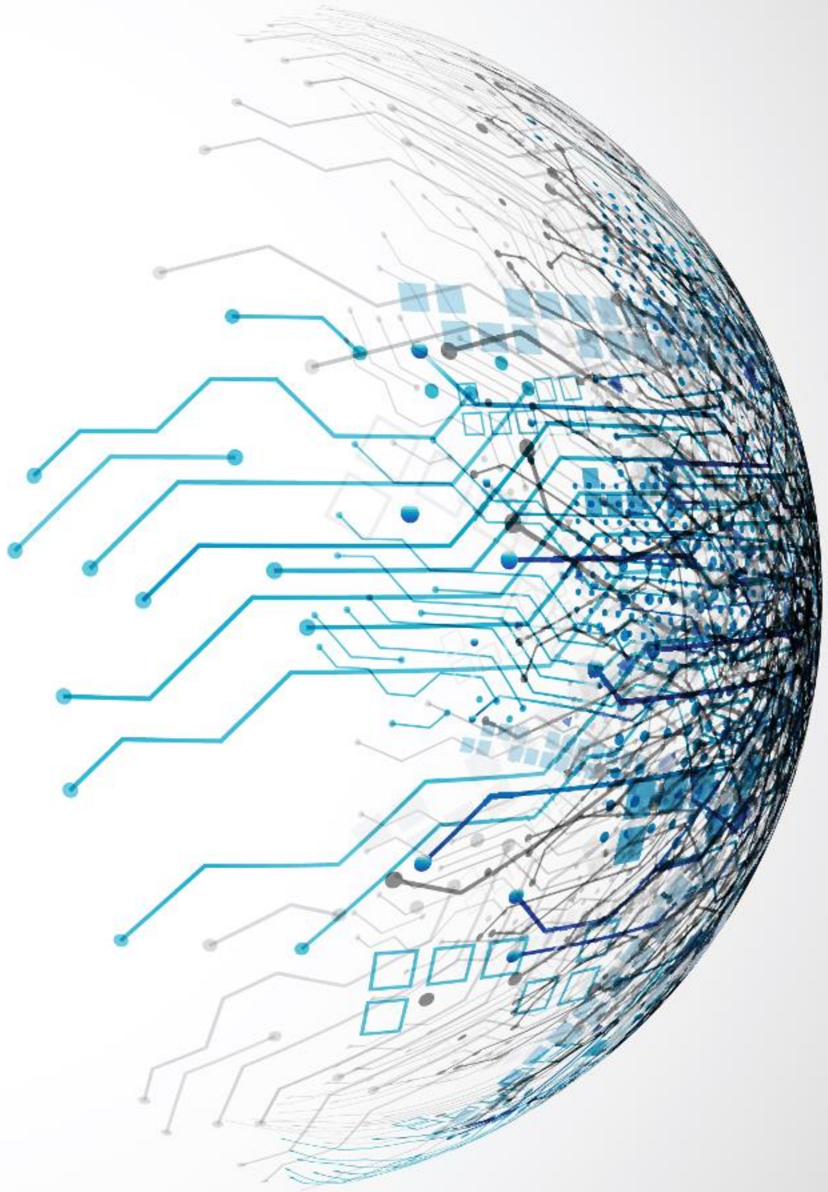
Deep Learning

Frameworks, PyTorch basics



Dr. Mohammed Salah Al-Radhi

(slides by: Dr. Bálint Gyires-Tóth)

Copyright

References

https://bit.ly/3AFIKuT

Microsoft

Deep Learning - BMEVITMMA19-EN | General | Microsoft Teams

Announcements

Project work

- Group and topic selection done

Milestone 1: data acquisition, data preparation (+ optional: containerization)

- Deadline: 7th week, **Oct 15**, Tuesday, 23:59, moodle, GitHub repo
- Oct 16, Wednesday class: consultation about projects (required for each group)

Milestone 2: baseline evaluation, baseline model

- Deadline: 11th week, **Nov 12**, Tuesday, 23:59, moodle, GitHub repo
- Nov 13, Wednesday class: consultation about projects (required for each group)

Final submission

- Deadline: end of 14th week, **Dec 6**, Friday, 23:59, moodle, GitHub repo and documentation

Outline

- DL frameworks
- PyTorch
  - Tensors
  - Computational path
  - AutoGrad
- PyTorch Lightning

