This document is meant to give an overview of running the Jefferson Lab Data Science Group's A(i)DAPT software stack. This was developed as a modification to the Jupyter Notebook framework originally built to run CLAS simulations with Generative Adversarial Networks (GANs). This new implementation allows users to run the full workflow in a streamlined and relatively simple manner. This documentation should be understood to be supplementary to the work already done by the A(i)DAPT group. More detailed descriptions of the basic concepts for the project can be found in Ref. [1].

This framework was designed for the specific reaction $\gamma p \to \pi^+\pi^- p$ for the CLAS g11 experiment. Therefore, the following documentation is directly relevant for this example. For others reactions, further work will be needed to address the specifics of that case. A more generic framework, in which an arbitrary reaction can be specified by the user, is being discussed for development by the Data Science Group. For the time being, it is recommended that anyone interested in employing this software for reactions other the one studied, consult the A(i)DAPT Group and/or the JLab Data Science Group for further instructions and/or suggestions.

1 Getting Started

To begin, access the GitHub repository at https://github.com/JeffersonLab/jlab_datascience_exp_hall/tree/main/Hall_B/AIDAPT. To set up the full framework, the user will need to copy and install two repositories. Once in the desired location to build the software stack, clone and install the core data science repo:

```
> git clone https://github.com/JeffersonLab/jlab_datascience_core.git
> cd jlab_datascience_core/
> pip install -e .
```

Then, return to the top directory and clone and install the A(i)DAPT software:

```
> cd ../
> git clone https://github.com/JeffersonLab/jlab_datascience_exp_hall.git
> cd jlab_datascience_exp_hall/
> pip install -e . --no-deps
```

NOTE: Use the "–no-deps" option to avoid uneccessary installation of library dependencies. The user can run the A(i)DAPT software through a prebuilt Singularity container that already has all of the needed packages. If however, the user is planning on utilizing their own environment, they may need to install some dependencies (tensorflow, pandas, etc.). To see the dependencies included for building the container, see the "Singularity" file in the same directory as the ".sif" image file, specified in Sec. 3.

2 Configurations

Assuming you're already in `jlab_datascience_exp_hall`, `cd` to `Hall_B/AIDAPT`. Most of the configurations, including hyperparameters and input specifications, are set in the yaml files located in `aidapt_toolkit/configs/`. For the inner GAN, the relevant yaml file is `hydra_basic_config.yaml`. Screenshots from this yaml file are shown in Fig. 1. Additionally, the user should check all parameters specified in the yaml file to ensure they're running the desired configurations.

At the end of the yaml file, there is a section called `metrics`. Here, the user can specify whether or not they want to track and plot various training metrics, including layer-specific gradient norms, χ^2 tests, and discriminator accuracy tests. If set to `True`, the user can decide how often they want these metrics to be calculated, in terms of epoch, by giving an integer next to `<metric>_frequency`. For full scale training, it is not recommended to track these every epoch, as they can slow down the training process.

NOTE: By default, `save_path` (under `driver`) is set as `\$\{hydra:runtime.output_dir\}`). This is utilizing the Python framework `Hydra` to automatically make and name the output directory. This will place the output directory (starting from `jlab_datascience_exp_hall/Hall_B/AIDAPT/`) in `outputs/`, following the format `<yyyy>-<mm>-<dd>/<hh>-<mm>-<ss>/`, where `y` is for year, `m` is for month, `d` is for day, `h` is for hour, `m` is for minute, and `s` is for second. For example, the output directory could be `outputs/2025-06-12/10-17-40/`. Of course, the user can choose to change (or override with command line arguments) this or any other configuration in the yaml file.

