다층 신경망의 활성화 함수 선택

- (머신러닝 교과서, 세바스찬 라시카, p. 560) 13.5 다층 신경망의 활성화 함수 선택

1. 로지스틱 함수 요약

$$z = w_o x_o + w_1 x_1 + \cdots + w_m x_m = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

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

실습 코드 1

- 데이터 *x*와 다음 코드과 같은 가중치 벡터 *w*로 구성된 모델을 가정
- 이 특성과 가중치 값을 사용하여 최종 입력(*z*)를 계산하고 이것으로 로지스틱 뉴런의 활성화 출력을 구하면 0.888을 얻음.
- 이를 샘플 *x*가 양성(+1) 클래스에 속할 확률이 88.8%라고 해석학 수 있음

In [1]:
```
import numpy as np

X = np.array([1, 1.4, 2.5]) ## first value must be 1
w = np.array([0.4, 0.3, 0.5])

def net_input(X, w):
    return np.dot(X, w)

def logistic(z):
    return 1.0 / (1.0 + np.exp(-z))

def logistic_activation(X, w):
    z = net_input(X, w)
    return logistic(z)

print('P(y=1|x) = %.3f' % logistic_activation(X, w))
```

```
P(y=1|x) = 0.888
```

실습 코드 2

- 다중 클래스
- 로지스틱 활성화 유닛으로 출력층 구성
- 하지만, 다음 코드의 출력은 의미 있게 해석할 만한 확률 값을 만들지 못함

In [2]:
```
# W : (n_output_units, n_hidden_units+1) 크기의 배열
# 첫 번째 열은 절편 유닛입니다

W = np.array([[1.1, 1.2, 0.8, 0.4],
              [0.2, 0.4, 1.0, 0.2],
              [0.6, 1.5, 1.2, 0.7]])

# A : (n_hidden_units + 1, n_samples) 크기의 배열
# 이 배열의 첫 번째 열은 1입니다

A = np.array([[1, 0.1, 0.4, 0.6]])
Z = np.dot(W, A[0])

y_probas = logistic(Z)
print('최종 입력: ', Z)

print('유닛 출력: ', y_probas)
```

```
최종 입력:  [1.78 0.76 1.65]
유닛 출력:  [0.85569687 0.68135373 0.83889105]
```

실습 코드 3

- 클래스의 레이블을 예측하기 위해 가장 큰 값을 선택

In [3]:
```
y_class = np.argmax(Z, axis=0)
print('예측 클래스 레이블: %d' % y_class)
```

```
예측 클래스 레이블: 0
```

2. 소프트맥스 함수를 사용한 다중 클래스 확률 예측

$$p(z) = \phi(z) = \frac{e^{z_i}}{\sum_{j=1}^{M} e^{z_j}}$$

실습 코드 4

- softmax 함수의 동작 확인

In [4]:
```
def softmax(z):
    return np.exp(z) / np.sum(np.exp(z))

y_probas = softmax(Z)
print('확률: ', y_probas)

print('합계: ', np.sum(y_probas))
```

```
확률:  [0.44668973 0.16107406 0.39223621]
합계:  1.0
```

3. Iris classification Task에 활성화 함수 적용 예제

In [5]:
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import OneHotEncoder
# 라이브러리 import 후 iris.csv read (dataset link 사용)
import os
import pandas as pd

s = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
```

In [6]:
```
df = pd.read_csv(s, header=None, encoding='utf-8')

ohe = OneHotEncoder(sparse_output=False)
y  = ohe.fit_transform(df.iloc[:][[4]])

d = {'Iris-setosa':0,  'Iris-versicolor':1, 'Iris-virginica':2}
y_ori = df.iloc[:, 4].map(d)

