APM120, Homework #9

Applied Linear Algebra and Big Data

Last updated: Tuesday 4th April, 2023, 16:32

Assigned April 4, due **Thursday, April 13**, 1:00 pm, via gradescope, **in pdf $\leq$ 20Mb, $\leq$ 30 pages.**

Supervised learning: classification

Show all steps in all calculations explicitly. Attach code used, well documented, and relevant plots and Matlab/python output, attaching code and figures *immediately following* the relevant question solution. A code printout is not a substitute for complete solutions, your solution should stand alone without the Matlab/python code or output. See needed python preliminaries at end of this HW[1]. It is fine to use Matlab/python unless a hand calculation (using only a simple calculator) is required *in orange*. For all questions, make sure you show all steps as if you are doing the problem by hand, and do not use library functions unless explicitly allowed in the question. Make sure you can do all calculations using no more than a hand-held calculator. See the end-note for guidelines and examples of hand-calculations.[2]

1. **Perceptron for 2d data:** Consider the data vectors given by the columns of X and the corresponding binary classification labels **y** given below. (**A**) Make a scatter plot of the data using different symbols and different colors for the two classes. (**B**) Estimate two possible perceptron solutions for $[\mathbf{w}, b]$ based on the plot (the perceptron line is given by $\mathbf{w}^T\mathbf{x} = b$). Plot the lines corresponding to the two solutions. Explain why the perceptron solution is not unique in general. (**C**) Hand-calculation: Consider first a smaller data set, corresponding to data points

```
% Matlab:
X(:,[6,10,16]) and y([6,10,16])
```

```
# python:
X[:,[5,9,15]] and y[[5,9,15]]
```

and perform the first two epochs, going over all data points twice, for calculating the perceptron `[w,b]` using a learning rate of $\eta = 0.2$, and an initial guess of `w=[1,-1]'; theta=1;`. (**D**) ***optional extra credit** write code to analyze the entire data set (can start from the code on the course web page), finding a perceptron solution that converged. You will need to adjust the learning rate and the number of iterations until you find values that lead to convergence.

```
X=[-3.6,  1.8, -7.2, -9.6,  0.5, -0.9, -2.1, -2.8, -3.2, -2.2,  1.3, -3.6, -6.3, -0.9, -2.4,  5.2;
   7.8, -7.5, -0.6,  4.5,  0.9,    0,   -7, -6.8, -6.4, -9.3, -5.9, -2.5, -7.9, -6.1,  8.6,  5.4];
y=[  1,   -1,    1,    1,   -1,   -1,   -1,   -1,   -1,   -1,   -1,   -1,   -1,   -1,    1,    1];
```

2. **Nonlinear perceptrons! ***optional extra credit**. Load the data file (Matlab: `load nonlin-percept-HW.mat; whos`, python: `from scipy.io import loadmat; mat=loadmat('n`
`XX=mat['XX']; y=mat['y']; w=mat['w'];`) which includes the (data, labels) X, **y** and an initial guess for the perceptron **w**. Letting Matlab: `u=XX(1,:);v=XX(2,:)`, python: `u=XX[0,:];v=XX[1,:];` use a nonlinear perceptron of the form $w_1u_i + w_2v_i + w_3u_i^2 + w_4u_iv_i + w_5 = 0$ to classify the data. Plot the initial and final perceptron lines. You may want to start from the example in `perceptron_nonlinear_classification_example.m/py`.

3. **Neural networks:** consider the following neural network: The hidden layer(s) are composed of "sigmoid" neurons and the output layer is linear,

```
% Matlab:
w2=[-0.31314,  0.38796;
    -0.60718,  0.67412;
     2.8908, -1.5217];
b2=[ 0.81989;
     0.81574;
      3.674];
w3=[ -1.007, -1.7446,  5.1881;
     16.877, -7.3917, -4.1027;
     50.772, -21.437, -38.171];
b3=[-15.236;
    -1.8673;
     20.628];
w4=[  1.812,  44.299, -35.869];
b4=[ 5.2558];
```

```
# python:
w2=[[-0.31314,  0.38796 ],
    [-0.60718,  0.67412 ],
    [ 2.8908, -1.5217 ]]
b2=[[ 0.81989 ],
    [ 0.81574 ],
    [  3.674 ]]
w3=[[ -1.007, -1.7446,  5.1881 ],
    [ 16.877, -7.3917, -4.1027 ],
    [ 50.772, -21.437, -38.171 ]]
b3=[[-15.236 ],
    [-1.8673 ],
    [ 20.628 ]]
w4=[  1.812,  44.299, -35.869 ]
b4=[ 5.2558 ]
```

