Basic Configuration Of Pytorch Package

Pytorch는 딥러닝을 통해 Classification을 만들어 보기 좋은 python library 입니다. Pytorch를이용해 다양한 기계학습 방법론을 Data에 적용시켜 볼 수 있습니다. 이 사이트는 Python으로 기계학습의 여러 방법론 중 Deep Learning을 실습하기 위해 만들어졌습니다. 이 교육과정은 Python 환경에서 pytorch를 이용해 CNN 모델을 만들고 Gradient Descent Algorithm을 적용시켜 모델에 Data(Cat and Dog)를 학습시키고, 이를 통해 새로운 Data가 들어왔을 때 Cat 인지 Dog인지 구분할 수 있게 하는 Cat and Dog Classification Model을 만드는 것을 최종 목표로 합니다.

**1. torch** 메인 네임스페이스입니다. 텐서 등의 다양한 수학 함수가 포함되어져 있으며 Numpy와 유사한 구조를 가집니다.

**2. torch.autograd** 자동 미분을 위한 함수들이 포함되어져 있습니다. 자체 미분 가능 함수를 정의할 때 사용하는 기반 클래스인 'Function' 등이 포함되어져 있습니다.

**3. torch.nn** 신경망을 구축하기 위한 다양한 데이터 구조나 레이어 등이 정의되어 있습니다. 예로는 ReLU와 같은 활성화 함수, MSELoss와 같은 손실 함수들이 있습니다.

**4. torch.optim** 확률적 경사 하강법(Stochastic Gradient Descent, SGD)를 중심으로 한 파라미터 최적화 알고리즘이 구현되어져 있습니다.

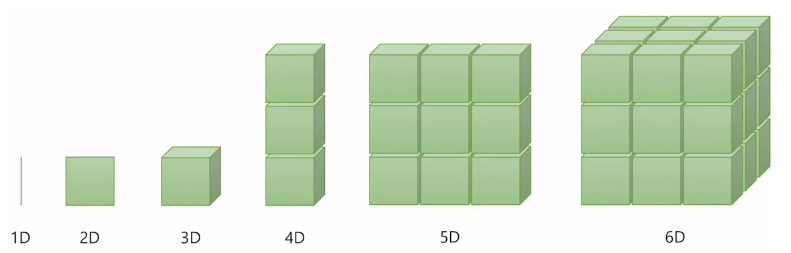
**5. torch.utils.data** SGD의 반복 연산을 실행할 때 사용하는 미니 배치용 유틸리티 함수가 포함되어져 있습니다.

**6. torch.onnx** ONNX(Open Neural Network Exchange)의 포맷으로 모델을 익스포트(export)할 때 사용합니다. ONNX는 서로 다른 딥 러닝 프레임워크 간에 모델을 공유할 때 사용하는 포맷입니다.

Step 1 – Understanding of Tensor

1 – 1 Tensor

Concept of Tensor



딥 러닝을 하게 되면 다루게 되는 가장 기본적인 단위는 벡터, 행렬, 텐서입니다. 차원이 없는 값을 스칼라(위의 그림에는 없음), 1차원으로 구성된 값을 우리는 벡터라고 합니다.

2차원으로 구성된 값을 행렬(Matrix)라고 합니다. 그리고 3차원이 되면 우리는 텐서(Tensor)라고 부릅니다. 사실 우리는 3차원의 세상에 살고 있으므로, 4차원 이상부터는 머리로 생각하기는 어렵습니다. 4차원은 3차원의 텐서를 위로 쌓아 올린 모습으로 상상해보겠습니다.

5차원은 그 4차원을 다시 옆으로 확장한 모습으로 생각해봅시다. 6차원은 5차원을 뒤로 확장한 모습으로 볼 수 있습니다.

* **옮긴이 주 : 데이터사이언스 분야 한정으로 3차원 이상의 텐서는 그냥 다차원 행렬 또는 배열로 간주할 수 있습니다. 또한 주로 3차원 이상을 텐서라고 하긴 하지만, 1차원 벡터나 2차원인 행렬도 텐서라고 표현하기도 합니다. 같은 표현입니다. 벡터 = 1차원 텐서, 2차원 행렬 = 2차원 텐서. 그리고 3차원 텐서, 4차원 텐서, 5차원 텐서 등...**

1 – 1 Example

Input

Import torch

t = torch.FloatTensor([[1., 2., 3.],

[4., 5., 6.],

[7., 8., 9.],

[10., 11., 12.]

])

print(t)

print(t.dim()) # rank (차원)

print(t.size()) # shape

print(t.shape)

Output

tensor([[ 1., 2., 3.],

[ 4., 5., 6.],

[ 7., 8., 9.],

[10., 11., 12.]])

2

torch.Size([4,3])

torch.Size([4,3])

1 – 1 Practice

아래의 Source code를 이용하여 Answer을 출력해보세요.

Import torch

t1 = torch.FloatTensor([[1., 2., 3.],

[4., 5., 6.],

[7., 8., 9.],

[10., 11., 12.]

