Variance-Preserving Initialization Schemes Improve Deep Network Training: But Which Variance is Preserved?

Kyle Luther 1 H. Sebastian Seung2 3

Abstract
Before training a neural net, a classic rule of thumb is to randomly initialize the weights so that the variance of the preactivation is preserved across all layers. This is traditionally interpreted using the total variance due to randomness in both networks (weights) and samples. Alternatively, one can interpret the rule of thumb as preservation of the sample mean and variance for a fixed network, i.e., preactivation statistics computed over the random sample of training samples. The two interpretations differ little for a shallow net, but the difference is shown to be large for a deep ReLU net by decomposing the total variance into the network-averaged sum of the sample variance and square of the sample mean. We demonstrate that the latter term dominates in the later layers through an analytical calculation in the limit of infinite network width, and numerical simulations for finite width. Our experimental results from training neural nets support the idea that preserving sample statistics can be better than preserving total variance. We discuss the implications for the alternative rule of thumb that a network should be initialized to be at the “edge of chaos.”

1. Introduction
The procedure used to initialize the weights and biases of a neural network has a large impact on how fast a network trains, and in some cases determines whether the network will train at all (He et al., 2015). One rule of thumb, dating back to the 90’s, is to randomly initialize the weights so that the preactivations have a fixed variance in all layers of the network. In a fully connected network with tanh nonlinearity and fixed layer width $n$, this can be achieved by drawing weights from a Gaussian distribution with zero mean and standard deviation $\sqrt{1/n}$ (LeCun et al., 1998). The rule of thumb was extended to networks with ReLU nonlinearity by (He et al., 2015), who increased the standard deviation to $\sqrt{2/n}$ to preserve variance in the presence of ReLU. We will refer to this prescription as “Kaiming initialization.”

In the preceding work, the rule of thumb was interpreted using the total variance of the preactivation due to the randomness in both networks (weights) and samples. Intuitively, it might be more relevant to fix the network, and focus only

![Figure 1. Pre-activations in a fully connected network with ReLU nonlinearity when initialized using (He et al., 2015). Each column shows the distribution of 3 individual pre-activations (row 1-3) and net distribution of values over all pre-activations (row 4) in a given layer when Gaussian noise is input to the network. The sample variance is the average variance over samples of the individual preactivations (variance in top 3 rows). The total variance is the variance over samples and pre-activations (variance in bottom row).](image-url)
on the fluctuations of the preactivation over the random sample of samples. In this alternative interpretation of the rule of thumb, a network should be initialized to preserve the sample mean and variance across layers.\(^1\) This alternative interpretation is implicit in the data-dependent initialization scheme of Krähenbühl et al. (2015).

The main goal of this paper is to demonstrate that the two interpretations of the rule of thumb can differ greatly for deep ReLU nets, although there is little difference for shallow nets. The simple but key theoretical insight is to decompose total variance into the network-averaged sum of sample variance and square of sample mean. In the limit of infinite network width, it turns out that both terms in the sum can be calculated using the technique introduced by Poole et al. (2016) for studying the similarity of two preactivation vectors corresponding to two different samples. The propagation of this similarity through the layers of the net determines the propagation of the sample mean and variance.

The formalism is applied to a fully connected ReLU net with constant width and Kaiming initialization to preserve the sample variance across layers. We show that the sample variance vanishes with depth, and the fluctuations in the sample mean across networks dominate the total variance. In other words, Kaiming initialization causes all input samples to be mapped to almost the same preactivation vector, up to small fluctuations around the sample mean. For most neurons in a deep layer, the preactivation has small fluctuations about a large positive mean or a large negative mean. For almost all samples, these neurons are either operating as if they were linear, or do not exist.

