Machine Learning

Pytorch Tutorial 2

Example of Using Pytorch

TA: 林子權 (Tzu-Quan Lin)
Task Description

- Given survey results in the past 5 days in a specific state in U.S., then predict the percentage of new tested positive cases in the 5th day.
Data

- In this case, data is included in a .csv file
- Each row represents a sample of data, containing 118 feature (id + 37 states + 16 features * 5 days)
- the last element of a row is its label

<table>
<thead>
<tr>
<th>id</th>
<th>AL</th>
<th>AK</th>
<th>AZ</th>
<th>AR</th>
<th>CA</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>smoothed_worried_finances</th>
<th>smoothed_tested_positive_14d</th>
</tr>
</thead>
<tbody>
<tr>
<td>37.3295118</td>
<td>7.4561538</td>
</tr>
<tr>
<td>32.5088806</td>
<td>8.010957</td>
</tr>
<tr>
<td>36.7455876</td>
<td>2.9069774</td>
</tr>
<tr>
<td>38.6801619</td>
<td>12.5758159</td>
</tr>
</tbody>
</table>
Load data / Preprocessing

Load data: You can use **pandas** to load a csv file.

```python
train_data = pd.read_csv('./covid.train.csv').drop(columns=['date']).values
```

Preprocessing: Get model inputs and labels.

```python
x_train, y_train = train_data[:, :-1], train_data[:, -1]
```

```python
print(x_train.shape)
pd(y_train.shape)
```

```
(2699, 117)
(2699,)
```
Dataset

- \_\_init\_: Read data and preprocess
- \_\_getitem\_: Return one sample at a time. In this case, one sample includes a 117 dimensional feature and a label
- \_\_len\_: Return the size of the dataset. In this case, it is 2699

```python
class COVID19Dataset(Dataset):
    ...
    x: Features.
    y: Targets, if none, do prediction.
    ...
    def \_\_init\_(self, x, y=None):
        if y is None:
            self.y = y
        else:
            self.y = torch.FloatTensor(y)
            self.x = torch.FloatTensor(x)

    def \_\_getitem\_(self, idx):
        if self.y is None:
            return self.x[idx]
        else:
            return self.x[idx], self.y[idx]

    def \_\_len\_(self):
        return len(self.x)
```

```python```
train_dataset = COVID19Dataset(x_train, y_train)
```
Dataloader

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, pin_memory=True)
```

- Group data into batches
- If you set `shuffle=True`, dataloader will permutes the indices of all samples automatically.
- We often set `shuffle=True` during training
- You can check this page [Advantage to shuffle a dataset](#) if you are curious about why we should shuffle the data during training
Model

- The input dimension of our model will be 117
- The output of our model will be a scalar, which represents the predicting value of the percentage of new tested positive cases in the 5th day

```python
class My_model(nn.Module):
    def __init__(self, input_dim):
        super(My_model, self).__init__()
        # TODO: modify model's structure, be aware of dimension.
        self.layers = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        x = self.layers(x)
        x = x.squeeze(1)  # (B, 1) -> (B)
        return x

model = My_Model(input_dim=x_train.shape[1]).to('cuda')
```
We are doing a regression task, choosing mean square error as our loss function would be a good idea!

criterion = torch.nn.MSELoss(reduction='mean')

More choices! : [pytorch documentation: torch.nn]
Optimizer

We need to declare an optimizer that adjusts network parameters in order to reduce error.

Here we choose stochastic gradient descent as our optimization algorithm.

```python
optimizer = torch.optim.SGD(model.parameters(), lr=1e-5, momentum=0.9)
```
Training loop

```python
for epoch in range(3000):
    model.train()  # Set your model to train mode.
    # tqdm is a package to visualize your training progress.
    train_pbar = tqdm(train_loader, position=0, leave=True)
    for x, y in train_pbar:
        x, y = x.to('cuda'), y.to('cuda')  # Move your data to device.
        pred = model(x)
        loss = criterion(pred, y)
        loss.backward()  # Compute gradient(backpropagation).
        optimizer.step()  # Update parameters.
        optimizer.zero_grad()  # Set gradient to zero.
```

Get model prediction, compute gradient, update parameters and reset the gradient of model parameters.
Note!

- The example code in this pytorch tutorial is slightly different from the sample code of hw1 for explanation convenience.
- Please refer to the sample code of hw1. 請以hw1的sample code為準。
Any Question?
Machine Learning
Pytorch Tutorial 3
Documentation and Common Errors

TA : 林子權 (Tzu-Quan Lin)
PyTorch Documentation

https://pytorch.org/docs/stable/
torch.nn -> neural network
torch.optim -> optimization algorithms
torch.utils.data -> dataset, dataloader
PyTorch Documentation Example

Function inputs and outputs

TORCH.MAX

torch.max(input) → Tensor

Returns the maximum value of all elements in the input tensor.

- WARNING

This function produces deterministic (sub)gradients unlike max(dim=0)

Parameters

input (Tensor) – the input tensor.
Some functions behave differently with different inputs

Parameters: You don’t need to specify the name of the argument (Positional Arguments)

Keyword Arguments: You have to specify the name of the argument

*NOTE*

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

Parameters

- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the dimension to reduce.
- **keepdim** (*bool*) – whether the output tensor has *dim* retained or not. Default: *False*.

Keyword Arguments

- **out** (*tuple*, optional) – the result tuple of two output tensors (*max*, *max_indices*)
PyTorch Documentation Example

Some functions behave differently with different inputs.

Arguments with default value:
Some arguments have a default value (keepdim=False), so passing a value of this argument is optional.

