Cyclical Learning Rates for Training Neural Networks

Leslie N. Smith
U.S. Naval Research Laboratory, Code 5514
4555 Overlook Ave., SW., Washington, D.C. 20375
leslie.smith@nrl.navy.mil

Abstract

It is known that the learning rate is the most important hyper-parameter to tune for training deep neural networks. This paper describes a new method for setting the learning rate, named cyclical learning rates, which practically eliminates the need to experimentally find the best values and schedule for the global learning rates. Instead of monotonically decreasing the learning rate, this method lets the learning rate cyclically vary between reasonable boundary values. Training with cyclical learning rates instead of fixed values achieves improved classification accuracy without a need to tune and often in fewer iterations. This paper also describes a simple way to estimate "reasonable bounds" – linearly increasing the learning rate of the network for a few epochs. In addition, cyclical learning rates are demonstrated on the CIFAR-10 and CIFAR-100 datasets with ResNets, Stochastic Depth networks, and DenseNets, and the ImageNet dataset with the AlexNet and GoogLeNet architectures. These are practical tools for everyone who trains neural networks.

1. Introduction

Deep neural networks are the basis of state-of-the-art results for image recognition [17, 23, 25], object detection [7], face recognition [26], speech recognition [8], machine translation [24], image caption generation [28], and driverless car technology [14]. However, training a deep neural network is a difficult global optimization problem.

A deep neural network is typically updated by stochastic gradient descent and the parameters θ (weights) are updated by $\theta^t = \theta^{t-1} - \epsilon_t \frac{\partial L}{\partial \theta}$, where L is a loss function and ϵ_t is the learning rate. It is well known that too small a learning rate will make a training algorithm converge slowly while too large a learning rate will make the training algorithm diverge [2]. Hence, one must experiment with a variety of learning rates and schedules.

Conventional wisdom dictates that the learning rate should be a single value that monotonically decreases during training. This paper demonstrates the surprising phenomenon that a varying learning rate during training is beneficial overall and thus proposes to let the global learning rate vary cyclically within a band of values instead of setting it to a fixed value. In addition, this cyclical learning rate (CLR) method practically eliminates the need to tune the learning rate yet achieve near optimal classification accuracy. Furthermore, unlike adaptive learning rates, the CLR methods require essentially no additional computation.

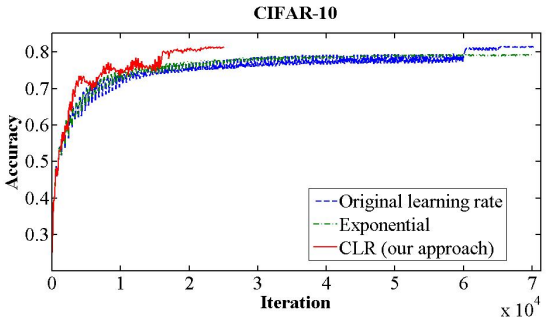

Figure 1. Classification accuracy while training CIFAR-10. The red curve shows the result of training with one of the new learning rate policies.

The potential benefits of CLR can be seen in Figure 1, which shows the test data classification accuracy of the CIFAR-10 dataset during training[1]. The baseline (blue curve) reaches a final accuracy of 81.4% after $70,000$ iterations. In contrast, it is possible to fully train the network using the CLR method instead of tuning (red curve) within 25,000 iterations and attain the same accuracy.

The contributions of this paper are:

1. A methodology for setting the global learning rates for training neural networks that eliminates the need to perform numerous experiments to find the best values and schedule with essentially no additional computation.

2. A surprising phenomenon is demonstrated - allowing

[1]Hyper-parameters and architecture were obtained in April 2015 from caffe.berkeleyvision.org/gathered/examples/cifar10.html

the learning rate to rise and fall is beneficial overall even though it might temporarily harm the network's performance.

3. Cyclical learning rates are demonstrated with ResNets, Stochastic Depth networks, and DenseNets on the CIFAR-10 and CIFAR-100 datasets, and on ImageNet with two well-known architectures: AlexNet [17] and GoogleNet [25].

2. Related work

The book "Neural Networks: Tricks of the Trade" is a terrific source of practical advice. In particular, Yoshua Bengio [2] discusses reasonable ranges for learning rates and stresses the importance of tuning the learning rate. A technical report by Breuel [3] provides guidance on a variety of hyper-parameters. There are also a numerous websites giving practical suggestions for setting the learning rates.

Adaptive learning rates: Adaptive learning rates can be considered a competitor to cyclical learning rates because one can rely on local adaptive learning rates in place of global learning rate experimentation but there is a significant computational cost in doing so. CLR does not possess this computational costs so it can be used freely.

A review of the early work on adaptive learning rates can be found in George and Powell [6]. Duchi, *et al*. [5] proposed AdaGrad, which is one of the early adaptive methods that estimates the learning rates from the gradients.

RMSProp is discussed in the slides by Geoffrey Hinton[2] [27]. RMSProp is described there as "Divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight." RMSProp is a fundamental adaptive learning rate method that others have built on.

Schaul *et al*. [22] discuss an adaptive learning rate based on a diagonal estimation of the Hessian of the gradients. One of the features of their method is that they allow their automatic method to decrease or increase the learning rate. However, their paper seems to limit the idea of increasing learning rate to non-stationary problems. On the other hand, this paper demonstrates that a schedule of increasing the learning rate is more universally valuable.

