06_doe_una_variable

June 15, 2024

1 Diseño de experimento

El diseño de experimentos en el entrenamiento de redes neuronales juega un papel crucial en la optimización del rendimiento del modelo. Este enfoque sistemático permite explorar y ajustar los hiperparámetros (como la tasa de aprendizaje, el número de capas y neuronas, y el tamaño del lote) de manera eficiente y efectiva. Al planificar y estructurar los experimentos, se pueden identificar las combinaciones óptimas de hiperparámetros que mejoran la precisión y la generalización del modelo, mientras se minimiza el tiempo y los recursos computacionales necesarios. Además, un buen diseño de experimentos reduce el impacto ecológico del entrenamiento de redes neuronales al disminuir el número de ejecuciones necesarias, optimizando así el consumo de energía y hardware.

1.1 Una variable a la vez

El enfoque de diseño de experimentos de una variable a la vez (One Variable At a Time, OVAT) es una metodología simple en la que se varía un solo factor o variable experimental mientras se mantienen constantes todos los demás factores. Este enfoque permite observar cómo cambios en esa única variable afectan el resultado del experimento.

Equipo:

-Elias Nieto Víctor David

-Pérez Lucio Kevyn Alejandro

-Rojas Alarcon Sergio Ulises

-Trejo Arriaga Rodrigo Gerardo

[23]:
```
# Cargamos paquetes necesarios

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import matplotlib.pyplot as plt
import numpy as np
import time
import pandas as pd
import torch
from torch import nn
from torch import optim
import torch.nn.functional as F
```

```
from torchvision import datasets, transforms
```

1.1.1 1. Definición de variables y rango de valores:

Identificar los factores o variables experimentales clave que se desean estudiar. En el contexto de redes neuronales, estas variables pueden incluir la tasa de aprendizaje, el número de capas, el número de neuronas por capa, el tamaño del lote, entre otros.

Variables que tomaremos en cuenta:

- Tasa de aprendizaje (eta)
- Número de épocas (n_epocas)
- Tamaño de lote (batch_size)

Una vez definidas las variables independientes definimos un rango de valores posibles para cada variable.

> TODO: Define un rango de valores posibles para cada variable. Incluye el valor mínimo y el valor máximo. Se sugiere utilizar una lista de valores obtenida con una separación uniforme. Probar almenos 5 valores por variable.

[24]:
```
# Tasa de aprendizaje
eta_values = [0.1, 0.01, 0.001, 0.0001, 0.00001]

# Número de épocas
n_epocas_values = [10, 15, 20, 25, 30]

# Tamaño de lote
batch_size_values = [16, 32, 64, 128, 256]
```

1.1.2 2. Configuración inicial:

Establecer una configuración inicial para la red neuronal con valores predeterminados para todos los hiperparámetros.

> TODO: Define la configuración inicial. Se sugiere usar un diccionario para contener dicha configuración.

[25]:
```
configuracion = {
    'epochs': 10,
    'batch_size': 64
}
```

1.1.3 3. Variación de una variable a la vez:

- Seleccionar la primera variable a estudiar (por ejemplo, la tasa de aprendizaje).
- Realizar una serie de experimentos donde se varía únicamente la tasa de aprendizaje, mientras se mantienen constantes todos los demás hiperparámetros.
- Registrar el rendimiento del modelo para cada valor de la tasa de aprendizaje.

> TODO: Modifica el código para que pueda aceptar la configuración deseada

[26]:
```
# Definimos una transformación de los datos
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])

# Descargamos el conjunto de entrenamiento y cargamos mediante un dataLoader
trainset = datasets.FashionMNIST('F_MNIST_data/', download=True, train=True,␣
 ↪transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

# Descargamos el conjunto de validación
validationset = datasets.FashionMNIST('F_MNIST_data/', download=True,␣
 ↪train=False, transform=transform)
validationloader = torch.utils.data.DataLoader(validationset, batch_size=64,␣
 ↪shuffle=True)
```

[27]:
```
# Definición del modelo de red neuronal
class RedNeuronal(nn.Module):
    def __init__(self, input_size, output_size, hidden_layers, drop_p=0.5):
        super().__init__()

        # Agregamos la primera capa
        self.hidden_layers = nn.ModuleList([nn.Linear(input_size,␣
 ↪hidden_layers[0])])

        # Agregamos cada una de las capas, zip empareja el número de entradas␣
 ↪con las salidas
        layer_sizes = zip(hidden_layers[:-1], hidden_layers[1:])
        self.hidden_layers.extend([nn.Linear(h1, h2) for h1, h2 in layer_sizes])

        # Agregamos la capa de salida final de la red
        self.output = nn.Linear(hidden_layers[-1], output_size)

        # Incluimos drop-out en la red
        self.dropout = nn.Dropout(p=drop_p)

    def forward(self, x):
        ''' Pase hacia adelante en la red, el regreso son las probabilidades en␣
 ↪el dominio log '''
        for linear in self.hidden_layers:
            x = F.relu(linear(x))
            x = self.dropout(x)

        x = self.output(x)

        return F.log_softmax(x, dim=1)
```

