심층신경망 훈련

박 경 규

본 챕터 전체 내용의 출처는 도서 "핸즈온 머신러닝 2판(Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition)"입니다.

심층신경망 훈련 문제점

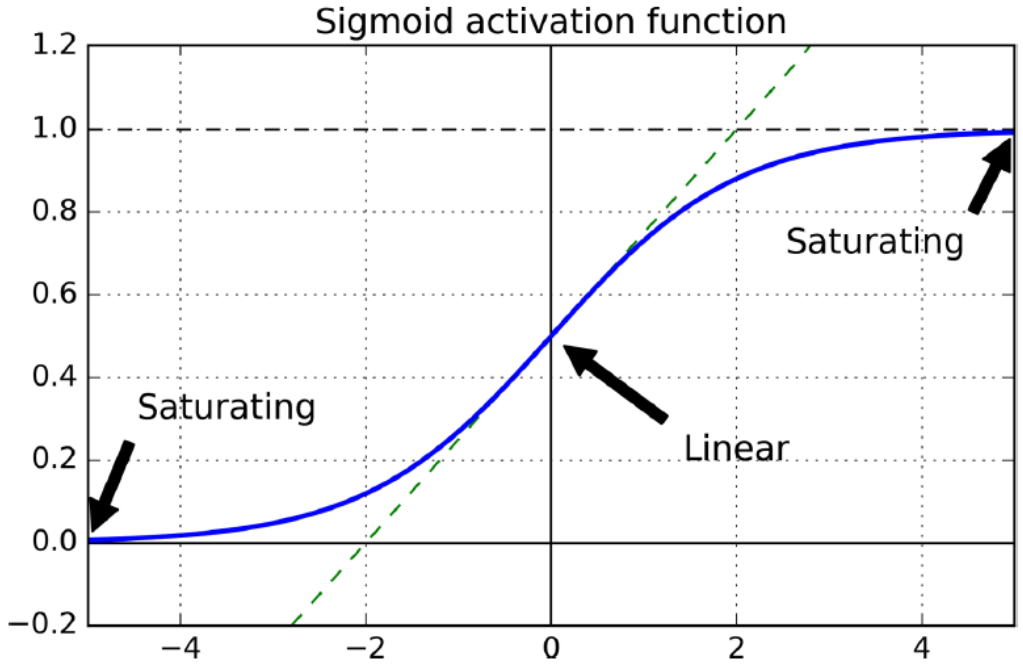
고해상도 이미지에서 수백 종류의 물체를 감지하는 것처럼 복잡한 문제에서는, 수백 개의 뉴런으로 구성된 10개 이상의 층을 수십만개의 가중치로 연결개 훨씬 더 깊은 심층신경망을 훈련해야 합니다.

■ 심층신경망 훈련 문제점

- 그레이디언트 소실(vanishing gradients)
- 그레이디언트 폭주(exploding gradients)
- 충분하지 않은 훈련데이터
- 레이블을 만드는 작업에 비용이 너무 많이 들 수 있습니다.
- 훈련이 극단적으로 느려질 수 있습니다.
- 수백만개의 파라미터를 가진 모델은 훈련 세트에 과대적합(overfitting) 될 위험이 매우 큽니다.

그레이디언트 소실과 폭주 문제

■ 로지스틱 활성화 함수의 수렴



■ Gradient Vanishing

- 심층신경망 학습(역전파) 과정에서 입력층으로 갈수록 기울기(Gradient)가 점차적으로 작아지는 현상이 발생할 수 있습니다.
- 입력층에 가까운 층들에서 가중치들이 업데이트가 제대로 되지 않으면 결국 최적의 모델을 찾을 수 없게 됩니다.

■ Gradient Exploding

- 기울기가 점차 커지더니 가중치들이 비정상적으로 큰 값이 되면서 결국 발산되기도 합니다.
- 이를 기울기 폭주(Gradient Exploding)이라고 하며, 순환신경망(Recurrent Neural Network, RNN)에서 발생할 수 있습니다.

