Link prediction using a Convolutional Neural Network

Bayanda Kutshwa - 15009972
Hanno Jacobs - 17000042

Research Question, Goals and Tasks

Research Question:

- Can a link prediction task be completed by leveraging a deep neural network when the only given information to predict a link are: id1, id2 (and a label to identify if there is a connection between the two or not in the training set)

Goals:

- Propose an effective and efficient solution for link prediction when the only information given are 2 node idx.

Tasks:

- The model must be trained on a given train.csv dataset.
- This model will be trained and tested with an 80/20 train/test split
- The model will be ran on the test.csv files which has no labels.
- These labels will be filled in as the model's predictions

Aim and Contribution

Aim:

- Be able to predict whether a label is connected or not given the 2 node idx

Contribution:

- Link prediction is a very useful application for communication networks, suggestions prediction on social networks, e-commerce platforms, and advertising platforms

Methodology: Why a CNN?

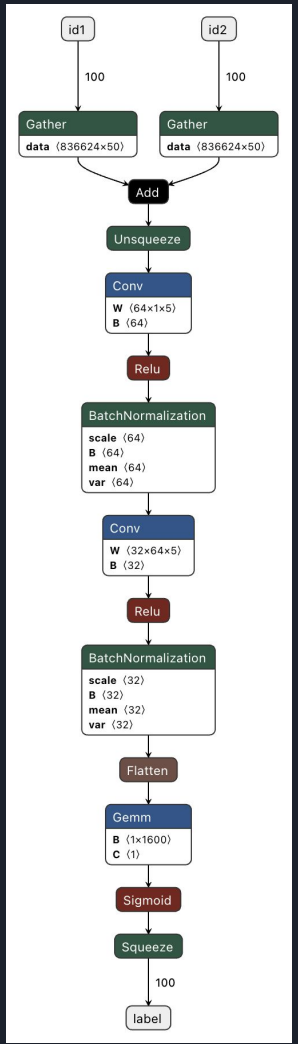
A CNN can be seen as analogous to a graph neural network.
If you take all the node numbers that a value can have and stack them in rows as if they are pixels in an image, then it can be seen that a CNN which is good for image recognition will also be good for identification tasks that require identifying connected nodes.

The link prediction task is analogous to an object detection task in images:

Nodes 0k-100k										
Nodes 100k-200k			id1							
Nodes 200k-300k										
Nodes 300k-400k										
Nodes 400k-500k										
Nodes 500k-600k					id2					
Nodes 600k-700k										
Nodes 700k-800k										
Nodes 800k-827k										

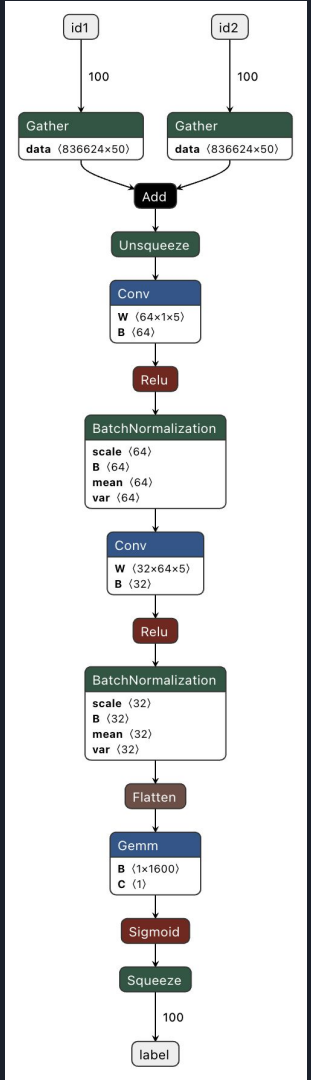
Model structure

```python
class NeuralNet(nn.Module):
    def __init__(self, input_size=input_layer_size, embedding_dim=50, dropout_rate=0.2):
        super(NeuralNet, self).__init__()

        self.embedding = nn.Embedding(num_embeddings=input_size, embedding_dim=embedding_dim)
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=64, kernel_size=5, padding=2, stride=1)
        self.bn1 = nn.BatchNorm1d(num_features=64)
        self.conv2 = nn.Conv1d(in_channels=64, out_channels=32, kernel_size=5, padding=2, stride=1)
        self.bn2 = nn.BatchNorm1d(num_features=32)
        self.seq = nn.Sequential(
            nn.Dropout(dropout_rate),
            nn.Flatten(),
            nn.Linear(in_features=(32 * embedding_dim), out_features=1),
            nn.Sigmoid(),
        )

    def forward(self, id1, id2):
        # embed the IDs to put it through the NN
        id1_embedded = self.embedding(id1)
        id2_embedded = self.embedding(id2)
        x = id1_embedded + id2_embedded

        x = x.unsqueeze(1) # Reshape for Conv1d
        x = F.relu(self.conv1(x))
        x = self.bn1(x)
        x = F.relu(self.conv2(x))
        x = self.bn2(x)
        y = self.seq(x).squeeze() # Pass through the sequential block
        return y
```



