Digital Signal Processing 2/ Advanced Digital Signal Processing
Lecture 13,
Matched Filter, Prediction

Gerald Schuller, TU Ilmenau

Matched Filters

Remember the goal of a matched filter $h_M(n)$:

$$y(n)*h_M(n) \rightarrow x(n+n_d)*h_M(n)$$

with some signal delay n_d (no signal fidelity, just high SNR for detection), where $y(n)=x(n+n_d)+v(n)$ is our (delayed) signal with additive noise $v(n)$.

Application examples are in communications, where you would like to detect a 0 or 1,for instance in CDMA, where each user gets a unique pseudo-random 1/0 sequence (so-called chip-sequences) to represent 0 or 1, in which different users and signals are separated using matched filters. Another example is for detecting **known** signals or patterns, like object or face recognition in images. In general we would like to detect **deterministic signals** $x(n)$.

This means our goal is to **maximize the SNR** at the moment of detection, with our original signal $x(n)$ and noise $v(n)$,

$$SNR = \frac{|x(n) * h_M(n)|^2}{E\left(|v(n) * h_M(n)|^2\right)}$$

We would like to maximize this SNR at the time of detection, using $h_M(n)$. To do that, first we assume $v(n)$ to be independent white noise. Then the denominator of the SNR expression is just a scaled fixed power expression. Using our formulation of a matrix $\boldsymbol{V}$ for the noise signal and the **row vector** $\boldsymbol{h_M}$ for our filter, we obtain

$$E\left(|v(n) * h_M(n)|^2\right) = E\left(|\boldsymbol{V} \cdot \boldsymbol{h_M^T}|^2\right) = E\left(\boldsymbol{h_M} \cdot \boldsymbol{V^T} \cdot \boldsymbol{V} \cdot \boldsymbol{h_M^T}\right) =$$
$$= \boldsymbol{h_M} \cdot E\left(\boldsymbol{V^T} \cdot \boldsymbol{V}\right) \cdot \boldsymbol{h_M} = \boldsymbol{h_M} \cdot \sigma_v^2 \cdot \boldsymbol{I} \cdot \boldsymbol{h_M^T} =$$
$$= \sigma_v^2 \cdot \boldsymbol{h_M} \cdot \boldsymbol{h_M^T}$$

(Remember: the autocorrelation function of white noise is a weighted delta function, since noise samples are uncorrelated to all their neighboring samples, they are only correlated with themselves, and the correlation with itself is simply the noise power σ^2 . The autocorrelation matrix $E\left(\boldsymbol{V^T} \cdot \boldsymbol{V}\right)$ hence has all zero entries, except on the diagonal, where it is the noise power, hence noise power time the identity matrix $\sigma^2 \cdot \boldsymbol{I}$)

The last expression is simply the squared norm (the sum of the squares of its coefficients) of our vector of our filter coefficients $\boldsymbol{h_M}$ multiplied with the noise power σ_v^2 .

Keeping the above **norm** of our filter vector $\boldsymbol{h_M} \cdot \boldsymbol{h_M^T}$

constant, the entire denominator is fixed, and we only need to **maximize the numerator** of our SNR fraction to maximize the SNR,

$$|x(n)*h_M(n)|^2$$

We rewrite our **numerator** as

$$|x(n)*h_M(n)|^2=|\boldsymbol{x}(n)\cdot\boldsymbol{h_M^T}|^2$$

analog to our matrix formulation, as a scalar vector multiplication, with $\boldsymbol{h_M}$ as our row **vector** of the matched filter impulse response, and now with only one row of the signal matrix **A** from last time at a time, for only one convolution sample at a time. Observe that for matched filters the signal to detect can be much shorter than for the Wiener filter, and hence we can decide to make the matched filter the **same size** as our signal to detect!
The signal **row vector** is

$$\boldsymbol{x}(n)=[x(L-1-n),x(L-2-n),\ldots,x(n)]$$

where L is the size of our filter vector. Observe that it is **time-reversed** to obtain a value of the convolution.

