FORNAX

Improving deep learning models with bag of tricks

Krzysztof Kolasiński, 12.12.2018

WWW.FORNAX.AI

Disclaimer:

- weakly coupled topics, so may be a little chaotic,
- short summaries of interesting problems,
- my random thoughts on those problems,

A rough plan of this presentation:

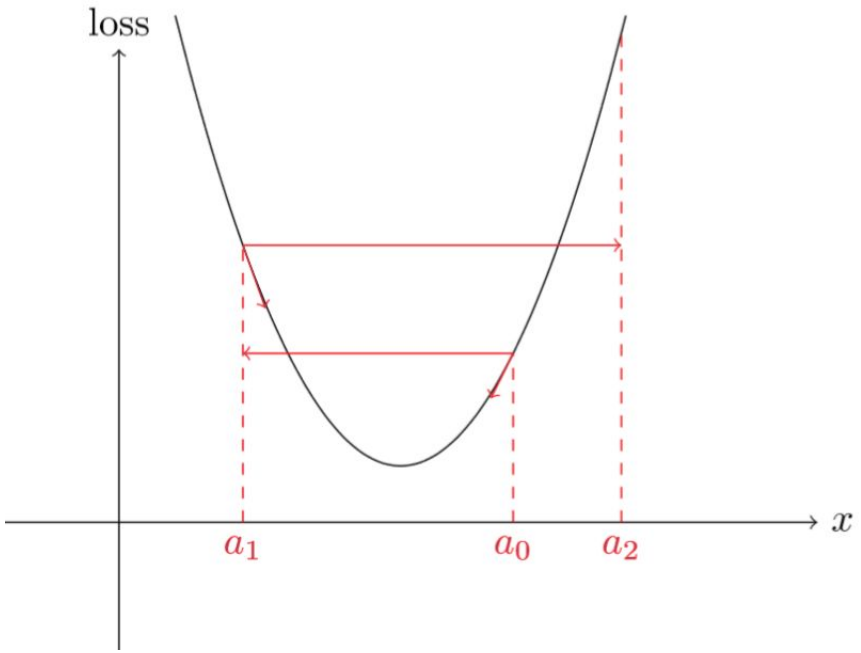
- learning schedulers, batch size
- optimizers
- ZeroInit
- mixup, label smoothing
- etc

Modifying learning rate

Selecting better learning rate

How to guess initial value for the learning rate?

- large learning rate: risk of weight explosion
- too small values: we will wait very long before we reach convergence, we can also



Note:

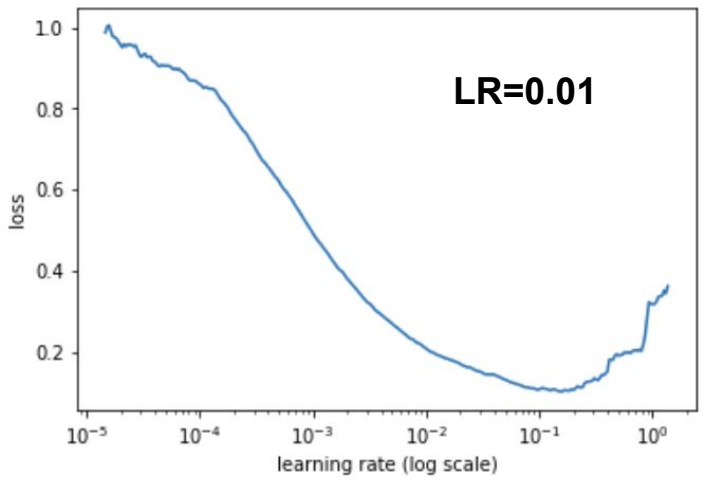
- small LR may lead to bigger overfitting,
- larger LR has regularizing property,
- looking for the best one is a reasonable problem

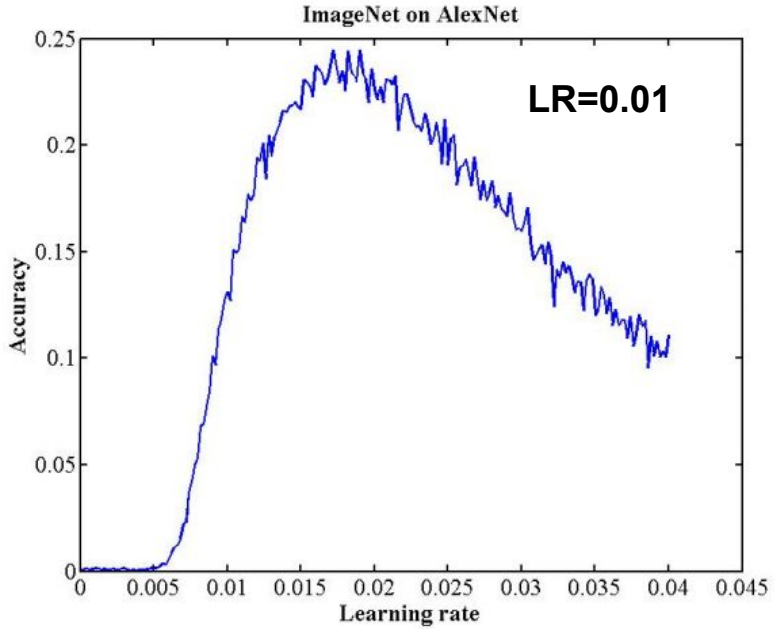
Sources:
https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html

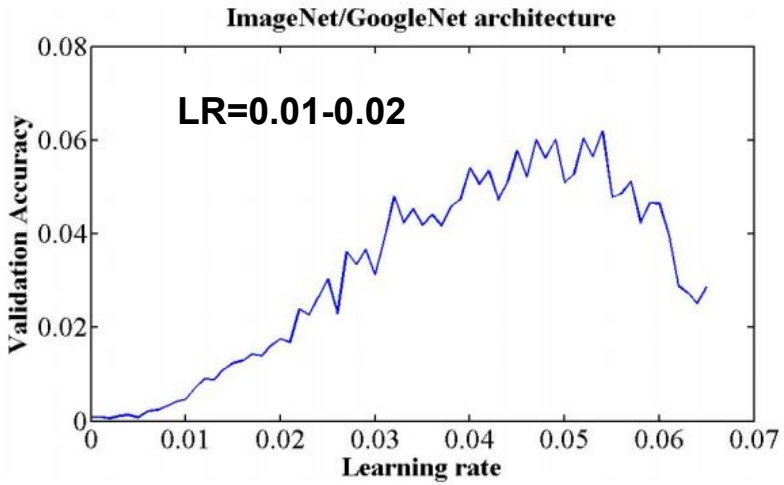
Overview: Cyclical Learning Rates for Training Neural Networks

A method for finding optimal LR proposed in CLR paper:

- before you actually start training create LR range test plot
- select the value of LR which leads to minimum (or maximum) in the plot
- select your optimal LR to be approx. one order smaller, or ½ - ⅓ as a rule of thumb







Sources:
https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html

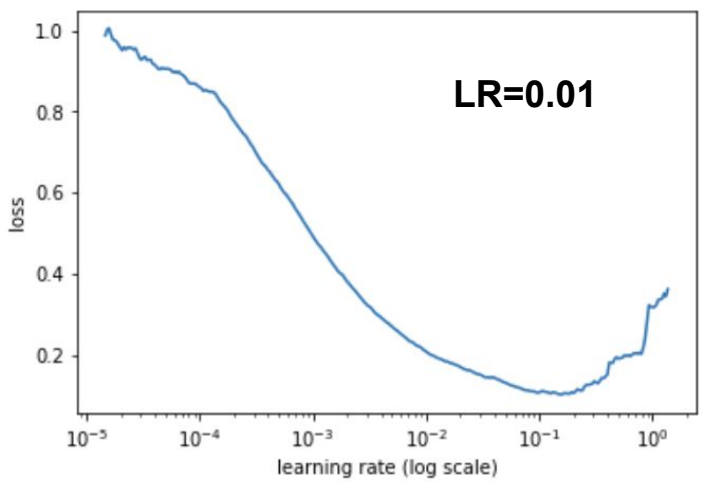
Implementation of LR test range is straight forward:

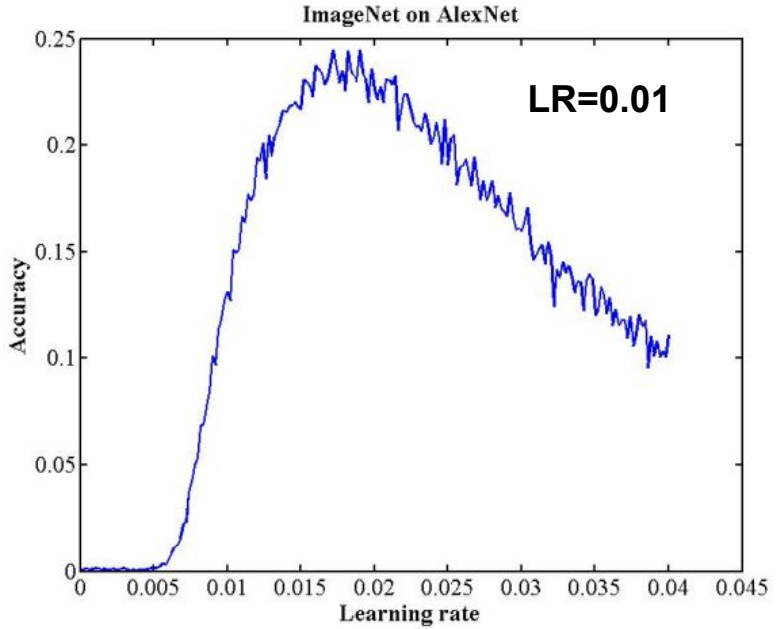
- select number of batches, initial lr and lr multiplier
- monitor loss with increasing LR

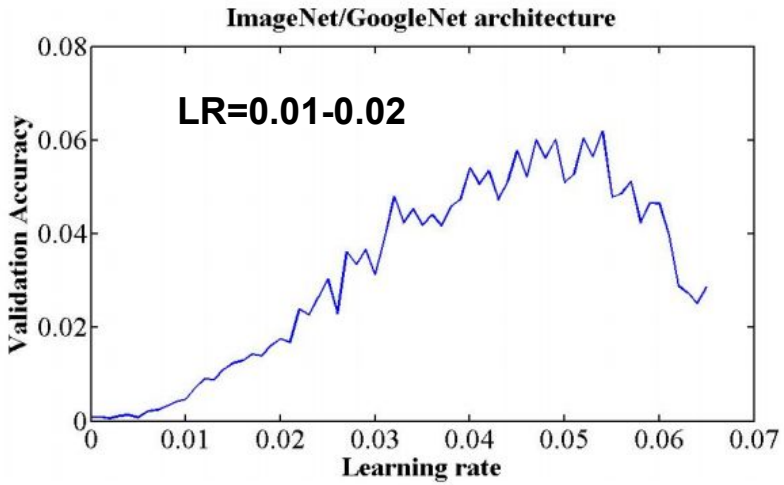
Overview: Cyclical Learning Rates for Training Neural Networks

Comments:

- it is worth to use this method to find initial value of LR when starting with some new project
- this method sounds reasonable and can be easily tested on your own problem







Sources:
https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html

Overview: Cyclical Learning Rates for Training Neural Networks

Discussion of the results:

- unfortunately, results are quite inconsistent in terms of final validation accuracy
- however, it may help to tune better initial LR or even achieve faster convergence
- remember, it doesn't mean you will have better final validation accuracy

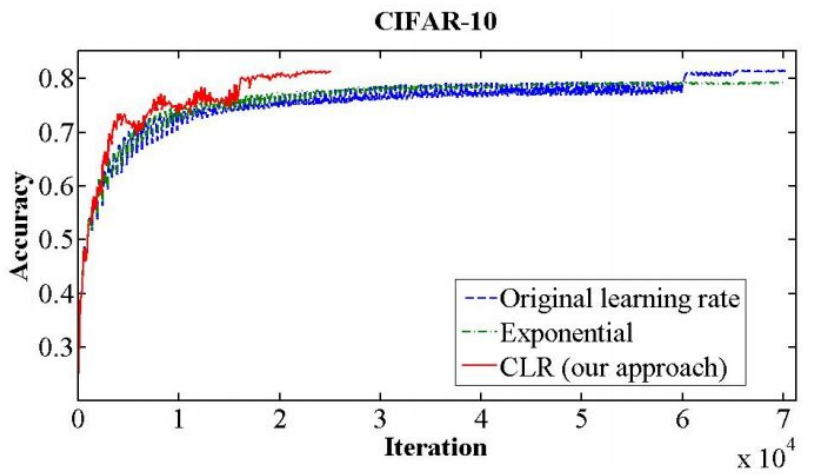


Figure 1. Classification accuracy while training CIFAR-10. The red curve shows the result of training with one of the new learning rate policies.

LR type/bounds	LR policy	Iterations	Accuracy (%)
Nesterov [19]	*fixed*	70,000	82.1
0.001 - 0.006	*triangular*	25,000	81.3
ADAM [16]	*fixed*	70,000	81.4
0.0005 - 0.002	*triangular*	25,000	79.8
	triangular	70,000	81.1
RMSprop [27]	*fixed*	70,000	75.2
0.0001 - 0.0003	*triangular*	25,000	72.8
	triangular	70,000	75.1
AdaGrad [5]	*fixed*	70,000	74.6
0.003 - 0.035	*triangular*	25,000	76.0
AdaDelta [29]	*fixed*	70,000	67.3
0.01 - 0.1	*triangular*	25,000	67.3

Table 3. Comparison of CLR with adaptive learning rate methods. The table shows accuracy results for the CIFAR-10 dataset on test data at the end of the training.

