Technical University of Munich

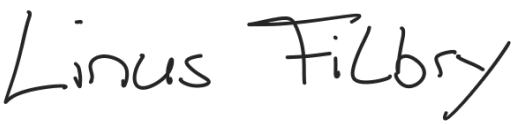
Department of Mathematics
Department of Computer Science

Frank-Wolfe Methods for Neural Network Training

Bachelor's Thesis

Author: Linus Filbry
Supervisor: Prof. Dr. Michael Ulbrich
Submission Date: 05.03.2025

I hereby declare that I have independently prepared this Bachelor's Thesis using only the resources and tools specified.



Place, Date: Munich, 05.03.2025

Contents

Abstract

This thesis explores the usage of the Frank-Wolfe method as optimizer for neural networks, demonstrating its effectiveness through both mathematical proof and empirical comparison with classical optimization methods such as SGD. It introduces parameter constraints that, when combined with the Frank-Wolfe method, lead to sparse and pruning-friendly neural networks. The same constraints, in a modified form which encourages structured sparsity, are further applied to convolutional neural networks. When optimized with the Frank-Wolfe method, some of these networks exhibit exceptional resistance not only to unstructured, but especially to structured filter pruning, where they significantly outperform approaches like SGD with weight decay.

1 Introduction

Neural networks have revolutionized modern fields such as computer vision and natural language processing. Most recently, large language models (LLMs) like ChatGPT, which rely on neural networks, have taken the world by storm. However, their impressive performance often comes at the cost of immense complexity, with modern LLMs containing up to hundreds of billions of tunable parameters. This has relevant implications not only for training but also for end users. Larger models require more memory, take longer to generate outputs, and consume more energy [Canziani et al. (2016), sec. 3.2-3.7].

To address these issues, researchers have explored ways to reduce network size while maintaining performance. One effective approach [Grigas et al. (2019), ch. 1] is training sparse neural networks, where many parameters are close to zero. Another is pruning, which involves removing individual parameters (unstructured pruning) or entire parameter groups of some structure (structured pruning) from a trained network. The goal is to achieve the efficiency of a smaller model while preserving the performance of the original. Sparse networks may be particularly well-suited for pruning since low-magnitude weights can often be removed with minimal impact.

This thesis investigates the Frank-Wolfe method as a neural network optimizer, which, under the right conditions, naturally promotes sparsity [Xie et al. (2020), ch. 2]. The

method is applied in combination with constraints on the network's parameters, introducing feasible regions that encourage sparse solutions in constrained optimization. The resulting networks are evaluated against traditionally trained models in terms of accuracy, sparsity, and performance after pruning.

Related Work The basis of this thesis is Pokutta et al. (2020), which introduces the idea of using the Frank-Wolfe method as constrained neural network optimizer and the combination of Frank-Wolfe and the K-sparse polytope as constraint to train sparse networks. This idea is further developed in Lu et al. (2022), who demonstrate that networks trained this way exhibit resistance to pruning. Subsequently, Zimmer et al. (2022) show even better performance of Frank-Wolfe optimized networks with respect to pruning with the introduction of the (Group) K-support norm ball as constraint. A proof of convergence for the Frank-Wolfe method as neural network optimizer is provided in Theorem 2 of Reddi et al. (2016), which all the mentioned papers cite.

Outline In chapter 2, different types of neural networks are introduced, with computational problems in the optimization process of constrained networks further motivating the usage of the Frank-Wolfe method as optimization algorithm. Chapter 3 goes on to mathematically define the Frank-Wolfe method as well as its variations SFW and MSFW and provides a formal proof of convergence for its use as neural network optimizer. Next, different constraints for neural network parameters are introduced in chapter 4, with the focus on finding constraints which, in combination with the Frank-Wolfe method, produce sparse update vectors. Finally, chapter 5 shows the results of experiments comparing different optimizers and constraints.

2 Neural Networks

2.1 Fully Connected Neural Networks

This section introduces fully connected neural networks; a more detailed explanation as well as further information can be found in Nielsen (2015).

A fully connected (or linear) neural network is a complex function consisting of several steps, so called layers, each of which contains a set of placeholders, so called neurons. Given some input data, the neurons will be filled one layer after the other, with the output (or prediction) of the network (also called model) being the values of the neurons in the final layer.

The neurons of the first (/input) layer are filled with exactly the values of the input data; their amount n therefore equals the dimension of the input data. Starting at the second layer (the first hidden layer), the number of neurons can be freely chosen. There is a weight $w_{i,j}$ connecting the i-th neuron of the first layer x_i to the j-th neuron of the second layer y_j (the name fully connected stems from all neurons of the previous layer being thus connected to all neurons of the next layer), as well as a bias b_j specific to y_j. Combined with a non-linear activation function σ used for the entire layer (or even network), the value of neurons in the second layer is determined as $y_j = \sigma\left(\sum_{i=1}^{n} w_{i,j} \cdot x_i + b_j\right)$. The values of the following layers are calculated the same way, taking the neurons of the previous layer as input and using new weights and biases.

Since the neurons of the final layer (which is also called output layer and, like the input layer, does not count as a hidden layer) serve as the prediction, their amount is chosen based on the goal of the network. For example, in image classification, a typical use case of neural networks which will be focused on in this thesis, data is provided in the form of a vector where each entry is the grayscale or RBG value of a pixel of an image, and the network should predict which of several predetermined classes the image belongs to. The number of output neurons would then be the number of different classes, with each class being represented by one of the neurons, and the prediction for a given image is the class whose corresponding neuron has the highest value.

The number of layers and neurons in each layer is fixed with the creation of a network, and can be called the architecture of a network. Fully connected neural networks are a type of supervised machine learning algorithm. That means that during an initial training phase, the tunable parameters of the network contributing to the prediction, i.e.

its weights and biases, are optimized using training data with known desired predictions. Specifically, a loss function $l(\theta, z)$ is defined which, for current parameters θ and input data z with known desired prediction, computes the quality of the networks' prediction. The network is trained by optimizing the parameters to minimize the expected loss over all training data. This minimization process is usually done using stochastic gradient descent (SGD) or related methods, since the gradient with regard to the loss function can be calculated efficiently using an algorithm known as backpropagation.

2.2 Convolutional Neural Networks

In this section, convolutional neural networks are defined and their usage is motivated; Wu (2017) gives a more exhaustive introduction and an explanation as to why convolutional filters are so effective when analyzing images.

Using a fully connected neural network to process images can lead to a very high number of weights. For a simple 28x28 pixel image with RBG values, $784 \cdot 3$ input neurons are required (one for every pixel and every color), and with 100 neurons in the first hidden layer, there would already be 235200 weights. While this is no problem whatsoever for modern computers, modern input data is typically a lot more complex. Additionally, the availability of computational resources in 1988 was a lot lower than today, which is why these observations in chapter 2 of LeCun et al. (1998) lead to the introduction of convolutional neural networks.

Convolutional networks are built the same way as fully connected ones, with so called convolutional layers added to the architecture. To explain convolutional layers, consider the example from above: An image of 28x28 pixels with RBG values can be considered as three separate images of dimension 28x28, one consisting of the red, one of the blue and one of the green values. A convolutional layer consists of a number of convolutional filters (or kernels), each of which is a matrix with a fixed width and height smaller than those of the input images. The matrices' elements are the weights of a convolutional layer. Each of these filters creates a new 'image' for the next layer in the following way: The value of the top left pixel of the new image is determined by fitting the filter in the top left corner of the original images and summing up the products of the weights with the corresponding pixel values in all input images. The value of the pixel one to the right is determined by moving the filter one step (or multiple, if another stride is specified) to the right and repeating this operation. Similarly, the pixel below is created by moving the filter downwards. The next convolutional layer treats all thus created images as inputs and creates new output images the same way. For classification tasks, the output layer is a fully connected layer, which treats all images it receives as one big flattened vector to process.

Using the same example and considering a first convolutional layer with filter width and height of 3, as well as 64 filters, there would only be 576 weights in a convolutional

layer, a massive improvement compared to the fully connected layer. Section 3.4 of Wu (2017) shows that it is still possible to calculate the gradient of loss functions of convolutional architectures efficiently, which means that just like in fully connected networks, optimization with SGD related methods is possible.

2.3 Constrained Neural Networks

For both introduced types of neural networks, the value of individual parameters can rise arbitrarily high during the training process. The consequences of this behavior have been analyzed in detail by I.E. Livieris, for example in chapter 1 of Livieris (2018) and chapter 3 of Livieris et al. (2019). He comes to the conclusion that this can lead to few very large individual weights dominating the output of the entire network, meaning that not all input data is sufficiently considered and worse prediction behavior is achieved. He also indicates that large individual weights can indicate a network's instability, with small changes in the input data leading to drastically different outputs. As a solution, he suggests constraining the values the parameters may attain to a convex and compact feasible region to prevent them from becoming too large, and has shown that constrained neural networks can achieve competitive accuracy on classical neural network benchmark datasets (see Livieris (2018), ch. 3) as well as on real world datasets (see Livieris et al. (2019), ch. 5).

Mathematically, the problem of training/optimizing a constrained neural network can be defined as follows:

$$\min_{\theta \in \mathcal{C}} L(\theta) := \mathbb{E}_Z[l(\theta, Z)] \tag{2.1}$$

where θ are the network parameters with range of values in the convex and compact feasible region $\mathcal{C}$ which has an L^2-diameter of D, Z is a random variable with some distribution over the training data $\mathcal{E}$ of the network, $l(\theta, z)$ is a loss function as defined in section 2.1 and L is the (possibly non-convex) overall loss function.

