# AbstractNet : A generative model for high density inputs

**Boris Musarais**

boris1284@gmail.com
ESGI Paris, France

## ABSTRACT

This paper introduces AbstractNet, a generative model for high density inputs. The model suggests a method that uses unsupervised learning to generate feature maps. The model drastically improves the performances of raw audio generation by reducing the required amount of input data and computing power necessary to achieve a similar result when compared to the state of the art.

**Keywords:** Unsupervised learning, Generative model, audio, Auto-Encoder, LSTM, RNN, deep neural networks, data compression

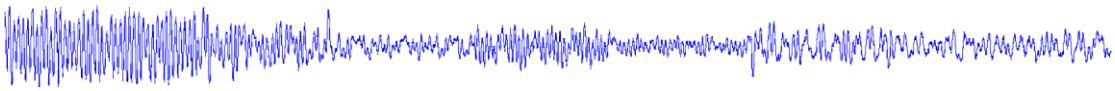

Figure 1: Real-time generated waveform (16000 Hz)

## 1. Introduction:

Among the vast amount of fields that are concerned by the dimensionality problem, audio generation is a very intuitive example. Anyone can understand why feeding such dense data to a neural network would prove itself to be a challenge. A known approach to generating raw audio is to simply give the raw data to a recurrent neural network such as LSTMs [1] and let the model try to guess what the next frame will be as experimented in GRUV [2].The output would be added to the input and fed back to the network. Wavenet [3] suggested another approach, using dilated convolutions [4, 5]:

$$p(x) = \prod_{t=1}^{T} p(x_t \mid x_1, \dots, x_{t-1}) \quad (1)$$

Formula (1) shows that the output has to be fed back to the input making it an autoregressive [6] model. Given the dimensionality of raw audio (up to 44100 samples per second), this approach requires a massive amount of time and computing power to produce raw audio. The larger the receptive field is, the longer it will take to compute.

Because it is so hard [7,8] to train deep neural networks since they struggle to get an abstract overview of things, and need a massive amount of input data to do so, we will try and take a different approach; maybe there is a way to reduce the dimensionality of generative models.

## 2. AbstractNet

Instead of letting the model figure everything out from scratch, we could divide the learning process into two (or more) steps, guiding it so it can get an abstract view faster. As suggested in "*The Sparsely-Gated Mixture-of-Experts Layer*" [9] approach, sub-networks offer a solid approach to reducing the computing dimensionality of a dataset. Other works [10] also suggest using auto-encoders [11] as an effective optimization solution.

The ideal approach would be to generalize our model, making no assumptions about the input shape (other than the fact that is it defined by recurrent patterns).
Here is how we will do this:

- Let the network find low level features in a sample of our dataset that represents the entire population using unsupervised learning (more specifically auto-encoders)
- Compress the entire dataset using those features.
- Generate a high level feature map using a recurrent neural network that defines how lower features are to be distributed
- Generate a dense, low level input from the feature map using a specialized conditional generative model.

Considering some raw input data (an audio sample or anything else), this is what the model looks like:

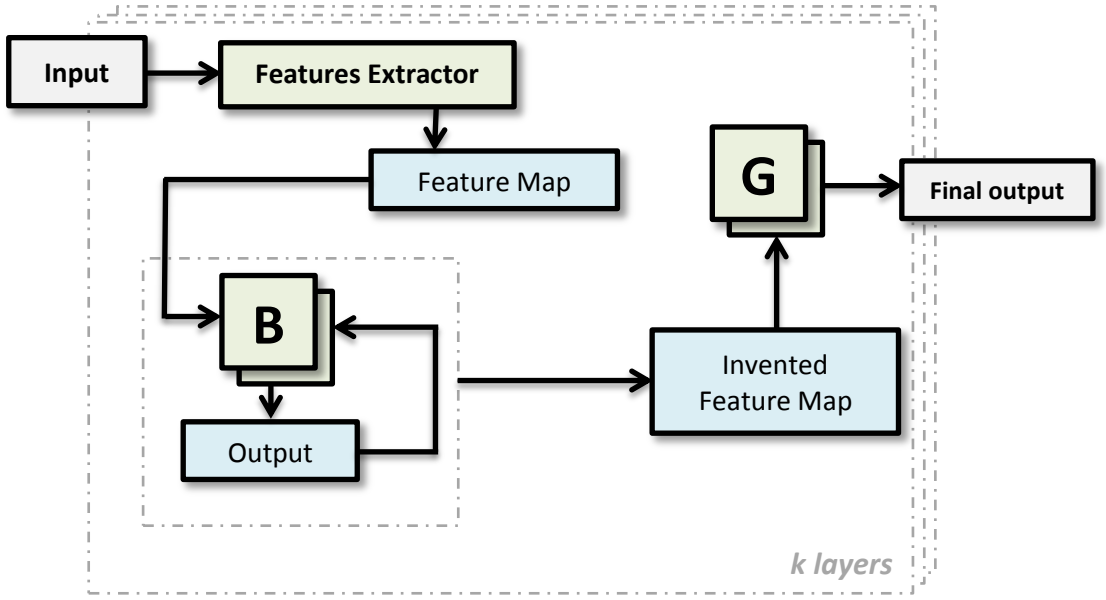


Figure 2: Architecture of the AbstractNet generation model

**G:** Generative model, used to rebuild low level signals from feature maps.
**B:** Recurrent generative model, used on a feature map. In the tests I used LSTMS [12].

This model can be stacked to go deeper; each layer is then conditioned on the previous one, allowing the system to understand more abstract concepts without the need of large datasets or great computing power!

- *Features Extraction:*

One of the key elements of AbstractNet is feature definition, this can be done programmatically (ex: amplitude and frequency are simple features that can be extracted manually), but the ideal setup is to have an unsupervised neural network extract the features from the low level raw input. The idea behind this approach is that a few samples of the large scale input data are enough to describe the low level features of a dataset. This means one should provide to the feature extractor at least one example for each feature present in the entire dataset (Obviously It is recommended to provide more than one).
The examples are then converted to a feature map by the auto-encoder.

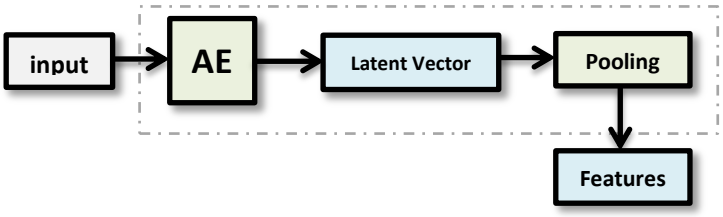


Figure 3: Architecture of the features extractor model

**AE**: Auto-encoder
**X1**: Resampled feature map.

A feature map is represented as a set of features for each frame in the raw waveform:

$$x = \{\{k_0, \dots, k_u\}_{[\![0;N]\!]}\} \tag{2}$$

**N:** Feature map length
**k:** a feature
**u:** the amount of features (hidden units) defined in the auto-encoder

The feature map is then fed to a neural network whose task is solely to train on multi-dimensional feature maps. To control the output, the network is conditioned by the feature channel:

$$\hat{x}(c) = \{x, c\} \tag{3}$$

**c**: desired feature channel

As a matter of fact, we don't have to use the decoding part of the auto-encoder. Using a conditional [13] network can drastically improve the quality of the generated low level output; the de-noising aspect of the auto encoder is not really required.

