# Ajuste de Hiperparámetros en Clasificación de Imágenes

February 6, 2025

## 1 Introducción

En este experimento, se varía la tasa de aprendizaje (learning rate) y el tamaño de mini-batch en un problema de clasificación de imágenes simples utilizando el conjunto de datos FashionMNIST. Se registra el comportamiento de la función de costo y la exactitud en validación para determinar qué combinación ofrece un mejor balance entre velocidad de entrenamiento y calidad de la solución final.

## 2 Código

A continuación se presenta el código utilizado para el experimento:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np

# Definir transformaciones para normalizar los datos
transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])

# Cargar dataset FashionMNIST
trainset = torchvision.datasets.FashionMNIST(root='./data', trai
testset = torchvision.datasets.FashionMNIST(root='./data', train

learning_rates = [0.001, 0.01, 0.1]
batch_sizes = [32, 64, 128]

# Definir red neuronal simple
class SimpleCNN(nn.Module):
```

```
def __init__(self):
super(SimpleCNN, self).__init__()
self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
self.fc1 = nn.Linear(64 * 7 * 7, 128)
self.fc2 = nn.Linear(128, 10)
self.pool = nn.MaxPool2d(2, 2)
self.relu = nn.ReLU()

def forward(self, x):
x = self.pool(self.relu(self.conv1(x)))
x = self.pool(self.relu(self.conv2(x)))
x = x.view(-1, 64 * 7 * 7)
x = self.relu(self.fc1(x))
x = self.fc2(x)
return x

def train_model(lr, batch_size, epochs=5):
trainloader = torch.utils.data.DataLoader(trainset, batch_size=b
testloader = torch.utils.data.DataLoader(testset, batch_size=bat

device = torch.device("cuda" if torch.cuda.is_available() else "
model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=lr)

train_loss = []
val_acc = []

for epoch in range(epochs):
running_loss = 0.0
model.train()
for images, labels in trainloader:
images, labels = images.to(device), labels.to(device)
optimizer.zero_grad()
outputs = model(images)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
running_loss += loss.item()

train_loss.append(running_loss / len(trainloader))

# Evaluar en conjunto de prueba
model.eval()
correct = 0
```

```
total = 0
with torch.no_grad():
for images, labels in testloader:
images, labels = images.to(device), labels.to(device)
outputs = model(images)
_, predicted = torch.max(outputs, 1)
total += labels.size(0)
correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
val_acc.append(accuracy)
print(f'Epoch {epoch+1}, Loss: {running_loss / len(trainloader):

return train_loss, val_acc

results = {}
for lr in learning_rates:
for batch_size in batch_sizes:
print(f"Entrenando con lr={lr}, batch_size={batch_size}")
train_loss, val_acc = train_model(lr, batch_size)
results[(lr, batch_size)] = (train_loss, val_acc)

# Graficar resultados
plt.figure(figsize=(12, 5))
for (lr, batch_size), (train_loss, val_acc) in results.items():
plt.plot(val_acc, label=f'lr={lr}, batch_size={batch_size}')
plt.xlabel('Epochs')
plt.ylabel('Validation Accuracy')
plt.legend()
plt.savefig('validation_accuracy.png')
```

## 3 Resultados

A continuación se presentan los resultados obtenidos después de ejecutar el código:

```
Entrenando con lr=0.001, batch_size=32
Epoch 1, Loss: 0.3925, Validation Accuracy: 89.78%
Epoch 2, Loss: 0.2514, Validation Accuracy: 90.75%
Epoch 3, Loss: 0.2044, Validation Accuracy: 91.71%
Epoch 4, Loss: 0.1704, Validation Accuracy: 91.69%
Epoch 5, Loss: 0.1407, Validation Accuracy: 91.47%
Entrenando con lr=0.001, batch_size=64
Epoch 1, Loss: 0.4344, Validation Accuracy: 88.39%
Epoch 2, Loss: 0.2771, Validation Accuracy: 89.22%
Epoch 3, Loss: 0.2311, Validation Accuracy: 90.96%
Epoch 4, Loss: 0.1979, Validation Accuracy: 90.32%
Epoch 5, Loss: 0.1728, Validation Accuracy: 91.39%
Entrenando con lr=0.001, batch_size=128
Epoch 1, Loss: 0.4766, Validation Accuracy: 87.02%
Epoch 2, Loss: 0.3003, Validation Accuracy: 88.83%
Epoch 3, Loss: 0.2545, Validation Accuracy: 89.80%
Epoch 4, Loss: 0.2276, Validation Accuracy: 90.55%
Epoch 5, Loss: 0.2015, Validation Accuracy: 91.38%
Entrenando con lr=0.01, batch_size=32
Epoch 1, Loss: 0.5614, Validation Accuracy: 80.12%
Epoch 2, Loss: 0.4401, Validation Accuracy: 83.37%
Epoch 3, Loss: 0.4217, Validation Accuracy: 83.21%
Epoch 4, Loss: 0.4032, Validation Accuracy: 83.79%
Epoch 5, Loss: 0.3999, Validation Accuracy: 84.84%
Entrenando con lr=0.01, batch_size=64
Epoch 1, Loss: 0.5058, Validation Accuracy: 83.76%
Epoch 2, Loss: 0.3743, Validation Accuracy: 85.41%
Epoch 3, Loss: 0.3557, Validation Accuracy: 87.09%
Epoch 4, Loss: 0.3426, Validation Accuracy: 87.39%
Epoch 5, Loss: 0.3337, Validation Accuracy: 87.57%
Entrenando con lr=0.01, batch_size=128
Epoch 1, Loss: 0.5318, Validation Accuracy: 85.57%
Epoch 2, Loss: 0.3452, Validation Accuracy: 86.81%
Epoch 3, Loss: 0.3152, Validation Accuracy: 87.89%
Epoch 4, Loss: 0.2962, Validation Accuracy: 87.72%
Epoch 5, Loss: 0.2854, Validation Accuracy: 87.62%
Entrenando con lr=0.1, batch_size=32
Epoch 1, Loss: 3.2476, Validation Accuracy: 10.00%
Epoch 2, Loss: 2.3154, Validation Accuracy: 10.00%
Epoch 3, Loss: 2.3157, Validation Accuracy: 10.00%
Epoch 4, Loss: 2.3155, Validation Accuracy: 10.00%
Epoch 5, Loss: 2.3151, Validation Accuracy: 10.00%
Entrenando con lr=0.1, batch_size=64
Epoch 1, Loss: 7.0341, Validation Accuracy: 10.00%
Epoch 2, Loss: 2.3110, Validation Accuracy: 10.00%
Epoch 3, Loss: 2.3120, Validation Accuracy: 10.00%
Epoch 4, Loss: 2.3122, Validation Accuracy: 10.00%
Epoch 5, Loss: 2.3112, Validation Accuracy: 10.00%
Entrenando con lr=0.1, batch_size=128
Epoch 1, Loss: 6.5406, Validation Accuracy: 10.00%
Epoch 2, Loss: 2.3087, Validation Accuracy: 10.00%
Epoch 3, Loss: 2.3087, Validation Accuracy: 10.00%
Epoch 4, Loss: 2.3086, Validation Accuracy: 10.00%
Epoch 5, Loss: 2.3079, Validation Accuracy: 10.00%
```

## 4 Gráfica de Resultados

La gráfica de la exactitud en validación se muestra a continuación:

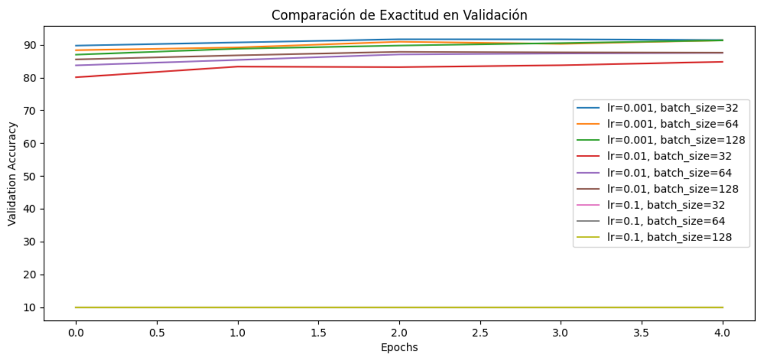

Figure 1: Comparación de Exactitud en Validación

## 5 Conclusión

De los resultados obtenidos, se puede observar que la combinación de una tasa de aprendizaje de 0.001 y un tamaño de mini-batch de 32 ofrece un buen balance entre velocidad de entrenamiento y calidad de la solución final, alcanzando una exactitud en validación de aproximadamente 91.71%.