Keras and neural nets for NLP: A whirlwind tutorial

Nelson Liu

April 3, 2017

What is Keras?

- Higher level API for building neural networks with Tensorflow and Theano
- Much more user-friendly
- Modular, it represents a model as a graph of stand-alone modules that you can swap and match.
- Extensible, you can easily add your own “modules” (actually called Layers)
- Works in Python, so you get the benefits of the PyData ecosystem (NumPy/SciPy/Matplotlib)

Computation Graphs

- Tensorflow and Theano (and thus Keras) treat models as a directed graph.
- Consider the following expressions and their associated graph:
  - c=a+b
  - d=b+1
  - e=c*d

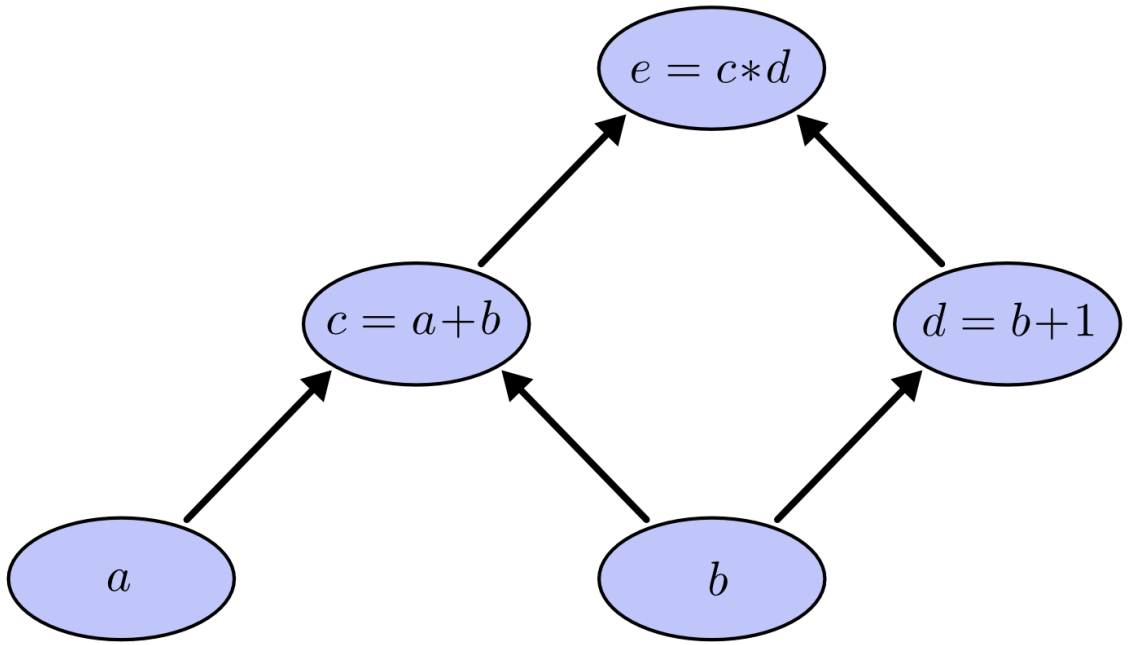


Image: Christopher Olah

Computation Graphs for ML

- Two main steps:
  - Step 1: Assemble the graph, which then needs to be compiled
  - Step 2: Feed data (tensors) through the graph (data flow).
- Keras makes both of these steps much easier.