글로럿과 He 초기화

Glorot 와 Bengio는 적절한 신호가 흐르기 위해서는 각 층의 출력에 대한 분산이 입력에 대한 분산과 같아야 한다고 주장

■ 세이비어 초기화 또는 글로럿 초기화(Xavier initialization or Glorot initialization)

- fan-in, fan-out : 층의 입력과 출력 연결 개수
- fan-avg = (fan-in + fan-out)/2
- 평균이 0이고 분산이 $\sigma^2 = \frac{1}{fan_{avg}}$ 인 정규분포
- 또는 $r = \sqrt{\frac{3}{fan_{avg}}}$ 일때 $-r$과 $+r$ 사이의 균등분포

■ 초기화 전략

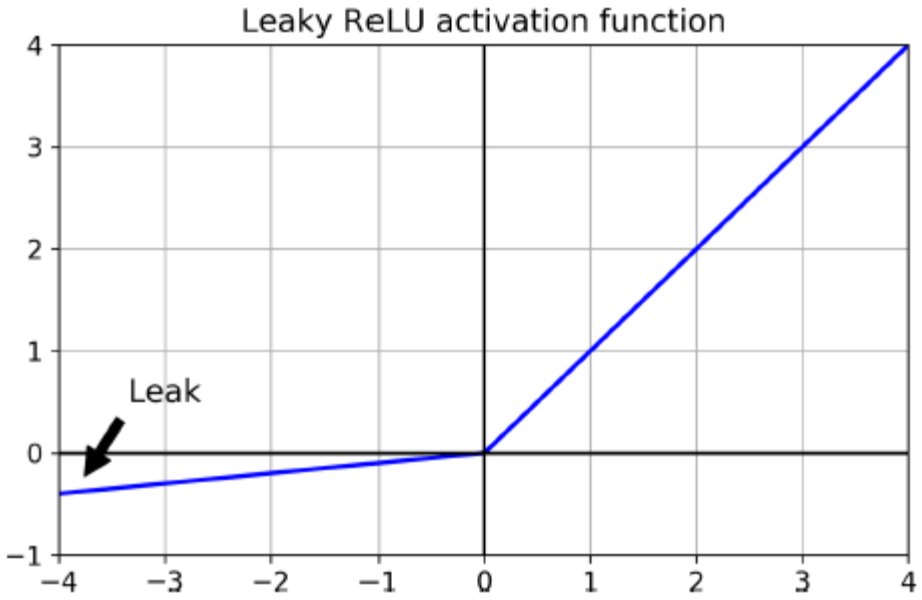
Initialization	Activation functions	σ^2 (Normal)
Glorot	None, tanh, logistic, softmax	$1 / fan_{avg}$
He	ReLU and variants	$2 / fan_{in}$
LeCun	SELU	$1 / fan_{in}$

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

수렴하지 않는 활성화 함수

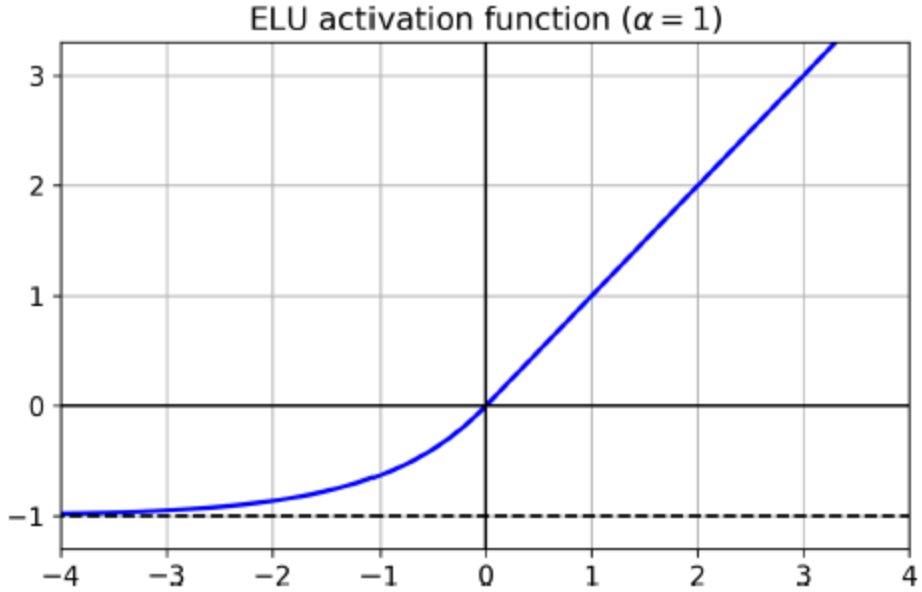
생물학적 뉴런의 방식과 비슷한 시그모이드 활성화 함수보다 다른 활성화함수가 심층신경망에서 훨씬 더 잘 작동

■ Leaky ReLU(Rectified Linear Unit)




```
model = keras.models.Sequential([
    [...]
    keras.layers.Dense(10, kernel_initializer="he_normal"),
    keras.layers.LeakyReLU(alpha=0.2),
    [...]
])
```

■ ELU(Exponential Linear Unit)




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

배치 정규화(Batch Normalization)

세리게이 이오페(Sergey Ioffe) 와 치리슈티언 세게지(Christian Szegedy)가 제안한 기법으로 각 층에서 활성화 함수를 통과하기 전이나 후에 모델에 연산을 하나 추가합니다.

■ 배치 정규화 알고리즘

1. $\mu_B = \frac{1}{m_B}\sum_{i=1}^{m_B} \mathbf{x}^{(i)}$

2. $\sigma_B^{\,2} = \frac{1}{m_B}\sum_{i=1}^{m_B} \left(\mathbf{x}^{(i)} - \mu_B\right)^2$

3. $\hat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mu_B}{\sqrt{\sigma_B^{\,2} + \varepsilon}}$

4. $\mathbf{z}^{(i)} = \gamma \otimes \hat{\mathbf{x}}^{(i)} + \beta$

■ 정규화 구현

```
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")
])
```

배치 정규화(Batch Normalization)

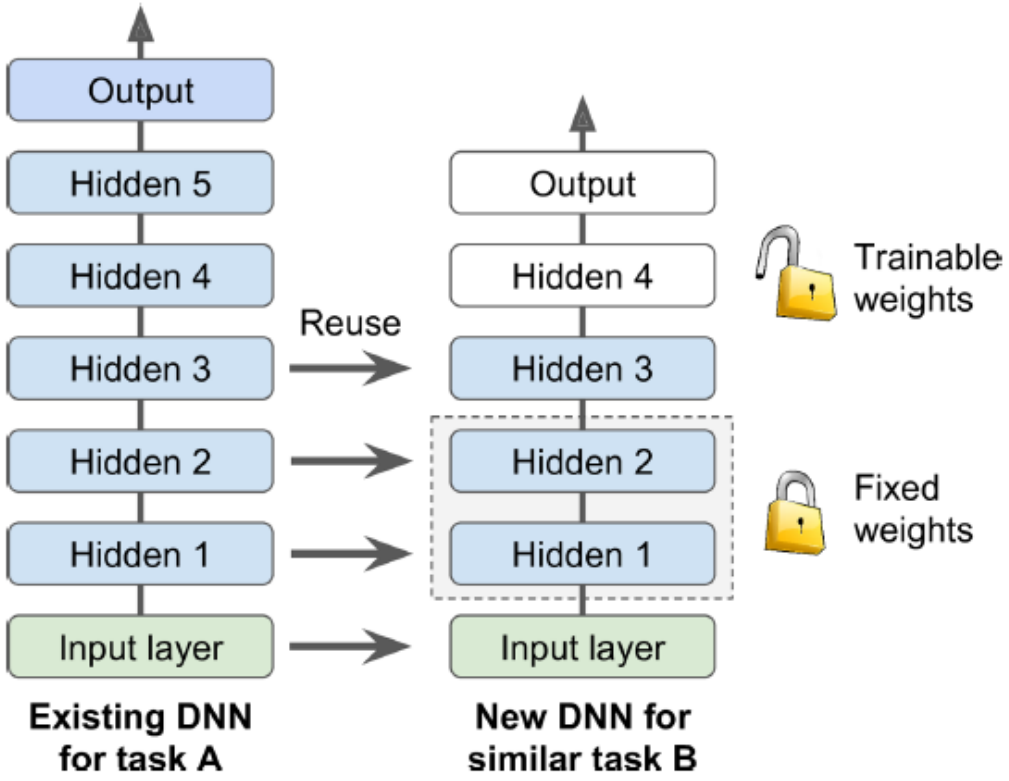
```
>>> model.summary()
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 784)               0
_________________________________________________________________
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
```

배치 정규화 층은 입력마다 4개의 파라미터 γ, β, μ, σ 를 추가합니다.