One problem with this approach is that the use of SGD becomes more complex. Since it is an unconstrained optimization algorithm, it can produce solutions during the training process which do not lie within the feasible region. Projecting those back into the feasible region seems like an obvious solution, however, as can be seen in section 3.3 of Combettes and Pokutta (2021), even for seemingly simple regions like the L^p-norm ball, it is not guaranteed that a closed formula for projection exists, which means that training becomes a lot more computationally expensive. A potential solution might be using a different optimization algorithm all together: Algorithms like the Frank-Wolfe method are of similar computational intensity as SGD and, having been designed for constrained optimization, only produce iterates which lie within the feasible region, making projection unnecessary. In the next section, there will be a mathematical prove that the Frank-Wolfe method, under the right assumptions, is indeed a suitable optimizer for constrained neural networks.

3 Frank-Wolfe Method

3.1 Algorithm

The Frank-Wolfe method was first introduced in Frank et al. (1956). It aims to solve a constrained non-linear optimization problem by repeatedly solving constrained linear optimization problems. Its algorithm can be found in Algorithm 1. As $\mathcal{C}$ is convex, the update formula in line 5 guarantees the feasibility of all iterates as long as the parameters are initialized within the feasible region.

Algorithm 1 Frank-Wolfe Method

Require: Loss function to minimize $L(\theta)$, convex and compact feasible region $\mathcal{C}$, initial parameters $\theta_0 \in \mathcal{C}$, number of iterations T, learning rate $\{\gamma_i\}_{i=0}^{T-1}$, where $\gamma_i \in [0,1]\ \forall i \in \{0,\dots,T-1\}$

1: **for** $t = 0$ to $T-1$ **do**
2: Compute gradient $\nabla L(\theta_t)$
3: Solve linear subproblem: $v_t \leftarrow \arg\max_{v_t \in \mathcal{C}} \langle v_t, -\nabla L(\theta_t) \rangle$
4: Compute step direction: $d_t \leftarrow v_t - \theta_t$
5: Update $\theta_{t+1} \leftarrow \theta_t + \gamma_t \cdot d_t$
6: **end for**
7: **return** θ_T

In the case of neural networks, even though backpropagation enables calculation of the gradient of the individual loss function $l(\theta, z)$, the large number of training examples means that it is usually not feasible to calculate the gradient of the overall loss function $L(\theta)$ in every iteration. That is why stochastic versions of classical optimization algorithms are used, in which the gradient of $L(\theta)$ is approximated by the gradients of randomly sampled training examples at every iteration. Applying this leads to the Stochastic Frank-Wolfe (SFW) method, detailed in Algorithm 2. The other change made, not returning the final parameters but instead uniformly sampling from the iterates, enables a formal convergence proof of the SFW method. This is further detailed in section 3.3.

Since this thesis is about the application of the Frank-Wolfe method in neural networks, the stochastic version will be used in the following.

Algorithm 2 Stochastic Frank-Wolfe Method

Require: Individual loss function $l(\theta, z)$, convex and bounded feasible region $\mathcal{C}$, initial parameters $\theta_0 \in \mathcal{C}$, number of iterations T, learning rate $\{\gamma_i\}_{i=0}^{T-1}$, where $\gamma_i \in [0,1]\ \forall i \in \{0,\dots,T-1\}$, minibatch size $\{b_i\}_{i=0}^{T-1}$, random variable Z from which to draw inputs.

1: **for** $t = 0$ to $T-1$ **do**
2: Sample $\{z_1^t, \dots, z_{b_t}^t\}$ according to Z
3: Compute approximate gradient $\widetilde{\nabla} L(\theta_t) \leftarrow \frac{1}{b_t} \sum_{i=1}^{b_t} \nabla l(\theta_t, z_i)$
4: Solve linear subproblem: $v_t \leftarrow \arg\max_{v_t \in \mathcal{C}} \langle v_t, -\widetilde{\nabla} L(\theta_t) \rangle$
5: Compute step direction: $d_t \leftarrow v_t - \theta_t$
6: Update $\theta_{t+1} \leftarrow \theta_t + \gamma_t \cdot d_t$
7: **end for**
8: **return** θ_a drawn uniformly from $\{\theta_0, \dots, \theta_{T-1}\}$

3.2 Convergence Criterion

To give a proof of convergence of SFW for scenario 2.1, two additional assumptions to the loss functions have to be made:

$$\begin{aligned} &\text{- } l \text{ is } G\text{-lipschitz: } \forall \theta \in \mathcal{C}, z \in \mathcal{E} : \|\nabla l(\theta, z)\| \leq G \\ &\text{- } L \text{ is } M\text{-smooth: } \forall \theta, \hat{\theta} \in \mathcal{C} : \|\nabla L(\theta) - \nabla L(\hat{\theta})\| \leq M \cdot \|\theta - \hat{\theta}\| \end{aligned} \tag{3.1}$$

These assumptions are common when proving convergence of stochastic machine learning algorithms, see chapter 2 of Reddi et al. (2016).

Furthermore, a convergence criterion is needed. With loss function L, the usual criterion, $\|\nabla L(\theta)\| = 0$, is not applicable due to the constrained nature of the problem: optimal solutions on the edge of $\mathcal{C}$ do not necessarily have a vanishing gradient of the loss function. Instead, the Frank-Wolfe Gap function

$$G(\theta) = \max_{v \in \mathcal{C}} \langle v - \theta, -\nabla L(\theta) \rangle$$

will be used. In particular, this section will show that $G(\theta) = 0$ is a suitable convergence criterion, and the next will give a proof of convergence of SFW with regards to G.

In the case of a convex loss function, the quality of the criterion is immediately apparent, since $G(\theta)$ gives a direct limit to the sub-optimality of θ, as seen in chapter 2 of Jaggi (2013):

Lemma 3.1. If L is convex and θ^* is an optimal solution, then $G(\theta) \geq L(\theta) - L(\theta^*)$.

Proof. Since L is convex:

$$\begin{aligned}
&\forall \hat{v} \in \mathcal{C} : L(\hat{v}) \geq L(\theta) + \langle \hat{v} - \theta, \nabla L(\theta) \rangle \\
\implies &\forall \hat{v} \in \mathcal{C} : \langle \hat{v} - \theta, -\nabla L(\theta) \rangle \geq L(\theta) - L(\hat{v}) \\
\implies &\forall \hat{v} \in \mathcal{C} : G(\theta) = \max_{v \in \mathcal{C}} \langle v - \theta, -\nabla L(\theta) \rangle \geq \langle \hat{v} - \theta, -\nabla L(\theta) \rangle \geq L(\theta) - L(\hat{v})
\end{aligned}$$

In particular, for $\hat{v} = \theta^*$, the desired inequality is obtained. □

However, in the case of neural network optimization, the loss function is not necessarily convex, which is why other properties of G have to be utilized:

Lemma 3.2. In the setting of 2.1 with assumptions 3.1, the Frank-Wolfe Gap function $G(\theta)$ satisfies the following properties:

1. G is continuous.
2. G is positive, i.e., $G(\theta) \geq 0$ for all $\theta \in \mathcal{C}$.
3. A necessary condition for θ^* to be a local minimizer of $L(\theta)$ over $\mathcal{C}$ is $G(\theta^*) = 0$.

Proof.

1. Since L is M-smooth, ∇L is continuous. Since $\mathcal{C}$ is convex and compact, the maximum theorem applies, making G a continuous function.
2. $\forall \theta \in \mathcal{C} : G(\theta) \geq \langle \theta - \theta, -\nabla L(\theta) \rangle = 0$.
3. The following proof is similar to Satz 16.3 of Ulbrich and Ulbrich (2012). Let θ^* be a local minimizer of L over $\mathcal{C}$. By Taylor-Expansion, if λ is small enough:

$$\begin{aligned}
&\forall \hat{v} \in \mathcal{C} : L(\theta^* + \lambda \cdot (\hat{v} - \theta^*)) = L(\theta^*) + \nabla L(\theta^*)^T (\lambda \cdot (\hat{v} - \theta^*)) + \mathcal{O}(\lambda^2) \\
\implies &\forall \hat{v} \in \mathcal{C} : L(\theta^* + \lambda \cdot (\hat{v} - \theta^*)) - L(\theta^*) = \lambda \cdot \nabla L(\theta^*)^T (\hat{v} - \theta^*) + \mathcal{O}(\lambda^2) \\
\implies &\forall \hat{v} \in \mathcal{C} : \frac{L(\theta^* + \lambda \cdot (\hat{v} - \theta^*)) - L(\theta^*)}{\lambda} = \nabla L(\theta^*)^T (\hat{v} - \theta^*) + \mathcal{O}(\lambda)
\end{aligned}$$

   As $\lambda \to 0$, since θ^* is a local minimizer, the left side of the last line is greater or equal to 0, while $\nabla L(\theta^*)^T (\hat{v} - \theta^*) + \mathcal{O}(\lambda) \to \nabla L(\theta^*)^T (\hat{v} - \theta^*)$.

$$\begin{aligned}
&\implies \forall \hat{v} \in \mathcal{C} : 0 \leq \nabla L(\theta^*)^T (\hat{v} - \theta^*) = -\langle \hat{v} - \theta^*, -\nabla L(\theta^*) \rangle \\
&\implies G(\theta^*) = \max_{\hat{v} \in \mathcal{C}} \langle \hat{v} - \theta^*, -\nabla L(\theta^*) \rangle \leq 0 \implies G(\theta^*) = 0
\end{aligned}$$

□

Lemma 3.2 shows that all properties that make $\|\nabla L(\theta)\| = 0$ a suitable convergence criterion in the unconstrained case apply to $G(\theta) = 0$ in the context of 2.1 with assumptions 3.1, which is why it can and will be used as a convergence criterion in this setting.

