Natural Language Inference With BERT

For this homework, we will work on (NLI) [https://nlp.stanford.edu/projects/snli/].

The task is, give two sentences: a premise and a hypothesis, to classify the relation between them. We have three classes to describe this relationship.

1. Entailment: the hypothesis follows from the fact that the premise is true
2. Contradiction: the hypothesis contradicts the fact that the premise is true
3. Neutral: There is not relationship between premise and hypothesis

See below for examples

Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

Prereqs

```
! pip install transformers datasets tqdm

Collecting transformers
  Downloading transformers-4.35.2-py3-none-any.whl (7.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.9/7.9 MB 58.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 521.2/521.2 kB 55.6 MB/s eta
0:00:00
ent already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(4.66.1)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from transformers) (3.13.1)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.19.4-py3-none-any.whl (311 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.7/311.7 kB 21.5 MB/s eta
0:00:00
ent already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers) (1.23.5)
```

```
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Collecting tokenizers<0.19,>=0.14 (from transformers)
  Downloading tokenizers-0.15.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 63.2 MB/s eta
0:00:00
 transformers)
  Downloading safetensors-0.4.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 49.3 MB/s eta
0:00:00
ent already satisfied: pyarrow>=8.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (9.0.0)
Collecting pyarrow-hotfix (from datasets)
  Downloading pyarrow_hotfix-0.5-py3-none-any.whl (7.8 kB)
Collecting dill<0.3.8,>=0.3.0 (from datasets)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 9.3 MB/s eta
0:00:00
ent already satisfied: pandas in /usr/local/lib/python3.10/dist-
packages (from datasets) (1.5.3)
Requirement already satisfied: xxhash in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 15.1 MB/s eta
0:00:00
ent already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.8.6)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(3.3.2)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(6.0.4)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
```

```
(4.0.3)
Requirement already satisfied: yarl<2.0,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.3.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.16.4->transformers) (4.5.0)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2023.7.22)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2023.3.post1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1-
>pandas->datasets) (1.16.0)
Installing collected packages: safetensors, pyarrow-hotfix, dill,
multiprocess, huggingface-hub, tokenizers, transformers, datasets
Successfully installed datasets-2.15.0 dill-0.3.7 huggingface-hub-
0.19.4 multiprocess-0.70.15 pyarrow-hotfix-0.5 safetensors-0.4.0
tokenizers-0.15.0 transformers-4.35.2
```

```python
# Imports for most of the notebook
import torch
from transformers import BertModel
from transformers import AutoTokenizer
from typing import Dict, List
import random
from tqdm.autonotebook import tqdm

print(torch.cuda.is_available())
device = torch.device("cpu")
# TODO: Uncomment the below line if you see True in the print
statement
# device = torch.device("cuda:0")
```

```
True
```

First let's load the Stanford NLI dataset from the huggingface datasets hub using the datasets package

Explore the dataset!

```
from datasets import load_dataset
dataset = load_dataset("snli")
print("Split sizes (num_samples, num_labels):\n", dataset.shape)
print("\nExample:\n", dataset['train'][0])
```

{"model_id":"8e348e1cfd6940ef8f187c45ebb1c780","version_major":2,"version_minor":0}

{"model_id":"332be4ea083d4c91b1920ea1f4cb63b2","version_major":2,"version_minor":0}

{"model_id":"24b8c7ff9ba045ada6d46b2b890d43d1","version_major":2,"version_minor":0}

{"model_id":"0112d08df73448809504fc1d594e84f2","version_major":2,"version_minor":0}

{"model_id":"e5598bf220674b3ca06a8ad2bf10a429","version_major":2,"version_minor":0}

{"model_id":"d546899da9aa4d8f8685632865a45c49","version_major":2,"version_minor":0}

{"model_id":"ad1969fe17f4432ca7e067dd5b91f4e6","version_major":2,"version_minor":0}

```
Split sizes (num_samples, num_labels):
 {'test': (10000, 3), 'train': (550152, 3), 'validation': (10000, 3)}

Example:
 {'premise': 'A person on a horse jumps over a broken down airplane.',
'hypothesis': 'A person is training his horse for a competition.',
'label': 1}
```

Each example is a dictionary with the keys: (premise, hypothesis, label).

Data Fields

- premise: a string used to determine the truthfulness of the hypothesis
- hypothesis: a string that may be true, false, or whose truth conditions may not be knowable when compared to the premise
- label: an integer whose value may be either 0, indicating that the hypothesis entails the premise, 1, indicating that the premise and hypothesis neither entail nor contradict each other, or 2, indicating that the hypothesis contradicts the premise.

