Random Matrix Initialization Methods

Peter Hoffman, Sage Simhon, Viktor Urvantsev, Mikaeel Yunus

May 21, 2021

Abstract

We study random matrix theory and its applications in machine learning, specifically the initialization methods used when training neural networks. We first provide a literary overview of random matrix theory and its applications in order to motivate a discussion of its mathematical underpinnings. We then design an image classification neural network to test the effectiveness of various initialization methods. We discover that Xavier and He initialization consistently outperform Gaussian initialization in terms of accuracy and speed of convergence.

Keywords: Random Matrix Theory, Applications of Random Matrices, Eigenvalue Density, Random Matrix Ensemble, Initialization Methods, He Initialization, Xavier Initialization, Wigner Matrices

Contents

1 Introduction

Random Matrix Theory seeks to provide a theoretical underpinning to the properties that arise from matrices whose entries are drawn from probability distributions. Many of these properties have numerous applications in a wide range of fields: notably statistics, computational physics, and most recently machine learning.

A single random matrix is a realization of the random matrix ensemble from which it is drawn. Two important ensembles studied in this paper are the Gaussian Orthogonal Ensemble (GOE) and the Gaussian Unitary Ensemble (GUE), which are defined over the set of real symmetric and complex Hermitian matrices, respectively. The matrix entries for both ensembles are independently and identically distributed (i.i.d.) according to a Gaussian distribution centered at 0 and whose variances are chosen so that their joint probability densities are invariant over Orthogonal and Unitary similarity transformations.

While the GOE and GUE are the two ensembles discussed in this paper, several others also appear often in literature. One example is the Circular Ensemble, which is composed of complex unitary matrices with the same invariance properties as the Gaussian examples above, yet differ in that their eigenvalues are confined to the unit circle in the complex plane rather than to the real line.

All random matrix ensembles introduced above have several interesting properties, especially the asymptotic behavior of their eigenvalues. It is these same properties that make random matrices especially relevant to machine learning.

2 Literary Overview

Given that random matrices have random-valued entries, it should be unsurprising that they are often found when dealing with natural processes: the world after all – from disease transmission to stock price fluctuations – is an undoubtedly random place.

Therefore, it is only natural that the importance of random matrices is tied to its wide range of applications. This is true especially when considering the extent to which various fields of computation attempt to model the world for the purpose of prediction and understanding. Data analysis and statistical models are both obvious applications, but random matrices also play a critical role in the physical sciences such as nuclear and particle physicals [1].

Random matrices saw their first concrete application in John Wishart's 1928 work in the field of mathematical statistics [2]. However, the importance of random matrices in the study of the physical world date back to Eugene Wigner's particle physics research in the 1950's [3]. When studying the energy levels of heavy nuclei, Wigner was unable to generate closed form solutions for calculations involving interactions of complex forces. This is because quantum mechanical forces are the primary components in systems at the microscopic level, which are notoriously difficult to observe and require immense computing power to do so. Instead, Wigner turned to random matrix theory to infer general properties of the system [4]. More specifically, Wigner used properties of the eigenvalues of random matrices to infer properties of the energy spectra of the atoms he was studying.

In the years that followed, random-matrix theory became increasingly prominent in early discoveries of quantum chaos (Bohigas, Giannoni, and Schmit, 1984), which led to a random-matrix theory of quantum transport [5][6]. Even today, random matrices are instrumental tools in various areas of mathematics such as multivariate statistical analysis and principal components analysis [7][8]. As Izenman notes, random matrices are applicable in many situations that require an "indirect method for solving complicated problems arising from physical or mathematical systems" [9].

Many of these discoveries relied on the Universality Conjecture, which states that local behavior in the eigenvalues of large random matrices have limits that are independent of the probability distribution of the matrix ensembles [9]. While it is well known that the Universality Conjecture doesn't hold in general, researchers instead constrain themselves to working with certain families of probability distributions for which it does hold. These probability families are closely related to the types of random matrix ensembles, but more on this in Section 2.

In addition to their multitude of applications, random matrices are an exciting mathematical object because of the wide range of tools used to study them. As Anderson et al. point out, Wigner, Dyson, Mehta and collaborators formulated theories on the spectrum of random matrices as far back as the 1960s using tools found in number theory and numerical analysis. Other early advances relied on enumerative combinatorics, Fredholm determinants, and diffusion processes [1].

3 Random Matrices and their Properties

A random matrix is a matrix A such that each entry a_{ij} is a random variable. Several interesting properties arise from this randomness, and this section provides a general overview of these key theories and observations.

3.1 Wigner Matrices

There are two classes of Wigner matrices, the first being real Wigner matrices [10]. For $1 \leq i < j \leq \infty$, let $X_{i,j}$ be i.i.d. (real) random variables with $\mu = 0$ and $\sigma = 1$ and set $X_{j,i} = X_{i,j}$. Let $X_{i,i}$ also be i.i.d. real random variables with $\mu = 0$ and $\sigma = 1$, yet with possibility of being from a different distribution. Then $M_n = [X_{i,j}]_{i,j=1}^{n}$ will be a random $n \times n$ symmetric matrix known as a real Wigner matrix.

