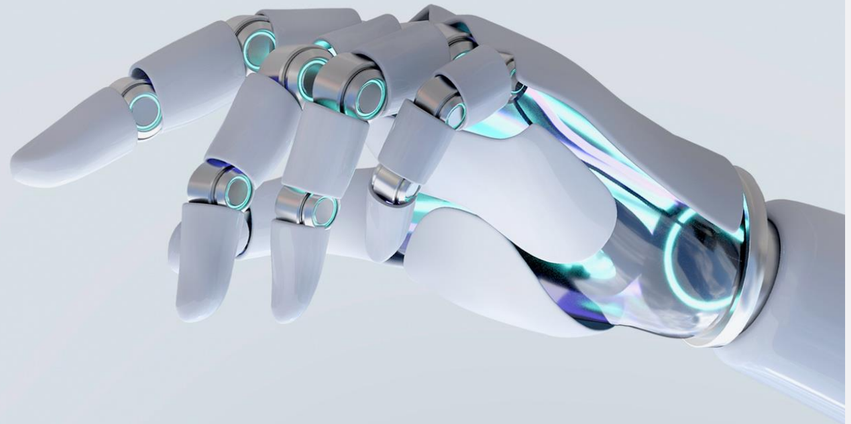
UNIVERSITAT D'ALACANT
UNIVERSIDAD DE ALICANTE
Escola Politècnica Superior
Escuela Politécnica Superior

MES CULTURAL EPS 2026



INTRODUCCIÓN AL

MACHINE LEARNING

Isabela Fons
isabela.fons@ua.es
Grupo COnCEPT-Departamento de Ingeniería Química

GoogleColab

Haz click en los siguientes links para acceder al contenido:

1. Entrenamiento de redes neuronales
2. Prevención del sobreajuste
3. Optimización de hiperparámetros

Local

```
git clone https://github.com/isfons/intro-ML.git
cd intro-ML
pip install -e .
```

* Necesaria instalación previa de Git, Python y solvers de optimización.

