Problemes Sessió 3

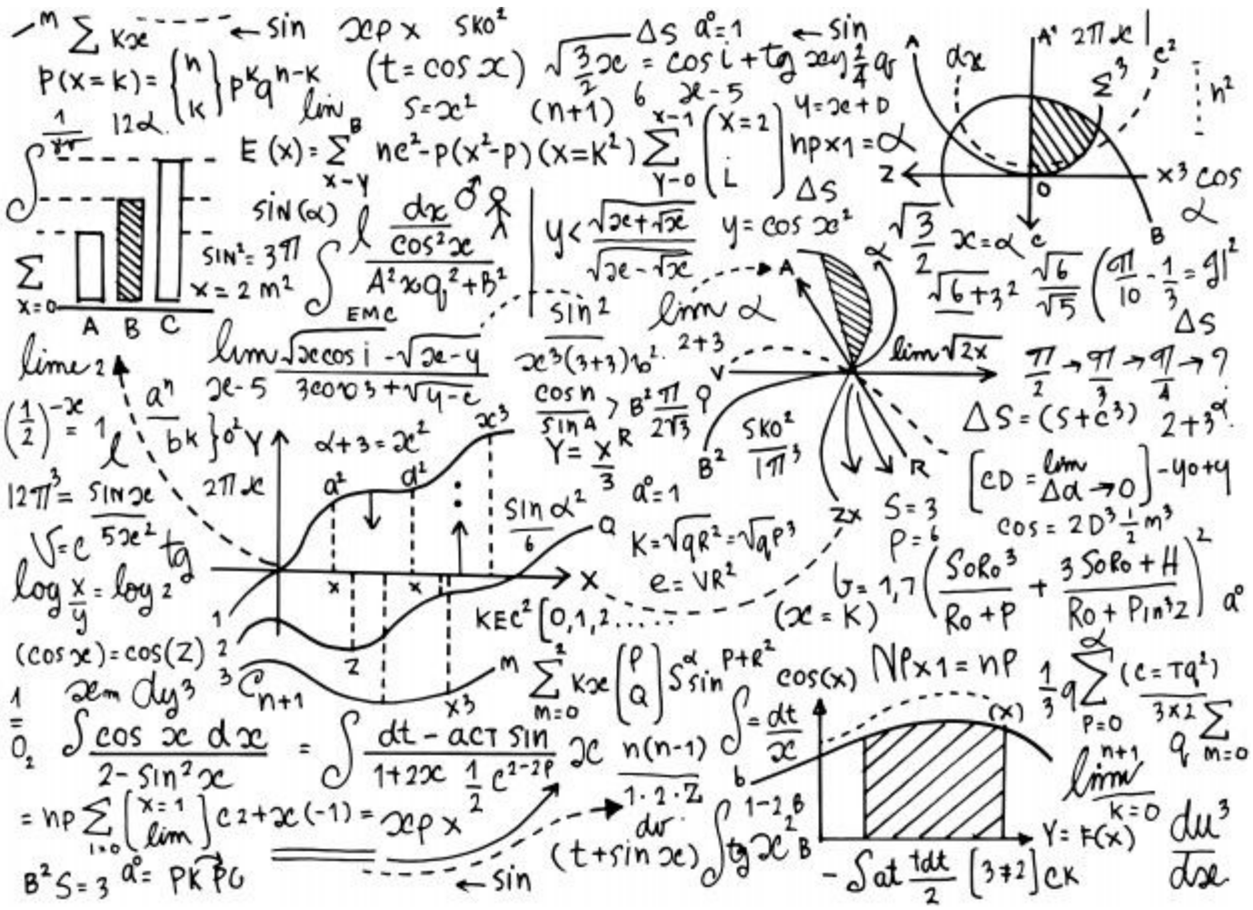
Universitat Autònoma
de Barcelona

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

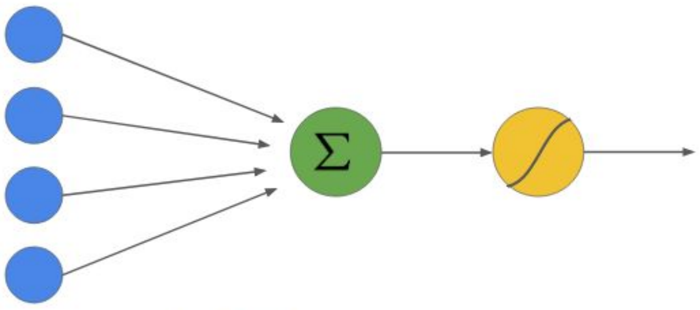
NN search (raw data + features)

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

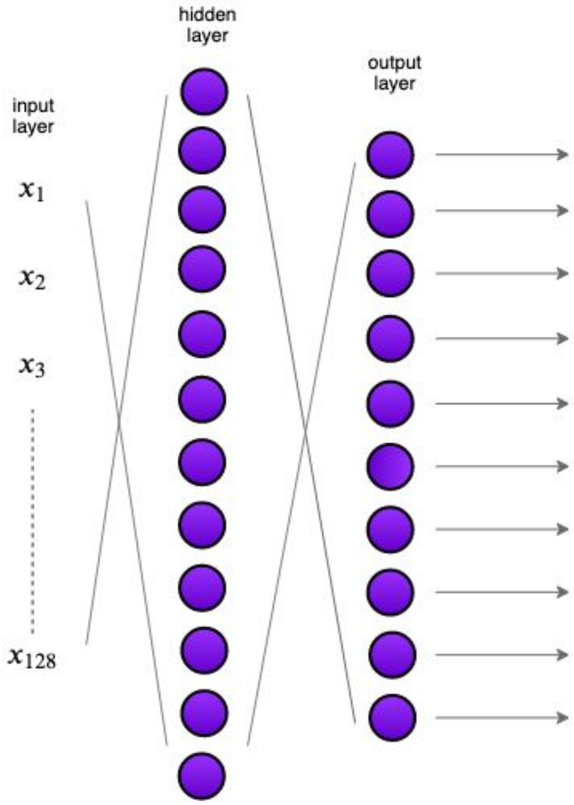
Sessió 3: Backpropagation

Què permet fer?

Què farem en aquesta sessió?



Regresor Logistic, SVM...



Perceptró, Multicapa...

Part 1

Introducció al backprop a través de Autograd

AUTOGRAD: Automatic Differentiation

Create a tensor and set `requires_grad=True` to track computation with it

In [2]:
```
import torch
x = torch.ones(2, 2, requires_grad=True)
print(x)
```

```
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
```

Do a tensor operation:

In [3]:
```
y = x + 2
print(y)
```

```
tensor([[3., 3.],
        [3., 3.]], grad_fn=<AddBackward0>)
```

`y` was created as a result of an operation, so it has a `grad_fn` .

Do more operations on `y`

In [5]:
```
z = y * y * 3
out = z.mean()
print(z, out)
```

```
tensor([[27., 27.],
        [27., 27.]], grad_fn=<MulBackward0>) tensor(27., grad_fn=<MeanBackward0>)
```

`.requires_grad_( ... )` changes an existing Tensor's `requires_grad` flag in-place. The input flag defaults to `False` if not given.

