# 10-703 Deep RL and Controls
# Homework 2
# Tensorflow, Keras, and Cluster Usage

Devin Schwab

Spring 2017

# Table of Contents

# DQN

- This homework is signficantly more work than the previous homework
  - Lots of coding
  - Training and experiments will take a long time
- If you do not start early you will not finish on time

# Code Outline

- We have provided a suggested outline for how to structure your implementation
- Feel free to modify or disregard this template
- You should use Tensorflow and/or Keras in your implementation
- If you use libraries other than those provided in the requirements.txt file, you should state so in your writeup.

# Table of Contents

# Pittsburgh Supercomputing Center (PSC)

- Pittsburgh Supercomputing Center (PSC) is a joint effort of CMU and University of Pittsburgh
- They provide a number of large-scale, supercomputing clusters
- We have an educational grant for the students of this class to use the **Bridges** machines
- Bridges is the newest cluster and it provides a number of k80 and P100 GPUs
- You should read over the user-guide for more details `https://www.psc.edu/index.php/users`

# Warnings

- We have a fixed allocation for all students in the class
- You may use this allocation for your projects and homeworks **BUT**

# Warnings

- We have a fixed allocation for all students in the class
- You may use this allocation for your projects and homeworks **BUT**
- You must be respectful and share the cluster resources

# Warnings

- We have a fixed allocation for all students in the class
- You may use this allocation for your projects and homeworks **BUT**
- You must be respectful and share the cluster resources
  - Don't run jobs you don't need to
  - Only use shared nodes
  - Minimize use of interactive sessions

# Warnings

- We have a fixed allocation for all students in the class
- You may use this allocation for your projects and homeworks **BUT**
- You must be respectful and share the cluster resources
  - Don't run jobs you don't need to
  - Only use shared nodes
  - Minimize use of interactive sessions

We will be monitoring

If we see excesive usage, we will restrict your access.

# Homework 2

- Given the limited cluster resources please do not run your homework on the cluster until you have debugged it on your own machine
- You can run for a couple hundred thousand iterations on your own CPU/GPU and see if the network is converging
- Once you see an upward trend you can run your full experiments on the cluster

# Logging in

- Over the next few days you will be given your user logins
- In most cases, the username will match the XSEDE portal login
  - If your account name was already taken then you will get a different username
- Your password is the same as your XSEDE portal login

# SSHing In

- All of your interactions with the cluster will be over SSH
- `ssh -p 2222 <username>@brdiges.psc.edu`
  - You must use port 2222
  - You have to use password based authentication

# SSH Config

- It is recommended that you add the following snippet to your `~/.ssh/config` file

```
Host     bridges
         Hostname bridges.psc.edu
         User <username>
         Port 2222
```

- This will let you login with the command `ssh bridges`

# Submitting Jobs

- When you ssh in you end up in a normal Linux home directory
- You can run basic commands here, copy files around, etc.
- Do **NOT** start any large computations here!
- To run a job you need to submit it into the job queue

# Submitting Jobs

- There are three main commands to interact with the jobs queue:
  - `sbatch` — Used to submit new jobs to the job queue
  - `squeue` — Used to check on status of jobs in the queue
  - `scancel` — Used to cancel a job in the queue

# Job Files

- Each job is specified via a batch script
- Basically a bash script with some extra commands at the top
- We have included a sample job file with the homework release.

# Example Job File

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p GPU-shared
#SBATCH --ntasks-per-node 2
#SBATCH --gres=gpu:k80:1
#SBATCH -t 10:00:00

# this will request 2 CPU cores, an one k80 on a shared GPU node
# if the job is still running after 10 hours, it will be automatically

set -x  # echo commands to stdout
set -u  # throw an error if unset variable referenced
set -e  # exit on error
```

# Example Job File

