Deep Learning with Deep Water

Wen Phan Magnus Stensmo Mateusz Dymczyk Arno Candel Qiang Kou

Edited by: Angela Bartz

http://h2o.ai/resources/

August 2018: First Edition

Deep Learning with Deep Water
by Wen Phan, Magnus Stensmo,
Mateusz Dymczyk, Arno Candel, & Qiang Kou
Edited by: Angela Bartz

Published by H2O.ai, Inc.
2307 Leghorn St.
Mountain View, CA 94043

August 2018: First Edition

Photos by ©H2O.ai, Inc.

Printed in the United States of America.

Contents

Introduction

This booklet introduces the reader to H2O Deep Water, a framework for GPU-accelerated deep learning on H2O. H2O Deep Water leverages prominent open source deep learning frameworks, such MXNet, TensorFlow, and Caffe, as backends. Throughout the booklet, Python examples and code snippets will be provided for the reader. A quick start is provided to quickly familiarize the reader with the Deep Water Python API and its key features. A section on image classification is also provided and demonstrates using pre-defined, user-defined, and pre-trained networks. As part of the H2O platform, Deep Water can take advantage of grid search, model checkpointing, and ensembles, and examples of these are also provided. This booklet also includes a section describing how Deep Water can be used for unsupervised learning tasks. Finally, deploying Deep Water models for inference is discussed. To learn more about the H2O platform, please visit: `docs.h2o.ai`.

What is H2O?

H2O.ai is focused on bringing AI to businesses through software. Its flagship product is H2O, the leading open source platform that makes it easy for financial services, insurance companies, and healthcare companies to deploy AI and deep learning to solve complex problems. More than 9,000 organizations and 80,000+ data scientists depend on H2O for critical applications like predictive maintenance and operational intelligence. The company – which was recently named to the CB Insights AI 100 – is used by 169 Fortune 500 enterprises, including 8 of the world's 10 largest banks, 7 of the 10 largest insurance companies, and 4 of the top 10 healthcare companies. Notable customers include Capital One, Progressive Insurance, Transamerica, Comcast, Nielsen Catalina Solutions, Macy's, Walgreens, and Kaiser Permanente.

Using in-memory compression, H2O handles billions of data rows in-memory, even with a small cluster. To make it easier for non-engineers to create complete analytic workflows, H2O's platform includes interfaces for R, Python, Scala, Java, JSON, and CoffeeScript/JavaScript, as well as a built-in web interface, Flow. H2O is designed to run in standalone mode, on Hadoop, or within a Spark Cluster, and typically deploys within minutes.

H2O includes many common machine learning algorithms, such as generalized linear modeling (linear regression, logistic regression, etc.), Naïve Bayes, principal components analysis, k-means clustering, and word2vec. H2O implements best-in-class algorithms at scale, such as distributed random forest, gradient boosting, and deep learning. H2O also includes a Stacked Ensembles method, which finds

the optimal combination of a collection of prediction algorithms using a process known as "stacking." With H2O, customers can build thousands of models and compare the results to get the best predictions.

H2O is nurturing a grassroots movement of physicists, mathematicians, and computer scientists to herald the new wave of discovery with data science by collaborating closely with academic researchers and industrial data scientists. Stanford university giants Stephen Boyd, Trevor Hastie, and Rob Tibshirani advise the H2O team on building scalable machine learning algorithms. And with hundreds of meetups over the past several years, H2O continues to remain a word-of-mouth phenomenon.

Try it out

- Download H2O directly at `http://h2o.ai/download`.
- Install H2O's R package from CRAN at `https://cran.r-project.org/web/packages/h2o/`.
- Install the Python package from PyPI at `https://pypi.python.org/pypi/h2o/`.

Join the community

- To learn about our training sessions, hackathons, and product updates, visit `http://h2o.ai`.
- To learn about our meetups, visit `https://www.meetup.com/topics/h2o/all/`.
- Have questions? Post them on Stack Overflow using the **h2o** tag at `http://stackoverflow.com/questions/tagged/h2o`.
- Have a Google account (such as Gmail or Google+)? Join the open source community forum at `https://groups.google.com/d/forum/h2ostream`.
- Join the chat at `https://gitter.im/h2oai/h2o-3`.

Installation

At the time of this writing, Deep Water has not yet been officially released. So the three options for installing and/or using Deep Water are to build from source, to try out the H2O Deep Water Amazon Machine Image (AMI), or to run the H2O Docker Image.

Build from Source

Build instructions can be found here: `https://github.com/h2oai/deepwater`. Different build configurations can target different hardware and leverage various linear algebra libraries, including MKL, OpenBLAS, ATLAS, and CUDA.

Amazon Machine Image

For convenience, H2O.ai releases Deep Water AMIs as a way to try out Deep Water on GPU-enabled Amazon EC2 instances. We are constantly updating the AMIs. To get information on the latest AMI and how it use it, please visit the following: `https://github.com/h2oai/deepwater/blob/master/docs/open-tour-dallas/deep-water-ami.md`. For more information on AWS GPU instances, please visit the following: `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/accelerated-computing-instances.html`.

Docker Image

H2O has released a GPU-enabled Docker image on Docker Hub. To use this image, you must have a Linux machine with at least one GPU. Docker and nvidia-docker must also be installed. For more information on how to run the H2O Docker Image, please visit the following: `https://github.com/h2oai/deepwater/blob/master/README.md`.

Sample Data

The examples in this booklet use sample datasets located in a folder named **bigdata**. It's assumed that this folder resides in the folder currently running H2O. After cloning the h2o-3 repository, run the following command in the **h2o-3** folder to retrieve these datasets:

```
./gradlew syncBigdataLaptop
```

Note: For more information about building and running H2O-3, please visit the following: `https://github.com/h2oai/h2o-3#41-building-from-the-command-line-quick-start`

Citation

To cite this booklet, use the following:

Phan, W., Stensmo, M., Dymczyk, M., Candel, A. and Kou, Q. (Aug 2018). *Deep Learning with Deep Water*. http://h2o.ai/resources.

