Basics, History, Meet and Greet, Pytorch

By Konrad Paul Kording

neuromatch academy

Welcome to the Neuromatch Deep Learning course

- We want to teach you what **you** want to learn
- Every aspect of the content should be truly useful
- While acknowledging a wide heterogeneity of students

- This is the first version, we need your feedback

Seriously welcome everyone

welkom
mirë se vini
welkomma
na gasana
bel bonjou
مرحبا
أهلا وسهلا
bari galoust
xos gelmissiniz
i bisimila
akwaba
ongi etorri
Шчыра запрашаем
swagata
amrehba sisswène
sveiki atvykę
welkum
wëllkom
добредојде
tonga soa
Тавтай морилогтун

ani kié
dobro došli
degemer mad
добре дошъл
kyo tzo pa eit
tirow
benvinguts
hafa adai
marsha vog'iyla
ulihelisdi
欢迎你来
bonavinuta
dobrodošli
vítejte
velkommen
selamat datang
swagatham
merħba
haere mai
kakwa o
miawezon

pô la bwam
welkom
welcome
bonvenon
tere tulemast
woezon
vælkomin
tervetuloa
welkom
bienvenue
wolkom
binvignut
awaa waa atuu
benvido
bin la v'nu
ne y waoongo
namaste
byivenun
bianvnu
bein'vnu
velkommen

mobrdzandit
mobrdzaneba
herzlich willkommen
waila andanemai
Καλώς ήλθατε
eguahé porá
mikouabô
bienvéni
e komo mai
ברוך הבא
swaagat
tos txais
üdvözlet
velkomin
nnoo
selamat datang
benvenguts
velkom
Табуафси
bon bini
خوش اومدی

fáilte
benvenuti
yôkoso
amrehva ysswène
suor sdei
nkuîzulu yambote
murakaza neza
환영합니다
nodé
bi xer hati
gnindi ton hap
nkuîzulu yambote
gratus mihi venis
laipni lūdzam
benvegnûo
boyeyi bolamu
witajcie
bem-vindo
mishto-avilian tú
bine aţi venit
добро пожаловать

afio mai
benènnidu)
fàilte
dobrodošli
karibu
mauya
bhali karay aaya
aayuboovan
vitame vás
dobrodošli
zupinje z te videtite
soo dhawaw
bienvenidos
wilujeng sumping
karibuni
hush kelibsiz
mới đến
bénvnou
croeso
dalal ak diam
ékouabô / ékabô

välkommen
härzliche wöikomme
maligayang pagdating
maeva
nal-varravu
rahim itegez
swagatham
ยินดีต้อนรับ
tashi delek
malo e lelei
difika dilenga
hoş geldiniz
gazhasa oetiśkom
Ласкаво просимо
khush amdeed

Who is Konrad?

- Data Scientist
- Neuroscientist
- Fitting Neural networks to neurons since 1999
- Asking all kinds of DL questions

- Big time salsa dancer
- Co-founder of Neuromatch movement
- Curriculum Chair for DL

The reasons for this course

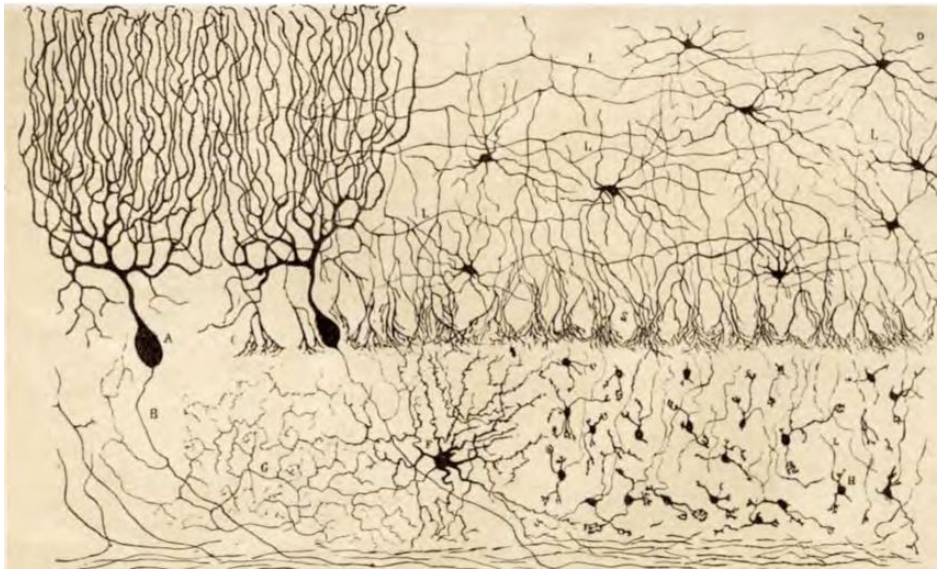
Last year Neuromatch was great

Many of us were interested in Deep Learning

We perceived a lack of a course that is

- friendly
- broad
- thorough

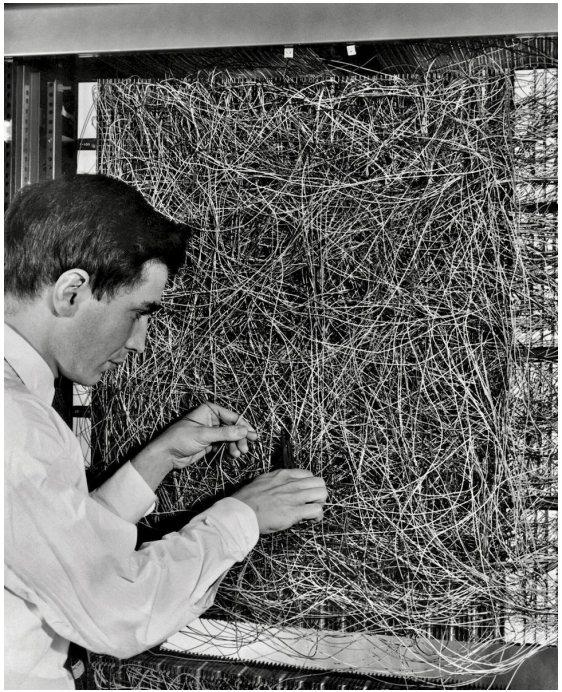
A very short history of Artificial Neural Networks/ Deep Learning



