


Do I need to consider the batch_size when defining own module

king_wang

Apr '18

From the tutorial, I got that



PyTorch: Custom nn Modules

Sometimes you will want to specify models that are more complex than a sequence of existing Modules; for these cases you can define your own Modules by subclassing `nn.Module` and defining a `forward` which receives input Variables and produces output Variables using other modules or other autograd operations on Variables.

In this example we implement our two-layer network as a custom Module subclass:

```
# -*- coding: utf-8 -*-
import torch
from torch.autograd import Variable

class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        """
        In the constructor we instantiate two nn.Linear modules and assign them as
        member variables.
        """
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Variables.
        """
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random Tensors to hold inputs and outputs, and wrap them in Variables
```

we need to consider the batch_size when using the Linear layer. Then if I define own module ,do i need to consider the batch_size, e.g if the original input is a input_feature Tensor, if we consider the batch_size, then the input will be batch_size*input_feature Tensor

ptrblck

Apr '18

You don't need to consider the batch size when initializing the Modules. The Linear layer for example takes `in_features` as an argument, which would be dimension 1 for `x = torch.randn(10, 20)`.

However, when you need another view on the Tensor, e.g when you need to flatten the Tensor coming from a Conv2d, you most likely want to keep the batch size and flat all remaining dimensions.

You would do it in the forward method:

```
x = self.conv(x)
x = x.view(x.size(0), -1) #keep batch size
x = self.fc(x)
```

Does this explanation make it clearer?