A Blog From Human-engineer-being

CODEBOOK, RESEARCH, RESEARCH NOTES, TIPS & TRICKS

# Duplicate Question Detection with Deep Learning on Quora Dataset

FEBRUARY 12, 2017 | EROGOL | 43 COMMENTS

Quora recently announced the first public dataset that they ever released. It includes 404351 question pairs with a label column indicating if they are duplicate or not. In this post, I like to investigate this dataset and at least propose a baseline method with deep learning.

Beside the proposed method, it includes some examples showing how to use Pandas, Gensim, Spacy and Keras. For the full code you check Github.

## Data Quirks

There are 255045 negative (non-duplicate) and 149306 positive (duplicate) instances. This induces a class imbalance however when you consider the nature of the problem, it seems reasonable to keep the same data bias with your ML model since negative instances are more expectable in a real-life scenario.

When we analyze the data, the shortest question is 1 character long (which is stupid and useless for the task) and the longest question is 1169 character (which is a long, complicated love affair question). I see that if any of the pairs is shorter than 10 characters, they do not make sense thus, I remove such pairs. The average length is 59 and std is 32.

There are two other columns "q1id" and "q2id" but I really do not know how they are useful since the same question used in different rows has different ids.

Some labels are not true, especially for the duplicate ones. In anyways, I decided to rely on the labels and defer pruning due to hard manual effort.

## Proposed Method

### Converting Questions into Vectors

Here, I plan to use Word2Vec to convert each question into a semantic vector then I stack a Siamese network to detect if the pair is duplicate.

Word2Vec is a general term used for similar algorithms that embed words into a vector space with 300 dimensions in general. These vectors capture semantics and even analogies between different words. The famous example is ;

```
king - man + woman = queen.
```

Word2Vec vectors can be used for may useful applications. You can compute semantic word similarity, classify documents or input these vectors to Recurrent Neural Networks for more advance applications.

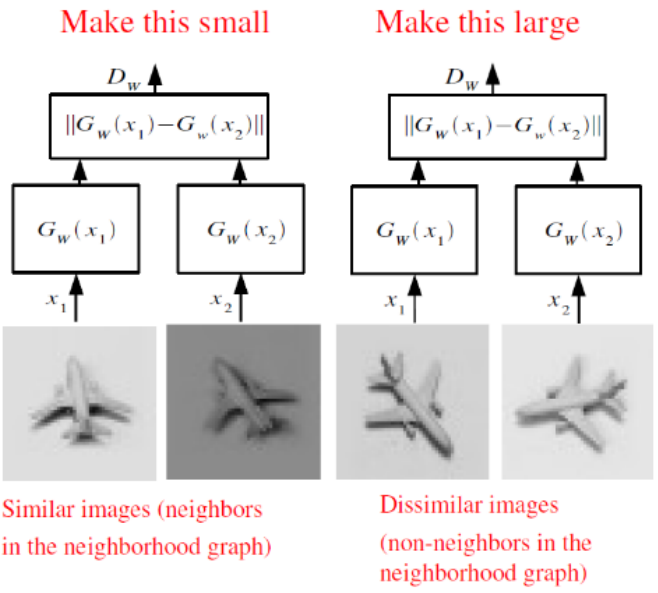
There are two well-known algorithms in this domain. One is Google's network architecture which learns representation by trying to predict surrounding words of a target word given certain window size. GLOVE is the another methos which relies on co-occurrence matrices. GLOVE is easy to train and it is flexible to add new words out-side of your vocabulary. You might like visit this tutorial to learn more and check this brilliant use-case Sense2Vec.

We still need a way to combine word vectors for singleton question representation. One simple alternative is taking the mean of all word vectors of each question. This is simple but really effective way for document classification and I expect it to work for this problem too. In addition, it is possible to enhance mean vector representation by using TF-IDF scores defined for each word. We apply weighted average of

word vectors by using these scores. It emphasizes importance of discriminating words and avoid useless, frequent words which are shared by many questions.

### Siamese Network

I described Siamese network in a previous post. In short, it is a two way network architecture which takes two inputs from the both side. It projects data into a space in which similar items are contracted and dissimilar ones are dispersed over the learned space. It is computationally efficient since networks are sharing parameters.




