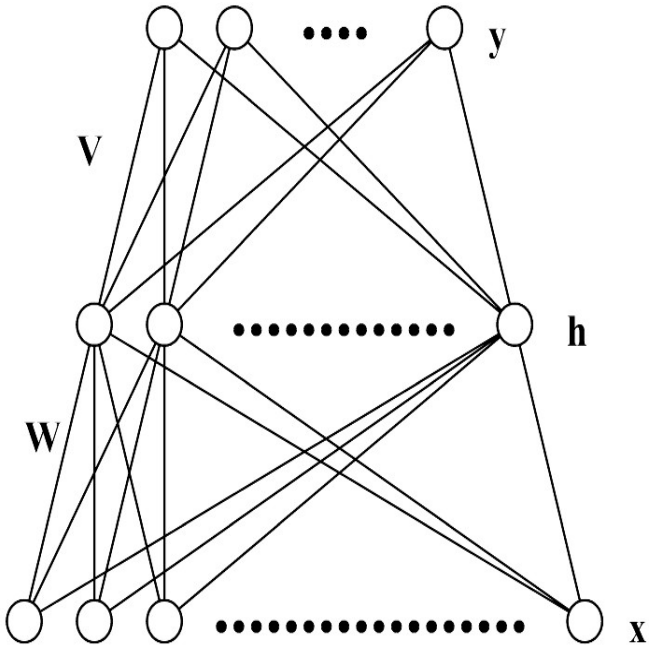
AI 실습

Contents

1. K-Nearest Neighbor Algorithm
2. Evalution of Classifiers(인식기의 성능 평가)
3. Perceptron
4. **Feed-Forward Neural Networks: Multi-Layer Perceptron**

5.1. Multi-Layer Perceptron (MLP)



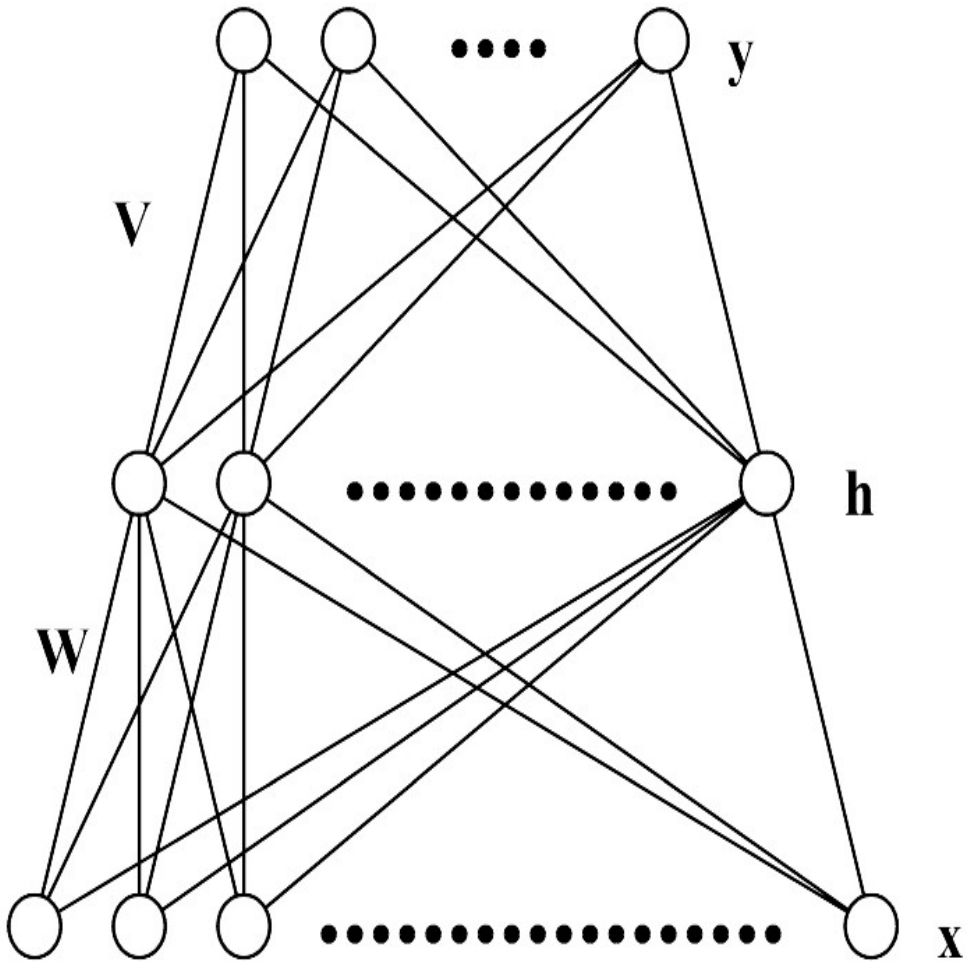
출력층 Output layer

은닉층 Hidden layer

입력층 Input layer

- Two-Layer Network (two-layer of weights)
- 전방향/순방향(Feed-forward) 구조: 전층에서 입력받아 다음 층으로 출력
- 출력층: 인식문제-시그모이드 활성화 함수, 회귀문제-선형 활성화 함수

MLP 구조 (*N-H-M*): 전방향 전파(Forward Propagation)



$$\text{Input Nodes}: \mathbf{x} = [x_1, x_2, \Lambda\ , x_N]^T$$

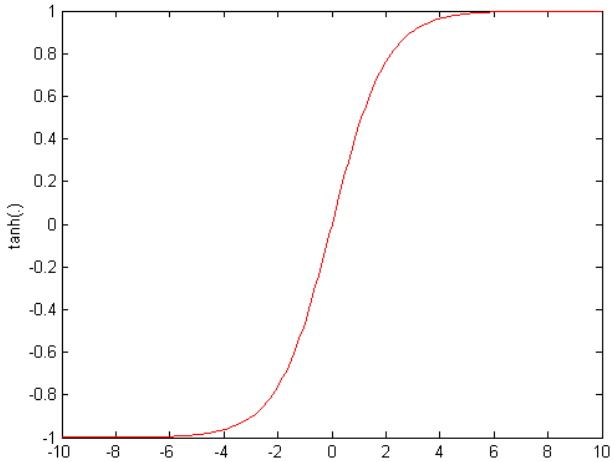
$$\text{Hidden Nodes}: \mathbf{h} = [h_1, h_2, \Lambda\ , h_H]^T$$

$$\text{Output Nodes}: \mathbf{y} = [y_1, y_2, \Lambda\ , y_M]^T$$

$$\text{Input} \longrightarrow \text{Output}(?)$$

$$h_j = f(\hat{h}_j) = \tanh(\hat{h}_j / 2)$$

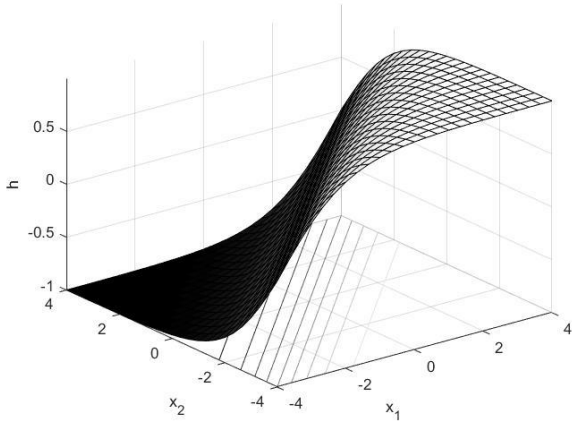
$$\text{where } \hat{h}_j = \sum_{i=1}^{N} w_{ji} x_i + w_{j0}$$