[28]:
```
def validation(model, validationloader, criterion):
    test_loss = 0
    accuracy = 0
    for images, labels in validationloader:
        images.resize_(images.shape[0], 784)
        output = model.forward(images)
        test_loss += criterion(output, labels).item()
        ps = torch.exp(output)
        equality = (labels.data == ps.max(dim=1)[1])
        accuracy += equality.type(torch.FloatTensor).mean()
    return test_loss, accuracy
```

[29]:
```
results = []

for i, eta in enumerate(eta_values):
    print(f"Experimento {i+1}: Evaluando tasa de aprendizaje {eta}")
    trainloader = torch.utils.data.DataLoader(trainset,␣
 ↪batch_size=configuracion['batch_size'], shuffle=True)
    validationloader = torch.utils.data.DataLoader(validationset,␣
 ↪batch_size=configuracion['batch_size'], shuffle=True)

    model = RedNeuronal(784, 10, [516, 256], drop_p=0.5)
    criterion = nn.NLLLoss()
    optimizer = optim.Adam(model.parameters(), lr=eta)
    epochs = configuracion['epochs']
    steps = 0
    running_loss = 0
    print_every = 40

    for e in range(epochs):
        model.train()
        for images, labels in trainloader:
            steps += 1
            images.resize_(images.size()[0], 784)
            optimizer.zero_grad()
            output = model.forward(images)
            loss = criterion(output, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        model.eval()
        with torch.no_grad():
            test_loss, accuracy = validation(model, validationloader, criterion)
        print(f"Epoch: {e+1}/{epochs}.. ",
              f"Tasa de aprendizaje: {eta}.. ",
              f"Pérdida de entrenamiento: {running_loss/print_every:.3f}.. ",
```

```
            f"Pérdida de validación: {test_loss/len(validationloader):.3f}..␣
 ↪",
            f"Exactitud de validación: {accuracy/len(validationloader):.3f}")
      results.append([eta, epochs, configuracion['batch_size'], running_loss/
 ↪print_every, test_loss/len(validationloader), accuracy/
 ↪len(validationloader)])
      running_loss = 0
      model.train()

df_results = pd.DataFrame(results, columns=['Learning Rate', 'Epochs', 'Batch␣
 ↪Size', 'Training Loss', 'Validation Loss', 'Validation Accuracy'])
```

```
Experimento 1: Evaluando tasa de aprendizaje 0.1
Epoch: 1/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 673.249..
Pérdida de validación: 2.818..  Exactitud de validación: 0.101
Epoch: 2/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 580.527..
Pérdida de validación: 2.624..  Exactitud de validación: 0.100
Epoch: 3/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 1127.660..
Pérdida de validación: 3.945..  Exactitud de validación: 0.100
Epoch: 4/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 1409.957..
Pérdida de validación: 7.246..  Exactitud de validación: 0.100
Epoch: 5/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 1452.069..
Pérdida de validación: 2.489..  Exactitud de validación: 0.100
Epoch: 6/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 1405.175..
Pérdida de validación: 2.332..  Exactitud de validación: 0.100
Epoch: 7/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 1543.589..
Pérdida de validación: 9.648..  Exactitud de validación: 0.100
Epoch: 8/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 1878.306..
Pérdida de validación: 4.151..  Exactitud de validación: 0.101
Epoch: 9/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 465.760..
Pérdida de validación: 2.310..  Exactitud de validación: 0.100
Epoch: 10/10..  Tasa de aprendizaje: 0.1..  Pérdida de entrenamiento: 657.126..
Pérdida de validación: 2.308..  Exactitud de validación: 0.100
Experimento 2: Evaluando tasa de aprendizaje 0.01
Epoch: 1/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 27.881..
Pérdida de validación: 0.810..  Exactitud de validación: 0.693
Epoch: 2/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 27.729..
Pérdida de validación: 0.837..  Exactitud de validación: 0.642
Epoch: 3/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 26.668..
Pérdida de validación: 0.736..  Exactitud de validación: 0.716
Epoch: 4/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 27.129..
Pérdida de validación: 0.777..  Exactitud de validación: 0.672
Epoch: 5/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 28.141..
Pérdida de validación: 0.731..  Exactitud de validación: 0.726
Epoch: 6/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 27.093..
Pérdida de validación: 0.710..  Exactitud de validación: 0.748
Epoch: 7/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 27.683..
```

```
Pérdida de validación: 0.935..  Exactitud de validación: 0.641
Epoch: 8/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 27.658..
