CMPTG 5 - Statistical Physics of Neural Networks

Sidharth Kannan

May 2025

1 Introduction

Over the past few weeks, we have spent our time studying how ideas from physics can be brought into bear to understand the problem of *generative modeling*. From this week onward, we are going to change tacks a little bit and turn our attention to more foundational questions in the study of neural networks. In particular, we are going to see how the physics of modeling complex system (where here, complex means systems of many many degrees of freedom) can be applied to neural networks. We are going to try to understand fundamental properties of neural networks, like how to think about the effect of depth and width of the network, the effect of weight initialization, and the effect of residual connections. The approach that we will take is that we will try and directly probe the *distribution of intermediate activations* of neural networks, and compute properties of this distribution like its mean, variance, and higher order correlators.

This will culminate in an understanding of two very contemporary topics in machine learning, one from theory and one from practice: the oversmoothing problem in graph neural networks, and the neural tangent kernel.

1.1 A Bit of History

The field of statistical mechanics, whose tools we will employ in our study, grew out of the desire to understand macroscopic properties of complex systems from a first principles, microscopic approach. Back in the 1800s, people had (through lots of experiment, trial and error) derived the laws of thermodynamics. This includes things like the ideal gas law, $PV = NRT$. However, their understanding of these things were empirical; the laws matched observations, but they could not rigorously be derived from the first principles, microscopic laws of nature, since these were laws that applies to astronomically complicated (10^{23} atom) systems.

Making this connection was further complicated by the fact that at this point in time, the existence of atoms was not widely accepted scientific consensus. It was not until the pioneering work of Maxwell and Boltzmann that the connection was drawn between the microscopic behavior of atoms and the *statistical* behavior of gases.

But this connection was made, and it proved to be an extraordinarily useful method for understanding complex systems. Today, we will begin our study of neural networks (comprised of many, many atoms), adopting a similar philosophy to Maxwell and Boltzmann. We have a slightly easier problem, in that unlike they, who did not know what the microscopic behavior of atoms was, we know precisely how the individual neurons in our network behave.

2 Problem Setting

For the majority of the remainder of this course, we will restrict ourselves to a detailed study of the multilayer perceptron. Later on, we will see how we can generalize this study to broader classes of neural networks.

As a brief recap, a multilayer perceptron is a class of neural network that takes a vector as input, and applies a sequence of linear transformations, interleaved by nonlinearities.

$$\mathbf{z}^{(l)} = W_l(\sigma(W_{l-1}(...(\sigma(W_1\mathbf{x} + b_1))) + b_{l-1})) + b_l \tag{1}$$

where here, W denotes the weight matrices, b denotes the biases, σ is the nonlinearity, $\mathbf{x}$ is the input and $\mathbf{z}^{(l)}$ is the output of the l-th layer. We will call the output of the linear transformation our *pre-activation*, which is transformed into the *activation* by the nonlinear *activation function*, σ. This is called the forward iteration equation for the MLP. We can also write this as a recurrence relation.

$$\mathbf{z}^{(l+1)} = W_l\sigma(\mathbf{z}^{(l)}) + b_l \tag{2}$$

The inputs $\mathbf{x}_i$ during training (along with a set of labels) comprise a training set, which we will denote with $\mathcal{D}$. Given this set of sample inputs and outputs, we want to learn to predict the output for inputs we have not seen before. The way this is usually phrased is that given a set of inputs and outputs, we wish to approximate the function f that maps those inputs to those outputs. We do this by first parameterizing our function with a neural network, f_θ. We initially sample the parameters from some computationally simple distribution. For example, we might sample the entries of the weight matrices and bias vectors from a Gaussian.

We then update the parameters, θ, using some learning procedure until the new parameters $\theta^\star$ are such that $f_{\theta^\star}$ is very close to f. Our goal is to understand the behavior of this trained network from the first principles microscopic behavior of the neurons.

We can illustrate the problems we will encounter by attempting a (very schematic) Taylor expansion of our trained network function around the initialization parameters, θ.

$$f_{\theta^\star}(x) = f(\theta) + \frac{df}{d\theta}(\theta^\star - \theta) + \frac{1}{2}\frac{d^2f}{d\theta^2}(\theta^\star - \theta)^2 + ... \tag{3}$$

This Taylor expansion, in principle, tells us everything we need to know about our trained network function. However, it runs up against three, very significant problems.

