NEURAL NETWORKS FOR IMAGE CLASSIFICATION USING QUADRATIC SCALARIZATION AND VARIABLE PROJECTIONS FOR FAST TRAINING

VICTOR PEREYRA

Abstract. We introduce several variants to the task of training large scale neural networks for image classification. The aim is to simplify the network architecture, decrease the number of parameters necessary for a given accuracy, and to enable scaling up via a proposed parallel procedure. For this purpose, we modify the standard usage of vectorizing the image data and instead introduce the idea of employing a quadratic form to scalarize the data in matrix form. This reduces the number of parameters per unit dramatically. We also revisit the idea of using Variable Projections (VARPRO) to eliminate the linear parameters in the output layer, which improves the condition of the problem and reduces the number of iterations required for a given accuracy. Finally, in order to tackle large scale problems and to introduce parallelism, we consider a somewhat different version of dropout and mini-batches, in an asynchronous, perfectly parallel algorithm. By using the trained network on the test set we make a best guess to its labels and with that output we "train" the test set. We exemplify the results of this approach on the *mnist* and *cifar10* classical data sets, obtaining 99.88% and 93.7% accuracy with up to 10x less weights than current results.

1. Introduction

In rodrigob.github.io there is an interesting discussion about the state of the art for image classification. Pointers to the best results are included. For mnist, the current record seems to be 0.21% error using a two hidden layers network and DropConnect [23], followed by [4] (0.23%) and [5] (0.35%). Both [4, 5] have interesting discussions and pointers to previous results and one of the points they make refers to architecture simplicity. They also emphasize that a main obstacle, until recently, was the very long time that NNs required for training in problems with large data sets, such as mnist. The use of gpu's and faster machines have helped to bring those times down considerably, although it is hard to say where we are, since most researchers do not report their training times. Other work that touts simplicity and parsimonious models are [1, 11]. Besides of following the Occam's Razor Principle, an

Date: January 12, 2019.

additional need for parsimonious models is required in applications that are to be deployed in portable, small devices, such as phones and watches.

In this quest for simplicity, we consider a single hidden layer, fully connected Neural Network for the classification of labeled images (X_j, Y_j). The label Y_j is a $K-$vector, with K the number of classes, zero everywhere except for a one pointing to the proper class. The first novelty of our approach is that we keep the image data in matrix form (just as is done in convolutional layers) and for each neuron we consider activation functions with quadratic scalarization terms of the form:

$$T_i(X_j, \mathbf{w_i}) = Sigmoid(\mathbf{w}_i^T X_j \mathbf{w}_i).$$

where $\mathbf{w}_i$ is a vector of parameters. Thus, the kth output looks like:

$$O_k^j = \Sigma_{i=1,}^I a_i^k T_i(X_j, \mathbf{w_i}).$$

where $A = \{a_i^k\}$ is the vector of linear parameters, which we normalize to sum 1 and k runs over the classes.

We use the Variable Projections (VP) method for training [7, 9, 10, 19]. The cost function to minimize is:

$$\Sigma_{k=1}^K \Sigma_{j=1}^N (O_k^j - t_k^j)^2, \tag{1.1}$$

where t_k^j contains the label corresponding to the input data j. In order to decide which is the calculated label, for each j we take the index of the maximum value of O_j^k, since they add up to 1 and therefore can stand for the probability of choosing a given class. If it coincides with the 1 in t^j we declare success and count 1 properly labelled example.

Observe that the nonlinear weights $W = \{\mathbf{w}_i\}$ are common to all the outputs and the only weights that change for each output class are the linear ones, so that this model is in the form of a separable nonlinear least squares problem with multiple right hand sides, as studied in [8, 13, 14, 15]. It is therefore amenable to efficient optimization via Variable Projections. Observe also that by introducing the data inputs as matrices and using just one vector to scalarize the argument of the sigmoid we have reduced the number of nonlinear parameters from the traditional $L^2 \times I$ when presenting the input image as a vector, to $(L \times I)$. With this simple procedure we also conserve the proximity information that is lost when the image is represented as a vector.

