Fetch Assessment

Task 1: Sentence Transformer Implementation

Model Architecture

- pre trained transformer model → encodes the input text into contextualized token embeddings.
  - tested 'bert-base-uncased' and 'distilbert-base-uncased'
- tokenizer → converts text into subword tokens using WordPiece Tokenization.Tokens are then converted into integer token IDs for the model.
- Pooling layer →Incorporated a pooling layer after the transformer component. Since transformer models output token-level embeddings, a pooling mechanism is needed to convert these into a single vector representing the entire sentence.
  - tested max(takes max value for each dimension across all token embeddings in a sentence) pooling strategy
  - tested mean (takes average of all token embeddings) pooling strategy
- Linear layer → used a linear layer to map transformer's output size to a dimension of 512 always regardless of which transformer is used.
- Layer Normalization → incorporated a layer norm to ensure the embeddings is balanced in every layer
  - it ensures consistency across all dimensions so that we have a uniformly distributed feature space
- L2 Normalization → ensures that all embeddings have unit length.
  - ensures consistency across the whole embedding so that we can accurately compute cosine similarity without letting length of one sentence unfairly affect the similarity score
- output → the model generates a dense vector representation of shape (batch size, embedding size).
- encode function → performs the forward pass and computes embeddings
- test_base_model function → creates embeddings of sample sentences and creates a cosine similarity matrix to show how closely the sentences relate to each other.

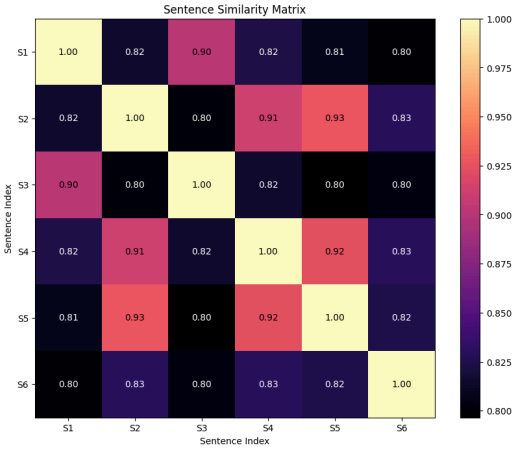
Observation

- Example sentences used in this task:

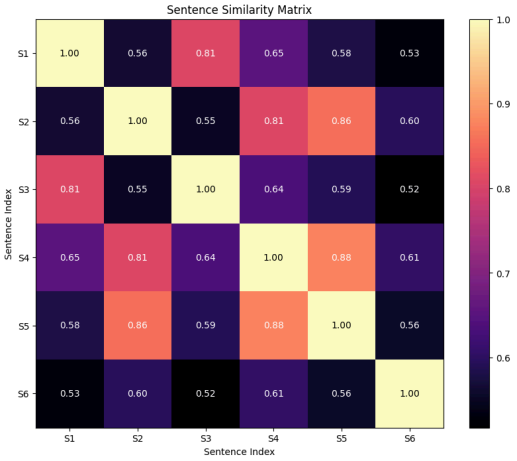
S1→ "The cat sat on the mat."
S2 → "The weather is nice today."
S3 → "A feline was resting on a rug."
S4 → "The sun is shining outside."
S5 → "It's a beautiful sunny day outside."
S6 → "I love working with transformers!"

- We can notice that S3 should be most similar to S1 while S5 and S6 should be least similar to S1

- Tested bert model(110m parameters) which is a popular choice for this task. The graphs below show the performance for max vs mean pooling strategy
  - it appears that mean pool performs much better in representing the similarity and dissimilarity between sentences as it showed a strong correlation between s1 and s3 while highlighting the stark difference between s1 with s5 and s6

