Task 1: Sentence Transformer Implementation

Model Architecture Choices

1. **Linear Layer for Fixed-Length Embeddings**: Added a linear layer after the BERT model to transform the variable-length BERT output into fixed-length embeddings. The output size of this linear layer (output_size) can be adjusted as needed.
2. **Normalization**: Applied L2 normalization (torch.nn.functional.normalize) to the pooled BERT output to ensure consistent scale and numerical stability of the embeddings.
3. **Tokenizer**: Used the BERT tokenizer to tokenize and convert input sentences into input tensors for the BERT model.

This implementation creates fixed-length embeddings for input sentences using the BERT model's pooler output and a linear layer for transformation, along with normalization for stability.

Embeddings for a few sample sentences are printed to showcase the model's encoding capabilities.

Task 2: Multi-Task Learning Expansion

DATASET :- This dataset was created for the Paper 'From Group to Individual Labels using Deep Features', Kotzias et. al,. KDD 2015
Kotzias,Dimitrios. (2015). Sentiment Labeled Sentences. UCI Machine Learning Repository. https://doi.org/10.24432/C57604.

It contains sentences labeled with positive or negative sentiment, extracted from reviews of products, movies, and restaurants
Score is either 1 (for positive) or 0 (for negative)

The sentences come from three different websites/fields:

imdb.com
amazon.com
yelp.com

For each website, there exist 500 positive and 500 negative sentences. Those were selected randomly for larger datasets of reviews.
Sentences have a clearly positive or negative connotation, the goal was for no neutral sentences to be selected.

The major changes were adding a SHARED LINEAR LAYER AND LOSS FUNCTIONS AS DESCRIBED ABOVE. I will be elaborating on before and after implementation since my code has the modified MultiTask Model implemented.

Task A and Task B in this sentence transformer model are akin to specialized roles it can fulfill:

1. Task A: Sentiment Verdict (Binary Classification)

- Task A involves rendering a verdict on whether sentences carry positive or negative sentiments, similar to judging if a movie review is thumbs-up or thumbs-down.
- Mechanism: The model employs a dedicated layer (`classification_head`) to convert textual inputs into categorical decisions: positive or negative, akin to a classifier.
- Training Method: It learns through examples using cross-entropy loss (`classification_loss_fn`), optimizing its ability to discern sentiment nuances effectively.

Example: Given a sentence like "The plot twists were unexpected," Task A would deem it positive.

2. Task B: Sentiment Intensity (Sentiment Analysis as Numerical Values)

- - Objective: Task B quantifies the intensity of sentiment in sentences, akin to rating a product from -1 (displeasing) to 1 (excellent).
- Mechanism: Another specialized layer (`sentiment_head`) transforms textual inputs into a numerical sentiment score, indicating sentiment strength.
- Training Method: It optimizes through binary cross-entropy loss (`sentiment_loss_fn`), fine-tuning its ability to assign precise sentiment scores.

Example: For a sentence like "The service was satisfactory," Task B might assign a score of 0.2, indicating a moderate positive sentiment.

Thus, Task A swiftly categorizes sentiment, while Task B provides a nuanced sentiment intensity, both crucial for tasks like sentiment analysis, review sorting, and opinion understanding.

Original Architecture (Before Multi-Task Learning)

1. **Single Task-Specific Head**: The original architecture would have a single head, say for classification, depending on the task at hand.
2. **Single Loss Function**: The loss function would be specific to the single task (CrossEntropyLoss for classification).
3. **Output**: The model would produce output specific to the single task (logits for classification).

Modified Architecture (After Multi-Task Learning)

1. **Shared Feature Extraction Layer**: A shared linear layer (self.shared_linear) was added to reduce the dimensionality of the BERT embeddings from 768 to 256 dimensions to extract common features useful for both tasks, allowing the model to learn a shared representation.
2. **Task-Specific Heads**: Two separate heads were added for different tasks:

Classification Head: A linear layer with an output size of 2 (for binary classification: Positive, Negative).

Sentiment Analysis Head: A linear layer with an output size of 1 (for regression of sentiment scores).

