Big Transfer

Amber Barksdale, (azb9713) Russell Wustenberg, (rw2873)

1 Summary of Transfer Learning

Children learn at an astonishing rate due in part to a phenomenon called transfer learning. In humans, transfer learning refers to the ability to generalize across a large number of classes and accurately identify objects—or more abstract concepts—based on as few as one single example (Brown and Kane 1988). If a child is shown a picture of a cow, they will then be able to extrapolate from that one image and point to cows on a farm. The child may also point to a giraffe because both cows and giraffes have spots and four legs; based on the two prominent features of a cow, this guess aligns with Edward Thorndike's foundational theories that physical or perceptual similarity is necessary for transfer learning (Brown and Kane 1988; Thorndike 1913). Within the realm of artificial intelligence and machine learning, "transfer learning" is based on the same concept of using prior knowledge in order to identify objects or complete tasks in a novel but similar domain.

In stark contrast to humans, a traditional object-classification system requires a large number of training examples, and addresses isolated tasks (Li 2006). Depending on the dimensionality of the image representations and the specific algorithm, the number of training examples could range from hundreds to thousands, resulting in an expensive and time-consuming computation process (Li 2006). Transfer learning aims to mitigate this issue (Torrey and Shavlik 2010). To back up the claim that transfer learning does indeed improve

learning, there are three metrics used: the initial performance using only transfer learning, the amount of time it takes to fully learn the task given transferred knowledge compared to the time it takes to learn the task from scratch, and the final performance level achievable in the target task compared to learning from scratch (Torrey and Shavlik 2010). The different forms of knowledge transfer can be categorized by model parameters, feature or part sharing, or contextual information (Li 2006). Contextual information refers to the fact that objects typically do not exist by themselves but rather surrounded by other objects that interact together. Use of the environment is common to help recognize objects (Li 2006). Naturally, it has been observed that a learning agent in a real-life setting is more likely to encounter situations that require transfer learning (Torrey and Shavlik 2010).

Unfortunately, it is possible for transfer learning to backfire. This undesirable outcome is called negative transfer, and causes performance to decrease (Torrey and Shavlik 2010). Negative transfer may occur when the source task is not sufficiently related, or the relationship is not well leveraged by the transfer method (Torrey and Shavlik 2010). Transfer methods should ideally produce positive transfer and avoid negative transfer in situations where the tasks are not a good match; however, this has been shown to be difficult to achieve simultaneously (Torrey and Shavlik 2010). When safeguards are put in place to avoid negative transfer, there is a weaker positive effect, but aggressive transfer methods that produce large positive effects have no protections against negative transfer (Torrey and Shavlik 2010). Three ways of avoiding negative transfer include rejecting bad information while learning the target task, choosing the best source task, and modeling the similarity between tasks and including this information in the transfer method (Torrey and Shavlik

2010).

Transfer learning as a whole is a promising and desirable avenue in the realm of machine learning. Relevant challenges include avoiding negative transfer and automating task mapping, something easy to perform for humans but difficult to recreate in artificial intelligence (Torrey and Shavlik 2010). The field of transfer learning is also moving toward enabling transfer between diverse tasks and performing transfer with more complex tasks. Overall, transfer learning is poised to become a standard in the field of machine learning and artificial intelligence.

2 The Residual Network Structure of *Big Transfer*

Big Transfer (BiT) is the outcome of a Google Brains research project based out of Zürich. The goal was to produce a large generalist residual neural network (ResNet) that could be cheaply adapted to so-called *downstream* specialized tasks. While developing their final models, the researchers investigated several aspects of neural network training and design, releasing their results in (Kolesnikov et al. 2020). Their choices in design optimization were driven by two main motivations: first, the massive size of the training data sets required careful consideration of hardware resources; second, the models have a high memory requirement, requiring small per-device batch sizes. Much can be learned about state-of-the-art neural network design by analysis of their design choices.

