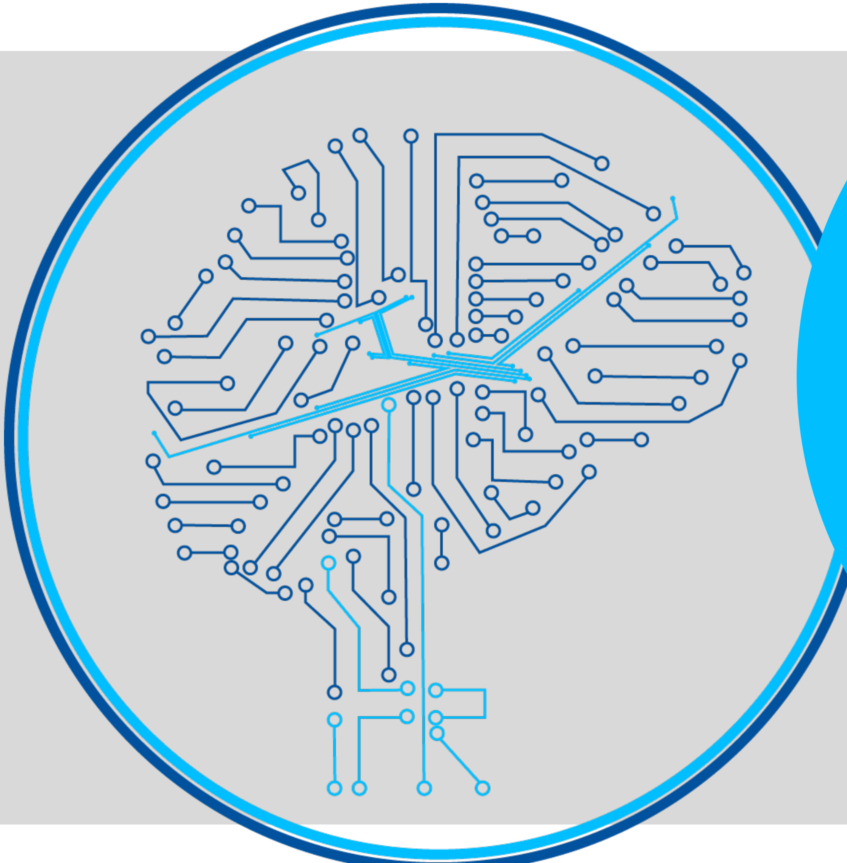
University of Stuttgart
Flight Mechanics and Controls Lab



Machine Learning Frameworks

Fabian Schimpf

Housekeeping

- The results of the student survey are in – there is going to be a separate video about it
- There is not going to be a separate assignment for this lecture but a **voluntary** assignment that lets you work on a real problem. The exercise will be uploaded early next week. You can team up one other participant and there will be a price for the best solution. Deadline for the submission will be April 30th .

Outline for the lecture

1. Motivation
2. Deep learning libraries
3. Setting up a ML project
4. Comments

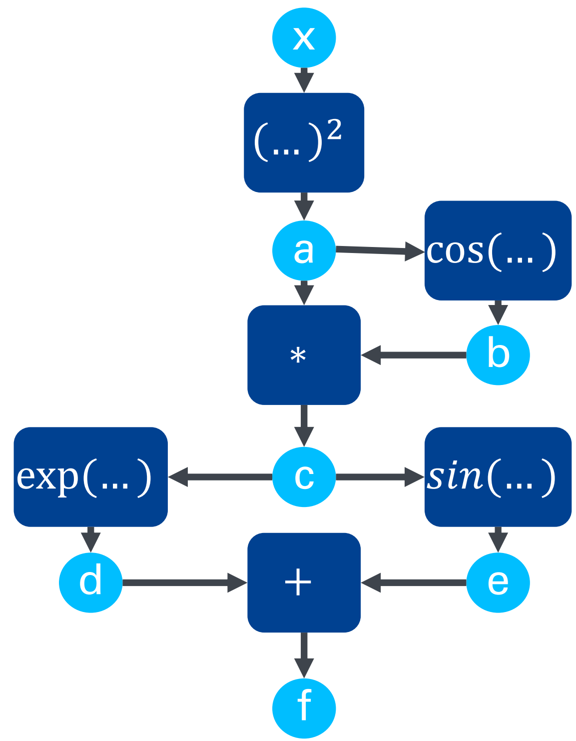
Motivation

Computational graphs

- Representation of a set of equations as a graph with nodes representing operations and variables
- Dependencies can be used to calculate derivatives

