Homework 4 Part 2

Attention Models

by Vedant and Chris

Topics for Today

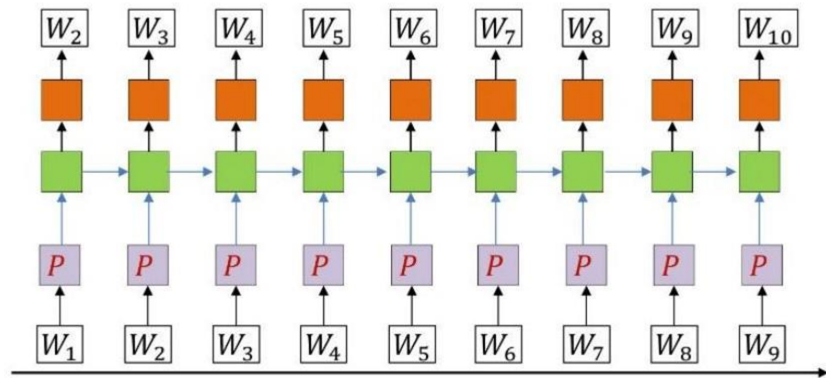
Quick Recap
Generation tasks and language models (recap of recitation 7)
Adding context with attention (recap of recitation 9)

HW4P2 Overview
Introduce Listen, Attend, and Spell
- Listener architecture
- Speller architecture
- Attention computation

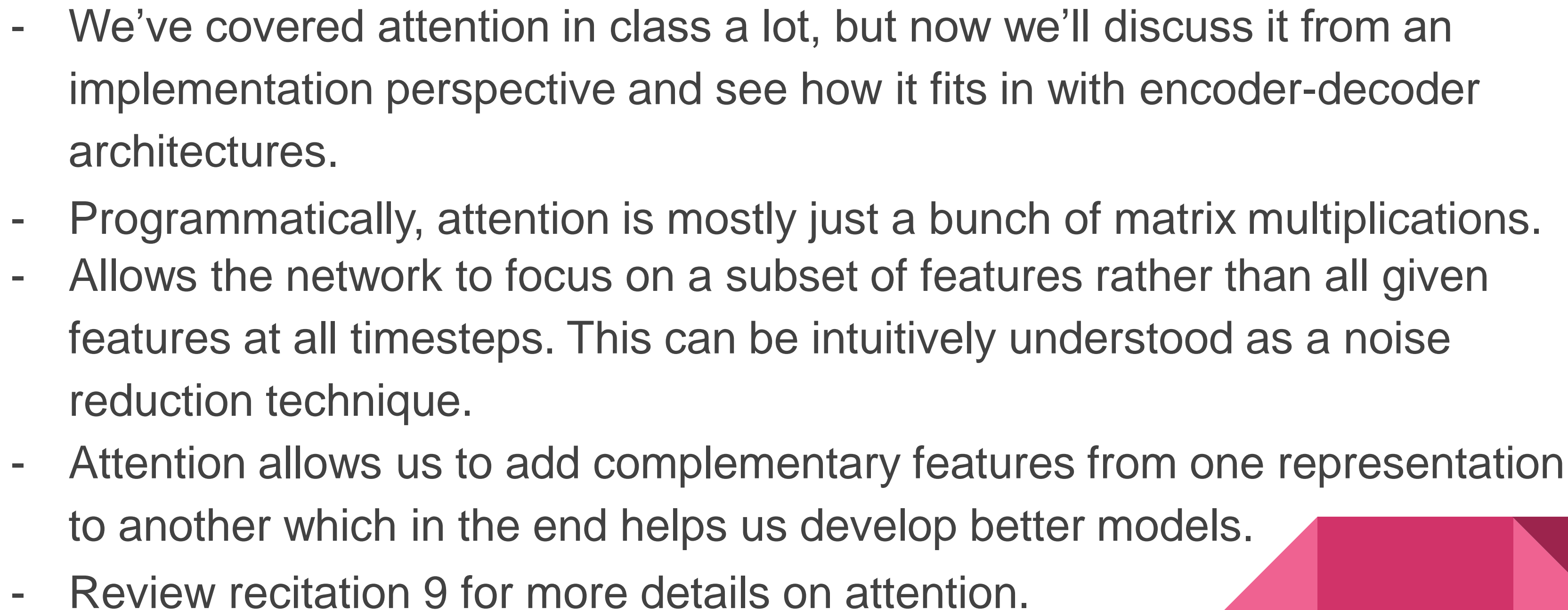
Implementation:
- Pooling with pBLSTMs
- LSTMCell
- Masking - both attention and loss
- Sampling with Gumbel noise
- Teacher forcing
- Prediction methods - greedy vs random vs beam
- Plotting attention and gradient flow

Language Models: Recap of Recitation 7

- RNN’s can learn to predict the next word given a sequence of words
- Create vocabulary of words/characters, then embed those into a lower dimension, input to the model
- Input and target are same sequence off by one character/word
- SOS and EOS tokens added to demarcate sequences
- In a generation task, input a sequence of words and predict the next word, then feed the predicted word back and continue the process
- This input sequence can be simply the SOS token; then the model generates the entire sequence itself (this is how you implement inference time generation in hw4p2)



Adding Context with Attention



Examples of Attention: Caption Generation

- Show, Attend and Tell - Attention-based Caption Generation
  - CNNs encode the image and RNNs are used to generate descriptions
  - RNN is similar to a language model supplemented with attention context
  - Attention maps reveal the network’s ability to focus on objects of interest
  - Attention maps are just a probability distribution over the input

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

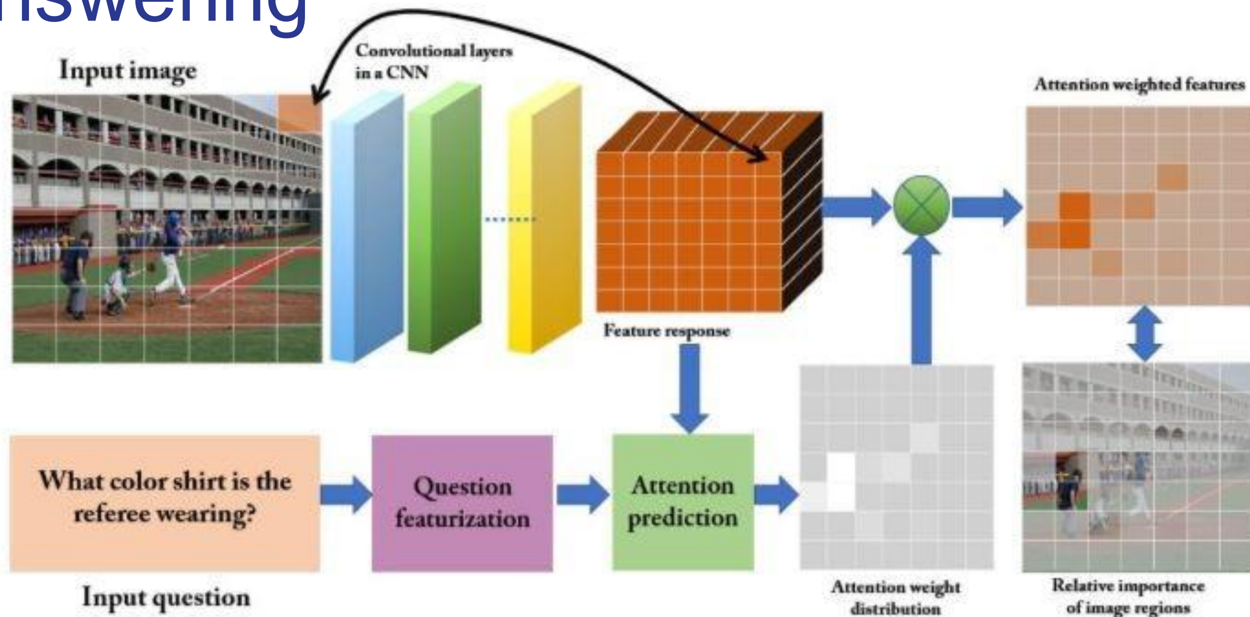
A stop sign is on a road with a mountain in the background.

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.

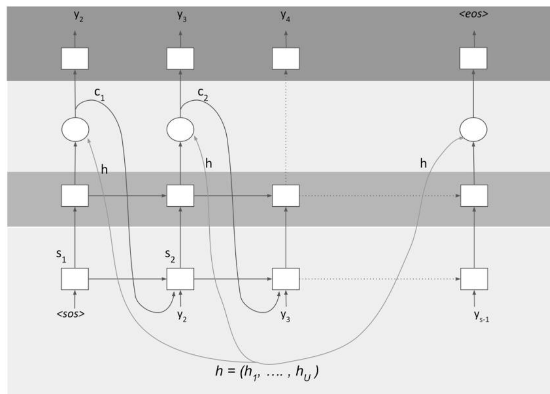
Another Example: Visual Question Answering

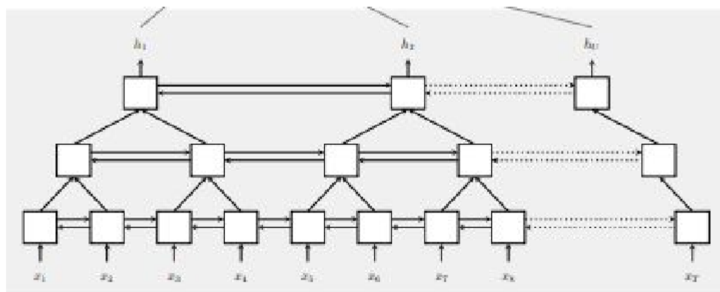


HW4P2 Overview

- In lecture, machine translation translation was used as an example for explaining attention. The encoder took as input the initial sentence, and the decoder was a language model to generate text in the translated language.
- For speech-to-text we want to follow the same principle as with machine translation or caption generation, but instead of a having text or an image as input to the encoder, we pass an utterance.
- Can be achieved with either character-based or word-based approaches
  - Word-based approaches avoid having any spelling mistakes but it is difficult to produce rare words from the vocabulary

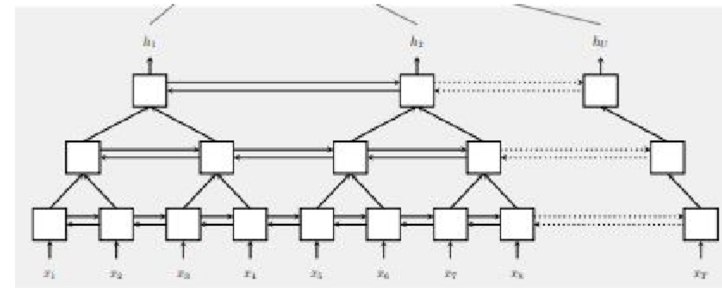
Listen, Attend and Spell



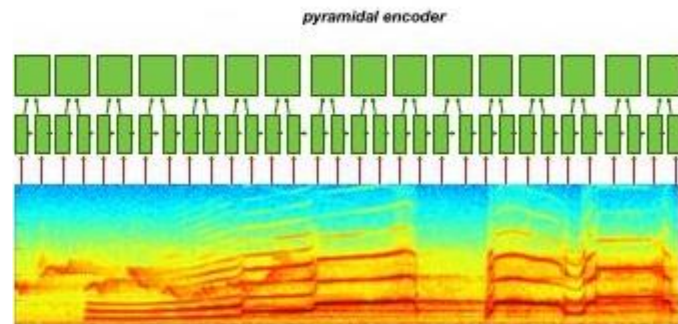


- Two sub-models
  - **Listener** - Encoder
  - **Speller** - Decoder
- **Listener** produces high-level representation from the given **utterance**
- The high-level representation is fed into the **Speller** to generate a probability distribution over the next **characters**

Listener Architecture



- Takes as input an utterance (or batch of utterances)
- Feeds through a series of stacked pBLSTM layers to reduce time resolution
  - Pyramid structure - concatenate pairs of inputs over the frequency dimension (can be done with a combination of .view() and .transpose() operations)
  - Concatenation takes input of size (L, B, F) and outputs size (L / 2, B, F * 2)
  - 3 stacked pBLSTMs follow a regular BLSTM layer



- Recommended variation from paper:
  - Instead of having a single output from the pBLSTM, take two separate linear projections from the output to get a key and value (used for attention, as discussed in recitation 9 and as we'll see again shortly)

Speller Architecture

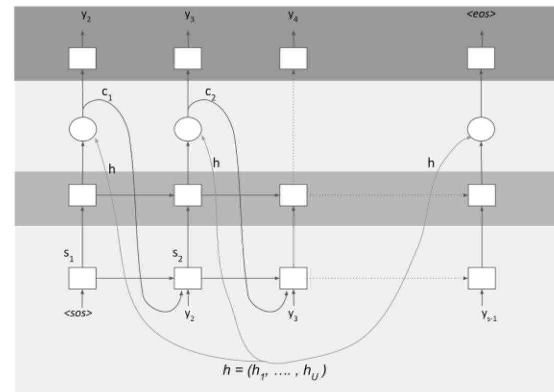
- Language model, i.e. takes in characters/words and predicts probabilities over vocabulary (exactly what you are doing in hw4p1)
- Predicts next character/word given all previous characters/words
- Trained using Cross Entropy - because we have a target char/word for each time step, and our predicted probabilities over chars/words at that time step (no alignment issue to solve)
- We recommend you train the speller as a language model without attention first to ensure that you have no problems with it
    - Listener is an encoder that only amplifies the results, your network should still learn with just the language model
    - Can even use the pretrained weights as initialization for your attention-based models

Attention: Connecting the Two

- Review from recitation 9:
    - Attention aims to generate a probability distribution over the inputs (i.e. identifies where to focus within the input)
    - Uses keys from the encoder and query from decoder to produce a probability distribution, and applies this probability distribution over the values (also from the encoder)
- Keys and values are high level representations of the utterances from the encoder
- At every timestep in the speller you will be computing the attention context using the keys, values, and query as follows:
    - *energy = bmm(key, query)*
    - *attention = softmax(energy)*
    - *context = bmm(attention, value)*

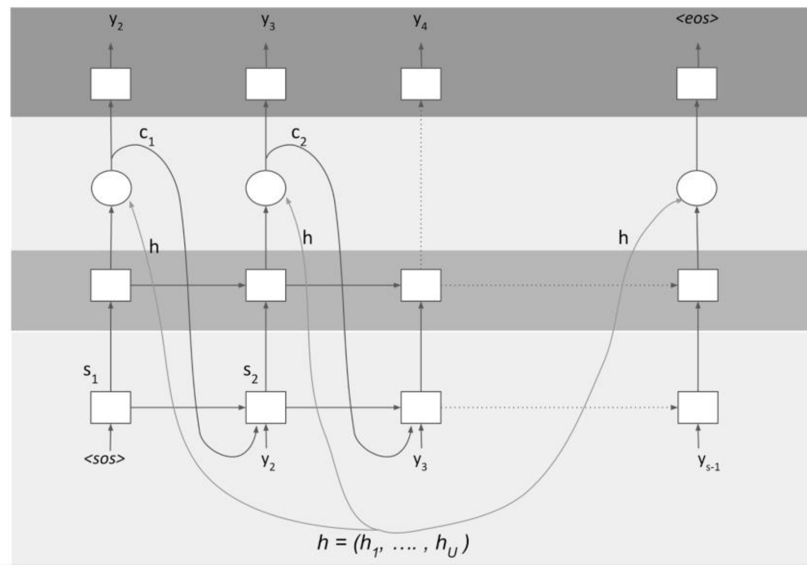
Putting It All Together

- 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

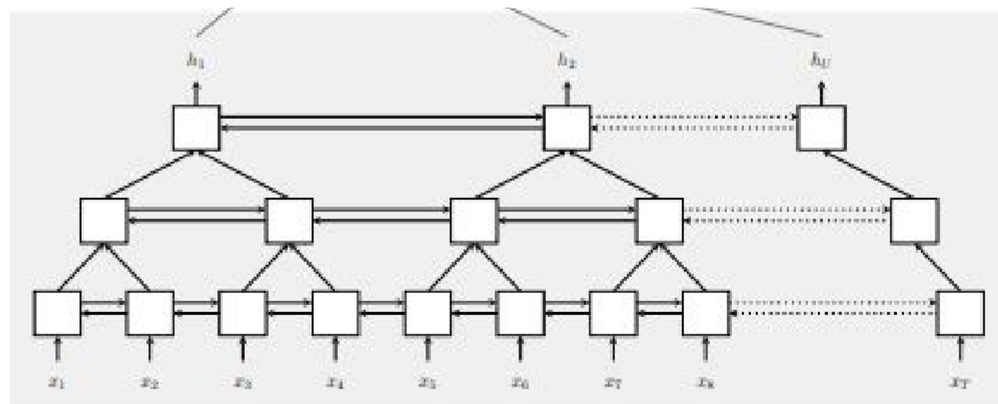


$$
\begin{aligned}
c_i &= \text{AttentionContext}(s_i, \mathbf{h}) \\
s_i &= \text{RNN}(s_{i-1}, y_{i-1}, c_{i-1}) \\
P(y_i|\mathbf{x}, y_{<i}) &= \text{CharacterDistribution}(s_i, c_i)
\end{aligned}
$$

Speller (Decoder)



Listener (Encoder)



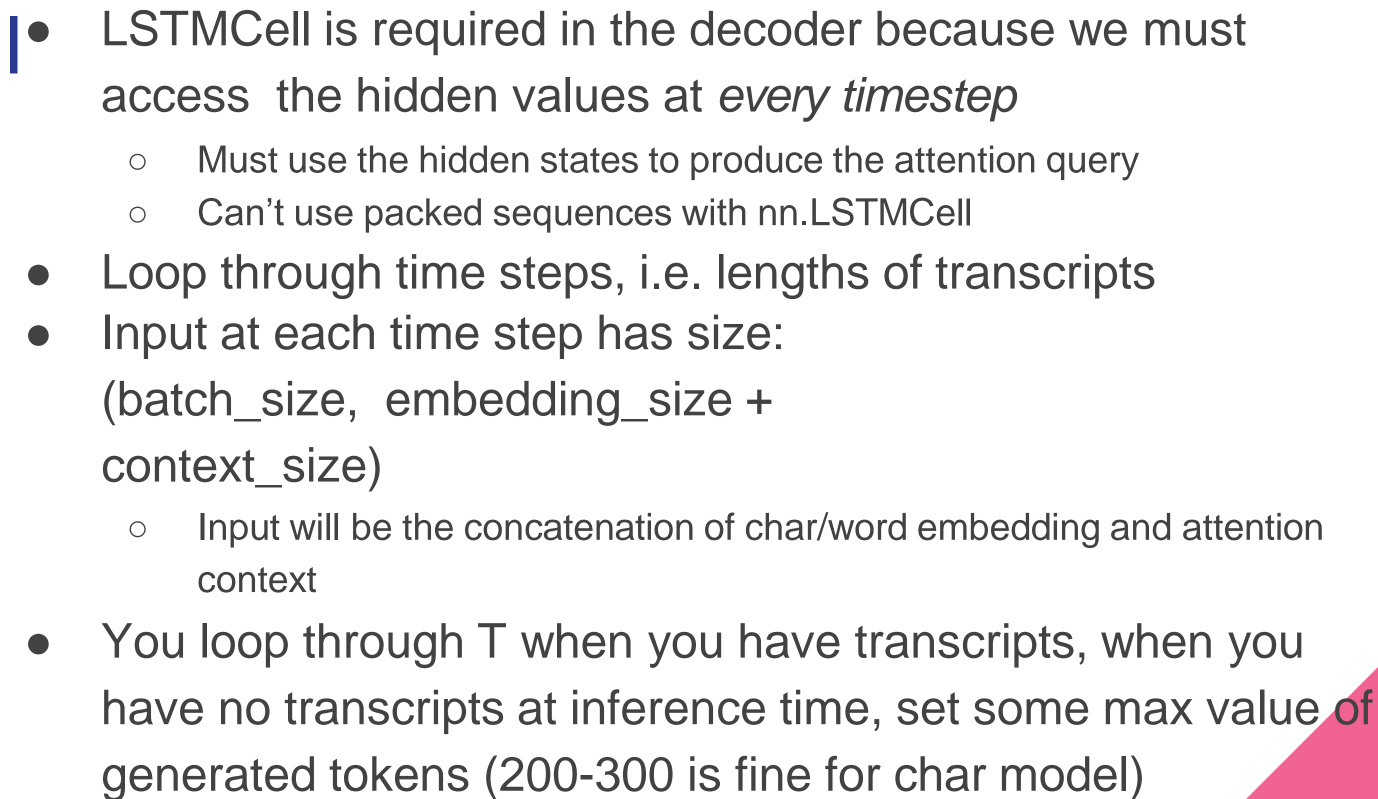
Handling Variable Length Utterances

- Input utterances are of variable length
  - We will need to use padded and packed sequences again
- pBLSTMs reduce time resolution by constant factor
  - Need to track lengths through these operations - divide by constant factor
  - What about odd lengths?
    - Slice or pad - one frame isn't going to make a difference
- Outputs of the listener are still variable length - the keys and values
  - We want use attention to generate a probability distribution over the value, but we want to disregard the padded portion
  - Solution - attention with binary masking:
    - $energy = bmm(key, query)$
    - $attention = softmax(energy)$
    - $masked_attention = normalize(attention * mask)$
    - $context = bmm(masked_attention, value)$

Handling Variable Length Transcripts

- 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

LSTMCel



- 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: (batch_size, 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

Sampling Methods

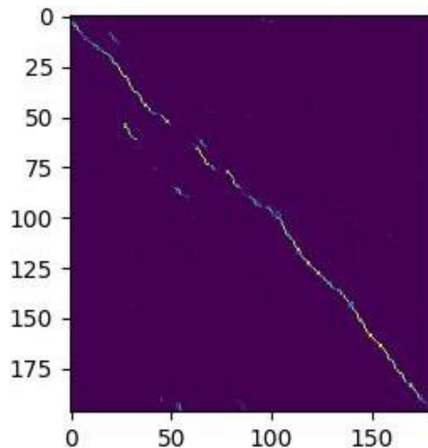
- Teacher forcing and generation require sampling the output probability distribution
- Method #1: Greedy
  - Choose the symbol with highest predicted probability
  - Simple to implement - we recommend you start with this
- Method #2: Gumbel noise
  - Adds some randomness to the prediction
  - Reparameterization trick to simulate a draw from a categorical distribution
  - Read more here: https://casmls.github.io/general/2017/02/01/GumbelSoftmax.html

$$X = \arg\max_k \left(\log \alpha_k + G_k\right)$$

Inference for the Test Set

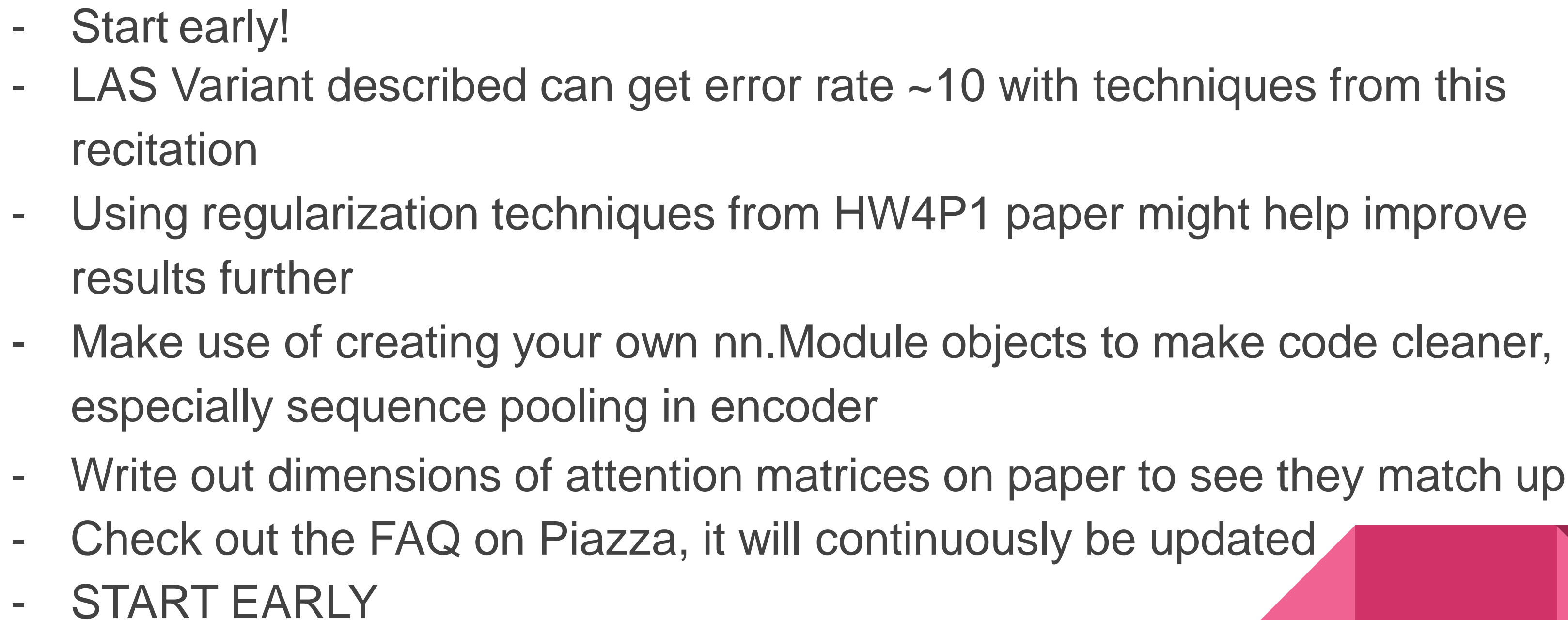
- Greedy is simple but will not get the best results
- If using another sampling method (like Gumbel noise) you can save selections at each timestep for another simple approach - probably better than greedy but still not great
- Better approaches are Random Search and Beam Search
- Beam Search:
    - 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 slower

Utilities/Sanity Checks



- Plotting attention weights
  - Will put code in FAQ's
  - 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 also put code in FAQ's
  - Check that speller correctly passes gradients back to listener
  - Can also inspect the weights to identify if they are changing but this gets tedious
- 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

Words of Advice



That’s all for today!

Good luck!