Create Train, Validation and Test sets

```
from datasets import load_dataset
from collections import defaultdict

def get_snli(train=10000, validation=1000, test=1000):
    snli = load_dataset('snli')
    train_dataset = get_even_datapoints(snli['train'], train)
    validation_dataset = get_even_datapoints(snli['validation'],
validation)
    test_dataset = get_even_datapoints(snli['test'], test)

    return train_dataset, validation_dataset, test_dataset

def get_even_datapoints(datapoints, n):
    random.seed(42)
    dp_by_label = defaultdict(list)
    for dp in tqdm(datapoints, desc='Reading Datapoints'):
        dp_by_label[dp['label']].append(dp)

    unique_labels = [0, 1, 2]

    split = n//len(unique_labels)

    result_datapoints = []

    for label in unique_labels:
        result_datapoints.extend(random.sample(dp_by_label[label],
split))

    return result_datapoints

train_dataset, validation_dataset, test_dataset = get_snli()
```

{"model_id":"6427a7ac03af4a75889754e9211a91e0","version_major":2,"version_minor":0}

{"model_id":"88f4c1870ac145be8823a224bd1cd258","version_major":2,"version_minor":0}

{"model_id":"08d77742d68a464293e7fde653771ddc","version_major":2,"version_minor":0}

```
## sub set stats
from collections import Counter

# num sample stats
print(len(train_dataset), len(validation_dataset), len(test_dataset))

# label distribution
print(Counter([t['label'] for t in train_dataset]))
```

```
print(Counter([t['label'] for t in validation_dataset]))
print(Counter([t['label'] for t in test_dataset]))

# We have a perfectly balanced dataset

9999 999 999
Counter({0: 3333, 1: 3333, 2: 3333})
Counter({0: 333, 1: 333, 2: 333})
Counter({0: 333, 1: 333, 2: 333})
```

We want a function to load samples from the huggingface dataset so that they can be batched and encoded for our model.

Now let's reimplement our tokenizer using the huggingface tokenizer.

Notice that our **call** method (the one called when we call an instance of our class) takes both a premise batch and a hypothesis batch.

The HuggingFace BERT tokenizer knows to join these with the special sentence seperator token between them. We let HuggingFace do most of the work here for making batches of tokenized and encoded sentences.

```
# Nothing to do for this class!

class BatchTokenizer:
    """Tokenizes and pads a batch of input sentences."""

    def __init__(self, model_name='prajjwal1/bert-small'):
        """Initializes the tokenizer

        Args:
            pad_symbol (Optional[str], optional): The symbol for a
pad. Defaults to "<P>".
        """
        self.hf_tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model_name = model_name

    def get_sep_token(self,):
        return self.hf_tokenizer.sep_token

    def __call__(self, prem_batch: List[str], hyp_batch: List[str]) ->
List[List[str]]:
        """Uses the huggingface tokenizer to tokenize and pad a batch.

        We return a dictionary of tensors per the huggingface model
specification.
```

```
        Args:
            batch (List[str]): A List of sentence strings

        Returns:
            Dict: The dictionary of token specifications provided by
HuggingFace
        """
        # The HF tokenizer will PAD for us, and additionally combine
        # The two sentences deimited by the [SEP] token.
        enc = self.hf_tokenizer(
            prem_batch,
            hyp_batch,
            padding=True,
            return_token_type_ids=False,
            return_tensors='pt'
        )

        return enc


# HERE IS AN EXAMPLE OF HOW TO USE THE BATCH TOKENIZER
tokenizer = BatchTokenizer()
x = tokenizer(*[["this is the first premise", "This is the second
premise"], ["This is first hypothesis", "This is the second
hypothesis"]])
print(x)
tokenizer.hf_tokenizer.batch_decode(x["input_ids"])
```

{"model_id":"eef836aa65e946ecb34025a859701c51","version_major":2,"vers
ion_minor":0}

{"model_id":"1728c96d68594d38b5550472f68fc36e","version_major":2,"vers
ion_minor":0}

```
{'input_ids': tensor([[  101,  2023,  2003,  1996,  2034, 18458,
102,  2023,  2003,  2034,
         10744,   102,     0],
        [  101,  2023,  2003,  1996,  2117, 18458,   102,  2023,
2003,  1996,
          2117, 10744,   102]]), 'attention_mask': tensor([[1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}

['[CLS] this is the first premise [SEP] this is first hypothesis [SEP]
[PAD]',
 '[CLS] this is the second premise [SEP] this is the second hypothesis
[SEP]']
```

