Machine Learning for Signal Processing

[5LSL0]

Assignment 1: Optimum Linear Filters

REPORT

Group number: 2

Names including ID:

1:Martin van Leeuwen	**0901497**
2:Frouke Hekker	**0897373**

Date: 3-5-19

KNOWN STATISTICS

WIENER FILTER

a)

$$E(e^2[k]) = E\{(y[k] - w^t \cdot x[k]) \cdot (y[k] - x^t[k] \cdot w)\}$$

$$= \mathrm{E}\{\mathrm{y}^2[\mathrm{k}]\} - \mathrm{w}^t E\{x[k]y[k]\} - E\{y[k]x^t[k]\}w + w^t E\{x[k]x^t[k]\}w$$

$$= \mathrm{E}\{\mathrm{y}^2[\mathrm{k}]\} - w^t r_{yx} - r_{yx}^t w + w^t R_x w$$

$$\underline{\boldsymbol{w}}_0 = \arg min\ E(e^2[k]) = R_x^{-1} \cdot r_{yx}$$

$$= \begin{pmatrix} 5 & -1 & -2 \\ -1 & 5 & -1 \\ -2 & -1 & 5 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 \\ 5.3 \\ -3.9 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 1 \\ -0.5 \end{pmatrix}$$

b)

$$\underline{r}_{xe} = \mathrm{E}\{\mathrm{x}[\mathrm{k}]\mathrm{e}[\mathrm{k}]\} = E\{\ x[k](y[k] - w^t[k]x[k])\}$$

$$= r_{yx} - E\{\ x[k]x[k]\}w = r_{xy} - R_x w$$

$$= \begin{pmatrix} 1 \\ 5.3 \\ -3.9 \end{pmatrix} - \begin{pmatrix} 5 & -1 & -2 \\ -1 & 5 & -1 \\ -2 & -1 & 5 \end{pmatrix} \begin{pmatrix} 0.2 \\ 1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

In the optimal solution the correlation between the error and the input is zero. This would mean that the errors produced by the model are not correlated with the input, and thus all non-random patterns are compensated within the model. Consequently, the errors that remain are caused by noise within the data.

c)

$$\widehat{\boldsymbol{R}}_x = X^t X$$

$$\underline{\hat{r}}_{yx} = X^t y$$

If statistical information is not available it might also be difficult to get it. Whit these approximations that isn't necessary. However this approximation will change for every X and thus the loss landscape will seem to change too. This results in a sub-optimal path to the minimum.

Steepest gradient descent

d)

The gradient descent algorithm goes to a steady state if

$$\lim_{k\to\infty} w[k] \cong R_x^{-1} \cdot r_{xy}$$

e)

$$\begin{pmatrix} 5 & -1 & -2 \\ -1 & 5 & -1 \\ -2 & -1 & 5 \end{pmatrix} - \lambda I = \begin{pmatrix} 5-\lambda & -1 & -2 \\ -1 & 5-\lambda & -1 \\ -2 & -1 & 5-\lambda \end{pmatrix}$$

$$= (5-\lambda)\big((5-\lambda)(5-\lambda)-1\big) + (-1(5-\lambda)-2) - 2(1+2(5-\lambda))$$

$$= (5-\lambda)(\lambda^2 - 10\lambda + 24) + -7 + \lambda - 2(1 + 10 - 2\lambda)$$

$$= -\lambda^3 + 10\lambda^2 - 24\lambda + 5\lambda^2 - 50\lambda + 120 - 7 + \lambda - 22 + 4\lambda$$