Chick
Cerebellum
Golgi Stain
Ramon y Cajal
Nobel: 1906

Linear learning in Perceptron: Mark I: 1958

$$y = \sigma\left(\mathbf{w}^t \mathbf{x}\right)$$



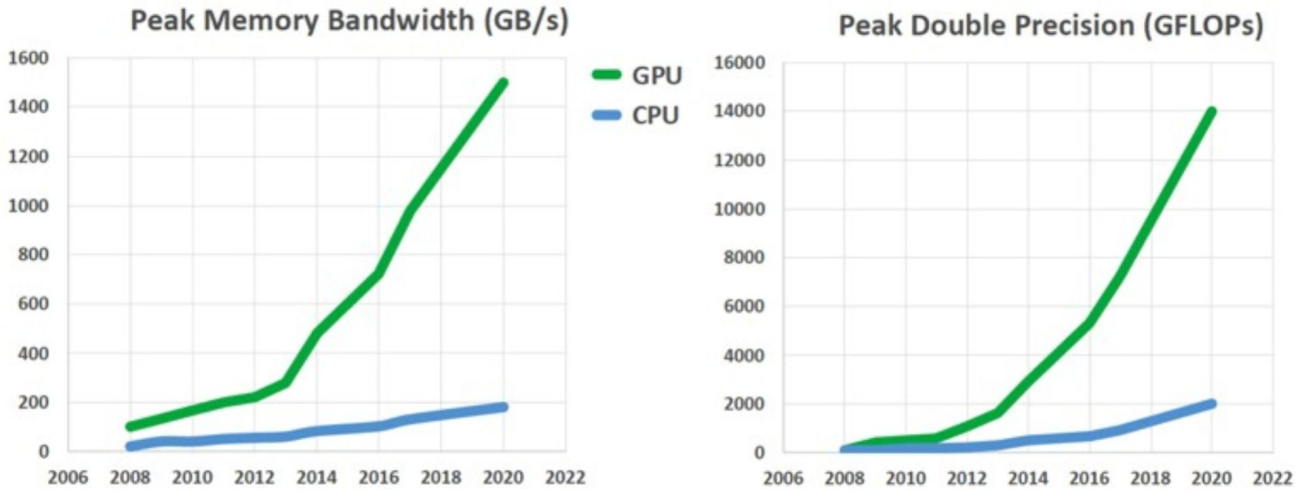
Rosenblatt
(source wikipedia)

Nonlinear learning in multi-layer perceptrons

$$y = \sigma\left(w_2^t \sigma\left(w_1^t \mathbf{x}\right)\right)$$

...

GPUs



source:
https://www.nextplatform.com/2019/07/10/a-decade-of-accelerated-computing-augurs-well-for-gpus/

Lots of Data!

- 2005-2012: Pascal Visual Object Classes
  - 20 classes, 27.5k annotations (in most recent)
- 2010: ImageNet
  - 27 categories, 21.3k subcategories, ~1M images with annotations!
- 2014-2017 COCO
  - 80 classes, ~250k images with bounding boxes and segmentations
- 2017-2019 OpenImages
  - 9M images, 16M bounding boxes, 600 classes, 2.8M segmentation masks

... And lots of data augmentation! (e.g. rotation, crop, add noise, etc)

Today, Deep Learning matters

- State of the art in most domains of Machine Learning
  - Computer Vision
  - Natural Language Processing
  - Time Series Processing
  - Reinforcement Learning

Deep Learning (DL) is Machine Learning (ML)

- DL Inherits most of its ideas from regular ML
  - Regularization
  - Establishing success
  - Interface to reality
  - Statistics
  - Concepts

Why is this course special?

Complete overview of the DL field

Heavy focus on learning by doing, flipped classroom

You code immediately

A focus on team work and group based learning

We believe in learning in teams

- Get to know your TA and your pod
- Learn to be comfortable around them about the things you do not know
- Be helpful. Help them learn.
- Your group can be a great source of strength

What is awesome about some pods I got to know

Some people great at math

Others good at coding

Some see the ethical/ society dimensions

Everyone is having a great time

Code of conduct

1. Be inclusive.
Welcome and support everyone - any sexual orientation, gender identity and expression, race, ethnicity, culture, national origin, social and economic class, educational level, color, immigration status, sex, age, size, family status, political belief, religion, and mental and physical ability.

2. Be considerate.
Think about how your decisions affect your fellow students, your TAs, and your colleagues around the world.

3. Be respectful.
Disagreement is no excuse for disrespectful behavior.

4. Choose your words carefully.
Conduct yourself professionally when you write and speak, and be kind. Do not insult, harass, or put down others.

5. Don't harass.
In general, if someone asks you to stop something, then stop. Try to resolve disagreements and differing views constructively.

6. Make differences into strengths.
We have strength in our diversity. Different people have different perspectives on issues, and that can be valuable for solving problems or generating new ideas.

https://docs.google.com/document/d/1eHKIkaNbAIbx_92tLQeIXnicKXEcvFzlyzzeWjEtifM/edit?usp=sharing

My expectations

I expect all of you to be able to use DL after the course

You will need more courses to become a DL researcher

But you should be at the level where you can have fun with it + solve real problems

See through the hype

I also expect you to help us make the course better

Back to you!

Introduce yourself to your pod
Discuss with your pod what you are most hoping to get out of this course, which week will be your favorite?
Talk about the code of conduct and why it matters

Come back in 30 minutes max

Why I am so happy I learned DL

Changed my way of thinking

Many things in the world get better by small steps:

- Evolution
- Economy
- Brains
- Most new things that actually work

Face manipulations

Captioning

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

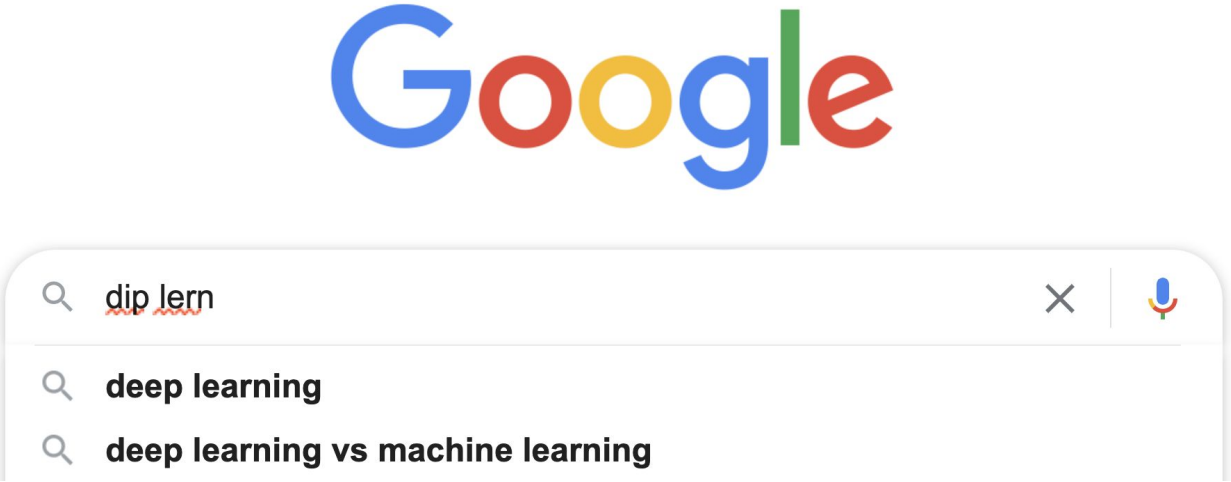
"two young girls are playing with lego toy."

