Project 제안서

냥냥 torch for Classification

• 작성자 : 31 기 19 양재민
• 작성일 : 2023. 09. 15

목 차

1. Project 소개

(1) 선정 동기

Deep Learning 을 공부하고 Project 와 다양한 competition 에 참가해보면서 매우 많은 실험을 진행해보았다. 하지만 문제의 본질을 파악하고 내부의 원리를 이해한 후 문제를 해결해야 하지만 단순히 성능(수치)를 높이기 위해, competition 에서 조금이라도 높은 리더보드 순위를 얻기 위해 기계적으로 실험을 돌리는 경우가 많았다. 또한 DeepLearning Framwork(Pytorch)를 사용하면서 그 내부 동작 원리에 대한 의문이 드는 경우가 많았으며 그것은 실제로 많은 사람들이 하는 의문이기도 하다. DeepLearning 분야의 대학원 진학을 희망하는 현재 상황에서 또 다른 task 에 관심을 갖고 공부하기 보다는 머리 속의 Deep Learning flow 에 대한 정리와 놓치고 지나갔던 부분의 공부가 필요하다고 느꼈으며 이러한 시간을 바탕으로 3 학년 때는 좀 더 advance 된 task 를 진행하고 싶었다. 그래서 2-2 때에는 이런 욕구를 충족시킬 수 있는 "나만의 DeepLearning Framework 제작"에 관심을 갖게 되었다.
결국 "냥냥 torch"라는 새로운 Deep Learning framework 를 직접 제작해 보기로 하였으며 "냥냥 torch"를 개발하는 과정을 통해 이러한 이점을 얻을 수 있을 것이라고 생각했다.
1. data, model, loss, optimizer, train, inference 에 이르는 부분을 구현해야 하니 Deep Learning 학습 과정에 대한 전반적인 이해도를 높일 수 있다.
2. 현대 프로그래밍에 필수적인 OOP(Object-Oriented Programming)를 연습할 수 있다.
3. 모든 과정을 직접 구현하기 때문에 Implementation 능력을 향상시킬 수 있다.

또한 현재 가장 널리 쓰이고 있는 Pytorch 의 방식을 일부 reference 하여 Pytorch Framework 자체의 이해도도 늘릴 수 있을 것으로 예상한다.
현재 Pytorch 를 비롯한 많은 Framework 들은 다양한 Task 에 활용할 수 있지만 "냥냥 torch"는 Classification Task 에 대해서만 사용할 수 있다. 그 이유 중 첫번째는 물리적인 구현시간에 한계가 있을 것이라고 생각했다. 두번째는 다른 framework 처럼 거대한 프로젝트를 혼자 진행하기에는 구현능력이 부족하다고 생각했다. 또한 "냥냥 torch"의 학습속도나 성능 등이 아직 미지수이기 때문에 먼저 비교적 간단한 Task 부터 수행할 수 있게 프로젝트의 방향을 설정하였다. 또한 이번 "냥냥 torch"의 구현이 성공적으로 끝난다면 추가적인 Task 에 대한 기능을 추가해 update 하는 방식으로 진행하고자 한다.

(2) Project 시나리오

1. torch 를 사용하지 않고 새로운 DeepLearning Framework 를 제작한다.

2. Data Management 부분에서 해당 Task train 에 바로 사용될 수 있게 Preprocessing 한다.

3. 구현된 layer 를 사용하여 Model 을 구현한다.

4. 구현된 function 을 이용하여 구상한 train strategy 에 맞게 train 한다.

5. train 과정을 monitering 하고 model saving function 을 이용해서 best model 을 saving 한다.(.pickle)

6. 학습 후 해당 .pickle 파일을 구현된 model load function 을 사용해서 load 후 test set 을 inference 한다.

7. 2 개의 metric 을 얻는다.

8. 이후 Pytorch 에서 같은 model, 같은 data 등 같은 train strategy 로 학습 후 둘의 성능을 비교한다.

9. 성능 차이를 확인하고 성능 차이가 나는 이유를 고찰한 후 개선한다.

10. Streamlit 을 활용해서 “냥냥 torch”의 Documentation page 를 제작한다.

(3) 목표

1. Dataset Class, Dataloader Implementation

2. Layer Function Implementation

3. Loss Function Implementation

4. Optimizer Function Implementation

5. Model Building

6. Train Function Implementation

7. Logging & Monitoring Implementation

8. Model Saving & Loading Function Implementation

9. Inference Function Implementation

10. Evaluation Implementation

11. Comparison between Pytorch and 냥냥torch

12. 냥냥torch Improvement

13. Documentation with Streamlit

2. Project 내용

(1) System Architecture

Framework 의 구성요소를 Data Management, Model Building, Training, Utility, Evaluation & Inference, 크게 5 개의 요소로 나누었으며 이를 폴더로 분리하였다. 각 구성요소 마다 포함하고 있는 file 을 명시하였으며 아래 그림은 각 구성요소의 상호작용을 통한 Deeplearning 과정을 도식화한 다이어그램이다.

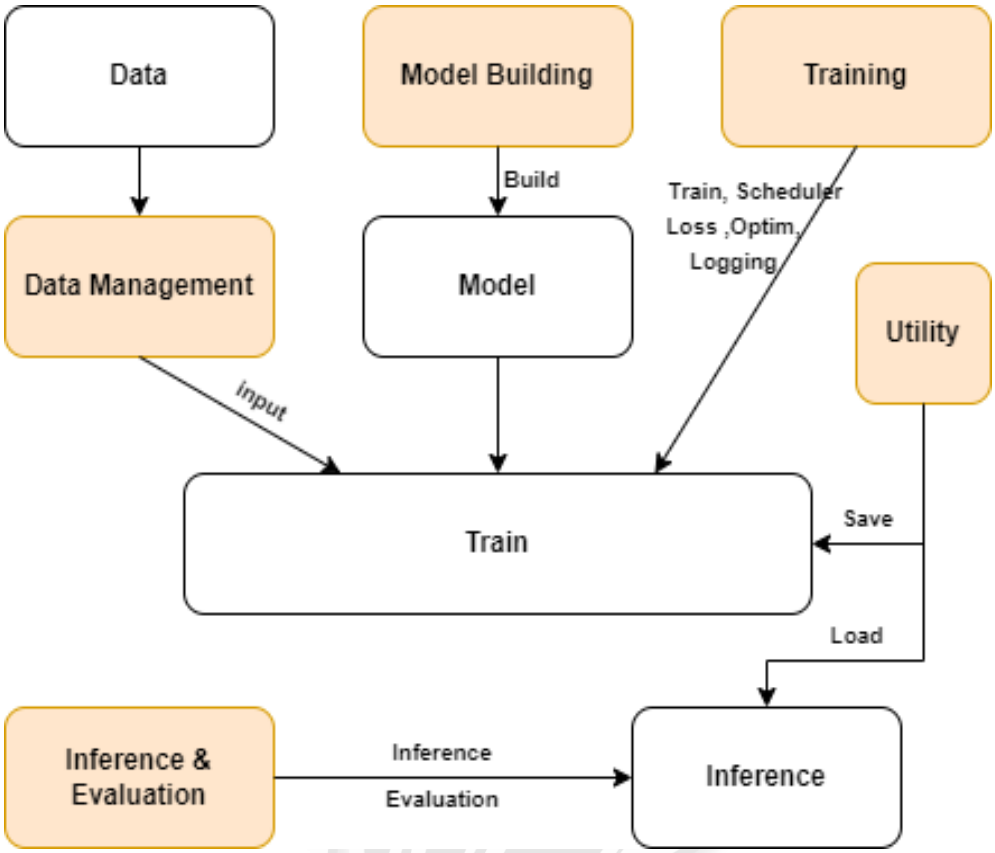
1. Data Management
- Nyang_dataset.py
- Nyang_dataloader.py
- Nyang_utils.py
- Nyang_transform.py
- Nyang_datapipeline.py

2. Model Building
- Nyang_layer.py

3. Training
- Nyang_loss.py
- Nyang_optimizer.py
- Nyang_scheduler.py
- Nyang_train.py

4. Utility
- Nyang_save_load.py

5. Evaluation & Inference
- Nyang_metric.py
- Nyang_inference.py



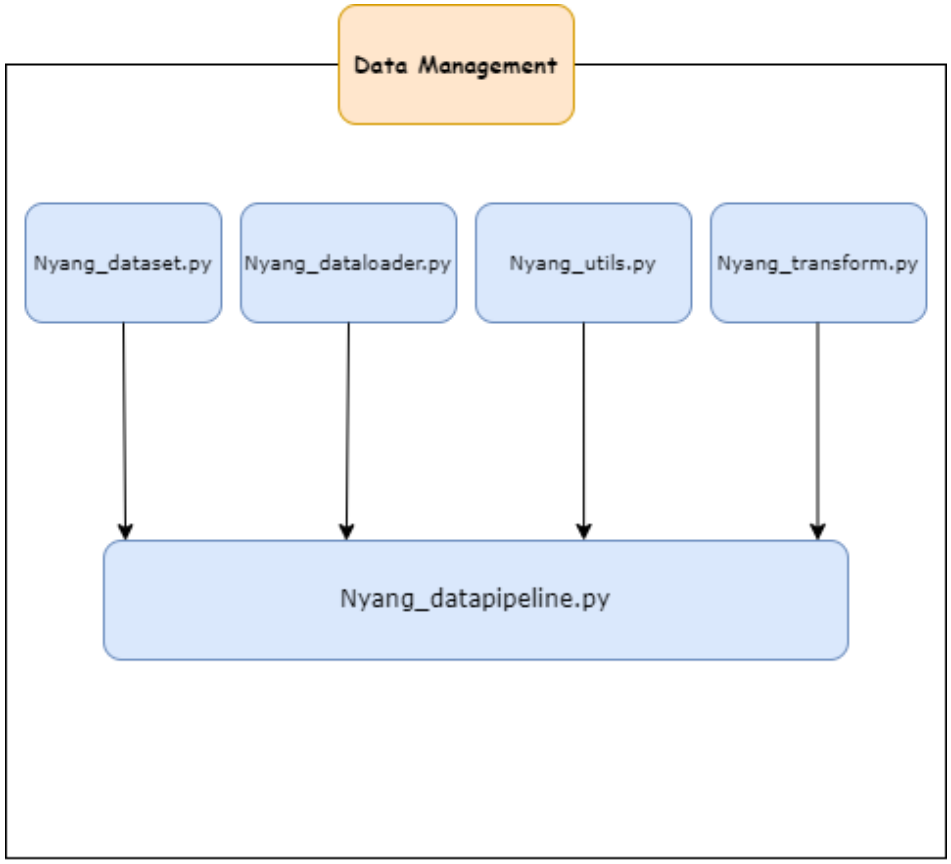
개발환경은 다음과 같다.

