Frank Wolfe for Neural Networks

Alexander Immer, Ignacio Aleman, Sidak Pal Singh
IC, EPFL, Switzerland

***Abstract*—We study how Frank-Wolfe methods perform when applied to neural networks and how they induce sparsity. We compare the performance as well as sparsity achieved using another method for constrained optimization: Projected Stochastic Gradient Descent. We will further compare it to the standard Stochastic Gradient Descent as a reference. Experiments are conducted on the popular MNIST dataset. The relevant code of the performed experiments can be found on GitHub.** [1]

I. Introduction

We are dealing with a problem of the following form:

$$\min_{W \in D} f(W) \tag{1}$$

W is the weight matrix where the i^{th} column represents the parameters of the corresponding layer of a neural network. We define D to be the intersection of the constraints, $||W^i||_1 \leq \kappa_i$, where κ_i are positive constants. Since the intersection of convex sets is still convex, D is convex. Further, the resulting intersection convex set is compact. Since f is the output of a neural network it is in general non-convex.

Our goal is to understand how, and what type of sparsity Frank-Wolfe (FW) methods induce in W in the case of ℓ_1 regularization. Using ℓ_1 regularization has been shown to produce sparse parameters, e.g. in the case of Lasso [1]. The Lasso in its unconstrained formulation is equivalent to ℓ_1 weight decay in neural networks **??**. In the particular case of FW, we expect sparsity due to the sparse FW updates applied to an initially sparse network. We conduct our experiments on the MNIST digit classification task, using a simple multi-layered fully connected network. In this setting, W^i denotes the vector of parameters of the i^{th} layer.[2]

We compare the results of FW to Projected Stochastic Gradient Descent (PSGD), which projects onto the l_1 ball of radius κ after each gradient update. Most optimization methods for the constrained setting rely on projections which allow the algorithm to ensure feasibility after every step. In the case of projection onto the l_1 ball, the projection cost is $O(n \log n)$. In general however, oracle calls can be very expensive (e.g. projection onto nuclear norm ball for matrices) [2]. In these points, Frank-Wolfe can have a high advantage as it is (1) projection-free and (2) its linear minimization oracle (LMO) for the vector ℓ_1 norm is $O(n)$. Further, it produces sparse updates, which we will exploit to train highly sparse networks.

It is important to note that we use Frank Wolfe based on stochastic gradients. This does not have an impact on feasibility [3]. As only the convergence guarantees would be affected, we can neglect this problem in the case of non-convex optimization. Consider w to be some layer's parameters W^i and $\tilde{\nabla} f(w^t)$ its stochastic gradient. Then, the update works as follows for step $t \geq 0$:

$$\hat{s} = \arg\min_{s \in D} \langle \tilde{\nabla} f(w^t), s \rangle$$

$$w^{t+1} = (1-\gamma) w^t + \gamma \hat{s} \; ; \; \gamma = \frac{2}{2+t}$$

Here $\hat{s}$ is the result of the LMO of the stochastic gradient subject to D at time t.

II. Frank Wolfe for Neural Networks

To apply FW to neural networks, we have to make sure to have a function of the type in Eq. 1. This is ensured since we place a Lasso-type regularization on the vector of each parameter set, e.g. weight matrix or bias vector in the fully connected case. If we do not constrain each parameter, D would not be compact since $\mathbb{R}$ is not. Here, PSGD seems advantageous despite the higher projection cost. However, there is a striking argument for FW in deep learning: its iterates are sparse. Therefore, in the case of ℓ_1 regularization, we only update a single weight parameter of each layer after each batch. Due to $\gamma^{t=0} = 1$, there is only one weight non-zero for each layer after the first iteration, hence the initial solution is sparse. The longer we train, the more parameters can be potentially non-zero, i.e. non-sparse. This has two interesting implications: (1) the earlier we stop, the fewer parameters are potentially *activated* (sparsity) and (2) we can investigate how many neurons and weights are actually required for a particular problem (minimum capacity). Lastly, the sparse iterates can help in distributed training settings. One could potentially compute several solutions $\hat{s}^i$ on m different machines for individual batches. The average of those, i.e. $\hat{s} = \frac{1}{m} \sum_{i=1}^{m} \hat{s}^i$, is a convex combination and hence does not harm feasibility. The communication would only consist in key-value pairs since the $\hat{s}^m$ are sparse. This is much more efficient than communicating dense gradients.