```
# helper vars
PYLON1=/pylon1/$(id -gn)/$USER
PYLON2=/pylon2/$(id -gn)/$USER

module load cuda/8.0

# select which python module you want 2/3
module load python3
# module load python2

# switch to pylon1
# NOTE: Files in this directory are deleted when 30 days old
pushd $PYLON1

# turn on the virtualenv
source $PYLON2/my-virtualenv/bin/activate

# run the experiment script
python $PYLON2/deeprl_hw2/dqn_atari.py --env Breakout-v0
```

# File Systems

- There are three main storage systems on Bridges:
  - Your home directory
    - 10 GB limit
    - Backed up daily
    - Available from all machines
    - Good for source code, temp files, job files, etc.
  - Pylon1
    - Part of allocation quota
    - Faster IO than home dir
    - Files older than 30 days are deleted!
    - Located at `/pylon1/$(id -gn)/$USER`
  - Pylon2
    - Part of allocation quota
    - Not backed up!
    - No timelimit on storage
    - Do not use for working space for running jobs
    - Located at `/pylon2/$(id -gn)/$USER`

# Software Modules

- Bridges provides a number of built in software packages
- Check the website for a full listing
- Or run the `module avail <search string>` command
  - e.g. `module avail python3`

# Setting up Virtualenv

- There is a Tensorflow module but:
  - Currently, no Python 3 version
  - Version 1.0 not on cluster
- I use a virtualenv for my jobs
- The commands are the same as for the first homework
- You can store the virtualenv in your home dir, pylon1 or pylon2

# Setting up Virtualenv

```
module load python3
virtualenv deeprl-hw2-gpu
source deeprl-hw2-gpu/bin/activate
pip install tensorflow-gpu
deactivate
```

# Running the Job File

- Once you have the job file created just run the command `sbatch example.job`

# Checking execution

- `squeue` can be used to check on running jobs
- `squeue -u $USER` will show you all of your submitted jobs
- `scancel $JOBID` will cancel the specified job id
- stderr and stdout are saved in a file called `slurm-$JOBID.out` in your home dir
  - `cat slurm-$JOBID.out`
  - `tail -f slurm-$JOBID.out` to follow output

# Table of Contents

# What is Tensorflow?

- An open source machine learning library from Google
- Great for Neural Networks
- Designed for general computations
- Automatically computes gradients

# Compared to Other Frameworks

- There are a ton of other frameworks available but Tensorflow has a few pros
  - Officially supports C++ and Python
  - The tooling is better: Tensorboard, TF Debugger, etc.
  - Easy to deploy models onto different hardware (phones, robots, etc.)
  - Designed to work with multiple GPUs and distributed systems

# Installing

- As of 1.0 just install one of the following packages with pip:
  - `tensorflow`
  - `tensorflow-gpu`

# Tensor

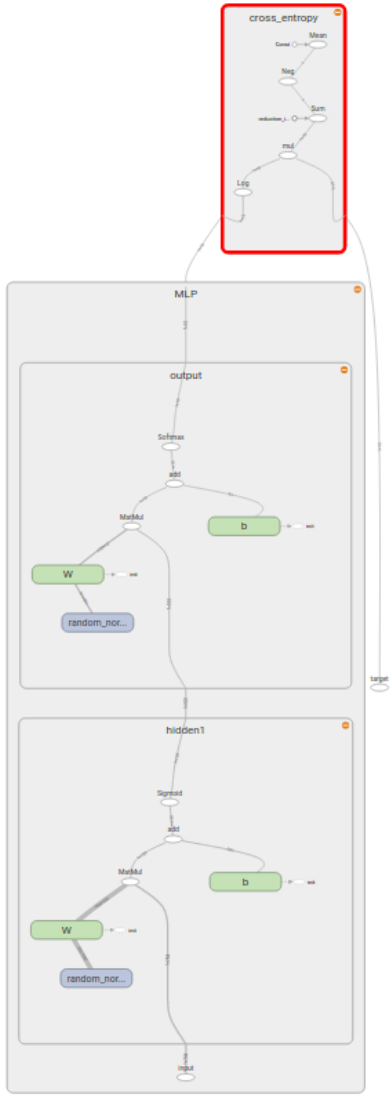
- Represents a multi-dimensional array of some type
- Most important datatype in Tensorflow programs
- Tensors are combined with operations to create new tensors
- All of the tensors are part the computation graph

# Computation Graphs

- Computations are organized into fixed graphs[1]
- Creating operations, variables, etc to the graph does **NOT** execute them immediately
  - They are simply added to the graph for later execution in a session
- Parts of the graph can be decoupled
  - You can execute any subgraph, provided you specify all required input values
  - You can mix and match Python/Numpy with tensorflow graph computations

[1] Now they have some dynamic graph support.

# Graph Example



# Basic TF Program Structure

- Tensorflow programs generally follow this structure:
  1. Build computation graph
  2. Execute computation graph

# Sessions

- To execute a graph, you need to be in a **Session**
- Sessions can be run on CPUs, GPUs or combinations
- Multiple sessions can share the same graphs, but have different variable values
- Multiple ways create a session:
  - `sess = tf.InteractiveSession()`
  - `with tf.Session() as sess:`
- To run a tensor:
  - `output = sess.run(my_tensor)`
  - The output will match the datatype and shape of `my_tensor`

# Example

