Project Presentation

Group06 김정연, 박서영, 서가은, 안민우

CONTENTS

1. 문제 정의

" 딥러닝 모델 task에 따른 최적의 활성화 함수 일반화 가능 여부 연구"

How? : 딥러닝 모델 학습시의 활성화 함수의 성능분석 및 시각화를 통해

Which Task? : Image and NLP (2 datasets for each task)

2. 팀원 구성

- 김정연 : APL unit 코드 작성 및 실험
- 박서영 : CNN 코드 작성 및 실험, Overleaf 작성
- 서가은 : BERT 코드 작성 및 실험, github 정리
- 안민우 : Resnet50 코드 작성 및 실험

3. 실험

task	**image classification**	**Text classification**
model	CNN, Resnet50 + ALU	BERT
data	MNIST, CIFAR-10	SST-2, CoLA

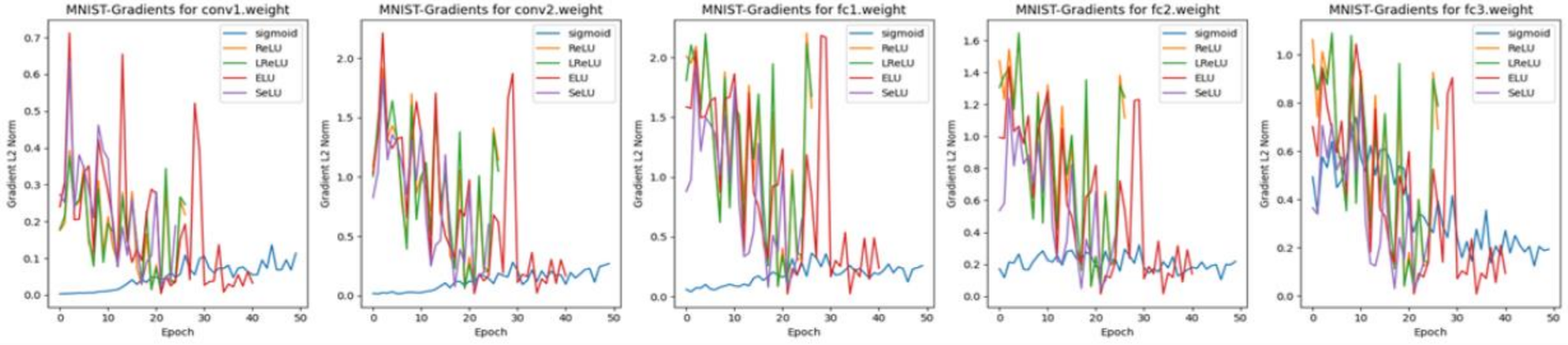
3.1. CNN 실험

- Model : Pytorch의 CIFAR-10 Tutorial 모델 사용
  - 입력 데이터의 채널 차원에 맞춰 수정
  - 기존에는 ReLU를 사용 -> 입력 activation function으로 바꿔줌
- Activation Functions : Sigmoid, ReLU, Leaky ReLU, ELU, SeLU
- 2가지 Optimizer에 대해 각각 실험
  - Adam, SGD
- Epoch 50 + Early Stopping (Validation Loss 기준, patience = 5)

```
def forward(self, x):
    x = self.pool(self.activation_function(self.conv1(x)))
    x = self.pool(self.activation_function(self.conv2(x)))
    x = torch.flatten(x, 1)
    x = self.activation_function(self.fc1(x))
    x = self.activation_function(self.fc2(x))
    x = self.fc3(x)
    return x
```

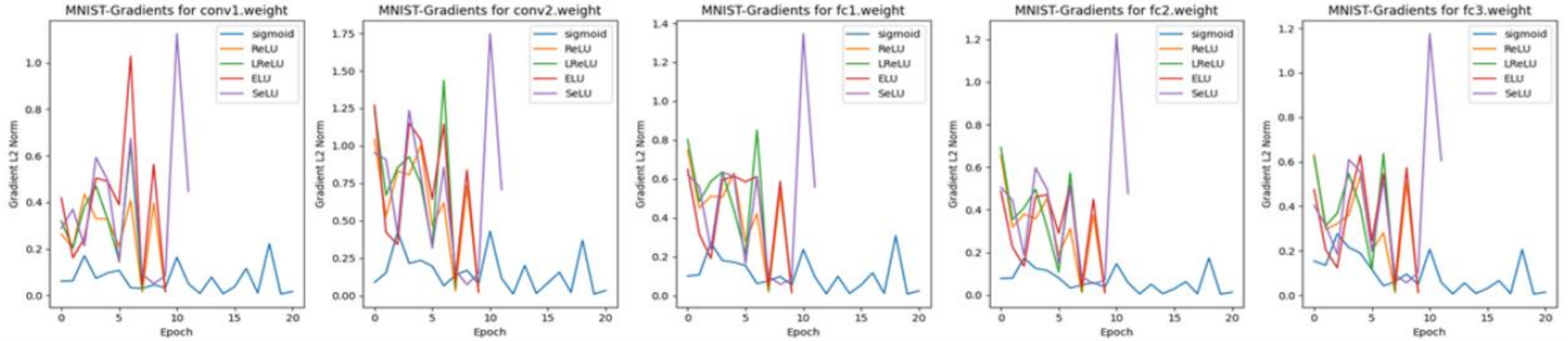
3.1. CNN 실험

- SGD



3.1. CNN 실험

- Adam



3.1. CNN 실험

- epoch
  - MNIST
    - Adam: LReLU(8) < ReLU(10) = ELU(10) < SeLU(12) < **Sigmoid**(21)
    - SGD: SeLU(25) < ReLU(27) = LReLU(27) < ELU(41) < **Sigmoid**(50)
  - CIFAR-10
    - Adam: LReLU(12) = SeLU(12) < ELU(13) < ReLU(14) < **Sigmoid**(40)
    - SGD: ELU(34) < SeLU(35) < LReLU(38) < ReLU(42) < **Sigmoid**(50)