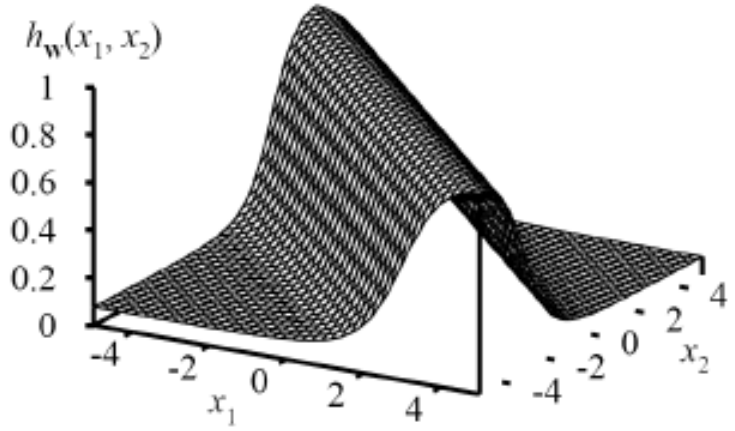


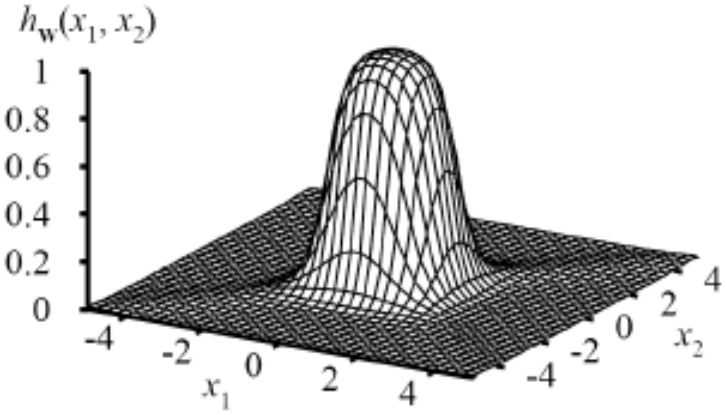
$$y_k = f(\hat{y}_k), \quad \text{where } \hat{y}_k = \sum_{j=1}^{H} v_{kj} h_j + v_{k0}$$

5.2. MLP의 표현능력

- MLP with two layers can represent arbitrary function
  - Each hidden unit represents a soft threshold function in the input space
  - Combine two opposite-facing threshold functions to make a ridge
  - Combine two perpendicular ridges to make a bump
  - Add bumps of various sizes and locations to fit any surface







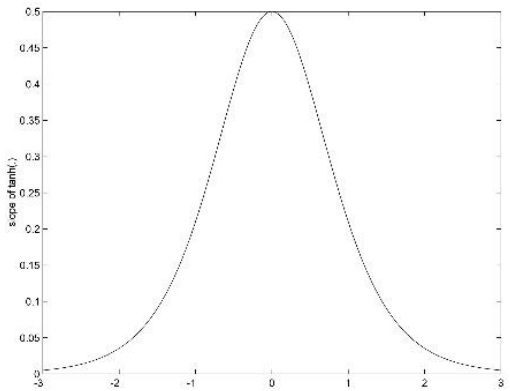
- 보편적 근사화 기계(Universal approximator)
  - Given a sufficiently large number of hidden units , a two layer (linear output) network can approximate any continuous function on a compact input domain to arbitrary accuracy.

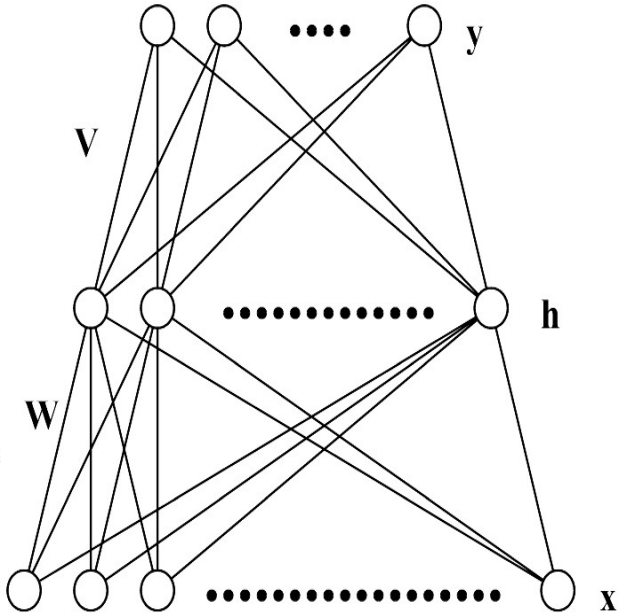
5.3. 오류 역전파 학습(Error Back-Propagation Algorithm)

Input Pattern: $\mathbf{x} = [x_1, x_2, \mathrm{L}\ , x_N]^T$

Output Vector: $\mathbf{y} = [y_1, y_2, \mathrm{L}\ , y_M]^T$

Desired Output Vector: $\mathbf{t} = [t_1, t_2, \mathrm{L}\ , t_M]^T$



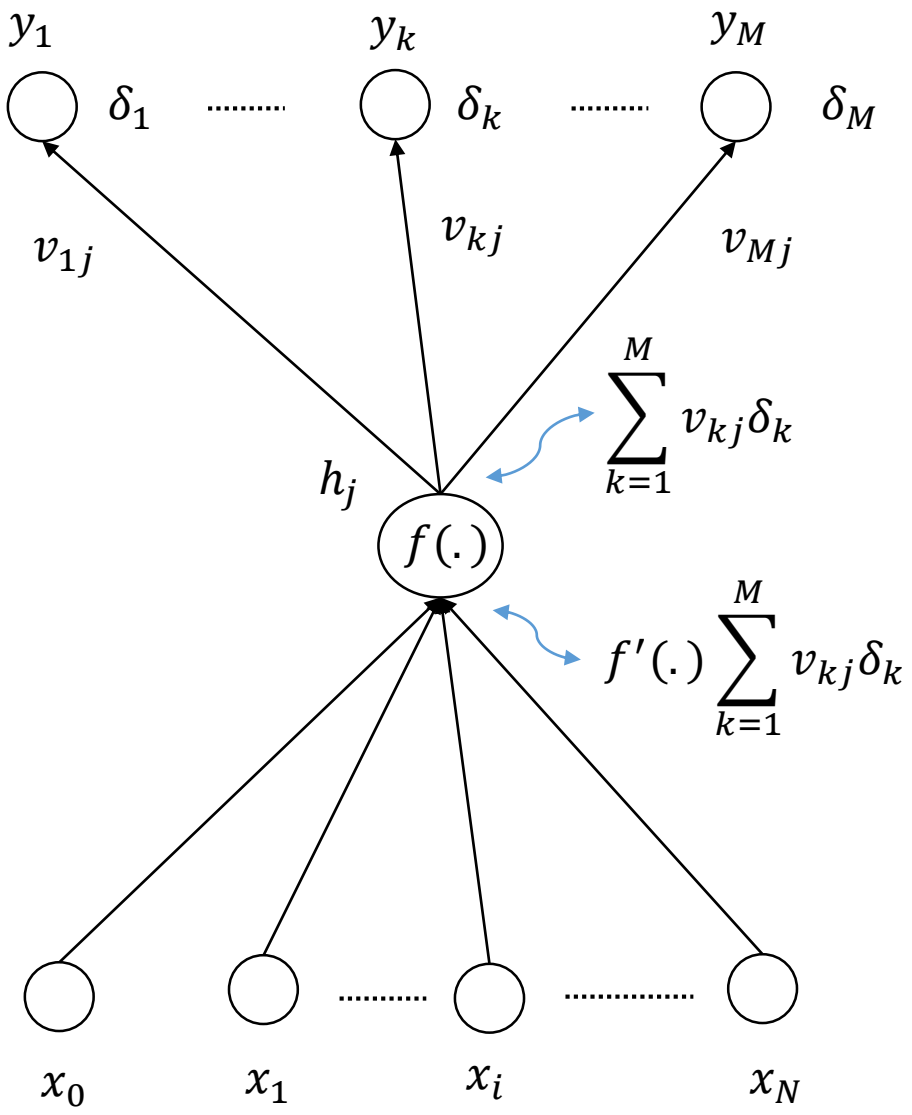


Mean-Squared Error Function:

$$E_m(\mathbf{x}) = \frac{1}{2}\sum_{k=1}^{M}(t_k - y_k)^2$$

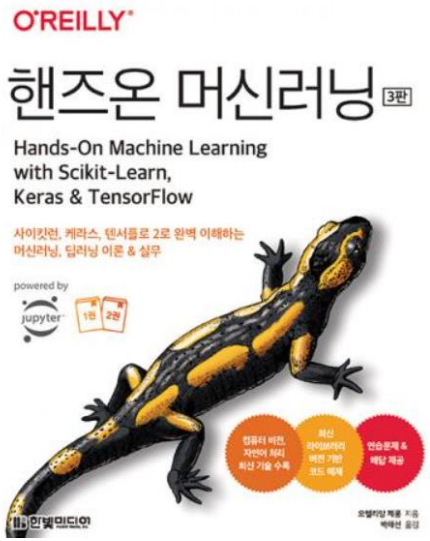
$$\Delta v_{kj} = -\eta \frac{\partial E_m(\mathbf{x})}{\partial v_{kj}} = \eta \delta_k^{(out)} h_j \quad \text{where } \delta_k^{(out)} = -\frac{\partial E_m(\mathbf{x})}{\partial \hat{y}_k} = (t_k - y_k) f'(\hat{y}_k)$$

$$\Delta w_{ji} = -\eta \frac{\partial E_m(\mathbf{x})}{\partial w_{ji}} = \eta \delta_j^{(hid)} x_i \quad \text{where } \delta_j^{(hid)} = -\frac{\partial E_m(\mathbf{x})}{\partial \hat{h}_j} = f'(\hat{h}_j)\sum_{k=1}^{M} v_{kj}\delta_k^{(out)}$$


y_1
y_k
y_M
δ_1
δ_k
δ_M
v_{1j}
v_{kj}
v_{Mj}
$\sum_{k=1}^{M} v_{kj}\delta_k$
h_j
$f(.)$
$f'(.)\sum_{k=1}^{M} v_{kj}\delta_k$
x_0
x_1
x_i
x_N

