Summary of Two NIPS 2015 Deep Learning Optimization Papers

Muthu Chidambaram

Department of Computer Science, University of Virginia

https://qdata.github.io/deep2Read/

Hessian-free Optimization for Learning Deep Multidimensional Recurrent Neural Networks

---

- Authors: Minhyung Cho, Chandra Shekhar Dir, Jaehyung Lee
- Proposes method to overcome difficulty of training multidimensional recurrent neural network (MDRNN) using Hessian-free optimization
- MDRNNs have not kept up in depth with typical feedforward neural networks, aims to address that via Hessian-free optimization

Hessian-free Optimization for Learning Deep Multidimensional Recurrent Neural Networks

---

- MDRNN's are a generalization of RNN's that have recurrent connections corresponding to the number of dimensions of the data
- Sequence labeling task uses connectionist temporal classification (CTC) as objective function
- Hessian-free optimization minimizes objective function by constructing and minimizing a local quadratic approximation (loss function L, curvature approx. G)

$$\delta_n = \theta - \theta_n \qquad Q_n(\theta) = \mathcal{L}(\theta_n) + \nabla_\theta \mathcal{L}|_{\theta_n}^\top \delta_n + \frac{1}{2}\delta_n^\top G \delta_n$$

Hessian-free Optimization for Learning Deep Multidimensional Recurrent Neural Networks

---

- Hessian of the objective: $H_{\mathcal{L}\circ\mathcal{N}} = J_{\mathcal{N}}^{\top} H_{\mathcal{L}} J_{\mathcal{N}} + \sum_{i=1}^{KT} [J_{\mathcal{L}}]_i H_{[\mathcal{N}]_i}$
- Approximation to the Hessian: $G_{\mathcal{L}\circ\mathcal{N}} = J_{\mathcal{N}}^{\top} H_{\mathcal{L}} J_{\mathcal{N}}$
  - Since it may not be positive semidefinite
- GGN can be written as: $G_{\mathcal{L}\circ\mathcal{N}} = \sum_t J_{\mathcal{N}_t}^{\top} H_{\mathcal{L},t} J_{\mathcal{N}_t},$
- Convex approximation for CTC derived from GGN approximation

Hessian-free Optimization for Learning Deep Multidimensional Recurrent Neural Networks

---

- CTC: Mapping from an output sequence a to scalar loss
  - Output activations at time t are normalized using softmax
- Conditional probability of the path pi is product of label probabilities at each timestep
- CTC loss function is sum of product of softmax components
  - Target sequence z
- CTC objective is reformulated to separate non-convexity

$$y_k^t = \frac{\exp(a_k^t)}{\sum_{k'} \exp(a_{k'}^t)} \qquad p(\pi|\mathbf{a}) = \prod_{t=1}^{T} y_{\pi_t}^t \qquad \mathcal{L}(\mathbf{a}) = -\log p(\mathbf{z}|\mathbf{a}).$$

Hessian-free Optimization for Learning Deep Multidimensional Recurrent Neural Networks

---

- Reformulated CTC function: $\mathcal{L}(\mathbf{a}) = \mathcal{L}_c \circ \mathcal{N}_c(\mathbf{a})$
- GGN approximation of reformulated CTC function gives a convex approximation for Hessian of CTC

$$G_{\mathcal{L}_c \circ \mathcal{N}_c} = J_{\mathcal{N}_c}^{\top} H_{\mathcal{L}_c} J_{\mathcal{N}_c} \qquad \mathcal{L}_p(F) = \log\left(\sum_{\mathbf{z}' \in S} \exp(f_{\mathbf{z}'})\right) \qquad \mathcal{L}_c(F) = -\log \frac{\exp(f_{\mathbf{z}})}{\sum_{\mathbf{z}' \in S} \exp(f_{\mathbf{z}'})} = -f_{\mathbf{z}} + \log\left(\sum_{\mathbf{z}' \in S} \exp(f_{\mathbf{z}'})\right)$$

1. $H_{\mathcal{L}} = H_{\mathcal{L}_c \circ \mathcal{N}_c}$ is not positive semidefinite.
2. $G_{\mathcal{L}_c \circ \mathcal{N}_c} = G_{\mathcal{L}_p \circ \mathcal{N}_c}$ is positive semidefinite, but not computationally tractable.
3. $H_{\mathcal{L}_p \circ \mathcal{N}_c}$ is positive semidefinite and computationally tractable.

