Lectures Notes from Beg Rohu 2025

Clark Miyamoto (cm6627@nyu.edu)

June 4, 2025

Abstract

A set of lecture note from the Beg Rohu Summer School in 2025. If any of the other participants would like to collaborate on these notes, please reach out to me!

Contents

I Yann LeCun: Self Supervised Models **3**

1 Overview: A Path to Advance Machine Intelligence **3**
1.1 What is an Energy Based Model (EBM) . . . 3
1.1.1 Training EBMs . . . 4
1.1.2 Avoiding Collapse in Training . . . 4
1.2 What is a World Model . . . 4

II Stéphane Mallat: Score Based Diffusion & Renormalization Group Flow **5**

2 Overview **5**
2.1 Transport . . . 5
2.2 Outline . . . 6

3 Historical Models: Energy Based Models, GANs, Normalization Flows **6**
3.1 Energy Based Models (EBM) . . . 6
3.1.1 Likelihood . . . 7
3.2 GANs . . . 7
3.2.1 Conditional GANs . . . 8
3.3 Normalizing Flows . . . 8
3.3.1 Continuous Flow . . . 8
3.4 Score Based Diffusion . . . 9
3.4.1 Orstein Ulhrbeck SDE . . . 9
3.4.2 Denoising . . . 10
3.4.3 Denoising OU . . . 10
3.4.4 Example of Denoising Gaussian . . . 11
3.5 Generalization vs Memorization? . . . 11
3.6 Architecture of CNNs . . . 11
3.6.1 Convolutional Operator . . . 11
3.6.2 Explaining Bias in the Model . . . 12

III Arvind Murugan: Learning without Neurons **14**

4 Overview **14**
4.1 Training Molecules 15
4.1.1 Hopefield Associative Memory 15
4.1.2 Multifavious Assembly Mixtures 16

5 Expressitivity and Trainability **17**
5.1 Training of Molecular Networks 17
5.2 Potts Model of Molecular Networks 17
5.3 Place Models 17

IV Etc. **18**

6 People Presentations **18**

7 Acknowledgements **18**

Part I
Yann LeCun: Self Supervised Models

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.

1.1 What is an Energy Based Model (EBM)

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

$$\hat{y} = (f_1 \circ f_2 \circ ... \circ f_n)(x) \tag{1.1}$$

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**.

$$\hat{y} = \text{argmin}_{y \in \mathcal{Y}} f(x, y) \tag{1.2}$$

Interestingly, you can also have multiple answers as well. For training such models, you make it learn the f_θ (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
>
> $$p(y|x) = \frac{e^{-\beta f(x,y)}}{Z} \tag{1.3}$$

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.

1.1.1 Training EBMs

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

A naive method is to just perform gradient descent

$$E(y) = -\sum_{ij} w_{ij} y_i y_j \tag{1.4}$$

$$\frac{\partial E}{\partial w_{ij}} = -y_i y_j \tag{1.5}$$

$$\therefore w_{ij} \leftarrow w_{ij} + \epsilon(y_i y_j) \qquad \text{Backprop Step} \tag{1.6}$$

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

$$w_{ij} \leftarrow w_{ij} + \epsilon(y_i y_j - \hat{y}_i \hat{y}_j) \tag{1.7}$$

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...

1.1.2 Avoiding Collapse in Training

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 $W : \mathcal{S} \times$ Actions $\to \mathcal{S}$ and attempts to predict what the next state of the world is. So an example of inference is

$$x_{t+1} = E^{-1}(W(x_t, q) \tag{1.8}$$

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.

Part II
Stéphane Mallat: Score Based Diffusion & Renormalization Group Flow

2 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 / ϕ^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 1 (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

$$p(x) = \prod_k p_k(x) \implies \log p(x) = \sum_k \log p_k(x) \tag{2.1}$$

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).

2.1 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

$$p(x) = \underbrace{p(x_T)}_{\text{Gaussian}} \prod_{t=T+1}^{1} p(x_t/x_{t-1}) \tag{2.2}$$

2.2 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.

3 Historical Models: Energy Based Models, GANs, Normalization Flows

3.1 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

$$p_\theta(x) = \frac{e^{-U_\theta(x)}}{Z_\theta}, \quad \text{s.t. } Z_\theta = \int dx\ e^{-U_\theta(x)} \tag{3.1}$$

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

$$U_\theta(x) = x^T \Sigma_\theta^{-1} x + \theta \sum_i V(x_i) \tag{3.2}$$

In physics the first one is the kinetic energy term, and the second is the potential energy term. You could start by making U_θ a neural network.

