Build and Train LeNet-5 model on MNIST dataset

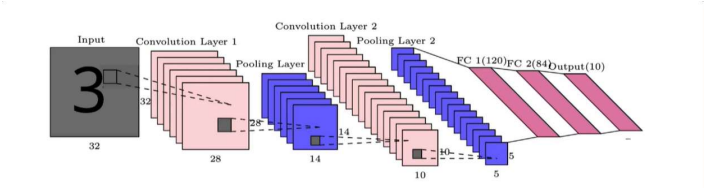
27 June 2023 19:44

What is MNIST ?
MNIST stands for Modified National Institute of Standards and Technology database .
MNIST dataset is a set of 70,000 small images of handwritten digits
The MNIST database contains 60,000 training images and 10,000 testing images.

What is CNN?
In deep learning, a convolutional neural network (CNN) is a class of artificial neural network most commonly applied to analyse visual imagery.[1] CNNs use a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers.[2] They are specifically designed to process pixel data and are used in image recognition and processing.

LeNet Model ->
LeNet is a convolutional neural network structure
The LeNet-5 model is trained on greyscale images and shape of images are 32 x 32 x 1
It is composed of 7 layers .
Goal of LeNet is to recognize handwritten images

7 layers->
Formula - n+2p-f/s +1
Input - 32 x 32 x 1
Convolution layer 1 - 6 filters of size 5 x 5, stride(compression factor) is 1
Size = 28 x 28 x 6
Max pooling - to reduce the size of image. Filter of size 2 x 2, stride is 2
Size = 14 x 14 x 6
Convolution layer 2 - 16 filters of size 5 x 5, stride is 1
Size = 10 x 10 x 16
Max pooling - Filter of size 2 x 2, stride is 2
Size = 5 x 5 16
(in above 4 layers operations- number of channels increased, image size got shrinked, all above 4 are convolution operations, next 3 layers are classification task)
(we need to convert 2D image to 1D array -> 5*5*16=400 pixels)
Now, 400 pixel will go to 120 neurons of fully connected layer, then to 84 neurons, last to 10 neurons



import numpy as np - numerical operations
import pandas as pd - data manipulation
import matplotlib.pyplot as plt - data visualizations
%matplotlib inline - formatting (display immeditly after code)
import seaborn as sns - data visualization. It is built on top of Matplotlib
import torch - library for deep learning and tensor computations
i**mport torchvision**- consists of popular datasets, model architectures
import torchvision.transforms as transforms
import torch.nn as nn - provides classes and functions for building neural networks
from torch.utils.data import DataLoader - efficient way to load and iterate over datasets
import torch.optim as optim - updating the parameters of neural network model

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
used to determine the device (CPU or GPU) on which the computations will be performed using PyTorch.

transform = transforms.Compose([
transforms.ToTensor(), - Converts the image data from PIL Image format to PyTorch Tensor format.
transforms.Normalize((0.5,), (0.5,))subtracting the mean value of 0.5 and dividing by the standard deviation of 0.5
])

trainset = torchvision.datasets.MNIST(root='./data', specifies the directory where the downloaded dataset will be stored.
train=True,
download=True,
transform=transform)

testset = torchvision.datasets.MNIST(root='./data',
train=False,
download=True,
transform=transform)

print('Len of train dataset ', len(trainset))
print('Len of test dataset ', len(testset))

```
print('-'*30)
print(trainset)
print(testset)
print('-'*30)
print(trainset[0])
print(testset[0])
print('-'*30)
```

batch_size = 128 - number of samples processed together in each iteration during training or testing.

```
trainloader = torch.utils.data.DataLoader(trainset,
                                          batch_size=batch_size,
                                          shuffle=True)

testloader = torch.utils.data.DataLoader(testset,
                                         batch_size=batch_size,
                                         shuffle=False)
```

classes = ('0','1','2','3','4','5','6','7','8','9') - Each digit from 0 to 9 represents a class.

```
dataiter = iter(trainloader)
```

images, labels = next(dataiter) - retrieves the next batch of data from the dataiter iterator. Each batch consists of a set of images and their corresponding labels.