Hessian-free Optimization for Learning Deep Multidimensional Recurrent Neural Networks

---

- GGN is identical to Fisher information matrix
- Proposed approximation has expectation-maximization interpretation

$$F = \mathbb{E}_{\mathbf{x}}\left[\sum_t J_{\mathcal{N}_t}^\top \mathbb{E}_{\mathbf{l}\sim p(\mathbf{l}|\mathbf{a})}\left[\left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial a^t}\right)^\top \left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial a^t}\right)\right] J_{\mathcal{N}_t}\right] \qquad F = \mathbb{E}_{\mathbf{x}}\left[\sum_t J_{\mathcal{N}_t}^\top (\mathrm{diag}(Y^t) - Y^t Y^{t\top}) J_{\mathcal{N}_t}\right]$$

Expectation step calculates: $\gamma_{\pi|\mathbf{x},\mathbf{z}} = \frac{p(\pi|\mathbf{x},\hat{\theta})}{\sum_{\pi\in\mathcal{B}^{-1}(\mathbf{z})} p(\pi|\mathbf{x},\hat{\theta})}$.

Maximization step updates: $\hat{\theta} = \mathrm{argmax}_\theta \mathcal{Q}(\theta)$, where $\mathcal{Q}(\theta) = \sum_{\pi\in\mathcal{B}^{-1}(\mathbf{z})} \gamma_{\pi|\mathbf{x},\mathbf{z}} \log p(\pi|\mathbf{x},\theta)$.

$H_{\mathcal{Q},t} = \mathrm{diag}(Y^t) - Y^t Y^{t\top}$

Hessian-free Optimization for Learning Deep Multidimensional Recurrent Neural Networks

---

- Datasets used: IFN/ENIT Database of handwritten arabic words, TIMIT corpus for speech recognition
- Experiment setup: Evaluated against stochastic gradient descent using LSTM networks
- Conclusion: Applying Hessian-free optimization to a convex approximation of the CTC loss function led to significant improvements in handling handwriting recognition

Hessian-free Optimization for Learning Deep Multidimensional Recurrent Neural Networks

Table 1: Experimental results for Arabic offline handwriting recognition. The label error rate is presented with the different network depths. A^B denotes a stack of B layers having A hidden LSTM cells in each layer. "Epochs" is the number of epochs required by the network using HF optimization so that the stopping criteria are fulfilled. ϵ is the learning rate and μ is the momentum.

NETWORKS	DEPTH	WEIGHTS	HF (%)	EPOCHS	SGD (%)	$\{\epsilon, \mu\}$
2-10-50	3	159,369	6.10	77	9.57	$\{10^{-4}, 0.9\}$
2-10-21^3	5	157,681	5.85	90	9.19	$\{10^{-5}, 0.99\}$
2-10-14^6	8	154,209	4.98	140	9.67	$\{10^{-4}, 0.95\}$
2-10-12^8	10	154,153	4.95	109	9.25	$\{10^{-4}, 0.95\}$
2-10-10^{11}	13	150,169	4.50	84	10.63	$\{10^{-4}, 0.9\}$
2-10-9^{13}	15	145,417	5.69	84	12.29	$\{10^{-5}, 0.99\}$

Table 2: Experimental results for phoneme recognition using the TIMIT corpus. PER is presented with the different MDRNN architectures (depth $\times$ block $\times$ cell/block). σ is the standard deviation of Gaussian weight noise. The remaining parameters are the same as in Table 1.

NETWORKS	WEIGHTS	HF (%)	EPOCHS	$\{\sigma\}$	SGD (%)	$\{\epsilon, \mu, \sigma\}$
$3 \times 20 \times 10$	771,542	20.14	22	{0.03}	20.96	$\{10^{-5}, 0.99, 0.05\}$
$5 \times 15 \times 10$	795,752	19.18	30	{0.05}	20.82	$\{10^{-4}, 0.9, 0.04\}$
$8 \times 11 \times 10$	720,826	19.09	29	{0.05}	19.68	$\{10^{-4}, 0.9, 0.04\}$
$10 \times 10 \times 10$	755,822	18.79	60	{0.04}	18.46	$\{10^{-5}, 0.95, 0.04\}$
$13 \times 9 \times 10$	806,588	18.59	93	{0.05}	18.49	$\{10^{-5}, 0.95, 0.04\}$
$15 \times 8 \times 10$	741,230	18.54	50	{0.04}	19.09	$\{10^{-5}, 0.95, 0.03\}$
$3 \times 250 \times 1^\dagger$	3.8M				18.6	$\{10^{-4}, 0.9, 0.075\}$
$5 \times 250 \times 1^\dagger$	6.8M				18.4	$\{10^{-4}, 0.9, 0.075\}$

