AMTH/CPSC 663b - Exercise IV

Wenxin Xu

Problem 1

In this problem, you will review some of the unique parameters involved in the construction of convolutional neural networks (CNNs). Then you will train a CNN on MNIST using the code provided in p1.py and visualize a sample of the learned filters of the network.

1. Tracking the dimensionality of representations as they pass through the CNN is slightly different than for fully-connected networks. For the following, give the final output dimensions in terms of height h and width w if the size of the input image is 50x50 (hxw). Here k, s, and p refer to kernel size, stride and padding, respectively. For a) and b), assume the image is passing through a convolutional layer while for c) and d), assume a max-pooling layer. Assume a square kernel.

   (a) k=4, s=1, p=1: 49 x 49

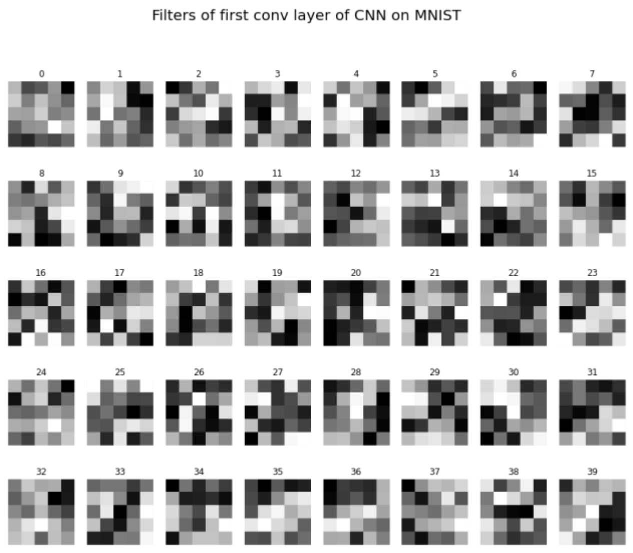
   (b) k=8, s=5, p=0: 9 x 9

   (c) k=10, s=2, p=2: 23 x 23

   (d) k=2, s=1, p=0: 49 x 49

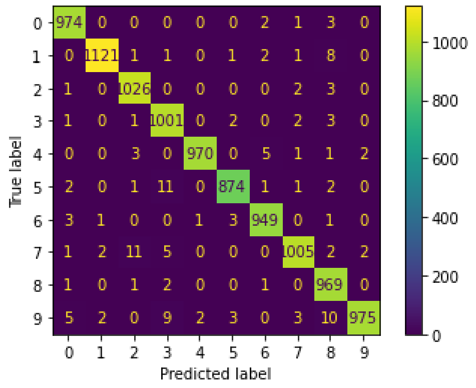
2. The following questions can be answered by looking at the provided code in p1.py

   (a) Train the model for 5 epochs. Visualize filters from the first convolutional layer (40 filters total). Include this image in your report.



(b) Produce a confusion matrix using the test split and comment on any noticeable class confusion (one class is commonly mislabeled as the other). You may use the sklearn for this. Include the plot in your report.

Comment: Among the 10,000 samples in test split, Digit 5 is misclassified as digit 3 by 11 times and digit 7 is misclassified as digit 2 by 11 times, which are the most frequent misclassifications; digit 9 is misclassified as digit 8 by 10 times and as digit 3 by 9 times. It makes sense because these digits are similar in some features.



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	974	0	0	0	0	0	2	1	3	0
1	0	1121	1	1	0	1	2	1	8	0
2	1	0	1026	0	0	0	0	2	3	0
3	1	0	1	1001	0	2	0	2	3	0
4	0	0	3	0	970	0	5	1	1	2
5	2	0	1	11	0	874	1	1	2	0
6	3	1	0	0	1	3	949	0	1	0
7	1	2	11	5	0	0	0	1005	2	2
8	1	0	1	2	0	0	1	0	969	0
9	5	2	0	9	2	3	0	3	10	975

Problem 2

1. Principal Component Analysis (PCA) is a very common method for dimensionality reduction.

(1) Conceptually, describe how the principal components in PCA are chosen.

