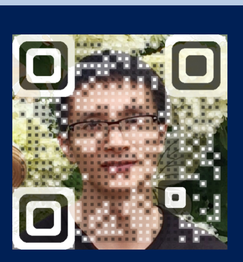
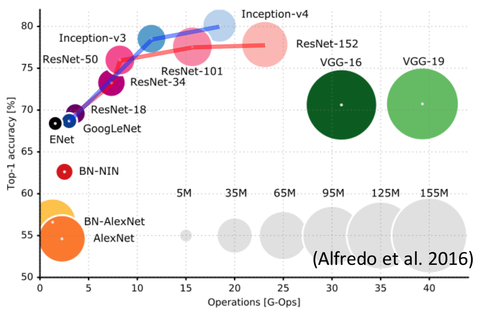
Efficient and Scalable Deep Learning

Wei Wen, Yiran Chen, Hai (Helen) Li, Duke University
{wei.wen, yiran.chen, hai.li}@duke.edu

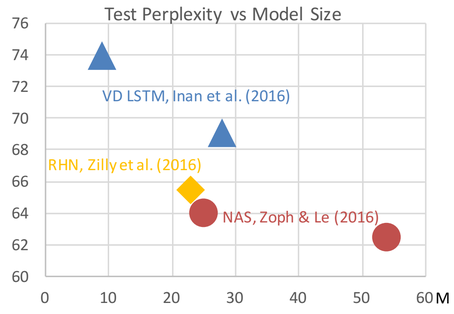


Efficient Deep Learning for Faster Inference on the Edge and in the Cloud

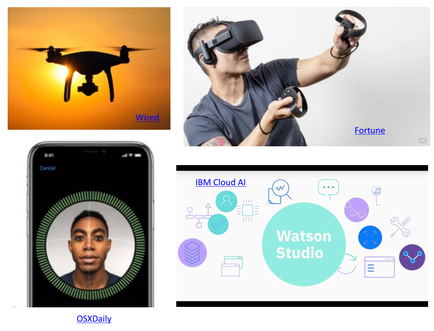
Image Classification on ImageNet

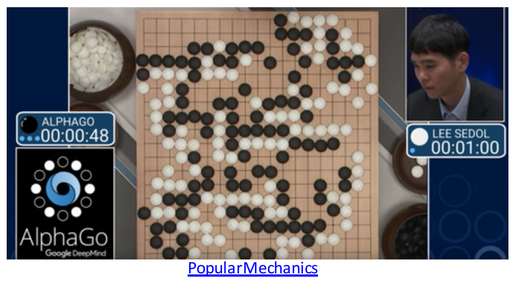


Language Modeling on Penn Treebank



More accurate prediction by paying computation cost using larger models.





PopularMechanics

A machine: 48 CPUs and 8 GPUs; a distributed version: 1,202 CPUs and 176 GPUs (Silver et al. 2016)

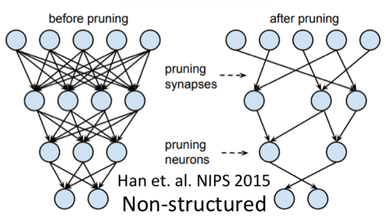
We must compress and accelerate deep learning models.

Inference on the edge:
- Limited computing capability
- Limited memory
- Limited battery energy

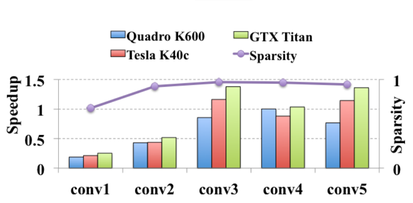
Inference in the cloud:
- A challenge: Real-time response to trillions of AI service requests
- Facebook (K. Hazelwood@ISCA'18)
  - 200+ Trillion predictions per day
  - 5+ Billion language translations per day

One of Our solutions: Structurally Sparse Deep Neural Networks (SSDNNs)