The preceding theoretical analysis suggests that the two interpretations of the rule of thumb are very different for deep ReLU nets. Which interpretation will lead to better training is unclear, and the theory as presented is not applicable to convolutional nets or more complex architectures. Therefore we conducted experiments in which convolutional nets were trained on object recognition and image segmentation tasks. We implemented both interpretations of the rule of thumb via data-dependent initializations. In one case, we scaled the weights in each successive layer to preserve total variance. In the other case, we adjusted the biases in each successive layer so that the sample mean of the preactivation vanished, and scaled the weights so that the sample variance remains constant.\(^2\) We found that sample mean-variance preservation led to faster training than initializing at the edge of chaos.
2.1. Sample Mean and Variance

We are interested in the behavior of the sample mean and variance of pre-activations in the network ensemble:

\[(m^t)^2 = \langle \langle u^t_i \rangle^2 \rangle_w \quad (3a)\]
\[(v^t)^2 = \langle \langle (u^t)^2 \rangle_t - \langle u^t_i \rangle^2 \rangle_w \quad (3b)\]

In other words, we are interested in the network averaged squared sample mean, and the network averaged sample variance of a pre-activation in layer \(l\). The average over networks ensures these are the same for all pre-activations in a layer so we omit the index \(i\) for notational clarity.

\(m^t\) and \(v^t\) can be calculated as follows: fix a network, propagate all samples through the network, and compute mean and standard deviation of that pre-activation over samples. Repeat this process and return the mean and variance, averaged over realizations of the network. In this regard, these statistics are closely related to the sample statistics of fixed networks.

2.2. Relation to Total Mean, Variance

\[\begin{align*}
\text{x} \quad \text{Wx-Fix W} \quad \text{Wx-Rand W}
\end{align*}\]

\[\begin{align*}
x_1 & \quad x_2
\end{align*}\]

Figure 2. Distribution of random set of features after multiplication by a fixed matrix compared to distribution of random set of features after multiplication by a single fixed matrix. Middle: features after multiplication by a single fixed matrix. Right: features when acted on by a random set of matrices.

The important statistics in the classic initialization schemes are the total mean and variance:

\[\mu^t = \langle u^t \rangle_w \quad (4a)\]
\[\langle \sigma^t \rangle^2 = \langle (u^t)^2 \rangle_w - \langle u^t \rangle^2 \quad (4b)\]

These can be easily computed by first averaging over the randomness in the network, and then the randomness in the samples, as shown in Glorot & Bengio (2010); He et al. (2015). In ReLU networks initialized using Equation 2, it can be shown that \(\mu^t = 0\) and \(\langle \sigma^t \rangle^2 = \langle (\sigma^t)^2 \rangle\) for any input distribution. That is, the mean of any pre-activation, over networks and inputs, is zero and the variance of all pre-activations, over networks and inputs, is the same.

The difference between the total statistics and network averaged sample statistics is perhaps best illustrated graphically (Figure 2). Assume we have some set of 2D features \(\{x^t : t = 1, 2, ..., T\}\). These are the activations for inputs \(1, 2, ..., T\) in some layer of a width 2 network. Compare the distributions of the following two sets: \(\{Wx^t : t = 1, 2, ..., T\}\) and \(\{W^tx^t, t = 1, 2, ..., T\}\) where \(W\) is a single randomly matrix, \(W^t\) are \(T\) independently chosen (gaussian) matrices and \(x^t\) are elements of the original feature set. The sample mean and variance are the mean and variance of the first set. The total mean and variance give the mean and variance of the second set. Clearly the first set has a different mean and variance, averaged over choices of \(W\), than the second set.

The network averaged sample statistics are related to the total statistics. Using their definitions and the fact that the network mean is zero, we have for all layers:

\[(m^t)^2 + (v^t)^2 = \langle \sigma^t \rangle^2 \quad (5)\]

The sum of the network averaged squared sample mean and the network averaged sample variance equals the sample averaged network variance. In other words, the sample averaged network variance contains two contributions, one from the typical sample mean and the other from the typical sample variance.

