4_pam_model

May 12, 2019

0.0.1 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
```

0.0.2 Required Parameters (4-PAM)

[0]:
```
msg_total = 4
channel = 8
epochs = 5000
sigma = 1e-4
batch_size = 1024
```

0.0.3 Defining Required Functions

[0]:
```
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

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

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

0.1 Modelling the Transmitter

0.1.1 1. Tx encoder architecture

[0]:

```
tx_inp = Input((1,))
embbedings_layer = Dense(msg_total, activation = 'relu')(tx_inp)
layer_dense = Dense(2*channel, activation = 'relu')(embbedings_layer)
to_complex = Reshape((channel,2))(layer_dense)
x = Lambda(lambda x: keras.backend.l2_normalize(x))(to_complex)
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.
```

0.1.2 2. Definng Models

[0]:

```
model_policy = keras.models.Model(inputs=tx_inp, outputs=policy)
model_tx = keras.models.Model(inputs=tx_inp, outputs=xp)
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, 4)            8           input_1[0][0]
________________________________________________________________________________
__________________
dense_1 (Dense)                 (None, 16)           80          dense[0][0]
________________________________________________________________________________
__________________
reshape (Reshape)               (None, 8, 2)         0           dense_1[0][0]
________________________________________________________________________________
__________________
```

```
lambda_1 (Lambda)               (None, 8, 2)         0           reshape[0][0]
--------------------------------------------------------------------------------
------------------
lambda (Lambda)                 (None, 8, 2)         0           reshape[0][0]
--------------------------------------------------------------------------------
------------------
lambda_2 (Lambda)               ()                   0           lambda_1[0][0]
                                                                 lambda[0][0]
================================================================================
==================
Total params: 88
Trainable params: 88
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
None
```

0.2 Modelling the Receiver

0.2.1 Rx architecture

[0]:

```
rx_inp = Input((channel,2))
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, 8, 2)              0
_________________________________________________________________
reshape_1 (Reshape)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               2176
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 516
=================================================================
Total params: 2,692
Trainable params: 2,692
Non-trainable params: 0
_________________________________________________________________
None
```