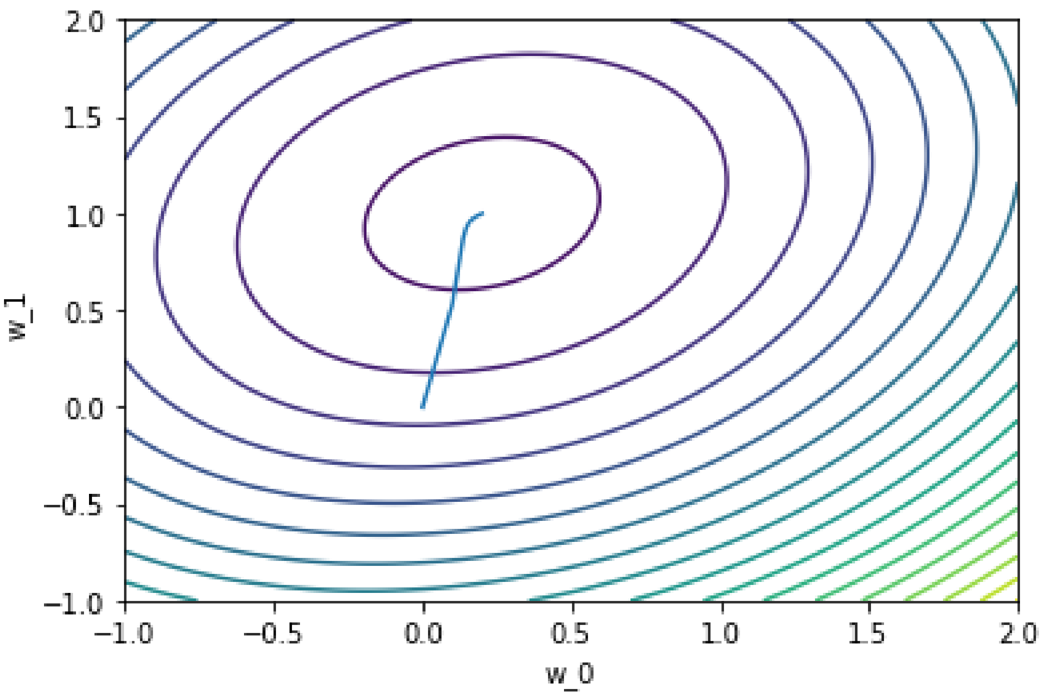
$$= -\lambda^3 + \ 15\lambda^2 - 69\lambda + 91$$

$$= (\lambda - 7)(\lambda^2 - 8\lambda + 13)$$

$$thus\ \lambda = 7\ and\ \lambda = 5.7321\ and\ \lambda = \ 2.2679$$

f) GD filter update Python code (insert only the relevant lines)

```
for k in range(N):
    w +=   [w[-1] + 2*alpha*(r_yx-np.matmul(R_x,w[-1]))]
```



α is chosen to be 1/20. The blue line shows how w converges to the minimum. It does not take the shortest route, it takes a curved route. However it is stable and doesn't overshoot.

Newton algorithm

g)

Newton converges for:

$$d[k+1] = (I - 2\alpha R_x^{-1} R_x) d[k]$$

$$= (1 - 2\alpha)\mathrm{d[k]}$$

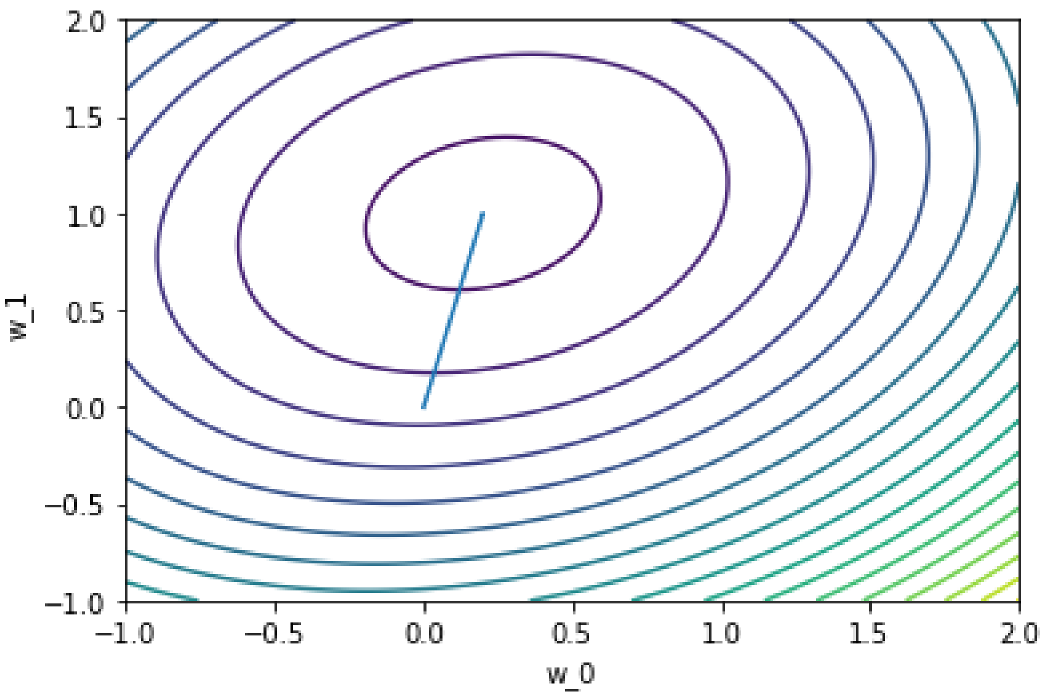
And α is not dependent on the filter weights so they can only converge at the same rate.

