Machine Learning for Physicists: Recitation Notes

Clark Miyamoto (cm6627@nyu.edu)

January 25, 2026

Contents

Part I
Mathematics

1 Review of Linear Algebra

Here's some linear algebra that you might not have learned in a regular class.

1.1 Singular Value Decomposition

Recall the eigen-decomposition of a matrix. Given a symmetric square matrix $A \in \mathbb{R}^{d\times d}$ with eigenvalues $\{\lambda_i\}_i$ and eigenvectors $\{e_i\}_i$. The matrix could be re-expressed as

$$A = U\Lambda U^T \tag{1.1}$$

where $\Lambda = \text{diag}(\lambda_1, ..., \lambda_d) \in \mathbb{R}^{d\times d}$ and $U \in \mathbb{R}^{d\times d}$ is a matrix whose columns are $\{e_i\}_i$.

This decomposition had a lot of nice properties. In particular, Λ is diagonal and U is orthogonal. This allowed us to do all sorts of stuff easily; for example, matrix power.

What happens if we want to do this on non-symmetric, or even non-square matrices? Well we can use the singular value decomposition (SVD).

1.1.1 Definitions

Definition 1 (Singular Values) *Let $A \in \mathbb{R}^{m\times n}$. Now consider $A^TA \in \mathbb{R}^{n\times n}$. This is a symmetric matrix so it has positive eigenvalues $0 \le \lambda_1 \le ... \le \lambda_n$. The singular values σ_i for matrix A are defined as*

$$\sigma_i \equiv \sqrt{\lambda_i},\ \ s.t.\ 0 \le \lambda_1 \le ... \le \lambda_n \tag{1.2}$$

Fact 1 *The number of non-zero singular values of A correspond to the rank of A.*

Proof: Let $A : \mathbb{R}^d \to \mathbb{R}^d$ be a linear map. Recall by Rank-Nullity theorem $\text{rank}(A) + \dim \text{Ker}(A) = \dim(\mathbb{R}^d)$. Recall $\text{Ker}(A) = \{v : A(v) = 0\}$, so the dimension of the kernel is the number of zero eigenvalues.

Also notice that $\text{Ker}(A) = \text{Ker}(A^TA)$. ($\implies$) Let $v \in \text{Ker}(A)$, then $A^TAv = 0$, therefore $v \in \text{Ker}(A^TA)$. ($\impliedby$) Let $v \in \text{Ker}(A^TA)$, then $A^TAv = 0$, meaning $x^TA^TAv = \|Av\|^2 = 0$, the vector norm is only zero when the vector is zero, therefore $Av = 0$, implying $v \in \text{Ker}(A)$.

This means the $\text{rank}(A) = \dim(\mathbb{R}^d) - \text{Ker}(A^TA)$. The dimension of the kernel of A^TA is the number of zero singular values of A.

□

Definition 2 (SVD) *$A \in \mathbb{R}^{m\times n}$ with singular values $0 \le \sigma_1 \le ... \le \sigma_n$. Let r denote the rank, or equivalently the number of singular values of A. The SVD of A is a decomposition*

$$A = U\Sigma V^T \tag{1.3}$$

where

- *$U \in \mathbb{R}^{m\times m}$ orthogonal matrix*
- *$V \in \mathbb{R}^{n\times n}$ orthogonal matrix*

- $\Sigma \in \mathbb{R}^{m \times n}$ *matrix such that* $[\Sigma]_{ii} = \sigma_i$ *for* $i \in [1, ..., r]$ *and* $[\Sigma]_{ii} = 0$ *for* $i > r$.

Theorem 1 (Computing SVD) *Let* $A \in \mathbb{R}^{m \times n}$. *Then* A *has a (non-unique) SVD* $A = U\Sigma V^T$, *where*

- *The columns of* V *are orthonormal eigenvectors of* $A^T A$, *where* $A^T A v_i = \sigma^2 v_i$.
- *If* $i \leq r$, *s.t.* $\sigma_i \neq 0$, *then the* i*'th column of* U *is given by* $\sigma_i^{-1} A v_i$

1.1.2 Illustration of SVD

Recall, by 3Blue1Brown, matrix operations transform the coordinate space of some vector. So the only hope we have at visualize the SVD is to use this intuition.

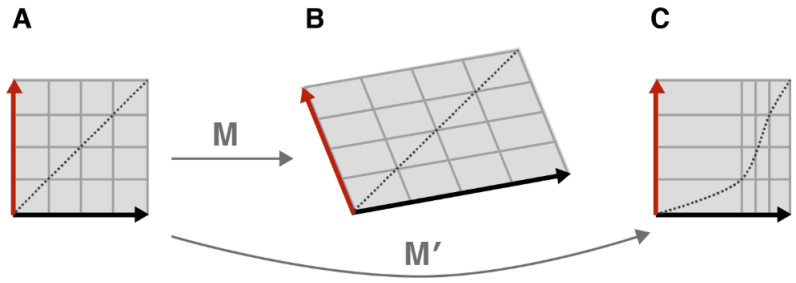

Figure 3: (A) Our original square under a linear transformation $\mathbf{M}$ (B) and a nonlinear transformation $\mathbf{M}'$ (C).