실습(회귀)

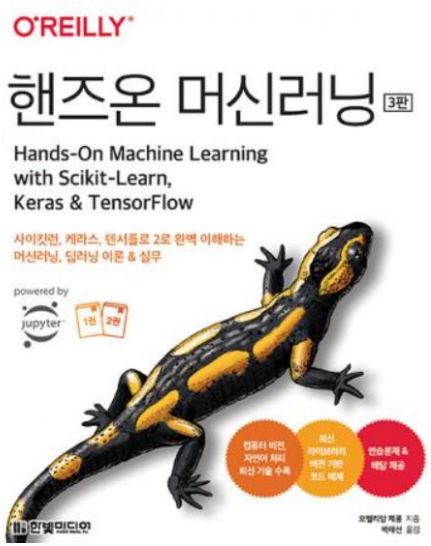
```
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

housing=fetch_california_housing()
print(housing.DESCR)
```



MLPRegressor-10장-p385-3rdEd.py

```
California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/

The target variable is the median house value for California districts.

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

실습



sklearn.neural_network.MLPRegressor

class sklearn.neural_network.**MLPRegressor**(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) [source]

Multi-layer Perceptron regressor.

This model optimizes the squared error using LBFGS or stochastic gradient descent.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*

Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

```
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

housing=fetch_california_housing()
print(housing.DESCR)
X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data, housing.target, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full, random_state=42)

mlp_reg=MLPRegressor(hidden_layer_sizes=[50,50,50], random_state=42)
pipeline=make_pipeline(StandardScaler(), mlp_reg)
pipeline.fit(X_train, y_train)

y_pred=pipeline.predict(X_valid)
rmse=mean_squared_error(y_valid, y_pred, squared=False)
print(rmse)
```

```
0.5053326657968681
```

실습(인식)

5-7c.py

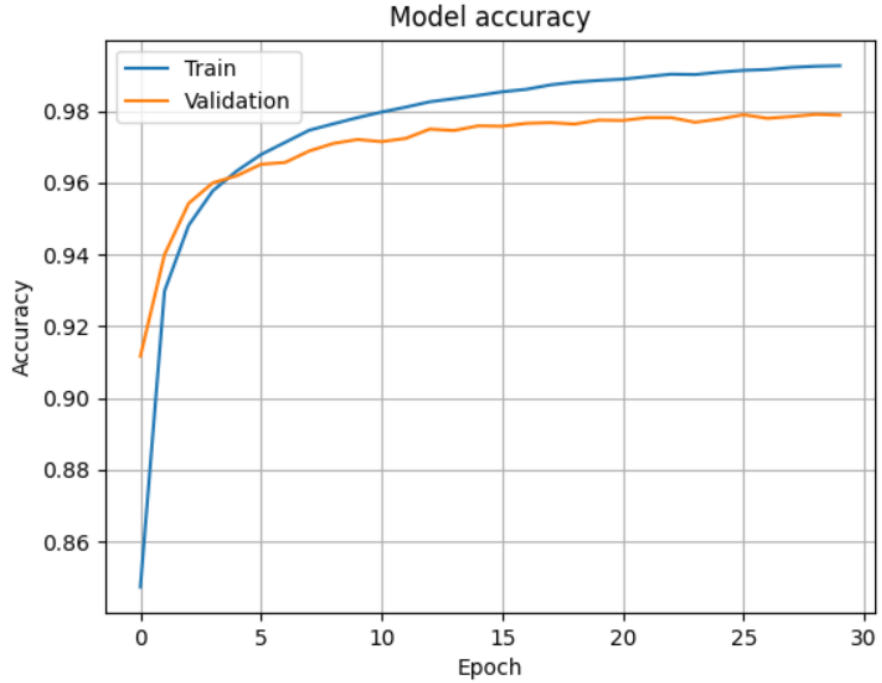
```
#MNIST load (핸즈온머신러닝3rdEd. 방법 사용)
from sklearn.datasets import fetch_openml
mnist=fetch_openml('mnist_784',as_frame=False)

X,y=mnist.data,mnist.target
x_train, x_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]


#파이썬으로 만드는 인공지능 5-7.py 가져옴
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

x_train=x_train.astype(np.float32)/255.0 # ndarray로 변환
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10) # 원핫 코드로 변환
y_test=tf.keras.utils.to_categorical(y_test,10)
```

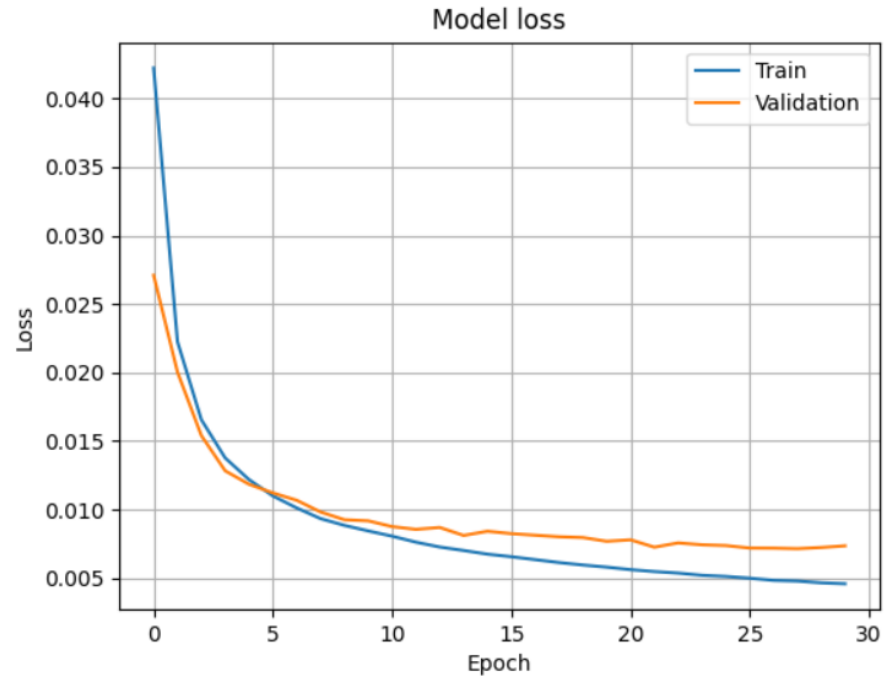
과도한 학습, 학습 곡선의 요동 관찰
Learning rate=0.0005, 0.0001
N_hidden = 2000, 500

```
n_input=784
n_hidden=1024
n_output=10

mlp=Sequential()
mlp.add (Dense(units=n_hidden,activation='tanh',input_shape=(n_input,),kernel_initializer='random_uniform',bias_initializer='zeros'))
mlp.add(Dense(units=n_output,activation='tanh',kernel_initializer='random_uniform',bias_initializer='zeros'))

mlp.compile(loss='mean_squared_error',optimizer=Adam(learning_rate=0.001),metrics=['accuracy'])
hist=mlp.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,y_test),verbose=2)

res=mlp.evaluate(x_test,y_test,verbose=0)
print("정확률은",res[1]*100)
```

```
Epoch 1/30
469/469 - 8s - loss: 0.0422 - accuracy: 0.8474 - val_loss: 0.0271 - val_accuracy: 0.9117
Epoch 2/30
469/469 - 5s - loss: 0.0223 - accuracy: 0.9298 - val_loss: 0.0200 - val_accuracy: 0.9400
```

