Homework#5

Yuhang Peng

5.6

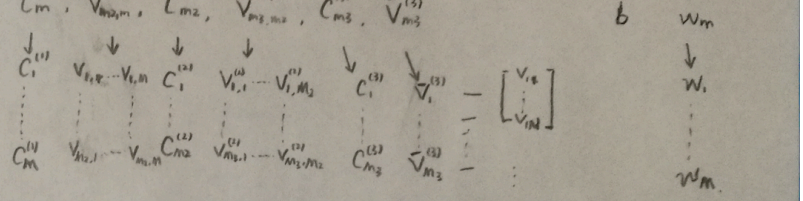
Exercise 5.6

$$r = b + \sum_{m=1}^{M_1} f_m(x) w_m$$

a). For three hidden layer, each $f(\vec{x})$:

$$f_m(\vec{x}) = \max\left(0, C_m^{(1)} + \sum_{m_2=1}^{M_2} \max\left(0, C_{m_2}^{(2)} + \sum_{m_3=1}^{M_3} \max\left(0, C_{m_3}^{(3)} + \vec{x}^T \vec{V}_{m_3}^{(3)}\right) V_{m_3,m_2}^{(2)}\right) V_{m_2,m}^{(1)}\right)$$

as we see, there are $C_m^{(1)}$, $V_{m_2,m}^{(1)}$, $C_{m_2}^{(2)}$, $V_{m_3,m_2}^{(2)}$, $C_{m_3}^{(3)}$, $\vec{V}_{m_3}^{(3)}$



Therefore, we totally have parameter $Q = M_3 + M_3 \cdot N + M_2 \cdot M_3 + M_2 + M_2 \cdot M + 2M + 1$

$$= \boxed{(1+N+M_2)\cdot M_3 + (1+M)\cdot M_2 + 2M_1 + 1}$$

$$= (1+N)M_3 + (1+M_3)M_2 + (1+M_2)M_1 + M_1 + 1$$

Generalize to L hidden layers:

We assume that $M_0 = 1$, $M_1 = M$, $M_{L+1} = N$.

$$Q = \boxed{(1+M_{L+1})M_L + (1+M_L)M_{L-1} + \cdots + (1+M_2)M_1 + (1+M_1)M_0}$$

where $M_0 = 1$, $M_1 = M$, and $M_{L+1} = N$

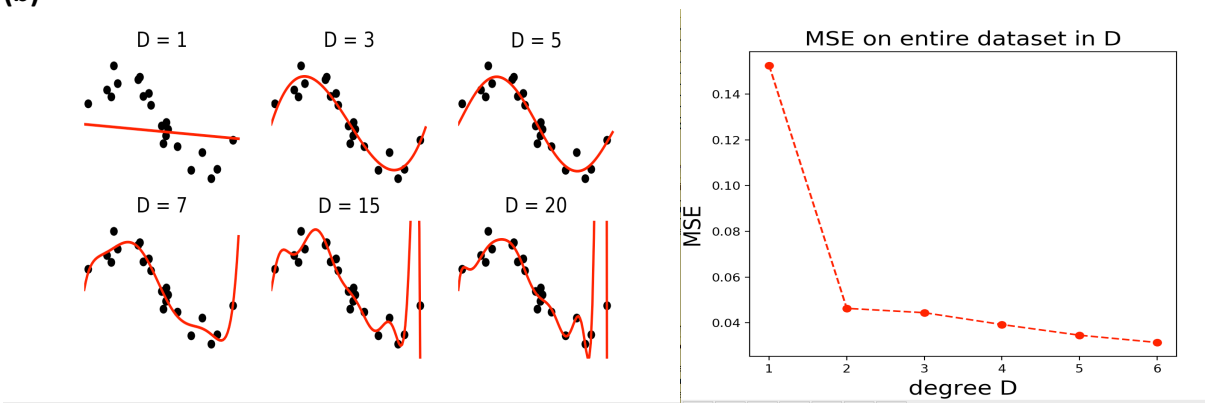
b). Increasing the layers, it will be greater flexibility. The input dimension is only small part of the number of parameters Q, when we increase the data points P, which is the input dimension N, the number of parameters will not change too much.

5.7

(a)

```python
# YOUR CODE GOES HERE takes poly features of the input
def poly_features(x,D):
    F = []
    for i in x:
        f_col = []
        for j in range(1 , D + 1):
            f_col.append(i ** j)
        F.append(f_col)
    F = np.array(F)
    F.shape = (len(x),D)
    F = F.T
    return F
```

(b)



The left figure shows the function we generate by choosing different D, and the right figure shows the mean squared error at each fit to the data set. When we increase the D, the error will decrease, and the figure will become more similar to the original function. However, these data are real data, which means there should have some error, not perfect, when the D is too large, the result will become different, which is called over-fit. We also need to make sure that D is enough, and if D is too small, there will be much errors, and result is not we expect, which is called under-fit. Therefore, we need to choose a suitable D fit for the data set. For this question, the D should be located at 3-5.

5.9

(a)

a).

