Rethinking Common Practices in Deep Learning

Elad Hoffer

TECHNION
Israel Institute
of Technology

Deep Learning practices

- Deep learning is evolving fast, while many fundamental questions remain unanswered
  - Many heuristics and intuition-guided decisions
  - It works!
  - But some may prove misguided

We'll cover 5 of these

1. The impact of batch-size on generalization
2. Early-stopping and determining "over-fitting"
3. The role of the last classification layer
4. Batch-norm and regularization
5. "No free lunch" in CNNs

1) Closing the “generalization gap”

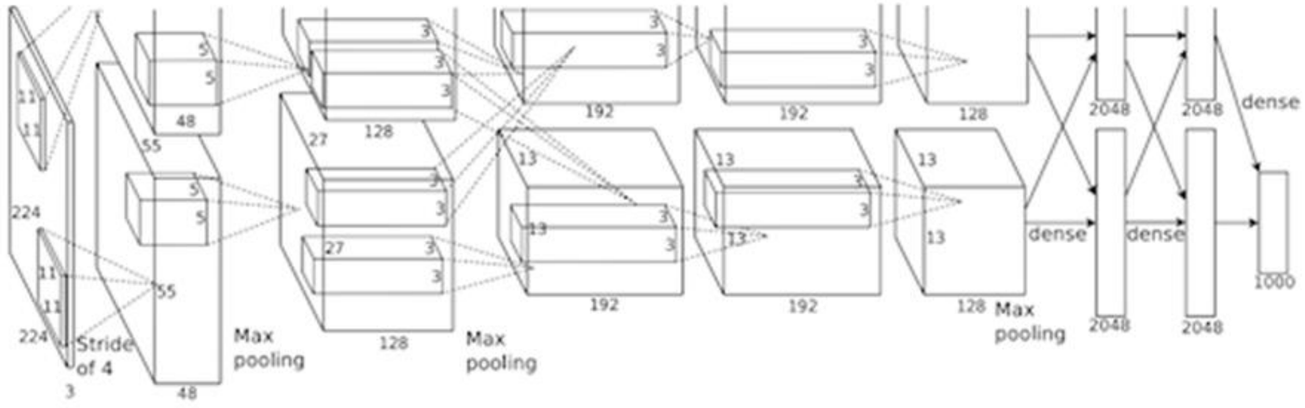
“Train longer, generalize better: closing the generalization gap in large batch training of neural networks” (NIPS 2017 oral)

Elad Hoffer*, Itay Hubara*, Daniel Soudry

Better models - parallelization is crucial

- Model parallelism:

Split model (same data)



AlexNet [Krizhevsky et al. 2012]: model split on two GPUs

- Data parallelism:

Split data (same model)

$$\Delta \mathbf{w} \propto -\frac{1}{b} \sum_{n=1}^{b} \nabla_{\mathbf{w}} L_n(\mathbf{w})$$

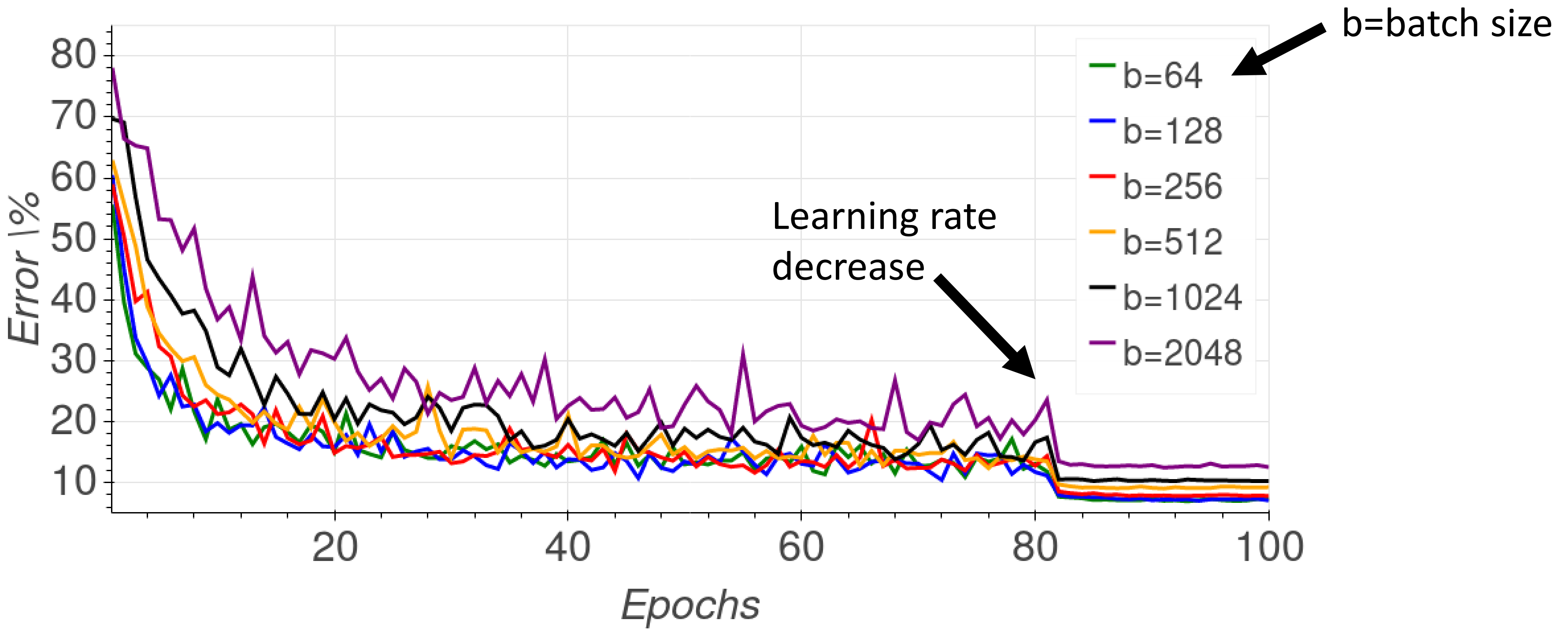
SGD: weight update proportional to gradients averaged over mini batch

Can we increase batch size and improve parallelization?

Large batch size hurts generalization?

Dataset: CIFAR10, **Architecture:** Resnet44, **Training**: SGD + momentum (+ gradient clipping)

Why?

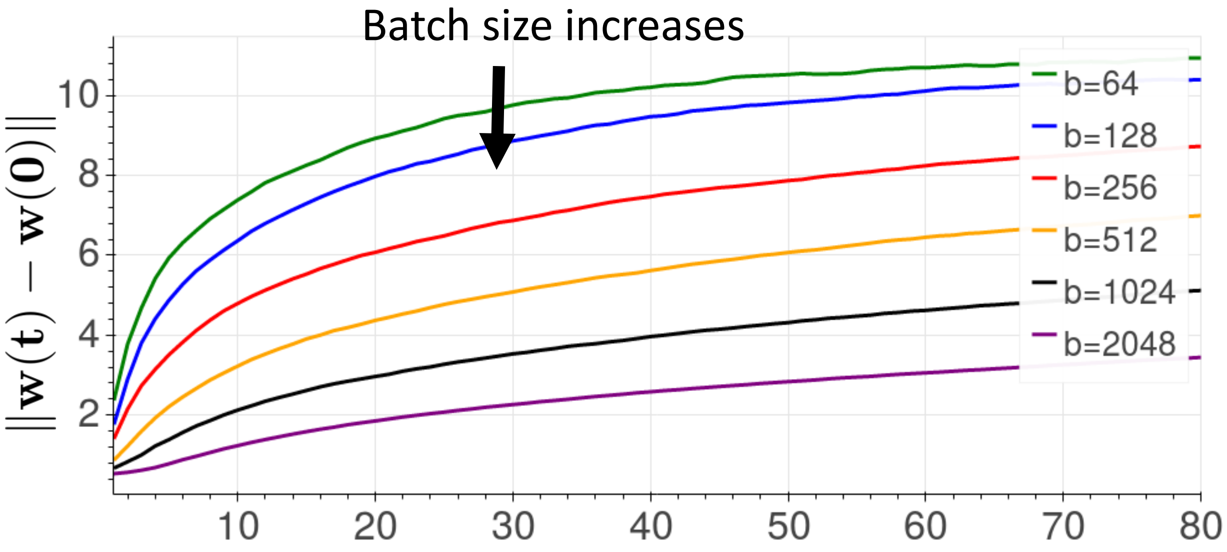


- Generalization gap persisted in models trained “without any budget or limits, until the loss function ceased to improve" [Keskar et al. 2017]

