What Do Neural Networks Learn When Trained With Random Labels?
Summary of paper by Google Research, Brain Team, Zurich

EECS 6127, Fall 2020
Final Report
Prof. Ruth Urner
By Peter Caruana and SaiKiran Tedla

One technique gaining popularity in the field of Machine Learning is the use of randomly assigned labels to pre-train a network. The obvious question that strikes someone who first comes across this is: "How can anything be learned from random labeling?". At surface value it seems contradictory to the very idea of supervised learning, namely the idea that you are trying to learn to generalize on a set of labeled data. Surprisingly, using random labels does actually offer some distinct advantages. One problem facing data scientists is the monumental task of labeling the enormous collections of data collected every day. For most of this data to be of any use, countless man hours must be put in. Some of the most impactful developments recently have simply been the creation of large labeled datasets like *Sports-1M* or *ImageNet*. With the vast amounts of data collected today, labeling an entire dataset is often simply impractical. Random labeling however, takes relatively little effort and no human input.

In the paper by Google's Brain Team aims to answer the question, what exactly do neural networks learn from random labeling? The topic of how neural networks learn in general is still being explored. Inherent over-parameterization is built-in to the structure of neural networks. Theory predicts that because of this, they should not generalize well, yet in practice the opposite is true. One reason to study random labeling is to gain further insight into the mechanisms behind neural network generalization. Understanding these techniques can help explain the critical stages of learning, as well as the mechanisms behind transfer learning. We know that in a neural network design, the earlier layers encode a useful structure of the data, whereas the later layers tend to specialize. It turns out that this is true even when training on random labels. Studies have shown through experiments that pre-training neural networks on random labelings can yield faster downstream training times. The brain team researchers showed that the useful structure encoded by neural networks in these scenarios are the principle values of the datasets, or eigenvalues. They present the theory behind this notion, as well as an implementable way to measure this effect.

Firstly, they define the problem of training on random labels. Given probability distribution D over $X \subset \mathbb{R}^d$, and finite set Y, randomly sample i.i.d. Instances $\{x_1, x_2, \ldots x_N\} \sim D$ and i.i.d. labels $\{y_1, y_2, \ldots y_N\} \in Y$. A network is trained using stochastic gradient descent (SGD) with weights randomly initialized. To understand the critical components of a set, they then introduce the notions of covariance and eigenvalues. (From p4) For input vector x, let $\mu_x = E[x]$ with covariance matrix $\Sigma_x = E[(x - \mu_x) \cdot (x - \mu_x)^T]$ where $\Sigma_x \in \mathbb{R}^{d \times d}$. It can be said that Σ_x is a

symmetric positive semi-definite matrix, which means that for all vectors $z \neq 0$, the statement $z \Sigma_x z \geq 0$ holds. The implication of this is that there exists an **orthogonal decomposition** $\mathbb{R}^d = V_1 \oplus ... \oplus V_k$ where each V_i is an **eigenspace** of Σ_x , with corresponding distinct **eigenvalue** σ_i^2 .Next, the idea of alignment is formalized (Section 2.1, p4, def 1) as:

*For symmetric matrices **A** and **B**, it can be said **A** is **aligned** with **B** if they share the same **eigenvectors**.*

As a toy example, consider a dataset drawn from a normal distribution $N(0, \Sigma_x)$ and network weights drawn from a standard gaussian distribution. Looking at the weights after training with SGD, let $w \in \mathbb{R}^d$ be a random variable drawn uniformly from the first layer of weights. The following properties hold: $E[w] = 0$ and $\Sigma_w = E[w \cdot w^T]$ is aligned with Σ_x (proof can be found in Appendix C of the original paper). This means that the principle components of the weights are aligned with those of the dataset, proving that this can happen for *some* datasets.

In practice real data is not drawn from a perfect normal distribution, so it is important to define some measure of **misalignment**. For positive definite matrices ***A*** and ***B***, they proposed the following definition of misalignment (section 2.2, p5, def 2):

$$M(A, B) := \inf_{\Sigma} \{ \tfrac{1}{2} \operatorname{tr} (\Sigma^{-1} B + B^{-1} \Sigma) - d \}$$

Where Σ is the variable being minimized, with $\Sigma > 0$ and Σ aligned with ***A***. This can be thought of as the measure of the "distance to an aligned matrix". This definition is used because the naive approach of euclidean distance between eigenvectors is not always feasible in practice when eigenvalues may be too similar to distinguish. Looking at Figure 1, alignment for image data can be understood as visual similarity.

