Notes on Neural Networks based on primal-dual splitting method

Rui, Zheng

November 23, 2018

Contents

1 Possible paper production

Ordered by priority from high to low

- Training NN with ADMM (Prime-Dual);
- Train CapsuleNet with ADMM
- Spectro-temporal Layer with ADMM
- CSC with ADMM

2 Introduction

Deep neural networks (DNNs) have have been used to solve a wide variety of applications such as image classification, object recognition, natural language processing, etc. Although, gradient based search techniques such as backpropagation are currently the most widely used optimization techniques for training neural networks, it has been shown that these gradient techniques are severely limited in their ability to computational efficiency.

3 Problem formulation

Consider a dataset $\mathscr{D}(\mathbf{X},\mathbf{y})$ of n observations where $\mathbf{X}$ denotes an input matrix (covariates) of dimension $D_x \times n$ and $\mathbf{y}$ denotes the scalar output (or target) variable. Given this training dataset, we wish to make predictions for new inputs $\mathbf{x}^*$. A typical objective function to learn a DNNs with L layers has the form

$$J(\mathbf{w};\mathscr{D}) = L(\mathbf{w};\mathscr{D}) + \lambda\phi(\mathbf{w}) \tag{1}$$

. Here, the loss functions $L(\mathbf{w};\mathscr{D})$ are usually MSE or crossentropy: For 1) mean square error (MSE), the above equation generalizes to:

$$\begin{aligned} J(\mathbf{w};\mathbf{X},\mathbf{y}) &= \frac{1}{2N}\|\mathbf{y} - f(\mathbf{w},\mathbf{X})\|^2 + \lambda\phi(\mathbf{w}) \\ &= \frac{1}{2N}\|\mathbf{y} - f(\mathbf{w},\mathbf{X})\|^2 + \lambda\|\mathbf{w}\|_1 \end{aligned} \tag{2}$$

where f is the prediction function, $\mathbf{w}$ is the parameter we optimize, and λ is the constant.

For 2) crossentropy:

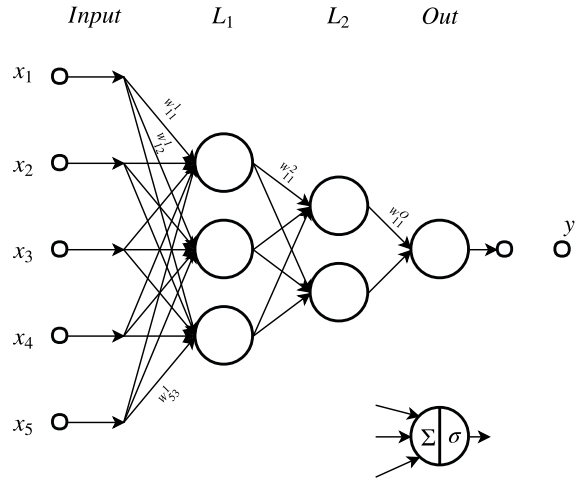
$$J(\mathbf{w};\mathbf{X},\mathbf{y}) = \sum_{i=1}^{N}(y_i \log f(\mathbf{w},\mathbf{x}) + (1-y_i)\log(1 - f(\mathbf{w},\mathbf{x}))) + \lambda\phi(\mathbf{w}) \tag{3}$$

where $\lambda\phi(\mathbf{w})$ can be any regularizers (L1, L2 etc.), and $f(\cdot)$ is network function.

The aim here is to find the optimize values weights $\mathbf{w}$ of neural network to minimize the objective function.

3.1 One Example

For a simplest NN example. Here we will use the notation: $\mathbf{x} \in \mathbb{R}^5 : \mathbf{x} = \{x_1, x_2, ..., x_5\}$ and $y : y \in \mathbb{R}$ as the pair of data for training; w_{ij}^l denotes the weight that connects i-th neuron on $(l-1)$-th layer and j-th neuron on l-th layer; $\sigma_l(\cdot)$ is the activation function for l-th layer; o_i^l is the output from the i-th neuron of l-th layer; Out is the output of the neural network (last neuron):



The loss is:

$$J(\mathbf{W};\mathbf{x},y) = \frac{1}{2}(Out_{\mathbf{w},\mathbf{x}} - y)^2 + \text{regularizer on } \mathbf{w} \tag{4}$$

