Vector Quantization of High-Dimensional Speech Spectra Using Deep Neural Network

JianFeng WU†, HuiBin QIN(a), YongZhu HUA†, LiHuan SHAO†, Ji HU†, Nonmembers, and ShengYing YANG†, Member

SUMMARY This paper proposes a deep neural network (DNN) based framework to address the problem of vector quantization (VQ) for high-dimensional data. The main challenge of applying DNN to VQ is how to reduce the binary coding error of the auto-encoder when the distribution of the coding units is far from binary. To address this problem, three fine-tuning methods have been adopted: 1) adding Gaussian noise to the input of the coding layer, 2) forcing the output of the coding layer to be binary, 3) adding a non-binary penalty term to the loss function. These fine-tuning methods have been extensively evaluated on quantizing speech magnitude spectra. The results demonstrated that each of the methods is useful for improving the coding performance. When implemented for quantizing 968-dimensional speech spectra using only 18-bit, the DNN-based VQ framework achieved an averaged PESQ of about 2.09, which is far beyond the capability of conventional VQ methods.

key words: deep neural network, vector quantization, auto-encoder, binary codecs

1. Introduction

Traditional Vector quantization (VQ) methods commonly employ the k-means [1] or Linde-Buzo-Gray (LBG) [2] algorithm for codebook training. These algorithms are generally impractical and inefficient for quantizing high-dimensional data or training large size codebook. Partitioned VQ and Multi-stage VQ methods have been proposed to reduce computational complexity and codebook size [3]. However, such a compromise may lead to a large quantization error.

To address this problem, a deep neural network (DNN) based coding method has been proposed in [4] for VQ of high-dimensional data, which is inspired by the success of DNN in data dimensionality reduction [5], [6]. This method is superior to traditional VQ methods in experiments on speech spectrogram coding. Nevertheless, the authors reported that “the hidden units behave in a fairly binary way for reasonably long windows, but not for short windows”. That is, the distribution of hidden units approximating to binary is only a special case. In practice, when that distribution is far from binary, quantizing the hidden units to binary values would result in a large quantization error.

In order to make the activations of the coding layer as close to binary as possible, Gaussian noise has been added to the input of the coding layer in [7]. With the added noise, the fine-tuning can make the coding layer closer to binary and thus reduce the binary coding error. In addition, it has been shown in [8] that, compared with the method in [7], forcing the coding layer to be binary during the forward pass of the fine-tuning is more efficient while yields comparable performance. Moreover, in sparse auto-encoder, non-zero penalty term has been added to the loss function to force the hidden units to be sparse [9], [10]. Inspired by this concept, we propose adding a non-binary penalty term in the loss function to force the distribution of the hidden units approximate to binary. In the whole, there generally exist three methods which can be employed to make the coding layer to be more binary in fine-tuning: 1) adding Gaussian noise to the input of the coding layer, 2) forcing the output of the coding layer to be binary, 3) adding a non-binary penalty term to the loss function.

In this work, we utilize DNN for VQ of high-dimensional data. The main challenge of applying DNN to VQ is how to get binary units in the coding layer in the fine-tuning procedure. We incorporate the three fine-tuning methods mentioned above into the DNN-based VQ framework to reduce the coding error. Experiments on speech magnitude spectra have been conducted to evaluate the performance of the new method. The results showed that each of these binarization fine-tuning methods has the capability to achieve a smaller quantization error compared with traditional fine-tuning methods. Furthermore, using the combined binarization methods in fine-tuning, the coding error can be further reduced significantly.

2. Deep Auto-Encoder with Binary Coding Layer

2.1 Training Stacked Auto-Encoders

An auto-encoder maps an input vector $x$ to a hidden representation $y$ using a non-linear mapping function $f_{\theta}(x)$, with $\theta = [W, b]$. Then, the resulting hidden representation $y$ is mapped back to a reconstructed vector $z$ using reverse function $f_{\theta'}(y)$, with $\theta' = [W^T, b']$. The parameters of this model are optimized by minimizing the average reconstruction error over the training set:

$$\theta^*, \theta'^* = \arg\min_{\theta, \theta'} \frac{1}{n} \sum_{i=1}^{n} E(x^{(i)}, z^{(i)})$$  \hspace{1cm} (1)

where $z^{(i)} = f_{\theta'}(f_{\theta}(x^{(i)}))$, and $E$ is a loss function that can be
measured by square error:

$$E(x, z) = \sum_{k=1}^{N} (x_k - z_k)^2$$  \hspace{1cm} (2)

A stacked auto-encoder (SAE) is a deep neural network that consists of multiple layers of auto-encoders, where the output of each layer is wired to the inputs of the successive layer. A popular method to train the stacked auto-encoder is to use greedy layer-wise training, which is illustrated in Fig. 1.