3.3 Convergence proof of SFW

Theorem 3.3. *Consider the SFW algorithm as optimizer for 2.1 with assumptions 3.1. Set parameters* $\gamma_t = \gamma = \mathcal{O}(\frac{1}{\sqrt{T}}), b_t = b = T \ \forall t \in \{0, \dots, T-1$. *Then,* $\mathbb{E}[G(\theta_a)] = \mathcal{O}(\frac{1}{\sqrt{T}})$.

The upcoming proof of this theorem builds upon and details Theorem 2 of Reddi et al. (2016). To simplify the structure, two lemmata are introduced, which will then be used in the proof of Theorem 3.3.

Lemma 3.4. Assume that L is M-smooth. Then:

$$L(\theta_{t+1}) \leq L(\theta_t) + \langle \nabla L(\theta_t), \theta_{t+1} - \theta_t \rangle + \frac{M}{2} \|\theta_{t+1} - \theta_t\|^2$$

Proof. The proof is obtained by applying the fundamental theorem of calculus to the differentiable function L:

$$L(\theta_{t+1}) = L(\theta_t) + \int_0^1 (\nabla L(\theta_t + x \cdot (\theta_{t+1} - \theta_t)))^T (\theta_{t+1} - \theta_t)\, dx$$

$$= L(\theta_t) + \nabla L(\theta_t))^T (\theta_{t+1} - \theta_t) + \int_0^1 (\nabla L(\theta_t + x \cdot (\theta_{t+1} - \theta_t)) - \nabla L(\theta_t))^T (\theta_{t+1} - \theta_t)\, dx$$

Limiting the value of the integral by using first the Cauchy-Schwarz inequality and then the M-smoothness of L gives the desired inequality:

$$\begin{aligned}
&\int_0^1 (\nabla L(\theta_t + x \cdot (\theta_{t+1} - \theta_t)) - \nabla L(\theta_t))^T (\theta_{t+1} - \theta_t)\, dx \\
\leq &\int_0^1 \|\nabla L(\theta_t + x \cdot (\theta_{t+1} - \theta_t)) - \nabla L(\theta_t)\| \|\theta_{t+1} - \theta_t\|\, dx \\
\leq &\int_0^1 M \cdot \|x \cdot (\theta_{t+1} - \theta_t)\| \|\theta_{t+1} - \theta_t\|\, dx \\
= &M \cdot \|\theta_{t+1} - \theta_t\|^2 \cdot \int_0^1 x\, dx \ = \frac{M}{2} \|\theta_{t+1} - \theta_t\|^2
\end{aligned}$$

□

The next lemma is particularly important, as it gives a way to limit the distance between the approximate gradient calculated in line 3 of the SFW method 2 and the actual gradient:

Lemma 3.5. Consider setting 2.1 with assumptions 3.1. It holds:

$$\forall \theta \in \mathcal{C} : \mathbb{E}[\|\widetilde{\nabla} L(\theta) - \nabla L(\theta)\|] \leq \frac{G}{\sqrt{b}}$$

Proof. Recall that $\widetilde{\nabla} L(\theta) = \frac{1}{b}\sum_{i=1}^{b} \nabla l(\theta, z_i)$, where the $\{z_i\}_{i=1}^{b}$ are drawn i.i.d. along the distribution of Z. Therefore:

$$\widetilde{\nabla} L(\theta) - \nabla L(\theta) = \sum_{i=1}^{b} \frac{1}{b}(\nabla l(\theta, z_i)) - \nabla L(\theta)) := \sum_{i=1}^{b} Z_i$$

Since the $\{z_i\}_{i=1}^{b}$ are drawn i.i.d., the Z_i are independent, and by definition of L, they have an expected value of 0. This makes it possible to treat each z_i individually:

$$\mathbb{E}[\|\widetilde{\nabla} L(\theta) - \nabla L(\theta)\|^2] = \mathbb{E}\left[\left\|\sum_{i=1}^{b} Z_i\right\|^2\right] = \sum_{i,j=1}^{b} \mathbb{E}[Z_i \cdot Z_j]$$

$$= \sum_{i=1}^{b} \mathbb{E}[\|Z_i\|^2] + \sum_{i,j=1, i\neq j}^{b} \mathbb{E}[Z_i]\cdot\mathbb{E}[Z_j] = \frac{1}{b^2}\sum_{i=1}^{b} \mathbb{E}[\|\nabla l(\theta, z_i)) - \nabla L(\theta)\|^2]$$

The expected value can now be limited from above by utilizing the fact that l is G-lipschitz:

$$\begin{aligned}
&\mathbb{E}\left[\|\nabla l(\theta, z_i) - \nabla L(\theta)\|^2\right] = \mathbb{E}\left[\|\nabla l(\theta, z_i)\|^2 - 2\cdot\nabla l(\theta, z_i)^T \nabla L(\theta) + \|\nabla L(\theta)\|^2\right] \\
=&\mathbb{E}\left[\|\nabla l(\theta, z_i)\|^2\right] - 2\cdot\mathbb{E}\left[\nabla l(\theta, z_i)\right]^T \nabla L(\theta) + \mathbb{E}\left[\|\nabla L(\theta)\|^2\right] \\
=&\mathbb{E}\left[\|\nabla l(\theta, z_i)\|^2\right] - \mathbb{E}\left[\|\nabla L(\theta)\|^2\right] \leq \mathbb{E}\left[\|\nabla l(\theta, z_i)\|^2\right] \leq G^2
\end{aligned}$$

Plugging this back into the equation above gives:

$$\mathbb{E}[\|\widetilde{\nabla} L(\theta) - \nabla L(\theta)\|^2] = \frac{1}{b^2}\sum_{i=1}^{b} \mathbb{E}[\|\nabla l(\theta, z_i)) - \nabla L(\theta)\|^2] \leq \frac{G^2}{b}$$

Applying Jensen's inequality with the concave function $f(x) = \sqrt{x}$ and then using f being monotonically increasing delivers the asserted inequality:

$$\begin{aligned}
&\mathbb{E}[\|\widetilde{\nabla} L(\theta) - \nabla L(\theta)\|] = \mathbb{E}[f(\|\widetilde{\nabla} L(\theta) - \nabla L(\theta)\|^2)] \\
&\leq f(\mathbb{E}[\|\widetilde{\nabla} L(\theta) - \nabla L(\theta)\|^2]) \leq f(\frac{G^2}{b}) = \frac{G}{\sqrt{b}}
\end{aligned}$$

□

Proof of Theorem 3.3. Using Lemma 3.4 on L, plugging in the definition of θ_{t+1} according to algorithm 2 and then using the diameter D of $\mathcal{C}$ gives:

$$\begin{aligned}
L(\theta_{t+1}) &\leq L(\theta_t) + \langle \nabla L(\theta_t), \theta_{t+1} - \theta_t\rangle + \frac{M}{2}\|\theta_{t+1} - \theta_t\|^2 \\
&= L(\theta_t) + \langle \nabla L(\theta_t), \gamma\cdot(v_t - \theta_t)\rangle + \frac{M}{2}\|\gamma\cdot(v_t - \theta_t)\|^2 \\
&\leq L(\theta_t) + \gamma\cdot\langle \nabla L(\theta_t), v_t - \theta_t\rangle + \frac{M\cdot D^2\cdot\gamma^2}{2}
\end{aligned}$$

To process the value of the scalar product, $\hat{v_t} = \arg\max_{v\in\mathcal{C}}\langle v, -\nabla L(\theta_t)\rangle$ is needed. There are two important observations for the next steps:

- By line 4 of algorithm 2, $v_t \in \arg\min_{v \in \mathcal{C}} \langle \widetilde{\nabla} L(\theta_t), v \rangle$
- By definition of $\hat{v}_t$, $G(\theta_t) = \max_{v \in \mathcal{C}} \langle v - \theta_t, -\nabla L(\theta_t) \rangle = \langle \hat{v}_t - \theta_t, -\nabla L(\theta_t) \rangle$

These two observations lead to the following chain of (in-)equalities, with the last inequality being obtained by an application of the Cauchy-Schwarz inequality and again utilizing the diameter of $\mathcal{C}$:

$$\begin{aligned}
\langle \nabla L(\theta_t), v_t - \theta_t \rangle &= \langle \nabla L(\theta_t), v_t - \theta_t + \hat{v}_t - \hat{v}_t \rangle + \langle \widetilde{\nabla} L(\theta_t), v_t - v_t \rangle \\
&\leq \langle \nabla L(\theta_t), \hat{v}_t - \theta_t \rangle + \langle \nabla L(\theta_t), v_t - \hat{v}_t \rangle + \langle \widetilde{\nabla} L(\theta_t), \hat{v}_t - v_t \rangle \\
&= -G(\theta_t) + \langle \nabla L(\theta_t) - \widetilde{\nabla} L(\theta_t), v_t - \hat{v}_t \rangle \\
&\leq -G(\theta_t) + \| L(\theta_t) - \widetilde{\nabla} L(\theta_t) \| \cdot D
\end{aligned}$$

