# Cyclical Learning Rates for Training Neural Networks With Unbalanced Data Sets

Basel Alyafi, Fakrul Islam Tushar, Zafar Toshpulatov
University of Cassino and Southern Latium

***Abstract*—As the learning rate is one of the most important hyper-parameters to tune for training convolutional neural networks. In this paper, a powerful technique to select a range of learning rates for a neural network that named cyclical learning rate was implemented with two different skewness degrees. It is an approach to adjust where the value is cycled between a lower bound and upper bound. CLR policies are computationally simpler and can avoid the computational expense of fine tuning with fixed learning rate. It is clearly shown that changing the learning rate during the training phase provides by far better results than fixed values with similar or even smaller number of epochs.**

## I. Introduction

The learning rate is a significant hyper-parameter that controls the adjustment of network weights with respect to loss function gradients. This parameter scales the magnitude of weight updates in order to minimize the network's cost function. If the learning rate is too small, the training process will progress very slowly. However, if it is set to a too high value, it may cause undesirable divergent behavior in the loss function.

In order to eliminate the need to experimentally find reasonable values, one of the most powerful techniques was proposed by [1], named cyclical learning rates. A cyclical learning rate is a method to adjust the learning rate by cycling it using a proper policy between reasonable pre-tuned boundary values, namely: a lower bound $base\_lr$ and an upper bound $max\_lr$, see Fig. 1. Training with cyclical learning rates, instead of fixed values achieves, relatively-improved classification accuracy without the need to tune and often in fewer iterations. The purpose of our work is to demonstrate the effectiveness of the Cyclical Learning Rates (CLR) method on a data set of 300k breast-micro-calcification gray-scale images. Different skewness degrees were used with more reliable metrics than accuracy, namely: area under the ROC curve, area under the precision-recall curve and F1 score. Three different CLR policies (triangular, triangular2 and exp_range) were used for training with a specific Convolutional Neural Network architecture. It is worth noting that adaptive learning rate methods requires noticeable increase in computational complexity in comparison with CLR, which is inherently simple.

## II. Cyclical Learning Rates

Instead of using a fixed value for learning rate during the whole training process, it was shown in [1] that oscillating the value inside a band helps to overpass saddle points plateaus more quickly. Updating the learning rate is done by using equations (1,2,3)[1].

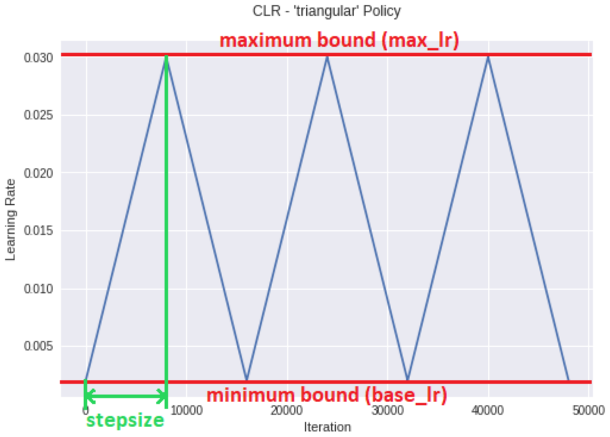


Fig. 1: Triangular learning rate policy. The blue lines represent learning rate values changing between red bounds. The input parameter step_size is the number of iterations in half a cycle.

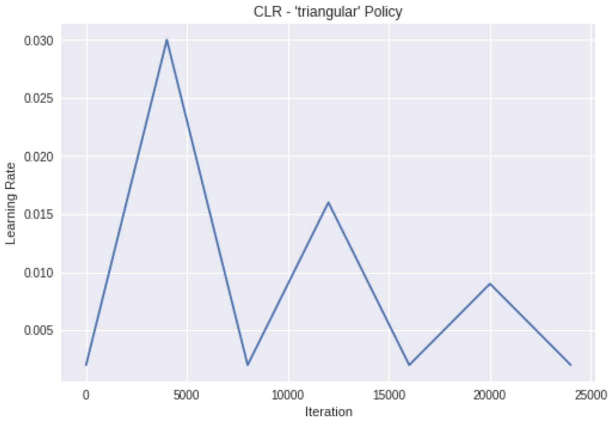


Fig. 2: Triangular 2 learning rate policy.

