RADIO MODULATION RECOGNITION USING DEEP LEARNING

PROJECT

ECE DUAL (INTEGRATED)

Submitted by

Animesh Srivastava **15MI445**

Sumit Kumar **15MI446**

To

Dr. Philemon Daniel

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
HAMIRPUR-177005, HP (INDIA)

NOVEMBER 2019

Radio Modulation Recognition using Deep Learning

Submitted By: Animesh Srivastava 15MI445

Sumit Kumar 15MI446

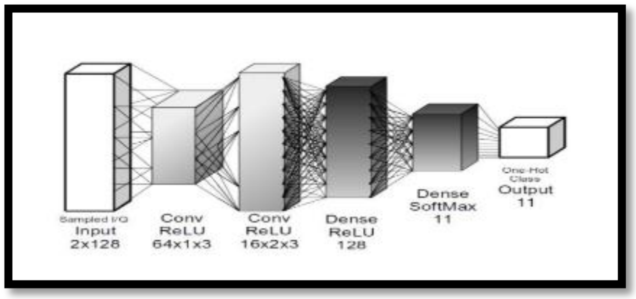
Objective

The generation of dataset consisting of radio signals modulated using different modulation techniques and with different signal to noise ratios. The task is to try different neural network architectures to classify the signals based on the modulation technique that is used.

Methodology

The steps taken to fulfill the above objective are as follows:

1. Generation of variable SNR dataset consisting of eleven different modulations. The dataset is modified with different fading and labeled SNR increments.
2. Creation of the Feature Space. Every sample is presented using two vectors each of 128 elements.
3. Supervised learning on the dataset. Split the dataset into 50% training/validation and 50% testing.
4. Use of a Fully Connected Neural Network. Use of ReLU and ADAM with default parameters. We will also use an early stopping on a validation set of size 5% of dataset
5. Application of the below shown CNN.



6. We focus on the following end result:
   a) Plot of the accuracy against the SNR.
   b) Display of the Confusion matrix and the most confusing class.

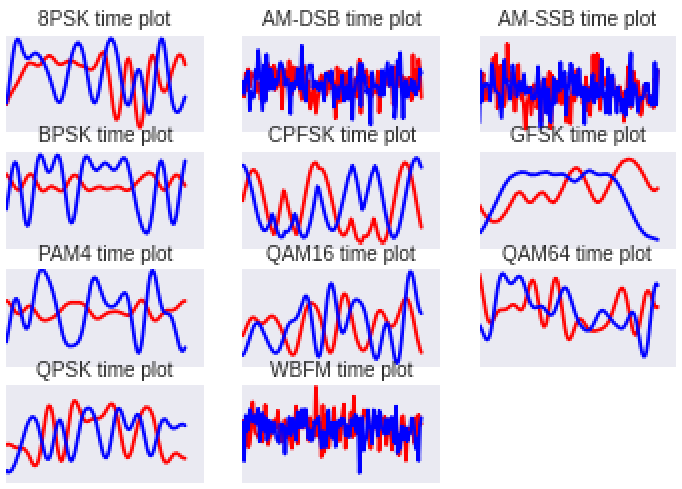
Data Generation

a) Data Generation Code

The detailed description of the code is done below:

```python
'''
Submitted by:
Animesh Srivastava (15MI445) Sumit Kumar (15MI446)
'''
#!/usr/bin/env python
from transmitters import transmitters
# We use transmitters to import the transmitters category
from source_alphabet import source_alphabet
# We source_alphabets to import various source alphabets that can
# be used
import timeseries_slicer
#pandas represent the timeseries dataset as series in python
import analyze_stats
#it is used for analyzing the status
'''
For example
def analyze_stats(filename):
    data = np.loadtxt(fname=filename, delimiter=',')
    return data.mean(0), data.max(0), data.min(0)

'''
from gnuradio import channels, gr, blocks
#import all the necessary channels, gr and mod blocks from gnu radio
import numpy as np
import numpy.fft, cPickle, gzip

'''
We work on to generate dataset with dynamic channel model across various range of SNRs
'''

apply_channel = True
output = {}
min_length = 9e9
snr_vals = range(-20,20,2)
for snr in snr_vals:
    for alphabet_type in transmitters.keys():
        print alphabet_type
        for i,mod_type in enumerate(transmitters[alphabet_type]):
            print "running test", i,mod_type

            tx_len = int(10e3)
            if mod_type.modname == "QAM64":
                tx_len = int(30e3)
            if mod_type.modname == "QAM16":
                tx_len = int(20e3)
            src = source_alphabet(alphabet_type, tx_len, True)
            mod = mod_type()
            fD = 1
            delays = [0.0, 0.9, 1.7]
            mags = [1, 0.8, 0.3]
            ntaps = 8
            noise_amp = 10**(-snr/10.0)
            print noise_amp
            #noise_amp = 0.1
            chan = channels.dynamic_channel_model( 200e3, 0.01, 1e2, 0.01, 1e3, 8, fD, True, 4, delays, mags, ntaps, noise_amp, 0x1337 )

            snk = blocks.vector_sink_c()

            tb = gr.top_block()

            # connect blocks
            if apply_channel:
                tb.connect(src, mod, chan, snk)
            else:
                tb.connect(src, mod, snk)
            tb.run()

            modulated_vector = np.array(snk.data(), dtype=np.complex64)
            if len(snk.data()) < min_length:
                min_length = len(snk.data())
                min_length_mod = mod_type
            output[(mod_type.modname, snr)] = modulated_vector

print "min length mod is %s with %i samples" % (min_length_mod, min_length)
# trim the beginning and ends, and make all mods have equal number of samples
start_indx = 100
fin_indx = min_length-100
for mod, snr in output:
 output[(mod,snr)] = output[(mod,snr)][start_indx:fin_indx]
X = timeseries_slicer.slice_timeseries_dict(output, 128, 64, 1000)
cPickle.dump( X, file("moddata_dict.dat", "wb" ) )
X = np.vstack(X.values())
cPickle.dump( X, file("moddata.dat", "wb" ) )
```

