Problemes

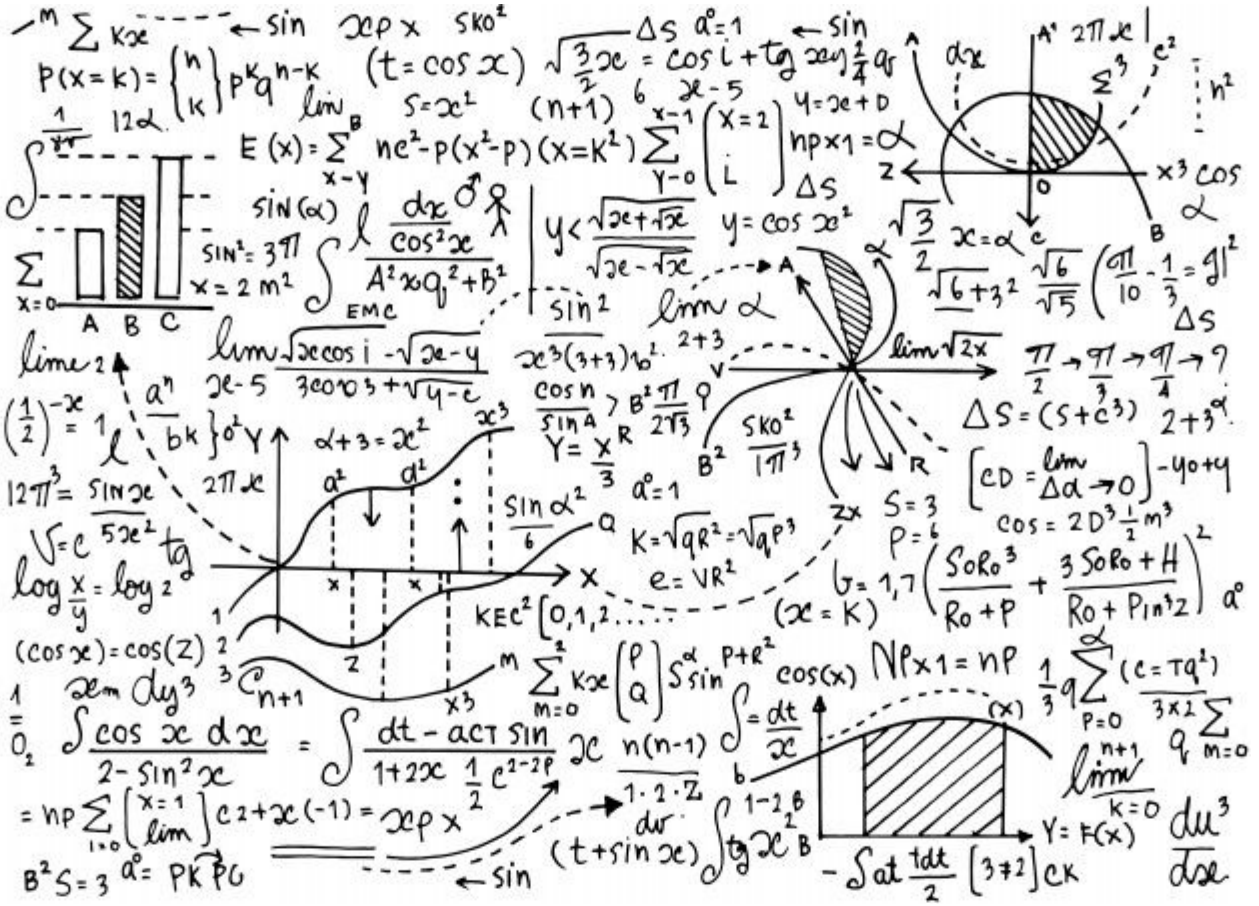
Universitat Autònoma
de Barcelona

Outline

Sessió 2: Classificació

Sessió 3: Backprop

Sessió 4: Memorització



Què es feia abans?

Outline

Sessió 2: Intro + Classificació

Sessió 3: Nets + Backprop

Sessió 4: KNN + Memorització

Què voldriem fer ara?

Entregues

Sessió 2: Intro + Classificació

Regresor Logistic + SVM

Sessió 3: Nets + Backprop

Feedforward + CNN

Sessió 4: KNN + Memorització

NN search (raw data + features)

jupyter

Haureu d'entregar un informe sobre Jupyter Notebook amb el codi explicant el que heu fet

Pytorch Introduction

An open source machine learning framework that accelerates the path from research prototyping to production deployment.

