A Symmetric Batch-Size Algorithm during Training and A Brief Introduction to Design of Experiment for Deep Learning Research

1st Enes Dilber
Department of Statistics
Middle East Technical University
Ankara, Turkey
dilber.enes@metu.edu.tr

2nd Emre Akbas
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
emre@ceng.metu.edu.tr

***Abstract*—This project paper discusses two questions in deep learning. First one is *"do changing batch size during training has an effect on generalization?"* We know that stochastic gradient descent plays an important role at generalization in deep learning. We took a closer look at configuration of vanilla SGD. Our work in stochastic gradient descent inspired by Smith and Le (2017) and Jastrzebski et al. (2017). They both suggest that optimum *batch size* is proportional to *learning rate*. We use that fact and add this statement: *we could exploit batch size during training without changing the validation accuracy*. Our implementation on adaptive batch size method uses a symmetric batch size scheduler. The second question we tried to address is *what should be the design of a scientific deep learning research?* Deep leaning research sky-rocketed after famous Alex-Net paper (Krizhevsky et al., 2012). Since current deep learning research continue to grow empirically, we suggest a simple *design of experiment setting* which uses *factorial design*. It is the most efficient experimental research method for researchers who want to analyze effect of the different factors to the model performance.**

***Index Terms*—Deep Learning, Adaptive Batch Size, Learning Rate Scheduler, Batch Size Scheduler, Design of Experiment, Factorial Design**

I. Introduction

Deep Learning models achieved remarkable success in machine learning literature. In contrast to its practical usage, the theory behind it very shallow. However there are researchers who discuss capabilities and the limits of deep learning. Gal (2016) discussed model uncertainties in A.I safety perspective of deep learning. It is doubtful that generalization capabilities of deep learning models are realistic in validation accuracy on benchmark data. State-of-the-art models should be trained challenging datasets and evaluation should be done by *Statistical Design of Experiment* setup. On the other hand, benchmark datasets are suitable for testing theoretical approaches. Since classical statistical generalization theory methods such as Radamacher Complexity and Uniform Stability are not suitable for deep learning, researchers developed new understating methods such as broadness of minima. However those methods criticized because of several reasons. First of all, the definition of broadness is not clear. Second, most of the measures are model and/or data dependent. A universal understanding is lacking in those settings. Lastly, the *loss* surface of deep rectified networks is not invariant to parametirazation due to non-homogeneous property of its layers.

A. Classical Statistical Learning Theory and Deep Learning

Memorizing, given facts, is an obvious task in learning. This can be done by storing the input samples explicitly. On the other hand, generalization is the ability to identify the rules, "to generalize", allows the system to make predictions on unknown data. The main interest of all "Machine Learning" methods is to make successful prediction on unseen data. However Zhang et al. (2016) showed that a state-of-the-art, carefully regularized neural network could memorize a training set with random labels and naturally obtains the non-informative rate at test set. After that, understanding of capacity in deep learning has changed. Any regularization method could not be able to stop model to memorize random labels. In contrast, A model with such capacity could generalize well with true labels. The classical complexity and stability measures in statistical learning theory failed to understand these results.

After all, those models are very successful at true labeled data. Several researchers showed that, Deep Neural Networks, first learn the meaningful structure and then memorize (Arpit et al., 2017). In fact, one could view deep neural networks as shortest path finders. They find a way to minimize the loss, but they seek to find it in a shortest, hence meaningful way. However those are all empirical results. It is the fact that Generalization in Deep Learning phenomena not been understandable by current statistical learning approach. Kawaguchi et al. (2017) summarizes "why deep learning models generalize well" as: *we can generalize well because we can obtain a good model via model search with validation*. We are randomly traveling through the hyper parameter space of deep learning, with brute force and the help of our experience, seek a good setting.

B. Batch-Size in Stochastic Gradient Descent

Since generalization in deep learning is a rich topic. We are going to only focus on "how batch size in SGD effect generalization". SGD and its Newtonian variants are at the heart of every Deep Learning training scheme. We are going to take a look at the *batch size-learning rate* dynamics. The common belief in Deep Learning community is "models that are trained by small batches find a broad minima", which could generalize well. Researchers believe, model could not trainable with large batches. Or it took very long to train. However Dinh et al. (2017) showed that sharp minima could also generalize. Other researchers even questioned the success of small batches and trained models with large batches (Jastrzebski et al., 2017; Smith and Le, 2017). While training models with large batches, they introduced *noise scale*, which has an optimum value that is obtained by trial-error scheme.