1. IDE(통합 개발 환경): Visual Studio Code
2. 가상환경 관리: Anaconda
3. 버전관리: Git, GitHub

현재 python 3.9.13 을 사용 중이나 개발과정 중에 pakage 와의 호환성을 고려해 변경될 가능성이 있다

(2) 목표 구현 내용

1. Data Management



DeepLearning 을 처음 접하는 이들은 train 을 해보기도 전에 dataset,dataloader 를 정의하고 data 를 나누는 시점에서 많은 어려움을 겪는다. 그래서 classification task 를 위한 필요한 기능은 포함되어 있지만 간단하게 train, valid, test set 을 준비할 수 있는 Data Mangement 를 구현하고자 한다.

1) Nyang_dataset.py

A. class NyangDataset

i. User 가 custom dataset 을 만들 수 있게 추상클래스로 구현하고자 한다.

B. class NyangNyangDataset

i. data split 기능을 구현하고자 합니다. (by def Nyang_split)

ii. data root 와 class list, data split ratio 를 parameter 로 받아서 각 mode(train, valid, test)에 맞는 label 과 image path 를 대응시킨 tuple 로 반환하게 구현하고자 한다.

2) Nyang_dataloader.py

A. Class NyangDataloader

i. Pytorch 의 dataloader 와 같이 batch size 로 data 를 나누어서 반환하게 구현하고자 한다.

ii. 다음 batch 에서 이번 배치의 마지막 index 부터 시작해야 함으로 __iter__를 사용해 구현하고 generator 로 반환하게 구현하고자 한다.

iii. Batch 사이즈로 나눈 후 batch size 로 나누어 떨어지지 않는 data 는 마지막 batch 로 사용할 수 있도록 구현하고자 한다.

3) Nyang_utils.py

A. def Nyang_split.py

i. class list 를 입력 받아서 one-hot Encoding 을 진행해주어 label 로 저장하는 기능을 구현하고자 한다.

ii. dataset 의 index 를 shuffle 하여서 data shuffle 기능을 구현하고자 한다.

iii. shuffle 후 입력 받은 mode(train, valid, test)와 ratio 에 맞게 data split 기능을 구현하고자 한다.

B. def Nyang_images_grid

i. image 와 label 을 grid 형태로 시각화 하는 함수를 구현하고자 한다.

ii. Batch 단위로 image 를 확인하고 적용된 augmentation 이나 shuffle 상태를 확인할 수 있게 구현하고자 한다.

4) Nyang_transform.py

A. def Nyang_transforms

i. train, valid, test 에 각각 augmentation 을 적용할 수 있는 기능을 구현하고자 한다.

ii. albumentations 를 이용해서 augmentation 을 적용할 수 있도록 구현하고자 한다.

5) Nyang_datapipeline.py

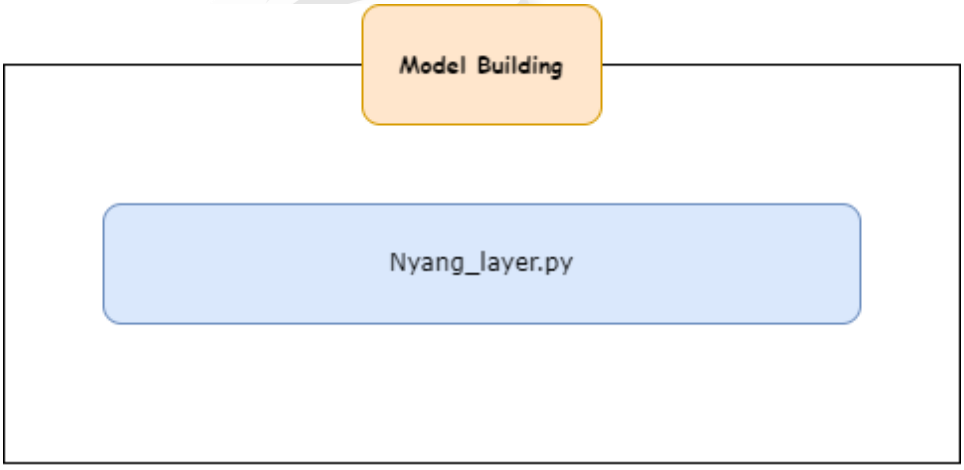
A. def NyangNyang_Dataset_Ready

i. class NyangNyangDataset 객체와 class NyangDataloader 객체를 정의하는 함수를 구현하고자 한다.

ii. def Nyang_transforms 에서 각 mode 마다 transform 을 받아와서 Nyang_Datalodaer 에 input 으로 넣어줄 수 있게 구현하고자 한다.

iii. NyangNyang_Dataset_Ready 를 사용함으로써 mode 별 data split 과 augmentation 적용, one-hot encoding 등 preprocessing 된 data 를 batch size 로 나누어서 반환할 수 있게 구현하고자 한다.

2. Model Building



Model 을 정의하기 위해서는 Model 을 구성할 수 있는 layer 가 있어야한다. Image classification task 에서 자주 사용되는 Model 의 layer 를 위주로 구현하고자 한다. parameter 학습이 필요한 layer 들은 기본적으로 backward 에서 해당 layer 의 gradient 를 계산하고 optimizer 를 통해서 weight 를 update 하도록 구현하고자 한다.

1) Nyang_layer.py

A. class NyangModel

i. 사용자가 model 을 정의할 때 상속받는 기본 class 로 구현하고자 한다.

ii. Batch Norm, Dropout 등 valid 나 test 를 할 때 다른 동작을 하는 layer 들을 eval mode 로 바꾸기 위해서 def eval()을 구현하고자 한다.

B. class Linear

i. FC Layer 를 구현하기 위한 layer class 이다.

ii. input dim 과 output dim 을 받는다.

iii. forward pass 시에 dot product 를 통해서 연산을 진행하며 backward pass 시에는 앞선 layer 의 gradient 를 받아서 해당 layer 의

gradient 를 계산한다. 또한 이전 layer 의 gradient 계산을 위해서 ‘ 해당 layer 의 gradient 를 return 한다. optimizer 를 통해 가지고 있는 weight 들을 update 한 후 gradient 를 0 으로 초기화 한다.

iv. weight 와 Layer 내의 사용 activation function 에 따라서 가중치 초기화를 다르게 구현하고자 한다.

C. class Conv

i. Convolution 연산을 구현한 layer class 이다.

ii. in_channels, out_channels, kernel_size, stride, padding 을 받는다.

Iii. layer 를 쌓을 때 보통 relu 계열의 activation function 과 함께 사용하는 경우가 많아서 가중치 초기화는 He initialize 를 사용한다.

iv. forward pass 시 sliding window 방식을 사용해서 convolution 연산을 진행하며 backward pass 시에는 앞선 layer 의 gradient 를 전달 받아서 해당 layer 의 gradient 를 계산한다. 또한 이전 layer 의 gradient 계산을 위해서 해당 layer 의 gradient 를 return 한다. optimizer 를 통해 가지고 있는 weight 들을 update 한 후 gradient 를 0 으로 초기화 한다.

D. class MaxPool

i. convolution layer 의 output feature 를 down sampling 하면서 강한 특징을 보존하는 layer class 이다.

ii. forward pass 시에 input 에 대해 입력 받은 크기의 window 내에서 최댓값을 선택하여 출력으로 반환한다. 마찬가지로 sliding ‘ window 방식을 사용한다.

iii. 학습해야하는 parameter 가 없으므로 backward 과정에서는 앞선 layer 의 gradient 를 입력 받아서 해당 layer 의 gradient 를 계산 후 이전 layer 로 전달한다.

F. class AvaragePool

i. convolution layer 의 output feature 를 down sampling 하면서 평균을 구하는 과정을 통해 대표성 있는 값을 추출하는 layer class 이다.

ii. forward pass 시에 input 에 대해 입력 받은 크기의 window 내에서 평균값을 구하여 출력으로 반환한다. sliding window 방식을 사용한다.

iii. average pooling 또한 학습해야 하는 parameter 가 없으므로 backward 과정에서는 앞선 layer 의 gradient 를 입력받아서 해당 layer 의 gradient 를 계산한 후 이전 layer 로 전달한다.

G. class Relu

i. convolution layer 와 자주 사용되는 activation function 인 Relu 를 구현한 class 이다.

ii. forward pass 시에는 input 이 0 보다 크면 그대로 x 를 출력하고 0 보다 같거나 작으면 0 을 출력한다.

iii. 학습할 parameter 가 없으므로 backward 시에 앞선 layer 에서 gradient 를 input 으로 받는다. input 으로 받은 앞선 layer 의 gradient 가 0 보다 크면 그대로 이전 layer 로 gradient 를 전달하고 0 보다 같거나 작다면 0 을 전달한다.

H. class LeakyRelu

i. 앞에서 언급되었던 relu 의 Dead Neuron 문제를 해결하기 위해 나온 leaky relu 를 구현한 class 이다.

ii. forward pass 시에는 input 이 0 보다 크면 x 를 출력하고 0 보다 작으면 0.01 을 곱해서 출력한다.

iii. 학습할 parameter 가 없으므로 backward 시에 앞선 layer 에서 gradient 를 input 으로 받는다. input 으로 받은 앞선 layer 의 gradient 가 0 보다 크면 그대로 이전 layer 로 gradient 를 전달하고 0 보다 같거나 작다면 0.01 을 곱해서 전달한다.

I. class Elu

i. input 이 0 보다 작을 때에도 지수 함수의 형태로 출력하여 Dead Neuron 문제를 방지 할 수 있게 하는 Elu 를 구현한 함수 입니다.

ii. α 는 1.0 으로 구현했습니다.

iii. forward pass 시에는 input 이 0 보다 크면 x 를 출력하고 0 보다 같거나 작으면 지수함수의 형태로 출력한다.

iv. 학습할 parameter 가 존재 하지 않으므로 backward 시에 input 이 0 보다 클 경우 그대로 이전 layer 에 전달되며 input 이 0 보다 작거나 같을 경우 지수함수의 미분을 거쳐 이전 layer 에 전달된다.

