Подготовка к сборке

Сборка библиотеки возможна под операционными системами Linux и Windows. Для сборки необходим OpenCV версии 2.9 или новее. Помимо этого потребуются следующие средства для сборки:
Для Winodws:
1. Visual Studio 2012 или новее;
2. CMake 2.8 или новее.
Для Linux:
1. g++ 4.7 или новее;
2. CMake 2.8 или новее;
3. Make.

Если сборка осуществляется под Windows, то перед сборкой необходимо открыть файл **CMakeLists.txt** любым текстовым редактором (не рекомендуется использовать встроенный в Windows блокнот Notepad) и отредактировать следующую строку, прописав сюда корректный путь до OpenCV в вашей системе:

```
set(OpenCV_DIR C:/NeuralNetworks/opencv/opencv/build)
```

Обратите внимание, что в пути используется косая черта «/», а не обратная косая черта «\», которая характерна для путей в ОС Windows.

OpenCV должен содержать набор библиотек, скомпилированных под вашу версию Visual Studio.

Сборка и компиляция

Откройте консоль и перейдите в папку с библиотекой BNet с помощью команды:

```
cd путь_до_папки
```

Затем выполните сборку:

```
mkdir build
cd build
cmake ..
```

Далее, если у вас Linux выполните команду make для компиляции библиотеки. Если у вас Windows откройте в проводнике папку build и запустите появившийся в ней файл проекта для Visual Studio, выберите тип сборки Release и выполните компиляцию.

Построение и обучение искусственной нейронной сети

Откройте файл BNet.cpp в вашей среде программирования.

Для того чтобы объявить новую полносвязную нейронную сеть просто объявите переменную типа `Net` в теле функции main(). Например, так

```
Net nn;
```

К новой объявленной нейронной сети в первую очередь нужно добавить входной слой нейронов. Для этого можно воспользоваться методом addInputLayer(количество_нейронов слоя) класса Net. Например:

```
nn.addInputLayer(2);
```

Далее можно добавить любое число скрытых слоев, воспользовавшись методом addHiddenLayer(количество_нейронов слоя, имя_функции_активации_нейронов_слоя) класса Net.
Например:

```
nn.addHiddenLayer(2,LOGISTIC);
```

Все доступные функции активации и их имена вы можете найти в файле include/activation.hpp в перечислении ActivationFunctions.

После добавления нужного числа скрытых слоев, для завершения построения нейронной сети, необходимо добавить выходной слой. Для этого можно воспользоваться методом addOutputLayer(количество_нейронов слоя, имя_функции_активации_нейронов_слоя) класса Net. Например:

```
nn.addOutputLayer(1,LOGISTIC);
```

На этом построение сети закончено. Теперь необходимо загрузить в программу данные для её обучения.

Для хранения таких данных библиотека использует класс trainDataCollection. Для объявления нового набора обучающих или тестовых данных необходимо объявить переменную типа trainDataCollection. Размерность данных должна соответствовать построенной сети. Размерность данных должна быть указана в полях inputsCount и outputsCount этого класса. Размерность самой коллекции задается полем collectionSize.
Внутри trainDataCollection данные хранятся в виде набора объектов класса trainData. Объект trainData хранит одну связанную пару входы-выходы в виде двух массивов (поля inputs и outputs класса trainData). После создания и заполнения, объект trainData должен быть добавлен к коллекции внутри trainDataCollection с помощью вызова trainCollection.push_back(указатель_на_объект_trainData) для нужно объекта. Следующий пример демонстрирует создание коллекции примеров работы оператора "исключающее или" (XOR) и их заполнение.

```
//Объвить коллекцию примеров
trainDataCollection tr;
tr.inputsCount = 2; //должно соответствовать размеру
входного слоя нейронной сети
tr.outputsCount = 1; //должно соответствовать размеру
выходного слоя нейронной сети
tr.collectionSize = 4; //количество примеров в коллекции
//Следующий код создает 4 примера работы оператора
"исключающее или" (XOR) и заполняет их.
```

```
trainData **dt = new trainData*[tr.collectionSize];
for(int i=0;i<tr.collectionSize;i++) dt[i] = new
trainData(tr.inputsCount, tr.outputsCount);
dt[0]->inputs[0] = 1; dt[0]->inputs[1] = 0; dt[0]-
>outputs[0] = 1;
dt[1]->inputs[0] = 1; dt[1]->inputs[1] = 1; dt[1]-
>outputs[0] = 0;
dt[2]->inputs[0] = 0; dt[2]->inputs[1] = 1; dt[2]-
>outputs[0] = 1;
dt[3]->inputs[0] = 0; dt[3]->inputs[1] = 0; dt[3]-
>outputs[0] = 0;
//доавить примеры в общую коллекцию
for(int i=0;i<tr.collectionSize;i++)
tr.trainCollection.push_back(dt[i]);
```

Все приготовления завершены. Теперь можно приступать к обучению нейронной сети. Обучение начинается при вызове метода train(набор_данных_для_обучения) класса Net.
В нашем примере это:

```
nn.train(tr);
```

Обучение может занимать продолжительное время в зависимости от размеров сети, а также сложности и количества данных для ее обучения. Во время обучения будет с некоторой периодичность выводиться информация о текущем состоянии обучения, такая как текущая итерация и ошибка сети. Обучение завершится при достижении сетью ошибки 0.0001 или после 10000000 итераций. Скорректировать эти критерии можно в методе `void Net::train(trainDataCollection &_trainCollection)` в файле sources/net.cpp.
Каждую десятую итерацию библиотека будет сохранять текущую обученную модель в файл models/model_tmp.mdl. Также при модель с самым лучшим показателем сети по ошибке будет автоматически сохраняться в файл models/model_best.mdl. Изменить эти параметры можно в том же методе.