We apply the Cauchy-Schwartz inequality (see e.g. http://en.wikipedia.org/wiki/Cauchy%E2%80%93Schwarz_inequality), which says, for 2 (column) vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ and their scalar product we get

$$\boldsymbol{a^T\cdot b}\leq\sqrt{\boldsymbol{a^T\cdot a}}\cdot\sqrt{\boldsymbol{b^T\cdot b}}$$

This is also written with the norm $\|\boldsymbol{a}\|$ and $\langle\boldsymbol{a},\boldsymbol{b}\rangle$ scalar product as

$$|\langle\boldsymbol{a},\boldsymbol{b}\rangle|\leq\|\boldsymbol{a}\|\cdot\|\boldsymbol{b}\|$$

We obtain equality if both vectors are co-linear, $\boldsymbol{b}=k\cdot\boldsymbol{a}$, with some scalar k . This tells us how to solve the maximization task.

We can now apply Cauchy-Schwartz if we set $\boldsymbol{a}=\boldsymbol{x}(n)$ and $\boldsymbol{b}=\boldsymbol{h}_{\boldsymbol{M}}$,

$$|\boldsymbol{x}(n)\cdot\boldsymbol{h}_{\boldsymbol{M}}^{\boldsymbol{T}}|^2\leq\left\|\boldsymbol{x}(n)\right\|^2\cdot\left\|\boldsymbol{h}_{\boldsymbol{M}}\right\|^2$$

and we get the equality (the maximum) if we set

$$\boldsymbol{h}_{\boldsymbol{M}}=k\cdot\boldsymbol{x}(n)$$

where we can choose the factor as $k=1$.
Since we have this inequality for all time steps n of our convolution, we choose n where the row vector $\boldsymbol{x}(n)$ has the maximum energy. This is where we **capture the entire, non-zero, wave form** of our signal $\boldsymbol{x}(n)$ with our filter. Since our filter vector $\boldsymbol{h}_{\boldsymbol{M}}$ contains the time-reversed impulse response, we obtain the entire **time reversed signal** as our **matched filter**:

$$h_M(n)=x(L-1-n)$$

assuming our signal to detect is located between $0\leq n<L$.
Since we have a convolution of the signal with its time reverse version, we get a convolution length of $2\mathrm{L}-1$ samples, with its maximum at the center, when both waveforms completely overlap, after L samples, which is exactly the signal length. Hence we get the **detection** of our signal after we completely received it, at the **end of our signal**.

Observe: Since we convolve the signal with the time-reversed version of our pattern to be detected, this becomes identical to computing the **correlation** of the signal with the pattern to be detected.
Also Observe: The **longer** the signal is, the **more energy** is captured in it, and the **higher the SNR** will be at the time of detection. This is important for instance for very weak signals, like in deep space communcations.

For the continuous-time version see, for instance: http://www.ece.gatech.edu/research/labs/sarl/tutorials/ECE4606/14-MatchedFilter.pdf

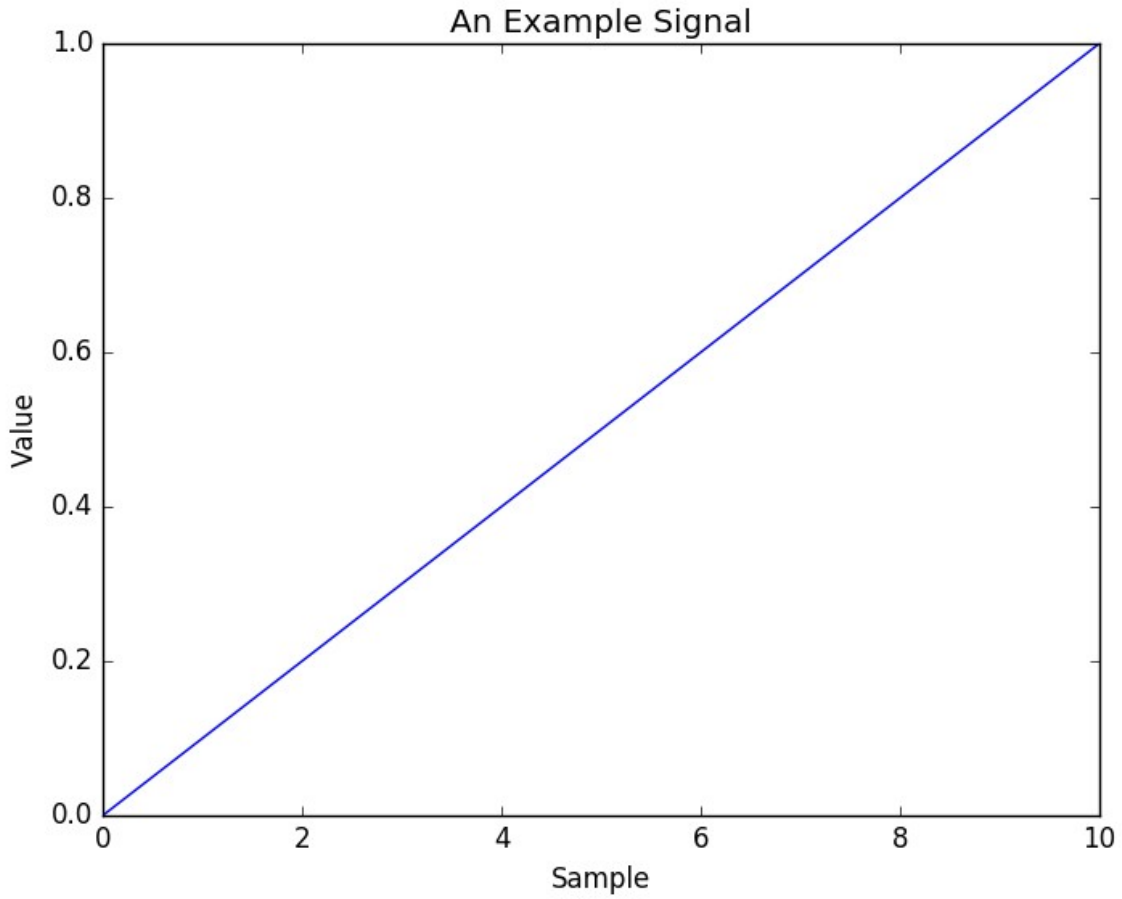
In Conclusion: The matched filter has the shape of the time reversed signal to be searched for.

