Training Deep Neural Nets [1] – Vanishing/Exploding Gradients Problems

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

국립한경대학교 HANKYONG NATIONAL UNIV.

복잡한 문제의 경우, 훨씬 더 깊은 DNN을 학습시켜야

⇒ 발생할 수 있는 문제점들:

- Vanishing/exploding gradients 문제 => 기울기 값이 점점 더 작아지거나 더 커지는 문제
- 충분한 training data를 확보하지 못한 경우 또는 라벨을 다는데 비용이 클 경우
- 학습이 극히 느려 질 수 있음
- 수백만 개의 파라메터를 가진 모델 => overfitting! (특히 training data가 충분하지 못할 경우)

Vanishing/Exploding Gradients Problems

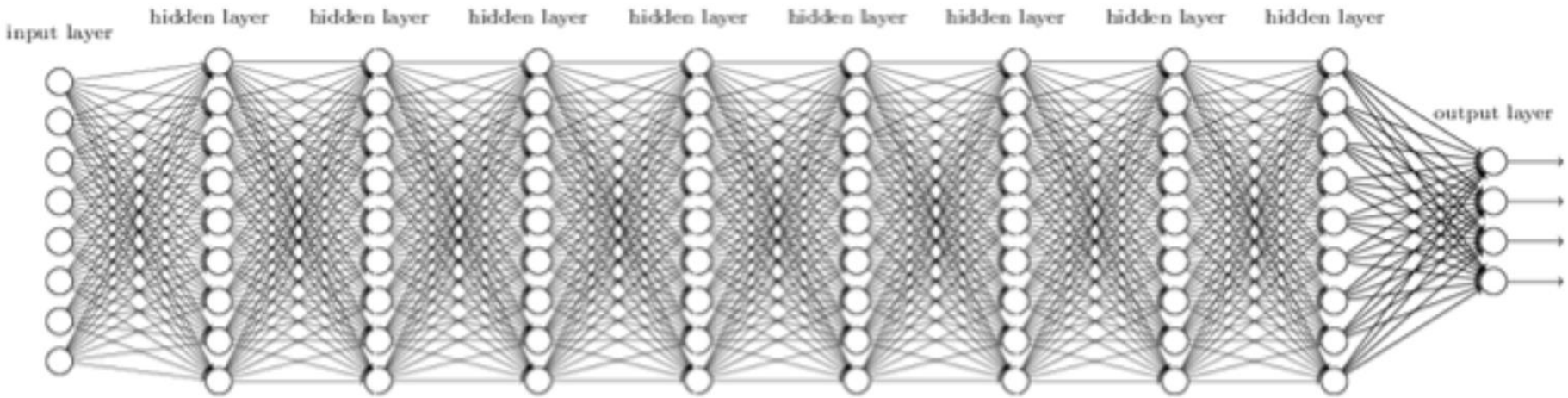
Backpropagation 알고리즘

- Output 레이어에서 Input 레이어 쪽으로 진행
- 각 파라메터에 대해 cost 함수의 기울기 계산
- 기울기를 이용해서 GD 스텝으로 각 파라메터 갱신

