

Week 2
Kexin Fan

- Codes of XNOR-Net/util.py
Understand and mark the codes.

```
import torch.nn as nn
import numpy

class BinOp():
    def __init__(self, model):
        # count the number of Conv2d
        count_Conv2d = 0
        for m in model.modules():
            if isinstance(m, nn.Conv2d):
                count_Conv2d = count_Conv2d + 1
        start_range = 1
        end_range = count_Conv2d-2
        self.bin_range = numpy.linspace(start_range,
                                         end_range, end_range-start_range+1)\
                                         .astype('int').tolist() #define the range
        self.num_of_params = len(self.bin_range)
        self.saved_params = []
        self.target_params = []
        self.target_modules = []
        index = -1
        for m in model.modules():
            if isinstance(m, nn.Conv2d):
                index = index + 1 # count from 0
                if index in self.bin_range:
                    tmp = m.weight.data.clone()
                    self.saved_params.append(tmp)
                    self.target_modules.append(m.weight)

    def binarization(self):
        self.meancenterConvParams() #cut the mean
        self.clampConvParams()
        self.save_params() #backup
        self.binarizeConvParams()

    def meancenterConvParams(self):
        for index in range(self.num_of_params):
            s = self.target_modules[index].data.size()
            negMean = self.target_modules[index].data.mean(1, keepdim=True).\
                mul(-1).expand_as(self.target_modules[index].data)
            self.target_modules[index].data = self.target_modules[index].data.add(negMean)
```

```

def clampConvParams(self):
    for index in range(self.num_of_params):
        self.target_modules[index].data = \
            self.target_modules[index].data.clamp(-1.0, 1.0) #set the range of weights

def save_params(self):
    for index in range(self.num_of_params):
        self.saved_params[index].copy_(self.target_modules[index].data) #backup the parameters

def binarizeConvParams(self):
    for index in range(self.num_of_params):
        n = self.target_modules[index].data[0].nelement()
        s = self.target_modules[index].data.size()
        m = self.target_modules[index].data.norm(1, 3, keepdim=True)\
            .sum(2, keepdim=True).sum(1, keepdim=True).div(n) # calculating the mean value of
each part of the weights
        self.target_modules[index].data = \
            self.target_modules[index].data.sign().mul(m.expand(s)) #binarize process

def restore(self):
    for index in range(self.num_of_params):
        self.target_modules[index].data.copy_(self.saved_params[index])

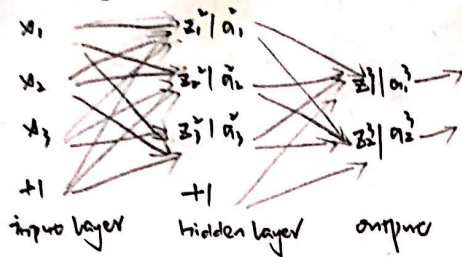
def updateBinaryGradWeight(self):
    for index in range(self.num_of_params):
        weight = self.target_modules[index].data
        n = weight[0].nelement()
        s = weight.size()
        m = weight.norm(1, 3, keepdim=True)\
            .sum(2, keepdim=True).sum(1, keepdim=True).div(n).expand(s) #calculate the mean
        m[weight.lt(-1.0)] = 0
        m[weight.gt(1.0)] = 0
        # m = m.add(1.0/n).mul(1.0-1.0/s[1]).mul(n)
        # self.target_modules[index].grad.data = \
        #     self.target_modules[index].grad.data.mul(m)
        m = m.mul(self.target_modules[index].grad.data)
        m_add = weight.sign().mul(self.target_modules[index].grad.data)
        m_add = m_add.sum(3, keepdim=True)\
            .sum(2, keepdim=True).sum(1, keepdim=True).div(n).expand(s)
        m_add = m_add.mul(weight.sign())
        self.target_modules[index].grad.data = m.add(m_add).mul(1.0-1.0/s[1]).mul(n)

```

Q: 为什么 binary 过程中要求平均值?

