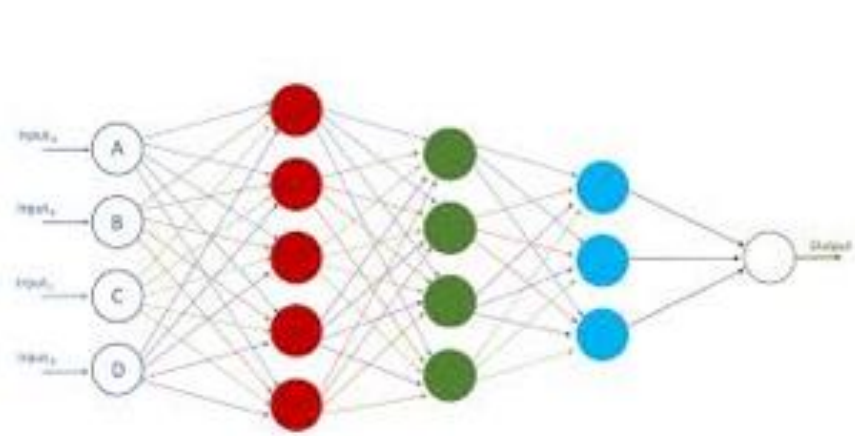
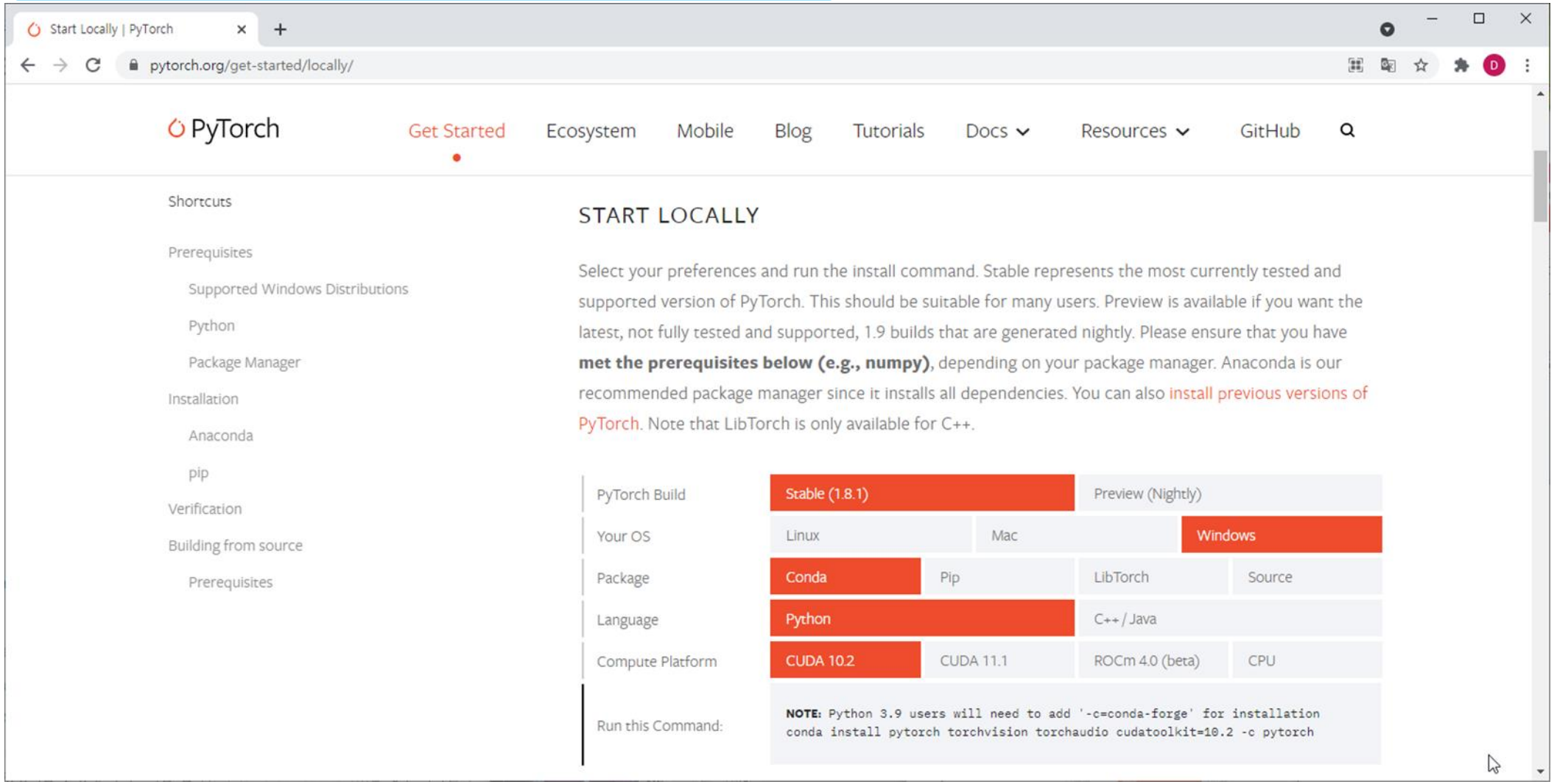


