MGM: Problemset

Category	
Files	
Created	@May 3, 2023 10:44 AM
Reminder	
Status	Open
URL	
Updated	@May 5, 2023 2:28 PM

Exercise 0.8

1.

A fundamental problem in statistics and machine learning is to come up with useful measures of “distance” between pairs of probability distributions.

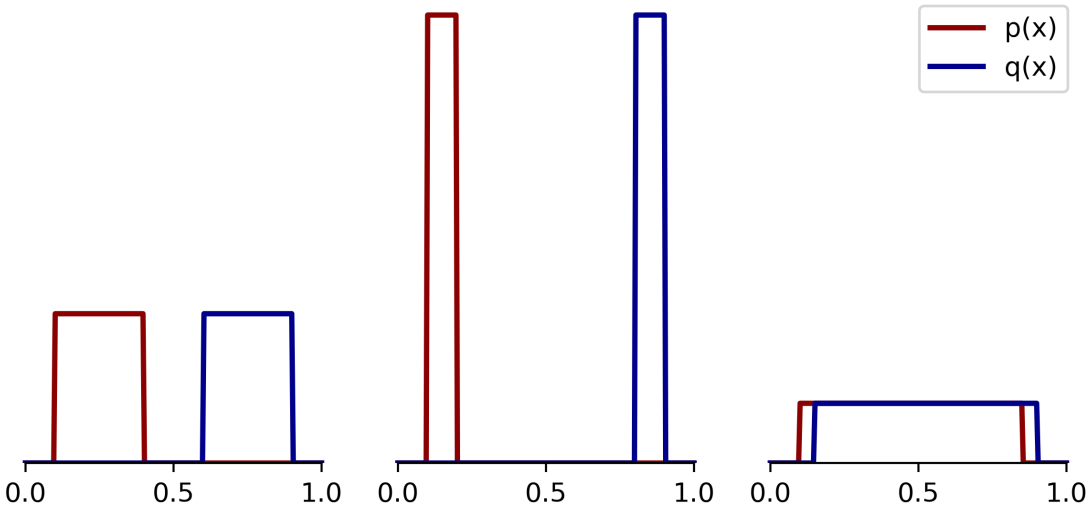
Two desirable properties of a distance function are symmetry and the triangle inequality. Unfortunately, many notions of “distance” between probability distributions do not satisfy these properties. These weaker notions of distance are often called *divergences*. Perhaps the most well-known divergence is the **Kullback-Lieibler (KL) divergence**:

$$D_{KL}(P||Q) = \int p(x) \cdot log\left(\frac{p(x)}{q(x)}\right)$$

While the KL divergence is incredibly useful and fundamental in information theory, it also has its shortcomings:

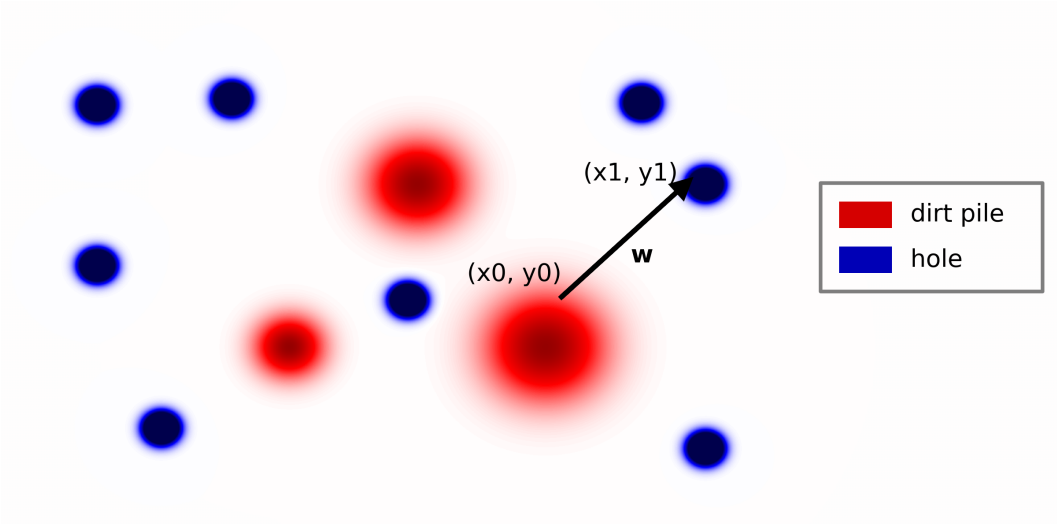
- For instance, one of the first things we learn about the KL divergence is that it is not symmetric, $D_{KL}(P||Q) \neq D_{KL}(Q||P)$. This is arguably not a huge problem, since various symmetrized analogues to the KL divergence exist.
- A bigger problem is that the divergence may be infinite if the support of P and Q are not equal.
  One way of circumventing this is to smooth (i.e. add blur to) the distributions

before computing the KL divergence, so that the support of P and Q matches. However, choosing the bandwidth parameter of the smoothing kernel is not always straightforward.



Optimal transport theory is one way to construct an alternative notion of distance between probability distributions.

Optimal transport (also known as Monge-Kantorovich or transportation) problem is a mathematical optimization problem that aims to find the optimal way of transporting a set of objects from one location to another, given some cost or distance measure between the objects and the transportation cost.



More formally, let X and Y be two metric spaces, and let μ and ν be two probability measures on X and Y, respectively. The optimal transport problem seeks to find a transport plan π that minimizes the cost functional:

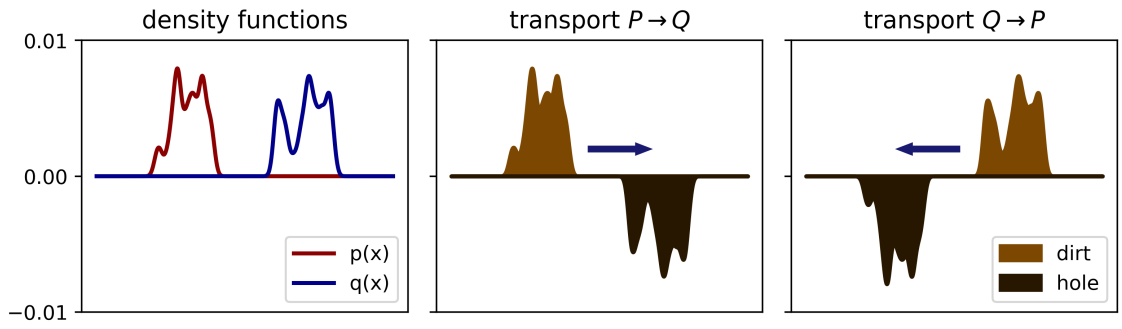
$$\inf_{\pi \in \Pi(\mu,\nu)} \int_{X \times Y} c(x,y)\, d\pi(x,y)$$

subject to the marginal constraints:

$$\pi(A \times Y) = \mu(A)$$
$$and$$
$$\pi(X \times B) = \nu(B)$$

where,

- $c(x,y)$ is the cost or distance function between x and y
- $\Pi(\mu,\nu)$ is the set of all joint probability measures on $X \times Y$ with marginals μ and ν.



The total transportation cost overcomes the two weaknesses of KL divergence. First, since the
cost function C is symmetric, the overall cost to transport $P \to Q$ is the same as transporting $Q \to P$.

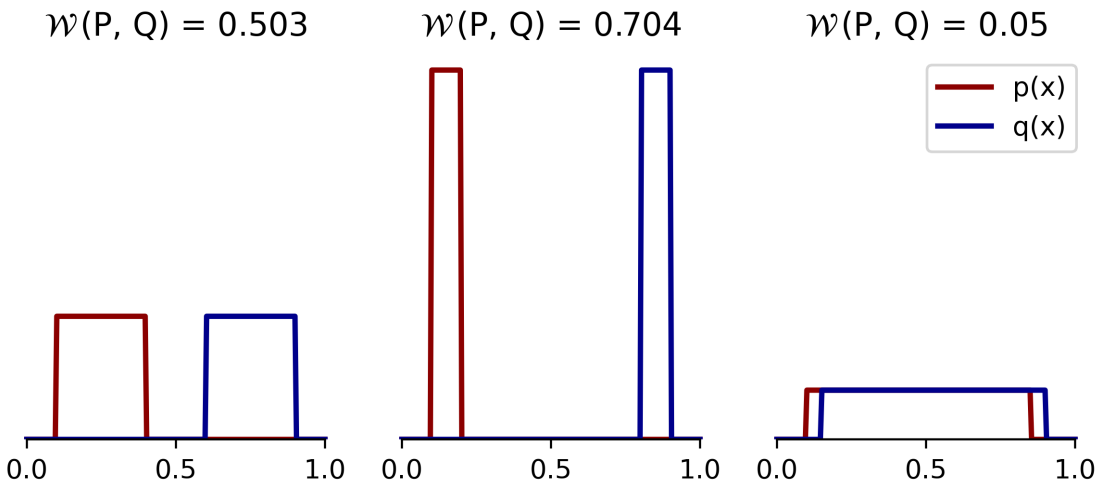
The **Wasserstein distance**, also known as the earthmover's distance, is a special case of the optimal transport problem, where the cost function $c(x, y)$ is the distance between x and y raised to some power p (usually p=1 or p=2).

The Wasserstein distance of order p between two probability measures μ and ν is defined as:

$$W_p(\mu, \nu) = \left(\inf_{\pi \in \Pi(\mu, \nu)} \int_{X \times Y} d(x, y)^p \, d\pi(x, y) \right)^{\frac{1}{p}}$$

where,

- $d(x, y)$ is the distance between x and y.



The Wasserstein distance provides a metric on the space of probability measures that captures the similarity between two distributions in terms of their shape and location. This distance is not only symmetric, but also satisfies the triangle inequality.2

Recall the second shortcoming of KL divergence — it was infinite for a variety of distributions with unequal support. Not only is the Wasserstein distance finite in all cases, but the distances agree with our natural intuitions: the panel on the right

results in the smallest Wasserstein distance, while the middle panel shows the largest distance.

2.

Entropy-regularized optimal transport (EROT) is a modification of the standard optimal transport problem that introduces a strongly convex regularizer to the cost functional.

The strong convex regularizer encourages the optimal transport plan to be more diffuse or spread out, leading to smoother and more stable solutions.

$$d_c^{\lambda}(p,q) = \min_{\pi} < c, \pi > -\lambda \cdot H(\pi)$$

Where,

- $H(\pi) = -\sum \pi(x,y)\ log(\pi(x,y))$
- Negative of H / Shannon Entropy is strongly convex
- Thus, d = linear + strongly convex = strongly convex
- Thus, this is guaranteed to have unique minimizer

The role of entropy regularization in EROT is to overcome some of the limitations of the standard optimal transport problem, such as the difficulty of computing the optimal transport plan and the senwsitivity to noise and outliers in the data.

Entropy regularization allows for efficient and stable computation of the optimal transport plan using convex optimization techniques, and it also provides a smooth and continuous approximation to the optimal transport problem, which can be easier to work with in practice

3.

(3)

Let 'f' be linear.

Let 'g' be strongly convex.

$\Rightarrow \quad h = f + g$

$\because$ f is also convex, we can write:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \quad -(1)$$

[1st order cond^n]

$\because$ g is strongly convex, we can write:

[strong conv^n cond^n]

$$g(y) \geq g(x) + \nabla g(x)^T (y - x) + \frac{\lambda}{2} \|y - x\|^2 \quad -(2)$$

(for some +ve const. λ)

(1) + (2)

$$[f(y) + g(y)] \geq [f(x) + g(x)] + (\nabla f(x)^T + \nabla g(x)^T)(y - x) + \frac{\lambda}{2} \|y - x\|^2$$

$$\Rightarrow \boxed{h(y) \geq h(x) + h(x)^T (y - x) + \frac{\lambda}{2} \|y - x\|^2}$$

Thus, 'h' is strongly convex

4.

The entropy-regularized optimal transport problem can be solved using the alternating projection algorithm, which alternates between finding the optimal

transport plan with respect to the cost function and the entropy term.

(4)

$$d_c^{\lambda}(p,q) = \min_{\pi} \langle c, \pi \rangle - \lambda . H(\pi)$$

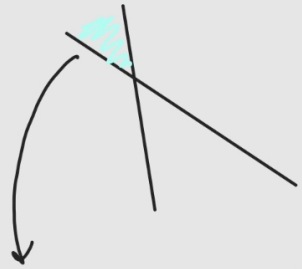
$\because d_c^{\lambda}(p,q)$ is strongly convex

↳ has existence of a unique sol^n:

$$\underset{\pi \in \Pi(p,q)}{\operatorname{argmin}} \; D_{KL}(\pi \,\|\, p_K^{\lambda})$$

Let $\Pi(p)$, $\Pi(q)$ denote row and column marginal constraints.

$$\Pi(p,q) = \Pi(p) \cap \Pi(q)$$



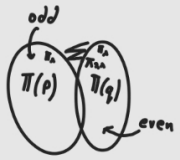
finding intersection b/w these 2 affine sets.

To do this we use Alternating Projection which finds intersection b/w 2 convex sets C_1, C_2.

Initialize

$$\pi_{\lambda}^{0} = p_K^{\lambda} \quad \& \text{ define alternating projections.}$$

$$\pi_{\lambda}^{(l+1)} = \begin{cases} \underset{\pi \in \Pi(p)}{\operatorname{argmin}} \; D_{KL}(\pi \,\|\, \pi_{\lambda}^{l}), & l = \text{even} \\ \underset{\pi \in \Pi(q)}{\operatorname{argmin}} \; D_{KL}(\pi \,\|\, \pi_{\lambda}^{(l+1)}), & l = \text{odd} \end{cases}$$



Note

$$\Pi(p) = \{\pi \mid \pi 1_m = p\}$$
$$\Pi(q) = \{\pi \mid 1_n^T \pi = q\}$$

↳ affine sets

↳ convex, can use alternating projection

For l even: Need to find

$$\arg\min_{\pi 1_m = p} D_{KL}(\pi \| \pi_\lambda^l)$$

Since constrained problem $\longrightarrow$ introduce Lagrange mul.

$$\Rightarrow \arg\min_{\pi} \max_{f} D(\pi \| \pi_\lambda^{(l)}) + \underbrace{\langle f, p - \pi 1_m \rangle}_{\text{Lagrange multiplier}}$$

Slater's condn $\downarrow$

Assume strong duality interchange min & max, and then look at the min prob.

