Making Deep Learning Revolution Practical Through Second Order Methods

Michael W. Mahoney

ICSI and Department of Statistics
University of California at Berkeley

Joint work with Amir Gholami, Zhewei Yao, and many others to be mentioned.

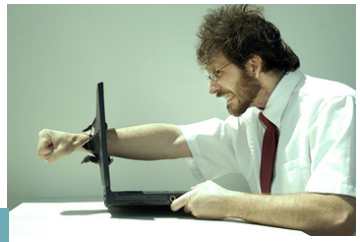
Training hard neural network models?

(slide 1 of 3 from 2017 “Second-order Machine Learning” talk on stochastic second order *theory*)

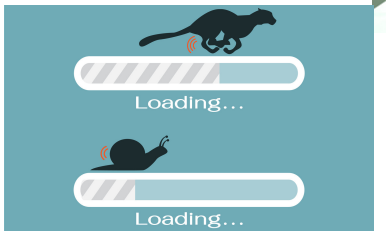
- Are you training difficult Machine Learning model?

- Is you current optimization algorithm slow?

- Do you feel like punching the monitor?



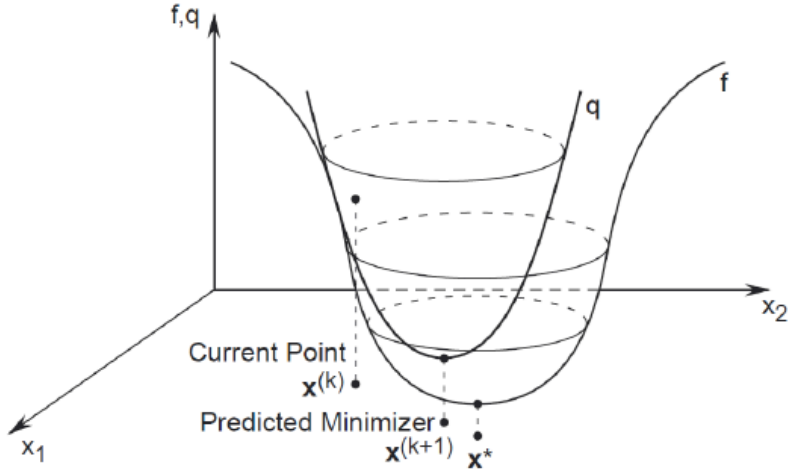
- Do you need a fast solver?



Under the Hood: Iterative Optimization

(slide 2 of 3 from 2017 "Second-order Machine Learning" talk on stochastic second order *theory*)

$$x^{(k+1)} = \arg\min_{x \in \mathcal{D} \cap \mathcal{X}} \left\{ F(\mathbf{x}^{(k)}) + (\mathbf{x} - \mathbf{x}^{(k)})^T \mathbf{g}(\mathbf{x}^{(k)}) + \frac{1}{2\alpha_k} (\mathbf{x} - \mathbf{x}^{(k)})^T H(\mathbf{x}^{(k)}) (\mathbf{x} - \mathbf{x}^{(k)}) \right\}$$



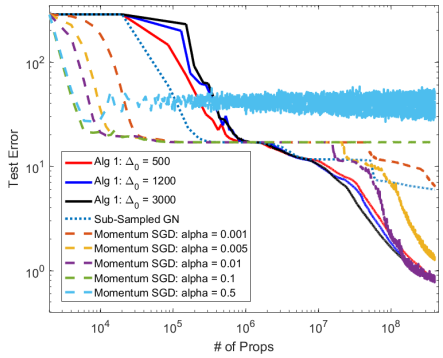
Typical approaches:

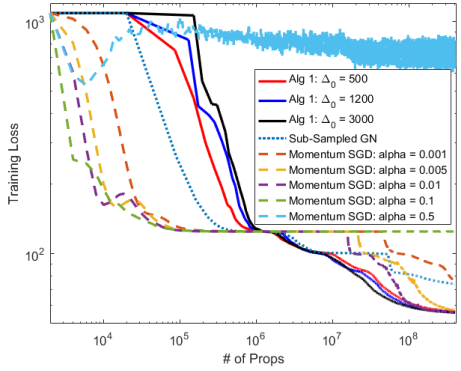
- "first order": use just $\mathbf{g}(\mathbf{x}^{(k)})$; fast, but many "knobs" to fiddle with
- "second order": use both $\mathbf{g}(\mathbf{x}^{(k)})$ and $H(\mathbf{x}^{(k)})$; slower (especially if poor implementations), but few "knobs" to fiddle with

Examples of improvements

(slide 3 of 3 from 2017 “Second-order Machine Learning” talk on stochastic second order *theory*)

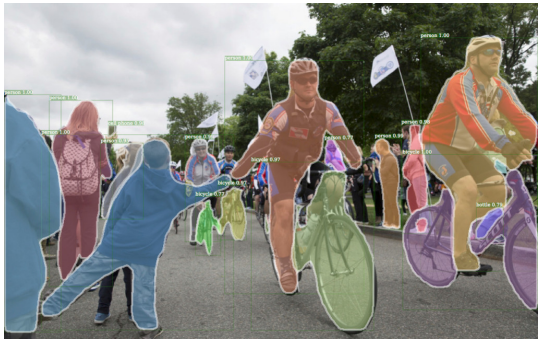
- resiliency to problem ill-conditioning
- good generalization error and robustness to hyper-parameter tuning

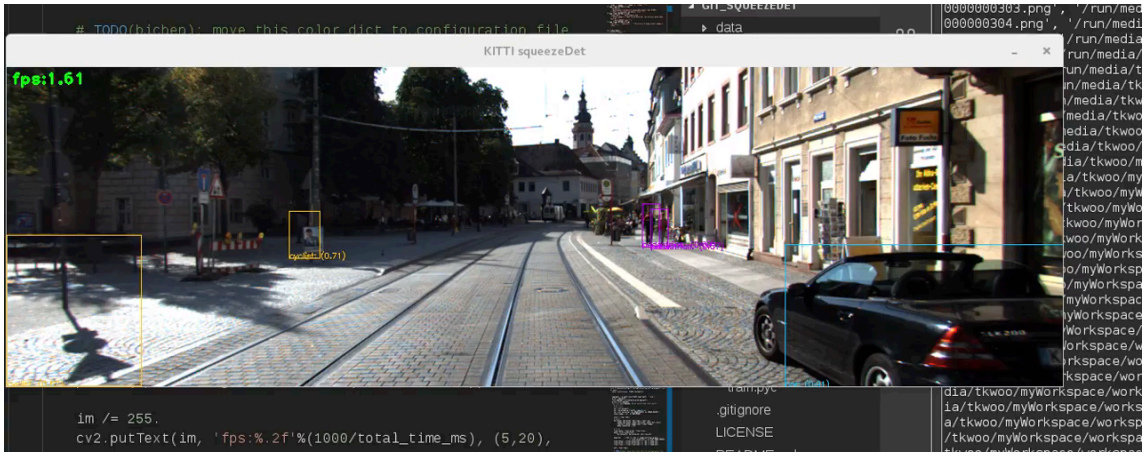




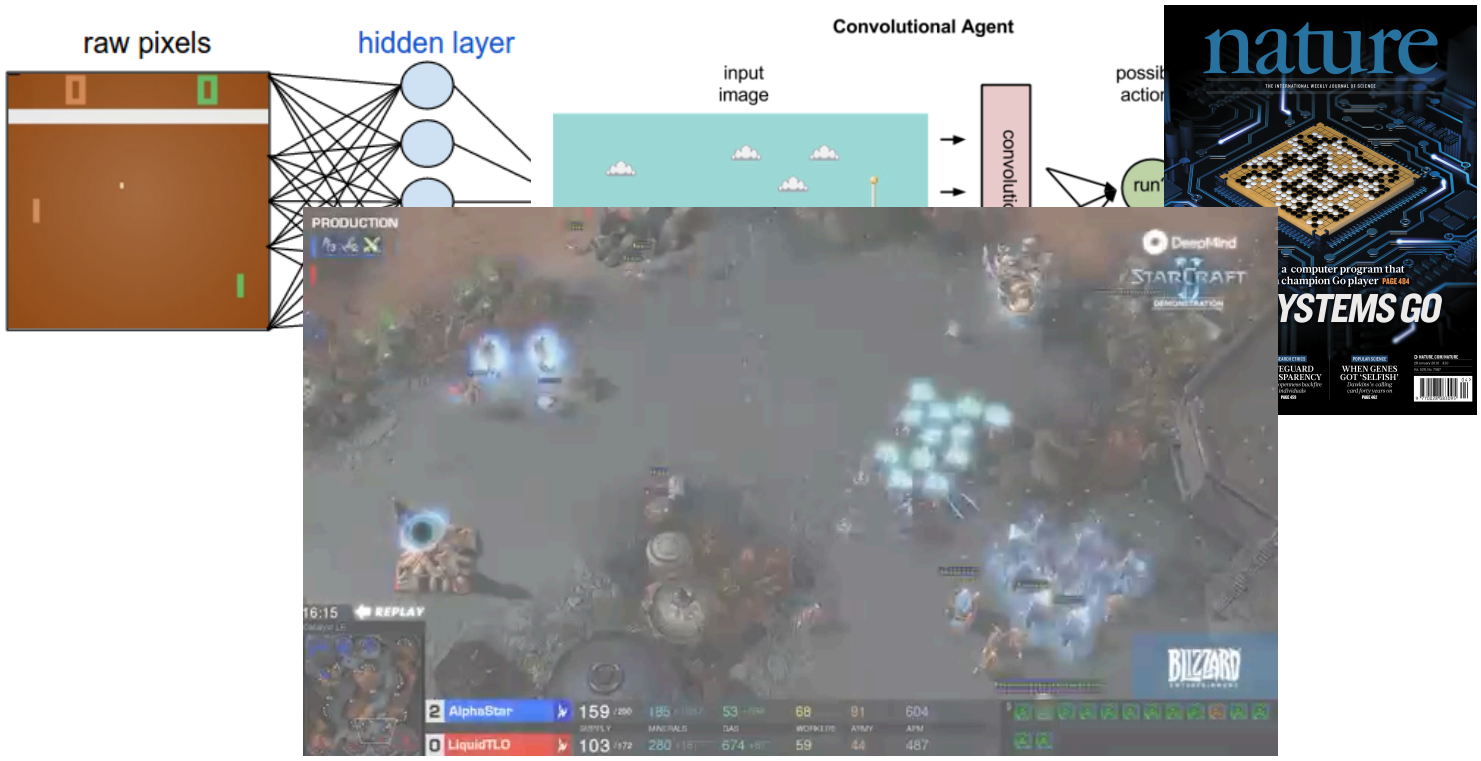
- ability to escape undesirable saddle-points
- low-communication costs in distributed settings
- computational advantages offered by leveraging the power of GPUs

Deep Learning Revolution





Deep Learning Revolution



But …

Remarkable results, but:

- Successfully applying methods for a *slightly* different problem: more of a **dark art** than science/engineering
- Lots of **“tricks”** that do not port
- **Extensive** hyper-parameter tuning that is **expensive**
- **Ad-hoc rules** that do not generalize
- **Quasi-random search** methods that do not scale

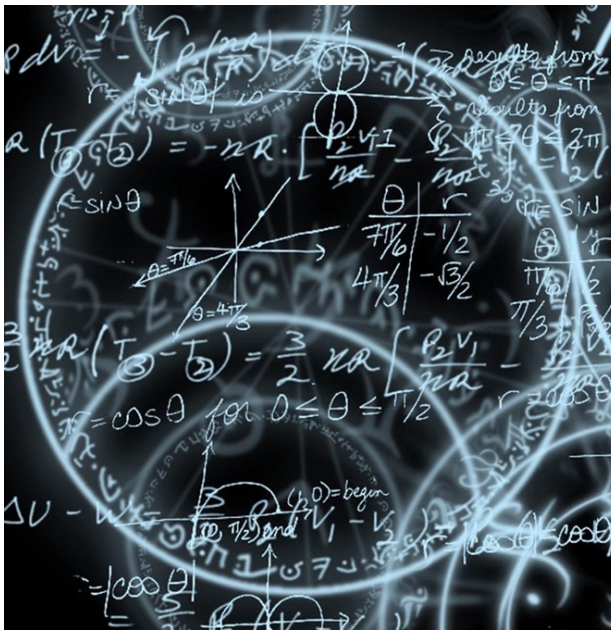


Image from here

One solution: from industry

Put a massive investment into computational power:

- State-of-the-art models are trained with AutoML requiring hundreds of thousands of GPU hours
  - A single training of RoBERTa takes **10 months** on a 100K DGX system
  - Training an NLP model can emit pounds of carbon dioxide equivalent to nearly **five times the lifetime emissions of the average American car**
  - Training AlphaStar had a cost of at about **ge 700M USD**
- ***The existing m.o. is not scalable. It is not robust. It is not reproducible.***

An Example

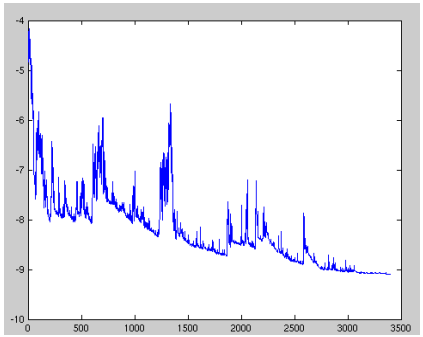
Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**

Is this enough number of steps?

Why not 1M steps?

Liu, Yinhan, et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." arXiv preprint arXiv:1907.11692 (2019).

