Sarah Ahmad (saz848)
Vikram Kohli (vkp260)
Junhao Li (jlo708)

Checkpoint 5: Modeling with Neural Networks

Classifier goal: To predict whether the officers involved in a complaint will receive complaints in the future based on the text of summaries from past allegations.

To train a classifier to predict whether an officer will receive a complaint in the future based on the summary of an allegation, we created two sets of data for the classifier. The first contained the summary for each allegation with a summary as well as a label denoting whether the majority of active officers in the allegation had received a recent allegation, with a recent allegation being an allegation after 2014. The second dataset contained the combination of all allegations with summaries with the officers in the allegations and a label denoting whether the specific officer had received a recent allegation. Both datasets were constructed by exporting SQL queries on the CPDB to csv files.

After splitting this data into training and test datasets, we fed it into a modified version of a classifier neural network developed by OpenAI, and written in PyTorch. The OpenAI model uses as combination of transformers and unsupervised pre-training followed by discriminative fine-tuning on the particular task, and has been shown to achieve state of the art results on a variety of language tasks. Since the code for this model was written to be used for a multiple-choice task, we rewrote a substantial amount of the code in order to take our datasets as inputs, as well as to perform binary classification based on whether or not an officer will receive a complaint in the future given the summary of their previous complaint. (For reference, the unedited OpenAI classifier can be found at https://github.com/huggingface/pytorch-openai-transformer-lm/blob/master/model_pytorch.py); screenshots have been included at the bottom of this file that display the modifications made.

Following this modification, we split our data into a training set of size 1000 and a test set of size 80, and ran the network for 4 epochs. When testing in the case where allegations were repeated for each officer that had the same allegation placed against them, we found that our neural net had a maximum validation set accuracy of 78%, which is both relatively high and convincingly beat the ZeroR result of 70% after 4 epochs. However, in the case where allegation summaries were only considered once (and not for each officer who received the allegation), we were able to receive a test accuracy of only around 62%, which actually ended up being lower than the ZeroR result of 67% (unfortunately, this result was deleted, and so we're unable to

display the trace that produced it. Furthermore, even in the case where validation accuracy was high, the test accuracy for the second case only peaked at around 60.3% --- less than the performance of ZeroR, which achieves 72% accuracy. This is most likely not an issue with the model, and rather with the fact that our data set is rather small and doesn't include very many distinct summaries. Alternatively, this may point to the fact that the language used in the summaries of complaints against officers isn't a very good predictor of whether or not they are likely to have another allegation placed against them in the future. Determining which of these is the case would likely require a greater amount of summary data, as this would allow us to determine if the current model's failings are due to there just being insufficient information for training the neural net. Unfortunately, that vast majority of allegations listed do not have summaries and this seems consistent through the years of allegations so the amount of allegations without summaries does not appear to be changing.

Using the Pytorch OpenAI classifier as a basis, the words within the summaries were translated into vectors using their positions within sentences and pre-learned encodings. We had considered simpler features for encoding the summary such as mapping the words to integers or inputting an array of word frequencies but believed the encoded vectors to provide the best representation. However, experimentation with the form of the input features could certainly be explored further.

One issue we did appear to run into was that the training of the neural net was somewhat prohibitively slow; it took around 2 to 3 hours for the training to fully complete and for us to get our final results. This could be resolved by running on GPU, or by using a fewer number of examples / fine-tuning the code further.

