Statistics Without Borders

Deep Learning II

Luca Alberto Rizzo

STATISTICS WITHOUT BORDERS

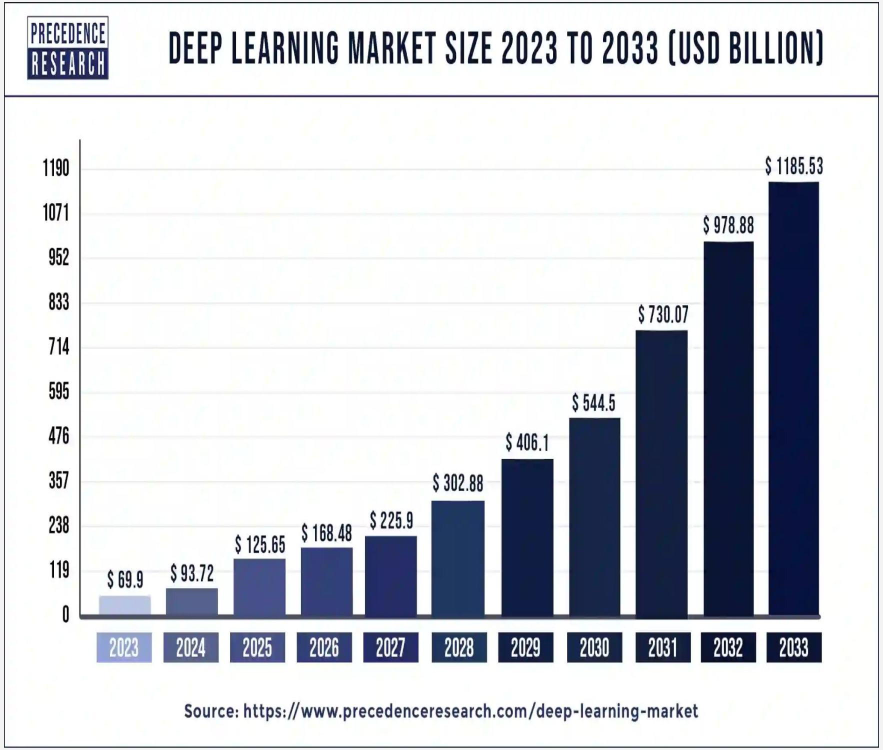
Course outline

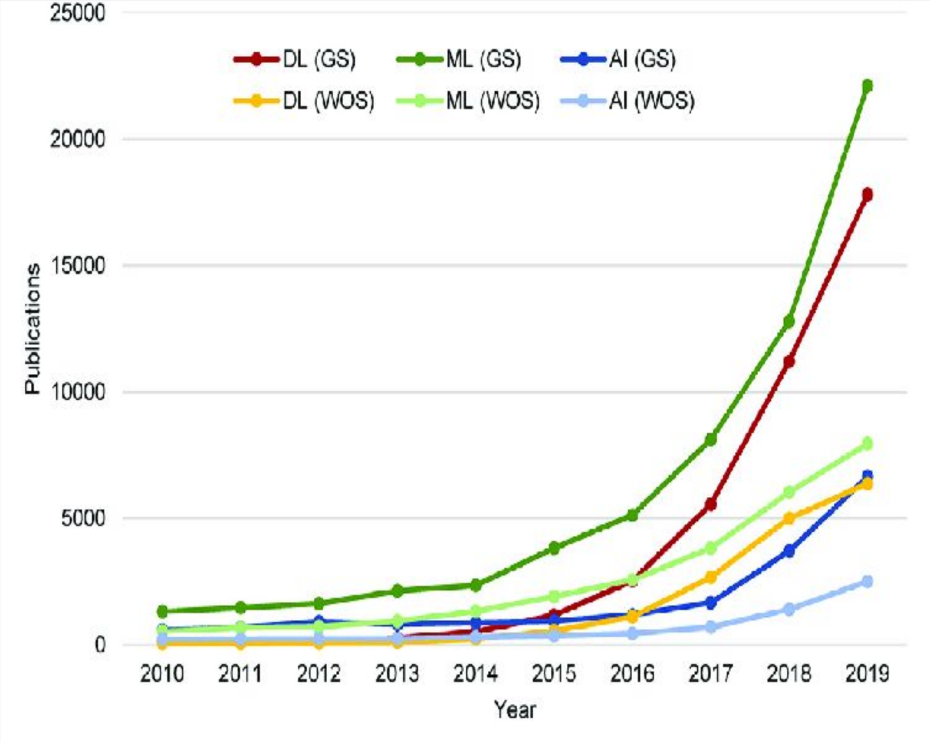
1. Applications of Deep Learning
2. What to do before you start your DL project
3. Introduction to torchvision
4. Opening (a bit) the DL blackbox
5. Additional reading
6. Presentation of the demonstration

Applications of Deep Learning

STATISTICS WITHOUT BORDERS

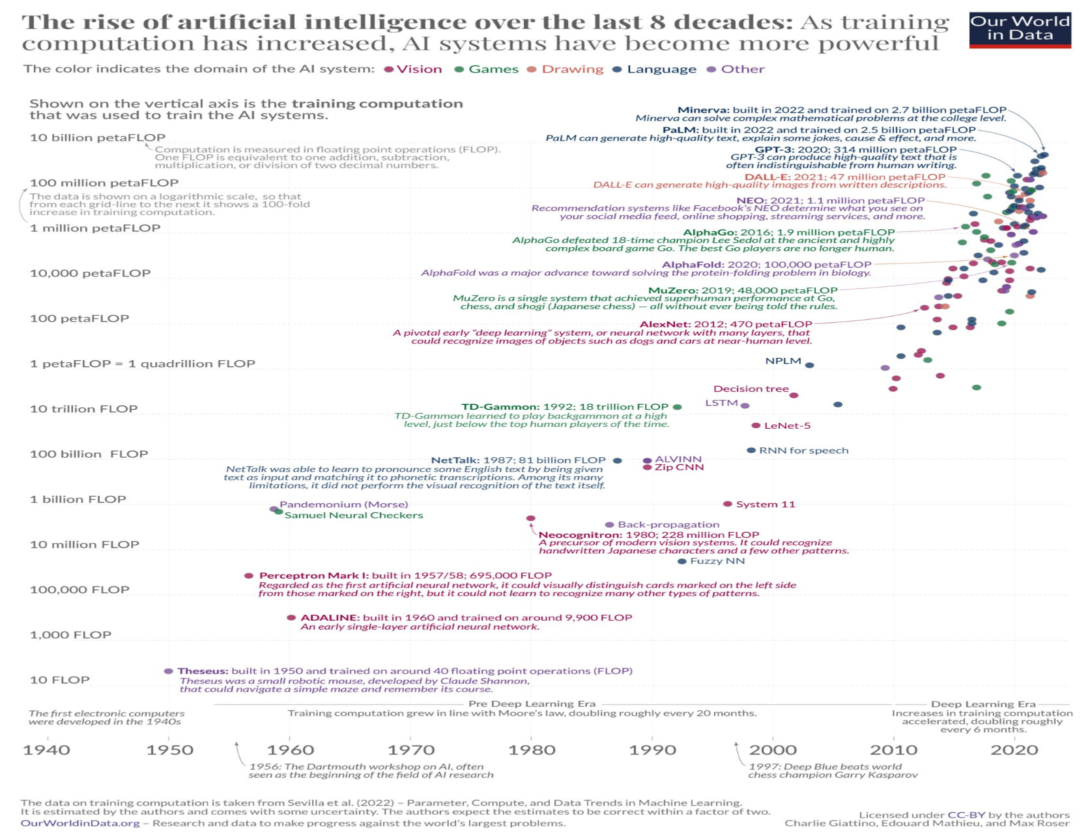
Applications of Deep Learning: exponential growth





[Predence research](#) [A Bird's-Eye View of Deep Learning in Bioimage Analysis, E. Meijering](#)

Applications of Deep Learning: why now?



