rnn_c

May 1, 2019

Reference: Tensorflow tutorials(https://www.tensorflow.org/tutorials/sequences/text_generation)

[1]:
```
import tensorflow as tf
tf.enable_eager_execution()

import numpy as np
import os
import time
import matplotlib.pyplot as plt
```

[2]:
```
# Read, then decode for py2 compat.
text = open('huge_c.txt', 'rb').read().decode(encoding='utf-8')
# length of text is the number of characters in it
print ('Length of text: {} characters'.format(len(text)))
```

```
Length of text: 26346 characters
```

[3]:
```
# The unique characters in the file
vocab = sorted(set(text))
print ('{} unique characters'.format(len(vocab)))
```

```
85 unique characters
```

[4]:
```
# Creating a mapping from unique characters to indices
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

text_as_int = np.array([char2idx[c] for c in text])
```

[5]:
```
# The maximum length sentence we want for a single input in characters
seq_length = 100
examples_per_epoch = len(text)//seq_length

# Create training examples / targets
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
sequences = char_dataset.batch(seq_length+1, drop_remainder = True)

for item in sequences.take(5):
```

```
    print(repr(''.join(idx2char[item.numpy()])))
```

```
'// SPDX-License-Identifier: GPL-2.0\n#include "audit.h"\n#include
<linux/fsnotify_backend.h>\n#include <'
'linux/namei.h>\n#include <linux/mount.h>\n#include <linux/kthread.h>\n#include
<linux/refcount.h>\n#inclu'
'de <linux/slab.h>\n\nstruct audit_tree;\nstruct audit_chunk;\n\nstruct
audit_tree {\n\trefcount_t count;\n\tin'
't goner;\n\tstruct audit_chunk *root;\n\tstruct list_head chunks;\n\tstruct
list_head rules;\n\tstruct list_h'
'ead list;\n\tstruct list_head same_root;\n\tstruct rcu_head head;\n\tchar
pathname[];\n};\n\nstruct audit_chun'
```

[6]:
```
def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[1:]
    return input_text, target_text

dataset = sequences.map(split_input_target)
```

[7]:
```
for input_example, target_example in  dataset.take(1):
    print ('Input data: ', repr(''.join(idx2char[input_example.numpy()])))
    print ('Target data:', repr(''.join(idx2char[target_example.numpy()])))
```

```
Input data:  '// SPDX-License-Identifier: GPL-2.0\n#include "audit.h"\n#include
<linux/fsnotify_backend.h>\n#include '
Target data: '/ SPDX-License-Identifier: GPL-2.0\n#include "audit.h"\n#include
<linux/fsnotify_backend.h>\n#include <'
```

[8]:
```
for i, (input_idx, target_idx) in enumerate(zip(input_example[:5],␣
 ↪target_example[:5])):
    print("Step {:4d}".format(i))
    print("  input: {} ({:s})".format(input_idx, repr(idx2char[input_idx])))
    print("  expected output: {} ({:s})".format(target_idx,␣
 ↪repr(idx2char[target_idx])))
```

```
Step    0
  input: 16 ('/')
  expected output: 16 ('/')
Step    1
  input: 16 ('/')
  expected output: 2 (' ')
Step    2
  input: 2 (' ')
  expected output: 44 ('S')
Step    3
  input: 44 ('S')
```

```
  expected output: 42 ('P')
Step    4
  input: 42 ('P')
  expected output: 31 ('D')
```

[9]:
```
# Batch size
BATCH_SIZE = 64
steps_per_epoch = examples_per_epoch//BATCH_SIZE

# Buffer size to shuffle the dataset
# (TF data is designed to work with possibly infinite sequences,
# so it doesn't attempt to shuffle the entire sequence in memory. Instead,
# it maintains a buffer in which it shuffles elements).
BUFFER_SIZE = 10000

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)

dataset
```

[9]:
```
<BatchDataset shapes: ((64, 100), (64, 100)), types: (tf.int64, tf.int64)>
```