2.3. Relation to Cosine Similarity

The critical technical contribution in this paper is relating the network averaged squared sample mean and sample variance to the dot product studied in Poole et al. (2016).

Let \(u^t_a\) and \(u^t_b\) be pre-activations in layer \(l\) from two independently chosen samples \(x^t_a\) and \(x^t_b\). For any network, it can be shown that:

\[\begin{align*}
(m^t)^2 &= \frac{1}{nt} \langle \langle u^t_a \cdot u^t_b \rangle \rangle_w \quad (6a) \\
(v^t)^2 &= \langle \langle (u^t)^2 \rangle_t - \langle u^t_i \rangle^2 \rangle_w \quad (6b)
\end{align*}\]

That is, the network averaged squared sample mean is the inner product between preactivations, averaged over networks and independently chosen inputs. Using Equation 5, this implies that the network and sample averaged inner product is the difference between the total variance and the network averaged sample variance.

Qualitatively, this says that when the average similarity between two sets of preactivations from two independently
Variance-Preserving Initialization Schemes Improve Deep Network Training: But Which Variance is Preserved?

chosen samples increases, the variances over samples decreases. If all inputs are mapped to similar outputs, the output has small variance over inputs.

This relationship can be shown by writing \( \langle u_a^l \cdot u_b^l \rangle_{0w} = \langle \langle (u_{a}^l)^2 \rangle_w \rangle_{c,0} \). Note that in ReLU networks, it can be shown that \( \frac{1}{m} \langle u_a^l \cdot u_b^l \rangle_{0w} = 2 \langle x_{a}^{l-1} \cdot x_{b}^{l-1} \rangle_{c,w} \). We show the map for the sample mean and the resulting dynamics of the sample mean and variance alongside the total mean and variance in Figure 3. For a more qualitative feel for what these equations are describing, examine Figure 1. One can see that the variance of individual pre-activations decays with depth and they are fluctuated around some large mean value.

3. Rectified Linear Networks

We now characterize the behavior of \( m^l, v^l \) in rectified linear nets initialized with Kaiming initialization. The results for wide networks are shown in Figure 3.

3.1. Infinite Width, Uncorrelated Gaussian Input

We have shown that the network averaged squared sample mean is just the inner product averaged over networks and samples. It was shown how to calculate this inner product in wide networks in (Poole et al., 2016). We provide the result for ReLU networks initialized using Kaiming initialization here. Fix two samples \( x_a^0 \) and \( x_b^0 \). Define \( c_1^0 = \frac{1}{m} \langle x_a^{l-1} \cdot x_b^{l-1} \rangle_{c,w} \). Assume we have normalized the inputs so that \( \frac{1}{m^0} x_a^0 \cdot x_b^0 = 1 \).

Then an iterated map, \( K \), for the inner product of activations in all layers can be defined:

\[
K(c^l) = 2 \int Dz_1 Dz_2 f(u_1) f(u_2)
\]

\[
u_1 = z_1,
\]

\[
u_2 = c^l z_1 + \sqrt{1 - (c^l)^2 z_2}
\]

where \( Dz_i = \frac{1}{\sqrt{2\pi}} e^{-z_i^2/2} \) are the unit Gaussian measures.

To get \( m^l \) we want the average of this map over the distribution of sample correlations: \( m^l = \langle c^l \rangle_t = \langle K^l(c^0) \rangle_t \)

where \( K^l = K \circ \ldots \circ K \) and \( c^l = \frac{1}{\sqrt{m^0}} x_a^0 \cdot x_b^0 \). For arbitrary input distributions, this is a problematic integral. For any distribution with sufficiently sharp distribution of sample correlations, we can move the expectation inside the map:

\( \langle K^l(c^0) \rangle_t = K^l(\langle c^0 \rangle_t) \)

Of course this is not true in general but this is true for uncorrelated gaussian inputs in the wide network limit. The justification is that for two independently chosen unit gaussian vectors, the expected (normalized) dot product converges to 0 with extremely high probability. In the large \( n^0 \) limit, we have \( \lim_{n^0 \to \infty} Var(\frac{1}{\sqrt{m^0}} x_a^0 \cdot x_b^0) = 0 \) and therefore we can replace \( \langle c^0 \rangle_t \) with its average, 0.