DL Frameworks

DL frameworks

- Tensorflow
- Tf.keras
- PyTorch
- PyTorch Lightning
- Lightning / Fabric
- Lightning / Bolts

TensorFlow, tf.keras

- Deep Learning in practice based on Python and LUA / 2023

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

model = Sequential()
model.add(Dense(256, activation='relu', input_shape=(784,)))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer=SGD(lr=0.001),
              metrics=['accuracy'])
```

PyTorch basic

PyTorch Introduction

Predecessor: Torch (last version Torch7)

- LUA programming language
- From 2002 to 2018
- General tensor operations with torch.nn package
- Staff of Facebook, Twitter, DeepMind, Nvidia, Idiap, NYU, Yandex, etc.
- Poor data preprocessing and inference ecosystem
- http://torch.ch/

Torch & LUA

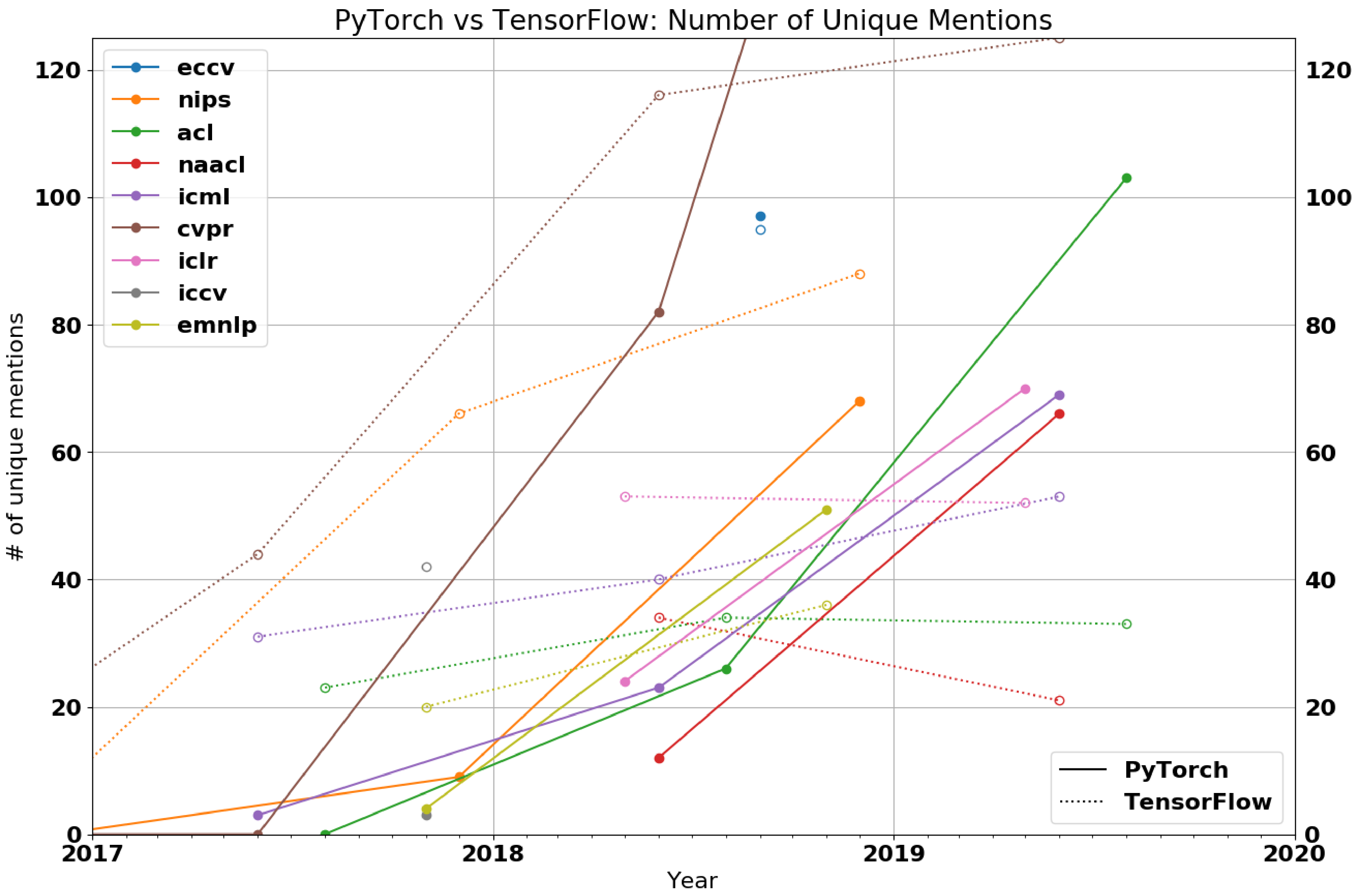
What we said in 2016? (on Hungarian course)

- „… one of the most widespread Deep Learning framework…"
- „in Torch7 can find the **Tensor** (= matrix = array) class, which is similar to a numpy array."
  - `z = torch.Tensor(4,5,6,2)`
- „moving tensors to GPU and from GPU"

PyTorch Introduction

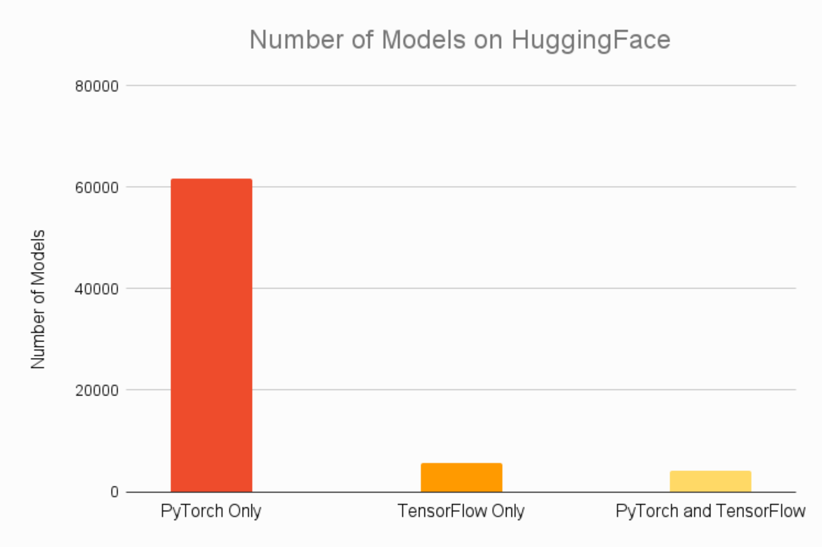
PyTorch

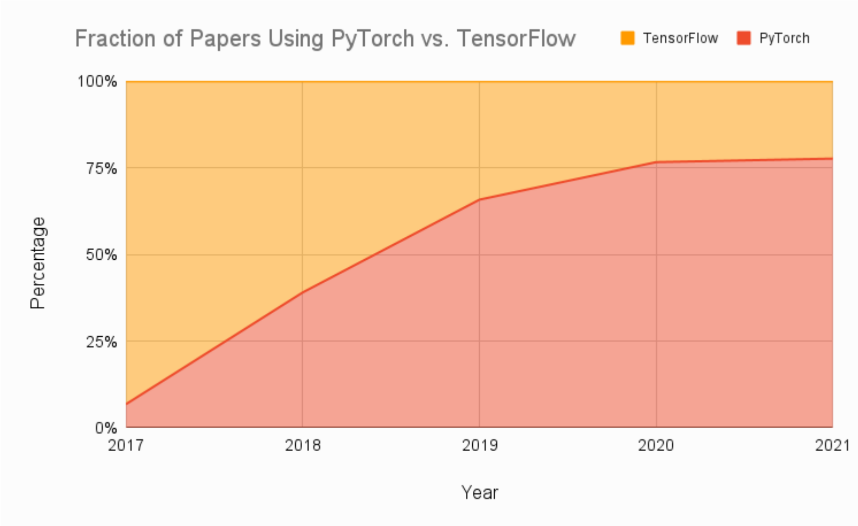
- Intial release Sept. 2016, latest stable release (2.0): March 2023
- Tensor computations on GPUs
