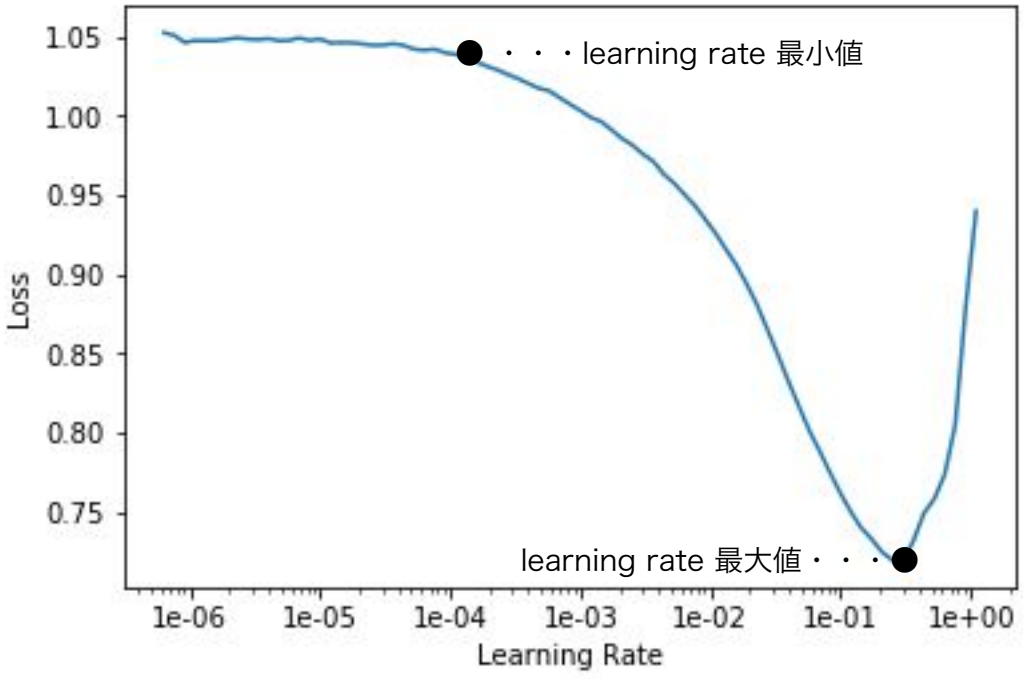
Cyclical Learning Rates for Training Neural Networks
意訳まとめ

1. Abstract

学習率はディープニューラルネットワークを学習される上で非常に重要なハイパーパラメータの一つです。今回はCyclical Learning Ratesという新しいメソッドを紹介します。これを使用することで、学習率の最適な値と変更スケジュールを実験的に変えながら検証するという作業を行わなくて良くなります。

Cyclical Learning Rates(CLR)では、学習率を短調に減少させるのではなく、境界値の間で周期的に変化させます。CRLを用いることで、学習率を短調に減少させる方法と比較し、多くの場合少ないイテレーション数で分類精度を向上させることができました。

学習率の変更範囲については、数エポックにわたりネットワークの学習率



を直線的に増加させることで得ることができます。

図1 学習率の上限値及び下限値の求め方

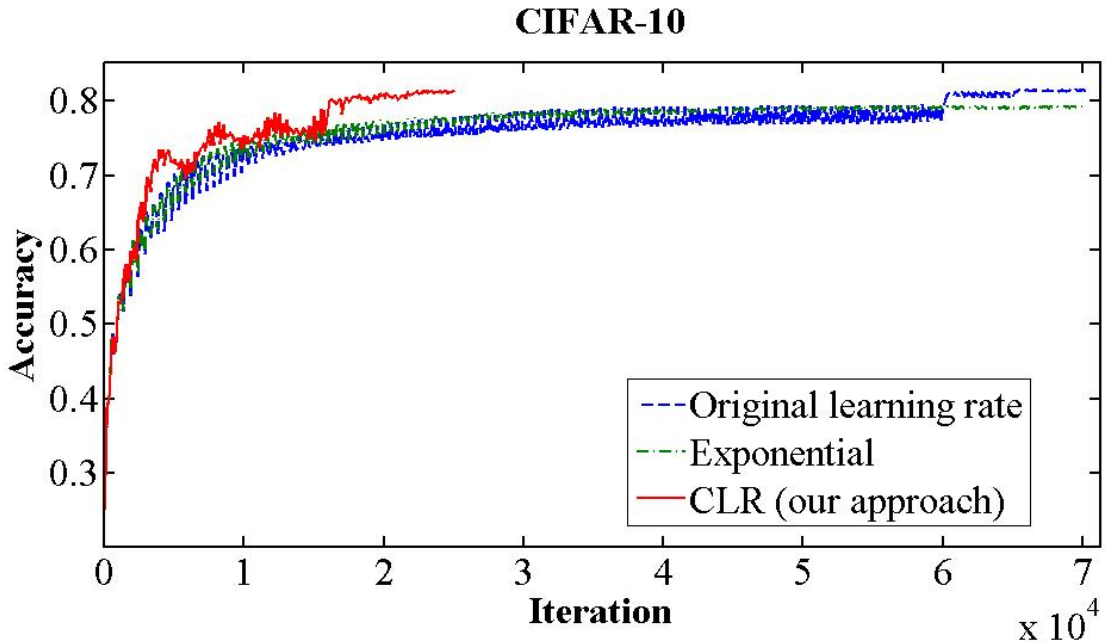
2. Introduction

ディープニューラルネットワークは通常、確率論的最急降下法により更新され、パラメータは以下の式により更新します。

$$\theta^t = \theta^{t-1} - \varepsilon_t \frac{\partial L}{\partial \theta}$$

式において、Lは損失関数でありε_tは学習率です。学習率が小さい場合、学習アルゴリズムが収束するまでに時間がかかり、また大きすぎる場合学習アルゴリズムが発散してしまうことが知られています。そのため、様々な学習率と学習スケジュールを試す必要があります。

