Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики
Кафедра Математического обеспечения и суперкомпьютерных технологий

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

Практическая работа №1 Разработка многослойной полностью связанной сети с использованием средств Intel® neon™ Framework

При поддержке компании Intel

Жильцов М.С.

Нижний Новгород
2018

Содержание

1 Введение

Данная практическая работа направлена на изучение общего цикла решения задач машинного обучения с использованием средств Intel® neon™ Framework. Демонстрация общего цикла проводится на примере решения задачи классификации пола человека по фотографии с использованием многослойных полносвязных сетей. В качестве набора данных используется IMDB-WIKI [6].

2 Методические указания

2.1 Цели и задачи работы

Цель настоящей работы состоит в том, чтобы получить базовые навыки работы с инструментом глубокого обучения Intel® neon™ Framework на примере применения полностью связанных нейронных сетей для решения задачи классификации пола человека по фотографии.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить общую схему решения задач машинного обучения.
2. Изучить общую схему работы с Intel® neon™ Framework.
3. Разработать основной скрипт для обучения и тестирования глубоких нейросетевых моделей, обеспечивающих решение задачи классификации пола человека по фотографии.
4. Разработать модель полносвязной нейронной сети и описать ее средствами Intel® neon™ Framework.
5. Выполнить запуск и оценить качество классификации.

2.2 Структура работы

В работе приводится общая схема разработки и применения глубоких нейросетевых моделей в Intel® neon™ Framework. Вначале выполняется загрузка тестовых и тренировочных данных. Далее приводится пример разработки многослойной полносвязной сети. Разрабатывается скрипт, обеспечивающий обучение и тестирование сети, а также сохранение результатов решения задачи. Работа выполняется на примере задачи классификации пола человека по фотографии. В качестве набора данных используется IMDB-WIKI [6].

2.3 Рекомендации по проведению занятий

При выполнении данной лабораторной работы рекомендуется следующая последовательность действий:

1. Изучить общую схему решения задач машинного обучения.
2. Изучить общую схему разработки глубоких нейросетевых моделей. Для этого можно использовать лекционные материалы и документацию инструмента Neon.
3. Разработать модель многослойной полносвязной сети.
4. Загрузить тренировочные и тестовые данные.
5. Разработать скрипт для обучения и тестирования построенной модели.
6. Добавить сохранение результатов решения задачи в разработанный на предыдущей шаге скрипт.

3 Общая схема решения задач машинного обучения

Решение задачи машинного обучения включает следующие основные этапы:

1. Подготовка данных. Предполагает предварительную обработку данных и преобразование в формат, необходимый для последующего использования.
2. Построение модели. Предполагает выбор метода машинного обучения и его параметров.
3. Обучение модели. Подразумевает настройку параметров модели машинного обучения.
4. Тестирование модели. Предполагает использование построенной модели для решения задачи на данных, которые не использовались на этапе обучения.
5. Сохранение результатов решения задачи и их анализ.

Далее описана реализация перечисленных этапов с использованием Intel® neon™ Framework на примере задачи классификации пола человека по фотографии на данных IMDB-WIKI [6].

4 Инструкция по выполнению работы

4.1 Разработка модели полносвязной сети

Создание модели включает в себя описание структуры модели нейронной сети и задание целевой функции, используемой во время обучения. Модель сети позволяет унифицированным образом представить последовательность преобразований над входными данными. Основным составным элементом модели является слой. Neon предоставляет доступ ко множеству типовых слоев, полный перечень которых можно найти в документации [7].

Для решения задачи классификации с двумя категориями создадим описание простой модели, содержащей один скрытый полносвязный слой (персептрон Розенблатта [4]). Создание полносвязного слоя можно выполнить следующими действиями:

```
from neon.layers import Affine
from neon.initializers import Gaussian, Constant
from neon.transforms import Logistic

layer = Affine(nout=2,
    init=Gaussian(0.1),
    bias=Constant(1),
    activation=Logistic(shortcut=True)
)
```

В этом фрагменте кода создается полносвязный слой с двумя нейронами. Для инициализации весов (связей нейронов с предыдущим слоем) использовано гауссово (нормальное) распределение с нулевым средним и среднеквадратичным отклонением 0.1. При создании полносвязного слоя с помощью **`Affine`** становится доступен параметр **`bias`**, указывающий на необходимость создания дополнительных параметров – смещений. Их инициализация выполнена константным значением, равным единице. Указана логистическая функция активации. Полную информацию о параметрах можно найти в документации [8].

Далее, необходимо создать список слоев сети и модель, которая представляет собой контейнер слоев.

```
from neon.models import Model

layers = [ layer ]
model = Model(layers)
```

В результате получена модель, состоящая из одного полносвязного слоя и имеющая 2 выхода, значения которых являются значениями логистической функции.

Пример более сложной модели (многослойный персептрон Румельхарта):

```
layers = [
    DataTransform(transform=Normalizer(divisor=128.0)),

    Affine(nout=128, init=Xavier(), bias=Uniform(0.3, 0.7),
        activation=Tanh()),
    Affine(nout=64, init=Kaiming(), bias=Constant(0)),
    Affine(nout=2, init=Gaussian(scale=0.1),
        activation=Logistic(shortcut=True))
]
model = Model(layers)
```

В этой модели определяется слой непараметрического входного преобразования данных – деление на константу 128. Такое преобразование позволяет имитировать деление на среднеквадратичное отклонение в случае достаточно большой и разнообразной выборки изображений. Далее создаются 3 полносвязных слоя с разными параметрами инициализации и количеством нейронов.

Такая сеть выполняет над входными данными последовательные преобразования, заданные слоями в описании сети.

Определим целевую функцию, которая используется во время оптимизации параметров сети. В задачах классификации, как правило, целевой функцией является кросс-энтропия. В данном случае используется функция бинарной кросс-энтропии.

```
from neon.transforms import CrossEntropyBinary

cost = GeneralizedCost(costfunc=CrossEntropyBinary())
```

Для удобства разместим код генерации модели в отдельный файл **`mlp_models.py`**. Для каждой модели создадим отдельную функцию генерации следующего вида:

```
def generate_mlp_model():
    layers = [
        DataTransform(transform=Normalizer(divisor=128.0)),

        Affine(nout=128, init=Xavier(), bias=Uniform(0.3, 0.7),
            activation=Tanh()),
        Affine(nout=64, init=Kaiming(), bias=Constant(0)),
        Affine(nout=2, init=Gaussian(scale=0.1),
            activation=Logistic(shortcut=True))
    ]
    model = Model(layers)
    cost = GeneralizedCost(costfunc=CrossEntropyBinary())
    return model, cost
```

В таком случае загрузку модели можно выполнить с помощью кода:

```
import mlp_models

model, cost = models.generate_mlp_model()
```

4.2 Разработка скрипта для обучения и тестирования модели

4.2.1 Общая структура скрипта

Скрипт для обучения и тестирования моделей состоит из нескольких логических частей:

1. Инициализация. Предусматривает, в частности, указание устройства, на котором выполняется обучение и тестирование модели.
2. Подготовка и загрузка данных. Предполагает вызов функций, разработанных в предыдущей практический работе.
3. Создание модели. Предусматривает вызов функций, разработанных ранее в настоящей практической работе.
4. Обучение модели. Предполагает выбор метода оптимизации, настройку его параметров и вызов метода обратного распространения ошибки.
5. Тестирование модели. Подразумевает прямой проход нейросетевой модели для тестового набора данных и определение качества работы построенной модели.
6. Сохранение вывода модели. Предусматривает сохранение значений выхода сети для тестового набора данных.

Рассмотрим более подробно разработку каждой логической части скрипта.

4.2.2 Инициализация

Перед использованием Neon необходимо выполнить инициализацию библиотеки. На этом шаге определяется устройство, на котором будут выполняться вычисления, количество изображений в пачке во время обучения, используемый тип данных и другие параметры. Инициализация выполняется функцией **`gen_backend`** [9]:

```
from neon.backends import gen_backend

be = gen_backend('gpu', batch_size=10)
```

4.2.3 Подготовка и загрузка данных

Действия по подготовке данных детально описаны в предыдущей практической работе. После выполнения этих действий должны быть сформированы файлы с данными обучающей и тестовой выборок набора данных IMDB-WIKI. Будем предполагать, что файлы размещены в директории **data_wiki** и называются **train.h5** и **test.h5**. Для загрузки данных в формате HDF5 используется тип **HDF5Iterator** [10]. Он позволяет загружать данные из внешней памяти пачками, размер которых указывается при инициализации Neon. Для загрузки данных потребуются следующие действия:

```
from neon.data import HDF5Iterator

train_set = HDF5Iterator('data_wiki/train.h5')
test_set = HDF5Iterator('data_wiki/test.h5')
```

В результате создаются объекты, обеспечивающие доступ к элементам набора данных.

Данные, которые предоставляет объект типа **HDF5Iterator**, возвращаются в сохраненном формате. В наборе данных метки классов хранятся в числовом виде. В случае задачи классификации Neon требует предоставления меток классов в one-hot-представлении, в котором целевой класс объектов описывается не числом, а вектором длины, равной количеству классов. Указанный вектор содержит нули во всех элементах, кроме одного, совпадающего с индексом целевого класса. Для автоматического преобразования данных из индексного представления в one-hot представление используется тип **HDF5IteratorOneHot**.

```
from neon.data import HDF5IteratorOneHot

train_set = HDF5IteratorOneHot('data_wiki/train.h5')
test_set = HDF5IteratorOneHot('data_wiki/test.h5')
```

4.2.4 Создание модели

Используя ранее созданный файл с описаниями моделей, создадим модель посредством вызова соответствующей функции.

```
import mlp_models

model, cost = models.generate_mlp_model()
```

4.2.5 Обучение модели

Для обучения модели требуется выбрать алгоритм оптимизации и задать его параметры. Neon предоставляет несколько алгоритмов оптимизации [11]. В глубоком обучении широко используется стохастический градиентный спуск (Stochastic Gradient Descend, SGD). Используем его для оптимизации параметров модели.

```
from neon.optimizers import GradientDescentMomentum

optimizer = GradientDescentMomentum(0.01, momentum_coef=0.9, wdecay=0.0005)
```

В приведенном фрагменте кода создается объект оптимизатора, реализующий вариацию алгоритма градиентного спуска с инерцией (Nesterov's Accelerated Gradient, NAG) [5]. Параметр множителя шага в антиградиентом направлении (learning rate) установлен в значение 0.01. Дополнительно используется L2-регуляризация параметров модели (weight decay) с коэффициентом 0.0005. Полный перечень параметров алгоритма приведен в документации [12].

Для обучения сети необходимо выполнить следующие действия:

```
from neon.callbacks.callbacks import Callbacks

callbacks = Callbacks(model)
model.fit(train_set, optimizer=optimizer,
    num_epochs=10, cost=cost, callbacks=callbacks)
```

Здесь выполняется обучение на тренировочной выборке набора данных. Длительность обучения равна 10 эпохам. Эпоха обучения эквивалентна одному полному обходу набора данных. Для сохранения свойств алгоритма стохастического градиентного спуска потребуется обеспечить случайный выбор данных из набора. Это было выполнено на шаге перемешивания набора данных во время подготовки. Параметр **`callbacks`** позволяет задать функции, которые будут вызываться в процессе обучения. Так, например, можно организовать обход тестового множества данных по завершении эпохи обучения.

4.2.6 Тестирование модели

Для оценки качества классификации используется следующий код:

```
from neon.transforms import Accuracy

accuracy = model.eval(test_set, metric=Accuracy())
print('Accuracy = %.1f%%' % (accuracy * 100))
```

Здесь указывается метрика качества. Для задачи классификации используется показатель точности (**`Accuracy`**), который отражает долю правильно проклассифицированных примеров. Указанная метрика вычисляется для тестового подмножества данных.

4.2.7 Сохранение вывода модели

Конечной целью обучения сети является получение вывода сети на некотором наборе данных. Для получения вывода можно использовать следующий код:

```
outputs = model.get_outputs(test_set)
```

Сохранение вывода в файл можно организовать следующим образом:

```
import numpy as np

def save_inference(output_file_name, outputs, subset):
    with open(output_file_name, 'w') as output_file:
        output_file.write('inference, target\n')
        outputs_iter = iter(outputs)
        try:
            for dataset_batch in subset:
                targets = np.transpose(dataset_batch[1].get())
                for target in targets:
                    output = next(outputs_iter)
                    target_class = np.argmax(target, axis=0)
                    output_file.write('%s, %s\n' % (output, target_class))
        except StopIteration as e: # the last batch might be incomplete
            pass
        output_file.close()

save_inference('inference.txt', outputs, test_set)
```

Данный фрагмент кода выполняет обход набора данных и выходов сети. Результаты сохраняются в файл в формате:

```
[вероятность класса 1, вероятность класса 2], правильный ответ
```

4.3 Запуск обучения и тестирования модели и проведение экспериментов

4.3.1 Запуск скрипта для обучения и тестирования модели без указания параметров

К настоящем моменту предполагается, что разработан скрипт для обучения и тестирования глубокой модели. Для определенности считаем, что он сохранен в файле с именем **`main_classify.py`**. Запуск скрипта осуществляется из командной строки:

```
. .venv/bin/activate
python main_classify.py
```

4.3.2 Параметризация запуска

Процесс запуска имеет множество параметров, относящихся к работе с Neon, источнику данных, нейронной сети, параметрам обучения и тестирования. Все эти параметры можно зафиксировать в скрипте, изменяя их по необходимости. Другим способом является введение параметров запуска скрипта.

Neon предоставляет средства обработки и использования параметров командной строки. Для этого предназначен тип **`NeonArgparser`** [13]. Для его использования потребуется следующий код:

```
from neon.util.argparser import NeonArgparser

parser = NeonArgparser()
args = parser.parse_args()
```

После выполнения этого кода переменная **`args`** будет содержать набор параметров командной строки. Инициализация Neon будет выполнена автоматически с учетом переданных параметров. Задание пользовательских параметров можно выполнить следующим образом:

```
parser.add_argument('--data_root', default='./data_wiki')
```

После разбора параметров значение нового параметра извлекается следующим образом:

```
data_root = args.data_root
train_set = HDF5IteratorOneHot(data_root + '/train.h5')
test_set = HDF5IteratorOneHot(data_root + '/test.h5')
```

4.3.3 Запуск скрипта с указанием входных параметров

Для запуска скрипта потребуется передача некоторых параметров. Запуск можно выполнить с использованием команды вида:

```
python main_classify.py -b gpu -e 10 -z 32 --data_root data_wiki \
    --serizalize 5 --save_path model.prm
```

Рассмотрим более подробно перечень параметров.

- **`b <cpu, mkl, gpu>`** обеспечивает выбор устройства для запуска обучения и тестирования.
- **`e <number>`** указывает количество эпох обучения.
- **`z <number>`** устанавливает размер пачки.
- **`data_root <dir>`** указывает директорию, содержащую данные.
- **`serialize <number>`**, **`save_path <name.prm>`** обеспечивают сохранение модели во время обучения каждые **`<number>`** эпох в файл с именем **`<name.prm>`**.

Полный перечень доступных параметров запуска можно получить, передав опцию **`--help`**.

Для повышения удобства работы рекомендуется создать shell-скрипт, содержащий командную строку запуска. В таком скрипте дополнительно можно выполнить инициализацию переменных окружения и активацию виртуального окружения.

Полные исходные коды примера можно найти в материалах данного курса:

`Practice1_mlp/main_classify.py` и **`Practice/models/mlp_models.py`**.

5 Литература

5.1 Основная литература

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

4. Rosenblatt F. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. – CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961. – №. VG-1196-G-8.
5. Nesterov Y. et al. Gradient methods for minimizing composite objective function. – 2007.

5.2 Ресурсы сети Интернет

6. Домашняя страница набора данных IMDB-WIKI [https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki].
7. Документация Intel® neon™ Framework: слои [http://neon.nervanasys.com/docs/latest/layers.html].
8. Документация Intel® neon™ Framework: слой Affine [http://neon.nervanasys.com/docs/latest/generated/neon.layers.layer.Affine.html#neon.layers.layer.Affine].
9. Документация Intel® neon™ Framework: инициализация [http://neon.nervanasys.com/docs/latest/generated/neon.backends.gen_backend.html#neon.backends.gen_backend].
10. Документация Intel® neon™ Framework: тип HDF5Iterator [http://neon.nervanasys.com/docs/latest/generated/neon.data.hdf5iterator.HDF5Iterator.html#neon.data.hdf5iterator.HDF5Iterator].
11. Документация Intel® neon™ Framework: алгоритмы оптимизации [http://neon.nervanasys.com/docs/latest/optimizers.html].
12. Документация Intel® neon™ Framework: тип GradientDescentMomentum [http://neon.nervanasys.com/docs/latest/generated/neon.optimizers.optimizer.GradientDescentMomentum.html#neon.optimizers.optimizer.GradientDescentMomentum].
13. Документация Intel® neon™ Framework: тип NeonArgparser [http://neon.nervanasys.com/docs/latest/generated/neon.util.argparser.NeonArgparser.html#neon.util.argparser.NeonArgparser].
14. Документация Intel® neon™ Framework: реализация простейшей сети [http://neon.nervanasys.com/docs/latest/mnist.html].