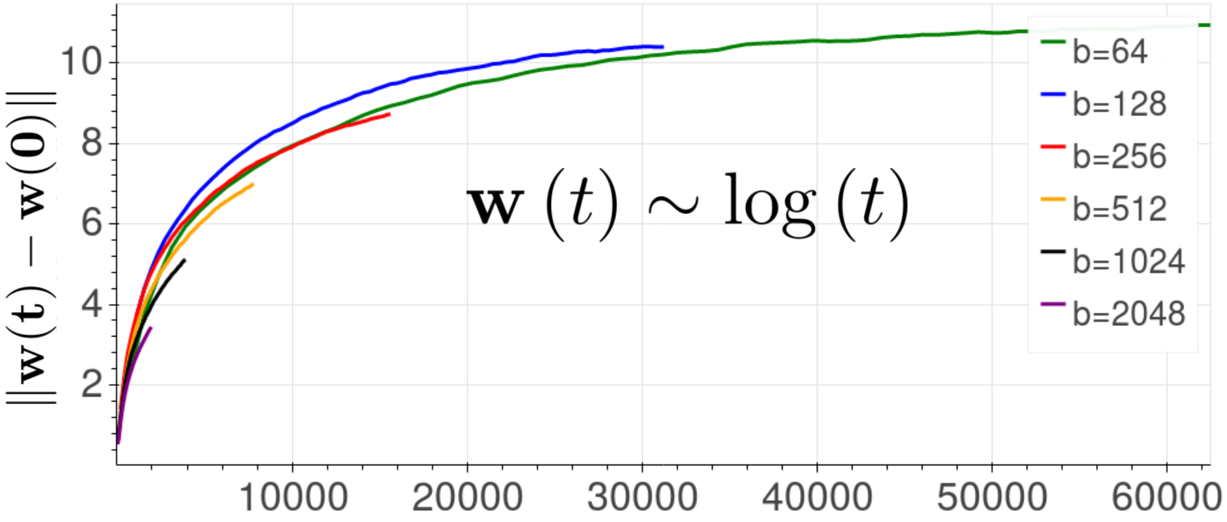
Observation

Weight distances from initialization increase

logarithmically **with iterations**



Epochs



Iterations (gradient updates)

Why logarithmic behavior? Theory later…

Experimental details

- We experiment with various datasets and models
- Optimizing using SGD + momentum + gradient clipping
  - Usually generalize better than adaptive methods (e.g Adam)
  - Grad clipping effectively creates a “warm-up” phase
- Noticeable generalization gap between small and large batch

Network	Dataset	SB	LB
F1 (Keskar et al., 2017)	MNIST	98.27%	97.05%
C1 (Keskar et al., 2017)	Cifar10	87.80%	83.95%
Resnet44 (He et al., 2016)	Cifar10	92.83%	86.10%
VGG (Simonyan, 2014)	Cifar10	92.30%	84.1%
C3 (Keskar et al., 2017)	Cifar100	61.25%	51.50%
WResnet16-4 (Zagoruyko, 2016)	Cifar100	73.70%	68.15%

Closing the generalization gap (2/4)

- Adapt learning rate. In CIFAR $\propto \sqrt{b}$
  - Idea: mimic small batch gradient statistics (dataset dependent)

- Noticeably improves generalization, the gap remains

Network	Dataset	SB	LB	+LR
F1 (Keskar et al., 2017)	MNIST	98.27%	97.05%	97.55%
C1 (Keskar et al., 2017)	Cifar10	87.80%	83.95%	86.15%
Resnet44 (He et al., 2016)	Cifar10	92.83%	86.10%	89.30%
VGG (Simonyan, 2014)	Cifar10	92.30%	84.1%	88.6%
C3 (Keskar et al., 2017)	Cifar100	61.25%	51.50%	57.38%
WResnet16-4 (Zagoruyko, 2016)	Cifar100	73.70%	68.15%	69.05%

Closing the generalization gap (3/4)

- Ghost batch norm
  - Idea again: mimic small batch size statistics
  - Also: reduces communication bandwidth
  - Further improves generalization without incurring overhead

$$\hat{x} = \frac{x - \mu_x}{\sigma_x}$$

$$\widehat{x_1} = \frac{x_1 - \mu_{x_1}}{\sigma_{x_1}}$$

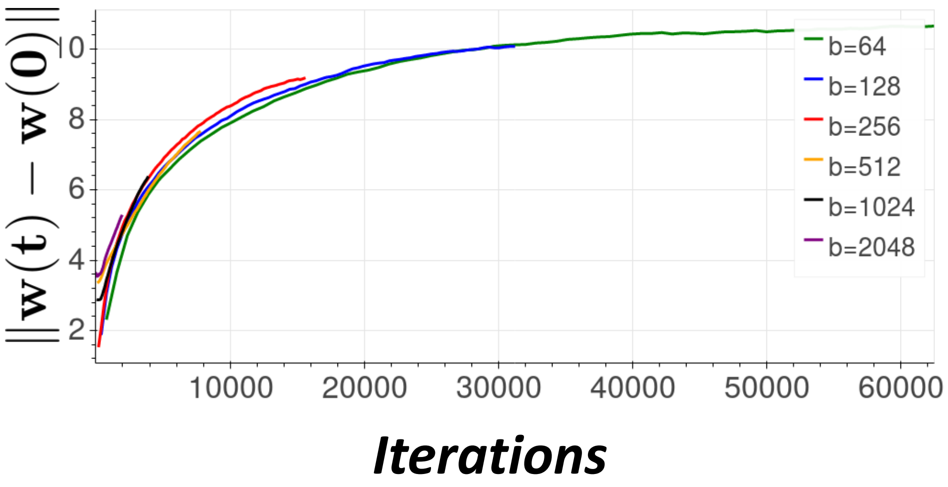
$$\vdots$$

$$\widehat{x_k} = \frac{x_k - \mu_{x_k}}{\sigma_{x_k}}$$

Network	Dataset	SB	LB	+LR	+GBN
F1 (Keskar et al., 2017)	MNIST	98.27%	97.05%	97.55%	97.60%
C1 (Keskar et al., 2017)	Cifar10	87.80%	83.95%	86.15%	86.4%
Resnet44 (He et al., 2016)	Cifar10	92.83%	86.10%	89.30%	90.50%
VGG (Simonyan, 2014)	Cifar10	92.30%	84.1%	88.6%	91.50%
C3 (Keskar et al., 2017)	Cifar100	61.25%	51.50%	57.38%	57.5%
WResnet16-4 (Zagoruyko, 2016)	Cifar100	73.70%	68.15%	69.05%	71.20%

Graph indicates: not enough iterations?

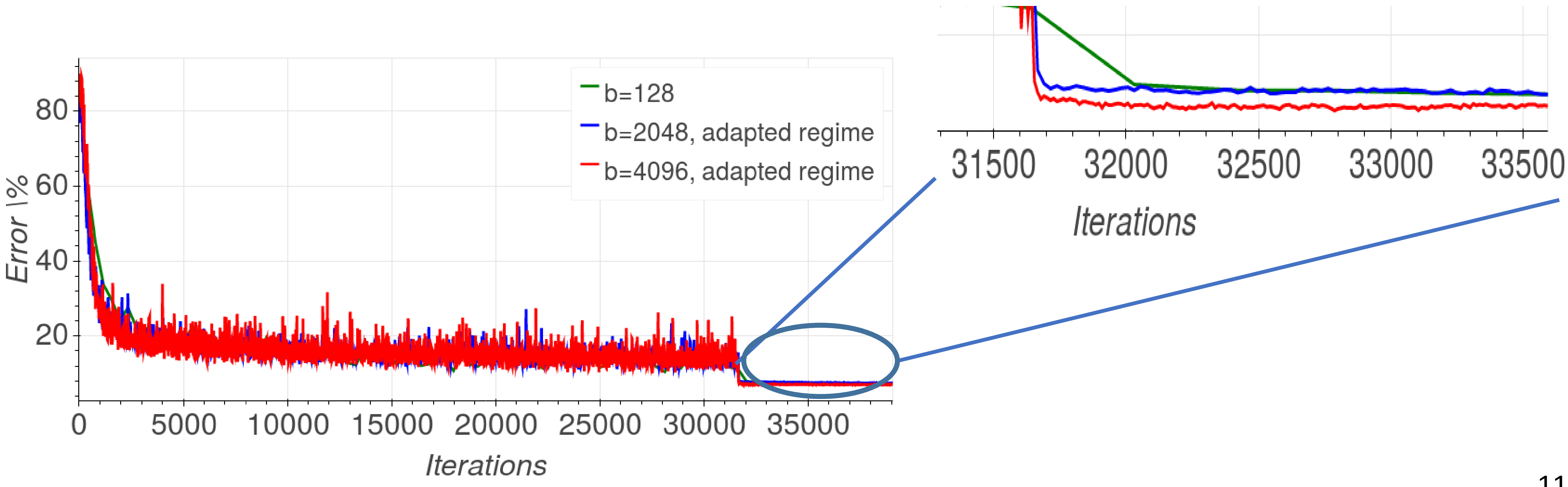
- Using these modifications – distance from initialization now better matched
- However, graph indicates: insufficient iterations with large batch



Network	Dataset	SB	LB	+LR	+GBN
F1 (Keskar et al., 2017)	MNIST	98.27%	97.05%	97.55%	97.60%
C1 (Keskar et al., 2017)	Cifar10	87.80%	83.95%	86.15%	86.4%
Resnet44 (He et al., 2016)	Cifar10	92.83%	86.10%	89.30%	90.50%
VGG (Simonyan, 2014)	Cifar10	92.30%	84.1%	88.6%	91.50%
C3 (Keskar et al., 2017)	Cifar100	61.25%	51.50%	57.38%	57.5%
WResnet16-4 (Zagoruyko, 2016)	Cifar100	73.70%	68.15%	69.05%	71.20%