Zeiler [29] describes his AdaDelta method, which improves on AdaGrad based on two ideas: limiting the sum of squared gradients over all time to a limited window, and making the parameter update rule consistent with a units evaluation on the relationship between the update and the Hessian.

More recently, several papers have appeared on adaptive learning rates. Gulcehre and Bengio [9] propose an adaptive learning rate algorithm, called AdaSecant, that utilizes the root mean square statistics and variance of the gradients. Dauphin *et al*. [4] show that RMSProp provides a biased estimate and go on to describe another estimator, named ESGD, that is unbiased. Kingma and Lei-Ba [16] introduce Adam that is designed to combine the advantages from AdaGrad and RMSProp. Bache, *et al*. [1] propose exploiting solutions to a multi-armed bandit problem for learning rate selection. A summary and tutorial of adaptive learning rates can be found in a recent paper by Ruder [20].

Adaptive learning rates are fundamentally different from CLR policies, and CLR can be combined with adaptive learning rates, as shown in Section 4.1. In addition, CLR policies are computationally simpler than adaptive learning rates. CLR is likely most similar to the SGDR method [18] that appeared recently.

3. Optimal Learning Rates

3.1. Cyclical Learning Rates

The essence of this learning rate policy comes from the observation that increasing the learning rate might have a short term negative effect and yet achieve a longer term beneficial effect. This observation leads to the idea of letting the learning rate vary within a range of values rather than adopting a stepwise fixed or exponentially decreasing value. That is, one sets minimum and maximum boundaries and the learning rate cyclically varies between these bounds. Experiments with numerous functional forms, such as a triangular window (linear), a Welch window (parabolic) and a Hann window (sinusoidal) all produced equivalent results This led to adopting a triangular window (linearly increasing then linearly decreasing), which is illustrated in Figure 2, because it is the simplest function that incorporates this idea. The rest of this paper refers to this as the *triangular* learning rate policy.

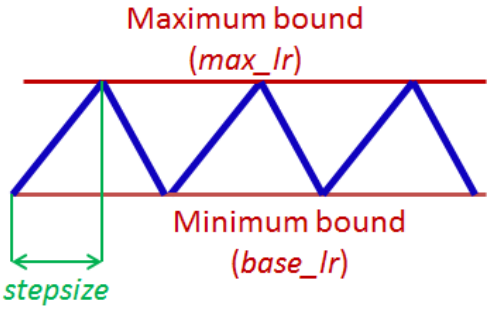

Figure 2. Triangular learning rate policy. The blue lines represent learning rate values changing between bounds. The input parameter *stepsize* is the number of iterations in half a cycle.

An intuitive understanding of why CLR methods work comes from considering the loss function topology. Dauphin *et al*. [4] argue that the difficulty in minimizing the loss arises from saddle points rather than poor local minima.

[2] www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

Saddle points have small gradients that slow the learning process. However, increasing the learning rate allows more rapid traversal of saddle point plateaus. A more practical reason as to why CLR works is that, by following the methods in Section 3.3, it is likely the optimum learning rate will be between the bounds and near optimal learning rates will be used throughout training.

The red curve in Figure 1 shows the result of the $triangular$ policy on CIFAR-10. The settings used to create the red curve were a minimum learning rate of 0.001 (as in the original parameter file) and a maximum of 0.006. Also, the cycle length (i.e., the number of iterations until the learning rate returns to the initial value) is set to $4,000$ iterations (i.e., $stepsize = 2000$) and Figure 1 shows that the accuracy peaks at the end of each cycle.

Implementation of the code for a new learning rate policy is straightforward. An example of the code added to Torch 7 in the experiments shown in Section 4.1.2 is the following few lines:

```
local cycle = math.floor(1 +
    epochCounter/(2*stepsize))
local x = math.abs(epochCounter/stepsize
    - 2*cycle + 1)
local lr = opt.LR + (maxLR - opt.LR)
    * math.max(0,  (1-x))
```

where $opt.LR$ is the specified lower (i.e., base) learning rate, $epochCounter$ is the number of epochs of training, and lr is the computed learning rate. This policy is named $triangular$ and is as described above, with two new input parameters defined: $stepsize$ (half the period or cycle length) and max_lr (the maximum learning rate boundary). This code varies the learning rate linearly between the minimum ($base\_lr$) and the maximum (max_lr).

In addition to the triangular policy, the following CLR policies are discussed in this paper:

1. $triangular2$; the same as the $triangular$ policy except the learning rate difference is cut in half at the end of each cycle. This means the learning rate difference drops after each cycle.

2. exp_range; the learning rate varies between the minimum and maximum boundaries and each boundary value declines by an exponential factor of $gamma^{iteration}$.

3.2. How can one estimate a good value for the cycle length?

The length of a cycle and the input parameter $stepsize$ can be easily computed from the number of iterations in an epoch. An epoch is calculated by dividing the number of training images by the $batchsize$ used. For example, CIFAR-10 has $50,000$ training images and the $batchsize$ is 100 so an epoch $= 50,000/100 = 500$ iterations. The final accuracy results are actually quite robust to cycle length but experiments show that it often is good to set $stepsize$ equal to $2 - 10$ times the number of iterations in an epoch. For example, setting $stepsize = 8 * epoch$ with the CIFAR-10 training run (as shown in Figure 1) only gives slightly better results than setting $stepsize = 2 * epoch$.

