One-Layer Neural Network Training with Frank-Wolfe

Mojtaba Amini[1], Kamile Dementaviciute[2], and Saeed Soufeh[3]

[1]Department of Mathematics, University of Padua
[2]Department of Computer Science, Vilnius University
[3]Department of Mathematics, University of Padua

Jun 2022

Abstract

In this paper we study the Frank-Wolfe method which is a projection free first-order methods for a simple neural network. Frank-Wolfe method is a useful optimization technique for constrained problems in machine learning because of its projection free property. We draw a comparison among different stochastic version of Frank-Wolfe method which is implemented for a one-layer neural network. We will compare the results of Standard FW, SFW, MSFW, and SVRF methods for a convex constraint problem on an l_1- ball region. The results are compared training the models on different binary classification datasets.

1 Introduction

Today, neural networks are one of the most commonly used machine learning techniques. Neural networks optimize a finite-sum problems. Minimization of these problems, which are also called empirical risk minimization, can be written differently for tasks such as multi-class classification, recommendation systems, etc. The finite sum function is(1):

$$\min_{x\in\Omega} F(x) = \min_{x\in\Omega} \frac{1}{m}\sum_{i=0}^{m} f_i(x) \tag{1}$$

We assume that F, f_i $(i \in 1, 2, ..., m)$ are all smooth, differentiable, and that the Ω is a convex set. Since in this study we will be working with binary classification tasks, the above function will be as follows(2):

$$\min_{\theta\in\Omega} J(\theta) = -\frac{1}{m}\sum_{i=0}^{m}(y^i log(\hat{y}^i) + (1-y^i)log(1-\hat{y}^i)) \tag{2}$$

Solving the above equation by one of the most popular methods in NN when dealing with a large datasets like SGD, requires the gradient descent calculations for each sample or mini-batch. One of the most common regularization techniques used on NN is parameter constraint. Implementing any gradient descent method for a constrained parameter space to a convex compact region requires one additional step at each iteration called projection, which in the case of SGD can be extremely costly[3]. This step is ensures that the parameters remain in the constrained domain, however it increases the computational cost of the methods. For this reason, the need of projection free methods like Frank-Wolfe to deal with constrained optimization problems arises. In this study, we will further explore the stochastic variants of Frank-Wolfe methods.

$$v_t = \arg\min_{v\in c}\langle\tilde{\nabla}(\theta_t, v)\rangle \tag{3}$$

Here, we study the Frank-Wolfe methods that solve a linear optimization problem (LMO)(3) to ensure that v_t moves in this descent direction and find the update value within the constrained region (4).

$$\theta_{t+1} = \theta_t + \alpha(v_t - \theta_t) \tag{4}$$

With assumption of Lipschitz continuous gradient having constant $L > 0$ and with step size $\alpha_k = \frac{2}{k+1}$, the Frank-Wolfe algorithm has sub-linear convergence rate of $\mathcal{O}(\frac{1}{k})$. This convergence rate will improve when we have a feasible set with special structure. In this study, we are going to compare the stochastic FW on a l_1-ball feasible set1., which is

$$C = \{x \in R^n : ||x||_1 \leq 1\} = conv(\{c, i = 1, 2, ..., n\}) \tag{5}$$

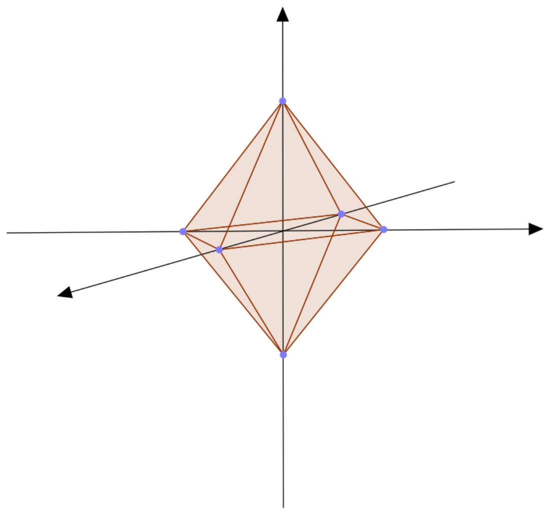

Figure 1: $l_1 - ball$[2]

and the descent direction here is

$$\hat{x}_k = sign(-\nabla_{i_k} f(x_k)) \cdot e_{i_k} \tag{6}$$

which the i_k is equal to

$$i_k = \arg\max_i |\nabla_i f(x_k)| \tag{7}$$

Since we only find the maximum and will consider it as direction, the cost for an iteration is only $\mathcal{O}(n)$. The pseudo-code related to Frank-Wolfe algorithm with l_1-ball is