Pérdida de validación: 0.812..  Exactitud de validación: 0.687
Epoch: 9/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 29.020..
Pérdida de validación: 1.056..  Exactitud de validación: 0.533
Epoch: 10/10..  Tasa de aprendizaje: 0.01..  Pérdida de entrenamiento: 28.056..
Pérdida de validación: 0.876..  Exactitud de validación: 0.659
Experimento 3: Evaluando tasa de aprendizaje 0.001
Epoch: 1/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 14.401..
Pérdida de validación: 0.456..  Exactitud de validación: 0.833
Epoch: 2/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 11.272..
Pérdida de validación: 0.415..  Exactitud de validación: 0.847
Epoch: 3/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 10.589..
Pérdida de validación: 0.397..  Exactitud de validación: 0.854
Epoch: 4/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 10.224..
Pérdida de validación: 0.390..  Exactitud de validación: 0.857
Epoch: 5/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 10.015..
Pérdida de validación: 0.392..  Exactitud de validación: 0.858
Epoch: 6/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 9.651..
Pérdida de validación: 0.378..  Exactitud de validación: 0.864
Epoch: 7/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 9.516..
Pérdida de validación: 0.378..  Exactitud de validación: 0.861
Epoch: 8/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 9.272..
Pérdida de validación: 0.371..  Exactitud de validación: 0.866
Epoch: 9/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 9.059..
Pérdida de validación: 0.372..  Exactitud de validación: 0.863
Epoch: 10/10..  Tasa de aprendizaje: 0.001..  Pérdida de entrenamiento: 8.941..
Pérdida de validación: 0.358..  Exactitud de validación: 0.868
Experimento 4: Evaluando tasa de aprendizaje 0.0001
Epoch: 1/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 18.755..
Pérdida de validación: 0.520..  Exactitud de validación: 0.808
Epoch: 2/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 12.104..
Pérdida de validación: 0.458..  Exactitud de validación: 0.832
Epoch: 3/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 10.789..
Pérdida de validación: 0.424..  Exactitud de validación: 0.846
Epoch: 4/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 10.063..
Pérdida de validación: 0.407..  Exactitud de validación: 0.852
Epoch: 5/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 9.545..
Pérdida de validación: 0.398..  Exactitud de validación: 0.855
Epoch: 6/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 9.100..
Pérdida de validación: 0.385..  Exactitud de validación: 0.860
Epoch: 7/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 8.809..
Pérdida de validación: 0.371..  Exactitud de validación: 0.864
Epoch: 8/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 8.544..
Pérdida de validación: 0.376..  Exactitud de validación: 0.865
Epoch: 9/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 8.283..
Pérdida de validación: 0.359..  Exactitud de validación: 0.867
Epoch: 10/10..  Tasa de aprendizaje: 0.0001..  Pérdida de entrenamiento: 8.075..
```

```
Pérdida de validación: 0.356..  Exactitud de validación: 0.870
Experimento 5: Evaluando tasa de aprendizaje 1e-05
Epoch: 1/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 38.791..
Pérdida de validación: 0.984..  Exactitud de validación: 0.716
Epoch: 2/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 22.375..
Pérdida de validación: 0.731..  Exactitud de validación: 0.745
Epoch: 3/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 18.599..
Pérdida de validación: 0.654..  Exactitud de validación: 0.762
Epoch: 4/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 16.858..
Pérdida de validación: 0.613..  Exactitud de validación: 0.774
Epoch: 5/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 15.704..
Pérdida de validación: 0.583..  Exactitud de validación: 0.785
Epoch: 6/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 14.872..
Pérdida de validación: 0.558..  Exactitud de validación: 0.795
Epoch: 7/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 14.169..
Pérdida de validación: 0.538..  Exactitud de validación: 0.802
Epoch: 8/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 13.570..
Pérdida de validación: 0.522..  Exactitud de validación: 0.810
Epoch: 9/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 13.161..
Pérdida de validación: 0.511..  Exactitud de validación: 0.814
Epoch: 10/10..  Tasa de aprendizaje: 1e-05..  Pérdida de entrenamiento: 12.799..
Pérdida de validación: 0.499..  Exactitud de validación: 0.818
```

1.1.4 4. Selección del mejor valor:

Analizar los resultados y seleccionar el valor de la tasa de aprendizaje que produce el mejor rendimiento del modelo.

