Acknowledgement

I would like to acknowledge and give thanks to my adviser Professor Laurent El Ghaoui for his invaluable guidance as my mentor. I would also like to thank PhD students Armin Askari and Geoffrey Negiar for all their support and help throughout the year.

Lifted Recurrent Neural Networks

Rajiv Sambharya [1]

Abstract

In this paper, we extend the lifted neural network framework (described in section 2) to apply to recurrent neural networks (RNNs). As with the general lifted neural network case, the activation functions are encoded via penalties in the training problem. The new framework allows for algorithms such as block-coordinate descent methods to be applied, in which each step is composed of a simple (no hidden layer) supervised learning problem that is parallelizable across data points and/or layers. The lifted methodology is particularly interesting in the case of recurrent neural networks because standard methods of optimization on recurrent neural networks perform poorly due to the vanishing and exploding gradient problems. Experiments on toy datasets indicate that our lifted model is more equipped to handle long-term dependencies and long sequences.

1. Introduction

Given current advances in computing power, dataset sizes and the availability of specialized hardware/ software packages, the popularity of neural networks continue to grow. The model has become standard in a large number of tasks, such as image recognition, image captioning and machine translation. RNNs are a natural extension to neural networks because they allow people to model sequential data such as language. Current state of the art is to train this model by variations of stochastic gradient descent (SGD), although these methods have several caveats. Most problems with SGD are discussed in (Taylor et al., 2016).

Optimization methods for neural networks has been an active research topic in the last decade. Specialized gradient-based algorithms such as Adam and ADAGRAD (Kingma & Ba, 2015; Duchi et al., 2011) are often used but were shown to generalize less than their non adaptive counterparts by (Wilson et al., 2017). Our work is related to a main current of research aimed at improving neural network optimization: non gradient-based approaches.

While training neural networks with SGD has problems, training RNNs introduces two additional widely known problems: the vanishing and the exploding gradient problems detailed in (Bengio et al.,1994). These issues arise from the use of gradient descent as an optimization technique. The proposed model circumvents this weakness as it is not a gradient-based method. We hope that our work serves as a useful alternate optimization method for RNNs.

(Taylor et al., 2016) and (Carreira-Perpinan & Wang, 2014) propose an approach similar to ours, adding variables in the training problem and using an l^2-norm penalization of equality constraints. They both break down the network training problem into easier sub-problems and use alternate minimization; however they do not exploit structure in the activation functions. For Convolutional Neural Networks (CNN), (Berrada et al., 2016) model the network training problem as a difference of convex functions optimization, where each subproblem is a Support Vector Machine (SVM).

The general lifted approach focuses on transforming the non-smooth optimization problem encountered when fitting neural network models into a smooth problem in an enlarged space; this ties to a well developed branch of optimization literature (see *e.g.* section 5.2 of (Bubeck, 2015) and references therein). Our approach can also be seen as a generalization of the parameterized rectified linear unit (PReLU) proposed by (He et al., 2015). Our work can be compared to the standard practice of initializing Gaussian Mixture Models using K-Means clustering; our model uses a simpler but similar algorithm for initialization.

Paper outline. In Section 2, we begin by describing the mathematical setting of neural networks and the lifted optimization problem to train the model. Then, we describe the mathematical formulation of RNNs in Section 3. Section 4.1 details the lifted structure of standard RNNs. Section 5 describes a block-coordinate descent method to solve the training problem. Section 6 describes numerical experiments that support a finding that lifted RNN is well-equipped to handle long-term dependencies and longer sequences.