Furthermore, there is a certain elegance to the rhythm of these cycles and it simplifies the decision of when to drop learning rates and when to stop the current training run. Experiments show that replacing each step of a constant learning rate with at least 3 cycles trains the network weights most of the way and running for 4 or more cycles will achieve even better performance. Also, it is best to stop training at the end of a cycle, which is when the learning rate is at the minimum value and the accuracy peaks.

3.3. How can one estimate reasonable minimum and maximum boundary values?

There is a simple way to estimate reasonable minimum and maximum boundary values with one training run of the network for a few epochs. It is a "LR range test"; run your model for several epochs while letting the learning rate increase linearly between low and high LR values. This test is enormously valuable whenever you are facing a new architecture or dataset.

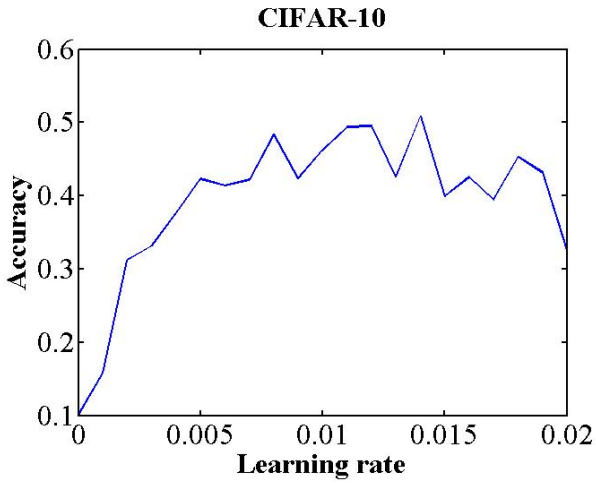

Figure 3. Classification accuracy as a function of increasing learning rate for 8 epochs (LR range test).

The $triangular$ learning rate policy provides a simple mechanism to do this. For example, in Caffe, set $base\_lr$ to the minimum value and set max_lr to the maximum value. Set both the $stepsize$ and max_iter to the same number of iterations. In this case, the learning rate will increase linearly from the minimum value to the maximum value during this short run. Next, plot the accuracy versus learning rate. Note the learning rate value when the accuracy starts to increase and when the accuracy slows, becomes ragged, or

Dataset	LR policy	Iterations	Accuracy (%)
CIFAR-10	$fixed$	70,000	81.4
CIFAR-10	$triangular2$	**25,000**	81.4
CIFAR-10	$decay$	25,000	78.5
CIFAR-10	exp	70,000	79.1
CIFAR-10	exp_range	42,000	**82.2**
AlexNet	$fixed$	400,000	58.0
AlexNet	$triangular2$	400,000	**58.4**
AlexNet	exp	300,000	56.0
AlexNet	exp	460,000	56.5
AlexNet	exp_range	300,000	56.5
GoogLeNet	$fixed$	420,000	63.0
GoogLeNet	$triangular2$	420,000	**64.4**
GoogLeNet	exp	240,000	58.2
GoogLeNet	exp_range	240,000	60.2

Table 1. Comparison of accuracy results on test/validation data at the end of the training.

starts to fall. These two learning rates are good choices for bounds; that is, set $base\_lr$ to the first value and set max_lr to the latter value. Alternatively, one can use the rule of thumb that the optimum learning rate is usually within a factor of two of the largest one that converges [2] and set $base\_lr$ to $\frac{1}{3}$ or $\frac{1}{4}$ of max_lr.

Figure 3 shows an example of making this type of run with the CIFAR-10 dataset, using the architecture and hyper-parameters provided by Caffe. One can see from Figure 3 that the model starts converging right away, so it is reasonable to set $base\_lr = 0.001$. Furthermore, above a learning rate of 0.006 the accuracy rise gets rough and eventually begins to drop so it is reasonable to set $max\_lr = 0.006$.

Whenever one is starting with a new architecture or dataset, a single LR range test provides both a good LR value and a good range. Then one should compare runs with a fixed LR versus CLR with this range. Whichever wins can be used with confidence for the rest of one's experiments.

4. Experiments

The purpose of this section is to demonstrate the effectiveness of the CLR methods on some standard datasets and with a range of architectures. In the subsections below, CLR policies are used for training with the CIFAR-10, CIFAR-100, and ImageNet datasets. These three datasets and a variety of architectures demonstrate the versatility of CLR.

4.1. CIFAR-10 and CIFAR-100

4.1.1 Caffe's CIFAR-10 architecture

The CIFAR-10 architecture and hyper-parameter settings on the Caffe website are fairly standard and were used here as a baseline. As discussed in Section 3.2, an epoch is equal

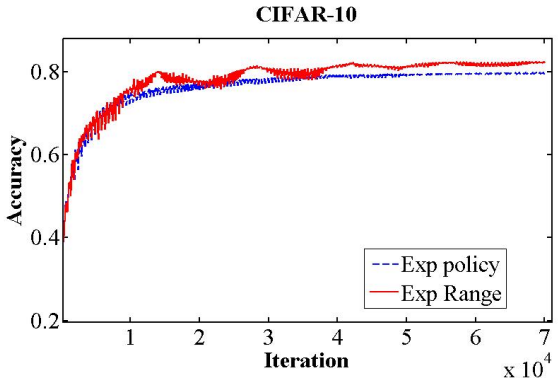

Figure 4. Classification accuracy as a function of iteration for $70,000$ iterations.