Train longer, generalize better

- With sufficient iterations in “plateau” region, generalization gap vanish:



Closing the generalization gap (4/4)

- Regime Adaptation – train so that the number of iterations is fixed for all batch sizes (train longer number of epochs)
  - Completely closes the generalization gap

Network	Dataset	SB	LB	+LR	+GBN	+RA
F1 (Keskar et al., 2017)	MNIST	98.27%	97.05%	97.55%	97.60%	98.53%
C1 (Keskar et al., 2017)	Cifar10	87.80%	83.95%	86.15%	86.4%	88.20%
Resnet44 (He et al., 2016)	Cifar10	92.83%	86.10%	89.30%	90.50%	93.07%
VGG (Simonyan, 2014)	Cifar10	92.30%	84.1%	88.6%	91.50%	93.03%
C3 (Keskar et al., 2017)	Cifar100	61.25%	51.50%	57.38%	57.5%	63.20%
WResnet16-4 (Zagoruyko, 2016)	Cifar100	73.70%	68.15%	69.05%	71.20%	73.57%

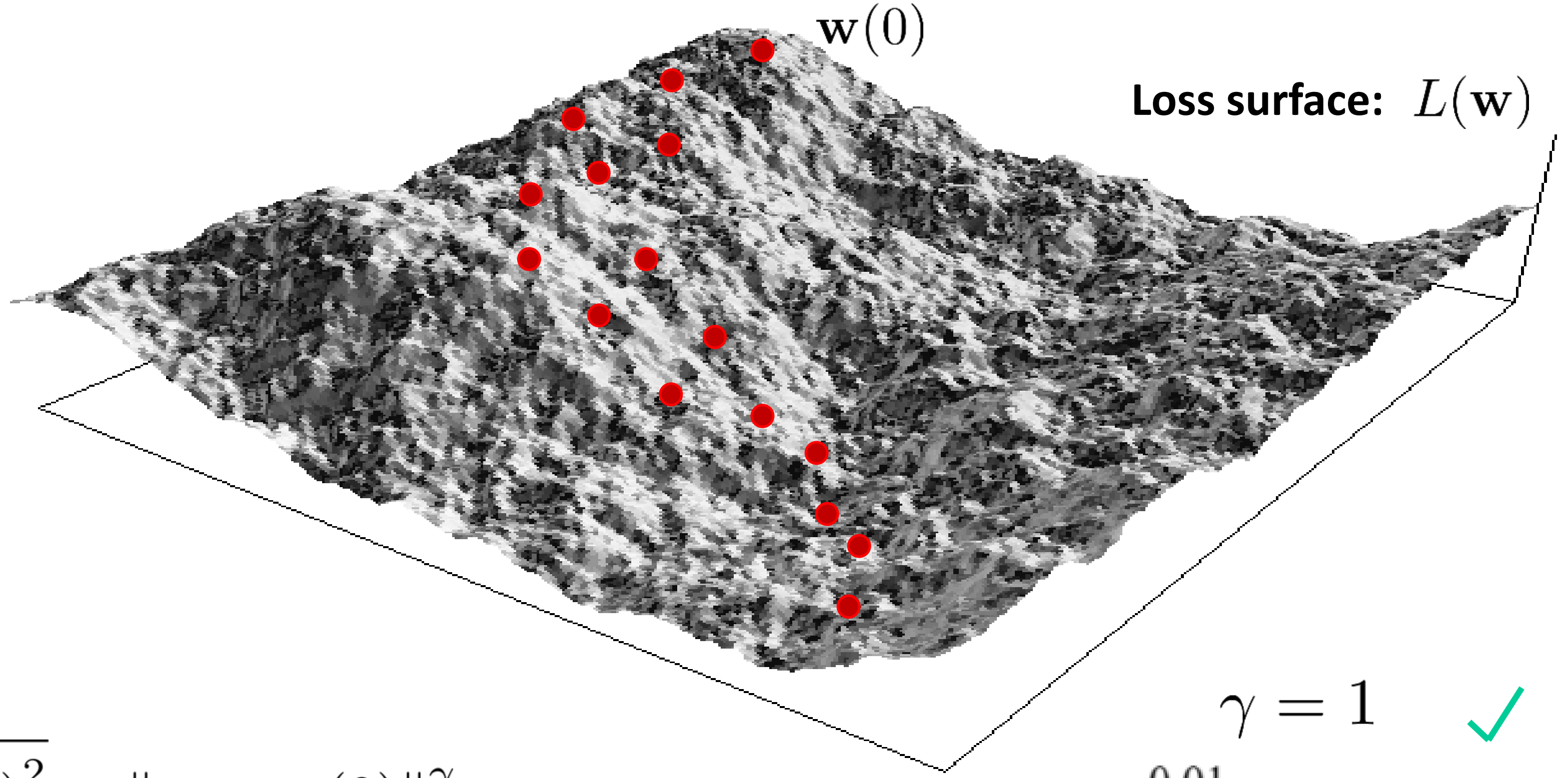
ImageNet (AlexNet):

LB size	Dataset	SB	LB[8]	+LR[8]	+GBN	+RA
4096	ImageNet	57.10%	41.23%	53.25%	54.92%	59.5%
8192	ImageNet	57.10%	41.23%	53.25%	53.93%	59.5%

Why weight distances increase logarithmically?

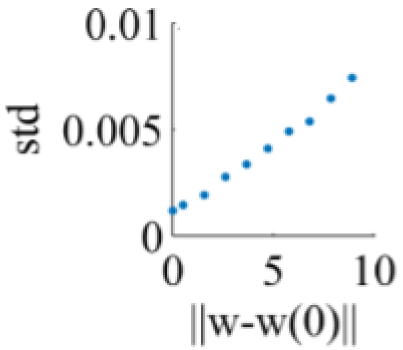
Hypothesis:

During initial high learning rate phase: "random walk on a random potential" where



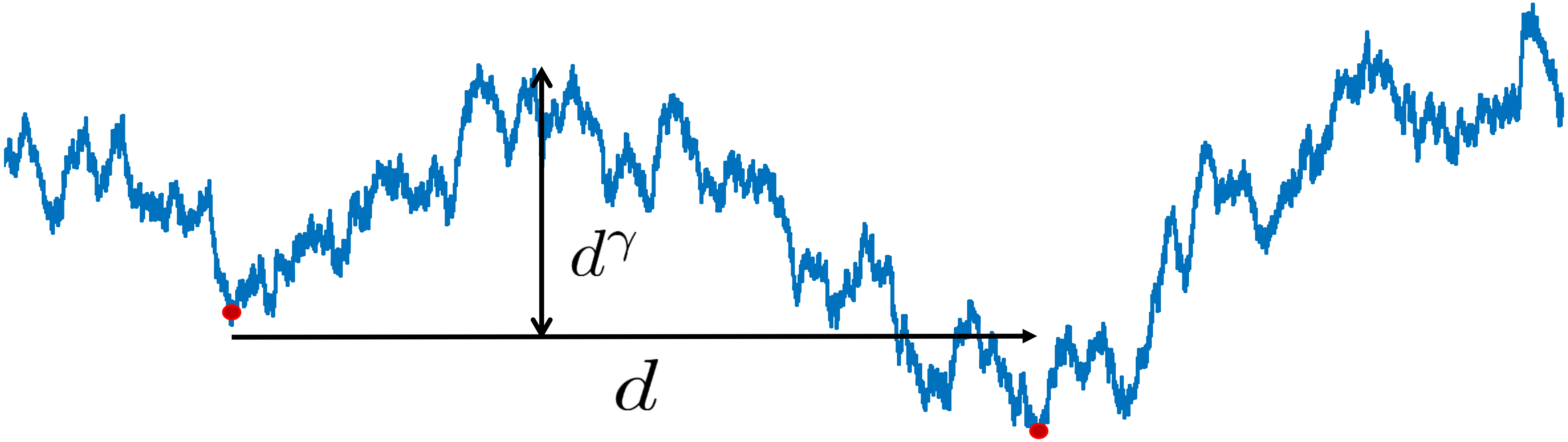
$$\mathrm{std} \triangleq \sqrt{\mathbb{E}\left(L\left(\mathbf{w}\right) - L\left(\mathbf{w}(0)\right)\right)^2} \sim \left\|\mathbf{w} - \mathbf{w}(0)\right\|^{\gamma}$$

$\gamma = 1$ ✓



Marinari et al., 1983: $\mathbf{w}\left(t\right) \sim \log^{\frac{1}{\gamma}}\left(t\right)$ **"ultra-slow diffusion"**

Ultra-slow diffusion: Basic idea



Time to pass tallest barrier: $t \propto \exp(d^\gamma)$ $\Rightarrow d \propto \log^{\frac{1}{\gamma}}(t)$

Summary so far

- **Q:** Is there inherent generalization problem with large batches?
  A: Observed: no, just adjust training regime.