$$lr = base\_lr + (max\_lr - base\_lr) * max(0, 1 - x) \quad (1)$$

$$x = abs(iterations/step\_size - 2 * cycle + 1) \quad (2)$$

$$cycle = floor(1 + iterations/(2 * step\_size)) \quad (3)$$

**Acknowledgement**: This work experiments were conducted using GOOGLE COLABORATORY web service.

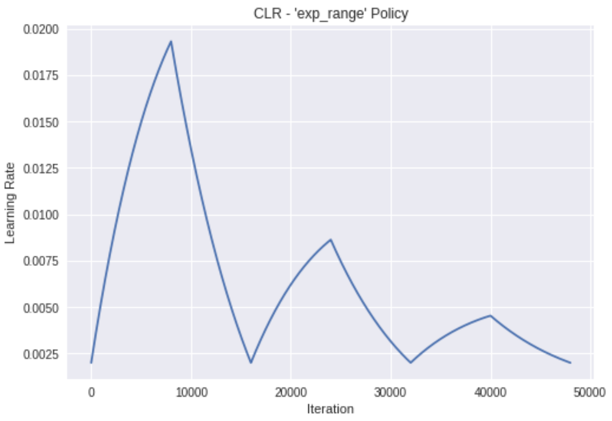


Fig. 3: Exponential-range learning rate policy.

To dig a bit deeper, in Fig 1, $base\_lr$ and $max\_lr$ for the algorithm are initially tuned. Starting from iteration one, the learning rate starts with the base_lr and moves towards the max_lr in a defined manner which can be linear, exponential range or decaying triangular. Each step is defined to involve a number of iterations (or epochs) in a way that assures ending sharply at the end of a falling edge, i.e., steps number should be an even integer, see eq(7). The size of the data set considered for training should be dividable by the batch size to ensure safe termination. The method of cyclical learning rates consists of three different policies:

### A. *Triangular Policy*

This varies the learning rate linearly between the minimum ($base\_lr$) and the maximum ($max\_lr$) as shown in Figure 1.

### B. *Triangular 2 Policy*

This policy is same as the triangular policy except that the maximum bound is cut in half at the end of each cycle as shown in Figure 2. This means the learning rate difference decreases after each cycle and learning rate is defined as

$$\begin{aligned} lr = base\_lr + (max\_lr - base\_lr)* \\ max(0,(1-x))/float(2^{cycle-1}) \end{aligned} \tag{4}$$

### C. *Exponential range Policy*

This scales the cycle amplitude by a factor gamma (iterations), while keeping the $base\_lr$ constant as shown in Figure 3. The learning rate varies between the minimum and maximum boundaries and each boundary value declines by an exponential factor of gamma iteration and learning rate is calculated as

$$\begin{aligned} lr = base\_lr + (max\_lr - base\_lr) * max(0,(1-x)) \\ *gamma^{iterations} \end{aligned} \tag{5}$$

## III. Methodology

### A. *CNN Architecture*

In this study we used Convolutional Neural Network (CNN) to test CLR approach. This architecture has 4 convolution layers, 2 max-pooling layers, 2 dropout layers and 3 fully connected layers with a final 2-way softmax activation, resulting in more than 225k parameters to be trained, Fig.5. Here we briefly discuss each layer. Fig. 4 shows the CNN architecture. Each conv. layer has 32 neurons. Each neuron in the conv layers compute product between their weights and input volume that is connected to its local region and produce activation map. The weights and bias of the neuron are adjusted by training and CNN learn to respond to the activation. For faster training with gradient descent, Rectified Linear Unit(ReLu) is used. The max pooling layers are used to reduce the spatial dimension of the input. We adopt pooling layers with filter of size 2 x 2 with a stride 2. In order to reduce test error and avoid over-fitting, dropout layer was implemented. The layers fc1, fc2 and prediction shown in Fig are the fully-connected layers. Neuron in the fully connected layers have full connections to all neurons in the previous layer. The last layer of prediction consists of 2 neurons, and each score represents the confidence degree that the input belongs to positive or negative class.