We can batch the train, validation, and test data, and then run it through the tokenizer

```
def generate_pairwise_input(dataset: List[Dict]) -> (List[str],
List[str], List[int]):
    """
    TODO: group all premises and corresponding hypotheses and labels
of the datapoints
    a datapoint as seen earlier is a dict of premis, hypothesis and
label
    """
    premises = []
    hypothesis = []
    labels = []
    for x in dataset:
        premises.append(x['premise'])
        hypothesis.append(x['hypothesis'])
        labels.append(x['label'])

    return premises, hypothesis, labels

train_premises, train_hypotheses, train_labels =
generate_pairwise_input(train_dataset)
validation_premises, validation_hypotheses, validation_labels =
generate_pairwise_input(validation_dataset)
test_premises, test_hypotheses, test_labels =
generate_pairwise_input(test_dataset)

def chunk(lst, n):
    """Yield successive n-sized chunks from lst."""
    for i in range(0, len(lst), n):
        yield lst[i:i + n]

def chunk_multi(lst1, lst2, n):
    for i in range(0, len(lst1), n):
        yield lst1[i: i + n], lst2[i: i + n]

batch_size = 16

# Notice that since we use huggingface, we tokenize and
# encode in all at once!
tokenizer = BatchTokenizer()
train_input_batches = [b for b in chunk_multi(train_premises,
train_hypotheses, batch_size)]
# Tokenize + encode
train_input_batches = [tokenizer(*batch) for batch in
train_input_batches]
```

Let's batch the labels, ensuring we get them in the same order as the inputs

```
def encode_labels(labels: List[int]) -> torch.FloatTensor:
    """Turns the batch of labels into a tensor

    Args:
        labels (List[int]): List of all labels in the batch

    Returns:
        torch.FloatTensor: Tensor of all labels in the batch
    """
    return torch.LongTensor([int(l) for l in labels])


train_label_batches = [b for b in chunk(train_labels, batch_size)]
train_label_batches = [encode_labels(batch) for batch in
train_label_batches]
```

Now we implement the model. Notice the TODO and the optional TODO (read why you may want to do this one.)

```
class NLIClassifier(torch.nn.Module):
    def __init__(self, output_size: int, hidden_size: int,
model_name='prajjwal1/bert-small'):
        super().__init__()
        self.output_size = output_size
        self.hidden_size = hidden_size

        # Initialize BERT, which we use instead of a single embedding
layer.
        self.bert = BertModel.from_pretrained(model_name)

        # TODO [OPTIONAL]: Updating all BERT parameters can be slow
and memory intensive.
        # Freeze them if training is too slow. Notice that the
learning
        # rate should probably be smaller in this case.
        # Uncommenting out the below 2 lines means only our
classification layer will be updated.

        for param in self.bert.parameters():
            param.requires_grad = False

        self.bert_hidden_dimension = self.bert.config.hidden_size

        # TODO: Add an extra hidden layer in the classifier,
projecting
        #       from the BERT hidden dimension to hidden size. Hint:
torch.nn.Linear()
```

```
        self.hidden_layer =
torch.nn.Linear(self.bert_hidden_dimension, self.hidden_size)

        # TODO: Add a relu nonlinearity to be used in the forward
method
        #
https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

        self.relu = torch.nn.ReLU()

        self.classifier = torch.nn.Linear(self.hidden_size,
self.output_size)

        # change: dim = 1?
        self.log_softmax = torch.nn.LogSoftmax(dim=2)

    def encode_text(
        self,
        symbols: Dict
    ) -> torch.Tensor:
        """Encode the (batch of) sequence(s) of token symbols BERT.
            Then, get CLS represenation.

        Args:
            symbols (Dict): The Dict of token specifications provided
by the HuggingFace tokenizer

        Returns:
            torch.Tensor: CLS token embedding
        """
        # First we get the contextualized embedding for each input
symbol
        # We no longer need an LSTM, since BERT encodes context and
        # gives us a single vector describing the sequence in the form
of the [CLS] token.
        encoded_sequence = self.bert(**symbols)
        # TODO: Get the [CLS] token
        #      The BertModel output. See here:
https://huggingface.co/docs/transformers/model_doc/bert#transformers.B
ertModel
        #      and check the returns for the forward method.
        # We want to return a tensor of the form batch_size x 1 x
bert_hidden_dimension
        # print(encoded_sequence.last_hidden_state.shape)
        # Return only the first token's embedding from the
last_hidden_state. Hint: using list slices
        # raise NotImplementedError

        cls_embedding = encoded_sequence.last_hidden_state[:, 0, :]
```

```
        return cls_embedding

    def forward(
        self,
        symbols: Dict,
    ) -> torch.Tensor:
        """_summary_

        Args:
            symbols (Dict): The Dict of token specifications provided
by the HuggingFace tokenizer

        Returns:
            torch.Tensor: _description_
        """
        encoded_sents = self.encode_text(symbols)
        output = self.hidden_layer(encoded_sents)
        output = self.relu(output)
        output = self.classifier(output)
        output = output.log_softmax(dim = -1)
        return output

# For making predictions at test time
def predict(model: torch.nn.Module, sents: torch.Tensor) -> List:
    logits = model(sents)
    return list(torch.argmax(logits, dim=-1).squeeze().numpy())
```