Figure 1: Visualization from [2]

Let $A \in \mathbb{R}^{2 \times 2}$. Let $v_1, v_2 \in \mathbb{R}^2$ be orthonormal vectors, that is $v_i \cdot v_j = \delta_{ij}$. Say we know how A acts on v_i, that is it rotates them to another orthonormal basis u_i and rescales them according to σ_i.

$$Av_1 = \sigma_1 u_1 \tag{1.4}$$
$$Av_2 = \sigma_2 u_2 \tag{1.5}$$

We've chosen the notations of these vectors in a very peculiar manner. You can think the SVD as just saying we map vectors in matrix V to vectors in U scaled by Σ.

But deriving is believing, so let's show that the matrix A emits an SVD where everything lines up.

Consider how A acts on an arbitrary test vector x.

$$\begin{aligned}
Mx &= M(\langle v_1, x\rangle v_1 + \langle v_2, x\rangle v_2) && \text{Basis} && (1.6)\\
&= Mv_1 \langle v_1, x\rangle + Mv_2 \langle v_2, x\rangle && && (1.7)\\
&= \sigma_1 u_1 \langle v_1, x\rangle + \sigma_2 u_2 \langle v_2, x\rangle && Av_i = \sigma_i u_i && (1.8)\\
&= \sigma_1 u_1 v_1^T x + \sigma_2 u_2 v_2^T x && \text{Def of inner product} && (1.9)\\
&= (\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T) x && && (1.10)
\end{aligned}$$

Since this holds for an arbitrary test vector

$$M = \sigma_1 \, u_1 v_1^T + \sigma_2 \, u_2 v_2^T = \underbrace{(u_1 \; u_2)}_{U} \underbrace{\begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} v_1^T \\ v_2^T \end{pmatrix}}_{V^T} \tag{1.11}$$

1.1.3 Appplication, inference of signals

It is said that SVD can pick out "interesting" signals from data. I'll illustrate this with a simple toy model. You have a rank-1 correction to a matrix full of noise, you want to infer this correction & it's strength. This is called the Spiked Wigner model.

$$A = \lambda x x^T + W \tag{1.12}$$

where $W \sim \text{GOE}(n)$, this means that on-diagonal entires $W_{ii} \sim \mathcal{N}(0, 2)$, and off-diaongal entries $W_{ij} = W_{ji} \sim \mathcal{N}(0, 1)$. For such a matrix, the expectation value element-wise yields: $\mathbb{E}[W] = 0$ and $\mathbb{E}[W^T W] = \sigma_n^2 \mathbb{I}$.

$$\mathbb{E}[A^T A] = \lambda^2 x x^T x x^T + \sigma^2 \mathbb{I} \tag{1.13}$$

$$= \lambda^2 x^2 \, x x^T + \sigma^2 \mathbb{I} \tag{1.14}$$

The eigensystem for this is

- One eigenvector x, with eigenvalue $\lambda^2 \|x|^4 + \sigma^2$
- $d - 1$ eigenvectors v s.t. $v \perp x$, with eigenvalue σ^2.

Recall the SVD $A = U \Sigma V^T$. the V were the eigenvectors of $A^T A$, which we found contains the signal we wanted to infer. The Σ contains the eigenvalues of $A^T A$, which contain information on the strength of the signal λ and noise σ^2.

For those who know random matrix theory, you are probably skeptical due to BBP transition. This calculation doesn't ask whether you can infer xx^T from a single observation of Y, it's given infinite observations here's what you expect

1.2 Matrix Calculus

In the next week you'll have to optimize your neural network. Optimization scheme rely on gradient access to your target function, so you'll have to learn how to compute derivatives of vectors & such.

A tiny note on notation. Consider a vector $\mathbf{x} : \mathbb{R} \to \mathbb{R}^{d \times 1}$ (for example a trajectory), where $\mathbf{x} = (x_1, ..., x_d)^T$:

$$\frac{d\mathbf{x}}{dt} \equiv \begin{pmatrix} \frac{\partial x_1}{\partial t} \\ \vdots \\ \frac{\partial x_d}{\partial t} \end{pmatrix}. \tag{1.15}$$

Now consider the scalar field $\phi : \mathbb{R}^d \to \mathbb{R}$

$$\frac{\partial \phi}{\partial \mathbf{x}} \equiv \begin{pmatrix} \frac{\partial \phi}{\partial x_1} & ... & \frac{\partial \phi}{\partial x_d} \end{pmatrix} = (\nabla \phi)^T \in \mathbb{R}^{1 \times d}. \tag{1.16}$$

A nice thing about the notation is for $\phi = \phi(\mathbf{x}(t))$, computing $\frac{\partial \phi}{\partial t} = \langle \nabla \phi, \frac{d\mathbf{x}}{dt} \rangle = \frac{\partial \phi}{\partial \mathbf{x}} \frac{d\mathbf{x}}{dt}$ becomes very natural. Another bonus is it mirrors the physics notation, the derivative of a contravariant vector $\frac{\partial}{\partial x^\alpha}$ transforms as a covariant vector ∂_α. I'll note most statisticians don't use this notation, they treat $\partial \phi / \partial \mathbf{x}$ as a column vector...

