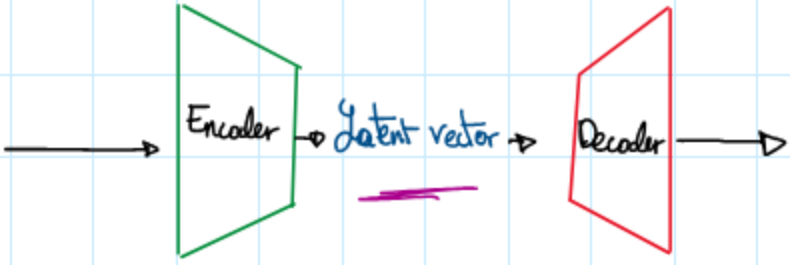
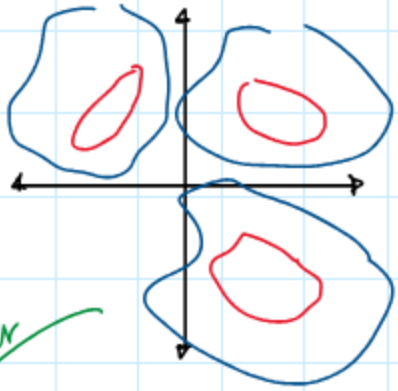
VQ VAE

Tuesday, October 8, 2024 10:11 PM

In Auto encoders the encoder input is mapped to a (usually low) Dim. latent space and reconstructed by the decoder to the original input.



In VAE, each point x encoded as a distribution in the latent space



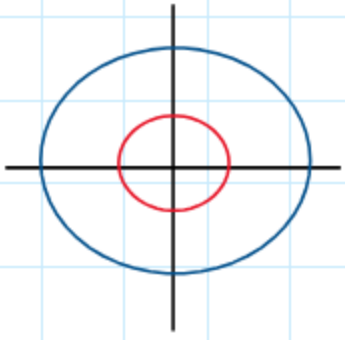
point distribution

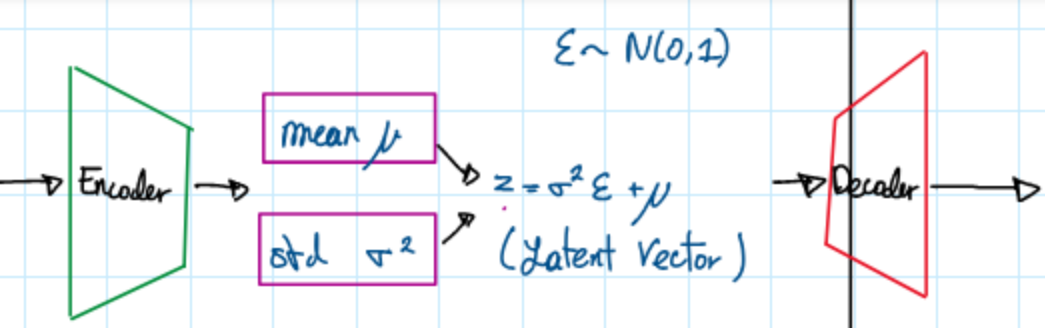
encoder

(Prior: assume isotropic Gaussian dist.)

covariance - $\sigma^2 I$

(latent dimensions are independent)





With some tricks we showed before that this is trainable etc.

So now we can sample points from a continuous latent space

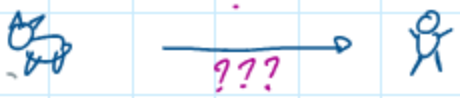
$z = \sigma^2 \varepsilon + \mu$, $\varepsilon \sim N(0, 4)$

We go from [input sample $\xrightarrow{\text{Encoder}}$ z latent space]

One issue is that a continuous representation does not always make sense.

If we were mapping a distribution that has cats vs. dogs, continuous latent space representations may not make sense.