J. class tanh

i. 출력 범위가 [-1,1]이며 S 형태의 곡선을 가지는 activation function 인 tanh 를 구현한 class 이다.

ii. forward 시에는 x 의 값이 tanh 의 입력으로 들어가 출력되며 backward 시에는 학습할 parameter 가 없으므로 앞선 layer 에서 input 으로 들어온 gradient 와 tanh 의 도함수와 곱한 후 이전 layer 로 전달한다.

K. class softmax

i. 분류모델 마지막 layer에서 흔하게 사용되며, 각 class에 대한 확률을 구할 수 있는 activation function인 softmax를 구현한 class이다.

ii. forward pass 시에 지수연산에 의한 overflow를 방지하기 위해서 input vector중 최대값을 뺀 후 지수연산을 진행하고 지수화 된 값을 그 합으로 나누어 정규화 한다.

iii. backward pass 시에는 학습시킬 parameter가 없기 때문에 앞선 layer의 gradient를 전달받아서 softmax의 도함수와 곱한 후 이전 layer로 전달한다.

L. class Selu

i. 네트워크가 깊어질 때에도 activation function output 의 평균과 분산을 유지하도록 도와주는 Selu를 구현한 class이다.

ii. α와 λ는 가장 흔하게 쓰이는 값인 1.6732, 1.0507으로 고정이다.

iii. forward와 backward 시에 위의 activation function들과 마찬가지로 작동한다.

N. class Dropout

i. overfitting을 방지하기 위해서 train 과정 중에 무작위로 일부 Neuron을 끄는 방식으로 작동하는 dropout을 구현한 class이다.

ii. forward 시에 Train mode일 때와 Eval mode일 때가 다르게 작동합니다. Train mode일 때는 일부 뉴런을 끔으로써 overfitting을 방지한다. 뉴런을 끄는 확률인 p를 input으로 받습니다. Eval mode일 때는 뉴런을 끄지 않지만 output에 (1-p)를 곱해줘서 train 중에 꺼진 뉴런의 기여도를 조정한다.

iii. backward 시에 학습 중에 끈 뉴런은 gradient를 0으로 설정하고, 나머지 뉴런은 gradient를 그대로 이전 layer에 전달한다.

M. class Batchnorm

i. 각 레이어의 입력 분포를 안정화시켜서 학습을 더 빠르고 안정적으로 만드는 것이 batch norm이며 이것을 구현한 class이다.

ii. forward 시에 train 할 때와 eval 할 때 동작이 다릅니다. train 시에는 입력 데이터의 평균과 분산을 계산하고 이를 사용하여 data를 정규화 합니다. eval 시에는 이전 학습 단계에서 계산된 이동 평균과 이동 분산을 사용하여 데이터를 정규화 한다. 정규화된 데이터에 gamma와 beta parameter를 사용하여 스케일링 및 시프팅 하는 것을 구현하고자

한다.

iii. backward 시에는 스케일 및 시프트에 대한 gradient, 정규화된 data 에 대한 gradient, 분산에 대한 gradient, 평균에 대한 gradient, input data 에 대한 gradient 를 계산하며 backward 시에 input 으로 받은 앞선 layer 의 gradient 와 해당 layer 의 미분을 곱하여 계산하는 과정을 구현하고자 한다.

O. class nyang_flow

i. layer 를 묶음으로 관리하게 하는 역할을 하고 forward 와 backward 를 순차적으로 수행 될 수 있게하는 class 이다.

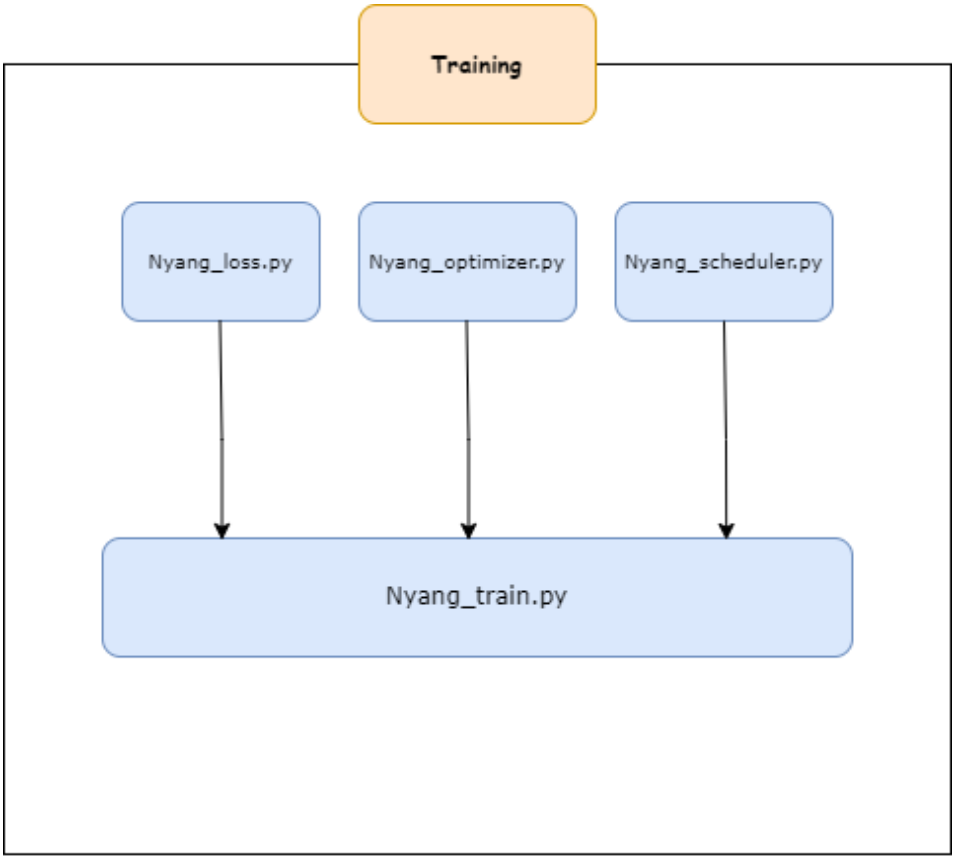
P. class flatten

i. Conv layer 를 지나서 1 차원의 벡터 형태로 바꿔서 FC layer 의 input 으로 사용될 수 있게 하는 class 이다.

ii. forward 시에 backward 를 위해 input shape 를 저장하며 batch size 는 유지하고 나머지를 1 차원 벡터로 펼쳐서 반환한다.

iii. backward 시에는 저장해놓은 input shape 의 형태로 gradient 를 복원해서 반환한다.

3. Training



train 과정을 거치기 위해서 꼭 필요한 요소들이 있다. Loss 함수와 weight 의 편미분을 통해서 gradient 를 update 하며 이 update 는 optimizer 를 통해서 이루어진다. Training Block 에서는 train 에 필요한 요소를 구현하고 꼭 필요하지 않더라도 scheduler 나 logging 및 monitering, early stopping 같은 기능을 train 에 도움이 될 것이라고 판단하여 구현하고자 한다.

1) Nyang_loss.py

A. class MSE

i. regression 또는 classification 에서 많이 사용되는 Mean Squared Error 를 구현한 class 이다.

ii. forward 시에 prediction 값과 label 에 대한 MSE 값을 계산하고 반환한다.

iii. backward 시에 Loss 함수를 prediction 에 대해서 편미분 한 값을 계산하고 이전 layer 에 반환한다.

B. class BCE

i. Binary Classification 에서 많이 사용하는 Loss 인 Binary Cross

Entropy 를 구현한 class 이다.

ii. model 에서 logit 을 입력으로 받아서 forward pass 안에 Sigmoid 를 구현하고자 합니다. Sigmoid 를 거친 후에 BCE 식 내의 input 으로 들어가 값을 계산하고 반환한다.

iii. backward 시에는 Loss 함수를 logit 으로 편미분한 값을 계산하고 이전 layer 에 반환한다.

C. class CE

i. Classification 문제에서 확률 분포 간의 차이를 측정하기 위해 주로 사용되는 Loss 함수인 Cross Entropy 를 구현한 class 이다.

ii. model 의 출력인 logit 을 입력으로 받아서 forward pass 안에서 Softmax 를 적용한다. Softmax 가 적용된 후에 CE 식의 입력으로 들어가 값을 계산하고 반환한다.

iii. backward 시에는 Loss 함수를 logit 에 대해 편미분한 값을 계산하고 이전 layer 에 반환한다.

D. class FocalLoss

i. dataset 에서 class imbalance 문제를 해결하기 위해 Cross Entropy 를 변형하여 만든 Focal Loss 를 구현한 class 이다.

ii. forward pass 안에서 gamma 와 alpha(parameter)를 활용하여 어려운 케이스 또는 쉬운 케이스에 대한 Loss 의 가중치를 조절합니다. 이 방법은 softmax 를 거친 확률 분포에 적용된다.

iii. backward 시에는 다른 Loss 함수와 같이 편미분한 값을 계산하고 이전 layer 에 반환한다.

2) Nyang_optimizer.py

A. class SGD

i. 기본적인 최적화 방법인 SGD 를 구현한 class 이다.
GD 는 전체 dataset 에 대한 gradient 를 계산하여 최적화 방법이며 SGD 는 batch 내에서의 gradient 를 계산하여 계산하는 최적화 방법이다.

ii. 각 layer 내의 weight 의 update 를 위해서 def step 을 구현하고자 한다. Hasattr 함수를 사용해서 해당 layer 의 weight 와 bias 를 해당 optimizer 의 방법으로 update 한다다.

iii. def zero_grad 함수를 구현하여서 update 후 layer 에 남아있는 gradient 를 0 으로 초기화 시킨다.

B. class Adam

i. momentum 과 RMSprop 의 아이디어를 결합한 최적화 방법인 Adam 을 구현한 class 이다.

ii. def step 에서 momentum parameter 를 반영하여 update 를 진행한다.

iii. 마찬가지로 그래디언트가 축적되지 않도록 하기 위해서 def zero_grad 가 구현되어 있다.

C. class RMSprop

i. weight update 시 gradient 의 제곱값의 이동 평균을 유지하여 학습률을 조정하는 Rmsprop 을 구현한 class 이다.

ii. gamma 와 epsilon 을 0.9, 1e-8 로 고정할 예정이다.

iii. def step 에서 gradient 의 제곱의 이동 평균을 계산하고 이를 사용해 weight 를 update 한다.

