실습 환경 설정과 파이토치 기초

2장 실습 환경 설정과 파이토치 기초

2.1 파이토치 개요

2.2 파이토치 기초 문법

2.3 실습 환경 설정

2.4 파이토치 코드 맛보기

2.1 파이토치 개요

2.1 파이토치 개요

- **파이토치 개요**
  - 오픈 소스 머신러닝 라이브러리
  - 토치는 파이썬의 넘파이(NumPy) 라이브러리처럼 과학 연산을 위한 라이브러리로 공개되었지만 이후 발전을 거듭하면서 딥러닝 프레임워크로 발전
  - 파이썬 기반의 과학 연산 패키지로 다음 두 집단을 대상으로 함
    - 넘파이를 대체하면서 GPU를 이용한 연산이 필요한 경우
    - 최대한의 유연성과 속도를 제공하는 딥러닝 연구 플랫폼이 필요한 경우
  - 장: GPU에서 텐서 조작 및 동적 신경망 구축이 가능한 프레임워크

2.1 파이토치 개요

- **파이토치 특징 및 장점 (** GPU, 텐서, 동적 신경망)

1. **GPU**(Graphics Processing Unit): 연산 속도를 빠르게 하는 역할
   - 딥러닝에서는 기울기를 계산할 때 미분을 쓰는데, GPU를 사용하면 빠른 계산이 가능
   - 딥러닝에서는 • 내부적으로 CUDA, cuDNN이라는 API를 통해 GPU를 연산에 사용할 수 있음

2. 텐서(Tensor): 파이토치에서 텐서 의미는 다음과 같음
   - 텐서는 단일 데이터 형식으로 된 자료들의 다차원 행렬
   - 텐서는 간단한 명령어(변수 뒤에 .cuda()를 추가)를 사용해서 GPU로 연산을 수행하게 할 수 있음

3. **동적 신경망**: 훈련을 반복할 때마다 네트워크 변경이 가능한 신경망
   - 예를 들어 학습 중에 은닉층을 추가하거나 제거하는 등 모델의 네트워크 조작이 가능
   - 연산 그래프를 정의하는 것과 동시에 값도 초기화되는 **'Define by Run'** 방식을 사용
   - 연산 그래프와 연산을 분리해서 생각할 필요가 없기 때문에 코드를 이해하기 쉬움

https://sdc-james.gitbook.io/onebook/2.-1/2.4.-pytorch

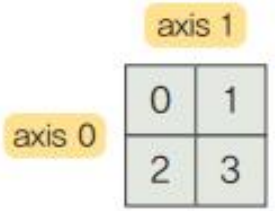
2.1 파이토치 개요

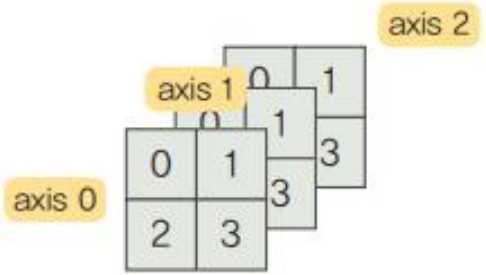
- **파이토치 특징 및 장점**

  벡터, 행렬, 텐서

  - 인공지능(머신 러닝/딥러닝)에서 데이터는 벡터(vector)로 표현
  - 벡터는 [1.0, 1.1, 1.2]처럼 숫자들의 리스트로, 1차원 배열 형태
  - 행렬(matrix)은 행과 열로 표현되는 2차원 배열 형태
  - 텐서는 3차원 이상의 배열 형태

    - 1차원 축(행)=axis 0=벡터
    - 2차원 축(열)=axis 1=행렬
    - 3차원 축(채널)=axis 2=텐서





2.1 파이토치 개요

- **파이토치 특징 및 장점**
  - 파이토치에서 텐서를 표현하기 위해서는 다음 코드와 같이 torch.tensor()를 사용

```
import torch
torch.tensor([[1., -1.], [1., -1.]])
```

  - 생성된 텐서의 형태는 다음과 같이 표현

```
tensor([[ 1., -1.],
        [ 1., -1.]])
```

  - 벡터, 행렬 등 자세한 내용은 선형대수학 도서를 참고

```
import torch
torch.tensor(([1.,-1],[1.,-1.]))
```

```
tensor([[ 1., -1.],
        [ 1., -1.]])
```