Evaluation metrics: Macro F1

```
import numpy as np
from numpy import sum as t_sum
from numpy import logical_and


def precision(predicted_labels, true_labels, which_label=1):
    """
    Precision is True Positives / All Positives Predictions
    """
    pred_which = np.array([pred == which_label for pred in
predicted_labels])
    true_which = np.array([lab == which_label for lab in true_labels])
    denominator = t_sum(pred_which)
    if denominator:
        return t_sum(logical_and(pred_which, true_which))/denominator
    else:
        return 0.


def recall(predicted_labels, true_labels, which_label=1):
```

```
    """
    Recall is True Positives / All Positive Labels
    """
    pred_which = np.array([pred == which_label for pred in
predicted_labels])
    true_which = np.array([lab == which_label for lab in true_labels])
    denominator = t_sum(true_which)
    if denominator:
        return t_sum(logical_and(pred_which, true_which))/denominator
    else:
        return 0.


def f1_score(
    predicted_labels: List[int],
    true_labels: List[int],
    which_label: int
):
    """
    F1 score is the harmonic mean of precision and recall
    """
    P = precision(predicted_labels, true_labels,
which_label=which_label)
    R = recall(predicted_labels, true_labels, which_label=which_label)

    if P and R:
        return 2*P*R/(P+R)
    else:
        return 0.


def macro_f1(
    predicted_labels: List[int],
    true_labels: List[int],
    possible_labels: List[int],
    label_map=None
):
    converted_prediction = [label_map[int(x)] for x in
predicted_labels] if label_map else predicted_labels
    scores = [f1_score(converted_prediction, true_labels, l) for l in
possible_labels]
    # Macro, so we take the uniform avg.
    return sum(scores) / len(scores)
```

Training loop.

```
def training_loop(
    num_epochs,
    train_features,
    train_labels,
```

```
    dev_sents,
    dev_labels,
    optimizer,
    model,
):
    print("Training...")
    loss_func = torch.nn.NLLLoss()
    batches = list(zip(train_features, train_labels))
    random.shuffle(batches)
    for i in range(num_epochs):
        losses = []
        for features, labels in tqdm(batches):
            # Empty the dynamic computation graph
            optimizer.zero_grad()
            preds = model(features.to(device)).squeeze(1)
            loss = loss_func(preds, labels.to(device))
            # Backpropogate the loss through our model
            loss.backward()
            optimizer.step()
            losses.append(loss.item())

        print(f"epoch {i}, loss: {sum(losses)/len(losses)}")
        # Estimate the f1 score for the development set
        print("Evaluating dev...")
        all_preds = []
        all_labels = []
        for sents, labels in tqdm(zip(dev_sents, dev_labels),
total=len(dev_sents)):
            pred = predict(model, sents)
            all_preds.extend(pred)
            all_labels.extend(list(labels.cpu().numpy()))

        dev_f1 = macro_f1(all_preds, all_labels, possible_labels)
        print(f"Dev F1 {dev_f1}")

    # Return the trained model
    return model

# # You can increase epochs if need be
# epochs = 40

# # TODO: Find a good learning rate and hidden size
# LR = 0.001
# hidden_size = 128

# possible_labels = set(train_labels)
# model = NLIClassifier(output_size=len(possible_labels),
hidden_size=hidden_size)
# model.to(device)
# optimizer = torch.optim.AdamW(model.parameters(), LR)
```

```
# batch_tokenizer = BatchTokenizer()

# validation_input_batches = [b for b in
chunk_multi(validation_premises, validation_hypotheses, batch_size)]

# # Tokenize + encode
# validation_input_batches = [batch_tokenizer(*batch) for batch in
validation_input_batches]
# validation_batch_labels = [b for b in chunk(validation_labels,
batch_size)]
# validation_batch_labels = [encode_labels(batch) for batch in
validation_batch_labels]

# training_loop(
#     epochs,
#     train_input_batches,
#     train_label_batches,
#     validation_input_batches,
#     validation_batch_labels,
#     optimizer,
#     model,
# )

# TODO: Get a final macro F1 on the test set.
# You should be able to mimic what we did with the validaiton set.
# You can increase epochs if need be
epochs = 20

# TODO: Find a good learning rate and hidden size
LR = 0.0005
hidden_size = 64

possible_labels = set(train_labels)
model = NLIClassifier(output_size=len(possible_labels),
hidden_size=hidden_size)
model.to(device)
optimizer = torch.optim.AdamW(model.parameters(), LR)

batch_tokenizer = BatchTokenizer()

test_input_batches = [b for b in chunk_multi(test_premises,
test_hypotheses, batch_size)]

# Tokenize + encode
test_input_batches = [batch_tokenizer(*batch) for batch in
test_input_batches]
test_batch_labels = [b for b in chunk(test_labels, batch_size)]
test_batch_labels = [encode_labels(batch) for batch in
test_batch_labels]
```

```
training_loop(
    epochs,
    train_input_batches,
    train_label_batches,
    test_input_batches,
    test_batch_labels,
    optimizer,
    model,
)
```

