활성화 함수의 중요성의 고찰
자율주행 드론 구현

INN(INJE NEURAL NETWORK)

● 목차(20pt): 목차와 목록페이지 표시

일반도서, 학위논문, 학술지 논문, 인터넷 사이트 주소와 사이트명을 모두 기재한다.

▣ 초록(분량-300자 안팎으로 요약) : 11pt, 진하게

Ⅰ. Activation function은 필요한가(with 자율주행드론)

1. 문제 인식

우리는 흔히 딥러닝 모델을 구현할 때 activation function을 자주 이용한다. activation function이 모델의 성능을 향상시킬 것이라는 은연중의 믿음이 있기 때문이다. 그러나 과한 activation function의 사용은 컴퓨터의 계산 복잡성을 증가시키기만 할 뿐 모델의 성능을 크게 향상시키지 않을 수 있다. 때문에 우린 activation function의 성능을 비교하고 여러 데이터셋에서의 activation funciton의 성능 관여도를 분석, activation function의 필요성을 탐구하고자 한다. 그리고 연구 결과를 이용해 자율주행 드론을 제작하고자 한다.

2. 연구 방법

1) 연구 대상 (4칸 스페이스, 글자크기 12p, 진하게)

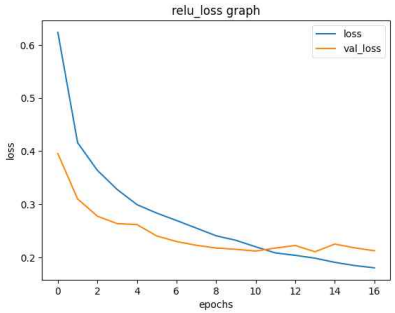
- activation function
- convolution network(Deep Learning)

2) 연구 방법

- 테스트 데이터 수집 후 간단한 데이터 전처리를 수행한다. 우선 convolution network를 기준으로 성능을 비교할 것이기 때문에 image데이터를 수집한다. 흑백 이미지를 이용해 연산량을 줄여 계산 효율성을 높이고 0~255까지의 범위를 표준화(Standart Scaling)하여 0~1까지의 범위로 만든다(데이터 전처리). 그 후 tensorflow의 keras 패키지를 활용하여 아래 그림과 같은 모델 구조를 만든다.

(예상 모델 구조)

그 후 model의 loss-epochs 그래프를 그려 모델의 성능을 비교한다. (아래 그림은 예시)



- Image 데이터에 대해서 분석을 끝마치면 주기성을 가진 연속형 데이터를 가지고 RNN 모델을 활용하여 순서가 있는 즉 주기성이 있는 데이터셋에선 activation function이 어떤 영향을 끼치는지 연구, 분석한다. 아무래도 주기성이 있는 데이터셋에선 시계열 데이터셋에서 주로 사용하는 time stemps 전처리 방식을 자주 사용하니 여기서도 그 방식을 사용하였다. 이 개념은 여기서 설명하긴 어려우므로 데이터들의 순서를 매겨준다고 생각하면 된다.

그 후 코드론을 사용하여 opencv mediapipe 페키지를 활용해 손가락의 방향에 따라 움직이는 자율주행 드론을 개발한다.

3) 연구 방향

현재 연구는 activation function의 필요성과 activation function을 커스터마이징 하는 것이 과연 성능의 큰 영향을 주는지 연구하였다. 이 연구를 통해 딥러닝 과정 중 연산량의 큰 비중을 가지는 activation function의 필요성을 알아볼 수 있다. 이 연구는 데이터 수집 > 데이터 전처리 > 모델 구축 > activation function에 따른 성능 비교 > 연구결과 도출 > 드론 제작의 순을 거친다. 최종 결과론 activation function이 모델 성능에 생각보다 큰 영향을 끼쳤다.

