RNN

December 4, 2021

1 RNN

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior.

1.0.1 IMDB sentiment classification task

This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. IMDB provided a set of 25,000 highly polar movie reviews for training, and 25,000 for testing. There is additional unlabeled data for use as well. Raw text and already processed bag of words formats are provided.

You can download the dataset from http://ai.stanford.edu/~amaas/data/sentiment/ or you can directly use " from keras.datasets import imdb " to import the dataset.

Few points to be noted:

Modules like SimpleRNN, LSTM, Activation layers, Dense layers, Dropout can be directly used from keras

For preprocessing, you can use required

[1]:
```
#load the imdb dataset
from keras.datasets import imdb
import keras
vocabulary_size = 5000

'''
Significance of the argument "num_words":
num_words: integer or None. Words are ranked by how often they occur (in the␣
 ↪training set) and
only the num_words most frequent words are kept.
Any less frequent word will appear as oov_char value in the sequence data.
If None, all words are kept. Defaults to None, so all words are kept.
```

```
'''
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words =␣
 ↪vocabulary_size)
print('Loaded dataset with {} training samples, {} test samples'.
 ↪format(len(X_train), len(X_test)))
```

```
Loaded dataset with 25000 training samples, 25000 test samples
```

Seeing sizes of the dataset and what features they have

[2]:
```
len(X_train)
```

[2]:
```
25000
```

[3]:
```
len(X_train[0])
```

[3]:
```
218
```

[4]:
```
X_train.shape
```

[4]:
```
(25000,)
```

[5]:
```
len(X_train[2])
```

[5]:
```
141
```

Comment this

[6]:
```
'''num=100
X_train=X_train[:num]
y_train=y_train[:num]
X_test=X_test[:num]
y_test=y_test[:num]'''
```

[6]:
```
'num=100\nX_train=X_train[:num]\ny_train=y_train[:num]\nX_test=X_test[:num]\ny_t
est=y_test[:num]'
```

[7]:
```
len(X_train[0])
```

[7]:
```
218
```

1.0.2 ==========

[8]:
```
#the review is stored as a sequence of integers.
# These are word IDs that have been pre-assigned to individual words, and the␣
 ↪label is an integer
idx_to_inspect=0
print('---review---')
print(X_train[idx_to_inspect])
print('---label---')
print(y_train[idx_to_inspect])

# to get the actual review
word2id = imdb.get_word_index()
# bug fixed using this https://github.com/udacity/AIND-NLP/pull/1/commits/
 ↪ef40e1716232e1c2f07a78cfdcf1983d82c358f1
id2word = {i+3: word for word, i in word2id.items()}
print('---review with words---')
print([id2word.get(i, ' ') for i in X_train[idx_to_inspect]])
print('---label---')
print(y_train[idx_to_inspect])
```

```
---review---
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36,
256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112,
167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16,
6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530,
38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8,
316, 8, 106, 5, 4, 2223, 2, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5,
25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407,
16, 82, 2, 8, 4, 107, 117, 2, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530,
476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32,
2071, 56, 26, 141, 6, 194, 2, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30,
2, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283,
5, 16, 4472, 113, 103, 32, 15, 16, 2, 19, 178, 32]
---label---
1
---review with words---
[' ', 'this', 'film', 'was', 'just', 'brilliant', 'casting', 'location',
'scenery', 'story', 'direction', "everyone's", 'really', 'suited', 'the',
'part', 'they', 'played', 'and', 'you', 'could', 'just', 'imagine', 'being',
'there', 'robert', ' ', 'is', 'an', 'amazing', 'actor', 'and', 'now', 'the',
'same', 'being', 'director', ' ', 'father', 'came', 'from', 'the', 'same',
'scottish', 'island', 'as', 'myself', 'so', 'i', 'loved', 'the', 'fact',
'there', 'was', 'a', 'real', 'connection', 'with', 'this', 'film', 'the',
'witty', 'remarks', 'throughout', 'the', 'film', 'were', 'great', 'it', 'was',
'just', 'brilliant', 'so', 'much', 'that', 'i', 'bought', 'the', 'film', 'as',
'soon', 'as', 'it', 'was', 'released', 'for', ' ', 'and', 'would', 'recommend',
```

```
'it', 'to', 'everyone', 'to', 'watch', 'and', 'the', 'fly', ' ', 'was',
'amazing', 'really', 'cried', 'at', 'the', 'end', 'it', 'was', 'so', 'sad',
'and', 'you', 'know', 'what', 'they', 'say', 'if', 'you', 'cry', 'at', 'a',
'film', 'it', 'must', 'have', 'been', 'good', 'and', 'this', 'definitely',
'was', 'also', ' ', 'to', 'the', 'two', 'little', ' ', 'that', 'played', 'the',
' ', 'of', 'norman', 'and', 'paul', 'they', 'were', 'just', 'brilliant',
'children', 'are', 'often', 'left', 'out', 'of', 'the', ' ', 'list', 'i',
'think', 'because', 'the', 'stars', 'that', 'play', 'them', 'all', 'grown',
'up', 'are', 'such', 'a', 'big', ' ', 'for', 'the', 'whole', 'film', 'but',
'these', 'children', 'are', 'amazing', 'and', 'should', 'be', ' ', 'for',
'what', 'they', 'have', 'done', "don't", 'you', 'think', 'the', 'whole',
'story', 'was', 'so', 'lovely', 'because', 'it', 'was', 'true', 'and', 'was',
"someone's", 'life', 'after', 'all', 'that', 'was', ' ', 'with', 'us', 'all']
---label---
1
```