Path-SGD: Path Normalized Optimization in Deep Neural Networks

---

- Authors: Behnam Neyshabur, Ruslan Salakhutdinov, Nathan Srebro
- Proposes an approximate steepest descent method with respect to a path-wise regularizer
- Investigates whether L2 geometry is the appropriate geometry for the space of deep neural networks

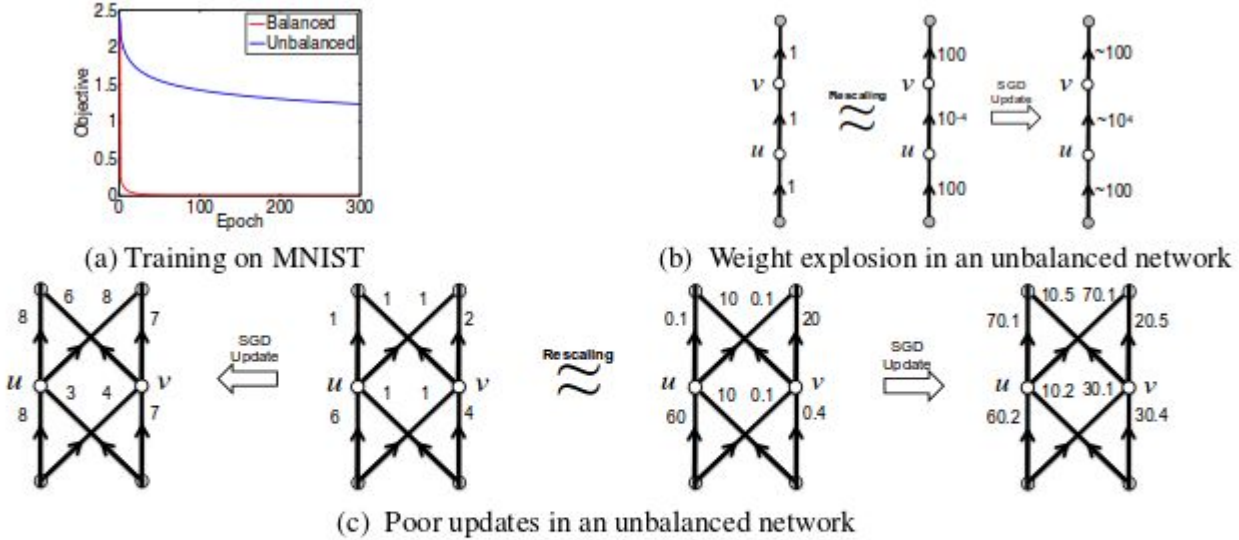
Path-SGD: Path Normalized Optimization in Deep Neural Networks

---

- Considers geometry inspired by max-norm regularization
  - Uses minimum max-norm over all rescalings of weights to achieve rescaling invariance
- Focuses specifically on networks using RELU activation function
  - Due to RELU activation having non-negative homogeneity
- Rescaling function multiplies incoming edges by c and outgoing edges by 1/c

$$\tilde{w}_{(u_1 \to u_2)} = \begin{cases} c.w_{(u_1 \to u_2)} & u_2 = v, \\ \frac{1}{c} w_{(u_1 \to u_2)} & u_1 = v, \\ w_{(u_1 \to u_2)} & \text{otherwise.} \end{cases}$$

Path-SGD: Path Normalized Optimization in Deep Neural Networks



(a) Training on MNIST

(b) Weight explosion in an unbalanced network

(c) Poor updates in an unbalanced network

Figure 1: (a): Evolution of the cross-entropy error function when training a feed-forward network on MNIST with two hidden layers, each containing 4000 hidden units. The unbalanced initialization (blue curve) is generated by applying a sequence of rescaling functions on the balanced initializations (red curve). (b): Updates for a simple case where the input is $x = 1$, thresholds are set to zero (constant), the stepsize is 1, and the gradient with respect to output is $\delta = -1$. (c): Updated network for the case where the input is $x = (1, 1)$, thresholds are set to zero (constant), the stepsize is 1, and the gradient with respect to output is $\delta = (-1, -1)$.