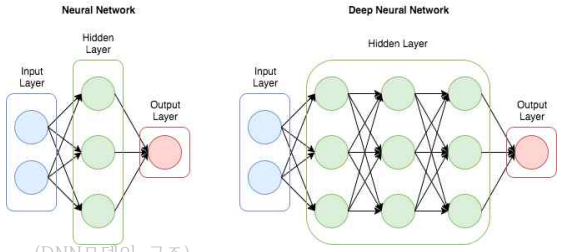
Ⅱ. 본론

1. 딥러닝의 개념

(1) CNN모델: 딥러닝 모델은 여러 데이터의 특성들 사이에서 특징을 추출해 다른 데이터를 예측하는 작업을 수행한다. CNN모델은 이미지 데이터를 분류하고 예측하는데에 최적화 되어있는 알고리즘이다. CNN모델의 특징은 이미지 데이터 안에서 각 클래스별 특징을 찾아낸다는 것이다. 예를들어 개와 고양이를 분류하는 모델을 개발한다면 내가 개발한 모델은 고양이의 수염, 고양이의 귀, 강아지의 꼬리 등의 우리 눈에도 두드러지게 보이는 특성들을 학습할 것이다. 아래 그림은 CNN모델의 학습과정의 예이다.

(고양이의 특성을 학습하고 있다.)

(2) DNN모델: DNN모델은 CNN모델과 달리 이미지 데이터에선 큰 성능을 보이진 못하지만 CNN, RNN, GAN 모델 등 모든 모델의 기반기술이며 모든 모델에 필수적으로 쓰이는 기술로써 우리가 자주 얘기하던 인공신경망 즉 우리 인간의 뉴런 모양을 모방한 모델이 바로 DNN모델인 것이다. DNN모델은 다중분류, 이진분류, 연속형 데이터, 회귀문제 등 다양한 데이터에 범용적으로 사용될 수 있으며 인공지능의 기초 기반기술이라고 얘기할 수 있다.

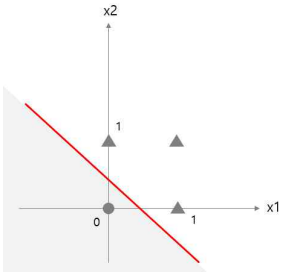


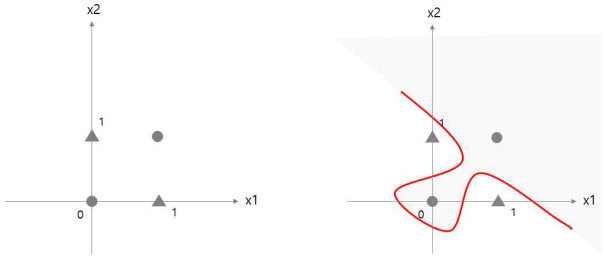
(DNN모델의 구조)

2. activation function의 개념

(1) activation function의 역할

딥러닝은 데이터들의 관계를 비선형적으로 학습해야 한다. 유명한 문제인 XOR문제로 예시를 들어보자. 아래 그래프에선 ▲모양을 분류하기 위해선 선형적인 관계로도 충분히 분류할 수 있다. (다항함수는 선형적인 관계(y=ax^n+…+b)꼴로 표현 가능). 때문에 activation function이 크게 중요하지 않을 것이다.



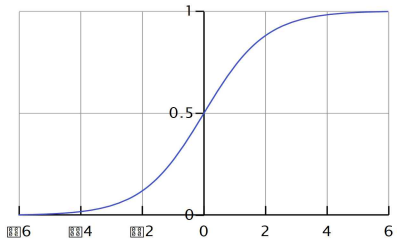


위 그래프에선 우리가 위에서 사용한 직선으로만 해당 데이터들을 분류하는것엔 무리가 있을 것이다. 그러나 직선이 아닌 곡선을 이용한다면 위 그래프의 데이터를 충분히 분류할 수 있을 것이다. activation function이 하는 일은 이러한 데이터들을 비선형적 관계로 분류할 수 있도록 만들어주는 것이다.

(2) activation function별 각각의 특징

activation function은 softmax, tanh, sigmoid, relu 함수 등 여러 가지 함수들이 있다.