Algorithm 1 Frank-Wolfe method for l_1-ball

```
Require: Set $Xx_1 = \pm e_i$ , with $i = 1, 2, ..., n$
  for k=1,.... do
    Set $\hat{x}_k = sign(-\nabla_{i_k} f(x_k)).e_{i_k}$ , with $i_k = \arg\max_i |\nabla_i f(x_k)|$
    if $\hat{x}_k$ satisfies some specific condition, then STOP
    Set $x_{k+1} = x_k + \alpha_k(\hat{x}_k - x_k)$ , with $\alpha_k = \frac{2}{k+1}$
  end for
```

Our contribution We have studied the stochastic variants of FW for a one-layer Neural Network, and have compared the results for different datasets. In the next chapter we will go into a more detailed explanation of the three Stochastic Frank-Wolfe methods, which are Stochastic Frank-Wolf (SFW), Stochastic Frank-Wolf with added momentum (MSFW), and Stochastic Variance Reduced Frank-Wolf (SVRF). Before that we will shortly introduce the previous work carried out in this field.

Related Work In Sebastian Pokutta et al. work [3] empirical efficacy of using stochastic Frank–Wolfe methods like SFW, MSFW and SVRF for training Neural Networks with different constrained parameters is examined. Comparisons both to current stochastic Gradient Descent methods and across different variants of stochastic Conditional Gradients is evaluated. Specifically, the general feasibility of training Neural Networks whose parameters are constrained by a convex feasible region using stochastic Frank–Wolfe algorithms is shown. In particular, as the primary purpose of the study, the impact of choosing a specific choice of constraints on the learned representations is studied. Furthermore, in Hazan, et al, work [1] by using recent variance reduction technique, SFW performance improves in terms of the number of stochastic gradient evaluations needed to achieve $1-\epsilon$ accuracy. And the performance is improved differently based on the objective function being smooth and strongly convex or smooth and Lipschitz. As a result, the variance reduction technique, previously shown to be highly useful for gradient descent variants, can also be very helpful in speeding up projection-free algorithms. Moreover, in Sashank J Reddi,et al, work [4] Frank-Wolfe methods for non-convex stochastic and finite-sum optimization problems are evaluated and their convergence properties are analyzed. Moreover, for objective functions that decompose into a finite-sum, ideas from variance reduction for convex optimization is used to obtain new variance reduced non-convex Frank-Wolfe methods that have provably faster convergence than the classical Frank-Wolfe method. The convergence rates in this paper were similar to those obtained for online Frank-Wolfe and only slightly worse than those obtained for stochastic Frank-Wolfe in the convex setting. In Xie,et al, work [5] the projection-free methods for for solving smooth Online Convex Optimization (OCO) problems is evaluated. To do so, two efficient projection-free online methods called ORGFW and MORGFW are used. These methods achieve optimal regret bounds while possessing low per-iteration computational costs. The proposed methods achieve nearly optimal regret bounds with low computational costs.

2 Stochastic Variants of Frank-Wolfe

Calculating the whole gradient at each iteration, when we are dealing with a large number of data points needs a large amount of memory and computational time. Therefore, using stochastic version of FW is proposed, which requires only the stochastic gradient calculation instead of the full gradient, and will solve a linear minimization problem to find the right direction to move. In this study, since we assume l_1-ball feasible region, we do not need to solve a linear optimization problem and only consider number of gradients calculated during each step. To achieve $1-\epsilon$ accuracy, we have compared the number of gradients calculated during the optimization in table1.

Table 1: Comparisons of different Frank-Wolfe variants[1].