Path-SGD: Path Normalized Optimization in Deep Neural Networks

---

- Goal is to minimize objective of the form: $L(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_w(x_i), y_i)$
- Updates are of the form: $w^{(t+1)} = w^{(t)} + \breve{\Delta} w^{(t+1)}$.
- Gradient descent is not rescaling invariant
- Balanced Network: incoming weights to different units are roughly the same
  - Gradient descent can “blow up” in unbalanced networks
- Group norm and max norm

$$\mu_{p,q}(w) = \left(\sum_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{q/p} \right)^{1/q} \qquad \mu_{p,\infty}(w) = \sup_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{1/p}$$

Path-SGD: Path Normalized Optimization in Deep Neural Networks

---

- Per-unit L2 regularization has been shown to be very effective in the case of RELU activation functions
  - Potentially due to rebalancing such that hidden units have same norm

$$\phi_p(w) = \|\pi(w)\|_p = \left(\sum_{v_{in}[i] \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \ldots \xrightarrow{e_d} v_{out}[j]} \left| \prod_{k=1}^{d} w_{e_k} \right|^p \right)^{1/p} \qquad \phi_p(w) = \min_{\tilde{w} \sim w} \left(\mu_{p,\infty}(\tilde{w}) \right)^d$$

$$\begin{aligned} w^{(t+1)} &= \arg\min_w \ \eta \left\langle \nabla L(w^{(t)}), w \right\rangle + \frac{1}{2} \left\| \pi(w) - \pi(w^{(t)}) \right\|_p^2 \qquad (6) \\ &= \arg\min_w \ \eta \left\langle \nabla L(w^{(t)}), w \right\rangle + \frac{1}{2} \left(\sum_{v_{in}[i] \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \ldots \xrightarrow{e_d} v_{out}[j]} \left| \prod_{k=1}^{d} w_{e_k} - \prod_{k=1}^{d} w_{e_k}^{(t)} \right|^p \right)^{2/p} \\ &= \arg\min_w J^{(t)}(w) \end{aligned}$$

Path-SGD: Path Normalized Optimization in Deep Neural Networks

---

Algorithm 1 Path-SGDupdate rule

1: $\forall_{v \in V_{\text{in}}^0} \ \gamma_{\text{in}}(v) = 1$ ▷ Initialization
2: $\forall_{v \in V_{\text{out}}^0} \ \gamma_{\text{out}}(v) = 1$
3: **for** $i = 1$ **to** d **do**
4: $\quad \forall_{v \in V_{\text{in}}^i} \gamma_{\text{in}}(v) = \sum_{(u \to v) \in E} \gamma_{\text{in}}(u) \left| w_{(u,v)} \right|^p$
5: $\quad \forall_{v \in V_{\text{out}}^i} \gamma_{\text{out}}(v) = \sum_{(v \to u) \in E} \left| w_{(v,u)} \right|^p \gamma_{\text{out}}(u)$
6: **end for**
7: $\forall_{(u \to v) \in E} \ \gamma(w^{(t)}, (u, v)) = \gamma_{\text{in}}(u)^{2/p} \gamma_{\text{out}}(v)^{2/p}$
8: $\forall_{e \in E} w_e^{(t+1)} = w_e^{(t)} - \frac{\eta}{\gamma(w^{(t)}, e)} \frac{\partial L}{\partial w_e}(w^{(t)})$ ▷ Update Rule

$$\gamma_p(w, e) = \left(\sum_{v_{\text{in}}[i] \cdots \xrightarrow{e} \cdots v_{\text{out}}[j]} \ \prod_{e' \neq e} |w_{e'}|^p \right)^{2/p} \qquad \hat{w}_e^{(t+1)} = w_e^{(t)} - \frac{\eta}{\gamma_p(w^{(t)}, e)} \frac{\partial L}{\partial w}(w^{(t)})$$