])

t2= torch.FloatTensor([[1]])

print(t1)

Print(t2)

/return 식으로 변경

Answer

torch.Size([2,2,3])

torch.Size([1,1,1,1])

요구사항: print(“torch.Size([1, 1, 1, 1]”) 를 입력하진 않았는지 검사해 줘야한다.

1 – 2 Tensor Manipulation

Squeeze, Unsqueeze

스퀴즈(Squeeze)는 차원이 1인 경우에는 해당 차원을 제거합니다.

언스퀴즈(Unsqueeze)는 특정 위치에 1인 차원을 추가할 수 있습니다.

1 – 2 Example

input

Import torch

tm = torch.FloatTensor([[1., 2., 3.],

[4., 5., 6.],

[7., 8., 9.],

[10., 11., 12.]

])

tm1 = tm.unsqueeze(0)

print(tm1)

print(tm1.shape)

tm2=tm.squeeze()

print(tm2.shape)

Output

tensor([[[ 1., 2., 3.],

[ 4., 5., 6.],

[ 7., 8., 9.],

[10., 11., 12.]]])

torch.Size([1, 4, 3])

torch.Size([4, 3])

1 – 2 Practice

아래의 Source code에 Squeeze와 Unsqueeze를 사용하여 Answer을 출력해보세요.

Import torch

t1 = torch.FloatTensor([[1., 2., 3.],

[4., 5., 6.],

[7., 8., 9.],

[10., 11., 12.]

])

t2= torch.FloatTensor([[1]])

# 아래에 코드를 작성하세요

print(t1)

Print(t2)

Answer

torch.Size([1, 4, 1, 3])

torch.Size([])

요구사항: print(“Answer”) 을 입력하진 않았는지 + unsqueeze와 squeeze를 사용했는지 검사해 줘야한다.

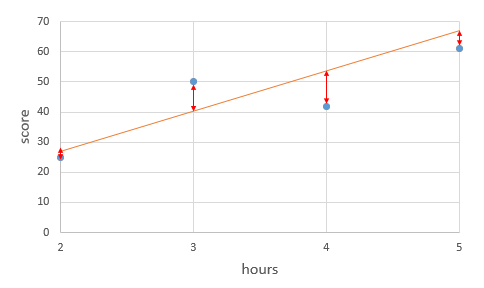
Step 2 – Gradient Descent Algorithm

2 - 1 Linear Regression

**선형 회귀(Linear Regression)**이란 학습 데이터와 가장 잘 맞는 하나의 직선을 찾는 일입니다.

선형 회귀의 가설(직선의 방정식)은 아래와 같은 형식을 가집니다.

y = Wx + b , x와 곱해지는 W를 가중치(Weight)라고 하며, b를 편향(bias)라고 합니다.

어떤 데이터를 이 가설에 대입하면 H(x) = Wx + b 라는 식으로 표현됩니다.

데이터와 직선이 얼마나 잘 맞는지에 대해 수학적인 방법으로 알 수 있습니다.

**비용 함수(cost function)** = **손실 함수(loss function)** = 오차 함수(error function) = 목적 함수(objective function)  
보통 비용 함수와 손실 함수라고 부릅니다.

n개의 ( x, y )의 학습데이터가 있는 상황을 가정했을 때 모든 데이터들은 임의의 직선과 차이인 오차를 가지고 있고, 음수와 양수 모두 존재하기 때문에 수학적인 방법을 통해서 Cost function을 표현해줘야 합니다. 이 값을 평균 제곱 오차(Mean Squared Error)로 정리해주면 아래와 같은 식으로 표현됩니다.

텍스트, 시계, 손목시계이(가) 표시된 사진

자동 생성된 설명

단순하게 생각해보면, cost function의 값이 가장 낮은 직선이 학습데이터와 가장 잘 맞는 직선이라 할 수 있을 것입니다.

2 – 1 Example

Python을 통해, Cost Function을 설정해 보고 Cost를 가장 낮추는 W와 그때의 Cost값을 출력해보세요.

Import torch

x\_train = torch.FloatTensor([[1], [2], [3]])

y\_train = torch.FloatTensor([[2], [4], [6]])

W = 0

b = 0

# H(x)

hypothesis = x\_train \* W + b

# cost

cost = torch.mean((hypothesis – y\_train) \*\* 2 )

print( W, b )

print(cost)

answer

2 0

0

2 – 1 Practice

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | 25 | 50 | 42 | 61 |
| y | 27 | 45 | 70 | 80 |

위 Data에 대해 Linear Regression 방법으로 Hypothesis와 Cost 식을 만들고 W=1, b=2일때의 Cost값을 출력하세요.

input

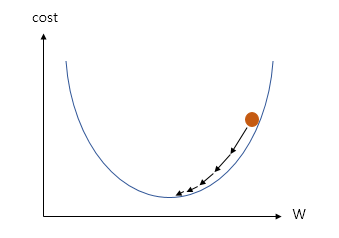
answer

tensor(253.5000)

2 – 2 Gradient Descent

전 단계에서 배운 비용 함수의 값을 최소로 하는 W와 b를 찾는 방법들을 최적화(Optimizer) 알고리즘이라고 부릅니다. 이 옵티마이저 알고리즘을 통해 적절한 W와 b를 찾아내는 과정을 머신 러닝에서 학습(training)이라고 부릅니다. 경사하강법(Gradient Descent)는 가장 기본적인 옵티마이저 알고리즘중 하나입니다.