[10]:
```
# Length of the vocabulary in chars
vocab_size = len(vocab)

# The embedding dimension
embedding_dim = 256

# Number of RNN units
rnn_units = 1024
```

[11]:
```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(len(vocab), embedding_dim,
                              batch_input_shape=[BATCH_SIZE, None]))
model.add(tf.keras.layers.CuDNNGRU(rnn_units,
        return_sequences=True,
        recurrent_initializer='glorot_uniform',
        stateful=True))
model.add(tf.keras.layers.Dense(len(vocab)))

print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (64, None, 256)           21760
_________________________________________________________________
cu_dnngru (CuDNNGRU)         (64, None, 1024)          3938304
_________________________________________________________________
dense (Dense)                (64, None, 85)            87125
```

```
=================================================================
Total params: 4,047,189
Trainable params: 4,047,189
Non-trainable params: 0
_________________________________________________________________
None
```

[12]:
```
for input_example_batch, target_example_batch in dataset.take(1):
    example_batch_predictions = model(input_example_batch)
    print(example_batch_predictions.shape, "# (batch_size, sequence_length,␣
 ↪vocab_size)")
```

```
(64, 100, 85) # (batch_size, sequence_length, vocab_size)
```

[13]:
```
def loss(labels, logits):
    return tf.nn.sparse_softmax_cross_entropy_with_logits(labels=labels,␣
 ↪logits=logits)

model.compile(
    optimizer = tf.train.AdamOptimizer(),
    loss = loss)
```

[14]:
```
# Directory where the checkpoints will be saved
checkpoint_dir = './training_checkpoints'
# Name of the checkpoint files
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_c_{epoch}")

checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True)
```