D. class AdamW

i. Adam 의 변형으로 weight decay 를 포함하고 있는 AdamW 를 구현한 class 이다.

ii. 2 개의 beta, epsilon, weight_decay 를 parameter 로 받는다.

iii. def step 에서 1 번째 및 2 번째 moment 를 계산하고 weight decay 를 적용하여 weight 를 update 한다.

3) Nyang_scheduler.py

A. class StepDecay

i. 일정한 간격(epoch or step)마다 learning rate 를 줄이는 전략인 Step Decay 를 구현한 class 이다.

ii. def get_lr 을 통해서 optimizer 의 learning rate 를 얻고 def step 을 통해서 optimizer 의 learning rate 를 update 한다.

B. class ExpoDecay

i. 일정한 간격(epoch or step)마다 지수적으로 learning rate 을 감소시키는 전략인 Exponetial Decay 를 구현한 class 이다.

ii. def get_lr 을 통해서 optimizer 의 learning rate 를 얻고 def step 을 통해서 optimizer 의 learning rate 를 update 한다.

C. class WarmUp

i. train 초기에 낮은 learning rate 에서 시작하여 점진적으로 설정된 learning rate 까지 높이는 Warm Up 을 구현한 class 이다.

ii. def get_lr 을 통해서 optimizer 의 learning rate 를 얻고 def step 을 통해서 optimizer 의 learning rate 를 update 한다,

4) Nyang_train.py

A. def Nyang_train

i. train 을 전체의 과정을 포함 하고 있는 함수이다.

ii. model, optimizer, loss, train dataloader, valid dataloader, epochs, patience, scheduler, model save path 를 input 으로 받는다.

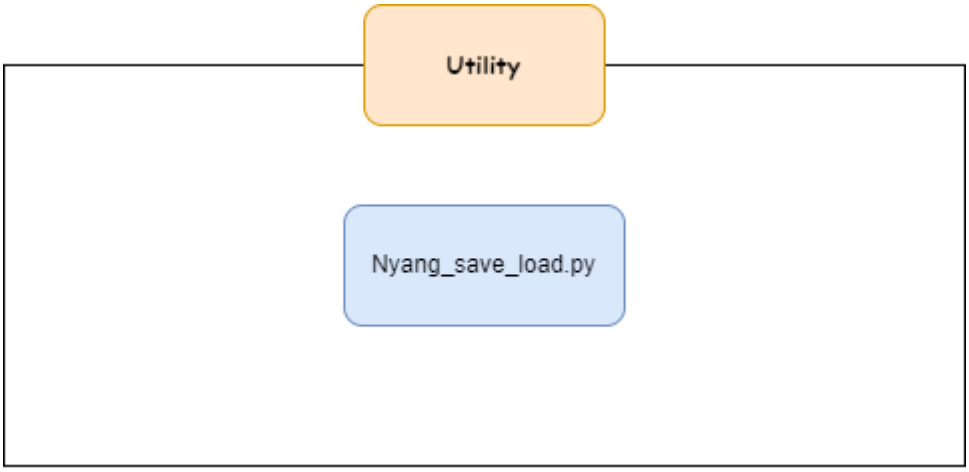
iii. wandb 를 import 해서 wandb 를 통해 실험 logging 및 monitering 이 가능하다.

iv. tqdm 을 통해서 train 과 valid 의 진행도를 시각적으로 확인 할 수 있게 구현하고자 한다.

v. valid 를 통해 계산 되는 loss 와 best loss 를 지속적으로 비교하며 더 낮은 loss 를 가진 model 을 def save_model 을 이용해서 pickle 로 저장한다.

vi. 만약 patience 의 숫자의 epoch 만큼 지속적으로 best loss 의 값보다 val_loss 의 값이 작다면 더 이상 유의미한 학습이 되지 않는다고 생각하여 Early Stopping 을 통해서 train 을 멈춘다.

4. Utility



Deeplearning Frame work 에서 train 과정을 구현하는 것은 매우 중요하다. Deeplearning Framework 는 User 가 DeepLearning 자체에 집중하고 활용할 수 있게 해줘야 한다고 생각한다. 그래서 간편하게 train 결과(weight,model)를 저장하고 불러오는 기능을 구현하고자 한다. .pt 확장자를 사용하고 싶었지만 tensor 형태가 아니면 저장을 하지 못해서 Python 의 객체 저장형태인 .pickle 을 사용해서 save 와 load 를 구현하고자 한다.

1) Nyang_save_load.py

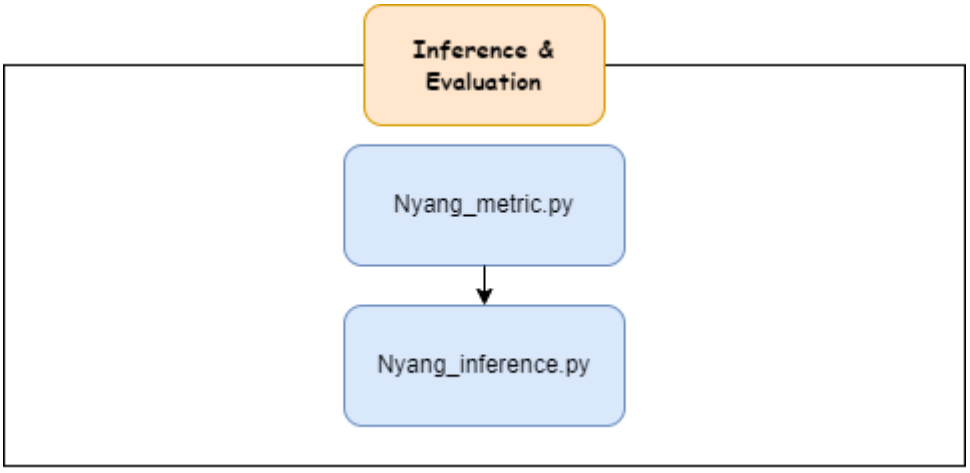
A. def Nyang_save_model

i. pickle 을 import 하고 pickle.dump()를 사용해서 pickle 형태로 save 하는 함수이다.

B. def Nyang_load_model

i. pikle 형태로 저장된 model 을 pickle.load()를 통해 inference 를 위해 load 하는 함수이다.

5. Inference & Evaluation



Train 이 Deeplearning 과정 내에서 많은 비중을 차지하는 것이 사실이지만, 사실 그 train 과정 또한 더 정확한 prediction 을 위한 과정이다. Train 한 model(weight)를 올바른 순서로 forward pass 를 거치게 하는 inference 를 구현하고자 하며 또한 classification task 에서 널리 사용되는 accuracy, F1 score 를 inference 동시에 계산할 수 있는 metric 함수를 구현하고자 한다.

1) Nyang_metric.py

A. def accuracy

i. 흔하게 사용하는 metric 중 하나인 accuracy 를 계산해주는 함수이다.

ii. predictions 와 targets 를 input 으로 받아 accuracy 를 output 으로 반환한다.

B. def f1_score

i. 흔하게 사용하는 metric 중 하나인 F1_score 를 계산해주는 함수이다.

ii. predictions 와 targets 와 positive label 을 input 으로 받아서 precision 과 recall 을 계산하여 조화평균으로 F1 score 를 output 으로 반환한다.

2) Nyang_inference.py

A. def Nyang_inference

i. inference 를 위한 함수입니다. Inference 시 Nyang_metric.py 를 import 받아서 2 개의 metric 을 계산합니다. Model 을 load 하고 forward pass 를 거치는 과정을 구현하고자 한다.

ii. tqdm 으로 inference 의 진행도를 시각화 할 수 있게 구현하고자 한다.

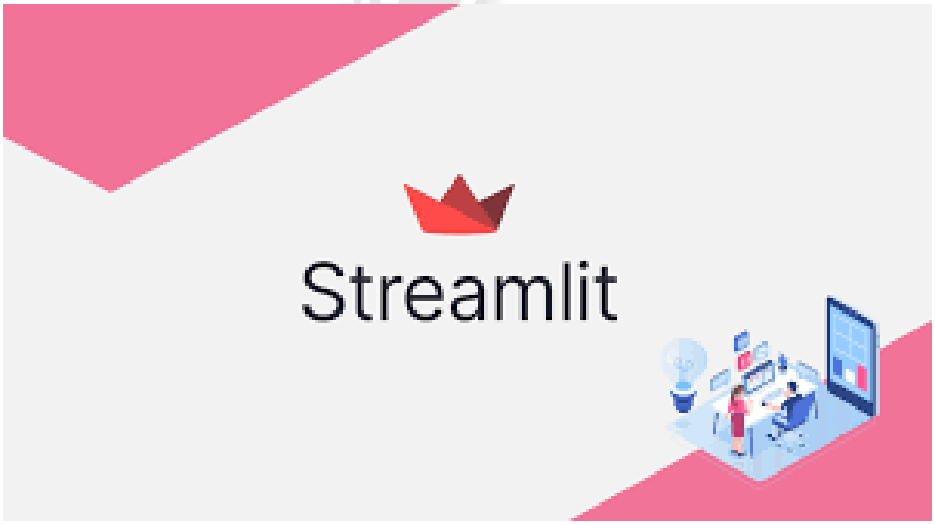
iii. dropout 이나 batchnorm 같은 layer 들은 eval()모드를 통해 train 때의

동작을 inference 때의 동작으로 바꿔준다.

6. Pytorch 와 성능 비교

현재 Pytorch 는 DeepLearning Framework 중 가장 높은 점유율을 차지하고 있는 framework 이다. 그렇기에 “냥냥 torch”와 강장 비교하기에 좋은 대상이라고 생각한다..
비교를 하기 위해서 같은 dataset, model, train strategy 로 비교를 할 예정이다. 당연하게도 Pytorch 구현 코드안에는 냥냥 torch 에서 구현한 내용보다 더 깊게 많은 기능이 구현되어 있다. 따라서 어느정도 성능차이와 학습 속도 차이는 있을 것으로 예상한다.
따라서 냥냥 torch 와 Pytorch 와 성능비교를 통해서 Pytorch 의 어떤 부분이 냥냥torch와 성능차이를 유발하는지 고찰을 거치고 해당 부분을 냥냥torch에 반영 해보고자 한다.

