# Separating Perception and Reasoning via Relation Networks

# Team member: YueSheng Xu (yx2599@nyu.edu) Zehao Wang (zw2596@nyu.edu) Haoxiang Wang (hw2766@nyu.edu)

# 1. Project Overall description:

# In this project, the goal is to use relation networks to solve relational reasoning tasks. During the project, the team is going to implement and reproduce the performance of RNs on the VQA task on the Sort-of-CLEVR dataset. Based on the paper “A simple neural network module for relational reasoning”, provides a great approach to augment the CNN and MLP in relational reasoning. The author constructed a dataset similar to CLEVR that call “Sort-of-CLEVR. And the graphic inputs are distributed into two kinds, the pixel version, and the state description version. We compared the performance of RNs in non-relational questions and relational questions. We focus on fitting two types of models: 1. CNN + RN + MLP; 2. State Description + RN + MLP

# 2. Sort-of-CLEVR dataset description:

# Based on the equipment limitation and complexity of calculation. The team has decided to use the sort-of-CLEVR dataset as the input of the model, where the sort-of-CLEVR dataset only contains 10000 images and 20 questions, different from the CLEVR dataset, the original CLEVR have about 70k image and over 700k questions, which is quite a small dataset size in sort-of-CLEVR. In another aspect, the image of the original CLEVR dataset contains 3D images, but sort-of-CLEVR only contains the 2D image. The reduced dimensions are making each image size reduced, where the training time of the dataset will be highly time efficient in the model training process. Another efficient aspect of the sort-of-CLEVR dataset is the hard code of the string, which represents the 6 different colors. The hardcode part makes a direct encode method to use 0 and 1 to define whether the object belongs to a certain color, and the representation string array will mark it as an input feature to the model, rather than defined as a specific color like “red” or “yellow”, the training data size will highly reduce. At last, the sort-of-CLEVR uses a feature “type” to represent the specific question as a relational question or a non-relational question. As relational reasoning is more sensitive to the neural networking architecture, the separation of questions makes a classification of the questions, which will make the question unambiguous to identify the relation between the questions and the object itself.

# 3. Relational Network

The Relational Network is on an explicit top hierarchy of Relational Reasoning. As the input matrix of a dataset (could be graphic, linguistic, or invariant objects), relational reasoning will use CNN to get the relational reasoning behind each object, and the relational reasoning result will be stored in a matrix map to represent each object’s relational reasoning of each other. This matrix map contains common properties like pixels or dimensions of a certain object. Then the relational network will capture the matrix map from relational reasoning to build up a network with multiple-layer perceptrons based on the common part of the matrix.

For the version of pixels input, the author first used a CNN to parse pixel inputs into a set of objects. In detail, the CNN took images of size 75 × 75 and convolved them through four convolutional layers to k feature maps of size d × d, where k is the number of kernels in the ﬁnal convolutional layer. In our experiment, k is 256, and d is 5.

Then, in order to construct an object from the image features, d × d feature maps was tagged with an arbitrary coordinate indicating its relative spatial position. As a result, it can be treated as an object for the RN.

x = self.conv(img) ## x = (64 x 256 x 5 x 5)

"""g"""

mb = x.size()[0] # batch size

n\_channels = x.size()[1] # number of filters per object

d = x.size()[2] # feature map size of dxd

d \*= d

x\_flat = x.view(mb,n\_channels,d).permute(0,2,1) # x\_flat = (64 x 25 x 256)

x\_flat = torch.cat([x\_flat, self.coord\_tensor],2) # add coordinates (64 x 25 x 258)

n\_channels += 2

However, RN considers the potential relations between all object pairs. And it expects input from a set of objects before feeding into fφ and gθ MLPs. Since questions were encoded as binary strings of length 11, so we need to concatenate all pairs of objects together.

