Universidade Vigo

Trabajo Fin de Máster

GEDA: A Novel Subset Selection Method for Data Efficiency

Jose Antonio Cribeiro Ramallo

Máster en Técnicas Estadísticas

Curso 2021-2022

Propuesta de Trabajo Fin de Máster

Título en galego: GEDA: Un novo método de selección de subconxuntos para Data Efficiency
Título en español: GEDA: Un nuevo método de selección de subconjuntos para Data Efficiency
English title: GEDA: A novel subset selection method for Data Efficiency
Modalidad: Modalidad B
Autor/a: Jose Antonio Cribeiro Ramallo, Universidade de Santiago de Compostela
Director/a: Marta Sestelo Pérez, Universidad de Vigo
Tutor/a: Marcos Baptista Ríos, Gradiant
Breve resumen del trabajo: En este trabajo se ha realizado una investigación en métodos para Data Efficiency, donde se han estudiado los métodos del SOTA de Data Efficiency y Data Valuation. Posteriormente se ha introducido nuestro nuevo método: GEDA, él cual hemos comparado con otros métodos del actual SOTA.
Recomendaciones:
Otras observaciones:

Doña Marta Sestelo Pérez, Dra. de la Universidad de Vigo y don Marcos Baptista Ríos, Investigador Senior de Gradiant, informan que el Trabajo Fin de Máster titulado

GEDA: A Novel Subset Selection Method for Data Efficiency

fue realizado bajo su dirección por don Jose Antonio Cribeiro Ramallo para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Vigo, a 28 de Enero de 2022.

La directora:

SESTELO PEREZ MARTA - 76998300G

Firmado digitalmente por SESTELO PEREZ MARTA - 76998300G
Fecha: 2022.02.02 13:54:14 +01'00'

Doña Marta Sestelo Pérez

El tutor:

BAPTISTA RIOS MARCOS - 09043584F

Firmado digitalmente por BAPTISTA RIOS MARCOS - 09043584F
Fecha: 2022.02.02 13:38:11 +01'00'

Don Marcos Baptista Ríos

El autor:

Don Jose Antonio Cribeiro Ramallo

Contents

Abstract

Resumen en español

En el paradigma actual de Machine/Deep Learning se tiende a utilizar bases de datos cada vez más grandes, lo cual conlleva a un aumento en el coste computacional del modelo. Esto significa, entre otras cosas, un aumento en el impacto medioambiental y en el coste financiero del modelo. Una forma de atacar este problema sería reducir el tamaño de las bases de datos empleadas, sin comprometer gravemente la precisión del modelo entrenado. En este trabajo, proponemos un nuevo método de selección de subconjuntos llamado GEDA (Gradient norms Empirical Distribution Approximation), que obtiene los subconjuntos resolviendo un sencillo problema de programación lineal de enteros mixta. Además, demostramos la validez de este método a través de una serie de experimentos empleando conjuntos de datos de visión artificial, en donde encontramos que emplear GEDA conlleva a un aumento en la velocidad de entrenamiento cuando empleamos un pequeño número de *epochs*. Finalmente, comparamos GEDA con los modelos de selección de subconjuntos del estado del arte actuales, demostrando que GEDA consistentemente supera los resultados obtenidos con el resto de métodos durante entrenamientos cortos.

English abstract

In the current Machine/Deep Learning paradigm, there is a tendency to use larger and larger databases, which leads to an increase in the computational cost of the model. This means, among other things, an increase in the environmental impact and in the financial cost of said models. One way to attack this problem would be to reduce the size of the databases used, without compromising the accuracy of the trained model. In this paper, we propose a new subset selection method called GEDA (Gradient norms Empirical Distribution Approximation), which obtains subsets by solving a simple mixed integer linear programming (MILP) problem. Furthermore, we demonstrate the validity of the method through a series of experiments using computer vision data sets, where we find that employing GEDA leads to an increase in training speed when we employ a small number of epochs. Finally, we compare GEDA with current state-of-the-art subset selection models, showing that GEDA consistently outperforms all other methods during short training runs.

Chapter 1

Introduction

Deep Learning is, without a doubt, one the most impactful technologies of the last decade, inspiring an entire generation of scientists to explore the field and research on its applications. However, Deep Learning-based methods are not really new. The first one dates back to 1958: Rosenblatt's Perceptron [1]. A computationally not implementable method until the appearance of the Back-Propagation algorithm proposed by Werbos [2] 18 years later, in 1974. Although the mathematical definition of the model was available, its implementation required such a massive level of computational resources at that time that it simply was not feasible, so the study of the methods was put on hold.

Over the years, the computational capabilities of the modern computer have advanced by leaps and bounds, which, together with the advances in parallel computing and the incessant increase in the amount of data collected every day, have put these methods back on the map, gaining a massive surge in popularity, due to the fact they can be used inside a plethora of different cases inside Machine Learning, and even helped solved problems that were previously unfeasible.

Although the power of Deep Learning models is compelling, they presents two major problems: they are becoming more and more computationally heavy and they tend to be trained with large volumes of data. This might prove to be a problem for small companies and research groups that might not have sufficient resources.

1.1 Motivation and objectives

The main resources we have for training Deep Learning models are: computational power and data. For large companies, increasing their computational power is simple, since they have the necessary resources to scale their servers as needed. This is, sadly, not the reality for the rest of companies or even university research groups, where buying a dozen of new GPUs to increase their power is not something viable; therefore, we need to find a different avenue to reduce the time it takes to compute said models. Since the computation times comes in hand with the number of data used (the more data, the more is going to take to train with it), reducing it would lead to a direct time save. This begs the question, how can we reduce the amount of data used, without hindering the performance of the resulting model?

While seemingly a simple question, the answer is completely unclear, since it deeply depend on how you want to train your model. There are a plethora of state of the art models that aim to solve this issue from different perspectives. For example, some models are really good at detecting and removing data that has been miss-labeled or that is noisy, others are really good at reducing a substantial amount of the data (70 to 80 percent) and reach similar results as those obtained during long trainings with the full data set. Although they have strengths of their own, they also present different weaknesses

that might be a deal breaker to us: either they are computationally expensive, extremely complex or they under-perform with shorter training processes.

In view of the state of the art, our objectives will be to study the effect on model performance of reducing the training data, and having this knowledge, to design a data selection algorithm capable of reducing the size of the data set, while still being computationally simple. With these objectives in mind, we have developed a model capable of reducing the data set, while minimizing the hit in performance, that we call *GEDA*. This model yields better results for shorter training processes in comparison to the other state of the art models, and we are going to achieve that with only minimizing a simple MILP (mixed integer linear programming) problem at the beginning of the training. Additionally, we test its efficacy for Deep Learning models for Computer Vision, a task where we have to work with images, and where it usually takes a large amount of time to train its models. The main applications of our method would be:

- Given a desired percentage, κ, GEDA is capable of removing the $(100-\kappa)\%$ of the training data set while having a smaller hit in performance for shorter training, than the other state of the art models.
- GEDA can be easily adapted into an online algorithm, where instead of selecting once the data, we are going to select a new subset every R epochs.
- Due to the good results during short trainings, GEDA could work as a great warm start method when using the entire data set is too prohibitive.

1.2 Structure of the document

Through out the following chapters we will enunciate the solution to the problem of reducing the data set while minimizing the performance hit. At the same, we will give context to the document, briefly introducing basic concepts about Deep Learning and Computer Vision.

- In Chapter 2 we will do a brief introduction to deep learning for Computer Vision, where we will introduce the reader to basic definitions and methodology, with the aim to familiarize the reader with the basic notation and nomenclature that we are going to use in the next chapters.
- Chapter 3 focuses on introducing the current state of the art methods for data efficiency and data valuation.
- Moving on, in Chapter 4 we will introduce the our proposed method for efficient subset selection: GEDA.
- In Chapter 5 we will lay down all the research that led to GEDA, as well as a comparison of our method with the current state of the art.
- Finally, in Chapter 6 we will talk about possible uses of GEDA, as well as improvements that could be perform on the method, that could prove to be a possible future work on the topic.

Chapter 2

Introduction to Deep Learning for Computer Vision

The importance of Computer Vision comes from the fact that everything could potentially be processed as an image or collection of images. This includes X-rays, faces, videos, PDF files and even sound files by processing an image of the sound waves. While there are a large number of methods within Computer Vision, the most widely used today are based on Deep Learning.

The aim of this chapter is to briefly introduce Computer Vision, as well as some concepts and ideas of Deep Learning, and the methods for its optimization, that are going to be essential for the research presented afterwards.

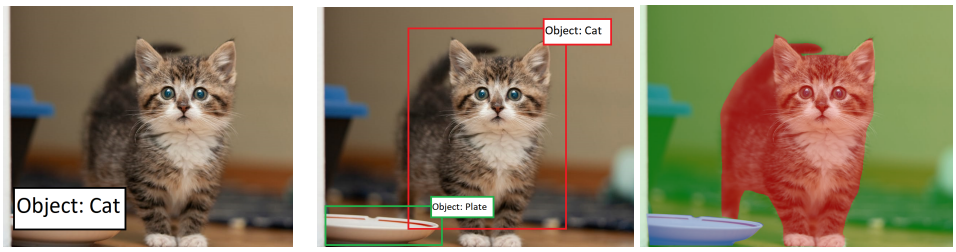
2.1 Computer Vision

In general, Computer Vision is the collection of techniques used to provide a computer with the ability of processing the information that an image or video contains. Once this information is understood, the computer can carry out a variety of tasks. The most common ones are described below and shown in Figure 2.1:

- **Classification.** The task of classifying or categorizing an image based on the pixels within it. See figure 2.1a.
- **Object Detection.**The task of localize and classifying objects inside an image, indicating their location with a bounding box. It can be thought of as the combination of the classification task with object localization. See figure 2.1b.
- **Segmentation.** The task of partitioning an image into multiple image segments (set of pixels within an image). See figure 2.1c.

