Pytorch - An Important Lesson about 1D Dimensions

Written by Tongyu Lu on 21-March-2021

Have you encountered such a problem when training a neural network: you are sure that your model is perfectly correct, and your training procedure is also correct. But you cannot get the expect result. Your model outputs seem blurred, and the loss curve does not show obvious descent. What is wrong?
You may have to check the dimensions of your variables! Sometimes, if one of your model variable has shape [B,X,1] and you so some calculation between it and another tensor [B,X], you may not get the correct result, although the calculation will proceed without runtime error. Here is a super simple example illustrating such phenomenon.

Introduction: a Super Simple Problem

I am using torch version 1.4.0+cu92 with Python 3.6

```
>>> import torch
>>> print(torch.__version__)
1.4.0+cu92
```

And I am trying to do such a super-simple regression problem: to approximate y=0.5x+0.5
The Python code is as follows:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data.dataset import Dataset
from torch.utils.data import DataLoader
import numpy as np
import math
import matplotlib.pyplot as plt
%matplotlib inline

class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fcy = nn.Linear(1, 1, bias=True)
    def forward(self, x):
        x = x.mean(1,keepdim=True)
        y = self.fcy(x)
        return y

class NonLinearFuncDataset(Dataset):
    def __init__(self, max_val = 10):
        self.max_val = max_val
    def __getitem__(self, index):
        x = np.random.rand(6)*self.max_val*2-self.max_val
        y = (np.mean(x)+1)/2
        return x.astype(np.float32), y.astype(np.float32)
    def __len__(self):
        return 1000000
```

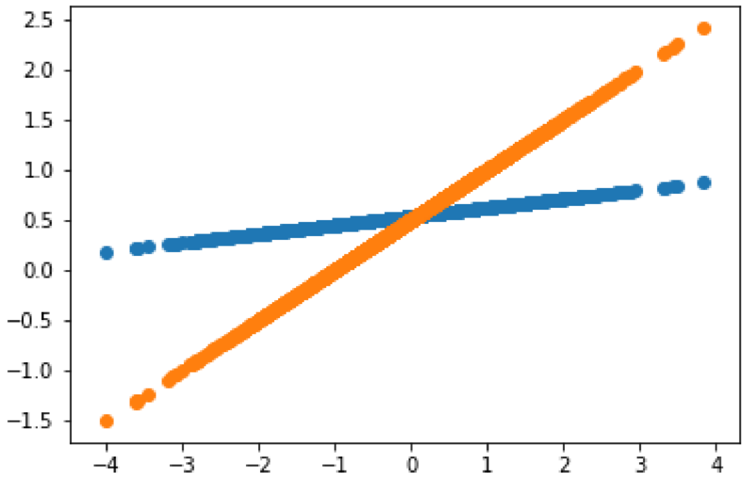
```
simple_nn = SimpleNN()
optimizer = optim.SGD(simple_nn.parameters(), lr=0.1)
train_dataset = NonLinearFuncDataset(max_val = 5)
train_loader = DataLoader(dataset = train_dataset, batch_size = 12)
train_iter = iter(train_loader)

loss_buf = 0
print_iter = 100
stop_iter = 2000
i = 0

while 1:
    x, y_true = next(train_iter)

    simple_nn.train()
    y_hat = simple_nn(x)
    loss = F.l1_loss(y_hat, y_true)
    loss.backward()
    loss_buf += loss.detach()
    optimizer.step()
    optimizer.zero_grad()

    if i%print_iter==0 and i>0:
        print(i, loss_buf.item()/print_iter, y_hat.mean().item(), y_true.mean().item())
        loss_buf = 0

    i += 1
    if i>=stop_iter:
        break

test_loader = DataLoader(dataset = train_dataset, batch_size = 2000)
test_iter = iter(test_loader)
simple_nn.eval()
x,y = next(test_iter)
y_hat = simple_nn(x)
fig, axes = plt.subplots(1,1)
axes.scatter(x.mean(1).detach().numpy(),y_hat.detach().numpy())
axes.scatter(x.mean(1).detach().numpy(),y.detach().numpy())
```

After running this python script on Jupyter, I got such results:

```
c:\users\administrator\appdata\local\programs\python\python35\lib\site-packages\ipykernel_launcher.py:6:
UserWarning: Using a target size (torch.Size([12])) that is different to the input size (torch.Size([12,
1])). This will likely lead to incorrect results due to broadcasting. Please ensure they have the same
size.
```

```
100 0.4973905563354492 0.4193195402622223 0.4412085711956024
200 0.4920030212402344 0.5123944282531738 0.5628079771995544
300 0.4645474243164063 0.47531524300575256 0.781419575214386
400 0.47979389190673827 0.5937886834144592 0.8502340316772461
500 0.48931766510009767 0.4544735252857208 0.5519954562187195
600 0.4721894454956055 0.46931830048561096 0.5291367769241333
700 0.4749597930908203 0.53408282995224 0.694856584072113
800 0.4737801742553711 0.5565791726112366 0.7290698885917664
900 0.4734691619873047 0.4045194387435913 0.36842620372772217
1000 0.47101482391357424 0.4549376964569092 0.3553701937198639
```

