誰でも良い結果が得られる、学習率 探索プログラムの開発

A program development ― anyone can obtain a good validation result by this Learning Rate Explorer

藤原　隆弘[*1]
Takahiro Fujiwara

[*1]ドナウイヴァーロシュ大学（ハンガリー）
University of Dunaújváros (Hungary)

Abstract: In deep learning or machine learning, the hyperparameters learning rate and epoch count affect the performance and convergence speed of learning models [Jasper Snoek, 2012]. Any AI learning program have to think somewhere about the learning rate. However, choosing a good learning rate takes a lot of effort because it requires experience and long-term trials. This time, a learning rate exploring program is successfully developed through the knowledge in manual operation with trial and error in the Study Course "Deep Learning Basic Course" and good result is obtained. This paper introduces firstly the solutions, ingenuity points, and considerations of this program, then describe on its actual achievements and practicality.

1. はじめに

深層学習または機械学習では、学習率とエポック数が学習モデルのパフォーマンスと収束速度に影響する [Jasper Snoek, 2012]。どの AI 学習プログラムでも、学習率をどこかで考慮する必要がある。しかし、適切な学習率を選択するには、経験と長期にわたる試行が必要なため、多大な労力がかかる。

今回、学習コース「ディープラーニング基礎講座」での手探りの経験をもとに、学習率探索プログラムの開発に成功し、良好な成果が得られた。本稿では、まずプログラムでの解決方法、工夫点、考慮点を紹介し、次に実際の成果と実用性について述べる。

2. 実行環境・データセットや、参考手法

2.1 使用した実行環境

実行環境は、講座で使用した Google Colaboratory とした。

今までの実習で使っていた環境であり、データセットも簡単に利用できることが選んだ理由である。無料版で利用できる範囲で GPU を使用した。

2.2 データセットと目標、目的

データセットは、第 11 回講義の宿題である、「**変分オートエンコーダ（VAE）を用いて FasionMNIST の画像を生成してみましょう**」とした。これを選んだ理由は、手動で行っていた結果との比較もできること、そしてエポック 1 回の実行速度が比較的速いので、デバッグやテストを繰り返すのが苦になりにくいからである。

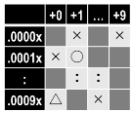
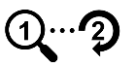
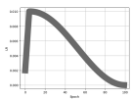
目標数値は宿題の目標と同じく、NLL（Negative Log-Likelihood: 負の対数尤度）=235 とする。なお、潜在空間の次元数(z_dim)を 10 ではなく 100 にすることや、Layer=64 を実装することで、エポック数が少なくても、NLL235 以下を達成することは事前に確認済みであるが、このような他の条件を変更せずに良い結果を得ることを本プログラムの目的とした。

連絡先: Takahiro Fujiwara (Hallgató),
Dunaújvárosi Egyetem, Táncsics Mihály út 1/a é,
+36 30 938 3708, dq4wx0@hallgato.uniduna.hu,
fujiwat0601@gmail.com

2.3 参考となるハイパーパラメータの最適化手法

参考となる最適化手法を、表 1 に記載した。

表 1 参考となる最適化手法

手法名	具体的な手法
グリッドサーチ 	あらかじめ指定した範囲内のハイパーパラメータの組み合わせをすべて試す方法。例えば、学習率を[0.001, 0.01, 0.1, ….]のような値から選び、評価結果が良い学習率を探す。手軽で初心者にとって一般的な手法。探索空間が大きく、時間と計算リソースがかかる。
ランダムサーチ 	ランダムサーチは、グリッドサーチの全数チェックの代わりに、チェックする値を、乱数を使って減らすもの。グリッドサーチよりも効率的なハイパーパラメータ探索手法とされる [James Bergstra, 2012]。
ベイズ最適化 	過去の試行結果を利用してつぎに試すべきハイパーパラメータの候補を選択していくことで、効率的に最適なハイパーパラメータ設定を見つける手法。探索フェーズと活用フェーズをバランス良く行いながら最適解に収束する [Eric Brochu, 2010]。
コサインアニーリング (Cosine Annealing) 	学習率を Cosine 関数の形状で変化させる。学習率をトレーニングの初めは大きく減らし、徐々に減少幅を小さくすることで、収束を改善し、より高い精度を得ることができる [Ilya Loshchilov, 2017]。主に深層学習に用いられることが多いが、機械学習にも利用可能。

