PATTERN RECOGNITION USING PYTHON

Multi-Layer Perceptron

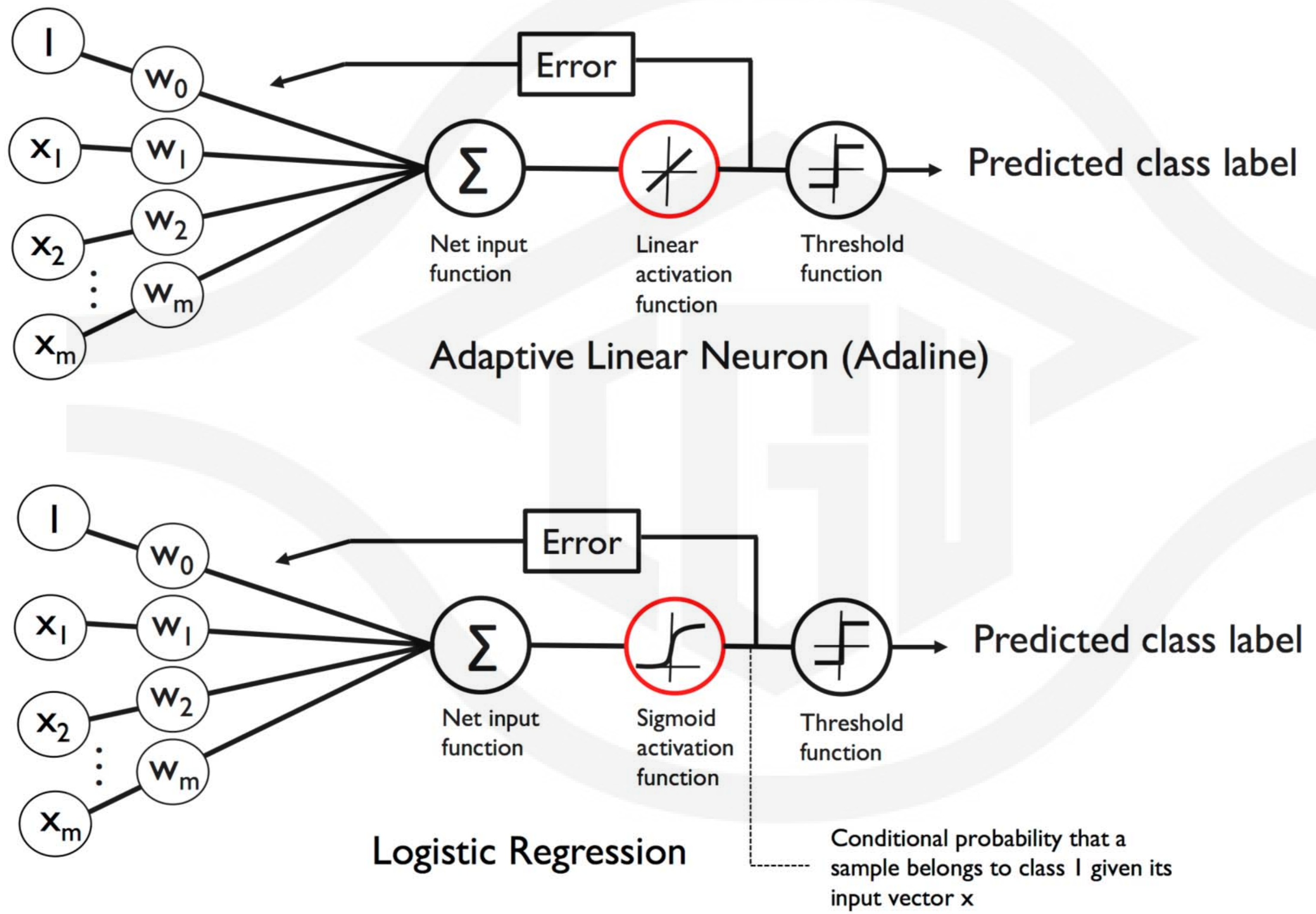
Wen-Yen Hsu
Dept Electrical Engineering
Chang Gung University, Taiwan

2019-Spring

More Powerful Architectures

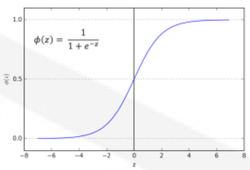
- Getting a conceptual understanding of multilayer neural networks
- Training a basic multilayer neural network for image classification

