基于向量的反向传播算法[*]

Laqudee Zhao[†]

2019-05-25

[*]Prof. Deng S.W

[†]E-mail:yunlongzhao1996@gmail.com

目录

1 向量化表示

将网络的一些参数用向量化的形式进行表示，如下
输入数据 $x \in R^N$，输出 $a \in R^M$。
网络参数：$Wv \in R^{M \times N, b \in R^M}$。
网络的净输入：$z = Wx + b$，其中 $z \in R^M$。非线性输出：$a = f(z)$，其中 $f(\cdot)$ 是作于向量元素上的函数，即；

$$f(z) = [f(z_1, \ldots, f(z_M)]^T$$

其中 z_i 是向量 z 的第 i 个元素。

函数 $f(z)$ 的微分为

$$\begin{aligned} df(z) &= [f'(z_1), \ldots, f'(z_M)]^T \circ dz \\ &= f'(z) \circ dz \end{aligned}$$

其中 $f(\cdot) : R \to R$ 为非线性神经元函数。

2 深层网络的表示

图 1 给出深层网络的架构模型，共包含 L 个层，其中第 1 层为输入层，第 2 层至第 L-1 层为隐藏层，第 L 层为输出层。在此网络结构中需要学习的网络参数为 $(W_i, b_i, i = 1, \ldots, L - 1$。

2.1 参数说明

由于我们要学习网络的参数，所以这里对参数进行必要的说明：
第 l 层的净输入：$z_l = W_{l-1}a_{l-1} + b_{l-1}$；
W_l：第 l 层的加权矩阵；
b_L：第 l 层的偏移 bias；
a_l：第 l 层的输出（也称为激励，activation）$a_l = f(z_l)$；
$J(\cdot)$：误差函数，度量网络输出与真实的判别。如：$J(z_L) = \frac{1}{2}||z_L - y||_2^2$。

2.2 信息的正向传播

这一部分介绍信息的正向传播：
第 1 层输出：$a_1 = x$；

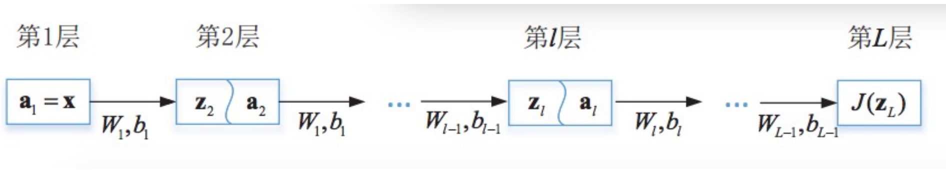

图 1: 深层网络架构模型

第 2 层输出：$a_2 = f(z_2); z_2 = W_1 a_1 + b_1$；
$\vdots$ 第 l 层：$a_l = f(z_l); z_l = W_{l-1} a_{l-1} + b_{l-1}$；
$\vdots$ 第 L 层：$J(z_L) = \frac{1}{2}||z_L - y||_2^2$；

3 反向传播算法

求解第 l 层网络参数相当于求解如下优化问题

$$\min W_l, b_l J(z_L)$$

令 y 是 x 的函数，$D[y;x]$ 表示 y 对 x 的 Jacobian 矩阵，即 $\nabla_x y = \dfrac{\partial y}{\partial x} = D[y;x]^T$； $d(y:x)$ 表示 y 对于 x 的微分。

3.1 J 关于参数 W_l 的梯度：$\nabla_{W_l} J$

与待求参数 W_l, b_l 有关的函数关系（线性）：$z_{l+1} = W_l a_l + b_l$。
J 关于参数 W_l 的微分

$$\begin{aligned} d(J;Wl) &= Tr(D[J;z_{l+1}]d(z_{l+1};W_l)) \\ &= Tr(D[J;z_{l+1}](dW_l a_l)) \\ &= Tr(a_l D[J;z_{l+1}]dW_l) \end{aligned}$$

J 关于参数 W_l 的梯度

$$\nabla_{W_l} J = D[J;z_{l+1}]^T a_l^T = \nabla J a_l^T = \delta_{l+1} a_l^T$$

其中：$\delta_{l+1} = \nabla_{z_{l+1}}$J 称为残差向量。

3.2 J 关于参数 b_l 的梯度：$\nabla_{b_l} J$

J 关于参数 b_l 的微分：

$$\begin{aligned} d(J;b_l) &= \mathrm{Tr}\left(D\left[J;\mathbf{z}_{l+1}\right]d\left(\mathbf{z}_l;\mathbf{b}_l\right)\right) \\ &= \mathrm{Tr}\left(D\left[J;\mathbf{z}_{l+1}\right]d\mathbf{b}_l\right) \end{aligned}$$