Also in `aidapt_toolkit/configs/` is `hydra_outer_config.yaml`, which is the config file for the outer GAN. The user will notice some architecture differences between the inner and outer GAN's, but most of what's already

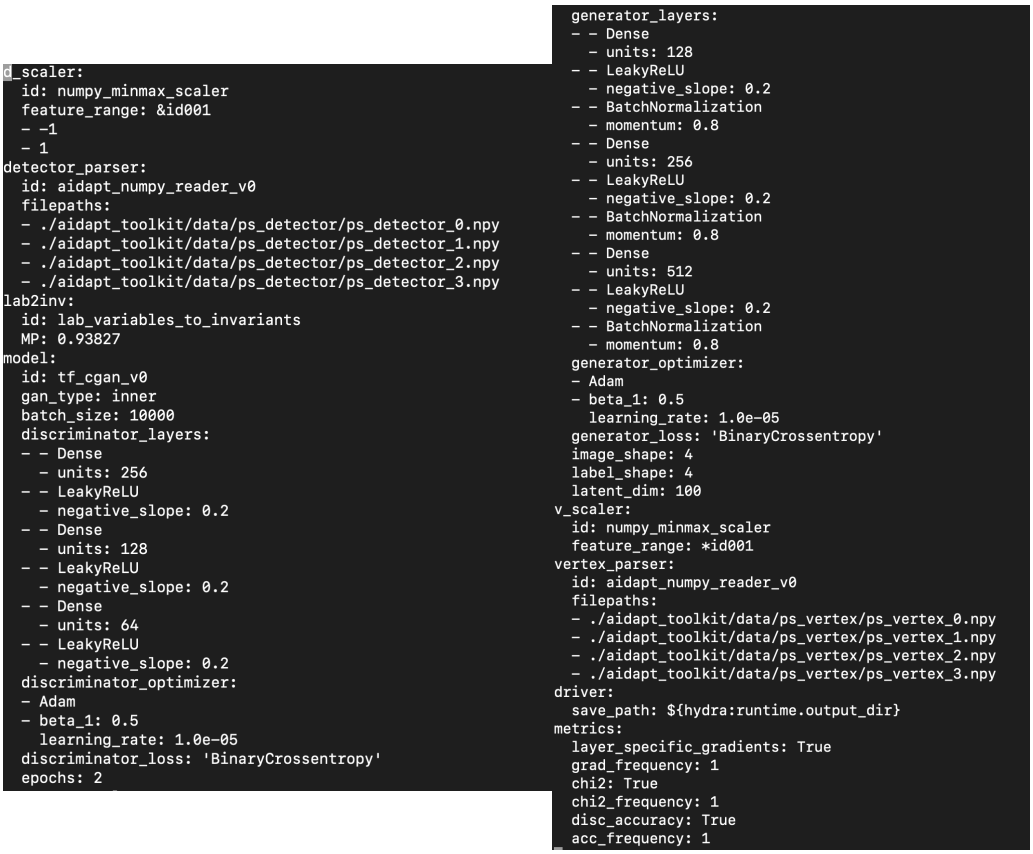

Figure 1: Screenshots of the yaml file for the inner GAN. The screenshot on the right is just a continuation of the file from what's shown on the left.

been discussed as it relates to the config files is the same for `hydra_outer_config.yaml`. One of the few important differences are the lines that tell the program where to find the trained inner GAN generator. These lines are located in the `model` sections and are named `folding_id` and `folding_path`. To run the outer GAN, an inner GAN model should already be trained. The path will depend on where the trained inner GAN model is saved.

2.1 Input Data

The input datasets are specificed by `filepaths` within the `detector_parser` and `vertex_parser` blocks in the yaml files. The data to train the inner GAN should be phasespace simulations of the desired reaction. There are two input datasets: vertex-level simulations (i.e. raw Monte Carlo) and detector-level simulations (i.e. the reconstructed Monte Carlo). The simulations should be done accordant with the energies and reconstruction software relevant for the experiment of interest. Unless the user is running the same reaction studied in this work, $\gamma p \to \pi^+\pi^- p$, modifications to `aidapt_toolkit/data_prep/lab_variables_to_invariants.py` will likely be needed. As it stands, the software is not set up to handle any arbitrary reaction. However, the information being read in `lab_variables_to_invariants.py` should be obvious enough for the user to accommodate their specific reaction.