Definition 1: Noise scale g controls the scale of random fluctuations that is introduced to the gradient update system by Stochastic Gradient Descent.

$$g \propto \epsilon \frac{N}{B} \ where;$$

$\epsilon = learning\ rate$
$N = total\ number\ of\ observations$
$B = batch\ size$

C. Our Contribution

In this paper, we are going to suggest two different view-point of Generalization in Deep Learning:

- We are going to investigate the effect of batch size during training with an adaptive algorithm which uses a symmetric scheduler. In other words, model training starts with a small batch size, increases during training and then starts to decrease until the final epoch. Which gives an insight about, how model performance changes with a dynamic batch size.
- The current Deep Learning researches lack of statistical significance. Most of them acquire "unrepeatable and/or incomparable" results. Or the hyper parameter seeking done by trial and error. We are going to introduce a *classical* experimental setting for a deep learning research and illustrate it with an example.

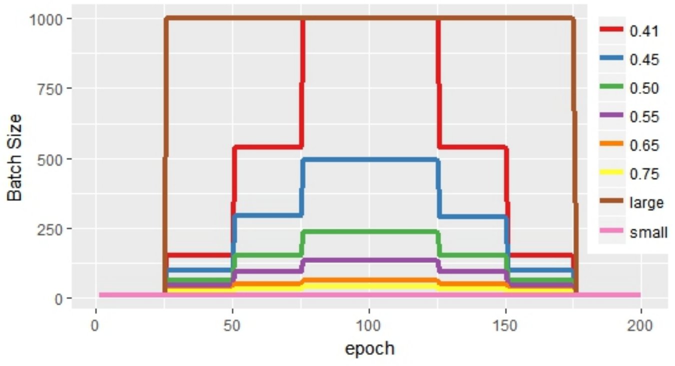

Fig. 1. Schedule of Batch Size through the Epochs

II. An Adaptive Batch-Size Algorithm

In this section, we compare several symmetric batch size approaches. Our aim is to show, *it does not matter whether we use a small or large batch in middle epochs, if we start and end with a small batch.*

A. Symmetric Batch Size

To work on full-batch, we sampled 1000 observations from $CIFAR10$ and evaluated model on full test set. Our model architecture was a very shallow VGG-like network (6) with Batch Normalization. We keep architecture as simple as possible, since our interest is "*how generalization capability of a model changes when we change the batch size*". In this experiment, we inspired by the cyclical learning rate experiment of Jastrzebski et al. (2017). Since they simultaneously scale the learning rate with batch size, both changes in a cycle. They suggest that cyclical schemes oscillate between sharp and wide regions of the parameter space and lands on wider minima at the same loss value.

In our experiment, we investigate a symmetric batch size scheduler. One can see our scheduling scheme at Figure 1. Their names indicate the deviation of the batch size. We are investigating, *if we start with same batch size and end the experiment with same batch size, do the batch size at the middle steps matter?* In all our runs, we started with *batch size = 10* until 26^{th} epoch. And, ended our training with again *batch size = 10* in last 25 epochs. As related papers suggested, we changed the learning rate proportional to batch size and we only used vanilla SGD.

We ran this symmetric batch-size scheme a couple of times and the results were very similar. Quite interestingly, all runs that uses symmetric batch-size landed on a minima that have similar validation performances (Figure 2). However, $large$ performed significantly bad compared to other methods. Similarly it starts and ends training with *batch size = 10*. However at the middle steps of the training it sticks at a local