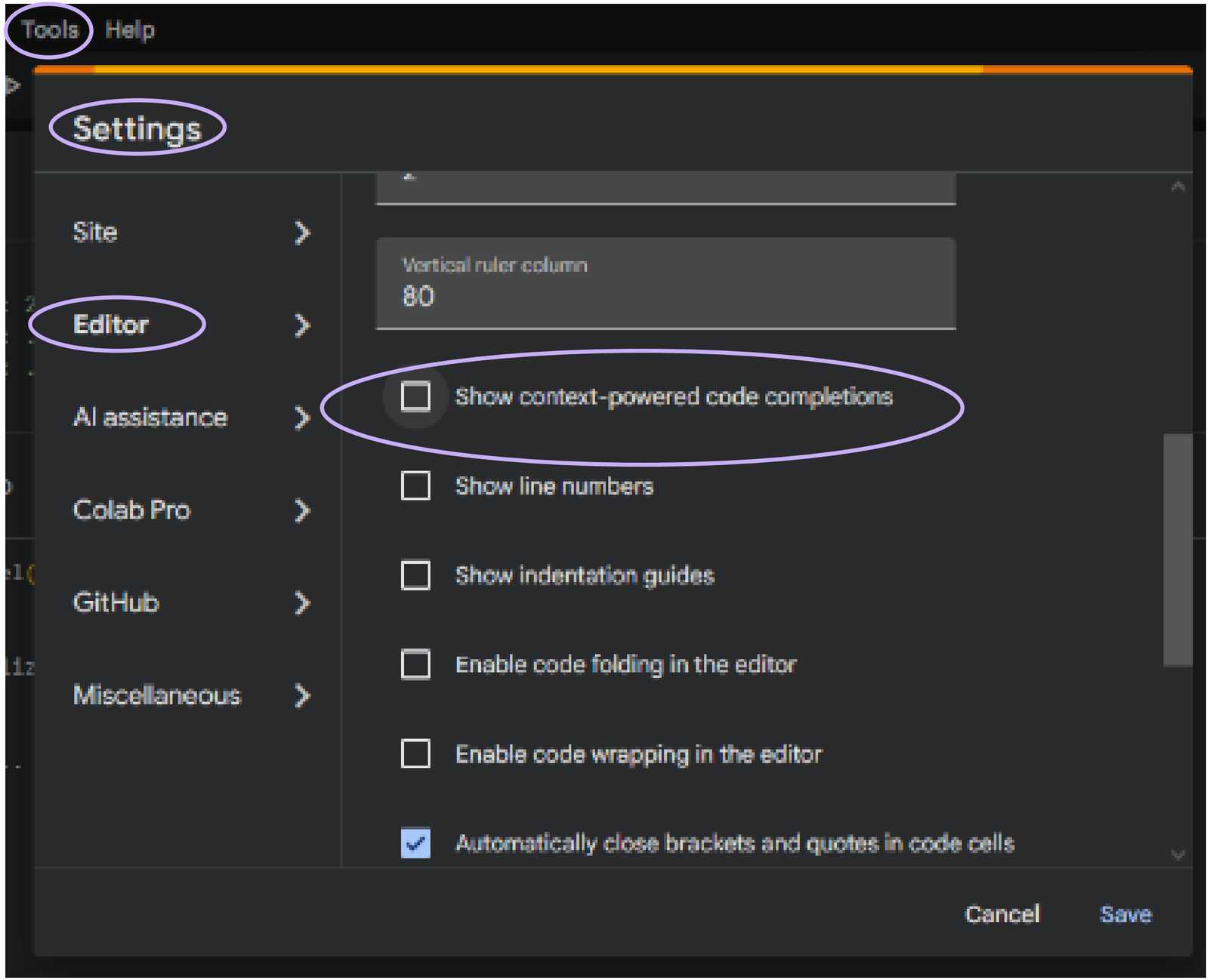
INSTRUCCIONES

1

2

3

4



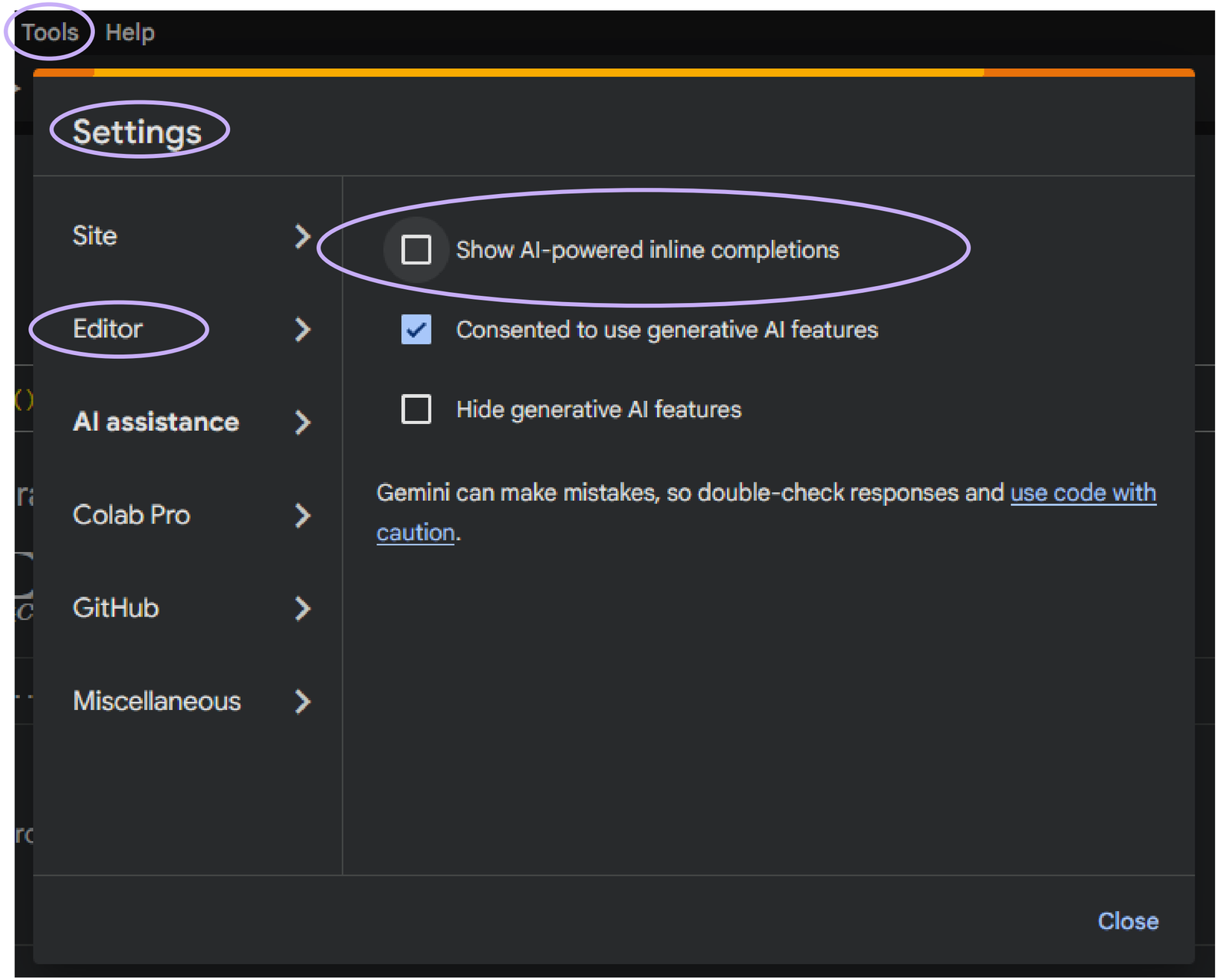
INSTRUCCIONES

1

2

3

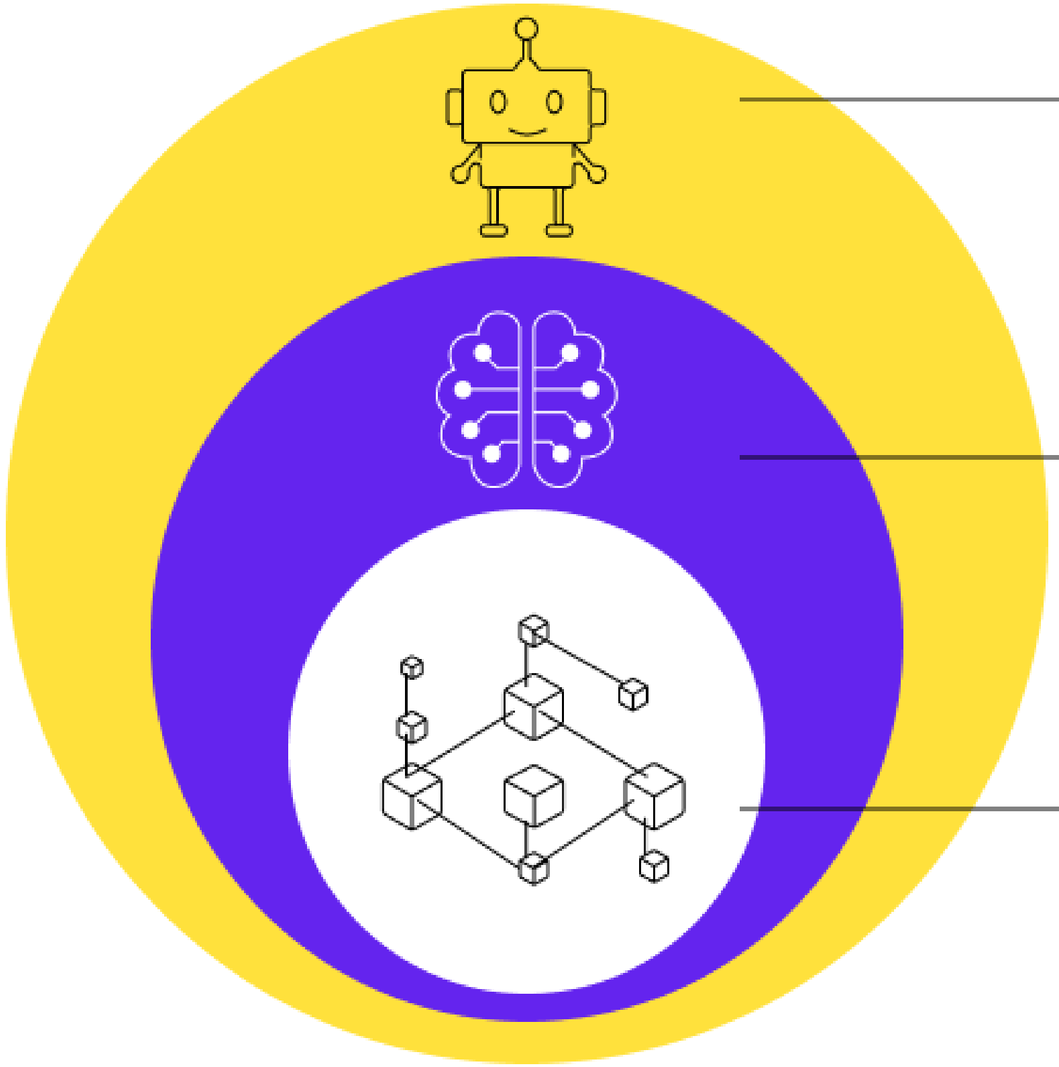
4



INTELIGENCIA ARTIFICIAL

“La ciencia y la ingeniería de hacer máquinas inteligentes, especialmente programas de computadora inteligentes”

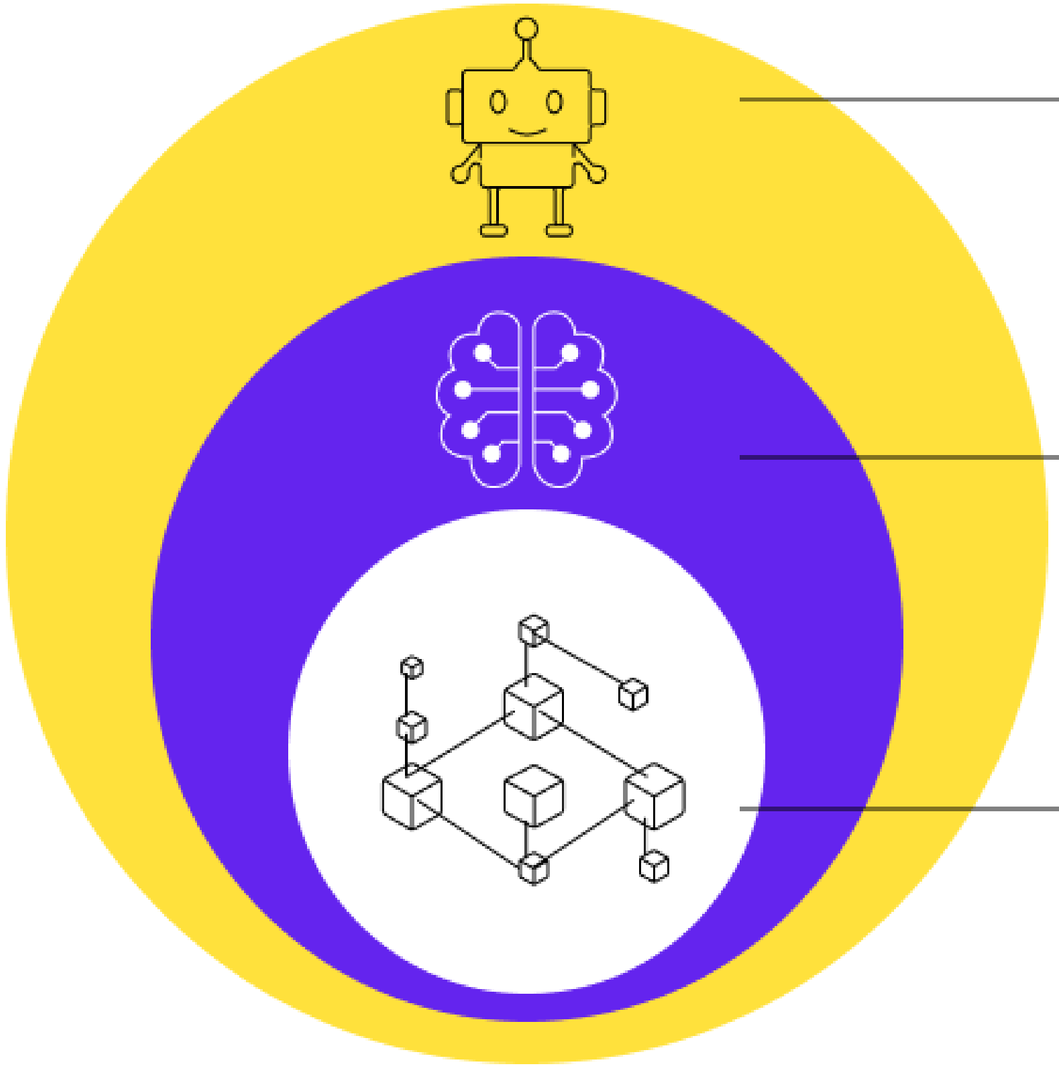
JOHN MCCARTHY, 1956



INTELIGENCIA ARTIFICIAL

MACHINE LEARNING

DEEP LEARNING



INTELIGENCIA ARTIFICIAL

Rama de la informática orientada a conseguir que las máquinas simulen comportamientos humanos y tomen decisiones de forma autónoma

MACHINE LEARNING

Subcampo de la IA basado en la estadística para construir modelos predictivos que aprenden de grandes cantidades de datos.