3. プログラム開発

3.1 実装した、学習率の探索方式

まず、ハイパーパラメータの最適化文献を調べる前に自分の思う方式で探索方式を決め、コーディング・テストを実施した。

ある程度テストが進み、検証結果も出せるようになってきたところで論文を探し始め、グリッドサーチ、ランダムサーチ、ベイズ最適化を知った。

そして、自作したプログラムの探索方式は、ベイズ最適化と同じであると理解した。つまり、学習率の探索フェーズと活用フェーズが存在しており、探索フェーズで見つけた良い学習率を使って、実際の学習を行う。以下に、各フェーズの特徴を記述する。

3.2 探索フェーズ

探索フェーズは表 2 のように設計した。有効桁数を増やすごとにグリッドサーチを行っている。それを第 1 探索、第 2 探索・・・と表現した。

表 2 探索フェーズの仕様

探索の設計項目	具体的な仕様
(a)最初に探索する学習率	グリッドサーチとして、0.009, 0.008, …, 0.001, 0.000 の学習率を実行する。
(b)モデル学習の試行(探索)	学習率ごとにモデルを初期化。エポックを 50 回実行。毎回検証結果を調べる。
(c)検証結果の取り扱い	エポックの実行・検証を繰り返し、最良値(ここでは最小値)を、当該学習率に対する検証結果とする。
(d)探索が終了したら	結果が良かった順に学習率を 2 つに厳選。下位桁を持つ学習率を生成する。 例)0.003 から生成するのは 15 種で、$0.003 + 0.0001 * n, (n = 4, 3, \ldots - 10)$。 nが-10まで必要な理由は、下位桁の学習率をまんべんなく生成するため。 なお、重複する値は削除。この学習率リストを使って、(b), (c), (d)を繰り返す。(有効桁数 3 桁まで繰り返し)
(e)最終的に得られる学習率の数	第 1 探索:10 個→検証後 2 つに厳選 第 2 探索:2×15 個→検証後 2 つに厳選 第 3 探索:2×15 個→検証後 2 つに厳選
(f)このフェーズの総エポック数	連続$50 \times (10 + 30 + 30) = 3{,}500$ (これは最大。実際は重複を削除する)

3.3 活用フェーズ

活用フェーズは表 3 のように設計した。

表 3 活用フェーズの仕様

活用の設計項目	具体的な仕様
(a)学習率	探索フェーズで得られた学習率リストから順に取り出す。
(b)エポック数	それぞれの学習率に対して、エポックを連続して実施。毎回検証結果を調べ、悪くなったら、中断してつぎの学習率へ。
(c)学習率リストが終わったあと	(a)(b)をさらに繰り返して、良い検証結果を得る。
(d)このフェーズの総エポック数	一つの学習率に対して、400 回程度繰り返すとすると、$400 \times 2 = 8{,}000$回。

3.4 実装における工夫

プログラム開発中に気付いた問題点は表 4 のように工夫した。

表 4 実装上の問題点と工夫

問題点	具体的な工夫
有効桁数 2～3 桁の浮動小数点を扱っているだけなのに、有効桁数が 10 桁以上になることがある。	四則演算(特に乗算)で誤差が発生する。本来の有効桁数になるように四捨五入。
全エポックの終了後にのみ、テストデータを推測すると、途中の良い学習状態の結果が得られない。	エポックごとに検証結果を得るが、その時点で最良であれば、得られたモデルパラメータの保存と、テストデータを推測し、結果を CSV ファイルに保存。

3.5 実装における考慮点

プログラムの実装時に迷った点を表 5 にまとめた。

表 5 実装上の考慮点

考慮事項	実装方法	考慮事項
■探索フェーズ		
学習率の厳選方法	学習率を厳選するとき、良い検証結果を出す学習率の上位 2 個とした。	コンピュータのリソースに余裕があれば、5 個程度あった方が、候補を増やせて良い。
■活用フェーズ		
得られた学習率の適用順序	学習率を降順にする方法が、収束速度が高いと考えた。	検証結果が良い順に並べたほうが、収束速度が速い可能性もある。
検証結果の確認タイミング	エポックごとに検証結果を確認した。またそれによって結果が悪いときの学習パラメータ破棄も可能。	全エポックごとに確認するのは CPU コストがかかる。50 エポックごとに検証結果を計算するだけにする方法もある。

