# Lectures Notes from Beg Rohu 2025

Clark Miyamoto (cm6627@nyu.edu)

June 11, 2025

## 0.1 Introduction

These are a set of lecture note from the Beg Rohu Summer School in 2025.

## 0.2 Request for Contributions

I know diffusion, so my notes are coherent for Stephan and Marc's lectures. So for those with better notes for Elena, Olivier, Jean, Yann, and Arvind, I would greatly appreciate contributions.

## 0.3 List of Participants

# Contents

0.1 Introduction . . . 2
0.2 Request for Contributions . . . 2
0.3 List of Participants . . . 2

**1 Elena Agliari: Information Processing in Hebbian Networks 5**
1.1 Overview . . . 5
1.2 A Model for Neurons . . . 5
  1.2.1 Noiseless Dynamics . . . 8
1.3 Jun 9th . . . 8
  1.3.1 Hopfield Model . . . 9
1.4 SK Model . . . 9

**2 Olivier Dauchot: Swarm Robotics as Smart Active Matter 11**

**3 Julia Kempe: Synthetic Data 13**
  3.0.1 Infinite Memory Model . . . 13
  3.0.2 Kernel and Regression . . . 14

**4 Jean Rémi King: Geometry of Thought 17**
4.1 Overview . . . 17

**5 Yann LeCun: Self Supervised Models 19**
5.1 Overview: A Path to Advance Machine Intelligence . . . 19
  5.1.1 What is an Energy Based Model (EBM) . . . 19
  5.1.2 What is a World Model . . . 21
  5.1.3 How do I learn more . . . 21
5.2 Jun 6th . . . 21
  5.2.1 Lagrangian Mechanics . . . 22
  5.2.2 . . . 22

**6 Stéphane Mallat: Score Based Diffusion & Renormalization Group Flow 25**
6.1 Overview . . . 25
  6.1.1 Transport . . . 26
  6.1.2 Outline . . . 26
6.2 Historical Models: Energy Based Models, GANs, Normalization Flows . . . 26
  6.2.1 Energy Based Models (EBM) . . . 26
  6.2.2 GANs . . . 27
  6.2.3 Normalizing Flows . . . 28
6.3 Score Based Diffusion . . . 29
  6.3.1 Generalization vs Memorization? . . . 31
  6.3.2 Architecture of CNNs . . . 31
6.4 Wavlets . . . 32

6.4.1 Optimal Denoising . . . 32
6.4.2 Compression . . . 34
6.5 Wavelet Slides . . . 34
6.6 Renormalization Group & Hierarchies . . . 35
6.7 Continuous Ising, $\phi^4$ Model . . . 36
6.8 Can we use this in other things? . . . 36

**7 Marc Mézard: Statistical Physics of Generative Diffusion 39**
7.1 Overview . . . 39
7.2 Recall: Stochastic Processes . . . 39
7.2.1 Langevin Equation . . . 39
7.2.2 Fokker Planck . . . 40
7.2.3 Ornstein Uhlenbeck . . . 40
7.2.4 General Time and Variance . . . 41
7.3 Principals of Generative Diffusion . . . 41
7.3.1 Forward Process . . . 41
7.3.2 Backwards Process . . . 42
7.3.3 Comment on Discretization . . . 42
7.3.4 The Score . . . 42
7.3.5 Approximating the Score . . . 43
7.3.6 Various Scores . . . 43
7.4 A First Example: Gaussian Data . . . 44
7.5 Related Approaches i.e. ODE, Stochastic Localization, & Interpolants . . . 46
7.5.1 Ordinary Differential Equations . . . 46
7.5.2 Example with Isotropic Gaussian . . . 46
7.5.3 Stochastic Interpolants . . . 47
7.5.4 Unifying Framework for Langevin, ODE, and Stochastic Interpolants . . . 47
7.5.5 Stochastic Localization . . . 48

**8 Arvind Murugan: Learning without Neurons 51**
8.1 Overview . . . 51
8.1.1 Training Molecules . . . 53
8.2 Expressitivity and Trainability . . . 54
8.2.1 Training of Molecular Networks . . . 54
8.2.2 Potts Model of Molecular Networks . . . 54
8.2.3 Place Models . . . 55
8.3 June 11th . . . 55

**9 Etc. 57**
9.1 People Presentations . . . 57
9.2 Q&A . . . 57
9.3 Acknowledgements . . . 57

# Chapter 1

# Elena Agliari: Information Processing in Hebbian Networks

## 1.1 Overview

In these lectures, we are interested in creating phase diagrams, where the macrostate is wether the neural network learns the data distribution, as we vary relevant quantities of the network (i.e. size dependence, architecture, etc.)

We'll go over

1. Intro to Hebbian Networks
2. Hopfield Model
3. Restricted Boltzmann Machines and Hebbian Networks
4. Architectures

## 1.2 A Model for Neurons

Our story begins with a MC Oulloch (in 1943) trying to create a mathematical model for neurons. Neurons can be effectively described as nodes on a network

- Edges of the graph physical correspond to "axon" (which are paths which connect neurons).
- They fire electrical signals which in-turn will choose wether or not another neuron will fire.

Now let's mathematize this. A system of $\{x_i : x_i \in \{0,1\}_{i=1}^N$ neurons. They are coupled with strengths $\{J_i : J_i \in \mathbb{R}\}_{i=1}^N$ to an output neuron $y \in \{0,1\}$. The state equation of the system is

$$y = \Theta(U - U^*), \quad \text{s.t. } U = \sum_i J_i x_i \tag{1.2.1}$$

where $\Theta$ is the heaviside function, and $U^* \in \mathbb{R}$ is interpreted as the activation energy for the neuron to fire.

However, neurons don't just point to one "god" neuron $y$, they all point towards each other (albeit sparsely). So our true state equation should be

$$x_i = \Theta(U_i - U_i^*), \quad \text{s.t. } U_i = \sum_{j=1}^{N} J_{ij} x_j \tag{1.2.2}$$

Ok, this is all cool, but what if we want to generlaize our model a little more? (Specifically we'd like to recover a spin-glass system that way we can take advantage of our knowledge from statistical physics). So a couple things

- Consider defining a random variable $z_i \sim \sigma_z$ where $\mathbb{E}[z] = 0$ and $\mathbb{E}[z] = 1$.
- Instead of using binary variables $\underline{x} = (x_1, ..., x_N) \in \{0,1\}^N$, what if we could use ising-spin variables $\underline{\sigma} \in (\sigma_1, ..., \sigma_N) \in \{\pm 1\}^N$. We can transform between the two using

$$x_i = \frac{1}{2}(1 + \sigma_i) \tag{1.2.3}$$

Using this transformaiton, we can rewrite our state equation in-terms of $\underline{\sigma}$

$$\sigma_i^{(t+1)} = \text{sgn}[\varphi_k^{(t)} + T z_i^{(t)}], \quad \text{s.t. } \varphi_i = \sum_j J_{ij}\sigma_i^{(t)} + h \tag{1.2.4}$$

**Definition 1** *The random variable $X \sim Rad(p)$ is defined to have the following properties*

$$\mathbb{P}(x = 1) = 1 - \mathbb{P}(x = -1) = \frac{1+p}{2} \tag{1.2.5}$$

**Hypothesis 1** *We assume the probability distribution over $p_Z$ is symmetric. That is $p_Z(-z) = p_Z(z)$*

If we assume this hypothesis to hold, and define the following

$$g_Z(z) \equiv \int_{-\infty}^{z} p_Z(u) du = \mathbb{P}[Z < z] \tag{1.2.6}$$

We can show that probability of $\mathbb{P}[\sigma_i^{(t+1)} = \pm 1] = g_Z(\pm \frac{\varphi_i^{(t)}}{T})$.

Also some notation for the future, we'll notate $p_Z(z) \equiv \frac{1}{2}[1 - \tanh^2(z)]$ and therefore $g_Z(z) = \frac{1}{2}[1 + \tanh(z)]$.

**Definition 2 (Update / Dynamics)** *We define two ways to perform dynamics*

- *Sequential. We select $i$ randomly, and update $\mathbb{P}[\sigma_i^{(t+1)}] = g(\sigma_i \varphi_i^{(t)}/T)$*
- *Parallel. $\mathbb{P}[\sigma^{(t+1)}] = \prod_i g(\sigma) i \varphi_i^{(t)}/T)$*

, Finally, we we assume the coupling matrix is symmetry ($J^T = J$), $J_{ii} \geq 0$ and $h$ is stationary. Then the Lyaponov function

$$L = -\frac{1}{2}\underline{\sigma}^T J \underline{\sigma} - \underline{h}^T \underline{\sigma} \tag{1.2.7}$$

What's nice about this function is that is always is lowered with the dynamics we've asserted

$$\Delta L(\sigma, J) \leq 0 \tag{1.2.8}$$

where $\Delta L$ is meant to be interpreted as a discrete difference in time $L(t = t' + \Delta t') - L(t = t')$. You can also show that $L$ is lower bounded.

The result of all of this, if you run your update step, you can create a diagram which shows where you end up at fix points dependent on where you start. So if data is encoded in $\bar{\sigma}$, and data flows to fixed points (attractors) $\underline{\xi}$ depending on initialization— we have created a classifier!

**Definition 3 (Attractor)** *We define the attractor* $\xi \in \{\pm 1\}^N$*, is the configuration of spins* $\underline{\sigma}$ *where you flow to a fixed point. So* $\underline{\xi} \in \{\{\{\pm 1\}^N\}^M$ *is defines the set of attractors emitted by the model.*

However, to make a *good* classifier, we'd like to be able to tune/train our couplings $J$ s.t.

- We can host a many attractors / stationary points
- The radius between the initalization and the attract is big enough s.t. we can find attractors. Aka generalize.

**Definition 4 (Retrivial)** *A definition for when the network emits attractors*

- *(Strong Definition).* $J = J(\xi)$ *retries* $\underline{\xi}$ *staritng from* $\underline{\sigma}$*.*
- *(Weak Definition).* $\underline{\xi}$ *is* $(\delta, \epsilon)$ *stable. That is when your state enter a* $\delta$*-radius ball away from the attractor, it will forever stay within an* $\epsilon$*-radius ball to the attractor.*

Ok, to make our life easier, assume we know what we want our attractors to be $\{\xi^\mu\}_{\mu=1}^K$, and all we need to do is tune our interaction matrix $J$. We can use **Hebb's Rule** to learn the parameters.

**Definition 5 (Hebb's Rule)** *Assert* $J_{ij} = \frac{1}{N}\sum_{\mu=1}^K \xi_i^\mu \xi_j^\mu$*. The* $1/N$ *is to make the rule non-extensive in the number of spins* $N$*. We perform the update*

$$J_{ij}^{(t+1)} \leftarrow J_{ij}^{(t)} + \frac{1}{N}\xi_i \xi_j \tag{1.2.9}$$

*We note that the time scale of the couplings* $J$ *is drastically different from the time scale of the neurons* $\underline{x}$*.*

> In the case the number of attractors is $K = 1$.
>
> $$L(\underline{\sigma}) = \frac{1}{2N}\sum_{ij} \sigma_i \sigma_j \xi_i \xi_j \tag{1.2.10}$$
>
> $$= -\frac{1}{2N}\sum_{ij} \tilde{\sigma}_i \tilde{\sigma}_j \geq -\frac{N}{2} \tag{1.2.11}$$
>
> where the transformation we did was $\sigma_i \to \tilde{\sigma}_i \xi_i$. So we've recovered a Potts Model!

**Definition 6 (Motts Magnetization)** *We define an overlap parameter*

$$m_\mu \equiv \frac{1}{N}\sum_i \tilde{\sigma}_i = \frac{1}{N}\sum_i \sigma_i \xi_i^\mu \in [-1,+1] \tag{1.2.12}$$

*What is nice about this parameter is that it allows us to access the quality of retrivial. That is if your states $\underline{\sigma}$ actually go to the attractor $\underline{\xi}$, then $m_1 \to +1$.*

**Theorem 1 (Kohonen's Projector Rule)** *If $J$ is s.t. $\sum_j J_{ij}\xi_j^\mu = \lambda \xi_i^\mu$ for $\lambda > 0$ for all $i$. Then*

$$sgn(J\underline{\xi}^\mu) = sgn(\lambda\underline{\xi}^\mu) = \underline{\xi}^\mu \tag{1.2.13}$$

*This means that $\underline{\xi}$ is a fixed point of the dynamics.*

### 1.2.1 Noiseless Dynamics

Consider the following system

$$\xi^1 \varphi_i(\xi^1) > 0 \tag{1.2.14}$$

If we expand this quantity this

$$\xi_i^1 \varphi_i(\xi^1) = \frac{\xi_i^1}{N}\sum_{j=1,j\neq i}^{N}\sum_{\mu}^{K} \xi_i^\mu \xi_j^\mu \xi_j^1 = \frac{N-1}{N} + \frac{1}{N}\sum_{\mu>1}\sum_{j=1,j\neq i} \xi_i^1 \xi_i^\mu \xi_j^\mu \xi_j^1 \tag{1.2.15}$$

We can see this leads to a signal-to-noise decomposition...