PyTorch

박경규

파이토치(PyTorch) 설치

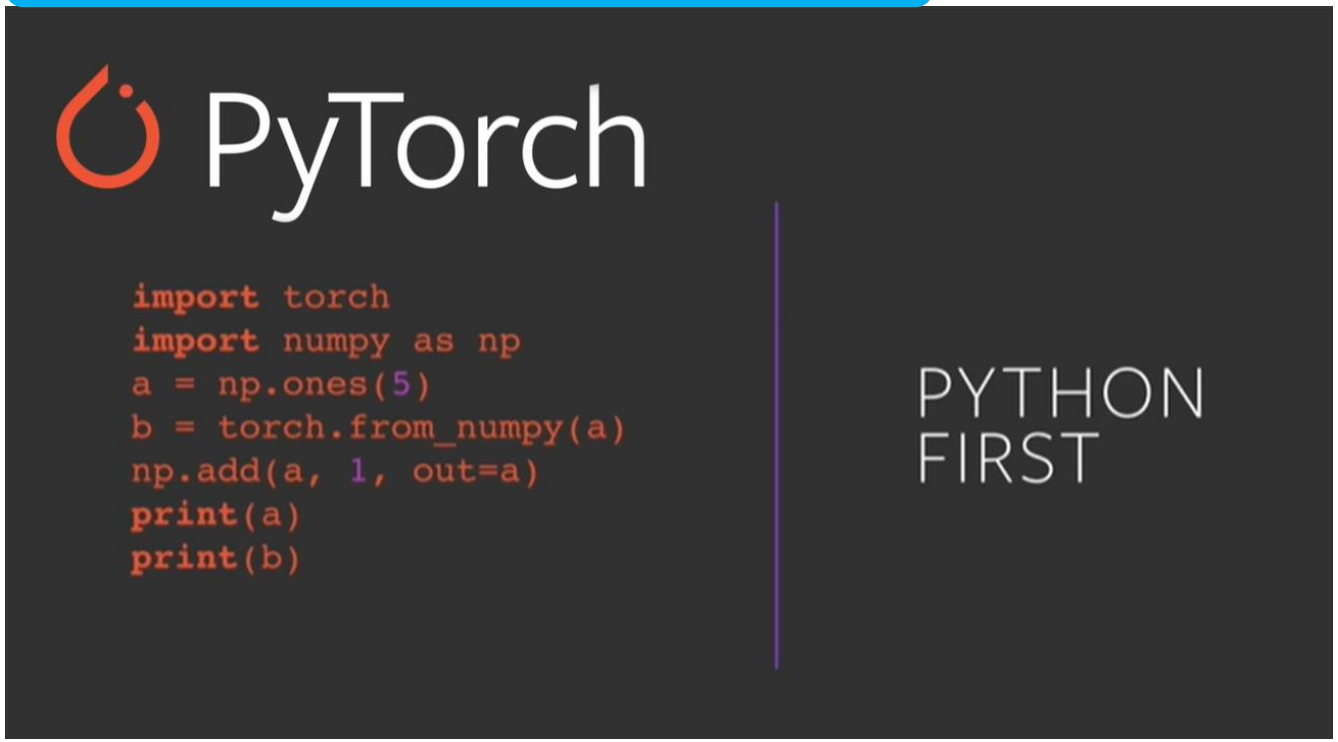
https://pytorch.org/get-started/locally/