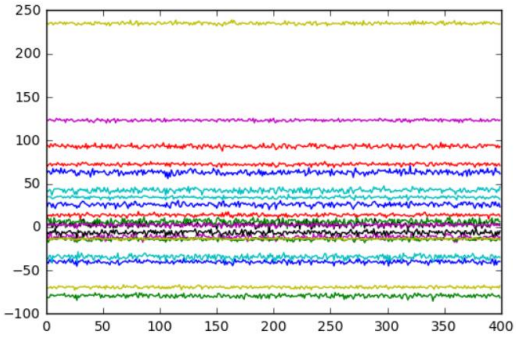


Figure 4: A 20 channels generated feature map, in practice it is normalized.

- *Entropy pooling:*

To improve the receptive field and increase the coherence and flexibility of the generator, an entropy [14] map is injected in the input to condition the high level generator. This allows the network to have a higher level (thus more abstract) view.

$$\mathrm{E(k)} = \sum_{n=1}^{N} \log x^{k+n} \tag{4}$$

**E:** entropy of a given sample
**k**: sample start index
**n:** sample length
**x:** input data

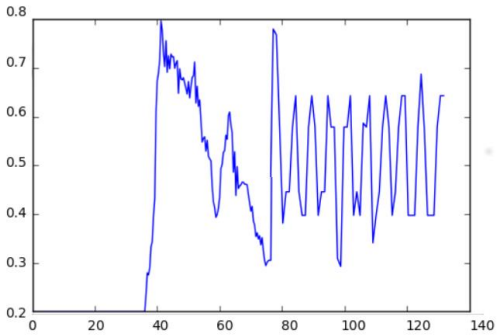


Figure 5: An input sample with its entropy map one the left side, and the raw audio sample on the right side.

- *Quantization*

In this generative model, previous outputs are fed back to the new input, this means that the error rate is subject to an exponential increase:

$$e(n) = \sum_{k=1}^{n} \varepsilon_k \tag{5}$$

**e**: error rate
**n**: generation iteration
**ε** : measured error

To improve the model's flexibility and its tolerance to unknown patterns, the inputs have been quantized [15], encoding each value to only 256 values:

$$x_n = \frac{\lfloor x_n * u \rfloor}{u} \tag{6}$$

**x**: input data
**u**: quantization (in our case 256)

## 3. Conditional AbstractNets

Just like the higher levels in the AbstractNet architecture control the lower levels output, one can manually inject his own conditions to an AbstractNet model to generate an output with custom characteristics:

$$p(x \mid c) = \prod_{n=1}^{T} p(x_n \mid x_1, \ldots, x_{n-1}, c) \tag{7}$$

**c**: a conditional input, similar to formula (3).

This is typically done for TTS [16, 17]. One could use this technique to define the words he wants the model to generate.

## 4. Experiments

To figure out if AbstractNet really is efficient at generating coherent high density signals, the best way is to test it and evaluate the quality of the generated output.

- Piano dataset:

  The generative model was trained on piano samples taken from DJ Oakawari's songs. The encoder was trained on small samples extracted from the dataset; the training phase took 3 hours on an **NVidia GTX860M** GPU.
  For this test, a single AbstractNet layer was used. The model was implemented with Tensorflow. To improve the compression performances, I used Fast Fourier Transforms on the input signals; this allowed the encoder to ignore the time dimensionality. The decoder has been conditioned on previous waveforms to regenerate an audio sample when combined with a

latent vector. The latent feature map for this test was **40** times smaller than the raw audio input; the output had a background noise due to the generator not being able to properly rebuild the audio sequence, even when conditioned on the previous samples. Several techniques [18] exist for noise removal, but it wasn't the point of this experiment.
Regardless, the model was able to generate high frequency audio samples (at **16000 Hz**) **in real time**, a very promising result.

- Complex Samples:

  More complex audio samples do not get compressed as effectively, because one sample (for example with dance music) contains more features than piano. This is where a deeper architecture could prove itself more efficient.

- *Activation function*

Because the experiments are not defined as a classification problem, but rather a regression problem; since we want to find the closest approximation of a signal, no specific activation function was used, only the identity function.

