rate: 0.2.
 - 8 attention heads.
 - Text input capped at 512 tokens.
 - Image resolution: 224x224.
 - Learning Rate: 5e-6
 - Optimizer: AdamW
 - Batch Size: 8

Loss Plots



Training Loss and Evaluation Metrics after Epochs:

```
Epoch 5/25, Train Loss: 2.6474, Validation Loss: 2.1880  
Best model saved at epoch 5 with validation loss 2.1880
```

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
Evaluation Metrics:  
ROUGE-1: 0.2183  
ROUGE-2: 0.0845  
ROUGE-L: 0.2018  
BLEU-1: 0.1275  
BLEU-2: 0.0815  
BLEU-3: 0.0603  
BLEU-4: 0.0459  
METEOR: 0.1265  
BERTScore (F1): 0.8746
```

```
Epoch 10/25, Train Loss: 2.0084, Validation Loss: 1.9044  
Best model saved at epoch 10 with validation loss 1.9044
```

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```



```
Evaluation Metrics:  
ROUGE-1: 0.2309  
ROUGE-2: 0.0760  
ROUGE-L: 0.2081  
BLEU-1: 0.1320  
BLEU-2: 0.0777  
BLEU-3: 0.0552  
BLEU-4: 0.0418  
METEOR: 0.1197  
BERTScore (F1): 0.8748
```

Epoch 15/25, Train Loss: 1.8468, Validation Loss: 1.8505
Best model saved at epoch 15 with validation loss 1.8505

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Evaluation Metrics:

ROUGE-1: 0.2496
ROUGE-2: 0.0854
ROUGE-L: 0.2273
BLEU-1: 0.1370
BLEU-2: 0.0855
BLEU-3: 0.0628
BLEU-4: 0.0477
METEOR: 0.1311
BERTScore (F1): 0.8766

Epoch 20/25 - Training: 100% 373/373 [03:20<00:00, 1.86it/s]
Epoch 20/25 - Validation: 100% 22/22 [00:17<00:00, 1.27it/s]

Epoch 20/25, Train Loss: 1.5234, Validation Loss: 1.8449

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Evaluation Metrics:

ROUGE-1: 0.2810
ROUGE-2: 0.1059
ROUGE-L: 0.2577
BLEU-1: 0.1638
BLEU-2: 0.1020
BLEU-3: 0.0750
BLEU-4: 0.0561
METEOR: 0.1573
BERTScore (F1): 0.8807

```
Epoch 25/25, Train Loss: 1.3103, Validation Loss: 1.8486
```

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
Evaluation Metrics:
```

```
ROUGE-1: 0.2976
```

```
ROUGE-2: 0.1140
```

```
ROUGE-L: 0.2699
```

```
BLEU-1: 0.1906
```

```
BLEU-2: 0.1179
```

```
BLEU-3: 0.0868
```

```
BLEU-4: 0.0674
```

```
METEOR: 0.1710
```

```
BERTScore (F1): 0.8830
```

Generated sarcasm

```
Model Predictions vs. Actual Explanations:
```

```
Predicted: the author is pissed at <user> for such poor internet.
```

```
Actual: the author is pissed at <user> for not getting network in malad.
```

```
Predicted: the author is pissed at <user> for such a long wait.
```

```
Actual: nothing worst than waiting for an hour on the tarmac for a gate to come open in snowy, windy chicago.
```

```
Predicted: it's not spring, it's spring.
```

```
Actual: nobody likes getting one hour of their life sucked away.
```

```
Predicted: the author is pissed at <user> for not treating her doctor like a doctor.
```

```
Actual: having a salivary gland biopsy on monday morning is not a good way to start the new week.
```

```
Predicted: it's not a perfect day for snow.
```

```
Actual: the author is worried that the weekend is going to be freezing with a high of -1 and wind chill probably -30.
```