NANYANG TECHNOLOGICAL UNIVERSITY

COLLEGE OF COMPUTING AND DATA SCIENCE

Project No.: SCSE23-1050

Project Title: Is sarcasm detection capability beneficial for sentiment analysis?

By: Brian Koh (U2022245E)

Abstract

This project’s main goal is to weigh the pros and cons of integrating sarcasm detection into sentiment analysis within Natural Language Processing (NLP). Sentiment Analysis aims to find out the emotional tone behind text. However, sarcasm, which often tells a meaning opposite to the meaning of the literal words, makes this task harder. This study investigates whether the integration of sarcasm detection can help improve the accuracy and robustness of sentiment analysis models. By using advanced machine learning algorithms and linguistic techniques, this research assesses the potential for sarcasm detection to lower errors in sentiment classification, therefore improving the overall effectiveness of NLP models in interpreting human emotions in text.

Acknowledgements

I would like to thank Assistant Professor (DR) Wang Wenya for her patience and support throughout this project. Her feedback and guidance were crucial to helping me complete my research on whether sarcasm detection can improve sentiment analysis. I am grateful for her time and dedication, which made a big difference in my work.

I also want to express my gratitude to the College of Computing and Data Science (CCDS) and NTU for providing me with the opportunity to work on this project. Through the modules I have gone through in the past years, I have been able to learn more about natural language processing and apply what I have learned to the real world. I have learned a lot from this research, including how sentiment analysis and sarcasm detection can work together, and I am eager to learn more about this area.

Contents

Motivation

In the past couple of years, sentiment analysis has become the go to method for understanding public views, customer feedback, and social media interactions. However, the complexity of human communication, especially the use of sarcasm, brings about different challenges to the accuracy of sentiment analysis models. Sarcasm often gives the intended sentiment the opposite sentiment, which leads to incorrect results by models that lack the ability to detect it.

Sarcasm often produces the opposite sentiment from the desired sentiment, which causes models that are unable to detect it to produce the wrong findings. The driving factor behind this project arose from the need to bridge this gap. The goal of this research is to increase the robustness and dependability of sentiment analysis models by experimenting if sarcasm detection can aid sentiment analysis. If the experiments are successful, the results may lead to more accurate ways of sentiment analysis in a variety of fields, such as social media monitoring and marketing, where determining the actual sentiment is important.

Introduction

Background

Sentiment analysis, an important part of NLP, targets to find out the emotional tone behind textual content. It is widely used in applications such as customer feedback analysis, social media monitoring, and market research to understand public opinion and sentiment. However, one significant problem in sentiment analysis is handling sarcasm. Sarcasm gives an opposite sentiment to the literal meaning of the words, making accurate sentiment detection more difficult.

Recently, large language models (LLMs) like GPT-4 have produced great results in understanding and generating human language, making them important tools for sentiment analysis [1]. These models took advantage of the large amounts of data and advanced machine learning techniques to understand context, nuances, and subtle cues in text. Despite this, LLMs still face problems trying to accurately identify sarcasm, which can result in wrong sentiment analysis results.

Sarcasm detection involves recognizing contextual and linguistic cues that show sarcasm and changing the supposed sentiment of a text. The usual sentiment analysis models, without sarcasm detection capabilities, may result in inaccurate predictions. This research aims to experiment whether integrating sarcasm detection mechanisms into sentiment analysis models, particularly those using advanced LLMs, can enhance their accuracy and robustness. Making use of sophisticated machine learning algorithms and linguistic techniques, this study aims to evaluate the potential benefits of such integration in helping the overall performance of sentiment analysis in telling apart human emotions in textual data.

Objective

The main objective of this project is to find out whether incorporating sarcasm detection components into sentiment analysis models can improve their accuracy and robustness. Through leveraging advanced machine learning techniques and linguistic analysis, the project targets to find out how much sarcasm detection can nullify misclassifications and improve the overall performance of sentiment analysis in interpreting human emotions in textual data.

Problem Statement

Sentiment analysis aims to tell the emotional tone behind textual content. However, one significant hurdle in reaching accurate sentiment analysis is the existence of sarcasm, where the intended sentiment is usually the opposite of the literal meaning. The usual sentiment analysis models find it hard to correctly tell apart sarcastic and non-sarcastic statements which often lead to misclassifications. This research aims to deal with this problem by including sarcasm detection capabilities into sentiment analysis models. The study aims to assess whether this addition can significantly improve the models' ability to accurately interpret and classify sentiments, thereby improving the reliability and effectiveness of NLP applications in understanding human emotions in text.

Literature Review

Sentiment Analysis

Sentiment analysis is a key part of NLP and is vastly being used for social media monitoring, analysing customer feedback, and powering recommendation systems. It enables businesses to gauge consumer emotions and opinions, which is needed for improved engagement. Initial approaches depended on lexicon-based methods, where sentiment dictionaries were used to assess text polarity. But these methods struggle with understanding context and the subtleties of language, especially in fast-changing spaces like social media [2].

With the rise of machine learning and, more recently, deep learning, sentiment analysis has advanced significantly. Techniques like Support Vector Machines (SVMs) and Naive Bayes were popular at first, but neural networks, including Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs), quickly proved more effective because they handle sequential information in text. Transformer-based models, like Bidirectional Encoder Representations from Transformers (BERT), have now set new standards in the field. BERT’s pre-training on massive text corpora allows it to be fine-tuned for specific sentiment analysis tasks, achieving highly accurate results [3].

Challenges and limitations

Even with all these improvements, accurately detecting sentiment is still a tough challenge, especially when it comes to tricky language features like sarcasm. Sentiment analysis models often misinterpret sarcastic comments since they may use positive words to express a negative feeling, or the other way around [4]. This issue points to the importance of adding sarcasm detection into sentiment analysis to make it more accurate, especially for real-world applications like analysing social media posts or customer reviews.

Sarcasm Detection

Sarcasm detection is a unique NLP task aimed at spotting sarcastic expressions, which can totally change the meaning of text. Often, sarcasm expresses criticism or a negative tone through what seems like positive wording, making it tough for standard sentiment models to understand accurately. Detecting sarcasm well can make a significant difference for sentiment analysis and other NLP tasks where grasping the true intent behind words matters [5].

Early sarcasm detection relied on rule-based methods, which looked for certain patterns or words, like intensifiers (e.g., “so great!”). But these methods had limits, since understanding sarcasm can require context and sometimes even cultural background. With advances in machine learning, sarcasm detection has started to use deep learning models—especially Transformers like BERT—that are better at catching subtle language clues and context [6]. Lots of recent research has focused on fine-tuning BERT for sarcasm detection, using datasets specifically created to capture sarcasm, such as collections of Twitter posts and news headlines.

Challenges and limitations

Sarcasm detection is still tough because of problems like class imbalance—sarcastic examples are usually much less common than non-sarcastic ones, which can make it hard for models to perform well and for metrics like F1 scores to be reliable. Plus, automatic labelling (like tagging posts with sarcasm hashtags) often introduces biases that do not always match the actual conversational tone, leading to inconsistent accuracy. Since sarcasm depends so much on context and subtle cues, creating big, annotated datasets that capture all its forms is a challenge. These issues make sarcasm detection models more likely to overfit to specific types of data, which limits how well they work across different contexts [7]. To tackle these problems, there’s a need for better model architectures, more reliable labelling methods, and a wider variety of training data to help make sarcasm detection models more accurate and versatile.

Integrating Sarcasm Detection into Sentiment Analysis

Many different methods have been developed to integrate sarcasm detection with sentiment analysis, with the aim to improve sentiment prediction accuracy. One traditional approach is feature engineering, where sarcasm-related features are extracted and combined with sentiment indicators. However, because sarcasm is highly context-dependent, this method has not been highly effective. More recent strategies use deep learning models to tackle sarcasm detection and sentiment analysis together, enabling the model to capture a shared representation of both sentiment and sarcasm-specific features [8].

A creative example of this integration is shown in multi-modal sarcasm detection models, which leverage both visual and textual data to capture complex sarcasm signals [9]. For instance, one model uses a Visual Transformer (ViT) to encode image features while processing text with a bidirectional LSTM network, which captures sequential nuances. Additionally, it incorporates a Sentiment Vector that embeds emotionally charged words from both image and text modalities with help from SenticNet [10], providing essential emotional context for sarcasm detection. The model's attention fusion module then aligns relevant image and text features with sarcasm markers through a multi-head attention mechanism. This multi-modal approach illustrates how combining different types of data can improve sentiment analysis by addressing the unique, context-heavy nature of sarcasm, showing the potential of fusion techniques to tackle sarcasm's complexity effectively.

Data Collection

The Sarcasm Detection dataset comes from Kaggle's News Headlines Dataset for Sarcasm Detection and is stored in a JSON file named Sarcasm_Headlines_Dataset, containing about 28,619 entries. Each entry includes a news headline and a sarcasm label, where a label of 1 signifies sarcasm and 0 indicates no sarcasm. This dataset also provides a link to the original news article, which can be useful for gathering additional context or data [11].

For Sentiment Analysis, data was gathered from the Twitter and Reddit Sentiment Analysis Dataset on Kaggle. This dataset initially consisted of two CSV files—Twitter.csv with around 163,000 tweets and Reddit.csv with roughly 37,000 comments. For this project, only the Twitter dataset was used, which includes columns for the cleaned text and a sentiment label that indicates the emotional tone: 0 for neutral, 1 for positive, and -1 for negative sentiment [12].

Data Preprocessing

Data Cleaning:

For sarcasm detection, the data was converted to strings and NaN values were replaced with empty strings. Contractions were expanded to their full forms for text standardization. HTML tags and URLs were removed, as they do not contribute to the understanding of sarcasm. Punctuation was also removed, and any empty data was removed, ensuring that the dataset was clean and ready for analysis.

For the sentiment analysis dataset, several preprocessing steps were applied to prepare the text data for model training. Similar to sarcasm detection data, the sentiment analysis data was first converted to strings, and any NaN values were replaced with empty strings to ensure consistency. Mentions, URLs, special characters, and numbers were removed to get rid of noise from the data. The text was then converted to lowercase to maintain consistency. Stop words, which do not contribute significant meaning, were also removed, followed by the elimination of any empty data entries. Finally, lemmatization was applied to reduce words to their base forms, aiding in the normalization of the text.

Data Transformation:

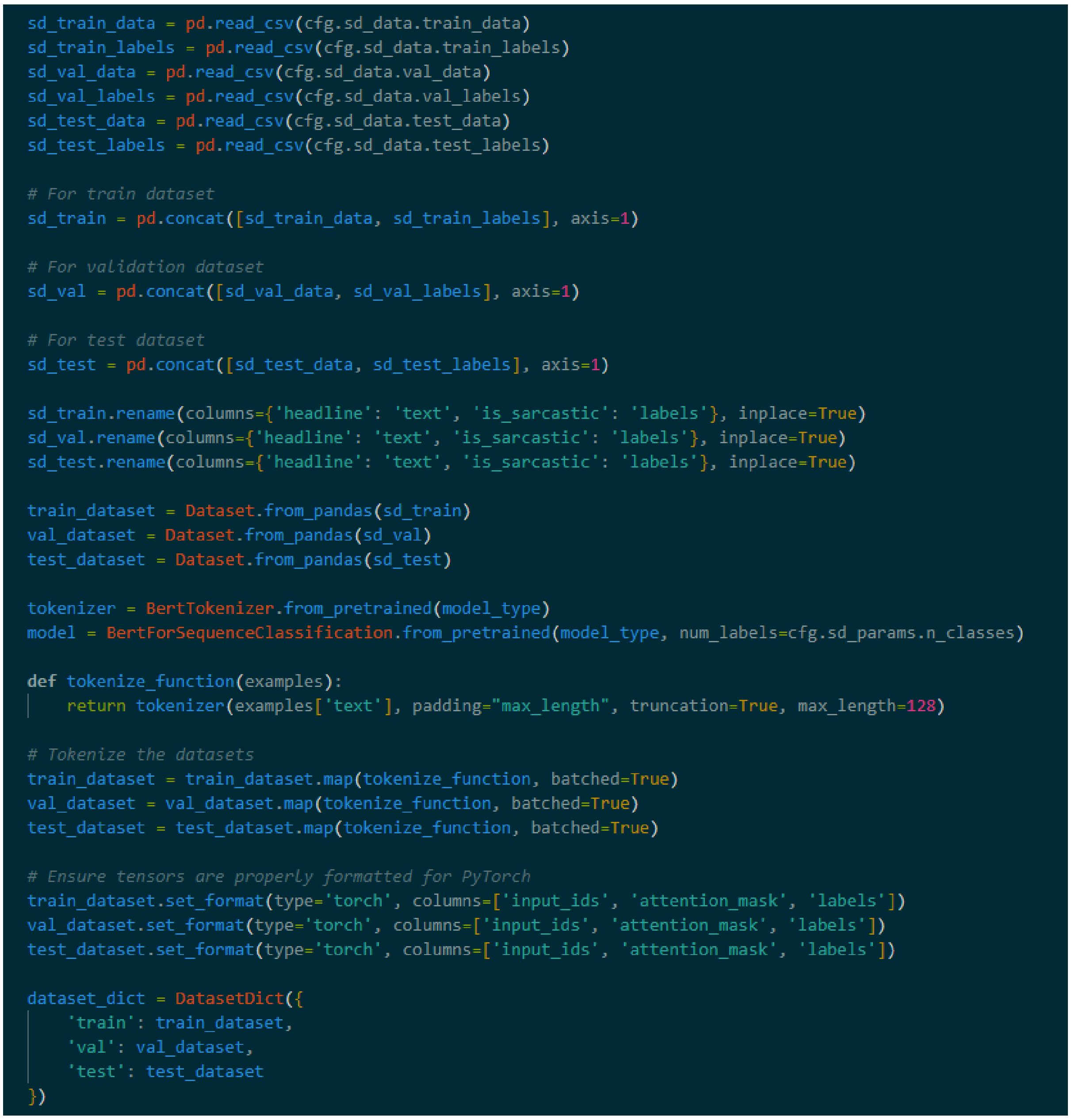
During data preprocessing, I also applied some transformation specifically to the sentiment analysis dataset. Initially, the dataset has marked the negative sentiments with a label of -1. However, to keep things consistent and prevent any confusion during training, I changed these -1 labels to 2 to represent negative sentiments. This adjustment was important for standardizing the labels so that each sentiment class was clearly defined and easy for the model to interpret.