{"model_id":"57e5322867ed4501a4dc7a5d172feb92","version_major":2,"version_minor":0}

```
Training...
```

{"model_id":"59a41988839d43b39421c04a0f3474a7","version_major":2,"version_minor":0}

```
epoch 0, loss: 1.085246192598343
Evaluating dev...
```

{"model_id":"29a04bc87049473c822a9bd713c1c2a2","version_major":2,"version_minor":0}

```
Dev F1 0.4137809016725819
```

{"model_id":"1dc81cd1db4a48a8bf44a6fb05adc1b5","version_major":2,"version_minor":0}

```
epoch 1, loss: 1.0427489953041076
Evaluating dev...
```

{"model_id":"0410390c835643389a56bab9cc47dc01","version_major":2,"version_minor":0}

```
Dev F1 0.4411942586793338
```

{"model_id":"0ee71d5e20a1423fb82b9a3f0c17251b","version_major":2,"version_minor":0}

```
epoch 2, loss: 1.0195589395999909
Evaluating dev...
```

{"model_id":"6cd391b95b1d4fa49d99ca6f579d8dcd","version_major":2,"version_minor":0}

```
Dev F1 0.46422387055839365
```

{"model_id":"a072779453154e22952413568afe67be","version_major":2,"version_minor":0}

```
epoch 3, loss: 1.0000359885692596
Evaluating dev...
```

{"model_id":"a00c4f5cafac4113849b3ec6fc6e0668","version_major":2,"version_minor":0}

```
Dev F1 0.4919540853293059
```

{"model_id":"ed6e07a266cf45dcaeaf9b7ac8e68cc9","version_major":2,"version_minor":0}

```
epoch 4, loss: 0.9808614409446716
Evaluating dev...
```

{"model_id":"b7701b442a6645b598a14db4daa40735","version_major":2,"version_minor":0}

```
Dev F1 0.45764545808868123
```

{"model_id":"94083e2ce5054323b95d9c54530501a5","version_major":2,"version_minor":0}

```
epoch 5, loss: 0.9656355168342591
Evaluating dev...
```

{"model_id":"d460fca9a742403dbce4f92589b9b1b5","version_major":2,"version_minor":0}

```
Dev F1 0.5085039412222204
```

{"model_id":"6596f44853d942699144db5424fd055c","version_major":2,"version_minor":0}

```
epoch 6, loss: 0.9486732285499573
Evaluating dev...
```

{"model_id":"070d5480f13c474cb7c58b7d291962fe","version_major":2,"version_minor":0}

```
Dev F1 0.504390183107782
```

{"model_id":"e7b3047c22ed43cd89ca6234c98a5c93","version_major":2,"version_minor":0}

```
epoch 7, loss: 0.9343814737319946
Evaluating dev...
```