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



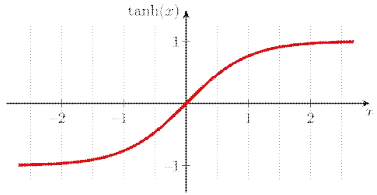
(sigmoid 함수 그래프 개형)

sigmoid 함수의 특징으로는 양수값만 가지고 점근선(y=0, y=1)을 가지며 특정 임계값을 지나면 값이 갑작스럽게 증가한다. 이 함수는 이진분류 문제에서 모델이 각 클래스 별 확률을 반환할 때 그 확률을 0~1 사이로 변환해줌과 동시에 반환된 확률을 증폭시켜준다.

- softmax function : $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ $OR\,\sigma(z)_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{K} e^{-\beta z_j}}$ $OR\,\sigma(z)_j = \frac{e^{-\beta z_i}}{\sum_{j=1}^{K} e^{-\beta z_i}}$ for $i = 1, \ldots, K$

소프트 맥스 함수는 K 개의 실수로 구성된 벡터를 K개의 가능한 결과의 확률 분포로 변환한다. 다중 분류 문제에서 자주 사용하는 softmax 함수는 모델이 출력한 여러개의 클래스별 확률 분포의 합을 1로 만들어준다. 예를들어 내가 사진 데이터를 입력받으면 이 사진이 가방인지, 신발인지, 상의, 하의, 목도리 등등 여러 의류의 사진을 분류하는 모델을 개발한다고 가정해보자. 모델이 학습한 후 내가 모델에게 사진을 입력하였더니 반환값이 [5, 0, 1, 0, 2, 0, 0, 1, 1] 이라고 생각해보자. 확률을 개산하기 위해선 저 분포의 합을 1로 만들어주어야 한다. 따라서 저 데이터들을 softmax function에 대입하면 [1, 0, 1/5, 0, 2/5, 0, 0, 1/5, 1/5]가 된다.

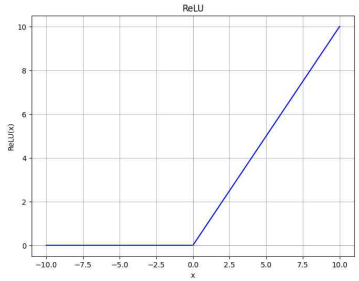
- tanh function(hyperbolic Tangent Function) : $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}$



(함수 그래프 개형)

이 함수는 sigmoid 함수와 비슷한 역할을 수행한다. 우리가 입력하는 실수값을 (−1, 1)로 압축시켜준다. sigmoid함수와의 차이점은 sigmoid함수는 양수의 값만 가지지만 tanh함수는 음수의 값도 가질 수 있다.

- ReLU function : $f(x) = \begin{cases} 0\ (x < 0) \\ x\ (x \geq 0) \end{cases}$



이 함수는 앞선 함수들에 비해선 식도 쉽고 그래프 개형도 단순하다. 그러나 이 활성화 함수는 CNN모델 정확히는 합성곱 연산을 수행할 때 강력한 성능을 보인다. 이 함수가 우리 팀이 activation function별 성능을 비교하게된 계기이다. 단순하게 생긴 겉모습에 비해 다른 함수들보다 훨씬 성능이 좋았기 때문이다.

3. 실험 수행

(1-1) 모델 구축-CNN(이미지 데이터)

모델은 tensorflow.keras 페키지를 사용하였고 서론에서 언급한 모델 구조 그대로를 이용하였다.

```
def test_activation(activation):
    model = keras.Sequential([
    keras.layers.Conv2D(64, kernel_size=(3, 3), activation=activation, input_shape=(28, 28, 1), padding='same'),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(32, kernel_size=(3, 3), activation=activation, padding='same'),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),

    keras.layers.Dense(100, activation=activation),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
    ])

    adam = keras.optimizers.Adam(learning_rate=1e-3)
    model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
    Early_stopping_cb = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)
    history = model.fit(train_input, train_target, epochs=100, batch_size=64,
                        validation_data=(test_input, test_target),
                        callbacks=[Early_stopping_cb])

    paint_graph(history, title='{}_loss graph'.format(activation)) # 해당 activation function의 loss-epochs 그래프
```

이용한 데이터는 저작권에 걸리지 않기위해 keras에서 제공하는 테스트용 데이터셋인 fashion_mnist 데이터셋을 활용하였다. 그 후 train_set과 validation_set으로 나누어주었다.