Requirements

pip3 install -r requirements.txt

```
torch
torchvision
```

Funciona sobre win, linux, mac… pip i anaconda

No cal Cuda per les pràctiques, pero si en teniu, anirà més ràpid

```
gonfaus@MacBook-Pro-de-Pep project_problemes % sudo pip3 install -r requirements.txt
Password:
WARNING: The directory '/Users/gonfaus/Library/Caches/pip' or its parent directory is not owned or is not writable by the current user. The cache
has been disabled. Check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
Collecting torch
  Downloading torch-1.6.0-cp38-none-macosx_10_9_x86_64.whl (97.5 MB)
     |████████████████████████████████| 97.5 MB 19.6 MB/s
Collecting torchvision
  Downloading torchvision-0.7.0-cp38-cp38-macosx_10_9_x86_64.whl (387 kB)
     |████████████████████████████████| 387 kB 27.6 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.8/site-packages (from torch->-r requirements.txt (line 1)) (1.19.1)
Requirement already satisfied: future in /usr/local/lib/python3.8/site-packages (from torch->-r requirements.txt (line 1)) (0.18.2)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.8/site-packages (from torchvision->-r requirements.txt (line 2)) (7.2.0)
Installing collected packages: torch, torchvision
Successfully installed torch-1.6.0 torchvision-0.7.0
```

Pytorch steps

dataloader	model	optimizer	loss
How to load the data for training. Small dataset can be feed as in sklearn. Larger datasets are loaded in batches	Define the structure	Is the optimization technique used to modify parameters.	It is just the metric to compute the error of the model.

the dataset (which specifies, how big it is and retrieves the data for each sample)

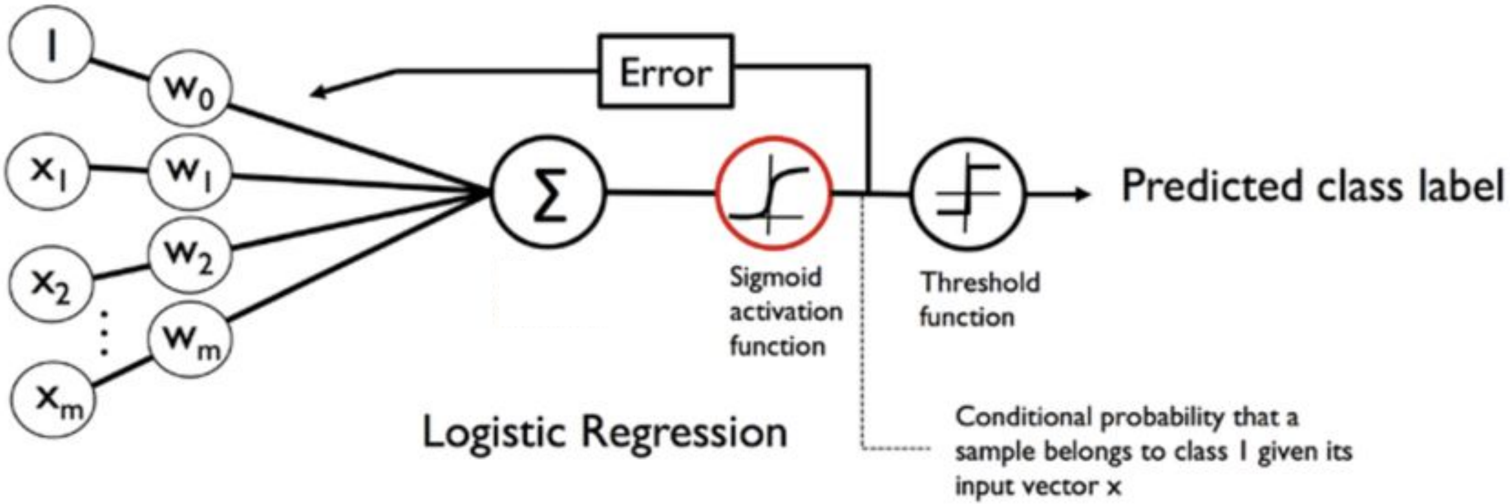
Dataloader

the loader (which iterates the dataset in the way we specify)

```python
## parameter denoting the batch size
BATCH_SIZE = 32

## transformations
transform = transforms.Compose(
    [transforms.ToTensor()])

## download and load training dataset
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)

## download and load testing dataset
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)
```