The inputs can be any pair (x_1, x_2) in the range of $1 < x_1 < 2$, $1 < x_2 < 2$. (**A**) draw the network and write all of its equations explicitly. (**B**) Calculate by hand and show all intermediate neuron values for the example input [1.8;1.1]. (**C**) Calculate the outputs for 10 or so additional possible inputs covering the relevant range of (x_1, x_2). Deduce which simple function of the inputs x_1, x_2 the network returns. Don't you think it's amazing that this actually works? :-) Does it also work for the input [-0.5;3.5]? Can you guess why? *Hint:* you can define activation functions using,

```
% Matlab:
sigmoid = @(x) 1.0./(1.0+exp(-x));
softmax = @(x) exp(x)/sum(exp(x(:)));
tansig = @(x) 2./(1+exp(-2.0*x))-1;
% and for rectified linear, place a
% regular function in a file rectlin.m:
function out=rectlin(x)
    out=x;
    out(out<0)=0;
% and then used as,
z=[1 2 3]';
sigmoid(z)
% or
softmax(z)
```

```
# python:
def sigmoid(x):
    return 1.0/(1.0+np.exp(-x))
def softmax(x):
    return np.exp(x)/np.sum(np.exp(x[:]))
def tansig(x):
    return 2/(1+np.exp(-2.0*x))-1
def rectlin(x):
    out=np.asarray(x); out[out<0]=0
    return out
# and then use as,
z=np.array([[1, 2, 3]]); z=z.T;
sigmoid(z) # or
softmax(z)
```

* [What's the point of ***optional extra credit** challenge problems: apart from the fun of doing them, they may bring the total score of this HW assignment up to 110%, making up for problems you may have missed in this or other HW assignments...]

Python preliminaries & notes

1 python commands within the HW assume you have first used the followings: `import numpy as np; from numpy import linalg; import scipy as scipy; from scipy import linalg; import matplotlib.pyplot as plt; import matplotlib;`
Input a matrix A, column vector **b** and row vector **c** into python in the form
`A=np.array([[a11,a12,a13],[a21,a22,a23],[a31,a32,a33]]; b=np.array([[b1],[b2],[b3]]); c=np.array([[c1,c2,c3]]);` or convert Matlab arrays given in HW directly to python arrays using, e.g., `A=np.array(np.matrixlib.defmatrix.matrix('[1 2 3; 4 5 6]'));`

2 **Hand calculations**, in which you are asked to use only a simple hand-held calculator, are required only when we want to make sure you understand exactly what each step of an algorithm is doing. These also prepare you for the quizzes that involve similar hand calculations. **How much work to show?** Just don't use scratch paper, show all steps that you actually use for the hand calculations, but no more, carrying out calculations to **three significant digits**. Trust the graders to be reasonable, they were students in the course last year. **Examples: (i)** If you are asked, *not* in orange, to calculate the LU decomposition of a matrix, you may use Matlab/python as in `A(2,:)=A(2,:)-A(1,:)*(A(2,1)/A(1,1))` etc, but you may not use a library routine as in `[L,U,P]=lu(A)` except for checking your results. **(ii)** If required to calculate by hand the element $C_{2,3}$ of a matrix product $\mathsf{C} = \mathsf{AB}$, you need to explicitly multiply using a hand calculator the second row of A with the third column of B. You may not use Matlab/python to calculate `C=A*B` and then take the needed element from that product, nor to calculate `C23=A(2,:)*B(:,3)`.