```
import tensorflow as tf
# create the tensor gamma
gamma = tf.constant(.99, tf.float32, name='gamma')
# open a session called sess
with tf.Session() as sess:
  # execute the tensor gamma in the session 'sess'
  print(sess.run(gamma))
```

The output will be: .99

# Variables

- Tensors which have mutable values
- `b = tf.Variable(tf.zeros([10]), name='bias')`
- Must have a unique name
  - If no name provided, then TF will generate one
  - Highly recommended that you name them!
- Variables must be initialized in every session before use!
  - `sess.run(tf.global_variables_initializer())`

# Name Scopes

- All tensors exist in some namescope
- You can set the names of most operation outputs by providing the `name` argument
- Slashes separate pieces of the naming hierarchy
  - "/name1/W" is different from "/name2/W"
- When initializing groups of tensors that should be in the same name scope, use the `tf.name_scope` function:

```
with tf.name_scope('dense'):
    W = tf.Variable(
            tf.zeros([784, 10]), name='W')
    b = tf.Variable(tf.zeros([10]), name='b')
```

# Placeholders

- Used for dynamic graph inputs such as:
  - batch inputs
  - target values for batch
  - Variable to control test/train behavior
- `input = tf.placeholder(tf.float32, shape=[None, 784])`
  - Must specify data type
  - Shape is optional, but better to specify if you know it
  - Unknown shape dimensions can be marked None

# Basic Dense Layer

```
def create_fc_layer(input, num_neurons, activation, name):
    input_shape = input.get_shape()
    with tf.name_scope(name):
        W = tf.Variable(
            tf.random_normal(
                [input_shape[-1].value, num_neurons], stddev=0.35),
            name='W')
        b = tf.Variable(tf.zeros([num_neurons]), name='b')
        preactivation = tf.matmul(input, W) + b
        output = activation(preactivation)
    return output, preactivation, [W, b]
```

# Stacking the Layers

```
def create_single_hidden_layer_net(net_name):
    with tf.name_scope(net_name):
        input = tf.placeholder(tf.float32,
                               shape=[None, 784], name='input')
        h_out, h_pre, h_vars = create_fc_layer(input,
                                               100,
                                               tf.sigmoid,
                                               'hidden1')
        out, out_pre, out_vars = create_fc_layer(h_out,
                                                 10,
                                                 tf.nn.softmax,
                                                 'output')
    return input, out
```

# Executing the Layers

```
output = sess.run(out,
          feed_dict={input: np.random.randn(10, 784)})
```

# Setting a Loss Func

```
def create_loss(predicted):
    target = tf.placeholder(
        tf.float32, shape=predicted.get_shape(), name='target')
    with tf.name_scope('cross_entropy'):
        cross_entropy = tf.reduce_mean(-tf.reduce_sum(
            target * tf.log(predicted), reduction_indices=[1]))

    return target, cross_entropy
```

# Optimizers

- TF train provides many optimizers
- All optimizers follow the same API, but some may have internal variables you need to save/reload
- Calling minimize on a tensor adds operations to the graph that run backpropagation wrt that tensor
- To update weights, only need to run the returned training operation

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
train_op = optimizer.minimize(cross_entropy)

sess.run(train_op, feed_dict={input: batch_input,
                              target: batch_targets})
```