Recall that the eigenvectors are shared but not the eigenvalues. Thus there must be some eigenvalue mapping function between the data and the weight vectors (section 2.3, p6, def 3),

$$f : \{\sigma_1, \sigma_2 ...\} \to \mathbb{R}, \ \sigma_i \mapsto \sqrt{v_i^T \cdot \Sigma_w \cdot v_i}$$

Where σ_i^2 are eigenvalues of Σ_x , and v_i are unit eigenvectors of Σ_w . This is done by finding the eigenvectors and eigenvalues of the data and then finding the corresponding eigenvector and values in the weights (via the misalignment) . The researchers were looking to see what these functions looked like so they concocted a couple different test scenarios. They had a simple neural network with 2 fully connected layers and plotted the eigenvalue mapping after being trained on synthetic data using random labeling (seen in Figure 2). A simple convolutional network was tested, with 1 convolutional layer and 1 fully connected layer. The main difference was that the convolutional network was trained on real data from the CIFAR-10 dataset. Figure

2 shows the eigenvalue mappings when using random labeling and real labeling. What was noticed is that the eigenvalue mappings for random and real labelings look quite similar, with the curves first increasing and then decreasing.

Next, a network was trained with either randomly or real labeled data. The objective was to look at what can be done when the covariance matrix of the weights (Σ_w) is known. The weights of the network were then re-initialized randomly using Σ_w , and trained it again (on real labels). Figure 3 shows plots of the times for training and testing. These plots show how using the covariance matrix from pre-training to initialize weights can result in a speed up in training. Additionally, it can be seen that this increase in speed applies for pre-training on both randomly labeled and real labeled data.

Knowing the eigenvalue mapping from the data to the weights, we can generate a covariance matrix that allows us to randomly re-initialize and get a speed up in training. To determine what eigenvalue mappings are useful, the Brain Team researchers looked at many eigenvalue mappings for different datasets and network configurations. Figure 4 shows some mappings that were found to be reasonably useful. Once an eigenvalue mapping is chosen, the weights for the first layer can be set, since the eigenvectors are shared between data and weights. Initialization is done by using the covariance matrix and randomly initializing the first layer. This process takes advantage of the training speedup found when using Σ_w to randomly initialize. The main difference is that pre-training is not needed for this method, making this a very efficient strategy given that you have chosen an appropriate eigenvalue mapping.

With the first layer weights, the same eigenvalue mapping can be used to initialize the second layer weights. The process is the same, treating the prior layer as the new input. The eigenvectors are shared, so the eigenvalue mapping can be used to get a covariance matrix for the second layer weights. This method is used to propagate through the network and initialize all weights. Figure 5 shows the effects on training speed, depending on how many layers are initialized this way. Researchers tested various choices for eigenvalue mappings (from Figure 4) and found that there was no significant performance increase between them. By having a simple eigenvalue mapping and knowing the eigenvectors of the data, the entire network can be initialized with weights that allow it to train quicker. Each additional layer that is initialized with this approach caused the training time to be shorter.

When strictly using pre-training, one issue which was found to arise was that of negative transfer. After pre-training, certain ReLU nodes in the network become inactive. This occurs when that node does not activate for any input in the dataset. Having inactive nodes reduces the capacity of the network to represent different models. To mitigate this effect the Brain Team researchers used networks with more width so that even with some inactive nodes, the network still has a higher representation capacity.

This paper demonstrated the possibility for impressive training speed increases, however did not discuss how using this method affects performance on the dataset itself. Improvements in training speed may not be very useful if the final trained network ends up performing much worse on the dataset. This is an additional area that will need to be explored to see if this method is feasible in practice to pre-train neural networks. Another interesting area to explore would be investigating how this effect changes when using networks with different non-linearities. All networks used in this paper had only ReLU activation functions leading to the inactive node effect. This might have been mitigated if the authors experimented with leaky ReLU functions or sigmoid non-linearities. Another area for future research is extending this method of using alignments and eigenvalue mappings to other types of datasets. Theory should extend to different data types, however experimenting on other datasets may lead to more understanding about neural network training procedure.

There are still large gaps in our understanding of how neural networks learn. As was discussed in lectures, neural networks have large numbers of parameters, yet do not overfit. Every research paper like the one discussed in this report helps us to gain better understanding in this area. In particular, the work done by Google's Brain Team illuminates what optimal training procedures for neural networks might look like, as well as demonstrates the usefulness of randomly labeled data. These insights help in the development of newer machine learning models that can exploit these useful properties. There is still much work to be done in the study of neural networks for machine learning, but the work done in this paper pushes us forward in our understanding.

Figures

Figure 1 - Eigenvectors for Data and First Layer weights on Simple convolutional network after pre-training with random labels on CIFAR-10 (256 filters, 64 fully connected)