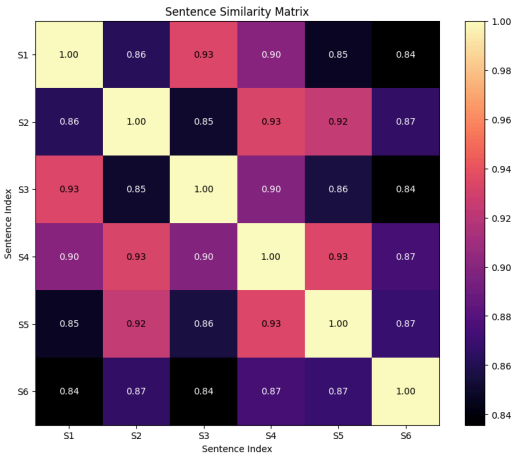


Bert with max pool

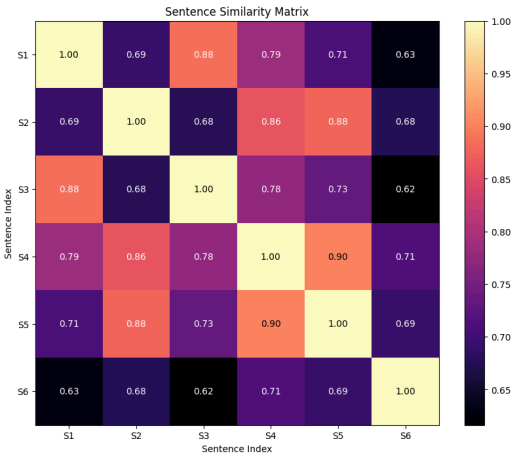


Bert with mean pool

- Tested Distillbert(66m parameters) which is a lighter and faster version of bert that still retains 97% of its performance. This is a preferred choice as its computationally efficient and performs well when dataset is small. The graphs below show the performance for max vs mean pooling strategy.
  - Mean pooling performs better than max pooling
  - distillbert is not able to show the contrast between s1 with s5 and s6 as strongly as bert
  - However, the performance is still good enough for our use case where we want to prioritize computational efficiency.



Distillbert with max



Distillbert with mean

Task 2: Multi-Task Learning Expansion

Model Architecture

- The model architecture is the same as before with distillbert and mean pooling which means sentence Transformer (transformer + projection + normalization) remains shared across both tasks.
- Two extra classification heads for each task has been added. After the forward pass the model returns the embeddings, Task A logits and Task B logits

```
MultiTaskSentenceTransformer(
  (transformer): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): DistilBertSdpaAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
```

```
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
    )
  )
 )
 (projection): Linear(in_features=768, out_features=512, bias=True)
 (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
 (taskA_classifier): Sequential(
  (0): Linear(in_features=512, out_features=256, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.6, inplace=False)
  (3): Linear(in_features=256, out_features=2, bias=True)
 )
 (taskB_classifier): Sequential(
  (0): Linear(in_features=512, out_features=256, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=256, out_features=2, bias=True)
 )
)
```

Part A: Sentence Classification

- Designed a binary sentence classification task where the labels are ‘Technology’ and ‘Entertainment’
- The output of the sentence embedding layer is passed through a classification head.
- The classification head consists of:
  - A fully connected (FC) layer that maps embeddings to class logits.
  - With ReLu activation and dropout of 0.6

Part B: sentiment analysis

- Designed a binary Sentiment analysis task where the labels are ‘Positive’ and ‘Negative’

- The output of the sentence embedding layer is passed through a classification head.
- The classification head consists of:
  - A fully connected (FC) layer that maps embeddings to class logits.
  - With ReLu activation and dropout of 0.5

Observation

- The idea is that the shared encoder will generate embeddings for input sentences **once**, which will be used for both tasks.
  - Common text representation benefits both tasks
- The task heads are separate even though the layers are the same because during training it will learning different features from the data depending on the objective of the task

Task 3: Training Considerations

If the entire network should be frozen

- implication → If the whole network is frozen, we arent really training anything. we are only using Distilbert as a feature extractor which doesnt get adapted to the specific labels we have
- advantage → There is no advantage of this. We should expect poor performance because there is no adaption to our specific task or no fine-tuning for domain-specific language patterns. The model only relies on its general language patterns that it already knows.
- how to train → To train this set 'param.requires_grad = False' for all layers of the model in our existing code. Alternatively, since the transformer is just a feature extractor, we can just get its output and funnel it through the task heads to get labels, omitting the train loop.

If only the transformer backbone should be frozen

- implication → This is a form of transfer learning as we are leveraging pre-trained weights while adapting the output layers for the specific tasks.
  - The transformer with pretrained weights are fantastic at extracting rich features from texts which is a mandatory shared step for both tasks.
- advantage → The advantage is that we get much faster training and lower memory requirements. It is ideal in a scenario where we have limited data
- how to train → To train we can simply add the following in the train function

```
for param in model.transformer.parameters():
    param.requires_grad = False
```

- This code ensures that only the transformer layers are frozen and the optimizer wont update the weights associated with them.