Sources:
https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html

Overview: Cyclical Learning Rates for Training Neural Networks

Cyclical Learning Rates - general overview of the method

- specify minimum and maximum learning rate boundaries,
- stepsize is the number of iteration to reach the half of the cycle
- use some policy to define the shape of the lr cycle - here example with triangular policy

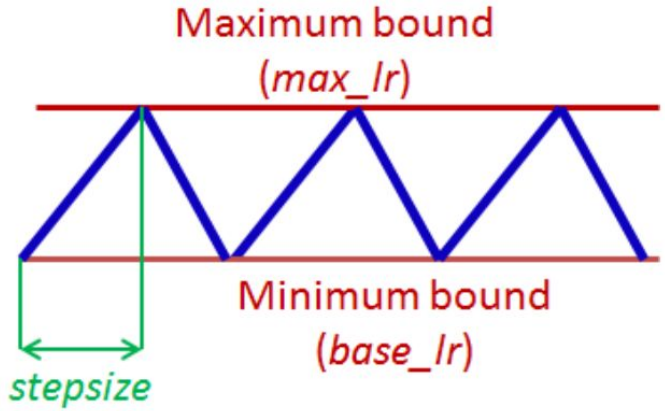
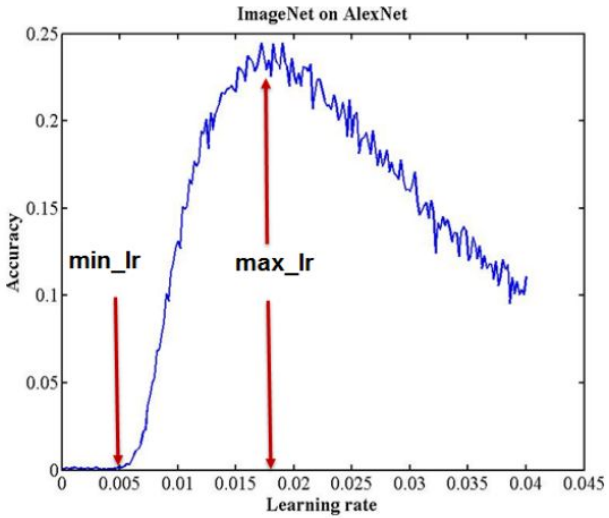


Figure 2. Triangular learning rate policy. The blue lines represent learning rate values changing between bounds. The input parameter $stepsize$ is the number of iterations in half a cycle.

Comments

- it seems that the shape of the policy does not change the final result significantly,
- the **base_lr** and **max_lr** can be deducted from the LR range test (see example fig),
- for properly tuned LR we can reach *super convergence* - a phenomena where network can reach convergence for extreme values of LR



(a) Typical learning rate range test result where there is a peak to indicate max_lr.

Overview: Cyclical Learning Rates for Training Neural Networks

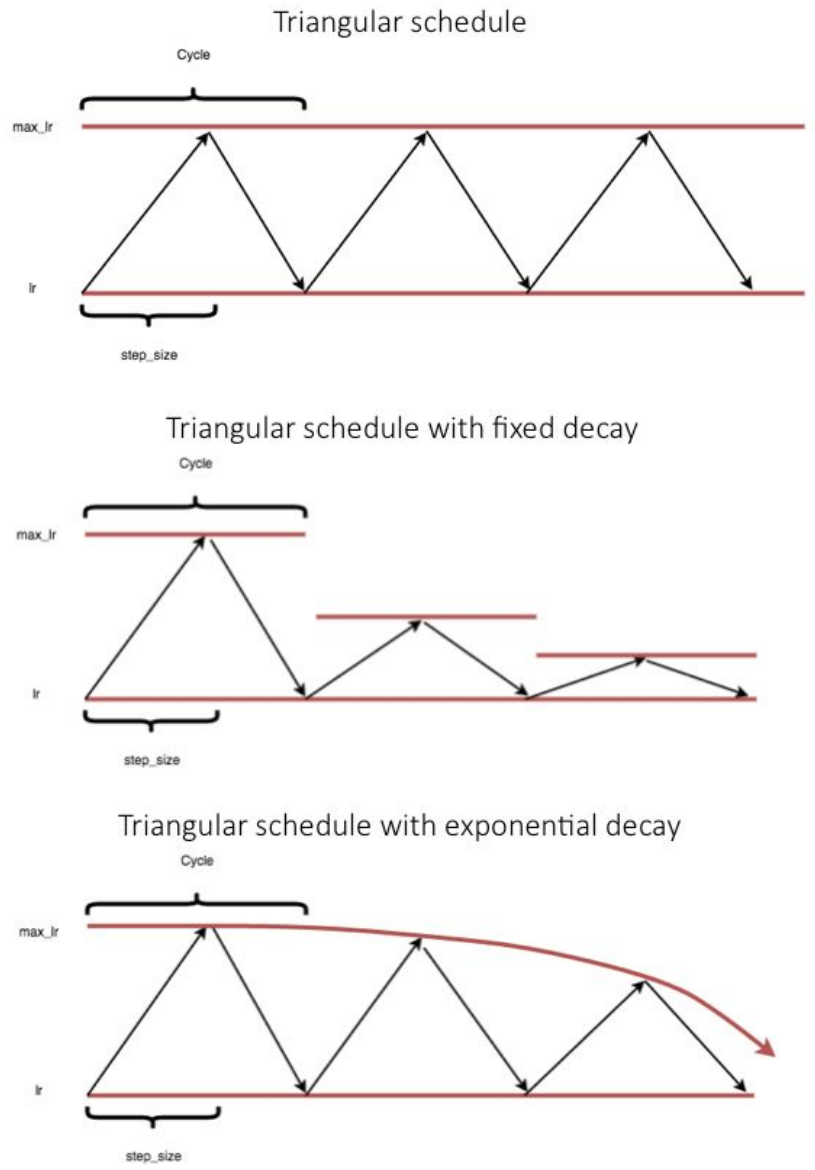
Cyclical Learning Rates - general overview of the method

- specify minimum and maximum learning rate boundaries,
- stepsize is the number of iteration to reach the half of the cycle
- use some policy to define the shape of the lr cycle - here example with triangular policy

Comments

- it seems that the shape of the policy does not change the final result significantly,
- the **base_lr** and **max_lr** can be deducted from the LR range test (see example fig),
- for properly tuned LR we can reach *super convergence* - a phenomena where network can reach convergence for extreme values of LR,
- more policies ...

Different policies:



Overview: SGDR: Stochastic Gradient Descent with Warm Restarts

SGDR (2017)

- In many contexts restarts in LR schedule seems to improve convergence
- Warm restarts have been already used in CLR paper
- It's another paper which shows the importance of LR scheduling

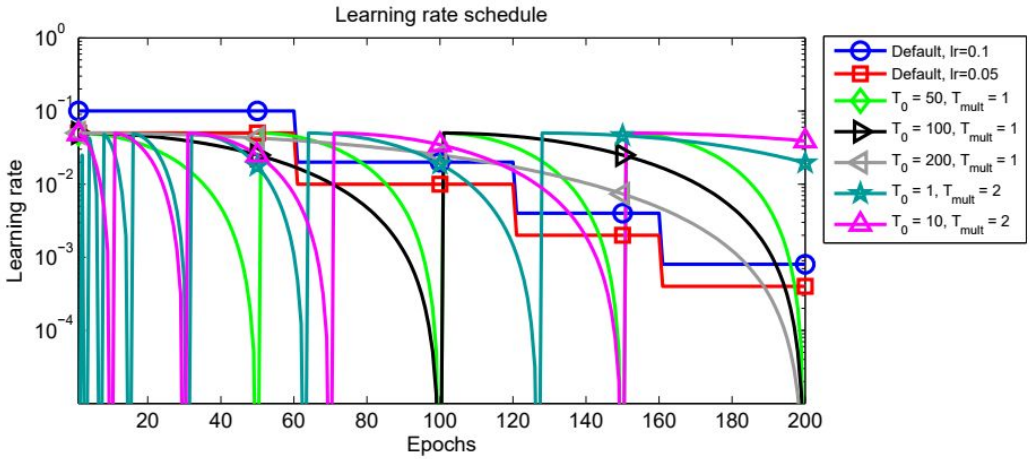
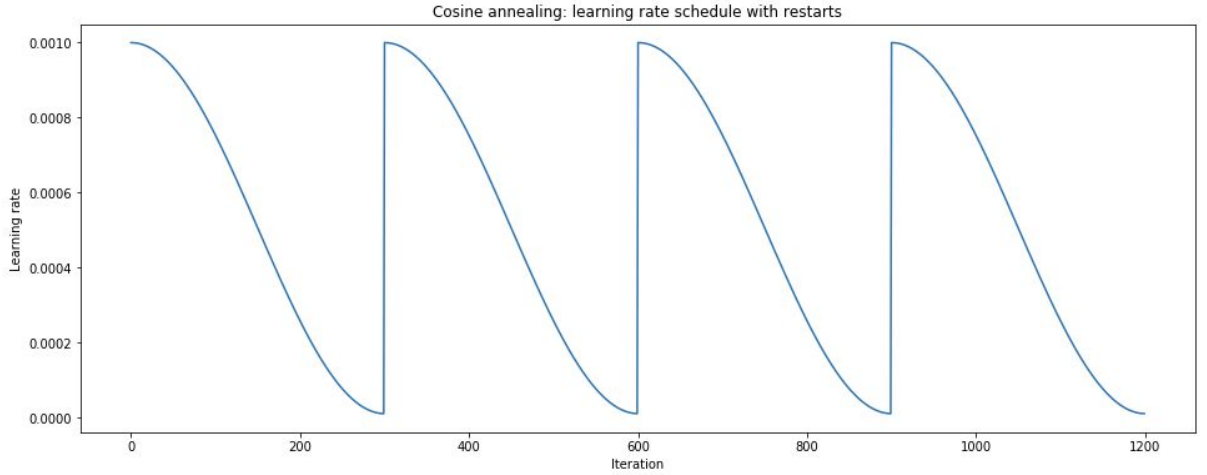


Figure 1: Alternative schedule schemes of learning rate η_t over batch index t: default schemes with $\eta_0 = 0.1$ (blue line) and $\eta_0 = 0.05$ (red line) as used by Zagoruyko & Komodakis (2016); warm restarts simulated every $T_0 = 50$ (green line), $T_0 = 100$ (black line) and $T_0 = 200$ (grey line) epochs with η_t decaying during i-th run from $\eta^i_{max} = 0.05$ to $\eta^i_{min} = 0$ according to eq. (5); warm restarts starting from epoch $T_0 = 1$ (dark green line) and $T_0 = 10$ (magenta line) with doubling ($T_{mult} = 2$) periods T_i at every new warm restart.

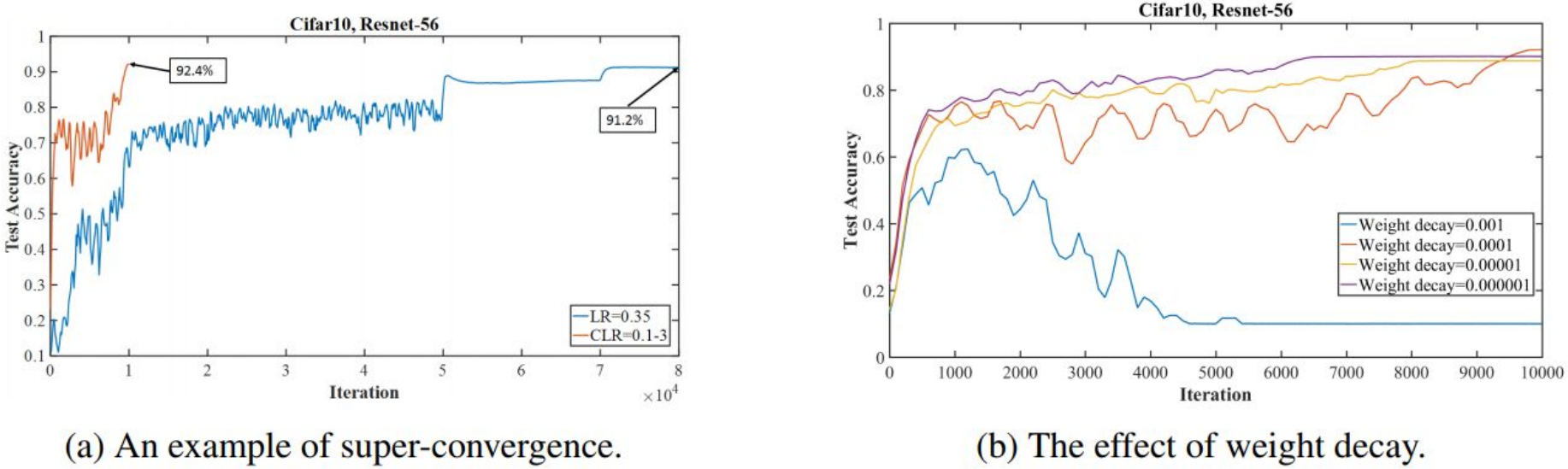


Overview: Super Convergence

Super Convergence (2018) - a phenomena where network can reach convergence for extreme values of LR. Extreme means by order of magnitude bigger than previously reported.

Comments:

- larger values of LR require smaller values of other regularizations
- this is because big LR already has some regularizing properties
- *"the amount of regularization must be balanced for each dataset and architecture"*



(a) An example of super-convergence.

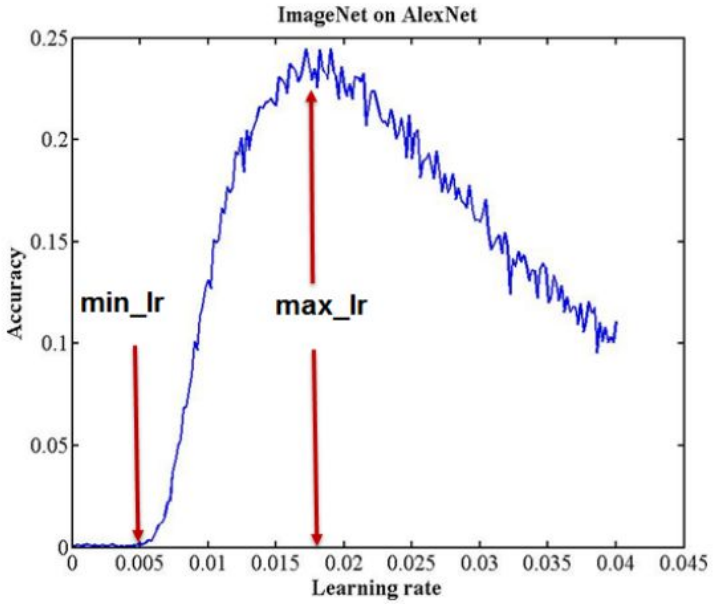
(b) The effect of weight decay.