A linear regression model as a graph

- y = Wx + b

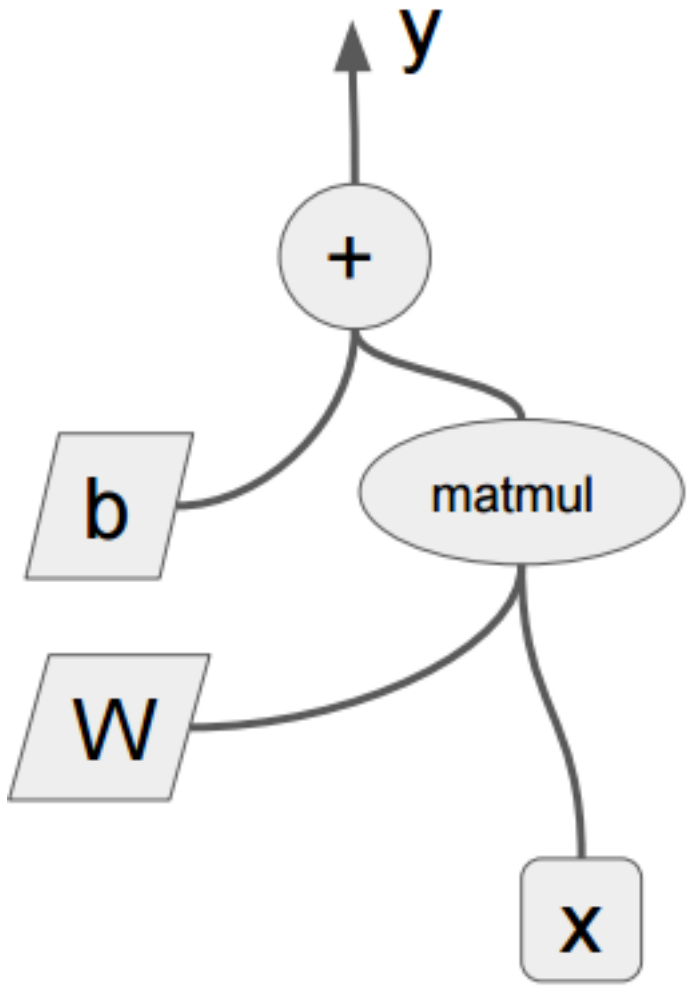


Image: David Rios Y Valles

Neural Networks for NLP

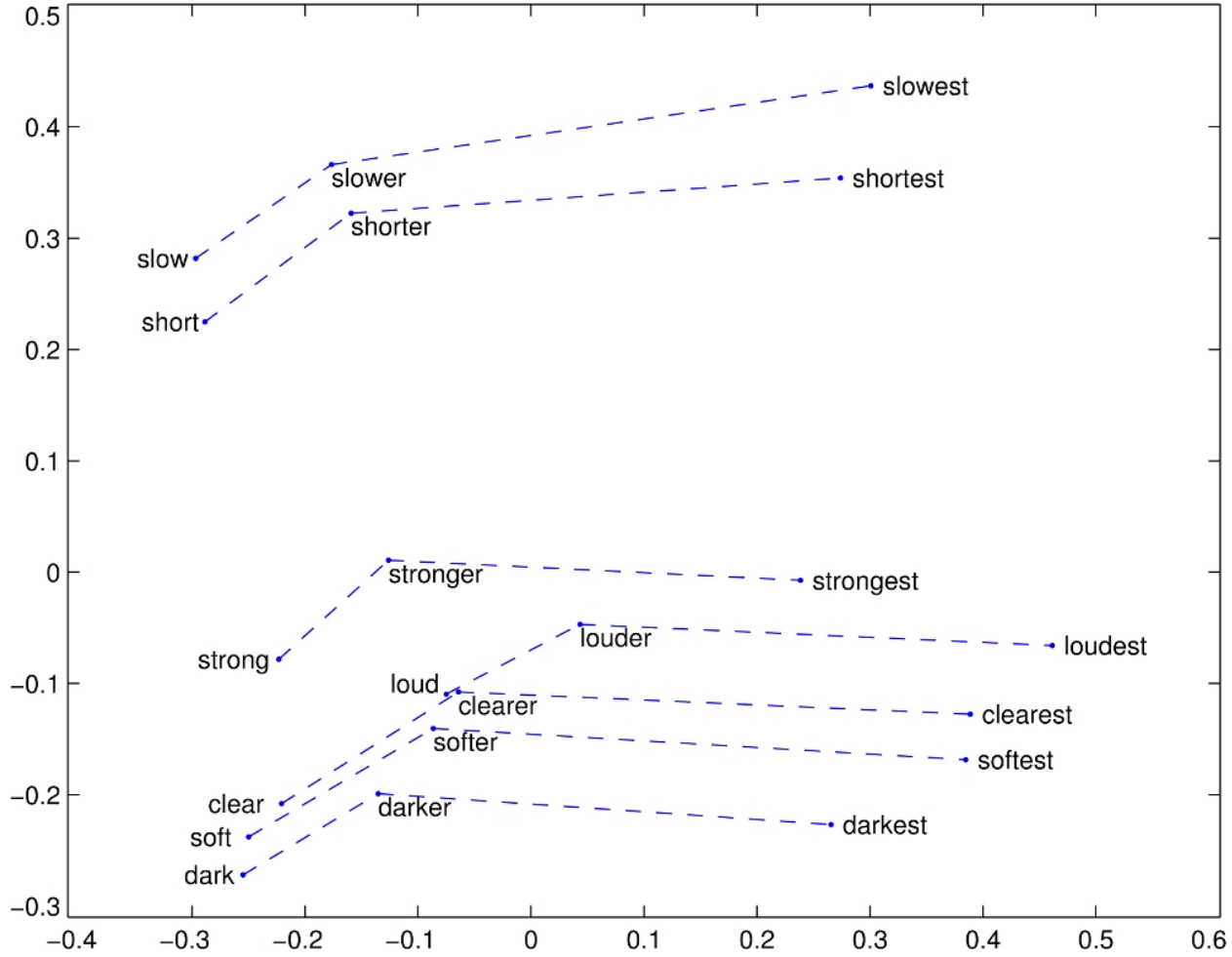
- Let's say we have the two sentences: "Man bit dog" and "Dog bit man" and we want to compute some notion of semantic similarity between them.
- Text data isn't suited for learning algorithms, so what do we do?
  - Replace each word with a numeric index!
  - "Man bit dog" -> [2, 3, 4] and "Dog bit man" -> [4, 3, 2].
- But these numeric indices don't actually encode any meaning
  - So let's represent each word with a vector, or "embed" it into a higher-dimensional space.

