```
from graphviz import Digraph
import networkx as nx
from matplotlib import pyplot
import random
import pandas as pd
from __future__ import print_function
from keras.callbacks import LambdaCallback
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import plot_model
from keras import metrics
from keras.optimizers import RMSprop
from keras.utils.data_utils import get_file
import tensorflow as tf
import numpy as np
import sys
import io
from keras import models
from keras import layers

# TODO: chnage 256 to a smaller number - check accuracy plots + different epochs
# TODO: hidden state calculation should be faster - can be enabled based on the need
# or disabled if only properties of Y
```

```
######################################################################
#    Parameters
######################################################################

# input and output
sizeX = 3
sizeY = sizeX   # next character prediction


# text information
alphabet = ['c', 'a', 't', 'e', ' ']
sizeAlphabet = len(alphabet)

# human words generated by scramble game
human_words = ['cate',  'tace', 'ace',  'act',  'ate',  'cat', 'eat', 'eta',
               'tae', 'tea', 'ae', 'at', 'et', 'ta']
n_words_in_corpus = 100000


print("Alphabet used: {}".format(alphabet))
print("Human words count : {}".format(len(human_words)))
```

```
Alphabet used: ['c', 'a', 't', 'e', ' ']
Human words count : 14
```

```
######################################################################
#    Create corpus to train a model
######################################################################
text = ''
for i in range(0, n_words_in_corpus):
    if i % 10000 == 0:
      print("i = {}".format(i))

    j = int(random.uniform(0, len(human_words)))
    text = text + ' ' + human_words[j]

file = open('cat_' + str(n_words_in_corpus) + '.txt', 'w')
file.write(text)
file.close()
```

```
_ = pd.Series(text.split(' '))
print("Words distribution: {}".format(_.value_counts()))
```

```
i = 0
i = 10000
i = 20000
i = 30000
i = 40000
i = 50000
i = 60000
i = 70000
i = 80000
i = 90000
Words distribution: et      7292
at      7257
ace     7216
cat     7208
ta      7181
eat     7141
tea     7138
act     7120
eta     7119
tae     7092
tace    7092
cate    7086
ate     7063
ae      6995
           1
dtype: int64
```

```
####################################################################
#    Train model
####################################################################
def sentence_to_code(sentence, char_indices, maxlen, sizeAlphabet):
  x = np.zeros((1, maxlen, sizeAlphabet))
  for t, char in enumerate(sentence):
      x[0, t, char_indices[char]] = 1.0
  return x

def sample(preds, temperature=1.0):
    # helper function to sample an index from a probability array
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)


def on_epoch_end(epoch, _):
    # Function invoked at end of each epoch. Prints generated text.
    print()
    print('----- Generating text after Epoch: %d' % epoch)

    start_index = random.randint(0, len(text) - maxlen - 1)
    for diversity in [0.2, 0.5, 1.0, 1.2]:
        print('----- diversity:', diversity)

        generated = ''
        sentence = text[start_index: start_index + maxlen]
        generated += sentence
```

```
        print('----- Generating with seed: "' + sentence + '"')
        sys.stdout.write(generated)

        for i in range(400):
            x_pred = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(sentence):
                x_pred[0, t, char_indices[char]] = 1.

            preds = model.predict(x_pred, verbose=0)[0]
            next_index = sample(preds, diversity)
            next_char = indices_char[next_index]

            sentence = sentence[1:] + next_char

            sys.stdout.write(next_char)
            sys.stdout.flush()
        print()

chars = sorted(list(set(text)))
print('total chars:', len(chars))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))

# cut the text in semi-redundant sequences of maxlen characters
maxlen = 3
step = 3
sentences = []
next_chars = []
for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i: i + maxlen])
    next_chars.append(text[i + maxlen])
print('nb sequences:', len(sentences))

print('Vectorization...')
x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1

print("Chars to indices encoding: {}".format(char_indices))

# Sequential API
print('Build model...')
model = Sequential()
lstm_hunits = 256  # TODO: tune this parameter
model.add(LSTM(lstm_hunits, input_shape=(maxlen, len(chars))))
model.add(Dense(len(chars), activation='softmax'))

optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=[metrics.mae, metrics.ca
plot_model(model, to_file='multilayer_perceptron_graph.png')
```

```
total chars: 5
nb sequences: 128484
Vectorization...
WARNING: Logging before flag parsing goes to stderr.