Algorithm	Extra Conditions	Exact Gradients	Stochastic Gradients
FW	G-Lipschitz	$\mathcal{O}(\frac{LD^2}{\epsilon})$	0
SFW	-	0	$\mathcal{O}(\frac{G^2LD^4}{\epsilon^3})$
MSFW	G-Lipschitz	0	$\mathcal{O}(\frac{G^2LD^4}{\epsilon^3})$
SVRF	-	$\mathcal{O}(\frac{LD^2}{\epsilon})$	$\mathcal{O}(\frac{L^2D^4}{\epsilon^2})$

L is Lipschitz constant, D is the feasible set diameter $||x-y|| \leq D$ for every $x, y \in \Omega$.

2.1 Stochastic Frank-Wolfe (SFW)

Stochastic version of FW calculates the gradient only for a set of random samples in the mini-batches, and the rest of the algorithm is the same as before. Therefore, we calculate the gradients based on samples we have in the mini batches. In [3], it has been proved that $\tilde{\nabla} L(\theta_k)$ is an unbiased estimator for total gradient matrix in the simple FW, and that the new point $x_{k+1}$is still in convex set[2]. The stochastic version of the algorithm without considering the momentum term is provided in the algorithm below.

Algorithm 2 Stochastic Frank-Wolfe method for l_1-ball

```
Require: Set $Xx_1 = \pm e_i$ , with $i = 1, 2, ..., n$
  for k=1,.... do
    Uniformly sample i.i.d. $i_1.i_2, ...., i_{b_t}$ from $[1, .., n]$
    $\tilde{\nabla} L(x_k) \leftarrow \frac{1}{b_t} \sum_{j=1}^{b_t} \nabla f_{i_j}(x_k)$
    Set $\hat{x}_k = sign(-\tilde{\nabla}_{i_k} L(x_k)).e_{i_k}$ , with $i_k = \arg\max_i |\tilde{\nabla}_i L(x_k)|$
    if $\hat{x}_k$ satisfies some specific condition, then STOP
    Set $x_{k+1} = x_k + \alpha_k(\hat{x}_k - x_k)$ , with $\alpha_k = \frac{2}{k+1}$
  end for
```

2.2 Momentum Stochastic Frank-Wolfe (MSFW)

Momentum is the moving average of our gradients. Stochastic Frank-Wolfe does not compute the exact derivative of our loss function. Instead, we are estimating it on mini-batches, which means we might not always be moving in the optimal direction, and because of that our derivatives are noisy. So, exponentially weighed averages can provide us a better estimate which is closer to the actual derivative than our noisy calculations.

Algorithm 3 Momentum Stochastic Frank-Wolfe method for l_1-ball

```
Require: Set $Xx_1 = \pm e_i$ , with $i = 1, 2, ..., n$
  for k=1,.... do
    Uniformly sample i.i.d. $i_1.i_2, ...., i_{b_t}$ from $[1, .., n]$
    $\tilde{\nabla} L(x_k) \leftarrow \frac{1}{b_t} \sum_{j=1}^{b_t} \nabla f_{i_j}(x_k)$
    $m_k \leftarrow (1 - \rho_k) m_{k-1} + \rho_k \tilde{\nabla} L(x_k)$
    Set $\hat{x}_k = sign(-\tilde{\nabla}_{i_k} L(x_k)).e_{i_k}$ , with $i_k = \arg\max_i |m_k|$
    if $\hat{x}_k$ satisfies some specific condition, then STOP
    Set $x_{k+1} = x_k + \alpha_k(\hat{x}_k - x_k)$ , with $\alpha_k = \frac{2}{k+1}$
  end for
```

2.3 Stochastic Variance Reduced Frank-Wolfe (SVRF)

Many variants have been proposed to improve the practical efficiency of SFW, most of these rely on modifying how the (unbiased) gradient estimator is obtained. Variance Reduced techniques originally try to reduce the variance in gradient estimation of mini-batches in stochastic methodologies[1]. In this method, we will store a larger number of different set of parameters to model the variance reduction technique[1]. The main difference in comparison to Stochastic Frank-Wolfe is we change the gradient calculation formula(7) to:

$$\tilde{\nabla} F(x_k) \leftarrow \nabla F(x_0) + \frac{1}{b_{k,t}} \sum_{j=1}^{b_{k,t}} (\nabla f_{i_j}(x_k) - \nabla f_{i_j}(x_0)) \tag{7}$$

The full algorithm for SVRF can be found below.