J 关于参数 b_l 的梯度：

$$\nabla_l J = D[J;z_{l+1}]^T = \nabla_{z_{l+1}} J = \delta_{l+1}$$

3.3 推导残差 δ_l 的反向传播关系

由信息的正向传播关系，可得到

$$a_l = f(z_l) z_{l+1} = W_l a_l + b_l$$

即

$$z_{l+1} = W_l f(z_l) + b_l$$

残差 δ_l 和 δ_{l+1} 的关系

$$\begin{aligned} d(J;\mathbf{z}_l) &= Tr\left(D\left[J;\mathbf{z}_{l+1}\right]d\left(\mathbf{z}_{l+1};\mathbf{z}_l\right)\right) \\ &= Tr\left(D\left[J;\mathbf{z}_{l+1}\right]W_l d\left(f\left(\mathbf{z}_l\right);\mathbf{z}_l\right)\right) \\ &= Tr\left(D\left[J;\mathbf{z}_{l+1}\right]W_l\left(f'\left(\mathbf{z}_l\right)\circ d\mathbf{z}_l\right)\right) \\ &= Tr\left(\left[\left(D\left[J;\mathbf{z}_{l+1}\right]W_l\right)\circ f'^T\left(\mathbf{z}_l\right)\right]d\mathbf{z}_l\right) \end{aligned}$$

因此得到

$$\frac{\partial J}{\partial z_l} = (W_l^T D[J;z_{l+1}]^T)\circ f'(z_l) = (W_l^T \frac{\partial J}{\partial z_{l+1}})\circ f'(z_l)$$

即

$$\delta_l = (W_l^T \delta_{l+1})\circ f'(z_l)$$

注：这里用到了 $Tr(A(B\circ C)) = Tr((A\circ B^T)C)$。

这个式子就体现了"误差的反向传播", 即当前层的误差 δ_l 是有当前层的误差 δ_{l+1} 经由当前层的网络参数 W_{l+1} 传播回来。

3.4 网络参数更新方法

$$\begin{aligned} W_l^{(k+1)} &= W_l^{(k)} - \lambda \nabla_{W_l} J \\ b_l^{(k+1)} &= b_l^{(k)} - \lambda \nabla_{b_l} J \end{aligned}$$

其中参数的梯度为

$$\nabla_{W_l} J = \delta_l a_{l-1}^T$$
$$\nabla_{b_l} J = \delta_l$$

4 Python Codes about Feedward Neural Network

使用 Python 语言对前馈神经网络（反向传播算法）进行编程，理论加上实践，加深理解！[1]

```
import numpy as np

# generate data
np.random.seed(1)
X = np.random.rand(12288, 200) # X is [12288 x 200]
Y = np.random.rand(1, 200)

# setup configuration of the network'
n0, m = X.shape
n1 = 20
n2 = 7
n3 = 5
n4 = 1
layers_dims = [n0, n1, n2, n3, n4] #[12288, 20, 7, 5, 1]
L = len(layers_dims) - 1 # the number of layers, excluding the input layer

# activation function
def sigmoid(z):
    '''
    z is the prev_activation, with sizze n1xm
    '''
    return 1 / (1 + np.exp(-z))

# the relu
def relu(z):
```