Stochastic Gradient Descent (SGD)



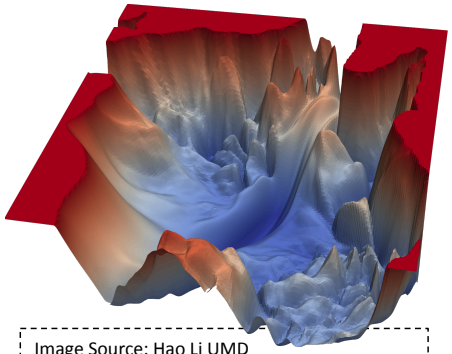


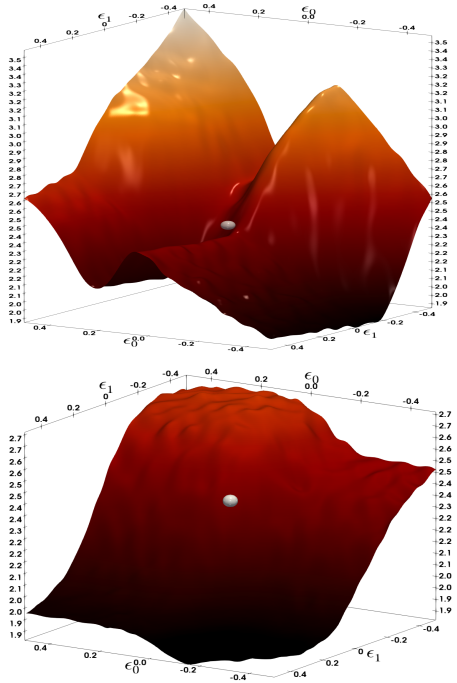
Image Source: Hao Li UMD

$$\text{Assume } f(W^t, x) = \frac{1}{n}\sum_{i=1}^{n} f_i(W^t, x)$$

$$W^{t+1} \leftarrow W^t - \alpha \cdot \nabla_W f_i(W^t, x)$$

$$W^{t+1} \leftarrow W^t - \alpha \cdot \frac{1}{b}\sum_{i=k+1}^{k+b} \nabla_W f_i(W^t, x)$$

Mini-batch: compute gradient using b samples

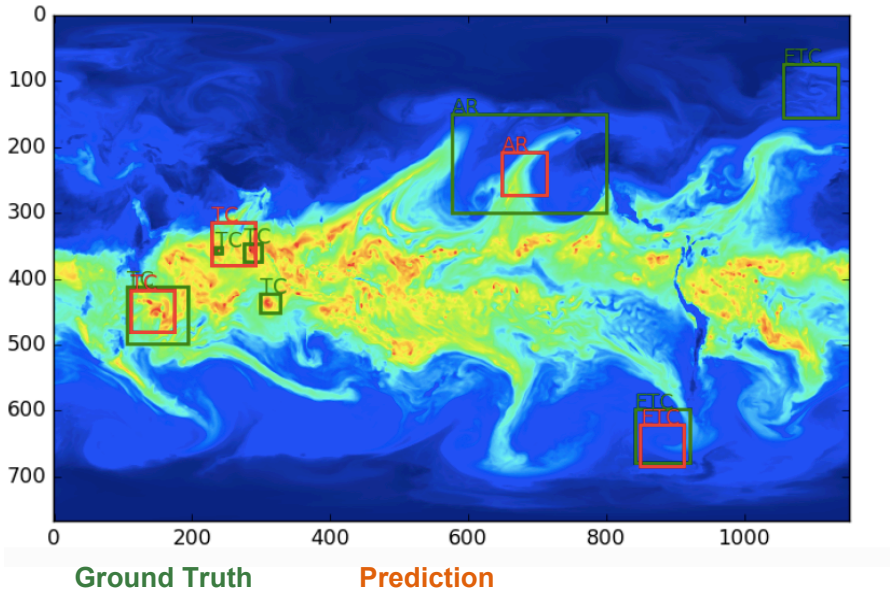


BERT model trained on SQuAD:
Clearly the model has NOT converged

Devlin, Jacob, et al. "BERT: Pre-training of deep bidirectional transformers for language understanding."*arXiv:1810.04805* (2018).
S. Sheng, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, K. Keutzer, "Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT" arXiv:1909.05840 (2019).

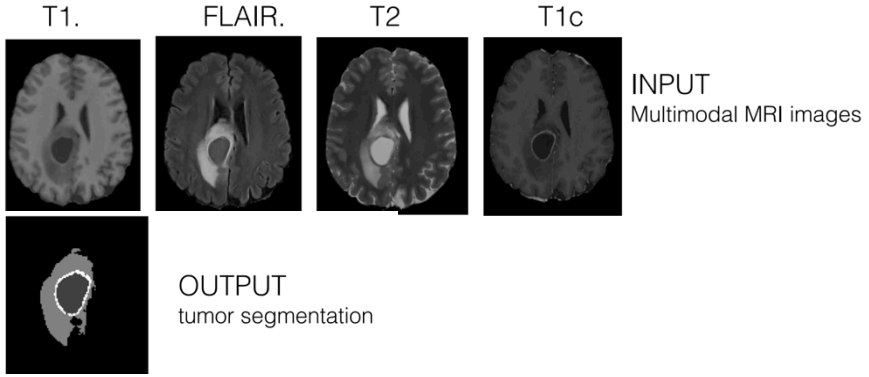
There are other types of applications ...

Finding Extreme Weather in Climate Data



Prabhat, et al., NERSC/LBNL (2018).

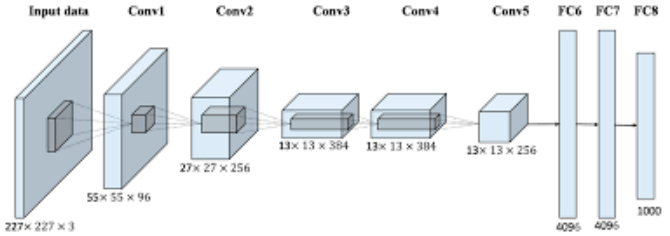
Tumor Identification



A. Mang, S. Tharakan A. Gholami, N. Himthani, S. Subramanian, J. Levitt, M. Azmat, K. Scheufele, M. Mehl, C. Davatzikos, B. Barth, and G. Biros. SIBIA-GIS: Scalable biophysics-based image analysis for glioma segmentation. The multimodal brain tumor image segmentation benchmark (BRATS), MICCAI, 2017

- ***The existing m.o. is not scalable. It is not robust. It is not reproducible.***

Using “computational mathematics” for ML/DNNs?



“Deep Learning” has two major tasks:

- “Training” of the Deep Neural Network
  - Faster training: compute/communication intensive process; extensive hyperparameter tuning and architecture search; takes days, weeks, …
- “Inference” (or deployment/prediction) that uses a trained DNN
  - Trained models are large; space constraints, etc. are bottlenecks

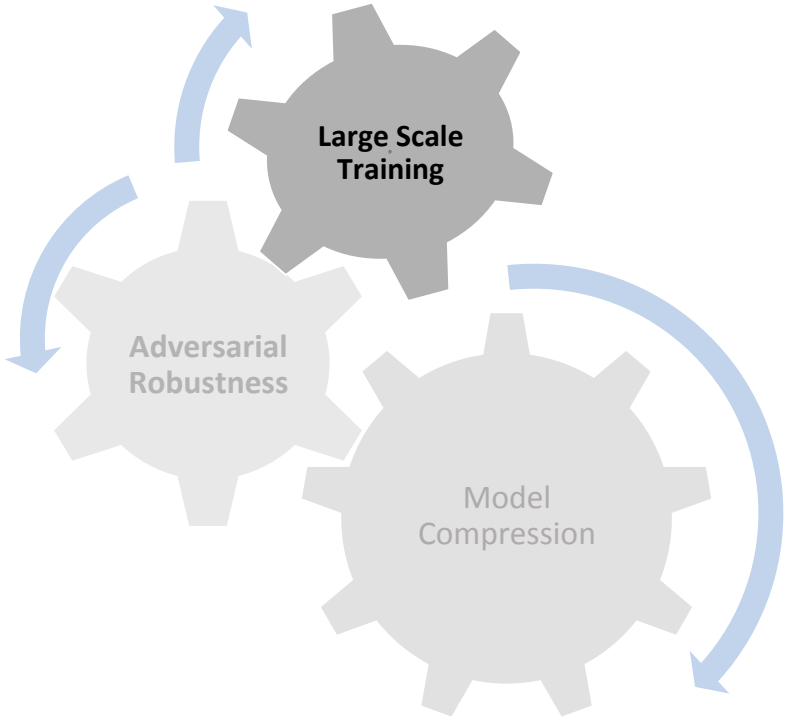
Faster training can be achieved by

- Newer/faster Hardware (but there is a limit!)
- Can we use more GPUs or nodes?
- Parallel and Distributed Training
- ***Understanding the Data …***

Holistic Evaluation is Important!!

- *My framework is faster than your framework!*
- Isolated view of performance (depends on entire execution environment) is not helpful
- ***Scientific computing ideas appear in unexpected places …***

Second Order Methods



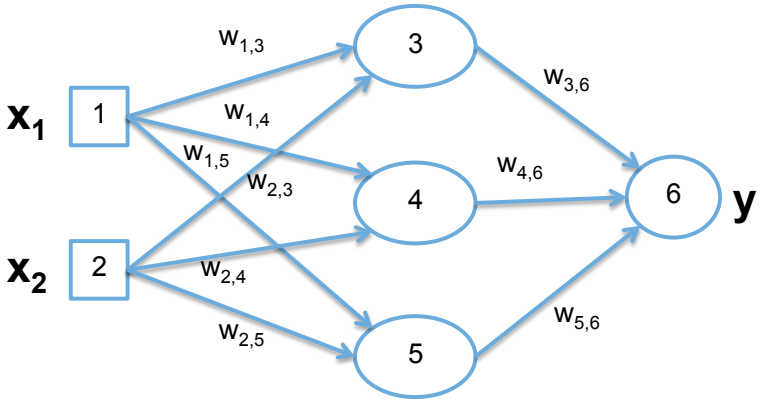
"Machine learning is high performance computing's first killer app for consumers" --- NVIDIA CEO 2015

Second-order methods (use Hessian info as well as gradient info) for:

- **Efficiency/inefficiency of training:** SGD, KFAC, and other 2nd order methods
- **Adversarial examples:** smoothing out ML objectives, using 2nd order methods
- **Quantizing large models:** using outlier metrics derived from 2nd order methods

Training Neural Networks

Training adjusts the weights (W) in the connections of the neural network, in order to change the function it represents.



Only parameters are weights for simplicity (i.e., ignore bias parameters)

W: the matrix of weights

A "shallow" neural network with only one hidden layer (nodes 3,4,5), two inputs and one output.

The size of both NNs and datasets

Image classification:

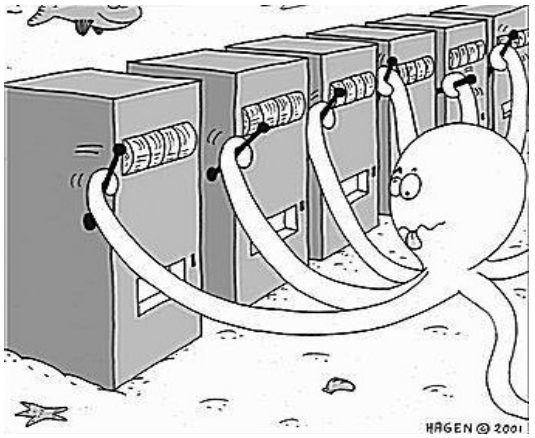
- Model size: 20M parameters (ResNet50)
- Dataset size: 1.2M images (224x224x3)
- Training time: 3 days on one V100 GPU

Natural Language Processing:

- Model size: 110M parameters (BERT-base)
- Dataset size: 50M sentences
- Training time: 14 days on eight V100 GPUs

Many Many Knobs!

- SGD is *very* sensitive to hyper-parameters
  - In particular batch size (size of stochastic mini-batch)
- Batch size is inter-dependent with:
  - Degradation in accuracy
  - Poor generalizability
  - Robustness of model
  - Training time
  - Parallel Scalability



$$W^{t+1} = W^t - \alpha \sum_{i=0}^{B} \nabla_W f_i(W^t, x)$$

The Desire for Large-scale Distributed Training

- For most of computations,

  We have reached hardware limits

- With abundance of large training data for many applications,

  Next milestone is fast training on large datasets

- The first opportunity is to use data parallel method

  Need/want **large batch** for efficient scaling

$$W^{t+1} = W^t - \alpha \sum_{i=0}^{B} \nabla_W f_i(W^t, x)$$

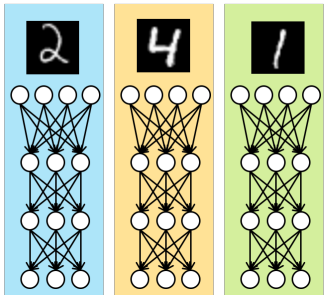
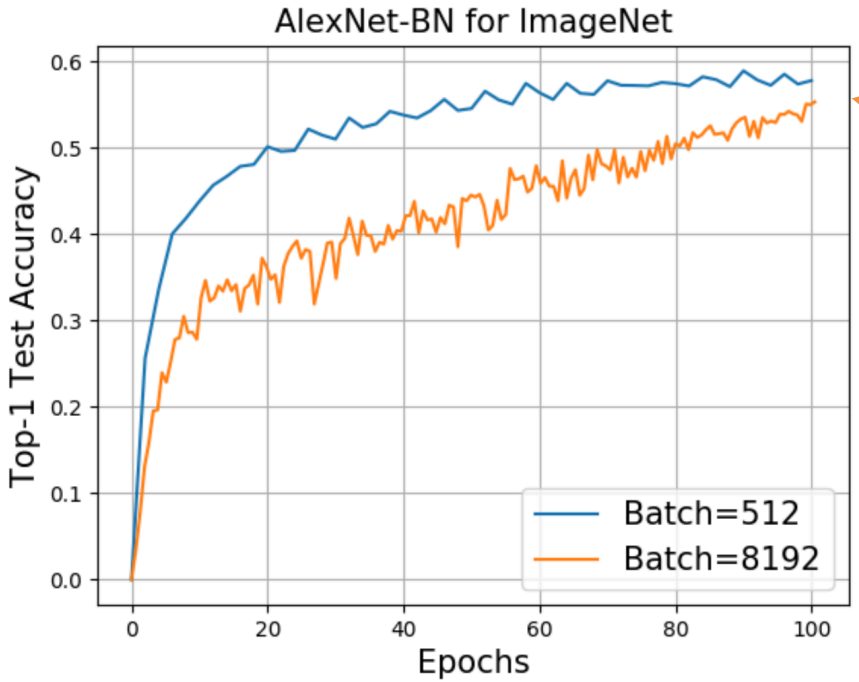


Image from http://chainermn.readthedocs.io/en/v1.0.0b2_a/tutorial/overview.html

Problems with Large Batch: Degradation in Accuracy

Larger Batch often leads to degradation in accuracy



Performance Gap

Current large-batch training methods:

- **Degrade accuracy**
- Lead to **poor robustness** to adversarial inputs
- Existing "solutions" either do not work or require **extensive hyper-parameter tuning**

Ginsburg, Boris, Igor Gitman, and Yang You, "Large Batch Training of Convolutional Networks with Layer-wise Adaptive Rate Scaling." arxiv:1708.03888.
C. H. Martin and M. W. Mahoney,,"Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning," arXiv:1810.01075.

Training budget selection

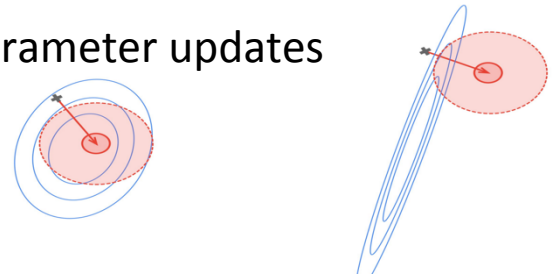
Different assumptions exist in literature about what is expensive:

Limiting factor is **Time**	**Limiting factor is** **Money/Computation**
° Example: **Owners** of computing resources, e.g. Google ° Easier hyperparameter tuning parallelization ° **Stop training based on number of updates: "iterations"**	° Example: **Renters** of computing resources, e.g. AWS users ° Paying for each hyperparameter tried ° **Stop training based on number of epochs or training examples**

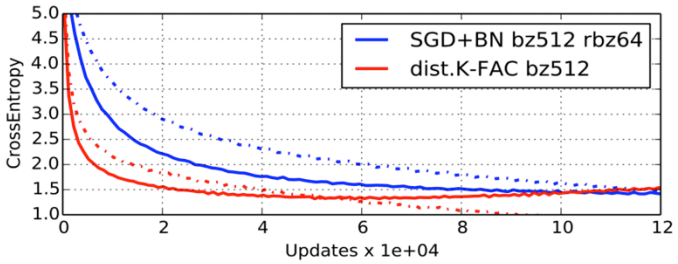
K-FAC: A *very* approximate second order method

Kronecker-Factored Approximate Curvature (K-FAC)

- Uses *heavily-approximated and heavily-parameterized second-order information* to compute parameter updates
- Problem it roughly solves:



- Comparable performance to SGD
- Better for large batch training?