1. The Taylor expansion contains an infinite number of terms. In particular, since we have no idea if θ is close to $\theta^\star$, we cannot say anything about how many terms to include.

2. The function f_θ is dependent on the *random* parameter θ, sampled from some distribution, so f_θ will in general be a *random function*, and its derivatives will also be random functions, with some extremely complex structure.

3. $\theta^\star$ is the result of training. Thus, we need to account for the training algorithm, which depends on the initialization parameters, the dataset, and a number of hyperparameters.

If we wish to study the trained network function using Taylor expansion, we need to solve all of these. The goal of our study will be to solve exactly these problems in some reasonable limits, and make certain conclusions about the behavior of neural networks. This is actually a case in which

the complexity (large number of neurons) can work to our advantage. This is because if we take the limit of many neurons, *each* neuron individually contributes very little to the network output. Thus, we needn't worry about their individual behavior, but instead their aggregate statistical behavior.

There are two sensible limits to take: *infinite width* and *infinite depth*. It turns out that infinite depth leads to a chaotic network, and infinite width leads to a simple, analytically tractable one. However, the infinite width limit is too simplistic. It cannot account for some of the complex behavior that neural networks exhibit. We will take a slightly weaker limit, that the network is much wider than it is deep, and in this case, our analysis will be both tractable analytically, and result in the ability to study a lot of phenomena of practical interest.

In particular, we will solve the three problems above because the Taylor series will become such that we can truncate it after a few terms. The distributions for the network output will become simple and of a tractable form, and in the pure infinite width limit, we will see that the dependence on the training algorithm actually vanishes entirely.

3 Mathematical Preliminaries

3.1 Gaussian Integrals

This section of the course will rely heavily on the ability to take Gaussian integrals of various flavors. We will go over that in brief here.

3.1.1 Univariate Gaussians

The simplest Gaussian integral is the univariate case

$$p(z) = \frac{1}{Z} e^{-\frac{x^2}{2}} \tag{4}$$

In order to make this a valid probability distribution, we must find the normalization constant, Z by taking the integral

$$Z = \int_{-\infty}^{\infty} e^{-\frac{s^2}{2}} ds \tag{5}$$

If you've never see the trick for integrating this before, its quite a neat one, so we'll show it here.

$$Z^2 = \left(\int_{-\infty}^{\infty} e^{-\frac{x^2}{2}} dx \right) \left(\int_{-\infty}^{\infty} e^{-\frac{y^2}{2}} dy \right) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(x^2+y^2)} dx dy \tag{6}$$

Now, we can transform this to polar coordinates.

$$Z^2 = \int_0^{\infty} \int_0^{2\pi} r e^{-\frac{1}{2}r^2} dr d\theta \tag{7}$$

The θ integral is trivial, since the integrand does not depend on θ. The r integral can be solved with the substitution $u = \frac{1}{2}r^2$, $du = r$

$$Z^2 = 2\pi \int_0^{\infty} e^{-u} du = 2\pi \rightarrow Z = \sqrt{2\pi} \tag{8}$$

For a Gaussian with non-zero mean, it is easy to see that the integral does not change. Shifting the mean simply translates the distribution, and so the total area under the curve does not change. For a Gaussian with variance K, the same approach gives that the integral is $\sqrt{2\pi K}$.

By definition, we know the mean is

$$\mathbb{E}[z] = \int_{-\infty}^{\infty} z e^{\frac{(z-\mu)^2}{2K}} = \mu \tag{9}$$

We will generally be interested in Gaussian *moments* of mean-zero Gaussians. That is, expectations of the form $\mathbb{E}[z^m]$ for some integer m. For a mean zero Gaussian, we can see that all of the *odd* moments vanish, since the integrand will be an odd function (symmetric w.r.t. the substitution $z \to -z$. The even moments are non-trivial to compute.

We will proceed by introducing a *Gaussian with source*.

$$Z = \int_{-\infty}^{\infty} dz e^{-\frac{z^2}{2K} + Jz} \tag{10}$$

where Jz is called the source term and $Z_{K,J}$ is called the partition function with source. We will see that this serves as the *moment generating function* for the even moments.

