MALCOM: Machine Learning for Communication Systems **(Due: 12/04/2021)**

Lab Assignment #1

Instructor: Marios Kountouris ,

Lab Preparation: The first lab session is on autoencoders (deep learning) and we will use TensorFlow and Keras for implementing them. For that, you will need to install them in your machines (assuming that you have already had Python installed). There are several ways to do that, i.e., TensorFlow and Keras documentatin at https://keras.io/. I recommend working with Anaconda platform (https://www.anaconda.com/) and install Tensorflow/Keras through it.

Once you have installed Anaconda, you open an Anaconda command prompt terminal window and you type:
conda create -n tf tensorflow
and once completed (use 'y' yes to install all packages) you type:
conda activate tf
This installs the CPU-only TensorFlow.

To install Keras, you type:
conda install -c anaconda keras
Once you are done, you will need to do few installations again (since this is an new environment). If you use jupyter and Spyder from Anaconda, you will need to re-install them either from the main page buttons, or typing: *conda install jupyter* and *conda install spyder*.

Another way to install TensorFlow and Keras is through Python (PyPI), typing:
pip install –upgrade tensorflow for TensorFlow and *pip install keras* for Keras.

Autoencoders for End-to-End Communication Systems

The objective of the lab session is to experiment with an autoencoder using Keras in Tensorflow for end-to-end communications.

Communication system model (recap): A simple communication system consists of a transmitter, a channel, and a receiver (see Fig. 1). The transmitter wants to communicate one out of M possible messages $s \in \mathcal{M} = \{1, 2, ..., M\}$ to the receiver making n discrete uses of the channel. To this end, it applies the transformation $f : \mathcal{M} \mapsto \mathbb{R}^n$ to the message s to generate the transmitted signal $\mathbf{x} = f(s) \in \mathbb{R}^n$. For simplicity, we focus here on real-valued signals only. Alternatively, one can consider a mapping to $\mathbb{R}^{2n}$, which can be interpreted as a concatenation of the real and imaginary parts of $\mathbf{x}$. Generally, the hardware of the transmitter imposes certain constraints on $\mathbf{x}$. The communication rate of this communications system is $R = \frac{k}{n}$ bit/channel use, where $k = \log_2(M)$. We use the notation (n,k) to represent a communication system that sends one out of $M = 2^k$ messages (i.e., k bits) through n channel uses. The channel is described by the conditional probability density function $p(\mathbf{y}|\mathbf{x})$, where $\mathbf{y} \in \mathbb{R}^n$ denotes the received signal. Upon reception of $\mathbf{y}$, the receiver applies the transformation $g : \mathbb{R}^n \mapsto \mathcal{M}$ to produce the estimate $\hat{s}$ of the transmitted message s.

$$\begin{tikzcd} s \arrow[r] & \boxed{\text{Transmitter}} \arrow[r, "\mathbf{x}"] & \boxed{\text{Channel}} \arrow[r, "\mathbf{y}"] & \boxed{\text{Receiver}} \arrow[r] & \hat{s} \end{tikzcd}$$

Figure 1: A simple communications system

Deep learning view: this simple communication system can be seen as an *autoencoder*. Typically, the goal of an autoencoder is to find a low-dimensional representation (dimensionality reduction) of its input at some intermediate layer which allows reconstruction at the output with minimal error. In this way, the autoencoder learns to non-linearly compress and reconstruct the input. In communication systems, the autoencoder seeks to learn representations $\mathbf{x}$ of the messages s that are robust with respect to the channel impairments mapping $\mathbf{x}$ to $\mathbf{y}$ (i.e., noise, fading, distortion, etc.), so that the transmitted message can be recovered with small probability of error. This is a particular case of autoencoders as instead of removing redundancy from input data for compression, it adds redundancy, learning an intermediate representation robust to channel perturbations.

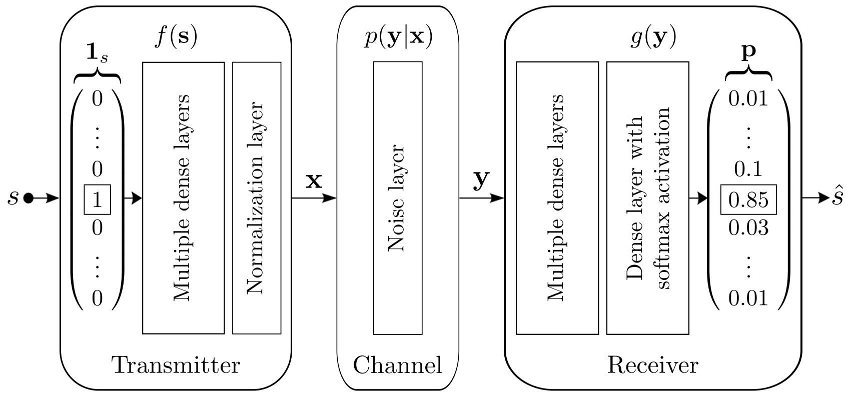

Figure 2: Autoencoder representation of a communications system over an AWGN channel.