[15]:
```
EPOCHS=200

# history = model.fit(dataset.repeat(), epochs=EPOCHS,␣
 ↪steps_per_epoch=steps_per_epoch, callbacks=[checkpoint_callback])

history = model.fit(dataset.repeat(), epochs=EPOCHS,␣
 ↪steps_per_epoch=steps_per_epoch, callbacks=[checkpoint_callback])
```

```
Epoch 1/200
4/4 [==============================] - 0s 82ms/step - loss: 4.3378
Epoch 2/200
4/4 [==============================] - 0s 74ms/step - loss: 4.8581
Epoch 3/200
4/4 [==============================] - 0s 77ms/step - loss: 4.1511
Epoch 4/200
4/4 [==============================] - 0s 83ms/step - loss: 3.9667
Epoch 5/200
```

```
4/4 [==============================] - 0s 81ms/step - loss: 3.7485
Epoch 6/200
4/4 [==============================] - 0s 79ms/step - loss: 3.4750
Epoch 7/200
4/4 [==============================] - 0s 78ms/step - loss: 3.2431
Epoch 8/200
4/4 [==============================] - 0s 86ms/step - loss: 3.1185
Epoch 9/200
4/4 [==============================] - 0s 79ms/step - loss: 2.9910
Epoch 10/200
4/4 [==============================] - 0s 77ms/step - loss: 2.8767
Epoch 11/200
4/4 [==============================] - 0s 76ms/step - loss: 2.7564
Epoch 12/200
4/4 [==============================] - 0s 76ms/step - loss: 2.6457
Epoch 13/200
4/4 [==============================] - 0s 76ms/step - loss: 2.5471
Epoch 14/200
4/4 [==============================] - 0s 77ms/step - loss: 2.4573
Epoch 15/200
4/4 [==============================] - 0s 82ms/step - loss: 2.3721
Epoch 16/200
4/4 [==============================] - 0s 88ms/step - loss: 2.3039
Epoch 17/200
4/4 [==============================] - 0s 96ms/step - loss: 2.2458
Epoch 18/200
4/4 [==============================] - 0s 80ms/step - loss: 2.1756
Epoch 19/200
4/4 [==============================] - 0s 79ms/step - loss: 2.1192
Epoch 20/200
4/4 [==============================] - 0s 82ms/step - loss: 2.0716
Epoch 21/200
4/4 [==============================] - 0s 76ms/step - loss: 2.0133
Epoch 22/200
4/4 [==============================] - 0s 80ms/step - loss: 1.9690
Epoch 23/200
4/4 [==============================] - 0s 78ms/step - loss: 1.9184
Epoch 24/200
4/4 [==============================] - 0s 75ms/step - loss: 1.8763
Epoch 25/200
4/4 [==============================] - 0s 73ms/step - loss: 1.8223
Epoch 26/200
4/4 [==============================] - 0s 81ms/step - loss: 1.7843
Epoch 27/200
4/4 [==============================] - 0s 81ms/step - loss: 1.7385
Epoch 28/200
4/4 [==============================] - 0s 84ms/step - loss: 1.6874
Epoch 29/200
```

```
4/4 [==============================] - 0s 81ms/step - loss: 1.6541
Epoch 30/200
4/4 [==============================] - 0s 79ms/step - loss: 1.6110
Epoch 31/200
4/4 [==============================] - 0s 81ms/step - loss: 1.5687
Epoch 32/200
4/4 [==============================] - 0s 84ms/step - loss: 1.5273
Epoch 33/200
4/4 [==============================] - 0s 81ms/step - loss: 1.4916
Epoch 34/200
4/4 [==============================] - 0s 74ms/step - loss: 1.4454
Epoch 35/200
4/4 [==============================] - 0s 74ms/step - loss: 1.4110
Epoch 36/200
4/4 [==============================] - 0s 74ms/step - loss: 1.3637
Epoch 37/200
4/4 [==============================] - 0s 76ms/step - loss: 1.3343
Epoch 38/200
4/4 [==============================] - 0s 76ms/step - loss: 1.2990
Epoch 39/200
4/4 [==============================] - 0s 74ms/step - loss: 1.2538
Epoch 40/200
4/4 [==============================] - 0s 75ms/step - loss: 1.2152
Epoch 41/200
4/4 [==============================] - 0s 76ms/step - loss: 1.1831
Epoch 42/200
4/4 [==============================] - 1s 154ms/step - loss: 1.1496
Epoch 43/200
4/4 [==============================] - 0s 78ms/step - loss: 1.1099
Epoch 44/200
4/4 [==============================] - 0s 77ms/step - loss: 1.0789
Epoch 45/200
4/4 [==============================] - 0s 80ms/step - loss: 1.0504
Epoch 46/200
4/4 [==============================] - 0s 82ms/step - loss: 1.0235
Epoch 47/200
4/4 [==============================] - 0s 73ms/step - loss: 0.9998
Epoch 48/200
4/4 [==============================] - 0s 80ms/step - loss: 0.9671
Epoch 49/200
4/4 [==============================] - 0s 76ms/step - loss: 0.9359
Epoch 50/200
4/4 [==============================] - 0s 88ms/step - loss: 0.9012
Epoch 51/200
4/4 [==============================] - 0s 76ms/step - loss: 0.8713
Epoch 52/200
4/4 [==============================] - 0s 78ms/step - loss: 0.8433
Epoch 53/200
```

```
4/4 [==============================] - 0s 77ms/step - loss: 0.8219
Epoch 54/200
4/4 [==============================] - 0s 84ms/step - loss: 0.7875
Epoch 55/200
4/4 [==============================] - 0s 74ms/step - loss: 0.7647
Epoch 56/200
4/4 [==============================] - 0s 74ms/step - loss: 0.7443
Epoch 57/200
4/4 [==============================] - 0s 89ms/step - loss: 0.7218
Epoch 58/200
4/4 [==============================] - 0s 80ms/step - loss: 0.6971
Epoch 59/200
4/4 [==============================] - 0s 94ms/step - loss: 0.6648
Epoch 60/200
4/4 [==============================] - 0s 73ms/step - loss: 0.6509
Epoch 61/200