We can proceed by completing the square in the exponent:

$$Z_{K,J} = \int_{-\infty}^{\infty} dz e^{-\frac{(z-JK)^2}{2K}} e^{\frac{KJ^2}{2}} = e^{\frac{KJ^2}{2}} \sqrt{2\pi K} \tag{11}$$

Now, if we try to compute the even moments of a Gaussian, we see that

$$\int_{-\infty}^{\infty} z^{2m} e^{-\frac{z^2}{2K}} dz \tag{12}$$

$$= \left(\frac{d}{dJ}\right)^{2m} \left(\int_{-\infty}^{\infty} e^{-\frac{z^2}{2K} + Jz}\, dz\right)\Big|_{J=0} = \left[\left(\frac{d}{dJ}\right)^{2m} Z_{K,J}\right]\Big|_{J=0} \tag{13}$$

Or, our moments are a simple function of the even Taylor coefficients of the partition function with source around 0. Let's compute a few examples. For instance, for $2m = 2$ we have:

$$\mathbb{E}[z^2] = \frac{I_{K,1}}{\sqrt{2\pi K}} = \frac{d^2}{dJ^2}\left(e^{\frac{KJ^2}{2}}\right)\Big|_{J=0} = e^{\frac{KJ^2}{2}}\left(K + K^2J^2\right)\Big|_{J=0} = K$$

and for $2m = 4$ we have:

$$\mathbb{E}[z^4] = \frac{I_{K,2}}{\sqrt{2\pi K}} = \frac{d^4}{dJ^4}\left(e^{\frac{KJ^2}{2}}\right)\Big|_{J=0} = e^{\frac{KJ^2}{2}}\left(3K^2 + 6K^3J^2 + K^4J^4\right)\Big|_{J=0} = 3K^2$$

The general pattern is that

$$\mathbb{E}[z^{2m}] = K^m(2m-1)!! \tag{14}$$

This is Wick's theorem (in the univariate case).

3.1.2 Multivariate Gaussians

The multivariate Gaussian in N dimensions is given by

$$\exp\left[-\frac{1}{2}\sum_{\mu,\nu=1}^{N} z_\mu (K^{-1})_{\mu\nu} z_\nu\right] \tag{15}$$

where now instead of 1 variance, we have a full *covariance* matrix. The covariance matrix is a symmetric positive definite matrix whose entries are given by

$$K_{\mu\nu} = \mathbb{E}[z_\mu z_\nu] - \mathbb{E}[z_\mu]\mathbb{E}[z_\nu] \tag{16}$$

Now, to make this a probability distribution, we again need to normalize it.

$$I_K = \int_{-\infty}^{\infty} dz^N \exp\left[-\frac{1}{2}\sum_{\mu,\nu=1}^{N} z_\mu (K^{-1})_{\mu\nu} z_\nu\right] \tag{17}$$

At first glance, this is tricky, since we have to integrate over N variables, z_μ, but the integrands all involve all the variables. However, since the covariance matrix is SPD, we know that there is a orthogonal matrix, O, which diagonalizes it. We can rewrite the integral as

$$= \int_{-\infty}^{\infty} dz^N \exp\left[-\frac{1}{2}\sum_{\mu,\nu=1}^{N} z_\mu (O^T O)(K^{-1})_{\mu\nu}(O^T O) z_\nu\right] \tag{18}$$

$$= \int_{-\infty}^{\infty} dz^N \exp\left[-\frac{1}{2}\sum_{\mu,\nu=1}^{N} (zO^T)_\mu (OK^{-1}O^T)_{\mu\nu}(Oz)_\nu\right] \tag{19}$$

Since the matrix in parentheses is diagonal, we can rewrite the sum as

$$= \int_{-\infty}^{\infty} dz^N \exp\left[-\frac{1}{2}\sum_{\mu=1}^{N} \frac{1}{\lambda_\mu}(Oz_\mu)^2\right] \tag{20}$$

Now, we have something which depends only on one index, so we see that this factorizes into the product on N independent Gaussian integrals.

