k-Nearest Neighbors, Multilayer Neural Network

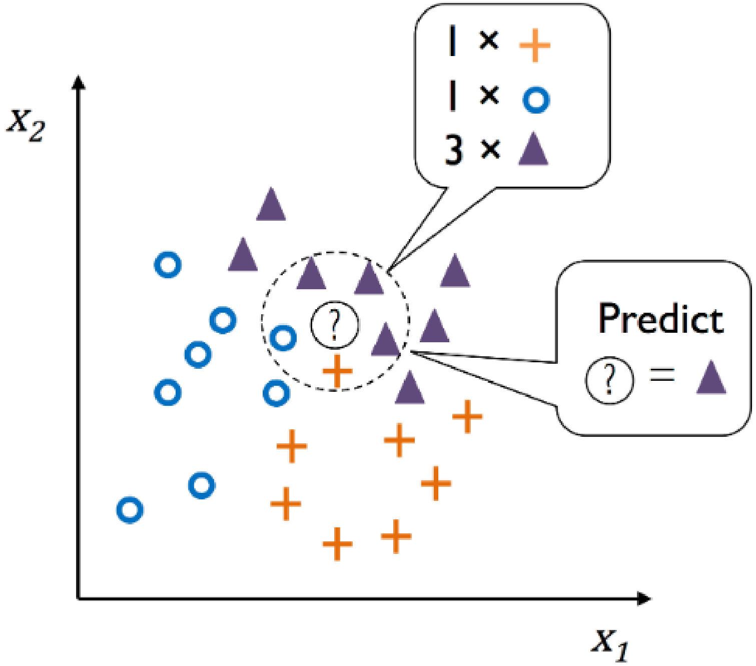
Machine Learning

K-Nearest Neighbors

- What is the K-Nearest Neighbors?
  - In pattern recognition, the K-Nearest Neighbors algorithm(K-NN) is non-parametric method used for classification and regression

- Algorithm

1. Choose the K and distance metric
2. Find K nearest neighbors for the test sample for test sample
3. Assign classified label by majority vote



K-Nearest Neighbors Using Scikit-learn

- Load Iris Dataset

```
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
X = iris.data[50:150, [2, 3]]
y = iris.target[50:150]
print('Class labels:', np.unique(y))
```

```
Class labels: [1 2]
```

- Splitting data into 70% training data & 30% test data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.3, random_state=1, stratify=y)

print('Labels counts in y:', np.bincount(y))
print('Labels counts in y_train:', np.bincount(y_train))
print('Labels counts in y_test:', np.bincount(y_test))
```

```
Labels counts in y: [0 50 50]
Labels counts in y_train: [0 35 35]
Labels counts in y_test: [0 15 15]
```

K-Nearest Neighbors Using Scikit-learn

- Standardize the dataset

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

- Building a K-Nearest Neighbor and Training the model

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5, p=2,metric='minkowski')

knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
      metric_params=None, n_jobs=1, n_neighbors=5, p=2,
      weights='uniform')
```

K-Nearest Neighbors Using Scikit-learn

- Evaluation

```
# Train accuracy
acc = knn.score(X_train_std, y_train)
print("Train accuracy : %.4f" % acc)
```

```
Train accuracy : 0.9429
```

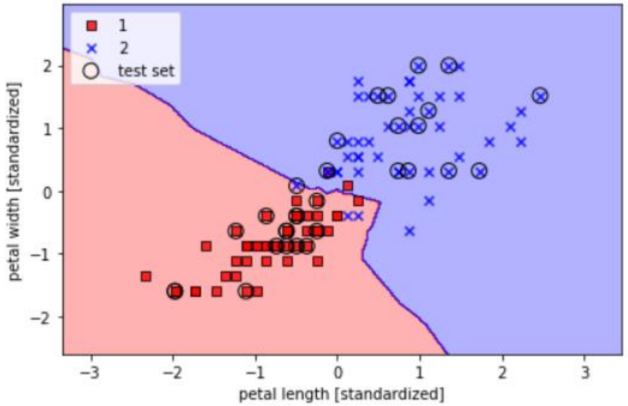
```
# Test accuracy
acc = knn.score(X_test_std, y_test)
print("Test accuracy : %.4f" % acc)
```

```
Test accuracy : 0.9667
```

K-Nearest Neighbors Using Scikit-learn

- Plotting decision boundary

```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

plot_decision_regions(X_combined_std, y_combined,
classifier=knn, test_idx=range(70, 100))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



K-Nearest Neighbors Using Scikit-learn

- Try other k values

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=9,
                           p=2,
                           metric='minkowski')
knn.fit(X_train_std, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=9, p=2,
                     weights='uniform')
```

```
# Train accuracy
acc = knn.score(X_train_std, y_train)
print("Train accuracy : %.4f" % acc)
```

```
Train accuracy : 0.9286
```

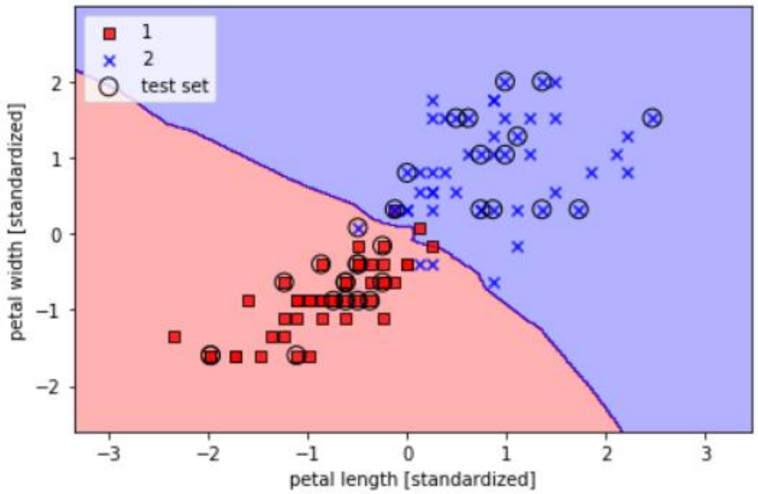
```
# Test accuracy
acc = knn.score(X_test_std, y_test)
print("Test accuracy : %.4f" % acc)
```

```
Test accuracy : 0.9667
```