- PCA finds directions in data that explain the most variance. An algorithm to find Principle components of PCA is to first compute the covariance matrix Σ of data X, then do eigendecomposition of that covariance matrix:

  $$\Sigma = U \Lambda U^{-1}$$

  where column vectors in the U are eigenvectors of covariance matrix, diagonal entries in the diagonal matrix Λ are corresponding eigenvalues λ, then the loadings can be found in the orthonormal matrix U. Principal components of data are UX. Any principal component is a linear combination of original features.

(2) If PCA is to implemented in PyTorch, the function torch.svd() will play an important role in the algorithm. Please briefly describewhat this function does, and why the results will be useful in implementing PCA.

- `torch.svd()` do singular value decomposition (SVD) of a real matrix. It's a generalization of eigendecomposition of a non-square matrix. Any real matrix has a SVD. a preferred method to compute PCA is via SVD rather than eigendecomposition, because SVD has more numerical stability for computing tiny singular values than EIG. Since our data is often a non-square matrix, we can directly apply SVD to our data without computing its covariance matrix.

  for data matrix X, by convention, we first flip the matrix to be X^T, then do SVD of X^T:

  $$X^T = USV^T$$

  The loadings are columns of matrix V and the principal components are US.

2. Describe the similarity between PCA and an autoencoder.

- Autoencoder can be used as a dimensionality reduction method as PCA when the dimensionality of code (bottleneck) is smaller than input. And can be used for denoising
- The latent space of Autoencoder will be similar to the embedding space of PCA if the activation function of encoder is linear.

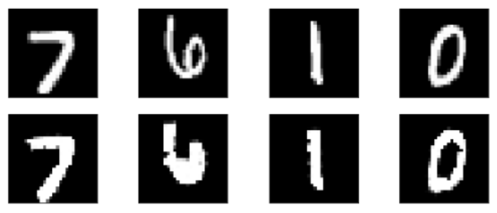
3. What is the difference between a convolutional autoencoder and linear autoencoder?

- Conv AE has convolutional layers while linear AE has linear layers.
- Conv AE is better at image tasks than linear AE because convolution operations are good at capturing spatial and local structures in the image.
- The inputs for these 2 kinds of AEs are different, the input of conv AE can be a 2D (height x width) tensor of original image shape while the input of linear AE must be a flattened 1D tensor.

Implement a convolutional autoencoder and save it as p2.py. Compare the results of your autoencoder with the original images. Include in your report both the original images and the reconstructed images (there should be 8 images in total). You may reuse parts of p1.py.

Comment: the reconstructed images are like a coarser version of original images.

CAE: Original images (Top) and reconstructed images (Bottom)



4. What similarities and differences are there between a denoising autoencoder and a variational autoencoder?

- Differences: loss function
  - Denoising AE: minimize reconstruction loss of input and reconstructed noisy input
  - VAE: reconstruction loss of input and reconstructed input and KL divergence which penalize the distribution in the latent space from being too far from the standard normal distribution.
- Similarity: both involve noise.
  - Denoising AE: add noise to corrupt the original input to get a noisy version of input to train the model together with the original input.
  - VAE: add noise sampling from standard normal distribution as input in the reparameterization trick to make the backpropagation of mean and standard deviation work.

Problem 3

1. Implement an LSTM autoencoder to learn your own word embeddings: Fill in the TODOs in rnn.py to train your own LSTM and learn word embeddings for a custom text file of your choosing.