2. Background: Feedforward and Lifted Neural Networks

Feedforward neural networks We begin by establishing notation. We are given an input data matrix $X = [x_1, \dots, x_m] \in \mathbb{R}^{n\times m}$ and response matrix $Y \in \mathbb{R}^{p\times m}$ and consider a supervised problem involving a neural network having $L \geq 1$ hidden layers. At test time, the network processes an input vector $x \in \mathbb{R}^n$ to produce a predicted value $\hat{y}(x) \in \mathbb{R}^p$ according to the prediction rule $\hat{y}(x) = x_{L+1}$ where x_{L+1} is defined via the recursion

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

with initial value $x_0 = x \in \mathbb{R}^n$ and $x_l \in \mathbb{R}^{p_l}$, $l = 0, \ldots, L$. Here, ϕ_l, $l = 1, \ldots, L$ are given activation functions, acting on a vector; the matrices $W_l \in \mathbb{R}^{p_{l+1} \times p_l}$ and vectors $b_l \in \mathbb{R}^{p_{l+1}}$, $l = 0, \ldots, L$ are parameters of the network. In our setup, the sizes $(p_l)_{l=0}^{L+1}$ are given with $p_0 = n$ (the dimension of the input) and $p_{L+1} = p$ (the dimension of the output).

We can express the predicted outputs for a given set of m data points contained in the $n \times m$ matrix X as the $p \times m$ matrix $\hat{Y}(X) = X_{L+1}$, as defined by the matrix recursion

$$X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}^T), \;\; l = 0, \ldots, L, \qquad (2)$$

with initial value $X_0 = X$ and $X_l \in \mathbb{R}^{p_l \times m}$, $l = 0, \ldots, L$. Here, $\mathbf{1}$ stands for the vector of ones in $\mathbb{R}^m$, and we use the convention that the activation functions act column-wise on a matrix input.

In a standard neural network, the matrix parameters of the network are fitted via an optimization problem, typically of the form

$$\begin{array}{rl} \min\limits_{(W_l,b_l)_0^L, (X_l)_1^L} & \mathcal{L}(Y, X_{L+1}) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ \text{s.t.} & X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}_m^T), \;\; l = 0, \ldots, L \\ & X_0 = X \end{array} \qquad (3)$$

where $\mathcal{L}$ is a loss function, $\rho \in \mathbb{R}_+^{L+1}$ is a hyper-parameter vector, and π_l's are penalty functions which can be used to encode convex constraints, network structure, etc. We refer to the collections $(W_l, b_l)_{l=0}^L$ and $(X_l)_{l=1}^L$ as the (W, b)- and X-variables, respectively.

To solve the training problem (3), the X-variables are usually eliminated via the recursion (2), and the resulting objective function of the (W, b)-variables is minimized without constraints, via stochastic gradients. While this appears to be a natural approach, it does make the objective function of the problem very complicated and difficult to minimize.

Lifted models. The lifted neural network model describes a family of models where the X-variables are kept, and the recursion constraints (1) are approximated instead, via penalties. We refer to these models as "lifted" because we lift the search space of (W, b)-variables to a higher-dimensional space of (W, b, X)-variables. The training problem is cast in the form of a matrix factorization problem with constraints on the variables encoding network structure and activation functions.

Lifted models have many more variables but a much more explicit structure than the original, allowing for training algorithms that can use efficient standard machine learning libraries in key steps. The block-coordinate descent algorithm described here involves steps that are parallelizable across either data points and/or layers; each step is a simple structured convex problem.

Example run-through To describe the basic idea, we consider a specific example, in which all the activation functions are the ReLUs, except for the last layer. There ϕ_L is the identity for regression tasks or a softmax for classification tasks. In addition, we assume in this section that the penalty functions are of the form $\pi_l(W) = \|W\|_F^2$, $l = 0, \ldots, L$.

We observe that the ReLU map, acting componentwise on a vector input u, can be represented as the "argmin" of an optimization problem involving a jointly convex function:

$$\phi(u) = \max(0, u) = \arg\min_{v \geq 0} \|v - u\|_2. \qquad (4)$$

As seen later, many activation functions can be represented as the "$\arg\min$" of an optimization problem, involving a jointly convex or bi-convex function.