Model

```
self.w = torch.nn.Linear(784, 1)
```



Model

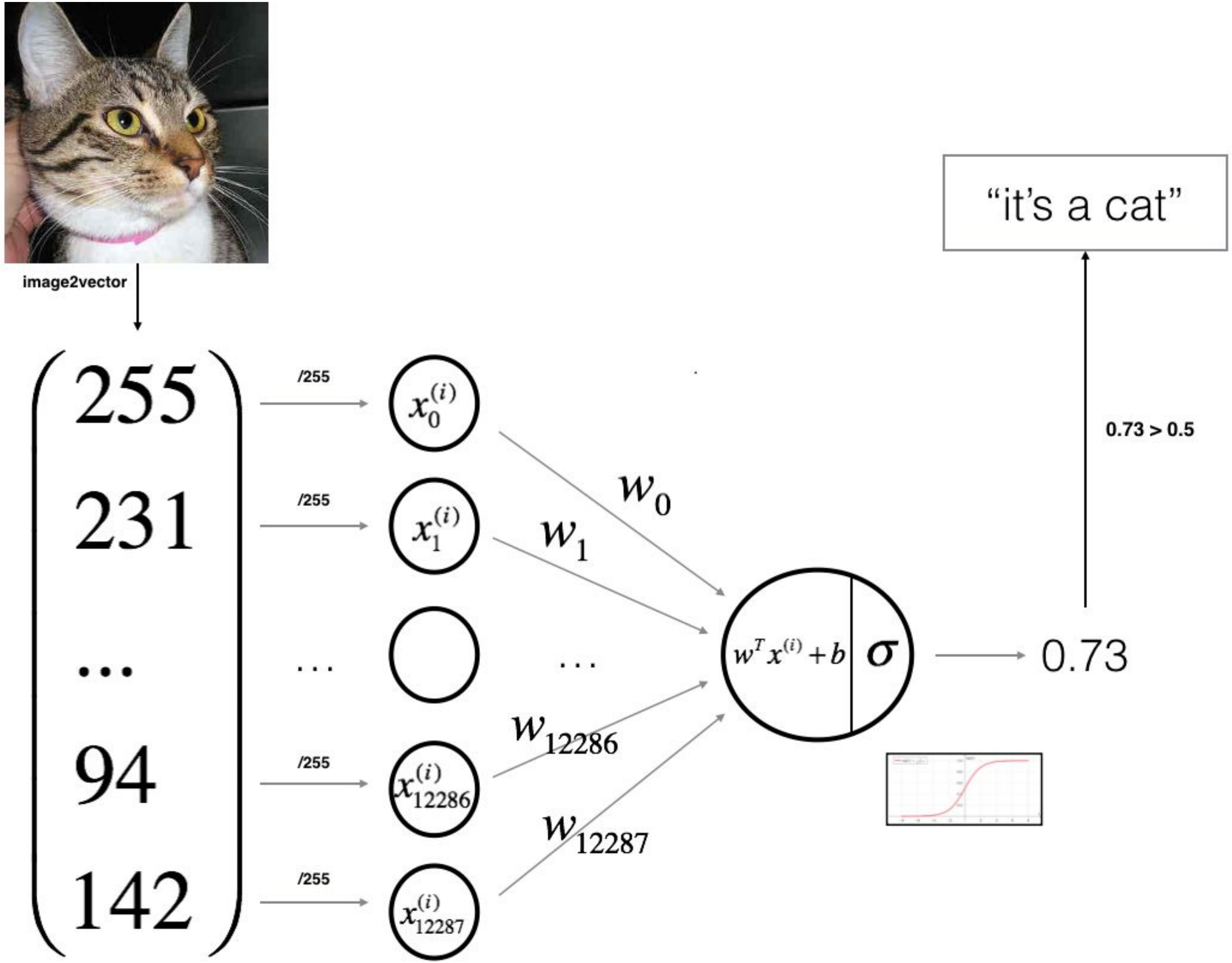
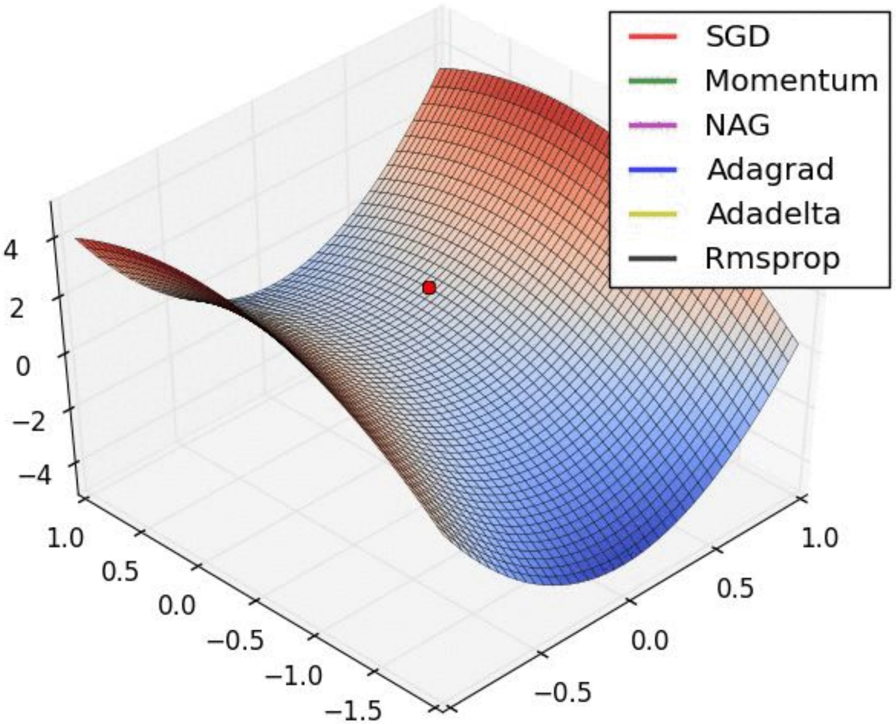


