8_pam_model

May 12, 2019

0.1 Importing Libraries required for the implementation

[0]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras.layers import *
from sklearn import preprocessing
import tensorflow.keras.backend as K
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
```

0.2 Hyper Parameters

[0]:
```
#length of message space
msg_total = 8
# number of channels
channel = 16
# number of epochs
epochs = 10000
# peturbation variance
sigma = 1e-4
# Batch size
batch_size = 1024
```

0.3 Defining Required Functions

[1]:
```
# Peturbation Sampling
def perturbation(x):
    w = K.random_normal(shape = (channel,2), mean=0.0,stddev=sigma**0.
 ↪5,dtype=None)
    xp = ((1-sigma)**0.5)*x + w
    return xp

# Defining transmitter loss
```

```
def loss_tx(y_true, y_pred):
    return -y_true*y_pred

# Defining the policy
def get_policy(inp):
    xp = inp[0]
    x = inp[1]
    w = xp - x
    policy = -K.sum(w*w)
    return policy
```

1 Transmitter Model

1.1 Defining Architecture

[4]:
```
tx_inp = Input((1,))
# Adding embedding layer
embbedings_layer = Dense(msg_total, activation = 'relu')(tx_inp)
layer_dense = Dense(2*channel, activation = 'relu')(embbedings_layer)
# real to complex
to_complex = Reshape((channel,2))(layer_dense)
# Normalising the output to unit energy
x = Lambda(lambda x: keras.backend.l2_normalize(x))(to_complex)
# Peturbation sampling
xp = Lambda(perturbation)(to_complex)
policy = Lambda(get_policy)([xp,x])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

1.2 Creating Required models (outputs from required layers)

[5]:
```
# model for policy training
model_policy = keras.models.Model(inputs=tx_inp, outputs=policy)
# model to get the peturbatated output
model_tx = keras.models.Model(inputs=tx_inp, outputs=xp)
# model to get the encoded message to transmit
model_x = keras.models.Model(inputs=tx_inp, outputs=x)

model_policy.compile(loss=loss_tx, optimizer=tf.keras.optimizers.SGD(lr = 1e-5))
print(model_policy.summary())
```

```
____________________________________________________________________________
__________________
Layer (type)                 Output Shape              Param #     Connected to
============================================================================
==================
input_1 (InputLayer)         (None, 1)                 0
____________________________________________________________________________
__________________
dense (Dense)                (None, 8)                 16          input_1[0][0]
____________________________________________________________________________
__________________
dense_1 (Dense)              (None, 32)                288         dense[0][0]
____________________________________________________________________________
__________________
reshape (Reshape)            (None, 16, 2)             0           dense_1[0][0]
____________________________________________________________________________
__________________
lambda_1 (Lambda)            (None, 16, 2)             0           reshape[0][0]
____________________________________________________________________________
__________________
lambda (Lambda)              (None, 16, 2)             0           reshape[0][0]
____________________________________________________________________________
__________________
lambda_2 (Lambda)            ()                        0           lambda_1[0][0]
                                                                   lambda[0][0]
============================================================================
==================
Total params: 304
Trainable params: 304
Non-trainable params: 0
____________________________________________________________________________
__________________
None
```

2 Receiver

2.1 Defining Architecture

[6]:
```
rx_inp = Input((channel,2))
# complex to real
to_flat = Reshape((2*channel,))(rx_inp)
fc = Dense(8*2*channel, activation = 'relu')(to_flat)
softmax = Dense(msg_total, activation = 'softmax')(fc)

model_rx = keras.models.Model(inputs=rx_inp, outputs=softmax)
```

```
model_rx.compile(loss=tf.keras.losses.categorical_crossentropy, optimizer=tf.
 ↪keras.optimizers.Adam())
print(model_rx.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 16, 2)             0
_________________________________________________________________
reshape_1 (Reshape)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               8448
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 2056
=================================================================
Total params: 10,504
Trainable params: 10,504
Non-trainable params: 0
_________________________________________________________________
None
```

