Convolutional Attention Consistency: Towards Efficiently and Flexibly Trainable TTS

Introduction

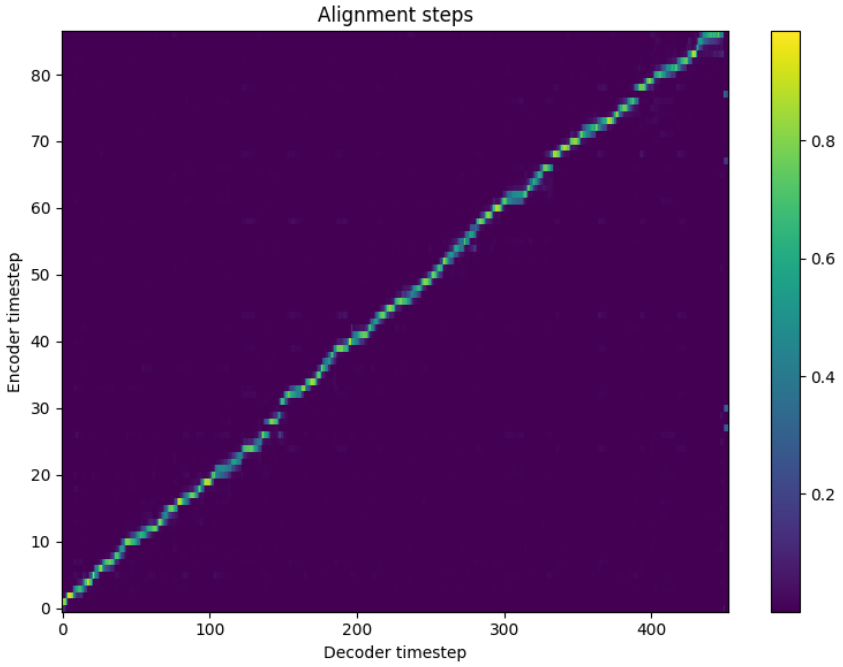
Autoregressive Text to Speech (TTS) models such as Tacotron 2 and Flowtron have shown great ability to synthesize natural speech, even with non-autoregressive options available. However, autoregressive models are delicate to train and can suffer from various problems such as skipping tokens, misspelling words, or even catastrophic collapse at inference time.

Various techniques have been already proposed in order to reduce these issues and training difficulty. However, many of these options are either inflexible, increase model size significantly, or depend on pre-alignment data.

In this paper, we propose Convolutional Attention Consistency, a method that helps alignment learning without any of the aforementioned disadvantages.