Simultaneously exponential growth of:

- **computational power**
- **data size**

Deep Learning performs **better with a lot of data** but **requires a lot of computational power**

[Source : Our World in Data]

Applications of Deep Learning: examples

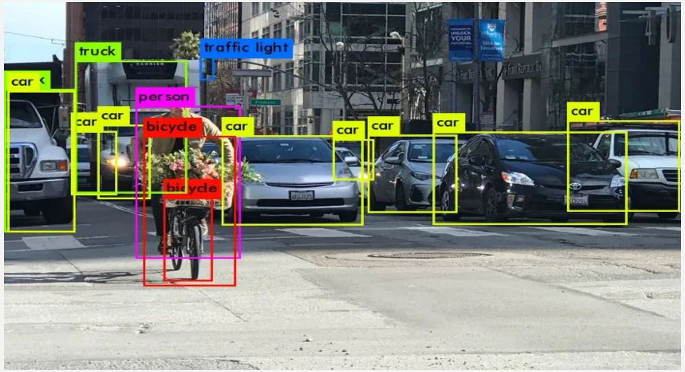
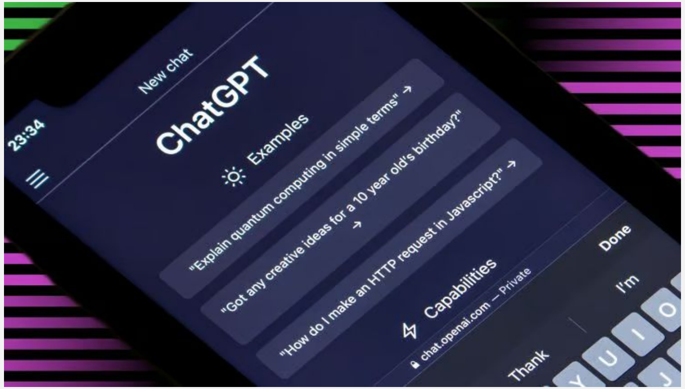
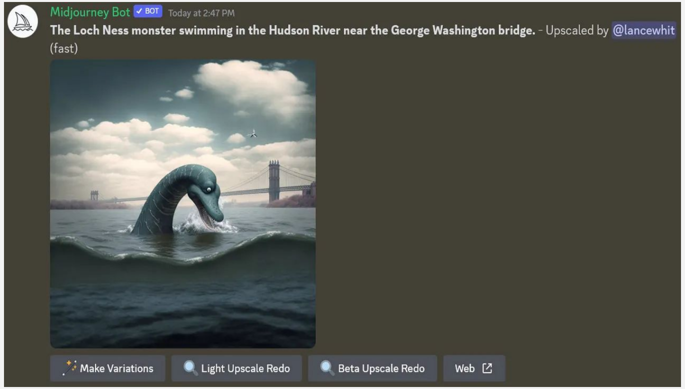


Image recognition with YOLO

Portfolio prediction in Finance



Natural language processing (e.g. ChatGPT)



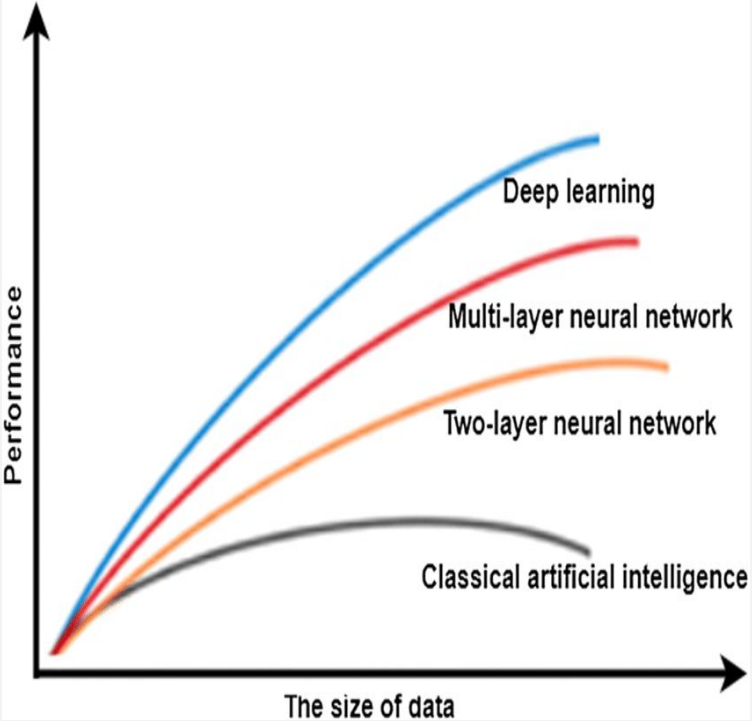
Natural language processing (e.g. ChatGPT)

What to do before you start your DL project

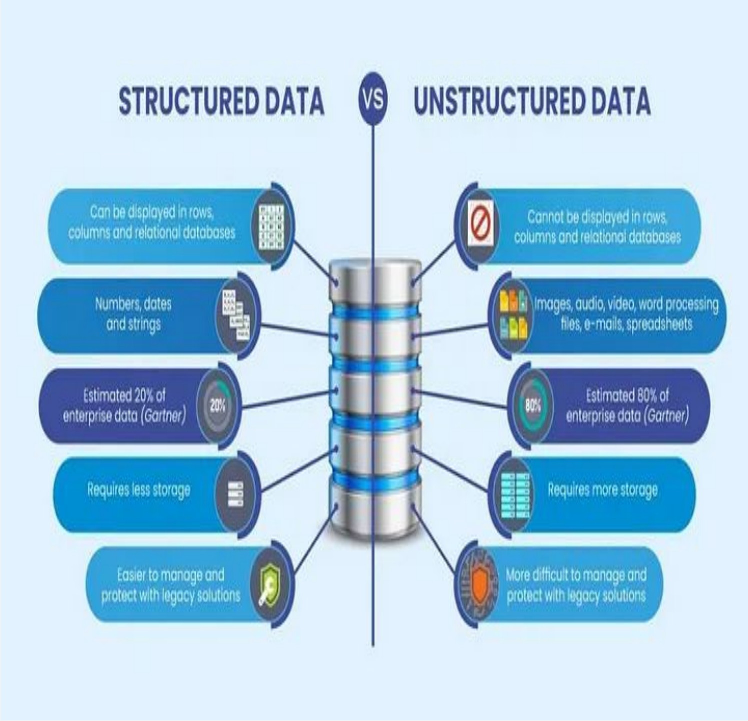
STATISTICS WITHOUT BORDERS

What to do before you start: how complex is your problem?

