César Isaí García Cornejo
cesar.cornejo@cimat.mx
Computo Científico

Tarea 11

Semestre 3

CIMAT

Ejercicio 1
Usando la base de datos MNIST realice lo siguiente y explique cada decisión tomada:

1. Diseñe una red neuronal de una sola capa oculta para la clasificación de las imágenes. Use una función de pérdida predefinida.
2. Entrene la red neuronal
3. Presente la matriz de confusión (Confusion matrix),
4. Describa que es la precisión y la exhaustividad (precisión and recall) y calculelo a partir de su matriz de confusión

Solución:
Los códigos se encuentran en aquí.

Procedemos a hacer cada paso en el siguiente orden:

1. Cargar la base de datos MNIST (entrenamiento y prueba)
2. Definir la red neuronal convolucional
3. Definir una función de perdida.
4. Entrenar la red con los datos de entrenamiento
5. Probar la red con los datos de prueba.

Cargamos la base MNIST usando *torchvision*. La salida de la base de datos de torchvision son imagenes PILImages de rango [0,1]. Transformamos los datos en tensores normalizados de rango [-1,1].

De igual forma con pytorch podemos construir la red neuronal de una sola capa, para ello observese en el código siguiente que se aplano la imagen en un vector, dado que la imagen es de 28×28 luego la entrada será de una capa de 784 neuronas. La capa de enmedio se diseño para tener 64 neuronas y finalmente la capa de salida tiene 10 neuronas ya que se desea clasificar para 10 clases que son cada uno de los dígitos.

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28 * 28, 64)  # Capa oculta con 64 neuronas
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(64, 10)  # Capa de salida con 10 neuronas

    def forward(self, x):
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return x
```

Tomemos 4 elementos de la base de datos con fines explorativos

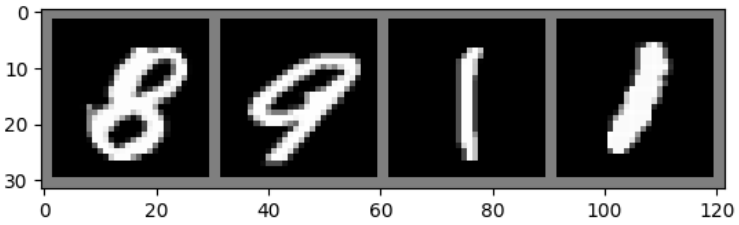

Figura 1: Grafica de 4 elementos aleatorios en la base MNIST

La clase correspondiente es el output:

```
Ocho  Nueve Uno   Uno
```

Empíricamente observamos que el número de épocas apropiado que disminuye el error de predicción es 6. Luego, entrenamos la red neuronal con 6 épocas. La función de activación es la función ReLu y la función de costos es la entropía cruzada a la cuál se optimiza con descenso de gradiente estocástico con momento.

Luego, con la base de prueba medimos la precisión de dicha red para la clasificación que obtuvo un valor de: 96 %.

Finalmente, la matriz de confusión es

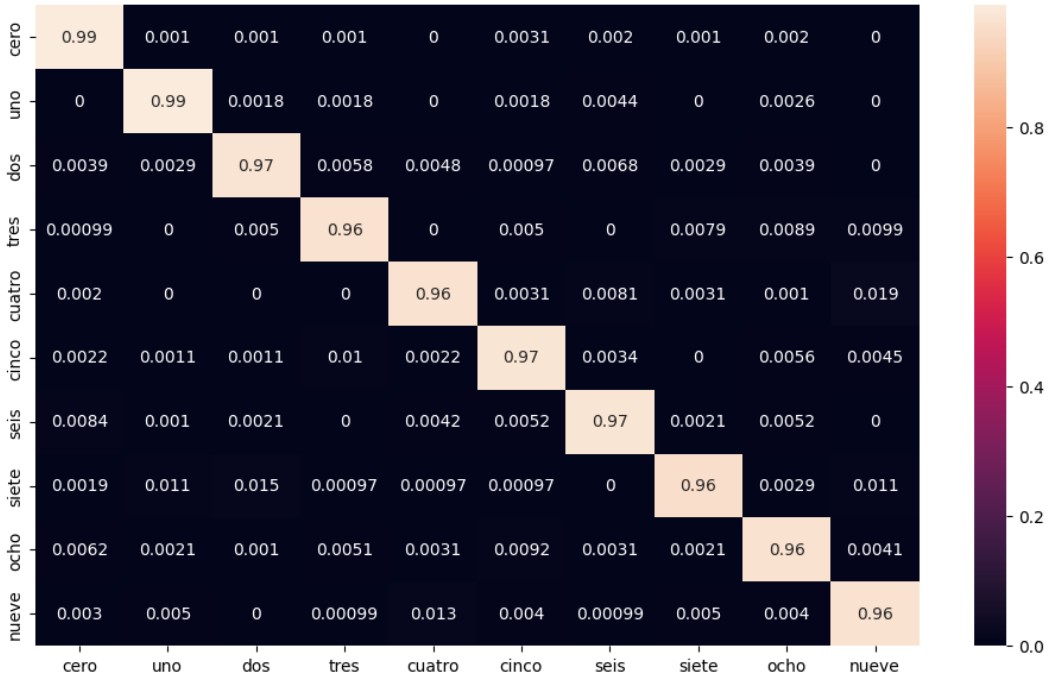

Figura 2: Matriz de confusión para la clasificación de la red neuronal entrenada con los datos de prueba en la base de datos MNIST.

Observamos que es un excelente clasificador, pues la diagonal tiene valores por encima de 96 % de precisión para cada dígito.

En el contexto de reconocimiento de patrones, se denomina precisión como a la fracción de instancias recuperadas que son relevantes, mientra que la exhaustividad es la fracción de instancias relevantes que han sido recuperadas. Tanto la precisión como la exhaustividad son entendidas como medidas de la relevancia [1].

La precisión es una proporción entre el número de documentos relevantes recuperados entre el número de documentos recuperados.

$$\text{Precisión} = \frac{|\{documentos\ relevantes\} \cap \{documentos\ recuperados\}|}{\{documentos\ recuperados\}}$$

La exhaustividad se emplea en menor medida que la precisión. Esta proporción viene a expresar la proporción de documentos relevantes recuperados comparado con el total de documentos que son relevantes.

$$\text{Exhaustividad} = \frac{|\{documentos\ relevantes\} \cap \{documentos\ recuperados\}|}{\{documentos\ relevantes\}}$$

De la matriz de confusión se deduce la precisión y la exhaustividad

```
Precisión: 0.9698
Exhaustividad (Recall): 0.9698
```

lo que concluye el ejercicio.

Ejercicio 2
Usando la base de datos Fashion MNIST realice o siguiente y explique cada decisión tomada:

1. Diseñe una red neuronal de dos capas ocultas para la clasificación de las imágenes, cada una con diferente función de activación y use una función de pérdida en la que el error de clasificar mal un calzado sea el doble que el resto de prendas.
2. Entrene la red neuronal
3. Presente la matriz de confusión (Confusion matrix).

Solución:

Cargamos los datos usando torchvision cargando FashionMNIST, tenemos images de 28*28 de un solo canal. Análogo al ejercicio previo, diseñamos una red neuronal completamente conectada, es decir es necesario aplanar la imagen en un vector 28*28. Luego, la primer capa oculta se propone de 256 capas, la segunda capa se propone de 64 capas y la capa de salida es de 10 neuronas, ya que hay 10 clases. Tomamos dos funciones de activación, la función ReLu para la primera capa, la función sigmoide para la segunda y para la capa de salida nuevamente usamos la función ReLu.

Con fines explorativos mostramos cuatro elementos de la base de datos Fashion MNIST tomados aleatoriamente.

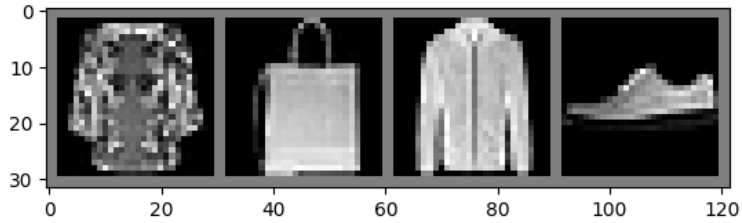

Figura 3: Grafica de 4 elementos aleatorios en la base FashionMNIST.

donde cada elemento se muestra con la clase