image2vector
255
231
...
94
142
/255
/255
/255
/255
$x_0^{(i)}$
$x_1^{(i)}$
$x_{12286}^{(i)}$
$x_{12287}^{(i)}$
w_0
w_1
w_{12286}
w_{12287}
$w^T x^{(i)} + b$
σ
0.73
0.73 > 0.5
"it's a cat"

Optimizer

In theory, multiple optimizers are seeked during research, while in practice, the most being used are still SGD and momentum

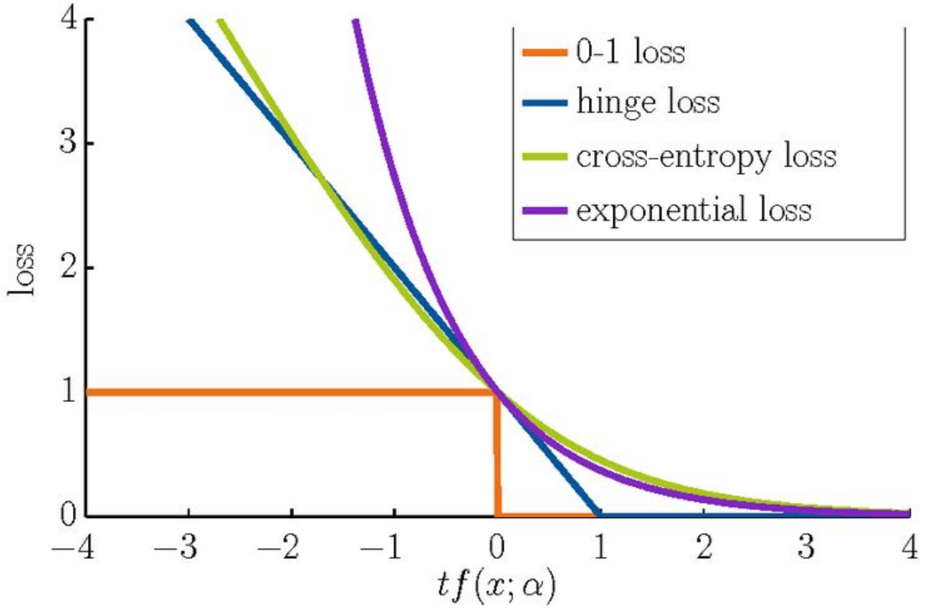



```
opt = optim.x(model.parameters(), ...)      # create optimizer
opt.step()                                  # update weights
optim.X                                     # where X is SGD, Adadelta, Adagrad, Adam,
                                            # SparseAdam, Adamax, ASGD,
                                            # LBFGS, RMSProp or Rprop
```

Loss

How to compute the error between the model predictions and the real target.

- MSELoss for regression
- CrossEntropyLoss for classification




```
nn.X                    # where X is BCELoss, CrossEntropyLoss,
                        # L1Loss, MSELoss, NLLLoss, SoftMarginLoss,
                        # MultiLabelSoftMarginLoss, CosineEmbeddingLoss,
                        # KLDivLoss, MarginRankingLoss, HingeEmbeddingLoss
                        # or CosineEmbeddingLoss
```