In modern Computer Vision, Machine Learning, and specifically Deep Learning models, yield the best results and it has been like this for the last 10 years. In 2010, the ImageNet data set [3], containing millions of images inside thousandths of categories, became available for researches. On a Computer Vision competition utilizing the ImageNet data set, Krizhevsky et al. [4] introduced in 2012 the first Convolutional Neural Network (CNN for short) called *AlexNet*. This new model managed to revolutionize the state of the art at the time, marking the biggest breakthrough inside Computer Vision. Since then, Convolutional Neural Networks have been the default inside any Computer Vision task.

A similar breakthrough happened in 2016, when He et al. [5] entered another Computer Vision competition with the ImageNet data base, with a new type of CNN *Residual Net* (ResNet). Currently,



(a) Classification (b) Object Detection (c) Segmentation

Figure 2.1: Visualisation of the Computer Vision main tasks.

variations of the ResNet are some of the state of the art models, and have been since the introduction of the first ResNet. Other relevant models are Vision Transformer [6] and I3D [7], among others.

2.2 Incremental Gradient Methods

Gradient descent is one of the (if not the) most well known methods for global optimization of differentiable multivariate real functions. It is performed in a iterative fashion by, given said differentiable function $F : \mathbb{R}^n \to \mathbb{R}$ and its gradient ∇F, select the next iterator as:

$$a_{i+1} = a_i - \gamma \nabla F(a_i),$$

where γ, the learning rate, symbolizes the size of the step in the direction given by $-\nabla F$ (known to be the direction of maximum descent). The learning rate is going to be selected, either by selecting the optimal value of γ on each step, picking a fixed value or defining a function that selects the value of γ on each step (called the *scheduler* function).

Now consider a given dataset $D \subset \mathbb{R}^n$ with points $x_i \in D$, and a differentiable parametric real function $\mathcal{L}_\theta$ written as:

$$\mathcal{L}_\theta(L_\theta, D) = \sum_{i=1}^{n} L_\theta(x_i),$$

with $L_\theta : D \to \mathbb{R}$ a differentiable parametric function. The gradient of said function $\mathcal{L}_\theta$, over the parameters θ, can be written as:

$$\nabla_\theta \mathcal{L}_\theta(L_\theta, D) = \sum_{i=1}^{n} \nabla_\theta L_\theta(x_i).$$

Note 2.1 *Since we are only going to consider the gradients over the parameters θ, we are going to write $\nabla L_\theta = \nabla_\theta L_\theta$ indistinctly.*

Under this hypothesis lays the next definition:

Algorithm 2.1 *Given a random subset of D (let's say $B \subset D$), Stochastic Gradient Descent is an iterative algorithm for the optimization of multivariate real functions defined by the sum of differentiable functions. The algorithm performs as follows:*

1. *Draw a random selection of points $B \subset D$ such as: $|B| = k_{batch}$*
2. *Perform an iteration of gradient descent with B over the parameters θ_i:*

$$\theta_{i+1} = \theta_i - \gamma \sum_{x \in B} \nabla L_{\theta_i}(x)$$

3. *Update* $D = D - B$

4. *Repeat step 1 through 3 until* $D = \emptyset$.

Performing the previous steps is commonly known as an epoch. Stochastic Gradient Descent (SGD) is commonly performed over multiple epochs.

Note 2.2 *Keep in mind that, given a random selection of points* $B \subset D$*, we can say that:*

$$\sum_{x \in B} \nabla L_\theta(x) \approx \sum_{x \in D} \nabla L_\theta(x).$$

All the methods that try to approximate the optimization performed with the full gradient are called Incremental Gradient Methods. Examples are Stochastic Approximate Gradient Descent [8], SGD with momentum [9] or SVRG [10].

Stochastic Gradient Descent is employed as an alternative to regular Gradient Descent to fix problems with the optimization surface (like zig-zagging), to avoid local minima that GD might get into or even when calculating the full gradient $\nabla \mathcal{L}_\theta$ is not feasible, all of those being problems that we could face when working with Neural Networks.

2.3 Introduction to Neural Networks

Deep Neural Networks have become the standard tool for solving a plethora of problems inside Machine Learning, been wrongly though of like an imitation of the human brain's working. In the end, neural networks are just a composition of differentiable parametric functions that, so far, has not been found to have any relation to an actual human brain. The following definition gives a general understanding of the type of models that we are going to be working with.

Definition 2.2 *Given* $\{f_j\}_{j=1}^{k_{lay}}$ *differentiable functions, where each* f_j *has an amount of parameters* n_j *(not necessary greater than 0)* $\{\theta_i^j\}_{i=1}^{n_j}$*, a general Neural Network can be defined as the composition of all the* f_j *(generally called hidden layers, with its components* $\{f_{ji}\}_{i=1}^{M_j}$ *generally referred as neurons) with the composition of another function* g *(known as the output layer):*

$$f_\theta(x) = g(f_{k_{lay}} \circ \cdots \circ f_1 \circ I(x)),$$

This last layer g *can change between tasks. Generally, for regression, we tend to use* $g = id_X$ *the identity function, whilst for* k*-class classification we use the* $softmax$ *function*[1] *composed with the max function. Finally,* I*, called the input layer, handles the data points to the network in a shape that can be easily processed. When working with simple data (like tabular data) we use* $I = id_X$*, but for more complex data, like images or sentences,* I *presents a more complex structure (see figure 2.3).*

Now, given the model f_θ, we are going to determine the value of the parameter via solving the optimization problem:

$$\min_\theta \sum_{i=1}^{n} L(f_\theta(x_i), y_i),$$

where L is typically chosen to be the cross entropy function inside the k-classification task. In practice, is typical to sum a term $\omega\|\theta\|_2$ to the minimization objective function, to help the convergence of the

[1] The $softmax$ function, $\sigma : \mathbb{R}^K \longrightarrow [0,1]^K$, is defined by:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_k}}$$

parameters. The term ω is called weight decay. More information about weight decay and any other hyper-parameter can be found in [11].

Because we are working with the sum of differentiable functions, we are able to use the algorithm 2.1 to solve the optimization problem. The process of obtaining such adjusted parameters $\hat{\theta}$ is often referred as the training process, and the data we use during is usually called the training set. At the same time, we will measure the validity of the model with some performance metric, the calculation of which is going to be obtained using a different set than the one which we trained the model with. We are going to call this last set the validation set.

The general definition of Neural Network allows for complex architectures. This is such the case for the Networks that we are going to use, Convolutional Networks. Convolutional Networks (or ConvNets) receive their name for the use of the convolution operation inside each hidden layer. Now, each layer f_l is going to be:

$$X_j^l = f_{lj}(X^{l-1}) = h(b_l + \sum_{i=1}^{M_{l-1}} K_{ij} \otimes X_i^{l-1}), \quad \forall j = 1, \dots, M_l, \ \forall l = 1, \dots, k_{lay} - 1;$$

where K_{ij} are linear functions[2] defined for every pair of neurons of different layers, b_l is a bias term and h is a non-linear function (called the activation function). At the end of the convolutional layers, ConvNets add a fully connected layer (derived from the Multi layer Perceptron network), so the last layer $f_{k_{lay}}$ can be defined through its components as follows:

$$f_{k_{lay}j}(X^{k_{lay}-1}) = b_{k_{lay}} + w_l^T \cdot X^{k_{lay}-1}, \quad \forall j = 1, \dots, M_{k_{lay}};$$

with w_l^T the vector of weights. This way, each neuron output $f_{lj}(X^{l-1}) = X_j^l$ is going to be a vector[3]. ConvNets typically present a pooling layer as well, which is just a dimension reduction, every few number of layers (check Figure 2.2 for a visual representation).

Note 2.3 *Further elaboration for the calculations of the ConvNets Loss function's gradients can be found in [12], as well as more in-depth discussion of ConvNets.*

We have introduced all the necessary information about Neural Networks, but more information about it can be found in [13].

[2] K_{ij} are referred as *the convolutional nucleus of the connection of the i-th neuron from the previous layer (X_i^{l-1}) with the j-th neuron of the current layer (X_j^l)*, and are trained alongside all the other parameters of the model.

[3] Keep in mind that at the convolutional layers each neuron outputs an image handled as a vector (check Figure 2.3), and at the FC layer the network handle all of the outputs of the last convolutional layer as a unique vector (just like in the input layer, shown at Figure 2.3)

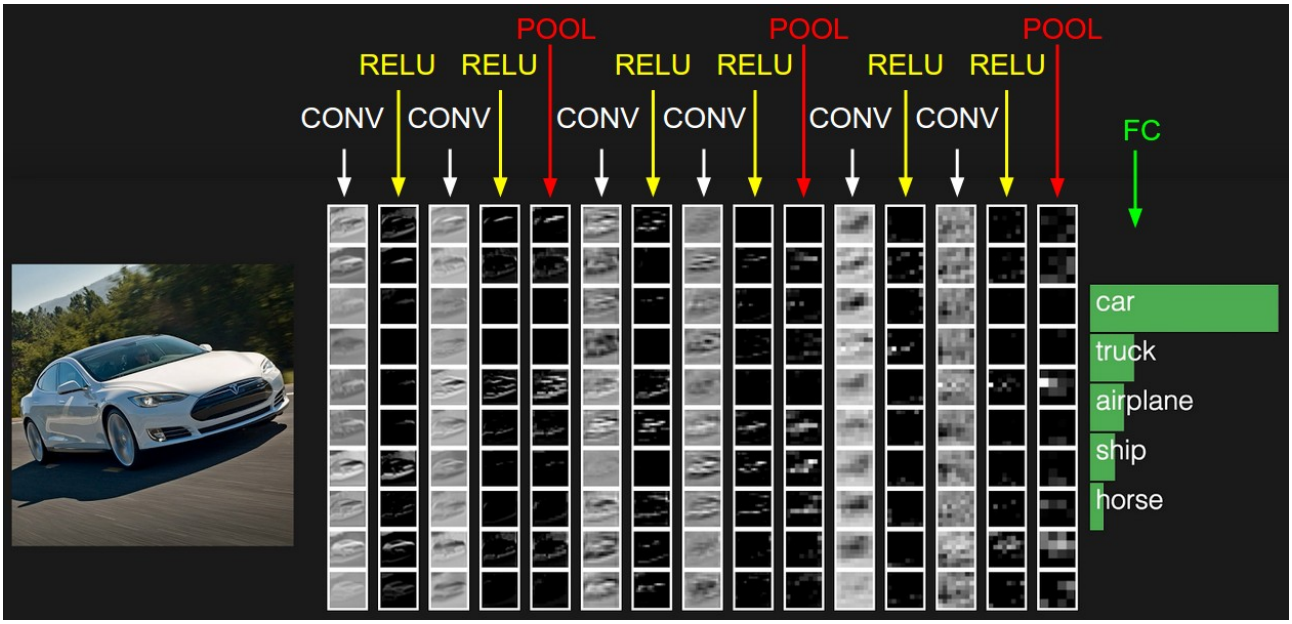


Figure 2.2: ConvNets architecture diagram. In this example, the activation function, h, of each hidden layer is going to be the ReLu function, and the output layer of the network, g, can be defined through its components like $g_j(x) = \sigma(x)$, with $\sigma(x)$ the sigmoid function. Source: `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf`

