Deep Learning with PyTorch

한양대학교 최솔비

Tensor 만들기

1. `torch.FloatTensor(3,2)`
2. `torch.FloatTensor([[1,2,3],[3,2,1]])`
3. `n = np.zeros(shape=(3,2), dtype = np.float32)`
   `torch.tensor(n)`
4. `torch.tensor(n, dtype = torch.float32)`

- tensor 의 data type 은 float를 주로 사용한다.
- 32나 16비트로도 충분한 경우가 대부분이기 때문에 numpy array 를 인자로 tensor 생성할 때에는 type 을 `float32` 등으로 지정한다.
  (default 는 `float64`)

Scalar tensors

```
>>> a = torch.tensor([1,2,3])
>>> a
tensor([1, 2, 3])
>>> s = a.sum()
>>> s
tensor(6)
>>> s.item()
6
>>> torch.tensor(1)
tensor(1)
```

- scalar 값에 접근하기 위해서는 `item()` 을 이용해야 함.

Tensor operations

- inplace operation: `a.zero_()` 와 같이 underscore 가 붙은 메서드는 tensor 객체의 내용을 바꾸는 연산을 한다.
- functional operation: `a.zero()` 와 같은 메서드는 tensor 를 copy 하여 연산한다. original tensor는 수정되지 않는다.

GPU tensors

- 모든 tensor 들은 cpu 버전, gpu 버전이 있다.
- `torch.FloatTensor` 대신 `torch.cuda.FloatTensor` 메서드로 gpu용 tensor 를 생성할 수 있다.
- 이미 생성된 tensor 를 바꾸려면 `to(device)` 를 사용한다.

Tensors and gradients

모든 tensor 들은 gradient 와 관련된 다음 attributes 를 가지고 있다.

- `grad` : 계산된 gradient를 담고 있는 같은 shape 의 tensor.
- `is_leaf` : user 가 만들었으면 `True` , function transformation의 결과라면 `False` .
- `requires_grad` : gradient 가 계산되어야 한다는 의미.
  leaf tensor 의 property이며 default 는 `False` 이므로 텐서 생성할 때 그라디언트 계산을 원한다면 해당 플래그를 넣어주어야 한다.

example

```
>>> v1 = torch.tensor([1.0, 1.0], requires_grad = True)
>>> v2 = torch.tensor([2.0, 2.0])
>>> v_sum = v1 + v2
>>> v_res = (v_sum*2).sum()
>>> v_res
tensor(12., grad_fn=<SumBackward0>)
```

- v1과 v2는 직접 생성한 tensor 이므로 leaf_node
- v_sum 과 v_res 는 leaf_node 가 아니다.
- v2 는 `requires_grad` 가 `False` 이며 나머지는 모두 `True` 이다.

example(continue)

```
>>> v1.is_leaf, v2.is_leaf
(True, True)
>>> v_sum.is_leaf, v_res.is_leaf
(False, False)
>>> v1.requires_grad
True
>>> v2.requires_grad
False
>>> v_sum.requires_grad
True
>>> v_res.requires_grad
True
```

example(continue)

```
>>> v_res.backward()
>>> v1.grad
tensor([2., 2.])
```

`.backward()` 메서드를 이용하여 v_res를 다른 variable 들에 대해 미분 한다. v1의 각 원소가 1이 증가할 때, v_res 의 각 원소는 2 증가함을 알수 있다.

```
>>> v2.grad
>>>
```

v2 는 `requires_grad` 가 `False` 였기 때문에 grad값이 없다.

Neural Network

NN building blocks

torch.nn 패키지의 인스턴스들은 그 자체로 함수로 사용될 수 있다(callable).

```
>>> import torch.nn as nn
>>> l = nn.Linear(2,5)  #2개 inputs, 5개 outputs
>>> l
Linear(in_features=2, out_features=5, bias=True)
>>> v =  torch.FloatTensor ([1,2])
>>> l(v)
tensor([-0.4071,  1.1070, -0.4442, -1.5530,  0.0566], gra
```

- (nn 의 weight 은 적절히 초기화 되어있음)

nn.Module 의 유용한 메소드

- `parameters()`
- `zero_grad()` : 모든 parameters 의 gradients 를 0으로 초기화.
- `to(device)`
- `state_dict()`
- `load_state_dict()`

Sequential class

```
>>> s = nn.Sequential(nn.Linear(2,5), nn.ReLU(),
nn.Linear(5,3), nn.Dropout(p=0.3), nn.Softmax(dim=1))
>>> s
Sequential(
  (0): Linear(in_features=2, out_features=5, bias=True)
  (1): ReLU()
  (2): Linear(in_features=5, out_features=3, bias=True)
  (3): Dropout(p=0.3)
  (4): Softmax()
)
>>> s(torch.FloatTensor([[1,2]]))
tensor([[0.2077, 0.4451, 0.3472]], grad_fn=<SoftmaxBackwa
```