$$= \prod_{i=1}^{N} \int_{-\infty}^{\infty} du_i \exp\left(-\frac{u_i^2}{2\lambda_i}\right) = \prod_{i=1}^{N} \sqrt{2\pi\lambda_i} = \sqrt{|2\pi K|} \tag{21}$$

We won't go through the derivation of Wick's theorem for the multivariate case, but we will state the result here.

$$\mathbb{E}[z_{\mu_1} z_{\mu_2} ... z_{\mu_{2m}}] = \sum_{\text{all pairs}} K_{\mu_{k_1}\mu_{k_2}} ... K_{\mu_{k_{2m-1}}\mu_{k_{2m}}} \tag{22}$$

That is, we first come up with all possible ways to pair up our $2m$ indices. For each pairing, we look at the pairs (i, j) and take the product of all the K_{ij}.

To make this more concrete, take the following examples.

$$\mathbb{E}[z_{\mu_1} z_{\mu_2}] = K_{\mu_1 \mu_2} \tag{23}$$

since there is only one way to pair them.

$$\mathbb{E}[z_{\mu_1} z_{\mu_2} z_{\mu_3} z_{\mu_4}] = K_{\mu_1 \mu_2} K_{\mu_3 \mu_4} + K_{\mu_1 \mu_3} K_{\mu_2 \mu_4} + K_{\mu_2 \mu_3} K_{\mu_1 \mu_4} \tag{24}$$

3.2 Probability

Say that we are given some probability distribution, $p(z)$. We will primarily be concerned with computing expectations of functions against this distribution.

We've already seen one class of functions for which we can do this: the moments. In principle, knowing the moments completely characterizes our distribution, since for any other function whose expectation we want to take, we can do so by Taylor expanding the function and then using the expectations of the moments.

A lot of our work will be done by quantifying how far our particular distribution is from Gaussian. One way that we can do that is by looking at something called the *connected correlators* or *cumulants* of our distribution.

Let's demonstrate by example. The first cumulant is the mean: $\mathbb{E}[z]$. The second is the variance: $\mathbb{E}[z_{\mu_1} z_{\mu_2}]|_{\text{connected}} = \mathbb{E}[z_{\mu_1} z_{\mu_2}] - \mathbb{E}[z_{\mu_1}]\mathbb{E}[z_{\mu_2}]$. The next cumulant of interest to us is the fourth:

$$\begin{aligned}\mathbb{E}[z_{\mu_1} z_{\mu_2} z_{\mu_3} z_{\mu_4}]|_{\text{connected}} = \mathbb{E}[z_{\mu_1} z_{\mu_2} z_{\mu_3} z_{\mu_4}] \\ - \mathbb{E}[z_{\mu_1} z_{\mu_2}]\mathbb{E}[z_{\mu_3} z_{\mu_4}] - \mathbb{E}[z_{\mu_1} z_{\mu_4}]\mathbb{E}[z_{\mu_2} z_{\mu_3}] - \mathbb{E}[z_{\mu_2} z_{\mu_4}]\mathbb{E}[z_{\mu_1} z_{\mu_3}]\end{aligned} \tag{25}$$

which for our Gaussian, we can apply Wick's theorem to see that three pairs of variances we subtract off is exactly equal to the fourth moment we have, and so this equals 0. This is our first hint as to what is so useful about these correlators. For Gaussians, all connected correlators higher than the second *vanish*. Thus, a non-zero four point connected correlator indicates a *deviation from Gaussianity*. We are going to analyze our networks in the regime where the deviation from Gaussianity of their outputs is non-zero but small, and so the 4 point correlator will be a useful tool for measuring that.

4 Deep Linear Networks

We are going to start today with the simplest, toy model of a neural network. That is, we are going to take our multilayer perceptron, and remove all of the activation functions, leaving us with just a sequence of linear transformations. We will call this the *deep linear network*.

Our deep linear network will take an input vector x_α from the training set, and we will index the elements of the vector with Roman indices $x_{i;\alpha}$. We will define the *width* of the network at the l-th layer with the number n_l. For the purposes of simplicity, we will also turn off the biases. In case you're getting worried that this toy model is too simple to tell us anything useful, do not fear. a) We will see some interesting physics despite these simplifications, and b) Next week, we will turn on the biases and the activations, and see how that changes things. For now, this is just a warm-up, to get us familiar with the approach we are going to take.

