EURO

Open Source Software for Machine Learning Projects

Joseph Pareti joepareti54@gmail.com https://www.joepareti54-ai.com/

- **Open-Source Software for Machine Learning Projects**
  - *Please note that this is not an exhaustive presentation of any open-source software products. I have put together these slides for you based on my own experimentation. Some very important open-source tools for machine learning may not be shown, or are just mentioned in the talk.*

Outline

- Scikit Learn: a range of supervised and unsupervised learning tools
- Pytorch and a sample model for classification
- Using existing networks, CNNS, Transfer Learning
- Recurrent Neural Networks
- Engineering Applications
- HPC for AI: how to speed up large model training
- Other Opens Source software
- Backup Slides

Scikit-learn basic usage and design principles

EURO

```
from sklearn.ensemble import
RandomForestClassifier



rf = RandomForestClassifier()

rf_model = rf.fit(X_train, y_train)


rf_predictions =
rf_model.predict(X_test)


It appears simple, yet most of the
work goes into data preprocessing,
cleaning, splitting the data into
train and test datasets
```

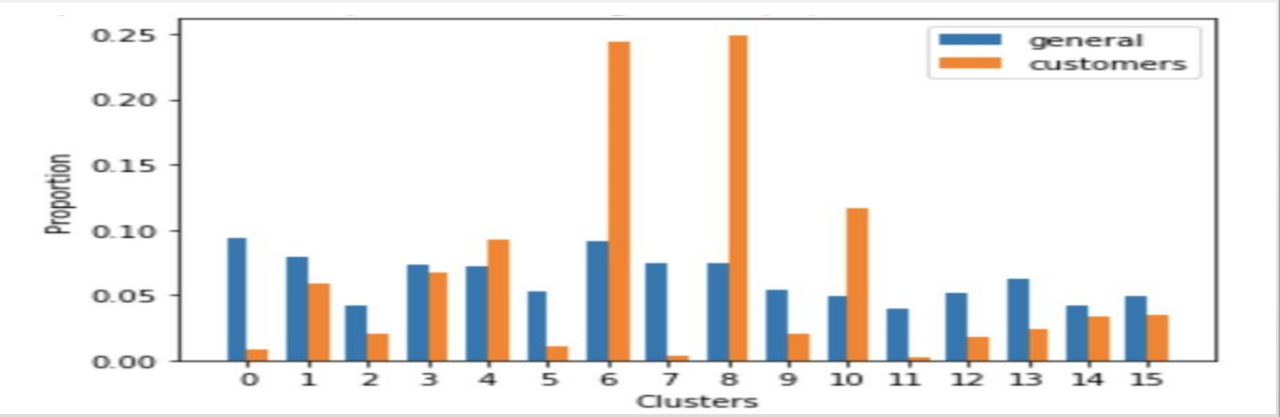
- Design Principles
  - All objects share a consistent interface
  - Objects that estimate some parameters are estimators, using the fit() method
  - Transformers transform a dataset and use the transform() method
  - Predictors use the predict() method and return a dataset of predictions
  - Estimators hyperparameters are accessible via an interface, e.g. imputer.strategy
  - Pipelines , reasonable defaults

My Projects implemented with Scikit-Learn

EURO

- Supervised Learning : *Finding Donors for CharityML*
  - *Input: US Census Data*
  - *Goal: predict whether an individual makes more than $50,000*
  - *Data pre-processing, scaling, splitting into training and test set*
  - *Apply supervised learning algorithms: RandomForest Classifier, AdaBoost Classifier and more*
  - *Select the best model and fine tune it using GridSearchCV*

- Unsupervised Learning : *Identify Customer Segments*
  - *Input: (i) Demographic, and (ii) Customer file*
  - *Goal: Increase sales by customer segmentation*
  - *Reduce dimensionality using **Principal Components Analysis***
  - *Clustering using **k-means***



PyTorch: a framework for accelerating deep learning development

- Developed by Meta
- Here is the code explained in the next slides

PyTorch: Binary Classification (1 of 5)

jupyter test_pytorch_bin_class Last Checkpoint: vor 2 Stunden (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [1]:

Slide Type Slide

```
#
# joepareti54@gmail.com  May 3 2022
# binary classification : cancer, no cancer
# using a dataset in scikit learn
#
# the model class definition is similar to the linear regression
# but the last layer is a sigmoid function
```

In [2]:

Slide Type Sub-Slide

```
import numpy as np
import torch
from torch import nn, optim
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
#
```

In [3]:

Slide Type Sub-Slide

```
bc = datasets.load_breast_cancer()
X, y = bc.data, bc.target
```

PyTorch: Binary Classification (2 of 5)

In [4]:

Slide Type: Sub-Slide

```
print('input shape ',X.shape)
print('label shape ',y.shape)
n_samples = X.shape[0]
n_features = X.shape[1]
print('n_samples ', n_samples)
print('n_features ', n_features)
```

```
input shape  (569, 30)
label shape  (569,)
n_samples  569
n_features  30
```

In [5]:

Slide Type: Slide

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print('input shape trn ',X_train.shape)
print('input shape tst ',X_test.shape)
```

```
input shape trn  (455, 30)
input shape tst  (114, 30)
```

In [6]:

Slide Type:

```
# apply scaling
scaler = StandardScaler().fit(X_train)
X_TRAIN = scaler.transform(X_train)
X_TEST  = scaler.transform(X_test)
```

PyTorch: Binary Classification (3 of 5)

In [7]:

```
#convert data to pytorch tensors
#XX_train = torch.from_numpy(X_TRAIN, dtype=np.float32)
XX_train = torch.from_numpy(X_TRAIN.astype(np.float32))
XX_test  = torch.from_numpy(X_TEST.astype(np.float32))
print('shape of train tensor ',XX_train.shape)
print('shape of test tensor ',XX_test.shape)
YY_train = torch.from_numpy(y_train.astype(np.float32))
YY_test = torch.from_numpy(y_test.astype(np.float32))
```

```
shape of train tensor  torch.Size([455, 30])
shape of test tensor  torch.Size([114, 30])
```

In [8]:

```
# in order to understand what input size and output size need to be for this case:
# https://stackoverflow.com/questions/54916135/what-is-the-class-definition-of-nn-linear-in-pytorch
#
class BinaryClassification(nn.Module):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.linear = nn.Linear(input_size, output_size)
    def forward(self, x):
        y_pred = self.linear(x)
        y_pred = torch.sigmoid(y_pred)
        return y_pred
```

PyTorch: Binary Classification (4 of 5)

In [14]:

```
model = BinaryClassification(XX_train.shape[1], 1)
loss = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 50000
```

In [15]:

```
yy_train = YY_train.view(-1,1)
yy_test  = YY_test.view(-1, 1)
```

In [16]:

```
for i in range(epochs):
    optimizer.zero_grad()
    output = model(XX_train)
    LOSS = loss(output, yy_train)
#    print(LOSS)
    LOSS.backward()
    optimizer.step()
    if i % 10000 == 0:
        print('epoch', i, 'loss', LOSS.item())
```

```
epoch 0 loss 0.4209327697753906
epoch 10000 loss 0.007230205461382866
epoch 20000 loss 0.0026007883716374636
epoch 30000 loss 0.002217780100181699
epoch 40000 loss 0.0021989112719893456
```

PyTorch: Binary Classification (5 of 5)

In [17]:

Slide Type

```
#test model
y_hat = model(XX_test)
for i in range(10):
    print(y_hat[i].item(),yy_test[i].item()  )
```