1.0.3 Some minor analysis

[9]:
```
max([4,3,1],[1,2,2,2,22,2,2],key=len)
```

[9]:
```
[1, 2, 2, 2, 22, 2, 2]
```

[10]:
```
print('Maximum review length: {}'.format(
len(max((X_train + X_test), key=len))))
print('Minimum review length: {}'.format(
len(min((X_test + X_test), key=len))))
```

```
Maximum review length: 2697
Minimum review length: 14
```

1.1 Padding and other things

https://stats.stackexchange.com/a/211415

It's common in neural network training to pad all sequences to be as long as the longest one. This is done to simplify the code. However, the theory fully supports having sequences of different lengths. Just to make one thing clearer: the reason to pad the sequence is to make the code more efficient, both in the sense of memory and computing time.

[11]:
```
#pad sequences (write your code here)
from keras.preprocessing import sequence
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
```

[12]:
```
print("Sequences have now been padded")
```

```
Sequences have now been padded
```

1.2 ==========

[ ]:
```
```

[13]:
```
#design a RNN model (write your code)

from keras import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout, SimpleRNN
############################
embedding_size=32
############################
# From the definition of Keras documentation the Sequential model is a linear␣
 ↪stack of layers.You
# A Sequential model is appropriate for a plain stack of layers where each␣
 ↪layer has exactly one input tensor and one output tensor.
model_rnn=Sequential()
#########################
# Add an Embedding layer expecting input vocab of size 500, and
# output embedding dimension of size 32.
# revisit great explanation of embedding layer here: https://stackoverflow.com/
 ↪q/45649520
model_rnn.add(Embedding(vocabulary_size, embedding_size,␣
 ↪input_length=max_words))
model_rnn.add(SimpleRNN(16,input_shape = (max_words, embedding_size),␣
 ↪return_sequences=False,activation="relu"))
model_rnn.add(Dense(1, activation='sigmoid'))
print(model_rnn.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 32)           160000

 simple_rnn (SimpleRNN)      (None, 16)                784

 dense (Dense)               (None, 1)                 17

=================================================================
Total params: 160,801
Trainable params: 160,801
Non-trainable params: 0
```

```
_________________________________________________________________
None
```

1.3 Mention reason to choose that particular loss function and optimizer

1.3.1 Binary cross entropy

The metric I choose to optimize is BINARY CROSS ENTROPY.

FIrstly, the classification problem has 2 classes, so I am using binary cross entropy. This is better than mean squared loss AND "classification error = correct classifications/TOTAL SAMPLES" because it also helps us consider how good/bad our prediction was for a particular sample.