1 1 1 7 7 7 (continuous) ✓



Language also has these discrete representations

VQ VAE splits off in two ways:

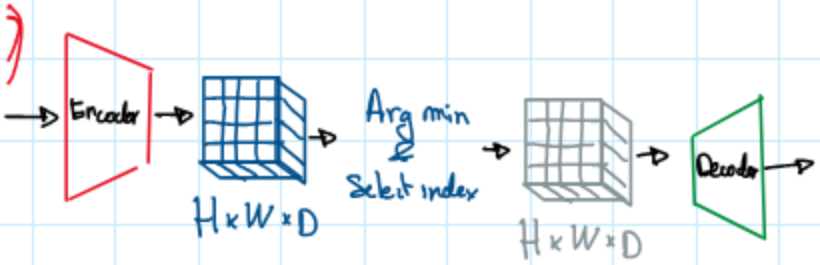
1) Embedding space becomes codebook. (discrete)

2) Prior is not static $N(0,1)$, it is learned

Input → Encoder → Feature map → argmin (NN) → Encoded discrete vector → decoder → Output

In pretty picture representation:

In pretty picture representation:



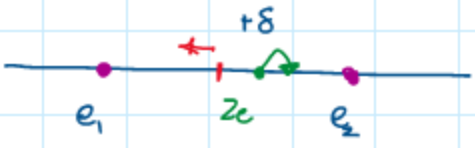
Issue is:

this step would bring all gradients to 0

That is due to our posterior being:

$$q(z = k|x) = \begin{cases} 1 & \text{if } k = \arg\min \|z_e(x) - e_j\|_2 \\ 0 & \text{otherwise} \end{cases}$$

For a simple example:



adding δ to z_e has no Δ output so deriv = 0

- Since we are working in high dimensions much much more points have no effect.

  (lots of volume, very few boundary points)
- Also the points that cause a switch $0 \mapsto 1$ have a derivative of ∞

What the authors do (Straight Through Estimation)

Feature Map → Argmin (Select Index) → Quantized Vectors

Feature Map → label them as A

Argmin (Select Index) → f

Quantized Vectors → B

node with non-differentiable function f

$B = f(A)$ ←

$= A + (f(A) - A).\text{detach}()$

separate from computational graph so gradients are not back propogated

$B = f(A)$

In pretty picture representation:

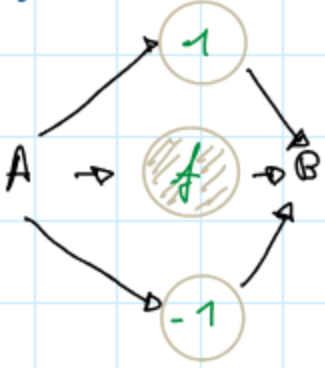
Original Setup

$B = f(A) \Rightarrow$

New Setup

$B = A + (f(A) - A).\text{detach}()$

$\Rightarrow$



In backprop, we skip the middle node, so its like if we had $B = A$ ← it goes "straight through"

In other words, the gradients jump over the non-differentiable step as if there was an Identity operation between them

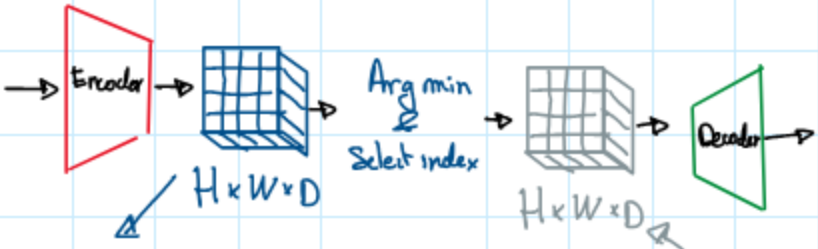
From the paper

"Since the output representation of the encoder and the input to the decoder share the same D dimensional shape, the gradients contain useful information for how the encoder has to change its input to lower the reconstruction loss"

