Hyperparameter Optimization using Tensor Completion via Robust PCA

Ahmed Almostafa Gashgash [*1] **Ziyang Wu** [*2] **Andreas Schoenleben** [*2]

Abstract

Modern Deep Learning models require a great number of hyperparameters. Optimizing these hyperparameters is a challenging problem for practitioners in this field. Classical hyperparameter optimization methods like grid search suffer from the curse of dimensionality, where an extensive search of the hyperparameter space can become very computationally expensive. We address this issue by only calculating the loss for some entries of the grid. The test losses obtained from these models form a sparse loss tensor. Using Robust Tensor PCA, we estimate a low rank approximation of the loss tensor from the available values. This low rank representation reconstructs the missing entries so that the best hyperparameter set can be chosen. Our method achieves a two times speedup compared to dense grid search on a 1000-entry hyperparameter search space at almost optimal accuracy.

1. Introduction & Related Work

In recent years deep learning models have been extensively applied to numerous tasks in a variety of disciplines like computer vision and natural language processing. However, these models are rarely parameter free and require manual tuning. Examples of these hyperparameters include the learning rate in stochastic gradient descent (SGD), the batch size, the momentum and regularization parameter, and capacity of the model through the number of hidden units.

There exist many approaches to optimizing hyperparameters, some rely on human expertise or prior knowledge where it was found empirically that certain values for hyperparameters give good results. However, there are no guarantees and there could exist instances where these values are the incorrect choices for that specific task or model. Another approach is Grid Search (GS). In this approach a grid containing the candidate settings is set up, each entry represents the test loss of the whole model under that setting and the best is chosen. This could be thought of as a brute force method. Due to the curse of dimensionality the number of possible hyperparameter settings increases exponentially in the number of hyperparameters, which makes grid search a very computationally expensive approach. Other methods such as random search (Bergstra & Bengio, 2012), tackle the problem of dimensionality. Here, a fixed number of experiments each with a random setting of hyperparameters is tested. However, there is no guarantee that it necessarily gets anywhere close to the optimal choice. Bayesian Optimization (Snoek et al., 2012) approaches have proven to give better results than grid and random search in fewer experiments empirically, but it is considered a heavyweight method that is not as easily implemented as grid search for example and whose iterations are computationally costly.

To that extent we propose a variant to grid search that has the same computational complexity as random search but achieves accuracy comparable to grid search. In essence we propose to compute a sparse grid containing the test loss of a few configurations. For three hyperparameters, the grid would represent a tensor of dimension $d = 3$, each represents the possible values for this hyperparameter. This sparse tensor is then factorized into a low rank model and the missing values are estimated accordingly. We address the problem of hyperparameter optimization in the setting of a neural network classifier. The idea was partly inspired by a paper which estimates the performance of models on a new dataset using a loss matrix obtained from previous experiments with a number of datasets and models (Yang et al., 2018).

2. Background & Methodology

In this paper we focus on tuning the hyperparameters of a simple neural network that performs classification on the MNIST dataset. For simplicity, we examine a neural network with one fully connected hidden layer and ReLu activation. The network is trained using SGD with momentum. We focus on optimizing three hyperparameters: the number of units in the hidden layer, the learning rate and the

[*]Equal contribution [1]Department of Electrical and Computer Engineering, Cornell University, Ithaca, United States [2]Department of Computing and Information Science, Cornell University, Ithaca, United States. Correspondence to: Ahmed Almostafa Gashgash <atg74@cornell.edu>, Ziyang Wu <zw287@cornell.edu>, Andreas Schoenleben <as3932@cornell.edu>.

momentum.

To validate our approach, we show that our loss tensor $\mathbf{M}$ has an underlying low rank structure. We could therefore approximate a sparse version of $\mathbf{M}$ by a low rank representation. We first calculate the losses of models trained with a limited number of hyperparameter settings selected from the grid. These losses form a sparse tensor depending on the dimensionality of the hyperparameter space. We then approximate these values using a low rank representation and estimate the losses of the remaining models which were not initially evaluated. This is done using Robust PCA introduced below. We will evaluate our approach on different sparsity[1] levels of our loss matrix. Our method would be compared with grid search and random search, and we hope to achieve a better trade-off between model accuracy and hyperparameter setting search time.