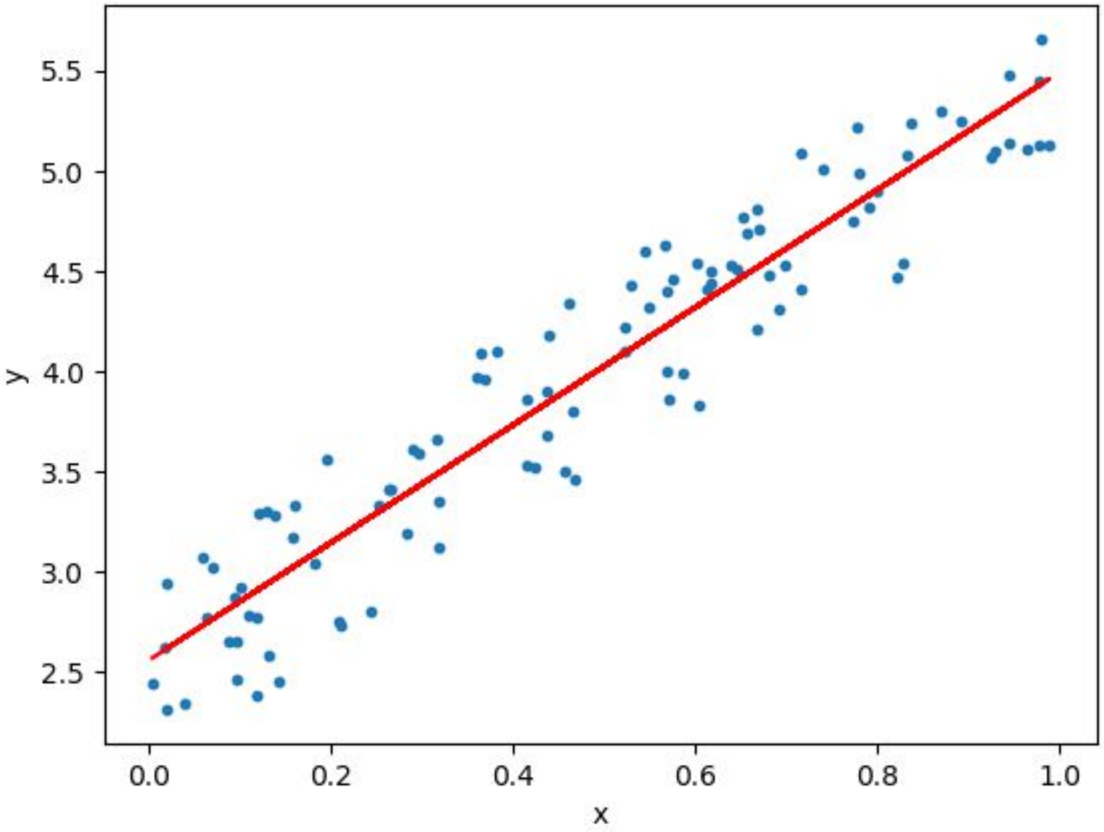
Sessió 2: Classificació Binaria

Introducció a pytorch fent un regresor logistic

Part 1

Regresió lineal & Regresor Logistic

Regressor Lineal



$$\hat{Y} = bX + a + e$$

where,

$\hat{Y}$ = Predicted value of Y

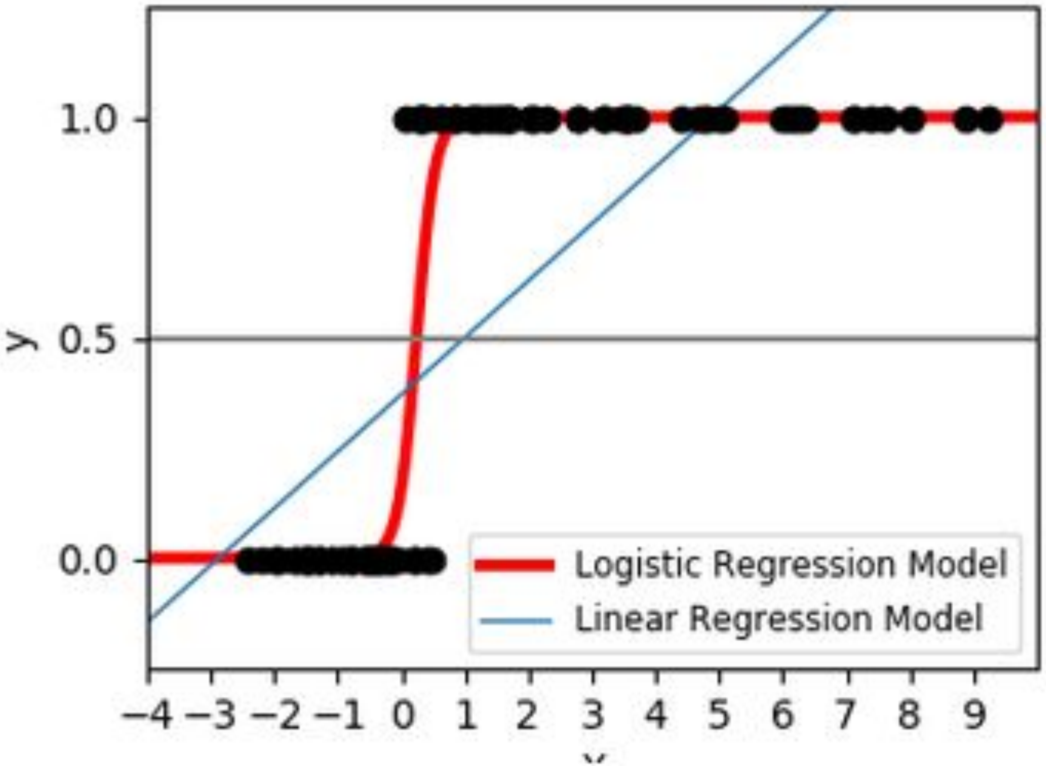
X = Independent variable

b = Slope coefficient based on best-fitting line

a = Intercept

e = Error term

Regressor Logistic



$$\hat{Y} = \sigma(bX + a + e)$$

where,

$\hat{Y}$ = Predicted value of Y

X = Independent variable

b = Slope coefficient based on best-fitting line

a = Intercept

e = Error term

$$\sigma = \frac{1}{1+e^{-z}}$$

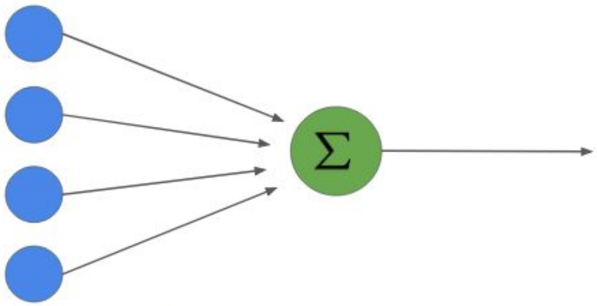
Regressor Lineal

dataloader

torch.utils.data.DataLoader

model

Linear(n_input, 1)



optimizer

SGD

loss

MSELoss

+ regularització

Regresor Logistic

dataloader

model

optimizer

loss

torch.utils.data.DataLoader

Linear(n_input, 1)

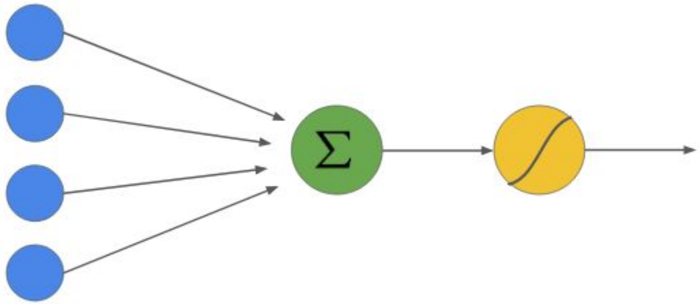
SGD

CrossEntropyLoss
BCELoss (binaria)

o

CustomLoss

+
regularització



Regressor Lineal

model

```python
class LinearRegression(torch.nn.Module):
    def __init__(self):
        super(LinearRegression, self).__init__()
        self.linear = torch.nn.Linear(1, 1)
    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred

model = LinearRegression()
```

dataloader

```python
x_data = torch.Tensor([[0.1], [0.4], [0.6], [0.7], [0.3], [.15], [0.5], [.45]])
