Neural Networks Report #2

Introduction

This homework has two problems, about PCA of some data distributed in 3 dimensions using 3 different probability functions and the model of neurons performance that is based on differential equations. I will give either written solution for each part or some description of the code with brief analysis of the result. I again used Jupyter notebook for coding.

Problem 1(a)

Here making the distribution of data was very similar to homework 1. But here we have 3 dimensions. We van use the fact the $\oint pdV = 1$, where p is probability distribution function and V is the volume enclosed by all points. SO if we have cylinders with area pi and length 1 then its volume is pi, multiplying by probability p = 1/pi we het 1. The same we get if we take the random distribution function p = 1/V i.e. use random function call in the region of some numbers. To cover all region we need cube with -1< x,y,z < 1 and then take the values that are inside the cylinder. Ones 200 points are generated, we are done. If the coordinates of center of cylinder are moved to some position as was in 2nd and 3rd samples, we simply move the cube coordinates.

The eigenvectors were calculated using powerful library sklearn that has a class PCA that can find eigenvalues and eigenvectors by one function call. However, to do this we need a matrix of data with correct representation of rows and columns. Each row there is coordinates of individual point. This is simply done by for loop. Each data is fed, function finds the result , 3 PCA components and eigenvalues are shown there.

For simplicity, all code parts of problem1 are solved in one notebook

Problem 1(b)

Below is my written solution.

Problem 1 (b)

We need to maximize variance of data with respect to coordinates

$L = \sum_j (s_j)^2$

where s_j is variance

$s_j^2 = \frac{(x_j - \mu)}{n-1}$ where μ is average

$\mu = \sum_{i=1}^{n} \frac{x_i}{n}$

assume we have data coordinates in matrix X so that each row is coordinate of single point, if PCA 1 has eigenvector w then coordinates $\overset{\text{of all points}}{\text{w.r. to}}$ PCA 1 are $w^T X$

so

$$\mathcal{L} = \sum \frac{(x_j - \mu)^2}{n-1} = \frac{(w^T X - \overset{\text{mean vector}}{\bar{X}})(w^T X - \bar{X})^T}{n-1}$$

$$= \frac{(w^T X)(w^T X)^T - \bar{X}(\ldots) - (\ldots)\bar{X}^T}{n-1}$$

for simplicity we can place coordinates such that mean of data is located at the origin (i.e. vector X is with respect to mean coordinates, in fact it doesn't changes the outcome) so $\bar{X} = 0$ & we can ignore $\frac{1}{n-1}$ as it is constant, doesn't change the loss maximization.

$\mathcal{L}_o = (w^T X)(w^T X)^T = w^T X X^T w = w^T R w$

where R is correlation matrix $R = \begin{bmatrix} R_{11} & R_{\cdot 2} \\ R_{21} & \ddots \\ \vdots & \end{bmatrix}$

$\mathcal{L}_o = \sum_{i,j} w_i R_{ij} w_j$

$w_{\cdot} = \text{argmax}\, \mathcal{L}_o(w_{\cdot}, R)$

If we use iterative approach, we go along increasing slope of $\mathcal{L}_o$

$w_{\cdot}(n+1) = w_{\cdot}(n) + \eta \frac{\partial \mathcal{L}_o}{\partial w}$ ← General iterative approach

→ for each comp

$\frac{\partial \mathcal{L}_o}{\partial w_j} = \frac{\partial}{\partial w_j}\left[w_1 R_{11} w_1 + w_1 R_{12} w_2 + \dots + w_2 R_{21} w_1 + \dots \right]$

$$= \frac{d}{dw_j}\left[\sum_{i,j} 2 w_i R_{ij} w_j + R_{jj} w_j^2 + R_{ii} w_i^2\right]$$

$$= 2 \sum_i w_i R_{ij}$$

Another notation

$$R_{ij} = \sum_s X_{is} \cdot (X^T)_{si} = \sum_s X_{is} \cdot X_{js} = \sum_s x_i^s x_j^s$$

$$\frac{dR_e}{dw_j} = 2 \sum_i w_i \sum_s x_i^s x_j^s = 2 \sum_s \left(\sum_i w_i x_i^s\right) x_j^s$$

$$= 2 \sum_s y^s x_j^s$$

Last step is because dot multiplying one data component by weight we get its activation y ———— (since it is w_i, not w_j)

Next, we need to do it for each weight

$$2\eta = \eta' = \eta$$

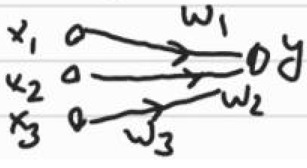
$$w_i(n+1) = w_i(n) + \boxed{2\eta} \sum_s y^s x_i^s$$

However in this NN weight is unit vector so it always should be normalised.