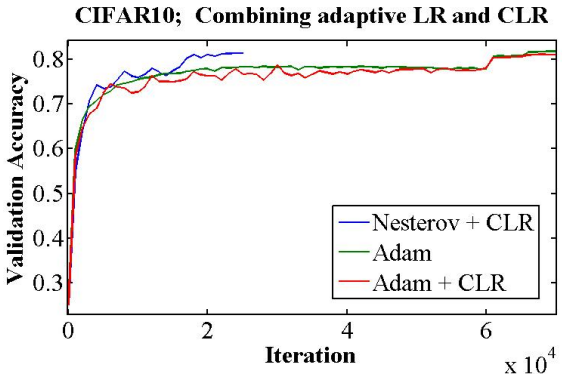

Figure 5. Classification accuracy as a function of iteration for the CIFAR-10 dataset using adaptive learning methods. See text for explanation.

to 500 iterations and a good setting for $stepsize$ is $2,000$. Section 3.3 discussed how to estimate reasonable minimum and maximum boundary values for the learning rate from Figure 3. All that is needed to optimally train the network is to set $base\_lr = 0.001$ and $max\_lr = 0.006$. This is all that is needed to optimally train the network. For the $triangular2$ policy run shown in Figure 1, the $stepsize$ and learning rate bounds are shown in Table 2.

base_lr	max_lr	stepsize	start	max_iter
0.001	0.005	2,000	0	16,000
0.0001	0.0005	1,000	16,000	22,000
0.00001	0.00005	500	22,000	25,000

Table 2. Hyper-parameter settings for CIFAR-10 example in Figure 1.

Figure 1 shows the result of running with the $triangular2$ policy with the parameter setting in Table 2. As shown in Table 1, one obtains the same test classification accuracy of 81.4% after only $25,000$ iterations with the $triangular2$ policy as obtained by running the standard hyper-parameter settings for $70,000$ iterations.

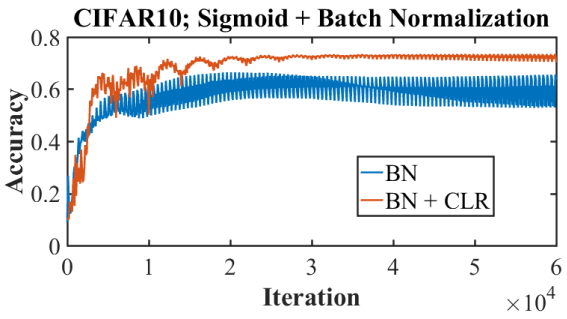

Figure 6. Batch Normalization CIFAR-10 example (provided with the Caffe download).

One might speculate that the benefits from the $triangular$ policy derive from reducing the learning rate because this is when the accuracy climbs the most. As a test, a $decay$ policy was implemented where the learning rate starts at the max_lr value and then is linearly reduced to the $base\_lr$ value for $stepsize$ number of iterations. After that, the learning rate is fixed to $base\_lr$. For the $decay$ policy, $max\_lr = 0.007$, $base\_lr = 0.001$, and $stepsize = 4000$. Table 1 shows that the final accuracy is only 78.5%, providing evidence that **both increasing and decreasing** the learning rate are essential for the benefits of the CLR method.

Figure 4 compares the exp learning rate policy in Caffe with the new exp_range policy using $gamma = 0.99994$ for both policies. The result is that when using the exp_range policy one can stop training at iteration $42,000$ with a test accuracy of 82.2% (going to iteration $70,000$ does not improve on this result). This is substantially better than the best test accuracy of 79.1% one obtains from using the exp learning rate policy.

The current Caffe download contains additional architectures and hyper-parameters for CIFAR-10 and in particular there is one with sigmoid non-linearities and batch normalization. Figure 6 compares the training accuracy using the downloaded hyper-parameters with a fixed learning rate (blue curve) to using a cyclical learning rate (red curve). As can be seen in this Figure, the final accuracy for the fixed learning rate (60.8%) is substantially lower than the cyclical learning rate final accuracy (72.2%). There is clear performance improvement when using CLR with this architecture containing sigmoids and batch normalization.

Experiments were carried out with architectures featuring both adaptive learning rate methods and CLR. Table 3 lists the final accuracy values from various adaptive learning rate methods, run with and without CLR. All of the adaptive methods in Table 3 were run by invoking the respective option in Caffe. The learning rate boundaries are given in Table 3 (just below the method's name), which were determined by using the technique described in Section 3.3. Just the lower bound was used for $base\_lr$ for the $fixed$ policy.