- Example:

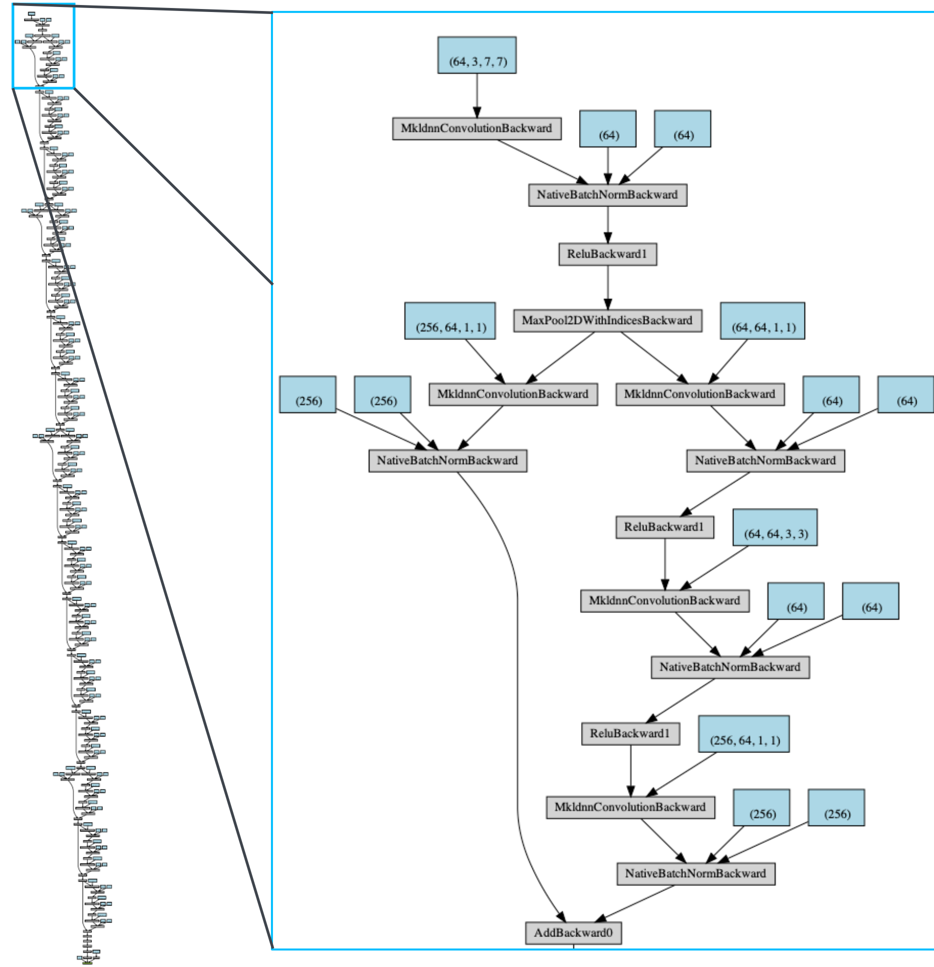
$$a = x^2$$
$$b = cos(a)$$
$$c = a \cdot b$$
$$d = exp(c)$$
$$e = sin(c)$$
$$f = d + e$$



Computational graphs

Example: Backward Pass ResNet50

- For larger networks (usually found in computer vision and language processing) computational graphs can fairly complex and large
- Manual implementations of these graphs and calculating the gradients can be prohibitively difficult



Why should you use a deep learning frameworks?

1. Easy language to build large computational graphs
2. Easy calculation of gradients on computational graphs
3. Easy usage of GPU(s)
4. Well tested codebase - fewer implementation errors
5. Community to help you with problems

Rough overview about computer architectures

- **CPU (Central Processing Unit):**
  Where programs are usually executed
- **GPU (Graphics Processing Unit):**
  Highly parallelized processing, each process by itself is usually slower than CPU. GPUs have their own RAM.
- **Memory Slots / RAM:**
  Memory that is available for computations with the CPU

Note:

- Computation can happen either in the CPU or GPU, but data and models have to be transferred there
- Moving samples to the GPU can result in a bottleneck