Andrej Karpathy, Li Fei-Fei

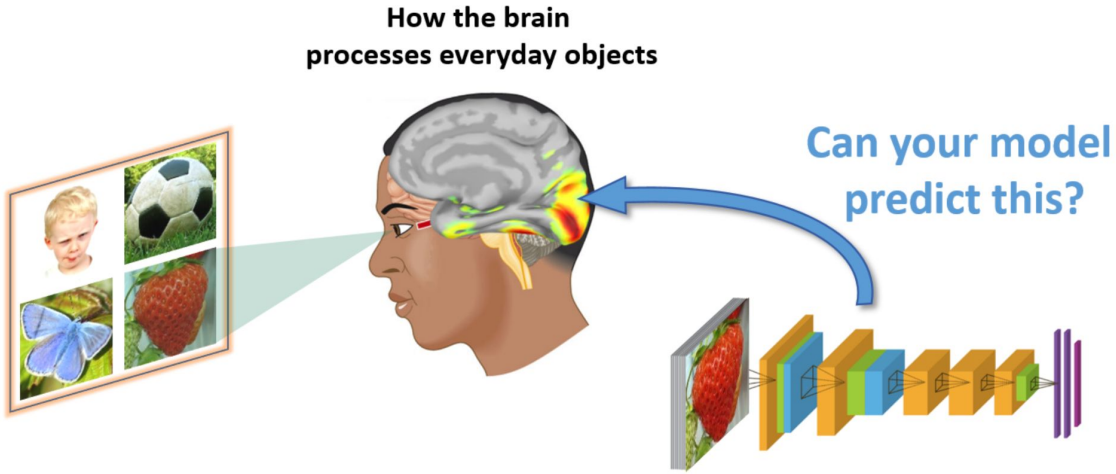
Alpha Zero


Tore verrore mperspedi cuptatur eta audicto p. XX
Quasi volut accae cullit ad queomni quis pp. XX & XX
Perspedi cuptatur audicto volut accae cullit p. XX
Science
$15
7 DECEMBER 2018
sciencemag.org
AAAS
31 63%
18 47%
16%
A DIGITAL
PRODIGY
AlphaZero teaches itself chess, shogi, and Go
pp. XXXX, XXXX, & XXXX

Actual google



Neuroscience



The algonauts project, brain-score.org

And also

I personally believe that the brain does something like gradient descent

e.g. Kording and Konig 2004, Richards et al 2020

What we will learn

- Today: **welcome.** get to know your pod. See how DL models work. Get used to pytorch
- Week 1 : **the nuts and bolts.** Universal components.
- Week 2: **doing more with less.** Convnets, RNNs, Self-supervised learning
- Week 3: **advanced topics.** RL, GANs, Continual Learning, Causality

Domains taught on side

- **Computer Vision:** convnet days
- **Natural Language processing:** RNN/ Attention days
- **Whatever else you are interested in:** Your projects

Can you just focus on what matters most to you?

In short: **No**

In more detail: DL consists of countless tricks. We make sure that as many as possible are taught. But without earlier lessons you will miss the crucial details in the later ones.

History of this course

Konrad has been (co)teaching DL at UPenn for 3 years
Since last year with Lyle Ungar
This year, brought it into Neuromatch format to test the curriculum

The neuromatch course is what the world's best Deep Learning professors do if they join forces

Week 1, Day 2: Linear Networks: Andrew Saxe

Done wonderful work on theory of deep learning, Oxford

Teaches:

Linear learning

Practical skills

Intro to designing networks

Week 1, Day 3: Surya Ganguli

Bringing physics thinking to deep learning, Stanford

Teaches:

Multilayer perceptrons

How to build them

How they actually work

Week 1, Day 4: Yannis Mitliagkas

Works on Optimization and Learning, Montreal

Teaches:

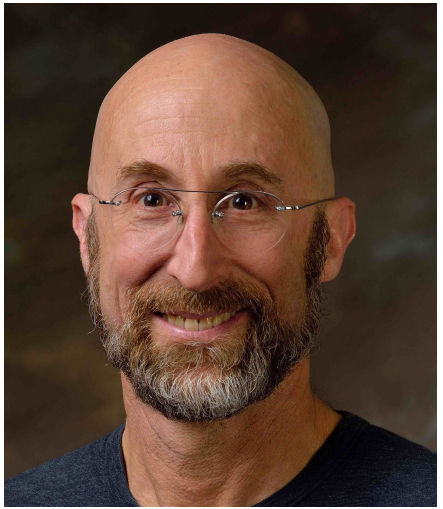
Optimization

Week 1, Day 5: Lyle Ungar

Works on Machine Learning in Medicine, explainable AI

Teaches:

Regularization



Week 2, Day 1: Alona Fyshe

Works on Computational Linguistics, Deep Learning and Neuroscience, Alberta

Teaches:

Parameter sharing: Convnets and Recurrent Neural Networks

Week 2, Day 2: Alexander Ecker

Works on Style transfer, computational neuroscience, Tuebingen

Teaches:

Modern Convnets

Week 2, Day 3: James Evans

Works on computational social science. Jost wrote DL textbook, University of Chicago

Teaches

Recurrent Neural Networks

Week 2, Day 4: He He

Works on Text generation and Language understanding

Teaches:

Attention and Transformers

Week 2, Day 5: Vikash Gilja and Akash Srivastava

Vikash works on Brain Machine Interfaces, San Diego

Akash works on Unsupervised learning, MIT/ IBM

Teaching

Variational Autoencoders and Generative Adversarial Networks

Week 3, Day 1: Blake Richards and Tim Lillicrap

Blake works on computational neuroscience, McGill
Tim works on reinforcement learning, Deepmind

Teaching:
Unsupervised and Self-supervised learning

Week 3, Day 2: Jane Wang and Feryal Behbahani

Brain inspired algorithms, Deepmind

Teaching:

Reinforcement learning

Week 3, Day 3: Blake Richards and Tim Lillicrap

Blake works on computational neuroscience, McGill
Tim works on reinforcement learning, Deepmind

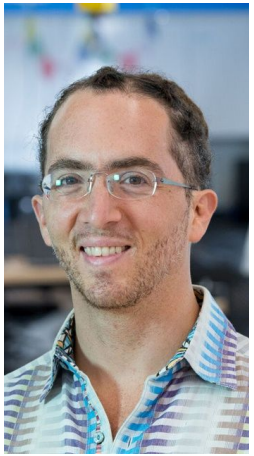
Teaching:
Reinforcement Learning for games


Week 3, Day 4: Josh Vogelstein and Vincenzo Lomonaco

Both work on continual learning

Teaching:

