TASK 1

```
# Required Libraries
import torch
from transformers import AutoTokenizer, AutoModel
import numpy as np

# Load Pretrained BERT Model & Tokenizer
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

# Sentence Encoder with Mean Pooling
def mean_pooling(model_output, attention_mask):
    token_embeddings = model_output.last_hidden_state  # (batch_size, seq_len, hidden_size)
    input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
    return torch.sum(token_embeddings * input_mask_expanded, dim=1) / \
           torch.clamp(input_mask_expanded.sum(dim=1), min=1e-9)

def encode_sentences(sentences):
    encoded_input = tokenizer(sentences, padding=True, truncation=True, return_tensors='pt')
    with torch.no_grad():
        model_output = model(**encoded_input)
    sentence_embeddings = mean_pooling(model_output, encoded_input['attention_mask'])
    return sentence_embeddings

# Example Sentences
sentences = [
    "The quick brown fox jumps over the lazy dog.",
    "A fast brown animal leaps above a sleepy dog.",
    "The sky is blue and clear today."
]

embeddings = encode_sentences(sentences)
print("Sentence Embeddings:")
print(embeddings)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%                48.0/48.0 [00:00<00:00, 2.50kB/s]
config.json: 100%                          570/570 [00:00<00:00, 38.4kB/s]
vocab.txt: 100%                            232k/232k [00:00<00:00, 1.25MB/s]
tokenizer.json: 100%                       466k/466k [00:00<00:00, 2.55MB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfo
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
model.safetensors: 100%                    440M/440M [00:01<00:00, 276MB/s]
Sentence Embeddings:
tensor([[-0.0145, -0.0749,  0.0564,  ..., -0.2625,  0.4954,  0.0740],
        [ 0.2230,  0.0594, -0.1303,  ..., -0.2426,  0.1587, -0.0096],
        [ 0.2534, -0.2380,  0.0057,  ..., -0.0501,  0.3261,  0.0875]])
```

TASK 2

```
# Required Libraries
import torch
import torch.nn as nn
from transformers import AutoTokenizer, AutoModel
import numpy as np

# Load Pretrained BERT Model & Tokenizer
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# === Multi-Task Model Definition ===
```

```python
class SentenceTransformerMultiTask(nn.Module):
    def __init__(self, model_name, num_classes_a=3, num_classes_b=3):
        super(SentenceTransformerMultiTask, self).__init__()
        self.encoder = AutoModel.from_pretrained(model_name)
        hidden_size = self.encoder.config.hidden_size

        # Task A: Sentence Classification
        self.classifier_a = nn.Sequential(
            nn.Linear(hidden_size, 256),
            nn.ReLU(),
            nn.Linear(256, num_classes_a)
        )

        # Task B: Sentiment Analysis
        self.classifier_b = nn.Sequential(
            nn.Linear(hidden_size, 128),
            nn.ReLU(),
            nn.Linear(128, num_classes_b)
        )

    def mean_pooling(self, model_output, attention_mask):
        token_embeddings = model_output.last_hidden_state
        mask_exp = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
        return torch.sum(token_embeddings * mask_exp, dim=1) / torch.clamp(mask_exp.sum(dim=1), min=1e-9)

    def forward(self, input_ids, attention_mask):
        output = self.encoder(input_ids=input_ids, attention_mask=attention_mask)
        pooled = self.mean_pooling(output, attention_mask)

        logits_a = self.classifier_a(pooled)
        logits_b = self.classifier_b(pooled)

        return logits_a, logits_b, pooled

# === Sample Input and Inference ===
# Example Sentences
sentences = [
    "Is it going to rain tomorrow?",              # Question, Neutral
    "The stock market is crashing badly!",        # News, Negative
    "I'm so happy about my promotion!"            # Statement, Positive
]

# Tokenize input
encoded_input = tokenizer(sentences, padding=True, truncation=True, return_tensors='pt')

# Initialize model
multi_task_model = SentenceTransformerMultiTask(model_name)

# Forward pass
with torch.no_grad():
    logits_a, logits_b, embeddings = multi_task_model(encoded_input['input_ids'], encoded_input['attention_mask'])

# Print outputs
print("\n[Task A] Sentence Classification Logits:")
print(logits_a)

print("\n[Task B] Sentiment Analysis Logits:")
print(logits_b)

print("\n[Shared] Sentence Embeddings:")
print(embeddings)
```

```

[Task A] Sentence Classification Logits:
tensor([[ 0.1006,  0.0342, -0.1837],
        [ 0.0350,  0.0588, -0.1790],
        [ 0.0508, -0.0007, -0.1427]])

[Task B] Sentiment Analysis Logits:
tensor([[-0.0699,  0.0092, -0.1204],
        [ 0.0334,  0.0132, -0.1114],
        [-0.0071, -0.0613, -0.1652]])

[Shared] Sentence Embeddings:
tensor([[ 0.3509, -0.3380,  0.5369,  ..., -0.2460,  0.0208,  0.0150],
        [ 0.1490, -0.1024, -0.1217,  ..., -0.1452,  0.1415, -0.4081],
        [ 0.1561, -0.0719,  0.4475,  ...,  0.0036,  0.1840,  0.0846]])
```

