4

Three Tricks in Training of Neural Network

4.1 Reviewing

- Why Kaiming initialization is not good enough for deep narrow network (i.e., large L/d)?

  Answer: Error accumulation, i.e.,

  $$\text{error} \sim \exp(L/d)$$

- Besides random initialization, what else is possible?

  Answer: orthogonalization.

- Besides weight matrices $W^{1:L}$, what eles can be tuned at initialization?

  Answer: Input strength (size) $\|x\|^2$.

Motivation Up to today, there are three tricks to train a deep neural network:

1. Intialization;
2. Batch Normalization;

3. Residual units (architecture)

Back to 2012, there were 6-10 tricks to train the AlexNet. As time goes by, scientists found that many of these tricks are not necessary. In this lecture, we will talk about highlights in these three tricks.

4.2 Intialization: Dynamical Isometry

Up to now, Dynamical Isometry is the peak of initilization theory for deep learning. Given a neural network $\mathrm{NN} : \mathcal{X} \to \mathcal{Y}$ such that

$$y = \mathrm{NN}(x; W),$$

the goal is to choose W such that $\|y\| \approx \|x\|, \forall x$. More precisely, we want to make singular values of the input-output Jacobian J all close to 1. Following the notation setting in section (1.3), we have

$$J = \frac{\partial z^L}{\partial z^0} = \prod_{\ell=1}^{L} \frac{\partial z^\ell}{\partial z^{\ell-1}} = \prod_{\ell=1}^{L} (D^\ell W^\ell)$$

with $D^\ell = \mathrm{diag}(\phi'(\{h_i^\ell\}_{i=1}^{d_{\ell_i}}))$.

The key turns to the understanding of the distribution of eigenvalues of the matrix JJ^{T} (i.e., singular values of J). There is a classical theory for the eigenvalues of random matrix, which claims that its distribution is like a semi-circle:

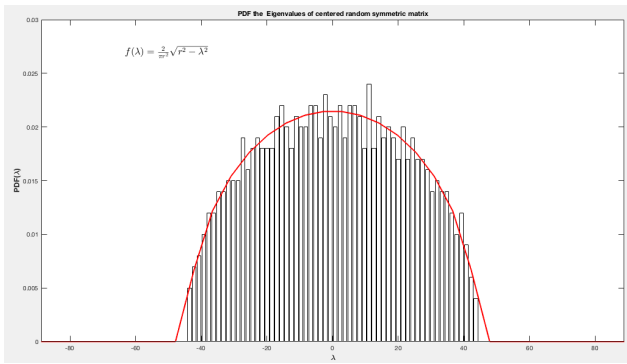


Figure 4.1: The pdf of eigenvalues for the scaled (symmetric) random matrix $\frac{1}{\sqrt{N}}W$ with $N = 1000$ and $w_{i,j}$ follows normal distribution.The simulation code is in https://www.mathworks.com/matlabcentral/fileexchange/46464-wigner-s-semicircle-law

Theorem 4.1 (Wigner's semi-circle law)**.** Given a symmetric matrix $W \in \mathbb{R}^{N\times N}$ whose entries $W_{i,j}, i \geq j$ are independent random variables, the asymptotic distribution (as $N \to \infty$) of the eigenvalues of W is like a semi-circle.

The semi-circle law is related to our problem: The Jacobian J is a random matrix, This semi-circle law is related to our problemm where the randomness comes from W. More precisely, it is a product of non-linear function of random matrices. In order to resolve the nonlinearity, we need to use the free probability(i.e., S-transform) tool to calculate the distribution of singular values of J.

The possibility of dynamical isometry The paper (Pennington *et al.*, 2017) gives possibility for realizing dynamical isometry for deep *nonlinear* neural networks in different scenarios.

Table 4.1: Summarization of the results for the possibilities for realizing dynamical isometry for deep neural networks with different types of activations and initializations

	Scenarios	possibilities
Section (2.5.1)	Guassian Initialization +Relu Activation	No
Section (2.5.1)	Guassian Initialization +Hard-tanh Activation	No
Section (2.5.2)	Orthogonal Initialization +Relu Activation	No
Section (2.5.2)	Orthogonal Initialization +Hard-tanh Activation	Yes

In particular, for the last scenario, one can achieve dynamical isometry by tuning $\|x\|^2, \sigma_w^2, \sigma_b^2$ properly[1]. In this case, the deep neural network enjoys depth-independent training time.

