Lecture Notes on

SALT

Will Lancer
will.m.lancer@gmail.com

Notes on SALT.

Contents

1 SALT tutorial

I'll be making notes here in an itemized list, as there shouldn't be deep theory behind this. These should be roughly in sequential order, but I'll make comments as well. Note that whenever you see something like wlancer, you should just replace it with whatever makes sense for your case.

Note that some of the text goes off of the page; this is because the `verbatim` environment that lets me write code-looking material forces that to occur. You can still highlight it like normal to copy and paste it.

- Fun remark: one of the talks that they recommend you look at before going through this tutorial was done by Professor Dao. It has many grumpy cat photos and memes. Apparently CMS does calibration better than us? Or something. According to slide 27 they are not making the right decision...
- You're going to have to run on a GPU shell to get anything done. This is accomplished by just changing your ssh login to `ssh -Y wlancer@lxplus-gpu.cern.ch`. You can then access all of your files like normal from here. You should check to see if your node is actually configured with GPUs by running the `nvidia-smi` command.
- You then have to copy over the files into your directory. Note that the files are 17GB, so check your EOS to see if you have enough storage (you totally should; I had almost two TB). Check your storage using this command.

  ```
  eos root://eosuser.cern.ch quota /eos/user/w/wlancer
  ```

  This will tell you how many GB you've used, and how many you have available.
- Copy over the files using

  ```
  rsync -vaP /eos/user/u/umami/tutorials/salt/2023/inputs/
  /eos/user/${USER:0:1}/$USER/training-samples
  ```

Idea 1.1 (Mounting a "SALT-ready singularity image")

Let's talk about what a **SALT-ready singularity image** is. I'll go in reverse order. An **image** is a file that stores a container's entire filesystem and metadata. It's kind of like putting an entire git repo into a file. **Singularity** is a tool that's popular on HPC (high-performance computing) systems. It reads in image files (which usually end with a `.sif` suffix; sif stands for Singularity image format), and runs them, giving you reproducible environments. SALT-ready just means that the environment that Singularity unpacks is one that you may directly run SALT on, i.e. the image contained the necessary files and dependencies for SALT.

Mounting a sif file means "unpacking" it. This is like having an envelope with ten pieces of paper in it (the sif file), and then opening said envelope (to get the `/usr`, `/bin`, `/lib`, etc. files within). It's taking the sif file, which you can't "look inside", and then basically turning

it into a directory.

- The path

```
/cvmfs/unpacked.cern.ch/gitlab-registry.cern.ch
/atlas-flavor-tagging-tools/algorithms/salt:0-3
```

  is the container holding the image tagged "0-3", which stands for the 0.3 version of SALT. Run the singularity shell command; if you get a prompt like `Singularity>`, that means you're in the shell and have succeeded. For posterity, here is the command, as you're going to have to run it every time you open a terminal.

```
singularity shell -e --env KRB5CCNAME=$KRB5CCNAME --nv --bind $PWD,/afs,/eos,/tmp,/
/cvmfs/unpacked.cern.ch/gitlab-registry.cern.ch/atlas-flavor-tagging-tools/algorith
```

1.1 Fork, clone, and install Salt

Just follow the steps they give. They are relatively straightforward.

- Fork it on Gitlab. Clone it to your computer using `git clone [link]`. Run the following commands to have the original project be upstream from yours, and to switch to the correct version of SALT.

```
cd salt
git remote add upstream ssh://git@gitlab.cern.ch:7999/atlas-flavor-tagging-tools/al
git checkout 0.3
```

- Cd into the singularity shell using the previous command. Then run the tests suite from the salt directory by running `pytest --cov=salt tests/`. This will run all of the tests in the test suite, and should take about 30 minutes.

1.2 Install logger

So, I honestly don't know why we're doing this, but they recommended it. Just follow the hints they give and you'll be fine. You should add the key and project name to your bashrc using the following commands:

```
echo 'export COMET_API_KEY="yourAPIkey"' >> ~/.bashrc
echo 'export COMET_WORKSPACE="yourCometWorkspace"' >> ~/.bashrc
```

1.3 Train a subject-based tagger (finally!)

Ok, now time to actually train something!