3.1. CNN 실험

- Test Loss
  - Adam
    - MNIST: Sigmoid(0.038) < LReLU(0.051) < ReLU(0.052) < SeLU(0.055) < ELU(0.058)
    - CIFAR-10: Sigmoid(1.114) < LReLU(1.132) < SeLU(1.142) < ELU(1.148) < ReLU(1.193)
  - SGD
  -
    - MNIST: SeLU(0.038) < ELU(0.042) < ReLU(0.054) < LReLU(0.057) < Sigmoid(0.207)
    - CIFAR-10: ELU(1.023) < SeLU(1.083) < LReLU(1.114) < ReLU(1.137) < Sigmoid(1.945)

3.1. CNN 실험

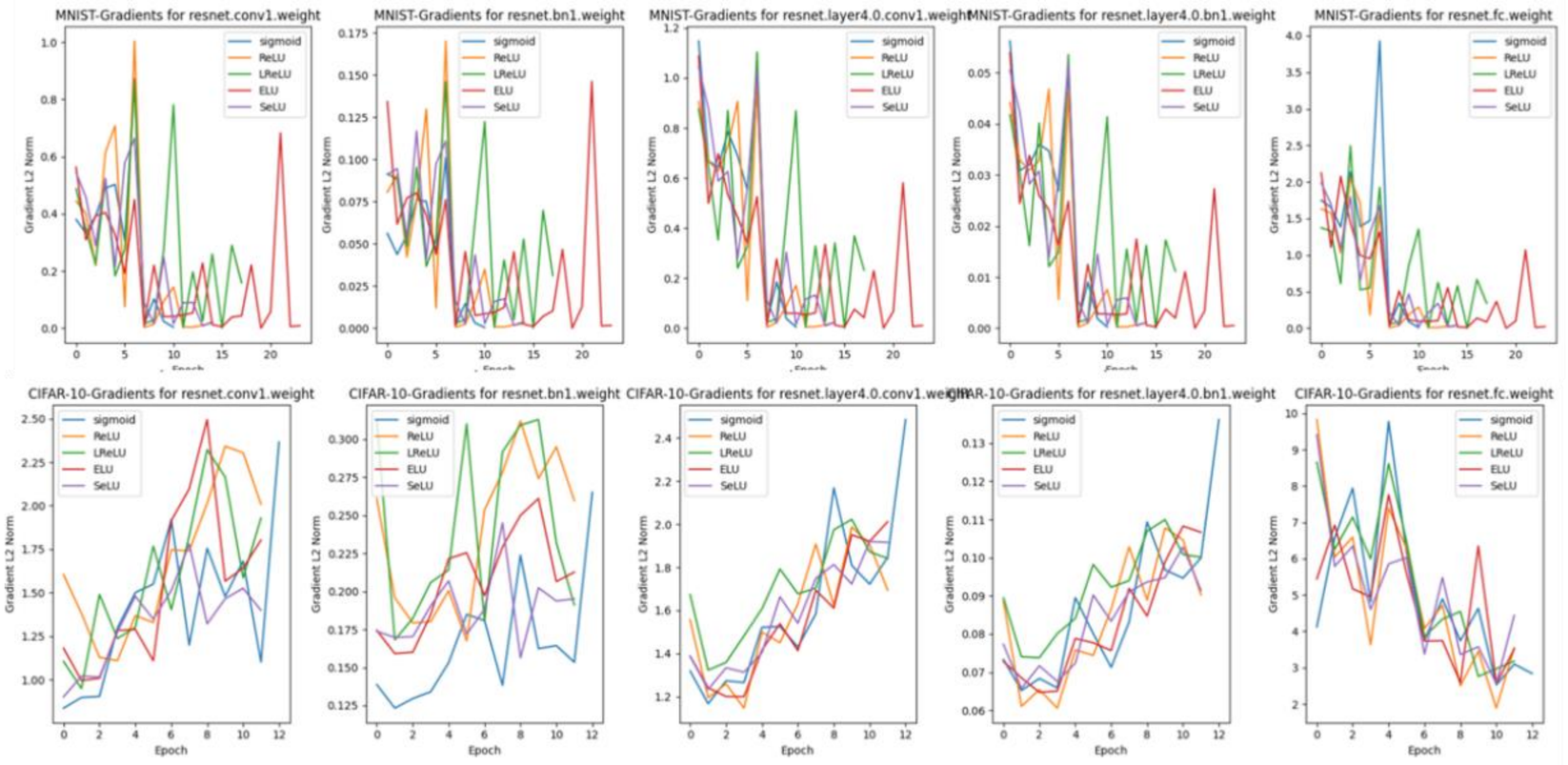
- Test Acc
  - Adam
    - MNIST: Sigmoid(98.84) > SeLU(98.74) > ELU(98.54) > LReLU(98.50) > ReLU(98.49)
    - CIFAR-10: **ELU**(63.70) > LReLU(62.88) > SeLU(61.83) > Sigmoid(61.57) > ReLU(61.51)
  - SGD
    - MNIST: SeLU(98.81) > ELU(98.78) > ReLU(98.35) > LReLU(98.26) > Sigmoid(94.22)
    - CIFAR-10: **ELU**(65.42) > SeLU(62.98) > ReLU(62.30) > LReLU(62.25) > Sigmoid(28.45)

3.2. Resnet50 실험

- Model: Pytorch의 기존 ResNet50 모델 사용
  - 입력 데이터의 채널 차원에 맞춰 수정
- Activation Functions: Sigmoid, ReLU, Leaky ReLU, ELU, SeLU
-
- 2가지 Optimizer에 대해 각각 실험
  - Adam, SGD
- Epoch 50 + Early Stopping (Validation Loss 기준, patience = 5)