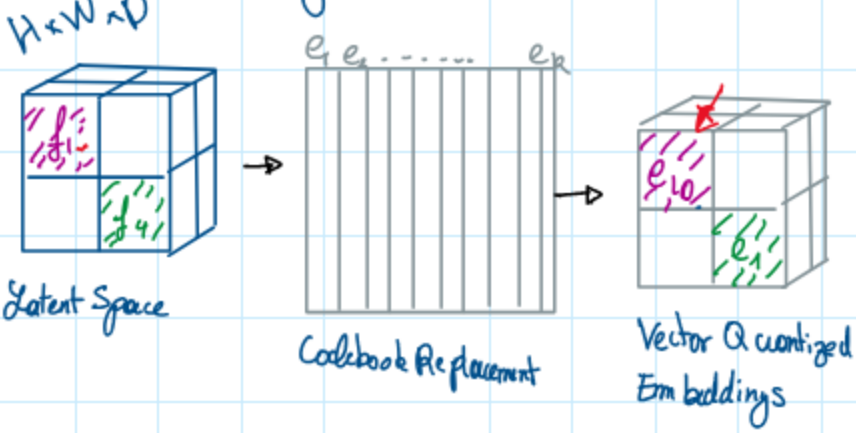
Learning Strategy:

If you remember

In pretty picture representation:



Vector in here are mapped to vector in here through a user decided number of embeddings in a codebook



So we need to add to our loss a way to learn these codebook embeddings.

Dictionary Learning Algorithm: Vector Quantization

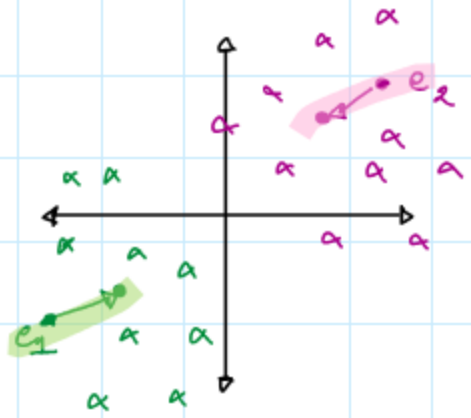
We regard the quantization operation as clustering encoder outputs into k-clusters

We need a new codebook loss to train the encoder outputs

This becomes a loss to improve clustering.

We need a new codebook loss to train the encoder outputs

This becomes a loss to improve clustering.



moves the e_1, e_2 closer to the centroid of the cluster clusters are generated through NN

Codebook Loss = $\| sg[z_e(x)] - e \|_2^2$ → We are moving embedding vectors e_i towards the encoder outputs $z_e(x)$

sg : stop gradient operation (don't update)

$z_e(x)$: encoder output latent space

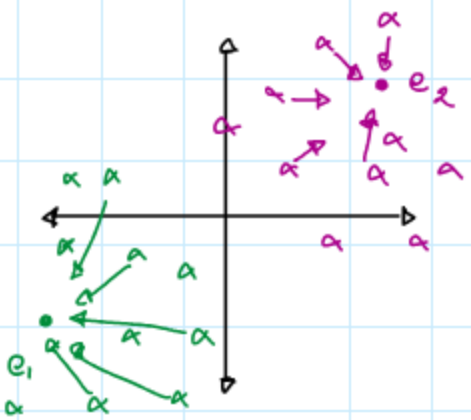
With only the codebook loss, the volume of the embeddings has no upper bound

So if the embedding train slower than the encoder, they could get arbitrarily big.

Therefore, we add a commitment loss:

hyper parameter → $\beta \| z_e(x) - sg[e] \|_2^2$

Treat embeddings as constant, and bring the latent space vectors closer



So the overall loss is 3-fold:

Codebook Loss: moves discrete embeddings to centroids

- $\|sg[z_e(x)] - e\|_2^2$

Commitment Loss moves encoder outputs $z_e(x)$ closer to discrete embeddings

- $\|z_e(x) - sg[e]\|_2^2$

Reconstruction loss from VAEs, we use

- MSE loss of the log likelihood

= $\log p(x|z_q(x))$

Sum them for overall loss.

KL divergence?