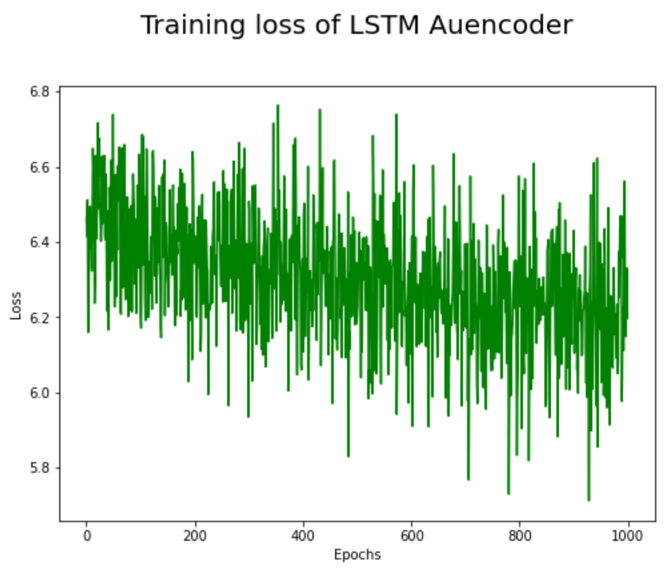
   The hyperparameters I used for the LSTM Autoencoders are: sequence length 10, embedding size 256, LSTM size 256, number of layers for both encoder and decoder 2, dropout rate for both encoder and decoder 0.5, teacher forcing rate 0.5, learning rate 0.001, Adam optimizer, batch size 128, epoch 1000.

2. Fill in the loss and backpropogation operations in rnn.py, and train your model.

   To evaluate the performance of the trained LSTM Autoencoder, I print the source sequence and reconstructed sequence pair, an examples are below. Though the training loss is decreasing but the accuracy is low.

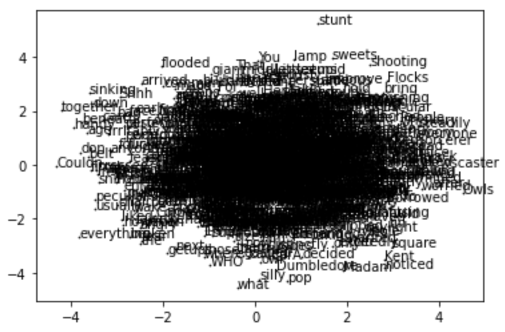
   Source seq: Yorkshire and Dundee have been phoning in to tell me

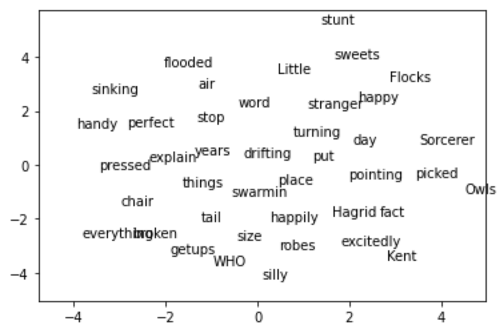
   Reconstructed seq: hands hold hissed landed lamps stumbled persuade somehow somebody some

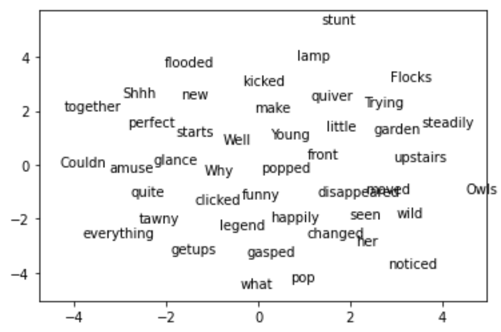


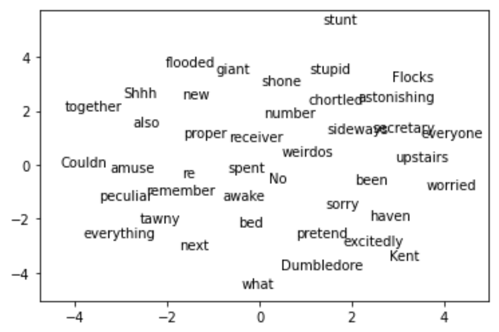
3. Plot the word embeddings with the given code. Discuss what you notice in your embeddings.