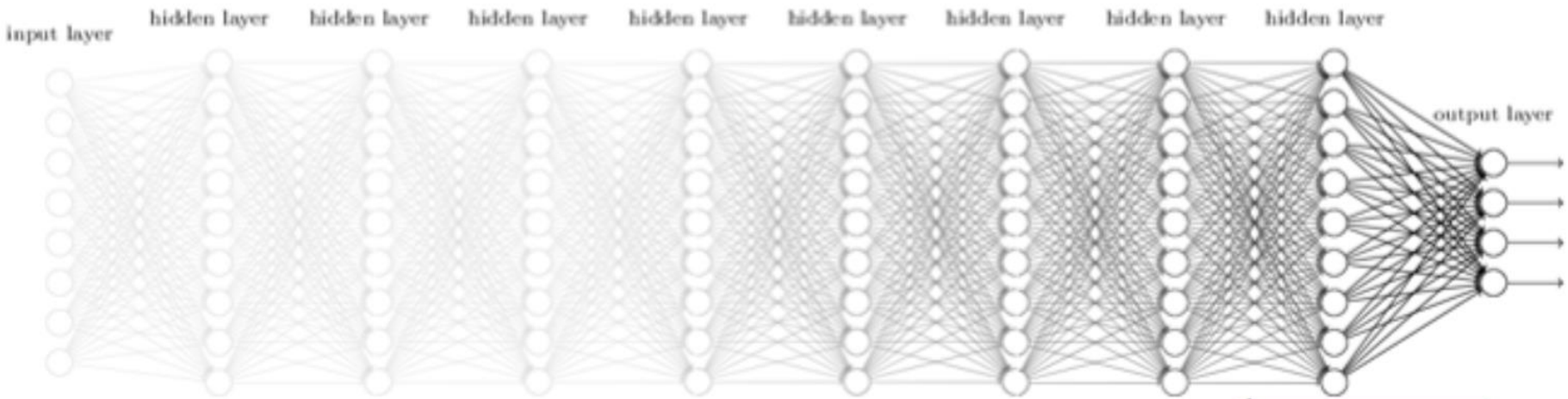
불행하게도, 낮은 레이어로 진행할수록 기울기 값이 점점 작아 짐

- 결국 낮은 레이어의 연결 가중치 값은 전혀 업데이트 되지 않음
- 학습 결과 => 좋은 솔루션에 수렴하지 못함
- "vanishing gradients 문제"라 함

Vanishing/Exploding Gradients Problems



Deep Neural Network



Vanishing Gradient

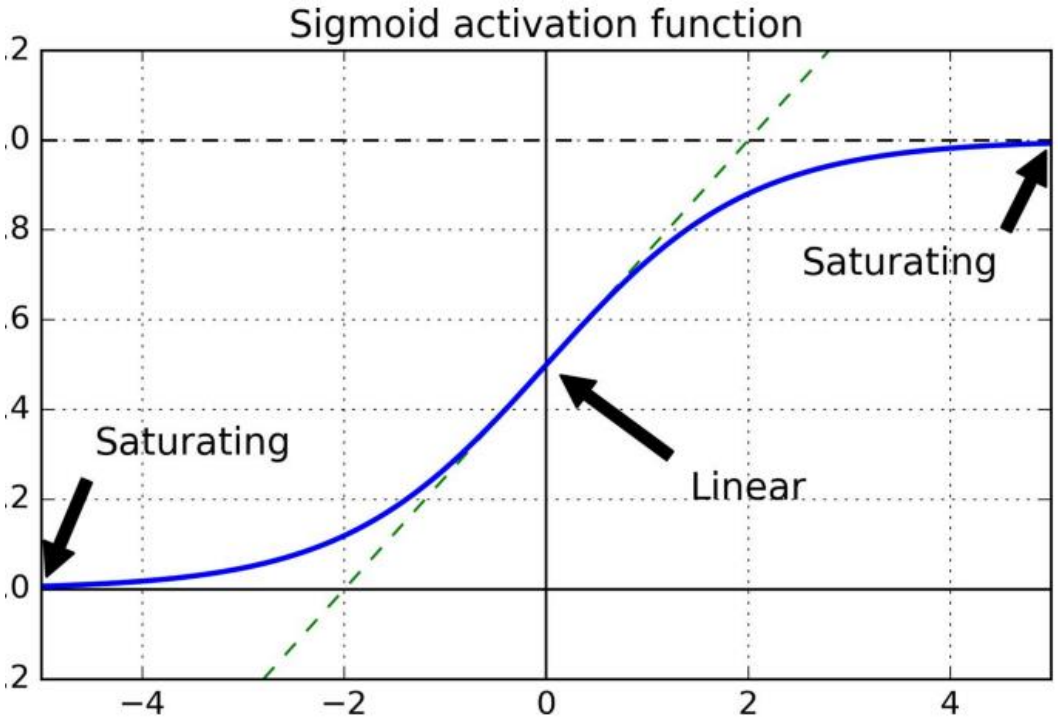
"vanishing gradients 문제" => 2000년대 초에 DNN에 대한 사람들의 관심을 잃게 한 이유 중의 하나