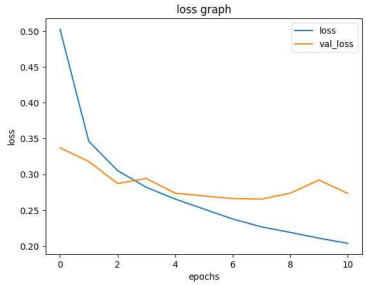
```
from sklearn.model_selection import train_test_split

input = input.reshape(-1, 28, 28, 1)
input = input / 255.0
train_input, test_input, train_target, test_target = train_test_split(
    input, target, test_size=0.2,
    random_state = 42)
```

데이터 전처리를 깊게 들어가진 않았고 사진 데이터가 (28, 28의 2차원 흑백 이미지여서 0~255의 값의 범위를 가지는 사진을 0~1사이의 범위로 줄여주는 standard scaling만 수행하고 사진 데이터의 복잡성 증가를 위해 2차원 이미지를 3차원으로 증강시켜주었다.

(2-1) 성능 비교(CNN모델)

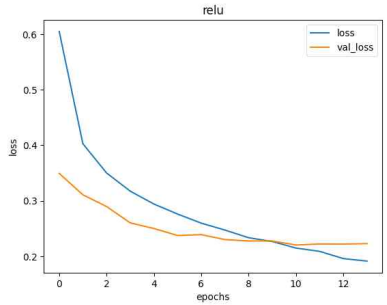
우선 activation function이 없을때에 모델의 성능값을 측정하고 각각의 activation function별로 성능값, 성능 그래프를 출력하여 대조군과 실험군을 생성한 후 각각의 결과를 비교, 분석하였다.

- No activation function



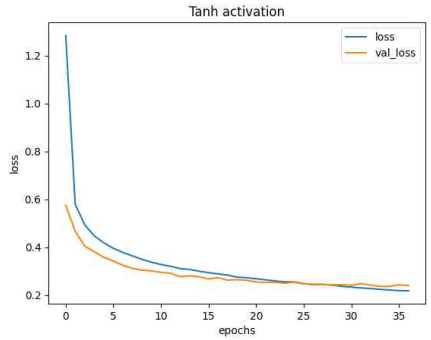
activation function이 없었을 때엔 epoch가 5~6번 정도 수행되었을 때 최적의 결과가 나왔다. 그래프를 보면 알 수 있듯 과대, 과소적합을 고려하고 보면 loss값이 0.25정도 인 것을 알 수 있다.

- ReLU activation function



우선 그래프가 두 개 모두 이상적으로 감소하고 있고 성능 자체로도 activation function을 넣었을땐 0.25정도였지만 여기선 loss값이 그보단 살짝 낮은 정도로 보이는 점에 주목하면 확실히 activation function이 있는 것이 성능에 더 좋은 영향을 주는 것은 분명해 보인다.

- tanh activation function

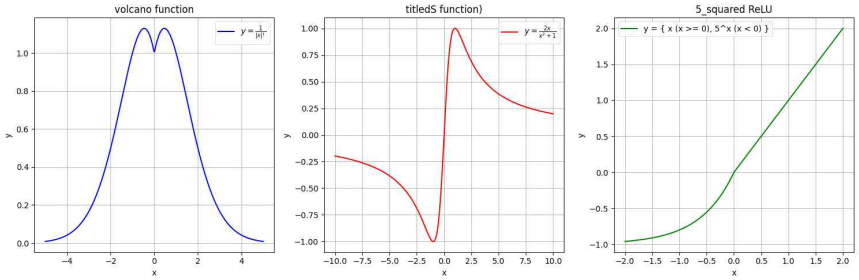


tanh 함수는 다른 함수들에 비해 val_loss와 train_loss의 값이 큰 차이 없이 완만하게 같이 감소한다. 이를 통해 이 함수는 다른 함수들과 달리 안정적으로 감소하는 loss값을 만들 수 있다는 것을 알 수 있다. 또한 activation function이 학습에 안정성을 추가한다는 점을 알 수 있었다.