파이토치(PyTorch)

딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공해주는 프레임워크입니다.

https://tutorials.pytorch.kr/



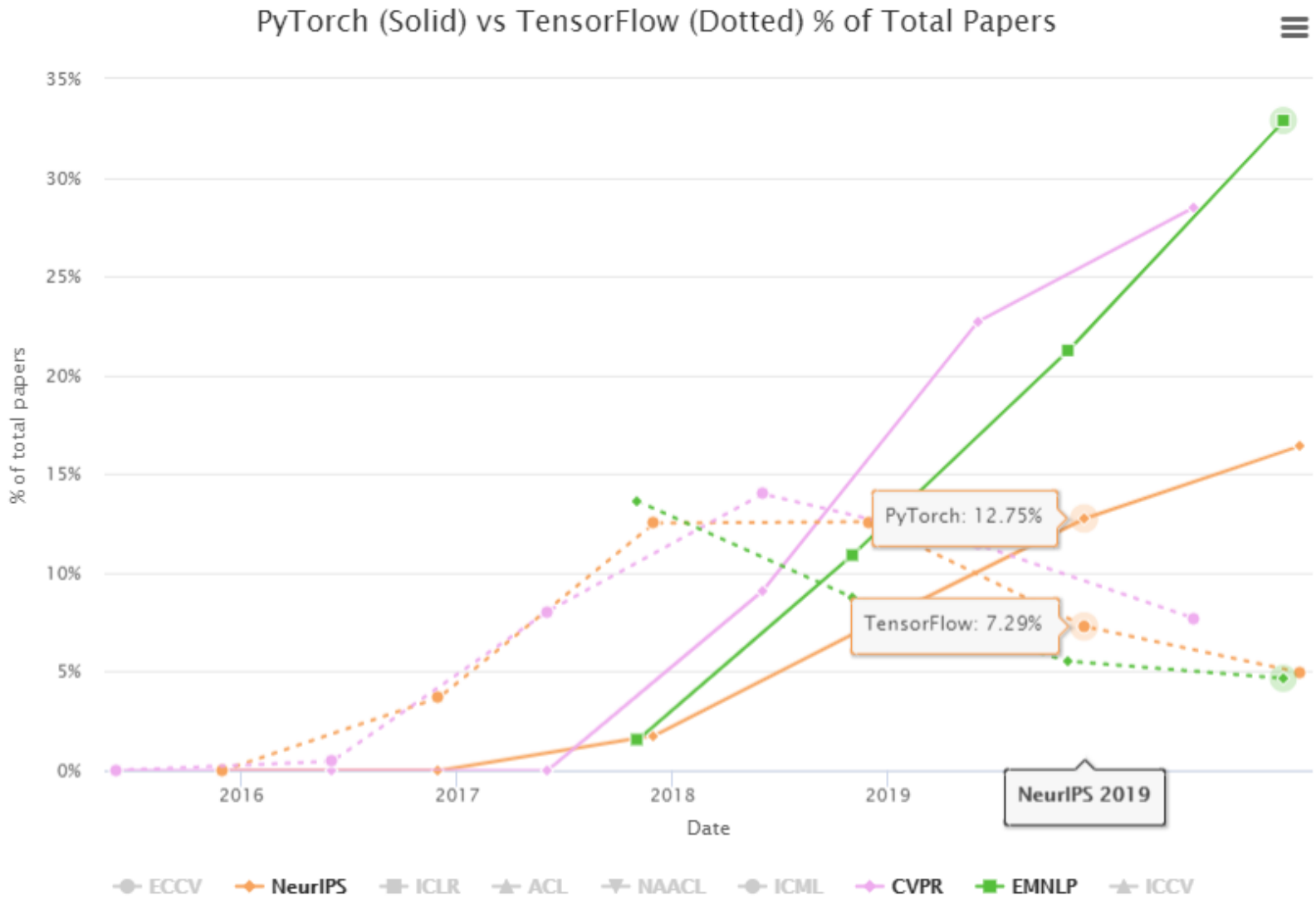
Pythonic

Great API

Dynamic Graph

PyTorch로 시작하는 딥러닝 입문 : https://wikidocs.net/book/2788

파이토치(PyTorch)



참고 : https://data-newbie.tistory.com/425

파이토치(PyTorch) 패키지

- torch : 메인 네임스페이스. 텐서 등의 다양한 수학 함수를 제공
  https://pytorch.org/docs/stable/torch.html

- torch.autograd : 자동 미분을 위한 함수들을 포함
  https://pytorch.org/docs/stable/autograd.html

- torch.nn : 신경망을 구축하기 위한 다양한 데이터 구조나 레이어 등을 정의
  - 레이어 : CNN, RNN, LSTM 등
  - 활성화 함수: ReLU, Sigmoid 등
  - 손실 함수: MSELoss, CrossEntropyLoss 등
  https://pytorch.org/docs/stable/nn.html

- torch.optim : 파라미터 최적화 알고리즘, SGD, Adam 등
  https://pytorch.org/docs/stable/optim.html

- torch.utils.data : 미니 배치용 유틸리티 함수 포함
  https://pytorch.org/docs/stable/data.html

파이토치 텐서(Tensor)

```
import torch

#  일반적인 파이썬 변수 사용 예
x = 3.0
y = x*x + 1
print(x, y)
```

```
3.0 10.0
```

```
# 파이토치 텐서
x = torch.tensor(3.0)
print(x)
```

```
tensor(3.)
```

```
# 텐서를 이용한 간단한 연산
y = x + 1
print(y)
```

```
tensor(4.)
```

자동기울기 계산(AUTOGRAD)

autograd 패키지는 Tensor의 모든 연산에 대해 자동 미분을 제공합니다.

```
x = torch.tensor(3.0, requires_grad=True)
print(x)
```

```
tensor(3., requires_grad=True)
```

```
y = (x-1)*(x-2)*(x-3)
print(y)
```