So for a wide ReLU network with uncorrelated gaussian inputs, we have the following propagation equations for \( m^l, v^l \):

\[
(m^l)^2 = 2 K^l(0)
\]

\[
(v^l)^2 = 2(1 - K^l(0))
\]

where \( K^l = K \circ \ldots \circ K \) and \( K \) is defined in Equation 7.

We show the map for the sample mean and the resulting dynamics of the sample mean and variance alongside the total mean and variance in Figure 3. For a more qualitative feel for what these equations are describing, examine Figure 1. One can see that the variance of individual pre-activations decays with depth and they are fluctuated around some large mean value.

3.1.1. Variance Decay

We show that the sample variance decays with depth. To do this, we show that \( K(c) > c \) for all \( c \in (−1, 1) \) Using the definitions \( u_1 = z_1 \) and \( u_2 = c z_1 + \sqrt{1 - c^2} z_2 \) and \( f = \text{ReLU} \) it follows that:

\[
K(c) = 2 \int Dz_1 Dz_2 f(u_1) f(u_2)
\]

\[
> \int Dz_1 Dz_2 u_1 u_2 = c
\]

Figure 3. Left: Iterated map for the network averaged squared sample mean in ReLU network. Right: network averaged squared sample mean (solid blue), network averaged sample standard deviation (solid red), total mean (dashed blue), total standard deviation (dashed red) as a function of layer for standard normal network inputs. The network averaged squared sample mean approaches the total standard deviation while the network averaged standard deviation approaches the total mean.
3.1.2. Deep Net Limit

The sample variance shows a different scaling with depth than the total variance. The network variance in general exhibit exponential scaling with depth, unless the scale of the weights is chosen precisely so that the total variance is fixed. Interestingly the sample variance does not vanish exponentially. It is sub-exponential. It can be shown that

$$\lim_{c \to 1} dK(c)/dc = 1$$

(10)

This basic fact was shown in Anonymous (2019). This implies a sub-exponential convergence of $m^l$, $v^l$ to their fixed points, $\sqrt{2}$ and 0. That is the sample variance decays slowly, even when the total variance remains fixed.

3.1.3. Distribution of Pre-Activations

With some algebraic manipulation, we can rewrite the propagation equation:

$$K(c) = 2 \int d\mu P(\mu) \left[ \int dv P(v|\mu) f(v)^2 \right]$$

(11a)

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

(11b)

$$P(v|\mu) = \frac{1}{\sqrt{2\pi}\sqrt{1-c^2}} e^{-\frac{(v-\mu)^2}{2(1-c^2)}}$$

(11c)

We interpret the outer integral as a gaussian integral over the mean over samples of individual pre-activations. This has variance over networks of $c^2$. We interpret the inner integral as the squared mean of an activation over samples of an individual activation conditioned on the fact that the mean over samples of its pre-activation is $\mu$. The form of this equation is more similar to the original definition of the sample mean:

$$\langle (u_i^2) \rangle_w = \int P(w) \left[ \int P(u|w) u^2 \right]$$

(12)

We can see this effect empirically in Figure 1.

