# Task no: 1

## Code:

```
# ==========================
# Text Classification with BERT (Complete & Correct)
# ==========================
# Install libraries
!pip install transformers datasets accelerate -q
# 1. Imports
import torch
import numpy as np
import random
from torch.utils.data import DataLoader
from torch.optim import AdamW
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    get_scheduler,
    DataCollatorWithPadding
)
from datasets import load_dataset
from sklearn.metrics import accuracy_score
# Set seed for reproducibility
seed = 42
torch.manual_seed(seed)
```

```python
np.random.seed(seed)
random.seed(seed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(seed)
# 2. Load Dataset (AG News)
dataset = load_dataset("ag_news")
train_dataset = dataset["train"].select(range(2000))   # Smaller subset for quick training
test_dataset = dataset["test"].select(range(500))      # Smaller test set
# 3. Tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
def tokenize_fn(batch):
    return tokenizer(batch["text"], truncation=True, max_length=256)
# Tokenize datasets
train_dataset = train_dataset.map(tokenize_fn, batched=True)
test_dataset = test_dataset.map(tokenize_fn, batched=True)
# Rename column
train_dataset = train_dataset.rename_column("label", "labels")
test_dataset = test_dataset.rename_column("label", "labels")
# Set format
train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])
test_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])
# 4. DataLoaders with dynamic padding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

```
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True,
collate_fn=data_collator)

test_loader = DataLoader(test_dataset, batch_size=16,
collate_fn=data_collator)

# 5. Model

model = AutoModelForSequenc    "bert-base-uncased",

eClassification.from_pretrained(

    num_labels=4

)

# 6. Optimizer

optimizer = AdamW(model.parameters(), lr=2e-5)  # Slightly lower learning
rate

# 7. Training Setup

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print(f"Using device: {device}")

model.to(device)

num_epochs = 1

num_training_steps = len(train_loader) * num_epochs

lr_scheduler = get_scheduler(

    "linear",

    optimizer=optimizer,

    num_warmup_steps=0,

    num_training_steps=num_training_steps

)

# 8. Training Loop

model.train()

print("Starting training...")
```

```
for epoch in range(num_epochs):
    total_loss = 0
    for step, batch in enumerate(train_loader):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        total_loss += loss.item()
        if (step + 1) % 50 == 0:
            avg_loss = total_loss / 50
            print(f"Step {step + 1}/{len(train_loader)} - Loss: {avg_loss:.4f}")
            total_loss = 0
print("✅ Training Finished!")
# 9. Evaluation
model.eval()
all_preds = []
all_labels = []
print("Running evaluation...")
with torch.no_grad():
    for batch in test_loader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        preds = torch.argmax(outputs.logits, dim=-1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(batch["labels"].cpu().numpy())
acc = accuracy_score(all_labels, all_preds)
print(f"✅ Test Accuracy: {acc:.4f}")
print(f"✅ Test Samples: {len(all_labels)}")
```