Here are some properties of the model that you may notice:

**Pre-attention and Post-attention LSTMs on both sides of the attention mechanism**

* There are two separate LSTMs in this model (see diagram on the left): pre-attention and post-attention LSTMs.
* *Pre-attention* Bi-LSTM is the one at the bottom of the picture is a Bi-directional LSTM and comes *before* the attention mechanism.
  + The attention mechanism is shown in the middle of the left-hand diagram.
  + The pre-attention Bi-LSTM goes through TxTx time steps
* *Post-attention* LSTM: at the top of the diagram comes *after* the attention mechanism.
  + The post-attention LSTM goes through TyTy time steps.
* The post-attention LSTM passes the hidden state s⟨t⟩s⟨t⟩ and cell state c⟨t⟩c⟨t⟩ from one time step to the next.

**An LSTM has both a hidden state and cell state**

* In the lecture videos, we were using only a basic RNN for the post-attention sequence model
  + This means that the state captured by the RNN was outputting only the hidden state s⟨t⟩s⟨t⟩.
* In this assignment, we are using an LSTM instead of a basic RNN.
  + So the LSTM has both the hidden state s⟨t⟩s⟨t⟩ and the cell state c⟨t⟩c⟨t⟩.

**Each time step does not use predictions from the previous time step**

* Unlike previous text generation examples earlier in the course, in this model, the post-attention LSTM at time tt does not take the previous time step's prediction y⟨t−1⟩y⟨t−1⟩ as input.
* The post-attention LSTM at time 't' only takes the hidden state s⟨t⟩s⟨t⟩ and cell state c⟨t⟩c⟨t⟩ as input.
* We have designed the model this way because unlike language generation (where adjacent characters are highly correlated) there isn't as strong a dependency between the previous character and the next character in a YYYY-MM-DD date.

**Concatenation of hidden states from the forward and backward pre-attention LSTMs**

* a→⟨t⟩a→⟨t⟩: hidden state of the forward-direction, pre-attention LSTM.
* a←⟨t⟩a←⟨t⟩: hidden state of the backward-direction, pre-attention LSTM.
* a⟨t⟩=[a→⟨t⟩,a←⟨t⟩]a⟨t⟩=[a→⟨t⟩,a←⟨t⟩]: the concatenation of the activations of both the forward-direction a→⟨t⟩a→⟨t⟩ and backward-directions a←⟨t⟩a←⟨t⟩ of the pre-attention Bi-LSTM.

**Computing "energies"**e⟨t,t′⟩e⟨t,t′⟩**as a function of**s⟨t−1⟩s⟨t−1⟩**and**a⟨t′⟩a⟨t′⟩

* Recall in the lesson videos "Attention Model", at time 6:45 to 8:16, the definition of "e" as a function of s⟨t−1⟩s⟨t−1⟩ and a⟨t⟩a⟨t⟩.
  + "e" is called the "energies" variable.
  + s⟨t−1⟩s⟨t−1⟩ is the hidden state of the post-attention LSTM
  + a⟨t′⟩a⟨t′⟩ is the hidden state of the pre-attention LSTM.
  + s⟨t−1⟩s⟨t−1⟩ and a⟨t⟩a⟨t⟩ are fed into a simple neural network, which learns the function to output e⟨t,t′⟩e⟨t,t′⟩.
  + e⟨t,t′⟩e⟨t,t′⟩ is then used when computing the attention a⟨t,t′⟩a⟨t,t′⟩ that y⟨t⟩y⟨t⟩ should pay to a⟨t′⟩a⟨t′⟩.
* The diagram on the right of figure 1 uses a RepeatVector node to copy s⟨t−1⟩s⟨t−1⟩'s value TxTx times.
* Then it uses Concatenation to concatenate s⟨t−1⟩s⟨t−1⟩ and a⟨t⟩a⟨t⟩.
* The concatenation of s⟨t−1⟩s⟨t−1⟩ and a⟨t⟩a⟨t⟩ is fed into a "Dense" layer, which computes e⟨t,t′⟩e⟨t,t′⟩.
* e⟨t,t′⟩e⟨t,t′⟩ is then passed through a softmax to compute α⟨t,t′⟩α⟨t,t′⟩.
* Note that the diagram doesn't explicitly show variable e⟨t,t′⟩e⟨t,t′⟩, but e⟨t,t′⟩e⟨t,t′⟩ is above the Dense layer and below the Softmax layer in the diagram in the right half of figure 1.
* We'll explain how to use RepeatVector and Concatenation in Keras below.

**Implementation Details**

Let's implement this neural translator. You will start by implementing two functions: one\_step\_attention() and model().

**one\_step\_attention**

* The inputs to the one\_step\_attention at time step tt are:
  + [a<1>,a<2>,...,a<Tx>][a<1>,a<2>,...,a<Tx>]: all hidden states of the pre-attention Bi-LSTM.
  + s<t−1>s<t−1>: the previous hidden state of the post-attention LSTM
* one\_step\_attention computes:
  + [α<t,1>,α<t,2>,...,α<t,Tx>][α<t,1>,α<t,2>,...,α<t,Tx>]: the attention weights
  + context⟨t⟩context⟨t⟩: the context vector:

context<t>=∑t′=1Txα<t,t′>a<t′>(1)(1)context<t>=∑t′=1Txα<t,t′>a<t′>

***Clarifying 'context' and 'c'***

* In the lecture videos, the context was denoted c⟨t⟩c⟨t⟩
* In the assignment, we are calling the context context⟨t⟩context⟨t⟩.
  + This is to avoid confusion with the post-attention LSTM's internal memory cell variable, which is also denoted c⟨t⟩c⟨t⟩.

**Implement one\_step\_attention**

**Exercise**: Implement one\_step\_attention().

* The function model() will call the layers in one\_step\_attention() TyTy using a for-loop.
* It is important that all TyTy copies have the same weights.
  + It should not reinitialize the weights every time.
  + In other words, all TyTy steps should have shared weights.
* Here's how you can implement layers with shareable weights in Keras:
  + Define the layer objects in a variable scope that is outside of the one\_step\_attention function. For example, defining the objects as global variables would work.
    - Note that defining these variables inside the scope of the function model would technically work, since model will then call the one\_step\_attention function. For the purposes of making grading and troubleshooting easier, we are defining these as global variables. Note that the automatic grader will expect these to be global variables as well.
  + Call these objects when propagating the input.
* We have defined the layers you need as global variables.
  + Please run the following cells to create them.
  + Please note that the automatic grader expects these global variables with the given variable names. For grading purposes, please do not rename the global variables.
* Please check the Keras documentation to learn more about these layers. The layers are functions. Below are examples of how to call these functions.
  + [RepeatVector()](https://keras.io/layers/core/#repeatvector)
  + var\_repeated = repeat\_layer(var1)
  + [Concatenate()](https://keras.io/layers/merge/" \l "concatenate" \t "_blank)
  + concatenated\_vars = concatenate\_layer([var1,var2,var3])
  + [Dense()](https://keras.io/layers/core/#dense)
  + var\_out = dense\_layer(var\_in)
  + [Activation()](https://keras.io/layers/core/#activation)
  + activation = activation\_layer(var\_in)
  + [Dot()](https://keras.io/layers/merge/#dot)

dot\_product = dot\_layer([var1,var2])

Now you can use these layers TyTy times in a for loop to generate the outputs, and their parameters will not be reinitialized. You will have to carry out the following steps:

1. Propagate the input X into a bi-directional LSTM.
   * [Bidirectional](https://keras.io/layers/wrappers/#bidirectional)
   * [LSTM](https://keras.io/layers/recurrent/#lstm)
   * Remember that we want the LSTM to return a full sequence instead of just the last hidden state.

Sample code:

sequence\_of\_hidden\_states = Bidirectional(LSTM(units=..., return\_sequences=...))(the\_input\_X)

1. Iterate for t=0,⋯,Ty−1t=0,⋯,Ty−1:
   1. Call one\_step\_attention(), passing in the sequence of hidden states [a⟨1⟩,a⟨2⟩,...,a⟨Tx⟩][a⟨1⟩,a⟨2⟩,...,a⟨Tx⟩] from the pre-attention bi-directional LSTM, and the previous hidden state s<t−1>s<t−1> from the post-attention LSTM to calculate the context vector context<t>context<t>.
   2. Give context<t>context<t> to the post-attention LSTM cell.
      * Remember to pass in the previous hidden-state s⟨t−1⟩s⟨t−1⟩ and cell-states c⟨t−1⟩c⟨t−1⟩ of this LSTM
      * This outputs the new hidden state s<t>s<t> and the new cell state c<t>c<t>.

Sample code:

next\_hidden\_state, \_ , next\_cell\_state =

post\_activation\_LSTM\_cell(inputs=..., initial\_state=[prev\_hidden\_state, prev\_cell\_state])

Please note that the layer is actually the "post attention LSTM cell". For the purposes of passing the automatic grader, please do not modify the naming of this global variable. This will be fixed when we deploy updates to the automatic grader.

* 1. Apply a dense, softmax layer to s<t>s<t>, get the output.  
     Sample code:
  2. output = output\_layer(inputs=...)
  3. Save the output by adding it to the list of outputs.

1. Create your Keras model instance.
   1. It should have three inputs:
      * X, the one-hot encoded inputs to the model, of shape (Tx,humanVocabSize)Tx,humanVocabSize)
      * s⟨0⟩s⟨0⟩, the initial hidden state of the post-attention LSTM
      * c⟨0⟩c⟨0⟩), the initial cell state of the post-attention LSTM
   2. The output is the list of outputs.  
      Sample code

model = Model(inputs=[...,...,...], outputs=...)