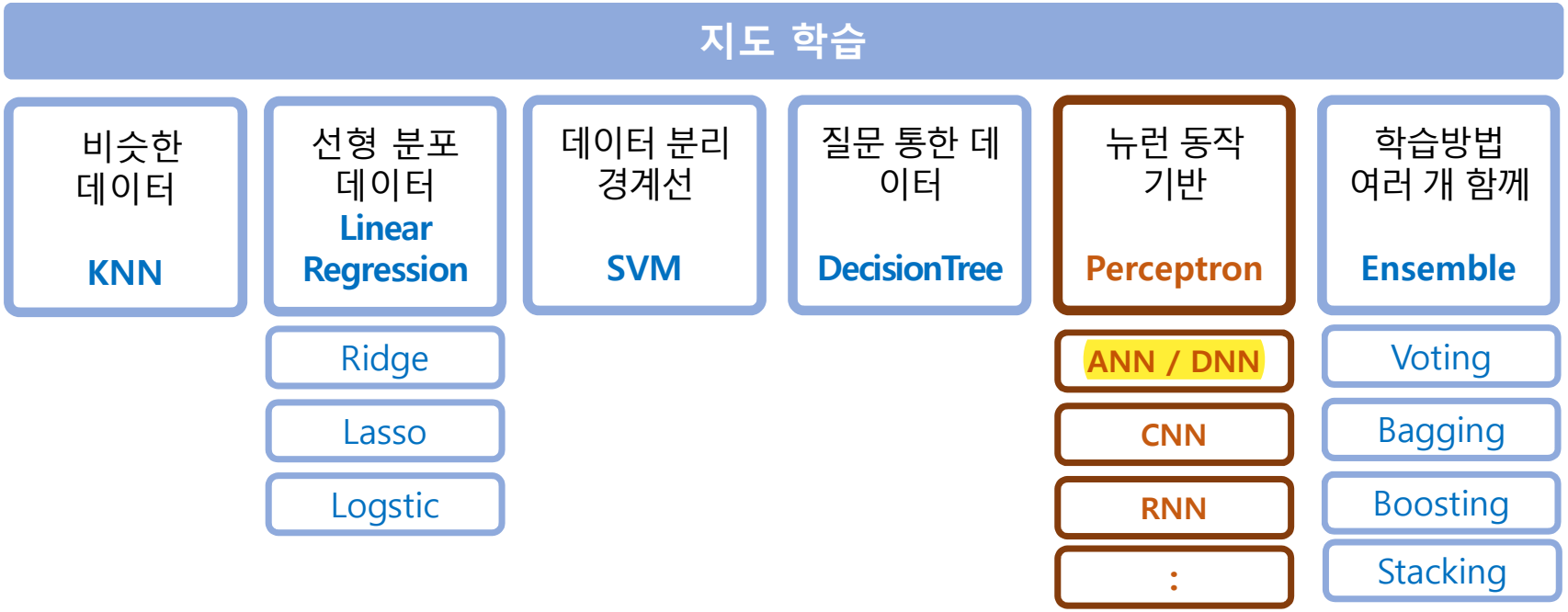
DEEP LEARNING WITH PYTORCH

ABOUT DATASET & DATALOADER

DATASET & LOADER

◆ ML 기계학습

데이터가 가진 패턴/규칙을 찾아서 수식화 하는 컴퓨터프로그래밍



DATASET & LOADER

◆ ANN 인공신경망

기계학습(ML) 중 **대량의 데이터를 학습하여 규칙/패턴** 찾아내는 학습 방법

대량의 데이터

학습

HW - 많은 메모리 필요

SW - 많은 연산 발생

PS - 많은 시간 필요

DATASET & LOADER

◆ DATASET & DATALODER

❖ Pytorch 데이터셋 처리 정책

- 데이터 처리의 유지보수 및 가독성, 모듈성 위해 **데이터셋 전담 기능 제공 CLASS 제공**
- **피쳐와 라벨을 텐서화** 시키고 **하나의 묶음으로 관리**
- **DataLoader를 통해서 데이터가 추출**

- Pytorch 제공 내장 데이터셋
- 사용자 데이터셋 처리 위한 커스텀 데이터셋 ➔ 개발자가 생성

DATASET & LOADER

◆ DATASET & DATALODER

❖ 관련 모듈들

- torch.utils.data.**DataLoader** ← 데이터셋에서 지정된 개수만큼의 인덱스 추출
- torch.utils.data.**Dataset** ← 데이터셋 부모클래스
  커스텀 데이터셋 생성 시 사용