4/4 [==============================] - 0s 85ms/step - loss: 0.6324
Epoch 62/200
4/4 [==============================] - 0s 112ms/step - loss: 0.6025
Epoch 63/200
4/4 [==============================] - 0s 74ms/step - loss: 0.5859
Epoch 64/200
4/4 [==============================] - 0s 82ms/step - loss: 0.5635
Epoch 65/200
4/4 [==============================] - 0s 82ms/step - loss: 0.5442
Epoch 66/200
4/4 [==============================] - 0s 82ms/step - loss: 0.5253
Epoch 67/200
4/4 [==============================] - 0s 96ms/step - loss: 0.5089
Epoch 68/200
4/4 [==============================] - 0s 76ms/step - loss: 0.4913
Epoch 69/200
4/4 [==============================] - 0s 73ms/step - loss: 0.4726
Epoch 70/200
4/4 [==============================] - 0s 79ms/step - loss: 0.4592
Epoch 71/200
4/4 [==============================] - 0s 80ms/step - loss: 0.4384
Epoch 72/200
4/4 [==============================] - 0s 78ms/step - loss: 0.4293
Epoch 73/200
4/4 [==============================] - 0s 86ms/step - loss: 0.4130
Epoch 74/200
4/4 [==============================] - 0s 75ms/step - loss: 0.3982
Epoch 75/200
4/4 [==============================] - 0s 84ms/step - loss: 0.3871
Epoch 76/200
4/4 [==============================] - 0s 81ms/step - loss: 0.3687
Epoch 77/200
```

```
4/4 [==============================] - 0s 77ms/step - loss: 0.3594
Epoch 78/200
4/4 [==============================] - 0s 118ms/step - loss: 0.3493
Epoch 79/200
4/4 [==============================] - 0s 92ms/step - loss: 0.3354
Epoch 80/200
4/4 [==============================] - 0s 86ms/step - loss: 0.3205
Epoch 81/200
4/4 [==============================] - 0s 114ms/step - loss: 0.3112
Epoch 82/200
4/4 [==============================] - 0s 79ms/step - loss: 0.3031
Epoch 83/200
4/4 [==============================] - 0s 97ms/step - loss: 0.2934
Epoch 84/200
4/4 [==============================] - 0s 77ms/step - loss: 0.2834
Epoch 85/200
4/4 [==============================] - 0s 74ms/step - loss: 0.2800
Epoch 86/200
4/4 [==============================] - 0s 84ms/step - loss: 0.2692
Epoch 87/200
4/4 [==============================] - 0s 78ms/step - loss: 0.2660
Epoch 88/200
4/4 [==============================] - 0s 99ms/step - loss: 0.2587
Epoch 89/200
4/4 [==============================] - 0s 77ms/step - loss: 0.2529
Epoch 90/200
4/4 [==============================] - 0s 75ms/step - loss: 0.2494
Epoch 91/200
4/4 [==============================] - 0s 79ms/step - loss: 0.2412
Epoch 92/200
4/4 [==============================] - 0s 76ms/step - loss: 0.2349
Epoch 93/200
4/4 [==============================] - 0s 77ms/step - loss: 0.2277
Epoch 94/200
4/4 [==============================] - 0s 115ms/step - loss: 0.2214
Epoch 95/200
4/4 [==============================] - 0s 76ms/step - loss: 0.2210
Epoch 96/200
4/4 [==============================] - 0s 104ms/step - loss: 0.2151
Epoch 97/200
4/4 [==============================] - 0s 71ms/step - loss: 0.2070
Epoch 98/200
4/4 [==============================] - 0s 78ms/step - loss: 0.2010
Epoch 99/200
4/4 [==============================] - 0s 95ms/step - loss: 0.2003
Epoch 100/200
4/4 [==============================] - 0s 74ms/step - loss: 0.1922
Epoch 101/200
```

```
4/4 [==============================] - 0s 83ms/step - loss: 0.1923
Epoch 102/200
4/4 [==============================] - 0s 81ms/step - loss: 0.1872
Epoch 103/200
4/4 [==============================] - 0s 75ms/step - loss: 0.1887
Epoch 104/200
4/4 [==============================] - 0s 87ms/step - loss: 0.1793
Epoch 105/200
4/4 [==============================] - 0s 76ms/step - loss: 0.1827
Epoch 106/200
4/4 [==============================] - 0s 75ms/step - loss: 0.1805
Epoch 107/200
4/4 [==============================] - 0s 74ms/step - loss: 0.1704
Epoch 108/200
4/4 [==============================] - 0s 75ms/step - loss: 0.1700
Epoch 109/200
4/4 [==============================] - 0s 75ms/step - loss: 0.1684
Epoch 110/200
4/4 [==============================] - 1s 129ms/step - loss: 0.1676
Epoch 111/200
4/4 [==============================] - 0s 72ms/step - loss: 0.1638
Epoch 112/200
4/4 [==============================] - 0s 74ms/step - loss: 0.1627
Epoch 113/200
4/4 [==============================] - 0s 93ms/step - loss: 0.1583
Epoch 114/200
4/4 [==============================] - 0s 82ms/step - loss: 0.1559
Epoch 115/200
4/4 [==============================] - 0s 75ms/step - loss: 0.1561
Epoch 116/200
4/4 [==============================] - 0s 75ms/step - loss: 0.1520
Epoch 117/200
4/4 [==============================] - 0s 94ms/step - loss: 0.1448
Epoch 118/200
4/4 [==============================] - 0s 75ms/step - loss: 0.1478
Epoch 119/200
4/4 [==============================] - 0s 82ms/step - loss: 0.1476
Epoch 120/200
4/4 [==============================] - 0s 87ms/step - loss: 0.1414
Epoch 121/200
4/4 [==============================] - 0s 84ms/step - loss: 0.1400
Epoch 122/200
4/4 [==============================] - 0s 87ms/step - loss: 0.1429
Epoch 123/200
4/4 [==============================] - 0s 85ms/step - loss: 0.1394
Epoch 124/200
4/4 [==============================] - 0s 68ms/step - loss: 0.1408
Epoch 125/200
```

