Visual AI

CPSC 533R – 2020/2021

Lecture 2. Deep learning basics and best practices

Helge Rhodin

Queer Coded UBC

Queer Coded is an affinity group at UBC aiming to create safe spaces for the LGBTQ+ community. During these stressful times, many queer & trans students may be disconnected from supportive environments.

We're here for you!

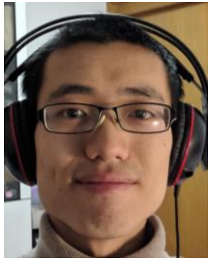
- Join our weekly online office hours (chat with peers, get support if needed)
- Attend our social & professional development events throughout the year
- Get access to other resources
- Connect with us:
  - Facebook: https://www.facebook.com/QCUBC
  - Slack: https://tinyurl.com/QCUBCslack
  - Email: queercodedubc@gmail.com

Organization

Instructor:
Helge Rhodin
rhodin@cs.ubc.ca

Office hours:
Tuesday 5 pm - 6 pm
Room: Zoom (via Canvas)

Teaching assistant:
Yuchi Zhang
yuchi45@cs.ubc.ca



Office hours:
Friday 3 pm - 4 pm
Room: Zoom (via Canvas)

Course Website

Curriculum	https://www.cs.ubc.ca/~rhodin/2020_2021_CPSC_533R/
Forum	https://piazza.com/ubc.ca/winterterm12020/cpsc533R
Canvas	https://canvas.ubc.ca/courses/53581

Lecture Overview

- Literature & research
- Compute resources
- Machine learning components in PyTorch
  - Interactive
- Best practices
  - Optimization
  - Loss functions
  - Training and evaluation
- Course project introduction

Literature & research

The Deep Learning Book

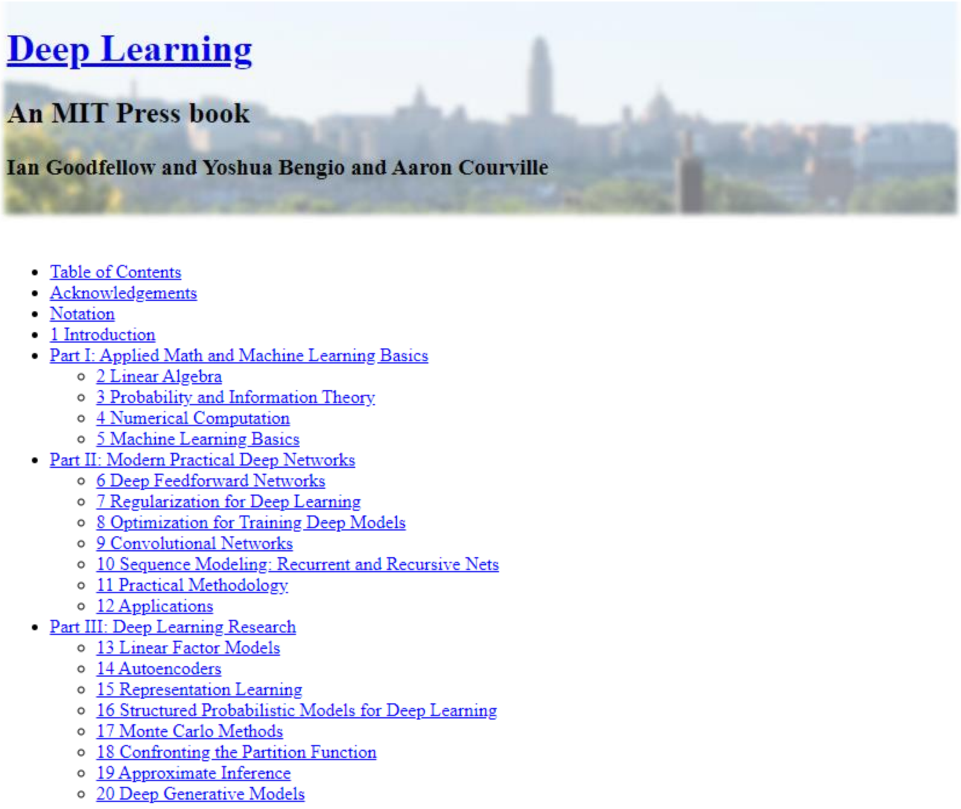
- links to relevant sections on the schedule web page

Online tutorials on PyTorch

- links on the assignment & Piazza

Research papers

- those we read as well as related work


Deep Learning

An MIT Press book

Ian Goodfellow and Yoshua Bengio and Aaron Courville

Recap: Deep learning – a new way of programming

Classical programming

- Write down computational rules

  c = a + b

- Requires human programmer

  (domain expert + CS skills)

https://futurism.com/2-whats-the-next-blue-collar-job-coding

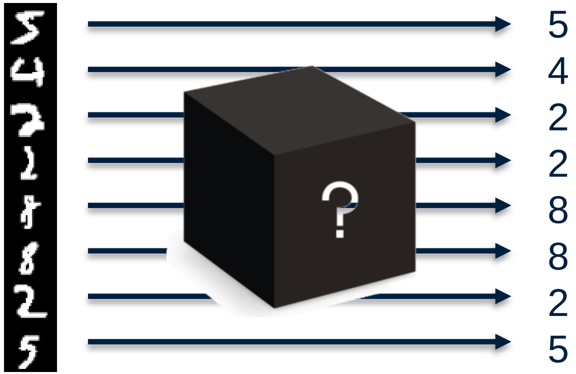
Data driven approach

- Give **lots of** input-output examples

  [(3,4) -> 7, (2,3) -> 5, (100,2) -> 102,

  (2,2) -> 4, (4,3) -> 7, …]