Figure 5: Faster training is possible by allowing the learning rates to become large. Other regularization methods must be reduced to compensate for the regularization effects of large learning rates. Practitioners must strive for an optimal balance of regularization.

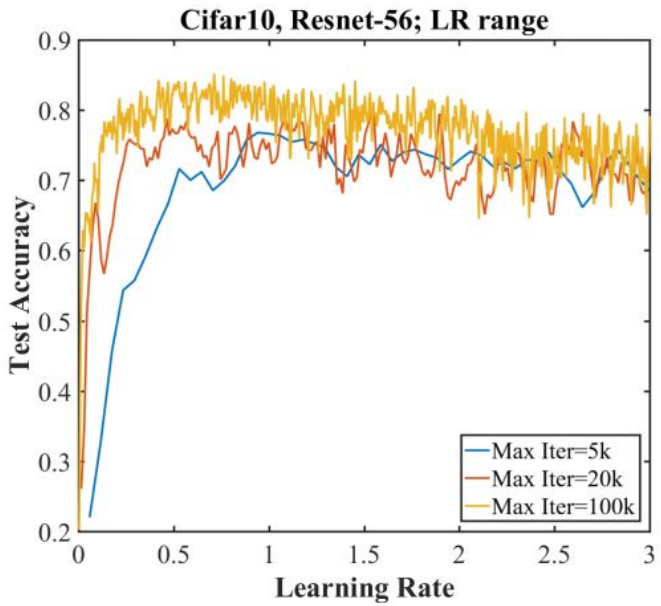
Overview: Super Convergence

Super Convergence (2018)

- use LR range test to check whether SC is possible for your architecture



(a) Typical learning rate range test result where there is a peak to indicate max_lr.



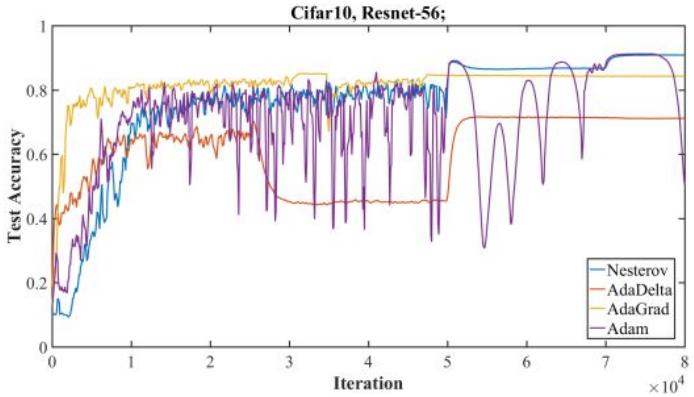
(b) Learning rate range test result with the Resnet-56 architecture on Cifar-10.

Figure 2: Comparison of learning rate range test results.

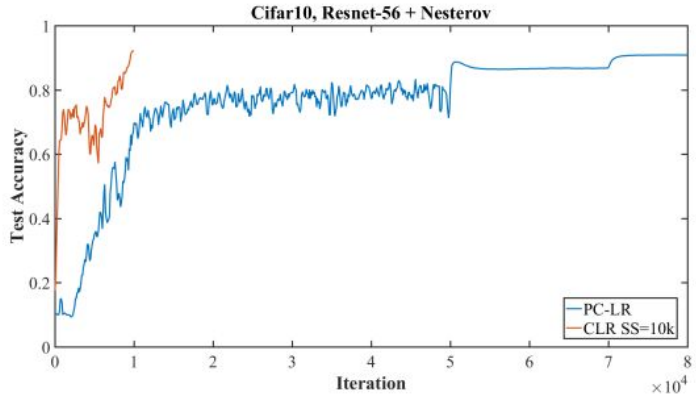
Overview: Super Convergence

Super Convergence (2018)

- SC was observed for limited number of optimizers e.g. Nesterov Momentum but not for Adam
- It may also depend on the architecture



(a) Comparison of test accuracies with various adaptive learning rate methods.



(b) Comparison of test accuracies With Nesterov method.

Figure 9: Comparisons for Cifar-10, Resnet-56 of super-convergence to piecewise constant training regime.

Modifying the size of the mini-batch

Overview: Training with extremely large batch size

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour (2018)

A common knowledge:

- larger batch size more accurate gradient directions - less variance,
- larger mini-batches are computationally more effective,
- however, current research suggests that **smaller minibatches yield better models** - in terms of generalization error e.g. with simple **GD you can stack at local minima**, SGD helps to escape from them
- on the other hand - larger mini-batches allow for taking larger steps

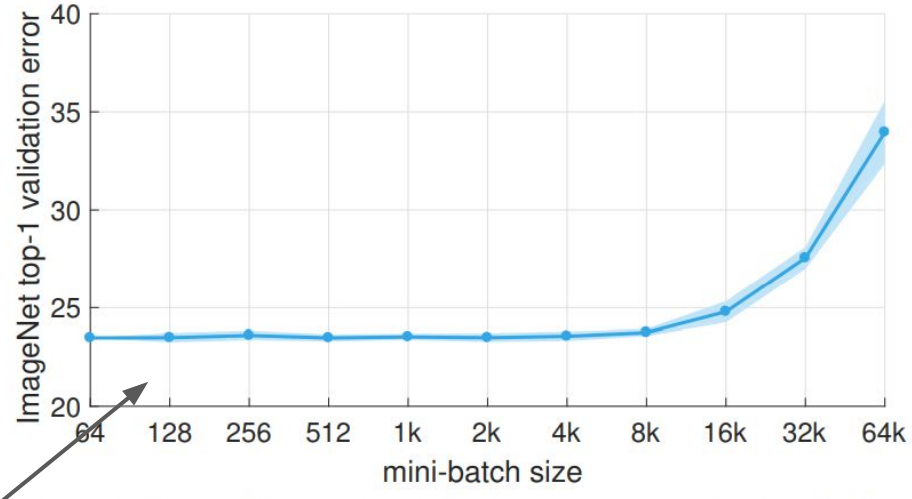


Figure 1. **ImageNet top-1 validation error *vs.* minibatch size.** Error range of plus/minus *two* standard deviations is shown. We present a simple and general technique for scaling distributed synchronous SGD to minibatches of up to 8k images *while maintaining the top-1 error of small minibatch training*. For all minibatch

Same Validation error up to 8k batch size

Overview: Training with extremely large batch size

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour (2018)

Discussion of the paper

- *probably* first result which showed that we can achieve good validation accuracy for extremely large mini-batches

- Their solution:

    - **linear scaling rule** - scale your learning rate with mini-batch,
    - **warmup rule** - use lower learning rates at initial stage of training

- With this rule their obtain: "*training/validation error curves that closely match the small minibatch baseline*"

Overview: Training with extremely large batch size

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour (2018)

Motivations for the **linear scaling rule**

Minibatch Stochastic Gradient Descent [32], usually referred to as simply as SGD in recent literature even though it operates on minibatches, performs the following update:

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t). \qquad (2)$$

Here $\mathcal{B}$ is a *minibatch* sampled from X and $n = |\mathcal{B}|$ is the *minibatch size*, η is the *learning rate*, and t is the iteration

- The **amplitude of the noise** of this gradient can be related with batch size B and the training set size N by [ref]

$$g = \lambda_{lr} \left(\frac{N}{B} - 1 \right)$$

- **Quick check:** when B=N there is no noise,
- This noise helps to escape from local minima and improve exploration, so we want to keep it constant, when changing batch size. For N >> B we have:

$$g = \lambda_{lr} \frac{N}{B} = \lambda'_{lr} \frac{N}{B'} \Rightarrow \lambda'_{lr} = \frac{B'}{B} \lambda_{lr}$$

- k-times bigger batch size requires k-times bigger LR, to keep noise amplitude constant

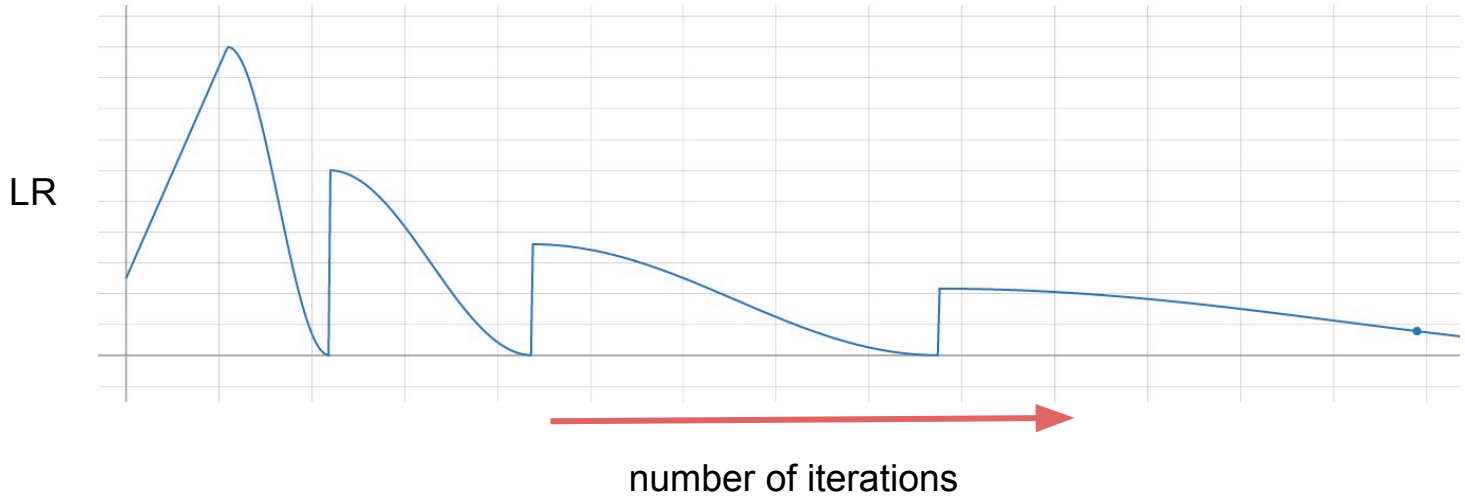
Overview: Training with extremely large batch size

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour (2018)

Motivation for the **warmup**

- At the beginning of the training gradients are large and training with larger LR will eventually break the training - weight explosion.

Example LR strategy with warmup:



Overview: Training with extremely large batch size

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour (2018)

Importance of the warmup on the results:

Baseline. Under these settings, we establish a ResNet-50 baseline using $k = 8$ (8 GPUs in one server) and $n = 32$ images per worker (minibatch size of $kn = 256$), as in [16]. Our baseline has a top-1 validation error of 23.60% ±0.12. As a reference, ResNet-50 from `fb.resnet.torch` [12] has 24.01% error, and that of the original ResNet paper [16] has 24.7% under weaker data augmentation.

	k	n	kn	η	top-1 error (%)
baseline (single server)	8	32	256	0.1	23.60 ±0.12
no warmup, Figure 2a	256	32	8k	3.2	24.84 ±0.37
constant warmup, Figure 2b	256	32	8k	3.2	25.88 ±0.56
gradual warmup, Figure 2c	256	32	8k	3.2	23.74 ±0.09

Table 1. **Validation error on ImageNet using ResNet-50** (mean and std computed over 5 trials). We compare the small minibatch model (kn=256) with large minibatch models (kn=8k) with various warmup strategies. Observe that the top-1 validation error for small and large minibatch training (with gradual warmup) is quite close: 23.60% ±0.12 *vs.* 23.74% ±0.09, respectively.

Overview: Training with extremely large batch size

Training error in function of batch size:

kn	η	top-1 error (%)
256	0.05	23.92 ±0.10
256	0.10	23.60 ±0.12
256	0.20	23.68 ±0.09
8k	$0.05 \cdot 32$	24.27 ±0.08
8k	$0.10 \cdot 32$	23.74 ±0.09
8k	$0.20 \cdot 32$	24.05 ±0.18
8k	0.10	41.67 ±0.10
8k	$0.10 \cdot \sqrt{32}$	26.22 ±0.03

(a) **Comparison of learning rate scaling rules.** A reference learning rate of $\eta = 0.1$ works best for $kn = 256$ (23.68% error). The linear scaling rule suggests $\eta = 0.1 \cdot 32$ when $kn = 8$k, which again gives best performance (23.74% error). Other ways of scaling η give worse results.

Scaling with larger batch size::

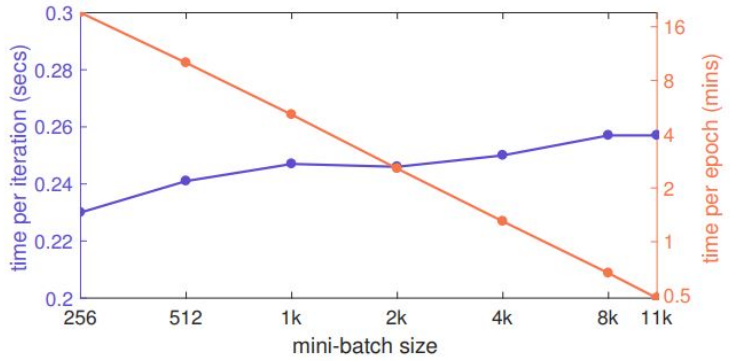


Figure 7. **Distributed synchronous SGD timing.** Time per iteration (seconds) and time per ImageNet epoch (minutes) for training with different minibatch sizes. The baseline ($kn = 256$) uses 8 GPUs in a single server , while all other training runs distribute training over ($kn/256$) server. With 352 GPUs (44 servers) our implementation completes one pass over all ~1.28 million ImageNet training images in about 30 seconds.

