Project1 : MLP, CNN and SNN

Yang Wang
Center for Speech and Language Technologies
Tsinghua University
wangyang@cslt.riit.tsinghua.edu.cn

1. Introduction

Recently, ANN has been a very hot research area in computer science. Owning to it strong learning ability, it achieved the state-of-the-art performance in many tasks. Furthermore, NN with kinds of structure are designed to deal with the specified problems. In this paper, we implement three neural networks: MLP, CNN and SNN. We also do some experiments to investigate the impact of activation function, tricks in CNN and so on.

2. Methods

In this section, I introduce how I implement the three neural networks: MLP, CNN and SNN. I choose tensorflow, an open source software library for deep learning to implement them. It can provides API for Python and C++. Moreover, it has a large community and extensive documentations so that it is easy to use. The official documents is enough to learn the implementation of MLP and CNN. When I implement SNN I refer the code [1] provided in [1].

2.1. MLP

The first task is to implement a multilayer perceptron (MLP). The basic components of a MLP are a linear function and a an activation function. They have the following form:

$$y = w \times x + b$$

$$z = fun_{activation}(y)$$

Where the $fun_{activation}$ can be sigmoid function, Relu function and so on. In our experiments we also compare the results of them.

When we stack several layers to form a deep structure, the model can be stronger to learn high-level features which helps classification. In our experiments, we design a 4 layer network with 784-64-128-10 neurons. Due to that it is a multi-label classification task, we use softmax function to compute the probability for each class. The formulation of softmax is as following:

$$y_i = softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

We have the true distribution y' and the predicted distribution y. Then we need to define what it means for the model to be good or bad. In a professional word, we need to define the loss function. The smaller the value of the loss function is, the better our model is. One very common, very nice function to determine the loss of a model is cross-entropy. Cross-entropy arises from thinking about information compressing codes in information theory but it winds up being an important idea in lots of areas, from gambling to machine learning. It's defined as:

$$H_{y'}y = -\sum_i {y_i}' log(y_i)$$

In Tensorflow, it is easy to do training because it knows the entire graph of our computations and automatic differentiation is achieved. Several optimization method is provided such as gradient descent algorithm and Adam algorithm. In our experiments, we use the simple gradient descent algorithm to minimize the cost. We use mini-batch to train our model. In each training step, we sample a mini-batch from the training dataset and feed them to the placeholders.

2.2. CNN

The second task is to implement a convolution neural network (CNN). CNN has local receptive field and shares parameters between each local area, which make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. Tensorflow gives a lot of flexibility in convolution and pooling operations. It is easy to handle the boundaries and set stride size by passing parameters to the build-in function conv2d. For example, our first convolution layer uses 8 filters with the size of 5×5. The stride is set to 1 and the images are zero padded so that the output is the same size as the input. Our

[1] https://github.com/dannyneil/spiking_relu_conversion

Layer	Size
Input	784
conv1	numfilter, 8, size, [5,5], stride, [1,1]
poo1	size, [2,2], stride, [1,1]
conv2	numfilter, 16, size, [5,5], stride, [2,2]
poo2	size, [2,2], stride, [1,1]
conv3	numfilter, 32, size, [4,4], stride, [2,2]
fcn1	200
fcn2	10

Table 1. Structure of the CNN

pooling is plain old max pooling over 2x2 blocks. The Tensorflow code is:

```
conv1_weights = tf.Variable(
  tf.truncated_normal([5,5,1,8],
  stddev=0.1,seed=SEED,
  dtype=data_type()))

conv = tf.nn.conv2d(data, conv1_weights,
   strides=[1,1,1,1], padding='SAME')

relu = tf.nn.relu(
   tf.nn.bias_add(conv, conv1_biases))

pool = tf.nn.max_pool(relu,
   ksize=[1,2,2,1],strides=[1,2,2,1],
   padding='SAME')
```

The structure of our model is shown in Table 1 We still use the softmax regression to compute the probability distribution and cross-entropy as loss function. To reduce overfitting, we apply dropout before the readout layer. We create a placeholder for the probability that a neuron's output is kept during dropout. This allows us to turn dropout on during training, and turn it off during testing. The tensorflow code is as flollowing:

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1,
                           keep_prob)
```

We also use batch normalization to make training easier. The batch_size is to 128 and we evaluate its performance after 30 epochs.

2.3. SNN

Deep neural network such as DBM and CNN has achieved state-of-the-art performance in many tasks. SNN is proposed recently to overcome the large computation cost by using the specialized hardware. Although it brings some loss on performance, it is promising to be used to simplify our device. In our experiments, the spiking neuron model used for this work is the simple integrate-and-fire (IF) model. The evolution of the membrane voltage v_{mem} is given by:

$$\frac{dv_{mem}(t)}{dt} = \sum_i \sum_{s \in S_i} w_i \delta(t-s)$$

where w_i is the weight of the ith incoming synapse, $\delta()$ is the delta function, and $Si = \{t_i^0, t_i^1 ...\}$ contains the spike-times of the ith presynaptic neuron. If the membrane voltage reach the spiking threshold v_{thr}, a spike is generated and the membrane voltage is reset to a reset potential v_{res}. In our simulations, this continuous-time description of the IF model is discretized into 1 ms timesteps.

Following the [1], we use Relu function as activation function for all the layers and use zero bias in training. First, we train a 4 layer MLP with 784-64-128-10 neurons . Then we convert it to a SNN according to the firing rule. The algorithm is given in Algorithm 1, where N and D are number of examples and dimension of a image. Spikes, mem and sum_spikes are matrixes to store the spikes, membrane voltage and sum_spikes respectively. W is the weight of MLP. Finally, the values of sum_spikes[end] represent the evidence of the example being in certain classes.

Algorithm 1 : SNN