```python
import matplotlib.pyplot as plt

# 정확률 곡선
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='upper left')
plt.grid()
plt.show()

# 손실 함수 곡선
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='upper right')
plt.grid()
plt.show()
```





5.4.3 fashion MNIST 인식

- fashion MNIST 데이터셋
  - MNIST와 비슷(28x28 입력크기, training sample 60,000, test sample 10,000)
  - 내용이 패션 관련 그림이고 레이블이 {T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot}인 점만 다름

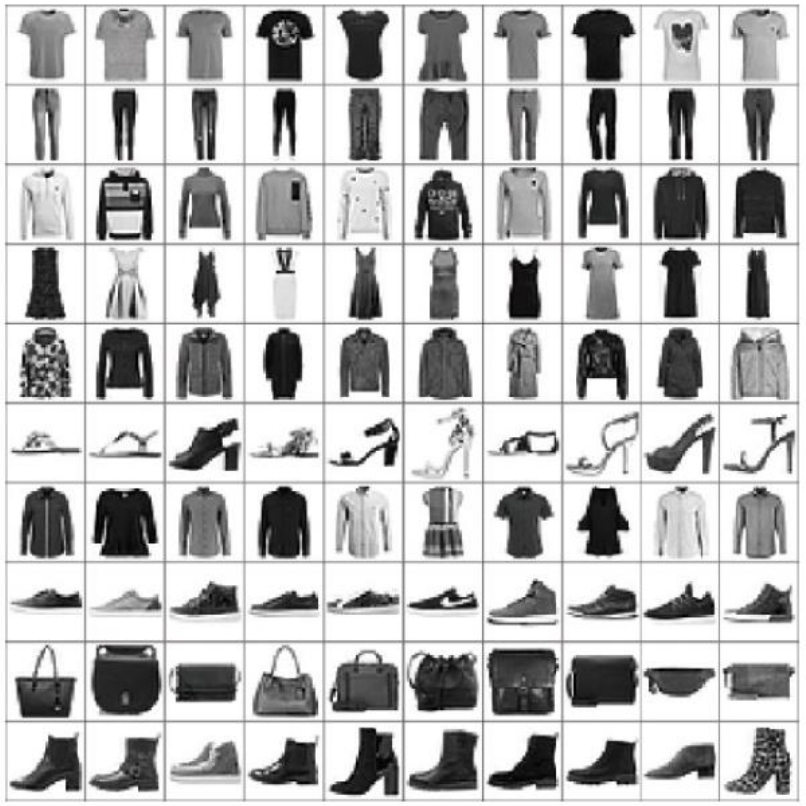


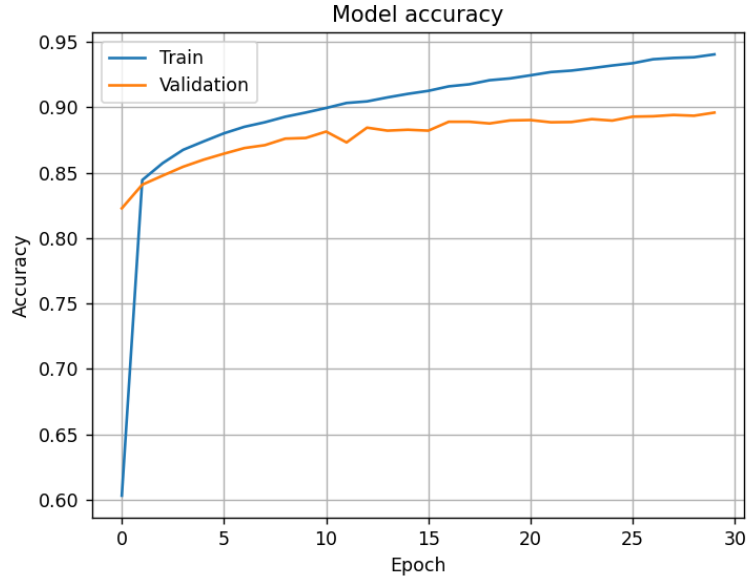
그림 5-7 fashion MNIST 데이터셋

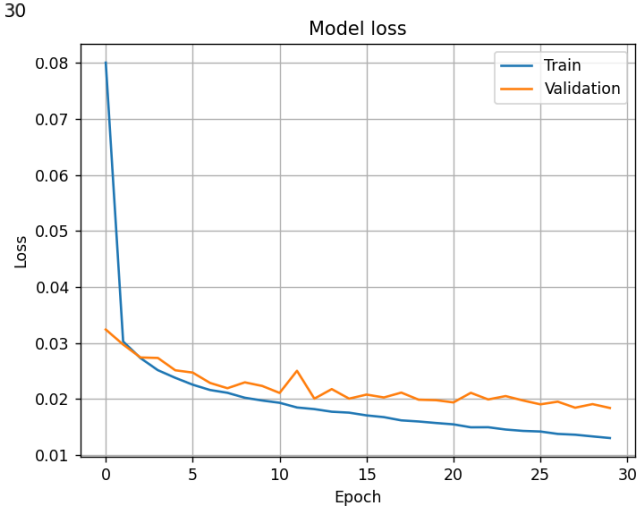
실습(인식)

5-8b.py

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정

path='dataset₩MNISTfashion'
os.chdir(path)

# fashion MNIST 데이터셋을 읽어와 신경망에 입력할 형태로 변환
x_train = np.load('x_train.npy')
y_train = np.load('y_train.npy')
x_test = np.load('x_test.npy')
y_test = np.load('y_test.npy')
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)

x_train = x_train.reshape(60000,784) # 텐서 모양 변환
x_test = x_test.reshape(10000,784)
x_train=x_train.astype(np.float32)/255.0 # ndarray로 변환
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10) # 원핫 코드로 변환
y_test=tf.keras.utils.to_categorical(y_test,10)

n_input=784
n_hidden=1024
n_output=10
```

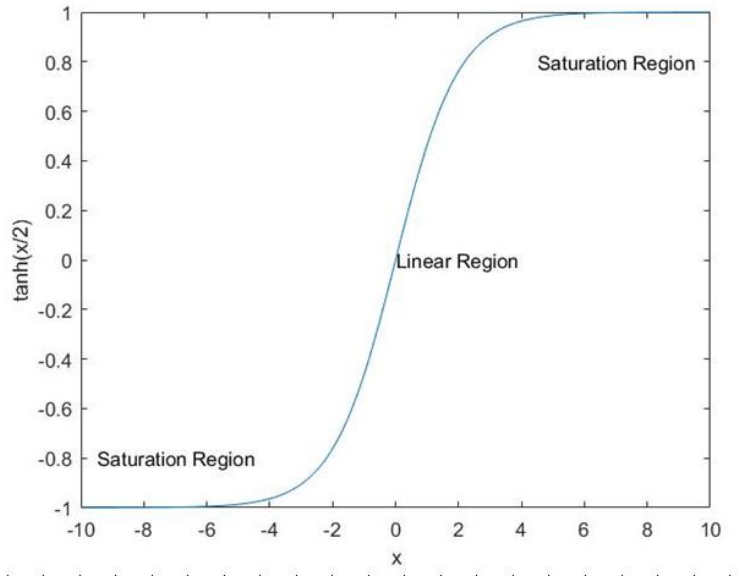
```
mlp=Sequential()
mlp.add (Dense(units=n_hidden,activation='tanh',input_shape=(n_input,),kernel_initializer='random_uniform',bias_initializer='zeros'))
mlp.add(Dense(units=n_output,activation='tanh',kernel_initializer='random_uniform',bias_initializer='zeros'))

mlp.compile(loss='mean_squared_error',optimizer=Adam(learning_rate=0.001),metrics=['accuracy'])
hist=mlp.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,y_test),verbose=2)

res=mlp.evaluate(x_test,y_test,verbose=0)
print("정확률은",res[1]*100)
import matplotlib.pyplot as plt

# 정확률 곡선
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='upper left')
plt.grid()
plt.show()

# 손실 함수 곡선
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='upper right')
plt.grid()
plt.show()
```






```
Epoch 30/30
469/469 - 12s - loss: 0.0130 - accuracy: 0.9405 - val_loss: 0.0184 - val_accuracy: 0.8960 - 12s/epoch - 26ms/step
정확률은 89.60000276565552
```

5.4. 부적절한 포화



$$\delta_k \equiv -\frac{\partial E_{MSE}}{\partial \widehat{y_k}} = (t_k - y_k) f'(\widehat{y_k}) \qquad (5.3.4)$$

Correct Saturation

$y_k \approx t_k, \delta_k \approx 0$

Incorrect Saturation

If $y_k \approx \pm 1$ and $t_k = \mp 1$, $\delta_k \approx 0$ although $|t_k - y_k| \approx 2$

Incorrect Saturation of Output Nodes → Very Slow Convergence of Learning due to $\delta_k \approx 0$