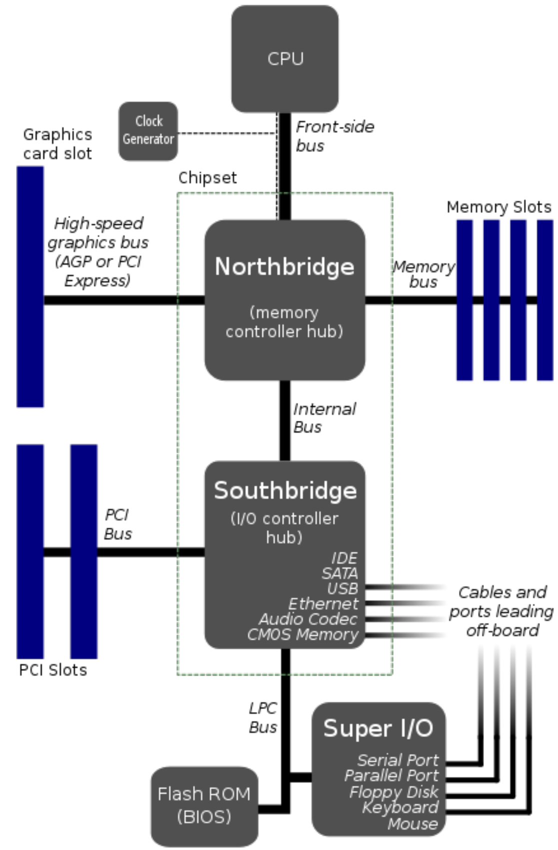


Image credit: https://de.wikipedia.org/wiki/Northbridge#/media/Datei:Motherboard_diagram.svg

Programming GPUs

Interpretation as a computation graph

- There are for the most part two frameworks
  - CUDA (proprietary by NVidia)
    - Supported by all major frameworks
  - OpenCL (hardware agnostic)
- Writing (efficient) functions for GPUs requires advanced programming skills
- ML Frameworks make let you run code on GPUs with a few additional lines of python code

C

```c
void c_hello(){
    printf("Hello World!\n");
}

int main() {
    c_hello();
    return 0;
}
```

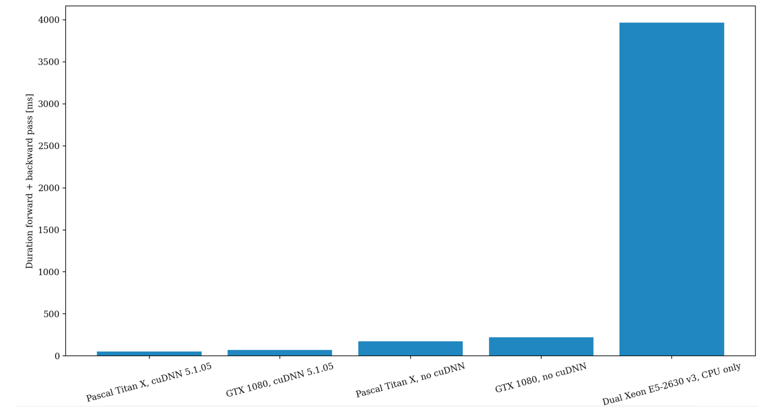
CUDA

```c
__global__ void cuda_hello(){
    printf("Hello World from GPU!\n");
}

int main() {
    cuda_hello<<<1,1>>>();
    return 0;
}
```

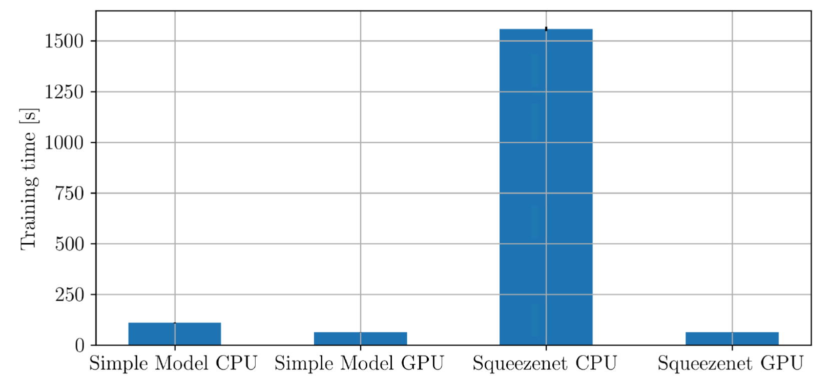
```c
#define N 10000000

void vector_add(float *out, float *a, float *b, int n) {
    for(int i = 0; i < n; i++){
        out[i] = a[i] + b[i];
    }
}

int main(){
    float *a, *b, *out;

    // Allocate memory
    a   = (float*)malloc(sizeof(float) * N);
    b   = (float*)malloc(sizeof(float) * N);
    out = (float*)malloc(sizeof(float) * N);

    // Initialize array
    for(int i = 0; i < N; i++){
        a[i] = 1.0f; b[i] = 2.0f;
    }

    // Main function
    vector_add(out, a, b, N);
}
```