Clearly, the savings will be more important for higher resolution and color images. This idea can be extended to higher dimensional tensors with an even more remarkable saving in the number of weights. In Table 1 we describe the main parameters of this problem.

N	# Data images X_j of dimension L^2
K	Labels (0/1) for each data image
I	# Neurons (units) in one hidden layer
W	Matrix of nonlinear weights $L \times I$
A	Matrix of linear weights $I \times K$

TABLE 1. Problem summary

2. Variable projection training

As we indicated above, the problem of finding the best parameters that fit the given data can be posed as a separable nonlinear least squares problem with multiple right hand sides:

$$min_{A,W} \|\Phi(W)A - \tau\|_F, \tag{2.1}$$

where the matrices are defined by $\Phi = \{T_i(X_j, \mathbf{w}_i)\}$, $A = a_i^k$, $\tau = t_k^j$, $W = \{w_{ij}\}$, and F stands for the Frobenius norm. From the references above we describe the Variable Projection algorithm that consists of two steps:

(1) For fixed W, equation 2.1 is a linear least squares problem with multiple right hand sides, whose minimal solution is given by the pseudo-inverse of $\Phi : W = \Phi^+\tau$. **This is the key difference with an straightforward algorithm that ignores the separability structure. In fact, what this says is that the linear parameters can (and should) be obtained from the nonlinear ones, indicating that they are correlated and that treating them as independent parameters can only lead to ill-conditioning, when approaching a solution to the minimization problem. As a bonus one has less weights to guess.**

(2) Replacing this in 2.1 we then eliminate the linear parameters A obtaining the Variable Projection functional:

$$min_W \|(\Phi\Phi^+ - I)\tau\| = min_W \ R(W). \tag{2.2}$$

Below we discuss in detail how to evaluate R by using the truncated Singular Value Decomposition that is enough for optimization methods that do not use derivatives. We then describe the evaluation of the residual's Jacobian for use in quasi-Newton optimization methods, such as Levenberg-Marquardt.

3. Optimization by quasi-Newton methods

A step up in speed of convergence over simple gradient descent methods can be gained by using Newton's method to solve the optimality condition equations. However, that requires second order derivatives information, which, for a problem in many dimensions can be very expensive and cumbersome to calculate. Fortunately, for nonlinear least squares (NLLS) there

are simplifications, such as Gauss-Newton and Levenberg-Marquardt, that provide faster convergence than gradient descent still using only first order derivative information.

What is required for these methods is the Jacobian of the residual vector, which, for small residual problems gives a good approximation to the Hessian of the problem by neglecting the second derivative terms. These algorithms apply to standard NLLSQ problems as well as to the Variable Projection approach. We derive now the Jacobian for the VP functional 2.2:

$$J = \partial(\Phi\Phi^+ - I)\tau/\partial W = -\partial P_\Phi^\perp \tau/\partial W,$$

where $P_\Phi^\perp = I - \Phi\Phi^+$ is the orthogonal complement of the projector to the space of columns of Φ.

In [7, 9, 10, 12] it is described how to differentiate an orthogonal projector in a multi-variable scenario:

$$\partial P_\Phi^\perp \tau/\partial W = -[P_\Phi^\perp D\Phi\Phi^\dagger + (P_\Phi^\perp D\Phi\Phi^\dagger)^T]\tau.$$

Kaufman argues that for small residuals the second term is negligible and that has been shown to make the VP iterations no more expensive than if one would use the full functional and still produce a convergence similar to VP in most cases; thus, we will drop this second term in our approximation:

$$\partial P_\Phi^\perp/\partial W \simeq -P_\Phi^\perp D\Phi\Phi^\dagger\tau = -P_\Phi^\perp D\Phi A.$$

The first significant issue is that of calculating $\partial\Phi_{ji}/\partial w^l$. By the chain rule we have:

$$\partial\Phi_{ji}/\partial w_{ik} = DT[X_{jk}w_i^T + w_i X_j^k],$$

where X_k, X^k stand for the kth row and column of the matrix X_j, respectively. In order to complete the calculation of the Jacobian we need to pre-multiply this quantity by the orthogonal projector and post-multiply it by W. If we used the SVD of Φ to calculate W and save it, then $P_\Phi^\dagger = I - Udiag(1)_r U^T$, where $diag(1)_r$ stands for $\Lambda\Lambda^\dagger$ with r equal to the numerical rank of Φ. Thus, basically we need to pre-multiply $D\Phi W$ by the first r rows of U^T and then by the first r columns of U. Finally, we subtract $D\Phi W$.

