Structured Preconditioning for Neural Network Training

Qiang Ye

(Joint work with Susanna Lange and Kyle Helfrich)
University of Kentucky

Conference on Fast Direct Solvers - Purdue University
Outline

1. Deep Learning

2. Batch Normalization

3. Batch Normalization Preconditioning (BNP)

4. Experiments
Deep Learning
Supervised Learning

Supervised Learning

Given a labeled data set \( \{(x_i, y_i)\}_{i=1}^{N} \subset \mathbb{R}^m \times \mathbb{R}^n \), fit a parametric family of functions \( y = f(x, \theta) \in \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}^n \) to the data;
Supervised Learning

Given a labeled data set \( \{(x_i, y_i)\}_{i=1}^{N} \subset \mathbb{R}^m \times \mathbb{R}^n \), fit a parametric family of functions \( y = f(x, \theta) \in \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}^n \) to the data:

- Use a neural network for \( f(x, \theta) \)
- Choose a loss function \( L(f(x_i, \theta), y_i) \)
- find \( \theta \in \mathbb{R}^p \) by minimizing \( \mathcal{L}(\theta) := \frac{1}{N} \sum_{i=1}^{N} L(f(x_i, \theta), y_i) \)
Supervised Learning

Given a labeled data set \( \{(x_i, y_i)\}_{i=1}^{N} \subset \mathbb{R}^m \times \mathbb{R}^n \), fit a parametric family of functions \( y = f(x, \theta) \in \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}^n \) to the data;

- Use a neural network for \( f(x, \theta) \)
- Choose a loss function \( L(f(x_i, \theta), y_i) \)
- find \( \theta \in \mathbb{R}^p \) by minimizing \( L(\theta) := \frac{1}{N} \sum_{i=1}^{N} L(f(x_i, \theta), y_i) \)
Deep Neural Network

- Composition of $L$ functions:
  
  $$f(x, \theta) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

- hidden variables at $\ell$-th layer:
  
  $$h^{(\ell)} = f^{(\ell)}(h^{(\ell-1)})$$
  
  $$:= g(W^{(\ell)}h^{(\ell-1)} + b^{(\ell)})$$

- $g(t)$: an elementwise nonlinear activation function: e.g.
  $$g(t) = \max\{t, 0\}$$

Image source: Goodfellow, et al.
Loss Function

Loss for a model output \( \hat{y} := f(x, \theta) \):

- **Regression**: MSE

  \[
  L(\hat{y}, y) = \| \hat{y} - y \|^2
  \]

- **Classification**: Cross-Entropy

  \[
  L(\hat{y}, y) = \sum_j y_j \log \hat{y}_j
  \]
Gradient descent:

\[ \theta \leftarrow \theta - \lambda \nabla L(\theta) \]

- \( \lambda > 0 \) - learning rate
- Mini-batch training: sample a mini-batch \( \{x_{i_1}, x_{i_2}, \ldots, x_{i_N}\} \) and train with

\[ \nabla L(\theta) = \frac{1}{N} \sum_{j=1}^{N} \nabla L(f(x_{i_j}, \theta), y_{i_j}) \]

- Accelerations: Momentum, Adagrad, RMSProp, Adams, Batch normalization, ...
Consider the $\ell$-th hidden layer:

$$h^{(\ell)} = g(W^{(\ell)}h^{(\ell-1)} + b^{(\ell)}); \quad h^{(0)} = x.$$
Batch Normalization (BN)

Consider the $\ell$-th hidden layer:

$$h^{(\ell)} = g(W^{(\ell)}h^{(\ell-1)} + b^{(\ell)}); \quad h^{(0)} = x.$$ 

For a mini-batch of inputs $\{x_1, x_2, \ldots, x_N\}$, the corresponding $h^{(\ell-1)}$:

$$A = \{h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)}\}.$$
Batch Normalization (BN)

Consider the $\ell$-th hidden layer:

$$h^{(\ell)} = g(W^{(\ell)}h^{(\ell-1)} + b^{(\ell)}); \quad h^{(0)} = x.$$ 

For a mini-batch of inputs $\{x_1, x_2, \ldots, x_N\}$, the corresponding $h^{(\ell-1)}$:

$$A = \{h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)}\}.$$

BN - Ioffe and Szegedy (2015)

