

In [2]:

```
!pip install datasets tqdm scikit-learn
```

```
Requirement already satisfied: datasets in /usr/local/lib/python3.12/dist-packages (4.0.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (4.67.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from datasets) (3.20.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.12/dist-packages (from datasets) (2.32.4)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from datasets) (3.6.0)
Requirement already satisfied: multiprocessing<0.70.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (2025.3.0)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (1.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from datasets) (26.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from datasets) (6.0.3)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: aiohttp!=4.0.0a0,!>4.0.0a1 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (3.13.3)
Requirement already satisfied: hf-xet<2.0.0,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (1.2.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (0.28.1)
Requirement already satisfied: shellingham in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (1.5.4)
Requirement already satisfied: typer-slim in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (0.21.1)
Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (4.15.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (2026.1.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2025.3)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!>4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (2.6.1)
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-package
```

```
s (from aiohttp!=4.0.0a0,!~=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.4.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!~=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (25.4.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!~=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.8.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!~=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (6.7.1)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!~=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (0.4.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!~=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.22.0)
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub>=0.24.0->datasets) (4.12.1)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub>=0.24.0->datasets) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx<1,>=0.23.0->huggingface-hub>=0.24.0->datasets) (0.16.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas->datasets) (1.17.0)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.12/dist-packages (from typer-slim->huggingface-hub>=0.24.0->datasets) (8.3.1)
```

In [3]:

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

In [64]:

```
import os, random
import torch
import torch.nn as nn
import torch.nn.functional as F
import math
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from tqdm import tqdm
from datasets import load_dataset
from collections import Counter
```

In [65]:

```
BASE = "/content/drive/MyDrive/Colab Notebooks/A4-do-you-agree"
ARTIFACTS = BASE + "/artifacts"
os.makedirs(ARTIFACTS, exist_ok=True)
```

In [66]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using:", device)
```

Using: cuda

TASK 1: Training BERT from Scratch

- Load Dataset

In [21]:

```
dt = load_dataset("wikitext", "wikitext-2-raw-v1")

texts = [t.strip() for t in dt["train"]["text"] if len(t.split()) > 5]
texts = texts[:100000] # 100k samples
```

```
print("Samples:", len(texts))
print(texts[0])
```

Samples: 21224

Senjō no Valkyria 3 : Unrecorded Chronicles (Japanese : 戦場のヴァルキリア3 , lit . Valkyria of the Battlefield 3) , commonly referred to as Valkyria Chronicles III outside Japan , is a tactical role @-@ playing video game developed by Sega and Media.Vision for the PlayStation Portable . Released in January 2011 in Japan , it is the third game in the Valkyria series . Employing the same fusion of tactical and real @-@ time gameplay as its predecessors , the story runs parallel to the first game and follows the " Nameless " , a penal military unit serving the nation of Gallia during the Second European War who perform secret black operations and are pitted against the Imperial unit " Calamity Raven " .

Dataset: We use WikiText-2 (wikitext-2-raw-v1) from HuggingFace datasets. After filtering short lines (<5 tokens), the dataset contains 21,224 training samples. This subset is used to enable efficient training

- **Build Vocabulary**

In [22]:

```
special_tokens = "[PAD]", "[CLS]", "[SEP]", "[MASK]", "[UNK]"]

words = " ".join(texts).lower().split()
vocab = Counter(words)

vocab_size = 20000
most_common = vocab.most_common(vocab_size)

word2idx = {t:i for i,t in enumerate(special_tokens)}
for w,_ in most_common:
    if w not in word2idx:
        word2idx[w] = len(word2idx)

idx2word = {i:w for w,i in word2idx.items()}

PAD = word2idx["[PAD]"]
CLS = word2idx["[CLS]"]
SEP = word2idx["[SEP]"]
MASK = word2idx["[MASK]"]
UNK = word2idx["[UNK]"]

print("Vocab size:", len(word2idx), "PAD id:", PAD)
```

Vocab size: 20005 PAD id: 0

- **Tokenizer**

In [23]:

```
def tokenize(text):
    return [word2idx.get(w, UNK) for w in text.lower().split()]
```

- **MLM Mask Function**

In [25]:

```
def bert_mlm_mask(input_ids, mlm_prob=0.15):    # 80% : 10% : 10%
    labels = [-100] * len(input_ids)
    masked = input_ids.copy()

    for i in range(len(input_ids)):
        token_id = input_ids[i]
```

```

if token_id in (PAD, CLS, SEP):
    continue

if random.random() < mlm_prob:
    labels[i] = token_id
    r = random.random()
    if r < 0.8:
        masked[i] = MASK
    elif r < 0.9:
        masked[i] = random.randint(len(special_tokens), len(word2idx)-1)
    else:
        masked[i] = token_id
return masked, labels

```

- **Make Sample**

In [26]:

```

max_len = 128

def make_sample(text):
    tokens = tokenize(text) [:max_len-2]
    input_ids = [CLS] + tokens + [SEP]

    # attention mask: 1=real token, 0=pad
    attn_mask = [1] * len(input_ids)

    # pad
    pad_len = max_len - len(input_ids)
    input_ids = input_ids + [PAD]*pad_len
    attn_mask = attn_mask + [0]*pad_len

    masked_ids, labels = bert_mlm_mask(input_ids)

    return masked_ids, labels, attn_mask

```

- **Mini BERT Model**

In [27]:

```

class MiniBERT(nn.Module):
    def __init__(self, vocab_size, hidden=256, max_len=64, n_layers=4, n_heads=4, dropout=0.1):
        super().__init__()
        self.hidden = hidden
        self.max_len = max_len

        self.tok_embed = nn.Embedding(vocab_size, hidden, padding_idx=PAD)
        self.pos_embed = nn.Embedding(max_len, hidden)

        self.ln = nn.LayerNorm(hidden)
        self.drop = nn.Dropout(dropout)

        enc_layer = nn.TransformerEncoderLayer(d_model=hidden, nhead=n_heads, dropout=dropout, batch_first=True, activation="gelu")
        self.encoder = nn.TransformerEncoder(enc_layer, num_layers=n_layers)
        self.mlm_head = nn.Linear(hidden, vocab_size)

    def forward(self, input_ids, attn_mask):

        B, L = input_ids.shape
        pos = torch.arange(L, device=input_ids.device).unsqueeze(0).expand(B, L)

        x = self.tok_embed(input_ids) + self.pos_embed(pos)
        x = self.drop(self.ln(x))

        src_key_padding_mask = (attn_mask == 0)

        h = self.encoder(x, src_key_padding_mask=src_key_padding_mask)

```

```
    logits = self.mlm_head(h)
    return logits
```

In [28]:

```
hidden = 256
model = MiniBERT(len(word2idx), hidden=hidden, max_len=max_len).to(device)
optimizer = torch.optim.AdamW(model.parameters(), lr=3e-4)
loss_fn = nn.CrossEntropyLoss(ignore_index=-100)
```

- **Training**

In [29]:

```
batch_size = 16
epochs = 5
train_texts = texts[:100000]

def batch_iter(data, bs):
    for i in range(0, len(data), bs):
        yield data[i:i+bs]

for ep in range(epochs):
    model.train()
    total_loss = 0.0
    steps = 0

    for batch_texts in tqdm(batch_iter(train_texts, batch_size), total=len(train_texts) / batch_size):
        batch_inp, batch_lbl, batch_attn = [], [], []
        for t in batch_texts:
            inp, lbl, attn = make_sample(t)
            batch_inp.append(inp)
            batch_lbl.append(lbl)
            batch_attn.append(attn)

        input_ids = torch.tensor(batch_inp, dtype=torch.long).to(device)
        labels = torch.tensor(batch_lbl, dtype=torch.long).to(device)
        attn_mask = torch.tensor(batch_attn, dtype=torch.long).to(device)

        logits = model(input_ids, attn_mask)
        loss = loss_fn(logits.view(-1, logits.size(-1)), labels.view(-1))

        optimizer.zero_grad()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()

        total_loss += loss.item()
        steps += 1

    print(f"Epoch {ep+1}: avg loss = {total_loss/steps:.4f}")
```

1327it [01:22, 16.17it/s]

Epoch 1: avg loss = 6.8058

1327it [01:19, 16.65it/s]

Epoch 2: avg loss = 6.4891

1327it [01:21, 16.37it/s]

Epoch 3: avg loss = 6.3583

1327it [01:20, 16.54it/s]

Epoch 4: avg loss = 6.2571

1327it [01:19, 16.64it/s]

Epoch 5: avg loss = 6.1693

- **Save Model**

In [30]:

```
SAVE_PATH = ARTIFACTS + "/BERT_scratch.pt"

model_cpu = model.to("cpu")
torch.save({
    "model_state": model_cpu.state_dict(),
    "word2idx": word2idx,
    "config": {
        "hidden": hidden,
        "max_len": max_len,
        "n_layers": 4,
        "n_heads": 4,
        "vocab_size": len(word2idx),
        "special_tokens": special_tokens,
        "dataset": "wikitext-2-raw-v1 (subset)"}}, SAVE_PATH)

print("Saved:", SAVE_PATH)
```

Saved: /content/drive/MyDrive/Colab Notebooks/A4-do-you-agree/artifacts/BERT_scratch.pt

Training Result:

The MLM training loss decreased from 6.80 to 6.17 across 5 epochs, indicating successful learning of contextual language representations.

Task 2: Sentence Embedding with Sentence BERT

In [54]:

```
BASE = "/content/drive/MyDrive/Colab Notebooks/A4-do-you-agree"
ARTIFACTS = BASE + "/artifacts"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using:", device)

BERT_CKPT = ARTIFACTS + "/BERT_scratch.pt"
```

Using: cuda

- **Load BERT checkpoint**

In [33]:

```
ckpt = torch.load(BERT_CKPT, map_location="cpu")
word2idx = ckpt["word2idx"]
config = ckpt["config"]

PAD = word2idx["[PAD]"]
CLS = word2idx["[CLS]"]
SEP = word2idx["[SEP]"]
UNK = word2idx["[UNK]"]

max_len = config["max_len"]
hidden = config["hidden"]

print("Loaded vocab:", len(word2idx), "max_len:", max_len, "hidden:", hidden)
```

Loaded vocab: 20005 max_len: 128 hidden: 256

In [46]:

```
class MiniBERT_Encoder(nn.Module):
```

```

def __init__(self, vocab_size, hidden=256, max_len=128, n_layers=4, n_heads=4, dropout=0.1):
    super().__init__()
    self.hidden = hidden
    self.max_len = max_len

    self.tok_embed = nn.Embedding(vocab_size, hidden, padding_idx=PAD)
    self.pos_embed = nn.Embedding(max_len, hidden)
    self.ln = nn.LayerNorm(hidden)
    self.drop = nn.Dropout(dropout)

    enc_layer = nn.TransformerEncoderLayer(d_model=hidden, nhead=n_heads, dropout=dropout, batch_first=True, activation="gelu")
    self.encoder = nn.TransformerEncoder(enc_layer, num_layers=n_layers)

def forward(self, input_ids, attn_mask):
    B, L = input_ids.shape
    pos = torch.arange(L, device=input_ids.device).unsqueeze(0).expand(B, L)

    x = self.tok_embed(input_ids) + self.pos_embed(pos)
    x = self.drop(self.ln(x))

    src_key_padding_mask = (attn_mask == 0)
    h = self.encoder(x, src_key_padding_mask=src_key_padding_mask)
    return h

```

- **Load encoder weights**

In [47]:

```

encoder = MiniBERT_Encoder(vocab_size=len(word2idx), hidden=config["hidden"], max_len=config["max_len"],
                            n_layers=config.get("n_layers", 4), n_heads=config.get("n_heads", 4)).to(device)

missing, unexpected = encoder.load_state_dict(ckpt["model_state"], strict=False)

print("missing:", missing)
print("unexpected:", unexpected)

missing: []
unexpected: ['mlm_head.weight', 'mlm_head.bias']

```

- **Tokenizer**

In [50]:

```

def tokenize(text):
    return [word2idx.get(w, UNK) for w in text.lower().split()]

def encode_inputs(text):
    tokens = tokenize(text)[:max_len-2]
    input_ids = [CLS] + tokens + [SEP]
    attn_mask = [1]*len(input_ids)

    pad_len = max_len - len(input_ids)
    input_ids += [PAD]*pad_len
    attn_mask += [0]*pad_len

    return input_ids, attn_mask

```

- **Mean Pooling**

In [51]:

```

def mean_pool(token_embeds, attn_mask):
    mask = attn_mask.unsqueeze(-1).float()
    summed = (token_embeds * mask).sum(dim=1)
    counts = mask.sum(dim=1).clamp(min=1e-9)

```

```
return summed / counts
```

- **SBERT & classifier**

In [52]:

```
class SBERTSoftmax(nn.Module):  
    def __init__(self, encoder, hidden):  
        super().__init__()  
        self.encoder = encoder  
        self.classifier = nn.Linear(hidden*3, 3)  
  
    def forward(self, ids_a, mask_a, ids_b, mask_b):  
        h_a = self.encoder(ids_a, mask_a)  
        h_b = self.encoder(ids_b, mask_b)  
  
        u = mean_pool(h_a, mask_a)  
        v = mean_pool(h_b, mask_b)  
  
        feats = torch.cat([u, v, torch.abs(u - v)], dim=-1)  
        return self.classifier(feats)
```

In [53]:

```
hidden = config["hidden"]  
sbert = SBERTSoftmax(encoder, hidden).to(device)
```

- **Training**

In [57]:

```
snli = load_dataset("snli")  
  
train_data = snli["train"].filter(lambda x: x["label"] != -1)  
val_data = snli["validation"].filter(lambda x: x["label"] != -1)  
train_data = train_data.select(range(min(50000, len(train_data))))  
val_data = val_data.select(range(min(10000, len(val_data))))  
  
loss_fn = nn.CrossEntropyLoss()  
opt = torch.optim.AdamW(sbert.parameters(), lr=2e-4)
```

In [58]:

```
def batch_encode_inputs(text_list):  
    B = len(text_list)  
    ids = torch.full((B, max_len), PAD, dtype=torch.long)  
    mask = torch.zeros((B, max_len), dtype=torch.long)  
  
    for i, text in enumerate(text_list):  
        toks = [word2idx.get(w, UNK) for w in text.lower().split()[:max_len-2]]  
        seq = [CLS] + toks + [SEP]  
        L = len(seq)  
        ids[i, :L] = torch.tensor(seq, dtype=torch.long)  
        mask[i, :L] = 1  
  
    return ids, mask  
  
def iterate_minibatches(ds, bs, shuffle=True):  
    idxs = list(range(len(ds)))  
    if shuffle:  
        random.shuffle(idxs)  
    for i in range(0, len(ds), bs):  
        yield [ds[j] for j in idxs[i:i+bs]]
```

In [60]:

```
batch_size = 32  
epochs = 5
```

```

for ep in range(epochs):
    sbert.train()
    total_loss, steps = 0.0, 0

    for batch in tqdm(iterate_minibatches(train_data, batch_size), total=len(train_data) //batch_size):
        prem = [b["premise"] for b in batch]
        hypo = [b["hypothesis"] for b in batch]
        y = torch.tensor([b["label"] for b in batch], dtype=torch.long).to(device)

        ids_a, mask_a = batch_encode_inputs(prem)
        ids_b, mask_b = batch_encode_inputs(hypo)

        ids_a, mask_a = ids_a.to(device), mask_a.to(device)
        ids_b, mask_b = ids_b.to(device), mask_b.to(device)