Run in Google Colab   Download Notebook   View on GitHub

https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

AUTOGRAD: Automatic Differentiation

Gradients

Let's backprop now. Because `out` contains a single scalar, `out.backward()` is equivalent to `out.backward(torch.tensor(1.))`.

```
In [8]: out.backward()
```

Print gradients d(out)/dx

```
In [9]: print(x.grad)

tensor([[4.5000, 4.5000],
        [4.5000, 4.5000]])
```

You should have got a matrix of `4.5`. Let's call the `out` *Tensor* "o"

We have that

$$o = \frac{1}{4}\sum_i z_i$$

where

$$z_i = 3(x_i + 2)^2$$

and

$$z_i\big|_{x_i=1} = 27$$

Therefore,

$$\frac{\partial o}{\partial x_i} = \frac{3}{2}(x_i + 2)$$

hence

$$\frac{\partial o}{\partial x_i}\bigg|_{x_i=1} = \frac{3}{2}(1+2) = \frac{9}{2} = 4.5$$

CO Run in Google Colab

Download Notebook

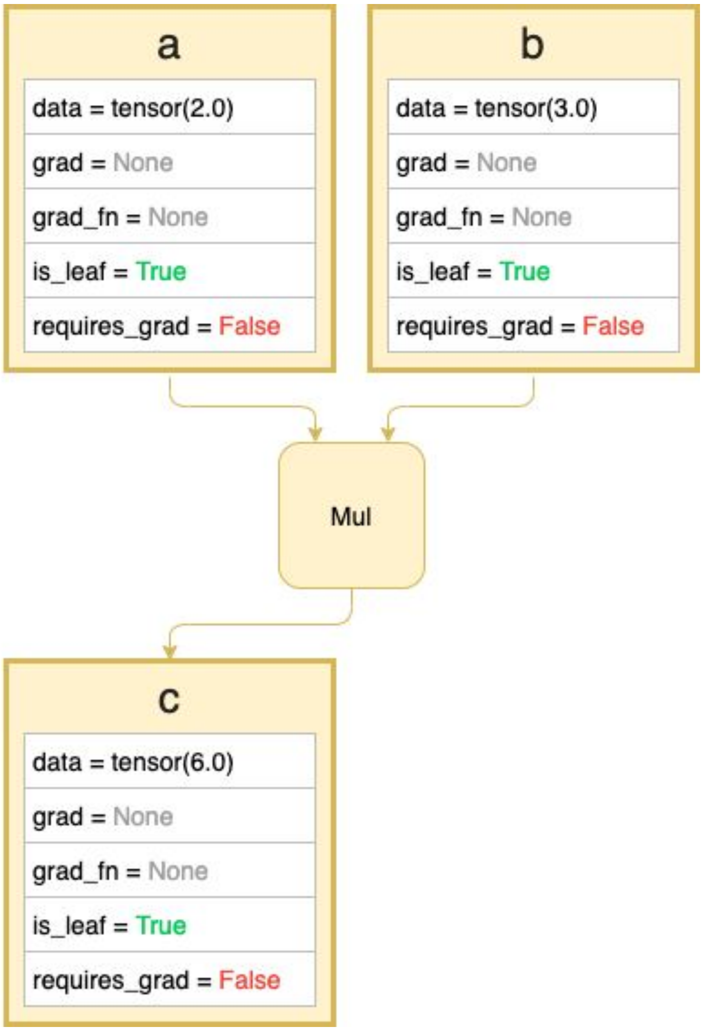
View on GitHub

https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

Com funciona autograd internament?

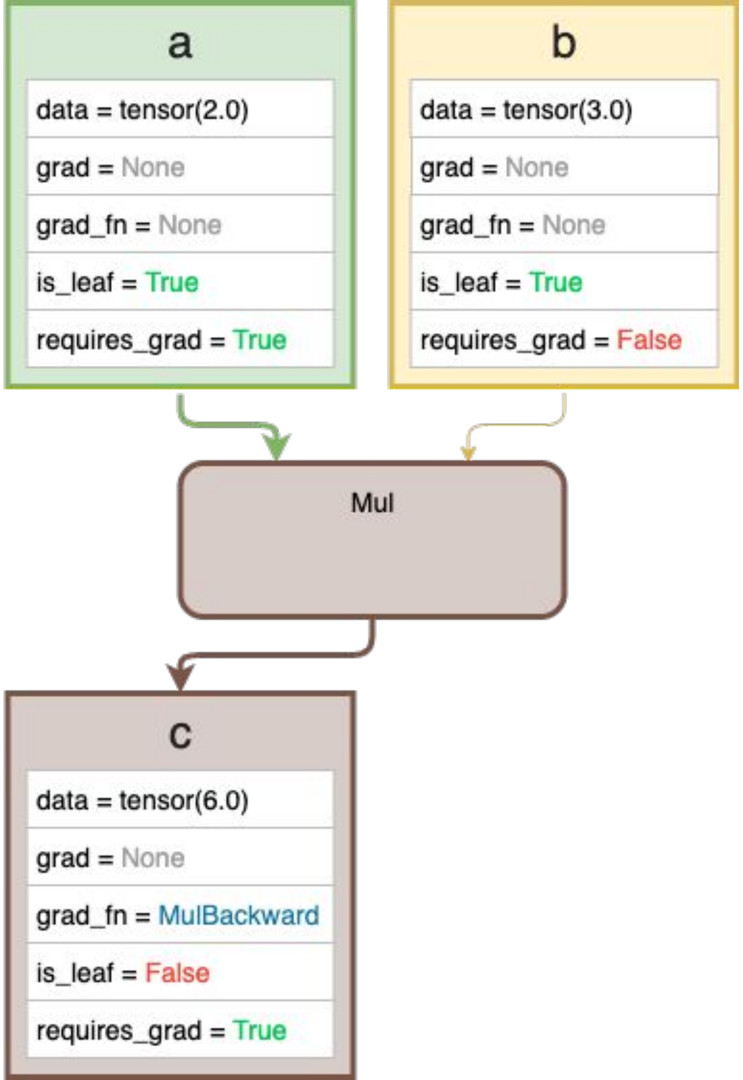
a = torch.tensor(2.0)

b = torch.tensor(3.0)