2.2 Fine-tuning of Hyperparameters

Depending on the specific task of the user, it may be necessary to adjust some of the hyperparameters that makeup the model. Any model that is trained within this GAN configuration is reaction and experiment dependent. Therefore, it is certainly possible that some of the hyperparameters will need to be fine-tuned when applying this to other reactions and/or experiments. It is recommended to utilize the metrics that are tracked and saved during training to help guide you through which hyperparameters should be modified and in what manner. For instance, if it is observed that the loss, gradient norms, χ^2, etc. are still steadily dropping when the training is complete, it may indicate that additional training time is necessary.

3 Running the Software

With the software framework built and the configurations set, the models are essentially ready to run. As mentioned in Sec. 1, the user may take advantage of an already built Singularity container to run all software in this project. This container can be found at

`/work/data_science/aidapt/tf216_container/tensorflow-2.16.1-gpu.sif` Information about the container can be found in the `Singularity` file in the same directory.

NOTE: As mentioned in Sec. 1, the user can run the software within they're own environment, without the pre-built container. The steps below are specifically for utilizing the container, but anyone should be able to follow these procedures within their own environment if desired.

It's possible that some users may already have a trainer inner GAN to utilize. In these cases, you can skip to Sec. 3.2. Just make sure you point to the correct inner (folding) GAN model (see Sec. 2). Otherwise, you'll want to first train the inner GAN.

3.1 Training the Inner GAN

The `drivers` are the main files, which run all functions needed to complete the training. For the inner GAN, the corresponding driver file is `hydra_driver.py`. Before running a full scale training, it is preferable to interactively test your model and the environment. To do this, and assuming you're utilizing the provided container, do:

`> singularity run --bind/path/to/jlab_datascience_exp_hall:/jlab_datascience_exp_hall /path/to/container_image/tensorflow-2.16.1-gpu.sif`

You should now be inside the container, which will be evident by a `Apptainer>` prompt. Next do:

```
Apptainer> cd /jlab_datascience_exp_hall/Hall_B/AIDAPT
```

Running the training interactively will be quite slow and will likely not be possible to get quality results. But it is a good practice to make sure the training behaves as expected. To run the inner GAN interactively from within the container, run the driver:

```
python3 ./aidapt_toolkit/drivers/hydra_driver.py
```

After successfully running the inner GAN interactively, you're ready to do a full scale training. The number of epochs required to reach good results may vary depending on several factors, but will likely be on the order of tens of thousands of epochs. For example, the results shown in Ref. [1] are after 80,000 epochs. To accomplish this in a reasonable amount of time, you will want to run the training on GPU's. The easiest way to do this is by submitting a job on the JLab farm via SLURM to run the container on GPU. Below is an example SLURM submission script:

```
#!/bin/bash

#SBATCH --partition=gpu
#SBATCH --job-name=innerGAN_train
#SBATCH --output=/farm_out/%u/%x-%j-%N.out
#SBATCH --error=/farm_out/%u/%x-%j-%N.err
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=12G
#SBATCH --mail-user=<username>@jlab.org
#SBATCH --gres=gpu:TitanRTX:1
#SBATCH --time=24:00:00

#Run on GPU
singularity exec --nv \
        --bind /absolute/path/to/jlab_datascience_exp_hall:/jlab_d
        atascience_exp_hall \
        /absolute/path/to/container_image/tensorflow-2.16.1-gpu.sif \
        sh -c "cd /jlab_datascience_exp_hall/Hall_B/AIDAPT && \
        python3 aidapt_toolkit/drivers/hydra_driver.py"
```

Submit the job as you would for any other SLURM submission:

```
sbatch <slurm_script_name>
```

3.2 Training the Outer GAN

To train the outer GAN, an inner GAN model must be given, since the generator of the inner GAN is used within the outer GAN. The corresponding config file for the outer GAN is `hydra_outer_config.yaml`. See Sec. 2

for more information about pointing to the correct inner GAN model. The driver for the outer GAN is `outer_gan_driver.py`. Again, it's a good idea to test this out with a few epochs before running many epochs of training. To do this, follow the same stapes as described in Sec. 3.1. The only difference will be the driver that you execute, `outer_gan_driver.py` instead of `hydra_driver.py`. The same goes for running full scale training of the outer GAN by submitting via SLURM to run on JLab's GPU resources. Just follow the same instruction provided for the inner GAN, only changing to the appropriate driver for the outer GAN.