2.2 Alternative Training

[7]:
```
loss_tx = []
loss_rx = []
for epoch in range(epochs):
#     trasmitter training
#     generating input
    raw_input = np.random.randint(0,msg_total,(batch_size))
#     Generating labels
    label = np.zeros((batch_size, msg_total))
    label[np.arange(batch_size), raw_input] = 1
    tx_input = raw_input/float(msg_total)
#     Transmitter prediction ( message encoding )
    xp = model_tx.predict(tx_input)
#     Adding noise ( modelling AWGN layer)
    y = xp + np.random.normal(0,0.001,(batch_size, channel,2))
#     Decoding the message
    pred = model_rx.predict(y)
#     Getting loss
    loss = np.sum(np.square(label - pred), axis = 1)
#     Transmitter model training
    history_tx = model_policy.fit(tx_input, loss, batch_size=batch_size,␣
 ↪epochs=1, verbose=0)
    loss_tx.append(history_tx.history['loss'][0])

#     Receiver Training
```

```
    raw_input = np.random.randint(0,msg_total,(batch_size))
    label = np.zeros((batch_size, msg_total))
    label[np.arange(batch_size), raw_input] = 1
    tx_input = raw_input/float(msg_total)
    x = model_x.predict(tx_input)
    y = x + np.random.normal(0,0.001,(batch_size, channel,2))
    history_rx = model_rx.fit(y, label, batch_size=batch_size, epochs=1,␣
 ↪verbose=0)
    loss_rx.append(history_rx.history['loss'][0])

#     Printing only after 100 epochs
    if(epoch % 100 == 0):
        print('epoch: ', epoch, 'tx_loss', history_tx.history['loss'][0],␣
 ↪'rx_loss', history_rx.history['loss'][0])
```

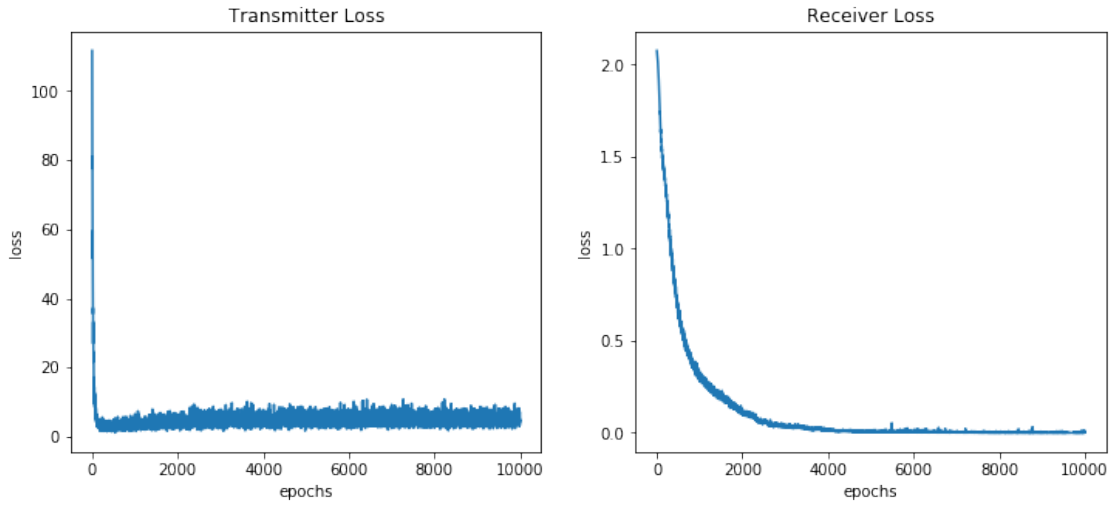
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
epoch:  0 tx_loss 111.65034484863281 rx_loss 2.075446128845215
epoch:  100 tx_loss 4.3518385887146 rx_loss 1.5956542491912842
epoch:  200 tx_loss 3.1458160877227783 rx_loss 1.3446216583251953
epoch:  300 tx_loss 2.183445930480957 rx_loss 1.0681068897247314
epoch:  400 tx_loss 2.750500440597534 rx_loss 0.8244391679763794
epoch:  500 tx_loss 3.4527690410614014 rx_loss 0.6615206003189087
epoch:  600 tx_loss 3.9687094688415527 rx_loss 0.5484654307365417
epoch:  700 tx_loss 4.75673246383667 rx_loss 0.44556933641433716
epoch:  800 tx_loss 2.3664300441741943 rx_loss 0.4057242274284363
epoch:  900 tx_loss 3.823850393295288 rx_loss 0.34978848695755005
epoch:  1000 tx_loss 4.562586784362793 rx_loss 0.3229084312915802
epoch:  1100 tx_loss 5.9120283126831055 rx_loss 0.2832150459289551
epoch:  1200 tx_loss 3.35794734954834 rx_loss 0.2697519361972809
epoch:  1300 tx_loss 2.7322275638580322 rx_loss 0.21645428240299225
epoch:  1400 tx_loss 4.766214370727539 rx_loss 0.21287575364112854
epoch:  1500 tx_loss 2.972025156021118 rx_loss 0.20674437284469604
epoch:  1600 tx_loss 3.2371692657470703 rx_loss 0.1805959790945053
epoch:  1700 tx_loss 3.3259565830230713 rx_loss 0.18627703189849854
epoch:  1800 tx_loss 2.6115152835845947 rx_loss 0.15560747683048248
epoch:  1900 tx_loss 4.833925247192383 rx_loss 0.13701024651527405
epoch:  2000 tx_loss 2.7226548194885254 rx_loss 0.12894144654273987
epoch:  2100 tx_loss 7.077600955963135 rx_loss 0.10896340012550354
epoch:  2200 tx_loss 4.855541229248047 rx_loss 0.09421651065349579
epoch:  2300 tx_loss 3.9910638332366943 rx_loss 0.10274013876914978
epoch:  2400 tx_loss 3.944584369659424 rx_loss 0.05820135027170181
epoch:  2500 tx_loss 4.085627555847168 rx_loss 0.06760348379611969
```