2.1. Robust PCA

We use Robust Tensor PCA introduced in (Candes et al., 2009) to construct the low rank approximation of the sparse loss tensor. Formally, assume we have a matrix $\mathbf{M} \in \mathbb{R}^{n\times m}$ that posses a low rank structure, so that it could be represented as:

$$\mathbf{M} = \mathbf{L} + \mathbf{S}$$

where $\mathbf{L} \in \mathbb{R}^{n\times m}$ is low rank and $\mathbf{S} \in \mathbb{R}^{n\times m}$ is sparse. Classical Principal Component Analysis (PCA) (Abdi & Williams, 2010) looks for the best rank-k estimate of $\mathbf{L}$ by solving:

$$\begin{aligned} &\min \|\mathbf{M} - \mathbf{L}\| \\ s.t. \quad &rank(\mathbf{L}) \leq k \end{aligned}$$

where $\|.\|$ denotes the 2-norm. This problem could be efficiently solved. However, PCA is very brittle if noise in the original matrix $\mathbf{M}$ is present, or if $\mathbf{M}$ contains missing entries. In these cases, the estimated $\hat{\mathbf{L}}$ from PCA could be very far from the true low rank matrix $\mathbf{L}$. In (Candes et al., 2009) Robust PCA (RPCA) is introduced and proved to be successful in recovering the low rank matrix even though the original matrix might contain missing entries or corrupt data. In the case of missing entries, the problem is formulated as follows: for matrices supported on $\Omega \subset [n] \times [m]$ let $\mathcal{P}_\Omega$ be the orthogonal projection onto the linear space such that:

$$\mathcal{P}_\Omega \mathbf{M} = \begin{cases} \mathbf{M}_{ij}, & \text{if } (i,j) \in \Omega \\ 0, & \text{if } (i,j) \notin \Omega \end{cases}$$

[1] In this paper, sparsity refers to the ratio of missing values in the matrix. Density is $1-$sparsity. For example, full grid search has a density of 1.

Then if we only have a few entries available of $\mathbf{M} = \mathbf{L} + \mathbf{S}$ we can write this set as

$$\mathbf{Y} = \mathcal{P}_{\Omega obs}\mathbf{M} = \mathcal{P}_{\Omega obs}(\mathbf{L} + \mathbf{S})$$

That is we only see the entries $(i,j) \in \Omega_{obs} \subset [n] \times [m]$. The proposed approach to recover the missing entries is by solving the following problem:

$$\begin{aligned} &\min \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 \\ s.t. \quad &\mathcal{P}_{\Omega obs}(\mathbf{L} + \mathbf{S}) = \mathbf{Y} \end{aligned}$$

where $\|.\|_*$ is the nuclear norm. This is saying that among all decompositions matching the available data, the previous program minimizes the weighted combination of the nuclear norm and the $\ell 1$ norm. In (Candes et al., 2009) its shown that under some conditions, the previous approach recovers the low rank component exactly. This method could also be extended to tensors of dimension $d \geq 2$.

The Tensor learning library Tensorly[2] supports Robust Tensor PCA with missing values. Alongside our original tensor $\mathbf{M}$ with missing values, the function requires passing a mask of the same size as $\mathbf{M}$ with 1s in the place of available values and 0s everywhere else. The function then returns both the low rank reconstruction of the input tensor as well as the sparse loss matrix.

3. Experimental Evaluation

In our experiments we use a neural network with one fully connected hidden layer for classifying the MNIST dataset. All hyperparameters are fixed besides the learning rate (LR), the momentum and the number of hidden neurons which we aim to optimize. Ten evenly spaced possible values are selected for each hyperparameter. We set the range for the learning rate to be 0.001 to 0.1, 0.0 to 1.0 for the momentum and 5 to 100 for the number of hidden units. Our network is trained for two epochs.

3.1. Singular Value Decay

We first test whether our loss tensor could be approximated by a low rank model. We fix one of the three hyperparameters and vary the other two. Grid search is then applied by calculating the training loss of our model under all different settings in the grid. In this case our tensor is $\mathbf{X} \in \mathbb{R}^{n\times m}$ where $n = m = 10$. After obtaining these dense matrices, we apply a Singular Value Decomposition (SVD) to examine their singular values.

