Lab 2A: Audio Processing

Ignacio Boero, Juan Cerviño, Juan Elenter,
Ignacio Hounie and Alejandro Ribeiro*

February 12, 2024

1 Audio Signals

Audio is mathematically modeled as a function $x(t)$ in which t represents time and $x(t)$ is an electric signal that is generated by transforming pressure waves with a microphone. The same pressure waves can be reconstructed from the electrical signal using a speaker.

We can create a digital representation of an audio signal through sampling; see Figure 1. To do so define a sampling time T_s and a number of components N and proceed to sample the signal $x(t)$ every T_s units of time. This results in the digital audio signal $\mathbf{x}$, which is mathematically represented by the vector

$$\mathbf{x} = [x(0); x(T_s); x(2T_s); \dots; x((N-1)T_s)] = [x_0; x_1; x_2; \dots; x_{N-1}]. \quad (1)$$

A digital audio signal is more convenient than the original analog representation because it is easier to process. The vector $\mathbf{x}$ can be manipulated to extract information or make it better in a number of ways. In this lab we have excerpts of human speech that are contaminated with background noise. Our goal is to remove as much of the background noise as possible.

*In alphabetical order.

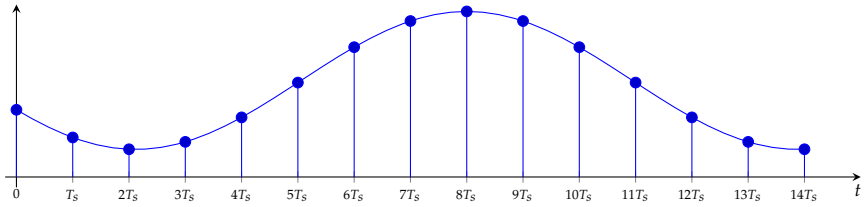

Figure 1. Sampling. Audio signals are waveforms in continuous time with digital representations obtained through sampling. Digital representations are advantageous because they are easier to process.

Task 1 Load audio data from the link provided in the course site , and unzip it. This file contains a Pytorch tensor with Q = 1500 pairs of audio recordings $(\mathbf{x}_q, \mathbf{y}_q)$. The signals $\mathbf{x}_q$ are human speech recorded with background noise. The signals $\mathbf{y}_q$ are the same speech files recorded without background noise. Each of these audio signals contains $N = 32000$ samples recorded with a sampling time of $T_s = 17\mu s$.

Play sample speech recordings for clean and contaminated audio. ■

It is clear that this is a problem that we can formulate as an empirical risk minimization (ERM). Given audio inputs $\mathbf{x}$ contaminate with background noise we want to produce estimates $\hat{\mathbf{y}}$ of the corresponding clean audio signals $\mathbf{y}$. Using a learning parametrization that produces estimates $\hat{\mathbf{y}} = \mathbf{\Phi}(\mathbf{x}; \mathbf{H})$ we want to find the parameter $\mathbf{H}$ that minimizes the empirical risk over the Q audio pairs for a given loss function $\ell(\mathbf{y}, \hat{\mathbf{y}})$,

$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \mathbf{\Phi}(\mathbf{x}_q; \mathbf{H}) \Big). \tag{2}$$

It is interesting that this mathematical formulation is searching for an artificial intelligence (AI) that is trying to *undo* a natural effect – rather than mimicking a natural effect. This point is emphasized in Figure 2. The original data are clean audio inputs $\mathbf{y}_q$. These data are contaminated with background noise to produce the audio files $\mathbf{x}_q$. Our AI takes as inputs these signals contaminated with background noise and attempts to estimate the clean audio signals that generated the data contaminated with background noise. This difference in interpretation of the role of the AI does not alter the mathematical formulation of the ERM problem.

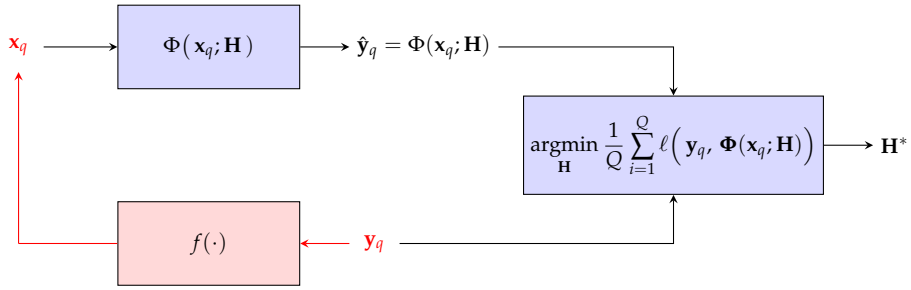

Figure 2. Elimination of background noise. We can formulate noise reduction as an empirical risk minimization problem [cf. (3)]. While it does not change the mathematical formulation, it is interesting that the artificial intelligence undoes the action of the world rather than mimic the world.