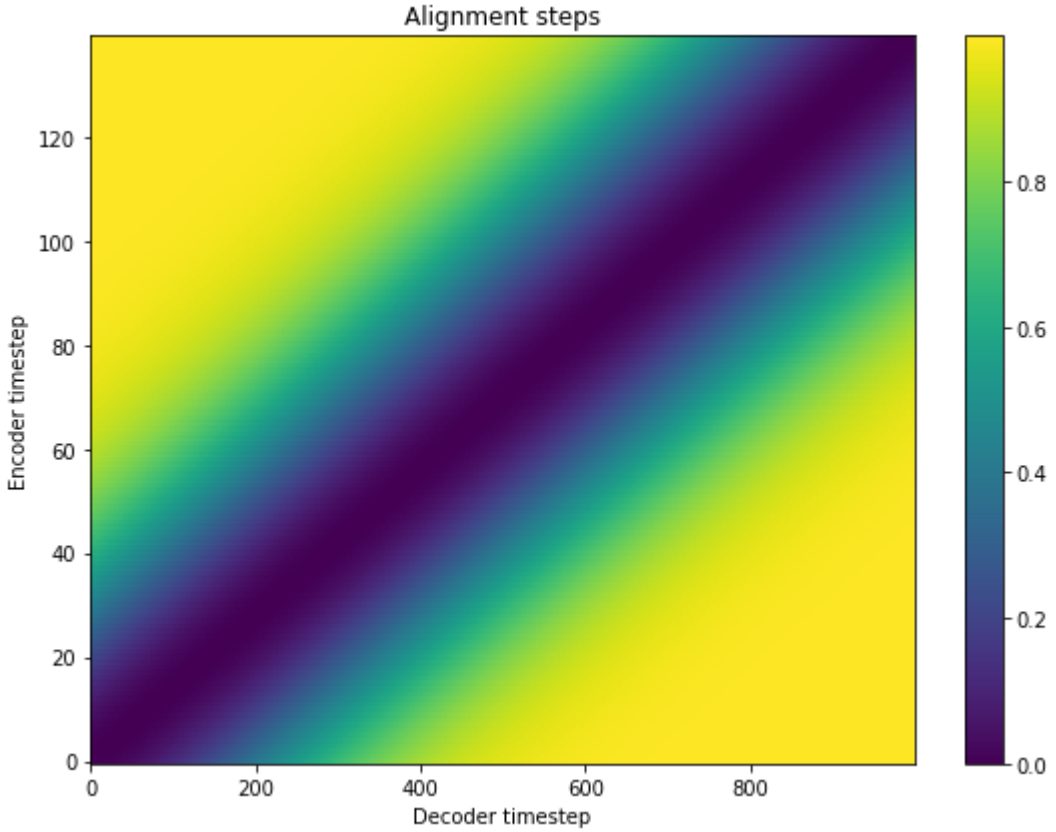
Previous Approaches

Diagonal Guided Attention



During training, the model has to learn the alignment between characters and mel-spectrogram frames. This is the hard part

The most widely used technique today is Diagonal Guided Attention, which makes the *mostly* true assumption that a healthy alignment line is diagonal, and thus creates a mask and punishes the decoder alignment for straying outside of it.



Example of a mask generated by the diagonal guided attention function. The more it deviates from a diagonal line, the higher the loss.

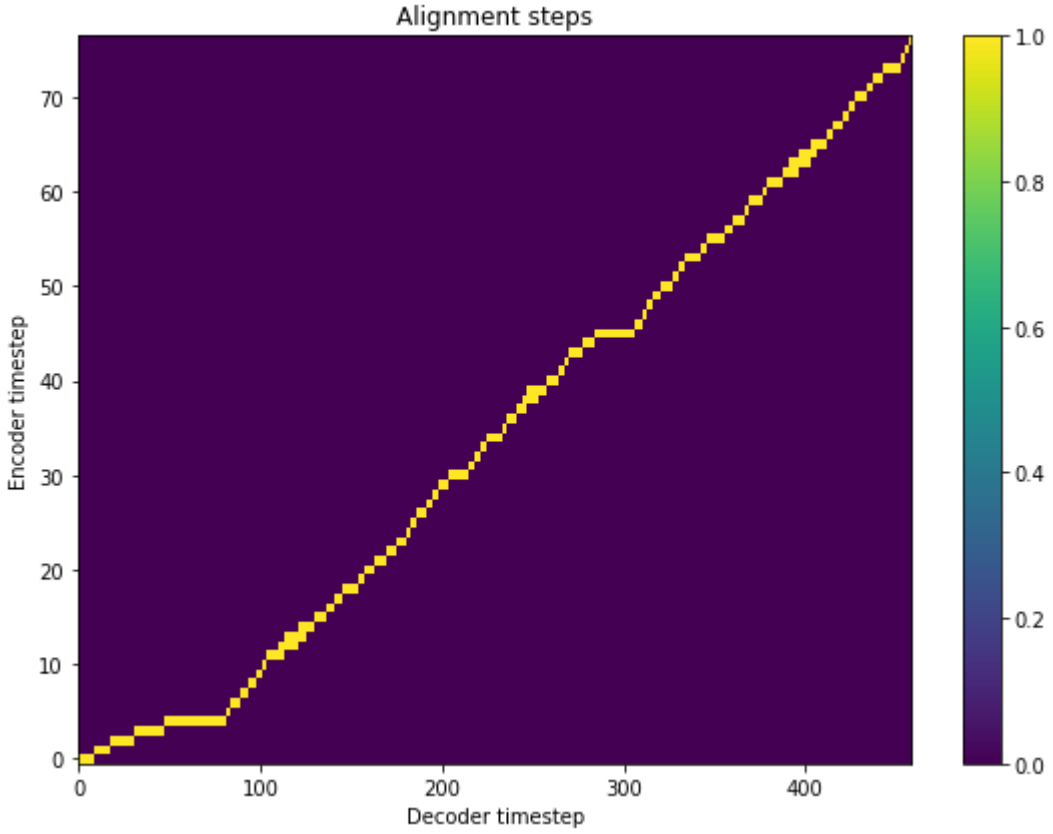
This approach works *fine,* however, it fails to account for various factors:

1. What if our speaker has an inconsistent speaking rate?
2. What if our speaker likes to extend vowels or silences a lot?
3. What if our speaker exhibits emotions?

Under scenarios like the three aforementioned, DGA will lead to unnatural speech, inexplicably high loss, or even collapse in training. This is because it's **inflexible.** While it is possible to tune the function so the mask isn't as "narrow", at that point one is trading training efficiency for flexibility. It would be nice if we had both.

Forced Aligner-based

Another technique is utilizing pre-alignment information from a forced aligner like Montreal Forced Aligner (MFA) as guidance during training, like Pre-Alignment Guided Attention, or Forced Alignment Guided Attention Loss (own previous work).



Example of attention extracted from MFA

The disadvantage here is obvious: it requires the setup and use of a forced aligner, which might not be 100% accurate, or even *be available at all* for more obscure languages.

Double Decoder Consistency

Double Decoder Consistency (DDC), which inspired this approach, proposes simultaneously training two decoders, a “fine” decoder with a low reduction factor, and, taking advantage of the fact that a higher reduction factor makes it easier to learn alignment, a “coarse” decoder with a higher r, the value which defines the number of output frames per decoder step. The coarse decoder’s attention output is upscaled to match the size of the fine one, and a loss function punishes differences between the two, guiding the fine decoder (used for inference) with the coarse decoder’s easier learning.

This one works, except that two decoders = bigger model. We observed that DDC Tacotron2 checkpoints in the disk are roughly 2x the size of regular ones. Also, the coarse decoder’s attention is less detailed.

Convolutional Attention Consistency

With our technique, taking advantage of the [One TTS Alignment to Rule Them All] paper, we take the convolutional attention module and AttentionCTCLoss from NVIDIA's Mixer-TTS, adding it to Tacotron 2. We then punish the alignment differences between the decoder and the convolutional attention, hence the name *Convolutional Attention Consistency* (CAC).

Since the convolutional attention module is very good at learning the alignment, it efficiently guides our autoregressive model.
Our loss function is as follows:

First, to define the Charbonnier loss function:

$$\text{charb}(\text{x},\ \text{y}) = \sum_{i=1}^{n} \sqrt{(x_i - y_i)^2 + \epsilon^2}$$

Let L_{fs} be the forward sum loss (from the paper mentioned previously), A_{dec} be the alignment from the decoder, A_{soft} and A_{hard} be the soft and hard outputs from the convolutional attention module.

$$L_{cs} = charb(A_{dec}, A_{soft})$$
$$L_{ch} = charb(A_{dec}, A_{hard})$$
$$L_{bin} = A_{hard} \odot logA_{soft}$$
$$L_{cac} = L_{cs} + L_{ch} \times 0.5$$

Therefore, the total loss becomes:

$$L_{align} = L_{fs} + L_{cac} + L_{bin}$$

In our experiments, we found that factoring only the difference in the decoder alignment and hard convolutional attention produced stronger alignment confidence, but the resulting speech would be unnatural. Therefore, we also factor in the soft attention output.
Importantly, as explained in the [RAD-TTS paper], we add in BinLoss (L_{bin}) after the model has learned to align, around 50 epochs in training from scratch – without it, the resulting speech is very unnatural.