Data Preprocessing for Compatibility with Hugging Face Models

To prepare the data for Hugging Face models, I came up with a series of preprocessing steps for both sarcasm detection and sentiment analysis datasets. For sarcasm detection, I combined the training, validation, and test sets with their labels to align the data correctly, renaming the 'headline' column to 'text' and 'is_sarcastic' to 'labels' for consistency with model input requirements.

The sentiment analysis dataset, containing tweets and comments, went through similar formatting steps to ensure compatibility with Hugging Face’s setup. After converting both datasets into Hugging Face Dataset objects, I used the tokenizer from the pre-trained 'prajjwal1/bert-tiny' model to tokenize them, padding and truncating each entry to a maximum of 128 tokens for consistency.

Lastly, I formatted the tokenized datasets for PyTorch, setting up the input_ids, attention_mask, and labels as tensors. I organized everything into a DatasetDict with training, validation, and test sets. This pipeline made sure both datasets were in their correct format and ready for fine-tuning on their specific tasks.



```python
sd_train_data = pd.read_csv(cfg.sd_data.train_data)
sd_train_labels = pd.read_csv(cfg.sd_data.train_labels)
sd_val_data = pd.read_csv(cfg.sd_data.val_data)
sd_val_labels = pd.read_csv(cfg.sd_data.val_labels)
sd_test_data = pd.read_csv(cfg.sd_data.test_data)
sd_test_labels = pd.read_csv(cfg.sd_data.test_labels)

# For train dataset
sd_train = pd.concat([sd_train_data, sd_train_labels], axis=1)

# For validation dataset
sd_val = pd.concat([sd_val_data, sd_val_labels], axis=1)

# For test dataset
sd_test = pd.concat([sd_test_data, sd_test_labels], axis=1)

sd_train.rename(columns={'headline': 'text', 'is_sarcastic': 'labels'}, inplace=True)
sd_val.rename(columns={'headline': 'text', 'is_sarcastic': 'labels'}, inplace=True)
sd_test.rename(columns={'headline': 'text', 'is_sarcastic': 'labels'}, inplace=True)

train_dataset = Dataset.from_pandas(sd_train)
val_dataset = Dataset.from_pandas(sd_val)
test_dataset = Dataset.from_pandas(sd_test)

tokenizer = BertTokenizer.from_pretrained(model_type)
model = BertForSequenceClassification.from_pretrained(model_type, num_labels=cfg.sd_params.n_classes)

def tokenize_function(examples):
    return tokenizer(examples['text'], padding="max_length", truncation=True, max_length=128)

# Tokenize the datasets
train_dataset = train_dataset.map(tokenize_function, batched=True)
val_dataset = val_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)

# Ensure tensors are properly formatted for PyTorch
train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
val_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
test_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])

dataset_dict = DatasetDict({
    'train': train_dataset,
    'val': val_dataset,
    'test': test_dataset
})
```

Fig 1: Code snippet of data preprocessing

Exploratory Data Analysis

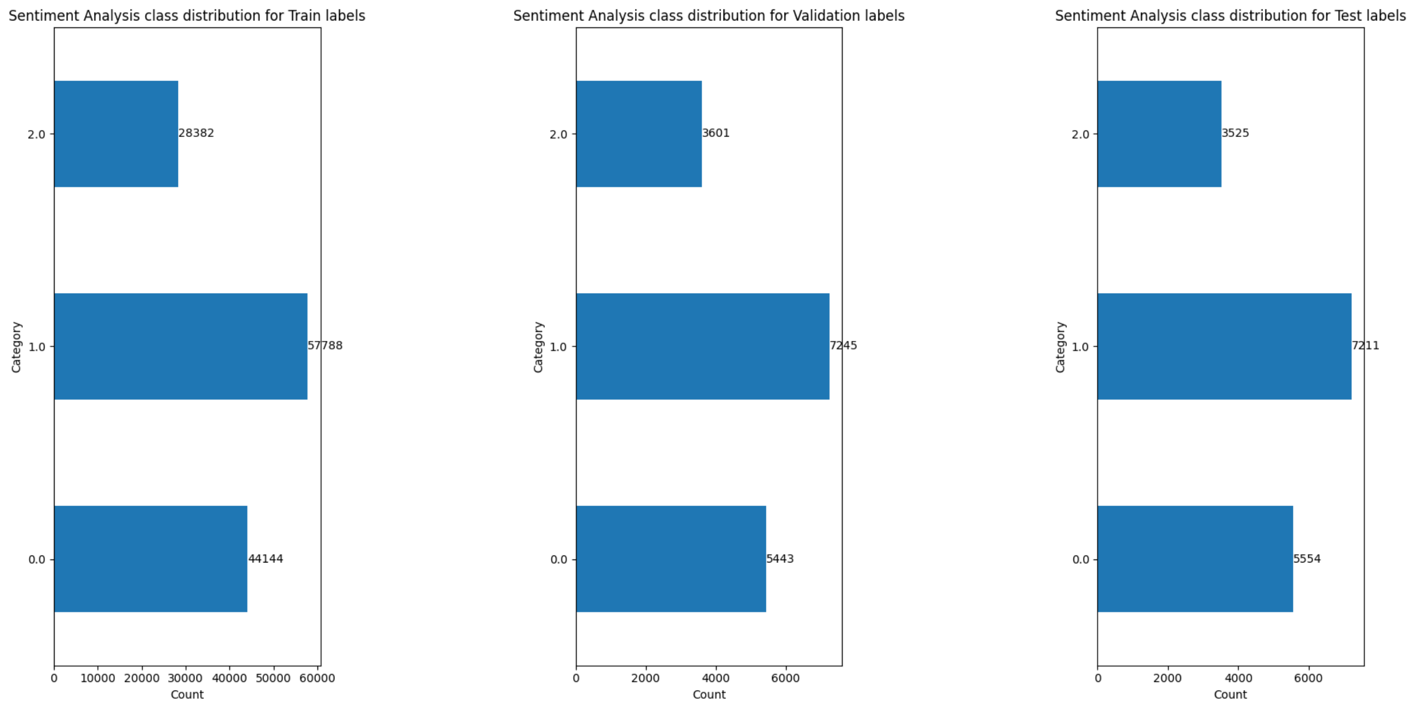


Figure 2.1: Data Distribution for Sentiment Analysis

The Sentiment Analysis dataset shows a diverse distribution of labels, with about 35,508 entries labelled as negative, 55,141 labelled as neutral, and 72,244 labelled as positive. This distribution is sufficiently balanced to provide a comprehensive understanding of the different sentiment categories.

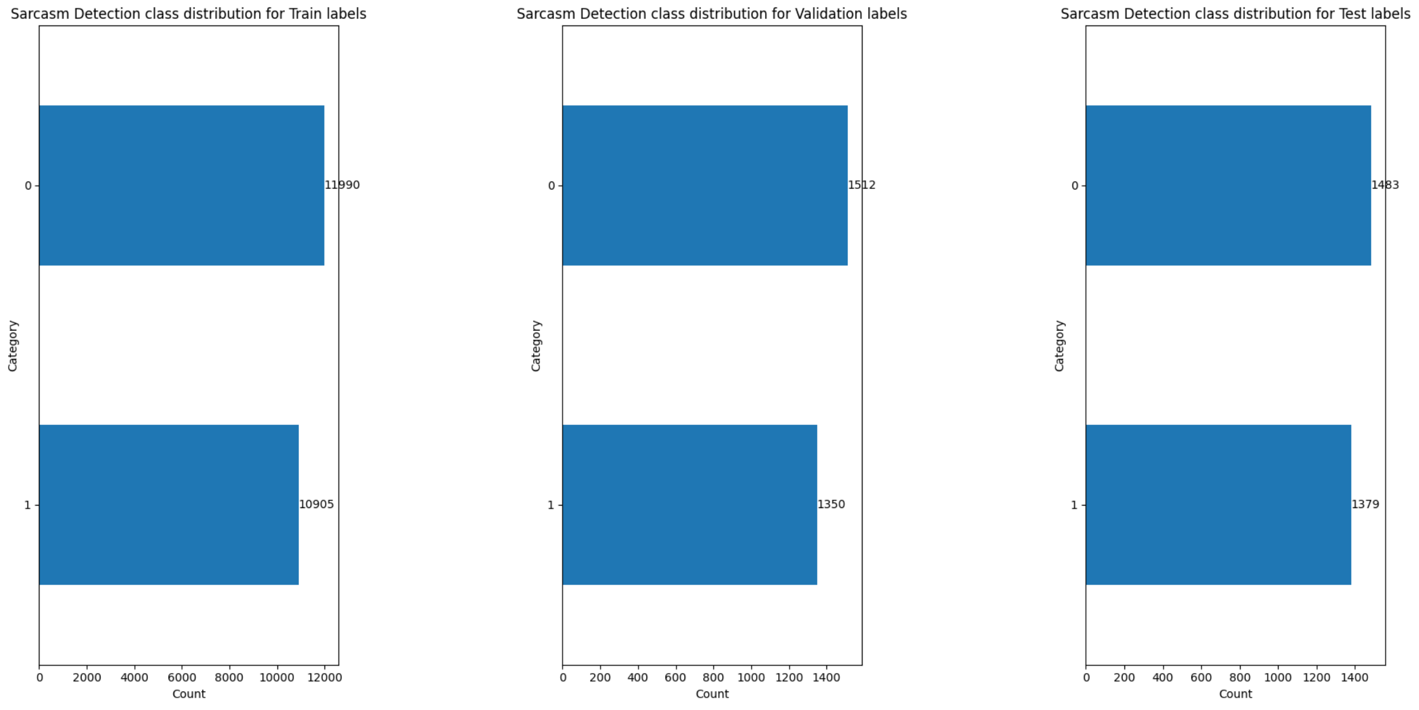


Figure 2.2: Data Distribution for Sarcasm Detection

The Sarcasm Detection dataset, the label distribution shows that out of 28,619 records, about 14,985 are non-sarcastic, and 13,634 are sarcastic. This close balance between sarcastic and non-sarcastic labels is ideal for training a sarcasm detection model, as it means there is no need for additional adjustments or balancing of the dataset.

Model Selection

Rationale for Using BERT in Sarcasm Detection and Sentiment Analysis

BERT has become one of the more powerful models in natural language processing due to its ability to understand language in context. Unlike older models where they treat words as isolated units or rely on fixed embeddings, BERT analyses words by looking at the entire sentence, processing them in both directions. This bidirectional approach allows BERT to capture the subtle meanings of words depending on their context, which is especially important for tasks like sarcasm detection, where the true sentiment often contradicts the literal words used.

For tasks like sarcasm detection and sentiment analysis, understanding these shifts in meaning is crucial. Sarcasm often involves a contradiction in sentiment, making it hard for simpler models to identify. Sentiment analysis also requires a deep understanding of the emotional tone behind the words. BERT's ability to pick up on these nuances makes it ideal for both tasks. In addition, BERT's extensive pre-training on large datasets gives it a strong background on language patterns, making it an excellent starting point that can be fine-tuned for specific tasks like sarcasm detection and sentiment analysis.

Why Tiny BERT is a Practical Choice

While BERT offers significant advantages in understanding language, it is also a model that uses up lots of resources, requiring lots of computational power for both training and inference. This is where Tiny BERT [1], a smaller and more efficient version of BERT can be used. Tiny BERT is built such that the model size and computational demands are reduced without compromising the performance too much. It is able to do this through a process known as knowledge distillation, where the smaller model is trained to mimic the behaviour of the larger BERT model.

For tasks like sarcasm detection and sentiment analysis, which usually comprises of shorter texts like tweets or headlines, the full capacity of a standard BERT model might be too much. Tiny BERT, with its reduced complexity, serves as an ideal balance between efficiency and accuracy. It offers enough capacity to handle the intricacies of these tasks while being more practical for deployment in environments with limited computational resources. This makes Tiny BERT a great choice, providing robust performance without the potential overkill of using a full-sized BERT model.