# cast all pairs against each other

x\_i = torch.unsqueeze(x\_flat, 1) # (64x1xdxn\_channels)

x\_i = x\_i.repeat(1, d, 1, 1) # (64xdxdxn\_channels)

x\_j = torch.unsqueeze(x\_flat, 2) # (64xdx1xn\_channels)

x\_j = x\_j.repeat(1, 1, d, 1) # (64xdxdxn\_channels)

# concatenate object pairs together

x\_full = torch.cat([x\_i,x\_j],3) # (64xdxdx2\*n\_channels)

# cast question and concatenate

qst = torch.unsqueeze(qst, 1) # (64x1x11)

qst = qst.repeat(1, d, 1) # (64xdx11)

qst = torch.unsqueeze(qst, 2) # (64xdx1x11)

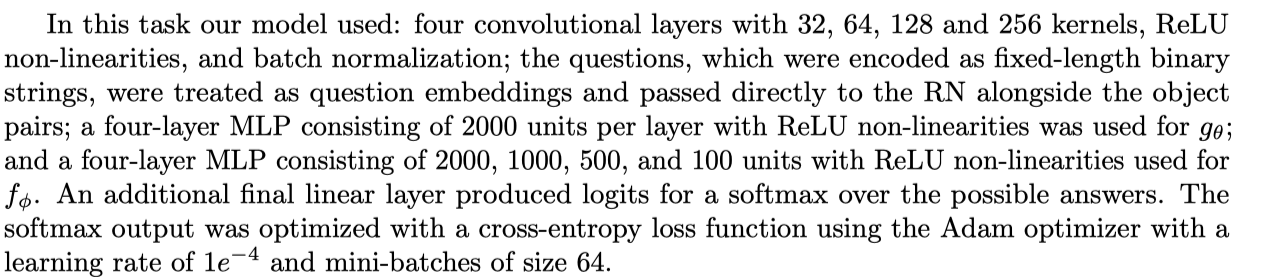
qst = qst.repeat(1,1,d,1) # (64xdxdx11)

# concatenate all together

x\_full = torch.cat([x\_full, qst], 3) # (64xdxdx2\*n\_channels+11)

The remaining steps are straightforward. We reshape the tensor for passing through f and g MLPs.

Following the experiment set in the paper, we modify the CNN layers, f & g MLPs, and loss function.



Finally, we conducted experiments multiple times. The training and testing performance results are shown in the .csv file.

# 4. State Description Version model

# In the paper, the author proposed 2 different versions of the input image. One is from the original pixel, where use CNN parses pixel inputs into a set of objects. Another one is dealing with state descriptions, since state descriptions are pre-factored object representations, they are input directly into the RN. In order to represent the information of the image. The author used state description matrices. In Sort-of-CLEVR, each image has a total of 6 objects with different colors, and each object contains a list of features - 2D coordinates (x, y); color (r, g, b); shape (square, circle). So the row of the state description matrix contains the features of a single object. And there are a total of 6 rows in the matrix.

# The following table illustrates the state description matrix.

| x | y | r | g | b | square | circle |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

The matrix has a size of 6x7. There are 5 float values and 2 int values in each row. The (x,y) is the center coordinate of the object, and following are the RGB values of color, and the last two values describe the shape.

Firstly, we need to modify the dataset generation process and store the state description matrix along with the image generation.

In sort\_of\_clevr\_generator.py, initialize and assign the value of an attribute of the object to the state matrix in build\_dataset(). Then save the data in the original format.

state\_column\_size = 7

state\_color\_idx = 2

state\_shape\_idx = 5

# coordinate

center = center\_generate(objects)

state[0] = center[0]

state[1] = center[1]

# color

state[state\_color\_idx:state\_color\_idx+3] = list(color)

# square

if random.random()<0.5:

state[state\_shape\_idx]=1

# circle

else:

state[state\_shape\_idx+1]=1

Next, in main.py, load the state matrix value into Tensor Variable named input\_state and forward into the train/test process.

input\_state = torch.FloatTensor(bs, 6, 7)

# Also, add an argument for model training, which helps decide which kind of version of input used to fit the model.

parser.add\_argument('--state\_desc', type=int, default=0,

help='what kind of input to learn. options: pixels or state descriptions')

# Then, we modify the original model in model.py. In RN class, add object pairs concatenation for state description matrix in forwarding (self, img, state, qst).

# state matrix input

if self.state\_desc != 0:

# x = (64 x 6 x 7)

x = state

mb = x.size()[0] # batch size

d = x.size()[1] # object counts

n\_channels = x.size()[2] # attribute counts

x\_flat = x.view(mb,n\_channels,d).permute(0,2,1)

Other steps and settings are the same as the pixels version. We also change the input shape of the first layer of g MLP in order to coordinate with object pairs.

Finally, we conducted experiments multiple times. The training and testing performance results are shown in the .csv file.

Specific Github implementation link below:

<https://github.com/yueshengxu/relational-networks>