Now consider the vector field $\mathbf{y} : \mathbb{R}^d \to \mathbb{R}^n$ (for example change of coordinates)

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \equiv \begin{pmatrix} - & \frac{\partial y_1}{\partial \mathbf{x}} & - \\ & \vdots & \\ - & \frac{\partial y_n}{\partial \mathbf{x}} & - \end{pmatrix} \in \mathbb{R}^{n \times d} \tag{1.17}$$

this is also known as the Jacobian. The notation is written s.t. $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\mathbf{x}$ is a sensible matrix multiplication.
Finally consider a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$

$$\frac{\partial \mathbf{M}}{\partial t} \equiv \begin{pmatrix} \frac{\partial M_{11}}{\partial t} & \cdots & \frac{\partial M_{1n}}{\partial t} \\ \vdots & \ddots & \vdots \\ \frac{\partial M_{m1}}{\partial t} & \cdots & \frac{\partial M_{mn}}{\partial t} \end{pmatrix} \tag{1.18}$$

$$\frac{\partial t}{\partial \mathbf{M}} = \begin{pmatrix} \frac{\partial t}{\partial M_{11}} & \cdots & \frac{\partial t}{\partial M_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial t}{\partial M_{m1}} & \cdots & \frac{\partial t}{\partial M_{mn}} \end{pmatrix} \tag{1.19}$$

Derivatives of matrices against vectors (and vice versa) (and higher order tensors), are defined in terms of index notation.

Here are some simple facts.

1.2.1 Derivatives of scalar forms

Let $\mathbf{a}, \mathbf{b}$ be constants

$$\frac{\partial(\mathbf{a}^T\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial(\mathbf{x}^T\mathbf{a})}{\partial \mathbf{x}} = \mathbf{a}^T \qquad \mathbf{a} \text{ constant} \tag{1.20}$$

$$\frac{\partial(\mathbf{x}^T\mathbf{M}\mathbf{x})}{\partial \mathbf{x}} = \mathbf{x}^T(\mathbf{M} + \mathbf{M}^T) \tag{1.21}$$

$$\frac{\partial(\mathbf{a}^T\mathbf{M}\mathbf{b})}{\partial \mathbf{M}} = \mathbf{a}\mathbf{b}^T \tag{1.22}$$

$$\frac{\partial(\mathbf{a}^T\mathbf{M}^T\mathbf{b})}{\partial \mathbf{M}} = \mathbf{b}\mathbf{a} \tag{1.23}$$

Note due to my notation, the $\frac{\partial}{\partial \mathbf{x}}$ terms have a relative transpose compared to Sam Roweis' notes.

1.2.2 Derivatives of vector forms

$$\frac{\partial \mathbf{x}}{\partial \mathbf{x}} = \mathbb{I} \tag{1.24}$$

$$\frac{\partial(\mathbf{M}\mathbf{x})}{\partial \mathbf{x}} = \mathbf{M} \tag{1.25}$$

Problem 1

Derive the back-propagation for a two layer neural network. That is given

$$\mathcal{L} = (y - \hat{y})^2 \tag{1.26}$$

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} a_i \, \sigma(\mathbf{w}_i^T \mathbf{x} + b_i) \tag{1.27}$$

where $y, \hat{y}, a_i, b_i \in \mathbb{R}$ are scalars, and $\mathbf{w}_i, \mathbf{x} \in \mathbb{R}^d$ are vectors. Note $\mathbf{w}_i$ is not the entry of a vector, there are $i = 1, ..., N$ $\mathbf{w}_i$ vectors.

Compute

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} \tag{1.28}$$

this is the notation for the gradient w.r.t. $\mathbf{w}_i$.