- 내장 데이터셋
  - torchvision.datasets ← 이미지 내장 데이터셋
  - torchtext.datasets ← 텍스트 내장 데이터셋

ABOUT DATASET

◆ DATASET

- 데이터 처리의 유지보수 및 가독성, 모듈성 위해 데이터셋 코드 분리
- 사용자 데이터셋을 위한 **커스텀 데이터셋 준비 필요**

- 관련 모듈

  → torch.utils.data.**DataLoader**

  → torch.utils.data.**Dataset**

ABOUT DATASET

◆ torch.utils.data.Dataset

❖ 미완성 추상 클래스(abstract class)

❖ 사용자 데이터의 Dataset 클래스 생성 시 부모(Super) 클래스

❖ 자식 클래스에서는 3개의 필수 메서드 오버라이딩(overriding)해야 함!

【 필수 오버라이딩 메서드 】

- def _ _init_ _(self) : 데이터셋 전처리 및 초기화 진행 후 Dataset 인스턴스 생성
- def _ _len_ _(self) : 데이터셋 길이 즉 샘플 수 반환 메서드
- def _ _getitem_ _(self, idx) : idx 해당 데이터와 라벨 반환 메서드

ABOUT DATASET

◆ DATASET - 커스텀 데이터셋 생성

```
from torch.utils.data import Dataset

class CustomDataset(Dataset):
    def __init__(self, 매개변수1, 매개변수n):
    def __len__(self, 매개변수1, 매개변수n):
    def __getitem__(self, idx):
```

- 데이터셋 전처리 / 초기화 진행 메서드
- 데이터셋 길이 / 샘플 수 반환 메서드
- idx 해당 데이터 1개 반환 메서드

ABOUT DATASET

◆ DATASET - 사용자 정의 일반 데이터

```
class CustomDataset(Dataset):
    # 데이터 초기화
    def __init__(self):
        self.x_data = [ [73, 80, 75],  [93, 88, 93], [89, 91, 90], [96, 98, 100], [73, 66, 70] ]
        self.y_data = [[152], [185], [180], [196], [142]]

    # 총 데이터의 개수 리턴
    def __len__(self):    return len(self.x_data)

    # 인덱스에 해당하는 입출력 데이터를 파이토치의 Tensor 형태로 리턴
    def __getitem__(self, idx):
        x = torch.FloatTensor(self.x_data[idx])
        y = torch.FloatTensor(self.y_data[idx])
        return x, y
```

ABOUT DATASET

◆ DATASET - 사용자 정의 일반 데이터

```
class CustomDataset(Dataset):
      # 데이터 초기화
       def__init__(self, file_path):
              df = pd.read_csv(file_path)
              self.x = df.iloc[:, 0].values
              self.y = df.iloc[:, 1].values
              self.length = len(df)

       # 인덱스에 해당하는 입출력 데이터를 파이토치의 Tensor 형태로 리턴
       def__getitem__(self, index):
              x = torch.FloatTensor([self.x[index] ** 2, self.x[index]])
              y = torch.FloatTensor([self.y[index]])
              return x, y

       # 총 데이터의 개수 리턴
      def_len_(self):  return self.length
```

ABOUT DATASET

◆ DATASET - 사용자 정의 일반 데이터

❖ 커스텀 데이터셋 생성 및 로딩

```
# 커스텀 데이터셋 인스턴스 생성
train_dataset = CustomDataset("./datas.csv")


# 배치크기만큼 데이터 로딩
train_dataloader = DataLoader( train_dataset,
                                batch_size=3,
                               shuffle=True,
                               drop_last=True )
```

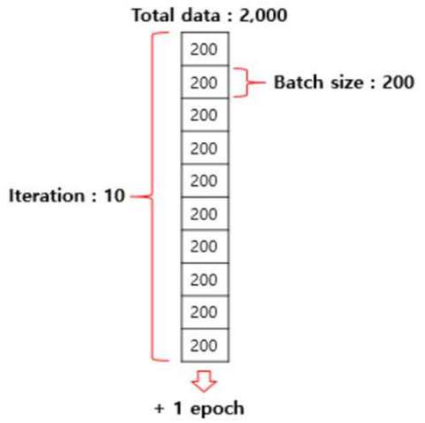
ABOUT DATALOADER

◆ DATALOADER

- PyTorch 데이터 로딩 유틸리티 Class
- 데이터셋에서 지정된 크기(batch_size)만큼 랜덤하게 인덱스 추출하는 Generator

