Elior Cohen
Data scientist, Pythonista
Jun 8 · 9 min read

Implementing MaLSTM on Kaggle's Quora Question Pairs competition

This article is about the MaLSTM Siamese LSTM network (link to article on the second paragraph) for sentence similarity and its appliance to Kaggle's Quora Pairs competition.
I will do my best to explain the network and go through the Keras code (if you are only here for the code, scroll down :)
Full code on github

. . .

In the past few years, deep learning is all the fuss in the tech industry. To keep up on things I like to get my hands dirty implementing interesting network architectures I come across in article readings.

Few months ago I came across a very nice article called Siamese Recurrent Architectures for Learning Sentence Similarity which offers a pretty straightforward approach at the common problem of sentence similarity.
Named **MaLSTM** ("Ma" for Manhattan distance), its architecture is depicted in *figure 1* (diagram excludes the sentence preprocessing part).
Notice that since this is a Siamese network, it is easier to train because it shares weights on both sides.

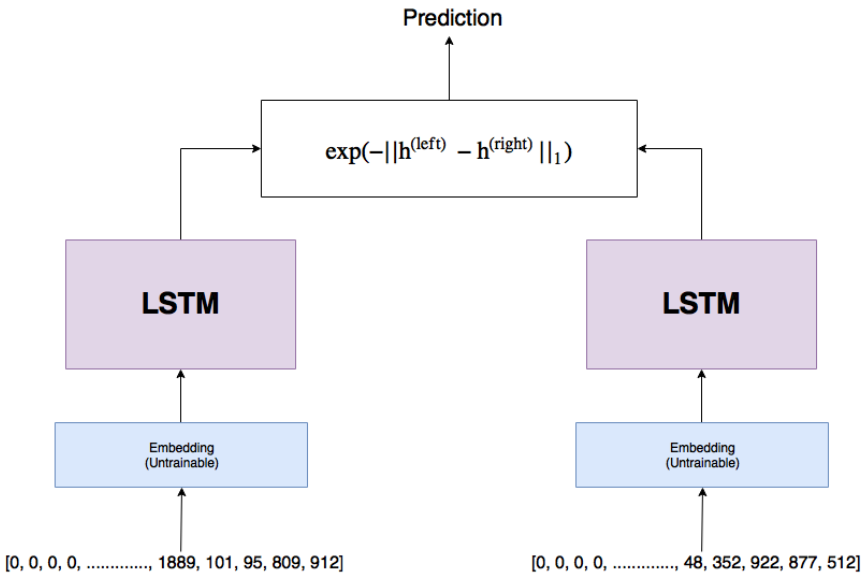


Figure 1 MaLSTM's architecture - **Similar color means the weights are shared between the same-colored elements**

Network explained

(I will be using Keras, so some technical details are related to the implementation)

So first of all, what is a "Siamese network"?
Siamese networks are networks that have two or more identical sub-networks in them.
Siamese network seem to perform good on similarity tasks and have been used for tasks like sentence semantic similarity, recognizing forged signatures and many more.

In MaLSTM the identical sub-network is all the way from the embedding up to the last LSTM hidden state.
Word embedding is a modern way to represent words in deep learning models, more about it can be found in this nice blog post.
Essentially it's a method to give words semantic meaning in a vector representation.

Inputs to the network are zero padded sequences of word indices, these inputs are vectors of fixed length, where the first zeros are being ignored and the non zeros are indices that uniquely identify words.
Those vectors are then fed into the the embedding layer, which looks ups the corresponding embedding for each word and encapsulates all of them into a matrix which represents the given text as a series of embeddings.
The embedding used are Google's word2vec as in the original paper.
The process is depicted in figure 2.

