How to set dimension for softmax function in PyTorch?

PyTorch May 4, 2023 October 3, 2022

Softmax is a function that takes a vector of values and produces another vector of the same dimension, where the values represent **probabilities**. It takes the elements of the vector, computes the elementwise exponential, and divides each element by the sum of exponentials. In code, it's something like this:

```
def softmax(x):
      retur n torch.exp(x) / torch.exp(x).sum()
```

We also use `torch.nn.functional.softmax()` to normalize our outputs to the range `[0,1]`. That gives us something roughly akin to the confidence that the model has in its prediction.

The **`nn`** module makes softmax available as a module. Since, as usual, input tensors may have an **additional batch 0th dimension**, or have dimensions along which they encode probabilities and others in which they don't, `nn.Softmax` requires us to specify the dimension along which the softmax function is applied. Let's test it on an input vector:

```
inputs = torch.tensor([[1.0, 2.0, 3.0],
                                    [1.0, 2.0, 3.0]])

softmax = torch.nn.Softmax(dim=1)
```

```
output=softmax(inputs)

print(output) #tensor([[0.0900, 0.2447, 0.6652],
                       [0.0900, 0.2447, 0.6652]])
```

As expected, it satisfies the constraints on probability:

```
print(output.numpy().sum(axis=1)) #[1. 1.]
```

Softmax is a monotone function, in that lower values in the input will correspond to lower values in the output. However, it does not scale invariant, in that the ratio between values is not preserved.

In this case, we have two input vectors in two rows (just like when we work with batches), so we initialize `nn.Softmax` to operate along dimension **1**. We can now add a softmax at the end of our model, and our network will be equipped to produce probabilities:

```
model = nn.Sequential(
                nn.Linear(512, 256),
                nn.Tanh(),
                nn.Linear(256, 2),
                nn.Softmax(dim=1))
```

Softmax is implemented through a neural network layer just before the output layer. The Softmax layer must have the same number of nodes as the output layer.

Dimension

We can summarize the dimensionality of a tensor by printing the "**shape**" property, which is a tuple, where the number of values in the tuple defines the number of dimensions, and the integer in each position defines the size of the dimension.

```
print(output.shape) #torch.Size([2, 3])
```

For example, we expect the shape of our output tensor to be `(2,3)` for two rows and three columns. But how do we access data in the tensor by row or column? More importantly, how can we perform operations on the tensor by row or by column?

Data in tensor can be accessed directly via column and row indexes, and this is reasonably straightforward. Sometimes we must perform operations on tensor data such as `argmax` or `softmax` of values by row or column and this requires the dimension to be specified.

We can specify the `axis` as the dimension across which the operation is to be performed, and this dimension does not match our intuition based on how we interpret the "**shape**" of the array and how we index data in the tensor.

```
softmax = torch.nn.Softmax(dim=0)
output=softmax(inputs)
print(output)    #tensor([[0.5000, 0.5000, 0.5000],
                         [0.5000, 0.5000, 0.5000]])
```

Specifically, operations like softmax can be performed **column-wise using dim=0** and **row-wise using dim=1.** That is, **dim=0** will perform the operation column-wise and **dim=1** will perform the operation row-wise.

```
print(output.numpy().sum(axis=0))  #[1. 1. 1.]
```

The dim argument is required, it specifies the axis along which to apply the softmax activation. Passing in **dim=-1** applies softmax to the last dimension. So, after you do this, the elements of the last dimension will sum to 1.

Related Post

- Advantage of using LogSoftmax vs Softmax vs Crossentropyloss in PyTorch
- How to implement softmax and cross-entropy in Python and PyTorch
- Loss function for multi-class and multi-label classification in Keras and PyTorch
- Activation function for Output Layer in Regression, Binary, Multi-Class, and Multi-Label Classification

Recent Posts