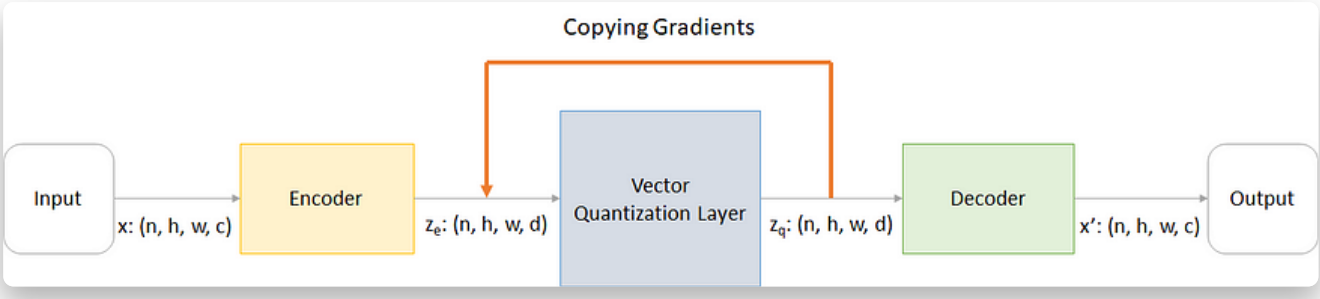
VQ-VAE

VAE 假设隐向量分布服从高斯分布

VQVAE 假设隐向量服从类别分布

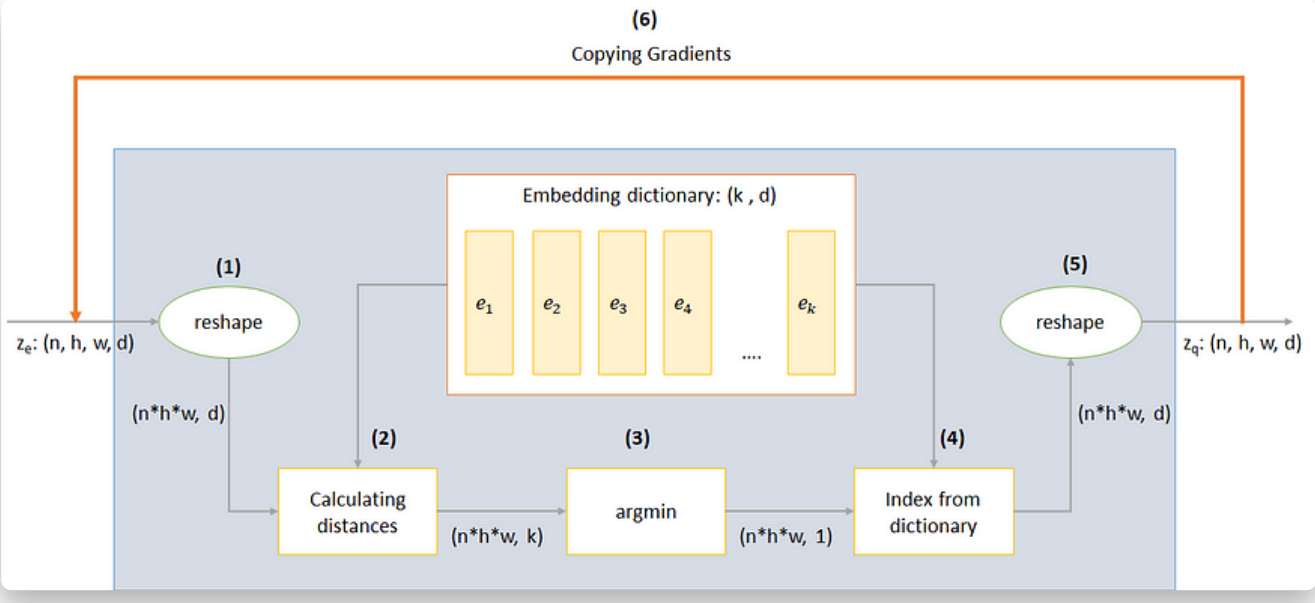
而 VQ-VAE 主要 argue 的点就是 VAE 的 prior 太简单了，容易造成 **posterior collapse,** 也就是 decoder 变得很强以后，normal distribution 的 prior 很容易被满足，decode 的结果也基本不靠 prior。更具体的说，VAE 的 prior 是建立在对角阵的 covariance 上，所以任意两个 latent code (z) 是认为正交 没有关联的。【这一点在 VQ-VAE 被改进了，尤其是引进了 pixelCNN 来 model.