III. Measures of Sparsity for Neural Networks

To talk about sparsity in neural networks more precisely, we use the following metrics. The first one is the **parameter activity**, which is

$$\frac{\#(|W_k^i| > \epsilon)}{\#W_k^i}, \quad \epsilon = 10^{-8}$$

i.e. the ratio of parameters that are not too close to zero. Correspondingly, the metric of **node activity** denotes the

[1] https://github.com/aleximmer/FWDL

[2] Actually, we have two different W^i for each layer: biases and weights. Both are regularized individually.

fraction of nodes that are redundant. Redundant nodes do not impact the output of a neural network, which happens if all its outgoing weights are close to 0. Finally, the metric of **path activity** is the ratio of paths that are not redundant and follows from the definition of active nodes. It is computed by

$$\prod_{layer_i \in layers} \text{\# active nodes in } layer_i$$

IV. Experiments

We compare the optimization methods using a multilayer perceptron [4] with three hidden layers, each comprising 80 neurons with biases. The last layer uses a softmax activation and the hidden layers Sigmoid. The resulting network has 76570 parameters. The largest parameter groups according to W^i in Eq. 1 are the weight matrices of the hidden layers with 80^2 variables. We use Cross Entropy as the loss function and train on the MNIST data-set which is composed of 28 * 28 images of digits from 0 to 9 [5]. We train this model using the three aforementioned optimizers FW, PSGD, and a unconstrained reference with SGD. For the constrained objectives, we use the same κ_i for each parameter group and run a grid search.

A. Results

In Fig. 1 (left) we compare the obtained training and testing accuracies for the models. We observe that SGD and PSGD require only a few iterations to achieve peak performance, but start to overfit after around 50 epochs. On the other side, FW requires around 150 epochs to reach the same performance and even seems to further improve while maintaining a minimal generalization loss.

Following our definitions of sparsity in Sec. III we compare PSGD and FW in Fig. 1 (right). In all metrics of interest, the network trained with FW shows a higher sparsity. In fact, the performance of the PSGD-network decreases while the sparsity increases. For the FW-network this is actually reversed: the performance increases with increasing activity but reaches peak-performance under still extremely sparse conditions. For completeness, we shall note that the network trained with SGD is at no time sparse, i.e. has 100% activity with respect to all metrics.

To understand the sparsity pattern of the FW-network, we visualize the weights between hidden layers towards the final layer in Fig. 2. The weights of the first layer are shown in Fig. 3. We observe that we have structured regions for both SGD and PSGD whereas the FW-network has very selective regions due to its sparsities and is non-smooth.

Role of κ on performance: Since the value of κ effectively controls the range of regularization, we observe that for very small values of it, the network isn't able to learn a lot and doesn't perform well. If the value is set to be very high, it tends to overfit using few very large weights. The sweet spot turns out to be in the order of the number of parameters: we obtain good results with $1000 \leq \kappa \leq 6000$ and as pointed out before, we have up to $80^2 = 6400$ parameters in a regularization group.

