SALT on SLAC SDF

This page gives some basic instructions to running the SALT Tutorial on SDF. The Salt framework is a framework for training ML-based flavor tagging algorithms in ATLAS. The tutorial page gives instructions for downloading the Salt package from gitlab (note that the tutorial uses Tag 0.1).

The Input Samples

The input files have been copied over from CERN's eos to Rachel's gpfs data area here: `/gpfs/slac/atlas/fs1/d/rhyneman/salt_tutorial/`

Submitting Training Jobs with Slurm

DO NOT TRAIN ON INTERACTIVE SDF MACHINES.

Edit whichever configuration file (i.e. `SubjetXbb.yaml`) with the correct paths for the training data on SDF:

```
train_file: /gpfs/slac/atlas/fs1/d/rhyneman/salt_tutorial/Xbb-hybrid-resampled_scaled_shuffled.h5
val_file: /gpfs/slac/atlas/fs1/d/rhyneman/salt_tutorial/Xbb-hybrid-validation-resampled_scaled_shuffled.h5
scale_dict: /gpfs/slac/atlas/fs1/d/rhyneman/salt_tutorial//Xbb-scale_dict.json
```

SLAC currently has limited GPU resources. You may want to only train on a single GPU. To do so, edit the `base.yaml` config file, setting the following: `devices: 1` (under the `trainer` block).

To train, you'll need to submit a batch job to slurm. This requires a .sh script, such as the one below (edited from the `submit_slurm.sh` script in the Salt repository):

```
#!/bin/bash

# Job name
#SBATCH --job-name=salt

# choose the GPU queue
# #SBATCH -p atlas
#SBATCH -p shared

# requesting one node
#SBATCH --nodes=1
#SBATCH --exclusive

# keep environment variables
#SBATCH --export=ALL

# requesting 4 V100 GPU
# (remove the "v100:" if you don't care what GPU)
# #SBATCH --gres=gpu:a100:4
#SBATCH --gpus=1

# note! this needs to match --trainer.devices!
#SBATCH --ntasks-per-node=1

# number of cpus per task
# useful if you don't have exclusive access to the node
# #SBATCH --cpus-per-task=10

# request enough memory
#SBATCH --mem=200G

# Change log names; %j gives job id, %x gives job name
#SBATCH --output=/sdf/home/r/rhyneman/salt_tutorial/salt/salt/out/slurm-%j.%x.out
#SBATCH --error=/sdf/home/r/rhyneman/salt_tutorial/salt/salt/out/slurm-%j.%x.err

# Comet
export COMET_API_KEY=MYKEY
export COMET_WORKSPACE=rhyneman
export COMET_PROJECT_NAME=salt-tutorial

# speedup
export OMP_NUM_THREADS=1

echo "CPU count: $(cat /proc/cpuinfo | awk '/^processor/{print $3}' | tail -1)"

echo "Current node: ${SLURMD_NODENAME}"

# move to workdir
cd /sdf/home/r/rhyneman/salt_tutorial/salt/salt/
echo "Moved dir, now in: ${PWD}"

# activate environment
source /sdf/home/r/rhyneman/miniconda3/etc/profile.d/conda.sh
conda activate salt
echo "Activated environment ${CONDA_DEFAULT_ENV}"
echo "CUDA_VISIBLE_DEVICES: $CUDA_VISIBLE_DEVICES"

# run the training
echo "Running training script..."
srun salt fit \
    --config configs/GN1Xbb.yaml \
    --data.num_jets_train 10000 \   # This is a very small number, just for testing! Change me :)
```

Note that you should change the Comet settings based on your own Comet account info. Also, I used "salt-tutorial" (instead of just "salt") as my project name; I also changed the `project_name` in the `logger` block of the `base.yaml` config file accordingly. You should change the output and error directories to your own spaces (`#SBATCH --output=...` and `#SBATCH --error=...`). You also need to use the "salt" directory of your own Salt installation for the work directory (the `cd` command). Lastly, you may or may not want to use your own miniconda installation (though Rachel's should work). If you do, change the path under the "`activate environment`" comment.

If you want to run with more than one GPU, make sure to edit the `#SBATCH --gpus=N` line (for N GPUs). I believe you also should edit the `#SBATCH --ntasks-per-node=N` option in the above script, as well as the `devices: 1` option from the `base.yaml` config file (in the `trainers` section).

With the above script edited as needed, you should be able to submit by doing:

```
sbatch submit_slurm.sh
```