Path-SGD: Path Normalized Optimization in Deep Neural Networks

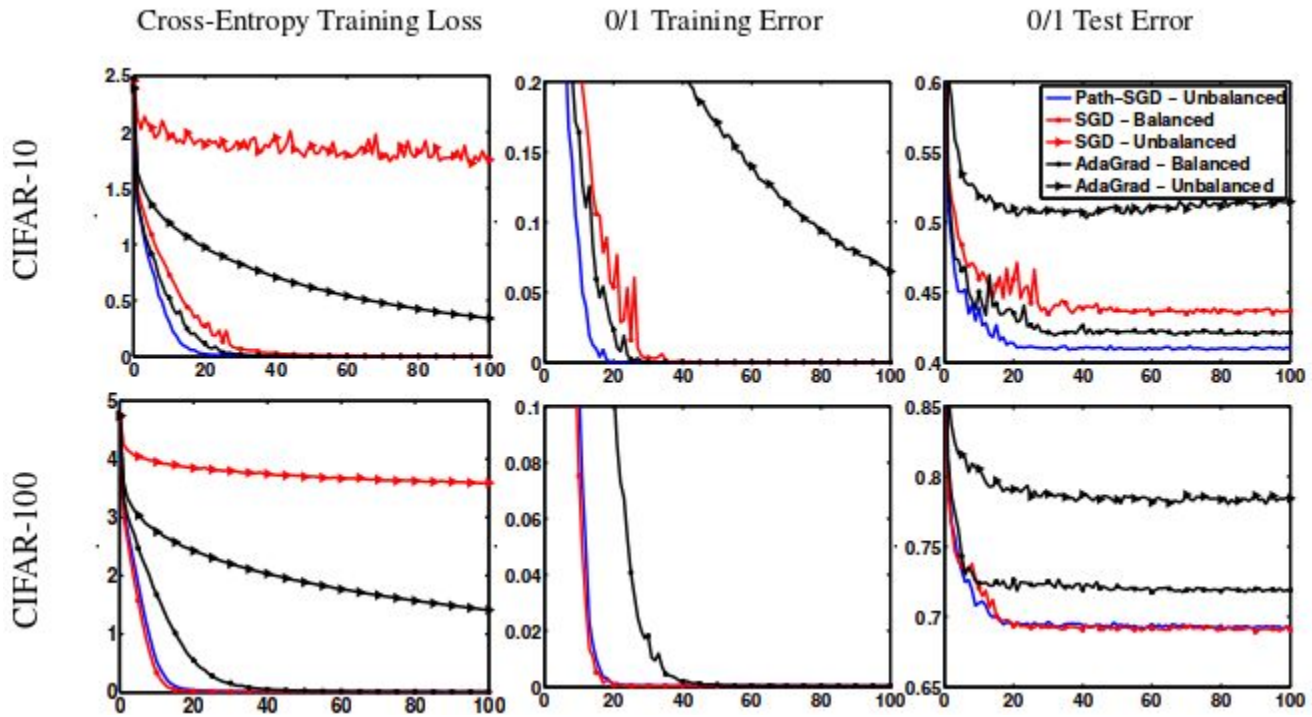
---

- Compare PathSGD to SGD and AdaGrad
  - Trained with both unbalanaced and balanced initializations
  - Trained with and without dropout

Table 1: General information on datasets used in the experiments.

Data Set	Dimensionality	Classes	Training Set	Test Set
CIFAR-10	3072 (32×32 color)	10	50000	10000
CIFAR-100	3072 (32×32 color)	100	50000	10000
MNIST	784 (28×28 grayscale)	10	60000	10000
SVHN	3072 (32×32 color)	10	73257	26032