LR type/bounds	LR policy	Iterations	Accuracy (%)
Nesterov [19]	$fixed$	70,000	82.1
0.001 - 0.006	$triangular$	25,000	81.3
ADAM [16]	$fixed$	70,000	81.4
0.0005 - 0.002	$triangular$	25,000	79.8
	$triangular$	70,000	81.1
RMSprop [27]	$fixed$	70,000	75.2
0.0001 - 0.0003	$triangular$	25,000	72.8
	$triangular$	70,000	75.1
AdaGrad [5]	$fixed$	70,000	74.6
0.003 - 0.035	$triangular$	25,000	76.0
AdaDelta [29]	$fixed$	70,000	67.3
0.01 - 0.1	$triangular$	25,000	67.3

Table 3. Comparison of CLR with adaptive learning rate methods. The table shows accuracy results for the CIFAR-10 dataset on test data at the end of the training.

Table 3 shows that for some adaptive learning rate methods combined with CLR, the final accuracy after only 25,000 iterations is equivalent to the accuracy obtained without CLR after 70,000 iterations. For others, it was necessary (even with CLR) to run until 70,000 iterations to obtain similar results. Figure 5 shows the curves from running the Nesterov method with CLR (reached 81.3% accuracy in only 25,000 iterations) and the Adam method both with and without CLR (both needed 70,000 iterations). When using adaptive learning rate methods, the benefits from CLR are sometimes reduced, but CLR can still valuable as it sometimes provides benefit at essentially no cost.

4.1.2 ResNets, Stochastic Depth, and DenseNets

Residual networks [10, 11], and the family of variations that have subsequently emerged, achieve state-of-the-art results on a variety of tasks. Here we provide comparison experiments between the original implementations and versions with CLR for three members of this residual network family: the original ResNet [10], Stochastic Depth networks [13], and the recent DenseNets [12]. Our experiments can be readily replicated because the authors of these papers make their Torch code available[3]. Since all three implementation are available using the Torch 7 framework, the experiments in this section were performed using Torch. In addition to the experiment in the previous Section, these networks also incorporate batch normalization [15] and demonstrate the value of CLR for architectures with batch normalization.

Both CIFAR-10 and the CIFAR-100 datasets were used

[3] https://github.com/facebook/fb.resnet.torch, https://github.com/yueatsprograms/Stochastic_Depth, https://github.com/liuzhuang13/DenseNet

in these experiments. The CIFAR-100 dataset is similar to the CIFAR-10 data but it has 100 classes instead of 10 and each class has 600 labeled examples.

Architecture	CIFAR-10 (LR)	CIFAR-100 (LR)
ResNet	$92.8(0.1)$	$71.2(0.1)$
ResNet	$93.3(0.2)$	$71.6(0.2)$
ResNet	$91.8(0.3)$	$71.9(0.3)$
ResNet+CLR	$93.6(0.1-0.3)$	$72.5(0.1-0.3)$
SD	$94.6(0.1)$	$75.2(0.1)$
SD	$94.5(0.2)$	$75.2(0.2)$
SD	$94.2(0.3)$	$74.6(0.3)$
SD+CLR	$94.5(0.1-0.3)$	$75.4(0.1-0.3)$
DenseNet	$94.5(0.1)$	$75.2(0.1)$
DenseNet	$94.5(0.2)$	$75.3(0.2)$
DenseNet	$94.2(0.3)$	$74.5(0.3)$
DenseNet+CLR	$94.9(0.1-0.2)$	$75.9(0.1-0.2)$

Table 4. Comparison of CLR with ResNets [10, 11], Stochastic Depth (SD) [13], and DenseNets [12]. The table shows the average accuracy of 5 runs for the CIFAR-10 and CIFAR-100 datasets on test data at the end of the training.

The results for these two datasets on these three architectures are summarized in Table 4. The left column give the architecture and whether CLR was used in the experiments. The other two columns gives the average final accuracy from five runs and the initial learning rate or range used in parenthesis, which are reduced (for both the fixed learning rate and the range) during the training according to the same schedule used in the original implementation. For all three architectures, the original implementation uses an initial LR of 0.1 which we use as a baseline.

The accuracy results in Table 4 in the right two columns are the average final test accuracies of five runs. The Stochastic Depth implementation was slightly different than the ResNet and DenseNet implementation in that the authors split the 50,000 training images into 45,000 training images and 5,000 validation images. However, the reported results in Table 4 for the SD architecture is only test accuracies for the five runs. The learning rate range used by CLR was determined by the LR range test method and the cycle length was choosen as a tenth of the maximum number of epochs that was specified in the original implementation.

In addition to the accuracy results shown in Table 4, similar results were obtained in Caffe for DenseNets [12] on CIFAR-10 using the prototxt files provided by the authors. The average accuracy of five runs with learning rates of $0.1, 0.2, 0.3$ was $91.67\%, 92.17\%, 92.46\%$, respectively, but running with CLR within the range of 0.1 to 0.3, the average accuracy was 93.33%.

The results from all of these experiments show similar or better accuracy performance when using CLR versus using a fixed learning rate, even though the performance drops at some of the learning rate values within this range. These experiments confirm that it is beneficial to use CLR for a variety of residual architectures and for both CIFAR-10 and CIFAR-100.