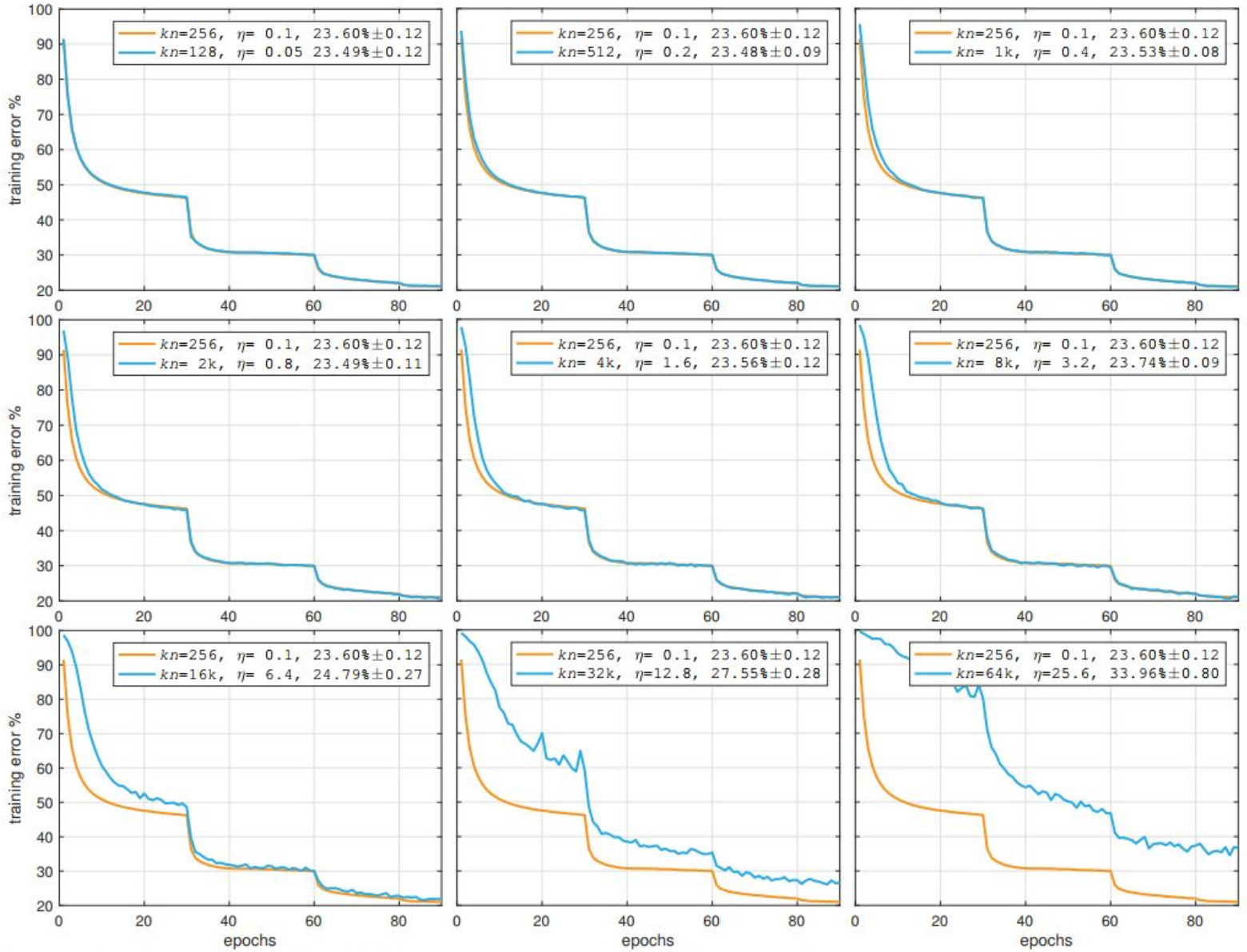


Figure 3. **Training error *vs.* minibatch size.** Training error curves for the 256 minibatch baseline and larger minibatches using gradual warmup and the linear scaling rule. Note how the training curves closely match the baseline (aside from the warmup period) up through 8k minibatches. *Validation* error (mean±std of 5 runs) is shown in the legend, along with minibatch size kn and reference learning rate η.

Don't Decay the Learning Rate, Increase the Batch Size (2018)

Discussion

- This time ImageNet in 30 min with batch size up to 65536
- Instead of decreasing learning rate by e.g. 2, increase batch size by the same amount
- This will decrease gradient noise roughly by factor of 2
- The results show the equivalence between those approaches

$$g = \lambda_{lr}\left(\frac{N}{B} - 1\right)$$

- **blue**: decrease LR by factor 5 after some number of steps
- **green**: increase batch size by factor 5, then decrease LR by factor 5. Repeat.
- **red**: increase batch size by factor 5 after some number of steps.

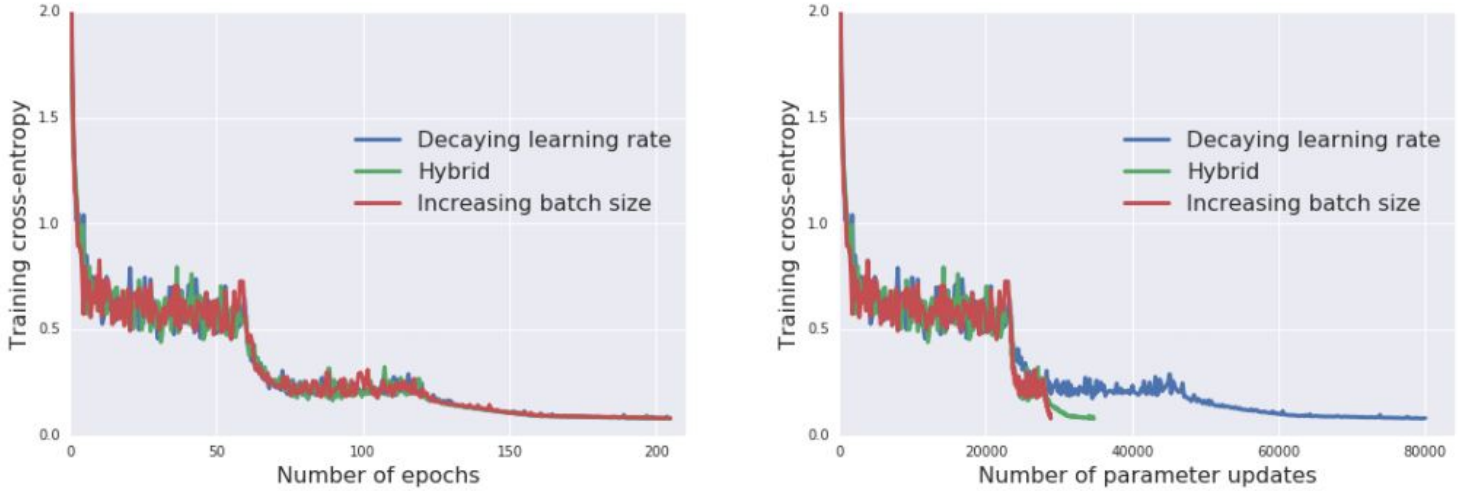


Figure 2: Wide ResNet on CIFAR10. Training set cross-entropy, evaluated as a function of the number of training epochs (a), or the number of parameter updates (b). The three learning curves are identical, but increasing the batch size reduces the number of parameter updates required.

Don't Decay the Learning Rate, Increase the Batch Size (2018)

Discussion

- This time ImageNet in 30 min with batch size up to 65536
- Instead of decreasing learning rate by e.g. 2, increase batch size by the same amount
- This will decrease gradient noise roughly by factor of 2
- The results show the equivalence between those approaches

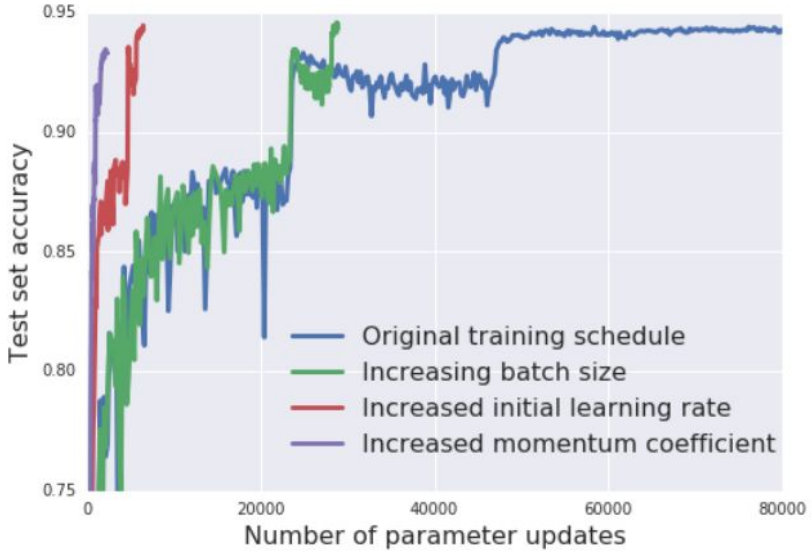


Figure 5: Wide ResNet on CIFAR10. Test accuracy as a function of the number of parameter updates. "Increasing batch size" replaces learning rate decay by batch size increases. "Increased initial learning rate" additionally increases the initial learning rate from 0.1 to 0.5. Finally "Increased momentum coefficient" also increases the momentum coefficient from 0.9 to 0.98.

Some records

- ImageNet ~75% accuracy in 35/100 epochs with batch size up to: 16384/131072 [NVIDIA Nov 2018]
- ImageNet ~76% accuracy in 60 minutes with batch size up to 8K [Facebook 2018]
- ImageNet ~75% accuracy in 15 minutes with batch size up to 32K [Preferred Networks Inc. Nov 2017]
- ImageNet ~75% accuracy in 14 minutes (64 epochs) with batch size 32K [UC Berkeley et. al. 2018]
- ImageNet ~75% accuracy in 224 seconds, batch size ~32K [Sony Oct 2018]

SOTA is around 81% now …

Modifying optimizer and weight decay

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: a confusion in terminology

- Authors show difference between L2 and weight decay when using optimizers with adaptive learning rate e.g. Adam
- According to authors (and some other people) we define **L2 regularization** as

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cost}}\left(\mathbf{x}_{\text{data}}, \mathbf{y}_{\text{labels}}; \mathbf{W}^{(l)}\right) + \frac{\beta}{2}\sum_{l,i}\left[W_i^{(l)}\right]^2$$

l - layer number
i - i-th element of **W** vector

- However, **weight decay** is defined as explicit exponential decay of parameters after each gradient step (Note: it's not an official definition)

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \beta\mathbf{W}^{(l)} = (1-\beta)\,\mathbf{W}^{(l)}$$

after each step we decrease W by 1-beta factor

- Both are equivalent when we use simple SGD optimizer. This is how it is commonly defined in terms of SGD

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \lambda\frac{\partial\mathcal{L}_{\text{cost}}}{\partial\mathbf{W}^{(l)}} - \lambda\beta\mathbf{W}^{(l)} = (1-\lambda\beta)\,\mathbf{W}^{(l)} - \lambda\frac{\partial\mathcal{L}_{\text{cost}}}{\partial\mathbf{W}^{(l)}}$$

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: a confusion in terminology

- This is quite simple thing, really?

stackoverflow.c

Weight decay vs L2 regularization - Quora
https://algorithmsdatascience.quora.com/Weight-decay-vs-L2-regularization

Why is L2 regularization incorrectly called weight decay in the ...
https://www.quora.com/Why-is-L2-regularization-incorrectly-called-weight-decay-in-th...
May 12, 2018 - Why is **L2 regularization** incorrectly called **weight decay** in the Deep ... While decay can be done in ways different from adding an **L2** term to the ...

neural networks - What is the weight decay loss? - Cross Validated
https://stats.stackexchange.com/questions/273189/what-is-the-weight-decay-loss
2 answers
Apr 13, 2017 - **Weight decay** specifies **regularization** in the neural network. During training, a **regularization** term is added to the network's loss to compute the ...

[D] Weight decay vs. L2 regularization : MachineLearning - Reddit
https://www.reddit.com/r/MachineLearning/.../d_weight_decay_vs_l2_regularization/
Apr 27, 2018 - 4 posts - 3 authors
the **L2** parameter norm penalty commonly known as **weight decay**" on ... When he cites "fancy solvers" he is only criticizing that **regularization** ...

Under review as a conference paper at ICLR 2019

THREE MECHANISMS OF WEIGHT DECAY REGULARIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Weight decay is one of the standard tricks in the neural network toolbox, but the reasons for its regularization effect are poorly understood, and recent results have cast doubt on the traditional interpretation in terms of L_2 regularization. Literal weight decay has been shown to outperform L_2 regularization for optimizers for which they differ. We empirically investigate weight decay for three optimization algorithms (SGD, Adam, and K-FAC) and a variety of network architectures. We identify three distinct mechanisms by which weight decay exerts a regularization effect, depending on the particular optimization algorithm and architecture: (1) increasing the effective learning rate, (2) approximately regularizing the input-output Jacobian norm, and (3) reducing the effective damping coefficient for second-order optimization. Our results provide insight into how to improve the regularization of neural networks.