- **에포크(epochs)** : 처음부터 끝까지 학습하는 횟수
- **배치크기(batch size)** : 전체 데이터를 작은 단위로 나눈 크기
  2의 제곱수 크기
- **이터레이션(iteration)** : 에포크, 배치크기로 계산한 반복 횟수
  W,b 업데이터 횟수

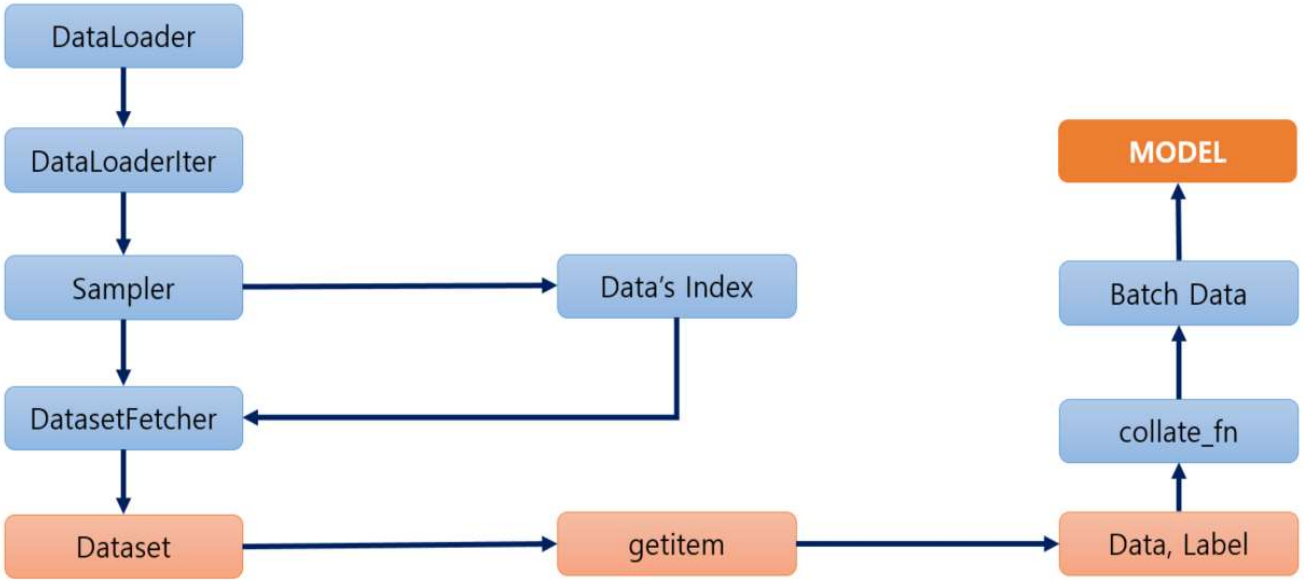
- 예) 100개 데이터, 배치크기 20개, 에포크 10번



ABOUT DATALOADER

◆ DATALOADER

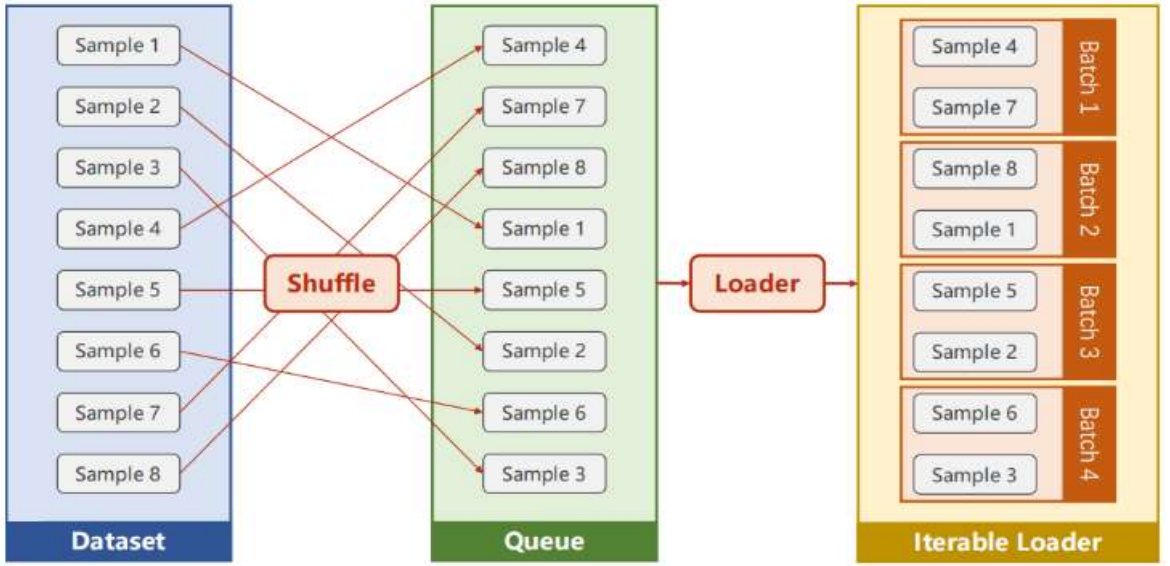
❖ 동작 구조



ABOUT DATALOADER

◆ DATALOADER

❖ 동작구조 : **shuffle = True**



ABOUT DATALOADER

◆ DATALOADER

❖ 클래스 매개변수

```
DataLoader(  dataset,
             batch_size=1,
             shuffle=False,          # 데이터셋 무작위 섞을지 여부 결정, True면 에포크단위
             sampler=None,           # 데이터셋에서 데이터 샘플링에 사용 객체
             batch_sampler=None,     # 배치 단위 데이터 샘플링에 사용 객체, batch_size 지정 X
             num_workers=0,
             collate_fn=None,        # 배치 단위로 데이터 처리 함수, 불러온 데이터 그대로 반환
             pin_memory=False,       # CUDA 호환 GPU 메모리에 고정
```

ABOUT DATALOADER

◆ DATALOADER

❖ 클래스 매개변수

```
        drop_last=False,                    # 마지막 배치(batch)가 batch_size보다 작을 경우 drop 여부
        timeout=0,
