VQ-VAE EXPLAINER: Learn Vector-Quantized Variational Autoencoders with Interactive Visualization

Donald Bertucci
School of Interactive Computing
Georgia Institute of Technology
Atlanta, USA
bertucci@gatech.edu

Duen Horng (Polo) Chau
School of Computational Science & Engineering
Georgia Institute of Technology
Atlanta, USA
polo@gatech.edu

***Abstract*—VQ-VAE EXPLAINER is a Vector-Quantized Variational Autoencoder (VQ-VAE) running live in the browser. VQ-VAE EXPLAINER aims to explain how VQ-VAEs work by connecting their implementation to interactive visuals. VQ-VAE EXPLAINER first displays the main idea behind the VQ-VAE in a summary view. Then, users can drill down into the quantization implementation with code and matrix visualizations. The VQ-VAE EXPLAINER site is live for anyone to use at https://xnought.github.io/vq-vae-explainer/ and the code is open source at https://github.com/xnought/vq-vae-explainer/.**

***Index Terms*—visualization, machine learning,**

I. INTRODUCTION

Vector-Quantized Variational Autoencoders (VQ-VAEs) [1] are a powerful variant of Autoencoders that have been used for photo realistic image generation [2], self-driving compression [3], 3D encoded protein sequences [4], and more! Although conceptually simple, the VQ-VAE implementation is not as simple to understand.

Luckily, existing explanations help people implement VQ-VAEs (like Keras Code Examples [5]). These examples are extremely valuable, but still do not adequately connect the dense Python implementation to the high-level concepts.

We kill two birds with one stone by connecting the actual model code to interactive visual explanation. VQ-VAE EXPLAINER specifically leverages the idea that interactive visualizations are extremely effective at helping people learn ideas in Machine Learning [6]–[10]. And while there are existing Autoencoder and Variational Autoencoder (VAE) interactive ML visualizations [11]–[13], there doesn't exist any interactive explanations for VQ-VAEs.

With VQ-VAE EXPLAINER we specifically visualize what occurs in the hidden layer / latent space. As you can see in Figure 1, a data input on the left ultimately is trained to reconstruct the data output on the right. In this case MNIST Digits [14] and specifically a hand draw "0" in Figure 1.

Within the hidden layer, we directly show the quantiazation process. In Figure 1 where the cursor is, you can see the continuous encoded features from the encoder. These continuous vectors are converted to discrete representations as shown with the colors in the middle of Figure 1. This quantization process works by finding the closest codebook vector (discrete code) and decoding those vectors into the output as shown in Figure 1.

When the user understands the high-level summary, they can click to view the implementation. We show the model code first on the left in Figure 2. Each piece of code corresponds to an operation between matrices. We show the data directly with matrix shapes as shown in Figure 2. Specifically, we display reshaping the features, finding each closest distance to the codebook vectors, and selecting the vectors with the closest distance.

To finish off this introduction, we present the following contributions:

- Summary view: an interactive version of the original VQ-VAE paper's explanation figure (Section II). See Figure 1 for a preview.
- Details view: directly linking code to VQ-VAE operations shown visually (Section III). See Figure 2 for a preview.
- Open source implementation[1] and live website[2] for anyone with a browser to use (Section IV).

II. SUMMARY VIEW

The VQ-VAE EXPLAINER summary view directly adapts the summary figure from the original VQ-VAE paper [1] into a live interactive version. Let me explain the interactive components in reference to Figure 1:

1) Left side (input): users can draw a digit and the model will update in real-time and update the reconstruction on the right.
2) The left grey cube (features): the encoded features from a Convolutional Neural Network encoder. A user can hover over the face of the cube to select a vector across the filter dimension.