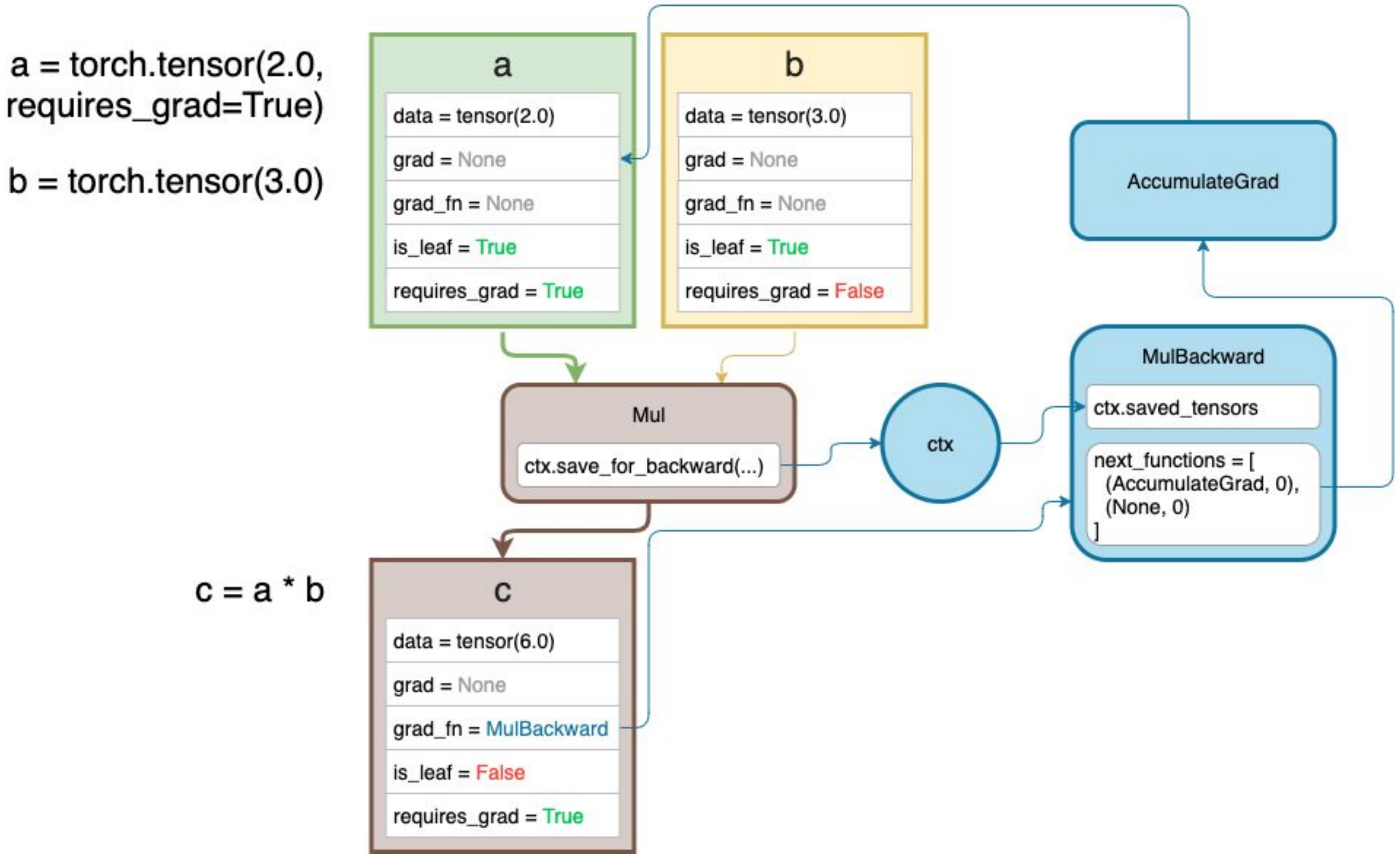


c = a * b

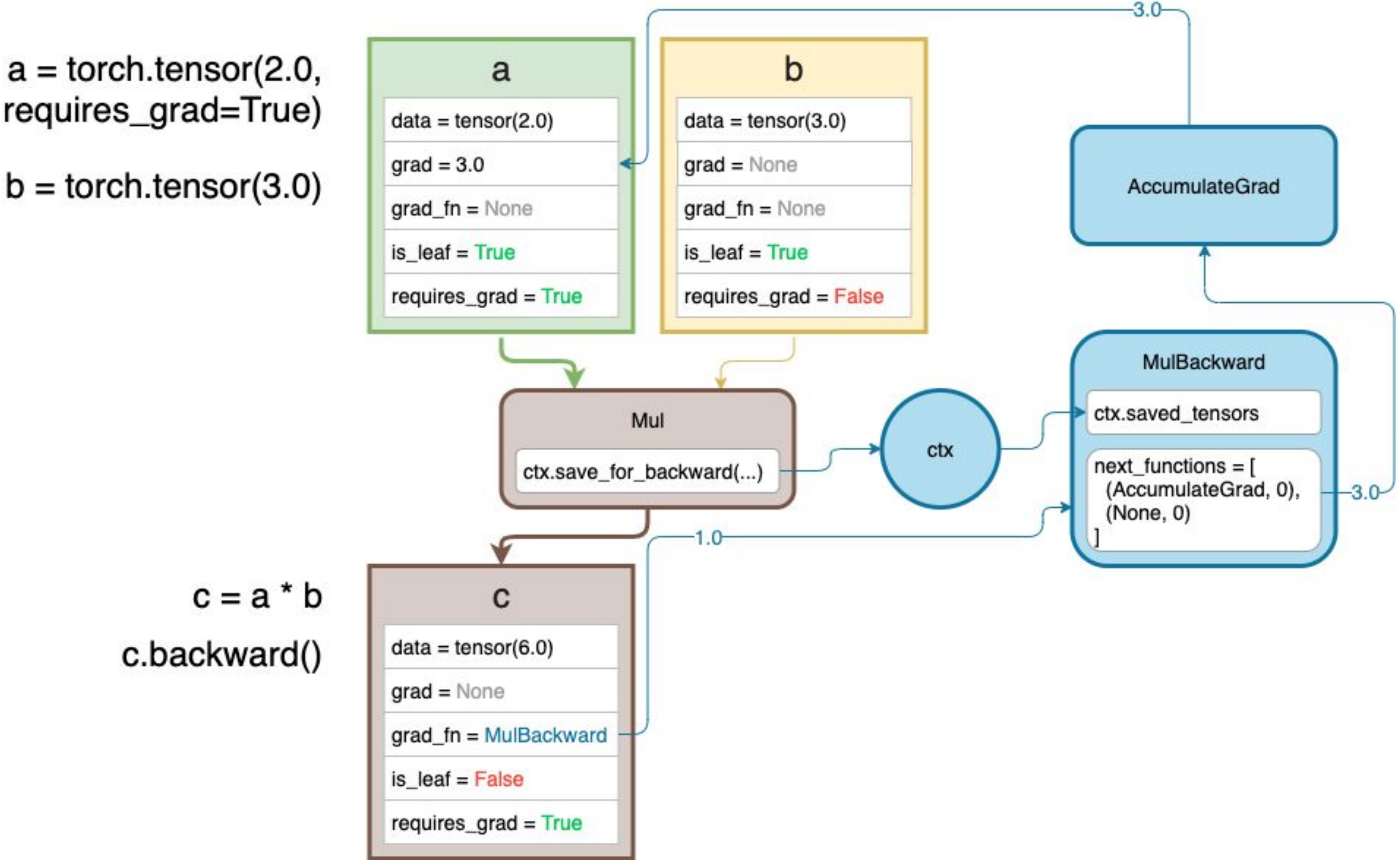
```
a = torch.tensor(2.0,
requires_grad=True)

b = torch.tensor(3.0)
```




```
c = a * b
```



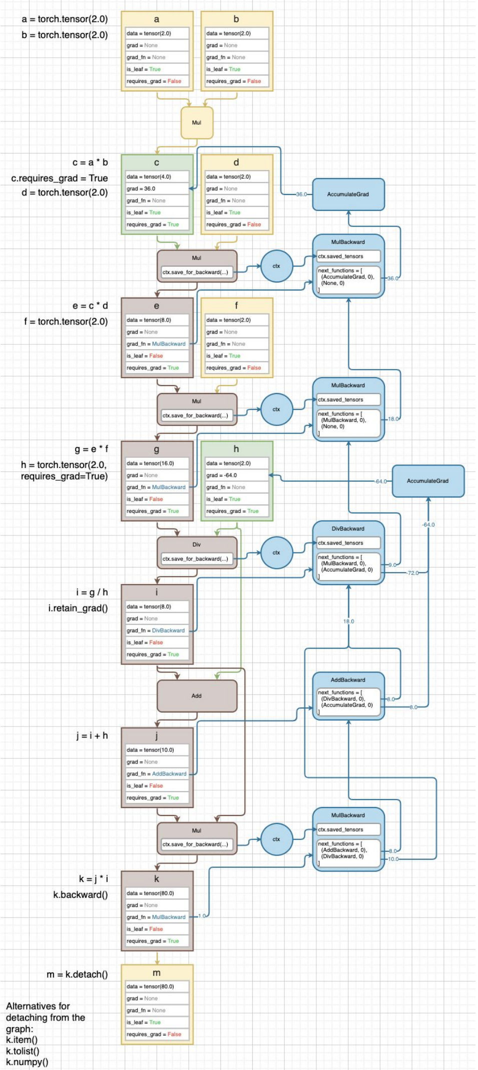
a = torch.tensor(2.0,
requires_grad=True)
b = torch.tensor(3.0)
c = a * b
a
data = tensor(2.0)
grad = None
grad_fn = None
is_leaf = True
requires_grad = True
b
data = tensor(3.0)
grad = None
grad_fn = None
is_leaf = True
requires_grad = False
AccumulateGrad
Mul
ctx.save_for_backward(...)
ctx
MulBackward
ctx.saved_tensors
next_functions = [
(AccumulateGrad, 0),
(None, 0)
]
c
data = tensor(6.0)
grad = None
grad_fn = MulBackward
is_leaf = False
requires_grad = True