- 당시에는 기울기 값이 사라지는 이유를 밝히지 못함
- Xavier Glorot & Yoshua Bengio 논문 (2010년)

  ⇒ Sigmoid 활성 함수와 가중치 초기화 기법에 원인이 있다고 밝혀 냄

Sigmoid 활성 함수: 입력 값이 클 때 활성 함수는 0 또는 1에서 포화 상태가 됨

- 도함수 => 0의 값에 가까움
- 네트워크를 통해 뒤쪽(입력) 방향으로 전파시킬 기울기 값이 없음 (0에 가까움)



Glorot and He Initialization

Glorot and Bengio 방법 => 불안정한 기울기 문제를 상당히 완화시켜 주는 방법 제안

- 신호가 양방향으로 적절히 흘러가야 한다는 점을 발견:
  - ✓ 예측을 수행할 때의 순방향
  - ✓ 기울기를 역전파 시킬 때의 역방향
- 신호가 적절히 흐르게 하려면 레이어의 fan-in과 fan-out의 수가 동일해야 한다고 주장

  ⇒ 실용적인 방법 제안: 각 레이어의 연결 가중치를 아래 식처럼 랜덤하게 초기화

$$\text{Normal distribution with mean 0 and variance } \sigma^2 = \frac{1}{fan_{\text{avg}}}$$

$$\text{Or a uniform distribution between } -r \text{ and } +r, \text{ with } r = \sqrt{\frac{3}{fan_{\text{avg}}}}$$

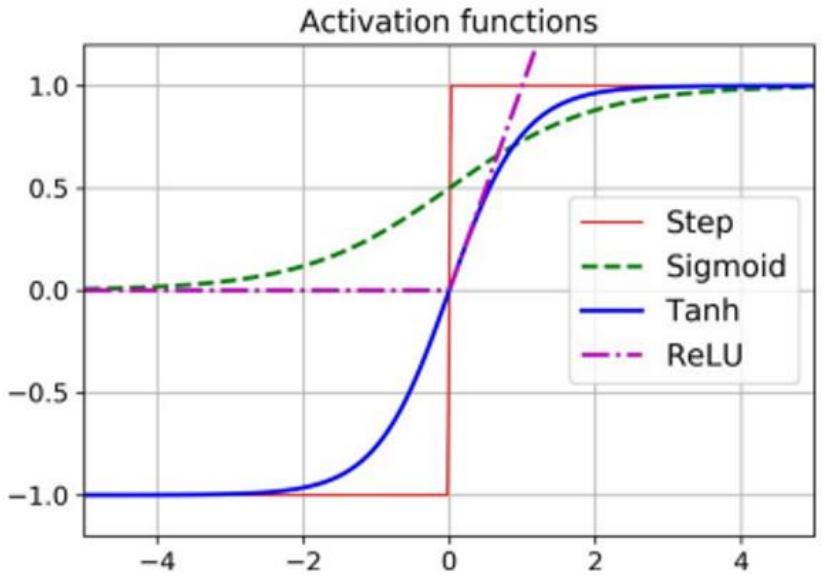
$$\text{where } fan_{\text{avg}} = (fan_{\text{in}} + fan_{\text{out}})/2.$$

Glorot and He Initialization

그 후 서로 다른 활성함수를 이용하는 유사한 방법들이 제안 됨

- Initialization parameters for each type of activation function

Initialization	Activation functions	σ^2 (Normal)
Glorot	None, tanh, logistic, softmax	$1 / fan_{avg}$
He	ReLU and variants	$2 / fan_{in}$
LeCun	SELU	$1 / fan_{in}$