GPUs

- **CUDA** is the “language” to program Nvidia’s graphics cards
- **CuDNN** is a library of functions that are optimized for functions used in neural networks
- **But:**
  Speed ups are more noticeable for larger networks, for smaller networks the overhead of copying data to the GPUs can equalize the speed ups. This effect becomes even more noticeable during the execution after training.



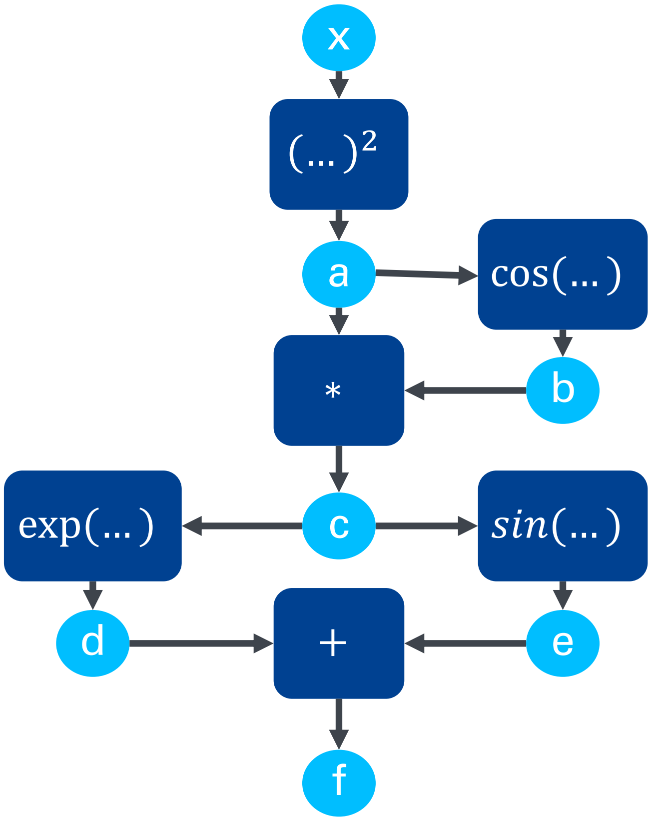
Data from: https://github.com/jcjohnson/cnn-benchmarks



Introduction to PyTorch

Building simple computation graphs

PyTorch




```python
def function(x):
    a = x * x
    b = torch.cos(a)
    c = a * b
    d = torch.exp(c)
    e = torch.sin(c)
    f = d + e
    return f.mean()

x = torch.ones(1, requires_grad=True)
f = function(x)
f.backward()
print(x.grad)
```

MLP in PyTorch

Defining the network

```
import torch.nn as nn
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(13, 20)
        self.fc2 = nn.Linear(20, 20)
        self.fc3 = nn.Linear(20, 1)

    def forward(self, x):
        x = torch.tanh(self.fc1(x))
        for ii in range(9):
            x = torch.tanh(self.fc2(x))
        x = self.fc3(x)
        return x
```

Adds a number of attributes and methods like .parameters() and .zero_grad()

Input size

Size of hidden layer

Output size: 1 as this network is for regression

- Define which layers and operations should be used

Number of hidden layers

- Define the forward pass: With which operations (partly defined above) is an input turned into an output

MLP in PyTorch

Defining the network

```
def mlp_pytorch(X, y):
    X = torch.Tensor(X)
    y = torch.Tensor(y)

    # Set up PyTorch
    model = Net()
    loss_fn = torch.nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.1)

    for ii in range(200):
        y_pred = model(X)
        loss = loss_fn(y_pred, y)
        loss.backward()
        optimizer.step()
        model.zero_grad()
```

X and y are numpy arrays

X and y can be transferred into objects (tensors) that PyTorch uses for its computation. Note, numpy arrays and tensors use the same memory in RAM and are merely different ways to interpret the zeros and ones.

