model_03a_bert_base_cos_sim_LR_insight

May 3, 2023

1 Model 03a

Evidence retrieval using a Siamese BERT classification model. This is similar to Model 01, however, it only uses official pre-trained models from hugging face.

Ref: - Hugging face pre-trained models - Hugging face guide to fine-tuning - Hugging face guide to fine-tuning easy - SO Guide

1.1 Setup

1.1.1 Working Directory

[ ]:
```
# Change the working directory to project root
from pathlib import Path
import os
ROOT_DIR = Path.cwd()
while not ROOT_DIR.joinpath("src").exists():
    ROOT_DIR = ROOT_DIR.parent
os.chdir(ROOT_DIR)
```

1.1.2 File paths

[ ]:
```
MODEL_PATH = ROOT_DIR.joinpath("./result/models/*")
DATA_PATH = ROOT_DIR.joinpath("./data/*")
NER_PATH = ROOT_DIR.joinpath("./result/ner/*")
```

1.1.3 Dependencies

[ ]:
```
# Imports and dependencies
import torch
from torch.utils.data import Dataset, DataLoader
from torch.nn import Module, CosineEmbeddingLoss
from transformers import BertModel, BertTokenizer
from torch.optim import Adam
from torch.optim.lr_scheduler import LinearLR
from torcheval.metrics import BinaryAccuracy, BinaryF1Score

from src.torch_utils import get_torch_device
```

```
import json
from dataclasses import dataclass
from typing import List, Union, Tuple
from tqdm import tqdm
import random
import numpy as np
from datetime import datetime
from math import exp

TORCH_DEVICE = get_torch_device()
```

```
/opt/homebrew/Caskroom/miniconda/base/envs/comp90042_project/lib/python3.8/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
Torch device is 'mps'
```

1.2 Dataset

[ ]:
```
@dataclass
class ClaimEvidencePair:
    claim_id:str
    evidence_id:str
    label:int = 0
```

[ ]:
```
class SiameseEvalDataset(Dataset):
    def __init__(
        self,
        dev_claims_path:Path,
        evidence_path:Path,
        device = None,
        verbose:bool=True
    ) -> None:
        super(SiameseEvalDataset, self).__init__()
        self.verbose = verbose
        self.device = device

        # Load claims data from json
        with open(dev_claims_path, mode="r") as f:
            self.claims = (json.load(fp=f))

        # Load evidence library
        self.evidence = dict()
        with open(evidence_path, mode="r") as f:
            self.evidence.update(json.load(fp=f))
```

```
        # Get a list of all evidences within the dev set
        self.related_evidences = sorted({
            evidence_id
            for claim in self.claims.values()
            for evidence_id in claim["evidences"]
        })

        # Generate the data
        self.data = self.__generate_data()
        return

    def __generate_data(self):
        data = []
        for claim_id, claim in tqdm(
            iterable=self.claims.items(),
            desc="claims",
            disable=not self.verbose
        ):
            evidence_ids = claim["evidences"]

            # Get the positives
            for evidence_id in evidence_ids:
                data.append(ClaimEvidencePair(
                    claim_id=claim_id,
                    evidence_id=evidence_id,
                    label=1
                ))

            # Get some negatives
            n_neg = 0
            for rel_evidence_id in self.related_evidences:
                if n_neg >= 10:
                    break
                if rel_evidence_id in evidence_ids:
                    continue
                data.append(ClaimEvidencePair(
                    claim_id=claim_id,
                    evidence_id=rel_evidence_id,
                    label=-1
                ))
                n_neg += 1
        return data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx) -> Tuple[Union[str, torch.Tensor]]:
```

```
        # Fetch the required data rows
        data = self.data[idx]

        # Get the label
        label = torch.tensor(data.label, device=self.device)

        # Get text ids
        claim_id = data.claim_id
        evidence_id = data.evidence_id

        # Get text
        claim_text = self.claims[claim_id]["claim_text"]
        evidence_text = self.evidence[evidence_id]

        return (claim_text, evidence_text, label)
```

