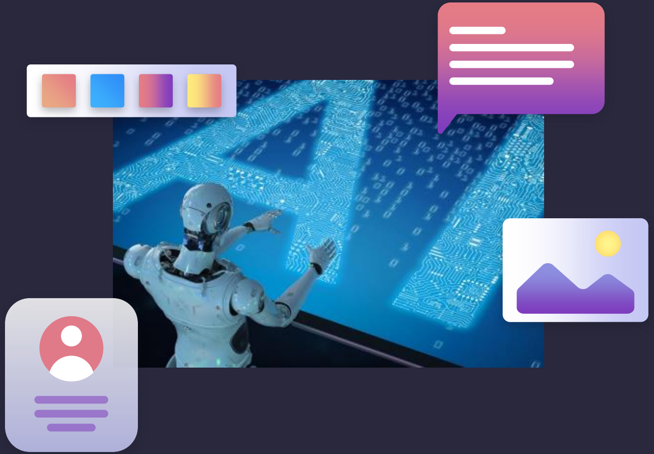
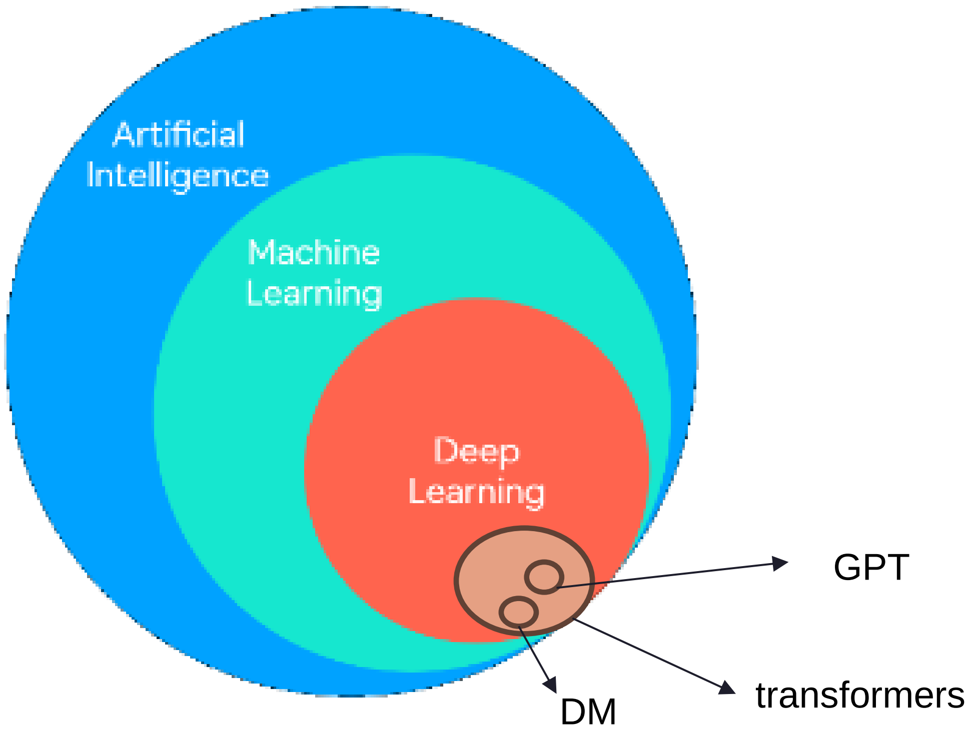
UT.6

VISION ARTIFICIAL
Previo: Sistemas inteligentes con PyTorch

Tensores: Fundamentos de PyTorch

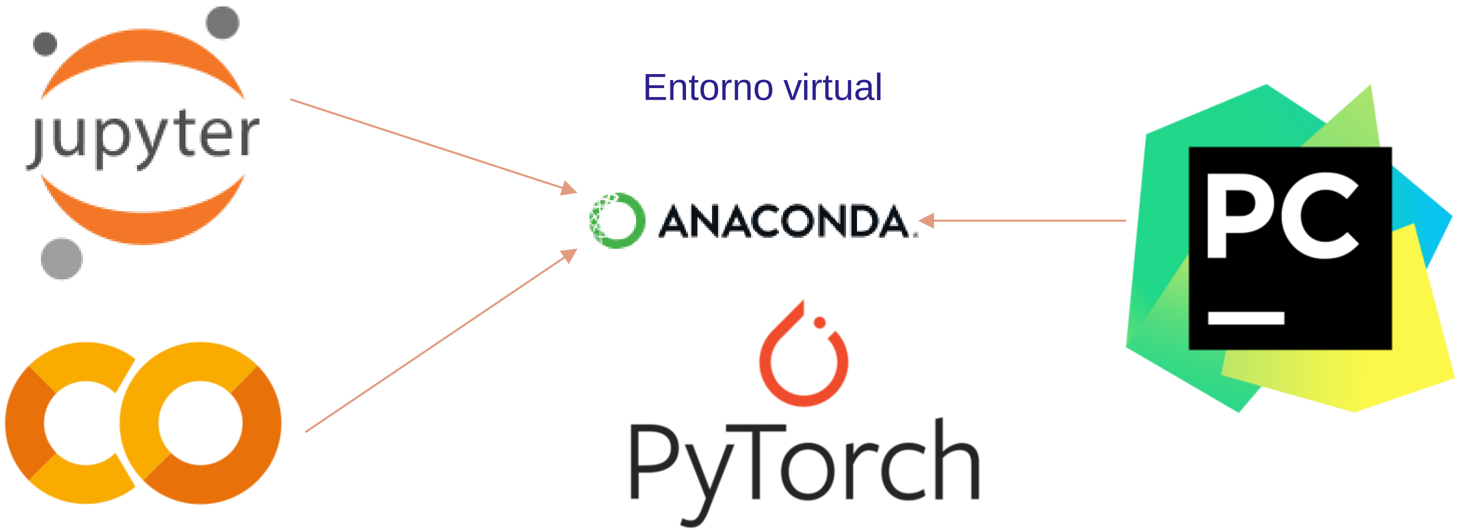



Artificial Intelligence
Machine Learning
Deep Learning
GPT
transformers
DM

Preparación del entorno:

Entorno de experimentación

Entorno de programación



Comienza con…

genera…

Programación tradicional

Entradas

Reglas (Algoritmo)

1. Trocear verduras
2. Sofreír verduras
3. Añadir marisco y carne
4. Sazonar
5. Añadir arroz
6. Cocer fuego fuerte 8mins
7. Cocer fuego medio 10mins
8. Reposar 5 mins