Let us show some forward example of it. Take output Out:

$$Out = \sigma_3(o_1^2 w_{11}^3 + o_2^2 w_{21}^3 + b^3) \tag{5}$$
$$= \sigma_3(\mathbf{o}^2 \mathbf{w}^3 + b_3) \tag{6}$$
$$= \sigma_3(\Sigma_1^3) \tag{7}$$

where $\mathbf{o}^2 = [o_1^2, o_2^2]$, $\mathbf{w}^3 = [w_{11}^3, w_{21}^3]^T$ and $\Sigma_1^3 = \mathbf{o}^2 \mathbf{w}^3 + b_3$. Let us continue for o_1^2 and o_2^2:

$$o_1^2 = \sigma_2(o_1^1 w_{11}^2 + o_2^1 w_{21}^2 + o_3^1 w_{31}^2 + b_1^2) \tag{8}$$
$$= \sigma_2(\Sigma_1^2) \tag{9}$$
$$o_2^2 = \sigma_2(o_1^1 w_{12}^2 + o_2^1 w_{22}^2 + o_3^1 w_{32}^2 + b_2^2) \tag{10}$$
$$= \sigma_2(\Sigma_2^2) \tag{11}$$
$$\mathbf{o}^2 = \sigma_2(\mathbf{o}^1 \mathbf{w}^2 + \mathbf{b}^2) \tag{12}$$

where $\mathbf{o}^1 = [o_1^1, o_2^1, o_3^1]$ and $\mathbf{w}^2 = \begin{bmatrix} w_{11}^2 & w_{21}^2 & w_{31}^2 \\ w_{12}^2 & w_{22}^2 & w_{32}^2 \end{bmatrix}^T$. and $\mathbf{o}^1 = \sigma_1(\mathbf{x}\mathbf{w}^1 + \mathbf{b}^1)$.

In traditional approach, we need to derive the gradient of the loss w.r.t each weight i.e.:

$$\frac{\partial J(\mathbf{W}; \mathbf{x}, y)}{\partial w_{ij}^l} \tag{13}$$

Those gradient can be efficiently obtained by backpropogation (BP). For simplicity, BP not discussed here.

If we use SGD, the update of weight is just: $w_{ij}^l(k+1) = w_{ij}^l(k) + \eta \frac{\partial J(\mathbf{W}; \mathbf{x}, y)}{\partial w_{ij}^l}$.

Based on SGD, we consider using ADMM to solve the objective function.

4 Spectro-temporal layer

Considering an LTI system:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}\mathbf{x}_{k-1} + \mathbf{q}, \mathbf{q}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \\ y_k &= \mathbf{H}\mathbf{x}_k + r_k, r_k \sim \mathcal{N}(0, R) \end{aligned} \tag{14}$$

where $\mathbf{A}$, $\mathbf{Q}$ and $\mathbf{H}$ are defined as:

$$\mathbf{A} = \begin{bmatrix} 1 & & & \\ & \mathbf{A}_k^1 & & \\ & & \ddots & \\ & & & \mathbf{A}_k^M \end{bmatrix} \tag{15}$$

$$\mathbf{Q} = \begin{bmatrix} qb\Delta t & & & \\ & \mathbf{Q}_k^1 & & \\ & & \ddots & \\ & & & \mathbf{Q}_k^M \end{bmatrix} \tag{16}$$

$$\mathbf{H} = [1, \mathbf{H}^1, \ldots, \mathbf{H}^M] = [11010, \ldots, 10] \tag{17}$$

where $\mathbf{A}_k^j$ and $\mathbf{Q}_k^j$ are given in closed form:

$$\mathbf{A}_k^j = \exp\big(\mathbf{F}_j \Delta t\big) \tag{18}$$

$$= \begin{bmatrix} \cos\big(2\pi f_j \Delta t\big) & -\sin\big(2\pi f_j \Delta t\big) \\ \sin\big(2\pi f_j \Delta t\big) & \cos\big(2\pi f_j \Delta t\big) \end{bmatrix} \exp\big(-\Delta t \lambda_j\big) \tag{19}$$

$$\mathbf{Q}_k^j = \int_0^{\Delta t} \exp\big(\mathbf{F}_j(\Delta t - s)\big) \mathbf{L} \mathbf{Q}_k^j \mathbf{L}^\top \exp\big(\mathbf{F}_j(\Delta t - s)\big)^\top ds \tag{20}$$

$$= \frac{Q_k^j}{2\lambda^j}(1 - \exp\big(-2\Delta t \lambda_j\big))\mathbf{I} \tag{21}$$

We can use Kalman filter to easily estimate $\mathbf{x}_k$.