Error Back-Propagation Algorithm: 오차함수 비교

< conv. MSE >

$\delta_k^{out}(\mathbf{x}) = (t_k - y_k) f'(\hat{y}_k)$

. Incorrect Saturation Problem

$$E_m(\mathbf{x}) = \frac{1}{2}\sum_{k=1}^{M}(t_k - y_k)^2$$

< Cross-Entropy Error >

$\delta_k^{out}(\mathbf{x}) = (t_k - y_k)$

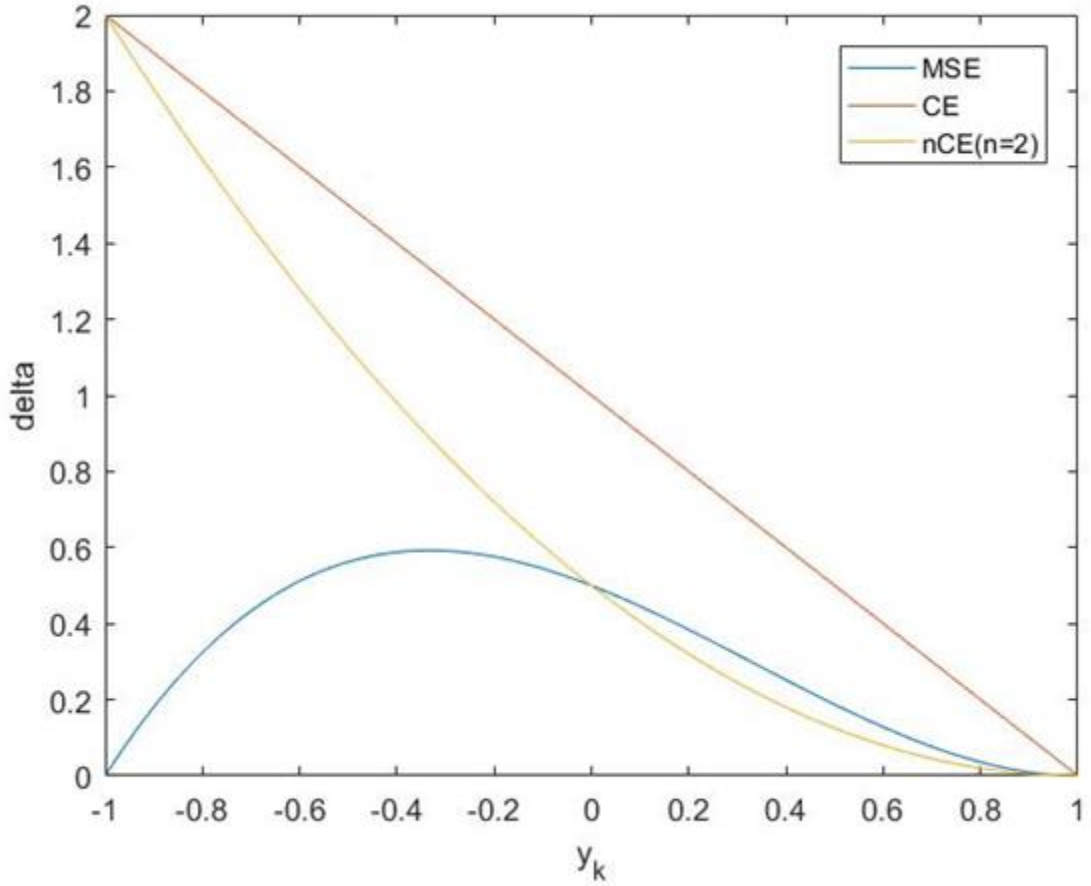
. Overspecialization Problem

$$E_{CE}(\mathbf{x}) = -\sum_{k=1}^{M}\left[(1+t_k)\ln(1+y_k(\mathbf{x})) + (1-t_k)\ln(1-y_k(\mathbf{x}))\right]$$

< n-th order Extension of CE >

$$\delta_k^{out}(\mathbf{x}) = \frac{t_k{}^{n+1}(t_k - y_k)^n}{2^{n-1}}$$

$$E_{nCE} = -\sum_{k=1}^{M}\int \frac{t_k^{n+1}(t_k - y_k)^n}{2^{n-2}(1-y_k)(1+y_k)}dy_k$$



Plot of $\delta_k^{out}(\mathbf{x}^{(p)})$ when $t_k = 1$

Error Back-Propagation Algorithm

오류역전파(EBP) 알고리즘

① 다층퍼셉트론의 가중치들을 초기화 시킨다.

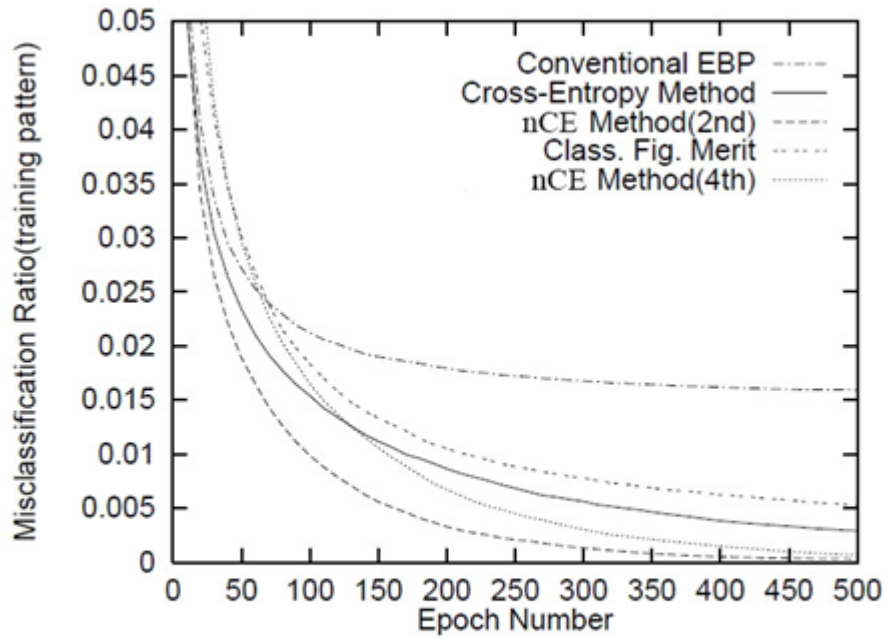
② 학습패턴 $\boldsymbol{x}$가 주어지면, 전방향 계산에 의해 출력노드 값을 구한다.

③ 출력노드의 오류 신호 δ_k를 계산한다.

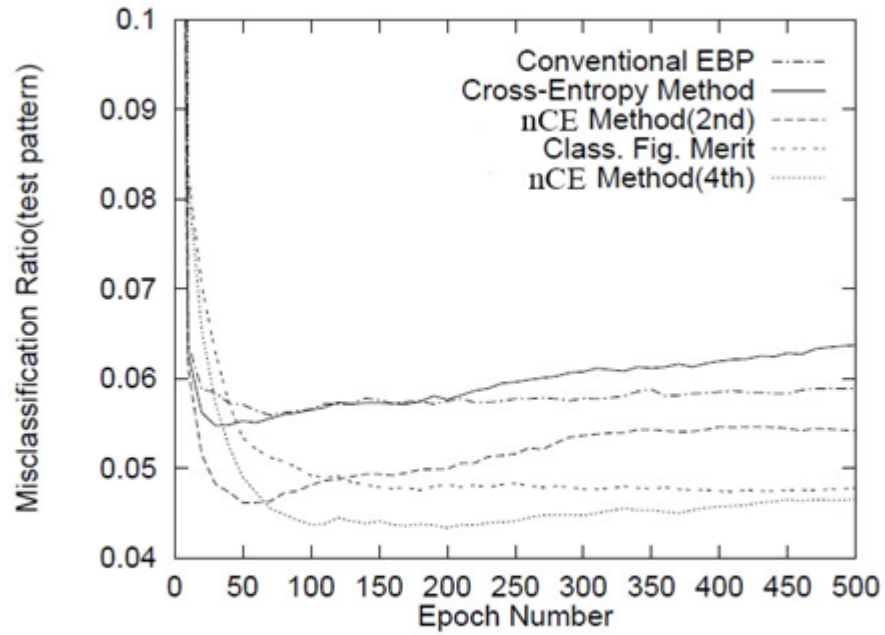
④ 은닉노드의 오류신호 $\delta_j^{(hidden)}$을 역전파 방법에 의해 계산한다.

⑤ 출력층 가중치와 은닉층 가중치의 변경량을 계산한다.

⑥ 출력층 가중치와 은닉층 가중치를 변경시킨다.