You may notice the KL divergence term from VAEs is missing

KL Loss would be: $\sum q(z|x) \log \frac{q(z|x)}{p(z)}$ (predicted)

$q(z|x)$ distribution of discrete embeddings

$p(z)$ prior for the discrete latents

In VQ VAE, our prior $p(z)$ is uniform ← all entries in the codebook are treated as equally likely without any information about the specific input.

= $\frac{1}{K}$ for all values

$q(z|x)$ is a 1-hot vector

$$= \begin{cases} 1 & \text{for correct } ij \\ 0 & \text{else} \end{cases}$$

So the KL term will only have 1 term which is equal to $\log(K)$

Why you should use it:

One weakness of VAEs that VQ VAEs helps to solve is posterior collapse

In VAEs we had

fixed prior: $p(z) \sim N(0, 1)$

learned posterior: $q(z|x)$

We train this with ELBO:

$$ELBO(x; p_\theta, q_\phi) = \mathbb{E}_{q_\phi}[\log p_\theta(x)] - KL(q_\phi(z|x), p_\theta(z|x))$$

reconstruction loss — Regularization

When the KL term 'dominates' during training the model will ignore the latent space representation and learn directly from $p(z)$

=> Model generates directly from $p(z)$

Decoder is too powerful and doesn't need the latent

=> Model stops learning useful latent vectors

=> Generation Quality decreases

VQ VAE fixes this through its codebook with a discrete set of embeddings instead of a continuous probability distribution

There is also no KL divergence

How to do just Generation

- Freeze the model
- Pass in inputs and save index map of NN or VQ output
- Train a transformer autoregressively to predict masked tokens then use that to fill in sequences from the start and generate images

Where it is used

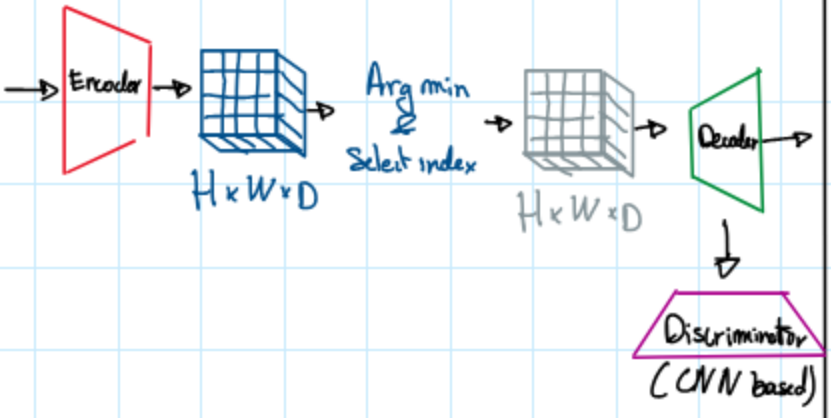
VQ GANs:

There is also VQ GANs which adds a discriminator

One issue with VQ VAEs is that they suffer from blurry reconstructions due to pixel wise reconstruction loss (MSE)

VQ GANs add a discriminator loss:

In pretty picture representation:



This allows for a high-level feature based perceptual loss

The adversarial nature of the training forces better reconstruction which requires better use of the latent space

This is caused by the addition of 2 loss terms:

Adversarial loss:

$$L_{GAN}^{Gen} = -\mathbb{E}[\log D(\hat{x})]$$ (generator should fool the discriminator)

$$L_{GAN}^{Disc} = -\mathbb{E}[\log D(x)] - \mathbb{E}[\log(1 - D(\hat{x}))]$$

(Real as real, Fake a Fake)

Perceptual loss: (Replaces MSE Loss)

Compares high level features in the input image with reconstruction.

$$L = \sum_i \| \phi_i(x) - \phi_i(\hat{x}) \|_2^2$$

ϕ_i is the ith layer activations of a frozen, pretrained network (such a very deep convolutional network for image classification)

Both VQ VAEs & VQ GANs are then used for:

Di: VAEs is diffusion using a VQ VAE architecture with a diffusion decoder

Apple:

4M uses VQ VAE for modality specific tokenization