```
1.0 1.0
0.0 0.0
0.0 0.0
1.0 1.0
1.0 1.0
0.0 0.0
0.0 0.0
8.306769272851408e-27 0.0
0.9759547114372253 1.0
1.0 1.0
```

In [18]:

Slide Type

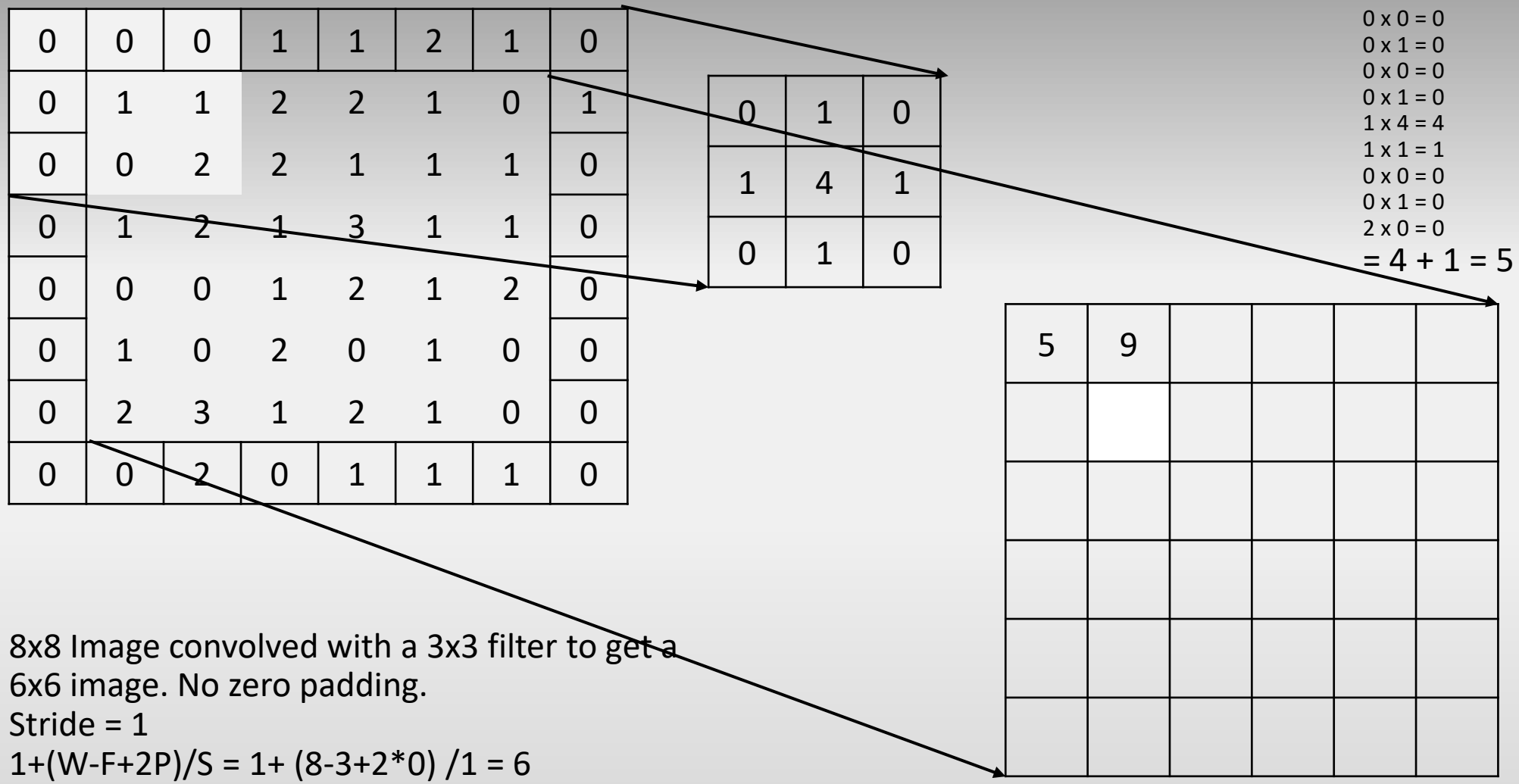
```
# compute accuracy
with torch.no_grad():

    a = y_hat.round()
    A = (a == yy_test)
    acc = A.sum() / float(YY_test.shape[0])
    print(acc)
```

```
tensor(0.9561)
```

Computer Vision, Image Classification, Convolution Neural Networks, Transfer Learning

Explaining 2D convolution



8x8 Image convolved with a 3x3 filter to get a 6x6 image. No zero padding.
Stride = 1
1+(W-F+2P)/S = 1+ (8-3+2*0) /1 = 6

Custom Image Classifier

- Import pre-trained model, VGG NN from torchvision.models
- Define an untrained feed-forward network as a classifier replacing the existing one
- Train the classifier layers using the pre-trained network to get the features
- Track the loss and accuracy on the validation set to determine the best hyperparameters

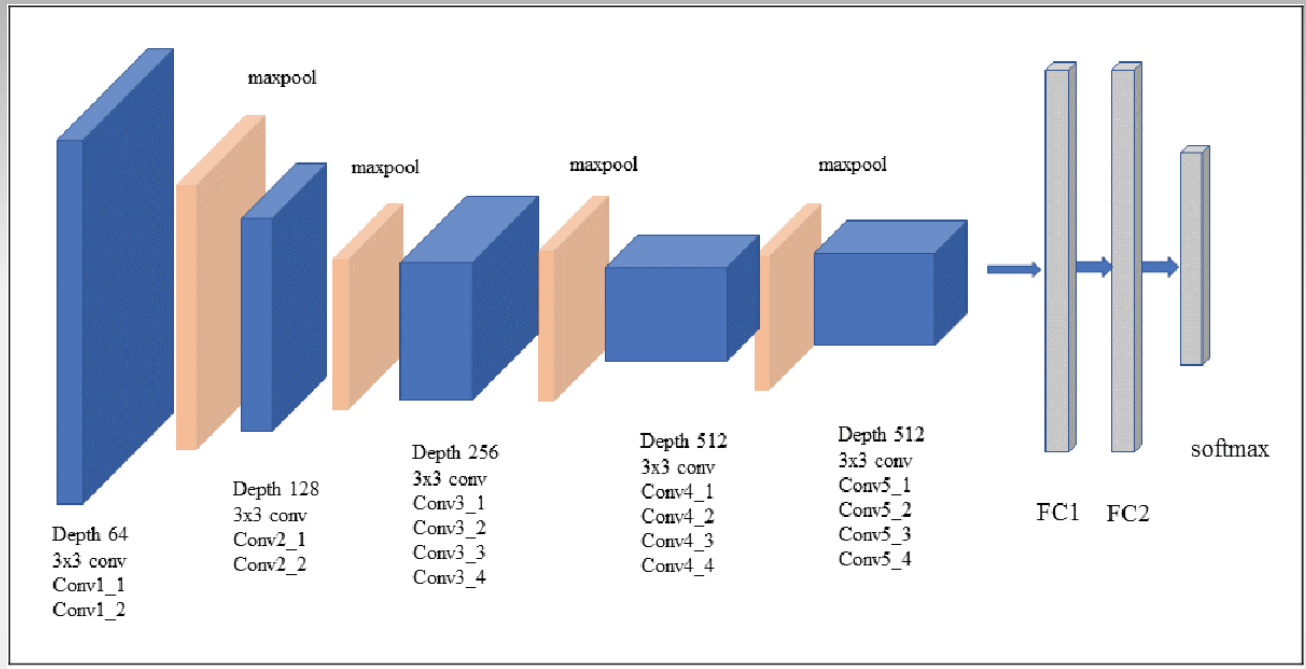


Fig. 3. VGG-19 network architecture

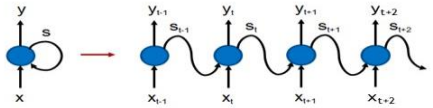
Recurrent Neural Networks, LSTM, Keras

Recurrent networks have an internal state that can represent context information. They keep information about past inputs for an amount of time that is not fixed, but rather depends on its weight and on the input data

RNNs and Long Short Term Memory

Recurrent Neural Networks

- RNNs are created to solve the memory-persistence issue found in conventional feed forward networks.
- RNNs maintain a loop that allows the information to be persistent in the network.



- At each time step t, the network receives input Xt and outputs a value of yt. The left side of the figure shows an RNN cell with input X and output y. There is also a loop-back of the state, which represents the memory. At a given timestep t, the output also depends on all previous timesteps $t-1, t-2, \ldots, 0$.

LSTM MODEL

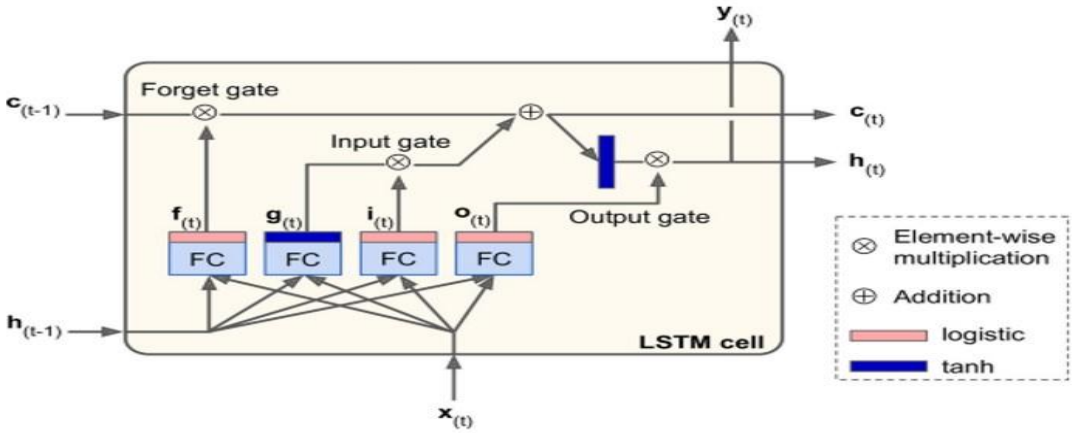
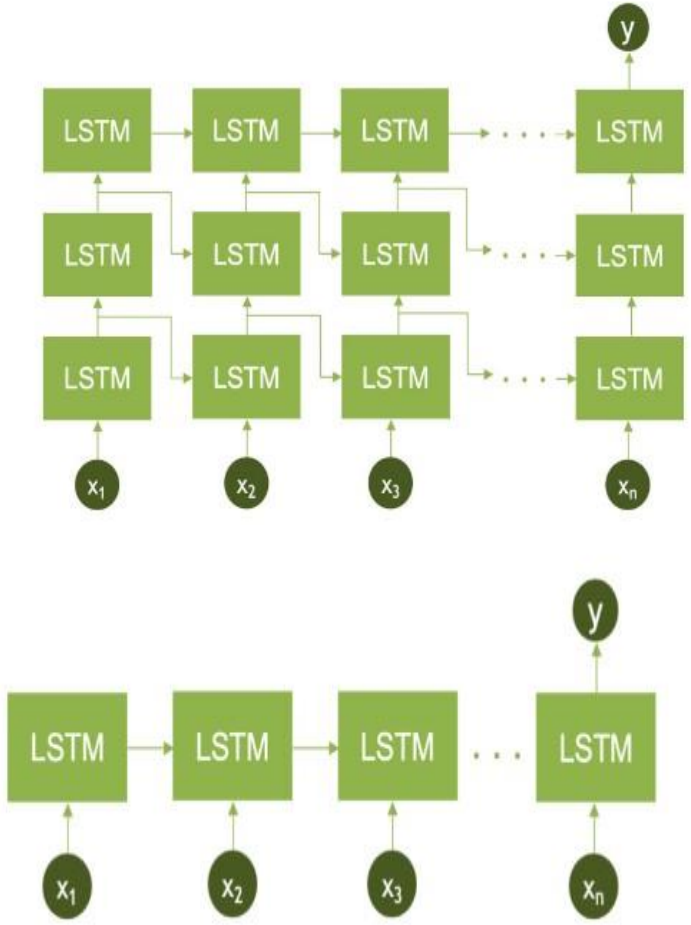


Image from book: Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems by Aurélien Géron

STACKED LSTMS



Use Cases for RNN and LSTM: time series, seq2seq, forecasting

EURO

Seq2seq tutorial built on an LSTM model and implemented in Keras

In that tutorial, I will **generate** X and y parallel datasets such that **y sequence** will be **the reverse of** the given **X sequence**. For example,

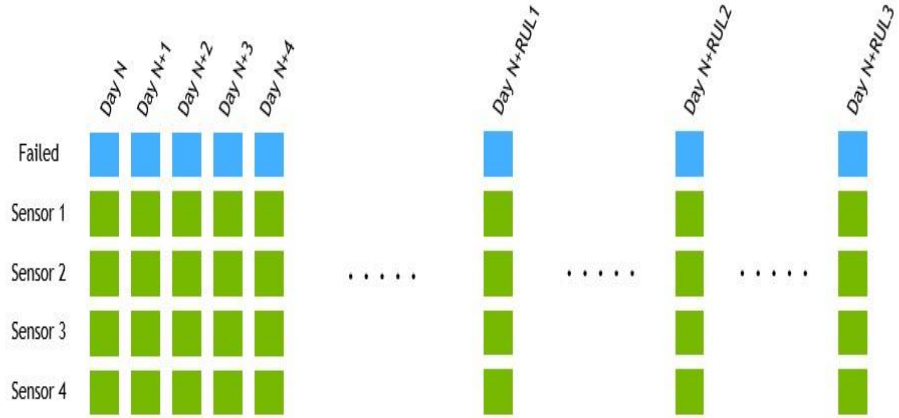
- **Given sequence** X[i] length of 4:

  X[i]=[3, 2, 9, 1]
- **Output sequence** (y[i]) is the **reversed** input sequence (X[i])

  y[i]=[1, 9, 2, 3]

I will call this parallel dataset: "**the reversed sequence problem**"

For Predictive Maintenance, create failed record sequences; from failure point back lookahead days



Keras: make programming with Tensorflow easier

EURO

KERAS

Keras: an API for specifying & training differentiable programs

KERAS – CODE SIMPLICITY