Modelling Approach 1: Separate Models for Sarcasm Detection and Sentiment Analysis

Sarcasm Detection Model

For the sarcasm detection model, the process began with fine-tuning a pre-trained Tiny BERT model to classify text as either sarcastic or non-sarcastic. The dataset used for this task was labelled specifically for sarcasm detection, with each example marked as either 1 (sarcastic) or 0 (non-sarcastic). During fine-tuning, the model's weights were adjusted to minimize the classification loss, aiming to accurately identify sarcastic remarks. Key hyperparameters, such as the learning rate, batch size, and number of epochs, were carefully tuned to maximize the model's performance. Regular validation was conducted to track the model's progress and prevent overfitting.

Sentiment Analysis Model

In a similar fashion, the sentiment analysis model was fine-tuned to classify text into three sentiment categories: positive, neutral, and negative. The sentiment dataset had been labelled with 1 for positive, 0 for neutral, and 2 for negative sentiments. During the fine-tuning process, the model's parameters were optimized for this multi-class classification task, using categorical cross-entropy loss to guide training. The model's performance was evaluated based on its accuracy across these sentiment categories. Like the sarcasm detection model, frequent validation checks ensured the model generalized well to new data and avoided overfitting to the training set.

Both models underwent careful fine-tuning to adapt Tiny BERT’s general language understanding to the specific requirements of sarcasm detection and sentiment analysis. This process was crucial for aligning the models’ predictions with the unique features of each task while leveraging the pre-trained knowledge in the most effective way.

Model Evaluation

Task	**Model Type**	**Epochs Trained**	**Batch Size**	**Weight Decay**	**Learning Rate**	**Model Accuracy**
Sentiment Analysis	Bert-Tiny	10	16	0.03	0.000015	89.239
Sarcasm Detection	Bert-Tiny	13	16	0.04	0.000015	87.317

The evaluation of the models for sentiment analysis and sarcasm detection shows robust performance and highlighted the effectiveness of the fine-tuning process. The sentiment analysis model which is based on the Bert-Tiny architecture, was trained for 10 epochs with a batch size of 16 and a weight decay of 0.03, achieving an accuracy of **89.239%**. This result displays the model's strong capability in classifying sentiment across the dataset. Similarly, the sarcasm detection model, also utilizing the Bert-Tiny architecture, was trained for 13 epochs with a batch size of 16 and a slightly higher weight decay of 0.04. This model achieved an accuracy of **87.317%**, which also shows its proficiency in detecting sarcasm within the data. These accuracy metrics reflect the reliability of the models in their respective tasks, validating the fine-tuning approach and parameter choices made during training.

Transfer Learning

The transfer learning process involves leveraging both the fine-tuned sarcasm detection model and the sentiment analysis model. At the start, the sentiment analysis model was set to train mode while the sarcasm detection model was set to evaluation mode to ensure that the sarcasm detection model’s weights do not change during the training of the sentiment analysis model. The sentiment analysis model is then trained using features extracted from both models, while keeping the sarcasm detection model frozen.

To begin, DataLoader instances are created for the training, validation, and test datasets. Next, the optimizer for the sentiment analysis model is defined as AdamW. The loss function used is CrossEntropyLoss.

During the training phase, the model iterates through the epochs specified. In each epoch, the training loop extracts features from both the sarcasm detection and sentiment analysis models. The features extracted are the last hidden states from both models. After extracting the last hidden states from both models, several different feature integration techniques were experimented with. These included: (1) Averaging the last hidden states from both models and then extracting the [CLS] token's hidden state before passing it through the sentiment model's classifier. (2) Averaging the last hidden states from both models, performing max pooling, and then passing the pooled representation through the sentiment model's classifier. (3) Concatenating the last hidden states from both models, extracting the [CLS] token's hidden state, and passing it through the sentiment model's classifier. (4) Concatenating the last hidden states from both models, performing max pooling, and passing this pooled representation through the sentiment model's classifier. (5)

Concatenating the last hidden states from both models, performing mean pooling, and passing the pooled features through the sentiment model's classifier to computer logits.

The loss is then computed based on the logits and the true labels, and backpropagation is performed to update the sentiment analysis model's weights. The training loop also monitors the loss and accuracy numbers for each batch, while constantly updating these metrics as the training progresses. After each epoch, the model is evaluated on the validation set to check its performance.

The evaluation phase involves calculating the loss and accuracy on both the validation and test datasets. The same feature extraction process is applied where combined features from both models are used to make predictions with the sentiment analysis model.

This approach of combining features from two models aims to enhance the performance of sentiment analysis by leveraging the strengths of both models. By keeping the sarcasm detection model’s weights frozen and focusing on fine-tuning the sentiment analysis model, these methods explore the potential benefits of integrating features from specialized models into the main classification task.

(Training code snippets are in the Appendix)

Integration Technique	**Epochs Trained**	**Batch Size**	**Weight Decay**	**Learning Rate**	**Test Accuracy**
Averaging the last hidden states from both models and then extracting the [CLS] token's hidden state	15	16	0.03	0.0000015	88.932
Averaging the last hidden states from both models, performing max pooling	15	16	0.03	0.0000015	87.980
Concatenating the last hidden states from both models, extracting the [CLS] token's hidden state	15	16	0.03	0.0000015	89.165
Concatenating the last hidden states from both models, performing max pooling	15	16	0.03	0.0000015	88.521
Concatenating the last hidden states from both models, performing mean pooling	20	16	0.04	0.0000015	87.815

The transfer learning approach made use of both the pre-trained sarcasm detection and sentiment analysis models, both based on the Bert-Tiny architecture, to improve sentiment analysis performance. The sentiment analysis model, fine-tuned with the integration method of concatenating the last hidden states from both models followed by extracting the [CLS] token's hidden state yields an accuracy of **89.165%**. In contrast, the pure sentiment analysis model has an accuracy of **89.239%**. Although transfer learning involved a different training configuration, the results indicate that the pure sentiment analysis model outperformed the transfer learning model by a small margin.

Modelling Approach 2: Single Backbone Model with Separate Top Classification Layers

Multitask Learning

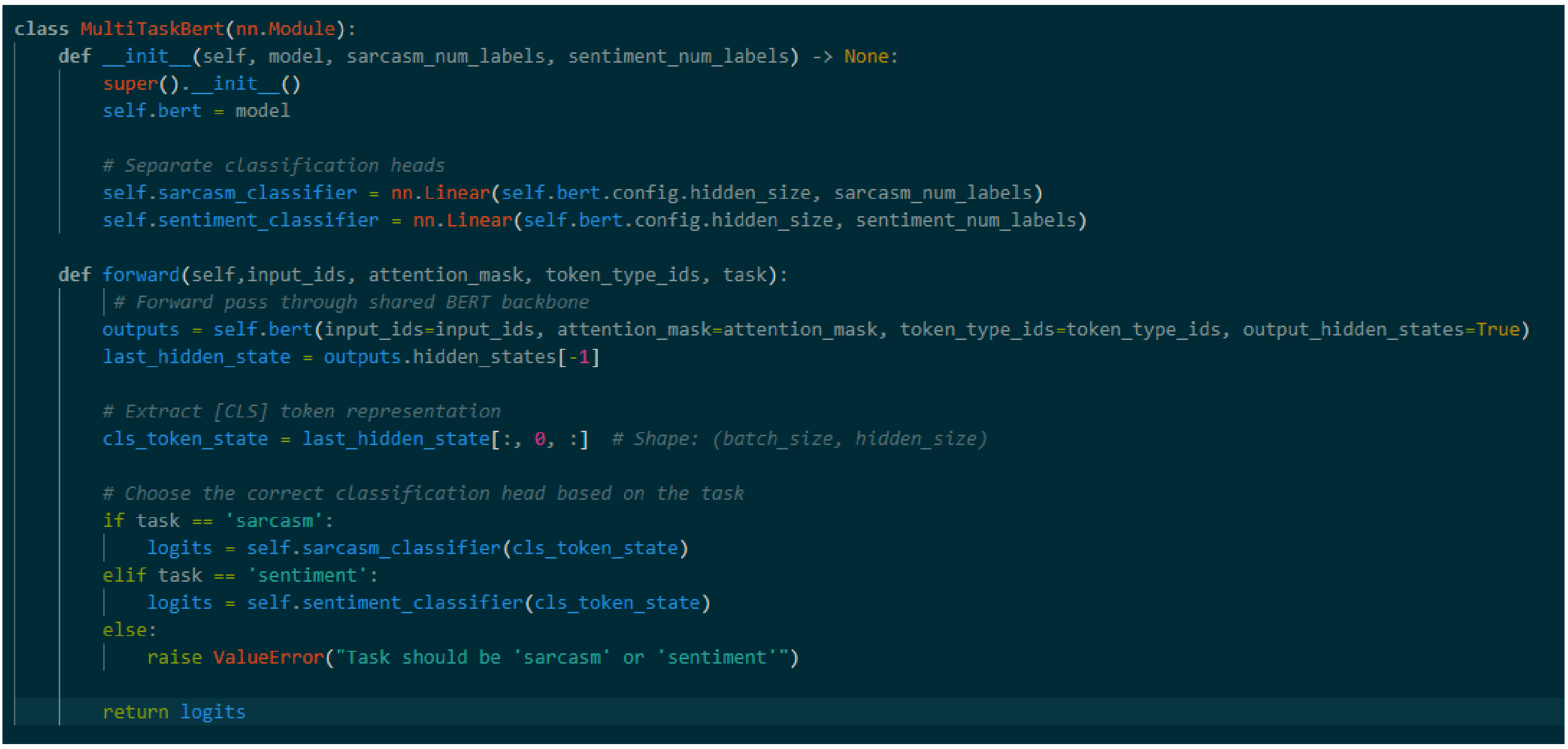
Multitask learning is a machine learning approach where a model is trained on multiple tasks at the same time. This allows it to share knowledge across tasks that may have underlying similarities [13]. Through learning these shared representations, multitask learning can improve generalization and results on individual tasks, often outperforming separate individual models trained for each task, as in modelling approach 1. In this project, multitask learning is applied to train a model on both sarcasm detection and sentiment analysis. Since both tasks require the model to understand nuances in meaning, tone, and context, they are well-suited for this shared learning setup.

Multitask Learning Approach

In the provided code below, a multitask model is designed using a shared Tiny-Bert model as the backbone, with two separate classification heads for sarcasm detection and sentiment analysis. The core of the implementation involves alternating between processing batches from the sarcasm detection and sentiment analysis datasets, training the shared model this way ensures that the model learns from both tasks during each epoch. The code integrates the two tasks by using the following components:

Shared Representation

The backbone of the model is a pre-trained Tiny-Bert model, which processes input text and outputs a set of embeddings that serve as shared representations for both tasks. The code captures this in a custom MultiTaskBert class as shown below, which uses a common feature extractor for both tasks.



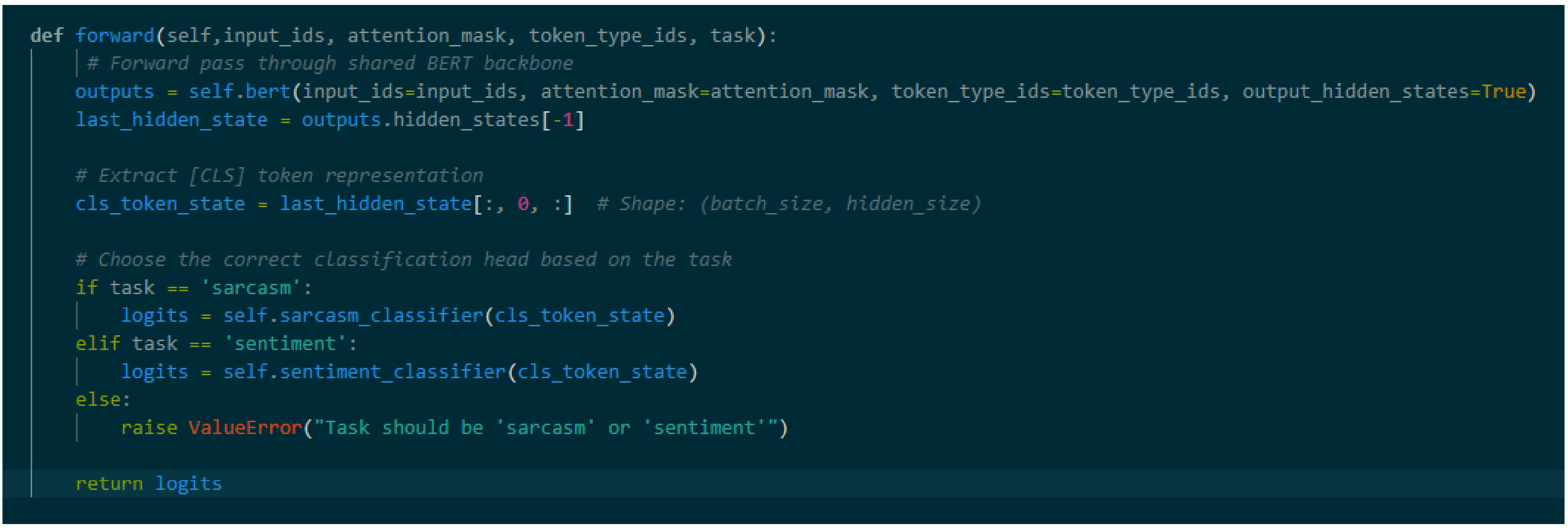
```python
class MultiTaskBert(nn.Module):
    def __init__(self, model, sarcasm_num_labels, sentiment_num_labels) -> None:
        super().__init__()
        self.bert = model

        # Separate classification heads
        self.sarcasm_classifier = nn.Linear(self.bert.config.hidden_size, sarcasm_num_labels)
        self.sentiment_classifier = nn.Linear(self.bert.config.hidden_size, sentiment_num_labels)

    def forward(self,input_ids, attention_mask, token_type_ids, task):
        # Forward pass through shared BERT backbone
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids, output_hidden_states=True)
        last_hidden_state = outputs.hidden_states[-1]

        # Extract [CLS] token representation
        cls_token_state = last_hidden_state[:, 0, :]  # Shape: (batch_size, hidden_size)

        # Choose the correct classification head based on the task
        if task == 'sarcasm':
            logits = self.sarcasm_classifier(cls_token_state)
        elif task == 'sentiment':
            logits = self.sentiment_classifier(cls_token_state)
        else:
            raise ValueError("Task should be 'sarcasm' or 'sentiment'")

        return logits
```

Figure 3.1: MultiTaskBert class

In this code screenshot above, self.bert is the shared Tiny-Bert backbone, while self.sa_classifier and self.sd_classifier are the task-specific classification heads for sentiment analysis and sarcasm detection, respectively.