The second class of Wigner matrices are complex Wigner matrices [1]. These can be formed from letting $X_{i,j}$ be i.i.d. complex random variables with $\mu = 0$ and $\mathbb{E}[X_{i,j}]^2 = 1$ and $X_{j,i} = \bar{X_{i,j}}$. Similarly, $X_{i,i}$ will be i.i.d. real random variables with $\mu = 0$ and $\sigma = 1$. Then $M_n = [X_{i,j}]_{i,j=1}^n$ will be a random $n \times n$ Hermitian matrix known as a complex Wigner matrix.

For random matrices, the eigenvalues $\lambda_1, \lambda_2 ... \lambda_n$ will be random variables themselves since they depend on the random entries of said matrix.

3.2 Gaussian Ensembles

Each time we recreate a random matrix with values drawn from a Gaussian distribution with density

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

we are recreating an instance of the Gaussian ensemble. Of all Gaussian ensembles, there are two that are particularly important: the Gaussian Orthogonal Ensemble and the Gaussian Unitary Ensemble.

The Gaussian Orthogonal Ensemble GOE(n) is a real Wigner Matrix whose entries are distributed according to the family of Gaussian distributions, where $X_{i,j} \sim N(0,1)$ and $X_{i,i} \sim \sqrt{2}N(0,1)$. However, it can also be constructed by letting $a_{i,j}$ be i.i.d standard normals and $A_n = [a_{i,j}]_{i,j=1}^n$ for $i, j \in \mathbb{Z}$, with $M_n = \frac{A_n + A_n^T}{\sqrt{2}}$ being a GOE(n). As the name might imply, the GOE(n) is invariant under orthogonal conjugation. Another useful property is the joint eigenvalue density, which is given by

$$f(\lambda_1 ... \lambda_n) = \frac{1}{Z_1} \prod_{i<j} |\lambda_j - \lambda_i| e^{-\frac{1}{4}\sum_{i=1}^n \lambda_i^2}$$

where Z_1 is a normalizing constant that depends on n.

A second example of a Gaussian Ensemble is the Gaussian Unitary Ensemble GUE(n), which is a complex Wigner Matrix with $X_{i,j}$ distributed according to a standard complex Gaussian $X_{i,j} \sim N(0, \frac{1}{2}) + iN(0, \frac{1}{2})$ and $X_{i,i} \sim N(0,1)$. The resulting random hermitian matrix is called Gaussian Unitary Ensemble (or GUE). As the name might imply, the GUE(n) is invariant under unitary conjugation.

Similarly to the GOE(n), there is a second construction that is worth noting. Let $a_{i,j}$ be i.i.d standard complex Gaussians and $A_n = [a_{i,j}]_{i,j=1}^n$ for $i, j \in \mathbb{Z}$, then $M_n = \frac{A_n + A_n^*}{\sqrt{2}}$ is the GUE(n).

The joint eigenvalue density is given by

$$f(\lambda_1 ... \lambda_n) = \frac{1}{Z_2} \prod_{i<j} |\lambda_j - \lambda_i| e^{-\frac{1}{2}\sum_{i=1}^n \lambda_i^2}$$

where Z_2 is a normalizing constant that depends on n.

3.3 Eigenvalues of Random Matrices

For a random matrix A of size $N \times N$, there are $K = 2^{N(N+1)/2}$ matrices, where each occurs with probability $K^{-1} = 2^{-N(N+1)/2}$. A natural question is: What is the typical behaviour of the eigenvalues?

Let's start by limiting ourselves to symmetric matrices, so that the eigenvalues are real. Let's also assume a distribution from which the a_{ij} are drawn from: an elementary choice might be Bernoulli $a_{ij} \in \{0, 1\}$ with $a_{ij} = a_{ji}$ for all i, j.

Some of these K matrices will indeed have eigenvalues that are abnormal, such as the matrix with $a_{ij} = 1 \; \forall \; i, j$ which will only have two unique eigenvalues, N and 0. These aberrations are more common when N is small, and diminish in probability as N becomes large. For example, **Figure 1** shows a histogram of the 8 eigenvalues of a random matrix with $N = 8$. It is clear that there is no discernible pattern for this small N. However, a pattern arises when $N = 1000$, as shown in **Figure 2**.

This demonstrates a key property of random matrices, known as Wigner's Semicircle Law, which states that the distribution of eigenvalues for a symmetric random matrix converge to Wigner's semicircle as $N \to \infty$. This convergence implies that the randomness of the matrix's eigenvalues goes to zero as N becomes large,

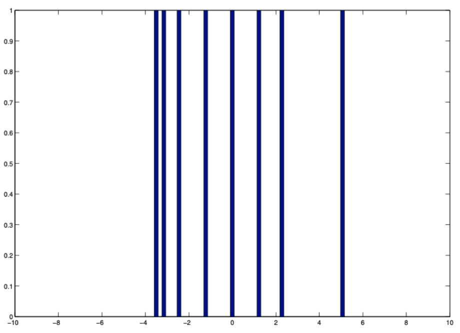

Figure 1: Eigenvalue histogram for $A_{n=8}$ [11]

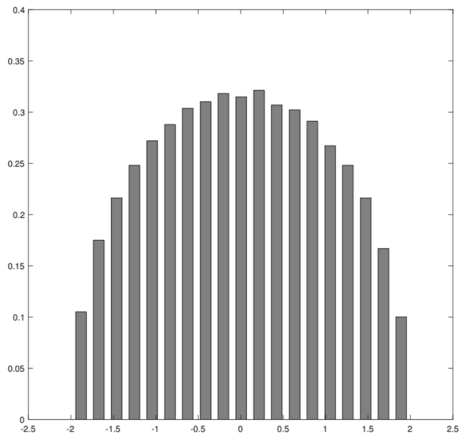

Figure 2: Eigenvalue histogram for $A_{n=1000}$ [11]