BiT consists of three trained models: BiT-S, for tasks with fewer than 20,000

training samples; BiT-M, for those with between 20,000 and 500,000; and BiT, for those with more than 500,000. It is worth noting that these labels are not correlated with the architecture size but rather the task and training data set. The creators of BiT trained all models on 'vanilla' ResNet-v2 architectures of various sizes with a number of modifications. They replaced all batch normalization (BN) layers with group normalization (GN) layers, all convolutional layers used Weight Standardization (WS), and used *mixup* for medium and large tasks (Kolesnikov et al. 2020).

The choice to use a ResNet-v2 architecture (instead of the -v1) is motivated by (He et al. 2016b). Following (He et al. 2016a), which showed that residual functions were crucial to preventing weight decay to zero in a deep convolutional neural network, (He et al. 2016b) asserts that identity mappings, and the specific sequencing of normalizations, weights, and activations, is crucial to leverage the benefits of a residual network.

In (Kolesnikov et al. 2020), experiments were performed to analyze the relationship between ResNet scale and the dataset size. To that end, ResNet-50x1, -50x3, -101x1, -101x3 and -152x4 architectures were trained on various image classification tasks and their performance measured. A salient point was discovered that smaller models, given sufficiently large data sets, performed *worse* than if they had been trained on a smaller data set. This has potential impact on past research that thought to benefit their model with a bounty of data without adjusting scale proportionally. Conversely, larger architectures performed better on all tasks, especially on severely constrained batch sizes (Kolesnikov et al. 2020).

The choice to exchange BN for GN was made in part due to hardware constraints

(Kolesnikov et al. 2020). The per-unit memory constraints of the more expensive ResNet architectures caused the maximum batch size to be set low. BN has been shown to perform poorly under such conditions as well as requiring costly synchronization in parallel processing environments (Ioffe 2017). GN, conversely, thrives in low batch conditions and is ideal for a parallel processing environment (Wu and He 2018). As shown in (Zhang et al. 2017), the combination of GN with WS extends data normalization to the learning space and greatly reduces training time.

With the above normalizations in place, it may come as some surprise that the designers forwent the common regularization of dropout, weight decay to initial parameters or weight decay to zero (Kolesnikov et al. 2020). The designers claim that by using a sufficiently long training schedule a model will naturally converge to a loss threshold that is as though it were regularized. It should be noted, for those following along at home, that at least one of their larger models was allowed to run for 8 GPU *months* to show sufficient results. This approach may not be tractable for all contexts (Kilcher 2020).

Finally, it is common to scale up the resolution of the image set between the training and test phases. Instead, (Kolesnikov et al. 2020) advocates adding an alternative step: instead of being treated as an extension of the training set, the generalist model is fine-tuned to the test data set as though it were a specialist task. This emulates the transfer learning process as a whole, showing the model works as intended, and includes the increase in data resolution.

3 Three Applications of Normalization for NNs

Normalization has become a key technique for obtaining optimal learning rates, but there are many variants that to the neophyte may be confusing. Normalization has been shown to ease optimization, speed up training, decrease the importance of initial weight and smooth the loss landscape (Santurkar et al. 2019; Wu and He 2018). Normalization is the redistribution of a set of data points in such a way that certain qualities are easier to distinguish when analyzing the set. Before normalization, a data set may consist of any set of values conceivable. After, all data points will exist between a set of known bounds, typically between 0 and 1. Most importantly, no information is lost in this transformation because the proportionality of each point to all other points is maintained. This is accomplished by treating each point as existing at a given *variance* from the *mean* of the data.

In a perfect world, the whole dataset would be fed into the network all at once. In reality, dataset sizes have now exceeded what one single machine could hold in memory at any time and the financial burden of operating such a machine would be prohibitively expensive. Batch Normalization (BN) is a natural solution to this problem. In BN, a number of data points are randomly pulled from the set of unprocessed data and normalized against each other. While the mean of the batch only approximates the mean of the whole dataset, successive batches will slowly converge upon the true value. This has the benefit of requiring fewer data samples to live in memory at one time and less 'up front' computation cost.