사전훈련된 층 재사용

큰 규모의 DNN 훈련시에는 비슷한 문제를 처리한 신경망을 찾아서 재사용하는 것이 좋으며 이를 전이학습이라고 합니다.

■ Pretrained Layer 재사용



■ 전이학습(Transfer Learning)

```
model_A = keras.models.load_model("my_model_A.h5")
model_B_on_A = keras.models.Sequential(model_A.layers[:-1])
model_B_on_A.add(keras.layers.Dense(1, activation="sigmoid"))
```

고속 옵티마이저

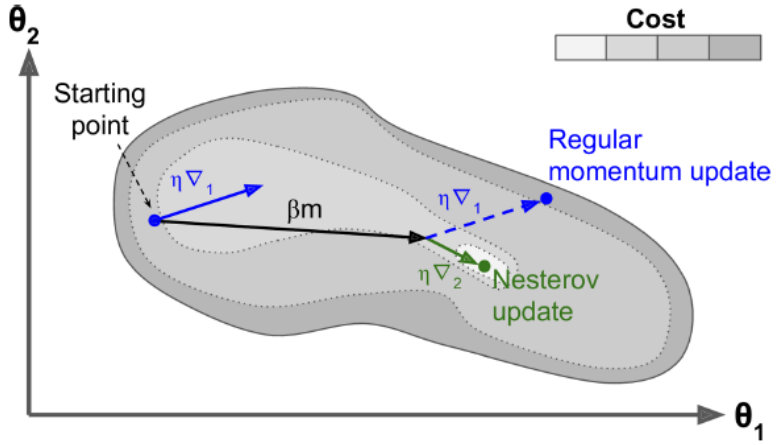
심층신경망 훈련속도를 크게 높일 수 있는 방법으로 보통의 경사하강법 대신 더 빠른 옵티마이저를 사용합니다.

Momentum Optimization

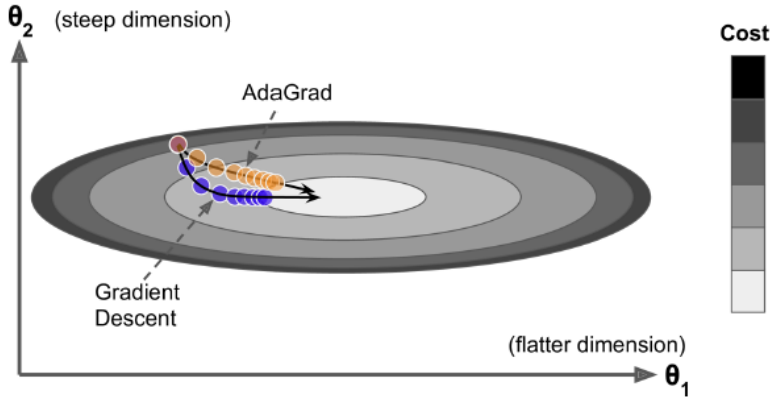
$$\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla_{\theta}J(\theta)$$

$$\theta \leftarrow \theta + \mathbf{m}$$

Nesterov Accelerated Gradient



AdaGrad



RMSProp

$$\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_{\theta}J(\theta) \otimes \nabla_{\theta}J(\theta)$$

$$\theta \leftarrow \theta - \eta\,\nabla_{\theta}J(\theta) \oslash \sqrt{\mathbf{s}+\varepsilon}$$

Adam and Nadam Optimization

1. $\mathbf{m} \leftarrow \beta_1\mathbf{m} - (1-\beta_1)\nabla_{\theta}J(\theta)$
2. $\mathbf{s} \leftarrow \beta_2\mathbf{s} + (1-\beta_2)\nabla_{\theta}J(\theta) \otimes \nabla_{\theta}J(\theta)$
3. $\widehat{\mathbf{m}} \leftarrow \dfrac{\mathbf{m}}{1-\beta_1^{\,t}}$
4. $\widehat{\mathbf{s}} \leftarrow \dfrac{\mathbf{s}}{1-\beta_2^{\,t}}$
5. $\theta \leftarrow \theta + \eta\,\widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}}+\varepsilon}$