(a) 학습패턴에 대한 오인식률

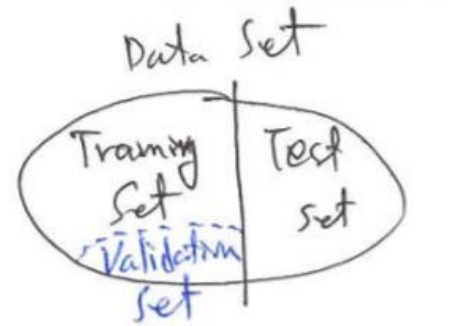


(b) 시험패턴에 대한 오인식률

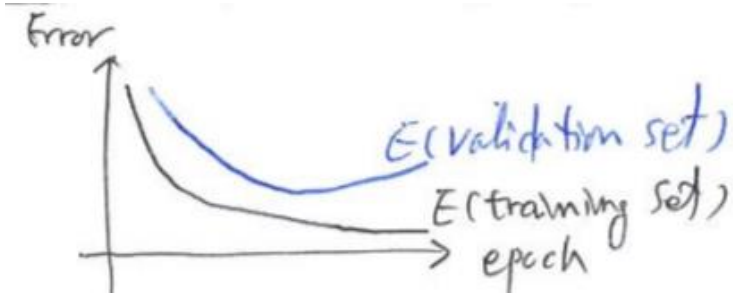
그림 5.7. 필기체 숫자인식 문제의 학습 시뮬레이션 결과

5.5. 학습에 필요한 사항들

- 수렴(Convergence)(?)… 오차가 진동하거나 국소 최소(local minima)에 도달.
- 데이터의 구분
  - 학습데이터: 신경회로망의 파라미터를 학습에 의해 변경시킴
  - 검증데이터: 학습에 따른 성능 향성정도를 파악하여 학습이 잘 된 모델을 선정하는 기준으로 사용
  - 시험데이터: 학습 과정 혹은 종료 및 학습이 잘된 모델 선정에 관여하지 않으며 성능 조사만 수행함
- 초기 가중치의 범위 설정:
  - 작게…혹은 가중치 초기화 방법 사용(조기포화 현상 방지)
- 학습종료 조건:
  - 지정된 epochs 동안 학습시키고 종료
  - 학습데이터에 대한 오차의 문턱값 보다 낮아지면 학습 종료
  - 검증데이터에 대한 오차의 증가 시 학습종료



- 성능 좋은 결과
  - 초기 가중치를 달리 하면서 여러 번 학습 시도
  - 학습데이터 혹은 검증데이터에 대한 성능이 가장 좋은 결과를 취득
  - 앙상블 학습 (8장)



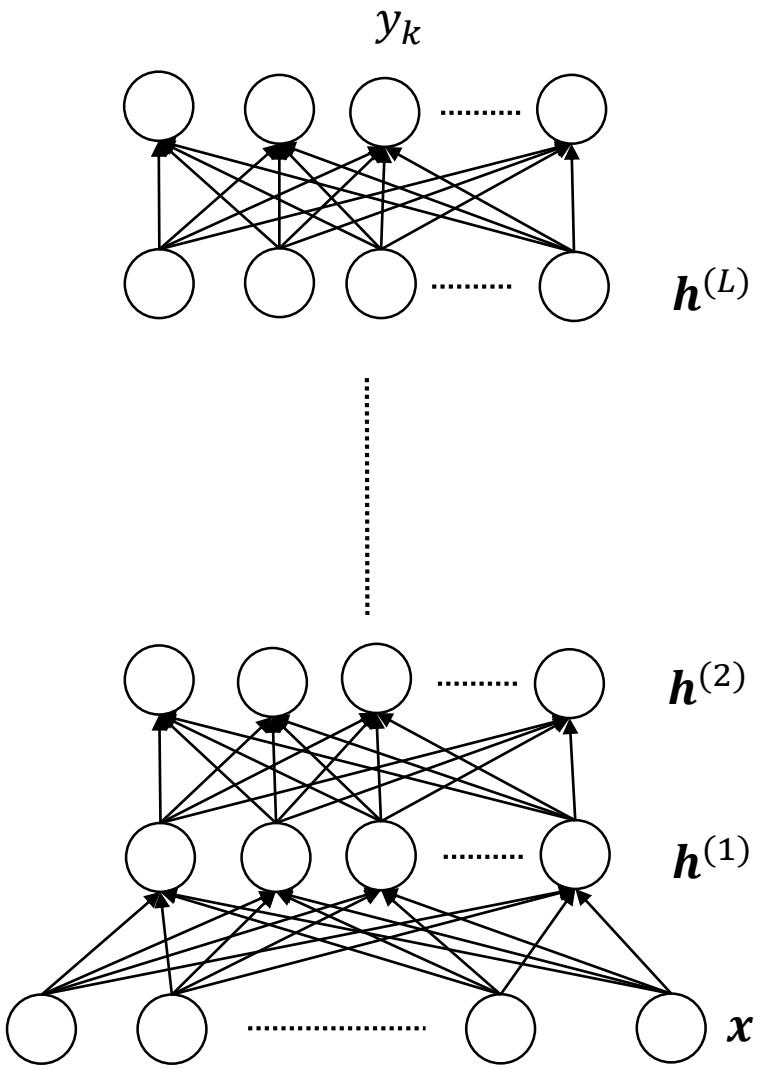
Online, Batch and Learning with Momentum

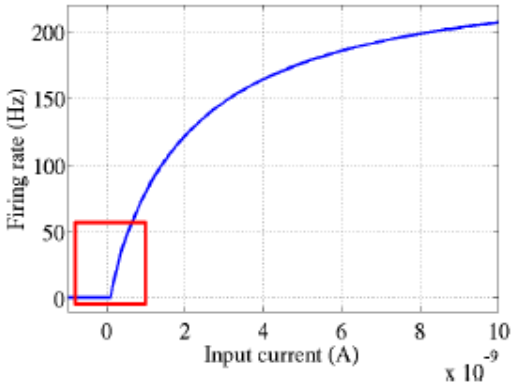
- **Online**:
  - 학습데이터를 하나씩 입력하면서 가중치 변경
- **Batch**:
  - 학습데이터를 입력하면서 가중치 변경량을 계산하여 모든 학습데이터에 대한 변경량을 다 더한 후 이들의 평균으로 가중치를 변경…mini-batch
- **Momentum factor**:
  - 현재 가중치 변경 시 이전 변경량을 반영함
  - 학습에 의한 가중치벡터 변경 방향이 일관성이 있게 하여 학습 속도 개선

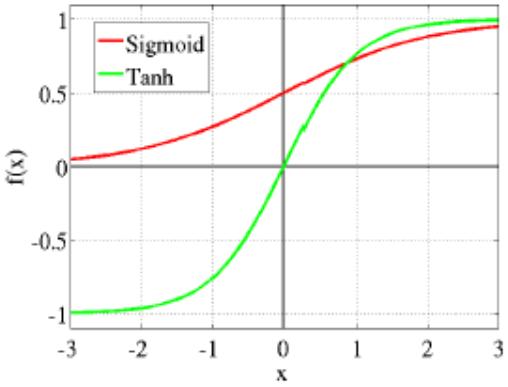
$$\Delta v_{kj}(t) = \eta \delta_k h_j + \alpha \Delta v_{kj}(t-1)$$

$$\Delta w_{ji}(t) = \eta \delta_j^{(hidden)} x_i + \alpha \Delta w_{ji}(t-1)$$

5.7. 심층 신경회로망(Deep Neural Networks)







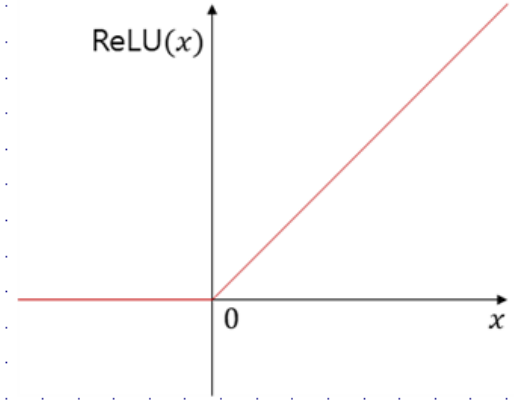
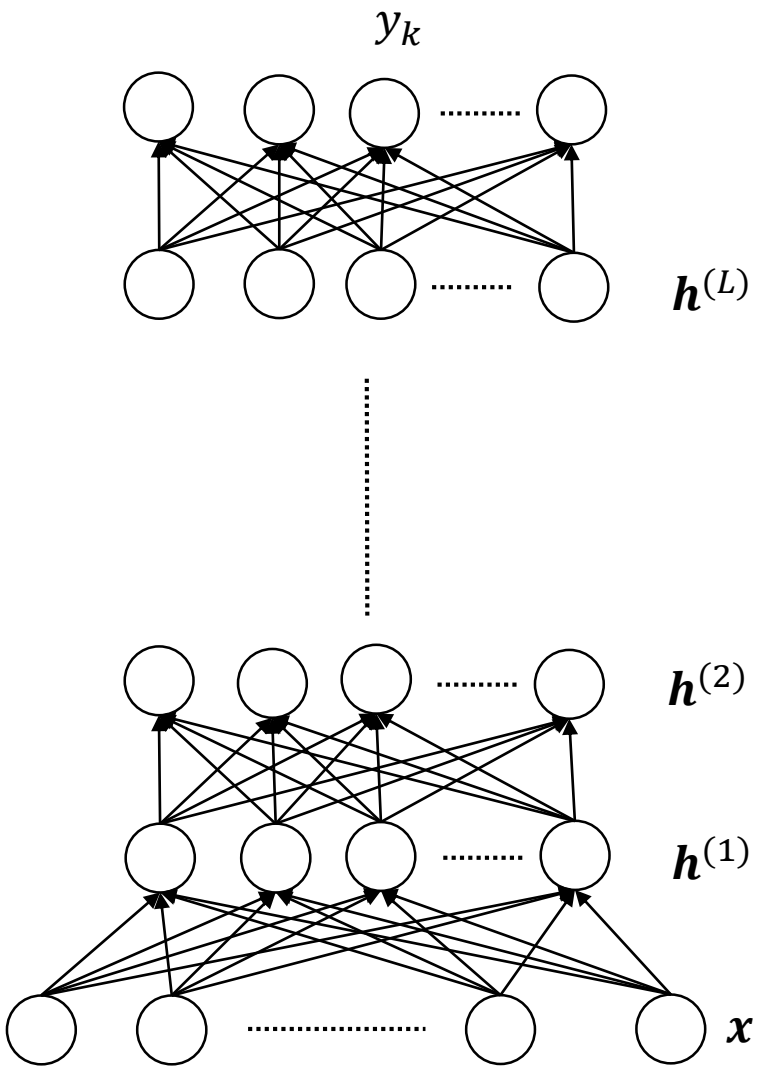