The results show that the loss matrices indeed possess a low rank structure. From the figures we see that the first two singular values hold most of the energy, therefore the

[2] http://tensorly.org/dev/

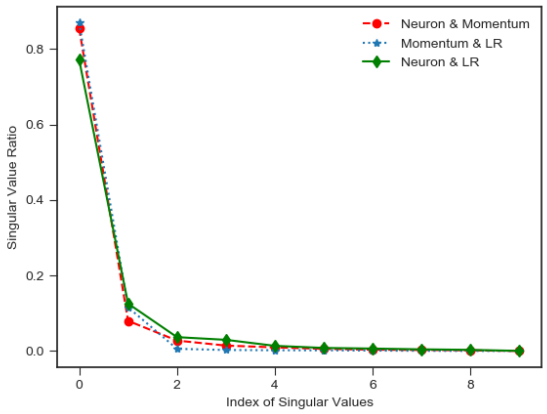


Figure 1. Singular value decay of the loss matrices on MNIST

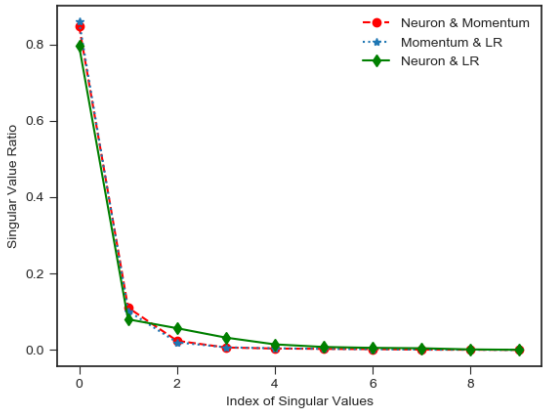


Figure 2. Singular value decay of the loss matrices on Fashion-MNIST

matrix could be estimated by a rank-2 matrix with minimal loss. Figure 2 shows the results of the same experiment on fashion-MNIST. Fashion-MNIST is a more complex dataset with the exact same dimensions as MNIST. The experiment confirms that the low rank assumption for the loss matrices holds for different datasets.

3.2. Comparison with Grid Search

We evaluate our method on matrices and a tensor. For our loss matrices, we fix one of the three hyperparameters to be constant and vary the other two. We also calculate a three way loss tensor for the combination of all three hyperparameters. We then randomly throw out a certain portion of entries in the loss matrix or tensor $\mathbf{M}$ to form a sparse version $\mathbf{M}_{sp}$, which is then imputed using Robust PCA as discussed before. We evaluate how sparse the matrix or tensor can be so that a meaningful approximation of the missing values can still be achieved. With increasing sparsity we expect the estimation errors to increase as well as the run time to decrease.

We use the following metric to assess the quality of reconstruction of our loss matrices and tensor. First we determine the optimal hyperparameter setting, this corresponds to the lowest loss value in our reconstructed tensor. We compare this setting with the original full tensor, and record whether it was in the top 1, 3, 5, or 10 of the original losses. We repeat each experiment 100 times and plot the average. The results for the loss matrices can be seen in figure 5 and the loss tensor in 3.

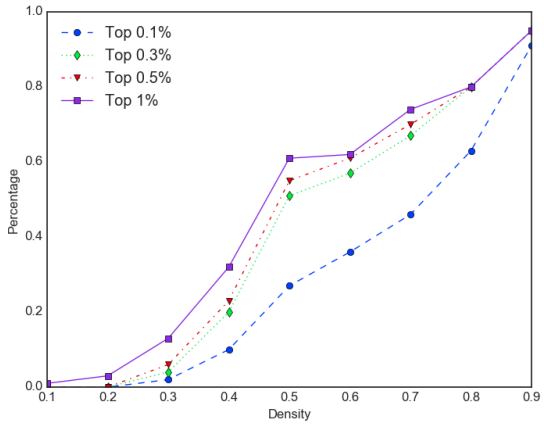


Figure 3. Percentage of 100 experiments where the best predicted setting lied in the top 0.1, 0.3, 0.5, and 1 percent of the original tensor