## 1.3 Jun 9th

Let $\underline{\sigma} = \{\pm 1\}^N$ bet a set of neurons, $\underline{\xi}^\mu \in \{\pm 1\}^N$ be the set of memories $\mu = 1, ..., K$, and $J = \frac{1}{N}\xi \cdot \xi^T$ and $J_{ij} = \frac{1}{N}\sum_\mu \xi_i^\mu \xi_j^\mu$, with the update step

$$\underline{\sigma}^{(t+1)} = \text{sgn}(\underline{\varphi}^{(t)}) \tag{1.3.1}$$

$$\varphi_i^{(t)} \equiv \varphi_i(\underline{\sigma}^{(t)}) = \sum_j J_{ij}\sigma_j^{(t)} \tag{1.3.2}$$

The question we'll be answering today is $\underline{\sigma} = \underline{\xi}^\mu$ a stable solution? For simplicity let's assume $K = 1$, so $\mu = 1$ is our target configuraton

$$\xi_i^1 = \text{sgn}(\sum_k J_{ik}\xi_k^1) \tag{1.3.3}$$

$$\xi_i^1 \sum_{j\neq i} J_{ij}\xi_j^1 = \underbrace{\frac{N-1}{N}}_{\mu=1,\text{ Term S}} + \underbrace{\frac{1}{N}\sum_{j\neq i}\sum_{\mu>1,\text{ Term R}} \xi_i^1 \xi_j^1 \xi_i^\mu \xi_j^\mu}_{\mu>1} \tag{1.3.4}$$

Note the second term is $\sqrt{(K-1)(N-1)}/N \approx \sqrt{K/N}$. So as long as $K/N \to_{N\to\infty} 0$, then the first term will be greater than the absolute value fo the second term.

> Consider an initial configuration of the neurons $\underline{\sigma}^{(0)} = \underline{\xi}^{\mu} \odot \underline{\chi}^{\mu}$ where $\chi_i^{\mu} \sim \mathrm{Rad}(p)$ (meaning $\mathbb{P}[\chi_i^{\mu} = \pm 1] = (1+p)/2$) and $\odot$ is the Hammard product (element wise multiplication). This is supposed to be interpreted as a $\chi$ perturbation away from the memory $\underline{\xi}^{\mu}$.

> Consider $\underline{\sigma}^{(0)} = \mathrm{sgn}(\underline{\xi}^1 + \underline{\xi}^2 + \underline{\xi}^3)$. You can show it is also stable under $K/N \to 0$.
>
> In General $\underline{\sigma}^{(0)} = \mathrm{sgn}(\sum_{\mu=1}^{L} \underline{\xi}^{\mu})$ where $L$ is odd is stable under $K/N \to 0$.

So we've seen that $K/N$ is crucial to determining stability in the system. So let's give it a name

**Definition 7 (Load)** *We define the **load** quantity*

$$\alpha \equiv \lim_{n \to \infty} \frac{K}{N} \tag{1.3.5}$$

*When*

- $\alpha = 0$, *is **low load**.*
- $\alpha > 0$ *is **high-load***

### 1.3.1 Hopfield Model

Consider the Hamiltonian

$$H_N(\underline{\sigma}; J) = -\frac{1}{N} \sum_{i<j} J_{ij} \sigma_i \sigma_j - \sum_i h_i \sigma_i \tag{1.3.6}$$

$$\mathcal{H}(\underline{\sigma}; J) = -\frac{1}{N} \sum_{i<j} \sum_{\mu} \xi_i^{\mu} \xi_j^{\mu} \sigma_i \sigma_j - \sum_{\mu} \sum_i \xi_i^{\mu} h_i \sigma_i \tag{1.3.7}$$

where $\xi_i^{\mu} \sim \mathrm{Rad}(0)$. We also introduce some notation $\omega(\dot{}) = \sum_{\underline{\sigma}} (\cdot) P_N(\underline{\sigma}, \underline{\xi})$, $\mathbb{E}[\cdot] = \sum_{\underline{\xi}} (\cdot)$, and $\langle \cdot \rangle_N = \mathbb{E}\omega_N(\cdot)$. We also denote hte Mattis Magnetization as $m_{\mu} \equiv \frac{1}{N} \sum_i \xi_i^{\mu} \sigma_i \in [-1, +1]$.

So now going back to $\mathcal{H}$, we can rewrite it int this notation as

$$\mathcal{H}_N = -\frac{1}{2N} \sum_{\mu} N m_{\mu} N m_{\mu} + \frac{1}{2} \sum_{\tau, \mu} (1) - \sum_{\mu} m_{\mu} N h_{\mu} \tag{1.3.8}$$

$$= -\frac{N}{2} \underline{m}^2 + \frac{K}{2} - N \underline{h}^T \underline{m} \tag{1.3.9}$$

Now our goal is to compute $\langle m \rangle$. We can accomplish this by computing the free energy $f_N(\xi) = \frac{1}{N\beta} \log Z_n(\xi)$ and use it as the moment generating function.

## 1.4 SK Model

# Chapter 2

# Olivier Dauchot: Swarm Robotics as Smart Active Matter

# Chapter 3

# Julia Kempe: Synthetic Data

Kempe's talk is titled "Synthetic Data: The Good, The Bad, The Ugly (and The Math)". I like the data.

The premise is we are at the point where a fair amount of data on the internet has been created via AI. And we all know about model collapse (that is if you train an LLM on data generated by an ML program will eventually lead to models performing horribly). This is not so surprising because if you sample a gaussian, and fit it, then resample, on and on— you'll find the Gaussian's variance will go down— regression to the mean. So how can we model & understand this phenomena!

So some causes

1. Finite sample error
2. Expressitivity error (model under capacity)
3. Approximation error (due to optimization bias, etc.)
4. Inteference error.

Kempe likes to study these from the perspective of scaling laws.

### 3.0.1 Infinite Memory Model

This was published by Hutler in 2022.

Consider a dataset $\mathcal{D}_T = \{(i_t, f(i_t))\}_{t=1}^{T} \subset \mathcal{D}$. And we have a memory model

$$\mathcal{A}(i, \mathcal{D}_T) = \begin{cases} f(i) & \text{if } i = \imath e \text{ s.t. } \exists 1 \leq e \leq T \\ \perp & \text{otherwise} \end{cases} \tag{3.0.1}$$

what we notice is that $p(i) \sim i^{-\beta}$. There's a test error

$$\begin{aligned} E_{test}(T) &= \mathbb{P}(i \neq ie : i \leq \ell \leq T)) & (3.0.2) \\ &= \sum_{i=1}^{\infty} p(i)(1 - p(i))^T & (3.0.3) \\ &\simeq \sum_{i=1}^{\infty} i^{-\beta}(1 - i^{-\beta})^T \simeq \sum i^{-\beta e^{-(i^{-\beta})T}} & (3.0.4) \end{aligned}$$

Let $c = 1 - 1/\beta$. Then

$$= T^c \sum_i i^{-\beta} e^{-(i^{-\beta})T} \asymp \Gamma(c, Tk^{-\beta}) - \Gamma(c, T) = \mathcal{O}(1) \tag{3.0.5}$$

Ok... Now let's see the behavior as we retrain based off of the previous model. That is our dataset will now be $\mathcal{D}_T = \{(i, \mathcal{A}_{T_0}(i))\}_i$, so $\mathcal{A}_{T_0}(i)$ is like ChatGPT trained on very clean internet data, and this current dataset is the internet 1 year after ChatGPT.

Notice that $i$ appears $T_0 i^{-\beta}$ times in $\mathcal{D}_{T_0}$. So in the limit $T_0 i^{-\beta} \ll 1$. Then if $i > k$, $\mathcal{A}_{T_0}$ hasn't seen it.

$$\mathcal{A}'_{T_0,T}(i) = \begin{cases} f(i) & \text{if } i \le k \text{ and } i = ie\ 1 \le \ell \le T \\ \perp & \text{otherwise} \end{cases} \tag{3.0.6}$$

The test error now chnage

$$E_{test} = \mathbb{P}(i > k : 1 \le \ell \le T) = \sum_{i=1}^{k} p(i)(1 - p(i))^T + \sum_{i>k} p(i) \asymp k^{-(p-1)} = k^{-c\beta} \tag{3.0.7}$$

If we redo this calculation in the limit $1 \ll T < k^{\beta}$. Then $\Gamma(c, Tk^{-\beta}) \sim \mathcal{O}(1)$, so

$$E_{test} \asymp k^{-\beta c} + T^{-c} \asymp T^{C} \tag{3.0.8}$$

Another case $T \le k^2$, then

$$E_{test} \asymp k^{-\beta c} + T^{-C} \asymp k^{-\beta c} \tag{3.0.9}$$

Basically, you redo this calculation over and over, and we see this scaling law apppear up everywhere. SO the test error is limited by these bounds.

### 3.0.2 Kernel and Regression

Consider your inputs $x \sim \mathcal{N}(0, \Sigma) \in \mathbb{R}^D$ and they're noisy $\epsilon \sim \mathcal{N}(0, \sigma^2) \in \mathbb{R}$ and the label $y = x^T W_0 + \epsilon$. Lets' start by assumping $d < T$, we have less data than parameters.

$$E_{test} = \mathbb{E}_{x,\epsilon} ||x^T \hat{w} - y||^2 - \sigma^2 \tag{3.0.10}$$

The best fit $\hat{w}$ is via oridnary least squares between $X$ and $Y$. Meaning

$$\begin{aligned} \hat{w} &= (X_0^T X_0)^{-1} X_0^T Y_0 && \text{(3.0.11)} \\ &= (X_0^T X_0)^{-1} X_0^T (X_0 w_0 + E_0) && Y_0 = X_0 w_0 + E_0 \text{ (Original fit)} \quad \text{(3.0.12)} \\ &= w_0 + X_0^T E_0 && \text{(3.0.13)} \end{aligned}$$

With this model, the test error we get is

$$\begin{aligned} E_{tot} &= ||(\hat{w} - w_0)||^2 && \text{(3.0.14)} \\ &= \mathbb{E}_\epsilon ||X_0^T E_0||^2 && \text{(3.0.15)} \\ &= \sigma^2 \mathbb{E}\ \mathrm{Tr}(X_0 (X_0^T X_0)^{-2} X_0^T) && \text{(3.0.16)} \\ &= \sigma^2 \mathbb{E}\ \mathrm{Tr}(X_0^T X_0)^{-1} && \text{(3.0.17)} \end{aligned}$$

Recall that $X_0 \sim \mathcal{N}(0, \Sigma)$, we can use some random matrix theory to compute this

$$= \sigma^2 \frac{d}{T - d - 1} \sim \frac{\sigma^2 d}{T} \tag{3.0.18}$$

### Model Collapse

So we did this just on one weight. So what happens if we do this recurisvely, over and over. Let's look at how the test error at the $i$'th iteration changes depending on the ground truth $w_0$

$$E^{(i)}_{test} = ||\hat{w}_i - w_0||_2 \tag{3.0.19}$$

$$\tag{3.0.20}$$

Let's start at $i = 2$

$$E^{(2)}_{test} = ||\hat{w}_2 - w_0||_2 = ||\underbrace{\hat{w}_2 - \hat{w}_1}_{X_1^T E_1} + \underbrace{\hat{w}_1 - w_0}_{X_0^T E_0}||^2 \tag{3.0.21}$$

$$\asymp \frac{d\epsilon^2}{T} + \frac{d\epsilon^2}{T_0} \tag{3.0.22}$$

Ok if we do this $n$ times

$$E^{(n)}_{test} = \frac{d\epsilon^2}{T} + \frac{nd\epsilon^2}{T_0} \tag{3.0.23}$$

# Chapter 4

# Jean Rémi King: Geometry of Thought

## 4.1 Overview

Jean Rémi King is a research at Meta Brain. In his talk he'll be going over:

1. The discovery of neural codes.
2. Evidence of convergence.
3. Coding compositional structures.

As an overview, the brain is obviously why humans are able to reason about the world. So the question becomes what are the *exact* mechanisms which allow for reasoning? In King's research they often use an MRI and to image the brain while a participants is being shown stimuli. Experimentaly, you can show that (in a spatial sense) the brain reacting to stimuli respects locality & hierarchy: that words go here and faces go there, and orientations of faces are within the faces section, and slightly to the side is if the face is smiling or not.

How does machine learning come into play? Well obviously use ML to help us decode what the brain is doing (so you can perform a bayesian inference on, given brain activity, ask what is this person looking at?). If you want to see these visualizations, see Ozcelik & van Rullen (2022), and then say wow.

**Definition 8 (Representation (according to King))** *A representation is linearly readable information. That is if there is linear map $f$ which $\hat{Y} = f(X)$ approximates $Y$ quite well, then $X$ is a representation of $Y$.*

A very interesting application of keeping representations restricted to linear maps is attmempting to perform linear regression from brain activity to various encoders (i.e. CNN, word2vec, and ChatGPT). You can do an experiment where you map how these perform as a function of time after exposed to a word and the location in the brain, and you recover what you expect...

# Chapter 5

# Yann LeCun: Self Supervised Models

## 5.1 Overview: A Path to Advance Machine Intelligence

We begin with a question: why do we want to build systems that are as smart as humans? The answer is trivial lol. Obviously if they're too smart, we will live in dystopia. So the question becomes how do we construct machines with human-level intelligence? By this, he means algorithms which can generalize their knowledge to solve new problems in zeros shot. Yann believes this can be answered by **self-supervised learning.**

He motivates his answer by noting a few things that occur in nature

- Humans & animals seem to do something quite close to self-supervised. And they are quite good at zero-shot tasks.
  - I.e. look at the amount of visual data that a baby sees in it's first 4 years of life, compared to the amount of data to train a SOTA LLM. They're comparable. But the baby can do zero shot tasks, LLMS are more good at doing lookup.