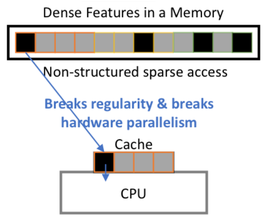
Sparsity Pattern



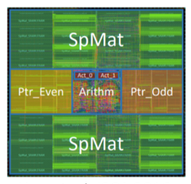
Speed



Locality

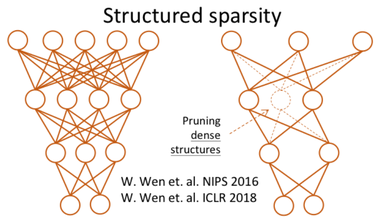


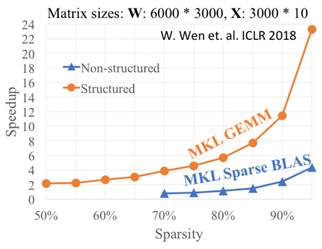
Deployment

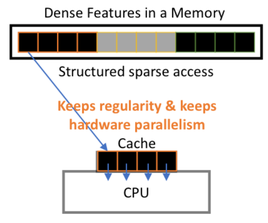


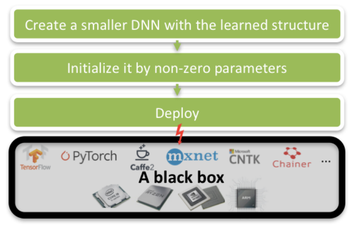
Han et. al. ISCA 2016

Structured sparsity









Group Lasso Regularization is All you Need for SSDNNs

Step 1: Weights are split to *G* groups $\mathbf{w}^{(1..G)}$

e.g. (w_1, w_2, w_3, w_4, w_5) -> group 1: (w_1, w_2, w_3), group 2: (w_4, w_5)

Step 2: Add group Lasso on each group $\mathbf{w}^{(g)}$ $||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$ i.e. vector length

sqrt($w_1^2 + w_2^2 + w_3^2$), sqrt($w_4^2 + w_5^2$)

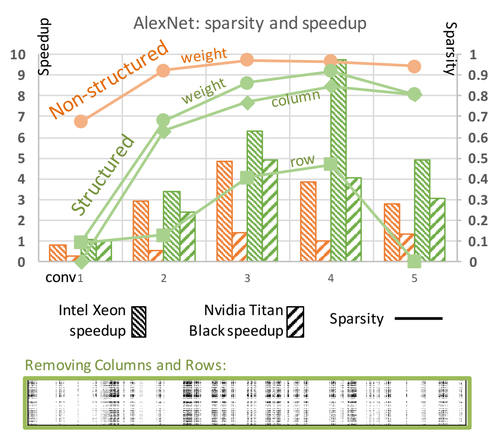
Step 3: Sum group Lasso over all groups as a regularization: $R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g$,

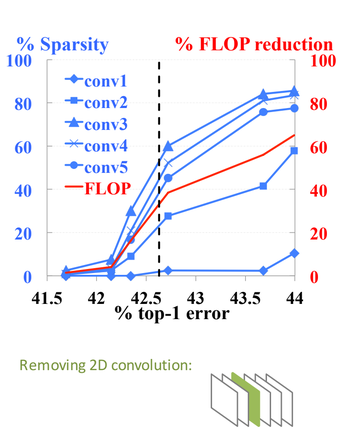
sqrt($w_1^2 + w_2^2 + w_3^2$) + sqrt($w_4^2 + w_5^2$)

Step 4: SGD optimizing $\arg\min_{\mathbf{w}}\{E(\mathbf{w})\} = \arg\min_{\mathbf{w}}\{E_D(\mathbf{w}) + \lambda_g \cdot R_g(\mathbf{w})\}$