Continual learning, Causality, and the future of Deep Learning



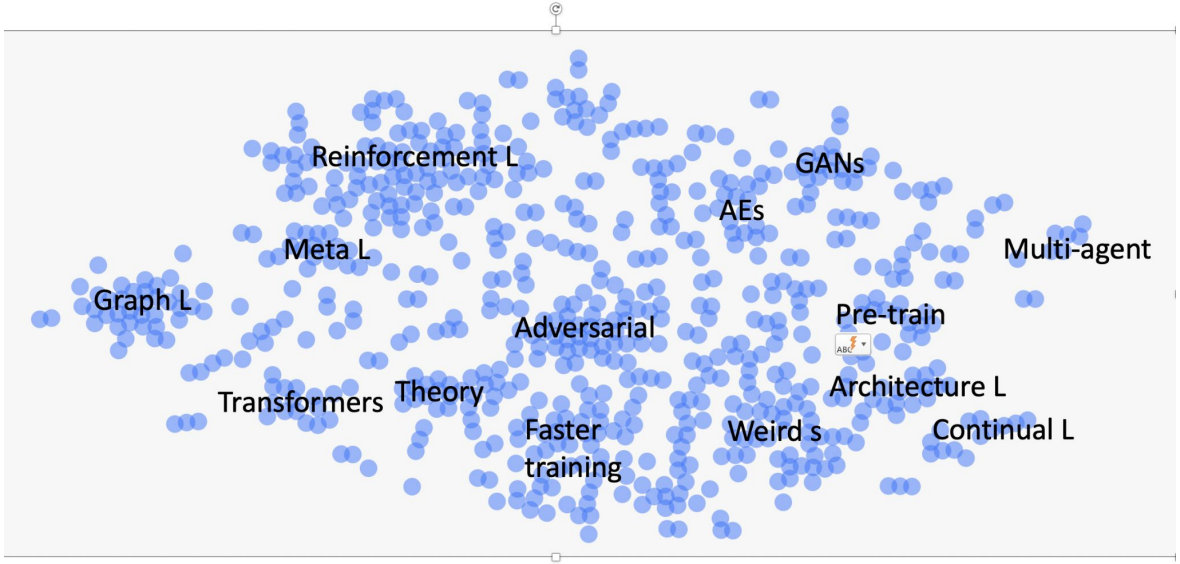
Week 3, Day 5: You

Your projects

Your ideas

Your celebrations

Gestalt of current DL



Find out where you want to be on the map

Run the visualization
Look at a few abstracts.

Come back in 15 minutes max

Course Logistics: points of contact

Primary - google colab

- tutorials run it it
- videos play out of it (they are also on youtube playlist for fullscreen)
- airtable questionnaires run out of it

Your TA/ pod

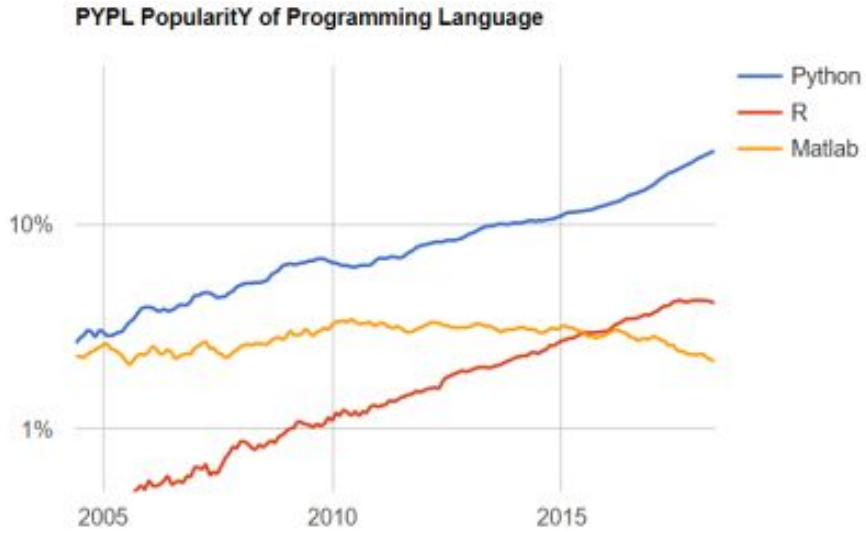
Neurostars - for questions/ discussion

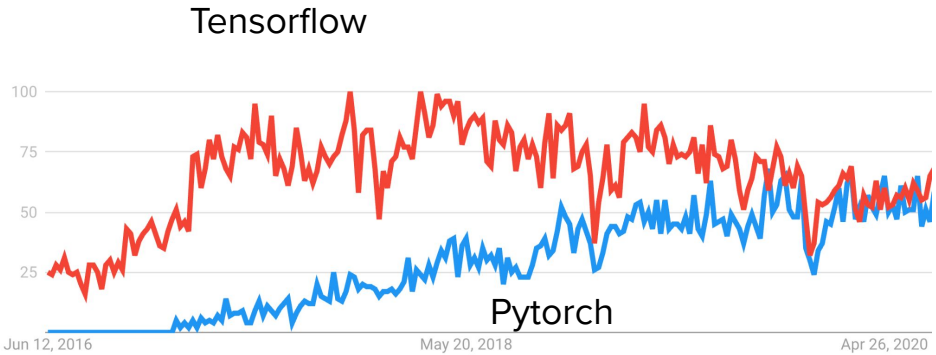
Video meeting software - for live sessions

Email - feel free to email us at nma@neuromatchacademy.org

and feel free to email me personally at koerding@gmail.com

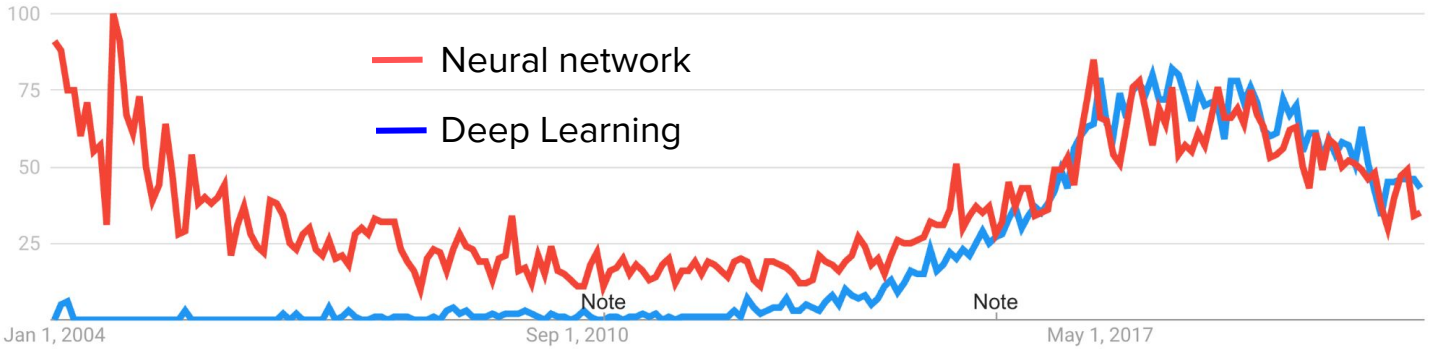
Python and Pytorch





source
https://blog.revolutionanalytics.com/2018/06/pypl-programming-language-trends.html

Are we beyond peak DL?