- Requires human annotator (domain expert)
- or Artificial Intelligence (AI) ?
  - Weak supervision, Self-supervision
  - Reinforcement learning …



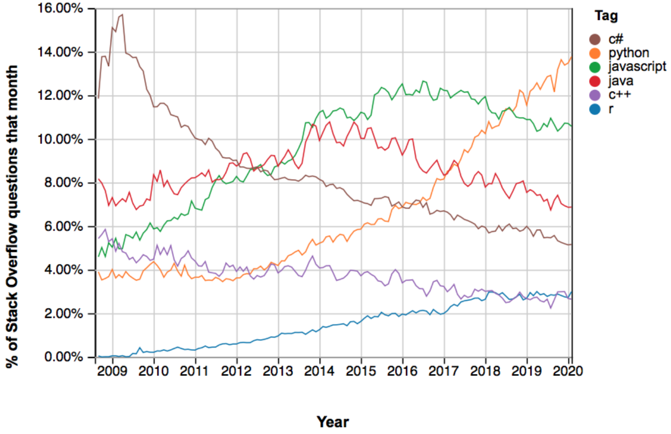
Programming environment - python™

Advantages

- high productivity / quick prototyping
- extensive support libraries
- high performance

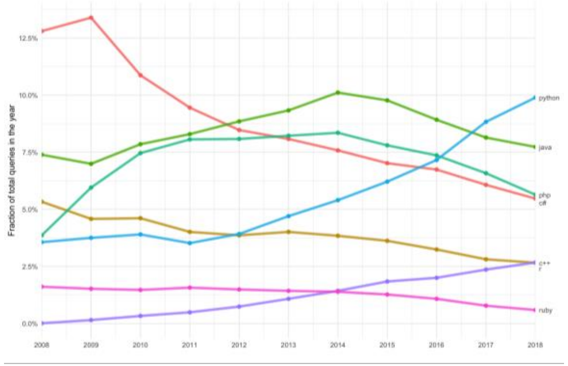
  (with libraries linked to programs compiled from FORTRAN, C++, Cuda, …)

- we will use python 3!



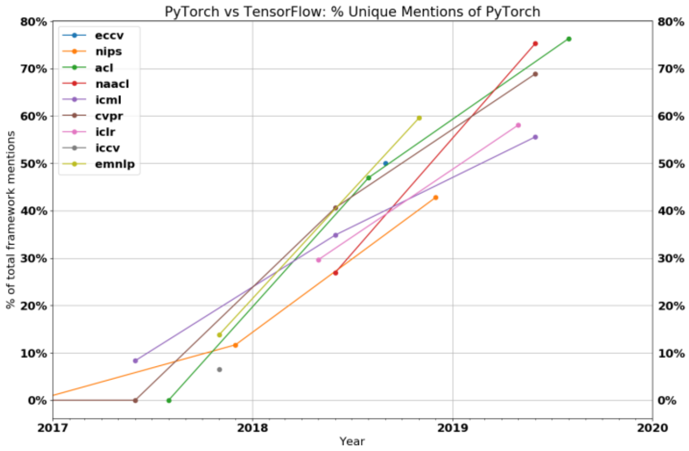
https://medium.com/@adithraghavs/python-is-getting-dethroned-37240d1c8ba3

Questions per year in Stack Overflow



https://towardsdatascience.com/predicting-the-future-popularity-of-programming-languages-4f28c80bd36f

Machine learning framework - PYTORCH



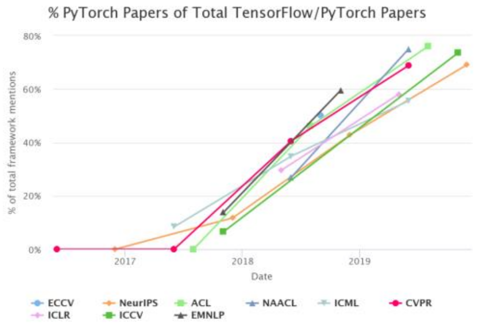
https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/

Features

- efficient matrix and tensor operations (like NumPy)
- automatic differentiation (dynamic)
- large number of tutorials
- many open source repositories

Resources

- PyTorch tutorials https://pytorch.org/tutorials/
- PyTorch introduction https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

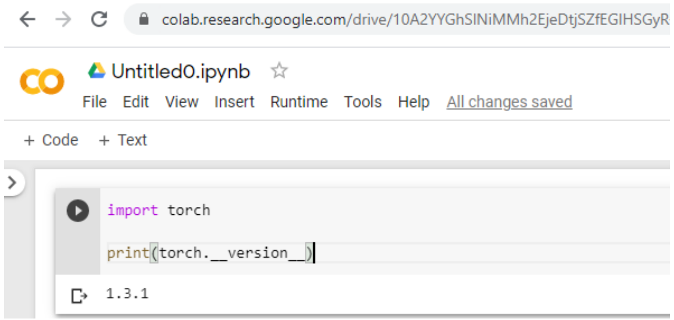


https://www.mygreatlearning.com/blog/pytorch-vs-tensorflow-explained/

Google colab

Cloud computing

- http://colab.research.google.com
- Provides a Jupyter notebook
- Incredible easy to setup
- Provides GPU access (for some time)
- Free of charge
- Interfaces with google drive



