Pruning in DNNs with SFW

2053346 Giovanni Dal Mas
2050573 Horațiu Andrei Palaghiu
2044713 Matilde Sboarina
2056100 Davide Varotto

July 13, 2022

1 Abstract

We study the paper **"Learning Pruning-Friendly Networks via Frank-Wolfe: One-Shot, Any-Sparsity, And No Retraining"** by [Lu et al]. [Mia+21]. Firstly, we work through the theoretical part of their analysis, investigating how the problem of network pruning can be solved with a Stochastic Frank Wolfe (SFW) framework.

To further elaborate on this, we will also discuss some other papers regarding the analysis of the FW methods for non-convex, variance-reduced, projection-free optimization.

Moreover, we planned to personally code these methods by replicating the experiments done in *Section 5* [Mia+21]: there, the datasets used are the classical CIFAR10 and Tiny-ImageNet, while two known architectures, ResNet18 VGG16, will serve as the understructure for testing different implementation, optimizing and pruning techniques.

Unfortunately, due to time constraints and the limited power of our personal mundane computers, we opted for a normal, simpler CNN, in order to arrive at the same conclusions as the original authors, while having lower accuracy scores.

2 Introduction

We study the very specific problem of pruning a large deep neural network. This is an extremely meaningful optimization issue, as deep neural networks (DNN) achieve great results but not without high computational costs.

Ways of addressing this problem are applied by neural network compression studies, such as removing superfluous parts of a network by acting on the network weights, i.e. pruning. This process might as well be quite burdensome, therefore we will be address various optimization problems that facilitate it[HL16]:

- the interest in avoiding iterative pruning or re-training the network while still preserve competitive accuracy
- the formulation of the process through an auxiliary K-sparse polytope constraint
- the resolution of leveraging a SFW algorithm to solve this constrained optimization
- the consequential problem regarding the initialisation scheme for SFW-based(Stochastic Frank-Wolf is not a gradient based optimization method) DNNs.

3 Pruning

Pruning can be unstructured or structured. The first one aims to cut out individual parameters in the networks, whilst the second one proceeds with grouping those parameters and acts on the entire group's removal. The second option is tidier and more effective in terms of saving energy and memory, yet unstructured pruning still remains more accurate.

Pruning methods before this study did not address the matter of retraining prohibition and did not consider our desired sparse structure. The traditional SGD algorithm gives unsatisfactory performances under these conditions, as it finds pruning weights through standard gradient-based optimization during training, and does not handle structured constraints as nicely as the SFW.

4 Any-Sparsity

Following the intention of not retraining the network, we must make some restrictions on the domain of our problem. Given a dataset $\mathbf{D} = (x_i, y_i)_{i=1}^n$ where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ and loss function $l : \mathcal{Y} \times \mathcal{Y} \to R^+$ we aim to train a deep neural network $f(\theta; .) : \mathcal{X} \to \mathcal{Y}$.

which minimizes the following pruning-aware objective:

$$\min_{\theta \in C} \frac{1}{n} \sum_{i=1}^{n} l(f(\theta; x_i), y_i) := \min_{\theta \in C} L(\theta)$$

(1)

The feasible parameter θ must lay in a convex region which we will more specifically impose to be a K-sparse polytope. This will make the process more pruning-aware and pruning-friendly. Further information about the optimization of the SFW in such domain may be found in [Mia+21].

5 Frank-Wolfe Method

The Frank-Wolfe method (otherwise known as conditional gradient method or reduced gradient method) is an iterative first-order optimization algorithm. It recently regained a tremendous popularity in machine-learning in terms of its projection free property and ability to exploit structured constraints. In comparison with the Projected Gradient method, its popular alternative, this greatly improves the computational aspect. In fact, the projection onto the constrained set (which generally requires solving a quadratic program) is replaced with a linear optimization over the constrained set at each iteration.

During the course of our experiment we will be using a one-shot Stochastic Frank Wolfe with no re-training.

Algorithm 1 `Frank-Wolfe method`

```
1 Choose a point $x_1 \in C$
2 For $k = 1, \ldots$
3     Set $\hat{x}_k = \underset{x \in C}{\text{Argmin}} \nabla f(x_k)^\top (x - x_k)$
4     If $\hat{x}_k$ satisfies some specific condition, then STOP
5     Set $x_{k+1} = x_k + \alpha_k(\hat{x}_k - x_k)$, with $\alpha_k \in (0, 1]$
      suitably chosen stepsize
6 End for
```

5.1 SGD Comparison

We consider the optimization problem:

$$\min_{x \in \Omega} F(x) := \begin{cases} \mathbf{E}_z[f(x, z)], & \text{(stochastic)} \\ \frac{1}{n} \sum_{i=1}^{n} f_i(x) & \text{(finite-sum)} \end{cases} \tag{2}$$

Where F, f and f_i $i \in \{1, ..., n\}$ are all differentiable but not convex, and the domain Ω is convex and compact. It can be studied with a classic Projected Stochastic Gradient Descent, but it becomes impractical as projecting onto Ω can be very costly (requiring to solve a quadratic program in the general case), while FW methods avoid this additional projection step and only require a *linear oracle*.