7. Documentation With Streamlit



Streamlit 은 웹, 앱에 대한 지식 없이도 Python 을 이용해서 빠르게 간단한 웹, 앱을 만들 수 있게 해준다. 또한 Github 를 이용하여 배포가 가능한 Streamlit Cloud 를 사용할 예정이다. Streamlit Cloud 를 사용할 경우 Docmentation 을 local 에서 수정 후 Github origin repository 에 Push 만 한다면 쉽게 웹에 반영을 할 수 있다. 또한 무료이며 URL 을 통해 이용자가 쉽게 접근을 할 수 있다는 장점도 있다.

세부 구현 계획

1) Documentation 을 작성한다.

2) NyangNyang_torch_streamlit.py 내에서 Streamlit 을 통해 웹을 구성하고 내용을 입력한다.

3) 해당 py file 을 Github 에 업로드하고 Streamlit Cloud 내에서 app 을 생성한다.

4) 생성한 app 과 Github 에 올린 repository 와 연동한다.

6) 생성된 app 을 monitering 하며 수정한다.

6) app 이 배포되면 고유한 URL 이 생성되는데, 이 URL 을 통해 이용자들에게 배포한다.

3. Project 진행 일정

진행 목표	9 월		10 월					11 월		
	4	5	1	2	3	4	5	1	2	3
Model building & Training 구현					중간고사					
Utility & Inference 구현										
디버깅 & 기능 추가										
Pytorch 와 비교 & 개선										
Documentation 제작										

4. 용어 정리

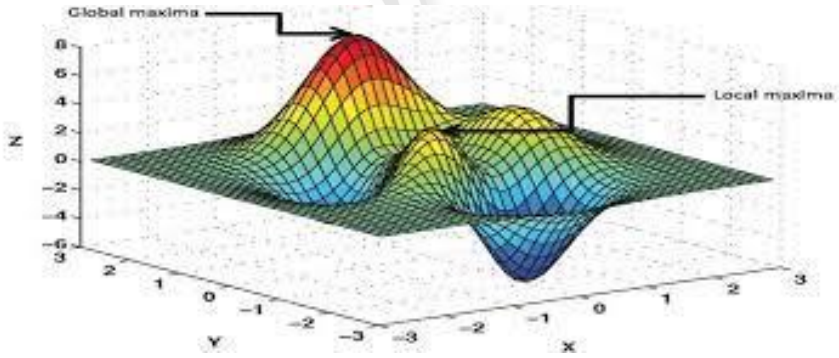
- DeepLearning Framework

  딥러닝 모델을 개발하고 학습시키기 위한 도구 및 라이브러리의 집합이다. Framwork 를 사용하면 복잡한 수학적 연산이나 최적화 과정을 직접 구현할 필요가 없이 모델을 쉽게 구축하고 학습시킬 수 있다.

  DeepLearning Framework 에는 Pytorch, TensorFlow, Keras, Caffe 등이 있다.

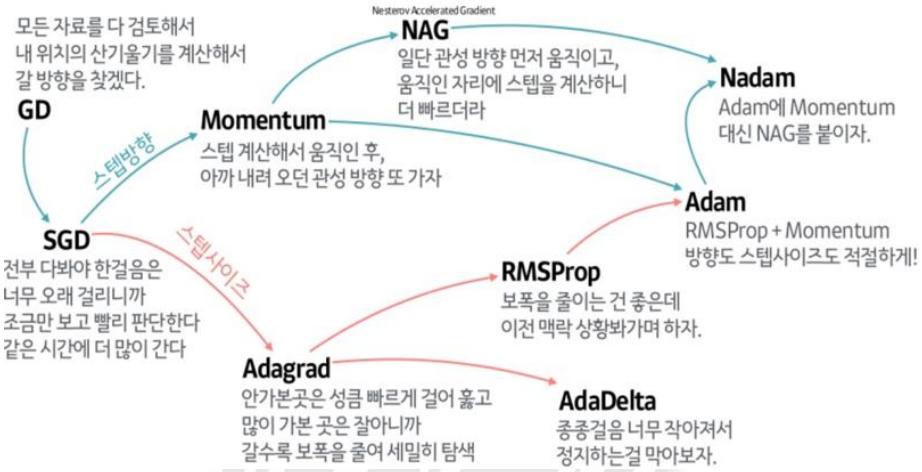
- Loss Function

  Loss Function 는 모델의 예측 값과 실제 값 사이의 차이를 계산하는 함수이다. 이 차이를 최소화하는 하는 방향으로 모델의 학습가능한 파라미터를 업데이트하게 되며, 이 과정을 통해 모델이 학습됩니다.



- Optimizer

Optimization 은 loss function 의 결과 값을 최소화하는 모델의 파라미터를 찾는 것을 의미한다. Optimizer 는 모델의 파라미터를 업데이트 하는 알고리즘을 의미한다.



- Inference

모델이 학습된 후에 새로운 데이터에 대한 예측을 수행하는 과정을 의미한다. 즉, 모델이 학습 과정에서 학습 데이터셋을 사용하여 최적화가 된 후, 학습하지 않은 데이터에 대해서 예측하는 단계를 “inference”라고 한다.

- OOP(Object-Oriented-Programming)

컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, ‘객체’들의 상호작용으로 프로그램 로직을 구성하는 프로그래밍 방식이다.

- Preprocessing

모델을 학습하기 전에 데이터를 처리하여 모델에 적합한 형태로 변환하는 과정을 의미한다. 모델의 성능을 크게 향상시킬 수 있으며, 올바른 preprocessing 없이는 학습이 제대로 이루어지지 않을 수 있다.

- pickle

표준 라이브러리 중 하나로 파이썬 객체를 저장하거나 역으로 불러올 때 할 때 사용한다. .pickle 로 파이썬 객체를 파일에 저장하거나 전송할 수 있게 한다.

- Streamlit

Streamlit 은 웹, 앱에 대한 지식 없이도 Python 을 이용해서 빠르게 간단한 웹, 앱을 만들 수 있게 해주는 라이브러리이다.

- Streamlit Cloud

Streamit app 을 호스팅하고 공유하기 위한 플랫폼이다. Streamit Cloud 를 통해서 간단하게 배포를 할 수 있다.

- Documentation

특정 제품, 서비스, 소프트웨어, 하드웨어 또는 시스템에 대한 설명서나 사용자 매뉴얼을 의미한다. 사용자나 개발자가 해당 제품이나 시스템을 이해하고 올바르게 사용하도록 도와주는 중요한 자료이다.

- Logging

학습, 평가 중에 발생하는 다양한 정보와 meric 을 기록하고 관리하는 과정을 의미합니다. Logging 은 모델의 성능을 monitering 하고, 문제점을 파악하기 위한 필수적인 과정이다.

- Evaluation

모델의 성능을 평가하는 과정을 의미한다. 딥러닝에서는 모델의 성능을 정량적으로 평가하기 위해 다양한 지표와 방법을 사용한다.

- Learning Rate

학습과정에서 최적화 알고리즘을 사용할 때 모델의 가중치를 업데이트 할 때 사용되는 hyperparameter 이다. Learning Rate 는 얼마나 가중치를 크게 변경할지 결정하며 학습의 속도와 안정성에 큰 영향을 준다.

- VSCode

Microsoft 에서 개발한 IDE(통합 개발 환경)이다. 다양한 프로그래밍 언어를 지원하며 속도, 확장성으로 인해서 전 세계 많은 개발자들 사이에서 인기가 있다.

- Anaconda

머신러닝과 딥러닝, 데이터 분석등의 작업을 수행하기 위한 다양한 라이브러리를 포함하며 패키지 관리와 환경 관리를 쉽게 해주는 도구를 제공한다. Conda 라는 가상환경 관리자를 포함하고 있어서 프로젝트 별로 독립된 환경을 생성하여 작업할 수 있다.

- Git

  버전 관리 시스템이다. 개발할 때 코드의 변경 사항을 추적하고 협업하는 데 많이 사용된다. Snapshot 방식으로 버전을 관리해서 빠른 속도가 특징이다.

- Github

  Git 을 기반으로 하는 웹 기반 호스팅 서비스이다. Github Action 이나 pull request 등 다양한 기능을 제공하고 협업관리를 도와준다. 전 세계 개발자의 커뮤니티 기능도 한다.

- Dataloader

  Pytorch 에서 사용하는 class 이다. 데이터셋에서 데이터를 효율적으로 불러오고 batch 처리를 도와주는 역할을 한다. Pytorch 에서는 데이터 로딩을 병렬로 처리한다.

- Train, Valid, Test set

학습하고 평가할 때, 전체 데이터 셋을 여러 부분으로 나눈다. Train set 은 학습할 때 직접적으로 사용되며 Valid set 은 학습 중인 모델의 성능을 평가할 때 사용한다. Test set 은 최종적으로 학습된 모델의 성능을 평가할 때 사용한다.

- 추상클래스

추상 클래스는 객체 지향 프로그래밍에서 중요한 개념 중하나로, 직접 instance 가 될 수 없는 class 이다. 대신 다른 class 들이 상속받아 기본 틀로 작동한다.

- Tuple

Tuple 은 list 와 유사하지만 내용이 변경될 수 없으며 순서 또한 변경될 수 없는 데이터 구조이다.

```
my_tuple1 = (18, 21, 59)

my_tuple2 = (4, 'banana', 3.14)
```

- Batch Size

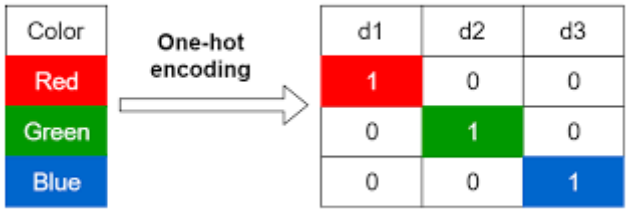
Batch Size 는 한번의 itreration 에서 사용되는 data 의 개수를 의미한다. Batch Size 가 학습의 효율성과 성능의 큰 영향을 준다.

- Generator

Iterator 를 생성하는 방법이다. 일반적인 함수와는 다르게 실행될 때마다 값을 하나씩 반환하고, 다음 값이 필요할 때까지 실행 상태를 유지한다.

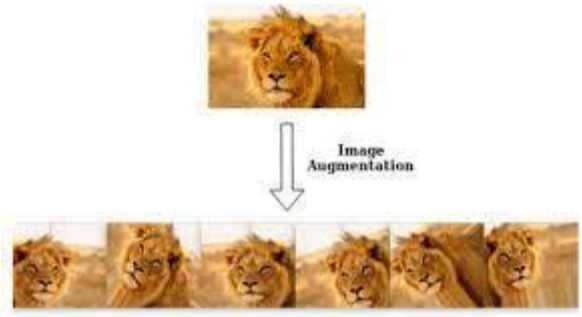
- One-hot Encoding

각 class 를 이진 코드로 변환하여 벡터 형태로 표현하는 방법이다. 이 방법을 사용했을 때 벡터는 주로 0 으로 구성되며, 해당 class 를 나타내는 한 위치만 1 의 값을 가진다.