Task-Specific Heads

The sarcasm detection task and sentiment analysis task both has its own classification head. The sa_classifier is tasked to predict the sentiment label, while the sd_classifier is tasked to predict the sarcasm label. The classification heads are defined as nn.Linear layers, which take the output embeddings from the shared backbone and put them through to their respective task outputs.



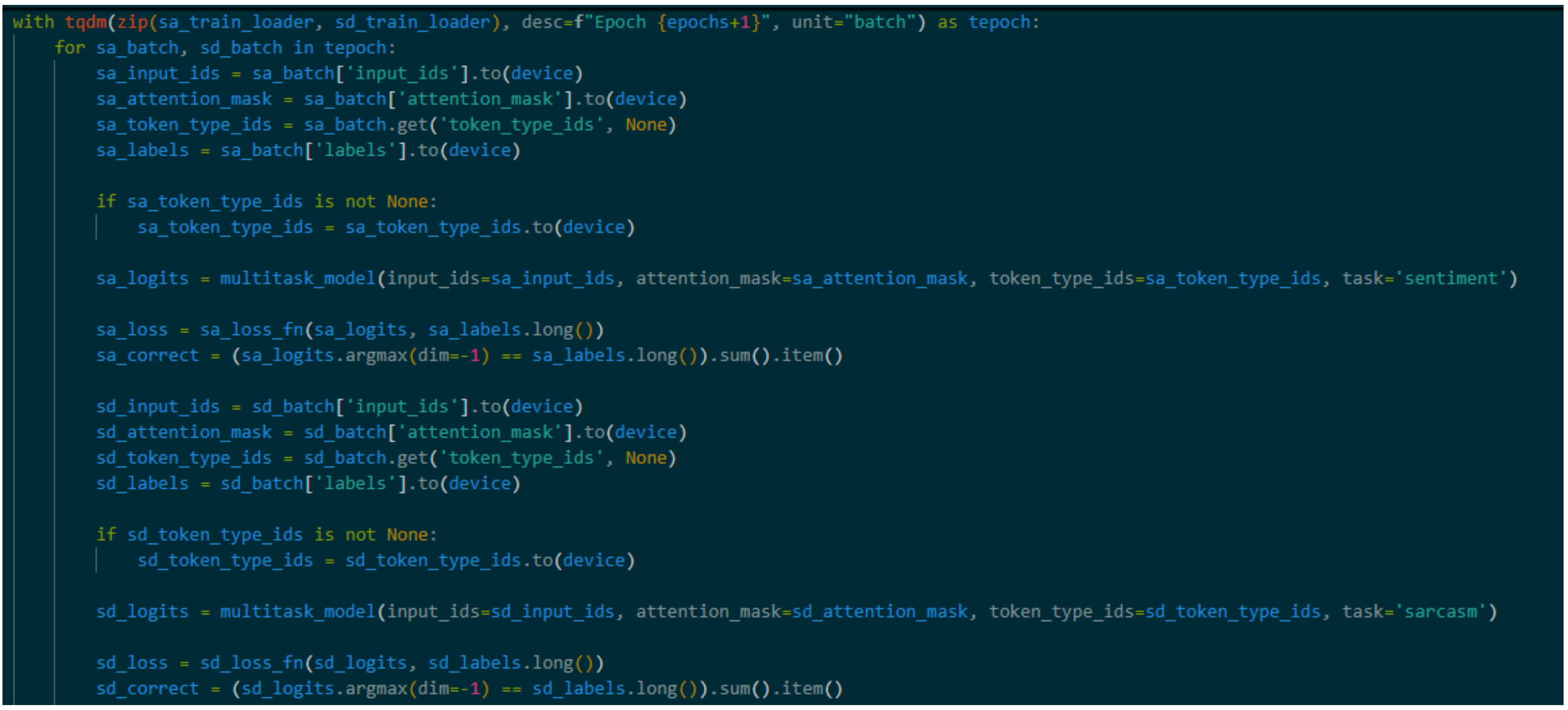
```python
def forward(self,input_ids, attention_mask, token_type_ids, task):
    # Forward pass through shared BERT backbone
    outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids, output_hidden_states=True)
    last_hidden_state = outputs.hidden_states[-1]

    # Extract [CLS] token representation
    cls_token_state = last_hidden_state[:, 0, :]  # Shape: (batch_size, hidden_size)

    # Choose the correct classification head based on the task
    if task == 'sarcasm':
        logits = self.sarcasm_classifier(cls_token_state)
    elif task == 'sentiment':
        logits = self.sentiment_classifier(cls_token_state)
    else:
        raise ValueError("Task should be 'sarcasm' or 'sentiment'")

    return logits
```

Figure 3.2: Forward function in MultiTaskBert class

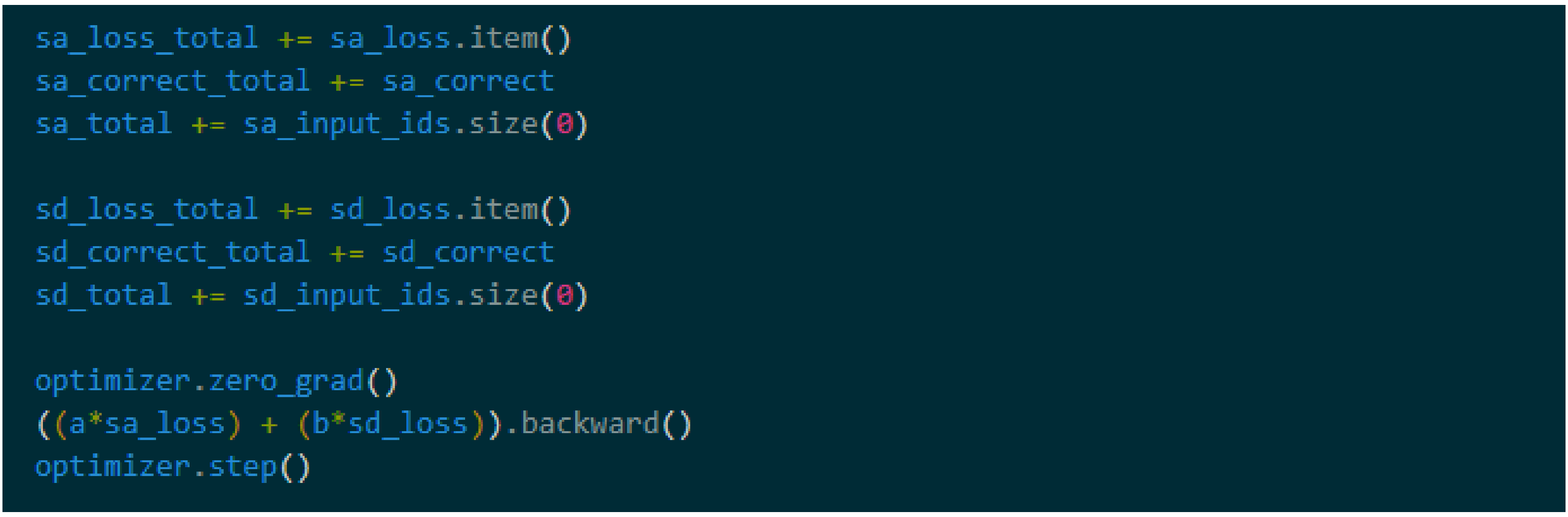
Here, depending on the task (either "sentiment" for sentiment analysis or "sarcasm" for sarcasm detection), the model forwards the embeddings to the corresponding classification head.

Loss Calculation

The code calculates the task specific loss for both sentiment analysis and sarcasm detection. Each task has its own loss function (CrossEntropyLoss), which measures how well the model’s predictions matches with the true labels for that task. The total loss during training is computed by adding the losses from both tasks.



```python
with tqdm(zip(sa_train_loader, sd_train_loader), desc=f"Epoch {epochs+1}", unit="batch") as tepoch:
    for sa_batch, sd_batch in tepoch:
        sa_input_ids = sa_batch['input_ids'].to(device)
        sa_attention_mask = sa_batch['attention_mask'].to(device)
        sa_token_type_ids = sa_batch.get('token_type_ids', None)
        sa_labels = sa_batch['labels'].to(device)

        if sa_token_type_ids is not None:
            sa_token_type_ids = sa_token_type_ids.to(device)

        sa_logits = multitask_model(input_ids=sa_input_ids, attention_mask=sa_attention_mask, token_type_ids=sa_token_type_ids, task='sentiment')

        sa_loss = sa_loss_fn(sa_logits, sa_labels.long())
        sa_correct = (sa_logits.argmax(dim=-1) == sa_labels.long()).sum().item()

        sd_input_ids = sd_batch['input_ids'].to(device)
        sd_attention_mask = sd_batch['attention_mask'].to(device)
        sd_token_type_ids = sd_batch.get('token_type_ids', None)
        sd_labels = sd_batch['labels'].to(device)

        if sd_token_type_ids is not None:
            sd_token_type_ids = sd_token_type_ids.to(device)

        sd_logits = multitask_model(input_ids=sd_input_ids, attention_mask=sd_attention_mask, token_type_ids=sd_token_type_ids, task='sarcasm')

        sd_loss = sd_loss_fn(sd_logits, sd_labels.long())
        sd_correct = (sd_logits.argmax(dim=-1) == sd_labels.long()).sum().item()
```

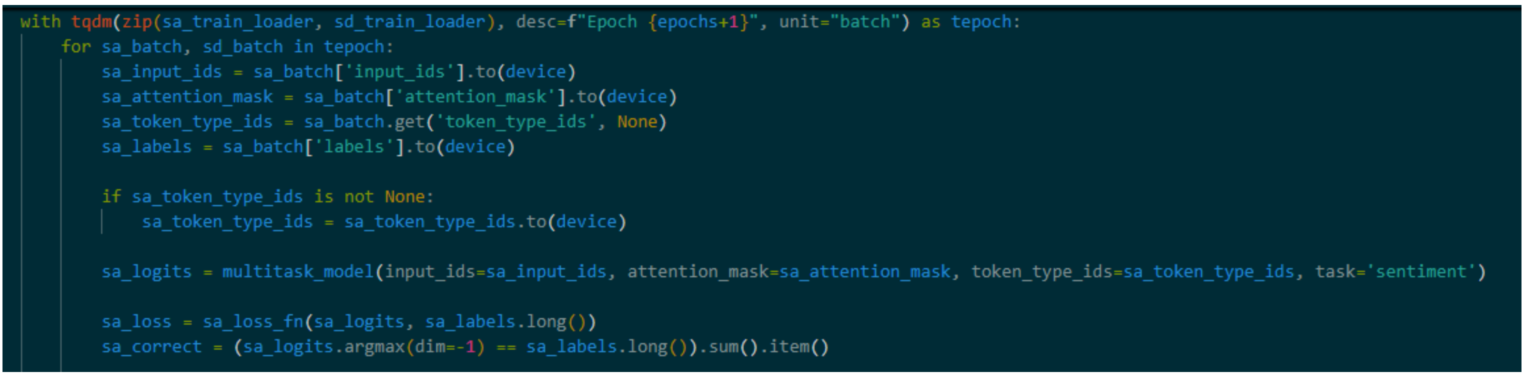


```python
sa_loss_total += sa_loss.item()
sa_correct_total += sa_correct
sa_total += sa_input_ids.size(0)

sd_loss_total += sd_loss.item()
sd_correct_total += sd_correct
sd_total += sd_input_ids.size(0)

optimizer.zero_grad()
((a*sa_loss) + (b*sd_loss)).backward()
optimizer.step()
```

Figure 3.3 and 3.4: Loss Calculation in the training loop

Alternating Task Training

A key aspect of the multitask learning approach implemented in the code is the alternating switch of processing of both tasks during training. For each iteration in an epoch, the model processes a batch from both the sarcasm detection and sentiment analysis datasets, ensuring that the shared Tiny-Bert model is updated based on knowledge from both tasks.