Version Control - git and GitHub

Version control system

- Local repositories enable to track changes

  `git init`

  `git add "Your_file.txt"`

  `git commit -am "new commit"`

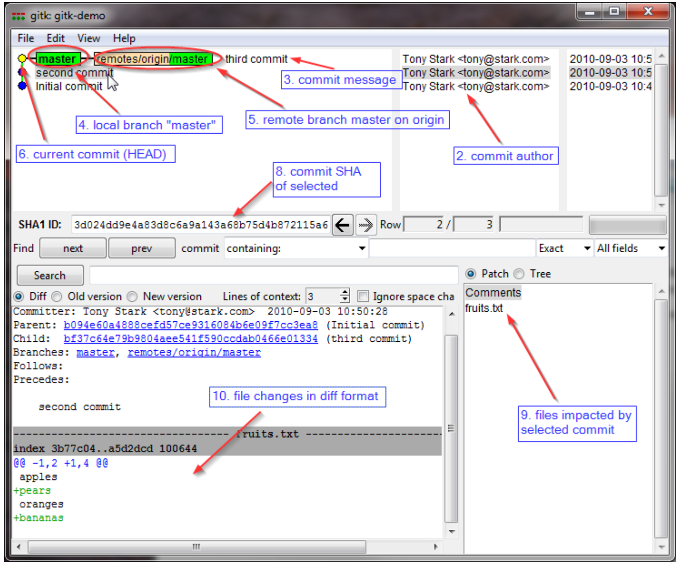
- Remote repositories for backup and collaboration

  `git clone` https://github.com/USER/REPO

  `git push origin master`

- Graphical version tree
  - `gitk`
  - More alternatives

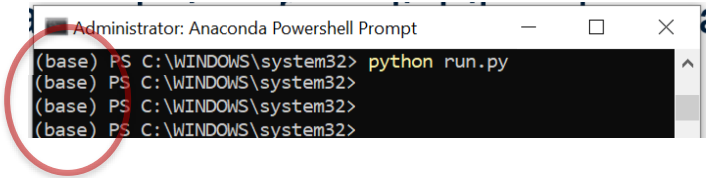
    https://git-scm.com/download/gui/linux



[https://lostechies.com/joshuaflanagan/2010/09/03/use-gitk-to-understand-git/]

Anaconda

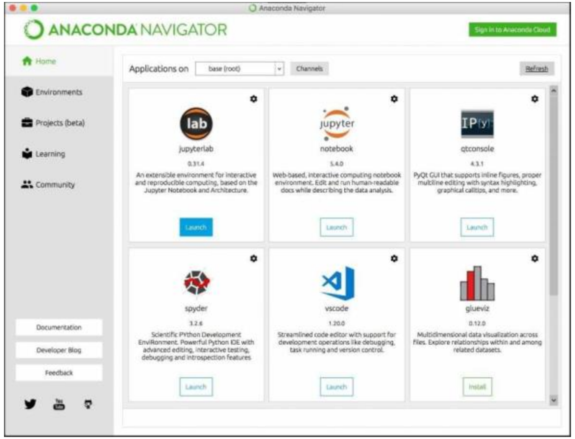
- Anaconda is a free distribution of the Python programming languages
  - build for scientific computing
  - simplifies package management



- Virtual environment manager
  - conflicting packages can be installed independently
  - local installation without root access

- Easy to use
  - graphical interface (Anaconda Navigator)
  - pre-compiled libraries

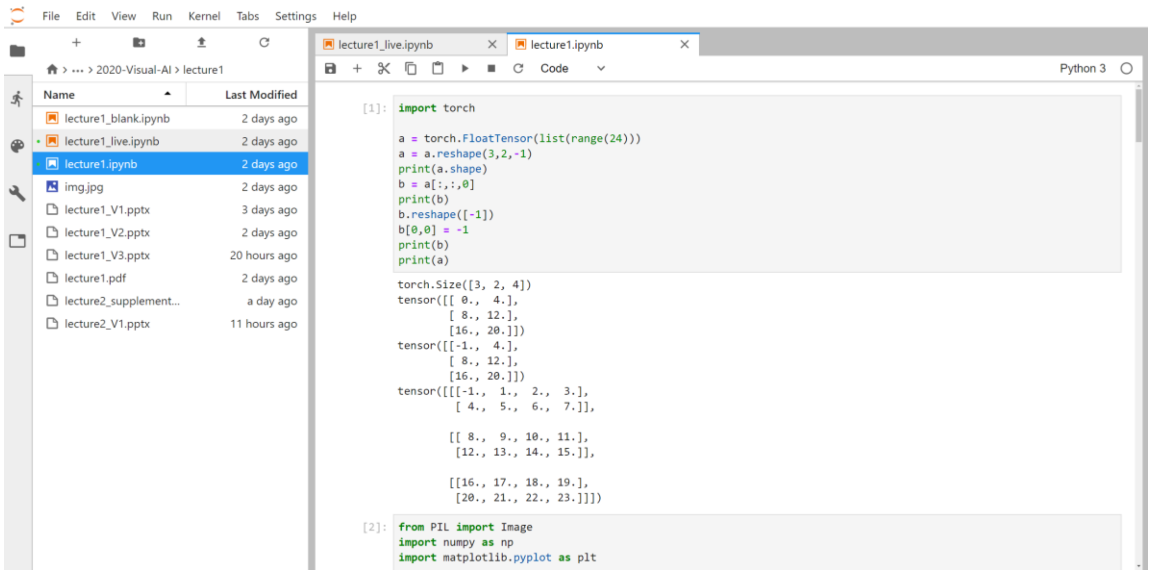
    “conda install pytorch torchvision cudatoolkit=10.1 -c pytorch”