Salida

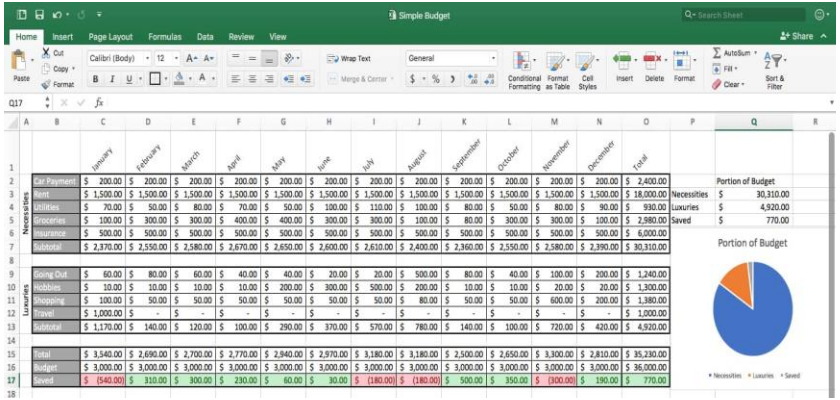
Machine Learning

Entradas

Reglas (Modelo)

1. Trocear verduras
2. Sofreír verduras
3. Añadir marisco y carne
4. Sazonar
5. Añadir arroz
6. Cocer fuego fuerte 8mins
7. Cocer fuego medio 10mins
8. Reposar 5 mins

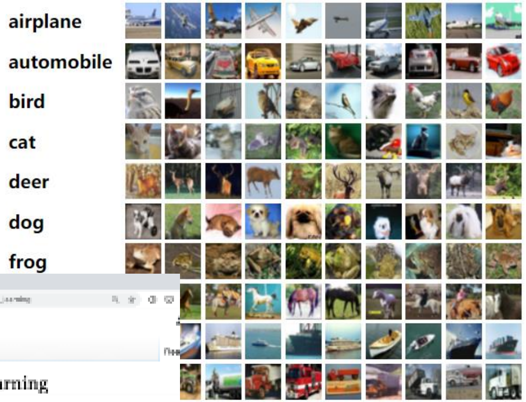
Machine Learning vs Deep Learning

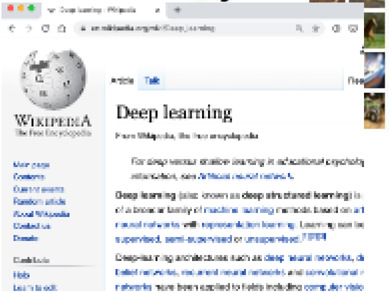


		January	February	March	April	May	June	July	August	September	October	November	December	Total
Necessities	Car Payment	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	2,400.00
	Rent	1,500.00	1,500.00	1,500.00	1,500.00	1,500.00	1,500.00	1,500.00	1,500.00	1,500.00	1,500.00	1,500.00	1,500.00	18,000.00
	Utilities	70.00	50.00	80.00	70.00	50.00	100.00	110.00	100.00	80.00	50.00	80.00	90.00	930.00
	Groceries	100.00	300.00	300.00	400.00	400.00	300.00	300.00	100.00	80.00	300.00	300.00	100.00	2,980.00
	Insurance	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	6,000.00
	Subtotal	2,370.00	2,550.00	2,580.00	2,670.00	2,650.00	2,600.00	2,610.00	2,400.00	2,360.00	2,550.00	2,580.00	2,390.00	30,310.00
Luxuries	Going Out	60.00	80.00	60.00	40.00	40.00	20.00	20.00	500.00	80.00	40.00	100.00	200.00	1,240.00
	Hobbies	10.00	10.00	10.00	10.00	200.00	300.00	500.00	200.00	10.00	10.00	20.00	20.00	1,300.00
	Shopping	100.00	50.00	50.00	50.00	50.00	50.00	50.00	80.00	50.00	50.00	600.00	200.00	1,380.00
	Travel	1,000.00	-	-	-	-	-	-	-	-	-	-	-	1,000.00
	Subtotal	1,170.00	140.00	120.00	100.00	290.00	370.00	570.00	780.00	140.00	100.00	720.00	420.00	4,920.00
	Total	3,540.00	2,690.00	2,700.00	2,770.00	2,940.00	2,970.00	3,180.00	3,180.00	2,500.00	2,650.00	3,300.00	2,810.00	35,230.00
	Budget	3,000.00	3,000.00	3,000.00	3,000.00	3,000.00	3,000.00	3,000.00	3,000.00	3,000.00	3,000.00	3,000.00	3,000.00	36,000.00
	Saved	(540.00)	310.00	300.00	230.00	60.00	30.00	(180.00)	(180.00)	500.00	350.00	(300.00)	190.00	770.00