*Siamese network tries to contract instances belonging to the same classes and disperse instances from different classes in the feature space.*

## Implementation

Let's load the training data first.

```
# avoid decoding problems
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
df = pd.read_csv("/media/eightbit/8bit_5tb/NLP_data/Quora/DuplicateQuestion/quora_duplicate_questions.tsv",delimiter='\t')

# encode questions to unicode
df['question1'] = df['question1'].apply(lambda x: unicode(str(x),"utf-8"))
df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"utf-8"))
```

1_duplicate.py hosted with ♡ by GitHub view raw

For this particular problem, I train my own GLOVE model by using Gensim.

```
import gensim

questions = list(df['question1']) + list(df['question2'])

# tokenize
c = 0
for question in tqdm(questions):
    questions[c] = list(gensim.utils.tokenize(question, deacc=True, lower=True))
```

```
    c += 1

# train model
model = gensim.models.Word2Vec(questions, size=300, workers=16, iter=10, negative=20)

# trim memory
model.init_sims(replace=True)

# creta a dict
w2v = dict(zip(model.index2word, model.syn0))
print "Number of tokens in Word2Vec:", len(w2v.keys())

# save model
model.save('data/3_word2vec.mdl')
model.save_word2vec_format('data/3_word2vec.bin', binary=True)
```

2_duplicate.py hosted with ♡ by GitHub view raw

The above code trains a GLOVE model and saves it. It generates 300 dimensional vectors for words. Hyper parameters would be chosen better but it is just a baseline to see a initial performance. However, as I'll show this model gives performance below than my expectation. I believe, this is because our questions are short and does not induce a semantic structure that GLOVE is able to learn a salient model.

Due to the performance issue and the observation above, I decide to use a pre-trained GLOVE model which comes free with Spacy. It is trained on Wikipedia and therefore, it is stronger in terms of word semantics. This is how we use Spacy for this purpose.

```
  # exctract word2vec vectors
   import spacy
   nlp = spacy.load('en')

   vecs1 = [doc.vector for doc in nlp.pipe(df['question1'], n_threads=50)]
   vecs1 =  np.array(vecs1)
   df['q1_feats'] = list(vecs1)

   vecs2 = [doc.vector for doc in nlp.pipe(df['question2'], n_threads=50)]
   vecs2 =  np.array(vecs2)
   df['q2_feats'] = list(vecs2)

   # save features
   pd.to_pickle(df, 'data/1_df.pkl')
```

3_duplicate.py hosted with ♡ by GitHub view raw

Before going further, I really like Spacy. It is really fast and it does everything you need for NLP in a flash of time by hiding many intrinsic details. It deserves a good remuneration.  Similar to Gensim model, it also provides 300 dimensional embedding vectors.

The result I get from Spacy vectors is above Gensim model I trained. It is a better choice to go further with TF-IDF scoring.  For TF-IDF, I used scikit-learn (heaven of ML).  It provides TfIdfVectorizer which does everything you need.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(df['question1']) + list(df['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
tfidf.fit_transform(questions)

# dict key:word and value:tf-idf score
word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

4_duplicate.py hosted with ♡ by GitHub view raw

After we find TF-IDF scores, we convert each question to a weighted average of word2vec vectors by these scores. The below code does this for just "question1" column.

```
# exctract word2vec vectors
import spacy
nlp = spacy.load('en')

vecs1 = []
for qu in tqdm(list(df['question1'])):
    doc = nlp(qu)
    mean_vec = np.zeros([len(doc), 300])
    for word in doc:
        # word2vec
        vec = word.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec += vec * idf
    mean_vec = mean_vec.mean(axis=0)
    vecs1.append(mean_vec)
df['q1_feats'] = list(vecs1)
```