# Saving Your Model

- Use the tf.train.Saver class
- You can specify specific variables to save, or save them all
- Only saves the variable values by default!
  - If you change your graph structure and then try to reload, things may not work
  - You can save a MetaGraph instead, which includes the graph structure

```
saver = tf.train.Saver()
saver.save(sess, '/tmp/checkpoint')
```

# Tensorboard Summaries

- Tensorboard is a visualization and debugging tool for TF programs
- Lets you plot scalar and multidimensional values
- Gives you an interactive display of the computation graph
- Can even show images and play audio inputs/outputs to the network

```
writer = tf.summary.FileWriter('logs', sess)
loss_summary_op = tf.summary.scalar('cross_entropy', cross_entropy)
summary = sess.run(loss_summary_op, feed_dict)
writer.add_summary(summary, global_step=1)
```

# Tensorboard Demo

# Table of Contents

# What is Keras?

- High level API for TF (and other libraries)
- Pure Python API

# Basic Layers

- Provides a ton of different layers such as:
  - Dense
  - Activation
  - Dropout
  - BatchNormalization
  - Convolution2D
- Train/Test controlled by the `keras.backend.learning_phase()` tensor
  - Pass this in via the feed_dict with 1 for training and 0 for test
  - `feed_dict={keras.backend.learning_phase(): 1}`

# Functional Model API

- Can define generic network layouts (vs Sequential model)
- Provides nice API for
  - Inference — `predict_on_batch`
  - Training — `train_on_batch`
  - All trainable weights — `trainable_weights` attribute
  - Any special updates — `updates` attribute

## Fit and Evaluate Methods

Also provides a fit and evalute method, but I recommend not trying to use these. They're designed for fixed datasets of like image classifiers.

# Example

```
def create_model(input_size, output_size):
    input = Input(shape=(input_size, ), name='input')
    with tf.name_scope('hidden1'):
        hidden1 = Dense(100, activation='sigmoid')(input)
    with tf.name_scope('output'):
        output = Dense(10, activation='softmax')(input)

    print(model.summary())

    return model
```

# Optimizer

- Provides a number of optimizer implementations
- All of them support gradient clipping out of the box
  - `adam = keras.optimizers.Adam(lr=.001)`
- `get_updates` method is the Keras version of TF's `minimize`

# How to train?

- Keras models must be compiled which:
  - initializes all model vars in the session
  - Adds optimization updates to graph
  - Adds metrics operations to the graphs
- `model.compile(optimizer='adam', loss='mse')`
- For this assignment you would need to specify your own loss function

# Saving and Loading a Model

- You can get a dictionary defining the model structure with `get_config`
  - This is a useful method for cloning a model (such as making a target network)
- You can save that dictionary using pickle or any other save method in python
- You can construct a new model from that dictionary with the `model_from_config`
- To save/reload weights just use the `save_weights` and `load_weights` functions

# Backend functions

- Keras can also work with Theano as a backend
- To write code that can work with either Theano or Tensorflow, use the backend functions
- `import keras.backend as K`
- Most tf functions have an equivalent function in the K module

# Manually running a model/layers

- You can use keras just to create the models and then train and use the model with regular TF code.
- This means you can add summary operations for Tensorboard logging just like in the pure TF example
- If you want to manually run training operations for a model you need to:
  - Make sure that the `updates` attribute is run during the session execution. This handles dropout and batchnorm layers
  - Pass in the `K.learning_phase()` feed dict value
  - Run the output operations from the optimizer `get_updates` method.

# Table of Contents

# Conclusion

- Start your homework early!
- Post questions on piazza
  - We will try and answer as soon as we can
- Don't waste cluster resources
  - Debug on your own machine, not on the cluster
- Refer to the TF and Keras docs for more info and in-depth examples
- Complete examples that go along with these slides will be posted to the website

# References

- tensorflow.org
- keras.io
- https://www.psc.edu/index.php/bridges/user-guide/connecting-to-bridges