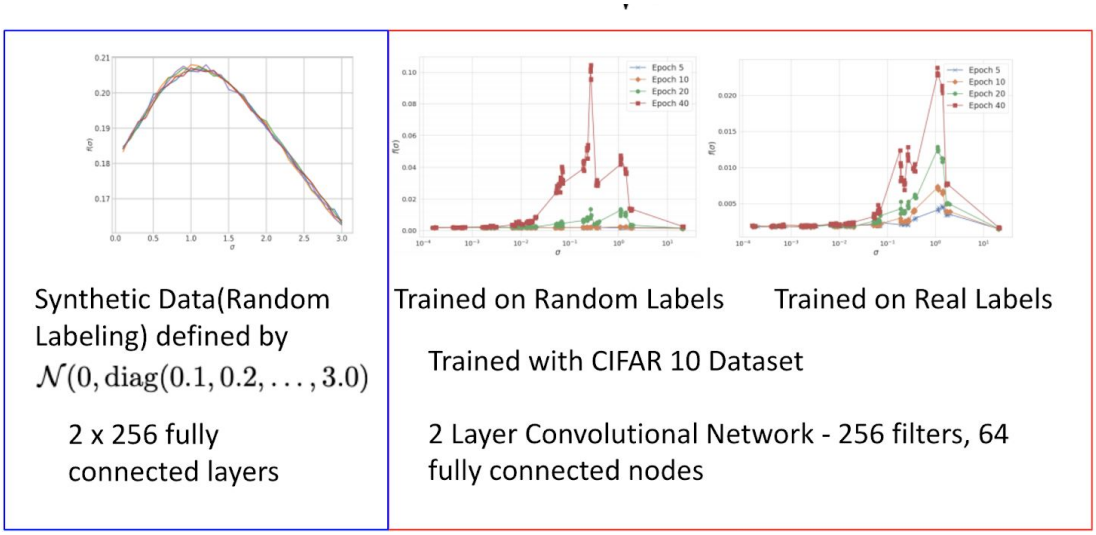

Figure 2 - Eigenvalue Mappings from different network setups

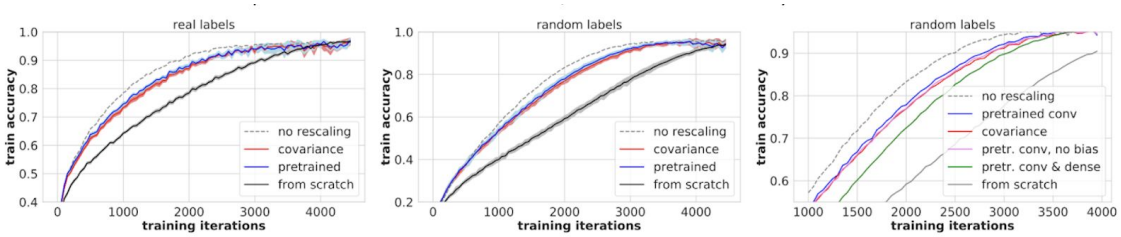

Figure 3- Results for Trials when using covariance matrix to randomly initialize network; left image is for real labels; middle and right images are the same trial but for random labels

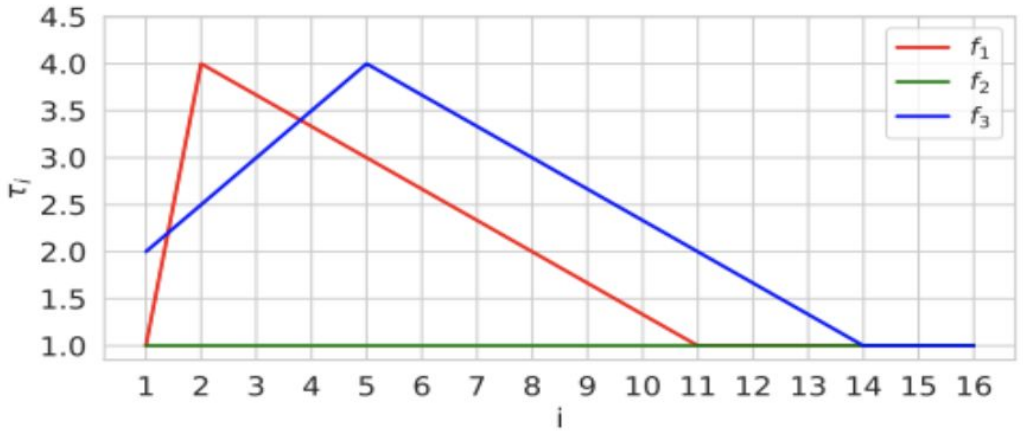

Figure 4 - Reasonable Eigenvalue Mappings Chosen by Researchers

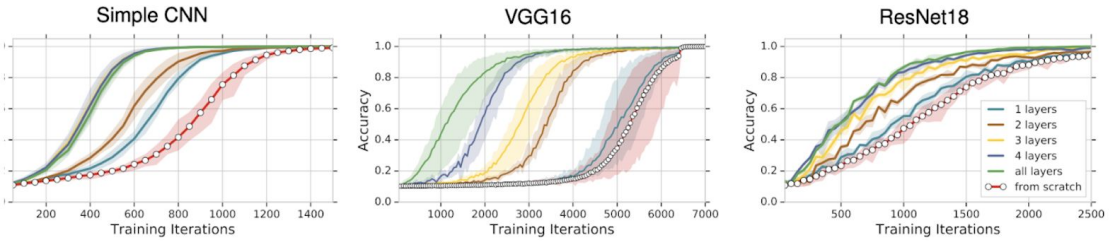

Figure 5 - Results of Layer Initialization technique on different networks

Citations

Hartmut Maennel, Ibrahim Alabdulmohsin, Ilya Tolstikhin, Robert J. N. Baldock, Olivier Bousquet, Sylvain Gelly, and Daniel Keysers. What do neural networks learn when trained with random labels?, 2020.