        prefetch_factor (int, optional):    # 미리 읽어올 데이터의 배치(batch) 수 결정
                                            # 다음 배치(batch) 데이터를 불러와서 메모리에 적재
                                            # 데이터를 불러오는 속도를 높일 수 있음
                                            # 값을  높이면 미리 읽어올 데이터 양 증가/ 속도 빨라짐
        worker_init_fn=None
)
```

ABOUT DATALOADER

◆ DATALOADER

```
# 피쳐+라벨=> 데이터셋
dataset = TensorDataset(x_train, y_train)
print(f'dataset => {type(dataset)}₩n{dataset.tensors[0].shape}')


# 데이터셋을 로딩하는 방식 설정
# 배치 크기(Batch Size), 데이터 순서 변경(Shuffle)
# 로드 프로세스 수(num_workers), 마지막 배치 데이터 제거(drop_last)
dataloader = DataLoader(dataset, batch_size=6, shuffle=True)
print(f'dataloader => {type(dataloader)}₩nLen : {len(dataloader)}')
print(f'batch_size : {dataloader.batch_size}')
```

ABOUT TORCHMETRICS

ABOUT TORCHMETRICS

◆ TorchMetrics

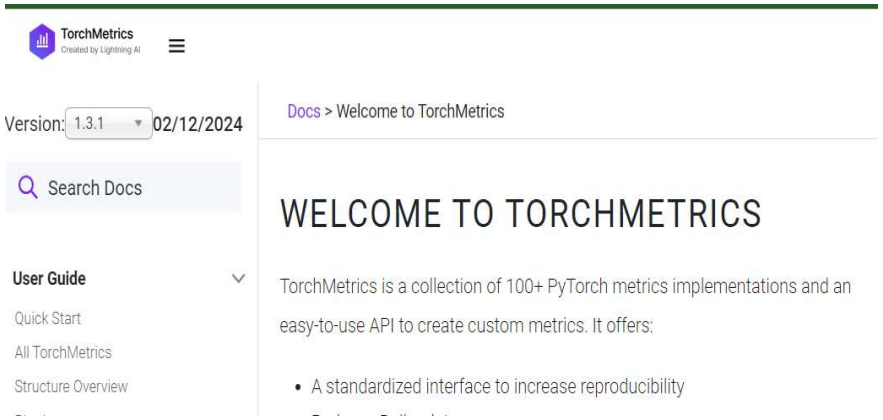
- PyTorch에서 사용 가능한 다양한 메트릭 함수 제공 패키지
- 메트릭 컬렉션을 제공하여 여러 메트릭 함수를 한번에 사용 가능
- GPU를 사용하여 빠른 속도로 메트릭을 계산할 수 있음
- 분산 학습 호환 및 자동 동기화
- metrics 계산 시 배치 단위로 누적 연산이 가능함

