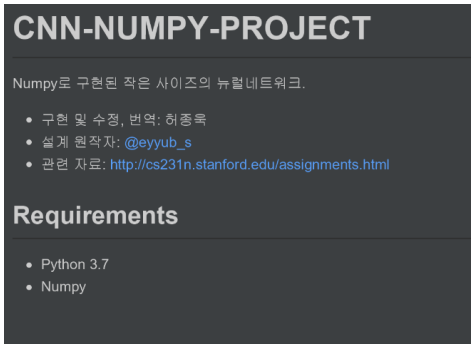
과제 개요

■ numpy_convent 프로젝트 개요

- mnist_test.py : 메인 파일, 실행 스크립트
- 위 코드를 실행하면 기본으로 제공되는 3-Layer 뉴럴넷이 학습됩니다.

- 필요 패키지:
  - numpy
  - python 3



MNIST Dataset Classification

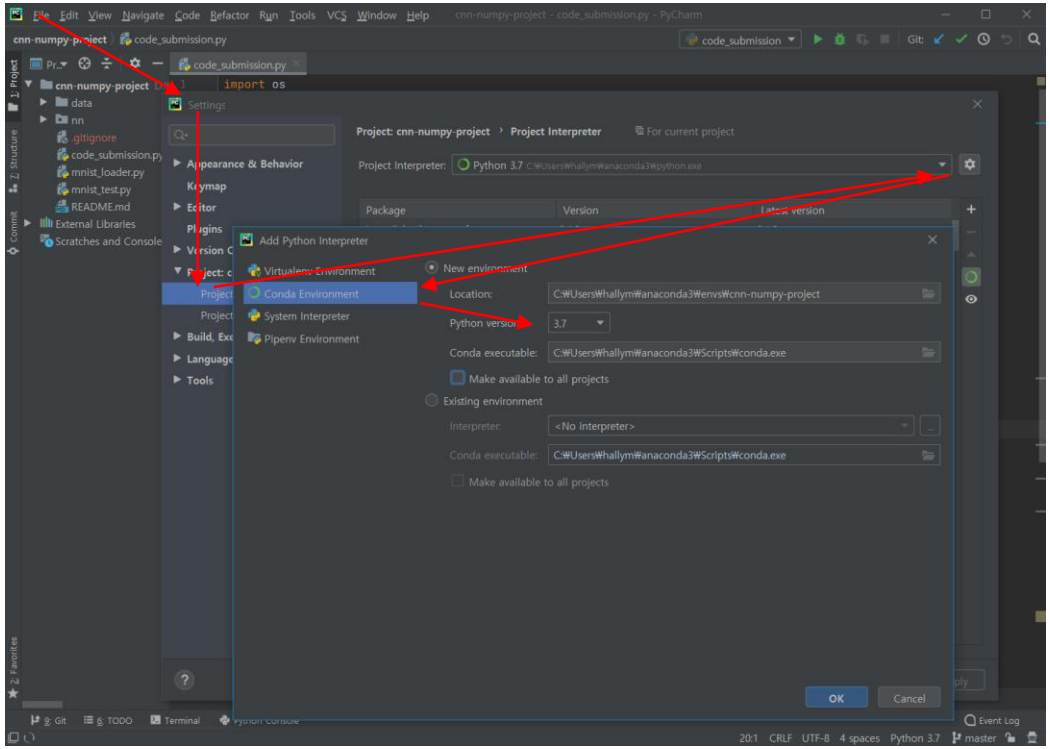
10종류의 문자를 구분하는 문제

과제 개요

■ Pycharm으로 설치하기

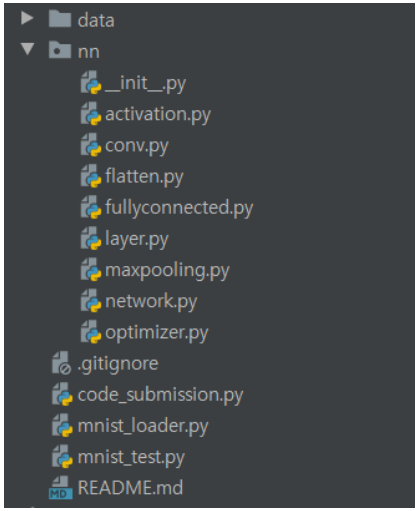
1. 프로젝트 생성
File -> Open -> 폴더 경로 열기

2. 파이썬 인터프리터 설정
File-> Setting
-> Project
-> Interpreter
-> Interpreter 설정하기



과제 개요

■ 프로젝트 개요



- nn 디렉토리
  - **activation.py (2번 문제)**
  - conv.py
  - **optimizer.py (3번 문제)**
  - flatten.py
  - fullyconnected.py
  - layer.py – 레이어 구조 (추상클래스)
  - network.py – 뉴럴넷 클래스 **(1번 문제)**
  - **maxpooling.py (6번 문제)**
- **mnist_test.py : 메인 파일, 실행 스크립트 (4, 5번 문제)**
- mnist_loader.py : 데이터 로딩용

