Neural Networks
Lab 4

EECS 738

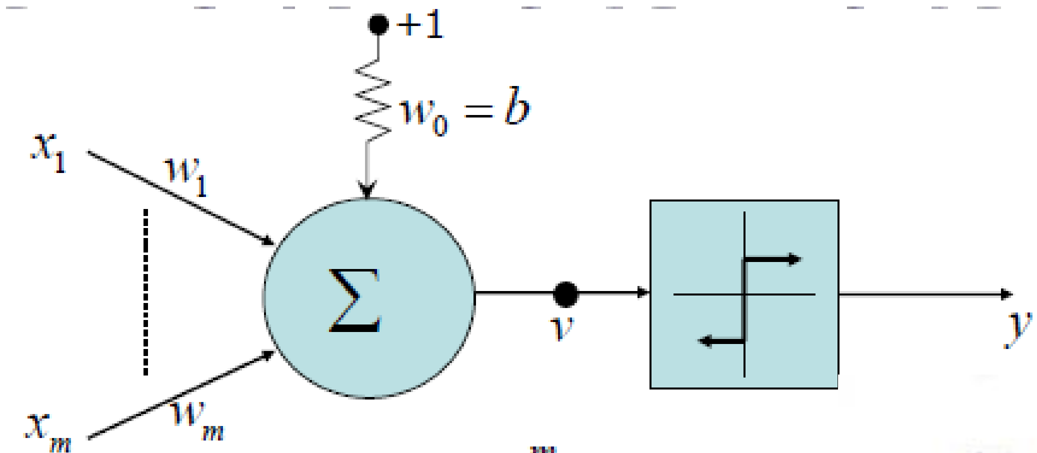
Submission Requirements

- No need for a .txt file
- Write a report, PDF, and explain your results
- Please include all of the steps of the lab in one .py or .ipynb file
- You are allowed to use all of the libraries that were used in the previous labs
- You may also use “statsmodels.formula.api” (although you might not need it)
- You may use Scikit-learn libraries except scikit-neuralnetwork or any other library that directly implements NN without the learning cliff!
- Construct the confusion matrix for ALL of the test sets you use whether it is the original test data or a sample of data set.

Perceptron Learning Rule

- **The Perceptron** is presented by Frank Rosenblatt (1958, 1962)
- **The Perceptron**, is a feedforward neural network with no hidden neurons. The goal of the operation of the perceptron is to learn a given transformation using learning samples with input x and corresponding output y = f (x).
- The perceptron output is limited to either 1 or –1.

Perceptron Network Architecture



$$v = \sum_{i=0}^{m} x_i w_i$$

$$y = f(v) = \begin{cases} 1 & if\ v > 0 \\ -1 & if\ v < 0 \end{cases}$$

- The update of the weights at iteration n+1 is:

$W_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$

Since:

$$\Delta w_{kj}(n) = \eta(d_k - y_k)x_j(n)$$

Limit of Perceptron Learning rule

- If there is no separating hyperplane, the perceptron will never classify the samples 100% correctly.
- But there is nothing from trying. So we need to add something to stop the training, such as:
    - Include an **error bound**. The algorithm can stop as soon as the portion of misclassified samples is less than this bound. This ideal is developed in the Adaline training algorithm.