Jupyter notebooks in Jupyter Lab

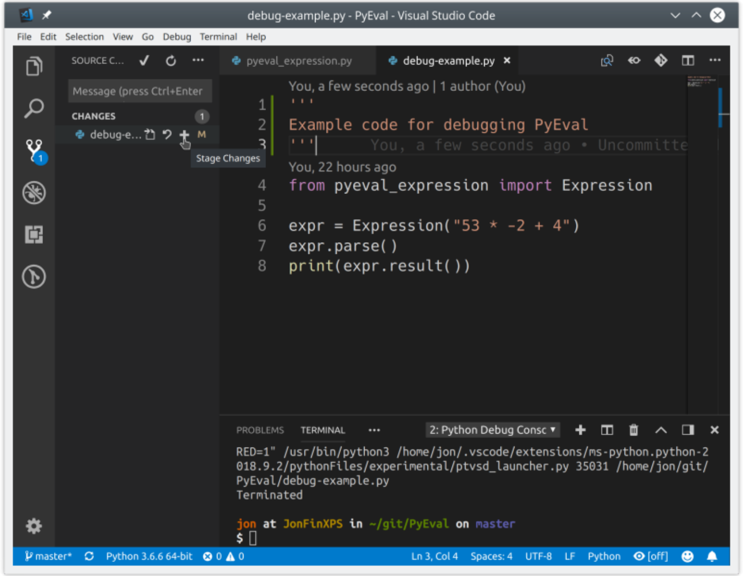
Browser-based editor

- easy to use
- cell-based notebooks (.ipynb)
- Good integration of plotting and interactive tools
- remote access possible (start Jupyter Lab on the server, access url on client)

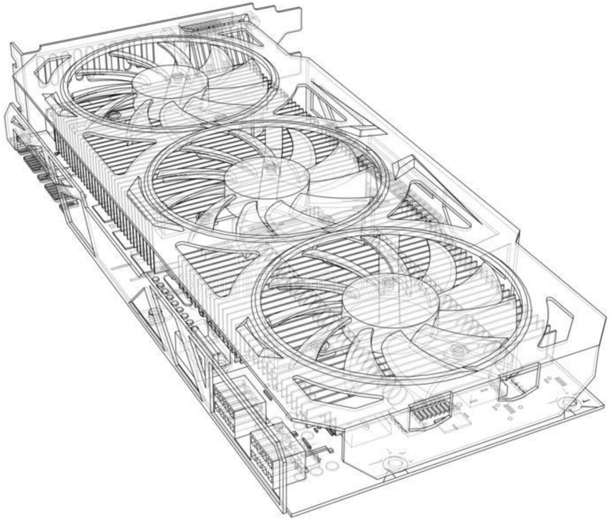


Visual Studio Code

- fast
- python code completion
- git/github integration
- remote server access via ssh
- **collaborative editing functionality**
    - teammates can edit and execute the same code, like google docs but for python



Compute resources



Personal

- Your laptop / desktop
    - No GPU?

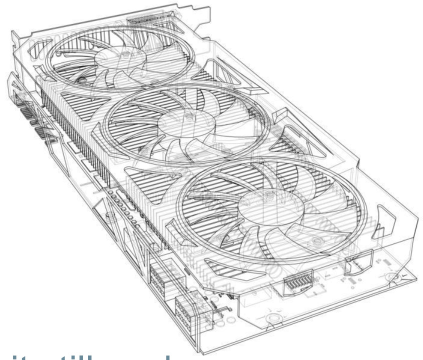
UBC

- lin01.students.cs.ubc.ca to lin25.students.cs.ubc.ca
    - GTX 1060, 3GB memory
    - Will be setup with pytorch for Assignment 2

Cloud computing

- google colab
    - Tesla K80 GPU
      (free so long you have limited workload)

Compute resources at UBC



UBC lin01.students.cs.ubc.ca to lin25.students.cs.ubc.ca

- GTX 1060, 3GB memory, anaconda and pytorch installed
- use UBC vpn: myvpn.ubc.ca using CWL.cs account (note, the .cs is important!)
- Create a session (replace colored parts with your name and ports)

  > ssh -N -f -L 9991 :localhost:9991 rhodin@lin01.students.cs.ubc.ca

  May throws error "bind [::1]:9991: Cannot assign requested address" … but it still works

  > ssh rhodin@lin01.students.cs.ubc.ca

  > /cs/local/lib/pkg/anaconda-2019.07/bin/conda init

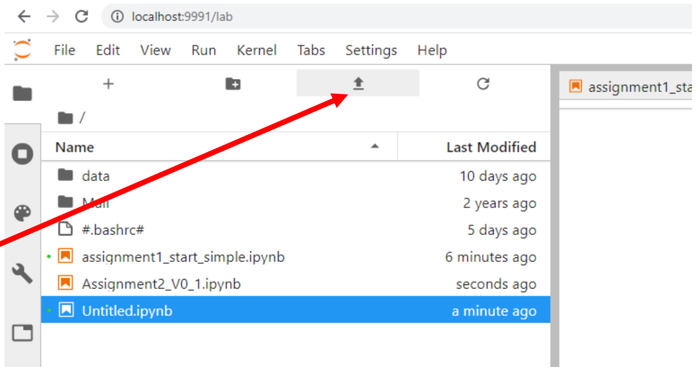
  > jupyter-lab --no-browser --port=9991

- Open link provided by jupyter-lab in browser

  http://localhost:9991/?token=6a7ee7feec81f...

