Deep Learning Batch Selection Algorithm

Tanítási halmaz szelekciós algoritmus mélytanuláshoz

Tamás Imets
Budapest University of Technology and Economics
Miercurea Ciuc, Romania
imetstamas@gmail.com

Kacsó Péter Gábor
Budapest University of Technology and Economics
Miercurea Ciuc, Romania
kacsop@edu.bme.hu

Czibor Dóra
Budapest University of Technology and Economics
Komárno, Slovakia
dora.cziborova@edu.bme.hu

***Abstract (EN)* — In our experiment we tried to create and use different methods that can be utilized to select batches while training a neural network to make the learning process more efficient. Our team performed stress tests on a variety of artificial neural network architectures and datasets to see if the current methods would be able to beat random shuffling. Though the results are not convincing in all cases there are some models which could have an edge by using other methods. Our solutions are based on calculating losses beforehand and using that information to help the network learn on the data slices that cause more error. We examined whether the algorithms cause improvement in epochs, if they do at all, or in time as well.**

***Kivonat (HU)* — A kísérletünkben olyan tanítási halmaz szelekciós algoritmusokat terveztünk és tettünk próbára, mely tanulás közben igyekszik megfelelő halmazt választani, ez által gyorsabbá téve a tanulási folyamatot. A csapatunk széleskörű tesztelést hajtott végre számos architektúrán és adathalmazon, hogy betekintést nyerjünk abba, hogy melyik algoritmus képes felülteljesíteni a véletlen válogatás gyorsaságát. Bár az eredményen nem minden esetben meggyőzőek vannak olyan modellek, melyek előnyt tudtak felmutatni a naiv módszerrel szemben. Megoldásaink a modell hibájának előleges számítására alapoznak, hogy ezt a tudást felhasználva mindig azon az adatszeleten tanítsuk a neurális hálózatot, mely a legnagyobb hibát okozza. Megvizsgáltuk, hogy az algoritmusok csak tanítási ciklusokhoz viszonyítva gyorsítanak (ha egyáltalán gyorsítanak), vagy időbeli különbségben is mérhető a gyorsabb tanulás.**

Keywords—deep learning, batch selection, optimization, shuffle, batch, random, faster learning

I. Introduction

One of the main problems in the field of deep machine learning is that our trainable algorithms take a lot of time to process and learn on the provided datasets. There aren't so many solutions on how to make this process faster and to this day the best method is manufacturing better hardware that is able to accelerate computations needed for a neural network. Solving this problem is not trivial, no surprise that we still use shuffling and random batch selection to train the models, sometimes maybe with a smarter optimization algorithm.

Our goal with this project is not creating another optimization algorithm but to create new batch selecting methods and test previous solutions to see if they only work in certain conditions or there are some which perform well in different scenarios.

To measure the success of our ideas we are using a variety of datasets and neural network architectures to get relevant benchmarks. Some of these are well-known preprocessed and toy-datasets while we also create some of our own data for a more advanced comparison. The reason for using some toy-datasets as well is to have a better mathematical understanding about how our algorithms affects the learning process. We measure the success rate of our custom algorithm based on how much time is needed for the neural network to achieve a given accuracy. When working on different computers we also check the number of epochs needed to get to a given loss. These results are compared to random shuffling with a predefined random state.

The result analysis, log files, and our interactive plots can be accessed in our working repository. We strongly recommend the reader to download these files and do their own inspection to see if it's worth using a specific method with a specific architecture.

II. Previuous Solutions

Though there aren't many, most previous solutions tried to perform online batch optimization similarly to our *windowed* algorithm which is presented below. Their benchmarks look promising but it is only tested on a single dataset os it could be irrelevant when measuring with other models or databases.

III. Approaching the Problem Properly

One thing we thought previous solutions lacked is proper testing. Developing an algorithm and testing it on a single famous dataset with only one architecture can be an overfit and the final conclusion could be misleading. In this section we provide the reader a short demonstration on how we approached this problem and in the next one we're presenting each of our testing models.