plt.figure(figsize=(12,12)) - creates a new figure object for the plot with a specified figure size of 12x12 inches.
plt.subplot(321) - subplot grid with 3 rows and 2 columns and selects the first subplot (top left corner) as the current subplot to plot the images.

```
for i in range(36):
   ax1 = plt.subplot(6,6,i+1)
   plt.imshow(images[i].numpy().squeeze(), cmap='gray')
   plt.title(classes[labels[i].item()])
   plt.axis('off')
plt.show()
```

class LeNet(nn.Module): - In PyTorch, defining a neural network model involves creating a class that inherits from nn.Module

def __init__(self): - function calling

super(LeNet, self).__init__() - It ensures that the necessary setup for the parent class is performed before adding any custom functionality.

self.cnn_model = nn.Sequential(- The layers defined inside the nn.Sequential will be applied in order to the input data.

```
            nn.Conv2d(1, 6, 5),
            nn.Tanh(),
            nn.AvgPool2d(2, stride=2),
            nn.Conv2d(6, 16, 5),
            nn.Tanh(),
            nn.AvgPool2d(2, stride=2)
        )
        self.fc_model = nn.Sequential(
            nn.Linear(256, 120),
            nn.Tanh(),
            nn.Linear(120, 84),
            nn.Tanh(),
            nn.Linear(84, 10)
        )

    def forward(self, x):
        x = self.cnn_model(x)
        x = x.view(x.size(0), -1)
        x = self.fc_model(x)
        return x

net = LeNet()
net.to(device)
```

 # Evaluate on the basis of accuracy - it calculates the accuracy by dividing the number of correct predictions by the total number of samples and returns the result as a percentage.

```
def evaluation(dataloader):
    total, correct = 0, 0
    for data in dataloader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        output = net(inputs)
        _, predicted = torch.max(output.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```

```
    return 100 * correct / total

# Fit the model and perform the training process
```
The fit function trains the LeNet model by iterating over multiple batches of data.
It uses the cross-entropy loss function and stochastic gradient descent (SGD) optimizer to update the model's parameters
 The function tracks the training loss for each iteration ,
 and also evaluates the model's accuracy on both the training and validation datasets after each iteration.
 The training process aims to minimize the loss and improve the model's accuracy over time.
Finally, the function plots the training loss per epoch.
```
def fit(max_epochs=16):
    loss_arr = []
    loss_epoch_arr = []

    loss_fn = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)

    for epoch in range(max_epochs):
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = net(inputs)
            loss = loss_fn(outputs, labels)
            loss.backward()
            optimizer.step()

            loss_arr.append(loss.item())
        loss_epoch_arr.append(loss.item())
        print('Val accuracy: %0.2f , Train accuracy: %0.2f' % (evaluation(testloader),
evaluation(trainloader)))

    plt.plot(loss_epoch_arr)
    plt.show()
fit()
```

The visualization helps to understand how the first layer of the LeNet architecture processes and extracts features from the input images.

```
# Visualize the behavior of images in the first layer of LeNet architecture
net = net.to('cpu')
out = net.cnn_model[0](images)
print('Image Shape:', out.shape)

image_id = 3
plt.figure(figsize=(6, 6))
plt.subplot(321)
for i in range(6):
    ax1 = plt.subplot(3, 2, i+1)
    plt.imshow(out[image_id, i, :, :].detach().numpy(), cmap='gray')
plt.show()
```

 # Function to load the prediction of our model

The predict_image function takes an image and a model as input and returns the predicted label for the image.

It preprocesses the image, passes it through the model, and retrieves the predicted label from the output.

The function provides a convenient way to obtain predictions for individual images using a trained model.