Extending the above to a matrix case yields that the condition $X_{l+1} = \phi(W_l X_l + b_l \mathbf{1}^T)$ for given l can be expressed via an "$\arg\min$":

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

This representation suggests a heuristic to solve (3), replacing the training problem by

$$\begin{array}{rl} \min\limits_{(W_l,b_l),(X_l)} & \mathcal{L}(Y, W_L X_L + b_L \mathbf{1}^T) + \sum_{l=0}^{L} \rho_l \|W_l\|_F^2 \\ & + \sum_{l=0}^{L-1} \left(\lambda_{l+1} \|X_{l+1} - W_l X_l - b_l \mathbf{1}^T\|_F^2 \right) \\ \text{s.t.} & X_l \geq 0, \;\; l = 1, \ldots, L-1, \;\; X_0 = X. \end{array} \qquad (5)$$

where $\lambda_{l+1} > 0$ are hyperparameters, ρ_l are regularization parameters as in (3) and $\mathcal{L}$ is a loss describing the learning task. In the above model, the activation function is not used in a pre-defined manner; rather, it is *adapted* to data, via the non-negativity constraints on the "state" matrices $(X_l)_{l=1}^{L+1}$. We refer to the above as a "lifted neural network" problem.

The above optimization problem is, of course, challenging, mainly due to the number of variables. However, for that price we gain a lot of insight on the training problem. In particular, the new model has the following useful characteristics:

- For fixed (W, b)-variables, the problem is convex in the X-variables X_l, $l = 1, \ldots, L$; more precisely it is a (matrix) non-negative least-squares problem. The problem is fully *parallelizable across the data points*.

- Likewise, for fixed X-variables, the problem is convex in the (W, b)-variables and *parallelizable across layers and data points*. In fact, the (W, b)-step is a set of parallel (matrix) ridge regression problems.

These characteristics allow for efficient block-coordinate descent methods to be applied to our learning problem. Each step reduces to a basic supervised learning problem, such as ridge regression or non-negative least-squares. The lifted neural network block-coordinate descent algorithm alternates between modifying the W-variables and the X-variables.

3. Background: Recurrent Neural Networks

RNNs are designed to handle data in which each sample is a sequence of data points. Time series and Natural Language Processing (NLP) offer many applications due to their sequential nature.

RNNs have several different structures depending on the application. There is a many-to-many structure in which we output a value at each point in time. There is a many-to-one structure in which we output a value only at the final time point. There is a translation-type many-to-many task in which we output many values only after all of the input data points have been processed. In this paper we focus on the first two structures.

We establish notation for RNNs here.

1. $\boldsymbol{i}$ is the dimensionality of each input sample at each time point.
2. $\boldsymbol{h}$ is the dimensionality of each hidden state at each time point.
3. $\boldsymbol{o}$ is the dimensionality of each output state at each time point.
4. $\boldsymbol{T}$ is the length of the longest sequence.
5. $\boldsymbol{m}$ is the number of training samples.

We are given an input data tensor $X = [x_1, \dots, x_m] \in \mathbb{R}^{m \times i \times T}$ and response matrix $Y \in \mathbb{R}^{m \times o \times T}$ and consider a supervised problem involving an RNN. At test time, the network processes an input matrix $x \in \mathbb{R}^{i \times T}$ to produce a predicted value $\hat{y}(x) \in \mathbb{R}^{o \times T}$ according to a basic feedforward prediction rule.

While RNNs can have any amount of depth, we focus on the standard RNN which has one layer of hidden units. This RNN has 3 layers of states: input states, hidden states, and output states. Each layer has a length T. The same weight matrix, U_0, connects each input unit with its corresponding hidden unit. The same weight matrix, U_1, connects each hidden unit with its corresponding output unit. In addition, a weight matrix, W, connects each hidden state to the next hidden state. In addition, RNNs in general can handle variable length sequences (by simply zero-padding data sequences with shorter length).