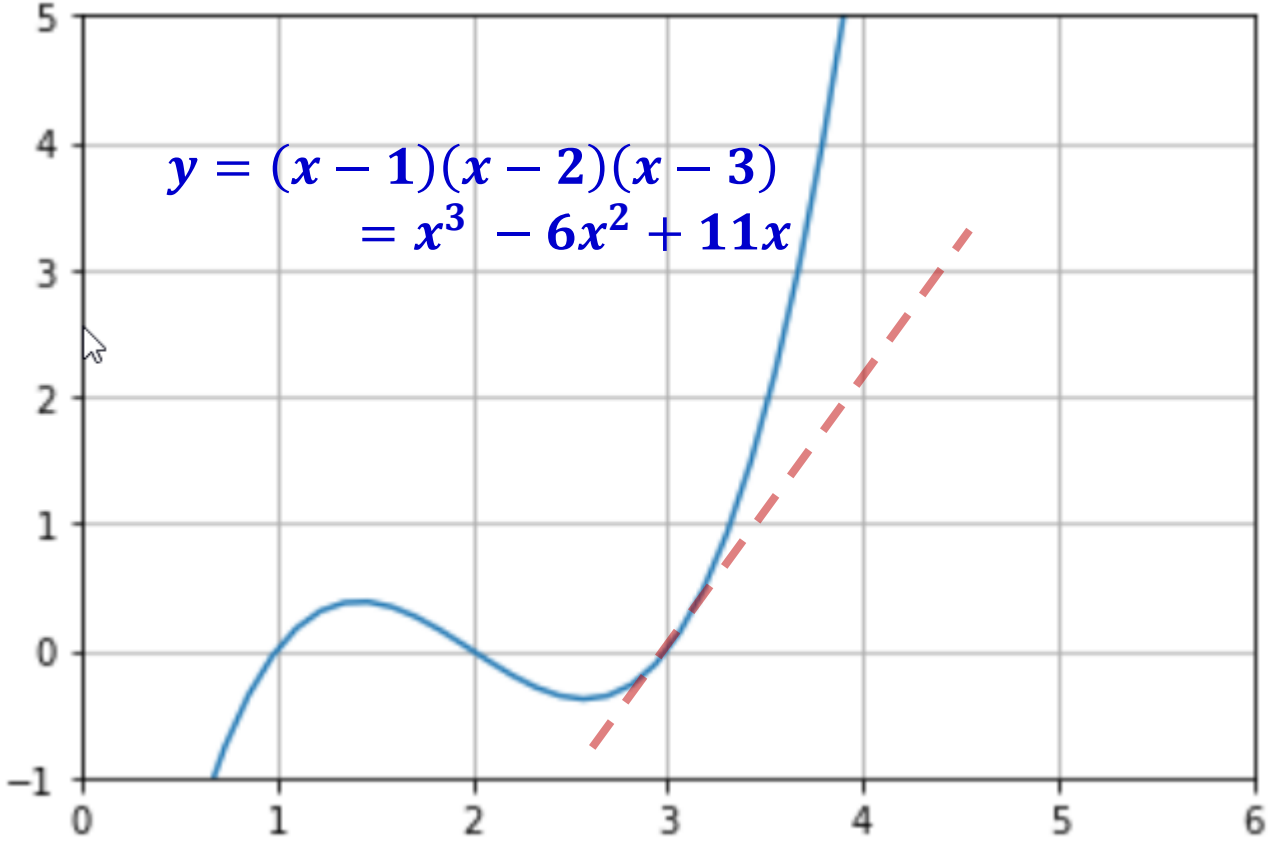
```
tensor(0., grad_fn=<MulBackward0>)
```

```
# 기울기 계산
y.backward()