### 5.1.1 What is an Energy Based Model (EBM)

Traditionally, we preform inference via a **feedforward model**. That is you run

$$\hat{y} = (f_1 \circ f_2 \circ ... \circ f_n)(x) \tag{5.1.1}$$

However, you are limited by the architecture, for any inference (simple or complex) you are set to do $n$ computations. I.e. why should asking an LLM "does 2+2 =4?" use the same amount of computation as "is P=NP?".

This leads us to another method, which is performing **inference via optimization**.

$$\hat{y} = \text{argmin}_{y \in \mathcal{Y}} f(x, y) \tag{5.1.2}$$

Interestingly, you can also have multiple answers as well. For training such models, you make it learn the $f_\theta$ (the **energy function**). You train this function s.t. $f$ takes low values on the training data, and it takes higher values away from the training data. Obviously we have assumed some sort of locality, that is $|\nabla f| <$ Constant– every model makes some assumptions, it more up to you. However, because inferences require an optimization, you kinda want some notion of locality, it'll makes the optimization much easier.

> What is the difference between energy-based models and probabilistic models? Well if you assume a Gibbs measure, and set the Hamiltonian to the energy function, you get
>
> $$p(y|x) = \frac{e^{-\beta f(x,y)}}{Z} \tag{5.1.3}$$

You can also add latent parameters. Consider the energy function $E(x, y, z)$ where $z$ relates to hyper parameters of the model, etc. We can do $\min_{z\in\mathcal{Z}} E(x, y, z)$ are then use the resulting quantity as our new energy function.

### Training EBMs

He shows how to train EBMs by walking through an Ising model example.

A naive method is to just perform gradient descent

$$E(y) = -\sum_{ij} w_{ij} y_i y_j \tag{5.1.4}$$

$$\frac{\partial E}{\partial w_{ij}} = -y_i y_j \tag{5.1.5}$$

$$\therefore w_{ij} \leftarrow w_{ij} + \epsilon(y_i y_j) \qquad \text{Backprop Step} \tag{5.1.6}$$

However this is dumb, you only low energy at the specific $y$'s. You don't instill any locality. Instead do **contrastive methods**

$$w_{ij} \leftarrow w_{ij} + \epsilon(y_i y_j - \hat{y}_i \hat{y}_j) \tag{5.1.7}$$

In this case we sample $\hat{y}_i \in \mathcal{Y}$, and then move the energy function $f$ near them as well.. This allows the energy function to smoothly change. Obviously this particular method is crude, but you can implement smarter ways to do this... In summary, a contrastive method is one which not only lowers the energy function at the data point, but also lowers near by points as well...

The other methods are **architectural**, that is you choose an architecture which is able to automatically do contrastive learning (i.e. bottleneck auto-encoders, transformers, etc.). And finally **regularized optimizations**.

This is explained throughly in https://arxiv.org/pdf/2101.03288

### Loss for Contrastive Learning

Going back to contrastive learning, let's see how to construct a loss. Given the EBM $f(x, y)$. We want to lower the energy at target point $(x, y)$, and increase the energy at $(\hat{x}, \hat{y})$.

You can consider **triplet loss** $\mathcal{L}$

$$\mathcal{L}((x, y), (x, \hat{y})) = f(x, y) + [m(y, \hat{y}) - f(x, \hat{y})] \tag{5.1.8}$$

It keeps similar labels together, and pushes dis-similar images away, as endowed by $m$.

If you want to model probability distributions, can also choose the loss to be the negative loglikelihood. (This naming makes sense if you choose the Gibbs distribution $p(y|x) = \exp(-\beta f(x, y))/Z$).

$$\mathcal{L}((x, y), (x, \hat{y})) = f_\theta(x, y) + \frac{1}{\beta} \log \int_{\mathcal{Y}} e^{-\beta f_\theta(x,y)} dy \tag{5.1.9}$$

$$\nabla_\theta \mathcal{L} = \nabla_\theta f - \mathbb{E}[\nabla_\theta f_\theta] \simeq \nabla_\theta f(x, y) - \nabla_\theta f(x, \hat{y}) \tag{5.1.10}$$

**Diffusion Models as Contrastive Methods**

### 5.1.2 What is a World Model

In this quest for better intelligence, we have motivated that living creates a world model inside their head. So the next-generation of machine intelligence should also copy the same.

Let's walk through a simple example. We can see the benefit of this approach when trying to create an agent which interacts with the world. Consider the following situation, you have an encoder $E$ : Observations of World $\to \mathcal{S}_t$ a computational model of the world which takes in observations and actions $W : \mathcal{S} \times \text{Actions} \to \mathcal{S}$ and attempts to predict what the next state of the world is. So an example of inference is

$$x_{t+1} = E^{-1}(W(x_t, q)) \tag{5.1.11}$$

where $x_t, x_{t+1} \in \mathcal{S}$ and $q \in$ Actions. You can see this is like a optimal control problem, but we don't solve Hamilton Jacobi Equation, but instead learn the optimal result based on data.

### 5.1.3 How do I learn more

Read the following papers

1. Opinion piece by Yann. https://openreview.net/forum?id=BZ5a1r-kVsf
2. SimCLR
3. (Hierarchical) JEPA
4. How to train your Energy Based Model
5. VicReg
6. MCR$^2$
7. MMCR

## 5.2 Jun 6th

Today we start with a simple example in the energy-based model framework. Here we set the energy function as the cost between label and predicted label.

$$E(y, z) = C(y, \text{Dec}(z)) \tag{5.2.1}$$

where $C(a, b) = ||a - b||^2$ is the cost, and $\text{Dec}(z) = Wz$ is a linear decoder. You can find the "free energy" of $y$ by minimizing over $z$

$$F(y) = \min_z E(y, z) \tag{5.2.2}$$

> Here Yann walks us through a simple calculation of the information you gain from seeing the true distribution compared to your prior. In general, that quantity is the KL divergence (see https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence#Bayesian_updating). In this case your prior is $p$ and your true distribution is $q$, and they go into the KL divergence as $\text{KL}(q, p)$.
>
> He does this because we want to minimize the information content of the latent variable. If you believe in data-science epistomology, you'd like your model to be as unbiased as possible, so you'd like to use this metric to make your prior as uninformed as possible.

We can construct a loss function

$$\mathcal{L}(q, y, w) = \int_z q(z|y) E_w(y, z) + \frac{1}{\beta} \int_z q(z|y) \log \frac{q(z|y)}{q(z)} \tag{5.2.3}$$

$$\text{argmin}_q \mathcal{L} = \frac{e^{-\beta E_w(y,z)}}{Z} \tag{5.2.4}$$

### 5.2.1 Lagrangian Mechanics

Consider a feedforward network $f(x) = (f_T \circ ... \circ f_0)(x)$. We denote the internal activations as $z = \{z_0, ..., z_T\}$,There is a cost function

$$L(z, w, \lambda, y) = C(y, z_T) + \sum_{k=0}^{T} \lambda_k (z_{k+1} - f_k(z_k; W_k) \tag{5.2.5}$$

where $\lambda_k$ are lagrange multiplier, $W_k$ is the weights associated with $f_k$. The backprop rules are

$$\frac{\partial L}{\partial \lambda_k} = 0 \implies z_{k+1} = f_k(z_k; W_k) \tag{5.2.6}$$

$$\frac{\partial L}{\partial z_k} = 0 \implies \lambda_k = \frac{\partial f_k(z_k; W_k)}{\partial z_k} \lambda_{k+1} \tag{5.2.7}$$

$$\frac{\partial L}{\partial W_k} = 0 \implies \frac{\partial C}{\partial W_k} + \frac{\partial f_k(z_k; W_k)}{\partial W_k} \tag{5.2.8}$$

So basically, he wanted to show that backprop constructs a canonical momentum? [I'm sorry Yann I don't see it, I'll have to go over this again. Which one is canonical position and momentum? I think it's $z$ is the position, and $\lambda$ is the momentum (as it is defined via a $z$ derivative). How do they define dual spaces?]

### 5.2.2

Consider the model $s_a = \text{Enc}(a)$ and a transformation $\hat{s}_y = f(s_x)$. Also consider we have a measure of information of the encoders $I(s_x)$, $I(s_y)$. We construct a loss function

$$\mathcal{L} = C(\hat{s}_y, s_y) + \lambda_x I(s_x) + \lambda_y I(s_y) \tag{5.2.9}$$

Yann spends some time to talk about different measures of information of the encoder.

1. You can assume $I(s) \in \mathbb{R}^d$, all dimensions are statistical independent. And here if you compute the entropy, it'll be an overestimate.
    - We can get around this by adding a regularizer in our loss function s.t. our encoder whitens the data / make it as independent as possible.
    - 
2. We can att

hi

# Chapter 6

# Stéphane Mallat: Score Based Diffusion & Renormalization Group Flow

## 6.1 Overview

We'll be going from score-diffusion and eventually make our way to the renormalization group.

The setup is you're given a set of input data $\{x_i : x_i \in \mathbb{R}^d\}_{i \leq n}$ which (we believe) to be sampled from a distribution $p(x)$. Your objective is to reconstruct this probability distribution.

Doing this problem in the physics context, the canonical example will be attempting to recreate samples from a Ising / $\phi^4$ model. Some more advanced problems are turbulence, cosmology, metrology, and climatology. Apart from physics, you can also generate new image (i.e. conditional generation), and use it to solve inverse problems.

As you start to think more about this problem, it might seem hopeless to learn $p(x)$. This is because the problem suffers from the **curse of dimensionality**. A crude way to calculate this is to first assume $p(x)$ is Lipschifts

**Definition 9 (Lipschifts)** *That is* $|p(x) - p(x')| \leq K \underbrace{||x - x'||}_{\epsilon \in [0,1]^d}$

This means you'll need $\sim C\epsilon^{-d}$ amount of data... You can see you're screwed.

We can start to tackle this problem by making a key assumption, factorize the distribution

$$p(x) = \prod_k p_k(x) \implies \log p(x) = \sum_k \log p_k(x) \tag{6.1.1}$$

Two problems now emerge

1. How do we choosing this factorization?
2. How do we prevent it from just simply memorizing a distribution?

The first method is to assume $p(x)$ is a empirical distribution, by this I mean $p(x) = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}$ (which are get all the observed data, and draw it uniformly).

### 6.1.1 Transport

By **transport**, we mean constructing a new function $p_t(x)$ which is a probability distribution $\forall t$, with boundary conditions $p_{t=0}(x) = p(x)$ and $p_{t=T}(x) = p_{gaussian}(x)$. So basically, can we find a new probability distribution which moves from complex things to NOT complex things.

There is a **forward process**, which is some noising operator. And there is a **inverse process**, which undo's the noise (usually involves the ML algorithm). What we'll end up seeing is that these processes ends up being equivalent to RG discovered by Kadanoff & Wilson.

A particular ansatz we can take is

$$p(x) = \underbrace{p(x_T)}_{\text{Gaussian}} \prod_{t=T+1}^{1} p(x_t/x_{t-1}) \tag{6.1.2}$$

### 6.1.2 Outline

1. Energy Based Models, GANs, and Normalizing Flows.
2. Score based diffusion, Fokker Planck, and denoising
3. Generalization and memorization.
4. Renormalization Group
5. Particular models: simple models based on wavelet / scattering transforms.

## 6.2 Historical Models: Energy Based Models, GANs, Normalization Flows

### 6.2.1 Energy Based Models (EBM)

When we say we have an energy based model, we have a probability distribution over $x = (x_1, ..., x_d)$ which takes the form

$$p_\theta(x) = \frac{e^{-U_\theta(x)}}{Z_\theta}, \quad \text{s.t. } Z_\theta = \int dx\, e^{-U_\theta(x)} \tag{6.2.1}$$

**Example 1** *An example model is a perturbation away from a Gaussian*

$$U_\theta(x) = x^T \Sigma_\theta^{-1} x + \theta \sum_i V(x_i) \tag{6.2.2}$$

*In physics the first one is the kinetic energy term, and the second is the potential energy term. You could start by making $U_\theta$ a neural network.*

**Example 2** *Another example model is the exponential family*

$$U_\theta(x) = \langle \theta, \Phi(x) \rangle \tag{6.2.3}$$

*Notice, this is the Boltzmann distribution in physics, where $\theta \leftrightarrow \beta$ (inverse temperature) and $\Phi(x) \leftrightarrow H(x)$ (Hamiltonian)*

**Likelihood**

**Definition 10 (Likelihood)** *The negative likelihood $\ell(\theta)$ of probability distribution p*

$$\ell(\theta) = -\mathbb{E}_{x\sim p}\log p_\theta(x) = KL(p||p_\theta) + H[p] \tag{6.2.4}$$

*where KL is the Kullback-Lieber divergence, and H is the entropy.*

*Our objective will be choose $\theta$ s.t. we minimize the negative likelihood*

$$\theta^* = argmin_\theta\ \ell(\theta) \tag{6.2.5}$$

To minimize this, we end up needing to calculate gradients of the likelihood. In the case of energy based models, that would mean we have to calculate the normalization $\nabla_\theta \log Z_\theta$, which in practice is quite hard. However, we can massage the problem into

$$\begin{aligned}\nabla_\theta\ell(\theta) &= \mathbb{E}_{x\sim p}[\nabla_\theta U(x)] + \log Z_\theta &(6.2.6)\\ &= \mathbb{E}_{x\sim p}[\nabla_\theta U_\theta(x)] - \mathbb{E}_{x\sim p_\theta}[\nabla_\theta U_\theta(x)] &(6.2.7)\end{aligned}$$

So now the problem becomes figuring out how to compute $\mathbb{E}_{x\sim p_\theta}$ which involves spending time to construct a Markov Chain Monte Carlo method.