1.2.3 Matrix Inversions

Fact 2 (Sherman-Morrison) *Let A be an invertible square matrix, and u, v be vectors.*

$$(A + uv^T)^{-1} = A^{-1} + \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \tag{1.29}$$

Proof: Here's a constructive proof. Say you want to solve for x

$$(A + uv^T)x = y \tag{1.30}$$

$$x = A^{-1}y - A^{-1}uv^Tx \tag{1.31}$$

Notice

$$v^Tx = v^TA^{-1}y - v^TA^{-1}uv^Tx \tag{1.32}$$

$$(1 + v^TA^{-1}u)v^Tx = v^TA^{-1}y \tag{1.33}$$

$$v^Tx = \frac{v^TA^{-1}}{1 + v^TA^{-1}u}y \tag{1.34}$$

Therefore

$$x = \left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right) y \tag{1.35}$$

□

The idea is that a rank one perturbation uv^T to a full rank matrix A, yields an inverse which is a rank one perturbation to A^{-1}.

Fact 3 (Woodbury) *A generalization of the previous fact is*

$$(A + UCV)^{-1} = A^{-1} + A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{1.36}$$

where A is $n \times n$, C is $k \times k$, U is $n \times k$ and V is $k \times n$.

Proof left as exercise.

1.3 Time Complexity

Since we're talking about these things in the context of a computational class, it'll be good to recap the time complexity of such algorithms. Just keep these in the back of your mind.

- Matrix multiplication: $\mathcal{O}(n^{2.8})$.
- Matrix inverse implemented in `numpy.linalg.solve`: $\mathcal{O}(n^3)$.
- SVD for a $n \times m$ matrix (s.t. $n \leq m$) : $\mathcal{O}(mn^2)$.
- Determinant $\mathcal{O}(n^3)$

As a final note, the time complexity of an algorithm doesn't translate to the actual run time of an algorithm.

See https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#Matrix_algebra for more information.

References

[1] Michael Hutchings, Notes on singular value decomposition for Math 54, https://math.berkeley.edu/~hutching/teach/54-2017/svd-notes.pdf.

[2] Gregory Gundersen, Singular Value Decomposition as Simply as Possible, https://gregorygundersen.com/blog/2018/12/10/svd/

[3] Leslie Lamport (1994) *LaTeX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.

2 Review of Probability

Definition 3 (Conditional Probability)

$$\mathbb{P}[A|B] = \frac{\mathbb{P}[A \cap B]}{P[B]} \tag{2.1}$$

Notice that $\mathbb{P}[A \cap B] = \mathbb{P}[B \cap A]$, this allows us to related $\mathbb{P}[A|B]$ and $\mathbb{P}[B|A]$.

$$\mathbb{P}[A|B] = \frac{\mathbb{P}[A \cap B]}{P[B]} \tag{2.2}$$

$$= \frac{\mathbb{P}[B \cap A]}{P[B]} \tag{2.3}$$

$$\boxed{\mathbb{P}[A|B] = \frac{\mathbb{P}[B|A]\,\mathbb{P}[A]}{\mathbb{P}[B]}} \tag{2.4}$$

This is **Bayes' Formula**.

Definition 4 (Probability Density Function) *A function with the following properties is a* ***probability density***

- *Positive:* $p : \mathcal{X} \to \mathbb{R}_{\geq 0}$
- *Normalized:* $\int_{\mathcal{X}} p(x)\,dx = 1$

It is interpreted as the probability of observing an event $A \subset \mathcal{X}$ *as*

$$\mathbb{P}[x \in A] = \int_{A \subset \mathcal{X}} p(x)\,dx \tag{2.5}$$

The nice part of densities is that you can compute statistics with that. I.e. what's the mean, variance.

$$\mathbb{E}_{x \sim p}[f(x)] = \int_{\mathcal{X}} f(x)\,p(x)\,dx \tag{2.6}$$

Definition 5 (Characteristic Function) *Consider the probability distribution* p_X*. It has an associated* ***characteristic function*** φ_X *which is it's Fourier Transform*

$$\varphi_X(k) = \int_{\mathbb{R}} e^{ikx} p(x)\,dx = \mathbb{E}_{x \sim p}[e^{ikx}] \tag{2.7}$$

3 Review of Statistics & Loss Functions

In machine learning, we adjust a model's parameters θ to minimize a loss function $\mathcal{L}(\theta)$. There's a bunch so I think it's nice to hear where they come from. We'll cover

- Mean squared error (MSE)

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \|y_i - f_\theta(x_i)\|^2$$

- Cross entropy

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \|$$

- MSE + L2 Regularization (Ridge)

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \|y_i - f_\theta(x_i)\|_2^2 + \lambda \|\theta\|_2^2$$

3.1 Maximum Likelihood Inference & Mean Squared Error

Say you have the dataset $\mathcal{D} = \{(y_i, x_i)\}_{i=1}^n$ (which we assume you observed in an iid way). You believe that y_i is a noisy observation of some model $f_\theta(x_i)$. Your objective is to come up with the "best" estimate of the parameter θ which matches the data $\mathcal{D}$... You think about it for some while, and realize you maximize the probability of seeing the data for a given θ. This is **maximum likelihood estimation (MLE)**.