- Dynamic computational graph
- Automatic differentiation
- torch and torch.nn main packages
- Complete Python ecosystem
- ONNX support
- https://pytorch.org/

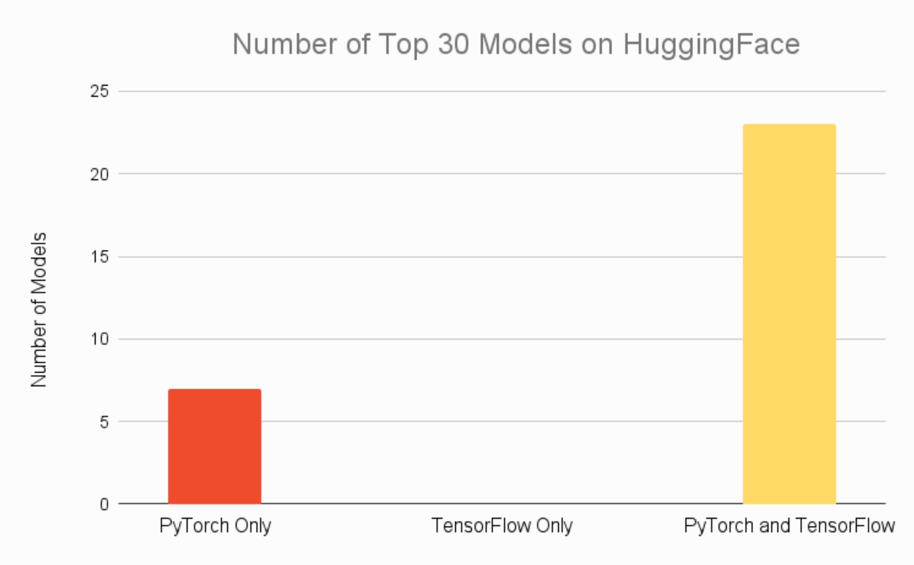


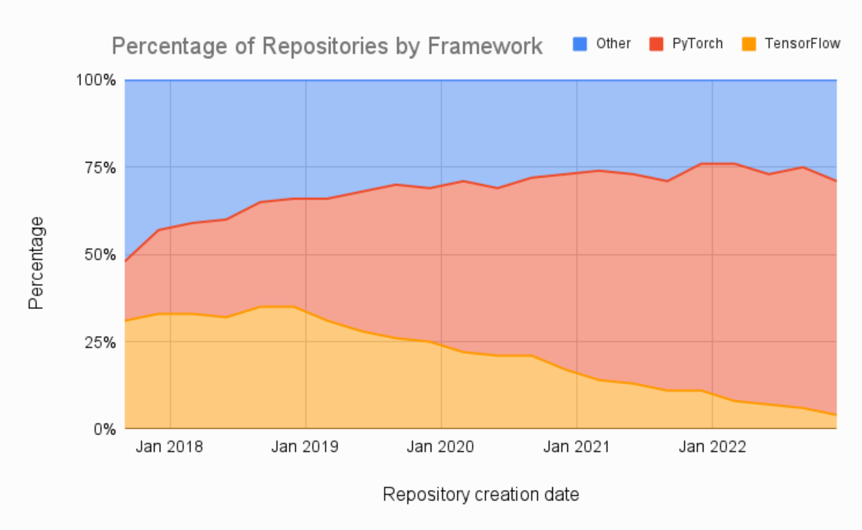
Forrás: https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/

PyTorch vs TensorFlow 2023







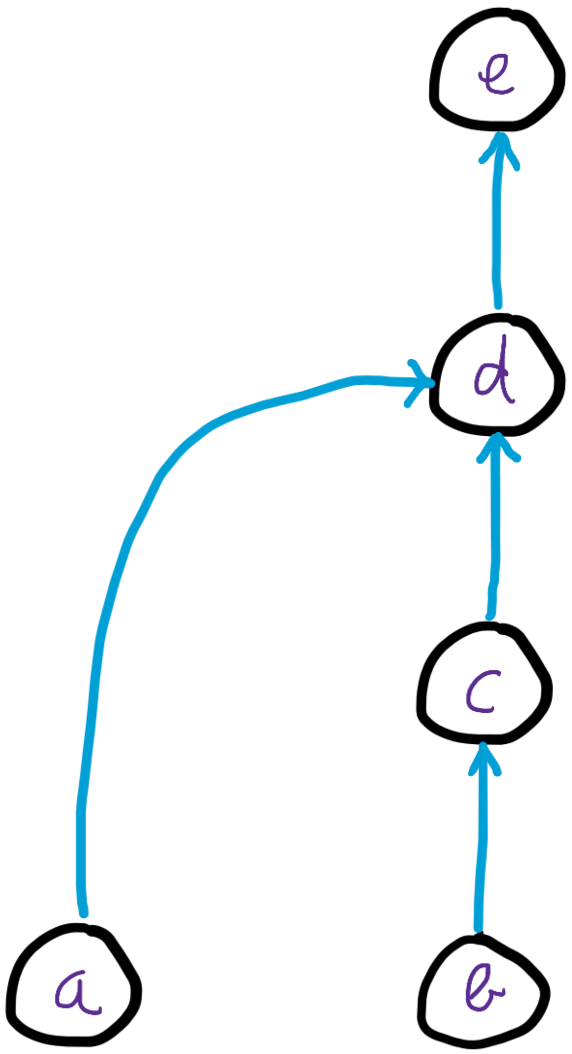


Source: https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/

Computational graph

- Directed Acyclic Graph (DAG)
  - Nodes:
    - Leaf nodes: variables (e.g. tensors, matrix, vectors, scalars)
    - Non-leaf nodes: results of operations
  - Edges: operations