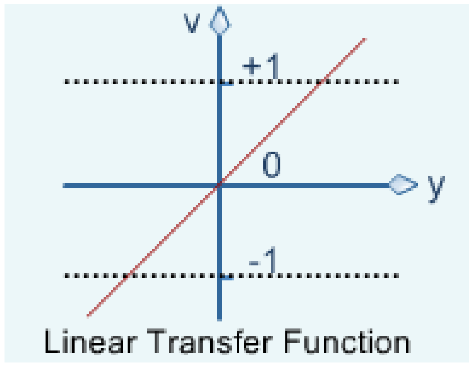
Error Correcting Learning

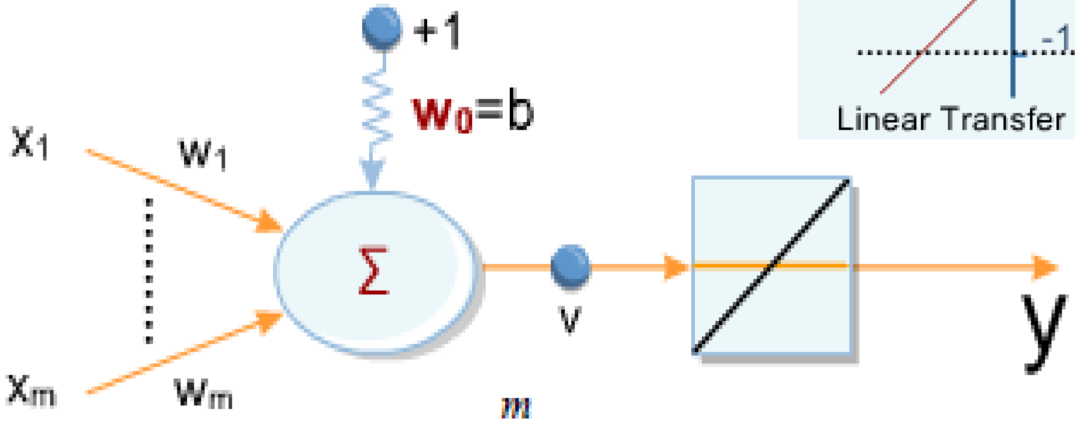
- The objective of this learning is to start from an arbitrary point error and then move toward a global minimum error, in a step-by-step fashion.
  - The arbitrary point error determined by the initial values assigned to the weights.
- Examples of error-correction learning:
  - the **least-mean-square (LMS) algorithm** (Windrow and Hoff), also called **delta rule**
  - and its generalization known as the **back-propagation (BP) algorithm.**

Adaline (Adaptive Linear Neuron) Networks

- **1960** - **Bernard Widrow** and his student **Marcian Hoff** introduced the ADALINE Networks and its learning rule which they called the Least mean square (LMS) algorithm (or Widrow-Hoff algorithm or delta rule)
- The Widrow-Hoff algorithm
  - can only train single-Layer networks.
- Adaline similar to the perceptron, the differences are ….?
- Both the Perceptron and Adaline can only solve linearly separable problems
  - (i.e., the input patterns can be separated by a linear plane into two groups, like AND and OR problems).

Adaline Architecture





$$v = \sum_{i=0}^{m} x_i w_i$$

$$y = f(v) = v$$

- Given:

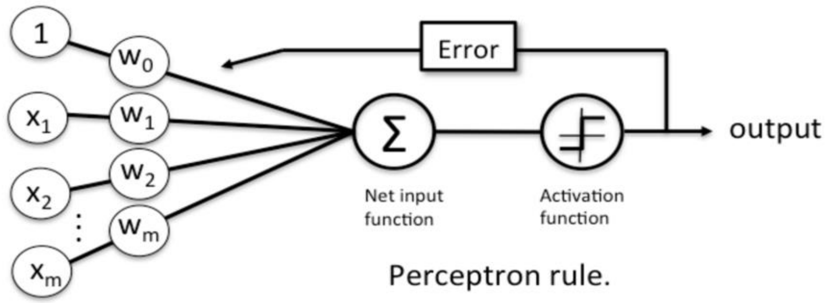
  $x_k(n)$: an input value for a neuron k at iteration n,

  $d_k(n)$: the desired response or the target response for neuron k.

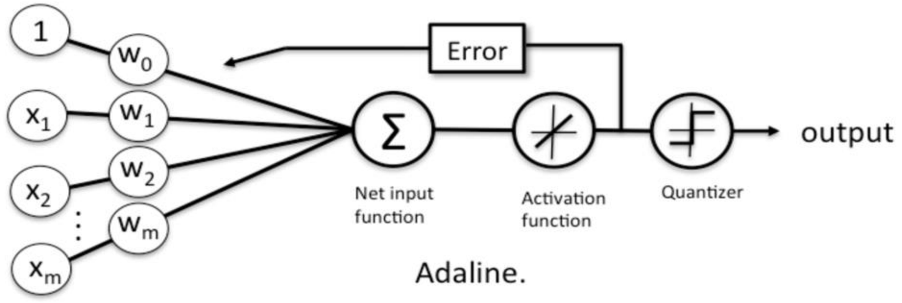
- Let:

  $y_k(n)$: the actual response of neuron k.

Adaline vs Perceptron



Perceptron rule.



Adaline.

