Homework 3

Parts A and B due in class April 3, 2019

Part C due online by 11:59pm April 7, 2019

A. Component analysis:

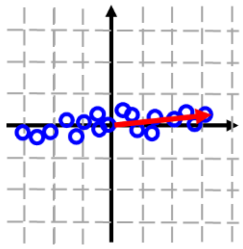
1. Convert the following vectors in (a), (b), and (c) to be unit vectors, each pointing in the direction of the original vector. (For example, $\begin{bmatrix}2\\-4\end{bmatrix}$, $\begin{bmatrix}8\\-16\end{bmatrix}$, and $\begin{bmatrix}0.4\\-0.8\end{bmatrix}$ are all **non-unit** vectors and all point in the same direction.)

(a) $\begin{bmatrix}1\\0\\-1\\2\\1\end{bmatrix}$ (b) $\begin{bmatrix}-4\\0\\0\\2\\-4\end{bmatrix}$ (c) $\begin{bmatrix}2\\0\\-3\\1\\-1\end{bmatrix}$

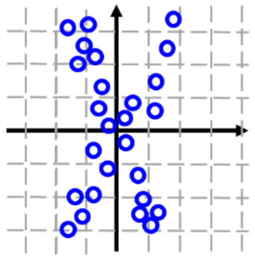
2. For each of the following data sets (set A and set B), provide the top two principal components and the top one or two independent components (if there are two clear independent components, you must provide both). Express each component as a 2-element vector $\begin{bmatrix}num1\\num2\end{bmatrix}$.

You may print out this page and draw arrows for partial credit. The vector estimate of each component should be estimated to the nearest tenth. E.g., for the following data, we may have a direction as a roughly horizontal arrow

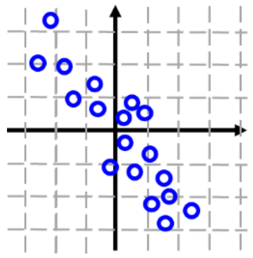
Roughly estimated as: $\boldsymbol{dir} \approx \begin{bmatrix}3.0\\0.5\end{bmatrix}$ or $\boldsymbol{u} \approx \begin{bmatrix}1.0\\0.2\end{bmatrix}$



Set A:



Set B:



3.
We can reconstruct an estimate of data point $\boldsymbol{x}$ using components $\boldsymbol{u}$ and their corresponding weights *z*

$$\widetilde{\boldsymbol{x}} = \sum_q z_q \boldsymbol{u}^q$$

where $\widetilde{\boldsymbol{x}}$ is the estimate of $\boldsymbol{x}$. However, the reconstruction will be inaccurate. A standard measure of inaccuracy between an original data vector $\boldsymbol{x}$ and the estimated version of this vector $\widetilde{\boldsymbol{x}}$ is called "mean squared error":

$$E(\boldsymbol{x}, \widetilde{\boldsymbol{x}}) = \sum_j \left(x_j - \widetilde{x_j}\right)^2$$

If our original data point is $\begin{bmatrix} 2 \\ 0 \\ -1 \\ 2 \\ 0 \end{bmatrix}$ and our estimate is $\begin{bmatrix} 1.8 \\ 0.5 \\ -0.8 \\ 2.3 \\ -0.3 \end{bmatrix}$, the error will be

$(2-1.8)^2 + (0-0.5)^2 + (-1+0.8)^2 + (2-2.3)^2 + (0+0.3)^2 =$

$0.2^2 + 0.5^2 + 0.2^2 + 0.3^2 + 0.3^2 = 0.04 + 0.25 + 0.04 + 0.09 + 0.09$: E=0.51

Data for Question 3:
We have the following components:

$$\boldsymbol{u}^1 = \begin{bmatrix} 0.4 \\ 0.7 \\ 0.4 \\ 0.4 \end{bmatrix}, \quad \boldsymbol{u}^2 = \begin{bmatrix} 0.6 \\ -0.8 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{u}^3 = \begin{bmatrix} 0.4 \\ 0 \\ 0 \\ -0.9 \end{bmatrix}$$