Results

Queries for training/testing data

```sql
WITH data_repeatcomplaints as (
    SELECT officer_id,
        CASE WHEN SUM(CASE WHEN incident_date > '2013-12-31 23:59:59-06'
                           THEN 1
                           ELSE 0
                      END) > 0
             THEN 1
             ELSE 0
        END AS label
    FROM data_allegation AS a
    JOIN data_officerallegation AS oa ON a.id = oa.allegation_id
    JOIN data_officer AS o ON oa.officer_id = o.id
    WHERE active <> 'No'
    GROUP BY officer_id
    ORDER BY officer_id
)
SELECT allegation_id, oa.officer_id, summary, label
FROM data_allegation AS a
JOIN data_officerallegation AS oa ON a.id = oa.allegation_id
JOIN data_repeatcomplaints AS rc ON rc.officer_id = oa.officer_id
WHERE summary <> '';
```

```sql
WITH data_repeatcomplaints as (
    SELECT officer_id,
        CASE WHEN SUM(CASE WHEN incident_date > '2013-12-31 23:59:59-06'
                           THEN 1
                           ELSE 0
                      END) > 0
             THEN 1
             ELSE 0
        END AS recent_complaint
    FROM data_allegation AS a
    JOIN data_officerallegation AS oa ON a.id = oa.allegation_id
    JOIN data_officer AS o ON oa.officer_id = o.id
    WHERE active <> 'No'
    GROUP BY officer_id
    ORDER BY officer_id
)
SELECT allegation_id, summary,
    CASE WHEN SUM(recent_complaint) > (COUNT(recent_complaint) / 2.0)
         THEN 1
         ELSE 0
    END AS label
FROM data_allegation AS a
JOIN data_officerallegation AS oa ON a.id = oa.allegation_id
JOIN data_repeatcomplaints AS rc ON rc.officer_id = oa.officer_id
WHERE summary <> ''
GROUP BY allegation_id, summary;
```

Results from PyTorch neural net

Results after 4 epochs of training ---

read as (epoch, num training examples, validation loss, test set loss, validation accuracy, test accuracy)

```
(base) C:\Users\Vikram\Desktop\checkpoint_5>python train.py --dataset checkpoint5 --desc checkpoint5 --submit --analysis

Namespace(afn='gelu', analysis=True, attn_pdrop=0.1, b1=0.9, b2=0.999, bpe_path='model/vocab_40000.bpe', clf_pdrop=0.1,
data_dir='data/', dataset='checkpoint5', desc='checkpoint5', e=1e-08, embd_pdrop=0.1, encoder_path='model/encoder_bpe_40
000.json', l2=0.01, lm_coef=0.5, log_dir='log/', lr=6.25e-05, lr_schedule='warmup_linear', lr_warmup=0.002, max_grad_nor
m=1, n_batch=8, n_ctx=512, n_embd=768, n_head=12, n_iter=5, n_layer=12, n_transfer=12, n_valid=199, opt='adam', resid_pd
rop=0.1, save_dir='save/', seed=42, submission_dir='submission/', submit=True, vector_l2=False)
device cpu n_gpu 0
Encoding dataset...
C:\Users\Vikram\AppData\Local\Continuum\anaconda3\lib\site-packages\torch\nn\functional.py:52: UserWarning: size_average
 and reduce args will be deprecated, please use reduction='none' instead.
  warnings.warn(warning.format(ret))
Loading weights...
running epoch 0
Logging
1 100 22.284 27.479 67.84 60.30
running epoch 1
Logging
2 200 10.815 16.141 68.34 59.80
running epoch 2
Logging
3 300 23.680 37.422 68.34 59.80
running epoch 3
Logging
4 400 10.852 23.223 78.89 60.30
```

Screenshots of code for PyTorch neural net (our modifications can be found at https://github.com/JunhaoSLi/DataScienceF2018):

datasets.py (written to pipe the CSV data into the network)

datasets.py train.py

```python

def _checkpoint5(path):
    with open(path, encoding='utf_8') as f:
        f = csv.reader(f)
        statements = []
        results = []
        for i, line in enumerate(tqdm(list(f), ncols=80, leave=False)):
            statements.append(line[1])
            results.append(int(line[2]))
        return statements, results

def checkpoint5(data_dir, n_train=800, n_valid=199):
    statements, results = _checkpoint5(os.path.join(data_dir, 'training.csv'))
    test_statements, _ = _checkpoint5(os.path.join(data_dir, 'testing.csv'))
    tr_statements, va_statements, tr_results, va_results = train_test_split(statements, results, test_size=n_valid, random_state=seed)
    train_statements = []
    train_results = []
    for statement, result in zip(tr_statements, tr_results):
        train_statements.append(statement)
        train_results.append(result)

    val_statements = []
    val_results = []
    for statement, result in zip(va_statements, va_results):
        val_statements.append(statement)
        val_results.append(result)

    train_results = np.asarray(train_results, dtype=np.int32)
    val_results = np.asarray(val_results, dtype=np.int32)

    return (train_statements, train_results), (val_statements, val_results), test_statements
```