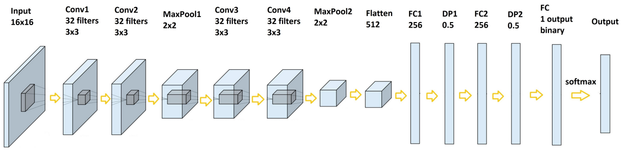
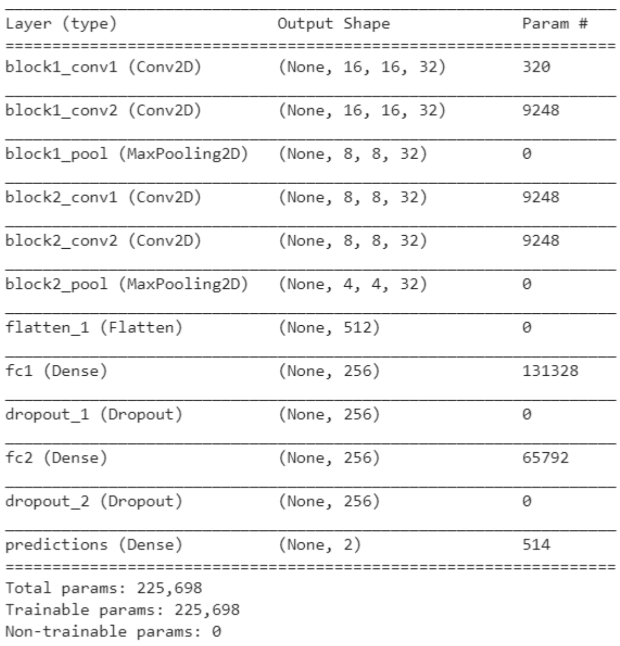