For each data point, we have three corresponding reconstruction weights:

$\boldsymbol{x}^1 = \begin{bmatrix} 1.3 \\ 0.5 \\ 0.4 \\ 0 \end{bmatrix}$ $z_1 = 1$ $z_2 = 0$ $z_3 = 2$

$\boldsymbol{x}^2 = \begin{bmatrix} 2.5 \\ -2 \\ 0.1 \\ -1 \end{bmatrix}$ $z_1 = 0$ $z_2 = 3$ $z_3 = 1.5$

$\boldsymbol{x}^3 = \begin{bmatrix} -1.2 \\ 0 \\ -0.6 \\ -0.4 \end{bmatrix}$ $z_1 = -1$ $z_2 = -1$ $z_3 = 0$

Actual questions:
(a) What are the estimated vectors for **x**^{1}, **x**^{2}, and **x**^{3} based on the corresponding *z* values above?

(b) What is the mean squared error between the estimated and actual data vectors for **x**^{1} and for **x**^{3}?

(c) Do we expect the components u and weights z in this question are derived from ICA, PCA, or NMF? Explain your answer (in 1-2 sentences).

4. Presume the following are independent components:

$$\boldsymbol{u}^1 = \begin{bmatrix} 0 \\ 0.67 \\ 0.67 \\ 0.33 \end{bmatrix}, \qquad \boldsymbol{u}^2 = \begin{bmatrix} 0.9 \\ -0.4 \\ 0 \\ 0.2 \end{bmatrix}, \qquad \boldsymbol{u}^3 = \begin{bmatrix} 0.3 \\ 0.9 \\ -0.3 \\ 0 \end{bmatrix}$$

(a) Which independent component **u** best describes each of the data points **x** below? In other words, which single component can most closely reconstruct each data point below?
(b) What is the corresponding weight z_{qj}^{i} for this strongest component?

$$\boldsymbol{x}^1 = \begin{bmatrix} -1.5 \\ -4.7 \\ 1.2 \\ 0.2 \end{bmatrix} \qquad \boldsymbol{x}^2 = \begin{bmatrix} -0.3 \\ 3.2 \\ 2.9 \\ 0.9 \end{bmatrix} \qquad \boldsymbol{x}^3 = \begin{bmatrix} 4.5 \\ 3.1 \\ -1.6 \\ -3.1 \end{bmatrix}$$

B. Neural networks

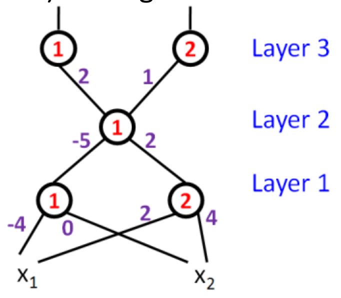
1. Let us assume we have a neural network with three layers. Layer 1 has 3 units, layer 2 has 5 units, and layer 3 has a 2 units. There are 8 features fed into the units in layer 1.

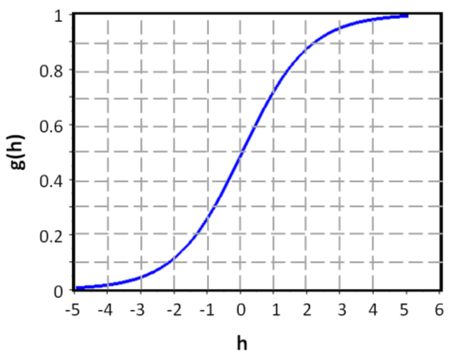
(a) Assuming we also have a unit-specific constant b_k^m offset for each unit, how many parameters must we learn for the network as described.

We establish a measurement of "likelihood" derived from the error $(r_1^{3,i}-y^i)^2$. Using this likelihood measure on a training data set and using additional variables, we observe the AIC for our current 3-layer neural network model is: -30.5 .

