Lifted Neural Nets

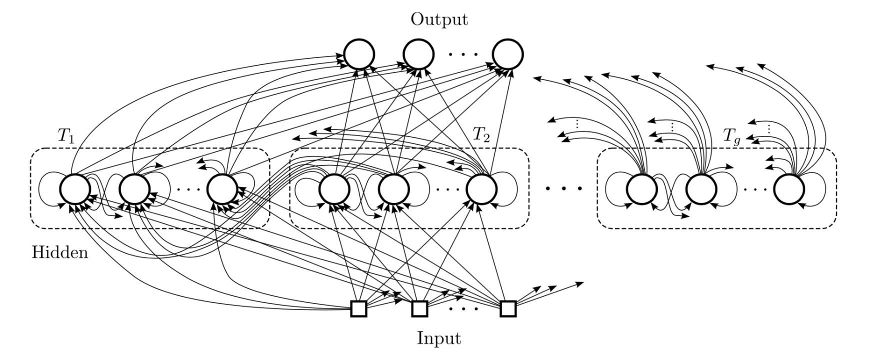
BoydFest 2018

Laurent El Ghaoui*
Joint work† with G. Negiar**, A. Askari**, R. Sambharya**, T. Roosta***
March 2, 2018

* EECS and IEOR Dept., UC Berkeley
** EECS Dept., UC Berkeley
*** SumUp Analytics, Inc
† Based on a presentation at NIPS Workshop on Optimization, 2017

Feedforward Networks

Why I have trouble with neural nets



A picture taken from [Koutnik et al., 2014].

Feedforward Networks

Given input point $x \in \mathbf{R}^n$, predicted output:

$$\hat{y}(x) = x_{L+1},$$

$$x_{l+1} = \phi_l(W_l x_l + b_l), \quad l = 0, \ldots, L,$$

with $x_0 = x$.

- $l = 1, \ldots, L$ denotes layer index;
- (W_l, b_l)'s are the parameters of the NN;
- ϕ_l's given non-linear maps ("activation functions");
- x_l's "state" ("hidden" or "feature" vector)—note size may vary from layer to layer.

For **multiple** input points contained in the $n \times m$ matrix X: set $\hat{Y}(X) = X_{L+1}$, where

$$X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}^T), \quad l = 0, \ldots, L,$$

with initial value $X_0 = X$.

Training problem

$$\begin{array}{rl} \displaystyle\min_{(W_l,b_l)_{l=0}^{L},(X_l)_{l=1}^{L+1}} & \mathcal{L}(Y, X_{L+1}) + \displaystyle\sum_{l=0}^{L} \pi_l(W_l) \\ \text{s.t.} & X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}_m^T), \quad l = 0, \ldots, L, \\ & X_0 = X, \end{array}$$

where

- $\mathcal{L}$ is a loss function;
- π_l's are penalty functions;
- $X = [x_1, \ldots, x_m] \in \mathbf{R}^{n \times m}$ contains m input points $x_i \in \mathbf{R}^n$
- $Y = [y_1, \ldots, y_m] \in \mathbf{R}^{p \times m}$ contains the corresponding responses (or labels)

Solving the training problem

To solve the training problem:

- eliminate X-variables via the recursion

$$X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}_m^T), \quad l = 0, \ldots, L, \quad X_0 = X.$$

- Minimize the resulting objective function of the (W, b)-variables.

The complicated structure of the resulting objective function points to stochastic gradients as the only viable solution method.

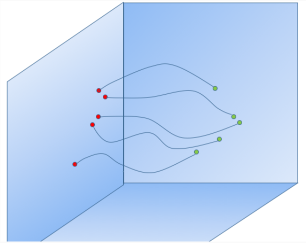
- Can take a long time to converge.
- Can fail to converge due to numerical issues (vanishing / exploding gradients)
- Difficult to handle constraints.

Side note: training NNs is an end-to-end control problem

Consider a dynamical system with state $x(t)$ and control variable $u(t)$

$$x(t+1) = \phi(u(t)), \quad t = 0, 1, 2, \ldots$$

Assume (WLOG) that all the layers, including the last one, have the same dimension, n; then $X, Y \in \mathbf{R}^{n \times m}$.



The training problem can be formulated as an *end-to-end control* synthesis problem: find a linear, state-feedback, time-varying control law $u(t) = W(t)x(t)$ such that each input point (column in X) is mapped onto the corresponding output (column in Y).