We can represent each unit as a matrix of all the data points. Let $H_{i,j} : i \in 0, 1, 2, j \in 0, \dots T-1$ represent each hidden unit. $H_{0,j} \in \mathbb{R}^{m \times i}$, $H_{2,j} \in \mathbb{R}^{m \times h}$, $H_{0,j} \in \mathbb{R}^{m \times o}$. Let $Y_j :\in \mathbb{R}^{m \times o}$ denote the target value matrices for a certain time point j. Let $X_j :\in \mathbb{R}^{m \times i}$ denote the input value matrices for a certain time point j.

The input units are trivially determined as the input data points.

Each hidden unit is calculated as ...

$$H_{1,j} = \phi_0(H_{1,j-1}W + H_{0,j}U_0 + \vec{1}b_0^T)$$

Each output unit is calculated as ...

$$H_{2,j} = \phi_1(H_{1,j}U_1 + \vec{1}b_1^T)$$

For a regression problem with an L2-loss, the RNN problem can be viewed as the following. Note that the F-norm is the Frobenius norm.

$$\begin{aligned} \min_{U_0, b_0, U_1, b_1, W} \sum_{j=0}^{K-1} \|Y_j - H_{2,j}\|_F^2 + p(U_0, U_1, W) \\ \text{s.t.} H_{2,j} = \phi_1(H_{1,j}U_1 + \vec{1}b_1^T) \\ H_{1,j} = \phi_0(H_{1,j-1}W + X_jU_0) \end{aligned}$$

RNNs can also handle classification. In this case the problem becomes. s is the cross-entropy loss function typically used for classification problems. The function includes a softmax over the $H_{2,j}$ variable to normalize into a probability distribution.

$$\begin{aligned} \min_{U_0, U_1, W} \sum_{j=0}^{K-1} -Y_j^T \log H_{2,j} + p(U_0, U_1, W) \\ \text{s.t.} H_{2,j} = s(H_{1,j}U_1 + \vec{1}b_1^T) \\ H_{1,j} = \phi(H_{1,j-1}W + X_jU_0) \end{aligned}$$

Usually we consider L2 regularization as the penalty on the U and W weight matrices.

$$p(U_0, U_1, W) = \rho_0 \| \begin{pmatrix} W \\ U_0 \end{pmatrix} \|_F^2 + \rho_1 \|U_1\|_F^2$$

The mathematical formulation proposed above relates to the many-to-many problem. It is easily adapted to the many-to-one problem by only including the loss for the last element in the sequence.

4. Lifted RNN Framework

4.1. Lifted RNNs

The RNN structure changes several things about the lifted problem. Each X-variable becomes T different state-variables. In our case we only consider the standard RNN, so the number of hidden layers is one. However, each layer now was T matrix variables instead of one. Furthermore, the weight-variables from the general lifted model are the same except we now have to account for the recurrent weight matrix (W). This recurrent weight matrix requires us to update each hidden state matrix separately.

The lifted model consists in replacing the constraints with penalties in the training problem. Specifically, the lifted network training problem takes the following form for the regression task with relu activations. The zeroth and second hidden layers are fixed to the inputs and the outputs respectively. $H_{0,j} = X_j$ and $H_{2,j} = Y_j$

$$\begin{aligned}
\min_{W,U_0,U_1,(H_{lj})_{l=0,j=0}^{2,K-1}} \; & \lambda_0[\|H_{1,0} - H_{0,0}U_0 - \vec{1}b_0^T\|_F^2 \\
& + \sum_{j=1}^{K-1} \|H_{1,j} - H_{0,j}U_0 - H_{1,j-1}W - \vec{1}b_0^T\|_F^2] \\
& + \lambda_1[\sum_{j=0}^{K-1} \|H_{2,j} - H_{1,j}U_1 - \vec{1}b_1^T\|_F^2] + \rho_0\| \begin{pmatrix} W \\ U_0 \end{pmatrix} \|_F^2 \\
& + \rho_1\|U_1\|_F^2 \\
\text{s.t.} \; & H_{0,j} = X_j : j = 0, \dots, T-1 \\
& H_{2,j} = Y_j : j = 0, \dots, T-1 \\
& H_{1,j} \geq 0 : j = 0, \dots, T-1
\end{aligned}$$

with λ_0, λ_1 given positive hyper-parameters. These hyper-parameters indicate how harshly we should penalize the loss in computing the hidden layer and the loss in estimating the true labels. ρ_0, ρ_1 are given hyperparameters that are regularization terms for the weight matrices. As with the model introduced in section 2, the lifted RNN model enjoys the same *parallel and convex* structure outlined earlier. In particular, it is convex in state-variables for fixed weight-variables.