DEEP LEARNING

Técnica de ML capaz de aprender patrones más complejos utilizando redes neuronales.

TIPOS DE APRENDIZAJE

Aprendizaje supervisado

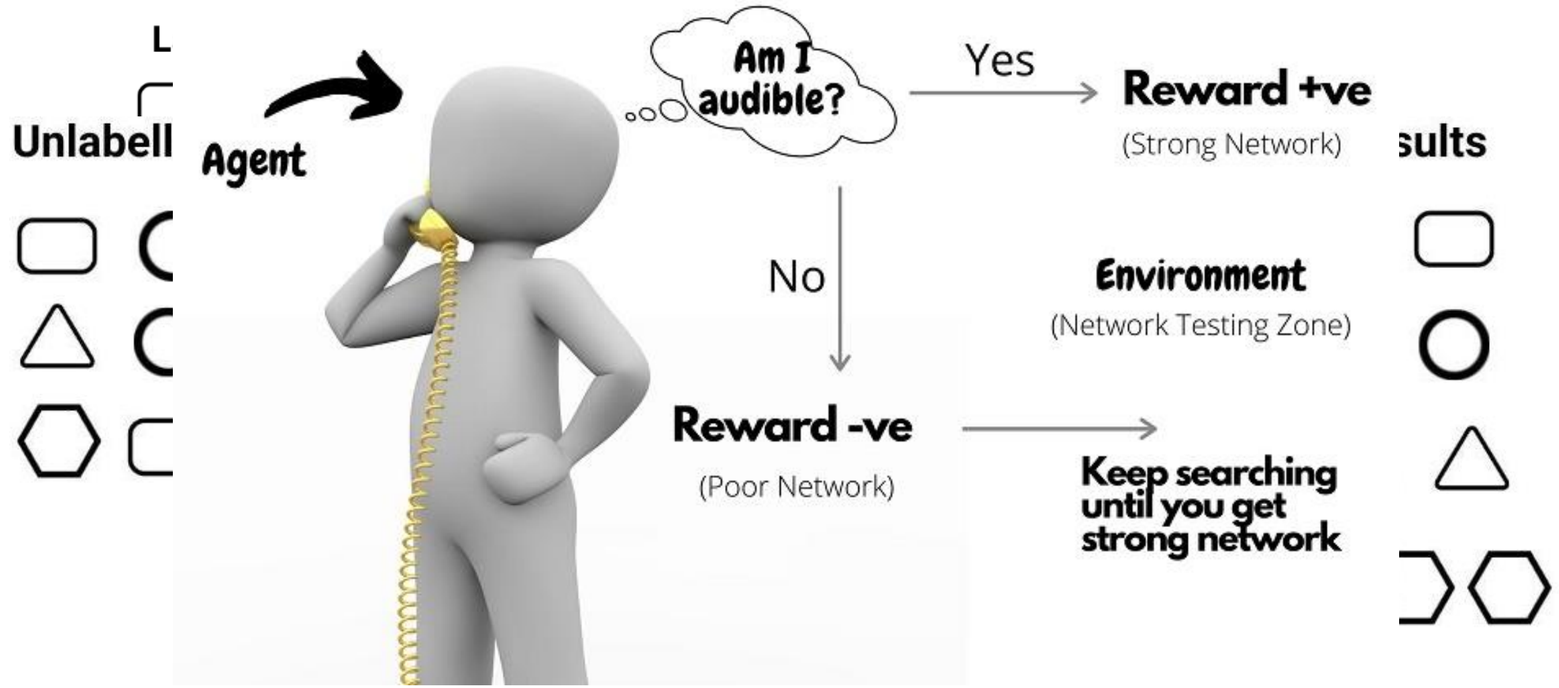
Relaciona variables entrada – salida

Aprendizaje no supervisado

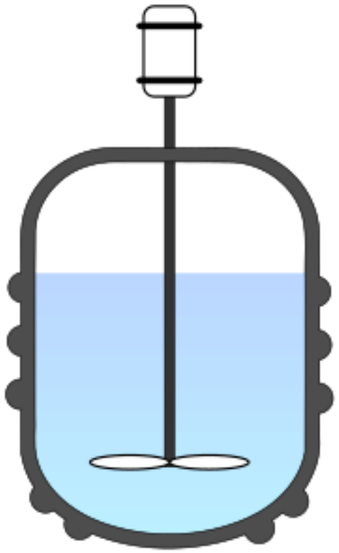
Encuentra patrones en los datos de entrada