However, the point of spectro-temporal layer is that we set $\mathbf{A}$ and $\mathbf{Q}$ are paramiterized by **trainable** weight $\mathbf{w}$.

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}(\mathbf{w})\mathbf{x}_{k-1} + \mathbf{q}, \mathbf{q}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}(\mathbf{w})) \\ y_k &= \mathbf{H}\mathbf{x}_k + r_k, r_k \sim \mathcal{N}(0, R) \end{aligned} \tag{22}$$

We put a signal y as input of this layer. After Kalman estimation, it outputs a spectrogram (image) $\mathbf{X}$, we then feed that $\mathbf{X}$ into NN to classify. Because $\mathbf{A}$ and $\mathbf{Q}$ are trainable for NN, thus just optimize that NN loss function, but the computational cost is very expensive.

5 ADMM solver for MSE

We consider the original optimization problem (24). Firstly, we introduce an auxiliary variable $\mathbf{V}$, and have the following constrained problem

$$\begin{aligned} &\min_W \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - f(\mathbf{W}, \mathbf{x})\|^2 + \lambda \|\mathbf{V}\|_1 \\ &\text{s.t. } \mathbf{V} = \mathbf{W}. \end{aligned} \tag{23}$$

Then, Let $\mathscr{L}_\rho(x, v; \eta)$ be the augmented Lagrangian function of Eq. (24) which is defined as follows

$$\mathscr{L}_\rho(\mathbf{W}, \mathbf{V}; \boldsymbol{\eta}) \triangleq \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - f(\mathbf{W}, \mathbf{x})\|^2 + \lambda \|\mathbf{V}\|_1 + \langle \mathbf{V} - \mathbf{W}, \boldsymbol{\eta} \rangle + \frac{\rho}{2} \|\mathbf{V} - \mathbf{W}\|^2 \tag{24}$$

The solution of (25) can be obtained by the following steps [**?**]:

$$\begin{aligned} \mathbf{W}^{k+1} &:= \arg\min_{\mathbf{W}} \mathscr{L}_\rho(\mathbf{W}, \mathbf{V}^k; \boldsymbol{\eta}^k) \\ \mathbf{V}^{k+1} &:= \arg\min_{\mathbf{W}} \mathscr{L}_\rho(\mathbf{W}^{k+1}, \mathbf{V}; \boldsymbol{\eta}^k) \\ \boldsymbol{\eta}^{k+1} &:= \boldsymbol{\eta}^k + \rho(\mathbf{V}^{k+1} - \mathbf{W}^{k+1}) \end{aligned} \tag{25}$$

1. Input: $\mathbf{V}^0$, $\boldsymbol{\eta}^0$, ρ,
2. Loop: For $k = 1, 2, \cdots$ until termination criteria are met.

- Calculate x^k by solving:

$$\begin{aligned}\mathbf{W}^{k+1} &= \arg\min_{\mathbf{W}} \mathscr{L}_\rho(\mathbf{W}, \mathbf{V}^k; \boldsymbol{\eta}^k) \\ &= \arg\min_{\mathbf{W}} \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}_n - f(\mathbf{W}, \mathbf{x})\|^2 + + \frac{\rho}{2}\|\mathbf{V} - \mathbf{W} + \boldsymbol{\eta}\|^2\end{aligned} \tag{26}$$

  Eq. 27 has the close-form expression. We could use RTS smoother to solve the problem (27)

- Calculate $\mathbf{V}^k$ by solving:

$$\begin{aligned}\mathbf{V}^{k+1} &= \arg\min_{\mathbf{V}} \mathscr{L}_\rho(\mathbf{W}^{k+1}, \mathbf{V}; \boldsymbol{\eta}^k) \\ &= \lambda\|\mathbf{V}\|_1 + \frac{\rho}{2}\left\|\mathbf{V} - \mathbf{W}^{k+1} + \boldsymbol{\eta}^k\right\|^2 \\ &= \text{sign}(\mathbf{W}^{k+1} + \boldsymbol{\eta}^k/\rho) \circ \max(|\mathbf{W}^{k+1} + \boldsymbol{\eta}^k/\rho| - \lambda/\rho, 0)\end{aligned} \tag{27}$$

  where sign represents the signum function, and $\circ$ is the pointwise product.

- Calculate η^k by solving:

$$\boldsymbol{\eta}^{k+1} = \boldsymbol{\eta}^k + \rho(\mathbf{V}^{k+1} - \mathbf{W}^{k+1}) \tag{28}$$

3. Output: $\mathbf{W}^{k+1}$

6 Matlab example