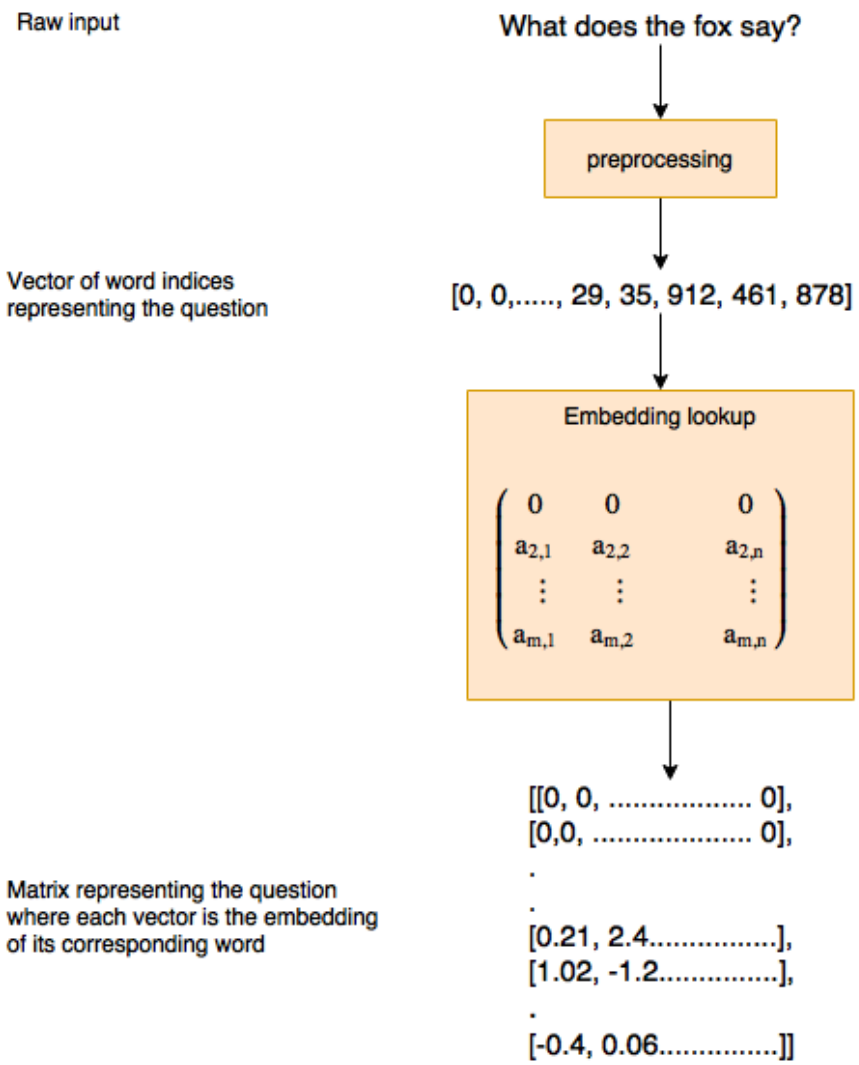


Figure 2 Embedding process

Then the two embedded matrices representing the candidate similar questions are fed into the LSTM (practically, there is only one LSTM) and the final state of the LSTM for each question is a 50 dimensional vector which is trained to capture the semantic meaning of the question.
In figure 1, this vector is denoted by the letter *h*.
If you don't entirely understand LSTMs, I suggest reading this wonderful post.

By now we have the two vectors that hold the semantic meaning of each question, now we put them through the defined similarity function (below)

$$\exp\left(-||h^{(left)} - h^{(right)}||_1\right)$$

MaLSTM similarity function

and since we have an exponent of a negative the output (the prediction in our case) will be between 0 and 1.

Training

The optimizer of choice in the article is the Adadelta optimizer, which can be read about in this article, using gradient clipping to avoid the exploding gradient problem.
Nice explanation taken from the Udacity deep learning course about gradient clipping in this video.

This is where I will diverge a little from the original paper for the sake of simplicity—in the original paper the LSTM weights are initialized in a specific manner and then (pre)trained on some other task.

Other parameters such as batch size, epochs and the gradient clipping norm value are chosen by me.

. . .