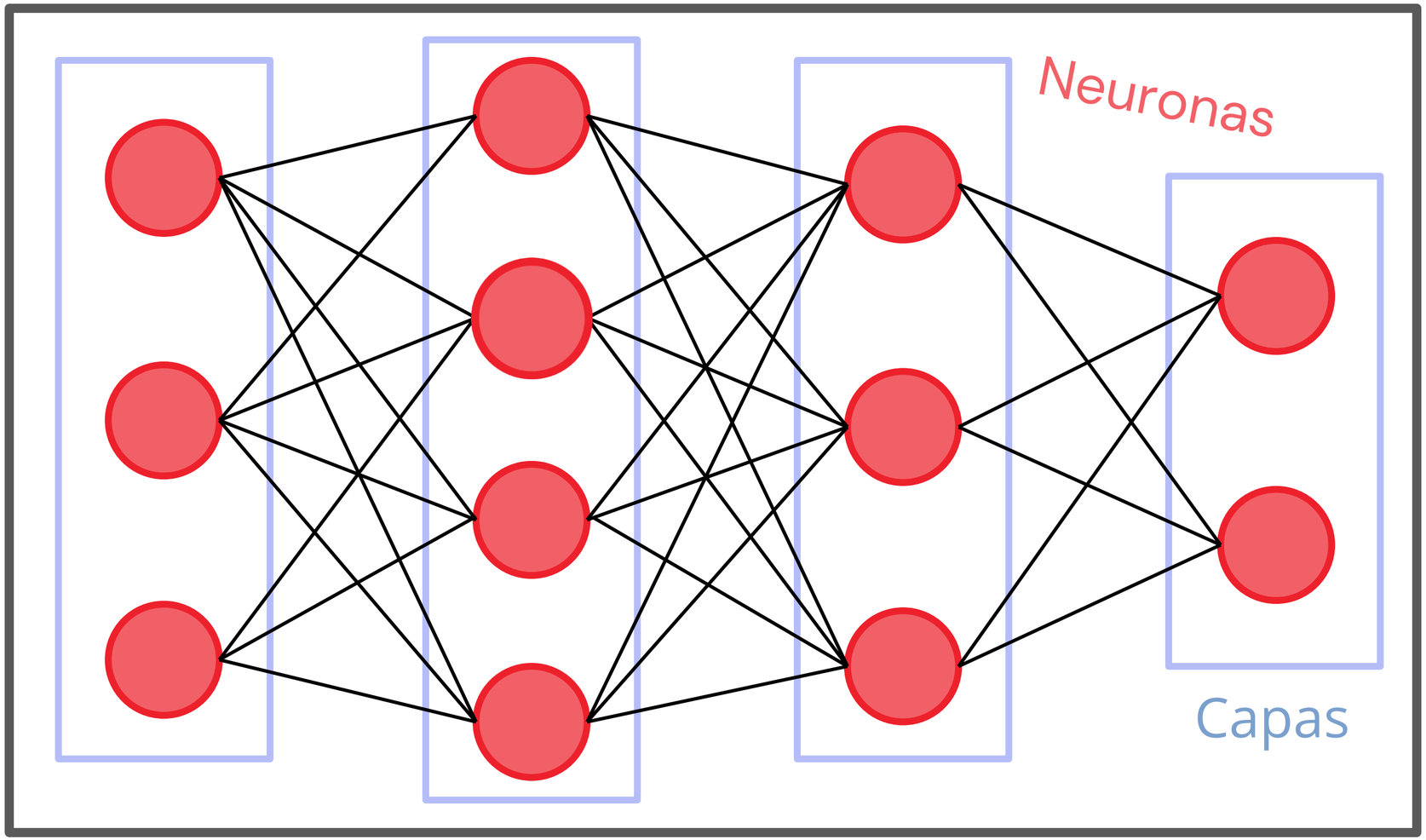
Aprendizaje por refuerzo

Método de prueba-error



REDES NEURONALES

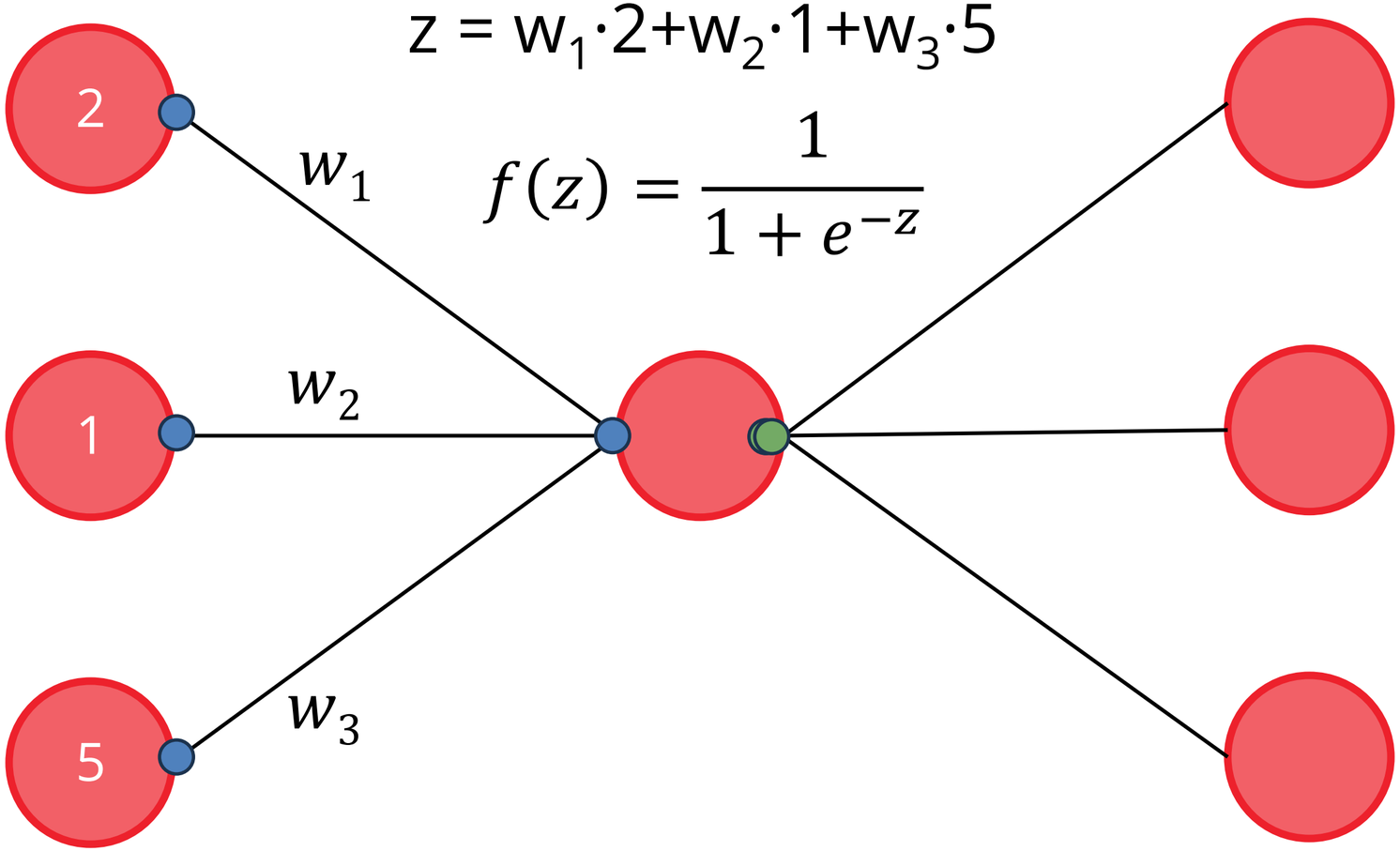



Neuronas
Capas

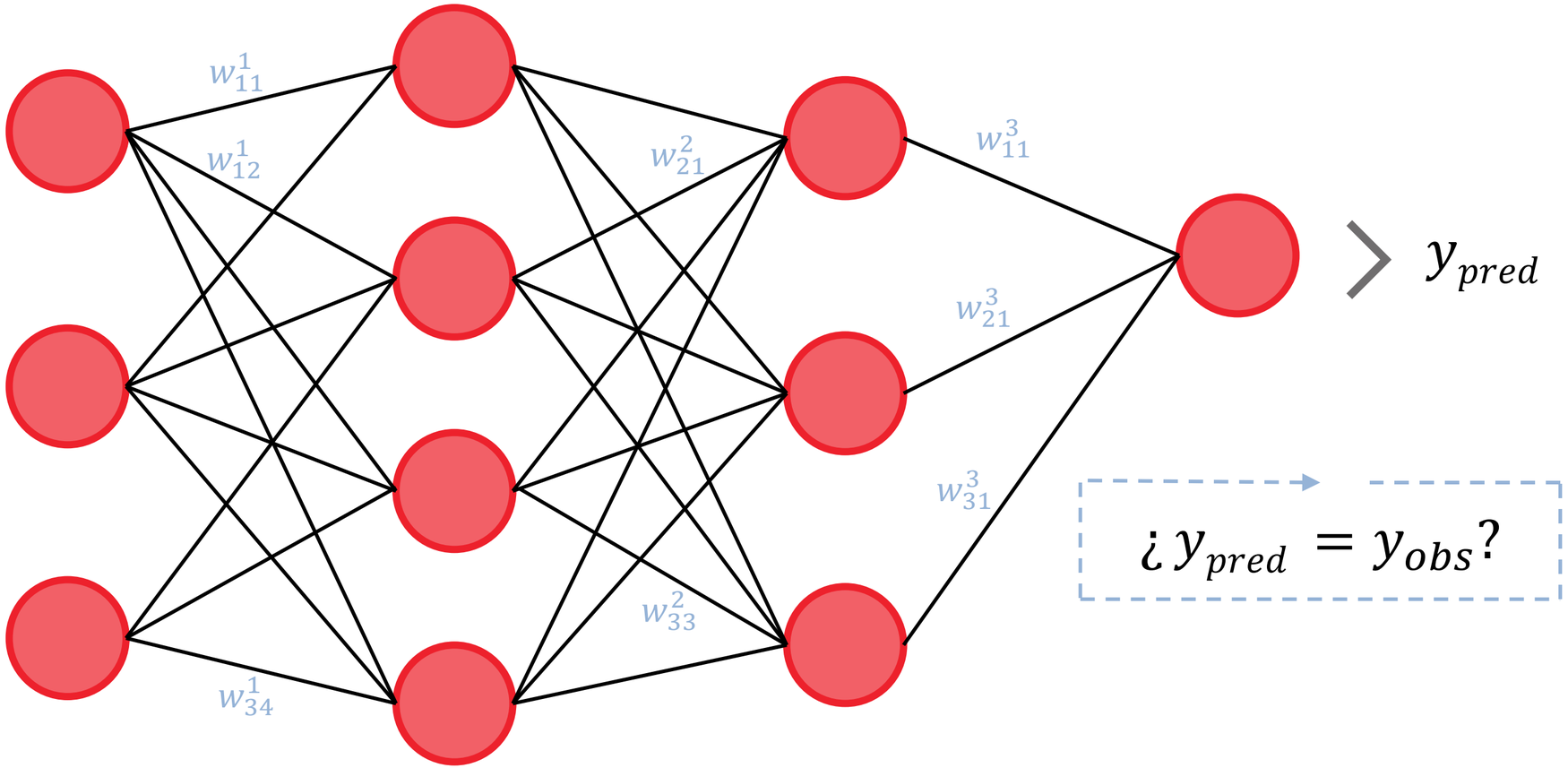
FORWARD PROPAGATION

$$z = w_1 \cdot 2 + w_2 \cdot 1 + w_3 \cdot 5$$

$$f(z) = \frac{1}{1 + e^{-z}}$$



BACKPROPAGATION



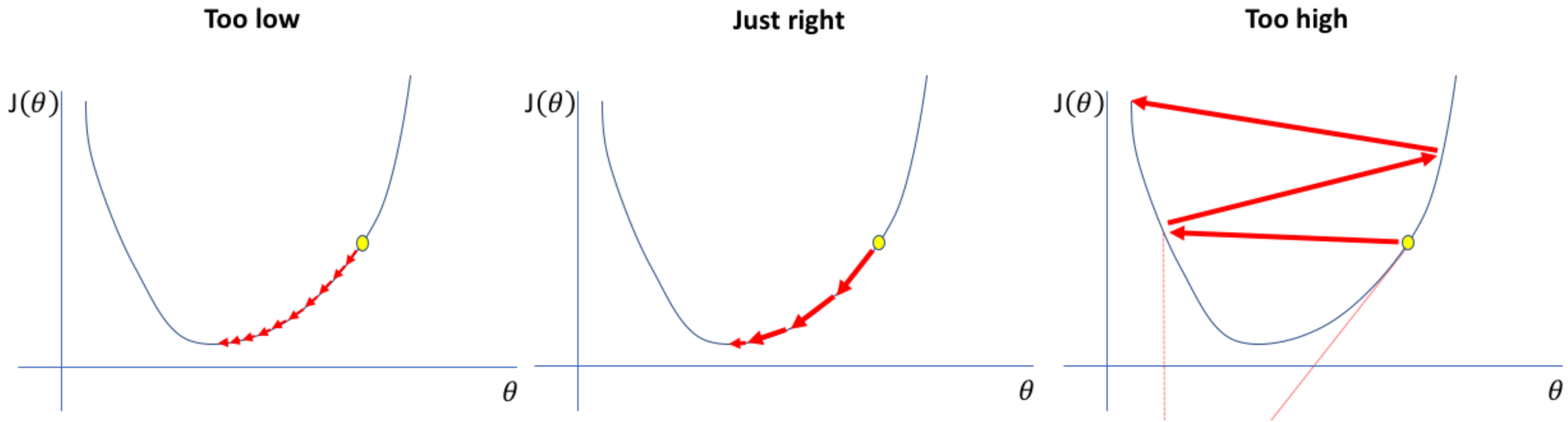
BACKPROPAGATION

$$\min \mathcal{L} = \sum_{i=1}^{N_{points}} \left(y_{obs,i} - y_{pred,i}\right)^2$$