# x = 3.0일 때의 기울기
print(x.grad)
```

```
tensor(2.)
```

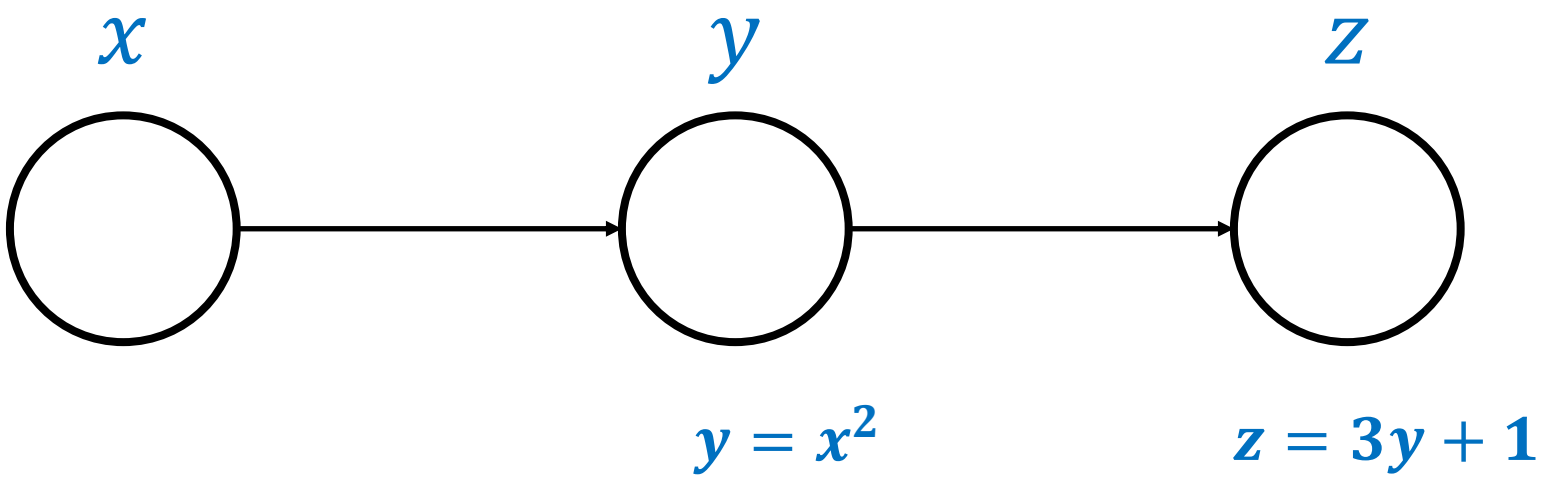
자동기울기 계산(AUTOGRAD)