Abstract Layer class

./nn/layer.py

뉴럴넷 layer 클래스의 구조 정의하는
추상클래스. 해당 클래스의 구조를 상속받아
새로운 layer를 구현하는데 사용

method	설명
forward()	forward-pass를 구현
get_activation_grad()	upstream gradient에 activation 함수의 변화율을 적용
backward()	downstream gradient를 계산
get_weight_grad()	weight update를 위한 gradient계산
update()	layer의 weight를 업데이트

```python
class AbstractLayer():
    __metaclass__ = ABCMeta

    @abstractmethod
    def forward(self, inputs):
        raise NotImplementedError()

    @abstractmethod
    def get_activation_grad(self, z, upstream_gradient):
        raise NotImplementedError()

    @abstractmethod
    def backward(self, layer_err):
        # get downstream-gradients
        raise NotImplementedError()

    @abstractmethod
    def get_weight_grad(self, inputs, layer_err):
        # get gradients for weight update
        raise NotImplementedError()

    @abstractmethod
    def update(self, grad, lr):
        raise NotImplementedError()
```

1번. Training 과정 코드 분석

■ Forward pass, backward pass 과정 코드 분석 (20점)

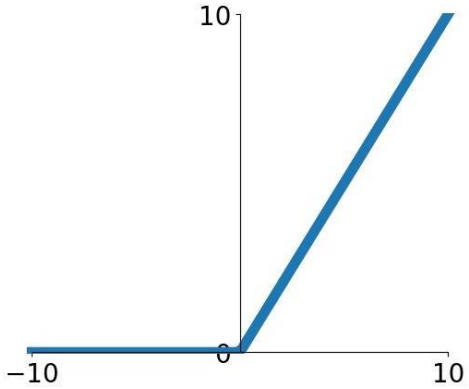
- nn/network.py
- class Network():
  - train_step() 분석 / 설명
  - Forward/backward step이 동작하는 과정을 1페이지 이내로 상세히 서술
  - Backpropagation algorithm이 어떻게 구현되었는지 반드시 확인

2번. Activation Function

■ ReLU 구현 (10점)

- nn/activation.py
- class ReLU() 구현

$$f(x) = \max(0, x)$$



- 보고서 실험결과를 통해 구현이 동작함을 보여야 점수를 얻습니다.

3번. Optimizer

■ Adam Optimizer 구현 (10점)

- nn/optimizer.py
- class AdamOptimizer(): update 구현

$$\mathrm{M(t)} = \beta_1 M(t-1) + (1-\beta_1)\frac{\partial}{\partial w(t)} Cost(w(t))$$

$$\mathrm{V(t)} = \beta_2 V(t-1) + (1-\beta_2)\left(\frac{\partial}{\partial w(i)} Cost(w(i))\right)^2$$

$$\widehat{M}(t) = \frac{M(\mathrm{t})}{1-{\beta_1}^t} \qquad \widehat{V}(t) = \frac{V(\mathrm{t})}{1-{\beta_2}^t}$$

$$\mathrm{W(t+1)} = W(t) - \alpha * \frac{\widehat{M}(t)}{\sqrt{\widehat{V}(t)+\epsilon}}$$

Cost(w(t)) : 손실함수

- 보고서 실험결과를 통해 구현이 동작함을 보여야 점수를 얻습니다.

4번. Weight initialization

■ Weight initialization (10점) – filter_init

- 주어진 네트워크 예시에 weight 초기화 함수가 구현되어 있다.
  - lambda 함수로 구현
  - 파라미터 값은 오른쪽 식과 같이 계산되었다.

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{1}{n_{in}}}$$

(n_{in} : 이전 layer(input)의 노드 수)

  - 본인이 디자인한 네트워크 구조 및 activation 함수에 맞도록 lambda함수의 내부 값을 수정하고 그 전략에 대해 설명하시오.

```
# 심플 CNN 예제
lr = 0.0002
layers = [
    Conv((5, 5, 1, 16), strides=1, activation=relu, optimizer=AdamOptimizer(),
         filter_init=lambda shp: np.random.normal(size=shp) * np.sqrt(1.0 / (28*28))),
    Conv((6, 6, 16, 32), strides=2, activation=relu, optimizer=AdamOptimizer(),
         filter_init=lambda shp:  np.random.normal(size=shp) * np.sqrt(1.0 / (16*24*24))),
    Conv((6, 6, 32, 64), strides=2, activation=relu, optimizer=AdamOptimizer(),
         filter_init=lambda shp: np.random.normal(size=shp) * np.sqrt(1.0 / (32*10*10))),
    Flatten((3, 3, 64)),

    FullyConnected((3*3*64, 100), activation=relu,
                   optimizer_=_AdamOptimizer(),
                   weight_init=lambda shp: np.random.normal(size=shp) * np.sqrt(1.0 / (3*3*64))),
    FullyConnected((100, 10), activation=linear,
                   optimizer_=_AdamOptimizer(),
                   weight_init=lambda shp: np.random.normal(size=shp) * np.sqrt(1.0 / (100.)))
]
net = Network(layers, lr=lr, loss=cross_entropy)
```

  - 보고서 실험결과를 통해 구현이 동작함을 보여야 점수를 얻습니다.