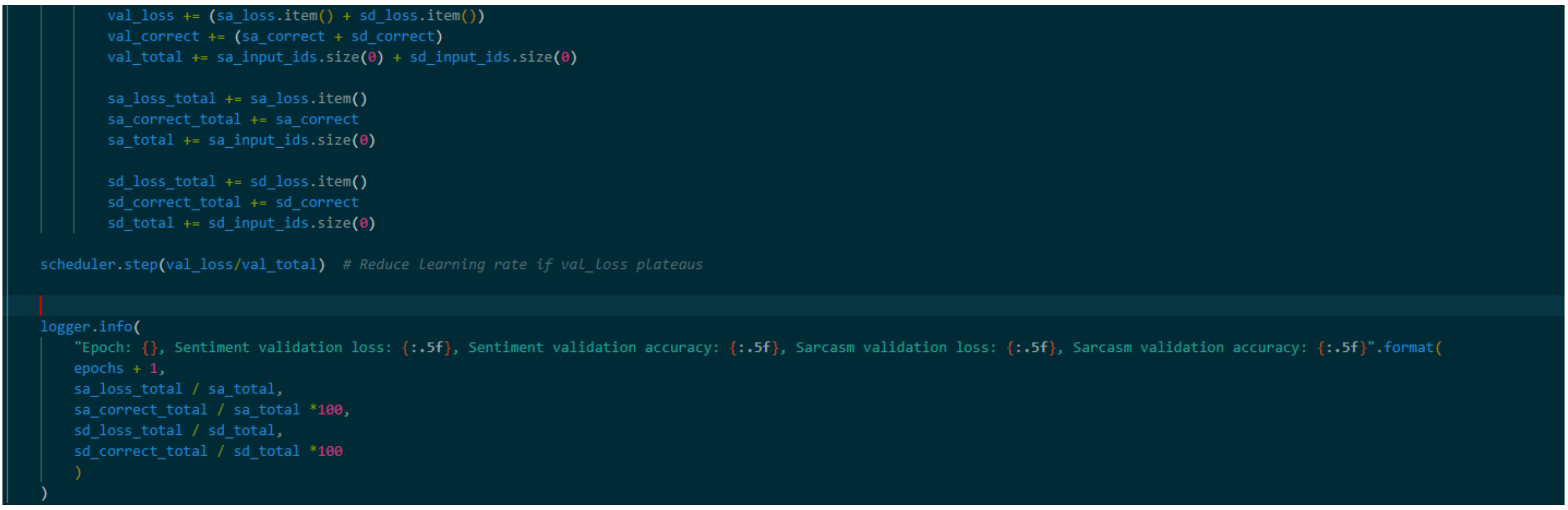
```
with tqdm(zip(sa_train_loader, sd_train_loader), desc=f"Epoch {epochs+1}", unit="batch") as tepoch:
    for sa_batch, sd_batch in tepoch:
        sa_input_ids = sa_batch['input_ids'].to(device)
        sa_attention_mask = sa_batch['attention_mask'].to(device)
        sa_token_type_ids = sa_batch.get('token_type_ids', None)
        sa_labels = sa_batch['labels'].to(device)

        if sa_token_type_ids is not None:
            sa_token_type_ids = sa_token_type_ids.to(device)

        sa_logits = multitask_model(input_ids=sa_input_ids, attention_mask=sa_attention_mask, token_type_ids=sa_token_type_ids, task='sentiment')

        sa_loss = sa_loss_fn(sa_logits, sa_labels.long())
        sa_correct = (sa_logits.argmax(dim=-1) == sa_labels.long()).sum().item()
```

Figure 3.5: Alternating of sarcasm and sentiment batches

By switching between sarcasm detection and sentiment analysis batches, the model can learn more robust representations that are relevant to both tasks. The shared learning helps improve generalization, especially for sentiment analysis, which can benefit from the model’s understanding of sarcasm.

Task-Specific Evaluation

During the validation loop, the model is evaluated on different validation datasets for both sarcasm detection and sentiment analysis. The code shows that the predictions for each task are calculated independently, and the respective validation losses and accuracy scores are tracked independently as well.



```
        val_loss += (sa_loss.item() + sd_loss.item())
        val_correct += (sa_correct + sd_correct)
        val_total += sa_input_ids.size(0) + sd_input_ids.size(0)

        sa_loss_total += sa_loss.item()
        sa_correct_total += sa_correct
        sa_total += sa_input_ids.size(0)

        sd_loss_total += sd_loss.item()
        sd_correct_total += sd_correct
        sd_total += sd_input_ids.size(0)

scheduler.step(val_loss/val_total)  # Reduce learning rate if val_loss plateaus


logger.info(
    "Epoch: {}, Sentiment validation loss: {:.5f}, Sentiment validation accuracy: {:.5f}, Sarcasm validation loss: {:.5f}, Sarcasm validation accuracy: {:.5f}".format(
    epochs + 1,
    sa_loss_total / sa_total,
    sa_correct_total / sa_total *100,
    sd_loss_total / sd_total,
    sd_correct_total / sd_total *100
    )
)
```

Figure 3.6: Evaluation on separate validation datasets

This ensures that the model's performance on each task is monitored separately, even though the model shares a backbone for both tasks.

Model Evaluation

Task	**Model Type**	**Epochs Trained**	**Batch Size**	**Weight Decay**	**Learning Rate**	**alpha**	**beta**	**Model Accuracy**
Multi-Task Learning	Bert-Tiny	15	16	0.04	0.00003	1.7	0.3	90.049

The multitask model, trained on both sarcasm detection and sentiment analysis tasks, achieved an accuracy of **90.049%** on the sentiment analysis test dataset. This is in comparison to the pure sentiment analysis model, which achieved **89.239%** accuracy.

This demonstrates an improvement of about **0.81%**, suggesting that the multitask approach, which is integrated with sarcasm detection features, is effective at enhancing sentiment analysis accuracy.

Explanation of Alpha and Beta:

In this multitask learning setup, the model optimizes for both sarcasm detection and sentiment analysis by minimizing a combined loss function, which is a weighted sum of each task's individual losses. Two weights are used here to control each task's influence on the total loss:

- **Alpha (α = 1.7):** This weight adjusts the sentiment analysis loss's contribution to the total loss.

- **Beta (β = 0.3):** This weight adjusts the sarcasm detection loss’s contribution, with a lower beta value reducing its impact relative to the sentiment loss.

Tuning these weights allows for control over the model's focus between the two tasks. By setting alpha higher, this approach prioritizes sentiment analysis while still drawing on the shared insights gained from sarcasm detection.

Evaluation of Multitask Learning Compared to Single-Task Model

The multitask learning approach shows a modest but notable improvement in sentiment analysis accuracy, increasing from 89.239% to 90.049%. Even though this improvement is small, it suggests that incorporating sarcasm detection has indeed helped the sentiment analysis model to better capture complex language nuances, especially when dealing with sarcastic expressions.

This marginal increase indicates that while sarcasm detection may not be the dominant factor in sentiment prediction, it still offers valuable insights that enhance the sentiment analysis model’s generalization. The multitask model likely benefits from shared language features between sarcasm detection and sentiment analysis, aiding in recognizing subtleties such as irony or nuanced negativity that might otherwise be misclassified without the sarcasm detection context.

Conclusion

The main goal of this project was to assess whether incorporating sarcasm detection could improve sentiment analysis accuracy. To do this, I explored two distinct approaches: transfer learning with various integration techniques and multitask learning. Both approaches aimed to take advantage of the connection between sarcasm detection and sentiment analysis to enhance the accuracy and reliability of sentiment predictions.

First Approach: Transfer Learning with Different Integration Techniques

In the first approach, I fine-tuned two Tiny-BERT models separately. One for sarcasm detection and the other for sentiment analysis. After training, I tried different ways to combine the models, such as concatenating their output layers and passing the combined features through a classifier focused on sentiment prediction.

This transfer learning setup allowed the sentiment model to incorporate some sarcasm-related features, but the improvement in accuracy was fairly modest. While the model gained some benefit from the shared representation between sarcasm and sentiment, the integration techniques didn’t lead to a substantial boost in performance over the sentiment-only model.

Second Approach: Multitask Learning

In the second approach, I used multitask learning where I trained a single Tiny-Bert model with two separate classification heads: one for sarcasm detection and one for sentiment analysis. This structure allowed the model to share lower-level representations while optimizing the tasks at the same time. I introduced a weighted loss function, where I used alpha (1.7) and beta (0.3) to control the balance between the sentiment and sarcasm losses, with a higher emphasis on the sentiment analysis task.

This multi-task learning approach achieved a test accuracy of 90.049% on the sentiment analysis dataset. This is better than both the pure sentiment analysis model and the transfer learning approach. The multitask model's ability to learn shared linguistic features between the tasks contributed to its improved performance in sentiment analysis.

All in all, the project shows that multitask learning is a better approach for improving sentiment analysis compared to transfer learning with different integration techniques. The multitask learning model's performance suggests that sarcasm detection provides meaningful context that improves sentiment analysis, particularly when handling nuanced language. The results highlight the potential of multitask learning as a robust framework for enhancing NLP tasks where multiple related tasks share useful linguistic features.

Future Improvements

Future work could work towards several directions to further improve the integration of sarcasm detection into sentiment analysis. One possible direction is to experiment with different architectures, such as more complex transformer models like RoBERTa or GPT, which might capture deeper contextual relationships between sarcasm and sentiment. Also, future work could try incorporating more diverse datasets, including different languages and domains. This could improve the model's generalization capabilities and robustness across various contexts. Future work could also explore other methods of feature extraction or fine-tuning strategies, such as adaptive learning rates or advanced regularization techniques, which might also lead to improved performance. Furthermore, trying out the use of ensemble methods, where multiple models contribute to the final prediction, could potentially boost accuracy and provide more accurate sentiment predictions. Lastly, by gathering user feedback, the feedback could provide valuable insights for continuous improvement and refinement of the model's performance.

References

[1] Plaue, M. (5 May, 2023). Large-scale language models for innovation and technology intelligence: sentiment analysis on news articles. Retrieved from Medium: https://medium.com/mapegy-tech/large-scale-language-models-for-innovation-and-technology-intelligence-sentiment-analysis-on-news-2c1ed1f6f2ad

[2] Raees, M., & Fazilat, S. (2024). Lexicon-Based Sentiment Analysis on Text Polarities with Evaluation of Classification Models. arXiv preprint arXiv:2409.12840.

[3] Kenton, J. D. M. W. C., & Toutanova, L. K. (2019, June). Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of naacL-HLT (Vol. 1, p. 2).

[4] Tan, Y. Y., Chow, C. O., Kanesan, J., Chuah, J. H., & Lim, Y. (2023). Sentiment analysis and sarcasm detection using deep multi-task learning. Wireless personal communications, 129(3), 2213-2237.

[5] Parde, N., & Nielsen, R. (2018, June). Detecting sarcasm is extremely easy;-. In Proceedings of the workshop on computational semantics beyond events and roles (pp. 21-26).

[6] Ali, R., Farhat, T., Abdullah, S., Akram, S., Alhajlah, M., Mahmood, A., & Iqbal, M. A. (2023). Deep learning for sarcasm identification in news headlines. Applied Sciences, 13(9), 5586.

[7] Abercrombie, G., & Hovy, D. (2016, August). Putting sarcasm detection into context: The effects of class imbalance and manual labelling on supervised machine classification of twitter conversations. In Proceedings of the ACL 2016 student research workshop (pp. 107-113).

[8] Yaghoobian, H., Arabnia, H. R., & Rasheed, K. (2021). Sarcasm detection: A comparative study. arXiv preprint arXiv:2107.02276.

[9] Fu, H., Liu, H., Wang, H., Xu, L., Lin, J., & Jiang, D. (2024). Multi-modal sarcasm detection with sentiment word embedding. Electronics, 13(5), 855.

[10] Cambria, E., Zhang, X., Mao, R., Chen, M., & Kwok, K. (2024). SenticNet 8: Fusing emotion AI and commonsense AI for interpretable, trustworthy, and explainable affective computing. In International Conference on Human-Computer Interaction (HCII).

