Experience Log

In this document, I will list out the failure, problems and other challenges faced while development of the whole Machinery. These include both the software/tensorflow specific and CNN specific problems and challenges.All the events will be grouped according to different modules present in the Software.

---

1. Interpolation Module

Problem 1:(Multiprocessing) The generation of the interpolation coefficient was a very time consuming step since it required reading the large geometry file from the hard disk from memory for each of the 40 layers of the detector. This IO time was consuming a huge percentage of time of whole transformation operation.

Solution: The simplest solution was to use the large number of cpu cores of the llr grid machine for doing the computation parallely. So, a multiprocessing approach was used where for each of the layers of detector a separate process was run in parallel. (Not exactly all at once since this will lead to heavy strain on cpu, at a time ~ncpu/2 parallel process ran).

This reduced the overall interpolation time by at least 5 times.

---

2. CNN Module

Problem 1: (tfrecords Size)

The very first problem was creation of dataset. I was using the recently added tf.data api which is very flexible and optimized for high IO performance instead of using the old feed-dictionary way of feeding the data to the model (which is very slow). The problem came because this api requires the data to be either present in the memory or we have to store them on the hard disk as tfrecords format.

But naively writing data to tfrecords format doubled and sometime quadrupled the data four-fold. So, it would have required me terabytes to store the whole dataset.

Solution:

Thanks to the very active community of Tensorflow user on both GitHub and StackOverflow where I got the way to compress the data using ZLIB in the tfrecords itself. Also, tf.data made it easy to read,decompress and deserialize the data and then finally feed it to the models.

Problem 2: (tf.data IO+transformation speed)

Since tf.data is relatively new, released in this year tensorflow 1.8, not a lot of practical guide are readily available (apart from the official documentation and Tensorflow summit videos.) Also, I was new to this api which added to the problem even more. I was getting a huge input bottleneck while training the model. This was because the GPU was waiting most of time of computation cycle to get the data from the CPU for training.

Solution:

Tensorflow tf.data has lot of open variable to tune this IO performance and a special Input performance Guideline which helps to do so. Some of them include

1. Parallel reads
2. Parallel data transformation
3. Mapping and batching together
4. Prefetching for Software Pipelining.

So, it required a lot of fine tuning of those parameter to have decent data input throughput that GPU is busy more more amount of time than wasting waiting for input. Also, making the chrome trace also helped a lot to see the actual bottle neck in the computational performance.

(Still the performance is not 100 % upto the mark)

Problem 3: (No training on initial models):

During the initial phase of the training, I was not able to come up with the CNN model architecture which reduced the error in the Energy Regression and other Positional Regression.

Solution:

So, at that point of time I had few choices:

1. **Small Filter size in z-dimension:** This would have lead to a high computational cost since out input-hit image is very high resolution (each image being 514x513x40), but this could have lead to the better extraction of features like torsion,curvature etc of the shower in the z-direction.
2. **Using the "global" filter in z-dimension:** We we were also doing the energy prediction, seeing all the layers would have been a import strategy for improvement in the energy prediction. So, after trail of this model, my hypothesis was stated true with an improvement in prediction of energy from **5-10 % error to ~3% error.**
3. **Experiment on both local-global filter:** Taking inspiration from both the above ideas and the architecture of Inception Net, I tried to use both the filters simultaneously in one layer, but there was no significant improvement in the accuracy of the model.

Even after the second idea, there was huge error in positional regression (Eta and Phi initially and later in barycenter position x/y/z). And this was mystery till long time which I found the problem later.

Problem 4: (Problem of zside in detector image):

Training/Regression was not happening in the positional variables of the target like posx,posy of the barycenter.

Solution:

The HGCAL detector has two separate zside(two different side of collision point), comprising of one at z>0 and other z<0 with respect to the collision point. Now, since we had found the interpolation coefficient for one of the zside and using it to interpolate both the zside, there was reflection of the position values for the other zside.

This was the reason that training was not happening properly. So verify this, I separated the images from the difference zside and then ran the training. As expected, in one of the zside the error reduces to all time lowest and also, in the other zside no training was observed.

This failure was important in sense that, this showed that CNN was not just **"memorizing"** the data but actually looking for the evidence in the data to make prediction. So, since the position was wrong so it could not find the location of hits also, hence the prediction of energy was also wrong.

This problem was most difficult to find.But the advice of Gilles Sir (one of my mentor) and forcefully moving me away from computer by my Dad, gave me time to think clearly about the problem and it suddenly clicked to me.

Thanks for the **"Sunday scheduling"** advice.

Problem 5: (**Batch normalization not working !!**)

Batch normalization is a powerful algorithm for improving the learning of the Deep Neural Networks. But it was surprising that it was not learning at all in my case, let alone improve the performance.

Solution:

Well, again playing with the hyperparameters helped a lot. There were two direction of thought to explain this in my mind, one was right and one was wrong(not sure still).

1. Either batch Normalization could not help in prediction task, since normalizing the inputs or other layer could lead to the loss of information about the energy, thus leading to a bad prediction in regression task.
2. Our learning rate had to be tuned to make this work.

So, after trying several learning rate, I found out that Batch Normalization actually "accelerates" learning and we actually need larger learning rate in that case for training to work. Thus, being completely empirical I can't explain the first reason not being right.

Problem 6: (Target Representation)

For me deciding how to represent the target for proper prediction was challenging and difficult task. Though it was rewarding in sense this made me think several direction of approach and helped me practice how to device a learning strategy for training.

Finally mentors came to rescue with a full plan of target representation and my task was just reduced to implement it. So, we need to use the domain expertise instead of any wild idea for this representation which might not have been very insightful.

---

3. Visualization Module

Problem 1: (Visualizing 3D data)

Since the energy hits by the particle in the detector is a 3D data, proper visualization platform was necessary for making us familiar with the shapes of shower in the interpolated 2D geometry from the hexagonal geometry of the detector.

Solution:

This idea was not completely mine but I got inspired by how the 3D brain scan data is visualized. So this helped me in making a animation plot of the input data which gave a good insight of the energy-hits in the detector.

Problem 2: (making inference from Saliency Maps)

I had integrated a learning visualization feature to help us understand how well CNN is learning and how is it actually learning. This is called as Saliency Map, which is a simply gradient of a particular output with respect to the input image. This help us show the region in image where the CNN is very sensitive to the changes. But this plot showed that, the end part of the detector was more sensitive than the central part where most of the hits are located. This was counter intuitive.

Solution:

After doing research form few sources, I found that Gradient * Image is a better thing to view, since it actually shows the regions which contribute to the prediction rather than just being sensitive to the prediction.

The contribution is of more importance than the sensitiveness since a particular region could be sensitive because even a small error in that region could be drastic to prediction. So it is just being more careful there. This does not show if it is actually contributing to the prediction.

3. RNN/LSTM Module

Problem 1: (Vanishing Gradient Problem)

Taking just a single output from the RNN block in the model, lead to almost no training. I was expecting that the model will benefit from the explicit representation of the time information of shower in the detector in the form of RNN sequence. So, facing this not training was surprising.

Solution:

There were two possible reason:

1. Either the whole RNN model is buggy and I have to rewrite the code.
2. Or, there is problem of vanishing gradient in the model.

Second idea was easy to check, so making a model with more connection from the RNN layer to output helped is passing of gradient by backpropagation easier and lead to almost equivalent learning like the CNN models.

But, this problem of not training still exists in the LSTM module and reason 2 in not right. So, I have to find the explanation.