ABOUT TORCHMETRICS

◆ TorchMetrics

❖ 설치

```
pip install torchmetrics
```



ABOUT TORCHMETRICS

◆ TorchMetrics - 함수 활용

```
import torch
import torchmetrics.functional as metrics

preds  = torch.randn(10, 5).softmax(dim=-1)
target = torch.randint(5, (10,))

acc = metrics.accuracy( preds, target, task="multiclass", num_classes=5 )
print(f'ACC : {acc}')
```

ABOUT TORCHMETRICS

◆ TorchMetrics - 모듈 활용

```
# 모듈 로딩
import torch
from torchmetrics.classification import Accuracy

# 정확도 인스턴스 생성
metric = Accuracy( task="multiclass",
                   num_classes=5)

n_batches = 10
for i in range(n_batches):
    preds = torch.randn(10, 5).softmax(dim=-1)
    target = torch.randint(5, (10,))
    # 현재 배치에서 모델 평가(정확도)
    acc = metric(preds, target)
    print(f"Accuracy on batch {i}: {acc}")

# 모든 배치에서 모델 평가(정확도)
acc = metric.compute()
print(f"Accuracy on all data: {acc}")
```

ABOUT TORCHMETRICS

◆ TorchMetrics - GPU 설정

```
# 모듈로딩
from torchmetrics.classification import BinaryAccuracy

# 데이터 생성
target = torch.tensor([1, 1, 0, 0], device=torch.device( " cuda " , 0))
preds = torch.tensor([0, 1, 0, 0], device=torch.device( " cuda " , 0))

# Metric은 항상 cpu에서 초기화 ➔ to()함수 : gpu device 설정
confmat = BinaryAccuracy().to(torch.device("cuda", 0))
out = confmat(preds, target)
print(out.device) # cuda:0
```

MODEL INFOMATION

MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ torchsummary / torchinfo

- 설치 : pip install torch-summary

  pip install torchinfo , conda install -c conda-forge torchinfo
- 버전 : Python >=3.6

- 특징

  → Tensorflow의 model.summary()API와 유사하며 모 시각화 기능

  → **torchsummary에서 torchinfo로 변경**

MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ torchsummary / torchinfo

```
from torchinfo import summary

model = MyModel()

# (샘플수 즉, 배치사이즈, 피쳐수)
summary(model, input_size=(506, 13) )
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
BostonModel                              [506, 1]                  --
├─Linear: 1-1                            [506, 50]                 700
├─Linear: 1-2                            [506, 10]                 510
├─Linear: 1-3                            [506, 1]                  11
==========================================================================================
Total params: 1,221
Trainable params: 1,221
Non-trainable params: 0
Total mult-adds (M): 0.62
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.25
Params size (MB): 0.00
Estimated Total Size (MB): 0.28
==========================================================================================
```

MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ visdom

- 설치 : pip install visdom
- 버전 : Python >=3.6

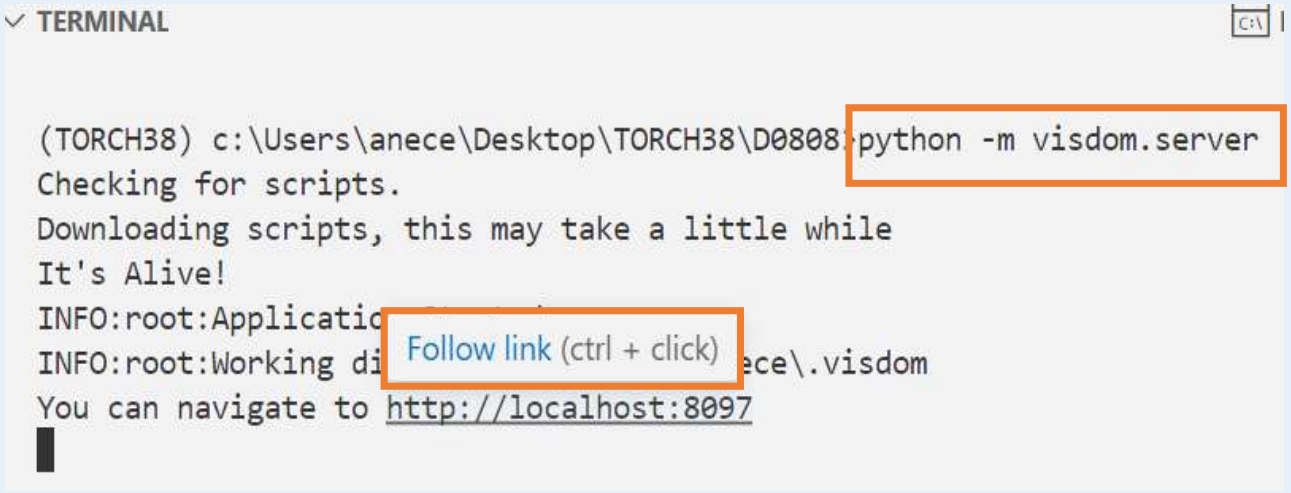
- 특징
  - → 딥러닝 학습할 때, 학습 상황 모니터링 가능 --> 텐서플로우의 tensorboard
  - → 데이터를 풍부하게 시각화 해주는 시각화 도구
  - → 서버실행 : command 창에서 서버 실행 필수 python -m visdom.server
  - → 서버접속 : command 창에 출력된 URL 클릭

MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ visdom

【 서버 실행 】

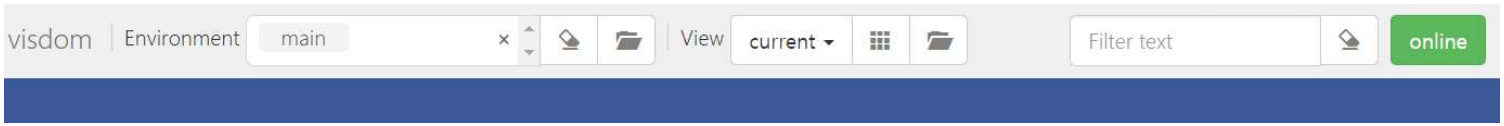


MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ visdom

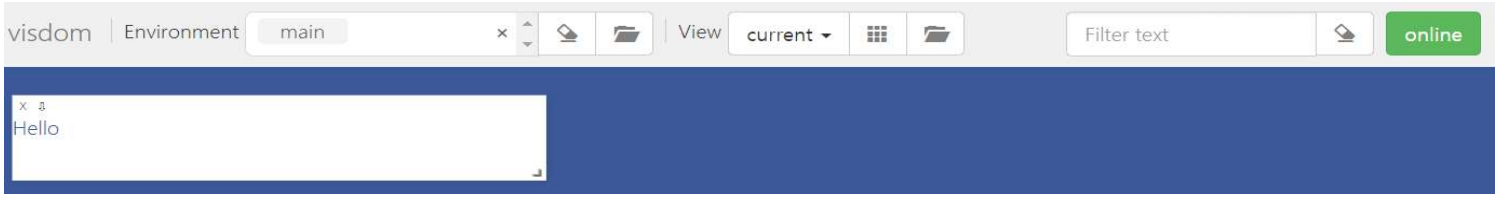
【브라우저】



【코 드】

```
from visdom import Visdom

viz = Visdom()
textwindow = viz.text("Hello")
```