これまでは、学習率を単一の値に固定することでモデルの学習速度を一定に保つことが一般的だった。しかし、今回の実験で学習率を固定するのではなく、設定した帯域内で周期的に変化させることでモデルを効率よく学習させることができるとわかりました。

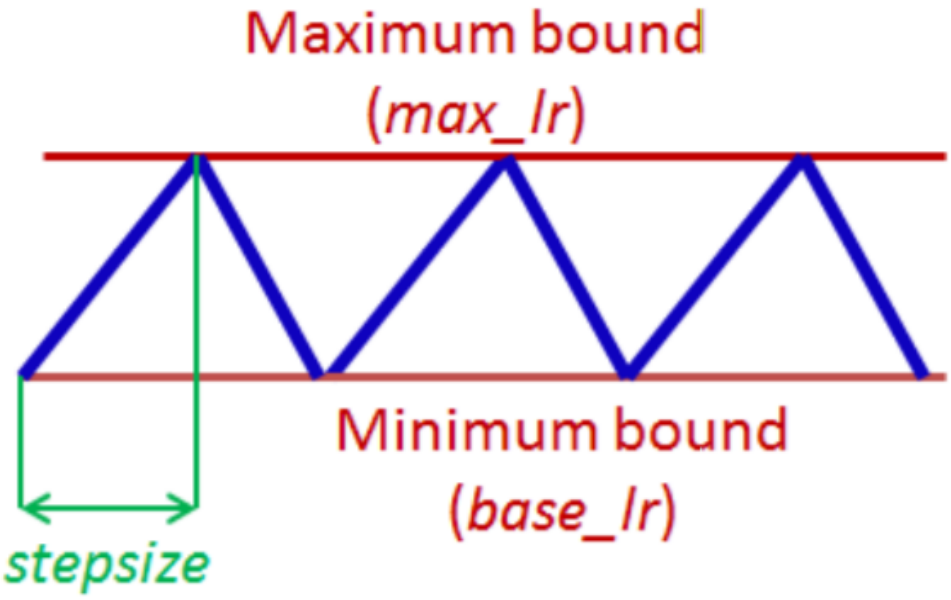


CLRの有用性について以下の図に示します。

図2 各学習率におけるCIFAR-10でのトレーニング中分類精度

ベースライン(青の線)はイテレーション数70000で精度81.4 %に到達しました。これに比べ、トレーニング中にCLR法を使用した場合(赤の線)イテレーション数25000で最大精度に達しました。

3 Optimal Learning Rates
3.1 Cyclical Learning Rates
　CLRは、ディープラーニングの学習フェーズにおいて学習率を上げた場合、短期的には悪影響が出るが、長期的には有益な効果を得ることができる可能性があるという先行研究に基づいた手法である。
(実際、CIFAR-10において、Accuracyが他に比べ大きく乱高下しているが最も早く最高精度に到達している。)
　CLRは、最小境界と最大境界を設定し境界間で学習率を周期的に変化させる手法です。
　学習率の変化のさせ方について、三角形ウィンドウ(線形)、ウェルチウィンドウ(放物線)、ハイウィンドウ(正弦波)など、多数の関数形式を使用した実験では全て同等の結果が得られました。そのため、最も単純な関数である三角ウィンドウ(線形増加と線形現象を繰り返す形)の関数形式を用いて実験を行ないました。
　学習率の変化を示す図を下に示します(赤い線が境界線、青い線が境界間で