```
def predict_image(img, model):
    xb = img.unsqueeze(0)
    yb = model(xb)
    _, pred = torch.max(yb, dim=1)
    return pred[0].item()
```

```
# View the actual label and prediction of our model
plt.figure(figsize=(12, 12))
plt.subplot(321)
for i in range(25):
    ax1 = plt.subplot(5, 5, i+1)
    img, label = testset[i]
    plt.imshow(img[0], cmap='gray')
    if predict_image(img, net) == label:
        color = 'green'
    else:
        color = 'red'
```

```
    plt.xticks([])
    plt.yticks([])
    plt.tight_layout(pad=2)
    plt.title('Pred:{}, True:{}'.format(predict_image(img, net), label), color=color)
plt.show()
```

Full process ->

- The MNIST Handwritten Digit Recognition Problem
- Loading the MNIST Dataset in PyTorch
- Baseline Model with Multilayer Perceptrons
- Simple Convolutional Neural Network for MNIST
- LeNet5 for MNIST

The MNIST Handwritten Digit Recognition Problem

- Each image is a 28×28-pixel square (784 pixels total) in grayscale
- 60,000 images are used to train a model, and a separate set of 10,000 images are used to test it.
- To goal of this problem is to identify the digits on the image

Loading the MNIST Dataset in PyTorch

- There is a function in torchvision that can download the MNIST dataset

Baseline Model with Multilayer Perceptrons

- The training dataset is structured as a 3-dimensional array of instance, image height, and image width. For a multilayer perceptron model, you must reduce the images down into a vector of pixels. In this case, the 28×28-sized images will be 784 pixel input vectors. You can do this transform easily using the reshape() function.
- The pixel values are grayscale between 0 and 255. It is almost always a good idea to perform some scaling of input values when using neural network models. Because the scale is well known and well behaved, you can very quickly normalize the pixel values to the range 0 and 1 by dividing each value by the maximum of 255.
- You are going to use the cross entropy function to evaluate the model performance
- Cross-entropy loss, also known as log loss or softmax loss, is a commonly used loss function in PyTorch for training classification models. It measures the difference between the predicted class probabilities and the true class labels.
- The output of this model are logits, meaning they are real numbers which can be transformed into probability-like values using a softmax function.

- LeNet-5 CNN architecture is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers.
- A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.
- Sub-sampling is a technique that has been devised to reduce the reliance of precise positioning within feature maps that are produced by convolutional layers within a CNN.

LeNet CNN Structure

# Layer	Layer Type	Feature Maps	Kernel/Filter or Units	Input / Feature Map size	Trainable parameters	Connections	Strides	Activation Function
Input	Image	-	-	32x32	-	-	-	-
C1	Convolution	6	5x5	28x28	156	122,304		Hyperbolic tangent (tanh)
S2	Sub Sampling	6	2x2	14x14	12	5,880		Sigmoid
C3	Convolution	16	5x5	10x10	1,516	151,600		Hyperbolic tangent (tanh)
S4	Sub Sampling	16	2x2	5x5	32	2,000		Sigmoid
C5	Convolution	120	5x5	1x1	48,120			Hyperbolic tangent (tanh)
F6	Fully Connected	-	-	84	10,164			Hyperbolic tangent (tanh)
Output	Fully Connected	-	-	10				Softmax

- TensorFlow: An open-source platform for the implementation, training, and deployment of machine learning models.
- Keras: An open-source library used for the implementation of neural network architectures that run on both CPUs and GPUs.
- Training Dataset: This is the group of our dataset used to train the neural network directly. Training data refers to the dataset partition exposed to the neural network during training.
- Validation Dataset: This group of the dataset is utilized during training to assess the performance of the network at various iterations.
- Test Dataset: This partition of the dataset evaluates the performance of our network after the completion of the training phase.
- Activation Function: A mathematical operation that transforms the result or signals of neurons into a normalized output. An activation function is a component of a neural network that introduces non-linearity within the network. The inclusion of the activation function enables the neural network to have greater representational power and solve complex functions.