### 6.2.2 GANs

GAN stands for Generative Adversarial Network, the original paper is by Goodfellow & Bengio (2014). What you do is you construct two neural networks $G_{\theta_G}$ and $D_{\theta_D}$ and make their tasks adversarial to one another. $G$'s task is to generate images from noise $z \sim \mathcal{N}(0, 1)$, and $D$'s task is to figure out which image was real vs generated. So to recap, $G : z \to$ Images and $D$ : Images $\to [0, 1]$, where your goal is that

$$D_{ideal}(x) = \begin{cases} 1 & \text{if } x \text{ is a real image} \\ 0 & \text{if } x \text{ is a fake image} \end{cases} \tag{6.2.8}$$

The loss function is

$$\mathcal{L}(\theta_D, \theta_G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim \text{Gaussian}}[\log(1 - D(G(z))] \tag{6.2.9}$$

What ends up happening in practice is that GANs almost always memorize the data instead of learning to gneerate. Another way of saying this is this is a difficult nash-equilibrium to obtain, or you see **mode collapse**. This is why people used them for image generation, and couldn't use them for physics data...

A small aside... If you successfully find the global minimizer/maximizer

**Lemma 1** *The optimal discriminator ends up being*

$$\begin{aligned}D^* &= argmax_D\ \mathcal{L}(D, G) &(6.2.10)\\ &= \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} &(6.2.11)\end{aligned}$$

**Theorem 2**

$$\min_G \max_D \mathcal{L}(D, G) \iff p_G = p_{data} \tag{6.2.12}$$

Both are theoretical results, so they don't consider using finite number of samples.

**Conditional GANs**

What if you want to condition your learned distribution on some variables (i.e. text to image, or in-painting). Your loss function will be

$$\mathcal{L}(D, G) = \mathbb{E}_{p_{data}}[\log D(x|y)] + \mathbb{E}_{z\sim p_{data}}[\log(1 - D(G(x|y)))] \tag{6.2.13}$$

### 6.2.3 Normalizing Flows

In this problem, we have an easy to sample distribution $p_z$ (which is Gaussian), and difficult to sample distribution $p_x$ (which is the target distribution). The way we'll do transport is to learn a diffeomorphism $T_\theta$, and now we can do change of basis between the two

$$p_x(x) = p_z(T(x)) \det \mathcal{J}_T(x) \tag{6.2.14}$$

where $\mathcal{J}_T$ is the Jacobian of $T$.

The likelihood ends up being

$$\ell(\theta) = -\mathbb{E}_{x\sim p}[\log p_z(T_\theta(x))] - \mathbb{E}_{x\sim p}[\log|\det -\mathcal{J}_{T_\theta}|] \tag{6.2.15}$$

You may be thinking that the normalization constant (2nd term) looks quite scary... However with the right ansatz, it's analytically tractable. Here's how to do it.

First, break the diffeomorphism into smaller diffeomorophisms $T_\theta^{-1} = T_1^{-1}...T_k^{-1}$. So the infal term becomes $\log|\det \mathcal{J}_{T_\theta}(x)|= \sum_i \log|\det \mathcal{J}_{T_i}(x_i)|$.

Second, we can use block inverse formula to get a nice closed form expression. To keep track of the block inverse, let's split up $x \in \mathbb{R}^d = (x_0, x_1) \in \mathbb{R}^{d_0} \times \mathbb{R}^{d_1}$

$$T_\theta(x) = (x_0, e^{S_\theta(x_0)} \odot x_1 + t_\theta(x_0)) \tag{6.2.16}$$

$$T_\theta^{-1}(z) = (z_0, (z_1 - t_\theta(z_0)) \odot e^{-S_\theta(z_0)} \tag{6.2.17}$$

where $\odot$ denotes the Hadamar product, and $z = (z_0, z_1)$. You can now show that the $\log \det$ term picks up a nice expression

$$\implies \log|\det \mathcal{J}_{T_\theta}(x)|= \sum_{i=1}^{d} s_\theta(x_0) \tag{6.2.18}$$

Finally we have our likelihood

$$\ell(\theta) = -\mathbb{E}_{x\sim p}[\log p_z(T_\theta(x))] - \sum_{i=1}^{d} \mathbb{E}_{x\sim p}[s_\theta(x_0)] \tag{6.2.19}$$

**Continuous Flow**

Consider the **Neural ODE**

$$\frac{dx_t}{dt} = v_\theta(x_t, t) \qquad \text{(Continuous)} \tag{6.2.20}$$

$$x_{t+\epsilon} = x_t + \epsilon\ v_\theta(x_t, t) \qquad \text{(Naive Discretization)} \tag{6.2.21}$$

where $v_\theta : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d$ is a residual (skipped connections) neural network, which is assumed to be Lipshifts.

However, what if we want to stop thinking about transporting just particles, but rather transporting probability distribution mass? Recall the **Liouville Equation**

$$\begin{aligned}\frac{\partial p_t}{\partial t} &= -\nabla \cdot (v_t \; p_t) && (6.2.22)\\ &= -\underbrace{v_t \cdot \nabla p_t}_{\text{Convection}} - \underbrace{p_t(\nabla \cdot v_t)}_{\text{compression / dilation}} && (6.2.23)\end{aligned}$$

Why would you want to do this? In metrology, it's quite difficult to make a point estimate of the future (due to chaoticity of fluid dynamics), so instead what if we make a probabilistic inference! This frame work more well posed!

## 6.3 Score Based Diffusion

If you want to add diffusion, you recover the **Fokker Planck Equation**

$$\frac{\partial p_t}{\partial t} = -\nabla \cdot (p_t(x)v_t) - \Delta \left(\frac{1}{2}\sigma^2(x)p_t(x)\right) \iff dx_t = v_t(x_t)dt + \sigma dW_t \tag{6.3.1}$$

where $\Delta$ is the Laplacian operator, where we interpret $\text{Law}(x_t) = p_t$.

> These ideas were developed by Physicists Reckstein & Ganguli at Stanford in 2015. It took a whole 5 years within the same university to make its way to the CS department where Song & Enman published their seminal paper.
>
> However this has been thought about from a signal processing perspective by Zhadoki & Semaicelli, and Horr & et al.

Now we can finally begin Score Based Diffusion!

### Orstein Ulhrbeck SDE

Consider the stochastic differential equation

$$dx_t = -\kappa x_t dt + \sigma dW_t \tag{6.3.2}$$

To solve it, perform the change of variable

$$\begin{aligned} y_t &= e^{\kappa t} x_t && (6.3.3)\\ dy_t &= e^{\kappa t}(-\kappa x_t dt + \sigma dW_t) + \kappa e^{\kappa t} x_t dt = e^{\kappa t}\sigma dW_t && (6.3.4)\end{aligned}$$

This yields

$$x_t = e^{-\kappa t}x_0 + \sigma\sqrt{\frac{1-e^{-2\kappa t}}{2\kappa}}z \tag{6.3.5}$$

where $z \sim N(0, \mathbb{I})$. Inspecting the limiting cases of our solution, $x_{t=0} = x_0$ and $x_{t\to+\infty} = \frac{\sigma}{2\sqrt{2\kappa}}z$. So this is a process is a linear interpolation between our input $x_0$ and white noise $z$.

> Consider the $\kappa = 1$ Orstein Ulhbreck SDE.
>
> Moving to the distribution picture, we can see
>
> $$p_t = (e^{\kappa t} p(\cdot e^{\kappa t})) * g_\sigma \tag{6.3.6}$$
>
> where $g_\sigma = \mathcal{N}(0, 1 - e^{-2\kappa t})$. We can also see this forms a semi-group, we don't get an inverse for free...
>
> Another way to see this is to Fourier transform. Generically if you have $h = f * g$, then $\hat{h} = \hat{f}\hat{g}$. In our case $\hat{g}(\omega) \sim e^{-(1-e^{-2\kappa t})||\omega||^2/2}$. You can see this expression explodes as time goes on

**Denoising**

Now our task is to denoise the OU process. For notation, let's use $y_t = x_{T-t}$ and $q_t = p_{T-t}$. We can plug our $q_t$ into the Fokker Planck (6.3.1), and derive a time evolution on the distribution for the reverse process

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (-p_t x_t) - \Delta p_t = 0 \qquad \text{Forward in time} \tag{6.3.7}$$

$$\implies \frac{\partial q_t}{\partial t} + \nabla \cdot (q_t y_t) + \Delta q_t = 0 \qquad \text{Backwards in time} \tag{6.3.8}$$

Notice your have inverse diffusion, which is completely unstable. Show how can we rewrite this to become more stable? Notice the trick

$$\Delta q_t = \nabla \cdot (q_t \underbrace{\nabla \log q_t}_{\text{score}}) \tag{6.3.9}$$

We finally get

$$\frac{\partial q_t}{\partial t} + \nabla \cdot (q_t y_t - (1+\alpha) q_t \nabla \log p_t) - \alpha \Delta q_t = 0 \tag{6.3.10}$$

$$dy_t = [y_t + (1+\alpha)\nabla \log q_t] + \sqrt{2\alpha}\, dW_t \tag{6.3.11}$$

We now have a generic way to talk about the reverse-time process!

**Denoising OU**

We have the $\kappa = 1$ OU,

$$x_t = e^{-t} x + \sqrt{1 - e^{-2t}} z \tag{6.3.12}$$

Taking the coordinate change $\tilde{x}_t = e^{t x_t}$, you now find at each point in time, you observe $y_t$

$$y_t = x + \sqrt{e^{2t} - 1} z \tag{6.3.13}$$

Recall we are attempting to denoise this process. In signal processing, your trying to minimize the mean square error

$$\mathbb{E}_{x,z}[||\hat{x}(y_t) - x||^2] \tag{6.3.14}$$

So given an observed noisy input $y_t$, can we construct a $\hat{x}$ (which takes in $y_t$) and yields $x$. We can use **Tweety's Formula** to construct the optimal denoiser

**Theorem 3** *Finding the minimum of*

$$\mathbb{E}_{x,z}[||\hat{x}(y_t) - x||^2] \tag{6.3.15}$$

*Ends up being*

$$\hat{x}(x_\sigma) = \mathbb{E}[x|x_\sigma] = x_\sigma + \sigma^2 \nabla_{x_\sigma} \log p_\sigma(x_\sigma) \tag{6.3.16}$$

*Proof:*

$$\begin{aligned}
p_\sigma(x_\sigma) &= \int p(x_\sigma|x)p(x)dx && (6.3.17)\\
\nabla p_\sigma(x_\sigma) &= \int g_\sigma(x_\sigma - x)p(x)dx && \textit{Due to solution of OU} \quad (6.3.18)\\
&= \frac{1}{\sigma^2}\int (x_\sigma - x)g_\sigma(x_\sigma - x)p(x)dx && (6.3.19)\\
\frac{\sigma^2 \nabla p_\sigma(x_\sigma)}{p_\sigma(x_\sigma)} &= \int (-x_\sigma + x)p_\sigma(x|x_\sigma)dx && p(x)/p_\sigma(x_\sigma) = p_\sigma(x|x_\sigma) \quad (6.3.20)\\
&= -x_\sigma + \hat{x}(x_\sigma) && (6.3.21)
\end{aligned}$$

*where* $g_\sigma(x_\sigma - x) = C_\sigma e^{-||x_\sigma - x||^2/2\sigma^2}$ *is a Gaussian distribution. You can now see that the score naturally arises in denoising problems.*

As a machine learning person, once you see, "minimize square error", you should just plug that into a neural network and let it rip. So you should try to $\min_\theta \ell(\theta) = \min_\theta \mathbb{E}_{x,z}[||\hat{x}_\theta(x_\sigma) - x||^2]$, and then by Tweedy's formula, at inference time you perform $\frac{\hat{x}_\theta(x) - x}{\sigma^2} = \nabla_{x_\sigma} \log p_\sigma(x_\sigma)$

**Example of Denoising Gaussian**

Consider the distribution $p_\sigma(x) = \frac{e^{-U_\sigma(x)}}{Z_\sigma}$, which means the score is $\nabla_x \log p_\sigma(x) = -\nabla_x U_\sigma(x)$. $p_\sigma$ is Gaussian when $U_\sigma(x) = \frac{1}{2}x^T C^{-1} x$. In this case let's choose a factorization $C\sigma = C + \sigma^2 \mathbb{I}$. Using Tweedy's Formula, our optimal denoiser (that is $\hat{x}(x_\sigma) \approx x$) is given by

$$\begin{aligned}
\hat{x}(\sigma) &= [\mathbb{I} - (C + \sigma^2 \mathbb{I})^{-1}\sigma]x_\sigma && (6.3.22)\\
&= (C + \sigma^2 \mathbb{I})^{-1}(C + \sigma^2 \mathbb{I} - \sigma^2 \mathbb{I})\hat{x})_\sigma && (6.3.23)\\
&= (C + \sigma^2 \mathbb{I})^{-1} C x_\sigma && (6.3.24)
\end{aligned}$$

So in this case, the optimal denoiser is found via a linear regression!