Experiments

***POST-NOTE 31 AUG 2023:* We found an error in the preprocessor that did not add an EOS token, therefore crippling the performance of all models. After a fix, CAC-V4 learns alignment faster and stronger.**

We use the 22.05KHz LJ Speech dataset from Keith Ito, and train each model up to 110k steps, with batch size 32 and a starting LR of 0.001. For the transcriptions we use a custom ARPA phonemizer.

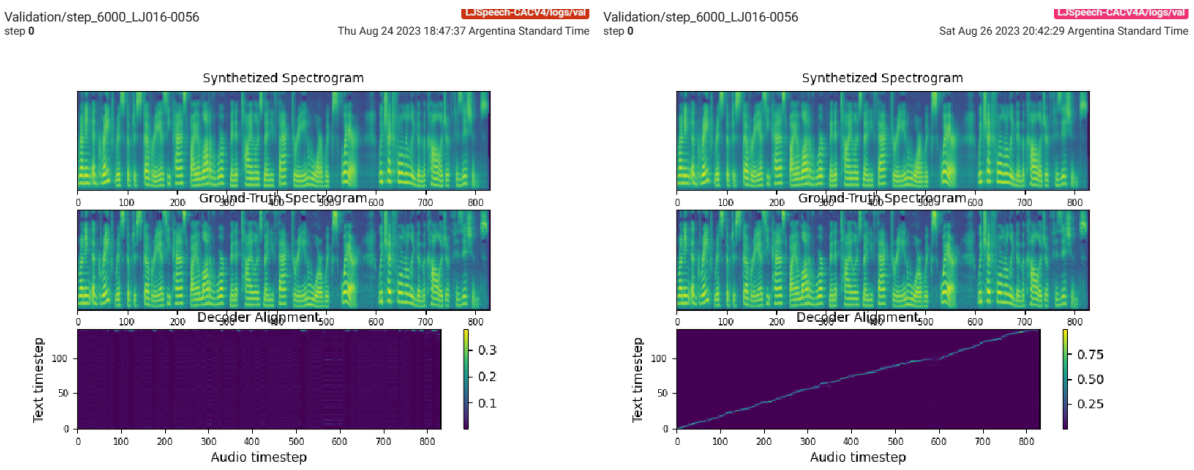
We trained many models, with our findings being:

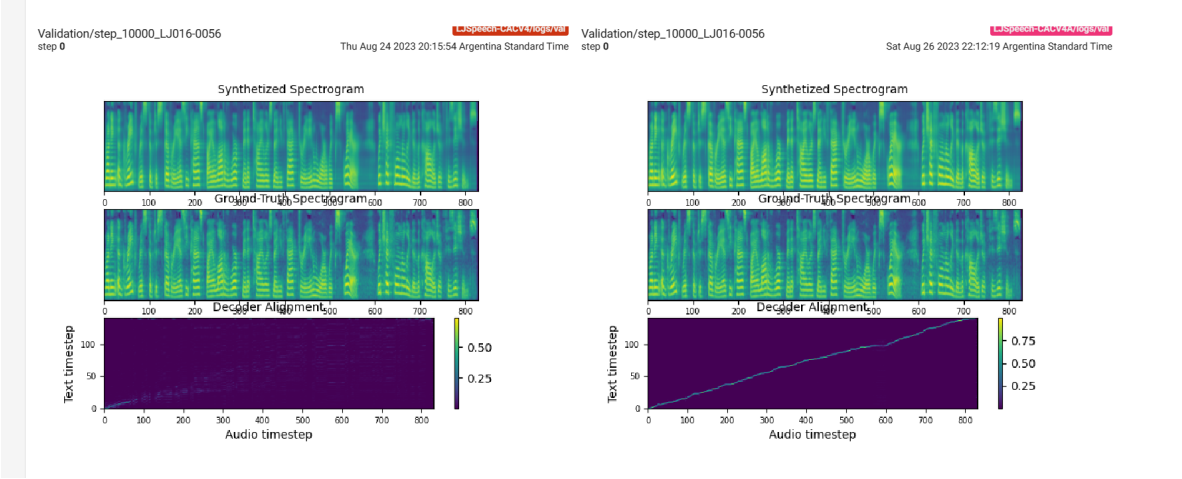
1. Charbonnier loss function is better than MSE for enforcing consistency
2. Factoring in both the soft and hard alignments is vital for good performance
3. Using BinLoss is also important for naturalness
4. Using LayerNorm and 0.1 dropout on the character embeddings yields stronger performance.