Plugging this back into the original sum and applying the expected value enables the usage of lemma 3.5:

$$\begin{aligned}
\mathbb{E}[L(\theta_{t+1})] &\leq \mathbb{E}[L(\theta_t)] - \gamma \cdot \mathbb{E}[G(\theta_t)] + \gamma \cdot D \cdot \mathbb{E}[\| L(\theta_t) - \widetilde{\nabla} L(\theta_t) \|] + \frac{M \cdot D^2 \cdot \gamma^2}{2} \\
&\leq \mathbb{E}[L(\theta_{t+1})] - \gamma \cdot \mathbb{E}[G(\theta_t)] + \frac{\gamma \cdot D \cdot G}{\sqrt{b}} + \frac{M \cdot D^2 \cdot \gamma^2}{2}
\end{aligned}$$

The inequality can now be rearranged to isolate $\mathbb{E}[G(\theta_t)]$, after which a telescope sum can be applied. Considering an optimal solution θ^*, which exists for the continuous function L over the convex and compact set $\mathcal{C}$, the value of $\mathbb{E}[L(\theta_T)]$ can be limited from above:

$$\begin{aligned}
\mathbb{E}[G(\theta_t)] &\leq \frac{\mathbb{E}[L(\theta_t)] - \mathbb{E}[L(\theta_{t+1})]}{\gamma} + \frac{D \cdot G}{\sqrt{b}} + \frac{M \cdot D^2 \cdot \gamma}{2} \\
\implies \sum_{t=0}^{T-1} \mathbb{E}[G(\theta_t)] &\leq \frac{L(\theta_0) - \mathbb{E}[L(\theta_T)]}{\gamma} + \frac{T \cdot D \cdot G}{\sqrt{b}} + \frac{T \cdot M \cdot D^2 \cdot \gamma}{2} \\
&\leq \frac{L(\theta_0) - L(\theta^*)}{\gamma} + \frac{T \cdot D \cdot G}{\sqrt{b}} + \frac{T \cdot M \cdot D^2 \cdot \gamma}{2}
\end{aligned}$$

Since a is drawn uniformly from all iterates but the last, $\mathbb{E}[G(\theta_a)]$ is the average of $\{\mathbb{E}[G(\theta_0)], \ldots, \mathbb{E}[G(\theta_{T-1})]\}$. Therefore:

$$\mathbb{E}[G(\theta_a)] = \frac{\sum_{t=0}^{T-1} \mathbb{E}[G(\theta_t)]}{T} \leq \frac{L(\theta_0) - L(\theta^*)}{\gamma \cdot T} + \frac{D \cdot G}{\sqrt{b}} + \frac{M \cdot D^2 \cdot \gamma}{2}$$

The proof is completed by remembering the values of the parameters: With $\gamma = \mathcal{O}(\frac{1}{\sqrt{T}})$ and $b = T$, $\mathbb{E}[G(\theta_a)] = \mathcal{O}(\frac{1}{\sqrt{T}})$ is obtained. □

3.4 Modifications to the Algorithm

Theorem 3.3 gives theoretical justification to the usage of SFW as constrained neural network optimizer. In practice, there are several common modifications for machine learning algorithms which are not theoretically proven to conserve convergence properties or improve performance, but empirically tend to do well. Some of these modifications are introduced in the modified SFW algorithm (MSFW) 3.

The first modification is rather obvious: In practice, it is expected that the iterates get closer to an optimal solution during the training process, which is why MSFW always returns the last iterate.
Secondly, a typical modification for machine learning algorithms is introducing momentum. This means that the gradient of the loss function is not fully reset in each iteration, but instead replaced by a convex sum of itself and the current momentum vector. That way, in every iteration, information of the gradients of all previous iterations is utilized. If only this modification is introduced and the loss function is convex, convergence can still be proven, see Theorem 4.12 of Braun et al. (2022). The modification is reflected in lines 4-5 of the new algorithm.
Lastly, the update vector d_t is not necessarily of the same magnitude as the gradient. Thus, the rate of learning might be influenced less by how good the current parameters already are, and more by the diameter of the feasible region $\mathcal{C}$. To fix this, the update vector will be rescaled to be proportional to the gradient, as seen in section 3.1 of Pokutta et al. (2020). It is important to not let the factor multiplied to the update be bigger than 1, as then it would no longer be a convex update and feasibility of the iterates would not be guaranteed anymore. This modification is reflected in line 7 of the new algorithm.

Algorithm 3 Modified Stochastic Frank-Wolfe Method

Require: Same requirements as algorithm 2 and momentum $\{\rho\}_{i=0}^{T-1}$, where $\rho_i \in [0,1]\ \forall i \in \{0,\dots,T-1\}$.

1: **for** $t = 0$ to $T-1$ **do**
2: Sample $\{z_1^t,\dots,z_{b_t}^t\}$ according to Z
3: Compute approximate gradient $\widetilde{\nabla} L(\theta_t) \leftarrow \frac{1}{b_t}\sum_{i=1}^{b_t} \nabla l(\theta_t, z_i)$
4: Compute current momentum vector $m_t \leftarrow (1-\rho_t)\cdot m_{t-1} + \rho_t \cdot \widetilde{\nabla} L(\theta_t)$
5: Solve linear subproblem: $v_t \leftarrow \arg\max_{v_t \in \mathcal{C}} \langle v_t, -m_t\rangle$
6: Compute step direction: $d_t \leftarrow v_t - \theta_t$
7: Update $\theta_{t+1} \leftarrow \theta_t + \min(\frac{\gamma_t \|\widetilde{\nabla} L(\theta_t)\|}{\|d_t\|}, 1)\cdot d_t$
8: **end for**
9: **return** θ_T

In section 5.2, the performance of SFW and MSFW will be compared, and the better performing algorithm will be utilized for subsequent experiments.

4 Feasible Regions

To ensure computational efficiency of (M)SFW on any feasible region $\mathcal{C}$, it is important that the linear subproblem $\arg\max_{v\in\mathcal{C}}\langle v, -x\rangle$ can be calculated without too much effort in each iteration. This will be achieved by providing a 'Linear Maximization Oracle' (LMO) for each introduced feasible region, which gives a closed formula solution to the Frank-Wolfe subproblem for all possible inputs. With such an LMO, the cost of one iteration of (M)SFW is comparable to that of SGD in the unconstrained setting.

For the LMOs, two notations need to be introduced: x_G is x with only the elements corresponding to the indices in the set G not set to 0, and $[\text{top } K]$ as subscript of a vector x is the set of indices of the K largest elements of x.

4.1 L^p-norm ball constraint

A simple way to obtain a convex and compact set is constraining the L^p-norm ($p \in [1, \infty]$) the set of parameters may reach: $\mathcal{B}_p(\tau) = \{v \in \mathbb{R}^n : \|v\|_p \leq \tau\}$ for some $\tau \in \mathbb{R}$.

Lemma 4.1. An LMO for $\mathcal{B}_p(\tau)$ is given by:

1. $p = 1$: $\arg\max_{v\in\mathcal{B}_1(\tau)}\langle v, -x\rangle = -\tau \cdot \operatorname{sgn}(x_{[\text{top } 1]})$
2. $p \in (1, \infty)$: $\arg\max_{v\in\mathcal{B}_p(\tau)}\langle v, -x\rangle = -\tau \cdot \operatorname{sgn}(x) \cdot \frac{|x|^{q/p}}{\|x\|_q^{q/p}}$, where $q = \frac{p}{p-1}$
3. $p = \infty$: $\arg\max_{v\in\mathcal{B}_\infty(\tau)}\langle v, -x\rangle = -\tau \cdot \operatorname{sgn}(x)$

Proof.

1. By definition of the L^1-norm: $v \in \mathcal{B}_1(\tau) \iff \sum_{i=1}^n |x_i| \leq \tau$. As $-\tau \cdot \operatorname{sgn}(x_{[\text{top } 1]})$ has only one non-zero entry which has absolute value τ, it is within the feasible region. What remains is proving the optimality:

$$\begin{aligned}\forall v \in \mathcal{B}_1(\tau) : \langle v, -x\rangle = \sum_{i=1}^n -v_i \cdot x_i \leq \sum_{i=1}^n |v_i| \cdot \max_{i\in\{1,\dots,n\}} |x_i| \\ \leq \tau \cdot \max_{i\in\{1,\dots,n\}} |x_i| = \langle -\tau \cdot \operatorname{sgn}(x_{[\text{top } 1]}), -x\rangle\end{aligned}$$

2. As seen in section 4.1 of Jaggi (2013), the solution is proportional to $-\text{sgn}(x) \cdot |x|^{q/p}$. It remains to verify that the given vector lies within $\mathcal{B}_p(\tau)$ and has maximum magnitude, i.e. its L^p-norm is exactly τ. To do so, the following common transformation from L^p-norm to L^q-norm is necessary:

$$\| \, |x|^{q/p} \, \|_p = \left(\sum_{i=1}^{n} \left(|x_i|^{q/p} \right)^p \right)^{1/p} = \left(\sum_{i=1}^{n} |x_i|^q \right)^{1/p} = \left(\left(\sum_{i=1}^{n} |x_i|^q \right)^{1/q} \right)^{q \cdot 1/p} = \|x\|_q^{q/p}$$

This can now be plugged into the calculation of the L^p-norm of the entire expression to obtain the wanted result:

$$\left\| -\tau \cdot \text{sgn}(x) \cdot \frac{|x|^{q/p}}{\|x\|_q^{q/p}} \right\|_p = \tau \cdot \frac{\| \, |x|^{q/p} \, \|_p}{\|x\|_q^{q/p}} = \tau \cdot \frac{\|x\|_q^{q/p}}{\|x\|_q^{q/p}} = \tau$$

3. By definition of the L^∞-norm: $v \in \mathcal{B}_1(\tau) \iff \max_{i \in \{1,\dots,n\}} |x_i| \leq \tau$. As all elements of $-\tau \cdot \text{sgn}(x)$ have an absolute value of τ, it lies within the feasible region. It again remains to verify the optimality:

$$\forall v \in \mathcal{B}_\infty(\tau) : \langle v, -x \rangle = \sum_{i=1}^{n} -v_i \cdot x_i \leq \sum_{i=1}^{n} \tau \cdot |x_i| = \langle -\tau \cdot \text{sgn}(x), -x \rangle$$

□

To bring lemma 4.1 back into the context of the Frank-Wolfe method and neural networks, it is important to remember what each vector represents: In the algorithm, x is the gradient/momentum vector of the loss function, while the returned vector is the direction in which the current parameters will be updated.