[1]Code: https://github.com/xnought/vq-vae-explainer/
[2]Site: https://xnought.github.io/vq-vae-explainer/

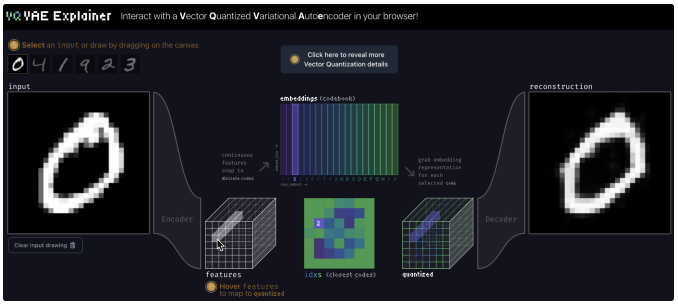

Fig. 1. The VQ-VAE EXPLAINER summary view: input, quantization, and reconstruction. A user can hover over data and see the mapping from features to quantized. In this example, we hover over a feature vector which reveals the closest embedding vector in purple (index of 2) which then acts as the quantized code.

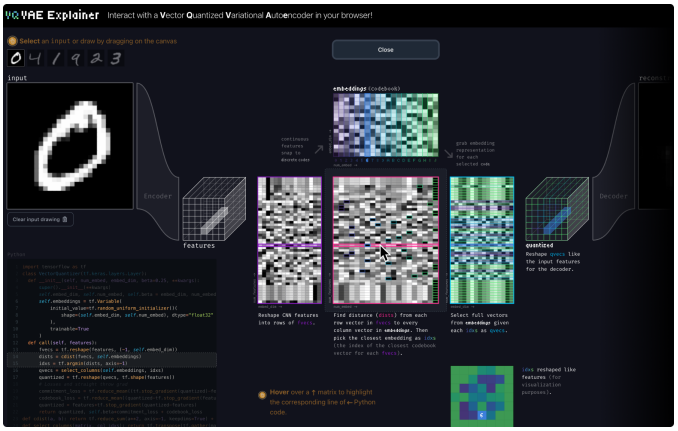

Fig. 2. The VQ-VAE EXPLAINER details view: model code visually linked to live data and quantization steps. A user can hover over a matrix to show vector mappings and highlight which piece of code it corresponds to.

3) The middle codebook (embeddings): the pool of discrete indexes which each continuous feature vector snaps on to (finds closest vector).
4) The middle matrix (idxs): the quantized codebook indexes that corresponds to each feature vector's closest embedding. A user can hover to show the corresponding vector quantization.
5) The right colored cube (quantized): uses the indexes from the previous step to select out entire vectors from the codebook and is input into the decoder. A user can also hover over vectors here to show the full mapping.

III. DETAILS VIEW

Going beyond a summary, a user can click the "Reveal more Vector Quantizer details" button as shown in Figure 2. These lower-level details connect the actual model code to data visualization.

We break up the quantization into a few steps: reshaping the CNN features, finding distances to each embedding vector, picking the index of the embedding with the closest distance, using the index to grab embedding vectors, then reshaping back to the CNN feature dimensions. Within Figure 2, the purple matrix shows the reshape, the pink shows the distances

Fig. 3. The details view after drawing a "0" in the input (left) over time. See how the middle data matrices change based on the input.

from each feature vector to each embedding vector where each closest index is highlighted (argmin), the blue matrix shows the embedding vector per closest code, and finally we reshape back into a cube to be decoded. Each value within the matrices are real numbers colored from negative (black) to positive (white) normalized by each row.

Each step is directly linked to the actual code in the bottom left Figure 2. When a user hovers over a line of code, the corresponding matrix is highlighted with a grey box. When a user hovers over a matrix, the code is highlighted with a grey box. In other words, we link the code to the visualizations and vice versa.

The intermediate data from the quantization operations update in real-time. As shown in Figure 3, as a user draws the digit "0," the data in the matrices change which represents the real data in the quantization process. Just with live interaction a user can see how distances are computed and how discrete codes are selected.