$\|W\| = 1$ To normalise, devide it by its magnitude.

→ sum of all points coordinate x_i & active. y^s

for simplicity $\sum_s y^s x_i^s = y x_i$



$$w_i(n+1) = \frac{w_i(n) + \eta\ y\ x_i}{\left(\sum_{j=1}^{M}[w_j(n) + \eta\ y(n)\ x_j(n)]^2\right)^{1/2}}$$

$$= \frac{w_j(n) + \eta \Sigma y^s x_j^s}{\left(\sum_{j=1}^{M} w_j^2 [\ 1 + \frac{\eta \Sigma y^s(n)\ x_j^s(n)}{w_j(n)}\ ]^2\right)^{1/2}}$$

We can state then $\eta \frac{y(n)\ x_j(n)}{w_j(n)} << 1$ because learning step is small, ~~[illegible]~~ ~~should be [illegible]~~ y & x coordinates are concentrated mostly near mean value (origin), so we use binomial approximation

$(1 \pm x)^n \approx 1 \pm n x$ if $x << 1$

$$w_i(n+1) = \frac{w_i(n) + \eta y(n) x_i(n)}{\left(\sum_{j=1}^{M} w_j^2(n) \left(1 + \frac{2\eta\ y(n) x_j(n)}{w_j(n)}\right)\right)^{1/2}}$$

Since $w(n)$ is normalised $\sum_1^k w_j(n) = 1$

$$w_i(n+1) = \frac{w_i(n) + \eta y(n) x_i(n)}{\left(\sum_{j=1}^{m} w_j^2(n) + 2 w_j(n) \eta y(n) x_j(n)\right)^{1/2}}$$

$$= \frac{w_i(n) + \eta y(n) x_i(n)}{\left(1 + \sum 2 w_j(n) \eta y(n) x_j(n)\right)^{1/2}} \xrightarrow{<<1} \text{as previously}$$

$$\approx \left[w_i(n) + \eta y(n) x_i(n)\right]\left[1 - \frac{\cancel{2}}{\cancel{2}} \eta y(n) \sum_{j=1}^{k} w_j(n) x_j(n)\right]$$

again $\sum w_j x_j = y$

$$w_i(n+1) = \left[w_i(n) + \eta y(n) x_i(n)\right]\left[1 - \eta y^2(n)\right]$$

$$= w_i(n) + \eta y(n) x_j(n) - w_i \eta y^2(n) - \eta^2 y(n)^3 x_i(n)$$

the last therm contains x_i, η^2 & y^3, all of them are very small so we can simply ignore it

$$w_i = w_i(n) + \eta\left(y(n) x_i(n) - y^2(n) w_i(n)\right)$$

$$w_i(n+1) \approx w_i(n) + \eta y(n) x_i'(n)$$

where $x_i'(n) = x_i(n) - y(n) w_i(n)$ $\blacksquare$

Note that in 3d case we have w = [w1, w2, w3].T so we make update for every component independently.

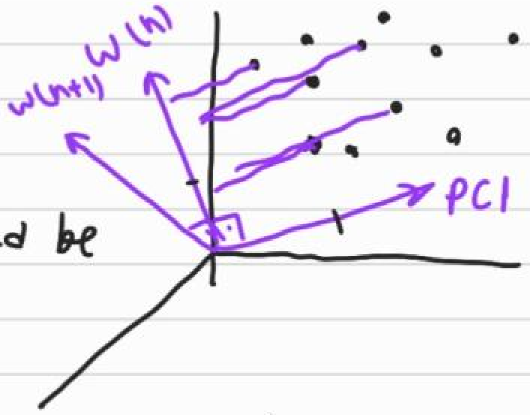
Problem 1(c)

The code is very simple. Initially create the matrix of data, note that it is made with respect to mean. For simplicity, its rows are data point coordinates. Weights are initialized in w list randomly, but then normalized so its magnitude is 1. Next, parameter epoch is 1000 and etha is 0.1. Loop for epoch, loop for each data, find the activation of made by each point, update weight with this activation and points coordinates. Second loop is done as it is online learning. The result vector at least seems to be component. However, it is not the same result as was fount in problem 1b. Possibly my result is incorrect or contains some noise, however all the equations are used seem to be precise