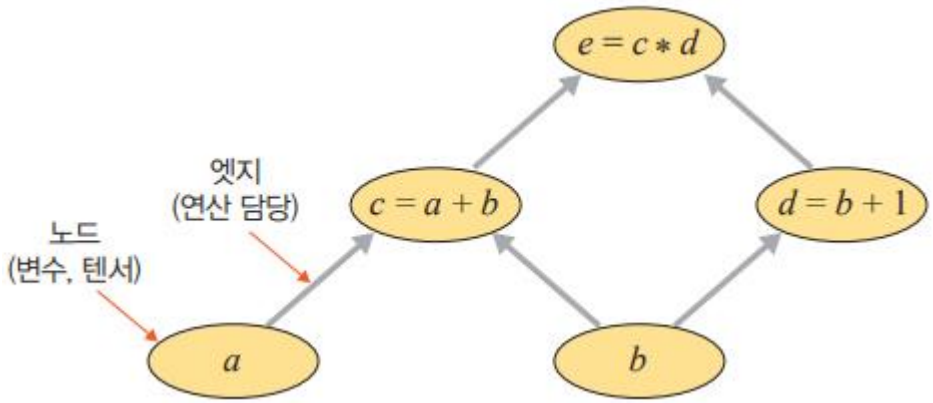
2.1 파이토치 개요

- **파이토치 특징 및 장점**

  연산 그래프

  - 연산 그래프는 방향성이 있으며 변수(예 텐서)를 의미하는 노드와 연산(곱하기, 더하기)을 담당하는 엣지로 구성
  - 다음 그림과 같이 노드는 변수(a, b)를 가지고 있으며 각 계산을 통해 새로운 텐서(c, d, e)를 구성할 수 있음