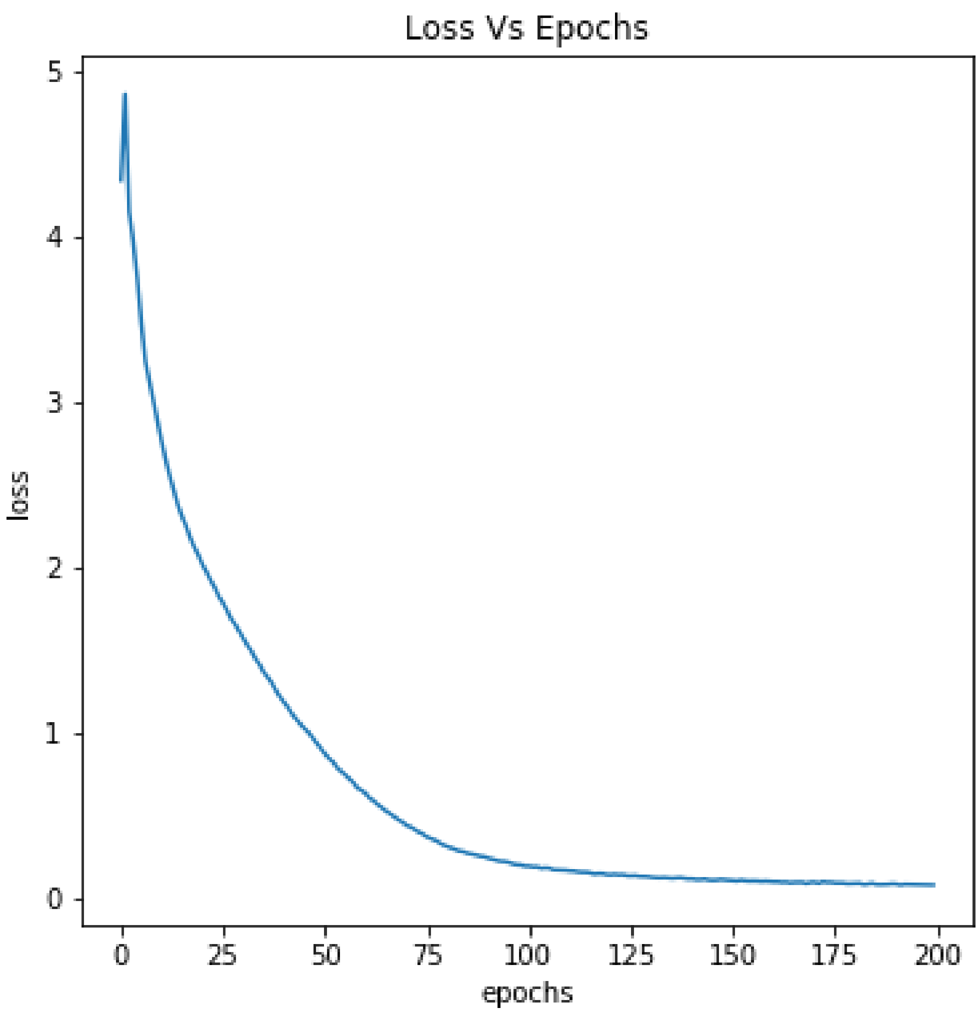
```
4/4 [==============================] - 0s 100ms/step - loss: 0.1347
Epoch 126/200
4/4 [==============================] - 0s 84ms/step - loss: 0.1337
Epoch 127/200
4/4 [==============================] - 0s 73ms/step - loss: 0.1356
Epoch 128/200
4/4 [==============================] - 1s 133ms/step - loss: 0.1310
Epoch 129/200
4/4 [==============================] - 0s 82ms/step - loss: 0.1308
Epoch 130/200
4/4 [==============================] - 0s 76ms/step - loss: 0.1294
Epoch 131/200
4/4 [==============================] - 0s 82ms/step - loss: 0.1249
Epoch 132/200
4/4 [==============================] - 0s 74ms/step - loss: 0.1231
Epoch 133/200
4/4 [==============================] - 0s 93ms/step - loss: 0.1212
Epoch 134/200
4/4 [==============================] - 0s 71ms/step - loss: 0.1234
Epoch 135/200
4/4 [==============================] - 0s 74ms/step - loss: 0.1214
Epoch 136/200
4/4 [==============================] - 1s 141ms/step - loss: 0.1188
Epoch 137/200
4/4 [==============================] - 0s 82ms/step - loss: 0.1201
Epoch 138/200
4/4 [==============================] - 0s 73ms/step - loss: 0.1228
Epoch 139/200
4/4 [==============================] - 0s 86ms/step - loss: 0.1193
Epoch 140/200
4/4 [==============================] - 0s 77ms/step - loss: 0.1152
Epoch 141/200
4/4 [==============================] - 0s 81ms/step - loss: 0.1159
Epoch 142/200
4/4 [==============================] - 0s 73ms/step - loss: 0.1115
Epoch 143/200
4/4 [==============================] - 0s 83ms/step - loss: 0.1116
Epoch 144/200
4/4 [==============================] - 0s 104ms/step - loss: 0.1145
Epoch 145/200
4/4 [==============================] - 0s 84ms/step - loss: 0.1091
Epoch 146/200
4/4 [==============================] - 0s 116ms/step - loss: 0.1077
Epoch 147/200
4/4 [==============================] - 0s 76ms/step - loss: 0.1086
Epoch 148/200
4/4 [==============================] - 0s 73ms/step - loss: 0.1121
Epoch 149/200
```

```
4/4 [==============================] - 0s 85ms/step - loss: 0.1100
Epoch 150/200
4/4 [==============================] - 0s 75ms/step - loss: 0.1073
Epoch 151/200
4/4 [==============================] - 0s 74ms/step - loss: 0.1073
Epoch 152/200
4/4 [==============================] - 0s 79ms/step - loss: 0.1032
Epoch 153/200
4/4 [==============================] - 0s 86ms/step - loss: 0.1084
Epoch 154/200
4/4 [==============================] - 0s 86ms/step - loss: 0.1042
Epoch 155/200
4/4 [==============================] - 0s 87ms/step - loss: 0.1023
Epoch 156/200
4/4 [==============================] - 0s 87ms/step - loss: 0.1033
Epoch 157/200
4/4 [==============================] - 0s 86ms/step - loss: 0.1025
Epoch 158/200
4/4 [==============================] - 0s 83ms/step - loss: 0.0998
Epoch 159/200
4/4 [==============================] - 0s 83ms/step - loss: 0.1020
Epoch 160/200
4/4 [==============================] - 0s 101ms/step - loss: 0.0996
Epoch 161/200
4/4 [==============================] - 0s 75ms/step - loss: 0.0977
Epoch 162/200
4/4 [==============================] - 0s 101ms/step - loss: 0.0980