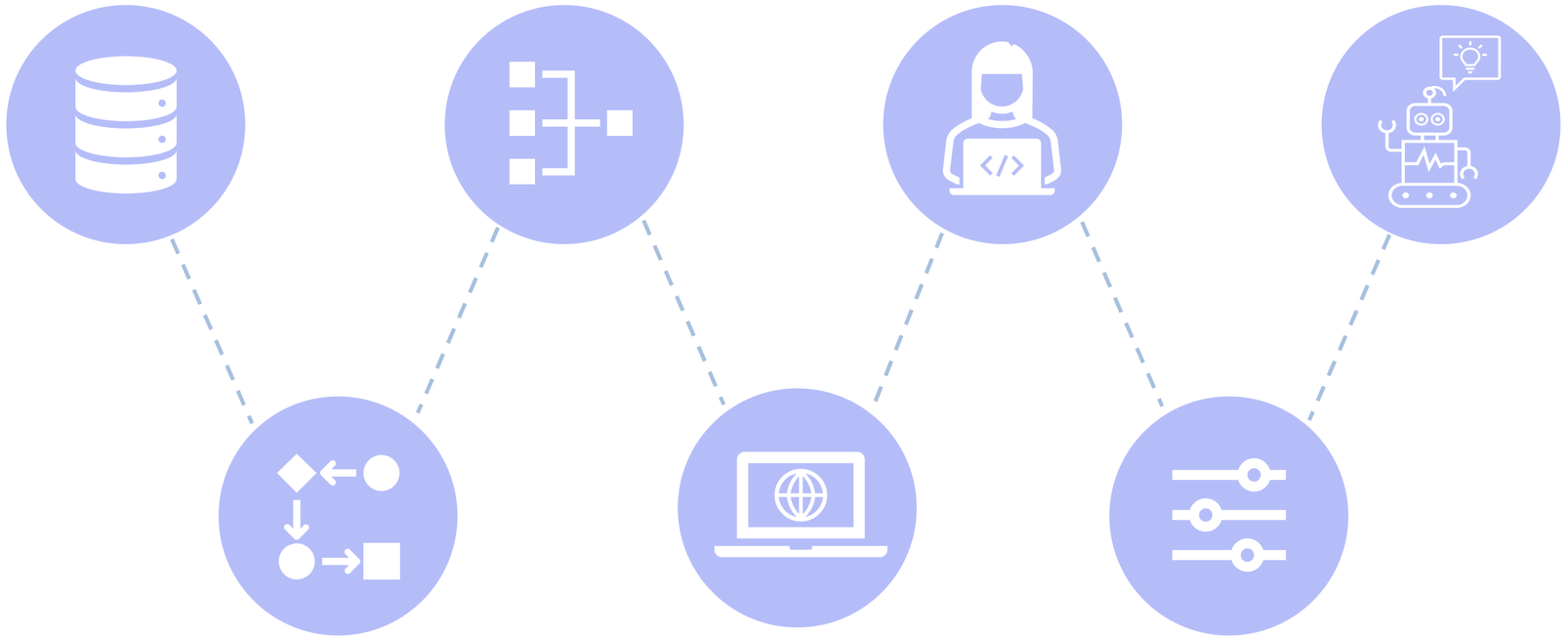
MÉTODO DESCENSO DEL GRADIENTE

$$w_{new} = w_{old} - \boldsymbol{\alpha} \frac{\delta \mathcal{L}}{\delta w}$$

α: Tamaño del paso o *learning rate*

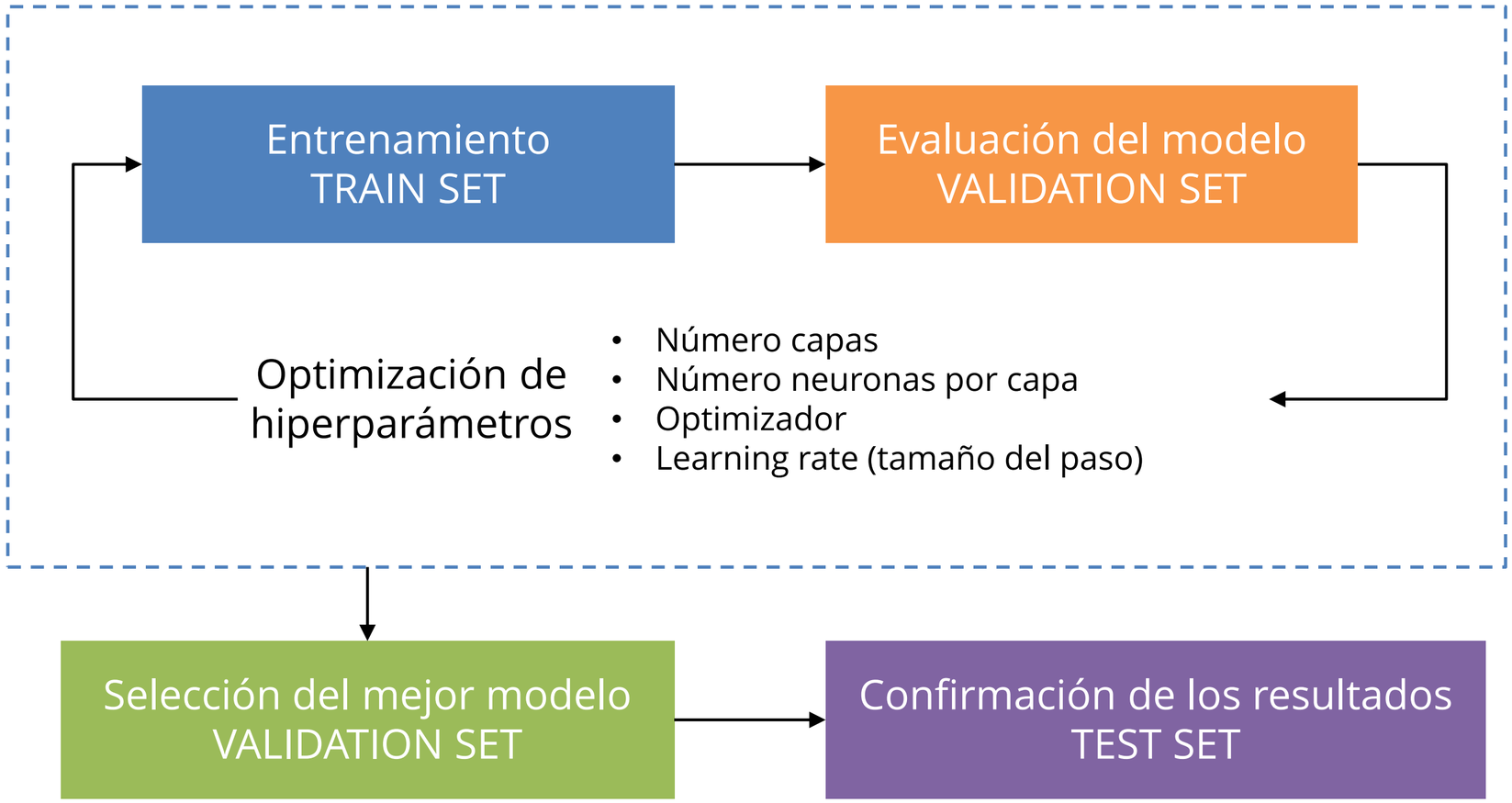


CONSTRUIR UN MODELO EN 7 PASOS



TRATAMIENTO DE DATOS

SEPARACIÓN TRAIN-TEST-VALIDATION



TRATAMIENTO DE DATOS

ESCALADO

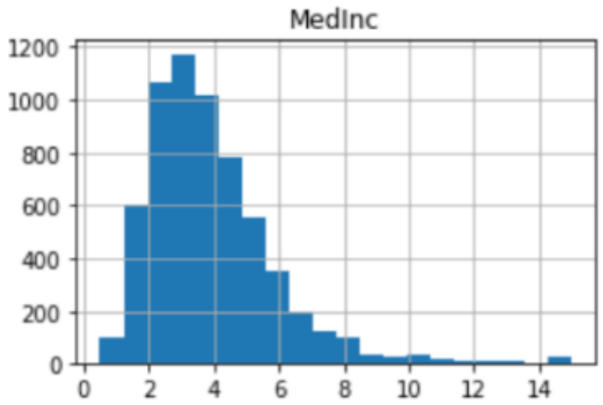
- Cambio en la magnitud de las variables sin afectar a la forma de su distribución.
- Facilita convergencia algoritmo optimización

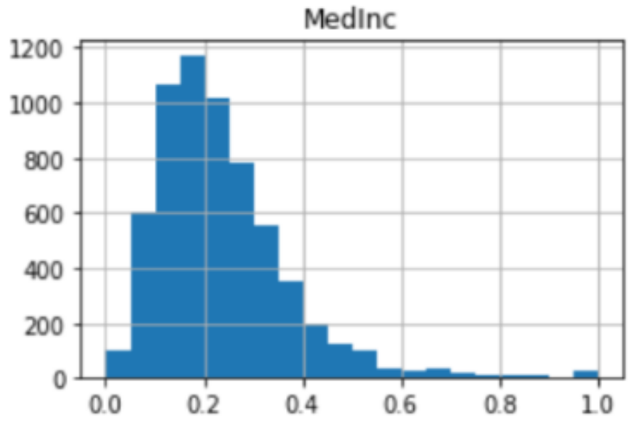
MIN-MAX

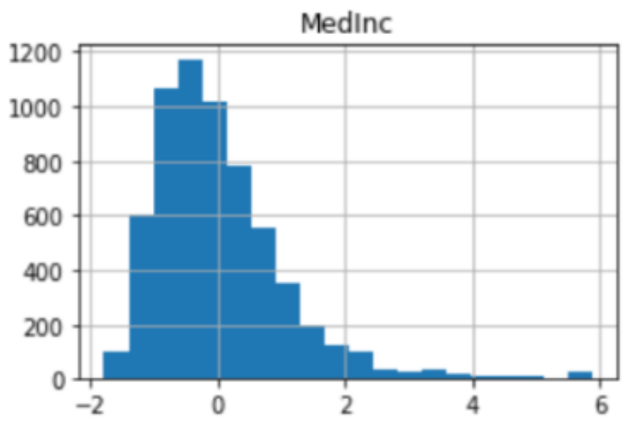
$$X_{scaled} = \frac{X - X_{min}}{X - X_{max}}$$

Z-SCORES

$$X_{scaled} = \frac{X - \mu_X}{\sigma_X}$$







MODELADO Y ENTRENAMIENTO

	scikit learn	TensorFlow	TensorFlow + Keras	PyTorch
ABSTRACCIÓN	Alto	Bajo–medio	Muy alto	Medio
TIPO DE REDES NEURONALES	Redes simples (MLP)	Redes profundas complejas	Redes profundas	Redes profundas
FACILIDAD	Muy alta	Media	Muy alta	Alta
FLEXIBILIDAD	Baja	Muy alta	Media	Muy alta
APLICACIÓN	• Prototipos rápidos • Datasets pequeños	• Producción e implementación	• Aprendizaje rápido • Educación	• Investigación • Modelos personalizados

DEFINICIÓN DEL MODELO

DEFINICIÓN DE LA ARQUITECTURA

- Método __init__(self)
- torch.nn package:
  - Tipos de capas (nn.Linear)
  - Funciones de activación (nn.ReLU)

ESPECIFICAR “FORWARD PASS”

- Método forward(self, x)
- Operaciones con Tensors
  - Unir o separar Tensors
  - Multiplicación

```
class Net(nn.Module):
    def __init__(self):
      super(Net, self).__init__()
      self.fc1 = nn.Linear(9216, 128)
      self.fc2 = nn.Linear(128, 1)
      self.act = nn.ReLU

def forward(self, x):# x represents data
      # Pass data through hidden layers
      x = self.fc1(x)
      x = self.act(x)
      x = self.fc2(x)
      return x
```

ENTRENAMIENTO

FUNCIÓN DE PÉRDIDAS $\mathcal{L}(\boldsymbol{\theta})$

torch.nn package:

- MSELoss (regresión)
- CrossEntropyLoss (clasificación)

ALGORITMO DE OPTIMIZACIÓN

torch.optim package:

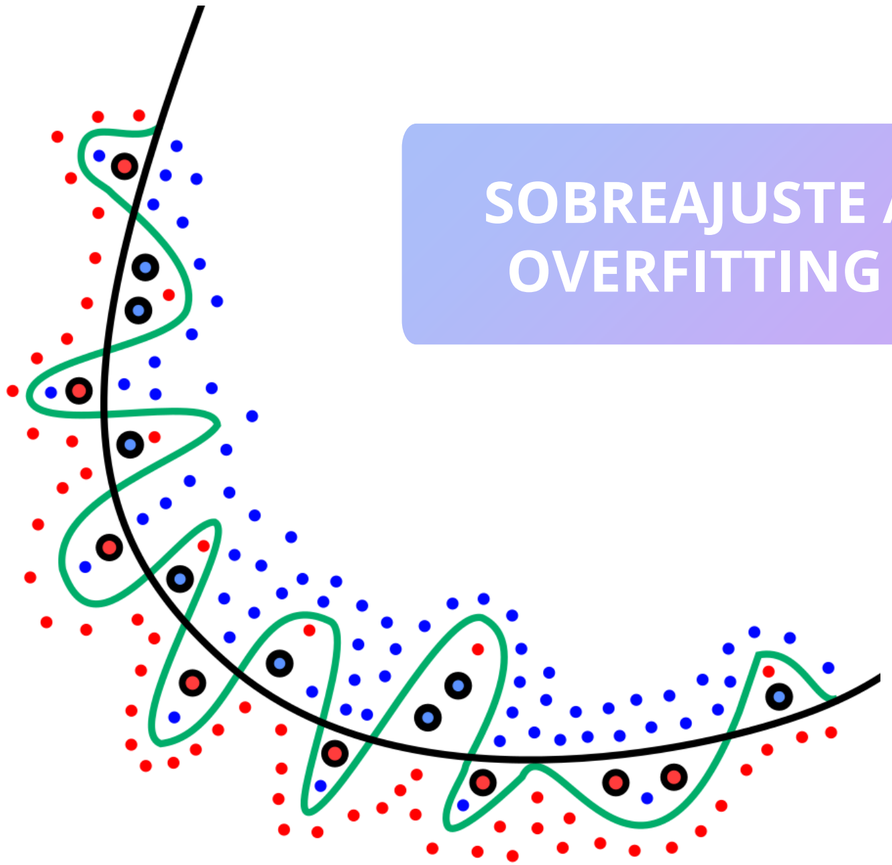
- Adam, SGD, RMSProp ...

