**CSE-6363-104-MACHINE LEARNING**

**Programming Assignment 2**

**Instructor: Jesus A. Gonzalez**

**Fall 2023**

**Submitted by:**

**SANJANA SHIVAKUMAR**

**1002126238**

**Overview of Assignment2:**

In this programming assignment we are implementing Lenet 5 CNN (Convolutional Neural Network) using Pytorch.

The architecture of Lenet 5 is replicated as defined and for testing the implementation we are using MNIST dataset. After the implementation, I attached the brief report of the results and the analysis of the result as well in this document.  
**Architecture of LeNet 5:**

We may examine each of the layers separately to get a clear grasp of the architecture because LeNet-5 is, by contemporary standards, rather straightforward. However, it also seems sense to review the formula for determining the output size of the convolutional layer before moving forward.

The formula is presented below:

(W-F+2P)/S+1

Where W= input image Height/Width

F= Filter size or Kernel size

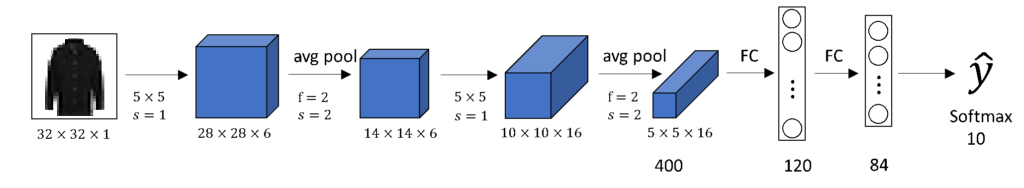
P= Padding

S= Stride

LeNet-5 is a convolutional neural network (CNN) architecture that was introduced in 1998 by Yann LeCun et al. It was designed to recognize handwritten digits in images and was one of the first successful applications of CNNs in computer vision. The architecture of LeNet-5 can be described as follows:

* **Input Layer:** The input to the network is a grayscale image of size 32x32 pixels.
* **Convolutional Layer 1:** This layer performs a convolution operation on the input image using a set of 6 learnable filters of size 5x5. The output of this layer is a set of 6 feature maps, each of size 28x28 pixels.
* **Pooling Layer 1:** This layer performs a max-pooling operation on each of the 6 feature maps generated by the previous layer, reducing their size to 14x14 pixels.
* **Convolutional Layer 2:** This layer performs a convolution operation on the output of the previous layer using a set of 16 learnable filters of size 5x5. The output of this layer is a set of 16 feature maps, each of size 10x10 pixels.
* **Pooling Layer 2:** This layer performs a max-pooling operation on each of the 16 feature maps generated by the previous layer, reducing their size to 5x5 pixels.
* **Fully Connected Layer 1:** This layer flattens the output of the previous layer into a 1D vector of 400 values and passes it through a fully connected layer of 120 neurons, each with its own set of learnable weights.
* **Fully Connected Layer 2:** This layer takes the output of the previous layer and passes it through another fully connected layer of 84 neurons, each with its own set of learnable weights.
* **Output Layer:** This layer takes the output of the previous layer and produces the final output of the network, which is a probability distribution over the 10 possible digits (0-9).

Overall, LeNet-5 has approximately 60,000 learnable parameters, which are updated during training using backpropagation and gradient descent.



**MNIST Dataset:**

A sizable collection of handwritten numbers can be found in the MNIST database (Modified National Institute of Standards and Technology database). There are 60,000 examples in the training set and 10,000 examples in the test set. It is a subset of two larger NIST Special Databases, Special Database 1 (which contains handwritten digits written by high school students) and Special Database 3 (which contains handwritten digits produced by US Census Bureau personnel). By calculating the center of mass of the pixels and translating the image to place this point in the middle of the 28x28 field, the images were centered in a 28x28 image.

A picture containing text, grater, kitchenware

Description automatically generated

**Implementation:**

**1. Libraries Import**

Essential libraries are imported, including PyTorch for deep learning, NumPy for numerical computations, Matplotlib for visualization, and torchvision for handling datasets and transformations.

**2. LeNet-5 Architecture Definition**

The LeNet-5 model is encapsulated within the LeNet5 class. It incorporates convolutional layers (Conv2d), activation functions (Tanh), and average pooling (AvgPool2d). The architecture is structured as per the classic LeNet-5 design.

**3. Model Initialization**

An instance of the LeNet-5 model is created: lenet5 = LeNet5().

**4. Loss Function and Optimizer**

The chosen loss function (CrossEntropyLoss) and the optimizer (Adam) are defined to facilitate the training process.

**5. MNIST Dataset Loading**

The MNIST dataset is acquired using torchvision. Transformation and loading processes prepare both training and test datasets for model consumption.