Therefore, for $p = 1$, only one parameter will receive an active update: the parameter corresponding to the entry with highest magnitude in the gradient, i.e. the parameter with which the loss can be decreased the most - the most important parameter. Since the new parameters are a convex combination of the old parameters and the update vector, the absolute value of all other parameters will get closer to 0. The higher p gets, the less this applies, until for $p = \infty$ all parameters receive an update of the same magnitude.

4.2 K-sparse polytope constraint

For the creation of sparse, pruning-friendly neural networks, the observations of the previous section for $p = 1$ are very enticing. However, updating just one parameter out of millions or even billions in every iteration may lead to inefficiency due to too slow convergence in the training process. Therefore, a feasible region which abstracts the

updates of the L^1-norm ball to more than one parameter is needed.

For the construction of the feasible region, it is important to remember the basics of linear optimization: When a linear objective function is optimized over a convex and compact set (as in the Frank-Wolfe subproblem), an optimal solution can always be found in one of the set's vertices. Therefore, if all vertices of a feasible region had only K non-zero entries, then in every iteration of the Frank-Wolfe method only K parameters would be updated, while the rest would be brought closer to 0.

These thoughts in section 3.2 of Pokutta et al. (2020) lead to the utilization of the K-sparse polytope $\mathcal{S}^K$ of radius τ, which is the convex hull of all vectors with K non-zero entries, all of which are $\pm\tau$.

Lemma 4.2. An LMO for $\mathcal{S}^K(\tau)$ is given by:

$$\arg\max_{v \in \mathcal{S}^K(\tau)} \langle v, -x \rangle = -\tau \cdot \operatorname{sgn}(x_{[\text{top } K]})$$

Proof. As one of the feasible regions vertices with K non-zero entries of magnitude τ, the returned vector is obviously feasible. Let the set of vertices of $\mathcal{S}^K$ be denoted by $\mathcal{S}^K_{\text{v}}$. Then:

$$\begin{aligned} &\max_{v \in \mathcal{S}^K} \langle v, -x \rangle = \max_{v \in \mathcal{S}^K_{\text{v}}} \langle v, -x \rangle = \max_{v \in \mathcal{S}^K_{\text{v}}} \sum_{i=1}^{n} v_i \cdot x_i \\ \leq &\sum_{i:[x_{[\textbf{top } K]}]_i \neq 0} \tau \cdot |x_i| = \langle -\tau \cdot \operatorname{sgn}(x_{[\text{top } K]}), -x \rangle \end{aligned}$$

□

Lemma 4.2 shows that the K-sparse polytope in combination with the Frank-Wolfe method successfully abstract the L^1-norm ball: In every iteration, the K most important parameters are updated, while all other parameters are pushed closer to 0.

4.3 K-support norm ball constraint

One drawback of the K-sparse polytope is clear: The K most important parameters are all updated with the same magnitude, somewhat similar to the updates for $\mathcal{B}_\infty(\tau)$. A natural improvement would be to adjust the K updates to have magnitude proportional to their entries in the gradient/momentum vector. As will be seen in the LMO, this can be achieved by using the K-support norm ball as constraint. It was introduced in Argyriou et al. (2012) and is defined as $\mathcal{C}^K(\tau) = \text{conv}\{v \in \mathbb{R}^n : \|v\|_0 \leq K, \|v\|_2 \leq \tau\}$, where $\|v\|_0$ is the number of non-zero entries in the vector v.

Lemma 4.3. An LMO for $\mathcal{C}^K(\tau)$ is given by:

$$\arg\max_{v \in \mathcal{C}^K(\tau)} \langle v, -x \rangle = -\tau \cdot \frac{x_{[\text{top } K]}}{\|x_{[\text{top } K]}\|_2}$$

Proof. The following proof is similar to the one of Lemma A.1 in Zimmer et al. (2022): By construction, the vertices of $\mathcal{C}^K(\tau)$ satisfy $\|v\|_2 = \tau, \|v\|_0 \leq K$ and the returned vector is one of them. Defining $\mathcal{C}_v^K(\tau)$ as the set of vertices and using the geometric definition of the dot product, the following holds:

$$\arg\max_{v \in \mathcal{C}^K(\tau)} \langle v, -x \rangle = \arg\max_{v \in \mathcal{C}_v^K(\tau)} \langle v, -x \rangle =$$
$$\arg\max_{v \in \mathcal{C}_v^K(\tau)} \|v\|_2 \cdot \|x\|_2 \cdot \cos\left(\angle\left(v, -x\right)\right) = \arg\max_{v \in \mathcal{C}_v^K(\tau)} \tau \cdot \|x\|_2 \cdot \cos\left(\angle\left(v, -x\right)\right)$$

An optimal solution is therefore the vertex whose angle is most similar to that of $-x$, which is precisely the returned vector. □

4.4 Group K-sparse polytope constraint

Structured pruning of convolutional layers works by removing entire convolutional filters as a group. Therefore, the approach of sections 4.2 and 4.3 does not perfectly translate from unstructured pruning-friendliness to structured pruning-friendliness: Even though only the most important weights are optimized, these are likely still spread across all filters, meaning that the optimization is not focused on the weights which remain after the pruning process. Instead, new constraints are needed which abstract the logic of the previous two norms to the filter level and, in every iteration, actively update exactly the K most important filters entirely, and no other weights.

To adapt the logic of the K-sparse polytope, given a disjoint partition of the parameter space $\mathcal{G} = \bigsqcup_{i=1}^{k} G_i = \{1, \ldots, n\}$ and a radius τ, the Group K-sparse polytope $\mathcal{S}_{\mathcal{G}}^K$ of radius τ is defined as the convex hull of all vectors which are $\pm\tau$ for the entries of K sets in $\mathcal{G}$, and 0 in all other entries.

Lemma 4.4. An LMO for $\mathcal{S}_{\mathcal{G}}^K(\tau)$ is given by:

$$\arg\max_{\mathcal{S}_{\mathcal{G}}^K(\tau)} \langle v, -x \rangle = -\tau \cdot \operatorname{sgn}\left(x_{\bigsqcup_{i=1}^{K} G_{K_i}}\right)$$

where $\bigsqcup_{i=1}^{K} G_{K_i}$ are the K sets of $\mathcal{G}$ with maximum $\|x_{G_i}\|_1$.

Proof. The proof works the same way as the proof for the LMO of the K-sparse polytope, only with the new set of vertices. □

In the context of convolutional neural networks, the weights of each convolutional filter would be contained in one of the sets. Lemma 4.4 shows that with such a partition, in every iteration the weights of the K most important filters would be updated, while the weights of all other filters get closer to 0, preparing for structural pruning.

4.5 Group K-support norm ball constraint

To adapt the logic of the K-support norm to convolutional networks, the Group K-support norm will be used, which was introduced in Rao et al. (2017). Given a partition $\mathcal{G}$ like in 4.4, it is defined as $\mathcal{C}_{\mathcal{G}}^{K}(\tau) = \text{conv}\{v \in \mathbb{R}^n : \|v\|_{0,\mathcal{G}} \leq K, \|v\|_2 \leq \tau\}$ where $\|v\|_{0,\mathcal{G}}$ is the number of groups in $\mathcal{G}$ that have a corresponding entry in v not equal to 0.

Lemma 4.5. An LMO for $\mathcal{C}_{\mathcal{G}}^{K}(\tau)$ is given by:

$$\arg\max_{\mathcal{C}_{\mathcal{G}}^{K}(\tau)} \langle v, -x \rangle = -\tau \cdot \frac{x_{\bigsqcup_{i=1}^{K} G_{K_i}}}{\|x_{\bigsqcup_{i=1}^{K} G_{K_i}}\|_2}$$

where $\bigsqcup_{i=1}^{K} G_{K_i}$ are the K sets of $\mathcal{G}$ with maximum $\|x_{G_i}\|_2$.

Proof. The proof works the same way as the proof for the LMO of the K-support norm ball, only with the new set of vertices. □

5 Computational Experiments

5.1 Experimental Setup

The entire implementation can be found at https://github.com/LinusFilbry/BachelorThesis. It was done using the Pytorch Neural Network framework and loosely follows the experiments found in chapter 4 of Pokutta et al. (2020), section 5.2 of Lu et al. (2022) and sections 3.1-3.2 of Zimmer et al. (2022).