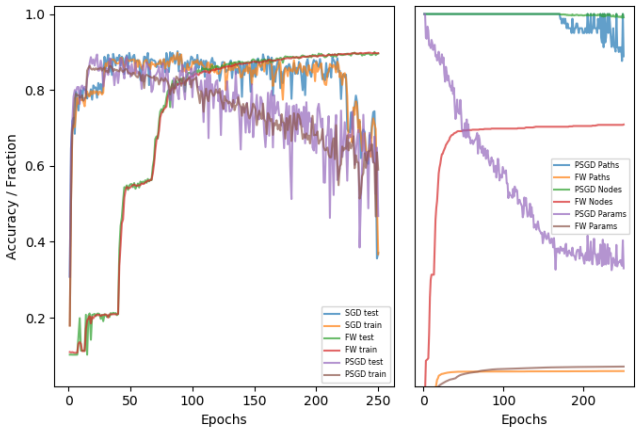

Fig. 1. Both figures share their y-axis. **Left**: test and train accuracy of the network trained with FW, PSGD, and SGD. **Right**: sparsity metrics on the best networks trained with for PSGD and FW. For FW, path, node, and parameter activity is in the end at 5.9%, 70%, and 7.2%, respectively. In same order for PSGD: 88%, 99%, and 33%.

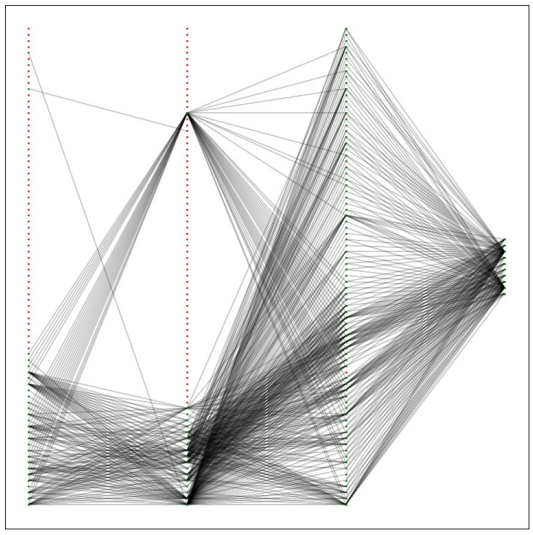

Fig. 2. Visualization of the hidden layers of the fully trained FW-network. The 10 output neurons are also shown on the right. We can see that the network is extremely sparse and that weights and nodes in higher layers are of higher activity. The weights connecting the first hidden layer to the input are not shown due to their dimensionality. For a visualization of the first-layer weights, see Fig. 3.

V. Conclusion and Future Directions

Frank Wolfe based optimization using ℓ_1 constraints does not only regularize well but also induces sparsity in neural networks. Since the oracle for the ℓ_1 norm results in a coordinate descent, the network is built up by choosing parameters one by one. This can give insights into the required capacity of neural networks for different tasks. We have shown that applying Frank Wolfe to nonconvex optimization seems very promising. However, due to the space constraints, many further ideas and questions have to be deferred: nuclear norm regularization for RNNs, impact of number of hidden layers, FW for ConvNets.

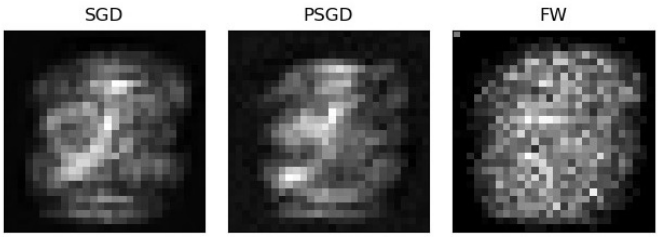

Fig. 3. The first layer connects all input pixels to the first hidden layer. This figure is a grayscale image of the maximum outgoing weight for each pixel. The image is shown for all three optimization methods.

REFERENCES

[1] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[2] M. Jaggi, "Revisiting frank-wolfe: Projection-free sparse convex optimization." in *ICML (1)*, 2013, pp. 427–435.

[3] D. Goldfarb, G. Iyengar, and C. Zhou, "Linear convergence of stochastic frank wolfe variants," *arXiv preprint arXiv:1703.07269*, 2017.

[4] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," CORNELL AERONAUTICAL LAB INC BUFFALO NY, Tech. Rep., 1961.

[5] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.