### 6.3.1 Generalization vs Memorization?

We want to ask the question does the neural network actually find a score which generalizes beyond the training data? He explains a couple interesting tests https://arxiv.org/abs/2310.02557

### 6.3.2 Architecture of CNNs

In a lot of these problems, CNNs (specifically UNets) end up being the more successful architectures. So perhaps we should spend some time to deconstrucitng them.

### Convolutional Operator

A convolutional operator (in ML), takes an image (Length, Width, Channels) → (Length', Width', Channe Over pixel space (Length, Width) it applies a convolution, over channel space we apply a linear transformation. The $\ell$'th nodes are given by

$$x_{\ell+1}(i,c') = \rho\left(\sum_{c\in\text{Channels}}\sum_{\tau\in\{1,2\}} W(\tau,c,c')x_\ell(i-\tau,c)]\right) + b \tag{6.3.25}$$

where $\rho$ is your activation function, $b$ is your bias. The $\sum_{c\in\text{Channels}}$ is the linear transformation between channel, and the $\sum_{\tau\in\{1,2\}}$ is the convolution. Notice that you can effectively reframe the operation inside of $\rho$ as a single linear operator.

Traditionally, as you go deeper into the neural network, people will reduce the spatial dimension and increase the number of channels.

### Explaining Bias in the Model

Now we want to explore they this architecture produces bias.
If assume there's no bias $b = 0$. The tweedy's formula

$$\hat{x} = x_\sigma + \sigma^2 \underbrace{\nabla \log \tilde{p}_\sigma(x_\sigma)}_{\equiv s_\sigma(x_\sigma)} \tag{6.3.26}$$

Since $s_\sigma(x_\sigma)$ is piecewise linear, we can rewrite it as $s_\sigma(x_\sigma) = -\mathcal{J}_{s_\sigma}(x_\sigma)x_\sigma$. This means we can rewrite Tweedy's formula as

$$\hat{x} = \left(\mathbb{I} + \sigma^2\mathcal{J}_{s_\sigma}(x_\sigma)\right)x_\sigma \tag{6.3.27}$$

we can decompose the interior in an eigenbasis.

$$\hat{x} = \sum_k h_k\langle x_\sigma, \psi_k\rangle\psi_k \tag{6.3.28}$$

This means the difference between prediction $\hat{x}$ and the true value $x$, we get

$$||\hat{x} - x||^2 = \sum_k |\langle \hat{x}, \psi_k\rangle - \langle x, \psi_k\rangle|^2 \tag{6.3.29}$$

> Let's work through an example, let's take the probability distribution to be $p(x) = e^{-\frac{1}{2}x^T C_\sigma^{-1} x}/Z$ where $C_\sigma = C + \sigma^2\mathbb{I}$. This means $\nabla^2 \log p(x) = -C_\sigma^{-1}$, which has eigenvalues $-\frac{1}{\beta k + \sigma^2}$

Mallat then presents some numerical experiments on visualizing these learned basis vectors (see https://arxiv.org/pdf/2310.02557, same link as the previous one).

## 6.4 Wavlets

### 6.4.1 Optimal Denoising

Consider observing a noisy signal $y = x + z$, where $x$ is the true signal, and $z \sim \mathcal{N}(0, \sigma^2\mathbb{I})$. Our objective is to find a prediction for the true signal $\hat{x} = Hy$ which is a linear function of the noisy signal. We want this predictor to minimize $\epsilon^2 = \mathbb{E}[||Hy - x||^2]$.

**Theorem 4 (Wiener Filter)** *The Winer Filter is*

$$H = (C + \sigma^2 \mathbb{I})^{-1} C \tag{6.4.1}$$

*It has eigenvalues*

$$\frac{\beta k}{\beta k + \sigma^2} \in [0, 1] \tag{6.4.2}$$

*You can show that this predictor minimizes the correlation* $\mathbb{E}[(\hat{x} - x)y]$

**Definition 11 (PCA Basis)** *Consider the basis vectors* $\{\psi_k\}_k$ *comparised of wavelets*

$$\psi_k(n) = \frac{1}{\sqrt{d}} e^{i2\pi nk/d} \tag{6.4.3}$$

$$\langle C\psi_{k'}, \psi_k \rangle = \beta k = \mathbb{E}[|\langle x, \psi_k \rangle|^2] \tag{6.4.4}$$

So, now we can change of basis our true signal $Hy = \sum_k h_k \langle y, \psi_k \rangle \psi_k$. So now how do we construct the coefficients $h_k$?

$$x = \sum_k \langle x, \psi_k \rangle \psi_k \tag{6.4.5}$$

$$\mathbb{E}||Hy - x||^2 = \sum_k \mathbb{E}\left|\left|(\lambda_k(x) + 1)\langle x, \psi_k \rangle + h_k \langle z, \psi_k \rangle\right|\right|^2 \qquad \text{Using } y = x + z \tag{6.4.6}$$

$$= \sum_k \left|\left|(\lambda_k(x) + 1)\langle x, \psi_k \rangle + h_k \sigma^2\right|\right|^2 \tag{6.4.7}$$

Now I can optimize to find the minimum of this quantity...

$$0 = \partial_{h_k} \mathbb{E}||Hy - x||^2 \implies h_k = \frac{|\langle x, \psi_k \rangle|^2}{|\langle x|\psi_k \rangle|^2 + \sigma^2} \qquad \text{Minimizer} \tag{6.4.8}$$

$$\therefore \epsilon = \sum_k \frac{\sigma^2 |\langle x, \psi_k \rangle|^2}{|\langle x, \psi_k \rangle|^2 + \sigma^2} \tag{6.4.9}$$

### Binary Basis

Ok how does the error change if I only allow for binary decisions on the coefficients (that is you either have that basis element on or off)

$$\lambda_k = \begin{cases} 1 & |\langle x, \psi_k \rangle| \geq \sigma \\ 0 & |\langle x, \psi_k \rangle| < \sigma \end{cases} \tag{6.4.10}$$

What ends up happening is

$$\epsilon^2 = \sum_k [(\lambda_k - 1)^2 |\langle x, \psi_k \rangle|^2 + \lambda_k \sigma^2] \tag{6.4.11}$$

If we use the fact $\frac{1}{2}\min(a, b) \leq \frac{ab}{a+b} \leq \min(a, b)$, we can show

$$\epsilon_m \leq \epsilon^2 \leq 2\epsilon_m \tag{6.4.12}$$

so basically you only loose a factor of two...

**Thresholded**

What if you consider the expansion

$$Hy = \sum_k \mathrm{sth}_T(\langle y, \psi_k\rangle)\psi_k \tag{6.4.13}$$

$$\mathrm{sth}_T(a) = \begin{cases} a - \mathrm{sign}(a)T & |a|> T \\ 0 & |a|< T \end{cases} \tag{6.4.14}$$

If we set $T = \sigma\sqrt{2\log d}$, you can show

$$\epsilon_m \leq \epsilon \leq (2\log d + 1)[\epsilon_m + \sigma^2] \tag{6.4.15}$$

You can also show that you cannot do asymptotically better as $d \to \infty$. So this choice is optimal.

### 6.4.2 Compression

In practice, denoising and compression end up being similar tasks. So if you have a good denoising basis, you should try it out on compression. Why is this? Consider the problem where e want to minimize the error

$$\epsilon_m \sim \epsilon_b = \sum_k \min(|\langle x, \psi_k\rangle|^2, \sigma^2) \tag{6.4.16}$$

$$= M\sigma + 2\sum_{|\langle x,\psi_k|>\sigma^2} |\langle x, \psi_k\rangle|^2 \tag{6.4.17}$$

So imagine you want to construct a approximation $x_M$ for signal $x$

$$x = \sum_{k=1}^{d}\langle x, \psi_k\rangle\psi_k \tag{6.4.18}$$

$$x_M = \sum_{k\in I_M}\langle x, \psi_k\rangle\psi_k \tag{6.4.19}$$

$$\therefore ||x - x_M||^2 = \sum_{k\notin I_M}|\langle x, \psi_k\rangle|^2 \tag{6.4.20}$$

Well $I_M = \{k : |\langle x, \psi_k\rangle| \geq \sigma\}$

## 6.5 Wavelet Slides

Since we're trying to denoise a time series signal. There are often discontinuous jumps, yet high frequency noise. So we want to construct a basis where we can pick out high frequency noise, yet also have short time scale. Fourier transforms work entirely in frequency or time basis, wavelets allows us to go "somewhere in between"...

Let's begin with the simple Haar wavelet

$$\{\psi_{j,n}(t) = \frac{1}{\sqrt{2^j}}\psi(\frac{t - 2^j n}{2^j})\}_{(j,n)\in\mathbb{Z}^2} \tag{6.5.1}$$

This forms an orthogonal basis of $L^2(\mathbb{R})$

## 6.6 Renormalization Group & Hierarchies

We start by recalling physics. Consider a physical state at equilibrium $x \in \mathbb{R}^d$. You'll measure the states according to a Gibbs distribution

$$p_t(x) = \frac{e^{-U_t(x)}}{Z} \tag{6.6.1}$$

Our goal is to find a parameterized version of $U(x)$. Often $U(x)$ is not convex, the associated hessian is singular (non-invertible), and it'll emit long-range correlations. Thinking about these correlations, this will depend on the "scale" that we probe at, which we'll denote by $t$. This motivates the usage of the **renormalization group**.

There are perhaps naive ways of performing RG (Kadnoff RG / Real Space RG).

Suppose your ansatz $U_t(x) = \langle O_t, \phi(x) \rangle$. So now let's ask the question, how do we go from $p_{j-1} \to p_j$ (that is change scales). Let $x_{j-1} = (x_j, \bar{x}_j)$ (kept variables, and complement of kept variables which are orthogonal) (orthognality is prevent Jacobian from popping out). We can now simply integrate out $\bar{x}_j$ and have the effective theory

$$\int p_{j-1}(x_j, \bar{x}_j) d\bar{x}_j = p_j(x_j) \tag{6.6.2}$$

and we'd hope that calculation of sufficient statistics is preserved

$$\langle \mathcal{O}(x_j) \rangle_{p_j} = \langle \mathcal{O}(x_j, \bar{x}_j) \rangle_{p_{j-1}} \qquad \text{(which holds up until the "scale")} \tag{6.6.3}$$

In Wilsonian RG, you end up performing the calculation

$$p_{j-1}(x_{j-1}) = p_{j-1}(x_j, \bar{x}_j) = p_j(x_j) \bar{p}_j(\bar{x}_j | x_j) \tag{6.6.4}$$

where you have created $\bar{p}_j$ s.t. it is convex and local.

For notational simplicity let's denote

$$\bar{p}_j(\bar{x}_j | x_j) = \frac{e^{-\bar{U}_j(\bar{x}_j, x_j)}}{\bar{Z}_j} \tag{6.6.5}$$

and now you can see $U_{j-1}(x_{j-1}) = U_j(x_j) + \bar{U}_j(\bar{x}_j, x_j)$.

In diffusion, you never want to calculate the transition between $p_{t=0}$ and $p_{t=T}$ (Gaussian to target distribution), we want to break it up into little steps (aka annealing)

$$p_0(x_0) = p_T(x_T) \prod_{j=T}^{1} \bar{p}_j(\bar{x}_j | x_j) \tag{6.6.6}$$

So intead of computing $O_0$ in $U_0 = \langle O_0, \phi(x) \rangle$. You want to instead of compute $\{O_s : \bar{O}_j\}_{1 \leq j \leq T}$.

**Sampling**

We use Langevin

$$dx_t = \nabla \log p(x) + \sqrt{2} dW_t \tag{6.6.7}$$

When you're using a maximum entropy model $p(x) \propto e^{-U(x)}$

$$dx_t = -U(x) + \sqrt{2} dW_t \tag{6.6.8}$$

- If $U(x)$ is convex, your mixing time is proportional to the condition number of the Hessian (associated with $U$)
- If $U(x)$ is not convex, it'll still converge, but it'll take a long time...

> For example, we can consider using wavlets on a gaussian
>
> $$p_0(x) = \frac{e^{-U}}{Z} \tag{6.6.9}$$
>
> Where $U(x) = x^T Kx/2 = (K)^T(\frac{1}{2}x^T x) = \langle O, \phi(x) \rangle$. This means $U(x)$ is convex, and therefore mixing time is set by the condition number is set by $K$.
>
> [Mallat makes a point that wavelets keep this condition number invariant as you vary the scale... Meaning, perform the orthognal decomposition $x_{j-1} = (x_j, \bar{x}_j)$ using wavelets, and now your $K_{j-1} \to \bar{K}_j$ should keep the same condition number. This seems a bit nontrivial to me, and I'm not sure how to see this...]
>
> In physics this is called renormalization. In machine learning language this is called batch-normalization.