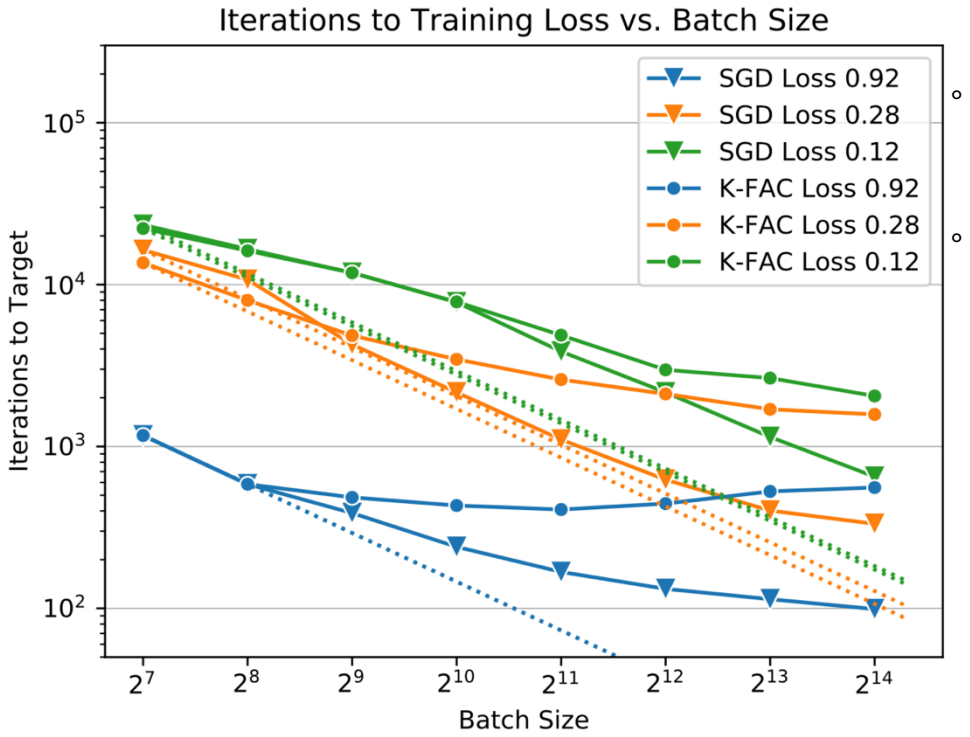
https://supercomputersfordl2017.github.io/Presentations/K-FAC.pdf
James Martens and Roger B. Grosse. "Optimizing Neural Networks with Kronecker-factored Approximate Curvature". In: *CoRR* abs/1503.05671 (2015)
Jimmy Ba, Roger Grosse, and James Martens. "Distributed second-order optimization using Kronecker- factored approximations". In: *ICLR* 2017.

A level-playing-field performance comparison

Hyperparameters of **SGD**	Hyperparameters of **K-FAC**
° **Learning rate** ° **Momentum** ° Weight decay	° **Learning rate** ° **Damping** ° Momentum ° Weight decay ° Clipping magnitude ° Second-order update frequency ° Second-order update momentum

- No relationships assumed between hyper-parameters
- We tune two hyper-parameters equally, giving the same amount of training to both methods

A level-playing-field performance comparison

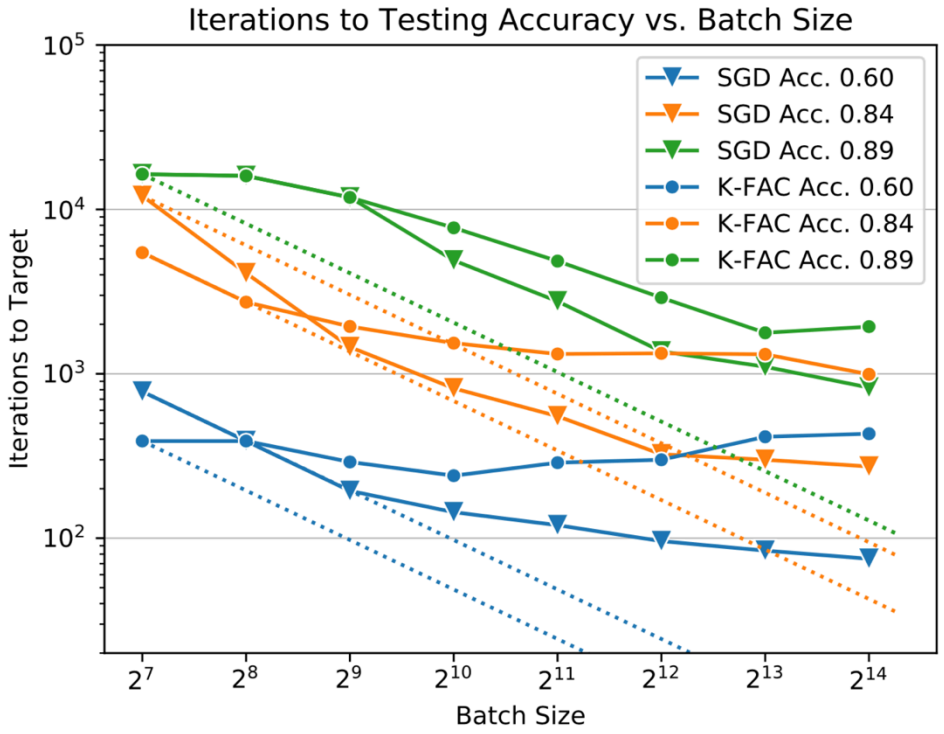


- We race SGD and K-FAC to 3 target ***losses***
- **Dotted: Ideal hyperbola** (linear because log-log)

iterations-to-target ~ speed

N. Golmant, N. Vemuri, Z. Yao, V. Feinberg, A. Gholami, K. Rothauge, M. W. Mahoney, and J. Gonzalez, "On the Computational Inefficiency of Large Batch Sizes for Stochastic Gradient Descent," arXiv:1811.12941 (2018).
L. Ma, G. Montague, J. Ye, Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, "Inefficiency of K-FAC for Large Batch Size Training," arXiv:1903.06237 (2019)

A level-playing-field performance comparison



- We race SGD and K-FAC to 3 target ***accuracies***
- **Dotted: Ideal hyperbola** (linear because log-log)

N. Golmant, N. Vemuri, Z. Yao, V. Feinberg, A. Gholami, K. Rothauge, M. W. Mahoney, and J. Gonzalez, "On the Computational Inefficiency of Large Batch Sizes for Stochastic Gradient Descent," arXiv:1811.12941 (2018).
L. Ma, G. Montague, J. Ye, Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, "Inefficiency of K-FAC for Large Batch Size Training," arXiv:1903.06237 (2019)

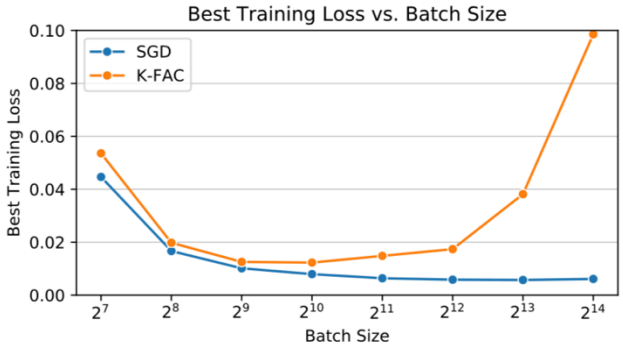
A level-playing-field performance comparison

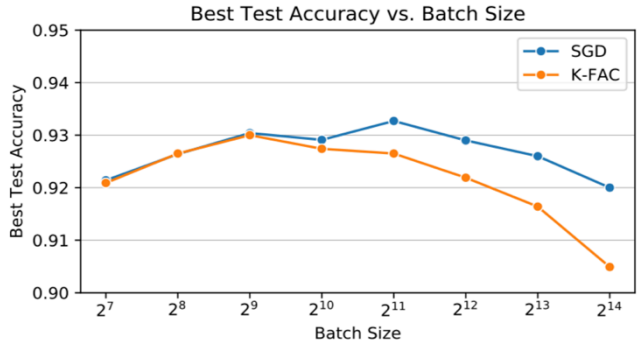
- **Consistently slower = consistently worse end-of training performance**

  (regardless of stopping rule)

- Our stopping rule:
  - Total training epochs = (log2(batch size/128) + 1) × 100

- End-of-training (SGD vs. K-FAC):

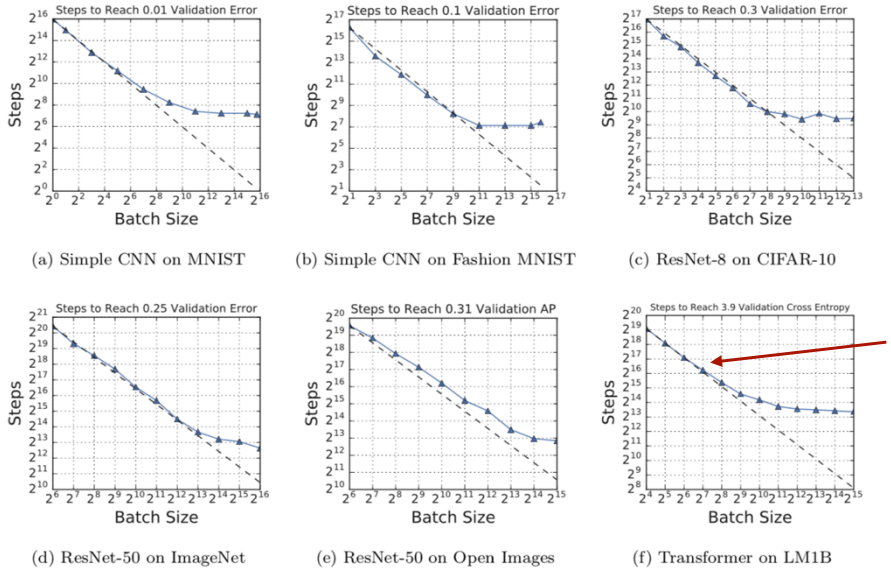




N. Golmant, N. Vemuri, Z. Yao, V. Feinberg, A. Gholami, K. Rothauge, M. W. Mahoney, and J. Gonzalez, "On the Computational Inefficiency of Large Batch Sizes for Stochastic Gradient Descent," arXiv:1811.12941 (2018).
L. Ma, G. Montague, J. Ye, Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, "Inefficiency of K-FAC for Large Batch Size Training," arXiv:1903.06237 (2019)

Empirical studies of large batch SGD

- **Best performance after training depends on:**
  - **Batch size**
  - **Traditional hyper-parameters**
  - **Stopping rule**
- **Google's approach: try *all* combinations of above, compare performance**



(a) Simple CNN on MNIST (b) Simple CNN on Fashion MNIST (c) ResNet-8 on CIFAR-10

(d) ResNet-50 on ImageNet (e) ResNet-50 on Open Images (f) Transformer on LM1B

For batch sizes outside linear scaling regime, we are wasting money for performance that is nearly the same

"Linear scaling regime"

An aside:

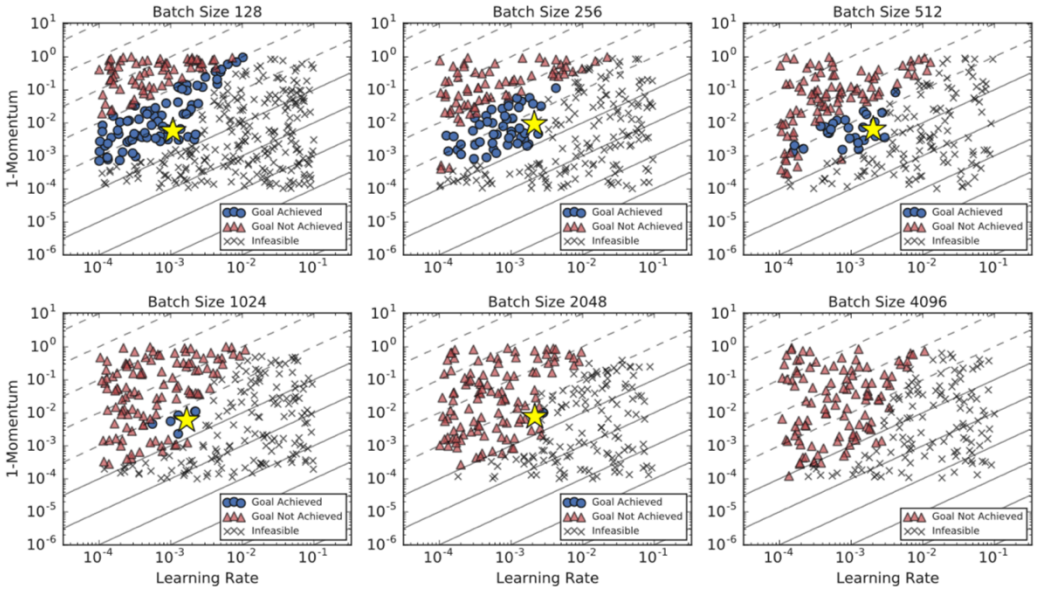
- Shallue et al cited *prior* work: Anonymous (2018). "**On the computational inefficiency ...**" In: under review, ICLR19.
- That paper is: N Golmant, N Vemuri, Z Yao, V Feinberg, A Gholami, K Rothauge, M W Mahoney, and J Gonzalez. "**On the computational inefficiency ...**" arXiv:1811.12941 (2018).
- Golmant et al *(who did careful experimental design*) had similar conclusions to Shallue et al. (*who tried all possibie combinations*)
- Shallue et al. now have a paragraph saying that Golmant et al. is inconclusive, *basically since latter couldn't try all combinations of hyperparameters*.

C. J. Shallue, J. Lee, J. Antognini, J.Sohl-Dickstein, R. Frostig, and G. E. Dahl. "Measuring the Effects of Data Parallelism on Neural Network Training". In: arXiv:1811.03600 (2018).

How do we measure hyperparameter robustness?

1. If limiting factor is money/computation (epochs):

Google SGD study: for SGD, *robustness* ***decreases*** with batch size *if training with epoch budget.*



Failure

Success

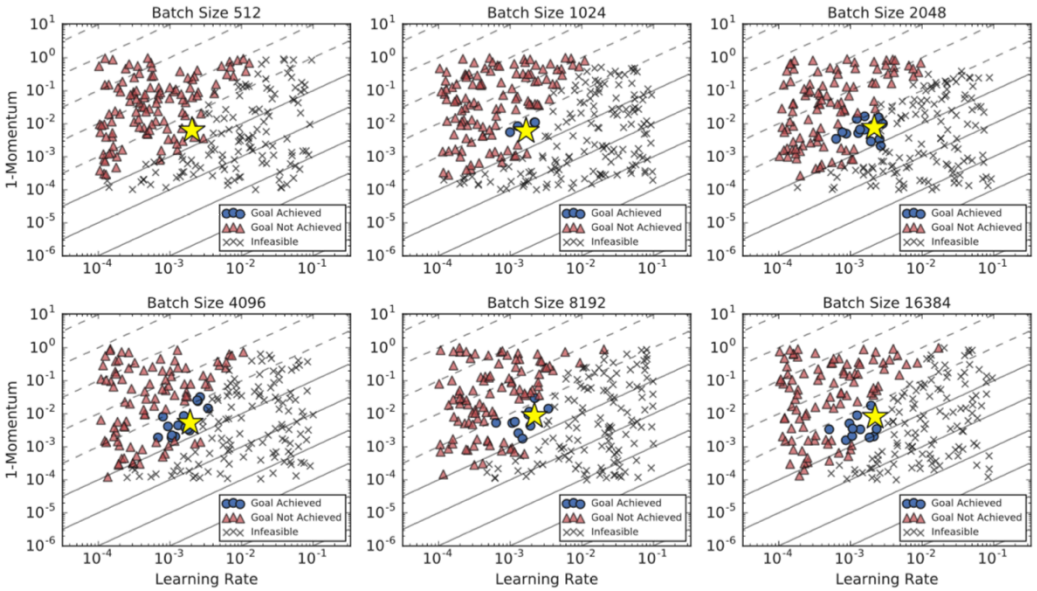
"Transformer on LM1B with a training budget of one epoch"

C. J. Shallue, J. Lee, J. Antognini, J.Sohl-Dickstein, R. Frostig, and G. E. Dahl. "Measuring the Effects of Data Parallelism on Neural Network Training". In: arXiv:1811.03600 (2018).