Projection free-methods replace projection onto the constraint set with a linear optimization over the constraint at each iteration. The standard Frank-Wolfe, while being project-free still needs to compute an exact gradient at each iteration, which may also be prohibitively expensive. For this reason the Frank-Wolfe method, in its stochastic and finite-sum versions, was the object of several studies which aimed to its optimization. Below may be found further insights of the Frank Wolfe algorithm and its variants.

5.2 Stochastic FW

The simplest way to think of a stochastic Frank-Wolfe variant is to replace the gradient at iteration k, $\nabla f(x_k)$ with its stochastic approximation $\nabla f_i(x_k)$, or the average of some number of independent identically distributed samples of $\nabla f_i(w_k)$. Its analysis is included in "Appendix B" of [HL16].
If we suppose each f_i is G-Lipschitz, then with $\gamma_k = \frac{2}{k+1}$ and $m_k = (\frac{G(k+1)}{LD})^2$, SFW ensures for any k $E[f(v_k) - f(w^*)] \leq \frac{4LD^2}{k+2}$ which assures a $O(\frac{1}{\epsilon^3})$ stochastic gradients and $O(\frac{1}{\epsilon})$ linear optimization steps to reach a ϵ-approximate optimum.

5.3 Non-convex SFW

While the Frank-Wolfe algorithm is commonly seen in its convex version, let us impose *convexity* and *compactness* on the domain Ω, but not on the differentiable functions F, f and f_i, which can possibly be *non convex*. The analysis of [Red+16] clarifies this issue, proposing a Stochastic Frank-Wolfe algorithm for the nonconvex stochastic setting, and two variance-reduced algorithms (SVFW and SAGAFW) for the nonconvex finite-sum which can also be faster than deterministic FW.

The classical FW algorithm requires a calculation of the gradient ∇f at each iteration which may not always be possible in the stochastic problem. In the convex setting this should be no trouble, as there are popular approximations which cope with this issue, such as the Robbins-Monro approximation to the gradient. In the non-convex case it is different issue, therefore the Stochastic FW algorithms helps us to face this scenario. *Theorem 2* in [Red+16] proves the convergence of such said stochastic setting.

Algorithm 2 Nonconvex SFW $(x_0, T, \{\gamma_i\}_{i=0}^{T-1}, \{b_i\}_{i=0}^{T-1})$

```
1: Input: $x_0 \in \Omega$, number of iterations $T$, $\{\gamma_i\}_{i=0}^{T-1}$ where $\gamma_i \in [0,1]$ for all $i \in \{0, \dots, T-1\}$, minibatch size $\{b_i\}_{i=0}^{T-1}$
2: for $t = 0$ to $T-1$ do
3:     Uniformly randomly pick i.i.d. samples $\{z_1^t, \dots, z_{b_t}^t\}$ according to the distribution $\mathcal{P}$.
4:     Compute $v_t = \arg\max_{v \in \Omega} \langle v, -\frac{1}{b_t}\sum_{i=1}^{b_t} \nabla f(x_t, z_i)\rangle$
5:     Compute update direction $d_t = v_t - x_t$
6:     $x_{t+1} = x_t + \gamma_t d_t$
7: end for
8: Output: Iterate $x_a$ chosen uniformly random from $\{x_t\}_{t=0}^{T-1}$.
```

Proof of Convergence:

Because of the constrained setting our problem is laying on, it will not be possible to measure convergence by letting $||F||$ go to 0. Instead we need set $\mathcal{G}(x) = 0$ where

$$\mathcal{G}(x) = \max_{v \in \Omega} < v - x, -\nabla F(x) > . \qquad (3)$$

This being said, let us consider $f(x, z)$ a G-Lipschitz function for all $z \in \Xi \subset R^p$ and F is L-smooth.

F(x_{t+1}) $\leq$
$\leq F(x_t) + < \nabla F(x_t), (x_{t+1} - x_t) > + \frac{L}{2}||x_{t+1} - x_t||^2 =$
$= F(x_t) + < \nabla F(x_t), \gamma(v_t - x_t) > + \frac{LD^2\gamma^2}{2} \leq$
$\leq F(x_t) + \gamma < \nabla F(x_t), (v_t - x_t) > +$
$+ \frac{LD^2\gamma^2}{2}$