b) Data Generated Visualized



A dictionary having 162060 samples each of 2 vectors each of length 128 representing time series representation of signals generated by GNU radio under different modulation techniques with different signal to noise ratios (SNRs).

c) Data Generation Explained

The following features are used for generation of dataset

i. snr_vals in range(-20,20,2)

ii. delays = [0.0, 0.9, 1.7]

iii. mags = [1, 0.8, 0.3]

iv. noise_amp = 10**(-snr/10.0)

v. for mod, snr in output:

```
    output[(mod,snr)] = output[(mod,snr)][start_indx:fin_indx]

 X = timeseries_slicer.slice_timeseries_dict(output, 128, 64, 1000)

cPickle.dump( X, file("moddata_dict.dat", "wb" ) )

 X = np.vstack(X.values())

 cPickle.dump( X, file("moddata.dat", "wb" ) )
```

Collection

Steps used in the Data Collection and generalizing it for use for ML problems:

1. The dataset we have created is generated with help of GNU Radio, consisting of 11 modulations. This is a variable-SNR dataset with moderate LO drift, light fading, and numerous different labeled SNR increments for use in measuring performance across different signal and noise power scenarios. The dataset created has a size of 1.2 GB.
2. The file is formatted as a "pickle" file which can be open for example in python by using cPickle.load(...)
3. We then load the data to Numpy Arrays
4. Split the data into 50% for training/validation and 50% for testing
5. Creation of Tensorflow Record for the dataset
6. Creation of Feature space for the dataset

7. Every sample is presented using two vectors each of them has 128 elements. Those two vectors are the in-phase and quadrature phase components of a sample.X array consists of those two raw features.
8. Integration Feature and Derivative Feature: One for the in-phase component and one for quadrature phase component.
9. Combination 1: Derivative Features + Raw Features (4 features)
10. Combination 2: Integration Features + Raw Features (4 features)
11. Combination 3: Integration Features + Derivative Features (4 features)
12. Combination 4: Integration Features + Derivative Features + Raw Features (6 features)

Explanation of the Features

Different types of features are used: first, we use raw data of time series of signals; second, features are the first derivative of the data; third, features are the integrals and the fourth features are a combination of the past 3 ones.

1. **Raw Time Series**

We will detect signals from their time representation showing value of each signal at a given time where each sample is represented by two vectors one for Quadrature and other for In-phase representation which is a way of representing signals in digital signal processing.

2. **First Derivative**

Since data is time series representation then we can get the derivative by getting the difference of each two consecutive samples for each channel separately and this generates the derivative. This is correct because the derivative is the slope of the tangent by definition and the difference between each two values is one second.

3. **Integral**

We will apply also integration process to our data-set by using scipy cumulative trapezoidal rule integration method as shown in the code below:

```
data_int=integrate. cumtrapz (data, initial=0)
```

In general, using same parameters (raw data case) found at setting our NN and applying them on the integrated version of data-set results in lower accuracy, about 50% maximum accuracy resulted @ 5 layers of NN, 256 input size for each layer, and batch size of 100.

Combined Data:

The data is combined by making more channels; in the previous features we used only 2 Channels but now we will use 6 channels two for each type of the first 3 feature types.
This will allow our NN to have more features to train on and hopefully more accuracy.
Differentiation features as a detailed use case:

1. **One hidden layer (dense):**

Having only one hidden layer with number of units {64,128,256} we observed that as we increase number of units accuracy increases (training, validation and testing).
But it is still low compared to multiple layered models with a maximum accuracy achieved of 21%.