How do we measure hyperparameter robustness?

2. If limiting factor is time (iterations):

Google SGD study: for SGD, *robustness* ***increases*** with batch size, *if training with iteration budget*.



Failure

Success

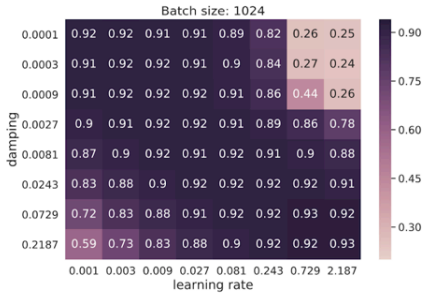
"Transformer on LM1B with a training budget of of 25,000 [iterations]"

C. J. Shallue, J. Lee, J. Antognini, J.Sohl-Dickstein, R. Frostig, and G. E. Dahl. "Measuring the Effects of Data Parallelism on Neural Network Training". In: arXiv:1811.03600 (2018).

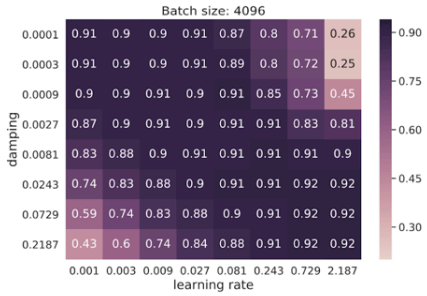
K-FAC Hyperparameter Robustness: Heatmaps

K-FAC Accuracy Heatmaps under our epoch-like budget:*

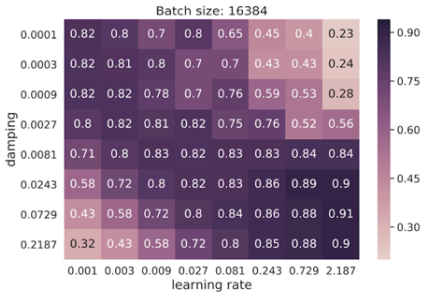
|B| = 1,024



|B| = 4,096



|B| = 16,384



- **Shrinking** of the high-accuracy region with increasing batch size
- (Coincidental: positive correlation observed between damping and learning rate)

*Total training epochs = $(\log_2(\text{batch size}/128) + 1) \times 100$

Less surprising:

- K-FAC would have to be doing something wrong not to benefit from extra progress that iteration budgets give to large batch sizes

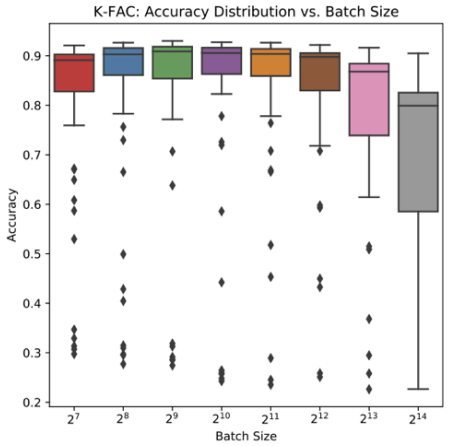
More surprising:

- K-FAC speedups inferior

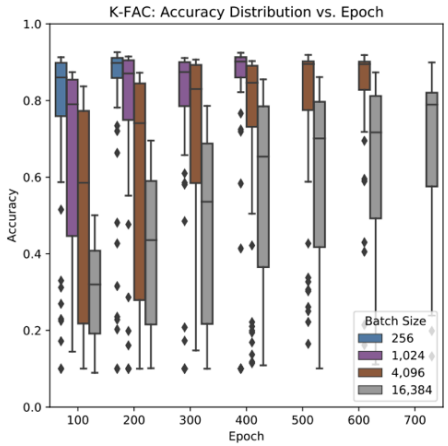
N. Golmant, N. Vemuri, Z. Yao, V. Feinberg, A. Gholami, K. Rothauge, M. W. Mahoney, and J. Gonzalez, "On the Computational Inefficiency of Large Batch Sizes for Stochastic Gradient Descent," arXiv:1811.12941 (2018).
L. Ma, G. Montague, J. Ye, Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, "Inefficiency of K-FAC for Large Batch Size Training," arXiv:1903.06237 (2019)

K-FAC Hyperparameter Robustness: Box Plots

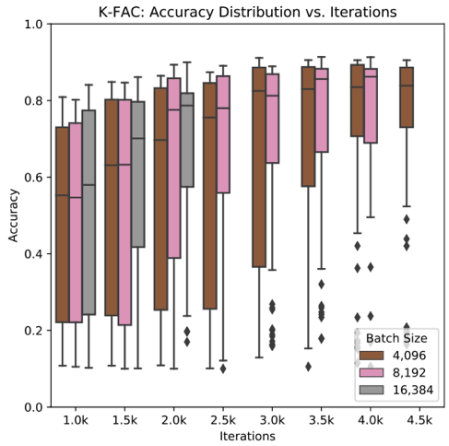
Accuracy vs. Batch Size



Accuracy vs. Epoch



Accuracy vs. Iterations



If epoch based budget:

- then **weaker** parameter robustness with increasing batch size.

If iteration based budget:

- then **stronger** parameter robustness with increasing batch size.

N. Golmant, N. Vemuri, Z. Yao, V. Feinberg, A. Gholami, K. Rothauge, M. W. Mahoney, and J. Gonzalez, "On the Computational Inefficiency of Large Batch Sizes for Stochastic Gradient Descent," arXiv:1811.12941 (2018).
L. Ma, G. Montague, J. Ye, Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, "Inefficiency of K-FAC for Large Batch Size Training," arXiv:1903.06237 (2019)

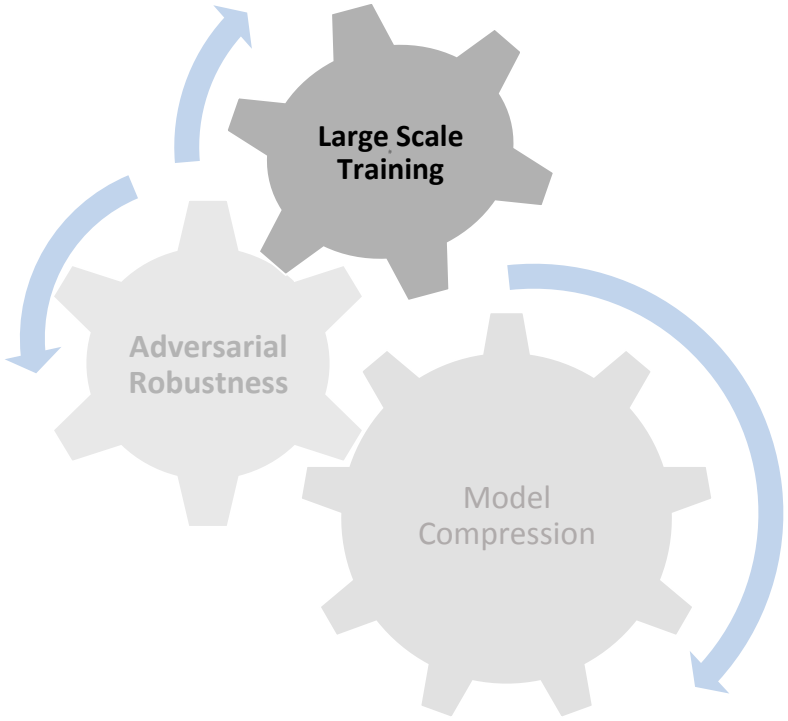
Summary (of Efficiency/Inefficiency of Large-Batch Training)

- Even with extensive hyperparameter tuning, K-FAC has **comparable**, not superior, train/test performance to SGD.
- Increasing batch size for K-FAC -> **lower** speedup, compared with SGD, when measured in terms of iterations.
- For fixed epochs, larger batch sizes -> **weaker** hyperparameter robustness.
- For fixed iterations, larger batch sizes -> **greater** hyperparameter robustness.

Common but unfair criticism: not a fair comparison---you did not fiddle with enough parameters long enough!

- Open question 1: What about "other" second order methods?
- Open question 2: Should we care?

Second Order Methods



"Machine learning is high performance computing's first killer app for consumers" --- NVIDIA CEO 2015

Second-order methods (use Hessian info as well as gradient info) for:

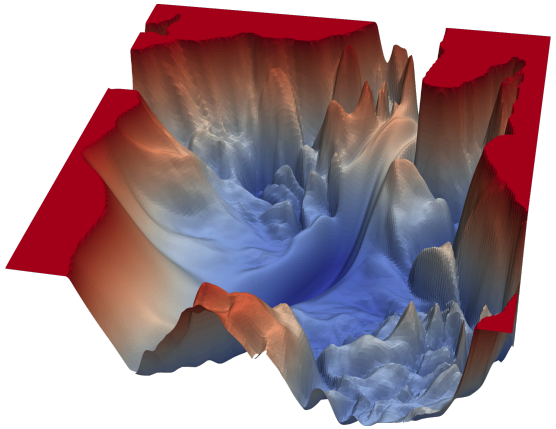
- **Efficiency/inefficiency of training:** SGD, KFAC, and other 2nd order methods
- **Adversarial examples:** smoothing out ML objectives, using 2nd order methods
- **Quantizing large models:** using outlier metrics derived from 2nd order methods

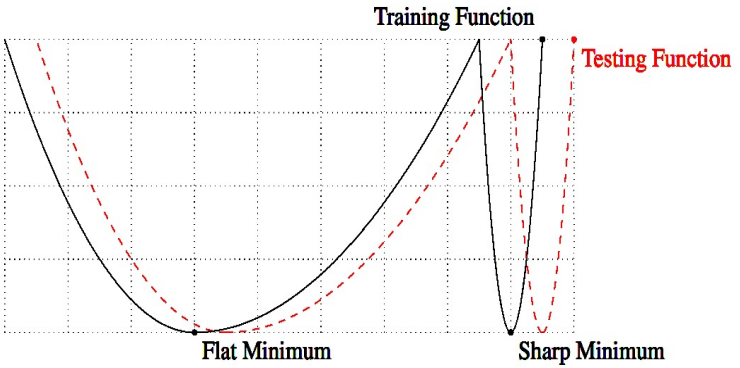
Poor Generalization: Sharp vs Flat Minima

° *Why* does large batch suffer from poor generalization performance?

- A common belief is that large batch training gets attracted to "sharp minimas"
- Another theory is that large batch may get stuck in saddle points

° A somewhat misleading picture (left) and a very misleading picture (right):





Analysis through Hessian Lens!

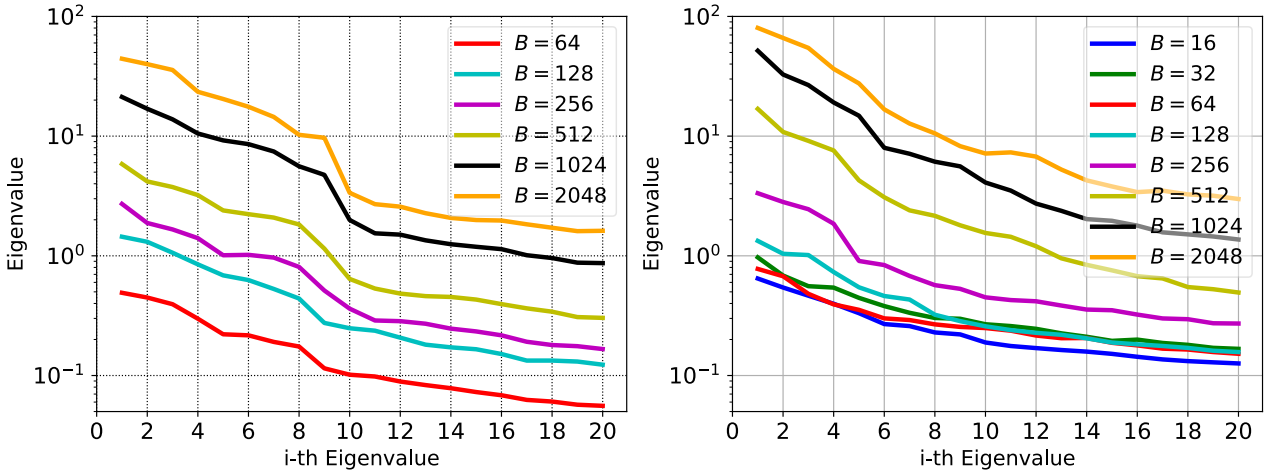
- We back-propagate the Hessian operator (second derivative) and compute its spectrum during training along with total gradient

- The Hessian spectrum is computed on all the training/testing examples using power method

- We visualize the landscape of loss along dominant eigenvectors of the Hessian

Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. W. Mahoney. Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18

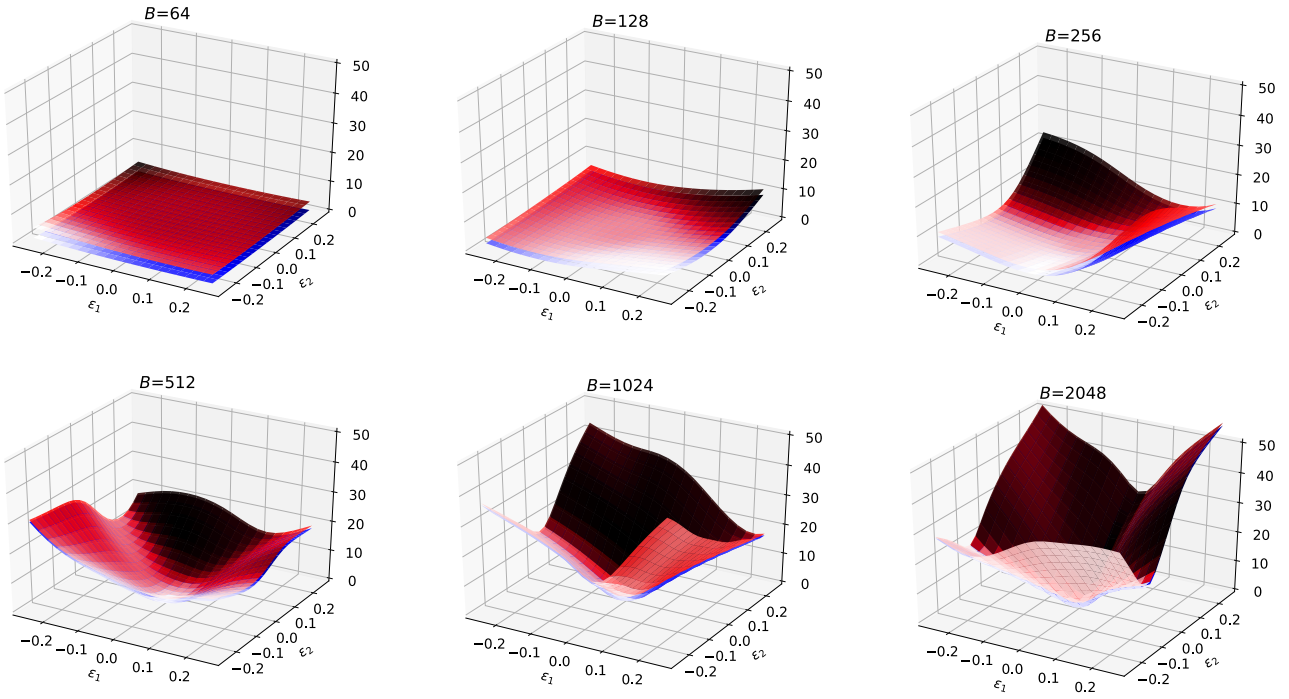
Large Batch Size Training (via the Hessian)



Top 20 eigenvalues of the total Hessian w.r.t. model parameters for different batch size. Larger batch size converges to points with higher Hessian spectrum

Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. W. Mahoney. Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18

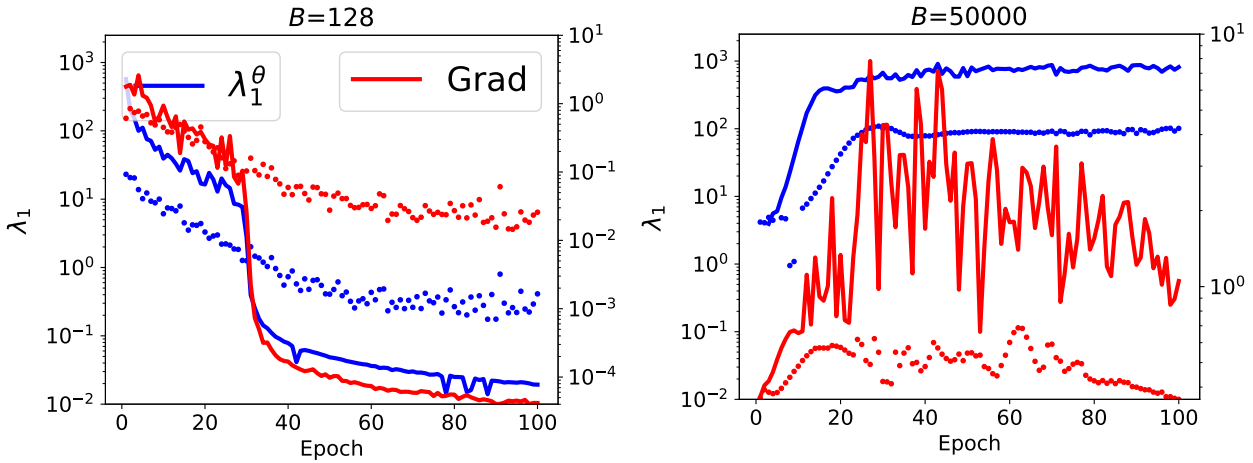
Large Batch Through the Lens of the Hessian



Training/testing loss at the end of training along the dominant eigenvector of the Hessian (for Cifar-10 dataset)

Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. W. Mahoney. Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18

Hessian Spectrum During Training



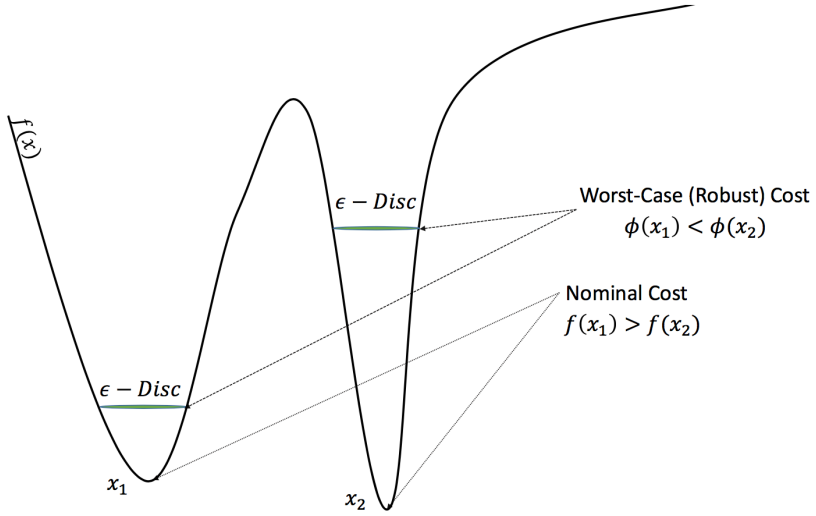
Changes in the dominant eigenvalue of the Hessian w.r.t. weights and the total gradient for different epochs during training.

- Large batch gets attracted to areas with larger Hessian spectrum (blue curve)
- "Robust optimization" leads to smoother surfaces (dotted lines)

Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. W. Mahoney. Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18

What is Robust Optimization?

- Instead of minimizing for the average case, consider the worst case under “some metric”



Keskar, Nitish Shirish, et al. "On large-batch training for deep learning: Generalization gap and sharp minima." ICLR’16 (arXiv:1609.04836).

Robust Optimization and Regularization/Stability

Robust solution to least squares with bounded uncertainty in A is equivalent to lasso regularized least squares:

$$\min_x \max_{\|\Delta A\|_{\infty,2} \leq \rho} \|(A + \Delta A)x - b\| \qquad \min_x \|Ax - b\| + \lambda \|x\|_1$$

El Ghaoui, Laurent, and Hervé Lebret. "Robust solutions to least-squares problems with uncertain data." SIAM Journal on matrix analysis and applications 18.4 (1997): 1035-1064.
Huan Xu, Constantine Caramanis, and Shie Mannor. Robust regression and lasso. In *Advances in Neural Information Processing Systems*, pages 1801–1808, 2009.

- Consider a simple linear programming problem

$$\min_x \{c^T x : Ax \leq b\}$$

- Assume input data (A,b,c) is uncertain (uncertainty set U)

$$\min_x \sup_{(A,b,c) \in \mathcal{U}} \{c^T x : Ax \leq b\}$$

- Model for neural network robustness

Shaham U, Yamada Y, Negahban S. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. arXiv:1511.05432. 2015 Nov 17.

Robust Optimization in DNNs

Find the NN's parameters using a min-max optimization, instead of just minimization

- Solving the max problem is computationally infeasible
- A practical solution is to perform **gradient ascent** to iteratively solve the max problem and then **gradient descent** for the min part
- I.e., implement an adversarial training rule

$$\min_{\theta} \tilde{J}(\theta, x, y) = \min_{\theta} \sum_{i=1}^{m} \max_{\tilde{x}_i \in \mathcal{U}_i} J(\theta, \tilde{x}_i, y_i)$$

Problems with DNN: Non-robustness of Models

(a) Strawberry (b) Toy poodle

(c) Buckeye (d) Toy poodle

[Chaowei Xiao, Bo Li, Jun-yan Zhu, Warren He, Mingyan Liu, Dawn Song, 2017]

Of interest by itself

(Bad if a stop sign is adversarially changed to a yield sign)

We use it to *implement* "robust training"

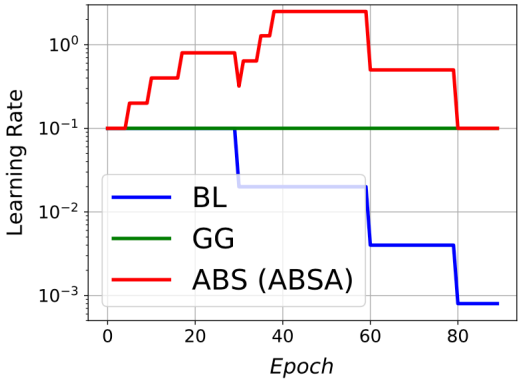
Adaptive Batch Size Schedules Using Hessian Information

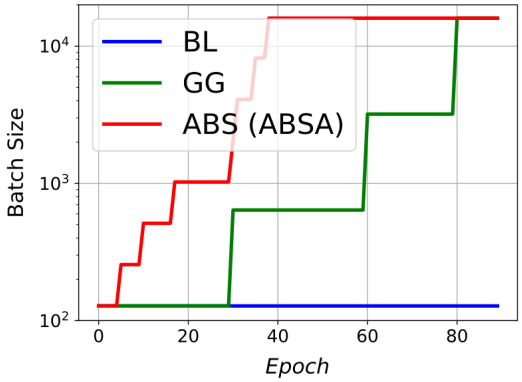
Adaptive Batch Size (ABS) schedule:

- *adaptively* uses Hessian information

Adaptive Batch Size Adversarial (ABSA) schedule:

- *adaptively* uses Hessian information and Adversarial/Robust Training





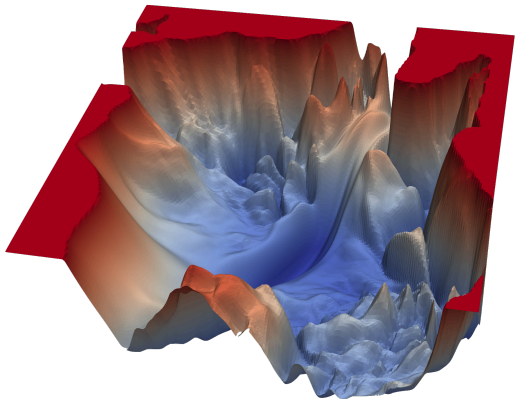


Illustration of learning rate (a) and batch size (b) schedules of adaptive batch size as a function of training epochs based on C2 model on Cifar-10

Z. Yao, A. Gholami, K. Keutzer, M. Mahoney, "Large Batch Size Training of Neural Networks with Adversarial Training and Second-Order Information"
Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

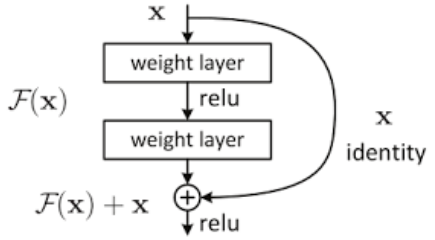
Cifar-10

- **Cifar-10 has ten classes**
  - **~5000 examples per class**
  - **Total 50,000 training images**
  - **10,000 testing images**

Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

ResNet-18 on Cifar-10

° Our proposed method (ABSA) achieves better performance



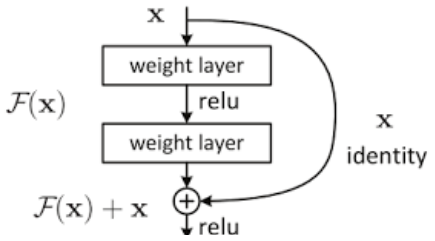
ResNet-18 on Cifar-10

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	83.05	35156	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
640	81.01	7031	**84.59**	7031	83.99	16380	83.30	10578	84.52	9631
3200	74.54	1406	78.70	1406	84.27	14508	83.33	6375	**84.42**	5168
5120	70.64	878	74.65	878	83.47	14449	83.83	6575	**85.01**	6265
10240	68.75	439	30.99	439	83.68	14400	83.56	5709	**84.29**	7491
16000	67.88	281	10.00	281	84.00	14383	83.50	5739	**84.24**	5357

FB: Goyal, Priya, et al. "Accurate, large minibatch SGD: training ImageNet in 1 hour." arXiv preprint arXiv:1706.02677 (2017).
GG: S. Samuel L., P. Kindermans, and Q. V. Le. "Don't Decay the Learning Rate, Increase the Batch Size." ICLR 2017.
ABS/ABSA: Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

WideResNet18 on Cifar10

° Our proposed method (ABSA) achieves better performance

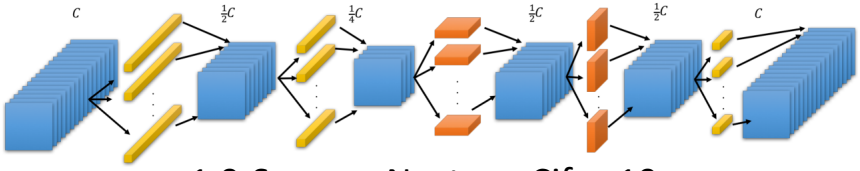


WResNet18-2 on Cifar-10

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	87.64	35156	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
640	86.20	7031	87.9	7031	87.84	16380	87.86	10399	**89.05**	10245
3200	82.59	1406	73.2	1406	87.59	14508	88.02	5869	**89.04**	4525
5120	81.40	878	63.27	878	87.85	14449	87.92	7479	**88.64**	5863
10240	79.85	439	10.00	439	87.52	14400	87.84	5619	**89.03**	3929
16000	81.06	281	10.00	281	88.28	14383	87.58	9321	**89.19**	4610

Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

SqueezeNext on Cifar-10

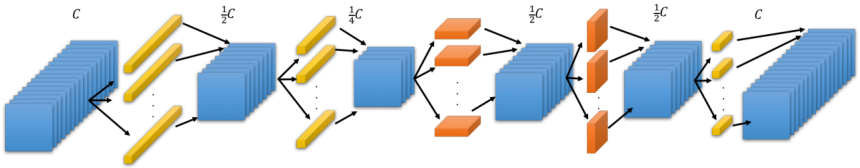


1.0-SqueezeNext on Cifar-10

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	92.02	78125	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
256	91.88	39062	91.75	39062	91.84	50700	91.7	40792	**92.11**	43352
512	91.68	19531	91.67	19531	91.19	37050	**92.15**	32428	91.61	25388
1024	89.44	9766	91.23	9766	91.12	31980	91.61	17046	**91.66**	23446
2048	83.17	4882	90.44	4882	89.19	30030	91.57	21579	**91.61**	14027
4096	73.74	2441	86.12	2441	91.83	29191	91.91	18293	**92.07**	21909
8192	63.71	1220	64.91	1220	91.51	28947	91.77	22802	**91.81**	16778
16384	47.84	610	32.57	610	90.19	28828	**92.12**	17485	91.97	24361

Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

SqueezeNext on Cifar-10



2.0-SqueezeNext on Cifar-10

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	93.25	78125	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
256	93.81	39062	89.51	39062	93.53	50700	93.47	43864	**93.76**	45912
512	93.61	19531	89.14	19531	93.87	37058	93.54	25132	**93.57**	28494
1024	91.51	9766	88.69	9766	93.21	31980	**93.40**	19030	93.26	22741
2048	87.90	4882	88.03	4882	93.54	30030	93.40	21387	**93.50**	23755
4096	80.77	2441	86.21	2441	93.32	29191	93.55	14245	**93.82**	20557

Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

SVHN

- **SVHN consists of ten classes**
  - **Total 600,000 training images**
  - **26,000 testing images**

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Ng Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011