- **Q:** What is the mechanism behind training dynamics?
  A: Hypothesis: "random walk on a random potential”

- **Q:** Can we reduce the total wall clock time?
  A: Yes, in some models

Significant speed-ups possible

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

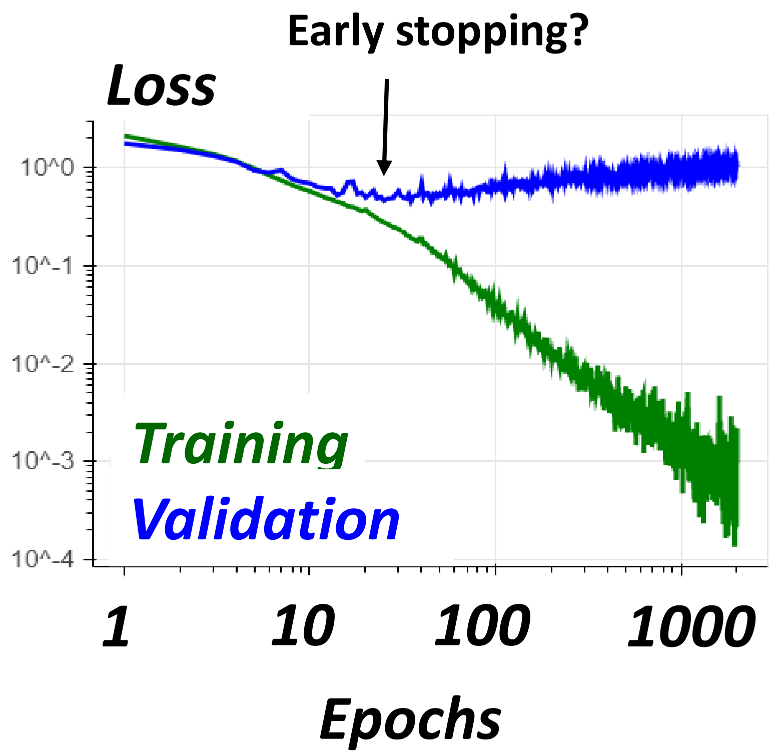
Goyal et al. (Facebook whitepaper, two weeks after us)

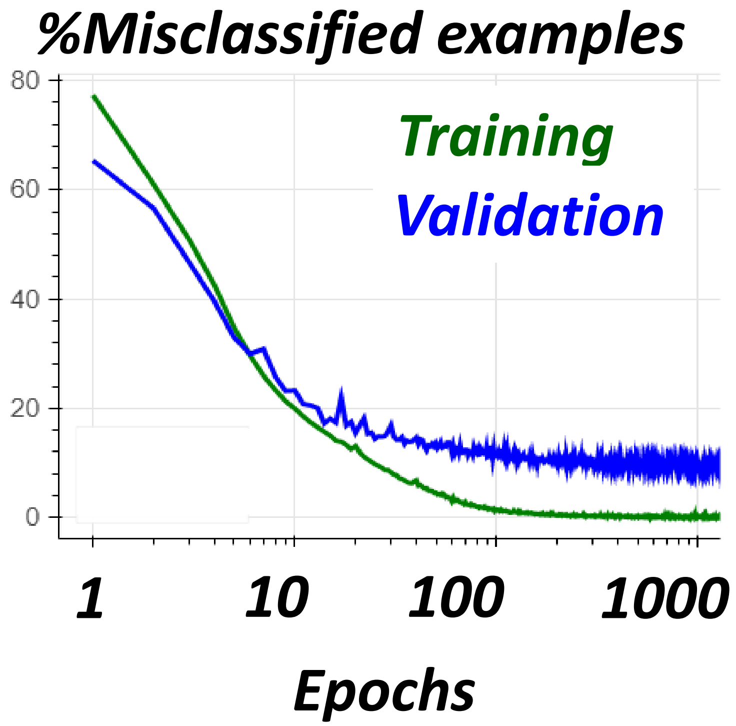
- Large scale experiments: ResNet over ImageNet, 256 GPUs
- Similar methods, except learning rate
- X29 times faster than a single worker

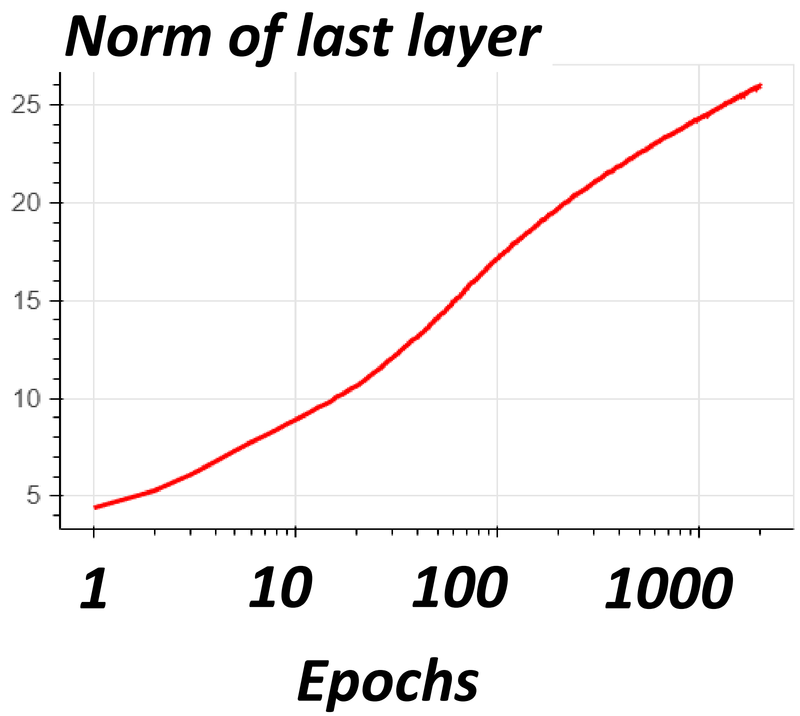
- More followed:
  - **Large Batch Training of Convolutional Networks** (You et al.)
  - **ImageNet Training in Minutes** (You et al.)
  - **Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes** (Akiba et al.)

2) Why “Overfitting” is good for generalization?

- In contrast to common practice: good generalization results from many gradient updates in an “overfitting regime”







Peculiar generalization dynamics - summary

- Validation Loss increases
- Training error + loss goes to zero
- Weight Norm diverges

Looks like we are overfitting… but

- Validation error (classification) seems to never stop decreasing (slowly)

Conclusion: No need for early stopping (!)

How all of this makes sense?

Why “Overfitting” is good for generalization?

- Can be shown to happen for logistic regression on separable data!
- Slow convergence to max-margin solution

The Implicit Bias of Gradient Descent on Separable Data (ICLR 2018)

- *Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Nati Srebro*

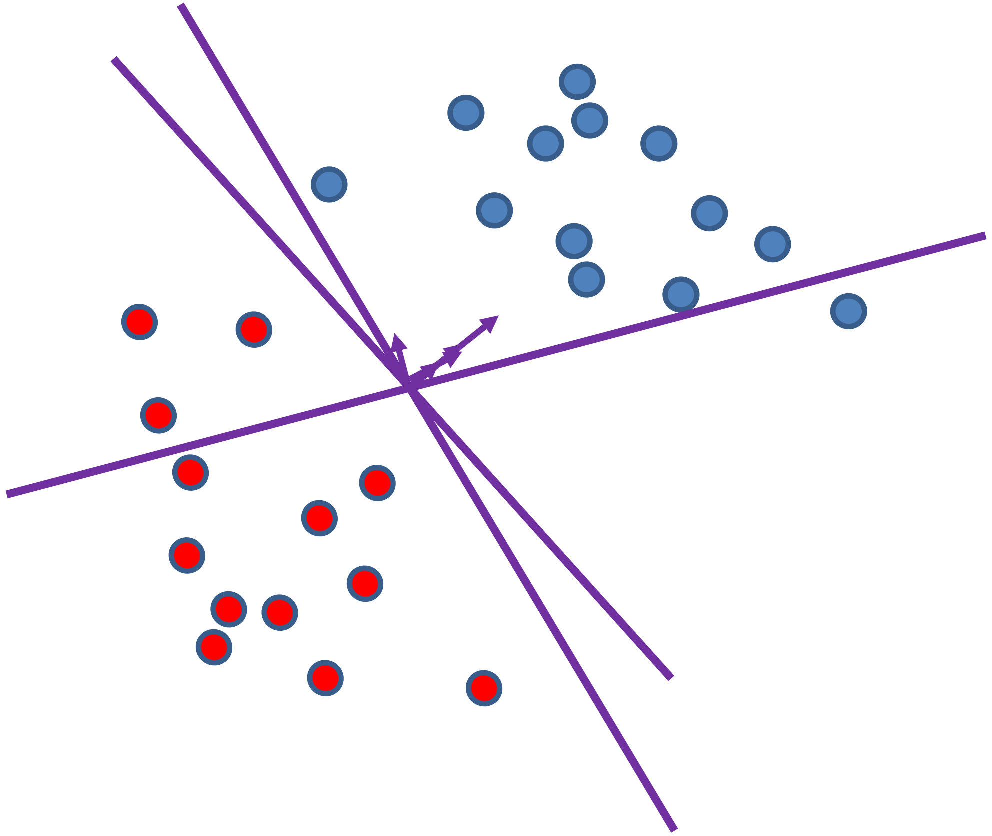
Main Theorem

Gradient descent on logistic loss: $\Delta \mathbf{w} = -\eta \nabla \mathcal{L}(\mathbf{w})$

Theorem 1: $\mathbf{w}(t) = \hat{\mathbf{w}} \log t + \boldsymbol{\rho}(t)$,

$\hat{\mathbf{w}}$ is the (L2) max margin vector

$\boldsymbol{\rho}(t)$ is bounded, for almost every dataset.