- Backpropagation of neural network

Backpropagation of neural network



$$z_1^1 = w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1$$

$$z_2^1 = w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1$$

$$z_3^1 = w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1$$

$$a_1^1 = f(z_1^1) \quad a_2^1 = f(z_2^1) \quad a_3^1 = f(z_3^1)$$

$$z_1^2 = w_{11}^2 a_1^1 + w_{12}^2 a_2^1 + w_{13}^2 a_3^1 + b_1^2$$

$$z_2^2 = w_{21}^2 a_1^1 + w_{22}^2 a_2^1 + w_{23}^2 a_3^1 + b_2^2$$

$$a_1^2 = f(z_1^2) \quad a_2^2 = f(z_2^2)$$

Forward propagation

Update weights

$$d_1^1 = \frac{\partial E}{\partial z_1^1}$$

$$= \frac{\partial E}{\partial a_1^1} \frac{\partial a_1^1}{\partial z_1^1} \frac{\partial z_1^1}{\partial z_1^1}$$

$$= \frac{\partial E}{\partial a_1^1} \frac{\partial a_1^1}{\partial z_1^1}$$

$$\frac{\partial E}{\partial x_1} = \frac{\partial E}{\partial z_1^1} \cdot \frac{\partial z_1^1}{\partial x_1}$$

$$= d_1^1$$

$$\frac{\partial E}{\partial w_{ij}^1} = d_1^1 a_j^1$$

Back propagation

$$E_{total} = \frac{1}{2} \|y^{(k)} - o^{(k)}\|^2$$

$$= \frac{1}{2} \sum_{k=1}^N (y_k^i - a_k^i)^2 \quad \text{cost function}$$

Expected output
actual output

$$\text{Then } E_{total} = \frac{1}{N} \sum_{i=1}^N E_i$$

$$w^1 = w^1 - \eta \frac{\partial E_{total}}{\partial w^1} \quad b^1 = b^1 - \eta \frac{\partial E_{total}}{\partial b^1}$$

$$= w^1 - \eta \sum_{i=1}^N \frac{\partial E_i}{\partial w^1} \quad = b^1 - \eta \sum_{i=1}^N \frac{\partial E_i}{\partial b^1}$$

$$\text{Hidden layer: } E = \frac{1}{2} ((y_1 - f(w_{11}^2 a_1^1 + w_{12}^2 a_2^1 + w_{13}^2 a_3^1 + b_1^2))^2 + (y_2 - f(w_{21}^2 a_1^1 + w_{22}^2 a_2^1 + w_{23}^2 a_3^1 + b_2^2))^2)$$

$$\frac{\partial E}{\partial w_{11}^2} = \frac{1}{2} \cdot (y_1 - a_1^2) \cdot (-f'(z_1^2))$$

$$= -(y_1 - a_1^2) f'(z_1^2) a_1^1$$

$$\text{So } d_1^1 = \frac{\partial E}{\partial z_1^1} \quad \text{then } \frac{\partial E}{\partial w_{11}^2} = \frac{\partial E}{\partial z_1^2} \frac{\partial z_1^2}{\partial w_{11}^2} = d_1^2 a_1^1 \quad d_1^2 = \frac{\partial E}{\partial z_1^2} = -(y_1 - a_1^2) f'(z_1^2)$$

$$\text{And } \frac{\partial E}{\partial w_{12}^2} = d_1^2 a_2^1 \quad \frac{\partial E}{\partial w_{13}^2} = d_1^2 a_3^1 \quad \frac{\partial E}{\partial w_{21}^2} = d_2^2 a_1^1 \quad \frac{\partial E}{\partial w_{22}^2} = d_2^2 a_2^1 \quad \frac{\partial E}{\partial w_{23}^2} = d_2^2 a_3^1$$

$$\text{Usually, } d_1^1 = -(y_1 - a_1^2) f'(z_1^2) \quad \frac{\partial E}{\partial w_{ij}^1} = d_1^1 a_j^1$$

- Backpropagation of quantization neural network

Forward propagation: Binarization \rightarrow Input \rightarrow Hidden layer \rightarrow Output

Backpropagation: partial derivatives \rightarrow update weights \rightarrow binarization

- Tensorflow version of xnor

Running codes on <https://github.com/ljhandlwt/xnor-net-tf>

Haven't get stable results.

- Plan for next week
 1. Get familiar with pytorch and C++.
 2. Finish work for week 1 and 3.
 3. Learn more about XNOR.