[36]:
```
# Si se ejecuta por primera vez, descomentar la linea siguiente
# df_results['Validation Accuracy'] = df_results['Validation Accuracy'].
 ↪apply(lambda x: x.item())
last_epoch_results = df_results.groupby(['Learning Rate']).tail(1).
 ↪reset_index(drop=True)
best_result = last_epoch_results.loc[last_epoch_results['Validation Accuracy'].
 ↪idxmax()]
```

[37]:
```
last_epoch_results
```

[37]:

	Learning Rate	Epochs	Batch Size	Training Loss	Validation Loss	Validation Accuracy
0	0.10000	10	64	657.125797	2.307695	0.099622
1	0.01000	10	64	28.055921	0.876105	0.658838
2	0.00100	10	64	8.940564	0.357506	
3	0.00010	10	64	8.074979	0.355960	
4	0.00001	10	64	12.799471	0.499012	

```
2             0.868332
3             0.869825
4             0.818073
```

[38]:
```
print("\nMejor tasa de aprendizaje:")
print(best_result['Learning Rate'])
print("\nResultados asociados al mejor valor:")
print(best_result)
```

```
Mejor tasa de aprendizaje:
0.0001

Resultados asociados al mejor valor:
Learning Rate          0.000100
Epochs                10.000000
Batch Size            64.000000
Training Loss          8.074979
Validation Loss        0.355960
Validation Accuracy    0.869825
Name: 3, dtype: float64
```

1.1.5 5. Repetición para otras variables:

Proceder con la siguiente variable (por ejemplo, el número de capas) y repetir el proceso: variar solo esta variable mientras se mantienen constantes todos los demás hiperparámetros, utilizando el mejor valor encontrado para la tasa de aprendizaje. Continuar este proceso para cada variable en la lista.

> TODO: Escribe una tabla con el resultado de cada experimento. Las columnas deben ser: ID, Configuración, Exactitud obtenida.