Path-SGD: Path Normalized Optimization in Deep Neural Networks



Path-SGD: Path Normalized Optimization in Deep Neural Networks

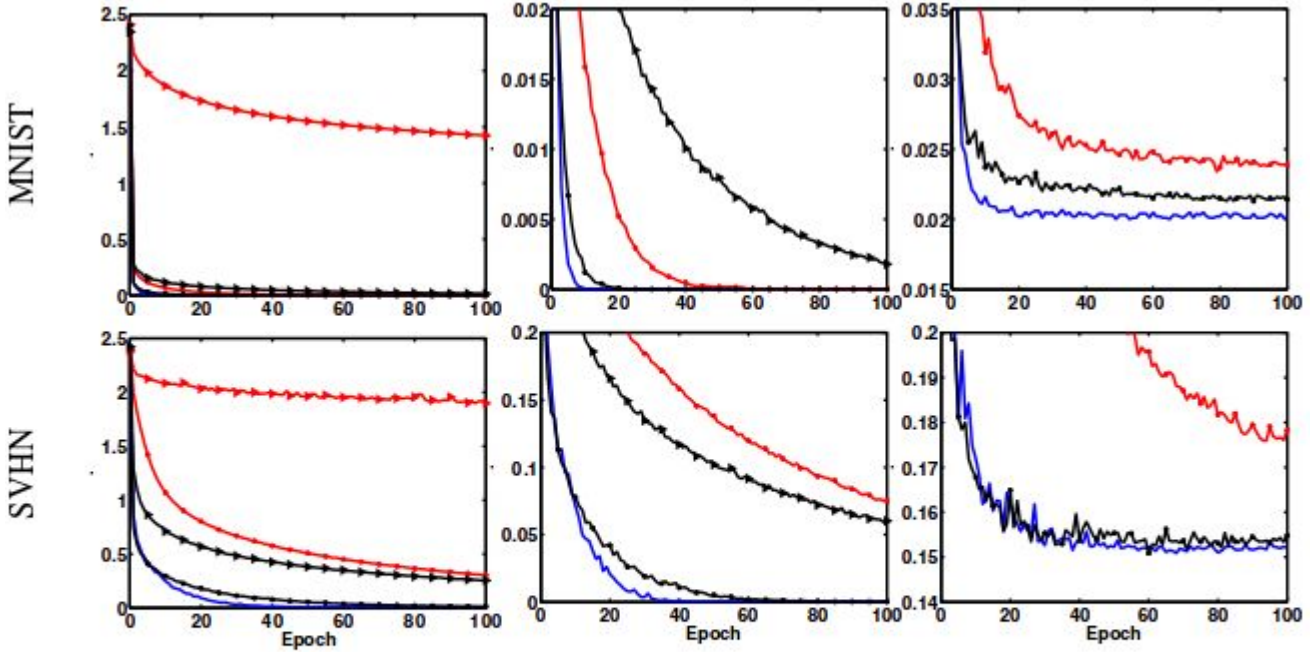
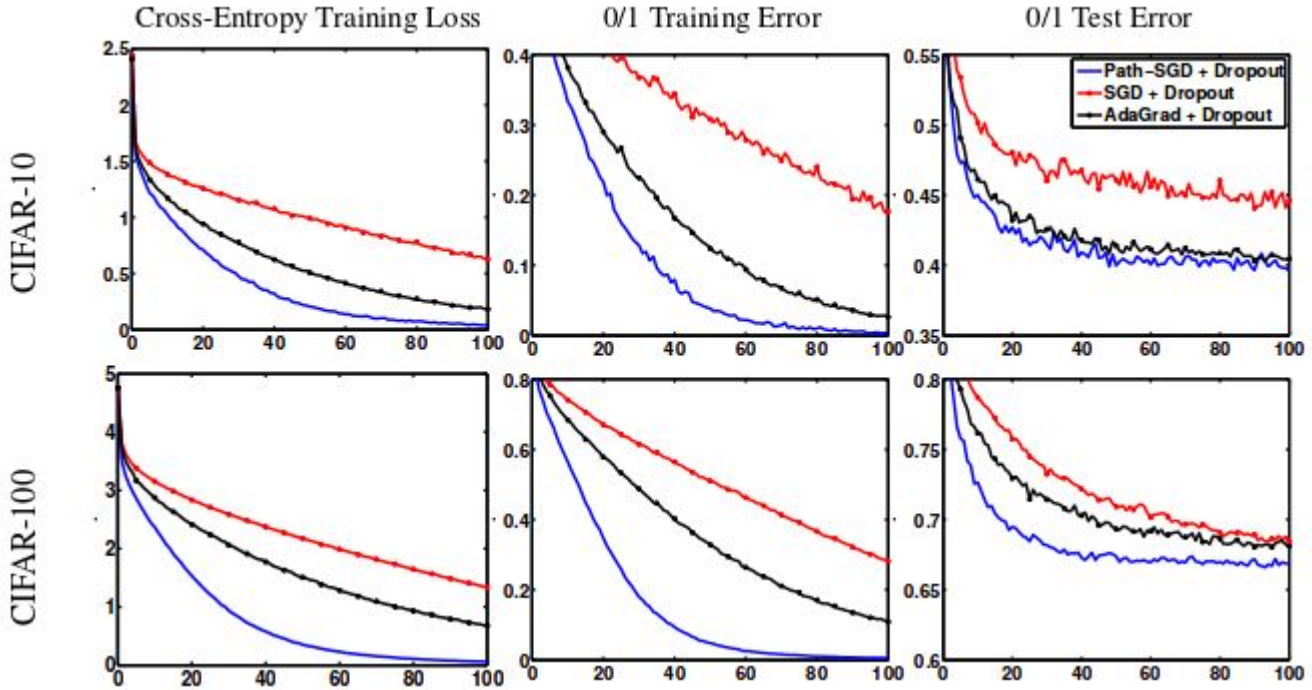


Figure 2: Learning curves using different optimization methods for 4 datasets without dropout. Left panel displays the cross-entropy objective function; middle and right panels show the corresponding values of the training and test errors, where the values are reported on different epochs during the course of optimization. Best viewed in color.