4 Outputs

The items that the training outputs are the same for both the inner and outer GANs. In the corresponding output directory, you will see several files and folders. The most important of the folders is `cgan_model/`. This is where the model weights and configuration info are saved. Depending on the metrics options selected by the user, there will be a varying number of `png` files in the output path. The output plots that are saved automatically are `distributions.png` (Fig. 2) and `training_analysis.png` (Fig. 3). Some additional metrics can be tracked and returned in the form of plots at the discretion of the user in the config file; these include `disc_layer_gradient_norms.png` (Fig. 4), `disc_acc_vs_epoch.png` (Fig. 5), and `chi2_vs_epoch.png` (Fig. 6).

References

[1] T. Alghamdi *et al.* Toward a generative modeling analysis of clas exclusive 2π photoproduction. *Phys. Rev. D*, 108:094030, 2023.

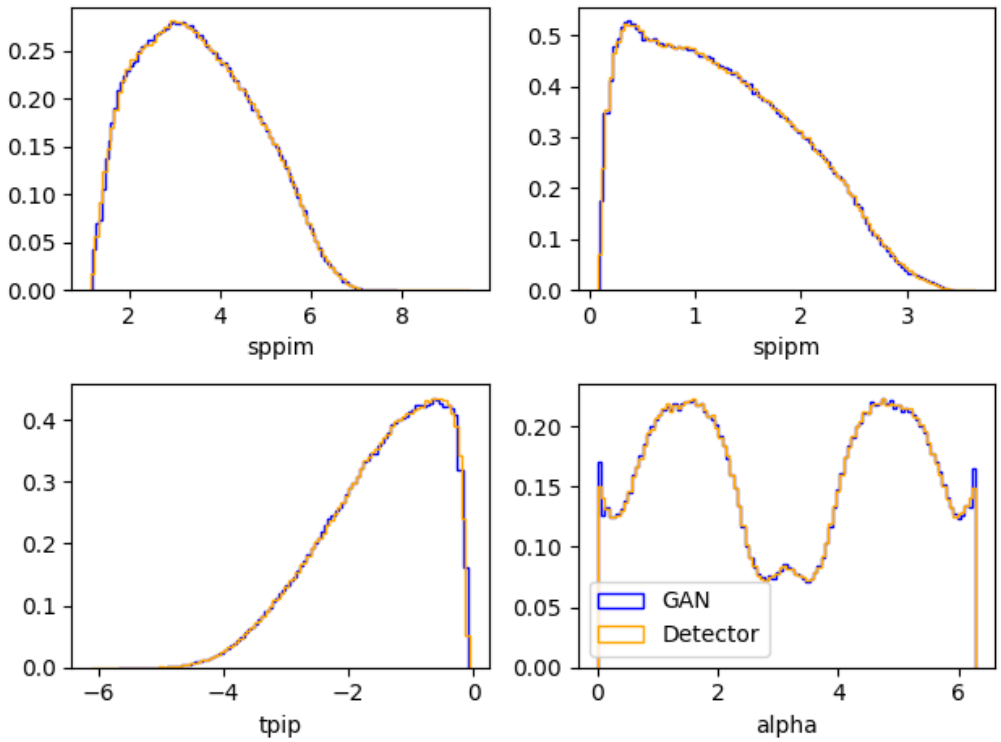

Figure 2: Distributions of the four variable on which the models are trained. These plots show an example of the inner GAN after 80,000 epochs of training. The blue histograms represent the output from the inner GAN generator, whereas the yellow histograms represent the target distributions (traditional simulations).