As a specific example, consider a multi-class classification problem where the hidden layer involves ReLUs. The last layer aims at producing a probability distribution to be compared against training labels via a cross entropy loss function. The training problem writes

$$\begin{aligned}
\min_{W,U_0,U_1,(H_{l,j})_{l=0,j=0}^{2,K-1}} \; & \lambda_0[\|H_{1,0} - H_{0,0}U_0 - \vec{1}b_0^T\|_F^2 \\
& + \sum_{j=1}^{K-1} \|H_{1,j} - H_{0,j}U_0 - H_{1,j-1}W - \vec{1}b_0^T\|_F^2] \\
& + \lambda_1 \sum_{j=0}^{K-1} - \mathbf{Tr}\, H_{2,j}^T \log s(H_{1,j}U_1 + \vec{1}b_1^T) \\
& + \rho_0\| \begin{pmatrix} W \\ U_0 \end{pmatrix} \|_F^2 + \rho_1\|U_1\|_F^2 \\
\text{s.t.} \; & H_{0,j} = X_j : j = 0, \dots, T-1 \\
& H_{2,j} = Y_j : j = 0, \dots, T-1 \\
& H_{1,j} \geq 0 : j = 0, \dots, T-1
\end{aligned}$$

Here, $s(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}^n$ is the softmax function.

4.2. Lifted prediction rule

For notation purposes in this paragraph, we specify the lower cases y_j, $h_{1,j}$, and x_j to specify 1 sample of Y_j, $H_{1,j}$, and X_j respectively. In our model, the prediction rule will be different from that of a standard neural network, but it is based on the same principle. In a standard network, the prediction rule can be obtained by solving the problem

$$\hat{y}_j(x) = \min_y \mathcal{L}(y, h_{2,j}) \; : \; (2), \;\; x_0 = x,$$

where the weights are now *fixed*, and $y \in \mathbb{R}^p$ is a *variable*. Of course, provided the loss is zero whenever its two arguments coincide, the above trivially reduces to the standard prediction rule: $\hat{y}_j(x) = h_{2,j}$, where $h_{2,j}$ is obtained via the basic feedforward network).

In a lifted framework, we use the same principle: solve the training problem, using the test point as input, fixing the weights, and letting the predicted output values be variables. In other words, the prediction rule for a given test point x in lifted networks is based on solving the problem

$$\begin{aligned}
\hat{y} = \arg\min_{y,(h_1)} \; & \lambda_1\mathcal{L}_1(y, h_1U_1 + b_1) \\
& + \lambda_0 \sum_{j=0}^{T-1} \mathcal{L}_2(h_{1,j-1}W + h_{0,j}U_0 + b_0, h_{1,j}) \\
\text{s.t. } & h_{0,j} = x_j.
\end{aligned}$$

The above prediction rule is a simple convex problem in the variables y_j and $h_1, j, j = 0, \dots, T-1$. In our experiments, we have found that applying the standard feedforward rule of traditional networks is often enough.

5. Block-Coordinate Descent Algorithm

In this section, we outline a block-coordinate descent approach to solve the training problem. There are two different types of variables: weights and states. The weights are U_0, b_0, U_1, b_1, W. The states are $(H_{l,j})_{l=0,j=0}^{2,K-1}$. Recall in the standard RNN formulation, the weights are the only non-trivial variables; the state variables are totally constrained. We have lifted the variable space to a higher degree. We alternate between optimizing over the weights and optimizing over the states. With the states fixed, optimizing over the weights is a convex problem. With the weights fixed, optimizing over the states is also a convex problem.