h)

Because the w dimensions are whitened 0<α<1 is stable.

i) Newton filter update Python code (insert only the relevant line)

```
for k in range(N):
    w += [ w[-1] + np.matmul( 2*alpha*Rinv,(r_yx-np.matmul(R_x,w[-1])))]
```



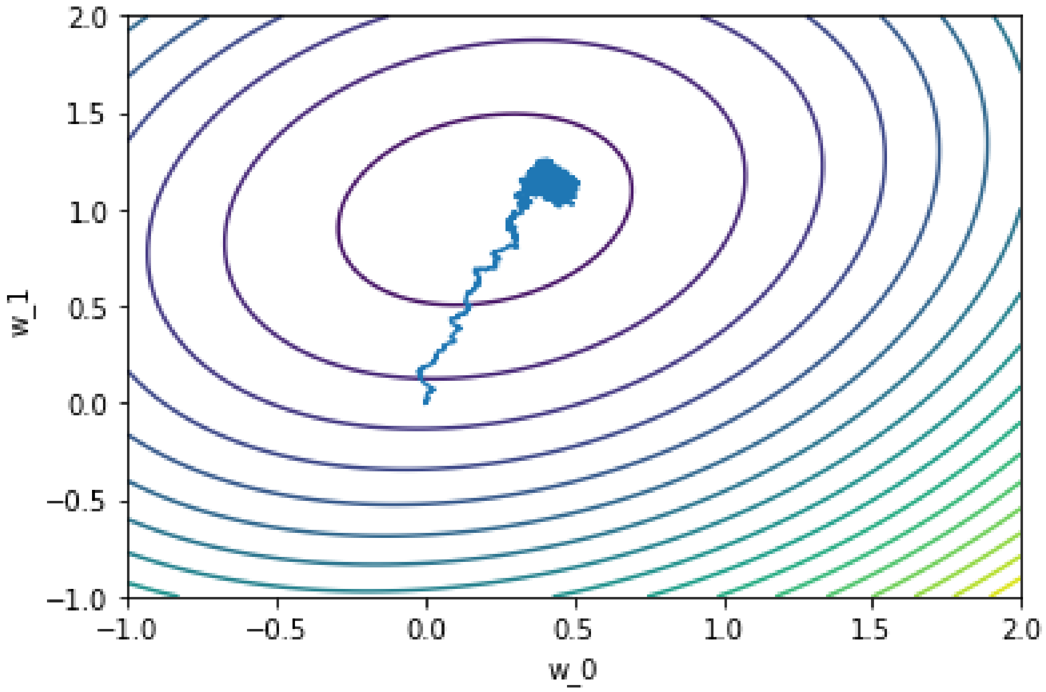
α *is chosen to be 0.5 . Now that Newton is used the weights converge in a straight line, which is the shortes route. The convergence is still smooth.*

Unknown statistics

(N)LMS

j)

```
for k in range(1,N-1):
   inp = x[k-1:k+2]
   y_pred += [np.sum(inp * w[-1])]
   e += [y[k]-y_pred[-1]]
   w += [ w[-1] + 2 * alpha * np.array(inp) * e[-1]]
```



Interestingly, the LMS algorithm does not converge to the center. A possible explanation is that the given R_x and r_{yx} have not been predicted accurately for the given input and output data. Alternatively, if the system is dynamic, the optimum for the weights of the system may have been displaced overtime. Consequently, the point the system convergence towards varies.