A. *Testing*

We started this project with the principle of comprehensive testing, so our first step was defining what kind of models we are going to use. We ended up choosing 4 simple fully connected toy-models, 1 slightly more advanced feed-forward artificial neural network, 2 convolutional neural networks and one Word2Vec supported LSTM model. All of these models work on different datasets to cover even more fields of use. We prepared a visualizer and benchmarking script for our training logs to analyze which algorithms performed the best in the whole experiment.

B. *Architectures*

We assume that the reader has previous knowledge about ANNs, but to understand how different artificial neural network architectures are built and how they work, in this section we present them briefly, with a reference for more detailed descriptions for every architecture we use.

- **Simple Fully Connected ANN:** These are the simplest type of neural networks, consisting of layers in which every node is connected to every node in the neighbouring layers. We thought that it would be easier to see how selecting batches differently changed the behaviour of a network, if the network was easier to understand.
- **Convolutional Neural Network:** Convolutional Neural Networks are neural networks in which structured data, such as images, are not flattened and fed through fully connected dense layers, they go through 2D (or possibly with other dimensions) convolutional layers, more successfully capturing spatial dependencies in the data. They are widely used in image processing, that is why we use one to benchmark our algorithms.
- **Long Short-Term Memory:** LSTM-s are recurrent neural network architectures that have internal state and recurrently connected outputs.

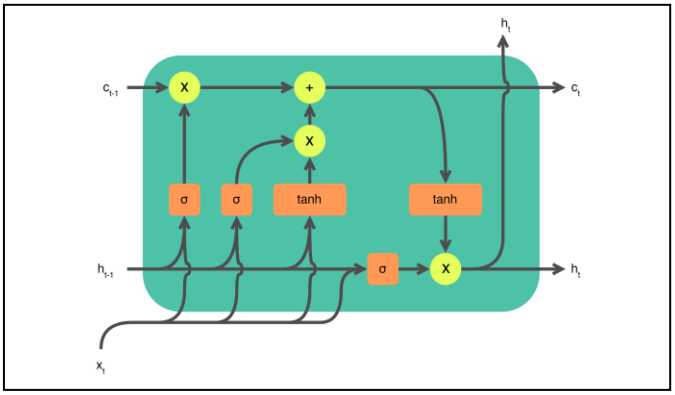

Fig. 1. LSTM neuron (source: en.wikipedia.org/wiki/LSTM)

- **Skip-gram:** This architecture is very simple one with a little twist. The model is mostly used in natural language processing (ex.: Word2Vec) to find the most related set of words for a given word. As its input we feed one-hot encoded vectors to it and a fully connected ANN tries to predict the neighboring or most related words – depends on how we prepare the data.

C. *Datasets*

After deciding what kind of models to test, our team selected corresponding dataset to each of our models. Below you can read about these datasets in a little bit more detail.

