Introduction

Andrew Jeffers, Haesung Jeong
GitHub Repo Link: https://github.com/ajeffers999/CS598-Project

Project Video Link:
https://drive.google.com/file/d/1q8SYeYhuX0JAGkiHEd1dMHqpRALThh7_/view?usp=drive_link

The paper explores an approach to determine how a patient's family members' medical history influences their disease risk. This is a meaningful problem because it could help inform patients about their risk for certain diseases based on relative and family information. Utlizing family medical history for predicting a patient's disease risk is also complicated by a variety of genetic, environmental, and lifestyle factors.

This paper proposes a novel solution to this problem by utilizing a graph-based deep learning approach for learning representations of family member's influence on patient's disease risk. A graph based approach is a more useful and natural way of modeling the connections between family members than previous methods.

Previous works have also recognized that it is useful to include information from family members when predicting the risk of disease. However, machine learning approaches using tabular data do not model the underlying geometric structure of family history. Using a graph based approach, this structure is much more easily obtained and modeled.

The main contributions of the paper are:

- a scalable, disease-agnostic machine learning tool making use of GNNs and LSTMs which learn representations of a patient's disease risk from family member's medical information.
- Data which shows graph-based approaches perform better than clinically-inspired or deep learning baselines used previously.
- Graph explanability techniques demonstrate that GNN-LSTM embeddings identify medical features which are more suitable for predicting disease risk than features identifies by an epidemiological baseline.

The researchers observed that graph-based models consistently outperformed the baseline approaches, although the best performing model between GNN and GNN-LSTM varied depending on the disease in question. Cancers typically performed better on the GNN model, which the researchers believe is due to cancers generally being less hereditary than other diseases.

› Mount Notebook to Google Drive

Upload the data, pretrianed model, figures, etc to your Google Drive, then mount this notebook to Google Drive. After that, you can access the resources freely.

[ ] ↳ *1 cell hidden*

Scope of Reproducibility:

List hypotheses from the paper you will test and the corresponding experiments you will run.

1. Hypothesis 1: Graph-based approaches predict disease risk better than the baseline model.
2. Hypothesis 2: The GNN model using GraphConv layers predicts disease risk better than the GNN model using GCN layers.

Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the experiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

› Python Version

Python 3.10.12

[ ] ↳ *1 cell hidden*

› Libraries

Below are the Python libraries used to implement graph representation learning for familial relationships.

[ ] ↳ *11 cells hidden*

› Custom Loss function

Class imbalance arises because the models are predicting diseases with prevalence < 10% in the selected cohorts. The authors of the paper alleviate this by using class-weighted loss functions, sampling strategies designed for imbalanced classification, methods to prevent overfitting, and careful choice and interpretation of evaluation metrics (Wharrie, Sophie, et al.).

[ ] ↳ *14 cells hidden*

Results

Baseline:

- Area Under the Receiver Operating Characteristic Curve (auc_roc): 0.7423616531503197
- Compute Area Under the Curve (auc): 0.24469842750122853
- F1 score: 0.3078584931136916
- Recall: 0.6529209621993127
- MCC: 0.21566619661004827

GNN (graphconv):

- Area Under the Receiver Operating Characteristic Curve (auc_roc): 0.7561574424503961
- Compute Area Under the Curve (auc): 0.2501534301041718
- F1 score: 0.3313351498637602
- Recall: 0.6964490263459335
- MCC: 0.25146318920227423

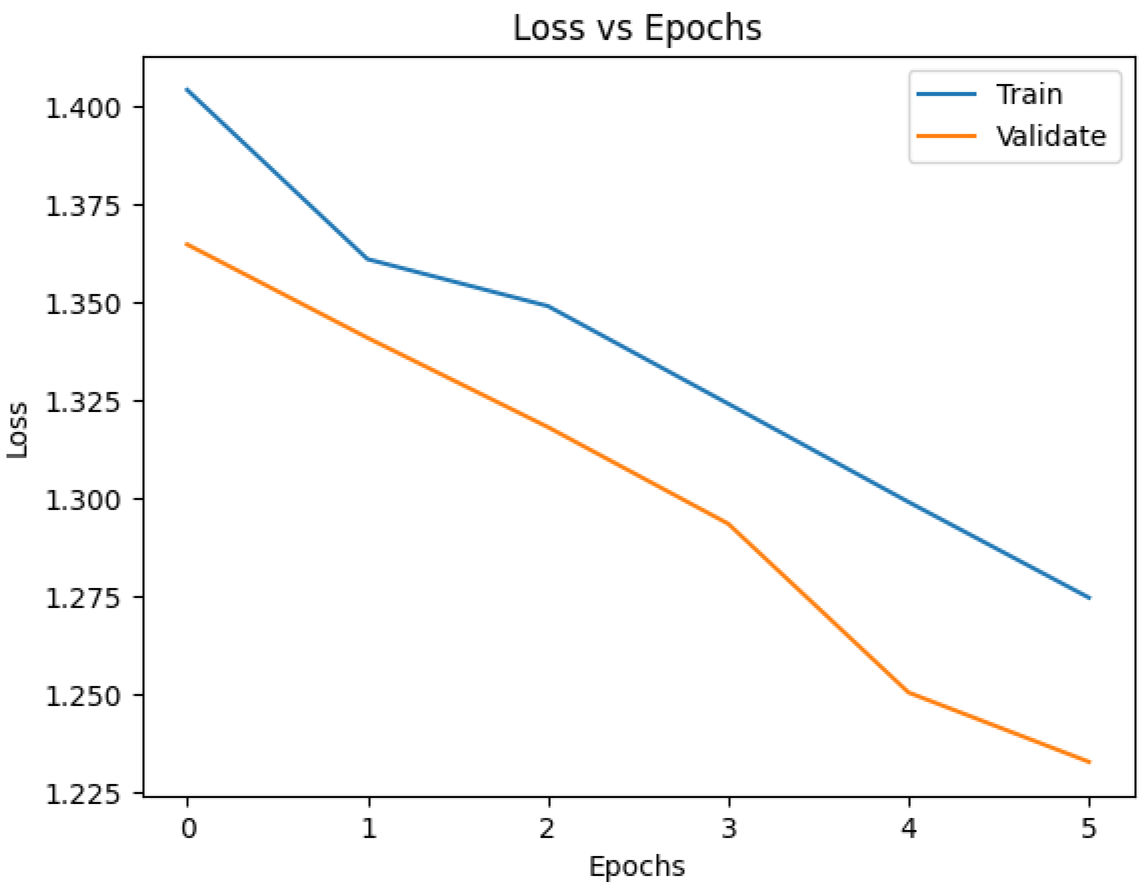
GNN (gcn):