Let us introduce

$$\hat{v} :=_{v \in \Omega} < v, -\nabla F(x_t) >, \qquad (4)$$

$$\nabla_t = \frac{1}{b}\sum_{i=1}^{b} \nabla f(x_t, z_i^t) \qquad (5)$$

and recall
$\mathcal{G}(x) = \max_{v \in \Omega} < v - x, -\nabla F(x) >$
We see that:
$F(x_{t+1}) \leq F(x_t) + \gamma < \nabla_t, v_t - x_t > +$
$+ \gamma < \nabla F(x_t) - \nabla_t, v_t - x_t > + \frac{LD^2\gamma^2}{2}$
$\leq F(x_t) + \gamma < \nabla_t, \hat{v}_t - x_t > +$
$+ \gamma < \nabla F(x_t) - \nabla_t, v_t - x_t > + \frac{LD^2\gamma^2}{2}$
$\leq F(x_t) - \gamma \mathcal{G}(x_t) + \gamma < \nabla F(x_t) - \nabla_t, v_t - \hat{v}_t >$
$+ \frac{LD^2\gamma^2}{2}$
$\leq F(x_t) - \gamma \mathcal{G}(x_t) + D\gamma||(x_t) - \nabla_t|| + \frac{LD^2\gamma^2}{2}$

Let us compute the expectation:
$E[F(x_{t+1})] \leq E[F(x_t)] - \gamma E[\mathcal{G}(x_t)] + \frac{GD\gamma}{\sqrt{b}} + \frac{LD^2\gamma^2}{2}$;
$\gamma \sum_{t=0}^{T-1} E[\mathcal{G}(x_t)] \leq F(x_0) - E[F(x_T)] + \frac{TGD\gamma}{\sqrt{b}} + \frac{TLD^2\gamma^2}{2}$

Assigning $\gamma = \sqrt{\frac{F(x_0) - F(x^*)}{TLd^2\beta}}$
We obtain:
$\gamma \sum_{t=0}^{T-1} E[\mathcal{G}(x_t)] \leq \frac{F(x_0) - F(x^*)}{T\gamma} + \frac{GD}{\sqrt{b} + \frac{LD^2\gamma}{2}} \leq$
$\leq \frac{D}{\sqrt{T}}(G + \sqrt{\frac{2L(F(x_0) - F(x^*))}{\beta}}(1 + \beta))$
The *Stochastic First-Order Oracle* complexity and *Linear Oracle* complexity are $O(\frac{1}{\epsilon^4})$ and $O(\frac{1}{\epsilon^2})$ respectively.

5.4 Stochastic Variance Reduced Frank-Wolfe SVRF

The SVRF algorithm simply replaces the classic stochastic gradient with a new variance-reduced gradient in the standard FW algorithm.

Algorithm 1 Stochastic Variance-Reduced Frank-Wolfe (SVRF)

```
1: Input: Objective function $f = \frac{1}{n}\sum_{i=1}^{n} f_i$.
2: Input: Parameters $\gamma_k$, $m_k$ and $N_k$.
3: Initialize: $\boldsymbol{w}_0 = \min_{\boldsymbol{w} \in \Omega} \nabla f(\boldsymbol{x})^\top \boldsymbol{w}$ for some arbitrary $\boldsymbol{x} \in \Omega$.
4: for $t = 1, 2, \dots, T$ do
5:     Take snapshot: $\boldsymbol{x}_0 = \boldsymbol{w}_{t-1}$ and compute $\nabla f(\boldsymbol{x}_0)$.
6:     for $k = 1$ to $N_t$ do
7:         Compute $\tilde{\nabla}_k$, the average of $m_k$ iid samples of $\tilde{\nabla} f(\boldsymbol{x}_{k-1}, \boldsymbol{x}_0)$.
8:         Compute $\boldsymbol{v}_k = \min_{\boldsymbol{v} \in \Omega} \tilde{\nabla}_k^\top \boldsymbol{v}$.
9:         Compute $\boldsymbol{x}_k = (1 - \gamma_k)\boldsymbol{x}_{k-1} + \gamma_k \boldsymbol{v}_k$.
10:    end for
11:    Set $\boldsymbol{w}_t = \boldsymbol{x}_{N_t}$.
12: end for
```

A variance-reduced stochastic gradient at some point $x \in \Omega$ with some snapshot $\tilde{x} \in \Omega$ may be defined as:

$$\tilde{\nabla} f(x; \tilde{x}) = \nabla f_i(x) - (\nabla f_i(\tilde{x}) - \nabla f(\tilde{x})) \qquad (6)$$

where i is picked from $\{1, ..., n\}$ uniformly at random. Because $\nabla f(\tilde{x}$ has already been computed in the previous part of the algorithm (step 3), the calculation of $\tilde{\nabla} f(x; \tilde{x})$ only requires two standard stochastic gradient evaluations $\nabla f_i(x)$ and $(\nabla f_i(\tilde{x})$.

Its convergence analysis may be found in [HL16]. The algorithm ensures $[f(w_t) - f(w*)] \leq \frac{LD^2}{2^{t+1}}$ with the parameters $\gamma_k = \frac{2}{k+1}$, $m_k = 96(k+1)$, $N_t = 2^{t+3} - 2$, and to achive $1 - \epsilon$ accuracy the algorithm requires $O(ln\frac{LD^2}{\epsilon})$ exact gradient evaluations, $O(\frac{L^2D^4}{\epsilon})$

stochastic gradient evaluations and $O(\frac{LD^2}{\epsilon})$ linear optimizations.