[Review of tool condition monitoring in machining and opportunities for deep learning]



[Review of tool condition monitoring in machining and opportunities for deep learning]



If you have enough unstructured data, it might be a good idea to use Deep Learning

What to do before you start: when you might not use DL

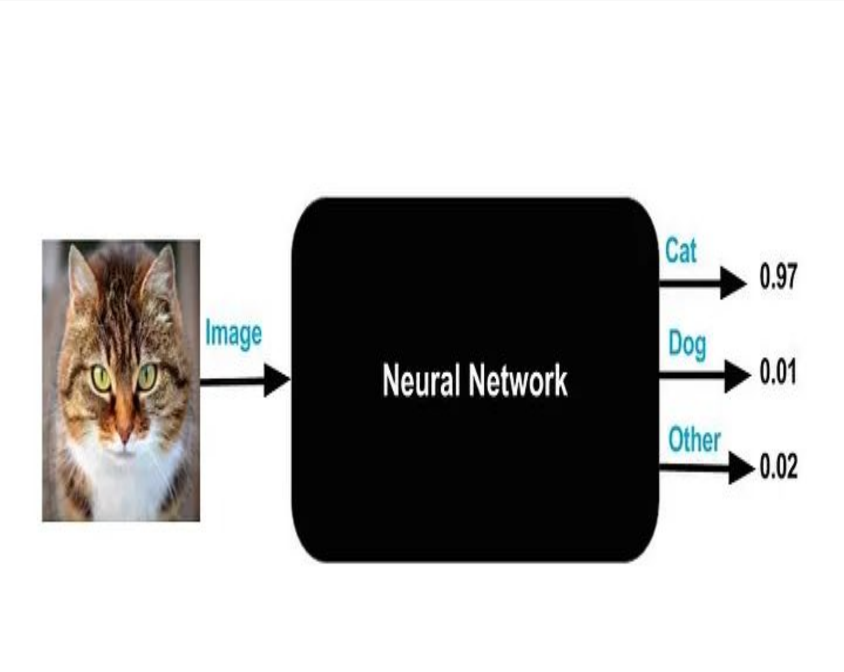
Deep Learning is:

- Expensive
- Not interpretable (even more than ML)
- Very hard to optimize
- Not so great with tabular data

However:

- It works extremely well for unstructured data!
- Plenty of resources around

PLEASE RESIST THE HYPE!

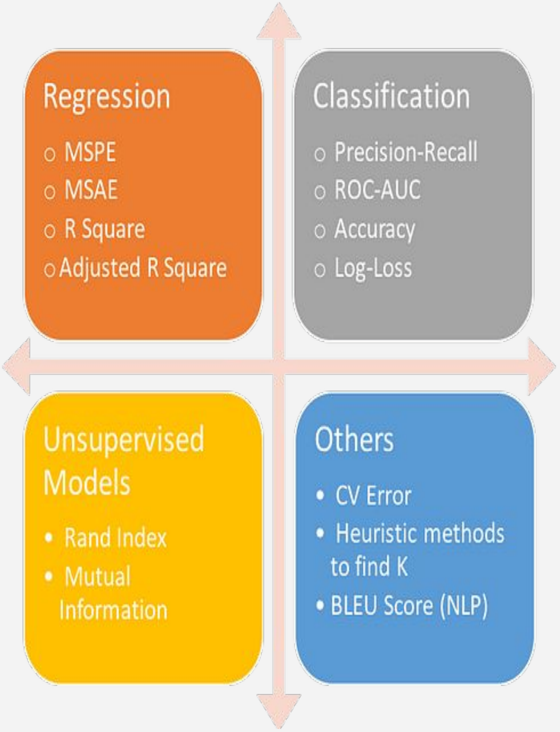


[How to explore Neural networks, the black box ?]

What to do before you start: how to choose the right metric

Your model is as good as your metric

- Talk with the final user (several times)
- Think about worst and best case scenarios
- Connect final metric with training metrics

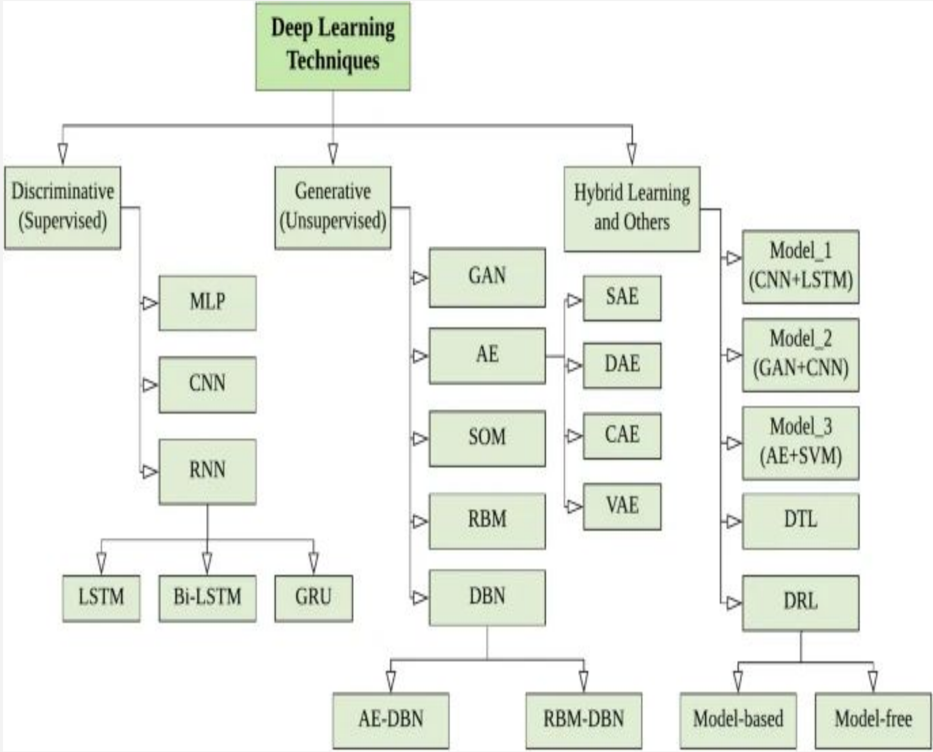


[Source: Performance Metrics for Classification Problems (Kaggle)]

What to do before you start: how to choose the right model

Most models are easy to implement

- ***Ask around***
- ***Read literature / blogs*** (e.g. machine learning mastery)
- ***Try on smaller version*** of your dataset