- Area Under the Receiver Operating Characteristic Curve (auc_roc): 0.7582678910645871
- Compute Area Under the Curve (auc): 0.2615118219594704
- F1 score: 0.33721922092692636
- Recall: 0.6792668957617412
- MCC: 0.2565234631792809

```
# Download model checkpoint
gdown.download('https://drive.google.com/uc?id=15w1eJRql0MzIBu5x4sxJYYS7hrXvRCO2')
checkpoint = torch.load('/content/baseline.pt')
model = Baseline(num_features_static, main_hidden_dim, dropout_rate)
model.load_state_dict(checkpoint['model_state_dict'])
model.eval()

baseline_train_losses = checkpoint['train_losses']
baseline_valid_losses = checkpoint['valid_losses']
baseline_threshold = checkpoint['threshold']

# Test
num_samples = 3
baseline_test_output = [np.array([]) for _ in range(num_samples)]
baseline_test_y = [np.array([]) for _ in range(num_samples)]
representations = pd.DataFrame()

test_patient_list = fetch_data.test_patient_list
num_batches_test = int(np.ceil(len(test_patient_list)/batch_size))
test_dataset, test_loader = get_data_and_loader(test_patient_list, fetch_data, model_type, b

for m in model.modules():
  if m.__class__.__name__.startswith('Dropout'):
    m.train()

for sample in range(num_samples):
    for test_batch in tqdm(test_loader, total=num_batches_test):
        x_static, y = test_batch[0][0], test_batch[1][0].unsqueeze(1)
        output = model(x_static)
        baseline_test_output[sample] = np.concatenate((baseline_test_output[sample], output.
        baseline_test_y[sample] = np.concatenate((baseline_test_y[sample], y.reshape(-1).det

# metrics to evaluate my model
plot_losses(baseline_train_losses, baseline_valid_losses, 'baseline')

# report standard error for uncertainty
baseline_test_output_se = np.array(baseline_test_output).std(axis=0) / np.sqrt(num_samples)

# take average over all samples to get expected value
baseline_test_output = np.array(baseline_test_output).mean(axis=0)
baseline_test_y = np.array(baseline_test_y).mean(axis=0)

baseline_results = pd.DataFrame({'actual':baseline_test_y, 'pred_raw':baseline_test_output,
baseline_results['pred_binary'] = (baseline_results['pred_raw']>baseline_threshold).astype(i
baseline_metric_results = calculate_metrics(baseline_results['actual'], baseline_results['pr
print(baseline_metric_results)

# plot figures to better show the results
plt.plot(baseline_train_losses, label='Train')
plt.plot(baseline_valid_losses, label='Validate')
plt.title('Loss vs Epochs')
```

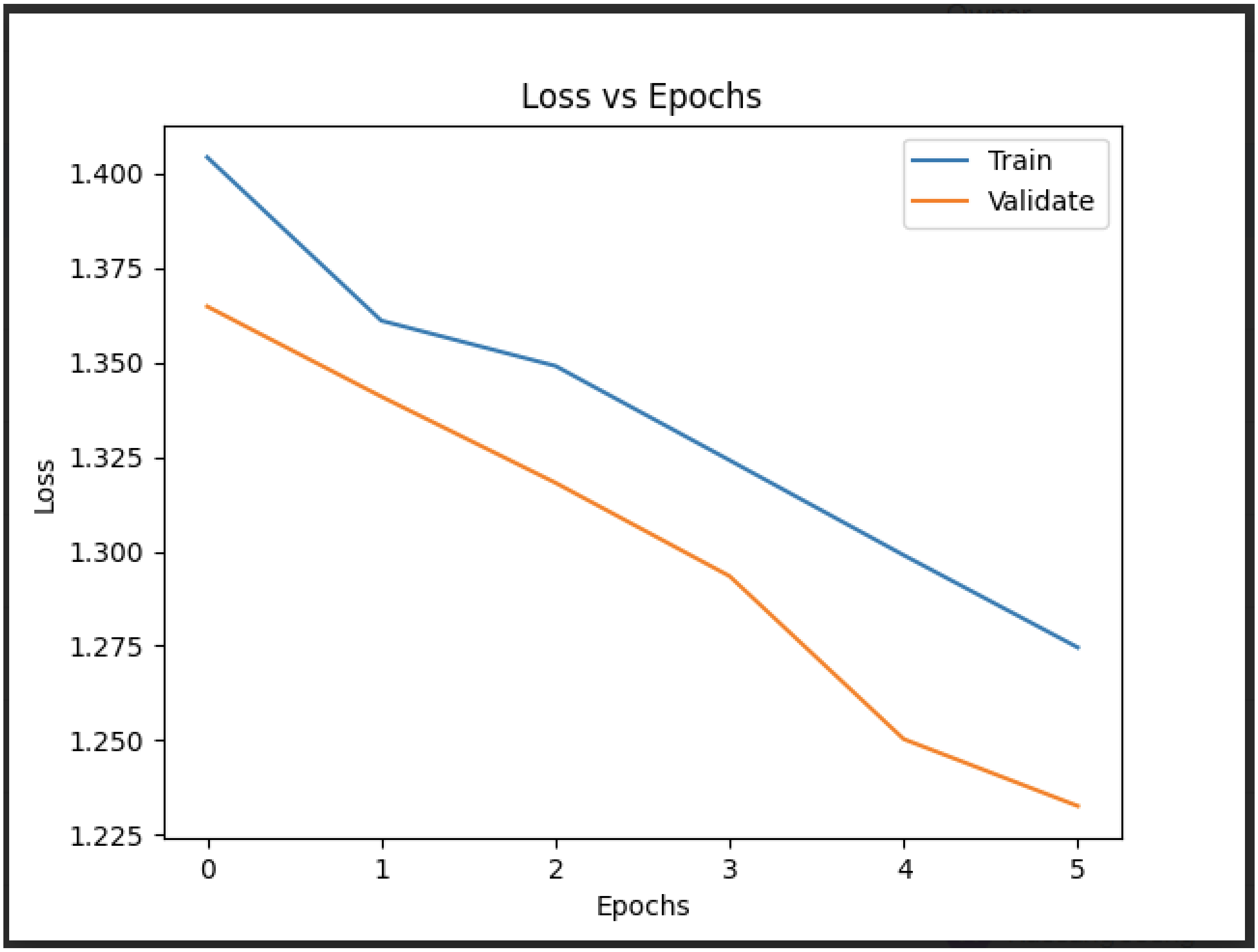
```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.clf()
```

```
Downloading...
From: https://drive.google.com/uc?id=15w1eJRql0MzIBu5x4sxJYYS7hrXvRCO2
To: /content/baseline.pt
100%|██████████| 16.9k/16.9k [00:00<00:00, 29.9MB/s]
  0%|          | 0/32 [00:00<?, ?it/s]/usr/lib/python3.10/multiprocessing/popen_fork.py:
  self.pid = os.fork()
100%|██████████| 32/32 [00:00<00:00, 189.65it/s]
100%|██████████| 32/32 [00:00<00:00, 56.14it/s]
100%|██████████| 32/32 [00:00<00:00, 206.13it/s]
{'metric_auc_roc': 0.7423616531503197, 'metric_auc_prc': 0.24469842750122853, 'metric_f1
```




```
<Figure size 640x480 with 0 Axes>
```

Baseline Model Loss:



Model comparison

GraphConv Graph model

The GNN model using GraphConv layers for non-longitudinal data.

```
# Declare data variables, loss function, and optimizer
model_type = 'graph'
gnn_layer='graphconv'
pooling_method = 'target'
main_hidden_dim=20 # used for both GNN and MLP
lstm_hidden_dim=20 # x2 for bidirectional LSTM
ratio = 0.5
gamma = 1
alpha = 1
beta = 1
delta = 1
batch_size=250
dropout_rate = 0.5
learning_rate = 0.001

fetch_data = DataFetch(model_type=model_type, featfile='featfile_G1.csv', gnn_layer=gnn_laye
train_patient_list = fetch_data.train_patient_list
validate_patient_list = fetch_data.validate_patient_list
num_features_static = len(fetch_data.static_features)
num_features_alt_static = len(fetch_data.alt_static_features)
num_samples_train_dataset = len(train_patient_list)
num_samples_valid_dataset = len(validate_patient_list)
num_samples_train_minority_class = fetch_data.num_samples_train_minority_class
num_samples_valid_minority_class = fetch_data.num_samples_valid_minority_class
num_samples_train_majority_class = fetch_data.num_samples_train_majority_class
num_samples_valid_majority_class = fetch_data.num_samples_valid_majority_class

model = GNN(num_features_static, num_features_alt_static, main_hidden_dim, gnn_layer, poolir

loss_func = WeightedBCELoss(num_samples_train_dataset, num_samples_train_minority_class, num
valid_loss_func = WeightedBCELoss(num_samples_valid_dataset, num_samples_valid_minority_clas
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

train_dataset, train_loader = get_data_and_loader(train_patient_list, fetch_data, model_type
validate_dataset, validate_loader = get_data_and_loader(validate_patient_list, fetch_data, m

print('static_data: ', len(fetch_data.static_data))
print('label data: ', len(fetch_data.label_data))
print('True label: ', fetch_data.label_data.sum().item())
print('False label: ', len(fetch_data.label_data) -fetch_data.label_data.sum().item())
print('Train dataset: ', len(train_patient_list))
print('Validate dataset: ', len(validate_patient_list))
print('Test dataset: ', len(fetch_data.test_patient_list))
```

```
Using GraphConv layers
static_data:  150000
label data:  150000
True label:  57297.0
False label:  92703.0
Train dataset:  27565
Validate dataset:  3872
Test dataset:  7860
```

Below training step is used to generate checkpoint file 'gnn-graphconv.pt'

you don't need to re-run all other experiments, instead, you can directly refer the metrics/numbers in the paper

```
# compare you model with others
# start_time_train = time.time()
# num_epoch = 100
# num_batches_train = int(np.ceil(len(train_patient_list)/batch_size))
# num_batches_validate = int(np.ceil(len(validate_patient_list)/batch_size))

# # model training loop: it is better to print the training/validation losses during the tra
# train_losses = []
# valid_losses = []
# separate_loss_terms = {'NN_train':[], 'target_train':[], 'family_train':[], 'lstm_train':[

# model_path = raw_data_dir + 'drive/MyDrive/CS598 DLH/Project/gnn-graphconv.pt'
# train_model_path = raw_data_dir + 'drive/MyDrive/CS598 DLH/Project/train-gnn-graphconv.pt'
# graphconv_early_stopping = EarlyStopping(path=train_model_path)

# for i in range(num_epoch):
#   model.train()
#   epoch_train_loss = []
#   separate_loss_terms_epoch = {'NN_train':[], 'target_train':[], 'family_train':[], 'lstm_

#   for train_batch in tqdm(train_loader, total=num_batches_train):
#     x_static_node, x_static_graph, y, edge_index, edge_weight, batch, target_index = trair
#     output, patient_output, family_output = model(x_static_node, x_static_graph, edge_inde

#     # combined loss that considers the additive effect of patient and family effects
#     loss_term_NN = gamma * loss_func(output, y)
#     loss_term_target = alpha * loss_func(patient_output, y)
#     loss_term_family = beta * loss_func(family_output, y)

#     separate_loss_terms_epoch['NN_train'].append(loss_term_NN.item())
#     separate_loss_terms_epoch['target_train'].append(loss_term_target.item())
#     separate_loss_terms_epoch['family_train'].append(loss_term_family.item())

#     loss = loss_term_NN + loss_term_target + loss_term_family

#     optimizer.zero_grad()
#     loss.backward()
#     optimizer.step()
#     epoch_train_loss.append(loss.item())

#   # eval on validset
#   model.eval()
#   epoch_valid_loss = []
#   valid_output = np.array([])
#   valid_y = np.array([])
#   for valid_batch in tqdm(validate_loader, total=num_batches_validate):
#     x_static_node, x_static_graph, y, edge_index, edge_weight, batch, target_index = valic
#     output, patient_output, family_output = model(x_static_node, x_static_graph, edge_inde

#     valid_output = np.concatenate((valid_output, output.reshape(-1).detach().cpu().numpy()
#     valid_y = np.concatenate((valid_y, y.reshape(-1).detach().cpu().numpy()))
```

```
#     # combined loss that considers the additive effect of patient and family effects
#     loss_term_NN = gamma * valid_loss_func(output, y)
#     loss_term_target = alpha * valid_loss_func(patient_output, y)
#     loss_term_family = beta * valid_loss_func(family_output, y)

#     separate_loss_terms_epoch['NN_valid'].append(loss_term_NN.item())
#     separate_loss_terms_epoch['target_valid'].append(loss_term_target.item())
#     separate_loss_terms_epoch['family_valid'].append(loss_term_family.item())

#     loss = loss_term_NN + loss_term_target + loss_term_family
#     epoch_valid_loss.append(loss.item())

#   train_loss, valid_loss = np.mean(epoch_train_loss), np.mean(epoch_valid_loss)
#   train_losses.append(train_loss)
#   valid_losses.append(valid_loss)

#   for term_name in separate_loss_terms:
#     separate_loss_terms[term_name].append(np.mean(separate_loss_terms_epoch[term_name]))
#   print("epoch {}\ttrain loss : {}\tvalidate loss : {}".format(i, train_loss, valid_loss))

#   graphconv_early_stopping(np.mean(epoch_valid_loss), model)
#   if graphconv_early_stopping.early_stop:
#     print('Early Stopping')
#     break

# fpr, tpr, thresholds = metrics.roc_curve(valid_y, valid_output)
# gmeans = np.sqrt(tpr * (1-fpr))
# ix = np.argmax(gmeans)
# auc_threshold = thresholds[ix]

# pr_thresholds = np.arange(0.1, 0.9, 0.001) # between 0.1 and 0.9 to exclude trivial values
# scores = [metrics.f1_score(valid_y, (valid_output >= t).astype('int')) for t in pr_thresho
# pr_ix = np.argmax(scores)
# pr_threshold = pr_thresholds[pr_ix]

# plot_losses(train_losses, valid_losses, 'graphconv')
# plot_separate_losses(separate_loss_terms['NN_train'], separate_loss_terms['target_train'],
# plot_separate_losses(separate_loss_terms['NN_valid'], separate_loss_terms['target_valid'],

# end_time_train = time.time()
# torch.save({
#     'epoch': num_epoch,
#     'model_state_dict': model.state_dict(),
#     'optimizer_state_dict': optimizer.state_dict(),
#     'threshold': auc_threshold,
#     'start_time': start_time_train,
#     'end_time': end_time_train
#     }, model_path)
# you don't need to re-run all other experiments, instead, you can directly refer the metric
```

```
  0%|          | 0/111 [00:00<?, ?it/s]/usr/lib/python3.10/multiprocessing/popen_fork.py
  self.pid = os.fork()
100%|██████████| 111/111 [02:01<00:00,  1.34it/s]/usr/lib/python3.10/multiprocessing/pop
  self.pid = os.fork()
100%|██████████| 111/111 [02:01<00:00,  1.10s/it]
100%|██████████| 16/16 [00:16<00:00,  1.05s/it]
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:3504: RuntimeWarning:
  return _methods._mean(a, axis=axis, dtype=dtype,
/usr/local/lib/python3.10/dist-packages/numpy/core/_methods.py:129: RuntimeWarning: inva
  ret = ret.dtype.type(ret / rcount)
epoch 0 train loss : 7.134208290426581  validate loss : 4.1087020337581635
100%|██████████| 111/111 [02:07<00:00,  1.15s/it]
100%|██████████| 16/16 [00:17<00:00,  1.09s/it]
epoch 1 train loss : 4.139859311215512  validate loss : 4.124679610133171
EarlyStopping counter: 1 out of 5
100%|██████████| 111/111 [02:07<00:00,  1.15s/it]
100%|██████████| 16/16 [00:17<00:00,  1.11s/it]
epoch 2 train loss : 4.112498246871673  validate loss : 4.036566853523254
EarlyStopping counter: 2 out of 5
100%|██████████| 111/111 [02:08<00:00,  1.15s/it]
100%|██████████| 16/16 [00:17<00:00,  1.11s/it]
epoch 3 train loss : 4.047164906252612  validate loss : 3.9392164796590805
EarlyStopping counter: 3 out of 5
100%|██████████| 111/111 [02:07<00:00,  1.15s/it]
100%|██████████| 16/16 [00:16<00:00,  1.05s/it]
epoch 4 train loss : 3.9511228982392734 validate loss : 3.8099250346422195
EarlyStopping counter: 4 out of 5
100%|██████████| 111/111 [02:02<00:00,  1.11s/it]
100%|██████████| 16/16 [00:16<00:00,  1.06s/it]
epoch 5 train loss : 3.849997937142312  validate loss : 3.741437718272209
EarlyStopping counter: 5 out of 5
Early Stopping
<Figure size 640x480 with 0 Axes>
```

