# Week report
# 2018,oct,15-20
# This Week

**linsuxin28@gmail.com**

**UESTC**

Suxin(Sam)LIN

| **15** | **16** | **17** | **18** | **19** | **20** |
|---|---|---|---|---|---|
| | | | | paper[4] | with medic? |
| **paper[1]** | torch tutorial 【6】 | experiment [1] | Backpro of neural network | paper[4] | with medic? |
| **paper[2]** | torch tutorial 【6】 | experiment [1] | experiment [2] | paper[5] | with medic? |
| **read code [3]** | experiment[1] | experiment [2] | experiment [2] | paper[5] | with 3d classification? |
| **metrial[4]** | experiment[1] | experiment [2] | experiment [2] | paper review | with 3d classification? |
| **metrial[5]** | metrial[4] | experiment [2] | | experiment [2] | with 3d classification? |

| task | status | issue | supply |
|---|---|---|---|
| 1paper[1]imagenet classificarion | finished | | note in github |
| 2 paper[2] Binarized Neural Networ | finished | QA in the note | note in github |
| 3 Xnor pytorch imple (no cuda) | finished | | result in report |
| 4 pytorch tutorial[6] | finished | | |
| 5 Analyze the function of util.py: | finished | | in report |
| 6 paper[4] Overcoming Challenges i | half | | detail uncleared |

| | | | |
|---|---|---|---|
| 7 paper[5] Quantizing DNN | half | | detail uncleared |
| 8 Xnor pytorch imple ( cuda) | finished | | learn curv report |
| 9 backpropagation | half | quan back agrithom not clear in[2] | |

## NEXT WEEK

| task | status | issue | supply |
|---|---|---|---|
| 1 Summarized optimization problems and methods | | | |
| 2 Read code of HWGQ | | | |
| 3 paper[7] | | | |
| 4 paper[8] | | | |
| 5 paper[9] | | | |
| 6 Reproduce HWGQ | | | |
| 7 Write HWGQ layer in HWGQ with Cuda. | | | |
| 8 review and summary previous work | | | |

# Reading meterial(list):


[1]: XNORT_NET    imagenet classification using Binary CNN

[2]: Binarized Neural Network: Training Neural Networks with Weights and Activations Constrained to -1 or 1

[3] Xnor pytorch implementation

[4] Overcoming Challenges in Fixed Point Training of Deep Convolutional Networks

[5] Quantizing deep convolutional networks for efficient inference: A whitepaper

[6] pytorch tutorial

[7] DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients [Megvii, Face ++]

[8] Training Quantized Nets: A Deeper Understanding

[9] Deep Learning with Low Precision by Half-wave Gaussian Quantization

[10] Training and Inference with Integers in Deep Neural Networks

[11] Summarized optimization problems and methods for quantization neural networks

# Reading meterial summary:

[1]: XNORT_NET  imagenet classification using Binary CNN

note link: https://github.com/XinDongol/reading_list/blob/master/Suxin_Sam_LIN/paper note/1.pdf

[2]: Binarized Neural Network: Training Neural Networks with Weights and Activations Constrained to -1 or 1

note link:

https://github.com/XinDongol/reading_list/blob/master/Suxin_Sam_LIN/paper note/2.pdf

# Experiment:

## [1] Xnor net torch debug and test on CIFAR 10

- Test result

| epoch | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| loss | 1.7562 | 1.2832 | 1.1674 | 1.1090 | 0.9439 | 0.9985 | 0.9517 | 0.8395 | 0.8391 |
| test_acc | 44.96 | 57.87 | 59.89 | 60.90 | 67.38 | 66.5 | 66.66 | 70.94 | 70.77 |
| parameter | LR:0.01 | eps=1e-4 | momentum =0.1 | | | | | | |
| epoch | 15 | 30 | 50 | 90 | 190 | 290 | 320 | | |
| loss | 0.9179 | 0.7331 | 0.7078 | 0.6607 | 0.4664 | 0.4890 | 0.5009 | | |
| test_acc | 71.83 | 76.83 | 77.23 | 78.48 | 84.83 | 84.76 | 84.00 | | |

- Analyze the function of util.py:

