A simple CNN model on Fashion MNIST dataset

Tam King Man 1155160072 Lai Cheuk Lam 1155159309

1. Background

Our project aims to provides basic deep learning background for readers to gain a brief overview of deep neural network. Topic includes the neural network architecture, Multilayer Perceptron, backpropagation, activation functions, CNN architecture, convolution, pooling, flattening, dropout. A working example for image classification on Fashion MNIST dataset using CNN is also illustrated in the end to provide an insight on how neural networks really works.

1. Deep Neural Network

To simulate how the brain works, the idea of neural network is proposed. In a neural network, we have input layer (the first layer), multiple hidden layers and output layer (the last layer). Each layer is composed with some neurons. The neurons in the layer may be connect to the neurons in the layer by some weighted edges. When the number of hidden layer increases, human may be difficult to follow what happens among the hidden layers and this is how the term “deep” comes from.

The most basic class of deep neural network is Multilayer Perceptron (MLP), it is a fully connected acyclic feedforward neural network. Feedforward means that the information moves in one direction through the hidden layer. Such a network might be drawn as follows:

Diagram

Description automatically generated

We can formulate it in mathematical way:

Let be the input, be the weights in layer , be the activation function in layer . Denote the number of layers (except input layer) by L, the number of neurons in layer by . Note that and for . Hence, we have

.

By this network, we may mimic the behavior of our brain. We only have one problem; how can we find the weights among each layer? We want to minimize the cost function in weight space by gradient descent. To do gradient descent efficiently, we have backpropagation algorithm, which is an efficient use of Chain rule for derivatives.

1. Backpropagation

Backpropagation algorithm:

1. Initialize network with random weight
2. For all training samples:
   1. Input sample and compute the output (forward pass)
   2. Compare the output with correct output to get loss term
   3. For all layers (backward pass from output layer)
      * Propagate the loss term back to the previous layer
      * Update the weights between the two layers

Diagram

Description automatically generated

Let and be the value of the neurons in the layer and the associated weight from neurons in the layer to the neuron in the layer respectively. Simply put, . Also, let be the output of row in layer . Then .

We first start looking at the output layer. By Chain rule, we have

Note that we know and in the forward pass and we can compute the derivative of cost function and activation function directly. We know what is .

For the hidden layers,

Hence, we only need , , and . Finally, we can update the weight by where is the learning rate.

The concept of forward pass and backward pass play an important role in neural network. We just summary parts of the main ideas and we will go to the next chapter to discuss CNN.

1. Activation function

When the network passes the features to the next layer, we need some non-linear activation functions to transform the features. Otherwise,

where are linear transformation. In other words, we can view multiple linear transformations as one linear transformation. We hereby discuss some common activation functions, which are non-linear.

* Sigmoid function

Chart

Description automatically generated

First, we have sigmoid function, which is very common, appear in the logistic regression. However, there are some drawbacks for using sigmoid function as an activation function. The computation of the exponential term is time consuming. The output is not zero-centered, and the network may converge slowly so we need unexcepted time for training. But the main problem is that the gradient is vanishing. The convergence speed here means the number of iterations. We can see from the graph of the derivative of sigmoid function that . It follows that the gradient tends to quickly when we update weightings by gradient descent. Hence, it is not suggested to use sigmoid function as the activation function.

* Tanh function

Chart

Description automatically generated

Secondly, we can see from the graph of that the output is zero-centered. However, the heavy computation cost of exponential term and gradient vanishing problem still exist. Hence, ReLU function is proposed to solve these problems.

* ReLU function

Chart

Description automatically generated

Reader may notice that the derivative of ReLU is not defined. This problem can be solved easily by some convention. i.e., when . ReLU converges quickly and the computation cost is since the network only need to determine the threshold. Also, it solves the gradient vanishing problem for . Therefore, we usually use ReLU function as the activation function. Specifically, note that ReLU is not zero-centered and sometimes it may cause some dead neurons due to bad weight initialization or high learning rate. People still use it due to its low computation cost. Some functions are proposed based on regular ReLU. For example, we have Leaky ReLU and ELU (Exponential Linear Units).

Leaky ReLU:

ELU:

We may skip the details of them and go to CNN. In the working example, we also mainly use ReLU as activation function for the neural network.

1. From NN to Convolutional NN

In deep learning, researcher builds a lot of extension on top of the neural network (NN) and develop powerful frameworks, such as Convolutional neural network (CNN). We may discuss it in this chapter. There are many image recognition models are based on the framework of CNN, which is a good starting point to learn different deep learning frameworks. A CNN arranges its neurons in three dimensions, and hence we may obtain a 3D output tensor. It is composed with three components, convolution layer, pooling and fully connected layer.

For better understanding, we use an example here.[[1]](#footnote-1) For convolution layer, we have a filter and we convolve the filter with a dimension input. The filter slide over the image spatially and compute the dot products. Then we have an activation map with dimension . Since we may have many filters, we may stack the activation maps together (let’s say we have 6 filters) and hence we obtain a tensor as the following graph.