5.1. Updating Weight-variables

For fixed state-variables, the problem of updating the weight-variables, *i.e.* the weighting matrices U_0, U_1, W, is not exactly parallelizable across both data points and layers. In order for the weight update to be parallelizable across the data points and layers, we need to update the W and U_0 variables simultaneously. They U_1 update is done separately. After we group the first layer weights together, the problem becomes parallelizable across the layers.

The above is a convex problem, which can be solved via standard machine learning libraries. Since the divergences are sums across columns (data points), the above problem is indeed parallelizable across data points.

For example, when the problem is regression, the activation function at the hidden layer is a ReLU, and the penalty is a squared Frobenius norm, the above problem reads

$$(W, U_0, b_0) = \arg\min_{W,U_0,b_0} \lambda_0[||H_{1,0} - \begin{pmatrix}0 & X_0 & \vec{1}\end{pmatrix} \begin{pmatrix} W \\ U_0 \\ b_0^T \end{pmatrix} ||_F^2$$

$$+ \sum_{j=1}^{k-1} ||H_{1,j} - \begin{pmatrix}H_{1,j-1} & X_j & \vec{1}\end{pmatrix} \begin{pmatrix} W \\ U_0 \\ b_0^T \end{pmatrix} ||_F^2] + \rho_0 || \begin{pmatrix} W \\ U_0 \end{pmatrix} ||_F^2$$

which is a standard (matrix) ridge regression problem. Modern sketching techniques for high-dimensional least-squares can be employed, see for example (Woodruff et al., 2014; Pilanci & Wainwright, 2016).

Once the first layer weights (W, U_0, b_0) have been updated, we update the second layer weights (U_1, b_1)

$$(U_1, b_1) = \mathit{arg} \min_{U_1,b_1} \lambda_1 \sum_{j=0}^{k-1} ||H_{2,j} - \begin{pmatrix}H_{1,j} & \vec{1}\end{pmatrix} \begin{pmatrix} U_1 \\ b_1^T \end{pmatrix} ||_F^2 +$$
$$\rho_1 ||U_1||_F^2$$

This update is also a simple ridge regression problem. When the problem is a classification one with a cross-entropy loss, this problem is no longer a ridge problem, but it is still convex.

5.2. Updating state-variables

In this step we minimize over the matrices $(H_1, j)_{j=0}^{L-1}$. Recall that the $H_{0,j}$ matrices are fixed to the inputs and that the $H_{2,j}$ matrices are fixed to the outputs. The sub-problem now has the weight-variables, (W, U_0, U_1, b_0, b_1), fixed. The update must be done cyclically over each time element, in a block-coordinate fashion.

Let us detail this approach in the case when the hidden layer units are all activated by ReLUs. The sub-problem above becomes

$$H_{1,j} = \mathit{arg} \min_{H \geq 0} \lambda_0 ||H - X_j U_0 - \vec{1} b_0^T - H_{1,j-1} W||_F^2 +$$
$$\lambda_0 ||H_{1,j+1} - X_{j+1} U_0 - \vec{1} b_0^T - HW||_F^2 + ||Y_j - HU_0||_F^2$$

The above is a (matrix) non-negative least-squares, for which many modern methods are available, see (Kim et al., 2007; 2014) and references therein. As before, the problem above is fully parallelizable across data points (columns), where each data point gives rise to a standard (vector) non-linear least-squares. Note that the cost of updating all columns can be reduced by taking into account that all column's updates share the same coefficient matrix.

6. Numerical Experiments

In this section we explore several different toy datasets. We find that the lifted framework achieves higher performance for tasks that are more numerically-based rather than tasks that are more based on memorization or interpretation. Each experiment was conducted 10 times: the average results are shown.

For all the experiments, some hyperparameters were kept the same.