If only one of the task-specific heads should be frozen

- implication → This means the transformer component and one head is being trained (weights are being updated) while the second head is frozen. The problem is that the model might lose some of the useful pre-trained knowledge as it adjusts to our tasks. the frozen task will perform poorly and the unfrozen will do well.
- advantage → Its important to note that since transformer component is unfrozen, the model is very prone to overfitting with our limited data. We would need a significantly larger dataset to compensate for it. However, the model will still perform poorly for the frozen head.
- how to train → simply set param.requires_grad = False for the desired task head. eg:

```
for param in model.taskA_classifier.parameters():
    param.requires_grad = False
alpha = alpha//2 # Reduce the weight of taskA loss
```

- Note that we should also reduce the weight associated with the task specific loss because we dont want it to unfairly influence our training. More on that explained in Task 4

Transfer Learning

- choice of a pre-trained model → We could perform transfer learning with Distillbert because its lightweight, distilled version of BERT with 97% of its performance. Its pretrained on a large corpus of data that contains general knowledge. This means it is very good at capturing the semantic nuances in texts from various topics.
- layers you would freeze/unfreeze → The strategy would be to perform progressive unfreezing to have a better control of fine tuning and prevent catastrophic forgetting of pre-trained knowledge
  - First we could freeze the full transformer block and only train the 2 heads.This would retain the model's natural understanding capabilities while tuning the model for specific tasks
  - Then we can gradually unfreeze higher transformer layers backwards, depending on the performance.

- We want to keep the model's initial transformer layers frozen because those extract more basic linguistic patterns. The layers at the end encode more semantic information and thus might need more finetuning
- The dropout in each task head can be changed to control the importance of each task should the performance of one task dominate the other.

Task 4: Training Loop Implementation (BONUS)

Dataset Preparation

- MultiTaskDataset class is defined to create a custom dataset for training the multi task model. In a multi-task learning scenario, we need to handle data that serves multiple tasks simultaneously. This is done by creating a unified dataset where sentences have 2 pairs of labels corresponding to the 2 task heads.
- First we make sure the dataset size defined by user is divisible by 4 which is necessary to create a balanced class distribution. The class combinations are:

```
Tech & Negative
Tech & Positive
Entertainment & Negative
Entertainment & Positive
```

- Then we define base templates for each of these categories and a large collection of word banks to fill the templates.
  - This allows dynamically creating a large pool of sample sentences with hundreds of combinations.
  - This is an efficient strategy given we dont have real dataset of sentences and labels.
- _generate_category_samples class takes the templates and word banks to generate random sample sentences until our dataset size is reached.
- I tried a stratified approach to ensure class balance otherwise our model would be biased towards one class.
- Each sample contains:
  - A sentence (str)
  - A label for Task A (0: technology, 1: entertainment)
  - A label for Task B (0: negative, 1: positive)
- During training, dataset size set was to 1000 with 80-20 split
  - Train dataset size: 800 and Test dataset size: 200

    - Train class distribution:
      {('Tech', 'Neg'): 200, ('Tech', 'Pos'): 200, ('Entertain', 'Neg'): 200, ('Entertain', 'Pos'): 200}
    - Test class distribution:
      {('Tech', 'Neg'): 50, ('Tech', 'Pos'): 50, ('Entertain', 'Neg'): 50, ('Entertain', 'Pos'): 50}
    - Example sentences from training set:
      Technology & Negative: The AI assistant frequently confuses and is very defective.
      Technology & Positive: The router is incredibly incredible and intuitive.
      Entertainment & Negative: The series was predictable and failed to enhancing my attention.
      Entertainment & Positive: The comedy special was immersive and wonderfully revolutionizing.
- Note that the dataset is small. For more robustness we could incorporate a validation set and perform k fold cross validation.

Training Strategy

- The model is trained jointly on both tasks**.**This allows the shared encoder to learn generalized sentence representations.
- Forward Pass:
    - Transformer Processing: Tokenized inputs are fed through the transformer, whose weights can be frozen to maintain pre-trained representations.
    - Task-Specific Outputs: The forward pass produces a shared embedding and separate logits for Task A and Task B, enabling the model to perform multitask learning.
- Loss weighting:
    - Both tasks get CrossEntropyLoss and AdamW optimizer with a weight_decay=1e-5 and learning rate of 1e-5.
    - CrossEntropyLoss takes raw task logits and then computes the loss. It combines the softmax operation and negative log-likelihood loss internally, so is a good choice here. Crossentropyloss is a popular choice for classification task. Moreover, using it instead of of binary cross entropy loss directly allows flexibility should we choose to perform multi-class classification. The computational overhead is marginal and can be ignored.
    - Finally this is how loss is computed:

```
lossA = criterion(taskA_logits, taskA_labels)
lossB = criterion(taskB_logits, taskB_labels)
loss = alpha * lossA + beta * lossB # Combined loss
```

  - alpha and beta are weights associated with taskA and taskB loss respectively to control the influence each task has on the model.The model weights are updated based on this combined loss.
  - Shared layers receive gradients from both tasks and so results in more robust representations

- The training script allows selective freezing of the transformer or specific task classifiers, reducing the number of parameters updated during training using a flag at the beginning of the script
  - **Note** When we freeze one head but not the other, the gradients still affect the shared components. The unfrozen head's gradients may move shared components in directions that hurt the frozen task. so to compensate we reduce the weight associated with its loss.