Throughout this lab we will use L_1 losses to compare clean audio signals and their estimates. The L_1 loss is defined as the sum of the absolute values of individual component differences. If we write $\mathbf{y} = [y_0; \ldots; y_{N-1}]$ and $\hat{\mathbf{y}} = [\hat{y}_0; \ldots; \hat{y}_{N-1}]$ the L_1 loss is given by

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) \;=\; \big|\, \mathbf{y} - \hat{\mathbf{y}} \,\big| \;=\; \sum_{n=0}^{N-1} \big|\, y_n - \hat{y}_n \,\big|. \tag{3}$$

We are now ready to make a first attempt at training an AI to remove background noise from audio signals.

Task 2 Split the data loaded in Task 1 into a test set with $\tilde{Q} = 100$ samples and a training set containing the remaining samples. Use as learning parametrization a neural network with $N_1 = 100$ hidden neurons in a single layer.

Evaluate the training and test error. Play some sample speech recordings of entries of the test dataset after they are cleaned with the neural network. You are likely surprised by the sound. Comment. ■

The neural network in Task 2 has failed to clean the audio files. This is a surprise because we have seen neural networks work in Lab 1C and the problem formulation in (3) is essentially the same problem formulation. What has changed between Lab 1C and Lab 2A is dimensionality. In Lab

1Aa we were processing signals with $N = 2$ components. We are now processing signals with $N = 32,000$. The complexity of the data is such that a neural network fails to learn any meaningful action that may result in removal of background noise.

This phenomenon is typical. When we consider problems in which the input dimension is small pretty much any learning parametrization works fine. Neural networks, in particular, work well and have become a defacto standard. When we consider signals in high dimensions, not all parameterizations work well. Finding good parameterizations requires that we leverage the structure of the signal. In the case of signals in time, this is done with convolutions.

2 Convolutions

Convolutions are linear operations that we use to process time signals. Consider then an input signal $\mathbf{x}$ with N components $x(n)$ along with a *filter* $\mathbf{h}$ having K coefficients $h(k)$. The convolution of the filter $\mathbf{h}$ with the signal $\mathbf{x}$ is a signal $\mathbf{y} = \mathbf{h} * \mathbf{x}$. This signal has N components $x(n)$ which are given by

$$y(n) = \sum_{k=0}^{K-1} h(k)x(n-k), \tag{4}$$

In this definition we adopt the convention that $x(n-k) = 0$ whenever the argument $(n-k) \notin [0, N-1]$. This is needed because for some values of n and k we may have that $n-k$ is outside of the range $[0, N-1]$ – for example, when $n = 0$ and $k > 0$. When this happens $x(n-k)$ is not defined. It follows that without the adoption of some convention for these values of the index $(n-k)$ the definition in (4) is improper. This convention is called a border artifact and it is not a significant issue if $K \ll N$. This is not required in the definition in (4) but it is almost always true in practice.

2.1 Shift Sequences

Convolutions can be equivalently written in terms of shift operators. This is a more abstract yet more illuminating definition that is also easier to

implement. As its name indicate, a shift operator shifts a signal in time. Formally, a shift operator $\mathcal{S}$ acts on a signal $\mathbf{x}$ to produce the signal $\mathbf{z} = \mathcal{S}\mathbf{x}$ whose components are

$$\mathbf{z}_1 = \mathcal{S}\mathbf{x} \quad \Longrightarrow \quad \mathbf{z}_1 = [0, x(0), x(1), \dots x(N-2)]. \tag{5}$$

The signal $\mathbf{z}_1$ has the same entries as the signal $\mathbf{x}$ but the entries of $\mathbf{x}$ have moved one time index up. The $n-1$st component of $\mathbf{x}$ is the nth component of $\mathbf{z}_1$. There is a border effect for $n = -1$, but we can maintain the interpretation if we maintain the same convention that $x(-1) = 0$.

The shift operator can be applied repeatedly. If we apply the shift operator to $\mathbf{z}_1$ we end up with the signal

$$\mathbf{z}_2 = \mathcal{S}\mathbf{z}_1 = \mathcal{S}^2\mathbf{x} \quad \Longrightarrow \quad \mathbf{z}_2 = [0, 0, x(0), x(1), \dots x(N-2)]. \tag{6}$$

This signal has the same components of $\mathbf{z}_1$ but they have moved one time index up. Since these were the entries of $x(n)$ moved one index up, the entries of $\mathbf{z}_2$ are the same entries of $\mathbf{x}$ but moved 2 time indexes up.