Epoch 163/200
4/4 [==============================] - 0s 74ms/step - loss: 0.0944
Epoch 164/200
4/4 [==============================] - 0s 74ms/step - loss: 0.0959
Epoch 165/200
4/4 [==============================] - 0s 94ms/step - loss: 0.0912
Epoch 166/200
4/4 [==============================] - 0s 84ms/step - loss: 0.0938
Epoch 167/200
4/4 [==============================] - 0s 83ms/step - loss: 0.0942
Epoch 168/200
4/4 [==============================] - 0s 77ms/step - loss: 0.0944
Epoch 169/200
4/4 [==============================] - 1s 148ms/step - loss: 0.0870
Epoch 170/200
4/4 [==============================] - 0s 116ms/step - loss: 0.0945
Epoch 171/200
4/4 [==============================] - 0s 70ms/step - loss: 0.0950
Epoch 172/200
4/4 [==============================] - 0s 75ms/step - loss: 0.0887
Epoch 173/200
```

```
4/4 [==============================] - 0s 80ms/step - loss: 0.0985
Epoch 174/200
4/4 [==============================] - 0s 70ms/step - loss: 0.0939
Epoch 175/200
4/4 [==============================] - 0s 73ms/step - loss: 0.0947
Epoch 176/200
4/4 [==============================] - 0s 75ms/step - loss: 0.0907
Epoch 177/200
4/4 [==============================] - 0s 76ms/step - loss: 0.0899
Epoch 178/200
4/4 [==============================] - 0s 74ms/step - loss: 0.0904
Epoch 179/200
4/4 [==============================] - 0s 74ms/step - loss: 0.0863
Epoch 180/200
4/4 [==============================] - 0s 73ms/step - loss: 0.0871
Epoch 181/200
4/4 [==============================] - 0s 74ms/step - loss: 0.0892
Epoch 182/200
4/4 [==============================] - 1s 131ms/step - loss: 0.0893
Epoch 183/200
4/4 [==============================] - 0s 73ms/step - loss: 0.0816
Epoch 184/200
4/4 [==============================] - 0s 82ms/step - loss: 0.0840
Epoch 185/200
4/4 [==============================] - 0s 78ms/step - loss: 0.0895
Epoch 186/200
4/4 [==============================] - 0s 77ms/step - loss: 0.0825
Epoch 187/200
4/4 [==============================] - 0s 76ms/step - loss: 0.0812
Epoch 188/200
4/4 [==============================] - 0s 81ms/step - loss: 0.0820
Epoch 189/200
4/4 [==============================] - 0s 81ms/step - loss: 0.0829
Epoch 190/200
4/4 [==============================] - 0s 83ms/step - loss: 0.0865
Epoch 191/200
4/4 [==============================] - 0s 80ms/step - loss: 0.0846
Epoch 192/200
4/4 [==============================] - 0s 100ms/step - loss: 0.0784
Epoch 193/200
4/4 [==============================] - 0s 80ms/step - loss: 0.0834
Epoch 194/200
4/4 [==============================] - 0s 124ms/step - loss: 0.0825
Epoch 195/200
4/4 [==============================] - 0s 81ms/step - loss: 0.0820
Epoch 196/200
4/4 [==============================] - 0s 79ms/step - loss: 0.0820
Epoch 197/200
```

```
4/4 [==============================] - 0s 78ms/step - loss: 0.0789
Epoch 198/200
4/4 [==============================] - 0s 78ms/step - loss: 0.0789
Epoch 199/200
4/4 [==============================] - 0s 82ms/step - loss: 0.0798
Epoch 200/200
4/4 [==============================] - 0s 84ms/step - loss: 0.0787
```