Embeddings

- Embeddings are a way to provide a model with the “semantic meaning” behind words.
  - Each word is assigned a ”vector”, which is supposed to contain a representation of its meaning. Popular algorithms for learning word embeddings are word2vec and GloVe.

Embedding visualization (GloVe)

- Plotting these GloVe / Word2Vec-trained embeddings in a 2D space shows that they encode some notion of meaning. In this case, the superlative relation is shown.



Embedding our example sentences

- “Man bit dog” -> [2, 3, 4] -> [[0.1, 0.78], [0.38, 0.18], [0.19, 0.8]]
- “Dog bit man” -> [4, 3, 2] -> [[0.19, 0.8], [0.38, 0.18], [0.1, 0.78]]

Encoding our sentence

- Great, so now our sentence went from a sequence of words, to a sequence of integers, to a sequence of vectors. What's next?
- It'd be nice if we could "encode" our sentence, and compress the sequence of vectors into one vector that represents the meaning of the sentence.
- Representing sentences as vectors would allow us to use a notion of vector similarity to quantify how semantically related the sentences are.

The Neural Bag of Words

- Let’s (naively) represent the sentence as the average of all of its constituent words!
  - This is called a ”neural bag of words”
- There are several issues with this approach:
  - Word ordering is lost, e.g. “Dog bit man” has the same sentence vector as “Man bit dog”
  - Word ordering seems pretty important for getting good sentence representations.

Enter Recurrent Neural Networks

- Recurrent neural networks, especially modified versions like the Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) can encode a sequence of vectors in a way that seems to preserve word order.