A neural network needs normalization not only before processing but internally at every node. As data passes through the network, each layer performs a function upon it,

extracting information that will contribute to its final analysis. This has the unwanted effect of de-normalizing data at each internal layer. The researchers who developed BN showed that the most popular learning mechanism for neural networks, stochastic gradient descent, suffers when this *internal covariate shift* occurs. The solution is to introduce normalization before each computational step. If each *unit* within a version 2 residual network (ResNet-v2) can be generalized as

$$x_{out} = Residual(x_{in}) + Weight(\ Activation(\ BN(\ x_{in})))^1$$

Then the output of each unit, x_{out}, will be computed by taking the input from the previous layer, x_{in}, normalizing the data, feeding it into an activation function, and applying the weight matrix to all values which were above the activation threshold. This right-side term will be repeated for every convolutional layer in this unit. Finally, the residual function, which is often simply the un-normalized value of x_{in}, is added to the result and passed as to the next layer.

During the learning phase of the training process, the degree of inaccuracy of the neural network's final calculation is measured. This is interchangeably referred to as *loss* or *error*. The network is informed of how correct and/or incorrect its final answer was, *propagates* this score back through to each unit and adjusts its calculations accordingly. Since we are using a ResNet as the basis of our example, the amount of loss for each layer along an input pathway will travel along the *skip connections* between residual units and deliver that unit's contribution to the overall error of the system. The left-side term of our hypothetical unit allows a continuous stream of propagation backwards into the network

[1]Equation taken from (He et al. 2016a).

while the right-side term is used to adjust the local calculations. Each unit computes its own error with respect to this overall score. Because the input into each layer is normalized, this process can be done faster and the calculations made more confidently by the system (Ioffe and Szegedy 2015).

In a neural network where the data, the network architecture, and the computation hardware are all on one device, batch normalization works well. In many ways, it is the 'gold standard' for normalization techniques. However, it is not without its flaws. BN performs poorly when the batch size is small because error increases as the batch size becomes smaller, due to inaccurate batch statistics estimation (Wu and He 2018). Many methods were proposed to resolve this issue, including (Ioffe 2017; Ren et al. 2017; Ulyanov et al. 2016). While some success was gained, all came at a cost to the overall performance of the network. That BN needs a sufficiently large batch results in high memory consumption, preventing researchers from exploring higher-capacity models that are limited by memory (Wu and He 2018). This limits BN's use in training larger models and for use in computer vision tasks such as detection and segmentation as these tasks have data sets with significant per-sample memory requirements (Wu and He 2018). Finally, as network architectures have grown larger, there has been increased need for handling asynchrony across distributed systems. BN, with its requirement for batch synchronization at every layer, incurs inter-device synchronization cost (Kolesnikov et al. 2020).

Group Normalization, GN, is the synthesis of approaches to normalization that shows equal performance to BN while batch sizes are of moderate size and *maintains* that level of performance even down to a batch size of one [Wu 2018]. GN grew out of a series of

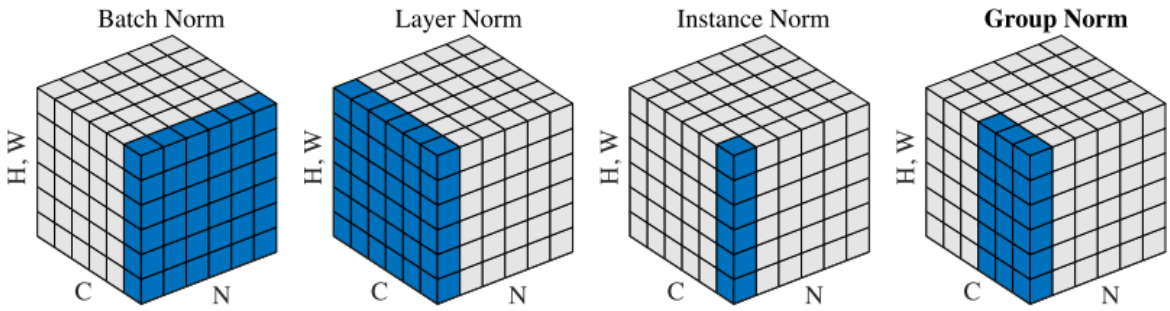

Figure 1: Cuboid illustrations of feature map tensors where N is the batch axis, C is the channel axis, and (H, W) are the spatial axes. Pixels in blue are normalized by the same mean and variance.[3]