Excluding the visualization of weights in figure 5.4, all results are averaged over three seeds. All networks are trained and tested on image classification, with the accuracy describing the percentage of images of the test dataset correctly classified. To compare the results of (M)SFW with different constraints to more conventional setups, all networks are also trained using SGD with and without weight decay, using a learning rate of 0.01, momentum of 0.1 and weight decay of 0 and 0.01 respectively. Cross entropy loss is used as loss function for all networks, and the batch size is 64 for all datasets.

For an efficient implementation, it works best to apply separate constraints on each set of weights and each set of biases. It is therefore feasible to apply different types of constraints to each layer, or leaving some layers constrained and some unconstrained (an example of the latter can be found in chapters 3-4 of Grigas et al. (2019)). However, since the goal of this thesis is to demonstrate the effectiveness of (M)SFW as neural network optimizer as well as the effects of specific feasible regions on sparseness and pruning-friendliness, the same type of constraint will be applied to all weights and biases.

Following this logic, instead of a universal radius τ, a distinct radius for each set of weights and biases can be chosen. Since ReLu is the activation function for all used networks, the parameters of linear and convolutional layers are initialized using He Normal Initialization (see He et al. (2015), sec. 2.2), projecting back into the feasible region afterwards if necessary. As the parameters are initialized according to a normal distribution, the approach in section 3.2 of Pokutta et al. (2020) is applied, i.e. τ is set such that the L^2-diameter of each constraint is a multiple of the expected L^2-norm after initialization: $D = 2 \cdot w \cdot \mathbb{E}[\|x\|_2^{\text{init}}]$, where w is a hyperparameter of the constraint.

5.2 Comparison of SFW and MSFW

To compare SFW and MSFW, a simple fully connected network and a more complex convolutional network are used. The linear network has two hidden layers, the first with

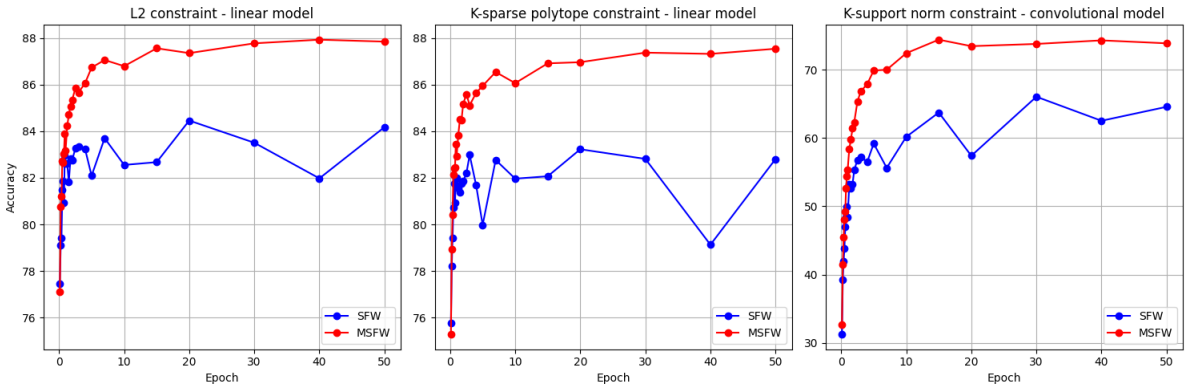


Figure 5.1: Prediction accuracy on test data for SFW and MSFW at various stages of training. All constraints were initialized with $w = 15$ and K as 15% of the parameters of each layer.

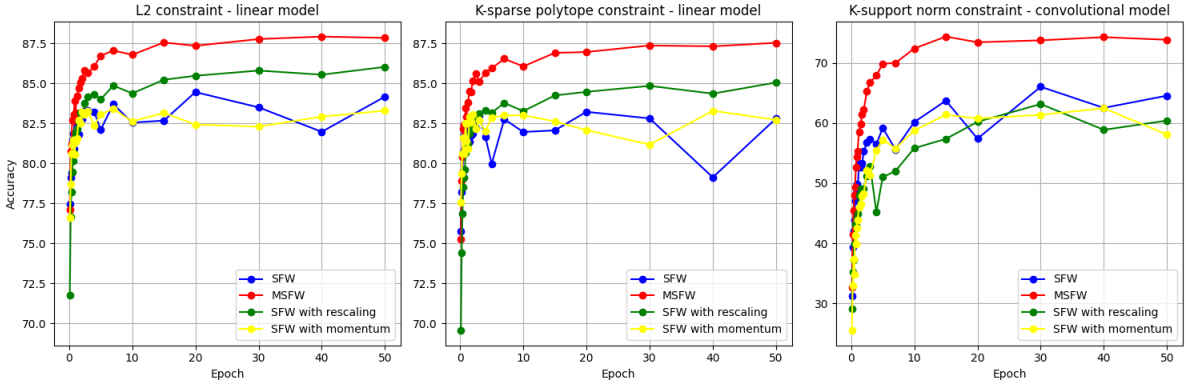


Figure 5.2: Figure 5.1 extended by other algorithms. SFW with rescaling was executed with a learning rate of 0.01, while SFW with momentum was executed with a learning rate of 0.001.

64 and the second with 32 neurons. It is applied to the Fashion MNIST dataset, which contains images of 10 different classes, each of which has 28x28 pixels of grayscale values [Xiao et al. (2017), ch. 2]. The convolutional network has five hidden layers. First, there are three convolutional layers, with the first having 32 and the other two having 64 convolutional filters. All filters have a width and height of three. Each convolutional layer is followed by a 2x2 max pooling layer. Then, there are two fully connected layers, the first with 128 and the second with 64 neurons. It is applied to the CIFAR10 dataset, which consists of images of also 10 different classes, each of which has 32x32 pixels of RBG values [Krizhevsky et al. (2009), sec. 3.1]. SFW and MSFW are used to train both networks on multiple different constraints for 50 epochs. The learning rate of SFW is fixed at 0.001, and the learning rate of MSFW is fixed at 0.01 with its momentum being 0.1. To enable a better comparison, the final parameters are used for testing also for the SFW method, which was generally observed as a boost to the performance of SFW.

	Acc: SFW - Linear	Acc: MSFW - Linear	Acc: SFW - Conv	Acc: MSFW - Conv
L1	32.75%	68.95%	16.47%	20.95%
L2	84.18%	87.85%	61.03%	76.27%
L5	82.74%	87.61%	57.09%	75.59%
K-sparse polytope	82.81%	87.54%	60.21%	74.19%
K-support	82.91%	86.70%	64.56%	73.85%
Group K-sparse polytope	–	–	42.26%	67.40%
Group K-support	–	–	54.98%	72.65%

Figure 5.3: Final accuracy of SFW and MSFW on different constraints and networks. All constraints were initialized with $w = 15$ and K as 15% of the parameters of each layer. Linear models were not trained with the group constraints as they are only different from their non-grouped form on convolutional layers.

Figure 5.1 shows the accuracy of both methods during the training process using two different constraints on the linear network and one constraint on the convolutional network. It is clear that not only does MSFW achieve a higher final accuracy in all three scenarios, the convergence is also more smooth and stable. Not visible in the graph is the increased stability of MSFW also with regard to hyperparameter tuning. For example, the learning rate of 0.001 for SFW is chosen very specifically: A higher learning rate like 0.01 leads to the network not converging anymore and staying at 10% accuracy (i.e. random guesses). For MSFW, a wide range of learning rates and constraint diameters lead to acceptable results.

To find out which of the changes in MSFW achieve this result, the networks are also trained using SFW modified by only gradient rescaling and only the introduction of momentum. The results are displayed in figure 5.2. The green curve shows that, in particular for the simple linear network, gradient rescaling is responsible for the smooth convergence of MSFW in comparison to SFW. It also adds resistance against hyperparameter changes, which is why training can happen with a learning rate of 0.01, leading to higher final accuracy scores in the linear models. Adding the momentum vector without rescaling, however, increased neither stability nor accuracy consistently. A too high learning rate leads to the same kind of unpredictable behavior as regular SFW. Only the combination of both adjustments in MSFW leads to stable convergence to a higher accuracy score, regardless of the learning rate.

To show that the observed trend of MSFW performing better than SFW can be generalized, figure 5.3 shows accuracy results of both algorithms on various constraints and both networks. For every single combination, MSFW performs better than SFW. Combining this finding with the increased stability both in the training process and with regard to different hyperparameters, the ease with which the changes can be implemented and the fact that both algorithms are of similar computational intensity, there is no good reason not to use MSFW instead of SFW. In the next sections, all constrained networks will therefore be optimized using MSFW, with learning rate and momentum staying at 0.01 and 0.1 respectively.

