CANDLE Tutorial: Library Overview

ECP Annual Meeting, Houston, TX
February 2020

Jamaludin Mohd Yusof
Advanced Architectures and Applications
Los Alamos National Laboratory
jamal@lanl.gov

EXASCALE COMPUTING PROJECT

Talk Outline

- Introduction
- Benefits
- Library Overview
- Example Benchmark Workflow
- Simple Parameter Sweeps
- Recent updates

Introduction

- Purpose
  - To streamline the writing of CANDLE-compliant codes
  - Allow rapid prototyping and exploration of hyperparameters
  - Integrate with the Supervisor framework
- Historical perspective
  - Consolidation of frequently used functionality from the Benchmark codes
  - Evolving to incorporate new functionality as needed
    - Improved usability over time

The CANDLE Environment

Hyperparameter Sweeps, Data Management (e.g. DIGITS, Swift, etc.)	Workflow
Network description, Execution scripting API (e.g. Keras, Mocha)	Scripting
Tensor/Graph Execution Engine (e.g. Theano, TensorFlow, LBANN-LL, etc.)	Engine
Architecture Specific Optimization Layer (e.g. cuDNN, MKL-DNN, etc.)	Optimization

Benefits provided by CANDLE

- Consistent
  - Standardized network specification with a “`default_model_file`”
  - Standardized command line intercept protocol
  - Standardized default values across frameworks
  - Ideal for testing the same problems with consistency on new DOE hardware
- Convenient
  - Pass arguments via command line
    - Standard keywords parsed automatically, user can add new ones
  - Modify the default file
    - Provide a new default model specification ‘`--config_file new_default_model.txt`’

Benefits provided by CANDLE

- Provides various utility packages that promote reuse and streamline code development
  - Actively developed, new functionality based on need
  - Added UQ, LOOCV, data subsetting this year
- Provides the pathway for inferencing, data-parallelism, automated sweeps of hyperparameters
- Availability of a robust framework for documentation and testing
- Pre-existing for containers such as Singularity (Ex. machines such as Theta, Titan, Cori, summitdev)
- Documentation: https://ecp-candle.github.io/Candle/html/index.html

Library Overview

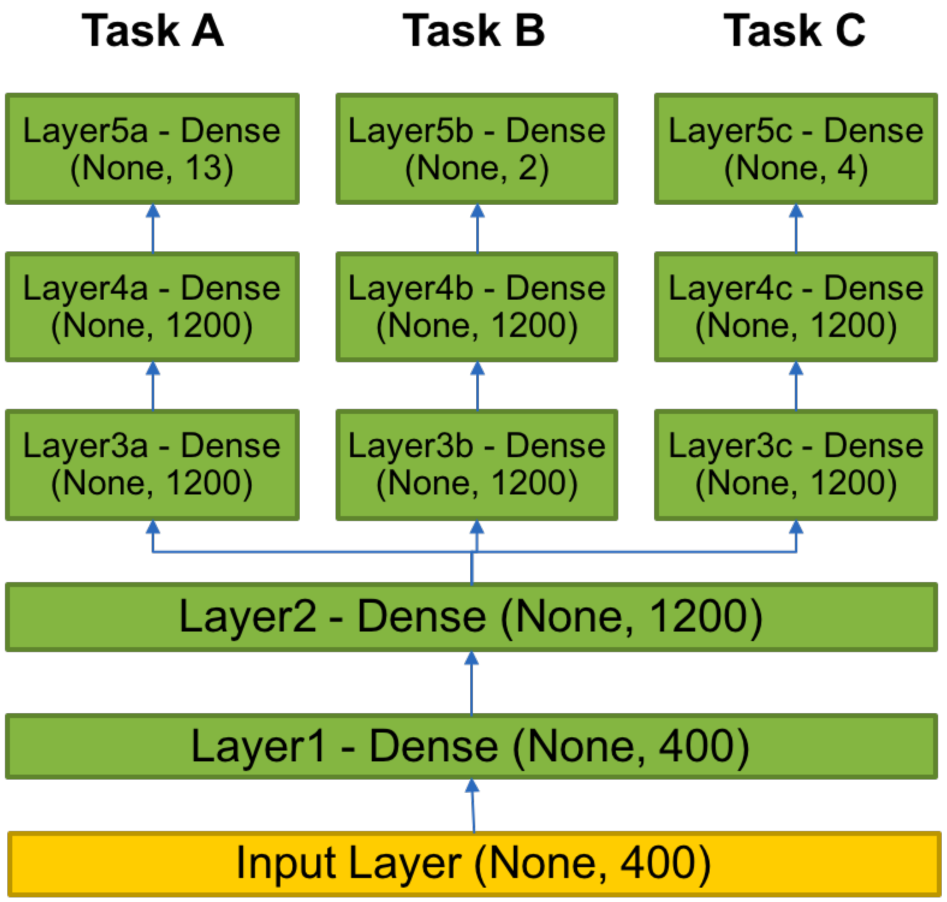
- Integrated into the scripting level of CANDLE stack
  - Keras, PyTorch versions
  - Allows multiple backends (Tensorflow, FlexFlow?)
- Provides a single namespace for inclusion of useful functions into Benchmark codes
- Allows developers to decide which functions are exposed to users
  - `e.g. candle_keras` directory with `__init__.py` file sets included functions
- Allows reuse of non-Keras specific functions to create libraries for other languages

