rp_ap5

June 20, 2022

Rede CNN implementada com backpropagation

José Geraldo Fernandes

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

Dados

[2]:
```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
X = x_train[0]

m, n = X.shape
```

1 Configuração

Parâmetros da Rede

[3]:
```
p = 6
q = 12
filter_shape = [5, 5]
mlp = [10, 192]
```

Inicialização

[4]:
```
k1 = np.random.uniform(low = 1, high = 6, size = filter_shape + [p])
b1 = np.zeros(shape = (1, p))

k2 = np.random.uniform(low = 6, high = 12, size = filter_shape + [p, q])
b2 = np.zeros(shape = (1, q))

W = np.random.uniform(low = -1, high = 1, size = mlp)
b = np.zeros(shape = (mlp[0], 1))
```

Funções de ativação, convolução e pooling

[5]:
```
def sigma(x):
    return 1 / (1 + np.exp(-x))
```

```
def conv(img, kernel, bias):
    x, y = img.shape
    m, n = kernel.shape
    x = x - m + 1
    y = y - m + 1

    out = np.zeros(shape = (x, y))
    for i in range(x):
        for j in range(y):
            h = np.sum(img[i:i+m, j:j+m]*kernel)
            out[i, j] = sigma(h + bias)

    return out

def pooling(C):
    W, L, samples = C.shape
    w, l = [int(W/2), int(L/2)]
    S = np.zeros(shape = (w, l, samples))
    for k in range(samples):
        for i in range(w):
            for j in range(l):
                S[i, j, k] += C[2*i, 2*j, k]
                S[i, j, k] += C[2*i, 2*j - 1, k]
                S[i, j, k] += C[2*i - 1, 2*j, k]
                S[i, j, k] += C[2*i - 1, 2*j - 1, k]
                S[i, j, k] /= 4
    return S
```

2 Forward

2.1 Primeira Camada

[6]:
```
C1 = np.zeros(shape = (m - filter_shape[0] + 1, n - filter_shape[0] + 1, p))
for i in range(b1.size):
    C1[:, :, i] = conv(X, k1[:, :, i], b1[:, i])

S1 = pooling(C1)
```

2.2 Segunda Camada

[7]:
```
C2 = np.zeros(shape = (S1.shape[0] - filter_shape[0] + 1, S1.shape[1] -␣
 ↪filter_shape[0] + 1, q))
for i in range(p):
    for j in range(q):
        C2[:, :, j] = conv(S1[:, :, i], k2[:, :, i, j], b2[:, j]) + C2[:, :, j]
```

```
S2 = pooling(C2)
```

2.3 Camada Final

[8]:
```
f = S2.reshape(-1, 1)

H = np.dot(W, f) + b
yhat = sigma(H)
```

2.4 Custo

[9]:
```
y = np.zeros(shape = (10, 1))
y[y_train[0] - 1] = 1

L = np.sum((y - yhat)**2) / 2
```

3 Backpropagation

3.1 Camada Final

[10]:
```
d_yhat = (yhat - y) * yhat * (1 - y)

d_W = np.dot(d_yhat, f.T)
d_b = d_yhat
d_f = np.dot(W.T, d_yhat)
```

3.2 Segunda Camada

[11]:
```
d_S2 = d_f.reshape(S2.shape)
d_C2 = np.zeros(shape = C2.shape)
for k in range(q):
    for i in range(C2.shape[0]):
        for j in range(C2.shape[1]):
            ii = int(np.ceil(i / 2) - 1)
            jj = int(np.ceil(j / 2) - 1)
            d_C2[i, j, k] = d_S2[ii, jj, k] / 4

d_sigma2 = d_C2 * C2 * (1 - C2)
d_k2 = np.zeros(shape = k2.shape)
for i in range(p):
    for j in range(q):
        d_k2[:, :, i, j] += conv(np.rot90(S1, 2)[:, :, i], d_sigma2[:, :, j], 0)

d_b2 = np.zeros(b2.shape)
for i in range(q):
    d_b2[:, i] = np.sum(d_sigma2[:, :, i])
```

3.3 Primeira Camada

[12]:
```
d_S1 = np.zeros(shape = S1.shape)
for i in range(p):
    for j in range(q):
        pad_size = d_S1.shape[0] - d_sigma2.shape[0]
        pad = np.pad(d_sigma2[:, :, j], pad_size)
        d_S1[:, :, i] += conv(pad, np.rot90(k2, 2)[:, :, i, j], 0)

d_C1 = np.zeros(C1.shape)
for k in range(p):
    for i in range(C1.shape[0]):
        for j in range(C1.shape[1]):
            ii = int(np.ceil(i / 2) - 1)
            jj = int(np.ceil(j / 2) - 1)
            d_C1 = d_S1[ii, jj, k] / 4

d_sigma1 = d_C1 * C1 * (1 - C1)
d_k1 = np.zeros(k1.shape)
for i in range(p):
    d_k1[:, :, i] += conv(np.rot90(X, 2), d_sigma1[:, :, i], 0)

d_b1 = np.zeros(b1.shape)
for i in range(p):
    d_b1[:, i] = np.sum(d_sigma1[:, :, i])
```

4 Atualização de Pesos

[13]:
```
alpha = 0.1
k1 -= alpha * d_k1
b1 -= alpha * d_b1

k2 -= alpha * d_k2
b2 -= alpha * d_b2

W -= alpha * d_W
b -= alpha * d_b
```