We refer to our method as ***Structured Sparsity Learning (SSL)***

Structurally Sparse *AlexNet*

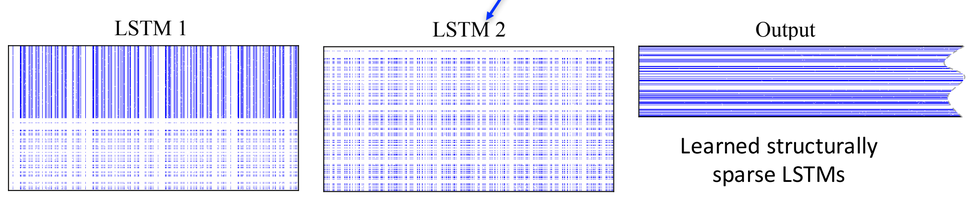




Structurally Sparse *LSTMs*

Method	Dropout keep ratio	Perplexity (validate, test)	ISS # in (1st , 2nd) LSTM	Weight #	Total time*	Speedup	Mult-add reduction†
baseline	0.35	(82.57, 78.57)	(1500, 1500)	66.0M	157.0ms	1.00×	1.00×
ours ISS	0.60	(82.59, 78.65)	(373, 315)	21.8M	14.82ms	10.59×	7.48×
		(80.24, 76.03)	(381, 535)	25.2M	22.11ms	7.10×	5.01×
direct design	0.55	(90.31, 85.66)	(373, 315)	21.8M	14.82ms	10.59×	7.48×

* Measured with 10 batch size and 30 unrolled steps.
† The reduction of multiplication-add operations in matrix multiplication. Defined as (original Mult-add)/(left Mult-add)



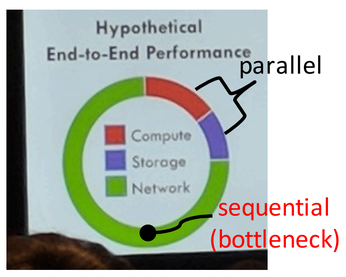
Learned structurally sparse LSTMs

SSL is able to remove: filters, channels, neurons, layers, matrix dimensions, hidden states, weight blocks, etc.

Applied to: intel AI Ref. Ref. Bing Ref.

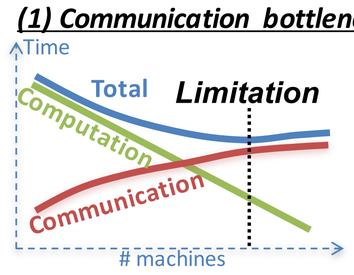
Our works on Efficient Deep Learning: NIPS 2016, ICLR 2018, ICCV 2017, ICLR 2017, CVPR 2017, ASP-DAC 2017 (Best Paper Award), DAC 2015 & 2016 (Two Best Paper Nominations).

