Homework 3

Parts A and B due in class April 11, 2018

Part C due online by noon April 13, 2018

A. Component analysis:

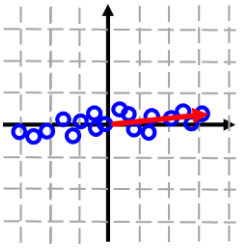
1. Unit vectors have magnitude of 1. Convert the following vectors in (a), (b), and (c) to be unit vectors, each pointing in the direction of the original vector. (For example, $\begin{bmatrix} 2 \\ -4 \end{bmatrix}$ and $\begin{bmatrix} 8 \\ -16 \end{bmatrix}$ are both **non-unit** vectors and both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

(a) $\begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix}$ (b) $\begin{bmatrix} -4 \\ 5 \\ 1 \end{bmatrix}$ (c) $\begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$

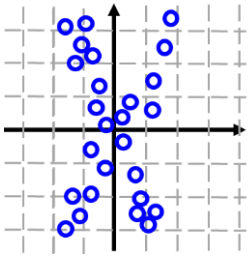
2. For each of the following data sets (set A and set B), provide the top two principal components and the top one or two independent components (if there are two clear independent components, you must provide both). Express each component as a 2-element vector $\begin{bmatrix} num1 \\ num2 \end{bmatrix}$.

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

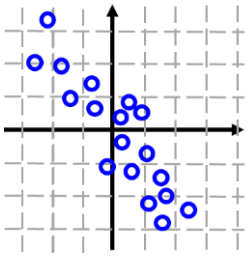
Roughly estimated as: $\boldsymbol{dir} \approx \begin{bmatrix} 3.0 \\ 0.5 \end{bmatrix}$ or $\boldsymbol{u} \approx \begin{bmatrix} 1.0 \\ 0.2 \end{bmatrix}$



Set A:



Set B:



3.

Review:

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

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

where $\tilde{\boldsymbol{x}}$ 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 $\tilde{\boldsymbol{x}}$ is called "mean squared error":

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

Data for Question 3:

We have the following components:

$$\boldsymbol{u}^1 = \begin{bmatrix} 0.4 \\ 0 \\ 0 \\ 0.9 \end{bmatrix}, \qquad \boldsymbol{u}^2 = \begin{bmatrix} 0 \\ 0 \\ 0.6 \\ -0.8 \end{bmatrix}, \qquad \boldsymbol{u}^3 = \begin{bmatrix} 0 \\ -0.7 \\ -0.2 \\ 0.7 \end{bmatrix}$$

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

$$\boldsymbol{x^1} = \begin{bmatrix} 0.4 \\ 1.5 \\ 0.7 \\ -1.0 \end{bmatrix} \qquad z_1 = 0.5 \quad z_2 = 0 \quad z_3 = -2.5$$

$$\boldsymbol{x^2} = \begin{bmatrix} 0.1 \\ -1.3 \\ -0.7 \\ 1.3 \end{bmatrix} \qquad z_1 = 0 \quad z_2 = 0 \quad z_3 = 1.5$$

Actual questions:

(a) What are the estimated vectors for $\mathbf{x}^1$ and $\mathbf{x}^2$, based on the corresponding *z* values above?

(b) What is the mean squared error between the estimated and actual data vectors for $\mathbf{x}^1$ and for $\mathbf{x}^2$?

(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 principal components:

$$\boldsymbol{u}^1 = \begin{bmatrix} 0 \\ 0.67 \\ 0.67 \\ 0.33 \end{bmatrix}, \qquad \boldsymbol{u}^2 = \begin{bmatrix} 0.41 \\ -0.41 \\ 0 \\ 0.82 \end{bmatrix}, \qquad \boldsymbol{u}^3 = \begin{bmatrix} 0.88 \\ 0 \\ 0.22 \\ -0.44 \end{bmatrix}$$

For each data point below, what are the three component weights z_q^i?

$$\boldsymbol{x}^1 = \begin{bmatrix} 0.9 \\ -4.3 \\ -3.4 \\ 0.2 \end{bmatrix} \qquad \boldsymbol{x}^2 = \begin{bmatrix} 0.5 \\ 1.1 \\ 1.6 \\ 1.8 \end{bmatrix} \qquad \boldsymbol{x}^3 = \begin{bmatrix} 0.8 \\ 3.2 \\ 4.0 \\ 3.6 \end{bmatrix}$$

B. Neural networks

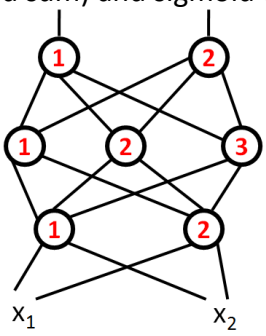
1. Let us assume we have a neural network with three layers. Layer 1 has 5 units, layer 2 has 3 units, and layer 3 has a 2 units. There are 10 features fed into the units in layer 1.

