Training an auto-regressive model over discrete latent embeddings for the purpose of winged horse image generation

Anonymous author

Abstract

This paper focuses on the generation of white pegasuses through the use of VQ-VAE and PixelCNN architectures. For such a task, attaining latent mappings that are high in information and can be easily manipulated is essential. Well known for their accurate image reconstructions, VQ-VAEs discretise the latent space and greatly reduce the number of data points utilised. However, the use of a uniform prior can lead to poor sampling with the purpose of image generation. The proposed model seeks to overcome this through learning an auto-regressive distribution over a subset of the learnt latent embeddings. The latter is achieved with use of a PixelCNN.

1 Methodology

1.1 Variational AutoEncoders

Traditional Autoencoders take an input $\boldsymbol{x} \sim p(X)$, feed this into to an encoder to get a compressed latent representation $\boldsymbol{z}$ and then build a reconstruction $\hat{\boldsymbol{x}} \approx \boldsymbol{x}$ with use of a decoder. While effective with respect to compression such models do not guarantee known structure in the latent space. This makes its traversal unpredictable. Variational Autoencoders [3], on the other hand, impose a prior distribution $p(\boldsymbol{z})$ over the latent space. This is usually defined as $\boldsymbol{z} \sim \mathcal{N}(0, I)$[4]. The encoder then approximates the posterior distribution $p(\boldsymbol{z}|\boldsymbol{x})$ while the decoder approximates the likelihood $q(\boldsymbol{x}|\boldsymbol{z})$. This imposition of $p(\boldsymbol{z})$ regularises the latent space, facilitating controlled sampling. Optimisation requires maximising the likelihood (minimising reconstruction error), and minimising the divergence of the approximated posterior from the true posterior (the regularisation error). This gives the loss function:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{regular}} + \mathcal{L}_{\text{recon}} \tag{1}$$

where

$$\mathcal{L}_{\text{regular}} = D_{\text{KL}}(p(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) \qquad \mathcal{L}_{\text{recon}} = -\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}[\log q(\boldsymbol{x}|\boldsymbol{z})] \tag{2}$$

In addition to this, the process of sampling $\boldsymbol{z}$ from the posterior distribution $p(\boldsymbol{z}|\boldsymbol{x})$ is discontinuous and so has no gradient [2]. To overcome this problem the 'reparameterisation trick' is used. Given the mean μ and variance σ of the approximated posterior, outputted by the encoder, the sample $\boldsymbol{z}$ can be formulated as

$$z = \mu + \sigma \odot \epsilon \qquad \epsilon \sim \mathcal{N}(0, 1) \tag{3}$$

This enables back propagation through the values of mean and variance, to update the encoder.

1.2 Vector Quantized Variational AutoEncoders

Despite the benefits of structure in the latent space that come with VAE's, the lack in complexity of the prior means they can struggle with fine details and so appear blurry. VQ-VAE's [5] help to overcome this by discretising the latent space. An embedding table,

made up of K embedding vectors $\boldsymbol{e}_i$ of size D, is used to store the most important latent variables, where $1 \leq i \leq K$. This means that many similar points found in the latent space need only be represented by one embedding vector $\boldsymbol{e}_i$. For the input $\boldsymbol{x}$ the encoder outputs L vectors such that $\boldsymbol{z}_e(x) \in R^{L \times D}$. Instead of feeding these directly to the decoder a nearest neighbours lookup [5] is used to identify the embedding table indices, z, of the L embedding vectors, $z_q(x)$, that represent $\boldsymbol{z}_e(x)$ best. For simplicity it is assumed that $L = 1$ here:

$$q(z = k|x) = \begin{cases} 1 & \text{k} = \text{argmin}_j \left\| z_e(x) - e_j \right\|_2 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The decoder then uses $z_q(x)$ to reconstruct the input $\boldsymbol{x}$. Unlike in VAE's a uniform prior is imposed, therefore, all latent variables $\boldsymbol{z}$ are equally likely to be sampled. It can be shown that this means that the KL divergence of the posterior has a constant value of $\log K$. As a result, the regularisation error can be ignored and only the reconstruction error must be minimised.

In addition to this the correct embedding vectors must be learnt. This is achieved via vector quantisation, iteratively moving embedding vectors $\boldsymbol{e}_i$ towards the outputs of the encoder $\boldsymbol{z}_e(x)$. Moving embeddings toward vectors outputted by the encoder that will then later be mapped to a different region of the latent space is undesirable. Therefore, a commitment loss, parameterised by β, is also added. In total this gives the loss function:

$$\mathcal{L}_{\text{VQ-VAE}} = \log p(x|z_q(x)) + \|sg[z_e(x)] - e\|_2^2 + \|z_e(x) - sg[e]\|_2^2 \tag{5}$$

The symbol sg indicating that error is not backpropagated through the enclosed parameters.