3. **Embedding Normalization**: Applying F.normalize to the output of the shared linear layer ensures the embeddings have a consistent scale, stabilizing training and aiding generalization. It acts as a form of regularization, preventing the embeddings from growing too large and helping the model to generalize better.
4. **Separate Loss Functions**: Defining separate loss functions for each task:

   Classification Loss using CrossEntropyLoss and Sentiment Analysis Loss using BCELoss.By defining separate loss functions, the model can optimize both tasks simultaneously. This setup allows balancing the contribution of each task during training by combining the losses.

5. During training, the total loss is computed as the sum of the classification loss and the sentiment analysis loss which ensures both tasks contribute to the optimization process, promoting balanced learning.
6. **Optimizer Selection :** Using **AdamW (Adam with Weight Decay)** optimizer.

   Learning Rate Adaptation: AdamW adapts the learning rate for each parameter, which helps in achieving better convergence, especially for models with many parameters like transformers.

   Weight Decay: Adding weight decay (L2 regularization) helps in preventing overfitting, which is crucial for models trained on relatively small datasets.

7. **Forward Method Modification**: The forward method was adjusted to handle the shared linear layer and produce outputs for both tasks:
   - **Logits** for classification.
   - **Sentiment Scores** for sentiment analysis.

The model is trained for 3 epochs using a combined loss function for both tasks.

After training, the model is evaluated on the test set, and performance metrics (accuracy, precision, recall, F1 score) are printed.

The model achieves a **Classification Accuracy of 94% when trained on 3000 instances with a 80-20 split.**

Task 3: Training Considerations

Implications and Advantages of Different Freezing Scenarios

> Freezing the Entire Network

- Restricted Learning: The model doesn't learn task-specific nuances as all weights stay fixed.
- Training speed is high due to no weight updates.
- Effective only if pre-trained model outputs are directly applicable without modifications.

Advantages:

- Fast Processing: Computation time is significantly reduced.
- Resource Conservation: Saves memory and computational power.
- Simplicity: Implementation is straightforward without backpropagation.

This strategy works best when the pre-trained model's representations align perfectly with new tasks, requiring no adjustments. However, i**t's rarely optimal** as most applications benefit from task-specific fine-tuning.

> Freezing Only the Transformer Backbone

- Backbone provides stable features while task-specific heads adapt.
- Only task-specific heads adjust to new tasks, leveraging pre-trained features.
- Faster than full network training yet allows some task-specific learning.

Advantages:

- Efficient Feature Use: Retains pre-trained knowledge while adapting to tasks.
- Reduced Training Time: Faster than full network training.
- Stability: Reduces forgetting risk as backbone remains stable.

This approach suits cases where **pre-trained model features are strong but tasks require some adaptation**, offering a **balance** between leveraging prior knowledge and task adaptation.

> Freezing Only One Task-Specific Head

- One head remains fixed while the other adapts, potentially causing performance imbalances.
- Prioritizes learning for one task over the other.
- Allows partial model adaptation, useful for tasks similar to the original pre-training.

Advantages:

- Targeted Adaptation: Efficient for tasks already handled well by the pre-trained model.
- Resource Management: More efficient than full network updates but less than freezing the backbone.

This strategy benefits when one **task aligns closely with the pre-trained model**, needing minimal adaptation, while the other task requires more specific training. It maintains performance for well-handled tasks while focusing resources on new ones.

CONSIDERING A SCENARIO

Let's consider a scenario where transfer learning can be beneficial, such as sentiment analysis on customer reviews in the e-commerce domain. We'll approach the transfer learning process step by step.

For this scenario, we can choose a pre-trained model like BERT (Bidirectional Encoder Representations from Transformers) due to its effectiveness in natural language understanding tasks, including sentiment analysis. BERT has been pre-trained on a large corpus of text and has shown remarkable performance in capturing contextual information.

Freezing Transformer Backbone:
We'll freeze the layers of the pre-trained BERT model up to a certain point to leverage its general language understanding capabilities. This frozen part will serve as a feature extractor.

- **Feature Extraction**: BERT's lower layers are adept at capturing syntax, semantics, and general language understanding, which are valuable features for sentiment analysis.
- **Prevent Overfitting**: Freezing these layers helps prevent overfitting on the target dataset, especially when the dataset is relatively small compared to the pre-training data.

Unfreezing Task-Specific Head:
We'll unfreeze the final layers of the model, including the classification head, to allow them to learn task-specific features related to sentiment analysis.