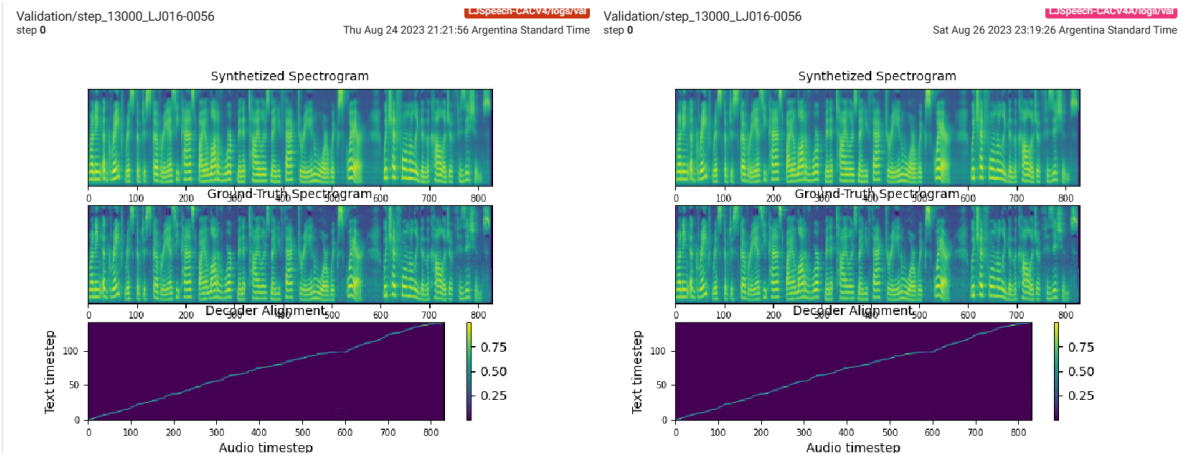
Snake1d vs ReLU AlignmentEncoder

For convolutional attention, we use the AlignmentEncoder module from Mixer-TTS. During experiments, we observed that if we replace the ReLUs with Snake1d, the model learns to align much faster.

V4 (regular ReLU AlignmentEncoder) vs **V4A** (Snake1d) at 6000, 10k, and 13k steps







As described in [Reproducibility Report: Neural Networks Fail to Learn Periodic Functions and How to Fix It], the Snake activation function claims to be better at learning periodic functions. We also noticed a negligible increase in the per-step time with Snake1d, but again it's basically insignificant especially considering the better learning. V4A takes 5 more minutes than V4 to reach 7k steps.

However, once the model is used on out of domain text, the speech (at 110k) is noticeably worse. We theorize this is because the Snake activation function focuses too much on minute details – or maybe it just requires more training. More research is needed.

Further ideas

Instead of using the convolutional attention as a loss directly, one could generate a mask around it and punish the decoder alignment for straying too much outside the region – acting effectively like guided attention loss, but from a function that accurately represents the rhythm of the utterance.
This approach would be called Convolutional Guided Attention (CGA), and will be looked into later.

References

We would like to extend our appreciation to the following people/repos:

- GitHub - keonlee9420/Comprehensive-Tacotron2l: Tacotron 2 implementation
- NVIDIA/NeMo: NeMo: a toolkit for conversational AI (github.com): Mixer and RAD-TTS, with the AlignmentEncoder
- TensorSpeech/TensorFlowTTS: Tacotron 2 implementation with LayerNorm and Dropout embeddings
- GPT-4 by OpenAI: Writing the demo site