a = torch.tensor(2.0,
requires_grad=True)
b = torch.tensor(3.0)
c = a * b
c.backward()
a
data = tensor(2.0)
grad = 3.0
grad_fn = None
is_leaf = True
requires_grad = True
b
data = tensor(3.0)
grad = None
grad_fn = None
is_leaf = True
requires_grad = False
AccumulateGrad
3.0
Mul
ctx.save_for_backward(...)
ctx
MulBackward
ctx.saved_tensors
next_functions = [
(AccumulateGrad, 0),
(None, 0)
]
3.0
1.0
c
data = tensor(6.0)
grad = None
grad_fn = MulBackward
is_leaf = False
requires_grad = True

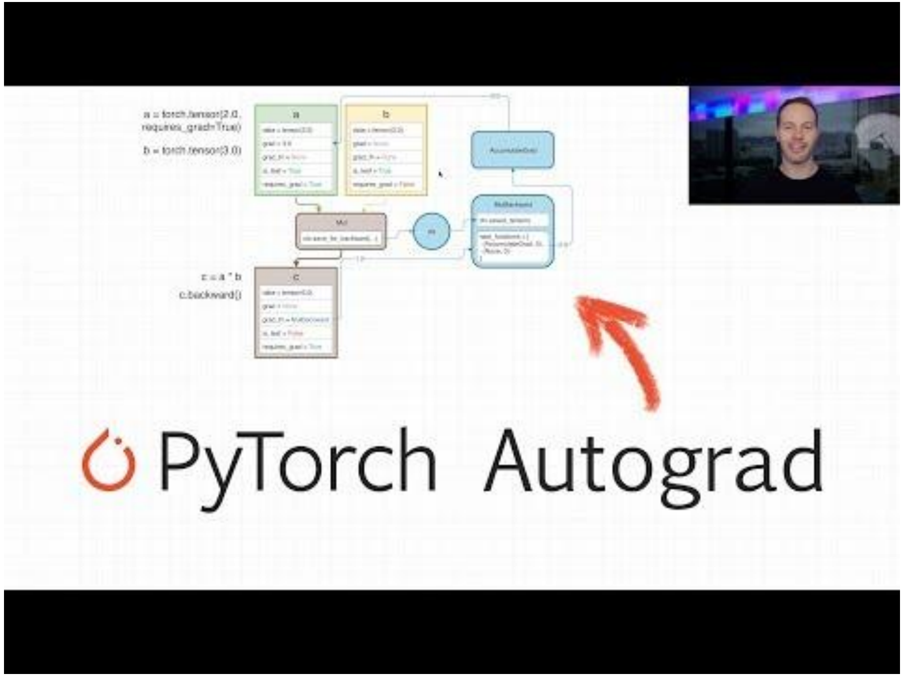
Com funciona autograd internament

Una mica més complicat..

4 variables
6 operacions...



Com funciona autograd internament



Elliot Waite
2750 suscriptores

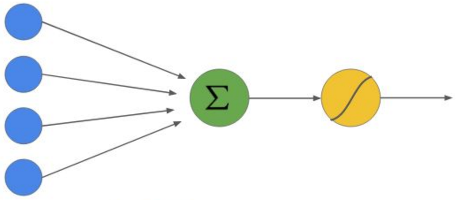
PyTorch Autograd Explained - In-depth Tutorial

https://www.youtube.com/watch?v=MswxJw-8PvE

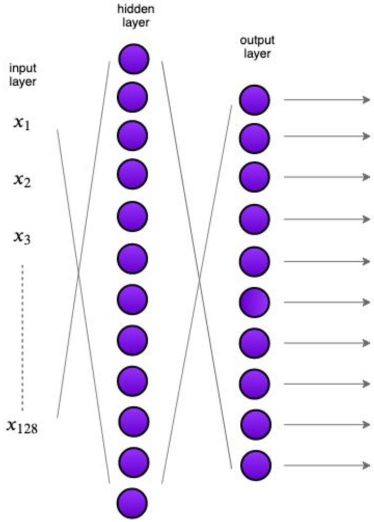
Què deu passar a la realitat?

To deal with hyper-planes in a 14-dimensional space, visualize a 3-D space and say 'fourteen' to yourself very loudly. Everyone does it —Geoffrey Hinton

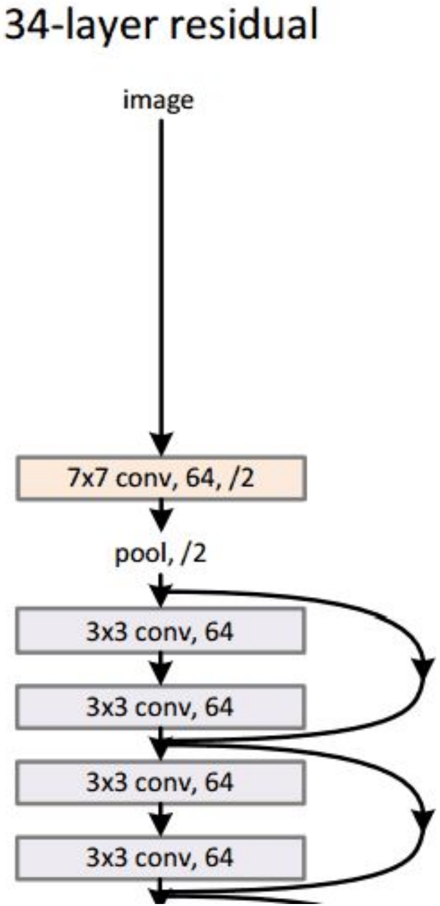
Què deu passar a la realitat?



Regresor
Logistic, SVM...



Perceptró,
Multicapa...