	Portion of Budget
Necessities	30,310.00
Luxuries	4,920.00
Saved	770.00

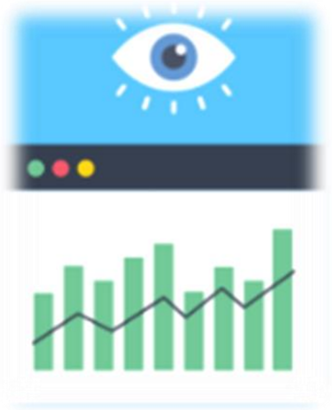
Datos estructurados



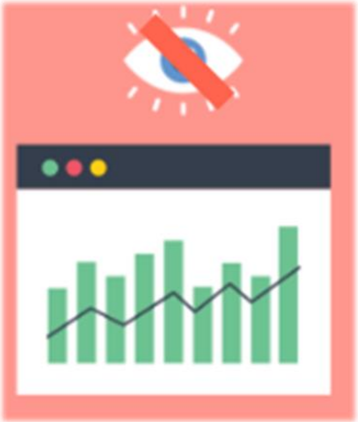


Datos no estructurados

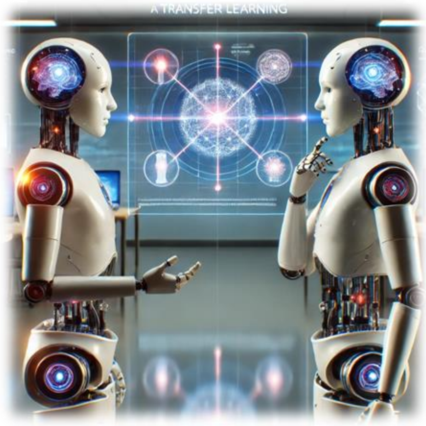
Tipos de aprendizaje



Aprendizaje supervisado



Aprendizaje NO supervisado



Transfer Learning

Machine Learning vs Deep Learning

Algoritmos fundacionales

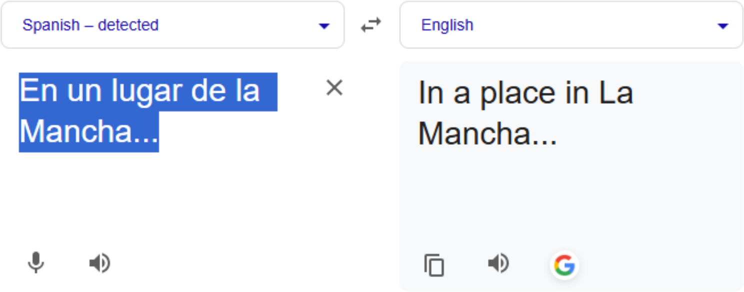
Random Forest
Modelos Gradient Boosted
Naive Bayes
Vecino más cercano (K means)
SVMs
….

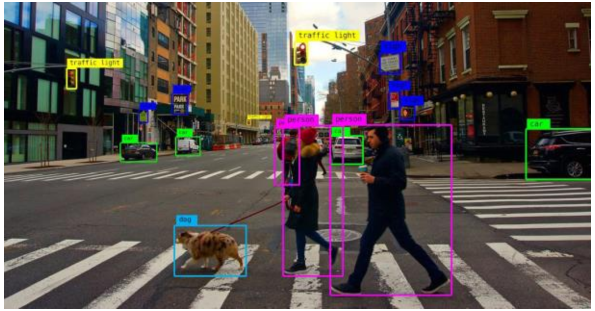
Redes neuronales
Redes convolucionales
Redes recurrentes
LSTM
Transformers
….

Depende de cómo representes la información puedes utilizar distintos tipos de algoritmos

Datos estructurados ⟷ Datos no estructurados

Deep Learning: Casos de uso





PyTorch ==

¿Qué me aporta pytorch frente a tensorflow?

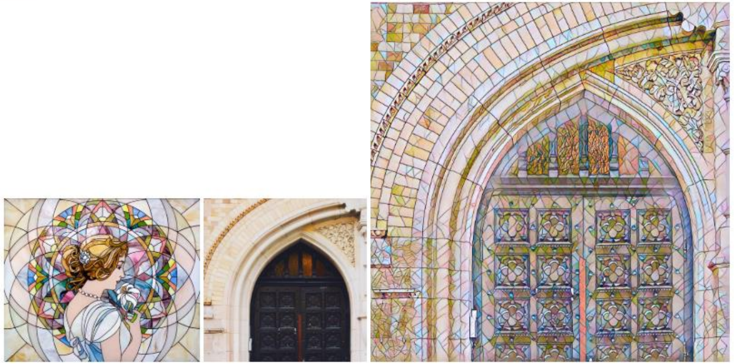
https://github.com/pytorch

https://github.com/pytorch/examples

fast-neural-style 🌆 🚀

This repository contains a pytorch implementation of an algorithm for artistic style transfer. The algorithm can be used to mix the content of an image with the style of another image. For example, here is a photograph of a door arch rendered in the style of a stained glass painting.

The model uses the method described in Perceptual Losses for Real-Time Style Transfer and Super-Resolution along with Instance Normalization. The saved-models for examples shown in the README can be downloaded from here.



Superresolution using an efficient sub-pixel convolutional neural network

This example illustrates how to use the efficient sub-pixel convolution layer described in "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network" - Shi et al. for increasing spatial resolution within your network for tasks such as superresolution.

Time Sequence Prediction

This is a toy example for beginners to start with. It helps learn both PyTorch and time sequence prediction. Two LSTMCell units are used in this example to learn some sine wave signals starting at different phases. After learning the sine waves, the network tries to predict the signal values in the future. The results are shown in the picture below.

Deep Convolution Generative Adversarial Networks

This example implements the paper Unsupervised Representation Learning with Deep Convolutional Generative A

Language Translation

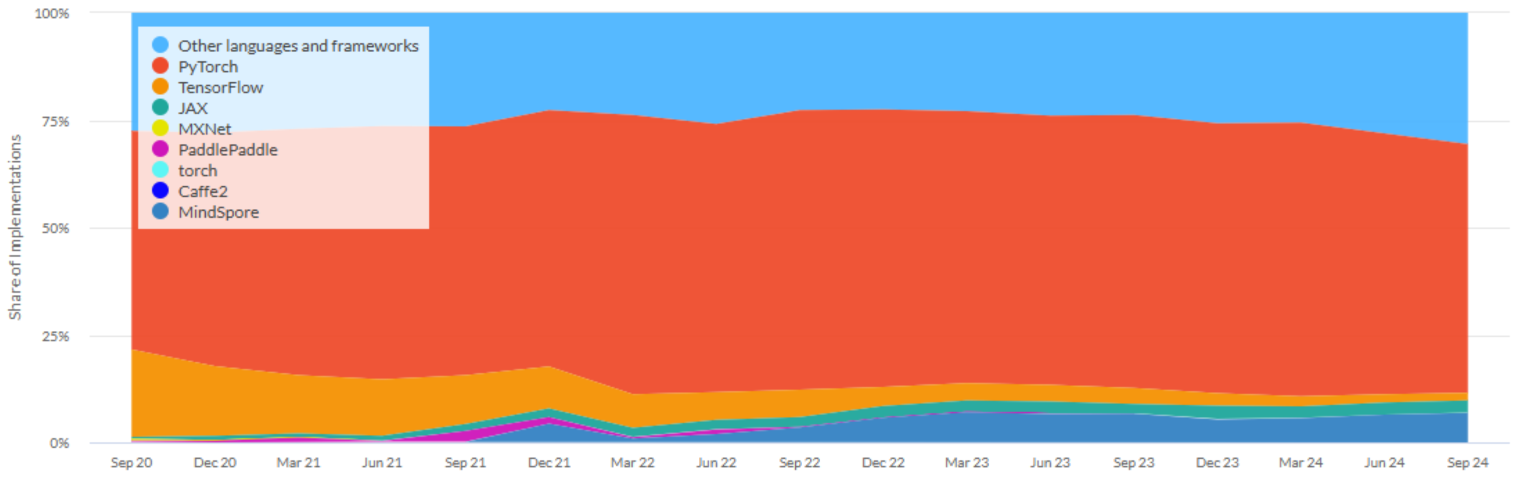
This example shows how one might use transformers for language translation. In particular, this implementation is loosely based on the Attention is All You Need paper.