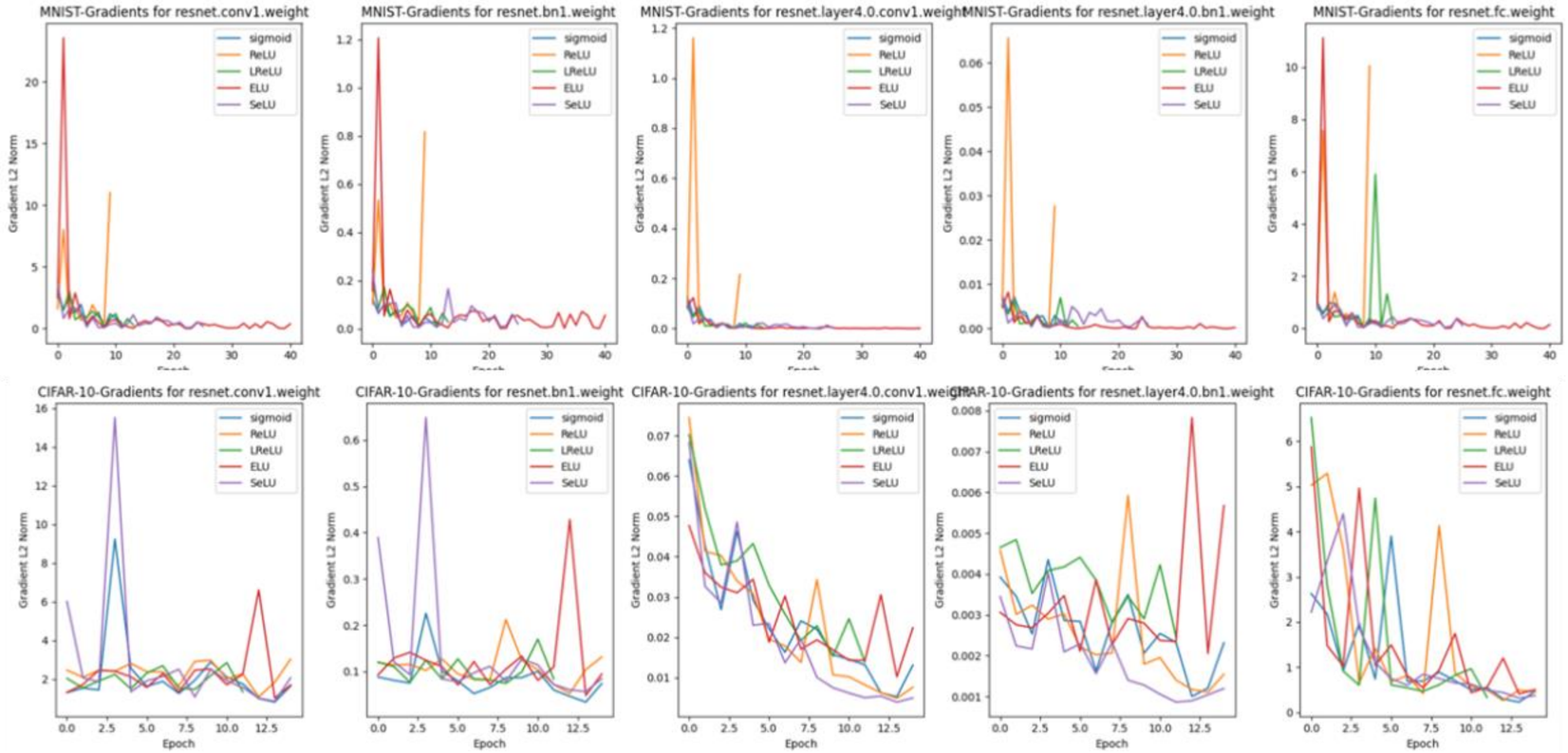
3.2. Resnet50 실험

- SGD



3.2. Resnet50 실험

- Adam



3.2. Resnet50 실험

- epoch
  - MNIST
    - Adam: ReLU(10) < Sigmoid(12) < LReLU(14) < SeLU(26) < ELU(41)
    - SGD: Sigmoid(11) < ReLU(15) = SeLU(15) < LReLU(18) < ELU(24)
  - CIFAR-10
    - Adam: LReLU(12) < Sigmoid(15) = ReLU(15) = ELU(15) = SeLU(15)
    - SGD: ReLU(12) = LReLU(12) = ELU(12) = SeLU(12) < Sigmoid(13)

3.2. Resnet50 실험

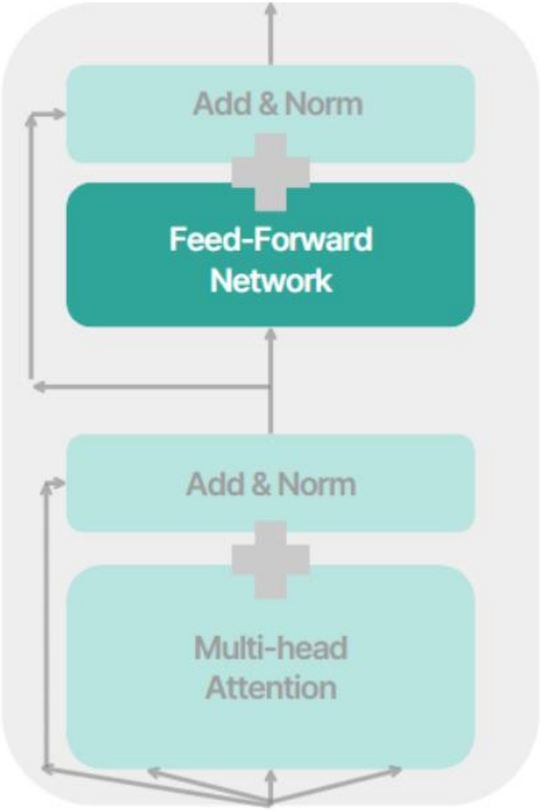
- Test Loss
  - Adam
    - MNIST: ELU(0.039) < SeLU(0.042) < LReLU(0.049) < Sigmoid(1.831) < ReLU(3.775)
    - CIFAR-10: LReLU(0.709) < Sigmoid(0.796) < ReLU(0.832) < SeLU(0.878) < ELU(0.971)
  - SGD
    - MNIST: LReLU(0.047) < ELU(0.054) < SeLU(0.054) < ReLU(0.051) < Sigmoid(0.062)
    - CIFAR-10: ELU(1.52) < ReLU(1.56) < SeLU(1.57) < Sigmoid(1.62) < LReLU(1.62)