5_duplicate.py hosted with ♡ by GitHub view raw

Now, we are ready to create training data for Siamese network. Basically, I've just fetch the labels and covert mean word2vec vectors to numpy format. I split the data into train and test set too.

```
# shuffle df
df = df.reindex(np.random.permutation(df.index))

# set number of train and test instances
num_train = int(df.shape[0] * 0.88)
num_test = df.shape[0] - num_train
print("Number of training pairs: %i"%(num_train))
print("Number of testing pairs: %i"%(num_test))

# init data data arrays
X_train = np.zeros([num_train, 2, 300])
X_test  = np.zeros([num_test, 2, 300])
Y_train = np.zeros([num_train])
Y_test = np.zeros([num_test])

# format data
b = [a[None,:] for a in list(df['q1_feats'].values)]
q1_feats = np.concatenate(b, axis=0)

b = [a[None,:] for a in list(df['q2_feats'].values)]
q2_feats = np.concatenate(b, axis=0)

# fill data arrays with features
X_train[:,0,:] = q1_feats[:num_train]
X_train[:,1,:] = q2_feats[:num_train]
Y_train = df[:num_train]['is_duplicate'].values

X_test[:,0,:] = q1_feats[num_train:]
X_test[:,1,:] = q2_feats[num_train:]
Y_test = df[num_train:]['is_duplicate'].values
```

```
31
32 # remove useless variables
33 del b
34 del q1_feats
35 del q2_feats
```

6_duplicate.py hosted with ♡ by GitHub view raw

In this stage, we need to define Siamese network structure. I use Keras for its simplicity. Below, it is the whole script that I used for the definition of the model.

```
1  from __future__ import absolute_import
2  from __future__ import print_function
3  import numpy as np
4
5  from keras.models import Sequential, Model
6  from keras.layers import Dense, Dropout, Lambda, merge, BatchNormalization, Activation, Input, Merge
7  from keras import backend as K
8
9
10 def euclidean_distance(vects):
11     x, y = vects
12     return K.sqrt(K.sum(K.square(x - y), axis=1, keepdims=True))
13
14 def eucl_dist_output_shape(shapes):
15     shape1, shape2 = shapes
16     return (shape1[0], 1)
17
18 def cosine_distance(vests):
19     x, y = vests
20     x = K.l2_normalize(x, axis=-1)
21     y = K.l2_normalize(y, axis=-1)
22     return -K.mean(x * y, axis=-1, keepdims=True)
23
24 def cos_dist_output_shape(shapes):
25     shape1, shape2 = shapes
26     return (shape1[0],1)
27
28 def contrastive_loss(y_true, y_pred):
29     '''Contrastive loss from Hadsell-et-al.'06
30     http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf
31     '''
32     margin = 1
33     return K.mean(y_true * K.square(y_pred) + (1 - y_true) * K.square(K.maximum(margin - y_pred, 0)))
34
35
36 def create_base_network(input_dim):
37     '''
38     Base network for feature extraction.
39     '''
40     input = Input(shape=(input_dim, ))
41     dense1 = Dense(128)(input)
42     bn1 = BatchNormalization(mode=2)(dense1)
43     relu1 = Activation('relu')(bn1)
44
45     dense2 = Dense(128)(relu1)
46     bn2 = BatchNormalization(mode=2)(dense2)
47     res2 = merge([relu1, bn2], mode='sum')
48     relu2 = Activation('relu')(res2)
49
50     dense3 = Dense(128)(relu2)
```

```
    bn3 = BatchNormalization(mode=2)(dense3)
    res3 = Merge(mode='sum')([relu2, bn3])
    relu3 = Activation('relu')(res3)

    feats = merge([relu3, relu2, relu1], mode='concat')
    bn4 = BatchNormalization(mode=2)(feats)

    model = Model(input=input, output=bn4)

    return model


def compute_accuracy(predictions, labels):
    '''
    Compute classification accuracy with a fixed threshold on distances.
    '''
    return np.mean(np.equal(predictions.ravel() < 0.5, labels))

def create_network(input_dim):
    # network definition
    base_network = create_base_network(input_dim)

    input_a = Input(shape=(input_dim,))
    input_b = Input(shape=(input_dim,))

    # because we re-use the same instance `base_network`,
    # the weights of the network
    # will be shared across the two branches
    processed_a = base_network(input_a)
    processed_b = base_network(input_b)

    distance = Lambda(euclidean_distance, output_shape=eucl_dist_output_shape)([processed_a, processed_b])

    model = Model(input=[input_a, input_b], output=distance)
    return model

```

siamese_keras.py hosted with ♡ by GitHub view raw

I share here the best performing network with residual connections. It is a 3 layers network using Euclidean distance as the measure of instance similarity. It has Batch Normalization per layer. It is particularly important since BN layers enhance the performance considerably. I believe, they are able to normalize the final feature vectors and Euclidean distance performances better in this normalized space.