[ ]:
```
class SiameseDataset(Dataset):

    def __init__(
        self,
        claims_paths:List[Path],
        claims_shortlist_paths:List[Path],
        evidence_path:Path,
        evidence_shortlists:List[Path] = None,
        device = None,
        n_neg_shortlist:int = 10,
        n_neg_general:int = 10,
        verbose:bool=True
    ) -> None:
        super(SiameseDataset, self).__init__()
        self.verbose = verbose
        self.device = device
        self.n_neg_shortlist = n_neg_shortlist
        self.n_neg_general = n_neg_general

        # Load claims data from json, this is a list as we could use
        # multiple json files in the same dataset
        self.claims = dict()
        for json_file in claims_paths:
            with open(json_file, mode="r") as f:
                self.claims.update(json.load(fp=f))
                # print(f"loaded claims: {json_file}")

        # Load the pre-retrieved shortlist of evidences by claim
        self.claims_shortlist = dict()
        for json_file in claims_shortlist_paths:
            with open(json_file, mode="r") as f:
```

```
                self.claims_shortlist.update(json.load(fp=f))
                # print(f"loaded claims_shortlist: {json_file}")

        # Load evidence library
        self.evidence = dict()
        with open(evidence_path, mode="r") as f:
            self.evidence.update(json.load(fp=f))
            # print(f"loaded evidences: {json_file}")

        # Load the evidence shortlists if available
        # Reduce the overall evidence list to the shortlist
        if evidence_shortlists is not None:
            self.evidence_shortlist = set()
            for json_file in evidence_shortlists:
                with open(json_file, mode="r") as f:
                    self.evidence_shortlist.update(json.load(fp=f))
                    # print(f"loaded evidence shortlist: {json_file}")

        # print(f"n_evidences: {len(self.evidence)}")

        # Generate the data
        self.data = self.__generate_data()
        return

    def __generate_data(self):
        print("Generate siamese dataset")

        data = []
        for claim_id, claim in tqdm(
            iterable=self.claims.items(),
            desc="claims",
            disable=not self.verbose
        ):
            # Check if we have evidences supplied, this will inform
            # whether this is for training
            is_training = "evidences" in claim.keys()
            pos_evidence_ids = set()

            # Get positive samples from evidences with label=1
            if is_training:
                pos_evidence_ids.update(claim["evidences"])

                for evidence_id in pos_evidence_ids:
                    data.append(ClaimEvidencePair(
                        claim_id=claim_id,
                        evidence_id=evidence_id,
                        label=1
```

```
                ))

            # Get negative samples from pre-retrieved evidences
            # for each claim with label=-1
            retrieved_evidence_ids = self.claims_shortlist.get(claim_id, [])
            if len(retrieved_evidence_ids) > 0:
                retrieved_neg_evidence_ids = random.sample(
                    population=retrieved_evidence_ids,
                    k=min(self.n_neg_shortlist, len(retrieved_evidence_ids))
                )

                # Generate claim and shortlisted negative evidence pairs
                for evidence_id in retrieved_neg_evidence_ids:
                    data.append(ClaimEvidencePair(
                        claim_id=claim_id,
                        evidence_id=evidence_id,
                        label=-1
                    ))

            # Get negative samples from shortlisted evidences list with label=0
            if len(self.evidence_shortlist) > 0:
                shortlist_neg_evidence_ids = random.sample(
                    population=self.evidence_shortlist,
                    k=min(self.n_neg_general, len(self.evidence_shortlist))
                )

                # Generate claim and shortlisted negative evidence pairs
                for evidence_id in shortlist_neg_evidence_ids:
                    data.append(ClaimEvidencePair(
                        claim_id=claim_id,
                        evidence_id=evidence_id,
                        label=-1
                    ))

            continue

        print(f"Generated data n={len(data)}")

        return data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx) -> Tuple[Union[str, torch.Tensor]]:
        # Fetch the required data rows
        data = self.data[idx]
```

```
        # Get the label
        label = torch.tensor(data.label, device=self.device)

        # Get text ids
        claim_id = data.claim_id
        evidence_id = data.evidence_id

        # Get text
        claim_text = self.claims[claim_id]["claim_text"]
        evidence_text = self.evidence[evidence_id]

        return (claim_text, evidence_text, label)
```

[ ]:
```
# WE WILL GENERATE THE DATASET PER EPOCH SO TO RANDOMISE THE NEGATIVE SAMPLES

# train_data = SiameseDataset(
#     claims_paths=[DATA_PATH.with_name("train-claims.json")],
#     claims_shortlist_paths=[NER_PATH.
 ↪with_name("train_claim_evidence_retrieved.json")],
#     evidence_shortlists=[NER_PATH.
 ↪with_name("shortlist_train_claim_evidence_retrieved.json")],
#     evidence_path=DATA_PATH.with_name("evidence.json"),
#     device=TORCH_DEVICE,
#     n_neg_shortlist=100,
#     n_neg_general=100
# )
```