K-Nearest Neighbors Using Scikit-learn

- Try other k values

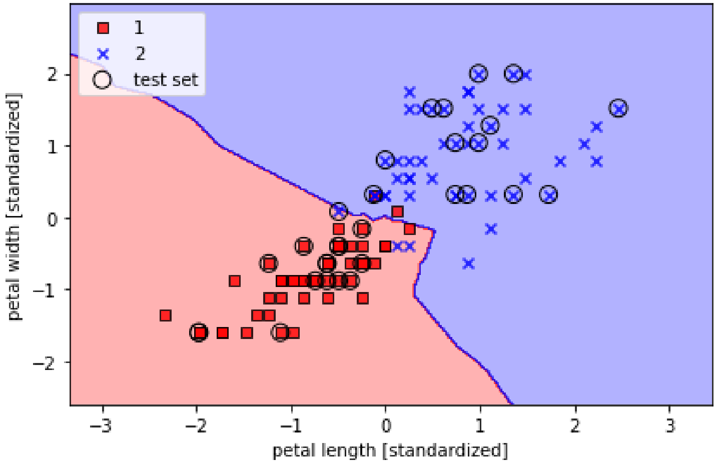
```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X_combined_std, y_combined,
classifier=knn, test_idx=range(70, 100))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



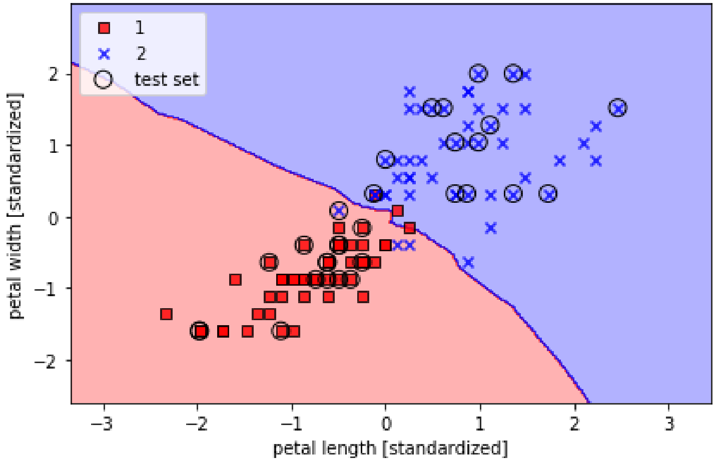
Multi-layer Neural Networks

- K : Number of Nearest Neighbors

K = 5

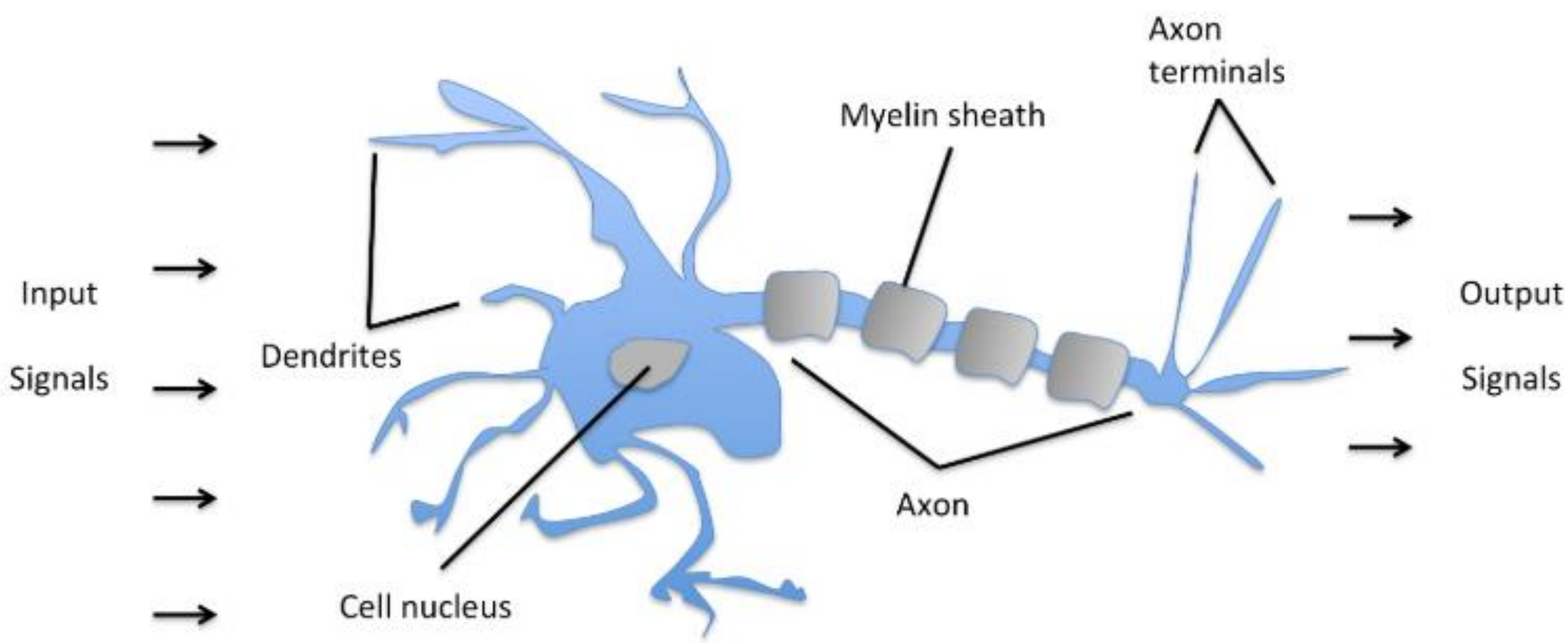


K = 9



Multi-layer Neural Networks

- The Neuron
  - Input signals accumulate and exceed a certain threshold value, an output signal is generated



Multi-layer Neural Networks

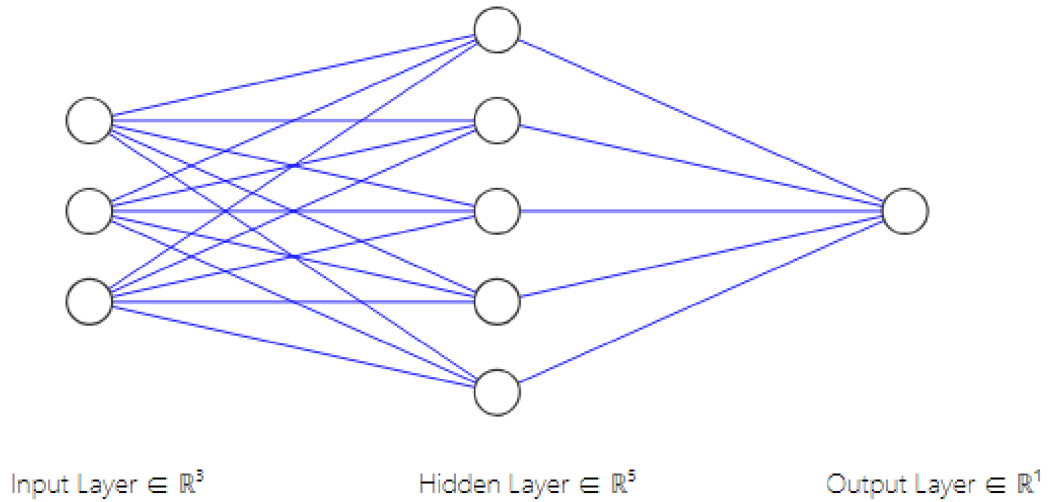
- Analogy with human brain
  - Human
    - 10^{11} Neurons
    - 10^4 synapses per neuron
    - 10^{16} operations per sec
    - 250M neurons per mm^3
    - 180,000km of “wires”

  - GPU(Graphics Processing Unit)
    - $8 * 10^{12}$ operations per sec
    - 5760 (small) cores