CODE

My full implementation can be found in this Jupyter notebook—keep following this post to see only the significant parts.

Here (and in the notebook) I've excluded all of the data analysis part, again to keep things simple and the article readable.

Preprocessing

We get the data as raw text, so our first mission is to take the text and convert it into lists of word indices.
When first opening the training data files in pandas, this is what you get.

id	qi...	qid2	question1	question2	is_duplicate
0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0
1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0
2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0
3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when [math]23^{24}[/math] is divided by 24,23?	0
4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0
5	11	12	Astrology: I am a Capricorn Sun Cap moon and cap rising...what does that say about me?	I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What does this say about me?	1
6	13	14	Should I buy tiago?	What keeps childern active and far from phone and video games?	0
7	15	16	How can I be a good geologist?	What should I do to be a great geologist?	1
8	17	18	When do you use ㄱ instead of ㅋ?	When do you use "&" instead of "and"?	0
9	19	20	Motorola (company): Can I hack my Charter Motorolla DCX3400?	How do I hack Motorola DCX3400 for free internet?	0
10	21	22	Method to find separation of slits using fresnel biprism?	What are some of the things technicians can tell about the durability and reliability of Laptops and its compo...	0
11	23	24	How do I read and find my YouTube comments?	How can I see all my Youtube comments?	1
12	25	26	What can make Physics easy to learn?	How can you make physics easy to learn?	1
13	27	28	What was your first sexual experience like?	What was your first sexual experience?	1
14	29	30	What are the laws to change your status from a student visa to a green card in the US, how do t...	What are the laws to change your status from a student visa to a green card in the US? How do they compa...	0
15	31	32	What would a Trump presidency mean for current international master's students on an F1 visa?	How will a Trump presidency affect the students presently in US or planning to study in US?	1
16	33	34	What does manipulation mean?	What does manipulation means?	1
17	35	36	Why do girls want to be friends with the guy they reject?	How do guys feel after rejecting a girl?	0
18	37	38	Why are so many Quora users posting questions that are readily answered on Google?	Why do people ask Quora questions which can be answered easily by Google?	1
19	39	40	Which is the best digital marketing institution in banglore?	Which is the best digital marketing institute in Pune?	0
20	41	42	Why do rockets look white?	Why are rockets and boosters painted white?	1
21	43	44	What's causing someone to be jealous?	What can I do to avoid being jealous of someone?	0
22	45	46	What are the questions should not ask on Quora?	Which question should I ask on Quora?	0
23	47	48	How much is 30 kV in HP?	Where can I find a conversion chart for CC to horsepower?	0
24	49	50	What does it mean that every time I look at the clock the numbers are the same?	How many times a day do a clock's hands overlap?	0
25	51	52	What are some tips on making it through the job interview process at Medicines?	What are some tips on making it through the job interview process at Foundation Medicine?	0
26	53	54	What is web application?	What is the web application framework?	0
27	55	56	Does society place too much importance on sports?	How do sports contribute to the society?	0
28	57	58	What is best way to make money online?	What is best way to ask for money online?	0
29	59	60	How should I prepare for CA final law?	How one should know that he/she completely prepare for CA final exam?	1
30	61	62	What's one thing you would like to do better?	What's one thing you do despite knowing better?	0
31	63	64	What are some special cares for someone with a nose that gets stuffy during the night?	How can I keep my nose from getting stuffy at night?	1
32	65	66	What Game of Thrones villain would be the most likely to give you mercy?	What Game of Thrones villain would you most like to be at the mercy of?	1
33	67	68	Does the United States government still blacklist (employment, etc.) some United States citizens...	How is the average speed of gas molecules determined?	0
34	69	70	What is the best travel website in spain?	What is the best travel website?	0
35	71	72	Why do some people think Obama will try to take their guns away?	Has there been a gun control initiative to take away guns people already own?	0
36	73	74	I'm a 19-year-old. How can I improve my skills or what should I do to become an entrepreneur i...	I am a 19 year old guy. How can I become a billionaire in the next 10 years?	0
37	75	76	When a girlfriend asks her boyfriend "Why did you choose me? What makes you want to be with...	My girlfriend said that we should end this because she is confused about her feelings for me. I wished her ...	0
38	77	78	How do we prepare for UPSC?	How do I prepare for civil service?	1
39	79	80	What is the stall speed and AOA of an f-14 with wings fully swept back?	Why did aircraft stop using variable-sweep wings, like those on an F-14?	0
40	81	82	Why do Slavs squat?	Will squats make my legs thicker?	0