- Augmentation

기존의 데이터를 다양한 방법으로 변형하여 데이터의 다양성이나 양을 인위적으로 늘리는 방법을 의미한다. 데이터에 다양성을 부여하여 모델의 overfitting 을 방지하고 일반화 성능을 높일 수 있다.



- Albumentations

Data Augmentation 을 위한 파이썬 라이브러리이며 다양한 증강 기법이 구현되어있다. CPU 에서도 빠르게 동작하며 labeling 을 agmentation 에 맞게 자동으로 바꿔주는 기능 또한 있다. ex) detection bbox annotation 변환.

Albumentations

Fast and flexible image augmentations

https://albumentations.ai

- Layer

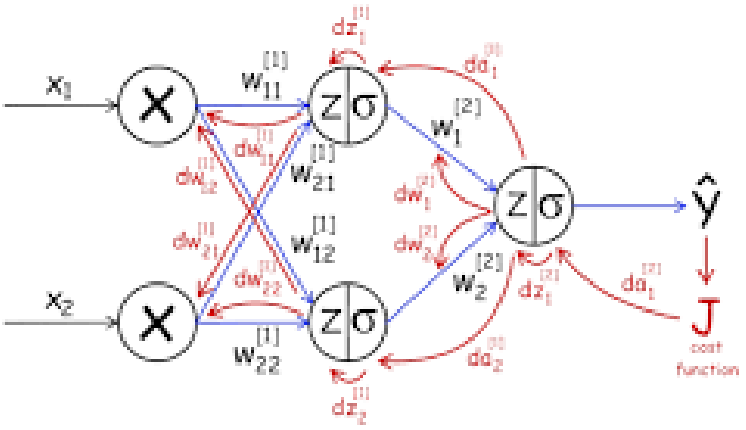
딥러닝 아키텍처에서 핵심적인 구성요소이다. 신경망의 각 단계나 층을 나타낸다.

- Forward pass

딥러닝의 연산 과정 중 하나로 입력 데이터가 layer 를 통과하면서 최종 출력까지 계산되는 과정을 의미한다. 학습 과정 중 모델의 prediction 을 생성하거나 test 시에 모델의 성능을 평가할 때 사용된다.

- Backward pass

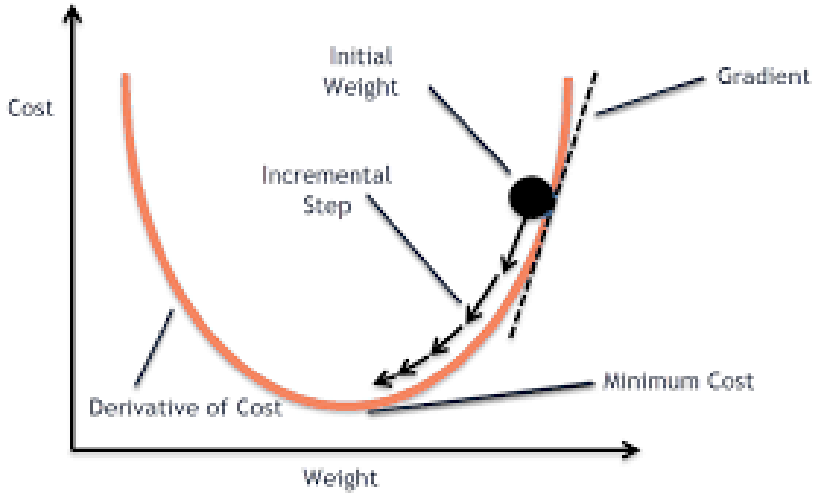
역전파 라고도 하며 딥러닝 학습에서 핵심적인 요소이다. Forward pass 에서 계산된 loss 를 바탕으로 파라미터를 조정하기 위해 사용된다. Prediction 과 label 사이의 차이를 나타내는 loss 의 gradient 를 계산하고 그 gradient 를 바탕으로 출력 layer 에서 입력 layer 방향으로 chain rule 을 사용하여 각 레이어의 gradient 를 계산하고 각 가중치와 편향에 대한 loss 의 gradient 을 구한다. 계산된 gradient 를 사용하여 가중치와 편향을 업데이트 합니다. 이과정을 backward 라고 한다.



Backpropagation

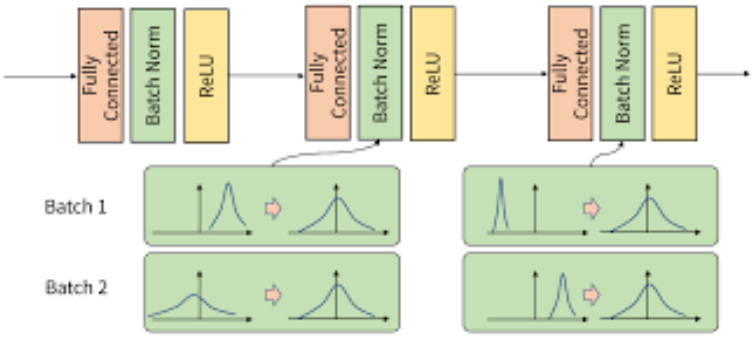
- Gradient

다변수 함수의 미분을 벡터형태로 나타낸 것이다. 함수의 입력에 대한 출력의 변화율을 나타내며 현재 지점에서 가장 빠르게 증가하는 방향과 크기를 제공한다. 따라서 gradient 의 반대 방향으로 학습해야 한다. 밑의 그림을 본다면 gradient 가 가장 빠르게 증가하는 방향과 크기를 나타내는 이유를 알 수 있다.



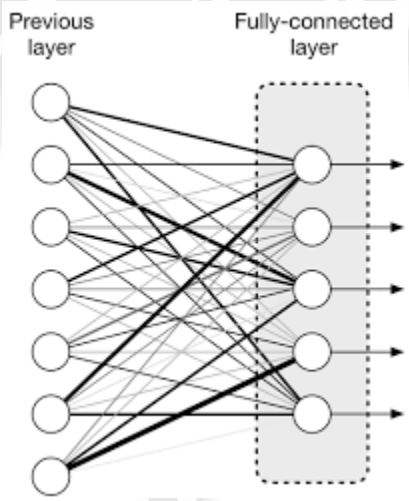
- Batch Normalization

각 레이어의 입력 분포를 안정화시켜서 학습을 더 빠르고 안정적으로 만드는 기법이다. 학생들에게 교재와 환경을 자꾸 바꾼다면 시험 결과가 좋지 않을 것이다. 하지만 같은 교재, 같은 환경에서 공부하면 이전보다 학습이 일관성을 가지게 되어 공부 효과가 좋을 것이다. 여기서 batch normalization 은 학생들의 교재와 환경을 통제할 수 있는 선생님이라고 할 수 있다. 이와 같이 batch normalization 은 각 layer 에 input data 의 분포를 일관되게 유지하는 역할을 한다. batch normalization 은 기본적으로 입력을 nomalization 하여 평균이 0 이고 표준편차가 1 이 되도록 만드는 것이다. 하지만 이렇게 nomalization 만 한다면 일부 정보나 네트워크의 표현력이 제한될 수 있다. 따라서 batch norm 은 scale 과 shift 라는 학습 가능한 파라미터를 도입했다. 각 feature 마다 서로 다른 scale 과 shift 파라미터가 있다. Scale 은 nomalization 된 데이터에 곱하는 계수를 말하고 shift 는 평행이동 하는 계수를 말한다. 즉, Scale 과 shift 의 목적은 해당 네트워크가 데이터의 최적의 scale 과 위치를 학습할 수 있도록 하는 것이다.



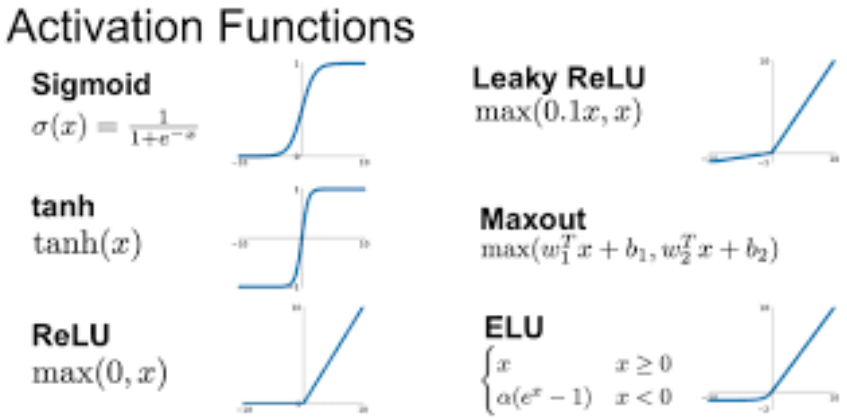
- Fc Layer

"Fully Connected Layer"의 약자로 Dense Layer 라고도 불린다. 이 Layer 의 모든 노드는 이전 layer 의 모든 노드와 연결 되어있다. 학습가능한 파라미터인 가중치와 편향을 가지고 있다. 기본적으로 연산은 행렬 곱과 덧셈으로 이루어진다.



- Activation Function

Activation Function 은 신경망에 적용되는 비선형 함수로 모델에 비선형성을 부여하여 좀 더 복잡한 학습을 할 수 있게 도와줍니다. 신경망에 activation function 이 없다면 비선형성을 부여할 수 없어서 선형 연산만 수행하게 되고 복잡한 문제를 해결할 때 성능이 떨어지게 된다.