    - $2,000

https://www.slideshare.net/braincreators/introduction-to-deep-neural-networks

Multi-layer Neural Networks

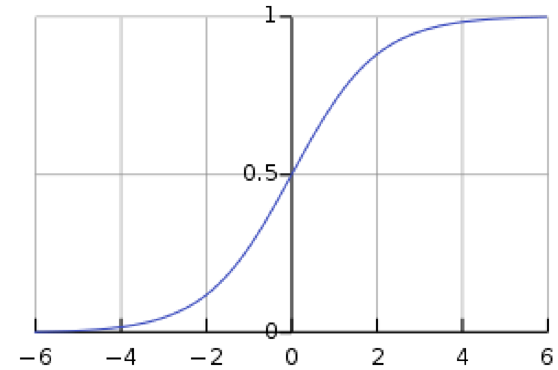
- Neural Network : One hidden layer
  - A computer model of the human brain and nervous system



  - Activation function : Sigmoid

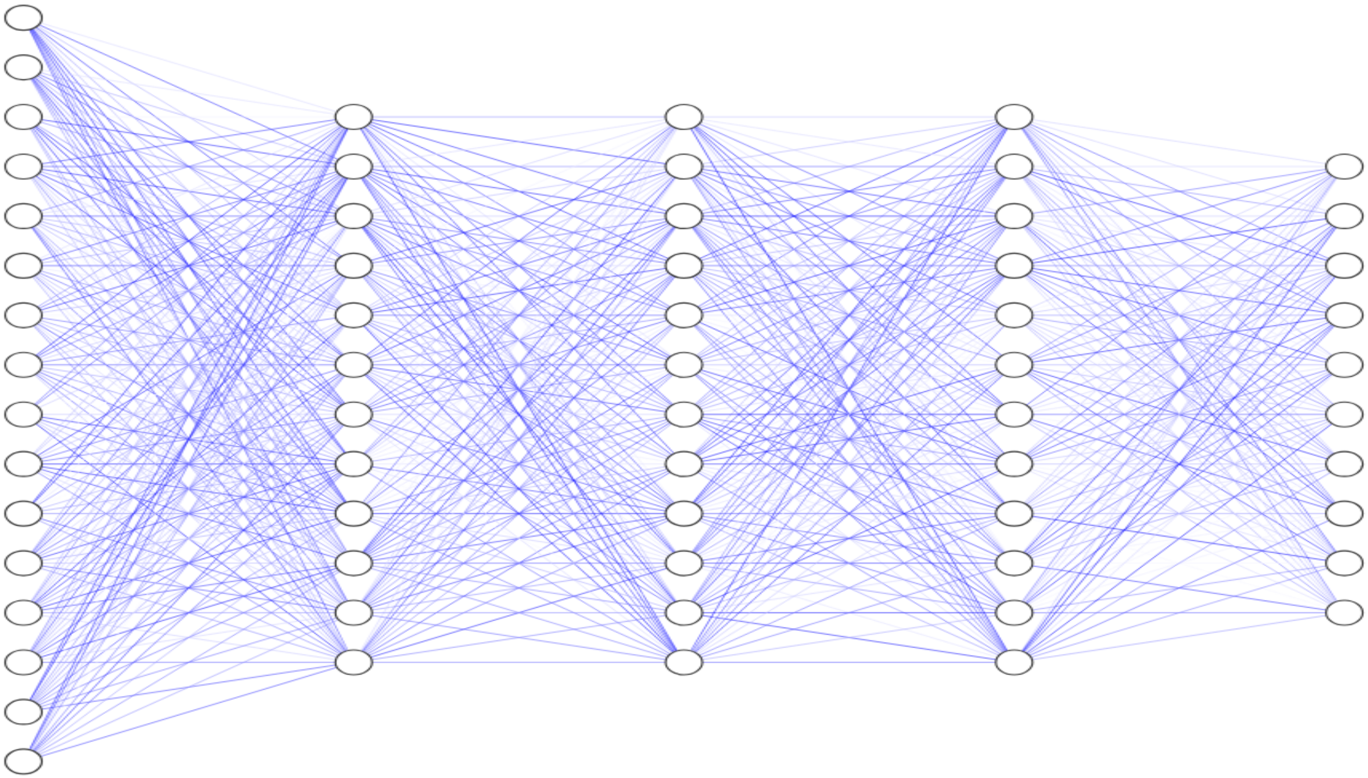
$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



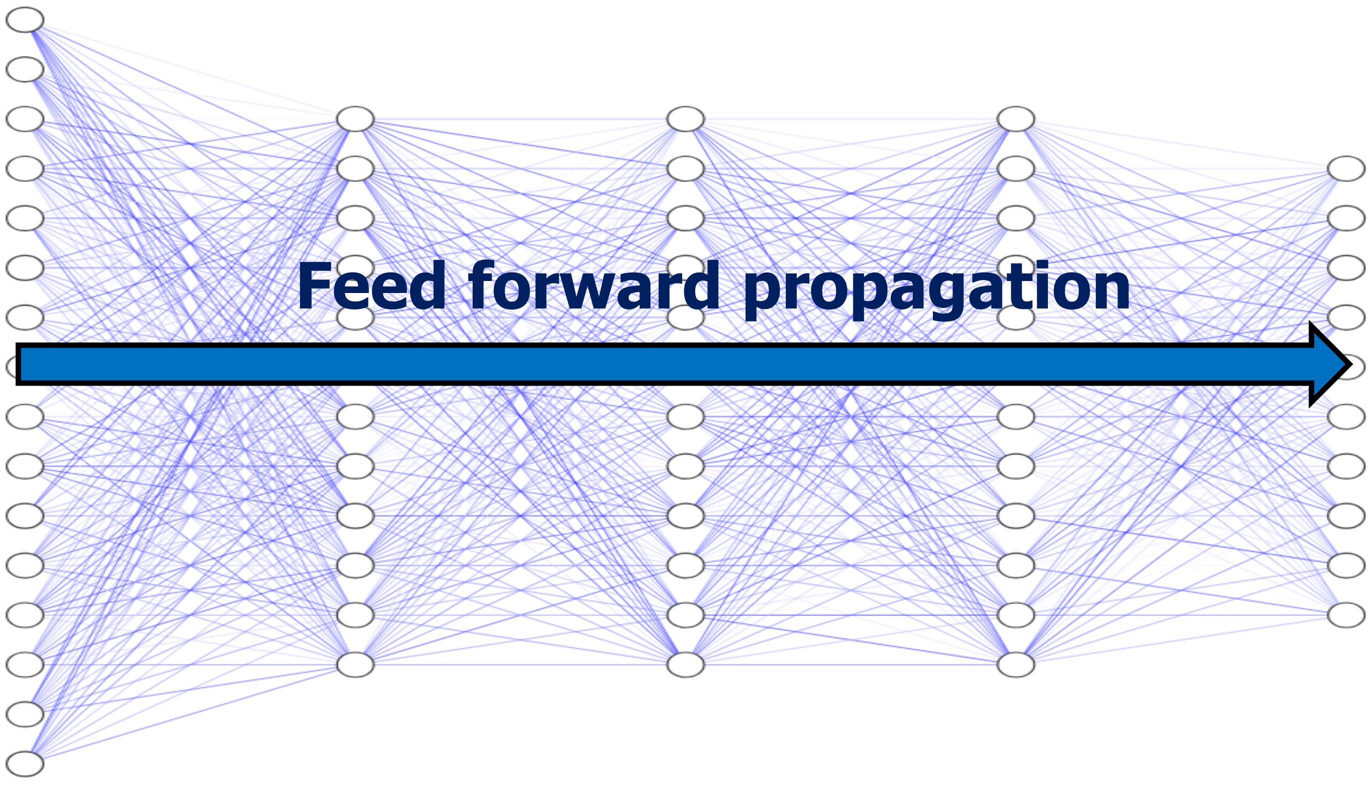
Multi-layer Neural Networks

- Deep Neural Network(Multi-layer Neural Networks)
  - A neural network with a certain level of complexity, a neural network with more than two layers



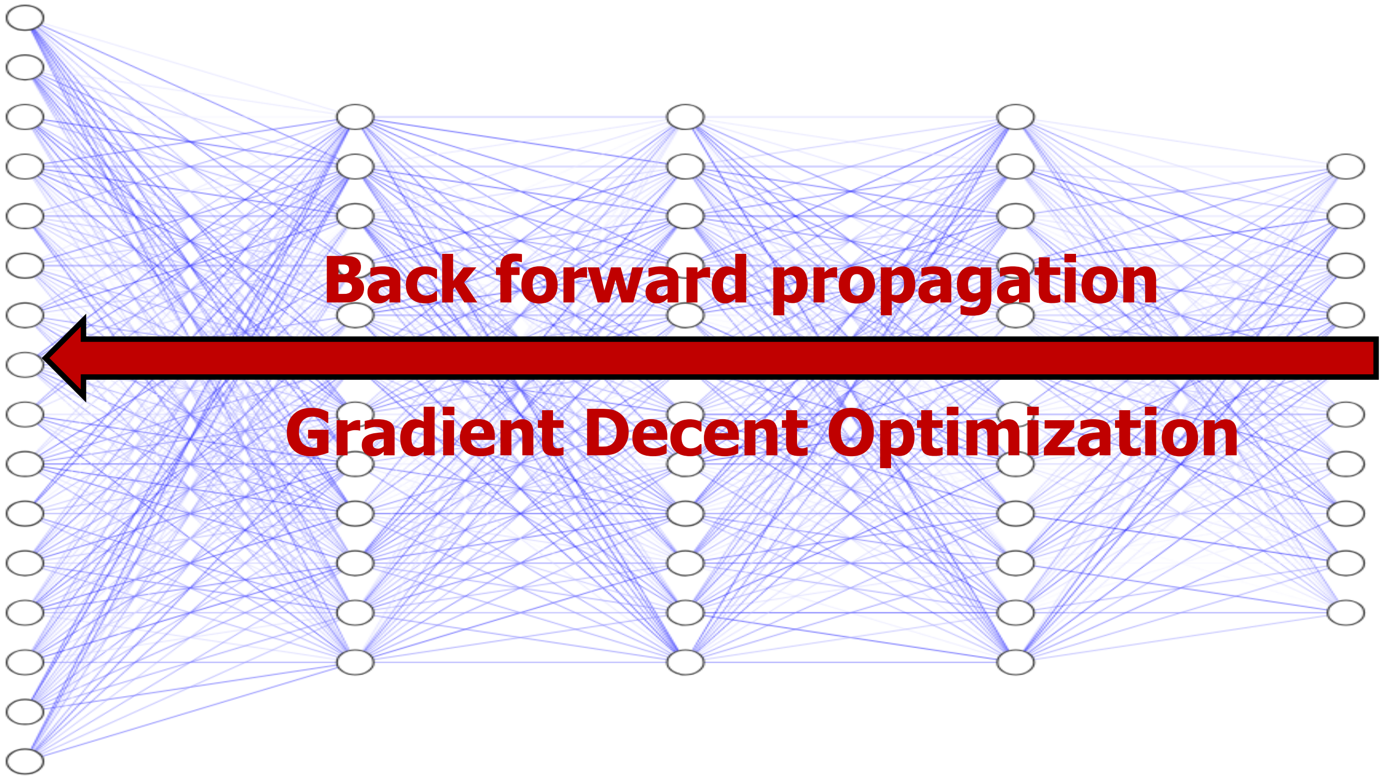
Multi-layer Neural Networks

- Deep Neural Network(Multi-layer Neural Networks)
  - A neural network with a certain level of complexity, a neural network with more than two layers



Multi-layer Neural Networks

- Deep Neural Network(Multi-layer Neural Networks)