1.3 PixelCNN

Every pixel $\boldsymbol{x}_i$ in an image can be seen as a value sampled from a categorical multinomial distribution of 256 values, that is defined over all possible images. Each pixel value is highly dependent on it's context and the values of all proceeding pixels in the image(those to its left and above). Therefore, it makes sense to model the distribution of values for a pixel as conditional on those that come before it. This allows the whole image to be described as:

$$p(\boldsymbol{x}) = \prod_{i=1}^{n^2} p(\boldsymbol{x}_i | \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{i-1}) \tag{6}$$

where the index i specifies pixels from left to right, row by row. PixelCNN [6] replicates this with the use of masked convolutional layers, with the output value of a pixel $\boldsymbol{x}_i$ in each layer only being dependent on the pixels $\boldsymbol{x}_1 ... \boldsymbol{x}_{i-1}$ in proceeding layers. However, kernels used in convolution do not generally cover the entire image. This means that one pass will only capture:

$$p(\boldsymbol{x}) = \prod_{i=1}^{n^2} p(\boldsymbol{x}_i | \boldsymbol{x}_{i-k}, ..., \boldsymbol{x}_{i-1}) \tag{7}$$

where k is the number of pixels that proceed the central value in the kernel. This can be partially overcome with the use of multiple layers. Each pixel draws information from the corresponding pixel in previous layer and all those pixels before it. However, during training, in the case of the first layer, such a corresponding pixel in the previous layer would belong to the input image itself. This is information that is not present during generation. Therefore, two mask types are used, one for the first layer and one for all other layers, see 1. The outputs of the network are the approximated conditional distributions of each pixel in the input. The cross entropy between the value of the i^{th} pixel in the input, $\boldsymbol{x}_i$, and it's corresponding distribution in the output, $\hat{\boldsymbol{x}}_i$, is used to calculate the loss:

$$\mathcal{L}(\hat{x}_i) = -\hat{x}_{i,x_i} + \log \sum_t^N exp(x_{i,t}) \tag{8}$$

where N is the number of values a pixel can take and $\hat{\boldsymbol{x}}_{i,j}$ is the outputted probability that $\boldsymbol{x}_i = j$.

1.4 Generating White Pegasuses

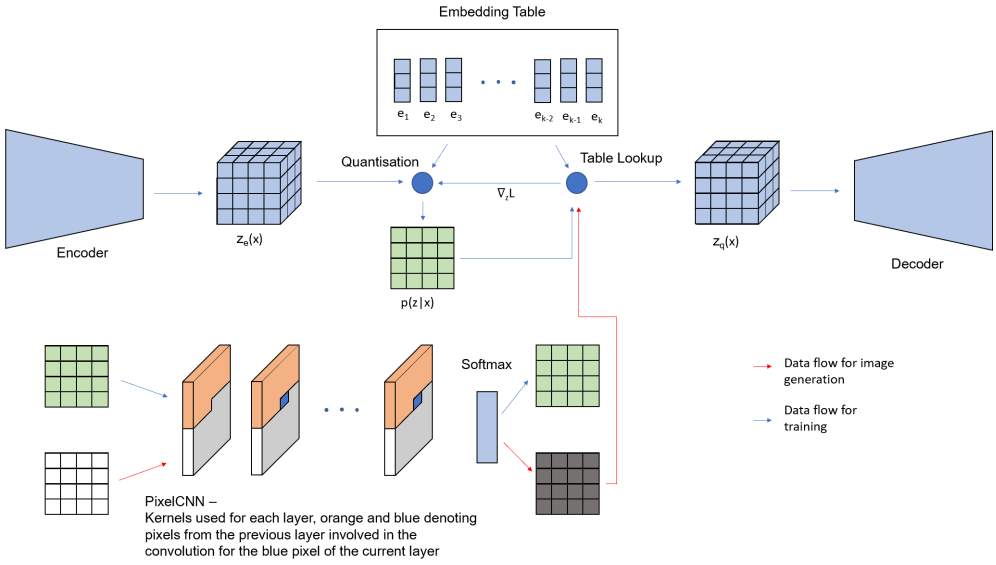

Figure 1: Network Architecture

Although the uniform prior used in the VQ-VAE leads to good reconstructions, it means that random sampling for the purpose of generation is ineffective. However, the categorical nature of the distribution means that a PixelCNN prior can be easily learnt over $\boldsymbol{z}$. To generate white pegasuses specifically requires training on the latent variables of such images, something not immediately possible with the provided data-sets. Interpolation in VQ-VAEs is also known to be poor[1], however, specific input images can still produce good interpolations. Therefore, having trained the VQ-VAE on the entirety of the data set to ensure good encodings, two subsets (one of white birds and the other of white horses) that were thought likely to give good interpolations were selected. The resulting latent variables of the interpolations between these subsets were then used to train the PixelCNN prior. Latent variables sampled from this distribution were then used to retrieve their corresponding vectors in the look-up table. These were then fed into the same decoder used in training for the VQ-VAE. In total this gave the network architecture seen in figure 1.