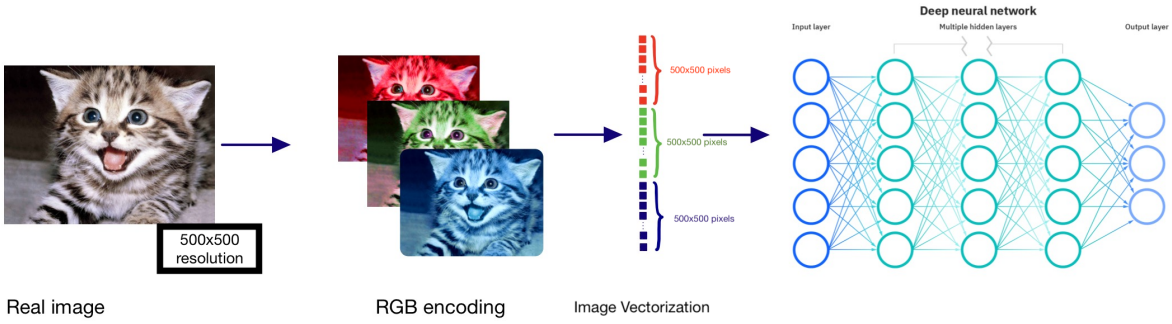


Figure 2.3: Image handling schema performed on the input layer of Convolutional Networks.

Chapter 3

Related Work

Even though inside the field of Data Efficiency and Data Valuation exists the idea that both refer to the same field of knowledge, it is important to clarify that this is not really true. Data Efficiency is the collection of techniques that seek to improve the efficiency of the training process for a Machine Learning model. This efficiency might come in the sense of energy efficiency [14], memory efficiency, storage efficiency, etc. But, in general, efficiency is often referred as a reduction of the training set for obtaining a similar result as the model trained with the full dataset[1]. It can be though of as the collection of techniques for answering the question: given a fraction of the dataset, how can I obtain similar results to those obtained using the full dataset?

Data Valuation has a more simpler definition: the collection of techniques that aim to assign value to all of our data, based on how useful it is for the training process. It can be though of as the collection of techniques for answering the question: given my training data, how useful is it?

Usually, the confusion comes because the most common scenario for the use of Data Valuation techniques is inside Data Efficiency, but it is important to keep in mind that it is not the only applicable scenario. Data Valuation is also used for modeling the data marketplace problem (which was the first time that it was introduced), and other complex problems that take roots in decision theory (like federated learning [15]).

It is important to clarify as well, that Data Valuation techniques for Data Efficiency give a more explainable solution for the Efficiency problem than what a general Data Efficiency method might give, but they tend to be more computationally heavy. The research presented here is inside the Data Efficiency framework due to this last reason.

3.1 Data Valuation Methods for Data Efficiency

Let us define for the rest of the chapter $D_T = \{(x_i, y_i)\}_1^n$ as the training set (not necessarily i.i.d.), $D_V = \{x_i^v, y_i^v\}_1^m$ as the validation set (not necessarily following the same distribution as the points in D_T), $\mathcal{A}$ as a learning algorithm that given D_T produces a predictor f, and $V(f) = V(\mathcal{A}, D_T)$ as a performance metric.

Now, under the previous hypothesis, Data Valuation tries to find a solution to both this questions:

1. What is an equitable measure of the value of each (x_i, y_i) to the learning algorithm $\mathcal{A}$, with respect to the performance metric V?

[1] When we are talking about the full dataset, keep in mind that we are talking about the training dataset.

2. How do we efficiently compute this data value in practical settings?

If we had such data value, we could define a subset $\mathcal{X} \subset D_T$ using some criterion for the data valuator (removing points with low data value, for example). We identify two main ways to valuate data for efficient training: the Cooperative Game approach and Reinforcement Learning.

3.1.1 Cooperative Games

Ghorbani et. al. [16] were the first to answer the previous questions using the Shapley value of a cooperative game, via a value called Data Shapley. Essentially, they argued that the only way for a data value $\phi_i(D_T, \mathcal{A}, V) \in \mathbb{R}$, for datum $x_i \in D_T$, to be equitable is by satisfying 3 properties: symmetry, additivity and fair contribution (if the datum contribution to every coalition is 0, then its value has to be 0). They find such data values by defining a cooperative game where each datum is a player, so the training data points $x \in S \subset D_T$ have to work together to achieve the prediction score $V(\mathcal{A}, S)$. This way, the Shapley value (a type of solution for a cooperative game) of the previous cooperative game is the only possible value verifying the previous properties (the full demonstration can be found in [16]).

While having interesting equitability properties, calculating the Data Shapley for our entire training set is exponentially large in the training set size, and involves learning a new f via $\mathcal{A}$ for each subset $S \subset D_T$ (i.e, learning $2^{|D_T|}$ predictors). As a result, calculating the exact data Shapley of each datum ends up being computationally impossible for Deep Learning, since the models are not trivial computationally speaking. Multiple methods for approximating the Data Shapley are given in the original paper, using simulation methods like Monte Carlo and other approximations obtained at the same time as the training process, but even though they work on simpler models, they still require an elevated amount of evaluations of $\mathcal{A}$.

Ghorbani et. al. [17] expanded on their previous work developing a distributional framework for Data Valuation that further improves the estimation and reduces the computational complexity, with the *Distributional Shapley*, which can be defined as the expected data Shapley value over all the data set of a given size, k, containing the point: $\mathbb{E}\left[\phi_i(D_T, \mathcal{A}, V)\right]$.

To close the topic of cooperative games, note that other solutions have been proposed as a potential valuator in [18], where they proposed the Core of the cooperative game defined by the data, achieving a solution computationally simpler than Data Shapley that yields better results for efficient learning with smaller data sets and lower epochs.

Note 3.1 *Because the data Shapley of each datum is calculated based on their contribution to all possible combination of subsets during the training process, data that worsen the training process (poorly labeled or noisy data, for example), are going to necessarily receive close to 0 or even negative values. This is a trait specific to Data Valuation methods for Data Efficiency.*

3.1.2 Reinforcement Learning

Yoon et. al. [19] introduced a method that involves learning a new model (called the Data Valuator Estimator, h with parameters ϕ) over the training process of the previous classificator model. This supposes a considerable increase in complexity, as well as the reduction in explainability of the valuation given to each datum.

The problem can be reduced to finding both ϕ^* and θ^* such that:

$$\begin{aligned}\phi^* = \ & \arg\min_{\phi} \quad \sum_{i=1}^{m} L_h(f_{\theta*}(x_i^v), y_i^v) \\ & s.t \quad \theta^* = \arg\min_{\theta} \sum_{i=1}^{n} \hat{h_\phi}(x_i, y_i) L_f(f_\theta(x_i), y_i),\end{aligned}$$

where L_f and L_h are (not necessarily different) loss functions, and $\hat{h}_\phi(x_i, y_i)$ is a binary selector constructed someway with the output of $h_\phi(x_i, y_i) \in [0, 1]$ (for example, by using it as the probability of success of a Bernoulli experiment). The main complexity of the problem lays on the selectors $\hat{h}_\phi(x_i, y_i)$, due to the non-differentiability of such variables. This, though, is nothing new in Machine Learning, as selection variables are something that appear on a lot of popular models (like Generative Adversarial Networks). The authors proposed a joint optimization method for both f and h, where θ_i get updated based on the current value of ϕ_i and the latter is updated using the REINFORCE algorithm proposed by Williams et al. in [20].

Das et. al. [21] proposed a different way to solve the selection bottleneck. Instead of utilizing the REINFORCE [20] algorithm, they solve the optimization bottleneck by updating their function h's parameters, ϕ, based on the results of a convex optimization problem. [2]

While both of these approaches have been proven to outperform the previous state-of-the-art models, they present an immense augment in complexity in comparison to the methods seen under the Cooperative Games approach, and also have an important caveat: The Reinforcement learning methods find better results for longer training processes due to their online nature. This is a problem, since training for a large amount of epochs can take a substantial amount of time. For shorter trainings, a method that can yield good results with a low amount of training epochs and does not have an elevated computational cost might be preferred, like the subset selection methods.

3.2 Subset Selection Methods for Data Efficiency

While Data Valuation methods for Data Efficiency focus on selecting subsets based on the value obtained through the valuation method, subset selection methods focus on obtaining a subset verifying some desired similarity properties (usually involves solving a sub-modular optimization problem with a greedy algorithm). Methods like CRAIG and GRAD-MATCH fit this definition.

Coresets for Accelerating Incremental Gradient Descent [22], or CRAIG for short, is a method focused on improving the efficiency of Incremental Gradient Methods by carefully selecting a data set $\mathcal{X} \subset D_T$, so that the model is trained only on the subset $\mathcal{X}$ while still converging to the globally optimal solution.[3] They achieved this by selecting the smallest weighted subset $\mathcal{X}$ that best approximates the full gradient of D_T, with an error at most $\varepsilon > 0$. In short, they want to solve the following problem:

$$\begin{aligned}&\min_{\mathcal{X} \subseteq D_T,\ \gamma_j \geq 0\ \forall j} \quad |\mathcal{X}| \\ &s.t \quad \max_{\theta \in \Theta} \| \sum_{x_i \in D_T} \nabla L(f_\theta(x_i), y_i) - \sum_{x_j \in \mathcal{X}} \gamma_j \nabla L(f_\theta(x_j), y_j) \| < \varepsilon,\end{aligned} \tag{3.1}$$

but since solving the above problem would lead to a computationally unfeasible problem, they solve a

[2] This convex problem is the linear convex relaxation of the facility location problem using the distance between all the training points.