Create an instance of the Net class from the last slide

Define a loss function, in this case MSE

Use Adam as optimizer on the model's parameters (from nn.Module) with a learning rate of 0.1

Make a prediction (forward pass of the model)

Calculate the loss

Calculate the gradients based on this loss

Update the parameters

Reset the gradients that they don't accumulate over epochs (from nn.Module), optimizer.zero_grad() would only set the parameters associated with the optimizer to zero.

Moving computations to the GPU

PyTorch

```
def mlp_pytorch(X, y):

    device = torch.device('cpu')
    if torch.cuda.is_available():
        device = torch.device('cuda')

    X = torch.Tensor(X).to(device)
    y = torch.Tensor(y).to(device)

    # Set up PyTorch
    model = Net().to(device)
    loss_fn = torch.nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.1)

    for ii in range(200):
        y_pred = model(X)
        loss = loss_fn(y_pred, y)
        loss.backward()
        optimizer.step()
        model.zero_grad()
```

- This example is for designed to use a single GPU – multi GPU usage is slightly different

Basically, there are two steps:

1. Figure out which device to use (GPU or CPU)
2. Move the model and the data to said device

Training with batches

WITHOUT DATALOADER

```
def mlp_pytorch_without_dataloader(X, y):
    X = torch.Tensor(X)
    y = torch.Tensor(y)

    # Set up PyTorch
    model = Net()
    loss_fn = torch.nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.1)

    n = X.shape[0]
    bs = 50

    for ii in range(200):
        for i in range(n // bs + 1):
            data = X[i * bs : i * bs + bs]
            target = y[i * bs : i * bs + bs]

            y_pred = model(data)

            loss = loss_fn(y_pred, target)
            loss.backward()
            optimizer.step()
            model.zero_grad()
```

WITH DATALOADER

```
def mlp_pytorch_with_dataloader(X, y):
    X = torch.Tensor(X)
    y = torch.Tensor(y)

    # Set up PyTorch
    model = Net()
    loss_fn = torch.nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.1)

    dataset = utils.TensorDataset(X, y)
    dataloader = utils.DataLoader(dataset, batch_size=256, shuffle=True)

    for ii in range(200):
        for data, target in dataloader:
            y_pred = model(data)
            loss = loss_fn(y_pred, target)
            loss.backward()
            optimizer.step()
            model.zero_grad()
```

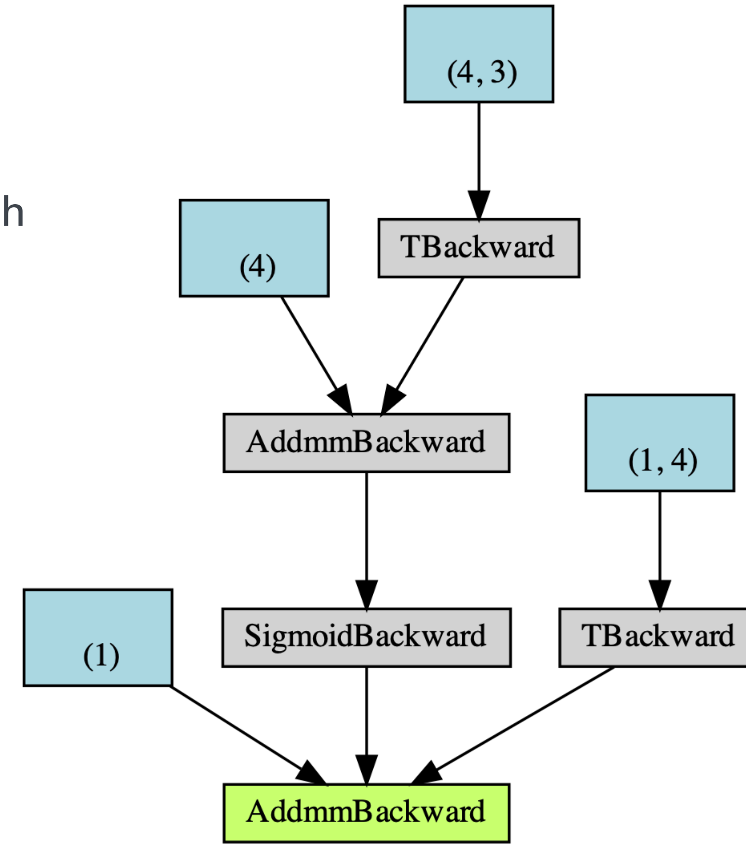
- Can allocate multiple workers to load and pre-fetch data
- Can include operations like shuffling, data augmentation / transformations

