Autoencoders

- Try to reconstruct input as its output
- Minimise reconstruction loss (diff b/w x, x')

Encoder: $z = g(w_e \cdot x + b_e)$

Decoder: $x' = f(w_d \cdot z \cdot b_d)$

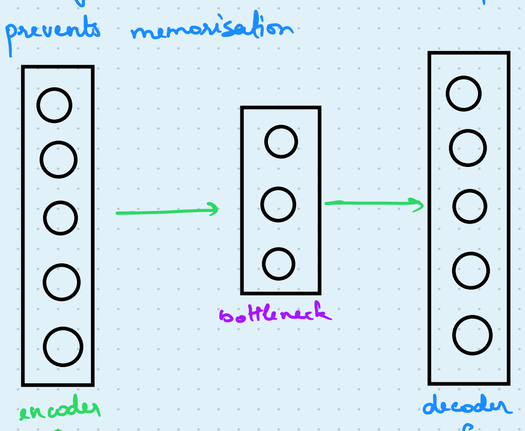
- Data can be represented non-linearly (unlike PCA)
- No interpretability
- Self-supervised → does prediction (unlike unsupervised) data labels implicit in data (predict input from input)

Limitations of basic AE

- Can memorise training data instead of generalising
- [from above] sensitive to input perturbations
- No structure in latent space

Undercomplete AE

Bottleneck layer → ensures dim(latent space) << dim(input space)
Thus prevents memorisation



Loss function:

$$L(x,x') = \frac{1}{N}\sum_{i=1}^{N}\left(x^{(i)} - x'^{(i)}\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left[x^{(i)} - f(g(x^{(i)}))^2\right]$$

Basically MSE

Sparse AE

- Only looks for "most important clues" → activate sparse subset of hidden neurons acc- to most relevant features
- How to pick relevant features? → L1 penalisation

$$L1 = \text{Reconstruction loss} + \lambda\left[h_i\right]$$

λ: controls sparsity; h_i: no of hidden activations

Implementation key:

Small hidden layer → limits capacity, ensure latent space dimensionality is low

Sparsity constraints → further limit active neurons

Effect:

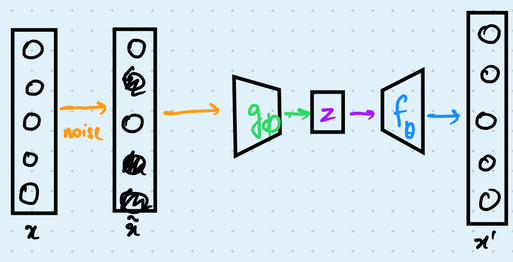
- Forces model to prioritise features essential for reconstruction
- Encourages disentangling → separating distinct aspects of data into different neurons

Trade-off:

- Higher compression → better feature extraction
  → higher reconstruction loss

Denoising AE

- Able to reconstruct original input from corrupted input
- More robust model



Masking/Noise: $\tilde{x}^{(i)} \sim f(\tilde{x}^{(i)} \mid x^{(i)})$

Loss:

$$L(\theta,\phi) = \frac{1}{n}\sum_{i=1}^{n}\left[x^{(i)} - f_\theta(g_\phi(\tilde{x}^{(i)}))\right]^2$$

$x^{(i)}$: clean input; $f_\theta(g_\phi(\tilde{x}^{(i)}))$: encoder-decoder on noised input

Types of noise

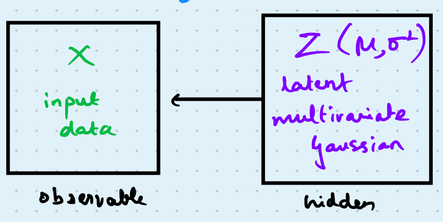
1. Gaussian: Add random values from $N(\mu,\sigma)$
2. Salt and pepper: Set random pixels to 0 or 1
3. Masking: Hide random parts
4. Dropout: Randomly set activations to 0