Fijamos el learning rate en 0.0001 y variamos el tamaño del batch

[39]:
```
learning_rate = 0.0001
epochs = 10
results = []

for i, batch_size in enumerate(batch_size_values):
    print(f"Experimento {i+1}: Evaluando tamaño de lote {batch_size}")
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,␣
 ↪shuffle=True)
    validationloader = torch.utils.data.DataLoader(validationset,␣
 ↪batch_size=batch_size, shuffle=True)

    model = RedNeuronal(784, 10, [516, 256], drop_p=0.5)
    criterion = nn.NLLLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
    steps = 0
    running_loss = 0
    print_every = 40

    for e in range(epochs):
        model.train()
        for images, labels in trainloader:
            steps += 1
            images.resize_(images.size()[0], 784)
            optimizer.zero_grad()
            output = model.forward(images)
            loss = criterion(output, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        model.eval()
        with torch.no_grad():
            test_loss, accuracy = validation(model, validationloader, criterion)
        print(f"Epoch: {e+1}/{epochs}.. ",
              f"Tamaño de lote: {batch_size}.. ",
              f"Pérdida de entrenamiento: {running_loss/print_every:.3f}.. ",
              f"Pérdida de validación: {test_loss/len(validationloader):.3f}..␣
 ↪",
              f"Exactitud de validación: {accuracy/len(validationloader):.3f}")
        results.append([learning_rate, epochs, batch_size, running_loss/
 ↪print_every, test_loss/len(validationloader), accuracy/
 ↪len(validationloader)])
        running_loss = 0
        model.train()

df_batch_results = pd.DataFrame(results, columns=['Learning Rate', 'Epochs',␣
 ↪'Batch Size', 'Training Loss', 'Validation Loss', 'Validation Accuracy'])
```

```
Experimento 1: Evaluando tamaño de lote 16
Epoch: 1/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 63.151..  Pérdida
de validación: 0.463..  Exactitud de validación: 0.830
Epoch: 2/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 43.787..  Pérdida
de validación: 0.417..  Exactitud de validación: 0.846
Epoch: 3/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 39.576..  Pérdida
de validación: 0.402..  Exactitud de validación: 0.853
Epoch: 4/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 37.214..  Pérdida
de validación: 0.382..  Exactitud de validación: 0.859
Epoch: 5/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 35.398..  Pérdida
de validación: 0.370..  Exactitud de validación: 0.865
Epoch: 6/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 34.101..  Pérdida
de validación: 0.363..  Exactitud de validación: 0.871
```

```
Epoch: 7/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 33.056..  Pérdida
de validación: 0.354..  Exactitud de validación: 0.873
Epoch: 8/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 31.928..  Pérdida
de validación: 0.344..  Exactitud de validación: 0.877
Epoch: 9/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 31.222..  Pérdida
de validación: 0.336..  Exactitud de validación: 0.881
Epoch: 10/10..  Tamaño de lote: 16..  Pérdida de entrenamiento: 30.526..
Pérdida de validación: 0.344..  Exactitud de validación: 0.874
Experimento 2: Evaluando tamaño de lote 32
Epoch: 1/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 34.179..  Pérdida
de validación: 0.488..  Exactitud de validación: 0.822
Epoch: 2/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 23.015..  Pérdida
de validación: 0.436..  Exactitud de validación: 0.841
Epoch: 3/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 20.559..  Pérdida
de validación: 0.423..  Exactitud de validación: 0.846
Epoch: 4/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 19.253..  Pérdida
de validación: 0.391..  Exactitud de validación: 0.858
Epoch: 5/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 18.247..  Pérdida
de validación: 0.380..  Exactitud de validación: 0.861
Epoch: 6/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 17.494..  Pérdida
de validación: 0.374..  Exactitud de validación: 0.865
Epoch: 7/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 16.847..  Pérdida
de validación: 0.361..  Exactitud de validación: 0.868
Epoch: 8/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 16.424..  Pérdida
de validación: 0.352..  Exactitud de validación: 0.874
Epoch: 9/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 15.873..  Pérdida
de validación: 0.346..  Exactitud de validación: 0.873
Epoch: 10/10..  Tamaño de lote: 32..  Pérdida de entrenamiento: 15.494..
Pérdida de validación: 0.347..  Exactitud de validación: 0.877
Experimento 3: Evaluando tamaño de lote 64
Epoch: 1/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 18.915..  Pérdida
de validación: 0.521..  Exactitud de validación: 0.808
Epoch: 2/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 12.187..  Pérdida
de validación: 0.461..  Exactitud de validación: 0.833
Epoch: 3/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 10.819..  Pérdida
de validación: 0.430..  Exactitud de validación: 0.840
Epoch: 4/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 10.085..  Pérdida
de validación: 0.410..  Exactitud de validación: 0.851
Epoch: 5/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 9.543..  Pérdida
de validación: 0.391..  Exactitud de validación: 0.858
Epoch: 6/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 9.160..  Pérdida
de validación: 0.386..  Exactitud de validación: 0.860
Epoch: 7/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 8.841..  Pérdida
de validación: 0.378..  Exactitud de validación: 0.862
Epoch: 8/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 8.537..  Pérdida
de validación: 0.367..  Exactitud de validación: 0.866
Epoch: 9/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 8.297..  Pérdida
de validación: 0.360..  Exactitud de validación: 0.871
```

```
Epoch: 10/10..  Tamaño de lote: 64..  Pérdida de entrenamiento: 8.065..  Pérdida
de validación: 0.352..  Exactitud de validación: 0.872
Experimento 4: Evaluando tamaño de lote 128
Epoch: 1/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 10.739..
Pérdida de validación: 0.558..  Exactitud de validación: 0.796
Epoch: 2/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 6.565..  Pérdida
de validación: 0.483..  Exactitud de validación: 0.823
Epoch: 3/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 5.759..  Pérdida
de validación: 0.445..  Exactitud de validación: 0.836
Epoch: 4/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 5.374..  Pérdida
de validación: 0.427..  Exactitud de validación: 0.844
Epoch: 5/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 5.042..  Pérdida
de validación: 0.415..  Exactitud de validación: 0.848
Epoch: 6/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 4.821..  Pérdida
de validación: 0.395..  Exactitud de validación: 0.856
Epoch: 7/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 4.650..  Pérdida
de validación: 0.384..  Exactitud de validación: 0.860
Epoch: 8/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 4.539..  Pérdida
de validación: 0.376..  Exactitud de validación: 0.864
Epoch: 9/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 4.379..  Pérdida
de validación: 0.376..  Exactitud de validación: 0.863
Epoch: 10/10..  Tamaño de lote: 128..  Pérdida de entrenamiento: 4.265..
Pérdida de validación: 0.367..  Exactitud de validación: 0.868
Experimento 5: Evaluando tamaño de lote 256
Epoch: 1/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 6.277..  Pérdida
de validación: 0.604..  Exactitud de validación: 0.779
Epoch: 2/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 3.647..  Pérdida
de validación: 0.521..  Exactitud de validación: 0.805
Epoch: 3/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 3.163..  Pérdida
de validación: 0.478..  Exactitud de validación: 0.824
Epoch: 4/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 2.914..  Pérdida
de validación: 0.456..  Exactitud de validación: 0.834
Epoch: 5/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 2.744..  Pérdida
de validación: 0.436..  Exactitud de validación: 0.842
Epoch: 6/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 2.615..  Pérdida
de validación: 0.424..  Exactitud de validación: 0.842
Epoch: 7/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 2.515..  Pérdida
de validación: 0.420..  Exactitud de validación: 0.844
Epoch: 8/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 2.435..  Pérdida
de validación: 0.417..  Exactitud de validación: 0.849
Epoch: 9/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 2.362..  Pérdida
de validación: 0.402..  Exactitud de validación: 0.855
Epoch: 10/10..  Tamaño de lote: 256..  Pérdida de entrenamiento: 2.307..
Pérdida de validación: 0.384..  Exactitud de validación: 0.860
```

[41]:
```
df_last_epoch = df_results[df_results['Epochs'] == epochs]
```

[43]:
```
best_result = df_last_epoch.loc[df_last_epoch['Validation Accuracy'].idxmax()]
```