Therefore: $$\frac{\mathbf{w}(t)}{\|\mathbf{w}(t)\|} \rightarrow \frac{\hat{\mathbf{w}}}{\|\hat{\mathbf{w}}\|}$$

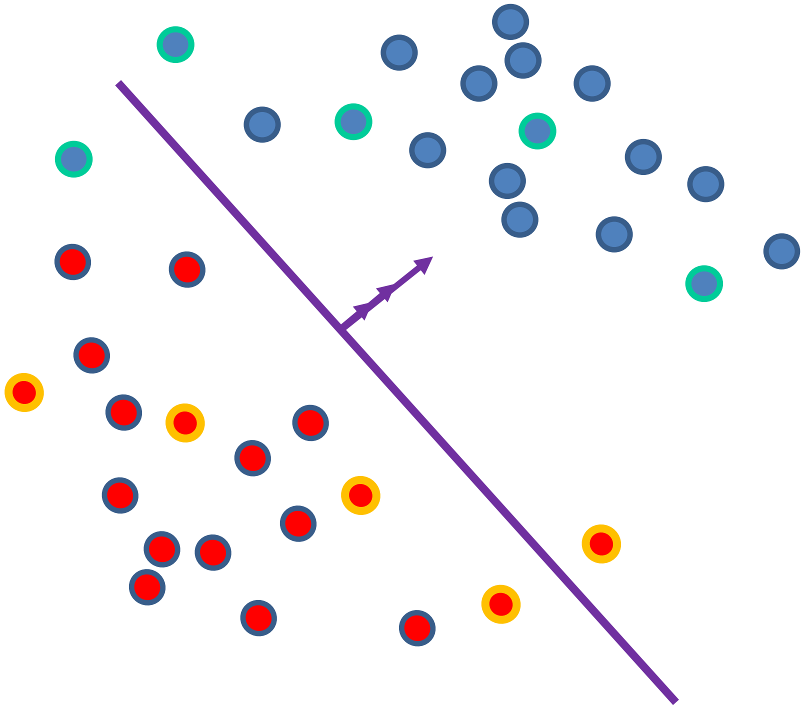
… While expected loss (and test loss) increases

$$\mathbf{w}(t) = \hat{\mathbf{w}} \log t + \boldsymbol{\rho}(t)$$

Expected loss:

$$\mathbb{E}[\mathcal{L}(\mathbf{w})] = \Omega(\log t).$$

Also true for test loss



Validation loss is expected to increase, although accuracy may still improve!

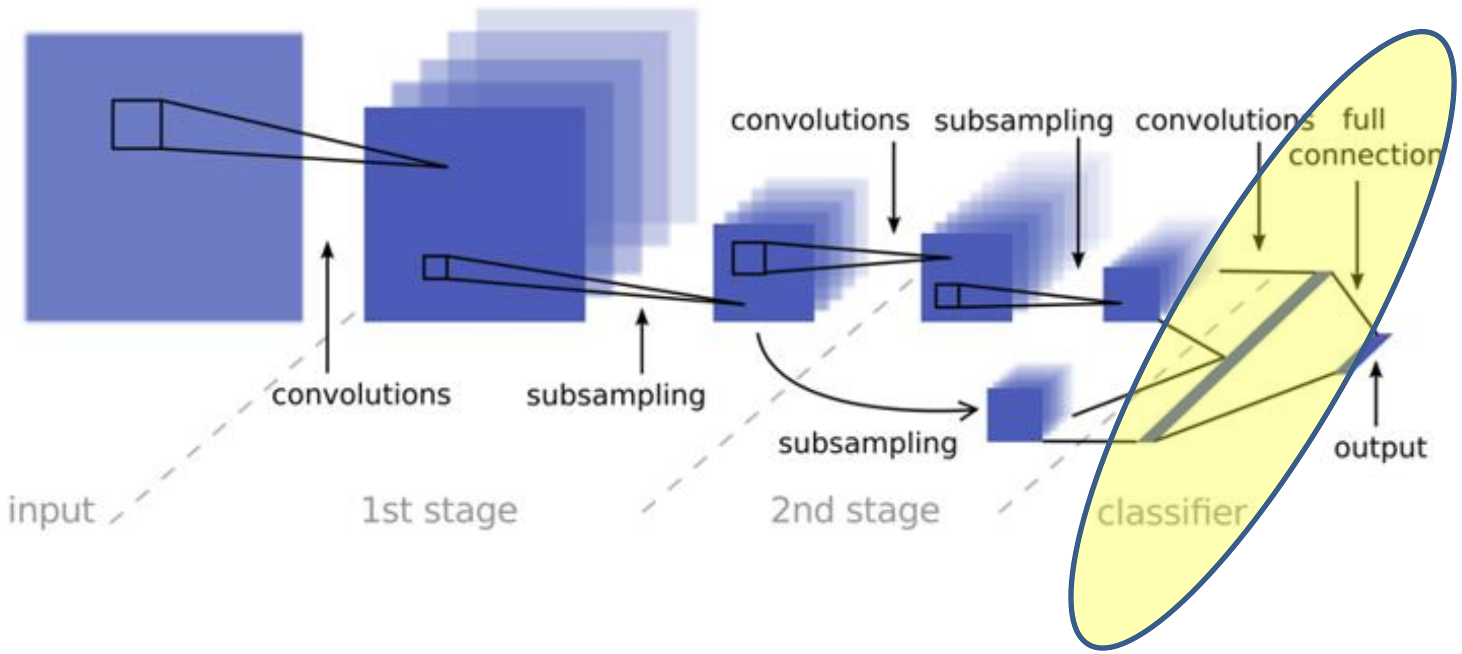
3) The role of the final classifier

Fix your classifier: the marginal value of training the last weight layer

Elad Hoffer, Itay Hubara, Daniel Soudry

Fully connected classifier

We focus on the final representation obtained by the network F before the classifier $x = F(z;\theta)$ (the last hidden layer) .



Fully connected classifier

- In common NN models, this representation is followed by an additional affine transformation on $x \in \mathbb{R}^{\mathrm{C}}$ to all possible classes C.

$$y = W^T x + b$$

- where the number of parameters is dependent on number of classes $W \in \mathbb{R}^{\mathrm{N} \times C}$ (can grow to be very large)

Fully connected classifier

Training is done using cross-entropy loss, by feeding the network outputs through a softmax activation

$$v_i = \frac{e^{y_i}}{\sum_j^C e^{y_j}}, \;\; i \in \{1, \ldots, C\}$$

and reducing the expected negative log likelihood with respect to ground-truth target

$$\mathcal{L}(x, t) = -\log v_t = -w_t \cdot x - b_t + \log\left(\sum_j^C e^{w_j \cdot x + b_j}\right)$$

where w_i is the i-th column of W.

Fully connected classifier?

But the final fully-connected transform is a linear classifier:

- The network learns features that are already separable at this point.
- They fully-connected layers are also notoriously redundant -- easily compressed and discarded

Can they be removed completely?