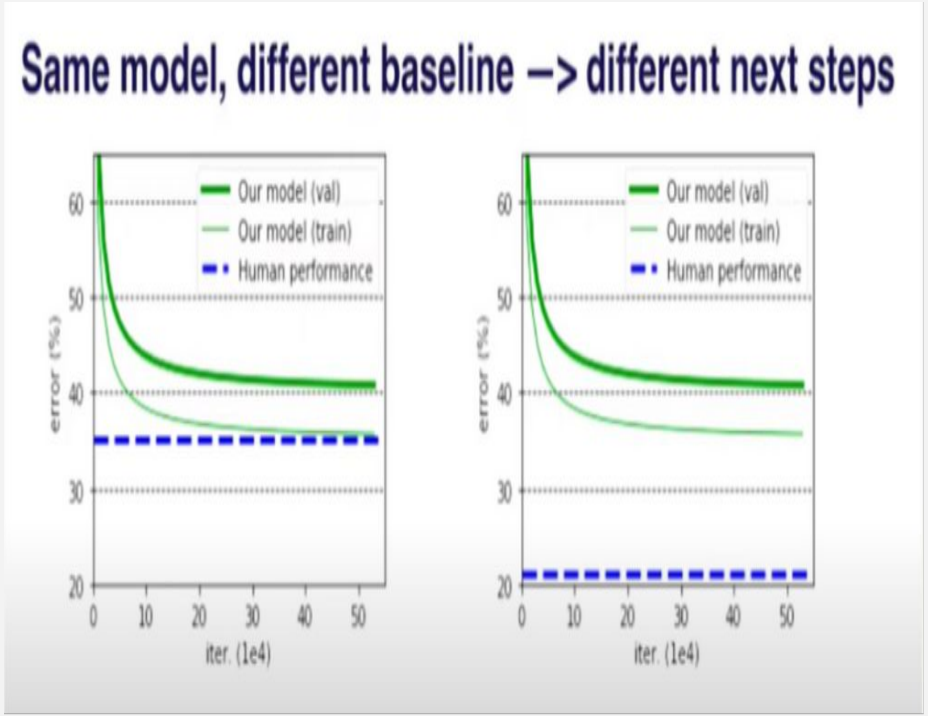


Deep Learning: A Comprehensive Overview

What to do before you start: how to choose the right baseline

Baselines are extremely important

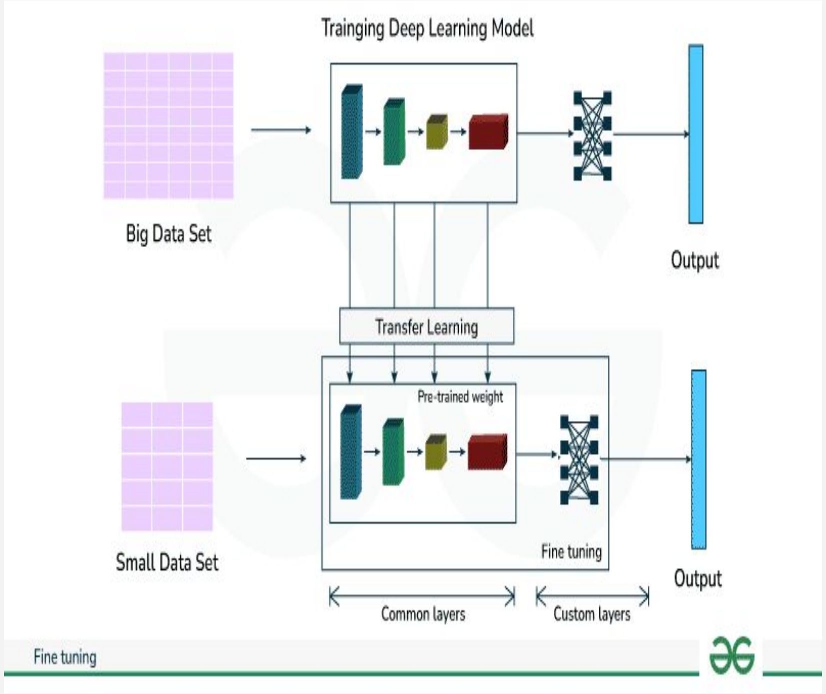
- ***Better than random*** (e.g. 50% accuracy for binary classification)
- ***DL must be better*** than simpler (ML) models
- ***DL could be better*** than humans



[Source: how to set a baseline (The Full Stack)]

What to do before you start: transfer learning

Larger, already trained models can be used for your problem



[Source: Geeks for Geeks]

Plenty of models are easily available on Pytorch

Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

Object Detection

The following object detection models are available, with or without pre-trained weights:

- Faster R-CNN
- FCOS
- RetinaNet
- SSD
- SSDlite

Semantic Segmentation

WARNING

The segmentation module is in Beta stage, and backward compatibility is not guaranteed.

The following semantic segmentation models are available, with or without pre-trained weights:

- DeepLabV3
- FCN
- LRASPP

Video Classification

WARNING

The video module is in Beta stage, and backward compatibility is not guaranteed.

The following video classification models are available, with or without pre-trained weights:

- Video MViT
- Video ResNet
- Video S3D
- Video SwinTransformer

TRY TO NOT REINVENT THE WHEEL!

Introduction to torchvision

STATISTICS WITHOUT BORDERS

Torchvision: why torchvision?

torchvision

This library is part of the PyTorch project. PyTorch is an open source machine learning framework.

- Easy to install and use
- Plenty of models available
- Plenty of tools (as in Pytorch)
- Seamless integration with video and audio tools

[Pytorch ecosystem]

Torchvision: MNIST

MNIST is a large handwritten digits database, commonly used for computer vision

- Published by the National Institute of Standards and Technology in 1994
- 6000 training and 10000 images
- 128x128 images grayscaled to 28x28

It has been used widely to test image recognition algorithm

Torchvision implements MNIST natively, along with several other important datasets

Source: wikipedia page

Built-in datasets

All datasets are subclasses of `torch.utils.data.Dataset` i.e, they have `__getitem__` and `__len__` methods implemented. Hence, they can all be passed to a `torch.utils.data.DataLoader` which can load multiple samples in parallel using `torch.multiprocessing` workers. For example:

```
imagenet_data = torchvision.datasets.ImageNet('path/to/imagenet_root/')
data_loader = torch.utils.data.DataLoader(imagenet_data,
                                          batch_size=4,
                                          shuffle=True,
                                          num_workers=args.nThreads)
```

All the datasets have almost similar API. They all have two common arguments: `transform` and `target_transform` to transform the input and target respectively. You can also create your own datasets using the provided base classes.

Image classification

`Caltech101`(root[, target_type, transform, ...])	Caltech 101 Dataset.
`Caltech256`(root[, transform, ...])	Caltech 256 Dataset.

Torchvision: load MNIST

With one-liner we can:

- Define image transformation
- Download (or load locally) MNIST
- Prepare dataset for training

```
transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081,))
        ])

batch_size = 10

trainset = torchvision.datasets.MNIST(root='./data', train=True,
                                        download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.MNIST(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
```

Dataset stores the samples and their corresponding labels,
and DataLoader wraps an iterable around the Dataset to enable easy access to the samples.

Torchvision: why torchvision?