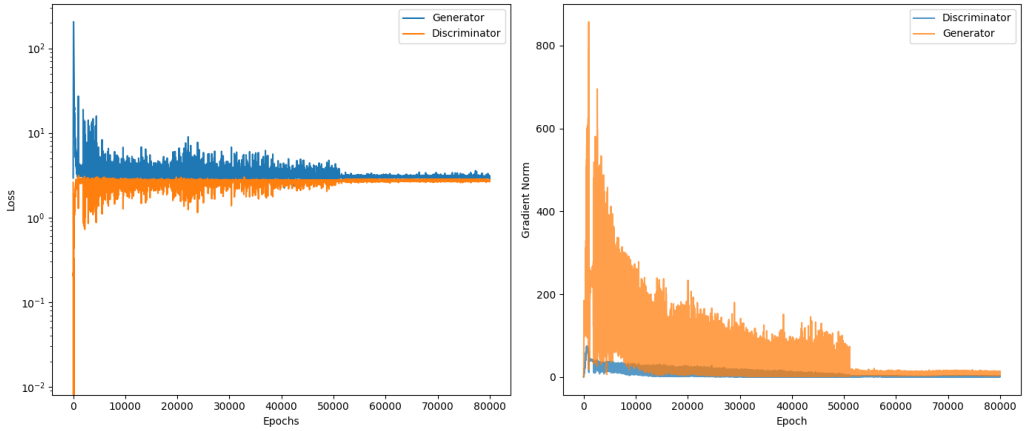

Figure 3: This figure is automatically saved under the name training_analysis.png to the output directory when training of a model is complete. The plot on the left shows the loss for the discriminator and generator as a function of epoch. The plot on the right shows the gradient norm for the discriminator and generator as a function of epoch.

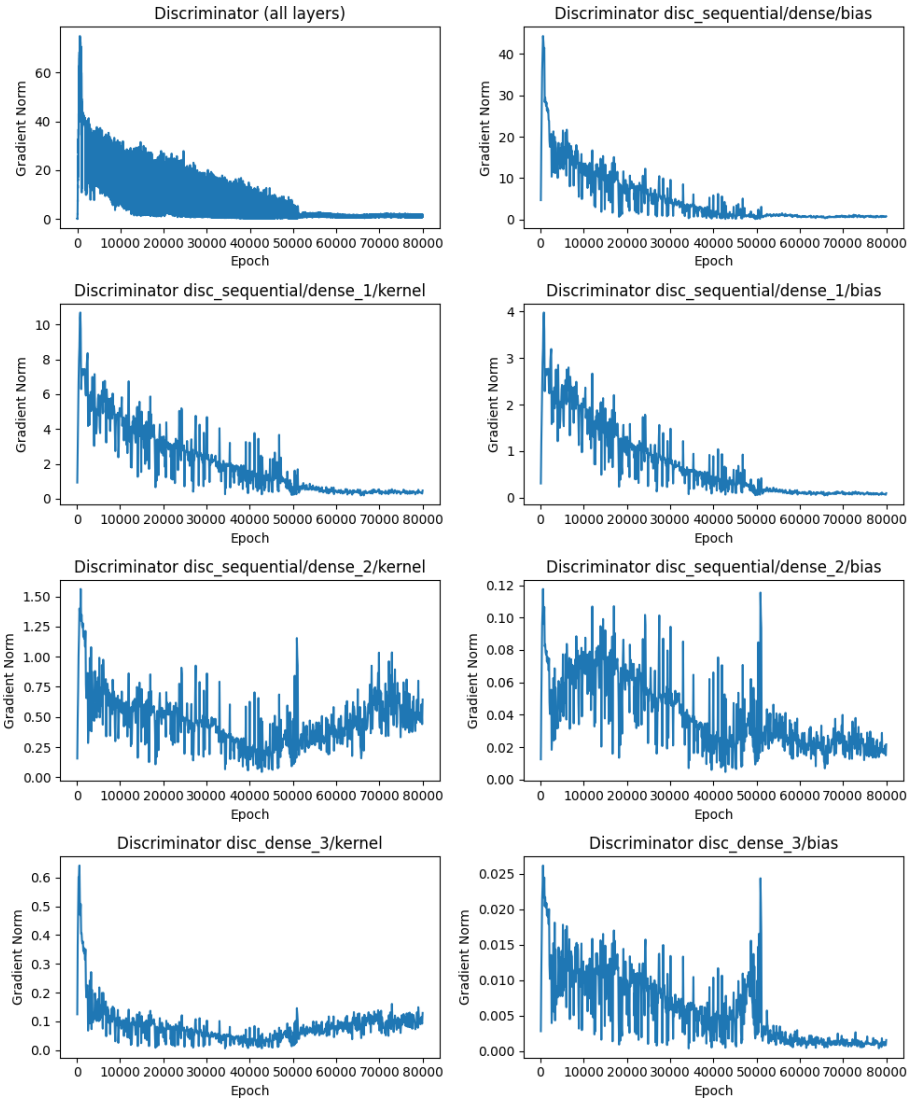

Figure 4: Plots showing the discriminator gradient norms as a function of epoch. The gradients for the discriminator overall are in the top left plot and the layer-by-layer gradients are shown in the subsequent subplots.