AlexNet on SVHN

AlexNet on SVHN

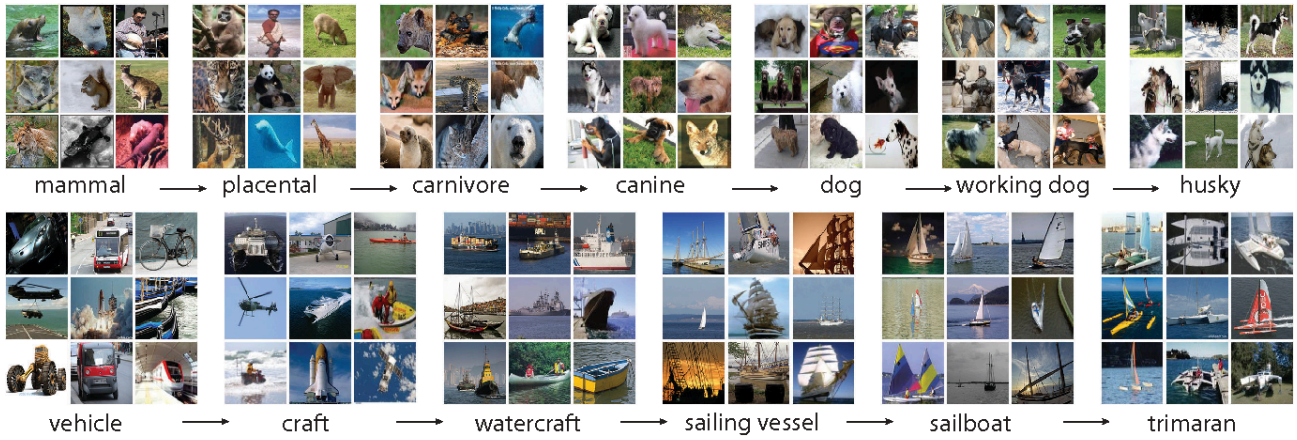
	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	94.90	81986	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
512	94.76	20747	95.24	20747	95.49	51862	95.65	25353	**95.72**	24329
2048	95.17	5186	95.00	5186	95.59	45935	95.51	10562	**95.82**	16578
8192	93.73	1296	19.58	1296	**95.70**	44407	95.56	14400	95.61	7776
32768	91.03	324	10.00	324	95.60	42867	95.60	7996	**95.90**	12616
131072	84.75	81	10.00	81	95.58	42158	95.61	11927	**95.92**	11267

Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

ImageNet

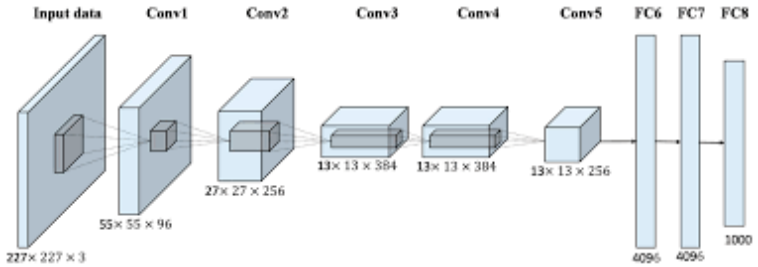
° **ImageNet consists of 1000 classes**

- **Total 1.2 million training images**
- **50,000 testing images**



Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." International Journal of Computer Vision 115.3 (2015): 211-252

AlexNet on Tiny ImageNet



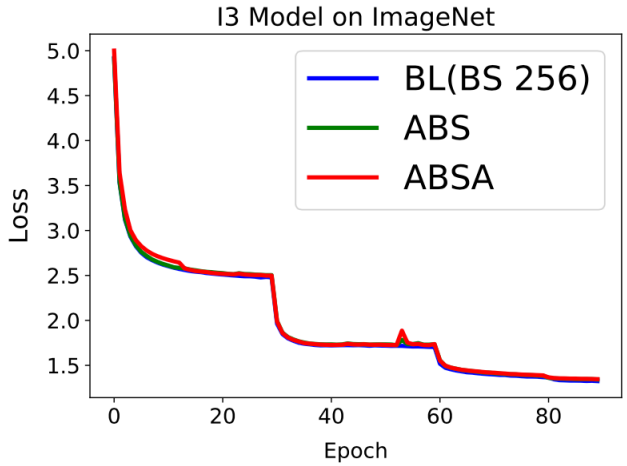
AlexNet on Tiny ImageNet

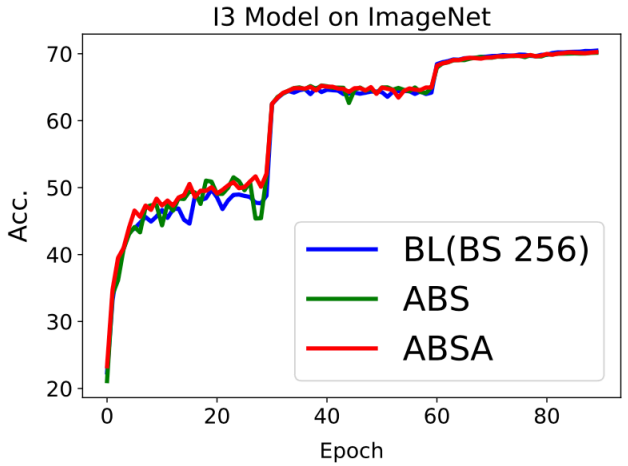
	BL		FB		GG		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	60.41	93750	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
256	58.24	46875	59.82	46875	60.31	70290	**61.28**	60684
512	57.48	23437	59.28	23437	59.94	58575	**60.55**	51078
1024	54.14	11718	59.62	11718	59.72	52717	**60.72**	19011
2048	50.89	5859	59.18	5859	59.82	50667	**60.43**	17313
4096	40.97	2929	58.26	2929	60.09	49935	**61.14**	22704
8192	25.01	1464	16.48	1464	60.00	49569	**60.71**	22334
16384	10.21	732	0.50	732	60.37	48995	**60.71**	20348

Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

ResNet18 on ImageNet

- Baseline:
  - **450k** SGD iterations, **70.4%** validation accuracy
- ABSA:
  - **66k** SGD iterations, **70.2%** validation accuracy
- GG would have required **166k** SGD iterations





Hessian Overhead (theory)

We consider a supervised learning framework where the goal is to minimize a loss function $L(\theta)$:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(z_i, \theta)\,.$$

Let us denote the gradient of $L(\theta)$ w.r.t. θ by g. Then for a random vector, v, whose dimension is the same as g, we have:

$$\frac{\partial(g^T v)}{\partial\theta} = \frac{\partial g^T}{\partial\theta}v + g^T\frac{\partial v}{\partial\theta} = \frac{\partial g^T}{\partial\theta}v = Hv,$$

Where the second equation comes from the fact that v and g are Independant.

https://github.com/amirgholami/HessianFlow

Hessian Eigenvalue Computation Illustration

Algorithm 2: Power Iteration for Eigenvalue Computation

Input: Parameter: θ.

Compute the gradient of θ by backpropagation, *i.e.*, $g = \frac{dL}{d\theta}$.

Draw a random vector v (same dimension as θ).

Normalize v, $v = \frac{v}{\|v\|_2}$

for $i = 1, 2, \ldots, n$ **do** // Power Iteration

  Compute $gv = g^T v$ // Inner product

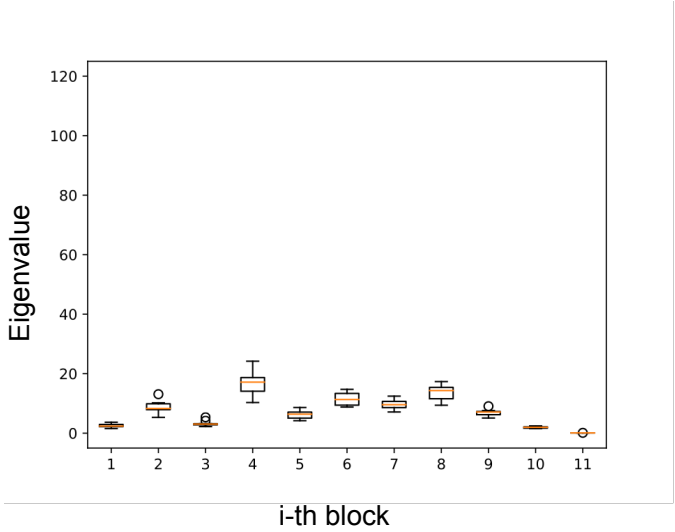
  Compute Hv by backpropagation, $Hv = \frac{d(gv)}{d\theta}$ // Get Hessian vector product

  Normalize and reset v, $v = \frac{Hv}{\|Hv\|_2}$

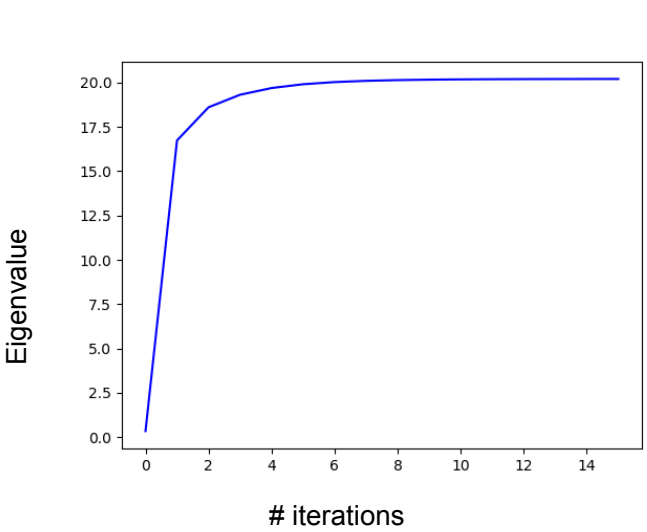
Remaining Questions:

- How many power iterations do we need to compute the top eigenvalues?
- How many data points do we need to get good estimation?

Hessian Eigenvalue Computation Illustration



Top eigenvalue for different blocks using batch size 128 with 10 runs: the variance is very small.



Power iterations needed to compute top eigenvalue is around 10.

Hessian Overhead (practice)

Breakdown of one SGD update training time in terms of

- forward/backwards computation (Tcomp),
- one step communication time (Tcomm),
- one total Hessian spectrum computation (THess), and
- the total training time (TotalTime).

(Results correspond to ResNet18 model on ImageNet.)

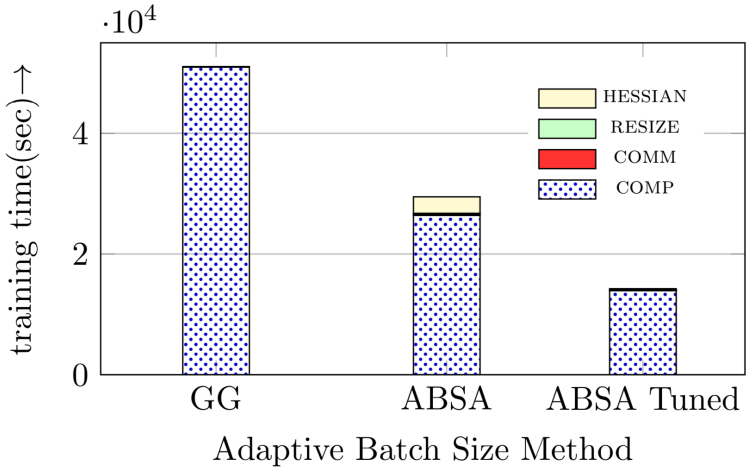
Method	T_{comp}	T_{comm}	T_{Hess}	Total Time
BL	2.2E-2	1.5E-2	0.	16666
GG	2.2E-2	1.5E-2	0.	6150 (2.71× faster)
ABS	2.2E-2	1.5E-2	1.15	2666 (6.25× faster)
ABSA	3.6E-2	1.5E-2	1.15	3467 (4.80× faster)

(Block approximation to Hessian helps even more.)

Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

5

Hessian Overhead (practice)



Amazon Website Service

p3.16x large (8 V100)

Small overhead of

- Hessian Computation
- Communication Time

Method	Comp	Comm	Resize	Hess	Total	Speedup
Baseline	125073	N/A	N/A	N/A	125073	1x
GG	50965	54	40	N/A	51059	2.45x
ABSA	26404	230	95	2746	29475	4.24x
ABSA Tuned	13935	58	39	220	**14252**	**8.78x**

Z. Yao, A. Gholami, D. Arfeen, R. Liaw, J. Gonzalez, K. Keutzer, M. Mahoney, Efficient Adaptive Batch Size Training of Neural Networks, (under review)

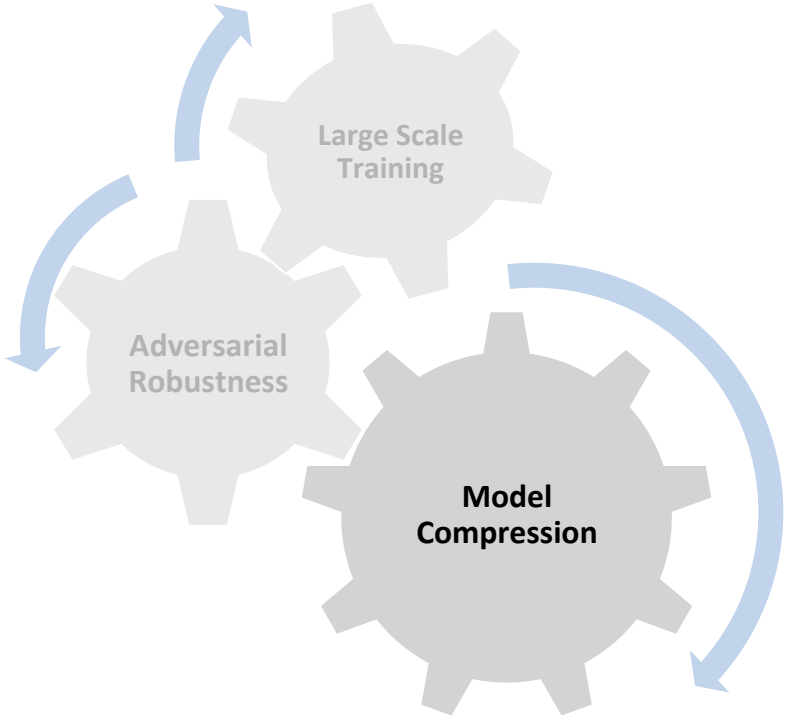
Summary (of Adversarial Examples in Training)

- Using SGD is often very sensitive to non-optimal hyper-parameters

- Robust optimization (during training), e.g., with adversarial data,
  - helps increase stability of the model to adversarial inputs
  - can lead to superior generalization performance than baseline (trained with SGD)
  - is more robust to adversarial perturbation

- Future work: combine robust optimization with subsampled second order methods
  - inexact sub-sampled Hessian
  - Trust region / cubic regularization
  - Newton-MinRes

Second Order Methods



"Machine learning is high performance computing's first killer app for consumers" --- NVIDIA CEO 2015

Second-order methods (use Hessian info as well as gradient info) for:

- **Efficiency/inefficiency of training:** SGD, KFAC, and other 2nd order methods
- **Adversarial examples:** smoothing out ML objectives, using 2nd order methods
- **Quantizing large models:** using outlier metrics derived from 2nd order methods

Existing Approaches for Efficient Inference

Find NNs with smaller memory footprint, latency, power consumption

- SqueezeNet, AutoML based methods (MNasNet, FBNet, MobileNetV3, …)

Co-design NN architecture and hardware together

- SqueezeNext

Prune redundant filters of NN layers

- Works from S. Hong and B. Dally

Quantization: reduce bits in weights/activations

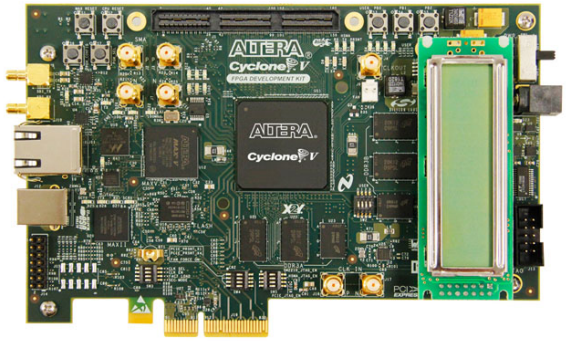
- Helps speed up computations with low arithmetic intensity
- Significantly reduce memory access volume
- State of the art: Lots of ad hoc tricks that don't port to new models

Problem Definition

Goal: Quantize a model in single precision without accuracy loss to enable efficient inference on the edge

Why do We Need Model Compression?