3.2. Resnet50 실험

- Test Acc
  - Adam
    - MNIST: ReLU(10.60) < Sigmoid(50.73) < LReLU(98.45) < SeLU(98.68) < ELU(99.04)
    - CIFAR-10: ELU(76.20) < ReLU(78.07) < SeLU(78.68) < Sigmoid(78.75) < LReLU(79.00)
  - SGD
    - MNIST: Sigmoid(98.43) < ReLU(98.64) < SeLU(98.71) < ELU(98.75) < LReLU(98.77)
    - CIFAR-10: LReLU(58.02) < SeLU(58.51) < ELU(58.72) < Sigmoid(58.74) < ReLU(59.35)

3.3. BERT 실험

- 사용한 모델 : BERT
  - BERT의 feed forward network의 활성화 함수 변경
- 사용한 데이터
  - CoLA : 23개의 언어학 출판물로 이루어진 데이터셋
    - train : 7551
    - validation : 1043
    - test : 1000
  - SST2 : 영화 코멘트 데이터
    - train : 66349
    - validation : 872
    - test : 1000



3.3. BERT 실험

- **활성화 함수**
  - 기존 활성화 함수 : GeLU
  - 변경한 활성화 함수
    - relu, tanh, sigmoid, swish
    - elu, leaky relu, squared relu
    - glu, swiglu
- **실험 세팅**
  - CoLA : epoch3, AdamW, learning rate 5e-5
  - SST2 : epoch1, AdamW, learning rate 5e-5
  - Colab NVIDIA L4
  - pytorch, transformers, wandb

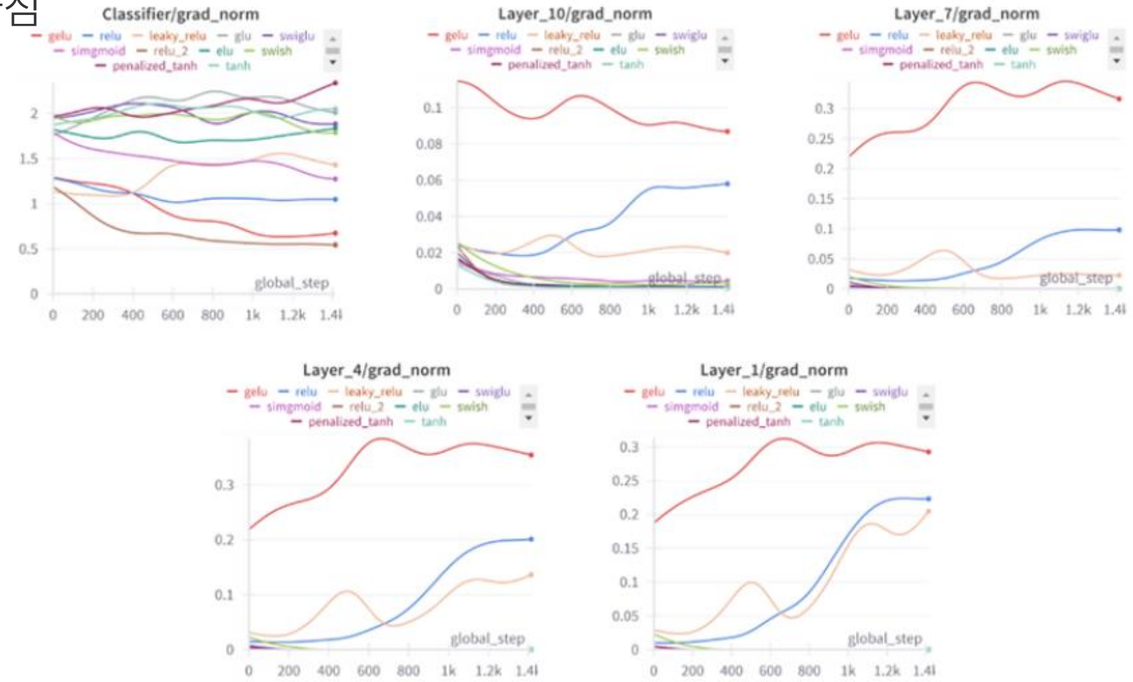
3.3. BERT 실험

- **실험 결과 : 정확도 & F1 score**
  - best score **:** gelu
  - relu, leaky relu, gelu를 제외하고는 모두 동일한 accuracy와 f1 score가 나옴
    - =학습이 되지 않음

data	acc/ f1score	relu	gelu	tanh	sigmoid	elu	leakyrelu	swish	relu_2	penalized tanh	glu	swiglu
cola	test accuracy	0.712	**0.806**	0.708	0.708	0.708	0.683	0.708	0.708	0.708	0.708	0.708
	test f1score	0.7192	**0.7938**	0.587	0.587	0.587	0.6345	0.587	0.587	0.587	0.587	0.587
sst2	test accuracy	0.938	**0.945**	0.558	0.558	0.558	0.931	0.558	0.558	0.558	0.558	0.558
	test f1score	0.9381	**0.9449**	0.3997	0.3997	0.3997	0.931	0.3997	0.3997	0.3997	0.3997	0.3997

