Homework 1

Your Name Benli Wu

Your Student ID 522031910763

Contents

1 Introduction

In Deep Learning, many things matter, and thus we prepare 5 questions to cover most of them. You shall understand Network Structure, Tensor Shape, and Execution Time in **Q1**; you shall understand Back Propagation (BP) and Automatic Differentiation (AD) in **Q2**; you shall grasp Optimization Scheme in **Q3**; you shall then understand basic hyper-parameter tuning, i.e., Batch Size and Epoch Number in **Q4**; after understanding everything above, you are doomed to dive into On-Edge Training to understand Dataset and the challenges for on-edge learning in **Q5**.
We sincerely hope you enjoy your journey through this homework as we do.

2 Q1: Tensor Shape & Execution Time (15')

Network Definition is the biggest components of Deep Learning. In this problem, you learn how to profile tensors' shape and execution time.

2.1 Problem Formulation

Network for Q1:

Listing 1: network structure

```
class Network(nn.Module):
    def __init__(self):
        super(Network, self).__init__()
        self.layers = []
        self.layers.append(nn.Conv2d(1, 10, kernel_size=5))
        self.layers.append(nn.ReLU(inplace=True))
        self.layers.append(nn.MaxPool2d(kernel_size=2))
        self.layers.append(nn.Conv2d(10, 20, kernel_size=5))
        self.layers.append(nn.ReLU(inplace=True))
        self.layers.append(nn.MaxPool2d(kernel_size=2))
        self.layers.append(nn.Flatten())
        self.layers.append(nn.Linear(320, 50))
        self.layers.append(nn.Linear(50, 10))
        self.layers = nn.ModuleList(self.layers)

    def forward(self, x):
        # print()
        for layer in self.layers:
            # start = time.time()
            x = layer(x)
            # end = time.time()
            # print(x.shape, "{0:d} mu s".format(int((end-start)*1e6)), sep
        return x
```

1. (5') Q: Fill in Table 1, print the shape of each layer's output. The first 2 are done for you.
   (Hint: See reference for *torch.Tensor* class.
   https://pytorch.org/docs/stable/tensors.html#torch-tensor)

2. (5') Q: Fill in Table 1, profile the execution time of each layer, both on CPU or on GPU. The first 2 are done for you.

   (Hint: timing in μs)

   (Hint: You shall not profile the first few batches on GPU, because GPU is warming up!!!! Also, measure multiple times on CPU, because CPU time is subject to multi-threading!!!)

3. (5') Q: Explain how to calculate the shape of the outputs of layers.

   (Hint: Look into the following links, learn how to compute the shape change of nn.Conv2d, nn.MaxPool2d, nn.Flatten, nn.Linear.

   https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

   https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html

   https://pytorch.org/docs/stable/generated/torch.nn.Flatten.html

   https://pytorch.org/docs/stable/generated/torch.nn.Linear.html)

2.2 Solution

1. (5') See Table 1.

2. (5') See Table 1. Layer id follows what presents in Listing 1. (32 is the batch size. represent it to be N.)

layer	type	shape of output tensor (N=32)	CPU time (μs)	GPU time (μs)
input	/	[N, 1, 28, 28]	/	/
0	Conv2d	[N, 10, 24, 24]	1387	738
1	ReLU	[N, 10, 24, 24]	3144	267
2	MaxPool2d	[N,10,12,12]	1583	241
3	Conv2d	[N,20,8,8]	2031	707
4	ReLU	[N,20,8,8]	2304	186
5	MAxPool2d	[N,20,4,4]	1309	201
6	Flatten	[N,320]	604	87
7	Linear	[N,50]	1021	372
8	Linear	[N,10]	1789	453

Table 1: Shape & Time Profile

3. (5') (Your answer here !!!)