W와 cost의 관계를 그래프로 표현하면 다음과 같습니다.



Gradient Descent는 cost(W)를 최소로 하는 W를 cost(W)의 기울기를 통해 찾아내는 과정입니다.

위 그림에서 어떤 W값의 cost(W)의 기울기가 양수라면 W의 값이 감소 되어야 하고, 음수라면 W의 값이 증가 해야 합니다. 이를 수식으로 표현하면 다음과 같습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

여기서cost(W)의 기울기 값에 따라 얼마나 W의 값을 변경시켜줄지를 결정하는 α는 학습률(learning rate)이라 합니다.

텍스트, 안테나이(가) 표시된 사진

자동 생성된 설명

위 그림처럼 α 값에 따라 최적의 W값을 찾지 못하는 경우도 있기 때문에 적당한 α값을 설정하는 것이 중요합니다.

Example-1

Step 3 – Data Preprocessing

2 – 1 Loading Data

어떤 방식으로 불러올지 정해야 함

2 – 2 Cleaning datasets

cv2 module을 이용해 data를 학습에 쓰일 형태로 변형 시킬 수 있다.

Append를 이용해 학습에 쓰일 Datasets을 list로 정리할 수 있다.

2 – 2 Example

input

import torch

import cv2

animal\_data=[]

data\_labels=[]

img\_rows = 256

img\_cols = 256

#data loading => files에 있다고 가정

for file in files:

    if '.jpg' in file:

        f = cv2.imread(file, 0)

        f = torch.FloatTensor(cv2.resize(f, (img\_rows, img\_cols)))

    animal\_data.append(f)

    if animal == "cats/":

        data\_labels.append(1)

    else:

        data\_labels.append(0)

print(animal\_data[0])

for i in animal\_data:

    print(i.shape)

for j in data\_labels:

    print(j, end=‘ ‘)

output

tensor([[ 42., 42., 43., ..., 203., 197., 195.],

[ 43., 43., 43., ..., 198., 194., 191.],

[ 43., 42., 42., ..., 193., 192., 193.],

...,

[ 30., 27., 23., ..., 28., 31., 46.],

[ 31., 28., 23., ..., 32., 34., 46.],

[ 31., 27., 22., ..., 44., 36., 33.]])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

torch.Size([256, 256])

1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0

2 – 2 Practice

아래의 Source code를 이용하여 Size([100 ,100, 1])인 Tensor들로 Label을 포함한 Dataset을 만들고, answer를 출력하세요. (Hint : unsqueeze를 이용하세요)

import torch

import cv2

from torch.utils.data import TensorDataset

animal\_data=[]

data\_labels=[]

img\_rows =

img\_cols =

#data loading => files에 있다고 가정

for file in files:

animal\_data=torch.stack(animal\_data)

animal\_data=animal\_data.unsqueeze(1)

data\_labels=torch.LongTensor(data\_labels)

dataset = TensorDataset(animal\_data, data\_labels)

print(dataset[0])

print(dataset[10])

for j in data\_labels:

    print(j, end=‘ ‘)

Answer

(tensor([[[ 42., 42., 43., ..., 203., 197., 195.],

[ 43., 43., 43., ..., 198., 194., 191.],

[ 43., 42., 42., ..., 193., 192., 193.],

...,

[ 30., 27., 23., ..., 28., 31., 46.],

[ 31., 28., 23., ..., 32., 34., 46.],

[ 31., 27., 22., ..., 44., 36., 33.]]]), tensor(1))

(tensor([[[221., 140., 122., ..., 109., 98., 95.],

[222., 134., 114., ..., 61., 56., 57.],

[227., 128., 105., ..., 80., 78., 74.],

...,

[252., 245., 239., ..., 233., 230., 231.],

[255., 250., 245., ..., 237., 236., 236.],

[255., 253., 251., ..., 247., 246., 245.]]]), tensor(0))

tensor(1) tensor(1) tensor(1) tensor(1) tensor(1) tensor(1) tensor(1) tensor(1) tensor(1) tensor(1) tensor(0) tensor(0) tensor(0) tensor(0) tensor(0) tensor(0) tensor(0) tensor(0) tensor(0) tensor(0)

요구사항: print(“Answer”)을 입력하진 않았는지

Step 4 – CNN ( Convolutional Neural Network )

Step 5 – Training Process

References: 위키독스 : wicidocs.net/32471

파이토치 공식 튜토리얼

모두를 위한 딥 러닝 시즌2 Pytorch