ECG CLASSIFICATION USING DEEP LEARNING

PROJECT

ECE DUAL

Submitted by

Kavya Sharma **15MI405**

Utkarsh Khabla **15MI406**

To

Dr. Philemon Daniel

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

NOVEMBER 2019

Objective

To generate a signal and create a Recurrent Neural Network (RNN) for denoising a signal.

Methodology

Data Generation

1.Generation of a clean sine wave:

```
# Generating a clean sine wave
def sine(X,signal_freq=60.):
return np.sin(2*np.pi*(X)/signal_freq)
```

2. Uniform noise addition:

```
# Adding uniform noise
def noisy(Y,noise_range=(-0.35,0.35)):
noise=np.random.uniform(noise_range[0],noise_range[1],size=Y.shape)
return Y+noise
# Create a noisy and clean sine wave
def sample(sample_size):
random_offset=random.rand int(0,sample_size)
X=np.arange(sample_size)
out=sine(X+random_offset)
inp=noisy(out)
return inp,out
```

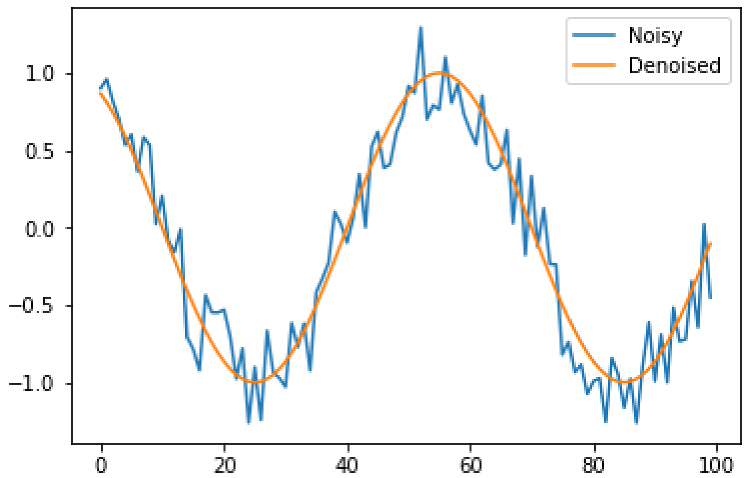

Fig. 1.Generated signal output

#Given a noisy sine wave as an input, we want to estimate the denoised signal.

3. Dataset Creation:

```
def create_dataset(n_samples=10000,sample_size=100):
data_inp=np.zeros((n_samples,sample_size))
data_out=np.zeros((n_samples,sample_size))

for i in range(n_samples):
sample_inp,sample_out=sample(sample_size)
data_inp[i,:]=sample_inp
data_out[i,:]=sample_out
return data_inp,data_out
```

Network Layers

1.Creating RNN Model

We have 1d sine waves, which we want to denoise. Thus, we have input dimension of 1. Let's create a simple 1-layer RNN with 30 hidden units.

```
input_dim=1

hidden_size=30

num_layers=1


class CustomRNN(nn.Module):

def__init__(self,input_size,hidden_size,output_size):

super(CustomRNN,self).__init__()

self.rnn=nn.RNN(input_size=input_size,hidden_size=hidden_size,batch_first=True)

self.linear=nn.Linear(hidden_size,output_size,)

self.act=nn.Tanh()

defforward(self,x):

pred,hidden=self.rnn(x,None)

pred=self.act(self.linear(pred)).view(pred.data.shape[0],-1,1)

return pred


r=CustomRNN(input_dim,hidden_size,1)
```

2.Bidirectional RNN Model

```
bidirectional=True

if bidirectional:

num_directions=2

else:
```

```
num_directions=1
class CustomRNN(nn.Module):
def__init__(self,input_size,hidden_size,output_size):
super(CustomRNN,self).__init__()
self.rnn=nn.RNN(input_size=input_size,hidden_size=hidden_size,
batch_first=True,bidirectional=bidirectional,dropout=0.1)
self.linear=nn.Linear(hidden_size*num_directions,output_size,)
self.act=nn.Tanh()
defforward(self,x):
pred,hidden=self.rnn(x,None)
pred=self.act(self.linear(pred)).view(pred.data.shape[0],-1,1)
return pred

r=CustomRNN(input_dim,hidden_size,1)
r
```