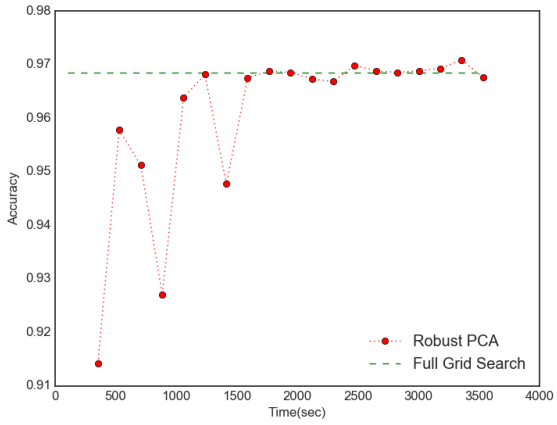
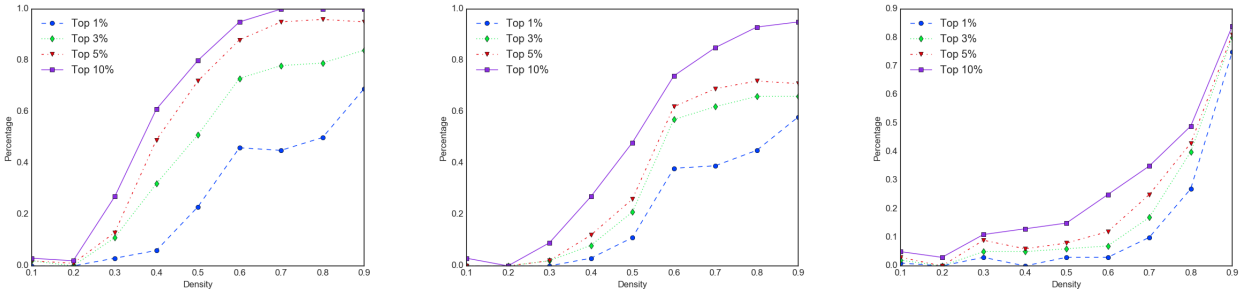


Figure 4. Wall clock time

Our results match the assumption that the loss matrices and tensors can be approximated well using a low rank model. Figure 5(a) shows that at a density of 60% the top recovered hyperparameter setting is in the real top 10 in



(a) Number of hidden units and momentum (LR = 0.001) (b) Number of hidden units and learning rate (Momentum = 0.9) (c) Momentum and and learning rate (50 hidden units)

Figure 5. Percentage of experiments where reconstructed top loss represents hyperparameter configuration with real loss value in Top 1, 3, 5 and 10 for three loss matrices spanning two hyperparameters each

more than 90% of the trials. Figure 5(c) seems to suggest that the reconstruction for momentum and learning rate does not perform as well. However, the accuracy values that the different models achieve are very similar so that the algorithm chooses between good models even if it fails to select the best. This claim is supported in Section 3.3.

Additionally, we compare wall clock time of our method with grid search on the loss tensor. Specifically, we plot the time needed for Robust PCA to select a top hyperparameter configuration at different sparsity levels from the three way loss tensor and the accuracy achieved by this configuration. We compare it with test accuracy found by Grid Search on the whole search space(1000 entries) and the result is shown in Figure 4. The result shows that with very sparse loss tensor, Robust PCA cannot fully recover the underlying structure of the dense tensor, leading to fluctuations in test accuracy. However, after a sparsity level of around 0.5, Robust PCA can stably recover the best hyperparameter configuration and achieves accuracy equivalent to that found by Grid Search. Our method, therefore, achieves a speedup of roughly 100% while maintaining good test accuracy compared to Grid Search. Note that the green line represents Grid Search which takes approximately (3500 sec) to run.

3.3. Comparison with Random Search

We compare our hyperparameter optimization scheme using tensor completion with random search (Bergstra & Bengio, 2012). In random search, for each hyperparamter, we choose the value according to a uniform distribution from the same set as grid search. When comparing to our method, we sample the same number of random search trials from this set as we have samples on the sparse grid. We calculate the test accuracy for the best model of each. Due to computational constraints, this process is repeated five times and the accuracy for both is averaged. Figure 7 shows the results of this experiments for the three matrices for different sample sizes. Figure 6 shows the same for the tensor of order 3.

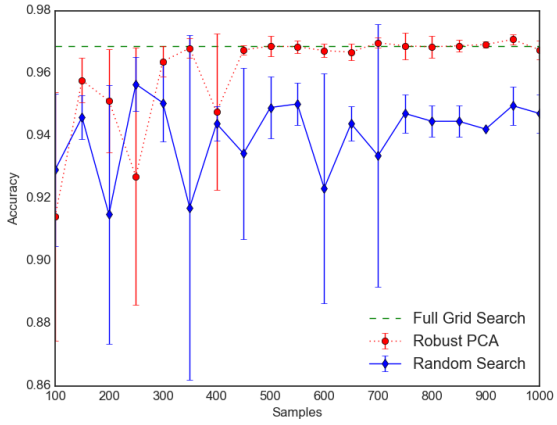
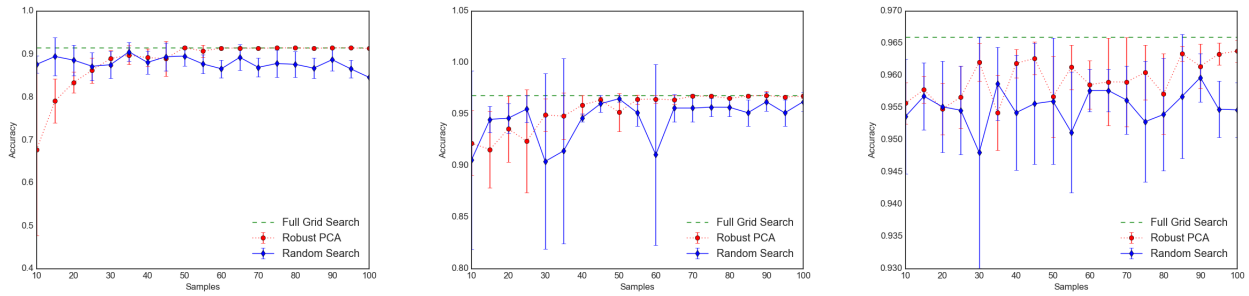