At this point is good to back up a bit and look to the dimensions of these various tensors (Table 2). Here, we recall, N is the number of training images, I is the number of units in the hidden layer, K is the number of outputs (classes), L is the number of pixels in each direction and $ipar = I * (K + 1)$ is the total number of nonlinear parameters. Thus, the dimensions of $DP_\Phi^\dagger$ are $[N, K, ipar]$.

4. Large scale optimization by decomposition

It is common practice now to decompose large networks in smaller networks that can be trained independently. In [18] we developed an algorithm

Tensor	Dimensions
Φ	$[N,\ I+1]$
$D\Phi$	$[N, I+1, ipar]$
U	$[N, r]$
$P_\Phi^\dagger D\Phi$	$[N, I+1, ipar]$
W	$[I+1, K]$

TABLE 2. Dimensions of the various tensors in the Jacobian calculation

for the nonlinear least squares inversion of large scale geophysical problems that is very effective and highly paralellizable in a distributed environment. We describe now a variant of that algorithm that is applicable to the training of large networks. Similar approaches have been used in recent times, such as DropConnect and distributed belief.

The idea is to break the large target network into smaller ones, by considering subsets of units and training examples. In our implementation we consider random choices of examples for a fixed number of mini-batches. For each mini-batch we consider an independent set of parameters. We train each mini-network fully and at the end we use their parameters to assemble the full network. Since we are using the VARPRO approach, only the nonlinear parameters are transmitted, while the final linear parameters are calculated for the whole network by a linear least squares step, thus providing a "glueing" for the sub-networks. This approach is embarrassedly parallel and leads to a totally distributed system that should scale to large problems. We test it later on a multicore box with 8 cpu's. Of course, the process can be further accelerated by using gpu's.

5. BOOTSTRAPPING

Once the whole network is trained as explained above, we use it to run the test set. Ideally, and to be consistent with the VARPRO philosophy, we should recalculate the linear parameters for the test data set, since they are different from those of the training set. Unfortunately, to do that we would need to use the labels of the testing set that is not allowed. What we do instead is to use the trained network to estimate the labels by processing the outputs. Let v_i be the $10-$vector of output corresponding to the test i-th data, normalized to add up to 1. Then we calculate the two largest values, v^1, v^2. If

$$v^1 - v^2 > 0.2 * v^1,$$

then we guess the label to be the index of v^1, otherwise the result is ambiguous and we choose a label at random. We use these guessed labels to improve the weights of the network for the testing set by doing a full

training pass as it was done earlier for the training set. This simple algorithm has some similarity to gradient boosting [6], although repetition of the process in our case does not lead to further improvement.

6. Applications

6.1. **Mnist.** We use the classical *mnist* data set to exercise the above algorithm. We start with $50,000$ labeled examples of hand written digit 28×28 pixel grey images. We will use a partitioning scheme akin to dropout and mini-batches [18], in which we select a number of groups (104 in the experiments below) and partition the data and the maximum number of neurons in equal size sub-networks. We choose $I = 5200$ for the total number of neurons on a single hidden layer. Thus the sub-networks in this small example will have during training $2,000$ labeled examples and 50 neurons. We use Variable Projections and Levenberg-Marquardt as the optimization algorithm for minimization. After training these 104 independent networks, we assemble them together and calculate the linear parameters of the assembled network to obtain its precision in the training set. Then we use that network to calculate the accuracy on the testing data set. This model results in a 90.15% success rate in testing. This is, of course, not competitive with the state of the art, so we use it to estimate the labels of the testing set and then run a full learning cycle for the test set using these guessed labels. Then a considerable improvement is obtained as we can see in Table 4.

Observe that due to the Variable Projection approach, in the whole model only the nonlinear parameters are inherited from the sub-networks training, while the linear parameters are recalculated, providing a glueing step for the independent sub-networks. Observe also that the training of the independent sub-networks can be done completely in parallel since there is no communication necessary between processes.