I tried Cosine distance which is more concordant to Word2Vec vectors theoretically but cannot handle to obtain better results. I also tried to normalize data into unit variance or L2 norm but nothing gives better results than the original feature values.



Let's train the network with the prepared data. I used the same model and hyper-parameters for all configurations. It is always possible to optimize these but hitherto I am able to give promising baseline results.

```
# create model
from siamese import *
from keras.optimizers import RMSprop, SGD, Adam
net = create_network(300)

```

```
# train
#optimizer = SGD(lr=1, momentum=0.8, nesterov=True, decay=0.004)
optimizer = Adam(lr=0.001)
net.compile(loss=contrastive_loss, optimizer=optimizer)

for epoch in range(50):
    net.fit([X_train_norm[:,0,:], X_train_norm[:,1,:]], Y_train,
          validation_data=([X_test_norm[:,0,:], X_test_norm[:,1,:]], Y_test),
          batch_size=128, nb_epoch=1, shuffle=True, )

    # compute final accuracy on training and test sets
    pred = net.predict([X_test_norm[:,0,:], X_test_norm[:,1,:]], batch_size=128)
    te_acc = compute_accuracy(pred, Y_test)

#    print('* Accuracy on training set: %0.2f%%' % (100 * tr_acc))
    print('* Accuracy on test set: %0.2f%%' % (100 * te_acc))
```

7_duplicate.py hosted with ♡ by GitHub view raw

## Results

In this section, I like to share test set accuracy values obtained by different model and feature extraction settings. We expect to see improvement over 0.63 since when we set all the labels as 0, it is the accuracy we get.

These are the best results I obtain with varying GLOVE models. they all use the same network and hyper-parameters after I find the best on the last configuration depicted below.

- Gensim (my model) + Siamese: 0.69
- Spacy + Siamese : 0.72
- Spacy + TD-IDF + Siamese : **0.79**

We can also investigate the effect of different model architectures. These are the values following the best word2vec model shown above.

- 2 layers net : 0.67
- 3 layers net + adam : 0.74
- 3 layers resnet (after relu BN) + adam : 0.77
- 3 layers resnet (before relu BN) + adam : 0.78
- 3 layers resnet (before relu BN) + adam + dropout : 0.75
- 3 layers resnet (before relu BN) + adam + layer concat : **0.79**
- 3 layers resnet (before relu BN) + adam + unit_norm + cosine_distance : Fail

Adam works quite well for this problem compared to SGD with learning rate scheduling. Batch Normalization also yields a good improvement. I tried to introduce Dropout between layers in different orders (before ReLU, after BN etc.), the best I obtain is 0.75. Concatenation of different layers improves the performance by 1 percent as the final gain.

In conclusion, here I tried to present a solution to this unique problem by composing different aspects of deep learning. We start with Word2Vec and combine it with TF-IDF and then use Siamese network to find duplicates. Results are not perfect and akin to different optimizations. However, it is just a small try to see the power of deep learning in this domain. I hope you find it useful :).

## Updates

- Switching last layer to FC layer improves performance to 0.84.
- By using bidirectional RNN and 1D convolutional layers together as feature extractors improves performance to 0.91. Maybe I'll explain details with another post.



### Related Posts:

1. **Stochastic Gradient formula for different learning algorithms**
2. **NegOut: Substitute for MaxOut units**
3. **Paper Review: Do Deep Convolutional Nets Really Need to be Deep (Or Even Convolutional)?**
4. **Paper review: CONVERGENT LEARNING: DO DIFFERENT NEURAL NETWORKS LEARN THE SAME REPRESENTATIONS?**

DEEP LEARNING · DUPLICATE DETECTION · MACHINE LEARNING · QUORA · SIAMESE NETWORK · WORD2VEC

The competition has finished and the hackers won !!. Anyway we still need your code updates to be posted please :)

Reply • Share ›

**erogol** Mod • 2 years ago

Anyone tried to replicate stuff here, please be aware of the wrong calculation of accuracy here in the example codes. I corrected it locally before releasing these values but forgot to correct it on Gist. Just now, I corrected it here as well !!! (Thanks **@Big Deeper** )

I do not plan to release the latest code until Kaggle ends and then still no promise, depends on the workload. But I should say hyper-parameter tuning is the silver-lining of this solution, at both feature extraction and model training steps. I suggest you to use probabilistic hyper-parameter optimization frameworks for this purpose not simple grid-search.