- **Constrained Resources**



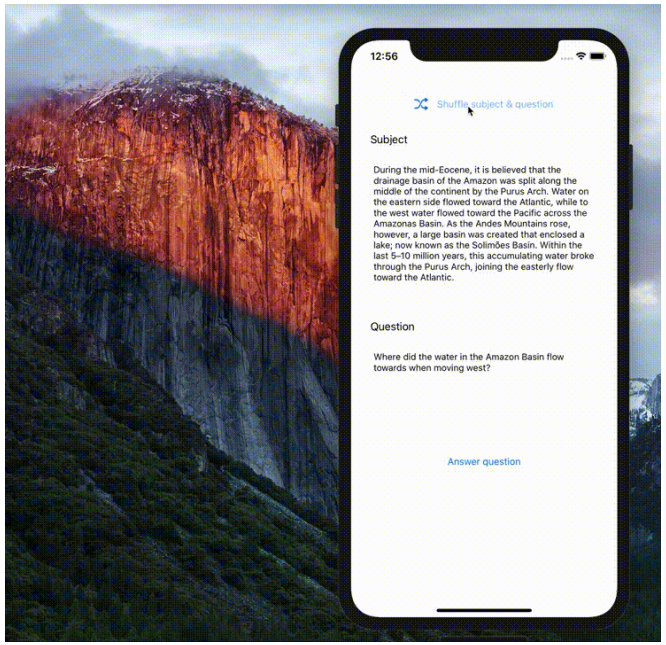
Significant increase in computational resources required for NN training/ inference

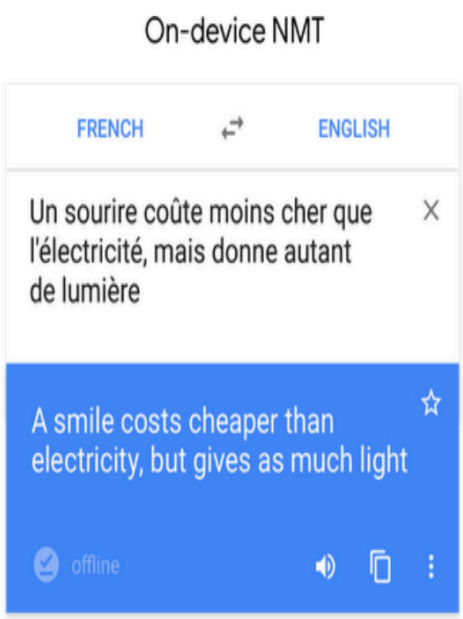
- Larger input sizes (e.g., higher image resolution)
- Larger models (larger memory footprint and more FLOPs, to train and apply)

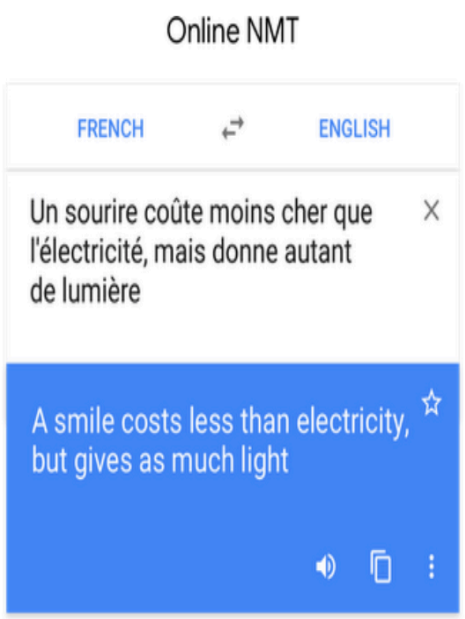
Why Quantization?

Recall: NLP:

- Model size: 110M parameters (BERT-base)







Why Quantization?

- Significantly **reduce memory access volume**
- Allows use of reduced precision ALUs -> **Faster Inference**
- Reduce **communication volume** by allowing execution on embedded devices

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	**640**

6400x

Table: Courtesy of S. Han

Image from here

Quantization is a promising approach but:

- Very hard to get right for a new model/dataset
- Lots of “tricks”
- Lots of expensive hyper-parameter tuning
- **Ad-hoc rules that do not generalize**

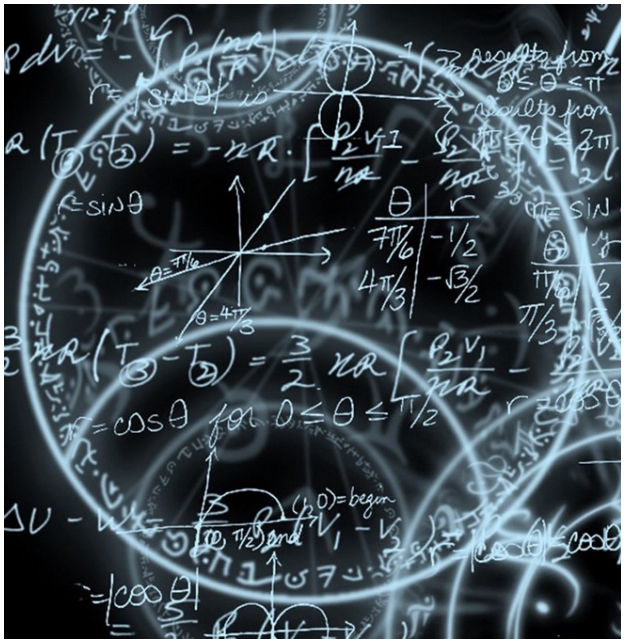


Image from here

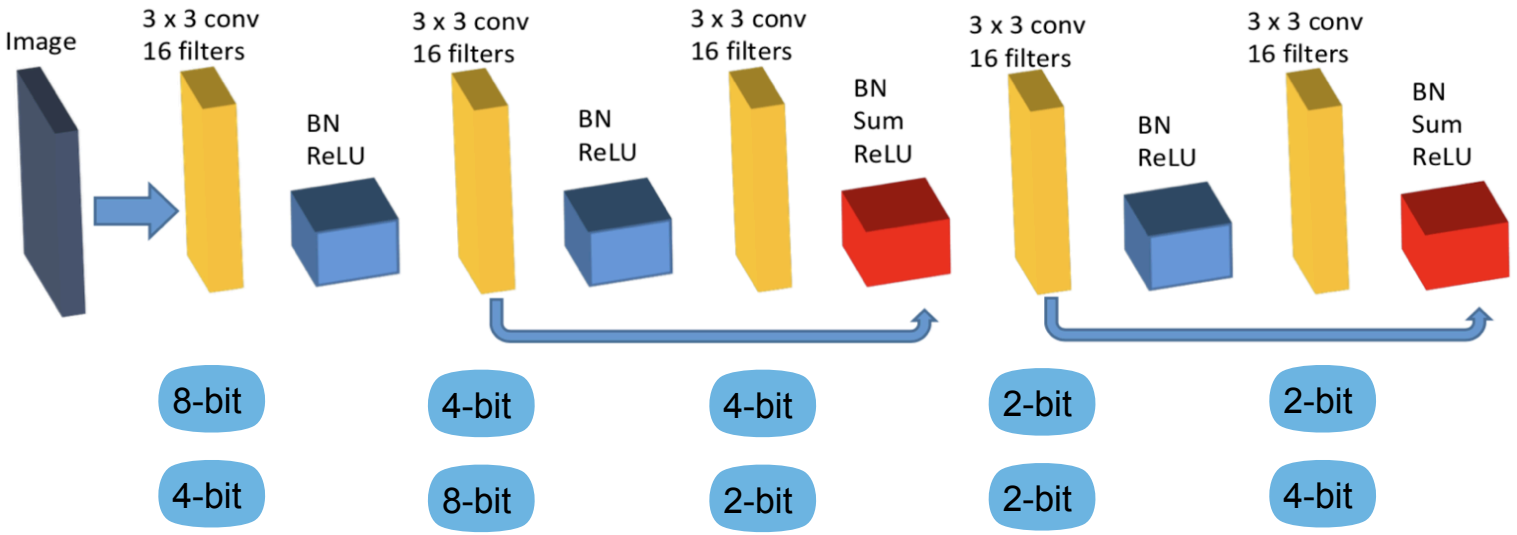
Hessian AWare Quantization

Contributions of **HAWQ**:

- A systematic, **second-order** (Hessian) algorithm for inference quantization
- Fine-tuning schedule based on **second-order** (Hessian) information
- Hessian information = sensitivity information
- Novel compression results exceeding **all existing state-of-the-art** methods for Classification, Object Detection, and NLP
- **No more ad-hoc tricks**

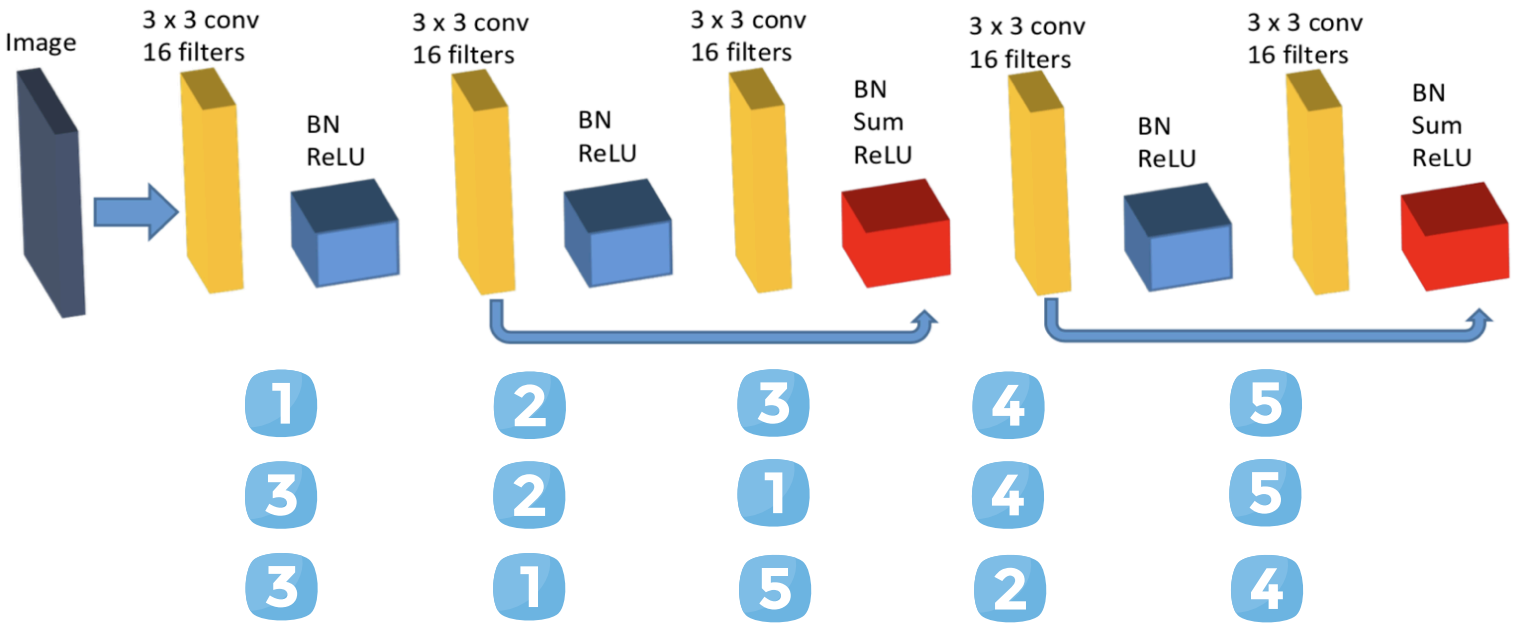
Z. Dong, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer, 2019. HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision. *ICCV'19 (arXiv:1905.03696)*.
S. Sheng, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. Mahoney, K. Keutzer, Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

Mixed-Precision Level: Exponential Search Space



Which mixed-precision setting works better?

Layer-wise Quantization Order: Factorial Search Space

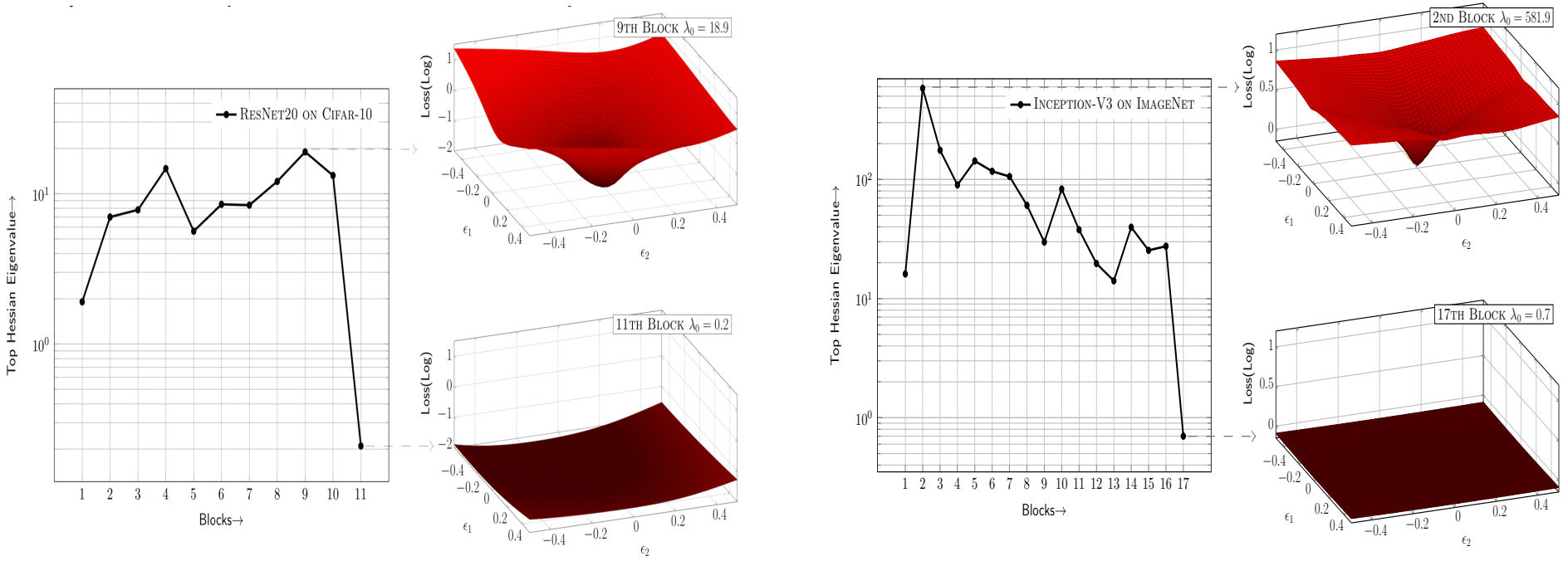


Which quantization order works better?