- Internal Covariate Shift during training, where the statistics of a hidden variable changes due to
  - mini-batch inputs
  - training
- Normalize the hidden variable statistics
BN replaces the $\ell$-th hidden layer by

$$h^{(\ell)} = g \left( \mathcal{W}^{(\ell)} B_{\beta,\gamma} (h^{(\ell-1)}) + b^{(\ell)} \right)$$

where

$$B_{\beta,\gamma} \left( h^{(\ell-1)} \right) = \gamma \frac{h^{(\ell-1)} - \mu_A}{\sigma_A} + \beta$$

$\mu_A, \sigma_A^2$ are mean and variance of $A$, and $\gamma, \beta$ are the re-scaling and re-centering trainable parameters.
Batch Normalization Training Network

Advantages of BN

- Faster Convergence
- Larger learning rate;
- Better generalization

Partial Analysis:
- Improved Lipschitzness of the loss and boundedness of Hessian - Santurkar et al. (2019)
Batch Normalization Training Network

Advantages of BN

- Faster Convergence
- Larger learning rate;
- Better generalization

Partial Analysis:

- Improved Lipschitzness of the loss and boundedness of Hessian - Santurkar et al. (2019)
Batch Normalization

Difficulties:

- Training Network contains $B_{\beta, \gamma}(h^{(\ell-1)})$ that depends on mini-batch
- Inference network has one input and $\mu_A$ and $\sigma_A$ are not defined:
  - Use mean $\mu_A$ and $\sigma_A$ computed during training.
- Small mini-batch sizes.
- Lack of theoretical understanding:
  - Different ways that are applied to CNNs and RNNs.
Batch Normalization

Difficulties:

- Training Network contains $B_{\beta, \gamma} (h^{(\ell-1)})$ that depends on mini-batch
- Inference network has one input and $\mu_A$ and $\sigma_A$ are not defined:
  - Use mean $\mu_A$ and $\sigma_A$ computed during training.
- Small mini-batch sizes.
- Lack of theoretical understanding:
  - Different ways that are applied to CNNs and RNNs.

Some Alternatives:

- Batch Renormalization - Ioffe (2017);
- Layer Normalization (LN) - Ba, Kiros, and Hinton (2016)
- Group Normalization (GN) - Wu and He (2018)
- $\text{BN+GN}$ and other techniques - Summers and Dinneen (2020)
Preconditioned Gradient Descent
Preconditioned Gradient Descent

Gradient Descent:

\[ \theta_{k+1} \leftarrow \theta_k - \alpha \nabla \theta L(\theta) \]

Let \( \theta^* \) be a local minimizer and \( \lambda_{\text{min}} > 0 \) and \( \lambda_{\text{max}} \) be the minimum and maximum eigenvalues of Hessian \( \nabla^2_\theta \mathcal{L}(\theta^*) \).

\[
\| \theta_{k+1} - \theta^* \|_2 \leq (r + \epsilon) \| \theta_k - \theta^* \|_2
\]

where \( r = \max\{ |1 - \alpha \lambda_{\text{min}}|, |1 - \alpha \lambda_{\text{max}}| \} \)
Preconditioned Gradient Descent

Gradient Descent:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_{\theta} L(\theta)$$

Let $\theta^*$ be a local minimizer and $\lambda_{\text{min}} > 0$ and $\lambda_{\text{max}}$ be the minimum and maximum eigenvalues of Hessian $\nabla^2_{\theta} L(\theta^*)$.

$$\|\theta_{k+1} - \theta^*\|_2 \leq (r + \epsilon)\|\theta_k - \theta^*\|_2$$

where $r = \max\{|1 - \alpha \lambda_{\text{min}}|, |1 - \alpha \lambda_{\text{max}}|\}$

- Need $\alpha < 2/\|\nabla^2_{\theta} L(\theta^*)\|$.
- Optimal $r = \frac{\kappa - 1}{\kappa + 1}$ where $\kappa = \kappa(\nabla^2_{\theta} L(\theta^*)) = \lambda_{\text{max}}/\lambda_{\text{min}}$ is the condition number.
Consider a change of variable: $\theta = Pz$ and $L = L(\theta) = L(Pz)$.
Preconditioned Gradient Descent

Consider a change of variable: $\theta = Pz$ and $L = L(\theta) = L(Pz)$.