- Upload Assignment.ipynb in Jupiter lab

  Note: running a cell with "import pytorch" might take some seconds



Compute resources at UBC – alternative access

Accessing UBC jupyter servers (slow but easy way)

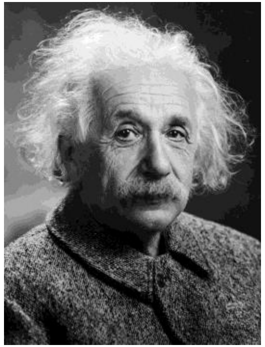
- > ssh -X rhodin@lin01.students.cs.ubc.ca
- > jupyter-lab
- > firefox &

Requires high-speed connection

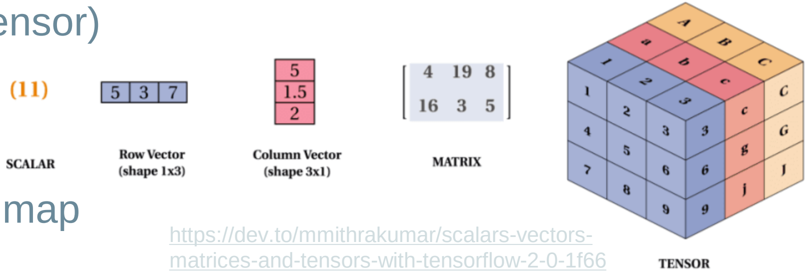
- e.g., within university network

PyTorch and Deep Learning intro

Tensors in pytorch



- Tensor: a multi-dimensional array
  - scaler, vector, matrix, … tensor
- Term hijacked by ML community (in the math/physics community a tensor is a function that can be represented by a multi-dimensional array, but not every array is a math tensor)
- Pytorch uses the NCHW convention:
  dim 0: N, the number of images in a batch
  dim 1: C, the number of channels of an image / feature map
  dim 2: H, the height of the image / feature map
  dim 3: W, the width of the image / feature map



https://dev.to/mmithrakumar/scalars-vectors-matrices-and-tensors-with-tensorflow-2-0-1f66

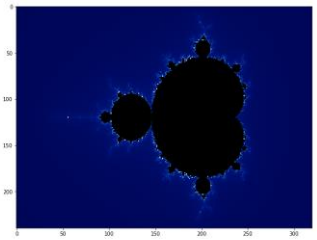
- Different #dimensions possible, dependent on the task

- Order of dimensions matters (cache locality, parallelization)
  - TensorFlow has C in the last dimension, Nervada Neon N

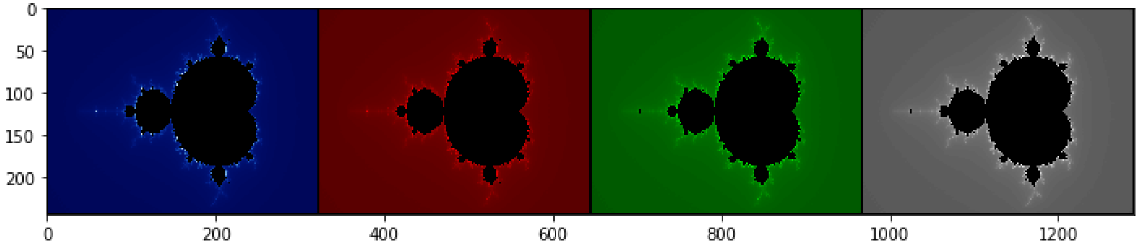
Why is the order of channels different in PyTorch?

```
from PIL import Image
pil_image = torch.FloatTensor(np.array(Image.open("img.jpg")))/256

pil_image.shape
  > torch.Size([240, 320, 3])

tensor_image = pil_image.permute(2, 0, 1)
tensor_image.shape
  > torch.Size([3, 240, 320])

pil_image = tensor_image.permute(1, 2, 0)
plt.imshow(tensor_image.permute(1, 2, 0))

batch = torch.stack([tensor_image , tensor_image , tensor_image])
```

Practical session: working with tensors

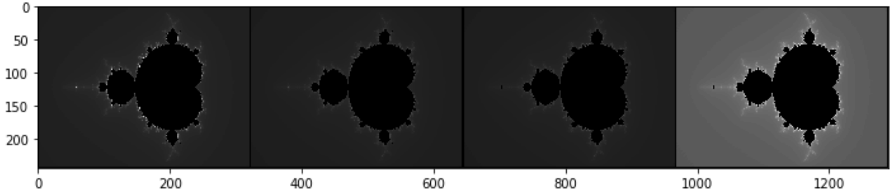


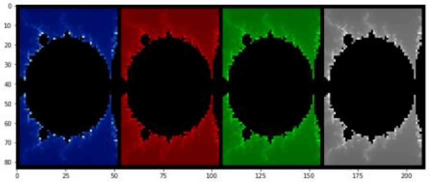
augmentation
and batch creation



zoom

gray-scale

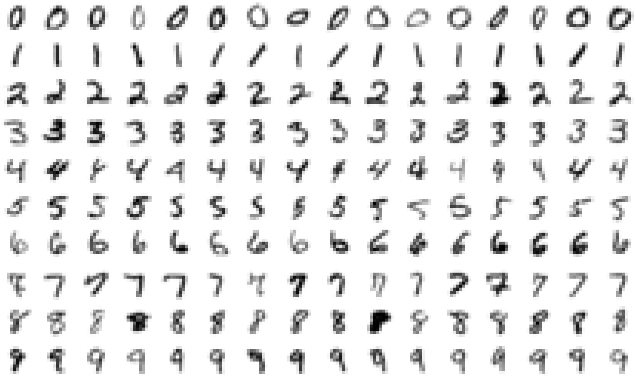




Assignment I

“Playing with PyTorch”

- Network architecture
- Dataloaders
- Evaluation
- Visualization



- Posted on course website
- Submit solution on Canvas

- Work in randomly assigned teams

(optional)