W0721 06:14:31.047876 139731467863936 deprecation_wrapper.py:119] From /usr/local/lib/py

W0721 06:14:31.066533 139731467863936 deprecation_wrapper.py:119] From /usr/local/lib/py

W0721 06:14:31.069121 139731467863936 deprecation_wrapper.py:119] From /usr/local/lib/py

Chars to indices encoding: {' ': 0, 'a': 1, 'c': 2, 'e': 3, 't': 4}
Build model...
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 256)               268288
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 1285
=================================================================
Total params: 269,573
Trainable params: 269,573
Non-trainable params: 0
_________________________________________________________________
```

```
print("x size = {}".format(x.shape))
print("LSTM Layer Parameters: {}".format(4*(lstm_hunits*lstm_hunits + lstm_hunits*sizeAlphabet + lst
print("Dense Layer Parameters: {}".format(lstm_hunits*sizeAlphabet + sizeAlphabet))  # dense 5*256 +
print("y size = {}".format(x.shape))
```

```
x size = (128484, 3, 5)
LSTM Layer Parameters: 268288
Dense Layer Parameters: 1285
y size = (128484, 3, 5)
```

```
print_callback = LambdaCallback(on_epoch_end=on_epoch_end)

# history = funcmodel1.fit(x, [y,0.1*np.ones((128570, len(chars))), 0.1*np.ones((128570, len(chars))
history = model.fit(x, y,
          batch_size=128,
          epochs=10,
          callbacks=[print_callback])
```

```
W0721 06:14:31.698454 139731467863936 deprecation.py:323] From /usr/local/lib/python3.6/
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
W0721 06:14:32.259089 139731467863936 deprecation_wrapper.py:119] From /usr/local/lib/py

Epoch 1/10
128484/128484 [==============================] - 10s 77us/step - loss: 0.7240 - mean_abs

----- Generating text after Epoch: 0
----- diversity: 0.2
----- Generating with seed: "ce "
ce ace tace tace at tace ate ate act eta act ace act ate et at ace ate act at ate ace ac
----- diversity: 0.5
----- Generating with seed: "ce "
ce tace act et ace ace act eta ate ace eta ta act act at at cate at et at ace ta tae tac
----- diversity: 1.0
----- Generating with seed: "ce "
ce eta cate eta cate cate ace tae tea ta eta cat act ace ae cate cat act cat cate tae ae
----- diversity: 1.2
----- Generating with seed: "ce "
ce tae act at eat eta eat eat cate ace cate ace tae tace ae ta ae tace ace ta eta tae ea
Epoch 2/10
128484/128484 [==============================] - 8s 64us/step - loss: 0.6878 - mean_abso

----- Generating text after Epoch: 1
----- diversity: 0.2
----- Generating with seed: "ta "
ta tace tae act tace act act ate ace at ate ate ta act tace ate at ate ate act tace act
----- diversity: 0.5
----- Generating with seed: "ta "
ta tae eta ate ace eta cate cat ta eat tace act ta at eta et tace tace tace ace tae eta
----- diversity: 1.0
----- Generating with seed: "ta "
ta ate et ae tae cate at et cat act ate tea eta ta eat tace eta act ae tea act cate tae
----- diversity: 1.2
----- Generating with seed: "ta "
ta eat ae ate ace cate ace et tea eta tace at act ta et eta tea cat tae cat eat ace eat
Epoch 3/10
128484/128484 [==============================] - 8s 64us/step - loss: 0.6867 - mean_abso

----- Generating text after Epoch: 2
----- diversity: 0.2
----- Generating with seed: "tac"
tace ace tae eta tace act tae act at tae tae act eta ta ta at ace ace et ta tace tace at
----- diversity: 0.5
----- Generating with seed: "tac"
tace at cat tae at act at act act tea at ate eta tae et ate ta at at ate eat at ate tace