Eg: let's say sample A has actual label L1 and our model outputs for A belonging to L1 = 0.1 Eg: let's say sample B has actual label L1 and our model outputs for B belonging to L1 = 0.49 Now, we will probably classify both A and B as having label L1 (as $0.1 < 0.5$ and $0.49 < 0.5$). But, the misclassification is more severe for sample A than sample B.

To account for cases like this, the ln() function in cross-entropy takes into account the closeness of a prediction and is a more granular way to compute error.

Also, I read that cross entropy converges faster than MSE and has less local minima candidates than MSE.

1.3.2 Adam

I chose Adam because i read it has some properties like:- * Robust to the choice of hyper parameters. * Self-learns learning rate on a per-parameter basis. (as opposed to normal gradient descent) * I also read in a blog that it combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

[14]:
```
batch_size=32
```

[15]:
```
#train and evaluate your model
#choose your loss function and optimizer and mention the reason to choose that␣
 ↪particular loss function and optimizer
# use accuracy as the evaluation metric

model_rnn.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

[16]:
```
valid_size = int(len(y_train)*0.2)
num_epochs = 3
X_valid, y_valid = X_train[:valid_size], y_train[:valid_size]
X_train2, y_train2 = X_train[valid_size:], y_train[valid_size:]
model_rnn.fit(X_train2, y_train2, validation_data=(X_valid, y_valid),␣
 ↪batch_size=batch_size, epochs=num_epochs)
```

```
Epoch 1/3
625/625 [==============================] - 58s 91ms/step - loss: 1.1054 -
accuracy: 0.6014 - val_loss: 0.6327 - val_accuracy: 0.6550
Epoch 2/3
625/625 [==============================] - 55s 87ms/step - loss: 0.5950 -
accuracy: 0.6980 - val_loss: 0.6017 - val_accuracy: 0.6668
Epoch 3/3
625/625 [==============================] - 49s 78ms/step - loss: 8030.4634 -
accuracy: 0.7035 - val_loss: 0.6277 - val_accuracy: 0.6204
```

[16]:
```
<keras.callbacks.History at 0x7f9abef37650>
```

[17]:
```
#evaluate the model using model.evaluate()
scores = model_rnn.evaluate(X_test, y_test, verbose=1)
print('Test accuracy:', scores[1])
```

```
782/782 [==============================] - 11s 14ms/step - loss: 0.6279 -
accuracy: 0.6180
Test accuracy: 0.6180400252342224
```

Save the results also for comparison with LSTMs

[18]:
```
y_pred_rnn=model_rnn.predict(X_test)
```

[19]:
```
len(y_pred_rnn)
```

[19]:
```
25000
```

2 LSTM

Instead of using a RNN, now try using a LSTM model and compare both of them. Which of those performed better and why ?

[20]:
```
embedding_size=32
model_lstm=Sequential()
model_lstm.add(Embedding(vocabulary_size, embedding_size,␣
 ↪input_length=max_words))
model_lstm.add(LSTM(100))
model_lstm.add(Dense(1, activation='sigmoid'))
print(model_lstm.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, 500, 32)           160000
```

```
 lstm (LSTM)                 (None, 100)               53200

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
```

[21]:
```
#train and evaluate your model
#choose your loss function and optimizer and mention the reason to choose that␣
 ↪particular loss function and optimizer
# use accuracy as the evaluation metric

model_lstm.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

[22]:
```
valid_size = int(len(y_train)*0.2)
num_epochs = 3
X_valid, y_valid = X_train[:valid_size], y_train[:valid_size]
X_train2, y_train2 = X_train[valid_size:], y_train[valid_size:]
model_lstm.fit(X_train2, y_train2, validation_data=(X_valid, y_valid),␣
 ↪batch_size=batch_size, epochs=num_epochs)
```

```
Epoch 1/3
625/625 [==============================] - 145s 231ms/step - loss: 0.4568 -
accuracy: 0.7782 - val_loss: 0.3935 - val_accuracy: 0.8202
Epoch 2/3
625/625 [==============================] - 157s 251ms/step - loss: 0.3605 -
accuracy: 0.8448 - val_loss: 0.3917 - val_accuracy: 0.8274
Epoch 3/3
625/625 [==============================] - 150s 240ms/step - loss: 0.3449 -
accuracy: 0.8641 - val_loss: 0.4575 - val_accuracy: 0.8066
```