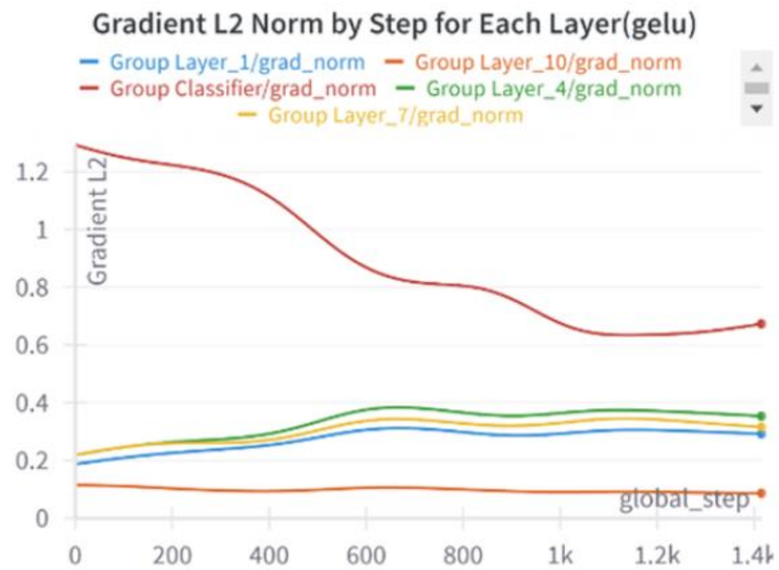
3.3. BERT 실험

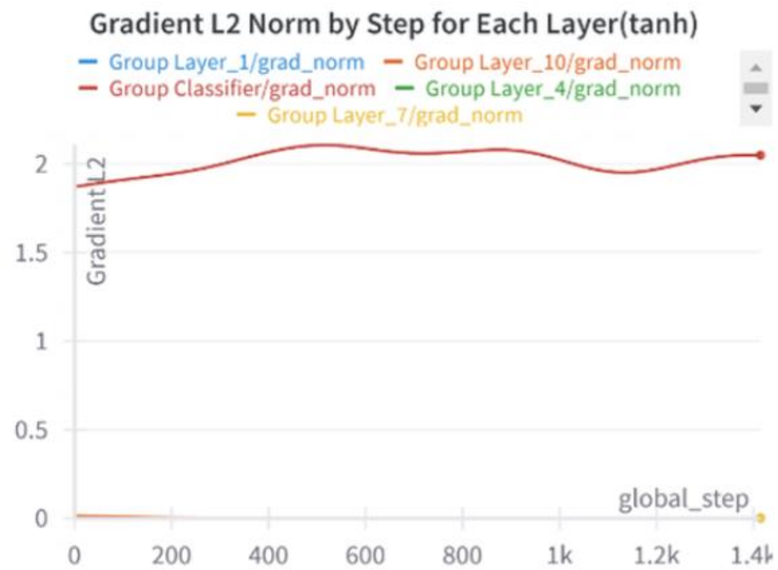
- **Gradient vanishing 문제**
  - 각 활성화 함수들의 layer별 gradient l2 norm 시각화
  - gelu, relu, leaky relu를 제외한 활성화 함수들의 gradient norm이 출력층에서 입력층으로 갈수록 극단적으로 작아짐



3.3. BERT 실험

- **Gradient vanishing 문제**

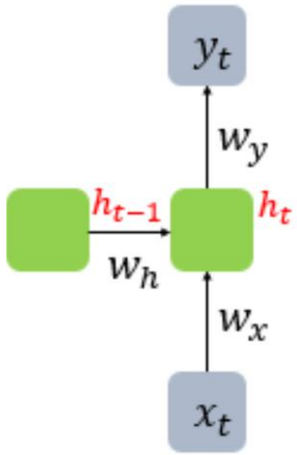




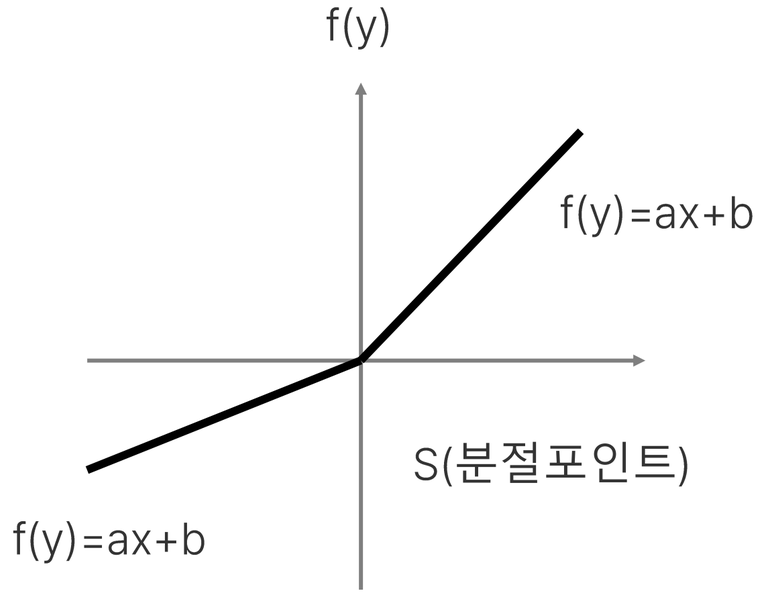
3.4. APL unit 실험

ADAPTIVE PIECEWISE LINEAR UNITS

- 실험 계기



RNN(Tanh)
$h_t = \mathrm{Tanh}(W_x x_t + W_h h_{t-1})$ $y_t = W_y h_t$



Nwankpa et al. (2018) arXiv:1811.03378v1
Activation Functions: Comparison of Trends in Practice and Research for Deep Learning

Eger et al. (2019) arXiv:1901.02671v1
Is it Time to Swish? Comparing Deep Learning Activation Functions Across NLP tasks

Szandała (2020) arXiv.2020.09458v1
Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks

3.4. APL unit 실험

ADAPTIVE PIECEWISE LINEAR UNITS

- 실험 계획1

S (힌지함수 개수)

$$h_i(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$$

2 Adaptive Piecewise Linear Units

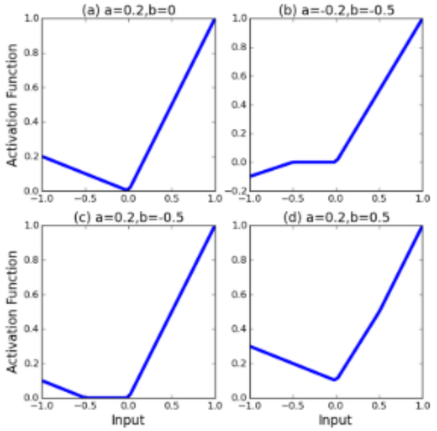
Here we define the adaptive piecewise linear (APL) activation unit. Our method formulates the activation function $h_i(x)$ of an APL unit i as a sum of hinge-shaped functions,

$$h_i(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s) \quad (1)$$

The result is a piecewise linear activation function. The number of hinges, S, is a hyperparameter set in advance, while the variables a_i^s, b_i^s for $i \in 1, ..., S$ are learned using standard gradient descent during training. The a_i^s variables control the slopes of the linear segments, while the b_i^s variables determine the locations of the hinges.