Problem 1(d)

Solution is below

, Problem 1 d



PC2 which I will express as w should be orthogonal to PC1

i.e $w \cdot PC1 = 0$ so we will find its components with respect to PC_1 while ignoring the distances of points that are along PC1 vector

create a plane with normal PC_1 so new coordinates are $x' = x - (PC1 \cdot x) PC1$

respect to data located in plane find vector w that maximizes variance.

component x of X is now $x' = x - (PC1 \cdot x) PC_1$

and $\mathcal{L} = \sum_{i,j} w_i \cdot R_{ij} w_j$

$$\frac{\partial \mathcal{L}}{\partial w_j} = 2 \sum_s y^s x_j'^s$$

$$w_i(n+1) = w_i(n) + \eta y \overbrace{(x_i - (PC1 \cdot x) PC1_i)}^{x'}$$

as it is the same equation as was for (c) but with x' we just follow the same steps & reach

$$w_i(n+1) = w_i(n) + \eta y(n) x_i^*(n)$$

$$x_i^*(n) = x_i(n) - \left(\sum_j x_j(n) P(j)\right) \cdot P(i) - y(n) w_i(n)$$

Problem 1(e)

This part is done very similarly as (c) part but inside the update loop, each point coordinate was projected into the plane with PC1 normal, i.e. every component of coordinate vector along PC1 was deleted so w should be orthogonal to PC1. Other part is same as in (c). Result again seems to be satisfying, but is different from result of (b) part.

Problem 2(a)

You can see solution below

Problem 2a

$\tau \frac{du(t)}{dt} = -u(t) + Ri(t)$

Assume $RI(t) = V_i = const$

$\frac{du}{dt}(t) + \frac{u(t)}{\tau} = \frac{V_i}{\tau}$ ①

$u(t) = u_h(t) + u_p(t)$

where $u_h(t)$ is homogeneous solution

$\frac{du_h(t)}{dt} + \frac{u_h(t)}{\tau} = 0$ ②

The fuction which derivative doesn't change is e^t. so assume $u_h(t) = Ce^{kt}$

substituting into ②

$\cancel{C}k\cancel{e^{kt}} + \frac{1}{\tau}\cancel{C}\cancel{e^{kt}} = 0 \rightarrow k = \frac{-1}{\tau}$

so $u_h(t) = Ce^{-t/\tau}$

To find particular solution we see that $u'_p(t) + \frac{1}{\tau}u(t) = \frac{V_i}{\tau}$ ③ answer is constant so assum $u_p = A$ putting into $\textcircled{3}$

$0 + \frac{A}{\tau} = \frac{V_i}{\tau} \rightarrow A = V_i$

So $u(t) = V_i + Ce^{-t/\tau}$

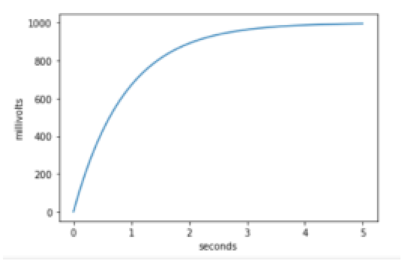
to find C we use initial conditions

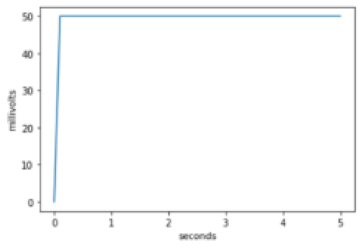
$u(0) = V_i + Ce^{0} = 0 \rightarrow C = -V_i$

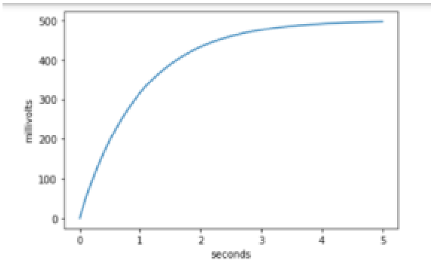
So $u(t) = V_i(1 - e^{-t/\tau})$

Problem 2(b)

I will make the Problem 2(b),(c) and (d) in one python notebook as they need the same code parts that will be defined via functions in the # Helper functions cell. Making the function of u(..) is very simple, it takes Vi, tau parameters and time as input and returns the u value. u_vs_t(..) takes constants and producoes graph upon values of every 0.1 seconds. Making arbitrary values of V and tau is tricky for me as I don't know what are the actual constants range in human brain. I assume the voltage is in millivolts and time in seconds. The values that I make are just arbitrary by my intuition, however they still give a good plot which shows that it reaches some maximum value with different speed which is good with real behavior.