Datasets, preprocessing, and efficient loading

- Well-known **datasets** readily available
  - MNIST, KMNIST, EMNIST, QMNIST, Fashion-MNIST
  - COCO, ImageNet, CIFAR, Cityscapes, Kinetics-400
  - Many more: pytorch.org/docs/stable/torchvision/datasets.html
- Loading custom **datasets**
  - FakeData, ImageFolder, DatasetFolder
  - Write your own __getitem__ function

```
train_set = datasets.FashionMNIST(
    root = './data/FashionMNIST',
    train = True,
    download = True,
    transform = transforms.Compose([
        transforms.ToTensor(),
     ])
)
```

- Efficient **data loaders**
  - queries elements of the dataset
    - using parallel threads
  - outputs batches
    - pinned memory

```
loader = torch.utils.data.DataLoader(
            train_set, batch_size = 8)
```

Hint on PyTorch data loader

```
torch.utils.data.DataLoader(
  dataset,       # dataset from which to load the data.
  batch_size=16,            # how many samples per batch to load
  shuffle=False,            # set to True to have the data reshuffled at every epoch
  sampler=None,             # defines the strategy to draw samples from the dataset
  batch_sampler=None,# like sampler, but returns a batch of indices at a time
  num_workers=0,            # how many subprocesses to use for data loading, 0 means using the main process
  collate_fn=None,          # merges a list of samples to form a mini-batch of Tensor(s)
  pin_memory=True,          # if True, the data loader will copy Tensors into CUDA pinned CPU memory
  drop_last=False,          # drop the last incomplete batch, if the size is not divisible by the batch size
  timeout=0,   # if positive, the timeout value for collecting a batch from workers
  worker_init_fn=None, multiprocessing_context=None) # threading stuff
```

Common issues:

- batch_size < 8 usually works poorly, may not converge
- never use .cuda() in a data loader with *num_workers>0*
- Use pin_memory, which makes the transfer instant

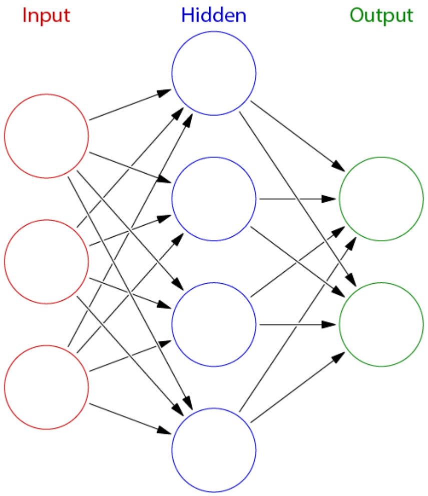
Make sure that you understand all arguments!

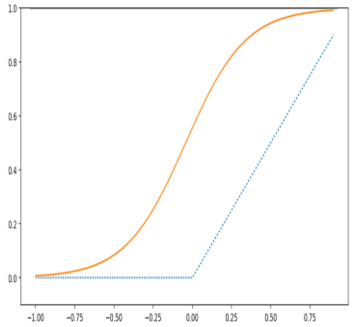
Neural network building blocks (basics)

A summary. More details are provided in the next lecture

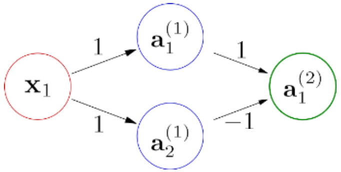
Fully-connected layer

- Linear transformation + activation function (ReLU, sigmoid, tanh, exp)
- Each layer is composed of multiple neurons (same computational rule, different weights)
- Multiple fully-connected layers form a multi layer perceptron (MLP)

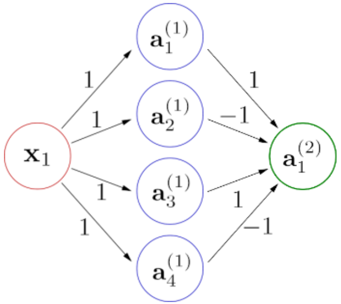




NN theory and universal approximation

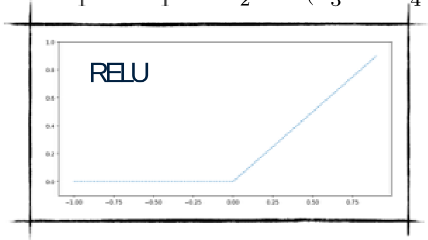


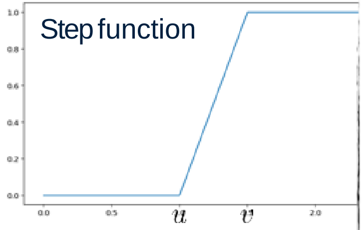
$$\mathbf{a}_1^{(1)} = \texttt{relu}(x-u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x-v)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)}$$

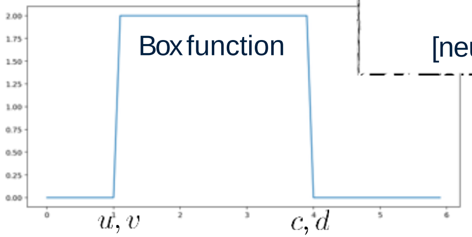


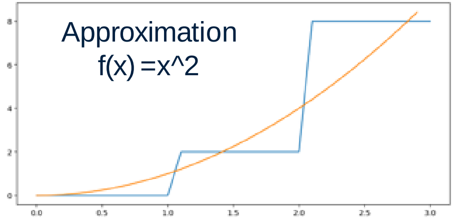
$$\mathbf{a}_1^{(1)} = \texttt{relu}(x-u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x-v)$$
$$\mathbf{a}_3^{(1)} = \texttt{relu}(x-c)$$
$$\mathbf{a}_4^{(1)} = \texttt{relu}(x-d)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)} - (\mathbf{a}_3^{(1)} - \mathbf{a}_4^{(1)})$$