Submitting Testing Jobs with Slurm

Testing on slurm is essentially the same as training, but with a few key changes needed to the above submission script, as shown below:

```
#!/bin/bash

# Job name
#SBATCH --job-name=salt_GN1Xbb

# choose the GPU queue
# #SBATCH -p atlas
#SBATCH -p shared

# requesting one node
#SBATCH --nodes=1
#SBATCH --exclusive

# keep environment variables
#SBATCH --export=ALL

# requesting 4 V100 GPU
# (remove the "v100:" if you don't care what GPU)
# #SBATCH --gres=gpu:a100:4
#SBATCH --gpus=1     ### NOTE! The training method in Salt can only use one GPU!

# note! this needs to match --trainer.devices!
#SBATCH --ntasks-per-node=1     ### NOTE! This needs to be set to 1, since we can only use one GPU!

# number of cpus per task
# useful if you don't have exclusive access to the node
# #SBATCH --cpus-per-task=10

# request enough memory
#SBATCH --mem=200G

# Change log names; %j gives job id, %x gives job name
#SBATCH --output=/sdf/home/r/rhyneman/salt_tutorial/salt/salt/out/slurm-%j.%x.out
#SBATCH --error=/sdf/home/r/rhyneman/salt_tutorial/salt/salt/out/slurm-%j.%x.err

# Comet
export COMET_API_KEY=5k4oCPRq8rFcxn5BShm83X0Fn
export COMET_WORKSPACE=rhyneman
export COMET_PROJECT_NAME=salt-tutorial

# speedup
export OMP_NUM_THREADS=1

echo "CPU count: $(cat /proc/cpuinfo | awk '/^processor/{print $3}' | tail -1)"

echo "Current node: ${SLURMD_NODENAME}"

# move to workdir
cd /sdf/home/r/rhyneman/salt_tutorial/salt/salt/
echo "Moved dir, now in: ${PWD}"

# activate environment
source /sdf/home/r/rhyneman/miniconda3/etc/profile.d/conda.sh
conda activate salt
echo "Activated environment ${CONDA_DEFAULT_ENV}"
echo "CUDA_VISIBLE_DEVICES: $CUDA_VISIBLE_DEVICES"

# run the testing
echo "Running testing script..."
srun salt test \
        --config logs/MYMODEL/config.yaml \     # Replace MYMODEL with your model name!
        --data.test_file /gpfs/slac/atlas/fs1/d/rhyneman/salt_tutorial/inclusive_testing_Mix.h5 \
        --data.num_jets_test 1000 \      # This is a very small number, just for testing! Change me :)
          --trainer.devices 1 \
```

The big changes to the Slrum options of the script are to set #SBATCH --gpus=1 and #SBATCH --ntasks-per-node=1 (these both must be set to 1 to reflect the fact that Salt can only use 1 GPU in testing, unlike in the training loop). The other change is (of course) to replace the "train" command with a "test" command (the `srun salt test` part). The `--trainer.devices 1` option in the salt command is there for the same reason.

The testing loop requires you to point to the config file in the `logs` area (the output area, which is a subdirectory of the `salt/salt` directory). This should look something like "GN1Xbb" or "SubjetXbb", followed by a set of numbers, like: 20230216-T200512. The first part gives the name of the algorithm model you were using, while the second is a time and date stamp. Make sure to replace "MYMODEL" in the above testing script with whichever model subdirectory you want to run the testing loop with. The testing loop also requires you to point to the training dataset (specify with the `--data.test_file` option). Feel free to keep using the one stored in Rachel's GPFS area.

Once the testing loop is complete, you can find the new, output .h5 file in the logs/MYMODEL/ckpts/ subdirectory. The above script allows salt to just use whichever checkpoint model in the training which has the lowest validation loss, which may or may not be that of the final epoch from the training. You can also specify which checkpoint to use by using the `--ckpt_path` argument. Regardless, the output .h5 file will be named something like "epoch=XXX-val_loss=XXX__test_Mix.h5".

Finally, to plot, just use the plotting script from the tutorial as usual (no need to run this on Slurm!).

Additional Notes for Environments

I (Rachel) am using a Singularity image for non-SALT studies, made from Ines Ochoa's Docker container (here). This is a big image, and compiling it on SDF causes a crash due to insufficient `tmp` directory space. The solution was to specify the `SINGULARITY_TMPDIR` when building the image, as follows (on SDF):

```
[rhyneman@sdf-login02 ~]$ export SINGULARITY_TMPDIR=/sdf/group/atlas/g/XbbXccTrainingData/singularity/
[rhyneman@sdf-login02 ~]$ singularity pull --dir /sdf/group/atlas/g/XbbXccTrainingData/singularity/ --disable-cache vertexing.sif docker://miaochoa/vertexing
```

To use this container in an interactive SDF shell, one can just do: `singularity shell /sdf/group/atlas/g/XbbXccTrainingData/singularity/vertexing.sif`

One can also use this container in an SDF Jupyter notebook/shell by calling this Singularity image on creation. Just put the following commands in the "Commands to initate Jupyter" box:

```
export SINGULARITY_IMAGE_PATH=/sdf/group/atlas/g/XbbXccTrainingData/singularity/vertexing.sif

function jupyter() { singularity exec --nv -B /sdf,/gpfs,/scratch,/lscratch ${SINGULARITY_IMAGE_PATH} jupyter $@; }
```

NOTE: For some reason, the interactive session will default to a different python than what is contained in the Singularity image. You can call the correct python from: `/opt/conda/bin/python` .

In order to install extra packages in Singularity, one can just pip these directly while in that singularity container. For example:

```
/opt/conda/bin/pip install atlas-ftag-tools
```

Note that for the plotting script (in progress, based on the SALT tutorial), one needs to pip-install: `puma-hep` and `atlas-ftag-tools`.