From First order optimality:

$$\frac{\delta}{\delta \pi} \left[D(\pi \| \pi_\lambda^{(l)}) + \langle f, p - \pi 1_m \rangle \right] = 0$$

But current π is $\pi_{\lambda,f}^{(l+1)}$:

$$\frac{\delta}{\delta \pi_\lambda^{l+1}} \left[D_{KL}(\pi_\lambda^{l+1} \| \pi_\lambda^l) \right]$$

$$\Rightarrow \frac{\delta}{\delta \pi_\lambda^{l+1}} \left[\pi_\lambda^{(l+1)}(x,y) \log \pi_\lambda^{(l+1)}(x,y) - \pi_\lambda^{(l+1)}(x,y) \underbrace{\log \pi_\lambda^l(x,y)}_{\text{L acts like a const.}} \right]$$

$$\Rightarrow 1 + \log \pi_\lambda^{l+1}(x,y) - \log \pi_\lambda^l(x,y) \quad — (1)$$

& we have

$$\frac{\delta}{\delta \pi_\lambda^{l+1}} \langle f, p - \pi_\lambda^{l+1} 1_m \rangle \Big|_{\text{for fixed } (x,y)} = -f_n \quad — (2)$$

Combining ① & ② (1st order optimality)

$$1 + \log \pi_\lambda^{(l+1)}(x,y) - \log \pi_\lambda^{l}(x,y) - t_x = 0$$

Solve for $\pi_\lambda^{(l+1)}(x,y)$,

$$\Rightarrow e^{t_x - 1} = \frac{\pi_\lambda^{l+1}(x,y)}{\pi_\lambda^{l}(x,y)}$$

$\because \pi_\lambda^{l+1} \in \Pi(p)$, we have:

$$p(x) = \sum \pi_\lambda^{l+1}(x,y) = e^{t_x - 1} \sum \pi_\lambda^{l}(x,y) \quad -③$$

$$\pi_\lambda^{(l+1)}(x,y) = \frac{p(x)}{\sum_y \pi_\lambda^{l}(x,y)} \pi_\lambda^{l}(x,y)$$

Changing $l+1$ to $2l$ (even)
& l to $2l-1$

$$\pi_\lambda^{(2l)}(x,y) = \frac{p(x)}{\sum \pi_\lambda^{(2l-1)}(x,y)} \pi^{(2l-1)}(x,y)$$

vectorize for $\forall (x,y)$:

$$\pi_\lambda^{(2l)} = \mathrm{diag}\left(\frac{p}{\pi_\lambda^{(2l-1)} 1_m}\right) \pi_\lambda^{(2l-1)} \quad -④$$

Similarly for odd:

$$\pi_\lambda^{(2l+1)} = \mathrm{diag}\left(\frac{q}{1_n^T \pi_\lambda^{(2l)}}\right) \pi_\lambda^{(2l)} \quad -⑤$$

5.

Sinkhorn's Theorem

The Sinkhorn's theorem states that every square matrix with positive elements can be transformed into a doubly stochastic matrix $D_1 A D_2$. Where $D1$ and $D2$ are diagonal matrices with all positive main diagonals. The matrices $D1$ and $D2$ are themselves unique up to a constant factor.

A doubly stochastic matrix is an all non-negative square matrix that is both row-normalized and column-normalized.

Sinkhorn–Knopp algorithm

A simple iterative method to approach the double stochastic matrix is to alternately rescale all rows and all columns of A to sum to 1.

```
def sinkhorn(A, N, L):
    # Pseudo-Code for calculating the doubly stochastic matrix
    # using Sinkhorn-Knopp algorithm.
    # ----------
    # Input: positive matrix A[N x N], max iteration L

    for i in range(L):
        A = A / np.matmul(A, np.ones(N, 1))
        A = A / np.matmul(np.ones(1, N), A)

        # Test for convergence and early stop.
        if _converge:
            break

    return A
```

Optimal Transport

The Sinkhorn algorithm can be used for computing the optimal transport plan between two probability measures. The algorithm is based on the idea of alternating projections, and uses a scaling parameter to regularize the computations.

Proposition:
Let $K \in \mathbb{R}^{n \times n}$ with $K_{x,y} = K(x, y)$. For some $u \in \mathbb{R}^n, v \in \mathbb{R}^m$:

$$\pi_\lambda = diag(u) \cdot K \cdot diag(v)$$

We Approximate the solution by initializing:

- $u^{(1)} = 1_n,\ v^{(1)} = 1_m$
- $u^{(l+1)} = \frac{p}{k.v^{(l)}},\ v^{(l+1)} = \frac{q}{k^T.u^{(l)}}$

$$\tilde{\pi} = diag(u) \cdot K \cdot diag(v)$$

$$\tilde{\pi}^{(2l)} = diag(u^{(l+1)}) \cdot K \cdot diag(v^{(l)})$$

$$\tilde{\pi}^{(2l+1)} = diag(u^{(l+1)}) \cdot K \cdot diag(v^{(l)})$$

6.

Matrix Scaling

Matrix Scaling problem is about finding u and v that scales rows and columns of K, such that it matches the row and column sum of the given distribution p and q.

In our case,

$p = \Pi \cdot 1_m = diag(u)\ K\ v$

$q = \Pi \cdot 1_m = diag(v)\ K^T\ u$

7.

$$p = \pi 1_m = \text{diag}(u)\, K\, v$$
$$q = \pi^T 1_n = \text{diag}(v)\, K^T u \quad \Big\} \quad -(\star)$$

We Approximate the soln to $(\star)$ by initializing

$$u^{(1)} \equiv 1_n \;, \quad v^{(1)} \equiv 1_m$$

$$u^{(l+1)} = \frac{p}{K v^{(l)}} \;, \quad v^{(l+1)} = \frac{q}{K^T u^{(l)}}$$

With these choices we get primal iterates:

Proof:

$$\tilde{\pi} = \text{diag}(u)\, K\, \text{diag}(v)$$
$$\tilde{\pi}^{(2l)} = \text{diag}(u^{(l+1)})\, K\, \text{diag}(v^{(l)}) \quad -(A)$$
$$\tilde{\pi}^{(2l+1)} = \text{diag}(u^{(l+1)})\, K\, \text{diag}(v^{(l+1)}) \quad -(B)$$

Rearranging terms from (B), $(l = l-1)$

$$K\, \text{diag}(v^{(l)}) = \frac{\tilde{\pi}^{(2l-1)}}{\text{diag}(u^{(l)})} \quad -(C)$$

Using in (A),

$$\tilde{\pi}^{(2l)} = \text{diag}(u^{(l+1)})\, \frac{\tilde{\pi}^{(2l-1)}}{\text{diag}(u^{(l)})}$$

$$= \text{diag}\left(\frac{p}{K v^{(l)}}\right) \frac{\tilde{\pi}^{(2l-1)}}{\text{diag}(u^{(l)})}$$

$$= \text{diag}\left(\frac{p}{\text{diag}(u^{(l)})\, K\, v^{(l)}}\right) \tilde{\pi}^{(2l-1)}$$

Recall (B) $(l = l-1)$

$\text{diag}(v^{(l+1)})\, 1_m = v^{(l+1)}$

(diagonal matrix) (vector)

from (B)

$$= \text{diag}\left(\frac{p}{\pi^{(2l-1)} 1_m}\right) \tilde{\pi}^{(2l-1)}$$

Similarly

$$\tilde{\pi}^{2l} = \text{diag}\left(\frac{q}{1_m^T \pi^{2l}}\right) \tilde{\pi}^{2l}$$

8.

Sinkhorn Generative Modelling

Paper: Learning generative modelling with Sinkhorn Divergence, AISTATS, 2018

Algorithm:

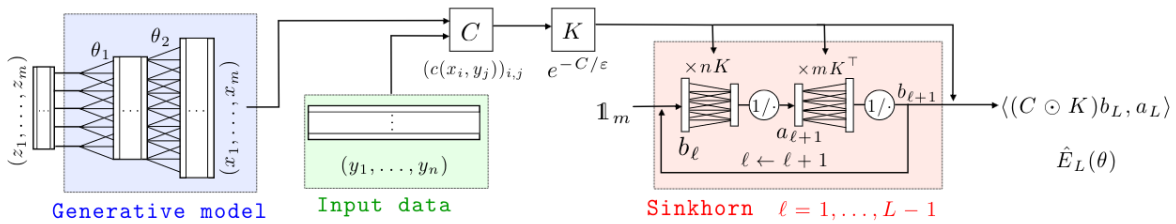
- Draw Mini-batches:
  $x_1, x_2, \ldots, x_B \sim p$
  $y_1, y_2, \ldots, y_B \sim p_\theta$

- Approximate $W_1(p, p_\theta) \approx W_1(\hat{p}, \hat{p}_\theta)$
  $$\hat{p}(x) = \frac{1}{B}\sum_{i=1}^{B} 1_{x_i = x}$$
  $$\hat{p}_\theta(x) = \frac{1}{B}\sum_{i=1}^{B} 1_{y_i = y}$$

- Estimate $W_1(\hat{p}, \hat{p}_\theta)$ using Sinkhorn Algorithm

- Compute Gradient Update:
  $\theta' = \theta - \eta \nabla_\theta W_1(\hat{p}, \hat{p}_\theta)$

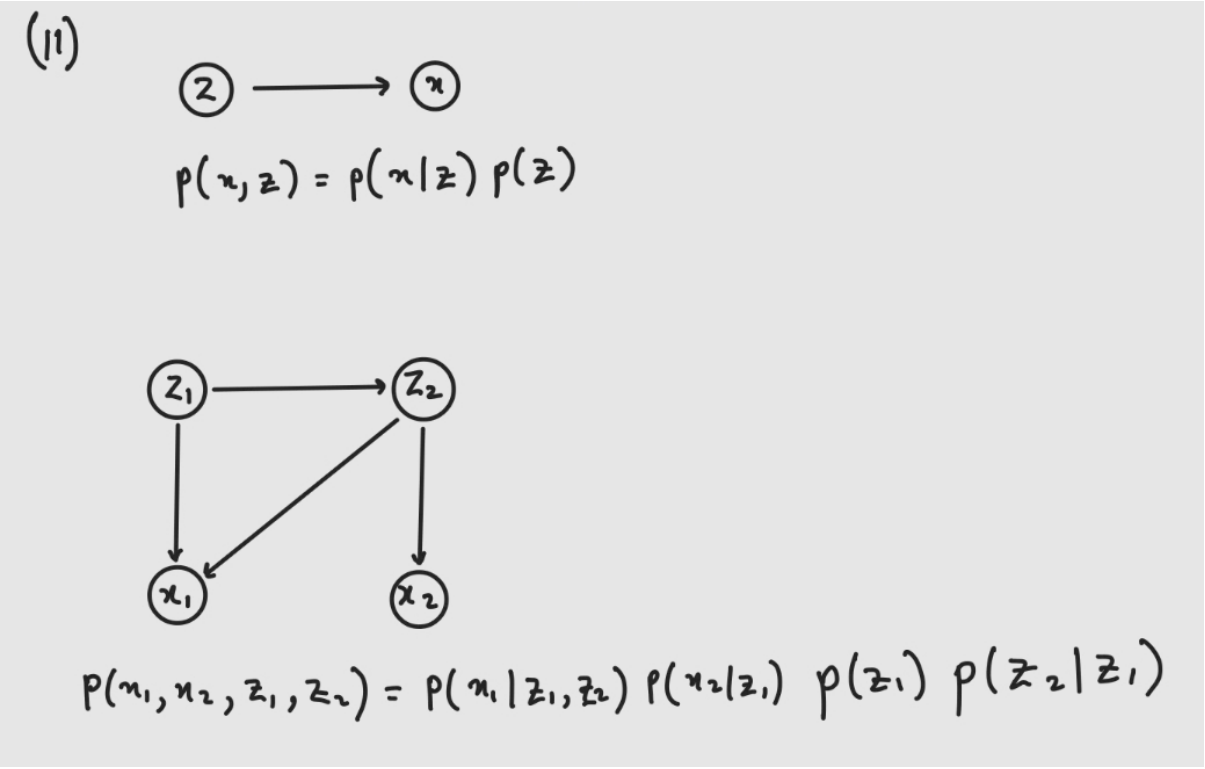
9.



10.

Probabilistic Graphical Models are powerful models that use graphs to describe dependency.

11.



12.

Variational Autoencoder (VAE) is called a latent variable model because it learns a compact and continuous representation of the input data, called the latent space, which captures the underlying structure and important features of the data.

13.

ELBO

Data likelihood: $p_\theta(x) = \int p_\theta(z)\, p_\theta(x|z)\, dz$

$p_\theta(x)$: we want to maximize this

$p_\theta(z)\, p_\theta(x|z)$: intractable

$\Rightarrow$ Using variational inference to approximate the unknown posterior distribution from only the observed data.