```
1100 0.48336479187011716 0.5444582104682922 0.4206082820892334
1200 0.46494224548339846 0.4717402756214142 0.5541265606880188
1300 0.46320755004882813 0.507659018039703 4 0.6052331328392029
1400 0.5017692184448242 0.54107666015625 0.5907941460609436
1500 0.47045547485351563 0.5345511436462402 0.38527199625968933
1600 0.4670344543457031 0.49572136998176575 0.3210914433002472
1700 0.4809828567504883 0.5050051808357239 0.5619488954544067
1800 0.473972091674804 7 0.5584373474121094 0.731599748134613
1900 0.4783402252197266 0.511594295501709 0.4001244306564331
```



I was kind of surprised when having this result!
It seems that there is no problem with my code. But the outputs are not reasonable! The model outputs (blue line) are flat, which deviate the ground truths (orange line). And the loss does not seem to go down. What is wrong?

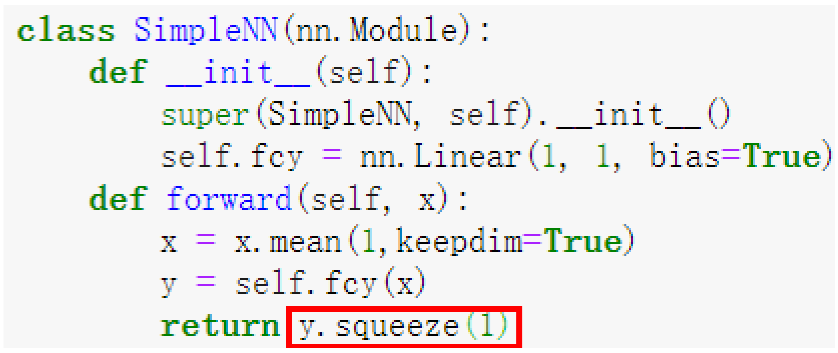
Warning: [12,1] is not [12]

The problem is the mismatch of tensor shapes. It seems that there is no problem when computing loss function between a [12,1]-shaped tensor and a [12]-shaped tensor. But that is wrong.
Pytorch treats `loss(tensor1_shape12_1,tensor2_shape12)` and `loss(tensor1_shape12,tensor2_shape12)` differently. And this experiment shows that the prior case lead the model to output the mean value of targets in the dataset.
Thus, we could infer that the latter case is correct. Let us change a little bit:

Previously, we have

```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fcy = nn.Linear(1, 1, bias=True)
    def forward(self, x):
        x = x.mean(1,keepdim=True)
        y = self.fcy(x)
        return y
```

The shape-flow is actually: **x(input): [12,6]→x(mid):[12,1]→y:[12,1]→return:[12,1]**
And after modification, we have

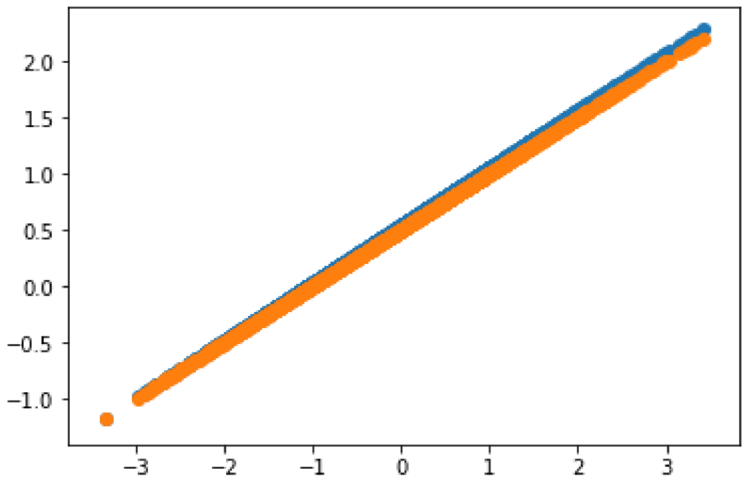


```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fcy = nn.Linear(1, 1, bias=True)
    def forward(self, x):
        x = x.mean(1,keepdim=True)
        y = self.fcy(x)
        return y.squeeze(1)
```

The shape-flow becomes: **x(input): [12,6]→x(mid):[12,1]→y:[12,1]→return:[12]**

Rerun our code, we get:

```
100 0.06891810417175293 0.6146644353866577 0.665600597858429
200 0.049247756004333496 0.5571919083595276 0.5793744325637817
300 0.050302953720092775 0.4834255278110504 0.4364989101886749
400 0.04972605228424072 0.6058599352836609 0.6007675528526306
500 0.05235084533691406 0.5475488305091858 0.5476792454719543
600 0.049787888526916506 0.6493135690689087 0.6331785321235657
700 0.050062098503112794 0.12336063385009766 0.15845249593257904
800 0.0486941146850586 0.7547333836555481 0.7954740524291992
900 0.05166142463684082 0.3957298994064331 0.3975466787815094
1000 0.05193905830383301 0.5756499171257019 0.5020708441734314
1100 0.04927664756774902 0.6729649901390076 0.6761811375617981
1200 0.04754136085510254 0.19828061759471893 0.21505200862884521
1300 0.0492149829864502 0.6152416467666626 0.5980979800224304
1400 0.04955647945404053 0.5035203099250793 0.5153853893280029
1500 0.04938608169555664 0.22917580604553223 0.22565217316150665
1600 0.048435945510864255 0.7864841818809509 0.7529808878898621
1700 0.048812971115112305 0.8886194229125977 0.8871995806694031
1800 0.0514029598236084 0.08457982540130615 0.06996902823448181
1900 0.05282351493835449 0.404921293258667 0.3546682596206665
```



This is definitely better!!! Problem solved!