The number of additional parameters that must be learned when using these APL units is $2SM$, where M is the total number of hidden units in the network. This number is small compared to the total number of weights in typical networks.

Figure 1 shows example APL functions for $S = 1$. Note that unlike maxout, the class of functions that can be learned by a single unit includes non-convex functions. In fact, for large enough S, $h_i(x)$ can approximate arbitrarily complex continuous functions, subject to two conditions:



Theorem 1 *Any continuous piecewise-linear function $g(x)$ can be expressed by Equation 1 for some S, and a_i, b_i, $i \in 1, ..., S$, assuming that:*

1. *There is a scalar u such that $g(x) = x$ for all $x \geq u$.*
2. *There are two scalars v and α such that $\nabla_x g(x) = \alpha$ for all $x < v$.*

https://arxiv.org/abs/1412.6830
Agostinelli et al. (2015) arXiv:1412.6830v3
LEARNING ACTIVATION FUNCTIONS TO IMPROVE DEEP NEURAL NETWORKS

3.4. APL unit 실험

ADAPTIVE PIECEWISE LINEAR UNITS

- 실험 계획2

* 관찰하고자 한 부분

1) 3층 CNN 아키텍처, CIFAR-10에 대해 에포크별로 활성화함수 개형이 변하는 모습 시각화

2) 그렇게 적절하게 찾은 활성화함수 세트 vs 렐루 vs 리키렐루

3) 아키텍처 층을 더 많이 쌓았을 때 정확도의 격차가 더 커지는지 vs 작아지는지

* 실험 예측

1) 적절히 학습된 활성화 함수 개형은 렐루, 리키렐루 꼴과 비슷해질까?

2) APL unit이 accuracy가 더 높겠지?

3) 층마다 딱 맞는 활성화함수를 찾으니 격차가 더 커질까?

렐루와 리키렐루는 층이 깊어지면 표현력이 커지니 격차가 줄어들까?

https://arxiv.org/abs/1412.6830
Agostinelli et al. (2015) arXiv:1412.6830v3
LEARNING ACTIVATION FUNCTIONS TO IMPROVE DEEP NEURAL NETWORKS

3.4. APL unit 실험

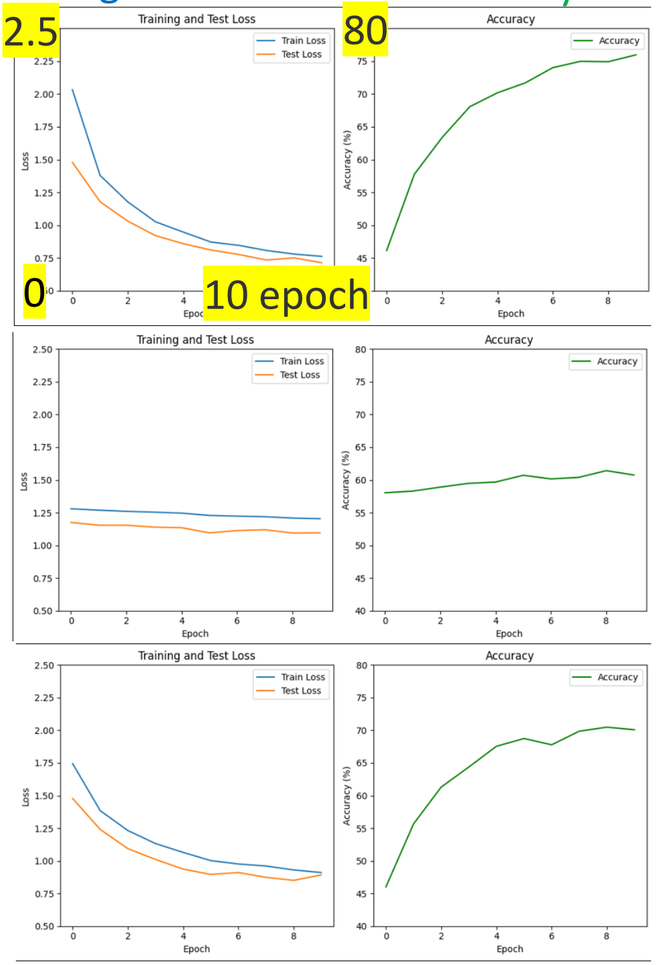
ADAPTIVE PIECEWISE LINEAR UNITS

- 실험 내용

APL unit

Relu

Leaky-Relu



Values of S	Accuracy
S=1	69.64
S=2	77.53
S=3	76.51
S=4	81.24
S=5	80.65
S=10	81.33