Path-SGD: Path Normalized Optimization in Deep Neural Networks



Path-SGD: Path Normalized Optimization in Deep Neural Networks

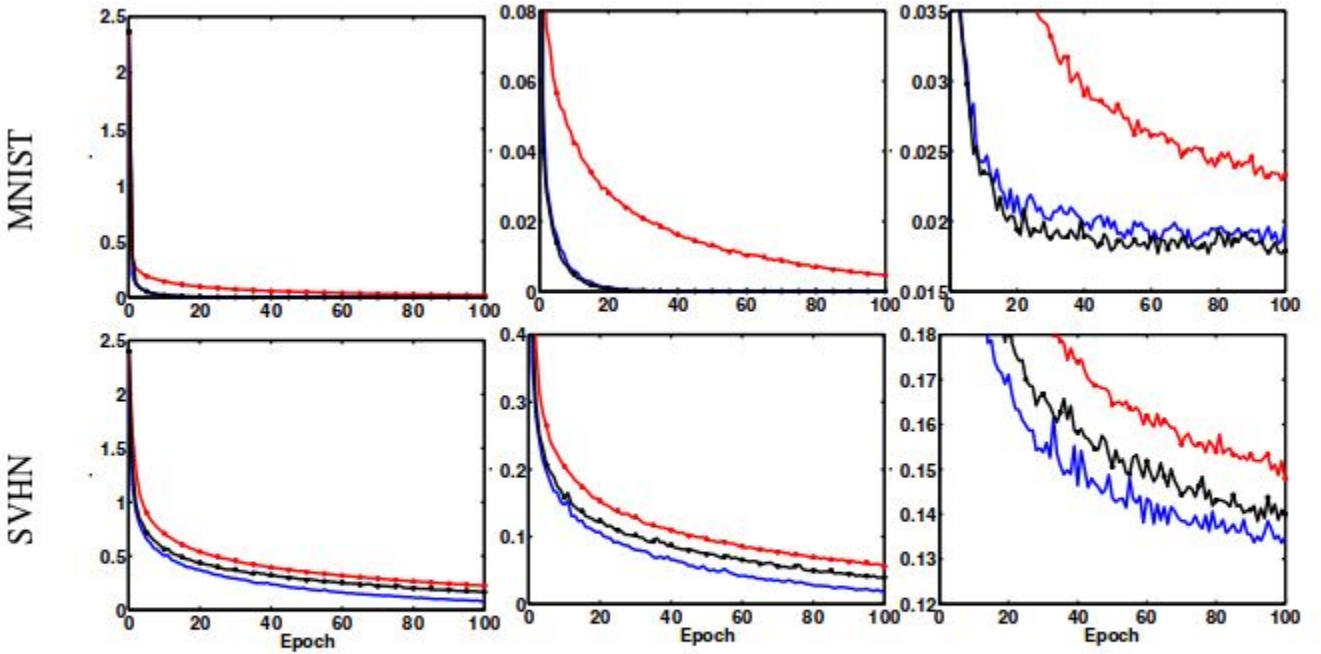


Figure 3: Learning curves using different optimization methods for 4 datasets with dropout. Left panel displays the cross-entropy objective function; middle and right panels show the corresponding values of the training and test errors. Best viewed in color.

Path-SGD: Path Normalized Optimization in Deep Neural Networks

---

- Conclusion: Path-SGD allows effective handling of balanced and unbalanced RELU networks, and can potentially be combined with AdaGrad update for even better results
  - Path-SGD can be viewed as tractable approximation of natural gradient
- Future Work: Could consider other geometries, as well as alternatives to steepest descent

References

- https://papers.nips.cc/paper/5797-path-sgd-path-normalized-optimization-in-deep-neural-networks.pdf
- https://papers.nips.cc/paper/5664-hessian-free-optimization-for-learning-deep-multidimensional-recurrent-neural-networks.pdf