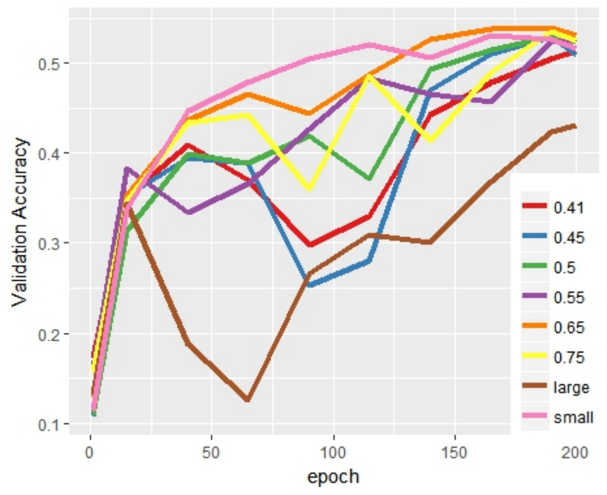

Fig. 2. Validation Accuracy of Different Symmetric Batch Size Settings

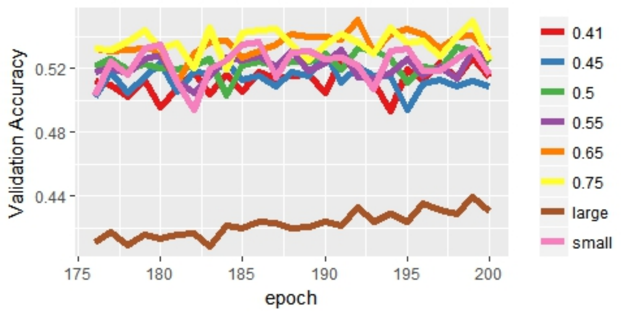

Fig. 3. Validation Accuracy between epoch:176 and epoch:200

optima. In fact, it is very hard to return from that point. We continue to train it 100 epochs more with small batch size, it could only reached to 49% accuracy. On the other hand, Setting:0.41 also trained with the full batch between epochs 75 and 125, it is still be able to land on a good minima. Although we do not have any intuition why this scheme holds, with right adaptation, model performance recover from bad local optima.

B. Step vs Continuous Change in Symmetric Batch Size

Dramatic changes in batch-size negatively effect the generalization performance. Our symmetric scheme works as good as small batch training, because it does not make dramatic changes in batch size. As an experiment, We trained the model with symmetric batch size scheme while changing the batch size continuously. In addition, we shuffled the batch size values of continuous symmetric scheme while keeping first and last 25 epochs same. One could see our batch size scheme at Figure 4.

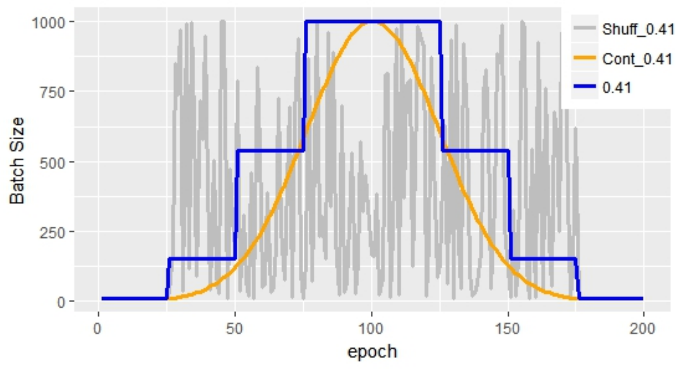

Fig. 4. Schedule of Batch Size through the Epochs

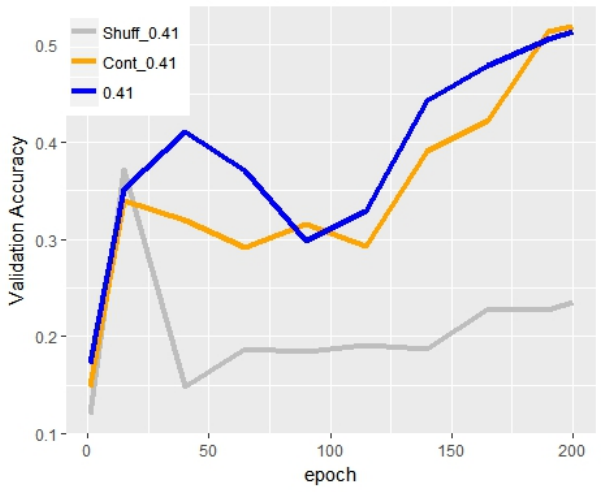

Fig. 5. Validation Accuracy of Continuous and Discrete Symmetric Batch Size Settings

We are going to investigate:

- Whether batch sizes at middle steps invariant to order or not?
- Does Changing Batch size continuously or discretely differ from each other?

We observed that Continuous and Step settings worked similar (Figure 5). However shuffled batch sizes preformed significantly bad. Even if we change the batch size to a small one, it could not be able to recover from bad local minima. It performed worse than $large$ batch training at Figure 2.

III. Design of Experiment for Deep Learning Research

Beside their high amount of weights, Deep Learning models also have a lot of hyper-parameters and custom settings. Day by day a new paper suggests a new training algorithm like we did at previous section. Most of them run their model with custom settings, custom seeds and construct some primitive confidence intervals with them. We over-viewed generalization in Deep Learning, and gave a summary about current understanding of it. However, as we mentioned, theory behind Deep Learning a little bit slow for now. We are not sure it will enlighten their success or not. We are not sure that in the future, we find an optimum regularization method or optimum training method with a mathematical proof. On the other hand, we do not have to approach deep learning models as they are 'alchemy', either. Until the theory gives us a solid proof about "mysteries of Deep Learning", we could continue research with a "solid" experimental setting, *statistical experimental design*.