```
import torch
a = torch.rand(1, 4, requires_grad=True)
b = torch.rand(1, 4, requires_grad=True)
c = b*2
d = a+c
e = d.sum()
```



Dynamic computational graphs

Like dynamic memory allocation: we don't know how much memory we will need

In each iteration:

1. The graph is created upon calling `.forward`
2. Gradients are calculated upon calling `.backward`
   2.1. Graph is freed if `retain_graph=False` (default)
   2.2. Leaf-nodes stay in the memory
3. Weights are updated `.step`

Practical advantage:

- faster startup
- can handle varing input sizes (eg. in NLP, CV)
- more possibilities for research purposes

PyTorch modules

torch	Base module.
torch.nn	Neural network module, defines nn.Module classes.
torch.nn.functional	Stateless neural network functions. (functions only)
torch.nn.init	Parameter initialization module.
torch.autograd	Automatic differentiation of arbitrary scalar valued function.
torch.cuda	GPU related module.
torch.distributed	PyTorch distributet module. Supports Gloo (~distributed CPU) and NCCL (~distributed GPU). MPI if built from source.
torch.distributions	Parameterizable probability distribution and sampling functions. For stachastic computational graphs and gradient estimators.
torch.hub	Pretrained model repository.
torch.onnx	ONNX support.
torch.optim	Optimization package.
torch.random	Random generator, setting random seed.
torch.jit	TorchScript support to serialize PyTorch code to non-Python environment.
torch.sparse	Sparse matrix calculations (beta)
torch.utils	bottleneck, checkpoint, cpp_extension, data, dlpack, mobile_optimizer, model_zoo, tensorboard

Tensors

- Single or multidimensional matrices
- Use torch.tensor
- Can be: 2/8/16/32/64/128 bit bool/integer/float/complex (not all apply)
  - Defined by `dtype=torch.<TYPE>` or `<TENSOR>.type(torch.<TYPE>)`
- Most important functions:
  - `.cuda(), .to(), cpu(), .get_device()`
  - `.as_tensor(), numpy(), .item(), .tolist()`
  - `.type(), .to()`
  - `(…, requires_grad=True), .requires_grad_(), .detach()`
  - `.backward(), .grad`
  - `.view(), .view_as(`*`<other_tensor>`*`), .expand(), .squeeze(), .unsqueeze()`
  - `.repeat(), resize_(), .cat`
  - `.clone()`

Documentation: https://pytorch.org/docs/stable/tensors.html

PyTorch important syntax rules

Small and capital letters

- .relu() → torch.nn.functional
- .ReLU() → torch.nn.module
- torch.tensor() → creates a tensor, has dtype attribute (*recommended*)
- Torch.Tensor() → creates a Tensor class, might have large overhead

Inplace operations

- relu_() → inplace
- relu() → result in the ouptut

GPU

GPU

- Tensors of a computational graph must be on the same device.
- Check GPU
  - `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")`
  - `torch.cuda.device_count()`
  - `torch.cuda.get_device_name()`
- Multi-GPU setup
  - `torch.cuda.manual_seed_all(123)`
  - `device = torch.device("cuda:0")`
  - `export CUDA_VISIBLE_DEVICES=1,2`
- To and from GPU
  - `.cuda(), .to()`
  - `.cpu(), .to()`

nn.Module

nn.Module

- Base class for neural network modules
- ~ packs single or multiple layers
- Custom modules can be built
- Multiple modules can be packed in `nn.Sequential({…})`
- Works with autograd, i.e. has parameters and gradients
- Important functions:
  - `forward(), backward()`
  - `zero_grad()`
  - `train(), eval()`
  - `save_state_dict(), load_state_dict()`
  - `modules(), parameters()` → `iterators`
  - `register_forward_hook, register_backward_hook`

Custom nn.Module