3.2. Finite Width, Uncorrelated Gaussian Input

To see how well the infinite width calculation describes propagation of gaussian noise in finite size networks we empirically compute the sample mean and variances. We sample 30 random feedforward networks of depth 50 with uniform width, $n$, per layer. We compare $n = 30, 100, 300, 1000, 3000$. 100 input vectors are chosen with elements sampled i.i.d. from a $N(0, 1)$ distribution. These are then propagated through each network and the empirical ratio of squared sample mean to sample variance for each network is computed:

$$r^l = \sqrt{\frac{\sum_{i=1}^{n^l} (u_i^l)^2}{\sum_{i=1}^{n^l} (\langle u_i^l \rangle_t)^2} - \langle (u_i^l)^2 \rangle_t}$$

(13)

Figure 4. Ratio $r = \frac{m}{v}$ as a function of layer in finite sized networks. The theoretical prediction (dashed line) seems to provide an overestimate that better matches the observed ratio as width increases and depth decreases. Shaded region shows standard deviation of each curve over 30 randomly initialized networks.

The mean and standard deviation of this empirical ratio over network configurations is shown in Figure 4.

3.3. Non-Gaussian Input

We have made a rather strong assumption in our calculation: inputs are uncorrelated gaussian random variables. This was used to argue that the inner product between any independently chosen samples is 0 in the wide net limit which greatly simplified the propagation equations. How important is this assumption?

Because $m^l$ is simply the average of $v^l$ over inputs, if $K(c) > c$ for all $c \in [-1, 1)$, then the average of $v^l$ over inputs increases and therefore $m^l$ increases in every layer. The precise rate may depend on the precise distribution. One can see from the map in Equation 7 for ReLU networks, that $K^l(c) > c$ for all $c \in [-1, 1)$, and therefore the network averaged sample variance decays to zero for any distribution in a wide ReLU network.

3.4. Intuition for Sample Variance Decay

Without any calculation, we can argue that random ReLU networks should cause sample variance decay. Applying the ReLU activation to any random variable increases its mean and decreases its variance. Using the definition of $f$ in Equation 1, it can be shown that for any random variable $x$ with distribution $P$ that has non-zero support for $x < 0$
We answer this question empirically. We propose an alternative initialization scheme that initializes the biases in a data-dependent manner to ensure the sample mean of every pre-activation is zero and the sample variance remains fixed with layer. The precise rate at which this occurs was calculated earlier in this section.

4. Bias Initialization

In rectified linear networks initialized to preserve the total variance, we have shown that the sample mean and variance are not preserved. The typical sample mean of pre-activations grows with depth while the typical sample variance decays. Is this just a theoretical curiosity? Does this actually impact training of networks?

We answer this question empirically. We propose an alternative initialization scheme that initializes the biases in a data-dependent manner to ensure the sample mean of every pre-activation is zero and the sample variance remains fixed with layer. We compare this to the standard initialization which only ensures the total variance is preserved.

4.1. Initialization Details

4.1.1. Weight Initialization

Recall the original argument for Kaiming initialization: scale the weights so the total variance remains constant with layer. In the simple ReLU networks we examined so far, this meant the weights should have a variance of \(\sqrt{2/n}\). A number of architectural elements arise in real world networks (skip connections and pooling in our two experiments) that make this scaling no longer hold. Additionally when we initialize the biases or apply mean-only normalization, the scaling argument no longer holds.

In principle we could analytically calculate the scale factor in all of these settings and use this instead of the previously derived \(\sqrt{2/n}\) scale. A much simpler alternative is to simply set the scale of the weights at each layer empirically so that the variance (over all pre-activations and samples over a fixed set of data points) is fixed in every layer. Note that the scale is per-layer, rather than per-feature. This idea was proposed before, notably in (Mishkin & Matas, 2015)

In particular we perform the following weight initialization for both settings:

\[
\begin{align*}
\sigma_B^l & = \frac{1}{n^l T} \sum_{i=1}^{n^l} \sum_{t=1}^{T} (u^l_{ti})^2 + \epsilon \\
W^l_{ij} & \leftarrow W^l_{ij} \frac{1}{\sqrt{(\sigma_B^l)^2 + \epsilon}}
\end{align*}
\]

where the sum over \(T\) indices the sum over 5 minibatches of data. \(\epsilon = 1e^{-5}\). Note that strictly speaking \((\sigma_B^l)^2\) is not the variance but the 2nd moment of the empirical distribution over pre-activations and samples. Because of the average over all pre-activations in a layer, this distribution is very nearly zero-mean and the empirical second moment is very nearly equal to the empirical variance.