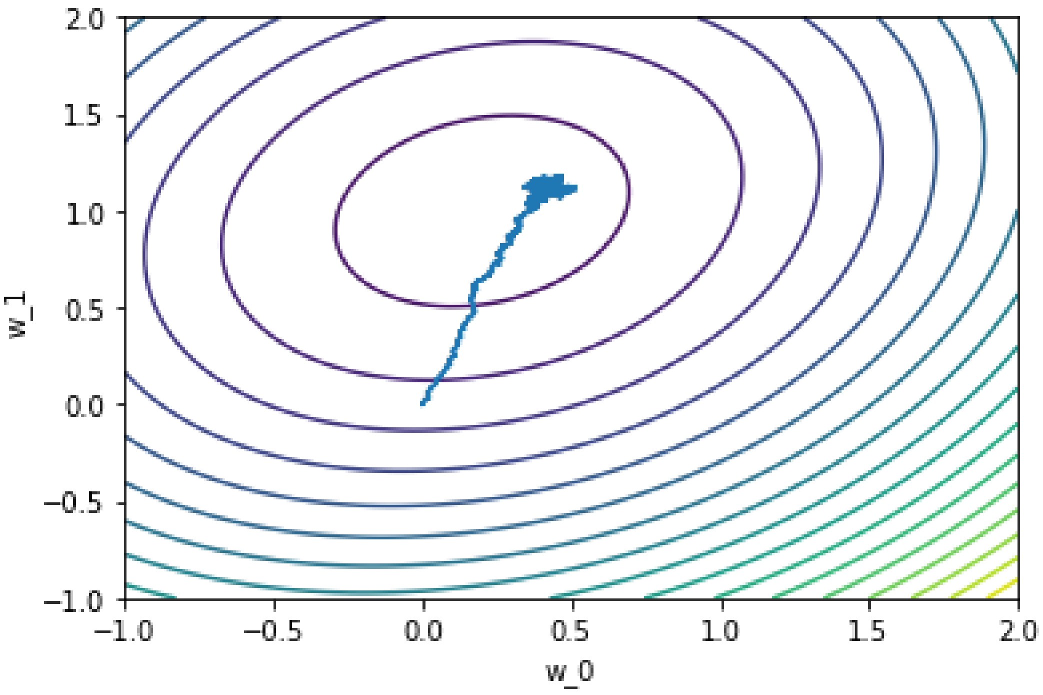
Trade-off choosing α:

A high α causes the system to converge faster. However, if the α is chosen too large the system might end up substantially overshooting the minimum. In the subsequent iteration the system attempts to move in the opposite direction but, overshoots the minimum again. In such a scenario the system will either keep oscillating around the local minimum or it might "jump" out of the valley completely and go in a random direction.

In conclusion the trade-off between the precision and convergence speed has to be made to choose α.

k)

```
for k in range(1,N-1):
    inp = np.array(x[k-1:k+2])
    y_pred += [np.sum(inp * w[-1])]
    e += [y[k]-y_pred[-1]]
    sigma = np.matmul(inp.T,inp)/3 + eps
    w += [ w[-1] + 2 * alpha/sigma * inp * e[-1]]
```



RLS

l)

RLS is related to the previous algorithms in the sense that it attempts to minimize a squared error and converge to the Wiener solution. However, the previously discussed methods contain various drawbacks that either hinders the convergence efficiency, may cause instability or require knowledge on the autocorrelation matrix R_x.