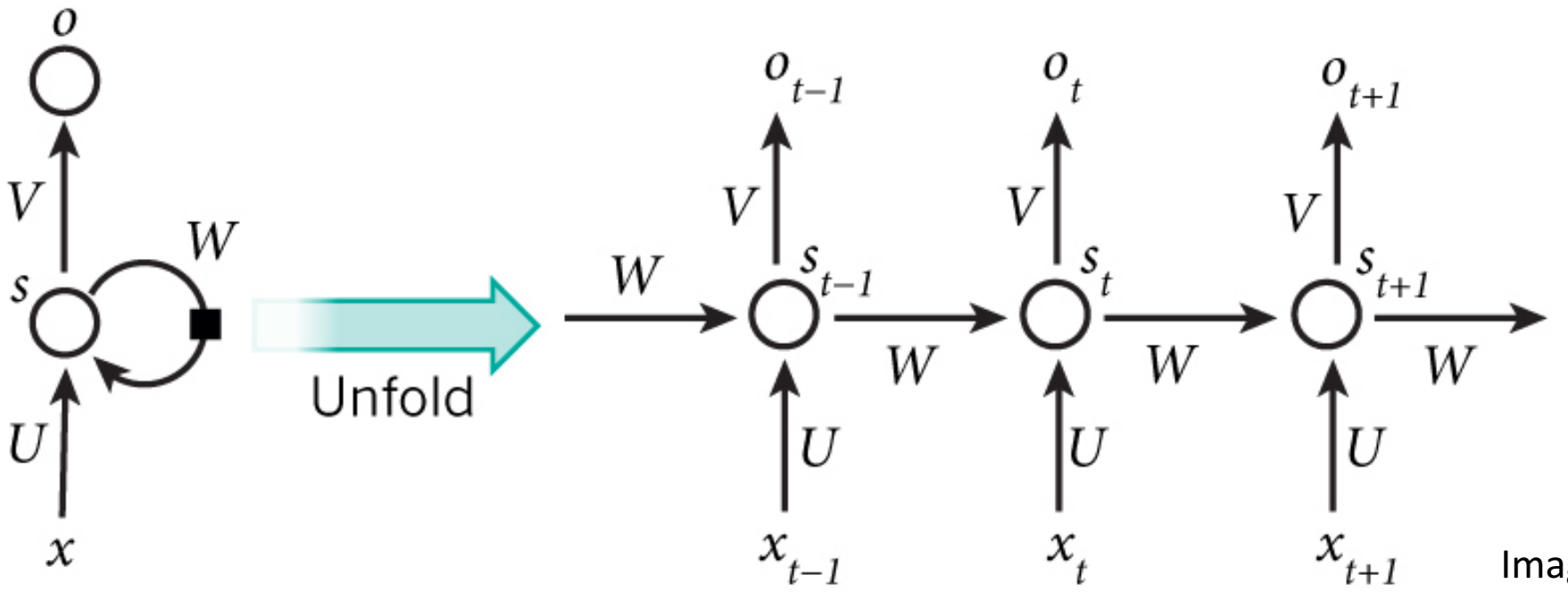


Image: Nature

“The BiLSTM Hegemony”

- “If you don’t know anything about natural language processing in 2017, here’s what you should know about how to do it. And this is actually a highly depressing result…basically, if you want to do a NLP task (it doesn’t matter what the task is) what you should do is throw a bidirectional LSTM network at it, if it’s a model that involves some need for information flow, one augmented with attention. Basically, you can do this and it will be pretty much the state of the art of anything anybody knows how to do” – Christopher Manning

So what's the "Bi" in "BiLSTM"?

- "BiLSTM" is short for "Bidirectional LSTM"
  - Run an input sequence through an LSTM as normal
  - Then, reverse the input sequence and run it through a LSTM
  - Concatenate the outputs.
  - "Reading both directions" helps with encoding the sentence's meaning.