---

redefinitions of what is normalized within the batch, in particular Layer Normalization (LN) and Instance Normalization (IN). Figure 1 shows illustrations comparing BN, LN, IN and GN. Each cuboid shape is composed of three axes. H, W represents an image input that has been flattened into a one-dimensional array of pixel values. C represents the number of channels in a given convolutional layer. An image input to the system will begin with a default of 3 channels—red, green, and blue—and more will develop as the channel input is convolved in future network layers. N represents the number of samples. As can be seen, BN calculates the mean and variance of all pixel values for a *single* channel across the entire batch of size N.

LN proposed to solve this issue by limiting its scope to only a single data sample but normalizing all points across all channels of that sample. The flaw in this method is that LN will take a complex sample, such as an image, and find one *universal* mean and variance for the entire image. This is computationally expensive and does not permit sensitivity to variance between channels (Wu and He 2018). The creators of IN, seeking to maintain

the independence of the channels, normalized each channel of a data sample independently. For example, IN would only calculate the mean and variance of only the red, green or blue channel of an image. Ultimately, neither LN nor IN had been able to approach BN's accuracy in many visual tasks, but the creators saw them as a point of inspiration for the design of GN.

GN maintains the innovation of only utilizing a single sample as the basis for normalization. This choice makes it agnostic to batch size and, theoretically, it will perform as well on a single-sample batch as on a large batch. Rather than an 'all-or-nothing' approach on channels, the algorithm divides the channels into a number of groups determined by a hyperparameter, Γ. In (Wu and He 2018) the default value is 32. This hyperparameter allows for experimentation during training. When $\Gamma = 1$ all channels are treated as in LN; when $\Gamma = C$, where C is the number of channels, then GN becomes IN (Wu and He 2018). In combination with weight standardization, it has been shown to be useful in transfer learning applications and to even outperform BN under certain conditions (Kolesnikov et al. 2020).

Normalization can also be applied to the weight matrices themselves, and doing so is extremely beneficial for learning. Deep neural networks without some form of weight normalization struggle to learn for a number of theorized reasons (Huang, 2017). Several attempts were made to combat this, including research into weight initialization values (Glorot and Bengio 2010), learned weight normalization vectors (Salimans and Kingma 2016) and the ominously named *pathological curvature* (Huang et al. 2017).

By far the most successful is referred to as Weight Standardization (WS) which

was developed to avoid *singularities* in the neural network and guarantee a *smoothness* of values. Smoothness, in the context of a deep neural network, refers to the ease with which the network will be able to discover the way to adjust the weights in respect to its loss. When a network is unsmooth, the learning rate must be set low or else the network may make a wrong choice in adjusting its weights. Singularities occur when a weight approaches, equals, or is less than zero. Singularities cause a network's learning rate to drop rapidly (Wei and Amari 2008).

The weights, as with the data, will begin as a set of arbitrary values and change over the course of training. If left unregulated, the network will struggle to learn. To solve this issue, WS separates the actual weight values with a matrix of normalized weights, which are calculated on the original values. This avoids the risk of singularity as it is not the specific weight values but rather their distribution that matters to the calculations. Each pass of the learning algorithm will adjust the distribution of weights rather than their raw values.

In essence, BN, GN and WS are all extensions of the same principle for deep neural learning. As has been shown, neural networks universally benefit when inputs are drawn from a smooth sample space. When working with data, batch normalization has proven to be a successful technique. Group normalization, seeking to handle contexts in which BN behaves poorly, is especially beneficial when batch sizes are small and data may only be partially in memory. Weight standardization, inspired by the application of normalization in data space, applies normalization to the weight matrix. Variations exist on these techniques, and research is ongoing to optimize deep neural networks as increasing layers and complex

learning topologies emerge.