Algorithm 4 Stochastic Variance Reduced Frank-Wolfe method for l_1-ball

```
Require: Set $Xx_1 = \pm e_i$ , with $i = 1, 2, ..., n$
  for t=0,...,T-1 do
    take snapshot $x_0 = \theta_t$ and compute $\nabla F(x_0)$
    for k=1,.... do
      Uniformly sample i.i.d. $i_1.i_2, ...., i_{b_t}$ from $[1, .., n]$
      $\tilde{\nabla} F(x_k) \leftarrow \nabla F(x_0) + \frac{1}{b_{k,t}} \sum_{j=1}^{b_{k,t}} (\nabla f_{i_j}(x_k) - \nabla f_{i_j}(x_0))$
      Set $\hat{x}_k = sign(-\tilde{\nabla}_{i_k} F(x_k)) \cdot e_{i_k}$ , with $i_k = \arg\max_i |\tilde{\nabla}_i F(x_k)|$
      Set $x_{k+1} = x_k + \alpha_k(\hat{x}_k - x_k)$ , with $\alpha_k = \frac{2}{k+1}$
    end for
  $\theta_{t+1} \leftarrow x_{K_t}$
  end for
```

3 Experiments

To support the theory, we conduct experiments in the binary classification for a neural network problem. Four different datasets have been selected with a large number of samples and features. The loss function of the binary classification is a binary cross-entropy (2). Further in the paper we compare three different variants of Frank-Wolfe (SFW, MSFW, and SVRF). Since the goal of the study is to evaluate the behavior of the stochastic FW methods, we pick a simple version of neural network with one hidden layer. We explain the procedure in the following paragraphs.

First, we built a simple neural network with one hidden layer. We had to determine the size of each layer, achieving different parameters (weight and biases), forward propagation (9), determine activation functions (10) and backward propagation2. In the following equations the superscript $[l]$ denotes a quantity associated with the l^{th} layer, superscript (i) denotes a quantity associated with the i^{th}, and lowerscript i denotes the i^{th} entry of a vector.

At first, the parameters are intialised to a number close to zero to avoid symmetry in network and to be in the region of feasible set. The linear forward module is as follows (9):

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]} \tag{9}$$

For computing the next layer's activation, two different activation functions are used :
Sigmoid function:

$$\sigma(Z) = \sigma(WA + b) = \frac{1}{1 + e^{-(WA+b)}} \tag{10}$$

ReLU function:

$$A = RELU(Z) = max(0, Z) \tag{11}$$

When we go through forward propagation, the last layer output is the predicted label. The performance of the neural network is compared with the true amounts of labels by a cost function. The cost function used in this study is binary cross-entropy (2). Having the cost function computed, in the backward propagation, we compute the derivatives with respect to the loss function2:

Table 2: Derivatives of parameters with respect to the loss function

Parameter	Formula
$dZ^{[l]}$	$\frac{\partial \mathcal{L}}{\partial Z^{[l]}}$
$dW^{[l]}$	$\frac{\partial \mathcal{J}}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$
$db^{[l]}$	$\frac{\partial \mathcal{J}}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)}$
$dA^{[l-1]}$	$\frac{\partial \mathcal{L}}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]}$

In the last part, we use the computed derivatives in the backward propagation and a specific learning rate to update the parameters based on different variants of stochastic Franke-Wolfe algorithms.

4 Results

4.1 Datasets Hyperparameters

The results of the different variants of the stochastic Frank–Wolfe algorithm introduced above (SFW, MSFW, SVRF) were run on 4 different datasets. First of all MNIST and Fashion MNIST datasets were used, for the purpose of rough comparison to experiments carried out by Sebastian Pokutta et al. [3]. However, as mentioned before, in the case of this study, a binary classification was implemented, therefore we do not expect the results to match completely. Additionally, two newly outsourced datasets were used: an image dataset consisting of different flower species and a toy dataset Moon that is made of half circle - moon clusters. Both of these were also used for binary classification.