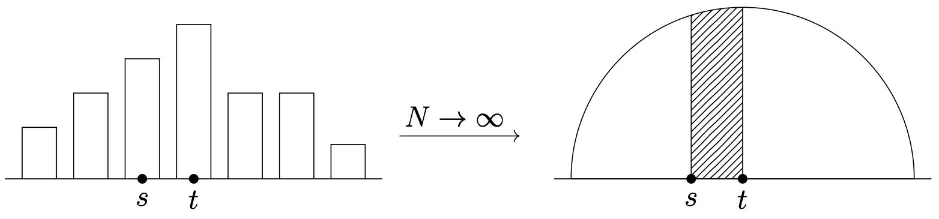

Figure 3: Eigenvalue histogram converges to Wigner's semicircle [11]

and that the asymptotic distribution of the eigenvalues is a deterministic object as opposed to a random one. What's interesting is that this also holds for many distributions of a_{ij}, not just the Bernoulli case we have examined thus far.

With a sufficiently large N, we can study the continuous moments of these random eigenvalues as opposed to their discrete histograms. This method treats Wigner's circle as a distribution W with density $\rho_W(x)$. This convergence in the distribution of eigenvalues is depicted in **Figure 3** and allows us to write:

$$\frac{\#\{\text{eigenvalues in } [s,t]\}}{N} \to \int_s^t \mathrm{d}W = \int_s^t \rho_W(x)\mathrm{d}x$$

We have not given a formula for distribution W, but one might notice that from the above convergence we can also express the distribution as

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{[s,t]}(\lambda_i) \xrightarrow{N\to\infty} \int_{\mathbb{R}} \mathbb{1}_{[s,t]}(x)\,\mathrm{d}W(x)$$

Where the indicator function $\mathbb{1}_{[s,t]}(x) = 1$ if $x \in [s,t]$ and is 0 otherwise. This is more tractable when written as

$$\frac{1}{N}\sum_{i=1}^{N} f(\lambda_i) \xrightarrow{N\to\infty} \int_{\mathbb{R}} f(x)\;\mathrm{d}W(x)$$

where f takes the form $f(x) = x^n$:

$$\frac{1}{N}\sum_{i=1}^{N} {\lambda_i}^n \xrightarrow{N\to\infty} \int_{\mathbb{R}} x^n\;\mathrm{d}W(x)\ [11].$$

In the above expression, the latter are the moments of W, the probability distribution of the random eigenvalues.

However, in order for this convergence to hold as $N \to \infty$ we must choose the proper scaling factor [12]. Let's assume that $a_{ij} \in \{-1, 1\}$ and $A_N = A_N^*$ so that we can express the trace of the square of the random matrix according to the following

$$\mathrm{tr}(A_N^2) = \frac{1}{N}\sum_{i,j=1}^{N} a_{ij}a_{ji} = \frac{1}{N}\sum_{i,j=1} a_{ij}^2 = \frac{1}{N}N^2 = N$$

We need this trace to converge as $N \to \infty$, therefore we should re-scale each random matrix as

$$A_N \Rightarrow \frac{1}{\sqrt{N}}A_N$$

so that $\mathrm{tr}(A_N) \xrightarrow{N\to\infty} \int x^n \mathrm{d}W(x)$.

We have shown how one can model the distribution of random eigenvalues for sufficiently large N using the fact that W converges to a deterministic distribution. The next section will provide additional detail on the explicit form of the semi-circle distribution.

3.4 The Semi-Circle Distribution

Now that we know the moments of the probability distribution, we can define its exact distribution.

First, it is important to note that the standard semi-circular distribution W spans the interval $[-2, 2]$ and has a density given by $\mathrm{d}W(x) = \frac{1}{2\pi}\sqrt{4 - x^2}\mathrm{d}x$.

Second, recall that the Catalan Numbers $(C_k)_{k\geq 0}$ are given by

$$C_k = \frac{1}{k+1}\binom{2k}{k}.$$

As Prof. Dr. Roland Speicher proves in full, the semi-circular distribution W is a probability measure such that

$$\frac{1}{2\pi}\int_{-2}^{2}\sqrt{4-x^2}\ \mathrm{d}x = 1$$

and its moments are given by

$$\frac{1}{2\pi}\int_{-2}^{2} x^n\sqrt{4-x^2}\ \mathrm{d}x = \begin{cases} 0 & n \text{ is odd} \\ C_n & \text{otherwise} \end{cases}.$$

4 Random Matrices in Machine Leaning

Aside from their classical applications discussed in Section 3, random matrices play a central role in machine learning. Applications are wide ranging, such as self-regularization in deep neural networks, image compression using Markov field theory, and loss surface Hessians of artificial neural networks [13][14].

4.1 Modeling High Dimensional Data

Perhaps the best way to introduce random matrices' inextricable ties to machine learning is by discussing the fundamental task of modeling the data that ML models run on. More specifically, random matrices enable the projection of high-dimensional data onto low-dimensional spaces, as is often required when visualizing the high-dimensional data that is used in state-of-the-art machine learning applications. This can be done using the Johnson-Lindenstrauss Lemma, which states that for any set of N points in high-dimensional space, there exists a map to a much lower dimensional space of size $\log(N)$ so that the pairwise distance between any two given points decreases at most by a controllable parameter. This mapping can be constructed by writing a matrix whose values are drawn from a Gaussian distribution. This method is known as random projections and is a key aspect of machine learning today.

