## Question to be answered (Q2.3)

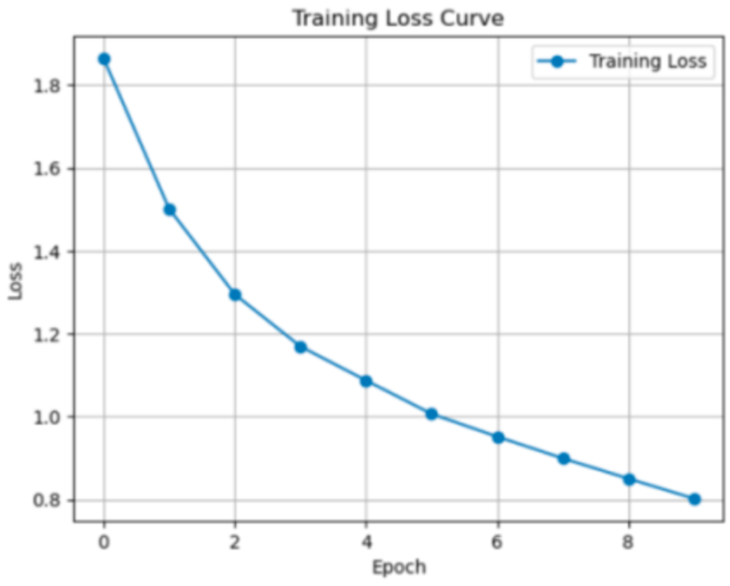
***What is the difference between the training and testing stage of a transformer?***

1. The training stage is for the model to learn from a large amount of data, so that it can accurately understand and generate predictions. The testing stage is to evaluate the model's performance on unseen data.
2. During the training process, the model makes predictions based on the input data, and its output is compared with the actual target data. A loss function is used to measure the difference between the model's predictions and the actual data. Then, through backpropagation and optimization algorithms, the model's parameters are adjusted to minimize this loss. However, in the testing process, there is no need to adjust the model's parameters.

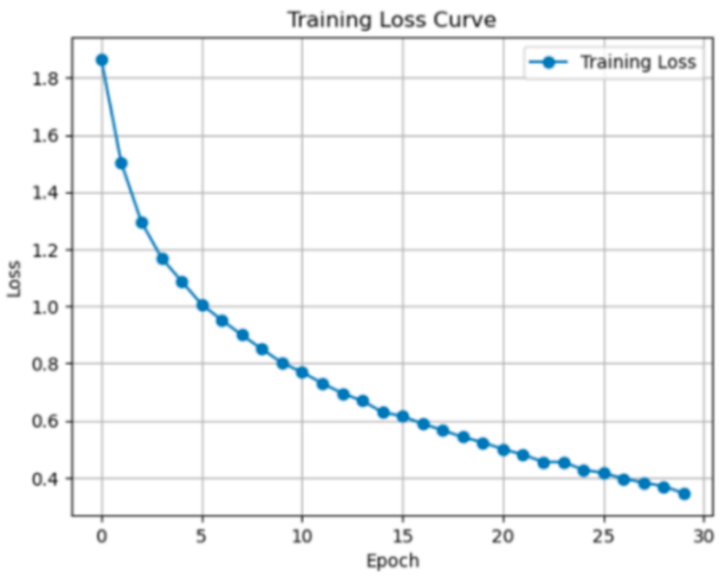
## Exercise to be conducted (E3.5)

***To plot the training loss curve to show its variation with the epoch.***

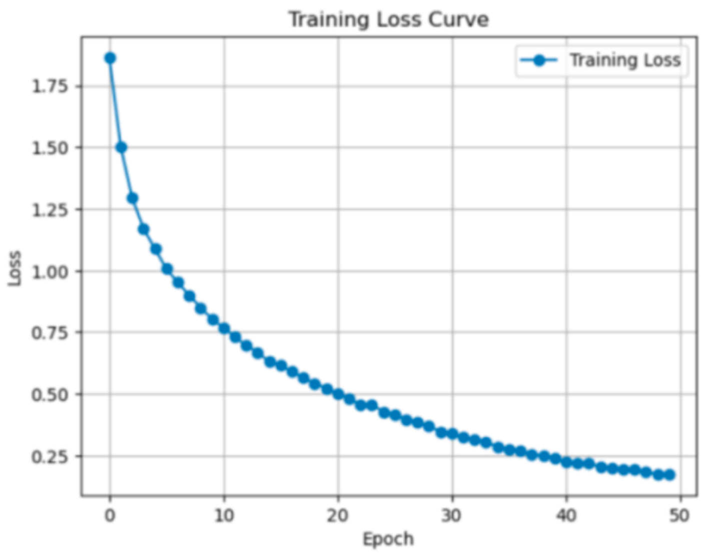
training loss curve that varies with epochs for three different settings: 10 epochs, 30 epochs, and 50 epochs:
10 epochs:



30 epochs:

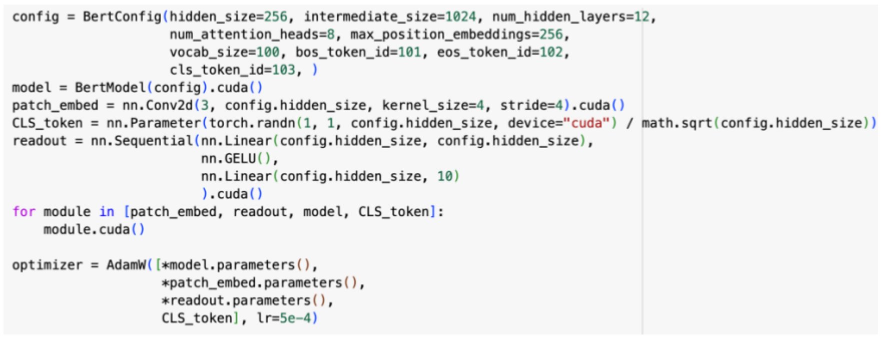


50 epochs:



Besides plotting, the following includes the model and training process that needs to be executed:

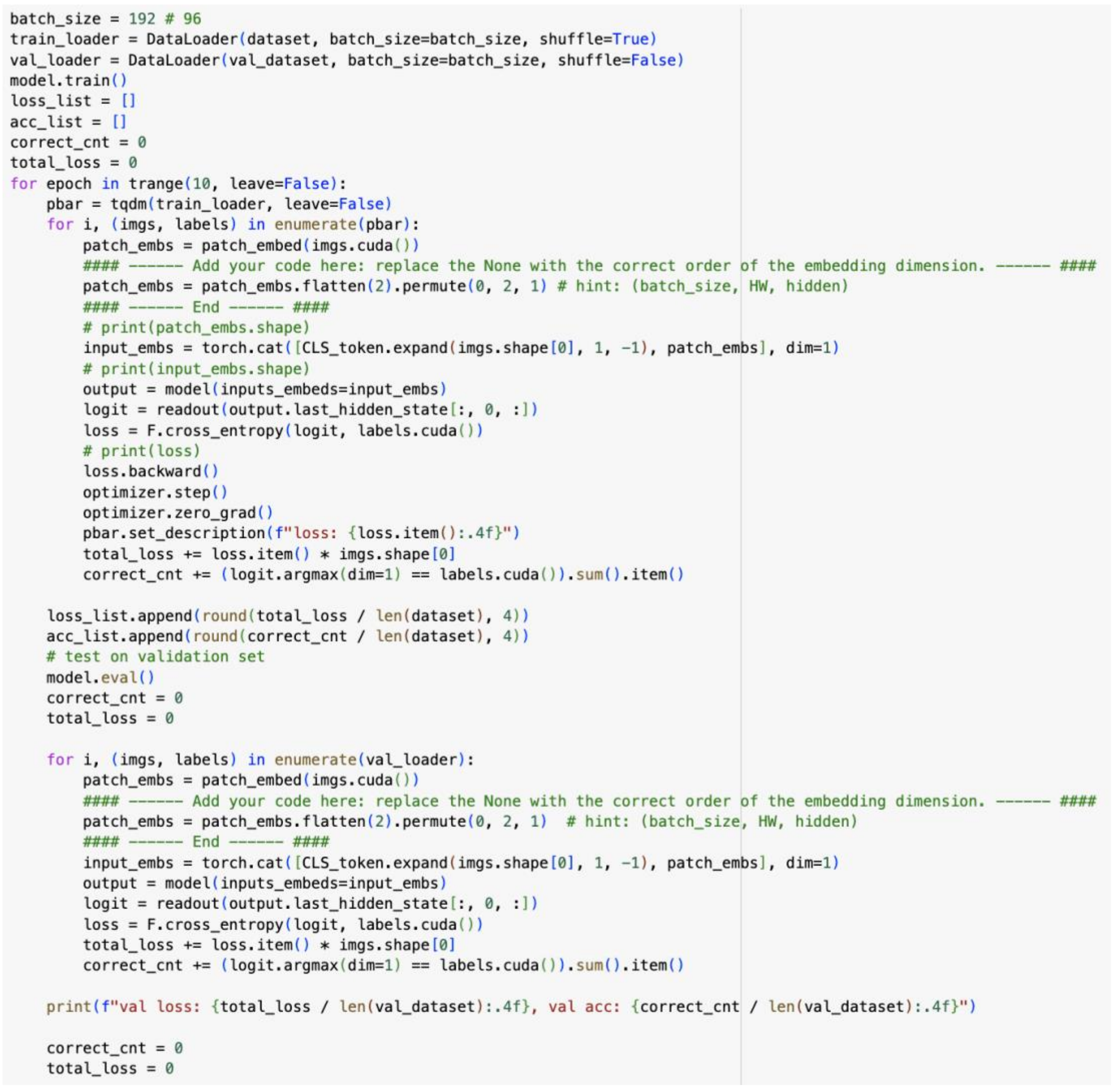
Here is the model:



```python
config = BertConfig(hidden_size=256, intermediate_size=1024, num_hidden_layers=12,
                    num_attention_heads=8, max_position_embeddings=256,
                    vocab_size=100, bos_token_id=101, eos_token_id=102,
                    cls_token_id=103, )
model = BertModel(config).cuda()
patch_embed = nn.Conv2d(3, config.hidden_size, kernel_size=4, stride=4).cuda()
CLS_token = nn.Parameter(torch.randn(1, 1, config.hidden_size, device="cuda") / math.sqrt(config.hidden_size))
readout = nn.Sequential(nn.Linear(config.hidden_size, config.hidden_size),
                        nn.GELU(),
                        nn.Linear(config.hidden_size, 10)
                        ).cuda()
for module in [patch_embed, readout, model, CLS_token]:
    module.cuda()

optimizer = AdamW([*model.parameters(),
                   *patch_embed.parameters(),
                   *readout.parameters(),
                   CLS_token], lr=5e-4)
```