To illustrate this method (and all others), we have to assume a particular model. So let's say you believe the noise is additive & gaussian:

$$y_i = f_\theta(x_i) + \epsilon_i, \quad \text{where } \epsilon_i \sim_{iid} \mathcal{N}(0, \mathbb{I}) \tag{3.1}$$

Since ϵ_i is a random variable, you can interpret y_i as a random variable as well.

$$y_i \sim \mathcal{N}(f_\theta(x_i), \mathbb{I}) \tag{3.2}$$

$$p(y_i|\theta) \propto \exp\left(-\frac{1}{2}(y_i - f_\theta(x_i))^2\right) \tag{3.3}$$

$$\log p(y_i|\theta) = -\frac{1}{2}(y_i - f_\theta(x_i))^2 + \text{Constant w.r.t. } \theta \tag{3.4}$$

I've wrote the log prob for reasons that will become clear in a moment.

Note you have more data $\{(y_i, x_i)\}_{i=1}^n$ (which is all iid), so you actually have a joint distribution.

$$p(y_1, ..., y_n|\theta) = \prod_i p(y_i|\theta) \tag{3.5}$$

We'll call this the **likelihood** $L(\theta)$ (that is the likeliness / probability of seeing the data given a configuration of model parameters). For MLE, you choose $\hat{\theta}$ which maximizes the likelihood. However arg max of a product of functions is quite difficult, we can compose the function w/ a monontonic function, and that leaves the arg max invariant.

$$\log L(\theta) = \log p(y_1, ..., y_n|\theta) \tag{3.6}$$

$$= \sum_i \log p(y_i|\theta) \tag{3.7}$$

$$\propto \sum_i (y_i - f_\theta(x_i))^2 \tag{3.8}$$

This recovers the MSE loss.

3.2 Cross Entropy & Another MLE

3.3 L2 Regularization

In Bayesian statistics, instead of asking what's the probability of seeing the data given a model parameter, we ask *what's the probability of seeing a model parameter given the data?* We can formalize the inverse question using Bayes' theorem

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \tag{3.9}$$

- $p(\theta)$ is your prior. It encodes your prior beliefs into the distribution.

- $p(y|\theta)$ is the likelihood (from the previous sections)
- $p(\theta|y)$ is the posterior. It accounts for your prior beliefs & what the data says (likelihood).
- $p(y)$ is the evidence. I won't say much about it today.

If you ask, what's the parameter maximizes the posterior (probability of seeing a parameter given the data), this is called **maximum aposteriori estimation (MAP)**.

$$\hat{\theta}_{MAP} = \arg\max_{\theta} p(\theta|y) \tag{3.10}$$

For an example, let's assume we have the additive noise model

$$y_i = f_\theta(x_i) + \epsilon_i \tag{3.11}$$

and that you believe the weights should look distributed according to a Gaussian

$$p(\theta) = \mathcal{N}(0, \lambda^{-1}\mathbb{I}) \tag{3.12}$$

You can see that log posterior has the form

$$\log p(\theta|y) = \sum_i \|y_i - f_\theta(x_i)\|^2 + \lambda\|\theta\|^2 \tag{3.13}$$

3.4 Minimizing the loss function

- The value of the MSE, in a traditional statistics setting, tells you about the uncertainity quantification of the model. However ML models tend to not obey this.
- Difficulty of optimizing via oracle access.
- However! Do you even want to perfectly minimize the loss function? Memorization.

4 Linear Regression

Consider making iid noisy observations of data $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$. We'll assume that the noise is additive, that is

$$y_i = f(x_i) + \epsilon_i \tag{4.1}$$

where $f(x) = \beta^T x$ is the model (we've assumed it's linear for this discussion) and the noise is gaussian $\epsilon_i = \mathcal{N}(0, \sigma_i^2)$ (which is another assumption for this discussion). Since ϵ_i is a random variable, this implies that y_i is also a random variable

$$y_i|\beta = \mathcal{N}(y_i; \beta^T x_i, \sigma_i^2) \tag{4.2}$$

This is just one observation, but in fact, we have a joint distribution $p(y|x) \equiv p(y_1, ..., y_N|x_1, ..., x_N)$ over all observations, which we'll call the **likelihood**. Since observations are iid, it factorizes.

$$p(y|\beta) = \prod_i p(y_i|x_i) \tag{4.3}$$

Your task is to find the β which "best" describes the data. I'll note that "best" is subjective and we'll discuss consequences of this later.

4.1 Frequentist, Maximum Likelihood Estimator

One method is **maximum likelihood estimation**, that is you selct the parameters which is the global maximizer of the likelihood. Why? Just read off what you're doing: adjust β s.t. the probability of having this combination of y's (given x's) is highest.