(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 $(r_1^{3,i}\text{-}y^i)^2$. 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 remove one unit to layer 2 and the log likelihood increases by 2, how will the AIC be affected? (Note AIC uses the natural logarithm, $\log_e$.)

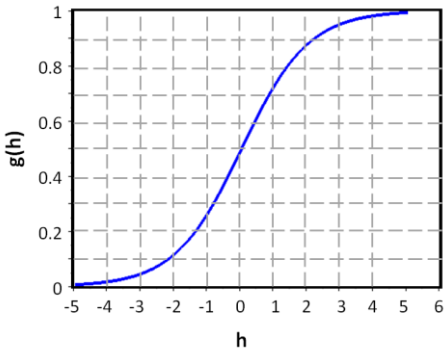
2. Presume the following Neural Network, where each unit performs the standard dot product (weighted sum) and sigmoid transformation.



Layer 3

Layer 2

Layer 1



The following feature values are input: x_1=0.5 x_2=0.8

The original weights are: $w^1_{1,1} = -4, w^1_{1,2} = 0, \quad w^1_{2,1} = 2, w^1_{2,2} = 4$ Layer 1

$w^2_{1,1} = -5, w^2_{1,2} = 2, w^2_{2,1} = 0, w^2_{2,2} = -1, \mathbf{w^2_{3,1} = 10, w^2_{3,2} = -5}$, Layer 2

$w^3_{1,1} = 0, w^3_{1,2} = 2, w^3_{1,3} = -4, \qquad w^3_{2,1} = 4, w^3_{2,2} = 0, w^3_{2,1} = -6$

Assume all b's are 0

(a) Compute the outputs of the 7 units given the input x_1=0.5 x_2=0.8

(b) Weight learning:

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

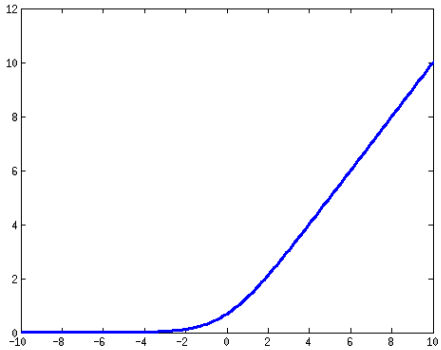
unit 1 output: 0 unit 2 output: 1

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

$w^3_{1,1} =$ $w^3_{2,1} =$ $w^2_{1,1} =$

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

$$g(h) = g(r; w) = \ln(1 + e^{\mathbf{w}\cdot\mathbf{r}})$$



We wish to calculate a new w update rule to minimize the error in the output for neuron 2 in layer 3 r^3_2 compared to the desired output y_2 :

$$E(y; w) = \sum_j \left(y^i - g(r^i; w)\right)^2$$

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

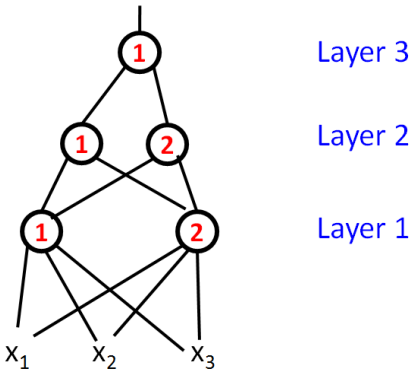
C. Programming

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/HW3 directory. For matlab, save code as feedforward.m, feedforwardRecurrent.m, dataProjection.m, componentRemoval.m, findWeights; for Python save as hw3.py .

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 matrix – layer1W, layer2W, layer3W. Each row will correspond to the weights for a single unit. So, layer1W will have dimensions 2x4 – 2 units each taking in inputs from 3 features **plus** a constant b offset.



1. Write a function **feedforward** that takes in the features $\mathbf{x}^i$ 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 vector containing 4 input features, the layer?W matrices are as described above. layer3Out will be a single numbers 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, or that is already available for Matlab/Python.