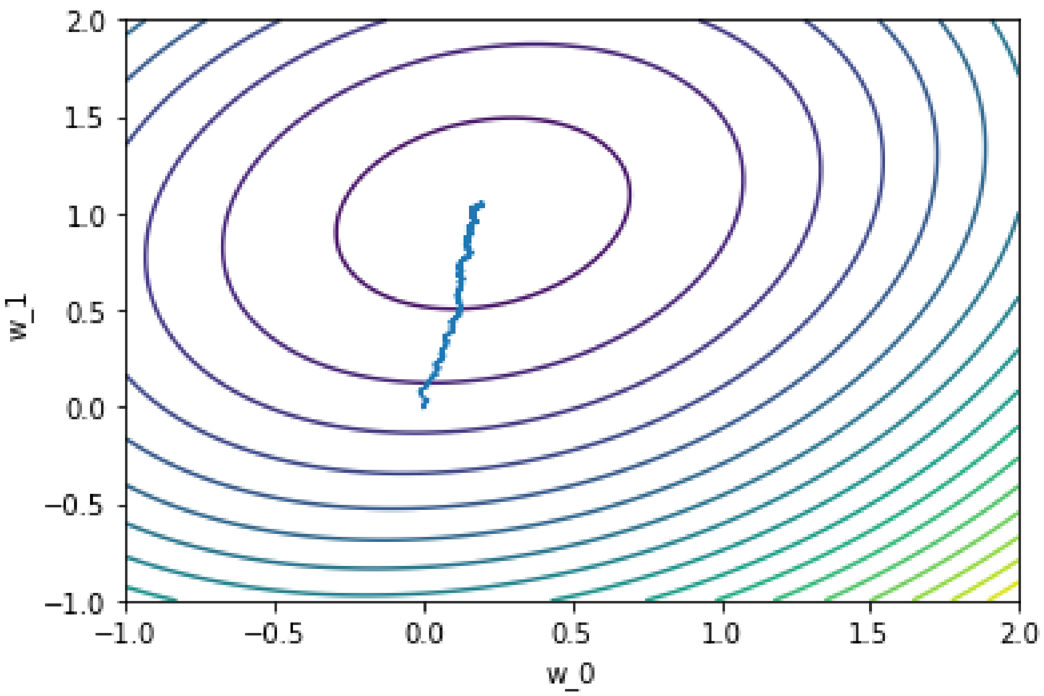
The problem in LS is that R_x^{-1} is required in order to determine the optimal wiener solution. In order to obtain R_x^{-1} expensive computations are required and the result can be unstable. Gradient descent avoids this issue completely by estimating the optimum. However, gradient descent converges to the optimal solution in a curved path instead of a more efficient straight line. Furthermore, the algorithm needs $R_x \; and \; r_{yx}$ in order to compute solutions. Like gradient descent, Newton uses knowledge of the matrix R_X to update its weights. However, by using R_x^{-1} the input process can be whitened and thus Newton convergences in a straight line to the optimal solution. NLMS and LMS are different from the gradient descent and Newton methods since they estimate $R_x \; and \; r_{yx}$, and thus do not depend on knowledge of $R_x \; and \; r_{yx}$ beforehand. However, these methods do not withen the input processes and thus often they do not converge in an efficient straight line to the optimal solution.

RLS iteratively estimates R_x^{-1} as well in order to include withening of the input process. Therefore, it has efficient convergence like the Newton method without depending on the knowledge of $R_x \; and \; r_{yx}$ beforehand.

m)

If $gamma$ is increased, less weights is given to older samples. They will be "forgotten"

If $gamma$ is decreased, older samples will be taken into account more for the new results.



```
inp_next = np.array(x[k:k+3])
# g_next_denominator is a scalar
g_next_denominator = gamma**2 + inp_next.T*R_xinv_pred*inp_next

g_next = R_xinv_pred*inp_next/g_next_denominator
R_xinv_pred_next = gamma**-2 * (R_xinv_pred-np.dot(g_next,inp_next.T)*R_xinv_pred)
r_yx_pred_next = gamma**2 * r_yx_pred + inp_next.T*y[k+1]
w+=[np.dot(R_xinv_pred_next, r_yx_pred_next)]
r_yx_pred = r_yx_pred_next
R_xinv_pred = R_xinv_pred_next
```

n)

	Computational complexity	Convergence speed/stability/accuracy
LMS	1	3
NLMS	2	2
RLS	3	1

The training chosen training method may have a significant impact for the computational complexity. Especially, the matrix multiplications within the training methods will require a large number of computations if a larger order is chosen for the filter.

NLMS is more computationally expensive because it adds an additional matrix multiplication to the LMS method in order to normalize the step size. However, RLS is the most expensive because it requires far more computation steps, most of them including matrix multiplications.

In accordance with the results in this report LMS converges to the least accurate solutions. With the time-varying step-size of the NLMS a improvement in accuracy can be obtained since the step size can be adjusted to better fit the scenario. The RLS attempts to withen the input process which improves the efficiency of the convergence. Consequently, RLS leads to the most desirable convergence behavior.