그림 5.10. ReLU 활성화 함수



$$f(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.7.1}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.7.2}$$

$$y_k = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}} \tag{5.7.3}$$

$$E_{CE} = -\sum_k t_k \log(y_k) \tag{5.7.4}$$

$$\partial y_k / \partial \hat{y}_j = \begin{cases} y_k(1 - y_k) & \text{if } j = k \\ -y_k y_j & \text{otherwise} \end{cases} \tag{5.7.5}$$

$$\frac{\partial E_{CE}}{\partial \hat{y}_k} = y_k - t_k \tag{5.7.6}$$

$$\delta_k = -\frac{\partial E_{CE}}{\partial \hat{y}_k} = t_k - y_k \tag{5.7.7}$$

참조: ELUs(Exponential Linear Units)와 GELUs(Gaussian Error Linear Units)

식 (5.7.1)로 주어진 ReLU 활성화 함수는 입력 값 $x<0$의 영역에서 0을 출력하며 기울기도 0이다. 즉, $x<0$에서 정보의 전달과 학습을 위한 가중치의 변경이 이루어지지 않으며, 이를 "dead ReLU" 문제라고 한다. 이 단점을 해결하기 위하여 제안된 활성화 함수가 ELUs(Exponential Linear Units)[17]이며, 그 수식은

$$\mathrm{ELU}_{\alpha}(x)=\begin{cases}x & \text{if } x>0\\ \alpha(\exp(x)-1) & \text{otherwise}\end{cases} \tag{5.7.31}$$

이다. ELU 활성화 함수는 $x<0$이어도 값이 출력되며, 기울기가 0이 아니어서 dead ReLU 문제가 해결된다. 또한, ReLU 활성화 함수는 $x=0$에서 급격하게 변하는 단점이 있는데, ELU는 이 문제도 해결되어 학습의 수렴 속도가 빠르다.

GELUs(Gaussian Error Linear Units)[18] 역시 ReLU 보다 뛰어난 학습 성능을 지니도록 제안된 활성화 함수이다. 식 (5.7.1)의 ReLU는

$$\mathrm{ReLU}(x) = \max(x,0) = x\mathbf{1}(x > 0) \tag{5.7.32}$$

와 같이 표현할 수 있다. 여기서, 1은 지시함수(indicator function)로써 1과 0을 출력한다. GELU는 지시함수 대신에 평균 μ이 0이고 표준편차 σ가 1인 Guassian 확률변수의 CDF(Cummulative Distribution Function) $\Phi(x)$을 사용하여

$$\mathrm{GELU}(x) = x\Phi(x) = x\frac{1}{2}\left[1 + \mathrm{erf}(x/\sqrt{2})\right] \tag{5.7.33}$$

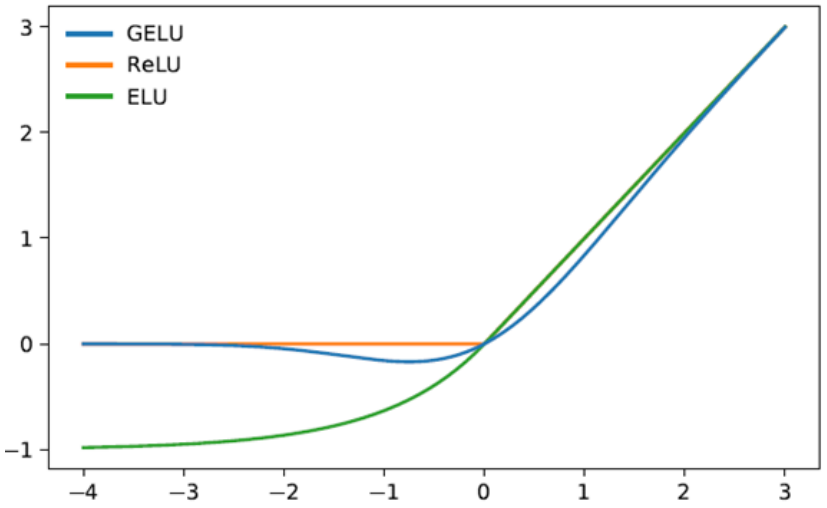
와 같이 주어진다. 여기서, $\Phi(x)$는 erf(error function)으로 계산할 수 있다. GELU의 정확한 값 계산보다 계산 속도를 중시하면

$$\mathrm{GELU}(x) \approx 0.5x\left(1 + \tanh\left[\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right]\right) \tag{5.7.34}$$

혹은

$$\mathrm{GELU}(x) \approx x\sigma(1.702x) \tag{5.7.35}$$

와 같이 근사치를 사용해도 된다. 여기서, $\sigma(.)$는 식 (4.6.3)으로 주어진 시그모이드 함수이다.



GELU는 ReLU를 매끄러운 모양으로 만들은 형태이다. GELU는 단조함수가 아니며 모든 지점에서 곡률을 지닌 것이 ReLU 및 ELU와 다른 특징이다.

이 특징이 GELU가 ReLU 및 ELU보다 더 뛰어난 학습 성능을 지니도록 해준다[18].

예제 5.7-1

그림 5.10과 같이 심층신경회로망이 주어졌다. $h_j^{(l)}(l=,1,2,,,,L)$과 아래층 사이의 연결 가중치를 $w_{ji}^{(l)}(l=,1,2,,,,L)$이라 하고 마지막층 출력노드 y_k와 아래층 $h_j^{(L)}$ 사이의 연결 가중치를 v_{kj} 라고 할 때, 정방향 전파의 계산 과정을 적어보아라. 또한, 출력노드의 오류신호가 $\delta_k = t_k - y_k$로 주어지면 역방향 전파에 의한 은닉노드의 오류신호를 적어보아라.

예제 5.7-2

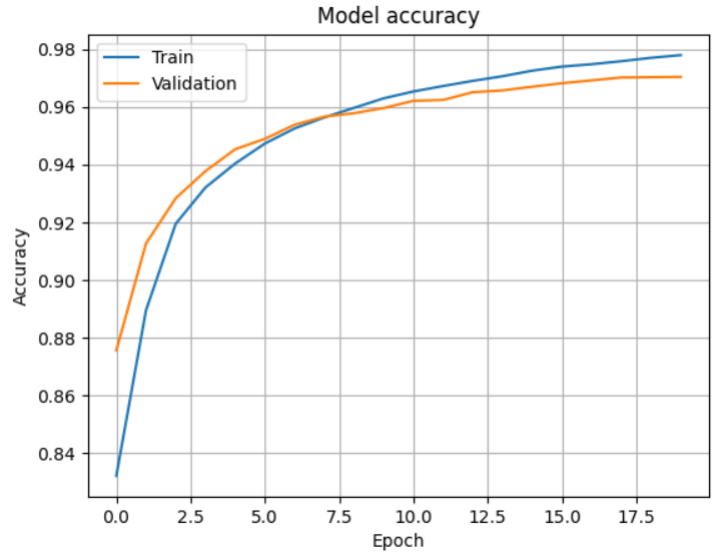
SoftMax 함수는 벡터 요소들 중에서 큰 값은 더 크게, 작은 값은 더 작게 상대적으로 조정하며 그 값들의 합은 1이 되게 한다. SoftMax 함수에 4차원 벡터 [4 10 6 5]가 입력되었을 때, SoftMax 함수를 통과한 후의 4차원 벡터를 구하여 보아라.