IV. IMPLEMENTATION

The VQ-VAE EXPLAINER interactive visualizations were implemented in the browser with Svelte[3]. The model was loaded with TensorflowJS [15]. We specifically trained the Tensorflow/Keras [16], [17] model with code modified from [5] in a notebook[4]. Then we exported the encoder and decoder

[3] https://svelte.dev/
[4] https://colab.research.google.com/drive/1Dt6ngF_J50RUxfe7ZZYf-GBRORv7Husy

as Tensorflow graph models and converted to the TensorflowJS format. We exported the embedding matrix separately and reimplemented the quantization step within JavaScript.

The VQ-VAE was specifically trained with the MNIST Digits Train Dataset [14] (60k digits) and normalized between [0, 1]. The encoder was a Convolutional Neural Network and the decoder was the opposite operations (Convolution Transposes). The Quantization layer used an embedding dimension of 16 and had 20 embeddings in total. For training we used the AdamW optimizer [18] with default parameters for 30 epochs, a batch size 128, and a quantization beta β of 0.75.

V. Conclusion

We presented VQ-VAE Explainer, an interactive way to connect the VQ-VAE implementation to concrete visualization. The code is open source at https://github.com/xnought/vq-vae-explainer and the live demo is at https://xnought.github.io/vq-vae-explainer/ for anyone to use.

In the future, it is worth deeply studying the idea of including code directly in interactive explainables. Without the code, one naturally asks if any existing interactive visualizations actually help people implement and use the concepts they learn.

References

[1] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[2] A. Razavi, A. Van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," *Advances in neural information processing systems*, vol. 32, 2019.

[3] comma.ai, "commavq: a dataset of tokenized driving video and a GPT model," Jun. 2023. [Online]. Available: https://github.com/commaai/commavq/

[4] M. van Kempen, S. S. Kim, C. Tumescheit, M. Mirdita, C. L. Gilchrist, J. Söding, and M. Steinegger, "Foldseek: fast and accurate protein structure search," *Biorxiv*, pp. 2022–02, 2022.

[5] S. Paul, "Vector-Quantized Variational Autoencoders." [Online]. Available: https://keras.io/examples/generative/vq_vae/

[6] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng, and D. H. P. Chau, "CNN explainer: learning convolutional neural networks with interactive visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1396–1406, 2020.

[7] S. Lee, B. Hoover, H. Strobelt, Z. J. Wang, S. Peng, A. Wright, K. Li, H. Park, H. Yang, and D. H. Chau, "Diffusion explainer: Visual explanation for text-to-image stable diffusion," *arXiv preprint arXiv:2305.03509*, 2023.

[8] A. Cho, G. C. Kim, A. Karpekov, A. Helbling, Z. J. Wang, S. Lee, B. Hoover, and D. H. Chau, "Transformer Explainer: Interactive Learning of Text-Generative Models," *arXiv preprint arXiv:2408.04619*, 2024.

[9] M. Kahng, N. Thorat, D. H. Chau, F. B. Viégas, and M. Wattenberg, "Gan lab: Understanding complex deep generative models using interactive visual experimentation," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 310–320, 2018.

[10] D. Smilkov, S. Carter, D. Sculley, F. B. Viégas, and M. Wattenberg, "Direct-manipulation visualization of deep networks," *arXiv preprint arXiv:1708.03788*, 2017.

[11] S. Rezaei, "variational autoencoder interactive demos with deeplearn.js." [Online]. Available: https://www.siarez.com/projects/variational-autoencoder

[12] D. Bertucci and A. Endert, "VAE Explainer: Supplement Learning Variational Autoencoders with Interactive Visualization," *arXiv preprint arXiv:2409.09011*, 2024.

[13] T. Youtube, "Visualizing autoencoders - Made with TensorFlow.js." [Online]. Available: https://www.youtube.com/watch?v=pQv_lswBeZw

[14] L. Yann, "The mnist database of handwritten digits," *R*, 1998.

[15] Google, "TensorflowJS," 2018. [Online]. Available: https://github.com/tensorflow/tfjs

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[17] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras

[18] I. Loshchilov, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.