VAE objective is to maximize

$$\log(p_\theta(x)) = \mathop{\mathbb{E}}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x)\right] \qquad \left(p_\theta(x) \text{ doesn't depend on } z\right)$$

$$= \mathop{\mathbb{E}}_{z}\left[\log\left(\frac{p_\theta(x|z)\, p_\theta(z)}{p_\theta(z|x)}\right)\right] \qquad (\text{Bayes})$$

$$= \mathop{\mathbb{E}}_{z}\left[\log\left(\frac{p_\theta(x|z)\, p_\theta(z)}{p_\theta(z|x)} \cdot \frac{q_\phi(z|x)}{q_\phi(z|x)}\right)\right] \qquad (\text{importance sampling})$$

$$= \mathop{\mathbb{E}}_{z}\left[\log(p_\theta(x|z))\right] - \mathop{\mathbb{E}}_{z}\left[\log\left(\frac{q_\phi(z|x)}{p_\theta(z)}\right)\right] + \mathop{\mathbb{E}}_{z}\left[\log\left(\frac{q_\phi(z|x)}{p_\theta(z|x)}\right)\right]$$

$$= \mathop{\mathbb{E}}_{z}\left[\log p_\theta(x|z)\right] - D_{KL}\left[q_\phi(z|x) \,\|\, p_\theta(z)\right] + \underbrace{D_{KL}\left[q_\phi(z|x) \,\|\, p_\theta(z|x)\right]}_{\geq 0} \quad (p_\theta(z|x) \text{ intractable})$$

$$\geq \underbrace{\mathop{\mathbb{E}}_{z}\left[\log p_\theta(x|z)\right] - D_{KL}\left[q_\phi(z|x) \,\|\, p_\theta(z)\right]}$$

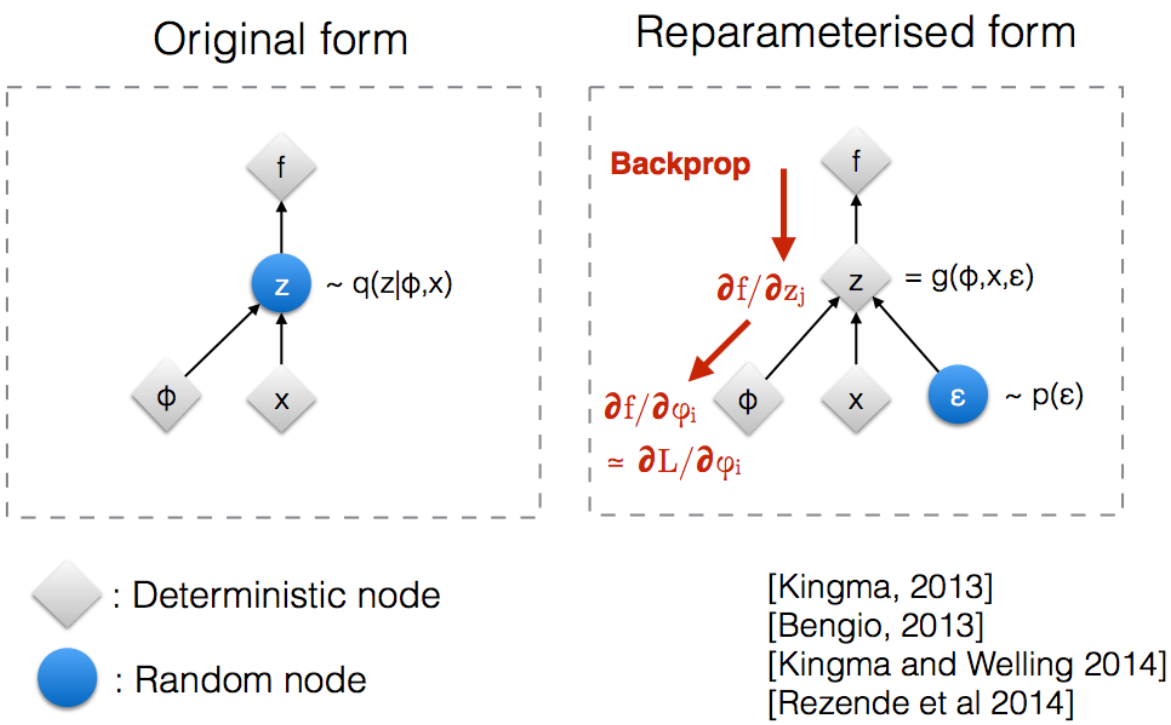
Tractable lower Bound (ELBO) $\hookrightarrow$ VAE objective function

VAE Loss = - VAE Objective

$$= - \text{ELBO}$$

$$= D_{KL}\left[q_\phi(z|x) \,\|\, p_\theta(z)\right] - \mathop{\mathbb{E}}_{z}\left[\log(p_\theta(x|z))\right]$$

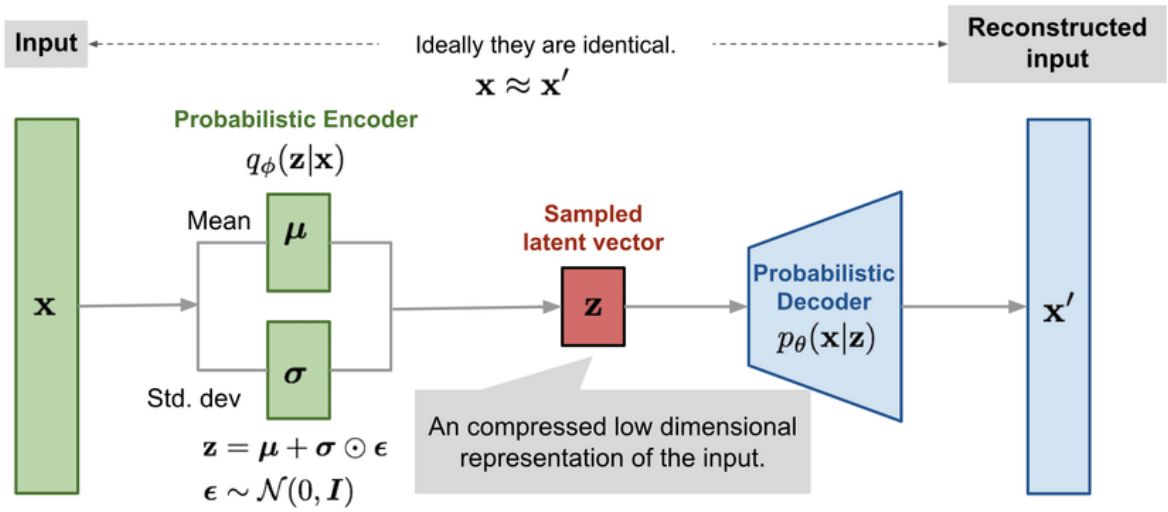
Re-parameterization



The reparametrization trick is a technique used in Variational Autoencoder (VAE) models to enable backpropagation through the stochastic layer, which is typically represented by a random sample from a normal distribution.

The reparametrization trick separates the random sampling operation from the model parameters, allowing gradients to flow through the stochastic layer and enabling efficient training via stochastic gradient descent.

Schematic Diagram



Training Procedure

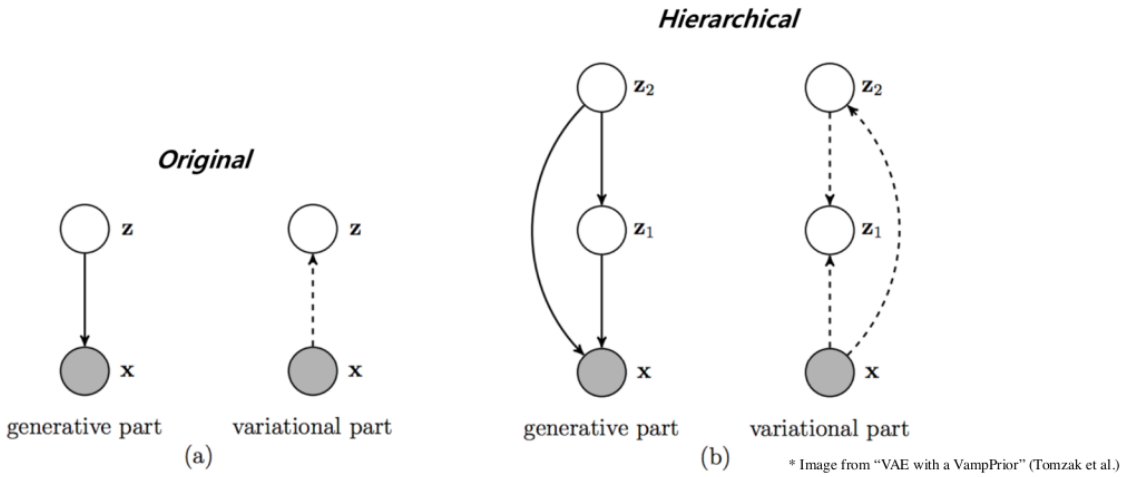
Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Initialize parameters
repeat
  $\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)
  $\boldsymbol{\epsilon} \leftarrow$ Random samples from noise distribution $p(\boldsymbol{\epsilon})$
  $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \widetilde{\mathcal{L}}^M(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^M, \boldsymbol{\epsilon})$ (Gradients of minibatch estimator (8))
  $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Update parameters using gradients $\mathbf{g}$ (e.g. SGD or Adagrad [DHS10])
until convergence of parameters $(\boldsymbol{\theta}, \boldsymbol{\phi})$
return $\boldsymbol{\theta}, \boldsymbol{\phi}$

14.

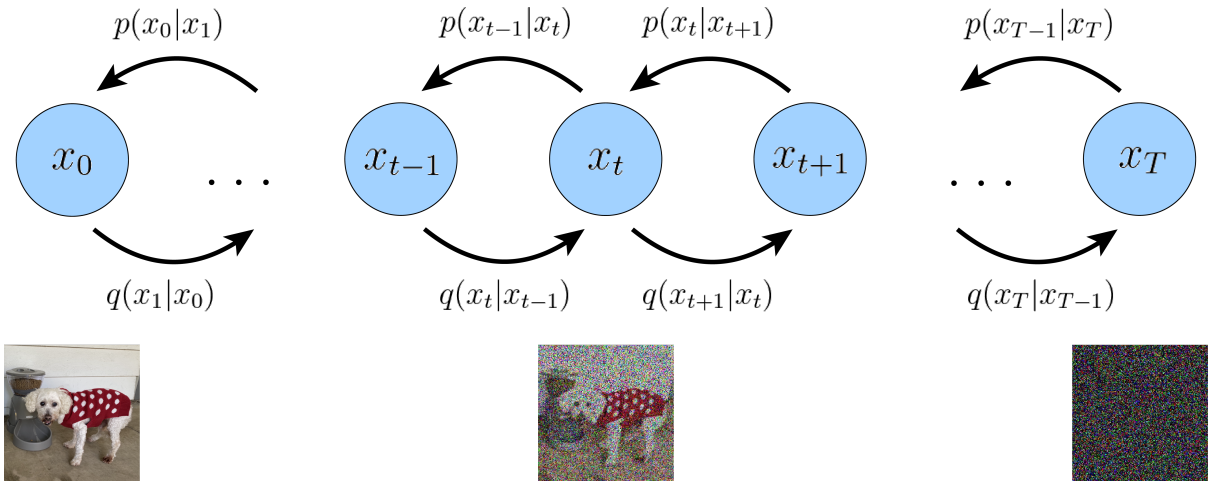
Hierarchical VAE (HVAE)

- Generalize VAE to multiple hierarchies over latent variables
- Latent variables themselves are generated from other higher level, more abstract latents.
- General HVAE is with T hierarchical levels, each latent is allowed to condition on all previous labels.



Markovian Hierarchical VAE (MHVAE)

- Assume the hierarchy to be Markovian.
- Decoding each latent Z_t only conditions on previous latent Z_{t+1}



15.

Papers of Variational Diffusion Models:

1. "Deep Unsupervised Learning using Non-equilibrium Thermodynamics", ICML 2015
2. "Denoising Diffusion Probabilistic Models", NeurIPS 2020

3. “Variational Diffusion Models”, Kingma, NeurIPS 2021

16.

Modelling Assumptions in VDM:

1. Latent Dimension is exactly equal to Data Dimension.
2. Structure of the latent encoder is not learnt at each time step.
   It is pre-defined as linear Gaussian model.
3. Gaussian parameters of the latent encoder vary over time in such a way that the distribution of the latent variable at the final time step T is a standard Gaussian.

17.

Implications of Modelling Assumptions in VDM:

1. Can represent both true data samples and latent variables as x_t
   where,
   x_0 = true data samples
   x_t = corresponding latent with hierarchy indexed by $t \in [1, T]$

   $$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

2. The distribution of each latent variable in the encoder is a Gaussian centred around its previous hierarchical latent.

   Mean
   $\mu_t(x_t) = \sqrt{\alpha_t} \cdot x_{t-1}$

   Variance
   $\Sigma_t(x_t) = (1 - \alpha_t)\, I$

   Where, the form of coefficient α_t are chosen such that the variance of the latent variables stay at a similar scale.

   a. Encoding process is variance preserving

b. Alternate parameterizations exist

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)\, I)$$

3. α_t evolves over time according to a fixed or learnable schedule, such that the distribution of the final latent $p(x_T)$ is a standard Gaussian.
   Hence, HVAE becomes:

$$p(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$
$$p(x_T) = \mathcal{N}(x_T; 0, I)$$

All assumptions together:

Steady notification of an input image over time, we progressively corrupt an image by adding Gaussian noise until it eventually becomes completely identical to pure Gaussian noise.

18.

- $p(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$
- $p(x_T) = \mathcal{N}(x_T; 0, I)$

19.

Diffusion models are a type of probabilistic model used for learning patterns in data. They work by simulating a process of gradual diffusion, where information is spread slowly and smoothly across the data space. This diffusion process is controlled by a set of learned parameters that determine the rate and direction of diffusion. By iteratively diffusing the data over many steps, the model is able to capture complex patterns and dependencies in the data, and can be used for tasks such as image generation, denoising, and inpainting. Overall, diffusion models provide a powerful tool for understanding and modeling complex data distributions.

20.

Like HVAE, VDM can be optimized by maximizing ELBO:

$$\log(p(x)) = \log \int p(x_{0:T})\, dx_{1:T}$$

$$= \log \int \frac{p(x_{0:T})\, q(x_{1:T}|x_0)}{q(x_{1:T}|x_0)}\, dx_{1:T} \qquad \left(\begin{matrix}\text{Importance}\\ \text{Sampling}\end{matrix}\right)$$

$$= \log \underset{q(x_{1:T}|x_0)}{E}\left[\frac{p(x_{0:T})}{q(x_{1:T}|x_0)}\right]$$

$$\geq \underset{q(x_{1:T}|x_0)}{E}\left[\log \frac{p(x_{0:T})}{q(x_{1:T}|x_0)}\right] \qquad (\text{Jensen's ineq})$$

$$= \underset{q(x_{1:T}|x_0)}{E}\left[\log \frac{p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)}{\prod_{t=1}^{T} q(x_t|x_{t-1})}\right]$$

make the indexing same

$$= \underset{q(x_{1:T}|x_0)}{E}\left[\log \frac{p(x_T)\, p_\theta(x_0|x_1)\prod_{t=2}^{T} p_\theta(x_{t-1}|x_t)}{q(x_T|x_{T-1})\prod_{t=1}^{T-1} q(x_t|x_{t-1})}\right]$$

$$= \underset{q(x_{1:T}|x_0)}{E}\left[\log \frac{p(x_T)\, p_\theta(x_0|x_1)\prod_{t=1}^{T-1} p_\theta(x_t|x_{t+1})}{q(x_T|x_{T-1})\prod_{t=1}^{T-1} q(x_t|x_{t-1})}\right]$$

$$= \underset{q(x_{1:T}|x_0)}{E}\left[\log \frac{p(x_T)\, p_\theta(x_0|x_1)}{q(x_T|x_{T-1})}\right]$$

$$+ \underset{q(x_{1:T}|x_0)}{E}\left[\log \prod_{t=1}^{T-1} \frac{p_\theta(x_t|x_{t+1})}{q(x_t|x_{t-1})}\right]$$

$$= \underset{q(x_{1:T}|x_0)}{E}\left[\log p_\theta(x_0|x_1)\right] + \underset{q(x_{1:T}|x_0)}{E}\left[\log \frac{p(x_T)}{q(x_T|x_{T-1})}\right]$$

$$+ \sum_{t=1}^{T-1} \underset{q(x_{t:T}|x_0)}{E}\left[\log \frac{p(x_T)}{q(x_T|x_{T-1})}\right]$$

simplify these to relevant vars

$$= \underset{q(x_1|x_0)}{E}\left[\log p_\theta(x_0|x_1)\right] + \underset{q(x_{T-1}, x_T|x_0)}{E}\left[\log \frac{p(x_T)}{q(x_T|x_{T-1})}\right]$$

$$+ \sum_{t=1}^{T-1} \underset{q(x_{t-1}, x_t, x_{t+1}|x_0)}{E}\left[\log \frac{p_\theta(x_t|x_{t+1})}{q(x_t|x_{t-1})}\right] \quad -(1)$$

↳ ELBO

Further Simplification

$$= \underset{q(x_1|x_0)}{E}\underbrace{\left[\log p_\theta(x_0|x_1)\right]}_{①} - \underset{q(x_{T-1}|x_0)}{E}\underbrace{\left[D_{KL}\left(q(x_T|x_{T-1}) \,\|\, p(x_T)\right)\right]}_{②}$$

$$- \sum_{t=1}^{T-1} \underset{q(x_{t-1}, x_{t+1}|x_0)}{E}\underbrace{\left[D_{KL}\left(q(x_t|x_{t-1}) \,\|\, p_\theta(x_t|x_{t+1})\right)\right]}_{③}$$

① → Reconstruction term

② → Prior Matching term

③ → Consistency term
or
denoising matching

21.