(3-1) activation function customizing(CNN-이미지 데이터)

우린 위의 결과로 activation function이 학습에 꼭 필요하다는 것을 알가되었다. 그리하여 자주 사용하는 activation function이 아닌 우리가 직접 함수를 만들어서 activation function을 만들고 이를 학습에 사용한 후 성능을 비교해 보았다.

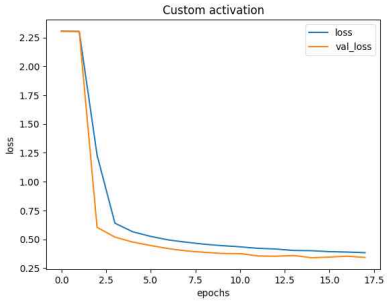
- **1. volcano function** : $y = \frac{1}{|x|!}$
- **2. tiltedS function** : $y = \frac{2x}{x^2+1}$
- **3. 5_squared ReLU function** : $y = \begin{cases} x & (x \geq 0) \\ 5^x & (x < 0) \end{cases}$



우리가 직접 만든 activation function은 이렇게 총 3개로 volcano function과 tilted function은 -2~2 까지의 범위를 매우 크게 변화시키도록 하였고 5_squared ReLU함수는 ReLU함수에서 음수값은 0으로 만들어 버리는 점이 음의 데이터 또한 어떠한 의미를 가질 수 있다는 생각 때문에 x<0 일때의 정보를 완전히 반환하진 않더라도 어느정도 남길 수 있도록 하기 위해 제작하였다.

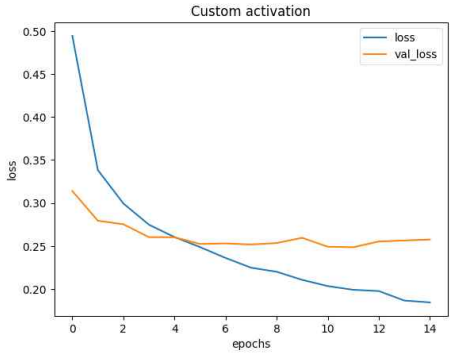
- **custom activaton 실험 결과**

1. volcano function



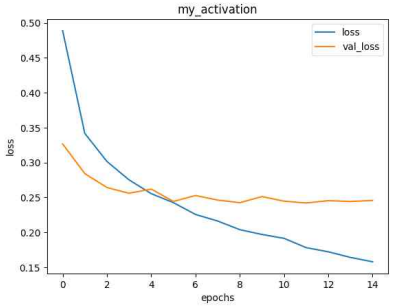
volcano function은 학습 결과가 activation function을 안넣었을때보다 더 성능이 낮게 나왔다. 이 결과를 통해 activation function이 확실히 학습에 큰 영향을 주지만 마냥 좋은 결과를 내어주진 않고 가끔은 단순하거나 선형적인 관계로도 충분히 좋은 성능을 낼 수 있다는 것을 알아내었다.

2. tiltedS function



이번 결과 또한 성능이 좋지 않게 나왔다. 위의 volcano function과의 공통점을 놓고보면 Image 데이터에서나 CNN모델에선 출력값이 낮은 (0.01, 0.1, 1 등)값을 증폭시켜서 학습에 영향을 주도록 하면 성능이 낮게 나오게 된다는 것을 알게 되었다. 우리가 예상하기론 Image 데이터에선 특징을 찾는 것이 중요한데 CNN모델에선 중요한 특징에 큰 값을 주기 때문에 큰 값을 증폭시키지 않고 작은 값을 증폭시키는 이 함수들이 잘 맞지 않는 것 같다.

3. 5_squared ReLU



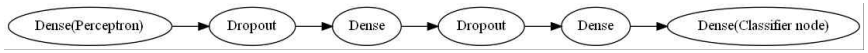
이번에도 성능이 그리 좋진 않지만 위 그래프들과 비교하면 그나마 괜찮다고 볼 수 있다. 그러나 ReLU함수와 비교하였을땐 확실히 성능이 떨어지므로 Image나 CNN모델에선 음의 값을 같는 데이터는 없애버리는 것이 학습에 조금 더 도움이 될 수 있다는 것을 알게되었다. 또한 과대적합이긴 하지만 그래도 모델이 위 그래프들의 train_loss 보단 너 낮은 train_loss를 가지기 때문에 충분히 activation function으로 활용할 수 있는 기회가 있을 수 있다고 생각하였다.