[22]:
```
<keras.callbacks.History at 0x7f9abe5eded0>
```

[23]:
```
#evaluate the model using model.evaluate()
scores = model_lstm.evaluate(X_test, y_test, verbose=1)
print('Test accuracy:', scores[1])
```

```
782/782 [==============================] - 61s 78ms/step - loss: 0.4499 -
accuracy: 0.8100
Test accuracy: 0.8099600076675415
```

[24]:
```
y_pred_lstm=model_lstm.predict(X_test)
```

[ ]:
```

```

2.1 Perform Error analysis and explain using few examples.

[25]:
```
import json
```

[26]:
```
dict_arr={"text":[],"actual_pred":list([int(x) for x in y_test]),
          "rnn_prob":list([float(x[0]) for x in y_pred_rnn]),
          "lstm_prob":list([float(x[0]) for x in y_pred_lstm])}
```

2.1.1 Preprocessing to retrieve text

[27]:
```
NUM_WORDS= vocabulary_size
INDEX_FROM=3

train,test = keras.datasets.imdb.load_data(num_words=NUM_WORDS,␣
 ↪index_from=INDEX_FROM)
train_x,train_y = train
test_x,test_y = test

word_to_id = keras.datasets.imdb.get_word_index()
word_to_id = {k:(v+INDEX_FROM) for k,v in word_to_id.items()}
word_to_id["<PAD>"] = 0
word_to_id["<START>"] = 1
word_to_id["<UNK>"] = 2
word_to_id["<UNUSED>"] = 3

id_to_word = {value:key for key,value in word_to_id.items()}
```

[28]:
```
def fetch_text(review_idx):
    ans=' '.join(id_to_word[id] for id in train_x[review_idx])
    return ans
```

[29]:
```
tot_test_len=len(y_test)
```

[30]:
```
for curr_id in range(tot_test_len):
    the_text=fetch_text(curr_id)
    dict_arr['text'].append(the_text)
```

[31]:
```
#dict_arr['text']
```

[32]:
```
with open("results_stored.json",'w') as fd:
    json.dump(dict_arr, fd, indent=4)
```

[33]:
```
import pandas as pd
df=pd.DataFrame(dict_arr)
```

[34]:
```
df.iloc[0]
```

[34]:
```
text           <START> this film was just brilliant casting l…
actual_pred                                                  0
rnn_prob                                              0.307954
lstm_prob                                             0.100013
Name: 0, dtype: object
```

[35]:
```
df['rnn_pred'] = df.apply(lambda row: 0 if row['rnn_prob'] < 0.5 else 1, axis=1)
df['lstm_pred'] = df.apply(lambda row: 0 if row['lstm_prob'] < 0.5 else 1,␣
↪axis=1)
```

[37]:
```
df
```

[37]:

	text	actual_pred	rnn_prob	lstm_prob	rnn_pred	lstm_pred
0	<START> this film was just brilliant casting l…	0	0.307954	0.100013	0	0
1	<START> big hair big <UNK> bad music and a gia…	1	0.845825	0.822048	1	1
2	<START> this has to be one of the worst films …	1	0.298401	0.186823	0	0
3	<START> the <UNK> <UNK> at storytelling the tr…	0	0.428570	0.448783	0	0
4	<START> worst mistake of my life br br i picke…	1	0.653494	0.879747	1	1
…	…	…	…	…	…	…
24995	<START> this is a racist movie but worthy of s…	1	0.409460	0.897186	0	1
24996	<START> bela lugosi plays a doctor who will do…	1	0.607366	0.832184	1	1
24997	<START> in a far away <UNK> is a planet called…	0	0.506651	0.301756	1	0
24998	<START> six <UNK> had me hooked i looked forwa…	0	0.462248	0.012618	0	0
24999	<START> as a big fan of the original film it's…	0	0.518565	0.881460	1	1