- **Twitter Sentiment140:** This dataset is a famous one for learning natural language processing, vectorial representations and performing sentiment analysis. Originally it was used by Stanford University for sentiment classification with distant supervision. The downloadable dataset is provided and made available by sentiment140.com and it contains more than 1.6 million tweets from Twitter which belong to two classes (positive and negative) that occur equal times, so the data is perfectly balanced. Since our computational capabilities are limited, for testing we're only using a randomly selected part of this dataset.
- **Archimedean Spirals:** These spirals are the 2D equivalent of a Swiss roll dataset. It is a simple classification task where the ANN (artificial neural network) has to learn to differentiate two spirals which start to grow in the opposite direction. The solution is simple, but it is visually spectacular and illustrative to see how a model learns with different algorithms. The dataset was generated by a simple Python script.
- **Boston Dataset:** A classic regression task is to predict house pricing based on a few attributes like crime rate, number of rooms, population status, distance to city center, etc. This dataset contains 506 cases and it is collected by the U.S. Census Service. It is provided by the SciKit-Learn framework for free usage.
- **Music Genres Dataset:** The dataset originates from the GTZAN Genre Collection, which consists of 1000 short audio tracks in 10 musical genres, the tracks distributed equally among the genres. Since the processing of audio signals is irrelevant in our work, we used a dataset that has the features of these audio tracks already extracted. These features include the zero crossing rate, spectral centroid, bandwidth, rolloff, chroma features and MFCCs, all of them averaged over the audio frames.
- **Wine Recognition Dataset:** The wine dataset loaded from the SciKit-Learn library contains data on wines grown in the same region by three different cultivators, with 59, 71 and 48 examples belonging to them. The wines are identified by thirteen different constituents found in them, like phenols and flavonoids, their values measured by a chemical analysis.
- **MNIST and Fashion-MNIST:** MNIST is a very famous dataset consisting of low resolution handwritten digits. It is widely used for training and testing machine learning solutions for image processing. We use it with a Convolutional Neural Network to see how our batch selection algorithm can help with speeding up the training of neural networks used for image recognition. Fashion-MNIST is a dataset which has the same layout

(image resolution, number of categories, dateset size), but instead of digits it contains images of clothing, and therefore it is harder to classify them. Both datasets can be loaded from Keras.

- **Iris Dataset:** The Iris dataset is a famous dataset consisting of iris flower measurements. We will use this dataset to benchmark our batch selection algorithm on a simple fully connected neural network. Compared to other datasets we use, this one is small, it has only 150 entries with 4 features.

D. Models

Since we have more than one model, each team member had their part in data collection, model building and hyperparameter tuning. Some of our models were easy to build and train, but there were some not so trivial. We provide a short description to each model for the reader to see what is included in our benchmarking system.

Generally, people use some libraries and optimized frameworks to train deep learning algorithms, but we had to access the training loop to modify the program's batch selection function. Based on Keras' documentation we wrote our own training loop cycle class which can be used by providing the program a specific loss function, training and validation metrics and optionally a batch selection algorithm. This solution was a lot slower than the standard *fit* function of this library because on default it used CPU as a hardware accelerator. As a workaround we used the Tensorflow framework's GPU accelerated functions so finally we got a similar, if not exactly the same training time as we would with the standard implementation.

- **Sentiment prediction with LSTM:** The most complex model of this project needs to differentiate emotionally positively and negatively charged sentences. Making a computer understand human languages is not an intuitive task. There are many methods on how to convert sentences to a numerical representation, but only so many can do it efficiently. We tried to avoid simple preprocessing methods like label encoding or one-hot encoding to extract as much information from a word as possible.

  We used Word2Vec which is a Skip-gram architecture based neural network that learns a vectorial representation of each word that occurs at least *n* times in a text. Word2Vec has a one-hot encoded vector as an input and tries to predict its neighboring words based on a textual dataset. The vectorial representation of a word is then finally given in the hidden layers of this network. We optimized this preprocessing tool based on the number of neighbors to predict and vector size.

  The W2V model was strictly trained on the training dataset and then later executed on the testing part. To provide correct input for the LSTM model we utilized the Word2Vec model's abilities and averaged the word vectors in a given window to achieve a fixed size matrix. Due to our computational power and memory size is limited, we used a tenth of this dataset which makes about 160k samples to work with. This step was necessary to test our batch selection algorithms efficiently without having to wait hours for a result.

  When building this network there was a slight overfit in the so we used dropout layers as regularization. We were able achieve 79% accuracy on this subset of the original data which totally acceptable on this dataset compared to the original methods (those peaked at 83% using the whole database). The model was compiled with *Adam* optimizer and binary cross-entropy loss function. As for the training, we used a fixed random seed for initialization and shuffling to get reproducible results when comparing to our algorithms.