Figure 6. Comparison of accuracy achieved by random search compared to hyperparameter optimization with tensor completion over the number of sampled configurations for a loss tensor spanning all three hyperparameters

As shown in the figures, random search tends to find very different hyperparameter configurations, thus leading to large variance in the test accuracy. Our method using Robust PCA, on the other hand, struggles to find good configurations when only very few samples are known. However, as more samples become available, Robust PCA can stably find very good hyperparameter sets, achieving consistently great test accuracy. In our experiments, Robust PCA would typically surpass random search after 0.4 density level in terms of accuracy and stability, indicated by the low variance in the figures. For specific values, see Table 1. Note that for the same hyperparamter search space size, our method and ran-

Table 1. Comparison of accuracy achieved by random search compared to hyperparameter optimization with tensor completion over the number of sampled configurations for a loss tensor spanning all three hyperparameters

METHOD/SAMPLE SIZE	100	200	300	400	500	600	700	800	900	1000
TENSOR COMPLETION	0.914	0.951	0.964	0.948	0.969	0.967	0.97	0969	0.969	0.969 (**GS**)
RANDOM SEARCH	0.929	0.915	0.951	0.944	0.949	0.923	0.934	0.945	0.942	0.947



(a) Number of hidden units and momentum (LR = 0.001)

(b) Number of hidden units and learning rate (Momentum = 0.9)

(c) Momentum and and learning rate (50 hidden units)

Figure 7. Comparison of accuracy achieved by random search compared to hyperparameter optimization with tensor completion over the number of sampled configurations for three loss matrices spanning two hyperparameters each

dom search are assumed to have similar running time. This is fair since were running the same number of experiments with same number of epochs.

4. Conclusion & Impact

In this project, we have applied Robust PCA to hyperparamter optimization by reconstructing a dense loss tensor from a sparse one. This is due to the inherent low rank structure of the loss tensors. We found that for densities greater than 0.5 the Robust Tensor PCA returns a good estimate of the real loss tensor. This enables us to estimate promising hyperparameter sets while significantly reducing the runtime of the hyperparameter search.

5. Future Work

We are planning to expand on this work by applying our technique to larger neural networks and more complex datasets. This would involve significantly more hyperparameters and therefore more dimensions to optimize over. It would also be fair to evaluate our method with bayesian optimization. As pointed out in the discussion of Figure 7 the Top value metric that we applied does not always convey a good understanding of the quality of loss tensor reconstruction when the performance of the different models is very similar. Therefore, we would like to develop an alternative metric that allows for an unbiased assessment of the algorithm's performance.

Additionally, we would like to compare the performance of Robust Tensor PCA which was used for low rank decomposition in this paper with the CP/Parafac algorithm. Parafac with missing values is outlined in (Tomasi & Bro, 2005) and is currently being integrated into Tensorly.

References

Abdi, H. and Williams, L. J. Principal component analysis. *WIREs Comput. Stat.*, 2(4):433–459, July 2010. ISSN 1939-5108. doi: 10.1002/wics.101. URL https://doi.org/10.1002/wics.101.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(1):281–305, February 2012. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2503308.2188395.

Candes, E. J., Li, X., Ma, Y., and Wright, J. Robust principal component analysis?, 2009.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms, 2012.

Tomasi, G. and Bro, R. Parafac and missing values. *Chemometrics and Intelligent Laboratory Systems*, 75:163–180, 02 2005. doi: 10.1016/j.chemolab.2004.07.003.

Yang, C., Akimoto, Y., Kim, D. W., and Udell, M. Oboe: Collaborative filtering for automl model selection. 2018. doi: 10.1145/3292500.3330909.