Pipeline de um processo de Deep Learning implementado em PyTorch:

1. Preparar os Dados
2. Definir o Modelo
3. Treinar o Modelo
4. Avaliar o Modelo
5. Usar o Modelo

MLP para classificação binária

Ionosphere binary classification dataset
https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.csv
(https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.csv)
https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.names
(https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.names))
Previsão da existencia de uma estrutura na atomosfera com dados de um radar.

0. Instalação PyTorch:

- Ir para https://pytorch.org/get-started/locally/ (https://pytorch.org/get-started/locally/)
- Gerar comando escolhendo as opções de acordo com o computador.
- Executar o comando gerado.

In [1]:
```
# Confirmar a instalação

import torch
print(torch.__version__)
```

```
1.8.1+cu111
```

Imports

In [2]:
```
# pytorch mlp for binary classification
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
from torch import Tensor
from torch.nn import Linear
from torch.nn import ReLU
from torch.nn import Sigmoid
from torch.nn import Module
from torch.optim import SGD, Adam
from torch.nn import BCELoss, BCEWithLogitsLoss
from torch.nn.init import kaiming_uniform_
from torch.nn.init import xavier_uniform_
```

In [3]:
```
#Constants

#path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master
/ionosphere.csv'
PATH = 'ionosphere.csv'

device = torch.device("cpu")

EPOCHS = 50
BATCH_SIZE = 64
LEARNING_RATE = 0.001
```

1. Preparar os Dados

In [4]:
```
# definição classe para o dataset
class CSVDataset(Dataset):
    # ler o dataset
    def __init__(self, path):
        # ler o ficheiro csv para um dataframe
        df = pd.read_csv(path, header=None)
        # separar os inputs e os outputs
        self.X = df.values[:, :-1]
        self.y = df.values[:, -1]
        # garantir que os inputs sejam floats
        self.X = self.X.astype('float32')
        # fazer o encoding dos outputs (label) e garantir que sejam
floats
        self.y = LabelEncoder().fit_transform(self.y) #faz o fit e t
ransforma no self.y o 'g' e o 'b' em 0 e 1
        self.y = self.y.astype('float32')
        self.y = self.y.reshape((len(self.y), 1))

    # número de casos no dataset
    def __len__(self):
        return len(self.X)

    # retornar um caso
    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    # retornar índices para casos de treino e de teste
    def get_splits(self, n_test=0.33):
        # calcular o tamanho para o split
        test_size = round(n_test * len(self.X))
        train_size = len(self.X) - test_size
        # calcular o split do houldout
        return random_split(self, [train_size, test_size])#, generat
or=torch.Generator().manual_seed(42))

# preparar o dataset
def prepare_data(path):
    # criar uma instância do dataset
    dataset = CSVDataset(path)
    # calcular o split
    train, test = dataset.get_splits()
    # preparar os data loaders
    train_dl = DataLoader(train, batch_size=len(train), shuffle=Tru
e) #32 len(train)
    test_dl = DataLoader(test, batch_size=1024, shuffle=False)
    train_dl_all = DataLoader(train, batch_size=len(train), shuffle=
False)
    test_dl_all = DataLoader(test, batch_size=len(test), shuffle=Fal
se)
    return train_dl, test_dl, train_dl_all, test_dl_all

# preparar os dados
train_dl, test_dl,  train_dl_all, test_dl_all = prepare_data(PATH)

# sanity check
x,y = next(iter(train_dl))
print(x.shape, y.shape)
x,y = next(iter(test_dl))
print(x.shape, y.shape)
```

```
torch.Size([235, 34]) torch.Size([235, 1])
torch.Size([116, 34]) torch.Size([116, 1])
```

2. Definir o Modelo

In [5]:
```
# Definição da classe para o modelo
class MLP(Module):
    # definir elementos do modelo
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        # input para a primeira camada - Linear - ReLU
        self.hidden1 = Linear(n_inputs, 10)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu') #
He initialization
        self.act1 = ReLU()
        # segunda camada - Linear - ReLU
        self.hidden2 = Linear(10, 8)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        # terceira camada e output Linear - Sigmoid
        self.hidden3 = Linear(8, 1)
        xavier_uniform_(self.hidden3.weight) # Glorot initialization
        self.act3 = Sigmoid()

    # sequência de propagação do input
    def forward(self, X):
        # input para a primeira camada
        X = self.hidden1(X)
        X = self.act1(X)
        # input para a segunda camada
        X = self.hidden2(X)
        X = self.act2(X)
        # input para a terceira camada e output
        X = self.hidden3(X)
        X = self.act3(X)
        return X

# definir a rede neuronal
model = MLP(34) #34 entradas
print(model.parameters)
```

```
<bound method Module.parameters of MLP(
  (hidden1): Linear(in_features=34, out_features=10, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=10, out_features=8, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=8, out_features=1, bias=True)
  (act3): Sigmoid()
)>
```