- **Spiral Classifier:** This model was inspired by Tensorflow Playground. It is a really comprehensible task and easier to understand how batch randomization affects the network compared to a more abstract ANN. The preprocessing proceedings include multiplying, dividing and calculating the sinus values of the coordinates and optionally adding some noise to the

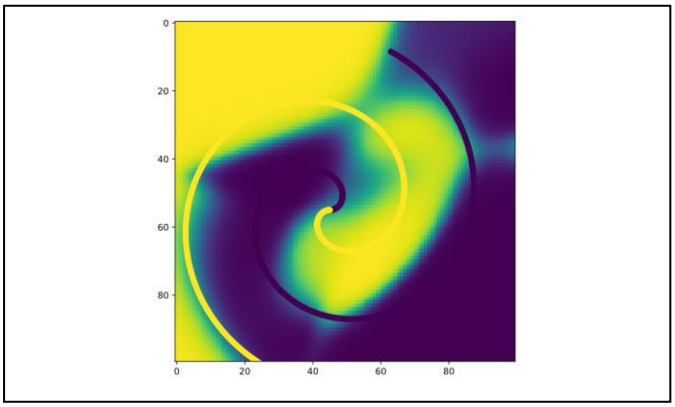

Fig. 2. Spiral dataset and prediction visualization.

  training dataset. The network itself is quite small, has only 1 hidden layer with 7 neurons. Again, *Adam* optimizer and binary cross-entropy helps the network to learn on the training data. It is very simple to achieve 100% accuracy on this toy-like task without noise, but we used a small learning rate with many epochs for better comparison logs. The testing accuracy this way decreased to 96.5%.

- **Regression Model:** The house price predicting problem has a quite a few working models online, but we created our own simple 4-layer model for a better approximation. Though it can be solved with linear regression the neural network added a noticeable edge compared to traditional methods.

- **Music Genres Classifier:** This model uses 4 fully connected layers, categorical cross-entropy loss function and Adam optimizer. It is very prone to overfitting due to the large number of features so dropout layers and L2 regularization are used to achieve higher generalization capability.

- **Wine Recognition Classifier:** Being our smallest model, the wine recognition classifier has two fully connected hidden layers with 15 and 8 neurons. Same as some of our other models, it uses Adam optimizer and categorical cross-entropy loss function. With small modifications this model could achieve 100% accuracy even faster, in just two or three epochs but the current architecture is better suited for evaluating our results on.
- **MNIST Classifier:** We used the same CNN model for both MNIST and Fashion-MNIST, as the latter is designed to be able to replace the first. The model has two convolutional layers and pooling layers after them. Our goal was to have a model that is simple, not necessarily the best, to see if our batch selection algorithms speed it up. It can achieve an accuracy of around 98% on MNIST and 90% on Fashion-MNIST.

IV. Algorithms

We tried two different algorithms for selecting batches and compared them to the default of random shuffling.

The first method, which we called Windowed, selects the batch with the greatest loss inside a window of a predefined length. This window moves through the whole training dataset. We hoped that by always selecting batches with greater losses, the networks would learn faster. Of course this method has its problems too, calculating the loss of every batch in the window takes time, and it is possible that certain batches would be selected too many times and others not enough times, resulting in the network not learning from important parts of the data. This could end up well too, if the batch selection successfully selects more of the important data pieces than the less important ones.

The second method, named Sorting, sorts the batches of the training data at the beginning of every epoch by the losses they generate in descending order. During training it selects batches earlier in the sorted data, batches with higher loss.

V. Result Analysis

The results were surprising. We expected to see a great improvement at least in comparing epochs, if not timestamp based accuracy. Calculating loss values beforehand took a lot of time but still we hoped for better outcome.

First, with the help of training log files, we tried to find which are the best performing algorithms when no training time is taken into account. A simple script does this by counting which methods have the highest training accuracy, validation accuracy and lowest training loss. Each of them is presented in table 1. that shows the number of times a given model won in a category (higher is better). Ties are allowed because the smaller datasets could provide similar results so there could be more than 8 points given in total in one category.