[16]:
```
plt.figure(figsize = (6,6))
plt.plot(history.history['loss'])
plt.title('Loss Vs Epochs')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.show()
```



[20]:
```
checkpoint_dir = './training_checkpoints/ckpt_c_200'
# tf.train.latest_checkpoint(checkpoint_dir)

model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(len(vocab), embedding_dim,
                              batch_input_shape=[1, None]))
model.add(tf.keras.layers.CuDNNGRU(rnn_units,
        return_sequences=True,
        recurrent_initializer='glorot_uniform',
        stateful=True))
model.add(tf.keras.layers.Dense(len(vocab)))

model.load_weights(checkpoint_dir)

model.build(tf.TensorShape([1, None]))

print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (1, None, 256)            21760
_________________________________________________________________
cu_dnngru_2 (CuDNNGRU)       (1, None, 1024)           3938304
_________________________________________________________________
dense_2 (Dense)              (1, None, 85)             87125
=================================================================
Total params: 4,047,189
Trainable params: 4,047,189
Non-trainable params: 0
_________________________________________________________________
None
```

[21]:
```
def generate_text(model, start_string):
    # Evaluation step (generating text using the learned model)

    # Number of characters to generate
    num_generate = 1000

    # Converting our start string to numbers (vectorizing)
    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    # Empty string to store our results
    text_generated = []

    # Low temperatures results in more predictable text.
```

```
    # Higher temperatures results in more surprising text.
    # Experiment to find the best setting.
    temperature = 1.0

    # Here batch size == 1
    model.reset_states()
    for i in range(num_generate):
        predictions = model(input_eval)
        # remove the batch dimension
        predictions = tf.squeeze(predictions, 0)

        # using a multinomial distribution to predict the word returned by the␣
 ↪model
        predictions = predictions / temperature
        predicted_id = tf.multinomial(predictions, num_samples=1)[-1,0].numpy()

        # We pass the predicted word as the next input to the model
        # along with the previous hidden state
        input_eval = tf.expand_dims([predicted_id], 0)

        text_generated.append(idx2char[predicted_id])

    return (start_string + ''.join(text_generated))
```