```
class MyLayer(nn.Module):
    def __init__(self, neurons=32, outputs=10):
        super(MyLayer, self).__init__()
        self.fc1 = nn.Linear(28*28, neurons)
        self.fc2 = nn.Linear(neurons, outputs)
        self.reset_parameters()

    def reset_parameters(self):
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.xavier_normal_(m.weight)
                #nn.init.kaiming_uniform_(m.weight, mode='fan_in',
                                          nonlinearity='relu')

    def forward(self, data):
        x = data.view(-1, 28*28)
        x = self.fc1(x)
        x = torch.relu(x)
        x = self.fc2(x)
        x = torch.log_softmax(x)
        return x
```

Symmetric activation

Asymmetric activation

e.g. data shape is (batch,28,28)

Saving and loading nn.Module

Parameters only:

```
torch.save(model.state_dict(), PATH)

model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
```

Parameters and model architecture:

```
torch.save(model, PATH)

model = torch.load(PATH)
model.eval()
```

More save and load methods: https://pytorch.org/tutorials/beginner/saving_loading_models.html#what-is-a-state-dict

Loss functions

Loss functions - part of torch.nn

Regression

- `nn.MSELoss` – Mean Squarred Error, activation function must match the range of the target data

Classification

- `nn.BCELoss` – Binary Cross Entropy loss, nn.Sigmoid activation function
- `nn.BCEWithLogitsLoss` – Binary Cross Entropy loss, linear activation function
- `nn.NLLLoss` – Negative log likelihood, multiple output, nn.LogsoftMax activation, dense layers.
- `nn.CrossEntropyLoss` – multiple output, combines nn.LogSoftMax() and nn.NLLLoss(), linear activation, dense layers.

More loss functions:
https://pytorch.org/docs/stable/nn.html#loss-functions

Optimizers

Optimizers – torch.optim

$$\Delta w^{(i)}(t) = -\mu \frac{\partial C}{\partial w^{(i)}(t)}.$$

$$w^{(i)}(t+1) = w^{(i)}(t) + \Delta w^{(i)}(t)$$

Optimizers – torch.optim

General usage:

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
for input, target in dataset:
    optimizer.zero_grad()
    output = model(input)
    loss = loss_fn(output, target)
    loss.backward()
    optimizer.step()
```

$$\Delta w^{(i)}(t) = -\mu \frac{\partial C}{\partial w^{(i)}(t)}$$

$$w^{(i)}(t+1) = w^{(i)}(t) + \Delta w^{(i)}(t)$$

If model runs on GPU, move it to GPU before constructing its optimizers.

Optimizers – torch.optim

Multiple optimizers:

```
Optimizer = optim.SGD([
                {'params': model.base.parameters()},
                {'params': model.classifier.parameters(), 'lr': 1e-3}
            ], lr=1e-2, momentum=0.9)
```