```
# load model definition API
from keras.models import Sequential
model = Sequential()

# load appropriate Keras layers and add to model
from keras.layers import Dense
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

# load appropriate optimizer and compile model
from keras.optimizers import SGD
optimizer1 = SGD(lr=0.01, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=optimizer1, metrics=['accuracy'])
```

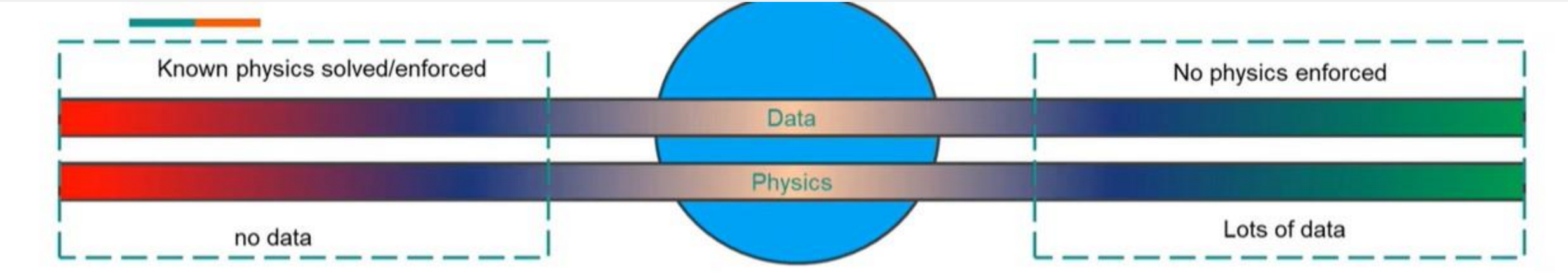

KERAS – CODE SIMPLICITY