HAWQ-V1: Provide Relative Weight Precision

Determine quantization precision based on sensitivity/curvature of each layer

- Only quantize layers to **ultra-low precision** that have **small Hessian spectrum**



Z. Dong, Z. Yao, A. Gholami, M. Mahoney, K. Keutzer, HAWQ: Hessian Aware Quantization of Neural Networks With Mixed Precision, ICCV'19 (Accepted)

Hessian Spectrum using Power Iteration

Theorem 1 *After quantizing the model to same precision, fine tuning layers that have smaller average trace of Hessian can achieve a smaller loss, compared to layers with larger* ***average trace of Hessian****.*

$$\text{average } tr(H) = \frac{1}{n}\sum_i H_{ii} = \frac{1}{n}\sum_i \lambda_i(H)$$

- Use power methods to compute Hessian spectrum of any DNN
  - Cost is O(10) gradient backprops
- Code open sourced in pytorch
  - **HessianFlow**
- Extended code to Full Spectrum computation
- Extended code to matrix-free trace computation (Stochastic Lanczos Algorithm)

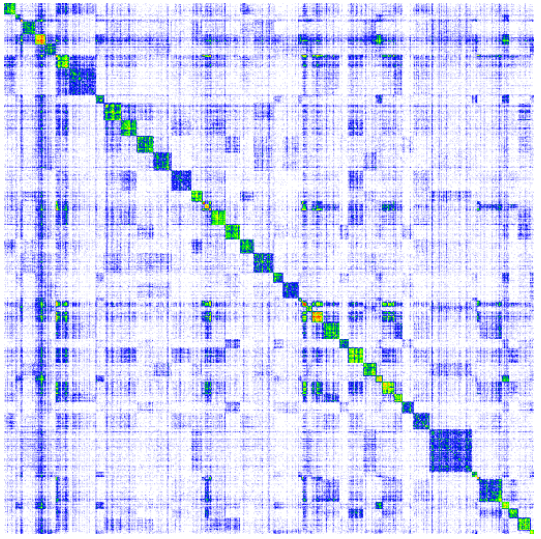


Image Source: Daniel Langr

[1] Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. Mahoney, Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18

HAWQ-V2: SqueezeNext on ImageNet

Precisions for all layers as well as block-wise fine-tuning orders are 100% automatically selected.

Method	w-bits	a-bits	Top-1	W-Comp	Size(MB)
Baseline	32	32	69.38	1.00×	10.1
ResNet18 [27]	32	32	69.76	1.00×	44.7
HAWQ	8	8	**69.34**	4.00×	2.53
Direct	3 MP	8	65.39	9.04×	1.12
HAWQ	3 MP	8	68.02	9.25×	**1.09**
HAWQ-V2	3 MP	8	**68.28**	9.16×	**1.10**

Z. Dong, Z. Yao, A. Gholami, M. Mahoney, K. Keutzer, HAWQ: Hessian Aware Quantization of Neural Networks With Mixed Precision, ICCV'19 (Accepted)

HAWQ-V2: ResNet50 on ImageNet

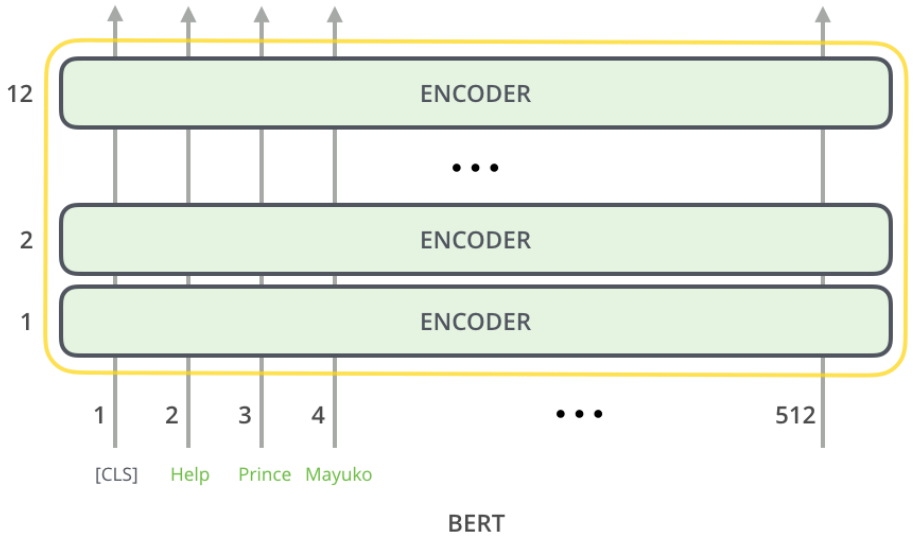
Precisions for all layers as well as block-wise fine-tuning orders are 100% automatically selected.

Method	w-bits	a-bits	Top-1	W-Comp	Size(MB)
Baseline	32	32	77.39	1.00×	97.8
Dorefa [43]	2	2	67.10	16.00×	6.11
Dorefa [43]	3	3	69.90	10.67×	9.17
PACT [2]	2	2	72.20	16.00×	6.11
PACT [2]	3	3	75.30	10.67×	9.17
LQ-Nets [40]	3	3	74.20	10.67×	9.17
Deep Comp. [8]	3	MP	75.10	10.41×	9.36
HAQ [35]	MP	MP	75.30	10.57×	9.22
HAWQ	2 MP	4 MP	75.48	12.28×	**7.96**
HAWQ-V2	2 MP	4 MP	**75.56**	12.25×	**7.98**

Z. Dong, Z. Yao, A. Gholami, M. Mahoney, K. Keutzer, HAWQ: Hessian Aware Quantization of Neural Networks With Mixed Precision, ICCV'19 (Accepted)

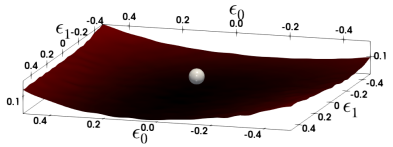
Natural Language Processing

BERT based models have become de-facto architecture for NLP tasks

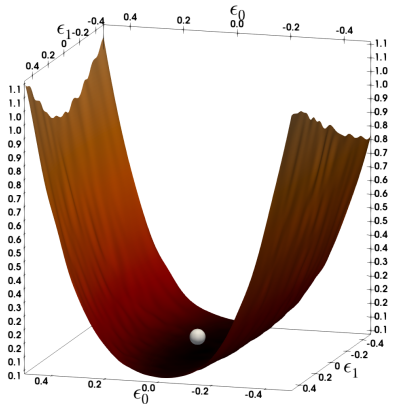


Devlin, Jacob, et al. "BERT: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

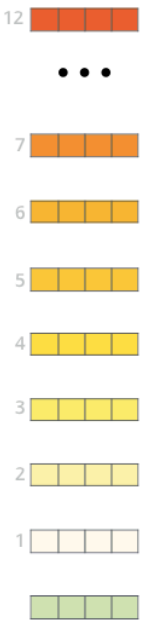
Natural Language Processing



10th Layer



3rd Layer

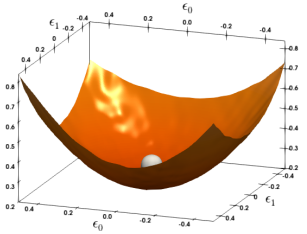


(a) SST-2

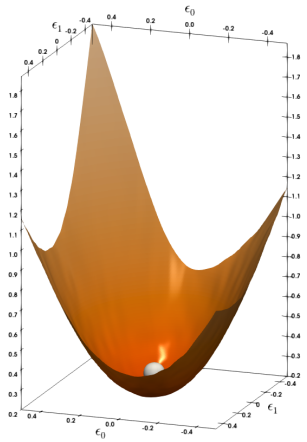
Method	w-bits	e-bits	Acc	Size	Size-w/o-e
Baseline	32	32	93.00	415.4	324.5
Q-BERT	8	8	92.88	103.9	81.2
DirectQ	4	8	85.67	63.4	40.6
Q-BERT	4	8	**92.66**	63.4	40.6
DirectQ	3	8	82.86	53.2	30.5
Q-BERT	3	8	**92.54**	53.2	30.5
Q-BERT$_{MP}$	2/4 $_{MP}$	8	**92.55**	53.2	30.5
DirectQ	2	8	80.62	43.1	20.4
Q-BERT	2	8	**84.63**	43.1	20.4
Q-BERT$_{MP}$	2/3 $_{MP}$	8	**92.08**	**48.1**	**25.4**

S. Sheng, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. Mahoney, K. Keutzer, Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

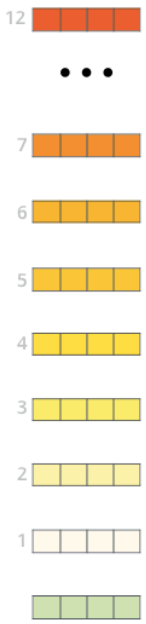
Natural Language Processing



10th Layer



4th Layer

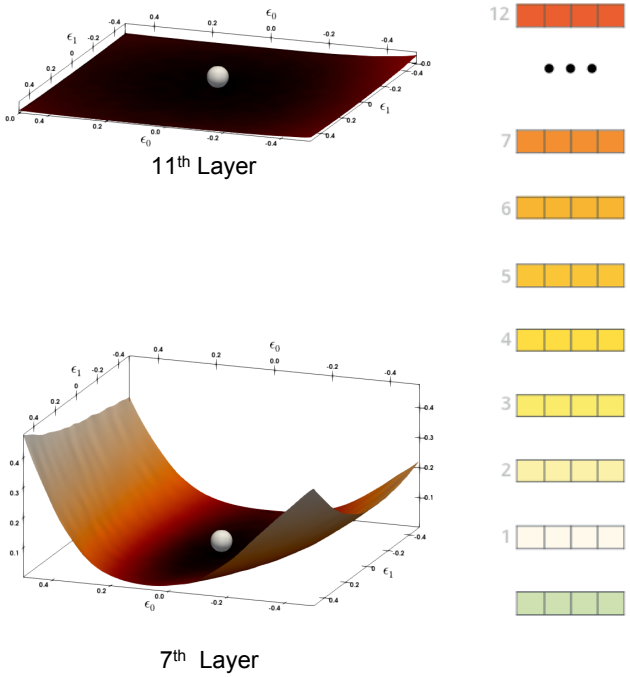


(b) MNLI

Method	w-bits	e-bits	Acc m	Acc mm	Size	Size w/o-e
Baseline	32	32	84.00	84.40	415.4	324.5
Q-BERT	8	8	83.91	83.83	103.9	81.2
DirectQ	4	8	76.69	77.00	63.4	40.6
Q-BERT	4	8	**83.89**	**84.17**	63.4	40.6
DirectQ	3	8	70.27	70.89	53.2	30.5
Q-BERT	3	8	**83.41**	**83.83**	53.2	30.5
Q-BERT$_{MP}$	2/4 $_{MP}$	8	**83.51**	**83.55**	53.2	30.5
DirectQ	2	8	53.29	53.32	43.1	20.4
Q-BERT	2	8	**76.56**	**77.02**	43.1	20.4
Q-BERT$_{MP}$	2/3 $_{MP}$	8	**81.75**	**82.29**	**46.1**	**23.4**

S. Sheng, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. Mahoney, K. Keutzer, Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

Natural Language Processing

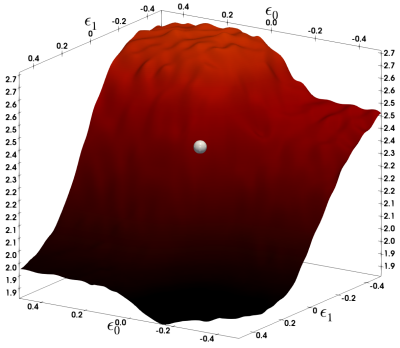


(c) CoNLL-03

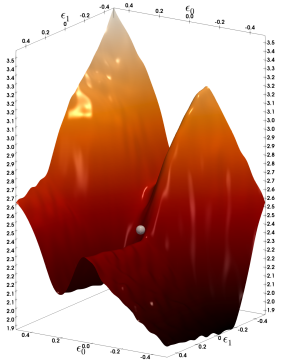
Method	w-bits	e-bits	F_1	Size	Size-w/o-e
Baseline	32	32	95.00	410.9	324.5
Q-BERT	8	8	94.79	102.8	81.2
DirectQ	4	8	89.86	62.2	40.6
Q-BERT	4	8	**94.90**	62.2	40.6
DirectQ	3	8	84.92	52.1	30.5
Q-BERT	3	8	**94.78**	52.1	30.5
Q-BERT$_{\text{MP}}$	2/4 $_{\text{MP}}$	8	**94.55**	52.1	30.5
DirectQ	2	8	54.50	42.0	20.4
Q-BERT	2	8	**91.06**	42.0	20.4
Q-BERT$_{\text{MP}}$	2/3 $_{\text{MP}}$	8	**94.37**	**45.0**	**23.4**

S. Sheng, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. Mahoney, K. Keutzer, Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

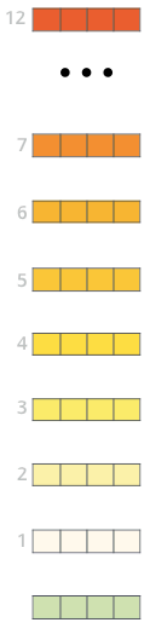
Natural Language Processing



11th Layer



7th Layer



(d) SQuAD

Method	w-bits	e-bits	EM	F_1	Size	Size-w/o-e
Baseline	32	32	81.54	88.69	415.4	324.5
Q-BERT	8	8	81.07	88.47	103.9	81.2
DirectQ	4	8	66.05	77.10	63.4	40.6
Q-BERT	4	8	**80.95**	**88.36**	63.4	40.6
DirectQ	3	8	46.77	59.83	53.2	30.5
Q-BERT	3	8	**79.96**	**87.66**	53.2	30.5
Q-BERT$_{MP}$	2/4 MP	8	**79.85**	**87.49**	53.2	30.5
DirectQ	2	8	4.77	10.32	43.1	20.4
Q-BERT	2	8	**69.68**	**79.60**	43.1	20.4
Q-BERT$_{MP}$	2/3 MP	8	**79.25**	**86.95**	**48.1**	**25.4**

S. Sheng, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. Mahoney, K. Keutzer, Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

Second Order Methods

Conclusions

"If I had asked people what they wanted, they would have said faster horses …"

—Henry Ford

Conclusions

"If I had asked people what they wanted, they would have said

- *faster SGD algorithms,*
- *better worst-case convergence rates,*
- *faster wall-clock times,*
- *better AutoML methods, ..."*

Second order methods

- sometimes do that,
- sometimes don't do that,
- more often lead to improvements---in timing/ robustness/reproducibility/understanding--- for more interesting and non-trivial reasons ...

Conclusions

Second order information:

- Battle-tested method to identify heterogeneities, sensitivity, nonuniformity, etc.

Better training:

- Challenges with large batch; and using adversarial information in novel ways

Better inference:

- Very basic sensitivity measures -> Very practical improved quantization performance

Better for robustness/reproducibility/understanding ...