----- diversity: 1.0
----- Generating with seed: "tac"
tace cat cate tae ta tea cate act at et at cat cate at eta eat ate act ate cat ae eat ca
----- diversity: 1.2
----- Generating with seed: "tac"
tace tae cat tea eta ae ace at ace et eat cat ae eta ate cat ae tea at ace ta cat eta ta
Epoch 4/10
128484/128484 [==============================] - 8s 64us/step - loss: 0.6865 - mean_abso

----- Generating text after Epoch: 3
```

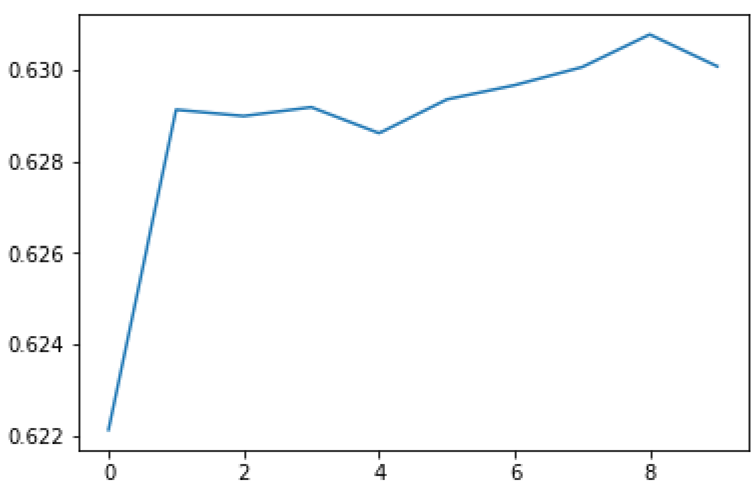
```
----- diversity: 0.2
----- Generating with seed: "a a"
a act ace act tae at act act act act ate tae at ace at at at ate tace at act ace ate cat
----- diversity: 0.5
----- Generating with seed: "a a"
a at cat tae at tace eat ate at at et et act act ace cate et ace eat eta cat cate tae ta
----- diversity: 1.0
----- Generating with seed: "a a"
a act et at tea ta cate eat tae ae ace cate tae ate et at tace ate cate ace cat ate act
----- diversity: 1.2
----- Generating with seed: "a a"
a act eta ace ae cat cate ae tace cate et cat tae et ae cat ta at cate cate eta eat ae a
Epoch 5/10
128484/128484 [==============================] - 8s 64us/step - loss: 0.6859 - mean_abso

----- Generating text after Epoch: 4
----- diversity: 0.2
----- Generating with seed: "a a"
a act at ace ta ace act eta ace at tae ace at ace ace ta ate ate ace tace ta cat ate ace
----- diversity: 0.5
----- Generating with seed: "a a"
a ate cat ace ta ace eta ta at cat tea ta et at at tace ate act act tae eat eta act tace
----- diversity: 1.0
----- Generating with seed: "a a"
a ace act at ta at act eat tea eat tace ta act act eat ace cate eta tea ace eta ace tace
----- diversity: 1.2
----- Generating with seed: "a a"
a at et cat ate tae act eat tace tae tae cate tea tae tea cat tace eat eat tae ae eat ac
Epoch 6/10
128484/128484 [==============================] - 8s 64us/step - loss: 0.6861 - mean_abso

----- Generating text after Epoch: 5
----- diversity: 0.2
----- Generating with seed: "eat"
eat act eta ate tace ate ace ace ate ta ate act tae ace act tace act ate ace act act tac
----- diversity: 0.5
----- Generating with seed: "eat"
eat eta ace tace ate et tae ate ae tace ta ate cat ace ace tace ate at ae at tea eta ate
----- diversity: 1.0
----- Generating with seed: "eat"
eat ace eta ate eta tace tea eta act ae act cate cat ta eta ace tae tea ae eta ta act ta
----- diversity: 1.2
----- Generating with seed: "eat"
eat act cat eat tace ae eat cat ate tace ae tea eat tae at ate ate ae eta tae tea et tac
Epoch 7/10
128484/128484 [==============================] - 8s 65us/step - loss: 0.6861 - mean_abso

----- Generating text after Epoch: 6
----- diversity: 0.2
----- Generating with seed: " ta"
 tae act at tace ate at at act ta act ta act at act at ate et act ate ate ta eta at act
----- diversity: 0.5
----- Generating with seed: " ta"
 tace tace tace at ta eat act et eta at at at ate eta ace tae act tace tace act ace et e
----- diversity: 1.0
----- Generating with seed: " ta"
 tae eta act act tea act ta tea ace cat et eta tae tea tae ace et cate ae ace ta eat ate
----- diversity: 1.2
```

```
----- Generating with seed: " ta"
 ta ta cat tace ta tace et ta cate tace cate et ace et ace cat cat tea ae eta tea at eat
Epoch 8/10
128484/128484 [==============================] - 8s 64us/step - loss: 0.6859 - mean_abso

----- Generating text after Epoch: 7
----- diversity: 0.2
----- Generating with seed: "act"
act ate act ate act ta act ace ace et ae ace ate at at at ate ate ate at tae act at act