Below training step is used to generate checkpoint file 'test-gnn-graphconv.pt'

you don't need to re-run all other experiments, instead, you can directly refer the metrics/numbers in the paper

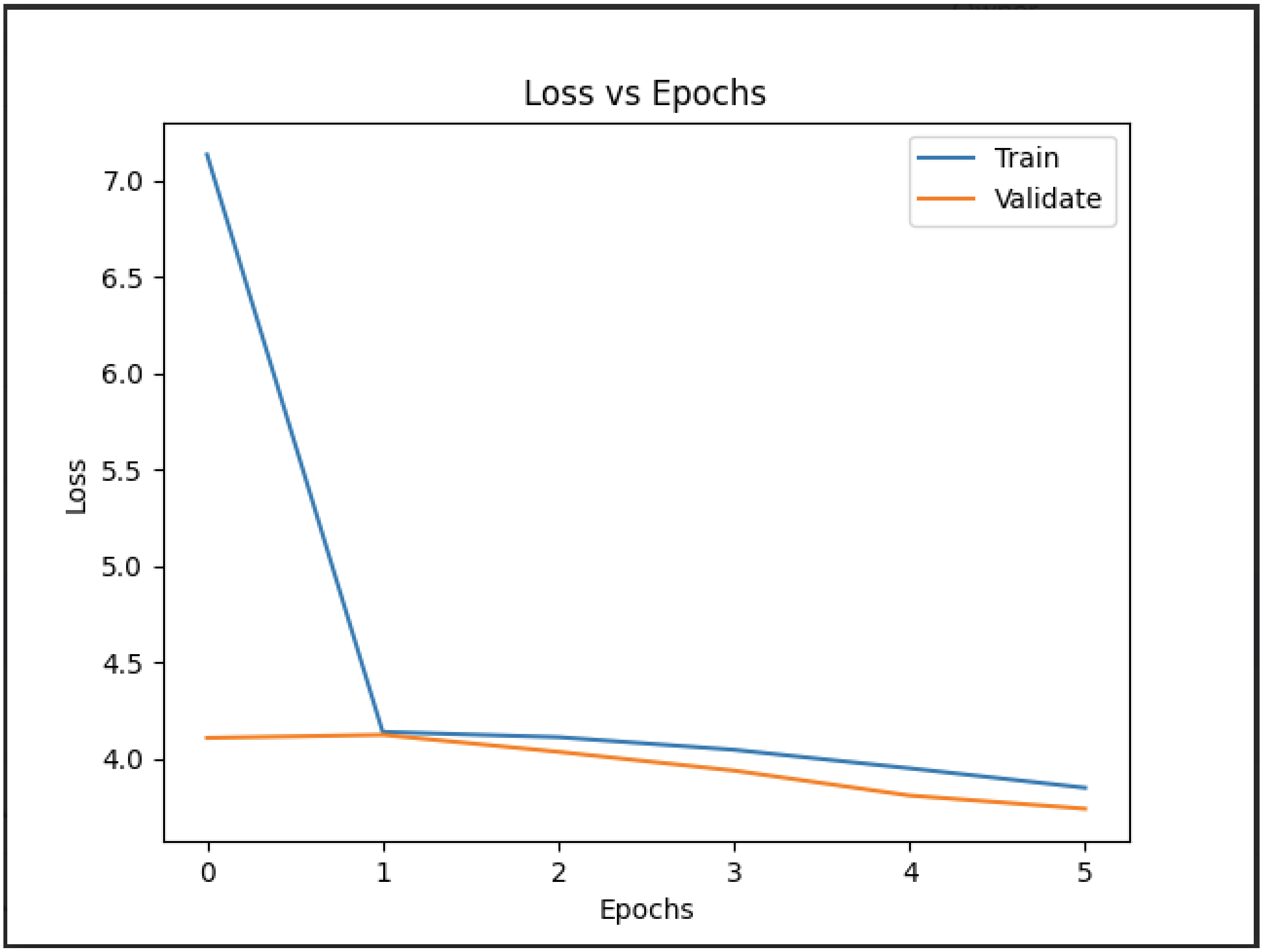
```
# metrics to evaluate my model
# gdown.download('https://drive.google.com/uc?id=1-2UuB6VQQpc_GyMf3ekwxweiZw6xGilc')
# gnn_checkpoint = torch.load('/content/gnn-graphconv.pt')
# gnn_model = GNN(num_features_static, num_features_alt_static, main_hidden_dim, gnn_layer,
# gnn_model.load_state_dict(gnn_checkpoint['model_state_dict'])
# gnn_model.eval()

# gnn_threshold = gnn_checkpoint['threshold']
# gnn_start_time = gnn_checkpoint['start_time']
# gnn_end_time = gnn_checkpoint['end_time']

# num_samples = 3
# gnn_graphconv_test_output = [np.array([]) for _ in range(num_samples)]
# gnn_graphconv_test_y = [np.array([]) for _ in range(num_samples)]
# representations = pd.DataFrame()

# test_patient_list = fetch_data.test_patient_list
# num_batches_test = int(np.ceil(len(test_patient_list)/batch_size))
# test_dataset, test_loader = get_data_and_loader(test_patient_list, fetch_data, model_type,

# test_model_path = raw_data_dir + 'drive/MyDrive/CS598 DLH/Project/test-gnn-graphconv.pt'
# for m in gnn_model.modules():
#   if m.__class__.__name__.startswith('Dropout'):
#     m.train()

# for sample in range(num_samples):
#   for test_batch in tqdm(test_loader, total=num_batches_test):
#     x_static_node, x_static_graph, y, edge_index, edge_weight, batch, target_index = test_
#     output, patient_output, family_output = gnn_model(x_static_node, x_static_graph, edge_
#     gnn_graphconv_test_output[sample] = np.concatenate((gnn_graphconv_test_output[sample],
#     gnn_graphconv_test_y[sample] = np.concatenate((gnn_graphconv_test_y[sample], y.reshape

# torch.save({
#     'epoch': num_samples,
#     'test_output': gnn_graphconv_test_output,
#     'test_y': gnn_graphconv_test_y,
#     'threshold': auc_threshold,
#     'start_time': gnn_start_time,
#     'end_time': gnn_end_time
#     }, test_model_path)
```

```
Downloading...
From: https://drive.google.com/uc?id=1-2UuB6VQQpc_GyMf3ekwxweiZw6xGilc
To: /content/gnn-graphconv.pt
100%|██████████| 54.4k/54.4k [00:00<00:00, 4.63MB/s]
Using GraphConv layers
  0%|          | 0/32 [00:00<?, ?it/s]/usr/lib/python3.10/multiprocessing/popen_fork.py:
  self.pid = os.fork()
100%|██████████| 32/32 [00:34<00:00,  1.07s/it]
100%|██████████| 32/32 [00:35<00:00,  1.11s/it]
100%|██████████| 32/32 [00:34<00:00,  1.09s/it]
```