# 꽃받침 길이와 꽃잎 길이를 특징 값으로 추출
X = df.iloc[:,[0, 2]].values
```

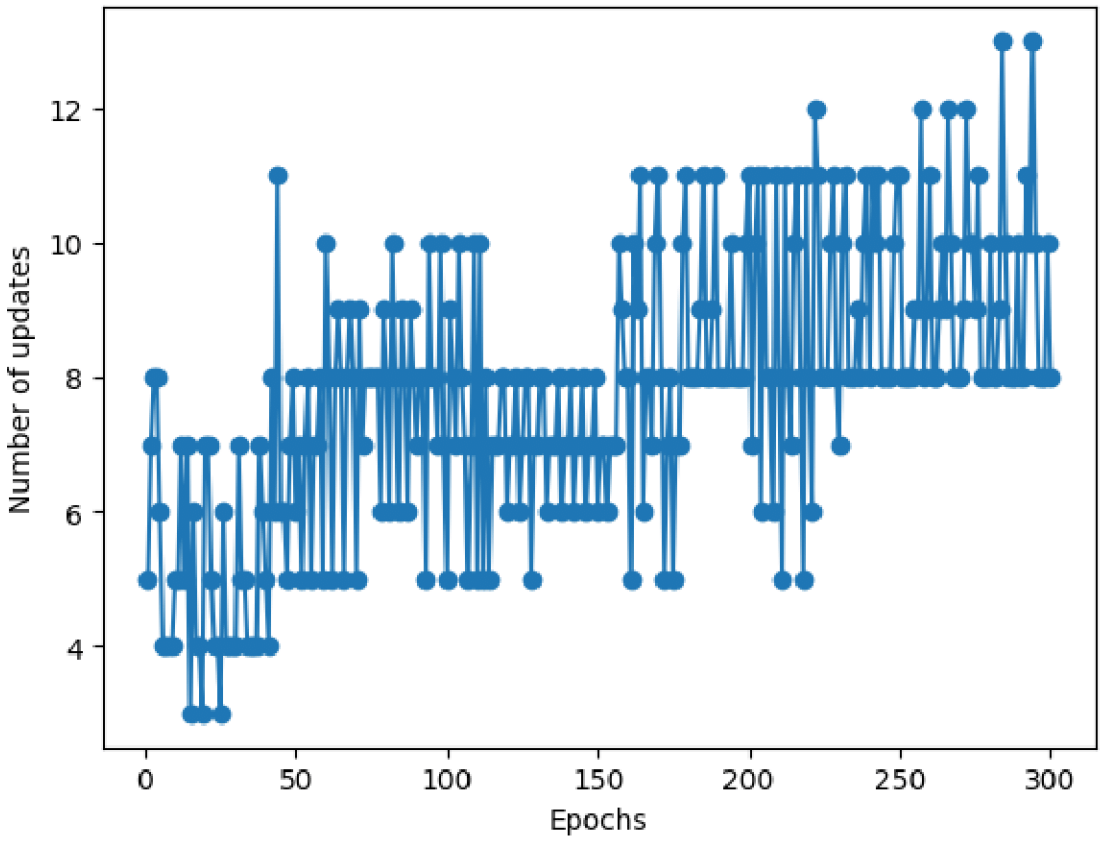
퍼셉트론 모델 구현

In [25]:
```
import numpy as np


class Perceptron(object):
    """퍼셉트론 분류기 구현"""
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        ''' 초기화 함수 구현 '''
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        """ 훈련 데이터 학습 함수 구현 """
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size= (y.shape[1], X.shape[1] + 1))
        self.errors_ = []
        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[:, 1:] += (np.transpose([update]) * xi)
                self.w_[:, 0] += update
                errors += int(sum(update != 0.0))
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        """입력 계산 함수 구현"""
        return np.dot(X, self.w_[:, 1:].T) + self.w_[:, 0]

    def predict(self, X, activation = 'Softmax_binary'):
        """ 예측 함수 구현"""
        if activation == 'Softmax_binary' :
          return np.where(softmax(self.net_input(X))< 0.5, 0, 1)
        elif activation == 'Softmax':
          return softmax(self.net_input(X))
```

```
        elif activation == 'Logistic':
          return logistic(self.net_input(X))

    def predict_argmax(self, X):
        return np.argmax(softmax(self.net_input(X)), axis=1)
```

퍼셉트론 모델 훈련하기

In [26]:
```
# Perceptron 선언 및 학습
ppn = Perceptron(eta=0.03, n_iter=300)
ppn.fit(X, y)