We can write the forward iteration equation of a deep linear network as a recursive sequence of matrix multiplications:

$$z_{i;\alpha}^{(\ell+1)} = b_i^{(\ell+1)} + \sum_{j=1}^{n_\ell} W_{ij}^{(\ell+1)} z_{j;\alpha}^{(\ell)} \tag{26}$$

or

$$z_{i;\alpha}^{(\ell)} = \sum_{j_0=1}^{n_0} \sum_{j_1=1}^{n_1} \cdots \sum_{j_{\ell-1}=1}^{n_{\ell-1}} W_{j_{\ell-1}j_{\ell-2}}^{(\ell-1)} \cdots W_{j_1 j_0}^{(1)} \, x_{j_0;\alpha} \equiv \sum_{j=1}^{n_0} W_{ij}^{(\ell)} \, x_{j;\alpha} \tag{27}$$

Where here we introduce the $n_l \times n_0$ matrix

$$W_{ij}^{(\ell)} = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \cdots \sum_{j_{\ell-1}=1}^{n_{\ell-1}} W_{ij_{\ell-1}}^{(\ell)} W_{j_{\ell-1}j_{\ell-2}}^{(\ell-1)} \cdots W_{j_1 j}^{(1)} \tag{28}$$

We will choose to initialize the weights of the matrices from a mean-zero Gaussian with variance C_W/n_{l-1}.

Somewhat counterintuitively, deep linear networks represent a smaller set of functions than general linear transformations. As an extreme example, take a two-layer deep linear network in which the first hidden layer consists of a single neuron $n_1 = 1$ and consider the network output in the second layer $l= 2$. In this case, all the information in the input is compressed through a bottleneck into a single number in the first layer before being converted into an n_2 dimensional vector in the output layer. Surely, such a deep linear network represents a tinier subspace of linear transformations than those given by all the possible n_2-by-n_0 matrices.

The goal of today is to work out how the distribution of outputs varies from layer to layer in such a network. In particular, even though the weights are drawn from a Gaussian, and in fact, the output distribution of a single layer is thus Gaussian, the output of the *composite* network is patently not Gaussian. We are going to spend the day attempting to characterize the highly non-trivial distribution below.

$$p\left(z^{(\ell)} \mid \mathcal{D}\right) \equiv p\left(z^{(\ell)}(x_1), \ldots, z^{(\ell)}(x_{N_\mathcal{D}})\right) \tag{29}$$

The first thing to note is that for such a product of Gaussians, as long as they are mean 0, the mean of the output distribution at each layer vanishes as well.

$$\mathbb{E}\left[z_{i;\alpha}^{(\ell)}\right] = \sum_{j_0=1}^{n_0} \sum_{j_1=1}^{n_1} \cdots \sum_{j_{\ell-1}=1}^{n_{\ell-1}} \mathbb{E}\left[W_{ij_{\ell-1}}^{(\ell)}\right] \mathbb{E}\left[W_{j_{\ell-1}j_{\ell-2}}^{(\ell-1)}\right] \cdots \mathbb{E}\left[W_{j_1 j_0}^{(1)}\right] x_{j_0;\alpha} \tag{30}$$

$$= 0 \tag{31}$$

In fact, so do all of the odd correlators, and so we only need to worry about the even ones. The next simplest observable is the *two-point correlator*, $\mathbb{E}[z_{\mu_1} z_{\mu_2}]$. We will start with the two-point correlator of the first layer preactivations. Going back to our defining equation for the MLP,

$$z_{i;\alpha}^{(1)} = \sum_{j=1}^{n_0} W_{ij}^{(1)} \, x_{j;\alpha} \tag{32}$$

So,

$$\mathbb{E}\left[z_{i_1;\alpha_1}^{(1)} \, z_{i_2;\alpha_2}^{(1)}\right] = \sum_{j_1=1}^{n_0} \sum_{j_2=1}^{n_0} \mathbb{E}\left[W_{i_1 j_1}^{(1)} W_{i_2 j_2}^{(1)} x_{j_1;\alpha_1} \, x_{j_2;\alpha_2}\right] \tag{33}$$