Layer	Output dimensions
Input	M
Dense + ReLU	M
Dense + linear	n
Normalization	n
Noise	n
Dense + ReLU	M
Dense + softmax	M

Table 1: Layout of the autoencoder for the assignment.

Implementation: In this lab session, we will implement the simple autoencoder shown in Fig. 2.

The transmitter consists of a feedforward neural network with multiple dense layers followed by a normalization layer that ensures that physical constraints on $\mathbf{x}$ are met. In the code provided, an energy constraint is implemented ($\|\mathbf{x}\|_2^2 = n$). In the first problem to solve, you will only need to implement two dense layers. The first layer uses a ReLU activation function and the second layer uses a linear activation function.

The input s to the transmitter is encoded as a *one-hot vector* $\mathbf{1}_s \in \mathbb{R}^M$, i.e., an M-dimensional vector, the sth element of which is equal to one and zero otherwise.

The channel is represented by an additive noise layer with a fixed variance $\beta = (2RE_b/N_0)^{-1}$, where E_b/N_0 denotes the energy per bit (E_b) to noise power spectral density (N_0) ratio.

The receiver is implemented as a feedforward neural network. For the first question/implementation, your decoder will consist of a dense layer with ReLU activation followed by another dense layer that uses a softmax activation. The output of this last layer $\mathbf{p} \in (0,1)^M$ is a probability vector over all possible messages. The decoded message $\hat{s}$ corresponds then to the index of the element of $\mathbf{p}$ with the highest probability.

Training: the autoencoder is trained end-to-end using stochastic gradient decent (SGD) on the set of all possible messages $s \in \mathcal{M}$ using the categorical cross-entropy loss function between $\mathbf{1}_s$ and $\mathbf{p}$. Training will be performed at a fixed value of $E_b/N_0 = 7$ dB using Adam optimizer with learning rate 0.001.

The layout of the autoencoder is provided in Table 1. It has $(2M+1)(M+n)+2M$ trainable parameters, resulting in 62, 791, and 135,944 parameters for the (2,2), (7,4), and (8,8) autoencoder, respectively.

Assignment

(a) Complete the autoencoder implementation (.py file provided) by implementing the transmitter (encoder) and the receiver (decoder). As said before, the encoder will have two dense layers, the first with ReLU activation and the second with Linear activation. For that use the function Dense from Tensorflow/Keras. For the decoder, you will implement two dense layers, the first one with activation function ReLU and the second one with Softmax. Place your code in the sections marked as 'Your code starts here'.

(b) To test if you implementation works, run the code for a (2,2) and a (2,4) autoencoder. The file outputs three plots, the learned constellation diagram, the model loss vs. epoch and the block error rate (BLER), i.e., $\Pr(\hat{s} \neq s)$, of the communications system for different for E_b/N_0 [in dB] values. Comment on the learned constellation. For example, how different are these constellations from known constellations (such as QPSK and 16-PSK)?

(c) Repeat the same as in **(b)** for an autoencoder(2,4) (only) but implementing an average power constraint ($\mathbb{E}\left[|x_i|^2\right] \leq 1\, \forall i$) instead of a fixed energy constraint (that forces the symbols to lie on the unit circle). Plot the three plots again and compare the results with those in **(b)**.

(d) We will explore here the effect of the activation function on the performance. Do you think that the linear activation in the second layer is necessary/important or it can be replaced by other activation functions (e.g. ReLU, tanh)? How about the activation of the output layer (softmax)? Can we replace it with some other activation function (if so which one?) without affecting the performance?

Then, replace the ReLU activation with linear in the first layer. What happens to the performance? What is the intuition behind using a linear activation instead of the widely used ReLU? Hint: think in terms of PCA (Principal Component Analysis) and what a linear function means for the autoencoder.

(e) We will explore here the *effect of depth* (number of dense layers) on the performance. Implement the autoencoder with one dense layer instead of two (at both transmitter and receiver). Compare the performance and comment on whether two layers are necessary as compared to one layer. What happens if we increase the number of layers. Do you expect that the performance will be significantly improved. Comment your answer.

(f) Going back to the initial implementation (as in (a)) and compare the BLER performance of Autoencoder(2,2), Autoencoder(2,4), and Autoencoder(4,2). Comment your results and the ranking between the different autoencoders.

(g) We will explore now the *effect of the optimization technique.* What happens if instread of Adam optimizer we use: SGD, RMSProp, or Adadelta. Do you see any significant difference in the performance? Comment your answer.

(h) (Bonus-optional) Compare/plot the BER performance vs. E_b/N_0 of an Autoencoder(7,4) (fixed energy constraint) with the performance of a Hamming code (7,4) with hard decision decoding and the theoretical BER performance of a BPSK modulation. Comment your results. Hint: you can use MATLAB's functions 'bercoding' and 'berawgn'.