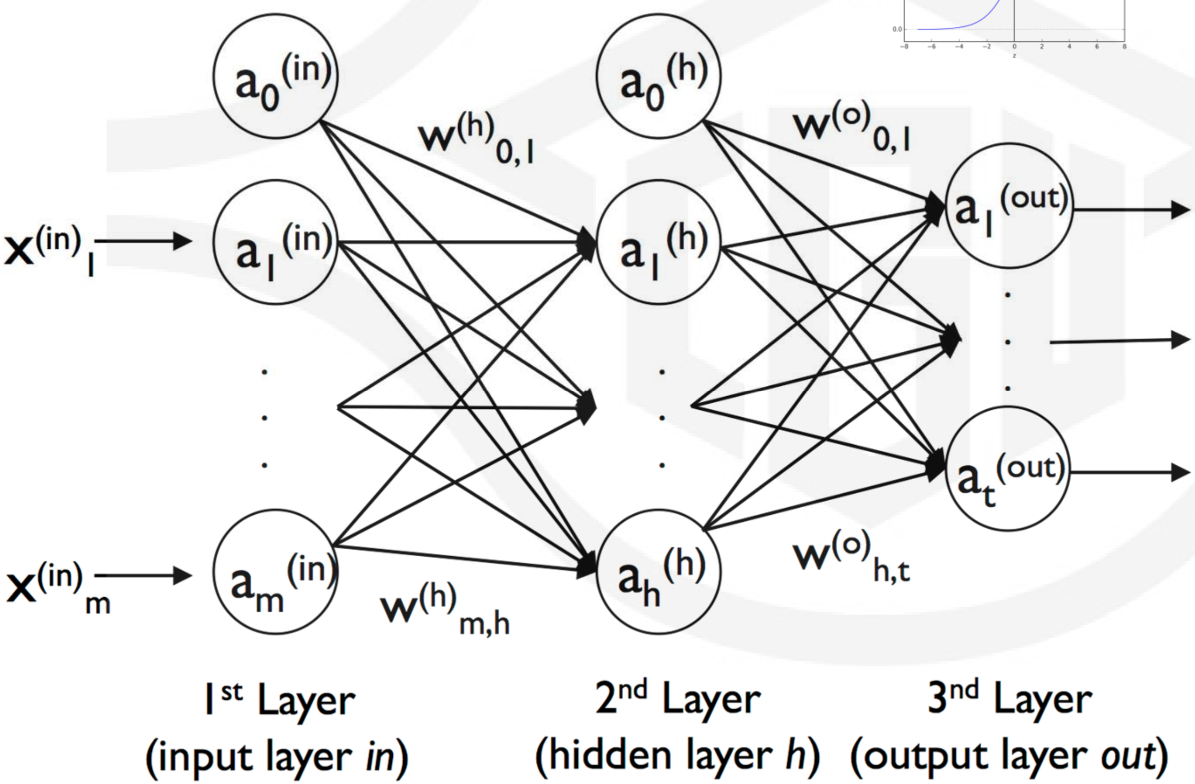
Single-layer Perceptron Recap



Multi-layer Perceptron (Fully Connected Network)

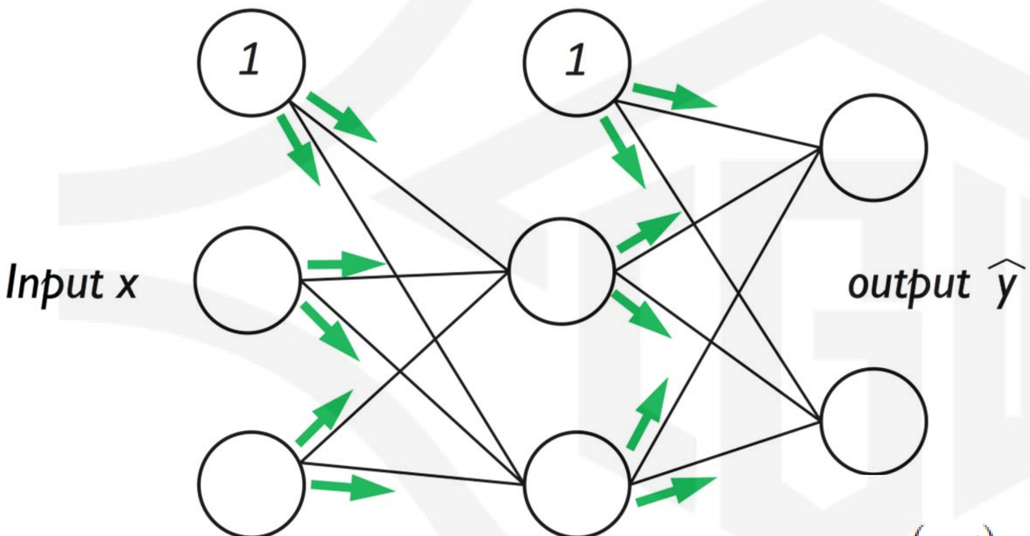
- Each layer serves as the input to the next layer without loops

