Национальный исследовательский

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Образовательный курс
«Введение в глубокое обучение с использованием
Intel® neon™ Framework»

Введение в Intel® neon™ Framework

При поддержке компании Intel

Кустикова Валентина,
к.т.н., ст.преп. каф. МОСТ ИИТММ,
ННГУ им. Н.И. Лобачевского

Содержание

- Общая информация об инструменте Intel® neon™ Framework
- Установка Intel® neon™ Framework
- Общая процедура разработки глубокой модели с использованием Intel® neon™ Framework
  - Генерация бэкенда
  - Загрузка данных
  - Определение архитектуры глубокой модели
  - Обучение глубокой модели
  - Оценка качества работы глубокой модели
- Использование Intel® neon™ Framework на примере многослойной полносвязной сети для решения задачи предсказания пола человека по фотографии

ОБЩАЯ ИНФОРМАЦИЯ ОБ INTEL® NEON™ FRAMEWORK

Общая информация об инструменте Intel® neon™ Framework (1)

- Intel® neon™ Framework – базовый инструмент глубокого обучения, разрабатываемый компанией Intel
- Цель разработки – простота использования библиотеки глубокого обучения и расширяемость
- Обеспечивает максимальную производительность на всех аппаратных средствах (превосходит Caffe, Theano, Torch и TensorFlow)
- Используется командой Intel Nervana для решения внутренних задач из разных предметных областей

Общая информация об инструменте Intel® neon™ Framework (2)

- Лицензия: Apache 2.0 License

- Страница для скачивания исходных кодов [https://github.com/NervanaSystems/neon]
- Документация (API + tutorials) [http://neon.nervanasys.com/docs/latest]
- Страница Intel AI Academy [https://software.intel.com/ru-ru/ai-academy/frameworks/neon]

- Коллекция обученных моделей [https://github.com/NervanaSystems/ModelZoo]

Общая информация об инструменте Intel® neon™ Framework (3)

- Поддерживаемые ОС: Linux, Mac OS X
- Язык программирования: Python
- Исполнение на следующих платформах:
  - CPU (+ Intel® Math Kernel Library)
  - GPU (Pascal, Maxwell, Kepler)
  - Nervana hardware
- Поддержка нескольких графических процессоров

Открытая вычислительная платформа Intel® AI DevCloud

- Открытая вычислительная платформа Intel® AI DevCloud [https://software.intel.com/en-us/ai-academy/tools/devcloud]
- Кластер из процессоров Intel® Xeon® Scalable для решения задач машинного и глубокого обучения
- Доступные инструменты:
  - Intel® Neon™ Framework
  - Intel® Optimization for Theano*
  - Intel® Optimization for TensorFlow*
  - Intel® Optimization for Caffe*
  - Intel® Distribution for Python* (including NumPy, SciPy, and scikit-learn*)
  - Keras* library

УСТАНОВКА INTEL® NEON™ FRAMEWORK ПОД LINUX

Установка Intel® neon™ Framework (1)

- ❑ Python 2.7 или Python 3.4+
- ❑ Обязательные зависимости:
  - – ***python-pip*** – инструмент для установки пакетов Python
  - – ***libhdf5-dev*** – библиотека для загрузки данных в формате hdf5
  - – ***libyaml-dev*** – библиотека для разбора описания глубокой модели и параметров ее обучения в формате YAML
  - – ***pkg-config*** восстанавливает информацию об установленных библиотеках
  - – ***python-virtualenv*** – инструмент для создания изолированных сред Python (создает среду, которая имеет собственные установочные директории и при необходимости не имеет доступа к глобальным библиотекам)

Установка Intel® neon™ Framework (2)

- Опциональные зависимости:
  - OpenCV для активации модуля загрузки данных aneon (ffmpeg для работы с аудио- и видеоданными)
  - Intel® Math Kernel Library (MKL используется по умолчанию для поддержки многопоточности при исполнении на Intel CPU)
  - CUDA SDK и драйвера (при исполнении на GPU)

- ***Примечание:*** использование Intel® Distribution for Python* (включает MKL, NumPy, SciPy, scikit-learn*) вместо Python+MKL позволяет ускорить обучение/тестирование глубоких моделей

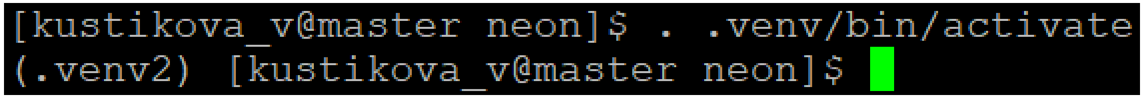
Установка Intel® neon™ Framework (3)

```
# клонирование репозитория
git clone https://github.com/NervanaSystems/neon.git
# переход в рабочую директорию
cd neon
# переход к последнему стабильному релизу
git checkout latest
# сборка инструмента (по умолчанию с Intel® MKL)
make
```

- ***Примечание:*** при установке инициализируется виртуальная среда, в рамках которой осуществляется установка всех необходимых зависимостей

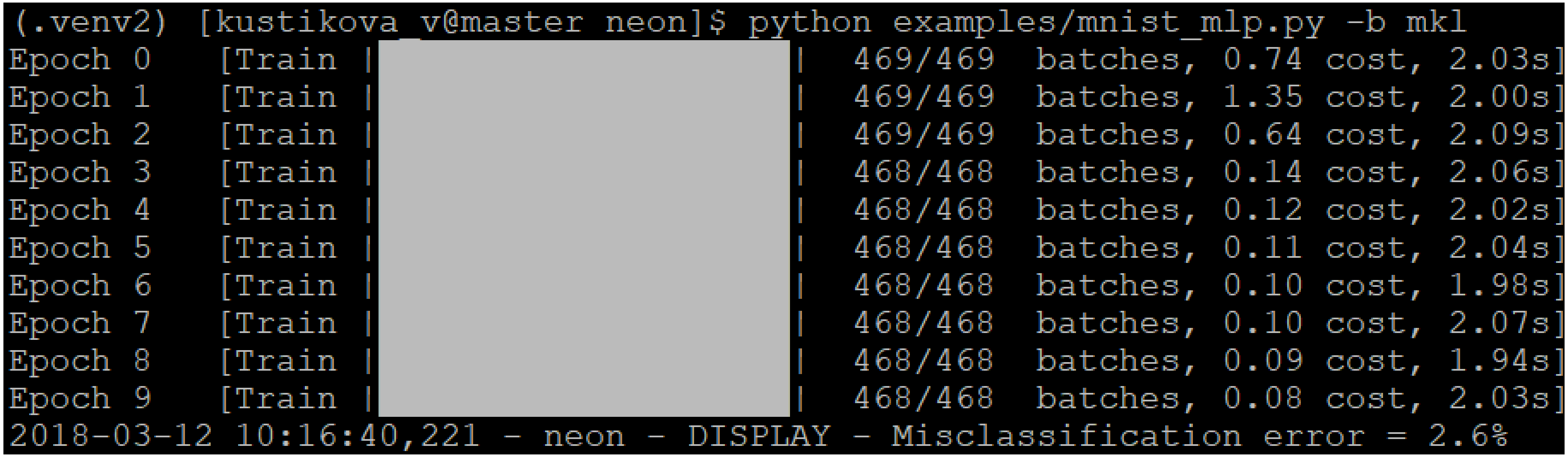
Запуск примера из пакета Intel® neon™ Framework (1)

```
cd neon/
# активация виртуальной среды для последующей работы
. .venv/bin/activate
```



```
[kustikova_v@master neon]$ . .venv/bin/activate
(.venv2) [kustikova_v@master neon]$ 
```

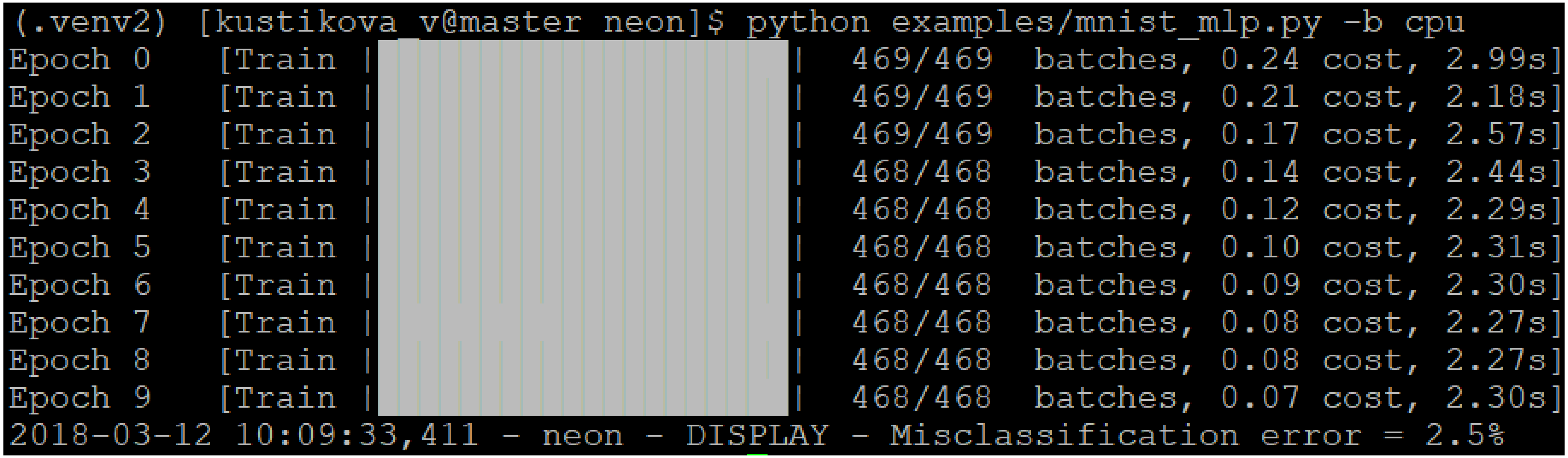
```
# исполнение на CPU с использованием Intel® MKL (нет GPU)
python examples/mnist_mlp.py -b mkl
```



```
(.venv2) [kustikova_v@master neon]$ python examples/mnist_mlp.py -b mkl
Epoch 0   [Train |                    |  469/469  batches, 0.74 cost, 2.03s]
Epoch 1   [Train |                    |  469/469  batches, 1.35 cost, 2.00s]
Epoch 2   [Train |                    |  469/469  batches, 0.64 cost, 2.09s]
Epoch 3   [Train |                    |  468/468  batches, 0.14 cost, 2.06s]
Epoch 4   [Train |                    |  468/468  batches, 0.12 cost, 2.02s]
Epoch 5   [Train |                    |  468/468  batches, 0.11 cost, 2.04s]
Epoch 6   [Train |                    |  468/468  batches, 0.10 cost, 1.98s]
Epoch 7   [Train |                    |  468/468  batches, 0.10 cost, 2.07s]
Epoch 8   [Train |                    |  468/468  batches, 0.09 cost, 1.94s]
Epoch 9   [Train |                    |  468/468  batches, 0.08 cost, 2.03s]
2018-03-12 10:16:40,221 - neon - DISPLAY - Misclassification error = 2.6%
```

Запуск примера из пакета Intel® neon™ Framework (2)

```
# исполнение на CPU (неоптимизированная версия)
python examples/mnist_mlp.py -b cpu
```



```
(.venv2) [kustikova_v@master neon]$ python examples/mnist_mlp.py -b cpu
Epoch 0   [Train |                    |  469/469  batches, 0.24 cost, 2.99s]
Epoch 1   [Train |                    |  469/469  batches, 0.21 cost, 2.18s]
Epoch 2   [Train |                    |  469/469  batches, 0.17 cost, 2.57s]
Epoch 3   [Train |                    |  468/468  batches, 0.14 cost, 2.44s]
Epoch 4   [Train |                    |  468/468  batches, 0.12 cost, 2.29s]
Epoch 5   [Train |                    |  468/468  batches, 0.10 cost, 2.31s]
Epoch 6   [Train |                    |  468/468  batches, 0.09 cost, 2.30s]
Epoch 7   [Train |                    |  468/468  batches, 0.08 cost, 2.27s]
Epoch 8   [Train |                    |  468/468  batches, 0.08 cost, 2.27s]
Epoch 9   [Train |                    |  468/468  batches, 0.07 cost, 2.30s]
2018-03-12 10:09:33,411 - neon - DISPLAY - Misclassification error = 2.5%
```

```
# исполнение на GPU (наличие GPU при сборке)
python examples/mnist_mlp.py -b gpu
```

Запуск примера из пакета Intel® neon™ Framework (3)

```
# запуск с использованием YAML-файла с описанием
# архитектуры модели и параметров обучения
neon examples/mnist_mlp.yaml

# деактивация виртуальной среды исполнения
deactivate
```

РАЗРАБОТКА ГЛУБОКОЙ МОДЕЛИ С ИСПОЛЬЗОВАНИЕМ INTEL® NEON™ FRAMEWORK

Общая процедура разработки глубокой модели с использованием Intel® neon™ Framework (1)

Разработка скрипта, содержащего следующие блоки:

1. ***Генерация бэкенда***
   – Бэкенд определяет, где будут выполняться вычисления: CPU, CPU+MKL, GPU (Pascal, Maxwell или Kepler)
2. ***Загрузка данных***
   – Загрузка широко известных наборов данных (MNIST, CIFAR10, PASCAL VOC, UCF101 и другие)
   – Реализация собственных загрузчиков данных
3. ***Определение архитектуры глубокой модели***
   – Слои
   – Функции активации
   – Инициализаторы весов

Общая процедура разработки глубокой модели с использованием Intel® neon™ Framework (2)

4. ***Определение параметров обучения модели***
   - Целевая функция
   - Метрика оценки качества
   - Метод оптимизации
   - Правило изменения параметров обучения
5. ***Обучение модели и оценка качества ее работы***
   - Модель
   - Метрика качества

- ❑ ***Примечание:*** этапы 2-5 представляют собой классическую схему решения задачи с использованием методов глубокого обучения

Общая процедура разработки глубокой модели с использованием Intel® neon™ Framework

<u>Разработка скрипта, содержащего следующие блоки:</u>

1. ***Генерация бэкенда***
   - Бэкенд определяет, где будут выполняться вычисления: CPU, CPU+MKL, GPU (Pascal, Maxwell или Kepler)
2. ***Загрузка данных***
3. ***Определение архитектуры глубокой модели***
4. ***Определение параметров обучения модели***
5. ***Обучение модели и оценка качества ее работы***

Генерация бэкенда (1)

- ❑ Автоматическая генерация бэкенда – использование встроенного класса **`neon.util.argparser.NeonArgparser`** для разбора аргументов командной строки
- ❑ Основные предусмотренные опции:
  - **`[-b {cpu, mkl, gpu}]`** – бэкенд
  - **`[-e EPOCHS]`** – количество эпох
  - **`[-z BATCH_SIZE]`** – размер мини-батча

```
from neon.util.argparser import NeonArgparser

# parse the command line arguments
parser = NeonArgparser(__doc__)
args = parser.parse_args()
```

Генерация бэкенда (2)

- ❑ Прямой доступ к бэкенд имеет значение при создании пользовательских слоев и целевых функций
- ❑ Рассмотрение находится за рамками настоящей лекции
- ❑ Ссылка на документацию [http://neon.nervanasys.com/docs/latest/backends.html]

Общая процедура разработки глубокой модели с использованием Intel® neon™ Framework

<u>Разработка скрипта, содержащего следующие блоки:</u>

1. ***Генерация бэкенда***
2. ***Загрузка данных***
   - Загрузка широко известных наборов данных (MNIST, CIFAR10, PASCAL VOC, UCF101 и другие)
   - Реализация собственных загрузчиков данных
3. ***Определение архитектуры глубокой модели***
4. ***Определение параметров обучения модели***
5. ***Обучение модели и оценка качества ее работы***

Загрузка широко известных наборов данных

- ❑ **`neon.data.datasets.Dataset`** – базовый класс для загрузки наборов данных
- ❑ Intel® neon™ Framework содержит абстракции представления наборов данных для обработки этих данных
- ❑ Пример загрузки набора данных MNIST:

```
from neon.data import MNIST

mnist = MNIST(path='path/to/save/downloadeddata/')
train_set = mnist.train_iter
valid_set = mnist.valid_iter
```

Загрузка собственных данных (1)

- Компоненты для работы с данными в Intel® neon™ Framework:
  - **`neon.data.dataiterator.NervanaDataIterator`** – итератор данных, который в процессе обучения и тестирования предоставляет модели мини-батчи данных
  - **`neon.data.datasets.Dataset`** – абстракция для загрузки и предварительной обработки данных набора данных. При работе с собственными данными компонент является опциональным

Загрузка собственных данных (2)

- Итераторы данных – итераторы Python, которые реализуют метод **`__iter__`**, возвращающий очередной мини-батч данных при каждом вызове
- Данные небольшого объема:
  - **`neon.data.dataiterator.ArrayIterator`** – итератор для данных в форме numpy-массивов

```
from neon.data import ArrayIterator
import numpy as np

X = np.random.rand(10000,3072) # X.shape = (10000, 3072)
y = np.random.randint(0,10,10000) # y.shape = (10000, )
train = ArrayIterator(X=X, y=y, nclass=10,
                      lshape=(3,32,32))
```

Загрузка собственных данных (3)

- Итераторы данных – итераторы Python, которые реализуют метод **`__iter__`**, возвращающий очередной мини-батч данных при каждом вызове
- Данные небольшого объема:
  - **`neon.data.dataiterator.ArrayIterator`** – итератор для данных в форме numpy-массивов
  - Специализированные итераторы (**`neon.data.Text`** и **`neon.data.ImageCaption`** для работы с текстовыми базами и базами изображений с описанием)
- Данные большого объема:
  - **`neon.data.hdf5iterator.HDF5Iterator`** – итератор для работы с данными в формате hdf5
  - Aeon – внешний модуль загрузки данных

Общая процедура разработки глубокой модели с использованием Intel® neon™ Framework

<u>Разработка скрипта, содержащего следующие блоки:</u>

1. ***Генерация бэкенда***
2. ***Загрузка данных***
3. ***Определение архитектуры глубокой модели***
   - Слои
   - Функции активации
   - Инициализаторы весов
4. ***Определение параметров обучения модели***
5. ***Обучение модели и оценка качества ее работы***

Определение архитектуры глубокой модели (1)

- Архитектура глубокой модели формируется посредством ***добавления слоев в список***
- Каждый слой имеет несколько ключевых методов:

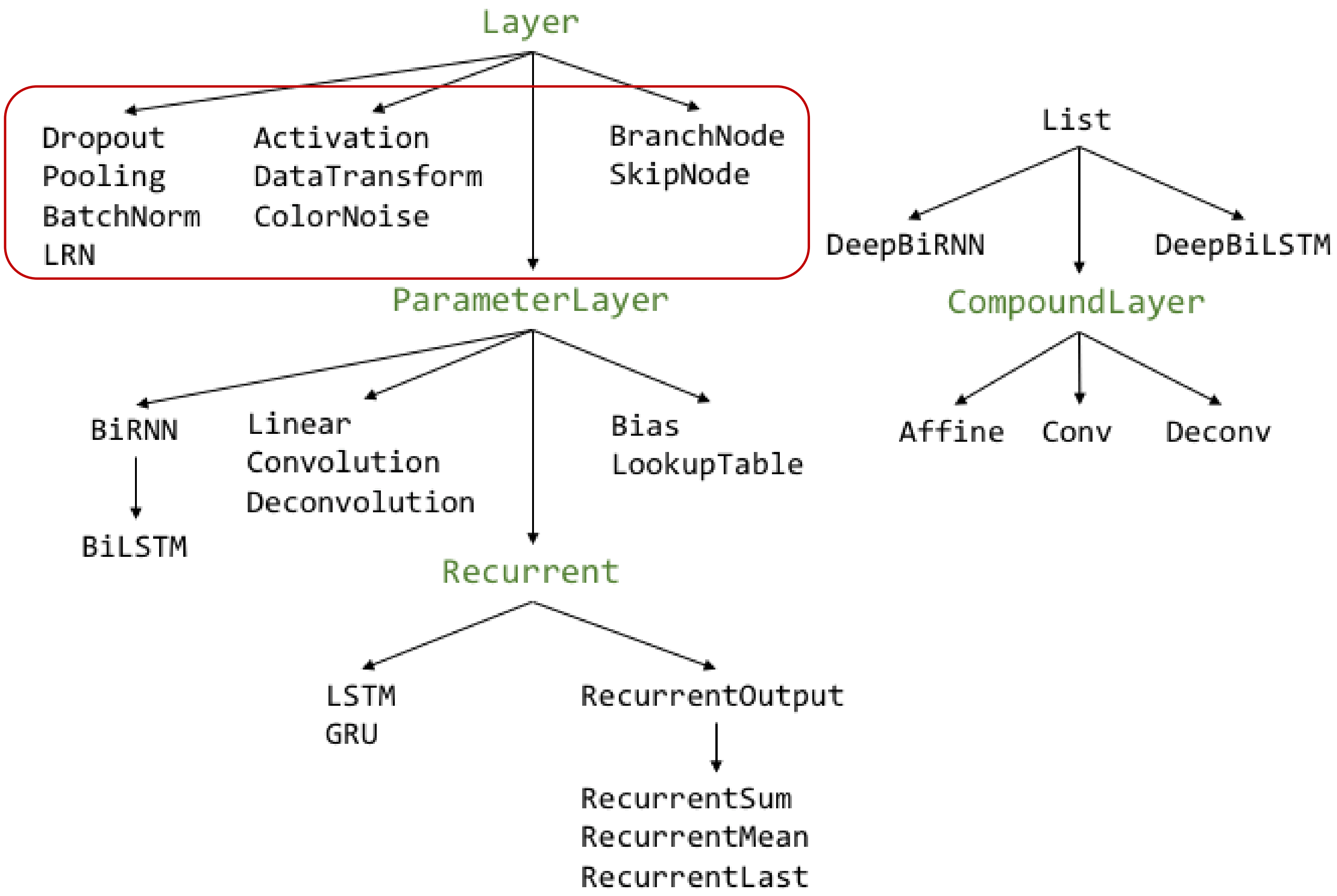
Метод	Назначение
`configure(self, in_obj)`	Определение размеров входа/выхода (**`out_shape`**/**`in_shape`**) слоев
`allocate(self, shared_outputs=None)`	Выделение памяти для выходного буфера
`fprop(self, inputs, inference=False)`	Прямой проход
`bprop(self, error)`	Обратный проход

Определение архитектуры глубокой модели (2)

- Вызов методов в процессе обучения:
  - Прямое распространение обучающих данных через слои модели и вызов метода `configure` для установки размера выходных карт признаков
  - Вызов метода `allocate` на каждом слое для выделения памяти под буферы карт признаков

Иерархия слоев глубоких моделей (1)

Слои, не имеющие весовых параметров (не требуют начальной инициализации)



Layer

Dropout
Pooling
BatchNorm
LRN

Activation
DataTransform
ColorNoise

BranchNode
SkipNode

ParameterLayer

BiRNN

BiLSTM

Linear
Convolution
Deconvolution

Bias
LookupTable

Recurrent

LSTM
GRU

RecurrentOutput

RecurrentSum
RecurrentMean
RecurrentLast

List

DeepBiRNN

DeepBiLSTM

CompoundLayer

Affine

Conv

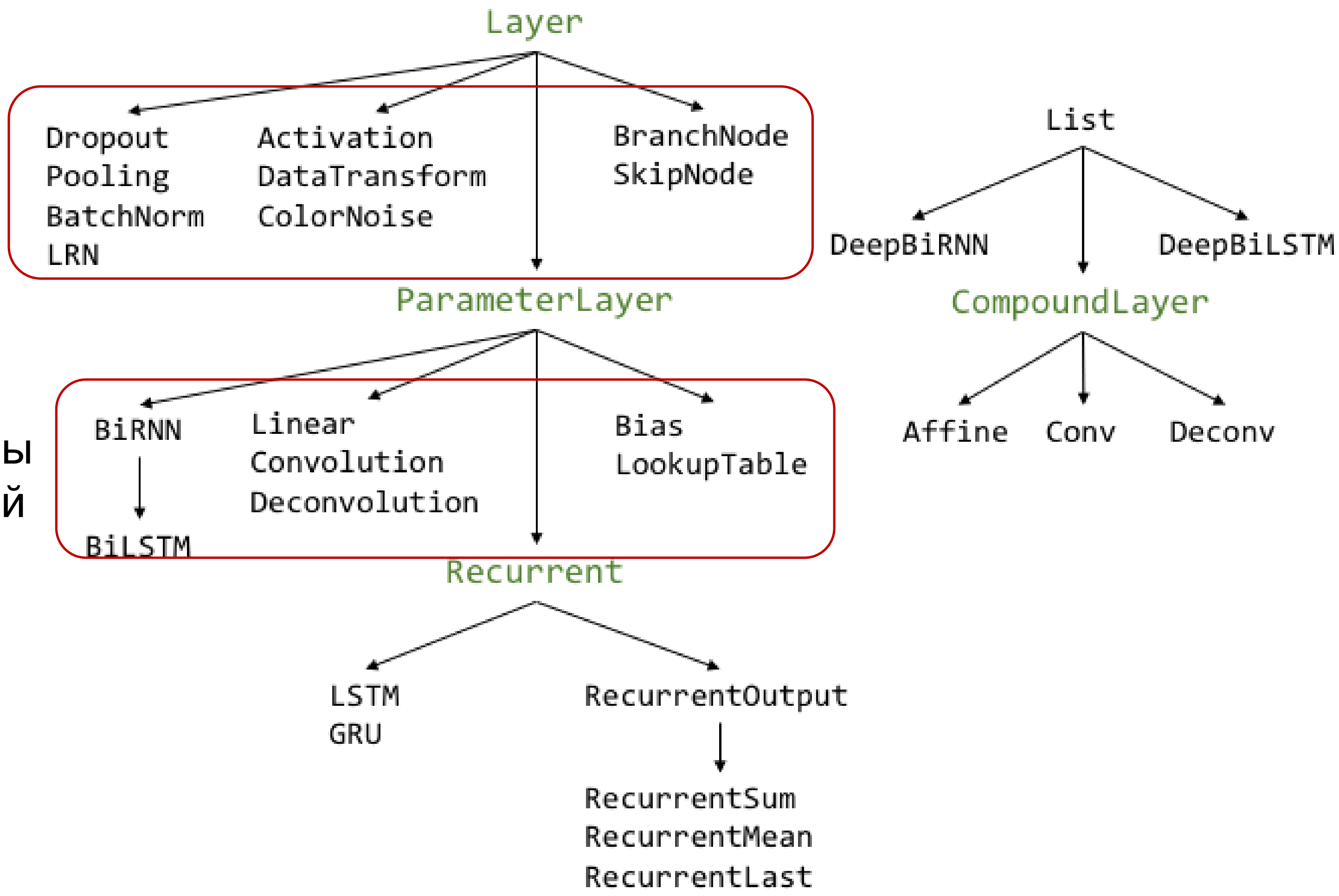
Deconv

* Документация Intel® neon™ Framework [http://neon.nervanasys.com/docs/latest/layers.html].

Иерархия слоев глубоких моделей (2)

Слои, не имеющие весовых параметров (не требуют начальной инициализации)

Слои, имеющие весовые параметры (требуют начальной инициализации)

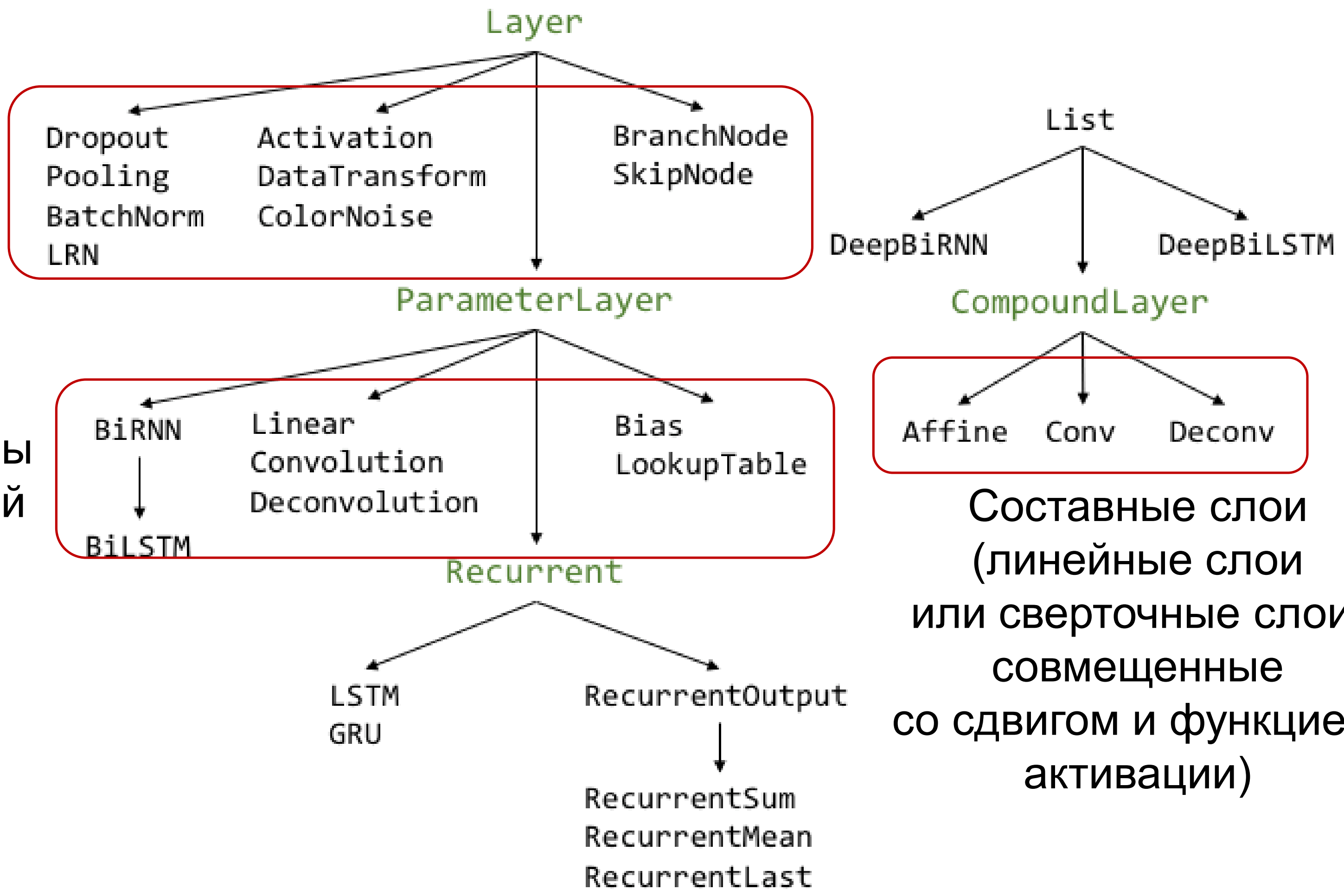


* Документация Intel® neon™ Framework [http://neon.nervanasys.com/docs/latest/layers.html].

Иерархия слоев глубоких моделей (3)

Слои, не имеющие весовых параметров (не требуют начальной инициализации)

Слои, имеющие весовые параметры (требуют начальной инициализации)



Layer

Dropout
Pooling
BatchNorm
LRN

Activation
DataTransform
ColorNoise

BranchNode
SkipNode

ParameterLayer

BiRNN
BiLSTM

Linear
Convolution
Deconvolution

Bias
LookupTable

Recurrent

LSTM
GRU

RecurrentOutput

RecurrentSum
RecurrentMean
RecurrentLast

List

DeepBiRNN

DeepBiLSTM

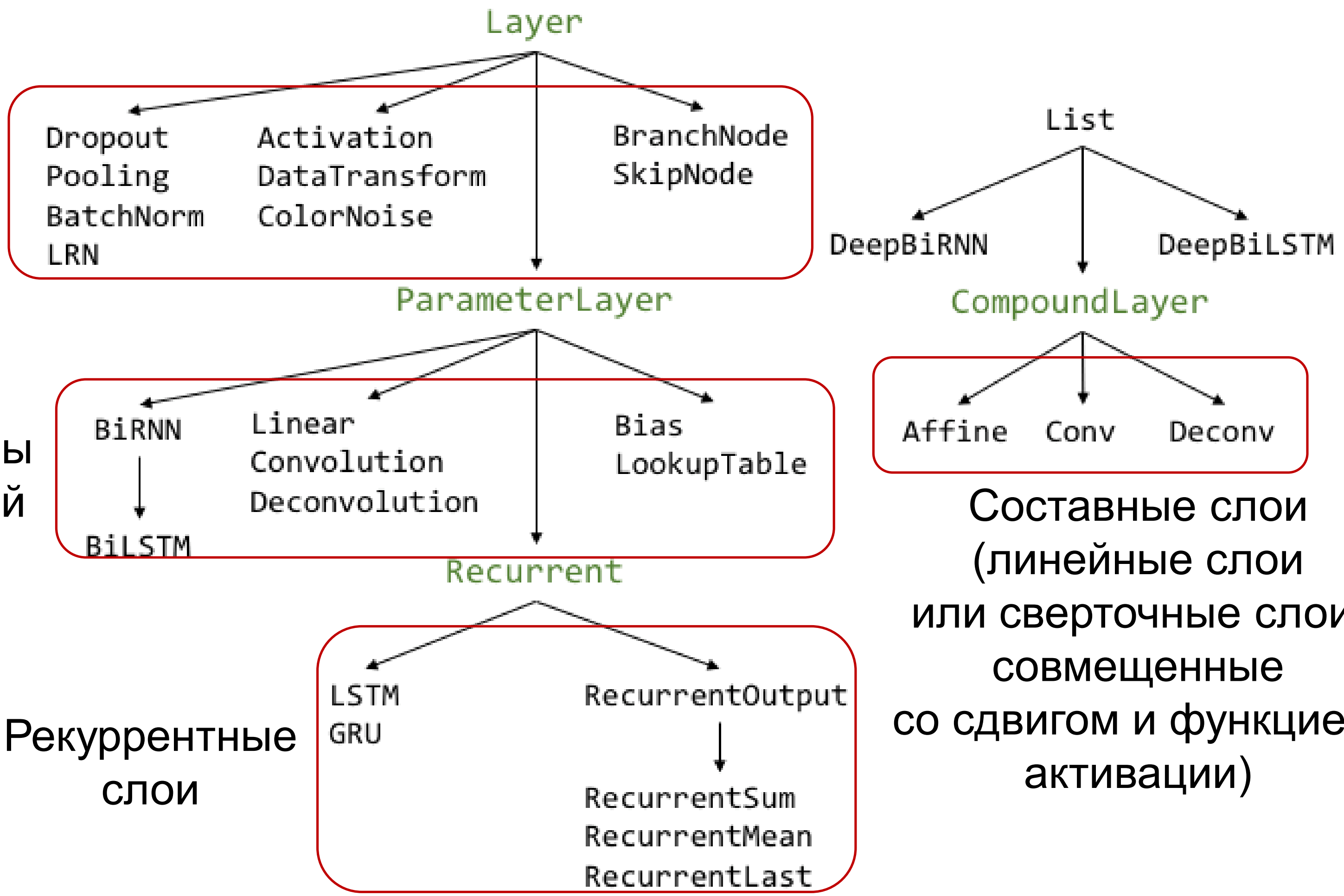
CompoundLayer

Affine
Conv
Deconv

Составные слои (линейные слои или сверточные слои, совмещенные со сдвигом и функцией активации)

* Документация Intel® neon™ Framework [http://neon.nervanasys.com/docs/latest/layers.html].

Иерархия слоев глубоких моделей (4)



Layer

Dropout
Pooling
BatchNorm
LRN

Activation
DataTransform
ColorNoise

BranchNode
SkipNode

Слои, не имеющие весовых параметров (не требуют начальной инициализации)

ParameterLayer

BiRNN
BiLSTM

Linear
Convolution
Deconvolution

Bias
LookupTable

Слои, имеющие весовые параметры (требуют начальной инициализации)

Recurrent

LSTM
GRU

RecurrentOutput
RecurrentSum
RecurrentMean
RecurrentLast

Рекуррентные слои

List

DeepBiRNN

DeepBiLSTM

CompoundLayer

Affine
Conv
Deconv

Составные слои (линейные слои или сверточные слои, совмещенные со сдвигом и функцией активации)

* Документация Intel® neon™ Framework [http://neon.nervanasys.com/docs/latest/layers.html].

Примеры типовых слоев (1)

`neon.layers.Linear`	**`nout`**	Полносвязный слой с числом выходов **`nout`**
`neon.layers.Convolution`	**`fshape,`** **`strides,`** **`padding`**	Сверточный слой с параметрами фильтров **`fshape=(height, width, num_filters)`**
`neon.layers.Pooling`	**`fshape,`** **`op,`** **`strides,`** **`padding`**	Пространственное объединение с параметрами фильтров **`fshape=(height, width, num_filters)`** и операцией **`op`** (“**`avg`**”/“**`sum`**”)
`neon.layers.Activation`	**`transform`**	Применение преобразования **`transform`** ко входу

Примеры типовых слоев (2)

`neon.layers.layer.Affine`	`nout, init, bias, activation, name`	Линейный слой с числом выходов `nout` со сдвигом `bias` и активацией `activation`
`neon.layers.Conv`	`fshape, init, strides, padding, dilation, bias, batch_norm, activation, name`	Сверточный слой со сдвигом `bias` и активацией `activation`

Примеры функций активации

`neon.transforms.Identity`	$f(x) = x$
`neon.transforms.Rectlin`	$f(x) = \max\{x, 0\}$
`neon.transforms.Softmax`	$f(x_j) = \frac{e^{x_j}}{\sum e^{x_i}}$
`neon.transforms.Tanh`	$f(x) = th(x)$
`neon.transforms.Logistic`	$f(x) = \frac{1}{1 + e^{-x}}$

Примеры инициализаторов слоев

`neon.initializers.Constant`	Инициализация всех элементов тензора постоянным значением **`val`**
`neon.initializers.Uniform`	Инициализация всех элементов тензора значениями из равномерного распределения на отрезке от **`low`** до **`high`**
`neon.initializers.Gaussian`	Инициализация всех элементов тензора значениями из нормального распределения с μ=**`loc`** и σ=**`scale`**

Пример формирования архитектуры модели (1)

```
from neon.initializers import Gaussian
from neon.layers import Conv, Pooling, Affine
from neon.transforms import Rectlin, Softmax
...
layers = [Conv((11, 11, 64), init=Gaussian(scale=0.01),
               activation=Rectlin(), padding=3,
               strides=4),
          Pooling(3, strides=2),
          Conv((5, 5, 192), init=Gaussian(scale=0.01),
               activation=Rectlin(), padding=2),
          Pooling(3, strides=2),
          Conv((3, 3, 384), init=Gaussian(scale=0.03),
               activation=Rectlin(), padding=1),
```

Пример формирования архитектуры модели (2)

```
Conv((3, 3, 256), init=Gaussian(scale=0.03),
     activation=Rectlin(), padding=1),
Conv((3, 3, 256), init=Gaussian(scale=0.03),
     activation=Rectlin(), padding=1),
Pooling(3, strides=2),
Affine(nout=4096, init=Gaussian(scale=0.01),
       activation=Rectlin()),
Affine(nout=4096, init=Gaussian(scale=0.01),
       activation=Rectlin()),
Affine(nout=1000, init=Gaussian(scale=0.01),
       activation=Softmax())]
```

Общая процедура разработки глубокой модели с использованием Intel® neon™ Framework

<u>Разработка скрипта, содержащего следующие блоки:</u>

1. ***Генерация бэкенда***
2. ***Загрузка данных***
3. ***Определение архитектуры глубокой модели***
4. ***Определение параметров обучения модели***
   - Целевая функция (cost function) – минимизируемый функционал в процессе обучения глубокой модели
   - Метрика оценки качества (metric) – показатель качества решения задачи
   - Метод оптимизации (optimization algorithm)
   - Расписание изменения параметров обучения (learning schedule)
5. ***Обучение модели и оценка качества ее работы***

Целевые функции (1)

`neon.transforms.CrossEntropyBinary`	Бинарная кросс-энтропия $-t\log y-(1-t)\log(1-y)$
`neon.transforms.CrossEntropyMulti`	Мультиномиальная кросс-энтропия $-\sum t_i\log y_i$
`neon.transforms.SumSquared`	Квадратичная функция $\sum(y_i-t_i)^2$
`neon.transforms.MeanSquared`	Нормированная квадратичная функция $\frac{1}{N}\sum(y_i-t_i)^2$

Целевые функции (2)

- Реализация собственных целевых функций:
  - Разработка наследника класса **`neon.transforms.Cost`**
  - Реализация методов **`__call__()`** и **`bprop()`** для вычисления значения целевой функции и ее производной соответственно

Метрики оценки качества модели (1)

`neon.transforms.Misclassification`	Ошибка классификации: $$\frac{fp + fn}{tp + tn + fp + fn}$$
`neon.transforms.Accuracy`	Точность классификации: $$\frac{tp + tn}{tp + tn + fp + fn}$$
`neon.transforms.PrecisionRecall`	Средняя точность: $$precision = \frac{tp}{tp + fp}$$ $$recall = \frac{tp}{tp + fn}$$

- $fp = false\ positives,\ \ tp = true\ positives$
- $fn = false\ negatives,\ \ fp = false\ positives$

Метрики оценки качества модели (2)

- Реализация собственных метрик оценки качества моделей:
  - Разработка наследника класса **`neon.transforms.cost.Metric`**
  - Реализация метода **`__call__()`** для сравнения разметки и выхода глубокой модели

Методы оптимизации

`neon.optimizers.GradientDescentMomentum`	Стохастический градиентный спуск с указанием момента (Stochastic Gradient Descent)
`neon.optimizers.RMSProp`	Распространение среднеквадратического отклонения (Root Mean Square propagation)
`neon.optimizers.Adagrad`	Метод Adagrad (Adaptive Gradient)
`neon.optimizers.Adadelta`	Метод Adadelta
`neon.optimizers.Adam`	Метод Adam (Adaptive Moment Estimation)
`neon.optimizers.MultiOptimizer`	Класс для назначения оптимизатора различным слоям

* Методы оптимизации нейронных сетей [https://habrahabr.ru/post/318970].

Правило изменения параметров обучения

- ❑ Правило изменения параметров обучения в конце эпохи

`neon.optimizers.Schedule`	Постоянная скорость обучения (learning rate), либо изменение с константным шагом
`neon.optimizers.ExpSchedule`	Экспоненциальное изменение скорости обучения $\alpha(t) = \frac{\alpha_0}{1+\beta t}$, α_0 – начальное значение скорости, t – номер эпохи
`neon.optimizers.PolySchedule`	Полиномиальное изменение скорости обучения $\alpha(t) = \alpha_0 \times \left(1 - \frac{t}{T}\right)^{\beta}$, T – количество эпох

Пример задания метода обучения модели

```
from neon.optimizers import GradientDescentMomentum,
                            MultiOptimizer, Schedule


...
# параметр скорости изменяется на итерациях 22, 44 и 65
weight_sched = Schedule([22, 44, 65],
                        (1 / 250.)**(1 / 3.))
opt_gdm = GradientDescentMomentum(0.01, 0.0,
                   wdecay=0.0005, schedule=weight_sched)


opt = MultiOptimizer({'default': opt_gdm})
```

Общая процедура разработки глубокой модели с использованием Intel® neon™ Framework

<u>Разработка скрипта, содержащего следующие блоки:</u>

1. ***Генерация бэкенда***
2. ***Загрузка данных***
3. ***Определение архитектуры глубокой модели***
4. ***Определение параметров обучения модели***
5. ***Обучение модели и оценка качества ее работы***
   - Модель – контейнер для объединения данных, глубокой модели и параметров обучения с целью последующего обучения/тестирования
   - Метрика оценки качества работы модели (используются те же, что и при обучении)

Модель – контейнер для объединения данных, глубокой модели и параметров обучения (1)

- Класс **`neon.models.model.Model`**
- Параметры конструктора:
  - **`layers`** – список слоев глубокой модели
- Некоторые методы:
  - **`fit(…)`** – обучение параметров глубокой модели
  - **`eval(…)`** – оценка качества работы модели по заданной метрике
  - **`get_outputs(…)`** – получение значений активационной функции на последнем слое сети
  - **`save_params(…)`** – сохранение параметров глубокой модели
  - **`load_params(…)`** – загрузка параметров глубокой модели

Модель – контейнер для объединения данных, глубокой модели и параметров обучения (2)

```
fit(dataset, cost, optimizer,
    num_epochs, callbacks)
```

- ☐ **`dataset`** – тренировочный набор данных
- ☐ **`cost`** – целевая функция
- ☐ **`optimizer`** – параметры обучения глубокой модели
- ☐ **`num_epochs`** – количество эпох
- ☐ **`callbacks`** – обратные вызовы, выполняемые по завершении обработки мини-батча или по окончании эпохи

```
eval(dataset, metric)
```

- ☐ **`dataset`** – валидационный набор данных
- ☐ **`metric`** – метрика оценки качества работы обученной глубокой модели

Модель – контейнер для объединения данных, глубокой модели и параметров обучения (3)

`get_outputs(dataset)`

- **`dataset`** – набор данных для получения выхода глубокой модели

`save_params(param_path, keep_states=True)`

- **`param_path`** – полное имя файла для сохранения параметров глубокой модели
- **`keep_states`** – флаг для сохранения состояния метода обучения

`load_params(param_path, load_states =True)`

- **`param_path`** – полное имя файла для чтения параметров
- **`load_states`** – флаг для чтения состояния метода обучения

Пример обучения глубокой модели (1)

```
from neon.models import Model
from neon.transforms import Misclassification
...
# создание контейнера модели
mlp = Model(layers=layers)

# создание контейнера обратных вызовов
#   model=mlp – модель
#   eval_set=valid_set – данные для оценки качества
callbacks = Callbacks(mlp, eval_set=valid_set,
                      **args.callback_args)
...
```

Пример обучения глубокой модели (2)

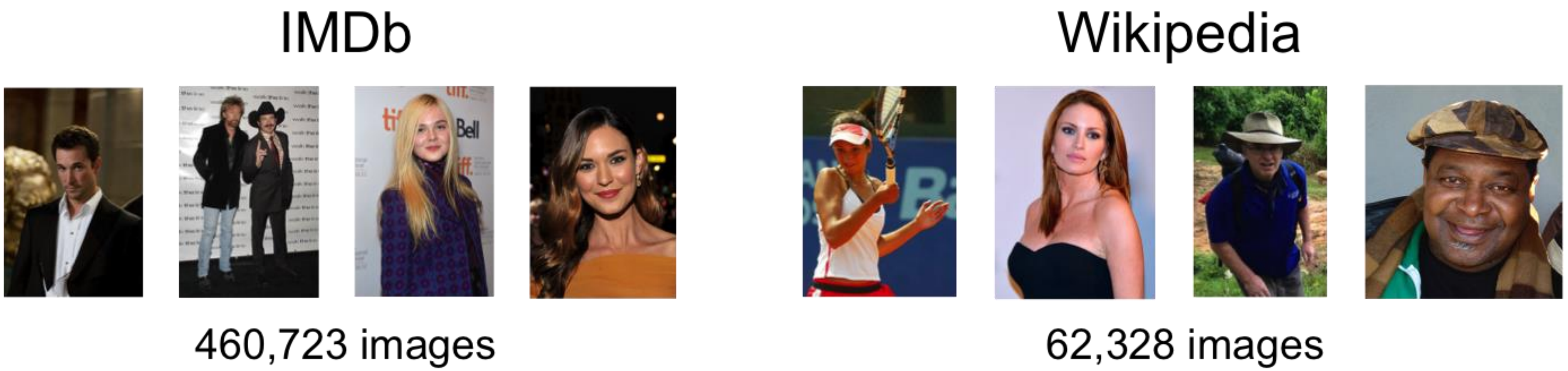
```
# обучение модели mlp
mlp.fit(train_set, optimizer=optimizer,
        num_epochs=args.epochs, cost=cost,
        callbacks=callbacks)

# оценка качества работы модели
error_rate = mlp.eval(valid_set,
                      metric=Misclassification())
```

ПРИМЕР ПОСТРОЕНИЯ СЕТИ ДЛЯ РЕШЕНИЯ ПРАКТИЧЕСКОЙ ЗАДАЧИ

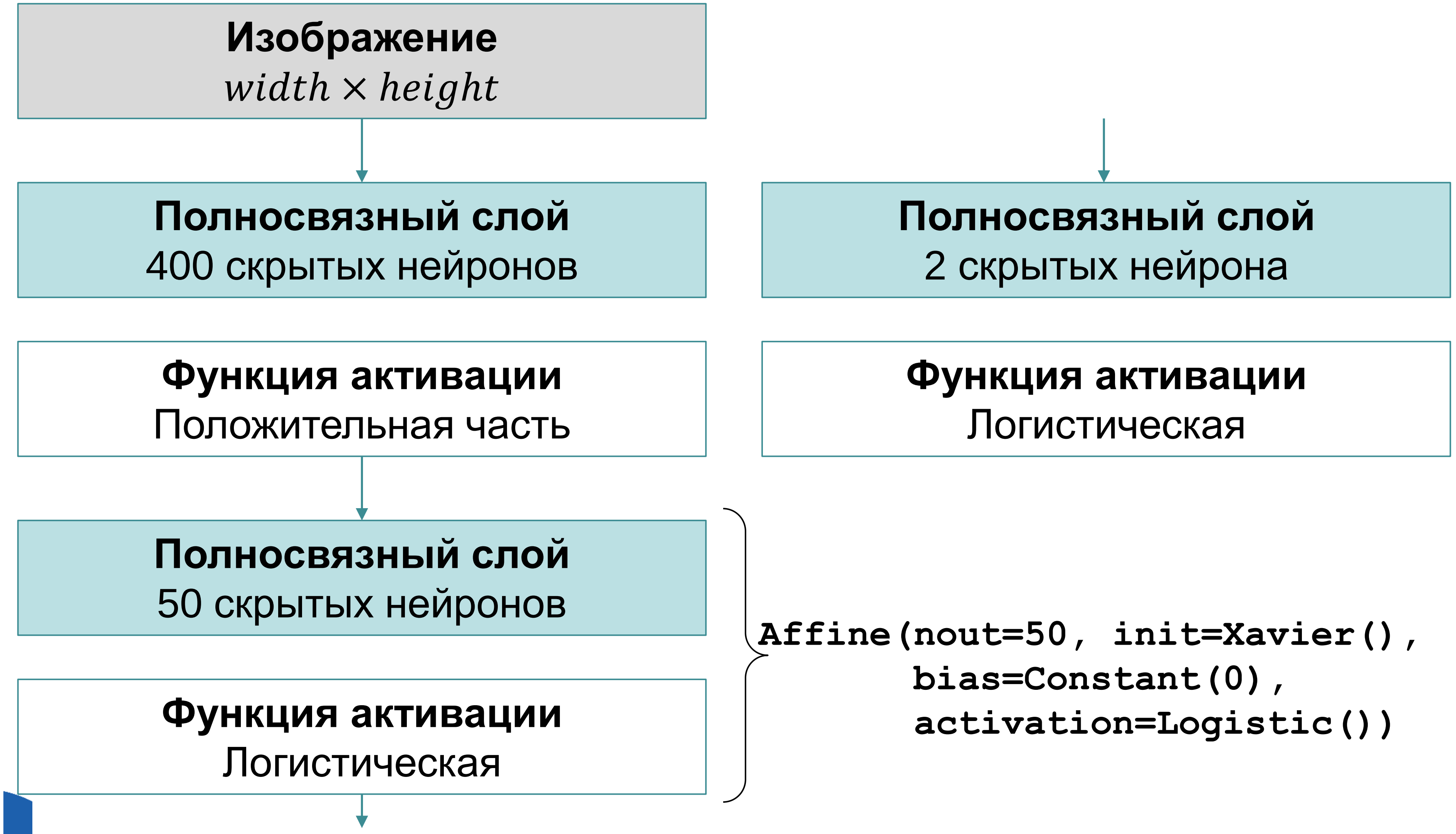
Задача предсказания пола человека по фотографии

- IMDB-WIKI dataset [https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki]



- Разметка:
  - окаймляющий прямоугольник для лица (задача детектирования)
  - возраст изображенного человека (задача классификации)
  - **пол изображенного человека (задача классификации)**
  - …

Архитектура полносвязной сети



Изображение
$width \times height$

Полносвязный слой
400 скрытых нейронов

Функция активации
Положительная часть

Полносвязный слой
50 скрытых нейронов

Функция активации
Логистическая

Полносвязный слой
2 скрытых нейрона

Функция активации
Логистическая

```
Affine(nout=50, init=Xavier(),
       bias=Constant(0),
       activation=Logistic())
```

Использование Intel® neon™ Framework на примере многослойной полносвязной сети (1)

```
from neon.callbacks.callbacks import Callbacks, LossCallback
from neon.models import Model
from neon.optimizers import GradientDescentMomentum
from neon.util.argparser import NeonArgparser
from neon.data import HDF5Iterator, ArrayIterator

import numpy as np
import numpy.linalg


# загрузка собственного класса для работы с данными
from datasets.imdb_wiki_face_dataset import IMDB_WIKI_FACE
...
```

Использование Intel® neon™ Framework на примере многослойной полносвязной сети (2)

```
def generate_mlp_model():
    layers = [
        DataTransform(transform=Normalizer(divisor=128.0)),
        Affine(nout=400, init=Xavier(), bias=Constant(0),
            activation=Rectlin()),
        Affine(nout=50, init=Xavier(), bias=Constant(0),
            activation=Logistic()),
        Affine(nout=2, init=Xavier(), bias=Constant(0),
            activation=Logistic(shortcut=True))
    ]
    model = Model(layers=layers)
    cost = GeneralizedCost(costfunc=CrossEntropyBinary())
    return (model, cost)
...
```

Использование Intel® neon™ Framework на примере многослойной полносвязной сети (3)

```
if (__name__ == '__main__'):
    parser = NeonArgparser(__doc__)
    args = parser.parse_args()

    # IMDB_WIKI_FACE – класс для работы с данными в HDF5
    dataset = IMDB_WIKI_FACE(data_root='data')
    X_train, y_train, X_val, y_val, lshape =
        dataset.load(2000, 4000)
    train_set = ArrayIterator(X=X_train, y=y_train,
        nclass=2, lshape=lshape)
    val_set = ArrayIterator(X=X_val, y=y_val,
        nclass=2, lshape=lshape)
    model, cost = generate_mlp_model()
...
```

Использование Intel® neon™ Framework на примере многослойной полносвязной сети (4)

```
    optimizer = GradientDescentMomentum(
        0.01, momentum_coef=0.9,
        stochastic_round=args.rounding, wdecay=0.0005)

    callbacks = Callbacks(model, eval_set=val_set,
                          **args.callback_args)

    model.fit(train_set, optimizer=optimizer,
        num_epochs=args.epochs, cost=cost,
        callbacks=callbacks)
    accuracy = model.eval(val_set, metric=Accuracy())
    print('Accuracy = %.1f%%' % (accuracy * 100))

...
```

Использование Intel® neon™ Framework на примере многослойной полносвязной сети (5)

```
print('Making inference')
outputs = model.get_outputs(val_set)
with open('inference.txt', 'w') as output_file:
    output_file.write('inference,target\n')
    for i, entry in enumerate(outputs):
        output_file.write('%s,%s\n' % (entry, y_val[i]))
```

Замечания

- Последующие примеры реализации различных глубоких нейросетевых моделей в основном отличаются реализацией функции генерации модели
- Вызов основной функции с разными параметрами позволяет работать с различными моделями и устанавливать необходимые параметры обучения

Тестовая инфраструктура

- CPU: Intel® Xeon® CPU E5-2660 0 @ 2.20GHz
- GPU: Tesla K40s 11Gb
- OS: Ubuntu 16.04.4 LTS
- Инструменты:
  - Intel® neon™ Framework 2.6.0
  - CUDA 8.0
  - Python 3.5.2
  - Intel® Math Kernel Library 2017 (Intel® MKL)

Результаты экспериментов

Название	Структура сети	Параметры обучения	Точность, %	Время обучения, с
FCNN-1	Affine(128,0,Tanh), Affine(2,0,Logistic)	batch_size = 128 epoch_count = 30 backend = gpu GradientDescentMomentum(0.01, momentum_coef=0.9, wdecay=0.0005)	71.2	932
FCNN-2	Affine(128,0,Tanh), Affine(64,0,Tanh), Affine(2,0,Logistic)		73.5	977
FCNN-3	Affine(400,0,Rectlin), Affine(50,0,Logistic), Affine(2,0,Logistic)		**77.7**	1013

Сравнение производительности при запуске на различных бэкендах (1)

Название	**CPU**		**GPU**				**CPU+MKL (16 потоков)**			
	S1, с	**S2, с**	**S1, с**	**SS1**	**S2, с**	**SS2**	**S1, с**	**SS1**	**S2, с**	**SS2**
FCNN-1	3735	27	932	4	14	1.93	2259	1.65	21	1.29
FCNN-2	3803	27	977	3.89	13	2.08	2251	1.69	21	1.29
FCNN-3	8067	36	1013	7.96	15	2.4	4406	1.83	30	1.2

- ☐ **S1** – этап обучения
- ☐ **S2** – этап тестирования
- ☐ **SS1 = S1(CPU) / S1** – ускорение этапа обучения по сравнению с CPU-версией
- ☐ **SS2 = S2(CPU) / S2** – ускорение этапа тестирования по сравнению с CPU-версией

Сравнение производительности при запуске на различных бэкендах (2)

Название	CPU		GPU				CPU+MKL (16 потоков)			
	S1, с	S2, с	S1, с	SS1	S2, с	SS2	S1, с	SS1	S2, с	SS2
FCNN-1	3735	27	932	**4**	14	1.93	2259	**1.65**	21	1.29
FCNN-2	3803	27	977	**3.89**	13	2.08	2251	**1.69**	21	1.29
FCNN-3	8067	36	1013	**7.96**	15	2.4	4406	**1.83**	30	1.2

- Время обучения с использованием CPU+MKL-бэкенда меньше, чем соответствующее время для CPU-бэкенда для данного набора моделей
- Время обучения с использованием GPU-бэкенда меньше, чем соответствующее время для CPU+MKL-бэкенда
- Использование MKL позволяет повысить эффективность вычислений, что представляет интерес в отсутствии GPUs
- ***Примечание:*** далее в курсе используется GPU-бэкенд

Повышение производительности этапа обучения за счет использования Intel® Distribution for Python (1)

- CPU: Intel® Xeon® CPU E5-2660 0 @ 2.20GHz
- GPU: Tesla K40s 11Gb
- OS: Ubuntu 16.04.4 LTS
- Intel® neon™ Framework 2.6.0
- Intel® Distribution for Python 2018.3 (includes Python 3.6 and Intel® Math Kernel Library)
- CUDA 8.0

Повышение производительности этапа обучения за счет использования Intel® Distribution for Python (2)

Название	CPU		GPU				CPU+MKL (16 потоков)			
	S1, с	S2, с	S1, с	SS1	S2, с	SS2	S1, с	SS1	S2, с	SS2
FCNN-1	1743	16	912	**1.91**	14	1.14	1495	**1.17**	15	1.07
FCNN-2	1754	17	978	**1.79**	14	1.21	1543	**1.14**	15	1.13
FCNN-3	3146	19	993	**3.17**	14	1.36	2505	**1.26**	18	1.06

- ❑ **S1** – этап обучения
- ❑ **S2** – этап тестирования
- ❑ **SS1 = S1(CPU) / S1** – ускорение этапа обучения по сравнению с CPU-версией
- ❑ **SS2 = S2(CPU) / S2** – ускорение этапа тестирования по сравнению с CPU-версией

Повышение производительности этапа обучения за счет использования Intel® Distribution for Python (3)

Название	**CPU**			**GPU**			**CPU+MKL (16 потоков)**		
	S1 (Python), с	**S1 (Intel Distribution for Python), с**	**S**	**S1 (Python), с**	**S1 (Intel Distribution for Python), с**	**S**	**S1 (Python), с**	**S1 (Intel Distribution for Python), с**	**S**
FCNN-1	3735	1743	**2.14**	932	912	**1.02**	2259	1495	**1.51**
FCNN-2	3803	1754	**2.17**	977	978	**1**	2251	1543	**1.46**
FCNN-3	8067	3146	**2.56**	1013	993	**1.02**	4406	2505	**1.76**

- **S1 (Python)** – этап обучения с использованием Python 3.5.2
- **S1 (Intel Distribution for Python)** – этап обучения с использованием Intel® Distribution for Python 2018.3 (включает Python 3.6 и Intel® Math Kernel Library)
- **S = S1 (Python) / S1 (Intel Distribution for Python)**

Повышение производительности этапа обучения за счет использования Intel® Distribution for Python (4)

Название	**CPU**			**GPU**			**CPU+MKL (16 потоков)**		
	S1 (Python), c	**S1 (Intel Distribution for Python), c**	**S**	**S1 (Python), c**	**S1 (Intel Distribution for Python), c**	**S**	**S1 (Python), c**	**S1 (Intel Distribution for Python), c**	**S**
FCNN-1	3735	1743	**2.14**	932	912	**1.02**	2259	1495	**1.51**
FCNN-2	3803	1754	**2.17**	977	978	**1**	2251	1543	**1.46**
FCNN-3	8067	3146	**2.56**	1013	993	**1.02**	4406	2505	**1.76**

- Intel® Distribution for Python 2018.3 позволяет повысить производительность этапа обучения примерно в 2 раза для CPU-версии и в 1.5 раза для CPU+MKL-версии

СРАВНЕНИЕ INTEL® NEON™ FRAMEWORK И ДРУГИХ ИНСТРУМЕНТОВ

Сравнение возможностей Intel® neon™ Framework и других инструментов глубокого обучения (1)

№	Название	Интерфейсы	ОС	FCNN	CNN	RNN	AE	RBM
1	TensorFlow	Python, Java, Go	Linux, Windows, Mac OS, Android	+	+	+	+	+
2	Caffe	C++, Python, MATLAB	Linux, Windows, OS X	+	+	+	+	−
3	Keras	Python	Linux, Vagrant	+	+	+	+	+
4	Torch	C, Lua	Linux, iOS, Android	+	+	+	+	+
5	MXNet	C++, Python, R, Scala, Julia, Perl, MATLAB, JavaScript	Linux, Windows, Mac OS	+	+	+	+	+
6	Intel® neon™ Framework	Python	Linux, Mac OS	+	+	+	+	−
7	Theano	Python	Linux, Windows, Mac OS	+	+	+	+	+
8	Deepnet	Python	Linux	+	+	−	+	+
9	Deepmat	MATLAB	Linux, Windows, Mac OS, Solaris	+	+	−	+	+
10	Darch	R	Linux, Windows	+	−	−	+	+

Сравнение возможностей Intel® neon™ Framework и других инструментов глубокого обучения (2)

- ☐ Простота развертывания в отличие от большинства библиотек
- ☐ Комплект доступных в Intel® neon™ Framework глубоких моделей аналогичен широко известным инструментам (Caffe, TensorFlow, Torch, MXNet и другим)
- ☐ Удобный объектно-ориентированный интерфейс языка Python
  - Структура аналогична интерфейсам Caffe, MXNet
  - Процедура обучения не требует непосредственной реализации обратного распространения ошибки, как в библиотеках Torch и TensorFlow
  - Возможность расширения имеющегося функционала (типовые слои, функции активации и прочее)
- ☐ Высокие показатели производительности на разном аппаратном обеспечении за счет оптимизации на уровне ассемблера

Заключение

- ☐ Выполнен обзор возможностей Intel® neon™ Framework
- ☐ Рассмотрен общий цикл разработки глубоких моделей средствами Intel® neon™ Framework
- ☐ Приведен пример разработки глубокой полносвязной сети с использованием Intel® neon™ Framework
- ☐ Сравнение производительности обучения на разных бэкендах показывает, что использование MKL позволяет повысить эффективность вычислений по сравнению с CPU
- ☐ Сравнение возможностей Intel® neon™ Framework с другими инструментами глубокого обучения показывает, что в neon реализован аналогичный набор моделей. При этом обеспечивается более высокая производительность

Основная литература

- Хайкин С. Нейронные сети. Полный курс. – М.: Издательский дом «Вильямс». – 2006. – 1104 с.
- Осовский С. Нейронные сети для обработки информации. – М.: Финансы и статистика. – 2002. – 344 с.
- Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org].

Авторский коллектив

- **Кустикова Валентина Дмитриевна**
  к.т.н., ст.преп. каф. МОСТ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  valentina.kustikova@itmm.unn.ru
- **Жильцов Максим Сергеевич**
  магистрант каф. МОСТ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  zhiltsov.max35@gmail.com
- **Золотых Николай Юрьевич**
  д.ф.-м.н., проф. каф. АГДМ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  nikolai.zolotykh@itmm.unn.ru