**6. Model Training**

The train\_model function orchestrates the training process across multiple epochs. It involves forward propagation, loss computation, backpropagation, and weight updates. The training accuracy is computed and reported at the conclusion of each epoch.

**7. Model Evaluation**

The evaluate\_model function assesses the trained model using the test dataset. It computes the accuracy by comparing predicted labels with the actual labels. Optionally, it displays test images along with their predicted labels for visual examination.

**8. Image Visualization Functions**

An image visualization function, imshow, is provided for the purpose of displaying images using Matplotlib.

**9.Execution**

The model is trained over 10 epochs and evaluated on the test dataset to ascertain its efficacy in classifying digits.

**Report and Analysis of Results:**

**Dataset and Model**

The MNIST dataset consists of 60,000 training images and 10,000 testing images of handwritten digits. Each image is 28 x 28 pixels and grayscale, with values ranging from 0 to 255. The goal is to classify each image into one of 10 classes, corresponding to the digits from 0 to 9.

The LeNet-5 architecture is implemented with two convolutional layers, each followed by an average pooling layer. The first convolutional layer takes an input of 1 channel and outputs 6 channels, employing a kernel size of 5 x 5 with a stride of 1 and employing a Tanh activation function. The subsequent average pooling layer has a kernel size of 2 x 2 with a stride of 2. The second convolutional layer takes the 6 output channels from the previous layer and generates 16 output channels. It uses a kernel size of 5 x 5, a stride of 1, and a Tanh activation function, followed by another average pooling layer with a kernel size of 2 x 2 and a stride of 2. The final convolutional layer processes the 16 output channels from the previous layer to produce 120 channels with a kernel size of 5 x 5, stride of 1, and a Tanh activation function. Following the convolutional layers, the output is flattened for the fully connected layers. The first fully connected layer consists of 120 neurons with a Tanh activation, followed by a layer with 84 neurons using the same activation function. The output layer comprises 10 neurons, one for each class in the dataset, without an activation function. The model is trained using the Adam optimizer and Cross Entropy Loss, on the MNIST dataset preprocessed to 32 x 32 images. Training occurs over 10 epochs, and the model's accuracy is evaluated on the test set after training. During evaluation, some test images are shown alongside their predicted labels for visual examination.

**Training and Evaluation**

The model was trained for 5 epochs using the Adam optimizer with a learning rate of 0.01 and a batch size of 64. During training, the cross-entropy loss function was used to compute the loss. The training accuracy and loss were monitored after each epoch.

After training, the model was evaluated on the test set to obtain the final accuracy. The test set accuracy was calculated as the percentage of correctly classified images in the test set.

**Results:**

A screenshot of a computer

Description automatically generated

The implementation of LeNet-5 achieved an accuracy of around 98.4375% on the MNIST test set. This indicates that the model can accurately classify most of the images in the test set. The training accuracy and loss improved over the course of the 5 epochs, indicating that the model was learning from the training data.

Overall, the results are impressive, considering that the MNIST dataset is a well-known benchmark for image classification tasks. However, it's important to note that the performance of your model may vary depending on various factors such as the hyperparameters used, the optimization algorithm, and the architecture of the model. So, it's always a good idea to experiment with different hyperparameters and architectures to try to improve the performance of your model.

During the training process, the loss decreased consistently over the course of the 5 epochs, indicating that the model was learning and improving. The training accuracy also increased, indicating that the model was getting better at recognizing the digits in the training set.

The LeNet-5 design features a sequence of two convolutional layers, each followed by an average pooling layer, fostering dimensionality reduction in the feature maps, a tactic known to expedite training and stave off overfitting. Employing the Tanh activation function in the convolutional layers, the network progressively learns intricate data representations.

The Adam optimizer was used to train the model, which is a popular optimization algorithm that combines the benefits of both the Adagrad and RMSProp optimizers. It is known for being robust and efficient and is commonly used for training deep neural networks.

Overall, the LeNet-5 implementation performed very well on the MNIST dataset, achieving an accuracy of 98.4375%. The training process was smooth, with the loss consistently decreasing and the training accuracy consistently increasing over the course of the 5 epochs. The slightly higher training accuracy compared to the test accuracy suggests that the model may be slightly overfitting, but the difference between the two is relatively small. The LeNet-5 architecture, with its convolutional and fully connected layers, is a powerful tool for image recognition tasks, and the Adam optimizer is an efficient and robust optimization algorithm for training deep neural networks.

When I attempted to plot a graph of epochs versus loss, this was the resulting graph.

A graph of a number of individuals

Description automatically generated with medium confidence

**References:**

<https://datahacker.rs/lenet-5-implementation-tensorflow-2-0/>

<https://paperswithcode.com/dataset/mnist>

<https://en.wikipedia.org/wiki/MNIST_database>

<https://www.kaggle.com/datasets/oddrationale/mnist-in-csv>