Lifted Framework

Lifted framework

Recall training problem:

$$\min_{(W_l,b_l)_{l=0}^{L},(X_l)_{l=1}^{L+1}} \quad \mathcal{L}(Y, X_{L+1}) + \sum_{l=0}^{L} \pi_l(W_l)$$
$$\text{s.t.} \quad X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}_m^T), \;\; l = 0, \ldots, L, \;\; X_0 = X.$$

Proposed approach:

- Keep the X-variables;
- Penalize the constraints, first representing activations as “argmin” maps;
- Solve via block-coordinate descent.

RELU activation as an "argmin" map

For a vector u, RELU defined as

$$\phi(u) = \max(0, u),$$

with max acting component-wise on the vector input.

RELU can be represented as the solution map of an optimization problem:

$$\phi(u) = \max(0, u) = \arg\min_{v \geq 0} \|v - u\|_2.$$

Hence the activation condition

$$X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}_m^T)$$

can be equivalently written

$$X_{l+1} \in \arg\min_{Z \geq 0} \|Z - W_l X_l - b_l \mathbf{1}^T\|_F^2.$$

Example: multi-layer ridge regression with RELUs

$$\begin{aligned}\min_{(W_l,b_l)_{l=0}^L,(X_l)_{l=1}^L} \quad & \|Y - X_{L+1}\|_F^2 + \\ & \sum_{l=0}^{L} \left(\lambda_{l+1} \|X_{l+1} - W_l X_l - b_l \mathbf{1}^T\|_F^2 + \rho_l \|W_l\|_F^2 \right) \\ \text{s.t.} \quad & X_l \geq 0, \;\; l = 1, \ldots, L, \;\; X_0 = X.\end{aligned}$$

where $(\lambda_l)_{l=1}^{L+1}$ are given hyper-parameters (WLOG can assume all equal).

Solve problem via block coordinate descent (BCD), *i.e.* alternate minimization over (W, b)- and X-variables:

- For fixed (W, b)-variables, the problem is is a (matrix) non-negative least-squares problem. The problem is fully *parallelizable across the data points*.
- For fixed X-variables, the problem is a set of parallel (matrix) ridge regression problems, and is *parallelizable across layers and data points*.

Variational representation of activations

Consider the following condition on a generic activation function $\phi : \mathbf{R}^k \to \mathbf{R}^h$.

> ***JC Condition.*** *The activation function $\phi : \mathbf{R}^k \to \mathbf{R}^h$ satisfies the jointly convex (JC) condition if it can be represented as follows:*
>
> $$\phi(u) = \arg\min_v \ \mathcal{D}_\phi(u, v),$$
>
> *where $\mathcal{D}_\phi : \mathbf{R}^k \times \mathbf{R}^h \to \mathbf{R}$ is a jointly convex function, which is referred to as a JC-divergence associated with the activation function.*

Note that for the JC condition to hold, the activation function needs to be monotone increasing.

Examples of JC activations

RELU:

$$\max(u,0) = \arg\min_v \ \mathcal{D}_\phi(u,v) := \begin{cases} \|v-u\|_2^2 & \text{if } v \geq 0, \\ +\infty & \text{otherwise.} \end{cases}$$

"leaky" ReLU with parameter $\alpha \in (0,1)$:

$$\max(u/\alpha, u) = \arg\min_v \ \|v-u\|_2^2 \ : \ v \geq (1/\alpha)u.$$

Piece-wise sigmoïd:

$$\min(1, \max(0,u)) = \arg\min_v \ \|v-u\|_2^2 \ : \ 0 \leq v \leq \mathbf{1}.$$

Examples of JC activations

Euclidean projection of a real vector $u \in \mathbf{R}^k$ onto the probability simplex in $\mathbf{R}^k$:

$$\phi(u) = \arg\min_v \ \|v - u\|_2^2 \ : \ v \geq 0, \ \ v^T \mathbf{1} = 1.$$

Max-pooling: for example

$$\phi(u) = (\max_{1 \leq i \leq p_1} \ u_i^{(1)}, \max_{1 \leq i \leq p_2} \ u_i^{(2)}) \in \mathbf{R}^2. \tag{1}$$

Then

$$\phi(u) = \arg\min_v \ \mathbf{1}^T v + \mathbf{1}^T (u - Dv)_+,$$

where D is an appropriate block-diagonal matrix of size $p \times 2$ that encodes the specifics of the max-pooling, namely in our case $D = \mathbf{diag}(\mathbf{1}_{p_1}, \mathbf{1}_{p_2})$.

Variational representation of activations

We express the activation at layer l as

$$X_{l+1} \in \arg\min_{Z} \; D_l(W_l X_l + b_l \mathbf{1}^T, Z), \quad l = 0, \ldots, L.$$

where D_l's are the JC-divergences associated with ϕ_l's.