```
# visualize model
model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                6464
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 7,114
Trainable params: 7,114
Non-trainable params: 0
_________________________________________________________________
```

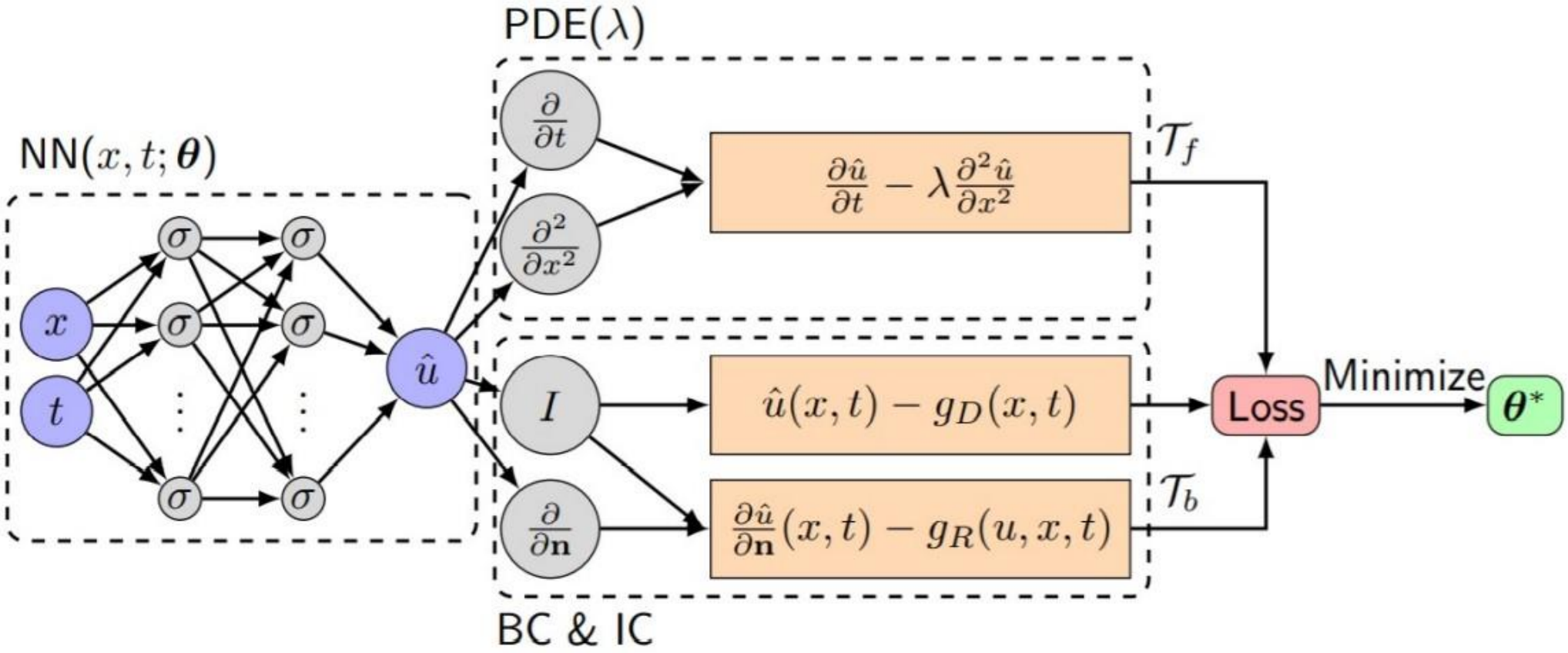
Machine Learning for Engineering Applications

EURO

- Faster time-to-solution of complex numerical simulations than using partial differential equations solvers, e.g. for CFD, FEM
- Learning from data, surrogate models
- Physics Informed Neural Networks



Physics Informed Neural Networks



Physics Informed Neural Networks

PINN: input and output

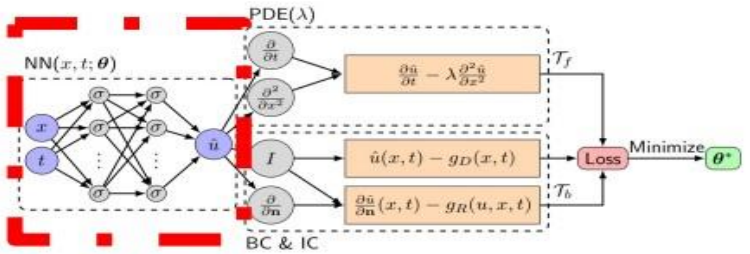
Inputs:
time domain coordinate and space domain coordinate.

- *Ex: I want to have the solution of the following PDE equation, in the range of* $-1 \leq x \leq 1$ *, and* $0 \leq t \leq 1$:

$$\frac{\partial u}{\partial t} = \lambda \frac{\partial^2 u}{\partial x^2}$$

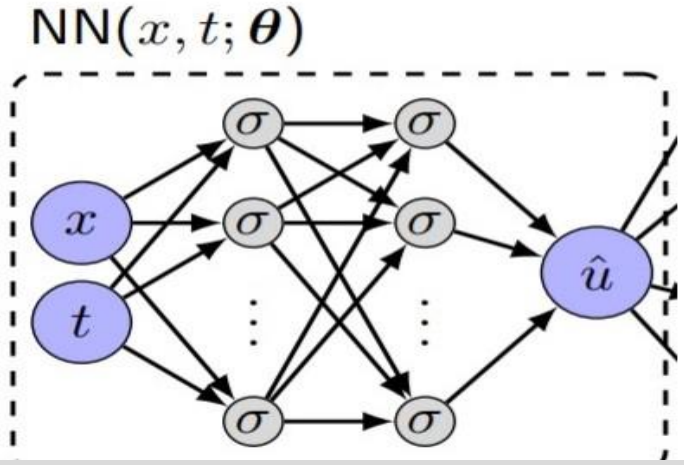
- *Inputs: (x,t), which* $-1 \leq x \leq 1$ *, and* $0 \leq t \leq 1$

Outputs:
$u(x,t)$ *of the equation*



4 keys:

1. Inputs and outputs
2. Minimize the loss
3. Backpropagation
4. Find optimal parameters



Physics Informed Neural Networks

PINN: loss function

To make the NN physical informed, the loss function is constituted with loss from the PDE equation T_f and loss from the condition equations T_b.

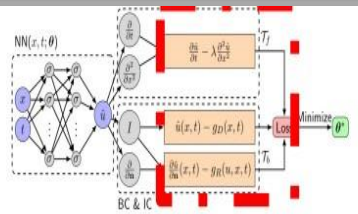
$$\mathcal{L}(\boldsymbol{\theta};\mathcal{T}) = w_f\mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f) + w_b\mathcal{L}_b(\boldsymbol{\theta};\mathcal{T}_b),$$

$$T_f = \frac{\partial u'}{\partial t} - \lambda\frac{\partial^2 u'}{\partial x^2}$$

$$T_b = u'(x,t) - g_D(x,t)$$

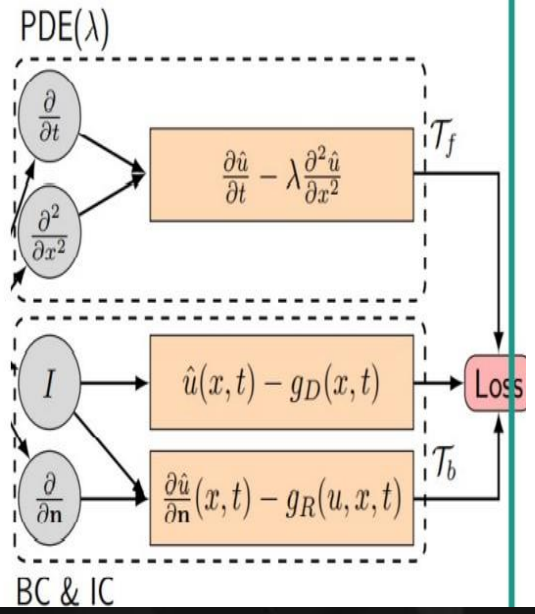
,where $u'$is the output of the nn

,g_D is the result for initial condition



4 keys:

1. Inputs and outputs
2. Minimize the loss
3. Backpropagation
4. Find optimal parameters



Get start from DeepXDE package

- We are using DeepXDE, an opensource package developed by Lu Lu.
- DeepXDE is based on a popular wide used package Tensorflow, which is written in Python.
- Dependency:
  - Pytorch
  - TensorFlow 1.x: TensorFlow>=2.2.0
  - TensorFlow 2.x: TensorFlow>=2.2.0 and TensorFlow Probability

DeepXDE requires TensorFlow to be installed. Then, you can install DeepXDE itself. If you use Python 2, you need to install DeepXDE using `pip`.

- Install the stable version with `pip`:

```
$ pip install deepxde
```

- Install the stable version with `conda`:

```
$ conda install -c conda-forge deepxde
```

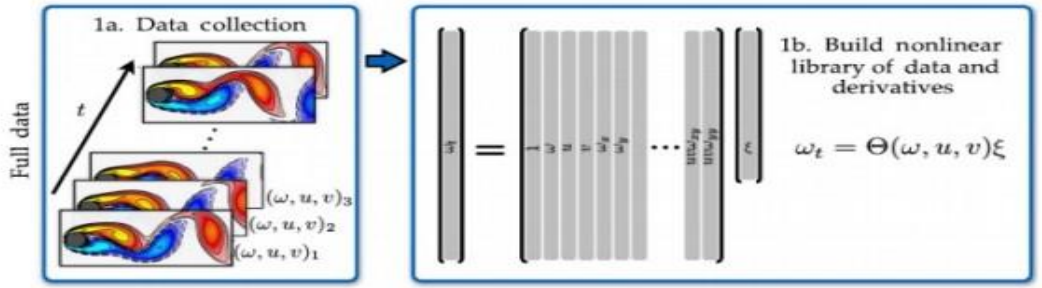
- For developers, you should clone the folder to your local machine and put it along with your project scripts:

```
$ git clone https://github.com/lululxvi/deepxde.git
```

Data driven Algorithms

Data driven discovery of PDEs

- More recent revival of these ideas before PINNs



Rudy, Brunton, Proctor, Kutz, 2017, Science Advances

Data-driven discovery of partial differential equations

Summary

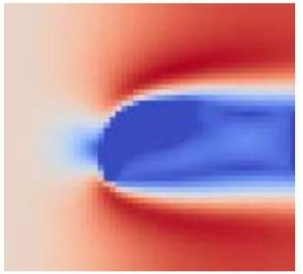
- Presented a method to infer the mathematical model from experimental data
- By trying to fit the experimental data to a large collection of differential and polynomial terms
- Used sparse regression techniques to come up with possible best fit

Machine Learning for Fluid Mechanics

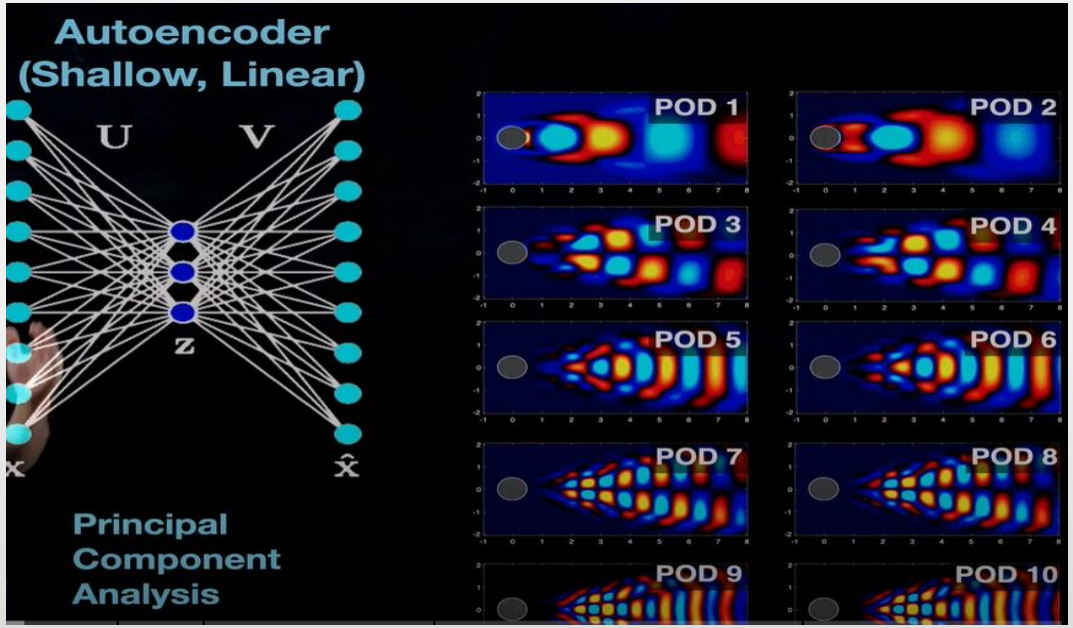
EURO

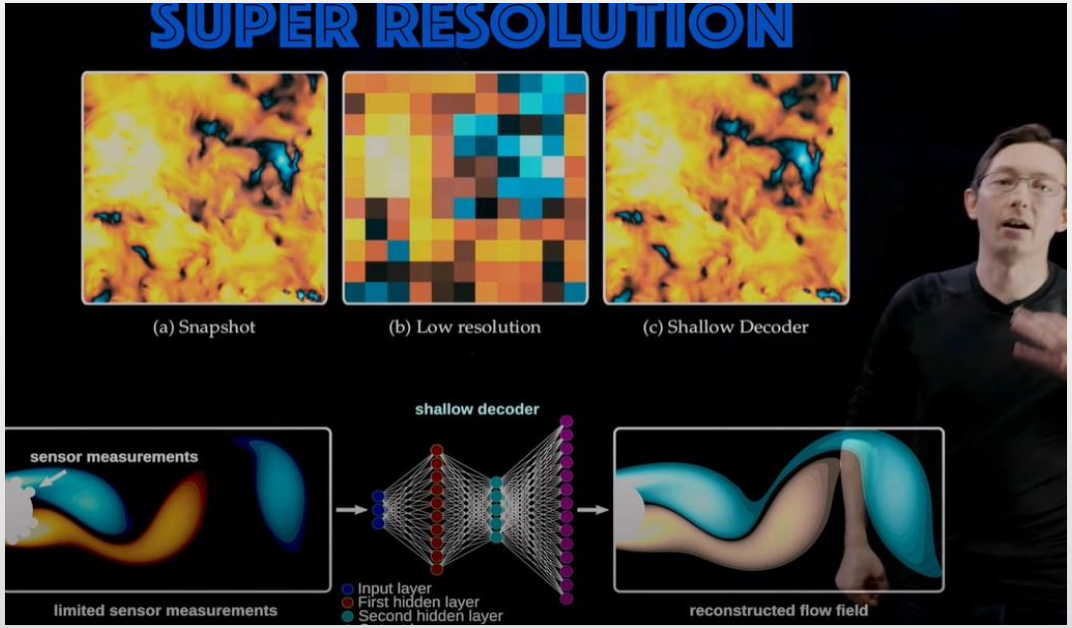
Deep Learning For Steady-State Fluid Flow Prediction

Brunton: Machine Learning for Fluid Mechanics



"The overhead of creating high volumes of samples can be effectively compensated by the high-performance containerized computing environment provided by UberCloud and Advania."





HPC algorithms and tools for AI

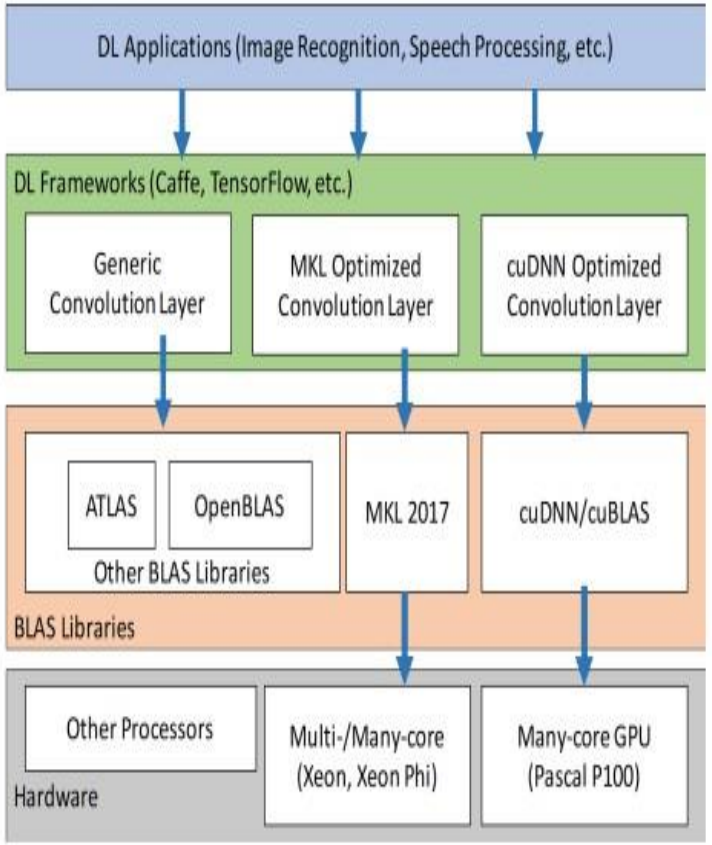
Training large Neural Networks is a lot similar as solving complex numerical simulations

Time-to-Solution for Deep Learning

EURO

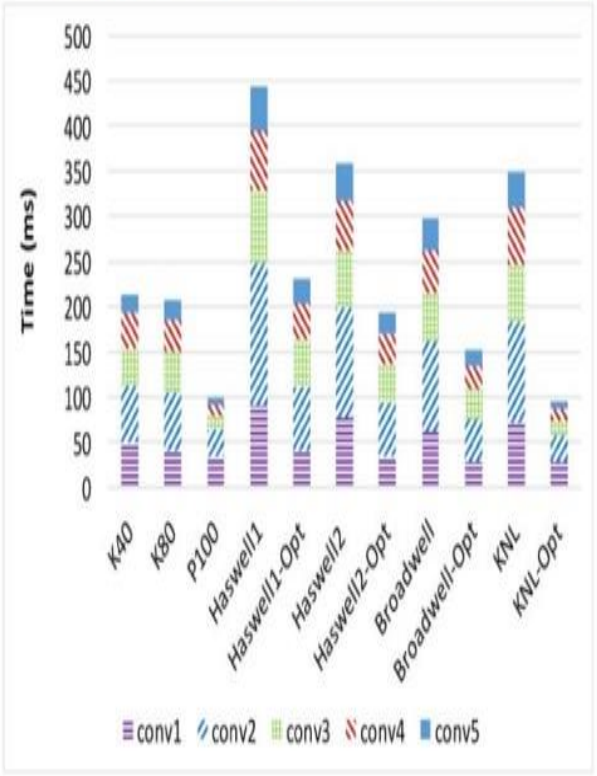
DL Frameworks and Underlying Libraries

- BLAS Libraries – the heart of math operations
  - Atlas/OpenBLAS
  - NVIDIA cuBlas
  - Intel Math Kernel Library (MKL)
- Most compute intensive layers are generally optimized for a specific hardware
  - E.g. Convolution Layer, Pooling Layer, etc.
- DNN Libraries – the heart of Convolutions!
  - NVIDIA cuDNN (already reached its 7th iteration – cudnn-v7)
  - Intel MKL-DNN – a promising development for CPU-based ML/DL training

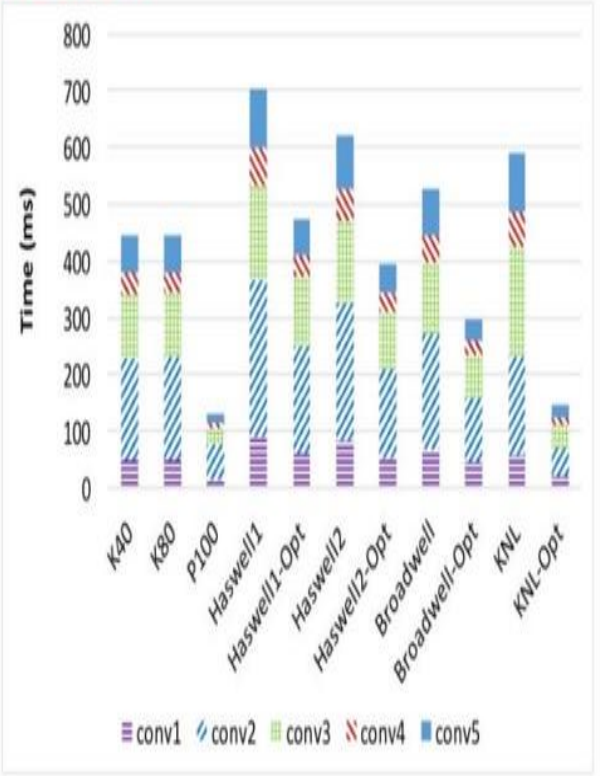


A. A. Awan, H. Subramoni, and Dhabaleswar K. Panda. "An In-depth Performance Characterization of CPU- and GPU-based DNN Training on Modern Architectures", In Proceedings of the Machine Learning on HPC Environments (MLHPC'17). ACM, New York, NY, USA, Article 8.

Where does the Performance come from?



(a) AlexNet: Forward Propagation

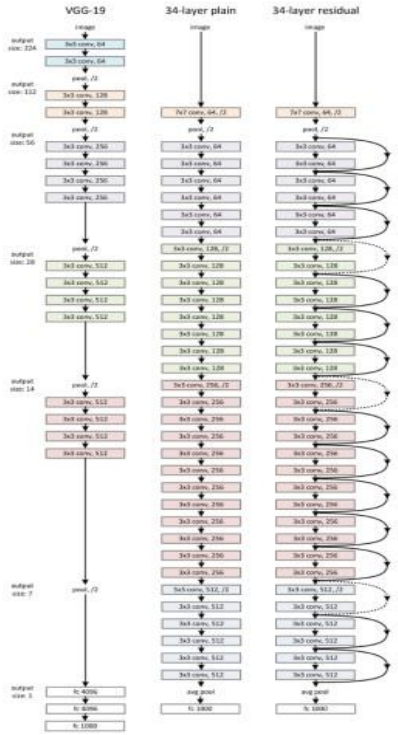


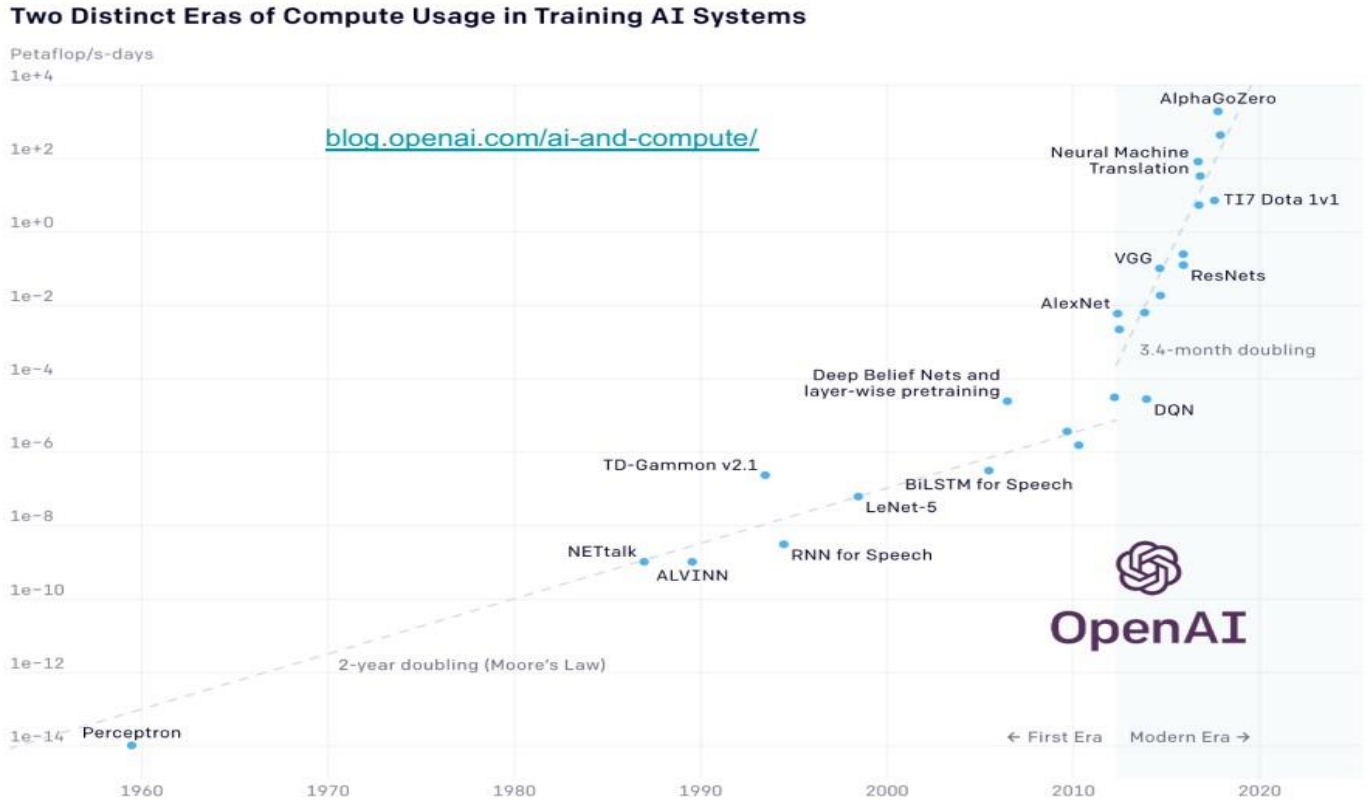
(b) AlexNet: Backward Propagation

- The full landscape: Forward and Backward Pass -- ***Faster Convolutions → Faster Training***

Large Neural Networks require parallel training

Why do we need to scale deep learning applications?





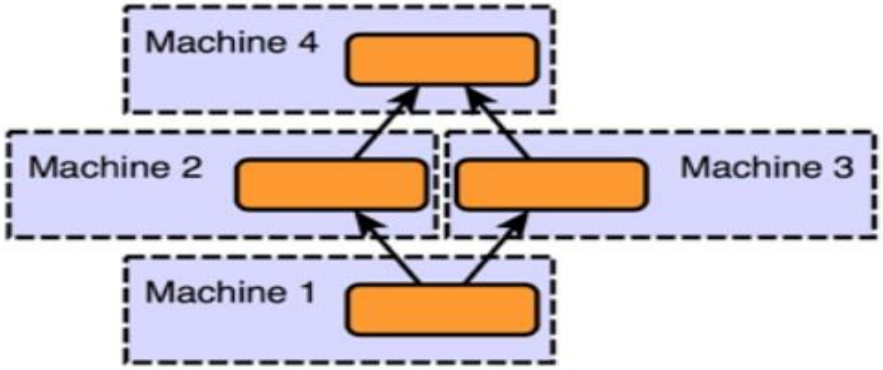
Models get bigger and more compute intensive as they tackle more complex tasks

"... total amount of compute, in petaflop/s-days, that was used to train selected results ... A petaflop/s-day (pfs-day) = ... 10^{15} neural net operations per second for one day, or a total of about 10^{20} operations." -- OpenAI Blog

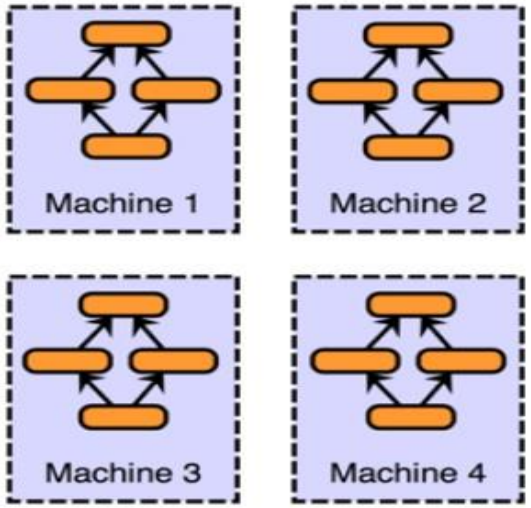
Parallelization algorithms for NN

Parallelization Strategies

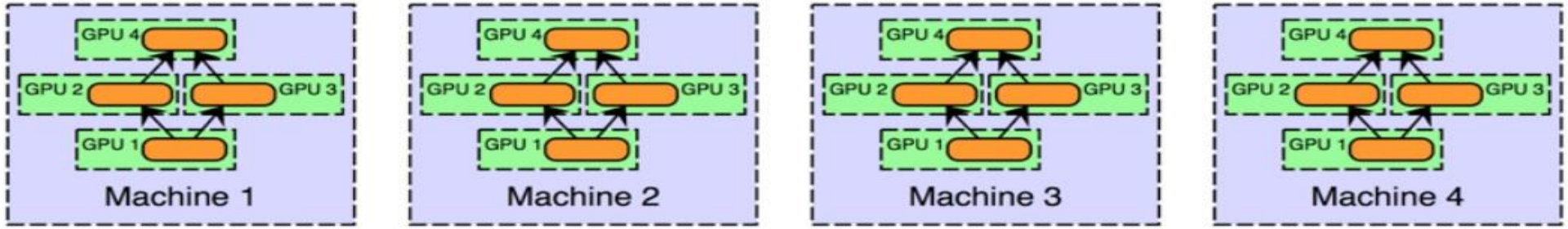
- Some parallelization strategies..
  - Data Parallelism or Model Parallelism
  - Hybrid Parallelism



Model Parallelism



Data Parallelism



Hybrid (Model and Data) Parallelism

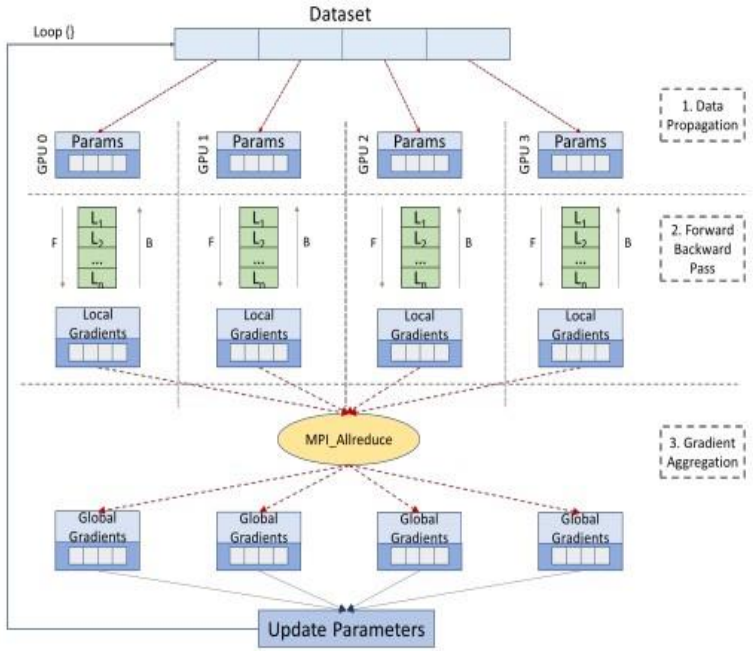
Courtesy: http://engineering.skymind.io/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks

Data Parallelism for Model Training

EURO

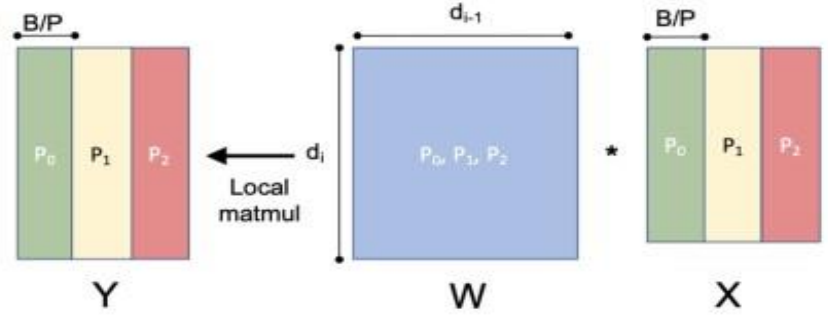
Data Parallelism and MPI Collectives

- **Step1:** Data Propagation
  - Distribute the Data among GPUs
- **Step2:** Forward Backward Pass
  - Perform forward pass and calculate the prediction
  - Calculate Error by comparing prediction with actual output
  - Perform backward pass and calculate gradients
- **Step3:** Gradient Aggregation
  - Call MPI_Allreduce to reduce the local gradients
  - Update parameters locally using global gradients



Forward Pass

- batches are distributed across ranks/workers
- weight matrices are replicated across ranks/workers
- local matrix multiplication
- no communication necessary



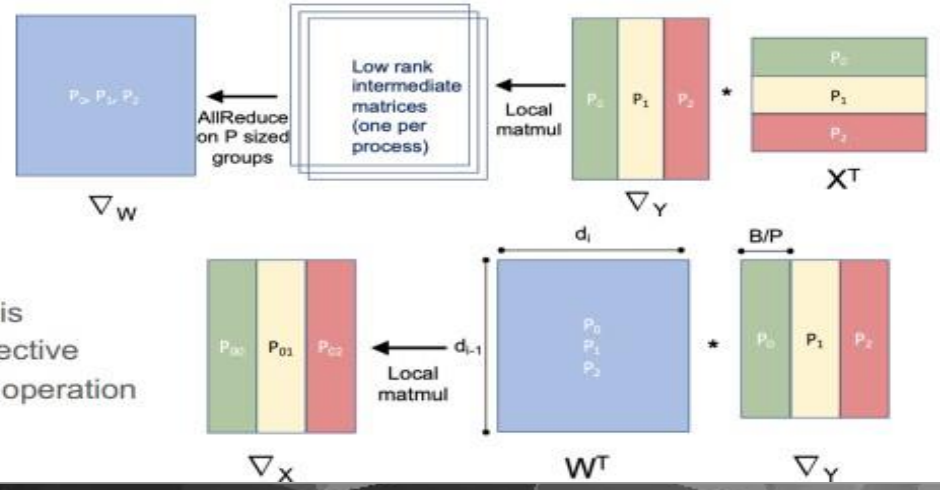
Backward Pass

$\nabla_Y = \partial L / \partial Y$

$\nabla_X = \partial L / \partial X$

$\nabla_W = \partial L / \partial W$

- only AllReduce is required as collective communication operation



Data Parallelism R&D

EURO

Synchronous Updates