We can repeat application of the shift operator however number of times we want. These results in a sequence of shifted signals that we define recursively as

$$\mathbf{z}_k = \mathcal{S}\mathbf{z}_{k-1} = \mathcal{S}^k\mathbf{x} \quad \Longrightarrow \quad \mathbf{z}_k = [0, \dots, 0, x(0), x(1), \dots x(N-k)]. \tag{7}$$

This is a signal in which the components of $\mathbf{x}$ are moved k time indexes up, with the convention that we add k zeros at the beginning of the vector $\mathbf{z}_k$ to account for the entries of $\mathbf{x}$ that are not defined.

The collection of signals $\mathbf{z}_k$ is called the shift sequence. If we further denote $\mathbf{z}_0 = \mathbf{x} = \mathcal{S}^0\mathbf{x}$ we have a collection of shifted versions of $\mathbf{x}$. The signal $\mathbf{z}_0$ is a 0-shifted version of $\mathbf{x}$, the signal $\mathbf{z}_1$ is a 1-shifted version of $\mathbf{x}$, and the signal $\mathbf{z}_2$ is a 2-shifted version of $\mathbf{x}$. In general, the signal $\mathbf{z}_k$ is a k-shifted version of $\mathbf{x}$. Using the definition of the shift sequence we can equivalently write convolutions as

$$\mathbf{y} = \mathbf{h} * \mathbf{x} = \sum_{k=0}^{K-1} h(k)\mathbf{z}_k = \sum_{k=0}^{K-1} h(k)\mathcal{S}^k\mathbf{x}. \tag{8}$$

The expression in (8) says that convolutions are linear combinations of components of the shift sequence. That is, linear combinations of shifted versions of the input signal $\mathbf{x}$.

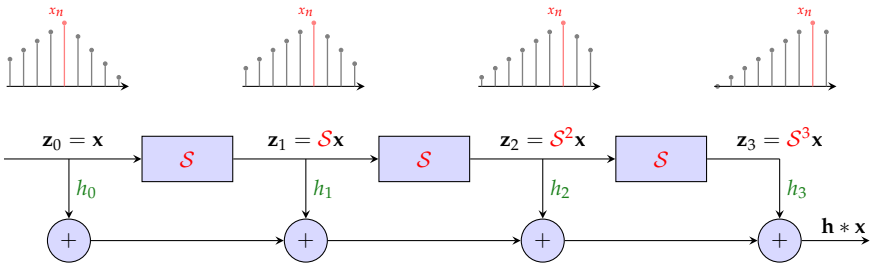

Figure 3. Convolutions. This block diagram representation of a convolution is called a shift register. It represents convolutions as linear combinations of shifted versions of the input signal $\mathbf{x}$

2.2 Shift Registers

A block diagram representation of (8) is shown in Figure 3. This is called a shift register. The input to the shift register is a signal $\mathbf{x}$. The signal is fed to a chain of shift operators $\mathcal{S}$. Each of these shift operators delays the signal by one unit of time. All of the resulting delayed signals contain the same components of $\mathbf{x}$ but their positions are shifted. Thus after passing through k shift operators we have the signal $\mathbf{z}_k = \mathcal{S}^k\mathbf{x}$ in which the $(n-k)$th component of the original input $\mathbf{x}$ has been moved to time position n. That is,

$$(\mathcal{S}^k\mathbf{x})_n = z_k(n) = x(n-k) \tag{9}$$

The convolution $\mathbf{h} * \mathbf{x}$ is obtained by taking the outputs of each delay element, scaling them by the filter coefficient h_k and summing them up.

Task 3 Implement a class to represent convolutional filters with K taps. In this class K is an initialization parameter and the filter taps h_k are class attributes. Endow the class with a forward method that takes a signal $\mathbf{x}$ as an input and produces as an output the signal $\mathbf{y} = \mathbf{h} * \mathbf{x}$.

To implement the forward method of this class you can use (4) or (8), but the latter is more efficient. You need to pay attention to border effects. ■

3 Locality and Shift Equivariance

Convolutions have two properties that make them useful in the processing of signals in time: locality and equivariance.

We say that convolutions are local because the output $y(n)$ depends only on input signal values $x(n-k)$ with $k \leq K$ [cf. (4)]. Thus, the output $y(n)$ does not depend on input values that are farther away than K time units. Since we often choose $K \ll N$, this means that only a comparatively small number, $K+1$, of input components affect each output component. Even if K is close to N it is worth pointing that (4) explicitly depends on the time difference $n-k$ and thus, nearby components of the input are processed with a coefficient that is explicitly different from the coefficient that processes far away information. This matches the intution that nearby and far away information are explicitly different.