Lifted neural networks

Replace constraints with penalties

$$\min_{(X_l)_{l=1}^{L+1},(W_l)_{l=0}^{L}} \quad \mathcal{L}(Y, X_{L+1}) + \sum_{l=0}^{L} \Big(\lambda_{l+1} D_l(W_l X_l + b_l \mathbf{1}^T, X_{l+1}) + \pi_l(W_l) \Big) \;:\; X_0 = X.$$

with $\lambda_1, \ldots, \lambda_{L+1}$ given positive hyper-parameters.

Solve problem via block coordinate descent (BCD):

- For fixed (W, b)-variables, the problem is convex in the X-variables X_l, $l = 1, \ldots, L$, and is fully *parallelizable across the data points*.
- For fixed X-variables, the problem is convex in the (W, b)-variables, and is *parallelizable across layers and data points*.

Prediction rule in a standard NN

In a standard NN:

$$\hat{y}(x) = \min_y \; \mathcal{L}(y, x_{L+1}) \;:\; x_{l+1} = \phi_l(W_l x_l + b_l), \;\; l = 0, \ldots, L, \;\; x_0 = x,$$

where

- weights are now *fixed*;
- $y \in \mathbf{R}^p$ is a *variable*.

Trivially reduces to the standard prediction rule, $\hat{y}(x) = x_{L+1}$, where x_{L+1} is obtained via the recursion above.

Prediction rule in lifted framework

$$\hat{y}(x) = \arg \min_{y,(x_l)_{l=1}^{L+1}} \mathcal{L}(y, x_{L+1}) + \sum_{l=0}^{L} \lambda_{l+1} D_l(W_l x_l + b_l, x_{l+1}) \ : \ x_0 = x.$$

where

- weights are now *fixed*;
- $y \in \mathbf{R}^p$ is a *variable*.

- Can solve as convex problem.
- Not the same as the standard rule!
- Activation now depends on data.

Links with matrix factorization

Lifted model can be written

$$\min_{\tilde{W}\in\mathcal{W},\, \tilde{X}\in\mathcal{X}} \mathbf{L}(\tilde{W}\tilde{X}, \tilde{Y})$$

where

- $\tilde{W}$ (resp. $\tilde{X}$) is a matrix containing the (W, b) (resp. X-variables);
- $\tilde{Y}$ contains the input and output matrices;
- $\mathbf{L}$ is a loss function, encoding that of last layer, and the JC-divergences representing the activation functions;
- Sets $\mathcal{X}$, $\mathcal{W}$ are convex.

- Connects with generalized low-rank models [Udell et al., 2016];
- can solve using alternative minimization (BCD);
- covers may extensions such as recurrent NNs, attention models, etc.

Links with Lagrange relaxations

We can represent any strictly monotone activation

$$v = \phi(u) \iff B_\phi(u, v) \le 0,$$

where B_ϕ is bi-convex:

$$B_\phi(u, v) := F(u) + F^*(v) - u^T v,$$

where F is a convex function:

$$F(u) := \sum_{i=1}^{p} \int_0^{v_i} \phi_i(\xi)\, d\xi,$$

with F^* the Fenchel conjugate of F.

In this setting, lifted model leads to a Lagrange relaxation; X-update prblem is then not jointly convex, but BCD methods apply.

Extensions and variants

Optimizing over activation functions

Parametrize activations via

$$\phi(u) = \max(\alpha, \min(\beta, u)) = \arg\min_v \ \|v - u\|_2^2 \ : \ \alpha \le v \le \beta,$$

where $\alpha \le \beta \in \mathbf{R}^k$ are now variables. In multi-layer ridge regression:

$$\begin{array}{rl} \min\limits_{(W_l,b_l)_{l=0}^L,(X_l)_{l=1}^L,(\alpha_l,\beta_l)_{l=1}^L} & \|Y - X_{L+1}\|_F^2 + \\ & \sum\limits_{l=0}^{L} \left(\lambda_{l+1} \|X_{l+1} - W_l X_l - b_l \mathbf{1}^T\|_F^2 + \rho_l \|W_l\|_F^2 \right) \\ \text{s.t.} & \alpha_l \mathbf{1}^T \le X_l \le \beta_l \mathbf{1}^T, \ \ l = 1, \ldots, L, \ \ X_0 = X. \end{array}$$

Update of X-variables can be done jointly with that of scale variables $(\alpha_l, \beta_l)_{l=1}^L$, and the resulting problem is jointly convex.