The mechanism for dynamical isometry is to realize three goals simultaneously, i.e., relies on solving three types of equations:

- Feedforward propagation: keep variance

[1]Here $\sigma_w^2 \triangleq \frac{1}{d}\mathrm{var}(w_{i,j}^{\ell})$ and σ_b^2 refers to the variance of bias for pre-activation

- Backforward propagation: keep variance
- Eigenvalues of $J^{\mathrm{T}}J$ close to 1.

The realization relies on choosing $\|x\|^2, \sigma_w^2, \sigma_b^2$. The failure of first three scenarios in Table (4.1) is due to the loss of freedom of these variables.

Why "Orthogonal" is "Difficult" Before the paper (Xiao *et al.*, 2018) comes out, the orthogonal initialization for CNN is difficult, One reason is that the orthogonality for this architecture is not well-defined:

- For fully connected neural network, we say the weight matrix is orthogonal if $W^{\mathrm{T}}W = I$.
- For CNN, consider the pre-activation process, i.e., $y = W \otimes x$, it is unclear how to define the orthogonality for tensor product. To resolve this issue, the paper (Xiao *et al.*, 2018) defines the orthogonality as $\|y\| = \|x\|, \forall x$.

After designing the orthogonal weights for CNN, they are able to train 10000-layer-net for CIFAR10. This result indicates that initialization is enough for training ultra-deep neural network.

4.3 Batch Normalization

Motivation Previous data analysis knowledage tells us that the success of data processing also depends on the *input normalization* (preprocessing of data), even for linear regression. It's nature to do the similar thing for the training of neural networks.

Example 4.1. Review the linear regression problem. Given the data points (x_i, y_i) for $i = 1, \ldots, n$, one wants to select parameter w^* such that the quadratic loss is minimized:

$$w^* = \arg\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (w^{\mathrm{T}} x_i - y_i)^2$$

For the data matrix $X = [x_1, x_2, \ldots, x_n]$, one should always do the row normalization:

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \xrightarrow{\eta} \begin{bmatrix} \tilde{x}_1 & \tilde{x}_2 & \cdots & \tilde{x}_n \end{bmatrix}$$

where $\tilde{x}_i \triangleq \frac{x_i-\mu}{\sigma}$ with $\mu = \left[\frac{1}{n}\sum_i x_{i,j}\right]_j$ and $\sigma = \left[1/n \cdot \sum_i (a_{i,j}-\mu)^2\right]_j$. Here the minus and division operator is performed *component-wisely*.

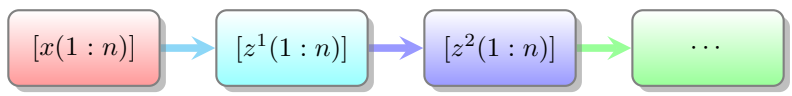
From the perspective of optimization, this opertion improves the *convergence speed*, since the operator η will reduce the condition number[2] of the data matrix.

From the experience of linear regression, we gain some knowledge for the training of neural nets:

- It is necessary to do pre-processing of the input.
- Besides, for each layer, we might have the same issue that the condition number for variables before activation is large or small.

Before (Glorot and Bengio, 2010), people uses the layerwise-per training technique, which also brings up some problems such as the inefficiency of training. Therefore, we need other techniques to solve the issue.

Consider the neural network represented in the figure below:



where z^ℓ's are variables before activation. The goal is to make matrices

$$Z^\ell = \begin{pmatrix} z^\ell(1) & \cdots & z_1^\ell(n) \end{pmatrix} = \begin{pmatrix} z_1^\ell(1) & \cdots & z_1^\ell(n) \\ \vdots & \ddots & \vdots \\ z_d^\ell(1) & \cdots & z_d^\ell(n) \end{pmatrix}$$

become well-conditioned for each $\ell = 1, \ldots, L$, i.e., each row has zero mean and unit variance.[3]

To fully understand how the batch normalization achieves the row normalization for each data matrices Z^ℓ, consider the toy example presented below:

[2] $\kappa(X) = \lambda_{\max}(X)/\lambda_{\min}(X)$ denotes the condition number of X, see Prof. Luo's Note Lecture 2 for more detailed explanations.