4.2 Weight Initialization in Machine Learning

Random matrices also play a crucial role in the initialization of neural networks. A significant amount of research has been dedicated to studying the initialization of the nonlinear bias function b in an arbitrary neural network layer, but less effort has been directed towards discovering optimal methods for weight matrix initialization. In fact, initializing weight matrices is quite a nontrivial task, as illustrated below.

As an extreme case, consider a neural network layer whose weights are initialized to the zero matrix. Since this would cause perfect symmetry to exist between each neuron in the layer, this would prevent the neural network from training properly. But in a less extreme case, even if the weights of this layer are initialized deterministically and as nonzero values, network training often runs into undesired issues such as exploding and vanishing gradients, which results in loss divergence and/or impractical slowdown in training time. We can notice that the final activation layer a^L (where L is the number of layers) is produced by a series of products of weight matrices from each previous layer W^l:

$$a^L = \prod_{i=0}^{L-1} W^{L-i} = W^L W^{L-1} W^{L-2} \dots W^3 W^2 W^1$$

In fact, backpropagation, which is used to find the gradients of the loss function with respect to the weight parameters, also relies on a similar product of many weight matrices. If the weights are arbitrarily initialized even just above or just below the identity matrix, the large series of multiplications in backpropagation will lead to a product that grows or shrinks exponentially (respectively) with L, hence the terms *exploding* and *vanishing gradients*. An exploding gradient makes it difficult for gradient descent to converge to the global minimum of the loss function – since the steps in training are too large, they end up oscillating around the global minimum. With too large a step size, divergence is a possible risk as well. A vanishing gradient faces the opposite problem – that is, either the small step size causes a major, impractical slowdown in the network, or loss convergence occurs too quickly and the global minimum in the loss function is again not reached. Ultimately, inappropriate weight initialization leads to significant reductions in the performance of a neural network [15].

However, if we use random matrices to initialize the weights of our neural network, we can reliably and effectively break the symmetry that causes zero initialization to fail, as well as mitigate the risk of exploding and vanishing gradients. With random weight initialization, we can express each individual neuron in a neural network layer as a unique linear combination of its inputs due to the random nature of the relevant coefficients.

Because of the above reasons, initializing the weights of a neural network using random matrices is now commonplace in modern machine learning. However, most weights matrices are initialized using a standard normal distribution that is centered around zero and has a variance of one. Ideally, when moving from layer to layer, we would make adjustments to the underlying weights matrix distributions in order to avoid exploding and vanishing gradients. Specifically, we can combat the issue of exploding and vanishing gradients, thereby improving training speed and accuracy, by ensuring that the means and variances of neurons across multiple layers do not change. In mathematical terms,

$$\mathrm{E}[a^{[l-1]}] = \mathrm{E}[a^{[l]}] \tag{1}$$

$$\mathrm{Var}[a^{[l-1]}] = \mathrm{Var}[a^{[l]}] \tag{2}$$

where $a^{[l]}$ is the activation of the l^{th} layer of a neural network [15][16][18].

There are two important weight initialization algorithms that facilitate the properties above, namely Xavier and He initialization. These methods are discussed at length in the following sections.

4.3 Xavier Initialization

In the past, neural networks have relied on sigmoid or hyperbolic tangent activation functions in their hidden layers. The purpose of Xavier initialization is to ensure that equations (1) and (2) above are satisfied for neural networks with sigmoid or hyperbolic tangent activation functions [16][25]. This will enable not only faster training, but also an increase in neural network accuracy as discussed above.

We know that for all layers l we must have the following:

$$\mathrm{Var}[a^{[l-1]}] = \mathrm{Var}[a^{[l]}]$$

Because all weight and input distributions are centered at zero, and sigmoid and hyperbolic tangent functions are approximately linear around zero, we can write

$$\mathrm{Var}[a^{[l-1]}] \approx \mathrm{Var}[z^{[l]}]$$

where

$$z^{[l]} = \sum_{i=1}^{n^{[l-1]}} w_i^{[l]} a_i^{[l-1]}$$

is the output of layer l before the activation function is applied: the pre-activation. Thus,

$$\begin{aligned}\mathrm{Var}[a^{[l-1]}] &\approx \mathrm{Var}\left[\sum_{i=1}^{n^{[l-1]}} w_i^{[l]} a_i^{[l-1]}\right] \\ &= \sum_{i=1}^{n^{[l-1]}} \mathrm{Var}[w_i^{[l]} a_i^{[l-1]}]\end{aligned}$$

since for any two independent random variables X and Y, $\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \mathrm{Var}[Y]$. We also know that for any two independent random variables X and Y, $\mathrm{Var}[XY] = \mathrm{E}[X]^2\mathrm{Var}[Y] + \mathrm{E}[Y]^2\mathrm{Var}[X] + \mathrm{Var}[X]\mathrm{Var}[Y]$ [16]. Thus, we have

$$\mathrm{Var}[a^{[l-1]}] \approx \sum_{i=1}^{n^{[l-1]}} \mathrm{E}[w_i^{[l]}]^2\mathrm{Var}[a_i^{[l-1]}] + \mathrm{E}[a_i^{[l-1]}]^2\mathrm{Var}[w_i^{[l]}] + \mathrm{Var}[w_i^{[l]}]\mathrm{Var}[a_i^{[l-1]}]$$