H2O Deep Water Overview

H2O Deep Water is the next generation deep learning addition to the H2O platform. H2O Deep Water supplements the existing H2O Deep Learning algorithm, which is a scalable, distributed, and in-memory implementation of multi-layer perception (MLP) deep learning networks.

H2O Deep Learning

For several years now, best-in-class deep learning has been part of the H2O platform, and the H2O deep learning algorithm remains one of the most used in the world. As with all H2O algorithms, H2O Deep Learning is optimized for speed and accuracy and is exposed via various adopted APIs and interfaces, including R, Python, Java, and web UI (H2O Flow). In addition, select features include:

- **Modern training options**: specifications for distributions (Bernoulli, Multinomial, Poisson, Gamma, Tweedie, Laplace, Huber, Quantile, Gaussian), loss functions (cross entropy, quadratic, absolute, Huber), learning rate, annealing, momentum, mini-batch size, and initialization
- **Automatic and flexible data handling to maximize productivity**: standardization, one-hot encoding, observation weights and offsets, class balancing, sampling factors, ignoring constant columns, sparse data handling, and input layer constraints
- **Tuning parameters to prevent model overfitting and efficient model development**: cross-validation, regularization, drop out, early stopping, model checkpointing, and hyperparameter search
- **Deep autoencoders for unsupervised learning**: deep features and anomaly detection

A complete treatment of H2O Deep Learning features can be found in our documentation at http://docs.h2o.ai/ and in the *Deep Learning with H2O* booklet at http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/DeepLearningBooklet.pdf [1].

Modern Trends in Deep Learning

Since the introduction of H2O Deep Learning, deep learning as a practice and science has changed significantly. Convolutional neural networks and recurrent neural networks, along with novel building blocks like Inception modules and residual networks, continue to demonstrate ground breaking results in many areas of artificial intelligence, including computer vision, speech, audio, and natural language processing. The depth and complexity of these modern network architectures ushered new algorithmic innovations and increased computational resources to train them. Today, the use of graphics processing units (GPU) for training deep neural networks has become more prominent, and the performance of GPU hardware continues to increase. A number of GPU-capable deep learning frameworks have emerged and maintain active development, including TensorFlow, MXNet, Caffe, Theano, and Torch.

Why H2O Deep Water?

H2O Deep Water is an extension of H2O Deep Learning and, as such, incorporates the modern trends in deep learning. In addition, Deep Water seeks to continue to make deep learning accessible for practicing data scientists and to drive value for enterprises. Deep Water offers:

- **Deep learning framework integration**: Deep Water leverages performant and scalable deep learning framework backends.
  - **TensorFlow, MXNet, and Caffe**: These are the initial targets of supported backends. Caffe support is still under development.
  - **GPU-accelerated training**: All backends allow for GPU-accelerated training while maintaining the option for CPU-based training.
  - **Modern deep learning architectures**: We offer easy-to-use predefined modern network architectures, such as VGG (see *Very Deep Convolutional Networks for Large-Scale Image Recognition* [8]) and ResNet (see *Deep Residual Learning for Image Recognition* [3]). At the same, custom-built or pre-trained networks can also be trained.
- **Machine learning platform**: Deep Water models can be compared against other world class H2O algorithms, such as gradient boosting machines. Deep Water models can also be ensembled along side other H2O models.
- **Ease of use and APIs**: Deep Water functionality is exposed via the H2O Flow Web UI and supported H2O APIs, including R, Python, and Java.

- **Deployment**: All Deep Water models can be deployed similarly to other H2O models. Specifically, Deep Water models can be exported as an H2O MOJO format, which can be consumed by any JVM-based languages. Additional language bindings can be added. For more information about MOJOs, please go here: `http://docs.h2o.ai/h2o/latest-stable/h2o-genmodel/javadoc/index.html`

Quick Start: MNIST Classification

The following example provides a quick start to using Deep Water. This example illustrates the API and shows that many of the capabilities from H2O Deep Learning are carried over to Deep Water. Using the MNIST handwritten digits data (see *The MNIST Database* [6]), this quick start example trains an MLP network using input drop out, cross-validation, early stopping, and GPU acceleration (default).

Example in Python

```
import h2o
from h2o.estimators.deepwater import
    H2ODeepWaterEstimator

# Start or connect to H2O
h2o.init()

# Import data and transform data
train = h2o.import_file("bigdata/laptop/mnist/train.
    csv.gz")

# Specify a subset of features to include in the model
features = list(range(0,784))
target = 784

train[target] = train[target].asfactor()

# Build model
model = H2ODeepWaterEstimator(epochs=100, activation="
    Rectifier", hidden=[200,200], ignore_const_cols=
    False, mini_batch_size=256, input_dropout_ratio
    =0.1, hidden_dropout_ratios=[0.5,0.5],
    stopping_rounds=3, stopping_tolerance=0.05,
```

```
    stopping_metric="misclassification",
    score_interval=2, score_duty_cycle=0.5,
    score_training_samples=1000,
    score_validation_samples=1000, nfolds=5, gpu=True,
     seed=1234)

model.train(x=features, y=target, training_frame=train
    )

# Evaluate model
model.show()
print(model.scoring_history())
```

Backends

By default, Deep Water uses the MXNet backend. We can change that by using the `backend` parameter.

Example in Python

```
model = H2ODeepWaterEstimator(epochs=100, activation="
    Rectifier", hidden=[200,200], ignore_const_cols=
    False, mini_batch_size=256, input_dropout_ratio
    =0.1, hidden_dropout_ratios=[0.5,0.5],
    stopping_rounds=3, stopping_tolerance=0.05,
    stopping_metric="misclassification",
    score_interval=2, score_duty_cycle=0.5,
    score_training_samples=1000,
    score_validation_samples=1000, nfolds=5, gpu=True,
     seed=1234, backend="tensorflow")
```

GPU and CPU

While GPU acceleration is the default, GPU computing is not required. Users can set `gpu=False` to fall back to CPU processing.

Example in Python

```
model = H2ODeepWaterEstimator(epochs=100, activation="
    Rectifier", hidden=[200,200], ignore_const_cols=
    False, mini_batch_size=256, input_dropout_ratio
    =0.1, hidden_dropout_ratios=[0.5,0.5],
    stopping_rounds=3, stopping_tolerance=0.05,
    stopping_metric="misclassification",
    score_interval=2, score_duty_cycle=0.5,
    score_training_samples=1000,
    score_validation_samples=1000, nfolds=5, gpu=False
    , seed=1234)
```

Using Deep Water with R

The examples for this booklet are done in Python, but an R API is also available for Deep Water.

Example in R

```
library(h2o)

# Start or connect to H2O
h2o.init()

# Import data and transform data
train <- h2o.importFile("bigdata/laptop/mnist/train.
    csv.gz")

target <- "C785"
features <- setdiff(names(train), target)

train[target] <- as.factor(train[target])

# Build model
```

```
15 model <-  h2o.deepwater(x=features, y=target, training
       _frame=train, epochs=100, activation="Rectifier",
       hidden=c(200,200), ignore_const_cols=FALSE, mini_
       batch_size=256, input_dropout_ratio=0.1, hidden_
       dropout_ratios=c(0.5,0.5), stopping_rounds=3,
       stopping_tolerance=0.05, stopping_metric="
       misclassification", score_interval=2, score_duty_
       cycle=0.5, score_training_samples=1000, score_
       validation_samples=1000, nfolds=5, gpu=TRUE, seed
       =1234)
16
17 # Evaluate model
18 summary(model)
```

Note that the rest of the booklet shows code snippets in Python to demonstrate Deep Water features. Complete examples of Jupyter notebooks can be found at `https://github.com/h2oai/h2o-3/tree/master/examples/deeplearning/notebooks`.

Image Classification

Data

Deep Water is able to consume standard H2O Frames with the following schema:

- **Data Set**: This is the same frame that any other H2O algorithm can consume, consisting of numeric and categorical (`enum`) features.
- **Image**: This is a two-column frame where one of the columns specifies the URI of images and the other column contains labels for supervised training.

The H2O Frame schema interpretation is set by the `problem_type` parameter. The user can explicitly set the parameter to `dataset` or `image` to interpret the frame in Data Set and Image schemas, respectively. By default, the `problem_type` parameter is set to `auto`, which specifies that Deep Water will auto-detect the schema.

Image Specification

A few key parameters can be specified for proper mapping of the input frame as an image:

- `image_shape`: A `List[int]` specifying the width and height of the image.
- `channels`: An `int` specifying the number of channels.
- `mean_image_file`: A string (`str`) specifying the path of the file containing the mean image data for data normalization.

Example in Python

```
import h2o
from h2o.estimators.deepwater import
    H2ODeepWaterEstimator

# Start or connect to H2O
h2o.init()

# Import data and transform data
train = h2o.import_file("bigdata/laptop/deepwater/
    imagenet/cat_dog_mouse.csv")

# Build model
model = H2ODeepWaterEstimator(epochs=10, network="
    lenet", problem_type="image", image_shape=[28,28],
     channels=3)

model.train(x=[0], y=1, training_frame=train)

# Evaluate model
model.show()
```

Pre-Defined Networks

Well known image classification pre-defined networks are built into Deep Water and can be used out of the box with the `network` parameter. These include:

- LeNet: Refer to *Gradient-Based Learning Applied to Document Recognition* [7]
- AlexNet: Refer to *ImageNet Classification with Deep Convolutional Neural Networks* [5]
- VGG: Refer to *Very Deep Convolutional Networks for Large-Scale Image Recognition* [8]
- GoogLeNet: Refer to *Going Deeper with Convolutions* [9]
- Inception-bn: Refer to *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* [4]
- ResNet: Refer to *Deep Residual Learning for Image Recognition* [3]

This list of `network` options will continue to grow.

User-Defined Networks

When the `network` parameter is set to `"user"`, users can define their own networks. User-defined or custom networks (graphs) are specified through the API of the native backend of choice, and it is assumed the user is familiar with their backend of choice. Networks are then saved and can be specified in the H2O Deep Water API via the `network_definition_file` parameter. Passing user-defined networks to H2O Deep Water is the same as importing pre-trained networks without specifying any network parameters (e.g. weights, biases). See Section 6.5 for code examples.

MXNet

To specify networks with MXNet, use the `mxnet.symbol` API and `Symbol` class. (Refer to `http://mxnet.io/api/python/symbol.html`.) MXNet networks are saved as a file via the `Symbol.save` method.

TensorFlow

To specify networks with TensorFlow, use the `tf.Graph` class or any high level API, such as Keras (`https://keras.io/`). TensorFlow networks are saved with graph collections, with the `tf.train.Saver` class (see `https://www.tensorflow.org/programmers_guide/variables` and `https://www.tensorflow.org/api_docs/python/tf/train/Saver`), and with the `tf.train.export_meta_graph()` method.

Pre-Trained Networks

Importing pre-trained networks requires specifying the `network_defintion_file` (network/graph information) and `network_parameters_file` (e.g. weights, biases) parameters.

MXNet

As previously mentioned, networks are trained in MXNet with the `mxnet.module` API and `Module` class. Once trained, network parameters can be saved via the `Module.save_params` method. Along with the network graph file, the network parameters can be loaded into Deep Water as shown below.

Example in Python

```
model = H2ODeepWaterEstimator(epochs=100, image_shape
    =[28,28], backend="mxnet", network="user",
    network_definition_file="/path/to/lenet.json",
    network_parameters_file="/path/to/lenet-100epochs-
    params.txt")
```

TensorFlow

The `tf.train.Saver` class also saves the network parameters. It can be loaded into Deep Water as shown below.

Example in Python

```
model = H2ODeepWaterEstimator(epochs=100, image_shape
    =[28,28], backend="tensorflow", network="user",
    network_definition_file="/path/to/lenet_28x28x3_3.
    meta", network_parameters_file="/path/to/lenet-100
    epochs")
```

H2O Flow (Web UI)

Deep Water can be accessed through the H2O Flow Web UI. Data sets can be imported and parsed as shown in Figures 1 and 2. Figure 3 shows how you can view the data after parsing. Note that in the example shown, the data is in the image schema described in section 6.1.

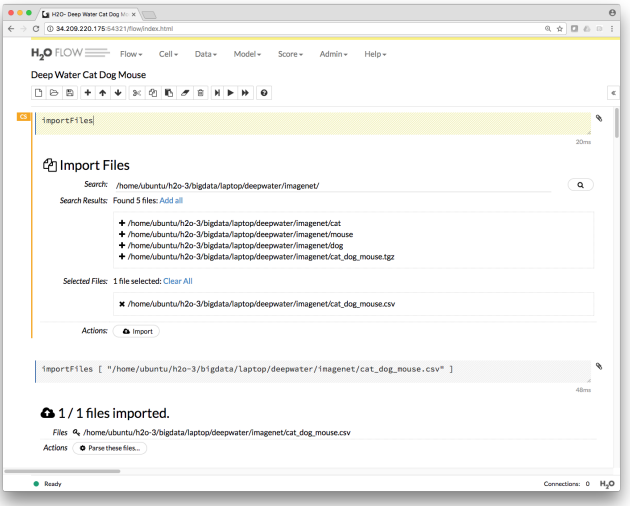

Figure 1: Import data

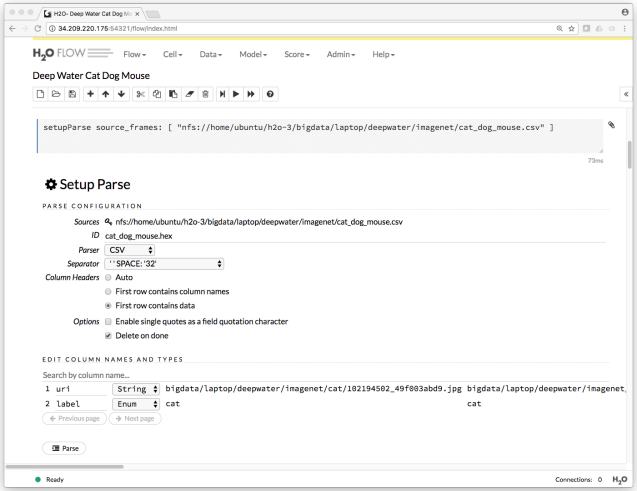

Figure 2: Parse data

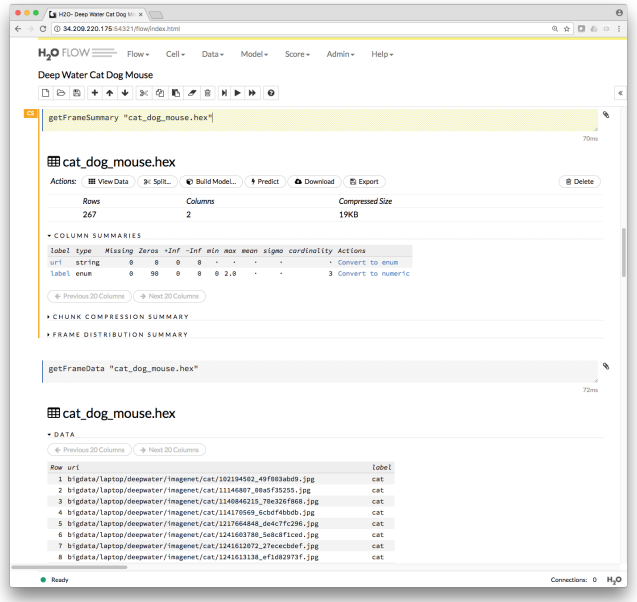

Figure 3: View data

A Deep Water model is built just like any other H2O algorithm as shown in Figure 4. In this example, we use a simple LeNet pre-defined network. (Refer to *Gradient-Based Learning Applied to Document Recognition* [7].) Best practice defaults are set for all parameters. Figures 5 and 6 highlight the key backend and GPU selection parameters in the Deep Water Flow configuration, respectively.

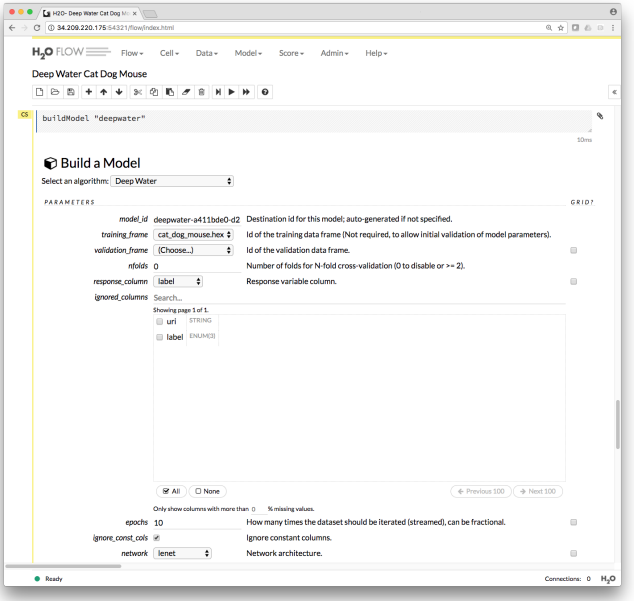

Figure 4: Build Deep Water model

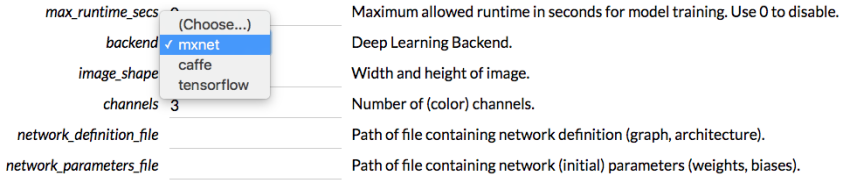

Figure 5: Deep Water backend options

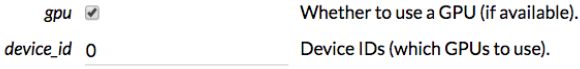

Figure 6: Deep Water GPU selection

Grid Search

H2O's grid search API can be used with Deep Water. Grid search allows users to specify sets of values for parameter arguments and observe changes in model behavior. This is useful for hyperparameter tuning. For all grid searches, the type of search and early stopping can be configured to stop searches if there is no substantial metric improvement in searches after successive rounds. Search criteria (`search_criteria`) are passed as a dictionary to the grid search class:

- `strategy`: Specify `"Cartesian"` (default), `"RandomDiscrete"`
- `stopping_metric`: Specify the metric to use for early stopping.
- `stopping_rounds`: Specify early stopping based on convergence of the `stopping_metric`. Stop if the simple moving average of length k of the `stopping_metric` does not improve from k `stopping_rounds` scoring events. (Use 0 to disable.)
- `stopping_tolerance`: Specify relative tolerance for metric-based stopping criterion. (Stop if relative improvement is not at least this much.)

You can read more about grid search in the **Hyperparameter Optimization in H2O** blog at `https://blog.h2o.ai/2016/06/hyperparameter-optimization-in-h2o-grid-search-random-search-and-the-future/`.

Cartesian Search

A cartesian grid search will run a model for each combination of parameters in the grid. In the example below, two sets of hidden layers and two learning rates are specified in the grid, which will result in four models being built.

Example in Python

```
# Import data and transform data
train = h2o.import_file("bigdata/laptop/mnist/train.csv.gz")

features = list(range(0,784))
target = 784

train[target] = train[target].asfactor()

# Set up grid
hidden_opt = [[200,200], [1024,1024]]
learn_rate_opt = [1e-6, 1e-5]
hyper_parameters = {"hidden": hidden_opt, "learning_rate":learn_rate_opt}

# Build model and train model grid
from h2o.grid.grid_search import H2OGridSearch
model_grid = H2OGridSearch(H2ODeepWaterEstimator, hyper_params=hyper_parameters)

model_grid.train(x=features, y=target, training_frame=train, epochs=100, activation="Rectifier", ignore_const_cols=False, mini_batch_size=256, input_dropout_ratio=0.1, hidden_dropout_ratios=[0.5,0.5], stopping_rounds=3, stopping_tolerance=0.05, stopping_metric="misclassification", score_interval=2, score_duty_cycle=0.5, score_training_samples=1000, score_validation_samples=1000, nfolds=5, gpu=True, seed=1234)

# Evaluate model
print(model_grid)
```

Random Search

The hyperparameter search space can become too large to compute exhaustively. Given a fixed amount of time, making random choices of hyperparameter values can give results that are on par with or even better than the best results of a Cartesian search. (See *Random Search for Hyper-parameter Optimization* [2].) This example expands the search space for hidden layers and learning rate and adds a parameter for input dropout. The max search time is set to five minutes.

Example in Python

```
# Set up grid
hidden_opt = [[200,200], [1024,1024], [1024,1024,2048], [200,200,200], [300,300]]
learn_rate_opt = [1e-6, 1e-5, 1e-3, 5e-3]
in_drop_opt = [0.1, 0.2, 0.3]
hyper_parameters = {"hidden": hidden_opt, "learning_rate":learn_rate_opt, "input_dropout_ratio": in_drop_opt}

search_criteria = {"strategy": "RandomDiscrete", "max_models": 10, "max_runtime_secs": 300, "seed": 1234}

# Build model and train model grid
from h2o.grid.grid_search import H2OGridSearch
model_grid = H2OGridSearch(H2ODeepWaterEstimator, hyper_params=hyper_parameters, search_criteria=search_criteria)

model_grid.train(x=features, y=target, training_frame=train, epochs=100, activation="Rectifier", ignore_const_cols=False, mini_batch_size=256, hidden_dropout_ratios=[0.5,0.5], stopping_rounds=3, stopping_tolerance=0.05, stopping_metric="misclassification", score_interval=2, score_duty_cycle=0.5, score_training_samples=1000, score_validation_samples=1000, nfolds=5, gpu=True, seed=1234)

# Evaluate model
print(model_grid)
```

Model Checkpoints

Model checkpoints are useful in saving models (i.e. training state) for long training runs or to resume model training, sometimes with different parameters. In the example below, a model is trained for 20 epochs and then saved via the `h2o.save_model` method. The model is then restored via the `h2o.load_model` method, and training is resumed.

Example in Python

```
# Import data and transform data
train = h2o.import_file("bigdata/laptop/mnist/train.csv.gz")
valid = h2o.import_file("bigdata/laptop/mnist/test.csv.gz")

features = list(range(0,784))
target = 784

train[target] = train[target].asfactor()
valid[target] = valid[target].asfactor()

# Build model
model = H2ODeepWaterEstimator(epochs=20, activation="Rectifier", hidden=[200,200], ignore_const_cols=False, mini_batch_size=256, input_dropout_ratio=0.1, hidden_dropout_ratios=[0.5,0.5], stopping_rounds=3, stopping_tolerance=0.05, stopping_metric="misclassification", score_interval=2, score_duty_cycle=0.5, score_training_samples=1000, score_validation_samples=1000, gpu=True, seed=1234)

model.train(x=features, y=target, training_frame=train, validation_frame=valid)

# Evaluate model
model.show()
print(model.scoring_history())

# Checkpoint model
```

```
model_path = h2o.save_model(model=model, force=True)

# Load model
model_ckpt = h2o.load_model(model_path)

# Start training from checkpoint
model_warm = H2ODeepWaterEstimator(checkpoint=
    model_ckpt.model_id, epochs=100, activation="
    Rectifier", hidden=[200,200], ignore_const_cols=
    False, mini_batch_size=256, input_dropout_ratio
    =0.1, hidden_dropout_ratios=[0.5,0.5],
    stopping_rounds=3, stopping_tolerance=0.05,
    stopping_metric="misclassification",
    score_interval=2, score_duty_cycle=0.5,
    score_training_samples=1000,
    score_validation_samples=1000, gpu=True, seed
    =1234)

model_warm.train(x=features, y=target, training_frame=
    train, validation_frame=valid)

# Evaluate checkpointed model
model_warm.show()
print(model_warm.scoring_history())
```

Ensemble

Deep Water models can be ensembled with other models built with H2O, leveraging the rich algorithmic capabilities of the H2O machine learning platform. Below, three base learners are built with 5-fold cross-validation: GBM, GLM, and Deep Water. The base learners are then ensembled together via the stacking method. You can read more about stacking here: `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html`.

Example in Python

```
import h2o
from h2o.estimators.deepwater import
    H2ODeepWaterEstimator
from h2o.estimators.gbm import
    H2OGradientBoostingEstimator
from h2o.estimators.glm import
    H2OGeneralizedLinearEstimator
from h2o.estimators.stackedensemble import
    H2OStackedEnsembleEstimator

# Import data
train = h2o.import_file("/path/to/train-odd.csv.gz",
    destination_frame="train.hex")
valid = h2o.import_file("/path/to/test-odd.csv.gz",
    destination_frame="valid.hex")

features = list(range(0,784))
target = 784

train[features] = train[features]/255
train[target] = train[target].asfactor()
valid[features] = valid[features]/255
valid[target] = valid[target].asfactor()

nfolds = 5

# GBM Model
gbm_model = H2OGradientBoostingEstimator(distribution=
    "bernoulli", ntrees=100, nfolds=nfolds,
    ignore_const_cols=False,
```

```
    keep_cross_validation_predictions=True,
    fold_assignment="Modulo")
gbm_model.train(x=features, y=target, training_frame=
    train, model_id="gbm_model")
gbm_model.show()

# GLM Model
glm_model = H2OGeneralizedLinearEstimator(family="
    binomial", lambda_=0.0001, alpha=0.5, nfolds=
    nfolds, ignore_const_cols=False,
    keep_cross_validation_predictions=True,
    fold_assignment="Modulo")
glm_model.train(x=features, y=target, training_frame=
    train, model_id="glm_model")
glm_model.show()

# Deep Water Model
dw_model = H2ODeepWaterEstimator(epochs=3, network="
    lenet", ignore_const_cols=False, image_shape
    =[28,28], channels=1, standardize=False, seed
    =1234, nfolds=nfolds,
    keep_cross_validation_predictions=True,
    fold_assignment="Modulo")
dw_model.train(x=features, y=target, training_frame=
    train, model_id="dw_model")
dw_model.show()

# Stacked Ensemble
stack_all = H2OStackedEnsembleEstimator(base_models=[
    gbm_model.model_id, glm_model.model_id, dw_model.
    model_id])
stack_all.train(x=features, y=target, training_frame=
    train, validation_frame=valid, model_id="stack_all
    ")
stack_all.model_performance()
```

Deep Features and Similarity

The hidden layers of a trained model can provide a useful feature representation of input data. A Deep Water model's `deepfeatures` method allows you to extract hidden layer feature representations of input data. These extracted feature representations can be used in several ways. In the example below, features are extracted from a layer of a pre-trained convolutional network. (Refer to *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* [4].) The extracted features are then used to train a multinomial GLM model.

Example in Python

```
# Load network
network_model = H2ODeepWaterEstimator(epochs=0,
    mini_batch_size=32, network="user",
    network_definition_file="Inception_BN-symbol.json"
    , network_parameters_file="Inception_BN-0039.
    params", mean_image_file="mean_224.nd",
    image_shape=[224,224], channels=3)

network_model.train(x=[0], y=1, training_frame=train)

# Extract deep features
extracted_features = network_model.deepfeatures(train,
     "global_pool_output")
print("shape: " + str(extracted_features.shape))
print(extracted_features[:5,:3])

# Merge deep features with target and split frame
extracted_features["target"] = train[1]
features = [x for x in extracted_features.columns if x
     not in ["target"]]
train, valid = extracted_features.split_frame(ratios
    =[0.8])

# Build multinomial GLM
glm_model = H2OGeneralizedLinearEstimator(family="
    multinomial")
glm_model.train(x=features, y="target", training_frame
    =train, validation_frame=valid)

```

```
# Evaluate model
glm_model.show()
```

```
(267, 1024)
  DF.global_pool_output.C1    DF.global_pool_output.C2    DF.global_pool_output.C3
--------------------------  --------------------------  --------------------------
                  0.801623                     0.42203                    0.416217
                   1.09336                    0.704138                    0.420898
                  0.594622                    0.161074                    0.357225
                  0.875428                    0.865322                    0.532098
                   1.11859                    0.625728                    0.348317

[5 rows x 3 columns]
```

Another use of hidden layer feature representation is for unsupervised applications, such as clustering or recommendations. The deep features are used as vector representations whereby similarity measures can be computed. Given two H2OFrames `X` and `Y`, the following will compute a resultant H2OFrame whereby a similarity measure, specified by the `similarity` parameter, is computed for each vector in `X` and `Y`: `X.distance(Y, similarity)`.

We can express this mathematically.

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \dots & x_{1,P} \\ \vdots & \ddots & \vdots \\ x_{N,1} & \dots & x_{N,P} \end{bmatrix}, \text{ where } \mathbf{x}_i = [x_{i,1}, \dots, x_{i,P}]$$

$$\mathbf{Y} = \begin{bmatrix} y_{1,1} & \dots & y_{1,P} \\ \vdots & \ddots & \vdots \\ y_{M,1} & \dots & y_{M,P} \end{bmatrix}, \text{ where } \mathbf{y}_i = [y_{i,1}, \dots, y_{i,P}]$$

$\mathbf{distance}(\mathbf{X}, \mathbf{Y}) = \mathbf{Z} : z_{i,j} = \mathbf{similarity}(\mathbf{x}_i, \mathbf{y}_j)$, where $\mathbf{X} \in \mathbb{R}^{N \times P}, \mathbf{Y} \in \mathbb{R}^{M \times P}, \mathbf{Z} \in \mathbb{R}^{N \times M}$

The following are the various similarity measures that can be computed.

ℓ_1 similarity (`"l1"`): $z_{i,j} = \sum_{k=1}^{P} |x_{i,k} - y_{j,k}|$

ℓ_2 similarity (`"l2"`): $z_{i,j} = \sqrt{\sum_{k=1}^{P} (x_{i,k} - y_{j,k})^2}$

cosine similarity (`"cosine"`): $z_{i,j} = \frac{\mathbf{x}_i \cdot \mathbf{y}_j}{||\mathbf{x}_i||_2 ||\mathbf{y}_j||_2} = \frac{\sum_{k=1}^{P} x_{i,k} y_{j,k}}{\sqrt{\sum_{k=1}^{P} x_{i,k}^2} \sqrt{\sum_{k=1}^{P} y_{j,k}^2}}$

cosine squared similarity (`"cosine_sq"`): $z_{i,j} = \left(\frac{\mathbf{x}_i \cdot \mathbf{y}_j}{||\mathbf{x}_i||_2 ||\mathbf{y}_j||_2} \right)^2$

The following code snippet uses the same extracted features from the previous example. This time, the extracted features frame is split into two frames, the first three rows/vectors become a `queries` frame, and the rest of the rows/vectors are assigned to a `references` frame. A similarity frame is created between the `references` and `queries` frames, where each element $x_{i,j}$ is the similarity measure between `reference` vector i and `queries` vector j.

Example in Python

```
# Seperate records to a references and queries
references = extracted_features[5:,:]
queries = extracted_features[:3,:]

# Compute similarity
similarity = references.distance(queries, "cosine")

# Verify shapes
print("references: " + str(references.shape))
print("queries: " + str(queries.shape))
print("similarity: " + str(similarity.shape))

# View similarity frame
print(similarity.head())
```

The following is the output of the code snippet.

```
references: (262, 1024)
queries: (3, 1024)
similarity: (262, 3)
         C1           C2           C3
-----------  -----------  -----------
0.000700166  0.000890456  0.00115243
0.000714771  0.000971895  0.00114015
0.000725556  0.000886771  0.00108941
0.000583118  0.000677621  0.000848235
0.000709113  0.00075652   0.000968125
0.000779529  0.00103488   0.00124044
0.000725078  0.00103037   0.00122527
0.00077362   0.000987806  0.00126681
0.000733625  0.000879774  0.00120423
0.000823687  0.000976036  0.00123983

[10 rows x 3 columns]
```

Multi-GPU

Multi-GPU support is available through backend-specific mechanisms. For example, in TensorFlow, multi-GPU specification can be done through the computational graph. For examples, please visit: `https://github.com/h2oai/h2o-3/tree/master/examples/deeplearning/notebooks`.

Deployment for Inference

Model Object Optimized (MOJO)

With H2O, you can convert your deep water models into a binary model object optimized (MOJO) formats. This format is easily embeddable in any Java environment and independent of an H2O cluster. The only compilation and runtime dependencies for generated models are the `h2o-genmodel.jar` and the `deepwater-all.jar` files, which are produced as part of the build output. Deep Water models can be exported as a MOJO and embedded in a custom Java application. You can read more about MOJOs here: `http://docs.h2o.ai/h2o/latest-stable/h2o-genmodel/javadoc/index.html`.

Deep Water MOJOs can be downloaded from H2O Flow by clicking **Download Model Deployment Package** from a Deep Water model. (See Figure 7.) From the Python API, you can use the `download_mojo` method for a model. For example:

```
model.download_mojo(path="/path/to/model_mojo",
                    get_genmodel_jar=True)
```

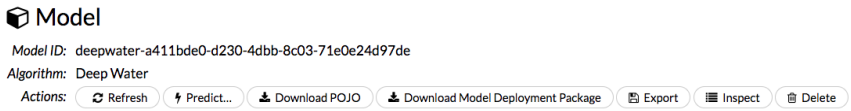

Figure 7: Deep Water model actions

Prediction Service Builder

The H2O Prediction Service Builder is a standalone web service application that can help users compile MOJOs and build Web Archive (War) files for prediction web services. The details of how to build the H2O Prediction Service Builder can be found here: `https://github.com/h2oai/steam/tree/master/prediction-service-builder`.

Before generating a War file, be sure that you have both the `h2o-genmodel.jar` and `deepwater-all.jar` files. You can obtain each of these by running the following:

```
curl localhost:54321/3/h2o-genmodel.jar > h2o-genmodel.jar
curl localhost:54321/3/deepwater-all.jar > deepwater-all.jar
```

War files can be generated using the Prediction Service Builder Web UI or via command line. For example, submitting the following command submits the necessary dependencies to the Prediction Server Builder (running on localhost on port 55000) to create an `example.war` file.

```
curl -X POST \
--form mojo=@mojo.zip \
--form jar=@h2o-genmodel.jar \
--form deepwater=@deepwater-all.jar \
localhost:55000/makewar > example.war
```

The `example.war` file can be started using an appropriate Jetty runner. For example, the following command starts the prediction service on port 55001:

```
java -jar jetty-runner-9.3.9.M1.jar --port 55001 example.war
```

Upon completion, a prediction service for scoring will be available at `http://localhost:55001`.

Upcoming

At the time of this writing, we have many exciting upcoming releases and initiatives at H2O.ai.

- **Machine Learning and GPUs**: H2O.ai has developed the fastest scalable, distributed in-memory machine learning platform, and we now extend its capabilities to GPUs, aiming to create the fastest artificial intelligence platform on GPUs. Stay tuned for more of our algorithms exploiting GPU-acceleration.
- **Automatic Machine Learning**: H2O AutoML is an automatic machine learning capability that will encapsulate and automate best practices in data cleaning, feature engineering, hyper-parameter search, and ensemble generation.
- **Machine Learning Interpretability**: Often times, especially in regulated industries, model transparency and explanation become just as paramount as predictive performance. Through visualizations and various techniques, machine learning interpretability functionality will continually make its way to the H2O platform. For details on the ideas around machine learning interpretability, please visit: `https://www.oreilly.com/ideas/ideas-on-interpreting-machine-learning`.

Acknowledgements

We would like to acknowledge the following individuals for their contributions to and review of this booklet: Jo-Fai (Joe) Chow, Megan Kurka, Erin LeDell, Ray Peck, Patrick Hall, and Surekha Jadhwani.

Errata

This version of H2O Deep Water is still a pre-release version. An errata document is available, describing current known issues that you might encounter when trying out Deep Water. This document is available in the h2o-3 GitHub repo at `https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/booklets/source/DeepWaterBookletErrata.md`.

If the Errata document does not answer your question, feel free to post your question to Stack Overflow using the **h2o** tag at `http://stackoverflow.com/questions/tagged/h2o`.

References

1. A. Arora, A. Candel, J. Lanford, E. LeDell, , and V. Parmar. **Deep Learning with H2O**, August 2015. URL http://h2o.ai/resources
2. James Bergstra and Yoshua Bengio. **Random Search for Hyper-parameter Optimization**. *J. Mach. Learn. Res.*, 13:281–305, February 2012. ISSN 1532-4435. URL http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf
3. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep Residual Learning for Image Recognition**. 2015. URL https://arxiv.org/pdf/1512.03385.pdf
4. Sergey Ioffe and Christian Szegedy. **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**. 2015. URL https://arxiv.org/pdf/1502.03167.pdf
5. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. **ImageNet Classification with Deep Convolutional Neural Networks**. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012. URL https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
6. Yann LeCun, Corinna Cortes, and Christopher J.C Burgres. **The MNIST Database**. URL http://yann.lecun.com/exdb/mnist/
7. Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. **Gradient-Based Learning Applied to Document Recognition**. In *Proceedings of the IEEE*, Nov. 1998. URL http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
8. Karen Simonyan and Andrew Zisserman. **Very Deep Convolutional Networks for Large-Scale Image Recognition**. 2014. URL http://arxiv.org/pdf/1207.0580.pdf
9. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. **Going Deeper with Convolutions**. 2014. URL https://arxiv.org/pdf/1409.4842.pdf

Authors

Wen Phan

Wen Phan is a senior solutions architect at H2O.ai. Wen works with customers and organizations to architect systems, smarter applications, and data products to make better decisions, achieve positive outcomes, and transform the way they do business. Wen holds a B.S. in electrical engineering and M.S. in analytics and decision sciences. Follow him on Twitter: @wenphan

Magnus Stensmo

Magnus has been a scientist and software engineer in workplaces ranging from new startups to large companies. Using machine learning and information retrieval, he has built numerous systems for internet and enterprise software companies, including large scale real-time similarity search, product recommendation systems, text analysis, and medical and machine diagnosis. Magnus holds a PhD in Computer Science and an MSc in Computer Science and Engineering from the Royal Institute of Technology (KTH) in Stockholm, Sweden. He has also studied neuroscience, philosophy, business, and linguistics.

Mateusz Dymczyk

Mateusz is a software developer who loves all things distributed, machine learning, and data juggling. He obtained his M.Sc. in Computer Science from AGH UST in Krakow, Poland, during which he did an exchange at LECE Paris in France and worked on distributed flight booking systems. After graduation, he moved to Tokyo, where he is still currently based, to work as a researcher at Fujitsu Laboratories on machine learning and NLP projects. In his spare time, he tries to be part of the IT community by organizing, attending, and speaking at conferences and meet ups. Follow him on Twitter: @mdymczyk.

Arno Candel

Arno is the Chief Technology Officer of H2Oa.ai. He is also the main author of H2O Deep Learning. Arno holds a PhD and a Masters summa cum laude in Physics from ETH Zurich, Switzerland. He has authored dozens of scientific papers and is a sought-after conference speaker. Arno was named 2014 Big Data All-Star by Fortune Magazine. Follow him on Twitter: @ArnoCandel.

Qiang Kou

Qiang Kou was an early developer for Deep Water. He is currently pursuing his PhD in informatics at Indiana University. Qiang is also an Rcpp core team member and Apache MXNet committer. Follow him on Twitter: @KKusingR