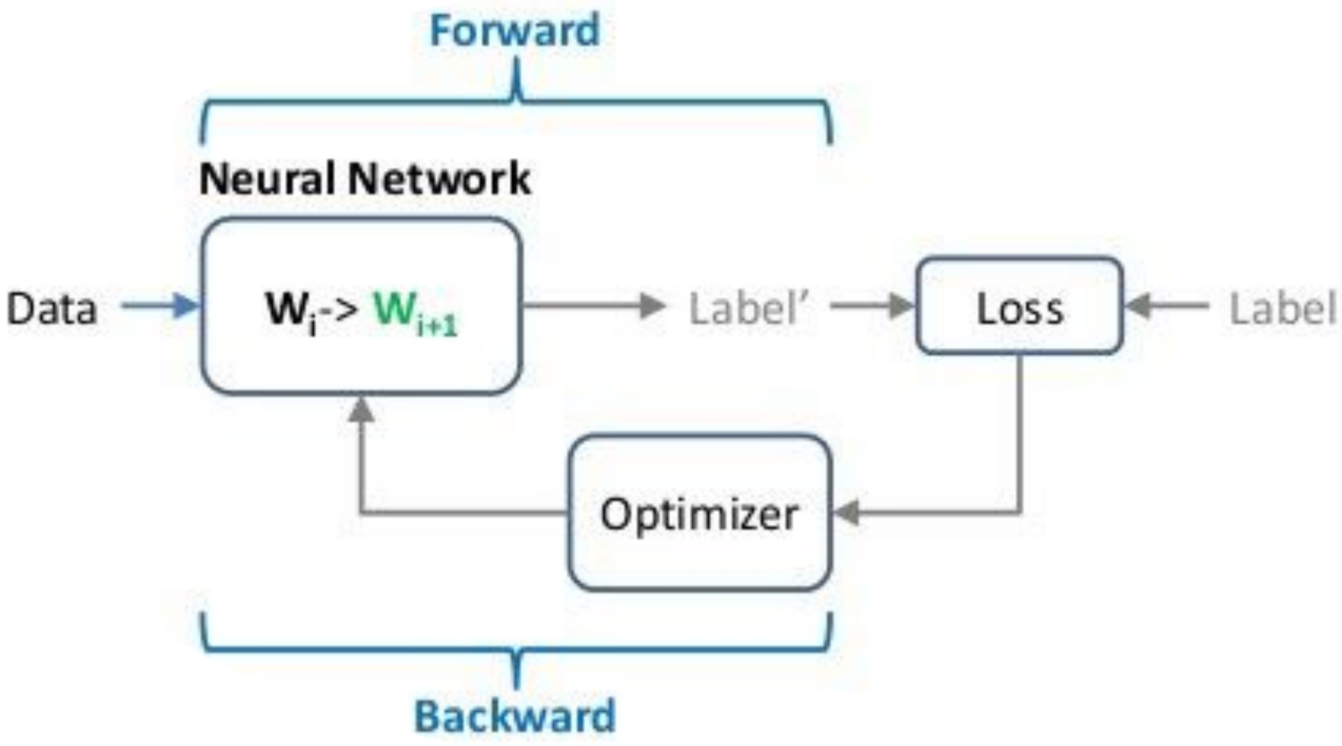


resnet-34 (la ràpida..)

Part 2

Què haureu de fer?

Pytorch steps



Data

Fashion-MNIST

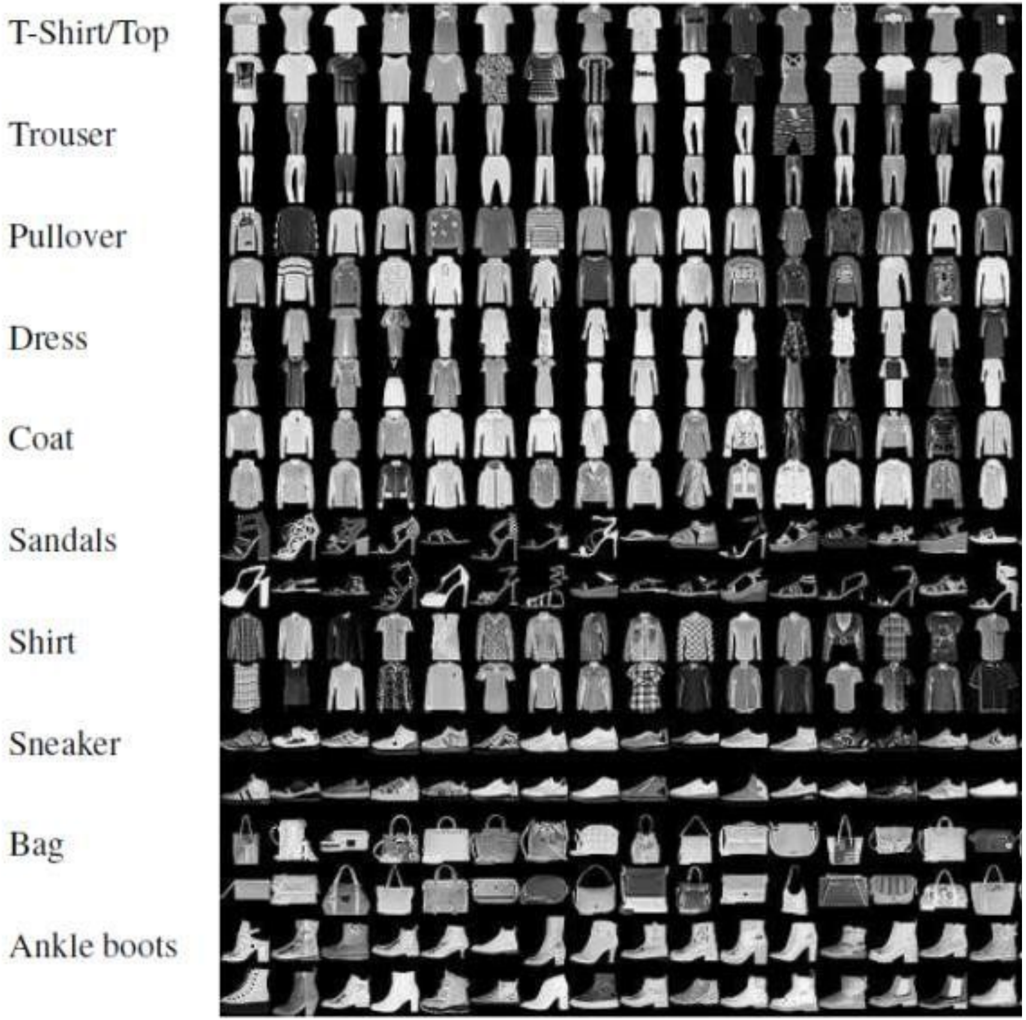
60.000 training samples

10.000 testing samples

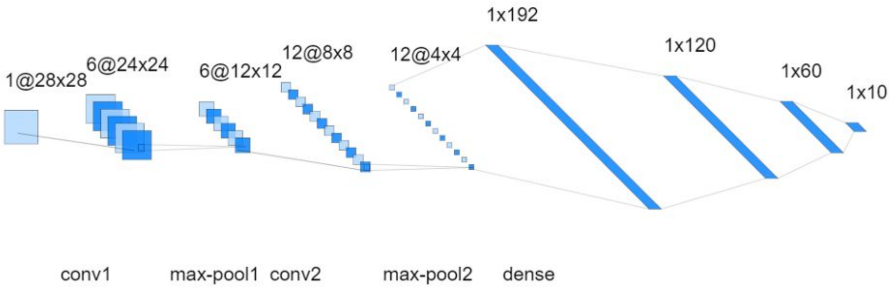
each sample is 28x28 pixels

784 dimensions

10 categories



Classificació Multi Categoria amb Pytorch

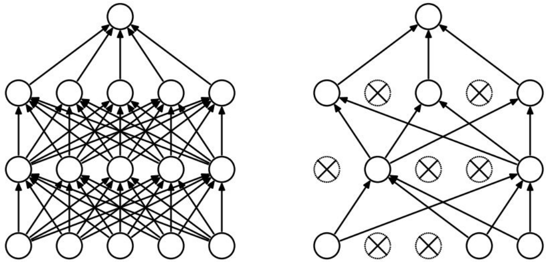


https://towardsdatascience.com/build-a-fashion-mnist-cnn-pytorch-style-efb297e22582

```python
# Build the neural network, expand on top of nn.Module
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        # define layers
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=12, kernel_size=5)

        self.fc1 = nn.Linear(in_features=12*4*4, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=60)
        self.out = nn.Linear(in_features=60, out_features=10)

    # define forward function
    def forward(self, x):
        # conv 1
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2, stride=2)

        # conv 2
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2, stride=2)

        # fc1
        x = x.reshape(-1, 12*4*4)
        x = self.fc1(x)
        x = F.relu(x)

        # fc2
        x = self.fc2(x)
        x = F.relu(x)

        # output
        x = self.out(x)
        # don't need softmax here since we'll use cross-entropy as activation.

        return x
```

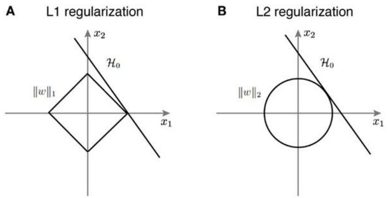
Regularització

Dropout



(a) Standard Neural Net (b) After applying dropout.