Pytorch

Numpy
on a GPU
with all kinds of ANN related things
with a focus on tensors
and automatic differentiation

Alternatives to pytorch

Tensorflow - strength in commercial applications

JAX - strength in flexibility

Matlab

DL in Numpy

```
    w2 = w2_init;
    b2 = b2_init;

    for iter = 1: 5000
        z = x_train * w1 + b1;
        a = 1./(1 + exp(-z));
        eta = 1./(1 + exp(-(a*w2+b2)));
        eps = eta - y_train;
        dgz = a .* (1 - a);

        dw1 = (w2 .* ((eps .* dgz)' * x_train))'./m_train;
        db1 = (w2 .* (dgz' * eps))'./m_train;
        dw2 = (eps' * a)'./m_train;
        db2 = sum(eps)/m_train;

        w1 = w1 - step .* dw1;
        b1 = b1 - step .* db1;
        w2 = w2 - step .* dw2;
        b2 = b2 - step .* db2;
    end

    pred_a_train = 1./(1 + exp(-(x_train * w1 + b1)));
    pred_eta_train = 1./(1 + exp(-(pred_a_train * w2 + b2)));
    pred_y_train = sign(pred_eta_train .* 2 - 1);
    pred_y_train = (pred_y_train + 1) ./ 2;
    err_train = classification_error(pred_y_train, y_train);
    err_train_arr(1, i) = err_train;

    pred_a_test = 1./(1 + exp(-(x_test * w1 + b1)));
    pred_eta_test = 1./(1 + exp(-(pred_a_test * w2 + b2)));
    pred_y_test = sign(pred_eta_test .* 2 - 1);
    pred_y_test = (pred_y_test + 1) ./ 2;
    err_test = classification_error(pred_y_test, y_test);
    err_test_arr(1, i) = err_test;

end

%%
plot(d1_arr, err_train_arr, 'color', 'b'); hold on;
plot(d1_arr, err_test_arr, 'color', 'r'); hold on;
plot(d1_arr, err_cv_arr, 'color', 'g');


%%
disp(err_train_arr); hold on; %0.3960    0.1440    0.1120    0.0760    0.0640    0.0360
disp(err_test_arr); hold on; %0.3939    0.2066    0.1726    0.1563    0.1538    0.1544
disp(err_cv_arr); %0.3960    0.1840    0.1640    0.1000    0.1520    0.1440



%%
w1_init = load(strcat(path_init, "w1_", num2str(5), ".mat"));
```

```
b1_init = load(strcat(path_init, "b1_", num2str(5), ".mat"));
b1_init = b1_init.b1;

w2_init = load(strcat(path_init, "w2_", num2str(5), ".mat"));
w2_init = w2_init.w_2;

b2_init = load(strcat(path_init, "b2_", num2str(5), ".mat"));
b2_init = b2_init.b2;

%%
x_train = trainData;
x_train(:, d) = [];
y_train = trainData(:, d);
y_train = (y_train + 1)./2;

%%
x_test = testData;
x_test(:, d) = [];
y_test = testData(:, d);
y_test = (y_test + 1)./2;

%%
step = 0.1;
%%
w1 = w1_init;
b1 = b1_init;
w2 = w2_init;
b2 = b2_init;
for iter = 1: 5000
    z = x_train * w1 + b1;
    a = 1./(1 + exp(-z));
    eta = 1./(1 + exp(-(a*w2+b2)));
    eps = eta - y_train;
    dgz = a .* (1 - a);

    dw1 = (w2 .* ((eps .* dgz)' * x_train))'./m_train;
    db1 = (w2 .* (dgz' * eps))'./m_train;
    dw2 = (eps' * a)'./m_train;
    db2 = sum(eps)/m_train;

    w1 = w1 - step .* dw1;
    b1 = b1 - step .* db1;
    w2 = w2 - step .* dw2;
    b2 = b2 - step .* db2;
end

pred_a_train = 1./(1 + exp(-(x_train * w1 + b1)));
pred_eta_train = 1./(1 + exp(-(pred_a_train * w2 + b2)));
pred_y_train = sign(pred_eta_train .* 2 - 1);
pred_y_train = (pred_y_train + 1) ./ 2;
```

DL in Pytorch

```
# Define the structure of your network
def __init__(self):
  super(NaiveNet, self).__init__()

  # The network is defined as a sequence of operations
  self.layers = nn.Sequential(
      nn.Linear(2, 16), # Transformation from the input to the hidden layer
      nn.ReLU(),        # Activation function (ReLU)
      nn.Linear(16, 2), # Transformation from the hidden to the output layer
  )

# Specify the computations performed on the data
def forward(self, x):
  # Pass the data through the layers
  return self.layers(x)
```

Everything in pytorch are tensors: how to make one

```
# tensor from a list
a = torch.tensor([0,1,2])

#tensor from a tuple of tuples
b = ((1.0, 1.1), (1.2,1.3))
b = torch.tensor(b)

# tensor from a numpy array
c = np.ones([2,3])
c = torch.tensor(c)
```

More tensors: common constructors

```
x = torch.ones(5, 3)
y = torch.zeros(2)
z = torch.empty(1,1,5)
```

Making random tensors

```
# uniform distribution
a = torch.rand(1,3)

# normal distribution
b = torch.randn(3,4)
```

Ranges in pytorch - just like in numpy

```
a = torch.arange(0,10,step=1)
b = np.arange(0,10,step=1)

c = torch.linspace(0,5,steps=11)
d = np.linspace(0,5,num=11)
```

Now that you heard how to make tensors

Make a couple of Tensors

Come back in 10 minutes max

What can we do with tensors?

Everything we do with numpy otherwise.

```
torch.add(a, b, out=c)
torch.multiply(a,b, out=d)
```

By default everything is pointwise

```
x + y, x - y, x * y, x / y, x**y
```

Sums, means etc

Just like in numpy

```
print("Sum of every element of x: ", x.sum())
print("Sum of the columns of x: ", x.sum(axis=0))
print("Sum of the rows of x: ", x.sum(axis=1))
```

Practice it.