{"model_id":"9a8bb1bad57e4b8c909baeced94800dd","version_major":2,"version_minor":0}

```
Dev F1 0.5043502159184752
```

{"model_id":"c898f2b6fbfe424ca59558530579e0c1","version_major":2,"version_minor":0}

```
epoch 8, loss: 0.9203002425193787
Evaluating dev...
```

{"model_id":"407339d62ef84bd2b780c84334b2e1d0","version_major":2,"version_minor":0}

```
Dev F1 0.510447589922895
```

{"model_id":"dab0890e1b644d4b92f246b3380ed8fe","version_major":2,"version_minor":0}

```
epoch 9, loss: 0.9058628752708435
Evaluating dev...
```

{"model_id":"fbc5f0e6f77a499cbc7b142bd2d52dd5","version_major":2,"version_minor":0}

```
Dev F1 0.50808652241317
```

{"model_id":"6428378f679140639b496f2eeeb66d2b","version_major":2,"version_minor":0}

```
epoch 10, loss: 0.8920252138614655
Evaluating dev...
```

{"model_id":"b303c147eea845ae90bd7e844c444df1","version_major":2,"version_minor":0}

```
Dev F1 0.5188608371823659
```

{"model_id":"765a208549144293bbad7af54eab3697","version_major":2,"version_minor":0}

```
epoch 11, loss: 0.8781673481464386
Evaluating dev...
```

{"model_id":"7040f048691645dea1966f597d64f44e","version_major":2,"version_minor":0}

```
Dev F1 0.5207383247835057
```

{"model_id":"573e9e9857fd43d58231a0f473eb075b","version_major":2,"version_minor":0}

```
epoch 12, loss: 0.8647987538337708
Evaluating dev...
```

{"model_id":"da0003215c974ec8ac878ab5764041be","version_major":2,"version_minor":0}

```
Dev F1 0.5271660388293249
```

{"model_id":"417526044c7047cd8853560f02204178","version_major":2,"version_minor":0}

```
epoch 13, loss: 0.8523064764022827
Evaluating dev...
```

{"model_id":"4be9167ce20e47d3863ef9bc786d39df","version_major":2,"version_minor":0}

```
Dev F1 0.527838238132157
```

{"model_id":"21e92ac1e05d474eae9acc12f30d0e94","version_major":2,"version_minor":0}

```
epoch 14, loss: 0.8387727097272873
Evaluating dev...
```

{"model_id":"bdae11a5718144b6a7c800f3bd4c1d11","version_major":2,"version_minor":0}

```
Dev F1 0.5326179957416123
```

{"model_id":"2a1c11cf966743de80799ce83e995864","version_major":2,"version_minor":0}

```
epoch 15, loss: 0.8261452343940735
Evaluating dev...
```

{"model_id":"c263b8a8533047279fbd863bcfc41de5","version_major":2,"version_minor":0}

```
Dev F1 0.5254608473577398
```

{"model_id":"05be380da9d349468ace1b90181a7fa1","version_major":2,"version_minor":0}

```
epoch 16, loss: 0.8125119399785995
Evaluating dev...
```

{"model_id":"06804e97ee2c4b9cbbc486415e1da4b8","version_major":2,"version_minor":0}

```
Dev F1 0.5269055022789523
```

{"model_id":"87ccd459f2424756a86bdc468ff9b8df","version_major":2,"version_minor":0}

```
epoch 17, loss: 0.7986942239761352
Evaluating dev...
```

```
{"model_id":"5214b0ba528b41ef92ea318ce31cd13f","version_major":2,"version_minor":0}
```

```
Dev F1 0.5263270683852762
```

```
{"model_id":"b02d35002b424963b28d7e010a83812a","version_major":2,"version_minor":0}
```

```
epoch 18, loss: 0.786133123922348
Evaluating dev...
```

```
{"model_id":"e17f16ea68054514b7e9a3b8c814e464","version_major":2,"version_minor":0}
```

```
Dev F1 0.5267568393504248
```

```
{"model_id":"c4b9439b1279457f82c07b23426aa839","version_major":2,"version_minor":0}
```

```
epoch 19, loss: 0.7732372853040695
Evaluating dev...
```

```
{"model_id":"7fc8419698bb4684878ce63f04bc9ee3","version_major":2,"version_minor":0}
```

```
Dev F1 0.5250253456769687
```

```
NLIClassifier(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 512, padding_idx=0)
      (position_embeddings): Embedding(512, 512)
      (token_type_embeddings): Embedding(2, 512)
      (LayerNorm): LayerNorm((512,), eps=1e-12,
elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-3): 4 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=512, out_features=512,
bias=True)
              (key): Linear(in_features=512, out_features=512,
bias=True)
              (value): Linear(in_features=512, out_features=512,
bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
```

```
              (dense): Linear(in_features=512, out_features=512, 
bias=True)
              (LayerNorm): LayerNorm((512,), eps=1e-12, 
elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=512, out_features=2048, 
bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=2048, out_features=512, 
bias=True)
            (LayerNorm): LayerNorm((512,), eps=1e-12, 
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=512, out_features=512, bias=True)
      (activation): Tanh()
    )
  )
  (hidden_layer): Linear(in_features=512, out_features=64, bias=True)
  (relu): ReLU()
  (classifier): Linear(in_features=64, out_features=3, bias=True)
  (log_softmax): LogSoftmax(dim=2)
)
```