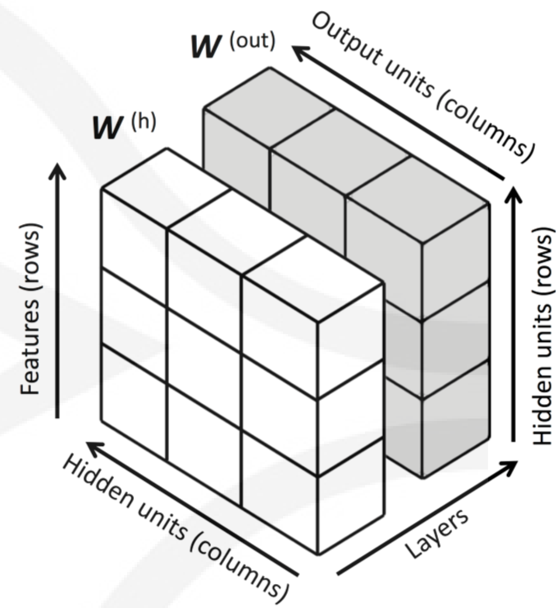




$$a^{(out)} = \begin{bmatrix} 0.1 \\ 0.9 \\ \vdots \\ 0.3 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Forward Propagation





$$\boldsymbol{A}^{(out)} = \phi\left(\boldsymbol{Z}^{(out)}\right)$$

$$\boldsymbol{Z}^{(h)} = \boldsymbol{A}^{(in)}\boldsymbol{W}^{(h)}$$

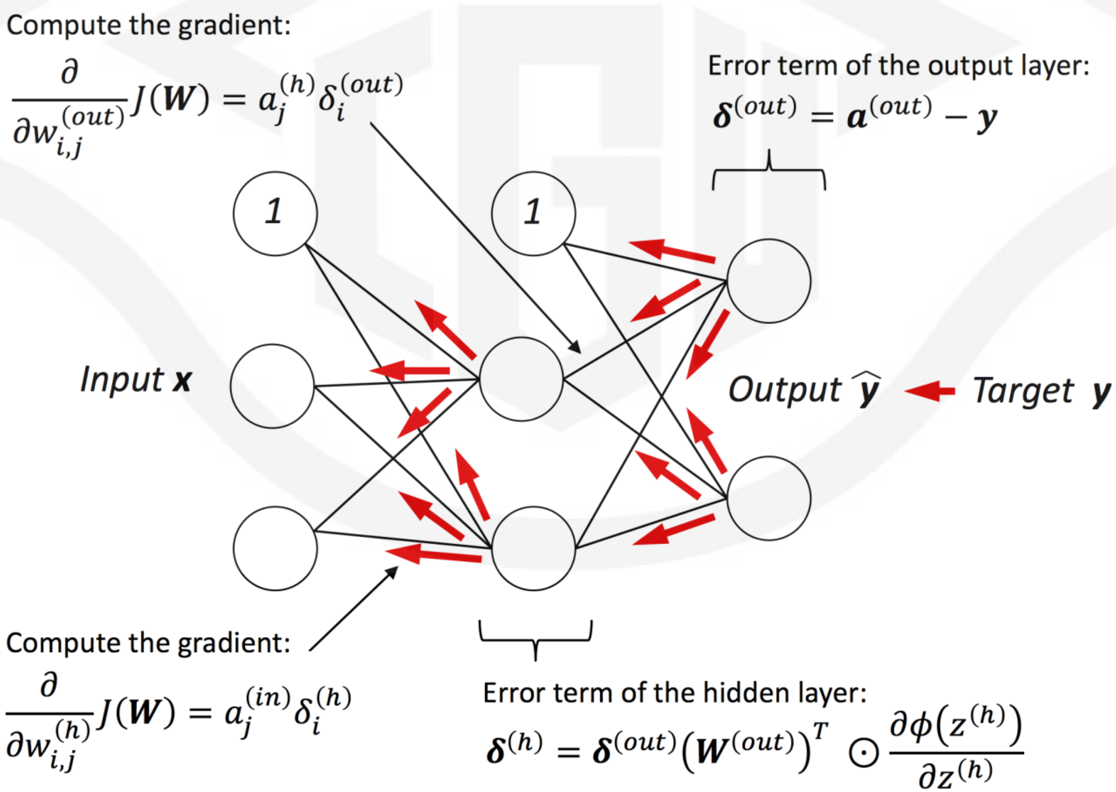
$$A^{(h)} = \phi\left(Z^{(h)}\right)$$