# Epochs당 Number of updates그래프 출력
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')

plt.show()
```



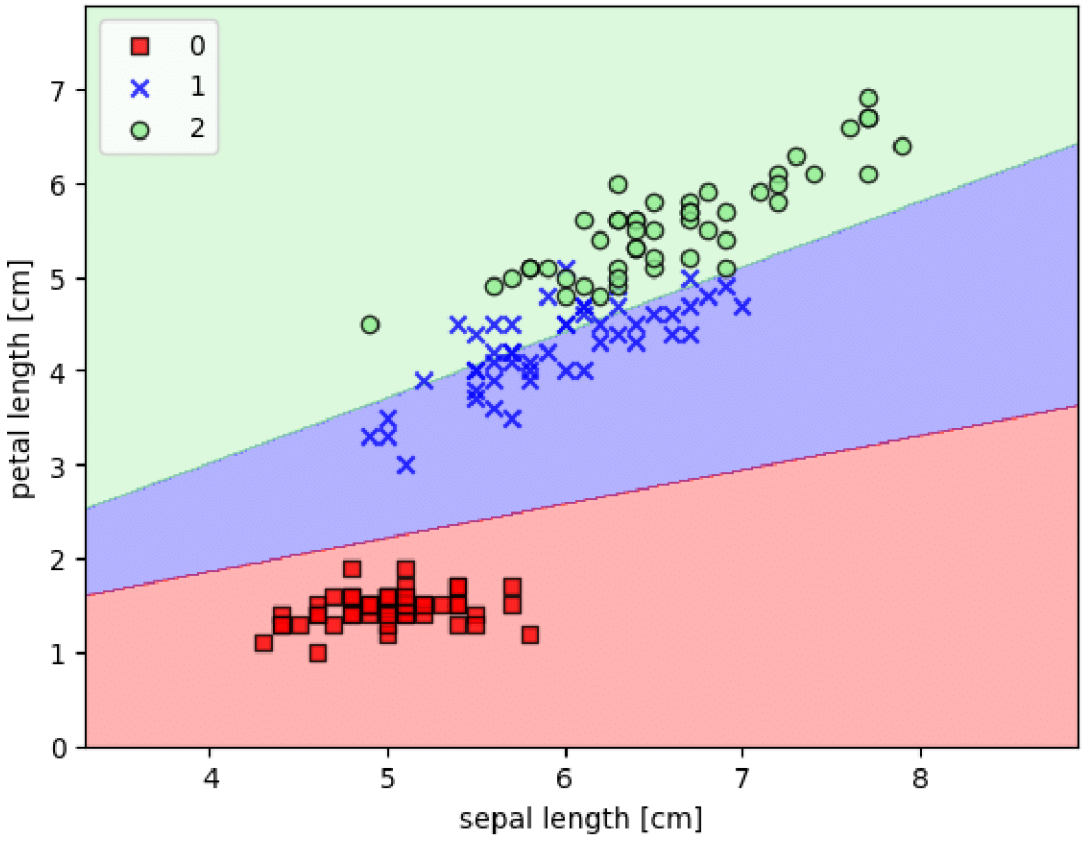
결정 경계 그래프 함수

In [21]:
```
from matplotlib.colors import ListedColormap

# 결정 경계 그래프 함수 정의
def plot_decision_regions(X, y, classifier, resolution=0.01):

    # 마커와 컬러맵을 설정합니다
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # 결정 경계를 그립니다
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1  # 꽃받침 길이 최소/최대
```

```
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1  # 꽃잎 길이 최소/최대
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict_argmax(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # 샘플의 산점도를 그립니다
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor=None if idx==1 else 'black')
```

In [22]:

```
# 결정 경계 그래프 출력
plot_decision_regions(X, y_ori, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')

plt.show()
```



In [11]:

```
# Logistic Activation Function

pred = ppn.predict([1.4, 2.3], activation = 'Logistic')
print(pred)
print("예측 클래스 레이블: ", np.argmax(pred))
```

```
[1. 1. 1.]
예측 클래스 레이블:  2
```

In [12]:
```
# Softmax Activation Function

pred = ppn.predict([1.4, 2.3], activation = 'Softmax')
print('확률: ', pred)
print('합계: ', np.sum(pred))
print("예측 클래스 레이블: ", np.argmax(pred))
```

```
확률:  [2.42055670e-05 2.05470478e-04 9.99770324e-01]
합계:  1.0
예측 클래스 레이블:  2
```