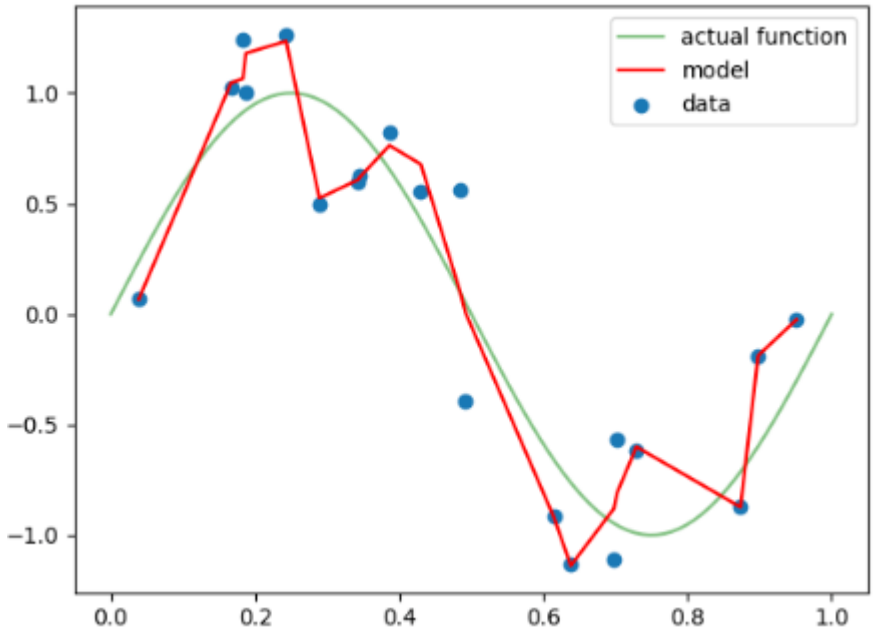
고속 옵티마이저 비교

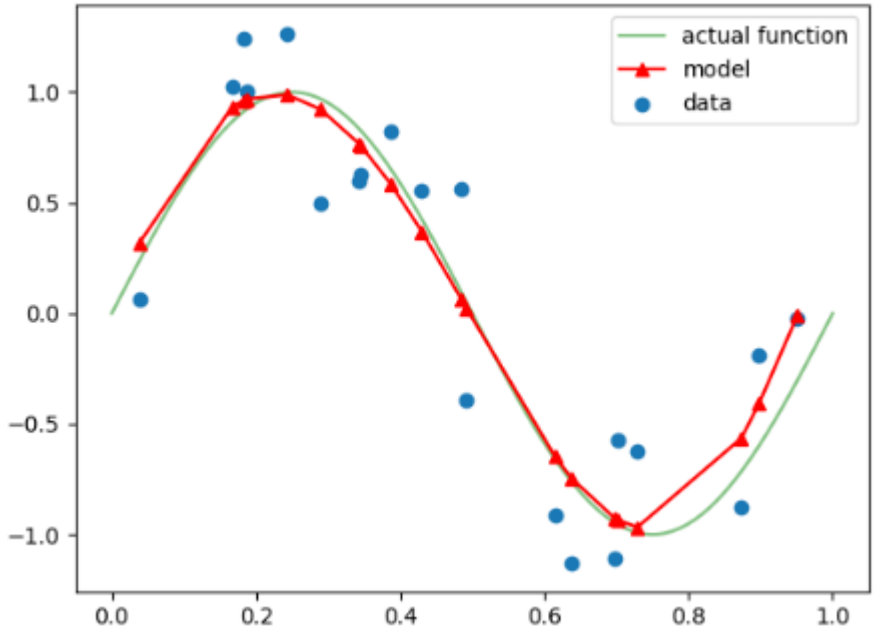
Class	Convergence speed	Convergence quality
`SGD`	*	***
`SGD(momentum=...)`	**	***
`SGD(momentum=..., nesterov=True)`	**	***
`Adagrad`	***	* (stops too early)
`RMSprop`	***	** or ***
`Adam`	***	** or ***
`Nadam`	***	** or ***
`AdaMax`	***	** or ***

L1 L2 규제

규제(Regularization)는 과적합을 예방하고 일반화(Generalization) 성능을 높이는데 도움을 주며, 신경망의 연결 가중치를 제한하기 위해 L2 규제를 사용하거나 많은 가중치가 0인 희소모델을 만들기 위해 L1규제를 사용할 수 있습니다.