Example 2 *Another example model is the exponential family*

$$U_\theta(x) = \langle \theta, \Phi(x) \rangle \tag{3.3}$$

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

3.1.1 Likelihood

Definition 2 (Likelihood) *The negative likelihood $\ell(\theta)$ of probability distribution p*

$$\ell(\theta) = -\mathbb{E}_{x\sim p}\log p_\theta(x) = KL(p||p_\theta) + H[p] \tag{3.4}$$

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

Our objective will be choose θ s.t. we minimize the negative likelihood

$$\theta^* = argmin_\theta\ \ell(\theta) \tag{3.5}$$

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

$$\begin{aligned}\nabla_\theta \ell(\theta) &= \mathbb{E}_{x\sim p}[\nabla_\theta U(x)] + \log Z_\theta &(3.6)\\ &= \mathbb{E}_{x\sim p}[\nabla_\theta U_\theta(x)] - \mathbb{E}_{x\sim p_\theta}[\nabla_\theta U_\theta(x)] &(3.7)\end{aligned}$$

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.

3.2 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_{θ_G} and D_{θ_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

$$D_{ideal}(x) = \begin{cases} 1 & \text{if } x \text{ is a real image} \\ 0 & \text{if } x \text{ is a fake image} \end{cases} \tag{3.8}$$

The loss function is

$$\mathcal{L}(\theta_D, \theta_G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim \text{Gaussian}}[\log(1 - D(G(z))] \tag{3.9}$$

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*

$$\begin{aligned}D^* &= argmax_D\ \mathcal{L}(D, G) &(3.10)\\ &= \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} &(3.11)\end{aligned}$$

Theorem 1

$$\min_G \max_D \mathcal{L}(D, G) \iff p_G = p_{data} \tag{3.12}$$

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

3.2.1 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

$$\mathcal{L}(D,G) = \mathbb{E}_{p_{data}}[\log D(x|y)] + \mathbb{E}_{z\sim p_{data}}[\log(1 - D(G(x|y)))] \tag{3.13}$$

3.3 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_θ, and now we can do change of basis between the two

$$p_x(x) = p_z(T(x)) \det \mathcal{J}_T(x) \tag{3.14}$$

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

The likelihood ends up being

$$\ell(\theta) = -\mathbb{E}_{x\sim p}[\log p_z(T_\theta(x))] - \mathbb{E}_{x\sim p}[\log|\det -\mathcal{J}_{T_\theta}|] \tag{3.15}$$

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 $x \in \mathbb{R}^d = (x_0, x_1) \in \mathbb{R}^{d_0} \times \mathbb{R}^{d_1}$

$$T_\theta(x) = (x_0, e^{S_\theta(x_0)} \odot x_1 + t_\theta(x_0)) \tag{3.16}$$

$$T_\theta^{-1}(z) = (z_0, (z_1 - t_\theta(z_0)) \odot e^{-S_\theta(z_0)} \tag{3.17}$$

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

$$\implies \log|\det \mathcal{J}_{T_\theta}(x)| = \sum_{i=1}^{d} s_\theta(x_0) \tag{3.18}$$

Finally we have our likelihood

$$\ell(\theta) = -\mathbb{E}_{x\sim p}[\log p_z(T_\theta(x))] - \sum_{i=1}^{d} \mathbb{E}_{x\sim p}[s_\theta(x_0)] \tag{3.19}$$

3.3.1 Continuous Flow

Consider the **Neural ODE**

$$\frac{dx_t}{dt} = v_\theta(x_t, t) \qquad \text{(Continuous)} \tag{3.20}$$

$$x_{t+\epsilon} = x_t + \epsilon\, v_\theta(x_t, t) \qquad \text{(Naive Discretization)} \tag{3.21}$$

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**

$$\frac{\partial p_t}{\partial t} = -\nabla \cdot (v_t \; p_t) \tag{3.22}$$

$$= - \underbrace{v_t \cdot \nabla p_t}_{\text{Convection}} - \underbrace{p_t(\nabla \cdot v_t)}_{\text{compression / dilation}} \tag{3.23}$$

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!

3.4 Score Based Diffusion

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

$$\frac{\partial p_t}{\partial t} = -\nabla \cdot (p_t(x)v_t) - \Delta \left(\frac{1}{2}\sigma^2(x)p_t(x) \right) \iff dx_t = v_t(x_t)dt + \sigma dW_t \tag{3.24}$$

where Δ 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!