Time to do a few things with Tensors

Come back in 10 minutes max

Manipulating tensors: indexing

```
x = torch.arange(0, 10)
print(x)
print(x[-1])
print(x[1:3])
print(x[:-2])
```

```
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
tensor(9)
tensor([1, 2])
tensor([0, 1, 2, 3, 4, 5, 6, 7])
```

Similar logic as numpy for n-dimensional tensors

```
x = torch.rand(1,2,3,4,5)

print(" shape of x[0]:", x[0].shape)
print(" shape of x[0][0]:", x[0][0].shape)
print(" shape of x[0][0][0]:", x[0][0][0].shape)
```

```
 shape of x[0]: torch.Size([2, 3, 4, 5])
 shape of x[0][0]: torch.Size([3, 4, 5])
 shape of x[0][0][0]: torch.Size([4, 5])
```

Flattening/ Reshaping

```
z = torch.arange(12).reshape(6,2)
print("Original z: \n ", z)

# 2D -> 1D
z = z.flatten()
print("Flattened z: \n ", z)
```

```
Original z:
  tensor([[ 0,  1],
          [ 2,  3],
          [ 4,  5],
          [ 6,  7],
          [ 8,  9],
          [10, 11]])
Flattened z:
  tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

Reshaping

```
# and back to 2D
z = z.reshape(3, 4)
print("Reshaped (3x4) z: \n", z)
```

```
Reshaped (3x4) z:
 tensor([[ 0,  1,  2,  3],
         [ 4,  5,  6,  7],
         [ 8,  9, 10, 11]])
```

Irrelevant dimensions

```
x = torch.randn(1,10)
# printing the zeroth element of the tensor will not give us the first number!

print(x.shape)
print("x[0]: ",x[0])
```

```
torch.Size([1, 10])
x[0]:  tensor([-0.7178,  0.2696, -0.7311, -0.7625,  0.6477, -0.5313, -0.0666, -0.7753,
        -2.3820, -0.8742])
```

Squeezing

```
# lets get rid of that singleton dimension and see what happens now
x = x.squeeze(0)
print(x.shape)
print("x[0]: ", x[0])
```

```
torch.Size([10])
x[0]:  tensor(-0.7178)
```

Also: `y = y.unsqueeze(1)`

Dimension permutation

E.g. going from RGB in dimension 1 to in dimension 3

```
# `x` has dimensions [color,image_height,image_width]
x = torch.rand(3,48,64)

# we want to permute our tensor to be [ image_height , image_width , color ]
x = x.permute(1,2,0)
# permute(1,2,0) means:
# the 0th dim of my new tensor = the 1st dim of my old tensor
# the 1st dim of my new tensor = the 2nd
# the 2nd dim of my new tensor = the 0th
print(x.shape)
```

```
torch.Size([48, 64, 3])
```

Concatenation

```
# Create two tensors of the same shape
x = torch.arange(12, dtype=torch.float32).reshape((3, 4))
y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])


#concatenate them along rows
cat_rows = torch.cat((x, y), dim=0 )

# concatenate along columns
cat_cols = torch.cat((x, y), dim=1 )
```

```
Concatenated by rows: shape[6, 4]
 tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.],
        [ 2.,  1.,  4.,  3.],
        [ 1.,  2.,  3.,  4.],
        [ 4.,  3.,  2.,  1.]])

 Concatenated by colums: shape[3, 8]
 tensor([[ 0.,  1.,  2.,  3.,  2.,  1.,  4.,  3.],
        [ 4.,  5.,  6.,  7.,  1.,  2.,  3.,  4.],
        [ 8.,  9., 10., 11.,  4.,  3.,  2.,  1.]])
```

torch and numpy are friends

```
x = torch.randn(5)
print(f"x: {x}   |   x type:   {x.type()}")

y = x.numpy()
print(f"y: {y}   |   y type:   {type(y)}")

z = torch.tensor(y)
print(f"z: {z}   |   z type:   {z.type()}")
```

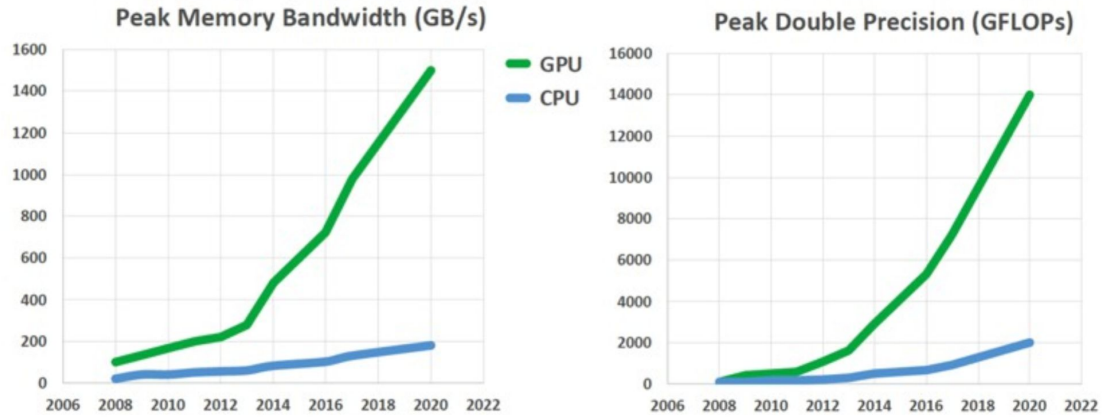
```
x: tensor([ 0.1647, -0.4206, -1.0624,  0.0760, -0.9115])  |  x type:  torch.FloatTensor
y: [ 0.1647326  -0.42064342 -1.062387    0.07596353 -0.91154546]  |  y type:  <class 'numpy.ndarray'>
z: tensor([ 0.1647, -0.4206, -1.0624,  0.0760, -0.9115])  |  z type:  torch.FloatTensor
```

Time to practice these

Trust me, these “easy” things are where the errors often happen

Do the tensor manipulation exercise

Graphics cards: using GPUs



Ask torch where a variable is

```
x = torch.randn(10)
print(x.device)
```

```
cpu
```

Ask torch if we have a GPU

```
print(torch.cuda.is_available())
```

```
True
```

Specifying devices

```
# common device agnostic way of writing code that can run on cpu OR gpu
# that we provide for you in each of the tutorials
device = "cuda" if torch.cuda.is_available() else "cpu"

# we can specify a device when we first create our tensor
x = torch.randn(2,2, device=device)
print(x.dtype)
print(x.device)

# we can also use the .to() method to change the device a tensor lives on
y = torch.randn(2,2)
print(f"y before calling to() |  device: {y.device} | dtype: {y.type()}")

y = y.to(device)
print(f"y after calling to() |  device: {y.device} | dtype: {y.type()}")
```

```
torch.float32
cuda:0
y before calling to() |  device: cpu | dtype: torch.FloatTensor
y after calling to() |  device: cuda:0 | dtype: torch.cuda.FloatTensor
```

Device matters: no mix and match

We can not just mix and match devices - it would be undefined where the computation happens

```
x = torch.tensor([0,1,2], device="cuda")
y = torch.tensor([3,4,5], device="cpu")

z = x + y
```

Moving CPU<->GPU is easy

```
x = torch.tensor([0,1,2], device="cuda")
y = torch.tensor([3,4,5], device="cpu")
z = torch.tensor([6,7,8], device="cuda")

