```
'''
Deep Learning Practice: XOR on the Supercomputer

Command line version

This is the full solution: no fair peeking!

Solution for both module 2 and module 3

Andrew H. Fagg (andrewhfagg@gmail.com)

'''
import sys
import argparse
import copy
import pickle
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import os
import time
import wandb
import socket

# This is the keras 3 way of doing things
from keras.layers import Input, Dense
from keras.models import Sequential
from keras.utils import plot_model

# Default plotting parameters
FONTSIZE = 18
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams['font.size'] = FONTSIZE


import matplotlib.pyplot as plt
from matplotlib import colors

################################################################
# Default plotting parameters
FONTSIZE = 18
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams['font.size'] = FONTSIZE

################################################################
```

```
def build_model(n_inputs:int,
                n_hidden:[int],
                n_output:int,
                activation_hidden:str='elu',
                activation_output:str='elu',
                lrate:float=0.001,
                verbose:int=0)->Sequential:
    '''
    Build a simple fully connected model

    :param n_inputs: Number of input dimensions
    :param n_hidden: List: number of units in each hidden layer
    :param n_output: Number of ouptut dimensions
    :param activation_hidden: Activation function to be used for hidden and output units
    :param activation_output: Activation function for the output unit(s)
    :param lrate: Learning rate for Adam Optimizer
    '''
    # Simple sequential model
    model = Sequential();

    # Input layer
    model.add(Input(shape=(n_inputs,), name = 'Input'))

    # Hidden layer
    for i,n in enumerate(n_hidden):
        model.add(Dense(n,
                        use_bias=True,
                        activation=activation_hidden,
                        name=f'hidden_{i}',
                       ))


    # Output layer
    model.add(Dense(n_output,
                    use_bias=True,
                    activation=activation_output,
                    name='output',
                   ))

    # My favorite optimizer
    opt = keras.optimizers.Adam(learning_rate=lrate,
                                amsgrad=False)


    # Compile the model.  Mean squared error loss
    model.compile(loss='mse', optimizer=opt)

    # Display the network
    if verbose >= 1:
```

```
        print(model.summary())

    #
    return model


def args2string(args:argparse.ArgumentParser)->str:
    '''
    Translate the current set of arguments into a string that can be
    used as a file name.

    :param args: Command line arguments
    '''
    return "exp_%02d_hidden_%s"%(args.exp, '_'.join([str(i) for i in args.hidden]))


########################################################
def execute_exp(args:argparse.ArgumentParser):
    '''
    Execute a single instance of an experiment.  The details are specified in the args object

    :param args: Command line arguments
    '''

    ##############################
    # Run the experiment

    # Create training set: XOR
    ins = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    outs = np.array([[0], [1], [1], [0]])
    #####

    # Create the model
    model = build_model(ins.shape[1], args.hidden, outs.shape[1],
                        activation_output='sigmoid',
                        verbose=args.verbose)

    ############################################
    # Callbacks
    #  All of the callbacks are executed once per epoch

    cbs = []

    # Stop training early if we stop making progress
    early_stopping_cb = keras.callbacks.EarlyStopping(patience=args.patience,
                                                      restore_best_weights=True,
                                                      min_delta=0.01,
                                                      monitor='loss')
    cbs.append(early_stopping_cb)
```

```
    ##############################################
    # Describe arguments
    argstring = args2string(args)
    print("EXPERIMENT: %s"%argstring)

    # Output pickle file
    fname_output = "results/xor_results_%s.pkl"%(argstring)

    # Does this file already exist?
    if not args.force and os.path.exists(fname_output):
        print("File %s already exists."%fname_output)
        return

    ##############################################
    # Plot the model to a file
    if args.render:
        plot_model(model, to_file='results/%s_model_plot.png'%argstring, show_shapes=True, show_layer_names=True)

    ##############################################
    # WandB
    if args.wandb:
        # Connect to WandB
        wandb.init(project=args.wandb_project,
                   name=f'{args.wandb_run_label}_{args.exp:02d}',
                   notes=argstring,
                   config=vars(args))

        if args.render:
            # Log the architecture diagram to WandB
            wandb.log({'architecture': wandb.Image('results/%s_model_plot.png'%argstring)})

        # Log hostname
        wandb.log({'hostname': socket.gethostname()})

        # Create WandB callback
        wandb_metrics_cb = wandb.keras.WandbMetricsLogger()
        cbs.append(wandb_metrics_cb)


    ##############################################
    # Training
    print("Training...")