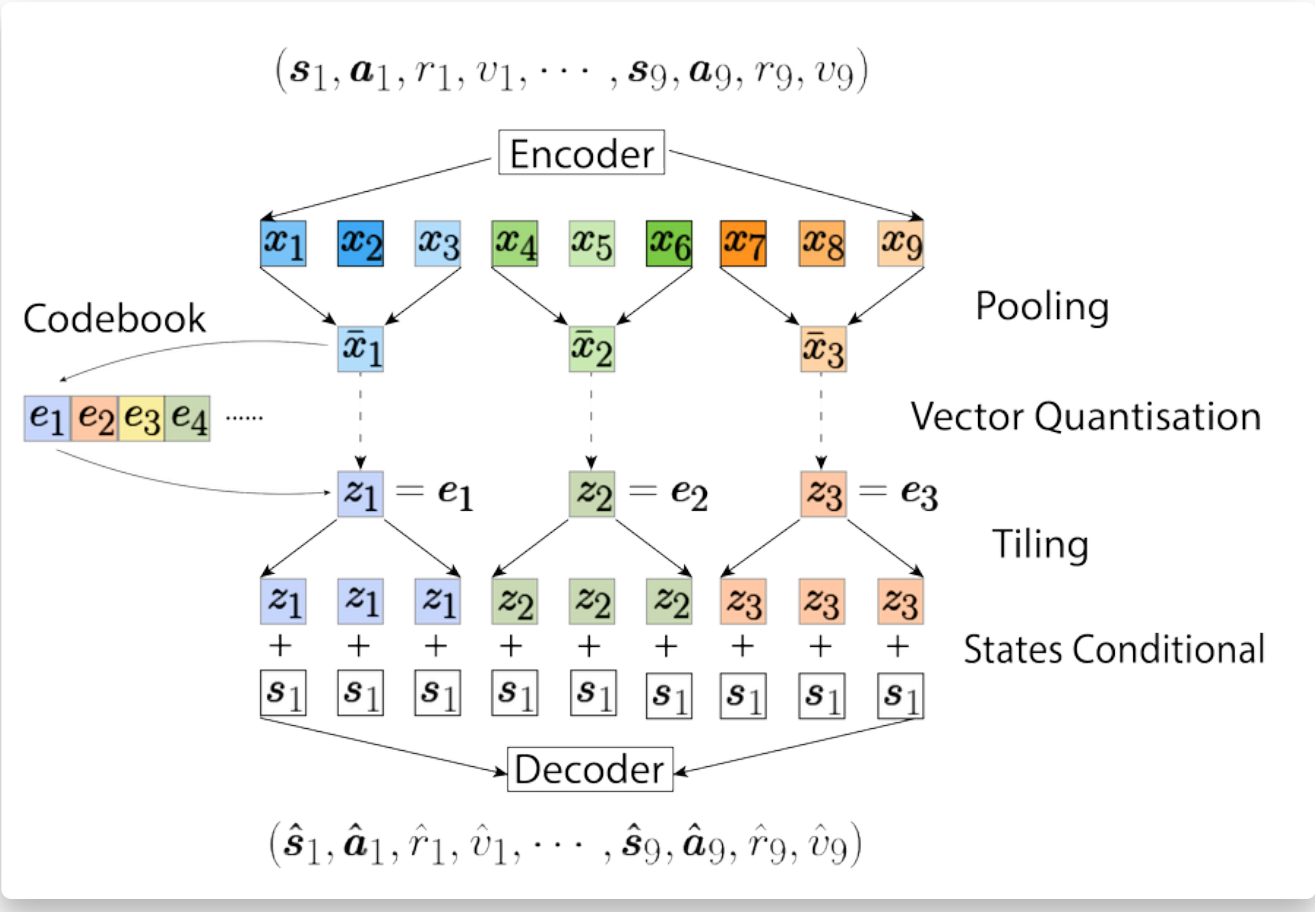


Vector Quantization Layer

01. **Reshape:** all dimensions except the last one are combined into one so that we have n*h*w vectors each of dimensionality d
02. **Calculating distances:** for each of the n*h*w vectors we calculate distance from each of k vectors of the embedding dictionary to obtain a matrix of shape (n*h*w, k)
03. **Argmin:** for each of the n*h*w vectors we find the the index of closest of the k vectors from dictionary
04. **Index from dictionary:** index the closest vector from the dictionary for each of n*h*w vectors
05. **Reshape:** convert back to shape (n, h, w, d)