Lasso (L1) - **Ridge** (L2. also known as weight decay)



Batch Normalisation: Batch Normalisation tends to fix the distribution of the hidden layer values as the training progresses.

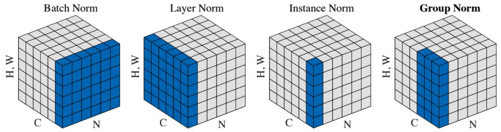
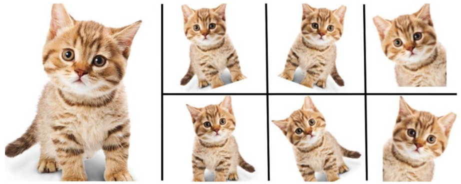


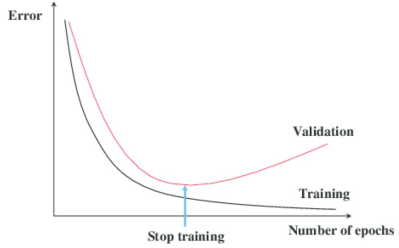
Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

More training data: Adding additional data will add more diversity to the train data and thus reducing the chances of overfitting .

Data Augmentations: It aids in increasing the variety of data for training models thus increasing the breadth of available information.



Early stopping: It implies to stop training of the model early before it reaches overfitting stage. Performance metrics (eg. accuracy, loss) can monitored for train and validation sets to implement this.



Mètode	Detall	Accuracy Test	Temps Train	Temps Test
SVC	{"C":10,"kernel":"poly"}	0.897	1:12:39	?
RandomForestClassifier	{"criterion":"entropy","max_depth":50,"n_estimators":100}	0.879	0:08:39	?
MLPClassifier	{"activation":"relu","hidden_layer_sizes":[100]}	0.877	0:16:03	?
KNeighborsClassifier	{"n_neighbors":5,"p":1,"weights":"distance"}	0.860	0:41:53	?
LinearSVC	{"C":1,"loss":"hinge","multi_class":"ovr","penalty":"l2"}	0.837	0:44:59	?
2 Conv	<100K parameters	0.925	?	?
MobileNet	augmentation (horizontal flips)	0.950	?	?
ResNet18	Normalization, random horizontal flip, random vertical flip, random translation, random rotation.	0.949	?	?
WRN40-4 8.9M params	standard preprocessing (mean/std subtraction/division) and augmentation (random crops/horizontal flips)	0.967	?	?
Google AutoML	24 compute hours (higher quality)	0.939	24h	?

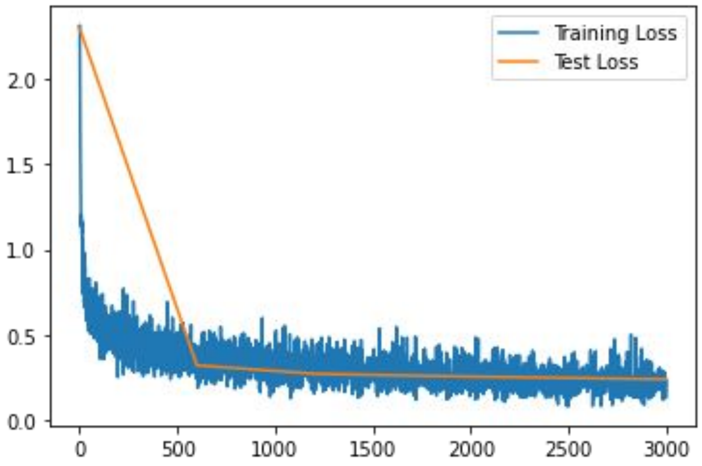
Entrega

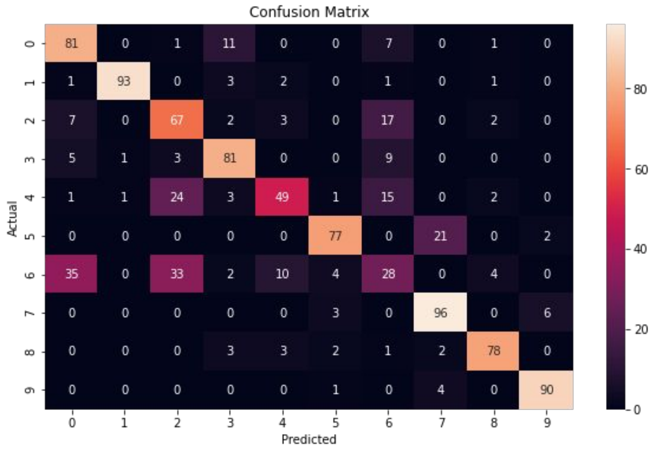
- A. Fes varis entrenaments amb diferents models.**(5pts)**
    - Utilitza els 4 models aqui definits.
    - **Defineix 3 nous models diferents, explica'ls i entrena'ls.** Pots crear-ne de simples, de més complexes, treure poolings o afegir-ne, modificar el percentatge de regularització del dropout, cambiar la funció d'activació, buscar altres definicions per Internet, altres tipus de capes, amb més o menys neurones per capa, capes residuals, provar d'agafar-ne alguna de pre-entrenada… Podeu provar altres configuracions d'entrenament, més o menys èpoques, diferent learning rate, diferent optimitzador, afegir weight_decay….

- B. **Mostra les corbes d'aprenentatge** de cadascun d'ells i compara-les amb les aquí definides. (7 en total) **(2pts)**
    - Per cada model, mostra la loss d'entrenament i la de test en una mateixa gráfica.
    - Mostra les matrius de confusió del model de la última època sobre el conjunt de test.