[25000 rows x 6 columns]

2.1.2 SOME ANALYSIS

[38]:
```
df['rnn_lstm'] = df.apply(lambda row: 0 if row['rnn_pred']!=row['lstm_pred']␣
↪else 1, axis=1)
df['rnn_actual'] = df.apply(lambda row: 0 if row['rnn_pred']!
↪=row['actual_pred'] else 1, axis=1)
df['lstm_actual'] = df.apply(lambda row: 0 if row['actual_pred']!
↪=row['lstm_pred'] else 1, axis=1)
```

Number of examples where both RNN and LSTM predicted wrong

[39]:
```
dummy_df=df[(df['rnn_actual']==0) & (df['lstm_actual']==0)]
```

[40]:
```
print(len(dummy_df))
```

```
2148
```

RNN WRONG, LSTM CORRECT

[41]:
```
dummy_df=df[(df['rnn_actual']==0) & (df['lstm_actual']==1)]
```

[42]:
```
print(len(dummy_df))
```

```
7401
```

LSTM WRONG, RNN CORRECT

[43]:
```
dummy_df=df[(df['rnn_actual']==1) & (df['lstm_actual']==0)]
```

[44]:
```
print(len(dummy_df))
```

```
2603
```

BOTH RNN, LSTM CORRECT

[45]:
```
dummy_df=df[(df['rnn_actual']==1) & (df['lstm_actual']==1)]
```

[46]:
```
print(len(dummy_df))
```

```
12848
```

2.1.3 From the above figure, we see there is a lot of examples (7401) to be exact where LSTM predicts correctly but RNN does not

ALso, LSTM has 80% accuracy whereas RNN has 61% accuracy only on the TEST SET. Clearly, LSTM performs much better than RNN. This is because addition of LSTMs helps normal RNNs to remember inputs over a long period of time. This is because LSTMs contain information in a

memory. Simple RNN fail to store information for a longer period of time. Sometimes data from long ago is required to predict the current output. But RNNs are absolutely incapable of handling such "long-term dependencies". LSTMs help establish finer control over which part of the context needs to be carried forward and what fraction of the past needs to be forgotten.

[48]:
```
df_check=df[(df['rnn_actual']==0) & (df['lstm_actual']==1)]
```

[53]:
```
df_check.iloc[9]
```

[53]:
```
text           <START> detective tony <UNK> frank sinatra ret…
actual_pred                                                  0
rnn_prob                                              0.615635
lstm_prob                                             0.030516
rnn_pred                                                     1
lstm_pred                                                    0
rnn_lstm                                                     0
rnn_actual                                                   0
lstm_actual                                                  1
Name: 35, dtype: object
```

2.1.4 In the below example, RNN predicts POSITIVE REVIEW WITH 60% probability whereas LSTM predicts CORRECT neg review with 97% probability.

Some keywords indicating a newgative review are:- * Sentence 1: sinatra tries hard to sell us the lame jokes * Sentence 2: project the frankly laughable

A "laughable" in sentence 2 captures that the director was not very good ans his efforts were laughable. Now, laughable could also mean that the movie was a comedy and we have a very laughable experience (nice comedy).

So, "laughable" can be used in both senses.

However, LSTM may have been able to link the "laughable" with "LAME JOKES" (negative feeling) and hence, predicts correctly,

But RNN would have forgotten about "lame" and interprets laughable in a positive way.

[52]:
```
df_check.iloc[9]['text']
```

[52]:
```
"<START> detective tony <UNK> frank sinatra returns to the screen after his self
titled debut this time it's a film that's played for <UNK> while on a <UNK> trip
<UNK> finds the body of a blonde beauty at the bottom of the sea her feet as you
might expect <UNK> in <UNK> <UNK> immediately on the case after being hired by
man mountain <UNK> <UNK> <UNK> finds himself immediately at risk as he has to
investigate some mafia types who turn the <UNK> on him and he is himself found
to be the main suspect he must now go on the run and hope to solve the case
alone the <UNK> sinatra tries hard to sell us the lame jokes and make us believe
he is a good detective oh and not to mention being sexually attractive to the
<UNK> <UNK> <UNK> but he fails miserably in this ham <UNK> <UNK> project the
```

```
frankly laughable <UNK> that <UNK> every female is quite <UNK> every woman in
the film is a <UNK> head who likes <UNK> over is front of the camera director
douglas of course <UNK> in <UNK> in on the <UNK> of their <UNK> each time as
they <UNK> their <UNK> <UNK> there's even a ridiculously campy gay character
that <UNK> belief this was a film made by real men for real men to <UNK> their
own <UNK> sexuality it's a <UNK> <UNK>"
```

