Neuronales Netz zur Bilderkennung mithilfe von MNIST Datensatz

Das neuronale Netz hat eine einfache zweischichtige Architektur (Gradientenverfahren). Die Eingabeschicht (Input Layer) $a^{[0]}$ besteht aus 784 Einheiten, die den 784 Pixeln in jedem 28x28-Eingabebild entsprechen. Die verborgene Schicht (Hidden Layer) $a^{[1]}$ besteht aus 10 Einheiten mit ReLU-Aktivierung, und die Ausgabeschicht (Output Layer) $a^{[2]}$ besteht aus 10 Einheiten, die den zehn Ziffernklassen mit Softmax-Aktivierung entsprechen.

Mathematik / Formeln

Forward Propagation

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g_{\text{ReLU}}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g_{\text{softmax}}(Z^{[2]})
\end{aligned}
$$

Backward Propagation

$$
\begin{aligned}
dZ^{[2]} &= A^{[2]} - Y \\
dW^{[2]} &= \frac{1}{m} dZ^{[2]} A^{[1]T} \\
dB^{[2]} &= \frac{1}{m} \sum dZ^{[2]} \\
dZ^{[1]} &= W^{[2]T} dZ^{[2]} \cdot g^{[1]\prime}(z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[0]T} \\
dB^{[1]} &= \frac{1}{m} \sum dZ^{[1]}
\end{aligned}
$$

Variablen

Forward Propagation:

- $A^{[0]} = X$: 784 x m
- $Z^{[1]} \sim A^{[1]}$: 10 x m
- $W^{[1]}$: 10 x 784 (as $W^{[1]}A^{[0]} \sim Z^{[1]}$)
- $B^{[1]}$: 10 x 1
- $Z^{[2]} \sim A^{[2]}$: 10 x m
- $W^{[2]}$: 10 x 10 (as $W^{[2]}A^{[1]} \sim Z^{[2]}$)
- $B^{[2]}$: 10 x 1

Backward Propagation:

- $dZ^{[2]}$: 10 x m ($\sim A^{[2]}$)
- $dW^{[2]}$: 10 x 10
- $dB^{[2]}$: 10 x 1
- $dZ^{[1]}$: 10 x m ($\sim A^{[1]}$)
- $dW^{[1]}$: 10 x 10
- $dB^{[1]}$: 10 x 1

Implementierung in Python

```
def init_params():
    W1 = np.random.rand(10, 784) - 0.5
    b1 = np.random.rand(10, 1) - 0.5
    W2 = np.random.rand(10, 10) - 0.5
    b2 = np.random.rand(10, 1) - 0.5
    return W1, b1, W2, b2

def ReLU(Z):
    return np.maximum(Z, 0)

def softmax(Z):
    A = np.exp(Z) / sum(np.exp(Z))
    return A

def forward_prop(W1, b1, W2, b2, X):
    Z1 = W1.dot(X) + b1
    A1 = ReLU(Z1)
    Z2 = W2.dot(A1) + b2
    A2 = softmax(Z2)
    return Z1, A1, Z2, A2

def ReLU_deriv(Z):
    return Z > 0

def one_hot(Y):
    one_hot_Y = np.zeros((Y.size, Y.max() + 1))
    one_hot_Y[np.arange(Y.size), Y] = 1
    one_hot_Y = one_hot_Y.T
    return one_hot_Y

def backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y):
    one_hot_Y = one_hot(Y)
    dZ2 = A2 - one_hot_Y
    dW2 = 1 / m * dZ2.dot(A1.T)
    db2 = 1 / m * np.sum(dZ2)
    dZ1 = W2.T.dot(dZ2) * ReLU_deriv(Z1)
    dW1 = 1 / m * dZ1.dot(X.T)
    db1 = 1 / m * np.sum(dZ1)
    return dW1, db1, dW2, db2

def update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha):
    W1 = W1 - alpha * dW1
    b1 = b1 - alpha * db1
    W2 = W2 - alpha * dW2
    b2 = b2 - alpha * db2
    return W1, b1, W2, b2

def gradient_descent(X, Y, alpha, iterations):
    W1, b1, W2, b2 = init_params()
    for i in range(iterations):
        Z1, A1, Z2, A2 = forward_prop(W1, b1, W2, b2, X)
        dW1, db1, dW2, db2 = backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y)
        W1, b1, W2, b2 = update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha)
    return W1, b1, W2, b2
```

Quellen

- Formeln, Algorithmus und Code von Samson Zhang
- `https://www.kaggle.com/code/wwsalmon/simple-mnist-nn-from-scratch-numpy-no-tf-keras/notebook` (Aufruf am 11.07.23)