Adaline’s Learning as a Search

- Supervised learning
- The task can be seen as a search problem in the weight space:
  - Start from a random position (defined by the initial weights) and find a set of weights that minimizes the error on the given training set

The error function: Mean Square Error

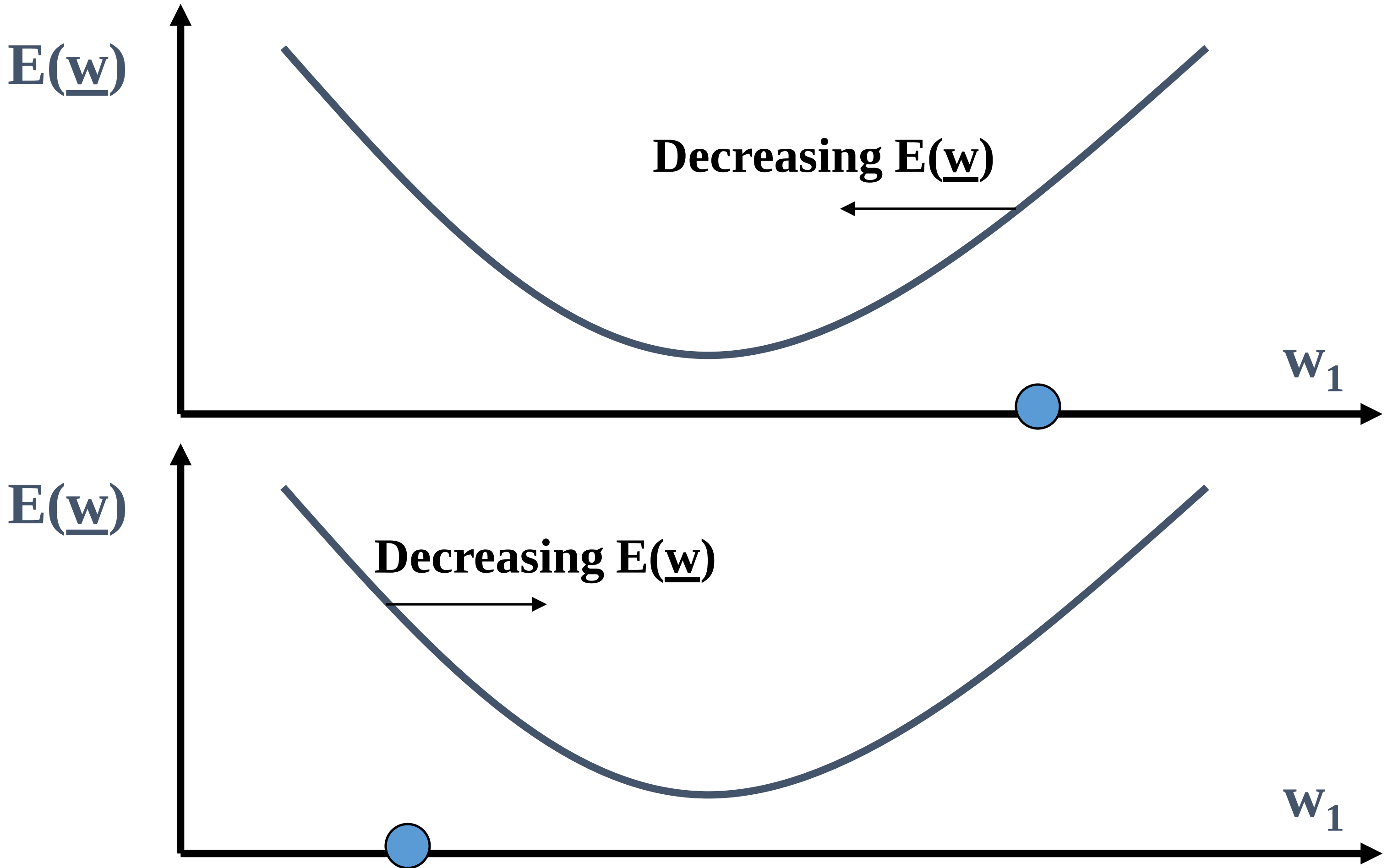
- ADALINEs use the Widrow-Hoff algorithm or Least Mean Square (LMS) algorithm to adjusts the weights of the linear network in order to minimize the mean square error
- Error : difference between the target and actual network output (delta rule).

  error signal for neuron k at iteration n: $e_k(n) = d_k(n) - y_k(n)$

Error Landscape in Weight Space

- Total error signal is a function of the weights

Ideally, we would like to find the global minimum (i.e. the optimal solution)



The Gradient Descent Rule

- It consists of computing the **gradient** of the error function, then taking a **small step** in the **direction of negative gradient**, which hopefully corresponds to decrease function value, then repeating for the new value of the dependent variable.
- In order to do that, we calculate the <u>partial derivative</u> of the error with respect to each weight.
- The change in the weight proportional to the derivative of the error with respect to each weight, and additional proportional constant (learning rate which here is η) is tied to adjust the weights.

$$\Delta \mathbf{w} = - \eta * \partial \mathbf{E} / \partial \mathbf{w}$$

LMS Algorithm - Derivation

- Steepest gradient descent rule for change of the weights:

Given

  - $x_k(n)$: an input value for a neuron k at iteration n,
  - $d_k(n)$: the desired response or the target response for neuron k.