Let us define a simple neural network:

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


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)


    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

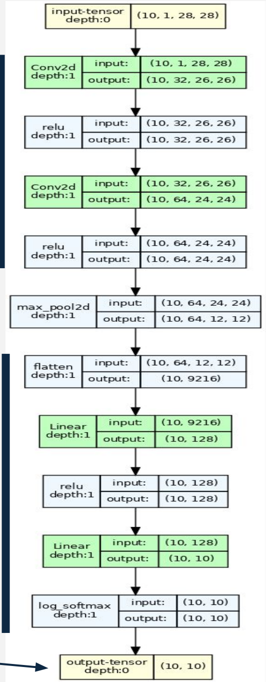
And visualize with torchview:

2 Convolutions

Pooling (reduce dimensionality)

Flatten data

Compute class probabilities



Moder neural networks architectures can be complicated (not necessarily better)

Torchvision: criterion and optimizer

Choose training criterion and optimizer

How do we choose the best model?

```
criterion = nn.CrossEntropyLoss()
```

Loss Functions

nn.L1Loss	Creates a criterion that measures the mean absolute error (MAE) between each element in the input x and target y.
nn.MSELoss	Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y.
nn.CrossEntropyLoss	This criterion computes the cross entropy loss between input logits and target.
nn.CTCLoss	The Connectionist Temporal Classification loss.
nn.NLLLoss	The negative log likelihood loss.
nn.PoissonNLLLoss	Negative log likelihood loss with Poisson distribution of target.
nn.GaussianNLLLoss	Gaussian negative log likelihood loss.

How do we find the best model?

```
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Algorithms

Adadelta	Implements Adadelta algorithm.
Adagrad	Implements Adagrad algorithm.
Adam	Implements Adam algorithm.
AdamW	Implements AdamW algorithm.
SparseAdam	SparseAdam implements a masked version of the Adam algorithm suitable for sparse gradients.
Adamax	Implements Adamax algorithm (a variant of Adam based on infinity norm).
ASGD	Implements Averaged Stochastic Gradient Descent.

A large family of both exist: results might depend drastically on criterion (less on optimizer)

Torchvision: train the model

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 0.208
[1,  4000] loss: 0.101
[1,  6000] loss: 0.072
[2,  2000] loss: 0.055
[2,  4000] loss: 0.053
[2,  6000] loss: 0.045
Finished Training
```

Looping several times on the dataset allows the fit to converge better

Pytorch accepts data as trainloader to optimizer training

Setting optimizer parameter to zero

Compute outputs with **current** model

Compute loss with **current** model

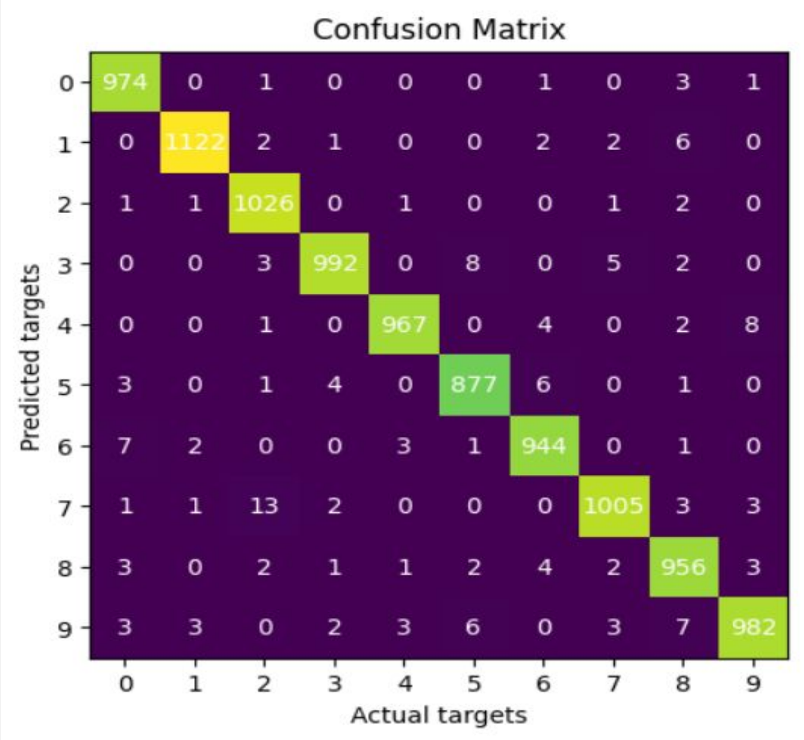
Pytorch's magic to tell the optimizer how much and in which direction it should go

Printing training loss every 2000 iterations

Torchvision: testing model

Compute results on test statistics and use them for the confusion matrix

```python
# Test function
def testing_accuracy(model, data_loader):
    model.eval()
    test_loss = 0
    device = 'cpu'

    y_pred = []
    y_actu = []
    with torch.no_grad():
        for data, target in data_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            y_pred.extend(torch.flatten(pred).tolist())
            y_actu.extend(target.tolist())

    y_pred = pd.Series(y_pred, name='Actual')
    y_actu = pd.Series(y_actu, name='Predicted')
    cm = pd.crosstab(y_actu, y_pred)
    correct = sum([cm.iloc[i,i] for i in range(len(cm))])

    test_loss /= len(data_loader.dataset)
    accuracy = 100*correct/len(data_loader.dataset)

    return(test_loss, accuracy, cm)
```



Torchvision: save and load models

Our model can be saved and loaded easily

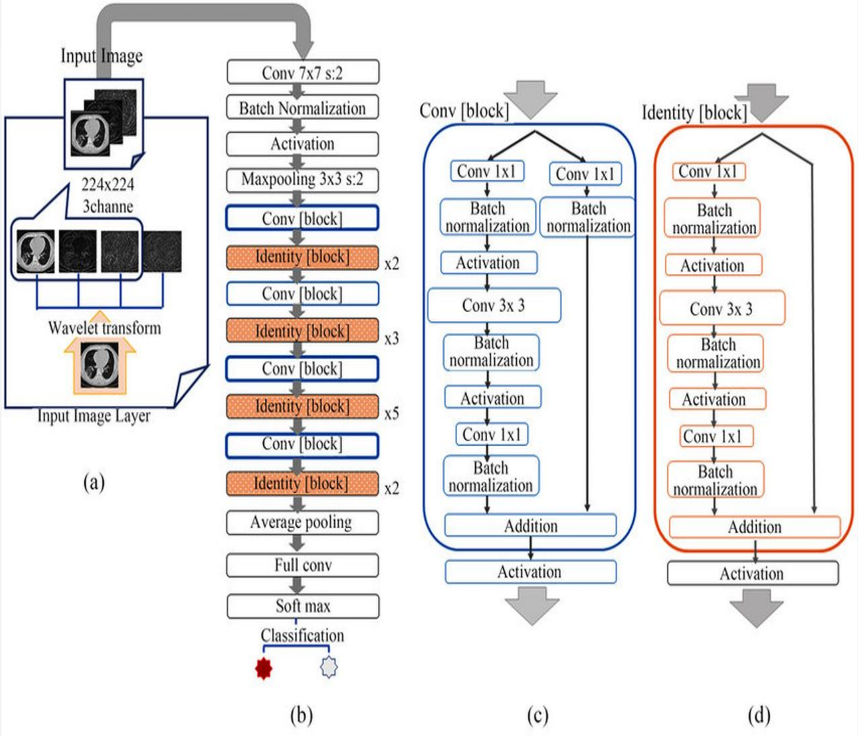
```
PATH = './my_minst_model_net.pth'
torch.save(net.state_dict(), PATH)
```