Library organization

- Utilities are organized by functionality
  - Transparent to the user, mostly framework agnostic
    - default_utils: create, modify parameter dictionary
    - file_utils: fetch and unpack da†a files
    - data_utils: load and manipulate data, enable UQ (via uq_utils)
    - generic_utils: callback function, standardize screen output
    - keras_utils: translation from CANDLE keywords, enhance Keras functionality
    - pytorch_utils: translation from CANDLE keywords, enhance PyTorch functionality
    - solr_keras: database functionality
    - uq_utils: UQ functionality
    - viz_utils: visualize networks and output (prototype)

Example benchmark

- **P3B1: Multi-task Deep Neural Net (DNN) for data extraction from clinical reports**
- **Overview**: Given a corpus of patient-level clinical reports, build a deep learning network that can simultaneously identify:(i) b tumor sites, (ii) t tumor laterality, and (iii) g clinical grade of tumors.
- **Relationship to core problem**: Instead of training individual deep learning networks for individual machine learning tasks, Build a multi-task DNN that can exploit task-relatedness to simultaneously learn multiple concepts.
- **Expected outcome**: Multi-task DNN that trains on same corpus and can automatically classify across three related tasks.



Original code

```
# Define network
shared_nnet_spec= [ 1200 ]
individual_nnet_spec0= [ 1200, 1200 ]
individual_nnet_spec1= [ 1200, 1200 ]
individual_nnet_spec2= [ 1200, 1200 ]
individual_nnet_spec = [ individual_nnet_spec0, individual_nnet_spec1, individual_nnet_spec2 ]

# Define hyperparameters
learning_rate = 0.01
batch_size = 10
n_epochs = 10
dropout = 0.0

## Read files
from data_utils import get_file
origin = 'http://ftp.mcs.anl.gov/pub/candle/public/benchmarks/P3B1/P3B1_data.tgz’
data_loc = get_file('P3B1_data.tgz', origin, untar=True, md5_hash=None, cache_subdir=‘Pilot3')
```

CANDLE model file

```
data_url = 'ftp://ftp.mcs.anl.gov/pub/candle/public/benchmarks/P3B1/’
train_data = 'P3B1_data.tar.gz’
model_name = 'p3b1’
learning_rate = 0.01
batch_size = 10
epochs = 10
drop = 0.0
activation = 'relu’
out_activation = 'softmax’
loss = 'categorical_crossentropy’
optimizer = 'sgd’
metrics = 'accuracy’
n_fold = 1
shared_nnet_spec = '1200’
ind_nnet_spec = '1200, 1200:1200, 1200:1200, 1200’
feature_names = 'Primary site:Tumor laterality:Histological grade’
timeout =1800
scaling = 'none’
output_dir = '.’
initialization='glorot_uniform’
```

CANDLE code

```
gParameters = candle.finalize_parameters()
….

# input layer
layer = Input( shape = ( input_dim, ), name= 'input' )
shared_layers.append( layer )

# shared layers
for k in range( len( shared_nnet_spec ) ):
      layer = Dense( shared_nnet_spec[ k ], activation=gParameters['activation'],
            name= 'shared_layer_' + str( k ) )( shared_layers[ -1 ] )

      if gParameters['drop'] > 0:
            layer = Dropout( gParameters['drop'] )( shared_layers[ -1 ] )
      shared_layers.append( layer )

# individual layers
……
path = gParameters['data_url']
fpath = candle.fetch_file(path + gParameters['train_data'], 'Pilot3', untar=True)
```

Simple parameter exploration

- Provide a new default model specification
  - `‘--config_file new_default_model.txt’`
- Overwrite individual parameters in the default model
  - `‘--learning_rate 0.1 -drop 0.1’`
- Provides an easy way to perform individual experiments to probe the hyperparameter space

```
python myDNN.py -learning_rate 0.01 -run_id “run1”
python myDNN.py -learning_rate 0.02 -run_id “run2”
```

- Basic NAS via layer size/shape modification
  - `‘--shared_nnet_spec '1200, 600’’`
- Provides the pathway for automated sweeps of hyperparameters
  - → Supervisor workflows

UQ functionality

- Cross-validation
  - Generate repeatable partitions of training, validation and testing sets
  - Fraction, block and individual data specification possible
- Leave-one-out cross-validation (LOOCV)
  - Extreme case of k-fold cross validation for large number of labels
  - Iterative refinement of data sets to identify outliers
  - See Data Analysis Workflow example later

UQ functionality (in development)

- Data subsetting
  - Data normalization and batch effect removal
  - Improve data quality for subsequent analysis
- Feature selection
  - identifies a subset of features that are predictive, decorrelated, and generalizable
  - reduce model complexity
  - increase model training speed
  - improve prediction performance

RL Support (in development)

- ExaLearn requires more complex workflows
- CANDLE-RL library module
  - Additional default keywords for RL networks
- Additions at Supervisor level
  - Support for more complex workflows
  - Multiple agents/learners with integrated environments (simulation)
  - May be distributed across a variety of resources

Acknowledgments

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations - the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative.

EXASCALE COMPUTING PROJECT