Fixed classifier

- To evaluate our conjecture, we replaced the trainable parameter matrix W with a fixed orthonormal projection

$$Q \in \mathbb{R}^{N \times C} \text{ such that } QQ^T = I_n$$

- As the rows of classifier weight matrix are fixed with an equal L_2 norm, we also restrict the representation of x to reside on the n-dimensional sphere

$$\hat{x} = \frac{x}{\|x\|_2}$$

Fixed classifier

Since $-1 \leq q_i \cdot \hat{x} \leq 1$ and softmax function is scale-sensitive, we introduce another temperature scaling coefficient α

$$v_i = \frac{e^{\alpha q_i \cdot \hat{x} + b_i}}{\sum_j^C e^{\alpha q_j \cdot \hat{x} + b_j}}, \ i \in \{1, \ldots, C\}$$

and we minimize the loss:

$$\mathcal{L}(x, t) = -\alpha q_t \cdot \frac{x}{\|x\|_2} + b_t + \log\left(\sum_{i=1}^{C} \exp\left(\alpha q_i \cdot \frac{x}{\|x\|_2} + b_i\right)\right)$$

Hadamard classifier

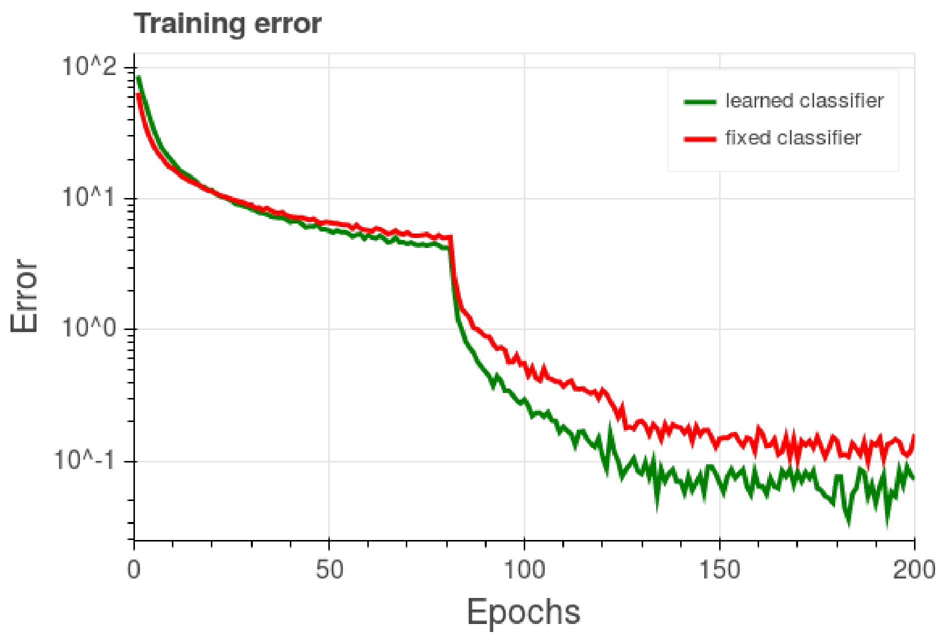
The fixed orthogonal weights can be chosen to be a Hadamard matrix:

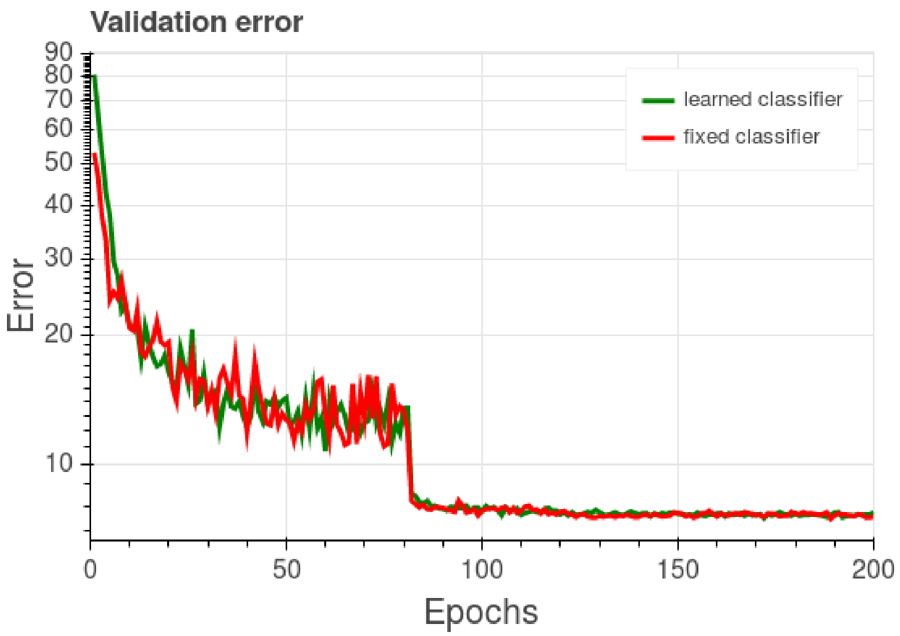
$$H^T H = nI_n \,, \qquad H \in \{-1,1\}^n$$

- A deterministic, low-memory and easily generated matrix that can be used to classify.
- Removal of the need to perform a full matrix-matrix multiplication -- as multiplying by a Hadamard matrix can be done by simple sign manipulation and addition.

Learned vs. Fixed classifier

- We compare the training of a **fully-learned classifier** with a **fixed classifier** (Cifar10, ResNet)
  - Training error is lower when using a learned classifier.
  - Both achieve the same accuracy on the validation set.





Empirical results

We find that this behavior remains in other datasets and models

- Negligible decrease in accuracy when final layer is fixed
- Reduces number of weights -- e.g ShuffleNet, where most of the parameters are in the last layer

Network	Dataset	Learned	Fixed	# Params	% Fixed params
Resnet56	Cifar10	93.03%	93.14%	855,770	0.07%
DenseNet(k=12)	Cifar100	77.73%	77.67%	800,032	4.2%
Resnet50	ImageNet	75.3%	75.3%	25,557,032	8.01%
DenseNet169	ImageNet	76.2%	76%	14,149,480	11.76%
ShuffleNet	ImageNet	65.9%	65.4%	1,826,555	52.56%

4) Batch-norm and regularzation

“Norm matters: efficient and accurate normalization schemes in deep networks”

Elad Hoffer, Ron Banner, Itay Golan, Daniel Soudry

Batch normalization

- Batch-norm (Ioffe, 15’) is widely used, and allowed us to train deeper models, faster and with better final accuracy.
- However, it also has several shortcomings:
  - Assumes independence between samples (problem when modeling time-series, RL, GANs, metric-learning etc.)
  - Requires high-precision operation and accumulation ($\sqrt{\sum x^2}$) and may be numerically instable
  - Significant computational and memory impact, with data-bound operations – makes up to 25% of computation time in current models (Gitman, 17’)
  - Not clear why it works so well (“reducing covariate shift” ???)

Batch-norm Leads to norm invariance

The key observation regarding Batch-norm:

- Given an input x
- Channels' weight vector w, where its direction is $\hat{w} = \frac{w}{\|w\|}$
- Batch-norm is norm invariant:

$$BN(\|w\|\hat{w}x) = BN(\hat{w}x)$$

- Weight-decay on these weights is just a learning-rate scaling (see paper for details)

Replacing Batch-norm with weight-norm

Can we do better than batch-norm?

- We can fix weight-norm (Salimans, 16’) which had issues in large scale models (Gitman, 17’) by detaching the norm from the weights completely:

In weight norm, for a channel i:

$$w_i = g_i \frac{v_i}{\|v_i\|}$$

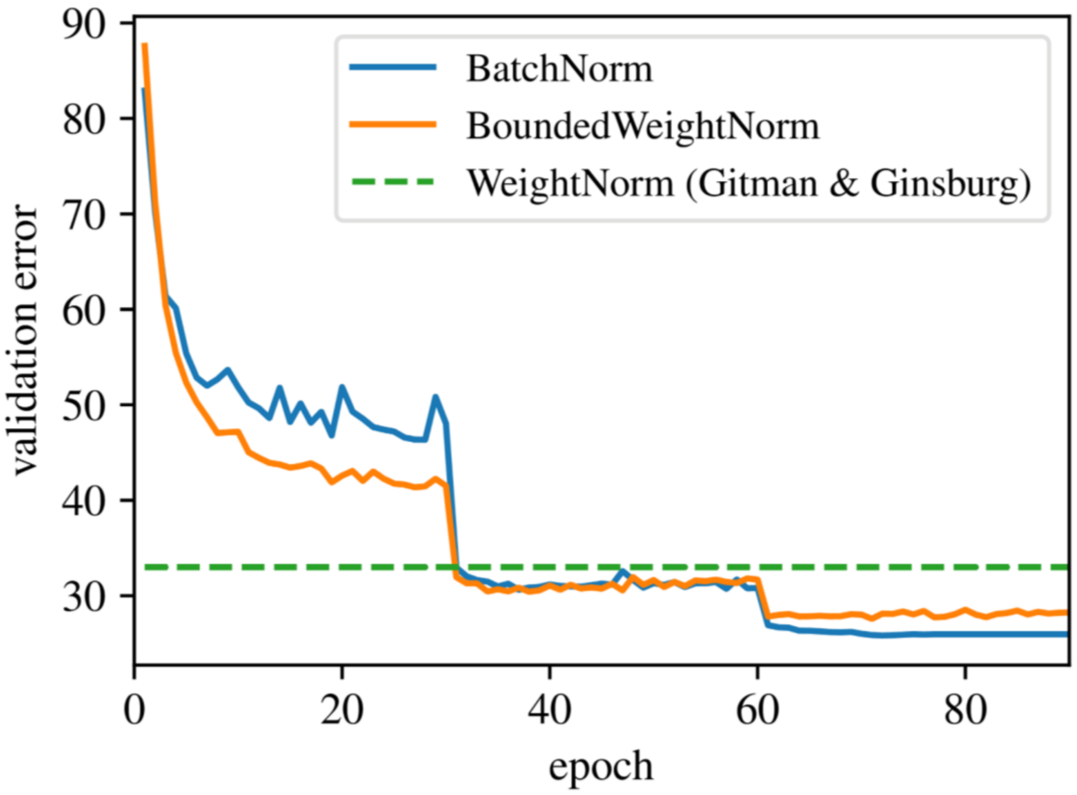
We now suggest to completely remove the norm by parameterizing