Apart from being very intuitive, there are also strong theoretical guaranties (which I won't have time to prove) (Notation: when I generically talk about model parameters, we use θ)

- Consistency: $\lim_{n\to\infty} \hat{\theta}_n = \theta$
- Normality: $\hat{\theta}_n \sim \mathcal{N}(0, \mathcal{I})$ (where $\mathcal{I}$ is the fisher information matrix)
- Efficiency: $\text{Var}(\hat{\theta}) \geq 1/\mathcal{I}(\theta)$.

Since arg max is invariant under compositions of monotonic functions, we can maximize the log-likelihood which emits a nicer function

$$\log p(y|x) = \sum_i \log p(y_i|x_i) \tag{4.4}$$

$$= \sum_i \log \mathcal{N}(y_i; \beta^T x_i, \sigma_i^2) \tag{4.5}$$

$$= \sum_i -\frac{1}{2}\frac{\left(y_i - \beta^T x_i\right)^2}{\sigma_i^2} + \text{Constant} \tag{4.6}$$

A small comment, this is why you "minimize the squared error" when fitting straight lines in lab, you have been secretly doing maximum likelihood inference this whole time. Notice this is a quadratic form, so you can rewrite it using matrix multiplication

$$\sum_{i=1}^{n}(y_i - \beta^T x_i)^2/\sigma_i^2 = (y - X\beta)^T\Sigma^{-1}(y - X\beta) \tag{4.7}$$

$$\text{where: } y = \begin{pmatrix} y_1 \\ \vdots \\ v_n \end{pmatrix} \in \mathbb{R}^n, \tag{4.8}$$

$$\beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \in \mathbb{R}^p \tag{4.9}$$

$$\Sigma = \text{diag}(\sigma_1^2, ..., \sigma_n^2) \in \mathbb{R}^{n\times n} \tag{4.10}$$

$$X = \begin{pmatrix} - & x_1^T & - \\ - & x_2^T & - \\ & \vdots & \\ - & x_n^T & - \end{pmatrix} \in \mathbb{R}^{n\times p} \tag{4.11}$$

From here we can find the argmax of the quantity

$$0 = \frac{\partial \log p(y|x)}{\partial \beta}\Big|_{\beta=\hat{\beta}} = X^T\Sigma^{-1}(y - X\beta) \tag{4.12}$$

$$\implies X^T\Sigma^{-1}y = X^T\Sigma^{-1}X\hat{\beta} \tag{4.13}$$

$$\boxed{\hat{\beta}_{MLE} = (X^T\Sigma^{-1}X)^{-1}X^T\Sigma^{-1}y} \tag{4.14}$$

and find the maximum likelihood estimate for β.

Now we can talk about inference. Say your boss gives you new data X_*, and you're asked what is the corresponding $\hat{y}_*$. You'll report back

$$\boxed{\hat{y}_* = X_* \hat{\beta}_{MLE}} \tag{4.15}$$

4.2 Bayesian Linear Regression

In the Bayesian framework, you're asked what is the probability of seeing the model parameters *given* the data $p(\beta|y)$. You can calculate this using Bayes's formula

$$p(\beta|y) = \frac{p(y|\beta)p(\beta)}{p(y)} \tag{4.16}$$

Part II
Supervised Learning

5 PyTorch 101

5.1 Training Dynamics

Thinking of back-propagation as a high-level chain rule is not enough to get a neural network to train. For example, here's some food for thought...

- We want to $\arg\min_\theta \mathcal{L}(\theta)$, what scheme do you implement? Note that $\mathcal{L}(\theta)$ implicitly depends on the data and model architecture, so perhaps that'll affect your answer.
- In the model architecture, how do you initialize the model weights?

5.2 Whitening

The following analysis will assume the data distribution $\mathcal{D} = \{x^{(i)}\}_i \sim^{iid} p_{data}$, has vanishing $\mathbb{E}[x^{(i)}] = 0$ and covariance $\mathbb{E}[x^{(i)}x^{(i)T}] = \sigma^2\mathbb{I}$.

You might ask, is it this a fair assumption, and the answer is totally. You can always transform data to have these properties.

Consider the data matrix $X \in \mathbb{R}^{b\times d}$ (each row is a d-dimensional data point)

$$X \equiv \begin{pmatrix} - & x^{(1)} & - \\ \vdots & \vdots & \vdots \\ - & x^{(b)} & - \end{pmatrix} \tag{5.1}$$

5.3 Back propagation

Using the rules from our matrix cheat sheet. Let $x \in \mathbb{R}^m$. Consider a linear layer

$$\text{Linear} : \mathbb{R}^m \to \mathbb{R}^n \tag{5.2}$$

$$z(x) = \text{Linear}(x) = Wx + b \tag{5.3}$$

$$\frac{\partial}{\partial W_{ij}}\text{Linear}(x) = x_j \tag{5.4}$$

$$\frac{\partial}{\partial W_{ij}}\sigma(z) \tag{5.5}$$

5.4 Weight Initialization

As you propogat

5.5 PyTorch Tutorial

See `pytorch_tutorial.ipynb` in the repo.