3.4.1 Orstein Ulhrbeck SDE

Consider the stochastic differential equation

$$dx_t = -\kappa x_t dt + \sigma dW_t \tag{3.25}$$

To solve it, perform the change of variable

$$y_t = e^{\kappa t} x_t \tag{3.26}$$

$$dy_t = e^{\kappa t}(-\kappa x_t dt + \sigma dW_t) + \kappa e^{\kappa t} x_t dt = e^{\kappa t}\sigma dW_t \tag{3.27}$$

This yields

$$x_t = e^{-\kappa t}x_0 + \sigma\sqrt{\frac{1 - e^{-2\kappa t}}{2\kappa}}z \tag{3.28}$$

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
>
> $$p_t = (e^{\kappa t}p(\cdot e^{\kappa t})) * g_\sigma \tag{3.29}$$
>
> 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

3.4.2 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 (3.24), and derive a time evolution on the distribution for the reverse process

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (-p_t x_t) - \Delta p_t = 0 \qquad \text{Forward in time} \tag{3.30}$$

$$\implies \frac{\partial q_t}{\partial t} + \nabla \cdot (q_t y_t) + \Delta q_t = 0 \qquad \text{Backwards in time} \tag{3.31}$$

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

$$\Delta q_t = \nabla \cdot (q_t \underbrace{\nabla \log q_t}_{\text{score}}) \tag{3.32}$$

We finally get

$$\frac{\partial q_t}{\partial t} + \nabla \cdot (q_t y_t - (1+\alpha) q_t \nabla \log p_t) - \alpha \Delta q_t = 0 \tag{3.33}$$

$$dy_t = [y_t + (1+\alpha)\nabla \log q_t] + \sqrt{2\alpha}\, dW_t \tag{3.34}$$

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

3.4.3 Denoising OU

We have the $\kappa = 1$ OU,

$$x_t = e^{-t}x + \sqrt{1 - e^{-2t}}z \tag{3.35}$$

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

$$y_t = x + \sqrt{e^{2t} - 1}z \tag{3.36}$$

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

$$\mathbb{E}_{x,z}[||\hat{x}(y_t) - x||^2] \tag{3.37}$$

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 2 *Finding the minimum of*

$$\mathbb{E}_{x,z}[||\hat{x}(y_t) - x||^2] \tag{3.38}$$

Ends up being

$$\hat{x}(x_\sigma) = \mathbb{E}[x|x_\sigma] = x_\sigma + \sigma^2 \nabla_{x_\sigma} \log p_\sigma(x_\sigma) \tag{3.39}$$

Proof:

$$p_\sigma(x_\sigma) = \int p(x_\sigma|x)p(x)dx \tag{3.40}$$

$$\nabla p_\sigma(x_\sigma) = \int g_\sigma(x_\sigma - x)p(x)dx \qquad \textit{Due to solution of OU} \tag{3.41}$$

$$= \frac{1}{\sigma^2}\int (x_\sigma - x)g_\sigma(x_\sigma - x)p(x)dx \tag{3.42}$$

$$\frac{\sigma^2\nabla p_\sigma(x_\sigma)}{p_\sigma(x_\sigma)} = \int (-x_\sigma + x)p_\sigma(x|x_\sigma)dx \qquad p(x)/p_\sigma(x_\sigma) = p_\sigma(x|x_\sigma) \tag{3.43}$$

$$= -x_\sigma + \hat{x}(x_\sigma) \tag{3.44}$$

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 $\frac{\hat{x}_\theta(x) - x}{\sigma^2} = \nabla_{x_\sigma} \log p_\sigma(x_\sigma)$

3.4.4 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_σ 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

$$\hat{x}(\sigma) = [\mathbb{I} - (C + \sigma^2\mathbb{I})^{-1}\sigma]x_\sigma \tag{3.45}$$

$$= (C + \sigma^2\mathbb{I})^{-1}(C + \sigma^2\mathbb{I} - \sigma^2\mathbb{I})\hat{x})_\sigma \tag{3.46}$$

$$= (C + \sigma^2\mathbb{I})^{-1}Cx_\sigma \tag{3.47}$$

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

3.5 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

3.6 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.

3.6.1 Convolutional Operator

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

$$x_{\ell+1}(i, c') = \rho\left(\sum_{c\in\text{Channels}}\sum_{\tau\in\{1,2\}} W(\tau, c, c')x_\ell(i - \tau, c)]\right) + b \tag{3.48}$$

where ρ 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 ρ 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.