  - A neural network with a certain level of complexity, a neural network with more than two layers



Multi-layer Neural Networks

- Deep Neural Network(Multi-layer Neural Networks)
  - Activation function

Sigmoid

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

Leaky ReLU

$$max(0.1x, x)$$

tanh

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

ELU(Exponential Linear Unit)

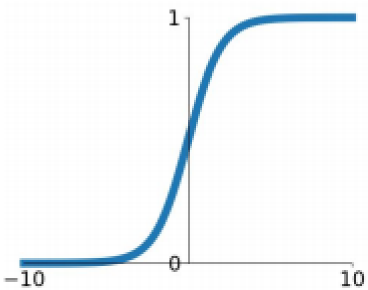
$$\begin{cases} x, & x < 0 \\ \alpha(e^{x}-1), & x \geq 0 \end{cases}$$

ReLU(Rectified Linear Unit)

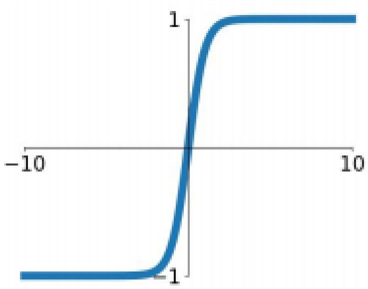
$$max(0, x)$$

Multi-layer Neural Networks

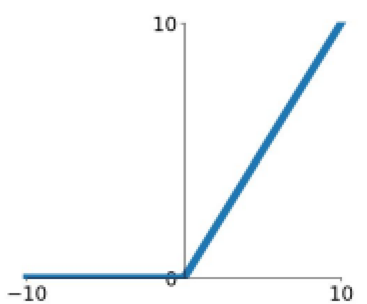
- Deep Neural Network(Multi-layer Neural Networks)
  - Activation function



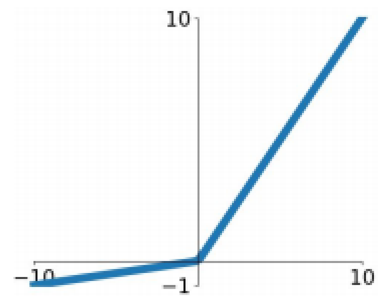
Sigmoid



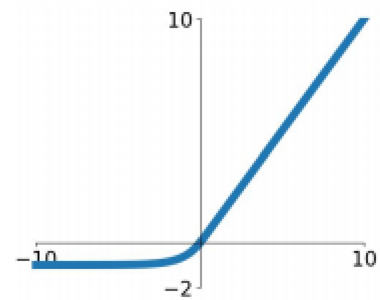
tanh



ReLU



Leaky ReLU



ELU

Multi-layer Neural Networks Using Scikit-learn

- Load Iris Dataset