1.3 Build model

[ ]:
```
class SiameseEmbedderBert(Module):

    def __init__(
            self,
            pretrained_name:str,
            device,
            **kwargs
        ) -> None:
        super(SiameseEmbedderBert, self).__init__(**kwargs)
        self.device = device

        # Use a pretrained tokenizer
        self.tokenizer = BertTokenizer.from_pretrained(pretrained_name)

        # Use a pretrained model
        self.bert = BertModel.from_pretrained(pretrained_name)
        self.bert.to(device=device)
        return
```

```
    def forward(self, claim_texts, evidence_texts, eval_mode:bool=False) ->␣
↪Tuple[torch.Tensor]:

        # Run the tokenizer
        t_kwargs = {
            "return_tensors": "pt",
            "padding": True,
            "truncation": True,
            "max_length": 100,
            "add_special_tokens":True
        }
        claim_x = self.tokenizer(claim_texts, **t_kwargs)
        evidence_x = self.tokenizer(evidence_texts, **t_kwargs)

        claim_x = claim_x["input_ids"].to(device=self.device)
        evidence_x = evidence_x["input_ids"].to(device=self.device)

        # Run Bert
        claim_x = self.bert(claim_x, return_dict=True).pooler_output
        evidence_x = self.bert(evidence_x, return_dict=True).pooler_output
        # dim=768

        # Cosine similarity
        if eval_mode:
            cos_sim = torch.cosine_similarity(x1=claim_x, x2=evidence_x)
            return claim_x, evidence_x, cos_sim

        return claim_x, evidence_x
```

1.4 Training and evaluation loop

[ ]:
```
model = SiameseEmbedderBert(
    pretrained_name="bert-base-cased",
    device=TORCH_DEVICE
)
```

```
Some weights of the model checkpoint at bert-base-cased were not used when
initializing BertModel: ['cls.predictions.decoder.weight',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.seq_relationship.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of
```

a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

[ ]:
```
loss_fn = CosineEmbeddingLoss()
optimizer = Adam(
    params=model.parameters(),
    lr=0.000002
) #! Hyperparams
```

[ ]:
```
run_time = datetime.now().strftime('%Y_%m_%d_%H_%M')
MODEL_NAME = f"model_03a_bert_base_cos_sim_{run_time}.pth"
N_EPOCHS = 100
BATCH_SIZE = 64
```

[ ]:
```
dev_data = SiameseEvalDataset(
    dev_claims_path=DATA_PATH.with_name("dev-claims.json"),
    evidence_path=DATA_PATH.with_name("evidence.json"),
    device=TORCH_DEVICE
)

dev_dataloader = DataLoader(
    dataset=dev_data,
    shuffle=False,
    batch_size=BATCH_SIZE
)
```

```
claims: 100%|      | 154/154 [00:00<00:00, 179472.86it/s]
claims: 100%|      | 154/154 [00:00<00:00, 179472.86it/s]
```

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
# Run evaluation before training to establish baseline
model.eval()

dev_batches = tqdm(dev_dataloader, desc="dev batches")
epoch_pos_cos_sim = []
epoch_neg_cos_sim = []
for batch in dev_batches:
    claim_texts, evidence_texts, labels = batch

    # Forward
    claim_emb, evidence_emb, cos_sim = model(claim_texts, evidence_texts,␣
 ↪eval_mode=True)

    # Cosine similarity
    labelled_cos_sim = cos_sim * labels
```

```
    pos_cos_sim = labelled_cos_sim[torch.where(labelled_cos_sim > 0)]
    neg_cos_sim = labelled_cos_sim[torch.where(labelled_cos_sim < 0)]

    batch_pos_cos_sim = torch.mean(pos_cos_sim).cpu().item()
    batch_neg_cos_sim = torch.mean(neg_cos_sim).cpu().item() * -1

    epoch_pos_cos_sim.append(batch_pos_cos_sim)
    epoch_neg_cos_sim.append(batch_neg_cos_sim)

    dev_batches.postfix = f"pos cos_sim: {batch_pos_cos_sim:.3f}" + \
        f" neg cos_sim: {batch_neg_cos_sim:.3f}"

    continue
```

```
dev batches: 100%|      | 32/32 [00:09<00:00,  3.51it/s, pos cos_sim: 0.972
neg cos_sim: 0.868]
```

[ ]:
```
print(f"Average cos sim (pos, neg): {np.mean(epoch_pos_cos_sim):3f}, {np.
↪mean(epoch_neg_cos_sim):3f}")
```