Evaluation of Pretrained Model

Huggingface also hosts models which users have already trained on SNLI -- we can load them here and evaluate their performance on the validation and test set.

These models include the BertModel which we were using before, but also the trained weights for the classifier layer. Because of this, we'll use the standard HuggingFace classification model instead of the classifier we used above, and modify the training and prediction functions to handle this correctly.

Try and find the differences between the training loop. One addition is the new usage of "label_map". Why may this be necessary?

```
def class_predict(model, sents):

    with torch.inference_mode():
```

```
        logits = model(**sents.to(device)).logits
        predictions = torch.argmax(logits, axis=1)

    return predictions

def prediction_loop(model, dev_sents, dev_labels, label_map=None):
    print("Evaluating...")
    all_preds = []
    all_labels = []
    for sents, labels in tqdm(zip(dev_sents, dev_labels),
total=len(dev_sents)):
        pred = class_predict(model, sents).cpu()
        all_preds.extend(pred)
        all_labels.extend(list(labels.cpu().numpy()))

    dev_f1 = macro_f1(all_preds, all_labels,
possible_labels=set(all_labels), label_map = label_map)
    print(f"F1 {dev_f1}")

def class_training_loop(
    num_epochs,
    train_features,
    train_labels,
    dev_sents,
    dev_labels,
    optimizer,
    model,
    label_map=None
):
    print("Training...")
    loss_func = torch.nn.CrossEntropyLoss()
    batches = list(zip(train_features, train_labels))
    random.shuffle(batches)
    for i in range(num_epochs):
        losses = []
        for features, labels in tqdm(batches):
            # Empty the dynamic computation graph
            optimizer.zero_grad()
            logits = model(**features.to(device)).logits
            # preds = torch.argmax(logits, axis=1)
            loss = loss_func(logits, labels)
            # Backpropogate the loss through our model
            loss.backward()
            optimizer.step()
            losses.append(loss.item())

        print(f"epoch {i}, loss: {sum(losses)/len(losses)}")
        # Estimate the f1 score for the development set
    print("Evaluating dev...")
```

```
    all_preds = []
    all_labels = []
    for sents, labels in tqdm(zip(dev_sents, dev_labels),
total=len(dev_sents)):
        pred = predict(model, sents).cpu()
        all_preds.extend(pred)
        all_labels.extend(list(labels.numpy()))

    all_preds
    dev_f1 = macro_f1(all_preds, all_labels,
possible_labels=set(all_labels), label_map = label_map)
    print(f"Dev F1 {dev_f1}")

    # Return the trained model
    return model
```

Now we can load a model and re-tokenize our data.

```
## TODO: Get the label_map
## For the snli dataset  0=Entailment, 1=Neutral, 2=Contradiction
label_map = {0: 2, 1: 0, 2: 1}

from transformers import  BertForSequenceClassification, BertTokenizer


model_name = 'textattack/bert-base-uncased-snli'
tokenizer_model_name = 'textattack/bert-base-uncased-snli' # This is
sometimes different from model_name, but should normally be the same

model =  BertForSequenceClassification.from_pretrained(model_name)

model.to(device)

batch_tokenizer = BatchTokenizer(model_name = tokenizer_model_name)

validation_input_batches = [b for b in
chunk_multi(validation_premises, validation_hypotheses, batch_size)]

# Tokenize + encode
validation_input_batches = [batch_tokenizer(*batch) for batch in
validation_input_batches]
validation_batch_labels = [b for b in chunk(validation_labels,
batch_size)]
validation_batch_labels = [encode_labels(batch) for batch in
validation_batch_labels]

prediction_loop(model, validation_input_batches,
validation_batch_labels, label_map=label_map)
```