```
Shirt Bag   Coat  Sneaker
```

Veamos en el código siguiente la construcción de la red neuronal

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28 * 28, 256)  # Capa oculta con 256 neuronas
        self.fc2 = nn.Linear(256, 64)  # Capa oculta con 64 neuronas
        self.fc3 = nn.Linear(64, 10)  # Capa de salida con 10 neuronas

    def forward(self, x):
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        x = F.relu(self.fc3(x))
        return x
```

Para la función de perdida, damos un vector de pesos donde a cada clase le asocia peso 1 a excepción de la última clase que tiene peso 2, es decir la clase de *Ankle Boot*.

Luego, dichos pesos se mandan a llamar como argumento para la función de perdida de entropía cruzada. Es decir, simplemente la implementación es

```
        weight = torch.tensor([1.0] * 9 + [2.0])  # Asignar peso 2.0 a la última clase
        criterion = nn.CrossEntropyLoss(weight=weight)
        optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Nótese que el peso es concatenación, obteniendo un vector de 10 elementos.

Ahora, tras varios experimentos concluimos que el número de épocas ideales son 8. Entrenando la red neuronal y haciendo la prueba de la precisión con los datos de prueba se obtiene que la red neuronal clasifica con un 86 % de precisión.

Podemos ver los detalles de la clasificación con los datos de prueba en la matriz de confusión siguiente

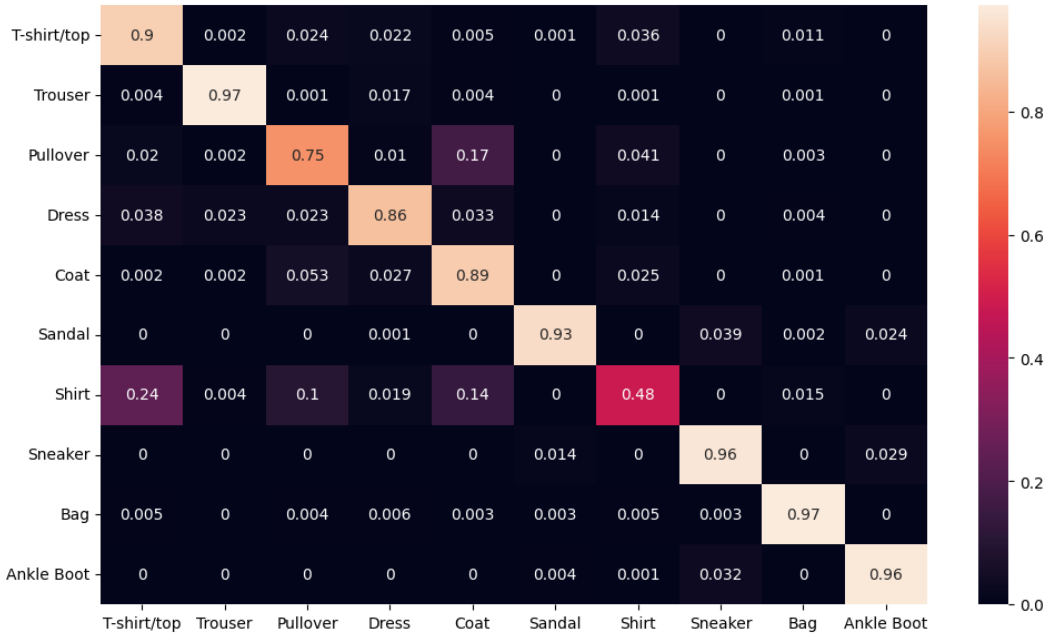

Figura 4: Matriz de confusión para la clasificación de la red neuronal entrenada con los datos de prueba en la base de datos MNIST.

Podemos ver que tiene valores muy altos en la diagonal, lo que significa que la mayoría de las prendas las consigue clasificar con precisión aceptable. Sin embargo, prendas como *Shirt* tiene problemas para decidir con precisión, de hecho vemos que el clasificador confunde *Shirt* con *T-Shirt* y también con *Coat*.

Ejercicio 3
Con la base de datos CIFAR-10 realice lo siguiente:

1. Diseñe una red neuronal de una capa oculta completamente conectada de al menos 10,000 neuronas para la clasificación de las imágenes.
2. Entrene la red neuronal
3. Presenta la matriz de confusión (Confusion matrix)
4. Entrene una segunda red neuronal usando la estructura de LeNet5
5. Presente la matriz de confusión (Confusion matrix)
6. Compare ambas redes. Para esta comparación será necesario que al menos presenta las curvas de error de entrenamiento y predicción para ambas redes, los tiempos de entrenamiento y que tome en cuenta las matrices de confusión.

Solución:

Observemos que la base de datos se compone de imagenes a color, es decir cada imagen tiene tres canales. Luego, se tiene que ajustar los parámetros en el diseño de la red y en la transformación de forma que sea compatible con el formato.

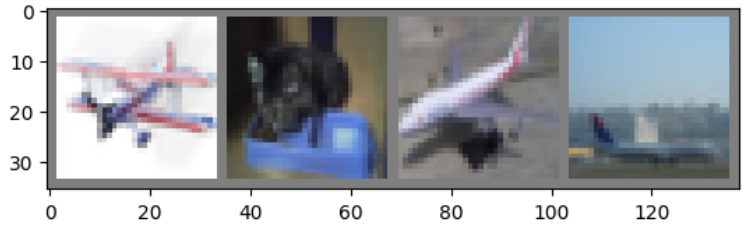

Figura 5: Grafica de 4 elementos aleatorios en la base CIFAR-10.

La primera estructura de red se compone de una capa completamente conectada con 10000 neuronas.

Entrenamos la red neuronal con 50 épocas y un batch size de 512, luego gráficamos la perdida de entrenamiento y de predicción obteniendo

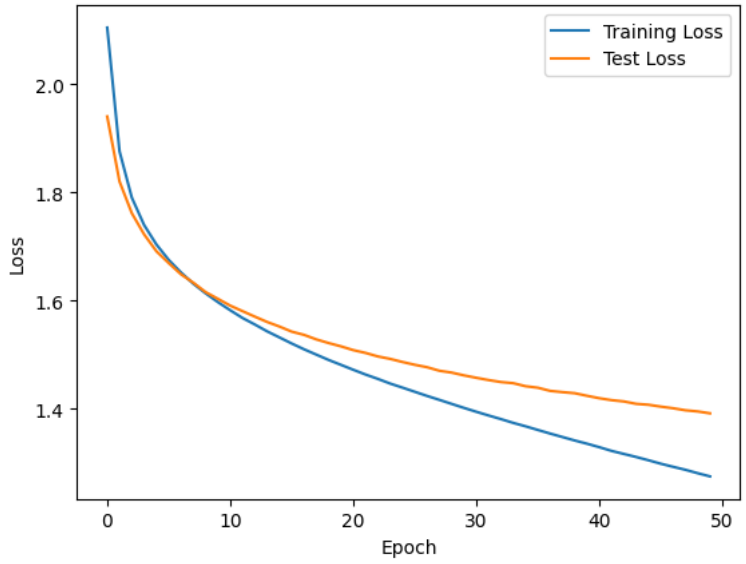

Figura 6: Grafica de la función de perdida de entrenamiento y predicción para la red neuronal de 10000 neuronas en una capa oculta.

Como vemos en la figura anterior, la perdida disminuye con cada época. Por tanto, nos quedamos con la red al máximo de épocas que es 50. Con esto vemos en el código que dicha red tiene una precisión con los datos de prueba del 51 %.

La matriz de confusión de la red neuronal como clasificador es

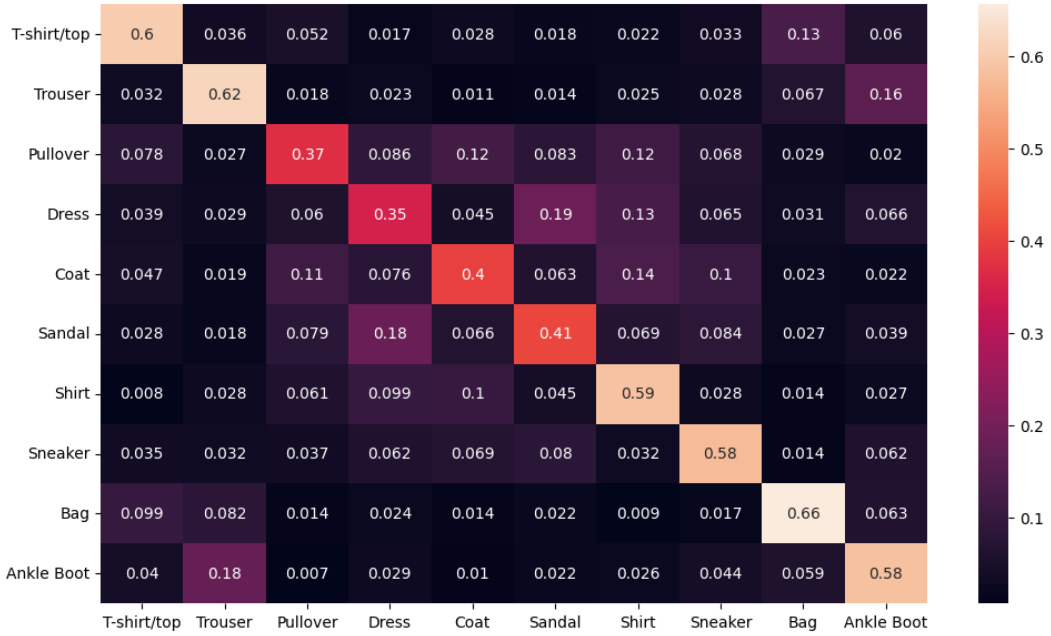

Figura 7: Matriz de confusión para la red neuronal de una capa oculta con 10000 neuronas.

En efecto vemos que en cada elemento se tiene una clasificación cercana al 50 %. Lo que no es un excelente clasificador pero es mejor que uno aleatorio con 10 % de precisión.