2. **Two hidden layers (dense):**

Having two hidden layers with different sets of number of units,
1. We tried making all layers with the same number of units and then made them different trying all combinations to make sense which combinations give best accuracy.
2. We also tried different batch sizes to see their effect.
3. We fixed number of epochs to 100

<u>Network Layer</u>

Fully Connected Layer

1. The core data structure of Keras is a model. The simplest type of model is the Sequential model, a linear stack of layers.
2. A 3-layer deep neural network consisting only of fully connected layers of size 512, 256, 11 neurons

CNN

1. The core data structure of Keras is a model. The simplest type of model is the Sequential model, a linear stack of layers.
2. A 3-layer deep neural network consisting only of fully connected layers of size 512, 256, 11 neurons

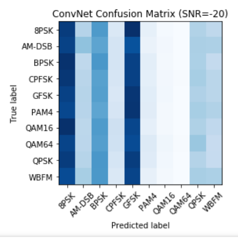
<u>Output Result</u>

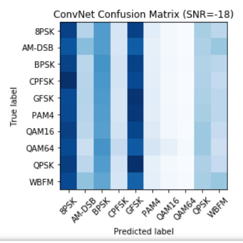
1. Fully Connected

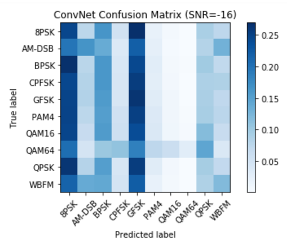
```
600000/600000 [==============================] - 110s 183us/sample - loss:
1.3647 - acc: 0.4587
```

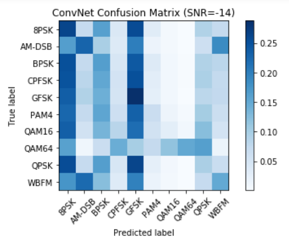
2. CNN

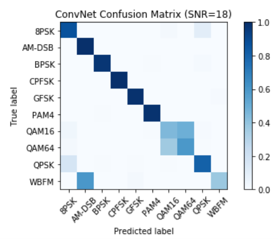
```
600000/600000 [==============================] - 77s 128us/sample - loss:
1.0965 - acc: 0.5454
```

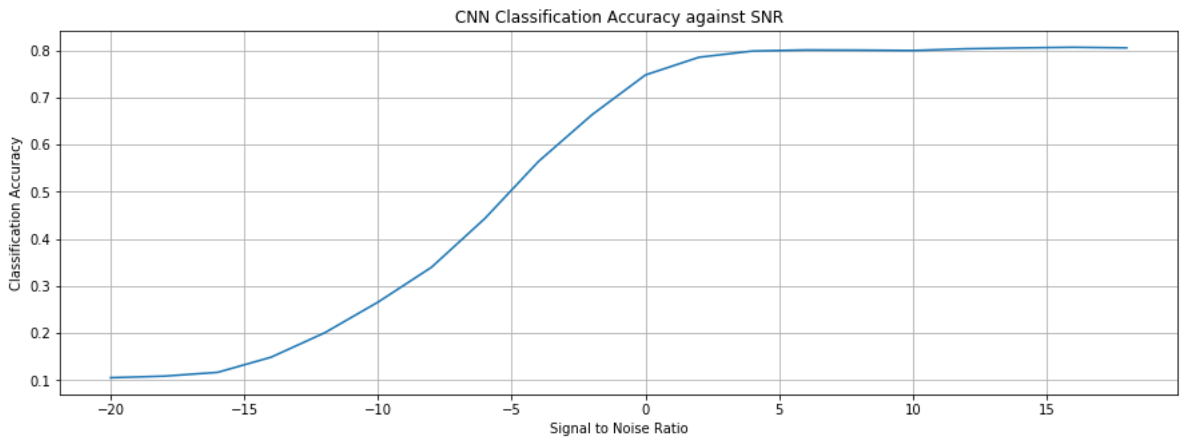

ConvNet Confusion Matrix (SNR=-20)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.25
0.20
0.15
0.10
0.05
0.00


ConvNet Confusion Matrix (SNR=-18)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.25
0.20
0.15
0.10
0.05


ConvNet Confusion Matrix (SNR=-16)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.25
0.20
0.15
0.10
0.05


ConvNet Confusion Matrix (SNR=-14)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.25
0.20
0.15
0.10
0.05


ConvNet Confusion Matrix (SNR=18)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
1.0
0.8
0.6
0.4
0.2
0.0


CNN Classification Accuracy against SNR
Classification Accuracy
Signal to Noise Ratio
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
−20
−15
−10
−5
0
5
10
15