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

Large models can be loaded and modified too

```
resnet =  models.resnet50(pretrained=True)

def change_layers(model):
    model.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    model.fc = nn.Linear(2048, 10, bias=True)
    return model

change_layers(resnet)
```

Out of the box accuracy is **8%** …

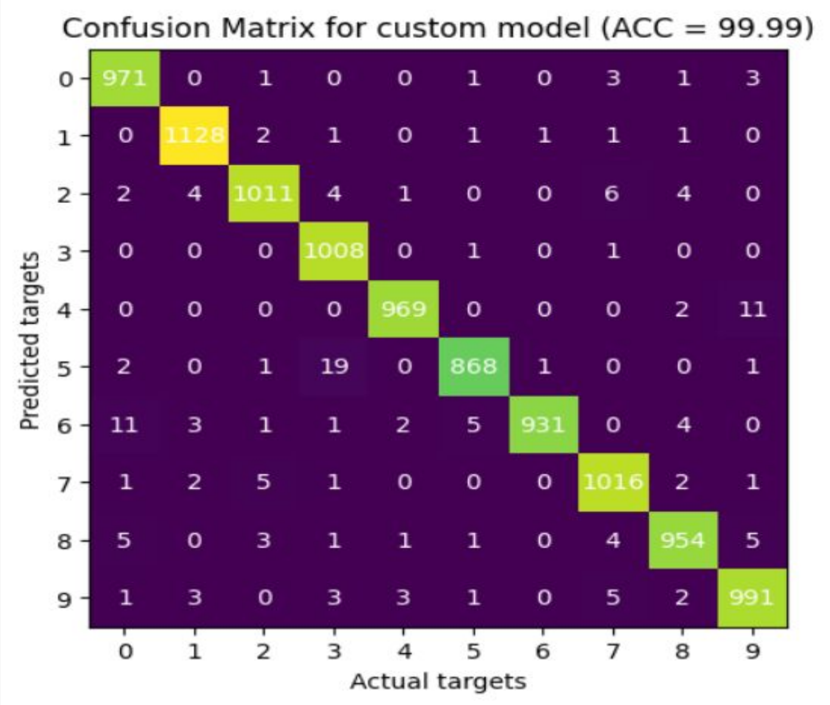


[Original architecture of ResNet50. Source: Wisdom ML]

Torchvision: save and load models

Fine tuning is needed to achieve better performances

```
#FINE TUNING

optimizer_resnet = optim.SGD(resnet.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer_resnet.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0
```



Slightly better acc. but model almost 25 bigger

Opening (a bit) the blackbox

STATISTICS WITHOUT BORDERS

Opening the black box: what is the network really learning

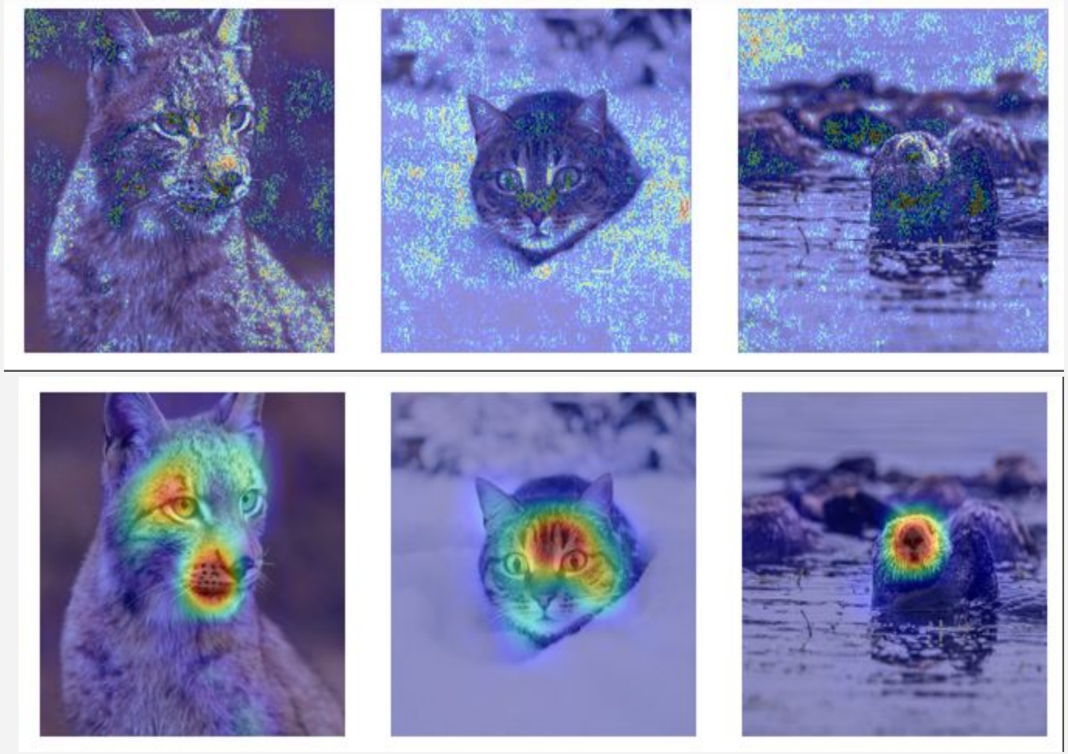
Saliency methods can be used to understand better deep learning results:

1. Perform forward pass
2. Compute gradient of class score with respect input pixels

$$E_{grad}(I_0) = \frac{\delta S_c}{\delta I}|_{I=I_0}$$

3. Visualize the gradients

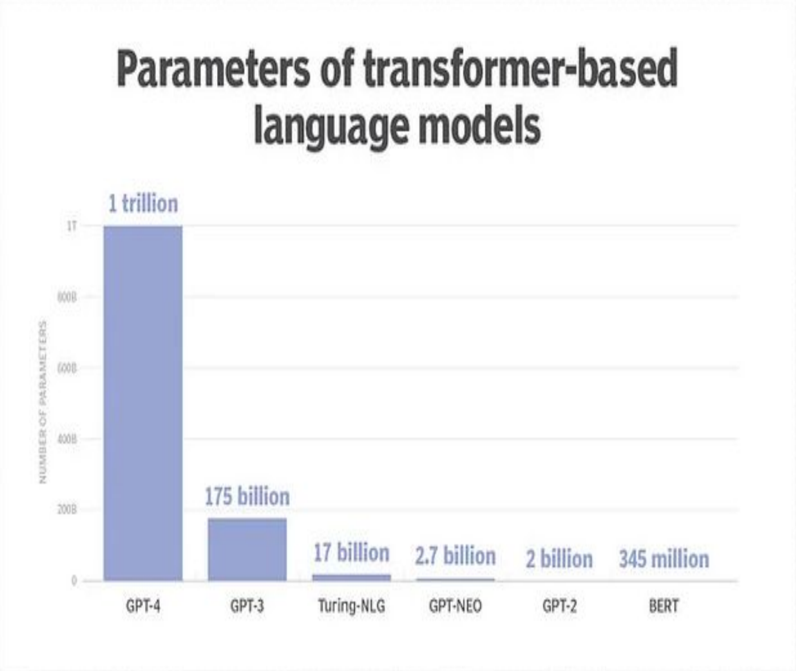
DL is learning differently than us!