0.2.2 Alternate Training

[0]:
```
loss_tx = []
loss_rx = []
for epoch in range(epochs):
#     Transmitter 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)
    xp = model_tx.predict(tx_input)
    y = xp + np.random.normal(0,0.001,(batch_size, channel,2))
    pred = model_rx.predict(y)
    loss = np.sum(np.square(label - pred), axis = 1)
    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])

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

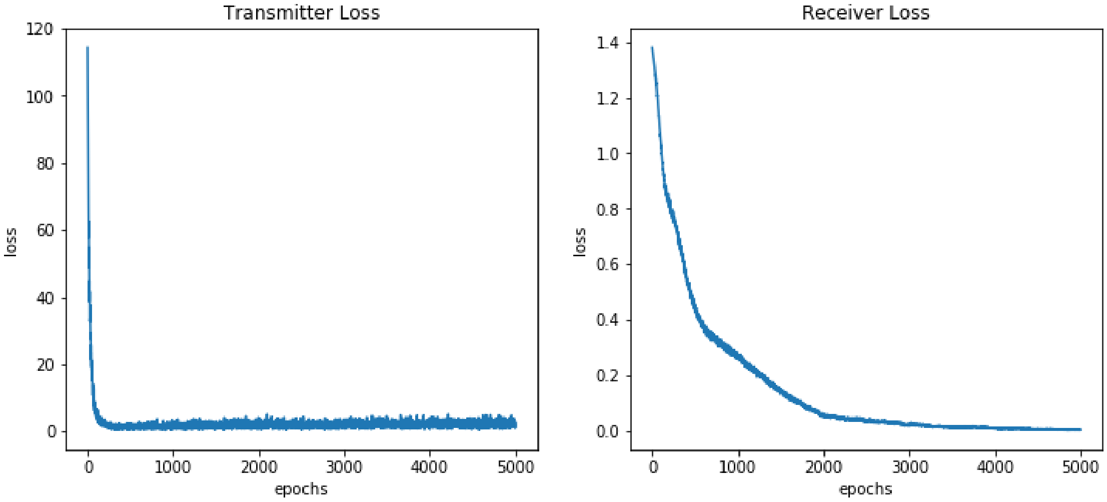
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
epoch:  0 tx_loss 114.35173034667969 rx_loss 1.3796947002410889
epoch:  100 tx_loss 6.021037578582764 rx_loss 1.0343232154846191
epoch:  200 tx_loss 3.665850877761841 rx_loss 0.8181131482124329
epoch:  300 tx_loss 1.520128846168518 rx_loss 0.7042916417121887
epoch:  400 tx_loss 1.4927887916564941 rx_loss 0.5554989576339722
epoch:  500 tx_loss 1.473479986190796 rx_loss 0.4429166615009308
epoch:  600 tx_loss 0.7329232692718506 rx_loss 0.375607967376709
epoch:  700 tx_loss 1.388350248336792 rx_loss 0.33823221921920776
epoch:  800 tx_loss 1.3279764652252197 rx_loss 0.3101707100868225
```

```
epoch:  900 tx_loss 1.0778858661651611 rx_loss 0.2860800325870514
epoch:  1000 tx_loss 1.438643455505371 rx_loss 0.26754045486450195
epoch:  1100 tx_loss 2.3404417037963867 rx_loss 0.23381781578063965
epoch:  1200 tx_loss 1.3312132358551025 rx_loss 0.21527284383773804
epoch:  1300 tx_loss 1.4248260259628296 rx_loss 0.19488610327243805
epoch:  1400 tx_loss 1.0382447242736816 rx_loss 0.16341015696525574
epoch:  1500 tx_loss 1.3557523488998413 rx_loss 0.1455385982990265
epoch:  1600 tx_loss 2.2829983234405518 rx_loss 0.12694334983825684
epoch:  1700 tx_loss 2.4766950607299805 rx_loss 0.10819017142057419
epoch:  1800 tx_loss 1.146506905555725 rx_loss 0.0801992416381836
epoch:  1900 tx_loss 2.664935827255249 rx_loss 0.07315555214881897
epoch:  2000 tx_loss 1.6458134651184082 rx_loss 0.05360748618841171
epoch:  2100 tx_loss 2.188070774078369 rx_loss 0.05371185764670372
epoch:  2200 tx_loss 1.9517011642456055 rx_loss 0.04446491599082947
epoch:  2300 tx_loss 4.250457763671875 rx_loss 0.04665788263082504
epoch:  2400 tx_loss 2.6695938110351562 rx_loss 0.04169715940952301
epoch:  2500 tx_loss 1.7643426656723022 rx_loss 0.035005323588848114
epoch:  2600 tx_loss 2.1374576091766357 rx_loss 0.03542027622461319
epoch:  2700 tx_loss 1.144116759300232 rx_loss 0.03213422745466232
epoch:  2800 tx_loss 3.3196122646331787 rx_loss 0.027622252702713013
epoch:  2900 tx_loss 3.393239736557007 rx_loss 0.02473902888596058
epoch:  3000 tx_loss 1.9258756637573242 rx_loss 0.01998080685734749
epoch:  3100 tx_loss 1.7442213296890259 rx_loss 0.022557873278856277
epoch:  3200 tx_loss 0.9708209037780762 rx_loss 0.02007543109357357
epoch:  3300 tx_loss 2.605186700820923 rx_loss 0.01592843048274517
epoch:  3400 tx_loss 3.0331175327301025 rx_loss 0.017260106280446053
epoch:  3500 tx_loss 2.038749933242798 rx_loss 0.013623886741697788
epoch:  3600 tx_loss 3.589238405227661 rx_loss 0.013174154795706272
epoch:  3700 tx_loss 3.232999801635742 rx_loss 0.014157405123114586
epoch:  3800 tx_loss 3.5114808082580566 rx_loss 0.012510138563811779
epoch:  3900 tx_loss 2.2777552604675293 rx_loss 0.01225421205163002
epoch:  4000 tx_loss 1.2391116619110107 rx_loss 0.00964068528264761
epoch:  4100 tx_loss 3.3996007442474365 rx_loss 0.00860733911395073
epoch:  4200 tx_loss 3.4578404426574707 rx_loss 0.007440897636115551
epoch:  4300 tx_loss 2.264455556869507 rx_loss 0.006812898442149162
epoch:  4400 tx_loss 1.8350210189819336 rx_loss 0.00730531383305788
epoch:  4500 tx_loss 1.1498150825500488 rx_loss 0.008417153730988503
epoch:  4600 tx_loss 3.6709954738616943 rx_loss 0.006787853315472603
epoch:  4700 tx_loss 1.7060564756393433 rx_loss 0.006307173985987902
epoch:  4800 tx_loss 1.1874375343322754 rx_loss 0.004096090793609619
epoch:  4900 tx_loss 3.6314749717712402 rx_loss 0.004732245579361916
```

0.2.3 Plotting Transmitter and Receiver Loss

[0]:

```
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()
```



0.2.4 Prediction

[0]:

```
#testing
batch_size = 100
raw_input = np.random.randint(0,msg_total,(batch_size))
print(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(pred_int)
```

```python
from sklearn.metrics import accuracy_score

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

```
[2 0 2 0 0 3 2 1 3 0 0 2 2 3 1 2 3 1 1 3 0 0 3 3 0 3 3 3 3 0 2 1 2 3 2 1 3
 2 0 0 0 2 2 2 0 0 3 3 0 2 3 2 1 2 3 1 0 3 1 0 1 2 0 2 3 1 3 3 0 2 2 1 1 1
 0 0 0 0 3 0 0 1 2 2 3 0 1 1 3 0 2 3 1 1 1 0 0 3 0 2]
[2 0 2 0 0 3 2 1 3 0 0 2 2 3 1 2 3 1 1 3 0 0 3 3 0 3 3 3 3 0 2 1 2 3 2 1 3
 2 0 0 0 2 2 2 0 0 3 3 0 2 3 2 1 2 3 1 0 3 1 0 1 2 0 2 3 1 3 3 0 2 2 1 1 1
 0 0 0 0 3 0 0 1 2 2 3 0 1 1 3 0 2 3 1 1 1 0 0 3 0 2]
accuracy: 1.0
```

[0]:

```
```