⋮

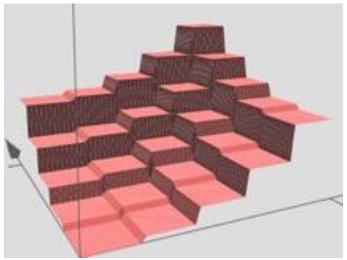








Approximation in 2D



[neuralnetworksanddeeplearning.com]

Mathematical prove in [Hornik et al., 1989; Cybenko, 1989]

Discussion: The role of activation functions

What if we build a NN without activation functions?

Is there an advantage of stacking linear layers?

What else could we do to limit dimensionality?

Neural network definition in pytorch

- Standard architectures

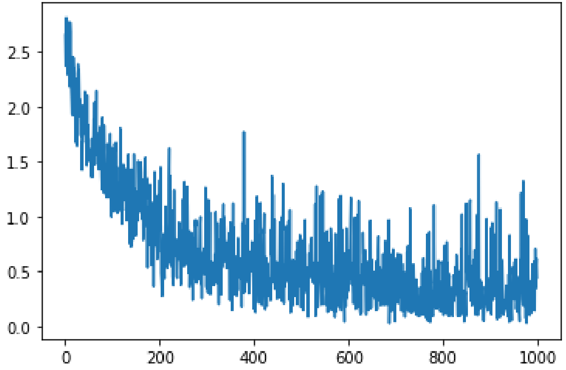
```
network = torchvision.models.resnet18(num_classes=10).cuda()
```

- Custom designs

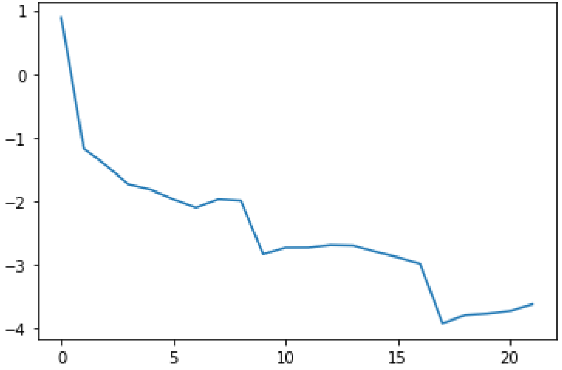
```
class Network(nn.Module):
  def __init__(self):
    super(Network, self).__init__()
    self.conv = nn.Sequential(
        nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(in_channels=6, out_channels=12, kernel_size=5),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
    self.MLP = nn.Linear(in_features=12*4*4, out_features=10)

  def forward(self, batch):
    t = self.conv(batch)
    t = t.reshape(-1, 12*4*4)
    return self.MLP(t)
```

Training and Evaluation

Training loss (per iteration)



Validation loss (per epoch)



Test accuracy (once and for all)

0.94

Golden rule of machine learning: Don’t touch the test set when building your model (including high-level design choices)!

My silver rule: Something is wrong if you don’t use a validation set. Validation is needed to systematically determine hyper parameters (stopping time, network architecture, …).

How else would you determine these? Grad student descent?

Optimization loop in Pytorch

- Iterative local optimization (`opt`) over minibatches (`x, y`) returned by the dataloader (`loader`) using automatic differentiation of the objective and neural network (`loss.backward()`)

```
iterator = iter(loader)
device = "cuda"
for i in range(len(loader)):
    x, y = next(iterator)
    preds = net(x)
    loss = nn.functional.cross_entropy(preds, y)
    opt = optim.SGD(net.parameters(), lr=0.001)

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

Don't forget to zero gradients, pytorch accumulates gradients by default

Separable objective and mini batches

Most common is a separable objective over independent samples x,y

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)},y^{(i)})\in D} l(\quad (\mathbf{x}^{(i)},\theta), y^{(i)})$$

Evaluated over mini batches of size N

Stored as tensor, e.g.,

dim 0: N, number of images in a batch

dim 1: C, number of channels

dim 2: H, height of the feature map

dim 3: W, width of the feature map

Objective function / Loss

General form

$$\arg\min_{\theta} E(D, \theta)$$

Separable form

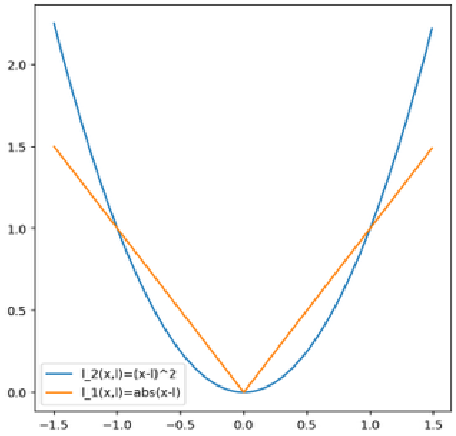
$$E(D, \theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} l(\; f(\mathbf{x}^{(i)}, \theta), y^{(i)})$$

MNIST example

$$E(D, \theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} (\; f(\mathbf{x}^{(i)}, \theta) - y^{(i)})^2$$

$=(\; f($$, \theta) - 7)^2 + (\; f($$, \theta) - 8)^2 \dots$

Note, in PyTorch, a loss is also called a criterion
(a more general term)



Quadratic loss

$$l_2(y, l) = (y - l)^2$$

Absolute loss

$$l_1(y, l) = |y - l|$$

Objective function in pytorch

Regression: squared loss, l1 loss, huber loss…

- nn.functional. MSELoss(pred, gt)

Classification: cross-entropy loss, hinge loss, …

- nn.functional.cross_entropy(pred_probabilities, gt_probabilities)