06. **Copying gradients:** If you followed up till now you'd realize that it's not possible to train this architecture through backpropagation as the gradient won't flow through

argmin. Hence we try to approximate by copying the gradients from z_q back to z_e. In this way we're not actually minimizing the loss function but are still able to pass some information back for training.





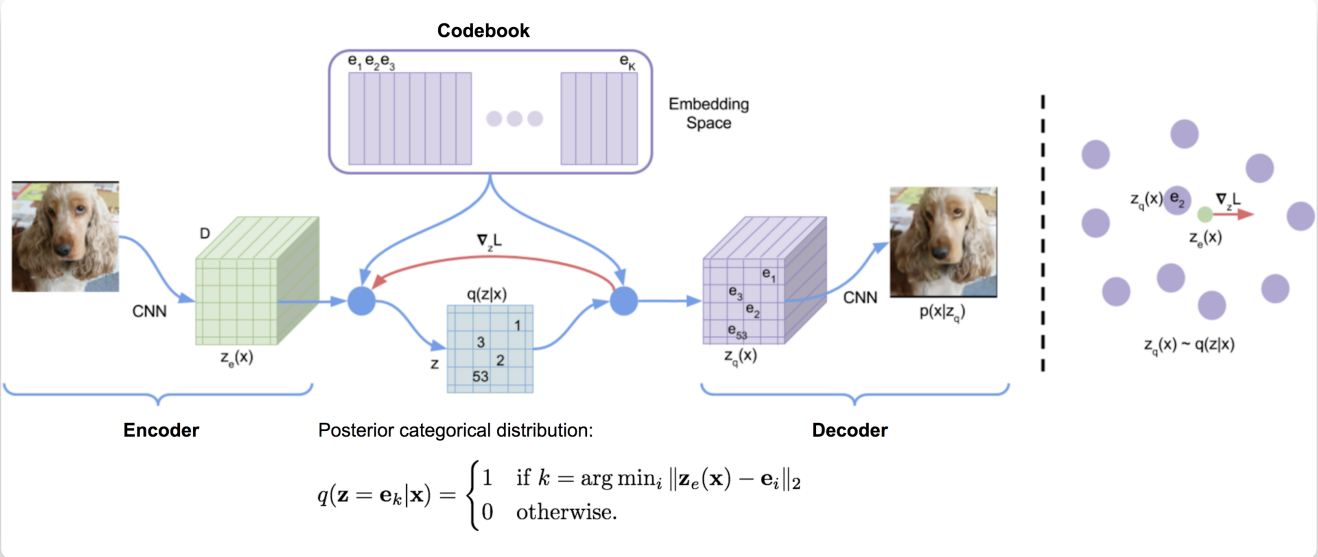
Vector quantisation (VQ) is a method to map $K-$dimensional vectors into a finite set of "code" vectors. The process is very much similar to KNN algorithm. The optimal centroid code vector that a sample should be mapped to is the one with minimum euclidean distance.

