Приложение 1

Deep Water

Установка

На момент написания данного текста библиотека Deep Water не была официально представлена, поэтому есть всего три варианта ее установки: сборка из исходных кодов, использование H2O Deep Water Amazon Machine Image (AMI) и запуск образа Docker.

Сборка из исходных кодов

Инструкцию можно найти по ссылке https://github.com/h2oai/deepwater. Доступны варианты сборки под разные платформы; можно использовать различные библиотеки линейной алгебры, включая MKL, OpenBLAS и ATLAS, а также CUDA.

Amazon Machine Image

Для удобства компания H2O.ai создала Deep Water AMI, которые можно устанавливать на GPU-серверы Amazon. Эти образы постоянно обновляются. Актуальную информацию можно найти по ссылке http://bit.ly/2viVKmw. Дополнительные сведения о GPU-серверах Amazon доступны на странице http://amzn.to/1OpWzIO.

Образ Docker

Образ Docker с поддержкой GPU размещен на Docker Hub. Для его использования нужен ПК под управлением ОС семейства Linux с хотя бы одной видеокартой и установленными приложениями Docker и nvidia-docker. Дополнительную информацию можно найти на странице https://github.com/h2oai/deepwater/blob/master/README.md.

Примеры данных

Примеры, рассмотренные в этом руководстве, подразумевают наличие наборов данных в папке *bigdata*, которая расположена в той же папке, откуда запущена библиотека H2O. После клонирования репозитория https://github.com/h2oai/deepwater на свой ПК запустите команду `./gradlew syncBigdataLaptop` (Linux) или `gradlew syncBigdataLaptop` (Windows), чтобы запустить скачивание данных.

Обзор библиотеки Deep Water

Deep Water представляет собой дополнение для библиотеки H2O, позволяющее использовать алгоритмы глубокого обучения помимо имеющейся в самой биб-

лиотеке масштабируемой распределенной реализации многослойного перцептрона (которая работает с данными, помещающимися в ОЗУ).

Глубокое обучение в библиотеке H2O

В библиотеке H2O реализован быстрый и точный алгоритм глубокого обучения, который широко используется в мире. Как и остальные методы машинного обучения в H2O, он обладает высокой скоростью работы и доступен при помощи интерфейсов для языков R, Python и Java, а также через веб-интерфейс Flow. Ключевые особенности:

- настройки процесса обучения модели: можно выбирать вид распределения (распределение Бернулли, мультиномиальное, пуассоновское, гамма-распределение, распределение Твиди, Лапласа, Хьюбера, квантильное и нормальное распределения), функцию потерь (перекрестная энтропия, квадратичная и абсолютная ошибки, функция потерь Хьюбера), скорость обучения, уменьшение скорости обучения, импульс, размер мини-выборки, способ инициализации;
- автоматическая и гибкая обработка данных: стандартизация, бинарное кодирование, указание весов наблюдений, балансировка классов, исключение переменных с нулевой дисперсией, обработка разреженных данных;
- настройки для борьбы с переобучением: перекрестная проверка, регуляризация, дропаут, ранняя остановка, контрольные точки, подбор значений гиперпараметров;
- глубокие автокодировщики для обучения на неразмеченных данных: создание признаков (использование выходных значений нейронов скрытых слоев), определение аномалий.

Полный перечень возможностей можно найти в официальной документации и в буклете http://bit.ly/2rnigXd.

Современные тенденции в глубоком обучении

Глубокое обучение как практическая дисциплина и как область знаний значительно изменилось с момента начала разработки Deep Water. Сверточные и рекуррентные нейронные сети (в том числе блоки, используемые в архитектуре Inception, и остаточные (residual) нейросети) ставят рекорды во многих областях ИИ, включая компьютерное зрение и обработку звуков, речи и естественного языка. Использование GPU для обучения таких сложных нейронных сетей выглядит многообещающим, и производительность GPU продолжает расти. Появилось и развивается множество фреймворков для глубокого обучения с поддержкой GPU, в том числе TensorFlow, MXNet, Caffe, Theano и Torch.

Почему нужно использовать Deep Water

Deep Water является дополнением к библиотеке H2O и предоставляет следующие возможности:

- интеграция с производительными и масштабируемыми фреймворками для глубокого обучения (TensorFlow, MXNet, Caffe), поддержка использования GPU, современные нейросетевые архитектуры (такие как VGG, ResNet), возможность обучения собственных или предварительно обученных сетей;

- платформа для машинного обучения: модели Deep Water можно сравнивать с другими моделями, созданными в H2O, а также объединять их в ансамбли;
- простота использования, наличие API для R, Python и Java, доступ через веб-интерфейс Flow;
- развертывание: модели Deep Water можно использовать точно так же, как и другие модели, созданные при помощи H2O. В частности, они могут быть экспортированы в формате MOJO, который используется в основанных на JVM языках. Также есть возможность добавления «оберток» для других языков. Дополнительную информацию можно найти по ссылке http://bit.ly/2hprGjY.

Начало работы: набор данных MNIST

Следующий пример описывает начало работы с Deep Water. В нем показаны работа с API и применение возможностей основной библиотеки H2O. Используется набор данных MNIST; обучается многослойная нейронная сеть с дропаутом для слоя входных данных, перекрестной проверкой, ранней остановкой и использованием GPU (по умолчанию).

Пример на языке Python:

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

# Запуск или соединение с H2O
h2o.init()

# Импортирование и преобразование данных
train = h2o.import_file("bigdata/laptop/mnist/train.csv.gz")

# Набор признаков для включения в модель
features = list(range(0, 784))
target = 784

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

# Создание модели
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)

# Оценка качества модели
model.show()
print(model.scoring_history())
```

Бекенды

По умолчанию в качестве бекенда используется mxnet. Другой вариант можно выбрать при помощи параметра `backend`.

Пример на языке 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")
```

CPU и GPU

GPU используются для вычислений по умолчанию, но это не является необходимым. Задав параметр `gpu = False`, можно запустить вычисления только с использованием CPU.

Пример на языке 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)
```

Использование Deep Water с R

В этом руководстве приведены примеры с использованием языка Python, но также доступно API для R.

Пример на языке R:

```
library(h2o)

# Запуск или соединение с H2O
h2o.init()

# Импортирование и преобразование данных
train <- h2o.importFile("bigdata/laptop/mnist/train.csv.gz")

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

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

# Создание модели
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)

# Оценка качества модели
summary(model)
```

Полные версии следующих примеров на языке Python можно найти в Jupyter-ноутбуках по ссылке http://bit.ly/2f9KCSS.

Классификация изображений

Данные

Библиотека Deep Water позволяет работать со стандартными таблицами H2O в следующих форматах:

- набор данных – обычная таблица с числовыми и категориальными признаками;
- изображения – таблица с двумя столбцами, в которой первый столбец содержит ссылки на изображения, а второй – значения целевой переменной (метки классов для задач классификации) для обучения на размеченных данных.

Формат задается при помощи параметра `problem_type` с возможными значениями «`dataset`», «`image`» или «`auto`» (вариант по умолчанию, при котором решение принимается автоматически).

Параметры изображений

Для корректной обработки изображений могут быть указаны следующие основные параметры:

- `image_shape` – список целых чисел, которые задают ширину и высоту изображения;
- `channels` – целое число, соответствующее количеству каналов (3 для изображений в RGB-формате);
- `mean_image_file` – строка, содержащая путь к усредненному изображению, которое используется для нормализации данных.

Пример на языке Python:

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

# Запуск или соединение с H2O
h2o.init()

# Импортирование и преобразование данных
train = h2o.import_file("bigdata/laptop/deepwater/imagenet/cat_dog_mouse.csv")

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

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

# Оценка качества модели
model.show()
```

Предварительно созданные архитектуры

В готовом виде доступны следующие известные архитектуры для классификации изображений:

- LeNet;
- AlexNet;
- VGG;
- GoogLeNet;
- Inception-bn;
- ResNet.

Список будет пополняться.

Архитектуры, создаваемые пользователем

Если задать параметр `network = "user"`, пользователь может сам определять архитектуру используемой нейросети. Создание архитектуры (графа вычислений) происходит с использованием API библиотеки, которая выступает в роли бекенда. Нейросеть сохраняется и затем может быть загружена при помощи параметра `network_definition_file`. Процесс аналогичен использованию предварительно обученной сети, но при этом не указываются параметры нейросети (веса и смещения). Примеры приведены в разделе 4.5.

Mxnet: используются API `mxnet.symbol` и класс `Symbol` (http://bit.ly/2hqnkZT). За сохранение нейросети отвечает метод `Symbol.save`.

Tensorflow: используется класс `tf.Graph` или высокоуровневый API, например библиотека Keras (https://keras.io/). Сохранение происходит при помощи класса `tf.train.Saver` (см. http://bit.ly/2ldnpgE и http://bit.ly/2fatrki) и метода `tf.train.export_meta_graph()`.

Предварительно обученные нейросети

В Deep Water могут быть загружены параметры нейросети (веса и смещения) вместе с описанием архитектуры, как показано ниже.

Пример на языке Python (mxnet):

```
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-100epochsparams.txt")
```

Пример на языке Python (Tensorflow):

```
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-100epochs")
```

Веб-интерфейс Flow

Доступ к функциям Deep Water можно получить также посредством веб-интерфейса Flow, как показано на следующих иллюстрациях.

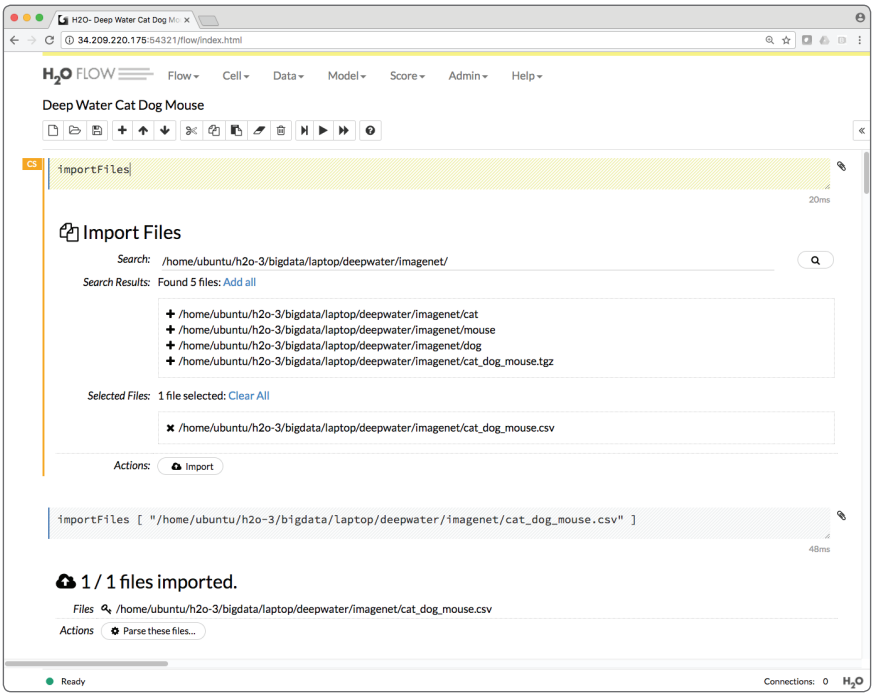

Рис. П.1 ❖ Импортирование данных

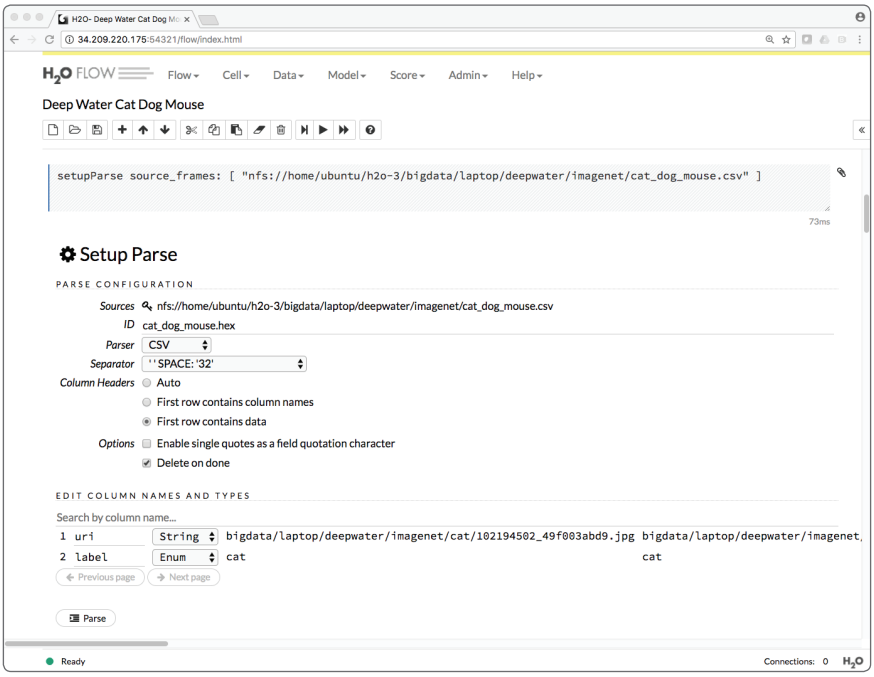

Рис. П.2 ❖ Парсинг данных

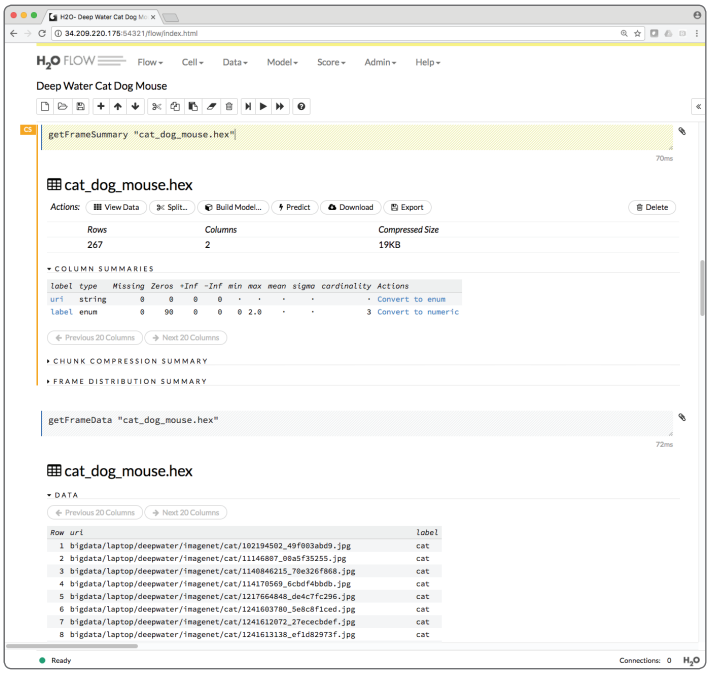

Рис. П.3 ❖ Просмотр данных

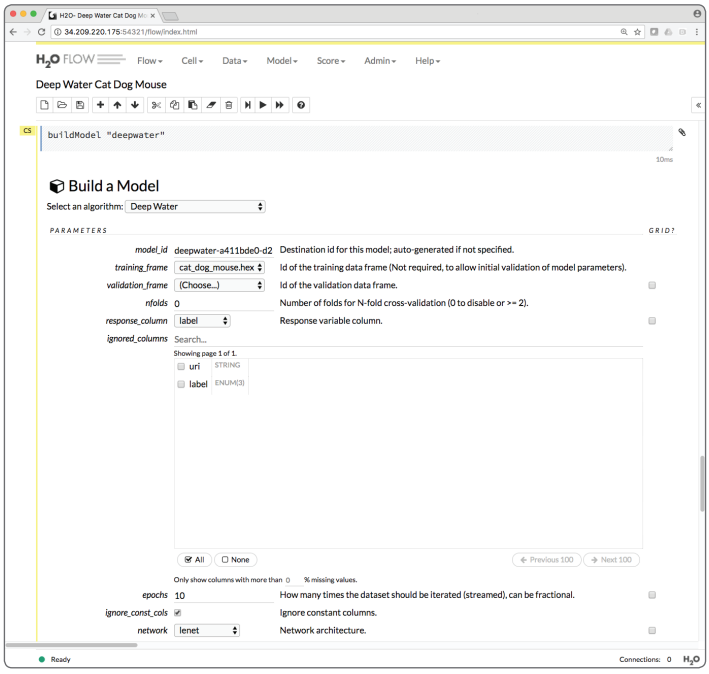

Рис. П.4 ❖ Создание модели

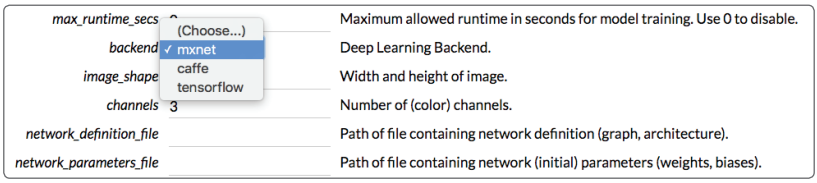

Рис. П.5 ❖ Выбор бекенда

Рис. П.6 ❖ Использование GPU

Поиск по сетке

Deep Water позволяет использовать возможности подбора значений гиперпараметров путем поиска по сетке, реализованные в библиотеке H2O. Можно указывать тип поиска и настраивать раннюю остановку. Все настройки передаются параметру `search_criteria` в виде словаря:

- `strategy`: `"Cartesian"` (по умолчанию) или `"RandomDiscrete"`;
- `stopping_metric` – метрика, используемая для ранней остановки;
- `stopping_rounds` – обучение останавливается, если используемая метрика не улучшается после заданного числа итераций оценивания (значение 0 отключает эту опцию);
- `stopping_tolerance` – задает требуемую величину улучшения по метрике качества; остановка происходит при отсутствии улучшения хотя бы на эту величину.

Более подробно подбор значений гиперпараметров рассматривается в сообщении http://bit.ly/2u6NPJN.

Полный перебор

В данном примере проверяются два варианта конфигурации скрытых слоев и две скорости обучения, то есть всего будут построены четыре модели.

Пример на языке Python:

```
# Импортирование и преобразование данных
train = h2o.import_file("bigdata/laptop/mnist/train.csv.gz")

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

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

# Набор проверяемых значений гиперпараметров
hidden_opt = [[200, 200], [1024, 1024]]
learn_rate_opt = [1e-6, 1e-5]
hyper_parameters = {"hidden": hidden_opt, "learning_rate": learn_rate_opt}

# Создание модели и поиск по сетке
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)

# Оценка качества модели
print(model_grid)
```

Случайный поиск

Пространство возможных значений гиперпараметров может быть слишком большим для полного перебора, поэтому за заданный промежуток времени можно случайным образом проверить лишь некоторые из комбинаций. В следующем примере настраиваемыми гиперпараметрами являются конфигурация скрытых слоев, скорость обучения и дропаут для входного слоя. Время поиска ограничено пятью минутами.

Пример на языке Python:

```
# Набор проверяемых значений гиперпараметров
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}

# Создание модели и поиск по сетке
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)

# Оценка качества модели
print(model_grid)
```

Контрольные точки

Контрольные точки полезно использовать для сохранения моделей (т. е. их состояния в процессе обучения), если обучение занимает много времени, а также для того, чтобы в дальнейшем продолжать обучение (иногда с другими параметрами). В примере ниже модель обучается в течение 20 эпох и сохраняется при помощи метода `h2o.save_model`. Затем она загружается при помощи метода `h2o.load_model`, после чего обучение продолжается.

Пример на языке Python:

```
# Импортирование и преобразование данных
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()

# Создание модели
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)

# Оценка качества модели
model.show()
print(model.scoring_history())

# Сохранение модели
model_path = h2o.save_model(model=model, force=True)

# Загрузка модели
model_ckpt = h2o.load_model(model_path)

# Продолжение обучения с контрольной точки
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)

# Оценка качества модели
model_warm.show()
print(model_warm.scoring_history())
```

Ансамбли

Модели Deep Water можно объединять в ансамбли с другими моделями, созданными в библиотеке H2O. В данном примере используются три базовые модели: GBM, GLM и Deep Water, объединенные при помощи стекинга. Более подробная информация о стекинге моделей доступна по ссылке http://bit.ly/2ht6M3v.

Пример на языке 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

# Импортирование данных
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
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
glm_model = H2OGeneralizedLinearEstimator(
    family = "binomial",
    lambda_min_ratio = 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()

# Модель Deep Water
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()

# Ансамбль
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()
```

Признаки скрытых слоев и меры сходства

Скрытые слои обученной модели могут дать нам полезное представление входных данных в другом признаковом пространстве (альтернативное признаковое описание). У моделей Deep Water есть метод `deepfeatures`, позволяющий извлекать эти признаки, которые затем можно использовать различными способами. В следующем примере происходит извлечение признаков из скрытого слоя предварительно обученной сверточной нейронной сети. Полученные признаки используются для обучения модели класса GLM.

Пример на языке Python:

```
# Загрузка нейросети
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)

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

# Объединение признаков с целевой переменной и разделение таблицы
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])

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

# Оценка качества модели
glm_model.show()
```

Другое применение таких признаков заключается в их использовании при решении задач обучения на неразмеченных данных, таких как кластеризация. Они выступают в качестве векторного преставления данных, что позволяет вычислять меры сходства (l1- и l2-норму, косинусную меру и квадрат косинусной меры).

Пример на языке Python:

```
# Разделение на таблицы references и queries
references = extracted_features[5:, :]
queries = extracted_features[:3, :]

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

# Проверка размерностей
print("references: " + str(references.shape))
print("queries: " + str(queries.shape))
print("similarity: " + str(similarity.shape))

# Вывод результатов
print(similarity.head())
```

Поддержка нескольких GPU

Поддержка вычислений на нескольких GPU зависит от используемого бекенда.

Развертывание моделей

MOJO

Модели могут быть сохранены в бинарном формате MOJO (см. http://bit.ly/2hprGjY), после чего их можно запускать в исполняющей среде Java независимо от H2O. Все, что требуется, – это наличие файлов *h2o-genmodel.jar* и *deepwater-all.jar*.

Модель в формате MOJO может быть получена в веб-интерфейсе Flow или при помощи команды вида

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

Prediction Service Builder

Prediction Service Builder является отдельным веб-приложением, помогающим получать модели в формате MOJO и файлы Web Archive (*war*) для создания сервисов на основе прогностических моделей. Инструкции по сборке Prediction Service Builder можно найти на странице http://bit.ly/2u9OU3B.

Перед тем как приступить к созданию *war*-файла, убедитесь в наличии у вас файлов *h2o-genmodel.jar* и *deepwater-all.jar*. Их можно получить, выполнив команды:

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

war-файлы можно создавать как с использованием веб-интерфейса Prediction Service Builder, так и из командной строки. Пример:

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

Файл `example.war` может быть запущен при помощи Jetty Runner:

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

Доступ к сервису на основе модели можно получить по адресу http://localhost:55001.