Luego, para la segunda estructura que es la correspondiente a la estructura LeNet 5, entrenamos la red neuronal con 40 épocas obteniendo la siguiente grafica de la función de perdida por época

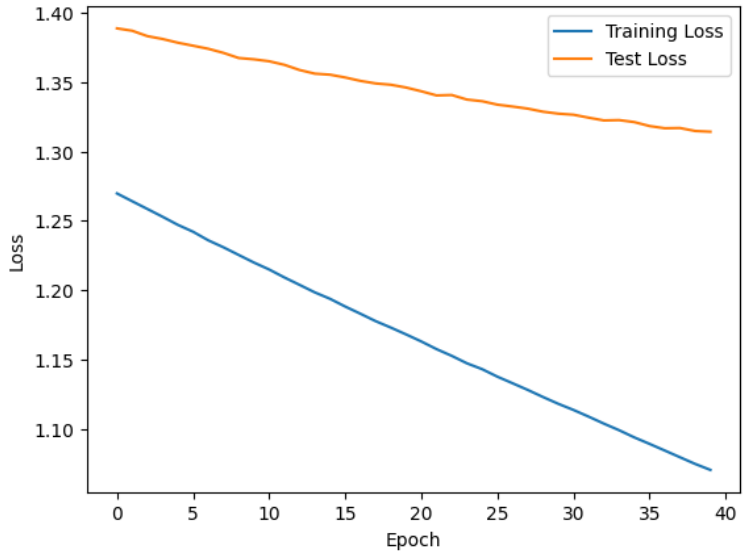

Figura 8: Gráfica de la función de perdida para la red neuronal de LeNet 5 con la base de datos de CIFAR-10.

Nuevamente, vemos que la función de perdida disminuye a medida con cada época. Luego tomamos la red neuronal entrenada con 40 épocas. La precisión con los datos de prueba nos da ligeramente mayor que la red neuronal que deseamos contrastar, con un 59 % de precisión. Vemos en su matriz de confusión los errores de clasificación.

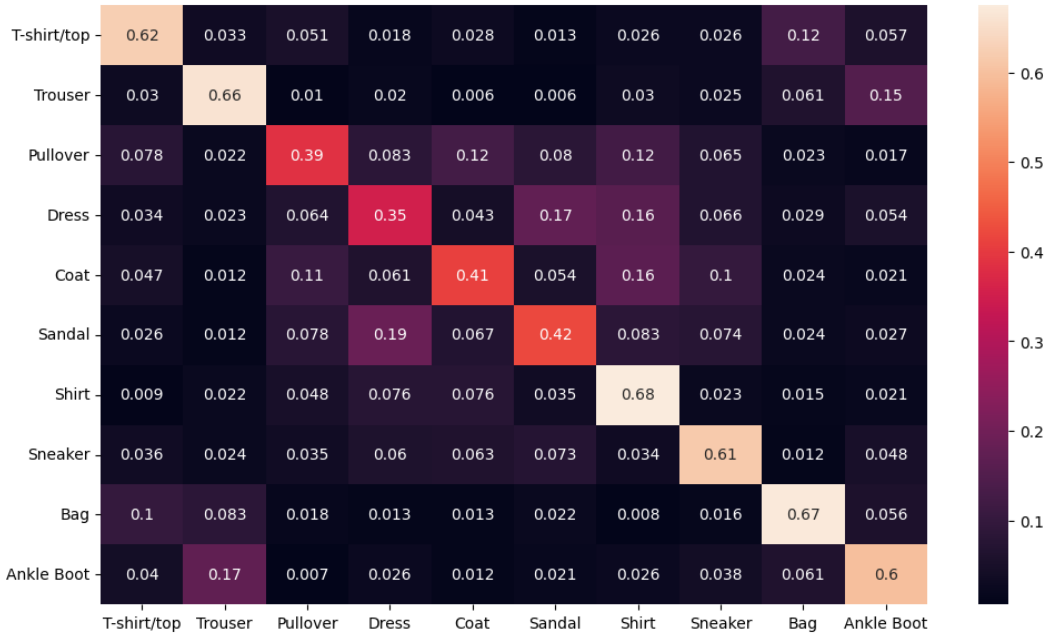

Figura 9: Matriz de confusión para la red neuronal LeNet 5.

La precisión de la red neuronal se aumenta bajando el número batch size, sin embargo algo muy bajo implica tiempos no considerables de cómputo que no bajan ni usando el GPU.

En consideraciones generales, la red LeNet5 fue mejor en precisión de prueba que la red simple sobrecargada, ya que se entreno 10 épocas menos y se obtuvo una precisión mayor a pesar de esto.

Referencias

[1] Precisión y exhaustividad. https://es.wikipedia.org/wiki/Precisi%C3%B3n_y_exhaustividad

[2] Tutorial redes neuronales. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html