Scalable Deep Learning for Faster Training in Distributed Systems

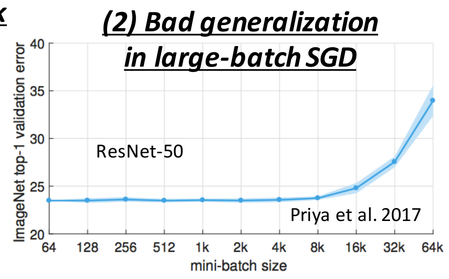


(K. Hazelwood@ISCA'18) Picture: leiphone

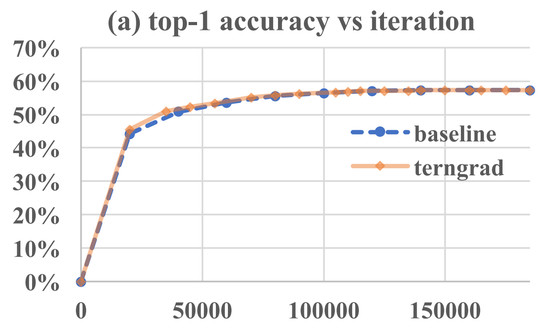
(1) Communication bottleneck

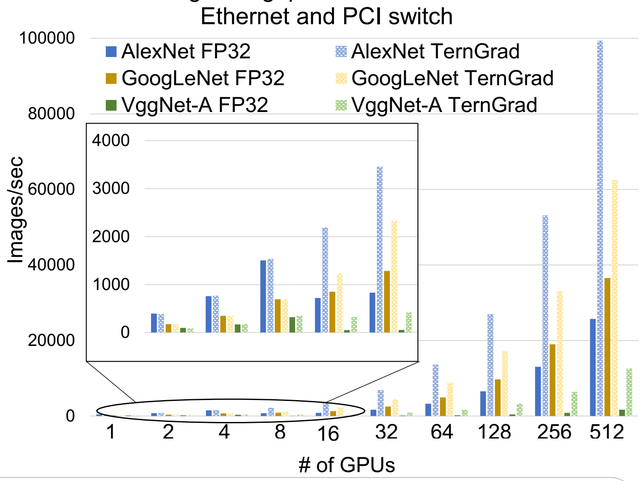


(2) Bad generalization in large-batch SGD



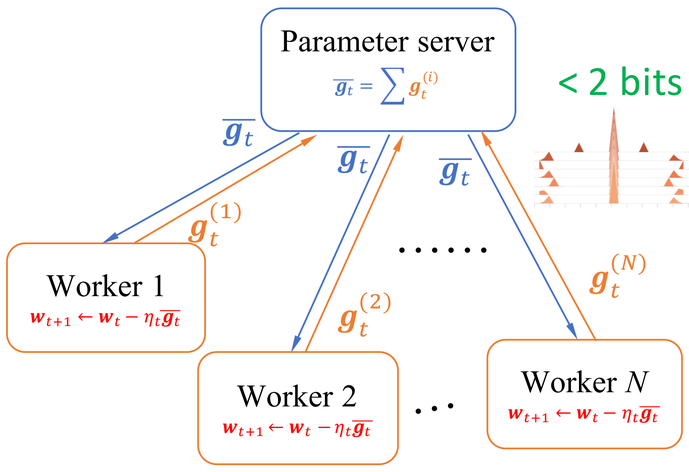
TernGrad Summary:
- ✓ No loss in AlexNet;
- ✓ <2% loss in GoogLeNet





Solution to (1)

TernGrad: quantizing gradients to reduce communication (NIPS 2017, Oral)



< 2 bits (>16x reduction)

ternary gradients (TernGrad)

An Example:
$\boldsymbol{g}_t^{(i)}$: [0.30, -1.20, …, 0.9]
s_t: 1.20
Signs: [1, -1, …, 1]
$P(b_{tk} = 1|g_t)$: $[\frac{0.3}{1.2}, \frac{1.2}{1.2}, \ldots, \frac{0.9}{1.2}]$
$\boldsymbol{b}_t$: [0,1,…,1]
$\widetilde{\boldsymbol{g}_t^{(i)}}$: [0, -1, …, 1]*1.20

$$\tilde{\boldsymbol{g}}_t = ternarize(\boldsymbol{g}_t) = s_t \cdot sign(\boldsymbol{g}_t) \circ \boldsymbol{b}_t$$

$$s_t \triangleq ||\boldsymbol{g}_t||_\infty \triangleq max(abs(\boldsymbol{g}_t))$$

$$\begin{cases} P(b_{tk} = 1 \mid \boldsymbol{g}_t) = |g_{tk}|/s_t \\ P(b_{tk} = 0 \mid \boldsymbol{g}_t) = 1 - |g_{tk}|/s_t \end{cases}$$

(Convergence guaranteed)

TernGrad is now in PyTorch/Caffe2 and Adopted in Facebook AI Infra.

Solution to (2)

W. Wen, Y. Wang, F. Yan, C. Xu, Y. Chen, H. Li, "***SmoothOut:*** Smoothing Out Sharp Minima to Improve Generalization in Deep Learning", AAAI 2018 Submission.