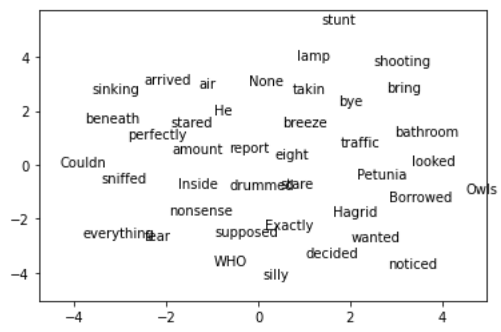
The text I used is the first chapter of book ***Harry Porter I Sorcerer's Stone*** (plot 1).. The words with similar meaning or used in the same context in the plots of word embeddings are expected to be close, but I don't find such patterns in the multiple plots I generated (plots 2-6), they are more like random words occurred together. This might because PCA isn't the most suitable dimensionality reduction method for this kind of word embedding, the words that are close in the PCA plot might be far away in the latent space.

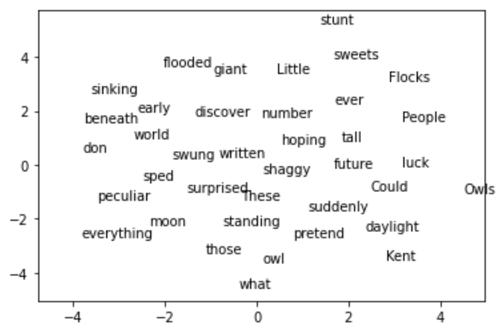

.stunt
.flooded
.You
.Lamp
.sweets
.shooting
.Flocks
.bring
.sinking
.arrived
.together
.Couldn
.peculiar
.everything
.getups
.WHO
.what
.silly
.pop
.Dumbledore
.Madam
.Kent
.noticed
.square
.decided
.Owls
.caster


stunt
flooded
sweets
Little
Flocks
sinking
air
word
stranger
happy
handy
perfect
stop
turning
day
Sorcerer
pressed
explain
years
drifting
put
things
swarmin
place
pointing
picked
Owls
chair
tail
happily
Hagrid
fact
everything
broken
size
robes
excitedly
getups
WHO
Kent
silly


stunt
lamp
flooded
kicked
Flocks
Shhh
new
quiver
Trying
together
make
perfect
little
garden
steadily
starts
Well
Young
front
upstairs
Couldn
amuse
glance
Why
popped
quite
clicked
funny
disappeared
moved
Owls
tawny
happily
seen
wild
everything
legend
changed
her
getups
gasped
noticed
what
pop


stunt
flooded
giant
shone
stupid
Flocks
Shhh
new
chortled
astonishing
together
number
also
proper
receiver
sideways
secretary
everyone
weirdos
upstairs
Couldn
amuse
re
spent
No
been
worried
peculiar
remember
awake
sorry
tawny
bed
haven
everything
pretend
excitedly
next
Kent
Dumbledore
what


stunt
lamp
shooting
arrived
air
None
sinking
takin
bring
bye
He
beneath
stared
breeze
perfectly
bathroom
traffic
amount
report
eight
looked
Couldn
Petunia
sniffed
Inside
drummed
stare
Owls
Borrowed
nonsense
Hagrid
Exactly
everything
fear
supposed
wanted
decided
WHO
noticed
silly


stunt
flooded
giant
Little
sweets
Flocks
sinking
ever
early
discover
number
People
beneath
world
hoping
tall
don
swung
written
future
luck
sped
shaggy
surprised
These
Could
Owls
peculiar
suddenly
moon
standing
daylight
everything
pretend
those
owl
Kent
what

Problem 4

1. Graph convolutional networks are modeled explicitly after classical CNNs. The graph domain, however, poses unique challenges: whereas every pixel in an image is connected to its neighbors in the same way, nodes in a graph exhibit a range of local structures that make it impossible to define a convolutional operator as a dot product with a filter.

   The spectral graph theory transforms the signal on a spatial domain to the frequency domain by graph Fourier transform of eigenvectors in the graph Laplacian matrix. The convolution is defined as the product of transformed signal and transformed filter which can be approximated by the first 2 order Chebyshev polynomials.