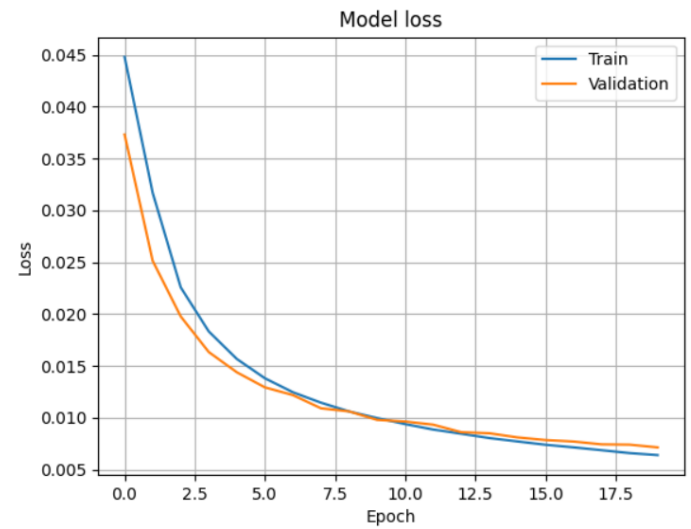
실습(인식)

5-9c.py

```
#MNIST load (핸즈온머신러닝3rdEd. 방법 사용)
from sklearn.datasets import fetch_openml
mnist=fetch_openml('mnist_784',as_frame=False)

X,y=mnist.data,mnist.target
x_train, x_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

# 숫자 그리기 기능 추가
from PIL import Image
import numpy as np
def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

img = x_train[0]
label = y_train[0]
print(label)  # 5
print(img.shape)  # (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
print(img.shape)  # (28, 28)
img_show(img)

#파이썬으로 만드는 인공지능 5-9.py 가져옴
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

x_train=x_train.astype(np.float32)/255.0 # ndarray로 변환
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10) # 원핫 코드로 변환
y_test=tf.keras.utils.to_categorical(y_test,10)

# 신경망 구조 설정
n_input=784
n_hidden1=1024
n_hidden2=512
n_hidden3=512
n_hidden4=512
n_output=10
```

```
# 신경망 구조 설계
mlp=Sequential()
mlp.add(Dense(units=n_hidden1,activation='tanh',input_shape=(n_input,),kernel_initializer='random_uniform',bias_initializer='zeros'))
mlp.add(Dense(units=n_hidden2,activation='tanh',kernel_initializer='random_uniform',bias_initializer='zeros'))
mlp.add(Dense(units=n_hidden3,activation='tanh',kernel_initializer='random_uniform',bias_initializer='zeros'))
mlp.add(Dense(units=n_hidden4,activation='tanh',kernel_initializer='random_uniform',bias_initializer='zeros'))
mlp.add(Dense(units=n_output,activation='tanh',kernel_initializer='random_uniform',bias_initializer='zeros'))

# 신경망 학습
mlp.compile(loss='mean_squared_error',optimizer=Adam(learning_rate=0.00005),metrics=['accuracy'])
hist=mlp.fit(x_train,y_train,batch_size=128,epochs=20,validation_data=(x_test,y_test),verbose=2)

# 신경망의 정확률 측정
res=mlp.evaluate(x_test,y_test,verbose=0)
print("정확률은",res[1]*100)

import matplotlib.pyplot as plt

# 정확률 곡선
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='upper left')
plt.grid()
plt.show()

# 손실 함수 곡선
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='upper right')
plt.grid()
plt.show()
```





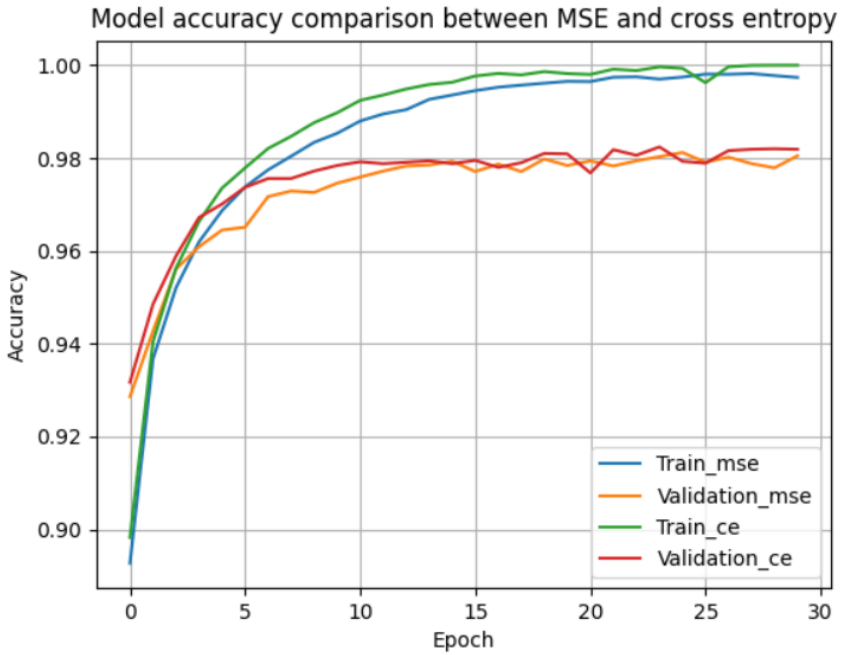

```
Epoch 20/20
469/469 - 20s - loss: 0.0064 - accuracy: 0.9781 - val_loss: 0.0071 - val_accurac
y: 0.9705
정확률은 97.04999923706055
```

실습(인식)

5-10c.py

```
#MNIST load (핸즈온머신러닝3rdEd. 방법 사용)
from sklearn.datasets import fetch_openml
mnist=fetch_openml('mnist_784',as_frame=False)

X,y=mnist.data,mnist.target
x_train, x_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

#파이썬으로 만드는 인공지능 5-10.py 가져옴
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

x_train=x_train.astype(np.float32)/255.0 # ndarray로 변환
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10) # 원핫 코드로 변환
y_test=tf.keras.utils.to_categorical(y_test,10)

# 신경망 구조 설정
n_input=784
n_hidden1=1024
n_hidden2=512
n_hidden3=512
n_hidden4=512
n_output=10
```

```
# 평균제곱오차를 사용한 모델
dmlp_mse=Sequential()
dmlp_mse.add(Dense(units=n_hidden1,activation='tanh',input_shape=(n_input,)))
dmlp_mse.add(Dense(units=n_hidden2,activation='tanh'))
dmlp_mse.add(Dense(units=n_hidden3,activation='tanh'))
dmlp_mse.add(Dense(units=n_hidden4,activation='tanh'))
dmlp_mse.add(Dense(units=n_output,activation='softmax'))
dmlp_mse.compile(loss='mean_squared_error',optimizer=Adam(learning_rate=0.0001),metrics=['accuracy'])
hist_mse=dmlp_mse.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,y_test),verbose=2)

# 교차 엔트로피를 사용한 모델
dmlp_ce=Sequential()
dmlp_ce.add(Dense(units=n_hidden1,activation='tanh',input_shape=(n_input,)))
dmlp_ce.add(Dense(units=n_hidden2,activation='tanh'))
dmlp_ce.add(Dense(units=n_hidden3,activation='tanh'))
dmlp_ce.add(Dense(units=n_hidden4,activation='tanh'))
dmlp_ce.add (Dense(units=n_output,activation='softmax'))
dmlp_ce.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.0001),metrics=['accuracy'])
hist_ce=dmlp_ce.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,y_test),verbose=2)

# 두 모델의 정확률 비교
res_mse=dmlp_mse.evaluate(x_test,y_test,verbose=0)
print("평균제곱오차의 정확률은",res_mse[1]*100)
res_ce=dmlp_ce.evaluate(x_test,y_test,verbose=0)
print("교차 엔트로피의 정확률은",res_ce[1]*100)

# 하나의 그래프에서 두 모델을 비교
import matplotlib.pyplot as plt
plt.plot(hist_mse.history['accuracy'])
plt.plot(hist_mse.history['val_accuracy'])
plt.plot(hist_ce.history['accuracy'])
plt.plot(hist_ce.history['val_accuracy'])
plt.title('Model accuracy comparison between MSE and cross entropy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train_mse','Validation_mse','Train_ce','Validation_ce'], loc='best')
plt.grid()
plt.show()
```



평균제곱오차의 정확률은 98.04999828338623
교차 엔트로피의 정확률은 98.18999767303467