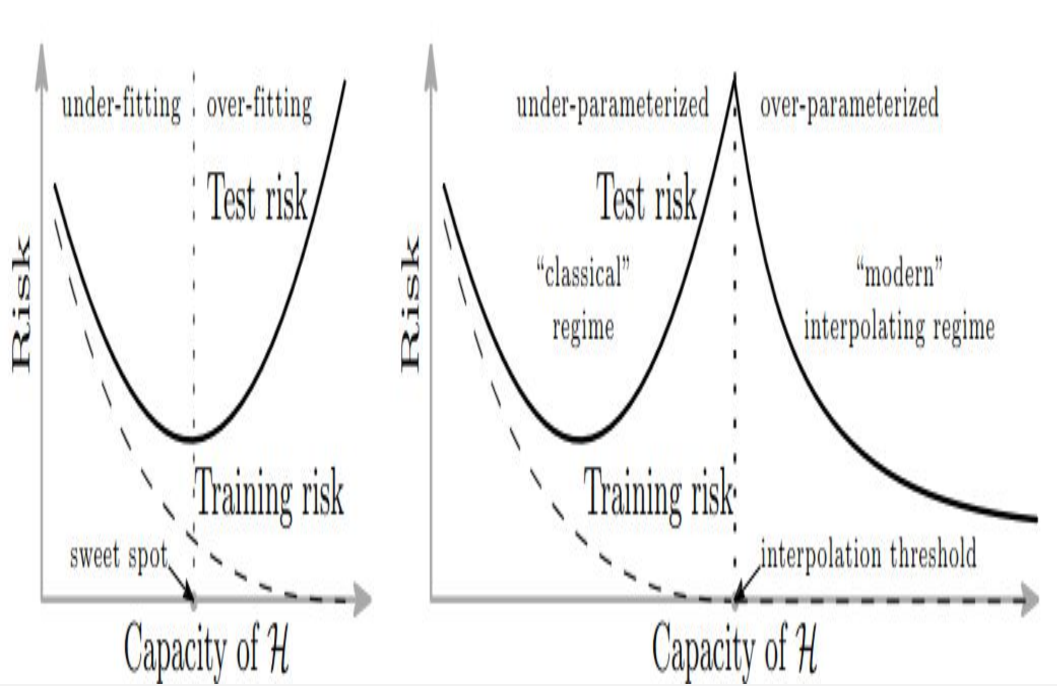
[Salency vs Solob methods (source Xplique documentation)]

Opening the black box: why so many parameters

Billions of parameter for LLMs

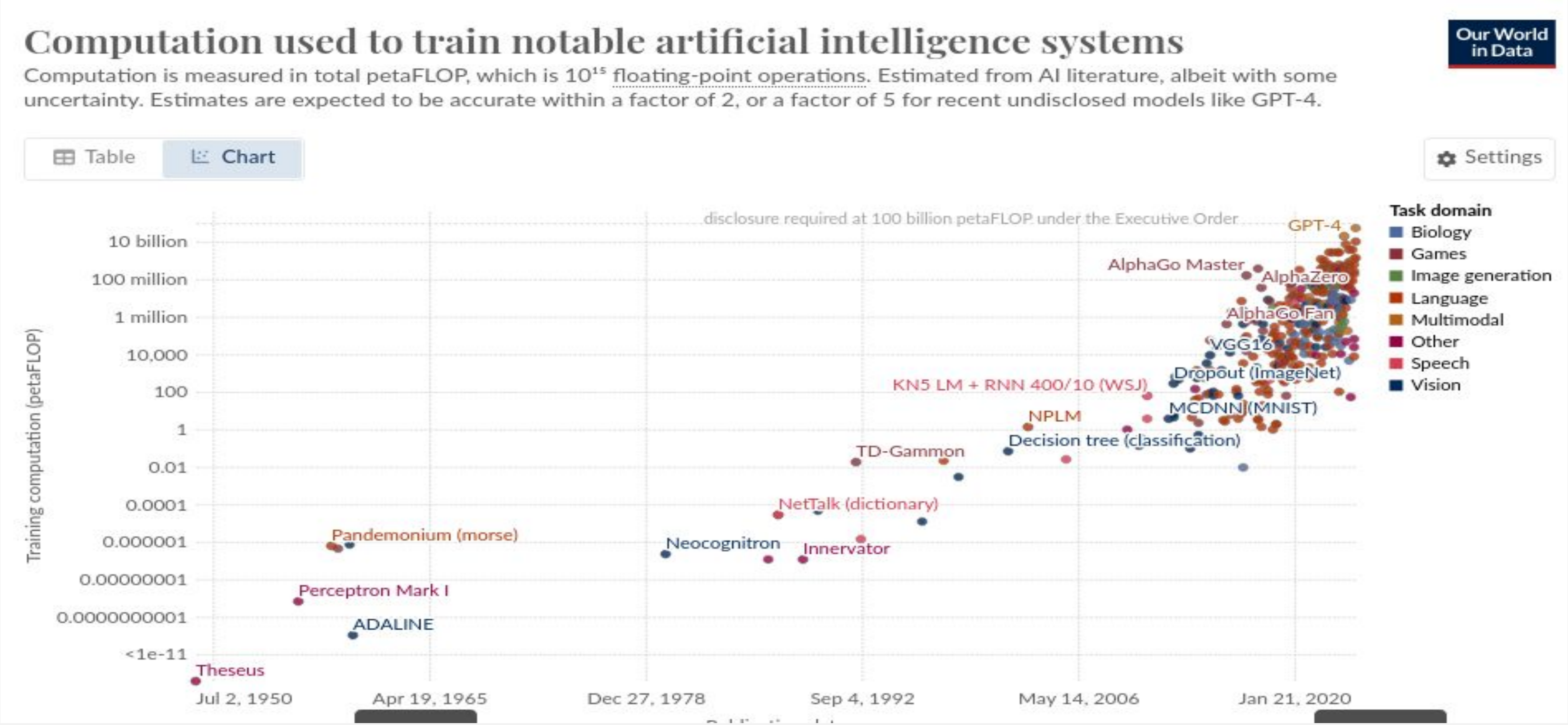


[Classical vs DL risk curve]



DL is overfit but acts differently than standard ML models: nobody knows why…

Opening the black box: why so many parameters (energy)



