Distortion Correction for Modulation Recognition

Amey Anjarlekar 170070013
Vedant Satav 170070012
Preetam Pinnada 170070042
Kevin Arvadia 170070001

I. Abstract

Modulation is a process by which an information-carrying signal is transformed onto a periodic wave called as the carrier wave. For a given set of signals, modulation recognition forms the first step to perform spectrum sensing in a cognitive radio set-up

II. About the Data set

The data set is a RadioML deepsig data set titled **'RML2016.10a-dict'** which is a python dictionary. The keys are given by ['Modulation type','SNR'].While there are 11 types of modulations (and hence, there are **11 classes**) that we deal in this code, SNR stands for Signal-to-noise ratio which varies from a value of $-20dB$ to $+20dB$ in steps of 2. A signal can be thought of as having a real and imaginary part. We have the information for 128 instances of each signal. At the same time, we have 1000 examples of signals of each modulation type. Hence each key maps to a **3D list** which is of the size 1000x2x128. The modulation recognition problem has been converted into a **classification problem** - given the real part and the imaginary part of a signal, classify the signal into its modulation type. This data set has been provided to us by **Prof. Prasanna Chaporkar**, Electrical Engineering Department

III. Imported libraries

For the purpose of execution, we have imported the following packages-

- pickle - The data set is pickle file which needed to be extracted with the help of **pickle.load()**
- numpy - it is a library which provides additional utility for multidimensional matrices and arrays along with high level mathematical functions
- keras - open-source neural network library written in python which was used for the implementation of the convolutional neural network

IV. Logic

The pickle file is extracted and the dictionary keys and values are stored as **separate numpy arrays**. Out of the 1000 examples that we have, **800** have been used as **training set** while the remaining serve the testing data set, and hence the values are again split into two arrays.
The 11 classes as **hot encoded** as follows-
If it is QAM64 type modulation, we denote it by [0,0,0,0,0,0,0,0,1,0,0]
If it is QPSK type, we denote it by [0,0,0,0,0,0,0,0,0,1,0]
Similarly with others. These training and testing arrays are once again converted into numpy arrays.
We now implement the standard convolution neural network to estimate the result by assigning **softmax probability** to each of the classes.
Convolution Neural network begins:
The Sequential model is a linear stack of connected layers. The layers that we use are Dense, Convolution2D, Flatten, Dropout, Activation, Reshape and ZeroPadding2D
The 2D 2x128 matrix is first reshaped into a 3D 1x2x128 matrix for better efficiency of kernels later and two columns of padding are added to the right most edge of the 3D matrix.

*	*	0	0
*	*	0	0

TABLE I: The type of padding we are aiming for

A set of 256 filters, each of shape [1x3], are convoluted with this matrix such that the padding type is "valid" and activation function used in ReLU. Dropout(rate) sets rate fraction of input to zero to avoid over-fitting. Another convolution layer is added consisting of 80 [1x3] filters and the same process is carried out again. BatchNormalization() normalizes all the elements of the filters. Dense(units) is a connected NN-layer with an output space of 'units'. Activation function, if unspecified, is taken to be linear. The set os examples is broken down randomly into various batches of size 1024 for better efficiency.

Evaluate() returns the loss in the predicted output for a given batch of inputs.

V. Results

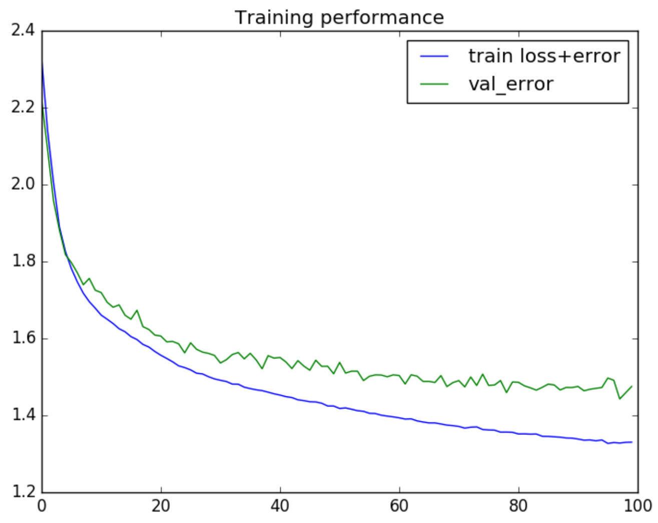

Fig. 1

We can see from the figure that the loss in the prediction reduces by a alarge amount once we start increasing the number of epochs.