- Keras => Glorot initialization 사용

  예: He initialization 방법으로 변경 가능 (by setting kernel_initializer="he_uniform" or kernel_initializer="he_normal" like this):

```
keras.layers.Dense(10, activation="relu", kernel_initializer="he_normal")
```

Nonsaturating Activation Functions

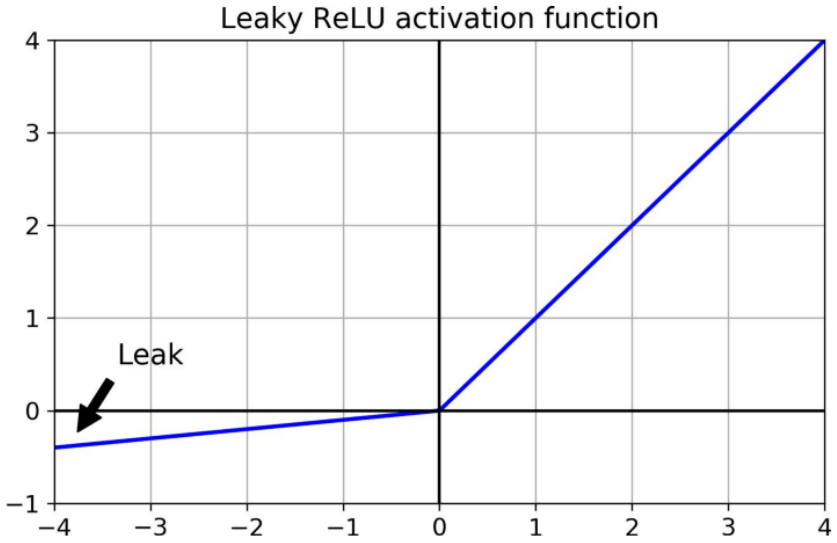
Glorot and Bengio 2010년 논문 : 불안정한 기울기 문제에 대한 통찰력

⇒ 활성 함수를 잘못 선택해도 부분적으로 기울기 문제에 영향을 미침

- ReLU 활성 함수 => 양의 값에 대해 포화 상태로 되지 않음 => 그러나 "dying ReLUs" 문제 발생

=> 이 문제를 해결하기 위해 다양한 변형된 방법들이 제안 됨

- ✓ Leaky ReLU : $\mathrm{LeakyReLU}_{\alpha}(z) = \max(\alpha z, z)$
- ✓ Randomized Leaky ReLU (RReLU)
- ✓ Parametric Leaky ReLU (PReLU)



Djork-Arné Clevert 제안 방법 (2015)

- proposed a new activation function called the exponential linear unit (ELU)
- 모든 ReLU 변형 방법들보다 성능이 좋음
- 학습시간이 줄어 들었고, 신경망의 경우 테스트 셋에 대해 성능이 더 좋음

$$\mathrm{ELU}_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

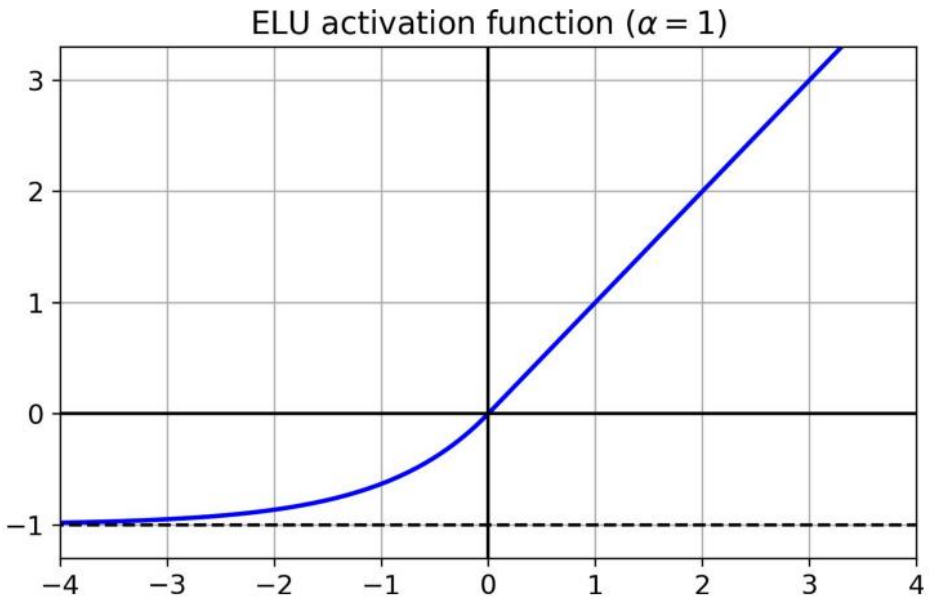
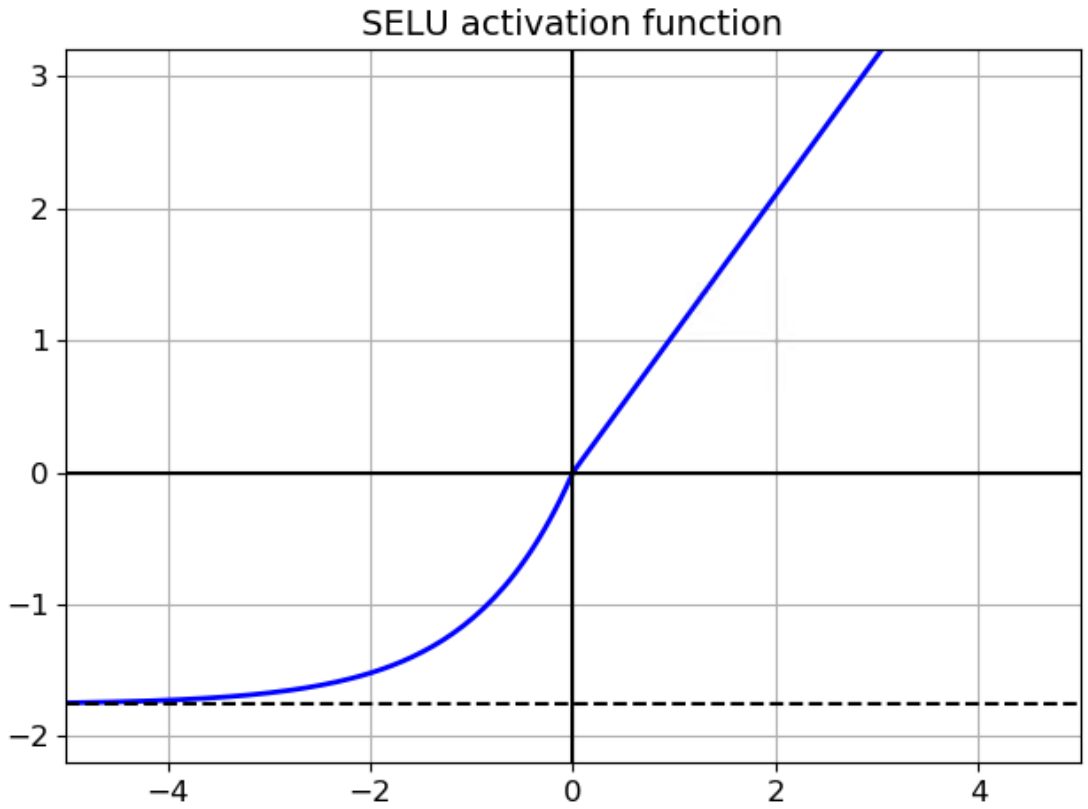


Figure 11-3. ELU activation function

Günter Klambauer 제안 방법 (2017)

- Introduced the **Scaled ELU** (SELU) activation function:



For SELU activation, set `activation="selu"` and `kernel_initializer="lecun_normal"` when creating a layer:

```
layer = keras.layers.Dense(10, activation="selu",
                           kernel_initializer="lecun_normal")
```

그렇다면 DNN의 히든 레이어에 어떤 활성함수를 사용하는 것이 좋을까?

⇒ 일반적으로 SELU > ELU > leaky ReLU (and its variants) > ReLU > tanh > logistic

Batch Normalization

He initialization/ELU(ReLU 변형 방법들) => 학습 초기에 "vanishing gradients 문제" 위험을 크게 감소시킴

- 그러나 학습이 진행되는 도중에 '기울기 사라짐' 문제가 다시 나타나지 않는다는 완전한 보증은 못함

Sergey Ioffe and Christian Szegedy 제안 방법 (2015)

- 이 문제를 해결하는 Batch Normalization (BN) 기법 제안
- BN 기법 => 각 히든 레이어의 활성 함수 바로 앞뒤에 **BN 오퍼레이션**을 추가
- BN 오퍼레이션 => 레이어의 각 입력에 대해 0(zero) 값을 중심으로 정규화를 수행

BN 레이어의 전체 오퍼레이션:

1. $\mu_B = \frac{1}{m_B}\sum_{i=1}^{m_B}\mathbf{x}^{(i)}$
2. $\sigma_B^{\,2} = \frac{1}{m_B}\sum_{i=1}^{m_B}(\mathbf{x}^{(i)} - \mu_B)^2$
3. $\hat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mu_B}{\sqrt{\sigma_B^{\,2} + \varepsilon}}$
4. $\mathbf{z}^{(i)} = \gamma \otimes \hat{\mathbf{x}}^{(i)} + \beta$

입력을 zero-center와 정규화를 하기 위해 각 입력의 평균과 표준편차를 추정

현재의 mini-batch 상에 평균과 표준편차 계산 => "Batch Normalization"이라 함

Implementing Batch Normalization with Keras

모델의 모든 히든 레이어 앞 또는 뒤에 BN 레이어 추가:

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(300, activation="elu", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(100, activation="elu", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(10, activation="softmax")
])
```

⇒ 많은 경우에 BN 레이어를 신경망의 첫 번째 레이어로 추가하면 StandardScaler를 사용하여 training set을 표준화 할 필요가 없다!

모델 Summary:

```
>>> model.summary()
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 784)               0
_________________________________________________________________
batch_normalization_v2 (Batc (None, 784)               3136
_________________________________________________________________
dense_50 (Dense)             (None, 300)               235500
_________________________________________________________________
batch_normalization_v2_1 (Ba (None, 300)               1200
_________________________________________________________________
dense_51 (Dense)             (None, 100)               30100
_________________________________________________________________
batch_normalization_v2_2 (Ba (None, 100)               400
_________________________________________________________________
dense_52 (Dense)             (None, 10)                1010
=================================================================
Total params: 271,346
Trainable params: 268,978
Non-trainable params: 2,368
```

각 BN 레이어 => 입력 당 4개의 파라메터 추가: $\boldsymbol{\gamma}, \boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\sigma}$

첫 번째 BN 레이어 : 3,136 개의 파라메터 추가 (4×784)

저자들의 주장 : BN 레이어를 활성함수 뒤 보다는 앞에 두는 것이 더 좋다!

- 활성함수 앞에 BN 레이어를 두기 위해 => 히든 레이어에서 활성함수를 분리하여 별도의 레이어로 추가
- Bias 항목 제거

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(300, kernel_initializer="he_normal", use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation("elu"),
    keras.layers.Dense(100, kernel_initializer="he_normal", use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation("elu"),
    keras.layers.Dense(10, activation="softmax")
])
```

실습과제 18-1

본문에 나오는 전체 내용 PyCharm에서 실행하기

★ 반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.

참고:

제 18강 실습과제 #18 Training Deep Neural Nets [1] - Vanishing&Exploding Gradients Problems.pdf