5.5 Variance Reduced SFW for Nonconvex Optimization

The paper [Red+16] also investigates two variance reduced methods for the finite-sum setting which are the SVFW and the SAGAFW. They both use a variance-reduced approximation of the gradient. In order to do so, the descent direction may be found by solving the problem:

$$\min_{x\in\Omega} < x, -\frac{1}{b_t}(\sum_{i\in I_t} \nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}_t^{s+1}) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{x}_t^{s+1})) > \quad (7)$$

We notice that the last term is the stochastic gradient of our problem, while the previous terms help with the variance reduction.

For the SVFW, which is based in the convex method and applied on the nonconvex setting, the IFO Complexity is $\Theta(m^3 + n)$ with m being the number of epochs for our code. Each inner iteration requires $O(1)$ LO calls making the total LO complexity $O(\frac{1}{\epsilon^2})$. For the SAGAFW, based on the SAGA reduction technique, the IFO complexity is $O(\frac{n+n^{1/3}}{\epsilon^2})$ and the LO complexity is $O(\frac{1}{\epsilon^2})$.

5.6 One-Shot SFW

In One-Shot Learning the model is given only one instance for training data and has to learn to re-identify that instance in the testing data. Since retraining is not allowed in our analysis, the choice to implement a One-Shot algorithm comes natural.

From the work of [Che+18] the One-Shot Frank-Wolfe is described as a projection-free algorithm which only requires a single stochastic gradient estimate in each round. Let us consider the following optimization problem:

$$\min_{w\in\Omega} f(w) = \min_{w\in\Omega} \frac{1}{n}\sum_{i=1}^{n} f_i(w) \quad (8)$$

Especially when dealing with a large data set (very large n), traditional gradient based methods (which include the FW algorithm that, although being projection-free, needs to compute an exact gradient computation at each iteration) require massive computational costs, therefore a stochastic evaluation for the gradient must be required. With stochastic methods, on the other hand, the variance estimation may cause an issue. This is why several reduction techniques were introduced and studied in the past years, and a One-Shot SFW with Variance Reduction was raised and compared to the classic One-Shot FW.

Algorithm 2 One-Shot Frank-Wolfe

```
Input: convex set $\mathcal{K}$, time horizon $T$, step sizes $\rho_t \in (0,1)$
    and $\eta_t \in (0,1)$, and initial point $\mathbf{x}_1$
Output: $\{\mathbf{x}_t : 1 \le t \le T\}$
1: $\mathbf{d}_0 \leftarrow 0$
2: for $t \leftarrow 1, 2, 3, \ldots, T$ do
3:     Play $\mathbf{x}_t$, then obtain value $f_t(\mathbf{x}_t)$ and unbiased oracle
       access to $\nabla f_t$
4:     $\mathbf{d}_t \leftarrow (1-\rho_t)\mathbf{d}_{t-1} + \rho_t \tilde{\nabla} f_t(\mathbf{x}_t)$
5:     $\mathbf{v}_t \leftarrow \arg\max_{\mathbf{v}\in\mathcal{K}} \langle \mathbf{d}_t, \mathbf{v} \rangle$
6:     $\mathbf{x}_{t+1} \leftarrow update(\mathbf{x}_t, \mathbf{v}_t, \eta_t)$
7: end for
```

The One-Shot Frank-Wolfe achieves $O(T^{2/3})$ regret for the convex case.

6 Stochastic Frank-Wolfe Pruning Framework

In the main paper, the DNN training process is formulated as an explicit pruning-aware process with an auxiliary K-sparse polytope constraint. The correspondent constrained optimization problem is solved via a SFW algorithm.

6.1 Pruning-Aware DNN Training and K-Sparse Polytope Constraints

Given a dataset $D = (x_i, y_i)_{i=1}^n$ where $(x_i, y_i) \in X \times Y$ and loss function $l : Y \times Y \to R^+$ we aim to train a deep neural network we aim to train a DNN that minimizes the pruning-aware object:

$$\min_{\theta\in C} \frac{1}{n}\sum_{i=1}^{n} l(f(\theta; x_i), y_i) := \min_{\theta\in C} L(\theta) \quad (9)$$

To make the training pruning-aware, we restrict the feasible parameter θ in a convex region C, that potentially results in more sparse weight matrices. We choose C as a K-sparse polytope and we solve (9) via SFW algorithm.

At each iteration t, SFW first calls a liner minimization oracle v_t that uses an estimate of the full gradient. Then, along the direction of v_t, SFW updates θ_t by a convex combination $\theta_{t+1} = \alpha_t v_t(1-\alpha)$, with learning rate $\alpha_t \in [0,1]$. This keeps θ_t always in C and saves any projection step.