2 Results

As can be seen in figures 2 and 3 most of the images produced are sharp and of white animals, some of which are pegasuses. However, numerous images are not. In addition to this it can be seen that in some cases generated images are very similar within the batch. This is likely due to the limited number of interpolation points used for training the PixelCNN.

3 Limitations

The ancestral sampling procedure used for generating images via PixelCNN provides very little control with respect to what features are present in the image. This combined with the limited training data has led to a lack in diversity of images generated along with some that don't look like pegasuses. To improve upon this more time could be spent collating images for interpolation and training the PixelCNN. In addition to this, hierarchical models such as that seen in [7] could be explored to facilitate the selection of shapes and textures found in the generated image.

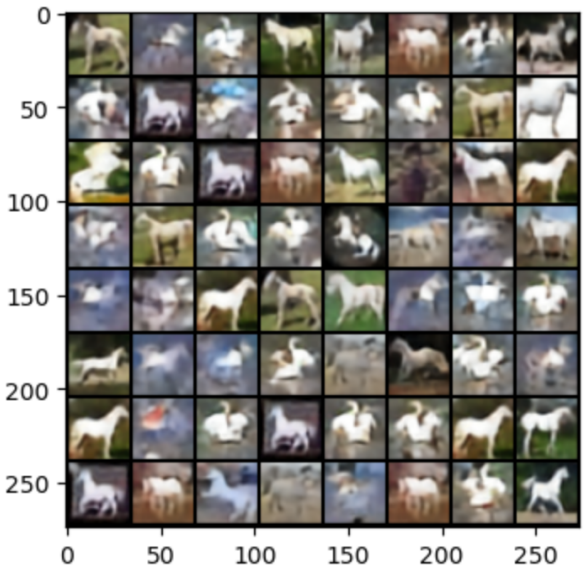

Figure 2: Trained on Cifar10

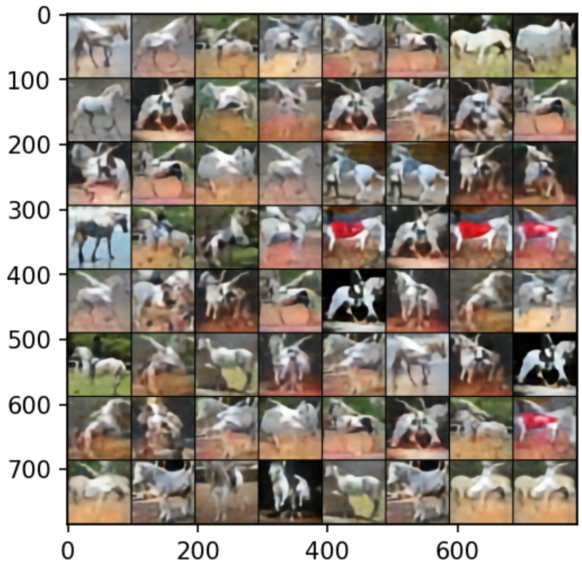

Figure 3: Trained on STL-10 96x96

Bonuses

The model has been trained on all possible datasets (+5 marks). The image chosen was generated after training on STL-10 at full resolution. For this batch of images almost all winged horses are white (+1 mark).

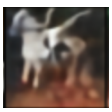

Figure 4: Best pegasus generated

References

[1] David Berthelot et al. “Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer”. In: (July 2018).

[2] Carl Doersch. “Tutorial on Variational Autoencoders”. In: (June 2016).

[3] Diederik Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: (Dec. 2014).

[4] Anders Larsen, Søren Sønderby, and Ole Winther. “Autoencoding beyond pixels using a learned similarity metric”. In: (Dec. 2015).

[5] Aaron Oord, Oriol Vinyals, and Koray Kavukcuoglu. “Neural Discrete Representation Learning”. In: (Nov. 2017).

[6] Aaron Oord et al. “Conditional Image Generation with PixelCNN Decoders”. In: (June 2016).

[7] Ali Razavi, Aaron Oord, and Oriol Vinyals. “Generating Diverse High-Fidelity Images with VQ-VAE-2”. In: (June 2019).

Code Used

https://github.com/jzbontar/pixelcnn-pytorch
https://github.com/singh-hrituraj/PixelCNN-Pytorch/blob/master/MaskedCNN.py
https://github.com/singh-hrituraj/PixelCNN-Pytorch/blob/master/Model.py
https://github.com/singh-hrituraj/PixelCNN-Pytorch/blob/master/train.py
https://github.com/singh-hrituraj/PixelCNN-Pytorch/blob/master/generate.py
https://stackoverflow.com/questions/65172786/how-to-load-one-type-of-image-in-cifar10-or-stl10-with-pytorch
https://github.com/zalandoresearch/pytorch-vq-vae/blob/master/vq-vae.ipynb