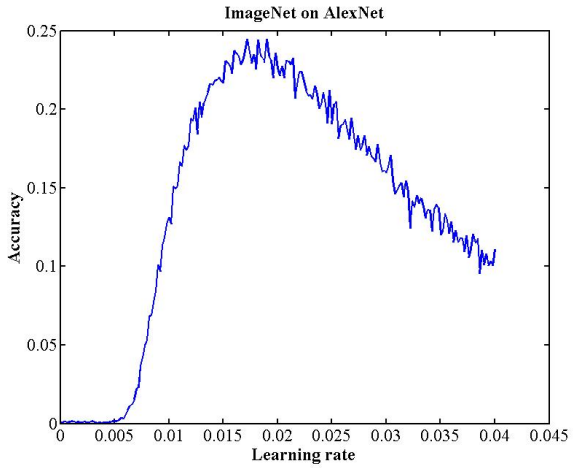

Figure 7. AlexNet LR range test; validation classification accuracy as a function of increasing learning rate.

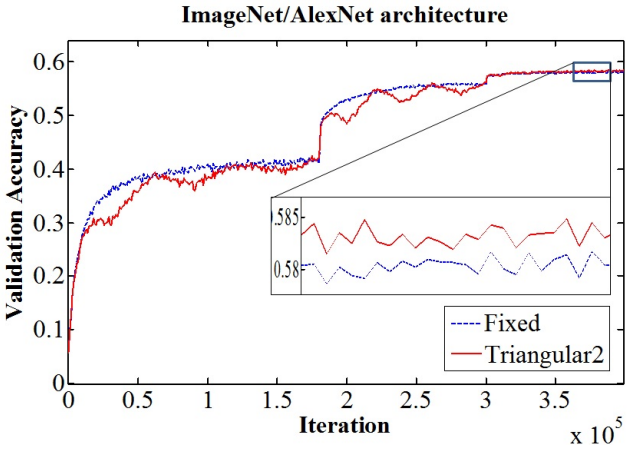

Figure 8. Validation data classification accuracy as a function of iteration for $fixed$ versus $triangular$.

4.2. ImageNet

The ImageNet dataset [21] is often used in deep learning literature as a standard for comparison. The ImageNet classification challenge provides about $1,000$ training images for each of the $1,000$ classes, giving a total of $1,281,167$ labeled training images.

4.2.1 AlexNet

The Caffe website provides the architecture and hyperparameter files for a slightly modified AlexNet [17]. These were downloaded from the website and used as a baseline. In the training results reported in this section, all weights

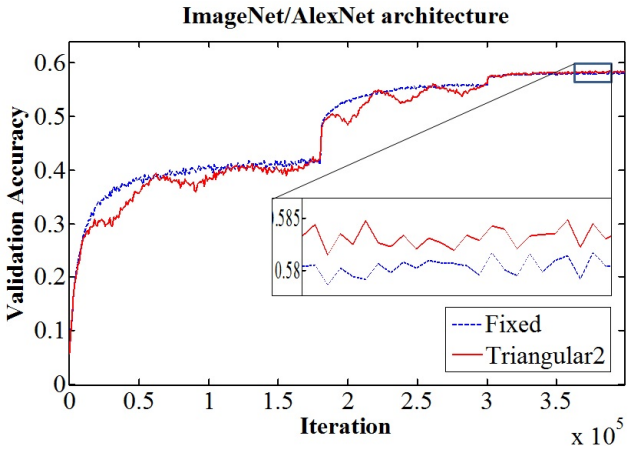

Figure 9. Validation data classification accuracy as a function of iteration for $fixed$ versus $triangular$.

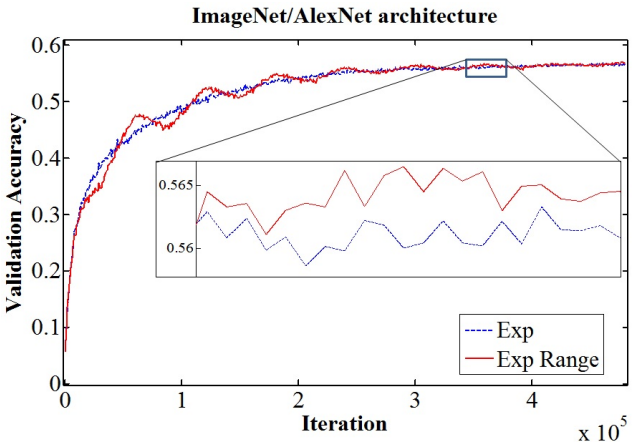

Figure 10. Validation data classification accuracy as a function of iteration for exp versus exp_range.

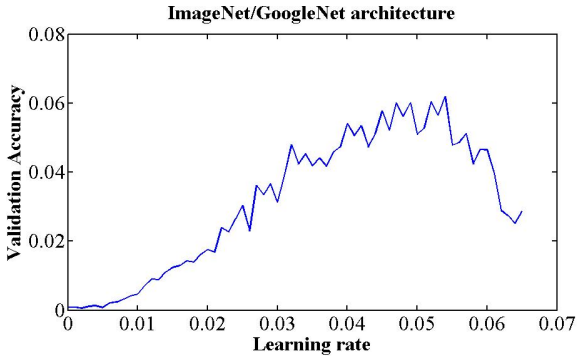

Figure 11. GoogleNet LR range test; validation classification accuracy as a function of increasing learning rate.

were initialized the same so as to avoid differences due to different random initializations.

Since the $batchsize$ in the architecture file is 256, an epoch is equal to $1,281,167/256 = 5,005$ iterations. Hence, a reasonable setting for $stepsize$ is 6 epochs or $30,000$ iterations.