We use Python optimize.leastsq, the wrapper for the classical Fortran code lmder from MINPACK that implements a Levenberg-Marquardt nonlinear least squares optimizer. It is important to remark that we obtain robust convergence in about fifteen iterations for all the mini-network's training and we do just one pass through the data. To put this result in perspective we introduce it in the comparative Table of [22], modified to show the number of parameters used by each model (if we understand correctly from the vague descriptions). By the way, the state of the art as of today is 99.79% success rate ("Regularization of Neural Networks using DropConnect", Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus, from Rodrigo Benenson github list), using two hidden layers and regularization of NN with DropConnect.)

We also use the Python module multiprocessing to start several independent processes at the same time. These processes perform the optimization for a mini-batch and they are totally independent, reading and writing their

Train Total	50,000
Test	10,000
Units Total	5200
Sub-nets	104
Mini-batches	2000
Units in sub-nets	50

TABLE 3. Network meta-parameters for MNISTt

Method	Layers, Units	#Params.	Error %
Standard NN	2, 800	1.25 M	1.60
SVM Gaussian Kernel	NA	NA	1.40
Dropout NN	3, 1024	8.125 M+	1.35
Dropout NN	3, 1024	8.125 M+	1.25
Dropout NN + max-norm	3, 1024	8.125 M +	1.06
Dropout NN + max-norm	3, 2048	16.25 M +	1.04
Dropout NN + max-norm	2, 4096	32.5 M +	1.01
Dropout NN + max-norm	2, 8192	65 M +	0.95
DropConnect	2, 1600	1.26 M	0.21
This algorithm	**1, 5200**	**0.151 M**	**0.12**

TABLE 4. Comparison of different models on MNIST

non-overlapping parameters from a central repository. We ran this experiment on a 8 core, 100 GB RAM 3325 GH Linux box. Upon completion the linear parameters for the assembled full network are calculated, and the classification accuracy is obtained. Then we estimate the labels of the test set and we use these calculated labels to fully train the network for the testing data set.

We observe that both the number of meta-parameters and data pre-processing have been minimized. We only scale the images to $[0, 1]$ and use the standard mnist data set, without deformations. The network architecture consists of a single layer fully connected network with sigmoid activation units. The various parameters describing the full network and its decomposition in mini-networks are listed in Table 3. Although we use more units than the previous best result, since the number of parameters per unit is $1/28$ less our 57% improvement in accuracy is obtained with a simpler architecture, no data massaging and almost one order of magnitude less parameters.

6.2. **Cifar10.** This is a more challenging data set, consisting of $60,000$ labeled color images of 32×32 pixels, $X[32, 32, 3]$. This 3-dimensional tensor is now scalarized by combining the three color planes with weights and then reducing the resulting matrix with the quadratic form scalarization:

$$\Sigma_i \Sigma_j w_i (\Sigma_k X_{ijk} w_k^c) w_j + w^b,$$

thus using 36 parameters per unit (including bias), instead of the 3,074 required by the usual flattening of the image to convert it into a vector.

WORK IN PROCESS.

7. Discussion

Industry is considering ever expanding data sets and corresponding large scale NN for solving **automatically** many classes of problems. Thus, although very large clusters of computers aided by large networks of GPU's and the combination of insightful new techniques have lead to the deployment of successful large scale practical applications, it is of interest for future even larger applications to consider further variants that approach the Occam's Razor Principle and reduce the computational complexity and storage footprint of algorithms, while providing comparative accuracy. There is also interest in smaller models than can be deployed in small portable devices, instead of having to connect with the cloud to access larger computational resources [1, 11].

We have described four variants to common practice in the very successful current crop of algorithms for defining and training large scale neural networks and we have exemplified them on two commonly used data sets, showing that they can help in providing more parsimonious models for the same final accuracy.

Observe that the idea is applicable, with additional gains, to tensor data of higher dimension. If all the dimensions are equal, then, just one vector of parameters of that dimension is necessary per neuron, instead of the dimension raised to the order of the tensor. Otherwise we would need as many vectors as different dimensions are present. Still, in the worse case, we would have the sum of the dimensions as the number of parameters, instead of their product.

