CMPTG 5 - Statistical Physics of Neural Networks

Sidharth Kannan

May 2025

1 Higher Point Correlators of Deep Linear Networks

1.1 Recap

Last week, we spent considerable time discussing Gaussian integrals, and derived Wick's probability theorem: For some Gaussian with $\mu = 0$ and covariance matrix K.

$$\mathbb{E}[z_{\mu_1} z_{\mu_2} z_{\mu_3} z_{\mu_4} ... z_{\mu_{2m}}] = \sum_{\text{all pairings of } \mu_i, \mu_j} \prod K_{\mu_i \mu_j} \tag{1}$$

For example, the 4-point correlator is

$$\mathbb{E}[z_{\mu_1} z_{\mu_2} z_{\mu_3} z_{\mu_4}] = K_{\mu_1\mu_2} K_{\mu_3\mu_4} + K_{\mu_1\mu_3} K_{\mu_2\mu_4} + K_{\mu_1\mu_4} K_{\mu_2\mu_3} \tag{2}$$

Recall that z_μ is an index into the random vector z, and $K_{\mu_1\mu_2}$ is the entry in the covariance matrix that corresponds to the μ_1-th row and μ_2-th column. Note how each term contains a distinct way to pair them up.

We applied Wick's theorem to the toy model of a deep linear network, which is a neural network of the form

$$f(x) = W_l W_{l-1} W_{l-2} ... W_1 x \tag{3}$$

where x is the input, and W_i is the matrix representing the i-th layer of the network. We made the assumption that for each matrix, the entries are drawn from an i.i.d. Gaussian with mean 0, $\sigma^2 = \frac{C_W}{n_l}$, where C_W is some user defined parameter, and n_l is the width of the l-th layer.

From there, our goal was to understand what the distribution of the network output is. In the end, we want to understand the *trained* network function, but since it is a function of the initialization distribution, we will start by analyzing the initialization distribution. That is, given a dataset $\mathcal{D} = \{x_i\}$, (or if you prefer, for simplicity, given a data point x), what is the distribution of network output at the l-th layer.

$$P\left(z_i^{(l)} | \mathcal{D}\right) \tag{4}$$

To make that more concrete, consider a single layer network, and a fixed data point. We are interested in understanding the distribution of the random vector $z^{(1)} = W_1 x$, where the stochasticity comes from the fact that the entries of W are drawn from an i.i.d. Gaussian. Now repeat that process for every intermediate layer in an L layer network.

The result that we came to is that for the first layer of the network, the entries in the output are given by independent Gaussians, with variance

$$\mathbb{E}[z_{i_1;\alpha_1} z_{i_2;\alpha_2}] = \delta_{i_1 i_2} \frac{C_W}{n_0} \sum_{j=1}^{n_0} x_{j_1;\alpha_1} x_{j_2;\alpha_2} \tag{5}$$

where again, i is the neuron index (the index into the random vector z), and α indexes the dataset. We introduced the *metric*,

$$K^{(0)}_{\alpha_1\alpha_2} = \frac{1}{n_0} \sum_{j=1}^{n_0} x_{j_1;\alpha_1} x_{j_2;\alpha_2} \tag{6}$$

which is analogous to the covariance between each pair of samples in the dataset, which simplifies our expression for the variance to

$$\mathbb{E}[z_{i_1;\alpha_1} z_{i_2;\alpha_2}] = \delta_{i_1 i_2} C_W K^{(0)}_{\alpha_1\alpha_2} \tag{7}$$

This should make sense, since we have just multiplied this (fixed) vector by a matrix with independent Gaussian entries. Of course the sum will be a Gaussian. The next step was to derive a general recurrence for the distribution in arbitrary layers. We plugged in the recursive definition of a linear network,

$$\mathbb{E}\left[z^{(l+1)}_{i_1;\alpha_1} z^{(l+1)}_{i_2;\alpha_2}\right] = \sum_{j_1,j_2} \mathbb{E}\left[W^{(l+1)}_{i_1 j_1} z^{(l)}_{i_1;\alpha_1} W^{(l+1)}_{i_2 j_2} z^{(l)}_{i_2;\alpha_2}\right] \tag{8}$$

$$= \sum_{j_1,j_2} \mathbb{E}\left[W^{(l+1)}_{i_1 j_1} W^{(l+1)}_{i_2 j_2}\right] \mathbb{E}\left[z^{(l)}_{i_1;\alpha_1} z^{(l)}_{i_2;\alpha_2}\right] \tag{9}$$

$$= \delta_{i_1 i_2} \frac{C_W}{n_l} \mathbb{E}\left[z^{(l)}_{i_1;\alpha_1} z^{(l)}_{i_2;\alpha_2}\right] \tag{10}$$

Thus, we see that at each layer, the variance is multiplied by a constant factor, C_W/n_l, leading to the kind of exponential behavior that leads to either collapse or decay of the variance.