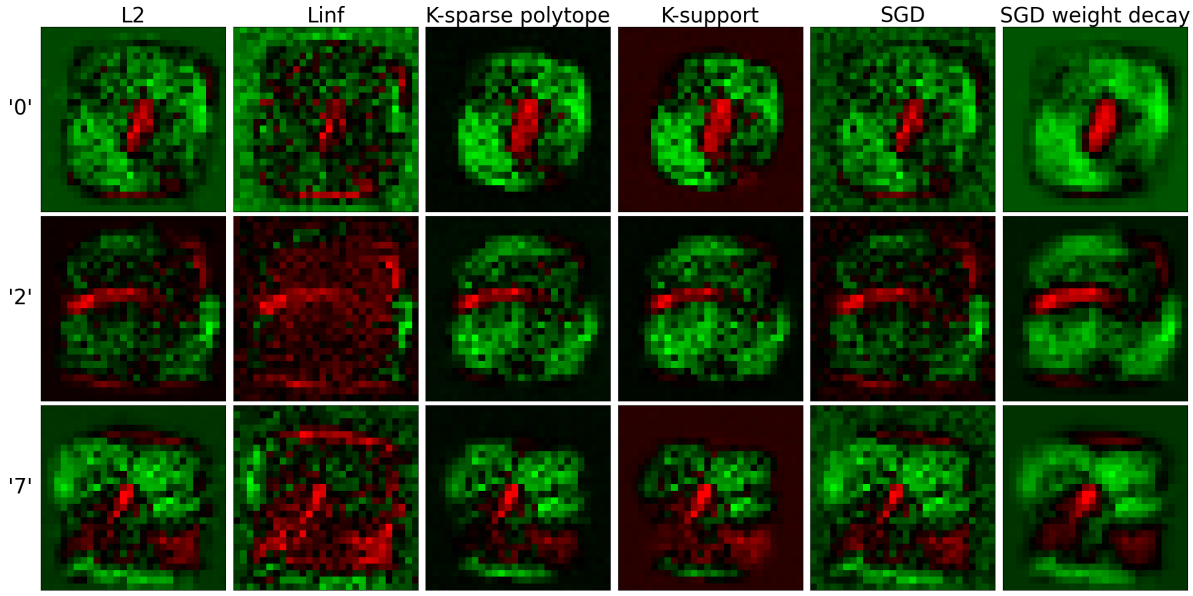


Figure 5.4: Visualization of the weights of a trained linear regression model. Each square is a heatmap of the weights connected to the output neuron of its row when training with the constraint/optimizer of its column. Green represents positive values, red negative values and black values close to 0. Constraints were initialized with $w = 50$ and K as 10% of the parameters of each layer. Training was done for 50 epochs.

Beyond the comparison of SFW and MSFW, figure 5.3 enables interesting observations both about the competitiveness of MSFW as Neural Network optimizer and the different feasible regions.
To judge the quality of MSFW, the accuracy of a traditional algorithm on the same (unconstrained) networks and datasets is needed. The final (averaged) accuracy when using SGD as optimizer was 87.80% on the linear model and 71.10% on the convolutional model. This shows that MSFW is competitive, as it comes close to the accuracy of SGD in the linear case and even surpasses it in the case of the more complex convolutional network for multiple constraints.
Regarding the different constraints, the table shows that the K-support norm ball does not produce better results than the K-sparse polytope in general, despite incorporating more information in its update vector. It is however clear that the Group constraints perform worse than their non-grouped forms. This is logical as currently, no weights or filters are pruned, which means that there is no clear benefit in focusing optimization on the most important filters instead of the most important weights. Finally, the performance of the L^1-norm ball constraint shows that the considerations in chapter 4 were valid: Updating just one parameter in every iteration, especially in more complex networks, leads to much slower convergence.

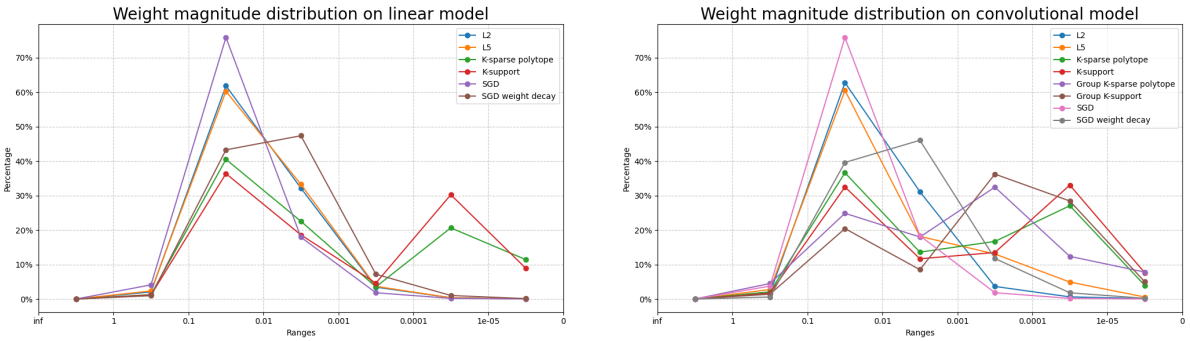


Figure 5.5: Weight magnitude distributions for different training methods. A point in the middle of two values represents the percentage of weights whose absolute value lies between those two bounds after training has taken place. Constraints were initialized with $w = 15$ and K as 15% of each layers parameters. Training was done for 50 epochs.

5.3 Weight visualizations

As figure 5.3 showed, the type of constraint used in combination with MSFW makes a difference in the quality of the model, as training with some constraints leads to higher accuracy than training with others. However, the exact impacts on the weights are unclear. To gain a better understanding, a fully connected network with no hidden layers is trained on the MNIST dataset, which is a classification dataset containing hand-written numbers from 0 to 9 represented by 28x28 pixels of grayscale values [Deng (2012), sec. DATA]. Since there is no hidden layer and therefore no non-linear activation function, the network is nothing more than linear regression, allowing for a better human understanding of the learned representation.

Figure 5.4 visualizes the weights connected to different output neurons when training with different constraints and optimizers. Despite obtaining similar final accuracies on the test data (from left to right: 92.36%, 92.67%, 91.90%, 91.78%, 92.53%, 91.07%), the form of the weights substantially differs from constraint to constraint. In particular, the theoretical considerations of section 4 seem to pay off: For the K-sparse polytope and the K-support norm ball constraint, the unimportant weights at the edges are a lot closer to 0 than for the other constraints, or for unconstrained optimization with SGD. The effects of weight decay in SGD can also be observed: In comparison to regular SGD, the weights at the edges are lower and more blurry, which represents their uniformly lower magnitude, although to a lesser degree than for the two mentioned constraints.

5.4 Parameter and Filter Magnitudes

The goal of this section is to formalize the observations of figure 5.4 on more complex networks and find out if some constraints indeed produce sparse networks. To do so, the distribution of weight magnitudes on the two models of section 5.2 when training with

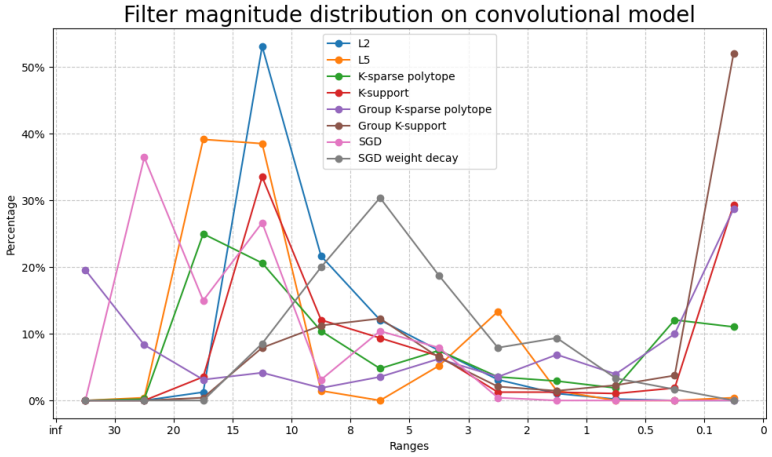


Figure 5.6: Filter L^1-norm distribution for different training methods. Setup is the same as in figure 5.5.

different constraints and optimizers is analyzed. The results can be found in figure 5.5. Both the K-sparse polytope and the K-support norm ball constraint successfully promote sparseness to a meaningful degree on both models, with the latter performing slightly better. The Group constraints are also successful in achieving sparse convolutional networks. Again, the effects of weight decay in SGD are noticeable, producing networks which are more sparse then most other methods do, although not reaching the level of the previously mentioned sparseness-inducing constraints.

To see if the logic of chapter 4 also successfully translates to structured sparsity, the distribution of L^1-norms of entire filters in the convolutional network is analyzed in figure 5.6. Again, the level of sparseness achieved is significantly higher for the (Group) K-support norm ball and (Group) K-sparse polytope than for all other methods. Additionally, the effect of the Group constraints seems to pay off, as both of them produce more filters with low L^1-norm than their regular counterparts. The Group K-support norm ball further stands out among these 4 constraints, as it produced networks where, on average, more than 50% of filters have an L^1-norm lower than 0.1, which should bring a definite performance boost in structured pruning.

As in the weight magnitude distributions, SGD with weight decay can be observed to have more filters with lower magnitude than a lot of other methods, as is to be expected when the weight are of lower magnitude. However, the filter distribution with most filters having a medium L^1-norm between 2 and 15 suggests that, in contrast to the Group constraints, the low magnitude is not focused on specific filters, but instead a general phenomenon across the entire network.

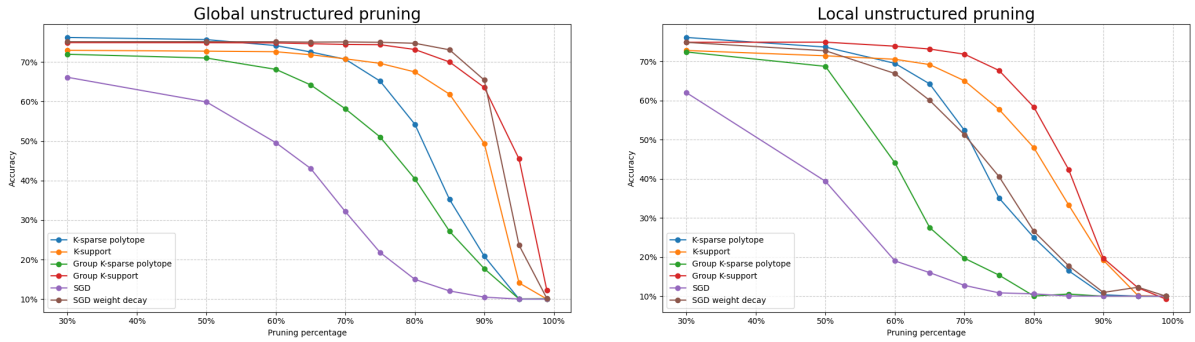


Figure 5.7: Test accuracies of ResNet18 on CIFAR10 for different training methods at different pruning percentages for global and local unstructured pruning. Constraints were initialized with $w = 15$ and K as 10% of each layers parameters. Training was done for 50 epochs.

