2020 Pattern Recognition and Machine Learning Technical Report

Ruian He, 16307110216,

I. TASK DESCRIPTION

DESIGN a proper convolutional neural network for the classification task based on the MINIST dataset.[1]

II. DATA DESCRIPTION

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. [2]

The data is stored in a very simple file format designed for storing vectors and multidimensional matrices.

A. Label File

The label file is like the following.The labels values are 0 to 9.

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
...	...	...	...
xxxx	unsigned byte	??	label

B. Image File

And the image file is as followed.Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
...	...	...	...
xxxx	unsigned byte	??	pixel

III. DATA PREPROCESSING

First of all, we uncompress the `.gz` files to get ubyte files described as above. And we use struct module in python to extract the magic number, the image number, the row number and col number. Then we read $num * row * col$ numbers stored in unsigned bytes from the file which is the data. We can visualize using `matpoltlib` module.

We also can plot the average image for every digit, and get a general view over all training data.

[1]http://yann.lecun.com/exdb/mnist/.

Fig. 1. Sample Digit

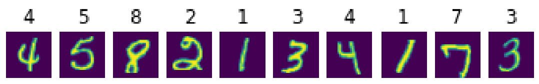
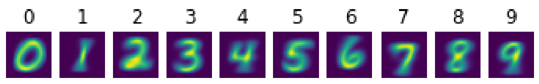


Fig. 2. Average Digit



IV. ALGORITHM INTRODUCTION

A. Algorithm principle

Convolutional networks, also known as convolutional neural networks, or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology. [1]

In its most general form, convolution is an operation on two functions of a real-valued argument. One is the input $x(t)$, and the other is the weight $w(a)$. If we apply such a weighted average operation at every position of x, we obtain a new function s providing a smoothed estimate of the x.

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

This operation is called convolution. In convolutional network terminology, w is referred as the kernel and s as the feature map.

Convolution leverages three important ideas that can help improve a machine learning system:sparse interactions, parameter sharing and equivariant representations.

Unlike dense connected layers in Multi-layer Perceptron Neural Networks, the convolutional layers only have few parameters shared by all output nodes, and one input only interacts with near inputs. That greatly reduced the amount of parameters and introduce the prior of local correlation.

A typical layer of a convolutional network consists of three stages .In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations.

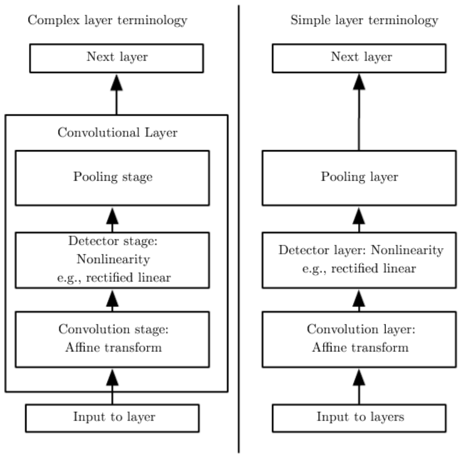
$$\text{out} = \text{bias} + \sum \text{weight} * \text{input}$$

In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the detector stage.

$$\text{ReLU}(x) = (x)^+ = \max(0, x)$$

In the third stage, we use a pooling function to modify the output of the layer further.

Fig. 3. Convolutional Layer


Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.For example, the max pooling operation reports the maximum output within a rectangular neighborhood.

In all cases, pooling helps to make the representation approximately invariant to small translations of the input. It can vary the layers' size and is proved to be very useful when attracting features.

B. Algorithm implementation

We applied the deep learning library `pytorch` to implement this algorithm. We chose the LeNet network mentioned in [1]. And it was proven to be effective on the MNIST dataset.

LeNet is composed of 2 convolutional layers and 2 fully-connected layers. The output then goes through Softmax function and the loss is cross-entropy loss.

Listing 1. LeNet
```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 3,
            1)
        self.conv2 = nn.Conv2d(6, 16, 3,
            1)
        self.fc1 = nn.Linear(400, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = torch.tanh(x)
        x = F.avg_pool2d(x, 2)
        x = self.conv2(x)
        x = torch.tanh(x)
        x = F.avg_pool2d(x, 2)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

V. EXPERIMENTAL RESULTS AND ANALYSIS

We train the model for 10 epochs optimized with Adadelta and learning rate 0.5. The accuracy and loss value thoughout training is at below. We finally achieved 98.3% accuracy on the test set.

Fig. 4. Accuracy on validation set

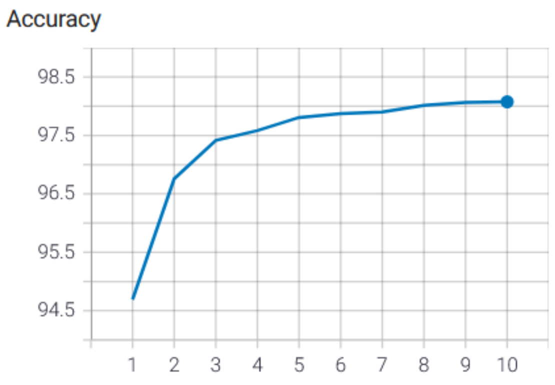
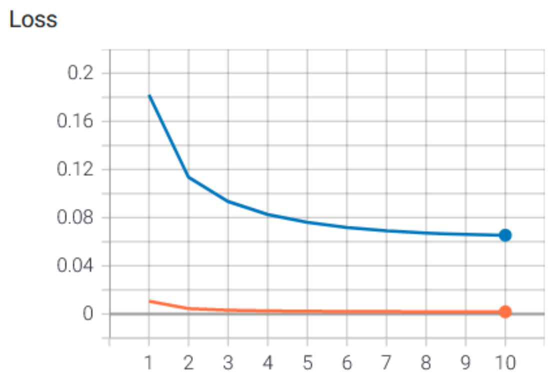


Fig. 5. Loss on training set and validation set



CNN show its impressive ability to classify the image. With only 1 epoch, the model achieved 95% accuracy on validation set. It outperforms other linear model like SVM and Perceptron with speed and result.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.