3.6.2 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

$$\hat{x} = x_\sigma + \sigma^2 \underbrace{\nabla \log \tilde{p}_\sigma(x_\sigma)}_{\equiv s_\sigma(x_\sigma)} \tag{3.49}$$

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

$$\hat{x} = \left(\mathbb{I} + \sigma^2 \mathcal{J}_{s_\sigma}(x_\sigma)\right) x_\sigma \tag{3.50}$$

we can decompose the interior in an eigenbasis.

$$\hat{x} = \sum_k h_k \langle x_\sigma, \psi_k \rangle \psi_k \tag{3.51}$$

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

$$||\hat{x} - x||^2 = \sum_k |\langle \hat{x}, \psi_k \rangle - \langle x, \psi_k \rangle|^2 \tag{3.52}$$

> 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).

hi

Part III
Arvind Murugan: Learning without Neurons

4 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.

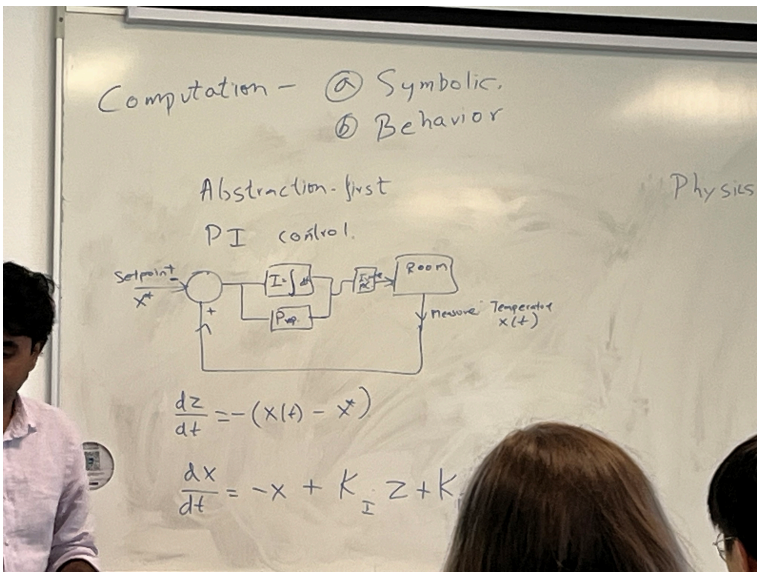

Figure 1: Diagram of control line

   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.

4.1 Training Molecules

4.1.1 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

$$x_i(t+1) = \text{sign}\left(\sum_j J_{ij} x_j(t) + h_i^{ext}(t)\right) \tag{4.1}$$

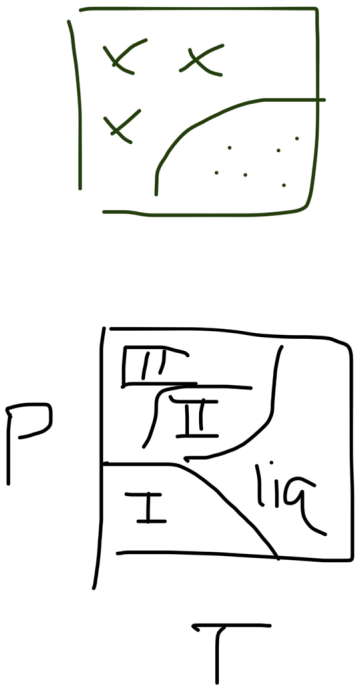

Figure 2: Decision boundary problems can be mapped into a physics system by inspecting their phase transition

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

$$\frac{dJ_{ij}}{dt} = x_i(t)x_j(t) \tag{4.2}$$

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 $M < M_c$.

4.1.2 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 μ_i). The molecules are held at a physical temperature T, and have diffusion constant D_i.

$$\frac{dJ_{ij}}{dt} = -\int d^3x\ dt\ C_i(x,t)C_j(x,t) \tag{4.3}$$

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.

5 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.

5.1 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.

$$\frac{\partial D_{KL}(Q_w||P)}{\partial w_{ij}} = \underbrace{\langle s_i s_j \rangle_P}_{\text{sleep}} - \underbrace{\langle s_i s_j \rangle_Q}_{\text{wake}} \tag{5.1}$$

5.2 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 $\sigma : \Lambda \to [N]$.

$$E = \sum_{x \in \Lambda} \sum_{x_n \in \partial x} J_{\sigma(x),\sigma(x_n)} + \sum_i \mu_i n_i(\sigma) \tag{5.2}$$

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 σ.

5.3 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.

Part IV
Etc.

6 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 ϕ^n theory.

7 Acknowledgements

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