ML Praktikum WS 17/18
Introduction to ML software frameworks

Martin Mundt

FIAS / Goethe Uni Frankfurt

February 2018

1. Some Deep Learning software requirements
2. TensorFlow
3. PyTorch
4. GPU acceleration

DL software equirements

- N-dimensional matrices/tensors with common operations & convenient slicing
- Ability to group mathematical functions into more abstract building blocks (like layers, nets)
- Definition of complicated execution sequences of more than two building blocks (think of RNNs)
- A set of common training algorithms with parameter choices & sensible defaults
- Access to databases that are common in the machine learning domain

DL software requirements

- Compatibility with fundamental backends such as BLAS or GPU SDKs
- Seamless GPU & multi-GPU support (abstract away transfers & synchronization)
- Potential to distribute data & networks to multiple machines (or cloud)
- A "model-zoo", i.e. pre-trained models
- Target common Operating Systems
- Be installable & usable for non-experts
- Little computational & memory overhead through interfaces to e.g. Python

TensorFlow

TensorFlow

TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API

TensorFlow

TensorFlow

Numerical computation library
Express in Python
Underlying implementation: C++, CUDA

TensorFlow

Created by: Google Brain
Initial beta release: November 2015
Latest stable release: TensorFlow 1.5, January 2018
License: Open source - Apache 2.0

Spiritual successor to Theano (2010) (discontinued 11/17)

Website: https://www.tensorflow.org/
Git: https://github.com/tensorflow/tensorflow

TensorFlow

A summary of core features:

- **Data Flow Graphs**: describe mathematical computation with a directed graph of nodes & edges
- **Deep Flexibility:** if you can express your computation as a data flow graph, you can use TensorFlow
- **True Portability:** TensorFlow runs on CPUs or GPUs, and on desktop, server, or mobile computing platforms
- **Auto-Differentiation**
- **Maximize Performance:** Support for threads, queues & asynchronous computation. Freely assign compute elements to different devices
- **Estimators:** Has pre-defined set of commonly used estimators.

TensorFlow

TensorFlow - ”old style example”

```
import tensorflow as tf
sess = tf.Session()
matrix1 = tf.constant([[3.],[3.]])
matrix2 = tf.constant([[3.],[3.]])
product = tf.matmul(matrix1,matrix2)
result = sess.run(product)
print(result)
sess.close()
```

Neural Network

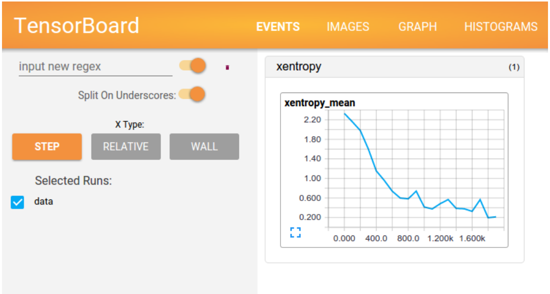
```
    import tensorflow as tf
    x = tf.placeholder("float", [None, n_input])
    y = tf.placeholder("float", [None, n_classes])
    def multilayer_perceptron(_X, _weights, _biases):
        layer_1 = tf.nn.relu(tf.add(tf.matmul(_X, _weights['h1']), _biases['b1']))
        layer_2 = tf.nn.relu(tf.add(tf.matmul(layer_1, _weights['h2']),_biases['b2']))
        return tf.matmul(layer_2, weights['out']) + biases['out']
...
# Initialize variables
# Launch the graph
```

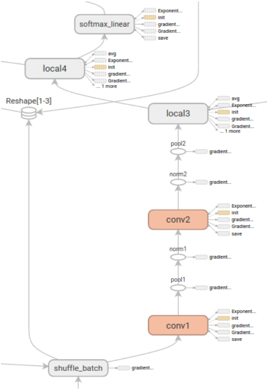
TensorFlow - Layers API

Neural Network from: https://github.com/tensorflow/models/blob/master/official/mnist/mnist.py

```
class Model(object):

    def __init__(self, data_format):
        self._input_shape = [-1, 1, 28, 28]

        self.conv1 = tf.layers.Conv2D(
        32, 5, padding='same', data_format=data_format, activation=tf.nn.relu)
        self.conv2 = tf.layers.Conv2D(
        64, 5, padding='same', data_format=data_format, activation=tf.nn.relu)
        self.fc1 = tf.layers.Dense(1024, activation=tf.nn.relu)
        self.fc2 = tf.layers.Dense(10)
        self.max_pool2d = tf.layers.MaxPooling2D(
    (2, 2), (2, 2), padding='same', data_format=data_format)

    def __call__(self, inputs, training):
        y = tf.reshape(inputs, self._input_shape)
        y = self.conv1(y)
        y = self.max_pool2d(y)
        y = self.conv2(y)
        y = self.max_pool2d(y)
        y = tf.layers.flatten(y)
        y = self.fc1(y)
        return self.fc2(y)
```