変化する学習率)。

図3 三角学習率

　Dauphinらは、損失を最小限に抑えることは鞍点が存在するため難しいと主張しています。鞍点では勾配が小さくなり、学習の進みが遅くなります。

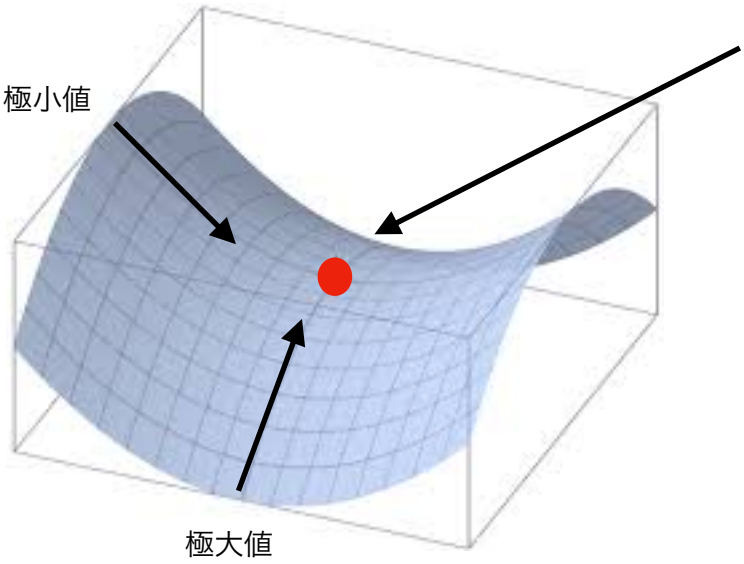
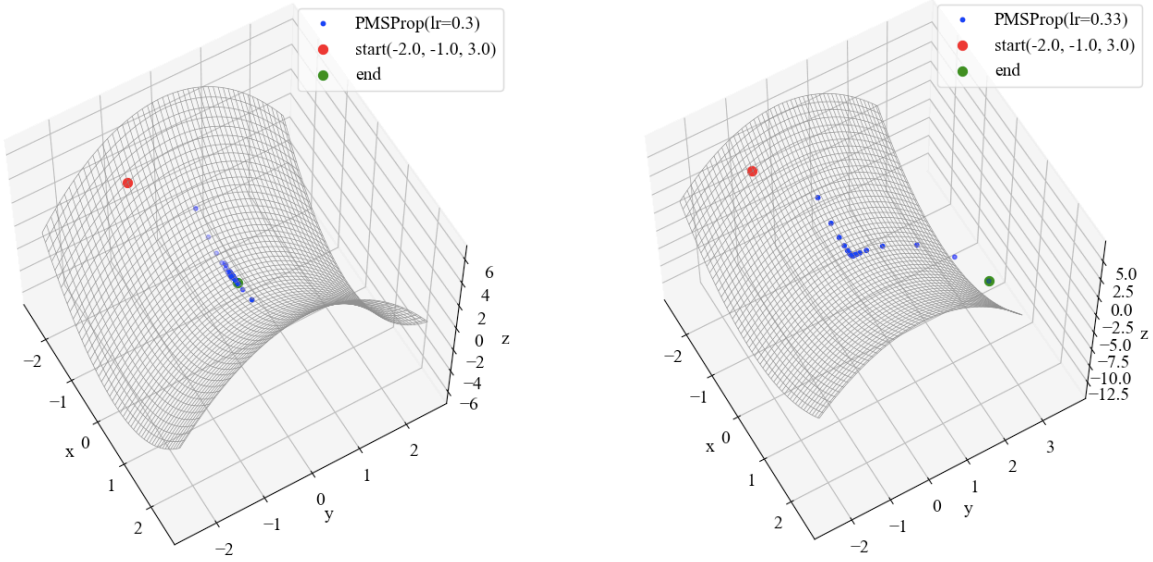


図4 鞍点

しかし、学習率を上げて学習速度を上げることで鞍点から早く抜け出すこ



とができるようになります。

図5 PMSPropにおけるの鞍点での重み更新と学習率の関係

CLRでは境界内で学習率を上下に変更するため、鞍点から早く抜け出すことができ、かつあらかじめ上限を決めているため学習率が大きくなりすぎた場合に起こる損失関数の発散を防ぐことができます。

図1の赤い曲線はCRLにより学習率を変更しながらトレーニングを行った結果を示しますが、境界の最大値を0.006、最小値を0.001に設定しています。また、図3に示すstep sizeは2000に設定しています。

以下は学習率を変更するコードです。

cycle = math.floor(1 + *epoch_counter* / (2 * *step_size*))
x = abs(*epoch_counter* / *step_size* - 2 * cycle + 1)
lr = base_*lr* + (*max_lr* - base_*lr*) * max(0, (1 - x))

base_lr = 最小学習率境界

max_lr = 最大学習率境界

epoch_counter = エポック数

step_size = 図*3*における*step_size*

また、三角形を描くように学習率を変更する手法に加え、今回の実験では以下2つの手法についても検証ていします。

triangle 2： 通常の三角学習率変更手法に加え各サイクルの終わりに学習率の差を半分に減らします。

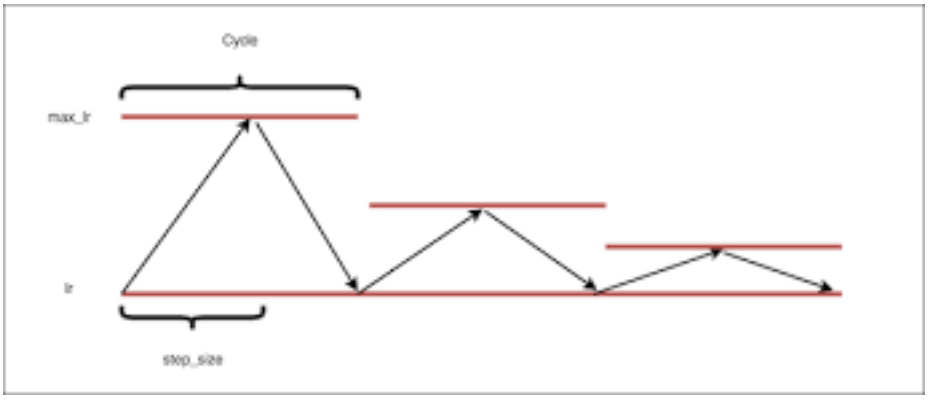


図6 triangle2

exp_range：通常の三角学習率変更手法加え、学習率の差を $\gamma^{iteration}$で減らします

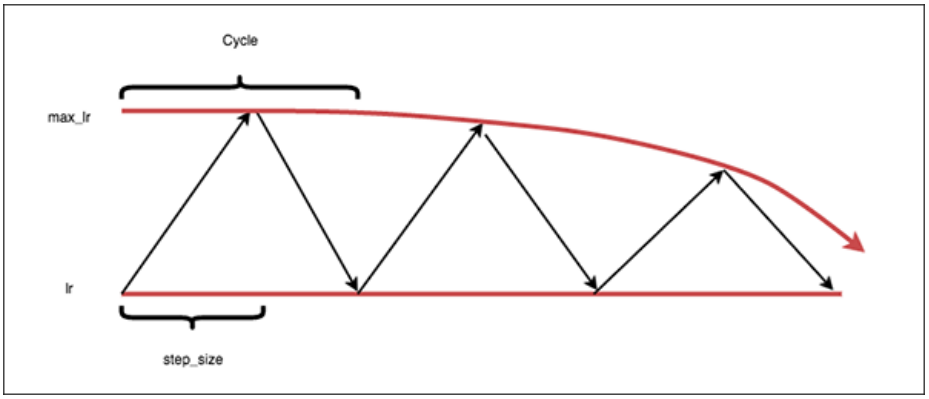


図7 exp_range

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

サイクル長さとパラメータであるステップサイズは1エポックあたりのイテレーション数から簡単に計算することができます。1エポックはトレーニング画像の数をバッチサイズで割ることにより計算できます。例えば、CIFAR-10には50000枚のトレーニング画像がありますが、バッチサイズを100とすると、1 epoch = 50000 / 100 = 500 iterations となります。