8. Appendix

8.1. **Optimization by intelligent search.** If we calculate the Singular Value Decomposition of the $N \times I$ matrix $\Phi = U \Lambda V^T$, then

$$R = \|U \begin{bmatrix} 0 & 0 \\ 0 & I_{N-\rho} \end{bmatrix} U^T \tau\|_F = \| \begin{bmatrix} 0 & 0 \\ 0 & I_{N-\rho} \end{bmatrix} U^T \tau\|_F,$$

where ρ is the numerical rank of Φ and we can eliminate the first U since it is an orthogonal matrix. Thus, if we call $\widetilde{\tau} = U^T \tau = \begin{bmatrix} \tilde{\tau}_\rho \\ \tilde{\tau}_{N-\rho} \end{bmatrix}$, then:

$$R = \|\tilde{\tau}_{N-\rho}\|_F.$$

Alternatively, we can follow [12] and use the QR decomposition instead of the SVD, keeping Q in implicit form. In this way, instead of the large $N \times N$

matrix U we only need $N \times I$ storage to keep the Householder vectors. The rest of the calculation is the same.

We can now use any search method to minimize $R(W)$. Observe that by choosing ρ appropriately we will be regularizing the problem in case of severe ill-conditioning or actual rank deficiency, which is an indicator also that too many neurons are been used for the given data set.

In reality, we need to calculate $\Phi\Phi^{+}\tau$ and manipulate the result. For every observation and every output, the result of this calculation might be negative, specially when far from the solution. Thus we include a squashing final step using a sigmoid of the outputs. Since the $\tau's$ are all zeroes except for a 1 that identifies the class, this is a reasonable choice.

This is all that is needed for intelligent search optimization algorithms that do not require derivatives. We can go a step further in complexity by calculating the derivatives of R by the recipe given in [7, 9], which would them enable us to use faster converging algorithms such as gradient descent, Gauss-Newton or Levenberg-Marquardt.

8.2. **Optimization by gradient descent.** To optimize by gradient descent we need the first derivatives of $R(W)$. From [7, 9, 19] we know that:

$$\frac{1}{2}\nabla R = WD\Phi^{T}\tilde{\tau}.$$

If we call $G^{T} = D\Phi^{T}\tilde{\tau}$, then

$$\frac{1}{2}\nabla R = GW$$

So, the main issue now is how to calculate the three-dimensional tensor $D\Phi$. First of all, we observe that this tensor has dimensions $[N, I+1,\ ipar]$, where $ipar = I*(L+1)$. Many of the derivatives are zero, since each function only depends upon $L+1$ parameters. Thus, of the $(I+1)*ipar$ possible derivatives only $ipar$ are nonzero for each neuron. Consequently, such as we did in the original paper and program VARPRO [7], we can use compressed storage for this sparse tensor. Since the number of parameters per function is fixed, we do not need an incidence matrix, just careful indexing. The derivative of a sigmoid function is: $T' = sech^2$ and in our case, each column j of Φ is generated by $T_j = tanh(w_j^{T}\tau w)$. Thus we have the following scheme for calculating and storing the derivatives:

(1) Define $K_j = [k_j,\ k_{j+1}]$, with $k_1 = 1,\ k_{j+1} = k_j + L + R + 1$.
(2) $D\Phi^{k}_{nj} = 0$ if $k \not\subseteq K_j$.
(3) $D\Phi^{k}_{nj} = T'_{n,j}\tau_{nk_l k_r} w^{r}_{j,k_r},\quad k\epsilon[k_j,\ k_j + L - 1]$.
(4) $D\Phi^{k}_{nj} = T'_{n,j}\tau_{nk_l k_r} w^{l}_{j,k_{lr}},\quad k\epsilon[k_j + L,\ k_{j+1} - 2]$.
(5) $D\Phi^{k}_{nj} = 1,\quad k = k_{j+1} - 1$.
(6) $G^{T} = D\Phi^{T}\tilde{\tau} = \Sigma_{n=\rho+1}^{N} D\Phi^{k}_{nj}\tilde{\tau}_n$.
(7) $(GW)_k = \Sigma_{j=1}^{I+1} G_j^{Tk} W_j$.