Note that the weights in layer \(l\) must be initialized after the weights in layer \(l - 1\). This initialization just changes the scale at each layer. It therefore does not changes the empirical sample mean to sample standard deviation curves shown in Figure 4.

We call this initialization scale initialization.

4.1.2. Bias Initialization

The difference between the two initializations we compare comes from the setting of the biases. In the bias initialization scheme, rather than setting the biases to zero, we use them to subtract out the preactivation mean:

\[
\begin{align*}
\mu_B^l & = \frac{1}{T} \sum_{i=1}^{T} u^l_{ti} \\
b^l_i & \leftarrow -\mu_B^l
\end{align*}
\]

where the sum over \(T\) indices the sum over 5 minibatches of data. Initializing the bias in layer \(l\) is performed after initializing the weights in layer \(l\) and before the bias in layer \(l - 1\).

Critically, this is done per-feature. This ensures the sample mean is zero for every feature.

The per-layer scaling ensures the average (over pre-activations in a layer) sample variance in each channel is fixed with layer. In principle we could have used a per-feature sample scaling as well. However that could introduce additional complications related to the distribution of preactivations that are not really related to sample statistics vs. total statistics question.

We call this initialization scale+bias initialization.

4.2. Object Recognition Details

We compare scale and scale+bias initialization for training a convolutional network on a popular object recognition task:
Variance-Preserving Initialization Schemes Improve Deep Network Training: But Which Variance is Preserved?

Figure 5. Empirical ratio of squared sample mean to sample variance in two popular convolutional networks. The theoretical curve shows the ratio we would expect to see in a wide network as a function of layer. Left: UNet on ISBI-2012 dataset. Residual connections begin between layer 10 and 15 and these cause the empirical ratio to come back down, giving the upside down "U" shape we see. Right: All Convolutional Net on CIFAR-10 dataset.

CIFAR10. We use the All-Conv-C architecture in (Springenberg et al., 2014). This contains 9 layers of convolution, a global pooling layer, and a single fully connected output layer. We use reflection padding wherever necessary.

The right half of Figure 5 shows the empirical ratio in this network when initialized using scale initialization. The curves are generated by sampling 30 random networks and computing the empirical ratio and standard deviation of this ratio over network configurations. The sample statistics are computed using 5 randomly chosen 512x512 images from the ISBI-2012 dataset. Max pooling operations in the network actually contribute to the ratio growth. The upside down U shape occurs because of the skip connections in the network.

For augmentation, we use 90 degree rotations, reflections, and elastic warps described in (Ronneberger et al., 2015). We use SGD with momentum of 0.9 in all settings. For each setting, we search over learning rates \{0.0003, 0.001, 0.003, 0.01, 0.03\} and choose the fastest converging learning rate (defined by learning rate with lowest loss after 10k iterations). We perform 5 runs for each configuration and plot the average learning curves and variance over runs for the best learning rate.

4.4. Results

Figure 6. Learning Curves. Training loss as a function of iteration is shown for two tasks using the total variance preserving scale initialization scheme, and sample variance preserving scale+bias initialization scheme. Left: UNet2D on ISBI-2012 segmentation task. Right: All Convolutional Net on CIFAR-10.

The results are shown in Figure 6. The learning curves and their standard deviation over 5 randomly initialized runs are plotted. We observe in both cases, simply initializing the biases results in a significant gain in training speed. These results are robust in the sense that the second fastest learning rate for scale+bias initialization was still faster than the scale only initialization.

It is interesting to compare the empirical ratio between the scale and scale+bias initialization for a popular biomedical image segmentation task: predicting cell boundaries in the ISBI2012 neuronal segmentation task. We use the UNet architecture in (Ronneberger et al., 2015) which contains 23 layers of convolutions and 4 layers of max pooling. Instead of using "valid" convolutions, we use reflection padding on each feature map to preserve the spatial size after each convolution operation.