[46]:
```
best_result
```

[46]:
```
Learning Rate          0.000100
Epochs                10.000000
Batch Size            64.000000
Training Loss          8.074979
Validation Loss        0.355960
Validation Accuracy    0.869825
Name: 39, dtype: float64
```

[48]:
```
print("\nMejor tasa de aprendizaje:")
print(best_result['Batch Size'])
print("\nResultados asociados al mejor valor:")
print(best_result)
```

```
Mejor tasa de aprendizaje:
64.0

Resultados asociados al mejor valor:
Learning Rate          0.000100
Epochs                10.000000
Batch Size            64.000000
Training Loss          8.074979
Validation Loss        0.355960
Validation Accuracy    0.869825
Name: 39, dtype: float64
```

Fijamos el learning rate en 0.0001 y el tamaño del batch en 64 mientras variamos el numero de épocas

[52]:
```
learning_rate = 0.0001
batch_size = 64

results = []

for i, epochs in enumerate(n_epocas_values):
    print(f"Experimento {i+1}: Evaluando número de épocas {epochs}")
    model = RedNeuronal(784, 10, [516, 256], drop_p=0.5)
    criterion = nn.NLLLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    print_every = 40

    for e in range(epochs):
        model.train()
        running_loss = 0
```

```
        for images, labels in trainloader:
            images.resize_(images.size()[0], 784)
            optimizer.zero_grad()
            output = model.forward(images)
            loss = criterion(output, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        model.eval()
        with torch.no_grad():
            test_loss, accuracy = validation(model, validationloader, criterion)
        print(f"Epoch: {e+1}/{epochs}.. ",
              f"Número de épocas: {epochs}.. ",
              f"Pérdida de entrenamiento: {running_loss/len(trainloader):.3f}..␣
 ↪",
              f"Pérdida de validación: {test_loss/len(validationloader):.3f}..␣
 ↪",
              f"Exactitud de validación: {accuracy/len(validationloader):.3f}")
        results.append([learning_rate, epochs, running_loss/len(trainloader),␣
 ↪test_loss/len(validationloader), accuracy/len(validationloader)])
        model.train()


df_results = pd.DataFrame(results, columns=['Learning Rate', 'Epochs',␣
 ↪'Training Loss', 'Validation Loss', 'Validation Accuracy'])
```

```
Experimento 1: Evaluando número de épocas 5
Epoch: 1/5..  Número de épocas: 5..  Pérdida de entrenamiento: 1.066..  Pérdida
de validación: 0.612..  Exactitud de validación: 0.777
Epoch: 2/5..  Número de épocas: 5..  Pérdida de entrenamiento: 0.619..  Pérdida
de validación: 0.524..  Exactitud de validación: 0.806
Epoch: 3/5..  Número de épocas: 5..  Pérdida de entrenamiento: 0.538..  Pérdida
de validación: 0.473..  Exactitud de validación: 0.825
Epoch: 4/5..  Número de épocas: 5..  Pérdida de entrenamiento: 0.495..  Pérdida
de validación: 0.458..  Exactitud de validación: 0.831
Epoch: 5/5..  Número de épocas: 5..  Pérdida de entrenamiento: 0.464..  Pérdida
de validación: 0.443..  Exactitud de validación: 0.840
Experimento 2: Evaluando número de épocas 10
Epoch: 1/10..  Número de épocas: 10..  Pérdida de entrenamiento: 1.091..
Pérdida de validación: 0.619..  Exactitud de validación: 0.775
Epoch: 2/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.629..
Pérdida de validación: 0.521..  Exactitud de validación: 0.813
Epoch: 3/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.542..
Pérdida de validación: 0.489..  Exactitud de validación: 0.818
Epoch: 4/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.498..
Pérdida de validación: 0.457..  Exactitud de validación: 0.834
```

```
Epoch: 5/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.470..
Pérdida de validación: 0.436..  Exactitud de validación: 0.839
Epoch: 6/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.447..
Pérdida de validación: 0.430..  Exactitud de validación: 0.844
Epoch: 7/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.430..
Pérdida de validación: 0.412..  Exactitud de validación: 0.848
Epoch: 8/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.414..
Pérdida de validación: 0.419..  Exactitud de validación: 0.848
Epoch: 9/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.404..
Pérdida de validación: 0.391..  Exactitud de validación: 0.860
Epoch: 10/10..  Número de épocas: 10..  Pérdida de entrenamiento: 0.391..
Pérdida de validación: 0.393..  Exactitud de validación: 0.860
Experimento 3: Evaluando número de épocas 15
Epoch: 1/15..  Número de épocas: 15..  Pérdida de entrenamiento: 1.068..
Pérdida de validación: 0.609..  Exactitud de validación: 0.774
Epoch: 2/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.616..
Pérdida de validación: 0.525..  Exactitud de validación: 0.808
Epoch: 3/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.538..
Pérdida de validación: 0.480..  Exactitud de validación: 0.823
Epoch: 4/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.497..
Pérdida de validación: 0.454..  Exactitud de validación: 0.834
Epoch: 5/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.466..
Pérdida de validación: 0.437..  Exactitud de validación: 0.838
Epoch: 6/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.445..
Pérdida de validación: 0.422..  Exactitud de validación: 0.845
Epoch: 7/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.428..
Pérdida de validación: 0.414..  Exactitud de validación: 0.845
Epoch: 8/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.414..
Pérdida de validación: 0.411..  Exactitud de validación: 0.851
Epoch: 9/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.402..
Pérdida de validación: 0.391..  Exactitud de validación: 0.855
Epoch: 10/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.391..
Pérdida de validación: 0.385..  Exactitud de validación: 0.857
Epoch: 11/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.383..
Pérdida de validación: 0.383..  Exactitud de validación: 0.859
Epoch: 12/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.374..
Pérdida de validación: 0.382..  Exactitud de validación: 0.864
Epoch: 13/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.367..
Pérdida de validación: 0.375..  Exactitud de validación: 0.864
Epoch: 14/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.358..
Pérdida de validación: 0.363..  Exactitud de validación: 0.869