1.2 Higher Point Correlators

This is where we left off. Before we move on, I would like to spend some time justifying some of the simplifying assumptions that we made. The first is that we removed the biases from our networks. This is the easiest to justify, as the major effect of the biases is to make the algebra nastier, without changing any of the qualitative results we will show. We also removed the activation functions. This one is harder to justify, and in fact, we will discuss how adding back the activation functions changes the mathematics we are doing. The general point is though that while the networks we study here represent a much smaller class of functions (just the linear ones) than a true MLP, many of the conclusions we draw will apply to MLPs as well, and for the full derivations of those equations, I refer you to the text.

The last simplification is the one I find most worthy of discussion. We are studying the networks *at initialization*. That is, we are essentially studying products of Gaussian random matrices. Surely by ignoring the training algorithm, we are losing something essential about the trained network

behavior! While this is certainly the case, and we will discuss how to incorporate training into this framework, I will also point out that the training procedure can't fix everything. For example, if the gradients explode or vanish at initialization, then gradient descent can't fix that, since the algorithm itself will be unstable. Thus, we can gain truly meaningful insight into network behavior by studying them at initialization. We will see a particular application of this later in this lecture.

For now, let's move on to the four point correlator. Recall from last lecture that a Gaussian distribution is specified completely by its mean and its covariance matrix. That is, while the four point correlator (fourth moment) may be nonzero, the *connected* four point correlator (fourth cumulant), and all higher cumulants will vanish. In keeping with the text, we will consider the distribution of network outputs for a single input x_α in this section.

Exactly as we did for the variance, let's start by looking at the first layer. The algebra is almost identical to what we did for the variance, so before you move on to this, make sure you understand that.

$$\mathbb{E}[z_{i_1} z_{i_2} z_{i_3} z_{i_4}] = \sum_{j_1,j_2,j_3,j_4=1}^{n_0} \mathbb{E}\left[W_{i_1 j_1} W_{i_2 j_2} W_{i_3 j_3} W_{i_4 j_4}\right] x_{j_1} x_{j_2} x_{j_3} x_{j_4} \tag{11}$$

$$= \frac{C_W^2}{n_0^2} \sum_{j_1,j_2,j_3,j_4=1}^{n_0} (\delta_{i_1 i_2}\delta_{i_3 i_4}\delta_{j_1 j_2}\delta_{j_3 j_4} + \delta_{i_2 i_3}\delta_{i_1 i_4}\delta_{j_2 j_3}\delta_{j_1 j_4} \tag{12}$$

$$+ \delta_{i_1 i_3}\delta_{i_2 i_4}\delta_{j_1 j_3}\delta_{j_2 j_4}) x_{j_1} x_{j_2} x_{j_3} x_{j_4} \tag{13}$$

Wow that's a lot of Kronecker deltas. We evaluated the expectation using Wick's theorem, which says that we can break down this 4 point correlator into sums of products of the covariances. However, we know that for the weights, their covariance is 0 if $i_1 \neq i_2$ or $j_1 \neq j_2$, since the entries of the matrix are drawn from independent Gaussians, and so we represent that through these Kronecker deltas.

If we evaluate the sums over the j indices, we get the following simple expression out

$$= C_W^2 (\delta_{i_1 i_2}\delta_{i_3 i_4} + \delta_{i_1 i_4}\delta_{i_2 i_3} + \delta_{i_1 i_3}\delta_{i_2 i_4}) \cdot \frac{1}{n_0} \sum_{j=1}^{n_0} x_j x_j \tag{14}$$

where we recognize the summation to be the dot product of the input with itself. This is quite similar to the *metric* we identified last week, $K_{\alpha_1 \alpha_2}$, but in this case since we are considering only one data sample, it is a diagonal entry along K.

Then, we have finally that

$$\mathbb{E}[z_{i_1} z_{i_2} z_{i_3} z_{i_4}] = C_W^2 \cdot (\delta_{i_1 i_2}\delta_{i_3 i_4} + \delta_{i_1 i_4}\delta_{i_2 i_3} + \delta_{i_1 i_3}\delta_{i_2 i_4}) \left(K^{(0)}\right)^2 \tag{15}$$

where K is understood to be the diagonal entry along that covariance matrix we were using earlier. Now, let's evaluate the *connected* four point correlator, since this will tell us if the distribution we have is Gaussian or not.

$$\mathbb{E}[z_{i_1} z_{i_2} z_{i_3} z_{i_4}] - \mathbb{E}[z_{i_1} z_{i_2}]\mathbb{E}[z_{i_3} z_{i_4}] - \mathbb{E}[z_{i_1} z_{i_3}]\mathbb{E}[z_{i_2} z_{i_4}] - \mathbb{E}[z_{i_2} z_{i_3}]\mathbb{E}[z_{i_1} z_{i_4}] \tag{16}$$

If you plug in the expression from Eq. 10, you will see that this exactly evaluates to 0. So, we seem to have shown that the preactivation distribution is exactly Gaussian in the first layer. If we

repeat the same exercise as we did last week again, but for the four point correlator, we would now derive a recursion for the fourth moment.

The algebra is quite similar to what we did for the recursion for the two point correlator, so I will leave this as an exercise for the homework. If you're curious, see Eq. 3.20 in the text.

$$\mathbb{E}[z_{i_1}^{(l+1)} z_{i_2}^{(l+1)} z_{i_3}^{(l+1)} z_{i_4}^{(l+1)}] = C_W^2 \cdot (\delta_{i_1 i_2}\delta_{i_3 i_4} + \delta_{i_1 i_4}\delta_{i_2 i_3} + \delta_{i_1 i_3}\delta_{i_2 i_4}) \frac{1}{n_l^2} \sum_{j,k=1}^{n_l} \mathbb{E}\left[z_j^{(l)} z_j^{(l)} z_k^{(l)} z_k^{(l)} \right] \quad (17)$$

$$= C_W^{2l} \cdot (\delta_{i_1 i_2}\delta_{i_3 i_4} + \delta_{i_1 i_4}\delta_{i_2 i_3} + \delta_{i_1 i_3}\delta_{i_2 i_4}) \left[\prod_{l'=1}^{l} \left(1 + \frac{2}{n_{l'}} \right) \right] (K^{(0)})^{2l} \quad (18)$$

The precise details of how one gets from the previous step to this one are again, not terribly relevant. For a better treatment, see the text. The point I would like us to focus on is the physics of what is going on. Notice that this expression *greatly* simplifies in the limit of an infinite number of neurons. Then, the product term vanishes entirely, and we recover our Gaussian 4 point correlator.

This is the first beautiful observation. At initialization, the preactivation distribution in an infinite width network like this is perfectly Gaussian. This result turns out to hold up even when we add the activation functions.

But again, recall that this infinite width limit is rather unrealistic. So, lets see what the deviation from the infinite width case is when the width is finite but large. For simplicity, let's assume all the hidden layers are the same width.

$$\Delta = \left[\left(1 + \frac{2}{n} \right)^{l-1} \right] \left(K^{(l)} \right)^2 - \left(K^{(l)} \right)^2 \quad (19)$$

Taylor expanding in $1/n$, we get

$$= \frac{2(l-1)}{n} \left(K^{(l)} \right)^2 \quad (20)$$

Pause for a moment and think about this. We have shown that in any *arbitrary* layer, the distribution at initialization is approximately Gaussian, as long as the network remains wide. This is what we mean when we discuss scaling. The network distribution's variance is of order 1, while the fluctuations from Gaussianity are order (1/n) or smaller. This justifies the choice to ignore higher moments beyond the second (or sometimes fourth) in our analysis. As long as we are working in the large width limit, we can still capture the qualitative features of our networks.

2 An Application: Oversmoothing in Graph Convolutional Networks

2.1 Graph Basics

A *graph* $\mathcal{G} = (V, E)$ consists of:

- A set of *nodes* (or vertices) $V = \{v_1, \ldots, v_N\}$.

- A set of *edges* $E \subseteq V \times V$, where each edge (v_i, v_j) indicates a relation.

The *adjacency matrix* $A \in \{0,1\}^{N \times N}$ encodes connectivity:

$$A_{ij} = \begin{cases} 1, & (v_i, v_j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

The *degree matrix* D is diagonal with entries $D_{ii} = \sum_j A_{ij}$.

In the context of graph learning, we often consider the *degree normalized adjacency matrix with self loops*. That is, we add self loops to the graph, and then we multiply by the inverse degree matrix.

2.2 Node Feature Vectors

In many tasks, each node has associated features. We assign to node v_i a vector

$$\mathbf{x}_i \in \mathbb{R}^F,$$

where F is the feature dimension. Stacking all node features gives the *feature matrix*

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \in \mathbb{R}^{N \times F}.$$

These features can represent any local information: user attributes in social networks, atom descriptors in molecules, or word embeddings in citation graphs.

2.3 Simple Message Passing Scheme

Graph neural networks often operate by iteratively exchanging information between neighbors. A *message passing* layer updates each node's representation by aggregating neighbors' messages.

Given current node embeddings $H^{(l)} \in \mathbb{R}^{N \times d}$ at layer l, a generic message passing update is:

$$\mathbf{m}_i^{(l)} = \text{AGGREGATE}\big(\{\mathbf{h}_j^{(l)} : j \in \mathcal{N}(i)\}\big),$$

$$\mathbf{h}_i^{(l+1)} = \text{UPDATE}\big(\mathbf{h}_i^{(l)}, \mathbf{m}_i^{(l)}\big),$$

where:

- $\mathcal{N}(i) = \{j : (v_j, v_i) \in E\}$ is the neighborhood of node i.
- AGGREGATE collects neighbor embeddings (e.g., sum, mean, or max).
- UPDATE combines the old embedding with the aggregated message (often via a neural network layer).

A popular choice is the *mean aggregation* with a linear transform:

$$\mathbf{h}_i^{(l+1)} = \sigma\Big(W^{(l)} \cdot \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{h}_j^{(l)} + b^{(l)}\Big),$$

where $W^{(l)}$ and $b^{(l)}$ are learnable parameters and σ is an activation (e.g., ReLU).

This defines one layer of the graph convolutional network (in its most basic form). If we stack many of these layers, we get a graph convolutional network (GCN).

The first thing I will point out is that the same MLP is applied to every node in the graph. This will greatly simplify our analysis. The next thing to note is that each GCN layer increases the "receptive field" of each node by 1 hop. That is, after k layers, information from k hops away in the graph can get to the node in question. So, stacking more layers would allow information to propagate further in the graph, enabling *long range connections*. In principle, this is a good thing. We also know from our experience in other domains that deeper networks are better. They can represent more complex functions, and do cooler things.

This is not the case in graph learning. As you increase the depth of the network, the performance drops *precipitously*. This is due to a phenomenon called oversmoothing, in which after many layers of a GCN, the feature vectors for all of the nodes begin to look the same, and the network loses its ability to discriminate between nodes. We will demonstrate, with our formalism, exactly why this happens.

It's actually quite a simple argument. Going back to equation 10, we said that in our MLP, the variance from layer to layer is multiplied by the constant C_W. In the case of a GCN, we just have to change our forward iteration equation. Instead of

$$z_{i;\alpha}^{(\ell+1)} = \sum_{j=1}^{n_\ell} W_{ij}^{(\ell+1)} z_{j;\alpha}^{(\ell)} \tag{21}$$

we have that

$$z_{i;\alpha}^{(\ell+1)} = \sum_{\text{neighbors}} \sum_{j=1}^{n_\ell} W_{ij}^{(\ell+1)} z_{j;\alpha}^{(\ell)} \tag{22}$$

That is, we change the forward aggregation to reflect the summation over neighbors. This, like the case of the linear network, is a little clearer in matrix notation.

$$z_{i;\alpha}^{(\ell+1)} = W^{(\ell)} \hat{A} Z_\alpha^{(\ell)} \tag{23}$$

where Z is the matrix of *all* of the preactivations, because, recall that we have one for each node now.

Then, we can see, by the same argument as we made with the deep linear network, that the variance of preactivations from layer to layer will go as

$$K^{(l)} = C_W^l \hat{A}^{(l)} K^{(0)} \tag{24}$$

where $K = X^T X$. I am skipping many details in this derivation, since it is almost identical to the result that we get in the deep linear network case. The key point is the variance goes now not as the power of some tunable constant, but as the power of the *adjacency matrix of the graph*.

We can use the fact that the matrix is symmetric, which means the eigenvalues are real. Since it is *degree normalized*, meaning that rows all sum to 1, we can show that the eigenvalues of this matrix are in the range [-1, 1]. Then, we see that again we have an exponential working against us. The spectrum of $\hat{A}^l$ decays rapidly, and so the covariance matrix rapidly decays to a low rank matrix, meaning that all of the node features lie on a low dimensional manifold (i.e. they lose variety).