**main.py** used the util.py in these place and make the coming effort:

main.py:

line 32_ def train(epoch):

line 35#process the weights including binarization

line 36 bin_op.binarization()---[util.py]

line 38#forwarding

line 43#backwarding

line 47#restore weights

line 48 bin_op.restore()---[util.py]

line 49 bin_op.updateBinaryGradWeight()

2 def test();

line 64 bin_op.binarization()---[util.py]

line 74 bin_op.restore()---[util.py]

util.py:

line 4 class Binop():

line 5 def init():

num_of_parameter get

saved_params get

traget_params not get

target_modules get

line 30 def binarization(self):

line 31 self.meancenterConvParams() get mean

line 32 self.clampConvParams()

line 33 self.save_params()

line 34 self.binarizeConParams()

line 36 def meancenterConvParams(self):

#get the -mean of parameter and expand it to original dimension

line 43 def clampConvParams(self):

clamp(-1,1)

**`torch.clamp`**(*input, min, max, out=None*) → Tensor

Clamp all elements in `input` into the range *[min, max]* and return a resulting Tensor.

```
      | min, if x_i < min
y_i = | x_i, if min <= x_i <= max
      | max, if x_i > max
```

line 48 def save_paras(self):

# save the binarized parameter

line 52 binarizeConvParams(self);

**`torch.div()`**

**`torch.div`**(*input, value, out=None*) → Tensor

Divides each element of the input `input` with the scalar `value` and returns a new resulting tensor.

$$out_i = \frac{input_i}{value}$$

If `input` is of type *FloatTensor* or *DoubleTensor*, `value` should be a real number, otherwise it should be an integer

**Parameters:**

- **input** (*Tensor*) – the input tensor
- **value** (*Number*) – the number to be divided to each element of `input`
- **out** (*Tensor, optional*) – the output tensor

**`torch.sign`**(*input, out=None*) → Tensor

Returns a new tensor with the sign of the elements of `input`.

**Parameters:**

- **input** (*Tensor*) – the input tensor
- **out** (*Tensor, optional*) – the output tensor

Example:

```
>>> a = torch.randn(4)
>>> a
tensor([ 1.0382, -1.4526, -0.9709,  0.4542])
>>> torch.sign(a)
tensor([ 1., -1., -1.,  1.])
```

`torch.normal()`

`torch.normal(mean, std, out=None) → Tensor`

Returns a tensor of random numbers drawn from separate normal distributions whose mean and standard deviation are given.

The `mean` is a tensor with the mean of each output element's normal distribution

The `std` is a tensor with the standard deviation of each output element's normal distribution

The shapes of `mean` and `std` don't need to match, but the total number of elements in each tensor need to be the same.

**Note**

When the shapes do not match, the shape of `mean` is used as the shape for the returned output tensor

**Parameters:**
- **mean** (*Tensor*) – the tensor of per-element means
- **std** (*Tensor*) – the tensor of per-element standard deviations
- **out** (*Tensor, optional*) – the output tensor

Example:

```
>>> torch.normal(mean=torch.arange(1., 11.), std=torch.arange(1, 0, -0.1))
tensor([  1.0425,   3.5672,   2.7969,   4.2925,   4.7229,   6.2134,
          8.0505,   8.1408,   9.0563,  10.0566])
```

n = data[0].nelement()

s = data.size

m = nomal(1,3).sum(2).sum(1).div(n)