Next, one can estimate reasonable minimum and maximum boundaries for the learning rate from Figure 7. It can be seen from this figure that the training doesn't start converging until at least 0.006 so setting $base\_lr = 0.006$ is reasonable. However, for a fair comparison to the baseline where $base\_lr = 0.01$, it is necessary to set the $base\_lr$ to 0.01 for the $triangular$ and $triangular2$ policies or else the majority of the apparent improvement in the accuracy will be from the smaller learning rate. As for the maximum boundary value, the training peaks and drops above a learning rate of 0.015 so $max\_lr = 0.015$ is reasonable. For comparing the exp_range policy to the exp policy, setting $base\_lr = 0.006$ and $max\_lr = 0.014$ is reasonable and in this case one expects that the average accuracy of the exp_range policy to be equal to the accuracy from the exp policy.

Figure 9 compares the results of running with the $fixed$ versus the $triangular2$ policy for the AlexNet architecture. Here, the peaks at iterations that are multiples of 60,000 should produce a classification accuracy that corresponds to the $fixed$ policy. Indeed, the accuracy peaks at the end of a cycle for the $triangular2$ policy are similar to the accuracies from the standard $fixed$ policy, which implies that the baseline learning rates are set quite well (this is also implied by Figure 7). As shown in Table 1, the final accuracies from the CLR training run are only 0.4% better than the accuracies from the $fixed$ policy.

Figure 10 compares the results of running with the exp versus the exp_range policy for the AlexNet architecture with $gamma = 0.999995$ for both policies. As expected, Figure 10 shows that the accuracies from the exp_range policy do oscillate around the exp policy accuracies. The advantage of the exp_range policy is that the accuracy of 56.5% is already obtained at iteration $300,000$ whereas the exp policy takes until iteration $460,000$ to reach 56.5%.

Finally, a comparison between the $fixed$ and exp policies in Table 1 shows the $fixed$ and $triangular2$ policies produce accuracies that are almost 2% better than their exponentially decreasing counterparts, but this difference is probably due to not having tuned $gamma$.

4.2.2 GoogLeNet/Inception Architecture

The GoogLeNet architecture was a winning entry to the ImageNet 2014 image classification competition. Szegedy *et al.* [25] describe the architecture in detail but did not provide the architecture file. The architecture file publicly available from Princeton[4] was used in the following experiments. The GoogLeNet paper does not state the learning rate values and the hyper-parameter solver file is not avail-

[4] vision.princeton.edu/pvt/GoogLeNet/

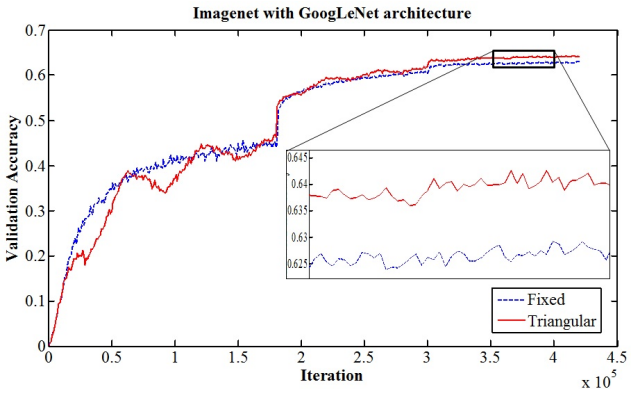

Figure 12. Validation data classification accuracy as a function of iteration for $fixed$ versus $triangular$.

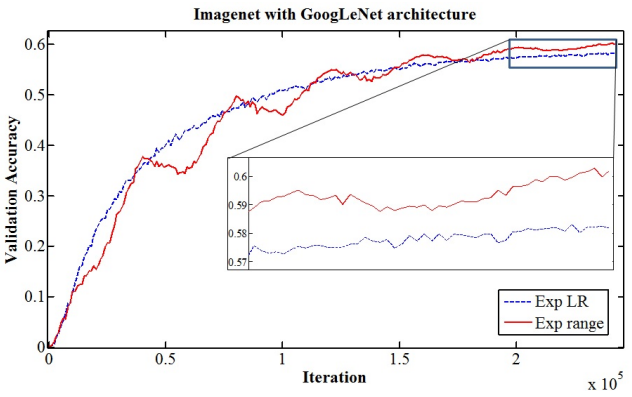

Figure 13. Validation data classification accuracy as a function of iteration for exp versus exp_range.

able for a baseline but not having these hyper-parameters is a typical situation when one is developing a new architecture or applying a network to a new dataset. This is a situation that CLR readily handles. Instead of running numerous experiments to find optimal learning rates, the $base\_lr$ was set to a best guess value of 0.01.

The first step is to estimate the $stepsize$ setting. Since the architecture uses a batchsize of 128 an epoch is equal to $1,281,167/128 = 10,009$ iterations. Hence, good settings for $stepsize$ would be $20,000$, $30,000$, or possibly $40,000$. The results in this section are based on $stepsize = 30000$.

The next step is to estimate the bounds for the learning rate, which is found with the LR range test by making a run for 4 epochs where the learning rate linearly increases from 0.001 to 0.065 (Figure 11). This figure shows that one can use bounds between 0.01 and 0.04 and still have the model reach convergence. However, learning rates above 0.025 cause the training to converge erratically. For both $triangular2$ and the exp_range policies, the $base\_lr$ was set to 0.01 and max_lr was set to 0.026. As above, the accuracy peaks for both these learning rate policies correspond to the same learning rate value as the $fixed$ and exp policies. Hence, the comparisons below will focus on the peak accuracies from the LCR methods.