```
Average cos sim (pos, neg): 0.863419, 0.838396
```

[ ]:
```
metric_accuracy = BinaryAccuracy()
metric_f1 = BinaryF1Score()
metric_recall = BinaryF1Score()

scheduler = LinearLR(
    optimizer=optimizer,
    start_factor=0.1,
    end_factor=1,
    total_iters=int(N_EPOCHS/10),
    verbose=True
)
best_epoch_loss = 999
for epoch in range(N_EPOCHS):

    print(f"Epoch: {epoch} of {N_EPOCHS}\n")

    # Run training
    model.train()

    train_data = SiameseDataset(
        claims_paths=[DATA_PATH.with_name("train-claims.json")],
        claims_shortlist_paths=[NER_PATH.
↪with_name("train_claim_evidence_retrieved.json")],
        evidence_shortlists=[NER_PATH.
↪with_name("shortlist_train_claim_evidence_retrieved.json")],
```

```
        evidence_path=DATA_PATH.with_name("evidence.json"),
        device=TORCH_DEVICE,
        n_neg_shortlist=2,
        n_neg_general=1
    )

    train_dataloader = DataLoader(
        dataset=train_data,
        shuffle=True,
        batch_size=BATCH_SIZE
    )

    train_batches = tqdm(train_dataloader, desc="train batches")
    running_losses = []
    for batch in train_batches:
        claim_texts, evidence_texts, labels = batch

        # Reset optimizer
        optimizer.zero_grad()

        # Forward + loss
        claim_emb, evidence_emb = model(claim_texts, evidence_texts)
        loss = loss_fn(input1=claim_emb, input2=evidence_emb, target=labels)

        # Backward + optimiser
        loss.backward()
        optimizer.step()

        # Update running loss
        batch_loss = loss.item() * len(batch)
        running_losses.append(batch_loss)

        train_batches.postfix = f"loss: {batch_loss:.3f}"

        continue

    scheduler.step()

    epoch_loss = np.average(running_losses)
    print(f"Average epoch loss: {epoch_loss}")

    # Save model
    if epoch_loss <= best_epoch_loss:
        best_epoch_loss = epoch_loss
        torch.save(model, MODEL_PATH.with_name(MODEL_NAME))
        print(f"Saved model to: {MODEL_PATH.with_name(MODEL_NAME)}")
```

```
    # Evaluate every 5 epochs
    # if epoch % 5 != 0:
    #     continue

    # Run evaluation before training to establish baseline
    model.eval()

    dev_batches = tqdm(dev_dataloader, desc="dev batches")
    epoch_pos_cos_sim = []
    epoch_neg_cos_sim = []
    for batch in dev_batches:
        claim_texts, evidence_texts, labels = batch

        # Forward
        claim_emb, evidence_emb, cos_sim = model(claim_texts, evidence_texts,␣
↪eval_mode=True)

        # Cosine similarity
        labelled_cos_sim = cos_sim * labels
        pos_cos_sim = labelled_cos_sim[torch.where(labelled_cos_sim > 0)]
        neg_cos_sim = labelled_cos_sim[torch.where(labelled_cos_sim < 0)]

        batch_pos_cos_sim = torch.mean(pos_cos_sim).cpu().item()
        batch_neg_cos_sim = torch.mean(neg_cos_sim).cpu().item() * -1

        epoch_pos_cos_sim.append(batch_pos_cos_sim)
        epoch_neg_cos_sim.append(batch_neg_cos_sim)

        dev_batches.postfix = f"pos cos_sim: {batch_pos_cos_sim:.3f}" + \
            f" neg cos_sim: {batch_neg_cos_sim:.3f}"

        continue

    print(f"Average cos sim (pos, neg): {np.mean(epoch_pos_cos_sim):3f}, {np.
↪mean(epoch_neg_cos_sim):3f}")

print("Done!")
```

```
Adjusting learning rate of group 0 to 2.0000e-07.
Epoch: 0 of 100

Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 157.84it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:48<00:00,  1.11it/s, loss: 1.960]
```

```
Adjusting learning rate of group 0 to 3.8000e-07.
Average epoch loss: 1.413777578221865
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.25it/s, pos cos_sim: 0.970
neg cos_sim: 0.888]

Average cos sim (pos, neg): 0.879555, 0.861016
Epoch: 1 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 155.89it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:46<00:00,  1.14it/s, loss: 1.451]