   How does the GCN of Kipf and Welling draw upon spectral graph theory to overcome these difficulties?

   The spectral graph theory transforms the signal on a spatial domain to the frequency domain by graph Fourier transform of eigenvectors in the graph Laplacian matrix. The convolution is defined as the product of transformed signal and transformed filter which can be approximated by the first 2 order Chebyshev polynomials.

   Are there any problems with this approach?

   The convolution kernel is isotropic because only the first 2 order polynomials are used. Therefore, this operator can't detect directional changes like CNN.

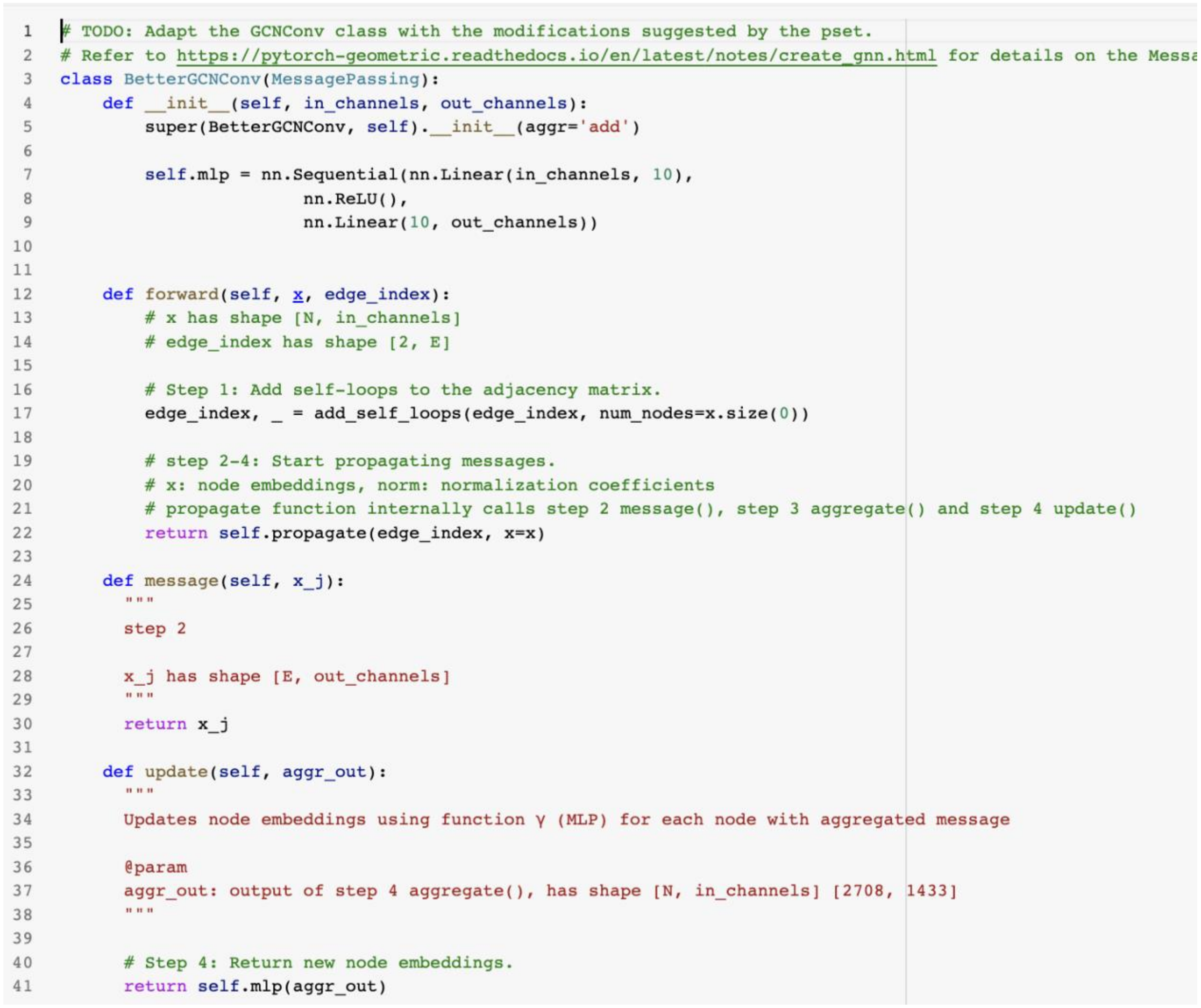
   Bonus: what alternate ways exist of defining convolutions in the graph domain?