(b) If we add 3 more units to layer 2 and the likelihood remains unchanged, how will the AIC be affected? (Note AIC uses the natural logarithm, $\log_e$.)

2. Presume the following Neural Network, where each unit performs the standard dot product (weighted sum) and sigmoid transformation.





The following feature values are input: x₁=0.5 x₂=0.8

The original weights are: $w_{1,1}^{1} = -4, w_{1,2}^{1} = 0,\ \ w_{2,1}^{1} = 2, w_{2,2}^{1} = 4,$
$w_{1,1}^{2} = -5, w_{1,2}^{2} = 2,$
$w_{1,1}^{3} = 2,\ w_{2,1}^{3} = 1$

Assume all b's are -0.5

Note all initial weights are shown in purple in the diagram above.

(a) Compute the outputs of the 5 units given the input $x_1=1$ $x_2=0.5$

(b) Weight learning:

Assume the desired output for the top (layer 3) units are:

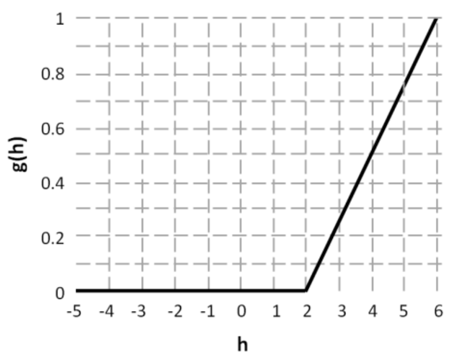
unit 1 output: 0 unit 2 output: 0

Assuming $\epsilon = 10$, compute new weight values for the weights below:

$w_{1,1}^{3} =$ $w_{2,1}^{3} =$ $w_{1,1}^{2} =$

Let us now replace the sum-and-sigmoid units in the current network with linear-rectifier units using the function to the right.

$$g(h) = g(r; w) = \begin{cases} 0 & w \cdot r \le 2 \\ \dfrac{(w \cdot r - 2)}{4} & w \cdot r > 2 \end{cases}$$



We wish to calculate a new w update rule to minimize the error in the output for neuron 2 in layer 3 r_2^3 compared to the desired output y_2 :

$$E(y; w) = \sum_{j} \left(y^{i} - g\left(r^{i}; w\right)\right)^{2}$$

Use your calculus tools on *E(y;w)* to compute the gradient ascent update rule for $w_{2,1}^{3}$.

C. Programming

In this section, you will work on two machine learning approaches – neural network classification and independent component analysis. There are a total of 6 questions – 2 for neural nets and 4 for ICA.

To submit Part C, create the directory HW3 inside your private/CIS5800 directory. Leave all relevant pieces of code in your private/CIS5800/ directory, in file hw3.py .

Your code must be able to run on Python 3. For example, you can test it on erdos as follows. On erdos terminal:

```
cd /usr/local/bin/anaconda3/bin
./python3
```

Then within python you can switch back to your own directory and load your hw2 using:

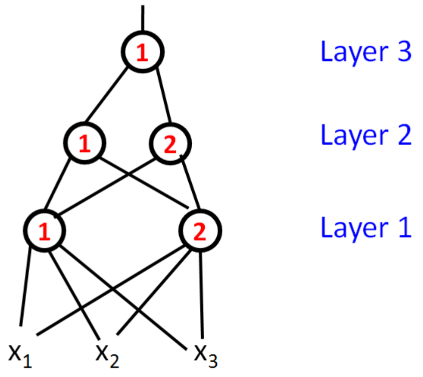
```
import os
os.chdir('/u/erdos/students/USERNAME/private/CIS5800/')
import hw2
```

It is fine if you have tested your code on your own machine, given you are using Python 3.

Before submitting your homework, comment out or delete any additional testing code that I did not request. If hw3.py includes code to load data sets, comment out this code and any code that depends on this code before submitting. I should be able to import your hw3.py file on my computer, where I will import the data files later from a different directory than you use.