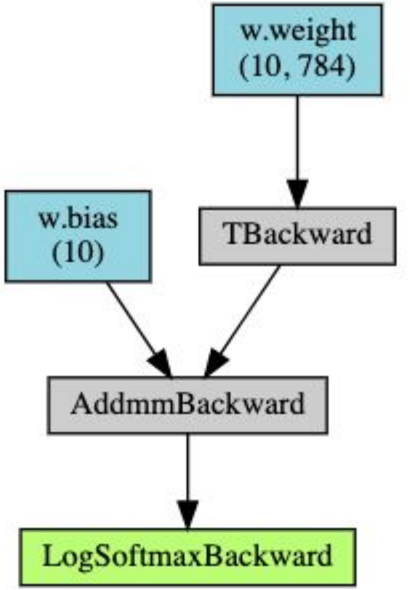
- C. **Mostra els models**, parametres, flops teòrics i temps real dels models **(3pts)**
    - Us he deixat varies funcions que us mostren els models. Haureu d'instal·lar les llibreries corresponents:
        - thop per mostrar número de parametres i flops (o macs)..
        - hiddenlayer i torchviz. Per mostrar els graphs. Aquestes també requereixen tenir instalat graphviz

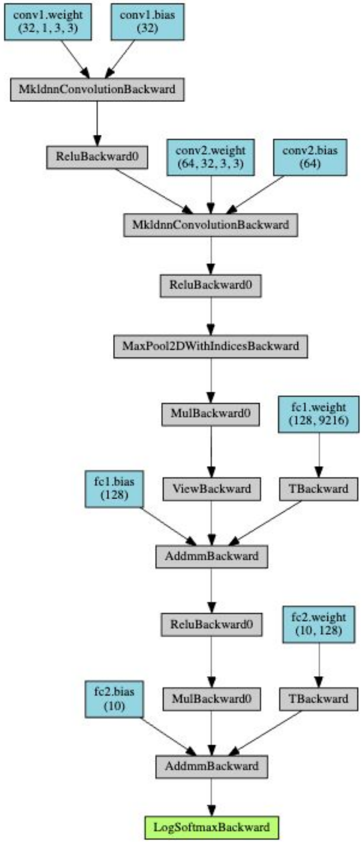
B.

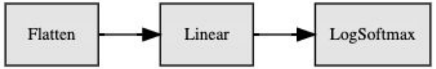

Training Loss
Test Loss
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000

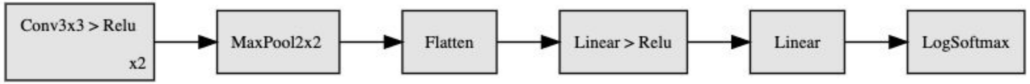

Confusion Matrix
Actual
Predicted

C.


w.weight
(10, 784)
w.bias
(10)
TBackward
AddmmBackward
LogSoftmaxBackward


conv1.weight
(32, 1, 3, 3)
conv1.bias
(32)
MkldnnConvolutionBackward
ReluBackward0
conv2.weight
(64, 32, 3, 3)
conv2.bias
(64)
MkldnnConvolutionBackward
ReluBackward0
MaxPool2DWithIndicesBackward
MulBackward0
fc1.weight
(128, 9216)
fc1.bias
(128)
ViewBackward
TBackward
AddmmBackward
ReluBackward0
fc2.weight
(10, 128)
fc2.bias
(10)
MulBackward0
TBackward
AddmmBackward
LogSoftmaxBackward


Flatten
Linear
LogSoftmax


Conv3x3 > Relu
x2
MaxPool2x2
Flatten
Linear > Relu
Linear
LogSoftmax

C.

1% of the DATA training
10% of the DATA testing

Net	Detalls	Elapsed Time	Accuracy Test	Flops	Params
Net_Linear	1 capa lineal	1.03s	66.5%		
LeNet_tiny	4 vegades més petita que LeNet	1.95s	55.0%		
LeNet	La que es feia servir per MNIST	6.15s	73.7%		
FashionCNN	d'un de kaggle.. (similar a LeNet + batchnorm)	4.75s	72.8%		
X1					
X2					
X3					

temps mesurats en cpu: macbook Pro 2,3 GHz Intel Core i9 de 8 nuclis

C.

10% of the DATA training
10% of the DATA testing

Net	Detalls	Elapsed Time	Accuracy Test	Flops	Params
Net_Linear	1 capa lineal	3.7s	80.2%		
LeNet_tiny	4 vegades més petita que LeNet	7.7s	79.8%		
LeNet	La que es feia servir per MNIST	30.9s	83.7%		
FashionCNN	d'un de kaggle.. (similar a LeNet + batchnorm)	27.4s	83.7%		
X1					
X2					
X3					

temps mesurats en cpu: macbook Pro 2,3 GHz Intel Core i9 de 8 nuclis

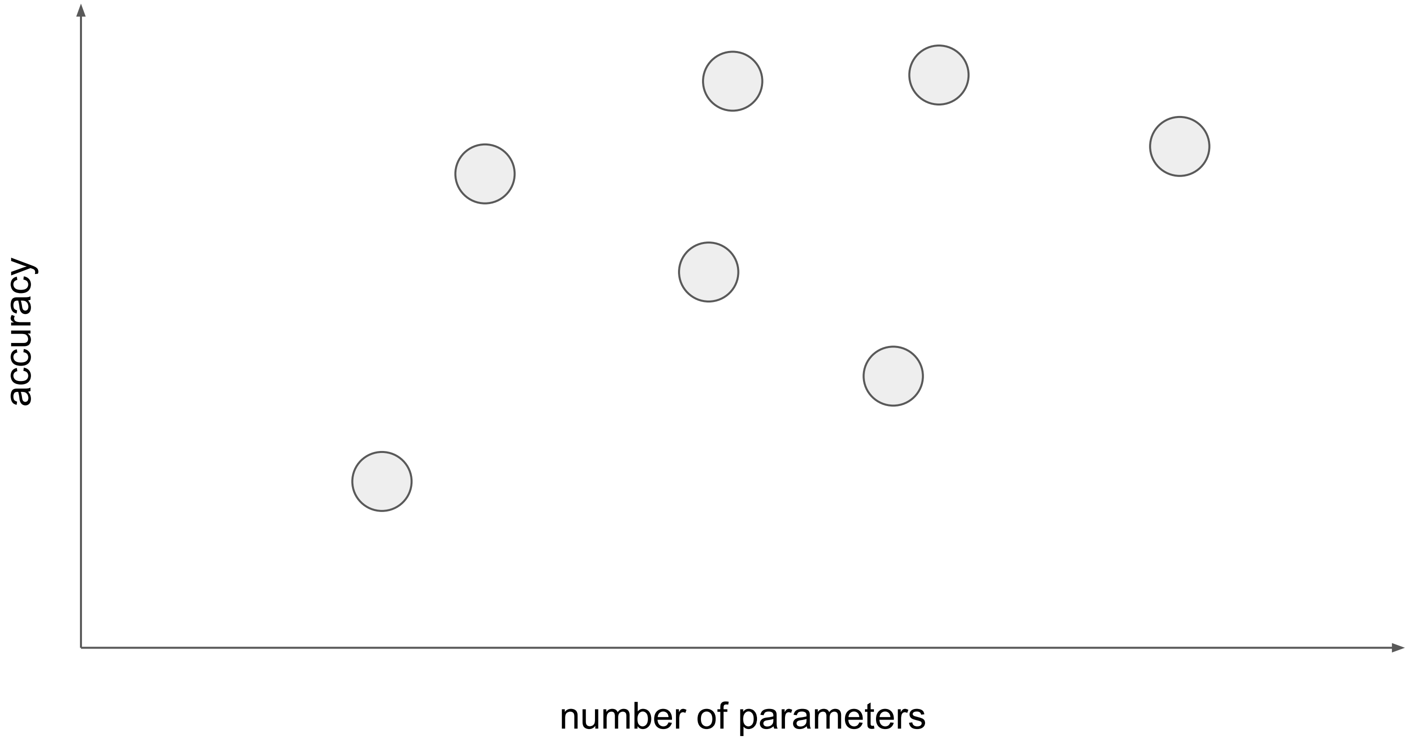
C.