5번. Build network

■ 네트워크 구조 수정 (20점)

- **mnist_test.py**
- 위에서 본인이 구현한 function들을 조합하여 네트워크 재구성
- 반드시 포함해야 하는 기능: **Convolution layer 및 ReLU**

  - Layer(t) outpu과 Layer(t+1)의 input 사이즈를 같게 맞춰 주어야한다.
  - 예. Conv_layer_2의 output (5, 5, 40) → FC_layer_1의 input (5*5*40)

```
# 심플 CNN 예제
lr = 0.0002
layers = [
    Conv((5, 5, 1, 16), strides=1, activation=relu, optimizer=AdamOptimizer(),
         filter_init=lambda shp: np.random.normal(size=shp) * np.sqrt(1.0 / (28*28))),
    Conv((6, 6, 16, 32), strides=2, activation=relu, optimizer=AdamOptimizer(),
         filter_init=lambda shp:  np.random.normal(size=shp) * np.sqrt(1.0 / (16*24*24))),
    Conv((6, 6, 32, 64), strides=2, activation=relu, optimizer=AdamOptimizer(),
         filter_init=lambda shp: np.random.normal(size=shp) * np.sqrt(1.0 / (32*10*10))),
    Flatten((3, 3, 64)),

    FullyConnected((3*3*64, 100), activation=relu,
                   optimizer_=_AdamOptimizer(),
                   weight_init=lambda shp: np.random.normal(size=shp) * np.sqrt(1.0 / (3*3*64))),
    FullyConnected((100, 10), activation=linear,
                   optimizer_=_AdamOptimizer(),
                   weight_init=lambda shp: np.random.normal(size=shp) * np.sqrt(1.0 / (100.)))
]
net = Network(layers, lr=lr, loss=cross_entropy)
```

- 보고서 실험결과를 통해 구현이 동작함을 보여야 점수를 얻습니다.

6번. Max pooling layer

■ Max pooling layer 구현 (30점)

- nn/maxpooling.py
- forward, backward pass 관련 함수를 모두 구현

- 보고서 실험결과를 통해 구현이 동작함을 보여야 점수를 얻습니다.

프로젝트 보고서

■ 각 문제 구현 및 해결 과정에 대한 핵심 내용 요약

- 문제별 코드 작성한 부분만 스크린샷으로 넣을 것
  - 오른쪽 참조
- 구현/해결 방식에 대해서 구체적으로 서술

■ 네트워크 구조나 구현에 따른 결과 분석

- 각 구현은 실험결과를 통해 동작함을 보여야 점수를 얻습니다.
- 구현에 따른 loss값 최종 그래프 최소한 1개 첨부
  - 5~10 epoch 가량 학습
    - Epoch 수란 훈련 데이터셋에 포함된 모든 데이터들이 한 번씩 모델을 통과한 횟수로, 모든 학습 데이터셋을 학습하는 횟수를 의미한다.
    - 예: batch_size=50, total_iteration = 10000 → 10 epoch

제출

■ 제출

- **보고서**
  - 보고서: **보고서_20201234_홍길동.pdf**
- **코드 제출**
  - 프로젝트 폴더 압축 후 업로드: **코드_20201234_홍길동.zip**
- SmartLead 사이트 과제란을 통해 제출

■ 참고사항

- 각종 풀이 설명은 반드시 이해한 뒤 반드시 자신의 글로 쓰시오.
  - 과제 풀이에 대한 설명이 없으면 0점
  - 코드 및 보고서 표절 절대 금지

FAQ

- **Q: 학습이 진행되지 않고 loss가 그대로입니다.**
  - A: 주로 hyper parameter 설정에서 문제가 많이 발생합니다. 본 프로젝트에서 발생할 수 있는 문제점으로는 잘못된 learning rate, weight initialization, network architecture가 있습니다. (7주차 강의자료 참고)

- **Q: 학습을 진행하는데 nan, inf와 같은 이상한 값이 나옵니다.**
  - A: 위와 동일한 해결방법을 사용할 수 있습니다. (관련 키워드: Gradient vanishing and Exploding (* https://wikidocs.net/61375))

- **Q: 네트워크를 구성하는데 자꾸 오류가 납니다.**
  - Layer(t) outpu과 Layer(t+1)의 input 사이즈를 같게 맞춰 주어야 합니다. (6주차 강의자료 참고)
  - 예. Conv_2 (5, 5, 40)의 output parameter수는? → FC_layer_1의 input과 일치 (5*5*40)

- **Q: Convlayer의 학습이 너무 느립니다.**
  - A: 오늘날의 CNN은 GPU 병렬처리 가속을 이용하지만, GPU 지원을 받지 않는 Numpy 기반 코드는 시간이 많이 걸릴 수 밖에 없습니다. 따라서, 충분한 여유 시간을 두고 과제를 진행하세요.
  - A: 적절한 learning rate, weight initialization과 Stride, Pooling layer, Adam optimizer등을 사용하면 최적화 시간을 크게 단축할 수 있습니다.