(sequential nn 의 마지막에 ReLU 가 아닌 softmax 함수를 적용하는 이유는 정규화를 위해서인듯)

Custom layers

custom module을 만드려면

1. submodules 을 등록하고,
2. `forward()` 메서드를 override해야 한다.

example

```
class OurModule(nn.Module):
        def __init__(self, num_inputs, num_classes, dropou
                super(OurModule, self).__init__()
                self.pipe = nn.Sequential(
                        nn.Linear(num_inputs, 5),
                        nn.ReLU(),
                        nn.Linear(5, 20),
                        nn.ReLU(),
                        nn.Linear(20, num_classes),
                        nn.Dropout(p = dropout_prob),
                        nn.Softmax()
                        )
```

- pipe field 에 할당하는 것이 submodule 등록이다.

example(cont.)

```
        def forward(self, x):
                return self.pipe(x)
```

- `forward()` 메소드를 새로운 pipe를 이용하도록 override 한다.

```
if __name__ == "__main__":
        net = OurModule(num_inputs=2, num_classes=3)
        v = torch.FloatTensor([2,3])
        out = net(v)
        print(net)
        print(out)
```

- `forward()` 를 직접 호출하면 안됨. callable로 (instance를 함수로 생각하고 인자를 주어서) 호출하여야 한다.

example(result)

```
  input = module(input)
OurModule(
  (pipe): Sequential(
    (0): Linear(in_features=2, out_features=5, bias=True)
    (1): ReLU()
    (2): Linear(in_features=5, out_features=20, bias=True
    (3): ReLU()
    (4): Linear(in_features=20, out_features=3, bias=True
    (5): Dropout(p=0.3)
    (6): Softmax()
  )
)
tensor([0.6613, 0.1694, 0.1694], grad_fn=<SoftmaxBackward
```

마지막 : Loss function 과 Optimizer

Loss functions

- `nn.Module` 의 subclass 이다.
- nn의 output(prediction)과 desired output 을 인자로 받는다.
- 가장 자주 쓰이는 loss functions
  - `nn.MSELoss` : mean square error
  - `nn.BCELoss` , `nn.BCEWithLogits` : binary cross entropy loss. binary classification 문제에 자주 사용된다.
  - `nn.CrossEntropyLoss` , `nn.NLLLoss` : maximum likelihood (최대우도)를 이용. multi-class 분류에서 자주 사용된다.

Optimizers

- `torch.optim` 에서 제공한다.
- 가장 많이 쓰이는 optimizers
  - `SGD` : 보통의 확률적 경사 하강법(stochastic gradient descent algorithm) 적용
  - `RMSprop`
  - `Adagrad`

> SGD란?
> 배치(batch) 크기가 1인 경사 하강법 알고리즘이다.
> 기울기를 계산 하기 위해 사용하는 예제를 1개로 잡는 것이다.

example : 전체 과정! 잘 보자

```
# 각각의 batch sample과 lable(desired data)에 대하여 실행합니다.
# 모든 data 에 대하여 이 과정을 한바퀴 다 수행한 것을 1 epoch(에포크)
# 라고 합니다.
# cpu 나 gpu 는 이를 같은 사이즈의 batch 들로 나누어 계산합니다.
for batch_samples, batch_labels in iterate_batches(data,
        batch_samples_t = torch.tensor(batch_samples)
        batch_lables_t = torch.tensor(batch_labels)
        # neural network 통과한 후
        out_t = net(batch_samples_t)
        # loss fuction 으로 loss value 계산
        loss_t = loss_function(out_t, batch_lables_t)
        # back propagation
        loss_t.backward()
        # parameter 갱신
        optimizer.step()
        optimizer.zero_grad()
```

example(cont.)

- `backward()` 를 수행할 때 마다 `require_grad=True` 인 모든 tensor 의 `grad` 를 계산하여 축적한다.
- `optimizer.step()` 는 구해진 gradients 를 이용해 parameters 를 갱신한다.
- `optimizer.zero_grad()` 는 `grad` field 를 0으로 초기화한다.

Monitoring with TensorBoard

- 딥러닝은 세팅해줘야하는 parameter 들이 무척 많으며 이를 조정하는 과정이 매우 고통스럽기 때문에 모니터링 스킬이 필요하다고 한다...
- 다음 요소들을 잘 관찰하면 좋다.

1. Loss value: total loss 와 individual components
2. training, test sets 의 유효 결과
3. gradients 와 weight 의 통계
4. learning rates 와 다른 hyper parameters

- TensorBoard 사용하자.

끝