And please be cautioned that this post is only about investigating Quora problem with SiameseNet. I do not intend to get SATO or anything else, just fun. Happ hacking!!!

Reply • Share ›

**Gareth Dwyer** • 2 years ago

As others in the comments worked out, the accuracy function, copied from the Keras example, is flawed. It calculates precision, not accuracy, so results can be pretty strange (and you can 100% by predicting only one label)

It should be

def compute_accuracy(predictions, labels):
return np.mean(np.equal(predictions.ravel() < 0.5, labels))

See https://github.com/fchollet...

Reply • Share ›

**Big Deeper** ➔ Gareth Dwyer • 2 years ago

I guess any hope that the author would answer any of the questions, pointing out problems, would be futile. although he was pretty quick to accept praise.

1 Reply • Share ›

**erogol** Mod ➔ Gareth Dwyer • 2 years ago

You are right. I changed it locally but forgot to reflect it here. But the values are computed by the correct code.

Reply • Share ›

**AAKASH NAIN** • 2 years ago

Thanks for the great insight. Can you please share the BRNN approach too ?

Reply • Share ›

**erogol** Mod ➔ AAKASH NAIN • 2 years ago

I don't plan to share this until Kaggle competition ends. Sorry about that.

Reply • Share ›

**Jenkins yu** • 2 years ago

Is there any reason that you set idf=0 if not existed in word2tfidf?
try:
idf = word2tfidf[str(word)]
except:
#print word
idf = 0
I mean, isn't it more reasonable to set idf=1?
I am also confusing that why some wors are thrown away in word2tfidf, I checked part of them and found their length are > 1. Here are some samples of those words
"sheik
bsd
roose
adarsh
mather
elliot
20a"

Reply • Share ›

**erogol** Mod ➔ Jenkins yu • 2 years ago

You are right by the first point.
Some words are not in the word2vec list. I think this is the reason for these removals.

Reply • Share ›

**Big Deeper** • 2 years ago

I don't think this function "te_acc = compute_accuracy(pred, Y_test)" correctly computes accuracy. Let's say there are five (5) data points that result in "distance" < 0.5. Then labels[predictions.ravel() < 0.5] would extract a slice from labels that is 5 items long. If those 5 items happen to be all ones (1) then labels[X_validate.ravel() < 0.5].mean() should result in one (1) or 100% accuracy, which is clearly wrong as there are hundreds of thousands of other points with label = 1 still not considered, there may be negative question pairs which are also incorrectly classified.

Do I misunderstand something about what labels[predictions.ravel() < 0.5].mean() computes?

Reply • Share ›

**erogol** Mod ➚ Big Deeper • 2 years ago

You are right. I used locally below code but forgot to change it here,

```
def compute_accuracy(predictions, labels):
return np.mean(np.equal(predictions.ravel() < 0.5, labels))
```

Reply • Share ›

**Manuel C** • 2 years ago

Great job. Two questions, 1) Would it be better to use CountVectorizer with binary=True instead of tf-idf, since the questions are short? 2) Since you use batch normalization, maybe you can use a sigmoid activation function instead of ReLU without gradient vanishing?

Reply • Share ›

**Big Deeper** • 2 years ago

I made an attempt to recreate the results for "3 layers resnet (before relu BN) + adam + layer concat" which is the best configuration proposed here. I have run it for 100 epochs/iterations and 2 passes per loop (200) and I did not see anything much barely above 74%, mostly some high 73%+ as computed by "te_acc = compute_accuracy(pred, Y_test)" I was testing it on Titan X. The score jumped up and down a lot, but it was mostly staying below 74% after touching it.

Can you shed some light on the specific conditions for the 79% measurement?

Reply • Share ›

**Ryan De Vera** • 2 years ago

Thanks for this post! It is very insightful. Would you be willing to share some of the details around your second update? Does your base network consist of a single 1D convolutional layer that feeds into a BiLSTM? How many filters does your convolutional layer use?

