Task 4: Training Loop Implementation (BONUS)

ML Apprentice Take-Home Exercise

Introduction

This document describes the design and structure of a multi-task training loop for the `MultiTaskTransformer` model (sentence classification + NER). We focus on:

- Handling of hypothetical batched data,
- The forward pass and combined loss computation,
- Metrics tracking for each task.

No actual training is executed here; the code is illustrative.

1 Assumptions and Data Handling

We assume a `DataLoader` yielding batches as dictionaries with:

- `input_ids`, `attention_mask`: `Tensor` of shape (B, L),
- `task_a_labels`: `Tensor` of shape $(B,)$ for sentence classes,
- `task_b_labels`: `Tensor` of shape (B, L) for token tags, padded with `-100` to mask in loss.

A custom `collate_fn` ensures correct stacking and padding:

```
def collate_fn(batch):
    input_ids      = torch.stack([ex['input_ids'] for ex in batch])
    attention_mask= torch.stack([ex['attention_mask'] for ex in batch])
    task_a_labels = torch.tensor([ex['task_a_labels'] for ex in batch])
    task_b_labels = torch.stack([
        F.pad(torch.tensor(ex['task_b_labels']), (0, L - len(ex['task_b_labels']))
            , value=-100)
        for ex in batch
    ])
    return {
        'input_ids': input_ids,
        'attention_mask': attention_mask,
        'task_a_labels': task_a_labels,
        'task_b_labels': task_b_labels
    }
```

2 Forward Pass and Loss Computation

Each batch undergoes a single forward pass:

```
# 1. Forward pass yielding two outputs:
logits_a, logits_b = model(input_ids, attention_mask)
# logits_a: (B, C_a), sentence classification
# logits_b: (B, L, C_b), token-level NER

# 2. Compute per-task losses:
loss_a = loss_fn_a(logits_a, task_a_labels)             # CrossEntropy on (B, C_a)
    vs (B,)
loss_b = loss_fn_b(
    logits_b.view(-1, C_b),                              # flatten to (B*L, C_b)
    task_b_labels.view(-1)                               # flatten to (B*L,)
)

# 3. Combined multi-task loss:
loss = loss_a + loss_b                                  # equal weighting
```

Where `C_a` is the number of sentence classes (e.g., 2) and `C_b` is the number of NER tags (e.g., 4).

3 Metrics Computation

We track separate accuracy metrics for each task:

- **Sentence Classification Accuracy (Task A):**

$$\text{Accuracy}_A = \frac{1}{B}\sum_{i=1}^{B} \mathbf{1}\big(\arg\max(\text{logits}_A^{(i)}) = y_A^{(i)}\big),$$

  where B is the batch size, $\text{logits}_A^{(i)}$ are the class scores for example i, and $y_A^{(i)}$ is its true label.

- **Token-Level NER Accuracy (Task B):**

$$\text{Accuracy}_B = \frac{\sum_{i=1}^{B}\sum_{j=1}^{L} \mathbf{1}\big(\hat{y}_{i,j} = y_{i,j}^{B}\big)\, m_{i,j}}{\sum_{i=1}^{B}\sum_{j=1}^{L} m_{i,j}},$$

  where:

  - $\hat{y}_{i,j} = \arg\max(\text{logits}_B^{(i,j)})$ is the predicted tag for token j in example i,
  - $y_{i,j}^{B}$ is the true tag (or -100 if padded),
  - $m_{i,j} = \mathbf{1}(y_{i,j}^{B} \neq -100)$ masks out padding positions,
  - L is the maximum sequence length.

4 Training Loop Pseudocode

Putting it all together, a single epoch looks like:

```
for epoch in range(num_epochs):
    model.train()
    total_loss = correct_a = total_a = correct_b = total_b = 0

    for batch in train_loader:
        optimizer.zero_grad()

        # Forward + loss
        logits_a, logits_b = model(batch['input_ids'], batch['attention_mask'])
```

```
        loss_a = loss_fn_a(logits_a, batch['task_a_labels'])
        loss_b = loss_fn_b(
            logits_b.view(-1, C_b), batch['task_b_labels'].view(-1)
        )
        loss = loss_a + loss_b

        # Backward & update
        loss.backward()
        optimizer.step()

        # Accumulate loss
        total_loss += loss.item()

        # Task A accuracy
        preds_a = logits_a.argmax(dim=1)
        correct_a += (preds_a == batch['task_a_labels']).sum().item()
        total_a   += batch['task_a_labels'].size(0)

        # Task B accuracy (mask padding)
        preds_b = logits_b.argmax(dim=2)
        mask    = batch['task_b_labels'] != -100
        correct_b += ((preds_b == batch['task_b_labels']) & mask).sum().item()
        total_b   += mask.sum().item()

    # Epoch metrics
    avg_loss = total_loss / len(train_loader)
    acc_a    = correct_a / total_a
    acc_b    = correct_b / total_b
    print(f"Epoch {epoch}: Loss={avg_loss:.4f}, "
          f"AccA={acc_a:.4f}, AccB={acc_b:.4f}")
```

Conclusion

This multi-task training loop:

- Efficiently handles batched data with custom collation.
- Executes a single shared forward pass for both tasks.
- Computes and aggregates separate metrics to monitor each task's performance.

Joint backpropagation of both losses encourages the shared encoder to learn representations useful across tasks, improving overall generalization in a multi-task learning framework.