Epoch: 15/15..  Número de épocas: 15..  Pérdida de entrenamiento: 0.354..
Pérdida de validación: 0.362..  Exactitud de validación: 0.870
Experimento 4: Evaluando número de épocas 20
Epoch: 1/20..  Número de épocas: 20..  Pérdida de entrenamiento: 1.068..
Pérdida de validación: 0.613..  Exactitud de validación: 0.767
Epoch: 2/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.620..
Pérdida de validación: 0.522..  Exactitud de validación: 0.808
```

```
Epoch: 3/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.539..
Pérdida de validación: 0.482..  Exactitud de validación: 0.824
Epoch: 4/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.498..
Pérdida de validación: 0.455..  Exactitud de validación: 0.835
Epoch: 5/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.468..
Pérdida de validación: 0.437..  Exactitud de validación: 0.840
Epoch: 6/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.447..
Pérdida de validación: 0.425..  Exactitud de validación: 0.843
Epoch: 7/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.430..
Pérdida de validación: 0.410..  Exactitud de validación: 0.850
Epoch: 8/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.415..
Pérdida de validación: 0.399..  Exactitud de validación: 0.853
Epoch: 9/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.403..
Pérdida de validación: 0.393..  Exactitud de validación: 0.856
Epoch: 10/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.393..
Pérdida de validación: 0.384..  Exactitud de validación: 0.860
Epoch: 11/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.383..
Pérdida de validación: 0.378..  Exactitud de validación: 0.863
Epoch: 12/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.376..
Pérdida de validación: 0.378..  Exactitud de validación: 0.860
Epoch: 13/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.369..
Pérdida de validación: 0.371..  Exactitud de validación: 0.863
Epoch: 14/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.363..
Pérdida de validación: 0.361..  Exactitud de validación: 0.870
Epoch: 15/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.353..
Pérdida de validación: 0.358..  Exactitud de validación: 0.872
Epoch: 16/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.349..
Pérdida de validación: 0.362..  Exactitud de validación: 0.870
Epoch: 17/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.341..
Pérdida de validación: 0.352..  Exactitud de validación: 0.873
Epoch: 18/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.338..
Pérdida de validación: 0.350..  Exactitud de validación: 0.873
Epoch: 19/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.331..
Pérdida de validación: 0.346..  Exactitud de validación: 0.875
Epoch: 20/20..  Número de épocas: 20..  Pérdida de entrenamiento: 0.327..
Pérdida de validación: 0.347..  Exactitud de validación: 0.875
Experimento 5: Evaluando número de épocas 25
Epoch: 1/25..  Número de épocas: 25..  Pérdida de entrenamiento: 1.070..
Pérdida de validación: 0.608..  Exactitud de validación: 0.774
Epoch: 2/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.619..
Pérdida de validación: 0.523..  Exactitud de validación: 0.807
Epoch: 3/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.539..
Pérdida de validación: 0.476..  Exactitud de validación: 0.826
Epoch: 4/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.495..
Pérdida de validación: 0.458..  Exactitud de validación: 0.831
Epoch: 5/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.465..
Pérdida de validación: 0.437..  Exactitud de validación: 0.838
Epoch: 6/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.445..
```

```
Pérdida de validación: 0.421..  Exactitud de validación: 0.844
Epoch: 7/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.428..
Pérdida de validación: 0.425..  Exactitud de validación: 0.846
Epoch: 8/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.415..
Pérdida de validación: 0.407..  Exactitud de validación: 0.850
Epoch: 9/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.402..
Pérdida de validación: 0.389..  Exactitud de validación: 0.859
Epoch: 10/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.392..
Pérdida de validación: 0.389..  Exactitud de validación: 0.862
Epoch: 11/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.382..
Pérdida de validación: 0.386..  Exactitud de validación: 0.860
Epoch: 12/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.374..
Pérdida de validación: 0.373..  Exactitud de validación: 0.864
Epoch: 13/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.367..
Pérdida de validación: 0.367..  Exactitud de validación: 0.866
Epoch: 14/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.360..
Pérdida de validación: 0.363..  Exactitud de validación: 0.870
Epoch: 15/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.353..
Pérdida de validación: 0.370..  Exactitud de validación: 0.870
Epoch: 16/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.348..
Pérdida de validación: 0.356..  Exactitud de validación: 0.871
Epoch: 17/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.341..
Pérdida de validación: 0.349..  Exactitud de validación: 0.877
Epoch: 18/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.336..
Pérdida de validación: 0.354..  Exactitud de validación: 0.868
Epoch: 19/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.331..
Pérdida de validación: 0.342..  Exactitud de validación: 0.875
Epoch: 20/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.326..
Pérdida de validación: 0.339..  Exactitud de validación: 0.878
Epoch: 21/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.321..
Pérdida de validación: 0.345..  Exactitud de validación: 0.875
Epoch: 22/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.317..
Pérdida de validación: 0.342..  Exactitud de validación: 0.880
Epoch: 23/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.314..
Pérdida de validación: 0.344..  Exactitud de validación: 0.876
Epoch: 24/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.307..
Pérdida de validación: 0.342..  Exactitud de validación: 0.877
Epoch: 25/25..  Número de épocas: 25..  Pérdida de entrenamiento: 0.305..
Pérdida de validación: 0.335..  Exactitud de validación: 0.883
```