Evaluation and Metrics

- Evaluate function evaluates the model on the test set and gives a final performance report to help us asses the quality of the trained model.
- For both training and evaluation, metrics such as accuracy, precision, recall, F1 score, and confusion matrix are calculated per task.
- The evaluation function additionally computes ROC curves and AUC, adapting the approach for binary versus multi class scenarios.
- These metrics provide insights into performance and help diagnose task-specific issues during training.
- visualize_metrics function plots multiple graphs based on the metrics dictionary. Metrics dictionary format:

```
metrics = {
    'overall': {
        'taskA_accuracy': taskA_acc,
        'taskB_accuracy': taskB_acc,
        'combined_accuracy': (taskA_acc + taskB_acc) / 2
    },
    'taskA': {
        'accuracy': taskA_acc,
        'precision': taskA_precision,
        'recall': taskA_recall,
```

```
            'f1': taskA_f1,
            'auc': taskA_roc_auc,
            'fpr': taskA_fpr,
            'tpr': taskA_tpr,
            'confusion_matrix': taskA_cm,

        },
        'taskB': {
            'accuracy': taskB_acc,
            'precision': taskB_precision,
            'recall': taskB_recall,
            'f1': taskB_f1,
            'auc': taskB_roc_auc,
            'fpr': taskB_fpr,
            'tpr': taskB_tpr,
            'confusion_matrix': taskB_cm,
        }
    }
```

Observation and issues

- The hyperparameters set for our training

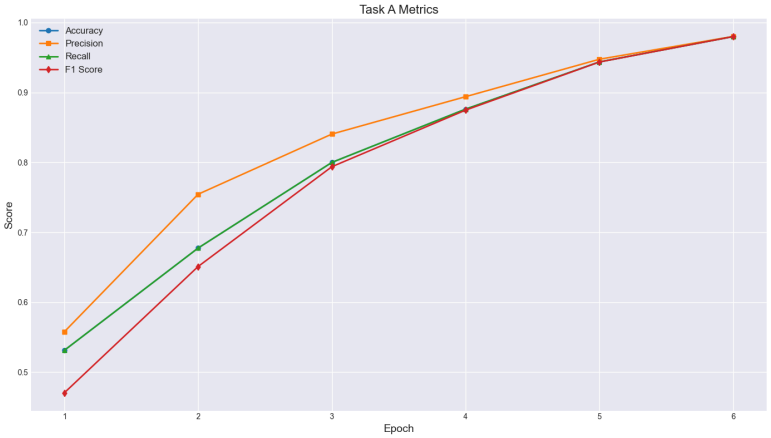
```
EMBED_DIM = 512
PRETRAINED_MODEL ='distilbert-base-uncased'
BATCH_SIZE = 32
LEARNING_RATE = 1e-5
NUM_EPOCHS = 6
DATASET_SIZE = 1000
freeze_transformer=True
freeze_taskA=False
freeze_taskB=False
alpha=0.6, beta=0.8
weight_decay=1e-5
```