The hyperparameters were chosen following the works of the above referenced papers. The reviewed studies chose hyperparameters in different ways, therefore for these experiments, all the different approaches were attempted, with the final choice being the one that brought the best results. Therefore, for all the experiments carried out on a one hidden layer Neural Network, the epoch number of 50 was used (except in the case of Moon dataset, where the epoch number was 100), the mini-batch size for all three methods was a constant number of 50, and finally in all cases 64 hidden units were used, as in all the experiments by Pokutta et al. [3]. Finally, all three methods have a moving learning rate $\alpha_k = \frac{2}{k+1}$, where k is the number of that specific iteration.

4.2 Result Analysis

The accuracy and loss result comparison for all three methods and all 4 datasets is presented in table 3 and 4 respectively.

Table 3: Test set Accuracy comparisons of different Stochastic Frank-Wolfe variants

Dataset	SFW	MSFW	SVRF
MNIST	99.62%	99.67%	**99.81%** %
Fashion MNIST	**96.65%**	**96.65%**	95.40%
Flowers	**62.30%**	60.32%	60.66%
Moon	83.80%	80.83%	**86.03%**

Table 4: Training loss comparisons of different Stochastic Frank-Wolfe variants.

Dataset	SFW	MSFW	SVRF
MNIST	0.140	**0.129**	0.138
Fashion MNIST	**0.174**	0.191	0.203
Flowers	0.693	0.707	**0.678**
Moon	**0.335**	0.410	0.396

Although the final results do not vary significantly among the different methods, from the plots below 2, 3, 4 and 5, we can observe a clear difference when comparing SVRF to SFW and MSFW. In all cases SVRF converges considerably faster, especially for MNIST and Fashion MNIST datasets. Here SVRF requires less than 5 iterations to reach the desirable loss and even less when it comes to the test set accuracy, while SFW and MSFW require around 20 epochs to reach a similar loss and just over 10 for a similar accuracy. Having said that it's important to note that this is not the case for all instances and that minor adjustments, such as ones of hyperparameters, random seed etc. can slow down the convergence of SVRF making it more similar to those of the other two methods. On the other hand, the memory space and CPU time required for the same number of iterations is much larger for SVRF in comparison to SFW and MSFW.