For the Least squares problem equation:

$$g = \sum_{p=1}^{P} (b + \bar{f}_p^T \bar{w} - y_p)^2$$

with the general activation function $\alpha(\cdot)$, then we can get

$$f_p = [\alpha(c_1 + \bar{x}_p^T \bar{v}_1), \alpha(c_2 + \bar{x}_p^T \bar{v}_2), \ldots, \alpha(c_m + \bar{x}_p^T \bar{v}_m)]^T$$

$$g = \sum_{p=1}^{P} (b + \sum_{m=1}^{M} \alpha(c_m + \bar{x}_p^T \bar{v}_m) w_m - y_p)^2$$

$$\frac{\partial g}{\partial b} = 2 \sum_{p=1}^{P} (b + \sum_{m=1}^{M} \alpha(c_m + \bar{x}_p^T \bar{v}_m) w_m - y_p)$$

$$\frac{\partial g}{\partial w_n} = 2 \sum_{p=1}^{P} (b + \sum_{m=1}^{M} \alpha(c_m + \bar{x}_p^T \bar{v}_m) w_m - y_p) \alpha(c_n + \bar{x}_p^T \bar{v}_n)$$

$$\frac{\partial g}{\partial c_n} = 2 \sum_{p=1}^{P} (b + \sum_{m=1}^{M} \alpha(c_m + \bar{x}_p^T \bar{v}_m) w_m - y_p) \alpha'(c_n + \bar{x}_p^T \bar{v}_n) w_n$$

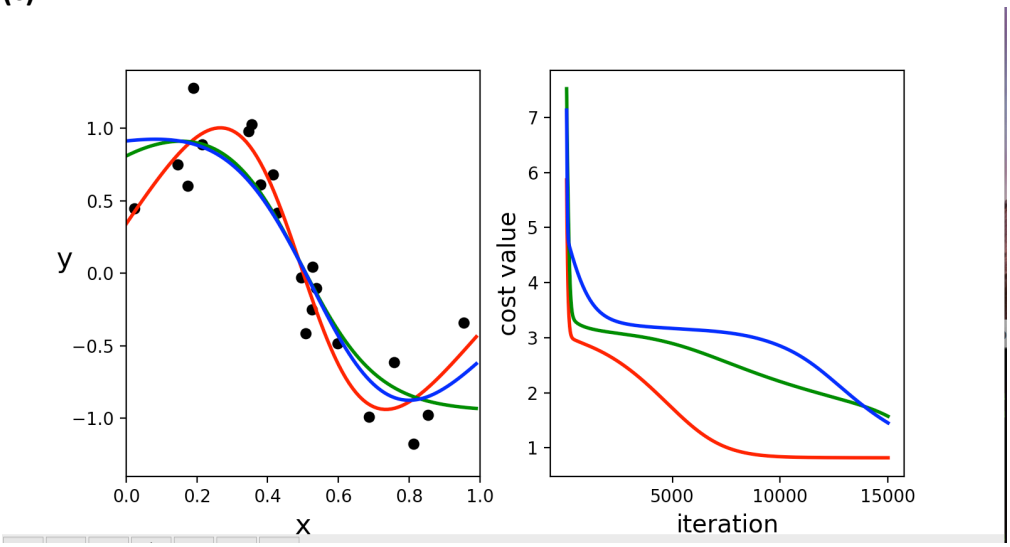
$$\nabla_{\bar{v}_n} g = 2 \sum_{p=1}^{P} (b + \sum_{m=1}^{M} \alpha(c_m + \bar{x}_p^T \bar{v}_m) w_m - y_p) \alpha'(c_n + \bar{x}_p^T \bar{v}_n) w_n \bar{x}_p$$

which is same with the one given by the question.

(b)

```python
def tanh_grad_descent(x,y,i):
    # initialize weights and other items
    b, w, c, v = initialize(i)
    P = np.size(x)
    M = 4
    alpha = 1e-3
    l_P = np.ones((P,1))

    # stoppers and containers
    max_its = 15000
    k = 1
    cost_val = []       # container for objective value at each iteration

    ### main ###
    for k in range(max_its):
        # update gradients
        q = np.zeros((P,1))
        for p in np.arange(0,P):
            # YOUR CODE HERE, for vector q
            q[p] = b + np.dot(w.T, np.tanh(c + x[p] * v)) - y[p]

        # YOUR CODE HERE, for grad_b
        grad_b = 2 * np.dot(l_P.T, q)

        grad_w = np.zeros((M,1))
        grad_c = np.zeros((M,1))
        grad_v = np.zeros((M,1))
        for m in np.arange(0,M):
            # YOUR CODE HERE
            t = np.tanh(c[m] + x * v[m])
            s = (1 / np.cosh(c[m] + x * v[m])) ** 2
            grad_w[m] = 2 * np.dot(l_P.T, q * t)
            grad_c[m] = 2 * np.dot(l_P.T, q * s) * w[m]
            grad_v[m] = 2 * np.dot(l_P.T, q * x * s) * w[m]

        # take gradient steps
        b = b - alpha*grad_b
        w = w - alpha*grad_w
        c = c - alpha*grad_c
        v = v - alpha*grad_v

        # update stopper and container
        k = k + 1
        cost_val.append(compute_cost(x,y,b,w,c,v))

    return b, w, c, v, cost_val
```

(c)



From the figure, we can see that when we increase the iteration, the result will become much more precise, and the cost value will be smaller and smaller. For different initialization, the function will be a little bit different, so does cost value.