The trial and error scheme in model architecture phase has been worked by several researchers. For instance Klein et al. (2016) suggested a Bayesian neural networks for modelling the learning curves of iterative machine learning methods. They train a joint model across hyper-parameters and time steps. That allows to make predictions for completely unobserved learning curves of new configurations. However we view this trial-error scheme in a different perspective. Beside the probabilistic framework, we could evaluate model settings with a statistical significance. In this section we are going to create a mock experimental setting for a deep learning model. It does not have any relation with section 2.

A. Experiment Setup

Suppose we have a traditional deep neural network architecture that have the shape of Figure 6, trained by stochastic gradient descent with $learning\ rate = 0.01$ and $momentum = 0.9$ and $batch\ size = 16$. As activation function model uses $ReLU$. Now we curios about whether below settings improve our validation performance or not:

- Change ReLU with Swish (2017)
- Change batch-size (16) with a larger one (128)
- Change SGD with Adam
- Add Batch Normalization

B. Factorial Design in Deep Learning Experiments

Complete factorial design provides the smallest number of runs which k factors can be studied (Montgomery, 2017). Suppose $ReLU$ and SGD are default settings. We want to study whether $Adam$ and/or $Swish$ makes model performs

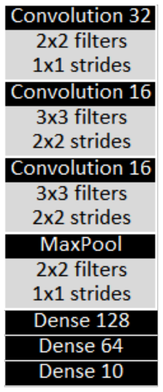

Fig. 6. Model Architecture

better or not. In first run we obtained the result of the current setting and one at a time we changed the factor levels.

Run	$Activation$	$Optimization$
1	$ReLU$	SGD
2	$ReLU$	$Adam$
3	$Swish$	SGD

To get interpretable results, we repeated the runs four times. In total 12 runs we obtained the effects of two factors. However one could do this experiment more efficiently with factorial design. If we obtain the factorial table.

Run	$Activation$	$Optimization$
1	$ReLU$	SGD
2	$ReLU$	$Adam$
3	$Swish$	SGD
4	$Swish$	$Adam$

In factorial table, for instance, effect of $Swish$ could be estimated by the difference between $Run : 3$, $Run : 1$ and $Run : 4$, $Run : 2$. Therefore, if we repeat these runs 2 times, we obtain 4 observations for every main factors. In total 8 runs, we obtained the same main effect information with changing the level of a factor one at a time. In addition we obtained interaction information.

Factorial Design is very suitable for Deep Learning experiments. They are very efficient methods and widely used by other fields of science where the experiment is expensive and takes long time. Furthermore, one could even implement more efficient designs by *fractional factorial design*, (of course with a cost). However this is beyond the scope of this paper.

Now let us focus on our experiment. We have four factors. Therefore we implemented a 2^4 factorial design. Suppose "0" level represents current settings and "1" represents the effect that we wanted to test. Here is our factorial design:

Run	A	O	BS	BN
1	1	1	1	1
2	0	1	1	1
3	1	0	1	1
4	0	0	1	1
5	1	1	0	1
6	0	1	0	1
7	1	0	0	1
8	0	0	0	1
9	1	1	1	0
10	0	1	1	0
11	1	0	1	0
12	0	0	1	0
13	1	1	0	0
14	0	1	0	0
15	1	0	0	0
16	0	0	0	0

A:Activation	$A_0 = ReLU$	$A_1 = Swish$
O:Optimization	$O_0 = SGD$	$O_1 = Adam$
BS:Batch Size	$BS_0 = 16$	$BS_1 = 128$
BN:Batch Normalization	$BN_O = No\ BN$	$BN_1 = BN$

We repeated this experiment three times to obtain more significant results. Also we evaluated this experiment on a subset of $CIFAR10$ dataset. Last but not least, we randomize our runs to obtain a **completely randomized design**.

C. Results and Significance Test

Here we run our experiment with different settings and repeated it three times. We obtained validation accuracy at the end of 100^{th} epoch. It varies between 0.5 and 0.65. Indeed our factors creates variation. Now we are going to make inference on level changes.