train.py (creates the network, does training, displays results)

```python
def transform_checkpoint5(statements):
    n_batch = len(statements)
    xmb = np.zeros((n_batch, n_ctx, 2), dtype=np.int32)
    mmb = np.zeros((n_batch, n_ctx), dtype=np.float32)
    start = encoder['_start_']
    for i, statement in enumerate(statements):
        if not isinstance(statement[0], int):
            statement = sum(statement, [])
        full = [start] + statement[:max_len] + [clf_token]
        length = len(full)
        xmb[i, :length, 0] = full
        mmb[i, :length] = 1
    xmb[:,:,1] = np.arange(n_vocab + n_special, n_vocab + n_special + n_ctx)
    return xmb, mmb

def iter_apply(Xs, Ms, Ys):
    # fns = [lambda x: np.concatenate(x, 0), lambda x: float(np.sum(x))]
    logits = []
    cost = 0
    with torch.no_grad():
        dh_model.eval()
        for xmb, mmb, ymb in iter_data(Xs, Ms, Ys, n_batch=n_batch_train, truncate=False, verbose=True):
            n = len(xmb)
            XMB = torch.tensor(xmb, dtype=torch.long).to(device)
            YMB = torch.tensor(ymb, dtype=torch.long).to(device)
            MMB = torch.tensor(mmb).to(device)
            _, clf_logits = dh_model(XMB)
            clf_logits *= n
            clf_losses = compute_loss_fct(XMB, YMB, MMB, clf_logits, only_return_losses=True)
            clf_losses *= n
            logits.append(clf_logits.to("cpu").numpy())
```

```python
def iter_predict(Xs, Ms):
    logits = []
    with torch.no_grad():
        dh_model.eval()
        for xmb, mmb in iter_data(Xs, Ms, n_batch=n_batch_train, truncate=False, verbose=True):
            n = len(xmb)
            XMB = torch.tensor(xmb, dtype=torch.long).to(device)
            MMB = torch.tensor(mmb).to(device)
            _, clf_logits = dh_model(XMB)
            logits.append(clf_logits.to("cpu").numpy())
    logits = np.concatenate(logits, 0)
    return logits


def log(save_dir, desc):
    global best_score
    print("Logging")
    tr_logits, tr_cost = iter_apply(trX[:n_valid], trM[:n_valid], trY[:n_valid])
    va_logits, va_cost = iter_apply(vaX, vaM, vaY)
    tr_cost = tr_cost / len(trY[:n_valid])
    va_cost = va_cost / n_valid
    tr_acc = accuracy_score(trY[:n_valid], np.argmax(tr_logits, 1)) * 100.
    va_acc = accuracy_score(vaY, np.argmax(va_logits, 1)) * 100.
    logger.log(n_epochs=n_epochs, n_updates=n_updates, tr_cost=tr_cost, va_cost=va_cost, tr_acc=tr_acc, va_acc=va_acc)
    print('%d %d %.3f %.3f %.2f %.2f' % (n_epochs, n_updates, tr_cost, va_cost, tr_acc, va_acc))
    if submit:
        score = va_acc
        if score > best_score:
            best_score = score
            path = os.path.join(save_dir, desc, 'best_params')
            torch.save(dh_model.state_dict(), make_path(path))
```

```python
def predict(dataset, submission_dir):
    filename = filenames[dataset]
    pred_fn = pred_fns[dataset]
    label_decoder = label_decoders[dataset]
    predictions = pred_fn(iter_predict(teX, teM))
    if label_decoder is not None:
        predictions = [label_decoder[prediction] for prediction in predictions]
    path = os.path.join(submission_dir, filename)
    os.makedirs(os.path.dirname(path), exist_ok=True)
    with open(path, 'w') as f:
        f.write('{}\t{}\n'.format('index', 'prediction'))
        for i, prediction in enumerate(predictions):
            f.write('{}\t{}\n'.format(i, prediction))


def run_epoch():
    for xmb, mmb, ymb in iter_data(*shuffle(trX, trM, trYt, random_state=np.random),