$$\boldsymbol{Z}^{(out)} = \boldsymbol{A}^{(h)}\boldsymbol{W}^{(out)}$$

Training Neural Networks via Backpropagation

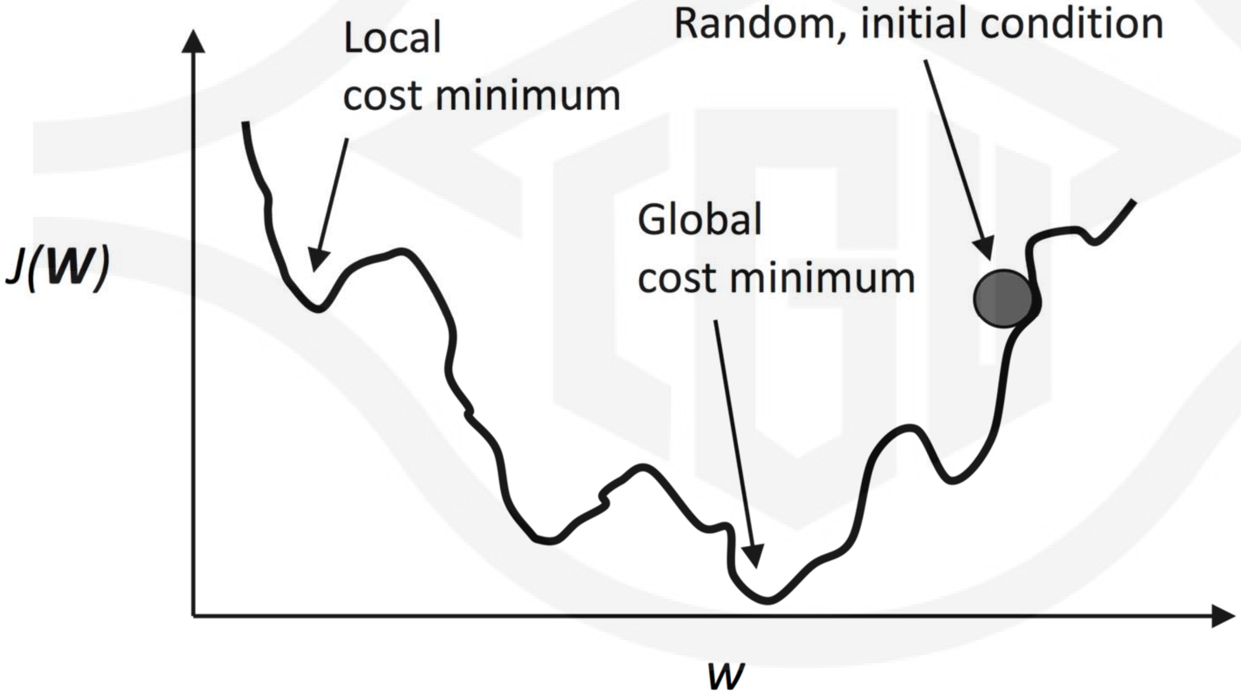
- Calculate the partial derivative of the parameters **W with respect to each weight** for every layer in the network

$$\frac{\partial}{\partial w_{j,i}^{(l)}} J(W) \qquad J(W) = -\sum_{i=1}^{n}\sum_{j=1}^{t} y_j^{[i]} log\left(a^{[i]}{}_j\right) + \left(1 - y_j^{[i]}\right) log\left(1 - a^{[i]}{}_j\right)$$