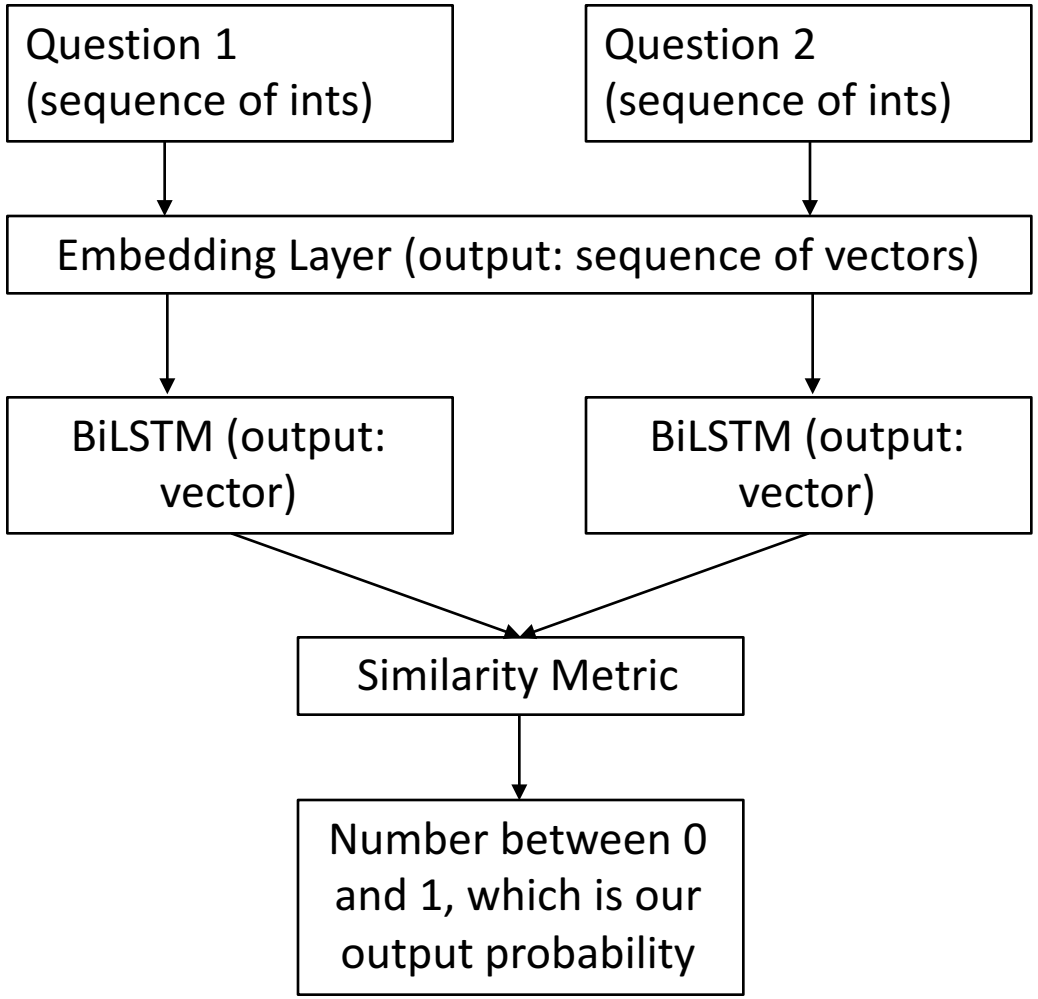
BiLSTM as an encoder

- So we run our sentence (sequence of vectors) through a BiLSTM, and get a single vector for the sentence.
- Repeat for the other sentence to get another vector
- Now, we can use a measure of vector similarity (manhattan distance, cosine similarity, euclidean distance) and classify the sentences as semantically the same or not.

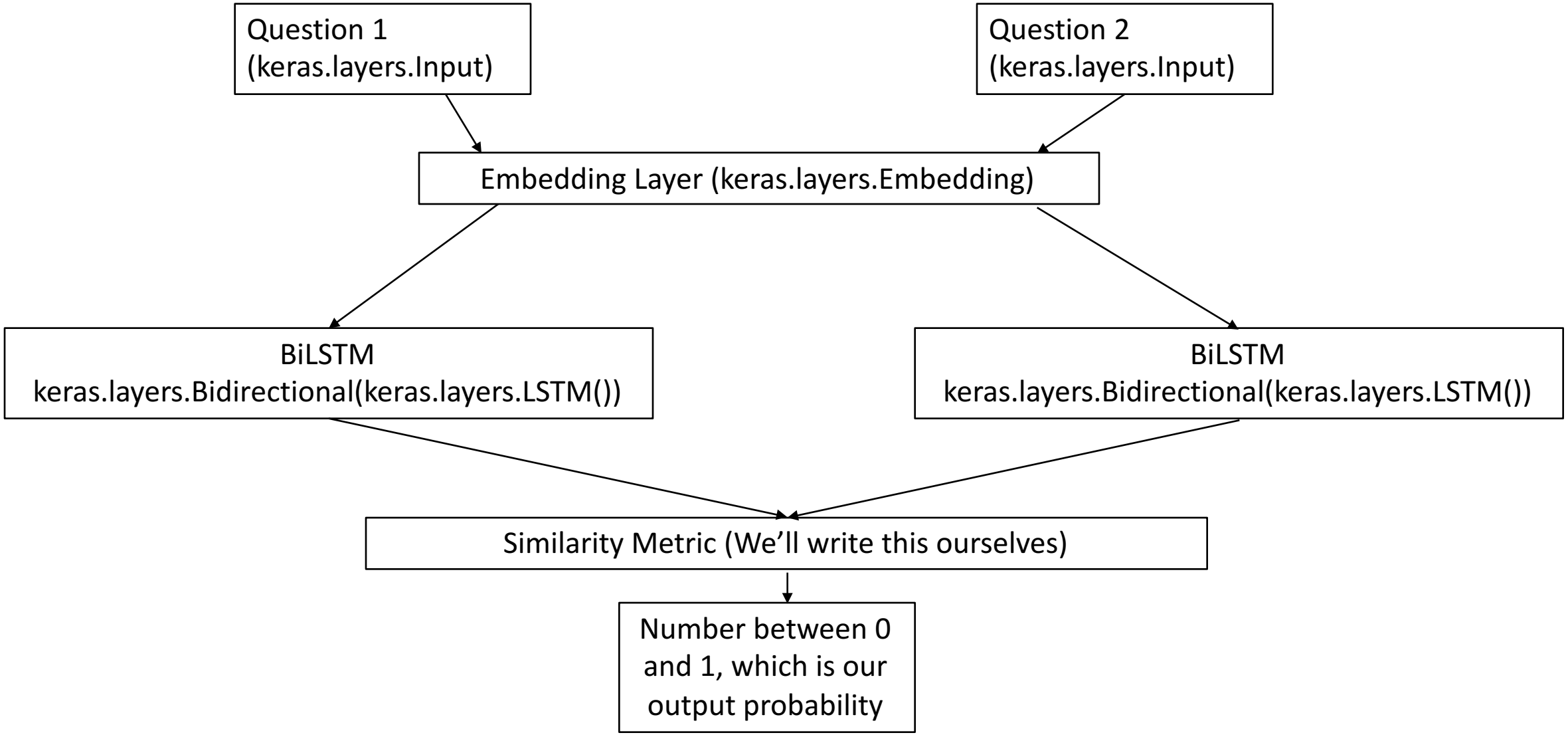
The task today

- Given a pair of questions from Quora, figure out if they’re duplicates or not.
- One way to approach this would be to get vector representations of both sentences, and then compare them.
- Let’s build a model to tackle this in Keras!

Proposed model as a computation graph



Proposed model as Keras Layer Objects



One last caveat: Padding

- Neural networks take as input fixed-size vectors. What do we do since our questions are sequences of integers with variable length?
- The solution is to “pad” (add dummy 0’s that don’t mean anything) the shorter questions to the length of the longest instance.
- Keras automatically infers that these 0’s are padding, and doesn’t take them into account when doing model computations (masking).
- We can also set a manual “max length” and pad shorter sequences to this length / truncate sequences to this length. This is useful if we have one outlier sequence that is super long, and it’s wasteful to pad everything to that length.