S의 값(활성화함수의 복잡성)이 accuracy와 선형적 관계가 없음을 확인

= 하이퍼파라미터 => 논문이 제안한 S=5 값으로 실험 진행

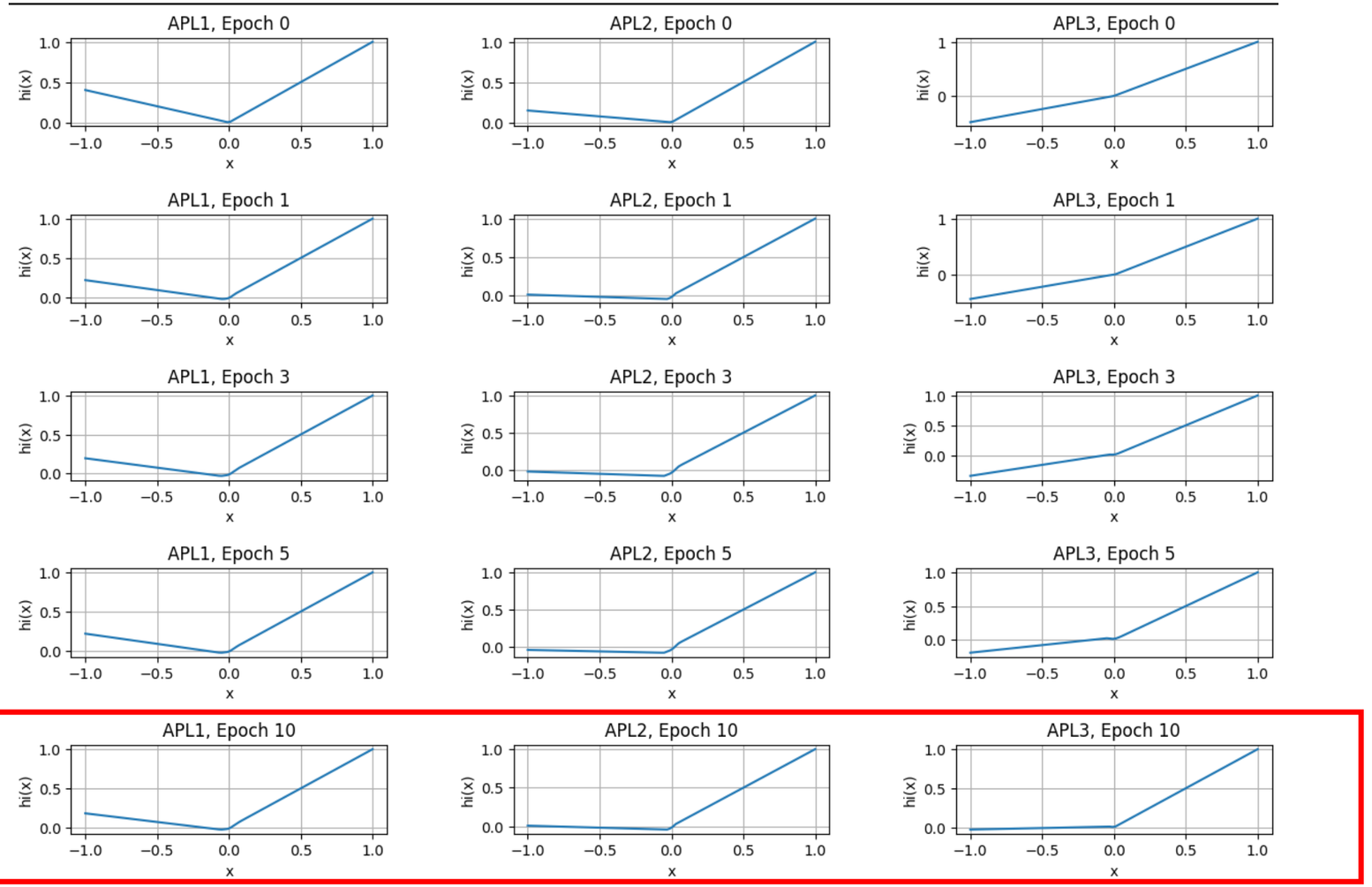
	First Epoch's Train Loss	Last Epoch's Test Loss	Final Accuracy
APL unit	2.0337	0.7134	75.98
ReLU	1.2797	1.0956	60.25
Leaky ReLU	1.7447	0.8923	70.08

Table 22. Overview of Results Obtained Through Code 1

3.4. APL unit 실험

```
self.conv1 = nn.Conv2d(3, 96, kernel_size=5, padding=2)
self.conv2 = nn.Conv2d(96, 128, kernel_size=5, padding=2)
self.conv3 = nn.Conv2d(128, 256, kernel_size=5, padding=2)
```

- 실험 내용

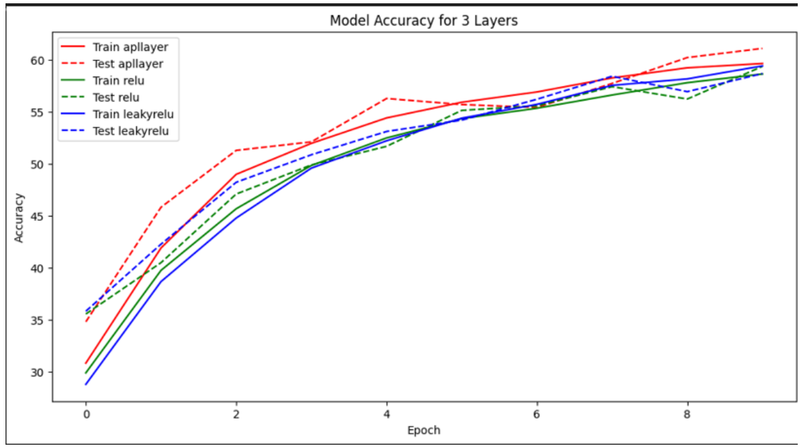


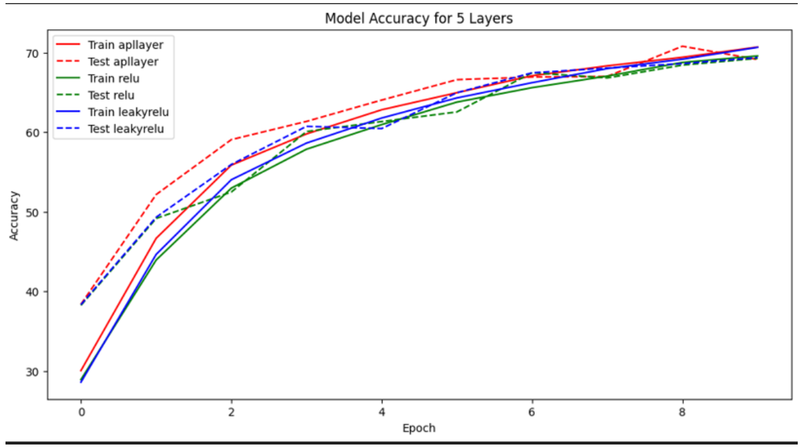
3.4. APL unit 실험

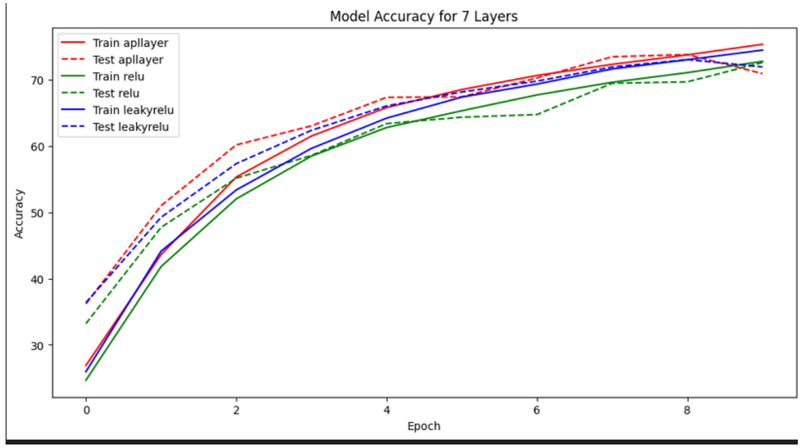
ADAPTIVE PIECEWISE LINEAR UNITS

- 실험 내용

3,5,7층의 train, test accuracy 시각화



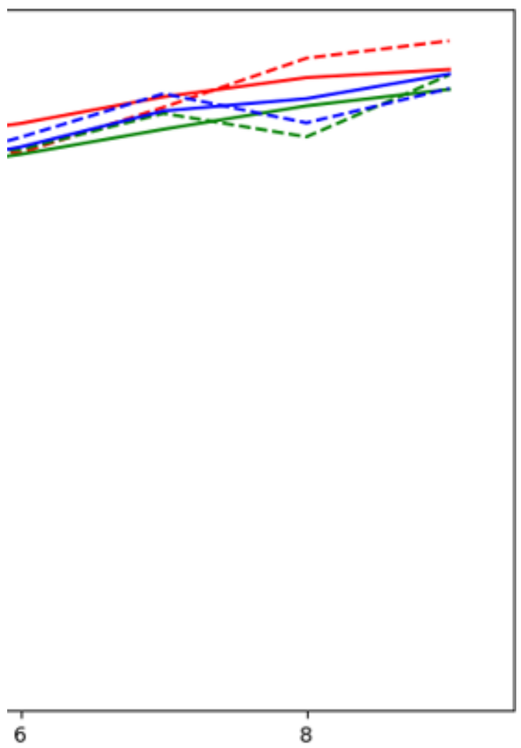




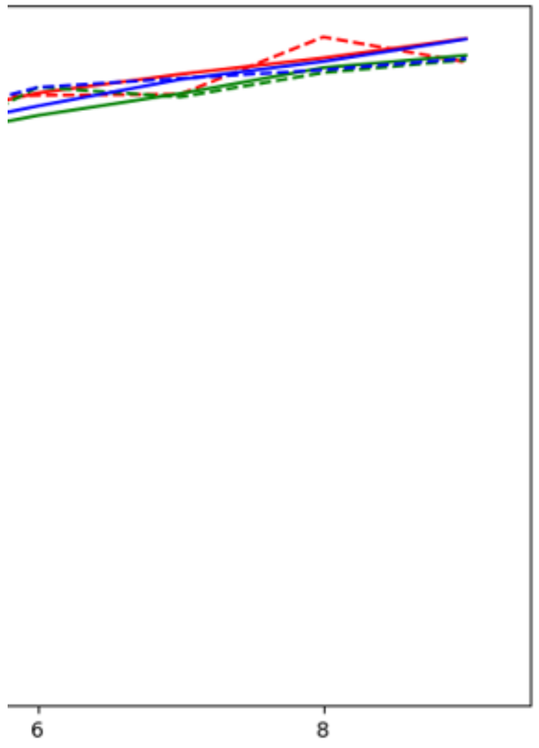
3.4. APL unit 실험

ADAPTIVE PIECEWISE LINEAR UNITS

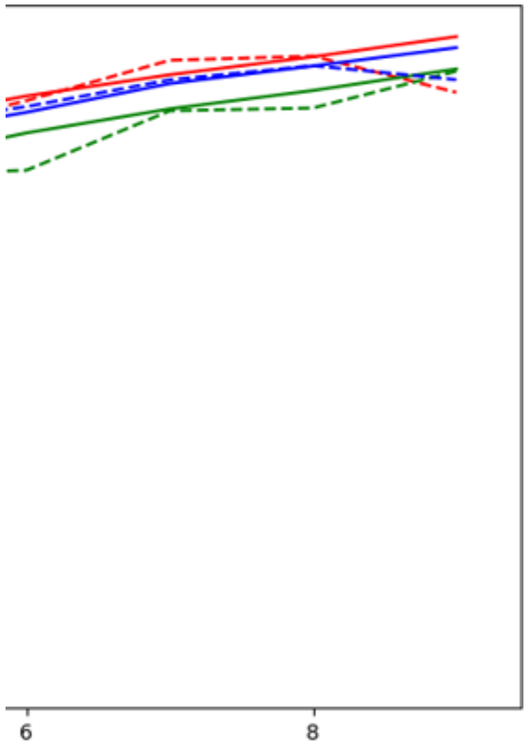
- 실험 내용 빨간점선 : APL unit의 test accuracy | 파란점선: 렐루 | 초록점선 | 리키렐루



3 layer



5 layer



7 layer

4. Result & Contribution

- **CNN&ResNet 실험**
  - Optimizer, dataset의 영향을 받지 않는 최적의 활성화함수를 찾을 수 없었음
  - CNN+CIFAR-10에서는 ELU의 Acc이 가장 높다 -> APL unit 실험과 일관된 결과
- **BERT 실험**
  - pretrain 과정에서 사용한 활성화 함수를 변경하면 성능이 떨어지는 경향이 있었음
  - relu 계열을 제외한 활성화 함수들은 심각한 gradient vanishing 문제가 발생함
- **ALU unit 실험**
  - ALU unit을 사용할 때 relu와 leaky relu를 층마다 고정해서 쓰는 것보다 나은 결과를 보임
  - 그러나, 전반적인 성능을 높이기 위해 보다 복잡한 아키텍처(깊은 층)을 사용할 때 성능 격차가 크지 않음
  - 따라서 적은 학습시간과 초기 에포크부터 안정적인 결과를 내는 렐루와 리키렐루의 사용이 왜 현재 인공지능업계에서 사용되는지 이해할 수 있는 결과를 보임
- **결론**
  - **절대적으로 성능이 좋은 활성화함수를 찾는 것보다 여러가지 조합을 시도해 보는 것이 바람직함**

감사합니다.

AI시스템group6