[3] Keep in mind that finding such $\mathcal{X}$ directly leads to a speed up of $|D_T|/|\mathcal{X}|$ on each epoch of the training process.

relaxation utilizing an upper bound of the previous restraints:

$$\begin{aligned} &\min_{\mathcal{X}\subseteq D_T} \quad |\mathcal{X}| \\ &s.t \quad \sum_{x_i\in D_T} \min_{x_j\in\mathcal{X}} \max_{\theta\in\Theta} \|\nabla L(f_\theta(x_i), y_i) - \nabla L(f_\theta(x_j), y_j)\| < \varepsilon. \end{aligned} \tag{3.2}$$

Later on, they prove that problem 3.2 can be written as a sub-modular set cover problem, that can be efficiently approximated using a greedy algorithm.

Recently we saw a growing number of approaches (Mirzasoleiman et al. [22], Killamsetty et al. [23]) on analysing the convergence bounds of adaptive data subset selection algorithms, where the algorithm (which selects the data subset depending on specific criteria) is applied in conjunction with the model training. Killamsetty et al. [25] worked with this boundings to create *a Gradient-Matching algorithm* (or GRAD-MATCH), that not only has boundings guarantees for the loss function, but have a convergence guarantee for the method to obtain such subset.

Since our own work can directly be extended using GRAD-MATCH results for adaptive data subset selection algorithms, we discuss such boundings and results.

3.2.1 Theoretical boundings for adaptive data subset selection methods

Normal data selection techniques focuses on obtaining a single subset that approximates the training with the full data set (like it is done in CRAIG, and we are going to do in Chapter 4 with GEDA). Now, in the adaptive data selection approach, we are going to perform the data selection in conjunction with the training. This refinement of the training subset allows the data selection to adapt to the learning and produces increasingly effective subsets with the progress of the learning algorithm. In this setting, we are going to have $X^t \subset D_T$, $\forall t = 1, \ldots, T_e$, where T_e is the number of epochs.

Before presenting the result from which GRAD-MATCH subsets are obtained, we need to define the term:

$$Err(\mathbf{w}^t, \mathcal{X}^t, L_T, L, D, \theta_t) = \| \sum_{(x_i,y_i)\in\mathcal{X}^t} w_i^t \nabla L_T(f_{\theta_t}(x_i), y_i) - \nabla \mathcal{L}_{\theta_t}(L, D)\|_2,$$

where $\mathcal{L}(L, D)$ is define like we did in section 2.2, D is a set of points not necessarily related to the training set D_T, $\mathcal{L}_{\theta_t}(L, D) = \sum_{(x,y)\in D} L(f_{\theta_t}(x), y)$ is the loss function of the model over D at epoch t, and L_T is the loss function for the each point of the training set (not necessarily the same as L). Under this assumptions lays the following theorem:

Theorem 3.1 *(Killamsetty et al. [25])*[4] *Lets assume that the parameters satisfy that $\|\theta_t\| \leq C$. Under this assumptions, any adaptive data selection algorithm runned with full gradient descent (GD), defined via weights $\mathbf{w}^t$ and subsets $\mathcal{X}^t$ for $t = 1, \ldots, T_e$, enjoys the following guarantees:*

1. *If L_T is Lipschitz continuous with parameter K_1, optimal model parameters θ^*, and $\alpha = \frac{C}{K_1\sqrt{T}}$ learning rate, then:*

$$\min_t \mathcal{L}_{\theta_t}(L, D) - \mathcal{L}_{\theta^*}(L, D) \leq \frac{CK_1}{\sqrt{T}} + \frac{C}{T}\sum_{t=1}^{T_e-1} Err(\mathbf{w}^t, \mathcal{X}^t, L_T, L, D, \theta_t).$$

2. *Let L_T be Lipschitz smooth with parameter K_2, θ^* the optimal model parameters ,and $0_T(f_\theta(x), y) \leq \beta$, $\forall (x, y) \in D_T$. Then, defining a learning rate $\alpha = \frac{1}{K_2}$:*

$$\min_t \mathcal{L}_{\theta_t}(L, D) - \mathcal{L}_{\theta^*}(L, D) \leq \frac{C^2K_2 + 2\beta}{2T} + \frac{C}{T}\sum_{t=1}^{T_e-1} Err(\mathbf{w}^t, \mathcal{X}^t, L_T, L, D, \theta_t).$$

[4]The proof can be found in Appendix B.1 of [25].

3. *If L_T is Lipschitz continuous with parameter K_1, optimal model parameters θ^* and L is strongly convex with parameter μ, then, setting a learning rate $\alpha_t = \frac{2}{\mu(1+t)}$ for each epoch $t = 1, \ldots, T_e$; the following inequality holds:*

$$\min_t \mathcal{L}_{\theta_t}(L, D) - \mathcal{L}_{\theta^*}(L, D) \leq \frac{2K_1^2}{\mu(T+1)} + \sum_{t=1}^{T} \frac{2tC}{T(T+1)} Err(\mathbf{w}^t, \mathcal{X}^t, L_T, L, D, \theta_t).$$

Note that, if $D = D_T$ and $L = L_T$, we are bounding the convergence error of the training process.

Observation 3.2 *Theorem 3.1 can be extended to any adaptive selection algorithm runned with SGD just by adding expectations on both sides of the inequalities. Proof of this can be found in Appendix B.2 of [25].*

The results of Theorem 3.1 gives us a direct way to find adaptive subsets $\mathcal{X}^t$, for each epoch t, that minimize the bounding of the convergence error to the optimal parameters of a loss function L over a subset D, just by selecting:

$$\mathbf{w}^t, \mathcal{X}^t = \underset{\mathbf{w}^t,\ \mathcal{X}^t:\ |\mathcal{X}| \leq k}{\arg\min} Err(\mathbf{w}^t, \mathcal{X}^t, L_T, L, D, \theta_t). \tag{3.3}$$

The optimization problem 3.3 might seem complicated, but the authors proved on the same paper that problem 3.3 can be rewritten as a weakly sub-modular set cover problem, which finds convergence guarantees using an algorithm called Orthogonal Matching Pursuit (proved in Elenberg et al. in [24]). We are not going to further elaborate on this, since this is not of relevance for our work.

Observation 3.3 *While having a general L and D might seem unnecessary and confusing, having generality over those two term allows us to bound the convergence error of the training using, for example, the validation loss L_V over the validation set D_V, while only training with the training set. This is an important application of data efficiency (usually reserved to data valuation methods for data efficiency) called the domain adaptation problem, where we want to select the points that better resemble the validation set (or, in general, any other data set following a different distribution from that of the training set). This is common when we want to train with expensive data sets without enough data (like the skin cancer example in [16]).*

Chapter 4

Introducing GEDA

Current subset selection methods present a problem, and it is that even for non-adaptive methods like CRAIG, they all focus on obtaining better results during long training processes (see the experiment section of [22], [23]). For this reason, we identify the need of a model that focuses on obtaining better results for short training processes, and introduced our own method of subset selection, one that not only performs better for shorter training processes, but also just needs to optimize a MILP problem at the beginning of the training process.

In this chapter we introduce the theoretical concepts of our proposed method: GEDA. In the next Chapter we contextualize the method by showing how the experiments that we performed led to it.

4.1 GEDA: Gradient norms Empirical Distribution Approximation

Let G_{D_T} be the set of gradient norms , where[1]:

$$g_i \in G_{D_T} :\Longleftrightarrow g_i = \|\nabla L(f_{\theta_0}(x_i), y_i)\|_2,\ \forall (x_i, y_i) \in D_T,$$

and $\mathcal{X} \subset D_T :\ |\mathcal{X}| = k$ a subset. Our proposed method, GEDA, simply finds the subset $\mathcal{X}$ such that $G_{\mathcal{X}} \subset G_{D_T}$ is the subset of norms of gradients that better approximates some estimation of the distribution function of G_{D_T}. Therefore, we want $\mathcal{X} \subset D_T$ such as:

$$\mathcal{X} = \underset{\mathcal{X} \subset D_T:\ |\mathcal{X}|=k}{\arg\min} \int_{-\infty}^{\infty} |\hat{F}_{D_T}(t) - \hat{F}_{\mathcal{X}}(t)|\ dt.$$

While this might seems like a hard problem to solve, we can reduce it to a simpler mixed integer linear programming problem (MILP)[2]. For this, we are going to take $\hat{F}_S$ as the ECDF of S:

$$\hat{F}_S(x) = F_{nS}(x) = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbb{I}\{X_i \leq x\}.$$

For the integral, we are going to use the composite trapezoidal rule[3]:

$$\int_A^B f(t)dt = \frac{B-A}{N} \left(\frac{f(A) + f(B)}{2} + \sum_{k=1}^{N-1} f\left(A + k\frac{B-A}{N}\right) \right),$$

[1] We are considering the model f at epoch 0.

[2] This is important to us, since we only have access to a MILP solver.