■ Regularization



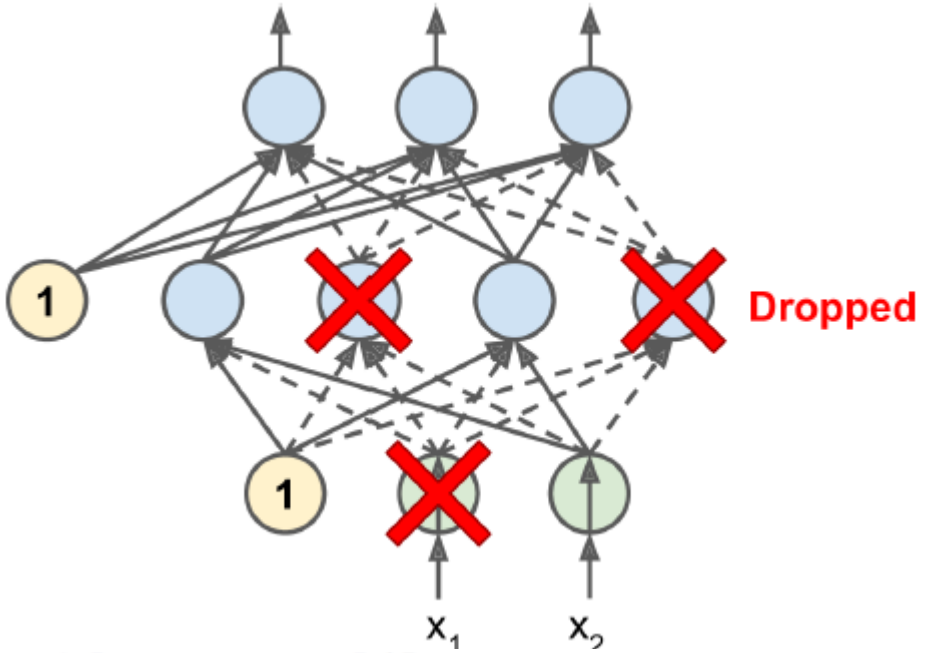


■ L2 규제 적용방법

```
layer = keras.layers.Dense(100, activation="elu",
                           kernel_initializer="he_normal",
                           kernel_regularizer=keras.regularizers.l2(0.01))
```

드롭아웃(Dropout)