# moving to cpu
x = x.cpu()
print(x + y)

# moving to gpu
y = y.cuda()
print(y + z)
```

```
tensor([3, 5, 7])
tensor([ 9, 11, 13], device='cuda:0')
```

I promised you GPUs are faster

Test the GPU effect

Datasets

Data

+

Model

+

Training

=

DL system

Doing data - basics

Data science =

50% figure out the question you want to answer

35% sweat the data

10% ML

5% glorious DL

How to get data

A lot of data is easy to load for our DL experiments

```
cifar10_data = datasets.CIFAR10(
    root="data",              # path where the images will be stored
    download=True,            # all images should be downloaded
    transform=ToTensor()      # transform the images to tensors
)
```

Let us look into this

Display CIFAR image

Data

Data is not made in heaven

Data is made to answer questions

We need to be agile with data

When we try to answer questions we do not want to lie to ourselves

Let us not lie about data

In DL we generally do prediction

Caution with Causality

The real world differs from our dataset (external validity)

Let us not lie about data: Validation

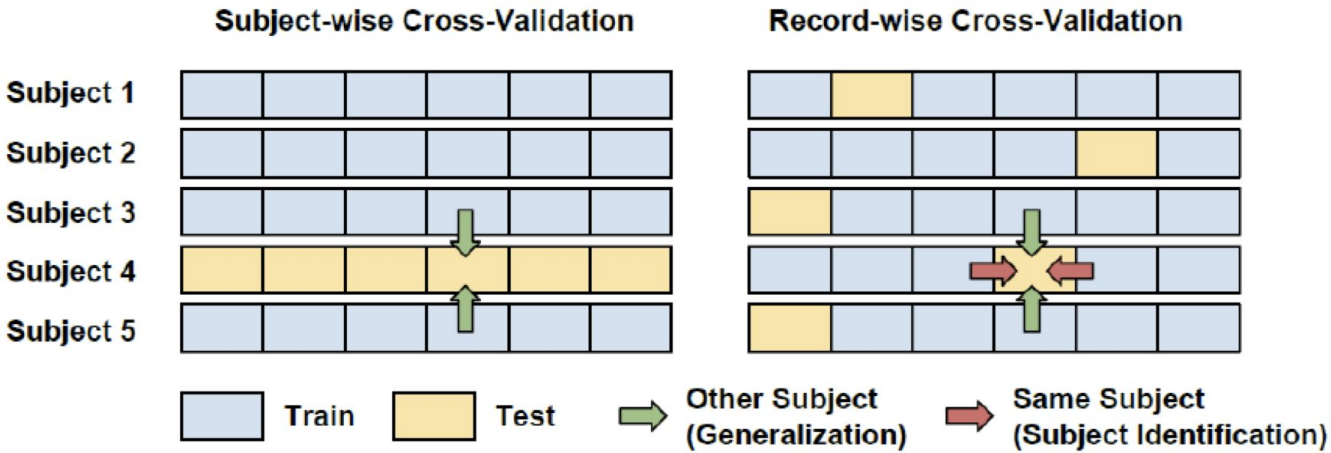
Always have a validation/ Test set not used for training.

Train on training set, test on test set

For hyperparameter optimization you need to further divide the training set

Match the cross-validation strategy to the use case

The cross-validation strategy must match the use case



Saeb ... Kording 2018

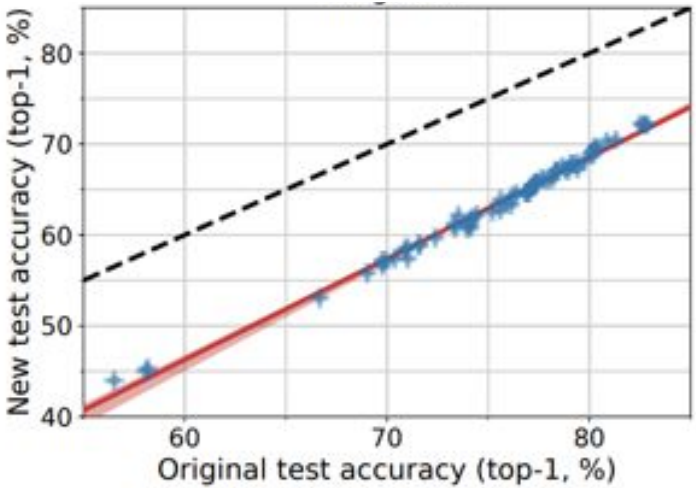
Overfitting! Validation set

Don’t trust yourself

Overfitting is massive for smaller datasets

Ideally have a part of the dataset you don’t have access to

Even some signs for *huge* datasets (imagenet)



Recht et al 2019

Always have both training and test data

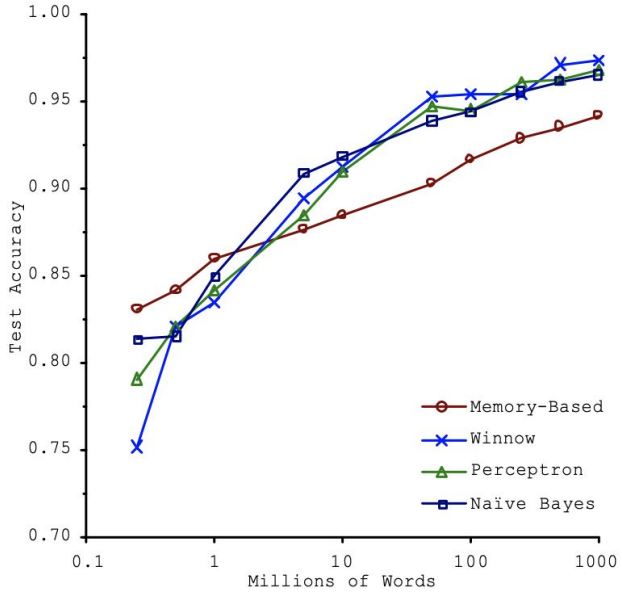
```
# Load the training samples
training_data = datasets.CIFAR10(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

# Load the test samples
test_data = datasets.CIFAR10(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

Let us load some data and divide into train and test dataset

See how CIFAR train and test data are loaded

More data is what it is all about



Banko and Brill 2001

Transformations

More data = better learning

How to get more data?

- Get more data
- Transform the data

e.g. add color variation, etc.

Transformations are crucial across DL

Data Loaders

In practice we do not load all data.