[3] As it can be deduced by the expression given, we are applying the composite trapezoidal rule to a partition that is uniformly spaced.

where N is the desired number of evaluation points for the rule, that have been set by the experimenter. For our case, since $m = \inf(D_T)$ and $M = \sup(D_T)$ exist because D_T is finite, and considering that $\mathcal{X} \subset D_T$, employing the definition of the ECDF we have that:

$$\forall m' \in \mathbb{R} : \; m' \leq m \Longrightarrow F_{n\,D_T}(m') = F_{n\,\mathcal{X}}(m') = 0,$$
$$\forall M' \in \mathbb{R} : \; M' \geq M \Longrightarrow F_{n\,D_T}(M') = F_{n\,\mathcal{X}}(M') = 1.$$

Therefore,

$$\int_{-\infty}^{m} |F_{n\,D_T}(t) - F_{n\,\mathcal{X}}(t)| \; dt = 0 = \int_{M}^{\infty} |F_{n\,D_T}(t) - F_{n\,\mathcal{X}}(t)| \; dt,$$

which means that:

$$\int_{-\infty}^{\infty} |F_{n\,D_T}(t) - F_{n\,\mathcal{X}}(t)| \; dt = \int_{m}^{M} |F_{n\,D_T}(t) - F_{n\,\mathcal{X}}(t)| \; dt.$$

So, we are going to consider $f(t) = |F_{n\,D_T}(t) - F_{n\,\mathcal{X}}(t)|$, $A = m$ and $B = M$ in the definition of the composite trapezoidal rule, to obtain:

$$\begin{aligned}
\int_A^B f(t)dt &= \frac{B-A}{N}\left(\frac{f(A)+f(B)}{2} + \sum_{k=1}^{N-1} f\left(A + k\frac{B-A}{N}\right)\right) \\
&= \frac{M-m}{N}\sum_{k=1}^{N-1}\left|F_{n\,D_T}\left(m + k\frac{M-m}{N}\right) - F_{n\,\mathcal{X}}\left(m + k\frac{M-m}{N}\right)\right| \\
&= \frac{M-m}{N}\sum_{k=1}^{N-1}\left|\frac{1}{|D_T|}\sum_{i=1}^{|D_T|}\mathbb{I}\left\{X_i \leq m + k\frac{M-m}{N}\right\} - \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|D_T|}\delta_i\mathbb{I}\left\{X_i \leq m + k\frac{M-m}{N}\right\}\right|,
\end{aligned}$$

where $\delta_i = 1$ if $X_i \in \mathcal{X}$, and is 0 otherwise.

Furthermore, since N is defined by the experimenter and the points $X_i \in D_T$ are data, every $\mathbb{I}\left\{X_i \leq m + k\frac{M-m}{N}\right\}$ is going to be a datum as well, thus, the only variables involved are the $\delta_i \in \{0, 1\}$. Due to this, we can write $c_{ik} = \mathbb{I}\left\{X_i \leq m + k\frac{M-m}{N}\right\}$, and so, our problem can be rewritten as:

$$\begin{aligned}
\min \quad & \frac{M-m}{N}\sum_{k=1}^{N-1}\left|\frac{1}{|D_T|}\sum_{i=1}^{|D_T|} c_{ik} - \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|D_T|}\delta_i c_{ik}\right| \\
s.t \quad & \delta_i \in \{0, 1\}.
\end{aligned}$$

Now, because the absolute value is not differentiable, we need to reformulate the problem to get rid of the non-differentiability. Firstly, let's consider that:

$$\left|\frac{1}{|D_T|}\sum_{i=1}^{|D_T|} c_{ik} - \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|D_T|}\delta_i c_{ik}\right| = \max\left\{\frac{1}{|D_T|}\sum_{i=1}^{|D_T|} c_{ik} - \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|D_T|}\delta_i c_{ik}, \; -\frac{1}{|D_T|}\sum_{i=1}^{|D_T|} c_{ik} + \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|D_T|}\delta_i c_{ik}\right\},$$

and so, our problem can be rewritten as a min-max problem, which we know that can be reformulated adding a variable $Z_k = \left|\frac{1}{|D_T|}\sum_{i=1}^{|D_T|} c_{ik} - \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|D_T|}\delta_i c_{ik}\right|$ for each absolute value, and a pair of restrictions that ensures that the previous equality holds. Doing this, we achieve a reformulation of

our original problem into a MILP problem:

$$
\begin{aligned}
\min \quad & \sum_{k=1}^{N-1} Z_k \\
s.t \quad & \frac{1}{|D_T|}\sum_{i=1}^{|D_T|} c_{ik} - \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|D_T|} \delta_i c_{ik} \leq Z_k;\ \forall k, \\
& -\frac{1}{|D_T|}\sum_{i=1}^{|D_T|} c_{ik} + \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|D_T|} \delta_i c_{ik} \leq Z_k;\ \forall k, \\
& \sum_{i=1}^{|D_T|} \delta_i = \kappa, \\
& \delta_i \in \{1, 0\}, \\
& Z_k \in \mathbb{R},\ \forall k.
\end{aligned}
\tag{P}
$$

This way, when solving problem (P), the resulting values of the variables δ_i are going to give us the points that conforms our desired subset $\mathcal{X}$.

Chapter 5

Experiments

Everything that we are going to do, has to be understood within the context of our employer and the project which we are working on. Gradiant Technologies is a technological center located in Vigo, Spain, and specialises on security, AI and connectivity. Inside their AI department, they focus on giving solution to real life problems using AI models, usually involving training neural networks. Here, to avoid repetitive code and as a *quality of life* measure, they deployed a series of templates to heighten the level of the code and make it more accessible and ready to use. This ecosystem of templates for AI (called Eco-AI) currently supports the Computer Vision tasks of classification, localization and segmentation (see figure 2.1), and they were looking for ways to improve the tools provided by Eco-AI. We focused our efforts into researching Data Efficiency methods that could both improve upon the current state of the art and work inside Eco-AI, specifically for low epoch training processes.

In this section we present how all the experiments that were conducted during our research led us towards the introduction of our data efficiency model: GEDA, contextualizing how the research process evolved from a general problem (we want to reduce our data sets without a big compromise in accuracy when training with a low amount of epochs) without an available solution, to a proposed model for solving said problem.

5.1 Experimental Set up

5.1.1 Data sets

We used CIFAR-10 and CIFAR-100 [26] for our data sets. Both CIFAR-10 and CIFAR-100 data sets consists of 60000 32x32 colour images in 10 and 100 classes, with 6000 and 600 images per class (respectively). There are 50000 training images and 10000 validation (or test) images on both data sets (see figure 5.1)

5.1.2 Evaluation Metric

The evaluation metric for our model was the top 1 accuracy, $\mathbb{A}_1$:

$$\mathbb{A}_1(f_\theta) = \frac{\text{Number of correct predictions of } f_\theta \text{ over } D_V}{|D_V|},$$

where f_θ is the trained model and D_V is the validation set.

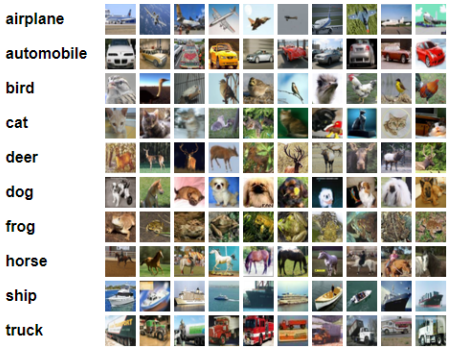


Figure 5.1: Example of images for each category of CIFAR-10. Source: `https://www.cs.toronto.edu/~kriz/cifar.html`

5.1.3 Implementation Details

We wrote all of our code in Python, employing a multitude of libraries and framework in the process. For the training of the models, we employed the library *mmcls* [27] over the *PyTorch* [28] framework, parallelizing all of the calculations using CUDA, over a *NVidia RTX 3090* GPU. Here, we trained a ResNet-18 for all of our training, always trying to achieve the best results possible.

All of the results presented in this section have been obtained by selecting the best hyper-parameters that we could find for the training, thus yielding the best results on our metrics, while always trying to keep the training under an hour long.

For the learning rate during the training process, we are going to use a cyclical learning rate scheduler [29], were the learning rate is going to increase from an initial value up to a maximum set value, and then decrease to a target value.

For GEDA, we employed the optimization library *PulP* [30], were we used their available MILP solver employing a Branch & Cut algorithm, and fixed the number of nodes in the trapezoidal rule to $N = 5$ due to empirical results.

5.2 Baselines and first approach

Results for the baselines can be seen in table 5.1. Here, we trained a ResNet-18 over the full training set for both specified data sets with the disclosed hyper-parameters, and the default weight decay value set by *PyTorch* (0.0001).

As a first approach, let us try to eliminate data randomly inside each category until we have reached our desired $\kappa\%$ of data on each category[1]. This is going to work as well as a good comparison for all the other methods that we are going to present afterwards. We have plotted the results obtained during our experiments for better visualization and left the full information about the hyper-parameters of all the experiments at the Appendix A.

Looking at Figure 5.2, we can see how our results are far from what we got at the baselines with the complete data set (both with and without pretraining, for both CIFAR-10/100), so we are in need for a better way to remove data from our data sets.

[1]We are removing the data inside each category to maintain the same proportion of data inside each category. Every other experiment presented here is going to be performed inside each categories due to the same reason.

Data Sets	Pretrained	Initial Learning Rate	Target Learning Rate	Batch Size	Epochs	Loss	Accuracy Top-1
	Yes	1e-04	1e-06	16	8	0,0013	95,95
CIFAR-10	No	1e-04	1e-06	16	12	0,0055	86,23
	Yes	1e-04	1e-06	16	8	0,0110	81,11
CIFAR-100	No	0,01	1e-04	16	12	0,0120	66,08

Table 5.1: Baseline results employing a ResNet-18 model over the full training set and never exceeding 1 hour of training. Weight decay has been set to 0.0001, the maximum learning rate has been set to 100 times the initial value, and the pretraining was conducted over the ImageNet data set [3].