3. GRU Model

Let's now replace our RNN with GRU to see if the model improves.

```
bidirectional=True
if bidirectional:
num_directions=2
else:
num_directions=1
class CustomRNN(nn.Module):
def__init__(self,input_size,hidden_size,output_size):
super(CustomRNN,self).__init__()
self.rnn=nn.GRU(input_size=input_size,hidden_size=hidden_size,
batch_first=True,bidirectional=bidirectional,dropout=0.1)
self.linear=nn.Linear(hidden_size*num_directions,output_size,)
```

```
self.act=nn.Tanh()
```

Results

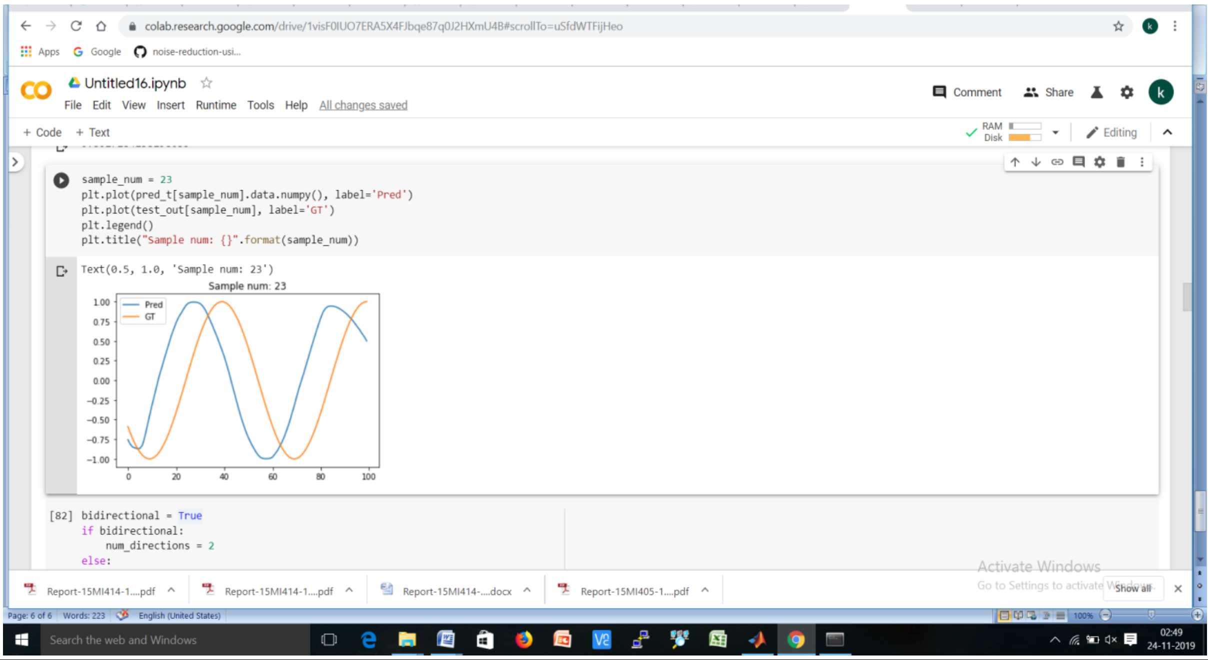

Fig. 2. RNN model output

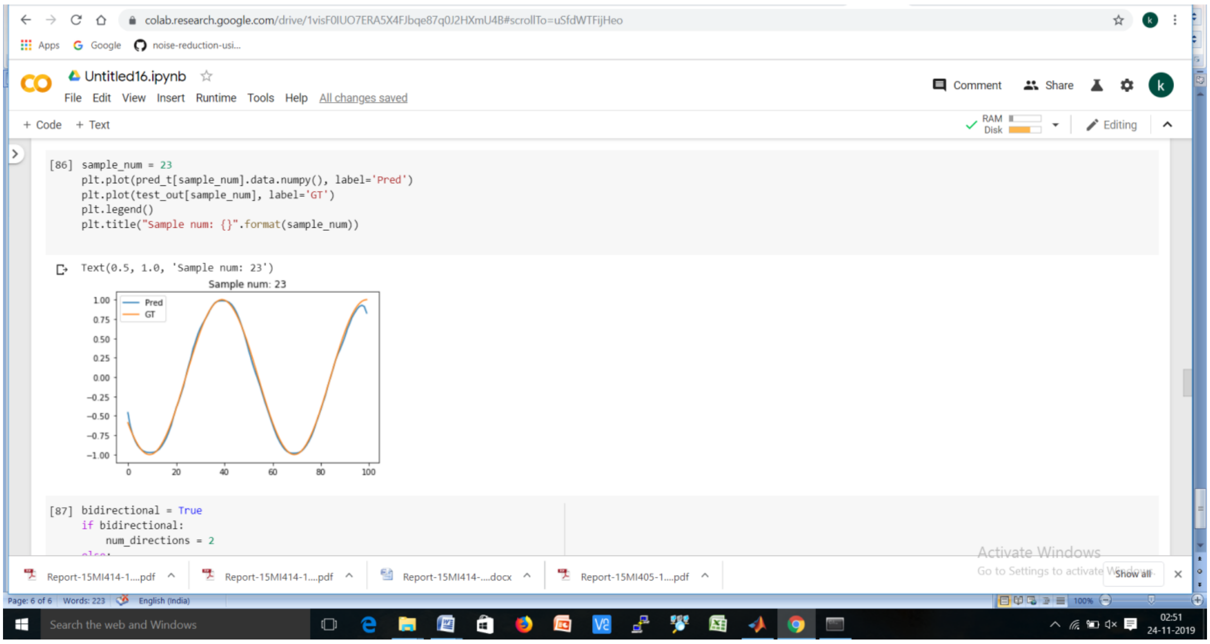

Fig. 3. Bidirectional RNN model output

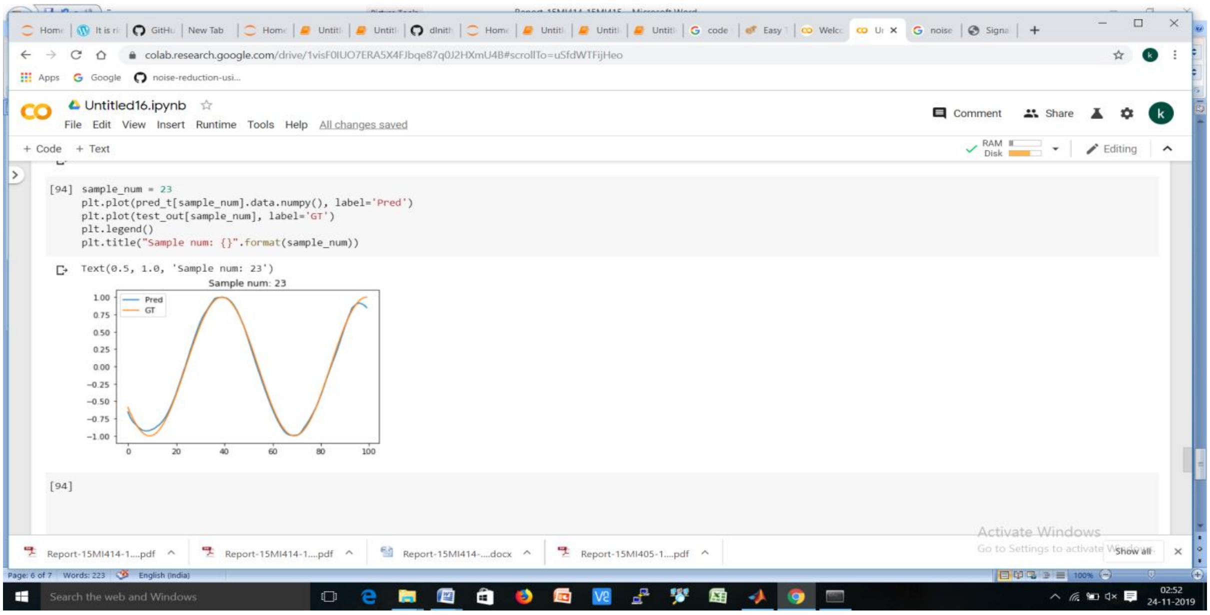

Fig. 4. GRU model output