Обучение можно в любой момент прервать нажав комбинацию клавиш «Ctrl+C», когда окно консоли находится в фокусе. Обучение сети можно будет продолжить позже, не потеряв достигнутых результатов. Как это сделать читайте в главе «Дообучение уже существующей сети».

Библиотека BNet не налагает никаких ограничений на тип данных, с которыми можно работать, будь то биржевые показатели или пиксели изображения. Пример загрузки изображений, можно посмотреть в файле sources/trainDataCollection.cpp в методе
bool trainDataCollection::loadImagesFromLst(const char* fileName, int width, int height).

После завершения обучения финальную модель можно сохранить воспользовавшись методом saveModel(путь_до_файла_сохранения) класса Net.

Например:
```
nn.saveModel("../models/model_xor.mdl");
```

Ниже представлен полный текст рассмотренного в этой главе примера с обучением нейронной сети действовать как оператор XOR.

```
Net nn;
nn.addInputLayer(2);
nn.addHiddenLayer(2,LOGISTIC);
nn.addOutputLayer(1,LOGISTIC);

//Объвить коллекцию примеров
trainDataCollection tr;
tr.inputsCount = 2; //должно соответствовать размеру
входного слоя нейронной сети
tr.outputsCount = 1; //должно соответствовать размеру
выходного слоя нейронной сети
tr.collectionSize = 4; //количество примеров в коллекции

//Следующий код создает 4 примера работы оператора
"исключающее или" (XOR) и заполняет их.
trainData **dt = new trainData*[tr.collectionSize];
for(int i=0;i<tr.collectionSize;i++) dt[i] = new
trainData(tr.inputsCount, tr.outputsCount);
dt[0]->inputs[0] = 1; dt[0]->inputs[1] = 0; dt[0]-
>outputs[0] = 1;
dt[1]->inputs[0] = 1; dt[1]->inputs[1] = 1; dt[1]-
>outputs[0] = 0;
dt[2]->inputs[0] = 0; dt[2]->inputs[1] = 1; dt[2]-
>outputs[0] = 1;
dt[3]->inputs[0] = 0; dt[3]->inputs[1] = 0; dt[3]-
>outputs[0] = 0;

//доавить примеры в общую коллекцию
for(int i=0;i<tr.collectionSize;i++)
tr.trainCollection.push_back(dt[i]);

//начать обучение сети
nn.train(tr);

//сохранить обученную модель
nn.saveModel("../models/model_xor.mdl");
```

Работа с обученной сетью.

Чтобы работать с обученной сетью, необходимо загрузить в программу файл с обученной моделью воспользовавшись методом

loadModel(путь_до_файла_сохранения) класса Net. Затем необходимо сформировать массив чисел с плавающей точкой с входными данными для нейронной сети и пустой массив для хранения ответов сети, и передать указатели на эти массивы в метод calculate(указатель_на_массив_входов, указатель_на_массив_результатов); класса Net.

Ниже приведен пример работы с обученной в прошлой главе нейронной сетью.

```
Net nn;
nn.loadModel("../models/model_xor.mdl");
double input[2];
double output[1];
input[0] = 0; input[1] = 1;
nn.calculate(input, output);
//вывод результата работы
printf("%f XOR %f -> %f\n", input[0], input[1],
output[0]);
```

Обученную модель можно протестировать на коллекции данных, записанной в trainDataCollection, как это было показано в предыдущей главе. Для этого загруженную коллекцию необходимо передать методу runTest(коллекция_trainDataCollection) класса Net.
Например,

```
nn.runTest(tr);
```

Выполнив тестирование сети можно получить отчет повремени, затраченному библиотекой на обработку входных данных, и точности работы сети.

Дообучение уже существующей сети

Если точность существующей сети не устраивает или нужно обучить старую сеть на новых данных, то библиотека предоставляет возможность выполнить дообучение. Чтобы дообучать сеть достаточно лишь загрузить существующую модель сети, как это показано в предыдущей главе, загрузить коллекцию данных trainDataCollection и вызвать обучение на этой коллекции, также как показано в главе «Построение и обучение искусственной нейронной сети».

Ниже представлен пример дооучения модели.

```
Net nn;
nn.loadModel("../models/model_xor.mdl");
//Объвить коллекцию примеров
trainDataCollection tr;
tr.inputsCount = 2; //должно соответствовать размеру
входного слоя нейронной сети
```

```
tr.outputsCount = 1; //должно соответствовать размеру
выходного слоя нейронной сети
tr.collectionSize = 4; //количество примеров в коллекции

//Следующий код создает 4 примера работы оператора
"исключающее или" (XOR) и заполняет их.
trainData **dt = new trainData*[tr.collectionSize];
for(int i=0;i<tr.collectionSize;i++) dt[i] = new
trainData(tr.inputsCount, tr.outputsCount);
dt[0]->inputs[0] = 1; dt[0]->inputs[1] = 0; dt[0]-
>outputs[0] = 1;
dt[1]->inputs[0] = 1; dt[1]->inputs[1] = 1; dt[1]-
>outputs[0] = 0;
dt[2]->inputs[0] = 0; dt[2]->inputs[1] = 1; dt[2]-
>outputs[0] = 1;
dt[3]->inputs[0] = 0; dt[3]->inputs[1] = 0; dt[3]-
>outputs[0] = 0;

//доавить примеры в общую коллекцию
for(int i=0;i<tr.collectionSize;i++)
tr.trainCollection.push_back(dt[i]);

//начать обучение сети
nn.train(tr);

//сохранить обученную модель
nn.saveModel("../models/model_xor.mdl");
```