Funciones de activación

Objetivo

- Aplicar diferentes funciones de activación en una red neuronal

Tabla de contenido

In [2]:
```
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

In [3]:
```
# Import the libraries we need for this lab

import torch.nn as nn
import torch

import matplotlib.pyplot as plt
torch.manual_seed(2)
```

Out[3]:
```
<torch._C.Generator at 0x1f6740d5330>
```

Función logística

Creamos un tensor:

In [4]:
```
# Create a tensor

z = torch.arange(-10, 10, 0.1,).view(-1, 1)
```

Cuando usa sequential, puede crear un objeto sigmoide:

In [5]:
```
# Create a sigmoid object

sig = nn.Sigmoid()
```

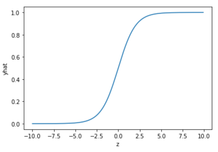
Realizamos una predicción:

In [6]:
```
# Make a prediction of sigmoid function

yhat = sig(z)
```

Graficamos:

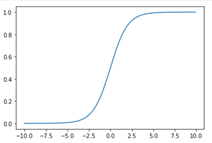
In [7]:
```
# Plot the result

plt.plot(z.detach().numpy(),yhat.detach().numpy())
plt.xlabel('z')
plt.ylabel('yhat')
```

Out[7]:
```
Text(0, 0.5, 'yhat')
```



Para módulos personalizados:

In [8]:
```
# Use the build in function to predict the result

yhat = torch.sigmoid(z)
plt.plot(z.numpy(), yhat.numpy())

plt.show()
```



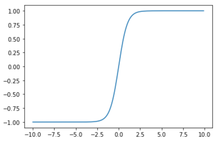
Tanh

Usando sequential:

In [9]:
```
# Create a tanh object

TANH = nn.Tanh()
```

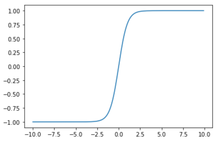
Llamamos al objeto y graficamos:

In [10]:
```
# Make the prediction using tanh object

yhat = TANH(z)
plt.plot(z.numpy(), yhat.numpy())
plt.show()
```



Para módulos personalizados:

In [11]:
```
# Make the prediction using the build-in tanh object

yhat = torch.tanh(z)
plt.plot(z.numpy(), yhat.numpy())
plt.show()
```

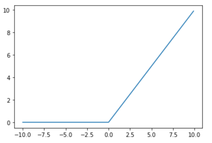


Relu

Con sequential:

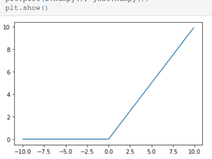
In [12]:
```
# Create a relu object and make the prediction

RELU = nn.ReLU()
yhat = RELU(z)
plt.plot(z.numpy(), yhat.numpy())
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x1f676fc5550>]
```



Para módulos personalizados:

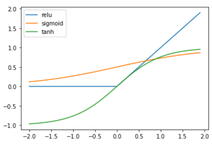
In [13]:
```
# Use the build-in function to make the prediction

yhat = torch.relu(z)
plt.plot(z.numpy(), yhat.numpy())
plt.show()
```



Comparamos las funciones de activación

In [14]:
```
# Plot the results to compare the activation functions

x = torch.arange(-2, 2, 0.1).view(-1, 1)
plt.plot(x.numpy(), torch.relu(x).numpy(), label='relu')
plt.plot(x.numpy(), torch.sigmoid(x).numpy(), label='sigmoid')
plt.plot(x.numpy(), torch.tanh(x).numpy(), label='tanh')
plt.legend()
```

Out[14]:
```
<matplotlib.legend.Legend at 0x1f67703df0>
```



Práctica

Compare las funciones de activación pero ahora con un tensor en el rango *(-1, 1)*

In [16]:
```
# Practice: Compare the activation functions again using a tensor in the range (-1, 1

#
x = torch.arange(-1, 1, 0.1).view(-1, 1)
plt.plot(x.numpy(), torch.relu(x).numpy(), label = 'relu')
plt.plot(x.numpy(), torch.sigmoid(x).numpy(), label = 'sigmoid')
plt.plot(x.numpy(), torch.tanh(x).numpy(), label = 'tanh')
plt.legend()
```

Out[16]:
```
<matplotlib.legend.Legend at 0x1f6770lefa0>
```