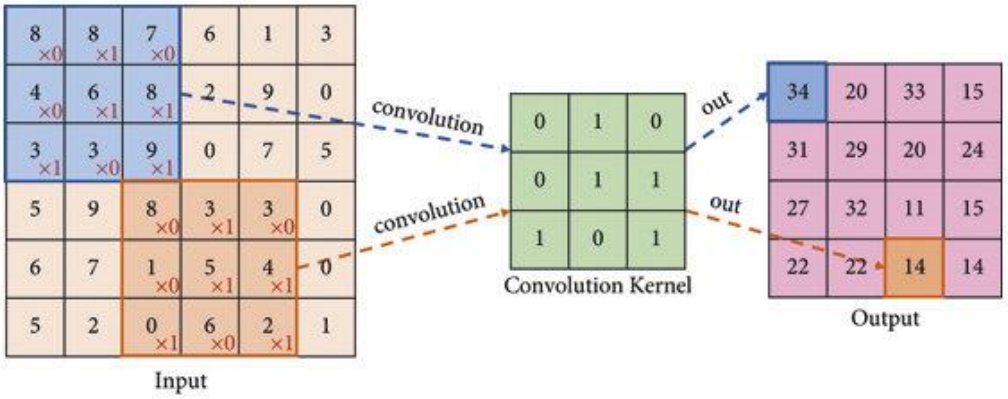


- 가중치 초기화

학습 시작 시에 가중치를 특정하게 바꿔주는 것을 의미한다. 초기의 가중치 초기화는 학습 속도와 학습 효율, 성능에 큰 영향을 줄 수 있다. 초기화가 잘못된다면 gradient vanishing 이다 gradient exploding 을 줄 수 있다.
흔하게 사용하는 가중치 초기화로는 S 자의 activation function(sigmoid 계열)을 사용할 때 효과적인 Xavier Initialization 과 Relu 계열 activation function 을 사용할 때 효과적인 He Initialization 등이 있다.

- Convolution, Kernel, Stride, Padding

이미지 처리와 딥러닝의 CNN 에서 사용되는 핵심 연산이다. Input data 에 필터(커널)을 적용하여 새로운 feature map 을 생성해는 과정이다.
Kernel 은 작은 크기의 행렬이며 특정 특징을 감지하는 역할을 한다. Kernel 과 input 의 행렬 곱이 다음 feature map 의 일부가 된다.
Stride 는 input data 위에서 kernel 이 이동하는 간격을 나타낸다. 만약 stride 가 1 이면 한 픽셀씩 이동하고 2 라면 두 픽셀씩 건너뛰며 이동한다.
Padding 은 input data 의 0 과 같은 추가적인 값을 붙이는 것을 의미한다. 주요 목적은 feature map 의 크기를 조절하거나 input data 의 가장 자리 특성을 잃지 않게 하는 것이다.



- He initialization

가중치 초기화 방법 중 하나로, Relu 계열의 activation function 과 사용하면 성능이 좋다. 평균 0, 분산은 $\sqrt{\frac{2}{n}}$ 인 정규분포로 가중치를 초기화를 한다. N 은 이전 layer 의 뉴런 개수이다.

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right)$$

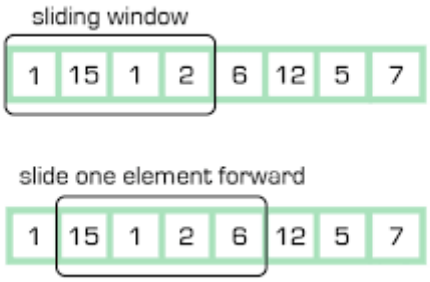
- Xavier Initialization

가중치 초기화 방법 중 하나로 S 자 형태의 activation function 과 함께 사용되었을 때 성능이 좋다. S 자 형태의 activation function 은 분산이 작아야 기울기가 1 에 가까워져서 gradient vanishing 이 일어나지 않기 때문이다.

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}}\right)$$

- Sliding window

특정한 크기와 모양의 "윈도우"를 가지고 입력 데이터를 순차적으로 탐색하면서 정보를 추출하는 방법을 나타낸다. Input data 의 시작점에서 시작하여 윈도우를 한 단계씩 움직이며 데이터를 탐색한다. 예를 들어 convolution 연산에서 sliding window 방식으로 Kernel 을 움직인다.



- Down Sampling

Feature map 을 줄여서 연산량을 감소시키고 저장 공간을 절약하는 기법이다. 데이터 크기를 줄이기 때문에 정보가 손실될 수 있지만 연산량을 줄이고 overfitting 을 방지할 수 있다.

- Relu

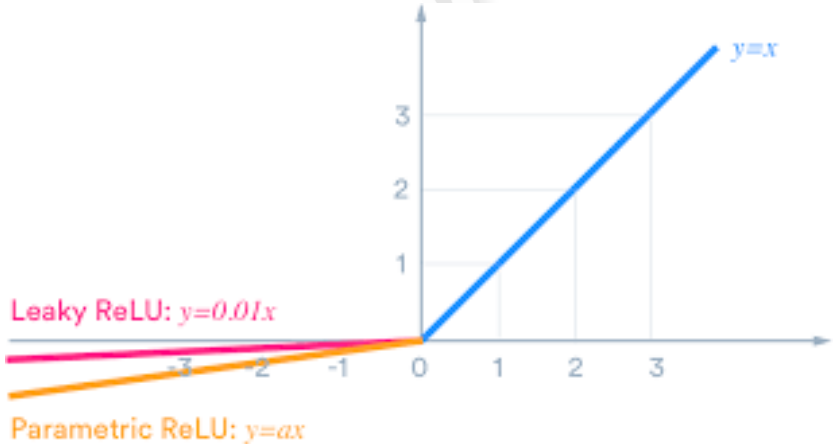
주로 사용되는 activation function 중 하나이다. 간단하면서도 효과적으로 모델에 비선형성을 부여한다. 또한 sigmoid 나 tanh 는 큰 양수값이 들어오게 되면 기울기가 0 에 가까워져 gradient vanishing 이 발생할 수 있는데 Relu 양수라면 기울기가 1 이기 때문에 gradient vanishing 문제가 일어나지 않는다. 하지만 음수의 값이 input 으로 들어오게 되면 gradient 가 0 이 되어 Dead Neuron 문제가 발생할 수 있다.

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

- Leaky Relu

Relu 의 발생한 Dead Neuron 문제를 방지하기 위해서 나온 activation function 이다. α 는 0.01 또는 0.1 과 같은 작은 값으로 설정되며 조절이 가능하다. Leaky Relu 역시 비선형성을 모델에 부여해주고 양수 입력에서는 Relu 의 이점을 유지한다.

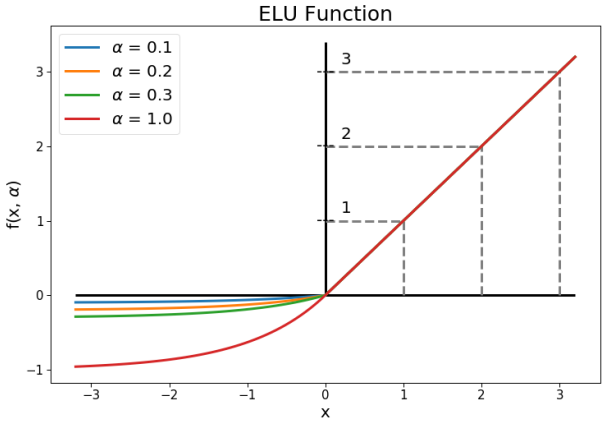
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$



- Elu

Elu 또한 Relu 의 변형으로 비선형성을 부여해주고 Dead Neuron 현상을 방지해준다. 0 에 가까운 음수 입력에 대해서도 1 에 가깝게 기울기를 가지고 있어서 음수 값에 대한 더 많은 정보를 전달할 수 있다. 또한 값이 많이 작아지는 음수 값에 대해서는 기울기가 작아서 gradient vanishing 문제를 완화시킨다.

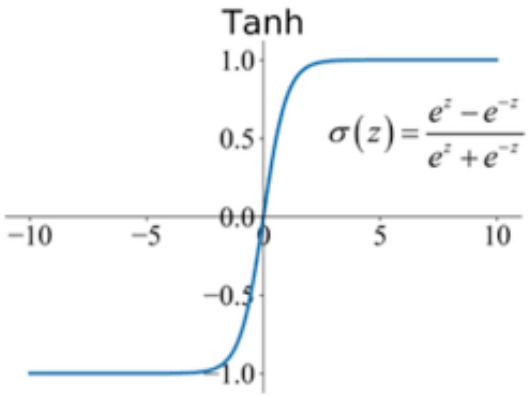
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$



- Tanh

S 자 형태의 곡선을 가지는 activation function 입니다. Input 값을 [-1,1] 범위로 출력한다. 이 함수도 역시 모델에 비선형성을 추가한다. 하지만 정규화 하지 않은 데이터가 들어올 경우 값이 커져 기울기가 0 이 될 수 있기 때문에 gradient vanishing 을 주의해야한다.

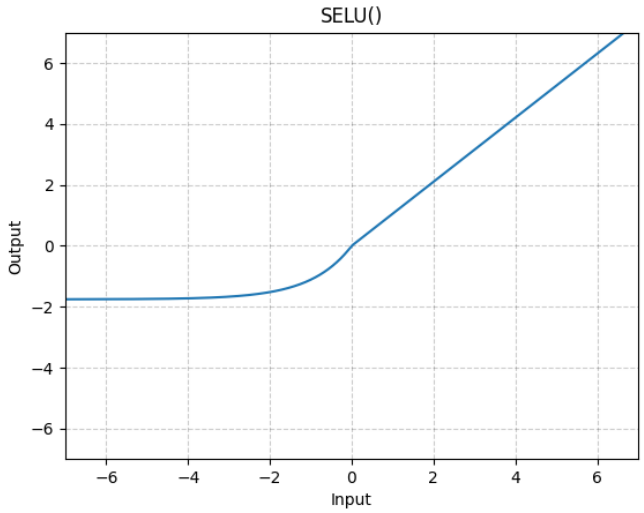
$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



- Selu

Elu 에서 파생되어 나온 activation 이다. Fc layer 만 쌓아서 신경망을 만들고 모든 activation function 을 사용하면 네트워크가 자기정규화(네트워크의 출력이 입력과 유사한 평균과 분산을 갖도록 함)가 된다고 저자는 주장한다.
Elu 에서 파생되어 나온 만큼 Elu 의 이점을 갖되 input 이 Standardization 이 되어있을 때 더욱 효과적으로 작동한다고 알려져 있다. λ는 보통 1.0507 로 설정하며 α는 보통 1.6733 으로 설정한다.

$$selu(x) = \begin{cases} \lambda \cdot x & \text{if } x > 0 \\ \alpha \cdot (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



- Nomalization

데이터의 값을 0~1 내로 조정하는 과정을 의미한다. 주요 목적은 data 의 scale 을 조정하여 다양한 특징들 간의 scale 차이를 줄이는 것이다. 이미지 데이터에서는 255 로 나누는 min-max scaling 이나 평균과 표준 편차를 사용하여 픽셀 값을 정규화하는 Strandardization 이 있다

- 시프팅

데이터의 값을 일정한 값만큼 이동시키는 것을 말한다. 일반적으로 Standardization 을 수행하기 전에 데이터의 평균을 맞추기 위해 시프팅을 사용할 수 있다.