Figure 3 First lines of the raw training dataset

Our columns of interest are *question1, question2* and *is_duplicate* which are self explanatory.
The training data is stored in `train_df` and the test data in `test_df` and both are pandas DataFrames.
Only difference between `train_df` and `test_df` is that the latter doesn't have the *is_duplicate* column

```
train_df = pd.read_csv(TRAIN_CSV)
test_df = pd.read_csv(TEST_CSV)
```

Next I created a helper function named `text_to_word_list(text)` which takes string as input and outputs a list where each entry is a single word from the text and does some preprocessing (removing specific signs etc).

Now our aim is to have the ability to turn a word into its embedding given by word2vec, in order to do that we will need to build:

- `vocabulary` which is a `dict` where the keys are words (`str`) and values are the corresponding indices (a unique id as `int`).

- `inverse_vocabulary` which is a list of words (`str`) where the index in the list is the matching id (from `vocabulary`).
  We reserve the first place for an all zeros embedding—this is needed for the zero padding to be ignored.

We also use `gensim.models.KeyedVectors` to load the word2vec embeddings.
Throughout the code only 2 functions of this class will be used, `.vocab` which will hold all of the word2vec words and `.word_vec(word)` which takes a word and returns its embedding.
Finally we will use `nltk`'s English stopwords and store them in `stops`.

Creating `vocabulary` and `inverse_vocabulary`:

```
vocabulary = dict()
```

```
inverse_vocabulary = ['<unk>']
# '<unk>' will never be used, it is only a placeholder for
the
# [0, 0, ....0] embedding
```

```
word2vec =
KeyedVectors.load_word2vec_format(EMBEDDING_FILE,binary=True
)

questions_cols = ['question1', 'question2']

# Iterate over the questions only of both training and test
datasets
for dataset in [train_df, test_df]:
    for index, row in dataset.iterrows():

        # Iterate through the text of both questions of the
row
        for question in questions_cols:

            q2n = []  # q2n -> question numbers
representation
            for word in text_to_word_list(row[question]):

                # Check for unwanted words
                if word in stops and word not in
word2vec.vocab:
                    continue

                if word not in vocabulary:
                    vocabulary[word] =
len(inverse_vocabulary)
                    q2n.append(len(inverse_vocabulary))
                    inverse_vocabulary.append(word)
                else:
                    q2n.append(vocabulary[word])
```

```
            # Replace questions with lists of word indices
            dataset.set_value(index, question, q2n)
```

So now we have `vocabulary`, `inverse_vocabulary` and both `train_df` and `test_df` converted to word indices, screenshot below.