実験では、1エポックにおけるイテレーション数の2~10倍に等しいステップサイズを設定するのが適切であるという結果が出ています。

また、Cyclical Learning Ratesではサイクルには一定のリズム(step sizeの2倍ごとに上がり下がりを行う)が存在するため、学習率を下げるタイミングとトレーニングを終了するタイミングを決めるのが簡単になります。

実験からは、トレーニング開始から終了まで3サイクル回るようにした場合に精度が向上し、4サイクル以上回るようにした場合さらに良い結果が得られました。また、筆者らは学習率が最小(学習率=境界の最小値)となったときにトレーニングが終了するようにitereation数を決めて学習させることを勧めます。

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

境界の最小値、最大値を決める方法はとても簡単です。学習率を低い値から高い値に直線的に増加させながら数エポックのトレーニングを行います。

training learning rateの発想はこの「直線的に学習率を増加させる」時にも役に立ちます。

例えば以下のような方法があります。

1. base_lrを最小値に、max_lrを最大値に設定する。
2. ステップサイズと反復回数を同じ回数に設定する。(このようにすることで、学習が始まってから終わるまでの学習率の増加率を一定にすることができます。)

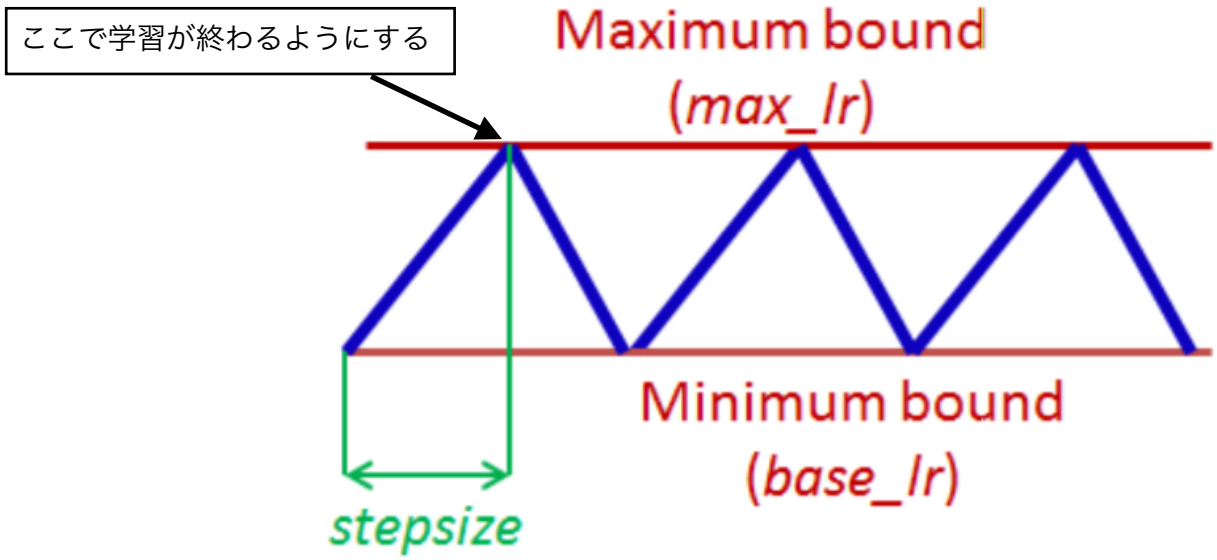


図8 方法2のイメージ

3. 学習が終了したら、各学習率における精度をプロットしていく。
4. 精度が上がり始めた学習率をbase_lr、精度が上がらなくなったり上下に不規則に推移するようになり始めた学習率をmax_lrに設定する。

図9はCaffeが提供するアーキテクチャとハイパーパラメータをとCAFAR-10データセットを使用し、上記の方法で学習率と精度をプロットした図を示します。

図9から、精度がすぐに収束し始めていることがわかります。よって、base_lrを0.001に設定するのが妥当であると言えます。また、学習率が0.006あたりを超えると精度の上昇が鈍くなってい区ため、max_lrを0.006に設定するのが妥当であると言えます。

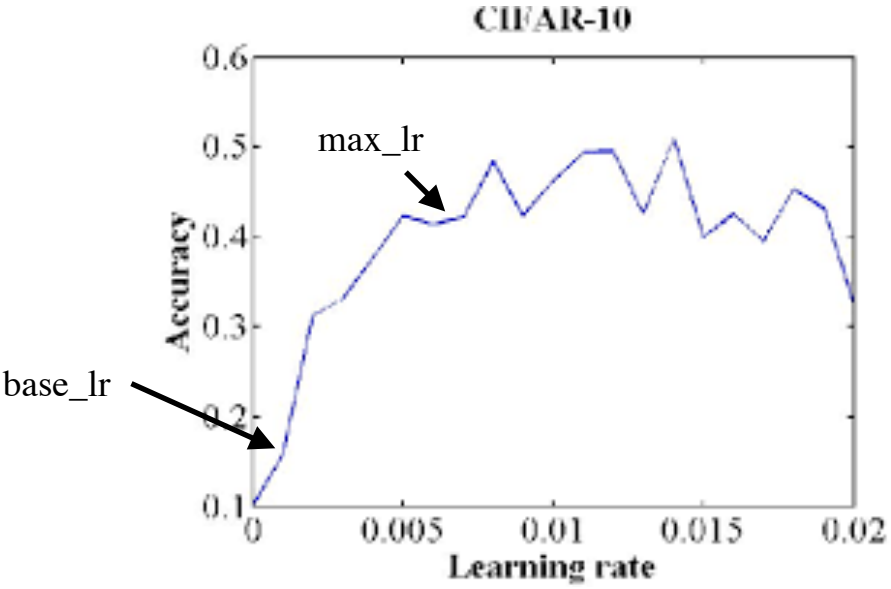


図9 エポック各学習率に対する精度(エポック数8)