We say that convolutions are equivariant to shifts because a shift of the input signal $\mathbf{x}$ is equivalent to a shift of the output signal $\mathbf{y}$. This is formally stated next

Proposition 1 *Given a filter* $\mathbf{h}$ *and an input signal* $\mathbf{x}$ *let* $\mathbf{y} = \mathbf{h} * \mathbf{x}$ *be the convolution of* $\mathbf{h}$ *and* $\mathbf{x}$ *[cf. (8)]. If* $\mathbf{x}_s = \mathcal{S}\mathbf{x}$ *is a shifted version of* $\mathbf{x}$ *the convolution output* $\mathbf{y}_s = \mathbf{h} * \mathbf{x}_s$ *is the corresponding shifted version of* $\mathbf{y}$*,*

$$\mathbf{y}_s = \mathbf{h} * \mathbf{x}_s = \mathbf{h} * (\mathcal{S}\mathbf{x}) = \mathcal{S}(\mathbf{h} * \mathbf{x}) = \mathcal{S}\mathbf{y} \tag{10}$$

Proof: This is easiest to see from the shift operator expression in (8). Using this expression we write the convolution between $\mathbf{h}$ and the shifted signal $\mathbf{x}_s$ as

$$\mathbf{y}_s = \mathbf{h} * \mathbf{x}_s = \sum_{k=0}^{K-1} h(k)\mathcal{S}^k \mathbf{x}_s. = \sum_{k=0}^{K-1} h(k)\mathcal{S}^k(\mathcal{S}\mathbf{x}). \tag{11}$$

We now use the fact that applying k shifts to a signal to which shift has been applied is the same as shifting the signal by $k+1$ time units. We therefore have

$$\mathbf{y}_s = \sum_{k=0}^{K-1} h(k)\mathcal{S}^{k+1}\mathbf{x}. \tag{12}$$

Observe now that *all* the summands in the right hand side of (12) have at least one shift. Further observing that a sum of shifted signals is the

same as shifting a sum we can pull out one shift as a common factor,

$$\mathbf{y}_s = \mathcal{S}\left(\sum_{k=0}^{K-1} h(k)\mathcal{S}^k \mathbf{x} \right). \tag{13}$$

To conclude the proof just observe that the sum in (14) is the convolution $\mathbf{y} = \mathbf{h} * \mathbf{x}$. Using this fact yields,

$$\mathbf{y}_s = \mathcal{S}(\mathbf{h} * \mathbf{x}) = \mathcal{S}\mathbf{y}. \tag{14}$$

This is the expression in (10) that we wanted to prove. ■

Proposition 1 shows that convolutions and shift operators commute. It is equivalent to shift a signal before it is processed by a convolution or after it is processed by a convolution. This is why we say that convolutions are equivariant to time shifts.

Shift equivariance is a fitting property because the processing of a time signal should be independent of the time index. Time $t = 0$ marks the time at which we commence observation and is therefore arbitrary. Shift equivariance also implies that the way in which different components are processed is the same. This is useful in learning because whatever we observe at a particular point in time is used to process the signal at that particular time instance and it is also used to process the signal at other time instants. This has a multiplier effect on the number of examples that we are using in the empirical risk.

Task 4 Split the data loaded in Task 1 into a test set with $\tilde{Q} = 100$ samples and a training set containing the remaining samples. Use the class of Task 3 to train a convolutional filter with $K = 10$ taps to remove background noise.

Evaluate the test error. Play some sample speech recordings of entries of the test dataset after they are cleaned with the neural network. You are likely surprised by the sound. Comment. ■

Contrary to the fully connected neural network of Task 2, this simple linear convolutional filter works well. This illustrates the importance of leveraging signal structure in machine learning. In Lab 2B we will see how to use convolutions to construct convolutional neural networks.

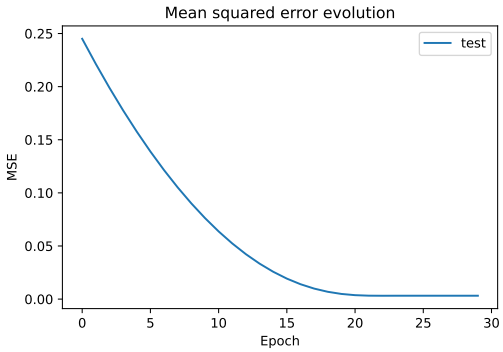

Figure 4. Test error evolution for convolutional filter.

4 Report

Do not take much time to prepare a lab report. We do not want you to report your code and we don't want you to report your work. Just give us answers to questions we ask. Specifically, give us the following:

Question	**Report deliverable**
Task 1	Do not report
Task 2	Report training loss
Task 2	Report test loss
Task 2	Paragraph with conclusions drawn from the fact that this neural network does *not* work
Task 3	Do not report
Task 4	Report test loss
Task 4	Paragraph with conclusions drawn from the fact that this neural network *does* work

We will check that your answers are correct. If they are not, we will get back to you and ask you to correct them. As long as you submit responses, you get an A for the assignment. It counts for 10% of your lab grade.