Let $\mathbf{e} \in \mathbb{R}^{K \times D}, i = 1, \ldots, K$ **be the latent embedding space (also known as "codebook") in VQ-VAE,** where K is the number of latent variable categories and D is the embedding size. An individual embedding vector is $\mathbf{e}_i \in \mathbb{R}^D, i = 1, \ldots, K$

The encoder output $E(\mathbf{x}) = \mathbf{z}_e$ goes through **a nearest-neighbor** lookup to match to one of K embedding vectors and then this matched code vector becomes the input for the decoder$D(.)$.

$$\mathbf{z}_q(\mathbf{x}) = \text{Quantize}(E(\mathbf{x})) = \mathbf{e}_k \text{ where } k = \arg\min_i \|E(\mathbf{x}) - \mathbf{e}_i\|_2$$

Note that the discrete latent variables can have different shapes in differnet applications; for example, 1D for speech, 2D for image and 3D for video. (VQ系列可以用在各种维度上) beitv1 维护codebook 编码。



Because argmin() is non-differentiable on a discrete space, the gradients $\nabla_z L$ from **decoder input** $\mathbf{z}_q$ **is copied to the encoder output** $\mathbf{z}_e$ Other than reconstruction loss, VQ-VAE also optimizes:

- ***VQ loss***: The L2 error between the embedding space and the encoder outputs.
- ***Commitment loss***: A measure to encourage the encoder output to stay close to the embedding space and to prevent it from fluctuating too frequently from one code

vector to another.

$$L = \underbrace{\|\mathbf{x} - D\left(\mathbf{e}_k\right)\|_2^2}_{\text{reconstruction loss}} + \underbrace{\|\text{sg}[E(\mathbf{x})] - \mathbf{e}_k\|_2^2}_{\text{VQ loss}} + \underbrace{\beta\|E(\mathbf{x}) - \text{sg}\left[\mathbf{e}_k\right]\|_2^2}_{\text{commitment loss}}$$

where $sq[.]$ is the `stop_gradient` operator.

01. **Reconstruction loss**: which optimizes the decoder and encoder:

```
reconstruction_loss = -log( p(x|z_q) )
```

02. **Codebook loss**: due to t**he fact that gradients bypass （绕过）the embedding,** we use a dictionary learning algorithm which uses an l_2 error to move the embedding vectors $e_i$towards the encoder output:

```
codebook_loss =  ‖ sg[z_e(x)]- e ‖^2
// sg represents stop gradient operator meaning no gradient
// flows through whatever it's applied on
```

03. **Commitment loss**: since the volume of the embedding space is dimensionless, it can grow arbirtarily if the embeddings e_i do not train as fast as the encoder parameters, and thus **we add a commitment loss to make sure that the encoder commits to an embedding**

```
commitment_loss = β‖ z_e(x)- sg[e] ‖^2
// β is a hyperparameter that controls how much we want to weigh
// commitment loss compared to other components
```

The embedding vectors in the codebook is updated through **EMA** (exponential moving average). Given a code vector $\mathbf{e}_i$, say we have n_i encoder output vectors, $\{\mathbf{z}_{i,j}\}_{j=1}^{ni}$ that are quantized to $\mathbf{e}_i$

$$N_i^{(t)} = \gamma N_i^{(t-1)} + (1-\gamma)n_i^{(t)} \quad \mathbf{m}_i^{(t)} = \gamma \mathbf{m}_i^{(t-1)} + (1-\gamma)\sum_{j=1}^{n_i^{(t)}} \mathbf{z}_{i,j}^{(t)} \mathbf{e}_i^{(t)} = \mathbf{m}_i^{(t)} / N_i^{(t)}$$

where (t) refers to batch sequence in time. N_i and $\mathbf{m}_i$ are accumulated vector count and volume. respectively.