4. Experiments

4.1.1 Caffe’s CIFAR-10 architecture

今回はCaffeのwebサイト載っているCIFAR-10用の分類モデルとハイパーパラメータをベースラインとして使用しました。1エポック数500 iterationとし、ステップサイズは2000としました。(セクション3.2における例と同様の値)。また、base_lrは0.001に、max_lrは0.006に設定しました。

traiangle2におけるステップサイズと学習率の範囲について表2に示します。

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

表1 使用する分類モデルにおけるハイパーパラメータの設定

表1は図1のパラメータ設定を使用してtraiangular2 手法を実行し得た間隔ごとのステップサイズとbase_lr、max_lrの値とステップサイズを示します。表2に示すように、標準のパラメータ設定をした場合70000 iteration　で81.4 %の分類精度が得られるのに対して、traiangle2 手法を使用した場合25000回のiterationで同程度の精度を得ることができました。

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

表2 LR設定方法別iterationと精度の関係

traiangle2 手法の比較対象として、学習率がmax_lrで始まり、base_lrに線形に減少するようlrを設定する手法での検証も行いました(表2のdecayが検証結果、lrはbase_lrまで減少した後固定される)。decayではmax_lrを0.007、base_lrを0.001、ステップサイズを4000に設定しました。

表2より、decayによる学習を行った場合、最終的な精度が78.5 %であることがわかります。このことから、学習率の増減両方がCLRメソッドでは不可欠であると言えます。

図4はCaffeのexp学習率変更法(exp policy、指数関数的に学習率を増加??) と、exp_range法を比較しています。gammaは0.99994に設定しました。

exp_range 方を使用した場合、iteration数42000で精度が82.2 %に到達しました(その後70000回学習させた場合、精度の向上は見られなかった)。