   Graph convolution can be also defined in the spatial domain (i.e., vertex domain) as the aggregations of node representations from the node neighborhood

2. **Set up PyTorch Geometric**: Geometric is a powerful library that makes building GNNs as easy as building CNNs with PyTorch.

3. **Build a more powerful GCNConv layer:** Fill out the skeleton code in GCN.py to build a more powerful variant of Kipf and Welling' s GCN, as proposed in Xu et al' s "How Powerful Are Graph Neural Networks?" . You can follow the Pytorch Geometric tutorial on Creating Message Passing Networks, but please make these modifications to the tutorial' s baseline:

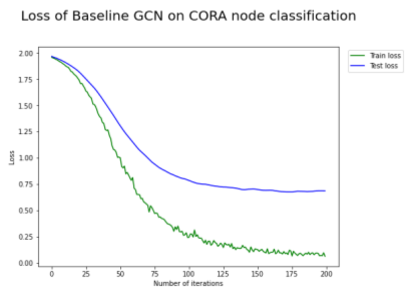
   - For γ, use a multi-layer perceptron network.
   - Do not perform any normalization in the aggregation step, φ.

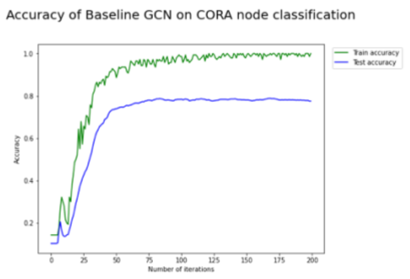


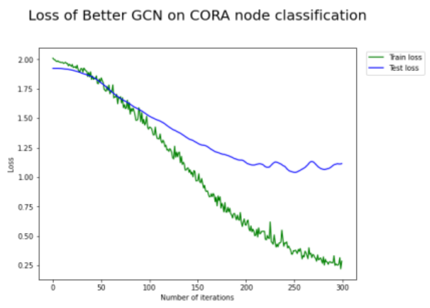
```
# TODO: Adapt the GCNConv class with the modifications suggested by the pset.
# Refer to https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html for details on the Messa
class BetterGCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super(BetterGCNConv, self).__init__(aggr='add')

        self.mlp = nn.Sequential(nn.Linear(in_channels, 10),
                       nn.ReLU(),
                       nn.Linear(10, out_channels))


    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index, _ = add_self_loops(edge_index, num_nodes=x.size(0))

        # step 2-4: Start propagating messages.
        # x: node embeddings, norm: normalization coefficients
        # propagate function internally calls step 2 message(), step 3 aggregate() and step 4 update()
        return self.propagate(edge_index, x=x)

    def message(self, x_j):
      """
      step 2

      x_j has shape [E, out_channels]
      """
      return x_j

    def update(self, aggr_out):
      """
      Updates node embeddings using function γ (MLP) for each node with aggregated message

      @param
      aggr_out: output of step 4 aggregate(), has shape [N, in_channels] [2708, 1433]
      """

      # Step 4: Return new node embeddings.
      return self.mlp(aggr_out)
```

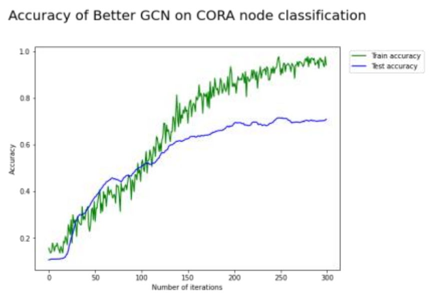
4. Node Classification on CORA graph

   The original GCN performs better than the modified GCN with test accuracy 0.7990 > 0.7330



