Clase 3

Introducción a PyTorch

Enzo Ferrante

eferrante@sinc.unl.edu.ar

@enzoferrante

Universidad de SanAndrés

UNL CONICET sinc(i)

Slides basadas en la presentación de Joaquín Seia

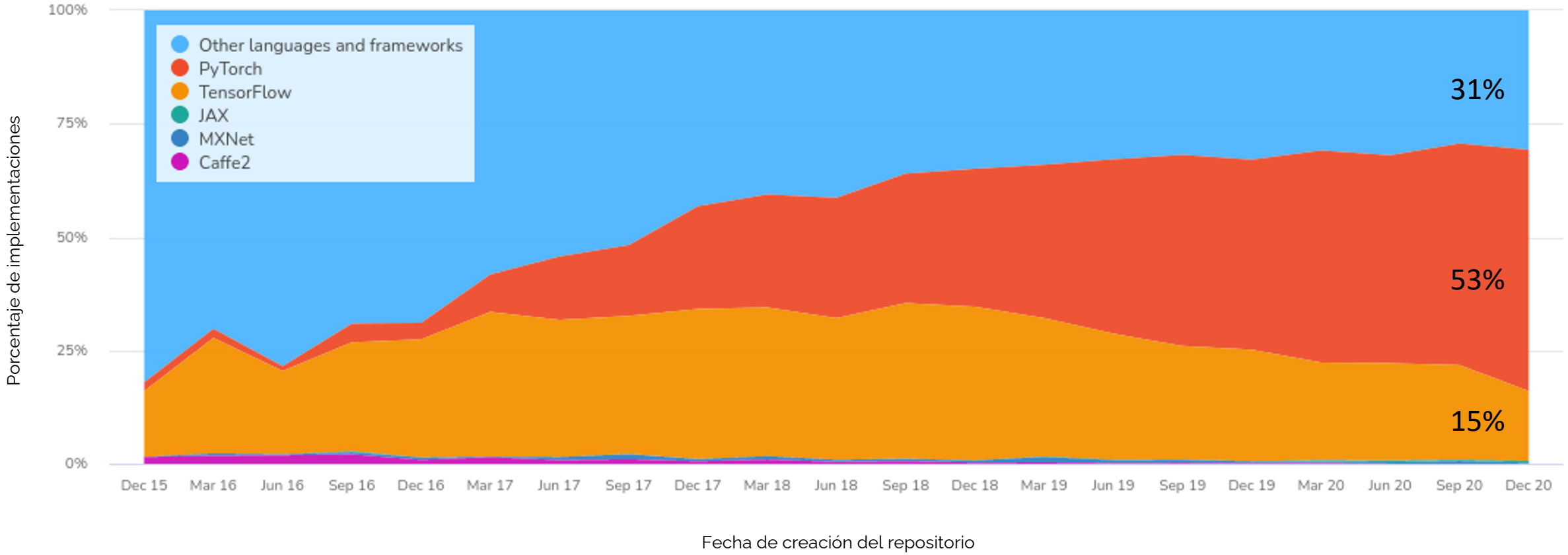
“PyTorch es un framework de deep learning open source desarrollado para ser flexible y modular para la investigación, pero con la estabilidad y el soporte necesarios para su uso en producción.” *

PYTORCH

facebook
Artificial Intelligence Research

Adopción de PyTorch en los últimos años

Implementaciones de papers agrupadas por framework.*



Fuentes: * https://ai.facebook.com/tools/ ** https://paperswithcode.com/trends

PYTORCH

Tensor
(array multidimensional)

↓

Operaciones
Numpy

Optimización

↓

Grafos de cómputo
dinámicos
+
Autograd

Soporte para
cómputo acelerado
en GPU

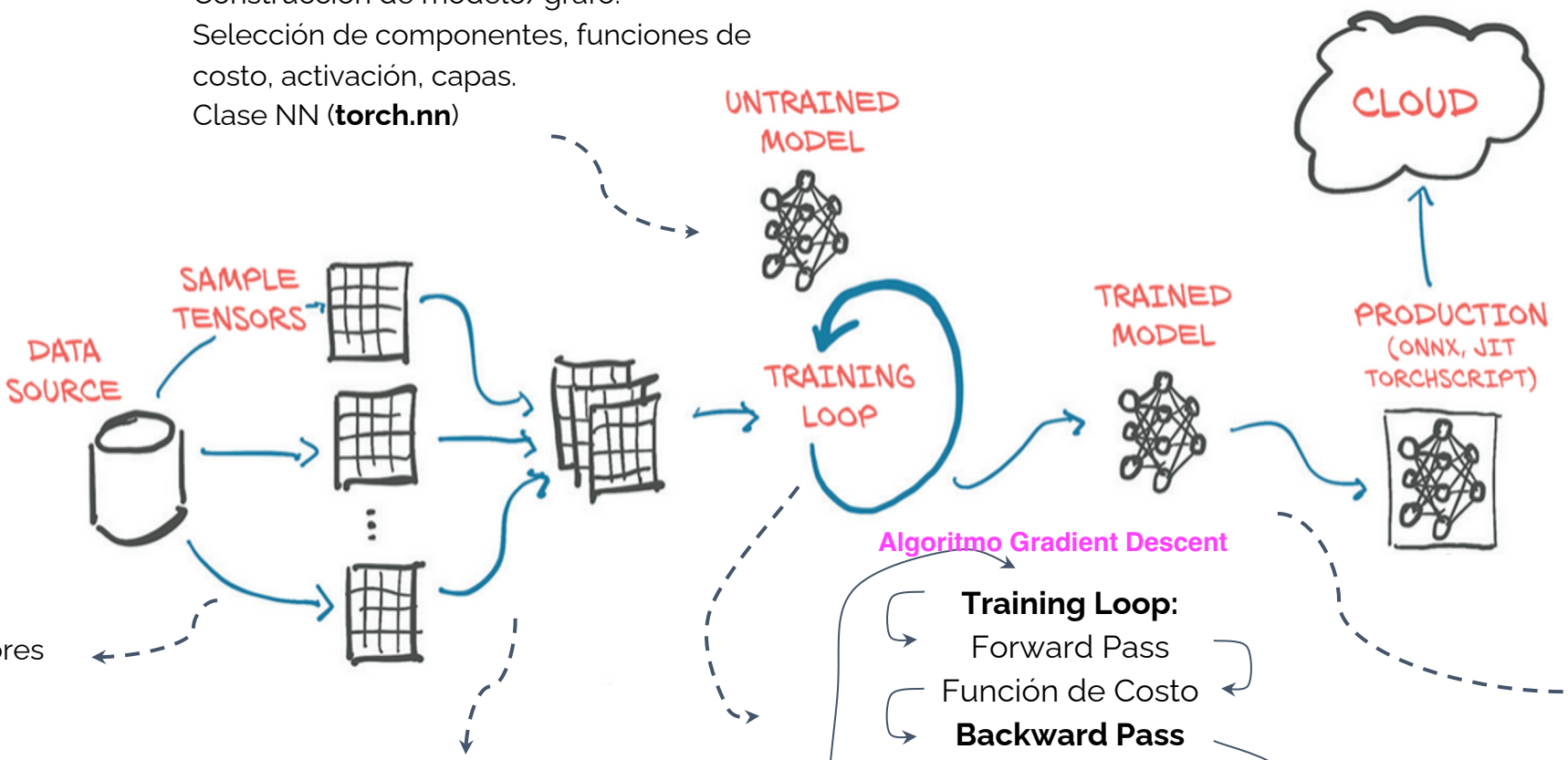
Librería de alto desempeño que permite realizar optimización mediante **derivación automática** para desarrollos de cómputo científico en Python.

↓

Deep Learning

PYTORCH

Construcción de modelo/grafo.
Selección de componentes, funciones de costo, activación, capas.
Clase NN (**torch.nn**)



Modelo pre-compilado (grafos estáticos).
Compatibilidad con Python y C++
TorchScript

Levantar datos y convertirlos a tensores (**toch.Tensor**).
Clase Dataset (**toch.utils.data**)

Volverlo eficiente, paralelizado y organización en grupos de datos (batches)
Clase DataLoader (**torch.utils.data.DataLoader**)

Algoritmo Gradient Descent

Training Loop:
Forward Pass
Función de Costo
Backward Pass
Clase Autograd
(**torch.autograd**)
Actualización de pesos
Clase Optimizer
(**torch.optim**)

Tensores en PyTorch

```python
import torch
torch.tensor([[2, 3, 5], [1, 2, 9]])
```

```
tensor([[ 2,  3,  5],
        [ 1,  2,  9]])
```

```python
torch.rand(2, 2)
```

```
tensor([[ 0.0374, -0.0936],
        [ 0.3135, -0.6961]])
```

```python
a = torch.rand((3, 5))
a.shape
```

```
torch.Size([3, 5])
```

```python
import numpy as np
np.array([[2, 3, 5], [1, 2, 9]])
```

```
array([[ 2,  3,  5],
       [ 1,  2,  9]])
```

```python
np.random.rand(2, 2)
```

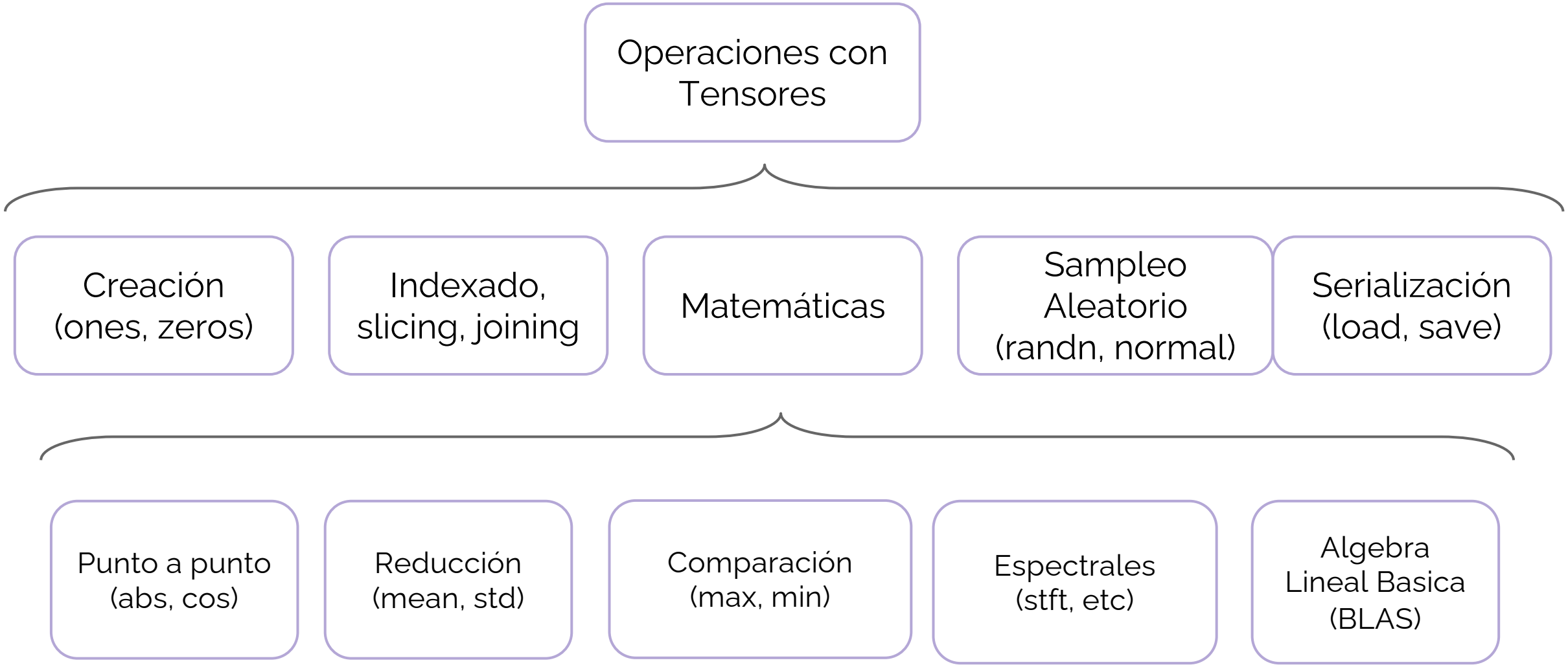
```
array([[ 0.0374, -0.0936],
       [ 0.3135, -0.6961]])
```

```python
a = np.random.randn(3, 5)
a.shape
```

```
(3, 5)
```

Extraído de la presentación ”Introduction to PyTorch” de Ismail Elizi

Tensores en PyTorch



Autograd

”Paquete de PyTorch que implementa diferenciación automática para funciones escalares arbitrarias por medio de la construcción de grafos computacionales dinámicos”

Tensores en PyTorch

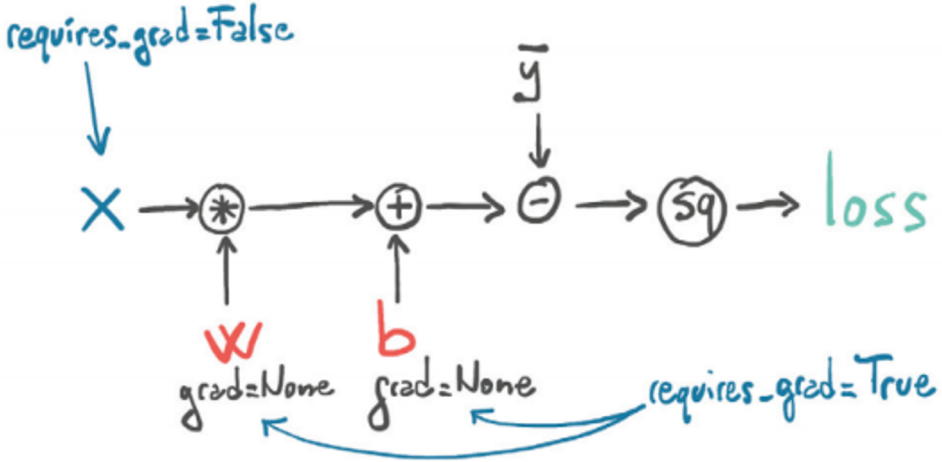
- Computar operaciones en forma paralela en GPU

- Distribuir operaciones en múltiples máquinas

- **Mantener un seguimiento del grafo de operaciones que les dan origen**.

```
c = torch.rand(3).cuda()
c = torch.rand(3).to('cuda')
```

```
c = torch.rand(3).cpu()
c = torch.rand(3).to('cpu')
```




```
params = torch.tensor([1.0, 0.0], requires_grad=True)
params
```

```
tensor([1., 0.], requires_grad=True)
```

```
print(params.grad)
```

```
None
```

Dynamic Computational graph

```python
import torch

# Pasada forward
x = torch.tensor([1.], requires_grad=True)
y = torch.tensor([2.], requires_grad=False)
z = x * y

print(x)
print(y)
print(z)
```

```
tensor([1.], requires_grad=True)
tensor([2.])
tensor([2.], grad_fn=<MulBackward0>)
```

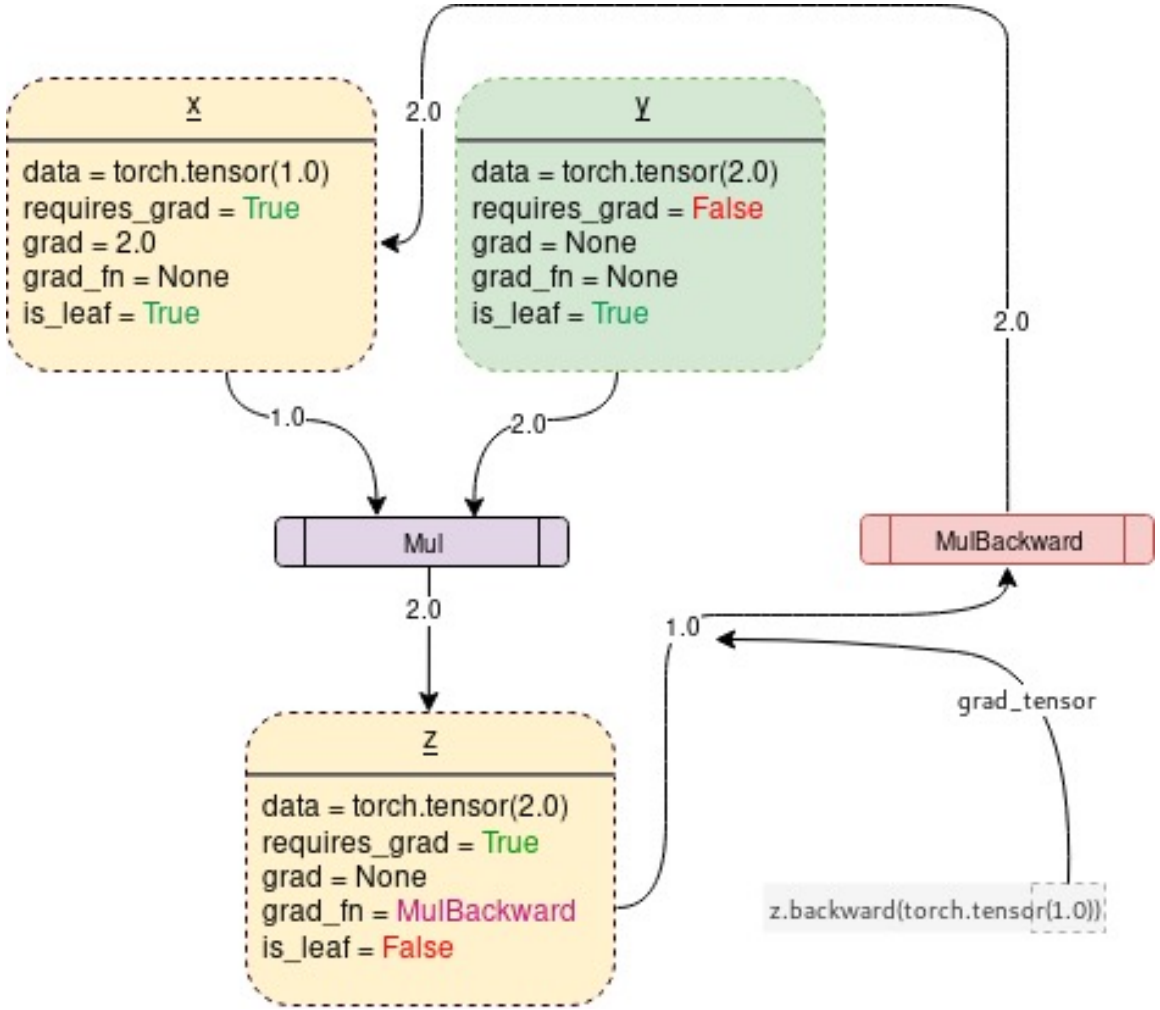


Imagen extraída del artículo https://towardsdatascience.com/pytorch-autograd-understanding-the-heart-of-pytorchs-magic-2686cd94ec95

Dynamic Computational graph

```python
import torch

# Pasada forward
x = torch.tensor([1.], requires_grad=True)
y = torch.tensor([2.], requires_grad=False)
z = x * y

print(x)
print(y)
print(z)
```

```
tensor([1.], requires_grad=True)
tensor([2.])
tensor([2.], grad_fn=<MulBackward0>)
```

"Desde donde empiezo a retropropagar"

```python
# Pasada backward
z.backward()
```

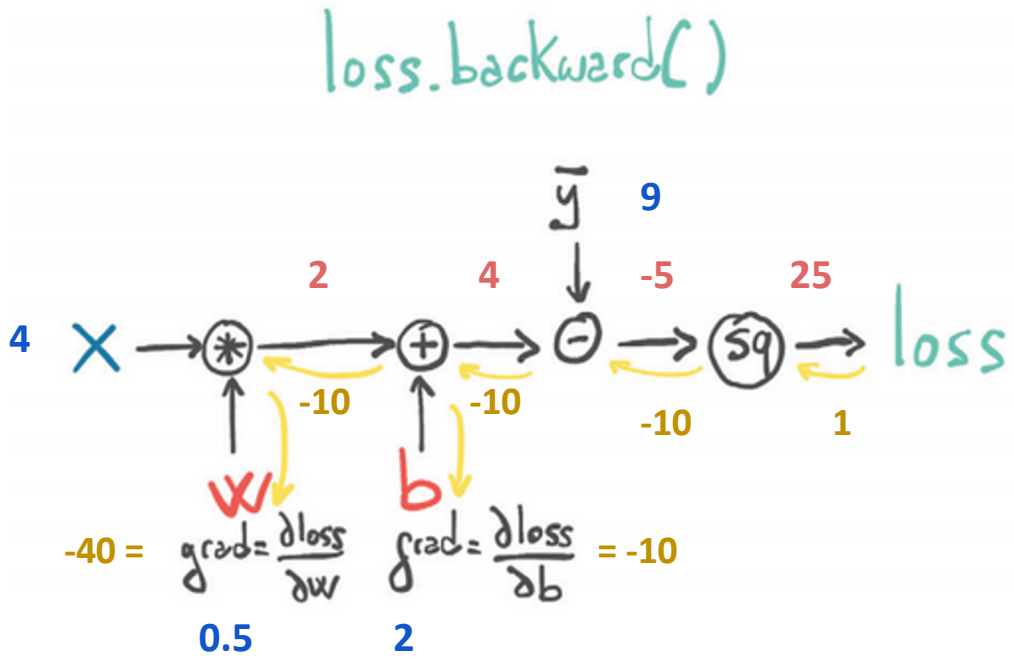
```python
#Cálculo del gradiente
print("dz/dx: " + str(x.grad))
print("dz/dy: " + str(y.grad))
```

```
dz/dx: tensor([2.])
dz/dy: None
```

Autograd

```python
# Definimos el modelo y la función de costo
def model(dato_ej, w, b):
    return w * dato_ej + b

def loss_fn(pred, real):
    squared_diffs = (pred - real)**2
    return squared_diffs.mean()

# Definimos el ejemplo y los parametros
dato_ej = torch.tensor([4.], requires_grad=False)
w = torch.tensor([0.5], requires_grad=True)
b = torch.tensor([2.], requires_grad=True)
print(w.grad, b.grad)

# Forward pass
resultado_ej = model(dato_ej, w, b)
loss = loss_fn(resultado_ej, 9.)

# Backward pass
loss.backward()
print(w.grad, b.grad)
```

```
None None
tensor([-40.]) tensor([-10.])
```



OJO: ¡El método backward() acumula los gradientes en los nodos hoja!

OJO, EL METODO .backward(), cada vez q lo ejecutamos, acumula el gradiente en los nodos hoja!, por eso, siempre se debe resetear los gradientes a 0, de los parametros. (“optimazer.zero_grad()”)

Blucle de entrenamiento

```python
# Ejemplo de un bucle de entrenamiento tradicional
for epoch in range(1, n_epochs + 1):
    # Se lleva a cero el gradiente acumulado
    if params.grad is not None:
        params.grad.zero_()

    # Forward pass
    prediccion = model(inputs, *params)
    # Función de costo
    loss = loss_fn(prediccion, ground_truth)
    # Backward pass
    loss.backward()

    # Con los gradientes, se actualizan los pesos
    with torch.no_grad():
        params -= learning_rate * params.grad

    return params
```

```python
# Ejemplo de un bucle de entrenamiento usando torch.optim
import torch.optim as optim

learning_rate = 1e-2
optimizer = optim.SGD([params], lr=learning_rate)

for epoch in range(1, n_epochs + 1):
    # Se lleva a cero el gradiente acumulado
    optimizer.zero_grad()

    # Forward pass
    prediccion = model(inputs, *params)
    # Función de costo
    loss = loss_fn(prediccion, ground_truth)
    # Backward pass
    loss.backward()

    # Con los gradientes, se actualizan los pesos
    optimizer.step()
    return params
```

Optimizador

Buscar el minimo a partir de gradient descent

Damos el paso

```
optimizer = torch.optim.SGD(lista_parametros, lr)
```

```
EJ: optimizer = torch.optim.SGD([x], lr=0.001)
```

Optimizando una función cuadrática

```
x = torch.tensor(.0, requires_grad=True)
optimizer = torch.optim.SGD([x], lr=0.001)
```

optimizer.zero_grad() ---> Setea en 0 los gradientes de los tensores

f = x ** 2 ---> Pasada forward que construye el grafo dinámico

f.backward() ---> Pasada backward que computa los gradientes

optimizer.step() ---> Avanza un "paso" y actualiza los parámetros con el correspondiente gradiente.

Modelos

```python
import torch.nn as nn
import torch.nn.functional as F

# Network architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(5, 7, bias=False)
        self.fc2 = nn.Linear(7, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
net = Net()
print(net)
```

En __init__ -> Defino las capas que componen la red

En forward -> Conectamos las capas, siempre recibe un x

Para procesar un dato(pasada forward)

```
Net(
  (fc1): Linear(in_features=5, out_features=7, bias=False)
  (fc2): Linear(in_features=7, out_features=1, bias=True)
)
```

```python
import torch.nn as nn
import torch.nn.functional as F

# Network architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.seq_model = nn.Sequential(
            nn.Linear(5, 7),
            nn.ReLU(),
            nn.Linear(7, 1)
        )
    def forward(self, x):
        x = self.seq_model(x)
        return x
net = Net()
print(net)
```

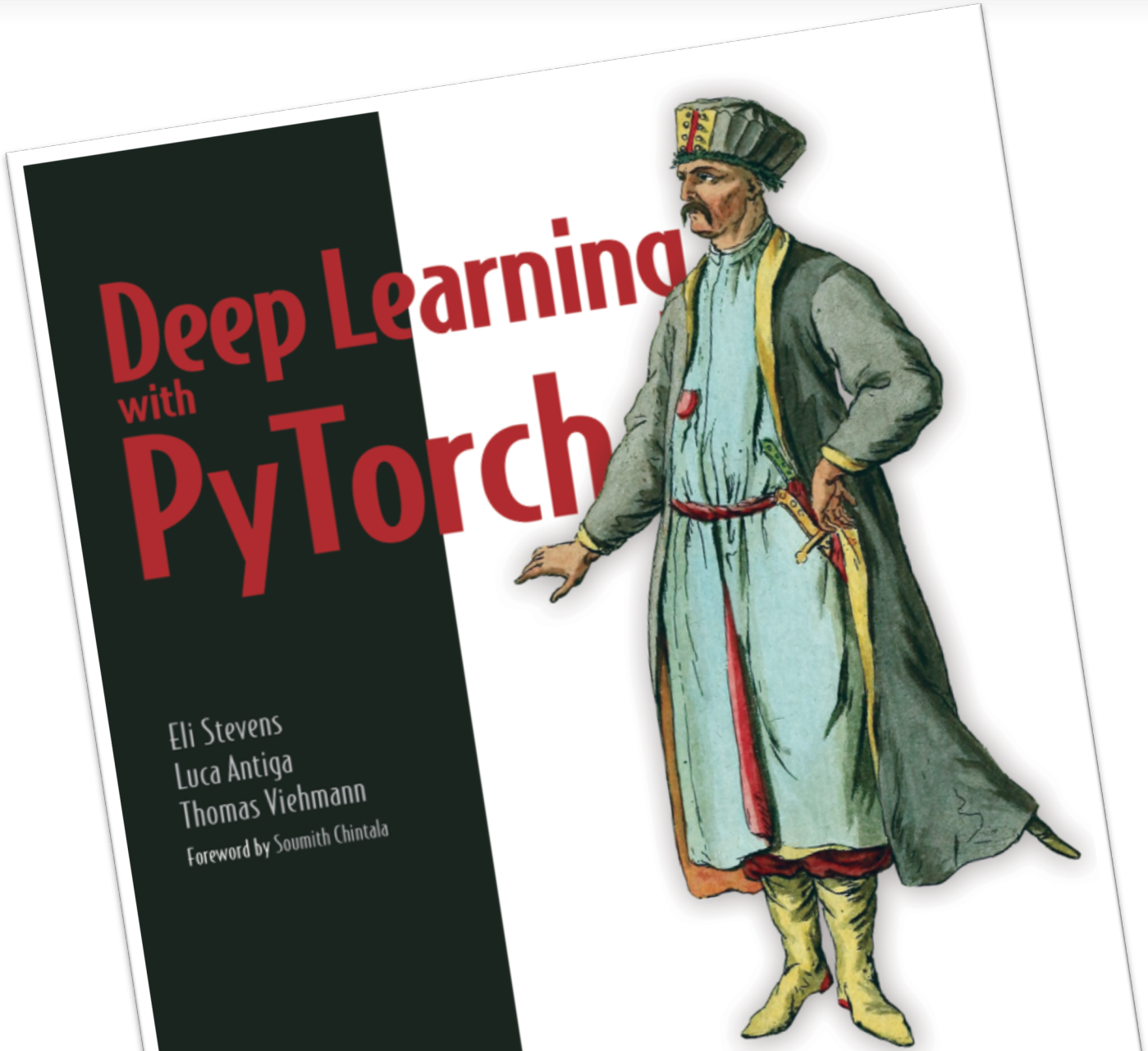
```
Net(
  (seq_model): Sequential(
    (0): Linear(in_features=5, out_features=7, bias=True)
    (1): ReLU()
    (2): Linear(in_features=7, out_features=1, bias=True)
  )
)
```

Modelos

```python
# Ejemplo de un bucle de entrenamiento usando torch.optim
import torch.nn as nn
import torch.optim as optim

learning_rate = 1e-2
optimizer = optim.SGD(net.parameters(), lr=learning_rate)
loss_fn = nn.MSELoss()
for epoch in range(1, n_epochs + 1):
    # Se lleva a cero el gradiente acumulado
    optimizer.zero_grad()

    # Forward pass
    prediccion = net(inputs)
    # Función de costo
    loss = loss_fn(prediccion, ground_truth)
    # Backward pass
    loss.backward()

    # Con los gradientes, se actualizan los pesos
    optimizer.step()
```

Me llama a todos los parametros que tiene la red “net”, para optimizarlos

Modelos


Deep Learning
with
PyTorch
Eli Stevens
Luca Antiga
Thomas Viehmann
Foreword by Soumith Chintala

Clase 3

Introducción a PyTorch

Enzo Ferrante

eferrante@sinc.unl.edu.ar

@enzoferrante

PYTORCH

Universidad de SanAndrés

UNL CONICET sinc(i)