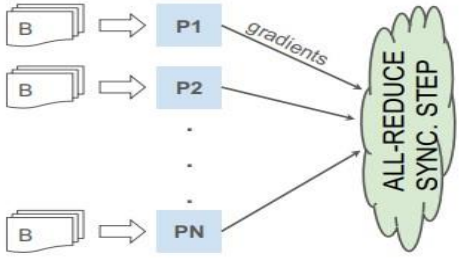
Gradients are computed locally and summed across nodes. Updates are propagated to all nodes

- stable convergence
- scaling is not optimal because all nodes have to wait for reduction to complete
- global (effective) batch size grows with number of nodes

Synchronous SGD, decentralized

Large-Batch Training (LBT), synchronous weak scaling

- applies to SGD-type algorithms
  - data batch per node. Model updates are computed independently
  - updates are collectively summed and applied to the local model

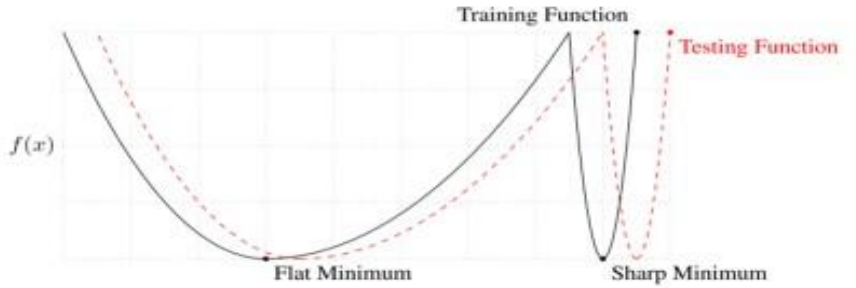


Local batch-size = B

Global batch-size = N * B

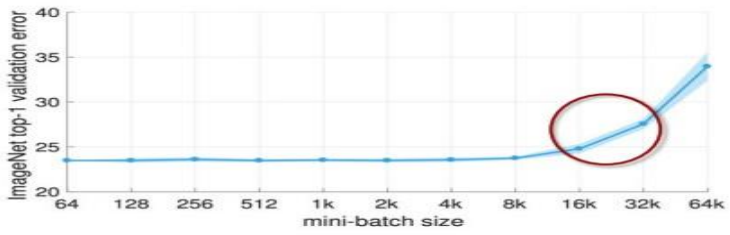
Explaining the generalization gap?

"... large-batch … converge to sharp minimizers of the training function … In contrast, small-batch methods converge to flat minimizers" -- Keskar et al, arXiv:1609.04836



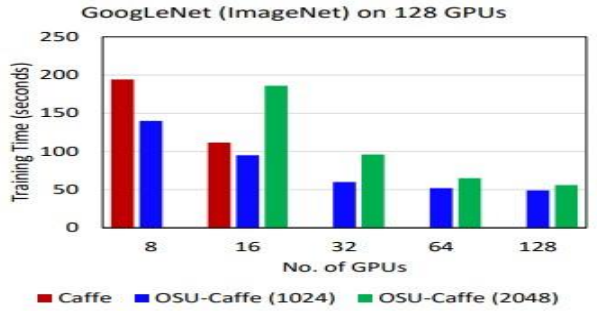
Conceptual sketch of sharp and flat minimas of a loss function

Impact of Large Batch Size



*Large Batch Size is **bad** for Accuracy*

Courtesy: https://research.fb.com/publications/imagenet1kin1h/



*But **good** for speed and scalability*

A. A. Awan et al., S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters. PPoPP '17

Other Open-Source software resources

Leverage the power of semi-supervised learning and of transformers

No code or low code approach

- Spark NLP & John Snow lab
  - Website
  - Wikipedia
- Huggingface models

- Open Source Software from Nvidia
  - NeMo
  - Clara
  - Large Language Models
  - French and Spanish translator
  - BERT and beyond

References

- Scikit-learn and cheat-sheet
- Pandas and cheat-sheet
- Numpy
- Neural networks cheat-sheet
- Pytorch
  - Pytorch tutorial and code
  - documentation
  - Joseph Pareti's github repo
- Tutorials on LSTM, Autoencoders for seq2seq
- Data-science guide and further resources
- Python
- Anaconda to install open-source tools on your PC

- Engineering Applications
  - Nvidia SimNet
  - Building and deploying PINN based solvers
  - Implementing PINNs
  - Building and deploying CNN based solvers
  - Monolith ai
  - Cardiac Valve Machine Learning Model by UberCloud
- HPC for AI
  - Ohio State University Tutorial
  - Scaling DL Training: NERSC Tutorial

Courtesy of www.theubercloud.com

EURO

Repairing Cardiac Valve Leakage with Patient-specific Data Generation and Machine Learning on an Azure/Google/UberCloud Multi-cloud Environment with SUSE Rancher Kubernetes Management

Customer:

3DT HOLDINGS, LLC

UCSF Health

CALIFORNIA MEDICAL INNOVATIONS INSTITUTE

Technology Partners:

UberCloud

Google Cloud

Azure

eDCV

SIMULIA

SUSE

RANCHER

Yaghoub Dabiri, Ghassan S. Kassab, and Julius M. Guccione, 3DT Holdings, San Diego, CA

Daniel Gruber and Wolfgang Gentzsch, UberCloud, Los Altos, CA

Zusammenfassung

- scikit-learn for machine learning applications, and pytorch for deep networks development are important open-source software tools
- CNNs and existing networks are essential for DL projects
- Time-dependent problems such as NLP, predictive maintenance require special models like LSTM, BERT, and more
- AI is useful for HPC applications, because it helps accelerate time-to-solution. HPC is useful for AI because it applies parallelization and optimization to train large models
- There is a wealth of resources in internet, including stackoverflow.com to help you debug

Backup slides

PyTorch: A more realistic model than shown before (0 of 6)

EURO

- Multi-class classification using PyTorch: here is the link to the code
- The input data is fed into the neural network in batches using the Dataset and dataloader classes of PyTorch
- Data normalization and split into training and test datasets are implemented using scikit-learn
- Use cross-entropy loss which is suitable for classification, easy to experiment with multiple fc layers
- Getting the data types right was not immediate. stackoverflow.com helps.

PyTorch: A more realistic model (1 of 6)

EURO

github.com/joepareti54/more_basic_tests_for_DL/blob/main/test_pytorch_3_classes_wine.ipynb

Apps YouTube Maps Gmail Imported Imported (1) NGC | Catalog

355 lines (355 sloc) | 9.83 KB

Raw Blame

In [1]:

```
# joepareti54@gmail.com  May 6-13 2022
#
# classification of wine types
# using a dataset built from wine.csv
#
# scaling and train-test split is implemented using sklearn
# some issues on datatypes required by pytorch was resolved here:
# https://stackoverflow.com/questions/60440292/runtimeerror-expected-scalar-type-long-but-found-float
#
# therefore the assignment in the training loop:
# LAB = LAB.type(torch.LongTensor)
#
# moreover, the following data conversions are required on the transformed data:
#
#X_TRAIN = X_TRAIN.astype(dtype=np.float32)
#y_train = y_train.astype(dtype=np.float32)
#
#The last layer of the nn is LogSoftmax
#This returns logits that need to be converted into probabilities, and since they are log you need the exp function
#
#the last cell compares the top class of the prediction on test data vs. label
```

PyTorch: A more realistic model (2 of 6)

EURO

In [2]:
```
import numpy as np
import torch
import torch.nn.functional as F
from torch import nn, optim
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torchvision import transforms
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
#
import pandas as pd
```

In [3]:
```
#read the wine.csv file into a pandas dataframe
data = np.asarray(pd.read_csv('./wine.csv', header = None))
X = data[1:, :]
print(X[0:2, :])
```

```
[['1' '14.23' '1.71' '2.43' '15.6' '127' '2.8' '3.06' '0.28' '2.29'
  '5.64' '1.04' '3.92' '1065']
 ['1' '13.2' '1.78' '2.14' '11.2' '100' '2.65' '2.76' '0.26' '1.28'
  '4.38' '1.05' '3.4' '1050']]
```

In [4]:
```
#split in train and test array
XX = X[:,1:]
y = X[:,0]
X_train, X_test, y_train, y_test = train_test_split(X[:,1:], X[:,0], test_size=0.2, random_state=42)
_train, X_test, y_train, y_test = train_test_split(XX, y, test_size=0.2, random_state=42)
```

PyTorch: A more realistic model (3 of 6)

EURO

In [5]:
```
#scaling
scaler = StandardScaler().fit(X_train)
X_TRAIN = scaler.transform(X_train)
X_TEST  = scaler.transform(X_test)
```

In [6]:
```
#convert to float32 numpy arrays
X_TEST = X_TEST.astype(dtype=np.float32)
y_test = y_test.astype(dtype=np.float32)
X_TRAIN = X_TRAIN.astype(dtype=np.float32)
y_train = y_train.astype(dtype=np.float32)
```

In [7]:
```
#define tensor transformation
transform =transforms.Compose([
    transforms.ToTensor()])
```

In [8]:
```
class Wine(Dataset):
    def __init__(self, file_flag,X_TRAIN, X_TEST, y_train,y_test, transform=transform):
        if file_flag == 'TEST':
            self.x = X_TEST
            self.y = y_test
            self.num_samples = X_TEST.shape[0]
        else:
            self.x = X_TRAIN
            self.y = y_train
            self.num_samples = X_TRAIN.shape[0]