(21)

$$q(x_{t-1} \mid x_t, x_0) = \frac{q(x_t \mid x_{t-1})\, q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}$$

$$\Rightarrow q(x_{t-1} \mid x_t, x_0) = \frac{\mathcal{N}(x_t; \sqrt{\alpha_t}\, x_{t-1}, (1-\alpha_t) I)\; \mathcal{N}(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\, x_0, (1-\bar{\alpha}_{t-1}) I)}{\mathcal{N}(x_t, \sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t) I)}$$

$$\propto \exp\left\{-\left[\frac{(x_t - \sqrt{\alpha_t}\, x_{t-1})^2}{2(1-\alpha_t)} + \frac{(x_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\, x_0)^2}{2(1-\bar{\alpha}_{t-1})} - \frac{(x_t - \sqrt{\bar{\alpha}_t}\, x_0)^2}{2(1-\bar{\alpha}_t)}\right]\right\}$$

$$= \exp\left\{-\frac{1}{2}\, \frac{1}{\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)}}\left[\frac{x_{t-1}^2 - 2\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1}) x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\, x_{t-1}}{1-\bar{\alpha}_t}\right]\right\}$$

$$\propto \mathcal{N}\left(x_{t-1}; \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1}) x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t) x_0}{1-\bar{\alpha}_t}, \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} I\right)$$

$$\therefore q(x_{t-1} \mid x_t, x_0) \longrightarrow \text{Normal Distribution}$$

22.

(22)

$$\log p(w) \geq \underbrace{\mathop{E}_{q(x_1|x_0)}\left[\log p_\theta(x_0|x_1)\right]}_{\text{reconstruc}^n \text{ error}} - \underbrace{D_{KL}\left(q(x_T|x_0)\,||\,p(x_T)\right)}_{\text{prior matching term}}$$

$$- \sum_{t=2}^{T} \mathop{E}_{q(x_t|x_0)}\Big[\underbrace{D_{KL}\left(q(x_{t-1}|x_t, x_0)\,\|\,p_\theta(x_{t-1}|x_t)\right)}_{\text{denoising matching term}}\Big]$$

$$\arg\min_\theta \; D_{KL}\left(q(x_{t-1}|x_t, x_0)\,\|\,p_\theta(x_{t-1}|x_t)\right)$$

$$= \arg\min_\theta \; D_{KL}\left(\mathcal{N}(x_{t-1}; \mu_q, \underbrace{\Sigma_q(t)}) \,||\, \mathcal{N}(x_{t-1}; \mu_\theta, \underbrace{\Sigma_q(t)})\right)$$

(we set the variances of these terms to be exactly same.)

$$= \arg\min_\theta \; \frac{1}{2}\Big[\log\frac{|\Sigma_q(t)|}{|\Sigma_q(t)|} - d + tr(\underbrace{\Sigma_q(t)^{-1}\Sigma_q(t)}_{I \in \mathbb{R}^{d\times d}})$$

$$+ (\mu_\theta - \mu_q)^T \Sigma_q(t)^{-1}(\mu_\theta - \mu_q)\Big]$$

$$= \arg\min_\theta \; \frac{1}{2}\Big[(\mu_\theta - \mu_q)^T \underbrace{\Sigma_q(t)^{-1}}_{(\sigma_q^2(t)I)^{-1}}(\mu_\theta - \mu_q)\Big]$$

$$= \arg\min_\theta \; \frac{1}{2\sigma_q^2(t)}\Big[\|\mu_\theta - \mu_q\|_2^2\Big] \qquad -(1)$$

where,

$$\mu_\theta = \mu_\theta(x_t, t) \quad , \quad \mu_q = \mu_q(x_t, x_0)$$

Recall

$$\mu_q(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\alpha_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} \quad -(2)$$

As $\mu_\theta(x_t)$ also cond^{n}s on x_t, we can match $\mu_q(x_t, x_0)$ closely by setting it as:

$$\mu_\theta(x_t, t) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\hat{x}_\theta(x_t, t)}{1-\bar{\alpha}_t} \quad -(3)$$

where, $\hat{x}_\theta(x_t, t)$ → parameterized by NN to predict x_0 from noisy image x_t at time index 't'.

Using (5), (3) in (1)

$$\Rightarrow \underset{\theta}{\operatorname{argmin}} \frac{1}{2\sigma_q^2(t)} \frac{\bar{\alpha}_{t-1}(1-\alpha_t)^2}{(1-\bar{\alpha}_t)^2} \left[\left\| \hat{x}_\theta(x_t, t) - x_0 \right\|_2^2 \right]$$

23.

(23)

Model α_t using a neural network $\hat{\alpha}_\eta(t)$, η parameters

$$\frac{1}{2\sigma_q^2(t)} \frac{\bar{\alpha}_{t-1}(1-\alpha_t)^2}{(1-\bar{\alpha}_t)^2} \left[\| \hat{x}_\theta(x_t, t) - x_0 \|_2^2 \right]$$

$$\hookrightarrow \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$$

$$= \frac{1}{2} \left(\frac{\bar{\alpha}_{t-1}}{1-\bar{\alpha}_{t-1}} - \frac{\bar{\alpha}_t}{1-\bar{\alpha}_t} \right) \left[\| \hat{x}_\theta(x_t, t) - x_0 \|_2^2 \right] \quad -①$$

Recall

$$q(x_t | x_0) \sim N(x_t; \sqrt{\alpha_t}\, x_0, (1-\bar{\alpha}_t) I)$$

Define signal-to-noise ratio (SNR) $= \frac{\mu_2^2}{\sigma^2}$

$$SNR(t) = \frac{\bar{\alpha}_t}{1-\bar{\alpha}_t} \quad -②$$

Using ② in ①

$$= \frac{1}{2} \left(SNR(t-1) - SNR(t) \right) \left[\| \hat{x}_\theta(x_t, t) - x_0 \|_2^2 \right]$$

24.

(24.)

Solving for x_0:

$$x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_0}{\sqrt{\bar{\alpha}_t}} \quad -(1)$$

$$\mu_q(x_t, x_0) = \frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})\,x_t + \sqrt{\bar{\alpha}_{t-1}}\,(1-\alpha_t)\,x_0}{1-\bar{\alpha}_t}$$

Using (1)

$$= \frac{1}{\sqrt{\alpha_t}}\,x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\,\sqrt{\alpha_t}}\,\epsilon_0 \quad -(2)$$

Using (2) we can approximate $\mu_\theta(x_t, t)$:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\,x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\,\sqrt{\bar{\alpha}_t}}\,\hat{\epsilon}_\theta(x_t, t) \quad -(3)$$

Hence corresponding optimization problem becomes:

$$\underset{\theta}{\operatorname{argmin}}\; D_{KL}\left(q(x_{t-1} \mid x_t, x_0) \;\|\; p_\theta(x_{t-1} \mid x_t)\right)$$

$$= \underset{\theta}{\operatorname{argmin}}\; D_{KL}\left(\mathcal{N}(x_{t-1}; \mu_q, \Sigma_q(t)) \;\|\; \mathcal{N}(x_{t-1}; \mu_\theta, \Sigma_q(t))\right)$$

Using (2), (3)

$$= \boxed{\frac{1}{2\sigma_q^2(t)}\;\frac{(1-\alpha_t)^2}{(1-\bar{\alpha}_t)\alpha_t}\left[\left\|\epsilon_0 - \hat{\epsilon}_\theta(x_t - t)\right\|_2^2\right]}$$

25.

(25.)

Tweedie's formula:

$$E[\mu_z | z] = z + \Sigma_z \nabla_z \log p(z) \quad -①$$

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t) I)$$

Using ①

$$E[\mu_{x_t}|x_t] = x_t + (1-\bar{\alpha}_t) \nabla_{x_t} \log p(x_t)$$

$$\Rightarrow \sqrt{\bar{\alpha}_t}\, x_0 = x_t + (1-\bar{\alpha}_t) \nabla \log p(x_t) \quad -②$$

↓

Using x_0 in ground truth denoising transition $\mu_q(x_t, x_0)$:

$$\mu_q(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\, x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t) x_0}{1-\bar{\alpha}_t}$$

Using ②

$$\Rightarrow \mu_q(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}} x_t + \frac{1-\alpha_t}{\sqrt{\alpha_t}} \nabla \log p(x_t) \quad -③$$

↓ using ③

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} x_t + \frac{1-\alpha_t}{\sqrt{\alpha_t}} S_\theta(x_t, t) \quad -④$$

Using ②

$$\Rightarrow \mu_q(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}} x_t + \frac{1-\alpha_t}{\sqrt{\alpha_t}} \nabla \log p(x_t) \quad -③$$

(using ③)

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} x_t + \frac{1-\alpha_t}{\sqrt{\alpha_t}} S_\theta(x_t, t) \quad -④$$

Corresponding optn. problem

$$\underset{\theta}{\text{argmin}} \; D_{KL}\left(q(x_{t-1} | x_t, x_0) \;||\; p_\theta(x_{t-1} | x_t)\right)$$

$$= \underset{\theta}{\text{argmin}} \; D_{KL}\left(N(x_{t-1}, \mu_q, \Sigma_q(t)) \;||\; N(x_{t-1}; \mu_\theta, \Sigma_\theta(t))\right)$$

Using ③, ④

$$\underset{\theta}{\text{argmin}} \; \frac{1}{2\sigma_q^2(t)} \frac{(1-\alpha_t)^2}{\alpha_t} \left[\| S_\theta(x_t, t) - \nabla \log p(x_t) \|_2^2 \right]$$

$S_\theta(x_t, t) \rightarrow$ NN that predicts score funⁿ

26.

(26)

Score func^n $\nabla_{x_t} \log p(x_t)$ looks similar to source noise ϵ_0.

From reparametr.

$$x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_0}{\sqrt{\bar{\alpha}_t}} \qquad \left(\begin{matrix}\text{recal reparam.}\\ q(x_t|x_0)\end{matrix}\right) \quad -(1)$$

. From tweedie's formula

$$x_0 = \frac{x_t + (1-\bar{\alpha}_t)\nabla \log p(x_t)}{\sqrt{\bar{\alpha}_t}} \quad -(2)$$

(1) = (2)

$$\nabla \log p(x_t) = \frac{-1}{\sqrt{1-\bar{\alpha}_t}}\epsilon_0$$ ← original source noise

27.

(27)

Implicit Score Matching (Hyvarinen, 2005)

$$\operatorname{argmin}_\theta \mathop{E}_{x\sim p}\left[\frac{1}{2}\|S_\theta(x) - \nabla_x \log p(x)\|_2^2\right] = \operatorname{argmin}_\theta \mathop{E}_{x\sim p}\left[\operatorname{tr}(\nabla_x S_\theta(x)) + \frac{1}{2}\|S_\theta(x)\|_2\right]$$

Proof:

$$\operatorname{argmin}_\theta \mathop{E}_{x\sim p}\left[\frac{1}{2}\|S_\theta(x) - \nabla_x \log(p(x)\|_2^2\right]$$

$$= \operatorname{argmin}_\theta \mathop{E}_{x\sim p}\left[\frac{1}{2}\|S_\theta(x)\|^2 - 2\langle S_\theta(x), \nabla_x \log p(x)\rangle + \underbrace{\|\nabla_x \log p(x)\|_2^2}_{\text{independent of } \theta}\right]$$

Just need to show inner product term is equivalent to $\operatorname{tr}(\nabla_x S_\theta(x))$

$$\mathop{E}_{x\sim p}\left[S_\theta(x)^T \nabla_x \log p(x)\right] = \sum_{i=1}^{d} \int_x S_\theta(x)_i \frac{\delta \log p(x)}{x_i} p(x)\,dx$$

$$= -\sum_{i=1}^{d} \int_x \frac{S_\theta(x)_i}{\delta x_i} p(x)\,dx$$

$$= -\int_x \operatorname{tr}(\nabla_x S_\theta(x))\, p(x)\,dx$$

$$= -\mathop{E}_{x\sim p}\left[\operatorname{tr}(\nabla_x S_\theta(x))\right]$$

M.P

28.

(28)

Conditional Diffusion Models

$$p(x_{0:T} | y) = p(x_T) \prod_{i=1}^{T} p_\theta(x_{t-1} | x_t, y_t)$$

Classifier Guidance

explicitly control amount of weight to cond^n.

Baye's rule

$$\nabla_{x_t} \log p(x_t|y) = \nabla_{x_t} \log \left(\frac{p(x_t)\, p(y|x_t)}{p(y)} \right)$$

$$= \underbrace{\nabla_{x_t} \log p(x_t)}_{\text{unconditional score}} + \underbrace{\nabla_{x_t} \log p(y|x_t)}_{\text{adverserial gradient}} - \cancel{\nabla_{x_t} \log p(y)} \to 0$$

→ score of uncond^n diff mode ($\nabla_{x_t} \log p(x_t)$) is learned.