- Gradient Descent in $z$:

$$z_{k+1} = z_k - \alpha P^T \nabla_\theta L(Pz_k)$$
Consider a change of variable: $\theta = Pz$ and $L = L(\theta) = L(Pz)$.

- Gradient Descent in $z$:

$$z_{k+1} = z_k - \alpha P^T \nabla_{\theta} L(Pz_k)$$

Equivalently

$$\theta_{k+1} = \theta_k - \alpha PP^T \nabla_{\theta} L(\theta_k).$$
Preconditioned Gradient Descent

Consider a change of variable: \( \theta = Pz \) and \( L = L(\theta) = L(Pz) \).

- **Gradient Descent in** \( z \):
  \[
  z_{k+1} = z_k - \alpha P^T \nabla_\theta L(Pz_k)
  \]

  Equivalently
  \[
  \theta_{k+1} = \theta_k - \alpha PP^T \nabla_\theta L(\theta_k).
  \]

- \( \|z_{k+1} - z^*\| \leq (r + \epsilon) \|z_k - z^*\| \) where \( r = \frac{\kappa' - 1}{\kappa' + 1} \)

  \[
  \kappa' = \kappa(\nabla^2_z L(Pz^*)) = \kappa(P^T \nabla^2_\theta L(\theta^*) P).
  \]
Preconditioned Gradient Descent

Consider a change of variable: \( \theta = Pz \) and \( L = L(\theta) = L(Pz) \).

- **Gradient Descent in** \( z \):
  \[
  z_{k+1} = z_k - \alpha P^T \nabla_\theta L(Pz_k)
  \]

  Equivalently
  \[
  \theta_{k+1} = \theta_k - \alpha PP^T \nabla_\theta L(\theta_k).
  \]

- \( \|z_{k+1} - z^*\| \leq (r + \epsilon) \|z_k - z^*\| \) where \( r = \frac{\kappa' - 1}{\kappa' + 1} \)

  \[
  \kappa' = \kappa(\nabla^2_z L(Pz^*)) = \kappa(P^T \nabla^2_\theta L(\theta^*) P).
  \]

- **Preconditioning**: choose \( P \) such that \( P^T \nabla^2_\theta L(\theta^*) P \) has a better condition number
Consider one weight and bias for layer $\ell$. Recall

$$h^{(\ell)} = g \left( W^{(\ell)} h^{(\ell-1)} + b^{(\ell)} \right) \in \mathbb{R}^n$$

Let $w_i^{(\ell)^T} \in \mathbb{R}^{1 \times m}$ be the $i$th row of $W^{(\ell)}$ and $b_i^{(\ell)}$ be the $i$th entry of $b^{(\ell)}$. Let

$$a_i^{(\ell)} = w_i^{(\ell)^T} h^{(\ell-1)} + b_i^{(\ell)} = \hat{w}^T \hat{h} \in \mathbb{R}$$

where

$$\hat{w}^T = \begin{bmatrix} b_i^{(\ell)} & w_i^{(\ell)^T} \end{bmatrix} \in \mathbb{R}^{1 \times (m+1)}, \quad \hat{h} = \begin{bmatrix} 1 \\ h^{(\ell-1)} \end{bmatrix} \in \mathbb{R}^{(m+1) \times 1}.$$
Theorem 1

Consider a loss function $L$ and write $L = L\left( a_i^{(\ell)} \right) = L\left( \hat{w}^T \hat{h} \right)$. When training over a mini-batch of $N$ inputs, let $\{ h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)} \}$ be the associated $h^{(\ell-1)}$ and let $\hat{h}_j = \begin{bmatrix} 1 \\ h_j^{(\ell-1)} \end{bmatrix} \in \mathbb{R}^{(m+1) \times 1}$. Let $\mathcal{L} = \mathcal{L}(\hat{w}) := \frac{1}{N} \sum_{j=1}^{N} L\left( \hat{w}^T \hat{h}_j \right)$.