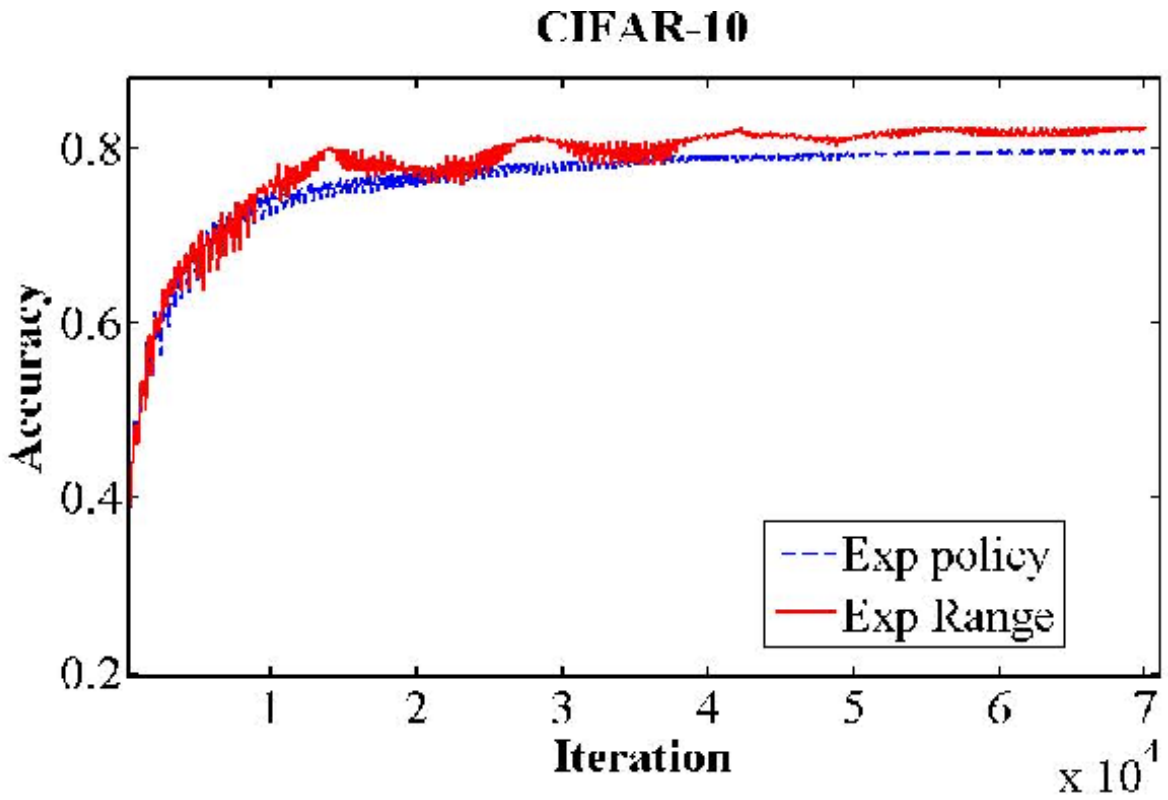


図10 学習率変更手法におけるiteration数とaccuracyの関係

現在のCaffeライブラリにはCIFAR-10用のアーキテクチャとハイパーパラメータがあらかじめ含まれています(バッチ正規化等が行われている)。図10にCaffeライブラリにおいてあらかじめ設定されている学習率を固定して学習させた場合の精度(青い線)と、Cyclical Learning Ratesを用いて学習させた場合の精度(赤い線)の比較結果を示します。学習率を固定させた場合最終的な予測精度は60.8 %だったのに対し、Cyclical Learning Ratesを使用した場合72.2 %と大幅に上がっています。今回の実験は適応学習率法(最適化アルゴリズムを用いた方法)とCLRを使用して行われました。表3に最適化アルゴリズムとCLRを合わせて使用した場合の精度について示します(アルゴリズム名の下の数値はbase_lrとmasx_lrの数値)。

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

表3 各最適化アルゴリズムとCLRを併用した場合の精度の比較

各アルゴリズムはCaffe内のメソッドとしてあらかじめ用意されているため、それを使用しました。表3より、CLRと組み合わせたアルゴリズムにおいて、AdaDeltaやAdaGradではイタレーション数25000でアルゴリズムのみ使用しイタレーション数70000で学習させた場合と最終精度が同等、もしくはそれ以上であることがわかります。また、アルゴリズムによっては同等の精度を得るためにイタレーション数70000学習させる必要があるものもありました(ADAMやRMSprop)。

図11はCLRを使用したNesterovとCLRを使用した場合、利用しなかった場合それぞれのAdamを用いて学習を行なった結果を示します。

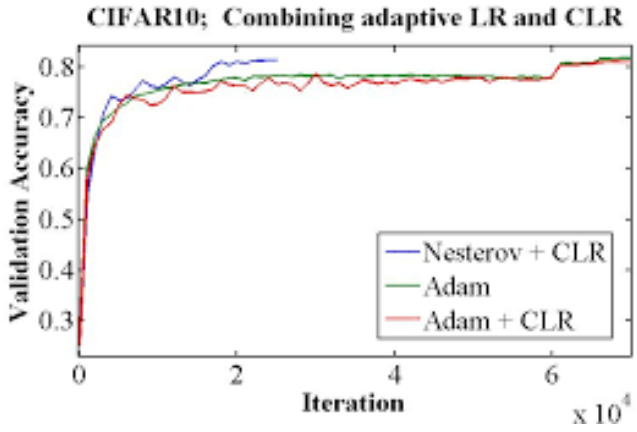


図11 アルゴリズム別CIFAR-10における精度

使用するアルゴリズムによっては優位性を示すことができませんでしたが、Nesterov等のアルゴリズムにおいてはメリットを見出すことができたため、使用する価値はあると考えます。

4.1.2 ResNets, Stochastic Depth, and DenseNets

このセクションでは、上記3つのDeep Learningアーキテクチャに CLRを使用した結果について取り扱っていきます。実験ではpytorchを使用しました。また、この実験ではCIFAR-10とCIFAR-100の両方のデータセットを使用しました。CIFAR-100においてはクラス数が100あり、各クラスには600枚のラベル付き画像が含まれています。各アーキテクチャと精度についての結果を表4に示します。

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	$94.9(0.1-0.2)$	$75.9(0.1-0.2)$

表4 アーキテクチャ別各データセットにおける精度

左列はアーキテクチャとCLRが実験で使用されたかどうかを示しています。また、他の列は最終的な精度(5回同様の実験を行い、その平均をとっている)とカッコ内は学習率を示しています。学習率については学習とともに徐々に減少するようスケジューリングされています(CLR以外)。CLRで使用される学習率の範囲はLR範囲テストによって決定され、step sizeは最大エポック数の1/10に設定しました。

全ての実験結果で、CLRを使用した場合と固定学習率を使用した場合を比較すると、 CLRを使用した場合は1部パフォーマンスが低下することがあっても、最終的な精度は同等かそれ以上の精度を出すことができることがわかる。

5. Conclusions

この論文では、cyclic learning rateの有用性について示しています。境界学習率(base_lr, max_lr)は線形に学習率を増加させ、数エポックで学習を行い各学習率での精度を比較することで算出することができます。また、CLRを使用することで、多くの場合少ないイタレーション数で最高精度を出すことができ学習時間を短縮することができます。CLRは実装が容易であるため、最適化アルゴリズムを使用する学習率決定法に比べ計算量も減らすことができます。

この論文では、学習率決定法に周期関数を使用することで、様々なアーキテクチャのパフォーマンスを大幅に向上させることができることについて言及しています。また、周期的な性質を利用することで学習率を下げる時間(一般的に3~5サイクル後)、トレーニングを停止させるタイミングについても言及しました。これらの要因は、最適な学習率を探索する必要性を減らし、結果として学習フェーズ全体の効率化につなげることができます。