Common optimizers: SGD, RMSProp, Adam, AdamW

Optimizer functions:
https://pytorch.org/docs/stable/optim.html

Autograd

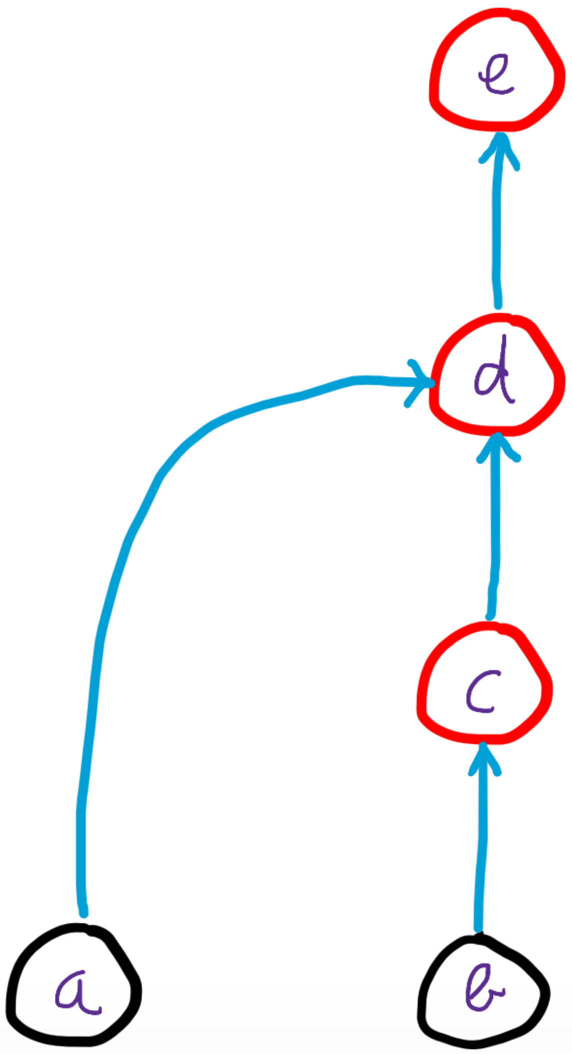
PyTorch autograd

- Everytime gradients are calculated, a backward calculation graph is constructed
- Functions of the backward computation graphs are defined in PyTorch
- Upon calculating the gradinets by .backward() the backward computation graph is freed (if `retain_graph=False`)

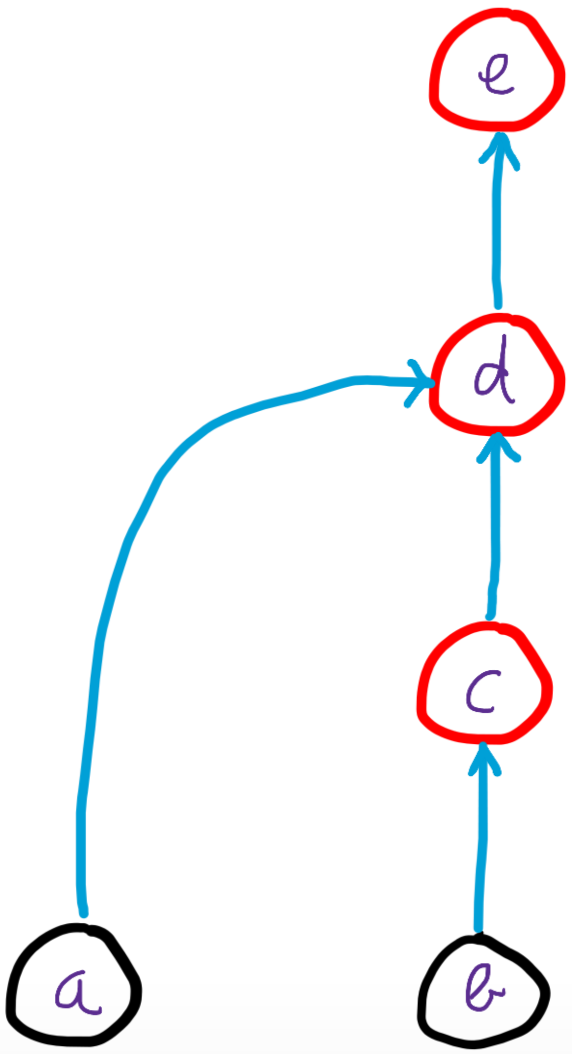
```python
import torch
a = torch.rand(16, 4, requires_grad=True)
b = torch.rand(16, 4, requires_grad=True)
c = b*2
d = a+c
e = d.sum()
```