```python
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
X = iris.data[50:150, [2, 3]]
y = iris.target[50:150]
print('Class labels:', np.unique(y))
```

```
Class labels: [1 2]
```

- Splitting data into 70% training data & 30% test data

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.3, random_state=1, stratify=y)

print('Labels counts in y:', np.bincount(y))
print('Labels counts in y_train:', np.bincount(y_train))
print('Labels counts in y_test:', np.bincount(y_test))
```

```
Labels counts in y: [0 50 50]
Labels counts in y_train: [0 35 35]
Labels counts in y_test: [0 15 15]
```

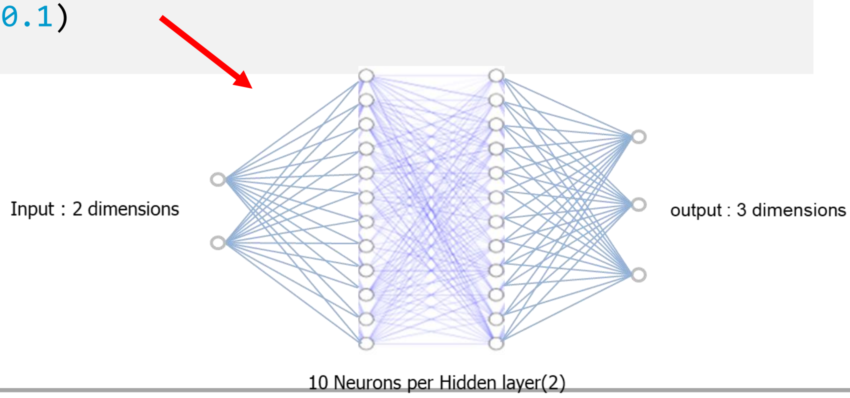
Multi-layer Neural Networks Using Scikit-learn

- Standardize the dataset

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

- Building a Multi-layer Neural Network and Training the model

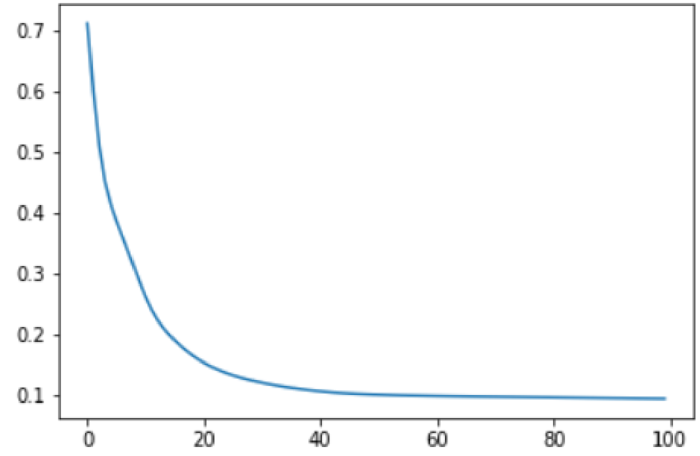
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=100, alpha=1e-4,
                    solver='sgd', verbose=1, tol=1e-4, random_state=0,
                    learning_rate_init=0.1)
mlp.fit(X_train_std, y_train)
```



Multi-layer Neural Networks Using Scikit-learn

- Loss curve

```
# plot the loss
plt.plot(mlp.loss_curve_)
plt.show()
```



- Evaluation

```
# Train accuracy
acc = mlp.score(X_train_std, y_train)
print("Train accuracy : %.4f" % acc)
```

```
Train accuracy : 0.9429
```

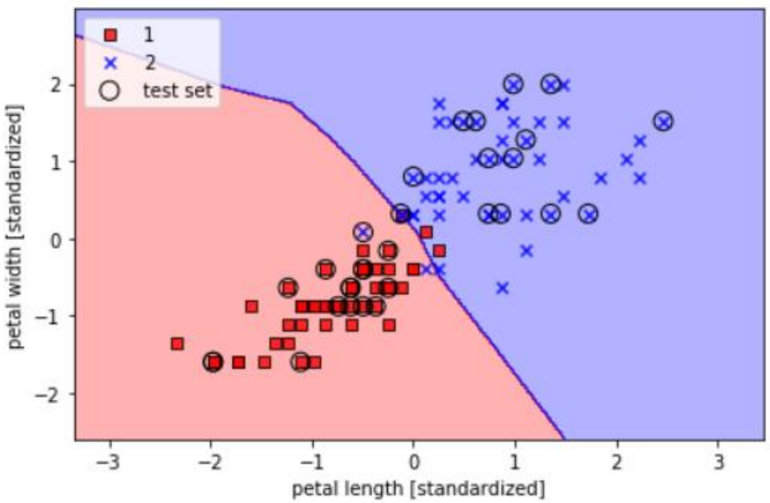
```
# Test accuracy
acc = mlp.score(X_test_std, y_test)
print("Test accuracy : %.4f" % acc)
```