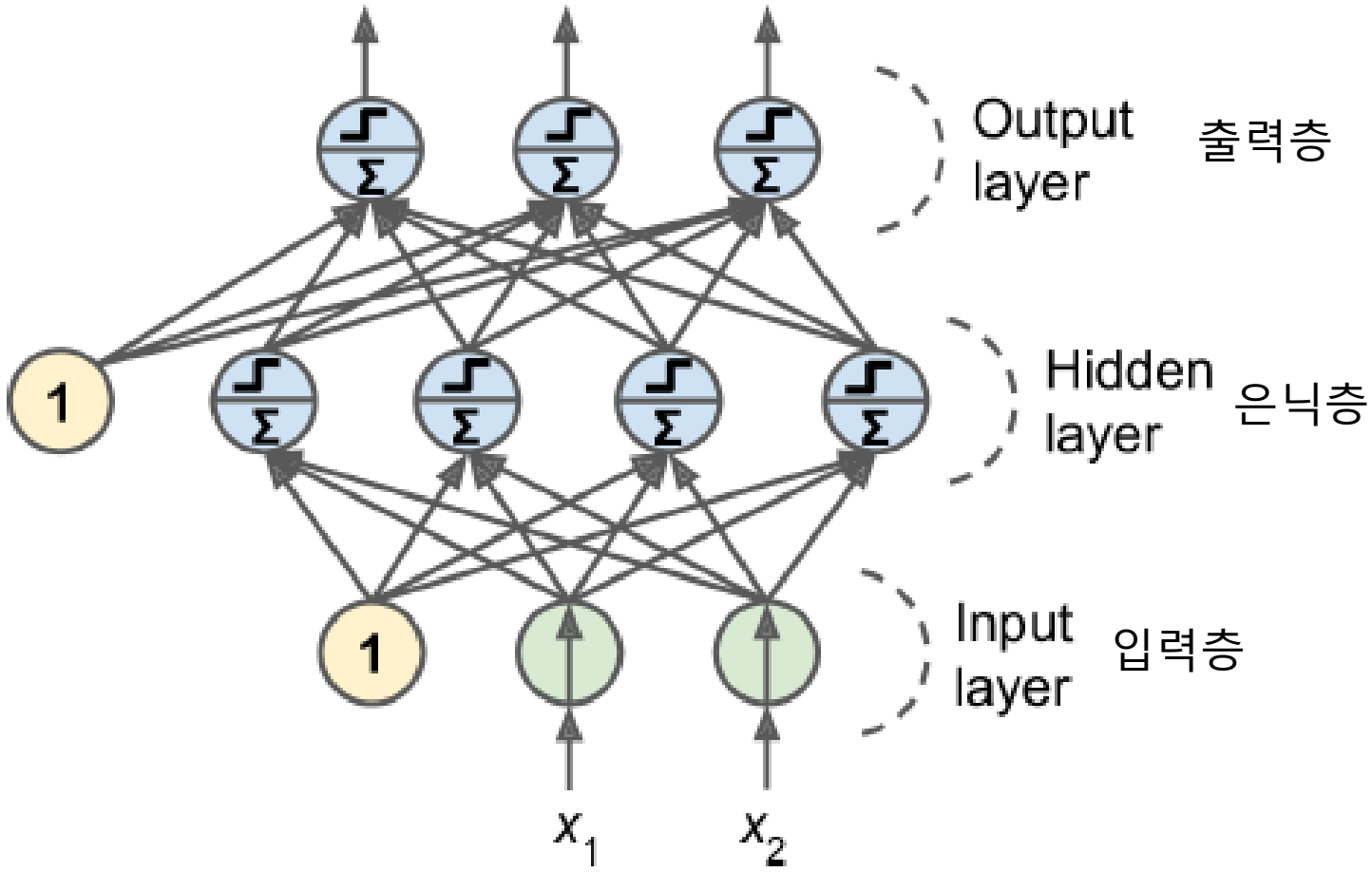
인기있는 규제기법으로 훈련할 때 임의의 뉴런을 삭제하여 신호를 전달하지 않게 하며, 테스트할 때는 모든 뉴런을 사용




```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(300, activation="elu", kernel_initializer="he_normal"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(100, activation="elu", kernel_initializer="he_normal"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(10, activation="softmax")
])
```

다층 퍼셉트론(MLP : Multilayer Perceptron)

입력층과 하나 이상의 은닉층과 출력층으로 구성되며, 은닉층을 여러 개 쌓아 올린 인공신경망을 심층신경망이라고 함



가이드라인

작업마다 좋은 기법은 다르며, 선택에 명확한 기준은 없음. 하이퍼파라미터 튜닝을 크게하지 않고 대부분의 경우에 맞는 설정

■ 기본 DNN 설정

하이퍼파라미터	일반적인 값
커널 초기화	He 초기화
활성화 함수	ELU
정규화	얕은 신경망일 경우 없음, 깊은 신경망이라면 배치 정규화
규제	조기종료 (필요하면 L2 규제 추가)
옵티마이저	모멘텀 최적화 (또는 RMSProp이나 Nadam)
학습률 사이클	1사이클

■ 자기 정규화를 위한 DNN 설정

하이퍼파라미터	일반적인 값
커널 초기화	르쿤 초기화
활성화 함수	SELU
정규화	없음(자기 정규화)
규제	필요하다면 알파 드롭아웃
옵티마이저	모멘텀 최적화 (또는 RMSProp이나 Nadam)
학습률 사이클	1사이클

Thank you

박경규, 010-7240-1128
kgpark88@gmail.com