Defining RNN/LSTM Model in Current Software Framework

For making any training-run first we have to define the model architecture, which we train the dataset on. This documentation will provide a walkthrough of an example model to familiarize the structure of model creation. This example is based on the following script in the Github Repository present at:
HAhRD/GSOC18/models/model_rnn_definition.py

Step 1: Importing the Required library and RNN,CNN modules

First of all we will import the necessary CNN,RNN/LSTM modules to make the model for training.

```
1 import tensorflow as tf
2 import sys
3 import os
4
5 #Imprting the RNN and CNN utiltites from CNN Module
6 from CNN_Module.utils.conv2d_utils import *
7 from CNN_Module.utils.rnn_utils import *
8
```

Line 6,7: We import the 2D convolution,RNN,LSTM utilities for further building the graph.

Step 2.1: Model Overview

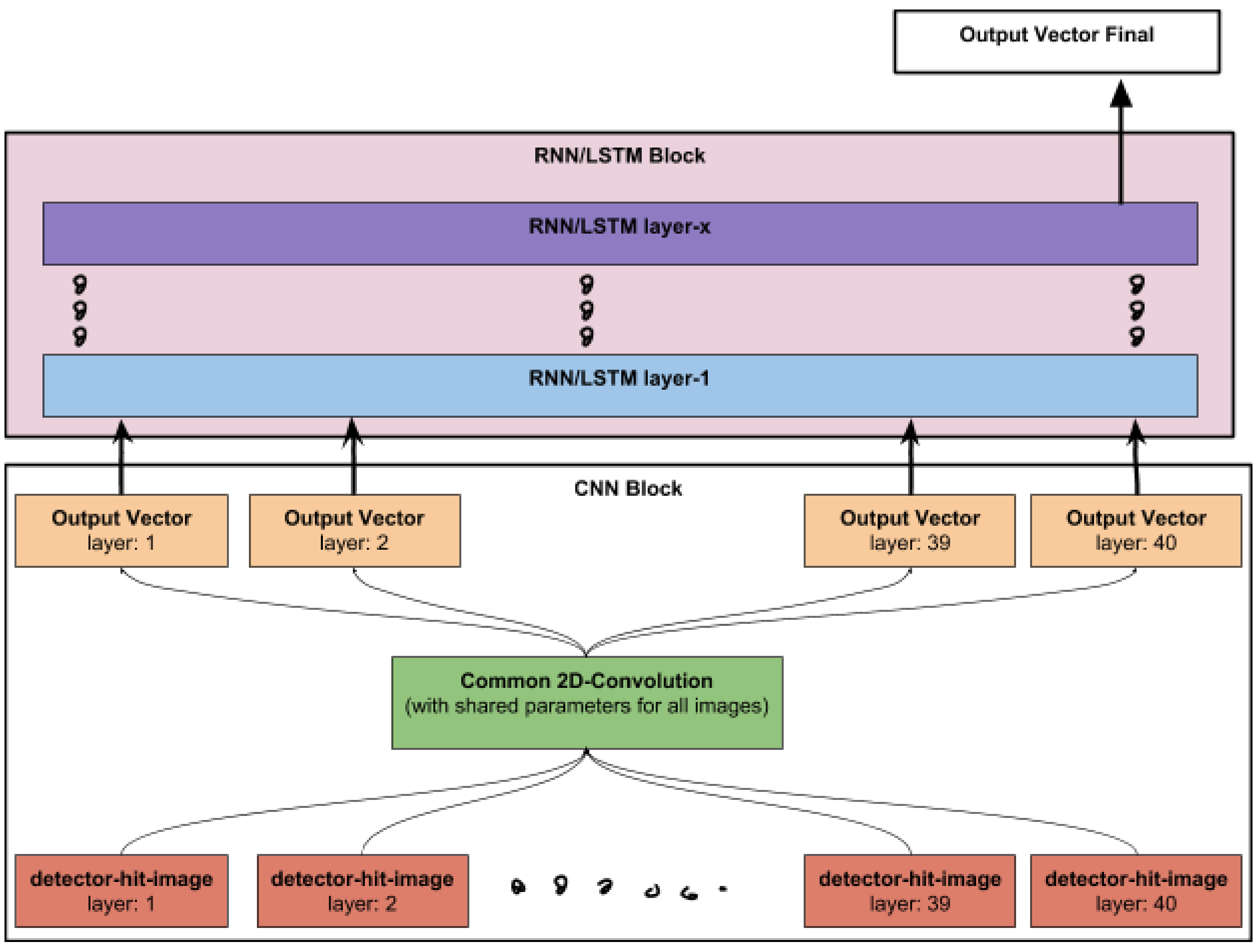
The diagram below show the overall gist of the RNN/LSTM model. The processing steps are:

1. First of all we have a common-shared convolution layer, (all the parameters shared by each layer of detector hit-image).
2. All the layers of hit-image of detector are passed through this convolutional layer which then produces a vector output (maybe ~1000 or more dimensional). This is kind of summary/information of each layer compressed by CNN Layer in a vector (termed vector encoding).
3. Now this “vector-encoding”(output vector of each layer is fed to the LSTM or RNN layer as input.
4. The RNN Module comprises of three hierarchy
    a. **RNN/LSTM cell**: the smallest unit inside the LSTM/RNN layer which will take the individual input of each of the CNN output.
    b. **RNN/LSTM layer:** this one layer of LSTM/RNN will comprise of the sequence of the RNN/LSTM cells which will carry the time information. We could compose

any number of layer, but around **2 layers are practically recommended,** to protect the model from vanishing gradient problem.

   c. **RNN/LSTM block:** This is highest in the hierarchy of this module. It simply composed of multiple RNN/LSTM layers. (as could be seen in the diagram)

5. Now this RNN block is capable of producing one vector as final output or a full sequence of 40 vector as output.
6. (Optional) Taking all the 40 vector as output (from each “time”) then concatenating them and then making final prediction has benefit that it helps in reducing the vanishing gradient problem.



Step 2.9: Defining the shared-2D convolutional model

As we can see from the above diagram,we need to define a common 2D convolutional model. All the images of detector (in each layer) will be passed through this same model with the same parameters for all the detector-images.

Here the 2D convolution function is name as **_conv2d_function_handle** with the following arguments:

1. **X_img:** the full 3D image of the detector.
2. **is_training:** this will specify whether we are in training mode or not. This is used internally by the CNN layers, so just pass it to the function as described below.
3. **iter_i,reg_loss,tensor_arra…** Please don't worry about them, they are used internally by the RNN/LSTM module. So just let them be there.

Our main input is **X_img** on which we will make all the CNN model.

```
79  def _conv2d_function_handle(X_img,is_training,iter_i,iter_end,reg_loss,tensor_array):
80      '''
81      DESCRIPTION:⋯
92      USAGE:⋯
105     '''
106     #Model Hyperparameter
107     bn_decision=False
108     lambd=0.00
109     dropout_rate=0.0
110 
111     #Running the convolution on each layers of the detector one by one
112     #Slicing the input image for getting a detector layer
113     X_layer=tf.expand_dims(X_img[:,:,:,iter_i],axis=-1,name='channel_dim')
114 
115     #Starting the model definition
116     #First the convolution
117     A1=rectified_conv2d(X_layer,
118                         name='conv2d1',
119                         filter_shape=(3,3),
120                         output_channel=10,
121                         stride=(1,1),
122                         padding_type='VALID',
123                         is_training=is_training,
124                         dropout_rate=dropout_rate,
125                         apply_batchnorm=bn_decision,
126                         weight_decay=lambd,
127                         apply_relu=True)
128     #Then maxpooling the output
129     A1Mp=max_pooling2d(A1,
130                         name='mpool1',
131                         filter_shape=(3,3),
132                         stride=(2,2),
133                         padding_type='VALID')
```

Now going line by line:

1. **bn_decision (line 107)**: this will tell the CNN whether we want to use batch normalization or not. This is particularly helpful in training of deep CNN.

2. **lambd**: This is the hyperparameter to control the L2-regularization rate. This will be multiplied to the L2-regularization cost then will be added to the final cost.
3. **Dropout_rate:** the rate of dropout in the layers of CNN. Its a number between 0 and 1.
4. **Line 133:** Then we slice the a particular layer from full 3D image using iter_i.(this iter_i is internally called later,so don't worry about it) and then we add the channel dimension (like RGB but here it will be just 1) to the image.
5. Now, we are ready for the convolution process. The shape of image at this point of code is [batch_size, resolution_x, resolution_y, 1]
6. Now using the different layer we will finally output a vector of any size (may be ~1000-10000 dimensional vector to pass it as input to the RNN/LSTM layer.)

