



# RECITATION 9

## ATTENTION NETWORKS

### HOMEWORK 4 PRIMER

— SAI NIHAR TADICHETTY

# TOPICS FOR TODAY

#	Topic	Slide #
I	What is Attention?	3
II	Typical Applications	8
III	Typical Structure of Attention Networks	12
IV	Listen Attend and Spell & Implementation Overview	16

# WHAT IS ATTENTION?

- Attention Mechanisms are very loosely based on visual attention mechanism found in humans.
- A lot is covered in the lectures about attention, so let's discuss about it from an implementation perspective.
- Programmatically, attention is just a bunch of matrix multiplications.
- Multiplications that allow the network to focus on learning from a subset of features rather than all given features. This can also be intuitively understood as a noise reduction technique.
- Human analogy would be the way we are able to focus on the words on the screen right now while conveniently ignoring the features of the monitor or keyboard and just focusing on what is important.

## Show, Attend and Tell

- Attention is applied to generate image descriptions
- CNNs are used to encode the image and RNNs are used to generate descriptions
- The images above are fused with attention maps, we can see higher weights ( $w > 0$ ) for objects of interest



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

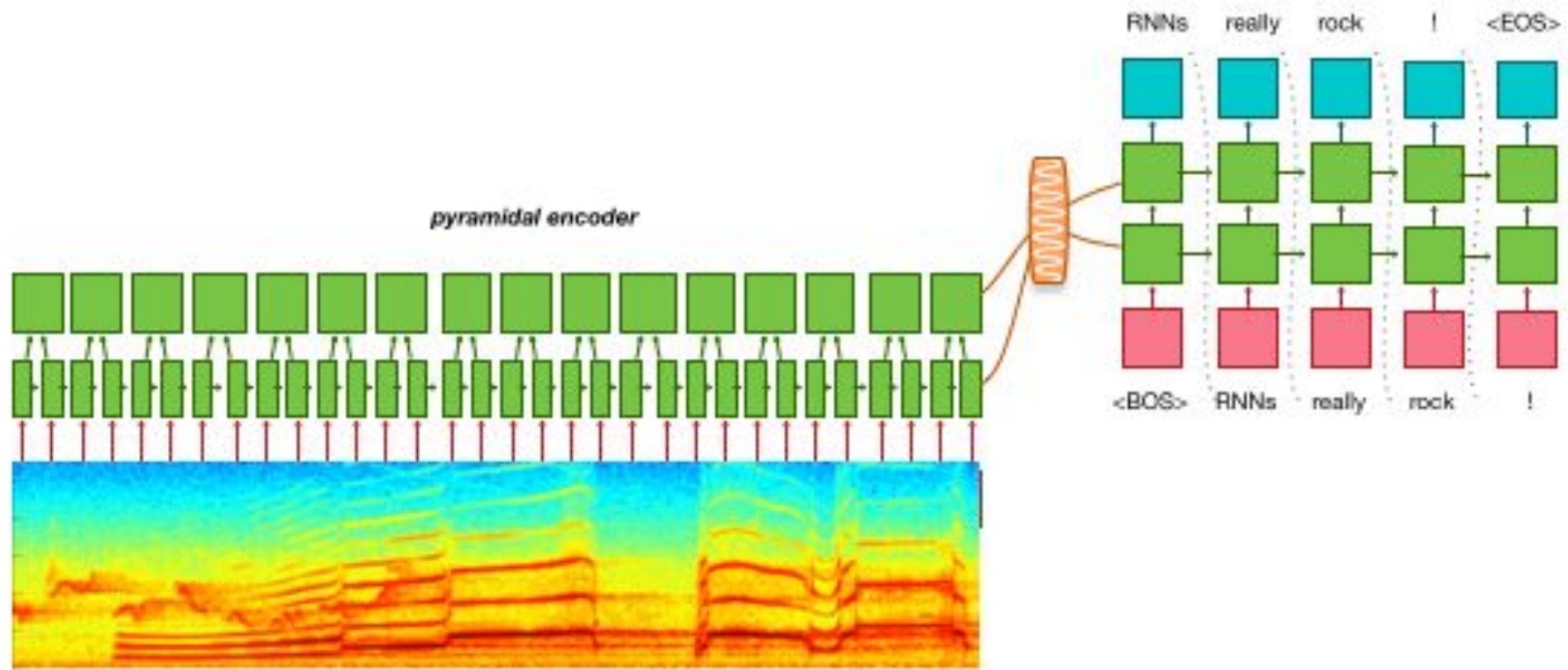



So, how are we producing these nice visualizations?

- Prof. Bhiksha discussed about hard and soft attention back in the lectures last week.
- Attention is usually computed with an attention mask of either binary values or a smooth function.
- We just superimpose attention masks and the input to obtain a masked representation which very intuitively puts forward the internal working of the network.

## Listen, Attend and Spell

- pBLSTMs are used to generate high-level representation for speech/utterances
- BLSTMs are used for language modelling
- Homework 4



- 
- In the lectures, Machine translation was taken as an example with which attention was explained.
  - We saw how useful it was to use a weighted combination of the input sentence representation and fuse with the hidden layers of each timestep of the decoder could help the network learn the weights that allow translation between languages.
  - LAS also follows the same principal (in fact most attention networks do), instead of having a sentence pass through an encoder, you pass an utterance.

# TYPICAL APPLICATIONS

- Neural Machine Translation – Covered in Lecture
- Speech to Text – Listen, Attend and Tell - Homework 4
- Visual Question Answering – Show, Attend and Tell
- Action/Scene Description – Take Visual Learning Course



## Visual Question Answering

- Does the image contain \_\_\_\_\_?
- How many \_\_\_\_\_ are in the image?
- Where is \_\_\_\_\_ in the image?



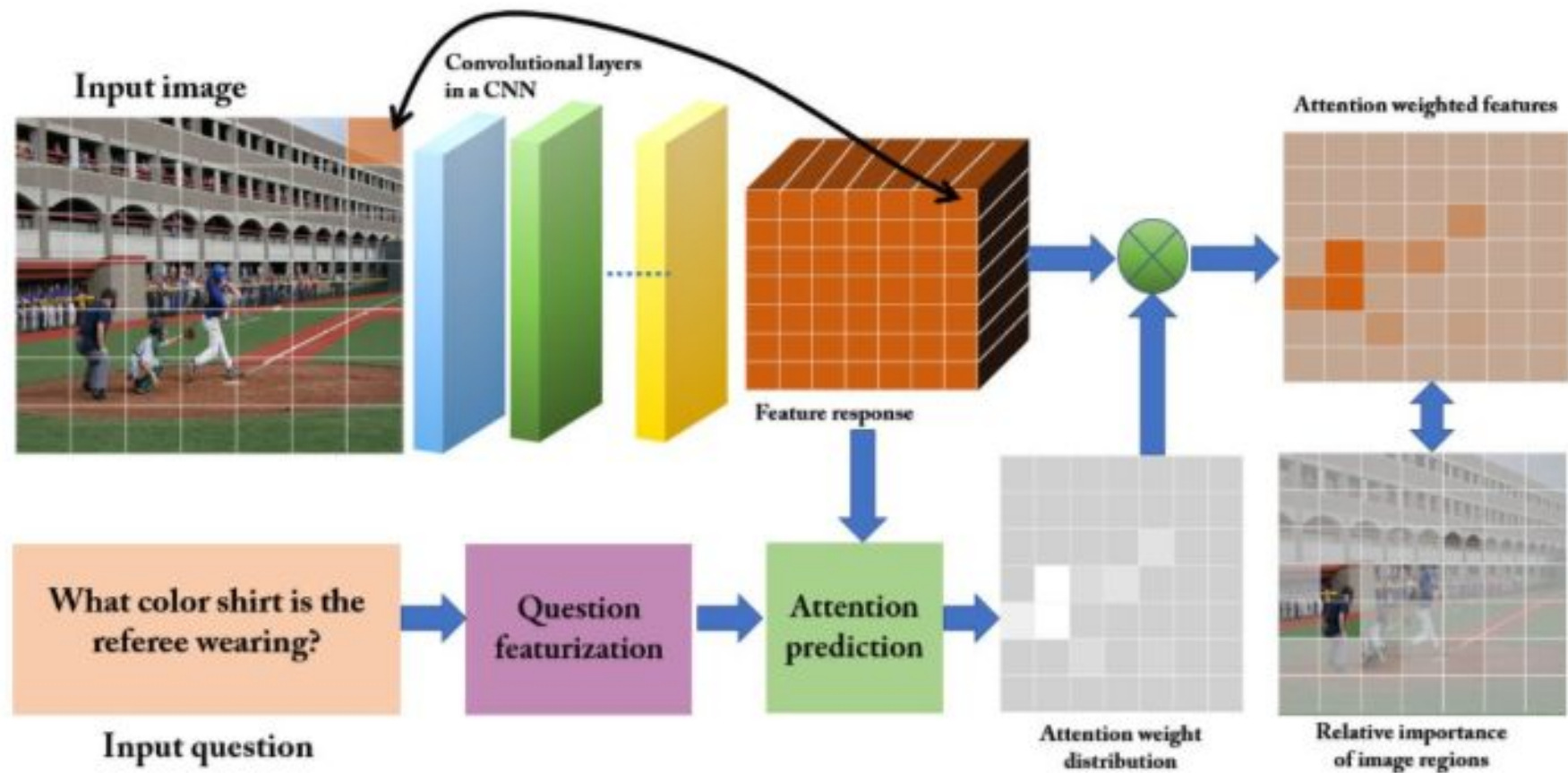
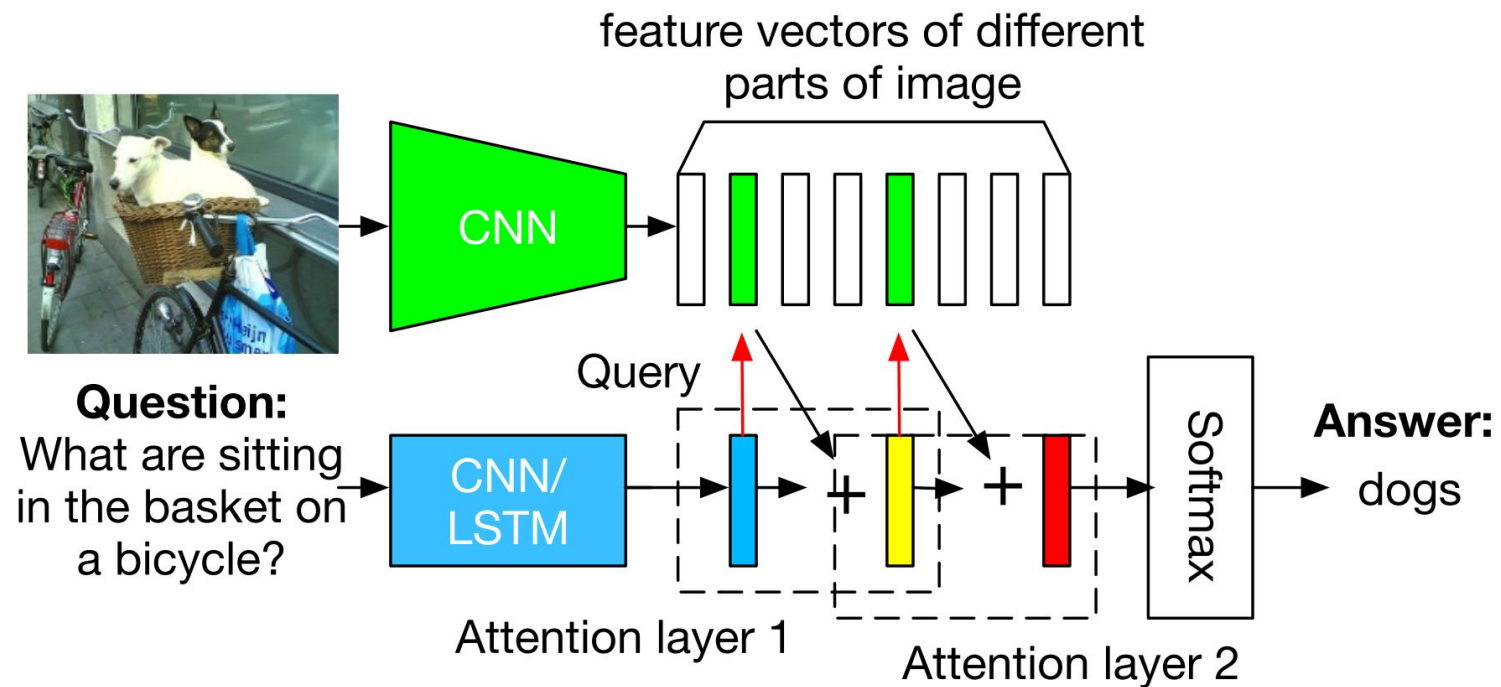


Image Source:  
<https://arxiv.org/abs/1610.0146>

## Multimodal Inputs

- By modality we just mean a medium of input e.g., speech, text, images.
- We most often deal with a single mode of information, but can we make better models using multiple uncorrelated representations of data?
- Think of it like this, we are doing an action recognition task and we have images and audio. We don't have to use both of them to recognize a scene e.g., a traffic jam, we can do just fine by using image classification. But by adding an audio input (honking, traffic noises, etc), we can very easily distinguish between a traffic signal and a parking lot.
- Attention allows us to add complementary features from one representation to another which in the end helps us develop better models.

# TYPICAL STRUCTURE OF ATTENTION NETWORKS





## Overview:

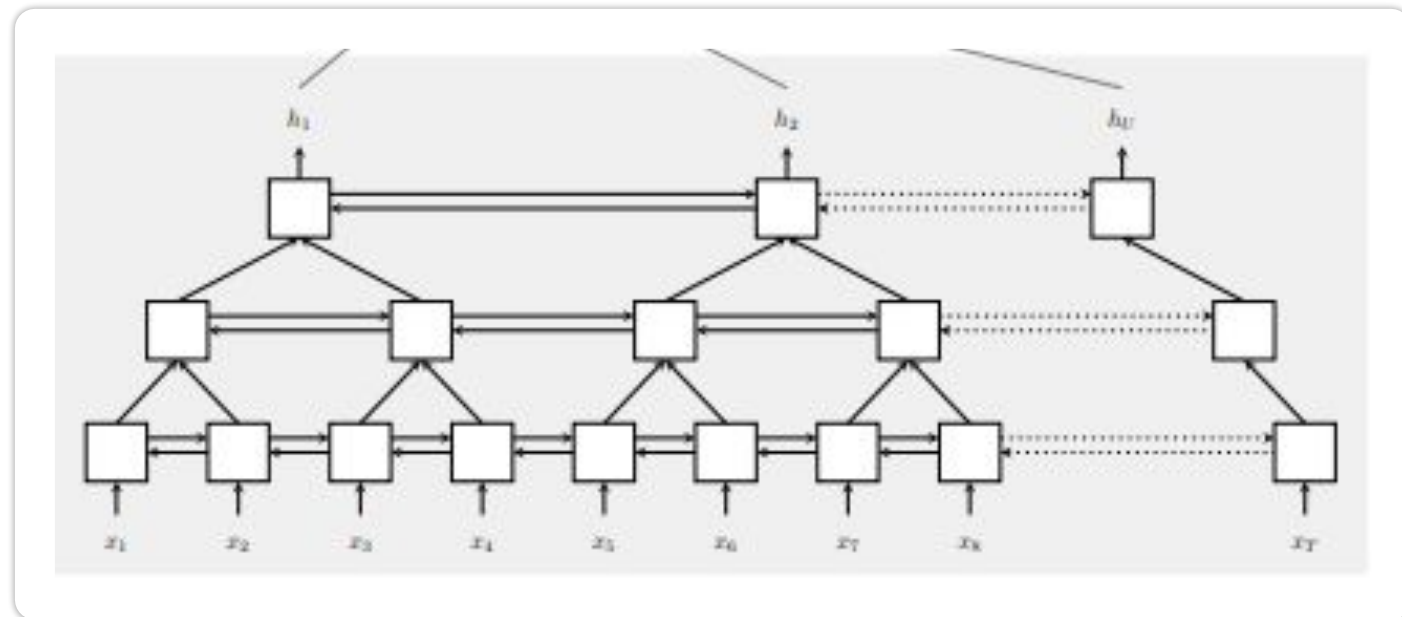
- Encoder – Decoder recurrent neural network architectures are the core technology inside Google's translate service.
- Attention Networks that we will be dealing with in Homework 4 will have 2 parts

1) Encoder

II) Decoder

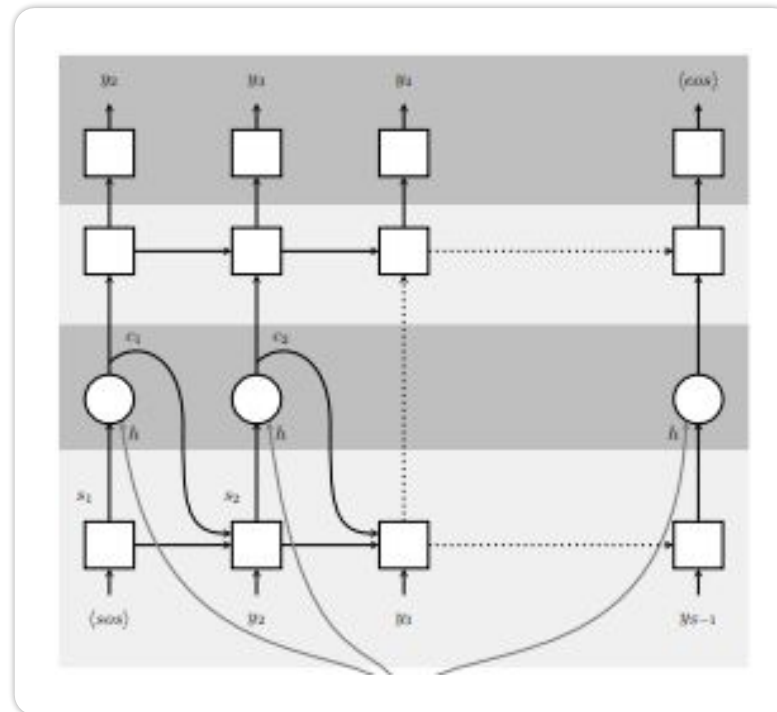
## Encoder:

- Typically used to generate high-level representations of given input data.
- There are no labels used to train encoders
- Are trained jointly with decoders.
- Can be any network, CNN, RNN or Linear.



## Decoder:

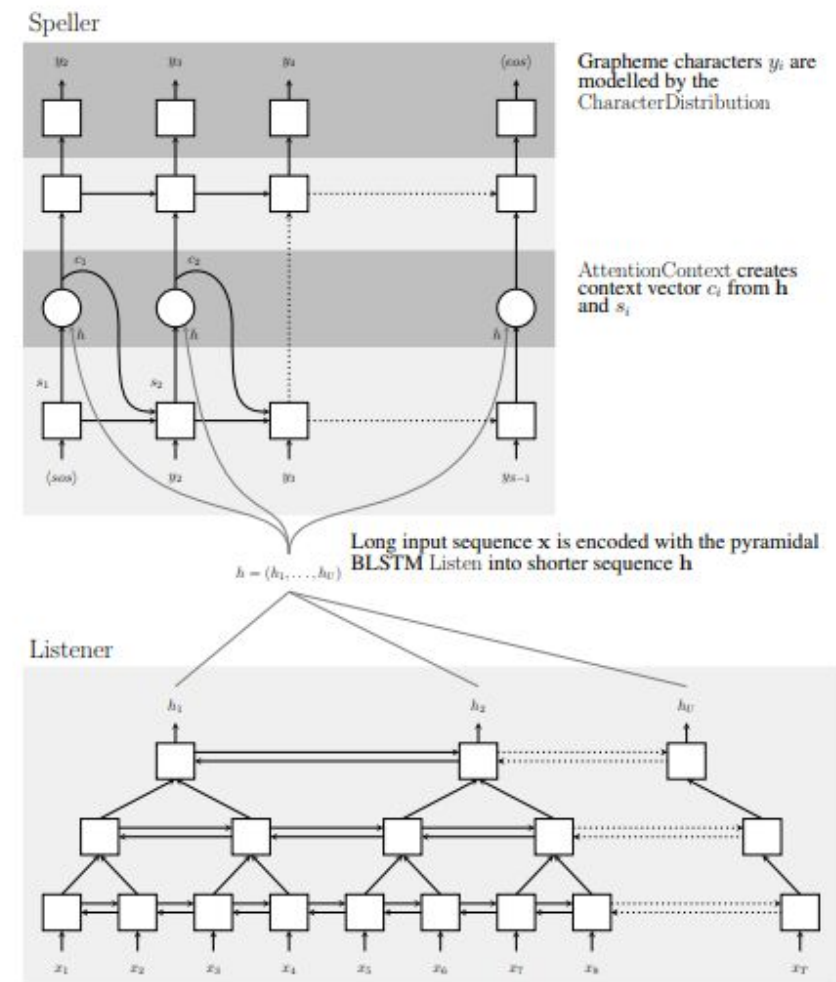
- It is a network that takes in the feature representation from the Encoder and tries to generate the closest match to the expected output.
- Loss function is applied on the output of the Decoder.
- Can also be trained without encoders, encoders are basically to amplify the results of the decoder.
- Most typically an RNN, but few exceptions like FasterRCNN (it is also an attention network, just not a typical case).





# LISTEN, ATTEND AND SPELL

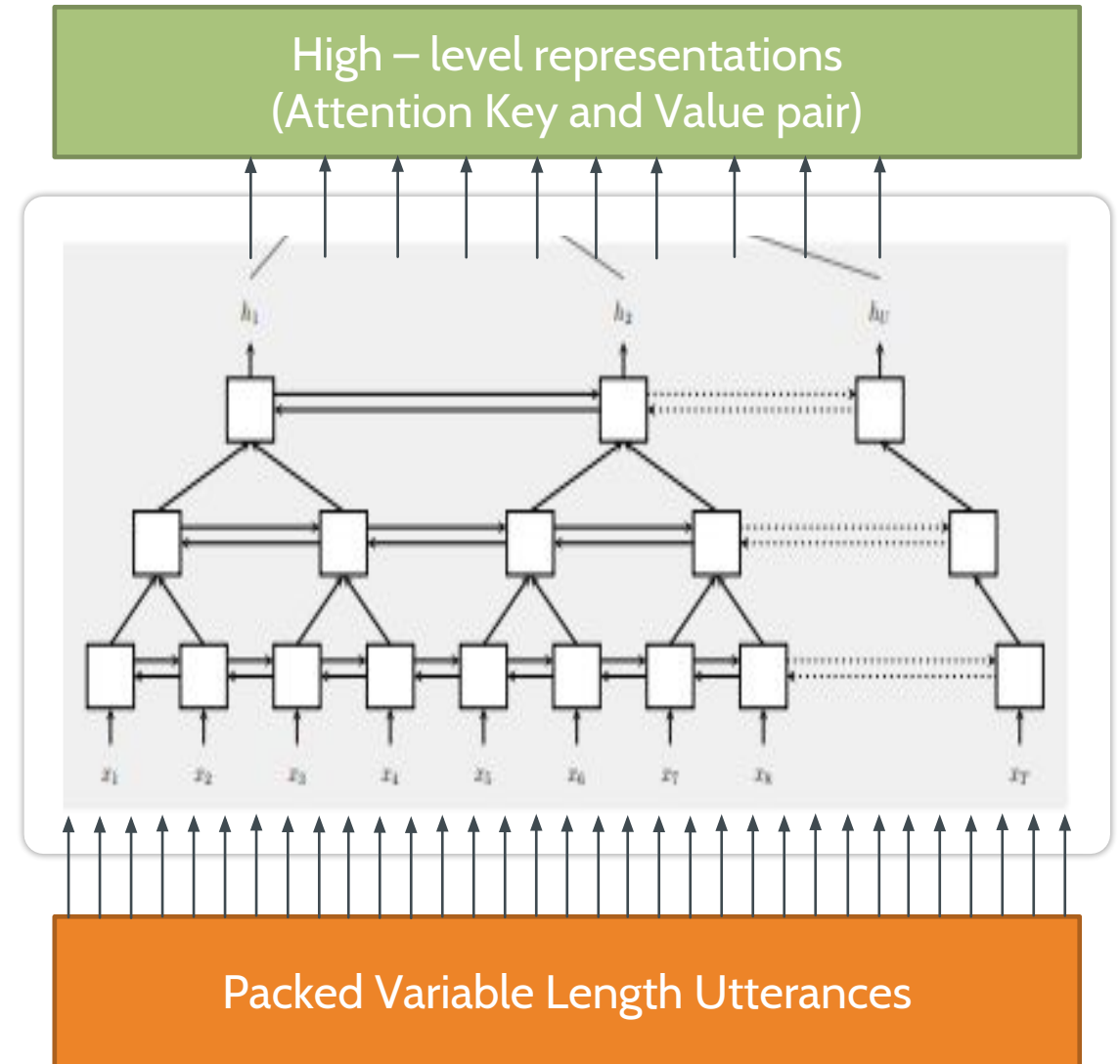
- You will be implementing this baseline model or a variant (from scratch) as your 4<sup>th</sup> homework.
- Contains two parts
  - 1) Listener – Encoder
  - 2) Speller – Decoder
- The concept is straight forward when you look at it. You have the Listener that produces high level representation  $h = (h_1, h_2, h_3 \dots h_v)$  from the given utterances.
- This high level representation is fed into the Speller to generate a probability distribution over the next characters.





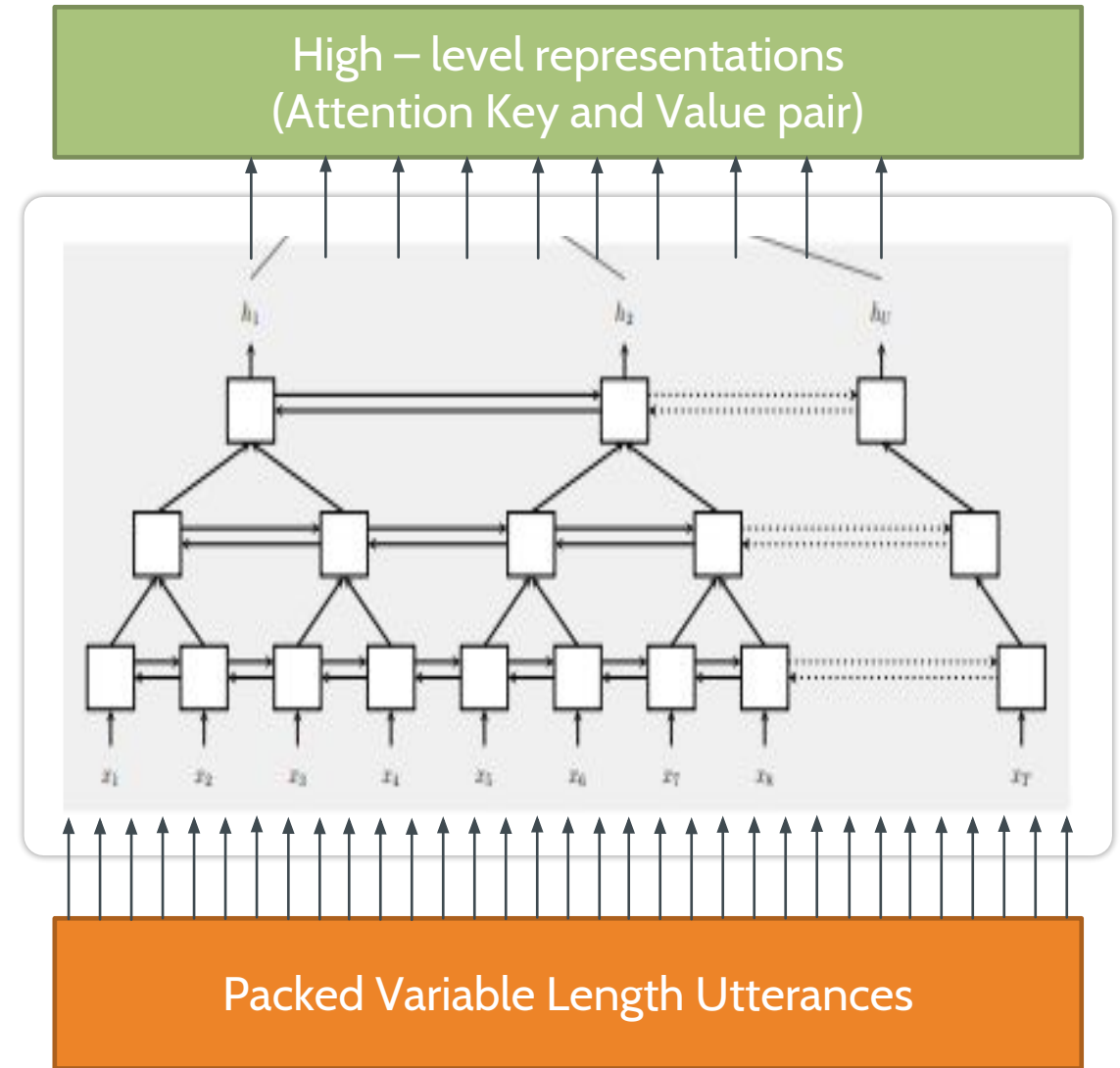
## Listener:

- The input to the Listener is an utterance (or a batch of utterances).
- We would like you to implement a variant of LAS (because it gave us very good results before).
- Instead of having a single output from the pBLSTM as described in the paper, take two separate outputs; one key and one value (you will understand the terminology in a while).
- The key and value will just be a linear projection from the pBLSTM output.
- In the paper, the output of each layer of the pyramid reduces the length of the utterance by 2. They concatenate over the frequency dimension.



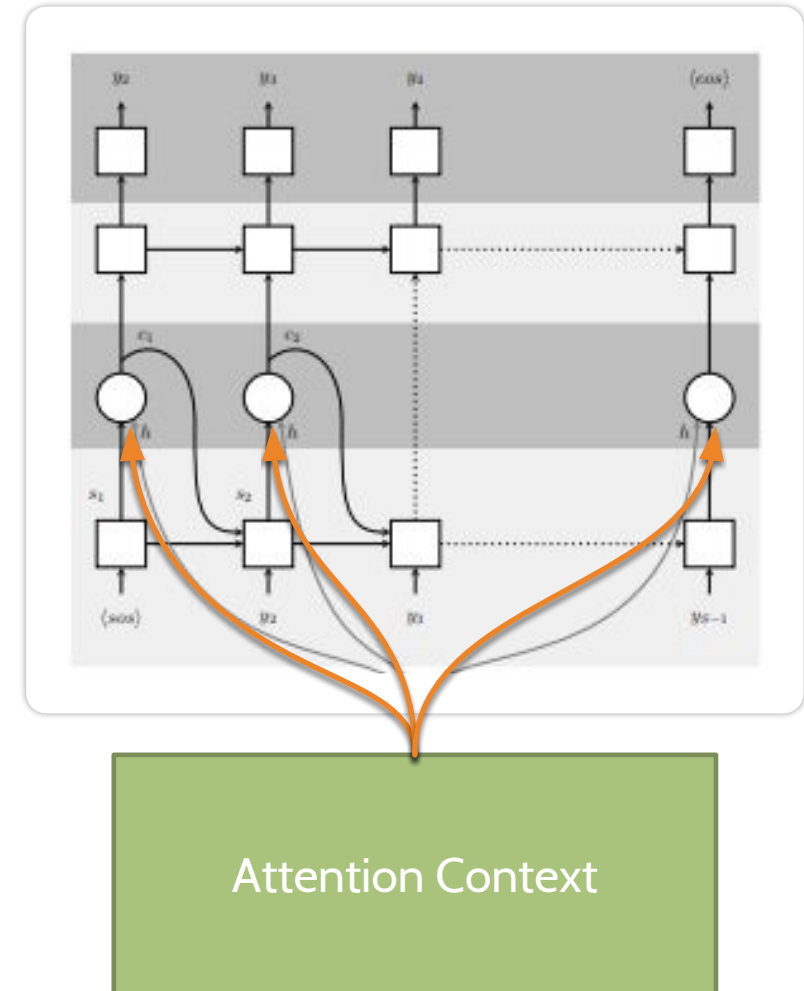
## Problems to look out for:

- The inputs are not all the same length, and this is going to bite you.
- Recitation 7 covered some really good methods on how you can circumvent this (`pack_padded_sequence/`  
`pad_packed_sequence`) and make it someone else's problem to solve. It is highly recommended that you go through recitation 7, you probably might get to sleep a few extra hours if you do it.
- Memory issues, the '`view()`' method in pytorch works only with contiguous memory. So when you try to reduce the dimensionality after every layer of the pyramid, this might throw errors.
- Masking is going to be a big problem. Since you have variable length inputs, you need to ensure that the loss propagates backward only through the values that aren't padding. Also, make sure you don't screw up masking when you half the utterance lengths.
- Overall, the network should ideally produce a key, value and a mask.



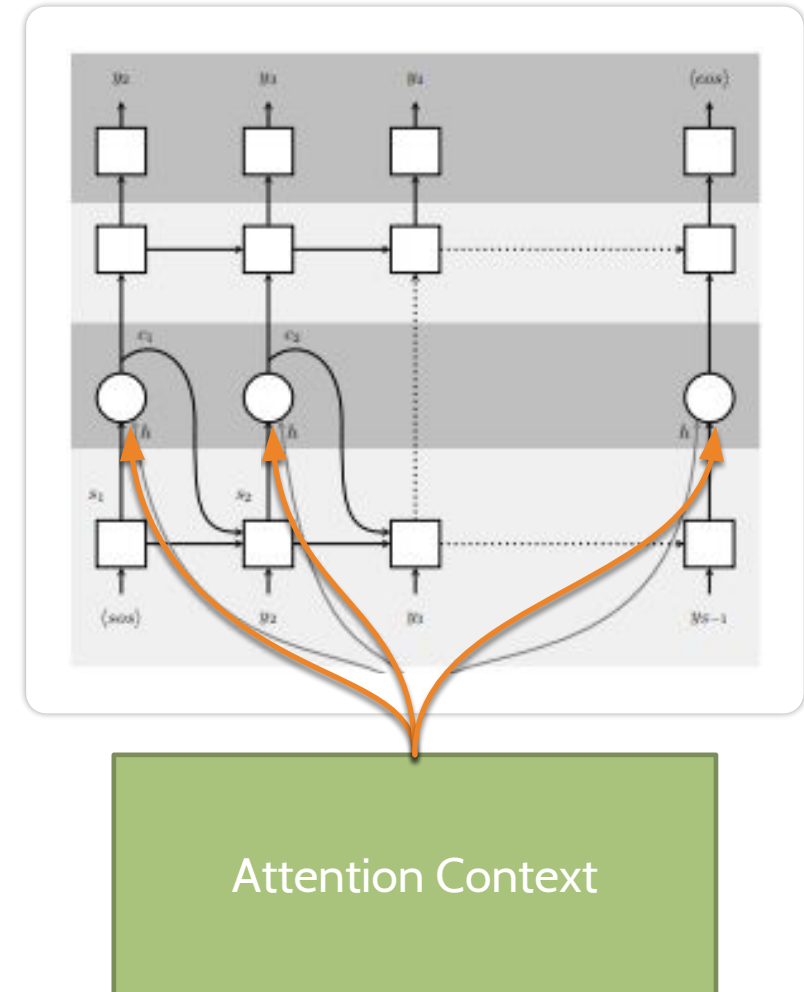
## Decoder:

- This is going to be a bit tricky. All this while your LSTM took care of looping through the time steps to give you a nice probability distribution for every input step. From here on, you do the looping yourself!
- Now, to add attention context you have to make some changes in the internal states of the LSTM cell at every time step, so you cannot use the old LSTM class.
- **LSTMCell** is another class that gives you more flexibility in controlling what flows in and out of each cell.
- So you can directly add/concat/mean-pool any value to the hidden state of the LSTMs.



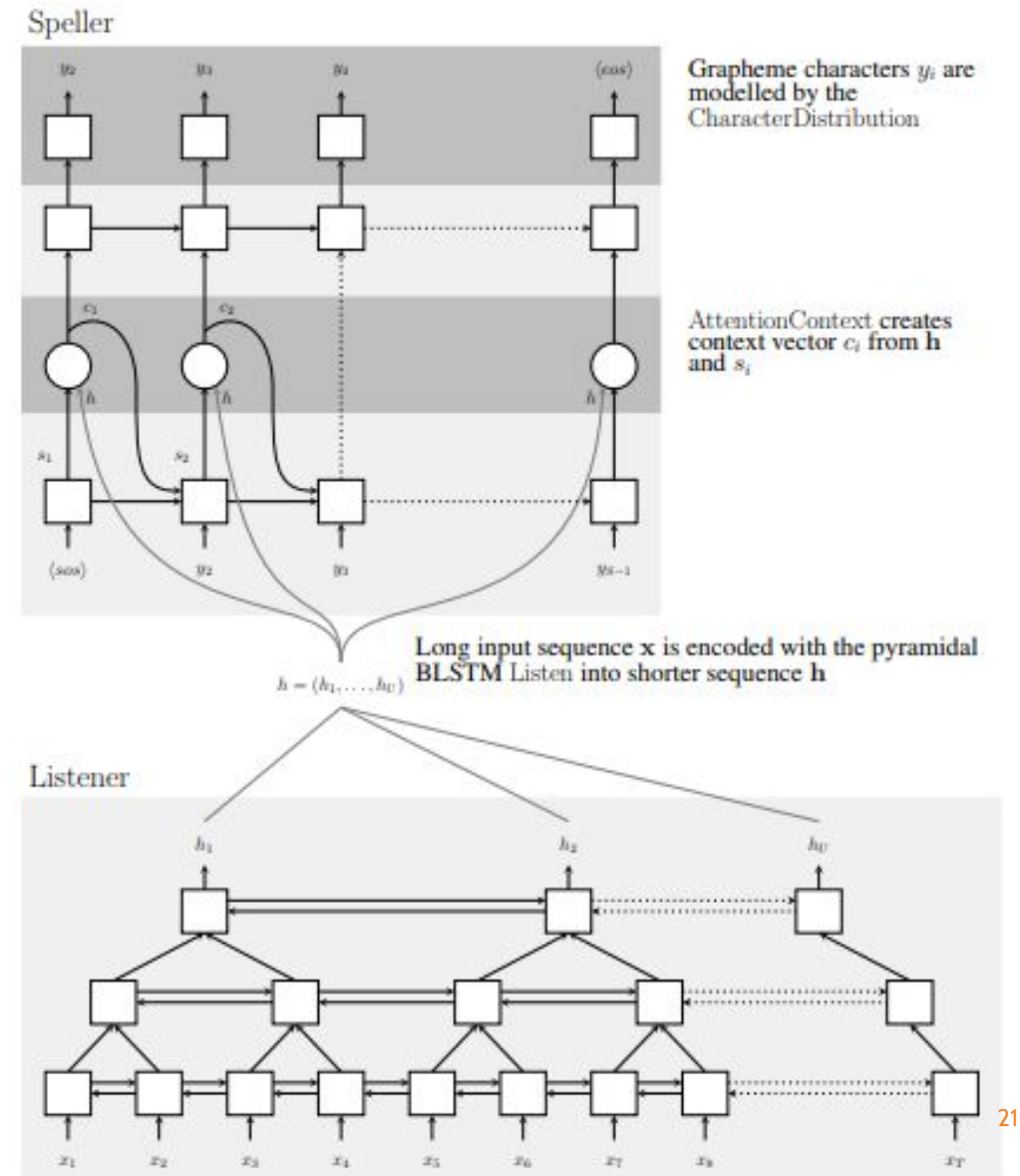
## Decoder:

- You will be manually looping over each timestep of the input sequence.
- At each timestep, you will be computing the attention context.
- The output (hidden layer) of the first LSTMCell of the decoder is called the query, the projections from the encoder are keys and values.
- Attention context can be computed as follows:  
$$\text{energy} = \text{bmm}(\text{key}, \text{query})$$
$$\text{attention} = \text{softmax}(\text{energy})$$
$$\text{context} = \text{bmm}(\text{attention}, \text{value})$$
- The LAS model incorporates this context by giving it as an input to the first LSTM.
- Just concatenate the context to the embedding, so the LSTM input dimension is (embedding\_dim + context\_dim)



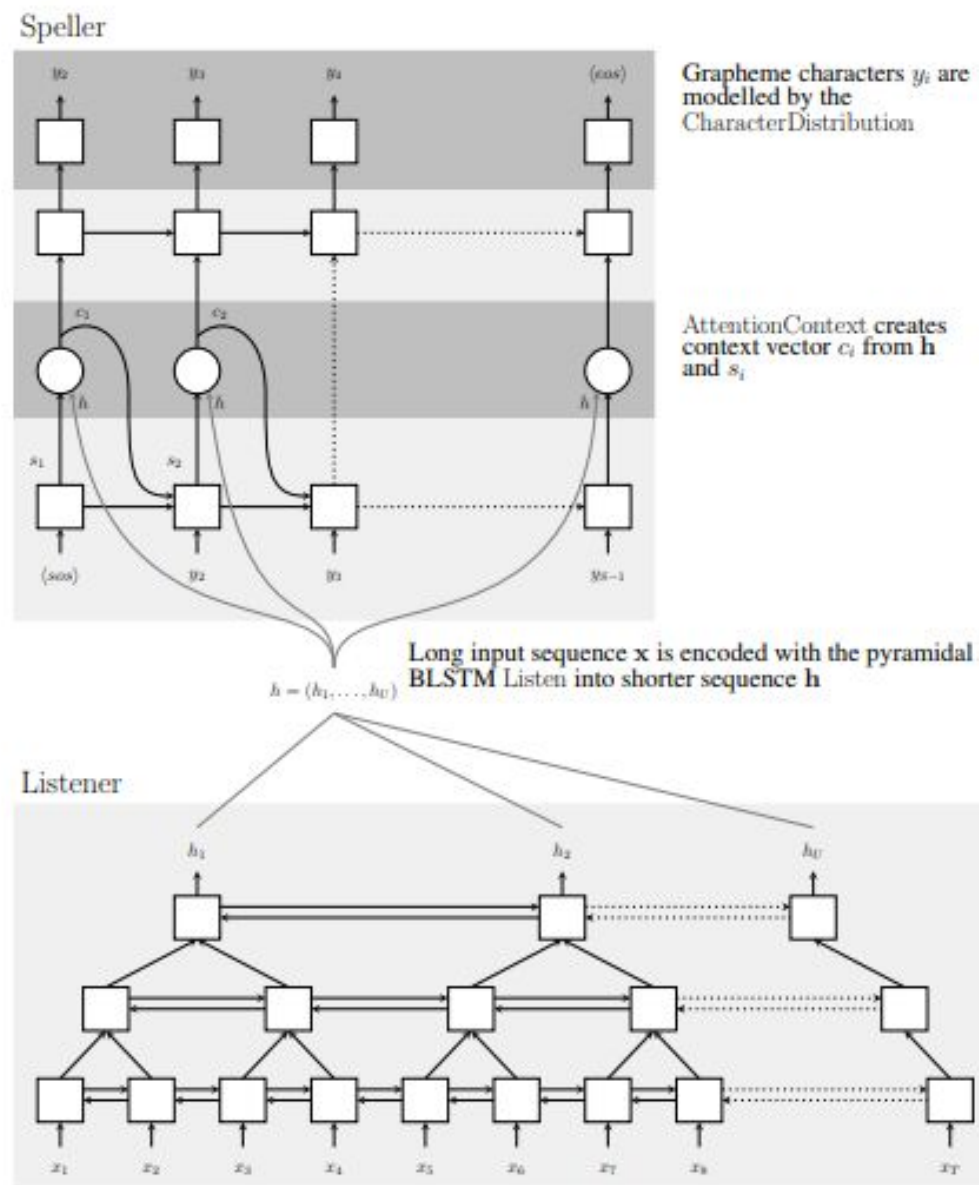
## Joint Training:

- Once we get this far, things should hopefully become straightforward.
- You should be able to do a complete forward and backward propagation on the network.
- Reiterating from earlier, Listener is an encoder that only amplifies the results. The network will even learn if there is not Listener (essentially, only the language model will be left).
- Debugging errors in connections between encoder and decoder are nasty. If you have a complicated code structure, even the TAs will not be of much help (from personal experience..☹)



## Problems to lookout for:

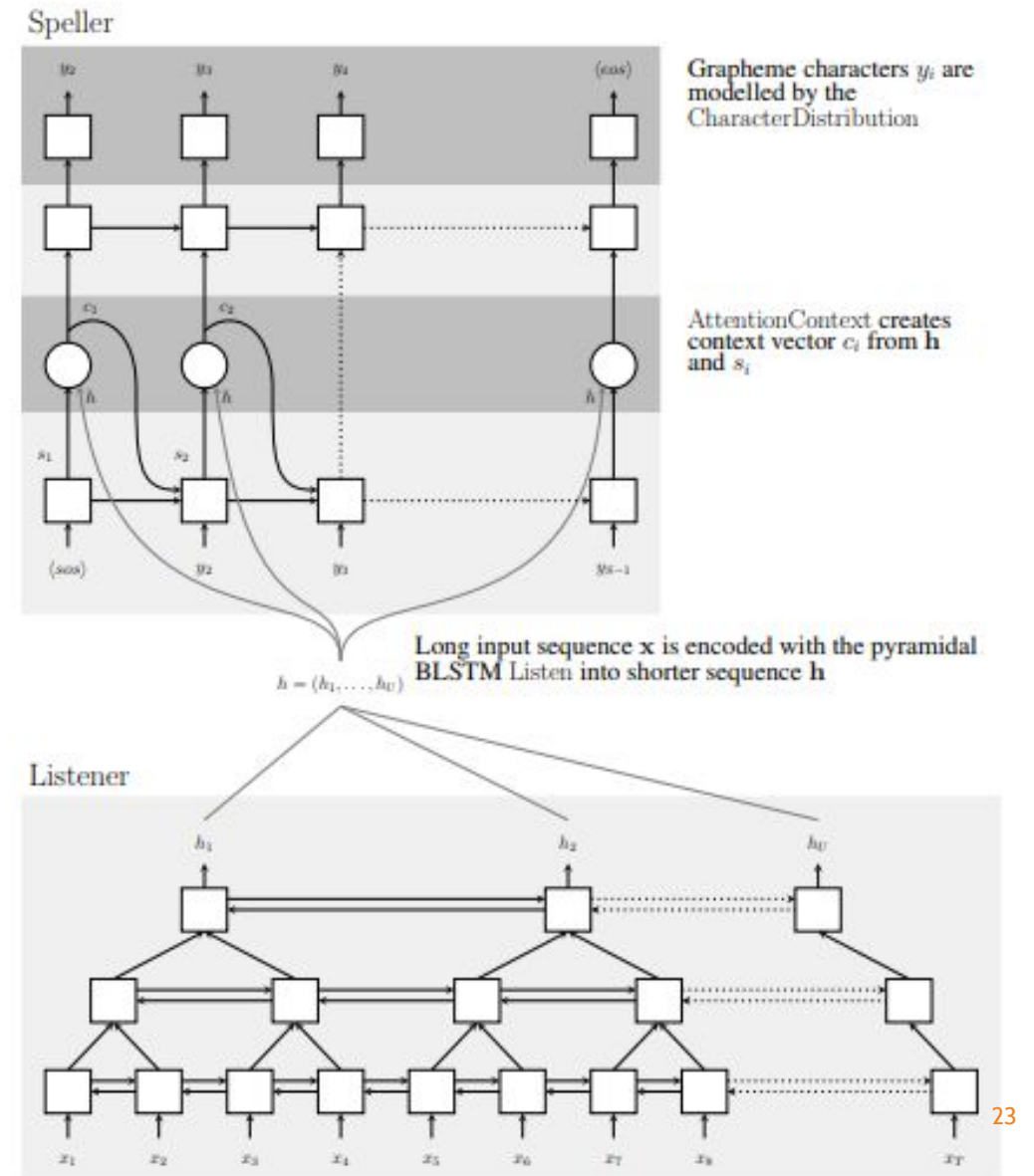
- Masking. This can be a huge problem. There is a lot going around, lot of squeezing, reshaping, packing, unpacking, concatenating, and other operations. One can very easily lose track of masks. If this happens, everything falls apart.
- You basically need to compute loss over only those values that are not padding. So if your mask is off, it might seriously affect your results.
- Network can get stuck in loops very easily, you should be looking at implementing teacher forcing to avoid that.





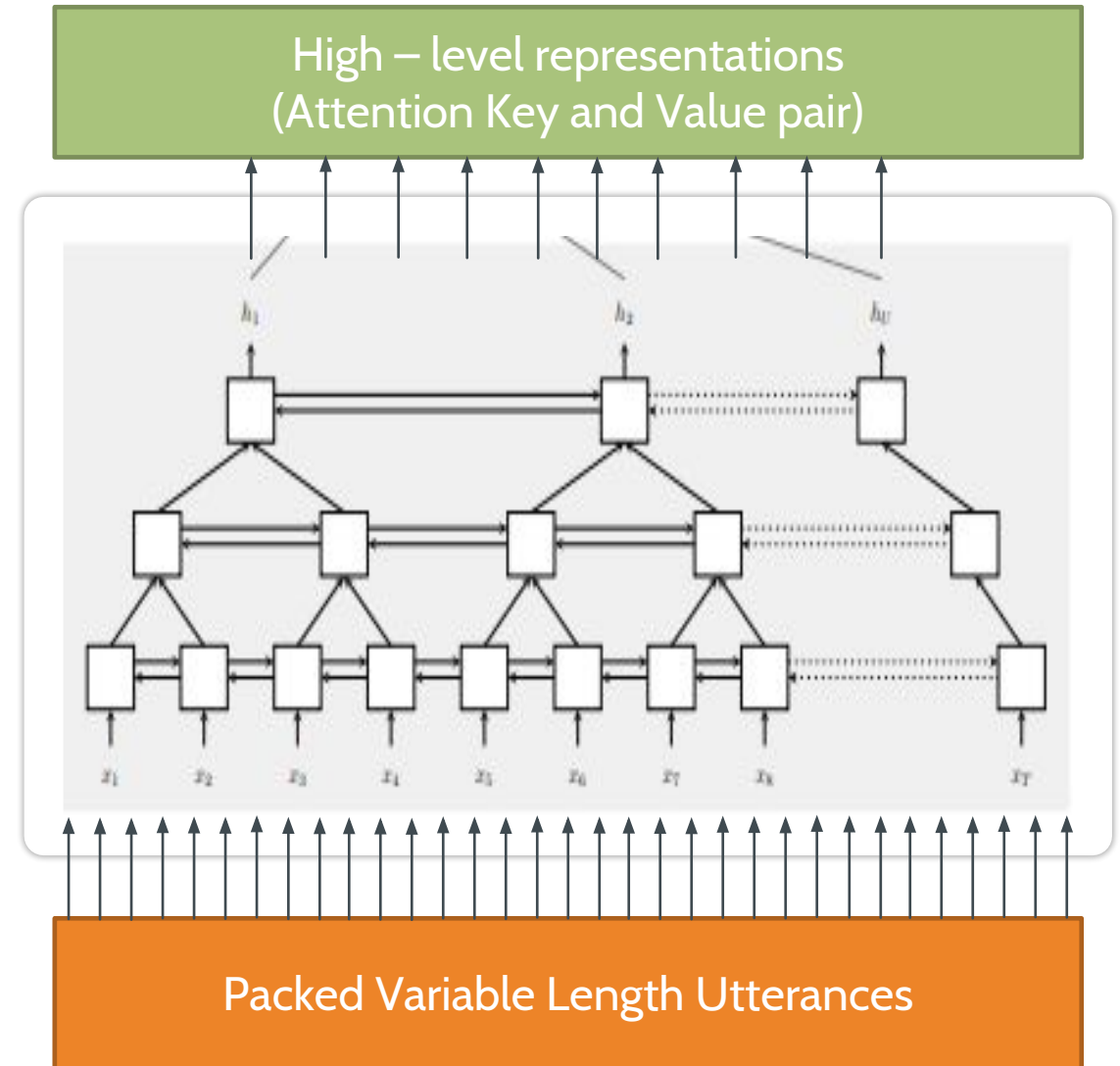
## Some Tricks:

- **Pretraining:** Since we have two separate networks here and the encoder network works well only if the decoder network works well. We can pretrain the decoder network (just as we trained a language model in hw3). It isn't going to be that straight forward though, because while pretraining you have no context to add. So, you might have to copy back weights layer by layer than using `torch.load()`.
- **Teacher Forcing:** This avoids letting your network getting stuck in generating loops and errors. Implementation wise it is just passing in the output of your network as the next input while training instead of sending the ground truth. This allows the network to learn to recover from errors.
- **Random search:** Almost as good as beam search and also simpler to implement.



## Attention Key, Value Computation:

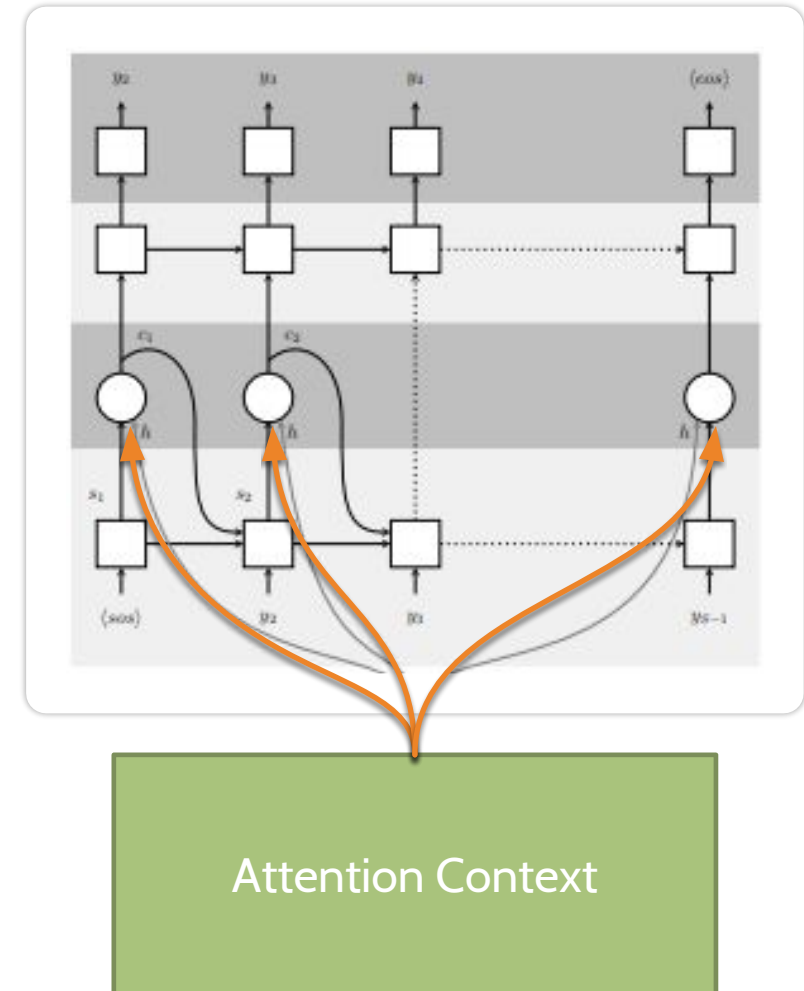
- This is simple.
- Take the output of the 3<sup>rd</sup> BiLSTM layer  $h$  and project it into a lower dimensional space.
- We suggest you keep the dimension size low here, you wouldn't need to go beyond 256.
- So you would need two linear layers parallelly on top of the output of the p-BLSTM network. Each linear layer gives you a projection, you call one a key and the other a value.
- The idea is that you teach the weights of one of these linear layers to act as a key to find the attention mask. With that attention mask, you obtain the context without losing any information.





## Attention Implementation:

- Reiterating previously mentioned in terms of tensor shapes.
- 1) Decoder produces query of shape  $(N, A)$
  - 2) Perform element-wise mult. for energy:  
 $\text{bmm}(\text{query}(N, 1, A), \text{keys}(N, A, L)) = (N, 1, L)$   
do this for each sample and each place in the utterance.
  - 3) Softmax over energy (along the utterance length dimension) to create the attention over the utterance. (think of why we are doing softmax?)
  - 4) Perform:  
 $\text{bmm}(\text{attention}(N, 1, L), \text{values}(N, 1, B))$   
to produce the context.
  - 5) Concatenate this context to the embedding and pass it into the next LSTMCell as input.



### Word of Advice:

- This homework is going to eat up a lot of your time. The base paper is already given to you, so there is no question of releasing another baseline sometime down the line.
- If you wait until the last week to start working on this assignment, you may not be able to pull it off.
- You might need to do a lot of training. So don't try to train on the entire training data at once, work with dev set and jump to train set only after everything looks good.
- The cutoffs are going to be very tough to beat. We are giving more than a month to work on this, so there will not be any scope to complain about having no time to tune the network.
- Make sure your code structure is simple to read and comprehend (follow good naming conventions and the 5x thumb rule to make a function). If you ever get stuck, it'll be very difficult for us to see where you went wrong if the code is not structured well.



That's all for today!

Good Luck 😊