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

```
x=[1, 2, 3];
lay1w=[2, 1, 0, 1; 0 2, 1, 0];
lay2w=[0, -2, 0; -1 0, 0];
lay3w=[1,-1,0];
out=feedforwardRecurrent(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 $\mathbf{x}^i$ 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 x 3 matrix 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 matrix – layer1W, layer2W, layer3W. Each row will correspond to the weights for a single unit. So, layer1W will have dimensions 2x5 – 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 vector 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 Matlab:

```
xMat=[1, 2, 3; 2, 0, 1; 3, 1, 0];
lay1w=[2, 1, 0, -2, 1; 0 2, 1, 0, 0];
lay2w=[0, -2, -1, 0; -1 0, 1, 0];
lay3w=[1,-1,3,0];
out=feedforwardRecurrent(x,lay1w,lay2w,lay3w);
% out will be 0.4624, 0.7724, 0.891
```

INDEPENDENT COMPONENT ANALYSIS

Accessing our data

For questions 3-6, the file emnist_letters.mat is available on our website (and on erdos using `cp ~dleeds/MLpublic/emnist_letters.mat .`) It contains pixel representations of hand-written letters, as well as corresponding independent components.

Each row of `letterPics` contains information for a single digit. The columns are 784 pixels corresponding to a 28x28 grid rendering of the hand-written letter. You can view a letter yourself by "reshaping" the vector of pixel values into the grid through the Matlab command:

```
letter=reshape(letterPics(n,:),[28 28]);
```

for the n^{th} digit data point. You can view the letter picture through the Matlab command:

```
imagesc(letter)
```

(For python, a single letter can be converted into a 28x28 grid with the code:

```
letter= letterPics[n,:].reshape((28,28))
```

This picture can be displayed with the code:

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

)

Each column of `ICs` contains the elements of a single independent component $\mathbf{u}^q$. So, `ICs(:,10)` contains the 784 entries corresponding to the tenth independent component.

ICA coordinate assignment

In class, we explained how PCA coordinates z_q^i could be found by taking the projection of the selected data point $\boldsymbol{x}^i$ onto the selected component $\boldsymbol{u}^q$. For ICA, the process is more complicated:

Given a data point $\boldsymbol{x}^i$ and a set of independent components $\boldsymbol{u}^q$, we

1. Identify which component has the **largest magnitude** projection value onto the data point.
$$q' = \underset{q}{\operatorname{argmax}}(\boldsymbol{u}^q)^T \boldsymbol{x}^i$$
2. Record the projection (weight) value of this component:
$$z_{q'}^i = (\boldsymbol{u}^{q'})^T \boldsymbol{x}^i$$
3. Subtract the component $\boldsymbol{u}^{q'}$ from the data point $\boldsymbol{x}^i$:
$$\boldsymbol{x}^{i,NEW} = \boldsymbol{x}^i - z_{q'}^i \boldsymbol{u}^{q'}$$
4. We now loop back to 1, **replacing** $\boldsymbol{x}^{i,NEW}$ for $\boldsymbol{x}^i$

We continue this loop until a coordinate z for every component **u** is assigned. For example, if there are 20 components, you will loop 20 times.

3. Write a function called `dataProjection` that takes in the vector of *n* pixel data points from a single digit $\mathbf{x}^i$ and the *n*-element vector for the selected independent component $\mathbf{u}^q$ and returns the single weight z_q^i. (For our letter data, *n=784* .) The syntax will be:

```
z = dataProjection(dataVector,component);
```

Note, the following set of commands (on very simple data) will get the following output in Matlab:

```
data=[2, -4, 5, 5, 0];
u1=[0.3, -.79, 0.2, 0.5, 0];
z = dataProjection(data,u1);
% z will be 7.26
```

4. Write a function called `componentRemoval` that takes in a data vector $\mathbf{x}^i$, a component $\mathbf{u}^q$, and the corresponding weight z_q^i and returns the updated data vector $\mathbf{x}^i$ with the component $\mathbf{u}^q$ removed. The syntax will be:

```
xNew = componentRemoval(xOld,component,z);
```

Note, the following set of commands (on very simple data) will get the following output in Matlab:

```
data=[2, -4, 5, 5, 0];
u1=[0.3, -.79, 0.2, 0.5, 0];
z=7.26;
dNew = componentRemoval(data,u1,z);
% dNew will be roughly [-0.17 1.74 3.55 1.37 0]
```

5. Write a function `findWeights` that take in the *n* pixel data points from a single digit $\mathbf{x}^i$ and the unit vectors for all independent components $\mathbf{u}^1$ … $\mathbf{u}^Q$ and returns the vector of weights z_1^i … z_Q^i . The matrix of independent components must contains *Q* rows (one row per component) and *n* columns (one column per pixel). The syntax will be:

```
allZs = findWeights(dataVector,components);
```

```
data=[2, -4, 5, 5, 0];
uMat=[0.3, -.79, 0.2, 0.5, 0; 0, -0.6, 0.6, -0.4, .35; 0, .45,
.77, .45, 0];
z=2.26;
zVec = findWeights(data,uMat);
% zVec will be roughly [7.26 0.49 4.13]
```

6. Try reconstructing a few letters using the weighted sum of independent components you have extracted. How many components are needed for you to recognize each letter (for a good-quality reconstruction of the original data)? You can judge quality by eye, or you are welcome to compute mean squared error. Include your answer as a comment in `findWeights.m` or `hw3.py`.