Here in this screenshot the final layer of CNN has the output dimension of 1000 (line 238), which is a fully connected layer.

```
236     Z7=simple_fully_connected(A6Mp,
237                                name='fc1',
238                                output_dim=1000,
239                                is_training=is_training,
240                                dropout_rate=dropout_rate,
241                                apply_batchnorm=bn_decision,
242                                weight_decay=lambd,
243                                flatten_first=True,
244                                apply_relu=False)
245
246     #Setting the final activation to the layer output
247     det_layer_activation=Z7
248
249
250     #################### CUSTOM OPS (Directly copy them)############
251     #################### No need to change with models ##############
252     #adding the output encoding to the tensorarray
253     tensor_array=tensor_array.write(iter_i,det_layer_activation)
254
255     #Updating the iter_i
256     iter_i=iter_i+1
257
258     #Calculating the regularization loss of the conv2d layer for passing it
259     #out of the tf.while_loop (though doing it each time it should remain same)
260     #Retreiving the collection in current scope and geting the collection
261     reg_loss_list=tf.get_collection('all_losses',
262                             scope=tf.contrib.framework.get_name_scope())
263     #print tf.contrib.framework.get_name_scope()
264     l2_reg_loss_conv=0.0
265     if not len(reg_loss_list)==0:
266         l2_reg_loss_conv=tf.add_n(reg_loss_list,name='l2_reg_loss_conv')
267         #tf.summary.scalar('l2_reg_loss_conv',l2_reg_loss)
```

As you can see from line **250- till last of this function, you don't have to make changes**. This code is being used internally by LSTM/RNN Module to perform certain function which is not required to define a model.
I will make this simple later removing the last section of this code separate from the model. But right now this has to be included as it is.

Step 3: Defining the RNN/LSTM BLOCK:

After we are done with the the creation of the CNN function handle we may now proceed to define the RNN/LSTM block.

Remember one thing from the overview diagram of model. The RNN/LSTM layer currently supports vector/"vector_encoding" as input. So the CNN Module will give 40 separate vector to the RNN BLOCK for processing.

Now as you can see in the screenshot below the defining RNN/LSTM block is quite easy. I will walk you through step-wise:

1. This will be the model we will be finally giving as input to the **training_manager.py** script to initiate the training.
2. The model take two inputs:
    a. **X_img:** the full 3D image of the energy-hits in detector.
    b. **is_training:** this will be used internally to specify the model if we are in training mode or testing mode. (dont worry about it)
    c. **rnn_lambd** in line 291: This will be multiplied with the L2-regularization of parameters in RNN Module. So this is a hyperparameter to tune for preventing overfitting.
    d. **Now finally we will define** the whole RNN/LSTM block in one single function. This will **also take 2D convolution function handle as argument** and do the convolution **internally.**
3. **simple_vector_rnn_block**: line 295: This is the single function enough to wrap up the whole RNN/LSTM layer. It take the following arguments:
    a. **X_img:** the full 3D image of the detector
    b. **is_training:** the same training_flag
    c. **_conv2d_function_handle:** the 2D convolution function handle we had defined above.
    d. **sequence_model_type**: whether we want RNN or LSTM layer
    e. **num_of_sequence_layer:** the number of RNN/LSTM layers we want in the block. Each layer is stacked on top of other layer. (as shown in the diagram). Practically using around 2 layers are recommended.

f. **hidden_state_dim_list:** the dimension of the hidden state in the in each of the layers in the block. These state are responsible for carrying memory between different time of sequence.

g. **output_dimension_list:** the dimension of the output of each layer in the block.

h. **output_type**: whether we want to output a single **vector** as output from the **whole** RNN/LSTM block, or we want to give a **sequence** of vector as output from each of the time. (give ‘sequence”/”vector”)

i. **output_norm_list:** the type of normalization we want to apply to the output of each of the layer. (currently: “relu”/”tanh”/None supported)

j. **num_detector_laye**r: the number of layers in detector. (currently 40)

k. **weight_decay:** the weight decay parameter for the L2-Regularization in the RNN block.

```
def model8(X_img,is_training):
    '''
    DESCRIPTION:
        In this model we will test the performance of the RNN module
        on the detector-hits.
    USAGE:
        INPUT:
            X_img       : the complete 3d hit-image of the detector
            is_training : the training flag which will be used by the
                            batchnorm and dropout layers
        OUTPUT:
            Z_out       : the final unnormalized output of the model
    '''
    #Rnn Hyperparameters
    rnn_lambd=0.0

    #Now invoking the RNN block to create create the RNN layers along
    #with the required convolution using the CNN function handle
    Z_list=simple_vector_RNN_block(X_img,
                                    is_training,
                                    _conv2d_function_handle,
                                    sequence_model_type='LSTM',
                                    num_of_sequence_layers=1,
                                    hidden_state_dim_list=[1000],
                                    output_dimension_list=[6],
                                    output_type='vector',
                                    output_norm_list=[None],
                                    num_detector_layers=40,
                                    weight_decay=rnn_lambd)

    #Returning the unnormalized output of the whole model
    Z_out=Z_list[0]
    return Z_out
```

Currently, there is no support for the batch normalization and dropout in the RNN/LSTM block, but will be implemented later after proper literature survey.