100% of the DATA training
100% of the DATA testing

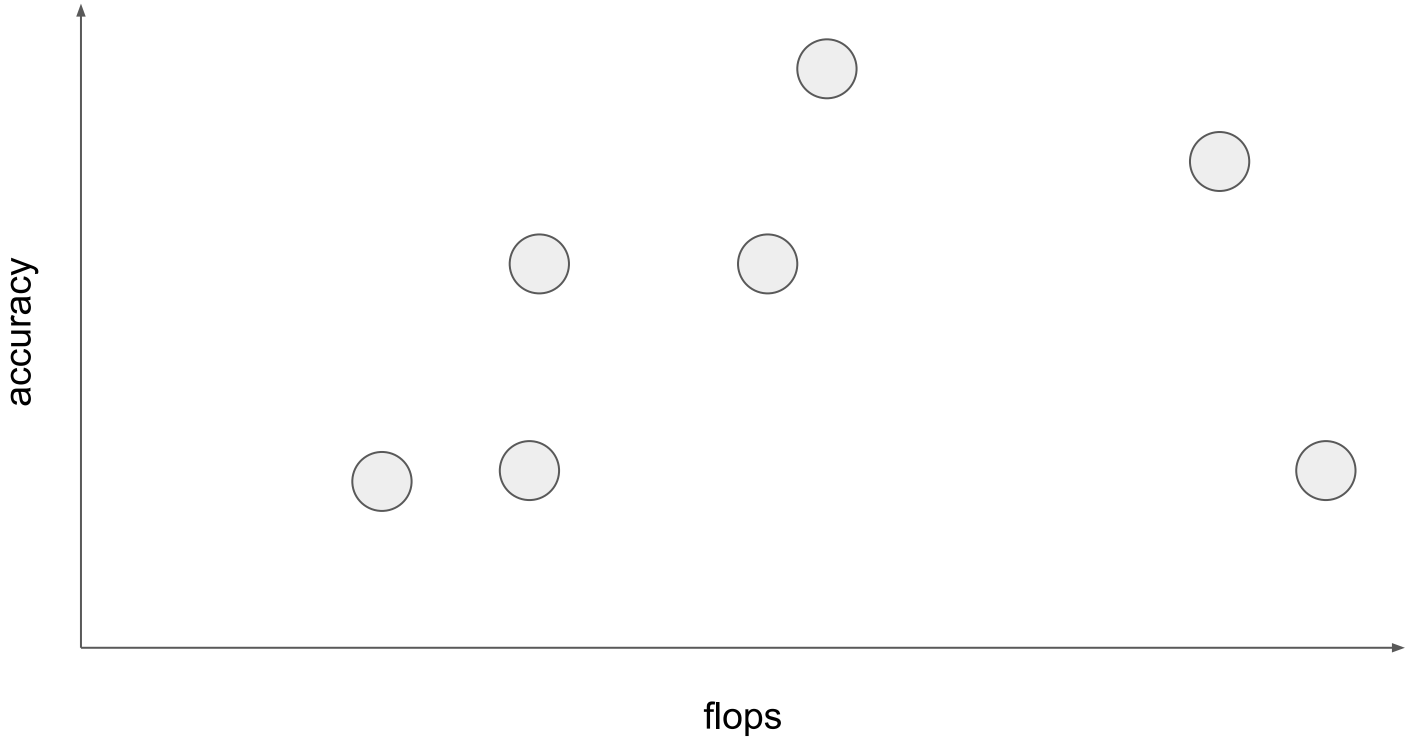
Net	Detalls	Elapsed Time	Accuracy Test	Flops	Params
Net_Linear	1 capa lineal	58.4s (~1min)	83.7%		
LeNet_tiny	4 vegades més petita que LeNet	129.2s (~2min)	87.4%		
LeNet	La que es feia servir per MNIST	536.0s (~9min)	91.2%		
FashionCNN	d'un de kaggle.. (similar a LeNet + batchnorm)	454.7s (~8min)	89.7%		
X1					
X2					
X3					

temps mesurats en cpu: macbook Pro 2,3 GHz Intel Core i9 de 8 nuclis

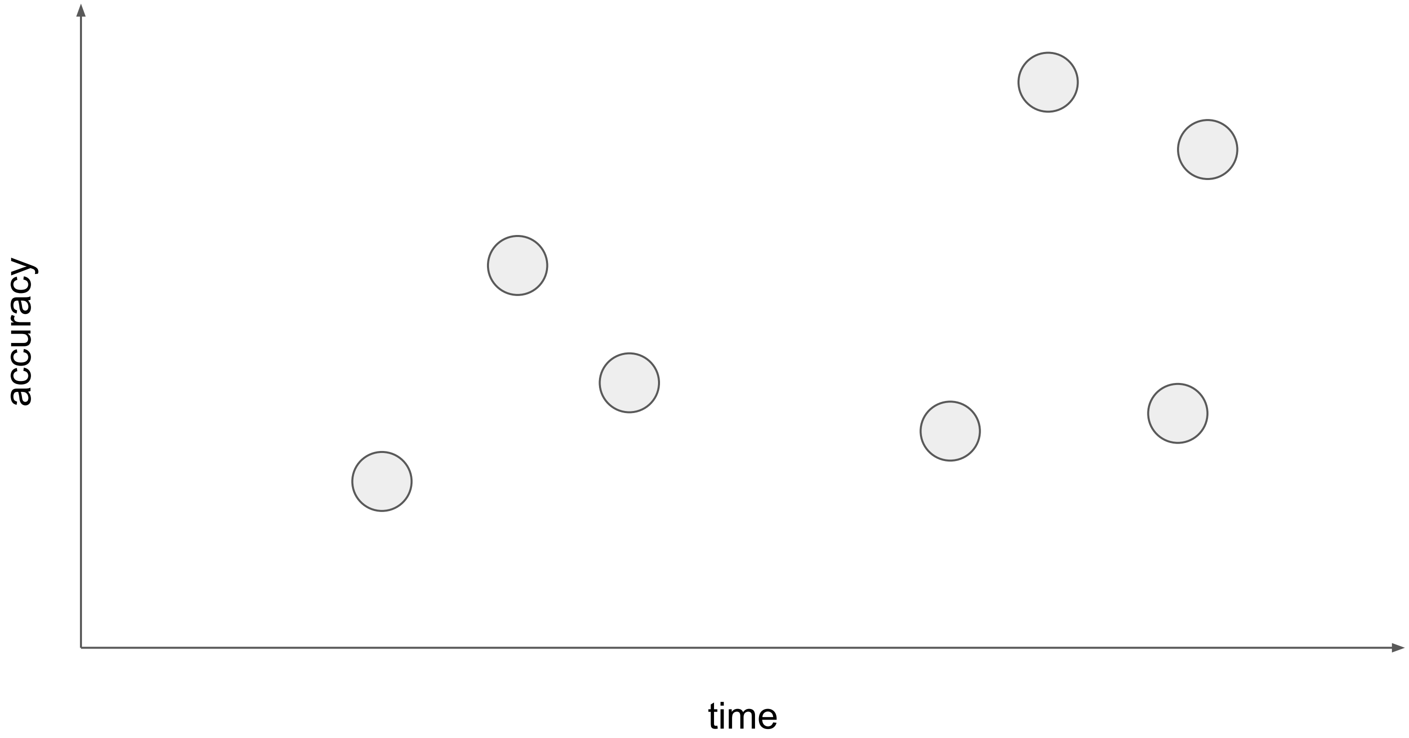
C.



c.



C.



Resum

A classe:

- Explicació de com es fa un Xarxa neuronal de Classificació Multi-categoria sobre BBDD Fashion MNIST

A entregar:

- **Jupyter Notebook** executat, mostrant les xarxes definides, entrenades i amb les gràfiques corresponents.