    def __len__(self):
        return(self.num_samples)

    def __getitem__(self, index):
        return (self.x[index], self.y[index])
#
```

PyTorch: A more realistic model (4 of 6)

EURO

In [9]:
```
#validate dataset class
WineDS_TEST = Wine('TEST' ,X_TRAIN, X_TEST, y_train,y_test)
WineDS_TRAIN= Wine('TRAIN',X_TRAIN, X_TEST, y_train,y_test)
print('size of train ds ',len(WineDS_TRAIN))
print('size of test  ds ',len(WineDS_TEST))
```

```
size of train ds  142
size of test  ds  36
```

In [10]:
```
#convert data to a dataloder suitable for mini-batch processing when training
train_dataloader = DataLoader(WineDS_TRAIN, batch_size=4, shuffle=True)
test_dataloader = DataLoader(WineDS_TEST, batch_size=4, shuffle=False)
#
train_features , train_labels = next(iter(train_dataloader))
print(train_features)
print('----')
print(train_labels)
```

```
tensor([[-0.3415,  1.0321, -0.0030,  0.5887,  0.3806, -0.9270, -0.8016, -1.5493,
         -1.3206, -0.0248, -0.7561, -1.8105, -0.4478],
        [ 1.7142, -0.4417,  0.0688, -2.1708,  0.1066,  1.5908,  1.6369, -0.6105,
          2.3246,  1.0515,  1.0443,  0.5659,  2.6957],
        [-0.3782, -1.2225, -0.4343, -0.4279, -0.0989, -0.1402, -0.0621, -0.5322,
         -0.2545, -1.0495,  1.1729,  0.7882, -0.9457],
        [ 1.0535,  1.5321,  0.0688,  0.0078, -0.7839, -0.7696, -1.1714,  0.8760,
         -0.0826,  1.7016, -1.6563, -1.3241, -0.8461]])
----
tensor([3., 1., 2., 3.])
```

PyTorch: A more realistic model (5 of 6)

```
#
class ThreeClassClassification(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(13, 6)
        self.fc2 = nn.Linear(6,4)
    def forward(self, x):
        x = x.view(x.shape[0], -1)
        x = F.sigmoid(self.fc1(x))
        x = F.log_softmax(self.fc2(x), dim=1)
        return x
```

In [12]:

```
model = ThreeClassClassification()
loss = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 100
```

In [13]:

```
for i in range(epochs):
    running_loss = 0
    for DATA, LAB in train_dataloader:
        optimizer.zero_grad()
        LAB = LAB.type(torch.LongTensor)
        log_ps = model(DATA)
        LOSS = loss(log_ps, LAB)

        LOSS.backward()
        optimizer.step()
        running_loss += LOSS.item()
    else:
        if i % 50 == 0:
            print(running_loss)
```

```
46.03226637840271
/home/joepareti54/anaconda3/lib/python3.8/site-packages/torch/nn/functional.py:1639: UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoi
d instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
5.674673497676849
```

PyTorch: A more realistic model (6 of 6)

In [14]:

```
#test the model
for DATA, LAB in test_dataloader:
    log_ps = model(DATA)
#    print(torch.exp(log_ps), LAB)
    ps = torch.exp(log_ps)
    top_p, top_class = ps.topk(1, dim=1)
    print(top_class.view(1,-1), LAB)
```

```
tensor([[1, 1, 3, 1]]) tensor([1., 1., 3., 1.])
tensor([[2, 1, 2, 3]]) tensor([2., 1., 2., 3.])
tensor([[2, 3, 1, 3]]) tensor([2., 3., 1., 3.])
tensor([[1, 2, 1, 2]]) tensor([1., 2., 1., 2.])
tensor([[2, 2, 1, 2]]) tensor([2., 2., 1., 2.])
tensor([[1, 2, 2, 3]]) tensor([1., 2., 2., 3.])
tensor([[3, 3, 2, 2]]) tensor([3., 3., 2., 2.])
tensor([[2, 1, 1, 2]]) tensor([2., 1., 1., 2.])
tensor([[3, 1, 1, 1]]) tensor([3., 1., 1., 1.])
```

In [15]:

```
#conclusion:
# all predictions on the test set are right
```

Thanks!

EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, United Kingdom, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, Republic of North Macedonia, Iceland, Montenegro