(1-2) 모델 구축(DNN모델, 연속형 데이터(회귀문제))

모델은 tensorflow의 keras.layers.Dense를 이용하여 DNN구조의 모델을 만들었다. 데이터는 저작권에 걸리지 않는 sklearn의 fetch_california_housing 데이터를 이용하였다. 그리고 이 데이터 또한 데이터 핸들링이 연구의 주 목적이 아니므로 최소한의 차원 축소와 standard scaling을 수행하고 train_set과 validation_set으로 나눠주었다.

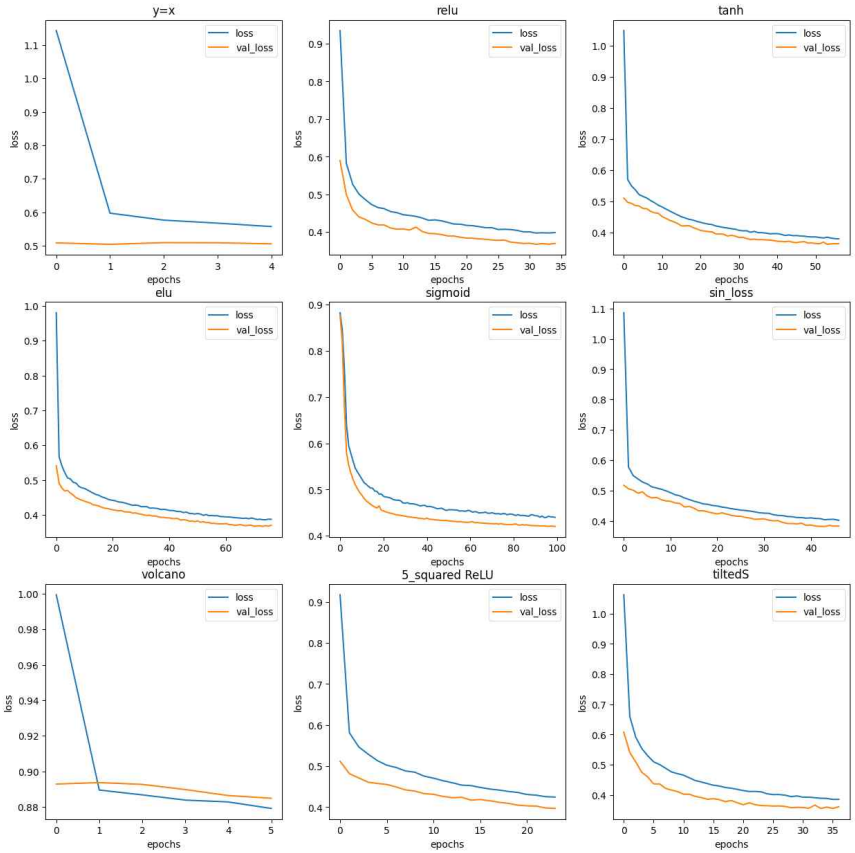
```
def test_activation(activation, title=None, learning_rate=1e-4, patience=3, epochs=100, return_list = False, no_limit=False):
    from graph import paint_graph
    model = keras.Sequential([
    layers.Dense(128, activation=activation),
    layers.Dropout(0.1),
    layers.Dense(64, activation=activation),
    layers.Dropout(0.1),
    layers.Dense(32, activation=activation),
    layers.Dense(1)
    ])

    adam = keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(loss='mean_absolute_error', optimizer=adam)
    Early_stopping_cb = keras.callbacks.EarlyStopping(patience=patience, restore_best_weights=True)
    if no_limit:
        history = model.fit(train_input, train_target, epochs=epochs,
                            validation_data=(test_input, test_target))
    else:
        history = model.fit(train_input, train_target, epochs=epochs,
                            validation_data=(test_input, test_target),
                            callbacks=[Early_stopping_cb])
    paint_graph(history, title=title) # 해당 activation function의 loss-epochs 그래프를 그려주는 함수

    if return_list:
        return {title:history}
```

(모델 구조)



(2-2) 성능 비교

이번에도 아까와 마찬가지로 activation function을 넣지 않은 모델 loss-epochs값을 우선 출력하였다. 그 후 여러 activation function들과의 성능을 비교해 보았다. 이번엔 특별히 sin함수도 activation function으로 사용하였다. 왜냐하면 데이터가 연속형 데이터인 만큼 주기성을 가질 수 있는데 sin함수의 특징이 주기성을 가지는것이므로 이가 도움이 될 수 있다고 생각하였다.