Run	A	0	BS	BN	$R1$	$R2$	$R3$
1	1	1	1	1	0.5551	0.6027	0.5239
2	0	1	1	1	0.6056	0.6049	0.5946
3	1	0	1	1	0.5872	0.5731	0.5531
4	0	0	1	1	0.5556	0.5897	0.5867
5	1	1	0	1	0.6424	0.6297	0.6324
6	0	1	0	1	0.6004	0.642	0.5976
7	1	0	0	1	0.5779	0.6132	0.5999
8	0	0	0	1	0.5958	0.6364	0.6152
9	1	1	1	0	0.5544	0.537	0.5415
10	0	1	1	0	0.5346	0.5604	0.4981
11	1	0	1	0	0.5282	0.5449	0.521
12	0	0	1	0	0.5505	0.5372	0.5681
13	1	1	0	0	0.5222	0.5653	0.5506
14	0	1	0	0	0.5534	0.5112	0.5136
15	1	0	0	0	0.5275	0.5393	0.543
16	0	0	0	0	0.5342	0.5228	0.5066

In below *analysis of variance* table. We conclude that, the effect of $Batch\ Normalization$ and $Batch\ Size$ are significant. In addition, higher interactions $Activation : Batch\ Size$, $Batch\ Normalization : Batch\ Size$ and full interaction are *statistically significant*. It is quite surprising that main effect of $Activation$ is not significant, however its higher order interactions are significant.

Factors	**p value**
A	0.72
O	0.24
BN	**0**
BS	**0.01**
$A:O$	0.35
$A:BN$	0.13
$O:BN$	0.36
$A:BS$	**0.05**
$O:BS$	0.35
$BN:BS$	**0**
$A:O:BN$	0.48
$A:O:BS$	0.33
$A:BN:BS$	0.87
$O:BN:BS$	0.99
$A:O:BN:BS$	**0.03**

Since we know the significant factors, we could conclude which level of the given significant factors make improvement in validation accuracy.

1) Effect of Batch Size: Overall, increasing batch size from 16 to 128 do not improve the validation accuracy. In fact, it negatively effected.

	$\mathbf{BS_0}$	$\mathbf{BS_1}$
Mean Response :	0.573	0.559

2) Effect of Batch Normalization: Batch Normalization is quite crucial in our architecture.

	$\mathbf{BN_0}$	$\mathbf{BN_1}$
Mean Response :	0.536	0.596

3) Effect of Batch Size $\times$ Activation Function: Using large batch goes well together with SGD.

	$\mathbf{A_0}$	$\mathbf{A_1}$	
Mean Response :	0.569	0.566	$\mathbf{BS_0}$
Mean Response :	0.58	0.55	$\mathbf{BS_1}$

4) Effect of Batch Size $\times$ Batch Normalization: Using large batch goes well together with Batch Normalization.

	$\mathbf{BN_0}$	$\mathbf{BN_1}$	
Mean Response :	0.532	0.615	$\mathbf{BS_0}$
Mean Response :	0.54	0.578	$\mathbf{BS_1}$

5) Best setting: We investigate the full interaction term and obtained the highest estimated mean response. It is 0.635 and belongs to following configuration:

- Activation Function: Swish
- Batch-size: 16
- Optimization: Adam
- Batch Normalization: Present

IV. Conclusion

Just like cyclic batch size methods, our symmetric batch size method performs similar, even better than small batch training. However we still need to do research to make a certain statement. Batch size dynamics in SGD is not well questioned and there is still room for improvement. We showed that, with right setup, one could change the batch size to full batch (or a larger batch) without effecting the generalization performance. We also gave insight about how “experimental design” should be in deep learning researches. We hope our work could lead deep learning researches to a more scientific direction.

References

Arpit, D., Jastrzkbski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y., et al. (2017). A closer look at memorization in deep networks. *arXiv preprint arXiv:1706.05394*.

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. (2017). Sharp minima can generalize for deep nets. *CoRR*, abs/1703.04933.

Gal, Y. (2016). *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge.

Jastrzebski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. J. (2017). Three factors influencing minima in SGD. *CoRR*, abs/1711.04623.

Kawaguchi, K., Pack Kaelbling, L., and Bengio, Y. (2017). Generalization in Deep Learning. *ArXiv e-prints*.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. (2016). Learning curve prediction with bayesian neural networks.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Montgomery, D. C. (2017). *Design and analysis of experiments*. John Wiley & Sons.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *CoRR*, abs/1710.05941.

Smith, S. L. and Le, Q. V. (2017). A bayesian perspective on generalization and stochastic gradient descent. *CoRR*, abs/1710.06451.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.