Recent work on Weight decay (2019)

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: weight decay (WD) and overfitting

- WD may improve generalization and final accuracy of the network,
- WD decreases overfitting in two ways
  - it forces weights to be small, it's a form of a soft constraint on the weights. In that sense network cannot learn arbitrary weights which would lead to zero training error
  - in stationary state (assuming such situation) you cannot have zero training error when using WD:

$$\mathbf{W}^{n+1} = (1 - \lambda\beta)\,\mathbf{W}^{n} - \lambda \frac{\partial \mathcal{L}^{n}_{\text{cost}}}{\partial \mathbf{W}^{n}}$$

n - denotes training step, n-th batch

in stationary state weights should not change (*this is a simplified picture, and stationary state should be estimated in statistical sense, additionally such stationary state may even not exist*)

$$\mathbf{W}^{n} = (1 - \lambda\beta)\,\mathbf{W}^{n} - \lambda \frac{\partial \mathcal{L}^{n}_{\text{cost}}}{\partial \mathbf{W}^{n}} \implies \frac{\partial \mathcal{L}^{n}_{\text{cost}}}{\partial \mathbf{W}^{n}} = -\beta \mathbf{W}^{n}$$

gradient cannot vanish during training, which will enhance exploration, however it may also cause that we will escape from the best minimum :(

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: decoupling WD from gradient based update

- Authors propose following change in SGD+momentum optimizer

Algorithm 1 SGD with L_2 regularization and SGD with weight decay (SGDW), both with momentum

1: **given** initial learning rate $\alpha \in \mathbb{R}$, momentum factor $\beta_1 \in \mathbb{R}$, weight decay / L_2 regularization factor $w \in \mathbb{R}$
2: **initialize** time step $t \leftarrow 0$, parameter vector $\boldsymbol{x}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3: **repeat**
4: $t \leftarrow t + 1$
5: $\nabla f_t(\boldsymbol{x}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{x}_{t-1})$ ▷ select batch and return the corresponding gradient
6: $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{x}_{t-1}) + w\boldsymbol{x}_{t-1}$
7: $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, be used for warm restarts
8: $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + \eta_t \alpha \boldsymbol{g}_t$
9: $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{m}_t - \eta_t w \boldsymbol{x}_{t-1}$
10: **until** *stopping criterion is met*
11: **return** optimized parameters $\boldsymbol{x}_t$

Most frameworks add weight decay with L2 regularization i.e. by regularization term in the loss function:

$$f_t^{\text{reg}}(\boldsymbol{x}_t) = f_t(\boldsymbol{x}_t) + \frac{w}{2} \|\boldsymbol{x}_t\|_2^2 ,$$

Note: that **x** is a weight vector!

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: decoupling WD from gradient based update

- So… the authors propose following change in Adam optimizer and call it AdamW

Algorithm 2 Adam with L_2 regularization and Adam with weight decay (AdamW)

1: **given** $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$
2: **initialize** time step $t \leftarrow 0$, parameter vector $\boldsymbol{x}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3: **repeat**
4: $t \leftarrow t + 1$
5: $\nabla f_t(\boldsymbol{x}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{x}_{t-1})$ ▷ select batch and return the corresponding gradient
6: $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{x}_{t-1}) + w\boldsymbol{x}_{t-1}$
7: $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
8: $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
9: $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
10: $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
11: $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
12: $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + w\boldsymbol{x}_{t-1} \right)$
13: **until** *stopping criterion is met*
14: **return** optimized parameters $\boldsymbol{x}_t$

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \eta_t \alpha \frac{\beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t}{\sqrt{\beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2} + \epsilon},$$
$$\text{with } \boldsymbol{g}_t = \nabla f_t(\boldsymbol{x}_{t-1}) + w\boldsymbol{x}_{t-1},$$

decoupling WD

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: experimental results

- **WD and LR are coupled** when using SGD and Adam optimizers,
- this is not the case when using SGDW and AdamW - **easier hyperparameter selection**,
- they observed that AdamW yield **better generalization performance and smaller test error**

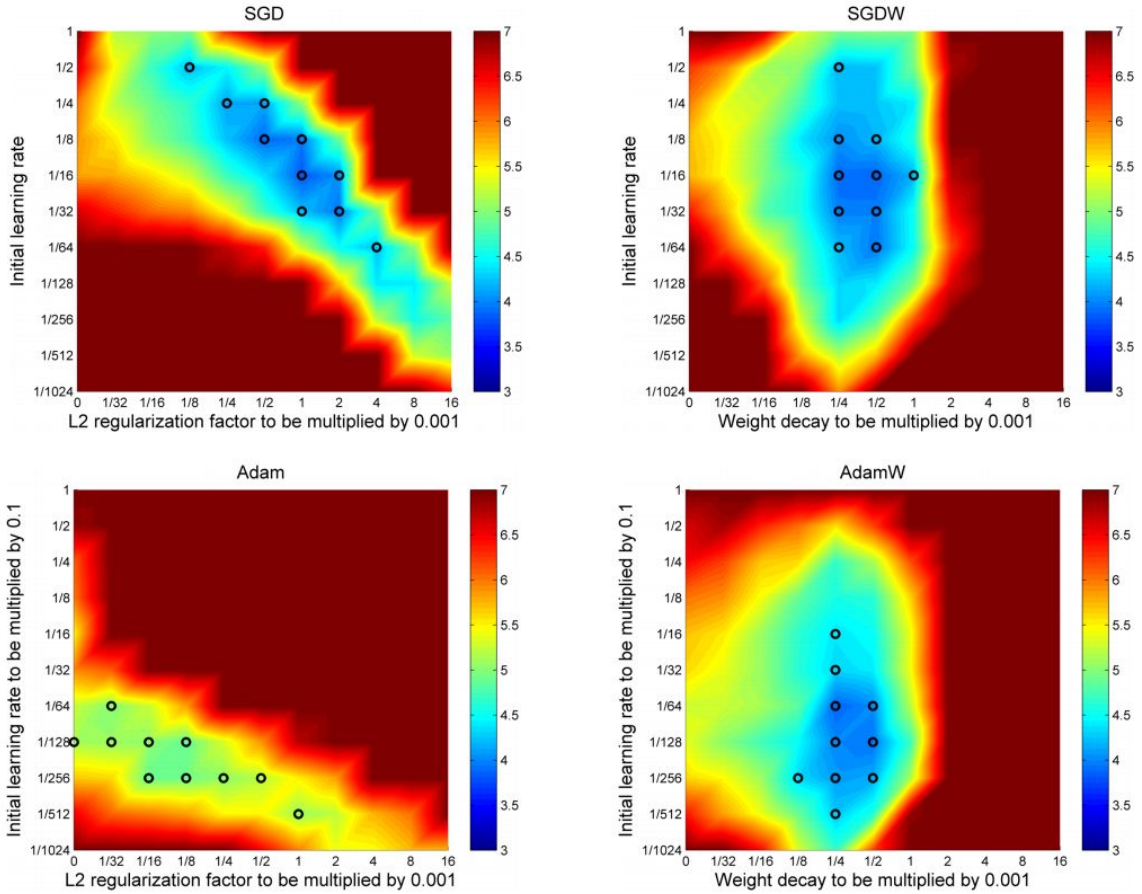
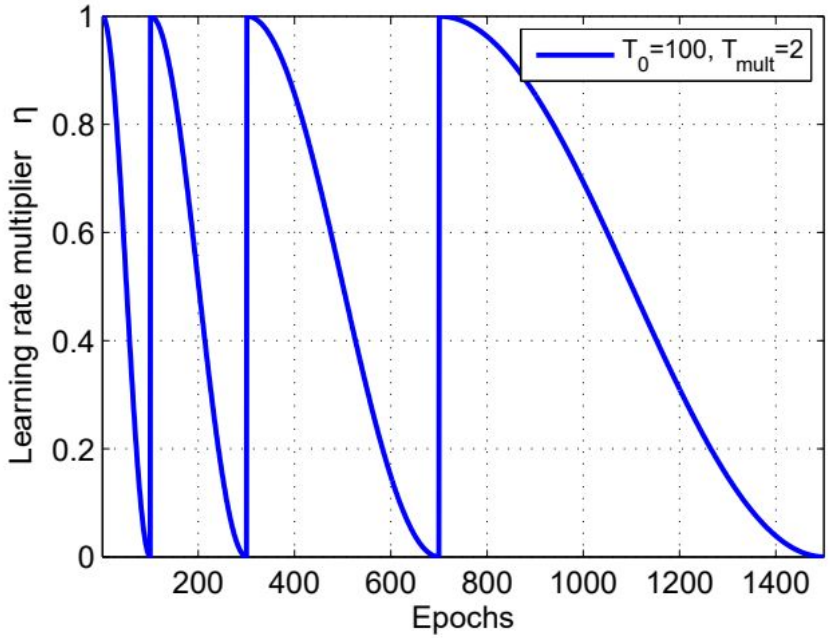


Figure 2. The Top-1 test error of a 26 2x64d ResNet on CIFAR-10 measured after 100 epochs. The proposed SGDW and AdamW (right column) have a more separable hyperparameter space.

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: experimental results

- Adam with Cosine Annealing and Warm Restarts (AdamWR): $\eta_t = 0.5 + 0.5\cos(\pi T_{cur}/T_i)$



SuppFigure 1. An example schedule of the learning rate multiplier as a function of epoch index. The first run is scheduled to converge at epoch $T_{i=0} = 100$, then the budget for the next run is doubled as $T_{i=1} = T_{i=0}T_{mult} = 200$, etc.

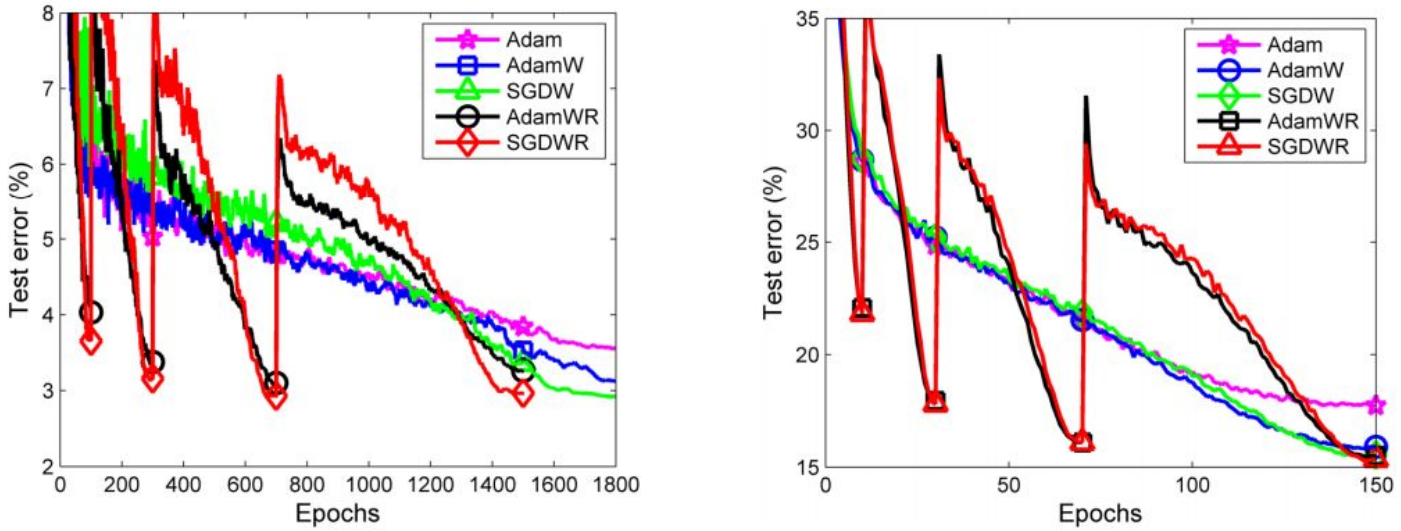


Figure 4. Top-1 test error on CIFAR-10 (left) and Top-5 test error on ImageNet32x32 (right).

- even better performance when using AdamWR, tested on CIFAR-10 and ImageNet 32x32

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: general observation

Note: WD/L2 is usually not the only thing we want to add to final loss, the problem of coupling any regularization to optimizer will also occur.

For example: **orthogonalization regularization**, when explicitly decoupled will lead different update when added to cost function.

$$\mathcal{L}_{\text{ortho}} = \lambda_o \left\| WW^T - 1 \right\|^2$$

Some references which use this regularization:

1. Neural Photo Editing with Introspective Adversarial Networks
2. Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNNs
3. Orthogonal Weight Normalization: Solution to Optimization over Multiple Dependent Stiefel Manifolds in Deep Neural Networks
4. On orthogonality and learning recurrent networks with long term dependencies
5. All You Need is Beyond a Good Init: Exploring Better Solution for Training Extremely Deep Convolutional Neural Networks with Orthonormality and Modulation

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: how to interpret momentum term ?

- What is a difference between 0.9, 0.95 and 0.99 and 0.999 ?
- Update with momentum:

$$\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + \eta_t \alpha \boldsymbol{g}_t$$
$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{m}_t \quad -\eta_t w \boldsymbol{x}_{t-1}$$

this is exponential moving average

- Momentum is an exponential moving average: $\mathbf{m}_{t+1} = \beta \mathbf{m}_t + \mathbf{G}_t$
- Let $\mathbf{m}_0 = \mathbf{0}$

$$\mathbf{m}_1 = \mathbf{G}_0$$
$$\mathbf{m}_2 = \beta \mathbf{m}_1 + \mathbf{G}_1 = \beta \mathbf{G}_0 + \mathbf{G}_1$$
$$\mathbf{m}_3 = \beta^2 \mathbf{G}_0 + \beta \mathbf{G}_1 + \mathbf{G}_2$$
$$\mathbf{m}_{t+1} = \sum_{t'=0}^{t-1} \beta^{t-t'-1} \mathbf{G}_{t'}$$

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: how to interpret momentum term ?

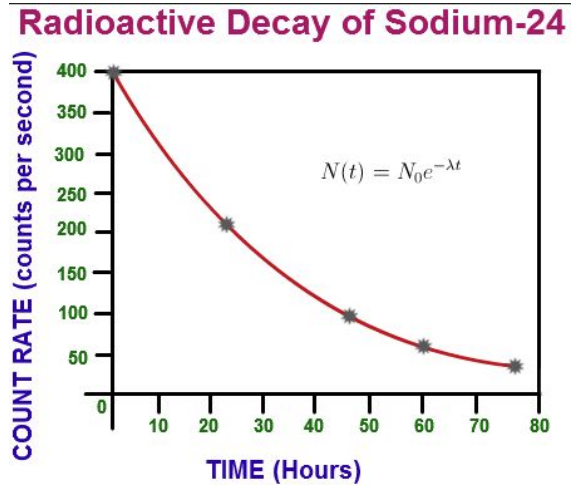
- The current value of gradient is an exponentially weighted sum of the previous values
- Let us change a little bit of notation, from:

$$\mathbf{m}_{t+1} = \beta^t \mathbf{G}_0 + \beta^{t-1} \mathbf{G}_1 + \cdots + \beta^1 \mathbf{G}_{t-1} + \beta^0 \mathbf{G}_t$$

- to such a representation where **t=0 always corresponds** to the **current** value of averaged gradients and t>0 corresponds to the historical ones

$$\mathbf{m}_0 = \beta^0 \mathbf{G}_0 + \beta^1 \mathbf{G}_1 + \cdots + \beta^t \mathbf{G}_t + \cdots = \sum_{t=0}^{+\infty} \beta^t \mathbf{G}_t$$

historical values



- from this, we can easily compute the number of steps after which the historical contribution of $\mathbf{G}_t$ will be smaller than e^{-1}:

$$\beta^t = e^{-\frac{t}{T}} \Rightarrow T = -\frac{1}{\ln \beta}$$

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: how to interpret momentum term ?

$$\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + \eta_t \alpha \boldsymbol{g}_t$$
$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{m}_t \quad -\eta_t w \boldsymbol{x}_{t-1}$$

- The characteristic constant T of the exponential average, says how many steps we need to perform in order to significantly change the value of running average

$$\beta^t = e^{-\frac{t}{T}} \Rightarrow T = -\frac{1}{\ln \beta}$$

β	0.5	0.9	0.95	0.99	0.995	0.999
T	~ 1.44	~ 10	~ 20	~ 100	~ 200	~ 1000

Example: AdamOptimizer

```
__init__(
    learning_rate=0.001,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam'
)
```

Note:

- consider beta=0.999, then you need about 1000 iterations to significantly change the updated vector
- when you train with small number of batches you will need to decrease T
- so when you may need to change the size of the batch you will have to tune the value of T

AdamW: Fixing Weight Decay Regularization in Adam (2017/2018)

Discussion: learning rate, batch size, weight decay are coupled

- Changing one of those parameter we will change the effect of the others
- We've already seen that when we increase batch size we should increase learning rate by the same factor.
- We obtained this rule by assuming that we want to keep gradient noise amplitude constant
- We can obtain the same conclusion by analysing very simple gradient descent where gradient is **constant**.
  Note: this is an extremely simplification of the problem!
- After one step we move our weights by:

$$\mathbf{W}^{(n+1)} = \mathbf{W}^{(n)} - \lambda \mathbf{G}$$

- After K steps we will get

$$\mathbf{W}^{(K)} = \mathbf{W}^{(0)} - \lambda K \mathbf{G} = \mathbf{W}^{(0)} - \lambda' \mathbf{G}$$

- When using **K-times bigger batch** the gradient **G** will be approximately the same, hence in order to reach the same point **after single step** we need to **multiply** LR by K. This is a simple motivation for the ***linear scaling rule***.

Summary:

Some short summary for all slides so far

- choose your LR wisely
- use better scheduler
- remember that size of the batch is coupled with LR, changing the latter affects the former
- try decoupled weight decay: **tf.contrib.opt.AdamWOptimizer**

Modifying features and labels

Modifying features and labels

I use the tensorflow estimators nomenclature:

- **features**: images, text data, sounds etc...
- **labels**: classes, boxes, segmentation masks

Label smoothing

An old idea proposed in: [Rethinking the Inception Architecture for Computer Vision]

- Modify your one-hot targets with equation (original notation from paper):

$$q'(k) = (1 - \epsilon)\delta_{k,y} + \frac{\epsilon}{K}.$$

K - is the number of classes
epsilon is a hyperparameter, usually 0.1

We refer to this change in ground-truth label distribution as *label-smoothing regularization*, or LSR.

- Interpretation in terms of minimized cost function: minimize standard CE with new term

standard CE regularization term

$$H(q', p) = -\sum_{k=1}^{K} \log p(k) q'(k) = (1-\epsilon) H(q, p) + \epsilon H(u, p)$$

Thus, LSR is equivalent to replacing a single cross-entropy loss $H(q, p)$ with a pair of such losses $H(q, p)$ and $H(u, p)$.

Label smoothing

- In tensorflow we can use: tf.losses.softmax_cross_entropy

```
tf.losses.softmax_cross_entropy(
    onehot_labels,
    logits,
    weights=1.0,
    label_smoothing=0,
    scope=None,
    loss_collection=tf.GraphKeys.LOSSES,
    reduction=Reduction.SUM_BY_NONZERO_WEIGHTS
)
```

From docs:

```
new_onehot_labels = onehot_labels * (1 - label_smoothing)
                  + label_smoothing / num_classes
```

Label smoothing - motivations with toy model

- As every regularization method label smoothing aims to reduce overfitting
- This overfitting usually comes from the fact we train the model to be overconfident in predictions i.e. model gives extremely high scores

A toy model which shows the importance of label smoothing:

- consider following classification problem with three classes
- we have following features and labels pairs:
  `{x=[1, 0, 1],y=0}, {x=[1, 0, -1],y=1}, {x=[0, 1, 0],y=2}`
- We create following one layer neural network:

$$\boldsymbol{y} = \mathrm{softmax}\left(\mathbf{W}\boldsymbol{x}\right) \qquad \boldsymbol{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}$$

- an example matrix (with a>0) which solves this problem may have following form

$$\boldsymbol{W} = \begin{pmatrix} a & 0 & a \\ a & 0 & -a \\ 0 & 2a & 0 \end{pmatrix} \quad \mathbf{W}\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 2a\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{W}\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} = 2a\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \mathbf{W}\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 2a\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

each class will be correctly predicted with this matrix, however what is an optimal value for a?

Label smoothing - motivations

- we have following features and labels pairs:
  `{x=[1, 0, 1],y=0}, {x=[1, 0, -1],y=1}, {x=[0, 1, 0],y=2}`
- We create following one layer neural network:

$$\boldsymbol{y} = \mathrm{softmax}\left(\mathbf{W}\boldsymbol{x}\right) \qquad \boldsymbol{W} = \begin{pmatrix} a & 0 & a \\ a & 0 & -a \\ 0 & 2a & 0 \end{pmatrix}$$

- probabilities for the first class:

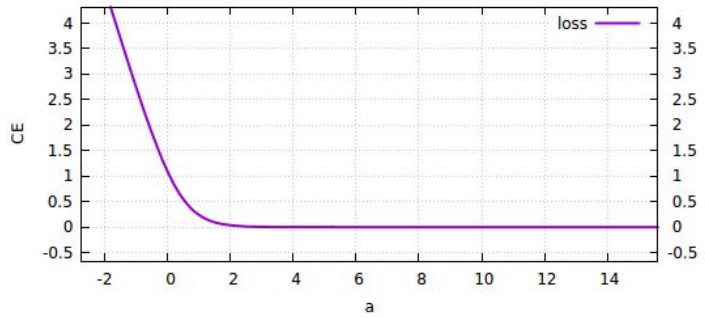
$$\boldsymbol{y}_1 = \mathrm{softmax}\left(\mathbf{W}\boldsymbol{x}_1\right) = \frac{1}{e^{2a}+2}\begin{pmatrix} e^{2a} \\ 1 \\ 1 \end{pmatrix}$$

- cross entropy cost for the first class (same for other two classes):

$$H\left(\hat{\mathbf{y}}_1, \mathbf{y}_1\right) = -\sum \hat{\mathbf{y}}_{1c} \log\left(\mathbf{y}_{1c}\right) = -\log \frac{e^{2a}}{e^{2a}+2} = \log\left(1+2e^{-2a}\right)$$

- Total loss:

$$\mathcal{L}_{\mathrm{CE}} = \frac{1}{3}\sum_{k}^{3} H\left(\hat{\mathbf{y}}_k, \mathbf{y}_k\right) = \log\left(1+2e^{-2a}\right)$$



Label smoothing - motivations

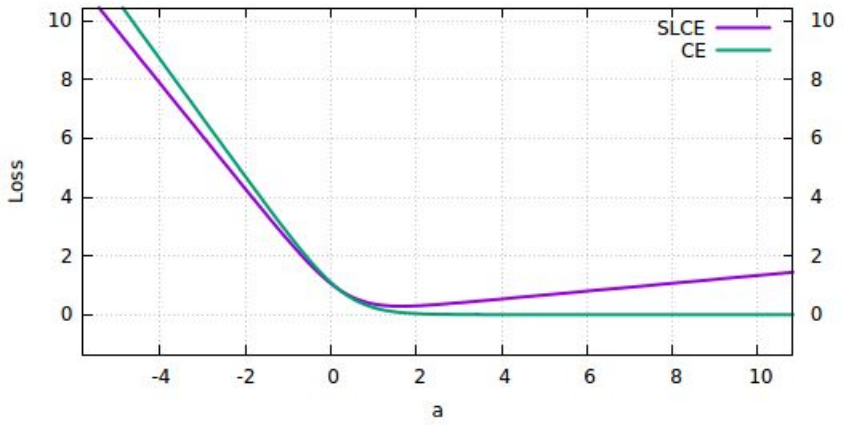
- The final solution for a which minimizes the loss: $\mathcal{L}_{\mathrm{CE}} = \frac{1}{3}\sum_{k}^{3} H\left(\hat{\mathbf{y}}_k, \mathbf{y}_k\right) = \log\left(1 + 2e^{-2a}\right)$

$$\boldsymbol{W} = \begin{pmatrix} +\infty & 0 & +\infty \\ +\infty & 0 & -\infty \\ 0 & +\infty & 0 \end{pmatrix}$$

an optimal solution ...

- What about label smoothing (eps=0.1) ?

$$\mathcal{L}_{\mathrm{LSCE}} = (1-\varepsilon)\log\left(1 + 2e^{-2a}\right) + \frac{2\varepsilon}{3}\log\left(2 + e^{2a}\right)$$



Consequences/properties of softmax CE:

- a monotonic function **with zero when scores are infinite** :(
- **training for a longer times** will make weights go to the infinity or very large values i.e. longer training may make your model overfit easier :(
- **overconfidence in scores** i.e. model returns 0 and 1 values - a nice signal of overfitting :(

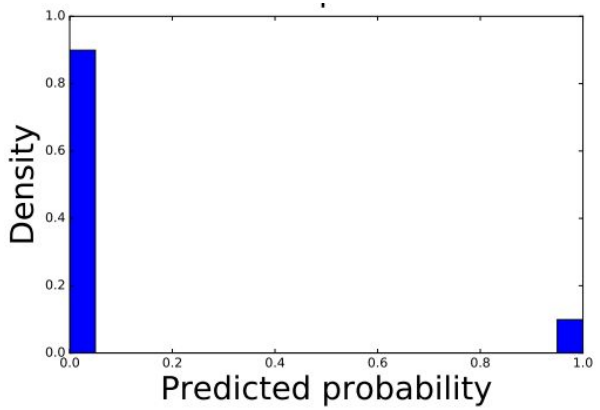


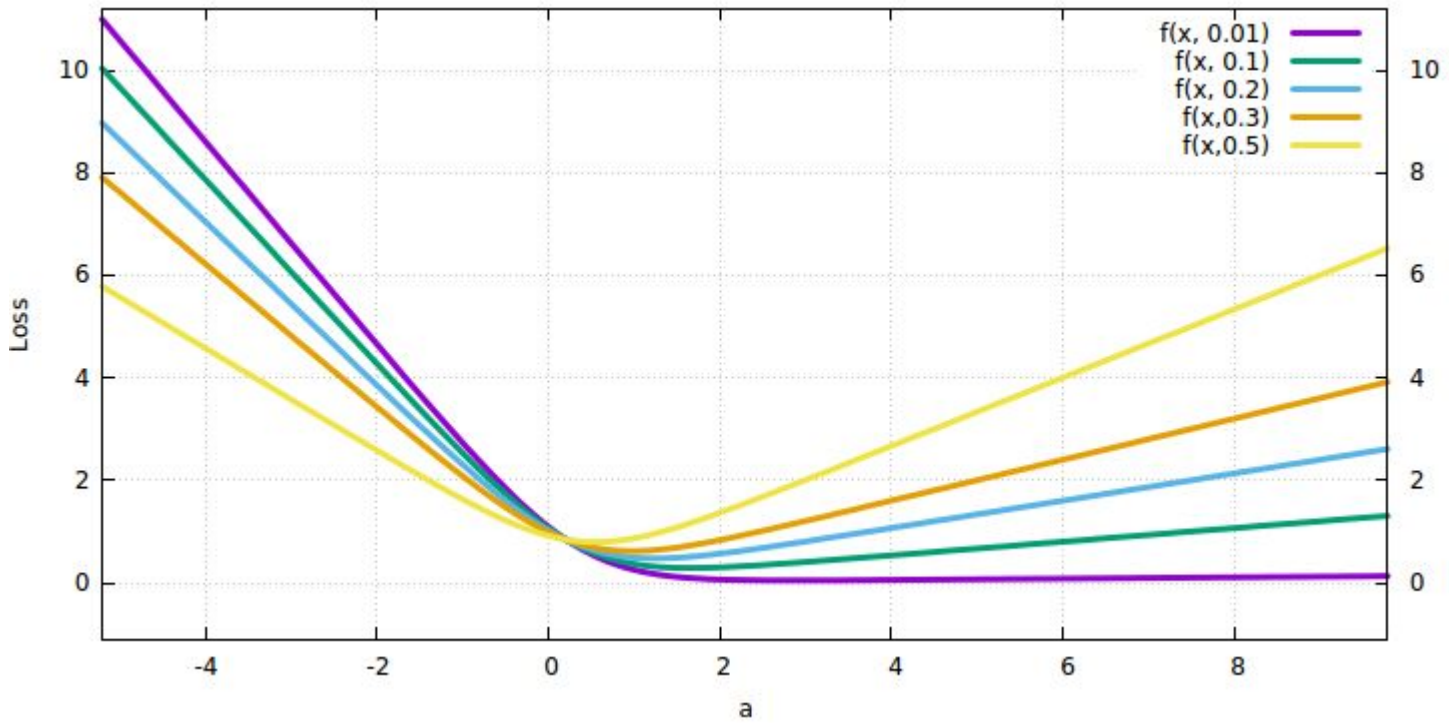
image source

There is a local minimum now i.e. we reduce overconfidence of the model!
Training for longer times can be also much safer

Label smoothing - motivations

- The final solution with LS for a which minimizes the loss:

$$\mathcal{L}_{\mathrm{LSCE}} = (1-\varepsilon)\log\left(1+2e^{-2a}\right) + \frac{2\varepsilon}{3}\log\left(2+e^{2a}\right)$$



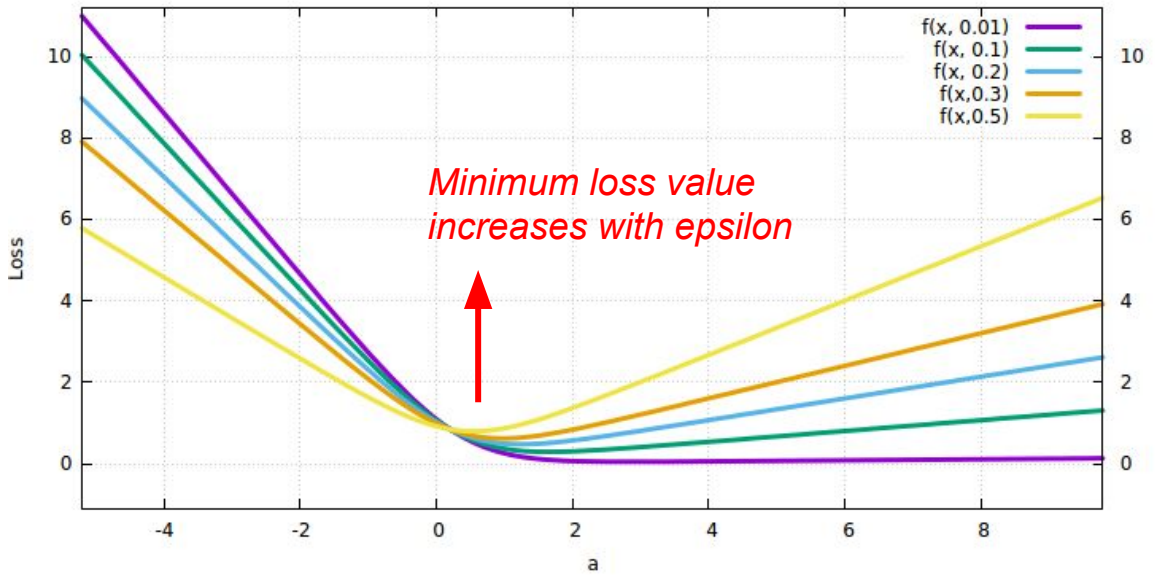
loss versus label smoothing parameter

Label smoothing - results

- The results show that using label smoothing may improve trained model,
- The cost of adding this regularization is minimal, so definitely one should try it in its own experiments

Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v2	23.4%	-	3.8
Inception-v2 RMSProp	23.1%	6.3	3.8
Inception-v2 Label Smoothing	22.8%	6.1	3.8
Inception-v2 Factorized 7×7	21.6%	5.8	4.8
Inception-v2 BN-auxiliary	**21.2%**	**5.6%**	4.8

Note: when using label smoothing, you loss will be much larger now, however your metrics like accuracy should improve.



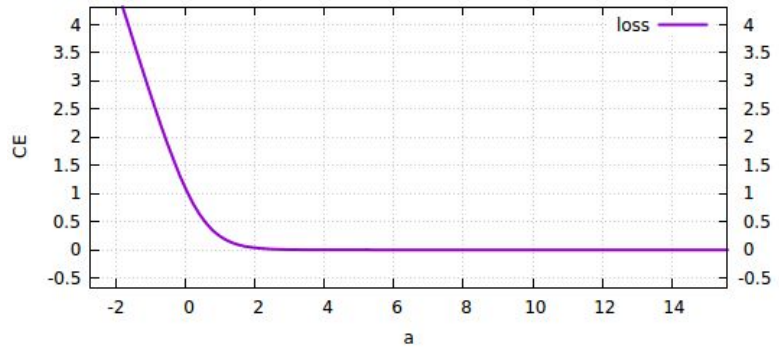
You can find more experiments in following, **not accepted ICLR 2017**, paper:

- Regularizing Neural Networks by Penalizing Confident Output Distributions
- This paper introduces similar method which is compared among others with **label smoothing** in various experiments.
- What is important that LS yields improvement w.r.t baseline in all of them

Label smoothing - softmax and final remarks

- We have arbitrarily guess the solution of our problem:
  `{x=[1, 0, 1],y=0}, {x=[1, 0, -1],y=1}, {x=[0, 1, 0],y=2}`
- with matrix:

$$\boldsymbol{W} = \begin{pmatrix} a & 0 & a \\ a & 0 & -a \\ 0 & 2a & 0 \end{pmatrix}$$



- which gives following loss:

$$\mathcal{L}_{\mathrm{CE}} = \frac{1}{3}\sum_{k}^{3} H\left(\hat{\mathbf{y}}_k, \mathbf{y}_k\right) = \log\left(1 + 2e^{-2a}\right)$$

- but there is an infinite many W which will give exactly the same final loss e.g.

$$\mathbf{W} = \begin{pmatrix} 2a & 0 & 0 \\ 0 & 0 & -2a \\ 0 & 2a & 0 \end{pmatrix}$$

- this form of solution **completely neglects some of the input** values of vector x,
- **inputs of form [1, alpha, beta]** will give exactly the same predictions for any alpha and beta,
- this can be viewed as a **form of information compression**, model will learn the easiest interpretation for the input data (from training set)
- **consider dataset of cars where every sport car is red**, it is clear the model will learn to distinguish between sport car and regular one by its color.

Modifying features and labels: mixup

mixup: Beyond Empirical Risk Minimization (ICLR 2018)

- an idea: mix features and labels with linear interpolation

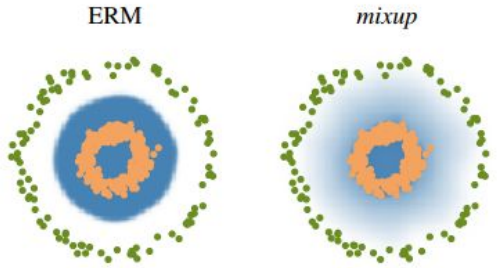
Contribution Motivated by these issues, we introduce a simple and data-agnostic data augmentation routine, termed *mixup* (Section 2). In a nutshell, *mixup* constructs virtual training examples

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \quad \text{where } x_i, x_j \text{ are raw input vectors}$$
$$\tilde{y} = \lambda y_i + (1-\lambda)y_j, \quad \text{where } y_i, y_j \text{ are one-hot label encodings}$$

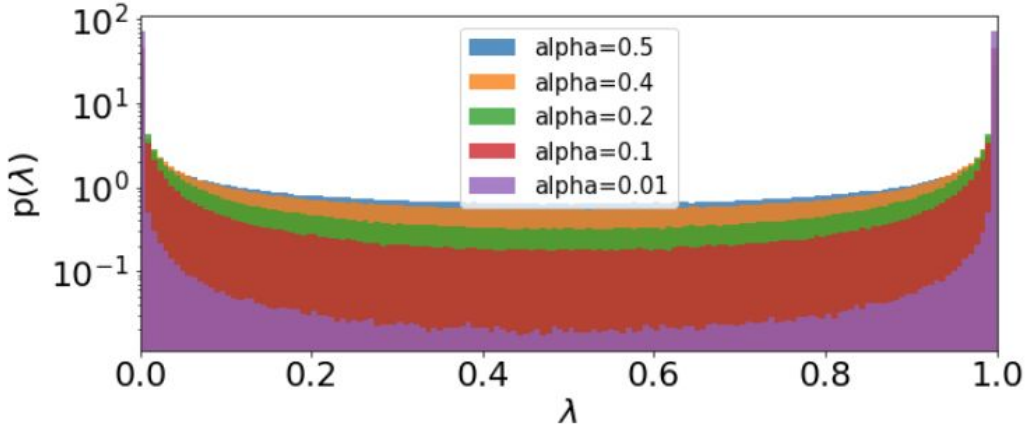
(x_i, y_i) and (x_j, y_j) are two examples drawn at random from our training data, and $\lambda \in [0, 1]$. Therefore, *mixup* extends the training distribution by incorporating the prior knowledge that linear interpolations of feature vectors should lead to linear interpolations of the associated targets. *mixup* can be implemented in a few lines of code, and introduces minimal computation overhead.

```
# y1, y2 should be one-hot vectors
for (x1, y1), (x2, y2) in zip(loader1, loader2):
    lam = numpy.random.beta(alpha, alpha)
    x = Variable(lam * x1 + (1. - lam) * x2)
    y = Variable(lam * y1 + (1. - lam) * y2)
    optimizer.zero_grad()
    loss(net(x), y).backward()
    optimizer.step()
```

(a) One epoch of *mixup* training in PyTorch.



(b) Effect of *mixup* ($\alpha = 1$) on a toy problem. Green: Class 0. Orange: Class 1. Blue shading indicates $p(y = 1|x)$.



beta distribution

mixup: Beyond Empirical Risk Minimization (ICLR 2018)

- an idea: mix features and labels with linear interpolation

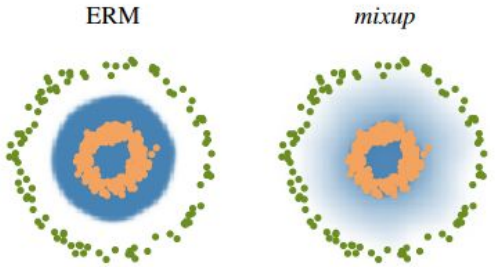
Contribution Motivated by these issues, we introduce a simple and data-agnostic data augmentation routine, termed *mixup* (Section 2). In a nutshell, *mixup* constructs virtual training examples

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j, \qquad \text{where } x_i, x_j \text{ are raw input vectors}$$
$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j, \qquad \text{where } y_i, y_j \text{ are one-hot label encodings}$$

(x_i, y_i) and (x_j, y_j) are two examples drawn at random from our training data, and $\lambda \in [0, 1]$. Therefore, *mixup* extends the training distribution by incorporating the prior knowledge that linear interpolations of feature vectors should lead to linear interpolations of the associated targets. *mixup* can be implemented in a few lines of code, and introduces minimal computation overhead.

```
# y1, y2 should be one-hot vectors
for (x1, y1), (x2, y2) in zip(loader1, loader2):
    lam = numpy.random.beta(alpha, alpha)
    x = Variable(lam * x1 + (1. - lam) * x2)
    y = Variable(lam * y1 + (1. - lam) * y2)
    optimizer.zero_grad()
    loss(net(x), y).backward()
    optimizer.step()
```

(a) One epoch of *mixup* training in PyTorch.



(b) Effect of *mixup* ($\alpha = 1$) on a toy problem. Green: Class 0. Orange: Class 1. Blue shading indicates $p(y = 1|x)$.

mixup:

- mixup is an extremely strong regularizer
- a form data augmentation,
- according to authors learned manifold is smoother: see toy model example

mixup: Beyond Empirical Risk Minimization (ICLR 2018)

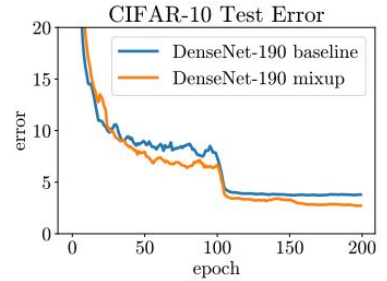
- selected results:

Model	Method	Epochs	Top-1 Error	Top-5 Error
ResNet-50	ERM (Goyal et al., 2017)	90	23.5	-
	mixup $\alpha = 0.2$	90	**23.3**	**6.6**
ResNet-101	ERM (Goyal et al., 2017)	90	22.1	-
	mixup $\alpha = 0.2$	90	**21.5**	**5.6**
ResNeXt-101 32*4d	ERM (Xie et al., 2016)	100	21.2	-
	ERM	90	21.2	5.6
	mixup $\alpha = 0.4$	90	**20.7**	**5.3**
ResNeXt-101 64*4d	ERM (Xie et al., 2016)	100	20.4	5.3
	mixup $\alpha = 0.4$	90	**19.8**	**4.9**
ResNet-50	ERM	200	23.6	7.0
	mixup $\alpha = 0.2$	200	**22.1**	**6.1**
ResNet-101	ERM	200	22.0	6.1
	mixup $\alpha = 0.2$	200	**20.8**	**5.4**
ResNeXt-101 32*4d	ERM	200	21.3	5.9
	mixup $\alpha = 0.4$	200	**20.1**	**5.0**

Table 1: Validation errors for ERM and *mixup* on the development set of ImageNet-2012.

Dataset	Model	ERM	*mixup*
CIFAR-10	PreAct ResNet-18	5.6	**4.2**
	WideResNet-28-10	3.8	**2.7**
	DenseNet-BC-190	3.7	**2.7**
CIFAR-100	PreAct ResNet-18	25.6	**21.1**
	WideResNet-28-10	19.4	**17.5**
	DenseNet-BC-190	19.0	**16.8**

(a) Test errors for the CIFAR experiments.



(b) Test error evolution for the best ERM and *mixup* models.

Figure 3: Test errors for ERM and *mixup* on the CIFAR experiments.