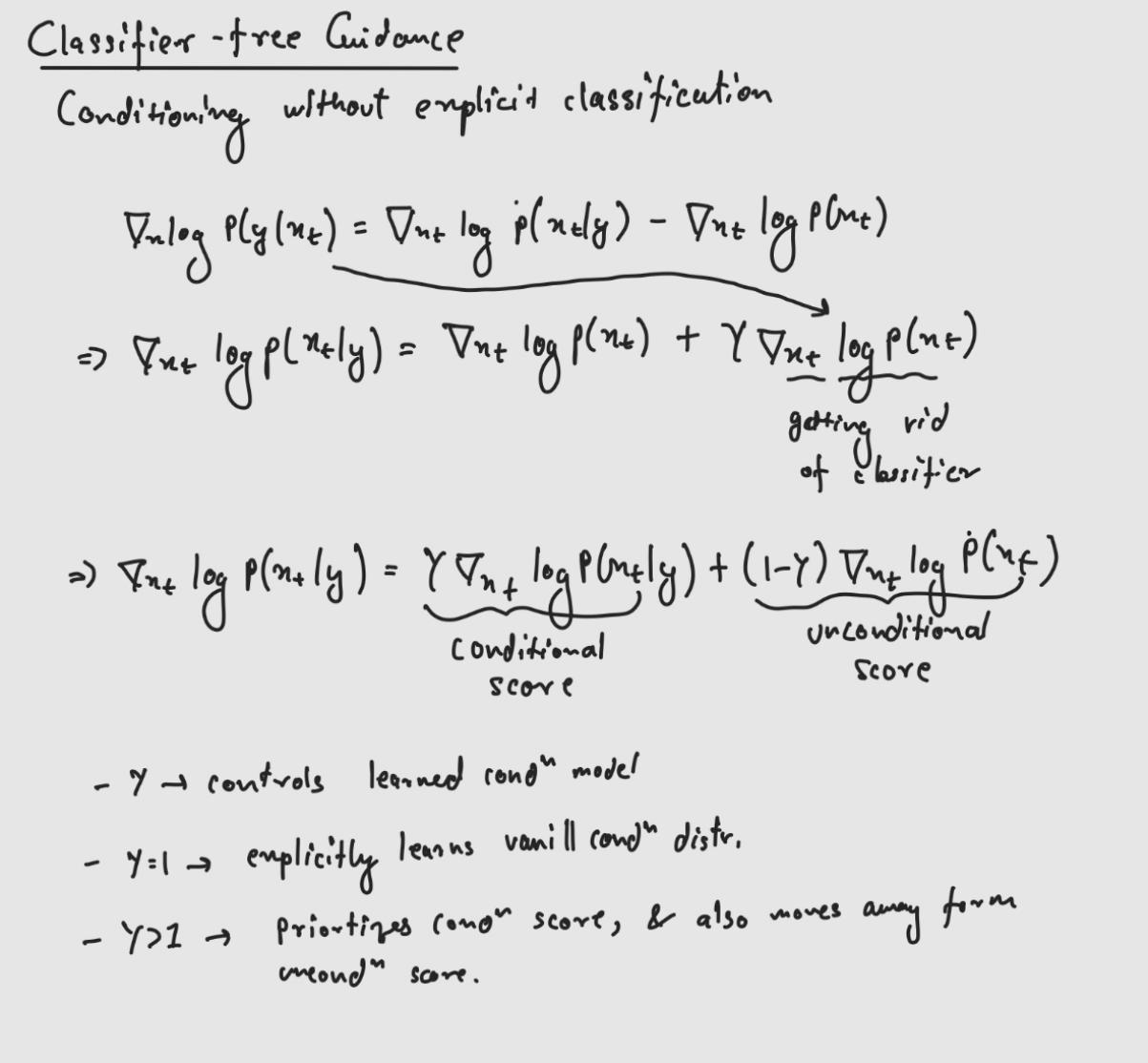
→ Alongside, classifier that takes arbitrary noise 'x_t' attempts to predict 'y'.

Introduce fine-grained control:

$$\nabla \log p(x_t|y) = \nabla \log p(x_t) + \boxed{\gamma}\, \nabla \log p(y|x_t)$$

Drawbacks

- Requires seperately learnt Classifier
- Classifier must handle arbitrary noise inputs, which most pretrained models are unable to do.



29.

Autoregressive Flows	Normalizing Flows
1) AR model the joint distribution using chain of conditional distribution	NF model probability distribution by applying a series of invertible transforms.
2) Are computationally efficient, but may suffer from slow convergence and are limited to modelling low-dim data.	NF are flexible in modelling high-dim data, but can be computationally expensive due to the need to evaluate the determinant of the Jacobian in each transformation.
3) Conditional distributions are typically modelled as Gaussians or discrete distributions.	In NF, transformations can be any invertible functions.

30.

Linear Autoregressive Models

$\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ (i.i.d), consider a scalar seperatio of observations $x \in \mathbb{R}^{T \times 1}$ defined by the linear dynamics:

$x_t = \rho(x_{t-1} - \mu) + \mu + \epsilon_t$

$x_t = \mu + \epsilon_o$

There dynamics are referred to as autoregressive process of order 1, or AR(1)

[Ref: “A course in time series analysis”]

- Order - 1, referes to the fact that x_t is a function of only one previous observation x_{t-1}
- Also know as Markov property

31.

Wide-Sense Stationary Process

A tie series X is called wide-sense stationary if it has stationary mean and $Cov(x_t, x_{t-k}) = Cov(x_s, x_{s-l}), \forall s, t, k \in N$

Define covairance function, $\gamma : N \to \mathbb{R}$

$\gamma(k) = Cov(x_{k+1}, x_1)$

32.

Fully Visible Sigmoid Belief Network

[Neal, “Connectionist learning of belief networks”, 1992]

Def.:

Let $P \in \mathbb{R}^T$, and define $f_t : \mathbb{R}^{(t-1)} \to \mathbb{R}$ by:

$f_t(x_{ct}) =< p_{ct}, x_{ct} >$

Then fully visible sigmoid belief network is given by:

$x_t \sim \text{Bernoulli}(\text{sigmoid}(f_t(x_{ct}))$

$x_1 \sim \text{Bernoulli}(P_o)$

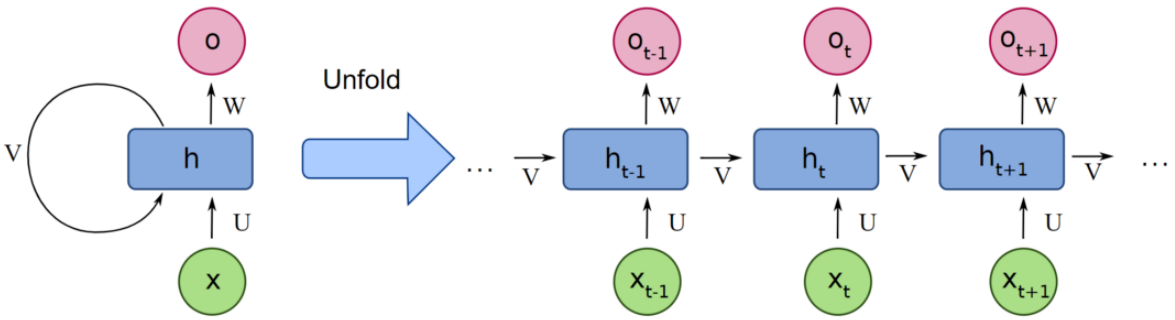
The goal is to predict $x_t \in \{0, 1\}$ using a log-linear combination of x_{ct}

Neural Autoregressive Distribution Estimation

Neural Autoregressive Distribution Estimation, on the other hand, is a type of autoregressive model where the joint probability distribution of the data is modeled as a product of conditional probabilities, one for each dimension of the data. In NADE, the conditional probability of each dimension is modeled using a feedforward neural network that takes as input the previous dimensions and outputs the conditional probability of the current dimension.

33.

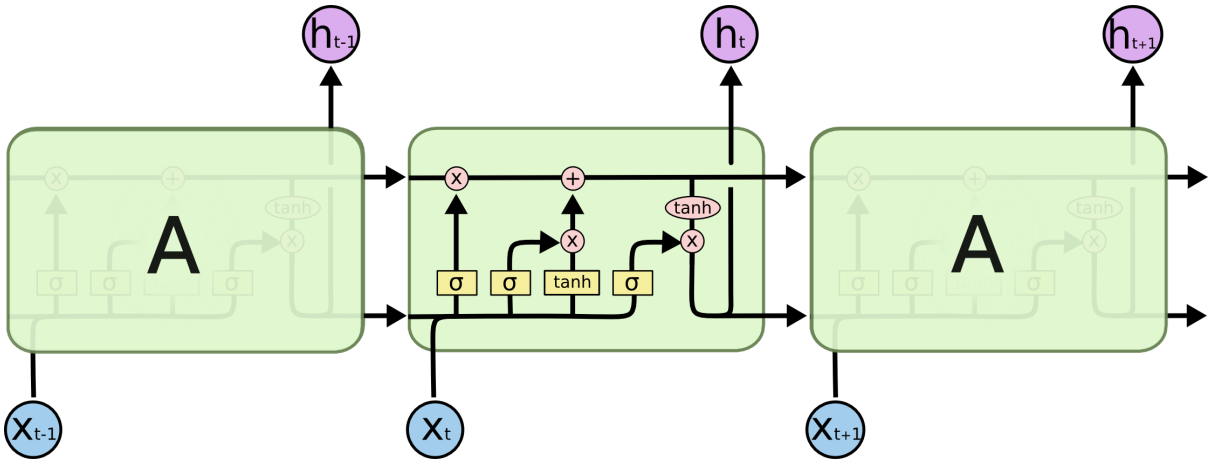
RNN



- RNNs are a type of neural network that can model sequential data by maintaining an internal state
- That is updated at each time step based on the current input and the previous hidden state.

- The output of an RNN at each time step is a function of the current hidden state and the input at that time step.
- The RNN is trained using backpropagation through time
- RNNs can suffer from the vanishing gradient problem, which occurs when the gradients become very small as they propagate backwards through time. This can lead to difficulties in learning long-term dependencies and can limit the performance of the network.

LSTM



- LSTM networks are a type of RNN that use memory cells and gates to control the flow of information in the network.
- The memory cell is a long-term memory unit that can store information over many time steps, and the gates control the information flow into and out of the memory cell.
- The gates consist of sigmoidal functions that output values between 0 and 1, which are used to decide which information to keep and which to discard.
- LSTM networks have been shown to be effective in modeling long-term dependencies in sequential data and have been used in various applications such as speech recognition, natural language processing, and image captioning.
- LSTM networks are trained using backpropagation through time with gradient clipping to prevent exploding gradients.

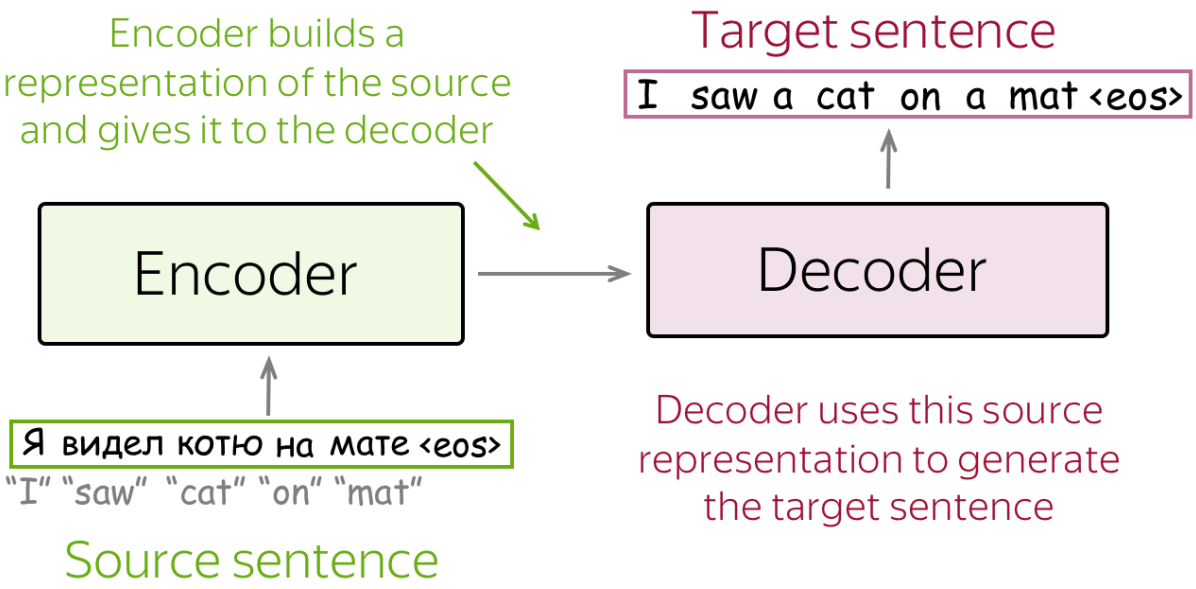
34.

Advantage of LSTM over RNN

- The main advantage of LSTM (Long Short-Term Memory) networks over traditional RNN (Recurrent Neural Network) models is their ability to effectively capture long-term dependencies in sequential data.
- In traditional RNNs, the hidden state at each time step is a function of the current input and the previous hidden state. However, as the sequence length increases, the gradients can become very small or even vanish
- LSTM networks address this issue by introducing a memory cell and several gates that control the flow of information into and out of the cell. The gates are designed to selectively forget or remember information based on the current input and the previous hidden state

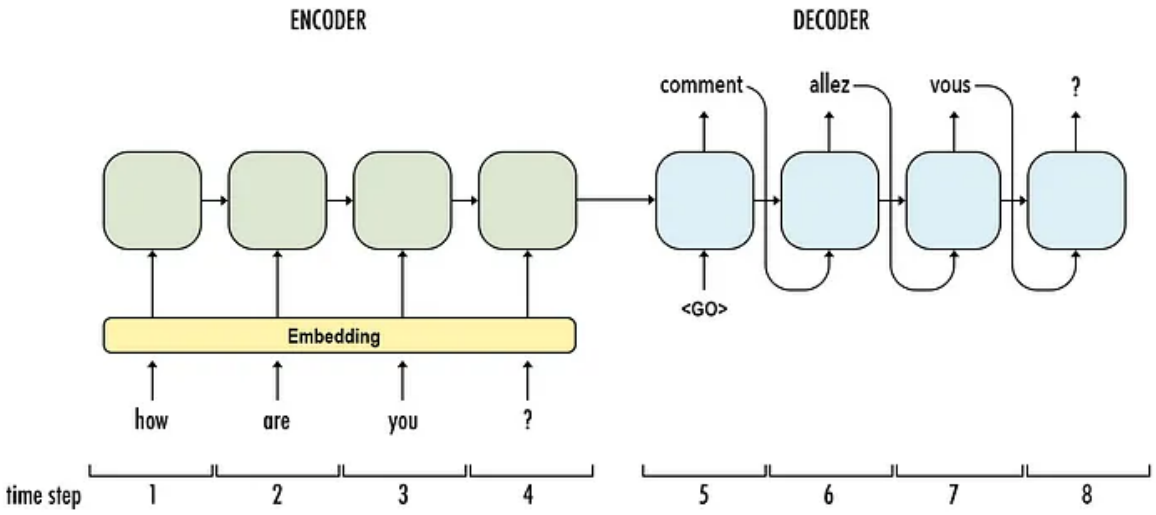
35.