BUCLE DE ENTRENAMIENTO

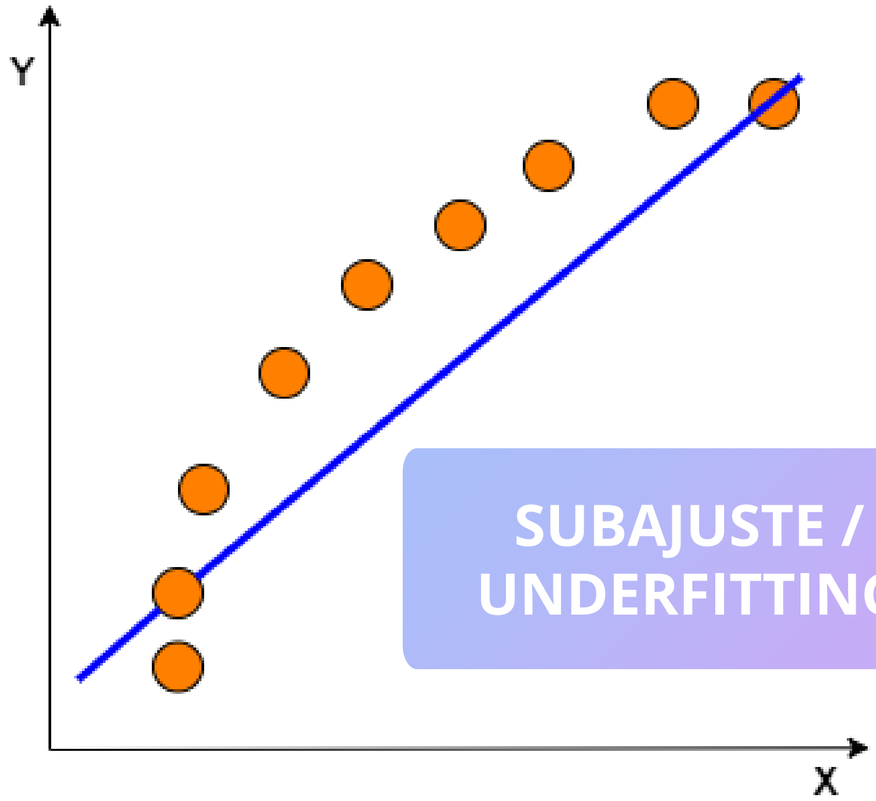
- zero_grad(): “limpia” los gradientes del paso anterior
- backward(): calcula la $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ usando *backpropagation*
- optimizer.step(): actualiza el valor de $\boldsymbol{\theta}$

```
Loss_fcn = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001)

# loop over the dataset multiple times
for epoch in range(EPOCHS):
        # loop over batches
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = loss_fcn(outputs, labels)
        loss.backward()
        optimizer.step()
print('Finished Training')
```

EVALUACIÓN DEL MODELO

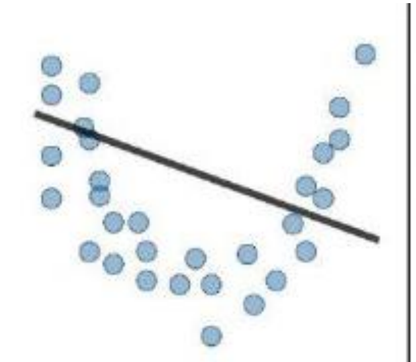
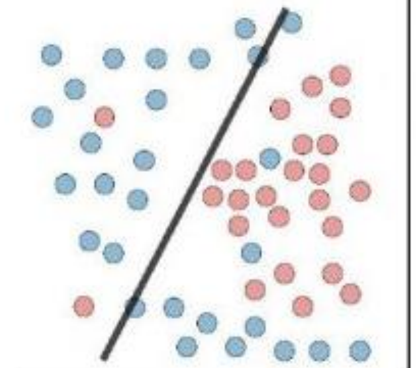
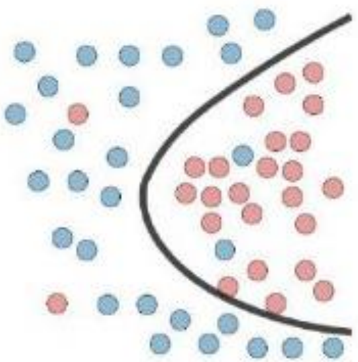
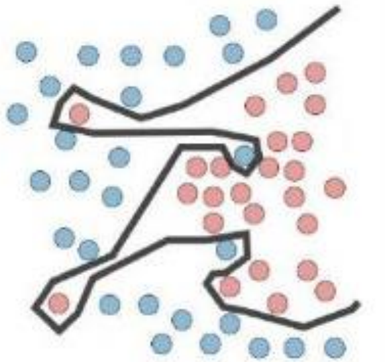
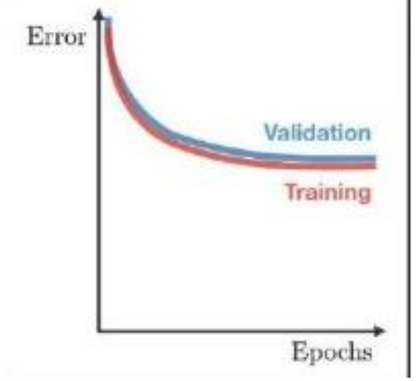
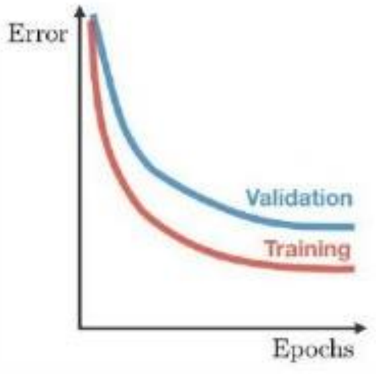
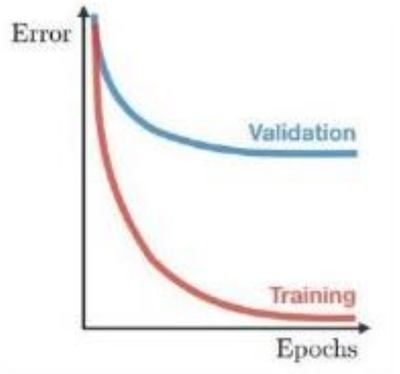