Python Example

Construct a signal to be detected, sig (length 11):

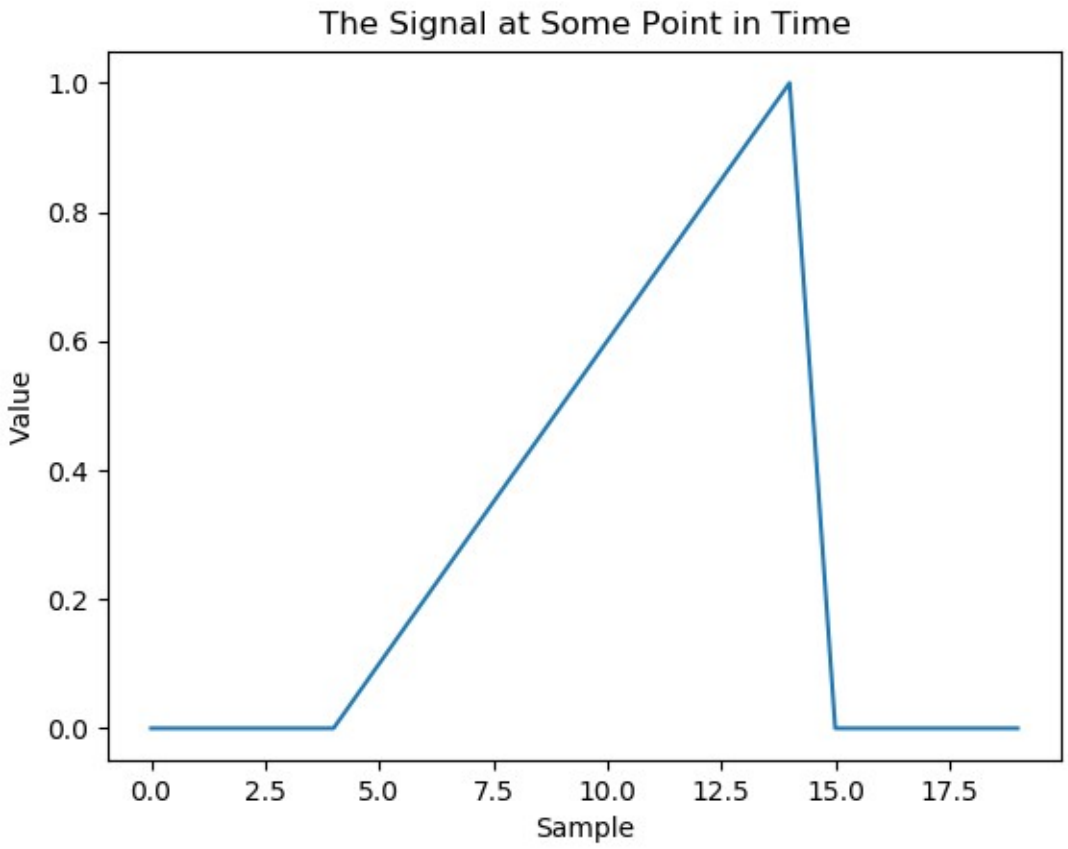
```
ipython –pylab
sig = arange(0, 1.1, 0.1)
sig
#Out: array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,
#0.6,  0.7,  0.8,  0.9,  1. ])
plot(sig)
xlabel('Sample')
```

```
ylabel('Value')
title('An Example Signal')
```