Diagram

Description automatically generatedDiagram, table

Description automatically generated

To make the layer smaller and more manageable (downsample), Pooling layer can be applied. There are two common implementations of pooling. In max-pooling, the representative value just becomes the largest of all the units in the window, while in average-pooling, the representative value is the average of all the units in the window. Note the volume depth is preserved.

Neurons in a fully connected layer have full connections to all activations in the previous layer, as the regular NN. Their activations can hence be computed with a matrix multiplication followed by a bias offset. It is remined that we should flatten the layer into a feature vector before putting it into the neural network. The following graph is an example of the illustrate convolutional neural network. There are also many examples of CNN, e.g. AlexNet, ResNet, GoogLeNet, etc.

Diagram

Description automatically generated

1. Dropout

Dropout was proposed by Hinton in 2019 and became famous by the influence of AlexNet. The idea of dropout is simple, prevent neurons from co-adapting too much. In each batch, some neurons may stop forward pass (can be simply done by setting their value as 0) with a given probability . This may reduce that some local properties dominate the network and make the network more generalized. In another word, this method can prevent overfitting.

Diagram, engineering drawing

Description automatically generated

The above figure may illustrate how dropout affect the network. We can see figure become more simplified. Note that the crossed neurons are just removed temporarily. After forward pass, the network will continue do backpropagation on the simplified network. Then we update the uncrossed neurons, recover the crossed neurons and repeat the process. A more formal explanation and implementation may leave to readers due to page limit.

1. Working Example[[2]](#footnote-2)

We try to implement CNN model using Fashion MNIST dataset. Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

|  |  |
| --- | --- |
| Class 0 | T-shirt/Top |
| Class 1 | Trouser |
| Class 2 | Pullover |
| Class 3 | Dress |
| Class 4 | Coat |
| Class 5 | Sandal |
| Class 6 | Shirt |
| Class 7 | Sneaker |
| Class 8 | Bag |
| Class 9 | Ankle Boot |

We may first illustrate how the dataset looks like.

A picture containing diagram

Description automatically generated

Now, we introduce our proposed CNN model for classification and discuss some problems we met during the experiments. We set our batch size as 256, epochs as 40 and learning rate as 0.05. After experiments with different setting, the batch size doesn’t affect the result a lot. We set epochs as 40 because of the long training time. The network can already achieve amazing result with 40 epochs. The most interesting part here is the learning rate. With different learning rate, our result may differ a lot and converge to eventually.

def \_\_init\_\_(self):

super(ConvNet, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(in\_channels=1, out\_channels=32, kernel\_size=3)

self.pool1 = nn.MaxPool2d(kernel\_size=2, stride=2)

self.conv2 = nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3)

self.pool2 = nn.MaxPool2d(kernel\_size=2, stride=2)

self.conv3 = nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3)

self.dropout = nn.Dropout(p=0.4)

# fully connected layer

self.fc1 = nn.Linear(3\*3\*128, 128)

self.fc2 = nn.Linear(128, 10)

def forward(self, x):

# first conv

x = self.pool1(F.relu(self.conv1(x)))

x = self.pool2(F.relu(self.conv2(x)))

x = F.relu(self.conv3(x))

x = self.dropout(x)

# flatten all dimensions except batch

x = torch.flatten(x, 1)

# fully connected layers

x = F.relu(self.fc1(x))

x = self.dropout(x)

x = F.softmax(self.fc2(x))

return x

Note that our input image is of size . In the first convolution layer, we use 32 filters with ReLU function. Then we get a tensor. We apply max-pooling with filters and stride . Note the tensor is of size and the volume depth is unaffected.

In the second convolution layer, we use filters with ReLU function. Then we get a tensor. We apply max-pooling with filters and stride . Note the tensor is of size .

We do the third convolution layer again, we use filters with ReLU function. Then we get a tensor. We use dropout to prevent overfitting with . After flattening the tensor, we put it into the fully connected network. The first fully connected layer use ReLU function as the activation function. Finally we use softmax function to output the result.

loss\_fn = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(), lr=0.05)

For the loss function, we choose Cross Entropy Loss function and use Stochastic gradient descent to update the parameter.

Text

Description automatically generated

We achieve testing accuracy with just 40 epochs. The result is quite impressive and how can’t we believe the power of deep learning? We can see the loss function is keep decreasing. Although there is a ‘angle’ around the epoch, it doesn’t affect the final result a lot and we can see that the loss function tends to stably after about 15 epochs.

Chart, line chart

Description automatically generated

For full code, readers may refer to

<https://github.com/marcotam2002/MATH3320-miniproject>

1. More mathematical detail refers to the lecture note. The formal definition is omitted due to page limits. [↑](#footnote-ref-1)
2. We have a reference on the dataset accessed on <https://www.kaggle.com/code/pavansanagapati/a-simple-cnn-model-beginner-guide>. Our model uses Pytorch instead of TensorFlow. [↑](#footnote-ref-2)