Reply • Share ›

**erogol** Mod ➚ Ryan De Vera • 2 years ago

unfortunately, I don't plan to share these details, at least to the end of Kaggle.

Reply • Share ›

**Big Deeper** ➚ Ryan De Vera • 2 years ago

I am having a problem recreating numbers from the code that is already here. Maybe you can try getting the code in this blog to run and get the 79% number to help either confirm it, or see if there is an issue. Replacing one layer with a Keras Dense layer and playing with a limited number of Conv1D and RNNs combos should not be too difficult.

Reply • Share ›

**Canoe** ➚ Ryan De Vera • 2 years ago

same questions here

Reply • Share ›

**Ryan de Vera** ➚ Canoe • 2 years ago

Hey Canoe! Wondering if you had any luck with this? I was able to run the Conv1D and BiLSTM but I was unable to reproduce the 0.91 accuracy. I tried using 128, 256 & 512 filters in the convolutional layer.

Reply • Share ›

**Big Deeper** ➚ Ryan de Vera • 2 years ago

I got 100% validation accuracy with a few network changes and a specific initialization, but I know it is absolutely wrong, because the accuracy function does not compute what it is supposed to compute.

Reply • Share ›

**Big Deeper** ➚ Ryan de Vera • 2 years ago
Are you able to reproduce even 0.79 accuracy? The accuracy function itself appears to be flawed.
▲ | ▼ • Reply • Share ›

**Fouad** • 2 years ago
Thank you for your amazing post , I am wondering how did you submit your results to the competition and what was your best score.
Siamese outcomes Euclidean distance , how did you transform the distance to PDF to calculate the LogLoss. are you using (1/1+d) or the expo transformation ?
▲ | ▼ • Reply • Share ›

**erogol** Mod ➚ Fouad • 2 years ago
I ve never submitted to challenge. But you might have your predictions by thresholding siamese distance. You might change siamese layer with an MLP and get better accuracy
▲ | ▼ • Reply • Share ›

**Sri** • 2 years ago
Great post! Love the explanation and the step by step code to follow along. One question - when you refer to resnet are you referring to the architecture proposed here: https://arxiv.org/pdf/1512.... ? If so how do you build that using Keras. The network in the `create_base_network` seems to be a 3 layer feed forward network with batch normalization.
▲ | ▼ • Reply • Share ›

**erogol** Mod ➚ Sri • 2 years ago
you can refer to github repo related with this blog https://github.com/erogol/Q...
▲ | ▼ • Reply • Share ›

**Big Deeper** ➚ erogol • 2 years ago
Which file in the github repo is the one with the highest performance? 1_, 2_, or 3_? Thanks
▲ | ▼ • Reply • Share ›

**Иван Косолапов** • 2 years ago
What versions of libraries do you use? I can not start your code (
▲ | ▼ • Reply • Share ›

**erogol** Mod ➚ Иван Косолапов • 2 years ago
it is really hard to check this since it was a temporary virtual env.
▲ | ▼ • Reply • Share ›

**Jeroen van Dalen** • 2 years ago
What is the baseline of just using the distance metric straight after vectorisation, with a (learned) cut off point? Thus: Input, Vector (glove or you other methods), distance, cutoff? Would expect it do better then chance?
▲ | ▼ • Reply • Share ›

**erogol** Mod ➚ Jeroen van Dalen • 2 years ago
Sorry for very late response. It is better than random but not crucially. I tried it before doing anything else but don't remember the exact values.
▲ | ▼ • Reply • Share ›

**Dan Ofer** • 2 years ago
What was your AUC?
(I have 85 without any W2V)
▲ | ▼ • Reply • Share ›

**erogol** Mod ➚ Dan Ofer • 2 years ago
Don't know yet. Let me return once I measure AUC. If you have accuracy we can also compare it too.
▲ | ▼ • Reply • Share ›

**Dan Ofer** ➚ erogol • 2 years ago
Sweet.
(Big fan of your blog btw)
1 ▲ | ▼ • Reply • Share ›

**erogol** Mod ➚ Dan Ofer • 2 years ago
thx :)
▲ | ▼ • Reply • Share ›