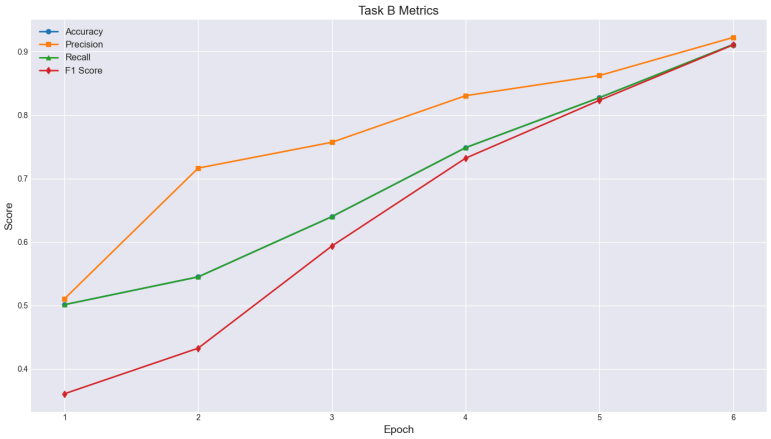
- The results obtained are

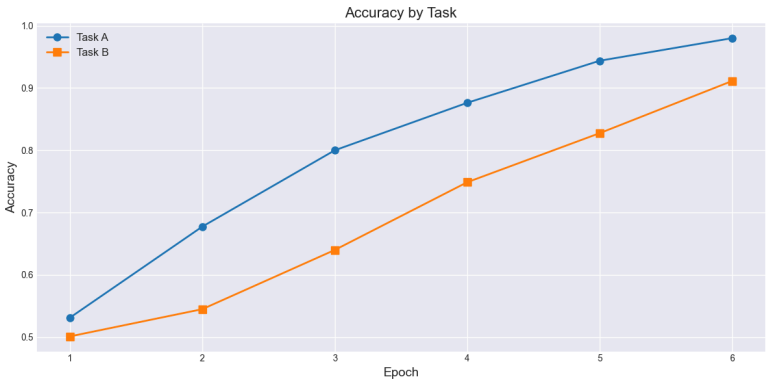
```
Training the model...
Epoch 1/6:
  Loss: 0.9687
  Task A - Acc: 0.5312, Prec: 0.5578, Rec: 0.5312, F1: 0.4705
  Task B - Acc: 0.5012, Prec: 0.5103, Rec: 0.5012, F1: 0.3608
Epoch 2/6:
```

```
  Loss: 0.9612
  Task A - Acc: 0.6775, Prec: 0.7545, Rec: 0.6775, F1: 0.6511
  Task B - Acc: 0.5450, Prec: 0.7165, Rec: 0.5450, F1: 0.4327
Epoch 3/6:
  Loss: 0.9541
  Task A - Acc: 0.8000, Prec: 0.8405, Rec: 0.8000, F1: 0.7939
  Task B - Acc: 0.6400, Prec: 0.7572, Rec: 0.6400, F1: 0.5937
Epoch 4/6:
  Loss: 0.9459
  Task A - Acc: 0.8762, Prec: 0.8940, Rec: 0.8762, F1: 0.8748
  Task B - Acc: 0.7488, Prec: 0.8306, Rec: 0.7488, F1: 0.7322
Epoch 5/6:
  Loss: 0.9370
  Task A - Acc: 0.9437, Prec: 0.9476, Rec: 0.9437, F1: 0.9436
  Task B - Acc: 0.8275, Prec: 0.8623, Rec: 0.8275, F1: 0.8233
Epoch 6/6:
  Loss: 0.9260
  Task A - Acc: 0.9800, Prec: 0.9802, Rec: 0.9800, F1: 0.9800
  Task B - Acc: 0.9113, Prec: 0.9224, Rec: 0.9113, F1: 0.9107
Training complete!

Evaluating the trained model...

Evaluation Results:
Task A - Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1: 1.0000
Task A - AUC: 1.0000
Task B - Accuracy: 0.9950, Precision: 0.9950, Recall: 0.9950, F1: 0.9950
Task B - AUC: 1.0000
Combined Accuracy: 0.9975
Evaluation complete!
All plots saved to .graphs/
```

- The graphs below plot the training metrics across all epochs.







- Training loss steadily decreased from 0.97 to 0.93 across epochs, showing stable optimization without plateauing.
- Task A metrics improved more rapidly in early epochs than Task B , particularly with F1 score (0.47→0.98 vs 0.36→0.91). This suggests Task A was inherently easier.
- Initially, precision outperformed recall for both tasks, but they converged by epoch 6, which might suggest that the model eventually balanced identifying positive cases with avoiding false positives. This is achieved by hyperparameter tuning where I had to try a few values for alpha , beta and dropout to get the right balance.

- Both tasks achieved near-perfect final metrics despite different learning trajectories, which means there is successful knowledge sharing between tasks within the multi-task sentence transformer.
- The ROC curves show both tasks achieved perfect discrimination with 100% AUC. AUC measures the quality of the ranking (how well the model can separate classes), and an AUC of 1.0 indicates that, in terms of ranking, the model is very good.
- The perfect evaluation result suggest the model memorized the test data rather than learning generalizable patterns. Some reasons are:
    - The dataset was incredibly short
    - and there is high chance of data leakage
- Performance is better in evaluation than in training indicates that the model has overfit because perfect scores are rarely the case in real life. Ways we can mitigate the problem:
    - incorporating k fold cross validation
    - ensuring test dataset is truly unseen data
    - Try different combination of hyperparameters to find what works best for this task