[1]结合 An Introduction to Neural Networks 学习

```
    '''
    z is the prev_activation, with sizze n1xm
    '''
    return np.maximum(0,z)

param_w = [i for i in range(L+1)]
param_b = [i for i in range(L+1)]

# neural network model
def neural_network(X, Y, learning_rate=0.01, num_iterations=2000, lambd =0):
    m = X.shape[1]
    # initialize forward prop
    np.random.seed(10)
    for l in range(1, L+1):
        if l < L:
            # use He initialization
            param_w[l] = np.random.randn(layers_dims[l], layers_dims[l - 1]) * np.sqrt
        if l == L:
            param_w[l] = np.random.randn(layers_dims[l], layers_dims[l - 1]) * 0.01
        param_b[l] = np.zeros((layers_dims[l], 1))

    activations = [X, ] + [i for i in range(L)]
    prev_activations = [i for i in range(L+1)]

    dA = [i for i in range(L+1)]
    dz = [i for i in range(L+1)]
    dw = [i for i in range(L+1)]
    db = [i for i in range(L+1)]

    for i in range(num_iterations):
        ### forward propagation
        for l in range(1, L+1):
            prev_activations[l] = np.dot(param_w[l], activations[l-1]) + param_b[l]
            if l < L:
                activations[l] = relu(prev_activations[l])
            else:
```

```
            activations[l] = sigmoid(prev_activations[l])

    cross_entropy_cost = -1/m * (np.dot(np.log(activations[L]), Y.T) \
                            + np.dot(np.log(1-activations[L]), 1-Y.T))
    regularization_cost = 0
    for l in range(1, L+1):
        regularization_cost += np.sum(np.square(param_w[l])) * lambd/(2*m)
    cost = cross_entropy_cost + regularization_cost

    ### initialize backward propagation
    dA[L] =  np.divide(1-Y, 1-activations[L]) - np.divide(Y, activations[L])
    assert dA[L].shape == (1, m)

    ### backward propagation
    for l in reversed(range(1, L+1)):
        if l == L:
            dz[l] = dA[l] * activations[l] * (1-activations[l])
        else:
            dz[l] = dA[l].copy()
            dz[l][prev_activations[l] <= 0] = 0

        dw[l] = 1/m * np.dot(dz[l], activations[l-1].T) + param_w[l] * lambd/m
        db[l] = 1/m * np.sum(dz[l], axis=1, keepdims=True)
        dA[l-1] = np.dot(param_w[l].T, dz[l])

        assert dz[l].shape == prev_activations[l].shape
        assert dw[l].shape == param_w[l].shape
        assert db[l].shape == param_b[l].shape
        assert dA[l-1].shape == activations[l-1].shape

        param_w[l] = param_w[l] - learning_rate * dw[l]
        param_b[l] = param_b[l] - learning_rate * db[l]

    if i % 100 == 0:
        print("cost after iteration {}: {}".format(i, cost))
print(param_w) #output param_w
```

```
    print(param_b) #output param_b

neural_network(X,Y,0.01,2000,0) #test

# predict
def predict(X_new, param_w, param_b, threshold=0.5):

    activations = [X_new, ] + [i for i in range(L)]
    prev_activations = [i for i in range(L + 1)]
    m = X_new.shape[1]

    for l in range(1, L + 1):
        prev_activations[l] = np.dot(param_w[l], activations[l - 1]) + param_b[l]
        if l < L:
            activations[l] = relu(prev_activations[l])
        else:
            activations[l] = sigmoid(prev_activations[l])
    prediction = (activations[L] > threshold).astype("int")
    return prediction

X_new = np.random.rand(1228, 200) #setting X for predict
print(predict(X_new,param_w,param_b,0.5))
```