5.5 Unstructured Pruning

To meaningfully analyze pruned accuracy, a deeper architecture is needed, such that more detailed results can be produced for different percentages of weights/filters pruned. To that end, the ResNet18 [He et al. (2016), sec. 4.1] architecture, a standard convolutional neural network benchmark architecture, is utilized. To have it work on CIFAR10, the first convolutional layer is modified to have a filter width and height of 3, and the final layer is updated to have 10 output neurons for the 10 classes of CIFAR10.

Two types of unstructured pruning are analyzed: global and local pruning. In global pruning, the removed weights are those with smallest absolute value across the entire network, while in local pruning, the same percentage of weights is removed from each individual layer. Both of these pruning techniques are applied to ResNet18 after it was trained with different methods, with figure 5.7 displaying the accuracy at various pruning percentages, i.e. percentages of removed weights. The figure shows that all used methods succeed in producing pruning-friendly networks, significantly outperforming the SGD optimized network starting at as little as 30% of weights pruned. In the case of global unstructured pruning, using SGD with weight decay, a common modification of SGD to create pruning-friendly networks, delivers similar accuracy as the best-performing constraint. However, in local unstructured pruning, both variants of the K-support norm ball constraint considerably beat SGD with weight decay, showing that MSFW in combination with the right constraints can create networks even more pruning-friendly than classical algorithms excelling at that task.

Looking beyond the comparison to SGD, the K-support norm ball constraint achieves a higher accuracy than the K-sparse polytope in both the Group and the regular form, showing that incorporating the momentum vector's magnitude in the update leads to much better performance after pruning. Surprisingly, there is no clear tendency if Group constraints perform better or worse than their regular counterparts in unstructured

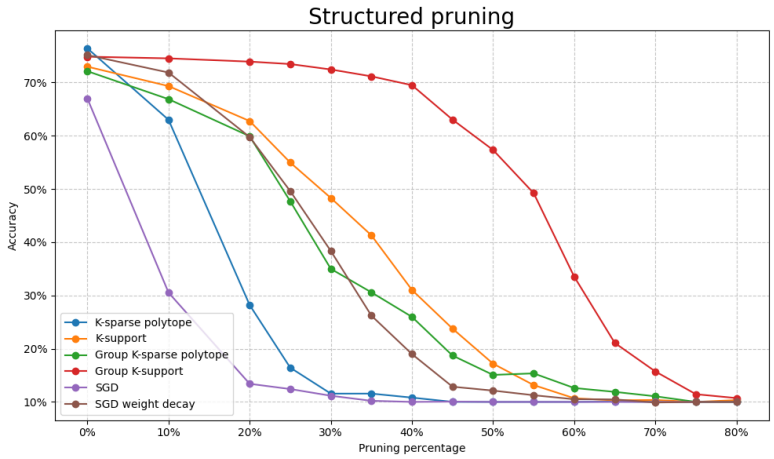


Figure 5.8: Test accuracies of ResNet18 on CIFAR10 for different training methods at different pruning percentages for structured pruning. Setup is the same as in figure 5.7.

pruning: Although the K-sparse polytope outperforms its grouped version, the Group K-support norm ball does significantly better than its regular version, having the best performance in the local case and on par with SGD with weight decay the best performance in the global case.

5.6 Structured pruning

The concept of global and local pruning can not trivially be applied on the structured level, as pruning here happens according to the L^1-norm of filters, which is strongly dependent on each layer. Even normalizing L^1-norms by the number of weights in a filter or the number of filters didn't sufficiently fix this problem, with all filters in one layer pruned and 10% accuracy reached already at small pruning rates like 10%. Therefore, in this section, classical structured pruning, i.e. removing the filters with smallest L^1-norm in each layer individually, will be applied. The resulting accuracies of ResNet18 on CIFAR10 can be found in figure 5.8.

These results show the promise of the introduced constraints and MSFW even stronger than the previous findings, with the Group K-support norm beating SGD with weight decay (which already does better than regular SGD) by more than 40% accuracy at various pruning percentages. The large gap can be explained by weight decay in SGD endorsing regular, but not really structured sparsity, as seen in figure 5.6. Furthermore, as hoped in chapter 4, the Group constraints both decisively defeat their non-grouped version, showing that focusing on the most important filters instead of the most important weights is a viable strategy to improve structured pruning-friendliness. However, the K-support norm ball is so effective that even in its non-grouped form, it produces a network more resistant to structured pruning than the Group K-sparse polytope.

6 Conclusion

This thesis examined the usage of the Stochastic Frank-Wolfe method as a structure-inducing optimizer for constrained neural networks. Through mathematical proof, it demonstrated that SFW is a viable neural network optimizer. Furthermore, with common modifications such as introducing momentum and rescaling the update vector, SFW achieved performance competitive with classical optimizers like SGD.

Additionally, constraints such as the K-sparse polytope and the K-support norm ball were introduced. When combined with SFW, these constraints produce sparse update vectors, selectively modifying only the K most important weights while pushing others closer to zero. Both led to extraordinarily sparse networks that were highly pruning-friendly, significantly outperforming SGD-trained networks across various pruning levels, with the K-support norm ball, in the case of local pruning, even beating networks trained by the already pruning-friendly method SGD with weight decay.

Expanding on this, grouped versions of the constraints were applied to convolutional networks, which, in combination with SFW, lead to updates of only K filters per iteration. The impact of this approach was evident not only in the L^1-norm analysis of filters after training with different methods, but already in unstructured pruning experiments, where the Group K-support norm ball achieved higher accuracy scores than its regular counterpart, and even more so in structured pruning experiments, where its performance towered above all other methods.

These findings establish the Frank-Wolfe method as a viable optimizer for neural networks, particularly for generating sparse, pruning-friendly models. This opens avenues for further research, such as applying constraints on a per-layer basis, e.g. using the Group K-support norm ball for convolutional layers due to its pruning-friendliness benefits while leaving other layers unconstrained. Additionally, since the structure-inducing properties of Frank-Wolfe extend beyond sparse updates, exploring alternative convex polytopes as weight constraints could yield further advancements in neural network optimization.

Bibliography

Argyriou, A., Foygel, R., and Srebro, N. (2012). Sparse prediction with the k-support norm. *Advances in Neural Information Processing Systems*, 25.

Braun, G., Carderera, A., Combettes, C. W., Hassani, H., Karbasi, A., Mokhtari, A., and Pokutta, S. (2022). Conditional gradient methods. *arXiv preprint arXiv:2211.14103*.

Canziani, A., Paszke, A., and Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

Combettes, C. W. and Pokutta, S. (2021). Complexity of linear minimization and projection on some sets. *Operations Research Letters*, 49(4):565–571.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142.

Frank, M., Wolfe, P., et al. (1956). An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110.

Grigas, P., Lobos, A., and Vermeersch, N. (2019). Stochastic in-face frank-wolfe methods for non-convex optimization and sparse neural network training. *arXiv preprint arXiv:1906.03580*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Jaggi, M. (2013). Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International conference on machine learning*, pages 427–435. PMLR.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Livieris, I. E. (2018). Improving the classification efficiency of an ann utilizing a new training methodology. In *Informatics*, volume 6, page 1. MDPI.

Livieris, I. E., Pintelas, E., Kotsilieris, T., Stavroyiannis, S., and Pintelas, P. (2019). Weight-constrained neural networks in forecasting tourist volumes: A case study. *Electronics*, 8(9):1005.

Lu, M., Luo, X., Chen, T., Chen, W., Liu, D., and Wang, Z. (2022). Learning pruning-friendly networks via frank-wolfe: One-shot, any-sparsity, and no retraining. In *International Conference on Learning Representations*.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA.

Pokutta, S., Spiegel, C., and Zimmer, M. (2020). Deep neural network training with frank-wolfe. arxiv e-prints, art. *arXiv preprint arXiv:2010.07243*.

Rao, N., Dudík, M., and Harchaoui, Z. (2017). The group k-support norm for learning with structured sparsity. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2402–2406. IEEE.

Reddi, S. J., Sra, S., Póczos, B., and Smola, A. (2016). Stochastic frank-wolfe methods for nonconvex optimization. In *2016 54th annual Allerton conference on communication, control, and computing (Allerton)*, pages 1244–1251. IEEE.

Ulbrich, M. and Ulbrich, S. (2012). *Nichtlineare Optimierung*. Springer-Verlag.

Wu, J. (2017). Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xie, J., Shen, Z., Zhang, C., Wang, B., and Qian, H. (2020). Efficient projection-free online methods with stochastic recursive gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6446–6453.

Zimmer, M., Spiegel, C., and Pokutta, S. (2022). Compression-aware training of neural networks using frank-wolfe. *arXiv preprint arXiv:2205.11921*.