(8) $\frac{1}{2}\nabla R = \Sigma_{k=1}^{K}(GW)_k$.

References

[1] Lei Jimmy Ba, Rich Caruana, *Do Deep Nets Really Need to be Deep?*. arXiv:1312.6184 (2014).

[2] E. Bendersky, *The Softmax function and its derivative.* https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative (2016).

[3] Y. Bengio and X. Glorot, *Understanding the difficulty of training deep feedforward neural-networks.* AISTATS (2010).

[4] D. Ciresan, U. Meier, L. M. Gambardella and J. Schmidhuber, *Deep big simple neural nets excel on handwritten digit recognition.* Neural Computation (2010).

[5] D. Ciresan, U. Meier and J. Schmidhuber, *Multi-column deep neural networks for image classification.* Technical Report IDSIA-04-12, CVPR (2012)

[6] Friedman, J. *Greedy boosting approximation: a gradient boosting machine.* Ann. Stat. 29, 1189–1232 (2001).

[7] G. H. Golub and V. Pereyra, *The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate.* SIAM J. Numer. Anal. **10**:413-432 (1973).

[8] G. H. Golub and R. Le Veque, *Extensions and uses of the variable projection algorithm for solving nonlinear least squares problems.* Proc.Army Numerical Analysis and Computers Conf. (1979).

[9] G. H. Golub and V. Pereyra, *Separable nonlinear least squares: The variable projection method and its applications.* Inverse Problems **19**:R1-R26 (2003).

[10] P. C. Hansen, V. Pereyra and G. Scherer, *Least Squares Data Fitting with Applications.* Johns Hopkins University Press, Baltimore (2013).

[11] S. H. Hasanpour, M. Rouhani, M. Fayyaz, M. Sabokrou, *Let's keep it simple, Using simple architectures to outperform deeper and more complex architectures. https://arxiv.org/pdf/1608.06037.pdf (2016).*

[12] L. Kaufman, Solving separable nonlinear least squares problems with multiple data sets. In [19] (2010).

[13] L. Kaufman and G. Sylvester, *Separable nonlinear least squares with multiple right hand sides.* SIAM J. on Matrix Analysis and Applications **13**:68-89 (1992).

[14] L. Kaufman, G. Sylvester and M. Wright, *Structured linear least squares problems in system identification and separable nonlinear data fitting.* SIAM J. on Optimization **4**:847-871 (1994).

[15] K. M. Mullen and I. H. M. van Stokkun, *TIMP: An R package for modeling multi-way spectroscopy measurements.* J. Stat. Software **18**:1-46 (2007).

[16] S. Mahadevan, S. Giguere and N. Jacek, *Basis adaptation for sparse nonlinear reinforcement learning.* Proc. **27** AAAI Conference on Artificial Intelligence (2013).

[17] D. O'Leary and B. Rust, *Variable projection for nonlinear least squares.* Comp. Opt. and Appl. **54**:579-593 (2013).

[18] V. Pereyra, *Asynchronous distributed solution of large scale nonlinear inversion problems.* J. Applied Numerical Mathematics **30:**31-40 (1999).

[19] V. Pereyra and G. Scherer, *Exponential data fitting. In Exponential Data Fitting and its Applications*, Editors V. Pereyra and G. Scherer, Bentham Books, Oak Park. Ill. (2010).

[20] V. Pereyra, G. Scherer and F. Wong, *Variable projection neural network training.* Mathematics and Computers in Simulation **73**:231-243 (2006).

[21] P. Simard, D. Steinkraus and J. Platt, *Best practices for for convolutional neural networks applied to visual document analysis.* Proceedings of the Seventh International Conference on Document Analysis and Recognition **2**:958-962 (2003).

[22] N. Srivastava, G. Hinton, A. Krishevsky, I. Sitskever and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting.* J. of Machine Learning Research **15**:1929-1958 (2014).

[23] L. Wn, M. Zeiler, S. Zhang, Y. LeCun and R. Fergus, *Regularization of neural networks using DropConnect.* ICML (2013).