y_data = torch.Tensor([[2.6], [3.5], [4.0], [4.5], [3.1], [2.8], [4.1], [3.7]])
```

optimizer

```python
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

loss

```python
criterion = torch.nn.MSELoss(reduction='mean')
```

Regressor Logistic

model

```
class LogisticRegression(torch.nn.Module):
    def __init__(self):
        super(LogisticRegression, self).__init__()
        self.linear = torch.nn.Linear(1, 1)
    def forward(self, x):
        y_pred = torch.sigmoid(self.linear(x))
        return y_pred

model_cls = LogisticRegression()
```

dataloader

```
x_data_cls = torch.Tensor([[-3], [2], [0], [4], [-2], [1], [5], [1.5]])
y_data_cls = torch.Tensor([[ 0], [1], [0], [1],  [0], [0], [1], [1]])
```

optimizer

```
optimizer = torch.optim.SGD(model_cls.parameters(), lr=0.01)
```

loss

```
criterion_cls = torch.nn.BCELoss(reduction='mean')
```

Training Loop → sklearn.fit(x_data, y_data)

```
model.train()  # ens posem en mode 'train'
for epoch in range(20):  # fem 20 epoques
    optimizer.zero_grad()  # resetejem els gradients a zero
    # Forward pass
    y_pred = model(x_data) # fem la predicció
    # Compute Loss
    loss = criterion(y_pred, y_data) # mirem el error
    # Backward pass
    loss.backward()      # calculem els gradients
    optimizer.step()     # actualitzem els pesos
```

Test → sklearn.predict(new_x)

