sheet3

November 7, 2017

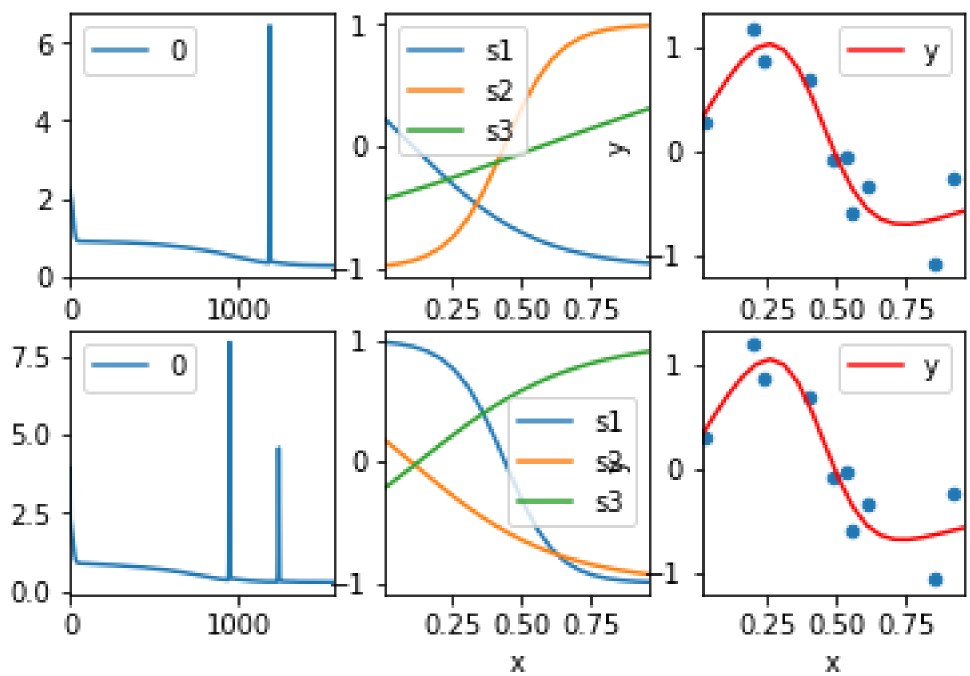
```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import random
        % matplotlib inline
        #x0 x1 y
        training_data = pd.read_csv('regressionData.txt', delimiter=" ", header=None, names=["x1
        training_data["x0"] = training_data["x1"] / training_data["x1"]

        x = training_data[["x0", "x1"]].values
        y_t = training_data["y"].values

        def h(w):
            def _h(x):
                return np.dot(w.T,x)
            return _h

        def training_step(learning_rate):
            global i
            global w_21_00
            global w_21_01
            global w_21_02
            global w_21_03
            global w_10_00
            global w_10_01
            global w_10_10
            global w_10_11
            global w_10_20
            global w_10_21
            global E_T

            i += 1
            # Weight vectors
            w_10 = np.array([w_10_00, w_10_01])
            w_11 = np.array([w_10_10, w_10_11])
            w_12 = np.array([w_10_20, w_10_21])
            w_20 = np.array([w_21_00, w_21_01, w_21_02, w_21_03])
```

```
# compute hs of hidden layer (wT * x)
h_10 = np.apply_along_axis(h(w_10), 1, x)
h_11 = np.apply_along_axis(h(w_11), 1, x)
h_12 = np.apply_along_axis(h(w_12), 1, x)

# compute ss of hidden layer (f(h))
s_10 = np.apply_along_axis(np.tanh, 0, h_10)
s_11 = np.apply_along_axis(np.tanh, 0, h_11)
s_12 = np.apply_along_axis(np.tanh, 0, h_12)

s_1 = np.array([training_data["x0"], s_10, s_11, s_12]).T

# compute input for last neuron (Add x0 for bias)
h_2 = np.apply_along_axis(h(w_20), 1, s_1)

# compute output (f(h_2))
y_pred = h_2

# compute error
E_T = 0.5*np.sum(np.power(np.subtract(y_pred, training_data["y"]), 2))
#print("Error: "+str(E_T))

# compute local errors
# da die ableitung der identität 0 ist
delta_2_0 = np.array([1,1,1,1,1,1,1,1,1,1])

# da die ableitung von tanh(x) 1/coshš(x)
delta_10 = np.multiply(np.power(np.cosh(h_10), -2), w_21_01)
delta_11 = np.multiply(np.power(np.cosh(h_11), -2), w_21_02)
delta_12 = np.multiply(np.power(np.cosh(h_12), -2), w_21_03)
# compute gradients

grad_10_00 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_10),
grad_10_01 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_10),

grad_10_10 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_11),
grad_10_11 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_11),

grad_10_20 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_12),
grad_10_21 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_12),


grad_21_00 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_2_0)
grad_21_01 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_2_0)
grad_21_02 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_2_0)
grad_21_03 = -0.1*np.sum(np.multiply(np.multiply(np.subtract(y_pred, y_t),delta_2_0)
```

```
    #adjust w
    w_21_00 = w_21_00 + learning_rate * grad_21_00
    w_21_01 = w_21_01 + learning_rate * grad_21_01
    w_21_02 = w_21_02 + learning_rate * grad_21_02
    w_21_03 = w_21_03 + learning_rate * grad_21_03

    w_10_00 = w_10_00 + learning_rate * grad_10_00
    w_10_01 = w_10_01 + learning_rate * grad_10_01