The training procedure is as follow:
1) Train the first layer on the raw input as a basic auto-encoder to obtain the parameters, $W_1$, $b_1$, and $b_1'$.
2) Iteratively train the successive layer on the hidden units’ output of previous auto-encoder to obtain the others parameters, $W_j$, $b_j$, $b_j'$, $j = \{2, 3, 4\}$.
3) Finally, “unroll” the stacked auto-encoders to deep auto-encoder, and fine-tune the deep auto-encoder using back-propagation to make its output as similar as possible to its input.

2.2 Deep Auto-Encoder with Binary Coding Layer

The fine-tuning procedure described in the previous section can make the output as similar to the input, but the coding units may follow an arbitrary distribution. There are three fine-tuning methods to get a more binary coding layer.

2.2.1 Adding Gaussian Noise to the Input of the Coding Layer

This is based on the assumption that the decoder network is insensitive to very small differences in the output of the coding layer. With the added noise, the output of the coding layer will reach the boundary of the activation function, e.g., 0 and 1 for a sigmoid function. In practice, a fixed Gaussian noise with zero mean and variance $\sigma^2$ is added to the input of the coding layer. In particular, the added noise is fixed in advance and does not change during training.

2.2.2 Forcing the Output of the Coding Layer to Be Binary

The second method is rounding the output of the coding layer to be binary during the forward pass of the back-propagation in fine-tuning, but ignore the rounding process during the backward pass. This method is simpler than the previous one since there is no need to cross-validate the variance $\sigma^2$ of the added noise. It has been shown in [8] that, in image retrieval, this method can achieve comparable performance as the method of adding Gaussian noise.

2.2.3 Adding Penalty Term to the Output of the Coding Layer

The third method is inspired by the sparse auto-encoder, which achieved a more zero hidden units by adding a non-zero penalty term in the loss function. We expect to get a more binary coding layer by adding a non-binary penalty term in the loss function. Intuitively, any hat-like functions, such as Gaussian function $G(x)$ or negative square function $S(x; a, b) = -(x - a)^2 + b$, can be used as a penalty function. The left of Fig. 2 shows the two penalty functions in details. The shape of $G(x)$ can be adjust by the variance, $\sigma^2$, whereas square function $S(x)$ can’t. We also study the deviation of the two penalty functions, since the back-propagation algorithm requires the deviation of the loss function. The deviation $S'(x)$ is a straight line, and the $G'(x)$ is a curved line when the variance is 0.25. Yet, as the variance $\sigma^2$ increased, $G'(x)$ will be approximated to a straight line for $x \in [0, 1]$.

In this paper, we adopt the negative square function $S(x)$ as the penalty term. With the penalty term, the loss function for auto-encoder is defined as: $L(x, z) = E(x, z) + \alpha P(x)$, where $E(x, z)$ is defined in (2), $\alpha$ controls the penalty weight, and the penalty term $P(x)$ is calculated by the output of the coding layer using the penalty function $S(x)$.

Compared with the coding layer with arbitrary values, the coding layer with binary constrained values would result in a smaller coding error. In the following section, we evaluate these binarization methods on VQ of high-dimensional speech spectra.

3. Spectrum Vector Quantization of Speech Spectra

The proposed DNN-based VQ framework can be applied to compression of various kinds of high-dimensional data, such as image, video, audio, etc. As an example, this section introduces VQ of the speech signal in the frequency domain using the deep auto-encoder. It should be noticed that we only take account of quantizing the speech magnitude spectra, as the magnitude is more important than a phase.
thus, the phase spectra are un-quantized.

The diagram of speech spectra VQ is shown in Fig. 3. In the encoder, the speech signal in the time domain is framed and transformed into the frequency domain. The log-power magnitude spectra are normalized and coded by the deep auto-encoder trained in advance. In the decoder, the log-power magnitude spectra are decoded by that deep auto-encoder. And then, a de-normalization process is applied to get the actual data. After that, an inverse transform is applied to get the time domain signal. Finally, an overlap-add method is applied to synthesize the waveform of the speech signal.