Figure 5 shows the empirical ratio in this network as a function of depth. The curves are generated by sampling 30 random networks and computing the empirical ratio and standard deviation of this ratio over network configurations. The sample statistics are computed using 5 randomly chosen 512x512 images from the ISBI-2012 dataset. Max pooling operations in the network actually contribute to the ratio growth. The upside down "U" shape occurs because of the skip connections in the network.

For augmentation, we use 90 degree rotations, reflections, and elastic warps described in (Ronneberger et al., 2015). We use SGD with momentum of 0.9 in all settings. For each setting, we search over learning rates \{0.0003, 0.001, 0.003, 0.01, 0.03\} and choose the fastest converging learning rate (defined by learning rate with lowest loss after 10k iterations). We perform 5 runs for each configuration and plot the average learning curves and variance over runs for the best learning rate.

4.3. Image Segmentation Details

We compare scale and scale+bias initialization for a popular biomedical image segmentation task: predicting cell boundaries in the ISBI2012 neuronal segmentation task. We use the UNet architecture in (Ronneberger et al., 2015) which contains 23 layers of convolutions and 4 layers of max pooling. Instead of using "valid" convolutions, we use reflection padding on each feature map to preserve the spatial size after each convolution operation.

Figure 5 shows the empirical ratio in this network as a function of depth. The curves are generated by sampling 30 random networks and computing the empirical ratio and standard deviation of this ratio over network configurations. The sample statistics are computed using 5 randomly chosen 512x512 images from the ISBI-2012 dataset. Max pooling operations in the network actually contribute to the ratio growth. The upside down U shape occurs because of the skip connections in the network.

For augmentation, we use 90 degree rotations, reflections, and elastic warps described in (Ronneberger et al., 2015). We use SGD with momentum of 0.9 in all settings. For each setting, we search over learning rates \{0.0003, 0.001, 0.003, 0.01, 0.03\} and choose the fastest converging learning rate (defined by learning rate with lowest loss after 10k iterations). We perform 5 runs for each configuration and plot the average learning curves and variance over runs for the best learning rate.
In other networks, we have observed a similar ordering of training speed between scale, and scale+bias initialization. The degree to which bias initialization helps depends on precise configuration and can be modified by modifying a number of architectural elements, including network width, depth, adding residual connections, using strided instead of pooling operations, zero-padding instead of reflection padding.

A caveat with both experiments is that they are only trained for a relatively small number of updates. One might wonder if the initialization is less important after a long training time, for instance when training networks on large datasets such as ImageNet.

5. Discussion

We have shown that in general the total variance over networks and samples can be quite different from the variance of individual networks over just samples. In wide rectified linear networks, the sample variance decays to zero. Intuitively what happens is that all samples collapse to small fluctuations around some large mean value in the output.

This picture of signal propagation, using the sample mean and variance, can actually be used to give an intuitive description of the “ordered” phase in random neural networks originally described in (Poole et al., 2016). The authors showed that networks in the ordered phase implement significantly less complicated functions than networks in the chaotic phase as measured by the curvature of a 1D manifold propagated through the network. This phenomenon is related to the sample mean and variance. In the ordered phase, pairs of samples are mapped to high cosine similarity activations in higher layers of the network. This implies that the mean of these pre-activations grow. When the total variance is preserved, growth of the sample mean implies decay of the sample variance and thus in the ordered state, inputs are mapped to small fluctuations around some large mean value.

Assuming differentiable activation function, this actually suggests something about the class of functions being implemented in a network: namely that the network is mostly linear. Decomposing states in higher layers as fluctuations $\delta_t$ around a mean $\mu$, we can write the effect of a differentiable activation function $f$ as: $f(\mu + \delta_t) = f(\mu) + f'(\mu)\delta_t$. That is, if all inputs are reduced to small fluctuations, the network is essentially implementing a linear function in higher layers so of course it behaves differently than a network in the chaotic phase.