3. Treinar o Modelo

In [7]:
```
# treino do modelo
def train_model(train_dl, model):
    # definir a função de loss e a função de otimização
    criterion = BCELoss() # Binary Cross Entropy - precisa de sigmoi
d como função de ativação na saída
    optimizer = SGD(model.parameters(), lr=LEARNING_RATE, momentum=
0.9) # stochastic gradient descent
    # iterar as epochs
    for epoch in range(EPOCHS):
        # iterar as batchs
        for i, (inputs, targets) in enumerate(train_dl): # backpropa
gation
            # inicializar os gradientes
            optimizer.zero_grad() # coloca os gradientes de todos os
parametros a zero
            # calcular o output do modelo - previsao/forward
            yprev = model(inputs)
            # calcular o loss
            loss = criterion(yprev, targets)
            # atribuição alteraçoes "In the backward pass we receive
a Tensor containing the gradient of the loss
            # with respect to the output, and we need to compute the
gradient of the loss with respect to the input.
            loss.backward() # backpropagation
            # update pesos do modelo
            optimizer.step()

# treinar o modelo
train_model(train_dl, model)
```

4. Avaliar o Modelo

In [8]:

```
# Avaliar o modelo
def evaluate_model(test_dl, model):
    predictions = list()
    actual_values = list()
    for i, (inputs, labels) in enumerate(test_dl):
        # avaliar o modelo com os casos de teste
        yprev = model(inputs)
        # retirar o array numpy
        yprev = yprev.detach().numpy()
        actual = labels.numpy()
        # arredondar para obter a classe
        yprev = yprev.round()
        # guardar
        predictions.append(yprev)
        actual_values.append(actual)
    predictions, actual_values = np.vstack(predictions), np.vstack(a
ctual_values)
    return predictions, actual_values

# avaliar o modelo
predictions, actual_values = evaluate_model(test_dl, model)
# calcular a accuracy
acc = accuracy_score(actual_values, predictions)
print(f'Accuracy: {acc:0.3f}\n')

acertou=0
falhou = 0
for r,p in zip(actual_values, predictions):
    print(f'real:{r} previsão:{p}')
    if r==p: acertou+=1
    else: falhou+=1
print(f'acertou:{acertou} falhou:{falhou}')

# relatório de classificação: precision, recall, f1-score, support v
s. 0,1, accuracy, macro avg, weighted avg
print(classification_report(actual_values, predictions))
```

```
Accuracy: 0.716

real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[0.]
real:[0.] previsão:[1.]
real:[1.] previsão:[0.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[0.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[0.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[0.] previsão:[0.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
```

```
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[0.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[0.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[0.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
acertou:83 falhou:33
              precision    recall  f1-score   support
```

```
              0.0       0.77      0.25      0.38        40
              1.0       0.71      0.96      0.82        76

    accuracy                              0.72       116
   macro avg       0.74      0.61      0.60       116
weighted avg       0.73      0.72      0.66       116
```

5. Usar o Modelo

In [9]:
```
# fazer uma previsão utilizando um caso
def predict(row, model):
    # converter row para tensor
    row = Tensor([row])
    # fazer a previsão
    yprev = model(row)
    # retirar o array numpy
    yprev = yprev.detach().numpy()
    return yprev

# fazer uma única previsão (classe esperada = 1)
row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.037
60,0.85243,-0.17755,0.59755,-0.44945,0.60536,-0.38223,0.84356,-0.385
42,0.58212,-0.32192,0.56971,-0.29674,0.36946,-0.47357,0.56811,-0.511
71,0.41078,-0.46168,0.21266,-0.34090,0.42267,-0.54487,0.18641,-0.453
00]
yprev = predict(row, model)
print('Predicted: %.3f (class=%d)' % (yprev, yprev.round()))
```

```
Predicted: 0.663 (class=1)
```

In [ ]:
```
```