Figure 12 compares the results of running with the $fixed$ versus the $triangular2$ policy for this architecture (due to time limitations, each training stage was not run until it fully plateaued). In this case, the peaks at the end of each cycle for the $triangular2$ policy produce better accuracies than the $fixed$ policy. The final accuracy shows an improvement from the network trained by the $triangular2$ policy (Table 1) to be 1.4% better than the accuracy from the $fixed$ policy. This demonstrates that the $triangular2$ policy improves on a "best guess" for a fixed learning rate.

Figure 13 compares the results of running with the exp versus the exp_range policy with $gamma = 0.99998$. Once again, the peaks at the end of each cycle for the exp_range policy produce better validation accuracies than the exp policy. The final accuracy from the exp_range policy (Table 1) is 2% better than from the exp policy.

5. Conclusions

The results presented in this paper demonstrate the benefits of the cyclic learning rate (CLR) methods. A short run of only a few epochs where the learning rate linearly increases is sufficient to estimate boundary learning rates for the CLR policies. Then a policy where the learning rate cyclically varies between these bounds is sufficient to obtain near optimal classification results, often with fewer iterations. This policy is easy to implement and unlike adaptive learning rate methods, incurs essentially no additional computational expense.

This paper shows that use of cyclic functions as a learning rate policy provides substantial improvements in performance for a range of architectures. In addition, the cyclic nature of these methods provides guidance as to times to drop the learning rate values (after 3 - 5 cycles) and when to stop the the training. All of these factors reduce the guesswork in setting the learning rates and make these methods practical tools for everyone who trains neural networks.

This work has not explored the full range of applications for cyclic learning rate methods. We plan to determine if equivalent policies work for training different architectures, such as recurrent neural networks. Furthermore, we believe that a theoretical analysis would provide an improved understanding of these methods, which might lead to improvements in the algorithms.

References

[1] K. Bache, D. DeCoste, and P. Smyth. Hot swapping for online adaptation of optimization hyperparameters. *arXiv preprint arXiv:1412.6599*, 2014. 2

[2] Y. Bengio. *Neural Networks: Tricks of the Trade*, chapter Practical recommendations for gradient-based training of

deep architectures, pages 437–478. Springer Berlin Heidelberg, 2012. 1, 2, 4

[3] T. M. Breuel. The effects of hyperparameters on sgd training of neural networks. *arXiv preprint arXiv:1508.02788*, 2015. 2

[4] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *arXiv preprint arXiv:1502.04390*, 2015. 2

[5] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011. 2, 5

[6] A. P. George and W. B. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine learning*, 65(1):167–198, 2006. 2

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014. 1

[8] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014. 1

[9] C. Gulcehre and Y. Bengio. Adasecant: Robust adaptive secant method for stochastic gradient. *arXiv preprint arXiv:1412.7419*, 2014. 2

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, 2015. 5, 6

[11] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016. 5, 6

[12] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. 5, 6

[13] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016. 5, 6

[14] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, R. Cheng-Yue, F. Mujica, A. Coates, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015. 1

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 5

[16] D. Kingma and J. Lei-Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015. 2, 5

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012. 1, 2, 6

[18] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016. 2

[19] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983. 5

[20] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1600.04747*, 2016. 2

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. 6

[22] T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. *arXiv preprint arXiv:1206.1106*, 2012. 2

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[24] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014. 1

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 1, 2, 7

[26] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014. 1

[27] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012. 2, 5

[28] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *arXiv preprint arXiv:1411.4555*, 2014. 1

[29] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 2, 5

A. Instructions for adding CLR to Caffe

Modify SGDSolver¡Dtype¿::GetLearningRate() which is in sgd_solver.cpp (near line 38):

```
} else if (lr_policy == "triangular") {
  int itr = this->iter_ - this->param_.start_lr_policy();
  if(itr > 0) {
    int cycle = itr / (2*this->param_.stepsize());
    float x = (float) (itr - (2*cycle+1)*this->param_.stepsize());
    x = x / this->param_.stepsize();
    rate = this->param_.base_lr() + (this->param_.max_lr()- this->param_.base_lr())
          * std::max(double(0),(1.0 - fabs(x)));
  } else {
    rate = this->param_.base_lr();
  }
} else if (lr_policy == "triangular2") {
  int itr = this->iter_ - this->param_.start_lr_policy();
  if(itr > 0) {
    int cycle = itr / (2*this->param_.stepsize());
    float x = (float) (itr - (2*cycle+1)*this->param_.stepsize());
    x = x / this->param_.stepsize();
    rate = this->param_.base_lr() + (this->param_.max_lr()- this->param_.base_lr())
           * std::min(double(1), std::max(double(0), (1.0 -
fabs(x))/pow(2.0,double(cycle))));
  } else {
    rate = this->param_.base_lr();
}
```

Modify message SolverParameter which is in caffe.proto (near line 100):

```
  optional float start_lr_policy = 41;
  optional float max_lr = 42; // The maximum learning rate for CLR policies
```

B. Instructions for adding CLR to Keras

Please see https://github.com/bckenstler/CLR.