Multi layer perceptron

Computational graph for backward pass

- Note, for simple, sequential models PyTorch allows for simplified model definitions

```
Net = nn.Sequential(
    nn.Linear(3, 4),
    nn.Sigmoid(),
    nn.Linear(4, 1),
)
```



Setting up a ML project

Overview of the stages

- Frame the problem you try to solve
- Data acquisition
- Setup of an environment
- Explorative data analysis (EDA)
- Data cleaning / Preparation of dataset
- Training of applicable models:
  - Training of a baseline
  - Training of a more complex model
  - Hyperparameter optimization
- Presentation
- Deployment / Monitoring / Maintenance

Problem specific

What we are going to discuss here

Covered in previous lectures

Assumption:
Our data makes sense

Out of scope:
Look into DevOps if you're curious

Data acquisition

Potential sources:

- Sensors
- Getting data from humans (Surveys)
- Simulations (Decision making problems, Robotics)

Problems that can arise here:

- Noise in the measurements / subjective impressions
- Different sources
  - No standardized representation / Missing data / NaN entries
- Data privacy requirements (GDPR)
- False labels (Jaguar is a brand and a cat)

Key Issues

Consent | Data Protection Officer | Email Marketing | Encryption | Fines / Penalties | Personal Data | Privacy by Design | Privacy Impact Assessment | Processing | Records of Processing Activities | Right of Access | Right to be Forgotten | Right to be Informed | Third Countries

Image credit: https://gdpr-info.eu

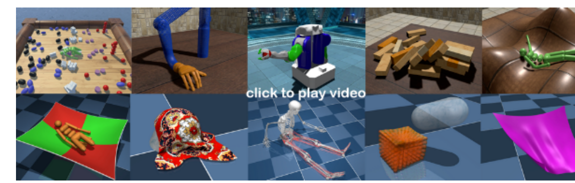


Image credit: Mujoco.org

```python
def clean_up(x):
    x = str(x)
    x = x.replace("+", "")
    x = x.replace("over", "")
    x = x.replace("plus", "")
    x = x.replace("more than", "")
    x = x.replace(",", "")
    x = x.replace("\'S (15 YEARS)", "")
    x = x.replace("200-300", "250")
    x = x.replace("15-30", "22")
    x = x.replace("\'s", "")
    x = x.replace("s", "")
    x = x.replace("Over", "")
    x = x.replace("Hundreds", "100")
    x = x.replace("hundreds", "100")
    x = x.replace("Hundred", "100")
    x = x.replace("hundred", "100")
```

Cleaning survey data

General software tools

Note: Doing this justice requires a lecture by itself – see further reading

- **Development environment**
  - Jupyter Notebooks / Jupyter Lab
  - IDEs like PyCharm or Spyder → Debugger
- **Version management**
  - Git, SVN, …
- **Creating a reproducible environment**
  - Virtual env / Conda for Python→ Manage the python environment
  - Docker or VMs for your system ("It works on my machine")
- **Testing**

Exploratory data analysis (EDA)

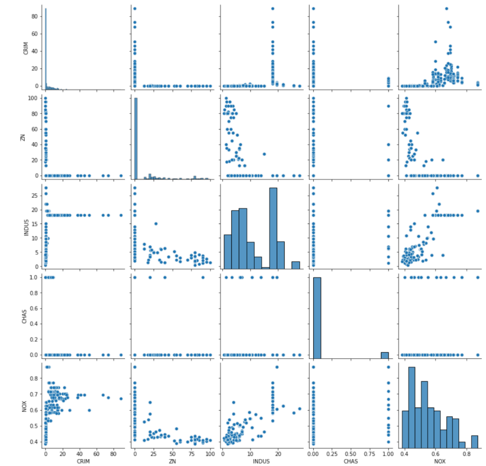
Objective:

- Explore the data → Understand the problem at hand
- The goal is explicitly **NOT** to make predictions

Main tools:

- Visualizations
  - Scatter plots (potentially w. dimensionality reduction)
  - Histograms
- Summary statistics
  - Correlation
  - Mean / Median / Variance
  - …

➡ **Insights can then be used to clean the data and to propose reasonable models in the next steps**