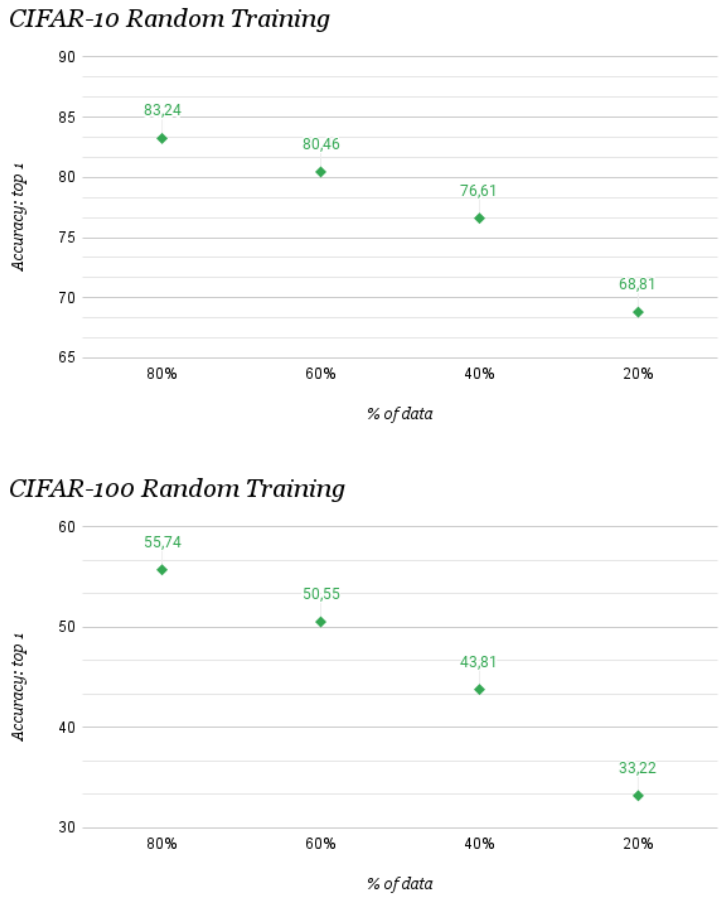


Figure 5.2: Results of training a ResNet-18 model over $80 - 20\%$ of the data of both CIFAR-10's and CIFAR-100's training data set. We removed the data randomly.

5.3 Working with the gradients

Inspired by what we have seen in section 3.2 about GRAD-MATCH, we are going to work with the gradients of the loss function over each data class. Furthermore, we are going to utilize the norm of

each gradient as reference. Let us consider the set of gradient norms G_{D_T} defined as in the previous chapter. The goal would be to find a subset $\mathcal{X}$ utilizing G_{D_T} in some way, shape or form.

Just as a start, let us do the same random experiment, but this time let us take data out data randomly between the quartiles of G_{D_T}. This way, we are going to randomly remove $(1-\kappa)\%$ of the data between the quartiles, so we would reach the desired percentage of total data points $\kappa\%$ in each category, although this time with less randomness involved overall. You can check the results of training with this method in CIFAR-10 in figure 5.3, where we see a minor improvement over the completely randomized method of extraction.

While better, randomly selecting which data to remove, even if we do it with more information, does not seem like a great way to obtain $\mathcal{X}$. It would be better to have some method to obtain such subsets without any randomness involve.

Another approach to the problem, that does not involve randomness, and stills employs G_{D_T}, could be removing data points over the data's ranking in G_{D_T}. In this case, we are going to remove data from D_T in such a way that every removed data point is evenly spaced in said rankings. As an example[2], if we wanted to remove the 50% of the total data inside D_T, once ordered by G_{D_T}, we would need to remove 1 of every 2.

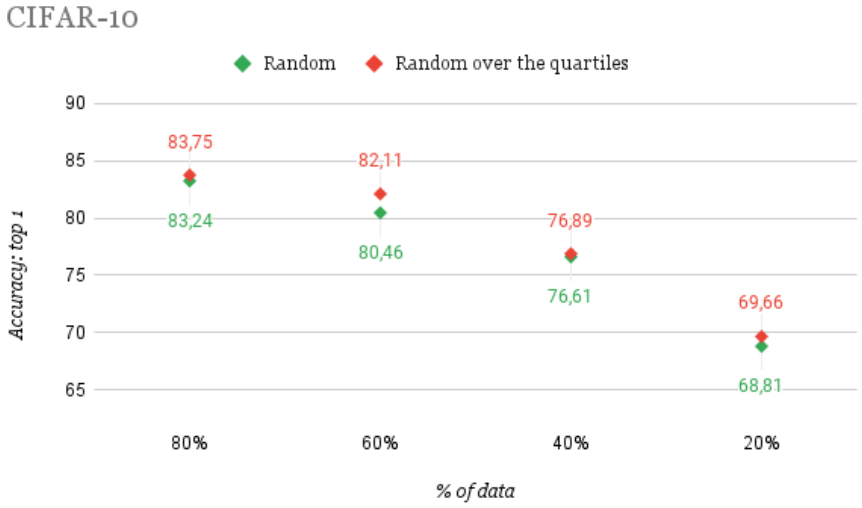


Figure 5.3: Results of training a fine-tuned ResNet-18 model over $80-20\%$ of CIFAR-10's training data set. We removed the data randomly between each quartile and at random inside each category.

We can see the results of this strategy for removing data points when training on CIFAR-10 in figure 5.4, where we see a noticeable improvement over the results obtained when we removed the data randomly between the quartiles.

[2]Generally speaking, given the ordered points of G_{D_T}, $\{g_{(i)}\}_i$, we are going to remove the point $(x_{(i)}, y_{(i)})$ if, and only if, $i \equiv 0 \mod \frac{|D_T|}{|\mathcal{X}|}$

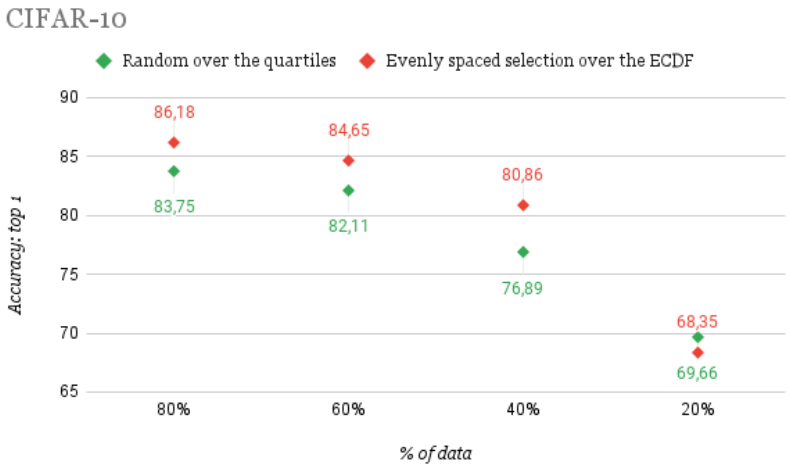


Figure 5.4: Results of training a fine-tuned ResNet-18 model over $80-20\%$ of the data of CIFAR-10's training data set. We removed the data evenly over the ranking given by G_{D_T}.

5.4 Introducing GEDA

Seen how much of an improvement over randomly taking out gradients we achieved once we started to work with the ordered data, it might be a good idea to work whit the distribution of G_{D_T}. Enter GEDA, the method that we are proposing as an alternative to CRAIG and GRAD-MATCH inside the subset selection methods framework. Now, we are going to select the subset $\mathcal{X} \subset D_T : |\mathcal{X}| = k$ such as $G_{\mathcal{X}} \subset G_{D_T}$ is the subset that better approximates some estimation of the distribution function of G_{D_T}.

The results of training a ResNet-18 on CIFAR-10, reducing the data set size utilizing this strategy can be seen in figure 5.5, compared to the results that we achieved with the best method of extraction as of yet. As we can see, the method presents an improvement on almost all of the scenarios, specially on low κ-percentages of data and on CIFAR-100, a tougher data set overall[3].

5.5 Comparisons with the S.O.T.A

In practice, adaptive methods achieve better results when they are computed every set number of epochs R, instead of every epoch. This is important, since having good empirical results with a high R can lead to a considerable speed up, since it reduces how many times it has to be performed. Since *Gradiant*, for the moment, can only implement offline methods, it is in our best interest to test how powerful all the methods are when $R = 0$, i.e: when we calculate the subset only at the beginning of the training process, in comparison to the other S.O.T.A methods.

At figure 5.6 we can see a comparison between GEDA and both GRAD-MATCH and CRAIG. As we can see, GEDA outperforms both CRAIG and GRAD-MATCH when $R = 0$ on a healthy dataset like CIFAR-10, and we get even better results using GEDA on a harder data set like CIFAR-100. While this is good news for us, it would be disingenuous to not clarify the actual use case of an adaptive method like GRAD-MATCH. The actual value of GRAD-MATCH comes with the ability to achieve accuracy results similar to those of the baselines, while having only an small percentage of data, during

[3]As it can be seen in the results on all of the experiments with CIFAR-100, we obtain lower accuracy when training a ResNet-18 on it, in comparison with CIFAR-10. The explanation is simple, we have the same amount of data overall, but have 10 times more categories on CIFAR-100 compared to CIFAR-10.

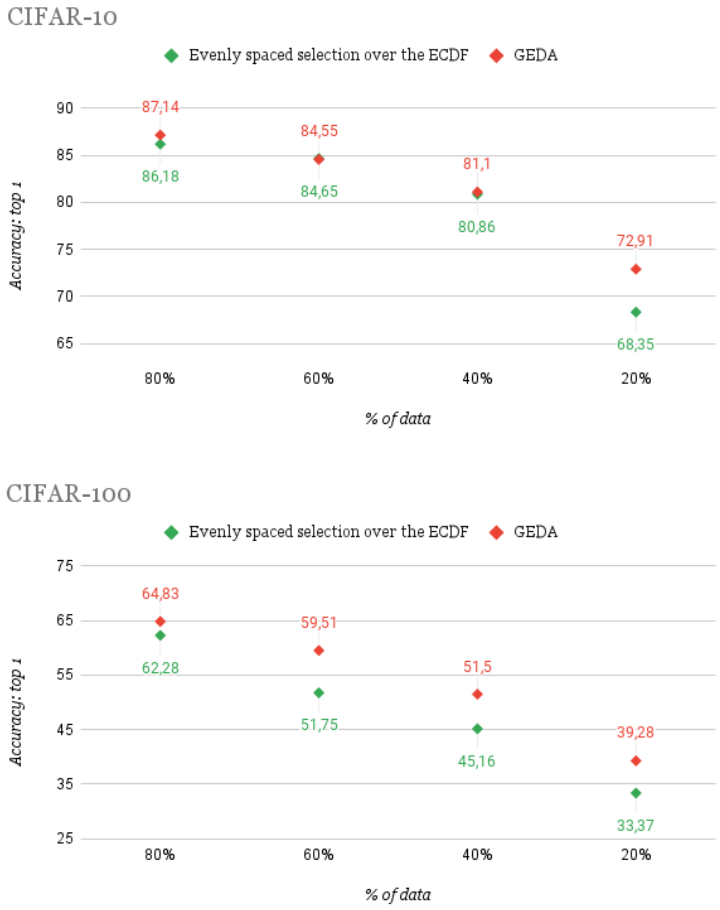


Figure 5.5: Results of training a fine-tuned ResNet-18 model over $80 - 20\%$ of the data of CIFAR-10/100's training data set. We removed the data evenly over the ranking given by G_{D_T} and using GEDA.

longer executions.