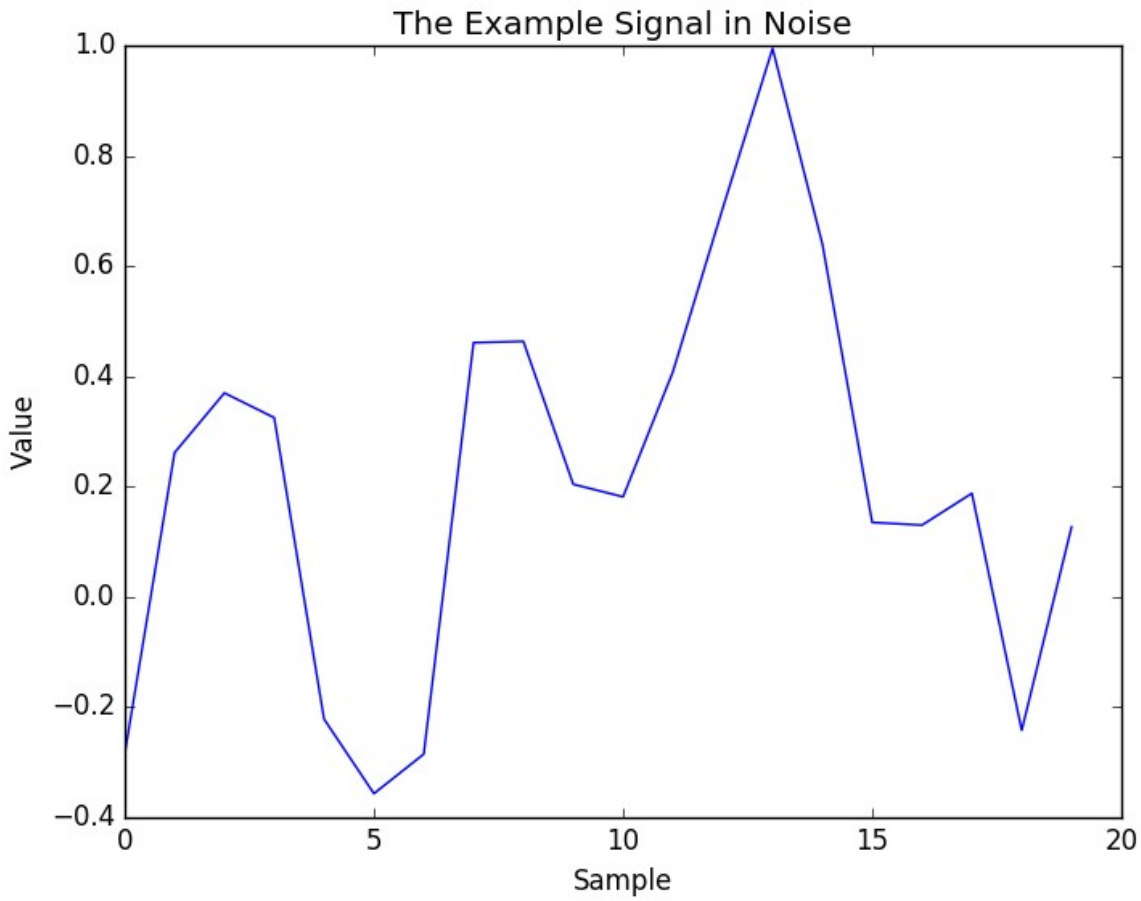
We put this signal to detect at some point in time in a longer signal,

```
sig_inzeros=hstack([zeros(4),sig,zeros(5)])
plot(sig_inzeros)
xlabel('Sample')
ylabel('Value')
title('The Signal at Some Point in Time')
```