TASK 3

Training Scenarios

1: Entire Network is Frozen- Neither the transformer backbone nor the task-specific heads are updated during training.

Implication:

- The model behaves like a feature extractor.
- Useful when you have very limited data or want very fast inference.

Advantage:

- Avoids overfitting on small datasets.
- Efficient training or no training.

Disadvantage:

- Model can't adapt to task-specific nuances.

Use Case:

- Quick prototyping, zero-shot tasks, or when using external embeddings in downstream ML pipelines.

2. Only the Transformer Backbone is Frozen- The BERT encoder is frozen; only task-specific heads are trained.

Implication:

- The model uses pre-trained semantic knowledge from BERT.
- The classifier layers learn to map these embeddings to the task labels.

Advantage:

- Faster training.
- Lower risk of overfitting.
- Good when you have moderate-size labeled datasets.

Disadvantage:

- Limited adaptability if BERT embeddings are not well-suited to your specific domain.

Use Case:

- When you have domain-agnostic tasks and limited computational resources.

3. Only One Task-Specific Head is Frozen- Either Task A or Task B head is frozen the rest of the model is trainable.

Implication:

- You can retain performance on one task while improving another.
- Useful when adding a new task to an existing multi-task model.

Advantage:

- Supports continual learning or transfer to new tasks.

Disadvantage:

- Risk of catastrophic forgetting if not balanced well.

Use Case:

- You're adapting a model trained for sentiment analysis to now also do topic classification.

Transfer Learning Process

Example Transfer Learning Setup

1. **Choice of Pre-trained Model**: Pick a model like bert-base, BioBERT for biomedical text, DistilBERT for efficiency. It leverages semantic understanding learned from millions of sentences.
2. **Freezing Strategy:** Lower BERT layers - Freeze - They capture general syntax/structure (shared across tasks). Higher BERT layers- Unfreeze- They capture task/domain-specific semantics. Task-specific heads-Unfreeze-Must adapt to your custom labels and objectives. You can also gradually unfreeze layers (“layer-wise unfreezing”) if you fine-tune carefully.
3. **Rationale:** Prevent overfitting in early stages. Leverage general language understanding from pretraining. Fine-tune only where task/domain-specific adaptation is needed.

Summary:

In training a multi-task sentence transformer model, different strategies can be applied based on data availability, compute resources, and task requirements. Freezing the entire network is useful for zero-shot or inference-only scenarios where adaptation isn't needed, offering speed and preventing overfitting. Freezing only the transformer backbone allows the model to leverage pretrained embeddings while adapting task-specific heads, making it ideal for moderate datasets. Freezing just one task head helps when adding a new task while retaining performance on an existing one. In transfer learning, starting with a well-suited pretrained model (like BERT or BioBERT), freezing lower layers to preserve general language features, and fine-tuning upper layers and task heads for domain-specific adaptation is effective. This approach balances stability and flexibility, enabling efficient learning without overfitting or catastrophic forgetting.

TASK 4

Summary:

In a multi-task learning setup, a shared transformer encoder (like BERT) is combined with task-specific heads—for example, one for sentence classification and another for sentiment analysis. During training, input sentences are tokenized and passed through the shared encoder. Each task head then produces its own output. For each task, a separate loss (e.g., CrossEntropyLoss) is computed, and the total loss is typically the sum or a weighted sum of these. Backpropagation updates the model based on this total loss. Metrics such as accuracy are tracked individually for each task to monitor performance. Dummy data and labels can be used to simulate the training process, and PyTorch's `DataLoader` helps manage batching. This approach enables efficient learning by allowing tasks to share underlying linguistic knowledge while specializing in their respective outputs.