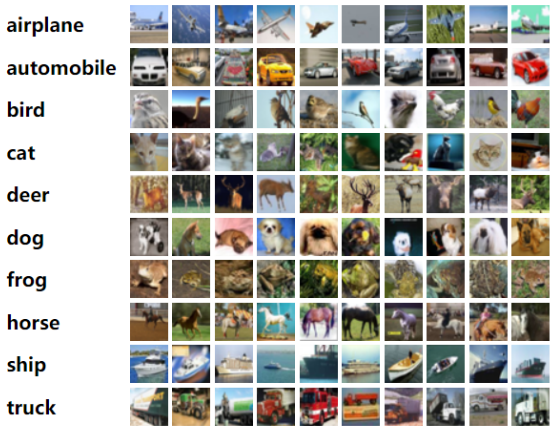
At each step of our optimization, the linear minimization oracle of a K-sparse polytope constraint returns a update vector with exactly K non-zero coordinates, which is then weighted-averaged with the current θ_t according to the SFW algorithm. By adding such a constraint, each SFW step pushes less important weights smaller, enhancing the important ones. Moreover, the amounts of weights, at different magnitude levels, change more smoothly, resulting in a good weight distribution that does not cause sudden "jump" when the pruning ratio increases.

7 Experiments

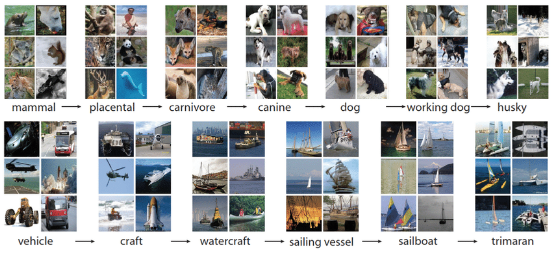
We tried to reproduce the paper's experiments [Mia+21] conducted with Res-Net-18 and VCG-16 on the datasets CIFAR-10. The running of the entire model performed by the paper's authors would take hundreds of hours, which was obviously not empirically doable in our case. Therefore we did implement a simplified version of it, which we will coherently justify in the course of this paragraph.

7.1 The Datasets

CIFAR10 is a dataset of 50,000 32x32 color training images and 10,000 test images, labeled over 10 (mutually exclusive) categories. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton [Kri09].



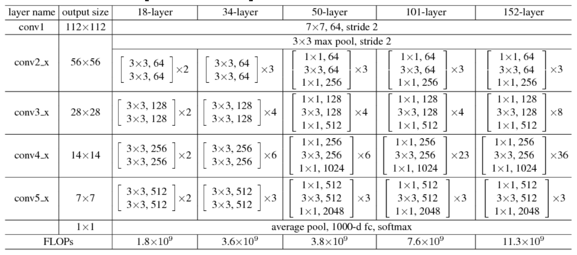
Tiny ImageNet is a smaller version of *The ImageNet Large Scale Visual Recognition Challenge(ILSVRC)* [Rus+15]. It has 64x64 images, split into 200 classes, with a training dataset of 100k images, a validation dataset of 10k images, and a test dataset of 10k images.



On both datasets we used data augmentation techniques so that our models generalize better. Moreover, all images have been re-scaled and normalized, in order to reduce data skewness.

7.2 The Architectures

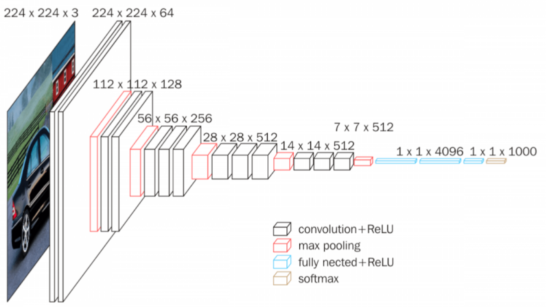
ResNet, or Residual Network Models, are NNs that make use of *skip connections* in order to reduce the typical problem of vanishing gradients, whilst speeding up the learning process. In particular, we will be using a **ResNet18** model, as seen in [**He2016**].



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

VGG16 is a CNN that makes use of multiple 3x3 convolutional stacks - it has been identified to be the best performing model on the aforementioned ImageNet dataset at its time, and was proposed by the Visual Geometry Group of the university of Oxford. [SZ14]



Both our models are trained from scratch, but we used the hyperparameters and specific architecture proposed by the authors:

- **ResNet18** is built strictly from basic small ResNet blocks, along with standard CONV2D, batch normalization and pooling layers
- **VGG16**, unlike **ResNet**, makes use of

fully connected layers, which makes it quite slow to train;

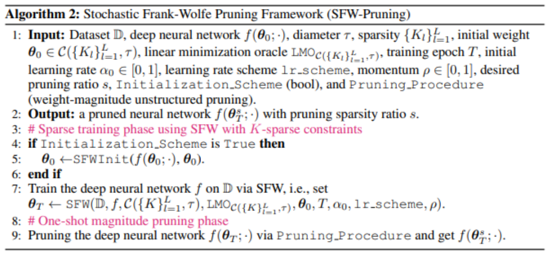
- both are trained for 150 epochs, and use convolutions on 3 color channels and ReLu activation functions where we change the number of classes depending on the dataset.