- If you copied the files over earlier like they recommended, then they should be in your eos under the training-samples directory. My lines look like:

```
train_file: /eos/user/w/wlancer/training-samples/pp_output_train.h5
val_file: /eos/user/w/wlancer/training-samples/pp_output_val.h5
norm_dict: /eos/user/w/wlancer/training-samples/norm_dict.yaml
class_dict: /eos/user/w/wlancer/training-samples/class_dict.yaml
```

- We want to train now. Note that you should be in the Singularity shell before you fit your model. The first thing you should do is increase your number of **workers**. Not really sure what these are, but the name is suggestive. Run

  ```
  cat /proc/cpuinfo | awk '/^processor/{print $3}' | tail -1
  ```

  to figure out how many you have, and then do that number or maybe a bit below it. Your system will warn you if you are using too many. More workers = faster.

- You should also change the max number of epochs to 10.

- **(we can somehow improve speed using HTCondor. will figure out later)**

- You can now run the training test using

  ```
  salt fit --config configs/SubjetXbb.yaml --trainer.fast_dev_run 2
  ```

  If this works out fine (it should run for a single epoch), then run

  ```
  salt fit --config configs/SubjetXbb.yaml --data.move_files_temp /tmp
  ```

  to train it fully. Note that you really should use the `--data.move_files_temp /tmp` addition. This copys the eos files onto your local drive temporarily, and significantly speeds up your training according to the documentation. This is because SALT doesn't have to query the eos each time it wants a file, it can just access them from your local directory.

- You may find your results in the `logs` directory.

1.4 Track-based taggers

Now we train a track-based tagger.

- Do the same thing you did for the subjet-based tagger, i.e. change the file path names, but now in the `GN2X.yaml` file.

- Note: just like last time, make sure to change the number of workers/epochs; the defaults are likely not what you want.

- The command you run to train should be

```
salt fit --config salt/configs/GN2X.yaml --data.move_files_temp /tmp --trainer.devi
```

- You also have to comment out the `track_origin` and `track_vertexing` tasks to avoid errors.
- I kept running into memory issues when I did this, and to fix it, I reduced the number of workers and the batch size. I found that reducing the number of workers frees more memory than reducing the batch size, e.g. 10 workers at 200 batch size is more memory efficient than 20 workers at 100 batch size.
- This training took way longer for me, around 20 minutes per epoch. So about three hours.
- You can find your results like last time.

1.5 Tinkering with stuff

I would just look at the solution in the tutorial after trying some stuff out. This is an exploratory phase that can last as long as you feel like.

1.6 Evaluation

- In general, you need to train and then evaluate your model on validation data. This is that second part.
- You just need to run

  ```
  salt test --config logs/<timestamp>/config.yaml --data.test_file path/to/pp_output_
  ```

  This uses the test file to run on the model you trained. You will need to change the paths to route to your files, so for example

  ```
  salt test --config salt/logs/SubjetXbb_20250711-T221705/config.yaml --data.test_fil
  ```

- **(you can find this data where?)**

1.7 Making plots

- Copy the plotting script into your working directory

  ```
  cp /eos/home-u/umami/tutorials/salt/2023/make_plots.py .
  ```

- Specify the script to our stuff, i.e.

```
networks = {
    "SubjetXbb": "salt/logs/SubjetXbb_20250711-T221705_test_pp_output_test.h5/pp_ou
    "GN2X": "salt/logs/GN2X_20250722-T220855_test_pp_output_test.h5/pp_output_test.
}

reference = "SubjetXbb"
test_path = '/eos/user/w/wlancer/training-samples/pp_output_test.h5'
```

- Use `python make_plots.py` to run the script.
- You will get the discriminant and ROC information from the plots that you can then use to evaluate the models.