Because the weights are all centered around zero, the first two terms on the right-hand side vanish:

$$\begin{aligned}\mathrm{Var}[a^{[l-1]}] &\approx \sum_{i=1}^{n^{[l-1]}} \mathrm{Var}[w_i^{[l]}]\mathrm{Var}[a_i^{[l-1]}] \\ &= n^{[l-1]}\mathrm{Var}[W]\mathrm{Var}[a^{[l-1]}]\end{aligned}$$

The $\mathrm{Var}[a^{[l-1]}]$ terms on each side cancel each other out, and after some rearranging, we are left with the following:

$$\boxed{\mathrm{Var}[W] = \frac{1}{n^{[l-1]}}} \tag{3}$$

This is the Xavier initialization equation. If we initialize the weights matrices of a neural network with sigmoid or hyperbolic tangent hidden-layer activation functions such that Equation (3) is satisfied, we should be able to speed up network training and significantly increase the accuracy of our model [16][25].

4.4 He Initialization

Modern neural network architectures tend to use ReLU activation functions in their hidden layers, rather than sigmoids or hyperbolic tangents. The purpose of He initialization is to speed up the training and increase the accuracy of a neural network that uses ReLU activation functions in its hidden layers [16][26].

We obviously cannot use the linearity-around-zero argument that we used in the case of Xavier initialization, because ReLU functions have a discontinuity at zero. Instead, we will first prove the following lemma from [20].

Lemma 1. *If* $a^{[l]} = \mathrm{ReLU}(z^{[l]}) = \max(0, z^{[l]})$*, then*

$$\mathrm{E}[(a^{[l]})^2] = \frac{1}{2}\mathrm{Var}[z^{[l]}]$$

Proof.

$$\begin{aligned}\mathrm{E}[(a^{[l]})^2] &= \int_{\mathbb{R}} a^{[l]} p(z^{[l]}) dz^{[l]} \\ &= \int_{\mathbb{R}} \max(0, z^{[l]})^2 p(z^{[l]}) dz^{[l]} \\ &= \int_0^{\infty} (z^{[l]})^2 p(z^{[l]}) dz^{[l]} \\ &= \frac{1}{2}\int_{\mathbb{R}} (z^{[l]})^2 p(z^{[l]}) dz^{[l]}\end{aligned}$$

Since the weights and inputs are still centered around zero, we can say that $\mathrm{E}[z^{[l]}] = 0$. Using this to our advantage, we can rewrite the above as

$$\begin{aligned}\mathrm{E}[(a^{[l]})^2] &= \frac{1}{2}\int_{\mathbb{R}} (z^{[l]} - \mathrm{E}[z^{[l]}])^2 p(z^{[l]}) dz^{[l]} \\ &= \frac{1}{2}\mathrm{E}[(z^{[l]} - \mathrm{E}[z^{[l]}])^2] \\ &= \frac{1}{2}\mathrm{Var}[z^{[l]}]\end{aligned}$$

The last two lines above result from the definitions of expected value and variance.

□

Let us now derive the He initialization formula, starting from equations (1) and (2) at the end of Section 4.2. We have the following:

$$\begin{aligned}\mathrm{Var}[a^{[l-1]}] &= \mathrm{Var}[a^{[l]}] \\ &= \mathrm{E}[(a^{[l]})^2] - (\mathrm{E}[a^{[l]}])^2\end{aligned}$$

due to the definition of variance. Since all weights and inputs are centered around zero, the second term above vanishes. Thus, we have

$$\begin{aligned}\mathrm{Var}[a^{[l-1]}] &= \mathrm{E}[(a^{[l]})^2] \\ &= \frac{1}{2}\mathrm{Var}[z^{[l]}]\end{aligned}$$

as a result of Lemma 1. Now we can follow the same exact steps that we took in the Xavier initialization derivation in Section 4.3, while carrying this new factor of $\frac{1}{2}$ in front of the right-hand side. We eventually arrive at the following:

$$\mathrm{Var}[a^{[l-1]}] = \frac{1}{2} n^{[l-1]} \mathrm{Var}[W] \mathrm{Var}[a^{[l-1]}]$$

which leads us to the He initialization equation:

$$\boxed{\mathrm{Var}[W] = \frac{2}{n^{[l-1]}}} \tag{4}$$

As before, if we initialize the weights matrices of a neural network with ReLU hidden-layer activation functions such that Equation (4) is satisfied, we should be able to speed up network training and significantly increase the accuracy of our model [16][26].

5 Comparing Different Random Matrix Initialization on a NN

We implement various initialization methods on two different neural networks – a three-layer feed forward network and a convolutional neural network – in order to evaluate each initialization method's convergence time and overall accuracy. We use the following weight initializations for each network in order to classify MNIST digits:

1. Normal Initialization (control)
2. Xavier Uniform
3. Xavier Normal
4. He Uniform
5. He Normal

As shown above, we create two instances of each type of advanced initialization (Xavier and He): one that uses an underlying uniform distribution and one that uses an underlying normal distribution. Both are centered at zero and have a variance of $\frac{1}{n^{[l-1]}}$ (for Xavier initialization) or $\frac{2}{n^{[l-1]}}$ (for He initialization). This is done to see how the underlying distribution makes a difference in training and testing for each of the two networks we designed.

For training, we use negative log likelihood (NLL) as our loss function, since it's a tried and true method of accentuating model certainty in image recognition techniques. This loss of each initialization method is stored for each epoch of training that we do, generally 10 epochs with one exception that we will detail in the sections below. At the end of the training, this is displayed, where the average training loss of each initialization method is graphed over the number of training epochs. In addition, we also show a few indicative accuracy training results, where the test image, model name, and model guess probabilities are shown in each entry.

5.1 Data

For our training and testing data, we used the popular MNIST dataset, given its accessibility and its wide use in machine learning applications. Each image in the MNIST dataset is a 28 x 28 matrix of integers valued between 0 and 255. Thus, each of our inputs for the networks start at a width of 784. For each session, a model is initialized, and the randomized weights from either a normal, Xavier or He initialization are applied. From there, over several epochs, images are flattened and used to train the model, and the overall loss is monitored in each epoch as the model is trained. In our suite, this is done for normal initialization, Xavier normal initialization, Xavier uniform initialization, He normal initialization, and He uniform initialization.

5.2 Three-layer Feed-Forward Network

The first of our two models used for experimentation was a simple three layer feed forward network with the following architecture:

A fully connected layer with 784 input units and 128 output units
A ReLU activation layer
A fully connected layer with 128 input units and 64 output units
A ReLU activation layer

A fully connected later with 64 input units and 10 output units
A final softmax to express probabilities of each possible result given the input

Training with ten epochs and a learning rate of 0.003 gives the following training loss for each of the different initialization schemes:

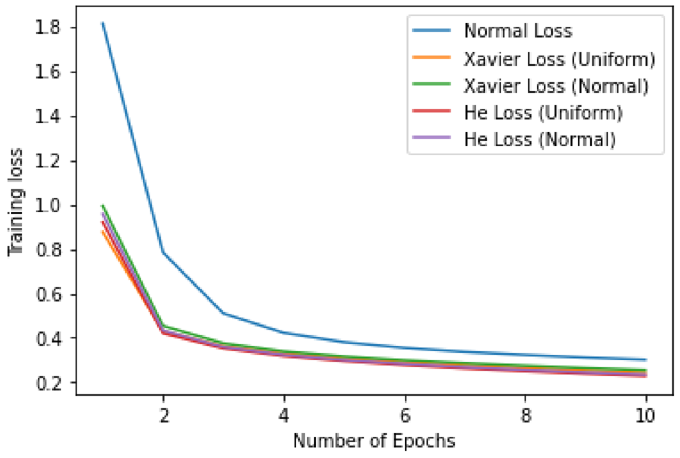

Figure 4: Training Results for the Five Initializations: Loss

As **Figure 4** shows, the normal initialization lags well behind the Xavier and He initialization, but the advanced initialization methods are just about equivalent after 10 epochs of training, with the He loss functions slightly leading the Xavier loss functions. What this means in more tangible terms is that all advanced models tend to predict with similar levels of certainty on the correct solution given test data, but the He initializations tend to yield better correctness and certainty than the Xavier initializations. And both blow the normal initialization out of the water.

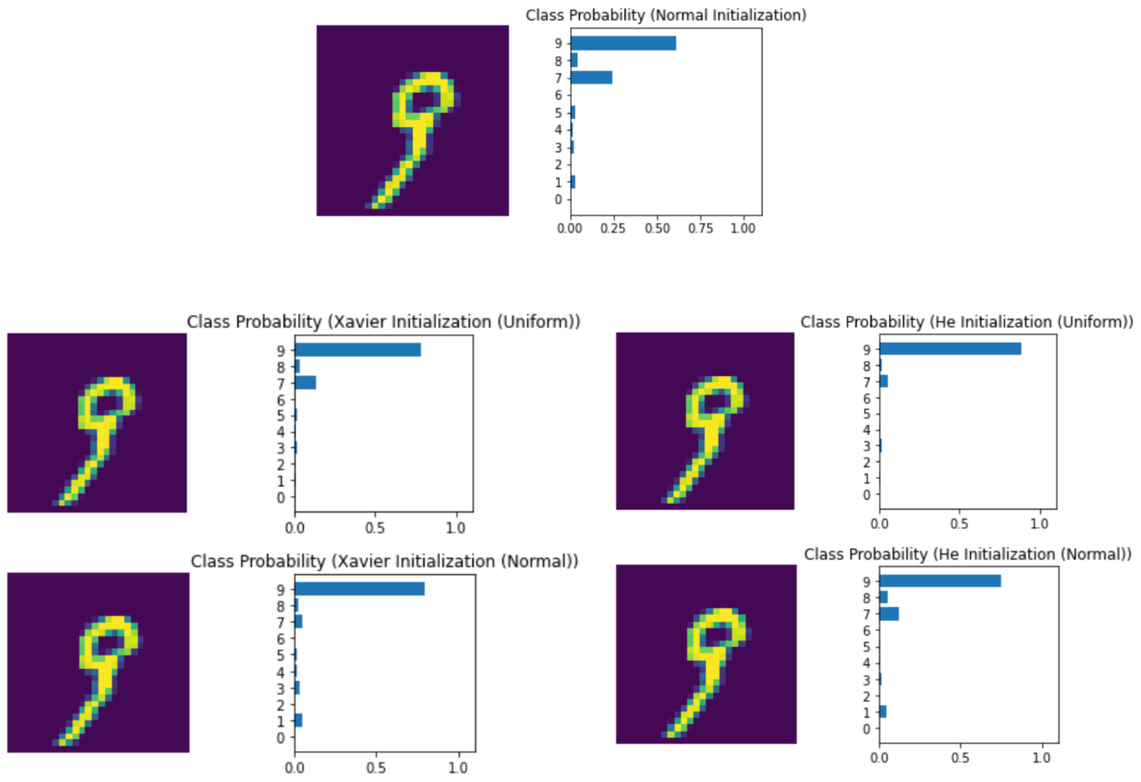

Figure 5: Training Results for the Five Initializations: Accuracy

In **Figure 5**, we show the accuracy of the models – in other words, the probability that each model yields the correct classification. The most important takeaway is that both variants (uniform and normal) of Xavier and He initialization consistently yield a more accurate classification than normal initialization.

5.3 Convolutional Network

We next implement a deeper network with convolutional layers. Convolutional Neural Networks, CNNs, have advantages over feed-forward networks in image classification because they can consider the spatial locality of features. The following network architecture is used [17]:

A convolutional layer (without padding) with 32 filters of size 3 × 3
A ReLU activation layer
A max pooling layer with size 2 × 2
A convolutional layer (without padding) with 64 filters of size 3 × 3
A ReLU activation layer
A max pooling layer with size 2 × 2
A flatten layer (will flatten to size 1600 = 5 × 5 × 64)
A fully connected layer with 128 units
A ReLU activation layer
A dropout layer with drop probability 0.5
A fully-connected layer with 10 output units
(The above architecture is copied from 6.036's MNIST classifier model in a homework assignment [17]).

Again, we finish with a final softmax layer, included in the cross-entropy loss function, which for our purposes is the same as the negative log-likelihood (NLL) Loss.

Training with stochastic gradient descent, ten epochs, and a learning rate of 0.003 gives the following results for the different initialization schemes:

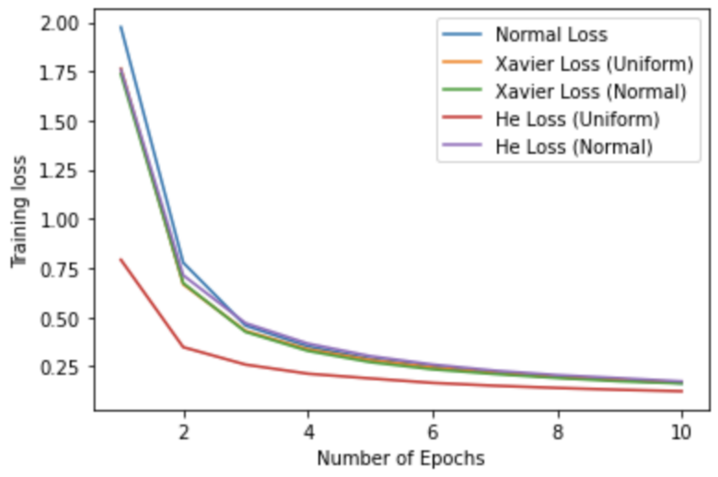

Figure 6: Training Results for the Five Initializations

As shown in **Figure 6**, there is a noticeable advantage in the He Uniform loss unlike in the non-convolutional network. It begins with a lower training loss, and remains strictly below the other loss curves, resulting in a lower final average error after 10 epochs. The other losses do converge more steeply, however. While it is already known and theoretically understood that He initialization performs better for ReLU activations, it is not exactly clear why He has performed better in this convolutional network, considering both networks considered only use ReLU activations.[28] The most obvious conclusion is that the CNN is deeper and has one additional ReLU layer in its architecture. However, performing significantly better than the other initializations due to just one additional ReLU layer is suspicious, and our experiment raises some interesting questions about the higher performance of He Uniform initialization raises a few questions: Why did the He Normal initialization not see the same advantages as the He Uniform? How significant is one additional ReLU layer in distinguishing between performance of Xavier and He initialization?

6 Conclusion

Random matrices have an immense number of applications, especially in statistics and the physical sciences. As we have shown in this paper, they are responsible for facilitating discoveries relating to physics and play central roles in commercial applications such as mobile communication systems.

The number and diversity of these application is in part due to the interesting properties that random matrices exhibit. In this paper, we focused on the distribution of eigenvalues, showing that this distribution converged to a deterministic object known as Wigner's semi-circle as the size of the matrix went to infinity. We then designed an experiment that implemented two neural networks using Xavier and He weight initialization. We trained these models on MNIST data and studied the negative log-likelihood loss function in order to evaluate the accuracy of the predictions.

We learned that the advanced random initialization techniques blew normal initialization out of the water in terms of both minimizing loss and maximizing accuracy, and did so performantly, in fewer epochs than standard normal initialization. The results reflect this across the board. We also saw that He initialization outperformed Xavier initialization in the context of CNNs, which, given the relatively small scope of our experiment, was a surprising phenomenon to observe, despite its agreement with the theoretical predictions. In the other experiment, Xavier and He were approximately neck and neck.

Neural networks obviously have a plethora of applications in the real world, but the convention in modern data science is to initialize weights matrices using standard Gaussian distributions with mean zero and variance 1. As we have shown in this paper, Xavier and He initializations tend to make neural network training not only faster but also more accurate. In the MNIST study that we conducted, the difference between the pure normal initialization and the Xavier and He initializations may not have been extreme, but for a deep neural network designed to solve a much more complex problem (such as facial recognition), the speedups that result from Xavier and He initialization could go a long way. Not only would they save time, but in an era where it is common to use interfaces such as the Google Cloud Platform or Amazon AWS to pay for computing resources, these advanced initialization methods could significantly lower the cost of training a deep neural network. They say that time is money, after all.

References

[1] Anderson, G.; Guionnet, A.; Zeitouni, O. (2005). An Introduction to Random Matrices. Cambridge University Press.

[2] Wishart, J. (1928). "Generalized product moment distribution in samples". Biometrika. 20A (1–2): 32–52. doi:10.1093/biomet/20a.1-2.32.

[3] Wigner, E. (1955). "Characteristic vectors of bordered matrices with infinite dimensions". Annals of Mathematics. 62 (3): 548–564. doi:10.2307/1970079. JSTOR 1970079.

[4] Miller, S.; Takloo-Bighash, R. (2006). An invitation to modern number theory. Princeton University Press.

[5] Bohigas, O.; Giannoni, M.J.; Schmit, Schmit (1984). "Characterization of Chaotic Quantum Spectra and Universality of Level Fluctuation Laws". Phys. Rev. Lett. 52(1): 1–4. Bibcode:1984PhRvL..52....1B. doi:10.1103/PhysRevLett.52.1.

[6] Beenakker, C. W. J. (1997, July 1). Random-matrix theory of quantum transport. Reviews of Modern Physics. https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.69.731.

[7] Tulino, A. M.; Verdú, S. (2004, June 27). Random Matrix Theory and Wireless Communications. Foundations and Trends in Communications and Information Theory. https://www.nowpublishers.com/article/Details/CIT-001.

[8] Johnstone, I. M. (2001). On the distribution of the largest eigenvalue in principal components analysis. The Annals of Statistics. https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-2/On-the-distribution-of-the-largest-eigenvalue-in-principal/10.1214/aos/1009210544.full

[9] Izenman, A. J. (n.d.). Introduction to Random-Matrix Theory. https://www.asc.ohio-state.edu/statistics/dmsl/Introduction_to_Random_Matrix_Theory.PDF

[10] Kemp, T. (n.d.). MATH 247A: INTRODUCTION TO RANDOM MATRIX THEORY.

[11] Speicher, R. (2020, September 10). Random Matrices. https://arxiv.org/pdf/2009.05157.pdf.

[12] Valko, B. (n.d.). Lecture 1: Basic Random Matrix Models. https://people.math.wisc.edu/~valko/courses/833/2009f/lec_01.pdf.

[13] Martin, C. H.; Mahoney, M. W. (2018, October 2). Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning. arXiv.org. https://arxiv.org/abs/1810.01075.

[14] Baskerville, N. P.; Granziol, D.; Keating, J. P. (2021, February 12). Applicability of Random Matrix Theory in Deep Learning. https://arxiv.org/abs/2102.06740.

[15] Guo, J. (n.d.). AI Notes: Initializing neural networks. deeplearning.ai. https://www.deeplearning.ai/ai-notes/initialization.

[16] Ng, A.; Katanforoosh, K. (n.d.). Section 4 (Week 4): Xavier Initialization and Regularization . Section 4 (Week 4). https://cs230.stanford.edu/section/4/.

[17] Kaelbling, L.; Drori, I.; et al. (n.d.). 6.036: Introduction to Machine Learning Spring 2021. 6.036 Spring 2021 Homework 7. https://introml.odl.mit.edu/cat-soop/6.036/homework/hw07.

[18] Dellinger, J. (2019, April 4). Weight Initialization in Neural Networks: A Journey From the Basics to Kaiming. Medium. https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79.

[19] Goel, S. (2019, July 14). Kaiming He initialization. Medium. https://medium.com/@shoray.goel/kaiming-he-initialization-a8d9ed0b5899.

[20] Nisonoff, T. (2018, November 6). Weight initialization for CNNs: A Deep Dive into He Initialization. Medium. https://medium.com/@tylernisonoff/weight-initialization-for-cnns-a-deep-dive-into-he-initialization-50b03f37f53d.

[21] Zaman, T. (2020, May 6). Training a Neural Network using PyTorch. Medium. https://towardsdatascience.com/training-a-neural-network-using-pytorch-72ab708da210.

[22] How to initialize weights in PyTorch? Stack Overflow. (1966, November 1). https://stackoverflow.com/questions/49433936/how-to-initialize-weights-in-pytorch.

[23] Bose, A. (2019, February 22). A Must Read Intro To Neural Networks Using PyTorch - Handwritten Digit Recognition. Medium. https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627.

[24] Amitrajitbose. (2019, February 12). Handwritten Digit Recognition. GitHub. https://github.com/amitrajitbose/handwritten-digit-recognition/blob/master/handwritten_digit_recognition_CPU.ipynb.

[25] Glorot, X.; Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, in Proceedings of Machine Learning Research 9:249-256 http://proceedings.mlr.press/v9/glorot10a.html.

[26] He, K.; Zhang, X.; Ren, S.; Sun, J.; (2015, February 6). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. https://arxiv.org/abs/1502.01852.