```
Test accuracy : 0.9667
```

Multi-layer Neural Networks Using Scikit-learn

- Plotting decision boundary

```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

plot_decision_regions(X_combined_std, y_combined,
classifier=knn, test_idx=range(70, 100))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



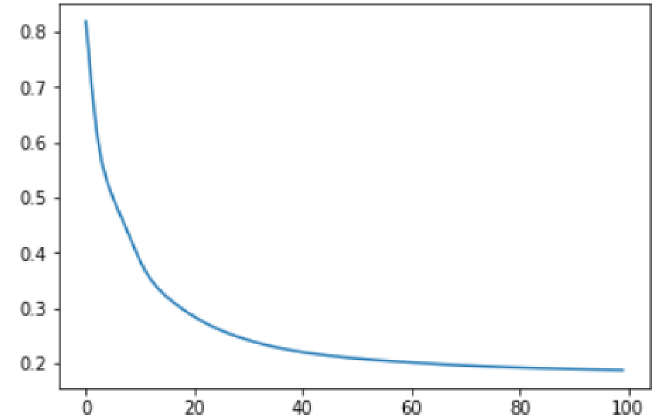
Multi-layer Neural Networks Using Scikit-learn

- Try other regularization parameters (alpha) – (1)

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=100, alpha=1,
                    solver='sgd', verbose=0, tol=1e-6, random_state=0,
                    learning_rate_init=0.1)
mlp.fit(X_train_std, y_train)
```

```
MLPClassifier(activation='relu', alpha=1, batch_size='auto', beta_1=0.9,
          beta_2=0.999, early_stopping=False, epsilon=1e-08,
          hidden_layer_sizes=(10, 10), learning_rate='constant',
          learning_rate_init=0.1, max_iter=100, momentum=0.9,
          nesterovs_momentum=True, power_t=0.5, random_state=0, shuffle=True,
          solver='sgd', tol=1e-06, validation_fraction=0.1, verbose=0,
          warm_start=False)
```

```
# plot the loss
plt.plot(mlp.loss_curve_)
plt.show()
```



Multi-layer Neural Networks Using Scikit-learn

- Try other regularization parameters (alpha) – (2)

```
# Train accuracy
acc = mlp.score(X_train_std, y_train)
print("Train accuracy : %.4f" % acc)
```

```
Train accuracy : 0.9286
```

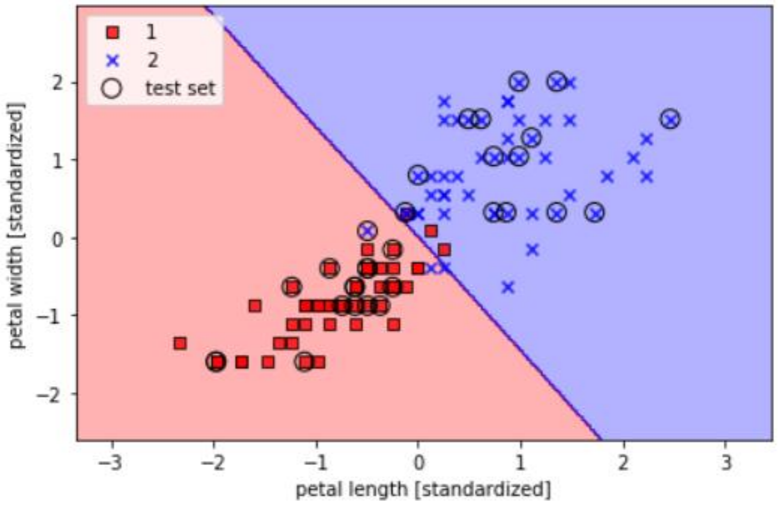
```
# Test accuracy
acc = mlp.score(X_test_std, y_test)
print("Test accuracy : %.4f" % acc)
```

```
Test accuracy : 0.9667
```

Multi-layer Neural Networks Using Scikit-learn

- Try other regularization parameters (alpha) – (3)

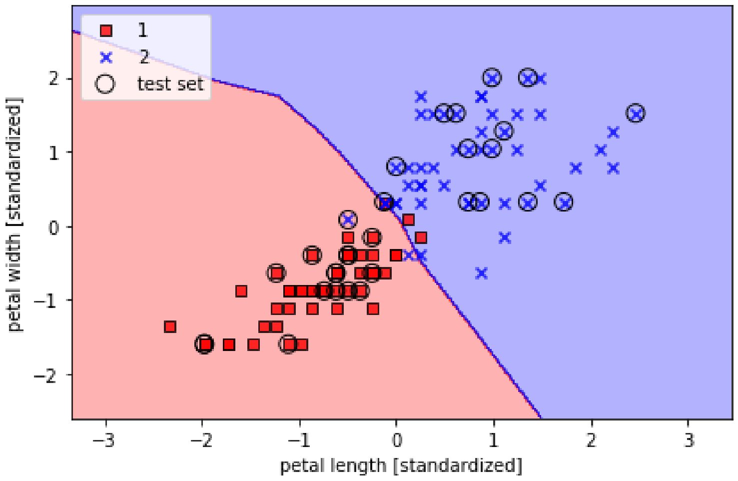
```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

plot_decision_regions(X_combined_std, y_combined,
classifier=mlp, test_idx=range(70, 100))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



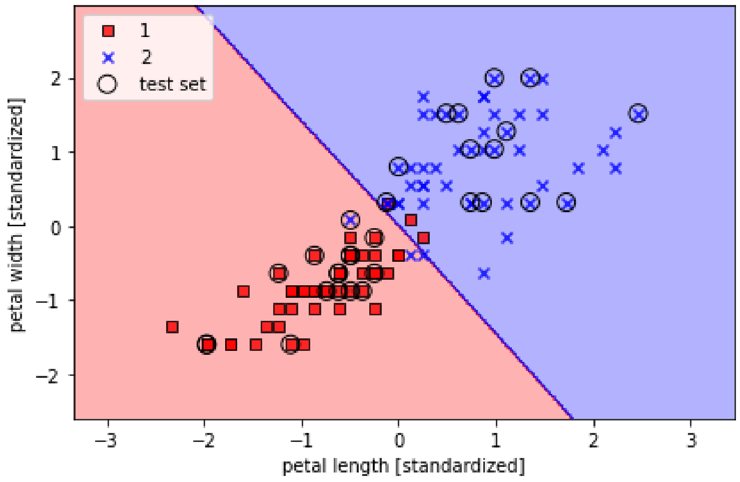
Multi-layer Neural Networks Using Scikit-learn