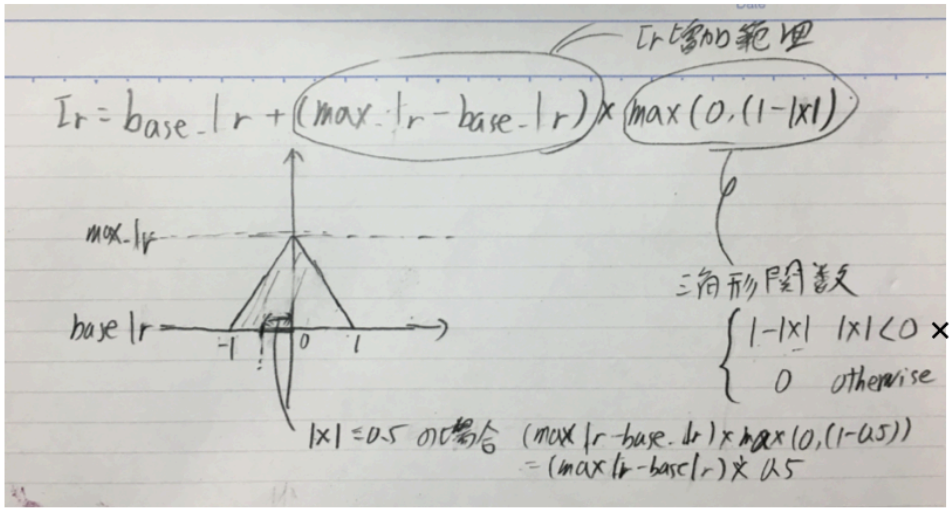
しかし、本研究では画像文やへのCLRの導入について研究を行っているだけであり、他分野での効果についてはまだ確認できていません。今後RNNなどのアーキテクチャについても検証していく予定です。

6. 式の理解とCLR実装結果

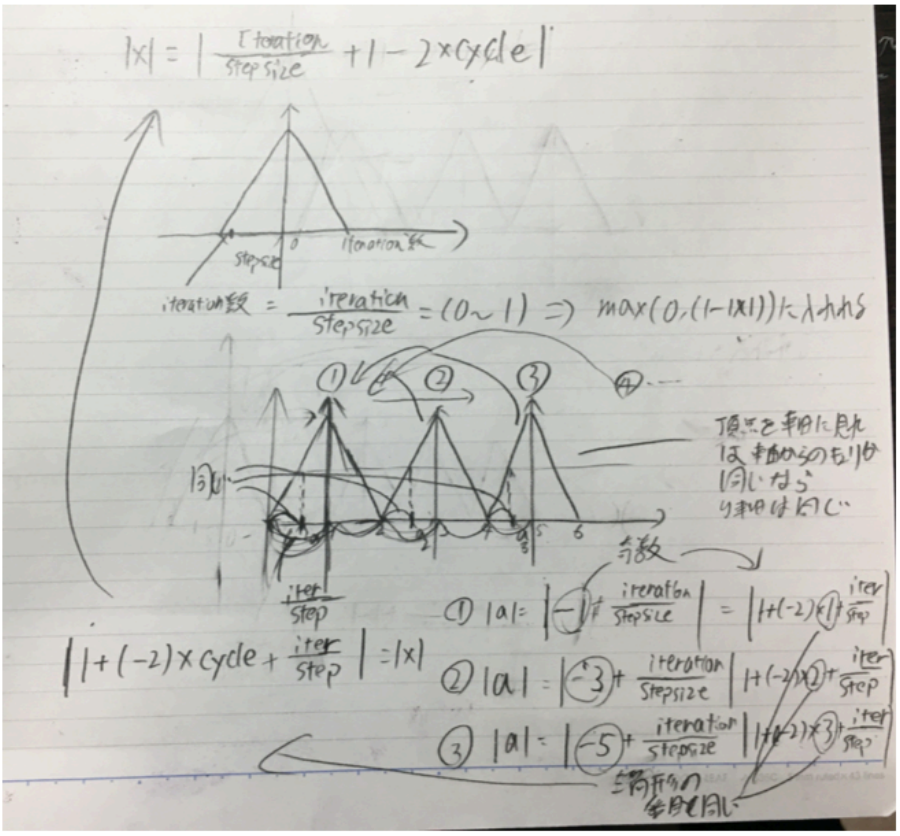
今回の論文では、プログラミング上でどのようにCLRを実装するのかのコードが載っているだけで各式についてどのように導き出したのかについての表記がなかった。そのため、各式の成り立ち等について自分なりにまとめてみた。

また、独自のデータセットでCLRを使用した場合にどのような変化があるかを検証したので以下に簡単に記します。

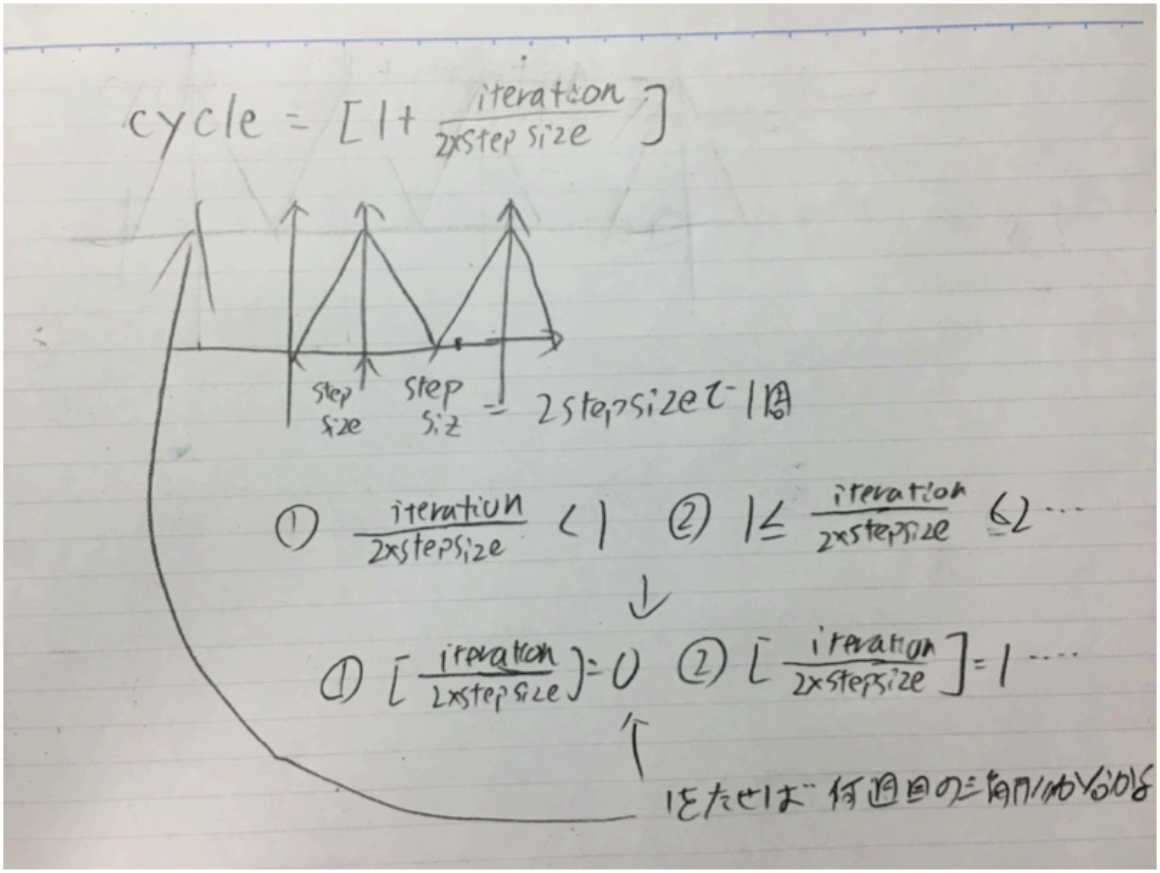
```
lr = base_lr + (max_lr - base_lr) * max(0, (1 - x))
```