id	qid1	qid2	question1	question2	is_duplicate
0	1	2	[1, 2, 3, 4, 5, 4, 6, 7, 8, 9, 10, 8, 11]	[1, 2, 3, 4, 5, 4, 6, 7, 8, 9, 10]	0
1	3	4	[1, 2, 3, 12, 13, 14, 15, 16, 15, 17, 18]	[1, 19, 20, 21, 3, 22, 23, 24, 3, 13, 14, 15, 16, 15, 17, 18, 25]	0
2	5	6	[26, 27, 16, 28, 3, 29, 30, 31, 32, 33, 34, 35]	[26, 27, 31, 29, 36, 37, 5, 38, 39, 40]	0
3	7	8	[41, 42, 16, 43, 44, 45, 26, 27, 16, 46, 47]	[48, 3, 49, 50, 51, 52, 53, 54, 51, 2, 55, 5, 54, 52]	0
4	9	10	[56, 57, 58, 8, 59, 60, 61, 62, 63, 64, 65, 66]	[56, 67, 19, 68, 8, 62, 59]	0
5	11	12	[69, 70, 16, 42, 71, 72, 73, 74, 73, 75, 1, 76, 7...	[16, 42, 81, 71, 72, 74, 82, 8, 71, 1, 76, 83, 78, 79, 80]	1
6	13	14	[84, 16, 85, 86]	[1, 87, 88, 89, 90, 91, 92, 93, 94]	0
7	15	16	[26, 27, 16, 36, 95, 96]	[1, 84, 16, 97, 36, 98, 96]	1
8	17	18	[50, 97, 99, 100, 101]	[50, 97, 99, 100, 101]	0
9	19	20	[102, 103, 70, 27, 16, 104, 30, 105, 106, 107]	[26, 97, 16, 104, 102, 107, 108, 109, 31]	0
10	21	22	[110, 48, 111, 112, 34, 113, 114]	[1, 115, 116, 3, 117, 118, 27, 119, 79, 3, 120, 121, 122, 123, 124]	0
11	23	24	[26, 97, 16, 125, 48, 30, 126, 127]	[26, 27, 16, 128, 129, 30, 126, 127]	1
12	25	26	[1, 27, 130, 131, 132, 133]	[26, 27, 99, 130, 131, 132, 133]	1
13	27	28	[1, 134, 135, 136, 137, 138, 139]	[1, 134, 135, 136, 137, 138]	1
14	29	30	[1, 115, 3, 140, 141, 135, 142, 91, 143, 144, ...	[1, 115, 3, 140, 141, 135, 142, 91, 143, 144, 145, 146, 8, 3, 147, 26, 97, 14...	0
15	31	32	[1, 19, 153, 154, 155, 108, 156, 157, 158, 15...	[26, 164, 153, 154, 165, 3, 160, 166, 8, 147, 167, 168, 169, 8, 147]	1
16	33	34	[1, 76, 170, 155]	[1, 76, 170, 171]	1
17	35	36	[41, 97, 172, 173, 36, 174, 175, 3, 176, 148, ...	[26, 97, 178, 179, 180, 181, 182]	0
18	37	38	[41, 115, 183, 184, 185, 186, 187, 188, 77, 1...	[41, 97, 192, 193, 185, 188, 56, 27, 36, 190, 194, 5, 191]	1
19	39	40	[56, 2, 3, 195, 196, 197, 198, 8, 199]	[56, 2, 3, 195, 196, 197, 200, 8, 201]	0
20	41	42	[41, 97, 202, 203, 204]	[41, 115, 202, 205, 206, 204]	1
21	43	44	[1, 2, 207, 208, 36, 209]	[1, 27, 16, 97, 210, 211, 209, 208]	0
22	45	46	[1, 115, 3, 188, 84, 212, 193, 161, 185]	[56, 213, 84, 16, 193, 161, 185]	0
23	47	48	[26, 214, 2, 215, 216, 8, 217]	[218, 27, 16, 48, 219, 220, 108, 221, 222]	0
24	49	50	[1, 76, 47, 155, 77, 223, 224, 16, 203, 225, 3,...	[26, 184, 229, 230, 97, 226, 159, 231, 232]	0
25	51	52	[1, 115, 116, 233, 161, 234, 47, 39, 3, 235, 2...	[1, 115, 116, 233, 161, 234, 47, 39, 3, 235, 236, 237, 225, 239, 240]	0
26	53	54	[1, 2, 241, 242]	[1, 2, 3, 241, 242, 243]	0
27	55	56	[76, 244, 245, 246, 214, 247, 161, 248]	[26, 97, 248, 249, 3, 244]	0
28	57	58	[1, 2, 195, 250, 130, 251, 252]	[1, 2, 195, 250, 193, 108, 251, 252]	0
29	59	60	[26, 84, 16, 253, 108, 254, 255, 256]	[26, 57, 84, 257, 77, 258, 259, 260, 253, 108, 254, 255, 261]	1
30	61	62	[1, 2, 57, 262, 99, 19, 139, 97, 263]	[1, 2, 57, 262, 99, 97, 264, 265, 263]	0
31	63	64	[1, 115, 116, 266, 267, 108, 208, 175, 268, 7...	[26, 27, 16, 273, 30, 268, 91, 274, 270, 225, 272]	1
32	65	66	[1, 275, 276, 277, 19, 36, 3, 278, 279, 280, 9...	[1, 275, 276, 277, 19, 99, 278, 139, 36, 225, 3, 281]	1
33	67	68	[76, 3, 282, 283, 23, 284, 285, 286, 287, 116,...	[26, 2, 3, 293, 29, 294, 295, 296]	0
34	69	70	[1, 2, 3, 195, 297, 298, 8, 299]	[1, 2, 3, 195, 297, 298]	0
35	71	72	[41, 97, 116, 192, 300, 301, 164, 302, 303, 2...	[306, 307, 308, 309, 310, 311, 303, 305, 304, 192, 312, 313]	0
36	73	74	[16, 42, 314, 15, 315, 15, 316, 26, 27, 16, 31...	[16, 42, 314, 315, 316, 176, 26, 27, 16, 319, 324, 8, 3, 321, 325, 323]	0
37	75	76	[50, 326, 327, 328, 329, 41, 330, 99, 331, 80,...	[30, 326, 334, 77, 335, 84, 336, 83, 289, 259, 2, 337, 79, 328, 338, 108, 80...	0
38	77	78	[26, 97, 335, 253, 108, 346]	[26, 97, 16, 253, 108, 347, 348]	1
39	79	80	[1, 2, 3, 349, 29, 350, 162, 351, 15, 352, 175,...	[41, 330, 356, 357, 34, 358, 15, 359, 353, 139, 360, 161, 162, 351, 15, 352]	0
40	81	82	[41, 97, 361, 362]	[164, 363, 130, 30, 364, 365]	0