3.6 開発したプログラム

開発したプログラムのコードと実行結果は以下に置いている。

`https://github.com/fujiwat/DLBasic_2023_MyProject`

4. 実行結果

4.1 探索フェーズの実行結果

まず、第 1 探索として、0.009 から 0.001 の学習率に対して、それぞれ検証を含めてエポックを 50 回実行し、最良値(NLL では最小値)をグラフにした。ほかの学習モデルでも再利用しやすいよう、高い数値ほど良い値とするために、縦軸は$1/NLL$とした。横軸は学習率。

$$NLL = -\sum_{i=1}^{D} x_i \log \hat{x}_i + (1 - x_i) \log (1 - \hat{x}_i)$$

第一探索の結果(図 1)は、0.001 と 0.002, 0.003 の順で結果が良いことが読み取れる。ここから結果が最良だった 2 つの学習率に厳選する。この場合は、0.002 と 0.001 であった。

なお、得られた数値はグラフから読み取るのではなく、プログラムの print 出力による。

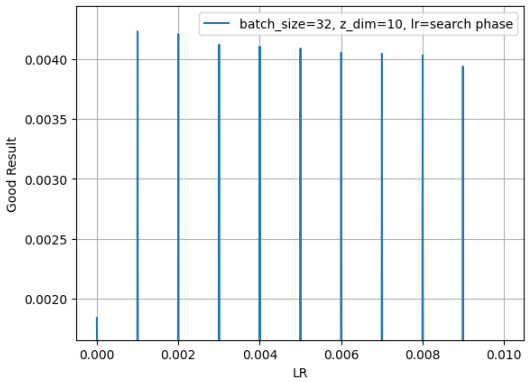


図 1　第1探索の検証結果

第 2 探索は、厳選された0.002と0.001に対して、0.0001間隔で、+0.0004～－0.0009を学習率にして実行した。結果を図 2 に表す。

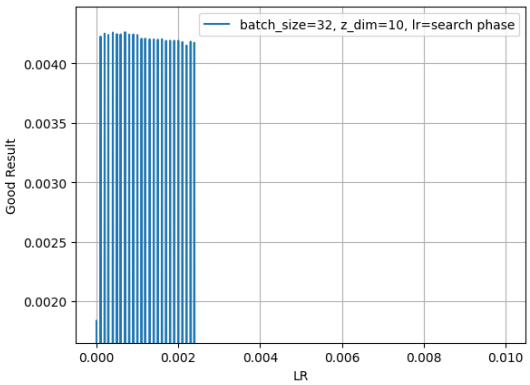


図 2　第 2 探索の検証結果

第 3 探索は、厳選された 0.0007 と 0.0004 に対して0.00001間隔で+0.00004～－0.00009を学習率にして実行する。結果を図 3 に表す。

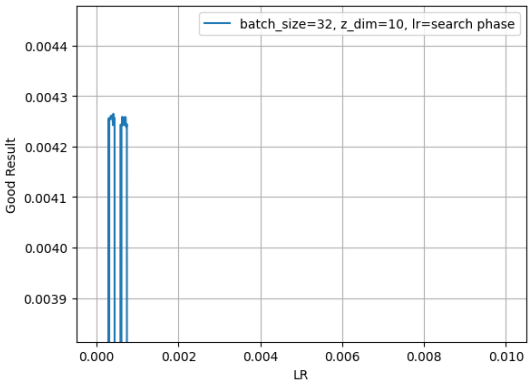


図 3　第 3 探索の検証結果

得られた学習率は 0.00041 と 0.00038 となった。

4.2　活用フェーズの実行結果

活用フェーズは、探索フェーズで得られた学習率を用いて、学習～検証を繰り返すものである。探索フェーズの最後に厳選された 2 つの学習率で、エポックを繰り返す。

活用フェーズの結果を例として、本プログラムで得られた最良だった学習率 0.00038 のグラフを掲載する（図 4）。ここでは、横軸をエポック数、縦軸を NLL とした。学習を繰り返すことで検証結果が良くなっている。