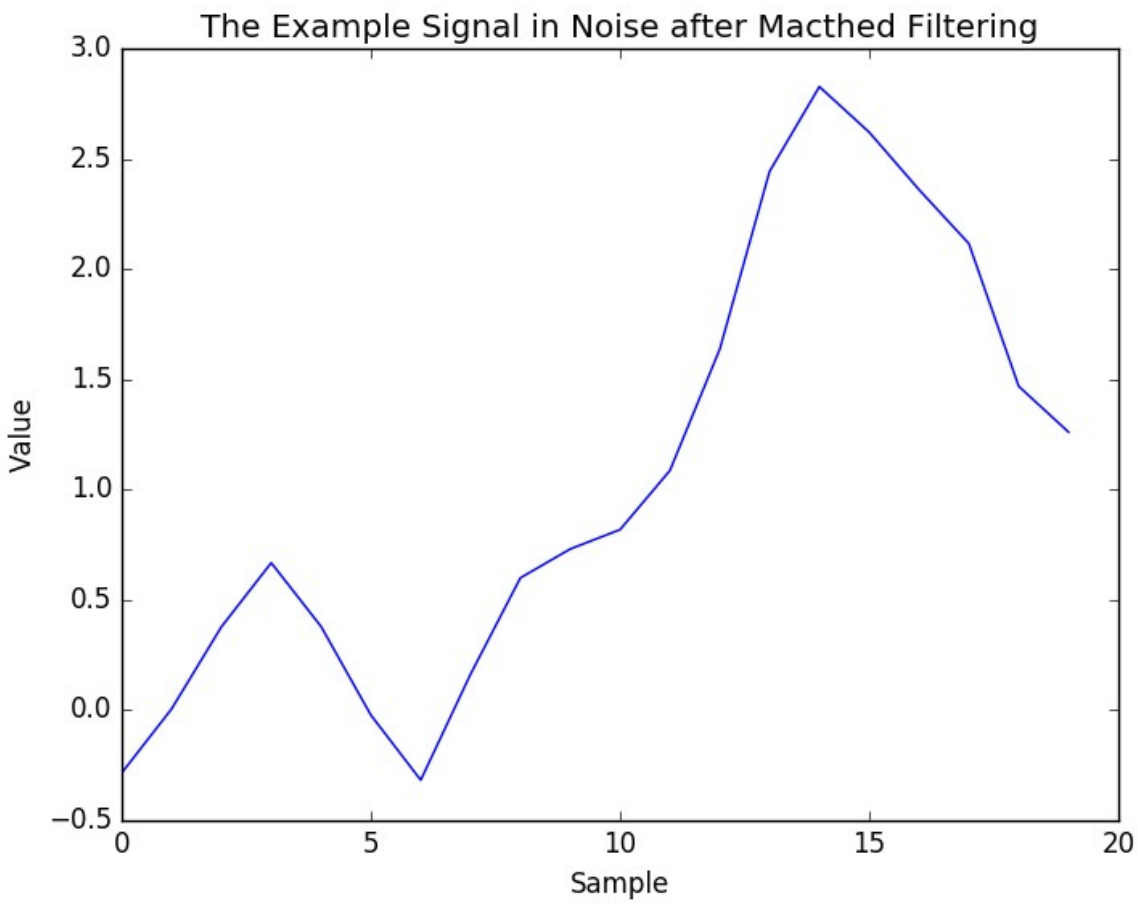
Now we add noise and extend the length of our signal:

```
signoise = random.rand(20)-0.5+sig_inzeros

plot(signoise)

xlabel('Sample')

ylabel('Value')

title('The Example Signal in Noise')
```



Now we apply our matched filter to it:

```
h = sig[::-1] # fliplr
signoisemf = sp.lfilter(h, 1, signoise)
plot(signoisemf)
xlabel('Sample')
ylabel('Value')
title('The Example Signal in Noise after Macthed Filtering')
```



This is now the output of our matched filter. We can see that we have a maximum at time 14, which marks the **end** of our **detected signal**. Hence we know that the signal started at sample 14-L(length of the filter)=14-11=3, which was indeed the case since we added 4 zeros in the beginning. So matched filtering did a good job!
The **matched filtering** process can also be viewed as computing the **correlation** of the noisy signal with the original signal.

Convolutional Neural Network Implementation

Observe that we got a high peak for the detection of our signal, but the peak was somewhat broad, which makes determining the precise location of the signal more difficult. This is because we specified as a target for our optimization that we want to have a high peak, but not necessarily a narrow peak. To remedy this, we can use **numerical optimization** instead of our closed form solution for matched filters.