                                   n_batch=n_batch_train, truncate=True, verbose=True):
        global n_updates
        dh_model.train()
        XMB = torch.tensor(xmb, dtype=torch.long).to(device)
        YMB = torch.tensor(ymb, dtype=torch.long).to(device)
        MMB = torch.tensor(mmb).to(device)
        lm_logits, clf_logits = dh_model(XMB)
        compute_loss_fct(XMB, YMB, MMB, clf_logits, lm_logits)
        n_updates += 1
        if n_updates in [1000, 2000, 4000, 8000, 16000, 32000] and n_epochs == 0:
            log(save_dir, desc)


argmax = lambda x: np.argmax(x, 1)
```

```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--desc', type=str, help="Description")
    parser.add_argument('--dataset', type=str)
    parser.add_argument('--log_dir', type=str, default='log/')
    parser.add_argument('--save_dir', type=str, default='save/')
    parser.add_argument('--data_dir', type=str, default='data/')
    parser.add_argument('--submission_dir', type=str, default='submission/')
    parser.add_argument('--submit', action='store_true')
    parser.add_argument('--analysis', action='store_true')
    parser.add_argument('--seed', type=int, default=42)
    parser.add_argument('--n_iter', type=int, default=5)
    parser.add_argument('--n_batch', type=int, default=8)
    parser.add_argument('--max_grad_norm', type=int, default=1)
    parser.add_argument('--lr', type=float, default=6.25e-5)
    parser.add_argument('--lr_warmup', type=float, default=0.002)
    parser.add_argument('--n_ctx', type=int, default=512)
    parser.add_argument('--n_embd', type=int, default=768)
    parser.add_argument('--n_head', type=int, default=12)
    parser.add_argument('--n_layer', type=int, default=12)
    parser.add_argument('--embd_pdrop', type=float, default=0.1)
    parser.add_argument('--attn_pdrop', type=float, default=0.1)
    parser.add_argument('--resid_pdrop', type=float, default=0.1)
    parser.add_argument('--clf_pdrop', type=float, default=0.1)
    parser.add_argument('--l2', type=float, default=0.01)
    parser.add_argument('--vector_l2', action='store_true')
    parser.add_argument('--opt', type=str, default='adam')
    parser.add_argument('--afn', type=str, default='gelu')
    parser.add_argument('--lr_schedule', type=str, default='warmup_linear')
    parser.add_argument('--encoder_path', type=str, default='model/encoder_bpe_40000.json')
    parser.add_argument('--bpe_path', type=str, default='model/vocab_40000.bpe')
```

```python
    random.seed(args.seed)
    np.random.seed(args.seed)
    torch.manual_seed(args.seed)
    torch.cuda.manual_seed_all(args.seed)

    # Constants
    submit = args.submit
    dataset = args.dataset
    n_ctx = args.n_ctx
    save_dir = args.save_dir
    desc = args.desc
    data_dir = args.data_dir
    log_dir = args.log_dir
    submission_dir = args.submission_dir

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    n_gpu = torch.cuda.device_count()
    print("device", device, "n_gpu", n_gpu)

    logger = ResultLogger(path=os.path.join(log_dir, '{}.jsonl'.format(desc)), **args.__dict__)
    text_encoder = TextEncoder(args.encoder_path, args.bpe_path)
    encoder = text_encoder.encoder
    n_vocab = len(text_encoder.encoder)

    print("Encoding dataset...")
    ((trX1, trY),
     (vaX1, vaY),
     teX1) = encode_dataset(*checkpoint5(data_dir, n_valid=args.n_valid),
                            encoder=text_encoder)

    #print(trX1)