Trends

Quarter 2020-09-01 to 2024-09-30

Frameworks

Paper Implementations grouped by framework



¿Quién usa pytorch?

PyTorch at Tesla - Andrej Karpathy, Tesla

PyTorch
Aug 7, 2020 · 11 min read · Listen

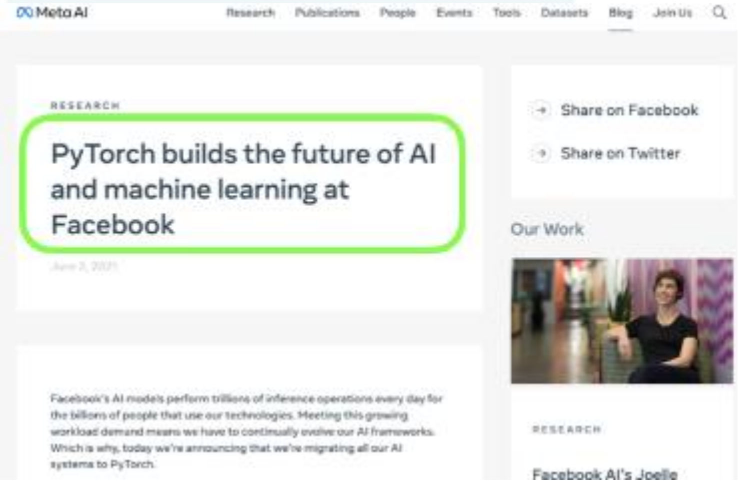
AI for AG: Production machine learning for agriculture

Author: Chris Padwick, Director of Computer Vision and Machine Learning at Blue River Technology

OpenAI Standardizes on PyTorch

We are standardizing OpenAI's deep learning framework on PyTorch. In the past, we implemented projects in many frameworks depending on their relative strengths. We've now chosen to standardize to make it easier for our team to create and share optimized implementations of our models.

OpenAI PyTorch



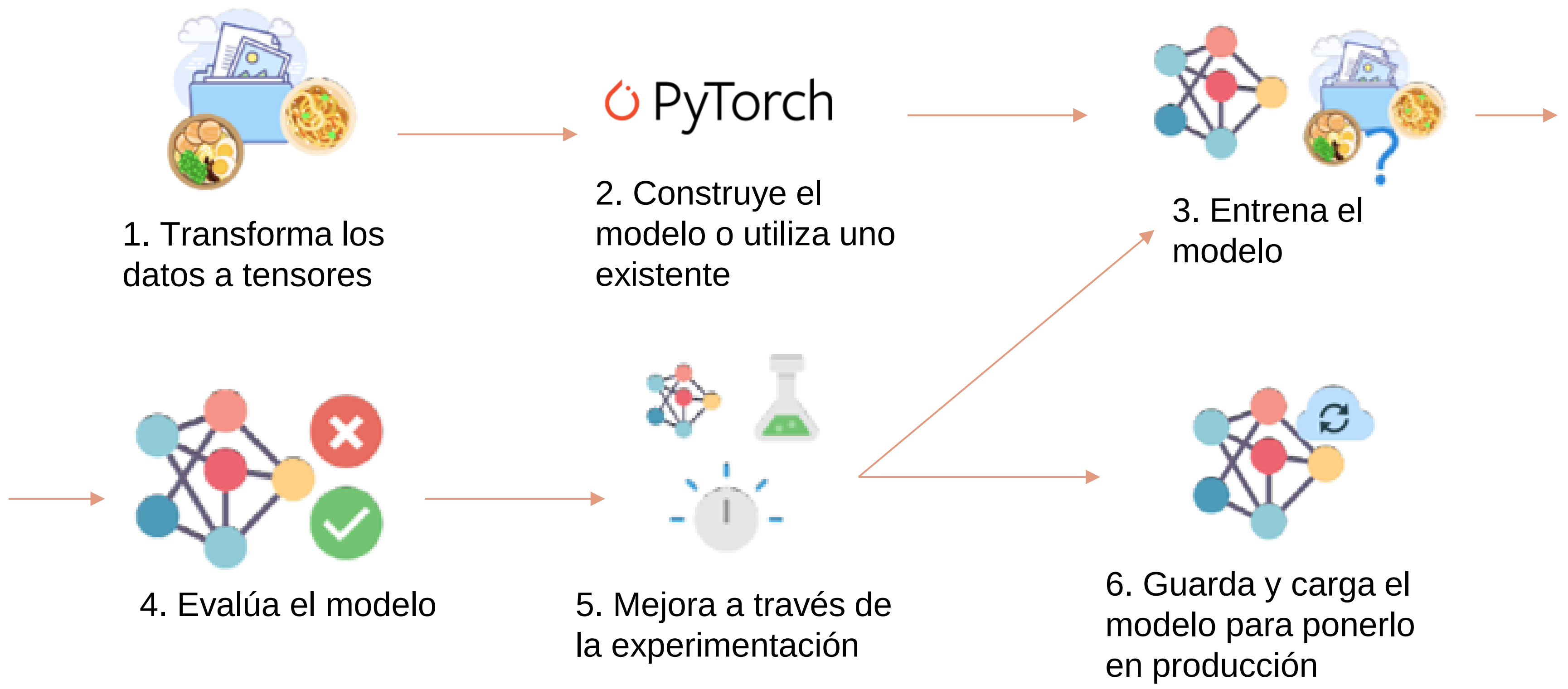
https://developers.google.com/machine-learning/guides/rules-of-ml

Rule #1: Don’t be afraid to launch a product without machine learning

if you can build a simple rule-based system that doesn’t require Machine learning, do that

Rule #2: First, design and implement metrics
Rule #3: Choose machine learning over a complex heuristic.
Rule #4: Keep the first model simple and get the infrastructure right.
Rule #5: Test the infrastructure independently from the machine learning
Rule #6: Be careful about dropped data when copying pipelines

….