    history = model.fit(x=ins,
                        y=outs,
                        epochs=args.epochs,
                        verbose=args.verbose>=2,
```

```
                        callbacks=cbs,
                        )

    ##############################################
    print("Done Training")

    # Accumulate the results together
    results = {}
    results['ins'] = ins
    results['outs'] = outs
    results['pred'] = model.predict(ins)
    results['eval_train'] = model.evaluate(ins,outs)

    # Save the training history
    with open(fname_output, "wb") as fp:
        pickle.dump(history.history, fp)
        pickle.dump(args, fp)
        pickle.dump(results, fp)
        # Could save other things

    ##############################################
    # Close connection to WandB
    if args.wandb:
        wandb.finish()


#####
# Miscellaneous functions for visualizing results

def display_learning_curve(fname:str):
    '''
    Display the learning curve that is stored in fname.
    As written, effectively assumes local execution
    (but could write the image out to a file)

    :param fname: Results file to load and dipslay

    '''

    # Load the history file and display it
    fp.close()

    # Display
    plt.plot(history['loss'])
    plt.ylabel('MSE')
    plt.xlabel('epochs')

def display_learning_curve_set(dir:str, base:str):
    '''
```

```
    Plot the learning curves for a set of results

    As written, effectively assumes local execution
    (but could write the image out to a file)

    :param base: Directory containing a set of results files
    '''
    # Find the list of files in the local directory that match base_[\d]+.pkl
    files = [f for f in os.listdir(dir) if re.match(r'%s.+.pkl'%(base), f)]
    files.sort()

    # Loop over the files
    for f in files:
        # Load the history data and plot it
        with open("%s/%s"%(dir,f), "rb") as fp:
            history = pickle.load(fp)
            plt.plot(history['loss'])

    # Finish figure
    plt.ylabel('MSE')
    plt.xlabel('epochs')
    plt.legend(files)

#####
def create_parser()->argparse.ArgumentParser:
    '''
    Create a command line parser for the XOR experiment
    '''
    # Create the parser
    parser = argparse.ArgumentParser(description='XOR Learner',
                                     fromfile_prefix_chars='@')

    # Experiment config
    parser.add_argument('--exp', type=int, default=0, help='Experiment number')

    # Training config
    parser.add_argument('--epochs', type=int, default=100, help='Number of training epochs')
    parser.add_argument('--patience', type=int, default=100, help='Number of epochs to wait before Early Stopping')

    parser.add_argument('--force', action='store_true', help='Execute experiment even if there is already a results file')
    parser.add_argument('--nogo', action='store_true', help='Do not perform the experiment')
    parser.add_argument('--verbose', '-v', action='count', default=0, help="Verbosity level")

    # Network config
    parser.add_argument('--hidden', nargs='+', type=int, default=[2], help='Number of hidden units')  # List of hidden layers

    # Misc
    parser.add_argument('--render', action='store_true', help='Generate a PNG of the network')
```

```
    # WandB support
    parser.add_argument('--wandb', action='store_true', help='Turn on reporting to WandB')
    parser.add_argument('--wandb_project', type=str, default='XOR', help='WandB project name')
    parser.add_argument('--wandb_run_label', type=str, default='version0', help='WandB project name')

    return parser

'''
This next bit of code is executed only if this python file itself is executed
(if it is imported into another file, then the code below is not executed)
'''
if __name__ == "__main__":
    # Parse the command-line arguments
    parser = create_parser()
    args = parser.parse_args()

    # Turn off GPUs in case they were allocated
    tf.config.set_visible_devices([], 'GPU')

    # Do the work
    execute_exp(args)
```