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 6)
y = (x-1)*(x-2)*(x-3)
plt.plot(x, y)
plt.xlim(0, 6)
plt.ylim(-1, 5)
plt.grid(True)
plt.show()
```



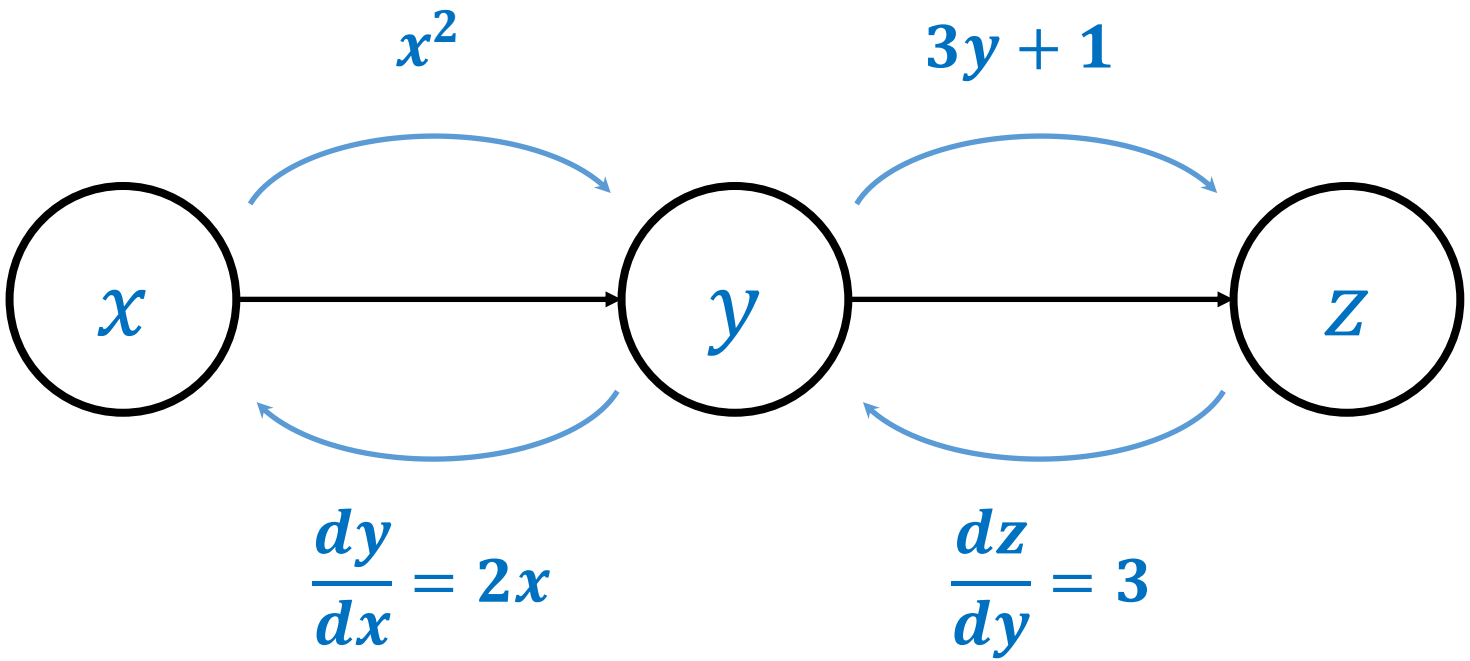
$$\frac{dy}{dx} = 3x^2 - 12x + 11$$

계산그래프(Computational Graph)



$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = 3 \cdot 2x = 6x$$

계산그래프(Computational Graph)



계산그래프(Computational Graph)