For that, we can use the optimization of a neural network library, like Pythons "Pytorch". Pytorch has the advantage, compared to e.g. Keras, that "print" commands work, which is important and useful for debugging.

The following example also serves as a short introduction into neural networks and its terminology.

We can implement our filter using a 1-dimensional "convolutional layer", object `conv1d`, without "bias" and without a non-linear "activation function".

The library "Pytorch" is obtained and installed from www.pytorch.org.

For it we need to specify a "training" signal, here our signal to detect (the "ramp" function) `x`, and the "target" signal `y`, which is the desired output of the convolutional layer which the optimization should reach as closely as possible during the optimization or "training",

Our training set is:

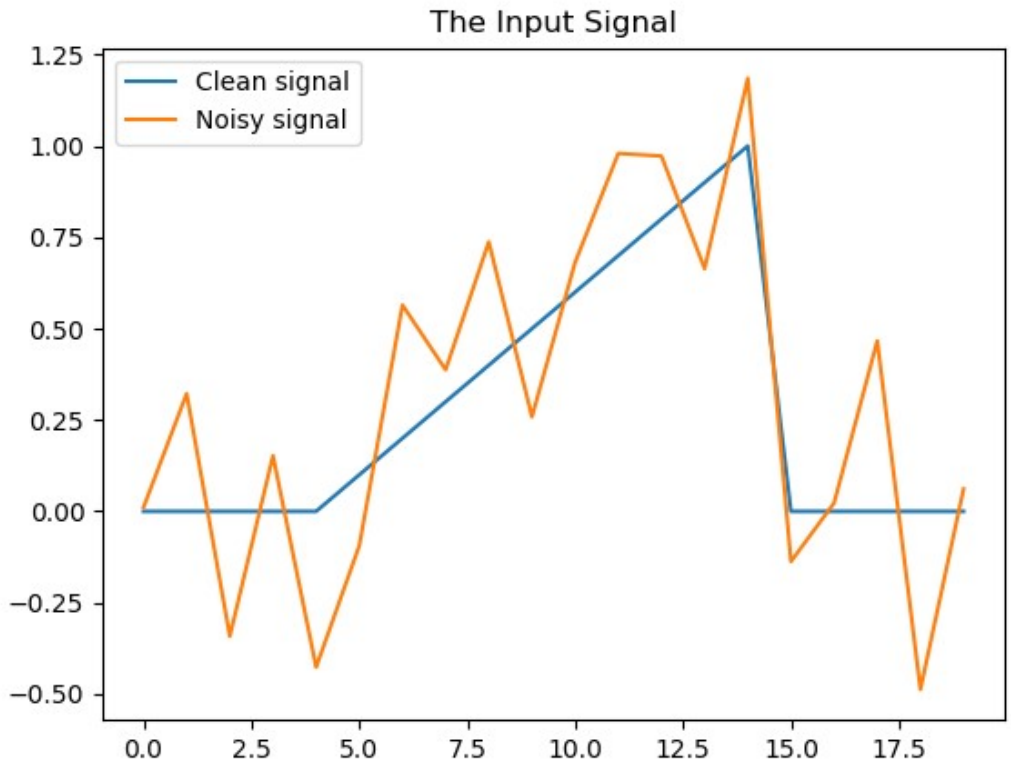
```
x= np.hstack((np.zeros(4),np.arange(0,1.1,0.1),np.zeros(5)))
y = np.zeros(30)
y[16]=1 #Detecting the signal at its end
```

Observe that for the target `y` we specified a **single peak** (the “1”) at the position the filter should detect the signal, and **zeros everywhere else**. This also leads to a minimization of the output outside the signal detection, which we didn’t have in our closed form solution!
We specify our convolutional detector layer as,