Let:

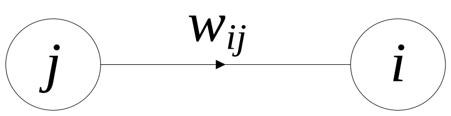
  - $y_k(n)$: the actual response of neuron k.
  - $e_k(n)$: error signal = $d_k(n)- y_k(n)$

Train the w_i's such that they minimize the squared error after each iteration

$$E(n) = \frac{1}{2} e_k^2(n)$$

LMS Algorithm – Derivation, cont.

- The derivative of the error with respect to each weight $\frac{\partial E}{\partial w_{ij}}$ can be written as:



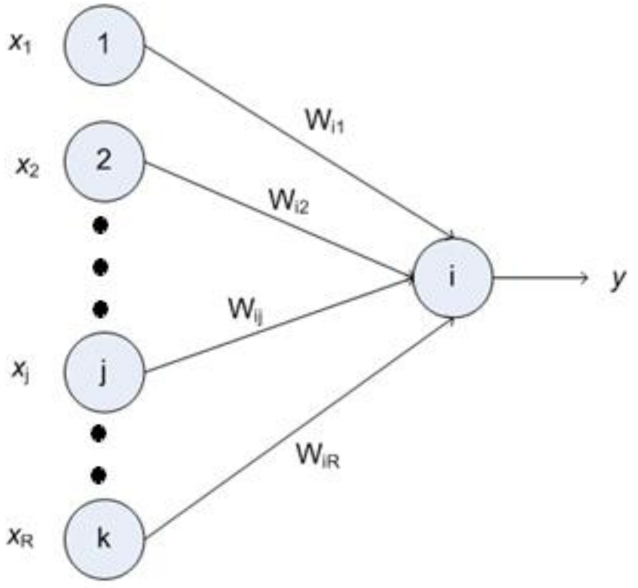
$$\nabla E(w_{ij}) = \frac{\partial E}{\partial w_{ij}} = \frac{\frac{1}{2}e_i^2}{\partial w_{ij}} = \frac{\partial \frac{1}{2}(d_i - y_i)^2}{\partial w_{ij}}$$

- Next we use the <u>chain rule</u> to split this into two derivatives:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial \frac{1}{2}(d_i - y_i)^2}{\partial y_i} \frac{\partial y_i}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = \left(\frac{1}{2} * 2(d_i - y_i) * -1\right) * \frac{\partial f\left(\sum_{j=1}^{R} w_{ij} x_j\right)}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = -(d_i - y_i) * x_j * f'\left(\sum_{j=1}^{R} w_{ij} x_j\right)$$



LMS Algorithm – Derivation, cont.

$$\nabla E(w_{ij}) = \frac{\partial E}{\partial w_{ij}} = -(d_i - y_i) * x_j * f^{'}(net_i)$$

$$\boxed{\Delta w_{ij} = -\eta \nabla E(w_{ij}) = \eta \sum_i (d_i - y_i) f'(net_i) x_j}$$

- This is called the Delta Learning rule.
- Then
  - The Delta Learning rule can therefore be used Neurons with differentiable activation functions like the sigmoid function.

LMS Algorithm – Derivation, cont.

- The Widrow-Hoff learning rule, LMS, is a special case of Delta learning rule. Since the Adaline's transfer function is linear function:

- then $f(net_i) = net_i \quad and \quad f^{'}(net_i) = 1$

$$\nabla E(w_{ij}) = \frac{\partial E}{\partial w_{ij}} = -(d_i - y_i) * x_j$$

- The Widrow-Hoff learning rule is:

$$\boxed{\Delta w_{ij} = -\eta \nabla E(w_{ij}) = \eta \sum_i (d_i - y_i) x_j}$$

Adaline Training Algorithm

1- initialize the weights to small random values and select a learning rate, (η)

2- Repeat

3- **for m** training patterns

select input vector **X** , with target output, **t,**

compute the output: **$y = f(v)$, $v = b + w^T x$**

Compute the output error **$e = t - y$**

update the bias and weights

$$w_i(\text{new}) = w_i(\text{old}) + \eta\,(t - y)\,x_i$$

4- end for

5- until the stopping criteria is reached by find the Mean square error across all the training samples

$$mse(n) = \frac{1}{m}\sum_{k=1}^{m} E(n)^2$$

stopping criteria: if the Mean Squared Error across all the training samples **is less than a specified value, stop the training.**

Otherwise , cycle through the training set again (go to step 2)

Convergence Phenomenon

- The performance of an ADALINE neuron depends heavily on the choice of the learning rate η.
- How to choose it?
- Too big
  - the system will oscillate and the system will not converge
- Too small
  - the system will take a long time to converge
- Typically, η is selected by trial and error
  - typical range: $0.01 < \eta < 1.0$
  - often start at 0.1
  - sometimes it is suggested that:
    $0.1/m < \eta < 1.0/m$
    where m is the number of inputs
- Choose of η depends on trial and error.