----- diversity: 0.5
----- Generating with seed: "act"
act cat eat ate ace cate ate at eat at at eta et act ate act eta at cate ae tae ta eat e
----- diversity: 1.0
----- Generating with seed: "act"
act ate ace cate tea eat at act tace eat tace ate tace tea at tae ae ta cat eta cate act
----- diversity: 1.2
----- Generating with seed: "act"
act ae tace tea tea cat tea eat eat ace ate at eta at cate cate at tea cate cat cate cat
Epoch 9/10
128484/128484 [==============================] - 9s 66us/step - loss: 0.6861 - mean_abso

----- Generating text after Epoch: 8
----- diversity: 0.2
----- Generating with seed: " ca"
 cat act ace tace act at ta at at at ate ta at at ta cat act ta act at ate act ta ta at
----- diversity: 0.5
----- Generating with seed: " ca"
 cat ta tea act eta at tace ace cat tae tae tace ta cat eat tae tace at at tace cate ta
----- diversity: 1.0
----- Generating with seed: " ca"
 cat cate eta tea tae eta ta eat act ace eat eta et ace cat cat cate cat tea ta ate cat
----- diversity: 1.2
----- Generating with seed: " ca"
 cat tea ace ta ae at act ace cate eta act cat tae ace ate tae eta tae ae ae eta eta tea
Epoch 10/10
128484/128484 [==============================] - 8s 65us/step - loss: 0.6859 - mean_abso

----- Generating text after Epoch: 9
----- diversity: 0.2
----- Generating with seed: "at "
at ta act ta at at et act act tae tace tace at at ate ta at at at ate ta at ta tace at t
----- diversity: 0.5
----- Generating with seed: "at "
at eat eta ta ta ta tea at tace ace tea ace tace cat at eta eta act ta at tace at eta at
```

```python
for layer in model.layers:
  print(str(layer))
  if "LSTM" in str(layer):
    weightLSTM = layer.get_weights()
    print(weightLSTM)
```

```
<keras.layers.recurrent.LSTM object at 0x7f15760e52b0>
[array([[-1.3610653 , -1.3279054 , -0.6153458 , ..., -1.5251782 ,
         -1.3008299 , -1.1628857 ],
        [-0.12506263, -1.635507  , -1.1862597 , ...,  0.08018703,
         -0.6738107 , -0.43749925],
        [ 0.9982252 , -0.76560426, -0.95338005, ..., -0.314902  ,
         -0.2327487 , -1.141851  ],
        [-0.56351906, -1.9125947 , -0.5253068 , ..., -0.31953603,
         -0.3886431 ,  1.9483305 ],
        [-0.13845842,  0.4560747 , -0.4073523 , ..., -0.36154073,
         -0.2606028 ,  0.66599137]], dtype=float32), array([[-6.2252611e-01,  7.5429671e-
          9.4445329e-04, -5.3323281e-01,  1.5707639e+00],
        [ 2.3081830e-01,  7.1552676e-01,  7.1807891e-01, ...,
          1.5968713e-01,  3.4890661e-01, -1.0767295e+00],
        [ 3.3487618e-01, -3.1060573e-01, -1.5030769e-01, ...,
         -4.3454442e-02, -3.8534954e-01, -4.1522077e-01],
        ...,
        [-2.3434786e-01,  3.7828699e-01,  1.9743834e-01, ...,
         -2.0245211e-01,  2.4687666e-01,  1.5148067e+00],
        [-9.1636974e-01, -4.0686935e-01, -1.2397735e-01, ...,
```

```
# plot metrics
pyplot.plot(history.history['categorical_accuracy'])
pyplot.show()
```




```
# plot metrics
pyplot.plot(history.history['mean_absolute_error'])
pyplot.show()
```


```
##################################################################
# Validate that models predicts well
##################################################################
generated = ''
sentence = ' ca'
generated += sentence
print('----- Generating with seed: "' + sentence + '"')

x_pred = np.zeros((1, maxlen, len(chars)))
for t, char in enumerate(sentence):
    x_pred[0, t, char_indices[char]] = 1.