Unitary networks

Idea of unitary constraints on W_l's proposed in [Arjovsky et al., 2016]. In the lifted model:

$$\begin{aligned}
\min_{(W_l,b_l)_{l=0}^{L},(X_l)_{l=0}^{L+1}} \quad & \|X_{L+1} - W_L X_L\|_F^2 + \rho\|W_L\|_F^2 + \\
& \lambda \sum_{l=0}^{L-1} \|X_{l+1} - W_l X_l\|_F^2 + \rho\|W_0\|_F^2 \\
\text{s.t.} \quad & W_l^T W_l = I_q, \;\; l = 1, \ldots, L-1, \\
& X_l \geq 0, \;\; l = 1, \ldots, L, \;\; X_0 = X, \;\; X_{L+1} = Y.
\end{aligned}$$

Unitary constraints on matrices W_l's is a form of regularization.

- Updating W-variables is a simple SVD.
- Updating X-variables can be done in closed-form.

Unitary networks

- W-update: orthogonal Procrustes problem

$$\begin{aligned} W_l &= \arg \min_{W \in \mathbf{R}^{q \times q}} \|X_{l+1} - WX_l\|_F \ : \ W^T W = I_q \\ &= \arg \max_W \ \mathbf{Tr}\, WX_l X_{l+1}^T \ : \ W^T W = I_q. \end{aligned}$$

  Can solve via SVD of $M_l := X_{l+1} X_l^T$. In typical architectures, these matrices are of order $\approx 100 - 500$.

- X-update: with RELUs, simple expression for intermediate layers

$$X_l^+ = \phi(\frac{1}{1+\lambda} W_l^T X_{l+1} + \frac{\lambda}{1+\lambda} W_{l-1} X_{l-1}), \quad l = 1, \ldots, L-1.$$

Input matrix completion

Can allow *partially known* entries in X to be variables in the problem:

$$\begin{aligned} \min_{(W_l, b_l)_{l=0}^{L}, (X_l)_{l=0}^{L}} \quad & \|Y - X_{L+1}\|_F^2 \\ & + \sum_{l=0}^{L} \left(\lambda_{l+1} \|X_{l+1} - W_l X_l - b_l \mathbf{1}^T\|_F^2 + \rho_l \|W_l\|_F^2 \right) \\ \text{s.t.} \quad & X_l \geq 0, \quad l = 1, \ldots, L, \quad X_{\text{low}} \leq X_0 \leq X_{\text{up}}. \end{aligned}$$

- The only difference being that X_0, which was fixed to the input X before, is now a variable.
- At test time, we may also allow for X_0 to be a variable.

Robustness

Assume X is unknown-but-bounded: $X \in \mathcal{X} \subseteq \mathbf{R}^{n \times m}$.

Robust counterpart [Ben-Tal et al., 2009] of training problem:

$$\begin{array}{rl} \min\limits_{(W_l,b_l)_{l=0}^L,(X_l)_{l=1}^L} \max\limits_{X \in \mathcal{X}} & \|Y - X_{L+1}\|_F + \sum\limits_{l=0}^{L} \|X_{l+1} - W_l X_l - b_l \mathbf{1}^T\|_F \\ \text{s.t.} & X_l \geq 0, \quad l = 1, \ldots, L, \quad X_0 = X. \end{array}$$

First layer problem is modified to

$$\max_{X \in \mathcal{X}} \|X_1 - W_0 X - b_0 \mathbf{1}^T\|_F$$

For example, with the uncertainty set

$$\mathcal{X} = \{X + \Delta \ : \ \|\Delta\| \leq \rho_0\}$$

with $\|\cdot\|$ the largest singular value norm, the expression above reads

$$\|X_1 - W_0 X - b_0 \mathbf{1}^T\|_F + \rho_0 \|W_0\|.$$

Numerical Results

MNIST

MNIST dataset:

- 70,000 images total, we randomly split to 60,000 training images, 10,000 test images;
- 10 classes (digits 0-9);
- Preprocess the data by zero-meaning and scale to unit variance
- Use RELUs throughout;
- For the last layer, use a cross entropy loss for training and a softmax function to ensure our output is a probability distribution over classes.
- Two 1-layer networks with 300, 1000 hidden units, two 2-layer $500-150$, $300-100$ hidden units and one 4-layer network were tested.

