Best Practices

The following project iterates over a large amount of data, and it's expected to take a number of hours to train, even on GPU. Follow the best practices outlined below to avoid common issues with GPU Workspaces.

- ### Keeping your connection alive during long processes

  Workspaces automatically disconnect when the connection is inactive for about 30 minutes, which includes inactivity while deep learning models are training. You can use the workspace_utils.py module **here** to keep your connection alive during training. The module provides a context manager and an iterator wrapper—see example use below.

  NOTE: The script sometimes raises a connection error if the request is opened too frequently; just restart the Jupyter kernel & run the cells again to reset the error.

  NOTE: These scripts will keep your connection alive while the training process is *running*, but the workspace will still disconnect 30 minutes after the last notebook cell finishes. Modify the notebook cells to **save your work** at the end of the last cell or else you'll lose all progress when the workspace terminates.

Example using context manager:

```
from workspace_utils import active_session

with active_session():
    # do long-running work here
```

Example using iterator wrapper:

```
from workspace_utils import keep_awake

for i in keep_awake(range(5)):
    # do iteration with lots of work here
```

- **Manage your GPU time**

  It is important to avoid wasting GPU time in Workspace projects that have GPU acceleration enabled. The benefits of GPU acceleration are most useful when evaluating deep learning models—especially during *training*. In most cases, you can build and test your model (including data pre-processing, defining model architecture, etc.) in CPU mode, then activate GPU mode to accelerate training.

- **Handling "Out of Memory" errors**

  This issue isn't specific to Workspaces, but rather it is an apparent issue between PyTorch & Jupyter, where Jupyter reports "out of memory" after a cell crashes. Jupyter holds references to active objects as long as the kernel is running—including objects created before an error is raised. This can cause Jupyter to persist large objects in memory long after they are no longer required. The only known solutions are:

  > - To reduce the `batch_size` of your data
  > - Reset the kernel and run the notebook cells again