7.3 SFW One-shot Pruning without Retraining

In our one-shot pruning framework, a DNN is trained using the pruning-aware objective (11) with K-sparse constraints via SFW for only once and then undergoes a one-shot pruning. The pruning procedure adopted is the standard weight-magnitude unstructured pruning. In order to discover pruning-friendly weights, the proper choice of several hyperparameters is of paramount importance.

$\{K_l\}_{l=1}^{L}$ controls the sparsity of the linear minimization oracle solution, and τ (the radius of C) controls the magnitude, which together influence the distribution of the learned parameters. $K_l = 5\%$ (of the weights in the layer) and $\tau = 15$ are proven to be appropriate choices to obtain good pruning performance and satisfactory full model performance.

Another important hyperparameter is the learning rate α. Larger learning rates encourage smaller weights and better performance. Therefore, it is possible to control the nature sparsity (largest pruning ratio that keeps the test performance undamaged) of the NN by controlling the scale of α_t.



Algorithm 2: Stochastic Frank-Wolfe Pruning Framework (SFW-Pruning)

```
1: Input: Dataset $\mathbb{D}$, deep neural network $f(\boldsymbol{\theta}_0;\cdot)$, diameter $\tau$, sparsity $\{K_l\}_{l=1}^{L}$, initial weight
   $\boldsymbol{\theta}_0 \in \mathcal{C}(\{K_l\}_{l=1}^{L},\tau)$, linear minimization oracle $\texttt{LMO}_{\mathcal{C}(\{K_l\}_{l=1}^{L},\tau)}$, training epoch $T$, initial
   learning rate $\alpha_0 \in [0,1]$, learning rate scheme lr_scheme, momentum $\rho \in [0,1]$, desired
   pruning ratio $s$, Initialization_Scheme (bool), and Pruning_Procedure
   (weight-magnitude unstructured pruning).
2: Output: a pruned neural network $f(\boldsymbol{\theta}_T^s;\cdot)$ with pruning sparsity ratio $s$.
3: # Sparse training phase using SFW with $K$-sparse constraints
4: if Initialization_Scheme is True then
5:     $\boldsymbol{\theta}_0 \leftarrow$ SFWInit($f(\boldsymbol{\theta}_0;\cdot), \boldsymbol{\theta}_0$).
6: end if
7: Train the deep neural network $f$ on $\mathbb{D}$ via SFW, i.e., set
   $\boldsymbol{\theta}_T \leftarrow$ SFW($\mathbb{D}, f, \mathcal{C}(\{K\}_{l=1}^{L},\tau), \texttt{LMO}_{\mathcal{C}(\{K\}_{l=1}^{L},\tau)}, \boldsymbol{\theta}_0, T, \alpha_0$, lr_scheme, $\rho$).
8: # One-shot magnitude pruning phase
9: Pruning the deep neural network $f(\boldsymbol{\theta}_T;\cdot)$ via Pruning_Procedure and get $f(\boldsymbol{\theta}_T^s;\cdot)$.
```

7.4 Initialization scheme for SFW

The greater part of traditional initialization schemes are designed to be applied on gradient-based methods. Those techniques are focused on avoiding a gradient explosion or collapse. As we are using a stochastic gradient estimate, the paper [Mia+21] proposes a new initialization sceme that further improves the accuracy score of this novel method.

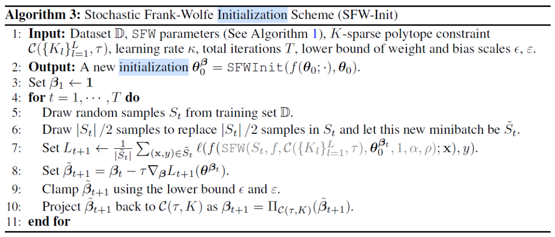
It aims to define the parameter θ_0 of weights matrices $\{W_l\}_{l=L}^{L}$ and bias vectors $\{b_l\}_{l=L}^{L}$ ($\theta = (W_1, b_1, ..., W_L, b_L)$) so that in the first SFW step the loss function

$$L(S;\theta) = \frac{1}{|S|} \sum_{(x,y)\in S} l(f(\theta;x),y) \tag{10}$$

will be maximized. To do so, they initialize each layer with a uniform distribution and multiply it by a layer-wise optimized scaling parameter β (chosen so that the loss on $\theta_1 = SFW(A, f, C(\{K_l\}_{l=1}^{L}, \tau), \theta_0, l, \alpha_o, \rho)$ is as low as possible), to obtain a good SFW initialization parameter θ^{β}. They optimize β through the following objective:

$$\min_{\beta s.t.\theta_0^{\beta} \in C(K,\tau)} L(\tilde{S}; SFW(S, f, C(\{K_l\}_{l=1}^{L}, \tau), \theta_0, 1, \alpha_0, \rho)) \tag{11}$$

where S and $\tilde{S}$ are two different random minibatches. The minibatches are different but overlapping, respectively to prevent overfitting and to have better train performance. Finally, they constrain the re-scaled parameter θ^{β} to $C(K,\tau)$, to make sure it is still a feasible initialization.