For the output of the deep auto-encoder, we utilize the log spectral distortion (LSD in dB) to examine the coding error. For the final speech signal, we utilize segmental signal to noise ratio (SegSNR in dB) and perceptual evaluation of speech quality (PESQ) to assess the reconstructed speech quality. The more details of the evaluation will be introduced in the next section.

4. Evaluation

We use the TIMIT database for training and testing. About 3 hours of speech data in the training set are used as training data, and about 30 minutes of speech data in the test set are used as validation data and test data, respectively. These speech data are down-sampled to 8 kHz and framed to 240 samples with a Hamming window. The frameshift is 120 samples (50% overlapped). A short-time Fourier analysis is used to compute the DFT of each overlapped frame.

Then, the 121 dimensions log-power spectra (the half of the conjugate symmetric component) are used to train the deep auto-encoder.

In our experiment, the deep auto-encoder is initialized with SAE and fine-tuned with the back-propagation algorithm. The details of the training procedures are discussed in Sect. 2.1. The datasets are divided into small “mini-batches” of 100 cases. In the pre-training, the number of epochs for each stacked auto-encoders is set to 20. The learning rate for the first auto-encoder is set to 0.002 and 0.05 for others. In the fine-tuning, the learning rate starts with 0.002 and decayed by a factor of 0.9 when the decrease of validation error between two consecutive epochs falls below 0.05%. The momentum starts at 0.5 and increased to 0.9 after 15 epochs. The fine-tuning is terminated when the validation error fails to decrease by 0.01% between two successive epochs.

4.1 Evaluation of the Deep Auto-Encoder with Different Binarization Methods

In this evaluation, we use 8 frames of speech spectra to construct an input vector, whose dimension is 968 (121 times 8). The architecture of the deep auto-encoder is 968-2000-1000-500-288-500-1000-2000-968, where the coding layer is 288. We use the three binarization methods discussed in Sect. 2.2 to fine-tune the deep auto-encoder: adding Gaussian noise to the input of the coding layer (denote as Add-Noise), forcing the output of the coding layer to be binary (denote as Binary), and adding penalty term to the output of the coding layer (denote as Add-Penalty). Besides, the deep auto-encoder without any binarization method (denote as None), is also trained as a baseline.

Table 1 shows the averaged LSD, SegSNR and PESQ results on the test set using the different binarization methods. For the Add-Noise and Add-Penalty methods, the standard deviation $\sigma$ and penalty weight $\alpha$ are chosen by cross-validation, respectively. “Binary” means the coding units in the test are quantized to 0 or 1 with threshold 0.5, while “Real” means the coding units are the real floating-point values. The results in Table 1 show that all these binarization methods can effectively improve the binary coding performance. Compared to the baseline, the LSD is decreased from 29.03 dB to 8.98 dB, and the SegSNR is increased from $-9.03$ dB to 4.90 dB when the FB method is employed. The PESQ is increased from 1.16 to 2.90 when the Add-Noise method is employed. The Add-Penalty method achieves slight performance improvement. Nevertheless, the results in Table 1 indicate that the performance of each binarization methods deteriorated in the “Real” case compared with the baseline. This is due to the fact that these binarization methods are designed for reducing the “Binary” coding error, rather than reducing the “Real” coding error.

Moreover, we have tried some combination of different binarization methods, the results are shown in Table 2. The “None & Add-Noise” denote for firstly there is no binarization method applied (None) and then apply the AN

| Binarization Method | Coding Layer | LSD  | SegSNR | PESQ  |
|---------------------|--------------|------|--------|-------|
| Add-Noise ($\sigma=0.3$) | Binary       | 8.98 | 3.37   | 2.90  |
|                      | Real         | 6.88 | 2.70   | 3.40  |
| Binary              | Binary       | 6.64 | 4.90   | 2.69  |
|                      | Real         | 7.62 | 1.91   | 2.63  |
| Add-Penalty ($\alpha=0.3$) | Binary     | 15.99 | -8.23  | 1.80  |
|                      | Real         | 4.47 | 10.19  | 3.12  |
| None (Baseline)     | Binary       | 29.03 | -9.03  | 1.16  |
|                      | Real         | 4.08 | 12.00  | 3.92  |
Table 2  Results of combination of different binarization methods

| Binarization Method | LSD | SegSNR | PESQ |
|---------------------|-----|--------|------|
| None & Add-Noise    | 7.23| 5.86   | 2.86 |
| None & Binary       | 6.15| 1.86   | 3.03 |
| Add-Noise & Binary  | 5.91| 6.44   | 3.17 |
| Binary & Add-Noise  | 6.80| 3.61   | 2.98 |

