実装演習：1-4.勾配降下法

In [13]:
```
import numpy as np
from common import functions
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

In [14]:
```
def print_vec(text, vec):
    print(f'{text} :\n{vec}')
```

確率勾配降下法

In [15]:
```
# サンプルとする関数：yの値を予想する
def f(x):
    """y=3x_0 + 2x_1"""
    y = 3 * x[0] + 2 * x[1]
    return y
```

In [16]:
```
def init_network():
    """ネットワークの初期化"""
    print("##### ネットワークの初期化 #####")
    network = {}
    nodesNum = 10
    network['W1'] = np.random.randn(2, nodesNum)
    network['W2'] = np.random.randn(nodesNum)
    network['b1'] = np.random.randn(nodesNum)
    network['b2'] = np.random.randn()

    print_vec("重み1", network['W1'])
    print_vec("重み2", network['W2'])
    print_vec("バイアス1", network['b1'])
    print_vec("バイアス2", network['b2'])

    return network
```

In [17]:
```
def forward(network, x):
    """順伝播"""
    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    u1 = np.dot(x, W1) + b1
    z1 = functions.relu(u1)

    u2 = np.dot(z1, W2) + b2
    y = u2

    return z1, y
```

In [18]:
```python
# ※勾配降下法の演習のため、誤差逆伝搬法についての深掘りは行わない。
def backward(x, d, z1, y):
    """誤差逆伝播"""
    grad = {}

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    # 出力層でのデルタ
    delta2 = functions.d_mean_squared_error(d, y)
    # b2の勾配
    grad['b2'] = np.sum(delta2, axis=0)
    # W2の勾配
    grad['W2'] = np.dot(z1.T, delta2)

    # 中間層でのデルタ
    delta1 = np.dot(delta2, W2.T) * functions.d_sigmoid(z1)
    delta1 = delta1[np.newaxis, :]
    # b1の勾配
    grad['b1'] = np.sum(delta1, axis=0)
    x = x[np.newaxis, :]
    # W1の勾配
    grad['W1'] = np.dot(x.T, delta1)

    return grad
```

In [19]:
```python
# サンプルデータを作成
data_sets_size = 100000
data_sets = [0 for i in range(data_sets_size)]
```

In [20]:
```python
for i in range(data_sets_size):
    data_sets[i] = {}
    # ランダムな値を設定
    data_sets[i]['x'] = np.random.rand(2)

    # 目標出力を設定
    data_sets[i]['d'] = f(data_sets[i]['x'])
```

In [21]:
```python
losses = []  # 損失
learning_rate = 0.07  # 学習率
epoch = 1000  # エポック数
```

In [22]:
```python
# パラメータの初期化
network = init_network()
```

```
##### ネットワークの初期化 #####
重み1：
[[ 0.03179469 -0.65580151 -1.05014931 -0.95991165 -1.71442249  2.49697393
  -1.12710321  0.07263078 -1.28432835 -0.63352764]
 [ 1.33957779  0.09876068  1.15336085 -1.20192046 -1.27906948 -0.31698447
  -0.1980017  -0.45523892 -0.7397935   0.33795826]]
重み2：
[-0.24279993  0.20530293  0.43517228  1.41684007 -0.78456474 -1.60324141
 -0.41477104  2.23685999 -1.07724056 -0.22435489]
バイアス1：
[ 0.00803233  0.7687969   0.60887701 -0.61285442 -0.53212624  0.3379684
 -0.45232803  0.39699546 -2.24865943 -0.59136552]
バイアス2：
0.23229489417041302
```

In [23]:
```python
# データのランダム抽出
random_datasets = np.random.choice(data_sets, epoch)
```

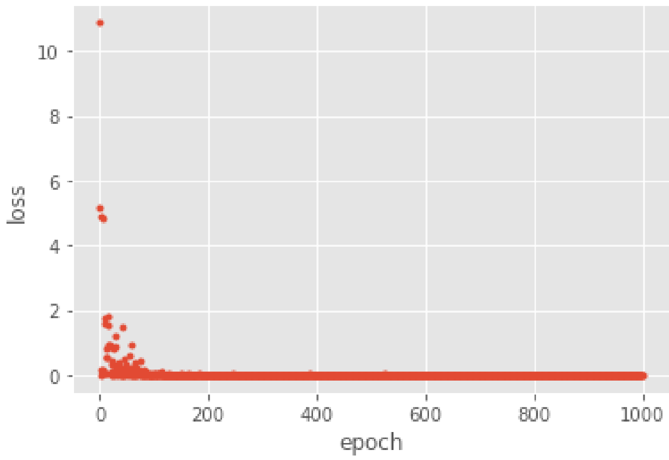
In [24]:
```python
# 勾配降下の繰り返し
for dataset in random_datasets:
    x, d = dataset['x'], dataset['d']
    z1, y = forward(network, x)  # 順伝搬
    grad = backward(x, d, z1, y)  # 逆伝搬

    # パラメータに勾配適用
    for key in ('W1', 'W2', 'b1', 'b2'):
        network[key]  -= learning_rate * grad[key]

    # 誤差関数(MSE：平均二乗誤差)
    loss = functions.mean_squared_error(d, y)
    losses.append(loss)
```

In [32]:
```python
x = range(epoch)

# グラフの表示
plt.plot(x, losses, '.')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



In [ ]:
```python
```