```
new_x = torch.Tensor([[1.0]])
y_pred = model(new_x)
print("predicted Y value: ", y_pred.data[0][0])
```

Part 2

Classificació de caràcters MNIST

la partició en la entrega dependrà del vostre NIU

Data

MNIST DIGITS

60.000 training samples

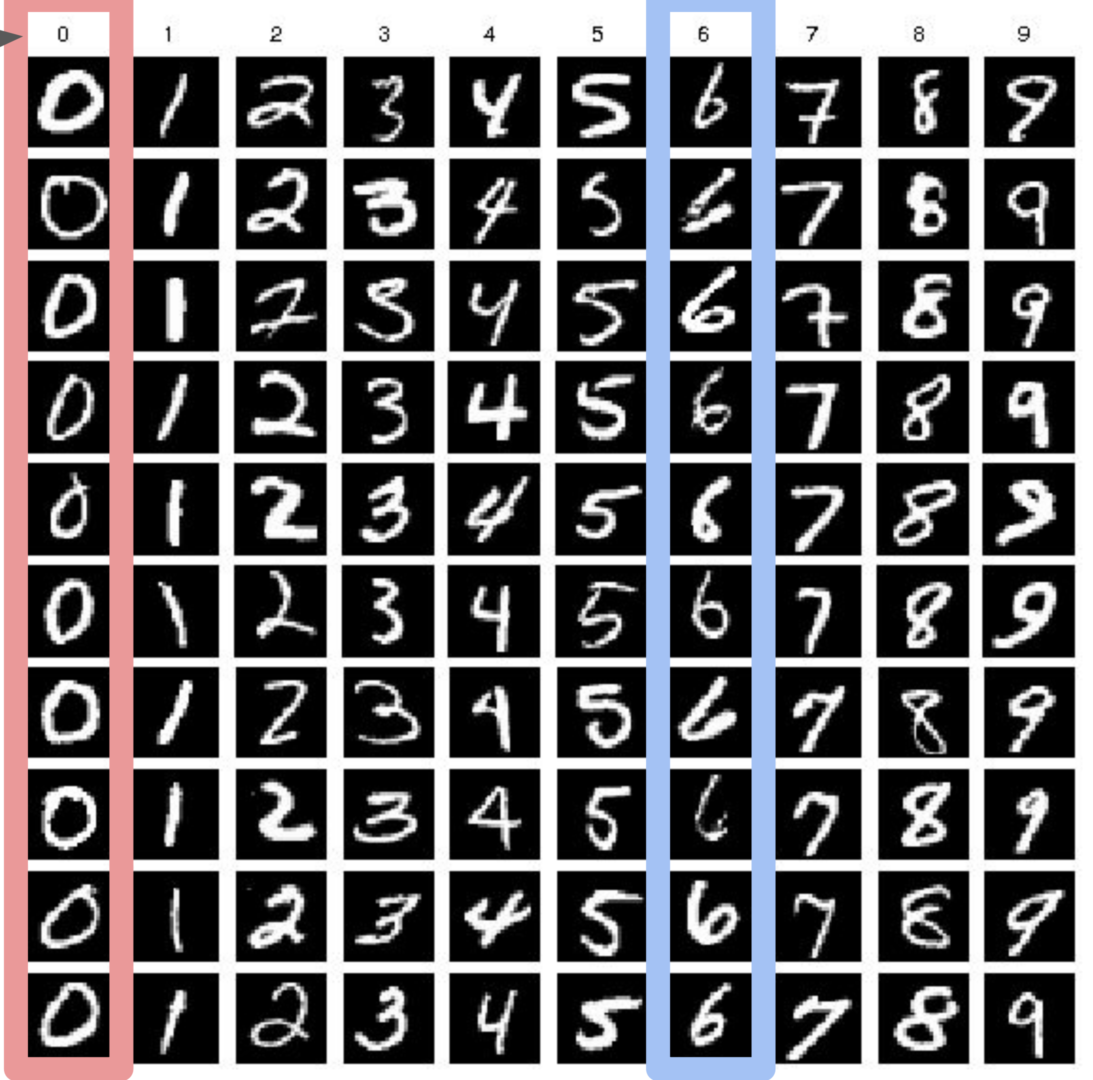
10.000 testing samples

each sample is 28x28 pixels

784 dimensions

10 categories

REDUIREM A 2 CATEGORIES
One-Vs-One



Com escollir les categories per entrenar

NIU: 123456**78**

- 8 positiu, 7 negatiu

NIU: 12345**6**7**7**

si es repeteix el número, agafar el anterior que no sigui el mateix..

- 7 positiu, 6 negatiu

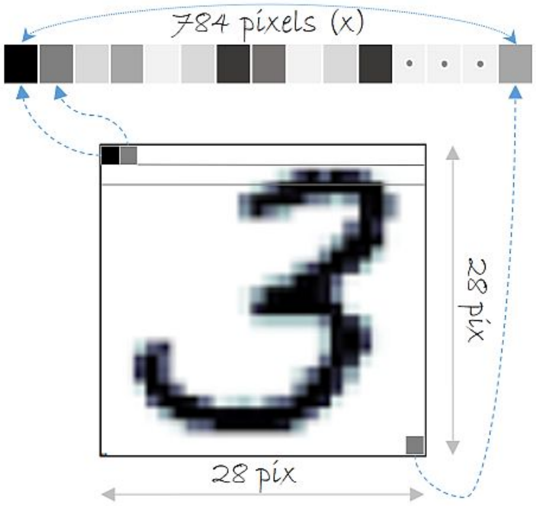
NIU: 1234**5**66**6**

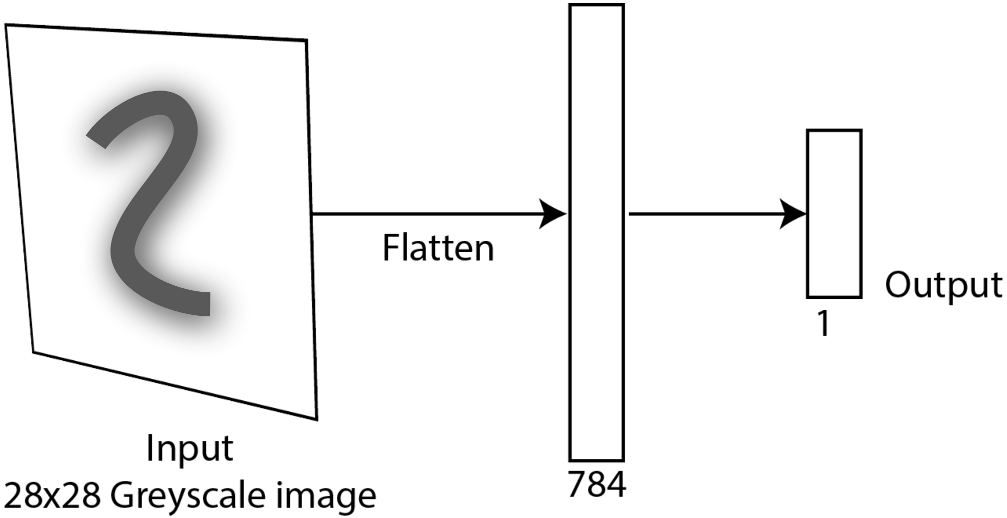
- 6 positiu, 5 negatiu

...

Classificació Binaria amb Pytorch

- Regresor Logistic
- Input:
  - Imatge de 28 x 28 pixels → 784 pixels





Esquelet

imports

model definition

Custom loss

train loop

test loop

main

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.w = nn.Linear(784, 1)

    def forward(self, x):
        x = torch.flatten(x, 1)
        x = self.w(x)
        output = torch.sigmoid(x)

        return torch.flatten(output, 0)
```

```
class LogisticLoss(nn.modules.Module):
    def __init__(self):
        super(LogisticLoss, self).__init__()

    def forward(self, outputs, labels):
        batch_size = outputs.size()[0]
        outputs = (outputs * 2) - 1
        labels = (labels * 2) - 1  # labels -> 1 or -1
        return torch.sum(torch.log(1 + torch.exp(-(outputs.t() * labels)))) / batch_size
```

Esquelet

- imports
- model definition
- Custom loss
- train loop
- test loop
- main

```python
def train(args, model, device, train_loader, optimizer, criterion, epoch, regularizer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output.view_as(target), target.type_as(output))
        loss += regularizer(model)  # this is just for teaching purposes
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                       100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break
```

```python
def test(model, device, test_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output.view_as(target), target.type_as(output)).item() * data.shape[0]
            pred = (output > 0.5) * 1
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

Esquelet

imports

model definition

Custom loss

train loop

test loop

main

```
# Step 1. Load Dataset
# Step 2. Make Dataset Iterable
# Step 3. Create Model Class
# Step 4. Instantiate Model Class
# Step 5. Instantiate Loss Class
# Step 6. Instantiate Optimizer Class
# Step 7. Train Model
```

Resum

A classe:

- Introducció Pytorch
- Implementació de LinearRegression i Logistic Regression
- Logistic Regression Binari amb la categories 0 vs rest

A entregar:

- SVM Binari One-vs-One segons NIU
- Visualització de pesos del model
- Regularització L2
- SVM Binari One-vs-Rest per les 10 categories