- Regularization ratio - alpha

alpha = 1e-4



alpha = 1



Multi-layer Neural Networks – Image Classification

- MNIST dataset

- A set of 70,000 small handwritten digits
- 60,000 training images and 10,000 test images
- 28 x 28 grayscale pixels
- commonly used for training various image processing systems

Multi-layer Neural Networks – Image Classification

- Load MNIST Dataset

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import io

mnist = io.loadmat('mnist-original.mat')
mnist
```

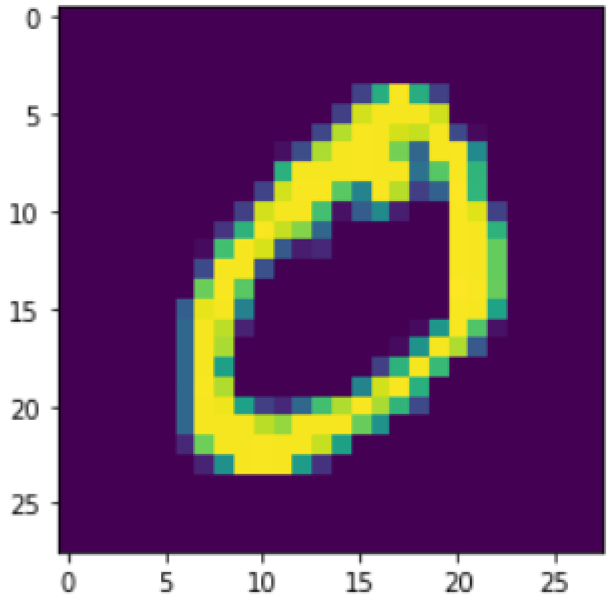
- Get X and y –(1)

```
X, y = mnist['data'], mnist['label']
X = np.array(X).T
y = np.array(y).T.ravel()
X.shape

# rescale the data, use the traditional train/test split
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

Multi-layer Neural Networks – Image Classification

- Get X and y – (2)

```
ex1 = X[0]
ex1_image = ex1.reshape(28, 28)
plt.imshow(ex1_image)
plt.show()
```



Multi-layer Neural Networks – Image Classification

- Splitting data into 60k training data & 10k test data

```
# rescale the data, use the traditional train/test split
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

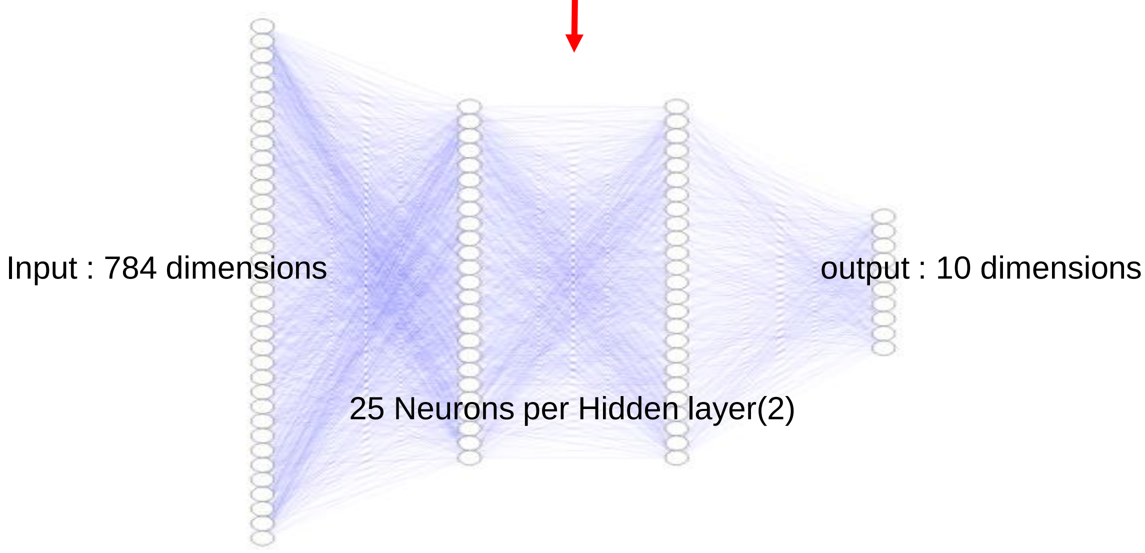
- Standardize the dataset

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Multi-layer Neural Networks – Image Classification

- Building a Multi-layer Neural Networks and Training the model

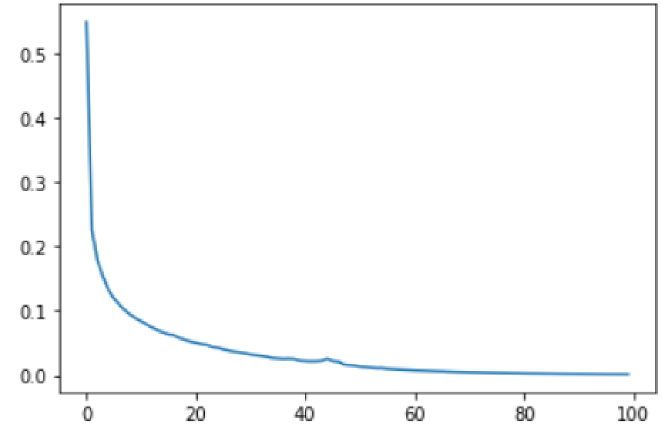
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(25,25), max_iter=100, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=0,
                    learning_rate_init=.1)

mlp.fit(X_train_std, y_train)
```



Multi-layer Neural Networks – Image Classification

- Loss curve

```
# plot the loss
plt.plot(mlp.loss_curve_)
plt.show()
```



- Evaluation

```
# Train accuracy
acc = mlp.score(X_train_std, y_train)
print("Train accuracy : %.4f" % acc)
```

```
Train accuracy : 1.0000
```

```
# Test accuracy
acc = mlp.score(X_test_std, y_test)
print("Test accuracy : %.4f" % acc)
```