Fig. 4: CNN architecture



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 16, 16, 32)        320
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 16, 16, 32)        9248
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 8, 8, 32)          0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 8, 8, 32)          9248
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 8, 8, 32)          9248
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 4, 4, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
fc1 (Dense)                  (None, 256)               131328
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
fc2 (Dense)                  (None, 256)               65792
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
predictions (Dense)          (None, 2)                 514
=================================================================
Total params: 225,698
Trainable params: 225,698
Non-trainable params: 0
```

Fig. 5: Model summary

### B. *Cycle Length Estimation*

In Cyclical learning rate, cycle length or step_size (half of the cycle length) is an important parameter that affects results significantly. Authors in [1] discuss the way of selecting the cycle length from the batch size and number of samples in the data set, eq (6). Step_size can be calculated using the number of iteration per epoch.

$$iter\_per\_epoch = examples\_num/batchsize \quad (6)$$

$$steps\_num = num\_epochs/epochs\_per\_step \quad (7)$$

Authors suggested from their different experiments to set the step_size 2 to 10 times of the number of iteration per epoch. Also, its better to end the training at the end of a cycle. In our experiments we used two different conditions. In the first case 10 epochs were used with step_size= 5*number of iteration per epoch, means half length of the circle is the 5 epoch and training ended at the end of the cycle. In the second case 12 epochs were used with $step\_size = 2*iterations\_per\_epoch$ , means half length of the cycle is the 2 epoch and training was done by 3 complete cycles.

### C. Minimum and Maximum LR Boundary Estimation

Another magnificent step in Cyclic learning rate is to estimate the base learning rate and maximum learning rate. Authors in [1] proposed a simple approach of using LR range test by observing learning rate for a small number of epochs and observing the accuracy (with a balanced under-sampled data set). Triangular policy can be used for LR range test. The testing was performed setting the lower bound to a small value (0.002), the max_lr to 0.03 , and set the step_size to the total number of iterations in the whole experiment ($epochs * iterations\_per\_epoch$) resulting in a linearly increasing learning rate, see Fig. 6. After that, accuracy vs learning rate was plotted. Base learning should be the point from where accuracy starts increasing and max learning rate to be the learning rate where accuracy starts to slow down, oscillate, or fall [1]. Fig. 7 was analyzed, base_lr selected to be 0.002 and max_lr to be 0.03. As the data set used in our experiments was highly unbalanced, accuracy was not a good performance measure. In our LR test, balanced subset of the entire data set was used consisting 6000 samples 3000 for each class.

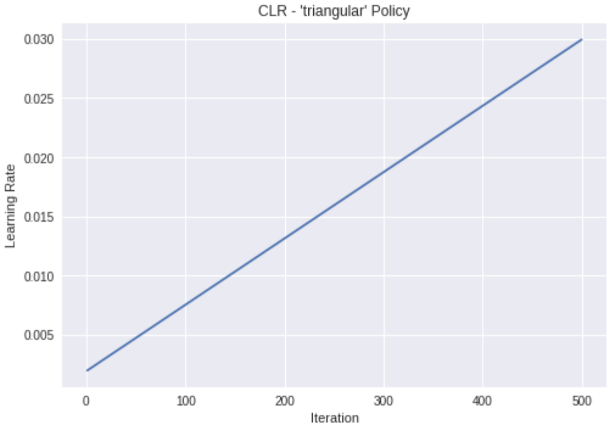


Fig. 6: Initialization, iteration vs learning rate

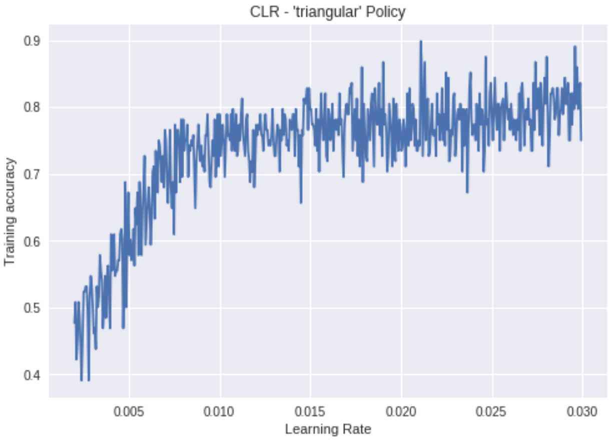


Fig. 7: Initialization, learning rate vs accuracy

## IV. ENVIRONMENT AND EXPERIMENTS

### A. Environment

All experiments were carried out using GOOGLE COLABORATORY web service. COLABORATORY is a Google research project created to help disseminate machine learning education and research. It is a helpful and free cloud service which can be used to develop deep learning applications using Python. It is a Jupyter notebook that requires no installation to use and runs entirely on the cloud. The implementation of neural network was carried out using the sequential model of Keras [2]. All measurements were made using SKLEARN [3], while ROC and Precision-Recall plots were made using SCIKIT-PLOT [4]

TABLE I: Environment Properties

| PROPERTY | VALUE |