6 Double Descent

6.1 Soft Inductive Biases

Another way to conceptualize this is **soft inductive biases** (see Andrew Gordon Willson's paper https://arxiv.org/pdf/2503.02113).

7 Training Large Models

7.1 Transformers

7.2 μP Optimizer

8 Geometric Deep Learning

Part III
Probabilistic Inference

9 Variational Inference

There were a couple of bottlenecks on MCMC

1. If the query time for the likelihood is quite long, then sampling the distribution will take very long.
2. In multimodal distributions, we have no guarantees when the chains will mix / find other mode, so MCMC may fail in that regime. Note, multimodal is NP-hard so this new technique won't entirely solve it.

The idea is

9.1 Distance of Probability Measures

There are couple ways to measure the distance between probability measures. Each is good for a certain context. A (very) non-exhaustive list

- Total Variation distance

$$\|p - q\|_{TV} = \frac{1}{2} \int |p(x) - q(x)| \, dx \tag{9.1}$$

  Comments: It's an L_1 bound on the distributions, so if you don't care about the moments but making sure the spaces are close, then it's good. However, you can NOT bound moments using this...

- Kullback-Leibler divergence

$$\mathrm{KL}(p\|q) = \int p(x) \, \log \frac{p(x)}{q(x)} \, dx \tag{9.2}$$

  Note this is not symmetric, hence why we don't call it a distance but instead a divergence.

  Comments: It's easy to numerically compute. You can rewrite it as

$$\mathrm{KL}(p\|q) = \mathbb{E}_{x\sim p}[\log p(x)] - \mathbb{E}_{x\sim p}[\log q(x)]. \tag{9.3}$$

  In computation, you usually have access to log-probabilities, and you can approximate the expectation using a sampling technique. Also, the KL is related to the TV distance by Pinsker's inequality: $\|p - q\|_{TV} \leq \sqrt{\frac{1}{2}\mathrm{KL}(p\|q)}$.

- Wasserstein Distance

$$W_p(p, q) = \sup_{\gamma \in \Gamma} \int \|x - y\|^p \, d\gamma(x, y)^{1/p} \tag{9.4}$$

  where Γ is the set of couplings on p, q. A coupling $\gamma(p, q)$ is defined a joint probability distribution s.t. it marginalizes to recover p, q; $\int \gamma(p(x), q(y))dx = q(y)$ and vice versa.

  Comments: This is considered the most natural way to compare distributions. You can also bound moments, unlike the TV distance.

Any of these measurements will equal zero if and only if, the two distributions are equal. Also they're all non-negative. Making them good loss functions for neural networks.

10 Normalizing Flows

To get variational inference to work better than just MCMC, here's a couple things I want

1. Better high dimensional scaling
2. I want IID samples
3. Ability to evaluate the log-probability of the proposal distribution.

Points (2) and (3) give us a hint as to a potential solution. Imagine constructing a map between an easy to evaluate distribution (i.e. Gaussian) and your target distribution, then transporting sampling according to this map. A potential problem is an approximate map will yield bias in your final answer (I'll touch upon this later).

An example of this is the Box-Muller transformation. It maps the uniform distribution $U_1, U_2 \sim \text{Unif}[0,1]^2$ to a Gaussian $\mathcal{N}(0, \mathbb{I}_2)$.

$$Z_1 = R\cos(\Theta) = \sqrt{-2\log U_1}\cos(2\pi U_2) \tag{10.1}$$

$$Z_2 = R\sin(\Theta) = \sqrt{-2\log U_1}\sin(2\pi U_2) \tag{10.2}$$

You can show that $\text{Law}(Z_1, Z_2) = \mathcal{N}(0, \mathbb{I}_2)$. I

$$z = f(x) \tag{10.3}$$

$$\int \nu(z)dz = \tag{10.4}$$

The idea is to construct a map between a base distribution μ and a target distribution π. You can imagine applying diffeomorphisms to μ iteratively, until it well approximates π.

11 Review of Stochastic Differential Equations

You've probably heard of SDEs before, but they aren't covered in the main-stream physics education. So I'll attempt to do a brief introduction.