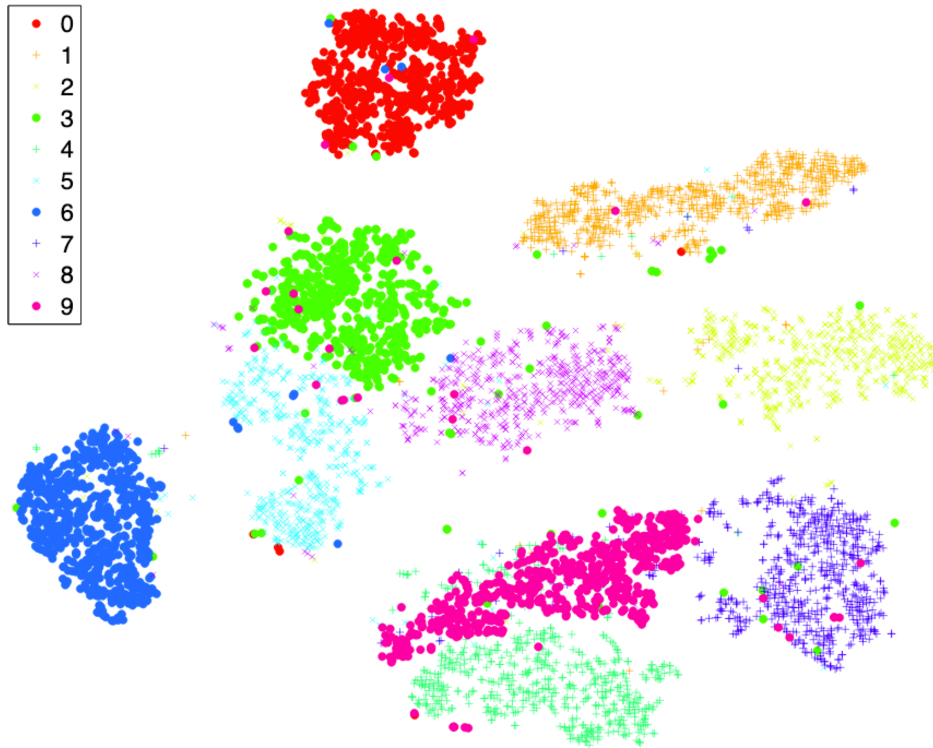
```
4  from sklearn.datasets import load_boston
   import pandas as pd
```

```
5  boston_dataset = load_boston()
   boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)

   boston.describe()
```

5

	CRIM	ZN	INDUS	CHAS	NOX	RM
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000

Outlier detection with dimensionality reduction

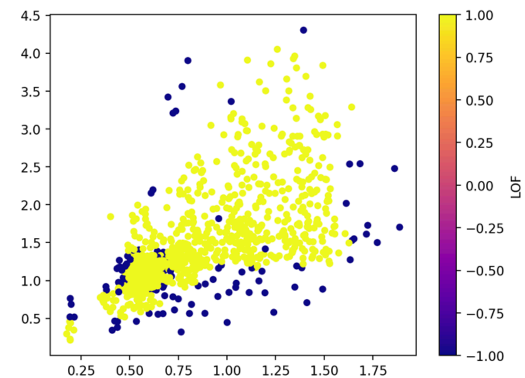


Van der Maaten, Hinton: “Visualizing Data using t-SNE”

Data cleaning / Preparation of dataset

- Handle missing data
  - Drop feature or sample / Fill it with heuristic
- Outlier detection / Attention checks
- Filtering of data: Drop samples or features based on EDA
- Feature engineering – combine attributes; Add nonlinearity
- Handle categorical inputs / Find a suitable encoding
- Feature scaling and normalization (lecture optimization II)
- Handling custom user input for quantitative analysis

Attention Check:
How many moons circle earth:
a) 1
b) 2
c) More than 5



Attempt to classify outliers
Local Outlier Factor (LOF)