PyTorch
1. Transforma los datos a tensores
2. Construye el modelo o utiliza uno existente
3. Entrena el modelo
4. Evalúa el modelo
5. Mejora a través de la experimentación
6. Guarda y carga el modelo para ponerlo en producción

Vamos a por el cuaderno de jupyter 1

Módulo 1. Fundamentos de PyTorch

¿Qué es PyTorch?

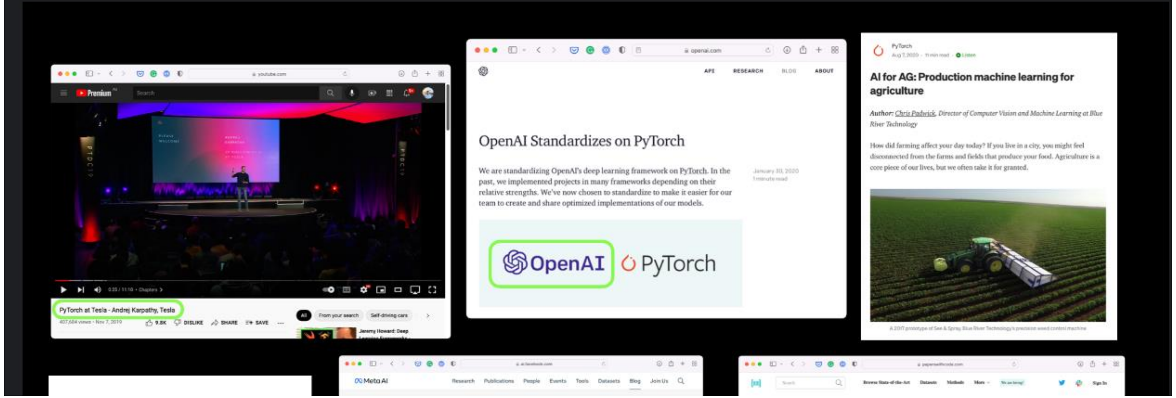
PyTorch es un framework de aprendizaje automático y profundo de código abierto.

¿Para qué se puede usar PyTorch?

PyTorch te permite manipular y procesar datos y escribir algoritmos de aprendizaje automático usando código en Python.

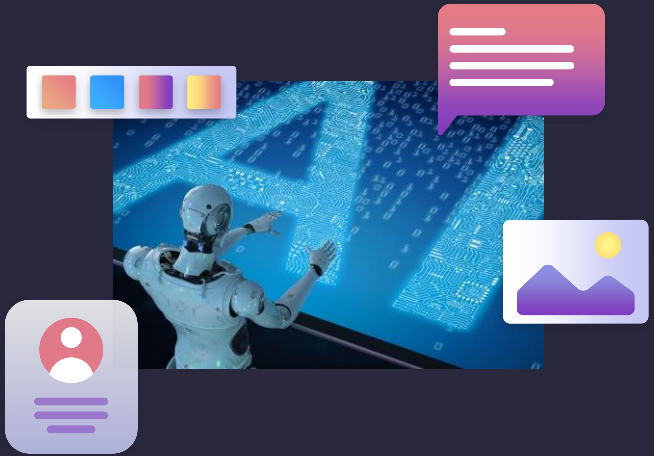
¿Quién usa PyTorch?

Muchas de las compañías de tecnología más grandes del mundo, como Meta (Facebook), Tesla y Microsoft, así como empresas de investigación en inteligencia artificial como OpenAI usan PyTorch para impulsar la investigación y llevar el aprendizaje automático a sus productos.

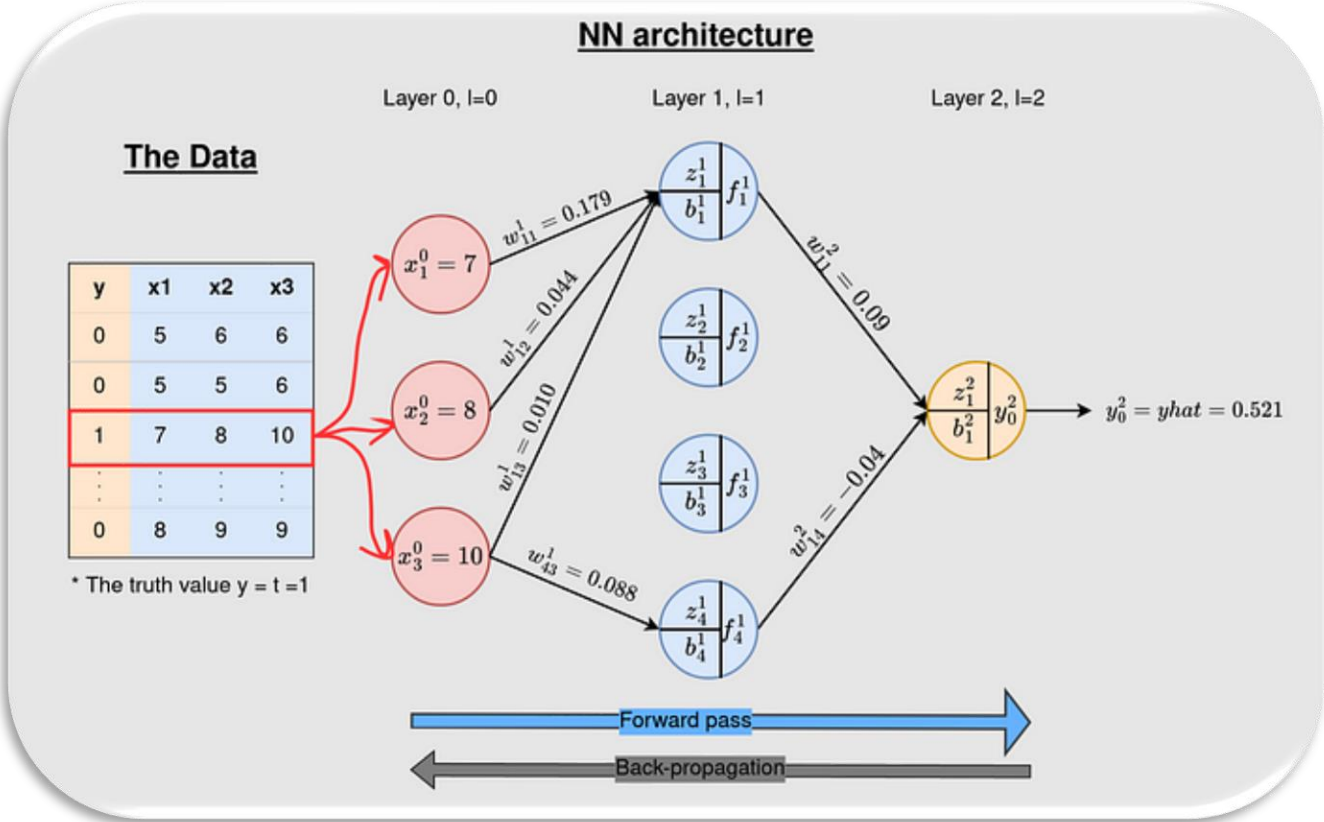


Construcción de redes neuronales con PyTorch

El flujo de trabajo de PyTorch



Recordatorio!