TensorFlow





- Interactive playground: http://playground.tensorflow.org/
- Has been adapted for other frameworks

PyTorch

PyTorch

PYTORCH Get Started About Blog Support Tutorials Discuss Docs

PyTorch is a python package that provides two high-level features:

- Tensor computation (like numpy) with strong GPU acceleration
- Deep Neural Networks built on a tape-based autograd system

You can reuse your favorite python packages such as numpy, scipy and Cython to extend PyTorch when needed.

At a granular level, PyTorch is a library that consists of the following components:

Package	Description
torch	a Tensor library like NumPy, with strong GPU support
torch.autograd	a tape based automatic differentiation library that supports all differentiable Tensor operations in torch
torch.nn	a neural networks library deeply integrated with autograd designed for maximum flexibility
torch.optim	an optimization package to be used with torch.nn with standard optimization methods such as SGD, RMSProp, LBFGS, Adam etc.
torch.multiprocessing	python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and hogwild training.
torch.utils	DataLoader, Trainer and other utility functions for convenience
torch.legacy(.nn/.optim)	legacy code that has been ported over from torch for backward compatibility reasons

Usually one uses PyTorch either as:

- A replacement for numpy to use the power of GPUs.
- a deep learning research platform that provides maximum flexibility and speed

PyTorch

Tensor computation library
Express in Python (or LUA)
Underlying implementation: C, CUDA

PyTorch

Created by: Facebook AI
Initial beta release: January 2017
Latest stable release: PyTorch 0.3, Dezember 2017
License: Open source - BSD-3

Addition to Torch version 7 (2012: same C backend + LUA, still maintained)

Website: https://www.tensorflow.org/
Git: https://github.com/tensorflow/tensorflow

PyTorch

A summary of core features:

- **GPU-ready Tensor library**: if you use numpy, you have used Tensors.
- **Dynamic Neural Networks: Tape based Autograd** unique way of building neural networks: using and replaying a tape recorder.
- **Python first**
- **Auto-Differentiation**
- **Fast and Lean** At the core, CPU and GPU Tensor and Neural Network backends (TH, THC, THNN, THCUNN) are written as independent libraries with a C99 API. They are mature and have been tested for years.
- **Extensions without pain** You can write new neural network layers in Python using the torch API.
- **Torchvision** datasets and utility for computer vision.

PYTORCH

PyTorch - "old style example"

```
import torch
matrix1 = torch.Tensor(3,3)
matrix2 = torch.Tensor(3,3)
product = torch.matmul(matrix1,matrix2)
print(product)
```

Neural Network

```
from torch import nn as nn
class Model(nn.Module):
        def __init__(self):
                super(Model, self).__init__()
                net = nn.Sequential(nn.Linear(2,2), nn.Linear(2,2))
        def forward(self, x):
                x = net(x)
                return x

model = Model()
result = model(torch.autograd.Variable(torch.rand(2)))
print(result)
```

PyTorch - (dynamically) defining forward

```
from torch import nn as nn
import torch.nn.functional as F

class Model(object):

        class Model(nn.Module):
                def __init__(self):
                        super(Model, self).__init__()
                        self.input_size = 28*28
                        self.conv1 = nn.Conv2d(1, 32, 5)
                        self.conv2 = nn.Conv2d(32, 64, 5)
                        self.pool = nn.MaxPool2d(2,2)
                        self.fc1 = nn.Linear(10*10, 1024)
                        self.fc2 = nn.Linear(1024, 10)

        def forward(self, x):
                x = F.relu(self.conv1(x))
                x = F.relu(self.conv2(x))
                x = self.pool(x)
                x = x.view(1, -1)
                x = F.relu(self.fc1(x))
                x = self.fc2(x)
                return x

        model = Model()
        result = model(torch.autograd.Variable(torch.rand(1,1,28,28)))
        print(result)
```