$$w_i = \rho \frac{g_i}{\|g\|} \frac{v_i}{\|v_i\|}$$

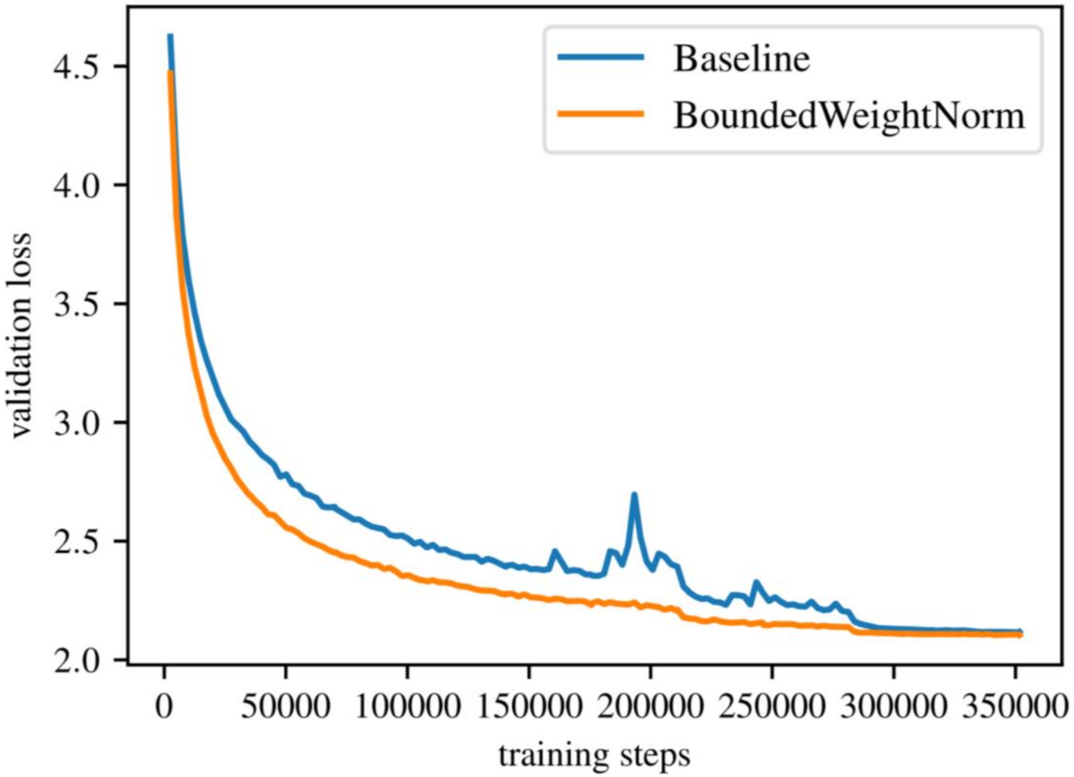
Where ρ is a constant determined from chosen initialization

Improving weight-norm

This can help to make weight-norm work for large-scale models



Resnet 50, ImageNet



En-De translation, LSTM

Replacing Batch-norm – switching norms

- In fact, Batch-normalization is just a scaled L^2 normalization
- We can use other, more numerically stable norms:

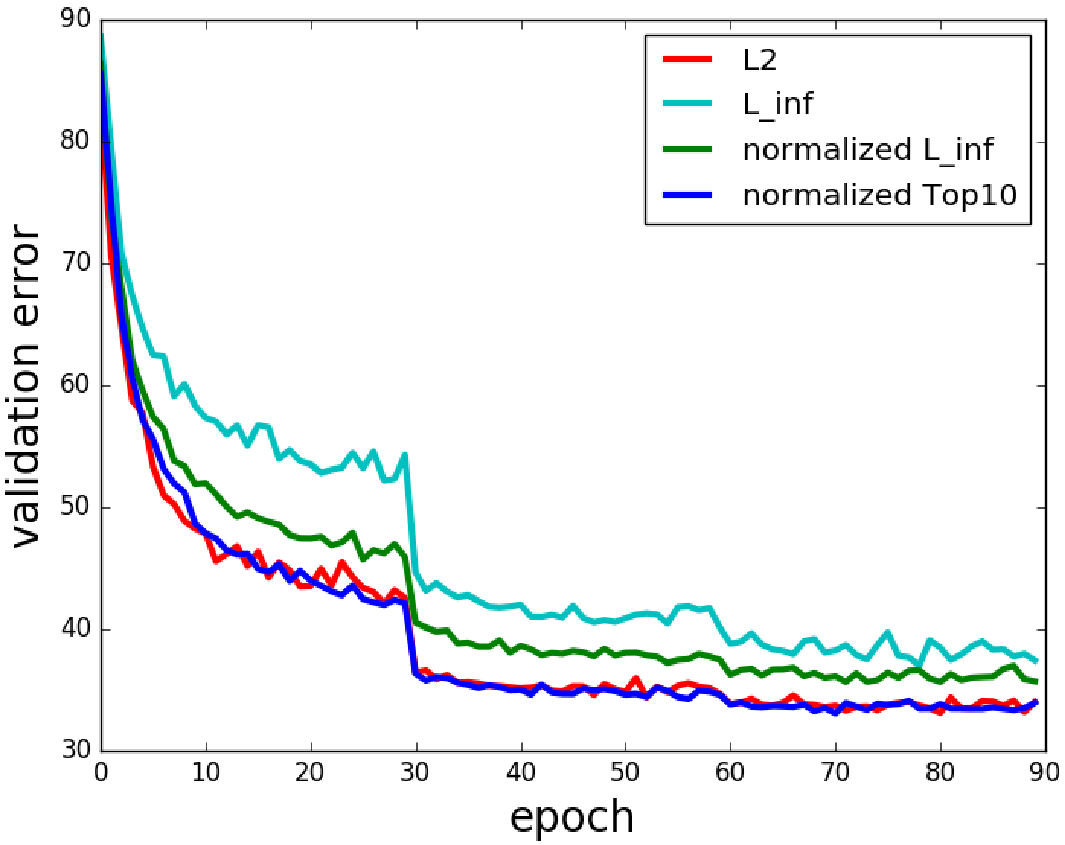
$$\|x\|_1 = \frac{1}{N}\sum|x| \qquad\qquad \|x\|_\infty = \max\{|x|\}$$

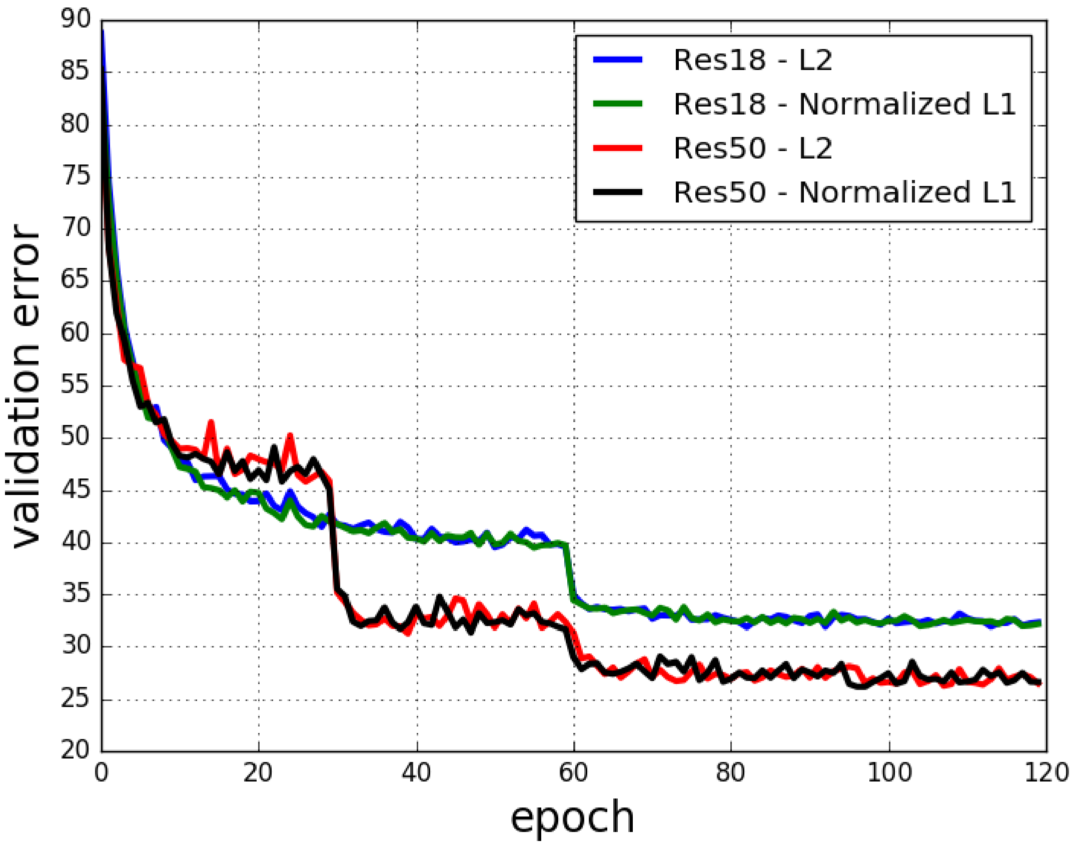
We use additional scaling constants so that the norm will behave similarly to L^2