这段代码的输出:

```
python3 "Feedward_neural_network.py"
cost after iteration 0: [[0.69301356]]
cost after iteration 100: [[0.69204859]]
cost after iteration 200: [[0.68890952]]
cost after iteration 300: [[0.66534268]]
cost after iteration 400: [[0.63627974]]
cost after iteration 500: [[0.59864491]]
cost after iteration 600: [[0.5739967]]
cost after iteration 700: [[0.55455259]]
cost after iteration 800: [[0.53362871]]
cost after iteration 900: [[0.52812387]]
cost after iteration 1000: [[0.52214822]]
```

```
cost after iteration 1100: [[0.50656239]]
cost after iteration 1200: [[0.51940142]]
cost after iteration 1300: [[0.51177312]]
cost after iteration 1400: [[0.50393255]]
cost after iteration 1500: [[0.5093858]]
cost after iteration 1600: [[0.54099861]]
cost after iteration 1700: [[0.51454733]]
cost after iteration 1800: [[0.50515268]]
cost after iteration 1900: [[0.50401095]]
[0, array([[ 0.01638766,  0.01138265, -0.02091715, ...,  0.00165189,
        -0.01230952,  0.00682659],
       [-0.00113278,  0.01496197, -0.00097725, ...,  0.00164064,
         0.01015594,  0.01345605],
       [-0.00416757, -0.00189723,  0.00234054, ..., -0.01189379,
         0.01006326,  0.00710751],
       ...,
       [-0.00197312,  0.02052377,  0.00614519, ..., -0.01311107,
        -0.03803787, -0.0138465 ],
       [ 0.00514454,  0.01111111, -0.01342621, ...,  0.01234376,
         0.00071711, -0.0066195 ],
       [ 0.03262465, -0.00947731,  0.01530282, ...,  0.00484326,
         0.01354119,  0.02126297]]), array([[ 0.01912191,  0.04196026,  0.27438578,  
        -0.21773836,  0.4825424 , -0.40669137, -0.34690234,  0.48052285,
        -0.33974462,  0.26520623, -0.01865162,  0.4241018 ,  0.20174834,
         0.1039465 ,  0.37880099,  0.1737232 ,  0.17187057,  0.26139308],
       [ 0.48421729,  0.10314613,  0.22588707, -0.20427408, -0.39853048,
         0.04080802, -0.23164776, -0.01279516,  0.47221056, -0.16227186,
         0.9641526 , -0.12169239,  0.35123843,  0.42705207, -0.50914111,
        -0.13650042,  0.28437599, -0.05770976,  0.15680806,  0.35728792],
       [ 0.35830798,  0.14216077, -0.38399224,  0.08555287,  0.22230298,
         0.15122015, -0.33419958, -0.21720598,  0.75175004,  0.11819726,
        -0.15339708, -0.26921506,  0.09312971,  0.67418608,  0.67905009,
        -0.46898748,  0.13862676, -0.28366271,  0.0240529 , -0.31381488],
       [-0.15973698,  0.20094888, -0.19895054, -0.14651321, -0.23449109,
        -0.13986819,  0.19774475,  0.02840331, -0.24391451, -0.68849919,
        -0.07820212, -0.51385715,  0.23765027, -0.04845484,  0.07101906,
```

```
          0.30209362, -0.34752361,  0.83278617,  0.07934278, -0.40506337],
        [ 0.44469925, -0.34742146,  0.02763839,  0.16792274,  0.22497983,
         -0.24825321, -0.21113133, -0.12917798,  0.19868364, -0.00796331,
          0.23200435, -0.37201899,  0.08778371, -0.37047047, -0.08562873,
          0.31229745, -0.25055157,  0.09665949, -0.38720322,  0.02096459],
        [ 0.51939386,  0.13965929,  0.3183007 , -0.13634163,  0.06882158,
          0.12774635,  0.52392282,  0.11908601, -0.52408946,  0.45495114,
          0.67460416,  0.1093505 ,  0.15247469, -0.38573521, -0.07825415,
         -0.06632824, -0.0930518 ,  0.02265276, -0.33761523, -0.38955973],
        [-0.0715185 ,  0.92267244, -0.30055864, -0.49622627,  0.53885152,
          0.43980193,  0.00468525,  0.21756674,  0.4789991 ,  0.49070254,
          0.39465037,  0.13812055, -0.03546575, -0.05881502, -0.23569456,
         -0.23993666,  0.05837225,  0.12733722, -0.65247895, -0.14843148]]), array([[-0
         -0.15698973,  0.63885495],
        [ 1.04488303, -0.1162235 ,  1.32286932, -0.52163963, -0.18955205,
         -0.3834946 ,  0.13148026],
        [-0.41788423,  0.40318132,  0.65351408, -0.19448527,  0.63302513,
         -0.4810707 ,  0.0951204 ],
        [ 0.55302784,  0.41724538, -0.82763521,  0.62488907,  0.47772772,
         -0.50575445, -0.13837589],
        [ 0.08779301,  0.78472521, -0.53616679,  0.38610954,  0.06211733,
         -0.16292641,  0.23040447]]), array([[-0.07163096, -0.1259331 , -0.07865121,  0
[0, array([[ 2.15687998e-03],
        [ 3.65281427e-04],
        [ 5.25673136e-04],
        [-3.81742282e-04],
        [ 2.45619490e-04],
        [ 0.00000000e+00],
        [-3.77014595e-05],
        [ 1.73270951e-04],
        [ 0.00000000e+00],
        [-2.93681257e-04],
        [ 9.32962025e-04],
        [ 1.26161452e-04],
        [-1.43886710e-04],
        [ 1.89096949e-03],
```

```
       [ 1.63073898e-05],
       [ 3.36940373e-04],
       [-6.08844691e-05],
       [-7.52277890e-05],
       [ 5.43276475e-05],
       [ 4.77160169e-04]]), array([[2.42649185e-03],
       [3.00505761e-03],
       [1.67293234e-03],
       [1.16826608e-05],
       [1.50744262e-03],
       [5.76163824e-04],
       [2.18044184e-04]]), array([[-0.00070068],
       [-0.00508723],
       [-0.00617541],
       [ 0.00928946],
       [ 0.00257481]]), array([[0.05181475]])]
[[1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0
  0 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0
  0 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1 1 1 1 0
  0 0 1 1 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1
  0 0 1 0 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 0 0 1 1 0 0 0 0 1 1 1 1 0 1 1 0 1
  1 0 1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0 0 1]]
[[1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0
  0 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0
  0 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1 1 1 1 0
  0 0 1 1 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1
  0 0 1 0 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 0 0 1 1 0 0 0 0 1 1 1 1 0 1 1 0 1
  1 0 1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0 0 1]]

Process finished with exit code 0
```