   The original shape of tensor is [N,1,28,28]
   layer0(Conv2d): inChannels=1,outChannels=10,so output 10 channels.The kernelSize=5, so W,H both is 28-5+1=24.
   layer1(ReLU): The layer does't change the shape of input tensor.
   layer2(MaxPool2d):The kernelSize=2, so W,H will be half of their size.
   layer3(Conv2d): inChannels=10,outChannels=20,so output 20 channels.The kernelSize=5, so W,H both is 12-5+1=8.
   layer4(ReLU): The layer does't change the shape of input tensor.
   layer5(MaxPool2d):The kernelSize=2, so W,H will be half of their size.
   layer6(Flatten): The layer will flatten a contiguous range of dims into a tensor. C*W*H=320.So

the second dim will be 320
layer7(Linear): inFeatures=320, outFeatures=50, so it change 320 to 50.
layer8(Linear): inFeatures=50, outFeatures=10, so it change 50 to 10.

3 Q2: Backward Propagation (15')

3.1 Problem Formulation

(15') Q: Calculate gradient for Linear layer.
That is, you need to calculate ($\frac{\partial L}{\partial X}$, $\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial b}$), given (X, W, b, $\frac{\partial L}{\partial Y}$).
Linear Layer Forward Algorithm is as follows:

$$Y[n,t,f] = b[f] + \sum_{c=0}^{C} X[n,t,c] \cdot W[c,f] \tag{1}$$

Linear Layer Backward Algorithm is as follows (chain rule):
Input: (X,W,b,$\frac{\partial L}{\partial Y}$)

$$\begin{aligned}
&\frac{\partial L}{\partial X[n,t,c]} = \sum_{f}^{F} \frac{\partial L}{\partial Y[n,t,f]} \cdot \frac{\partial Y[n,t,f]}{\partial X[n,t,c]} = \sum_{f}^{F} \frac{\partial L}{\partial Y}[n,t,f] \cdot W[c,f] \\
&\frac{\partial L}{\partial W[c,f]} = TODO \\
&\frac{\partial L}{\partial b[f]} = TODO
\end{aligned} \tag{2}$$

Please Calculate $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$ yourself!!!
(You can learn more of PyTorch backward implementation yourself on https://pytorch.org/docs/stable/notes/extending.html#extending-autograd)

3.2 Solution

Calculate ($\frac{\partial L}{\partial X}$, $\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial b}$), given (X, W, b, $\frac{\partial L}{\partial Y}$).
For example, $\frac{\partial L}{\partial X}$ is calculated for you as follows.
Please Calculate $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$ yourself!!!
(Your answer here!!!)
Input: (X,W,b,$\frac{\partial L}{\partial Y}$)

$$\begin{aligned}
&\frac{\partial L}{\partial X[n,t,c]} = \sum_{f}^{F} \frac{\partial L}{\partial Y[n,t,f]} \cdot \frac{\partial Y[n,t,f]}{\partial X[n,t,c]} = \sum_{f}^{F} \frac{\partial L}{\partial Y}[n,t,f] \cdot W[c,f] \\
&\frac{\partial L}{\partial W[c,f]} = \sum_{n}^{N} \sum_{t}^{T} \frac{\partial L}{\partial Y[n,t,f]} \cdot \frac{\partial Y[n,t,f]}{\partial W[c,f]} = \sum_{n}^{N} \sum_{t}^{T} \frac{\partial L}{\partial Y}[n,t,f] \cdot X[n,t,c] \\
&\frac{\partial L}{\partial b[f]} = \sum_{n}^{N} \sum_{t}^{T} \frac{\partial L}{\partial Y[n,t,f]} \cdot \frac{\partial Y[n,t,f]}{\partial b[f]} = \sum_{n}^{N} \sum_{t}^{T} \frac{\partial L}{\partial Y}[n,t,f] \cdot 1
\end{aligned} \tag{3}$$

4 Q3: Optimization: Gradient Descent (15')

4.1 Problem Formulation

(15') You shall implement your own optimizer class *Nesterov*.
SGD with Nesterov Momentum follows the paper "A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$". It has better convergence results than standard sgd. The pseudo code is as follows in Alg 1, you shall implement it in PyTorch.

Algorithm 1: Nesterov

Input: dataset: $X = \{x_1, \ldots, x_N\}$, learning rate lr, Parameters W, loss function $f(x_i, W)$
Output: Best W

initialize W, 2 buffers V1, V2.
V1=V2=W, k=2
for *e=1 to E* **do**
  for *i=1 to N* **do**
    $V2 \longleftarrow V1$
    $V1 \longleftarrow W - lr \cdot \nabla_W f(x_i, W)$
    $W \longleftarrow V1 + \frac{k-2}{k+1}(V1 - V2)$ // V1-V2 is nesterov momentum
    $k \longleftarrow k + 1$ // k is iteration counter.
return W

Compare it with the Standard Gradient Descent Scheme in Alg 2.

Algorithm 2: Standard SGD

Input: dataset: $X = \{x_1, \ldots, x_N\}$, learning rate lr, Parameters W, loss function $f(x_i, W)$
Output: Best W

initialize W
for *e=1 to E* **do**
  for *i=1 to N* **do**
    $W \longleftarrow W - lr \cdot \nabla_W f(x_i, W)$
return W

4.2 Solution

Most of code is done for you in *q3code.py*. **However, you still need to implement Alg 1 in *step()*.**

Listing 2: Nesterov Optimizer

```
class Nesterov(object):
    def __init__(self, parameters, lr=1e-3) -> None:
        self.counter=2 # k
        self.lr = lr
        self.parameters=list(parameters) # W
        self.buffer1=[] # V_{k-1}
        self.buffer2=[] # V_{k-2}
        for param in self.parameters:
```

```
            tensor1 = torch.zeros_like(param.data)
            tensor2 = torch.zeros_like(param.data)
            tensor1[:] = param.data
            tensor2[:] = param.data
            self.buffer1.append(tensor1)
            self.buffer2.append(tensor2)
            self.buffer1[-1]._require_grad=False
            self.buffer2[-1]._require_grad=False

    def zero_grad(self):
        for param in self.parameters:
            param._grad = None

    def step(self):
        for param, param_m1, param_m2 in zip(
            self.parameters, self.buffer1, self.buffer2):
            # Example: SGD updates
            # param.data[:] = param.data - self.lr * param.grad
            # ##############################
           # TODO: Your code here!
            param_m2.data[:]=param_m1
            param_m1.data[:]=param -self.lr*param.grad
            param.data[:]=param_m1+(self.counter-2)/
            (self.counter+1)*(param_m1-param_m2)
            ################################
            None
        self.counter += 1
```

You shall install matplotlib on your nano. *sudo apt-get install python-matplotlib*
You shall reproduce my results in Figure 4, and **replace my figure with yours**.

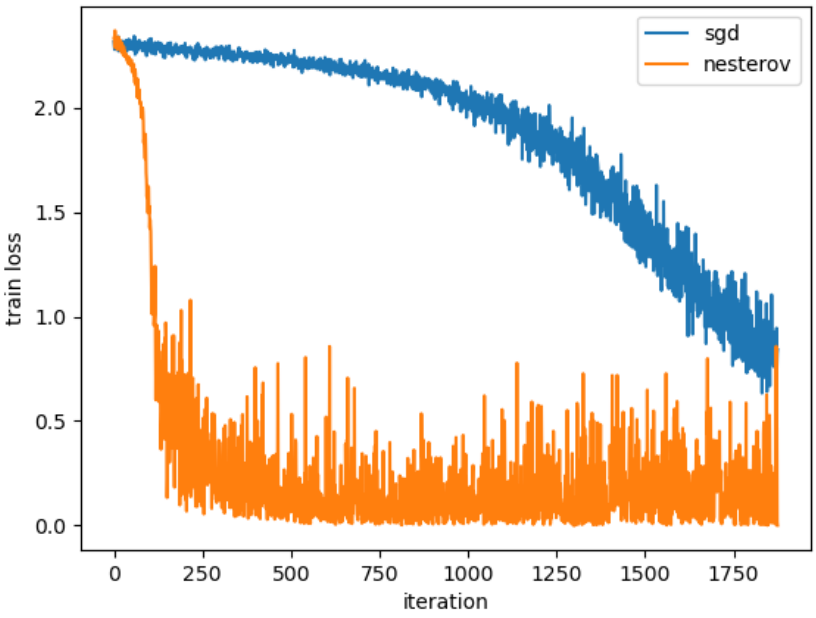

Figure 1: My Optimizer Comparison

(Your answer here!!!)