Convergence in Neural Networks

- Non-convex loss function



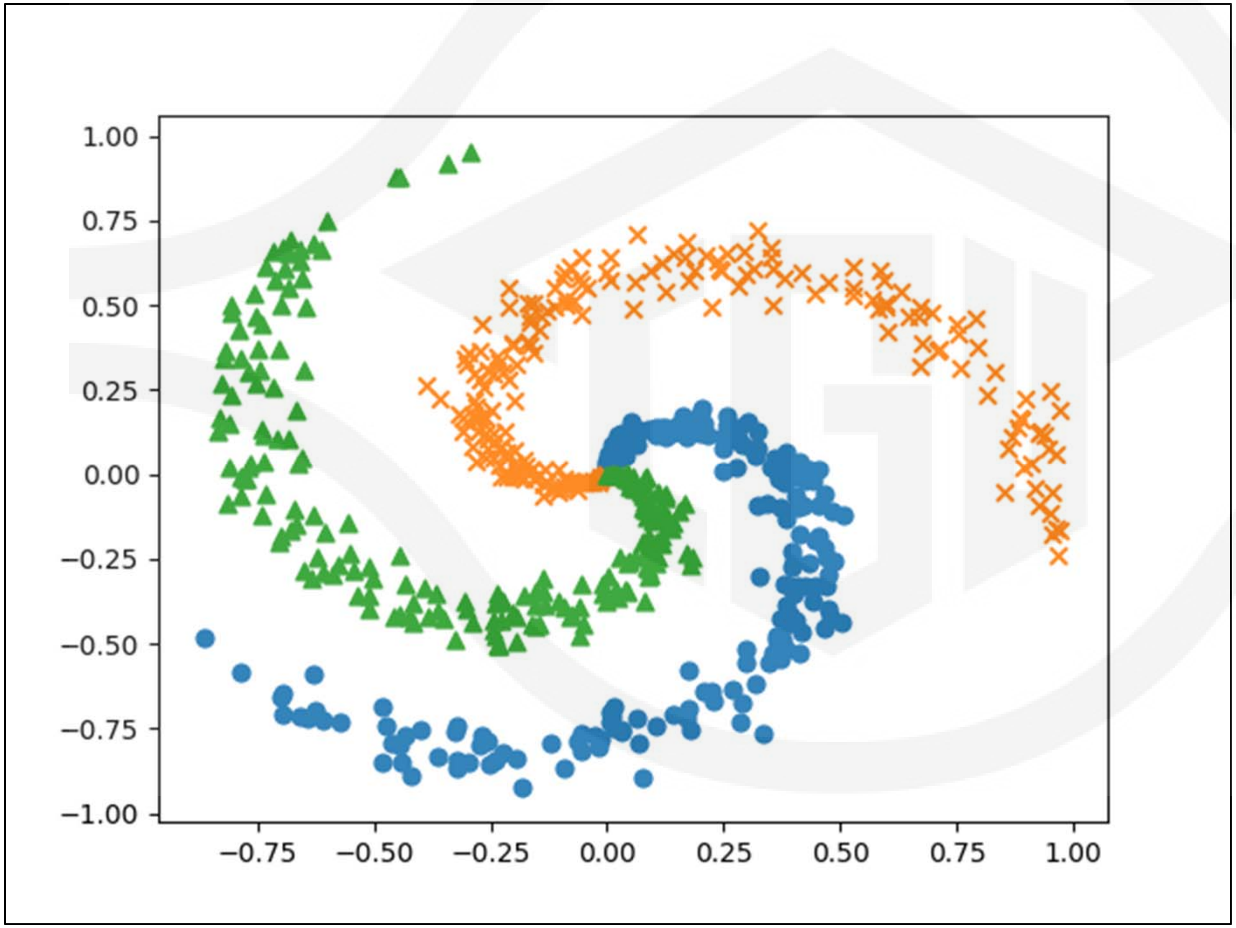
MLP Classifier utilize sklearn

- Classify wine data with MLP

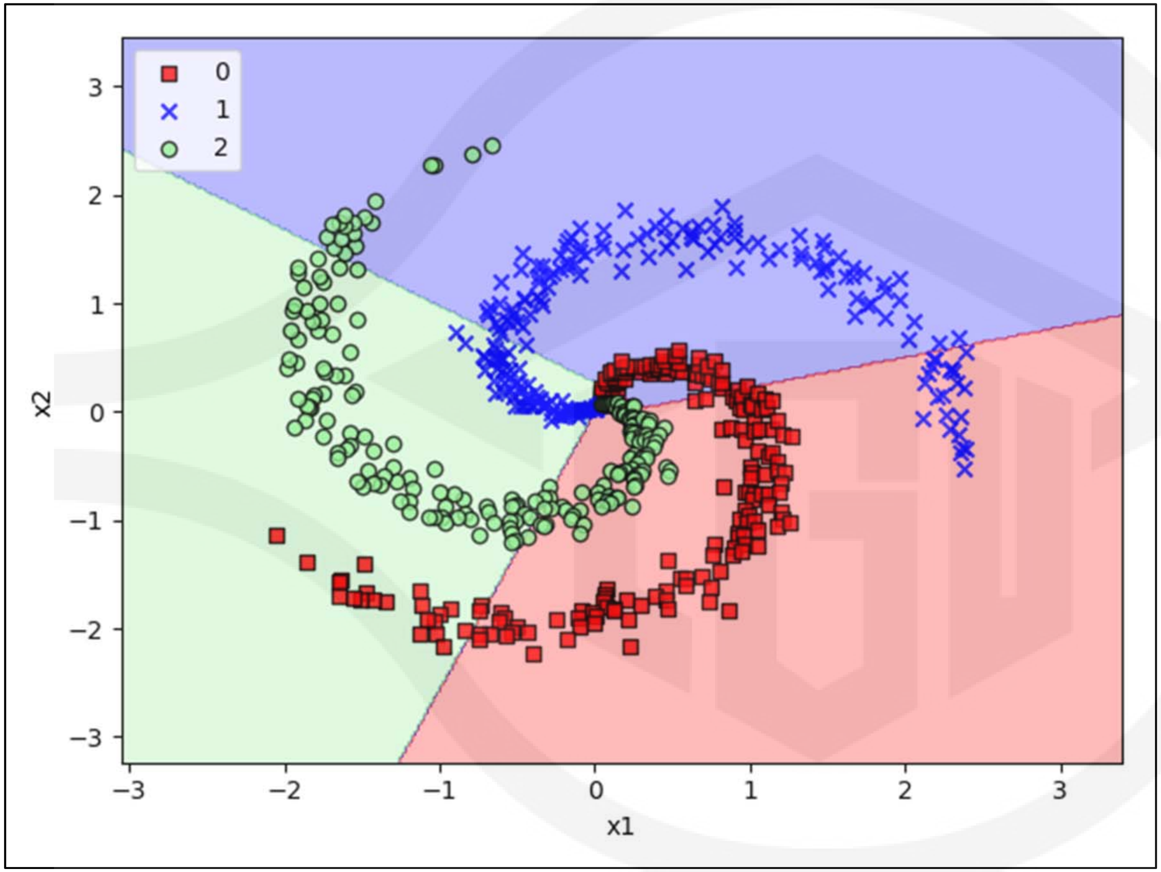
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=500,
alpha=1e-4, solver='sgd', verbose=False, tol=1e-4,
random_state=1, learning_rate_init=0.1)
mlp.fit(X_train_std, y_train)
print("Training set score: %f" % mlp.score(mlp.fit(X_train_std,