Fig. 4  Results of deep auto-encoder using various binary coding units

method, the “Add-Noise & Binary” denote for firstly apply Add-Noise method and then apply the Binary method, and so on. For the Add-Noise method, the parameter $\sigma$ is set to 0.3. We get the best binary coding performance with a combination of Add-Noise and Binary methods, the LSD is 5.91 dB, the SegSNR is 6.44 dB, and PESQ is 3.17. The Add-Penalty method’s results are not listed in Table 2, due to the performance is almost equal to the “None” case. Generally speaking, the combined methods are superior to single methods. The combined methods can be considered that the preceding method is an initializer of the following method. For example, the Binary method in Table 1 only get a PESQ score of 2.69. Yet, with Add-Noise method as an initializer, the “Add-Noise & Binary” method in Table 2 get a PESQ score of 3.17.

4.2 Deep Auto-Encoder with Less Coding Units

In this evaluation, we use deep auto-encoder with less coding units to quantize the speech spectra. The architecture of the deep auto-encoders is 968-2000-1000-500-xxx-500-1000-2000-968, where the coding layer xxx = 144, 72, 36, 18. That is, the 968-dimensional floating-point vector will be quantized to binary vector of 144-bit, 72-bit, 36-bit and 18-bit, respectively. Also, we utilize those binarization methods to reduce the binary coding error. Again, the results demonstrated that applying the combined binarization methods can improve the coding performance significantly. Theoretically, with decrease of the coding bits, the coding error will increased. This is demonstrated in Fig. 4: when the coding units decreased from 288 to 18, the LSD increased from 5.91 dB to 8.92 dB, the SegSNR decreased from 6.44 dB to 2.9 dB, and the PESQ score of the reconstructed speech is up to 2.09. Although there are some annoying artifacts, the speech signal is roughly intelligible. For the conventional VQ methods, it is impracticable to quantize 968-dimensional data using only 18-bit.

5. Conclusion

This paper presented a DNN-based VQ framework to address the challenging problem of quantizing high-dimensional data. The problem of applying DNN to VQ is how to get binary units in the central coding layer, for there will be large quantization error in the case of the distribution of coding units is far from binary. We have studied three fine-tuning methods to reduce the binary coding error. Evaluation results on VQ of speech magnitude spectra showed that each of the three binarization methods can significantly improve the coding performance. Moreover, with the combined binarization method, the coding error was reduced considerably. Using the DNN-based VQ framework with these binarization methods, we have implemented VQ of 968-dimensional speech spectra using only 18-bit, and the speech signal was roughly intelligible.

References

[1] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol.1, no.14, 1967.
[2] Y. Linde, A. Buzo, and R. Gray, “An algorithm for vector quantizer design,” IEEE Trans. Commun., vol.28, no.1, pp.84–95, 1980.
[3] R. Gray, “Vector quantization,” IEEE Trans. Acoust., Speech, Signal Process. Mag., vol.1, no.2, pp.4–29, 1984.
[4] Deng, Li, et al., “Binary coding of speech spectrograms using a deep auto-encoder,” Interspeech, 2010.
[5] G.E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” Neural computation, vol.18, no.7, pp.1527–1554, 2006.
[6] G.E. Hinton and R.R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” Science, vol.313, no.5786, pp.504–507, 2006.
[7] R. Salakhutdinov and G. Hinton, “Semantic hashing,” International Journal of Approximate Reasoning, vol.50, no.7, pp.969–978, 2009.
[8] A. Krizhevsky and G.E. Hinton, “Using very deep autoencoders for content-based image retrieval,” ESANN, 2011.
[9] Y. Bengio, “Learning deep architectures for AI,” Foundations and trends® in Machine Learning, vol.2, no.1, pp.1–127, 2009.
[10] Y. Bourou and Y.L. Cun, “Sparse feature learning for deep belief networks,” Advances in neural information processing systems, 2008.