There was a botanist studying pollen grains in water. He noticed the motion was jittery, moving randomly in all directions. You can imagine a heuristic model being

$$X_{t+h} = X_t + h^\alpha z_t \tag{11.1}$$

where $z_t \sim \mathcal{N}(0, \mathbb{I})$ (iid at every time t) is random noise, and h is the step size (according to the time-discretization) to the power α. In an attempt to find a continuous time model in the limit $h \to 0$ (discretization goes to zero), I'll recurse to time zero.

$$X_t = X_{t-h} + h^\alpha z_{t-h} \tag{11.2}$$

$$= X_{t-2h} + h^\alpha (z_{t-2h} + z_{t-h}) \tag{11.3}$$

$$= X_{t-3h} + h^\alpha (z_{t-3h} + z_{t-2h} + z_{t-h}) \tag{11.4}$$

$$= X_0 + h^\alpha \sum_{n=1}^{t/h} z_{t-nh} \tag{11.5}$$

Since we're physicists, let's center the initial position $X_0 = 0$. We now note that

$$h^\alpha \sum_{n=1}^{t/h+1} z_{t-nh} \sim \mathcal{N}(\big(0, h^{2\alpha-1}t\big) \tag{11.6}$$

To keep the model independent on the size of the discretization, I'll choose $\alpha = 1/2$. Leaving us with

$$X_t - X_0 \sim \mathcal{N}(0, t) \tag{11.7}$$

This was quite heuristic, but we have some take aways. When making an infinitesimal that behaves randomly, it has units $\sqrt{dt}$.

Now that we have some intuition for the system, we can develop something more rigorous.

Definition 6 (Weiner Process / Brownian Motion) *Brownian motion $(W_t)_{t\geq 0}$ is a stochastic process such that*

1. *Initializes at zero: $W_0 = 0$*
2. *Normal increments: $W_t - W_s \sim \mathcal{N}(0, (t-s)\mathbb{I})$, for $0 \leq s \leq t$.*
3. *Independent increments: $W_{t_1} - W_{t_0}$ is independent from $W_{t_i} - W_{t_j}$.*

The idea of a stochastic differential equations is to extend the dynamics of ODEs to the dynamics where you have random fluctuations of force. Such things are no-where differentiable, so how can we recover a derivative-esq operation w/o using a derivative? Well ODEs have that

$$\frac{dX_t}{dt} = \mu_t(X_t) \implies X_{t+h} = X_t + h\, u_t(X_t) + \mathcal{O}(h^2) \tag{11.8}$$

Similarly for an SDE (ODE with stochastic fluctuations)

$$X_{t+h} = X_t + X_t + h u_t(X_t) + (W_{t+h} - W_t)\, \sigma_t(X_t) + \mathcal{O}(h^{3/2}) \tag{11.9}$$

The $\mathcal{O}(h^{3/2})$ is due to fluctuations on the order $h\,(W_{t+h} - W_t)$, as we've noted that $W_{t+h} - W_t$ is order $\sqrt{h}$.

For brevity, we'll use a shorthand for 11.9

$$dX_t = \mu_t(X_t)\, dt + \sigma_t(X_t)\, dW_t \tag{11.10}$$

Theorem 2 (Fokker-Planck Equation) *Consider the stochastic differential equation*

$$dX_t = \mu_t(X_t)\, dt + \sigma_t dW_t \tag{11.11}$$

$$X_0 \sim p_0 \qquad \textit{Boundary condition} \tag{11.12}$$

where $\mu_t : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma_t : [0,1] \to \mathbb{R}^d$ are deterministic functions. Then the corresponding probability distribution $X_t \sim p_t$ solves a partial differential equation of the following form

$$\partial_t p_t(x) = -\nabla \cdot (\mu_t\, p_t) + \frac{\sigma_t^2}{2} \Delta p_t \tag{11.13}$$

$$p_{t=0} = p_0 \qquad \textit{Boundary condition} \tag{11.14}$$

Proof: Since X_t is a random variable, it has a corresponding probability density function. I'll notate this as p_t. Now you need to show that p_t have a the corresponding time evolution. The trick to do this, is to recall the trick you employ when you show something is secretly a delta function. You would integrate it against a test function $f(x)$ and show

it behaved as expected. We'll do the same thing.

$$\partial_t \mathbb{E}[f(X_t)] = \lim_{h\to 0} \frac{1}{h}\mathbb{E}[f(X_{t+h}) - f(X_t)] \tag{11.15}$$

$$= \lim_{h\to 0} \mathbb{E}[\nabla f^T\, u_t(X_t) + \frac{\sigma_t^2}{2}\Delta f(X_t) + \mathcal{O}(h)] \tag{11.16}$$

$$= \int \nabla f^T(x) u_t(x) p_t(x) + \frac{\sigma_t^2}{2}\Delta f(x) p_t(x)\, dx \tag{11.17}$$

$$= \int -f(x)\, \nabla \cdot (u_t(x)p_t(x)) + f(x)\frac{\sigma_t^2}{2}\Delta p_t(x)\, dx \tag{11.18}$$

On the LHS

$$\partial_t \mathbb{E}[f(X_t)] = \int f(x)\partial_t p_t(x)\, dx \tag{11.19}$$

Put LHS = RHS, and you're done. □

As a side note, this proof is typically done using Ito's lemma, and (I think) it holds in weak-convergence. This proof uses the Euler Maruyama, and taking $h \to 0$ gives strong convergence (which in turn implies weak-convergence).

12 Score Based Diffusion

13 Stochastic Interpolants