Problem 2(c)

Solution is below

$u(t) = V_i(1-e^{-t/\tau})$

The time when potential difference reaches θ is

$\theta = V_i(1-e^{-t_0/\tau}) \rightarrow t_0 = -\tau \ln\left(1-\frac{\theta}{V_i}\right)$

Because voltage becomes zero after the spike, the process is periodic with perion t_0. So one spike happens in t_0.

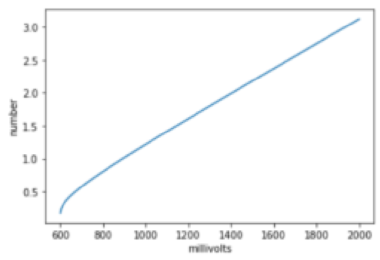
$$\frac{\text{number of spikes}}{\text{unit time}} = \frac{N}{t} = \frac{1}{t_0}$$

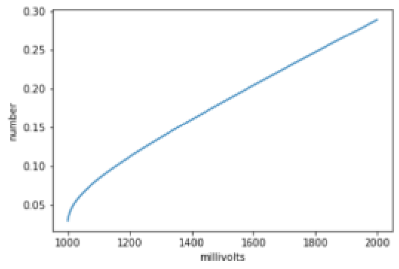
$$\frac{\#\text{ of spikes}}{\text{unit time}} = \frac{1}{-\tau \ln\left(1-\frac{\theta}{V_i}\right)}$$

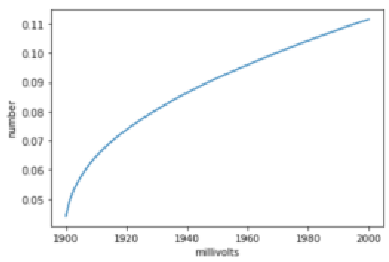
Prob 2(C)

Problem 2(d)

This is more challenging case. Using previous section solution, I develop function num_per_sec(..) which takes Vi constant, tau and threshold and returns the value. Hard thing that there is condition that Vi>threshold so when graph is drawn, Vi should start from greater than threshold value and increase furthermore. That’s why function that draws the graph num_vs_v needs to take also initial value of Vi. 3 results are below. It is seen that the it is seen that if threshold is small, the shape is very similar to the linear curve. And curvier when the threshold is larger.

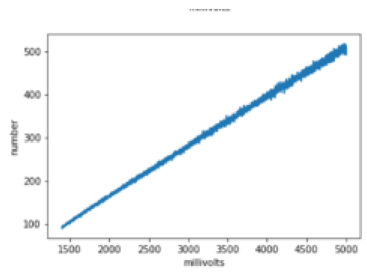


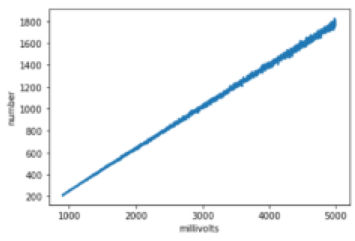


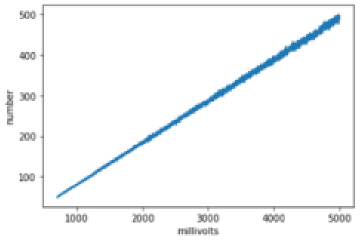


Problem 2(e)

As there are 200 neurons we simply ass the number of spikes of each. The function num200_vs_v works similar to num_vs_v as it also has v0 that should be larger than treshold. Distributions are created using gauss function and since treshold is different for each, while drawing the graph v0 should be chosen to be at least 2 larger than mean so it is less probable that threshold will be larger, but code still can throw exception sometimes, just run it again. Results below show that the larger tau and threshold value, the more noisy result is and now it is not resembles ln anymore as noise disturbs the shape.







Conclusion

In conclusion I would like to say that this homework took a lot of time, however helprs me to understand the underneath of neural networks. Some of my code works not as expected, so I will appreciate for the feedback about what did I do wrong. Thanks