

Основы практического использования нейронных сетей.

Лекция 10. Средства работы с НС.

Дмитрий Буряк.
к.ф.-м.н.
dyb04@yandex.ru

Top 8 Deep Learning Frameworks



Caffe



Средства работы с НС



cuDNN

- ❑ Необходима низкоуровневая работа с памятью GPU (CUDA функции).
- ❑ Реализованы базовые типы слоев НС, градиенты, базовый тип данных – тензор.
- ❑ Для практической работы с НС требуется много времени на реализацию и отладку – практически всегда используются более высокоуровневые средства.
- ❑ Максимальная скорость работы по сравнению с высокоуровневыми средствами.

TensorFlow/Torch

- ❑ Вычисления представляются в виде вычислительного графа
- ❑ Реализовано автоматическое дифференцирование

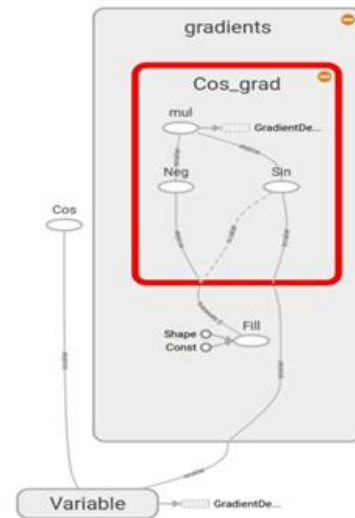
```
x = tf.Variable(initial_value=3.0)
```

```
y = tf.cos(x)
```

```
train = tf.train.GradientDescentOptimizer().minimize(y) with tf.Session() as sess:
```

```
writer = tf.summary.FileWriter('logs', sess.graph) writer.close()
```

- ❑ Вызовы функций cuDNN для реализации вычислений
- ❑ Максимальная скорость работы по сравнению с высокоуровневыми средствами.



Keras/PyTorch

- ❑ Удобное API для работы с НС.
- ❑ Большое количество реализованных типов слоев, функций потерь, алгоритмов обучения.
- ❑ Применение предобработки/аугментации данных предусмотрено в общем конвейере обучения НС.
- ❑ Возможности создания собственных функций потерь, слоев и их простая интеграция в основной функционал.
- ❑ Функции поддержки распределенных вычислений для НС.
- ❑ Наборы предобученных сетей.
- ❑ Оперативная реализация и включение в релиз новых подходов

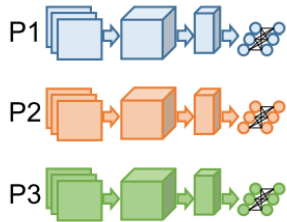
НС в среде Matlab

- Neural Network toolbox.
- Поддержка глубоких сетей с 2016г.
- Встроенные средства распараллеливания (требуется установка Parallel Computing Toolbox).
- Высокий уровень абстракции при обработке данных.
- Наличие различных средств обработки данных разной природы в той же среде.
- Ограниченный набор архитектур и алгоритмов обучения.
- Сложность внесения изменений в существующие архитектуры и алгоритмы обучения.

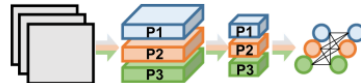


Распределенные вычисления НС

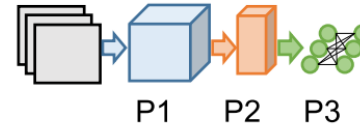
- ❑ Нехватка вычислительных ресурсов
- ❑ Потребность в памяти для модели/данных



(a) Data Parallelism



(b) Model Parallelism



(c) Layer Pipelining

-
- Много параметров в памяти
 - + Меньше коммуникаций

- + Выигрыш в памяти
- Затратные коммуникации

Распараллеливание по данным (1)

- ❑ Разделение обучающих данных на части (по числу вычислительных узлов)
- ❑ На каждом узле вычисляется градиент по своей обучающей подвыборке
- ❑ Формирование общего градиента
- ❑ Обновление моделей на узлах

- ❑ Синхронный SGD
 - «усреднение» градиентов от узлов → более устойчивый процесс обучения
 - однократное обновление модели (в рамках одной итерации)
 - быстродействие определяется самым «медленным» узлом.
- ❑ Асинхронный SGD
 - Модель обновляется по градиенту от каждого узла
 - Требуется меньше времени на обработку пакета обучения (минибатча)

Распараллеливание по данным (2)

❑ Реализация в PyTorch

▪ Синхронный SGD

```
from torch.nn.parallel import DistributedDataParallel as DDP

# `model` is the model we previously initialized
model = ...

# `rank` is a device number starting from 0
model = model.to(rank)
ddp_model = DDP(model, device_ids=[rank])
```

▪ Асинхронный SGD: требуется самостоятельная [реализация](#).

❑ Реализация в TensorFlow (Keras)

▪ Синхронный SGD

```
import tensorflow as tf
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
    model = Model(...)
    model.compile(...)
```

▪ Асинхронный SGD: `tf.distribute.experimental.ParameterServerStrategy`

Распараллеливание модели

❑ Разделение НС на N частей (1 часть на узел)

- По слоям
- Учитывая особенности архитектуры
- Требуется синхронизация

❑ Реализация в PyTorch

- Создание двух линейных слоев, размещенных на разных GPU

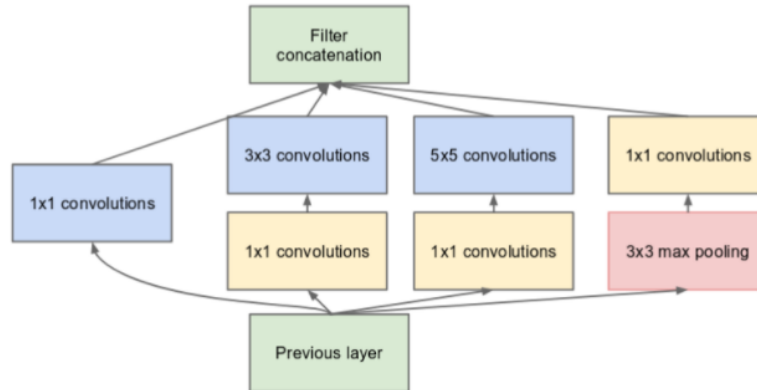
```
import torch.nn as nn
linear1 = nn.Linear(16, 8).to('cuda:0')
linear2 = nn.Linear(8, 4).to('cuda:1')
```

❑ Реализация в TensorFlow (Keras)

```
import tensorflow as tf from tensorflow.keras
import layers
with tf.device('/GPU:0'):
    linear1 = layers.Dense(8, input_dim=16)
with tf.device('/GPU:1'):
    linear2 = layers.Dense(4, input_dim=8)
```

Сравнение

- ❑ Распараллеливание по данным применяют чаще
 - Проще реализация и отладка
 - Подходит для многих задач
- ❑ Особенности распараллеливания модели
 - Необходима, если модель слишком большая для одного узла
 - Требуется учитывать архитектуру НС
 - Хорошо подходит для сетей с параллельными ветвями



Вопросы

- Иерархия программных средств работы с НС
- Отличие распараллеливания по данным от распараллеливания модели.
- Синхронный и асинхронный SGD