Model structure

1. The embedding layer takes the extremely tall and sparse input id1 and id2.
   a. id1 and id2 are both of dimensions 836624x1 which is the maximum possible idx
   b. These carry batches of 100 each resulting in 836624x100 each
2. These 2 are both embedded to an embedding dimension of 836624x50 to reduce the sparsity of the representation. This could be reduced by even more drastic embedding if necessary for storage purposes if larger than 100 batch sizes are used.
3. Then these 2 are added to create a fully embedded representation.
4. The embedded input is then put through a 1D Convolution layer with a Relu activation function to ensure non-linearities that the model can learn.
   a. The convolution applies 64 filters that allow it to capture the local patterns that the connected ids create.
5. The output is then batch normalised to stabilize the output of the Convolution layer and speed up training.
6. The output is then sent through another Convolution layer and batch normalisation step.
7. This output is then flattened to create a 1D tensor that can be sent through a final linear layer that creates a binary output of 1 or 0 that is a label that tells us if the two nodes are connected



Tests taken to tune the model

The optimizers and the loss functions that result in the best performance are shown in the table below. In this table it shows how the best optimizer and loss metrics that can be used for this specific model to yield the best performance are Adam with a BCE loss function.

Rank	Optimizer	Loss Metric
1	Adam	Binary Cross Entropy
2	AdamW	Cross Entropy
3	Adamax	Mean Squared Error
4	ASGD	Smooth L1 Loss
5	Adagrad	BCE Loss with Logits
6	SGD	—
7	SGD Momentum=0.9	—
8	RMSprop	—
9	Adadelta	—

Results

Metric	Training Set Results	Testing Set Results
Epochs	10	10
Optimizer	Adam	Adam
Loss Metric	BCE Loss	BCE Loss
F1 Score	1.0	1.0
Accuracy	99.9997%	99.991%
Precision	1.0	1.0
Recall	1.0	1.0
TP (True Positive)	348,659	87,164
TN (True Negative)	409,924	102,465
FP (False Positive)	0	0
FN (False Negative)	2	18
Correct Predictions	758,583	189,629
Total Predictions	758,585	189,647

Near 100% test accuracy? That can’t be right, can it?

Actually it can…

The input of the model has the number of input nodes is equal to the number of possible idx therefore it can memorise all possible combinations in the train and therefore the test can nearly always assume the solution. Isn’t that overfitting? No, the input size needs to be as big as the total number of ids. If the number of input idx is less than the number of possible nodes then it would be analogous to an image-based NN inferencing on an image and not having the information for each pixel to train on. Thereby, throwing away data that it has every right to have.

The given train.csv file in this problem is analogous to getting only 1 “image” to inference on and then only training it to learn to recognise that one image. Ideally in different link prediction task there would be "many" train.csv files that all have different networks. These can then be trained on to get a network to learn the relationships that “networks” have "in general". This is obviously unfeasible since “networks” don’t all have the same structure that can be learnt and generalised to.

Since our given problem has only one dataset (analogous to 1 image to train on) it makes sense that we would theoretically be able to obtain 1.0 precision and recall since the network should be able to just regurgitate what we already taught it in the training process.

In the general sense of applications of link prediction there is generally the use of many characteristics and not just an idx to make a prediction. If only the idx are used for prediction and another idx is added then this is analogous to just adding a “new class” that the network doesn’t know about and cannot therefore classify. However, if the link prediction network uses many characteristics for prediction and not just idx: such as close friends prediction in social networks then the network/model will already have every possible “interest” in the database therefore adding a new person to your link prediction will not add anything that is analogous to a “new classes” that the network doesn’t know how to classify as in the case of purely idx based classification models.