Week 2

Kexin Fan

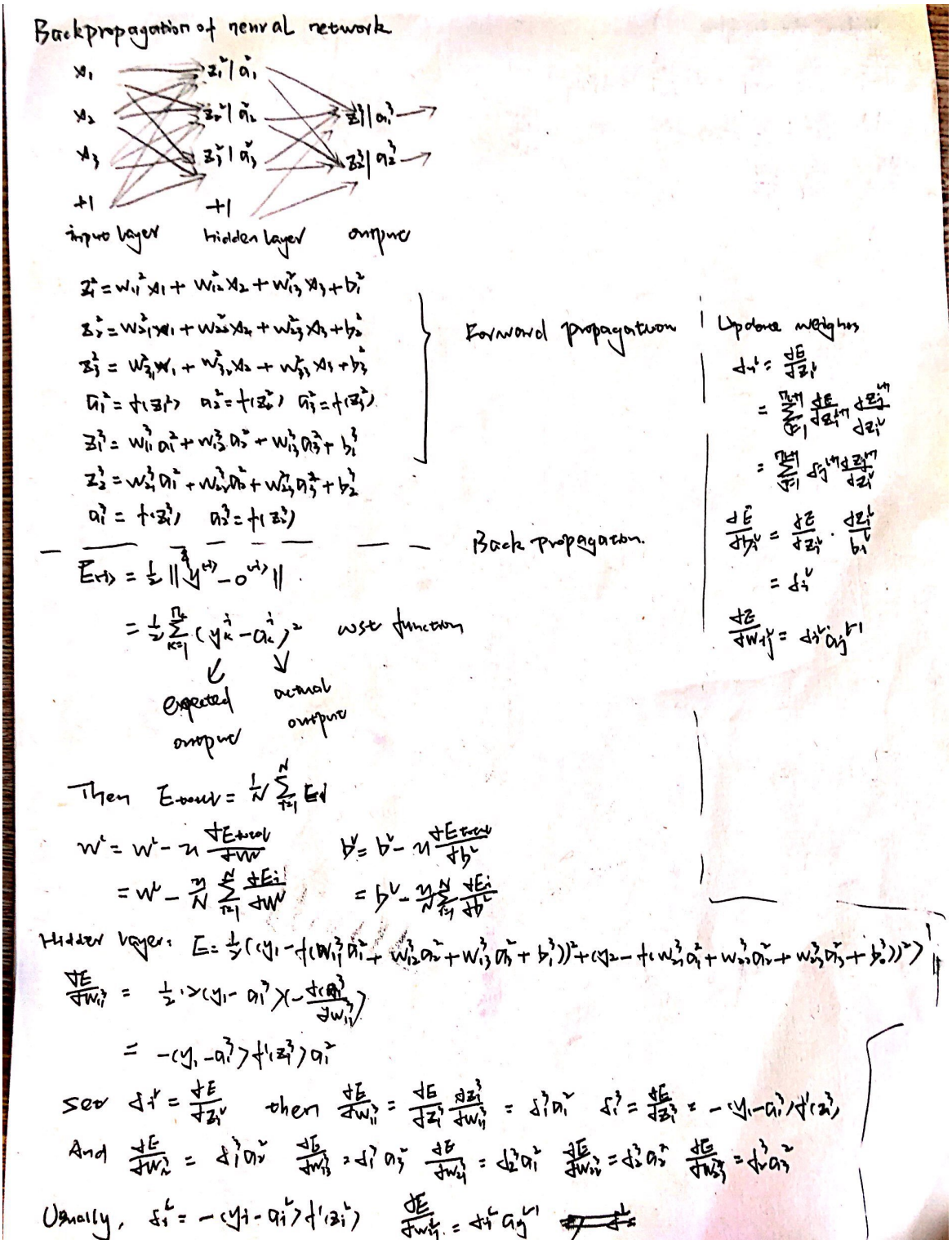
- Codes of XNOR-Net/util.py
  Understand and mark the codes.

```
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 过程中需要求平均值？

- Backpropagation of neural network



- Backpropagation of quantization neural network

Forward propagation: Binarization->Input ->Hidden layer -> Output

Backpropagation: partial derivatives -> update weights -> 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.