CS224n: Practical Tips for using Virtual Machines

Contents

Managing Processes on a VM

In developing your deep learning models, you will likely have to leave certain processes, such as Tensorboard and your training script, running for multiple hours. If you leave a script running on a VM and log-off, your process will likely be disrupted. Furthermore, it is often quite nice to be able to have multiple terminal windows open with different processes all visible at the same time, without having to SSH into the same machine multiple different times.

TMUX or "Terminal Multiplexer" is a very simple solution to all the problems above.

Essentially, TMUX makes it such that in a single SSH session, you can virtually have multiple terminal windows open, all doing completely separate things. Also, you can actually tile these windows such that you have multiple terminal sessions all visible in the same window.

The basic commands are below. Terminal commands are prefaced with a " $ " otherwise the command is a keyboard shortcut.

TMUX Cheatsheet

1. Start a new session with the default name (an integer) `$ tmux`
2. Start a new session with a user-specified name `$ tmux new -s [name]`
3. Attach to a new session `$ tmux a -t [name]`
4. Switch to a session `$ tmux switch -t [name]`
5. Detach from a session `$ tmux detach OR ctrl - b - d`
6. List sessions `$ tmux list-sessions`
7. Kill a sessions `ctrl - b - x`
8. Split a pane horizontally `ctrl - b - "`

Split a pane vertically `ctrl - b - %`

9. Move to pane `ctrl - b - [arrow_key]`

Managing Code Deployment to a VM

You are welcome to use scp/rsync to manage your code deployments to the VM. However, a better solution is to use a version control system, such as **Git**. This way, you can easily keep track of the code you have deployed, what state it's in and even create multiple branches on a VM or locally and keep them sync'd.

The simplest way to accomplish this is as follows.

1. Create a Git repo on Github, Bitbucket or whatever hosted service your prefer.
2. Create a SSH key on your VM. (see the link below)
3. Add this SSH key to your Github/service profile.
4. Clone the repo via SSH on your laptop and your VM.
5. When the project is over, delete the VM SSH key from your Github/service account.

Resources:
- Github SSH key tutorial
- Codecademy Git tutorial (great for Git beginners to get started)

temu
*Note: If you use Github to manage your code, you must keep the repository **private** until the class is over. Github offers free private repositories to students.*

Managing Memory, CPU and GPU Usage on a VM

If your processes are suddenly stopping or being killed after you start a new process, it's probably because you're running out of memory (either on the GPU or just normal RAM).

First of all, it's important to check that you not running multiple memory hungry processes that maybe have slipped into the background (or a stray TMUX session).

You can **see/modify which processes you are running** by using the following commands.

1. View all processes `$ ps au`

2. To search among processes for those containing the a query, use `$ ps -fA | grep [query]`.
   For example, to see all python processes run `ps -fA | grep python.`
3. Kill a process `$ kill -9 [PID]`

You can find the PID (or Process ID) from the output of (1) and (2).

To **monitor your normal RAM and CPU usage**, you can use the following command: `$ htop` (Hit `q` on your keyboard to quit.)

To **monitor your GPU memory usage**, you can use the `$ nvidia-smi` command. If training is running very slowly, it can be useful to see whether you are actually using your GPU fully. (In most cases, when using the GPU for any major task, utilization will be close to 100%, so that number itself doesn't indicate an Out of Memory (OOM) problem.)

However, it may be that **your GPU is running out of memory simply because your model is too large** (i.e. requires too much memory for a single forward and backward pass) to fit on the GPU. In that case, you need to either:

1. Train using multiple GPUs (this is troublesome to implement, and costs much more on Azure)
2. Reduce the size of your model to fit on one GPU. This means reducing e.g. the number of layers, the size of the hidden layers, or the maximum length of your sequences (if you're training a model that takes sequences as input).
3. Lower the batch size used for the model. Note however, that this will have other effects as well (as we have discussed previously in class).
4. Use techniques like gradient accumulation (also additional work). You can find a discussion of some of these techniques in this article: https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255 . They're all also implemented in the huggingface transformers library: https://github.com/huggingface/transformers