Tensorflow vs Pytorch

```
model_7 = tf.keras.Sequential([
  tf.keras.layers.Input(shape=[2]),
  tf.keras.layers.Dense(4, activation=tf.keras.activations.relu),
  tf.keras.layers.Dense(4, activation=tf.keras.activations.relu),
  tf.keras.layers.Dense(1, activation=tf.keras.activations.sigmoid)
])

model_7.compile(loss=tf.keras.losses.binary_crossentropy,
                optimizer=tf.keras.optimizers.Adam(),
                metrics=['accuracy'])

history = model_7.fit(X, y, epochs=100, verbose=0)
```

```
class ClasificadorBinario(torch.nn.Module):  2 usages
    def __init__(self):
        super(ClasificadorBinario, self).__init__()
        self.layer1 = torch.nn.Linear( in_features: 2, out_features: 4)
        self.layer2 = torch.nn.Linear( in_features: 4, out_features: 4)
        self.layer3 = torch.nn.Linear( in_features: 4, out_features: 1)
        self.relu=torch.nn.ReLU()
        self.sigmoid=torch.nn.Sigmoid()
        self.loss=torch.nn.MSELoss()
        self.optimizer = torch.optim.Adam(self.parameters(), lr=0.01)

    def forward(self,x):  2 usages
        x=self.layer1(x)
        x=self.relu(x)
        x=self.layer2(x)
        x=self.relu(x)
        x=self.layer3(x)
        x=self.sigmoid(x)
        return x
```

Forward =predict

```
def fit(self,X,y,epocas):  1 usage
    for i in range(epocas):
        loss_epoca=0
        for xi, yi in zip(X,y):
            xi=torch.tensor(xi,dtype=torch.float).view(1,-1)
            yi=torch.tensor(yi,dtype=torch.float).view(1,-1)
            self.optimizer.zero_grad()
            output=self.forward(xi)
            loss=self.loss(output,yi)
            loss_epoca=loss_epoca+loss.item()
            loss.backward()
            self.optimizer.step()
        print(f"pérdida media en la época [{i}]:{loss_epoca/len(X)}")
```

El flujo de trabajo:

1. Preparación

```
Inicializar modelo
Definir función de pérdida
Definir optimizador
Cargar datos de entrenamiento y validación
Configurar hiperparámetros: epochs, batch_size,
learning_rate
```

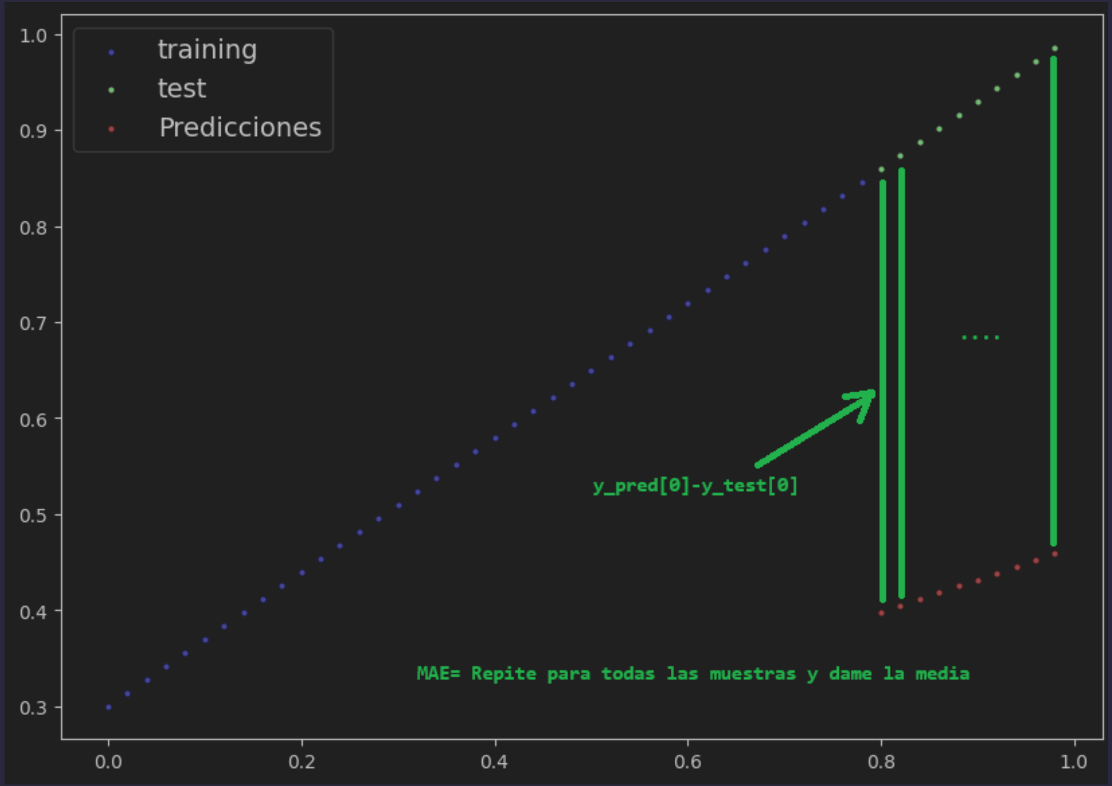
2. Entrenamiento

```
Para cada epoch en range(num_epochs):
    model.train()
    Para cada batch en dataloader de entrenamiento:
        optimizador.zero_grad()
        entradas, etiquetas = batch
        predicciones = modelo(entradas)
        pérdida = función_de_pérdida(predicciones, etiquetas)
        pérdida.backward()
        optimizador.step()
```

3. Evaluación / Validación

```
model.eval()
Con torch.no_grad():  # Desactiva el cálculo de gradientes
  Para cada batch en dataloader de validación:
      entradas, etiquetas = batch
      predicciones = modelo(entradas)
      pérdida_val = función_de_pérdida(predicciones, etiquetas)
      Guardar métricas de validación (precisión, pérdida, etc.)
```

Función de pérdida



El optimizador - cálculo automático de los gradientes: **autograd**

Es un sistema automático de diferenciación que calcula los gradientes de las operaciones realizadas sobre tensores

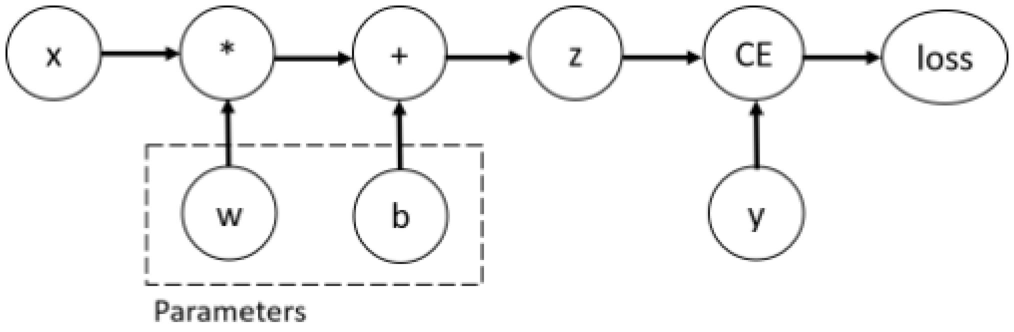
Al realizar operaciones con tensores que tienen `requires_grad=True`, PyTorch construye dinámicamente un gráfico computacional que registra las dependencias entre las operaciones. Luego, mediante el método `backward()`, el autograd recorre este gráfico para calcular los gradientes de manera eficiente, permitiendo optimizar modelos de aprendizaje profundo.

https://pytorch.org/tutorials/beginner/introyt/autogradyt_tutorial.html

El modelo en modo.train SI realiza autograd
en modo evaluación NO realiza autograd

El grafo computación de AutoGrad: *BackPropagation*

```
import torch

x = torch.ones(5)  # tensor de entrada
y = torch.zeros(3)  # salida esperada
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss =
torch.nn.functional.binary_cross_entropy_with_logits(z, y)
```



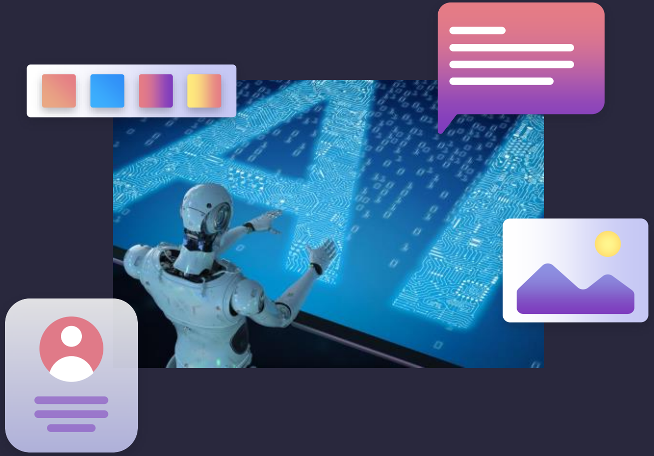
Vamos a por el cuaderno pytorch_autograd

```python
def print_graph(g, level=0):
    if g is None:
        return
    print("\t" * level, g)
    for subg in g.next_functions:
        print_graph(subg[0], level + 1)


# Imprimir el grafo
print_graph(out.grad_fn)

[11]
```

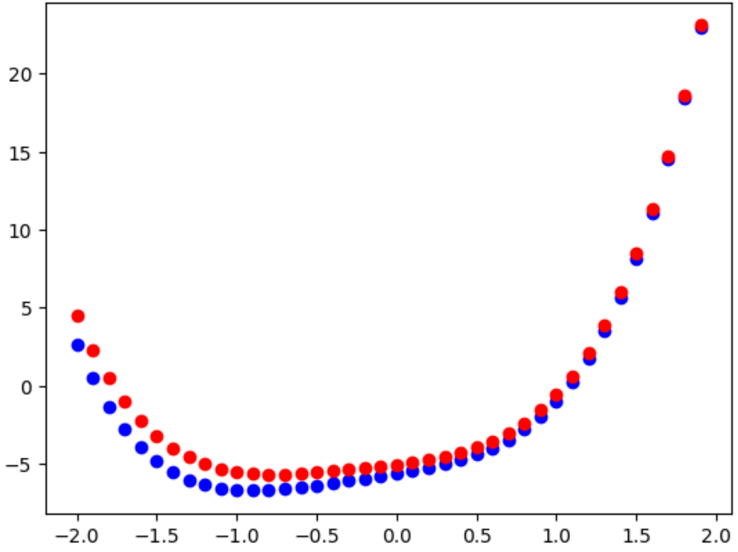
```
 <SumBackward0 object at 0x7f87169337f0>
     <AddBackward0 object at 0x7f8716933d00>
         <MulBackward0 object at 0x7f8716933580>
             <SinBackward0 object at 0x7f87169335e0>
```

Construcción de redes neuronales con PyTorch

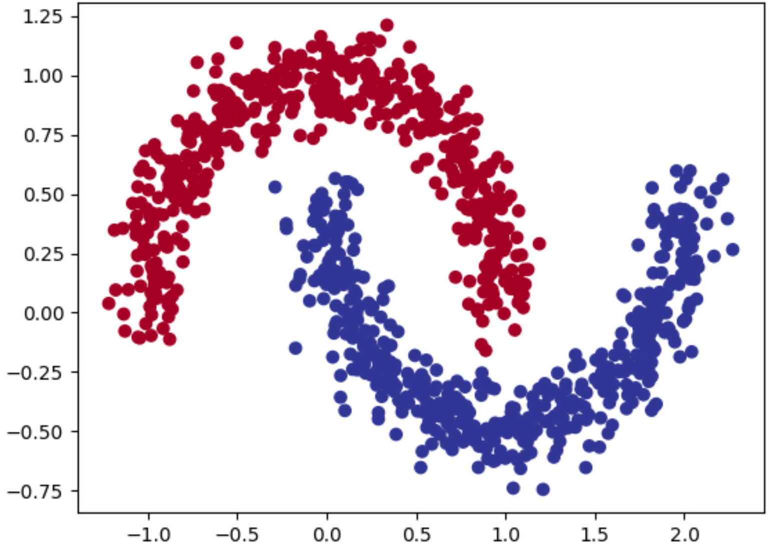
Regresión y clasificación con redes neuronales