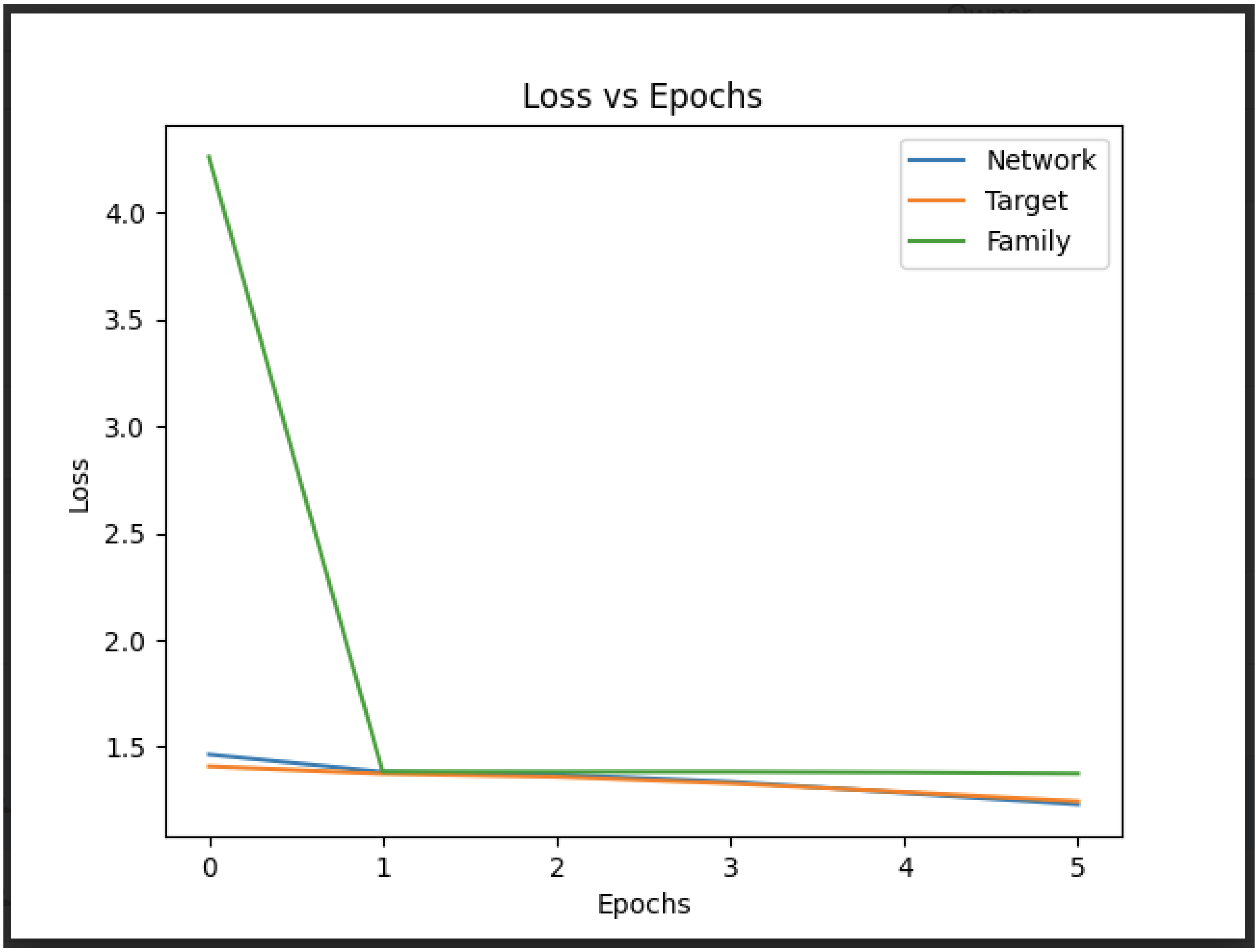
```
gdown.download('https://drive.google.com/uc?id=1-UdB2BTBC2cK51yJp0HOgloLvW3M0lGw')
gnn_graphconv_test_checkpoint = torch.load('/content/test-gnn-graphconv.pt')
num_samples = gnn_graphconv_test_checkpoint['epoch']
gnn_graphconv_test_output = gnn_graphconv_test_checkpoint['test_output']
gnn_graphconv_test_y = gnn_graphconv_test_checkpoint['test_y']
gnn_graphconv_threshold = gnn_graphconv_test_checkpoint['threshold']
gnn_graphconv_start_time = gnn_graphconv_test_checkpoint['start_time']
gnn_graphconv_end_time = gnn_graphconv_test_checkpoint['end_time']

# report standard error for uncertainty
gnn_graphconv_test_output_se = np.array(gnn_graphconv_test_output).std(axis=0) / np.sqrt(num

# take average over all samples to get expected value
gnn_graphconv_test_output = np.array(gnn_graphconv_test_output).mean(axis=0)
gnn_graphconv_test_y = np.array(gnn_graphconv_test_y).mean(axis=0)

gnn_graphconv_results = pd.DataFrame({'actual': gnn_graphconv_test_y, 'pred_raw': gnn_graphc
gnn_graphconv_results['pred_binary'] = (gnn_graphconv_results['pred_raw']>gnn_graphconv_thre
gnn_graphconv_metric_results = calculate_metrics(gnn_graphconv_results['actual'], gnn_graphc

print(gnn_graphconv_metric_results)
print("Training duration: {} minutes".format((gnn_graphconv_end_time - gnn_graphconv_start_t
# it is better to save the numbers and figures for your presentation.
```

```
Downloading...
From: https://drive.google.com/uc?id=1-UdB2BTBC2cK51yJp0HOgloLvW3M0lGw
To: /content/test-gnn-graphconv.pt
100%|██████████| 440k/440k [00:00<00:00, 5.13MB/s]
{'metric_auc_roc': 0.7561574424503961, 'metric_auc_prc': 0.2501534301041718, 'metric_f1'
Training duration: 14.372869650522867 minutes
```