```
torch.max(input, dim, keepdim=False, out=None) -> (Tensor, LongTensor)
```

Returns a namedtuple `values, indices` where `values` is the maximum value of each row of the `input` tensor in the given dimension `dim`. And `indices` is the index location of each maximum value found (argmax).

If `keepdim` is True, the output tensors are of the same size as `input` except in the dimension `dim` where they are of size 1. Otherwise, `dim` is squeezed (see `torch.squeeze()`) resulting in the output tensors having 1 fewer dimension than `input`.

NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

Parameters

- `input (Tensor)` – the input tensor.
- `dim (int)` – the dimension to reduce.
- `keepdim (bool)` – whether the output tensor has `dim` retained or not. Default: False.

Keyword Arguments

- `out (tuple, optional)` – the result tuple of two output tensors (`max`, `max_indices`)
PyTorch Documentation Example

Three Kinds of torch.max

1. torch.max(input) → Tensor
2. torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)
3. torch.max(input, other, *, out=None) → Tensor
   input : Tensor, dim : int, keepdim : bool
   other : Tensor
PyTorch Documentation Example

1. `torch.max(input) → Tensor`

Find the maximum value of a tensor, and return that value.

```
input
[[1  2  3]
 [5  6  4]]
```
PyTorch Documentation Example

2. `torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)`

Find the maximum value of a tensor along a dimension, and return that value, along with the index corresponding to that value.

```
input
[[1  2  7]]
[[5  6  4]]
```
PyTorch Documentation Example

`torch.max(input, other) → Tensor`

Perform element-wise comparison between two tensors of the same size, and select the maximum of the two to construct a tensor with the same size.

<table>
<thead>
<tr>
<th>input</th>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[1 2 3]]</td>
<td>[[2 4 6]]</td>
</tr>
<tr>
<td>[[5 6 4]]</td>
<td>[1 3 5]]</td>
</tr>
</tbody>
</table>
PyTorch Documentation Example (Colab)

Three Kinds of \texttt{torch.max}

1. \texttt{torch.max(input)} → Tensor
2. \texttt{torch.max(input, dim, keepdim=False, *, out=None)} → (Tensor, LongTensor)
3. \texttt{torch.max(input, other, *, out=None)} → Tensor

**Colab code**

```
x = torch.randn(4,5)
y = torch.randn(4,5)
1. m = torch.max(x)
2. m, idx = torch.max(x,0)
   m, idx = torch.max(input = x,dim=0)
   m, idx = torch.max(x,0,keepdim=True)
   m, idx = torch.max(x,0,keepdim=False,out=p)
   *out is a keyword argument
   m, idx = torch.max(x,True)
3. t = torch.max(x,y)
```

*did not specify dim*
Common Errors -- Tensor on Different Device to Model

```python
model = torch.nn.Linear(5,1).to("cuda:0")
x = torch.randn(5).to("cpu")
y = model(x)

Tensor for * is on CPU, but expected them to be on GPU

=> send the tensor to GPU

x = torch.randn(5).to("cuda:0")
y = model(x)
print(y.shape)
```
Common Errors -- Mismatched Dimensions

```python
x = torch.randn(4,5)
y = torch.randn(5,4)
z = x + y
The size of tensor a (5) must match the size of tensor b (4) at non-singleton dimension 1

=> the shape of a tensor is incorrect, use transpose, squeeze, unsqueeze to align the dimensions

y = y.transpose(0,1)
z = x + y
print(z.shape)
```
CUDA out of memory. Tried to allocate 350.00 MiB (GPU 0; 14.76 GiB total capacity; 11.94 GiB already allocated; 123.75 MiB free; 13.71 GiB reserved in total by PyTorch)

=> The batch size of data is too large to fit in the GPU. Reduce the batch size.
Common Errors -- Cuda Out of Memory

If the data is iterated (batch size = 1), the problem will be solved. You can also use DataLoader

```python
for d in data:
    out = resnet18(d.to("cuda:0").unsqueeze(0))
print(out.shape)
```
import torch.nn as nn
L = nn.CrossEntropyLoss()
outs = torch.randn(5, 5)
labels = torch.Tensor([1, 2, 3, 4, 0])
lossval = L(outs, labels)  # Calculate CrossEntropyLoss between outs and labels

expected scalar type Long but found Float

=> labels must be long tensors, cast it to type “Long” to fix this issue

labels = labels.long()
lossval = L(outs, labels)
print(lossval)
Any Question?