{"model_id":"7590f77c9e1e4f20bef492fb21f137f8","version_major":2,"version_minor":0}

{"model_id":"2053a91508f94f658ffa0fdefb382678","version_major":2,"version_minor":0}

{"model_id":"11d68df8407c4078bdb0f729bab801d4","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"5e1b6da311e84d17990bd625a134d425","version_major":2,"version_minor":0}

```
F1 0.6671987159037203
```

To complete this section, find 2 other BERT-based models which have been trained on natural language inference and evaluate them on the dev set. Note the scores for each model and any high-level differences you can find between the models (architecture, sizes, training data, etc.)

If you don't have access to a GPU, inference may be slow, particularly for larger models. In this case, take a sample of the validation set; the size should be large enough such that all labels are covered, and a score will still be meaningful, but also so that inference doesn't take more than 3-5 minutes.

Written Assignment

1. Describe the task and what capability is required to solve it.

The task is Natural Language Inference. More specifically, the task is to determine the relationship between two given sentences. Given the two sentences, the task will determine whether they entail each other, contradict each other, or are neutral. The capability required to solve this task is to understand the contextual nuances of the language and the ability to capture intricate relationships between sentences. The model needs to grasp the meaning of each word and their contextual significance.

2. How does the method of encoding sequences of words in our model differ here, compared to the word embeddings in HW 4. What is different? Why benefit does this method have?

Assumption: I am confused about the word embedding in HW 4, given that this is HW 4. By digging through Piazza, I found others to have the same question but no answer from the instructors. Therefore, I will compare word embedding representing words in a fixed-dimensional vector space and the one used in this model.

When word embeddings represent words in a fixed-dimensional vector space, it treats each word as an independent entity, ignoring the sequential and contextual information of the entire sentence. In contrast, the model employs BERT for sequence encoding. BERT utilizes a transformer architecture, allowing it to consider the entire context of a word and considering both the left and right contexts in all model layers. This bidirectional approach captures dependencies and relationships between words in a sentence more effectively than traditional word embeddings.

The benefit of BERT's contextualized embeddings lies in its ability to understand the meaning of a word in different contexts, which is crucial for tasks like NLI, where the relationship between sentences depends on the broader context. BERT captures syntactic and semantic relationships more comprehensively.

3. Discuss your results. Did you do any hyperparameter tuning? Did the model solve the task?

Epochs	LR	hidden size	loss	f1 score
20	0.01	10	1.102684445	0.1666666667
20	0.001	10	0.9560519908	0.5058339618
20	0.0005	10	0.914820546	0.5531573833
20	0.0001	10	0.9552818194	0.5323938593
20	0.00005	10	0.9928392407	0.4929300063
20	0.00001	10	1.063552415	0.4440518942
20	0.0005	20	0.8639361113	0.5441366922
20	0.0005	64	0.7498146749	0.5550072787
40	0.0005	64	0.572250533	0.5142913802
40	0.001	64	0.6929833419	0.5427011885
40	0.001	128	0.5330149962	0.4987773923

Hyperparameter tuning

LR

First, I started trying different learning rates. Given the small hint in the code provided and the fact that I uncommented the two lines of code, I started trying LR, which is very small. Through this, I saw a dramatic increase in the f1 score until it set around 0.001 to 0.00001.

hidden size

After confirming the range of the LR, I started trying different hidden sizes. However, based on the results, it was clear that the hidden size was not increasing the f1 score regardless of which ones I ran.

Epochs

Therefore, I started running more epochs to see if the f1 score would increase with more epochs. However, as shown in the results, there was no increase in the f1 score.

Did the model solve the task?

Given three classifications, and our model results in roughly 0.5, I think the model did solve the task, performing better than random. However, as the result indicates, there is a huge room for improvement.

Final discussion

Through the final test results, 0.5250253456769687. The model itself is better than randomly guessing the relationship. However, it is also clear that the model has room for improvement. For example, the f1 score in each epoch made it clear how the score would decrease if the loss also decreased. Hinting at possible overfitting. Additionally, when I had to switch between accounts when running the program and accidentally commented the two lines (only update classification layer), it yielded a f1 score of roughly 0.7. Therefore, it could indicate that the model is overfitting and needs additional information. However, that run without the two lines was extremely long. Additionally, given the site often checks inactivity, it was impossible to leave the code running and return to it later. With enough resources, I would like to run the model without the two lines.