Adjusting learning rate of group 0 to 5.6000e-07.
Average epoch loss: 1.3904771886088632
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.29it/s, pos cos_sim: 0.947
neg cos_sim: 0.851]

Average cos sim (pos, neg): 0.837746, 0.812627
Epoch: 2 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 162.05it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:46<00:00,  1.14it/s, loss: 1.118]

Adjusting learning rate of group 0 to 7.4000e-07.
Average epoch loss: 1.3746617729506216
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.29it/s, pos cos_sim: 0.910
neg cos_sim: 0.819]

Average cos sim (pos, neg): 0.825345, 0.786491
Epoch: 3 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 168.03it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:46<00:00,  1.14it/s, loss: 0.821]
```

```
Adjusting learning rate of group 0 to 9.2000e-07.
Average epoch loss: 1.3272475949988878
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.29it/s, pos cos_sim: 0.896
neg cos_sim: 0.878]

Average cos sim (pos, neg): 0.865243, 0.848583
Epoch: 4 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 172.60it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:45<00:00,  1.14it/s, loss: 0.392]

Adjusting learning rate of group 0 to 1.1000e-06.
Average epoch loss: 0.9673865113622886
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.28it/s, pos cos_sim: 0.926
neg cos_sim: 0.842]

Average cos sim (pos, neg): 0.835551, 0.822032
Epoch: 5 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 159.28it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:46<00:00,  1.14it/s, loss: 1.265]

Adjusting learning rate of group 0 to 1.2800e-06.
Average epoch loss: 0.7218735043174964
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.30it/s, pos cos_sim: 0.942
neg cos_sim: 0.882]

Average cos sim (pos, neg): 0.901280, 0.867277
Epoch: 6 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 173.51it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:46<00:00,  1.14it/s, loss: 0.648]
```

```
Adjusting learning rate of group 0 to 1.4600e-06.
Average epoch loss: 0.6569000715928629
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.29it/s, pos cos_sim: 0.924
neg cos_sim: 0.867]

Average cos sim (pos, neg): 0.888637, 0.858377
Epoch: 7 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 164.89it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:45<00:00,  1.15it/s, loss: 1.187]

Adjusting learning rate of group 0 to 1.6400e-06.
Average epoch loss: 0.6235487781280329
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.25it/s, pos cos_sim: 0.832
neg cos_sim: 0.840]

Average cos sim (pos, neg): 0.823294, 0.844328
Epoch: 8 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 165.54it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:46<00:00,  1.14it/s, loss: 0.087]

Adjusting learning rate of group 0 to 1.8200e-06.
Average epoch loss: 0.596648235074129
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.27it/s, pos cos_sim: 0.939
neg cos_sim: 0.837]

Average cos sim (pos, neg): 0.903299, 0.826131
Epoch: 9 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 166.75it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:45<00:00,  1.14it/s, loss: 0.613]
```

```
Adjusting learning rate of group 0 to 2.0000e-06.
Average epoch loss: 0.587202386730466
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.24it/s, pos cos_sim: 0.622
neg cos_sim: 0.891]

Average cos sim (pos, neg): 0.764762, 0.838932
Epoch: 10 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 165.99it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:45<00:00,  1.15it/s, loss: 0.801]

Adjusting learning rate of group 0 to 2.0000e-06.
Average epoch loss: 0.5764014214154117
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.27it/s, pos cos_sim: 0.664
neg cos_sim: 0.896]

Average cos sim (pos, neg): 0.785200, 0.840454
Epoch: 11 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 163.86it/s]

Generated data n=7693

train batches: 100%|      | 121/121 [01:44<00:00,  1.16it/s, loss: 0.251]

Adjusting learning rate of group 0 to 2.0000e-06.
Average epoch loss: 0.53346385880689
Saved model to: /Users/johnsonzhou/git/comp90042-project/result/models/model_03a
_bert_base_cos_sim_2023_05_03_20_43.pth

dev batches: 100%|      | 32/32 [00:07<00:00,  4.32it/s, pos cos_sim: 0.635
neg cos_sim: 0.887]

Average cos sim (pos, neg): 0.762964, 0.835725
Epoch: 12 of 100


Generate siamese dataset

claims: 100%|      | 1228/1228 [00:07<00:00, 174.40it/s]

Generated data n=7693

train batches:  14%|         | 17/121 [00:15<01:26,  1.20it/s, loss: 0.528]
```