```
class MyHingeLoss(torch.nn.Module):
  def __init__(self):
      super(MyHingeLoss, self).__init__()
  def forward(self, output, target):
      hinge_loss = 1 - torch.mul(output, target)
      hinge_loss[hinge_loss < 0] = 0
      return hinge_loss
```

How to read a research paper?

Papers aren’t read front to back like a novel! More at: https://web.stanford.edu/class/ee384m/Handouts/HowtoReadPaper.pdf

First, determine your goal:

- Are you searching for something particular?
- Skip paper parts that are irrelevant for your goal
- Use the search functionality if you look for a keyword.

Then:

1. read title
   - is it relevant?
2. read abstract and teaser image
   - is it relevant and insightful?
3. quick pass, focus on figures
   - get general idea about the paper
4. content pass
   - grasp paper contents, but skip details
5. details pass
   - understand the paper in depth, lookup concepts that you don’t understand in other sources


To appear in ACM TOG (SIGGRAPH Asia 2016).

EgoCap: Egocentric Marker-less Motion Capture with Two Fisheye Cameras

Helge Rhodin[1] Christian Richardt[1, 2, 3] Dan Casas[1] Eldar Insafutdinov[1]
Mohammad Shafiei[1] Hans-Peter Seidel[1] Bernt Schiele[1] Christian Theobalt[1]

[1]Max Planck Institute for Informatics [2]Intel Visual Computing Institute [3]University of Bath

Figure 1: *We propose a marker-less optical motion-capture approach that only uses two head-mounted fisheye cameras (see rigs on the left). Our approach enables three new application scenarios: (1) capturing human motions in outdoor environments of virtually unlimited size, (2) capturing motions in space-constrained environments, e.g. during social interactions, and (3) rendering the reconstruction of one’s real body in virtual reality for embodied immersion.*

Abstract

Marker-based and marker-less optical skeletal motion-capture methods use an *outside-in* arrangement of cameras placed around a scene, with viewpoints converging on the center. They often create discomfort with marker suits, and their recording volume is severely restricted and often constrained to indoor scenes with controlled backgrounds. Alternative suit-based systems use several inertial measurement units or an exoskeleton to capture motion with an *inside-in* setup, i.e. without external sensors. This makes capture independent of a confined volume, but requires substantial, often constraining, and hard to set up body instrumentation. Therefore, we propose a new method for real-time, marker-less, and egocentric motion capture: estimating the full-body skeleton pose from a lightweight stereo pair of fisheye cameras attached to a helmet or virtual reality headset – an *optical inside-in* method, so to speak. This allows full-body motion capture in general indoor and outdoor scenes, including crowded scenes with many people nearby, which enables reconstruction in larger-scale activities. Our approach combines the strength of a new generative pose estimation framework for fisheye views with a ConvNet-based body-part detector trained on a large new dataset. It is particularly useful in virtual reality to freely roam and interact, while seeing the fully motion-captured virtual body.

Keywords: Motion capture, first-person vision, markerless, optical, inside-in, crowded scenes, large-scale

Concepts: •**Computing methodologies** → **Motion capture;**

1 Introduction

Traditional optical skeletal motion-capture methods – both marker-based and marker-less – use several cameras typically placed around a scene in an *outside-in* arrangement, with camera views approximately converging in the center of a confined recording volume. This greatly constrains the spatial extent of motions that can be recorded; simply enlarging the recording volume by using more cameras, for instance to capture an athlete, is not scalable. Outside-in arrangements also constrain the type of scene that can be recorded, even if it fits into a confined space. If a recording location is too small, cameras can often not be placed sufficiently far away. In other cases, a scene may be cluttered with objects or furniture, or other dynamic scene elements, such as people in close interaction, may obstruct a motion-captured person in the scene or create unwanted dynamics in the background. In such cases, even state-of-the-art outside-in marker-less optical methods that succeed with just a few cameras, and are designed for less controlled and outdoor scenes [Elhayek et al. 2015], quickly fail. Scenes with dense social interaction were previously captured with outside-in camera arrays of a few hundred sensors [Joo et al. 2015], a very complex and difficult to scale setup.

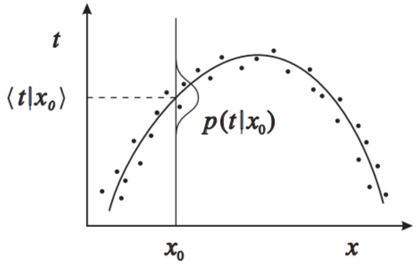
These strong constraints on recording volume and scene density prevent the use of optical motion capture in the majority of real-world scenes. This problem can partly be bypassed with *inside-in* motion-capture methods that use body-worn sensors exclusively [Menache 2010], such as the Xsens MVN inertial measurement unit suit. However, the special suit and cabling are obstructive and require tedious calibration. Shiratori et al. [2011] propose to wear 16 cameras placed on body parts facing *inside-out*, and capture the skeletal motion through structure-from-motion relative to the environment. This clever solution requires instrumentation, calibration and a static background, but allows free roaming. This design was inspirational for our egocentric approach.

1

First two presentations

Week 3, Sep 24:

- Christopher Bishop, *Mixture Density Networks*
    - 1994



- Jonathan T. Barron, *A General and Adaptive Robust Loss Function*
    - CVPR 2019