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

Deep Learning

Supervised Learning

Supervised Learning

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

Supervised Learning

Supervised Learning

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

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

Supervised Learning

Supervised Learning

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

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

Deep Neural Network

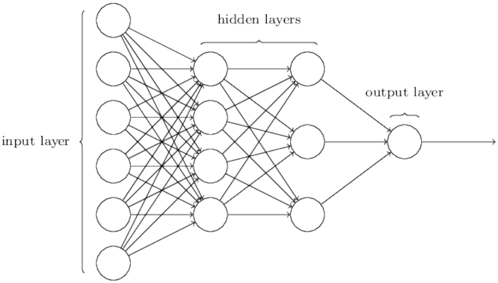

Image source: Goodfellow, et al.

- Composition of L functions:

$$f(\mathbf{x}, \theta) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$$

- hidden variables at ℓ-th layer:

$$\begin{aligned} h^{(\ell)} &= f^{(\ell)}(h^{(\ell-1)}) \\ &:= g(W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}) \end{aligned}$$

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

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

$$L(\hat{y}, y) = \sum_j y_j \log \hat{y}_j$$

Optimization/Training

Gradient descent:

$$\theta \leftarrow \theta - \lambda \nabla \mathcal{L}(\theta)$$

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

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

- Accelerations: Momentum, Adagrad, RMSProp, Adams, Batch normalization, ...

Batch Normalization (BN)

Consider the ℓ-th hidden layer:

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

Batch Normalization (BN)

Consider the ℓ-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 ℓ-th hidden layer:

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

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

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

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 ℓ-th hidden layer by

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

where

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

μ_A, σ_A^2 are mean and variance of A, and γ, β are the re-scaling and re-centering trainable parameters.

Batch Normalization Training Network

Advantages of BN

- Faster Convergence
- Larger learning rate;
- Better generalization

Batch Normalization Training Network

Advantages of BN

- Faster Convergence
- Larger learning rate;
- Better generalization

Partial Analysis:

- Scale-invariant properties - Arora et al. (2018), Cho and Lee (2017)
- Improved Lipschitzness of the loss and boundedness of Hessian - Santurkar et al. (2019)
- Convergence analysis of special 1-layer/2-layer networks - Cai et al. (2019), Kohler et al. (2018), Ma and Klabjan (2019)

Batch Normalization

Difficulties:

- Training Network contains $\mathcal{B}_{\beta,\gamma}\left(h^{(\ell-1)}\right)$ that depends on mini-batch
- Inference network has one input and μ_A and σ_A are not defined:
  - Use mean μ_A and σ_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 $\mathcal{B}_{\beta,\gamma}\left(h^{(\ell-1)}\right)$ that depends on mini-batch
- Inference network has one input and μ_A and σ_A are not defined:
  - Use mean μ_A and σ_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)`
- 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 θ^* be a local minimizer and $\lambda_{\text{min}} > 0$ and λ_{max} be the minimum and maximum eigenvalues of Hessian $\nabla_\theta^2 \mathcal{L}(\theta^*)$.

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

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 θ^* be a local minimizer and $\lambda_{\min} > 0$ and $\lambda_{\max}$ 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 $r = \max\{|1 - \alpha\lambda_{\min}|, |1 - \alpha\lambda_{\max}|\}$

- Need $\alpha < 2/\|\nabla^2_\theta L(\theta^*)\|$.
- Optimal $r = \frac{\kappa - 1}{\kappa + 1}$ where $\kappa = \kappa(\nabla^2_\theta \mathcal{L}(\theta^*)) = \lambda_{\max}/\lambda_{\min}$ is the condition number.

Preconditioned Gradient Descent

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:

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

Preconditioned Gradient Descent

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

- Gradient Descent in z:

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

  Equivalently

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

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

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

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

$$\kappa' = \kappa(\nabla_z^2 L(Pz^*)) = \kappa(P^T \nabla_\theta^2 L(\theta^*) P).$$

Preconditioned Gradient Descent

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

- Gradient Descent in z:

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

  Equivalently

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

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

$$\kappa' = \kappa(\nabla_z^2 L\left(Pz^*\right)) = \kappa(P^T \nabla_\theta^2 L\left(\theta^*\right) P).$$

- Preconditioning: choose P such that $P^T \nabla_\theta^2 L\left(\theta^*\right) P$ has a better condition number

Neural Networks Loss

Consider one weight and bias for layer ℓ. 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 ith row of $W^{(\ell)}$ and $b_i^{(\ell)}$ be the ith entry of $b^{(\ell)}$. Let