▼ **그림 2-3 파이토치 연산 그래프**

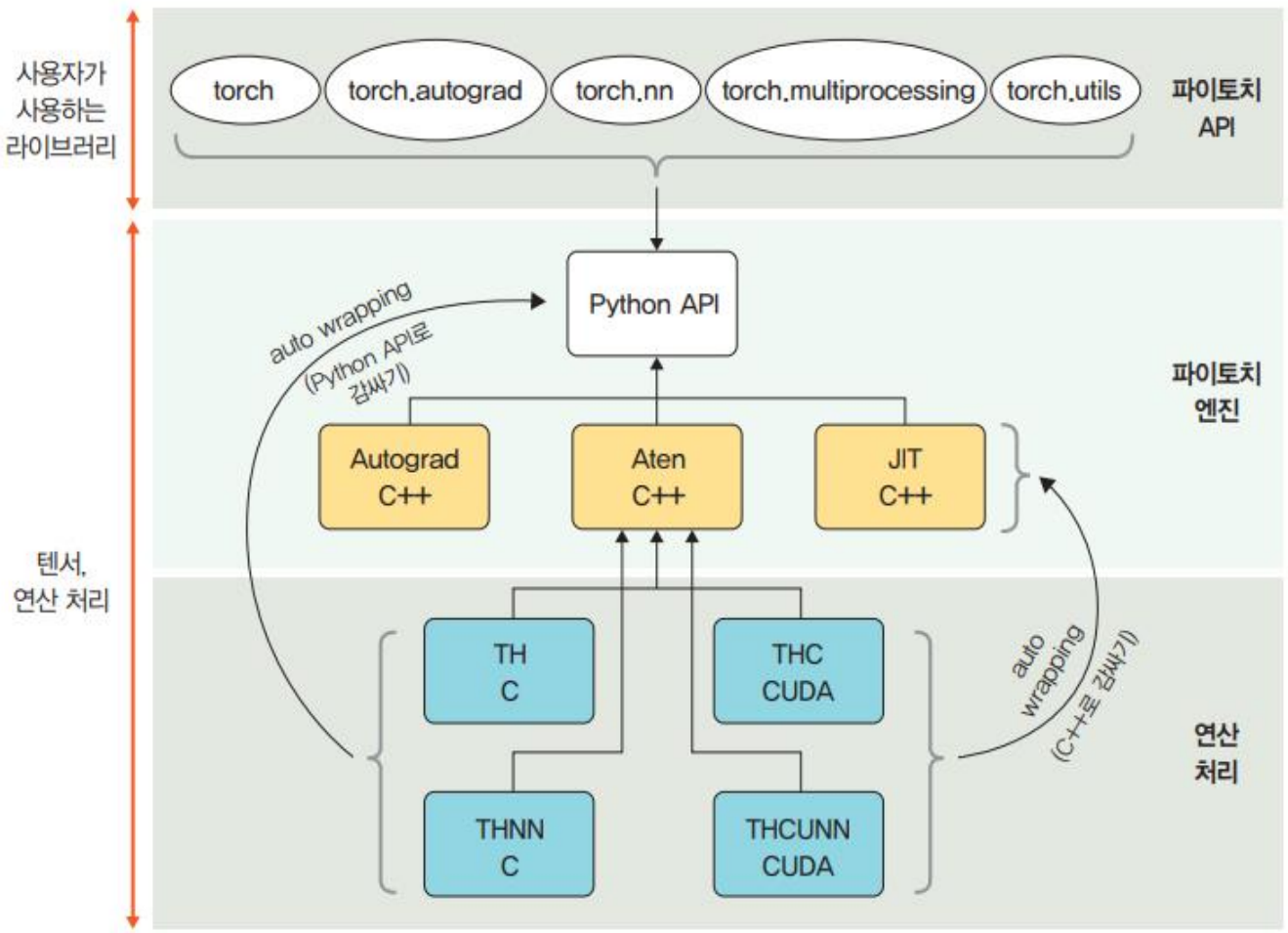


2.1 파이토치 개요

- **파이토치의 아키텍처**
  - 크게 세 개의 계층으로 나누어 설명할 수 있음. 가장 상위 계층은 파이토치 API가 위치해 있으며 그 아래에는 파이토치 엔진이 있음
  - 파이토치 엔진에서는 다차원 텐서 및 자동 미분을 처리
  - 마지막으로 가장 아래에는 텐서에 대한 연산을 처리
  - CPU/GPU를 이용하는 텐서의 실질적인 계산을 위한 C, CUDA 등 라이브러리가 위치

2.1 파이토치 개요

▼ 그림 2-4 파이토치의 아키텍처



2.1 파이토치 개요

- **파이토치의 아키텍처**

  파이토치 API

  - 파이토치 API 계층에서는 사용자가 이해하기 쉬운 API를 제공하여 텐서에 대한 처리와 신경망을 구축하고 훈련할 수 있도록 도움
  - 이 계층에서는 사용자 인터페이스를 제공하지만 실제 계산은 수행하지 않음
  - 그 대신 C++로 작성된 파이토치 엔진으로 그 작업을 전달하는 역할만 함
  - 파이토치 API 계층에서는 사용자의 편의성을 위해 다음 패키지들이 제공

2.1 파이토치 개요

- **파이토치의 아키텍처**

  torch: GPU를 지원하는 텐서 패키지

  - 다차원 텐서를 기반으로 다양한 수학적 연산이 가능하도록 함
  - CPU뿐만 아니라 GPU에서 연산이 가능하므로 빠른 속도로 많은 양의 계산을 할 수 있음

  torch.autograd: 자동 미분 패키지

  torch.nn: 신경망 구축 및 훈련 패키지

  torch.multiprocessing: 파이썬 멀티프로세싱 패키지

  torch.utils: DataLoader 및 기타 유틸리티를 제공하는 패키지

2.1 파이토치 개요

- **파이토치의 아키텍처**

  텐서를 메모리에 저장하기

  - 텐서는 1차원 배열 형태여야만 메모리에 저장할 수 있음
  - 변환된 1차원 배열을 스토리지(storage)라고 함

  • 오프셋(offset): 텐서에서 첫 번째 요소가 스토리지에 저장된 인덱스

  • 스트라이드(stride): 각 차원에 따라 다음 요소를 얻기 위해 건너뛰기(skip)가 필요한 스토리지의 요소 개수

  즉, 스트라이드는 메모리에서의 텐서 레이아웃을 표현하는 것으로 이해하면 됨

  요소가 연속적으로 저장되기 때문에 행 중심으로 스트라이드는 항상 1

2.2 파이토치 기초 문법

2.2 파이토치 기초 문법

- **텐서 다루기**

  텐서 생성 및 변환

  - 텐서는 파이토치의 가장 기본이 되는 데이터 구조
  - 넘파이의 ndarray와 비슷하며 GPU에서의 연산도 가능
  - 텐서 생성은 다음과 같은 코드를 이용

```
import torch
print(torch.tensor([[1,2],[3,4]])) ------ 2차원 형태의 텐서 생성
print(torch.tensor([[1,2],[3,4]], device="cuda:0")) ------ GPU에 텐서 생성
print(torch.tensor([[1,2],[3,4]], dtype=torch.float64)) ------ dtype을 이용하여 텐서 생성
```

```
#텐서 생성 및 변환
import torch
print(torch.tensor([[1,2],[3,4]]))
print(torch.tensor([[1,2],[3,4]],device="cuda:0"))
print(torch.tensor([[1,2],[3,4]],dtype=torch.float64))
```

2.2 파이토치 기초 문법

```
#텐서 생성 및 변환
```

```
tensor([[1, 2],
        [3, 4]])
tensor([[1, 2],
        [3, 4]], device='cuda:0')
tensor([[1., 2.],
        [3., 4.]], dtype=torch.float64)
```