4 Summary of *mixup* for NN Regularization

As was seen in the previous section, deep neural networks benefit greatly from smooth topologies in data and learning space. Most often, machine learning designers achieve smoothness by normalization. In the evolution of batch, layer, instance and group normalization techniques, several attempts must be made to find an optimal normalization approach given practical constraints. *mixup* is a method of data normalization shown to have an impact on a number of areas requiring regularization (Carratino et al. 2020). *mixup* recontextualizes the data provided to the network by randomly interpolating virtual data points between known data points (Zhang et al. 2017). Following *mixup*, a network is able to evaluate its hypotheses with respect to their *vicinity* to known ground truths rather than empirical error measurements.

Traditionally in supervised learning, a neural network is provided a set of training data along with the correct labels correlating with each sample. The data is fed into the network for classification, treated as a hypothesis, and the resulting predictions compared to the labels as a ground truth. Empirical Risk Measurement (ERM) measures the distance between each hypothesis made and the ground truth and calls this distance error (Vapnik 1999). With its dependence on provided data, a network trained on unmodified ERM tends to draw strong class definitions within the sample space. This can lead to two major weaknesses in a finalized model: *memorization* and openness to *adversarial attack* (Zhang et al. 2017).

mixup proposes to solve such problems by constructing virtual training examples from those provided at training time. Using an inexpensive computation, *mixup* takes known data points with confident ground truths and interpolates virtual points with ground truths relative to their vicinity to known points. When a network hypothesizes such an ambiguous result, the network can reference the virtual point and affirm that such an interpretation has a valid basis given the data. The result is a sort of 'Gaussian noise' where data is seen as a smooth probabilistic landscape rather than a set of discrete definitions (Zhang et al. 2017).

These virtual training points are constructed using *Vicinal Risk Minimization* (VRM) (Chapelle et al. 2001). VRM translates each known data point into a probability distribution with a *vicinity of likelihood*. By sampling this distribution, the network is now attempting to learn by minimizing the *empirical vicinal risk* rather than simply *empirical risk*. VRM introduces proximity to the data, and the network can utilize this spatiality to orient itself within the data space.

Given two known data points, x_i and x_j, with known labels, y_i and y_j, the virtual point, $\tilde{x}$, and label $\tilde{y}$ is interpolated at a proportional distance, λ, between the two points. In equation form, *mixup* is written as

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

The resulting virtual point is a linear combination of the known data points and labels. When sampled by the network, its label will exist in the liminal space between the two discrete examples. When sampling the error in respect to this virtual point, EVM will now

report smaller (i.e.. smoother) adjustments back to the network. This has been shown to speed network learning and produce more confident likelihood reporting in generalization (Carratino et al. 2020).

mixup provides a number of explicit and implicit regularizations to the network. The original motivation for its creation was to alleviate overfitting during neural network training (Zhang et al. 2017). By extension, adversarial attack is mitigated as formerly deceptive samples with ambiguous content are now viewed within context and the data space is less exploitable due to harsh definition boundaries. A number of other regularizing effects have been noted, including model calibration, increased robustness to input adversarial noise, and robustness to label corruption (Thulasidasan et al. 2019; Zhang et al. 2017. Traditionally, amelioration of these issues has required separate methods, but mixup, in a deceptively intuitive procedure, cohesively regularizes the data. Variations on mixup have been researched, all showing positive impact on performance (Guo et al. 2018; Verma et al. 2019; Yun et al. 2019. Some researchers have cautioned against using the process when training set sizes are below a certain threshold (Kolesnikov et al. 2020).