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

where

$$\widehat{w}^T = \left[b_i^{(\ell)}, w_i^{(\ell)^T}\right] \in \mathbb{R}^{1\times(m+1)},\ \widehat{h} = \begin{bmatrix} 1 \\ h^{(\ell-1)} \end{bmatrix} \in \mathbb{R}^{(m+1)\times 1},$$

Neural Network Loss Hessian

Theorem 1

Consider a loss function L *and write* $L = L\left(a_i^{(\ell)}\right) = L\left(\widehat{w}^T \widehat{h}\right)$. *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* $\widehat{h}_j = \begin{bmatrix} 1 \\ h_j^{(\ell-1)} \end{bmatrix} \in \mathbb{R}^{(m+1)\times 1}$. *Let* $\mathcal{L} = \mathcal{L}(\widehat{w}) := \frac{1}{N}\sum_{j=1}^{N} L\left(\widehat{w}^T \widehat{h}_j\right)$. *Then,*

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

where

$$\widehat{H} = \begin{bmatrix} 1 & {h_1^{(\ell-1)}}^T \\ \vdots & \vdots \\ 1 & {h_N^{(\ell-1)}}^T \end{bmatrix} \text{ and } S = \frac{1}{N}\begin{bmatrix} L''\left(\widehat{w}^T \widehat{h}_1\right) & & \\ & \ddots & \\ & & L''\left(\widehat{w}^T \widehat{h}_N\right) \end{bmatrix},$$

Batch Normalization Preconditioning (BNP)

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

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

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

where

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

Preconditioned Gradient Descent

Theorem 2

The preconditioned Hessian matrix is

$$\nabla_z^2 \mathcal{L} = P^T \nabla_{\widehat{w}}^2 \mathcal{L} P = \widehat{G}^T S \widehat{G}.$$

where $\widehat{G} := \widehat{H} P$, *i.e.*

$$\widehat{G} = \begin{bmatrix} 1 & g_1^T \\ \vdots & \vdots \\ 1 & g_N^T \end{bmatrix} = \begin{bmatrix} 1 & h_1^{(\ell-1)^T} \\ \vdots & \vdots \\ 1 & h_N^{(\ell-1)^T} \end{bmatrix} \begin{bmatrix} 1 & -\mu_A^T \\ 0 & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & diag(\sigma_A) \end{bmatrix}^{-1}, \quad (1)$$

and $g_j = (h_j^{(\ell-1)} - \mu_A)/\sigma_A$ *is* $h_j^{(\ell-1)}$ *normalized to have zero mean and unit variance.*

Preconditioned Gradient Descent

$\widehat{G} = HUD$ or $g_j = (h_j^{(\ell-1)} - \mu_A)/\sigma_A$ improves conditioning in two ways:

Preconditioned Gradient Descent

$\widehat{G} = HUD$ or $g_j = (h_j^{(\ell-1)} - \mu_A)/\sigma_A$ improves conditioning in two ways:

Theorem 3

$$\kappa(\widehat{H}U) \leq \kappa(\widehat{H})$$

and (by a theorem of van der Sluis)

$$\kappa(\widehat{G}) \leq \sqrt{m+1} \min_{D_0 \text{ is diagonal}} \kappa(\widehat{H}UD_0).$$

Preconditioned Gradient Descent

$\widehat{G} = HUD$ or $g_j = (h_j^{(\ell-1)} - \mu_A)/\sigma_A$ improves conditioning in two ways:

Theorem 3

$$\kappa(\widehat{H}U) \le \kappa(\widehat{H})$$

and (by a theorem of van der Sluis)

$$\kappa(\widehat{G}) \le \sqrt{m+1} \min_{D_0 \text{ is diagonal}} \kappa(\widehat{H}UD_0).$$

If there is a large variations in σ_A, then $\kappa(\widehat{G}) \ll \sqrt{m+1}\kappa(\widehat{H})$.

Balancing the norms of Hessians

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

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

$$\mathbb{E}[\|\widehat{G}\|] = \max\{\sqrt{N}, \mathbb{E}[\|G\|]\} \le C' \max\{\sqrt{m}, \sqrt{N}\}$$

Balancing the norms of Hessians

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

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

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

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

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

- Learning rate: $\alpha < 2/\|\nabla_z^2 L(\theta^*)\|$.
- A large $\|\nabla_z^2 L(\theta^*)\|$ at one layer will require a smaller α;

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: $G_w \leftarrow \frac{\partial \mathcal{L}}{\partial W^{(\ell)}} \in \mathbb{R}^{n\times m}$, $G_b \leftarrow \frac{\partial \mathcal{L}}{\partial b^{(\ell)}} \in \mathbb{R}^{1\times n}$

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

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: $G_w \leftarrow \frac{\partial \mathcal{L}}{\partial W^{(\ell)}} \in \mathbb{R}^{n \times m}$, $G_b \leftarrow \frac{\partial \mathcal{L}}{\partial b^{(\ell)}} \in \mathbb{R}^{1 \times n}$