SOBREAJUSTE / OVERFITTING



SUBAJUSTE / UNDERFITTING

Linear Regression

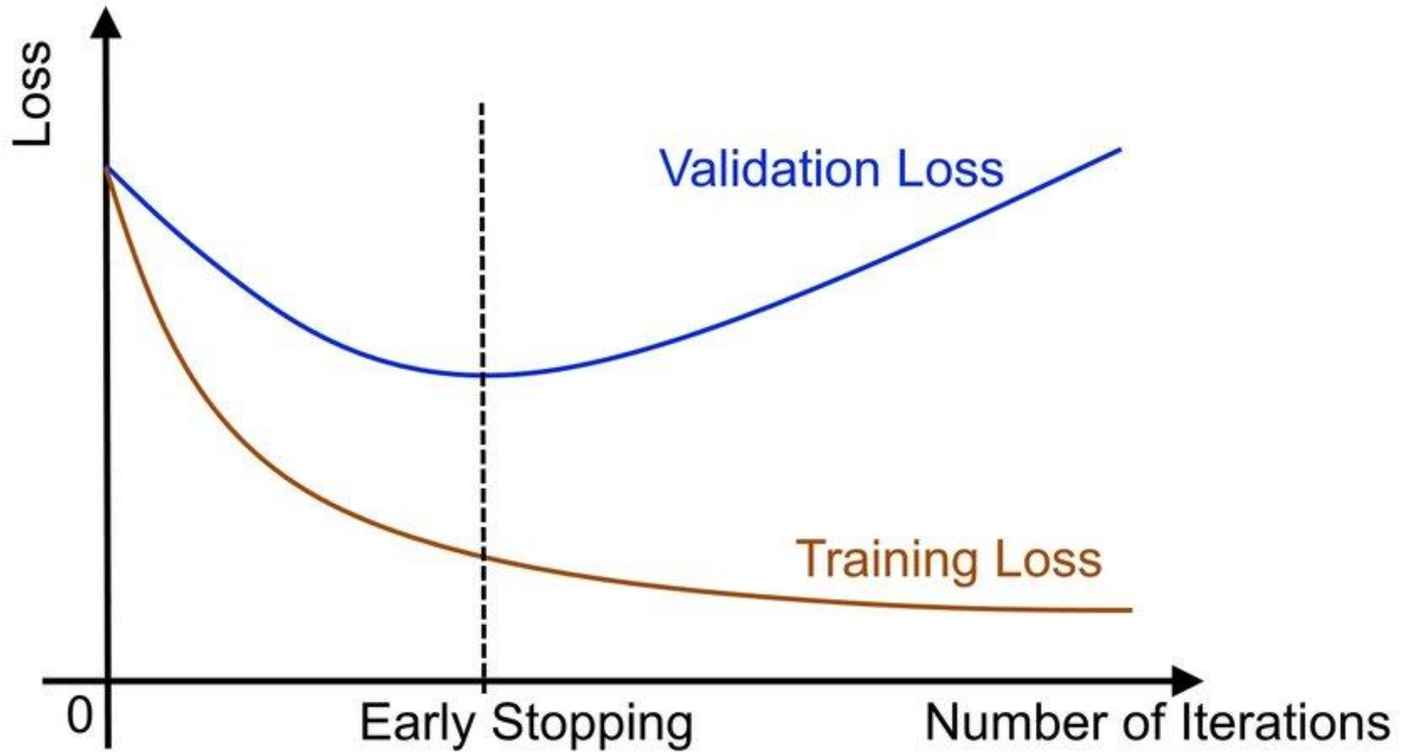
	SUBAJUSTE	CORRECTO	SOBREAJUSTE
REGRESIÓN			
CLASIFICACIÓN			
DEEP LEARNING			

SOBREAJUSTE

¿CÓMO EVITARLO?

EARLY STOPPING

- Interrupción del entrenamiento cuando el error en validación empeora.



SOBREAJUSTE

¿CÓMO EVITARLO?

EARLY STOPPING

REGULARIZACIÓN

- Penalización añadida a la *loss function* proporcional al valor de los pesos

$$\mathcal{L} = \sum_{i=1}^{N_{points}} \left(y_{obs,i} - y_{pred,i}\right)^2$$

$$+\lambda \sum_{j=1}^{N_{param}} |w_j|$$

L1
Lasso

$$+\lambda \sum_{j=1}^{N_{param}} w_j^2$$

L2
Ridge

SOBREAJUSTE

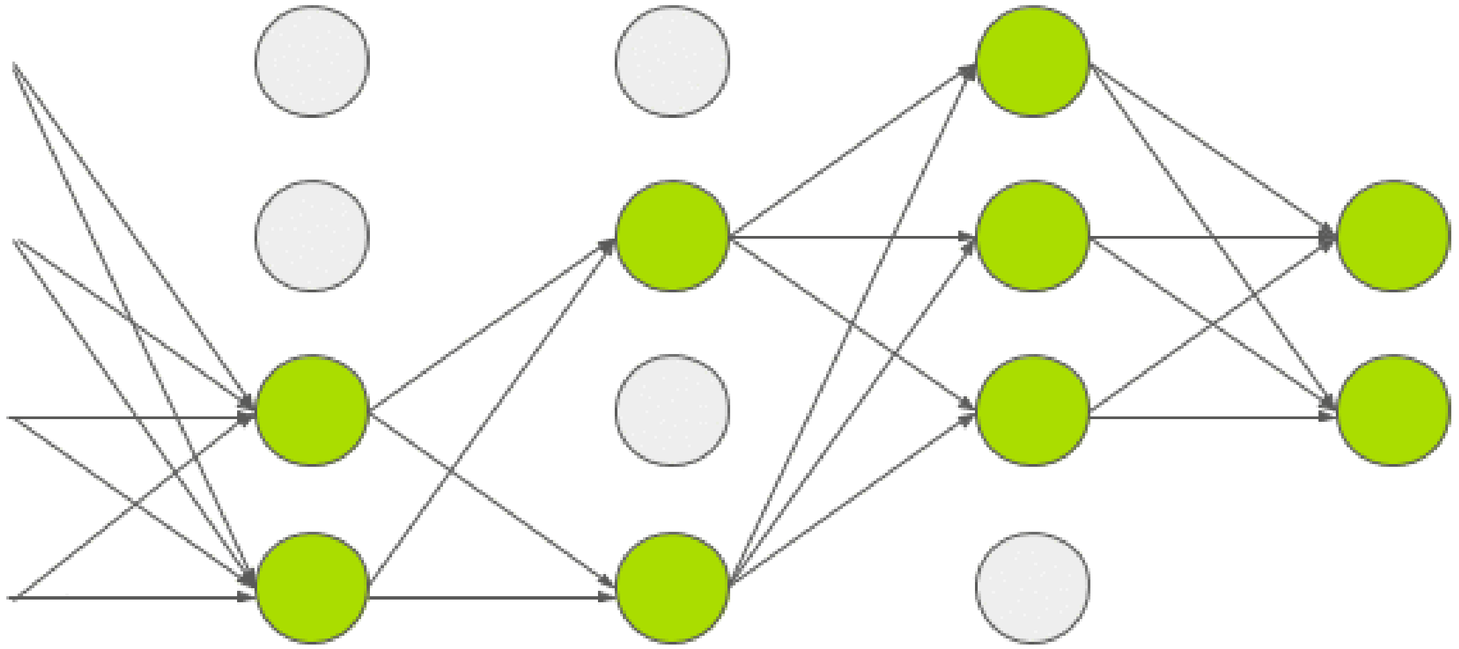
¿CÓMO EVITARLO?

EARLY STOPPING

REGULARIZACIÓN

DROPOUT

- Desactivación aleatoria de neuronas durante el entrenamiento



AJUSTE DE HIPERPARÁMETROS

HIPERPARÁMETROS

- No forman parte del modelo
- Se fijan antes de entrenar el modelo
- Independientes de los datos

PARÁMETROS

- Forman parte del modelo
- Se aprenden durante el entrenamiento a partir de los datos
- Dependen del *train set*

OPTUNA

AJUSTE DE HIPERPARÁMETROS

CONCEPTOS BÁSCICOS DE OPTUNA

STUDY

- Optimización para encontrar el mejor conjunto de hiperparámetros
- Objetivo: minimizar el error en el *validation set*

TRIAL

- Evaluación de la función objetivo para una combinación de hiperparámetros

AJUSTE DE HIPERPARÁMETROS

WORKFLOW

1. Compacta el proceso de entrenamiento en una función `objective()`
2. Sugiere hiperparámetros utilizando un `Trial`
3. Crea un `Study` y ejecuta la optimización

```python
# PASO 1
def objective(trial):
    # PASO 2
    n_layers = trial.suggest_int('n_layers', 1, 3)
    layers = []

    in_features = 28 * 28
    for i in range(n_layers):
        out_features = trial.suggest_int(f'n_units_l{i}', 4, 128)
        layers.append(torch.nn.Linear(in_features, out_features))
        layers.append(torch.nn.ReLU())
        in_features = out_features
    layers.append(torch.nn.Linear(in_features, 10))
    layers.append(torch.nn.LogSoftmax(dim=1))
    model = torch.nn.Sequential(*layers).to(torch.device('cpu'))
    ...
    return accuracy

# PASO 3
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

¡GRACIAS POR VUESTRA ATENCIÓN!

Isabela Fons
isabela.fons@ua.es
Grupo COnCEPT-Departamento de Ingeniería Química