2.2 ==========

2.2.1 An example where LSTM fails also

Here, "low grade" meant low budget. Also, the text does have negative terms but they are related to THE PLOT OF THE MOVIE AND NOT TO THE REVIEW of the movie. BOTH, LSTM AND RNN FAIL TO CAPTURE THIS.

[54]:
```
df_check=df[(df['rnn_actual']==0) & (df['lstm_actual']==0)]
```

[58]:
```
df_check.iloc[2]
```

[58]:
```
text           <START> this low grade universal <UNK> has jus…
actual_pred                                                  1
rnn_prob                                              0.417258
lstm_prob                                             0.393616
rnn_pred                                                     0
lstm_pred                                                    0
rnn_lstm                                                     1
rnn_actual                                                   0
lstm_actual                                                  0
Name: 52, dtype: object
```

[57]:
```
df_check.iloc[2]['text']
```

[57]:
```
"<START> this low grade universal <UNK> has just been <UNK> as an <UNK> dvd
release but intended as part of a collection of similar movies that i already
had in my <UNK> i decided to <UNK> it from other channels rather than wait for
that <UNK> release which is just as well since the end result was not anything
particularly special if <UNK> atmospheric at that for <UNK> the plot is pretty
weak \x96 even though in a way it <UNK> the vincent price vehicle theatre of
blood 1973 <UNK> without any of that film's campy <UNK> what we have here in
fact is a <UNK> <UNK> martin <UNK> \x96 whom we even see <UNK> his <UNK> <UNK>
of cheese with his pet cat \x96 who upon finding himself on the <UNK> end of art
critic alan <UNK> <UNK> <UNK> one time too many decides to end it all by <UNK>
himself into the nearby river however while <UNK> just that action he is <UNK>
by <UNK> <UNK> escaped killer dubbed the <UNK> and naturally enough saves the
poor <UNK> life with the intention of having the latter do all the dirty work
for him in <UNK> although it is supposedly set in the art <UNK> of new york all
```

```
we really see at work is <UNK> and commercial <UNK> robert <UNK> who keeps
painting the same <UNK> blonde girl joan <UNK> over and over in <UNK> <UNK> \x96
how is that for art who <UNK> enough is engaged to a rival art critic virginia
grey of <UNK> before long the latter is discovered with his <UNK> broken and
<UNK> is <UNK> but then <UNK> detective bill <UNK> gets the bright idea of
engaging another critic to <UNK> a <UNK> review of <UNK> work i did not know
that <UNK> <UNK> got <UNK> so as to <UNK> how violent his reaction is going to
be in the <UNK> <UNK> <UNK> himself into thinking that he is creating his
masterpiece by <UNK> <UNK> <UNK> <UNK> \x96 and <UNK> \x96 <UNK> which needless
to say <UNK> the attention of the constantly <UNK> grey we are led to believe
that she lacks material for her <UNK> <UNK> <UNK> to the <UNK> of both artist
and model <UNK> although the <UNK> is fully aware of how grey looks thanks to
her aforementioned haunting of <UNK> <UNK> <UNK> <UNK> he <UNK> off <UNK> \x96
who had by then become <UNK> girl \x96 in <UNK> apartment and <UNK> <UNK>
talking to you guessed it grey about his intention to <UNK> him as the fall guy
for the police sends the slow <UNK> giant off his deep <UNK> down to destroying
his own now <UNK> <UNK> image <UNK> enough although this was <UNK> <UNK> film
his name in the credits is <UNK> by the <UNK> <UNK>"
```

[ ]: