Topics for Today

Pre-requisite Recape

1. Language Modeling
2. Attention

HW4P2 Overview

Intro to LAS

- Listener
- Speller
- Attention

Implementation High-Level Suggestion

- Pooling
- LSTMCell
- Masking
- Sampling
- Teacher Forcing
- Inference
- Debugging

Language Modeling

Wiki: “*A statistical language model is a probability distribution over sequences of words.*”

Language Modeling

The usage of Language Model is to assign a likelihood to a sentence.

- Intuition:

P(*high winds tonight*) > P(*large winds tonight*)
P(*I saw a cat*) > P(*cat saw cat I*)

- Formulations:

Probability of the next word

$$P(w_5 \mid w_4, w_3, w_2, w_1)$$

Probability of a sentence (or a sequence of words)

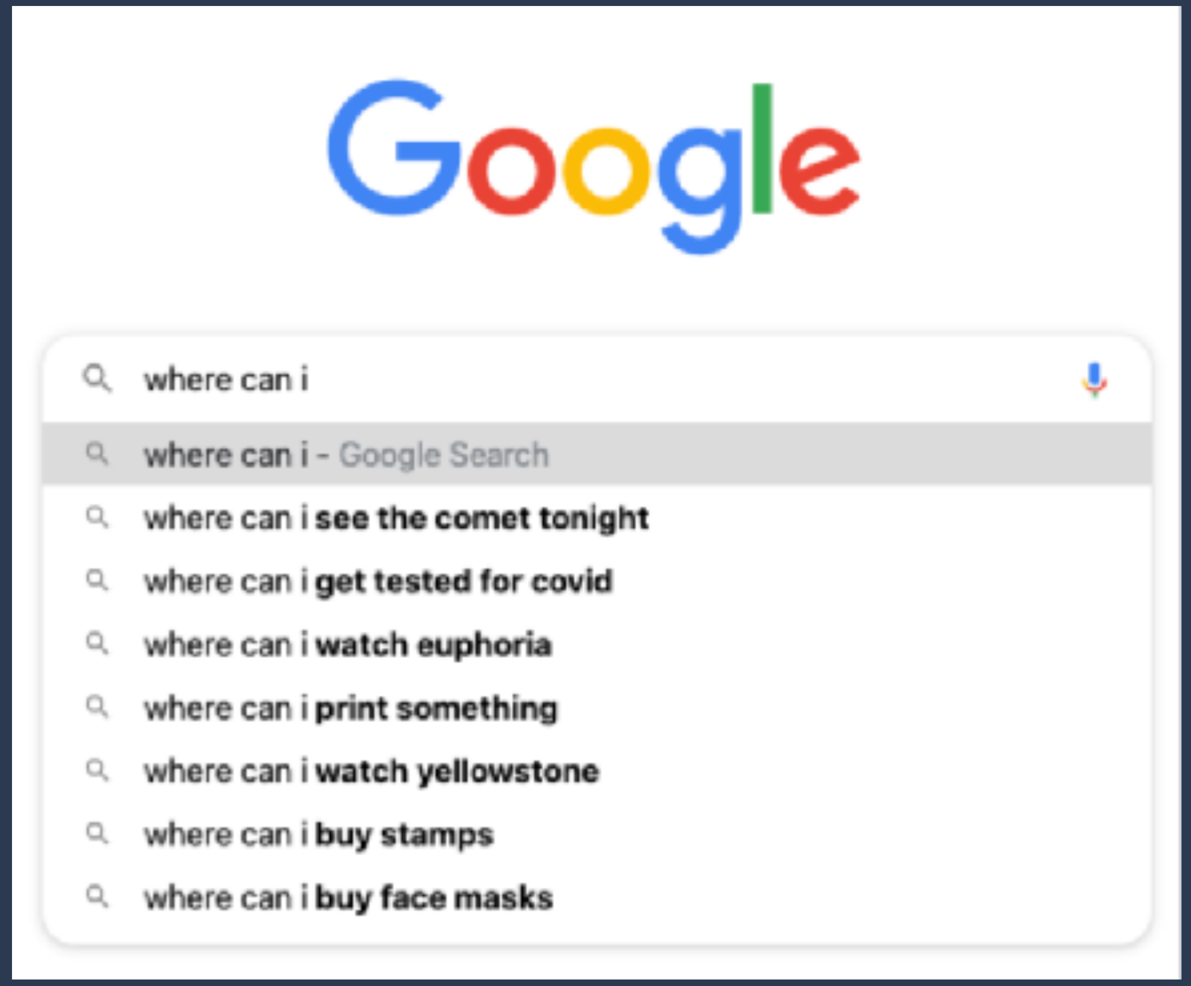
$$P(w) = P(w_1, w_2, \ldots, w_n)$$

$$P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2, x_1)\ldots P(x_n \mid x_{n-1}, \ldots, x_1)$$

- Simple Assumptions

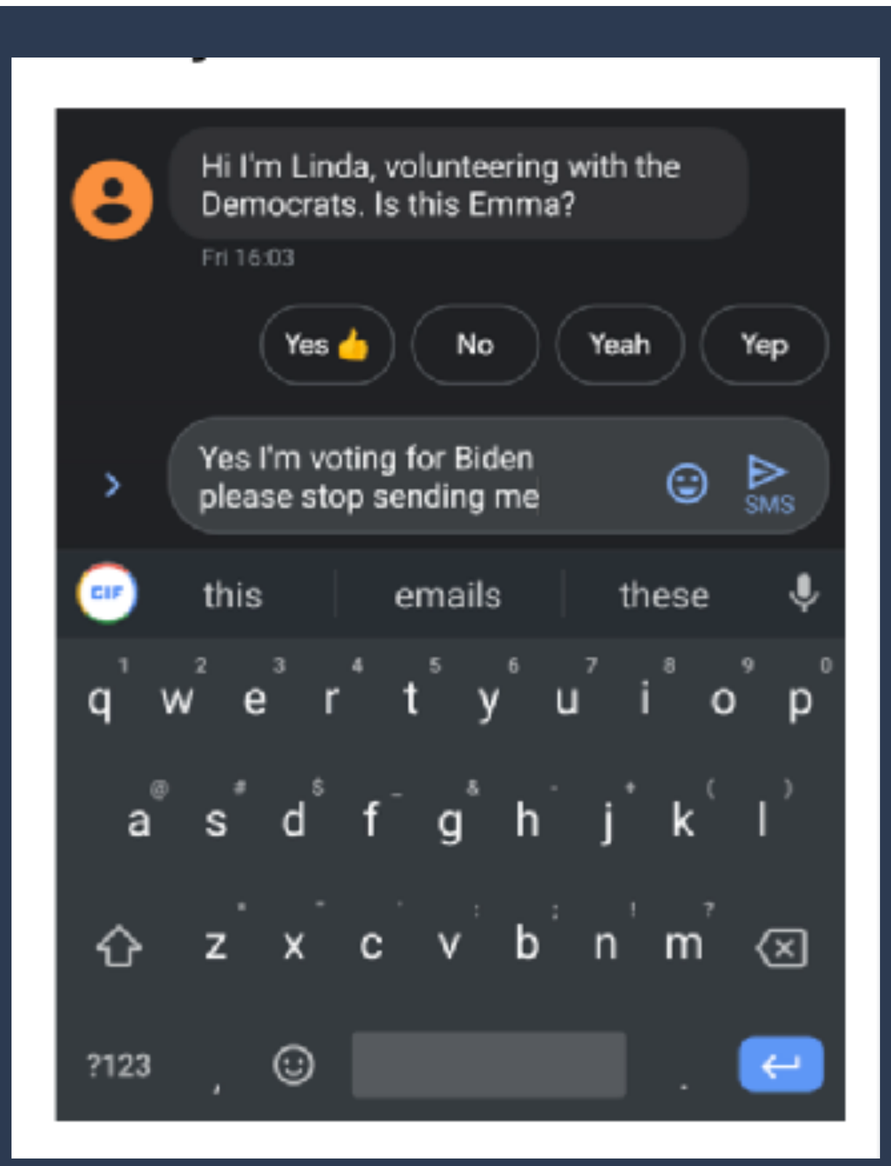
1. Unigram Model $P(x_t \mid x_{t-1}, \ldots, x_1) = P(x_t \mid x_{t-1})$

2. Bigram Model $P(x_t \mid x_{t-1}, \ldots, x_1) = P(x_t \mid x_{t-1}, x_{t-2})$



where can i

where can i - Google Search
where can i **see the comet tonight**
where can i **get tested for covid**
where can i **watch euphoria**
where can i **print something**
where can i **watch yellowstone**
where can i **buy stamps**
where can i **buy face masks**



Hi I'm Linda, volunteering with the Democrats. Is this Emma?
Fri 16:03

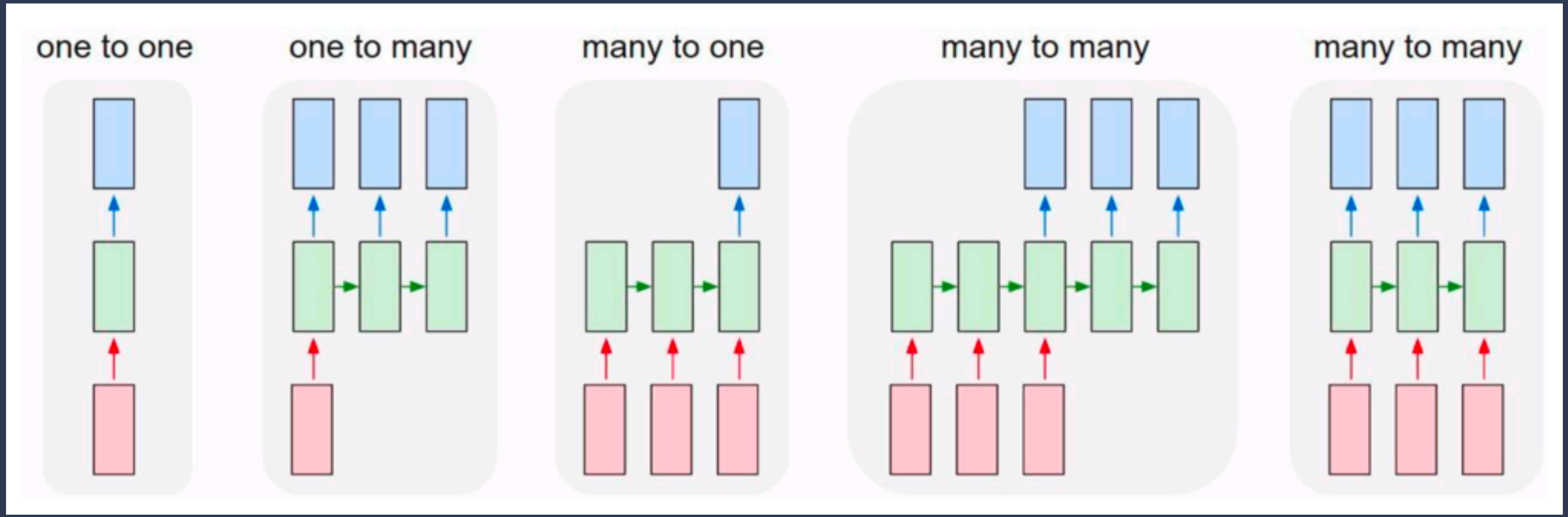
Yes 👍 | No | Yeah | Yep

Yes I'm voting for Biden please stop sending me

this | emails | these

Language Modeling in RNN

How to use RNN for language modeling?



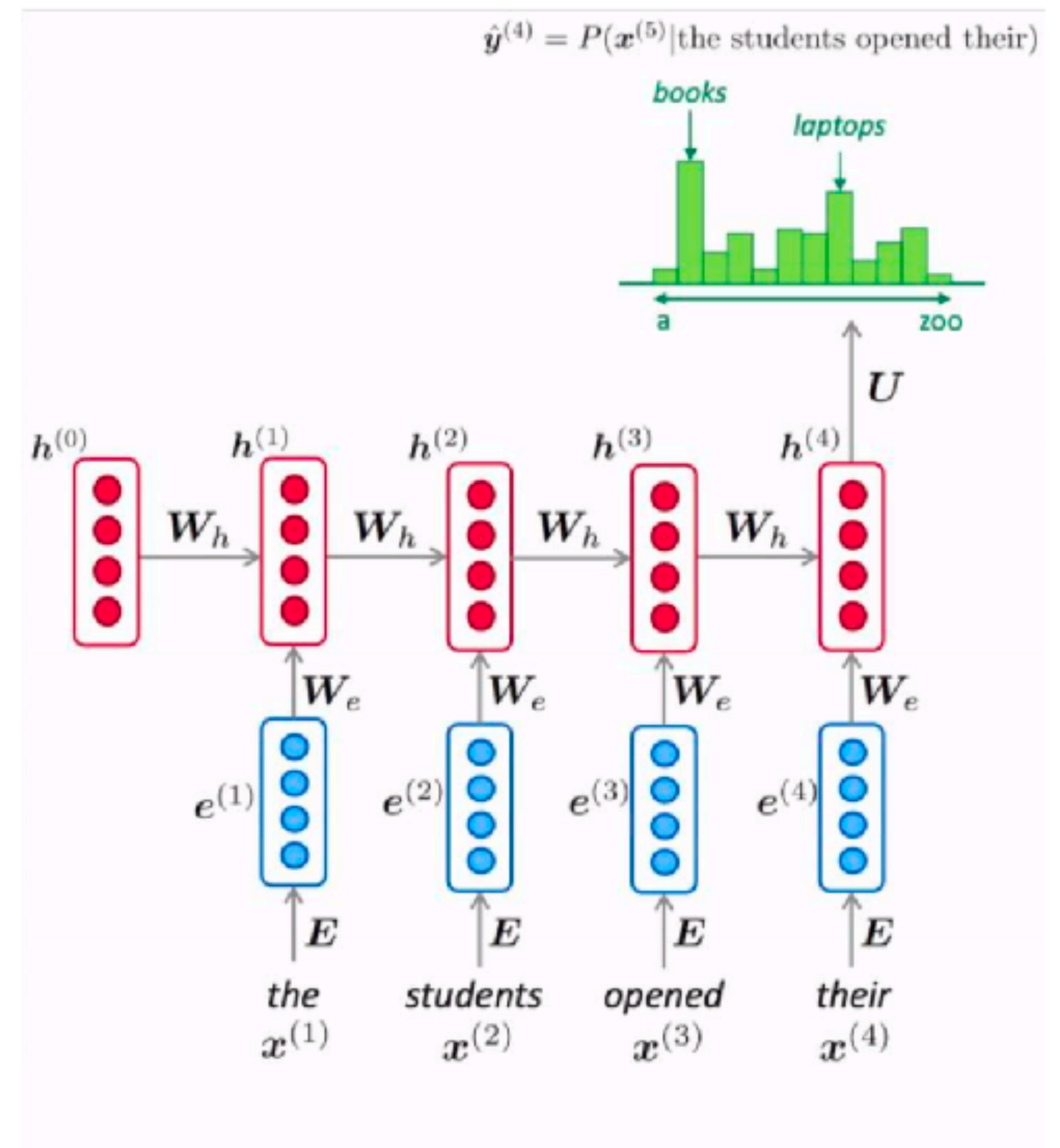


Figure 5: An RNN Language Model

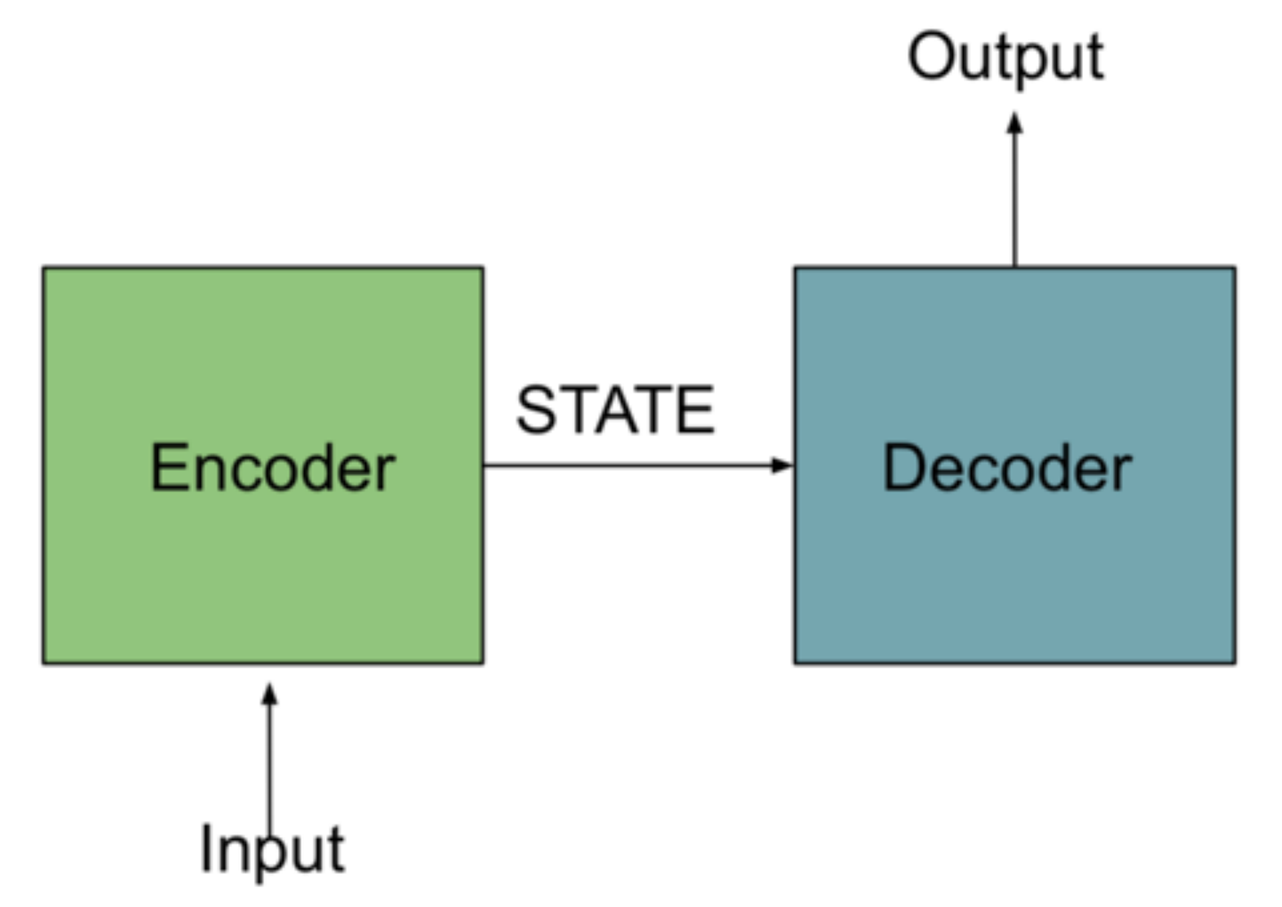
For more details of LM, please refer to last semester's recitation

https://deeplearning.cs.cmu.edu/S20/document/recitation/recitation-7.pdf

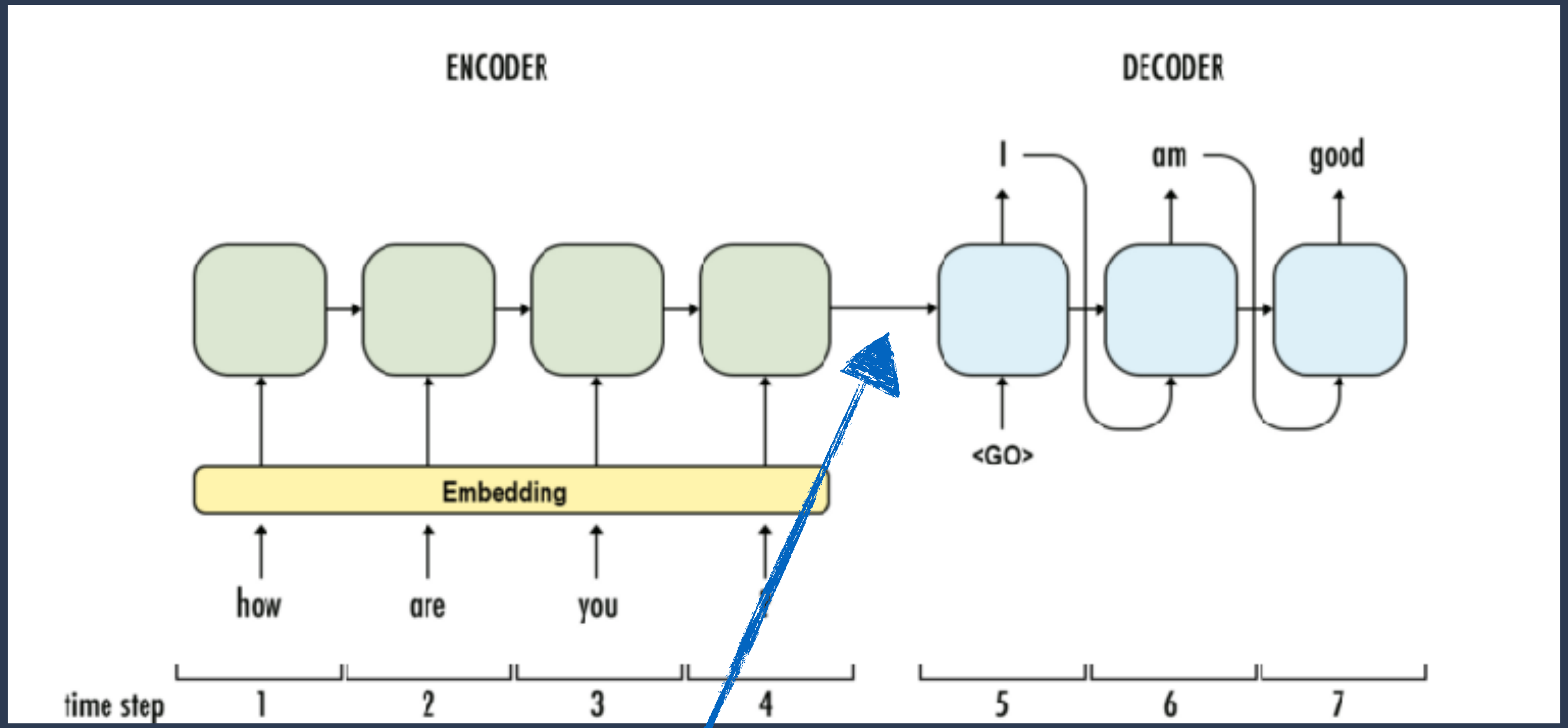
https://mediaservices.cmu.edu/media/Introduction+to+Deep+Learning+Recitation+7/1_irsncxxg/148590631

Attention Recap From Last Recitation

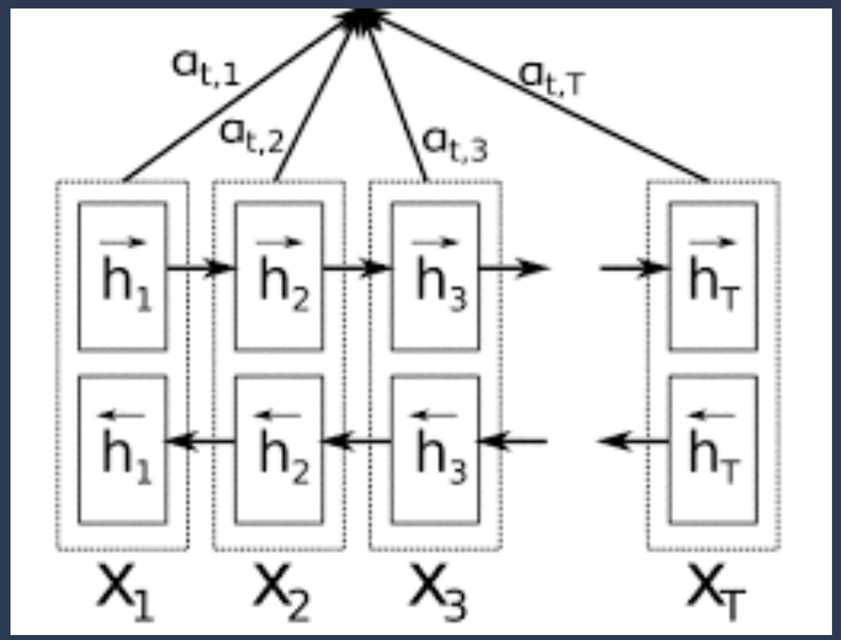
- Motivations



A typical generative architecture

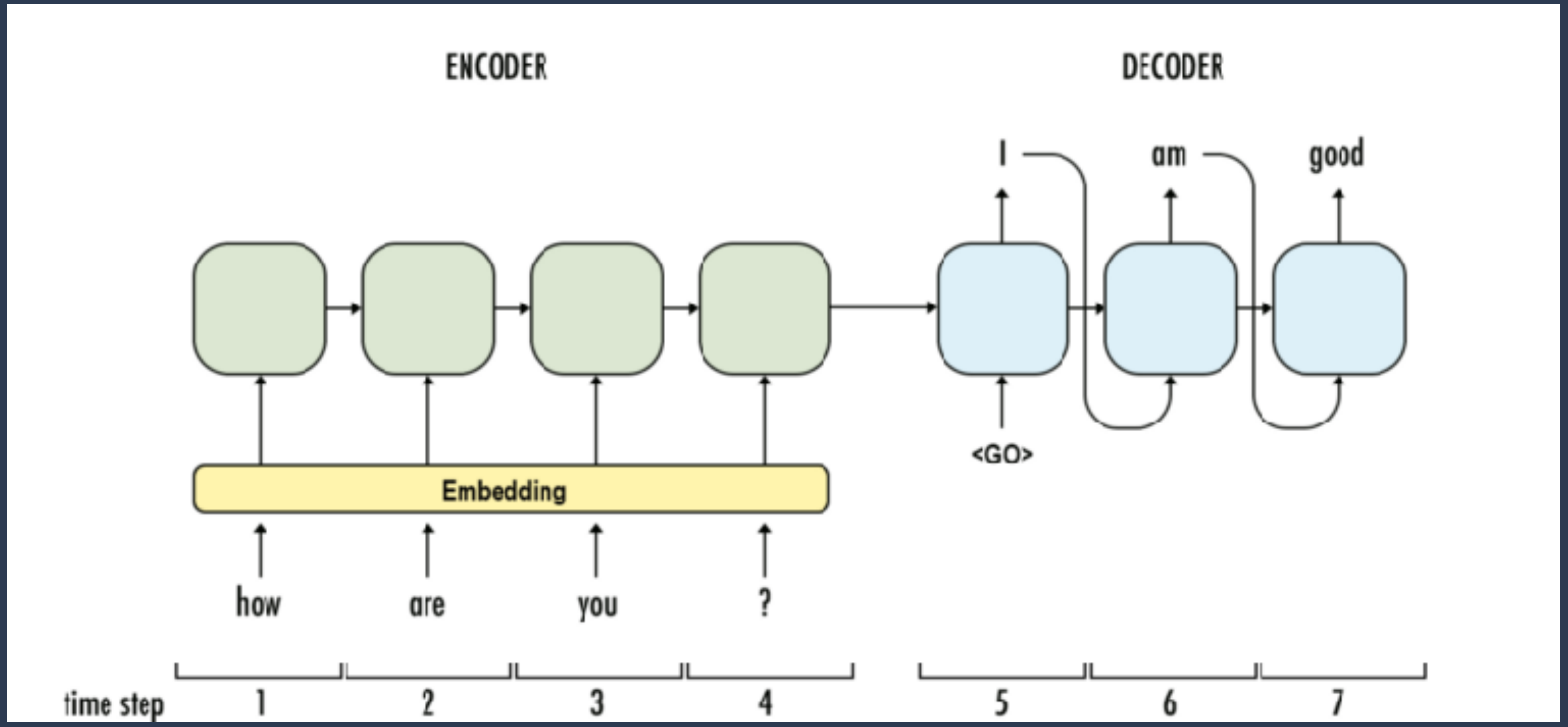


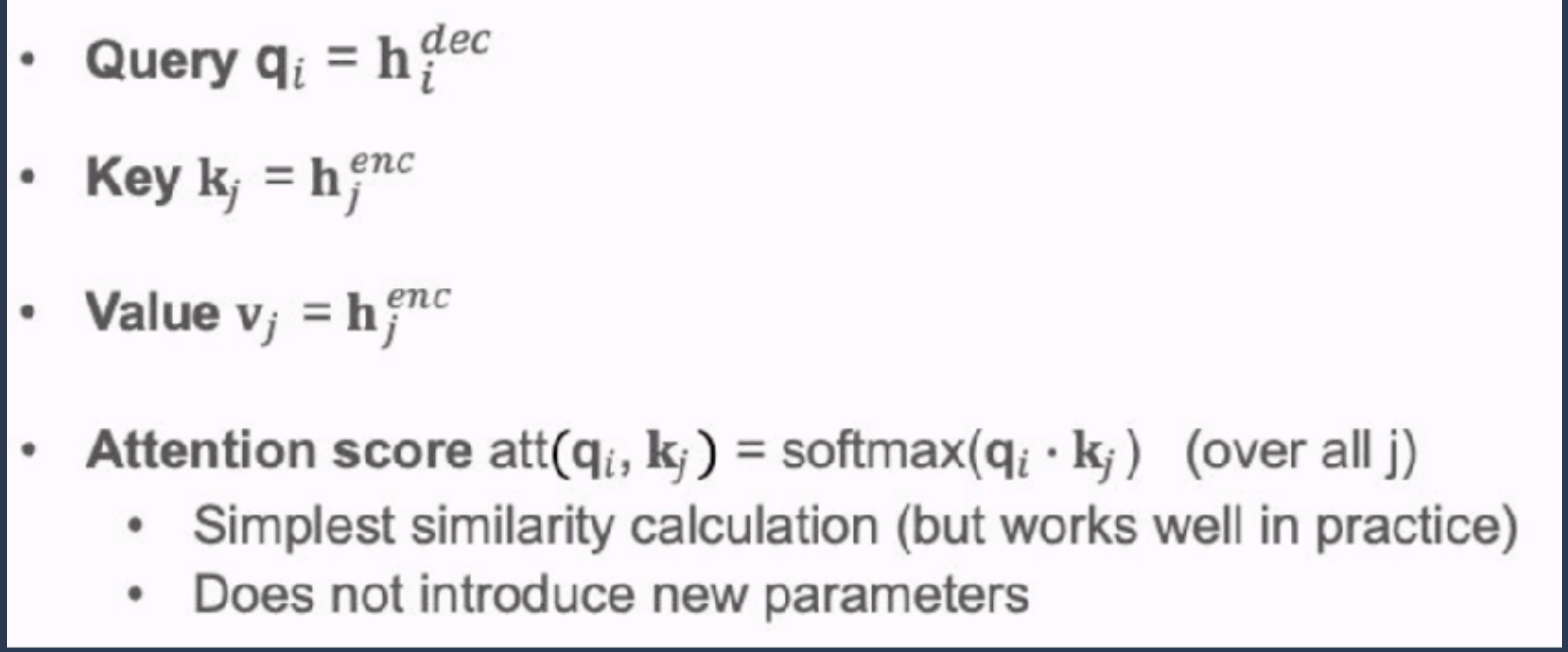
Problem? The hidden state is too condensed and not informative
Solution? Add Context using Attention.



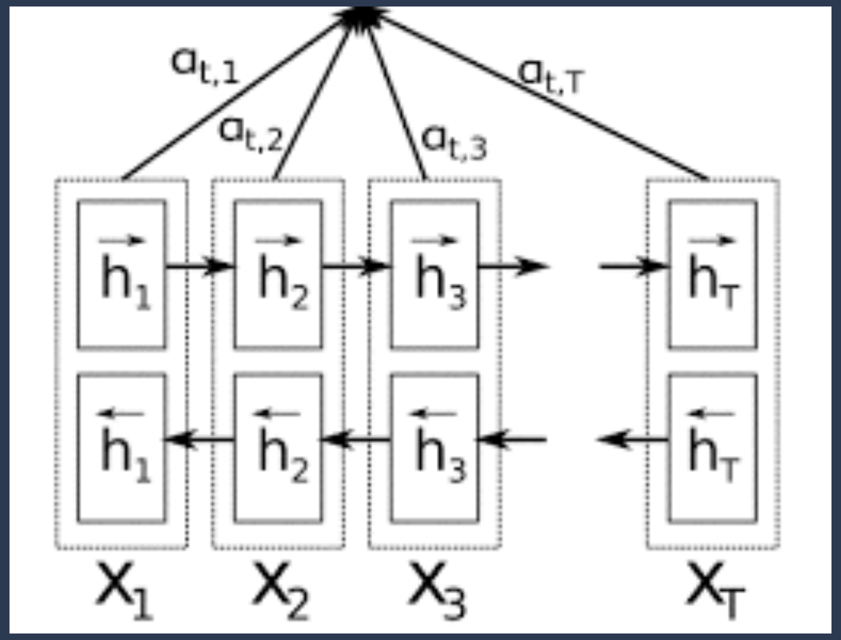
Attention Recap From Last Recitation

- Approach (Dot Product Attention)





- **Query** $\mathbf{q}_i = \mathbf{h}_i^{dec}$
- **Key** $\mathbf{k}_j = \mathbf{h}_j^{enc}$
- **Value** $\mathbf{v}_j = \mathbf{h}_j^{enc}$
- **Attention score** $\text{att}(\mathbf{q}_i, \mathbf{k}_j) = \text{softmax}(\mathbf{q}_i \cdot \mathbf{k}_j)$ (over all j)
  - Simplest similarity calculation (but works well in practice)
  - Does not introduce new parameters



Query Dimension:	$(Batch, Time_{dec}, D_{key})$
Key Dimension:	$(Batch, Time_{enc}, D_{key})$
Value Dimension:	$(Batch, Time_{enc}, D_{value})$

Attention Weights Dimension:	$(Batch, Time_{enc}, Time_{out})$
Context Dimension:	$(Batch, Time_{dec}, D_{value})$

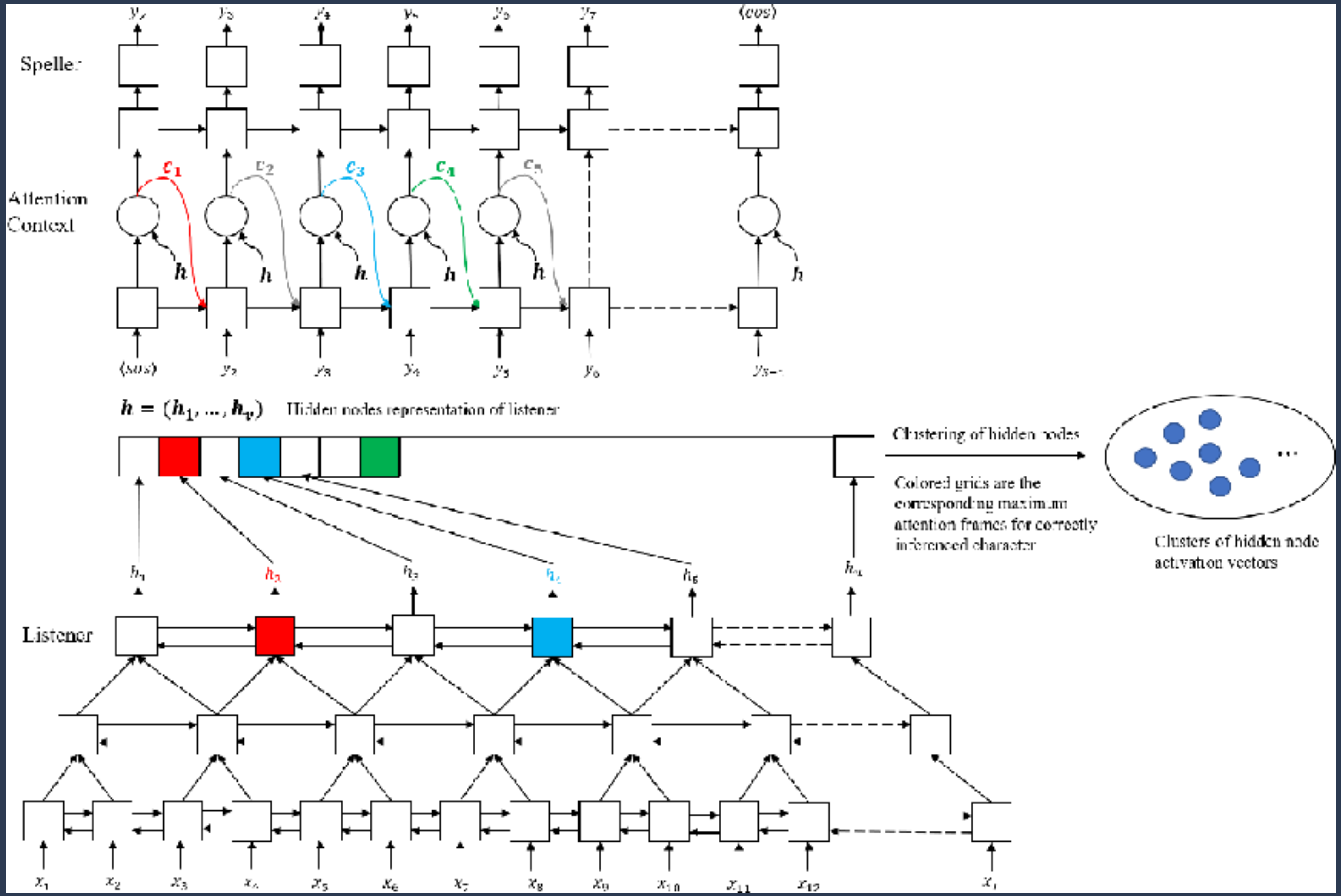
For simplicity, I will use B for Batch, T for Time and D for the other dimension in the following recitation.

Listen, Attend and Spell Architecture

Overview of the architecture

- Listener: Encoder
- Speller: Decoder, also, it's a language model.
- Attention: Doc Product

- **Listener** produces a high-level overview of the given utterance.
- Then the high-level representation is fed into the **Speller** to generate a probability distribution over the next <u>characters</u>.
- Dot Product Attention is used to add the context information.

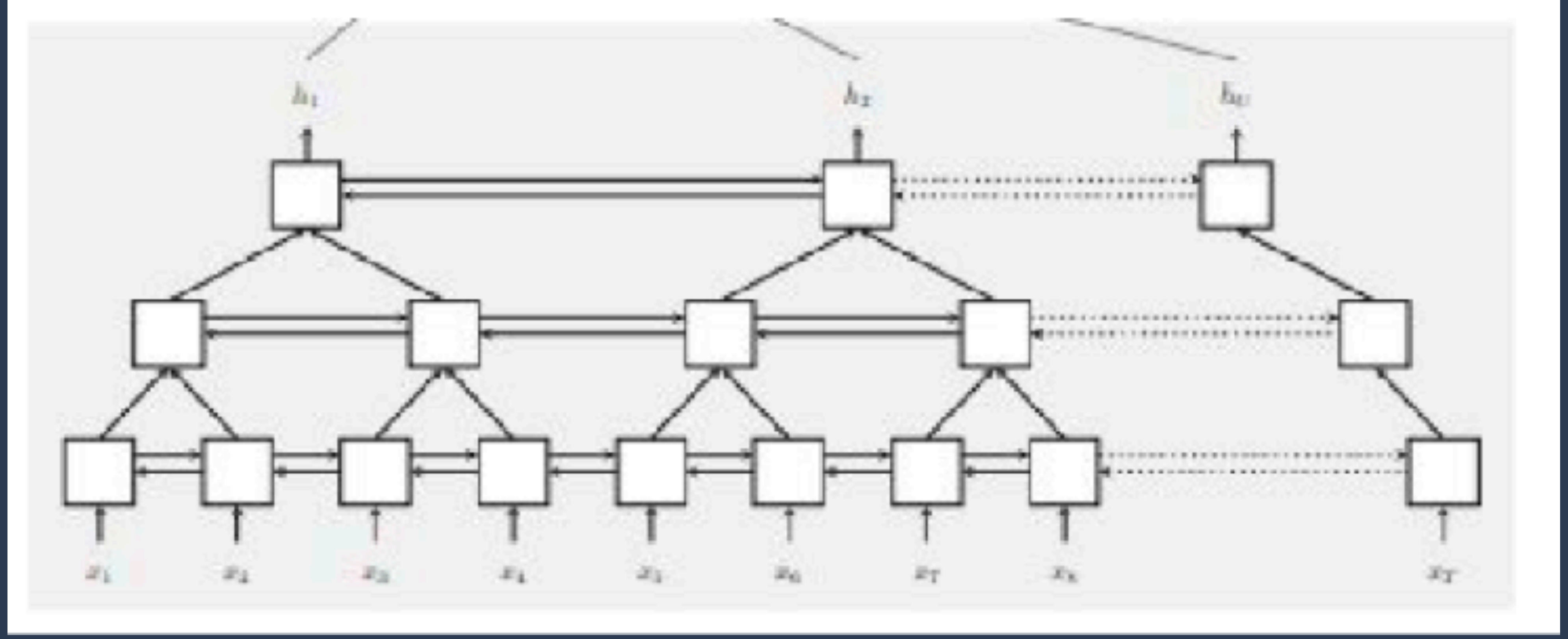

Speller
Attention Context
Listener
c_1
c_2
c_3
c_4
c_5
h
$\langle sos\rangle$
$\langle eos\rangle$
$h = (h_1, \ldots, h_u)$ Hidden nodes representation of listener
Clustering of hidden nodes
Colored grids are the corresponding maximum attention frames for correctly inferenced character
Clusters of hidden node activation vectors
h_1
h_2
h_3
h_4
h_5
h_u
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
x_{11}
x_{12}
x_t

Listener

- Takes as input an utterance. (B, T, D)

- **Feeds through a series of stacked pBLSTM layers to reduce the time resolution.**
  - Intuition: Each utterance is too long for the LSTM to learn (gradient vanishing problem) and also reduce the time resolution could generate better representations just like CNN pooling method.
  - pLSTM: Pyramid Structure. Concatenate pairs of inputs over the frequency dimension. (B, T, D) -> (B, T//2, D * 2)
  - 3 stacked pBLSTM follow a regular BLSTM.

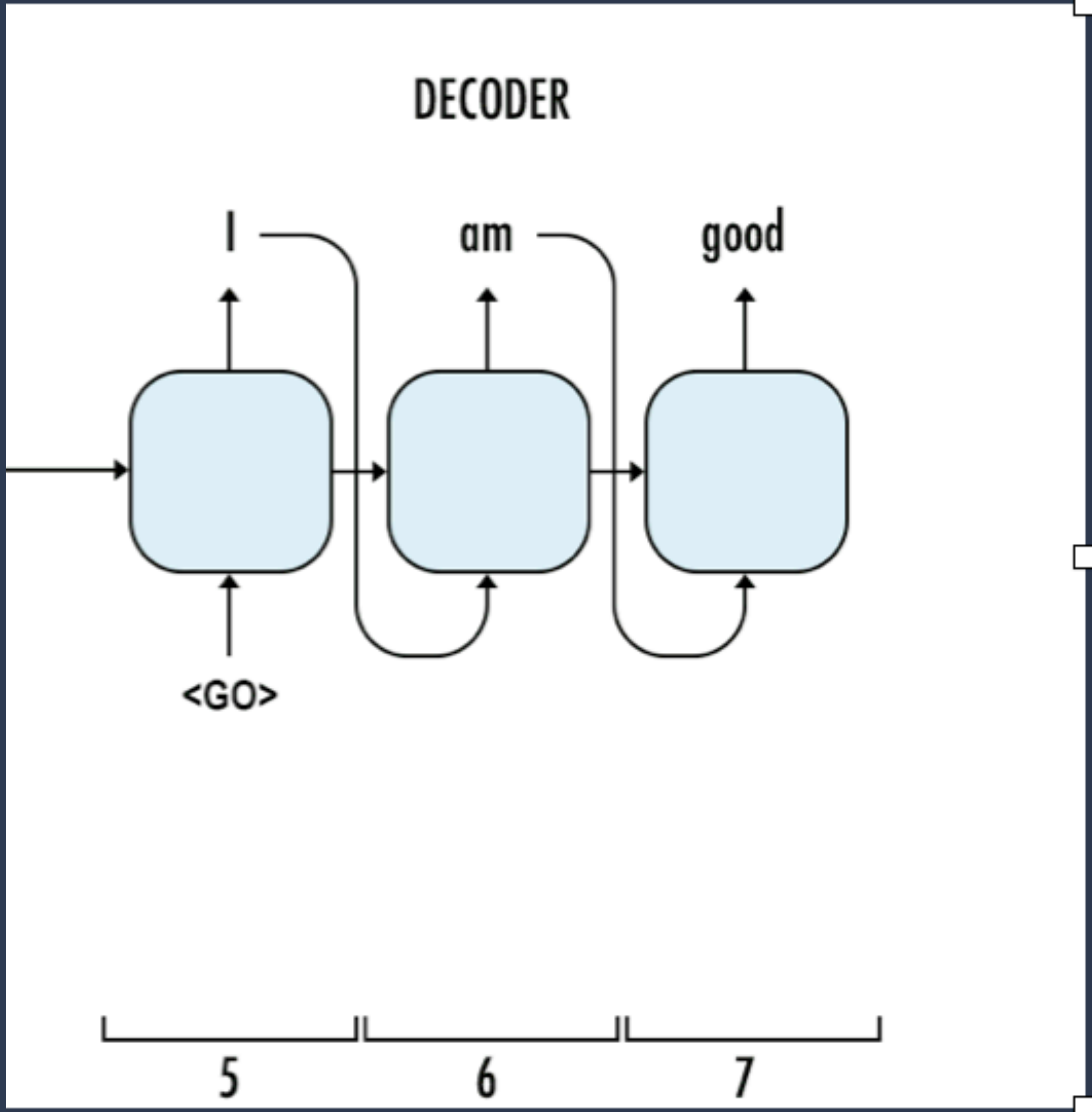
- **Recommended Variation From the Paper:**
  - Instead of having a single output from the pBLSTM as the key, value, use Two separate MLP to project the hidden states to get a key and values. (Just like scaled dot product attention in the last recitation.)



Speller

- Language Model, i.e. takes in characters/words + hidden states, and predicts the probability over the vocabulary. (hw4p1)
  - Instead of unigram LM, this time LM would predict the next word given all previous inputs + context information(attention).