```
epoch:  2600 tx_loss 4.186509132385254 rx_loss 0.043666109442710876
epoch:  2700 tx_loss 4.096609592437744 rx_loss 0.039834339171648026
epoch:  2800 tx_loss 3.936901807785034 rx_loss 0.04208402335643768
epoch:  2900 tx_loss 3.0933609008789062 rx_loss 0.031584665179252625
epoch:  3000 tx_loss 2.5114285945892334 rx_loss 0.03643415868282318
epoch:  3100 tx_loss 3.968400716781616 rx_loss 0.03069741651415825
epoch:  3200 tx_loss 4.790694713592529 rx_loss 0.039969541132450104
epoch:  3300 tx_loss 5.108456611633301 rx_loss 0.03195992484688759
epoch:  3400 tx_loss 5.536928176879883 rx_loss 0.027466170489788055
epoch:  3500 tx_loss 5.189927577972412 rx_loss 0.023680424317717552
epoch:  3600 tx_loss 5.032960414886475 rx_loss 0.019849564880132675
epoch:  3700 tx_loss 5.154823303222656 rx_loss 0.01682446338236332
epoch:  3800 tx_loss 6.82951021194458 rx_loss 0.017644288018345833
epoch:  3900 tx_loss 6.10154914855957 rx_loss 0.013960335403680801
epoch:  4000 tx_loss 6.852365493774414 rx_loss 0.015919698402285576
epoch:  4100 tx_loss 4.90704345703125 rx_loss 0.015047567896544933
epoch:  4200 tx_loss 5.894028663635254 rx_loss 0.010916702449321747
epoch:  4300 tx_loss 3.0122227668762207 rx_loss 0.010609409771859646
epoch:  4400 tx_loss 4.682041168212891 rx_loss 0.012999525293707848
epoch:  4500 tx_loss 8.213058471679688 rx_loss 0.008963012136518955
epoch:  4600 tx_loss 4.626161575317383 rx_loss 0.007602107245475054
epoch:  4700 tx_loss 3.7452304363250732 rx_loss 0.007352523505687714
epoch:  4800 tx_loss 6.248739242553711 rx_loss 0.0069618928246200085
epoch:  4900 tx_loss 4.939318656921387 rx_loss 0.010771127417683601
epoch:  5000 tx_loss 4.491368293762207 rx_loss 0.008541966788470745
epoch:  5100 tx_loss 5.126115798950195 rx_loss 0.005239963997155428
epoch:  5200 tx_loss 5.8089118003845215 rx_loss 0.004610837437212467
epoch:  5300 tx_loss 2.4534215927124023 rx_loss 0.005705135874450207
epoch:  5400 tx_loss 6.818317890167236 rx_loss 0.006106524262577295
epoch:  5500 tx_loss 4.380892276763916 rx_loss 0.005216584540903568
epoch:  5600 tx_loss 3.577620506286621 rx_loss 0.006254799664020538
epoch:  5700 tx_loss 6.1088457107543945 rx_loss 0.002732411725446582
epoch:  5800 tx_loss 7.872372627258301 rx_loss 0.006918097846210003
epoch:  5900 tx_loss 5.95105504989624 rx_loss 0.0022812550887465477
epoch:  6000 tx_loss 4.555535793304443 rx_loss 0.0022870583925396204
epoch:  6100 tx_loss 4.640500068664551 rx_loss 0.010331875644624233
epoch:  6200 tx_loss 3.6990368366241455 rx_loss 0.0021935012191534042
epoch:  6300 tx_loss 4.898043155670166 rx_loss 0.0020838286727666855
epoch:  6400 tx_loss 5.4040446281433105 rx_loss 0.0017388788983225822
epoch:  6500 tx_loss 5.057991981506348 rx_loss 0.002344192937016487
epoch:  6600 tx_loss 6.815639019012451 rx_loss 0.002434772439301014
epoch:  6700 tx_loss 4.923605918884277 rx_loss 0.0021747329737991095
epoch:  6800 tx_loss 4.367935657501221 rx_loss 0.001837260671891272
epoch:  6900 tx_loss 5.003668785095215 rx_loss 0.0030599073506891727
epoch:  7000 tx_loss 6.880077838897705 rx_loss 0.001666948664933443
epoch:  7100 tx_loss 4.616969108581543 rx_loss 0.006662178318947554
epoch:  7200 tx_loss 4.946462631225586 rx_loss 0.005256841890513897
epoch:  7300 tx_loss 4.189868450164795 rx_loss 0.001434624777175486
```