Figure 4 First lines of the converted to word indices training dataset.

Notice we start at index 1, index 0 is reserved for the zero padding. Also notice I do not exclude stopwords if they have embeddings, I will later give them a random representation—this is done for the sake of simplicity, a far better approach will be to train your own embeddings to better capture context of the problem.

Embedding matrix

Our next goal is to create the embedding matrix.
We will assign each word its word2vec embedding and leave the

unrecognized ones (less than 0.5%) random.
Also we keep the first index all zeros.

```
embedding_dim = 300
```

```
# This will be the embedding matrix
embeddings = 1 * np.random.randn(len(vocabulary) + 1,
embedding_dim)
embeddings[0] = 0  # So that the padding will be ignored


# Build the embedding matrix
for word, index in vocabulary.items():
    if word in word2vec.vocab:
        embeddings[index] = word2vec.word_vec(word)
```

Great, we have our embedding matrix in place.

Data preparation

In order to prepare our data for use in Keras we have to do two things:

- Split our data to 'left' and 'right' inputs (one for each side of the MaLSTM)
- Pad all of the word number sequences with zeros

We will also create a validation dataset, to measure our model using scikit-learn's `train_test_split` function—it keeps the labels distribution between the datasets by default.
In `max_seq_length` we have the length of the longest question, and here is the code

```
# Split to train validation
validation_size = 40000
training_size = len(train_df) - validation_size

X = train_df[questions_cols]
Y = train_df['is_duplicate']

X_train, X_validation, Y_train, Y_validation =
train_test_split(X, Y, test_size=validation_size)

# Split to dicts
X_train = {'left': X_train.question1, 'right':
X_train.question2}
X_validation = {'left': X_validation.question1, 'right':
```

```
X_validation.question2}
X_test = {'left': test_df.question1, 'right':
test_df.question2}

# Convert labels to their numpy representations
Y_train = Y_train.values
Y_validation = Y_validation.values

# Zero padding
for dataset, side in itertools.product([X_train,
X_validation], ['left', 'right']):
    dataset[side] = pad_sequences(dataset[side],
maxlen=max_seq_length)
```

`itertools.product` simply gives all the combinations between the two lists.