The sample mean and variance can also provide some insight into a popular algorithm: batchnorm (Ioffe & Szegedy, 2015). It is often found to improve training speed, allow for higher learning rates, reduce the sensitivity to initialization and additionally seems to improve generalization performance in a variety of settings. The precise reason for these benefits however is still unknown. By examining the sample mean and variance of pre-activations in the forward pass, it is easy to show that batchnorm can qualitatively change the forward pass of a random network.

By construction, batchnorm subtracts the mean and divides by the standard deviation for every pre-activation in the network. Importantly these are the mean and standard deviation, of the fixed network, over samples in a minibatch. Assuming the minibatch is sufficiently large, these are close to the mean and standard deviation of the dataset and therefore networks using batchnorm preserve the sample mean and sample variance. Therefore in networks, such as rectified linear networks initialized using the standard scaling argument, simply inserting batchnorm qualitatively changes the forward pass dynamics and initialization of the network.

We have shown in our experiments that this initization may in fact be helpful for training networks. This suggests that perhaps part of the successes of batchnorm, at least in certain rectified linear networks, may be attributable to its impact on initialization, independent of any reparameterization it is inducing.

We have not discussed the backward pass at all in this paper. Intuitively, it can be seen in ReLU networks at least, that by preserving the sample variance, instead of the total variance, we have to increase the scale of the weights to prevent the sample variance from decaying. Going through the standard variance of gradient calculation (He et al., 2015), we can see that gradients actually explode with depth as a result of our initialization and batchnorm. This basic claim about exploding gradients was argued in the recent paper (Yang et al., 2019). However, a simple examination using just the sample mean and variance allows for a quick explanation of this phenomenon, at least in rectified linear networks. Another way to intuitively see this phenomenon is to consider that by centering preactivations, one is requiring that the nonlinearity be used more effectively than in the case where the sample mean is far away from zero.

In light of this observation of exploding gradients, it may actually seem quite surprising that initializing the biases seemed to improve performance. It was proposed in (Schoenholz et al., 2016) that initializing on edge of chaos is necessary for training deep networks (where gradients neither vanish nor explode). Perhaps exploding gradients becomes more important in extremely deep networks, rather than the ones we are training. Perhaps more fundamental understanding is required to properly relate properties of random networks to their training dynamics. Future work may investigate this relationship in more depth.
References

Anonymous. On the selection of initialization and activation function for deep neural networks. In Submitted to International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=H1lJws05K7. under review.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M. (eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, volume 9 of Proceedings of Machine Learning Research, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Krähenbühl, P., Doersch, C., Donahue, J., and Darrell, T. Data-dependent initializations of convolutional neural networks. CoRR, abs/1511.06856, 2015. URL http://arxiv.org/abs/1511.06856.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K. R. Efficient BackProp, pp. 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/bisbn-94308-8_2. URL https://doi.org/10.1007/3-540-49430-8_2.

Mishkin, D. and Matas, J. All you need is a good init. CoRR, abs/1511.06422, 2015. URL http://arxiv.org/abs/1511.06422.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. Exponential expressivity in deep neural networks through transient chaos. arXiv e-prints, art. arXiv:1606.05340, June 2016.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. CoRR, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

Schoenholz, S. S., Gilmer, J., Ganguli, S., and Sohl-Dickstein, J. Deep Information Propagation. arXiv e-prints, art. arXiv:1611.01232, November 2016.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. Striving for simplicity: The all convolutional net. CoRR, abs/1412.6806, 2014. URL http://arxiv.org/abs/1412.6806.

Yang, G., Pennington, J., Rao, V., Sohl-Dickstein, J., and Schoenholz, S. S. A mean field theory of batch normalization. In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=SyMDXnCcF7.