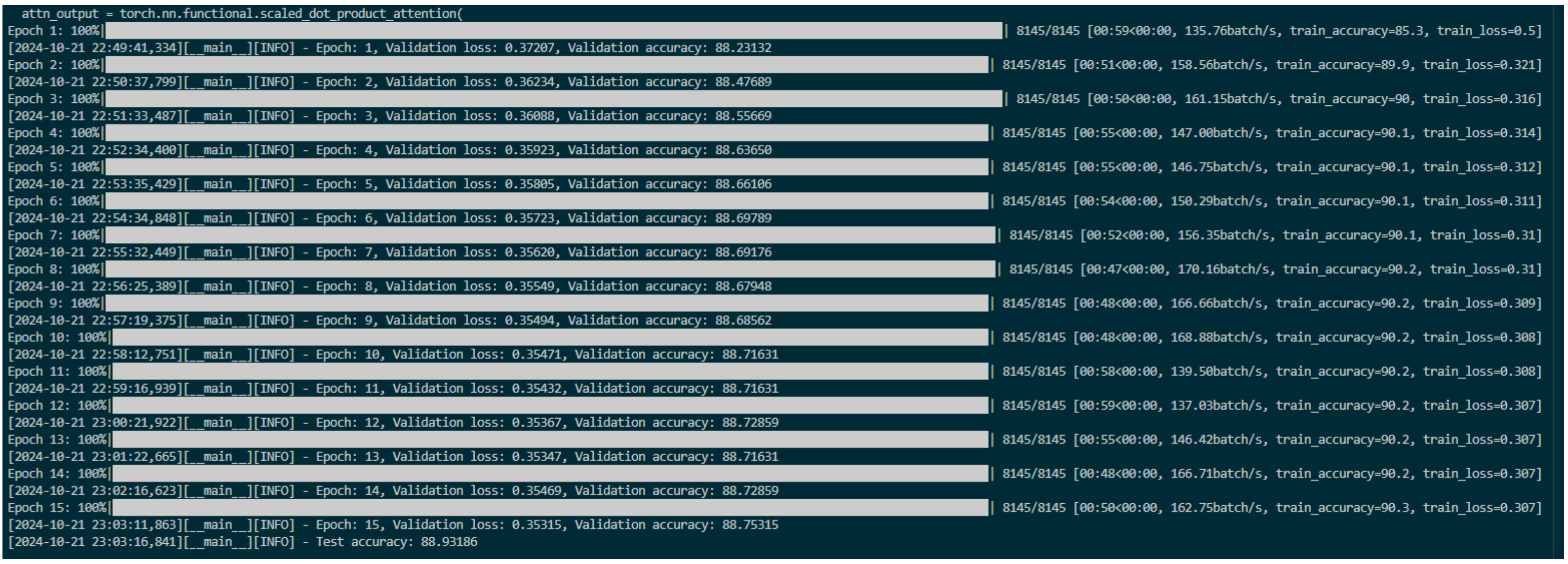
[11] News Headlines Dataset For Sarcasm Detection. (n.d.). Retrieved from Kaggle: https://www.kaggle.com/datasets/rmisra/news-headlines-dataset-for-sarcasm-detection

[12] Twitter and Reddit Sentimental analysis Dataset. (n.d.). Retrieved from Kaggle: https://www.kaggle.com/datasets/cosmos98/twitter-and-reddit-sentimental-analysis-dataset

[13] V7. (n.d.). Retrieved from V7labs: https://www.v7labs.com/blog/multi-task-learning-guide

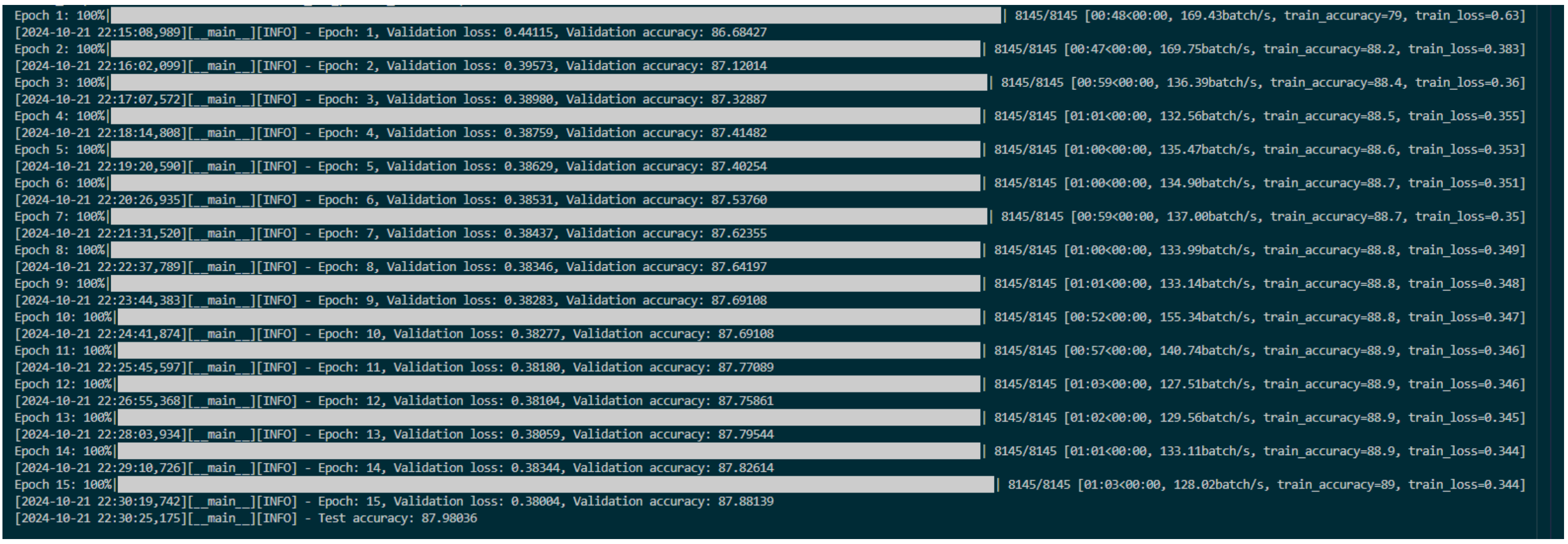
Appendix

Training for Model Approach 1



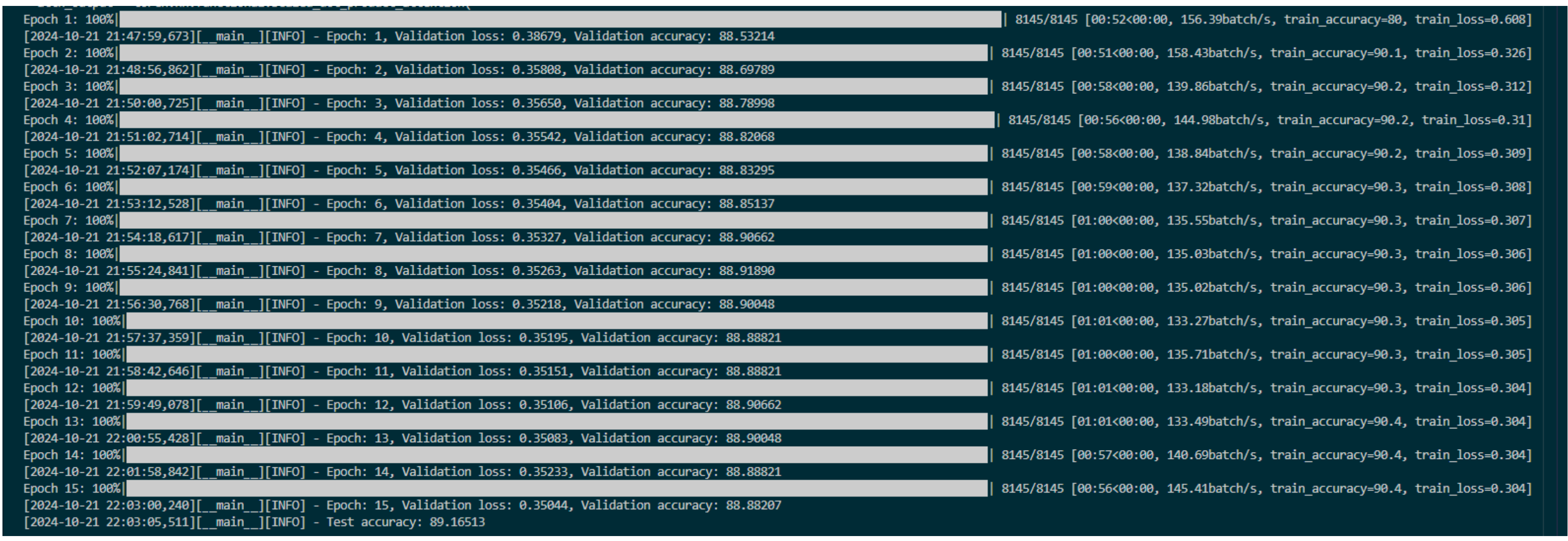
```
  attn_output = torch.nn.functional.scaled_dot_product_attention(
Epoch 1: 100%|| 8145/8145 [00:59<00:00, 135.76batch/s, train_accuracy=85.3, train_loss=0.5]
[2024-10-21 22:49:41,334][__main__][INFO] - Epoch: 1, Validation loss: 0.37207, Validation accuracy: 88.23132
Epoch 2: 100%|| 8145/8145 [00:51<00:00, 158.56batch/s, train_accuracy=89.9, train_loss=0.321]
[2024-10-21 22:50:37,799][__main__][INFO] - Epoch: 2, Validation loss: 0.36234, Validation accuracy: 88.47689
Epoch 3: 100%|| 8145/8145 [00:50<00:00, 161.15batch/s, train_accuracy=90, train_loss=0.316]
[2024-10-21 22:51:33,487][__main__][INFO] - Epoch: 3, Validation loss: 0.36088, Validation accuracy: 88.55669
Epoch 4: 100%|| 8145/8145 [00:55<00:00, 147.00batch/s, train_accuracy=90.1, train_loss=0.314]
[2024-10-21 22:52:34,400][__main__][INFO] - Epoch: 4, Validation loss: 0.35923, Validation accuracy: 88.63650
Epoch 5: 100%|| 8145/8145 [00:55<00:00, 146.75batch/s, train_accuracy=90.1, train_loss=0.312]
[2024-10-21 22:53:35,429][__main__][INFO] - Epoch: 5, Validation loss: 0.35805, Validation accuracy: 88.66106
Epoch 6: 100%|| 8145/8145 [00:54<00:00, 150.29batch/s, train_accuracy=90.1, train_loss=0.311]
[2024-10-21 22:54:34,848][__main__][INFO] - Epoch: 6, Validation loss: 0.35723, Validation accuracy: 88.69789
Epoch 7: 100%|| 8145/8145 [00:52<00:00, 156.35batch/s, train_accuracy=90.1, train_loss=0.31]
[2024-10-21 22:55:32,449][__main__][INFO] - Epoch: 7, Validation loss: 0.35620, Validation accuracy: 88.69176
Epoch 8: 100%|| 8145/8145 [00:47<00:00, 170.16batch/s, train_accuracy=90.2, train_loss=0.31]
[2024-10-21 22:56:25,389][__main__][INFO] - Epoch: 8, Validation loss: 0.35549, Validation accuracy: 88.67948
Epoch 9: 100%|| 8145/8145 [00:48<00:00, 166.66batch/s, train_accuracy=90.2, train_loss=0.309]
[2024-10-21 22:57:19,375][__main__][INFO] - Epoch: 9, Validation loss: 0.35494, Validation accuracy: 88.68562
Epoch 10: 100%|| 8145/8145 [00:48<00:00, 168.88batch/s, train_accuracy=90.2, train_loss=0.308]
[2024-10-21 22:58:12,751][__main__][INFO] - Epoch: 10, Validation loss: 0.35471, Validation accuracy: 88.71631
Epoch 11: 100%|| 8145/8145 [00:58<00:00, 139.50batch/s, train_accuracy=90.2, train_loss=0.308]
[2024-10-21 22:59:16,939][__main__][INFO] - Epoch: 11, Validation loss: 0.35432, Validation accuracy: 88.71631
Epoch 12: 100%|| 8145/8145 [00:59<00:00, 137.03batch/s, train_accuracy=90.2, train_loss=0.307]
[2024-10-21 23:00:21,922][__main__][INFO] - Epoch: 12, Validation loss: 0.35367, Validation accuracy: 88.72859
Epoch 13: 100%|| 8145/8145 [00:55<00:00, 146.42batch/s, train_accuracy=90.2, train_loss=0.307]
[2024-10-21 23:01:22,665][__main__][INFO] - Epoch: 13, Validation loss: 0.35347, Validation accuracy: 88.71631
Epoch 14: 100%|| 8145/8145 [00:48<00:00, 166.71batch/s, train_accuracy=90.2, train_loss=0.307]
[2024-10-21 23:02:16,623][__main__][INFO] - Epoch: 14, Validation loss: 0.35469, Validation accuracy: 88.72859
Epoch 15: 100%|| 8145/8145 [00:50<00:00, 162.75batch/s, train_accuracy=90.3, train_loss=0.307]
[2024-10-21 23:03:11,863][__main__][INFO] - Epoch: 15, Validation loss: 0.35315, Validation accuracy: 88.75315
[2024-10-21 23:03:16,841][__main__][INFO] - Test accuracy: 88.93186
```