- *Error function*

The cross entropy [19] function was used for error minimization during the experiments:

$$H(p,q) = -\sum_{x} p(x) \log q(x)$$

While I do not have the kind of computing power needed to train on video samples or to generate real time TTS, I believe this is a problem AbstractNet is able to approach with decent hardware. More tests have yet to be performed to explore the real potential of this technique; my tests were done on a laptop. The Auto-Encoder approach could also be tested with other techniques, such as Generative Adverserial Networks (GANs) [20], Variational-Auto-Encoders (VAE) [21] or even combined VAE+GAN [22]. I am eager to see the results of other people using the AbstractNet architecture on other datasets with powerful machines and great ideas.

## 5. Conclusion

This paper presented AbstractNet, a generative model designed for dense inputs that takes a higher abstract view on the data, while drastically improving performances. By combining multiple layers of the AbstractNet architecture, the model could go further in its understanding of high level data structures, making the generated output more coherent. Because lower layers are conditional networks, it is possible to use them independently to generate low level features. This interesting behavior allows us to "talk" to the AbstractNet model and ask it to explain why it has behaved in a certain way. This means one can follow and understand the reasoning of an AbstractNet model to make it behave as expected.

## ACKNOWLEDGEMENTS

I want to thank Alain Lioret from Université Paris 8, Aurélien Schlossman from Ariane Group, Nicolas Vidal, Martin Tricaud and everyone at *Ecole Superieure De Génie Informatique* (ESGI). I would also like to thank all the people who believed in this project.

# REFERENCES


[1] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[2] Nayebi, A., & Vitelli, M. (2015). GRUV: Algorithmic Music Generation using Recurrent Neural Networks.

[3] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio.*CoRR abs/1609.03499*.

[4] Dutilleux, P. (1989). An implementation of the "algorithme à trous" to compute the wavelet transform. In *Wavelets* (pp. 298-304). Springer Berlin Heidelberg.

[5] Holschneider, M., Kronland-Martinet, R., Morlet, J., & Tchamitchian, P. (1990). A real-time algorithm for signal analysis with the help of the wavelet transform. In Wavelets (pp. 286-297). Springer Berlin Heidelberg.

[6] Akaike, H. (1969). Fitting autoregressive models for prediction. Annals of the institute of Statistical Mathematics, 21(1), 243-247.

[7] Glorot, X., & Bengio, Y. (2010, May). Understanding the difficulty of training deep feedforward neural networks. In Aistats (Vol. 9, pp. 249-256).

[8] Montana, D. J., & Davis, L. (1989, August). Training Feedforward Neural Networks Using Genetic Algorithms. In IJCAI (Vol. 89, pp. 762-767).

[9] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

[10] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. science, 313(5786), 504-507.

[11] David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams. Learning representations by back-propagating errors

[12] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[13] Adali, T., Liu, X., & Sonmez, M. K. (1997). Conditional distribution learning with neural networks and its application to channel equalization. IEEE Transactions on Signal Processing, 45(4), 1051-1064.

[14] Cox, G. (2010). On the relationship between entropy and meaning in music: An exploration with recurrent neural networks. In *Proceedings of the Annual Meeting of the Cognitive Science Society*.

[15] Ahalt, S. C., Krishnamurthy, A. K., Chen, P., & Melton, D. E. (1990). Competitive learning algorithms for vector quantization. Neural networks, 3(3), 277-290.

[16] Taylor, P. (2009). Text-to-speech synthesis. Cambridge university press.

[17] Ze, H., Senior, A., & Schuster, M. (2013, May). Statistical parametric speech synthesis using deep neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on (pp. 7962-7966). IEEE.

[18] Rudin, L. I., Osher, S., & Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, *60*(1-4), 259-268.

[19] Deng, L. Y. (2006). The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning.

[20] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

[21] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

[22] Larsen, A. B. L., Sønderby, S. K., Larochelle, H., & Winther, O. (2015). Autoencoding beyond pixels using a learned similarity metric. arXiv preprint arXiv:1512.09300.