1. Learning rate schedule: 1e-5 with a decay rate of 0.9 every 1000 training steps.
2. batch size = 100
3. train size = 200
4. test size = 1000
5. SGD: ρ_0, ρ_1 = 1e-3
6. Lifted: ρ_0, ρ_1 = 1e-1
7. Lifted: λ_0, λ_1 = 10, 1

6.1. Lag echo sequence

Experiment description This synthetic dataset is a simple many-to-many memorization task. The input is a sequence of 1-hot vectors. The output is the same as the input sequence but at a specified time lag. The parameters necessary for the dataset are the number of classes, the lag, and the sequence length.

The following is an example of an input-output sequence pair with lag 1, length 6, and 4 different classes. Note that the ith column of x matches the (i+1)th column of y. The first several sequences (before the lag kicks in) of the output are randomly assigned. Therefore, it is unreasonable to expect any classifier to attain 100% accuracy on this dataset.

$$x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} y = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Experiment details

1. hidden unit size = 20
2. num classes = 10
3. length = 10

Lag	Lifted Model	SGD	maxScorePossible
1	**0.91**	**0.91**	0.91
2	**0.82**	**0.82**	0.82
3	0.25	**0.73**	0.73
4	0.15	**0.64**	0.64

6.2. Sum of random values thresholded

Experiment description: This is another classification task. The input is a sequence of uniformly random values between -1 and 1. At each time point, the output is a 1-hot vector. The output is on (index 0 is 1) if the sum of the input values up to that point are positive. If the sum of the values are negative, then the output is off (index 1 is 1). The running sum, RS, is written to help illustrate the mechanics of the dataset.

$$x = \begin{pmatrix} -0.26 & 0.55 & -0.78 & 0.05 & 0.89 & 0.12 \end{pmatrix}$$
$$y = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$
$$RS = \begin{pmatrix} -0.26 & 0.29 & -0.49 & -0.44 & 0.45 & 0.57 \end{pmatrix}$$

Experiment details:

1. hidden unit size = 10

Length	Lifted Model	SGD
10	**0.92**	0.79
20	**0.92**	0.75
50	**0.88**	0.72
75	**0.875**	0.65
100	**0.85**	0.60
200	**0.80**	0.50
300	**0.78**	0.50
400	**0.72**	0.50

6.3. Timer

Experiment description Here we describe a timer dataset for binary classification. The input data is three-dimensional at each timestep. The first input is a random integer between 1 and p inclusive. p indicates the longest possible timer value. This value will indicate how long our timer is on. The second/third inputs indicates whether the on switch is on for the timer. We imagine that there is a running timer which is a single integer. At each time step, the running timer decreases by 1 (or if it's zero it stays at zero). Then we consider the next input vector. If the second input is off then we ignore that timestep input. If the second input is on, then we update our running timer to be the max of the previous running timer and the timer input. We specify the fraction of the time that the inputs are on as a hyperparameter. The fraction and the actual time value are independent of each other.

The output is a two-dimensional one-hot vector at each time step. Our output at each time point will be ”on” if the running timer has a positive value and ”off” if the running timer has a negative value. The running timer labelled as RT is given here to illustrate the mechanics of the example.

$$x = \begin{pmatrix} 3 & 2 & 5 & 4 & 2 & 4 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$
$$y = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$
$$RT = \begin{pmatrix} 0 & 2 & 1 & 0 & 2 & 1 \end{pmatrix}$$

Experiment details: varying the dependency

1. hidden unit size = 10
2. Length = 60
3. the fraction of the time that the inputs are on is chosen such that the dataset is 50/50 on/off on the output

Max-dependency	Lifted Model	SGD
5	**1.0**	**1.0**
10	0.88	**0.97**
20	0.84	**0.89**
30	**0.80**	0.60
40	**0.65**	0.50

Experiment details: varying the length

1. hidden unit size = 10
2. Dependency = 20
3. the fraction of the time that the inputs are on is chosen such that the dataset is 50/50 on/off on the output: in this case, the fraction is set to 0.08

Length	Lifted Model	SGD
60	0.99	**1.0**
100	**0.96**	0.89
200	**0.84**	0.50
300	**0.80**	0.50
400	**0.65**	0.50

We see some mixed results across the toy datasets. Our lifted RNN model does not perform well on the lag sequence task. High lags are captured perfectly in the SGD model, but our model fails after lag 2. This indicates that our model may not perform well on memorization-based tasks.