**Training**

We perform maximum likelihood to find the parameters. We accomplish this by minimizing the KL divergence. So let the loss function be

$$\begin{aligned} \ell(\theta) &= \mathrm{KL}(p|p_\theta) \qquad (6.6.10) \\ &= -\mathbb{E}_{x\sim p}[\log p_\theta] + (\text{Constant w.r.t. } \theta) \qquad (6.6.11) \end{aligned}$$

If $U = \langle \theta, \phi(x) \rangle$, then $\nabla^2 \ell(\theta) = \mathrm{Cov}_{p_\theta}[\phi]$

## 6.7 Continuous Ising, $\phi^4$ Model

Consider the target distribution defined by $U_0(x) = \frac{1}{2}x^T Kx + \sum_i V(x(i))$. In particular let's focus on the case $V(x) = \sum_a \alpha_k p_k(a)$. So in the $U = \langle O, \phi(x) \rangle$ formulation, $O = (K, \alpha_k)$ and $\phi(x) = (x^T x, \sum_i x^k(i))$.

Experiments in https://arxiv.org/abs/2207.04941 show that this works well!

## 6.8 Can we use this in other things?

See Wavelet Score-Based Generative Modeling, Learning Multi-scale local conditional probability model of images, scattering spectral models for Physics.

hi

# Chapter 7

# Marc Mézard: Statistical Physics of Generative Diffusion

## 7.1 Overview

1. Langevin-Fokker Planck & Ornstien Uhlenbeck
2. Principles of Generative Diffusion
3. A simple case: Gaussian Data
4. Aside: Related approaches ODE and Stochastic Localization & Interpolants
5. Intermission: Thermodynamic Score
6. A harder case: Curie Weiss
7. Speciation Transition: Classifier Free Guidance
8. Generalization vs Memorization

## 7.2 Recall: Stochastic Processes

### 7.2.1 Langevin Equation

Let $x \in \mathbb{R}^d$, and has the state equation

$$\frac{dx}{dt} = F(x) + \eta(t) \tag{7.2.1}$$

where $F : \mathbb{R}^d \to \mathbb{R}^d$ is the force field, and $\eta(t)$ is a Gaussian with properties $\langle \eta(t) \rangle = 0$ and $\langle \eta(t)\eta(t') \rangle = 2\delta(t - t')$. In stochastic process notation it is

$$dx_t = f(x)dt + \sqrt{2}dW_t \tag{7.2.2}$$

> In general, Mezard will work in continuous time. However whenever we want to make a comment about implementations on a computer, we'll use the Ito descretization.
>
> $$x(t + \delta t) = x(t) + \delta t \ F(x(t)) + \int_t^{t+\delta t} d\tau \eta \ (\tau) \tag{7.2.3}$$

and

$$\delta x = \delta t F(x(t)) + \sqrt{2\delta t} z_t \tag{7.2.4}$$

where $z_t$ is a unit gaussian.

Note the integral of the time-dependent gaussian is given by

$$\langle y \rangle := \int_t^{t+\delta t} \eta(\tau) d\tau \tag{7.2.5}$$

has the following properties $\langle y \rangle = 0$ and $\langle y^2 \rangle = 2\delta t$.

### 7.2.2 Fokker Planck

Consider the conditional distribution $p_t(x|x_0)$, the equation which keeps it probabliity distribution is the Fokker Planck equation

$$\frac{\partial p_t}{\partial t} = \Delta p_t - \nabla \cdot (F p_t) \tag{7.2.6}$$

where $F$ is a vector field. You can then ask what's condition of the stationary distribution, that is $\partial_t p_t = 0$, it must satisify

$$\Delta p_t + \nabla \cdot (p \nabla V) = 0 \tag{7.2.7}$$

$$p_{stationary}(x') = \frac{1}{Z} e^{-V(x)} \tag{7.2.8}$$

Note that our usage of $\sqrt{2}$ in the previous section was to ensure we get a temperature $\beta = 1$.

### 7.2.3 Ornstein Uhlenbeck

This is when $V(x) = \frac{1}{2}|x|^2$. So the Langevin equation becomes

$$\dot{x} = F(x) + \eta(t) \implies \dot{x} = -x + \eta(t) \tag{7.2.9}$$

and your stationary distribution becomes

$$p_{st}(x) = e^{-|x|^2/2} \tag{7.2.10}$$

For example, if you have $x(t=0) = a \in \mathbb{R}^d$. Then the solution to the Langevin becomes

$$x(t) = a e^{-t} + \int_0^t e^{-(t-\tau)} \eta(\tau) d\tau \tag{7.2.11}$$

the second term is equivalent to the random variable $\mathcal{N}(0, 1 - e^{-2t})$, and we'll notate $\Delta t = 1 - e^{-2t}$.

So you can interpret this process as something which always goes to white noise.

### 7.2.4 General Time and Variance

Consider the equation

$$\frac{dx}{dt} = f(t)x(t) + g(t)\eta(t) \tag{7.2.12}$$

previously $g = 1$, but now $f : \mathbb{R}^d \to \mathbb{R}^d$ and $g : \mathbb{R}^d \to \mathbb{R}^d$. Solving the equation with the same boundary condtion as before ($x(t = 0) = a$), you get

$$x(t) = as(t) + s(t)\sigma(t)z(t) \tag{7.2.13}$$

$$\text{s.t.} \begin{cases} s(t) = \exp\left[\int_0^t d\tau \ f(\tau)\right] \\ \sigma(t)^2 = 2\int_0^t d\tau \left(\frac{g(\tau)}{s(\tau)}\right)^2 \end{cases} \tag{7.2.14}$$

> Example: Brownian Motion is recovered when $f = 0$ and $g = 1$.

## 7.3 Principals of Generative Diffusion

> For some background reading for this section there's: Sohl-Dickstein et al (2015), Yang Song & Stefano Ermon (2019), and a Review by Ling Yang et al (ArXiv:2209.00796).

In generative modeling, *the problem setup* is that you have an a target distribution $p_0(a)$ s.t. $a \in \mathbb{R}^d$, however you only have an empirical estimation of said target distribution $\{a^\mu\}_{\mu=1}^n$ (and was assume $a^\mu$ was sampled iid from the target distribution). Your goal is to use a ML to reconstruct & sample from the target distribution using only knowledge from the empirical distribution.

### 7.3.1 Forward Process

Let $a \in \mathbb{R}^d$ (where $a \sim p_0$), and $x(t = 0) = a$. The forward process is the OU process, which we recall as

$$\dot{x} = -x + \eta(t) \implies x(t) = ae^{-t} + \sqrt{\Delta_t} z_t \tag{7.3.1}$$

where $\Delta_t = 1 - e^{-2t}$. At time $t$ we say that $x \sim p_t(x)$ which has time evolution of the Langevin

$$\frac{\partial p_t}{\partial t} = \Delta p_t + \nabla \cdot (x p_t) \implies p_t(x) = \frac{e^{-(x - ae^{-t})/2\Delta_t}}{(2\pi\Delta_t)^{d/2}} = \mathcal{N}(x; ae^{-t}, \Delta_t) \tag{7.3.2}$$

This means that if $a \sim p_0$, then we have a joint probability distribution over $a$ and it's (partially) noised counter part $x$.

$$p_t(a, x) = p_0(a)\mathcal{N}(x_t; ae^{-t}, \Delta_t) \tag{7.3.3}$$

$$p_t(x) = \int da \ p(a, x) \tag{7.3.4}$$

$$p(a|x_t) = \frac{p(a)\mathcal{N}(x; ae^{-t}, \Delta_t)}{\int da' p_0(a')\mathcal{N}(x_t; ae^{-t}, \Delta t)} \tag{7.3.5}$$

### 7.3.2 Backwards Process

To notate the backwards process, let's notate our time notation. Previously $t$ starts at 0, and ends at $t_f$. In our backwards process, $\tau$ starts at 0 (which corresponds to $t = t_f$) and ends at $\tau_f$ (which correspond to $t = 0$). So $\tau = t_f - t$. If we plug in this coordinate change into our Langevin

$$-\frac{\partial p}{\partial \tau} = \Delta p + \nabla \cdot (xp) \implies \frac{\partial p}{\partial \tau} = -\Delta p - \nabla(xp) \tag{7.3.6}$$

Ok, restarting from the general Fokker Planck

$$\frac{\partial p}{\partial \tau} = \Delta p - \nabla(Fp) \tag{7.3.7}$$

What force-field $F$ do we need to recover the backwards process (7.3.6). If you ansatz $F = x + 2\nabla \log p$, you can recover the right thing. In general, if you want to undo the forward process (of OU process), the probability in the score is taken to be $p^F_{t_f-\tau}(x)$ (the probability of the forward process).

> A comment on notation. The forward probability distribution $P^F_t(x)$ and the backwards probability distribution $P^B_\tau(x)$ are related via
>
> $$P^F_t(x) = p^B_{t_f-t}(x) \tag{7.3.8}$$

In the case $t_f \gg 1$

$$p^F_{t_f}(x) = \frac{e^{-x^2/2}}{Z} \tag{7.3.9}$$

$$\log p = -\frac{x^2}{2} + C \tag{7.3.10}$$

$$\nabla \log p = -x \tag{7.3.11}$$

In the case $t \simeq t_f \gg 1$

$$F(x) = x + 2\nabla \log p = -x \tag{7.3.12}$$

### 7.3.3 Comment on Discretization

See Ho Jain Abbeel (NeurIPS 2020).

### 7.3.4 The Score

**Theorem 5 (Tweedie's Formula)**

Consider your probability $p_t(x) = \int da\ p_0(a)\frac{e^{-(x-ae^{-t})^2/(2\Delta_t)}}{\sqrt{2\pi\Delta_t}}$. This means the joint probability $p_t(a, x) = p_0(a)\mathcal{N}(x; ae^{-t}, \Delta_t)$ (where $a$ is $x_{t=0}$). The score of $p_t(x)$ becomes

$$\varphi = \nabla_x \log p_t(x) = \frac{1}{p_t(x)} \int da\ p_0(a) \frac{-(x - ae^{-t})}{\Delta_t} \frac{e^{-(x-ae^{-t})^2}/(2\Delta_t)}{\sqrt{2\pi\Delta_t}} \tag{7.3.13}$$

$$= -\frac{x}{\Delta_t} + \frac{e^{-t}}{\Delta_t}\langle a\rangle_{x,t} \tag{7.3.14}$$

where $\langle a\rangle_{x,t} = \int da\ a\ P(a|x,t) = \frac{\int da\ \ a\ \ p_0(a)e^{-(x-ae^{-t})^2/2\Delta t}}{\int da\ \ p_0(a)e^{-(x-ae^{-t})^2/2\Delta t}}$

### 7.3.5 Approximating the Score

Say $p_0$ is unknown, but we have a dataset $\{a^\mu\}_{\mu=1}^n \sim_{iid} p_0$. We will parameterize an estimate of the score $\varphi_\theta(x,t)$, so now we need a loss function to minimize. A typical loss function is

$$\mathcal{L} = \int_0^{t_f} \lambda(t)\, \underbrace{\mathbb{E}_{x\sim p_t}[|\varphi_\theta(x,t) - \nabla \log p_t(x)|^2]}_{L^t} \tag{7.3.15}$$

The $\lambda(t)$ is a Lagrange multiplier throughout time. Let's inspect the $L^t$ term. If you integrate by parts you can show

$$L^t = \mathbb{E}_x[|\varphi_\theta(x,t)|^2 - 2\varphi_\theta(x,t)\cdot\nabla \log p] + \text{Constant} \tag{7.3.16}$$

$$-2\mathbb{E}[\varphi_\theta(x,t)\cdot\nabla\log p] = -2\int dx \int da p_0(a) \frac{x-ae^t}{\Delta_t}\frac{e^{-(x-ae^{-t})^2/2\Delta_t}}{\sqrt{2\pi\Delta_t}}\varphi_\theta(x,t) \tag{7.3.17}$$

$$\therefore L^t = \mathbb{E}_{a\sim p_0}\mathbb{E}_{x\sim p_{OU}(x|a)}\left[\left|\varphi_\theta(x,t) + \frac{x-ae^{-t}}{\Delta_t}\right|^2\right] \tag{7.3.18}$$

Because the OU process has the dynamics

$$x = ae^{-t} + z\sqrt{\Delta_t} \implies \frac{x-ae^{-t}}{\Delta_t} = \frac{z}{\sqrt{\Delta_t}} \tag{7.3.19}$$

where $z \sim \mathcal{N}(0,1)$, this implies that $L^t$ is heuristically $\mathbb{E}_z[|\varphi_\theta(ae^{-t} + z\sqrt{\Delta_t}, t) + z|^2]$, so you can interpret this as matching noise $z$!

### 7.3.6 Various Scores

#### Perfect Score

Consider the model

$$p_0(a) = \frac{e^{-E(a)}}{Z},\ \varphi(x,t) = \frac{1}{\Delta_t}(-x + e^{-t}\langle a\rangle_{x,t}) \tag{7.3.20}$$

asdf

$$p(a|x,t) = \frac{1}{Z(x)} p_0(a) e^{-(x-ae^{-t})^2/2\Delta_t} \tag{7.3.21}$$

> See KPZ and Burger's Equation...

#### Empirical Score

If you only have a database $\{a^\mu\}_{\mu=1}^n$, the best thing you can do is $p_0^{emp} = \frac{1}{n}\sum_{\mu=1}^n \delta(a - a^\mu)$. This means our OU processed empirical becomes

$$p_t^{emp}(x) = \frac{1}{n}\sum_{\mu=1}^n \frac{e^{(x-a^\mu e^{-t})^2/2\Delta_t}}{(2\pi\Delta)^{n/2}} \tag{7.3.22}$$

Notice this is just a Gaussian mixture model. The empirical score is

$$\varphi^{emp}(t,x) = \nabla \log p_t^{emp}(x) \tag{7.3.23}$$

$$= \sum_\mu -\frac{x - a^\mu e^{-t}}{\Delta_t} e^{-(x-a^\mu e^{-t})^2/2\Delta_t} \Big/ \sum_\mu e^{-(x-a^\mu e^{-t})^2/2\Delta_t} \tag{7.3.24}$$

**Fitted Score**

$$\mathcal{L} = \mathbb{E}_{a,x\sim p_t(a,x)}[\varphi_\theta(x,t) + \frac{x - ae^{-t}}{\Delta_t}|^2 \tag{7.3.25}$$

$$= \frac{1}{n}\sum_\mu \mathbb{E}_{z\sim\mathcal{N}(0,1)}\left[\varphi_\theta(x^\mu, t) + \frac{z}{\sqrt{\Delta_t}}\right] \tag{7.3.26}$$

where $x^\mu = a^\mu e^{-t} + z\sqrt{\Delta_t}$ is the input noised via OU process.

Unfortunately there's not much more we can do with this, we must now put it on the computer and see what comes out.

**More General Time + Variance**

Instead of just using OU process, what if we noise the image according to

$$\frac{dx_i}{dt} = f_i(t)x_i + g_i(t)\eta_i(t) \tag{7.3.27}$$

where $x = (x_1, ..., x_d)$ and boundary condition $x(t=0) = a_i \sim p_0(a)$. This yields a PDE on the probability $\mathrm{Law}(x(t)) \sim p_t$, this is

$$p_t(x) = p_0(a)\frac{e^{-(x-as(t))^2/2s(t)^2\sigma(t)^2}}{\sqrt{2\pi s(t)\sigma(t)}} \tag{7.3.28}$$

$$\implies \varphi(x,t) = \nabla \log p_t(x) = -\frac{x}{s(t)^2\sigma(t)^2} + \frac{s(t)}{s(t)^2\sigma(t)^2}\langle a\rangle_{x,t} \tag{7.3.29}$$

## 7.4 A First Example: Gaussian Data

Consider your data's distribution is $p_0 = \mathcal{N}(m, \Sigma)$. We'll notate $\sum_{ij} a_i\Sigma_{ij}a_j = \langle a|\Sigma|a\rangle$ using bra-ket notation. So this means our noised distribution is

$$p(a,x,t) = \frac{e^{-\frac{1}{2}\langle a-m|\Sigma^{-1}|a-m\rangle}}{\sqrt{2\pi}^d\sqrt{\det\Sigma}}\frac{e^{-\frac{1}{2\Delta_t}(x-ae^{-t})^2}}{\sqrt{2\pi\Delta_t}^d} \tag{7.4.1}$$

So our marginalized distribution on $p_t(x)$ becomes

$$p_t(x) = \int da\ p(a,x,t) = \mathcal{N}(me^{-t}, \Gamma_t) \tag{7.4.2}$$

where $\Gamma_t = \Delta_t\mathbb{I}_d + e^{-2t}\Sigma$.

**Perfect Score**

$$\nabla \log p_t = -(\Gamma_t^{-1})(x - me^{-t}) \tag{7.4.3}$$

In the case when $t \gg 1$: you recover $\nabla \log p_t = -x$ (so it's a Gaussian!). In $t \ll 1$: $\nabla \log p_t \sim (\Sigma ...)$

### Optimized Score

Now your objective is to construct a parameterized score $\varphi_\theta$ from data $\{a^\mu\}_\mu \sim_{iid} p_0$. If you're smart you know that the answer must be linear, so let's ansatz our parameterized score as

$$\varphi_\theta(x,t) = -Wx - b \tag{7.4.4}$$

Now we can argmin the loss. However, we'll assume the case when $m = 0$ and $b = 0$.

$$\mathcal{L}(t) = \mathbb{E}_{a\sim p_0^{emp}} \mathbb{E}_{x\sim\mathcal{N}(ae^{-t},\Delta_t \mathbb{I})} |\varphi_\theta(x,t) = \frac{x - ae^{-t}}{\Delta_t}|^2 \tag{7.4.5}$$

where $x = a^\mu e^{-t} + \sqrt{\Delta_t} z$ is the noised initial data according to OU. Plugging in our ansatz (7.4.4), we get

$$|\varphi_\theta(x,t) + \frac{x - a^\mu e^{-t}}{\Delta_t}|^2 = |-W(a^\mu e^{-t} + \sqrt{\Delta_t} z) + \frac{z}{\sqrt{\Delta_t}}|^2 \tag{7.4.6}$$

$$L^t = \mathbb{E}_z[|\varphi_\theta(x,t) + \frac{x - a^\mu e^{-t}}{\Delta_t}|^2] = \frac{1}{\Delta_t}\mathrm{Tr}[(\mathbb{I} - \Delta_t W)^T(\mathbb{I} - \Delta_t W)] + \frac{e^{-2t}}{\Delta_t}\langle a^\mu | W^T W | a^\mu\rangle \tag{7.4.7}$$

We note that $\langle a^\mu | W^T W | a^\mu \rangle = \mathrm{Tr}(W^T W C^e)$ where $C^e_{ij} = \frac{1}{n}\sum_\mu a^\mu_i a^\mu_j = \frac{1}{n} a a^T$ is the empirical correlation function. Finally let's optimize over $W$ to find the minimum of the loss

$$\frac{\partial L^t}{\partial W} = 0 \implies W_{opt} = [(1 - e^{-2t})\mathbb{I} + e^{-2t} C^e]^{-1} \tag{7.4.8}$$

So in the limit as your amount of data $n \to \infty$, you find that $C^e \to \Sigma$ and $W_{opt} \to W_{perfect}$.

We can attempt to generalize some of these results using Random Matrix Theory.

Say for example the covariance matrix $\Sigma$ gets some Gaussian Noise. Meaning I observe an empirical noisy covariance $H$

$$H = \Sigma + \frac{1}{d^{a/2}} G \tag{7.4.9}$$

where $G \sim \mathrm{GOE}(d)$. There are known results on this (since $\Sigma \approx aa^T$ is a rank-1 perturbation).

- When $a > 1$: Eigenvalues of eigenvectors of $H$ are on the order of eigenvalues/vectors of $\Sigma$. Only in this case will your reconstruct the original distribution.
- If $a \in (0, 1)$, the eigenvalues of $H$ are on the order of $\Sigma$, however the eigenvectors differ in directions.

The lesson to be learned is that, what you find for Isotropic Gaussian, is not same for high-covariance Gaussian.

## 7.5 Related Approaches i.e. ODE, Stochastic Localization, & Interpolants

### 7.5.1 Ordinary Differential Equations

Consider the stochastic differential equation

$$\dot{x} = F(x) + \eta(t) \tag{7.5.1}$$

where $\langle \eta \rangle = 0$ and $\langle \eta(t)\eta(t') \rangle = 2\delta(t-t')$. It has an associated time evolution on $\text{Law}(x_t) = p_t$

$$\frac{\partial p}{\partial t} = \Delta p - \nabla(Fp) = \nabla(Fp - \nabla p) \tag{7.5.2}$$

So if we introduce the function $g(x) = F(x,t) - \varphi(x,t)$ (where $\varphi = \nabla_x \log p_t(x)$ denotes the score). Then you get the **continuity equation** (Mezard calls this a "flow equation").

$$\frac{\partial p}{\partial t} + \nabla(g(x,t)p) = 0 \tag{7.5.3}$$

Using this transformation, we now see our forward stochastic differential equation is

$$\dot{x} = g(x,t) \tag{7.5.4}$$

> There's an interesting paper (https://arxiv.org/abs/2006.00702) which shows how to estimate the solution to the Fokker Planck equation by using a mean-field limit of evolving many particles. Could be cool.

Anyways, what is the flow for the backwards process

$$\frac{dx}{d\tau} = F^B(x,\tau) = \eta(\tau) \tag{7.5.5}$$

$$F^B(x,\tau) = 2\varphi(x,\tau) \tag{7.5.6}$$

$$\frac{dp^B}{d\tau} + \nabla(p^B(F^B(x,\tau) - \nabla \log p^B)) = 0 \tag{7.5.7}$$

$$\implies \dot{x} = x + \varphi(x,t) \tag{7.5.8}$$

> So you might be tempted to say we should do this over diffusion, as you don't have to sampling random numbers in the dynamics... However it is (said in the literature to be) harder to sample multimodal distributions.

### 7.5.2 Example with Isotropic Gaussian

Consider the target distribution

$$p_0(a) = \frac{e^{-|a|^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}} \tag{7.5.9}$$

The interpolated distribution (via OU process) is

$$p_t(x) = \frac{e^{|x|^2/2\Gamma_t}}{\sqrt{2\pi\Gamma_t}} \implies \varphi(x,t) = \nabla_x \log p_t(x) = -x/\Gamma_t \tag{7.5.10}$$

And our ODE is

$$\dot{x} = \frac{x}{1 + (\sigma^2 - 1)e^{-2(t_g - \tau)}} \tag{7.5.11}$$

### 7.5.3 Stochastic Interpolants

For the original method, see Albergo & Vanden-Eijnden. What they realized is that we start with some inital distribution $p_0$ and we map it to a final gaussian $p_{final} \propto e^{-x^2/2}$.

At each time $t$ , you generate $x(t) = \alpha(t)a + \beta(t)z$ (where $z \sim \mathcal{N}(0, \mathbb{I})$). And since you want to recover $x_{t=0} \sim p_0$ and $x_{t=t_f} \sim p_{final}$ we assert boundary conditions $\alpha(0) = 1, \beta(0) = 0$ and $\alpha(t_f) = 0, \beta(t_f) = 1$, which asserts $x(0) = a$ and $x(t_f) = z$. We denote $\alpha, \beta$ as the **interpolant**.

So what you should have noticed is that we don't assert any Langevin. We just assert our interpolation scheme. This allows for flexibility in the engineering process! If we assert $\mathrm{Law}(x_t) = p_t(x)$, we can once again derive a time evolution on the probability distribution

$$p_t(x|a) = \frac{e^{-\frac{1}{2\beta(t)^2}|x-\alpha(t)a|^2}}{\sqrt{2\pi\beta(t)^2}^d} \tag{7.5.12}$$

$$p(a|x,t) = \frac{1}{Z}p_0(a)e^{-\frac{1}{2\beta(t)^2}|x-\alpha(t)a|^2} \tag{7.5.13}$$

And this probability distribution satisfies

$$\frac{\partial p_t}{\partial t} + \nabla(b(x,t)p_t(x)) = 0 \tag{7.5.14}$$

where $b(x,t) = \mathbb{E}_{a,z}[\dot{\alpha}a + \dot{\beta}z|\alpha a + \beta z = x]$.

Let's prove this!

$$p_t(x) = \int da\ p_0(a) \int \mathcal{D}z\delta(x - (\alpha(t)a + \beta(t)z)) \tag{7.5.15}$$

Plugging this into the continuity equation

$$\frac{\partial p_t(x)}{\partial t} = -\int da\ p_0(a) \int \mathcal{D}z\ [\dot{\alpha}(t)a + \dot{\beta}(t)z]\dot{\nabla}\delta(x - (\alpha(t)a + \beta(t)z)) \tag{7.5.16}$$

$$= -\nabla \cdot \left[p(x,t) \int da\ p_0(a) \int \mathcal{D}z[\dot{\alpha}a + \dot{\beta}z]\frac{\delta(x - (\alpha a + \beta z))}{p(x,t)}\right] \tag{7.5.17}$$

So you'll notice, $\int da\ p_0(a) \int \mathcal{D}z\ [\dot{\alpha}a + \dot{\beta}z]\frac{\delta(x-(\alpha a+\beta z)}{p(x,t)} = \mathbb{E}_{a,z}[\dot{\alpha}a + \dot{\beta}z|\alpha a + \beta z = x]$, which is $b(x,t)$.

QED.

Why bring this up? Well diffusion is just a subset of potential stochastic iunterpolants where

$$z = \frac{x - \alpha a}{\beta}, b(x,t) = \frac{e^{-2t}}{\Delta_t x - \frac{e^{-t}}{\Delta_t}}\langle a\rangle_x \tag{7.5.18}$$

### 7.5.4 Unifying Framework for Langevin, ODE, and Stochastic Interpolants

Starting from the continuity equation for stochastic interpolants (7.5.14). If we ansatz $b$

$$b^F_\lambda(x,t) = b(x,t) + \lambda\varphi(x,t) \tag{7.5.19}$$

Then the continuity equation becomes

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (b_\lambda^F(x,t)p_t) = \lambda \nabla \cdot (\varphi(x,t)p_t) = \lambda \Delta p_t \tag{7.5.20}$$

You now have a Fokker Planck equation which emits a SDE interpretation

$$\frac{dx}{dt} = b_\lambda^F(x,t) + \sqrt{\lambda}\eta(t) \tag{7.5.21}$$

If we look at a couple cases

- $\lambda = 1$ we recover our usual Langevin
- $\lambda = 0$ we recover ODE (flow based methods)

### 7.5.5 Stochastic Localization

For the original, see R. Eldan (2013, 2020, 2022); El Aloni Montanar Selke (2022); Montanari (2023).

Montanari's "interpolant" is of the form

$$x(\tau) = a\tau + \sqrt{\tau}z \tag{7.5.22}$$

where $z \sim \mathcal{N}(0, \mathbb{I})$. You now find that

$$p_\tau(x|a) = \frac{1}{\sqrt{2\pi\tau}^d} e^{-\frac{\tau}{2}(\frac{x}{\tau}-a)^2} \tag{7.5.23}$$

When I look at the limit $\tau \to \infty$, you get $\frac{x}{\tau} \to a$. You can think of this as performing a random walk where you converge to $a$, but your variance explodes.

Looking at the conservation equation

$$\frac{\partial p_\tau(x)}{\partial \tau} + \nabla \cdot (bp) = 0 \tag{7.5.24}$$

$$\text{s.t. } b(x,\tau) = \mathbb{E}[a + \frac{1}{2\sqrt{\tau}}z | a\tau + \sqrt{\tau}z + x] = \frac{x}{2\tau} + \frac{1}{2}\langle a \rangle_{x,\tau} \tag{7.5.25}$$

And once again using Fokker Planck, you get the SDE

$$\dot{x} = \langle a \rangle_{x,t} + \eta(\tau) \tag{7.5.26}$$

**Example on EBM**

Consider the exponential family distribution $p(a) = e^{-\beta E_0(a)}/Z$ where $\sum_{i=1}^d a_i^2 = d$ and $E_0(a) = -\sum_{i<j<k} J_{ijk} a_i a_j a_k$ (which is a $p = 3$ spin glass).

We know as a function of temperature, we know there's a low temperature spin glass phase ($\beta > \beta_k$), high temperature phase ($\beta < \beta_d$), and there's an intermediate phase $\beta \in [\beta_d, \beta_k]$. In the glassy & intermediate phase, MCMC / Langevin don't work (because of critical slowing down).

$$p(a|x) = e^{\beta \sum J_{ijk} a_i a_j a_k - \gamma \sum_i (a_i - x_i)^2} \tag{7.5.27}$$

Let's inspect the phase diagram as we tune the pinning field $\gamma(\tau)$ (s.t. $\gamma(\tau = 0) = 0$ and $\gamma(\tau = \infty) = \infty$).

What this suggests is that below $T_{crit}$, diffusion cannot sample well, while BP still can. (What Mezard means by this is that we can use BP to compute marginals / expectation values without sampling. This is NOT contradictory to UNets can realized BP algorithms because that paper talks about using BP for sampling).

# Chapter 8

# Arvind Murugan: Learning without Neurons

## 8.1 Overview

Since we're talking about learning without neurons, perhaps we should talk about what we mean about making computation. Murugan states there are two types of computation

1. Symbolic: These are analytic expressions which a compute can crunch / a human can set

2. Behavior: Living organism / smart materials which respond and interact with their environment.

The talk will focus on the latter. However computations are used to solve problems, so let's also define some ways of solving problems. Let's work through an example of heating and cooling a room with an AC.

1. Abstraction First:

   An example of this is PI control. So you set up a control flow.

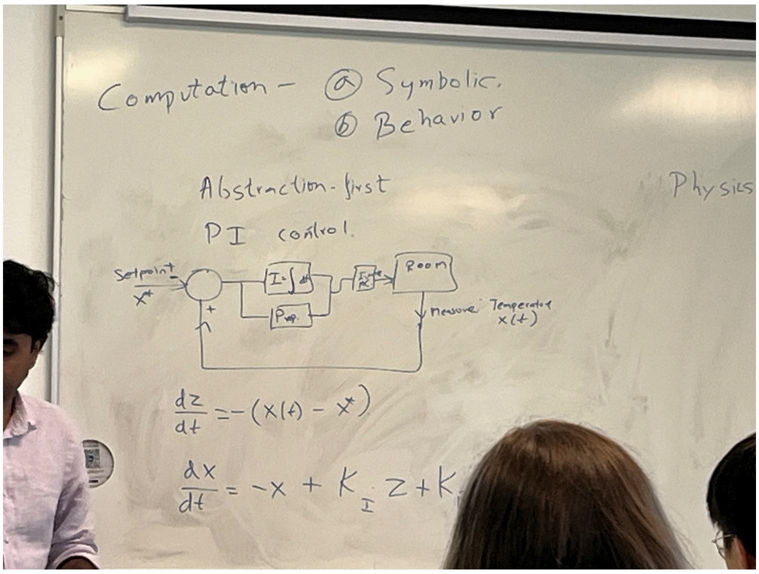
Computation – ⓐ Symbolic.
ⓑ Behavior
Abstraction-first
Physics
PI control.
Setpoint
$x^*$
+
$I = \int dt$
P
Room
measure
Temperature $x(t)$
$\frac{dz}{dt} = -(x(t) - x^*)$
$\frac{dx}{dt} = -x + K_I z + K$

Figure 8.1: Diagram of control line

   This defines a differentiation equation which defines your control line $z$ due to observation of a process $x$

- Pros:
- General / modular
- Cons
- Cannot exploit specific physics
- The translation between theory vs implementation is non-trivial.

2. Physics First:

   You can set up a thermodynamic state equation, and solve for the phase diagram. Pros and Cons

   - Pros:
   - Fewer parts (measurement, computation), which means its more robust
   - It's passive, meaning it self computes
   - Cons
   - Specific to the physics

This talk will focus on how to move toward the 2nd way of thinking. An example of this is the Liquid-Liquid phase transition of two liquids inside of a cell (see Zechnor 2020). You can think of this as oil and water separating.

There are two communities working on this

1. Energy efficient computations (vs silicon).

   - Example. We currently run computations on a silicon chip, however this is not energy efficient, so much so governments are thinking about building more nuclear reactors to power the next generation of AI). So perhaps we can build biological circuits which run these computations natively on some bio-efficient chip.
   - Example. Encode your optimization problem onto a Ising-like physical system, and anneal that system! Now you can read off optimized parameters.
   - Example. You can use optimal computing to make better matrix multiplication.

2. Biological computations

   - Backprop. That is each organism is reacting and adjusting it's behavior. Much more complex than just physics or chemical behavior.
   - Evolution. No one organism does the computation, but rather it's the ensemble of them which can perform computations.
   - Physics / Chemistry. It's just a reaction based on laws of physics / chemistry.

Now, how does this all relate to machine learning. Let's consider a high dimensional classifier– you could think of the decision boundary as phase boundaries. You can also go one step further and use the transition between phases as the classifier (see https://www.jbc.org/article/S0021-9258(20)36794-6/pdf for more details)! So your features correspond to temperature, pressure, etc., and your classification is the phase. So the questions become what in the material / organism do I have to tune s.t. you can learn, and this also may find new interesting physical mechanisms for scientists to play with.

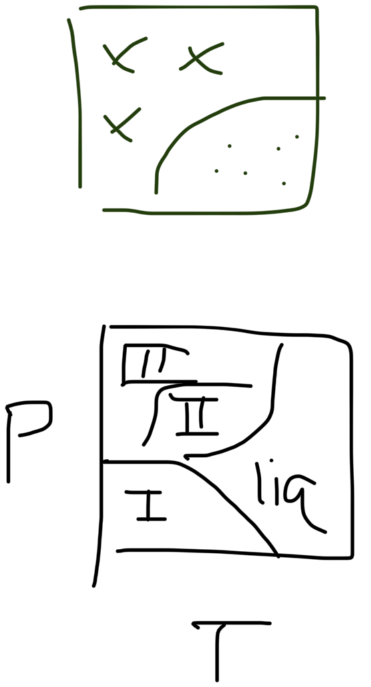

Figure 8.2: Decision boundary problems can be mapped into a physics system by inspecting their phase transition

### 8.1.1 Training Molecules

**Hopefield Associative Memory**

Consider a set of neurons $\{x_i\}_{i=1}^N$ which take on binary values $x_i \in \{\pm 1\}$. It takes on discrete dyanmics

$$x_i(t+1) = \text{sign}\left(\sum_j J_{ij} x_j(t) + h_i^{ext}(t)\right) \tag{8.1.1}$$

Your task is a supervised learning problem... The goal is to store a set of memories $m_i^\alpha \in \mathbb{R}^N$ (s.t. $\alpha = 1, ..., M$). And you want to retrieve a memory $m$ from a just showing a partial part of $m$ (association). That is you want your neurons $\{x_i\}$ to fire in the same pattern as $m$

To train this model, we use a **Hebbian** learning rule

$$\frac{dJ_{ij}}{dt} = x_i(t) x_j(t) \tag{8.1.2}$$

We run this every time we should it a new memory. So for example, after being exposed to 3 memories $\{m^{(1)}, m^{(2)}, m^{(3)}\}$, our couplings look like $J_{ij} = m_i^{(1)} m_j^{(1)} + m_i^{(2)} m_j^{(2)} + m_i^{(3)} m_j^{(3)}$. Running this process will make the energy landscape become low where $x = m^{(i)}$– so if you train on images, and then show noised versions of those images, then you'll be-able to denoise said images! Very cool!

Note that hopfield networks have a memory capacity $M_c$, so this only works when the total memories $M < M_c$.

**Multifavious Assembly Mixtures**

Here we'll consider a physical system which is analogous to Hopfield's Assocative Memory model.

Consider $N$ molecular species in a solvent (this is different from the number of molecules, for example these can be $N$ proteins with different names), with binding interactions $J_{ij}$, and concentrations $C_i(x,t)$ at position $x$ at time $t$ (and associated chemical potentials $\mu_i$). The molecules are held at a physical temperature $T$, and have diffusion constant $D_i$.

$$\frac{dJ_{ij}}{dt} = -\int d^3x \; dt \; C_i(x,t)C_j(x,t) \tag{8.1.3}$$

This rule says, these molecules will increase their bond ($J_{ij}$ will increase) if there are more of said molecules in the same place ($C_i(x,t)C_j(x,t)$ is the concentrations of molecules $i$ and $j$ at position $x$ at time $t$).

In practice, we need to use **linkers** (these are other molecules which mediate interactions between molecules). So they can be used to tune your $J_{ij}$'s. You can think of this has having a large interaction matirx $J$ (say 100 molecules), but you only observe a fraction of them yielding a tunable effective interaction matrix $\tilde{J}$ (say you actually only observe 10 molecules).

The associative memory is to reconstruct a set of protein binded together. So you can construct a memory as a sequence of proteins, cut it up, place it into the test tube, and see if it can reconstruct the previous protein.

## 8.2 Expressitivity and Trainability

Now that we've covered a one to one mapping between a machine learning method and a biological system which has a similar.

### 8.2.1 Training of Molecular Networks

We can train molecular networks using the sleep-wake algorithm. At a high-level, the algorithm is similar to the expectaiton-maximization algorithm. In this case, you have two distributions defined by a Spin-Glass like distribution, one is trainable $Q_w$ and the other is target distribution $P$, you update your couplings $w$ according to the difference of two point functions.

$$\frac{\partial D_{KL}(Q_w||P)}{\partial w_{ij}} = \underbrace{\langle s_i s_j \rangle_P}_{\text{sleep}} - \underbrace{\langle s_i s_j \rangle_Q}_{\text{wake}} \tag{8.2.1}$$

### 8.2.2 Potts Model of Molecular Networks

So, perhaps to make things more concrete, let's try to create a model of a molecular network. The molecules live in a test tube, so all interactions must occur at some real space $x \in \Lambda \subset \mathbb{R}^3$. Consider the Potts Model. Configurations are which specifies of molecules $[N] = \{1, ..., N\}$ are where, so the configurations are $\sigma : \Lambda \to [N]$.

$$E = \sum_{x\in\Lambda} \sum_{x_n \in \partial x} J_{\sigma(x),\sigma(x_n)} + \sum_i \mu_i n_i(\sigma) \tag{8.2.2}$$

Where $J_{ij}$ is an interaction matrix between species (so $i = 1, ..., N$ number of species), $\sum_{x_n\in\partial x}$ is a sum over the spatial neighbors of $x$, $\mu_i = -\log c_i$ is the chemical potential of species $i$ (which is the negative log of the concentration, and $n_i$ is the number of species at $\sigma$.

### 8.2.3 Place Models

Consider an Ising model of $N$ spins on a lattice. The place model now does all possible permutations of the spins with the same lattice structure.

## 8.3 June 11th

# Chapter 9

# Etc.

## 9.1 People Presentations

1. Paolo Balioni, working on feature learning in Bayesian Neural Networks. Also works on MCMC methods.
2. Claudian Merger. Mappping normalizing flows to $\phi^n$ theory.

## 9.2 Q&A

**Question:** What is the manifold hypothesis? Could you give me a test for computing the effective dimension of such a manifold? (@Marc Mezard)

**Answer:** It is not a hypothesis, it's a fact. We observe that data often has an effective dimension $d_{eff}$ which is significantly lower than the original data. An algorithm which can get you $d_{eff}$, is your perform KNN on your data, and see $k$ which has ”decent” enough performance. This algorithm obviously abstracts away manifold properties,

## 9.3 Acknowledgements

Thank you to Prof. David Hogg & the Flatiron Institute for providing funding for this trip!