We can also use a "$Top(k)$" that bridges these two norms

$$\|x\|_{Top(k)} = \frac{1}{K}\sum \max_{1..k}\{|x|\}$$

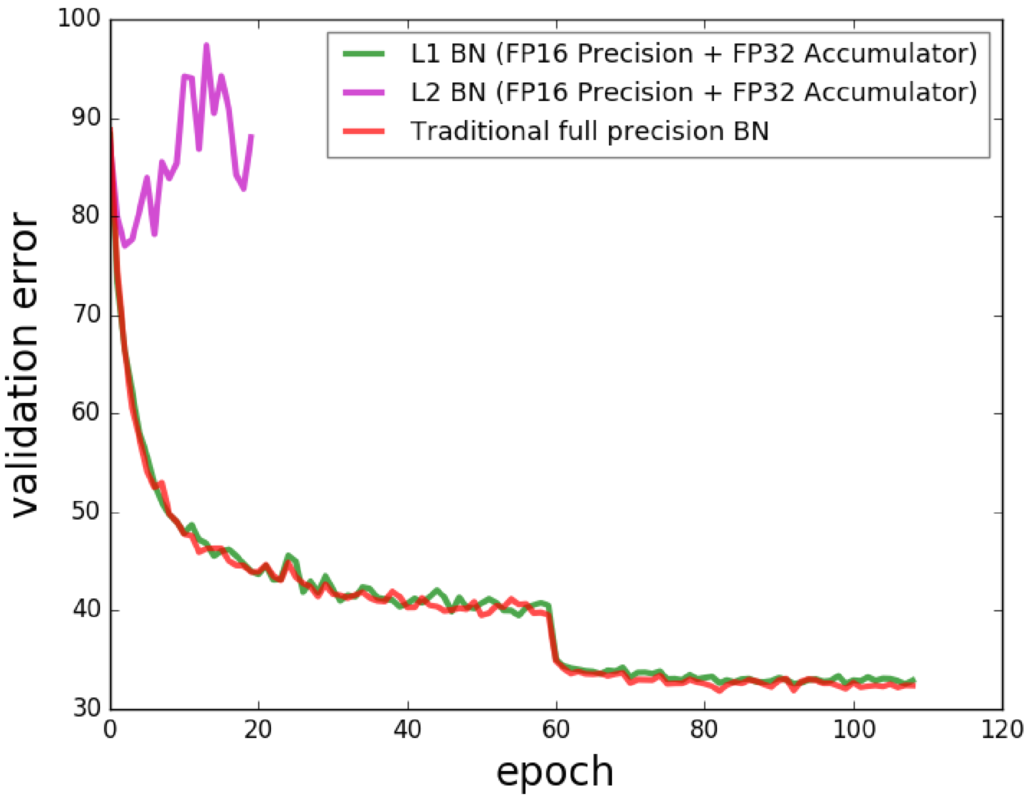
$L^1, L^\infty, L^{Top(10)}$ Batch-norm (Imagenet, Resnet)

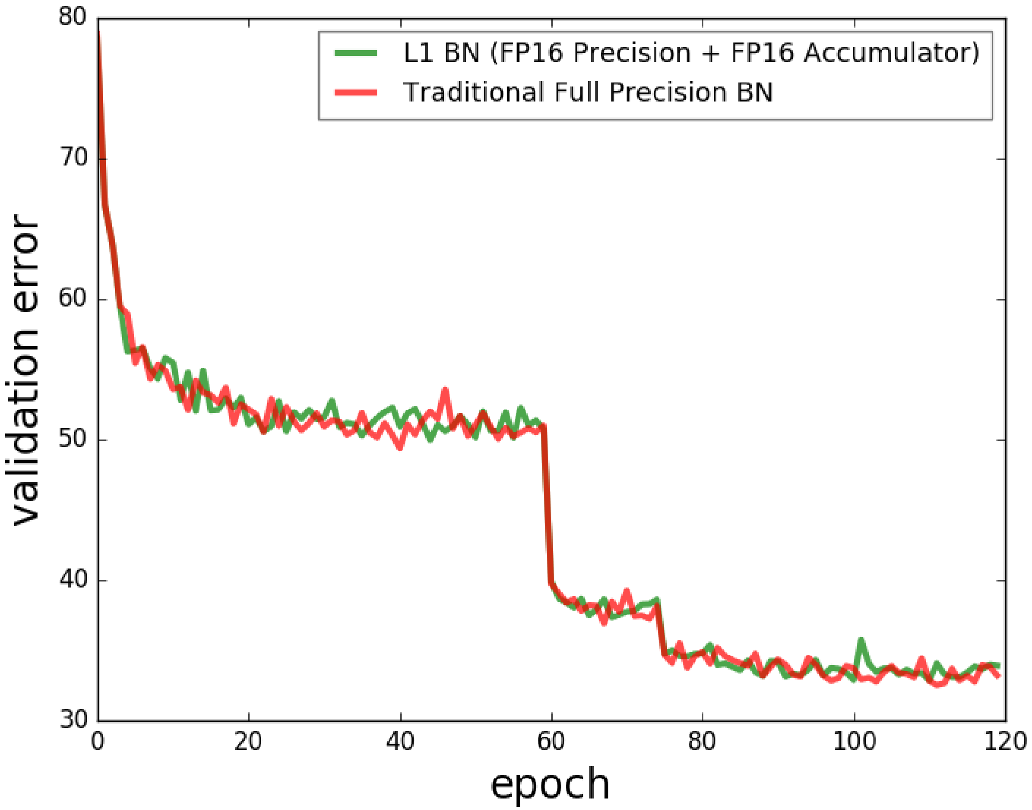




Low precision batch-norm

Using L^1 batch-norm alleviates some of the low-precision difficulties of batch-norm. We can now train without issues on FP16:





4) Acknowledging blindspots of CNNs

"On the Blindspots of Convolutional Networks"

Elad Hoffer*, Shai Fine*, Daniel Soudry

CNN blindspots

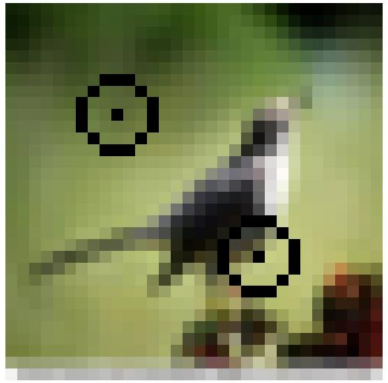
- Choosing a CNN model for a particular class enforces a strong (and useful) prior on the data:
  - Information exhibits local properties (stationarity)
  - Not too local…

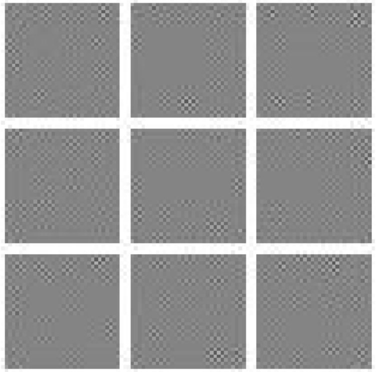
This means that on some tasks, these models will perform worst than much simpler approaches (“No free lunch theorem”)

CNN blindspots

We use “truth-revealing” signals in images to show several of these blindspots:

1. Label is encoded within a specific pixel
2. Label is encoded over several specific, distant, pixels
3. Label is encoded by location of specific pixels (value may change)
4. Label is encoded globally using image-sized orthogonal vectors
5. Label is encoded globally using the image mean-value





CNN blindspots

- While CNNs (Resnet) completely miss these signals (same accuracy as original images), “classic” ML models find them easily

Truth Signal	ConvNet	Best Model	
One Pixel	84.9%	100%	Dec. Tree
Pattern Pixels	84.9%	100%	Dec. Tree
Random Pixel	84.9%	100%	Shallow NN
Multiple Locations	84.9%	98.4%	QDA
Noise	84.9%	100%	Perceptron
Mean	84.9%	100%	Shallow NN

- Similar results appear for text-based CNNs

Know your model

- Our main message – know your model
- Choosing a CNN as your model makes strong assumption about your data
  - High dimensional information (global signal) may be lost
  - Low dimensional information (very local) may be lost
  - Relational data (location) may be lost

- Convolutional networks are not always the right solution

Summary

- Large batch training $\nRightarrow$ generalization decrease
- Validation loss increases $\neq$ overfitting occurs
- Validation error monotonically decreases: no early stopping
- Linear classification layers have marginal effect on accuracy
- Batch-norm success might have a different reason than believed. Alternatives may prove better.
- Be aware of CNNs inductive bias – "No Free Lunch"

Thank you for your time! Questions?

For more information, visit my page at:
www.DeepLearning.co.il