The other two tasks show favorable result for our lifted model. The random values summation task shows that our model does well across all time lengths. In addition, our model does better in comparison as the length of the sequence increases.

The timer task had two different experiments: one that varied the length of the sequence while keeping the time dependencies at the same length and one that varied time the length of the dependencies within the data while keeping the sequence length constant. The timer task indicates that for shorter sequences, our model does not perform as well as the SGD model. However, as the length of the sequence increase, the SGD model breaks down, while ours only sees a small decrease in performance.

Additionally, This may indicate that our model performs well as the sequence length increases as well as when the time dependencies within the data increase.

As a supplementary note, generally, the lifted model was much quicker than SGD with the longer sequences.

7. Conclusion

In this work we adapt the lifted neural network framework to recurrent neural networks. We propose a structure in which the lifted version just in itself yields successful results as it circumvents the common issues of gradient-based optimization techniques for RNNs: the exploding and vanishing gradient problems.

In this work we have proposed a novel model for supervised learning specifically for data that comes as sequences. The key idea behind our method is replacing non-smooth activation functions by smooth penalties in the training problem; we have shown how to do this for general monotonic activation functions. This modifies the recurrent neural networks optimization problem to a similar problem which we called a lifted recurrent neural network.

8. Future Work

While lifted recurrent neural networks give good results for many of the toy tasks described, they perform poorly on more complicated tasks such as Natural Language Processing tasks (preliminary results not shown). We can further extend our lifted framework to Long-short-term-memory cells (LSTMs); we hope this will allow our model to interpret less arithmetic data such as language, pictures, etc. We can also easily extend our model to deeper networks and bi-directional structures. We plan on extending the applications to the many-to-many translation tasks.

References

Berrada, Leonard, Zisserman, Andrew, and Kumar, M. Pawan. Trusting svm for piecewise linear cnns. *CoRR*, abs/1611.02185, 2016.

Bubeck, Sébastien. Convex optimization: Algorithms and complexity. *Found. Trends Mach. Learn.*, 2015. doi: 10.1561/2200000050. URL http://dx.doi.org/10.1561/2200000050.

Carreira-Perpinan, Miguel and Wang, Weiran. Distributed optimization of deeply nested systems. In Kaski, Samuel and Corander, Jukka (eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pp. 10–19, Reykjavik, Iceland, 22–25 Apr 2014. PMLR. URL http://proceedings.mlr.press/v33/carreira-perpinan14.html.

Duchi, John C., Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2011. URL http://dl.acm.org/citation.cfm?id=2021068.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

Kim, Dongmin, Sra, Suvrit, and Dhillon, Inderjit S. Fast Newton-type methods for the least squares nonnegative matrix approximation problem. In *Proceedings of the 2007 SIAM international conference on data mining*, pp. 343–354. SIAM, 2007.

Kim, Jingu, He, Yunlong, and Park, Haesun. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58(2):285–319, 2014.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations (ICLR)*, 2015.

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

Taylor, Gavin, Burmeister, Ryan, Xu, Zheng, Singh, Bharat, Patel, Ankit, and Goldstein, Tom. Training neural networks without gradients: A scalable admm approach. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 2722–2731. JMLR.org, 2016. URL http://dl.acm.org/citation.cfm?id=3045390.3045677.

Wilson, Ashia C., Roelofs, Rebecca, Stern, Mitchell, Srebro, Nati, and Recht, Benjamin. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 4151–4161, 2017.

Woodruff, David P et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.