Transformer Based Joke-Generator

Vansh Dhar 22156

November 2024

Report

Question 1

In this report I cover my approach and results of training a custom decoder only model on the **Joke Dataset**. I only trained the model on the **Reddit Jokes Dataset** First I cover the techniques used to pre-process the dataset.

Pre-processing Techniques

Pre-processing text data is a critical step in ensuring that the model receives clean, meaningful input. The following common approaches were applied during the pre-processing of the dataset:

- **Text Cleaning:** Unnecessary characters, such as special symbols, HTML tags, and excessive whitespace, were removed to clean the text data.
- **Lowercasing:** All text was converted to lowercase to ensure consistency and reduce the vocabulary size.
- **Tokenization:** The dataset was tokenized into smaller units, such as words or subwords, using a tokenizer compatible with the target model.
- **Padding and Truncation:** All sequences were padded to the same length or truncated to meet the maximum sequence length requirement for the model.
- **Handling Missing Data:** Jokes with incomplete or missing titles and punchlines were excluded from the dataset to maintain data quality.
- **Separator Tokens:** Custom tokens, such as '$< |sep| >$' and '$< |endofjoke| >$', were added to separate the joke setup and punchline and mark the end of each joke.
- **Filtering Short Entries:** Jokes with very short setups or punchlines (less than 3 words) were removed to avoid feeding uninformative content to the model.

Use of Pre-trained Embedding Matrix

To improve the performance of the decoder-only model, a pre-trained embedding matrix was used. Specifically, the **GloVe 6 Billion Word Corpus with 100-Dimensional Embeddings** was utilized.

- **Advantages of Pre-trained Embeddings:**
  - Pre-trained embeddings capture semantic and syntactic relationships between words, reducing the burden on the model to learn these from scratch.
  - They are particularly useful when working with limited data, as they provide a robust initialization for the embedding layer.
- **Implementation:** The GloVe embeddings were loaded and aligned with the vocabulary of the joke dataset. Words not present in the GloVe vocabulary were assigned random embeddings, while the embeddings for known words were directly mapped from the pre-trained matrix.
- **Benefits Observed:** Using GloVe embeddings accelerated the convergence of the model during training and improved overall performance on the validation set.

By leveraging these pre-processing techniques and integrating GloVe embeddings, the dataset was transformed into a structured and meaningful format suitable for model training. These steps were essential to achieve efficient and effective training outcomes.

Training and Validation Loss

The training and validation losses over epochs for the first set of hyperparameters are presented in the table below:

Table 1: Training and Validation Loss for First Set of Hyperparameters

Epoch	Training Loss	Validation Loss	Learning Rate
1	5.5252	5.2135	0.001
2	5.3006	5.1476	0.001
3	5.2315	5.0846	0.001
4	5.1891	5.0444	0.001
5	5.1623	5.0227	0.001
6	5.1451	5.0168	0.001
7	5.1315	4.9947	0.001
8	5.1207	4.9823	0.001
9	5.1137	4.9891	0.001
10	5.1077	4.9732	0.001

Table 2: Training and Validation Loss for the Second Set of Hyperparameters

Epoch	Train Loss	Validation Loss	Learning Rate
1	4.2205	4.0775	0.001
2	4.1838	4.0615	0.001
3	4.1610	4.0379	0.001
4	4.1464	4.0398	0.001
5	4.1337	4.0074	0.001
6	4.1266	4.0097	0.001
7	4.1219	3.9965	0.001
8	4.1196	3.9951	0.001
9	4.1192	3.9914	0.001
10	4.1160	4.0040	0.001

1 Hyperparameter Tuning

In this section, we describe the different sets of hyperparameters tested during the model tuning process, followed by the selection of the optimal set.
The first set of hyperparameters were used to train the model on the semi pre-processed dataset. This dataset did not contain the special sep and endofjoke tokens, instead the json file contained only one entry per joke, where the joke was a single string pre-processed as earlier.

1.1 Hyperparameter Sets

We experimented with various sets of hyperparameters to find the optimal configuration for our model. The key parameters include the dimensions, batch size, number of heads, number of layers, and dropout rate. Table 3 lists the different configurations tested.

Table 3: Hyperparameter Sets for Model Tuning

Set	Dimensions	Batch Size	Num Heads	Num Layers	D_FF	Dropout	Learning Rate	Num Epochs	Max Seq Len
1	100	16	10	4	2048	0.2	3e-4	10	128
2	100	8	10	4	2048	0.2	1e-3	10	256

1.2 Optimal Hyperparameter Set

After evaluating the model performance with each hyperparameter set, we identified the optimal configuration that yielded the best results. The selected hyperparameters, shown in Table 4, provided the most stable and accurate model performance.
It was noted that the model performed better when the dataset structure was changed to include the special tokens. Since the jokes were in the form of title and body, the body generally contained the punchline for the joke.

Table 4: Best Hyperparameter Set

Hyperparameter	**Value**
Dimensions	100
Batch Size	8
Num Heads	10
Num Layers	4
D_FF	2048
Dropout	0.2
Learning Rate	1e-3
Num of Epochs	10
Vocab Size	10000
Max Sequence Length	256
Input Sentence Size	200000

The chosen hyperparameters in Table 4 resulted in the best model performance based on validation metrics, which included accuracy, loss, and other task-specific criteria.