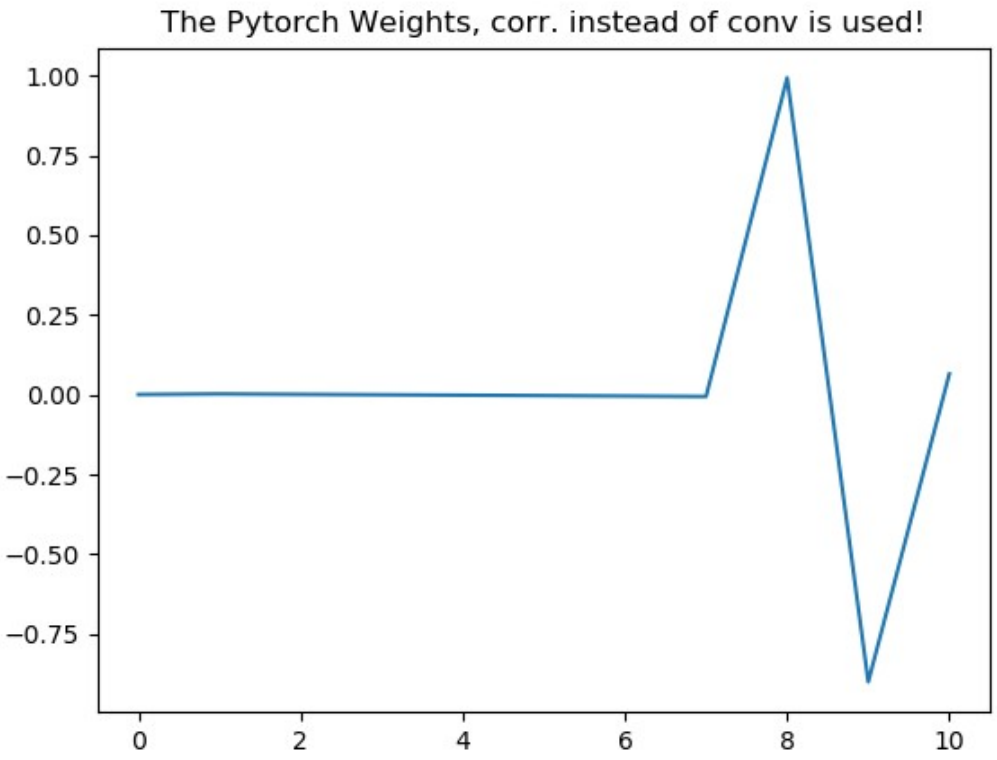
```
detector=nn.Sequential(nn.Conv1d(in_channels=1,
out_channels=1, kernel_size=11, stride=1, padding=10,
bias=False))
```

Observe that here `padding=kernel_size-1`, which leads to a **causal** filter, meaning all its inputs are not in the future. This implements our linear filter as detector. The optimization we apply minimizes the **mean squared error** (the so called “L2 loss function”).
We can let the optimization run on the same ramp function as before and display the output of the detector with the clean and noisy signal as input with, in our Lunux terminal shell,

```
python3 pytorch_simpl_convnet_detector.py
```