1. Compute μ_A, σ_A^2;
2. Compute: $\mu \leftarrow \rho\mu + (1-\rho)\mu_A$, $\sigma^2 \leftarrow \rho\sigma^2 + (1-\rho)\sigma_A^2$;
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 Network/CIFAR 10

Fully-Connected Neural Network: three hidden layers of size 100 each and an output layer of size 10

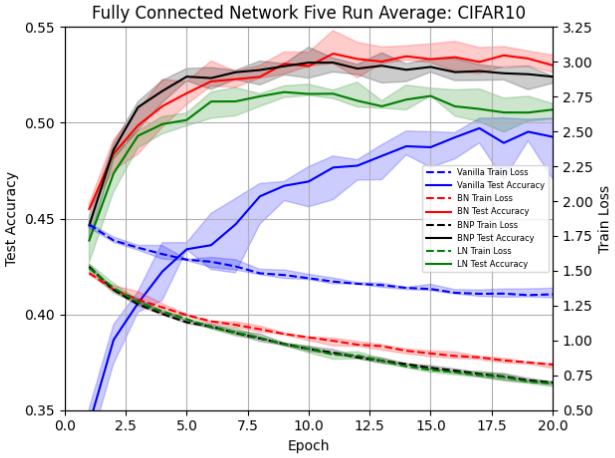

Figure: Mini-batch size = 60.

Fully Connected Network/CIFAR 10

Batchsize = 6

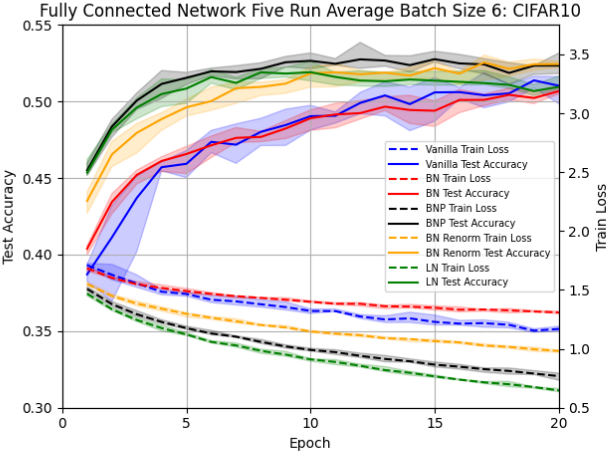

Figure: Mini-batch size = 6.

5-layer CNN: 3 convolution layers of 3×3 kernel with 32-64-32 filters, followed by two dense layers.

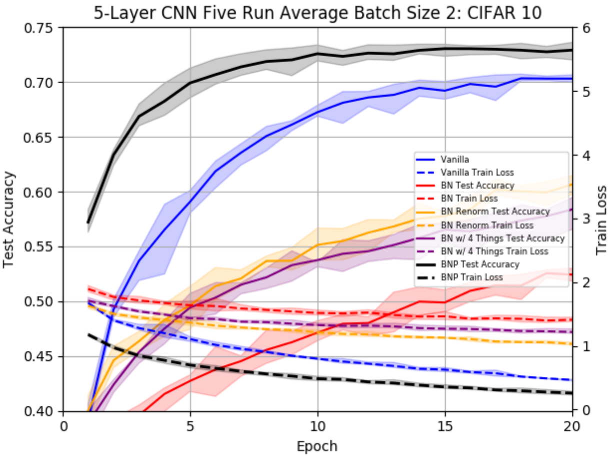

Figure: Mini-batch size $=2$

ResNet110/CIFAR10

ResNet-110: 54 residual blocks, containing two 3×3 convolution layer each.

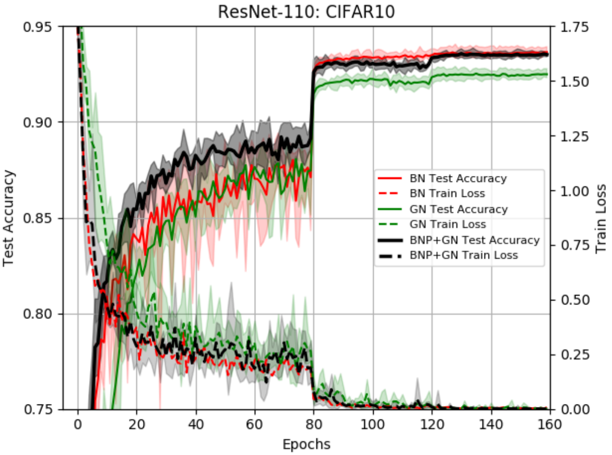

Figure: ResNet BS=128

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.