CC3092 - Deep Learning y Sistemas Inteligentes

Hoja de Trabajo 2

Semestre II - 2025

Integrantes:

- Sofía Mishell Velásquez, 22049
- José Rodrigo Marchena, 22398
- José Joaquín Campos, 22115

Ejercicio 1 - Experimentación Práctica

Task 1 - Preparación del conjunto de datos

En esta sección se carga el conjunto de datos Iris utilizando `sklearn.datasets`. Posteriormente, se dividen los datos en entrenamiento y validación para asegurar que los modelos se prueben adecuadamente en datos no vistos.

In [3]:
```
# Task 1 - Preparación del conjunto de datos
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import torch
from torch.utils.data import TensorDataset, DataLoader

# Cargar dataset Iris
iris = load_iris()
X, y = iris.data, iris.target

# Normalización
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Separación en train/val
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state

# Conversión a tensores
X_train_t = torch.tensor(X_train, dtype=torch.float32)
y_train_t = torch.tensor(y_train, dtype=torch.long)
X_val_t   = torch.tensor(X_val, dtype=torch.float32)
y_val_t   = torch.tensor(y_val, dtype=torch.long)

# Crear DataLoaders
```

```
train_loader = DataLoader(TensorDataset(X_train_t, y_train_t), batch_size=16, shuff
val_loader   = DataLoader(TensorDataset(X_val_t, y_val_t), batch_size=16, shuffle=F
```

```
C:\Users\50250\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra
8p0\LocalCache\local-packages\Python38\site-packages\tqdm\auto.py:21: TqdmWarning: I
Progress not found. Please update jupyter and ipywidgets. See https://ipywidgets.rea
dthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Task 2 - Arquitectura modelo

Aquí se construye una red neuronal feedforward simple con nn.Module. La red incluye una capa de entrada, capas ocultas y una capa de salida con número de neuronas igual al número de clases en Iris (3).

In [4]:
```
# Task 2 - Arquitectura del modelo
import torch.nn as nn

class IrisNet(nn.Module):
    def __init__(self, input_dim=4, hidden_dim=16, output_dim=3):
        super(IrisNet, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        return self.fc3(x)

# Instanciar modelo
model = IrisNet()
print(model)
```

```
IrisNet(
  (fc1): Linear(in_features=4, out_features=16, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=16, out_features=16, bias=True)
  (fc3): Linear(in_features=16, out_features=3, bias=True)
)
```

Task 3 - Funciones de Pérdida

El objetivo es comparar el rendimiento del modelo utilizando distintas funciones de pérdida. Se probarán al menos tres pérdidas comunes:

- nn.CrossEntropyLoss (adecuada para clasificación multiclase)
- nn.MSELoss (Mean Squared Error)
- nn.NLLLoss (Negative Log Likelihood, usando LogSoftmax)

In [5]:
```python
# Task 3 - Funciones de Pérdida
loss_functions = {
    "CrossEntropy": nn.CrossEntropyLoss(),
    "MSE": nn.MSELoss(),
    "NLL": nn.NLLLoss()
}

# Ejemplo de selección de función de pérdida
criterion = loss_functions["CrossEntropy"]
```