- **Task-Specific Adaptation**: The task-specific head (classification layer) needs to adapt to the sentiment analysis task by learning sentiment-related patterns from the target dataset.
- **Fine-Tuning**: Unfreezing allows fine-tuning of the model's final layers to better align with the sentiment analysis task, improving model performance.

This approach effectively combines the benefits of transfer learning from BERT and task-specific adaptation for sentiment analysis.

Task 4: Layer-wise Learning Rate Implementation

Below are the metrics and output when the model is trained on 1000 instances (due to time constraints)

METRICS FOR WITHOUT LAYER WISE LEARNING RATES
Classification Accuracy: 91.00%
Precision: 0.9192
Recall: 0.9010
F1 Score: 0.9100
Epoch 1/3, Loss: 1.2992243707180022

Epoch 2/3, Loss: 1.058097721338272
Epoch 3/3, Loss: 0.9869545352458954

optimizer = optim.AdamW(model.parameters(), lr=1e-5)

In this scenario, a uniform learning rate of 1e-5 is used for all model parameters. The accuracy and other metrics indicate decent performance, with a moderate loss value indicating some room for improvement in model convergence.

METRICS FOR WITH LAYER WISE LEARNING RATES
Classification Accuracy: 91.50%
Precision: 0.8962
Recall: 0.9406
F1 Score: 0.9179
Epoch 1/3, Loss: 0.9483633637428284
Epoch 2/3, Loss: 0.9125769972801209
Epoch 3/3, Loss: 0.8977178597450256

optimizer = optim.AdamW([{'params': model.bert.parameters(), 'lr': 1e-5}, # Lower learning rate for BERT layers
{'params': model.shared_linear.parameters(), 'lr': 2e-5}, # Higher learning rate for the shared linear layer
{'params': model.classification_head.parameters(), 'lr': 2e-5}, # Higher learning rate for classification head
{'params': model.sentiment_head.parameters(), 'lr': 2e-5} # Higher learning rate for sentiment head])

In this scenario, different learning rates are applied to specific parts of the model:

- BERT Layers: 1e-5
- Shared Linear Layer: 2e-5
- Classification Head: 2e-5
- Sentiment Head: 2e-5

The accuracy slightly improves compared to the uniform learning rate scenario, reaching 91.50%. Precision and recall also show improvements, indicating better overall performance in classification tasks. The loss value decreases significantly, suggesting faster convergence and better model optimization.

1. BERT Layers (model.bert.parameters()) - Learning Rate: 1e-5

BERT is a pre-trained model with well-established representations. It's beneficial to update these layers slowly to avoid catastrophic forgetting and preserve the general knowledge captured during pre-training.

2. Shared Linear Layer (model.shared_linear.parameters()) - Learning Rate: 2e-5

This layer connects the BERT representation to task-specific heads. It's crucial for fine-tuning the model's representations to specific tasks, hence a higher learning rate is chosen.

3. Classification Head (model.classification_head.parameters()) - Learning Rate: 2e-5

This head is responsible for task A, which involves classifying sentences into categories (e.g., positive or negative sentiment). It needs a higher learning rate to adapt quickly to the specific classification task.

4. Sentiment Head (model.sentiment_head.parameters()) - Learning Rate: 2e-5

Task B involves sentiment analysis, which is also a classification task but with a different focus. Similar to the classification head, a higher learning rate is chosen to facilitate faster adaptation to sentiment-related features.

Analysis and Benefits:

1. **Accuracy and Task Performance:**

   The layer-wise learning rates scenario shows a slight improvement in accuracy, precision, recall, and F1 score compared to the uniform learning rate scenario. This indicates that fine-tuning specific parts of the model with higher learning rates has a positive impact on task performance.

2. **Loss and Convergence:**

   The lower loss value in the layer-wise learning rates scenario indicates faster convergence and better optimization of the model. This can lead to more stable training dynamics and potentially better generalization to unseen data.

3. **Task-Specific Adaptation:**

   By applying higher learning rates to task-specific components (e.g., classification and sentiment heads), the model can adapt more effectively to the nuances of each task. This results in improved performance metrics and overall task-specific adaptation.

4. **Efficient Training:**

   Layer-wise learning rates allow for more efficient training by focusing updates on parts of the model that require more adaptation. This can lead to faster training times and better utilization of computational resources..