NEURAL NETS

First, you will implement the 3-layer neural network shown to the right. Each unit computes a weighted sum and then applies the sigmoid function. You will represent the weights for each layer in a separate numpy array – layer1W, layer2W, layer3W. Each row will correspond to the weights for a single unit. So, layer1W will have shape [2, 4] – 2 units each taking in inputs from 3 features **plus** a constant b offset.



1. Write a function **feedforward** that takes in the features **x**^{i} and the network weights, and outputs the response from layer 3.

Specifically: you will be able to call the function as

```
  layer3Out=feedforward(x,layer1W,layer2W,layer3W);
```

x is a numpy array with shape [1,3] containing 3 input features, the layer?W numpy are as described above. layer3Out will be a single number for the **output from layer 3 r_1^3 .**

Use of matrix mathematics and/or the dot command may help you in this function. You also may use a sigmoid function you wrote for a previous homework.

Note, the following set of commands will get the following output in Python:

```
x=[1, 2, 3];
lay1w=np.array([[2, 1, 0, 1],[ 0 2, 1, 0]]);
lay2w=np.array([[0, -2, 0],[ -1 0, 0]]);
lay3w=np.array([1,-1,0]);
out=feedforward (x,lay1w,lay2w,lay3w);
% out will be 0.4624
```

Now, let us consider a **recurrent** neural network, where the output of each neuron is determined by the weighted sum of the inputs from the previous layer **and** the output of the neuron from the previous time step:

$$r_k^{m,t} = g\left(\sum_i w_{k,j}^m r_j^{m-1,t} + b_k^m + w_{k,past}^m r_k^{m,t-1}\right)$$

Note the output of neuron k in layer m at time t is denoted as $r_k^{m,t}$ – it depends on the inputs from the neurons at the previous layer at the same time t: $r_j^{m-1,t}$. The output **also** depends on the previous output from the same neuron at time point t-1 $r_k^{m,t-1}$. For each neuron, the weights stay the same at every time step – it is $w_{k,j}^m$ not $w_{k,j}^{m,t}$. Thus, the **only** change from our model in lecture is the addition of the neuron's previous output: $+w_{k,past}^m r_k^{m,t-1}$

Note, we assume $r_k^{m,t=0} = 0$ for all neurons and layers at time t=0.

2. Write a function **feedforwardRecurrent** that takes in the features **x**^{i} at two or more time steps and the network weights, and outputs the responses from layer 3 at each time step.

Specifically: you will be able to call the function as

```
  layer3Out=feedforwardRecurrent(x,layer1W,layer2W,layer3W);
```

`x` is a [t, 3] numpy array containing t rows of input features (one for each time point), each with 3 columns of input features. As before, you will represent the weights for each layer in a separate numpy array – layer1W, layer2W, layer3W. Each row will correspond to the weights for a single unit. So, layer1W will have shape [2, 5] – 2 units each taking in inputs from 3 features **plus** the output from unit k at time t-1 **plus** the constant b offset, e.g., **[$w_{1,1}^1$, $w_{1,2}^1$, $w_{1,3}^1$, $w_{1,PAST}^1$, b_1^1]**. layer3Out will be a numpy array of t numbers for the **output from layer 3 r_1^3 at each time point.**

Note, the following set of commands will get the following output in Python:

```
xMat=np.array([[1, 2, 3],[2, 0, 1],[ 3, 1, 0]]);
lay1w=np.array([[2, 1, 0, -2, 1],[ 0 2, 1, 0, 0]]);
lay2w=np.array([[0, -2, -1, 0],[ -1 0, 1, 0]]);
lay3w=np.array([1,-1,3,0]);
```

```
out=feedforwardRecurrent(x,lay1w,lay2w,lay3w);
% out will be 0.4624, 0.7724, 0.891
```

Note also that we assume the output of all neurons is 0 at time t=-1 (the time point prior to any input being fed in).