```
epoch:  7400 tx_loss 5.299427032470703 rx_loss 0.0010790545493364334
epoch:  7500 tx_loss 2.579153299331665 rx_loss 0.0010170130990445614
epoch:  7600 tx_loss 5.300052642822266 rx_loss 0.0006194217712618411
epoch:  7700 tx_loss 3.060616970062256 rx_loss 0.001046905410476029
epoch:  7800 tx_loss 3.76302433013916 rx_loss 0.0010550954611971974
epoch:  7900 tx_loss 4.558821201324463 rx_loss 0.000819277367554605
epoch:  8000 tx_loss 2.506747245788574 rx_loss 0.0001316565030720085
epoch:  8100 tx_loss 5.678400039672852 rx_loss 0.0002782098308671266
epoch:  8200 tx_loss 4.359503269195557 rx_loss 0.0002059138787444681
epoch:  8300 tx_loss 3.469759702682495 rx_loss 0.0005165160982869565
epoch:  8400 tx_loss 4.149544715881348 rx_loss 0.0011060985270887613
epoch:  8500 tx_loss 4.888036251068115 rx_loss 0.005483447108417749
epoch:  8600 tx_loss 5.130725860595703 rx_loss 0.00027724370011128485
epoch:  8700 tx_loss 3.6243844032287598 rx_loss 0.0018223000224679708
epoch:  8800 tx_loss 5.096489429473877 rx_loss 0.0002794914471451193
epoch:  8900 tx_loss 4.828254222869873 rx_loss 0.0002615988196339458
epoch:  9000 tx_loss 4.805609703063965 rx_loss 0.0010482058860361576
epoch:  9100 tx_loss 3.2759108543395996 rx_loss 0.0007440315675921738
epoch:  9200 tx_loss 7.272687911987305 rx_loss 0.0002482463023625314
epoch:  9300 tx_loss 3.771700143814087 rx_loss 0.0002673517738003284
epoch:  9400 tx_loss 4.501392364501953 rx_loss 0.00035931519232690334
epoch:  9500 tx_loss 6.863140106201172 rx_loss 0.00046710309106856585
epoch:  9600 tx_loss 5.743268966674805 rx_loss 0.0016094766324386
epoch:  9700 tx_loss 4.276912689208984 rx_loss 0.00021542655304074287
epoch:  9800 tx_loss 5.513436794281006 rx_loss 0.00012897647684440017
epoch:  9900 tx_loss 4.092998027801514 rx_loss 0.00040719134267419577
```

2.3 Plotting the Transmitter and Receiver Loss

[8]:
```
plt.figure(figsize = (12,5))
plt.subplot(1,2,1)
plt.plot(loss_tx)
plt.title('Transmitter Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.subplot(1,2,2)
plt.plot(loss_rx)
plt.title('Receiver Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.show()
```



2.4 Prediction

[11]:
```
#testing
batch_size = 100
raw_input = np.random.randint(0,msg_total,(batch_size))
print('Transmitted Signal:',raw_input)
label = np.zeros((batch_size, msg_total))
label[np.arange(batch_size), raw_input] = 1
tx_input = raw_input/float(msg_total)
xp = model_x.predict(tx_input)
y = xp + np.random.normal(0,0.001,(batch_size, channel,2))
pred = model_rx.predict(y)
pred_int = np.argmax(pred, axis = 1)
print('Received Signal:',pred_int)


print('accuracy:',accuracy_score(raw_input, pred_int))
```

```
Transmitted Signal: [3 1 0 2 5 3 7 3 6 4 3 6 3 2 2 7 1 0 4 4 2 6 7 1 5 7 0 2 0 6
6 2 2 0 6 4 1
 6 7 5 7 2 3 3 0 4 1 2 6 3 4 3 3 3 4 3 5 3 3 0 7 7 4 4 1 6 2 2 5 4 2 4 3 0
 0 1 4 0 2 4 6 6 6 5 6 5 6 0 5 2 7 5 1 4 3 3 6 1 5 2]
Received Signal: [3 1 0 2 5 3 7 3 6 4 3 6 3 2 2 7 1 0 4 4 2 6 7 1 5 7 0 2 0 6 6
2 2 0 6 4 1
 6 7 5 7 2 3 3 0 4 1 2 6 3 4 3 3 3 4 3 5 3 3 0 7 7 4 4 1 6 2 2 5 4 2 4 3 0
 0 1 4 0 2 4 6 6 6 5 6 5 6 0 5 2 7 5 1 4 3 3 6 1 5 2]
accuracy: 1.0
```