5 Q4: Hyper-parameters: Batchsize v.s. Epoch (15')

5.1 Problem Formulation

In Deep Learning training loop, there are many hyper-parameters, among which batchsize and epoch number matter most.

1. (5') Fill in Table 2, to see the effect of batchsize (B) on the variance of loss ($Var[L]$). You shall also conduct experiment to visualize the results with matplotlib.

2. (10') Fill in Table 2, to see the effect of iteration number on the mean of loss ($\mathbb{E}[L]$). You shall also conduct experiment to visualize the results with matplotlib.

   (*Hint:* iteration number=$\frac{N}{B} \cdot E$)

5.2 Solution

expr	B	E	$\frac{N}{B} \cdot E$	Var[L]	$\mathbb{E}[L]$
1	8	4	N/2	0.02127	0.04470
1	16	4	N/4	0.00651	0.03508
1	32	4	N/8	0.00429	0.03357
1	64	4	N/16	0.00246	0.04240
2	8	1	N/8	0.01650	0.04729
2	16	1	N/16	0.00916	0.05894
2	32	1	N/32	0.00581	0.05976
2	64	1	N/64	0.00312	0.05631
2	128	1	N/128	0.00322	0.07558

Table 2: Hyper Parameters

(Your answer here !!!) (Your figures here !!!)

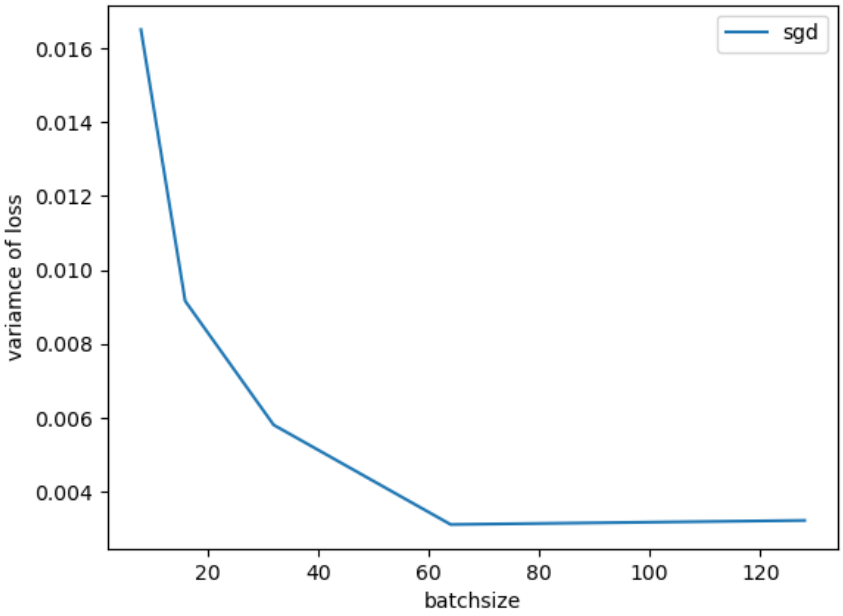

Figure 2: batchsize effect when E=1

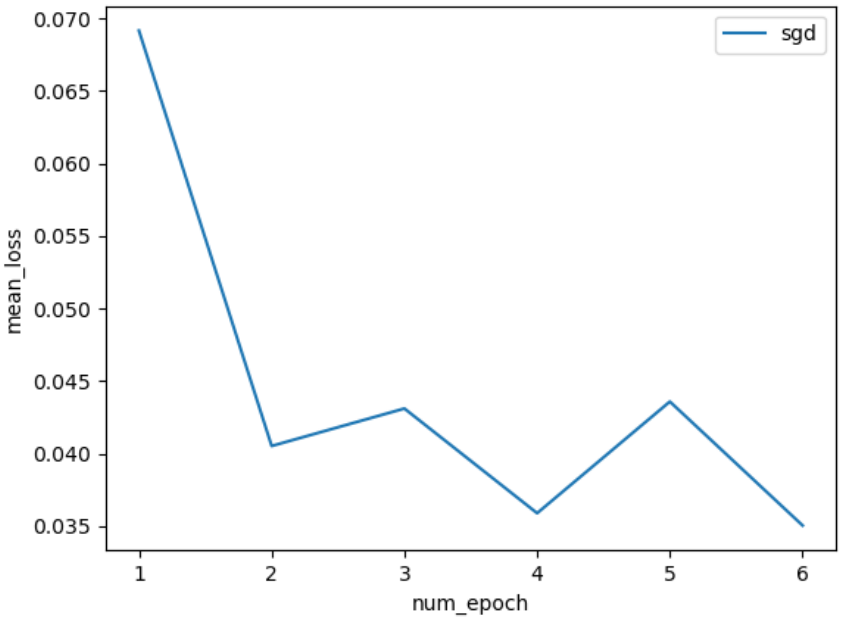

Figure 3: epoch effect when B=64

6 Q5: Have Fun on cifar100 (40')

6.1 Problem Formulation

Now, it's time for you to train your own network on cifar100. You shall take care of your Dataset and DataLoader yourself. You can devise your own network to beat mine. However, you shall keep in mind that the memory upper bound for you is limited, and thus any attempts for a large model are not possible.

1. (40') Finish the code, with your own model.

   (Hint:) Remember what you have learned in Q1,2,3,4. You can adjust your network structure, optimizer, batchsize and epoch number, and even resize swap size to enlarge the memory available.

6.2 Solution

(Your answer here!!!! Try to beat my results in Table 3. No problem if you can't.)

	B	E	train loss	test accuracy
hzx	4	10	4.38	22.1%
hzx	4	50	3.95	38.0%
Yours	4	10	4.28	30.9%

Table 3: Your Results

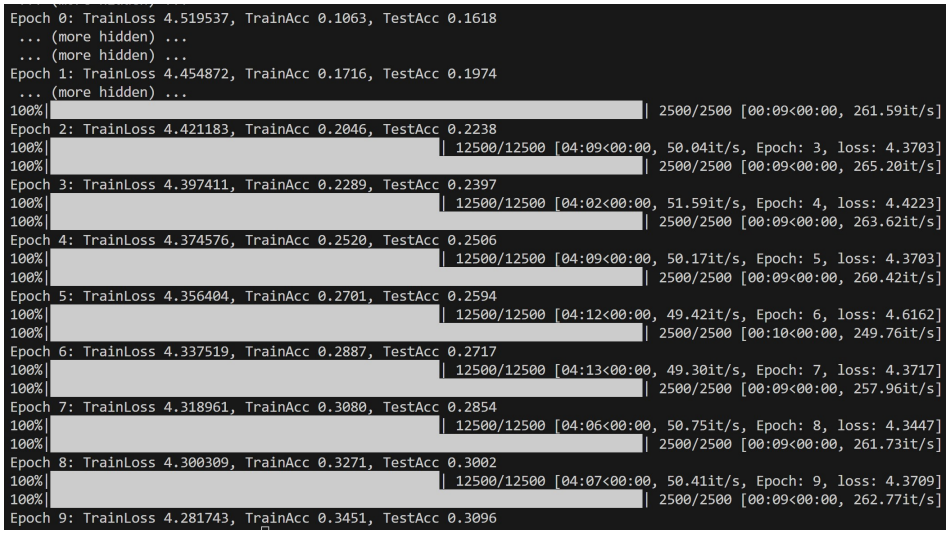

Figure 4: results

7 What have you learned?

7.1 Problem Formulation

Tell TA what you have learned, and the more you learned, the more points you will earn.

7.2 Tell me what you have learned

(Your answer here !!!)
1. I learned some basic grammar in python and pytorch.
2. I know how the tensor's shape change in different layers. Such as the convolutional layerpooling layer and linear layer.
3. I learned backward method to optimize the loss.
4. I learned different optimization algorithm will have different rate of gradient descent.
5. I learned the effect of batchsize and epoch on the variance and mean of loss.
6. I can program a simple model by things I have learned.

8 Acknowledgement

Thanks to the teacher and the teaching assistant for their guidance and assistance !

References