Accessing our data
For questions 3-4, the file mnist_train.csv is available on our website (and on erdos using `cp ~dleeds/MLpublic/mnist_train.csv .`) It contains pixel representations of hand-written digits. `mnist_train.csv` can be loaded with the code:

```
  import csv
  import numpy
  ~~reader = csv.reader(open("mnist_train.csv", "rb"), delimiter=",")~~
  reader = csv.reader(open("mnist_train.csv", "rt",encoding='utf8'), 
delimiter=",")
  x = list(reader)
  digits = numpy.array(x).astype("float")
```

Each row contains information for a single digit. The first column has a value between 0 and 9 to indicate the digit. The remaining columns are 784 pixels corresponding to a 28x28 grid rendering of the hand-written digit. You can view a digit yourself by "reshaping" the vector of pixel values into a 28x28 grid with the code:

```
  digitPic= digits[n,1:].reshape((28,28))
```

This picture can be displayed with the code:

```
  import matplotlib.pyplot as plt
  plt.imshow(digitPic)
  plt.show()
```

We are given 40 components **u** , also referred to as "non-negative factors". They are provided in `hw3NNfactors.mat` in the matrix `nnFactors`. (See previous homeworks to review how to load .mat files into Python.) Since these factors are non-orthogonal, computing optimal component weights z is non-trivial. We will try several approaches to compute component weights z.

3. Write a function **findWeights3** that takes in the features from a single data point **x**^{i} and the components **u** and returns the weights **z** using the technique specified below

Specifically: you will be able to call the function as

```
  z=findWeights3(x,uMat);
```

`x` is a [1, 784] numpy array containing pixel values for one number-picture and uMat is a [784,40] numpy array containing a component/factor **u** in each column. The output will be a [1,40] numpy array of **u** weights.

Method for computing z: Use the PCA approach for computing z, $z_q^i = \boldsymbol{u}^{q^T}\boldsymbol{x}^i$

4. Write a function **findWeights4** that takes in the features from a single data point **x**^{i} and the components **u** and returns the weights **z** using the technique specified below

Specifically: you will be able to call the function as

```
  z=findWeights4(x,uMat);
```

`x` is a [1, 784] numpy array containing pixel values for one number-picture and uMat is a [784,40] numpy array containing a component/factor **u** in each column. The output will be a [1,40] numpy array of **u** weights.

Method for computing z:

```
  Initialize all z to 0
  Loop 40 times OR UNTIL highest $z^q<0$
     Find $u^q$ with highest dot product with $\mathbf{x}$
     Save this highest magnitude product in $z^q$
     Remove component $\mathbf{u}^q$ from data point: $\mathbf{x} \leftarrow \mathbf{x} - z^q \mathbf{u}^q$
     Do not consider this $\mathbf{u}^q$ in any future loop
```

5. Write a function **xEstimate** that takes in the NMF z weights a single data point **x**^{i} and the factors/components **u** and returns the reconstructed/"estimated" original data point **x**^{i}

Specifically: you will be able to call the function as

```
  x=xEstimate(z,uMat);
```

`z` is a [40, 1] numpy array containing pixel values for one number-picture and uMat is a [784,40] numpy array containing a component/factor **u** in each column. The output will be a [784,1] numpy array of pixel values for the estimated data point **x**.

6. Report mean-square reconstruction error on the mnist data based on the z weights learned from findWeights3 and based on the z weights learned from findWeights4 . For this question alone, you can use any library/package you want to compute the reconstruction errors.

Report the mean-square error for findWeights3 and findWeights4 as comments in hw3.py

For questions 1-5, you may use the numpy commands addressed in the online resources slide, in this homework, and in all previous homeworks. Additionally, you may use any of the numpy commands listed below. If the numpy command you want to use is **not** on any of these lists, you will have to implement the command yourself.

```
absolute  array    concatenate   copy     dot
exp       mean     max           min      multiply
```

```
ones      power     reshape     shape     sqrt
std       T         where       zeros
```