Then,

$$\nabla^2_{\hat{w}} \mathcal{L}(\hat{w}) = \hat{H}^T S \hat{H}$$

where

$$\hat{H} = \begin{bmatrix} 1 & h_1^{(\ell-1)^T} \\ \vdots & \vdots \\ 1 & h_N^{(\ell-1)^T} \end{bmatrix} \quad \text{and} \quad S = \frac{1}{N} \begin{bmatrix} L'' \left( \hat{w}^T \hat{h}_1 \right) \\ \vdots \\ L'' \left( \hat{w}^T \hat{h}_N \right) \end{bmatrix}.$$
Batch Normalization Preconditioning (BNP)

Precondition $\hat{H} = [e, H]$:

- $\hat{w} = Pz$, where

$$P := UD, \quad U := \begin{bmatrix} 1 & -\mu_A^T \\ 0 & I \end{bmatrix}, \quad D := \begin{bmatrix} 1 & 0 \\ 0 & \text{diag}(\sigma_A) \end{bmatrix}^{-1},$$

where

$$\mu_A := \frac{1}{N} \sum_{j=1}^{N} h_j^{(\ell-1)}, \quad \text{and} \quad \sigma_A^2 := \frac{1}{N} \sum_{j=1}^{N} (h_j^{(\ell-1)} - \mu_A)^2$$
Theorem 2

The preconditioned Hessian matrix is

\[
\nabla^2_{z} \mathcal{L} = P^T \nabla^2_{\hat{w}} \mathcal{L} P = \hat{G}^T S \hat{G}.
\]

where \( \hat{G} := \hat{H} P \), i.e.

\[
\hat{G} = \begin{bmatrix}
1 & g_1^T \\
\vdots & \vdots \\
1 & g_N^T
\end{bmatrix} = \begin{bmatrix}
1 & h_{1}^{(\ell-1)^T} \\
\vdots & \vdots \\
1 & h_{N}^{(\ell-1)^T}
\end{bmatrix} \begin{bmatrix}
1 & -\mu_A^T \\
0 & I
\end{bmatrix} \begin{bmatrix}
1 & 0 \\
0 & \text{diag}(\sigma_A)
\end{bmatrix}^{-1}, \tag{1}
\]

and \( g_j = (h_j^{(\ell-1)} - \mu_A)/\sigma_A \) is \( h_j^{(\ell-1)} \) normalized to have zero mean and unit variance.
\( \hat{G} = HUD \) or \( g_j = (h_j^{(\ell-1)} - \mu_A)/\sigma_A \) improves conditioning in two ways:
\[ \hat{G} = HUD \text{ or } g_j = (h_j^{(\ell-1)} - \mu_A)/\sigma_A \text{ improves conditioning in two ways:} \]

**Theorem 3**

\[ \kappa(\hat{HU}) \leq \kappa(\hat{H}) \]

and (by a theorem of van der Sluis)

\[ \kappa(\hat{G}) \leq \sqrt{m + 1} \min_{D_0 \text{ is diagonal}} \kappa(\hat{HUD}_0). \]
$\hat{G} = HUD$ or $g_j = (h_j^{(\ell-1)} - \mu_A)/\sigma_A$ improves conditioning in two ways:

**Theorem 3**

$$\kappa(\hat{HU}) \leq \kappa(\hat{H})$$

and (by a theorem of van der Sluis)

$$\kappa(\hat{G}) \leq \sqrt{m + 1} \min_{D_0 \text{ is diagonal}} \kappa(\hat{HUD}_0).$$

If there is a large variations in $\sigma_A$, then $\kappa(\hat{G}) \ll \sqrt{m + 1} \kappa(\hat{H})$. 
G’s entries has mean 0 and variance 1. By a theorem of Seginer:

\[ \mathbb{E}[\| G \|] \leq C \max\{\sqrt{m}, \sqrt{N}\} \]

\[ \mathbb{E}[\| \hat{G} \|] = \max\{\sqrt{N}, \mathbb{E}[\| G \|]\} \leq C' \max\{\sqrt{m}, \sqrt{N}\} \]
Balancing the norms of Hessians