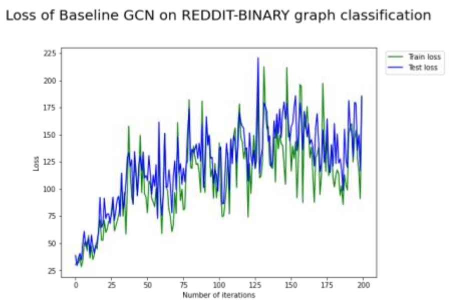


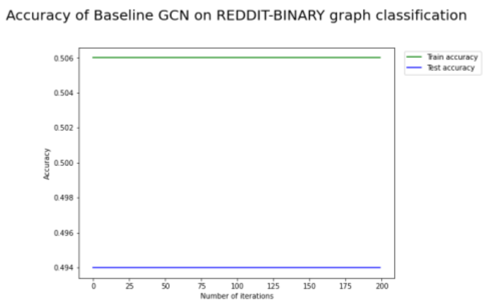


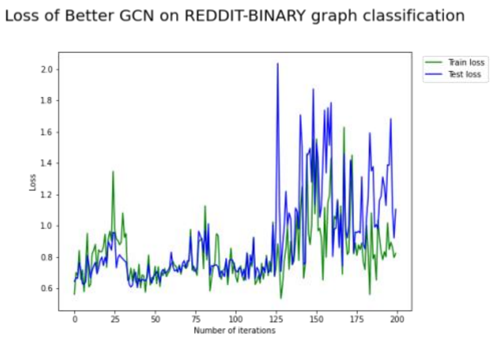


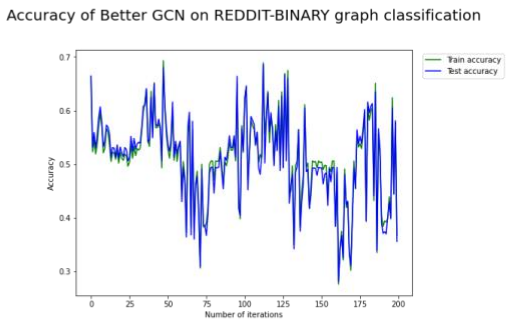
5. Graph Classification on REDDIT-BINARY dataset

   Comment: the modified model performs much better than the original model with test accuracy 0.6860 > 0.4940. The modified model is better at graph classification than node classification when compared to original model. The reason is that the vanilla GCN is built for node classification task rather than the graph classification task. For the graph classification task, the vanilla GCN is like a semi-random guessing with accuracy lower than 0.5 because the Reddit dataset doesn't contain any descriptive node features. However, a modified GCN with MLP update function can learn something meaningful about the geometry of the graph.









References

[1] “Convolutional Neural Networks Cheat sheet.” [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks#layer
[2] “Padding and Strides.” [Online]. Available: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html?fbclid=IwAR3ED3ndc6EFXU_06Ede7j0ji7HIN2-ADNRVp58FH2byUIPSVh9YX7Xhko4
[3] “PCA VS. EIG.” [Online]. Available: https://stats.stackexchange.com/questions/314046/why-does-andrew-ng-prefer-to-use-svd-and-not-eig-of-covariance-matrix-to-do-pca
[4] “Convolutional Autoencoder Solution.” [Online]. Available: https://github.com/udacity/deep-learning-v2-pytorch/blob/master/autoencoder/convolutional-autoencoder/Convolutional_Autoencoder_Solution.ipynb
[5] “Seq2Seq.” [Online]. Available: http://ethen8181.github.io/machine-learning/deep_learning/seq2seq/1_torch_seq2seq_intro.html
[6] “Install PyG.” [Online]. Available: https://gist.github.com/ameya98/b193856171d11d37ada46458f60e73e7
[7] “Graph PyG.” [Online]. Available: https://zqfang.github.io/2021-08-07-graph-pyg/