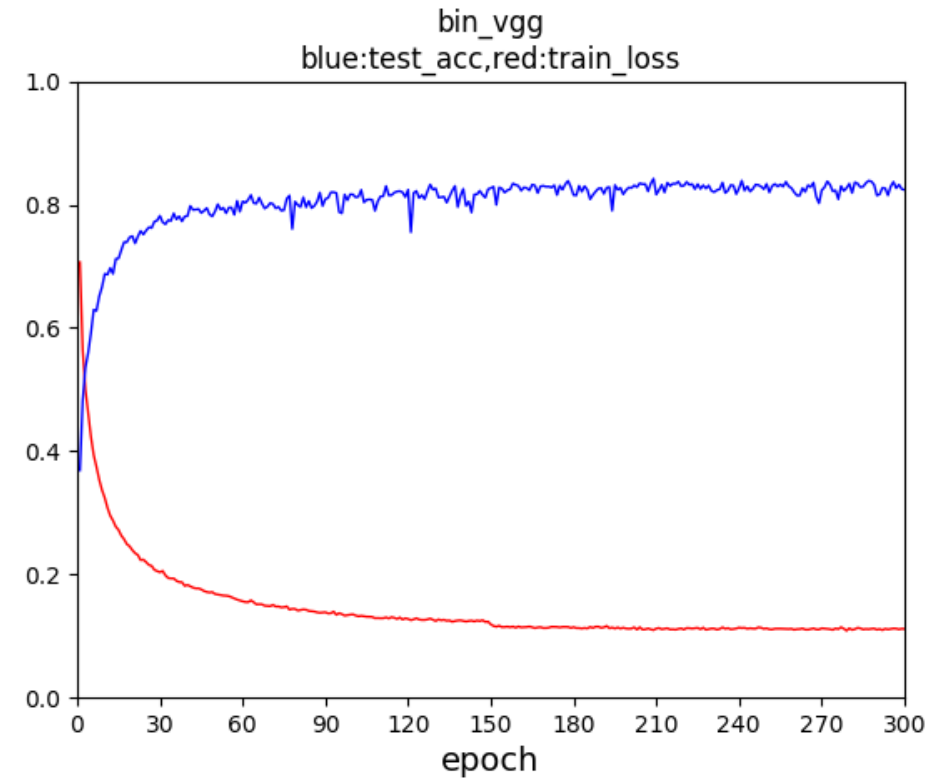
target.data = data.sign().mul(m.expends)

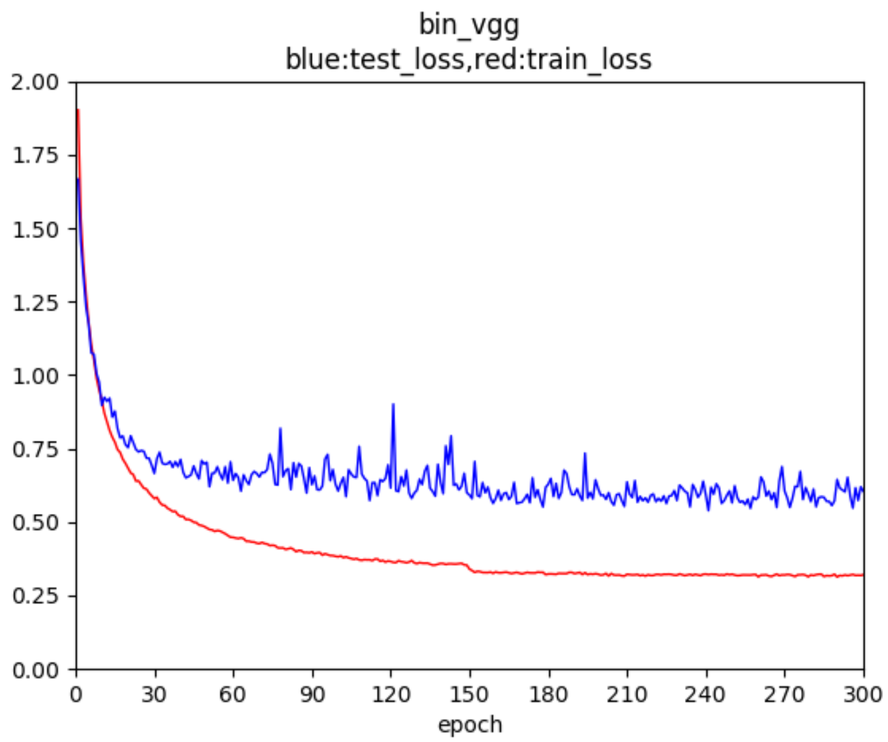
line 65 updateBinaryGradWeight(self):

#up date the gra with normalization and binarization.

# [2] Xnor net torch(with cuda in c++) debug and test on CIFAR 10

- Test result


bin_vgg
blue:test_acc,red:train_loss
1.0
0.8
0.6
0.4
0.2
0.0
0
30
60
90
120
150
180
210
240
270
300
epoch


bin_vgg
blue:test_loss,red:train_loss
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
30
60
90
120
150
180
210
240
270
300
epoch

**QA:**