[22]:
```
print(generate_text(model, start_string=u"struct"))
```

```
struct audit_tree *tree)
{
        put_tree *tree)
{
        return tree->pathname, 0, &path);
        if (IS_ERR(mark))
                return;
        out_mutex:
        mutex_unlock(&audit_tree_group->mark_mutex);
                return -ENOMEM;
        }

        mark = ale;
                fsnotify_free_mark(mark);
                fsnotify_put_mark(mark);
                kfree(chunk);
                return 0;
        }
        replace_mark_chunk(old->mark, new);
        /*
         /      struct audit_chunk *chunk)
{
```

```
        struct audit_chunk *old;

        assert_spin_locted->chunks, list) {
                        list_move(&owner->list, &prune_list);
                        need_prune = 1;
                } else {
                struct path path1, path2;
        struct vfsmount *tagged;
        int err;

        err = kern_path(new, 0, &path2);
                if (!err) {
                        good_one = canlic inline struct audit_tree_mark
*audit_mark(struct fsnotify_mark *mark,
                                   err)
                        goto skip_it;

                root_mnt = collect_mounts(&path);
                path_put(&path);
                if (IS_ERR(rootagner) /* reorder */
        for (p = tree->chunks.next; p != &tree->chunks; p = q) {
                struct node *node = list_entry(prune_list.next,
                                        struct audit_chunk *chunk =
from_chunk(mark) != chunk)
                goto o
```