[3] The i-th row denotes the data points for the i-th feature, and the row normalization makes these data points well-scaled

Example 4.2 (Toy Example)**.** Consider an aritificial problem defined as

$$\begin{array}{ll} \min_h & F(h) \triangleq \sum_{i=1}^n g(h_i) \\ \text{with} & \sum_i h_i = 0 \\ & \sum_i h_i^2 = 1 \end{array} \tag{4.1}$$

Combining with gradent descent method, there are at least 3 ways to deal with it:

Method 1: Pure Algorithmic Correction The intuitive way is that in each iteration one performs the gradient descent, and then project the new iterates within the constraint set. However, the projection into this constraint set could be diffucult due to its non-convexity. Therefore, we modify the projection step with the *normalization operator* η:

$$\begin{array}{ll} \eta(h_{1:n}) & \triangleq \frac{h_{1:n}-\mu}{\sigma} \\ \text{with} & \mu = \frac{1}{n}\sum_i h_i \\ & \sigma = \frac{1}{n}\sum_i \|h_i - \mu\|^2 \end{array}$$

The whole algorithm is presented below:

$$\begin{cases} h^{t+1/2} \leftarrow h - \nabla F(h) \\ h^{t+1} \leftarrow \eta(h^{t+1/2}) \end{cases}$$

Method 2: Constrained optimization Consider this problem as a constrained optimization problem, and thus it's natural to solve this problem from its dual (or solve primal and dual simultaneously), e.g., applying Lagrangian method or ADMM.

Remark 4.1. The method 1 solves the problem in an algorithmic way, i.e., modify the gradient project method in a heuristic way; The method 2 is more about reformulation, i.e., reformulate this problem from its dual and therefore solve the new convex problem instead. We observe that both of them don't work well in practice. The method 3 reformulates this problem in another way and we found it works well.

Method 3: New way of Formulation We reformulate this problem into unconstrained optimization, by substituting the constraint into the

objective function:

$$\min_h \quad F(\eta(h)) \tag{4.2}$$

Then the optimal solution of the origin problem (3.1) is tractable by setting $h^* = \eta(\arg\min_h F(\eta(h)))$. For this unconstrained optimization problem, we can apply the gradient descent to solve it.

Motivation for Batch Normalization (BN) The insights of BN is similar to that in Method 3. Consider a 2-layer neural network computing

$$e = F_2(F_1(x, W_1), W_2) \tag{4.3}$$

where F_1, F_2 are arbitary transformations, and the parameters W_1, W_2 are to be chosen so as to minimize the loss e. Learning W_2 can be viewed as if the inputs $z \triangleq F_1(x, W_1)$ are fed into the sub-network

$$e = F_2(z, W_2).$$

The goal for BN is to ensure the distribution of nonlinearity inputs (i.e., z) remains more stable during the training process, i.e., pick W_2, W_2 to minimize the loss function and within the constraint set

$$\mathbb{E}[x] = 0, \mathrm{Var}[x] = 1, \qquad \mathbb{E}[z] = 0, \mathrm{Var}[z] = 1.$$

From the experience of Method 3, it suffices to reformulate (4.3) by adding the *normalization* process before the non-linear activation in each layer.

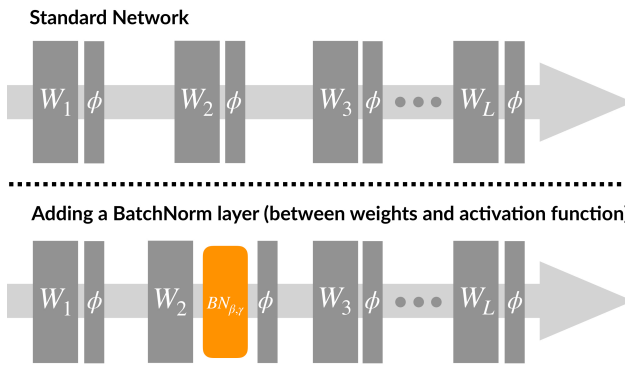


Figure 4.2: Adding the Bacth Normalization process in the *second* layer

The *normalization* process can be changed in different scenarios, e.g., for capsule-net, change it with the *clustering* process.