G’s entries has mean 0 and variance 1. By a theorem of Seginer:

\[ \mathbb{E}[\|G\|] \leq C \max\{\sqrt{m}, \sqrt{N}\} \]

\[ \mathbb{E}[\|\hat{G}\|] = \max\{\sqrt{N}, \mathbb{E}[\|G\|]\} \leq C' \max\{\sqrt{m}, \sqrt{N}\} \]

Scale \( \nabla^2_z L(\theta^*) \) by \( q = \max\{\sqrt{m/N}, 1\} \) to get similar norms for all layers:

\[ (1/q) \mathbb{E}[\|\hat{G}\|] \leq C' \sqrt{N} \]

- Learning rate: \( \alpha < 2/\|\nabla^2_z L(\theta^*)\| \).
- A large \( \|\nabla^2_z L(\theta^*)\| \) at one layer will require a smaller \( \alpha \);
BNP Gradient Algorithm

### BNP Gradients on $W^{(\ell)}, b^{(\ell)}$

**Input:** $A = \{h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)}\} \subset \mathbb{R}^m$ and the parameter gradients: $G_w \leftarrow \frac{\partial L}{\partial W^{(\ell)}} \in \mathbb{R}^{n \times m}$, $G_b \leftarrow \frac{\partial L}{\partial b^{(\ell)}} \in \mathbb{R}^{1 \times n}$

1. Compute $\mu_A, \sigma^2_A$;
2. Compute:\[ \mu \leftarrow \rho \mu + (1 - \rho) \mu_A, \quad \sigma^2 \leftarrow \rho \sigma^2 + (1 - \rho) \sigma^2_A; \]
3. Set $\tilde{\sigma}^2 = \sigma^2 + \epsilon_1 \max\{\sigma^2\} + \epsilon_2$ and $q^2 = \max\{m/N, 1\}$;
4. Update:\[ G_w \leftarrow \frac{1}{q} (G_w - \mu G_b)/\tilde{\sigma}^2; \quad G_b \leftarrow \frac{1}{q} G_b - \mu^T G_w; \]

The same framework is applied to CNNs.

Use mean and variance of hidden tensor over the mini-batch and the spacial dimensions, as used in BN.
BNP Algorithm

BNP Gradients on $W^{(\ell)}, b^{(\ell)}$

**Input:** $A = \{ h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)} \} \subset \mathbb{R}^m$ and the parameter gradients: $G_w \leftarrow \frac{\partial L}{\partial W^{(\ell)}} \in \mathbb{R}^{n \times m}$, $G_b \leftarrow \frac{\partial L}{\partial b^{(\ell)}} \in \mathbb{R}^{1 \times n}$

1. Compute $\mu_A, \sigma^2_A$;
2. Compute: $\mu \leftarrow \rho \mu + (1 - \rho) \mu_A$, $\sigma^2 \leftarrow \rho \sigma^2 + (1 - \rho) \sigma^2_A$;
3. Set $\tilde{\sigma}^2 = \sigma^2 + \epsilon_1 \max\{\sigma^2\} + \epsilon_2$ and $q^2 = \max\{m/N, 1\}$;
4. Update: $G_w \leftarrow \frac{1}{q} (G_w - \mu G_b)/\tilde{\sigma}^2$; $G_b \leftarrow \frac{1}{q} G_b - \mu^T G_w$;

The same framework is applied to CNNs.

- Use mean and variance of hidden tensor over the mini-batch and the spacial dimensions, as used in BN.
Experiments
Datasets

- CIFAR10: 60,000 labeled 32x32 color images with 50,000/10,000 split for training/testing. There are 10 classes.
Fully Connected Neural Network: three hidden layers of size 100 each and an output layer of size 10

Figure: Mini-batch size = 60.
Fully Connected Network/CIFAR 10

Batchsize = 6

**Figure:** Mini-batch size = 6.
5-layer CNN: 3 convolution layers of $3 \times 3$ kernel with 32-64-32 filters, followed by two dense layers.

**Figure:** Mini-batch size = 2
ResNet-110: 54 residual blocks, containing two $3 \times 3$ convolution layer each.

**Figure:** ResNet BS=128
Conclusions

- Preconditioning framework applicable to a variety of networks.
- Outperform BN for small mini-batches.
- Provide partial theoretical justifications for BN.
- Work in progress: applications to other network architectures.