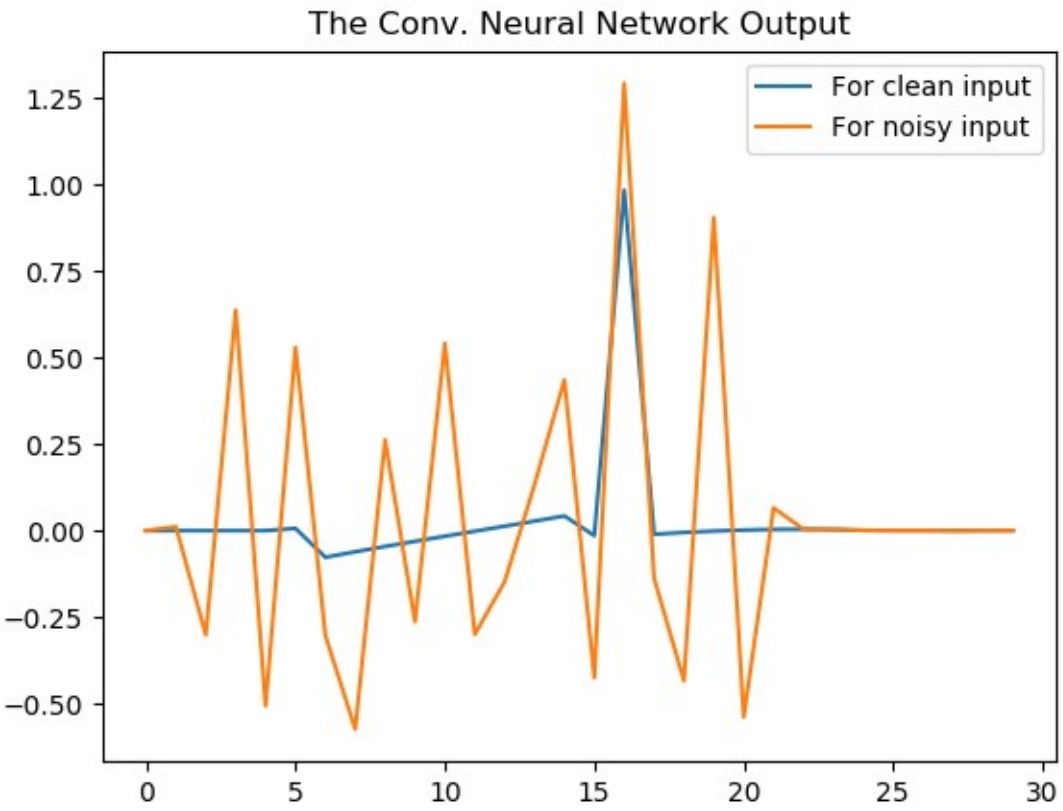


We use the same type as input as before.



The figure shows the obtained weights or coefficients, in pytorch they are used as correlation instead of

convolution, and hence the plot is a **time-reversed** impulse response.
Observe that it looks indeed different from our matched filter, which was simply the (time-reversed) ramp signal. This is because the optimization also tries to minimized the output outside the detection time point.



The result of the detection for the clean and noisy signals. Observe the peak at the detection time point is indeed much **more narrow** than for the matched filter!

Prediction

Prediction can be seen as a special case of a Wiener filter, where the noise of our signal corresponds to a shift of our signal into the past. Our goal is to make a “good” estimation of the present sample of our signal, based on **past signal samples.** “Good” here means, again in a mean squared error sense.

Basically we can now take our Wiener Filter formulation, and specialize it to this case. Looking at our matrix formulation, we get

$$\begin{bmatrix} x(1) & x(0) \\ x(2) & x(1) \\ x(3) & x(2) \\ \vdots & \vdots \end{bmatrix} \cdot \begin{bmatrix} h(0) \\ h(1) \end{bmatrix} = \begin{bmatrix} x(2) \\ x(3) \\ x(4) \\ \vdots \end{bmatrix}$$

or

$$\boldsymbol{A} \cdot \boldsymbol{h}^{\boldsymbol{T}} \rightarrow \boldsymbol{x}$$

This means, the input to our filter is always starting at one sample in the past, going down further into the past. Its goal is to estimate or “predict” the next coming sample. Basically this means that instead of additive white noise, our **distortion** is now a **delay** operator (fortunately, this is a linear operator).

Now we can again use our approach with pseudo-inverses to obtain our mean-squared error solution,

$$\boldsymbol{h} = \left(\boldsymbol{A}^{\boldsymbol{T}} \cdot \boldsymbol{A} \right)^{-1} \boldsymbol{A}^{\boldsymbol{T}} \cdot \boldsymbol{x}$$

with the matrix $\boldsymbol{A}$ now defined as our above matrix. This now also leads to a statistical description, with $\boldsymbol{A}^{\boldsymbol{T}} \cdot \boldsymbol{A}$

converging to

$$\boldsymbol{A}^{\boldsymbol{T}}\cdot\boldsymbol{A}\rightarrow\boldsymbol{R}_{\boldsymbol{xx}}=\begin{bmatrix} r_{xx}(0) & r_{xx}(1) & r_{xx}(2) & \dots \\ r_{xx}(1) & r_{xx}(0) & r_{xx}(1) & \dots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

This is plausible, because now y(n) is just the delayed signal, and the auto-correlation function of the delayed signal is identical to the auto-correlation function of the original function.

Next we need the cross-correlation $\boldsymbol{A}^{\boldsymbol{T}}\cdot\boldsymbol{x}$. Since we now just have this 1 sample in the future as our target vector, this converges to the auto-correlation vector, starting at **lag 1**,

$$\boldsymbol{A}^{\boldsymbol{T}}\cdot\boldsymbol{x}\rightarrow\boldsymbol{r}_{\boldsymbol{xx}}=\begin{bmatrix} r_{xx}(1) \\ r_{xx}(2) \\ \vdots \end{bmatrix}$$

So together we get the solution for our prediction filter as

$$\boldsymbol{h}=\left(\boldsymbol{R}_{\boldsymbol{xx}}\right)^{-1}\boldsymbol{r}_{\boldsymbol{xx}}$$