Gradient Computation if adding the BN The standard one-layer neural network can be described as a non-linear parametric function

$$z = g(Wu),$$

where g is the non-linear transformation, W is the weight matrix. By adding the BN, the standard function is replaced with

$$z = g(\mathrm{BN}(Wu)),$$

where the BN transform is applied independently to each dimension of $x = Wu$. We set $h = Wu$, then we show how to compute $\frac{\partial z}{\partial h}$:

1. Step 1: decompose the transformation from h to z into paths.

   Note that $z = z(h, \mu(h), \sigma(h))$. Therefore, the paths could be:

   $$\begin{aligned} &h \to z; && \text{provided that } \mu, \sigma \text{ are fixed} \\ &h \to \mu \to z; \\ &h \to \sigma \to z. \end{aligned}$$

2. Step 2: Apply the Chain Rule.

   $$\frac{\tilde{\partial} z}{\tilde{\partial} h} = \left.\frac{\tilde{\partial} z}{\tilde{\partial} h}\right|_{\mu,\sigma \text{ fixed}} + \frac{\tilde{\partial} z}{\tilde{\partial} \mu}\frac{\tilde{\partial} \mu}{\tilde{\partial} h} + \frac{\tilde{\partial} z}{\tilde{\partial} \sigma}\frac{\tilde{\partial} \sigma}{\tilde{\partial} h}$$

   in which the notion of generalized Jacobian is adpoted.

Remark 4.2 (Representation Power)**.** Re-consider the formula (4.1). We find that

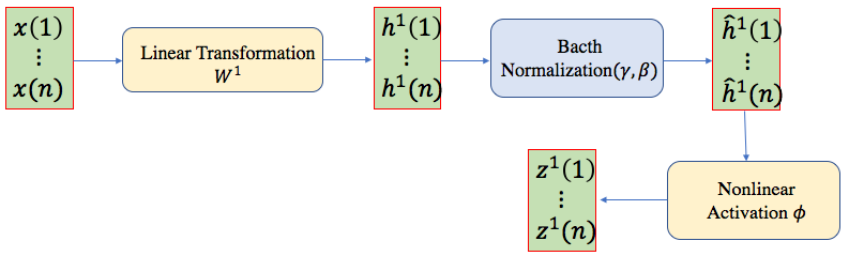
$$\{F(h) \mid h \in \mathbb{R}^n\} \supseteq \{F(\hat{h}) \mid \sum_i \hat{h} = 0, \sum_i \hat{h}^2 = 1\}.$$

Therefore, the $\min F(h)$ is not necessarily equivalent to $\min F(\eta(h))$. In order to resolve this issue, we need to "store" the representation power by scale and shift $\eta(h)$ in (4.2), i.e., introducing γ, β to form a new problem

$$\min_{h,\gamma,\beta} F(\gamma \cdot \eta(h) + \beta) \tag{4.4}$$

Now the set $\{F(h) \mid h \in \mathbb{R}^n\} = \{F(\gamma \cdot \eta(h) + \beta) \mid h \in \mathbb{R}^n, \gamma, \beta \in \mathbb{R}\}$.

In the remaining of this section, let's discuss more about applying BN in practice.



Remark 4.3. Batch Normalization is applied before the non-linear activation in each layer.