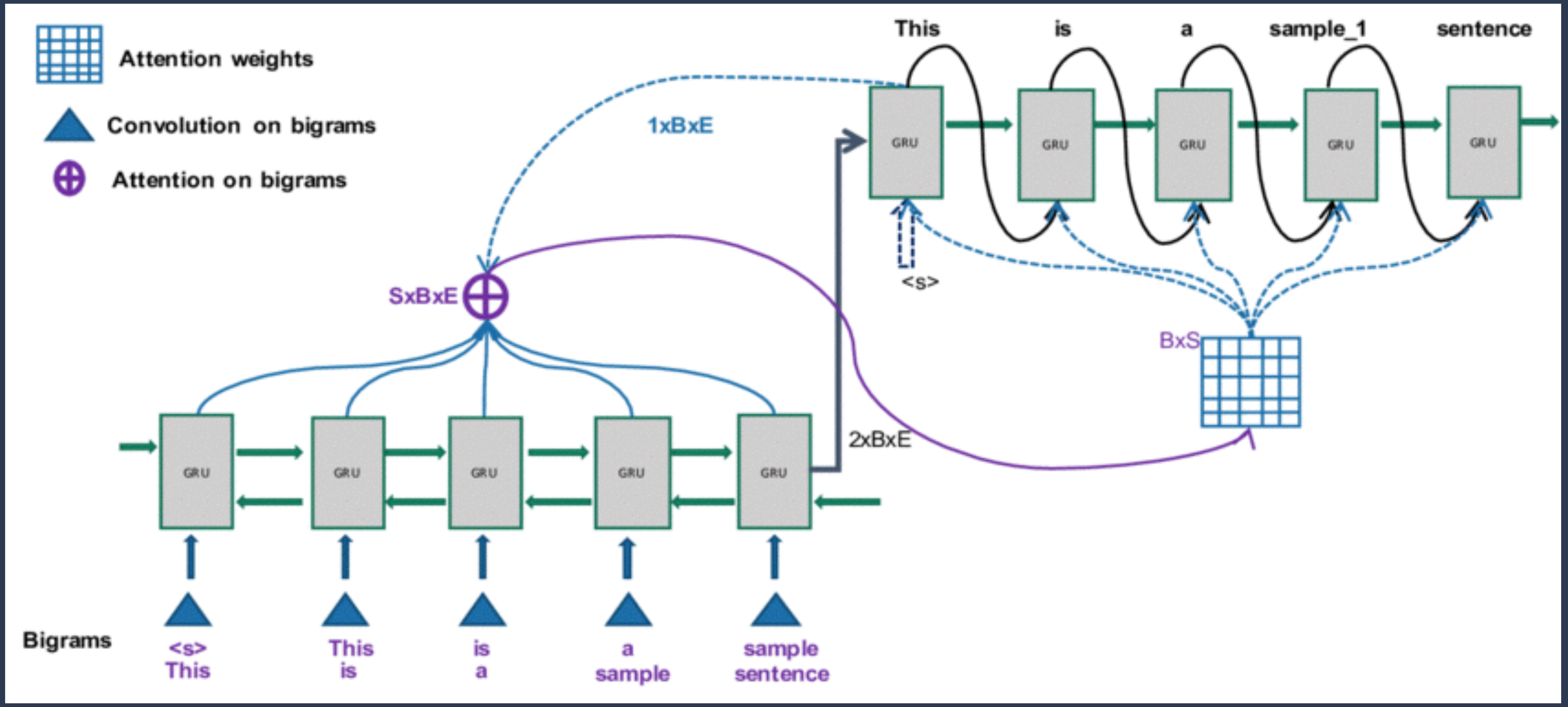
- Trained using Cross-Entropy - because we have the group truth next char/word for each time stamp. (No alignment issue like what you had in HW3P2)



Attention: Connecting the Two

- Attention aims to generate a probability distribution over all keys. It uses the keys from the encoder and query from the decoder to produce a weights over all values from the encoder. Then weights are applied to those values to get a condense context representation.

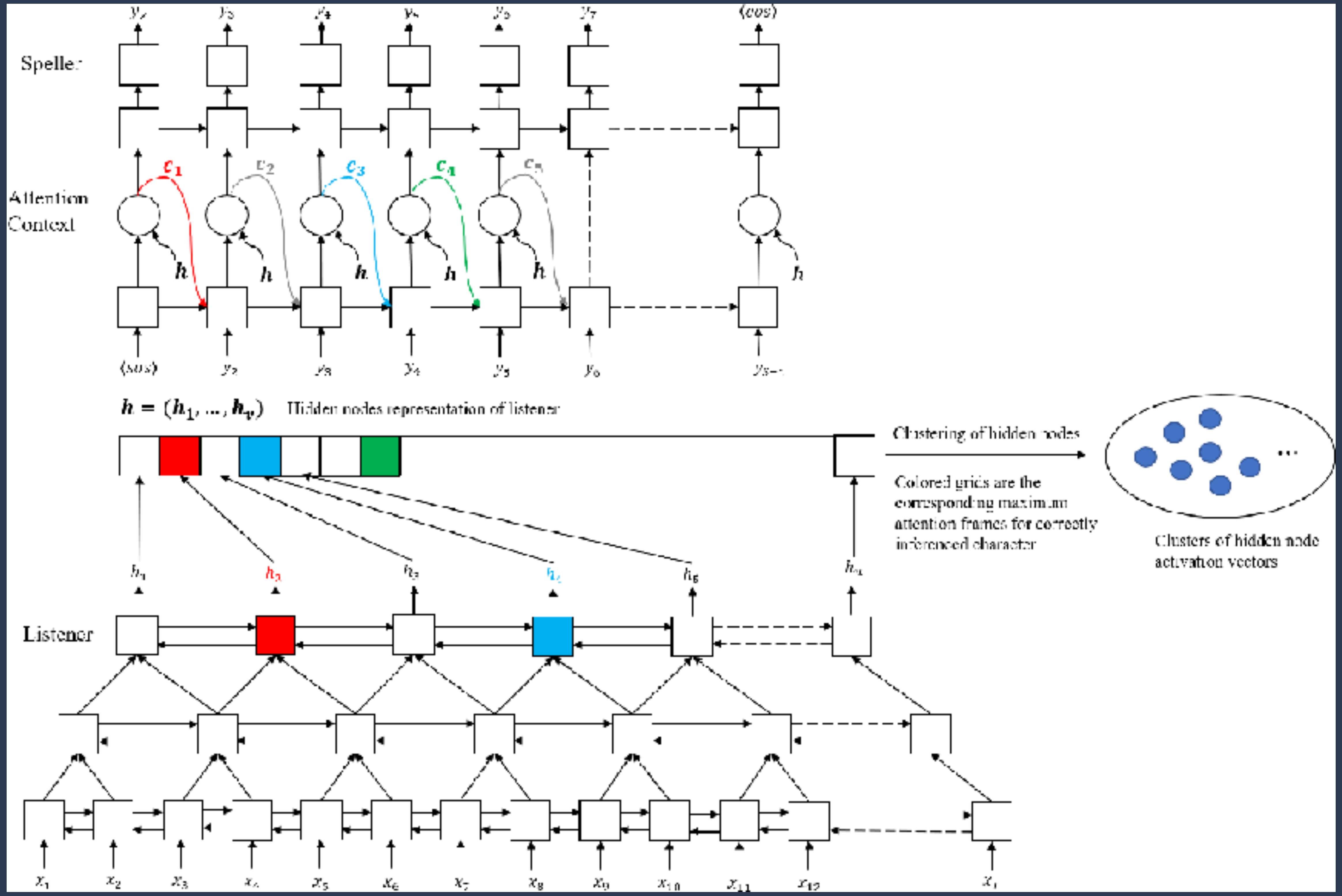
- Keys and Values are high-level representations of the utterances from the encoder.
  - It would the same. Like keys and values could both be the hidden states.
  - It could be different. Use two different MLP to project the original hidden states.



- At every tilmestep in the decoder, you will be computing the attention context using the keys, values and query as follows:
  - *Energy = bmm(key, query). # (B, T, D) x (B, D) —> (B, T)*
  - *Weight = softmax(energy)   # (B, T)*
  - *Context = bmm(weights, values). # (B, T) x (B, T, D) -> (B, D)*

Putting It All Together

- Decoder uses 2-layer LSTM as described in paper
- Decoder input at every timestep is the concatenated character embedding and attention context vector
- You can use initial context of 0’s in the first timestep when generating the first query
- Could also try to learn this initial context by wrapping it in nn.Parameter so gradients will flow to it. However this can lead to bugs (and has in the past semester), so start with 0’s
- Outputs probabilities over the vocabulary as well as a query for attention at the next time step using two distinct projection/linear layers
- Paper suggests concatenating attention context with the final hidden value for the character distribution projection as well



Speller
Attention Context
Listener
y_1 y_2 y_3 y_4 y_5 y_7 $\langle eos \rangle$
c_1 c_2 c_3 c_4 c_5
h
$\langle sos \rangle$ y_2 y_3 y_4 y_5 y_6 y_{S-1}
$h = (h_1, \ldots, h_U)$ Hidden nodes representation of listener
Clustering of hidden nodes
Colored grids are the corresponding maximum attention frames for correctly inferenced character
Clusters of hidden node activation vectors
h_1 h_2 h_3 h_4 h_5 h_U
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_T

- 1) train the speller using LM without attention context to make sure you have bug in your decoder.
  - Encoder could be decoupled with Decoder. Adding Encoder context should only boost your performance. Your network should learn even without context.
  - Trained the LM, get the weights and embeddings for each char/word. Use them to initialize your encoder-decoder architecture.
- 2) Train End-2-End. You may take the risk of debugging a monolithic model. (based on my experience, I spent more than 1 week on debugging. Rewrite the whole architecture for two times.)

Handling Variable Length

★Transcripts are also of variable length

★In hw3p2, CTCLoss internally masked using the target lengths

★With CrossEntropyLoss we must mask manually

★Use *reduction = “none”* to return a matrix where every entry is the loss w.r.t a single element

★Mask out every value in a position that is longer than the transcript

★Each row will be padded to the longest length, so mask out values that are in position greater than the true length

★Only want loss in a position that corresponds to an actual part of the sequence

★Include EOS token - we still need to learn when to end the sequence

★Sum this masked loss and divide by sum of target lengths

LSTM Cell

- LSTMCell is required in the decoder because we must access the hidden values at *every timestep*
  - Must use the hidden states to produce the attention query
  - Can’t use packed sequences with nn.LSTMCell
- Loop through time steps, i.e. lengths of transcripts
  - Input at each time step has size: (B, embedding_size + context_size)
  - Input will be the concatenation of char/word embedding and attention context
- You loop through T when you have transcripts, when you have no transcripts at inference time, set some max value of generated tokens (200-300 is fine for char model)

Teacher Forcing

★A large problem between train and inference: during training, the speller is always fed true labels from the previous timestep, but during prediction it must use its predicted output.

  ★- If you predict one wrong character, the model would have no idea how to recover and you will get bad results.

★One solution is teacher forcing - add noise to the input symbols provided to the speller by sampling from the output distribution of the previous timestep some fraction of the time

★ - In this way the model learns to handle an input that is generated from your model, instead of the gold standard

★ - Becomes resilient to false predictions

★Increase teacher forcing rate -> more focus on syntactic information;

★Lower the teacher forcing rate -> more focus on phonological information;

Sampling

★Teacher forcing and generation require sampling the output probability distribution

★Method #1: Greedy

  ★Simple to implement but is powerful enough

★Method #2: Gumbel Noise

  ★Add some randomness to the prediction.

  ★Reparameterization trick to simulate a draw from a categorical distribution

  ★Read more here https://casmls.github.io/general/2017/02/01/GumbelSoftmax.html

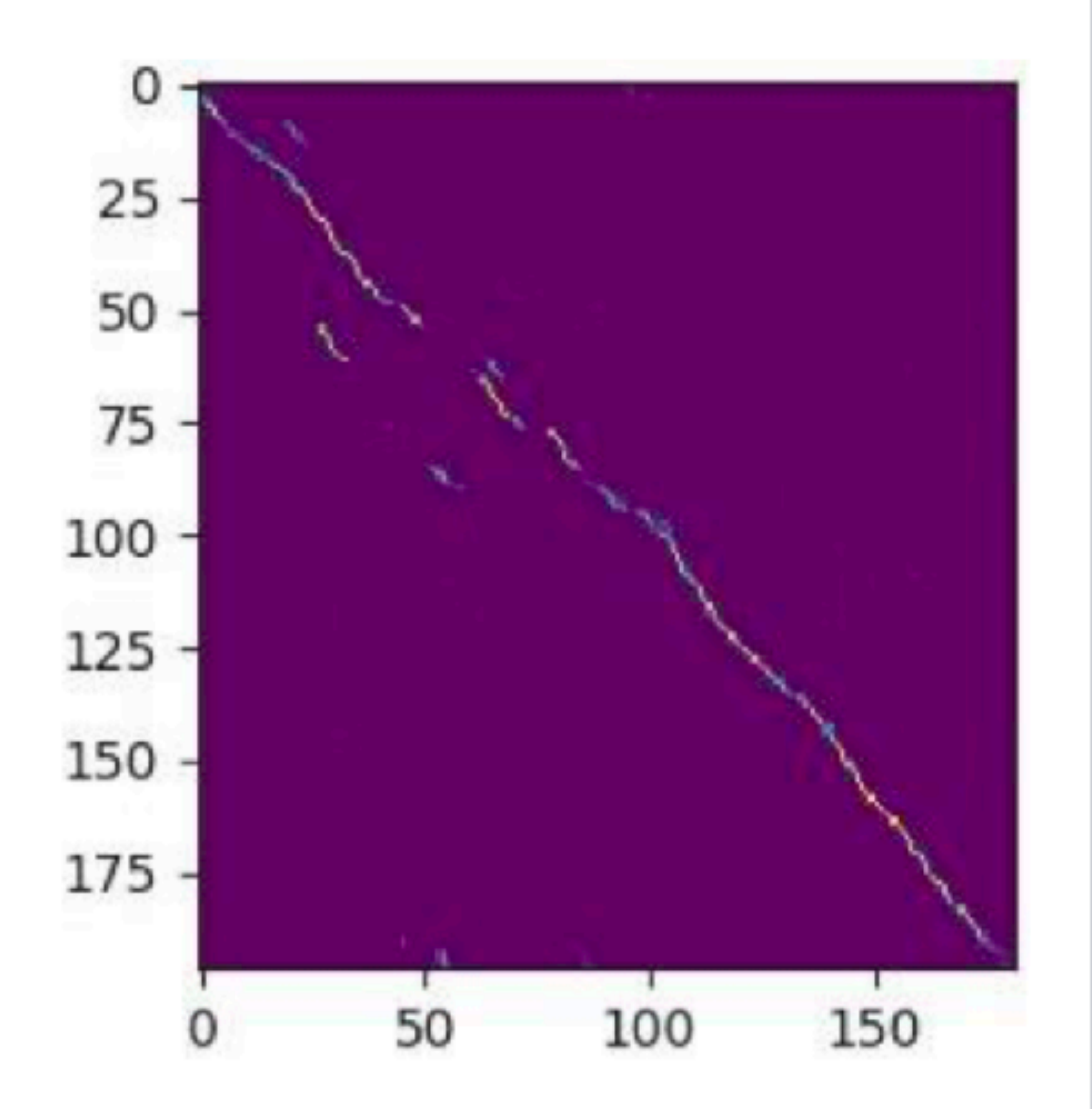
  ★Use torch.nn.functional.gumble_softmax

Inference

- Greedy is simple but will not get the best results
- Beam Search:
  - Requires modifying the forward function of the Speller to expand each of K best beams at every timestep
  - Best results but the most tricky to implement
- Random Search:
  - For each utterance, generate a large number of predicted outputs with some randomness and score each prediction to select the best one
  - Score by feeding the predicted sequences back through the network as targets for the corresponding utterance and compute the loss. Pick the original generated sequence that received the smallest loss.
  - Almost as good as beam search, simpler to implement, but much much slower

Utilities

- ★Plotting attention weights
  - ★Will provide code.
  - ★Plot attention over time for a random utterance
  - ★Attention should traverse the utterance, forming a diagonal
  - ★Should start to form a diagonal after 2-3 epochs
- ★Plotting gradient flow
  - ★Will provide code.
  - ★Check that speller correctly passes gradients back to listener
  - ★Can also inspect the weights to identify if they are changing but this gets tedious
  - ★Use gradient clipping.
- ★Check that normalized attention weights sums to 1
- ★Check that binary masks sum to intended lengths
- ★Transposing and reshaping are different
- ★Be careful you don't make a silly mistake in the pBLSTMs



Only the IDL guys could see this secrete message