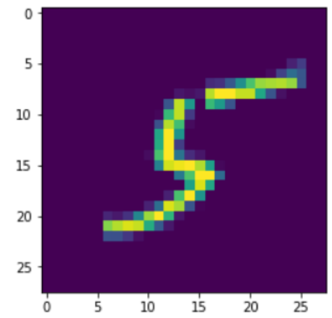
```
Test accuracy : 0.9561
```

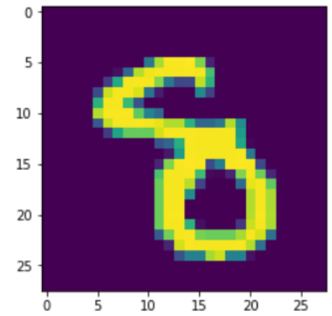
Multi-layer Neural Networks – Image Classification

- Classification example

```
%matplotlib inline
some_digit1 = X_train_std[35000]
some_digit_image1 = some_digit1.reshape(28, 28)
plt.imshow(some_digit_image1)
plt.show()

some_digit2 = X_train_std[50000]
some_digit_image2 = some_digit2.reshape(28, 28)
plt.imshow(some_digit_image2)
plt.show()
# classification
mlp.predict(sc.transform([some_digit1]))
# classification
mlp.predict(sc.transform([some_digit2]))
```

```
array([5.])
array([8.])
```





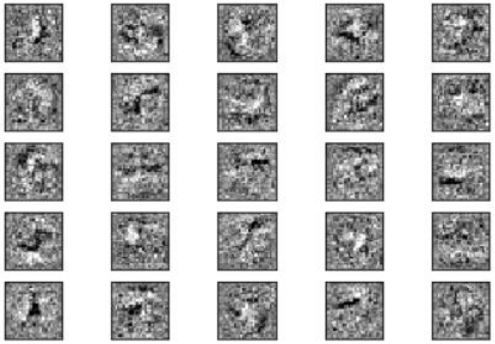
Multi-layer Neural Networks – Image Classification

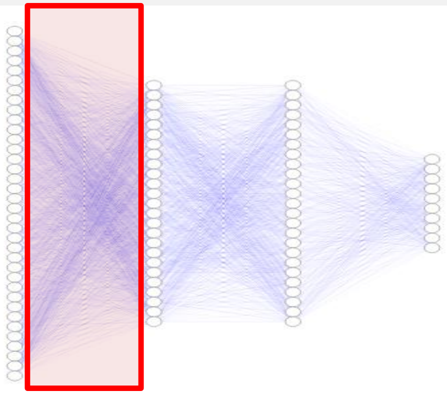
- Visualization of MLP weights on MNIST

```
mlp.coefs_[0].shape
```

```
(784, 25)
```

```
fig, axes = plt.subplots(5, 5)
# use global min / max to ensure all weights are shown on the same scale
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()

for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=.5 * vmin, vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```





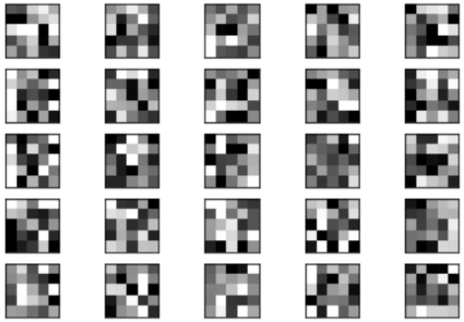
Multi-layer Neural Networks – Image Classification

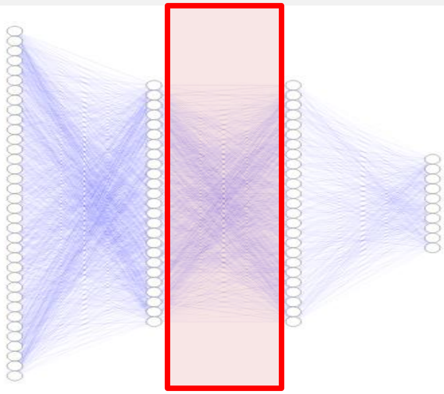
- Visualization of MLP weights on MNIST

```
mlp.coefs_[1].shape
```

```
(25, 25)
```

```
fig, axes = plt.subplots(5, 5)
# use global min / max to ensure all weights are shown on the same scale
vmin, vmax = mlp.coefs_[1].min(), mlp.coefs_[1].max()

for coef, ax in zip(mlp.coefs_[1].T, axes.ravel()):
    ax.matshow(coef.reshape(5, 5), cmap=plt.cm.gray, vmin=.5 * vmin, vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```





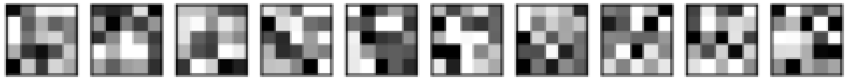
Multi-layer Neural Networks – Image Classification

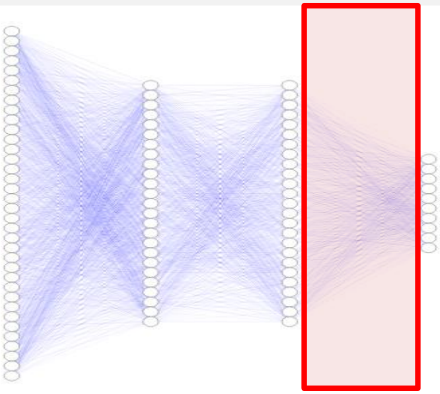
- Visualization of MLP weights on MNIST

```
mlp.coefs_[2].shape
```

```
(25, 10)
```

```
fig, axes = plt.subplots(1, 10)
# use global min / max to ensure all weights are shown on the same scale
vmin, vmax = mlp.coefs_[2].min(), mlp.coefs_[2].max()

for coef, ax in zip(mlp.coefs_[2].T, axes.ravel()):
    ax.matshow(coef.reshape(5, 5), cmap=plt.cm.gray, vmin=.5 * vmin, vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```





Submit

- To make sure if you have completed this practice,
  Submit your practice file(Week07_givencode.ipynb) to e-class.
- **Deadline : tomorrow 11:59pm**
- Modify your ipynb file name as “Week07_StudentNum_Name.ipynb”
  Ex) **Week07_2020123456_홍길동.ipynb**
- You can upload this file without taking the quiz, but homework will be provided like a quiz every three weeks, so it is recommended to take the quiz as well.

Quiz 1

- K-Nearest Neighbors
  - Dataset : Iris dataset(use all features)
  - Model : K-Nearest Neighbors
  - Find the hyperparameter that makes highest test accuracy(0.9556)
    - Number of Neighbors : 3, 5, 7, ...
    - distance metric : manhattan(p=1) / euclidean(p=2)

Quiz 2

- Multi-layer Neural Networks :
  - Dataset : MNIST dataset
  - Model : Multi-layer Neural Network
  - Find the hyperparameter that makes highest test accuracy
    - Number of hidden layers
    - Regularization ratio(alpha)