Here is the training process and test process:



```python
batch_size = 192 # 96
train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
model.train()
loss_list = []
acc_list = []
correct_cnt = 0
total_loss = 0
for epoch in trange(10, leave=False):
    pbar = tqdm(train_loader, leave=False)
    for i, (imgs, labels) in enumerate(pbar):
        patch_embs = patch_embed(imgs.cuda())
        #### ------ Add your code here: replace the None with the correct order of the embedding dimension. ------ ####
        patch_embs = patch_embs.flatten(2).permute(0, 2, 1) # hint: (batch_size, HW, hidden)
        #### ------ End ------ ####
        # print(patch_embs.shape)
        input_embs = torch.cat([CLS_token.expand(imgs.shape[0], 1, -1), patch_embs], dim=1)
        # print(input_embs.shape)
        output = model(inputs_embeds=input_embs)
        logit = readout(output.last_hidden_state[:, 0, :])
        loss = F.cross_entropy(logit, labels.cuda())
        # print(loss)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        pbar.set_description(f"loss: {loss.item():.4f}")
        total_loss += loss.item() * imgs.shape[0]
        correct_cnt += (logit.argmax(dim=1) == labels.cuda()).sum().item()

    loss_list.append(round(total_loss / len(dataset), 4))
    acc_list.append(round(correct_cnt / len(dataset), 4))
    # test on validation set
    model.eval()
    correct_cnt = 0
    total_loss = 0

    for i, (imgs, labels) in enumerate(val_loader):
        patch_embs = patch_embed(imgs.cuda())
        #### ------ Add your code here: replace the None with the correct order of the embedding dimension. ------ ####
        patch_embs = patch_embs.flatten(2).permute(0, 2, 1)  # hint: (batch_size, HW, hidden)
        #### ------ End ------ ####
        input_embs = torch.cat([CLS_token.expand(imgs.shape[0], 1, -1), patch_embs], dim=1)
        output = model(inputs_embeds=input_embs)
        logit = readout(output.last_hidden_state[:, 0, :])
        loss = F.cross_entropy(logit, labels.cuda())
        total_loss += loss.item() * imgs.shape[0]
        correct_cnt += (logit.argmax(dim=1) == labels.cuda()).sum().item()

    print(f"val loss: {total_loss / len(val_dataset):.4f}, val acc: {correct_cnt / len(val_dataset):.4f}")

    correct_cnt = 0
    total_loss = 0
```