preds = model.predict(x_pred, verbose=0)[0]
print("preds =", preds)
next_index = sample(preds, 0.001)
next_char = indices_char[next_index]

sys.stdout.write("next_char = {} \n".format(next_char))
```

```
----- Generating with seed: " ca"
preds = [5.43377459e-16 1.09001814e-20 4.55302300e-14 3.57639726e-13
 1.00000000e+00]
next_char = t
```

```
##################################################################
#    Save model
##################################################################
model.save('model_train{}_alphabet{}_x{}.h5'.format(n_words_in_corpus, sizeAlphabet, sizeX))
```

```
##################################################################
#    Create StateTransition System from RNN
##################################################################

# I. Create Graph Structure (using only metadata of the network)
#--------------------------------------------------------------------------------

# G is the graph constructed from RNN Inputs
G = nx.balanced_tree(r=sizeAlphabet, h=sizeX)
f = Digraph('balanced_tree', filename='balanced_'+str(sizeX)+'_'+str(sizeAlphabet)+'.gv')
f.attr(rankdir='LR', size='8,5')

for b_node in G.nodes:
    f.node("S"+str(b_node), H="", Y="")

for i, b_edge in enumerate(G.edges):
    label = alphabet[i%sizeAlphabet]
    f.edge("S"+str(b_edge[0]), "S"+str(b_edge[1]), label)

# II. Create Attributes Empty and assign to the nodes
#--------------------------------------------------------------------------------
```

```
    labels = {}
    rnn_states = {}

    r = sizeAlphabet
    h = sizeX

    # root:
    state_placeholder = np.array([])
    rnn_states[0] = state_placeholder  # state will contain rnn H and Y of the model
    next_node_id = 1

    for level in range(1,h+1):
      print("Tree level:", level)
      elemnts_count = pow(r,level)
      print("elemnts_count = ", elemnts_count)

      print("Nodes btw :", next_node_id, " and ", next_node_id + elemnts_count)
      for node_id in range(next_node_id,  next_node_id + elemnts_count):
        rnn_states[node_id] = state_placeholder
        labels[node_id] = alphabet[node_id%sizeAlphabet]
      next_node_id = node_id+1
      print("next_node_id = ", next_node_id)

    nx.set_node_attributes(G, labels, 'label')
    nx.set_node_attributes(G, rnn_states, 'rnn_state')

    # III. Get x per node
    #-----------------------------------------------------------------------------

    x_val = {}
    above_labels = {}

    for node_id in G.nodes:
      node = G.nodes[node_id]
      print("node_id=",node_id, " node = ",node)
      shortest_path = nx.shortest_path(G, source=0, target=node_id)
      above_labels_str = ""
      for j in shortest_path[1:]:
        above_labels_str = above_labels_str + G.nodes[j]['label']
      print("above_labels_str = ", above_labels_str)


      if 'label' in node.keys():
        label = node['label']
        print("label = ", label)
        above_labels[node_id] = above_labels_str

        if len(above_labels_str) <= maxlen:
          x_val[node_id] = sentence_to_code(above_labels_str, char_indices, maxlen, sizeAlphabet)
        else:
          x_val[node_id] =np.array([])
    nx.set_node_attributes(G, above_labels, 'above_labels')
    nx.set_node_attributes(G, x_val, 'x_val')

    # IV. Get y per node
    #-----------------------------------------------------------------------------
    y_val = {}
    for node in G.nodes():
      if 'x_val' in  G.nodes[node].keys():
        x_val_node = G.nodes[node]['x_val']
        y_val_node = model.predict(x_val_node, verbose=0)[0]
        y_val[node] = y_val_node
    nx.set_node_attributes(G, y_val, 'y_val')

    # IV. Calculate hidden states per node
    #-----------------------------------------------------------------------------
    for layer in model.layers:
            if "LSTM" in str(layer):
                weightLSTM = layer.get_weights()
    warr,uarr, barr = weightLSTM
```

```
warr.shape,uarr.shape,barr.shape


def get_hidden_states_keras(model, xs, sizeX, sizeAlphabet, lstm_hunits):
  batch_size = 1
  len_ts = sizeX
  nfeature = sizeAlphabet
  inp = layers.Input(batch_shape= (batch_size, len_ts, nfeature),
                       name="input")
  rnn,s,c = layers.LSTM(lstm_hunits,
                         return_sequences=True,
                         stateful=False,
                         return_state=True,
                         name="RNN")(inp)
  states = models.Model(inputs=[inp],outputs=[s,c, rnn])
  for layer in states.layers:
      for layer1 in model.layers:
          if layer.name == layer1.name:
              layer.set_weights(layer1.get_weights())
  h_t_keras, c_t_keras, rnn = states.predict(xs.reshape(1,len_ts,5))
  return (h_t_keras, c_t_keras)


# # Example:

# xs = np.array([[[0., 0., 0., 0., 1.],
#         [1., 0., 0., 0., 0.],
#         [0., 0., 0., 0., 0.]]])
# tmp = get_hidden_states_keras(model, xs, sizeX, sizeAlphabet, lstm_hunits)
# print("tmp ={}".format(tmp))



h_val = {}
for node in G.nodes():
  if 'x_val' in  G.nodes[node].keys():
    x_val_node = G.nodes[node]['x_val']
    h_val_node = get_hidden_states_keras(model, x_val_node, sizeX, sizeAlphabet, lstm_hunits)
    h_val[node] = h_val_node

nx.set_node_attributes(G, h_val, 'h_val')



######################################################################
#    Ground truth property satisfaction
######################################################################
total_combinations = len(G.nodes()) # total nodes
n_satisfy = 0
# calculate Property rate over ground truth Graph
for node in G.nodes():
  if 'y_val' in  G.nodes[node].keys():
    y_val_node = G.nodes[node]['y_val']
    if any(y_val_node>0.8):
      n_satisfy = n_satisfy + 1
gt_satisfy = n_satisfy/total_combinations
print("Ground truth property satisfaction rate:{}".format(gt_satisfy))
```

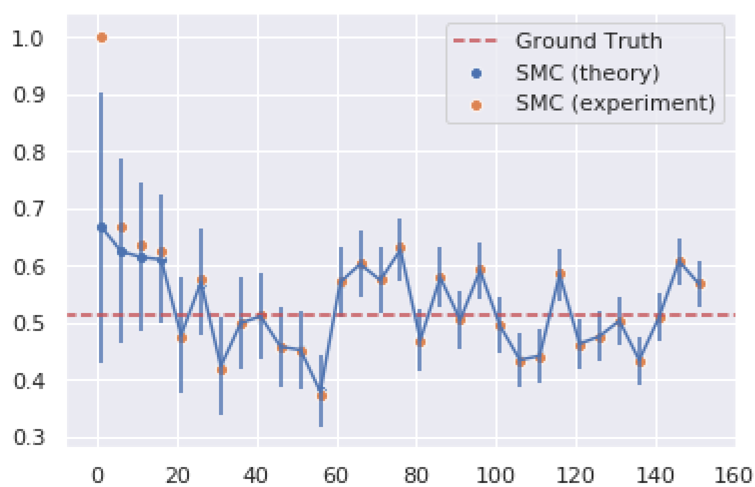
```
Ground truth property satisfaction rate:0.5128205128205128
```

```
####################################################
#  TensorSMC
####################################################
alpha = 1
beta = 1

smc_satisfy_rates = []
smc_ro_estimates = []
```

```
smc_nu_estimates = []
increasing_samples  = range(1, total_combinations, 5)

for j in increasing_samples:
  n_trajectories = 0   # number trajectories drawn so far
  n_satisfy = 0     # number of trajectories satisfying property so far

  for n_trajectories in range(0,j+1):
    index_rand = random.randrange(total_combinations)
    if 'y_val' in  G.nodes[index_rand].keys():
      y_val_node = G.nodes[index_rand]['y_val']
      if any(y_val_node>0.8):
        n_satisfy = n_satisfy + 1
  print("n_satisfy={}, n_trajectories={}".format(n_satisfy, n_trajectories))
  print("SMC property satisfaction rate:{}".format(n_satisfy/n_trajectories))
  ro = (n_satisfy + alpha)/(n_trajectories + alpha + beta)
  nu = np.sqrt(((alpha + n_satisfy)*(n_trajectories - n_satisfy + beta)) / (pow((alpha + n_trajector
  print("Estiamted property satisfaction:{} +/- {}".format(ro, nu))
  smc_satisfy_rates.append(n_satisfy/n_trajectories)
  smc_ro_estimates.append(ro)
  smc_nu_estimates.append(nu)
```

```
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
ax = sns.scatterplot(x=increasing_samples, y=smc_ro_estimates)
ax.errorbar(increasing_samples, smc_ro_estimates, yerr=smc_nu_estimates)
sns.scatterplot(x=increasing_samples, y=smc_satisfy_rates, ax = ax)
ax.axhline(gt_satisfy, ls='--', color='r')
ax.legend(labels=['Ground Truth', 'SMC (theory)', 'SMC (experiment)'])
```

```
<matplotlib.legend.Legend at 0x7f14d6f02080>
```




```
f
```