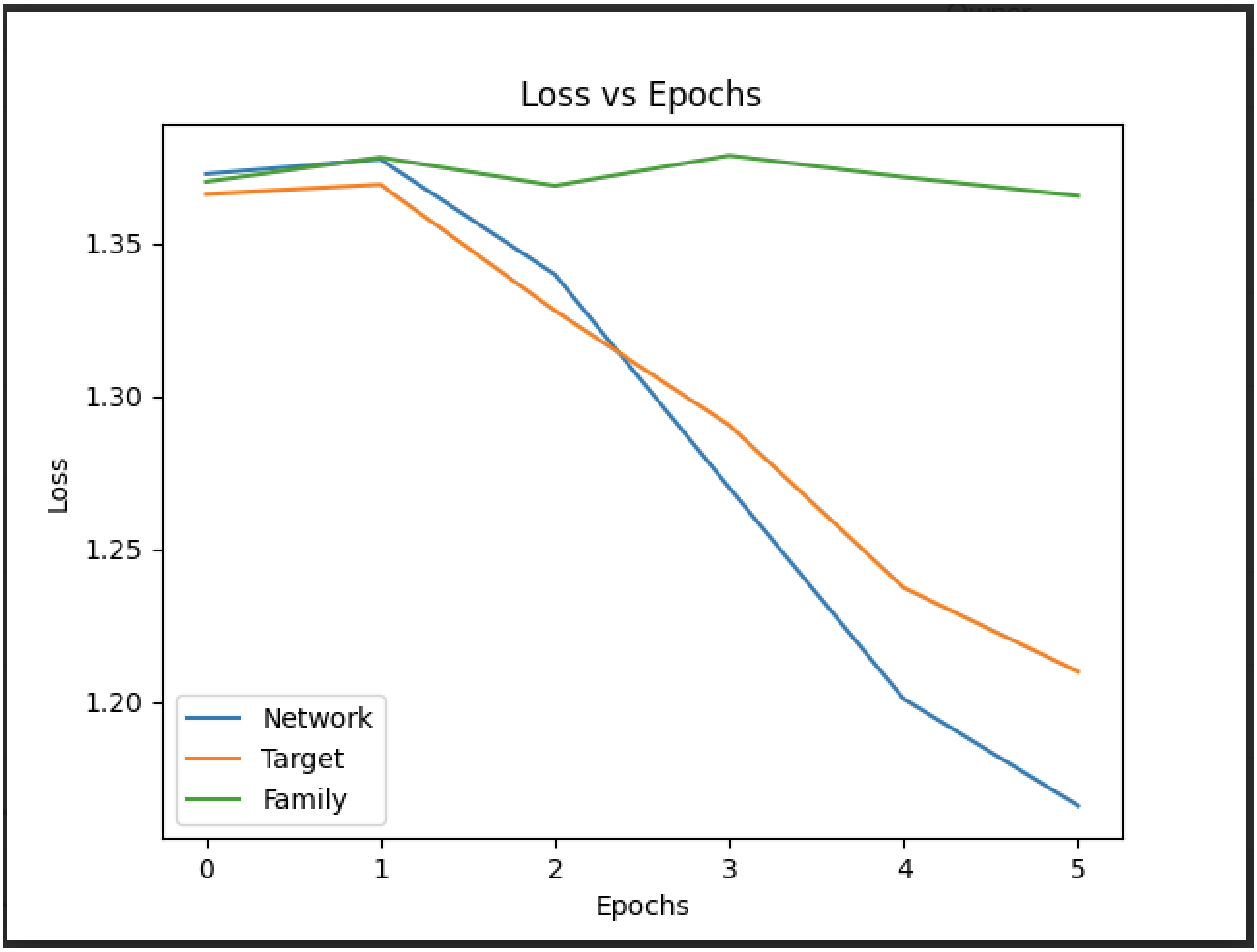
Loss:



Separate Training Loss:



Separate Validation Loss:



GCN Graph Model

The GNN model using GCN layers for non-logitudinal data.

```
# Declare data variables, loss function, and optimizer
model_type = 'graph'
gnn_layer = 'gcn'
pooling_method = 'target'
batch_size=250
main_hidden_dim = 20
lstm_hidden_dim = 20
dropout_rate = 0.5
learning_rate = 0.001
ratio = 0.5
gamma = 1
alpha = 1
beta = 1
delta = 1

fetch_data = DataFetch(model_type=model_type, featfile='featfile_G1.csv', gnn_layer=gnn_laye
train_patient_list = fetch_data.train_patient_list
validate_patient_list = fetch_data.validate_patient_list
num_features_static = len(fetch_data.static_features)
num_features_alt_static = len(fetch_data.alt_static_features)
num_samples_train_dataset = len(train_patient_list)
num_samples_valid_dataset = len(validate_patient_list)
num_samples_train_minority_class = fetch_data.num_samples_train_minority_class
num_samples_valid_minority_class = fetch_data.num_samples_valid_minority_class
num_samples_train_majority_class = fetch_data.num_samples_train_majority_class
num_samples_valid_majority_class = fetch_data.num_samples_valid_majority_class

model = GNN(num_features_static, num_features_alt_static, main_hidden_dim, gnn_layer, poolir

loss_func = WeightedBCELoss(num_samples_train_dataset, num_samples_train_minority_class, num
valid_loss_func = WeightedBCELoss(num_samples_valid_dataset, num_samples_valid_minority_clas
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

train_dataset, train_loader = get_data_and_loader(train_patient_list, fetch_data, model_type
validate_dataset, validate_loader = get_data_and_loader(validate_patient_list, fetch_data, m
```

```
Using GCN layers
```

Below is the training step used to generate checkpoint file 'gnn-gcn.pt'

you don't need to re-run all other experiments, instead, you can directly refer the metrics/numbers in the paper

```
# compare you model with others
# num_epoch = 100
# num_batches_train = int(np.ceil(len(train_patient_list)/batch_size))
# num_batches_validate = int(np.ceil(len(validate_patient_list)/batch_size))

# start_time_train = time.time()
# # model training loop: it is better to print the training/validation losses during the tra
# train_losses = []
# valid_losses = []
# separate_loss_terms = {'NN_train':[], 'target_train':[], 'family_train':[], 'lstm_train':[

# model_path = raw_data_dir + 'drive/MyDrive/CS598 DLH/Project/gnn-gcn.pt'
# gcn_early_stopping = EarlyStopping(path=model_path)
# for i in range(num_epoch):
#   model.train()
#   epoch_train_loss = []
#   separate_loss_terms_epoch = {'NN_train':[], 'target_train':[], 'family_train':[], 'lstm_

#   for train_batch in tqdm(train_loader, total=num_batches_train):
#     x_static_node, x_static_graph, y, edge_index, edge_weight, batch, target_index = trair
#     output, patient_output, family_output = model(x_static_node, x_static_graph, edge_inde

#     # combined loss that considers the additive effect of patient and family effects
#     loss_term_NN = gamma * loss_func(output, y)
#     loss_term_target = alpha * loss_func(patient_output, y)
#     loss_term_family = beta * loss_func(family_output, y)

#     separate_loss_terms_epoch['NN_train'].append(loss_term_NN.item())
#     separate_loss_terms_epoch['target_train'].append(loss_term_target.item())
#     separate_loss_terms_epoch['family_train'].append(loss_term_family.item())

#     loss = loss_term_NN + loss_term_target + loss_term_family

#     optimizer.zero_grad()
#     loss.backward()
#     optimizer.step()
#     epoch_train_loss.append(loss.item())

#   # eval on validset
#   model.eval()
#   epoch_valid_loss = []
#   valid_output = np.array([])
#   valid_y = np.array([])
#   for valid_batch in tqdm(validate_loader, total=num_batches_validate):
#     x_static_node, x_static_graph, y, edge_index, edge_weight, batch, target_index = valic
#     output, patient_output, family_output = model(x_static_node, x_static_graph, edge_inde

#     valid_output = np.concatenate((valid_output, output.reshape(-1).detach().cpu().numpy()
#     valid_y = np.concatenate((valid_y, y.reshape(-1).detach().cpu().numpy()))

#     # combined loss that considers the additive effect of patient and family effects
```

```
#     loss_term_NN = gamma * valid_loss_func(output, y)
#     loss_term_target = alpha * valid_loss_func(patient_output, y)
#     loss_term_family = beta * valid_loss_func(family_output, y)

#     separate_loss_terms_epoch['NN_train'].append(loss_term_NN.item())
#     separate_loss_terms_epoch['target_train'].append(loss_term_target.item())
#     separate_loss_terms_epoch['family_train'].append(loss_term_family.item())

#     loss = loss_term_NN + loss_term_target + loss_term_family
#     epoch_valid_loss.append(loss.item())

#   train_loss, valid_loss = np.mean(epoch_train_loss), np.mean(epoch_valid_loss)
#   train_losses.append(train_loss)
#   valid_losses.append(valid_loss)
#   for term_name in separate_loss_terms:
#     separate_loss_terms[term_name].append(np.mean(separate_loss_terms_epoch[term_name]))
#   print("epoch {}\ttrain loss : {}\tvalidate loss : {}".format(i, train_loss, valid_loss))
#   gcn_early_stopping(np.mean(epoch_valid_loss), model)
#   if gcn_early_stopping.early_stop:
#     print('Early Stopping')
#     break

# fpr, tpr, thresholds = metrics.roc_curve(valid_y, valid_output)
# gmeans = np.sqrt(tpr * (1-fpr))
# ix = np.argmax(gmeans)
# threshold = thresholds[ix]

# plot_losses(train_losses, valid_losses, 'gcn')
# plot_separate_losses(separate_loss_terms['NN_train'], separate_loss_terms['target_train'],
# plot_separate_losses(separate_loss_terms['NN_valid'], separate_loss_terms['target_valid'],

# end_time_train = time.time()

# torch.save({
#     'epoch': num_epoch,
#     'model_state_dict': model.state_dict(),
#     'optimizer_state_dict': optimizer.state_dict(),
#     'threshold': threshold,
#     'start_time': start_time_train,
#     'end_time': end_time_train
#     }, model_path)

# you don't need to re-run all other experiments, instead, you can directly refer the metric
```

```
  0%|          | 0/111 [00:00<?, ?it/s]/usr/lib/python3.10/multiprocessing/popen_fork.py
  self.pid = os.fork()
100%|██████████| 111/111 [02:15<00:00,  1.32it/s]/usr/lib/python3.10/multiprocessing/pop
  self.pid = os.fork()
100%|██████████| 111/111 [02:15<00:00,  1.22s/it]
100%|██████████| 16/16 [00:16<00:00,  1.06s/it]
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:3504: RuntimeWarning:
  return _methods._mean(a, axis=axis, dtype=dtype,
/usr/local/lib/python3.10/dist-packages/numpy/core/_methods.py:129: RuntimeWarning: inva
  ret = ret.dtype.type(ret / rcount)
epoch 0 train loss : 3.8721207519909284 validate loss : 3.759416416287422
100%|██████████| 111/111 [02:05<00:00,  1.13s/it]
100%|██████████| 16/16 [00:16<00:00,  1.05s/it]
epoch 1 train loss : 3.853707350052155  validate loss : 3.722113162279129
EarlyStopping counter: 1 out of 5
100%|██████████| 111/111 [02:04<00:00,  1.12s/it]
100%|██████████| 16/16 [00:16<00:00,  1.04s/it]
epoch 2 train loss : 3.8099506631627813 validate loss : 3.686437338590622
EarlyStopping counter: 2 out of 5
100%|██████████| 111/111 [02:01<00:00,  1.10s/it]
100%|██████████| 16/16 [00:17<00:00,  1.10s/it]
epoch 3 train loss : 3.7877691092791856 validate loss : 3.6865831464529037
EarlyStopping counter: 3 out of 5
100%|██████████| 111/111 [02:03<00:00,  1.11s/it]
100%|██████████| 16/16 [00:16<00:00,  1.05s/it]
epoch 4 train loss : 3.7570300596254365 validate loss : 3.6524819284677505
EarlyStopping counter: 4 out of 5
100%|██████████| 111/111 [02:01<00:00,  1.10s/it]
100%|██████████| 16/16 [00:17<00:00,  1.12s/it]
epoch 5 train loss : 3.734771595344887  validate loss : 3.686576634645462
EarlyStopping counter: 5 out of 5
Early Stopping
<Figure size 640x480 with 0 Axes>
```