Algorithm 3: Stochastic Frank-Wolfe Initialization Scheme (SFW-Init)

```
1: Input: Dataset $\mathbb{D}$, SFW parameters (See Algorithm 1), $K$-sparse polytope constraint
   $\mathcal{C}(\{K_l\}_{l=1}^{L},\tau)$, learning rate $\kappa$, total iterations $T$, lower bound of weight and bias scales $\epsilon$, $\varepsilon$.
2: Output: A new initialization $\boldsymbol{\theta}_0^{\boldsymbol{\beta}}$ = SFWInit($f(\boldsymbol{\theta}_0;\cdot), \boldsymbol{\theta}_0$).
3: Set $\boldsymbol{\beta}_1 \leftarrow \mathbf{1}$
4: for $t = 1, \cdots, T$ do
5:     Draw random samples $S_t$ from training set $\mathbb{D}$.
6:     Draw $|S_t|/2$ samples to replace $|S_t|/2$ samples in $S_t$ and let this new minibatch be $\tilde{S}_t$.
7:     Set $L_{t+1} \leftarrow \frac{1}{|\tilde{S}_t|}\sum_{(\mathbf{x},y)\in\tilde{S}_t} \ell(f(\texttt{SFW}(S_t, f, \mathcal{C}(\{K_l\}_{l=1}^{L},\tau), \boldsymbol{\theta}_0^{\boldsymbol{\beta}_t}, 1, \alpha, \rho);\mathbf{x}), y)$.
8:     Set $\tilde{\boldsymbol{\beta}}_{t+1} = \boldsymbol{\beta}_t - \tau\nabla_{\boldsymbol{\beta}} L_{t+1}(\boldsymbol{\theta}^{\boldsymbol{\beta}_t})$.
9:     Clamp $\tilde{\boldsymbol{\beta}}_{t+1}$ using the lower bound $\epsilon$ and $\varepsilon$.
10:    Project $\tilde{\boldsymbol{\beta}}_{t+1}$ back to $\mathcal{C}(\tau, K)$ as $\boldsymbol{\beta}_{t+1} = \Pi_{\mathcal{C}(\tau,K)}(\tilde{\boldsymbol{\beta}}_{t+1})$.
11: end for
```

7.5 Implementation Details

We implemented the code to replicate the experiments with ResNet-18 and VGG-16 models, but due to excessively long training time we were forced to use a simpler model to run the experiments. With ResNet-18 and VGG-16 each epoch would have taken one hour to run on our computers, multiplied for 180 epoch per model it is more than 350 hours, definitely too much. This is why we opted to run everything we have implemented on a classic Convolutional Neural Network with 2 convolutional layers and 3 standard ones. Nevertheless, to work around training running time problems and test anyway the two main models we turned to the so called `state_dict` , a file that contains parameters from already trained model, where we took the weights to use. Unfortunately, this was only possible for SGD because SFW was not implemented by other people and therefore there was not an available `state_dict` to download. Hence, for ResNet and VGG models we only have the accuracy of the model with and without pruning, using SGD. We preserved the hyperparameters in the paper, namely:

Hyperparameters	Value	Hyperparameters	Value	Hyperparameters	Value
Initial learning rate α_0	1.0	Training batchsize	128	Test batchsize	100
Radius τ	15	K-frac $\{K_l\}_{l=1}^{L}$	5%	Training epoch T	180
Momentum ρ	0.9				

The learning rate α follows a changing scheme. We decrease the learning rate by 10 at epoch 61 and 121. If the 5-epoch average loss is greater than the 10-epoch average loss it is multiplied by 0.7 , while it is increased by 1.06 if the 10-epoch average loss is greater than the 5-epoch average loss.
Moreover the effective learning rate is rescaled following this update rule:

$$\theta_{t+1} = \theta_t + \min\{\alpha||\hat{\nabla} L(\theta_t)||_2/||v_t - \theta_t||_2, 1\}(v_t - \theta_t) \tag{12}$$

7.6 Prerequisites

To give a nice professional feel to our code and make it easier to read, we opted for the use of OOP (Object oriented programming). Creating our own classes, objects and methods makes our code easily reusable and extensible, whilst being able to extend the functionality of already known packages (like `torch.optim`).

7.6.1 Optimizers

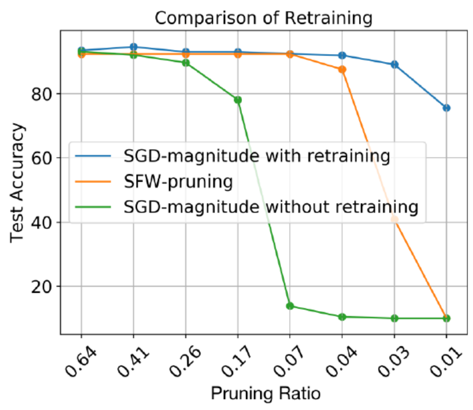
We compare the classical **SGD** (with extra retraining costs) with our **SFW** optimizer, as presented in *Section 5* above. For both, a learning rate of 0.1 is optimal. When implementing SFW, we found that a momentum of 0.9 yields the best results for these particular datasets. Moreover, we made it so that one can experiment with different types of constraints:

Besides the **K-Sparse polytope** constraint detailed in our *Section 6.1*, we also coded an Unconstrained parameter class, that takes as only parameter the dimension of the parameter space.