Online vs Batch vs Mini-batch Learning

- The weight update is calculated based on all samples in the training set (instead of updating the weights incrementally after each sample), which is why this approach is also called "batch" gradient descent. (cost function is minimized based on the complete training data set)
- weight update incrementally after each individual training sample. This approach is called "online" learning

- The process of incrementally updating the weights is also called "stochastic" gradient descent since it approximates the minimization of the cost function

- "mini-batches", a compromise with smoother convergence than stochastic gradient descent. For batch vs online:

https://www.youtube.com/watch?v=tIovUOirJkE&index=12&list=PLoRl3Ht4JOcdU872GhiYWf6jwrk_SNhz9

For mini-batch:
http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Example

- The input/target pairs for our test problem are

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \qquad \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$$

- Learning rate: $\eta = 0.4$
- Stopping criteria: mse < 0.03

Show how the learning proceeds using the LMS algorithm?

Example Iteration One

- First iteration – p_1

$$y = w(0) \ * P_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = 0$$

e = t – y = -1 – 0 = -1

$$w(1) \ = w(0) \ + \ \eta * e \ * \ P_1$$

$$w(1) \ = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - 0.4 \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.4 \\ -0.4 \\ 0.4 \end{bmatrix}$$

Example Iteration Two

- Second iteration – p_2

$$y = w(1) \; * P_2 = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4$$

e = t – y = 1 – (-0.4) = 1.4

$$w(2) \; = w(1) \; + \; \eta * e \; * \; P_2$$

$$w(2) \; = \begin{bmatrix} 0.4 \\ -0.4 \\ 0.4 \end{bmatrix} + 0.4(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.96 \\ 0.16 \\ -0.16 \end{bmatrix}$$

End of epoch 1, check the stopping criteria

Example – Check Stopping Criteria

For input P_1

$$e_1 = t_1 - y_1 = t_1 - w(2)^T * P_1$$

$$= -1 - \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -1 - (-0.64) = -0.36$$

For input P_2

$$e_2 = t_2 - y_2 = t_2 - w(2)^T * P_2$$

$$= 1 - \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = 1 - (1.28) = -0.28$$

$$mse(1) = \frac{(-0.36)^2 + (-0.28)^2}{2} = 0.1037 > 0.03$$

Stopping criteria is not satisfied, continue with epoch 2

Example – Next Epoch (epoch 2)

- Third iteration – p_1

$$y = w(2)\ * P_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

e = t – y = -1 – 0.64 = -0.36

$$w(3)\ = w(2)\ +\ \eta * e\ *\ P_1$$

$$w(3)\ = \begin{bmatrix} 0.96 \\ 0.16 \\ -0.16 \end{bmatrix} + 0.4(-0.36) \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1.104 \\ -0.016 \\ -0.016 \end{bmatrix}$$

if we continue this procedure, the algorithm converges to:

W(…) = [1 0 0]

LMS vs Perceptron

- LMS more powerful than Perceptron
- Perceptron's rule is guaranteed to converge to a solution that correctly categorizes the training patterns but the resulting network can be sensitive to noise as patterns often lie close to the decision boundary
- if the patterns are not linearly separable, i.e. the perfect solution does not exist, but an ADALINE will find the best solution possible by minimizing the error (given the learning rate is small enough)
- **LMS** output is a real number and not a class label as in the perceptron learning rule
- Both use updating rule changing with each input
- One fixes binary error; the other minimizes continuous error
- Adaline always converges; see what happens with XOR
- Both can REPRESENT Linearly separable functions
- **The Adaline,** is similar to the **perceptron**, but their transfer function is linear rather than hard limiting. This allows their output to take on any value.

References

- http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html
- xa.yimg.com/kq/groups/23111477/1569856059/name/ANN_4_LMSalgorithm.pptx → (Dr. Hala Moushir Ebied powerpoints)
- https://www.youtube.com/watch?v=LOc_y67AzCA&list=PLoRl3Ht4JOcdU872GhiYWf6jwrk_SNhz9&index=14