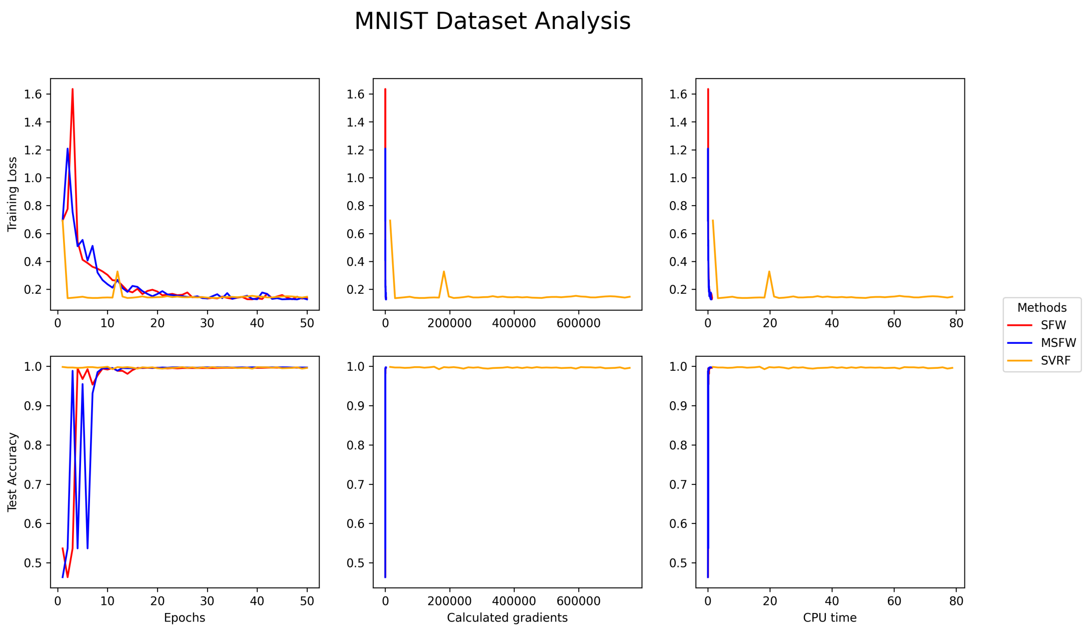

Figure 2: Epochs = 50; hidden layer size = 64; batch size = 50

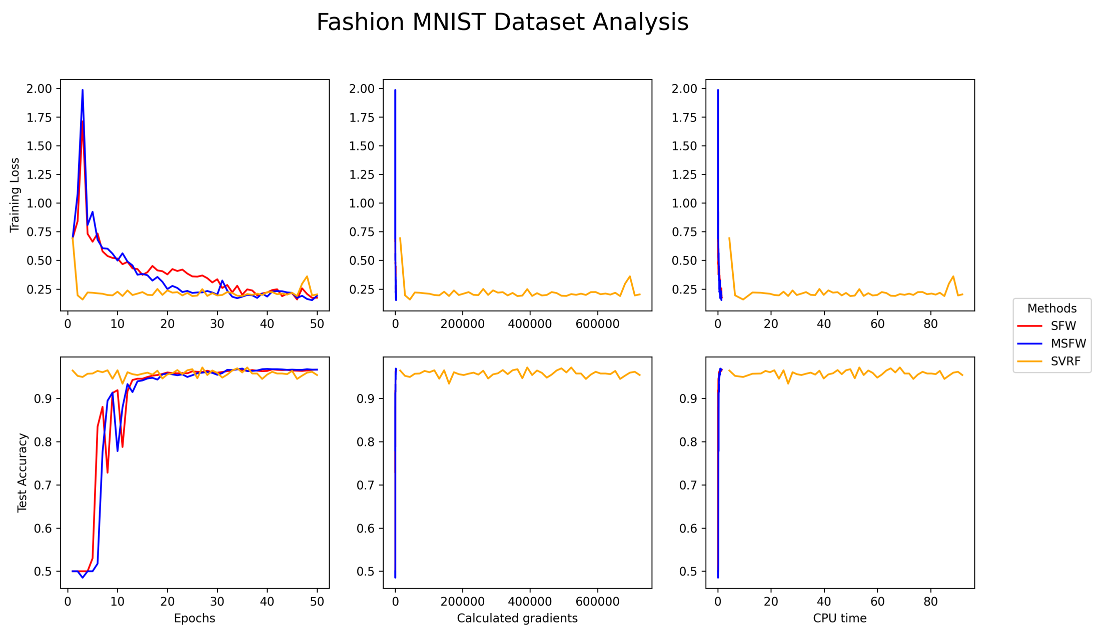

Figure 3: Epochs = 50; hidden layer size = 64; batch size = 50

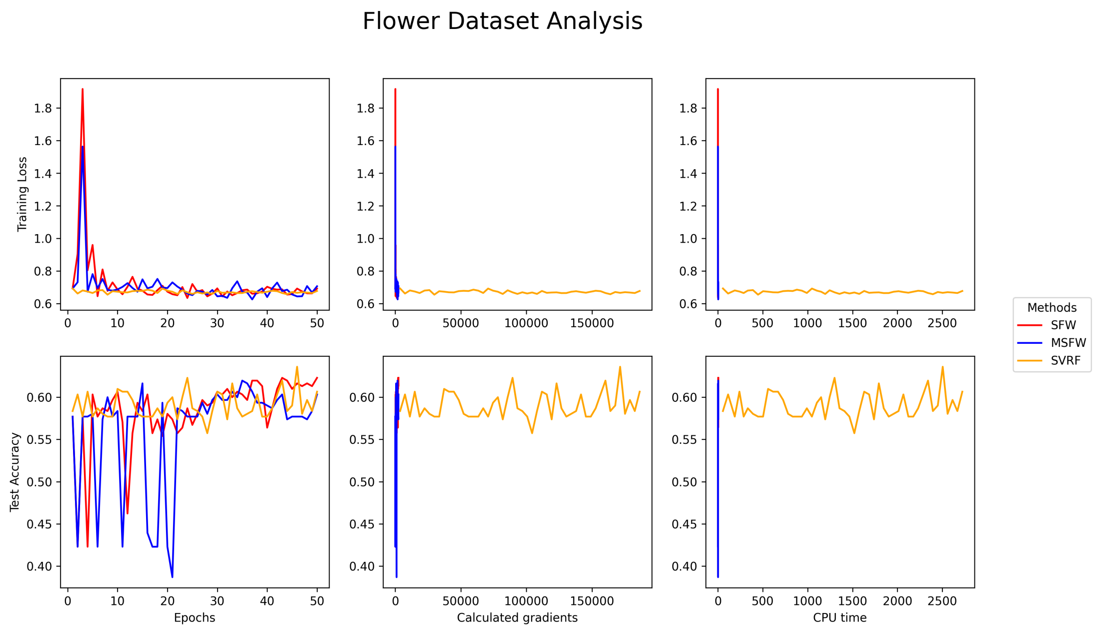

Figure 4: Epochs = 50; hidden layer size = 64; batch size = 50

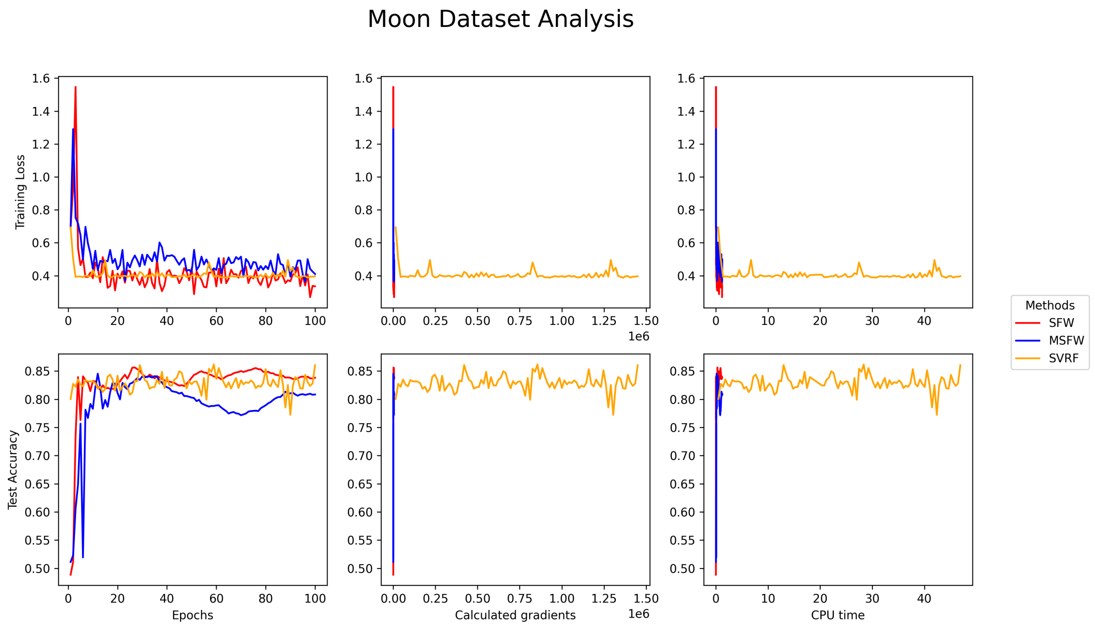

Figure 5: Epochs = 100; hidden layer size = 64; batch size = 50

5 Conclusion

Similarly as in the previously conducted studies, the results conclude that SVRF substantially reduces the number of iterations required to achieve the same test accuracy and training loss in comparison to the other two methods. However, as suggested in previous studies, the disadvantage of SVRF as well as SFW is that they offer little benefit in deep neural networks, however this is where MSFW has an advantage with its easy implementation and significant improvement when considering the metrics vs. CPU time. Therefore in the future we would hope to explore its application for deep neural networks.

References

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

[2] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[3] Sebastian Pokutta, Christoph Spiegel, and Max Zimmer. Deep neural network training with frank-wolfe. 2020.

[4] Sashank J Reddi, Suvrit Sra, Barnabás Póczos, and Alex Smola. Stochastic frank-wolfe methods for nonconvex optimization. In *2016 54th annual Allerton conference on communication, control, and computing (Allerton)*, pages 1244–1251. IEEE, 2016.

[5] Jiahao Xie, Zebang Shen, Chao Zhang, Boyu Wang, and Hui Qian. Efficient projection-free online methods with stochastic recursive gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6446–6453, 2020.