    w_10_10 = w_10_10 + learning_rate * grad_10_10
    w_10_11 = w_10_11 + learning_rate * grad_10_11

    w_10_20 = w_10_20 + learning_rate * grad_10_20
    w_10_21 = w_10_21 + learning_rate * grad_10_21

def plotNetwork(fig, pos):
    x = np.arange(0.01, 1, 0.05)
    x = np.array([x/x, x]).T
    # Weight vectors
    w_10 = np.array([w_10_00, w_10_01])
    w_11 = np.array([w_10_10, w_10_11])
    w_12 = np.array([w_10_20, w_10_21])
    w_20 = np.array([w_21_00, w_21_01, w_21_02, w_21_03])

    # compute hs of hidden layer (wT * x)
    h_10 = np.apply_along_axis(h(w_10), 1, x)
    h_11 = np.apply_along_axis(h(w_11), 1, x)
    h_12 = np.apply_along_axis(h(w_12), 1, x)

    # compute ss of hidden layer (f(h))
    s_10 = np.apply_along_axis(np.tanh, 0, h_10)
    s_11 = np.apply_along_axis(np.tanh, 0, h_11)
    s_12 = np.apply_along_axis(np.tanh, 0, h_12)

    s_1 = np.array([x[:,0], s_10, s_11, s_12]).T
    ax = fig.add_subplot(2,3,pos)
    pd.DataFrame(np.array([x[:,1], s_10, s_11, s_12]).T, columns=["x", "s1", "s2", "s3"]

    # compute input for last neuron (Add x0 for bias)
    h_2 = np.apply_along_axis(h(w_20), 1, s_1)
    # compute output (f(h_2))
    y_pred = h_2

    ax = fig.add_subplot(2,3,pos+1)
    ax = training_data.plot(x="x1", y="y", kind="scatter", ax=ax)
    pd.DataFrame(np.array([x[:,1], y_pred]).T, columns=["x","y"]).plot(ax=ax, x="x", y="
```

```
def train(learning_rate, fig, figure_offset):
    global i
    global w_21_00
    global w_21_01
    global w_21_02
    global w_21_03
    global w_10_00
    global w_10_01
    global w_10_10
    global w_10_11
    global w_10_20
    global w_10_21
    global E_T

    w_10_00 = random.uniform(-0.5, 0.5)
    w_10_01 = random.uniform(-0.5, 0.5)

    w_10_10 = random.uniform(-0.5, 0.5)
    w_10_11 = random.uniform(-0.5, 0.5)

    w_10_20 = random.uniform(-0.5, 0.5)
    w_10_21 = random.uniform(-0.5, 0.5)

    w_21_00 = random.uniform(-0.5, 0.5)
    w_21_01 = random.uniform(-0.5, 0.5)
    w_21_02 = random.uniform(-0.5, 0.5)
    w_21_03 = random.uniform(-0.5, 0.5)

    i = 0

    learning_rate = 0.5
    errors = []
    E_T = 0
    E_T_last = 999999

    MAX_ITERATIONS = 3000

    while True:
        E_T_last = E_T
        training_step(learning_rate)
        errors.append(E_T)
        #if (i % 100 == 0):
        #    print(i)
        if (i > MAX_ITERATIONS or abs(E_T_last - E_T) <= 10 ** -5):
            break;
```

```
    #print(errors)
    errors = pd.DataFrame(errors)
    ax = fig.add_subplot(2,3,figure_offset)
    errors.plot(ax=ax)


    plotNetwork(fig, figure_offset+1)

fig = plt.figure()
train(0.5, fig, 1)
train(0.5, fig, 4)
```



d) Although the overall prediction of y is very similar in both MLPs, the output functions of the hidden neurons are completely different. This is caused by the random initialization of the weights: The weights are the starting point for gradient descent, which based on it finds different local optima.

e) We know that the noise is Gaussian distributed, which makes big outliners very unlikely. Furthermore, the y_T values vary in $[-1, 1]$, thus quite strongly. Therefore, we want a cost functions that is strongly affected outliners.