    encoder['_start_'] = len(encoder)
```

```
    #print(trX1)
    encoder['_start_'] = len(encoder)
    encoder['_delimiter_'] = len(encoder)
    encoder['_classify_'] = len(encoder)
    clf_token = encoder['_classify_']
    n_special = 3
    max_len = n_ctx // 2 - 2
    n_ctx = min(max(
            [len(x1[:max_len]) for x1 in trX1]
            + [len(x1[:max_len]) for x1 in vaX1]
            + [len(x1[:max_len]) for x1 in teX1]
            ) + 2, n_ctx)

    vocab = n_vocab + n_special + n_ctx
    trX, trM = transform_checkpoint5(trX1)
    vaX, vaM = transform_checkpoint5(vaX1)
    if submit:
        teX, teM = transform_checkpoint5(teX1)

    n_train = len(trY)
    n_valid = len(vaY)
    n_batch_train = args.n_batch * max(n_gpu, 1)
    n_updates_total = (n_train // n_batch_train) * args.n_iter

    dh_model = DoubleHeadModel(args, clf_token, ('classification', 2), vocab, n_ctx)

    criterion = nn.CrossEntropyLoss(reduce=False)
    model_opt = OpenAIAdam(dh_model.parameters(),
                           lr=args.lr,
                           schedule=args.lr_schedule,
                           warmup=args.lr_warmup,
                           t_total=n_updates_total,
```

```
                                max_grad_norm=args.max_grad_norm)
    compute_loss_fct = ClassificationLossCompute(criterion,
                                                 criterion,
                                                 args.lm_coef,
                                                 model_opt)
    load_openai_pretrained_model(dh_model.transformer, n_ctx=n_ctx, n_special=n_special)

    dh_model.to(device)
    dh_model = nn.DataParallel(dh_model)

    n_updates = 0
    n_epochs = 0
    if dataset != 'stsb':
        trYt = trY
    if submit:
        path = os.path.join(save_dir, desc, 'best_params')
        torch.save(dh_model.state_dict(), make_path(path))
    best_score = 0
    for i in range(args.n_iter):
        print("running epoch", i)
        run_epoch()
        n_epochs += 1
        log(save_dir, desc)
    if submit:
        path = os.path.join(save_dir, desc, 'best_params')
        dh_model.load_state_dict(torch.load(path))
        predict(dataset, args.submission_dir)
        if args.analysis:
            checkpoint5_analysis(data_dir, os.path.join(args.submission_dir, 'Checkpoint5.tsv'),
                                 os.path.join(log_dir, 'checkpoint5.jsonl'))
```

```
import argparse
import os
import random

import numpy as np
import torch
import torch.nn as nn
from sklearn.metrics import accuracy_score
from sklearn.utils import shuffle

from analysis import checkpoint5 as checkpoint5_analysis
from datasets import checkpoint5
from model_pytorch import DoubleHeadModel, load_openai_pretrained_model
from opt import OpenAIAdam
from text_utils import TextEncoder
from utils import (encode_dataset, iter_data,
                   ResultLogger, make_path)
from loss import MultipleChoiceLossCompute, ClassificationLossCompute

'''
to run:
    python train.py --dataset checkpoint5 --desc checkpoint5 --submit --analysis
'''

def transform_checkpoint5(statements):
    n_batch = len(statements)
    xmb = np.zeros((n_batch, n_ctx, 2), dtype=np.int32)
    mmb = np.zeros((n_batch, n_ctx), dtype=np.float32)
    start = encoder['_start_']
    for i, statement in enumerate(statements):
        if not isinstance(statement[0], int):
            statement = sum(statement, [])
        full = [start] + statement[:max_len] + [clf_token]
```