IRIT-UPS DCASE 2022 TASK6A SYSTEM: STOCHASTIC DECODING METHODS FOR AUDIO CAPTIONING

Technical Report

Etienne Labbé, Thomas Pellegrini, Julien Pinquier

IRIT (UMR 5505), Université Paul Sabatier, CNRS, Toulouse, France
{etienne.labbe,thomas.pellegrini,julien.pinquier}@irit.fr

ABSTRACT

This document presents a summary of our models used in the Automated Audio Captioning task (6a) for the DCASE2022 challenge. Four submissions were made using different decoding methods : beam search, top k sampling, nucleus sampling and typical decoding.

***Index Terms*—** convolutional encoder, transformer decoder, beam search, top-k sampling, nucleus sampling, typical sampling

1. INTRODUCTION

Automated Audio Captioning (AAC) is a task that aims to describe an audio signal using natural language. An AAC system must be able to detect different types and lengths of audio events, which may overlap each other. The output is a short sentence containing a description of the events with temporal, spatial or physical relationships between them. Those systems could be used for hearing-impaired people, machine-to-machine interaction, surveillance or information retrieval.

The DCASE2022 AAC challenge propose to build an AAC systems design for the Clotho dataset, and this report summarizes the models used in our submissions. Our submission focuses on experimenting with stochastic methods used in text generation for automated audio captioning. The source code will be available on Github [1] after the end of the challenge. We also developed a facility package to load AAC datasets called "aac-datasets" [1] available on PyPi.

2. SYSTEM DESCRIPTION

2.1. Data processing

The dataset provided for the challenge is Clotho [2], which contains 6974 audio files from Freesound between 15 and 30 seconds. Each audio is described by 5 captions annotated by humans. To extract audio features, we resample audio signals from 44.1KHz to 32KHz and compute log-Mel spectrograms with a window size of 32ms, a hop size of 10ms and 64 Mel bands. Captions are put in lowercase and punctuation is removed. The captions are tokenized using the spaCy tokenizer [3], which gave a vocabulary containing 4370 different words in the training subset.

[1] https://github.com/Labbeti/dcase2022task6a

2.2. Model architecture

We adopt a standard encoder-decoder structure used in most AAC systems, with a pre-trained encoder to extract audio features and a transformer decoder to generate our caption. The encoder is the CNN10 model, a convolutional network from the Pretrained Audio Neural Networks study [4] (PANN) pretrained on AudioSet [5] for audio-tagging. We used the weights available on Zenodo [2] to initialize the model at the beginning of the training. An affine layer was added to project 512-dimensional to 256-dimensional embeddings. We kept the time axis in the audio embedding for the decoder.

The decoder is a standard transformer decoder [6] based on the official PyTorch implementation. It takes the audio embeddings as inputs and all the previous words predicted. The word-embeddings are randomly initialized and learned during training.

2.3. Decoding methods

During inference, we typically use beam search to generate our sentences. It improves the generation by exploring multiple sequences at the same time, that usually helps to find better sentences than greedy search. However, both of these methods tend to generate repetitive sequences and produce the most frequent words [7, 8]. In our experiment with beam search, the model only used 398 different words out of 4370 possible words for the model and out of 3516 words in the evaluation subset. This is why we propose to use sampling methods used in text generative models.

The network models the conditional probability $P(y|x)$, where $y = (y_0, ..., y_n)$ and x the audio embedding input. At each step i of greedy decoding, we choose the next word with the highest conditional probability $P(y_i|x, y_{0..(i-1)})$. For beam search, we kept the top b next words for each step until an end-of-sentence token is reached. Once all sequences are finished, the sentence with the highest conditional probability divided by the number of words will be chosen.

For sampling methods, directly sample from the model's probabilities leads to degenerated sentences, so we need to more carefully choose which words can be sampled at each step. All of these 3 methods define a subset of words S from which the next word will be sampled.

Top-k sampling [9] propose to sample only from the k most probables words. k is a hyperparameter between 1 and the vocabulary size.

Nucleus sampling [8] propose to define the subset of words by using each word's probabilities. We only sample a word from the smallest subset whose sum of word probabilities is greater or equal

[2] https://zenodo.org/record/3987831

N°	System	Hparams	B1	B2	B3	B4	METEOR	ROUGE-L	CIDEr	SPICE	SPIDEr	Vocab
-	Baseline	$b = 4$	**0.555**	**0.358**	0.239	0.156	0.164	0.364	0.358	0.109	0.233	-
-	Greedy search	$b = 1$	0.542	0.335	0.213	0.132	0.163	0.364	0.328	0.113	0.220	463
1	Beam search	$b = 9$	**0.555**	0.357	**0.240**	**0.157**	**0.170**	**0.374**	**0.367**	**0.118**	**0.242**	395
2	Top-k	$k = 4$	0.488	0.279	0.160	0.085	0.154	0.329	0.241	0.106	0.174	**602**
3	Nucleus	$p = 0.3$	0.532	0.322	0.200	0.121	0.161	0.354	0.303	0.111	0.207	506
4	Typical	$\tau = 0.8$	0.457	0.245	0.135	0.067	0.140	0.310	0.198	0.090	0.144	584