– Leaf nodes, grad_fn=None

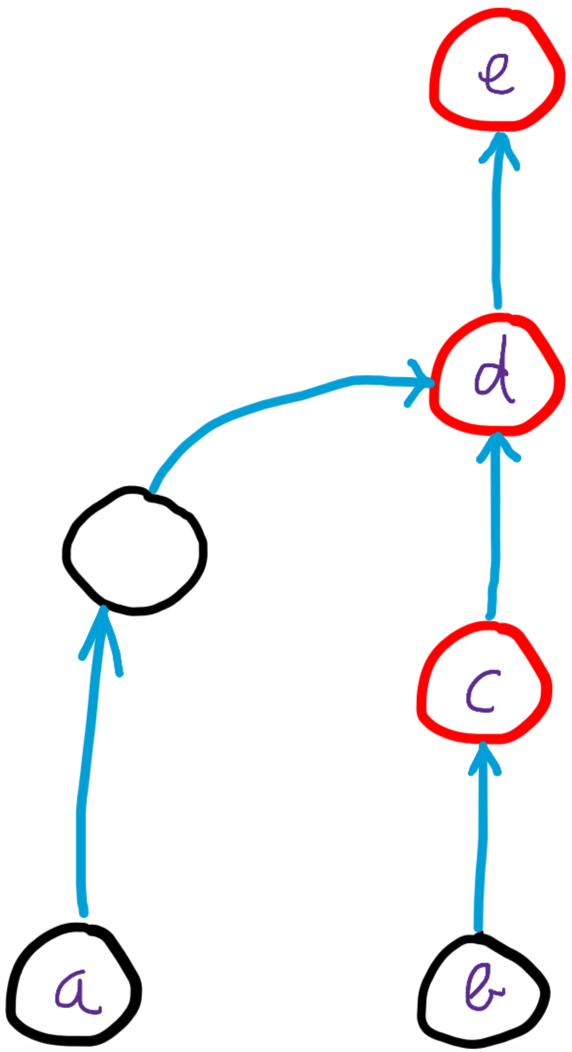
– Non-leaf nodes, have grad_fn



```python
import torch
a = torch.rand(16, 4, requires_grad=True)
b = torch.rand(16, 4, requires_grad=True)
c = b*2
d = a+c
e = d.sum()

loss = (10-e).sum()
loss.backward()


print(a.grad)
print(b.grad)
print(a.grad_fn)
print(e.grad_fn)
```



```python
import torch
a = torch.rand(16, 4, requires_grad=True)
b = torch.rand(16, 4, requires_grad=True)
c = b*2
d = a**2+c
e = d.sum()

loss = (10-e).sum()
loss.backward()


print(a.grad)
print(b.grad)
print(a.grad_fn)
print(e.grad_fn)
```



Basic autograd functions

- Add gradient

```
torch.Tensor(...requires_grad=True)
.requires_grad_(…)
```

- Detach from computational graph

```
.detach()
```

- Don’t calculate gradients

```
with torch.nograd():
```

Computing gradients

When the computation graph is ready

```
.backward()
```

calculates the gradient, which can be accessed with

```
.grad
```

The gradient function can be inspected by:

```
.grad_fn
```

Autograd function

Autograd function

We can create autograd function by defining `forward` and `backward` in a `torch.autograd.Function` class.

```
from torch.autograd import Function
class Custom(Function):
        @staticmethod
        def forward(ctx, inp):
        ...
        return result

        @staticmethod
        def backward(ctx, grad_output):
        ...
        return grad_result
```

`ctx`	Context, that stores tensors and can be retrieved during the backward pass.
`grad_output`	the gradient w.r.t the given output
`grad_result`	the gradient w.r.t. the corresponding input.

`inp` and `grad_out` should have the same shape

PyTorch Lightning

PyTorch Lightning / 1

PyTorch Lightning

- High-level DL framework (vs PyTorch: more low-level)
- Scaling ML/DL models to run on any hardware (CPU, GPUs, TPUs) without changing the model.
- Standardized steps, less boilerplate code
- Code more compact and clean

PyTorch vs PyTorch Lightning

- https://towardsdatascience.com/from-pytorch-to-pytorch-lightning-a-gentle-introduction-b371b7caaf09

PyTorch Lightning vs Lightning

- in March 2023, it was renamed to Lightning
- Company behind: [Lightning AI]
- But: many of the documentations still refer to PyTorch Lightning!

- Currently both work
  - `import pytorch_lightning as pl`
  - `import lightning as L`

- https://github.com/Lightning-AI/lightning/discussions/16688

Lightning variants

- Lightning Fabric, https://lightning.ai/docs/fabric/stable/
- Lightning Bolts, https://lightning-bolts.readthedocs.io/en/latest/

PyTorch mobile

- https://pytorch.org/mobile/home/

ONNX

ONNX: Open Neural Network eXchange

Standardized neural network model format, supports multiple frameworks for interopability.

- Caffe, Caffe2, Pytorch
- TensorFlow, Keras
- Chainer Microsoft CNTK, Apple CoreML, Apache MXNet
- MatLab
- SkLearn

Model zoo: https://github.com/onnx/models

References

- https://www.tensorflow.org
- https://pytorch.org
- https://lightning.ai

Please, don't forget to send feedback:

https://bit.ly/bme-dl

Thank you for your attention

Dr. Mohammed Salah Al-Radhi
malradhi@tmit.bme.hu

(slides by: Dr. Bálint Gyires-Tóth)

24 September 2024