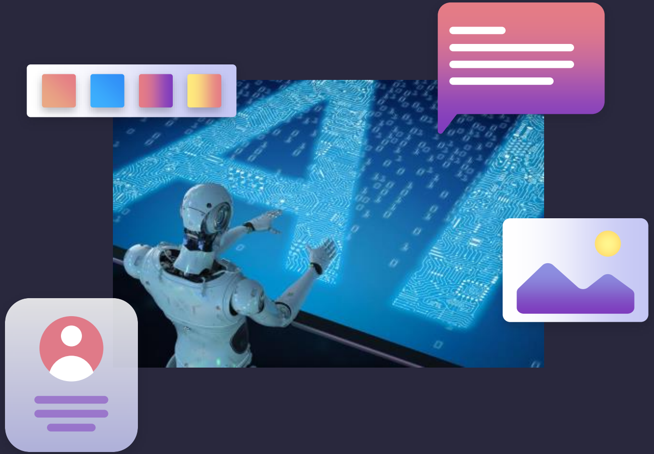
Ejercicio 1: Regresión de un polinomio



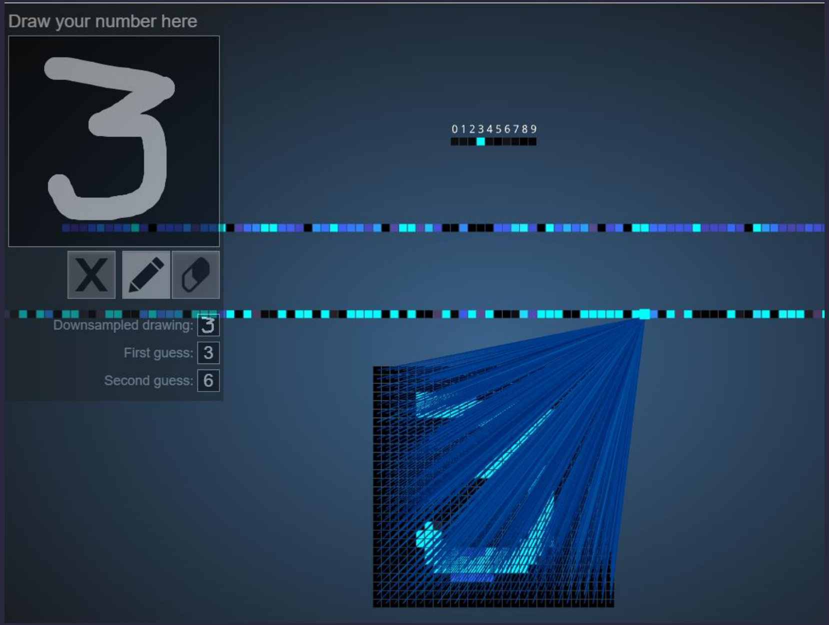
Ejercicio 2: Clasificación: make_moons

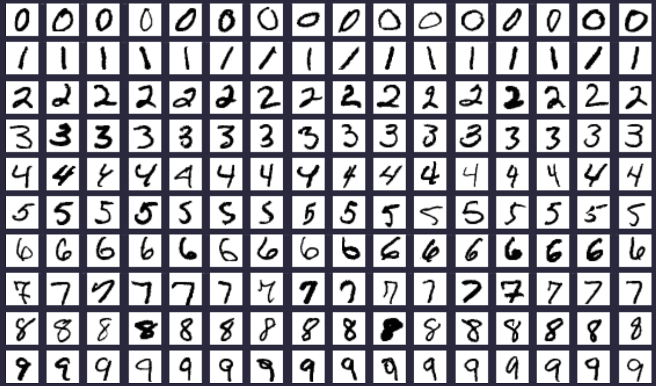


Manejos de datasets y loaders en PyTorch



Un problema clásico: Datasets y DataLoaders con Dígitos MNIST https://adamharley.com/nn_vis/mlp/2d.html





https://pytorch.org/vision/main/datasets.html

- Incorpora datasets “Built-in” para problemas de ML y DL
- Datos para visión, texto y audio.
- Soporte para datasets personalizados.
- Optimizado: Descarga y preprocesamiento eficiente
- Con DataLoader permite la gestión de batches
- Integrado con torchvision.transforms para aumentación de datos.

CLASS torch.utils.data.Dataset [SOURCE]

An abstract class representing a `Dataset`.

All datasets that represent a map from keys to data samples should subclass it. All subclasses should overwrite `__getitem__()`, supporting fetching a data sample for a given key. Subclasses could also optionally overwrite `__len__()`, which is expected to return the size of the dataset by many `Sampler` implementations and the default options of `DataLoader`. Subclasses could also optionally implement `__getitems__()`, for speedup batched samples loading. This method accepts list of indices of samples of batch and returns list of samples.

NOTE

`DataLoader` by default constructs an index sampler that yields integral indices. To make it work with a map-style dataset with non-integral indices/keys, a custom sampler must be provided.

EJEMPLO DE CLASIFICACIÓN:BUILT-IN MNIST DIGITS DATASET

```python
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch


# Transformaciones: convertir a tensor y normalizar
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Media y desviación estándar
])

# Cargar conjunto de datos MNIST (Dígitos)
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)

# Crear loaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

https://pytorch.org/vision/main/datasets.html

DataLoader

- divide en lotes (batches)
- aplicar transformaciones o preprocesamientos al acceder a un dato en concreto
- carga los datos en memoria usando múltiples procesos

```
batch_size

shuffle

num_workers
```

```python
for batch in train_loader:
    procesar(batch)

with torch.no_grad():
    for batch in test_loader:
        images, labels = batch
        evaluar(images,labels)
```

Creando un Dataset personalizado

```
class CustomImageDataset(Dataset):
  def __init__(self, image_dir, transform=None):
    self.image_dir = image_dir
    self.transform = transform
    self.image_paths = [os.path.join(image_dir, img) for img in os.listdir(image_dir)]

  def __len__(self):
    return len(self.image_paths)

  def __getitem__(self, idx):
    img_path = self.image_paths[idx]
    image = Image.open(img_path)
    if self.transform:
      image = self.transform(image)
    return image
```

```
t = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

dataset = CustomImageDataset("path", transform=t)
dataloader = DataLoader(dataset, batch_size=32)
```