```
# x, y, z에 관련된 간단한 그래프 설정
x = torch.tensor(2.3, requires_grad=True)
y = x*x
z = 3*y + 1

# 기울기 계산
z.backward()

# 기울기 dz/dx는 x.grad에 저장됩니다.
print(x.grad)
```

```
tensor(13.8000)
```

CustomDataset

CustomDataset.ipynb

- 파이토치에서는 데이터셋을 좀 더 쉽게 다룰 수 있도록 유용한 도구로서 torch.utils.data.Dataset과 torch.utils.data.DataLoader를 제공합니다.
- 이를 사용하면 미니 배치 학습, 데이터 셔플(shuffle), 병렬 처리까지 간단히 수행할 수 있습니다.
- 기본적인 사용 방법은 Dataset을 정의하고, 이를 DataLoader에 전달하는 것입니다.

DataLoader

DataLoader.ipynb

- PyTorch의 Dataset과 DataLoader를 이용하면 학습을 위한 방대한 데이터를 미니배치 단위로 처리할 수 있고, 데이터를 무작위로 섞음으로써 학습의 효율성을 향상시킬 수 있습니다.
- 여러개의 GPU를 사용해 데이터를 병렬처리하여 학습할 수 있습니다.

파이토치(PyTorch) 모델링 절차

SimpleNeuralNetwork.ipynb

1. nn.Module 클래스를 상속받아 모델 아키텍처 클래스 선언
2. 모델 클래스 객체 생성
3. SGD 또는 Adam 등의 옵티마이저를 생성하고,
   생성한 모델의 파라미터를 최적화 대상으로 등록
4. 데이터로 미니배치를 구성하여 피드포워드 연산 그래프 생성
5. 손실 함수를 통해 최종 결과값과 손실값 계산
6. 손실에 대해 backward() 호출 -> 연산 그래프 상의 텐서들의 기울기(gradient)가 채워짐
7. 3번의 옵티마이저에서 step()을 호출하여 경사하강법 1스텝 수행
8. 4번으로 돌아가 수렴조건이 만족할 때까지 반복 수행

PyTorch vs TensorFlow2

https://data-newbie.tistory.com/425

https://jeongukjae.github.io/posts/pingpong-torch-to-tf-tf-to-torch/

https://www.machinelearningplus.com/deep-learning/tensorflow1-vs-tensorflow2-vs-pytorch/

```python
# Representative Code in PyTorch
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d()
        self.conv2 = nn.Conv2d()

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

```python
# Equivalent Representative Code in TensorFlow2.0
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, Model

class TFModel(Model):
  def __init__(self):
    super(TFModel, self).__init__()
    self.conv1 = layers.Conv2D()
    self.conv1 = layers.Conv2D()

  def call(self, x):
    x = layers.ReLU(self.conv1(x))
    return layers.ReLU(self.conv2(x))
```

PyTorch vs TensorFlow2

https://www.machinelearningplus.com/deep-learning/tensorflow1-vs-tensorflow2-vs-pytorch/

TensorFlow1.x	PyTorch	TensorFlow2.0
Only static computation graphs supported	Only dynamic computation graphs supported	Both static and dynamic computation graphs supported
There is a need to use `tf.session` for separation from Python	PyTorch is tightly integrated with python	No requirement of initialising sessions as only functions are used
Low level APIs are used but support for high level APIs is available	REST API is used along with Flask for deployment	Keras API, which is also a high level API, is used for deployment

Thank you