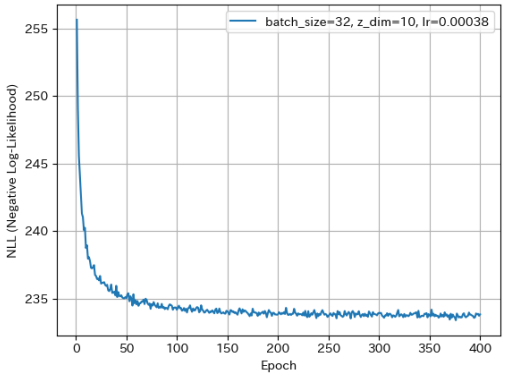


図 4　活用フェーズにおける検証結果（学習率 0.00038）

5.　他の学習率の設定手法との比較

つぎに、コサインアニーリング、および学習率を固定する方法でエポック数と検証結果の比較を行った（表 6）。

表 6　学習率の設定手法と、検証結果の比較（NLL 値）

エポック→ / ↓学習率設定	50回	100回	200回	400回
コサイン アニーリング 50	233.74	取得不可	取得不可	取得不可
コサイン アニーリング 100	234.44	👑233.30	取得不可	取得不可
コサイン アニーリング 200	234.75	233.95	👑233.15	取得不可
コサイン アニーリング 400	✖235.90	✖235.35	234.79	234.24
手探り 0.0003	✖235.11	234.16	233.66	✔233.53
探索で優良1位 0.00041	234.63	234.23	233.71	✔233.53
探索で優良2位 0.00038	✖235.01	234.17	233.61	👑233.39

✖目標未達、✔優秀な検証結果、👑 ベスト 3

表におけるコサインアニーリングに続く数字「50, 100, 200, 400」は、cosine カーブの分割数であり、この回数のエポックを実行し終わらないと良い結果が出ない。また、例えば「コサインアニーリング 50」であれば、50 回で学習率が限りなくゼロに近づき、それを超えるエポックを実行することはできない。よって、超える回数の NLL 値は「取得不可」と記載した。

「手探り 0.0003」は、経験と長い試行時間をかけて得た学習率である良い値がでているが、ベスト 3 には入らなかった。

「探索で優良」は、本プログラムの探索フェーズで行った 50 回連続エポックでベスト 2 に残った学習率である。それを固定値としてエポックをさらに連続実行した数値を記載。結果は 2 位。

コサインアニーリングは収束が早く、ベスト1も取得しているが、「コサインアニーリング 400」の行のように、すぐれない場合があることもわかった。これに対し、探索方式は、探索が優良であれば、活用フェーズでも優れた結果が出るようだ。

6.　まとめ

様々な学習モデルで検証していないので、一概には言えないが、今回の経験では、表 7 の通りである。このプログラムは、「（時間をかければ、手探りなく）誰でも、優れた学習率を見つけることができる」と言える。

コサインアニーリングは収束が早く良い結果が出やすいが、カーブの分割数を試行錯誤する点が「誰でも」とは言えない理由である。

表 7 学習率の設定方法（まとめ）

学習率の設定手法	利点、実装の複雑さ	欠点
コサインアニーリング	- 収束が早く、優れた結果となることが多い。 - 実装は比較的単純だが、テストは必要。	- 優れた結果にならないこともある。 - 分割数を事前に決める必要がある。良い分割数は試行錯誤が必要。 - 試行錯誤の結果は、最後まで実行しなければ判定できない。
手探り（固定値を繰り返す）	- 実装は単純 - わかりやすい	- 良い結果を得るには、経験と長期間に及ぶ試行錯誤が必要 - 労力がかかる。良い学習率は見つけにくい。
探索後に実行（本プログラム）	- 時間とコンピュータリソースをかければ、誰でも良い学習率を見つけることができる。 - プログラム開発は、ほかの比較方式より複雑。	- 1つの学習率に対して 50 回程度の試行は必要。 - 有効桁数を増やすためには、多くの学習率を生成して試行する。今回の探索フェーズで 3000 回以上の試行を行う必要があった。

最終課題を通して経験したことは、「**3.5 実装における考慮点**」に記載したように、より深い視点で深層学習を考えることができるようなったこと。そして、自分の考えに自信を持つこと。これは最初、探索プログラムに確信が持てず、「候補を多く残し、その代わりエポック数を減らす」ように実験していたが、良い結果が得られず何日も悩んでいた。その後「候補を絞り、エポック数を増やす」ことで成功したことからである。

良い学習モデルを設計するには、学習率だけではなく、新しい活性化関数 Mish, Swish や Optimizer や Loss Function その他多くのチェック箇所があることを心得ながらも、ほかの AI 学習モデルにおいても、時間を作って、今回の手法を試したい。

参考文献

deeplearning.jp. (2023). 参照先: deeplearning.jp: https://deeplearning.jp/

Eric BrochuM. Cora, Nando de FreitasVlad. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. https://arxiv.org/abs/1012.2599.

Ilya LoshchilovHutterFrank. (2017). Cosine Annealing with Warm Restarts. https://arxiv.org/abs/1608.03983.

James BergstraBengioYoshua. (2012). Random Search for Hyper-Parameter Optimization. https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf: jmlr.org/.

Jasper SnoekLarochelle, Ryan P. AdamsHugo. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf: papers.nips.cc.