As an example of this, we presented a result obtained while training a ResNet-18 over CIFAR-10 for 100 epochs with only 20% of the training data, employing both GRAD-MATCH with $R = 20$ and GEDA, in figure 5.7. Not only GRAD-MATCH beats GEDA in this experiment, but the accuracy obtained this way rivals that of the baselines, and even the one obtained with 80% of the data using GEDA. Furthermore, Killamsetty et al. [25] included a detailed experiment results section in the appendix of their work, where they achieved an accuracy of 90.90 with only 10% of data, and 92.15 with 20%. This results not only beats every data efficiency method that we have tested, but also rivals the results obtained with a pretrained ResNet on our baselines.

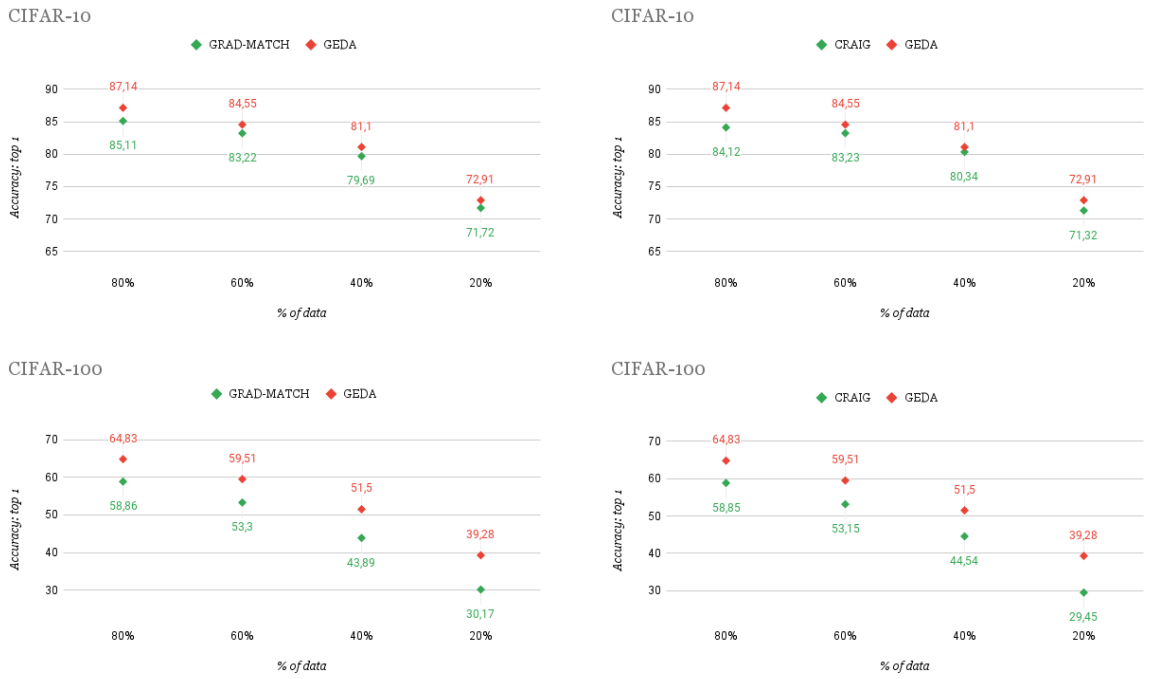


Figure 5.6: Results of training a fine-tuned ResNet-18 model over $80-20\%$ of the data of CIFAR-10's and CIFAR-100's training data sets. We removed using GEDA and both GRAD-MATCH and CRAIG.

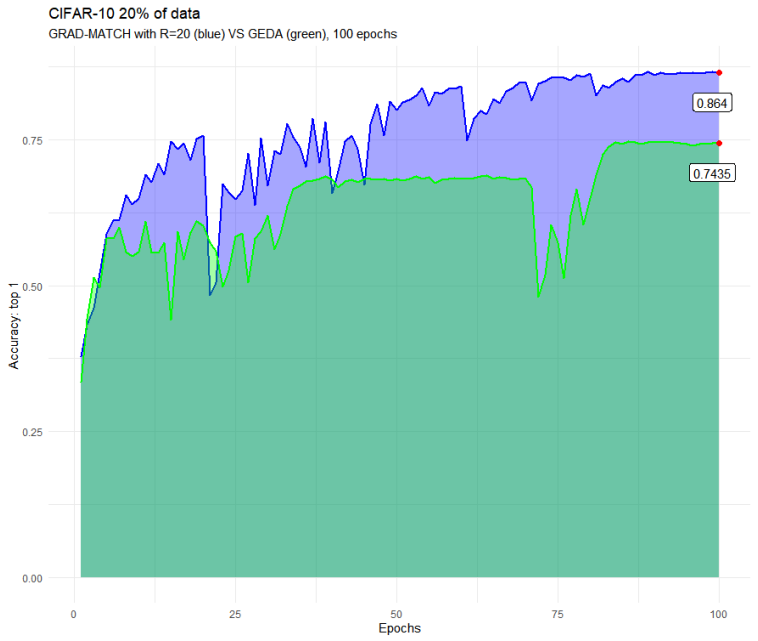


Figure 5.7: Evolution of the accuracy during the training of a fine-tuned ResNet-18 model over 20% of CIFAR-10's training data set, during 100 epochs. We removed data using GRAD-MATCH with $R = 20$ (blue) and GEDA (green).

5.6 Final thoughts and conclusions

GEDA showed better results than both GRAD-MATCH and CRAIG as an offline algorithm, but it could not keep up with GRAD-MATCH when we set $R > 0$. Although it might seem pointless to introduce GEDA if we compare it with the latest results of GRAD-MATCH, it is important to clarify that the time taken by these experiments vastly exceeds what it took us to obtain the results with GEDA, and even the baselines (see table 5.2).

With this, we can see a difference on the actual value that each method brings to the table. When time is not an issue, but we want to achieve the best results possible employing the least amount of data possible, GRAD-MATCH seems to be the best option inside of the method we have compared[4], while for speed, an offline algorithm like GEDA seems to be the best for the data sets that we used.

Since our objective was always to obtain a method capable to be efficient in time for low epoch training, GEDA, an offline algorithm, fits every desired quality that our employer asked for. Although we have reached to a conclusion, it is important to note that a lot could be done to improve upon the method outside our contractor limitations, like adapting GEDA to be an online method or even employing a smoothing of the empirical distribution function. All of this is going to be discussed in our next, and final section of this work.

Method	Accuracy: top 1 (%)	Time (minutes)
GEDA: 80%	87,14	6,98
GRAD-MATCH: 20%	86,4	29,988
GRAD-MATCH: 10%	90,9	53,4

Table 5.2: Time analysis of the trainings of a ResNet-18 with CIFAR-10, when removing data employing GRAD-MATCH ($R = 20$) and GEDA. Results for the GRAD-MATCH experiment with 10% where obtained from [25].

[4] A lengthier comparison between every relevant adaptive subset selection method can be found at appendix C.3 of Killamsetty et al. (2021), where they reached to our same conclusion.

Chapter 6

Future Work

We presented GEDA, a data efficiency method based on selecting the subset $\mathcal{X}$ that better resemble the ECDF of the full training set D_T. In this last chapter we are presenting a multitude of different ideas and extensions of GEDA that can be interesting to explore in the future.

6.1 Improving GEDA

We detect different ways to improve the method itself. A good first approach might come in utilizing a smoothing of the ECDF. This could improve the precision of the approximation performed in GEDA, and still would lead up to a MILP problem. Another thing that could be done is to experiment with different numerical integration methods with better error guarantees (like the composite Simpon's Rule), and compare their performance in both time and accuracy improvement. Finally, a study of the optimization model proposed for GEDA it is still to be made. As a MILP problem, we could study specific cuts for the Branch & Cut algorithm that could lead to a tighter formulation of our problem.

6.2 Online GEDA

The most obvious extension to GEDA is to extend it as an adaptive subset selection method, using the same strategy that GRAD-MATCH used with the boundings in section 3.2.1. This way, we can use the following algorithm:

Algorithm 6.1 *Employing the notations of section 3.2.1, we can define GEDA as an online algorithm as follows:*

1. *Calculate the subset $\mathcal{X}^t$ using GEDA*
2. *Calculate the weights $\mathbf{w}^t$ that:*

$$\mathbf{w}^t = \underset{\mathbf{w}\in\mathbb{R}^{|\mathcal{X}|}}{\arg\min}\ Err(\mathbf{w}, \mathcal{X}^t, L_T, L, D, \theta_t) + \lambda\|\mathbf{w}^t\|,$$

   with $\lambda \in \mathbb{R}$, and $L_T,\ L,\ D$ and θ_t as defined originally in section 3.2.1.

3. *Repeat each $t \equiv 0 \mod R$, otherwise $(\mathcal{X}^t, \mathbf{w}^t) = (\mathcal{X}^{t-1}, \mathbf{w}^{t-1})$.*

We end up having an unconstrained convex problem, that can be easily solved using, for example, Gradient Descent.

While easier than GRAD-MATCH's approach, we have to keep in mind that the pair of subset and weights obtained at each epoch are going to have, necessarily, a higher Err than that of GRAD-MATCH. That is, using the same model f_{θ_t} at epoch t during training, the pair of subset and weights $(\mathcal{X}_G^t, \mathbf{w}_G^t)$ obtained using GEDA, and another pair $(\mathcal{X}_{GM}^t, \mathbf{w}_{GM}^t)$ obtained using GRAD-MATCH, are going to fulfill the following inequality:

$$Err(\mathbf{w}_G^t, \mathcal{X}_G^t, L_T, L, D, \theta_t) \geq Err(\mathbf{w}_{GM}^t, \mathcal{X}_{GM}^t, L_T, L, D, \theta_t),$$

with $L_T,\ L,\ D$ and θ_t as defined originally in section 3.2.1, for all t congruent with $0 \mod R$.