Why this works

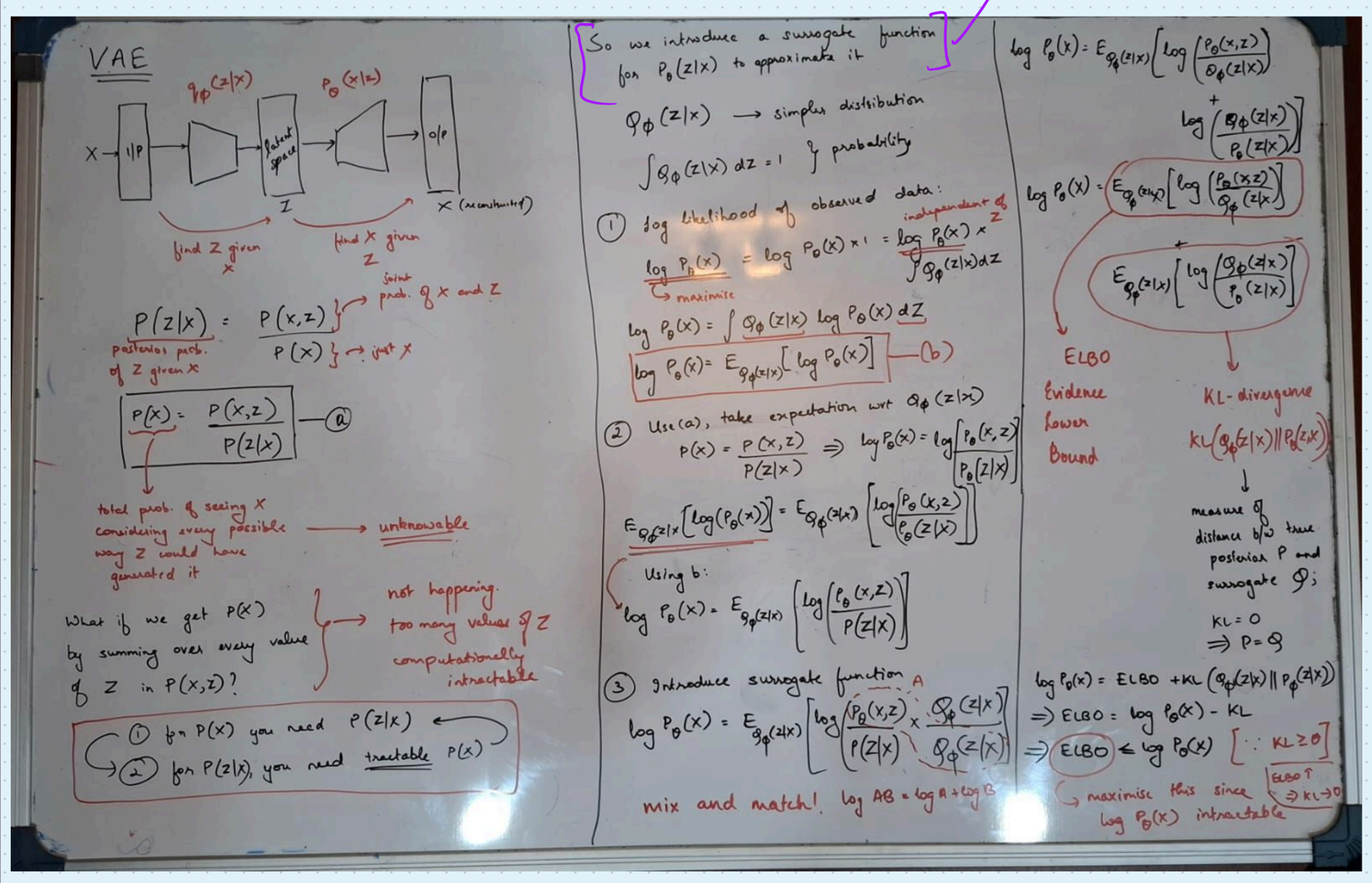
- Forces model to learn underlying structure, cannot memorise
- Better generalisation to variations in the test set
- Feature hierarchy: lower layers learn local features
  higher layers learn global features
- Learns underlying data manifold → lower dimensional space within input space that has meaningful data
- Essentially learns this probability: $P(x \mid \tilde{x})$ → entire possible

Variational Autoencoder (VAE)

- Generative model: learn a latent space distribution and sample from that instead of reconstructing
- Assume latent space to be some random variable Z
- Also assume input X is conditioned on Z
- Z → modelled as multivariate gaussian



Tweak Z according to assumptions to learn distribution of X



ELBO → reconstruction loss (MSE)
→ KL divergence b/w prior Z and learnt Z

Reparameterisation Trick

maximisation of ELBO → calculate gradient.

$$ELBO = \mathbb{E}_{q_\phi(z|x)}\left[\log\left(\frac{P(x,z)}{q_\phi(z|x)}\right)\right]$$

Calculating gradient with Z very difficult → Z → stochastic, random values → gradient shows very high variance → sampling is a non-differentiable operation

Solution: Separate the randomness from model parameters!

1. Z supposed to be from $N(\mu,\sigma)$ where μ,σ learnt by encoder ($P(z|x)$)
2. Sample an external source of noise for randomness: $\varepsilon \sim N(0,1)$ → std. normal
3. Calculate latent Z using a deterministic function.

$$z = \mu + \sigma \cdot \varepsilon$$