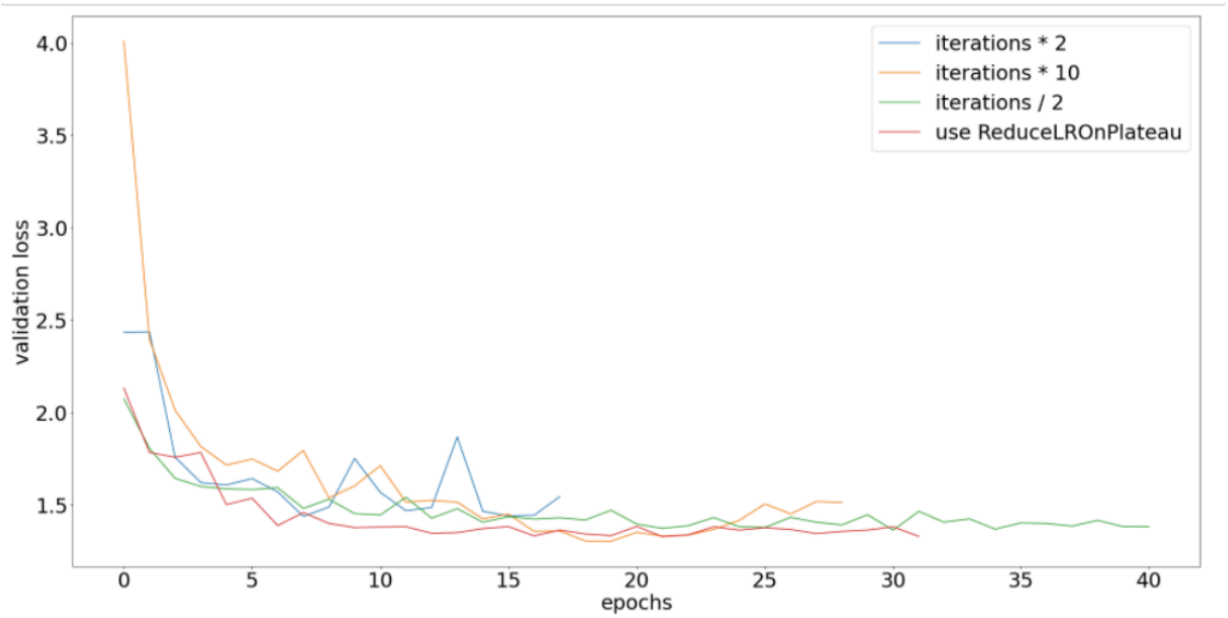



```
x = abs(iterations / step_size + 1 - 2 * cycle)
```



```
cycle = math.floor(1 + iterations / (2*step_size))
```





上のグラフはstep sizeをベースに対照実験(1つ以外の条件を同じにする実験法)を行なった結果を示します。GPUのスペックの問題で処理に時間がかかるためeary stoppingを使用し5回改善が見られなかったら学習を終了するように設定しました。また、「use PreduceLROnPlateau」はkaggle等でよく使用されている一定回数精度改善がみられなかった場合に学習率を下げるkerasのライブラリを使用して実験したものです。

step sizeが論文と同様の1エポック分のイタレーション数 × 2とした場合、学習は早く終了しましたが最高精度は「use PreduceLROnPlateau」より低い結果となりました。

step size を1エポック分のイタレーション数 / 2とした場合、精度の更新は長く続きそれに伴い学習時間も長引きましたが最高精度は「use PreduceLROnPlateau」より低い結果となりました。

step size を1エポック分のイタレーション数 × 10とした場合は、学習終了時間(学習終了までに要したepoch数)は「use PreduceLROnPlateau」とあまり変わりませんでしたが、最高精度は「use PreduceLROnPlateau」を上回りました。

学習率の増減率を小さくすることで、局所最適解を抜けつつ最適解付近を誤って抜けてしまうことを避けることができ、結果として精度が向上したと考えられます。

参考文献

Cyclical Learning Rates for Training Neural Networks