Question and answers - Leigh Whitehead Lecture

The following questions were submitted through Google Form. Some may have been answered in the Q&A session already. Nevertheless, we request our lecturers to provide written answers here for the benefit of those who could not attend that session. Thank you!

Slide 17. Question about the two plots on the left. Efficiency on full MC drops noticeably for high reconstructed energy. Why is this? It is not the case for fast MC.

I think there are a number of contributing factors here. Firstly, the training sample for the CNN followed the expected energy distribution, so there are quite low statistics in the high energy tail. One could attempt to train on a flat energy distribution, but it isn't a trivial thing to do since the different types of events behave very differently as a function of energy, so we made the decision to use the beam energy distribution. Neutrino interactions are also considerably more complex at high energy, so they are just more difficult to reconstruct, and this is more of an issue for electron neutrinos and their NC background than it is for muon neutrino events that have long muon tracks. As for why this isn't the case for the fast MC, the decisions on the efficiency chosen for the fast MC was made before I joined the experiment, but it is possible that it was a bit too optimistic in that region of the parameter space.

Slide 22. How do you get to truth ? Is it through simulation?

Yes, the truth is directly obtained from the simulation. It can be a little tricky to define in places where there are overlapping energy deposits from different particles, particularly at the vertex. I'm not certain what was done in this study, but generally we would assign the truth based on the MC particle that contributed the most energy in the case that a pixel contains energy from more than one MC particle.

Slide 40. You mention that "we had to change the final layer". How do you do it ? Is it through analysis that is done before training?

This is typically an easy thing to do in modern deep learning frameworks. At some level the network layers are stored in an array, so one can simply just change the final layer from Dense(1000) to Dense(3). A couple of details for keras and Pytorch follow:

- Keras: when you load the pre-defined network using keras (whether you use transfer learning or not) then you can load it without the MLP classifier network

(include_top=False below) and then add on the Dense layer yourself.

```
# Define our input type. The models we will test have 224 x 224 x 3 input images, so let's use that for now
input_layer = keras.Input(shape=(224,224,3),dtype='float32')
feature_extractor_model = keras.applications.ResNet50(include_top=False, weights="imagenet",
input_tensor=input_layer)
```

- In PyTorch I think the network always comes with its default MLP classifier network. For the ResNet, this is actually just a single Dense layer. In this specific case the name of the final layer is “fc” so we can just redefine it:

```
model = torchvision.models.resnet18(pretrained=pretrained)
input_size = model.fc.in_features
target_size = 3 # The number of classes
model.fc = torch.nn.Linear(input_size, target_size)
```

This is done prior to training, yes, so we then train these weights at the same time as the other weights are fine-tuned on our chosen data set