Below is the testing step used to generate checkpoint file 'test-gnn-gcn.pt'

you don't need to re-run all other experiments, instead, you can directly refer the metrics/numbers in the paper

```
# you don't need to re-run all other experiments, instead, you can directly refer the metric
# metrics to evaluate my model
# gdown.download('https://drive.google.com/uc?id=1-3ZZGCHYJbxOHWLE6SfEHSh2mfsblqW0')
# gnn_gcn_checkpoint = torch.load('/content/gnn-gcn.pt')
# gcn_model = GNN(num_features_static, num_features_alt_static, main_hidden_dim, 'gcn', pool
# gcn_model.load_state_dict(gnn_gcn_checkpoint['model_state_dict'])
# gcn_model.eval()

# gnn_gcn_threshold = gnn_gcn_checkpoint['threshold']
# gnn_gcn_start_time = gnn_gcn_checkpoint['start_time']
# gnn_gcn_end_time = gnn_gcn_checkpoint['end_time']

# num_samples = 3
# gnn_gcn_test_output = [np.array([]) for _ in range(num_samples)]
# gnn_gcn_test_y = [np.array([]) for _ in range(num_samples)]
# representations = pd.DataFrame()

# test_patient_list = fetch_data.test_patient_list
# num_batches_test = int(np.ceil(len(test_patient_list)/batch_size))
# test_dataset, test_loader = get_data_and_loader(test_patient_list, fetch_data, model_type,

# gcn_test_model_path = raw_data_dir + 'drive/MyDrive/CS598 DLH/Project/test-gnn-gcn.pt'
# for m in gcn_model.modules():
#   if m.__class__.__name__.startswith('Dropout'):
#     m.train()

# for sample in range(num_samples):
#     for test_batch in tqdm(test_loader, total=num_batches_test):
#         x_static_node, x_static_graph, y, edge_index, edge_weight, batch, target_index = t
#         output, patient_output, family_output = gcn_model(x_static_node, x_static_graph, e
#         gnn_gcn_test_output[sample] = np.concatenate((gnn_gcn_test_output[sample], output.
#         gnn_gcn_test_y[sample] = np.concatenate((gnn_gcn_test_y[sample], y.reshape(-1).det

# torch.save({
#     'epoch': num_samples,
#     'test_output': gnn_gcn_test_output,
#     'test_y': gnn_gcn_test_y,
#     'threshold': gnn_gcn_threshold,
#     'start_time': gnn_gcn_start_time,
#     'end_time': gnn_gcn_end_time
# }, gcn_test_model_path)
```

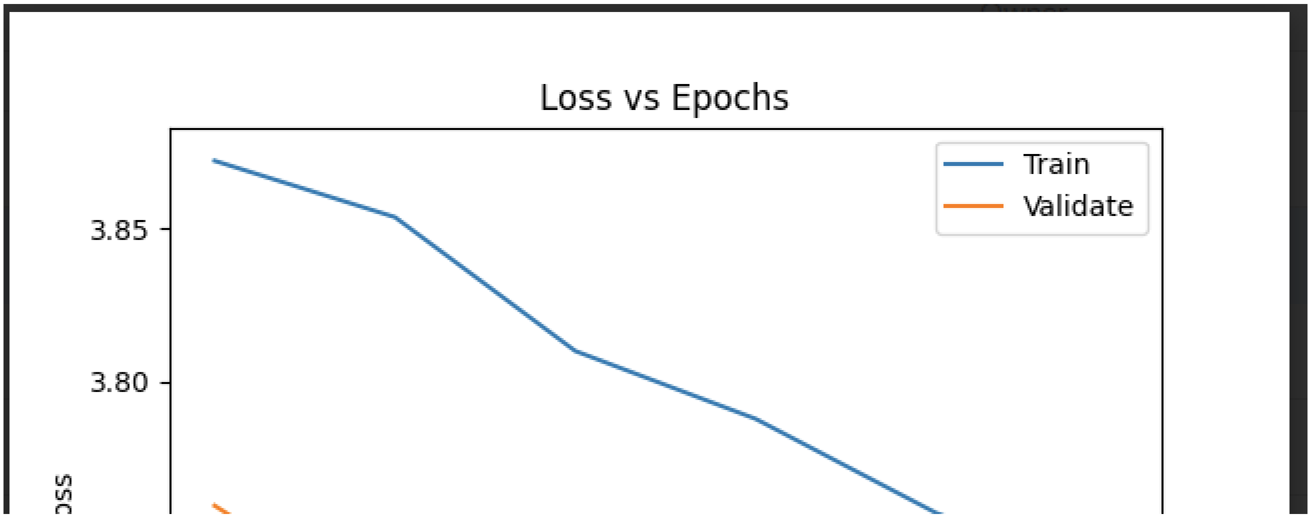
```
Downloading...
From: https://drive.google.com/uc?id=1-3ZZGCHYJbxOHWLE6SfEHSh2mfsblqW0
To: /content/gnn-gcn.pt
100%|██████████| 46.1k/46.1k [00:00<00:00, 24.9MB/s]
Using GCN layers
  0%|          | 0/32 [00:00<?, ?it/s]/usr/lib/python3.10/multiprocessing/popen_fork.py:
  self.pid = os.fork()
100%|██████████| 32/32 [00:37<00:00,  1.17it/s]/usr/lib/python3.10/multiprocessing/poper
  self.pid = os.fork()
100%|██████████| 32/32 [00:37<00:00,  1.18s/it]
```

```
100%|██████████| 32/32 [00:38<00:00,  1.20s/it]
100%|██████████| 32/32 [00:36<00:00,  1.13s/it]
```

```
gdown.download('https://drive.google.com/uc?id=1ZwhjddpQ_4raLrN8598RhykWMbSLr5u4')
gnn_gcn_test_checkpoint = torch.load('/content/test-gnn-gcn.pt')
num_samples = gnn_gcn_test_checkpoint['epoch']
gnn_gcn_test_output = gnn_gcn_test_checkpoint['test_output']
gnn_gcn_test_y = gnn_gcn_test_checkpoint['test_y']
gnn_gcn_threshold = gnn_gcn_checkpoint['threshold']
gnn_gcn_start_time = gnn_gcn_checkpoint['start_time']
gnn_gcn_end_time = gnn_gcn_checkpoint['end_time']

# report standard error for uncertainty
gnn_gcn_test_output_se = np.array(gnn_gcn_test_output).std(axis=0) / np.sqrt(num_samples)

# take average over all samples to get expected value
gnn_gcn_test_output = np.array(gnn_gcn_test_output).mean(axis=0)
gnn_gcn_test_y = np.array(gnn_gcn_test_y).mean(axis=0)

gnn_gcn_results = pd.DataFrame({'actual':gnn_gcn_test_y, 'pred_raw':gnn_gcn_test_output, 'pr
gnn_gcn_results['pred_binary'] = (gnn_gcn_results['pred_raw']>gnn_gcn_threshold).astype(int)
gnn_gcn_metric_results = calculate_metrics(gnn_gcn_results['actual'], gnn_gcn_results['pred_
print(gnn_gcn_metric_results)
print("start time: {}\tend time".format(gnn_gcn_start_time, gnn_gcn_end_time))
```

```
Downloading...
From: https://drive.google.com/uc?id=1ZwhjddpQ_4raLrN8598RhykWMbSLr5u4
To: /content/test-gnn-gcn.pt
100%|██████████| 440k/440k [00:00<00:00, 5.87MB/s]
{'metric_auc_roc': 0.7582678910645871, 'metric_auc_prc': 0.2615118219594704, 'metric_f1'
start time: 1714321995.3606708	end time
```

Loss:



Separate Loss: