Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая Школа Программной Инженерии

Глубокое обучение на основе пропорциональных маркировок

Курсовая работа

Выполнил
студент гр. 43504/2 С.А. Мишин

Руководитель Черноруцкий И.Г.

«___» _____ 2017 г.

Санкт-Петербург
2017

Содержание

1 Введение

Работа сделана на основе статьи Ehsan Mohammady Ardehaly и Aron Culotta, представителей Иллинойсского технологического института.

Оригинал статьи:

Co-training for Demographic Classification Using Deep Learning from Label Proportions [1]

Опубликована на сайте библиотеки Корнеллского университета, https://arxiv.org/pdf/1709.04108.pdf

В последние несколько лет в сфере принятия решений очень активно развиваются нейронные сети и технологии связанные с ними. Одной из новых идей стало “Глубокое обучение на основе пропорциональных маркировок”, когда конкретные экземпляры (samples) объединяются в группы, которым присваиваются пропорциональные маркеры (labels) [2]. В дальнейшем именно эти группы используются в качестве основы для обучения сети.

Глубокое обучение на основе пропорциональных маркировок (Deep LLP) хорошо показало себя в классификации неточных данных[1][3][4], к примеру, на демографической классификации по изображению.

Из-за новизны данного направления, пока не существует эталонных реализаций нейронных сетей с Deep LLP, поэтому появилась задача создать понятную реализацию.

1.1 Цель работы

Целью курсовой работы является реализация алгоритма **Обучения на основании пропорциональных маркировок**, который является гибридным алгоритмом обучения за счёт того, что известна только вероятность нахождения определённого класса среди группы объектов.

2 Основная часть

2.1 Терминология

- **Deep LLP** — Deep Learning from Label Proportions, обучение на основании пропорциональных маркировок
- **CNN** — Convolution Neural Network, свёрточная нейронная сеть
- **MNIST** — Modified National Institute of Standards and Technology database, база данных образцов рукописного написания цифр
- **batch** — группа, содержащая в себе несколько экземпляров из набора исходных данных.
- **bag** — сумка, содержит в себе набор исходных данных для обучения и маркировку пропорций. Обычно, размер сумки совпадает с размером группы.

2.2 Реализация

Реализация алгоритма обучения на основе пропорциональных маркировок произведена с помощью модификации стандартной свёрточной нейронной сети для классификации рукописных цифр (MNIST).

Для реализации был выбран фреймворк Keras[1], поддерживающий два низкоуровневых фреймворка TensorFlow[2] и Theano[3]

[1] https://keras.io
[2] https://tensorflow.org/
[3] http://deeplearning.net/software/theano/

2.2.1 Настройка модели

Для корректной работы алгоритма обучения на основе пропорциональных маркировок необходимо реализовать функцию определения пропорционального маркера – *create_bags()*:

```
def create_bags(input_data, labels, max_batch_size):

    # output probabilities
    the_y_probs = None

    # get next `max_batch_size' pieces of the `input_data'
    input_data_length = len(input_data)
    for lower_bound in range(0, input_data_length, max_batch_size):

        # Check the top limit
        if lower_bound + max_batch_size >= input_data_length:
            upper_bound = input_data_length
        else:
            upper_bound = lower_bound + max_batch_size

        the_batch = input_data[lower_bound : upper_bound]

        # Find the probability of each class
        the_labels = labels[lower_bound : upper_bound]

        the_probs = sum(the_labels) / max_batch_size # an array [y0, y1, ...]
        the_probs = the_probs.reshape(1, -1)
        the_probs = np.repeat(the_probs, upper_bound - lower_bound, axis=0)

        if the_y_probs is None:
            the_y_probs = the_probs
        else:
            the_y_probs = np.append(the_y_probs, the_probs, axis=0)

    return the_y_probs
```

Принцип работы функции следующий:

1. Из входных данных выбирается группа экземпляров, например размером 16 штук с сумке.
2. Для этой группы определяется пропорциональность (статистическая вероятность) вхождения каждого класса элемента. В случае с MNIST таких классов 10 (цифры от 0 до 9).
3. Далее полученные маркировки распространяются на все элементы, вошедшие в группу.
4. Все маркировки добавляются в один список (аналогично вектору y, предназначенному для обучения), который в последствии заменяет набор y для обучения, **y_train**.

Таким образом, будем использовать полученные сумки в качестве набора для обучения **y**:

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_train = create_bags(x_train, y_train, batch_size)
```

Также, для функции потерь (loss function) будем использовать KL-Divergence[4] (Kullback-Leibler divergence).

В общем случае, если μ – любая мера на X, для которой существуют абсолютно непрерывные относительно μ функции $p = \frac{\mathrm{dP}}{\mathrm{d}\mu}$ и $q = \frac{\mathrm{dQ}}{\mathrm{d}\mu}$, тогда расхождение Кульбака–Лейблера распределения Q относительно P определяется как:

$$D_{KL}(P||Q) = \int_X p \log \frac{p}{q} d\mu \tag{1}$$

Укажем её в качестве функции потерь:

```
model.compile(loss=keras.losses.kullback_leibler_divergence, # using KL-divergence
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

В качестве размера группы будем использовать 16:

```
batch_size = 16
```

2.3 Тестирование

Тестирование проводилось в сравнении с **mnist_cnn.py**, были получены следующие результаты:

1. MNIST CNN [5]: Точность на тестовых данных 98.2%
2. Собственный Deep LLP: Точность на тестовых данных 95.44%

К сожалению, побить результат отлаженного алгоритма MNIST CNN, возможные причины:

- В “чистом” варианте MNIST изначально используются “строгие” ручные маркировки. В варианте LLP моделируется ситуация отсутствия таких точных значений, а система больше рассчитывается на использование с неизвестными данными (“in the wild”)
- В MNIST достаточно большое количество классов – 10, это резко уменьшает количество сумок и точность их маркировки, т.к. для одной сумки используется несколько экземпляров в группе (batch). Чем больше группа, тем точнее маркировка, но тем меньше общее количество групп, что сильно влияет на качество обучения.

К примеру, если уменьшить размер группы с 16 до 10, то точность правильного определения цифры повышается до 97.47%.

[4] https://en.wikipedia.org/wiki/Kullback–Leibler_divergence

3 Выводы

Глубокое обучение на основе пропорциональных маркировок показывает неплохой потенциал в машинном обучении. Исследования показывают, что они отлично справляются с распознаванием нечётких характеристик (демографических, или, например, обледенелой поверхности воды). Но, к сожалению, сейчас не существует проектов с открытым исходным кодом для изучения.

В данном проекте на текущий момент реализована удобная система нахождения маркера для группы, и использования его при обучении.

4 Литература

[1] E. M. Ardehaly и A. Culotta, ”Co-training for Demographic Classification Using Deep Learning from Label Proportions Illinois Institute of Technology

[2] N. Quadrianto, A. J. Smola, T. S. Caetano и Q. V. Le, ”Estimating Labels from Label Proportions Journal of Machine Learning Research

[3] F. Li и G. Taylor, “Alter-cnn: An approach to learning from label proportions with application to ice-water classification,” in Neural Information Processing Systems Workshops (NIPSW) on Learning and privacy with incomplete data and weak supervision, 2015.

[4] X. Yu, Scalable Machine Learning for Visual Data, Columbia University

[5] Y. Lecun, L. Bottou, Y. Bengio и P Haffner, ”Gradient-Based Learning Applied to Document Recognition Proceedings Of The IEEE, 1998

5 Приложения

5.1 Системные требования

Для работы и запуска нейронной сети требуется:

- Unix-совместимая операционная система
- Python 3.6, библиотеки:
  - NumPy 1.12.1
  - TensorFlow 1.4.0
  - Keras 2.1.2

5.2 Исходный код

Исходный код модели может быть найден по адресу https://github.com/qezz/simple-deep-llp/blob/master/keras/project/src.py

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras import backend as K

import random
import numpy as np

def create_bags(input_data, labels, max_batch_size):

    # the_y_probs = keras.utils.to_categorical(y_train, num_classes)
    the_y_probs = None

    # get next `max_batch_size' pieces of the `input_data'
    input_data_length = len(input_data)
    for lower_bound in range(0, input_data_length, max_batch_size):

        # Check the top limit
        if lower_bound + max_batch_size >= input_data_length:
            upper_bound = input_data_length
        else:
            upper_bound = lower_bound + max_batch_size

        the_batch = input_data[lower_bound : upper_bound]

        # Find the probability of each class
        the_labels = labels[lower_bound : upper_bound]

        the_probs = sum(the_labels) / max_batch_size # an array [y0, y1, ...]
        the_probs = the_probs.reshape(1, -1)
        the_probs = np.repeat(the_probs, upper_bound - lower_bound, axis=0)

        if the_y_probs is None:
            the_y_probs = the_probs
        else:
            the_y_probs = np.append(the_y_probs, the_probs, axis=0)

    return the_y_probs


batch_size = 10
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)


x_train = x_train.astype('float32')
```

```
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

y_train = create_bags(x_train, y_train, batch_size)
# y_test = create_bags(x_test, y_test, batch_size)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
# model.add(BatchNormalization()) # not so good for this dataset; drops accuracy from 0.98 to 0.92

model.compile(loss=keras.losses.kullback_leibler_divergence, # using KL-divergence
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))

# for epoch in epochs:



score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```