Table 1. Batch Selection Performance

Category	**Algorithms**		
	Random	***Sorting***	***Windowed***
Train Accuracy	3	3	3
Validation Accuracy	6	4	3
Training Loss	0	8	0

There is no clear winner based on the chart above, though it looks like the windowed selection model is lagging behind. The sorting algorithm achieved the lowest loss value every time, but this does not necessarily mean higher accuracy as we can see. Looking at the appendix we can observe how the sorting algorithm has usually the highest loss in the first epoch in almost every case but then instantly drops below the other two algorithms, which is due to the fact that it selects the batch with the highest loss first.

Table 2. Batch Selection Efficiency

Category	**Algorithms**		
	Random	***Sorting***	***Windowed***
Train Accuracy (Epoch)	5	1	2
Validation Accuracy (Epoch)	3	3	3
Training Loss (Epoch)	1	7	0
Train Accuracy (Time)	8	0	0
Validation Accuracy (Time)	6	2	0
Training Loss (Time)	2	6	0
Total Score	25	19	5

Table 2. on the other hand, shows how well the algorithms performed when we take time and epochs into account. To calculate this, we select the maximum accuracy of a given strategy and weigh this number by the time or number of epochs needed to achieve this. Having these two categories the scoreboard shows that the windowed selection algorithm does not even have a chance when taking time into consideration. It was a lot slower than the other two so this one is a clear loser of this competition. The sorting algorithm, however, was able to overperform the simple batch shuffling sometimes. Again, looking at the appendix, we could conclude that the sorting method was somewhat more efficient with complex models like LSTM and CNNs.

VI. Conclusion

Even though further testing is needed on this subject, it is really possible that a well implemented batch sorting algorithm can in fact accelerate learning.

References

[1] Understanding of a convolutional neural network, IEEE, Saad Albawi; Tareq Abed Mohammed; Saad Al-Zawi, 2017 https://ieeexplore.ieee.org/document/8308186

[2] Benchmarking and Analyzing Deep Neural Network Training, Hongyu Zhu; Mohamed Akrout; Bojian Zheng; Andrew Pelegris; Anand Jayarajan; Amar Phanishayee; Bianca Schroeder; Gennady Pekhimenko, http://www.cs.toronto.edu/ecosystem/papers/TBD-IISWC_18.pdf,

[3] Twitter Sentiment Classification using Distant Supervision, Alec Go; Richa Bhayani; Lei Huang, 2018, http://www.cs.toronto.edu/ecosystem/papers/TBD-IISWC_18.pdf

[4] Xin Rong, word2vec Parameter Learning Explained, https://arxiv.org/pdf/1411.2738.pdf, 2016, arXiv:1411.2738 [cs.CL]

[5] Thomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Efficient Estimation of Word Representations in Vector Space, https://arxiv.org/pdf/1301.3781.pdf, 2013. szept. 13., arXiv:1301.3781 [cs.CL]

[6] Sepp Hochreiter, Jürgen Schmidhuber, Long Short-Term Memory, https://www.bioinf.jku.at/publications/older/2604.pdf, 1997

[7] Understanding LSTMs, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, by.: Cristopher Olah, 2015, Downloaded on: 2020-12-01

[8] Online Batch Seleciton for Faster Trainign of Neural Networks, Ilya Loshchilov & Frank Hutter, https://arxiv.org/pdf/1511.06343.pdf, 2016, arXiv: 1511.06343v4

[9] Improving the Convergence Speed of Deep Neural Networks with Biased Sampling, George Iannou, Thanos Tagaris, Andreas Stafylopatis, 2019, https://doi.org/10.1145/3369114.3369116

[10] wevi: word embedding visual inspector, https://ronxin.github.io/wevi/, by.: Xin Rong, 2016. February, Downloaded on: 2020-12-01