(elu그래프는 ReLU함수의 진화형) 우선 아무 activation function도 없는 y=x activation 형태가 가장 성능도 낮고 학습 그래프도 좋지 않게 나왔다. 그리고 image학습에서도 뛰어난 성능을 보인 ReLU함수가 여기서도 좋은 성능을 보였고 tanh함수가 그보다 조금 더 괜찮은 학습 결과를 보여주었다. 그리고 의외인점이 우리가 직접 만든 titledS 함수가 생각보다 성능이 잘 나왔다. 그리고 또한 연속형 데이터는 주기성을 가지므로 activation function도 주기를 가진다면 모델 성능이 향상될것이라 예측하였지만 sin_loss 그래프는 생각보다 성능이 크게 향상되지 않았음을 볼 수 있다. 이번 학습 결과를 통해서 activation function을 커스터마이징 하는 것이 생각보다 모델의 성능에 큰 영향을 끼칠 수 있고 직접 데이터에 맞는 activation function을 고안하고 만들어 내는 것이 모델 성능 향상의 하나의 방법이 될 수 있다는 점을 알게 되었다.

Ⅲ. 결론 및 소감

– 연구 결과:

activation function은 딥러닝 학습 과정에서 중요한 역할을 수행하고 있고 학습 성능에 큰 영향을 끼친다. 또한 해당 모델이 처리하는 데이터의 종류와 특징에 따라 해당 모델에 맞는 activation function이 각각 다를 수 있다. 또한 관습적으로 사용하는 sigmoid, softmax, relu, tanh 함수 말고도 자신이 직접 해당 데이터에 잘 맞는다고 생각하는 activation function을 사용한다면 모델 성능 향상에 큰 도움을 줄 수 있다.

[마치면서]

평소 딥러닝 모델을 구현할 때 아무생각없이 activation function을 사용한다거나 습관적으로 relu함수를 활용하였는데 이번 실험 결과를 통해 앞으론 아무생각없이 아무 함수나 사용하면 안되겠다는 생각이 들었고 과연 optimizer 또한 이런 성능 차이가 있을지 궁금해지게 되었다. 그리고 딥러닝에 대해서 더 공부하게 되면서 딥러닝이 매우 추상적인 학문이고 아직 연구되어야 할 분야가 많이 쌓여있다는 것을 느끼게 되었다. 그리고 데이터를 좀 더 복잡한 것을 가지고 조금 더 많은 연산량을 처리할 수 있는 환경이 갖추어져 있었으면 연구 결과가 조금 더 정밀하고 좋았을것이라는 아쉬움을 느끼게 되었다. 그리고 또한 드론을 직접 제작해보면서 소프트웨어는 이상적인 환경을 제공하므로 가끔은 오류가 발생하긴 해도 우리가 원하는 바를 만들고 실행시킬 수 있었지만 드론을 제작해보니 현실에선 생각보다 많은 문제점과 변수가 있다는 것을 깨닫게 되었다. 그리고 아무리 친한 친구들과 프로젝트를 진행한다고 하더라도 분업이 생각보다 힘들고 팀원들과 협업하는 것이 어렵다는 것을 알게되었다. 그렇지만 최종적으론 서로가 서로에게 많은 도움이 되었던 것 같고 정말 뜻깊은 시간이었던 것 같다. 나중엔 자율주행 드론에 카메라를 달아서 드론이 직접 길을 찾고 드론이 사람을 따라다니는 애완드론을 제작해보고 싶다.

학번 이름	활동 소감문(11pt, 진하게)
11pt	11pt로 작성