Model

Now we create the model itself.
Most of the code is pretty clear just by reading it but I would like to take a moment to talk about Keras `Merge` layer.
The `Merge` layer allows us to merge elements with some built in methods, but also supports custom methods and this is where it comes in handy since we need to "merge" our two LSTMs output using the MaLSTM similarity function.
First lets define the MaLSTM similarity function.

```
def exponent_neg_manhattan_distance(left, right):
    return K.exp(-K.sum(K.abs(left-right), axis=1,
keepdims=True))
```

Now lets build the model (using functional API)

```
# The visible layer
left_input = Input(shape=(max_seq_length,), dtype='int32')
right_input = Input(shape=(max_seq_length,), dtype='int32')

embedding_layer = Embedding(len(embeddings), embedding_dim,
weights=[embeddings], input_length=max_seq_length,
trainable=False)

# Embedded version of the inputs
encoded_left = embedding_layer(left_input)
encoded_right = embedding_layer(right_input)

# Since this is a siamese network, both sides share the same
LSTM
```

```
shared_lstm = LSTM(n_hidden)

left_output = shared_lstm(encoded_left)
right_output = shared_lstm(encoded_right)

# Calculates the distance as defined by the MaLSTM model
malstm_distance = Merge(mode=lambda x:
exponent_neg_manhattan_distance(x[0], x[1]),
output_shape=lambda x: (x[0][0], 1))([left_output,
right_output])

# Pack it all up into a model
malstm = Model([left_input, right_input], [malstm_distance])
```

Don't you love it how simple Keras is?
Next we define the optimizer and compile our model.

```
# Adadelta optimizer, with gradient clipping by norm
optimizer = Adadelta(clipnorm=gradient_clipping_norm)

malstm.compile(loss='mean_squared_error',
optimizer=optimizer, metrics=['accuracy'])
```

Now all that is left is to train it!