Figure 4.1: Averaging the last hidden states from both models and then extracting the [CLS] token's hidden state



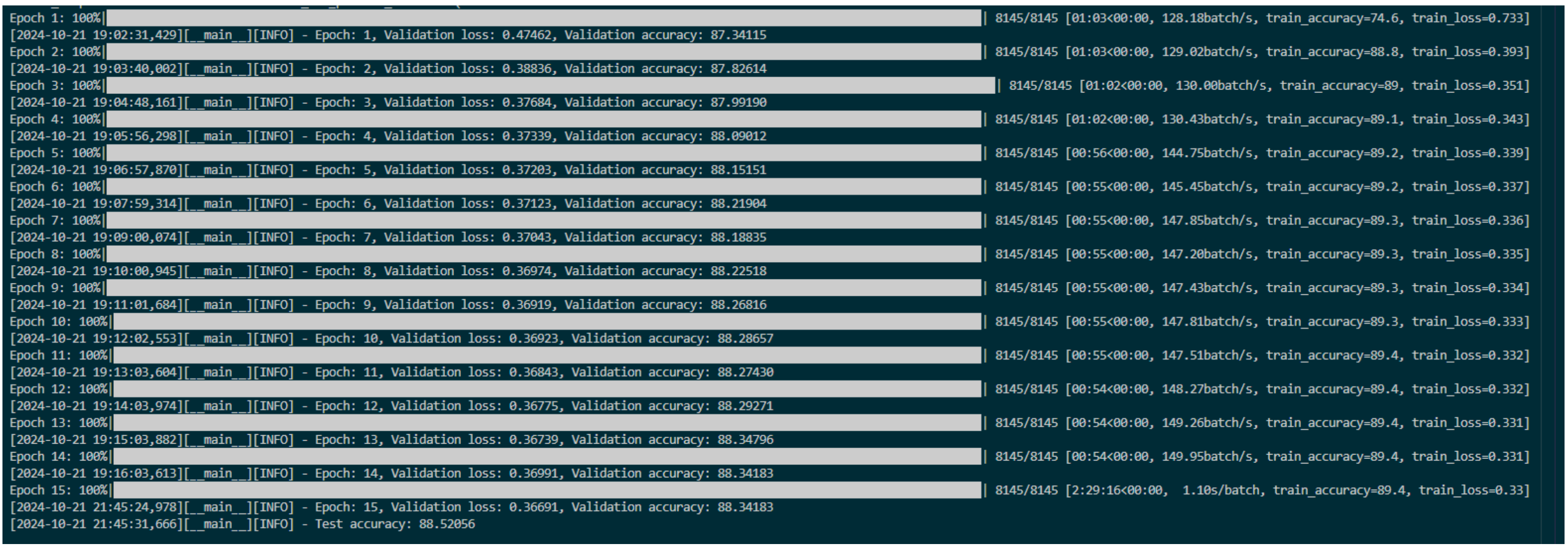
```
Epoch 1: 100%|| 8145/8145 [00:48<00:00, 169.43batch/s, train_accuracy=79, train_loss=0.63]
[2024-10-21 22:15:08,989][__main__][INFO] - Epoch: 1, Validation loss: 0.44115, Validation accuracy: 86.68427
Epoch 2: 100%|| 8145/8145 [00:47<00:00, 169.75batch/s, train_accuracy=88.2, train_loss=0.383]
[2024-10-21 22:16:02,099][__main__][INFO] - Epoch: 2, Validation loss: 0.39573, Validation accuracy: 87.12014
Epoch 3: 100%|| 8145/8145 [00:59<00:00, 136.39batch/s, train_accuracy=88.4, train_loss=0.36]
[2024-10-21 22:17:07,572][__main__][INFO] - Epoch: 3, Validation loss: 0.38980, Validation accuracy: 87.32887
Epoch 4: 100%|| 8145/8145 [01:01<00:00, 132.56batch/s, train_accuracy=88.5, train_loss=0.355]
[2024-10-21 22:18:14,808][__main__][INFO] - Epoch: 4, Validation loss: 0.38759, Validation accuracy: 87.41482
Epoch 5: 100%|| 8145/8145 [01:00<00:00, 135.47batch/s, train_accuracy=88.6, train_loss=0.353]
[2024-10-21 22:19:20,590][__main__][INFO] - Epoch: 5, Validation loss: 0.38629, Validation accuracy: 87.40254
Epoch 6: 100%|| 8145/8145 [01:00<00:00, 134.90batch/s, train_accuracy=88.7, train_loss=0.351]
[2024-10-21 22:20:26,935][__main__][INFO] - Epoch: 6, Validation loss: 0.38531, Validation accuracy: 87.53760
Epoch 7: 100%|| 8145/8145 [00:59<00:00, 137.00batch/s, train_accuracy=88.7, train_loss=0.35]
[2024-10-21 22:21:31,520][__main__][INFO] - Epoch: 7, Validation loss: 0.38437, Validation accuracy: 87.62355
Epoch 8: 100%|| 8145/8145 [01:00<00:00, 133.99batch/s, train_accuracy=88.8, train_loss=0.349]
[2024-10-21 22:22:37,789][__main__][INFO] - Epoch: 8, Validation loss: 0.38346, Validation accuracy: 87.64197
Epoch 9: 100%|| 8145/8145 [01:01<00:00, 133.14batch/s, train_accuracy=88.8, train_loss=0.348]
[2024-10-21 22:23:44,383][__main__][INFO] - Epoch: 9, Validation loss: 0.38283, Validation accuracy: 87.69108
Epoch 10: 100%|| 8145/8145 [00:52<00:00, 155.34batch/s, train_accuracy=88.8, train_loss=0.347]
[2024-10-21 22:24:41,874][__main__][INFO] - Epoch: 10, Validation loss: 0.38277, Validation accuracy: 87.69108
Epoch 11: 100%|| 8145/8145 [00:57<00:00, 140.74batch/s, train_accuracy=88.9, train_loss=0.346]
[2024-10-21 22:25:45,597][__main__][INFO] - Epoch: 11, Validation loss: 0.38180, Validation accuracy: 87.77089
Epoch 12: 100%|| 8145/8145 [01:03<00:00, 127.51batch/s, train_accuracy=88.9, train_loss=0.346]
[2024-10-21 22:26:55,368][__main__][INFO] - Epoch: 12, Validation loss: 0.38104, Validation accuracy: 87.75861
Epoch 13: 100%|| 8145/8145 [01:02<00:00, 129.56batch/s, train_accuracy=88.9, train_loss=0.345]
[2024-10-21 22:28:03,934][__main__][INFO] - Epoch: 13, Validation loss: 0.38059, Validation accuracy: 87.79544
Epoch 14: 100%|| 8145/8145 [01:01<00:00, 133.11batch/s, train_accuracy=88.9, train_loss=0.344]
[2024-10-21 22:29:10,726][__main__][INFO] - Epoch: 14, Validation loss: 0.38344, Validation accuracy: 87.82614
Epoch 15: 100%|| 8145/8145 [01:03<00:00, 128.02batch/s, train_accuracy=89, train_loss=0.344]
[2024-10-21 22:30:19,742][__main__][INFO] - Epoch: 15, Validation loss: 0.38004, Validation accuracy: 87.88139
[2024-10-21 22:30:25,175][__main__][INFO] - Test accuracy: 87.98036
```

Figure 4.2: Averaging the last hidden states from both models, performing max pooling



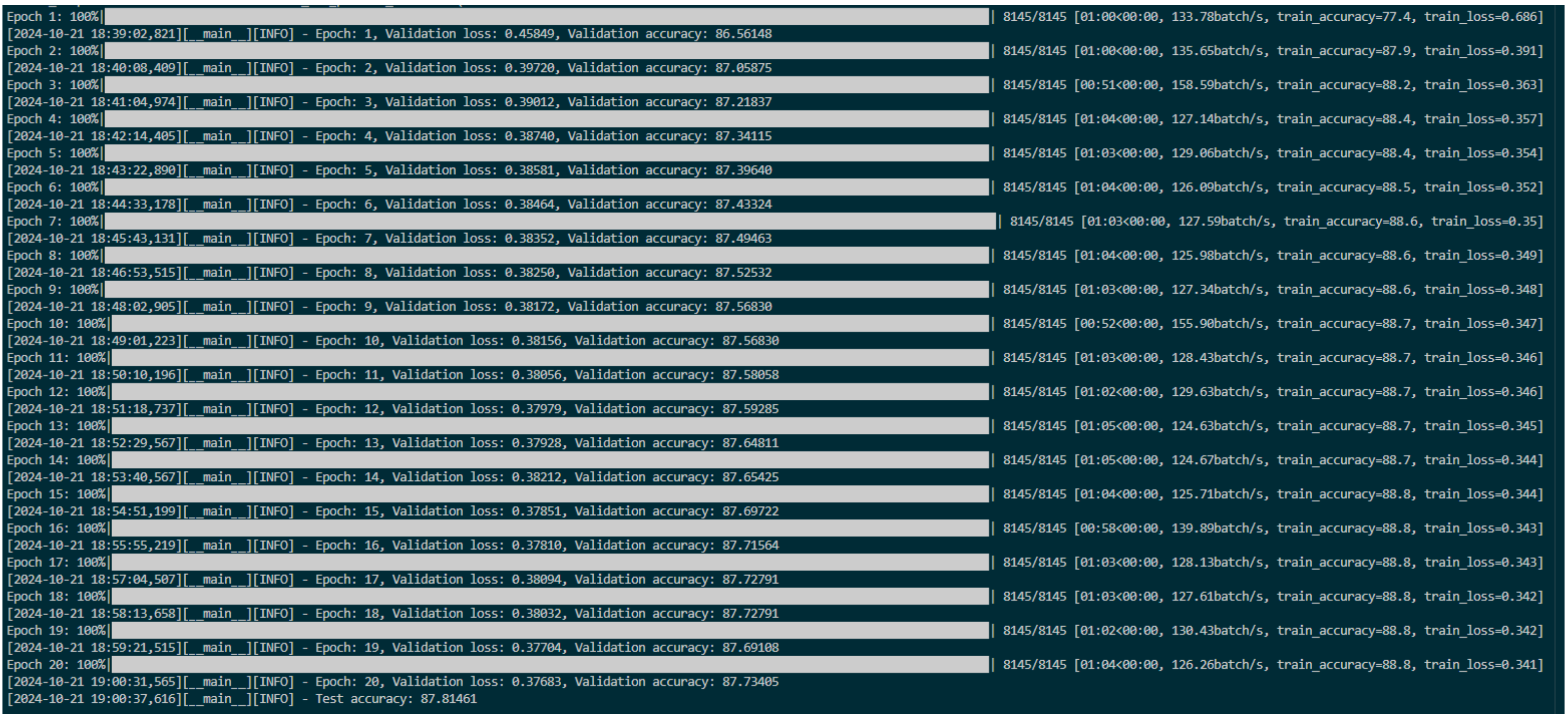
```
Epoch 1: 100%|██████████| 8145/8145 [00:52<00:00, 156.39batch/s, train_accuracy=80, train_loss=0.608]
[2024-10-21 21:47:59,673][__main__][INFO] - Epoch: 1, Validation loss: 0.38679, Validation accuracy: 88.53214
Epoch 2: 100%|██████████| 8145/8145 [00:51<00:00, 158.43batch/s, train_accuracy=90.1, train_loss=0.326]
[2024-10-21 21:48:56,862][__main__][INFO] - Epoch: 2, Validation loss: 0.35808, Validation accuracy: 88.69789
Epoch 3: 100%|██████████| 8145/8145 [00:58<00:00, 139.86batch/s, train_accuracy=90.2, train_loss=0.312]
[2024-10-21 21:50:00,725][__main__][INFO] - Epoch: 3, Validation loss: 0.35650, Validation accuracy: 88.78998
Epoch 4: 100%|██████████| 8145/8145 [00:56<00:00, 144.98batch/s, train_accuracy=90.2, train_loss=0.31]
[2024-10-21 21:51:02,714][__main__][INFO] - Epoch: 4, Validation loss: 0.35542, Validation accuracy: 88.82068
Epoch 5: 100%|██████████| 8145/8145 [00:58<00:00, 138.84batch/s, train_accuracy=90.2, train_loss=0.309]
[2024-10-21 21:52:07,174][__main__][INFO] - Epoch: 5, Validation loss: 0.35466, Validation accuracy: 88.83295
Epoch 6: 100%|██████████| 8145/8145 [00:59<00:00, 137.32batch/s, train_accuracy=90.3, train_loss=0.308]
[2024-10-21 21:53:12,528][__main__][INFO] - Epoch: 6, Validation loss: 0.35404, Validation accuracy: 88.85137
Epoch 7: 100%|██████████| 8145/8145 [01:00<00:00, 135.55batch/s, train_accuracy=90.3, train_loss=0.307]
[2024-10-21 21:54:18,617][__main__][INFO] - Epoch: 7, Validation loss: 0.35327, Validation accuracy: 88.90662
Epoch 8: 100%|██████████| 8145/8145 [01:00<00:00, 135.03batch/s, train_accuracy=90.3, train_loss=0.306]
[2024-10-21 21:55:24,841][__main__][INFO] - Epoch: 8, Validation loss: 0.35263, Validation accuracy: 88.91890
Epoch 9: 100%|██████████| 8145/8145 [01:00<00:00, 135.02batch/s, train_accuracy=90.3, train_loss=0.306]
[2024-10-21 21:56:30,768][__main__][INFO] - Epoch: 9, Validation loss: 0.35218, Validation accuracy: 88.90048
Epoch 10: 100%|██████████| 8145/8145 [01:01<00:00, 133.27batch/s, train_accuracy=90.3, train_loss=0.305]
[2024-10-21 21:57:37,359][__main__][INFO] - Epoch: 10, Validation loss: 0.35195, Validation accuracy: 88.88821
Epoch 11: 100%|██████████| 8145/8145 [01:00<00:00, 135.71batch/s, train_accuracy=90.3, train_loss=0.305]
[2024-10-21 21:58:42,646][__main__][INFO] - Epoch: 11, Validation loss: 0.35151, Validation accuracy: 88.88821
Epoch 12: 100%|██████████| 8145/8145 [01:01<00:00, 133.18batch/s, train_accuracy=90.3, train_loss=0.304]
[2024-10-21 21:59:49,078][__main__][INFO] - Epoch: 12, Validation loss: 0.35106, Validation accuracy: 88.90662
Epoch 13: 100%|██████████| 8145/8145 [01:01<00:00, 133.49batch/s, train_accuracy=90.4, train_loss=0.304]
[2024-10-21 22:00:55,428][__main__][INFO] - Epoch: 13, Validation loss: 0.35083, Validation accuracy: 88.90048
Epoch 14: 100%|██████████| 8145/8145 [00:57<00:00, 140.69batch/s, train_accuracy=90.4, train_loss=0.304]
[2024-10-21 22:01:58,842][__main__][INFO] - Epoch: 14, Validation loss: 0.35233, Validation accuracy: 88.88821
Epoch 15: 100%|██████████| 8145/8145 [00:56<00:00, 145.41batch/s, train_accuracy=90.4, train_loss=0.304]
[2024-10-21 22:03:00,240][__main__][INFO] - Epoch: 15, Validation loss: 0.35044, Validation accuracy: 88.88207
[2024-10-21 22:03:05,511][__main__][INFO] - Test accuracy: 89.16513
```

Figure 4.3: Concatenating the last hidden states from both models, extracting the [CLS] token's hidden state



```
Epoch 1: 100%|██████████| 8145/8145 [01:03<00:00, 128.18batch/s, train_accuracy=74.6, train_loss=0.733]
[2024-10-21 19:02:31,429][__main__][INFO] - Epoch: 1, Validation loss: 0.47462, Validation accuracy: 87.34115
Epoch 2: 100%|██████████| 8145/8145 [01:03<00:00, 129.02batch/s, train_accuracy=88.8, train_loss=0.393]
[2024-10-21 19:03:40,002][__main__][INFO] - Epoch: 2, Validation loss: 0.38836, Validation accuracy: 87.82614
Epoch 3: 100%|██████████| 8145/8145 [01:02<00:00, 130.00batch/s, train_accuracy=89, train_loss=0.351]
[2024-10-21 19:04:48,161][__main__][INFO] - Epoch: 3, Validation loss: 0.37684, Validation accuracy: 87.99190
Epoch 4: 100%|██████████| 8145/8145 [01:02<00:00, 130.43batch/s, train_accuracy=89.1, train_loss=0.343]
[2024-10-21 19:05:56,298][__main__][INFO] - Epoch: 4, Validation loss: 0.37339, Validation accuracy: 88.09012
Epoch 5: 100%|██████████| 8145/8145 [00:56<00:00, 144.75batch/s, train_accuracy=89.2, train_loss=0.339]
[2024-10-21 19:06:57,870][__main__][INFO] - Epoch: 5, Validation loss: 0.37203, Validation accuracy: 88.15151
Epoch 6: 100%|██████████| 8145/8145 [00:55<00:00, 145.45batch/s, train_accuracy=89.2, train_loss=0.337]
[2024-10-21 19:07:59,314][__main__][INFO] - Epoch: 6, Validation loss: 0.37123, Validation accuracy: 88.21904
Epoch 7: 100%|██████████| 8145/8145 [00:55<00:00, 147.85batch/s, train_accuracy=89.3, train_loss=0.336]
[2024-10-21 19:09:00,074][__main__][INFO] - Epoch: 7, Validation loss: 0.37043, Validation accuracy: 88.18835
Epoch 8: 100%|██████████| 8145/8145 [00:55<00:00, 147.20batch/s, train_accuracy=89.3, train_loss=0.335]
[2024-10-21 19:10:00,945][__main__][INFO] - Epoch: 8, Validation loss: 0.36974, Validation accuracy: 88.22518
Epoch 9: 100%|██████████| 8145/8145 [00:55<00:00, 147.43batch/s, train_accuracy=89.3, train_loss=0.334]
[2024-10-21 19:11:01,684][__main__][INFO] - Epoch: 9, Validation loss: 0.36919, Validation accuracy: 88.26816
Epoch 10: 100%|██████████| 8145/8145 [00:55<00:00, 147.81batch/s, train_accuracy=89.3, train_loss=0.333]
[2024-10-21 19:12:02,553][__main__][INFO] - Epoch: 10, Validation loss: 0.36923, Validation accuracy: 88.28657
Epoch 11: 100%|██████████| 8145/8145 [00:55<00:00, 147.51batch/s, train_accuracy=89.4, train_loss=0.332]
[2024-10-21 19:13:03,604][__main__][INFO] - Epoch: 11, Validation loss: 0.36843, Validation accuracy: 88.27430
Epoch 12: 100%|██████████| 8145/8145 [00:54<00:00, 148.27batch/s, train_accuracy=89.4, train_loss=0.332]
[2024-10-21 19:14:03,974][__main__][INFO] - Epoch: 12, Validation loss: 0.36775, Validation accuracy: 88.29271
Epoch 13: 100%|██████████| 8145/8145 [00:54<00:00, 149.26batch/s, train_accuracy=89.4, train_loss=0.331]
[2024-10-21 19:15:03,882][__main__][INFO] - Epoch: 13, Validation loss: 0.36739, Validation accuracy: 88.34796
Epoch 14: 100%|██████████| 8145/8145 [00:54<00:00, 149.95batch/s, train_accuracy=89.4, train_loss=0.331]
[2024-10-21 19:16:03,613][__main__][INFO] - Epoch: 14, Validation loss: 0.36991, Validation accuracy: 88.34183
Epoch 15: 100%|██████████| 8145/8145 [2:29:16<00:00,  1.10s/batch, train_accuracy=89.4, train_loss=0.33]
[2024-10-21 21:45:24,978][__main__][INFO] - Epoch: 15, Validation loss: 0.36691, Validation accuracy: 88.34183
[2024-10-21 21:45:31,666][__main__][INFO] - Test accuracy: 88.52056
```

Figure 4.4: Concatenating the last hidden states from both models, performing max pooling



```
Epoch 1: 100%|██████████| 8145/8145 [01:00<00:00, 133.78batch/s, train_accuracy=77.4, train_loss=0.686]
[2024-10-21 18:39:02,821][__main__][INFO] - Epoch: 1, Validation loss: 0.45849, Validation accuracy: 86.56148
Epoch 2: 100%|██████████| 8145/8145 [01:00<00:00, 135.65batch/s, train_accuracy=87.9, train_loss=0.391]
[2024-10-21 18:40:08,409][__main__][INFO] - Epoch: 2, Validation loss: 0.39720, Validation accuracy: 87.05875
Epoch 3: 100%|██████████| 8145/8145 [00:51<00:00, 158.59batch/s, train_accuracy=88.2, train_loss=0.363]
[2024-10-21 18:41:04,974][__main__][INFO] - Epoch: 3, Validation loss: 0.39012, Validation accuracy: 87.21837
Epoch 4: 100%|██████████| 8145/8145 [01:04<00:00, 127.14batch/s, train_accuracy=88.4, train_loss=0.357]
[2024-10-21 18:42:14,405][__main__][INFO] - Epoch: 4, Validation loss: 0.38740, Validation accuracy: 87.34115
Epoch 5: 100%|██████████| 8145/8145 [01:03<00:00, 129.06batch/s, train_accuracy=88.4, train_loss=0.354]
[2024-10-21 18:43:22,890][__main__][INFO] - Epoch: 5, Validation loss: 0.38581, Validation accuracy: 87.39640
Epoch 6: 100%|██████████| 8145/8145 [01:04<00:00, 126.09batch/s, train_accuracy=88.5, train_loss=0.352]
[2024-10-21 18:44:33,178][__main__][INFO] - Epoch: 6, Validation loss: 0.38464, Validation accuracy: 87.43324
Epoch 7: 100%|██████████| 8145/8145 [01:03<00:00, 127.59batch/s, train_accuracy=88.6, train_loss=0.35]
[2024-10-21 18:45:43,131][__main__][INFO] - Epoch: 7, Validation loss: 0.38352, Validation accuracy: 87.49463
Epoch 8: 100%|██████████| 8145/8145 [01:04<00:00, 125.98batch/s, train_accuracy=88.6, train_loss=0.349]
[2024-10-21 18:46:53,515][__main__][INFO] - Epoch: 8, Validation loss: 0.38250, Validation accuracy: 87.52532
Epoch 9: 100%|██████████| 8145/8145 [01:03<00:00, 127.34batch/s, train_accuracy=88.6, train_loss=0.348]
[2024-10-21 18:48:02,905][__main__][INFO] - Epoch: 9, Validation loss: 0.38172, Validation accuracy: 87.56830
Epoch 10: 100%|██████████| 8145/8145 [00:52<00:00, 155.90batch/s, train_accuracy=88.7, train_loss=0.347]
[2024-10-21 18:49:01,223][__main__][INFO] - Epoch: 10, Validation loss: 0.38156, Validation accuracy: 87.56830
Epoch 11: 100%|██████████| 8145/8145 [01:03<00:00, 128.43batch/s, train_accuracy=88.7, train_loss=0.346]
[2024-10-21 18:50:10,196][__main__][INFO] - Epoch: 11, Validation loss: 0.38056, Validation accuracy: 87.58058
Epoch 12: 100%|██████████| 8145/8145 [01:02<00:00, 129.63batch/s, train_accuracy=88.7, train_loss=0.346]
[2024-10-21 18:51:18,737][__main__][INFO] - Epoch: 12, Validation loss: 0.37979, Validation accuracy: 87.59285
Epoch 13: 100%|██████████| 8145/8145 [01:05<00:00, 124.63batch/s, train_accuracy=88.7, train_loss=0.345]
[2024-10-21 18:52:29,567][__main__][INFO] - Epoch: 13, Validation loss: 0.37928, Validation accuracy: 87.64811
Epoch 14: 100%|██████████| 8145/8145 [01:05<00:00, 124.67batch/s, train_accuracy=88.7, train_loss=0.344]
[2024-10-21 18:53:40,567][__main__][INFO] - Epoch: 14, Validation loss: 0.38212, Validation accuracy: 87.65425
Epoch 15: 100%|██████████| 8145/8145 [01:04<00:00, 125.71batch/s, train_accuracy=88.8, train_loss=0.344]
[2024-10-21 18:54:51,199][__main__][INFO] - Epoch: 15, Validation loss: 0.37851, Validation accuracy: 87.69722
Epoch 16: 100%|██████████| 8145/8145 [00:58<00:00, 139.89batch/s, train_accuracy=88.8, train_loss=0.343]
[2024-10-21 18:55:55,219][__main__][INFO] - Epoch: 16, Validation loss: 0.37810, Validation accuracy: 87.71564
Epoch 17: 100%|██████████| 8145/8145 [01:03<00:00, 128.13batch/s, train_accuracy=88.8, train_loss=0.343]
[2024-10-21 18:57:04,507][__main__][INFO] - Epoch: 17, Validation loss: 0.38094, Validation accuracy: 87.72791
Epoch 18: 100%|██████████| 8145/8145 [01:03<00:00, 127.61batch/s, train_accuracy=88.8, train_loss=0.342]
[2024-10-21 18:58:13,658][__main__][INFO] - Epoch: 18, Validation loss: 0.38032, Validation accuracy: 87.72791
Epoch 19: 100%|██████████| 8145/8145 [01:02<00:00, 130.43batch/s, train_accuracy=88.8, train_loss=0.342]
[2024-10-21 18:59:21,515][__main__][INFO] - Epoch: 19, Validation loss: 0.37704, Validation accuracy: 87.69108
Epoch 20: 100%|██████████| 8145/8145 [01:04<00:00, 126.26batch/s, train_accuracy=88.8, train_loss=0.341]
[2024-10-21 19:00:31,565][__main__][INFO] - Epoch: 20, Validation loss: 0.37683, Validation accuracy: 87.73405
[2024-10-21 19:00:37,616][__main__][INFO] - Test accuracy: 87.81461
```

Figure 4.5: Concatenating the last hidden states from both models, performing mean pooling