y_train))
print("Test set score: %f" % mlp.score(mlp.fit(X_test_std,
y_test))
```

Non-linear Data Classification

- Spiral Pattern

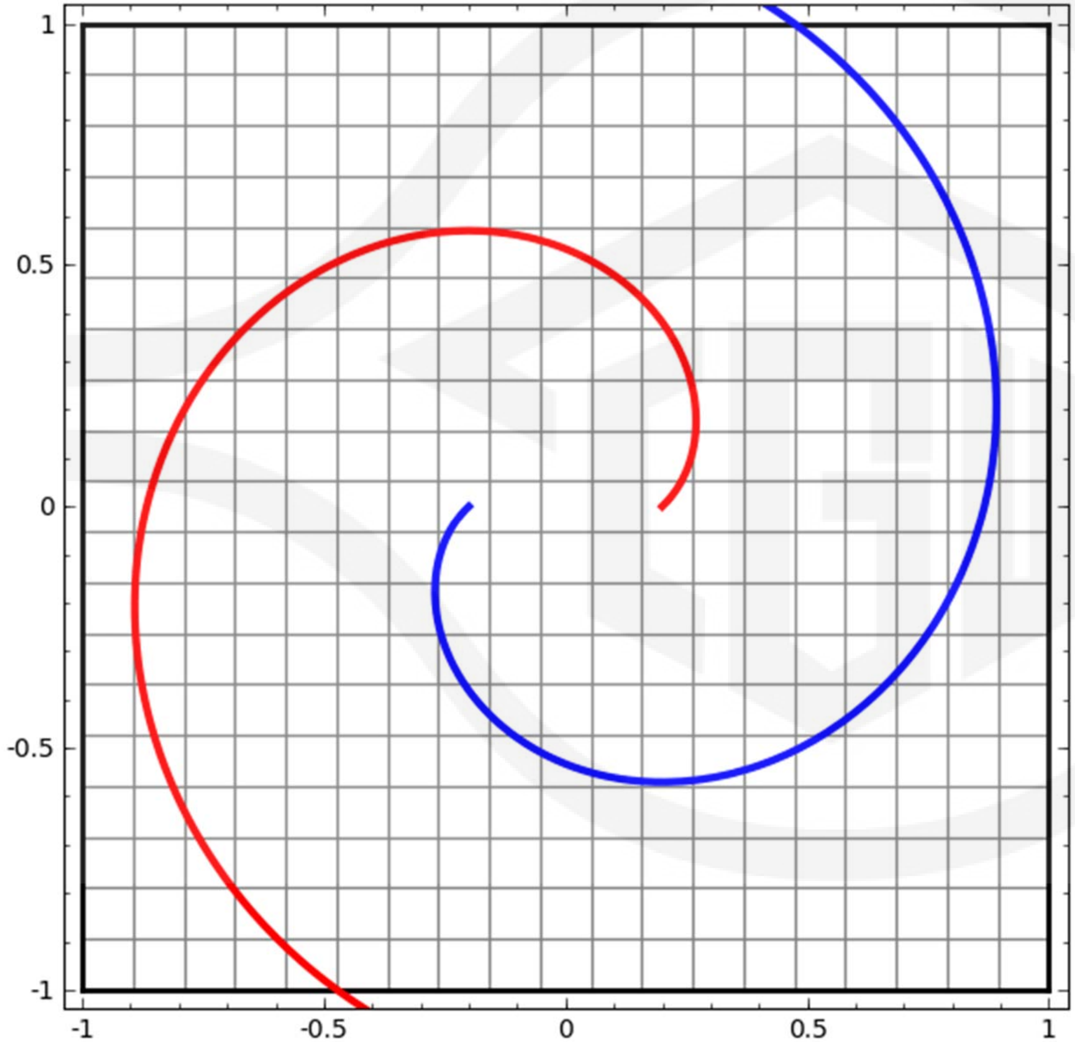


General Classifier



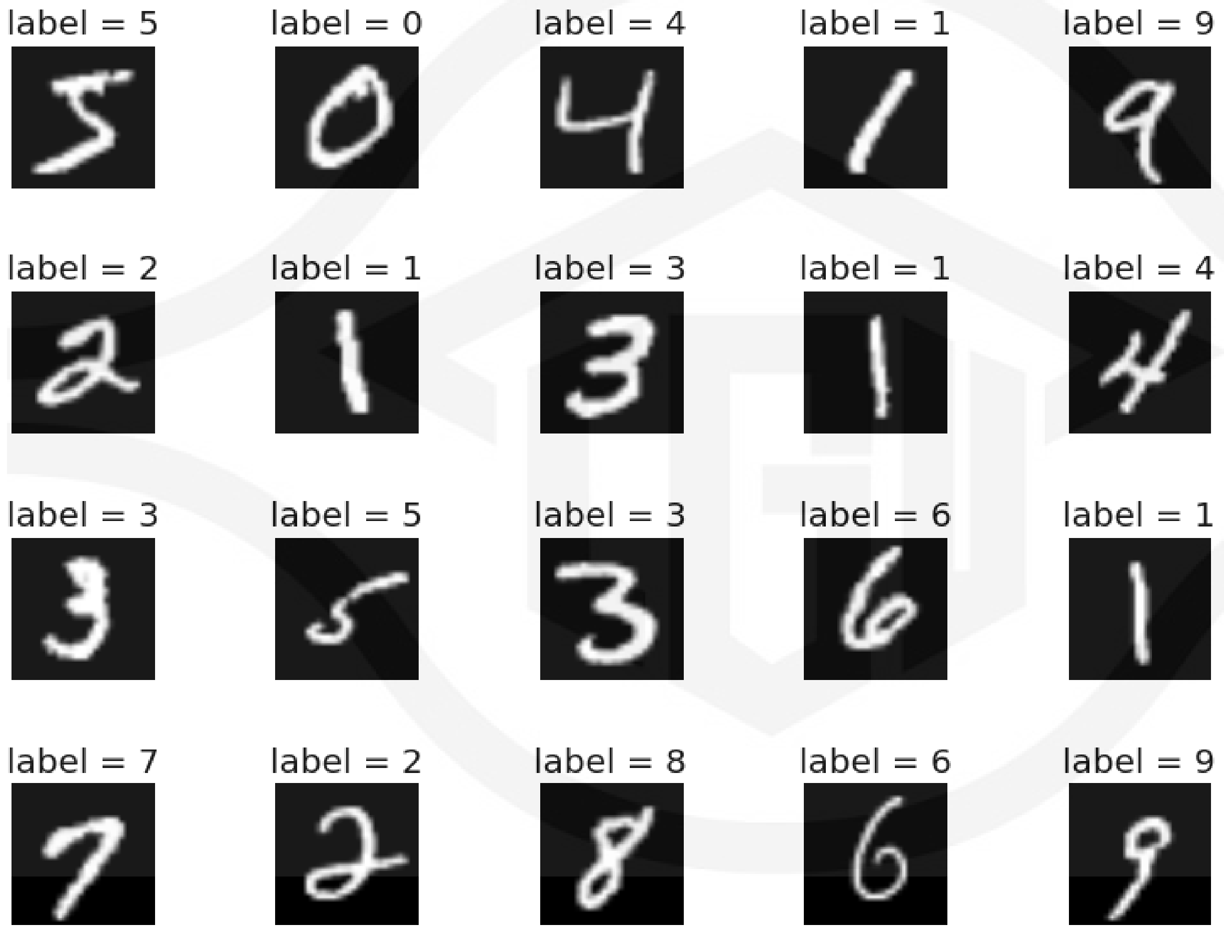
- Not Good, Try MLP Classifier ?

The Ability of Non-linear Mapping



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Classifying Handwritten Digits



http://yann.lecun.com/exdb/lenet/

Load MNIST Data & Preprocessing

- Dim = 784 (H28*W28)

```
## load dataset
df = pd.read_csv('mnist_784.csv', header=0)
y = df.iloc[:, -1].values
print(y.shape)
X = df.iloc[:, 0:-1].values
print(X.shape)
```

- 8 bits gray scale (0~255)

```
X = X / 255.
```

Hidden Layer Setting

- Hidden layer (100, 100)

```
## classifier MLP two hidden layer
from sklearn.neural_network import MLPClassifier
mlp2 = MLPClassifier(hidden_layer_sizes=(100, 100),
max_iter=50, alpha=1e-4, solver='sgd', verbose= False,
tol=1e-4, random_state=1, learning_rate_init=0.1)
mlp2.fit(X_train, y_train)
print('Two hidden layer')
print("Training set score: %f" % mlp2.score(X_train,
y_train))
print("Test set score: %f" % mlp2.score(X_test, y_test))
```

Reference

- Sebastian Raschka, Vahid Mirjalili. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow. Second Edition. Packt Publishing, 2017.