Table 1: Results on the evaluation subset of Clotho v2.1. Higher score is better.

to a hyperparameter p. More formally, the nucleus sampling defines the subset of words S which minimize :

$$\min_{S} \sum_{y \in S} P(y|x, y_{0..(i-1)}) \tag{1}$$

Typical decoding [10] is similar to nucleus sampling by adding a parameter τ that defines a range of conditional probabilities. The subset S is defined by the following equation :

$$\min_{S} \sum_{y \in S} |H(P(.|x, y_{0..(i-1)})) + \log P(y|x, y_{0..(i-1)})| \tag{2}$$

where $H(.)$ denote the entropy and $P(.|x, y_{0..(i-1)})$ the output probabilities of the model at step i.

2.4. Implementation

To optimize our network, we used Adam [11], with learning rate set to 5.10^{-4} at the first epoch, weight decay set to 10^{-6}, β_1 set to 0.9, β_2 set to 0.999 and ϵ set to 10^{-8}. We used a cosine learning rate scheduler with the following rule :

$$\text{lr}_k = \frac{1}{2}\left(1 + \cos\left(\frac{k\pi}{K}\right)\right)\text{lr}_0 \tag{3}$$

with k being the current epoch index, and K the total number of epochs.

The transformer decoder use a main embedding dimension d_{model} set to 256, with the number of attention heads h set to 4, a sequence of 6 standard decoder layers and a dropout P_{drop} set to 0.2. The last affine layer contains 4370 neurons matching the vocabulary size. We also added a label smoothing of 0.1 to reduce overfitting and a gradient clipping of 10 to avoid collapsing during training. The hyperparameters used by decoding methods are in Table 1. The final encoder-decoder model results in 16.5M trainable parameters (4.8M in CNN10 and 11.7M in transformer decoder). Our implementation uses PyTorch [12] and PyTorch-Lightning [13].

3. RESULTS

The results in Table 1 shows our 4 submissions with the standard beam search and the 3 stochastic methods. The sampling reveal to be lower than using beam search. As expected, the vocabulary used is increased in the sampling methods, and we can see a negative relation between SPIDEr and the vocabulary size used. Some generated captions may be syntactically incorrect due to sampling, but it would be interesting to see if these captions can be preferred by humans over the beam search captions.

4. ACKNOWLEDGMENT

We used the OSIRIM platform, administered by IRIT and supported by CNRS, the Region Midi-Pyrénées, the French Government and ERDF (http://osirim.irit.fr/site/en).

5. REFERENCES

[1] E. Labbé, "aac-datasets," v0.1.1. [Online]. Available: https://pypi.org/project/aac-datasets/

[2] K. Drossos, S. Lipping, and T. Virtanen, "Clotho: An Audio Captioning Dataset," *arXiv:1910.09387 [cs, eess]*, Oct. 2019, arXiv: 1910.09387. [Online]. Available: http://arxiv.org/abs/1910.09387

[3] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength Natural Language Processing in Python," 2020.

[4] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, "PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition," *arXiv:1912.10211 [cs, eess]*, Aug. 2020, arXiv: 1912.10211. [Online]. Available: http://arxiv.org/abs/1912.10211

[5] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio Set: An ontology and human-labeled dataset for audio events," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA: IEEE, Mar. 2017, pp. 776–780. [Online]. Available: http://ieeexplore.ieee.org/document/7952261/

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[7] B. Eikema and W. Aziz, "Is map decoding all you need? the inadequacy of the mode in neural machine translation," 2020.

[8] A. Holtzman, J. Buys, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," *CoRR*, vol. abs/1904.09751, 2019. [Online]. Available: http://arxiv.org/abs/1904.09751

[9] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical neural story generation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 889–898. [Online]. Available: https://aclanthology.org/P18-1082

[10] C. Meister, T. Pimentel, G. Wiher, and R. Cotterell, "Typical Decoding for Natural Language Generation,"

arXiv:2202.00666 [cs], Mar. 2022, arXiv: 2202.00666. [Online]. Available: http://arxiv.org/abs/2202.00666

[11] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017, arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *proc. NeurIPS*, 2019, pp. 8026–8037.

[13] W. Falcon and .al, "Pytorch lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019.