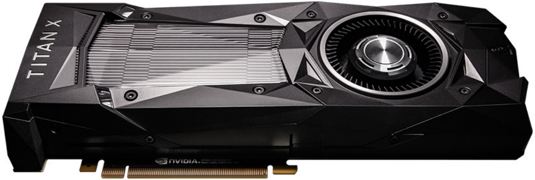
Training the network

```
epochs = 5
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)


for e in range(epochs):
        for i, (input, target) in enumerate(train_loader):
                input_var = torch.autograd.Variable(input)
                target_var = torch.autograd.Variable(target)

                output = model(input_var)
                loss = criterion(output, target_var)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

        # cross-validation, testing etc.
```

(GP)GPU computation in Machine Learning

- In principle multiple GPU vendors & software.
- In practice in ML almost exclusive application of Nvidia GPUs with CUDA.
- Is very useful because a large amount of operations in e.g. NNs are elementwise. Elementwise operations imply that the individual elements can be computed fully in parallel. E.g. convolutions, Hadamard products etc.
- Is particularly useful because one update in algorithms like SGD is typically based on a population of inputs. For these inputs (e.g. different images) the application of the complete pipeline can be calculated independently in parallel.
- Does not help with temporally correlated data or the necessary sequentiality of updates itself (relying on information of previous steps).
- Application typically limited by specific hardware constraints like memory limits.

GPU acceleration

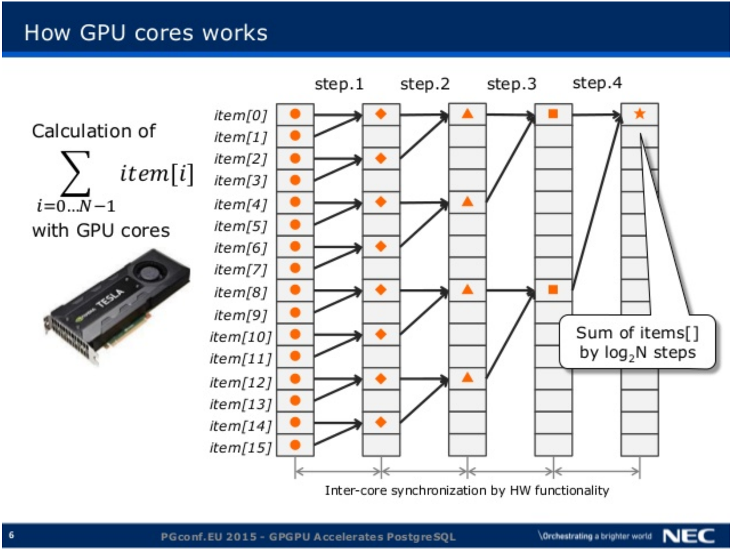


NVIDIA GP102-based Graphics Cards

VideoCardz.com	TITAN X "Pascal"	GeForce GTX 1080 Ti	NVIDIA TITAN Xp
GPU	GP102-400	GP102-350	GP102
CUDA Cores	3584	3584	3840
TMUs	224	224	240
Boost Clock	1531 MHz	1584MHz	1582 MHz
Computing Power	10.97 TFLOPs	11.34 TFLOPs	12.15 TFLOPs
Memory Clock	10.0 Gbps	11.0 Gbps	11.4 Gbps
Memory Capacity	12 GB	11 GB	12 GB
Memory Bus & Type	384-bit / GDDR5X	352-bit / GDDR5X	384-bit / GDDR5X
Memory Bandwidth	480 GB/s	484 GB/s	547.7 GB/s
MSRP	1200 USD	700 USD	1200 USD

nvidia.com; http://hitechgazette.com/nvidia-titan-xp-new-graphics-card-by-nvidia-to-beat-the-gtx-1080-ti/

A non-trivial example



https://www.slideshare.net/kaigai/gpgpu-accelerates-postgresql-unlock-the-power-of-multithousand-cores

GPU acceleration in PyTorch

- A Nvidia GPU with corresponding CUDA version needs to be installed
- CUDNN can be further used for even better acceleration

```
is_gpu = torch.cuda.is_available()

if is_gpu:
criterion = criterion.cuda()
        model = model.cuda()

for e in range(epochs):
        for i, (input, target) in enumerate(train_loader):
                input_var = torch.autograd.Variable(input)
                target_var = torch.autograd.Variable(target)

                if is_gpu:
                        input_var = input_var.cuda()
                        target_var = target_var.cuda()

                output = model(input_var)
                loss = criterion(output, target_var)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
```

PYTORCH