7.6.2 Pruning and K-sparse polytope constraints

For the implementation of pruning we use the library *prune* from Pytorch (torch.nn.utils.prune). This library allows us to implement both unstructured and structured pruning. We chose the pruning technique the paper addresses as more efficient, namely weight-magnitude unstructured pruning (prune.l1_unstructured(module, name='weight', amount= 0.1)). This type of pruning does not eliminate connections among layers, instead it sets them close to zero, creating a sort of 'mask'. These weights are chosen through the *weight-magnitude criterion*, under which less important weights are pushed smaller, while important ones are enhanced. This process is further aided by restricting the feasible parameter j of (9) in a convex region C (see section 6.1), which results in more sparse weight matrices and will be more pruning-friendly for only a small percentage of the weights (the ones of large magnitudes). Specifically, we choose C as a K-sparse polytope and we solve (9) via SFW algorithm. Concretely we implement K-sparse polytope constraints to apply to SFW. As per the original authors suggested, we included different pruning ratios. The plot below from the paper suggests that we should expect FW method to provide competitive results with SGD-pruning with extra retraining costs, while also tolerating pruning ratios below.

Comparing the SFW and the Gradient Descent methods in the paper, the SGD performs worse when not retrained right from the beginning. Both (SGD&SFW) have comparable accuracy scores until a 26% pruning ratio. Instead, the SGD-pruning *with* extra retraining cost works better then the SFW-pruning but the methods have a similar (> 80%) accuracy until a pruning ratio of 0.5%.



7.7 Diagnostics & Plots

7.7.1 Accuracy comparison on CIFAR-10

In the following table one may compare the accuracy of SGD and SFW methods for the CIFAR-10 dataset with and without a 40% pruning ratio, with and without retraining.

Optimizer	Pruning	Retraining	Accuracy
SGD	NO	NO	0.574
SGD	YES (40%)	NO	0.5486
SGD	YES (40%)	YES	0.6297
SFW	NO	NO	0.5777
SFW	YES (40%)	NO	0.5522
SFW	YES (40%)	YES	0.5386

Due to the empirical reasons we mentioned above in our abstract, our model can not be as complex as the one used in [Mia+21]. Consequentially, it makes several mistakes in the classification, therefore the performances in our model obtain lower results. This being said, we can still see some main features being comparable between our results and theirs.

The ResNet18 and VGG16 work better with SGD with pruning and retraining. Our simplified model also finds itself to be the most competitive with a 0.6297 accuracy. SFW detains a worse accuracy with pruning and retraining both in the original sophisticated [Mia+21] model and in ours, with a 0.5386 accuracy vs a 0.5777 and 0.5522 for the SFW with pruning and without retraining, and without pruning and without retraining respectively.

8 Conclusions and further research

8.1 Conclusion

All in all, with our simpler model, which remains faithful to the implementations of all the requests of section (5) of [Mia+21] (SFW/SGD, constrained/unconstrained, with/without pruning, with/without retraining), it is clear that training the network weights within in a convex polytope spanned by K-sparse vectors, yields much lower weights, yet "smoother" magnitude distribution, that is more amendable to gradual weight removal towards any pruning ratio.

8.2 Further research

But, if we were to paraphrase the original text : *"their work is a pilot study to demonstrate the potential of non-gradient optimization methods in training deep networks, especially when additional weight structures like sparsity are desired."* This means that the introduction of a better initialization scheme and a new constraint class opens up a whole world of possibilities. Thus, many of the SFW algorithm variations could be used as optimizers for experimentation, such as the ones we explained in *section 5* above. Furthermore, other constraints, such as the usual L_p balls, could be integrated into this approach.

References

[Kri09] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009, pp. 32–35.

[SZ14] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[Rus+15] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[HL16] Elad Hazan and Haipeng Luo. "Variance-reduced and projection-free stochastic optimization". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1263–1271.

[Red+16] Sashank J Reddi et al. "Stochastic frank-wolfe methods for nonconvex optimization". In: *2016 54th annual Allerton conference on communication, control, and computing (Allerton)*. IEEE. 2016, pp. 1244–1251.

[Che+18] Lin Chen et al. "Projection-free online optimization with stochastic gradient: From convexity to submodularity". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 814–823.

[Mia+21] Lu Miao et al. "Learning Pruning-Friendly Networks via Frank-Wolfe: One-Shot, Any-Sparsity, And No Retraining". In: *International Conference on Learning Representations*. 2021.