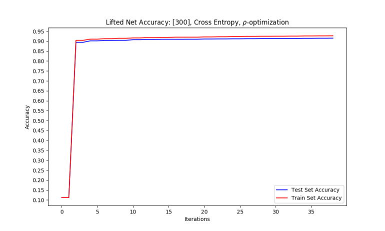
Error rates

Architecture	Our Model	NN [random]	NN [init]
$28 \times 28 - 300 - 10$	0.102 ± 0.001	0.022 ± 0.001	$\mathbf{0.0210 \pm 0.0017}$
$28 \times 28 - 1000 - 10$	0.096 ± 0.004	0.019 ± 0.001	$\mathbf{0.0182 \pm 0.0007}$
$28 \times 28 - 300 - 100 - 10$	0.139 ± 0.003	0.071 ± 0.015	$\mathbf{0.0224 \pm 0.0005}$
$28 \times 28 - 500 - 150 - 10$	0.128 ± 0.002	0.080 ± 0.025	$\mathbf{0.0218 \pm 0.0005}$
$28 \times 28 - 500 - 300 - 150 - 100 - 10$	0.148 ± 0.002	0.83 ± 0.07	$\mathbf{0.0223 \pm 0.0005}$

Error rate on the test set using different networks, best result is highlighted in boldface. NN[random] is a standard neural network with random initialization while NN[init] is a neural network initialized with the weights and biases learned from training our model.The neural networks were trained for 20 epochs using RMSprop in Tensorflow.

Test and training accuracy vs. # BCD iterations

Model: lifted NN using 1 hidden layer with 300 nodes

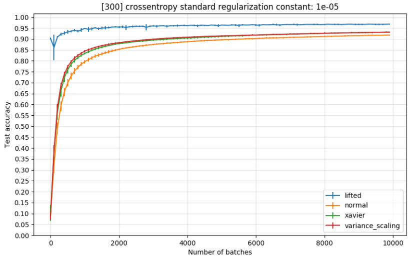


- x-axis is # iterations, 1 iteration is either a W update or an X-update, so a total of 40 iterations is 20 W-updates and 20 X-updates;
- ρ-optimization is used to determine the optimal regularization parameter at every step of the optimization; at every step for the W update, ρ is optimized using 10-fold CV over 10 different values of ρ.

Note: after just one update of both W- and X-variables, the training accuracy immediately jumps up to around 90%.

Using lifted model as initialization

Model: lifted NN using 1 hidden layer with 300 nodes, 4 different initialization schemes



- Right at step one for the lifted initialization we are already at 90% accuracy, indicates that we are definitely learning something similar to regular NN
- We are already near optimal;
- There is a noticeable gap between training using our initialization and other initialization methods.

Conclusions and References

Some take-aways

- Lifted model provides competitive accuracy;
- Appears to be an **excellent initialization** scheme;
- Allows for **block-coordinate descent** methods to be applied;
- Much speed gains can result from **exploiting simple structure** of sub-problems, via modern methods such as sketching [Woodruff, 2014, Pilanci and Wainwright, 2016];
- Lifted framework can handle constraints on weight matrices, or various variants, quite easily;
- Connects NNs with other areas such as **matrix factorization** [Udell et al., 2016].

Arjovsky, M., Shah, A., and Bengio, Y. (2016).
Unitary evolution recurrent neural networks.
In *International Conference on Machine Learning*, pages 1120–1128.

Ben-Tal, A., El Ghaoui, L., and Nemirovski, A. (2009).
***Robust optimization*.**
Princeton University Press.

Bubeck, S. (2015).
Convex optimization: Algorithms and complexity.
Found. Trends Mach. Learn.

He, K., Zhang, X., Ren, S., and Sun, J. (2015).
Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.
CoRR, abs/1502.01852.

Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014).
A clockwork RNN.
In *International Conference on Machine Learning*, pages 1863–1871.

Maclin, R. and Shavlik, J. W. (1995).
Combining the predictions of multiple classifiers: Using competitive learning to initialize neural networks.
In *Proc. 14th Int. Joint Conference on Artificial Intelligence*, IJCAI'95.

Negiar, G., Askari, A., Fabian, P., and El Ghaoui, L. (2017).
Lifted neural networks for weight initialization.
In *10th NIPS Workshop on Optimization for Machine Learning*.

Pilanci, M. and Wainwright, M. J. (2016).
Iterative Hessian sketch: Fast and accurate solution approximation for constrained least-squares.
The Journal of Machine Learning Research, 17(1).

Seuret, M., Alberti, M., Ingold, R., and Liwicki, M. (2017).
PCA-initialized deep neural networks applied to document image analysis.
CoRR, abs/1702.00177.

Udell, M., Horn, C., Zadeh, R., Boyd, S., et al. (2016).
Generalized low rank models.
Foundations and Trends® in Machine Learning, 9(1):1–118.

Woodruff, D. P. (2014).
Sketching as a tool for numerical linear algebra.
CoRR, abs/1411.4357.