Comments

Static vs. dynamic computation graphs

DEFINE BY RUN / DYNAMIC

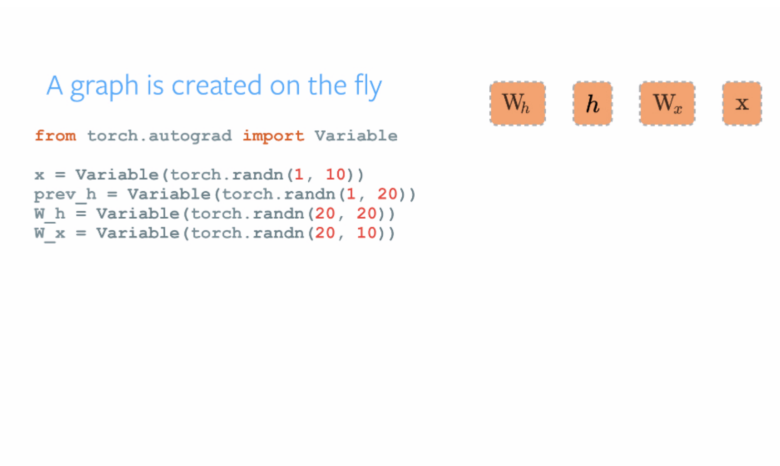


Image credit: PyTorch

DEFINE AND RUN / STATIC

- First step: Define computational graph abstractly (size and type of variables and whether they need a gradient computation or not)
- Second step: Run computational graph with given parameters and inputs
- Allows for more optimized computation, therefore usually (slightly) faster

Tensorflow vs. PyTorch

Everything said here might be outdated soon !!!

TENSORFLOW

- Developed by Google
- Has been around for a while – very mature and stable
- Traditionally, static computation graphs
- Often used for large scale deployment
- High adoption in the industry
- Harder to debug therefore potentially steeper learning curve

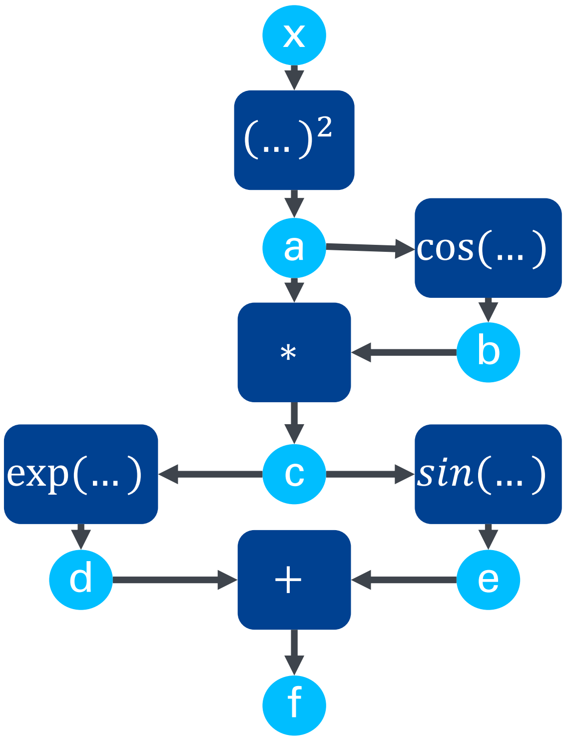
PYTORCH

- Developed by Facebook
- Traditionally, dynamic computation graphs
- Fast growing in research
- Might lack some functionalities that Tensorflow provides out of the box
- Tough / impossible to use 2^{nd} order derivatives
- Easier to debug

There is no wrong choice here!

Up and coming framework: JAX

JAX is basically aspires to be Numpy on steroids



- Use accelerators like GPUs and TPUs
- JIT compile functions for faster execution
- Nice handling of derivatives (below)

```python
import jax.numpy as np
from jax import grad

def function(x):
    a = x * x
    b = np.cos(a)
    c = a * b
    d = np.exp(c)
    e = np.sin(c)
    f = d + e
    return f

first_derivative    = grad(function)
second_derivative   = grad(grad(function))
third_derivative    = grad(grad(grad(function)))
```

What makes ML tough?

- **Different skills**
  - Software Engineering
  - Domain Knowledge
  - Mathematics / Statistics / Optimization
  - Presentation
- **Multitude of failure cases related to each skill**
  - Hard to debug / find root cause of problems
- **Dependance on data of unknown quality**
- **No cookie cutter solution to many new problems**

Further reading

Lectures

- Effizientes Programmieren I und II
- Mustererkennung und Optimierung
- Nichtlineare Optimierung

Textbooks

- Aurélien Géron: “Hands-On Machine Learning with Scikit-Learn and Tensorflow”
- Deisenroth, Faisal, Ong: ”Mathematics for Machine Learning”
- Russell, Norvig: “Artificial Intelligence a Modern Approach” (currently 4th edition)

University of Stuttgart
Flight Mechanics and Controls Lab

Thank you!

Fabian Schimpf

e-mail Fabian.schimpf@ifr.uni-stuttgart.de
phone +49 (0) 711 685-
fax +49 (0) 711 685-

University of Stuttgart
Flight Mechanics and Controls Lab
Pfaffenwaldring 27, 70569 Stuttgart