런타임>런타임 유형 변경>하드웨어 가속기

2.2 파이토치 기초 문법

- **텐서 다루기**

  텐서의 인덱스 조작

  - 텐서의 인덱스를 조작하는 방법은 여러 가지가 있음
  - 텐서는 넘파이의 ndarray를 조작하는 것과 유사하게 동작하기 때문에 배열처럼 인덱스를 바로 지정하거나 슬라이스 등을 사용할 수 있음
  - 텐서의 자료형은 다음과 같음

    - **torch.FloatTensor**: 32비트의 부동 소수점
    - **torch.DoubleTensor**: 64비트의 부동 소수점
    - **torch.LongTensor**: 64비트의 부호가 있는 정수

2.2 파이토치 기초 문법

- **텐서 다루기**
  - 텐서의 인덱스 조작은 다음과 같은 코드를 이용

```
temp = torch.FloatTensor([1, 2, 3, 4, 5, 6, 7]) ------ 파이토치로 1차원 벡터 생성
print(temp[0], temp[1], temp[-1]) ------ 인덱스로 접근
print('----------------------')
print(temp[2:5], temp[4:-1]) ------ 슬라이스로 접근
```

```
# 텐서 연산
temp=torch.FloatTensor([1,2,3,4,5,6,7])
print(temp[0],temp[1],temp[-1])
print('----------------------')
print(temp[2:5],temp[4:-1])
#
v=torch.tensor([1,2,3])
w=torch.tensor([3,4,6])
print(w-v)
```

```
tensor(1.) tensor(2.) tensor(7.)
----------------------
tensor([3., 4., 5.]) tensor([5., 6.])
tensor([2, 2, 3])
```

2.2 파이토치 기초 문법

- **텐서 다루기**
  - 텐서의 차원에 대한 문제는 신경망에서 자주 다루어지므로 상당히 중요함
  - 텐서의 차원을 변경하는 가장 대표적인 방법은 view를 이용하는 것
  - 이외에도 텐서를 결합하는 stack, cat과 차원을 교환하는 t, transpose도 사용
  - view는 넘파이의 reshape과 유사하며 cat은 다른 길이의 텐서를 하나로 병합할 때 사용
  - 또한, transpose는 행렬의 전치 외에도 차원의 순서를 변경할 때도 사용

2.2 파이토치 기초 문법

- **텐서 다루기**
  - 텐서의 차원을 조작하는 코드는 다음과 같음

```
temp = torch.tensor([
    [1, 2], [3, 4]]) ------ 2×2 행렬 생성

print(temp.shape)
print('----------------------')
print(temp.view(4, 1)) ------ 2×2 행렬을 4×1로 변형
print('----------------------')
print(temp.view(-1)) ------ 2×2 행렬을 1차원 벡터로 변형
print('----------------------')
print(temp.view(1, -1)) ---------- -1은 (1, ?)와 같은 의미로 다른 차원으로부터 해당 값을 유추하겠다는 것입니다. temp의 원소 개수(2×2=4)를 유지한 채 (1, ?)의 형태를 만족해야 하므로 (1, 4)가 됩니다.
print('----------------------')
print(temp.view(-1, 1)) ------ 앞에서와 마찬가지로 (?, 1)의 의미로 temp의 원소 개수(2×2=4)를 유지한 채 (?, 1)의 형태를 만족해야 하므로 (4, 1)이 됩니다.
```

2.2 파이토치 기초 문법

- **텐서 다루기**
  - 다음은 텐서의 차원을 조작한 결과

```
torch.Size([2, 2])
----------------------
tensor([[1],
        [2],
        [3],
        [4]])
----------------------
tensor([1, 2, 3, 4])
----------------------
tensor([[1, 2, 3, 4]])
----------------------
tensor([[1],
        [2],
        [3],
        [4]])
```

2.3 실습 환경 설정

- **가상 환경 생성 및 파이토치 설치**

  코랩일 경우

  - 구글 코랩(Colab)은 클라우드 기반의 무료 주피터 노트북 개발 환경
  - 코랩은 구글 드라이브, 도커, 리눅스, 구글 클라우드로 구성되어 있음
  - 구글 코랩을 사용하는 이유는 사용하는 PC의 성능 한계 때문임
  - 일반적으로 딥러닝 모델을 이용하여 데이터를 분석할 때는 대량의 데이터를 다룸
  - 이때 고성능 PC(혹은 서버)가 필요한데, 고성능 PC 환경을 개인이 갖추기는 어렵기 때문에 코랩을 많이 사용하는 추세

감사합니다.