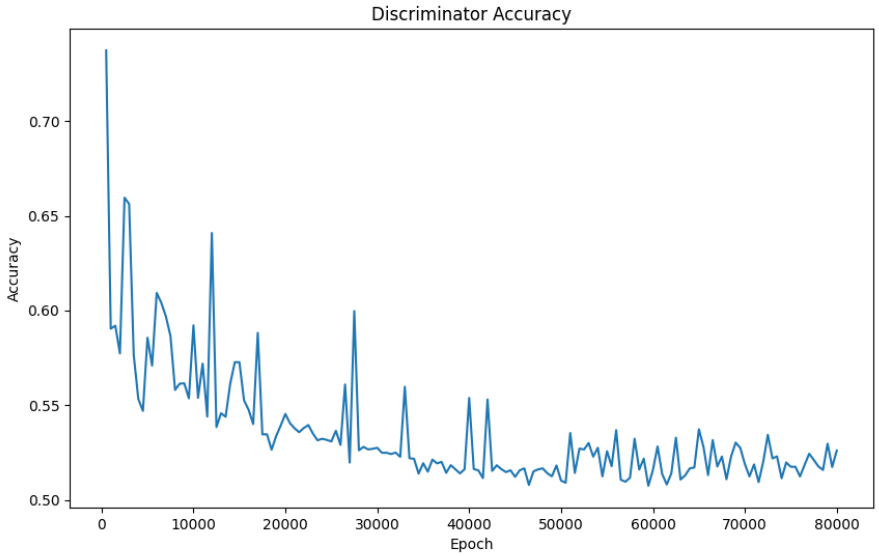

Figure 5: Discriminator accuracy test as a function of epoch. An accuracy near 1 means the discriminator is nearly 100% effective at distinguishing the GAN generator output from the target distributions.

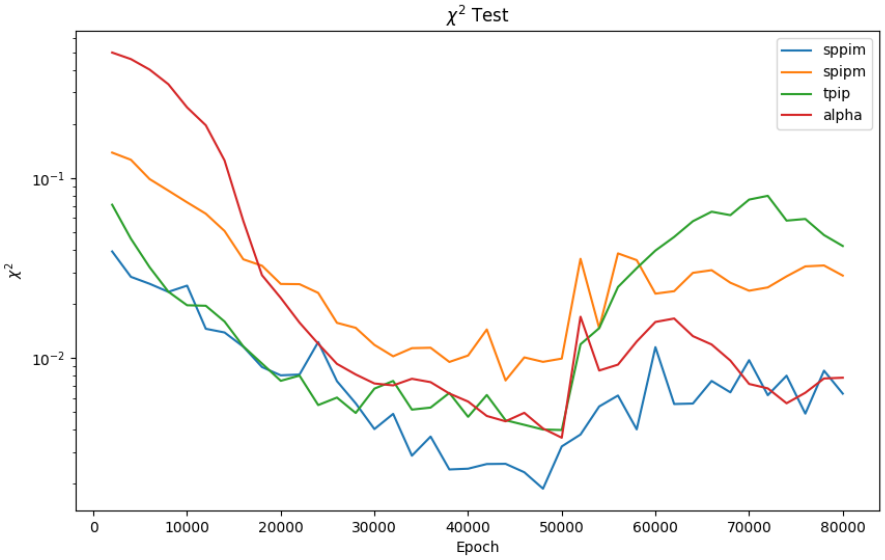

Figure 6: This is an optionally saved figure that shows the (non-normalized) χ^2 for each of the four training variables as a function of epoch.