Дата последней сборки документа:
14 февраля 2024 г. в 21:40

Приемы работы с библиотекой PyTorch

Содержание

1. Вводные замечания

Чаще всего цикл обучения модели реализуют в виде обычного цикла `for` Python. Оптимизатор, доступный в модуле `torch.optim` PyTorch, который будет отвечать за обновление параметров.

По умолчанию в PyTorch используется модель немедленного выполнения (eager mode). Как только интерпретатор Python выполняет инструкцию, связанную с PyTorch, базовая реализация C++ или CUDA сразу же производит соответствующую операцию.

PyTorch также предоставляет возможности предварительной компиляции моделей с помощью TorchScript. Используя TorchScript, PyTorch может преобразовать модель в набор инструкций, которые можнро независимо вызывать из Python, допустим, из программ на C++ или на мобильных устройствах. Это можно считать своего рода виртуальной машиной с ограниченным набором инструкций, предназначенным для опреаций с тензорами. Экспортировать модель можно либо в виде TorchScript для использования со средой выполнения Python, либо в стандартизированном формате ONNX (платформонезависимый формат описания моделей).

Сети среднего размера могут потребовать от нескольких часов до нескольких дней для обучения с нуля на больших реальных наборах данных на рабочих станциях с хорошим GPU [1, стр. 48]. Длительность обучения можно сократить за счет использования на одной машине нескольких GPU или даже еще сильнее – на кластере машин, оснащенных несколькими GPU.

Для примера создадим сеть AlexNet

```
# TorchVision включает несколько лучших нейросетевых архитектур для машинного зрения
from torchvision import models

alexnet = models.AlexNet()
```

Подав на вход `alexnet` данные четко определенного размера, мы выполним прямой проход (forward pass) по сети, при котором входной сигнал пройдет через первый набор нейронов, выходные сигналы которых будут поданы на вход следующего набора нейронов, и так до самого итогового выходного сигнала. На практике это означает, что при наличии объекта `input` нужного типа можно произвести прямой проход с помощью оператора `ouput = alexnet(input)`.

Но если мы так поступим, то получим мусор. А все потому, что сеть не была инициализирована: ее веса, числа, с которыми складываются и на которые умножаются входные сигналы, не были обучены на чем-либо, сеть сама по себе – чистый (или, точнее, сказать случайный) лист. Необходимо либо обучить ее с нуля, либо загрузить веса, полученные в результате предыдущего обучения [1, стр. 58].

В `models` названия в верхнем регистре соответствуют классам, реализующим популярные архитектуры, предназначенные для машинного зрения. С другой стороны, названия в нижнем регистре соответствуют функциям, создающим экземпляры моделей с заранее определенным количеством слоев и нейронов, а также, возможно, скачивающие и загружающие в них предобученные веса.

Для того чтобы привести входные изображения к нужному размеру, а их значения (цвета) примерно в один числовой диапазон, можно воспользоваться преобразованиями модуля `torchvision`

```
from torchvision import transforms

# это функция
preprocess = transformers.Compose([
    transformers.Resize(256),
    transformers.CenterCrop(224),
    transformers.ToTensor(),
    transformers.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

Здесь описана функция `preprocess`, масштабирующую входное изображение до размера 256×256, обрезающую его до 224×224 по центру, преобразующую в тензор (многомерный массив PyTorch: в данном случае трехмерных массив, содержащий цвет, высоту и ширину) и нормализующую его компоненты RGB (красный, зеленый, синий) до заданных среднего значения и стандартного отклонения.

Если мы хотим получить от сети осмысленные ответы, все это должно соответствовать данным, полученным сетью во время обучения.

Процесс выполнения обученной модели на новых данных в сфере глубокого обучения называется *выводом* (inference). Для выполнения вывода необходимо перевести сеть в режим `eval`

```
resnet.eval()
```

Если забыть сделать это, некоторые предоубченные модели, например включающие нормализацию по мини-батчам и дропаут, не дадут никаких осмысленных результатов просто по причине их внутреннего устройства. Теперь, после установки режима `eval`, можно выполнять вывод

```
out = resnet(batch_t)
# получается что-то вроде степени уверенности модели в конкретном предсказании
percentage = torch.nn.functional.softmax(out, dim=1)[0] * 100
```

Успешность работы сети во многом зависит от наличия соответствующих объектов в обучающем наборе данных. Если подать нейронной сети нечто выходящее за рамки обучающего набора данных, вполне возможно, что она достаточно уверенно вернет неправильный ответ [1, стр. 64].

Список литературы

1. *Стивенс Э.* PyTorch. Освещая глубокое обучение. – СПб.: Питер, 2022. – 576 с.