- Scheduler

학습 중에 Learning rate 를 조절하는 도구이다. 일반적으로 Learning rate 를 점차 감소시키지만 특정한 값에 수렴하게 Learning rate 를 올리는 Warm Up 방식도 사용한다.

- Early Stopping

Overfitting 을 방지하기 위한 방법 중 하나이다. Overfitting 은 너무 과도하게 data 에 대해 학습되어 test data 에 대한 성능이 떨어지는 현상인데 Early Stopping 은 학습 과정을 주기적으로 모니터링하여 loss 의 값이 더 이상 떨어지지 않을 때 학습을 종료하는 방법이다.

- MSE(Mean Squared Error) Loss

평균 제곱 오차의 약자이며 가장 기본적인 loss 이다. MSE 는 label 과 prediction 의 차이를 제곱한 평균을 구한 것이다. 하지만 제곱을 하기 때문에 이상치에 민감하다.

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Binary Cross Entropy Loss

이진 분류 문제에서 많이 사용하는 Loss 함수이다. Label 의 확률과 prediction 의 확률 차이를 계산한다. Prediction 의 확률이 높을수록 식은 0 에 가까워진다. 일반적으로 logit 이 아닌 확률 분포를 input 으로 받는다.

$$\mathrm{BCE}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

- Logit

Logit 은 모델의 출력 중 하나로, input 에 대한 model 의 prediction 을 나타낸다. 보통 classification task 에서 softmax 함수를 거쳐서 class 의 확률 값으로 바뀐다.

- Sigmoid

Sigmoid 함수는 주로 이진 분류에서 사용됩니다. 실수 값을 input 으로 받아서 0 과 1 사이의 값으로 변환한다. S 자 형태의 곡선을 가진다. 입력이 크면 1 에 가까워지고 작으면 0 에 가까워지므로 이진 분류에 대해서 확률 값을 나타내기 적합하다.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Cross Entropy Loss

이진 분류 문제, 다항 분류 문제에서 많이 사용하는 Loss 함수이다. One hot-Encoding 되어 label 이 주어지므로 label 의 index 에는 1 이 들어있기 때문에(나머지는 0) prediction 이 확률로 주어질 때 label 에서 1 이 담겨있는 index 에 나온 확률이 1 에 가까울수록 loss 는 줄어든다.

$$H(p, q) = -\sum_i p_i \log(q_i)$$

- Focal Loss

Class Imbalance 문제를 다루기 위한 Loss Function 중 하나이다. Class 간의 가중치를 도입한다. α_t는 class 별로 다른 가중치를 나타냅니다. 모든 class 에 대해서 다른 값을 가질 수도 있다. 올바르지 못한 prediction 을 했을 때 민감하게 반응해야 하기에 γ는 보통 1 이상의 값을 갖는다.

$$\mathrm{FL}(p_t) = -\alpha_t(1 - p_t)^{\gamma} \log(p_t)$$

- Class Imbalance

classification 에서 특정 class 의 data 가 부족하거나 많은 경우를 가리키는 현상이다. Focal Loss 의 가중치를 이용해서 해결하는 방법도 있고 offline augmentation 을 사용해서 data 자체를 늘리는 방법 또한 있다.

- SGD

최적화 알고리즘 중 하나이다. Loss 를 최소화하는 방향으로 model 을 학습하는 역할을 한다. 사실 무작위로 1 개를 뽑아서 forward pass 를 거치고 loss function 에서 gradient 계산하고 그 gradient 를 시작으로 update 해야 진정한 SGD 이지만 mini-batch GD 도 SGD 라고 부르기도 한다. Mini- batch GD 는 데이터를 1 개를 뽑지 않고 batch size 만큼의 데이터의 사용하며 이 data 들의 loss 평균을 통해서 update 한다.

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$$

- Adam

현재 가장 많이 사용되는 형태의 optimizer 이며 moment optimizer 와 RMSprop 의 parameter 를 반영하여 발전시킨 형태이다. $\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon}$ 에서 분자는 gradient 의 이동평균을 나타내며 이전 gradient 들에서 오는 관성을 가진 momentum 변수이다. 분모는 gradient 제곱의 이동 평균이며 RMSprop 과 같이 이동 평균이 너무 크다면 하향 조정해서 update 를 하고 너무 작다면 상향 조정해서 update 를한다.

$$\theta_{t+1} = \theta_t - \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$$

- RMSprop(Root Mean Square Propagation)

최적화 알고리즘이며 v_t 는 이동 평균이며 각 파라미터에 대한 누적 gradient 의 제곱의 평균이다. β 는 이동 평균을 계산할 때 사용되는 지수 감소 비율이다. 일반적으로 0.9 와 같은 값이 사용된다. 이 값은 이전 gradient 의 값을 얼마나 고려할지 결정한다. $\nabla J(\theta_t)$ 은 θ_t 의 gradient 를 나타낸다.

$$v_t = \beta v_{t-1} + (1-\beta)\nabla J(\theta_t)^2$$

$\frac{\eta}{\sqrt{v_t+\epsilon}}$ 이 update 의 크기를 결정한다. 분모는 이동평균에 아주 작은 수를 더해서 루트를 씌운 값이다 이것이 의미하는 것은 이동 평균을 고려해서 이동 평균이 너무 크다면 하향 조정해서 update 하고 너무 작다면 상향 조정해서 update 를 하는 것을 나타낸다.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \nabla J(\theta_t)$$

- AdamW

AdamW 는 Adam 에서 파생된 optimizer 로 $(1-\lambda\alpha)\theta_t$ 항이 특징이다.
이 항은 weight 를 감쇠 시키는 역할을 하는데 이는 모멘텀의 감소로 볼 수 있다. 즉, 모멘텀을 감소시켜 update 의 안정성을 높였다.

$$\theta_{t+1} = (1-\lambda\alpha)\theta_t - \alpha\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon}$$

- Weight Decay

AdamW 에서 나온 weight 를 감쇠 시키는 기법이다. 모멘텀을 감소시켜 update 를 안정시키는 효과를 줄 수 있다.

- Epoch

Epoch 는 모델이 train data 전체를 한번 통과하는 것을 나타낸다. Epoch 는 학습에 중요한 parameter 이며 validation 을 통해서 적절한 epoch 를 찾는 것이 중요하다.

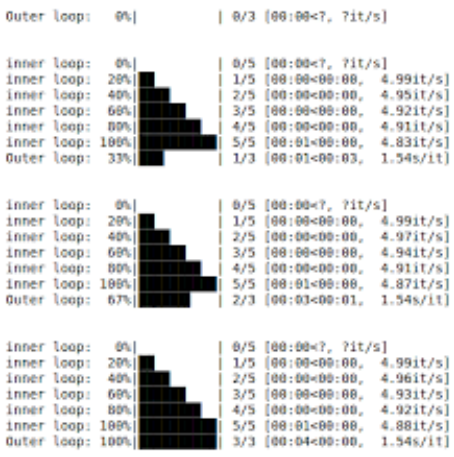
- Wandb

Weights and Biases 를 줄인 말로, 모델의 실험 및 monitering 을 위한 tool 이다. Log 를 통해 실험 결과를 시각화 할 수 있으며 협업하며 실험을 관리할 때에도 많이 사용된다.

Weights & Biases

- tqdm

python 에서 사용되며 진행률 표시를 하기 위한 라이브러리이다. 사용할 시 밑의 그림처럼 진행률을 표시할 수 있다.



- Early Stopping

모델의 학습 과정을 모니터링 하고 학습을 조기에 중지하는 기법이다. Early Stopping 을 사용한다면 overfitting 을 방지할 수 있고 train resource 를 아낄 수도 있습니다. 조기 종료 조건은 정의하기 나름이나 valid set 에 대한 성능이 일정 epoch 동안 개선되지 않을 때를 조건으로 한다.

- Tensor

Tensor 는 행렬 데이터를 표현할 수 있으며 DeepLearning 에서 핵심적인 데이터 구조 중 하나이다. 다양한 DeepLearning Framework 에 사용되고 있다.

- Accuracy

정확도는 전체 예측 중에서 맞춘 예측의 비율을 백분율로 나타낸 값이다. Metric 으로써 직관적이고 이해하기 쉽지만 class Imbalance 가 있는 문제라면 적절하지 않을 수 있다.

- Precision

Precision 은 모델이 positive class 로 예측한 data 중에서 실제로 positive 인 비율을 나타냅니다. 다중 분류에서는 각 class 별로 TP, FP 를 계산해서 (총 TP)/(총 TP + 총 FP)를 계산한다.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- Recall

Recall 은 positive class 중에서 model 이 positive 로 정확하게 예측한 비율을 나타낸다. Precision 과 마찬가지로 다중분류에서는 (총 TP)/(총 TP +총 FN)으로 계산한다.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- F1 score

Precision 과 Recall 의 조화평균을 나타내는 지표로 Precision 과 Recall 을 모두 반영하여 metric 으로서 기능한다. 다중 분류의 경우 각 class 별 F1 score 를 전부 더한 후 class 개수로 나눈다.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. 참고 자료

[1] Pytorch documentation

[2] https://eehoeskrap.tistory.com/430 (batch norm)

[3] https://gaussian37.github.io/dl-concept-batchnorm/ (batch norm)

[4] 밑바닥부터 시작하는 딥러닝 1 권 – 사이토 고키

[5] https://hwk0702.github.io/ml/dl/deep%20learning/2020/07/09/activation_function/ (activation)

[6]https://realblack0.github.io/2020/03/29/normalization-standardization-regularization.html (normalization)

[7] https://reniew.github.io/13/ (weight initialization)

[8] https://acdongpgm.tistory.com/215 (weight initialization)

[9] https://go-hard.tistory.com/64 (drop out)

[10] https://dojang.io/mod/page/view.php?id=2406 (iterater)

[11] https://blog.naver.com/ttddcc119/223206459493 (gradient vanishing)

[12] https://data-newbie.tistory.com/262 (selu)

[13] 혁펜하임 AI Deepdive

[14] 딥러닝 pytorch 교과서 – 서지영