Training and results

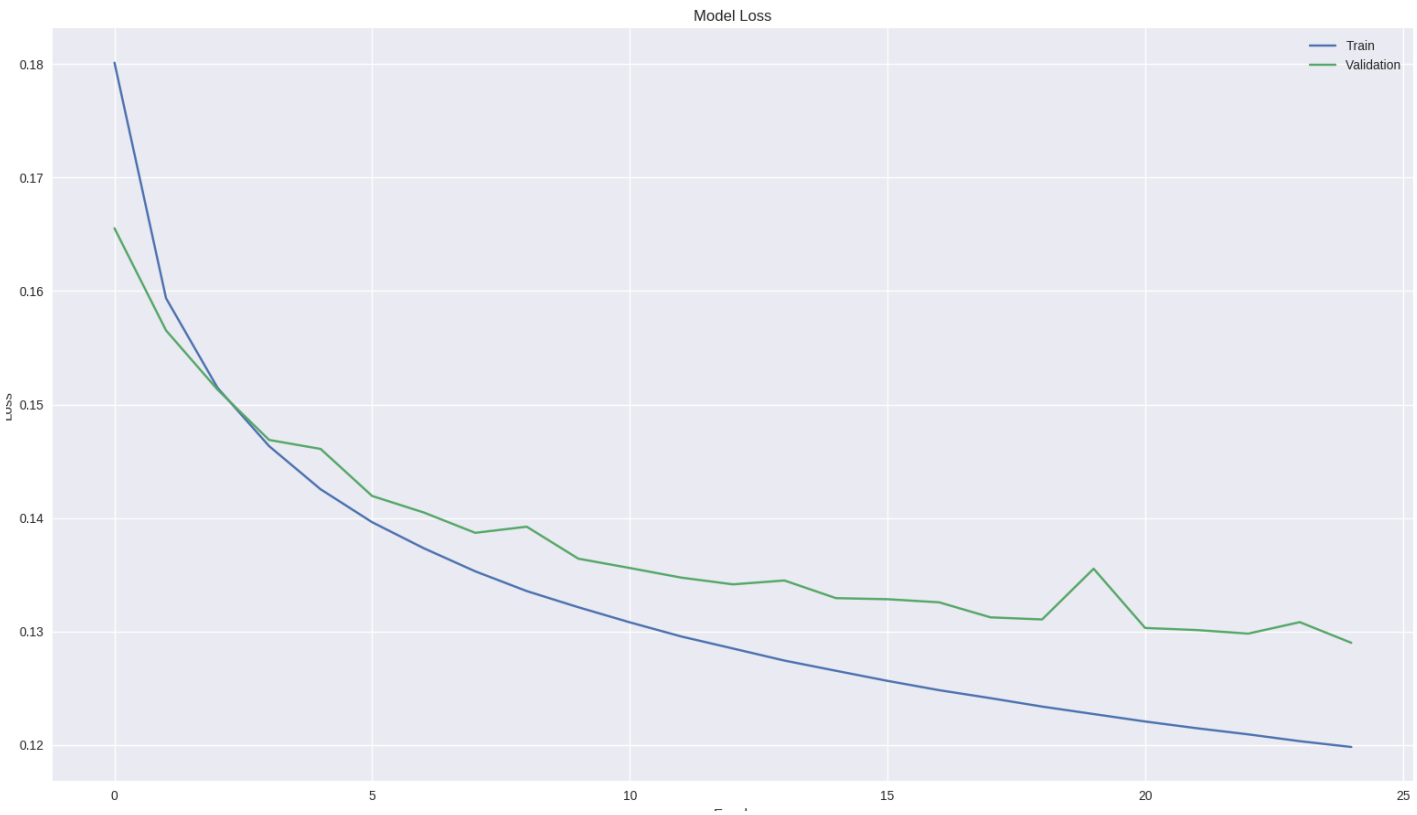
I launched it on my local machine, which has a GTX 1070.
The whole script and including preparation and training took about 21 hours.

```
malstm_trained = malstm.fit([X_train['left'],
X_train['right']], Y_train, batch_size=batch_size,
nb_epoch=n_epoch,
                            validation_data=
([X_validation['left'], X_validation['right']],
Y_validation),
                            callbacks=[checkpointer])
```

To properly evaluate the model performance, lets plot training data vs validation data accuracy and loss

Accuracy:

Loss:


Model Loss
Train
Validation
0.18
0.17
0.16
0.15
0.14
0.13
0.12
Loss
0
5
10
15
20
25

So just like that out of the box, seems that MaLSTM is doing OK, getting an 82.5% accuracy rate on the validation data.

Improvements

This article and the code as well was written with simplicity in mind. To achieve state of the art results tuning and adjusting to your specific use case will always be needed.

Here are some thoughts of mine where can we go from here:

- Use transfer learning just like in the article, to pretrain your LSTM
- Train word embeddings, on the data questions.
- Create new data, from my data exploration (is not present in this article) there are some duplicate questions that appear twice against different questions, it is possible that using this we can create more data.
- Create new data using data augmentation swapping words from the text with synonyms
- Choose a different optimizer, I've read a lot that Adadelta doesn't perform as good other methods when finely tuned.

The double edged sword of deep learning is that this list is infinite, I've put there a tiny bit of possible things that keep us within the MaLSTM architecture.

If you want to explore further and to get the best results possible, I advise you to look at the discussion about the competition—they achieved some really impressive results there using various models and ensembling.

. . .

The purpose of this post was to put into work a good article that implements some things that you don't really see in tutorials and stuff, I hope this post and the code has taught you a thing or two. Happy coding :)