References

Brown, A. L., & Kane, M. J. (1988). Preschool children can learn to transfer: Learning to learn and learning from example. *Cognitive Psychology*, *20*(4), 493–523. https://doi.org/10.1016/0010-0285(88)90014-X

Carratino, L., Cissé, M., Jenatton, R., & Vert, J. P. (2020). On Mixup Regularization. http://arxiv.org/abs/2006.06049

Chapelle, O., Weston, J., Bottou, L., & Vapnik, V. (2001). Vicinal risk minimization. *Advances in Neural Information Processing Systems*.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, *9*, 249–256. http://www.iro.umontreal.

Guo, H., Mao, Y., & Zhang, R. (2018). Mixup as locally linear out-of-manifold regularization. www.aaai.org

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Identity mappings in deep residual networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9908 LNCS*, 630–645. https://doi.org/10.1007/978-3-319-46493-0{_}38

Huang, L., Liu, X., Liu, Y., Lang, B., & Tao, D. (2017). Centered Weight Normalization in Accelerating Training of Deep Neural Networks. *Proceedings of the IEEE Interna-*

tional Conference on Computer Vision, *2017-Octob*, 2822–2830. https://doi.org/10.1109/ICCV.2017.305

Ioffe, S. (2017). Batch Renormalization: Towards reducing minibatch dependence in batch-normalized models. *Advances in Neural Information Processing Systems*, *2017-Decem*, 1946–1954.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, *1*, 448–456.

Kilcher, Y. (2020). Group Normalization (Paper Explained) - YouTube. https://www.youtube.com/watch?v=l_3zj6HeWUE

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., & Houlsby, N. (2020). Big Transfer (BiT): General Visual Representation Learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12350 LNCS*, 491–507. https://doi.org/10.1007/978-3-030-58558-7{_}29

Li, F.-F. (2006). Knowledge transfer in learning to recognize visual objects classes. *Proceedings of the Fifth International Conference ...* http://www-cs.stanford.edu/groups/vision/documents/Fei-Fei_ICDL2006.pdf

Ren, M., Liao, R., Urtasun, R., Sinz, F. H., & Zemel, R. S. (2017). Normalizing the normalizers: Comparing and extending network normalization schemes. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings.*

Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems*, 901–909. https://github.com/openai/weightnorm.

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2019). How does batch normalization help binary training?

Thorndike, E. (1913). *Educational Psychology. Vol. 1. The Original Nature of Man.* (Vol. Volume 1). Columbia University.

Thulasidasan, S., Chennupati1, G., Bilmes, J., Bhattacharya1, T., & Michalak, S. (2019). On mixup training: Improved calibration and predictive uncertainty for deep neural networks.

Torrey, L., & Shavlik, J. (2010). Transfer Learning. *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques2* (p. 852). IGI Global.

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance Normalization: The Missing Ingredient for Fast Stylization. https://arxiv.org/abs/1701.02096.%20http://arxiv.org/abs/1607.08022

Vapnik, V. N. (1999). An overview of statistical learning theory. https://doi.org/10.1109/72.788640

Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Lopez-Paz, D., & Bengio, Y. (2019). Manifold mixup: Better representations by interpolating hidden states. *36th International Conference on Machine Learning, ICML 2019*, *2019-June*, 11196–11205. https://github.com/vikasverma1077/manifold_mixup

Wei, H., & Amari, S. i. (2008). Dynamics of learning near singularities in radial basis function networks. *Neural Networks*, *21*(7), 989–1005. https://doi.org/10.1016/j.neunet.2008.06.017

Wu, Y., & He, K. (2018). Group Normalization. *International Journal of Computer Vision*, *128*(3), 742–755. https://github.com/facebookresearch/Detectron/

Yun, S., Han, D., Chun, S., Oh, S. J., Choe, J., & Yoo, Y. (2019). CutMix: Regularization strategy to train strong classifiers with localizable features. *Proceedings of the IEEE International Conference on Computer Vision*, *2019-Octob*, 6022–6031. https://doi.org/10.1109/ICCV.2019.00612

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2017). Mixup: Beyond empirical risk minimization. *arXiv*. http://arxiv.org/abs/1710.09412