【브라우저】



MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ visdom

【코 드】

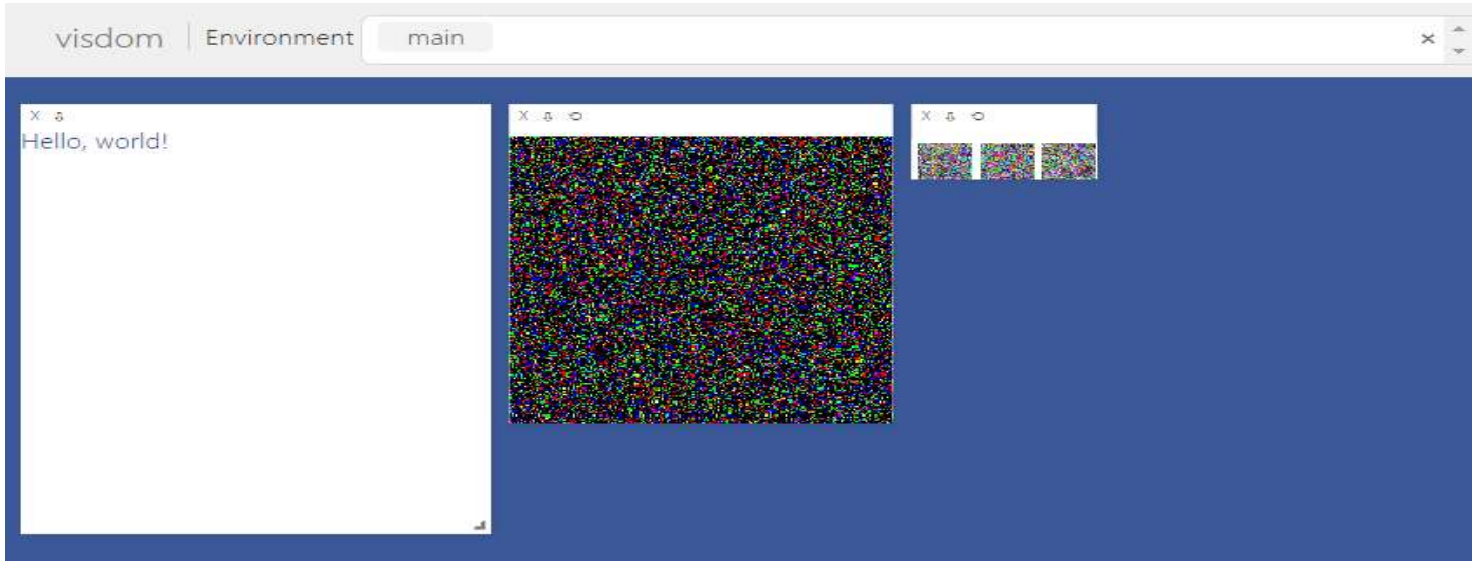
```
from visdom import Visdom

# Text
viz.text("Hello, world!",env="main")

# Image
a=torch.randn(3,200,200)
viz.image(a)

# Images
viz.images(torch.Tensor(3,3,28,28))
```

MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ visdom

【브라우저】



MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ visdom + Model

【코 드】

```
# loss 저장 리스트
loss_history=[]

# 실시간 학습 상태 시각화 선 그래프 생성
plt = viz.line(Y=torch.tensor([[0.]]), X=torch.tensor([[0.]]),
               opts=dict(title="EPOCH & LOSS", legend=['loss'] , showlegend=True))
```

MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ visdom + Model

【코 드】

```
# 학습진행
for epoch in range(EPOCHS):
    # 배치 크기만큼 학습 진행
    for i in range(len(X)//BATCH_SIZE):
        start = i*BATCH_SIZE
        end = start + BATCH_SIZE

        # ndarray ==> tensor변환
        x = torch.FloatTensor(X[start:end])
        y = torch.FloatTensor(Y[start:end])

        # 학습 진행
        pre_y = model(x)
```

MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ visdom + Model

【코 드】

```
        # 가중치/절편 업데이트
        optimizer.zero_grad()                                  # 가중치 기울기 0 초기화
        loss = nn.MSELoss()(pre_y, y.reshape(-1,1))            # 손실 계산
        loss.backward()                                        # 역전파 진행
        optimizer.step()                                       # 가중치/절편 업데이트

        # 현재 진행 상황 출력
        if epoch%20==0: print(f"epoch{epoch} loss:{loss.item():.3f}")

# epoch당 손실 저장
loss_history.append(round(loss.item(), 2))
# 그래프 실시간 업데이트
viz.line(Y=loss.reshape(-1,1), X=torch.tensor([epoch]), win=plt, update='append')
```

MODEL INFORMATION

◆ 모델 정보 및 구조 확인 패키지

❖ **visdom + Model**

【브라우저】