|---|---|
| CPU | Intel Xeon CPU @ 2.20GHz, 1 core |
| RAM | 12GB |
| GPU | Nvidia Tesla K80, 24GB |

### B. Experiments

Two values of skewness were used to reflect realistic results. With the first case, skewness = 1/17, i.e., for each positive example there exist 17 different negative examples. To visualize the performance of the classifier, ROC and Precision-Recall curves were used along with the area under them. Accuracy could have been a misleading choice if it had been used with imbalanced data sets. In the second case, skewness was set to 1/36. For hyper-parameters values, see TABLE II. For every case, three runs were averaged to check the stability of results with different random initializations.

### C. Case 1, Skewness = 1/17

Table III show the average results for triangular, triangular2, exp_range and fixed. From the table, it is clear that exp_range is relatively better among others with high F1 score and AUC with small deviations. All policies outperformed the fixed learning rate with lr = 0.002 (base_lr).

| HYP-PARAM | VALUE |
|---|---|
| epochs | 12 |
| basel_lr | 0.002 |
| max_lr | 0.03 |
| batch_size | 32 |
| Train set size | 64k (case 1), 128k( case 2) |
| epochs per step | 2 |

TABLE II: Hyper-parameters values

TABLE III: Case-1 Averages

| Policy | AUC | PRECISION | RECALL | F1Score |
|---|---|---|---|---|
| triang1 | 0.9940 | 0.4750 | 0.9590 | 0.6350 |
| triang2 | 0.9950 | 0.4790 | 0.9600 | 0.6300 |
| exp | 0.9950 | 0.5720 | 0.9420 | 0.7000 |
| fixed | 0.6867 | 0.1327 | 0.4940 | 0.0914 |

### *D. Case 2, Skewness = 1/36*

TABLE V and TABLE VI show results for the second case. Here, the fixed policy gave the worst results among others, it gave around AUC = 0.38 with very low precision recall with learning rate = 0.002. While triangular1 and triangular2 are best performing. Fig 8 and Fig. 9 show ROC and precision-recall curves, respectively.

## V. Conclusion

The experimental results presented in this paper demonstrate the benefits of using cyclical learning rate (CLR) method for unbalanced data sets. These cyclical variations for the learning rate between the base_lr and max_lr can provide optimal classification results, also in case of highly-skewed unbalanced data sets. Simple LR range test can provide the estimation of the learning rate boundaries avoiding the computational complexity and expense of tuning the classifier model with fixed learning rate. We tried different modes of triangular learning rate with a simple CNN with a few epochs to train and test on unbalanced data set. In comparison with fixed learning rate, all the cases of CLR provided better metrics (AUC, and F1 score). In future Clr could be applied on deeper CNN with different architectures to check its performance.

TABLE IV: Case-1 Standard deviations

| Policy | AUC | PRECISION | RECALL | F1Score |
|---|---|---|---|---|
| triang1 | 0.0004 | 0.0482 | 0.0009 | 0.0423 |
| triang2 | 0.0005 | 0.0450 | 0.0017 | 0.0401 |
| exp | 0.0005 | 0.0700 | 0.0253 | 0.0531 |
| fixed | 0.1222 | 0.1880 | 0.3690 | 0.0798 |

TABLE V: Case-2 Averages

| Policy | AUC | PRECISION | RECALL | F1Score |
|---|---|---|---|---|
| triang1 | 0.9969 | 0.9232 | 0.9527 | 0.9377 |
| triang2 | 0.9970 | 0.9155 | 0.9568 | 0.9356 |
| exp | 0.9971 | 0.8757 | 0.9547 | 0.9135 |

TABLE VI: Case-2 Standard deviations

| Policy | AUC | PRECISION | RECALL | F1Score |
|---|---|---|---|---|
| triang1 | 0.0005 | 0.0068 | 0.0009 | 0.0039 |
| triang2 | 0.0004 | 0.0142 | 0.0043 | 0.0074 |
| exp | 0.0007 | 0.0122 | 0.0070 | 0.0034 |

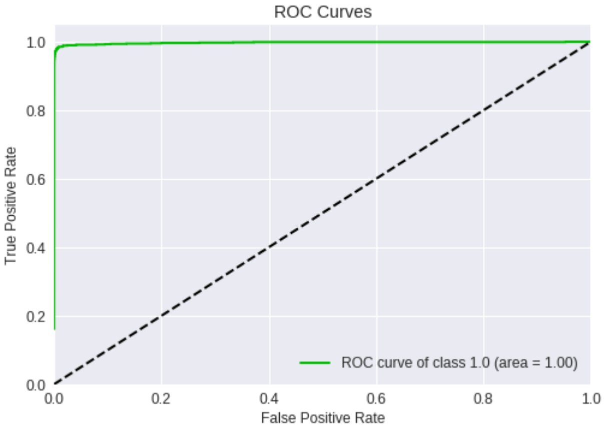


Fig. 8: ROC curve for triangular 1, skew = 1/36

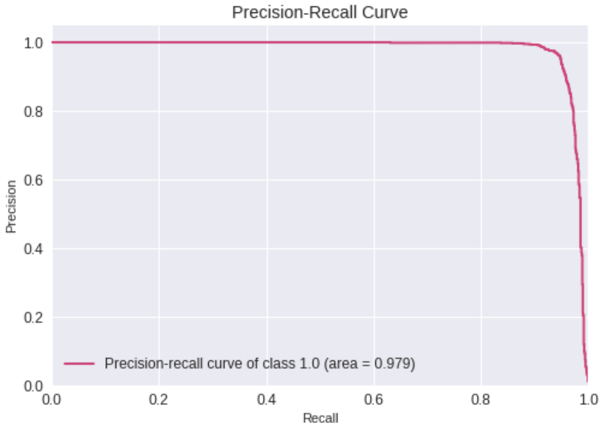


Fig. 9: PR curve for triangular 1, skew = 1/36

## References

[1] Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. *CoRR*, U.S. Naval Research Laboratory, http://arxiv.org/abs/1506.01186, Apr. 2017.

[2] Chollet, François and others. Keras. https://keras.io, 2015.

[3] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] Reiichiro Nakano. (2017, February 19). reiinakano/scikit-plot: v0.2.1 (Version v0.2.1). Zenodo. http://doi.org/10.5281/zenodo.293191.