[56]:
```
df_last_epoch = df_results[df_results.groupby('Epochs')['Epochs'].
  ↪transform('max') == df_results['Epochs']]
best_result = df_last_epoch.loc[df_last_epoch['Validation Accuracy'].idxmax()]

print("\nMejor número de épocas:")
print(best_result['Epochs'])
print("\nResultados asociados al mejor valor:")
```

```
print(best_result)
```

```
Mejor número de épocas:
25

Resultados asociados al mejor valor:
Learning Rate                  0.0001
Epochs                             25
Training Loss                0.305473
Validation Loss              0.334719
Validation Accuracy    tensor(0.8825)
Name: 74, dtype: object
```

1.2 Conclusiones

¿Cual fue el mejor valor encontrado? ¿Cuantas ejecuciones se realizaron? ¿Que tiempo tomó realizar todos los experimentos?

1.2.1 Mejor valor encontrado:

El mejor valor encontrado en los experimentos fue con una tasa de aprendizaje de **0.0001**, un tamaño de lote de **64** y **25** épocas. Este conjunto de hiperparámetros resultó en la mejor exactitud de validación.

1.2.2 Número de ejecuciones realizadas:

En total, se realizaron **15** ejecuciones experimentales: - **5** ejecuciones variando la tasa de aprendizaje. - **5** ejecuciones variando el tamaño del lote. - **5** ejecuciones variando el número de épocas.

1.2.3 Tiempo total de los experimentos:

El tiempo total para realizar todos los experimentos fue de aproximadamente **35 minutos y 50 segundos**, desglosado de la siguiente manera: - Experimentos variando la tasa de aprendizaje: **11 minutos y 54 segundos**. - Experimentos variando el tamaño del lote: **11 minutos y 13 segundos**. - Experimentos variando el número de épocas: **12 minutos y 43 segundos**.

Estos resultados muestran que la red neuronal alcanza su mejor rendimiento con una configuración específica de hiperparámetros y que el tiempo de entrenamiento puede variar dependiendo de los valores de los hiperparámetros utilizados. La configuración óptima hallada puede ser utilizada como referencia para futuros modelos y tareas similares, optimizando así tanto el tiempo de entrenamiento como la precisión del modelo.

El diseño de experimentos es crucial en el entrenamiento de redes neuronales para identificar configuraciones óptimas de hiperparámetros. Utilizar la metodología de variar una variable a la vez (OVAT) ofrece varias ventajas significativas. En primer lugar, proporciona simplicidad y claridad al permitir observar el efecto directo de un solo hiperparámetro en el rendimiento del modelo, facilitando así la comprensión y el ajuste de los parámetros.

Además, OVAT reduce la complejidad computacional y es ideal para la optimización iterativa, permitiendo ajustes progresivos basados en los resultados obtenidos. Esto mejora la eficiencia de los programadores en proyectos de Deep Learning, al simplificar el proceso de optimización y permitir una mejor gestión de los recursos computacionales.