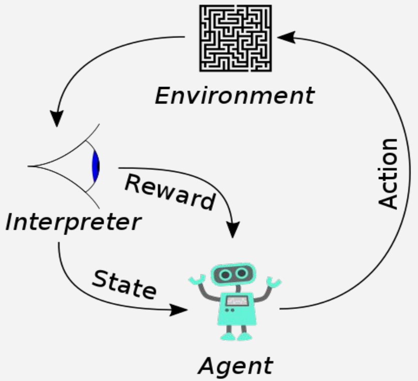
DL is particularly computationally expensive and becoming more so

Opening the black box: ML vs DL

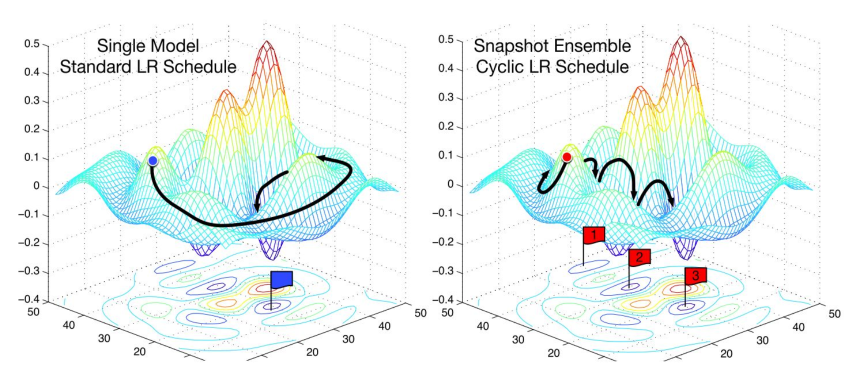
	Machine Learning	Deep Learning
Data Volume	Fewer, structured data	More, structured data
Computational Cost	Faster, lighter model	Very expensive training
Feature engineering	Explicitly done by humans	Often DL handles automatically
Interpretability	Usually easi(er)	Extremely hard to interpret
Applications examples	Customer segmentation, Recommender systems,Fraud detection, …	Computer Vision, Natural Language Processing, …

The differences are blurry and DL can be used for all applications

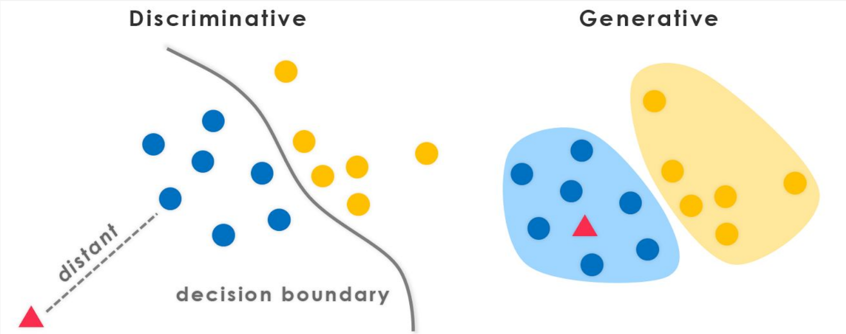
Opening the black box: topics we could not possibly cover



[Reinforcement learning](#)



[Role of optimizers](#)



[Generative AI (deep learning)](#)

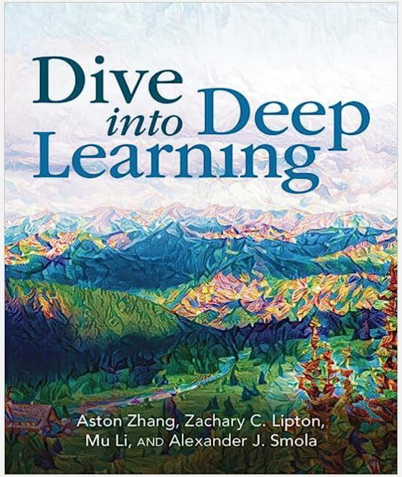
Conclusions and further reading

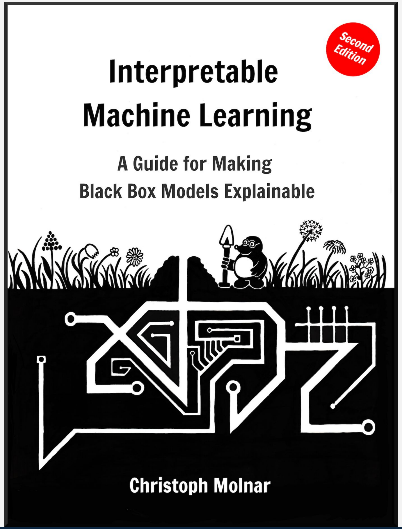
STATISTICS WITHOUT BORDERS

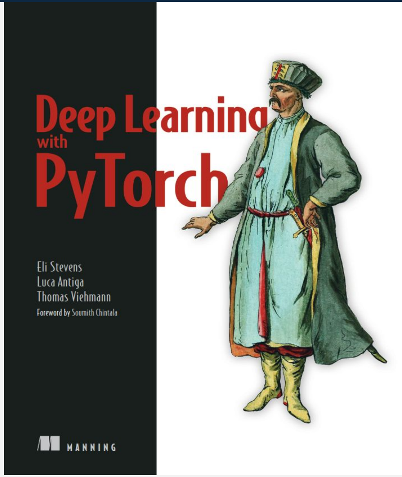
Conclusions

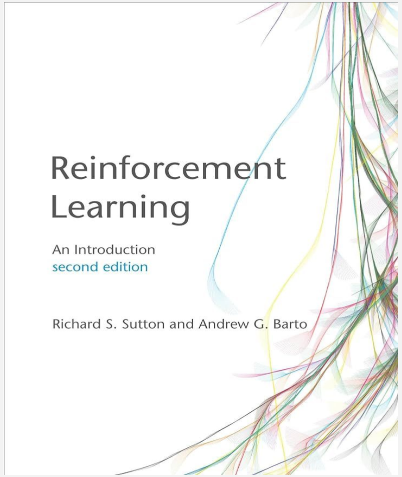
- Deep learning is a powerful and “easy-to-use” tool: please don’t reinvent the wheel!
- Deep learning should be used with care: complex, unstructured data work better
- Deep learning usage exploded but there are still a lot of unknowns
- Pytorch (and its sister libraries) is a powerful tool
- Deep learning is still a black box: use at your own risk!
- Reinforcement learning, Generative AI, …

Further reading: free resources









Presentation of the demonstration

STATISTICS WITHOUT BORDERS

Presentation of the demonstration: up to you now!

- Install torchvision on your PC
- Load and explore MNIST dataset
- Built your first CNN
- Train your first CNN
- Explore your CNN results
- Experiments with different architectures
- Load pretrained network
- Fine tune a pretrained network and compare
- Is my network overtrained? Train-Test loss
- …
- **WORK IN PROGRESS**

Don't hesitate to contact me!

deep-learning-II-lagos-lectures

Luca Alberto Rizzo

luca.alb.rizzo@gmail.com

Thank you for your attention

STATISTICS WITHOUT BORDERS

SWB is grateful for the partnership of University of Lagos Department of Statistics

https://StatisticsWithoutBorders.org

StatisticsWithoutBorders@gmail.com

STATISTICS WITHOUT BORDERS