**Machine Learning assignment Explanation**

**model.py:**

Contains the base transformer model which is utilized. **BERT model­­**

**Sample\_data.py:**

Contains random sample data for our tasks:

Task A: sentence classification

Task B: sentiment Analysis

**TASK 1:**

Import the model from the model.py file which uses loads the pre-trained BERT model (bert-base-uncased) and its tokenizer using Transformers library.

Model Selection: I selected the BERT Transformer model as its pre-training on massive text, ability to consider bidirectional context in sentences make it ideal for sentence embedding & tasks like sentiment analysis, classification.

Processing Sample Data:

* Retrieves sample sentences.
* Loads the BERT model and tokenizer.
* Generates sentence embeddings for the sample sentences.
* Fixed length embeddings are produced by using the mean pooling of tokens
* Prints the first few elements of each embedding for illustration.

**TASK 2:**

**1. Multi-Task Model Definition:**

• The MultiTaskModel class inherits from nn.Module and accepts a pre-trained base model and the number of classes for two tasks (A and B) as input.

The \_\_init\_\_ method stores the underlying model and builds linear layers for task classification.

The forward method uses mean pooling, softmax, and provides a dictionary with task A and B outputs.

**Changes made to base Model**

1] We construct separate task-specific heads.

2] After acquiring the sentence embeddings, distinct heads are created for each task A, B (sentence classifying and sentiment analysis).

These heads are linear layers that use the text embedding as input to forecast the number of classes for each task.

3] We make no changes to the base pre-trained BERT model.

* Using a BERT pre-trained model for feature extraction saves us from having to build a huge model from scratch for both Tasks individually. This saves both training time and computational resources.
* Knowledge Transfer:

Since the tasks are related, the pre-trained model can help with both tasks . BERT's common grasp of language can increase performance on both tasks A and Task B over training separate models from scratch

* Flexibility:

This architecture makes it easy to add new tasks in the future. As long as they can use sentence embedding, you can add new heads for those tasks without retraining the model.

**TASK 3:**

How to train the model by freezing various parts:

**Freezing Choices:**

1. Using the complete network allows for faster training and less adaptation, which is useful for sentence classification and sentiment analysis because they are similar tasks.

2. The transformer backbone enables faster training by utilizing pre-trained information.

3. One Task Head: Retains knowledge of one task while adapting to another. Task A adapts while task B preserves its knowledge, and vice versa.

**Transfer Learning Approach:**

1. Pre-trained model: I have choose a high-performing model that meets the assignments requirements (for example, BERT for sentiment and sentence classification).   
  
2. Freezing and unfreezing: First, freeze the transformer backbone, then unfreeze the task-specific heads for tasks A and B. Consider unfreezing the last transformer layers.   
  
Rationale: Use pre-trained information to boost productivity while allowing for adaptability to multiple activities. Experiment to find the optimal technique for your data and tasks.

As the pre-trained model [BERT] is really relevant we can :

- Freeze the early layers (common low-level features).   
- Unfreeze the later layers (task-specific features).  
  
If the pre-trained model was less relevant, we can unfreeze additional layers to adapt to the new demands.

**TASK 4:**  
The Multi-Task Model with Layer-Wise Learning Rates improves on the preceding model by using separate classifiers for tasks A and B, as well as various learning rates and weight decay for different network sections during training.

**Key Points:**

• Groups parameters by location and attributes (e.g. bias, weight decay).   
• Uses add\_param\_group to prevent parameters from being added numerous times.   
• Generates optimizer groups with tailored learning rates and weight decay: Lower levels of the pre-trained model (excluding layer 11): Small learning rate, regular weight decay (encourages fine-tuning and prevents overfitting).   
• Layer 11 of the pre-trained model: Higher learning rate and regular weight decay ,allowing for greater adaptation in this layer.   
• Task-specific classifiers: Maximum learning rate with no weight decay focuses on learning task-specific representations.

**Training:**

* Iterates for one epoch (typically many epochs in practice).
  + Clears gradients.
  + Performs a forward pass.
  + Calculates loss for each task.
  + Combines losses.
  + Backpropagates the loss.
  + Updates model parameters with the optimizer considering defined groups.

The code configures distinct learning rates and weight decay values for different groups of model parameters, for layer-wise learning rates. It allows various sections of the model to learn at different rates hence increasing training efficiency.

Lower learning rates are used to fine-tune the parameters of the underlying model, whereas higher learning rates are utilized for newly added layers and task-specific

classifiers to learn faster.

Weight decay, helps prevent overfitting, eliminating

specific parameters like as biases. This careful adjustment keeps the model stable while it learns task-specific properties.