6.3 GEDA as a warm start method

Seeing how GEDA have worse guarantees than GRAD-MATCH as an online method, this begs the question: could there be another way to use GEDA inside an adaptive subset selection method with good guarantees?

Killamsetty et al. performed in [25] a lengthy study on all state-of-the-art subset selection methods at the time, where they studied both the accuracy and the time taken to perform the training. They found out that the best results were obtained when a warm start was performed at the beginning of the training process. We define a warm start inside an adaptive subset selection method as taking the full training set D_T as the first subset $\mathcal{X}^1$.

Now, because $GEDA$'s subset can be obtained solving a rather simple MILP problem and how well they perform when $R = 0$, we could think of using the GEDA subsets as a warm start for an adaptive subset selection method. This would lead to an overall speed up of $\frac{|D_T|}{|\mathcal{X}|}$ during the first R epochs when using the GEDA subsets, instead of the full training set. This might be interesting when training with the full dataset might be computationally expensive, specially when utilizing large R.

Appendix A

Tables and Results

The goal of this appendix is to summarize all the information regarding the hyper-parameters employed during all training processes.

Database	Method	Percentage of data (%)	Initial Learning Rate	Target Learning Rate	Batch Size	Epochs	Accuracy: top 1
CIFAR-10	GRAD-MATCH	80	10^{-3}	10^{-5}	20	12	85, 11
		60	10^{-3}	10^{-5}	20	12	83, 22
		40	10^{-2}	10^{-4}	20	12	79, 69
		20	10^{-2}	10^{-4}	20	12	71, 72
	CRAIG	80	10^{-3}	10^{-5}	20	12	84, 12
		60	10^{-3}	10^{-5}	20	12	83, 23
		40	10^{-3}	10^{-5}	20	12	80, 34
		20	10^{-3}	10^{-5}	20	12	71, 32
CIFAR-100	GRAD-MATCH	80	10^{-3}	10^{-5}	32	12	58, 86
		60	10^{-3}	10^{-4}	32	12	53.3
		40	10^{-3}	10^{-5}	32	12	43, 89
		20	10^{-3}	10^{-5}	32	12	30, 17
	CRAIG	80	10^{-3}	10^{-5}	32	12	58, 85
		60	10^{-3}	10^{-5}	32	12	53.17
		40	10^{-3}	10^{-5}	32	12	44, 54
		20	10^{-3}	10^{-5}	32	12	29, 45

Table A.1: Results table of all the experiment with both GRAD-MATCH and CRAIG. Every experiment was conducted with a Kaggle notebook employing a single NVidia K80 GPU.

Database	Method	Percentage of data (%)	Initial Learning Rate	Target Learning Rate	Batch Size	Epochs	Loss	Accuracy: top 1
CIFAR-10	Random	80	10^{-3}	10^{-5}	64	12	10^{-3}	83.24
		60	10^{-3}	10^{-4}	64	12	$2\ 10^{-3}$	80.46
		40	10^{-3}	10^{-4}	64	12	$3\ 10^{-3}$	76.61
		20	10^{-3}	10^{-4}	64	12	$4\ 10^{-3}$	68.81
	Random over the quantiles	80	10^{-3}	10^{-5}	64	12	$2\ 10^{-3}$	83.75
		60	10^{-3}	10^{-5}	64	12	$2\ 10^{-3}$	82.11
		40	10^{-3}	10^{-4}	64	12	$2\ 10^{-3}$	76.89
		20	10^{-3}	10^{-4}	64	12	$3\ 10^{-3}$	69.66
	Evenly spaced selection	80	10^{-3}	10^{-5}	64	12	$4\ 10^{-3}$	86.18
		60	10^{-3}	10^{-5}	64	12	10^{-3}	84.65
		40	10^{-3}	10^{-5}	64	12	$6\ 10^{-3}$	80.86
		20	10^{-3}	10^{-4}	64	12	$3\ 10^{-3}$	68.35
	GEDA	80	10^{-3}	10^{-5}	64	12	$6\ 10^{-3}$	87.14
		60	10^{-3}	10^{-4}	64	12	$6\ 10^{-3}$	84.55
		40	10^{-3}	10^{-4}	64	12	$7\ 10^{-3}$	81.1
		20	10^{-3}	10^{-5}	64	12	$1.1\ 10^{-2}$	72.91
CIFAR-100	Random	80	10^{-3}	10^{-4}	64	12	$5\ 10^{-3}$	55.74
		60	10^{-3}	10^{-4}	64	12	$2.30\ 10^{-2}$	50.55
		40	10^{-3}	10^{-4}	64	12	$2.80\ 10^{-2}$	43.81
		20	10^{-3}	10^{-4}	64	12	$9.70\ 10^{-2}$	33.22
	Random over the quantiles	80	10^{-3}	10^{-4}	32	12	$6\ 10^{-3}$	54.64
		60	10^{-3}	10^{-4}	32	12	$2.20\ 10^{-2}$	50.84
		40	10^{-3}	10^{-4}	32	12	$2.80\ 10^{-2}$	44.08
		20	10^{-3}	10^{-4}	32	12	$7.30\ 10^{-2}$	34.36
	Evenly spaced selection	80	10^{-3}	10^{-4}	32	12	$4.40\ 10^{-2}$	62.28
		60	10^{-3}	10^{-4}	32	12	$2.20\ 10^{-2}$	51.75
		40	10^{-3}	10^{-4}	32	12	$3\ 10^{-2}$	45.16
		20	10^{-3}	10^{-4}	32	12	$1.04\ 10^{-1}$	33.37
	GEDA	80	10^{-3}	10^{-4}	64	12	$3.49\ 10^{-1}$	64.83
		60	10^{-3}	10^{-4}	64	12	$1.10\ 10^{-1}$	59.51
		40	10^{-3}	10^{-4}	64	12	$2.06\ 10^{-1}$	51.5
		20	10^{-3}	10^{-4}	64	12	$5.79\ 10^{-1}$	39.28

Table A.2: Results table of all our experiments with their hyperparameters. The experiments were conducted using parallel computing with a RTX 3090 GPU

Bibliography

[1] Rosenblatt, F. (1958). The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* (Vol. 65 - 6).

[2] P. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences*, Ph.D. dissertation, Committee on Appl. Math., Haivard Univ., Cambridge, MA, Nov. 1974.

[3] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. *IEEE Computer Vision and Pattern Recognition (CVPR).*

[4] Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems* (Vol. 25). Curran Associates, Inc. Cornell Aeronautical Laboratory.

[5] He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-77.

[6] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations.*

[7] Carreira, J. Zisserman, Andrew. (2017). Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In proceedings of *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4724-4733.

[8] Qiu, Y., Wang, X. (2020). Stochastic Approximate Gradient Descent via the Langevin Algorithm. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04), pp. 5428-5435.

[9] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks: The Official Journal of the International Neural Network Society*, 12 1, 145–151.

[10] Johnson, R., Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems,* pp. 315-323.

[11] Smith, L. (2018). A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820.*

[12] Simard, P., Steinkraus, D., Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. *Seventh International Conference on Document Analysis and Recognition.* Proceedings., 2003, pp. 958-963.

[13] Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd ed., Springer, New York.

[14] Strubell, E., Ganesh, A., McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

[15] Wang, T., Rausch, J., Zhang, C., Jia, R., Song, D. (2020) A Principled Approach to Data Valuation for Federated Learning. In: Yang, Q., Fan, L., Yu, H. (eds). *Federated Learning. Lecture Notes in Computer Science*, vol 12500. Springer, Cham.

[16] Ghorbani, A., Zou, J. (2019). Data Shapley: Equitable Valuation of Data for Machine Learning. *arXiv preprint arXiv:1904.02868v2*.

[17] Ghorbani, A., Kim, M. P., Zou, J. Y. (2020). A Distributional Framework for Data Valuation. *International Conference on Machine Learning (ICML)*.

[18] Yan, T., Procaccia, A. (2021). If You Like Shapley Then You'll Love the Core. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(6), pp. 5751-5759.

[19] Yoon, J., Arik, S., Pfister, T. (2020). Data Valuation using Reinforcement Learning. *Proceedings of the 37th International Conference on Machine Learning*, in *Proceedings of Machine Learning Research* 119:10842-10851.

[20] Williams, R.J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn 8*, 229–256.

[21] Das, S., Singh, A., Chatterjee, S., Bhattacharya, S. (2021). Finding High-Value Training Data Subset Through Differentiable Convex Programming. In: Oliver, N., Pérez-Cruz, F., Kramer, S., Read, J., Lozano, J. (eds) *Machine Learning and Knowledge Discovery in Databases. Research Track*, pp. 666–681. ECML PKDD 2021. Lecture Notes in Computer Science, vol 12976. Springer, Cham.

[22] Mirzasoleiman, B., Bilmes, J., Leskovec, J. (2020). Coresets for Data-efficient Training of Machine Learning Models. *International Conference on Machine Learning (ICML) 2020, July*.

[23] Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., Iyer, R. (2021a). Glister: Generalization based data subset selection for efficient and robust learning. *In AAAI, 2021*.

[24] Elenberg, E. R., Khanna, R., Dimakis, A. G., Negahban, S., et al. Restricted strong convexity implies weak submodularity. *The Annals of Statistics*, 46(6B):3539–3568, 2018.

[25] Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., De, A., Iyer, R. (2021b). GRAD-MATCH: Gradient Matching based Data Subset Selection for Efficient Deep Model Training. *International Conference on Machine Learning (ICML) 2021, February*.

[26] Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*.

[27] MMClassification Contributors. (2020). *OpenMMLab's Image Classification Toolbox and Benchmark (Version 0.15.0)* [Computer software]. `https://github.com/open-mmlab/mmclassification`

[28] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc.

[29] Smith, L., Cyclical Learning Rates for Training Neural Networks (2017). *IEEE Winter Conference on Applications of Computer Vision (WACV)* , 2017, pp. 464-472.

[30] Roy,J.S., Mitchell, S.A. and Peschiera, F. (2003). *PulP 2.6.0*. Python Package Index - PyPI. (n.d.). Python Software Foundation. Retrieved from `https://pypi.org/project/PuLP/`