But small pieces (minibatches)

For that we have a function that does the loading

```
# Create dataloaders with
train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

Practice transformation

Load CIFAR images as grayscale

Now, let us design a neural network

(step 0) Get Data

(step 1) All the variables and structures we need.

We need to initialize them

(step 2) And then we need to use these variables to define the compute in our network

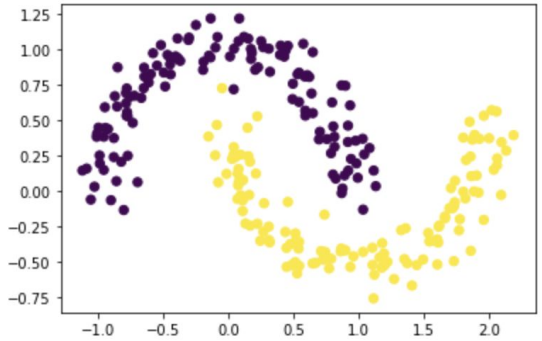
(step 3) And then we need gradients

(step 4) And then we need to optimize

(step 5) And then we need to test

Let us get the data from a csv file

Why? Because many real world datasets are in that format



Look through the code

Can you think of other datasets you could load this way?

Load from csv and put on GPU into Torch

Let us see the anatomy of the network

First, we need to initialize the relevant variables

```
__init__
```

And then we need to specify how information travels through network

```
forward
```

We will often need to make predictions

While many people just use `forward` we will separate it and use

`predict`

and then we need to

`train`

With __init__ we make network structure

```
# Define the structure of your network
def __init__(self):
  super(NaiveNet, self).__init__()

  # The network is defined as a sequence of operations
  self.layers = nn.Sequential(
      nn.Linear(2, 16), # Transformation from the input to the hidden layer
      nn.ReLU(),        # Activation function (ReLU)
      nn.Linear(16, 2), # Transformation from the hidden to the output layer
  )
```

The other components

```
# Specify the computations performed on the data
def forward(self, x):
  # Pass the data through the layers
  return self.layers(x)

# Convert the output of the network to a probability distribution
def predict(self, x):
  # Pass the data through the networks
  output = self.forward(x)

  # Choose the label with the highest score
  return torch.argmax(output, 1)

# Train the neural network (will be implemented later)
def train(seld, X, y):
  pass
```

Check if it actually works and provides the outputs we expect

Run your first neural network

Ok hold on. What has just happened

We have a neural network

It is initialized

It produces outputs

But these outputs are not better than chance yet!

Training = lots of small steps into good direction

```
# The Cross Entropy Loss is suitable for classification problems
loss_function = nn.CrossEntropyLoss()

# Create an optimizer (Stochastic Gradient Descent) that will be used to train the network
learning_rate = 1e-2
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

# Number of epochs
epochs = 15000
```

The anatomy of the training loop

```
for i in range(epochs):
  # Pass the data through the network and compute the loss
  y_logits = model(X)
  loss = loss_function(y_logits, y)

  # Clear the previous gradients and compute the new ones
  optimizer.zero_grad()
  loss.backward()

  # Adapt the weights of the network
  optimizer.step()
```

Just like magic

Woodcut illustration from an edition of Pliny the Elder's *Naturalis Historia* (1582)

Train your network

We give you code. What happens?

Now play with this

All the rest of this course are just variations of this
So it is a good time to really play.

Try changes to code

The rest of this course

We now need intuition for what works and what does not

Architectures

Transfer functions

etc.

One way to start is to take a simple network and explore

The XOR problem

Two inputs 1 output

Output = 1 if the two inputs are different (0 1) or (1 0), 0 otherwise

There is a history to this

Run the XOR widget

Ethics

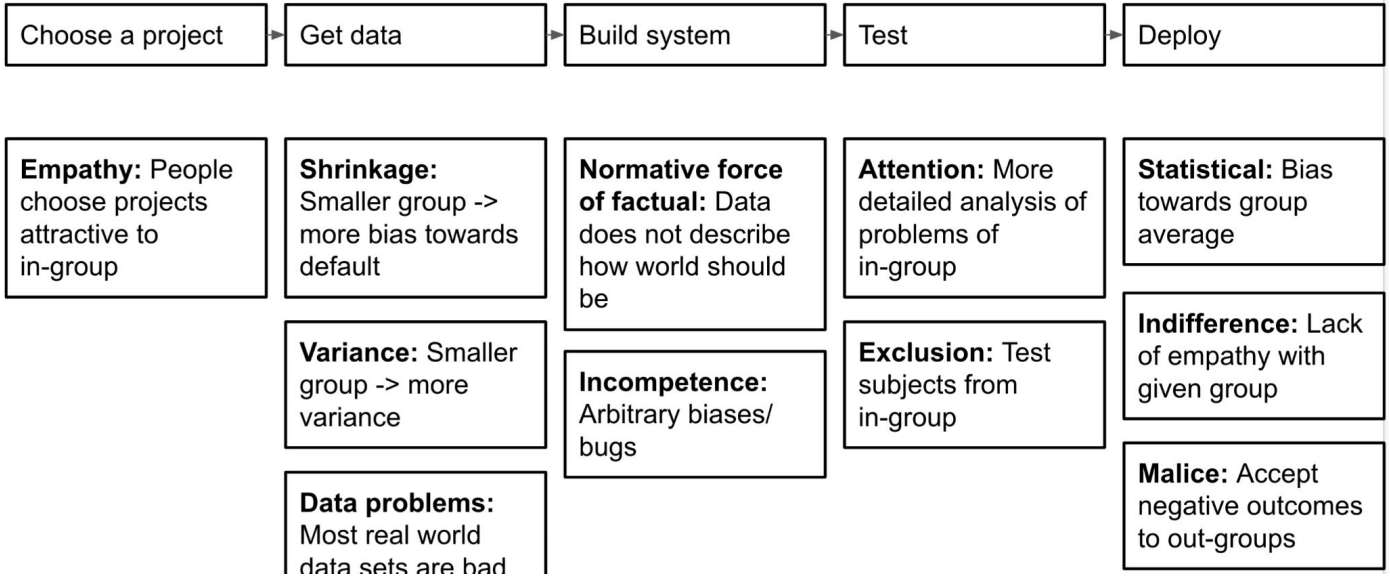
ML is powerful

ML affects lives

Lots of things to say about ethics

But we will instead just listen to victims

Biases, from the unavoidable to the bad



Instead of me telling you about it

Watch coded bias as a pod

What we learned today

Where data is coming from

How to construct a network (init and forward)

How to optimize a network (training loop)

How to evaluate a network

The need to consider the impact on society

These steps always matter

We also learned about the landscape of DL

Computer Vision

Natural Language Processing

Reinforcement Learning

Generative Adversarial Networks

Recurrent Neural Networks

Why are we here?

We believe in global inclusion.

We believe in group based learning.

We believe in the world coming together to make amazing materials.

Group based learning

You will not know/ understand things. Help one another. Not knowing = good

This will be hard. Be there for one another

Trust one another. Not knowing something is the only reason why you are here

Network with one another. Many of you will know one another for live

Hold one another accountable. Email one another if someone is missing.

Get to know your group

Write about 3 of your pod-mates

We use airtable for many things

Get the answers to the questions we ask

Tell us how long which component takes

etc.

Please fill out our questionnaires every day

Lastly

We need your feedback

Submit your work (airtable)

And once you do, please please please

Do the airtable “how does it feel” questionnaire

Scratch cells

Separate video. No frame