Seq2Seq Model



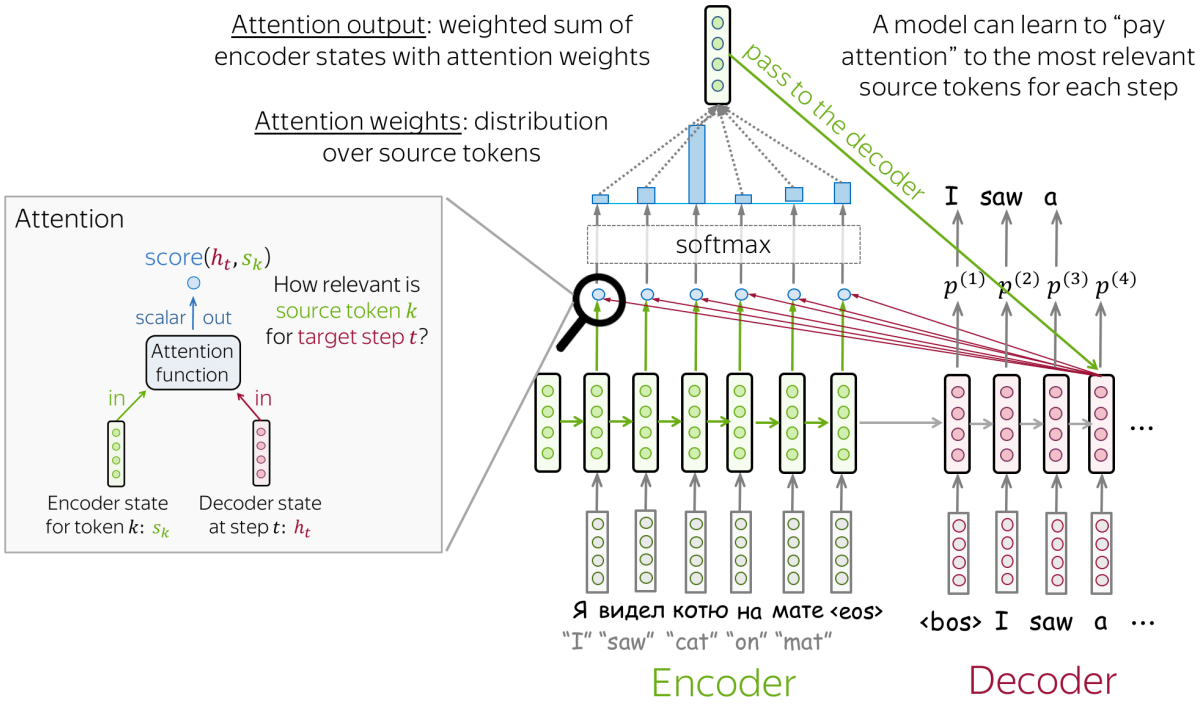
- Sequence-to-sequence (seq2seq) models are a type of neural network architecture used for tasks that involve transforming an input sequence into an output sequence, such as machine translation, text summarization, or speech recognition.

- Seq2seq models consist of an encoder and a decoder
- The encoder maps the input sequence to a fixed-length vector representation
- The decoder generates the output sequence from this representation.

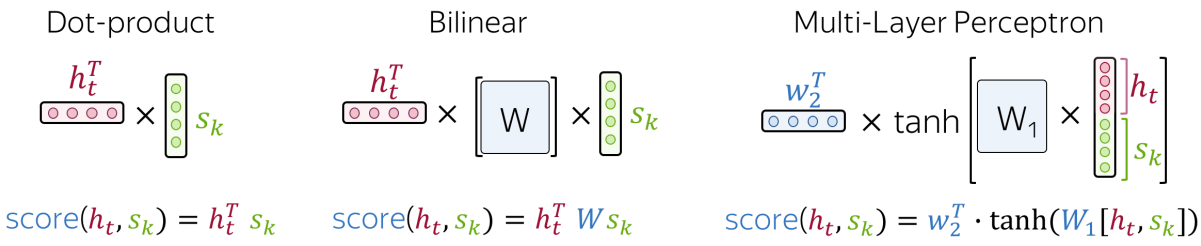


Seq2Seq Model + Attention

- Seq2seq models with attention, also known as attentional seq2seq models, extend the basic seq2seq architecture by adding an attention mechanism to the decoder.
- The attention mechanism allows the decoder to selectively focus on different parts of the input sequence, depending on the current output generated by the decoder.
- The attention mechanism works by computing a set of attention weights for each input sequence element, which indicate how much attention the decoder should pay to that element when generating the current output.
- The attention weights are computed as a function of the decoder's current hidden state and the encoder's output at each time step.



- The computation of attention weights can be done in different ways, but a common approach is to use a dot product between the decoder's current hidden state and the encoder's output at each time step
- Followed by a softmax operation to obtain a probability distribution over the input sequence.
- The weighted sum of the encoder's output, using the attention weights as weights, is then concatenated with the decoder's input at the current time step, and fed into the decoder.



36.

Tokenisation is the task of splitting the text into tokens which are then converted to numbers. These numbers are in turn used by the machine learning models for further processing and training.

Subword tokenization algorithms rely on the principle that frequently used words should not be split into smaller subwords, but rare words should be decomposed into meaningful subwords.

For instance `"annoyingly"` might be considered a rare word and could be decomposed into `"annoying"` and `"ly"`. Both `"annoying"` and `"ly"` as stand-alone subwords would appear more frequently while at the same time the meaning of `"annoyingly"` is kept by the composite meaning of `"annoying"` and `"ly"`.

This is especially useful in agglutinative languages such as Turkish, where you can form (almost) arbitrarily long complex words by stringing together subwords.

Subword tokenization allows the model to have a reasonable vocabulary size while being able to learn meaningful context-independent representations. In addition, subword tokenization enables the model to process words it has never seen before, by decomposing them into known subwords.

37.

Transformer Architecture

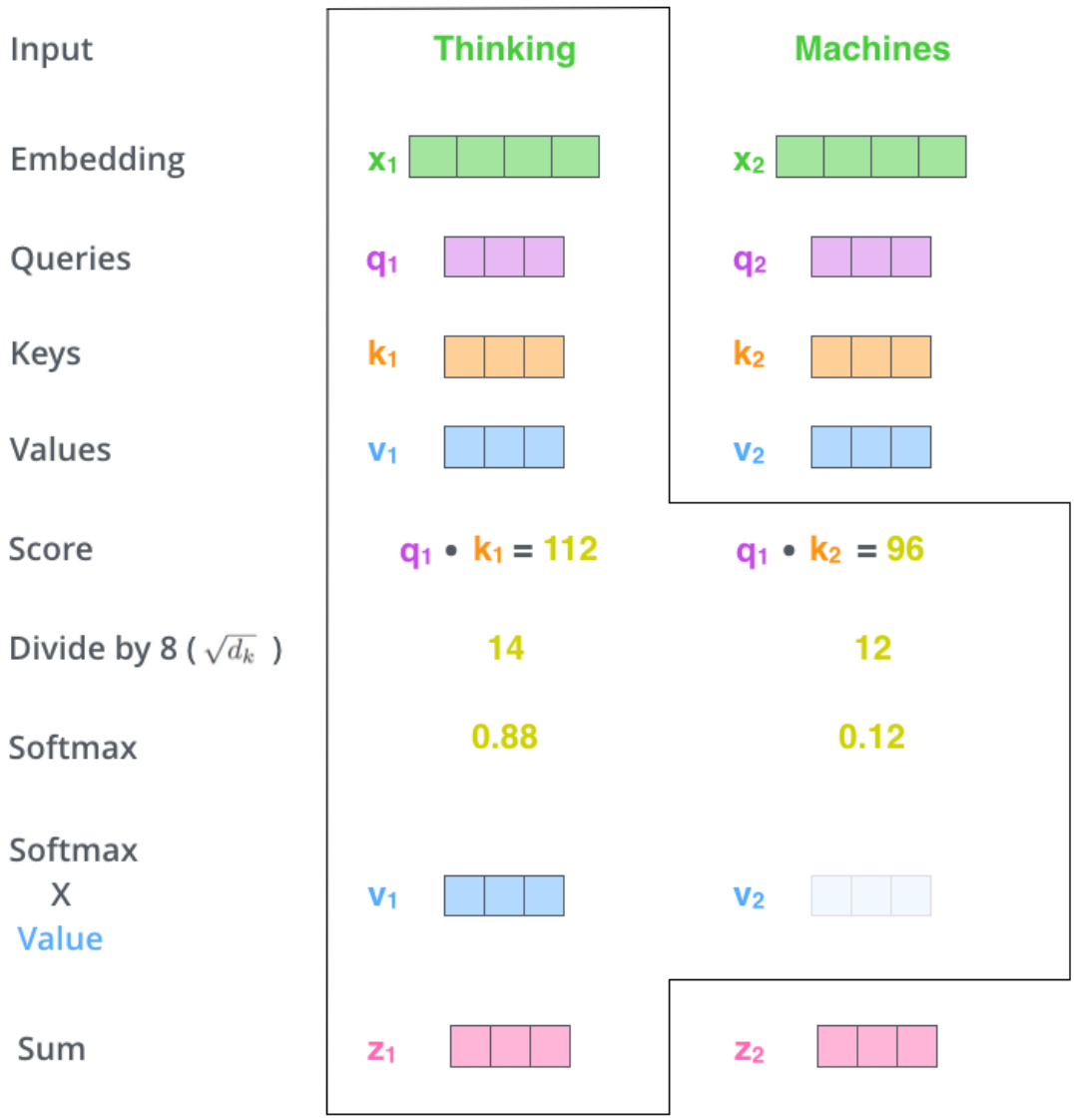
Scaled Self-Attention

- The **first step** in calculating self-attention is to create three vectors from each of the encoder's input vectors (in this case, the embedding of each word). So for each word, we create a Query vector, a Key vector, and a Value vector. These vectors are created by multiplying the embedding by three matrices that we trained during the training process.
- The **second step** in calculating self-attention is to calculate a score. Say we're calculating the self-attention for the first word in this example, "Thinking". We need to score each word of the input sentence against this word. The score determines how much focus to place on other parts of the input sentence as we

encode a word at a certain position.

The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring. So if we're processing the self-attention for the word in position #1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.

- The **third and fourth steps** are to divide the scores by 8 (the square root of the dimension of the key vectors used in the paper – 64. This leads to having more stable gradients. There could be other possible values here, but this is the default), then pass the result through a softmax operation. Softmax normalizes the scores so they're all positive and add up to 1.
- The **fifth step** is to multiply each value vector by the softmax score (in preparation to sum them up). The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words (by multiplying them by tiny numbers like 0.001, for example).
- The **sixth step** is to sum up the weighted value vectors. This produces the output of the self-attention layer at this position (for the first word).

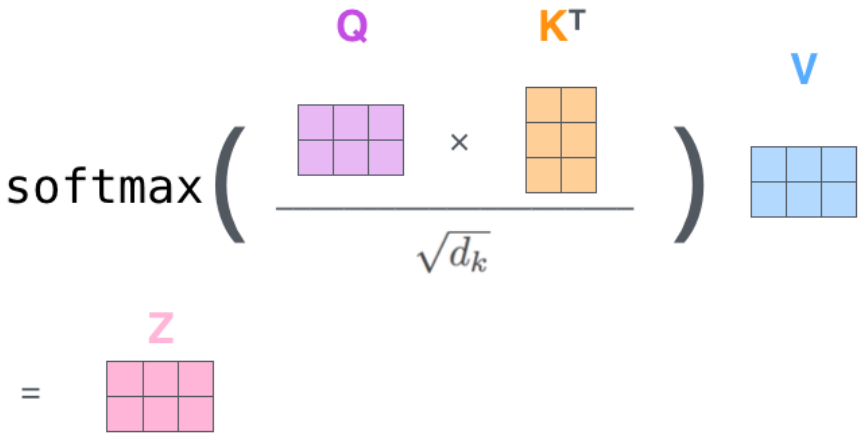

Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 (√dk)
14
12
Softmax
0.88
0.12
Softmax X Value
v1
v2
Sum
z1
z2

Matrix Form

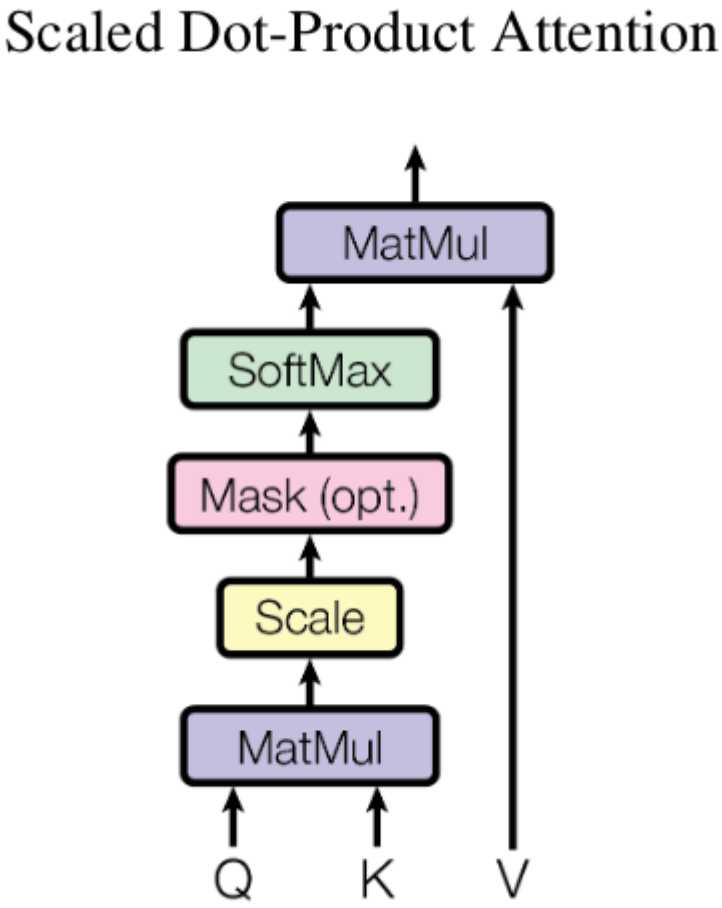

X
W^Q
Q
×
=


X
W^K
K
×
=


X
W^V
V
×
=



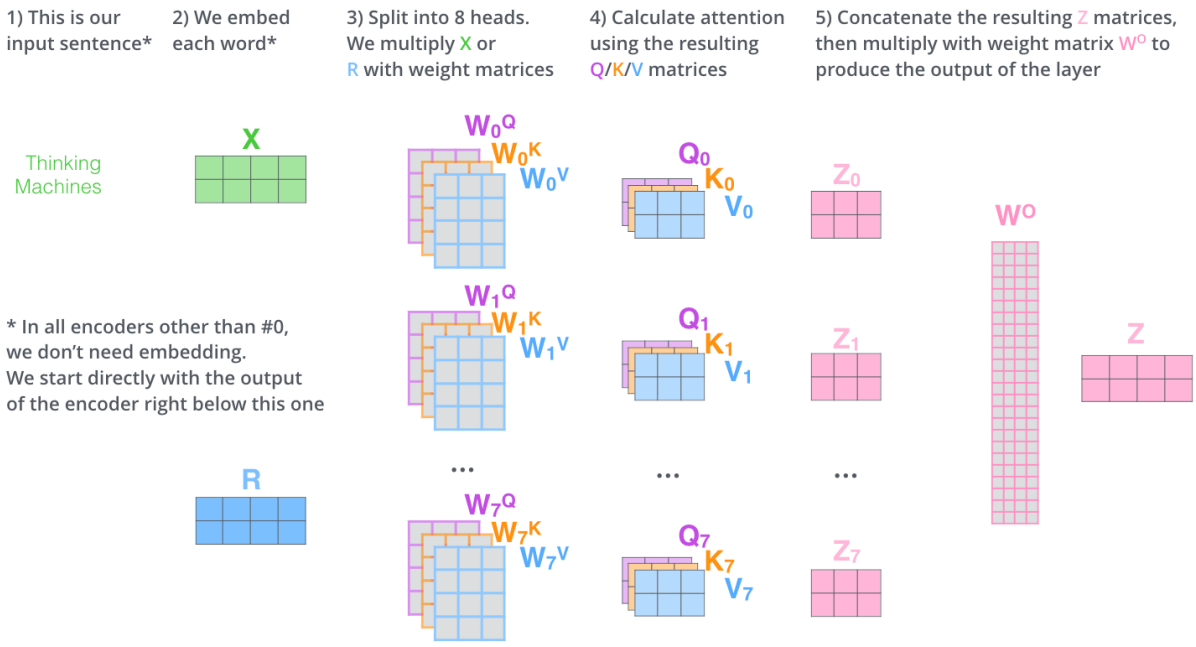
Architecture



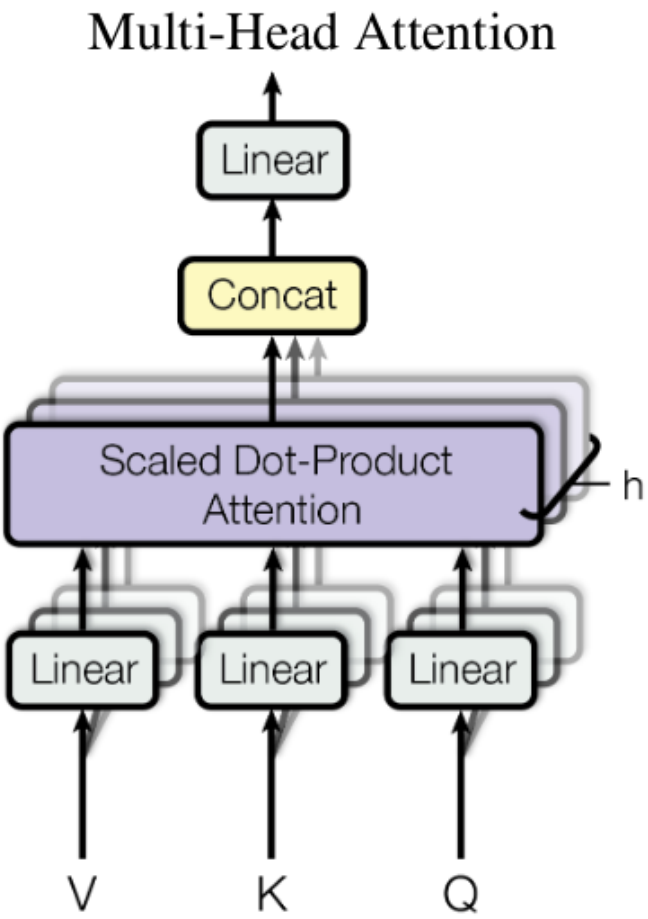
Multi-Head Attention

The paper further refined the self-attention layer by adding a mechanism called “multi-headed” attention. This improves the performance of the attention layer in two ways:

1. It expands the model’s ability to focus on different positions. Yes, in the example above, z1 contains a little bit of every other encoding, but it could be dominated by the actual word itself. If we’re translating a sentence like “The animal didn’t cross the street because it was too tired”, it would be useful to know which word “it” refers to.
2. It gives the attention layer multiple “representation subspaces”. As we’ll see next, with multi-headed attention we have not only one, but multiple sets of Query/Key/Value weight matrices (the Transformer uses eight attention heads, so we end up with eight sets for each encoder/decoder). Each of these sets is randomly initialized. Then, after training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace.



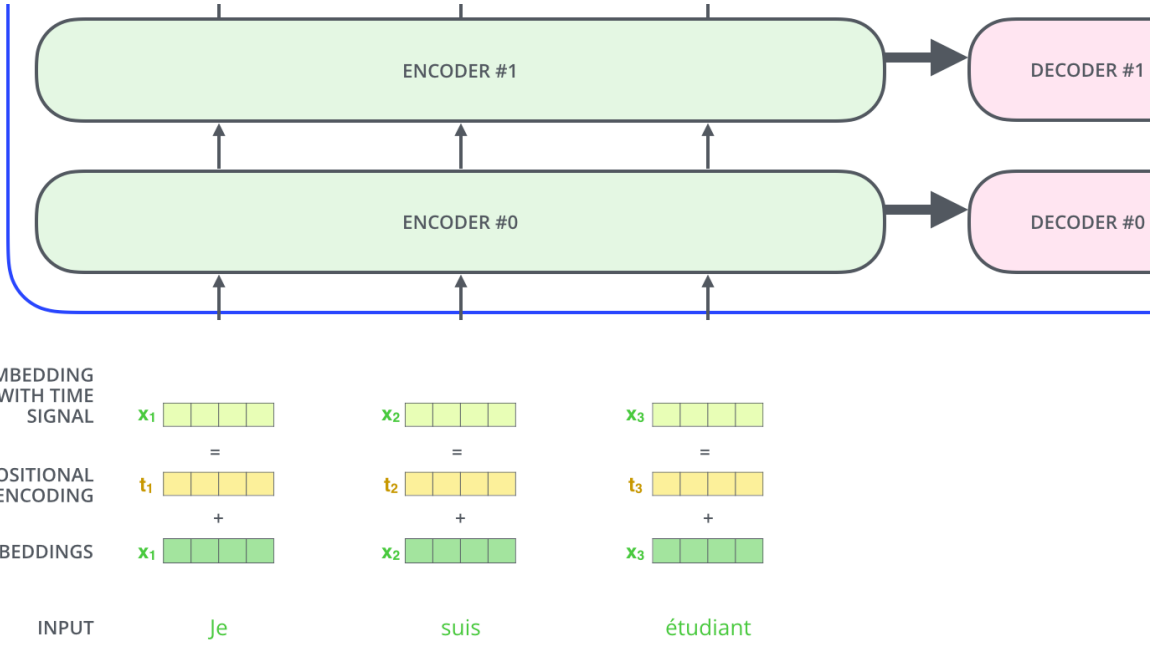
Architecture

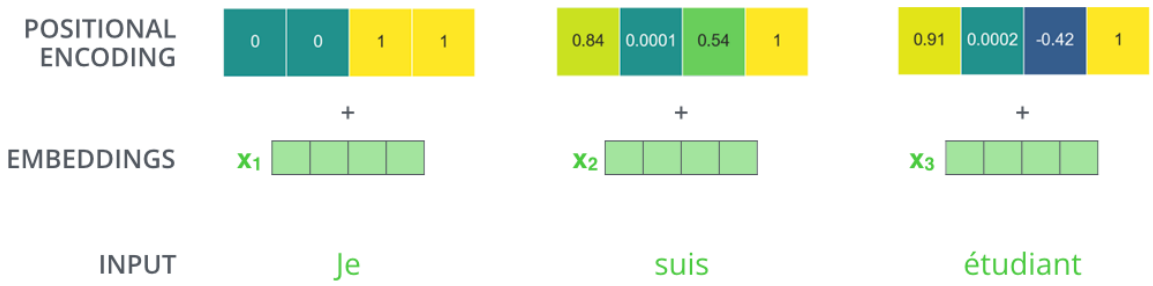


Positional Encoding

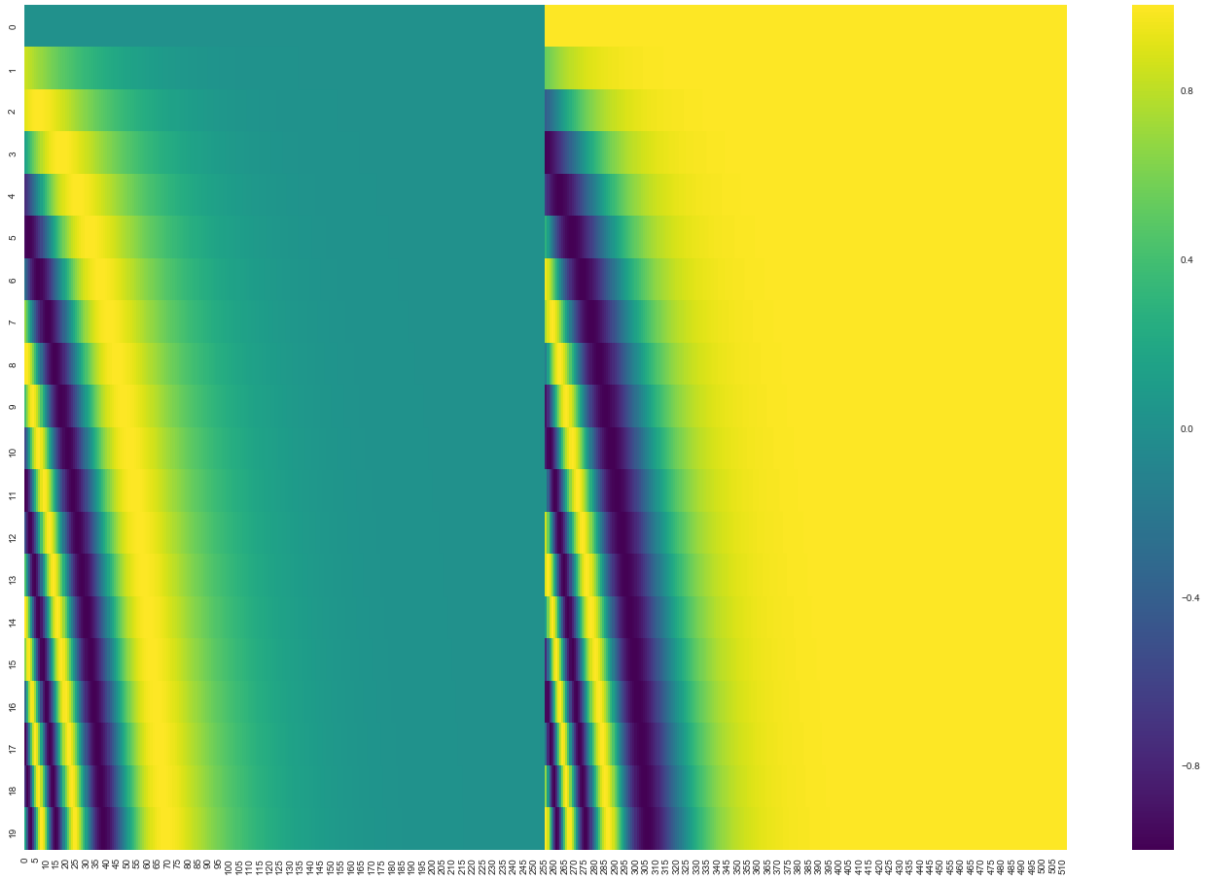
One thing that's missing from the model as we have described it so far is a way to account for the order of the words in the input sequence.

To address this, the transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

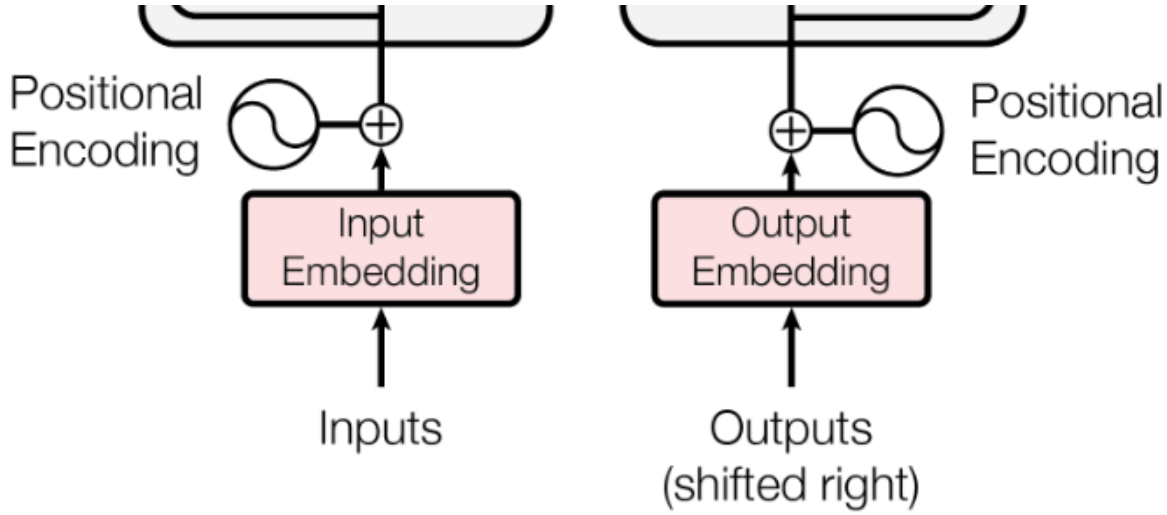




In the following figure, each row corresponds to a positional encoding of a vector. So the first row would be the vector we'd add to the embedding of the first word in an input sequence. Each row contains 512 values – each with a value between 1 and -1. We've color-coded them so the pattern is visible.



In Architecture



Layer-Normalzation and Skip Connections

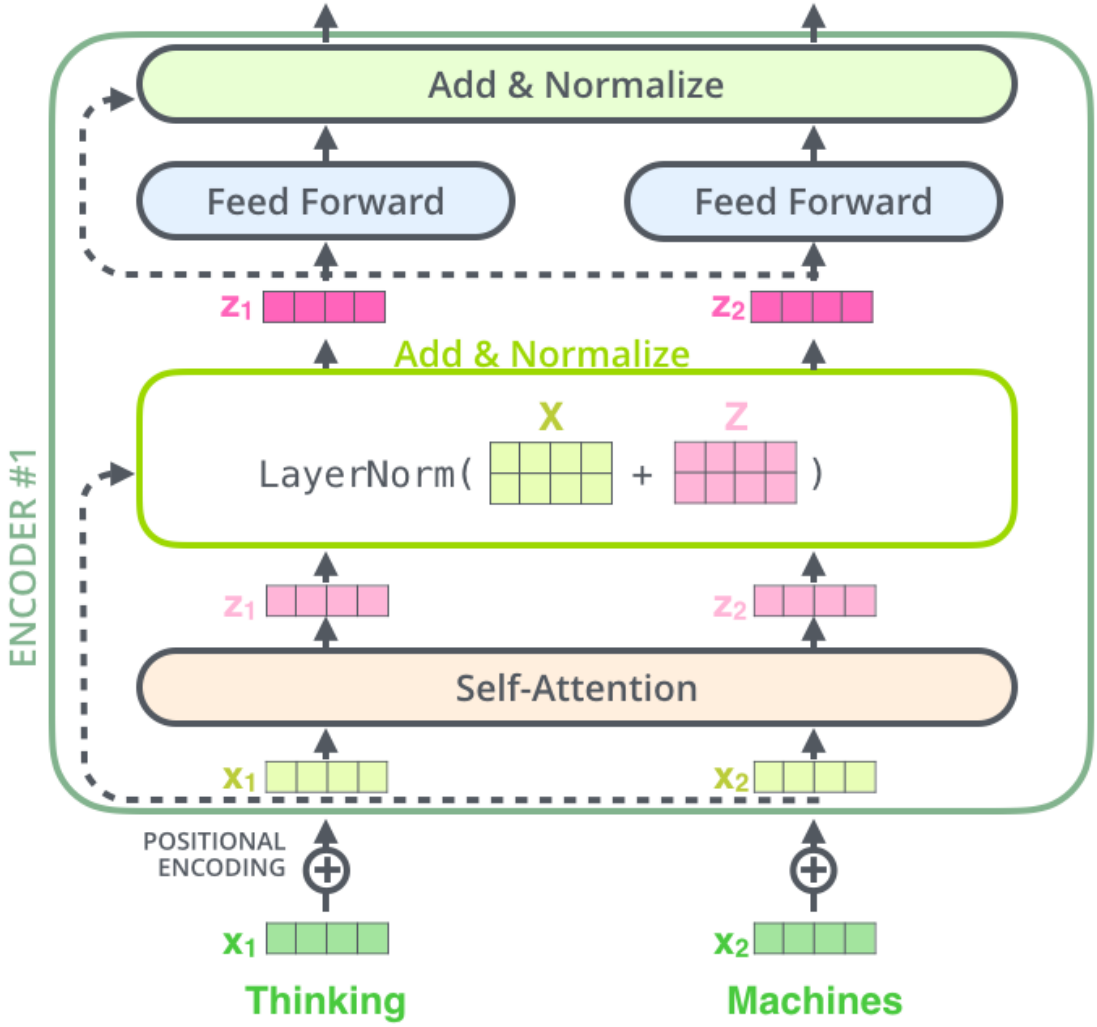
One detail in the architecture of the encoder that we need to mention before moving on, is that each sub-layer (self-attention, ffnn) in each encoder has a residual connection around it, and is followed by a layer-normalization step.

Residual Conneciton → Help regain context

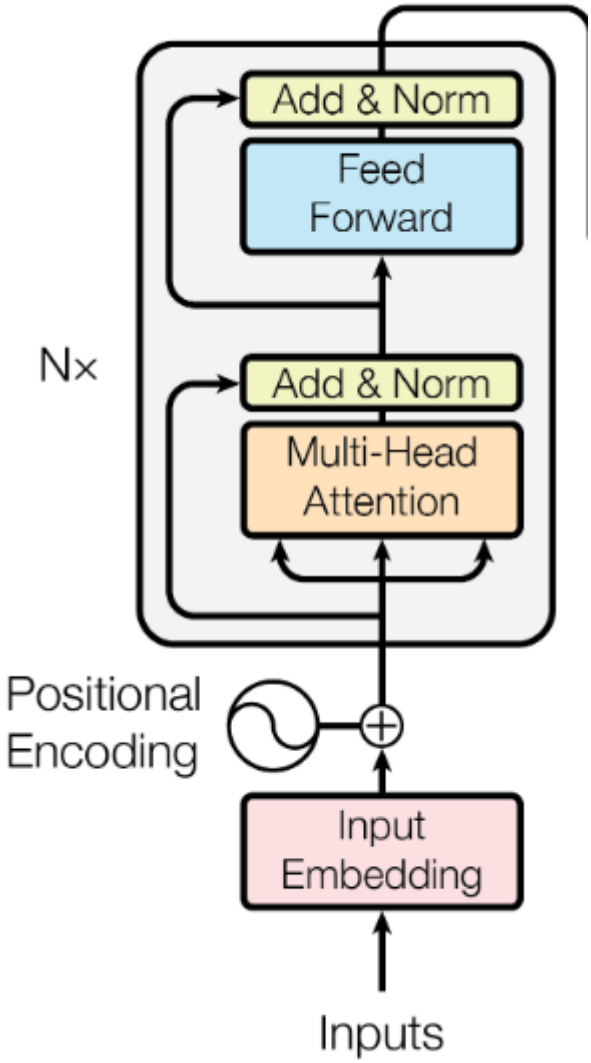
A skip connection is added to allow the output of the block to bypass the feedforward layer and be added to the input of the next block, which can help to mitigate the vanishing gradient problem and improve the training of the model.

Layer-Normalization → Form of Regularization

Layer normalization is a normalization technique used to reduce the internal covariate shift in neural networks, which can help to improve the stability and performance of the model.



Encoder Archiecture



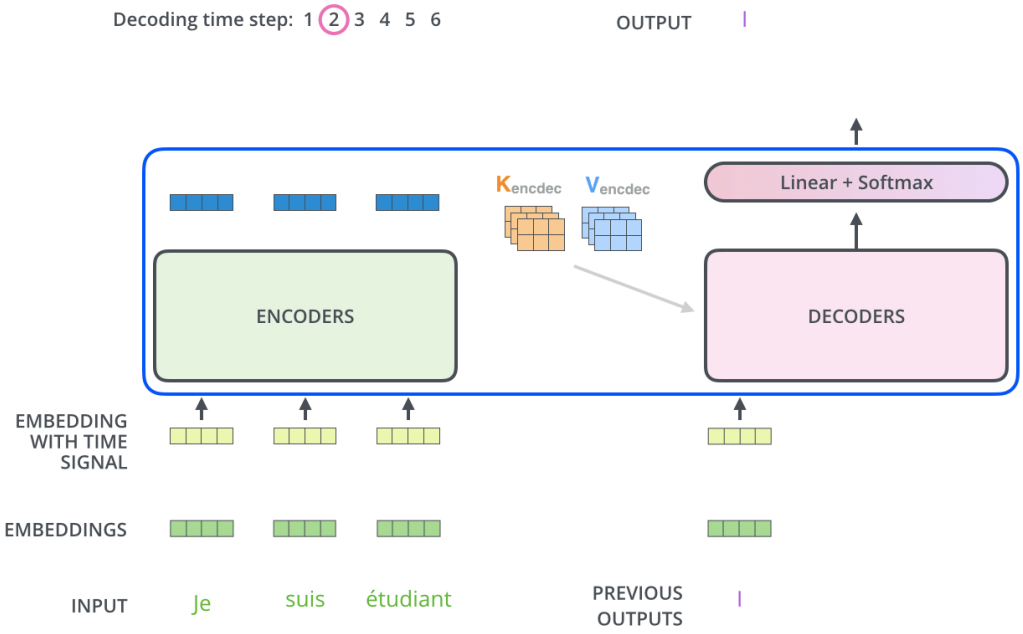
Decoder

The encoder start by processing the input sequence. The output of the top encoder is then transformed into a set of attention vectors K and V.

These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence:

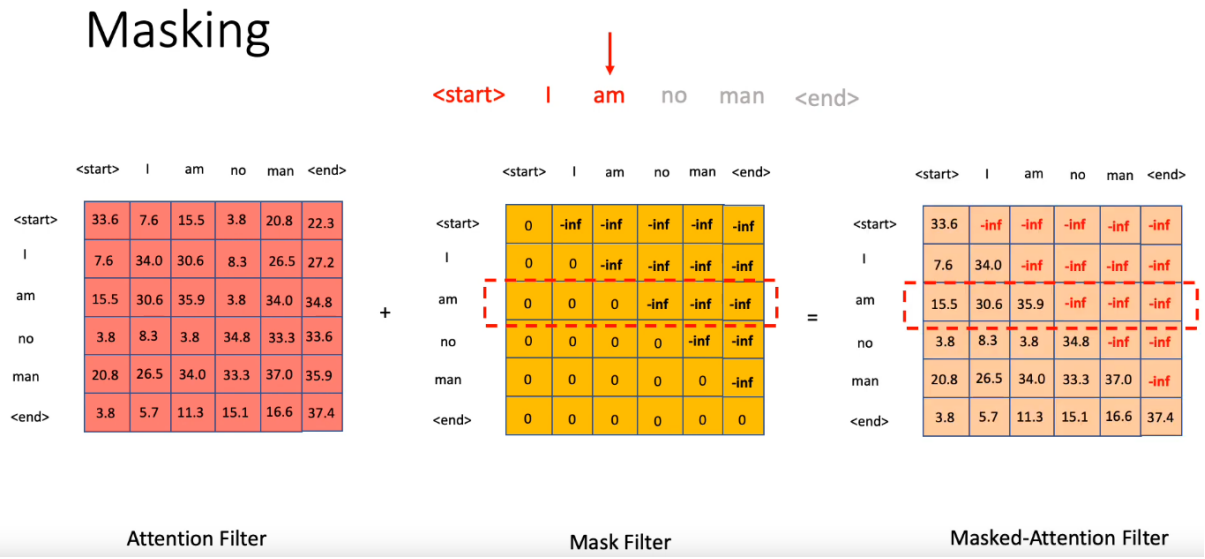
The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output. The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did. And just like we did with the encoder inputs, we

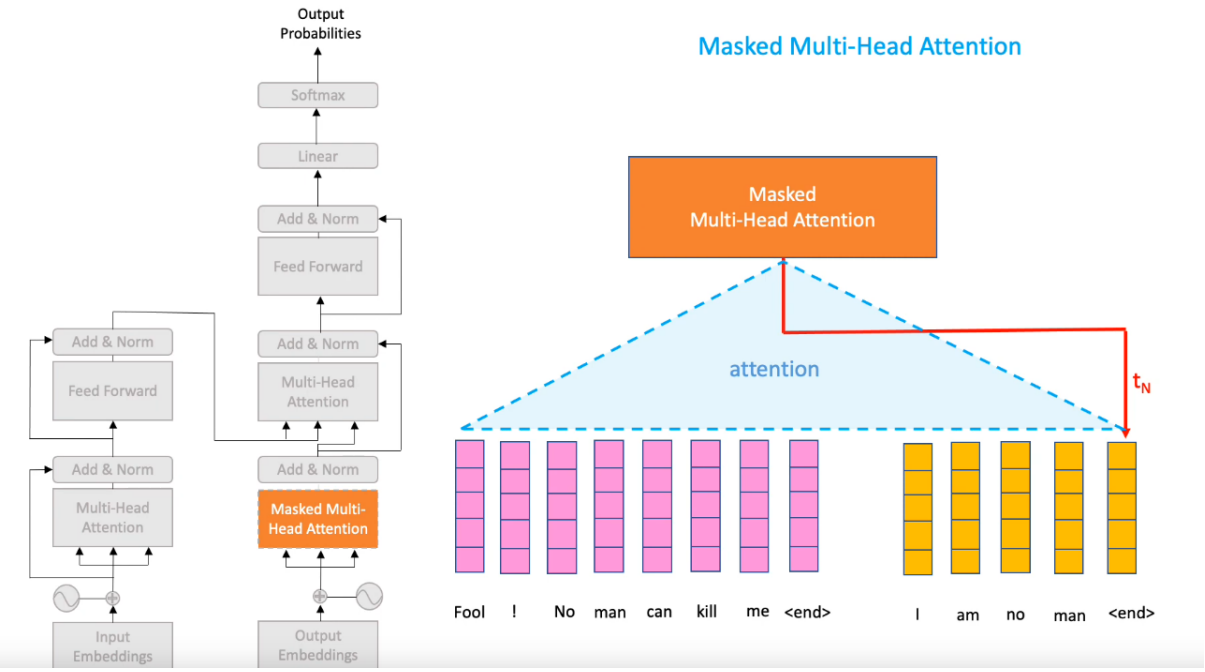
embed and add positional encoding to those decoder inputs to indicate the position of each word.



Masked-self Attention

The input to the decoder block is the output of the previous decoder block or the encoded input sequence, depending on the specific architecture used. The masked self-attention mechanism allows the decoder to only attend to previous tokens in the output sequence, while masking out future tokens that have not yet been generated





Cross-multihead Attention

The output of the masked self-attention mechanism is then passed through a cross multihead attention layer, which allows the decoder to attend to the encoded input sequence. This layer enables the model to generate contextually appropriate outputs based on the input sequence.

Architecture

The decoder block also includes positional encoding, layer normalization, and skip connections, similar to the encoder block.