Remark 4.4. The function $\mathrm{BN}_{\gamma,\beta}(\cdot)$ is applied to each row of the matrix

$$\begin{pmatrix} h^1(1) & \cdots & h^1(n) \end{pmatrix}.$$

Remark 4.5. In practice, the BN is applied for mini-batch with batch size B. In other words, the total n inputs are separated into N batches, where each batch contains B inputs. The BN is performed on each single batch. This technique changes the problem form:

Standard Formulation	$\min_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$
Mini-Batch Formulation	$\min_\theta \frac{1}{N}\sum_{i=1}^{N} \tilde{\ell}(\tilde{f}_\theta(x_{i,1:B}), (y_{i,1:B}))$

Previously, the neural network is a single-input-single-output (SISO) function:

$$x_i \xrightarrow{f_\theta} \hat{y}_i$$

Now, with mini-batch BN, the neural network is a multi-input-multi-output (MIMO) function:

$$x_{i,1:B} \xrightarrow{\tilde{f}_\theta} \hat{y}_{i,1:B}$$

Remark 4.6. In each batch i, $\tilde{f}_\theta$ is also dependent on the parameter $\bar{\mu}_i, \bar{\sigma}_i$. Therefore, for the whole neural network $\tilde{f}_\theta(\bar{\mu}, \bar{\sigma})$, there could be an issue about how to select the parameters $\bar{\mu}$ and $\bar{\sigma}$.

Remark 4.7. For the problem (4.4), if we choose $\gamma = \|h\|$ and $\beta = \frac{1}{n}\sum_i h_i$, then the representation power is strong, but we may have difficulty in training, i.e, solving this optimization problem; if we choose $\gamma = 1$ and $\beta = 0$, it will lead to the weak representation power. Therefore, the choice of γ, β is a trade-off.

Remark 4.8. An reasonable mini-bacth size is needed to calculate μ, σ in each single bacth, i.e., $B \geq 16, 32, \ldots$. For non-ImageNet tasks, it will be an issue on how to choose B.

Remark 4.9. Besides BN, there are other types of normalization methods:

- Layer normalization; (which is standard in language processing)
- Instance normalization;
- Weight normalization;
- Group normalization; (Wu and He, 2018)
- Column normalization. The intuition is that there are lots of headache come from normalizing rows, it may be better to normalize columns instead. We skip the discussion for column normalization in this lecture.

4.4 ResNet

Recommended Reading: (He *et al.*, 2016).

Motivation It is intuitive that more layers for neural networks will lead smaller traning error. However, some numerical experiments show it is not true in general:

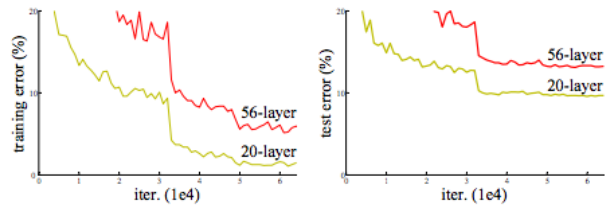


Figure 4.3: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is also observed.

Let's analysis what is the most possible reason leading to this phenomena. Let's introduce few terminologies first. In data science, the testing error can be separated into three types of errors:

- Representation Error, which is related to the representation power of the fitting model.
- Optimziation Error, which is related to how well we minimize the loss function.
- Generalization Error, which is related to the over-fitting issue.

In Fig. 4.3 we observe that the training error for 56-layer network is larger. There are two factors related to the training error:

$$\text{Tranining error} = \begin{cases} \text{Representation Error} + \\ \quad \text{Optimziation Error} \end{cases}$$

It seems that the high training error for deeper neural network is because of the optimziation error, since deeper neural network admits stronger power in representation.

The optimization error for solving $\min_\theta F(\theta)$ using iteration formula can also be decomposed into two types of errors:

$$\underbrace{[F(\hat{\theta}) - F^*]}_{\text{Optimization Error}} = \underbrace{[F(\hat{\theta}) - F(\theta^\infty)]}_{\text{Convergence Error}} + \underbrace{[F(\theta^\infty) - F^*]}_{\text{Global-optimality Gap}}$$

Here θ^∞ is the solution we can obtain if given ∞ many iterations, i.e., the limit point where the optimiztion algorithm converge to.

- If $F(\theta^\infty) - F^*$ is large, one often needs reformulation of the original problem and smart initialization. Check (Frankle and Carbin, 2019) with its comments (*How to comment the paper "The Lottery Ticket Hypothesis"* n.d.) in Zhihu.
- If $F(\hat{\theta}) - F(\theta^\infty)$ is large, one often improves his algorithm to accelerate, such as momentum-based acceleration.

It seems that the global-optimality gap is large for our training, and therefore some smart initialization is needed. Moreover, one may ask does 56-layer have more "representation power" than 20-layer networks? Not neceesarily, unless the extra 36-layer can be approximated to the identity operator.

It's known that the deep neural network is sensitivity to initialization, i.e., we can only travel a small region around the initialization. Therefore, it seems that the high training error for deep neural network is because that we may never travel have chance to travel to the identity operator for the extra 36-layer operations.

Solution The solution for solving this issue is to design the architecture to recover the identity operator. The Resnet is designed:

$$f_\theta(x) = \mathcal{F}(x, \{W_i\}) + x,$$

where x is the input vectors of the layers. More generally, for same-width network in the layer $\ell = 1, \ldots, L$,

$$z^\ell = \phi(W^\ell z^{\ell-1}) + z^{\ell-1},$$

Remark 4.10. The dimension of input and output does not always match. Two tricks may be needed:

- Perform a linear projection W_s by the shortcut connections to match the dimensions:

$$f_\theta(x) = \mathcal{F}(x, \{W_i\}) + W_s x.$$

- Use pooling to change the dimension.

Remark 4.11. ResNet uses 2-layer and 3-layer net as residual module.

Remark 4.12. Motivated by LSTM, the paper (Srivastava *et al.*, 2015) uses "gate" to resolve this issue, but their performance is no better than ResNet.

References

Billingsley, P. (1986). *Probability and Measure*. Second. John Wiley and Sons.

Frankle, J. and M. Carbin (2019). "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rJl-b3RcF7.

Gilboa, D., B. Chang, M. Chen, G. Yang, S. S. Schoenholz, E. H. Chi, and J. Pennington (2019). "Dynamical Isometry and a Mean Field Theory of LSTMs and GRUs". *CoRR*. abs/1901.08987. arXiv: 1901.08987. URL: http://arxiv.org/abs/1901.08987.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS?10). Society for Artificial Intelligence and Statistics*.

Hanin, B. and D. Rolnick (2018). "How to Start Training: The Effect of Initialization and Architecture". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc. 571–581. URL: http://papers.nips.cc/paper/7338-how-to-start-training-the-effect-of-initialization-and-architecture.pdf.

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep Residual Learning for Image Recognition". In: 770–778. DOI: 10.1109/CVPR.2016.90.

"How to comment the paper "The Lottery Ticket Hypothesis"" (n.d.). https://www.zhihu.com/question/323214798. Accessed: 2019-08-14.

Li, P. and P.-M. Nguyen (2019). "On Random Deep Weight-Tied Autoencoders: Exact Asymptotic Analysis, Phase Transitions, and Implications to Training". In: *International Conference on Learning Representations*.

Pennington, J., S. S. Schoenholz, and S. Ganguli (2017). "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 4785–4795.

Pennington, J., S. S. Schoenholz, and S. Ganguli (2018). "The Emergence of Spectral Universality in Deep Networks". In: *AISTATS*.

Poole, B., S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli (2016). "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc. 3360–3368. URL: http://papers.nips.cc/paper/6322-exponential-expressivity-in-deep-neural-networks-through-transient-chaos.pdf.

Saxe, A. M., J. L. Mcclelland, and S. Ganguli (2014). "Exact solutions to the nonlinear dynamics of learning in deep linear neural network". In: *In International Conference on Learning Representations*.

Srivastava, R. K., K. Greff, and J. Schmidhuber (2015). "Highway Networks". cite arxiv:1505.00387Comment: 6 pages, 2 figures. Presented at ICML 2015 Deep Learning workshop. Full paper is at arXiv:1507.06228. URL: http://arxiv.org/abs/1505.00387.

Wu, Y. and K. He (2018). "Group Normalization". In: *The European Conference on Computer Vision (ECCV)*.

Xiao, L., Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington (2018). "Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. *Proceedings of Machine Learning Research*. Stockholmsmassan, Stockholm Sweden: PMLR. 5393–5402.

Zhang, H., Y. N. Dauphin, and T. Ma (2019). “Residual Learning Without Normalization via Better Initialization”. In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=H1gsz30cKX.