```
for dt in dt:dt:duration do
  $rescale\_fac = 1/(dt * max\_rate)$
  $spike\_snap = rand(N, D) * rescale\_fac$
  $inp\_image = spike\_snap <= test\_x$
  $spikes[0] = inp\_image$
  $sum\_spike[0] = sum\_spikes[0] + inp\_image$
  for l in 1:len(layer)-1 do
    $impulse = spikes[l-1] * W[l-1]$
    $mem[l] = mem[l] + impluse$
    $spikes[l] = mem[l] >= v_{thr}$
    $mem[l][spikes[l]] = v_{res}$
    $sum\_spike[l] = sum\_spikes[l] + spikes[l]$
  end for
end for
$y = argmax(sum\_spikes[end])$
We set the dt to 0.001s and and simulated time is 0.07s.
```

3. Results

3.1. MLP

In the experiments of MLP, we compare the results between sigmoid function and Relu function. The results is shown in Table. 2 We can see that Relu activation function achieve slightly better performance than sigmoid function.

3.2. CNN

In the experiments of CNN, we investigate the impact of dropout and batch normalization. The results is shown in

Activation Function	Acc Rate (%)
Sigmoid	97.75
Relu	97.77

Table 2. Results of experiments of MLP

Trick	Acc Rate (%)
None	98.36
Dropout	98.51
Batch normalization	99.36
Dropout & Batch normalization	99.39

Table 3. Results of experiments of CNN

Network	Acc Rate (%)
MLP	97.82
SNN	96.72

Table 4. Results of experiments of SNN

Table. 3. From the results we can seen that both the batch normalization and dropout are helpful, especially the batch normalization. It improves the accuracy rate from 98.36% to 99.36%. Dropout is often very effective at reducing overfitting, but it is most useful when training very large neural networks. So in our experiment the effect is not significant.

Figure 1 show the images that our model fail to predict correctly.

3.3. SNN

In the experiments of SNN, we compare the results of MLP and SNN. The results are shown in Table 4. After 0.07s simulated time it can achieve 96.72% accuracy rate.

4. Discussion

In this paper, we just do some preliminary experiments. There is still many problem to be studied, for examples, how to speed up the SNN and what spiking architectures is better and why they are better. More experiments need to be done to reveal the secret. However, we obtain some useful experience through this project, such as the effect of dropout and batch normalization. This is helpful in our further research and work.

References

[1] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

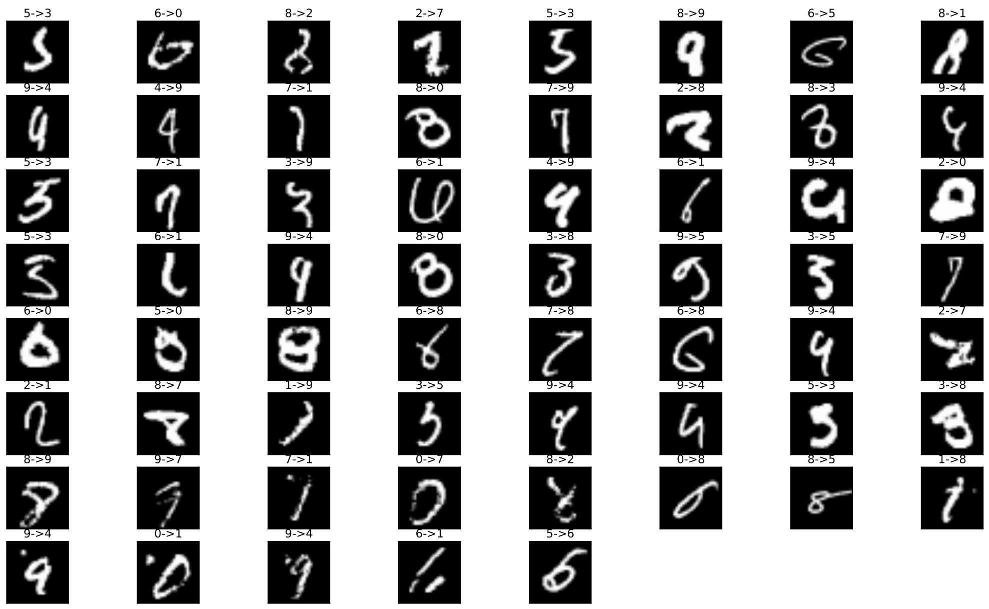

Figure 1. Mis-classified images by CNN