Now, the dataset is fixed, and so it does not depend on the distribution we are taking the expected value against, so

$$= \sum_{j_1=1}^{n_0} \sum_{j_2=1}^{n_0} \mathbb{E}\left[W^{(1)}_{i_1 j_1} W^{(1)}_{i_2 j_2}\right] x_{j_1;\alpha_1}\, x_{j_2;\alpha_2} \tag{34}$$

Okay, so what is the two point correlator of the two weights? Well, they are drawn from i.i.d. Gaussians, so if $i_1 = i_2$ and $j_1 = j_2$, it should be $\frac{C_W}{n_0}$, and 0 otherwise. To express this in a notationally convenient way, we will introduce the Kronecker delta,

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{35}$$

With this notation in hand,

$$= \sum_{j_1, j_2}^{n_0} \frac{C_W}{n_0} \delta_{i_1 i_2} \delta_{j_1 j_2} x_{j_1;\alpha_1} x_{j_2;\alpha_2} \tag{36}$$

Since our summation doesn't depend on i_1, i_2, we can pull that Kronecker delta out. Secondly, since our summation runs over j_1, j_2, but we have a Kronecker delta which says that only terms with $j_1 = j_2$ contribute, we can collapse this into a single index:

$$\mathbb{E}[z_{i_1;\alpha_1} z_{i_2;\alpha_2}] = \delta_{i_1 i_2} \frac{C_W}{n_0} \sum_{j=1}^{n_0} x_{j_1;\alpha_1} x_{j_2;\alpha_2} \tag{37}$$

That is, for two inputs, α_1, α_2, the two point correlator (and the variance) of the output of the ith neuron is some constant times the dot product of the two inputs. That dot product of the two inputs (under the assumption that the inputs are 0-mean) can be interpreted analogously to their covariance. So, we introduce the matrix K,

$$K^{(0)}_{\alpha_1 \alpha_2} = \frac{1}{n_0} \sum_{j=1}^{n_0} x_{j_1;\alpha_1} x_{j_2;\alpha_2} \tag{38}$$

to represent the *covariance* of the inputs. Now that we have the first layer out of the way, the easiest way to proceed is to compute deeper layers as a recursion.

$$\mathbb{E}\left[z^{(\ell+1)}_{i_1;\alpha_1}\, z^{(\ell+1)}_{i_2;\alpha_2}\right] = \sum_{j_1=1}^{n_\ell} \sum_{j_2=1}^{n_\ell} \mathbb{E}\left[W^{(\ell+1)}_{i_1 j_1} W^{(\ell+1)}_{i_2 j_2} z^{(\ell)}_{j_1;\alpha_1}\, z^{(\ell)}_{j_2;\alpha_2}\right] \tag{39}$$

$$= \sum_{j_1=1}^{n_\ell} \sum_{j_2=1}^{n_\ell} \mathbb{E}\left[W^{(\ell+1)}_{i_1 j_1} W^{(\ell+1)}_{i_2 j_2}\right] \mathbb{E}\left[z^{(\ell)}_{j_1;\alpha_1}\, z^{(\ell)}_{j_2;\alpha_2}\right] \tag{40}$$

$$= \delta_{i_1 i_2} \frac{C_W}{n_l} \sum_{j=1}^{n_\ell} \mathbb{E}\left[z^{(\ell)}_{j;\alpha_1}\, z^{(\ell)}_{j;\alpha_2}\right] \tag{41}$$

$$= C_W K^{(l)}_{\alpha_1 \alpha_2} \tag{42}$$

Notice that at *any layer*, the two point correlator is proportional to $\delta_{i_1 i_2}$. That is, it vanishes unless the two neuronal indices are the same. We also note that this covariance that we have written down scales from layer to layer. In fact, at every layer in our deep linear network, it gets multiplied by another factor of C_W. This is exponential behavior. If $C_W > 1$, the network output's covariance diverges quickly. If $C_W < 1$, it collapses quickly. Only when $C_W = 1$ does the network output remain stable for large depths. This is a baby version of the famous *exploding and vanishing gradient* problem in deep neural networks, and of the oversmoothing problem in graph neural networks.

Of course, this is a toy model. When we insert back our activation functions, we will get much more interesting dynamics than we have with our single hyperparameter. However, that's a discussion for next week. Ta!