        logits = sbert(ids_a, mask_a, ids_b, mask_b)
        loss = loss_fn(logits, y)

        opt.zero_grad()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(sbert.parameters(), 1.0)
        opt.step()

        total_loss += loss.item()
        steps += 1

    print(f"Epoch {ep+1}: avg loss = {total_loss/steps:.4f}")

```

1563it [03:12, 8.13it/s]

Epoch 1: avg loss = 0.9353

1563it [03:12, 8.14it/s]

Epoch 2: avg loss = 0.8820

1563it [03:22, 7.73it/s]

Epoch 3: avg loss = 0.8432

1563it [03:12, 8.11it/s]

Epoch 4: avg loss = 0.8078

1563it [03:10, 8.21it/s]

Epoch 5: avg loss = 0.7724

Task 3: Evaluation and Analysis

- Accuracy & Cosine similarity

In [67]:

```

def eval_accuracy(sbert, val_data, batch_size=32, n_samples=2000, device="cuda"):
    sbert.eval()
    correct, total = 0, 0

    with torch.no_grad():
        val_small = val_data.select(range(min(n_samples, len(val_data)))))

        for batch in tqdm(iterate_minibatches(val_small, batch_size, shuffle=False),
                           total=math.ceil(len(val_small)/batch_size)):
            prem = [b["premise"] for b in batch]
            hypo = [b["hypothesis"] for b in batch]
            y = torch.tensor([b["label"] for b in batch], dtype=torch.long).to(device)

            ids_a, mask_a = batch_encode_inputs(prem)

```

```

        ids_b, mask_b = batch_encode_inputs(hypo)

        ids_a, mask_a = ids_a.to(device), mask_a.to(device)
        ids_b, mask_b = ids_b.to(device), mask_b.to(device)

        logits = sbert(ids_a, mask_a, ids_b, mask_b)
        pred = torch.argmax(logits, dim=-1)

        correct += (pred == y).sum().item()
        total += y.size(0)

    return correct / max(total, 1)

val_acc = eval_accuracy(sbert, val_data, batch_size=32, n_samples=2000, device=device)
print("Val accuracy (2k subset):", val_acc)

0%|          | 0/63 [00:00<?, ?it/s]/usr/local/lib/python3.12/dist-packages/torch/nn/modules/transformer.py:515: UserWarning: The PyTorch API of nested tensors is in prototype stage and will change in the near future. We recommend specifying layout=torch.jagged when constructing a nested tensor, as this layout receives active development, has better operator coverage, and works with torch.compile. (Triggered internally at /pytorch/aten/src/ATen/NestedTensorImpl.cpp:178.)
    output = torch._nested_tensor_from_mask(
100%|██████████| 63/63 [00:00<00:00, 63.29it/s]

```

Val accuracy (2k subset): 0.6165

In [68]:

```

sbert.eval()
with torch.no_grad():
    sentA = "A man is playing guitar."
    sentB = "A person is performing music."

    ids_a, mask_a = batch_encode_inputs([sentA])
    ids_b, mask_b = batch_encode_inputs([sentB])

    ids_a, mask_a = ids_a.to(device), mask_a.to(device)
    ids_b, mask_b = ids_b.to(device), mask_b.to(device)

    h_a = sbert.encoder(ids_a, mask_a)
    h_b = sbert.encoder(ids_b, mask_b)

    u = mean_pool(h_a, mask_a)
    v = mean_pool(h_b, mask_b)

    cos = F.cosine_similarity(u, v).item()
    print("Cosine similarity:", cos)

```

Cosine similarity: 0.4799080193042755

In [69]:

```

def eval_report(sbert, val_data, batch_size=32, n_samples=2000, device="cuda"):
    sbert.eval()
    y_true, y_pred = [], []

    with torch.no_grad():
        val_small = val_data.select(range(min(n_samples, len(val_data)))))

        for batch in tqdm(iterate_minibatches(val_small, batch_size, shuffle=False),
                           total=math.ceil(len(val_small)/batch_size)):
            prem = [b["premise"] for b in batch]
            hypo = [b["hypothesis"] for b in batch]
            y = [b["label"] for b in batch]

            ids_a, mask_a = batch_encode_inputs(prem)
            ids_b, mask_b = batch_encode_inputs(hypo)

            ids_a, mask_a = ids_a.to(device), mask_a.to(device)
            ids_b, mask_b = ids_b.to(device), mask_b.to(device)

```

```

logits = sbert(ids_a, mask_a, ids_b, mask_b)
pred = torch.argmax(logits, dim=-1).cpu().numpy().tolist()

y_true.extend(y)
y_pred.extend(pred)

acc = (np.array(y_true) == np.array(y_pred)).mean()

print("Accuracy:", acc)
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=["entailment", "neutral", "contradiction"], digits=4))
print("\nConfusion Matrix:")
print(confusion_matrix(y_true, y_pred))

return acc

```

```
eval_report(sbert, val_data, batch_size=32, n_samples=2000, device=device)
```

```
100%|██████████| 63/63 [00:00<00:00, 69.43it/s]
```

Accuracy: 0.6165

Classification Report:

| | precision | recall | f1-score | support |
|---------------|-----------|--------|----------|---------|
| entailment | 0.6414 | 0.5988 | 0.6193 | 663 |
| neutral | 0.5989 | 0.6440 | 0.6206 | 677 |
| contradiction | 0.6126 | 0.6061 | 0.6093 | 660 |
| accuracy | | | 0.6165 | 2000 |
| macro avg | 0.6176 | 0.6163 | 0.6164 | 2000 |
| weighted avg | 0.6175 | 0.6165 | 0.6165 | 2000 |

Confusion Matrix:

```
[[397 145 121]
 [109 436 132]
 [113 147 400]]
```

Out[69]:

```
np.float64(0.6165)
```

- **Save model**

In [70]:

```

SAVE_SBERT = ARTIFACTS + "/SBERT_nli.pt"

sbert_cpu = sbert.to("cpu")
torch.save({
    "encoder_state": sbert_cpu.encoder.state_dict(),
    "classifier_state": sbert_cpu.classifier.state_dict(),
    "word2idx": word2idx,
    "config": config,
    "label2id": {"entailment":0, "neutral":1, "contradiction":2}, SAVE_SBERT}

print("Saved:", SAVE_SBERT)

```

```
Saved: /content/drive/MyDrive/Colab Notebooks/A4-do-you-agree/artifacts/SBERT_nli.pt
```

Discussion and Analysis

The experimental results demonstrate that the proposed SBERT model successfully learns meaningful sentence representations using a Siamese architecture and SoftmaxLoss objective.

Key observations:

- The model achieves balanced performance across all three NLI classes.
 - Mean pooling effectively aggregates token representations into sentence embeddings.
 - Training BERT from scratch still produces reasonable semantic understanding despite limited data and computational resources.
 - The cosine similarity experiment confirms that embeddings encode semantic similarity beyond classification labels.
-

Limitations

Several limitations were observed during implementation:

- Training BERT from scratch requires substantial computational resources and longer training time.
 - The model was trained on a subset of SNLI rather than the full dataset due to GPU constraints.
 - Simple whitespace tokenization limits linguistic understanding compared to subword tokenizers (e.g., WordPiece).
-

Potential Improvements

Future improvements may include:

- Training on larger datasets such as MNLI or full SNLI.
- Using subword tokenization (WordPiece/BPE).
- Increasing model depth or training epochs.
- Applying contrastive learning objectives for stronger semantic embeddings.
- Fine-tuning with pretrained transformer weights.

Task 4: Text similarity - Web Application Development

A Dash-based web application was developed to demonstrate sentence similarity and Natural Language Inference (NLI).

Features:

- Input two sentences
- Generate SBERT embeddings
- Compute cosine similarity
- Predict NLI label (entailment / neutral / contradiction)
- Display probability distribution

The application loads the trained SBERT model from Task 2 and performs real-time inference.