Model	Method	Validation set	Test set
LeNet	ERM	**9.8**	**10.3**
	mixup ($\alpha = 0.1$)	10.1	10.8
	mixup ($\alpha = 0.2$)	10.2	11.3
VGG-11	ERM	5.0	4.6
	mixup ($\alpha = 0.1$)	4.0	3.8
	mixup ($\alpha = 0.2$)	**3.9**	**3.4**

Figure 4: Classification errors of ERM and *mixup* on the Google commands dataset.

note dependency on the architecture, this is actually bad sign

Modifying initial conditions

Residual Learning Without Normalization via Better Initialization (ICLR 2019)

- A better initialization and network construction allows one to remove normalization layers and train deep neural networks,
- According to authors: *All you need to train deep residual networks is a good initialization; normalization layers are not necessary,*
- Paper is under review process,
- they propose ***ZeroInit*** method of network initialization. They train ResNet up to 10000 layers.

The authors tackle following problems/benchmarks:

- **Why normalization helps training.** We derive a lower bound for the gradient norm of a residual network at initialization, which explains why with standard initializations, normalization techniques are *essential* for training deep residual networks at maximal learning rate. (Section 2)
- **Training without normalization.** We propose ZeroInit, a method that rescales the standard initialization of residual branches by adjusting for the network architecture. ZeroInit enables training very deep residual networks stably at maximal learning rate without normalization. (Section 3)
- **Image classification.** We apply ZeroInit to replace batch normalization on image classification benchmarks CIFAR-10 (with Wide-ResNet) and ImageNet (with ResNet), and find ZeroInit with
- **Machine translation.** We apply ZeroInit to replace layer normalization on machine translation benchmarks IWSLT and WMT using the Transformer model, and find it outperforms the baseline and achieves new state-of-the-art results. (Section 4.3)

Residual Learning Without Normalization via Better Initialization (ICLR 2019)

- An overview of architecture

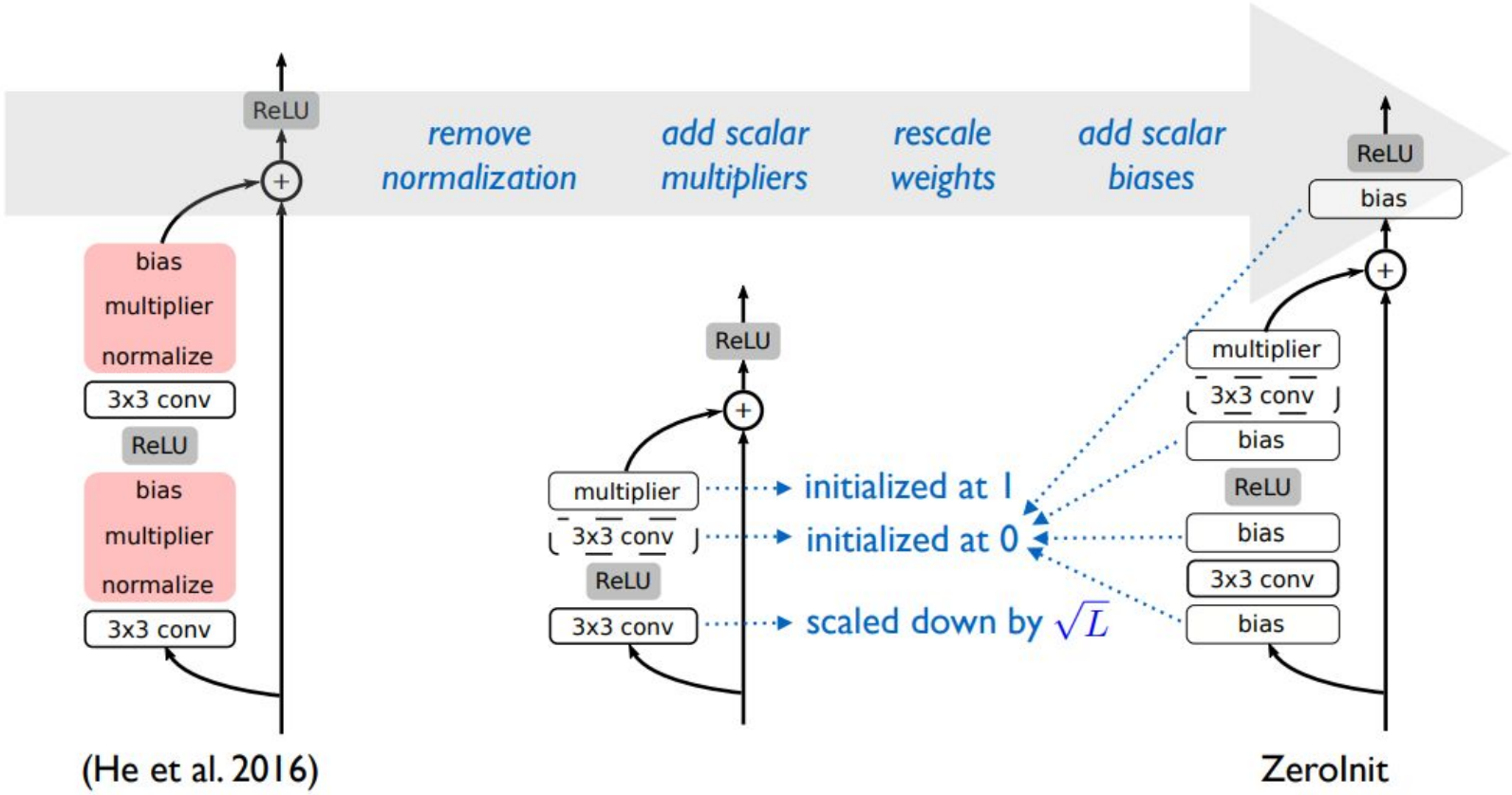


Figure 1: **Left:** ResNet basic block. Batch normalization (Ioffe & Szegedy, 2015) layers are marked in red. **Middle:** A simple network block that trains stably when stacked together. **Right:** ZeroInit further improves by adding bias parameters. (See Section 3 for details.)

Residual Learning Without Normalization via Better Initialization (ICLR 2019)

- The authors show that at initialization the variance of activations grows exponentially with depth (when there is no normalization in the network)
- The idea is to construct such network which will have desired properties at initialization e.g. no exploding and vanishing activations with depth

ZeroInit (or: How to train a deep residual network without normalization)

1. Initialize the classification layer and the last layer of each residual branch to 0.
2. Initialize every other layer using a standard method, e.g. He et al. (2015), and scale only the weight layers inside residual branches by $L^{-\frac{1}{2m-2}}$.
3. Add a scalar multiplier (initialized at 1) in every branch and a scalar bias (initialized at 0) before each convolution, linear, and element-wise activation layer.

L - is the number of residual blocks
m - number of layers inside block

Note: Authors provides theorems which justify this choice, see paper for more details.

Residual Learning Without Normalization via Better Initialization (ICLR 2019)

Selected results:

- CIFAR-10 test accuracy after first epoch for different initialization schemes

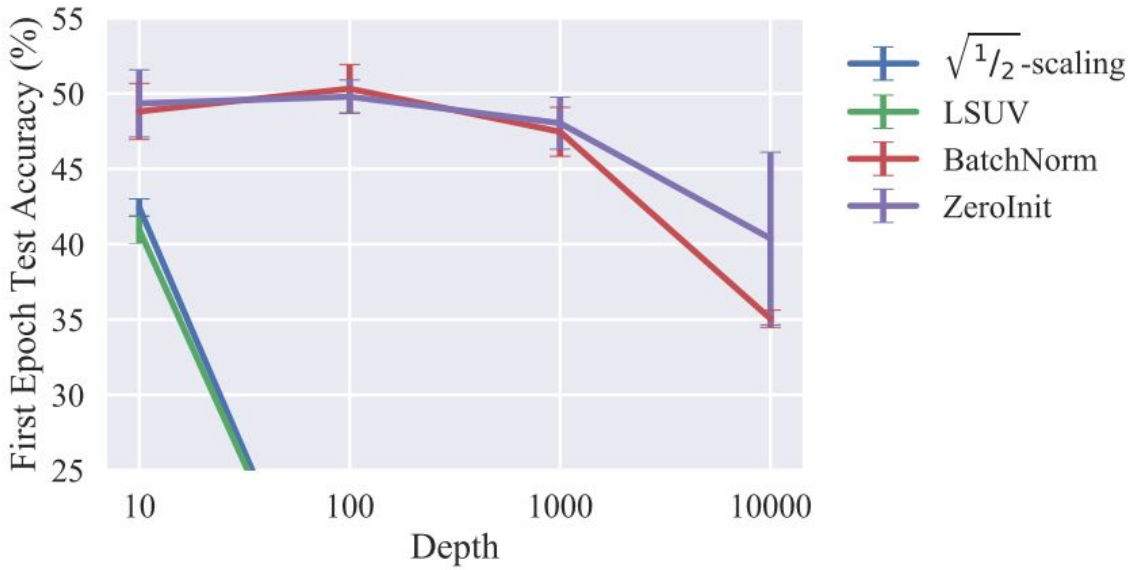


Figure 3: Depth of residual networks versus test accuracy at the first epoch for various methods on CIFAR-10 with the default BatchNorm learning rate. We observe that ZeroInit is able to train very deep networks with the same learning rate as batch normalization. (Higher is better.)

- high learning rate 0.1,
- I'm not sure if LSUV baseline was done properly (it should be done for every layer, but authors did it for the output of residual block): "*post-process an orthogonal initialization such that the **output variance of each residual block** is close to 1*"

Residual Learning Without Normalization via Better Initialization (ICLR 2019)

Selected results:

- A check whether deviation from theoretical initialization decreases performance.

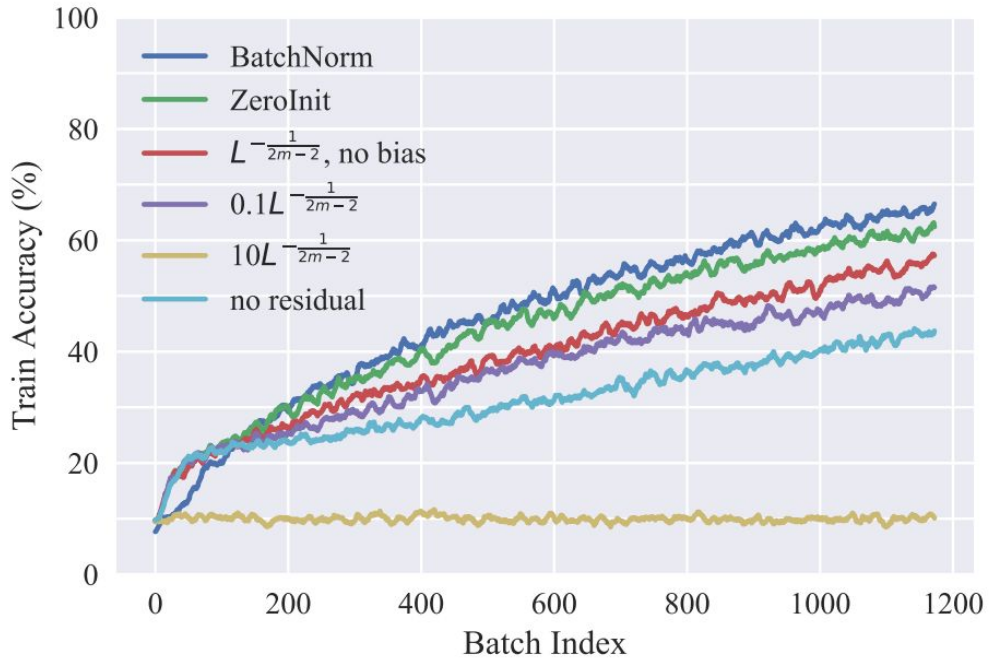
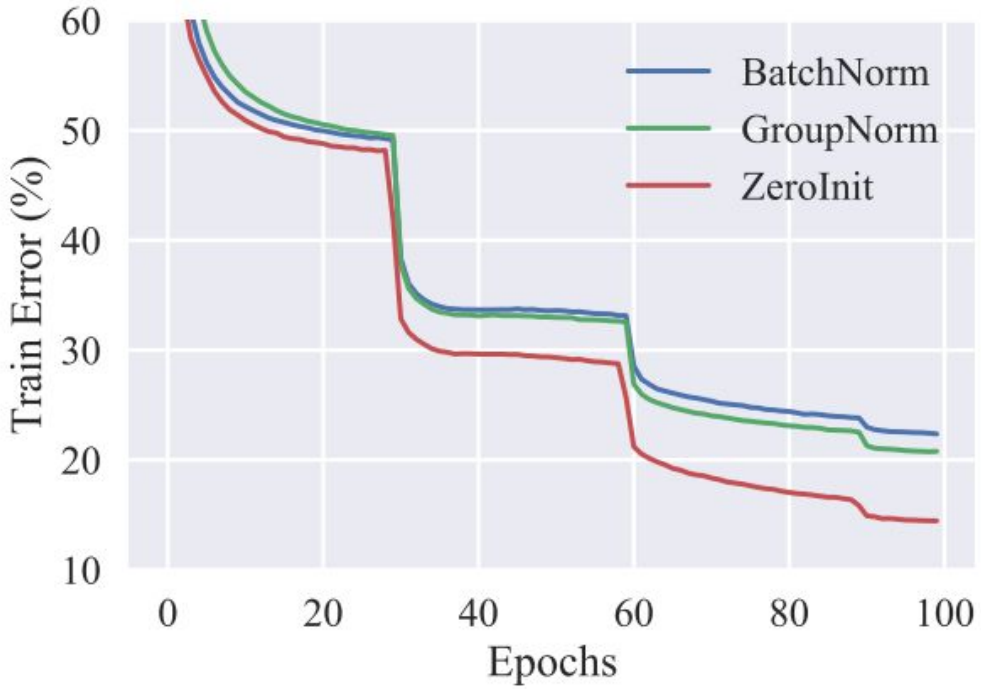


Figure 4: Minibatch training accuracy of ResNet-110 on CIFAR-10 dataset with different configurations in the first 3 epochs. We use minibatch size of 128 and smooth the curves using 10-step moving average.

Residual Learning Without Normalization via Better Initialization (ICLR 2019)

Selected results:

- Image classification: **ZeroInit** requires additional regularization



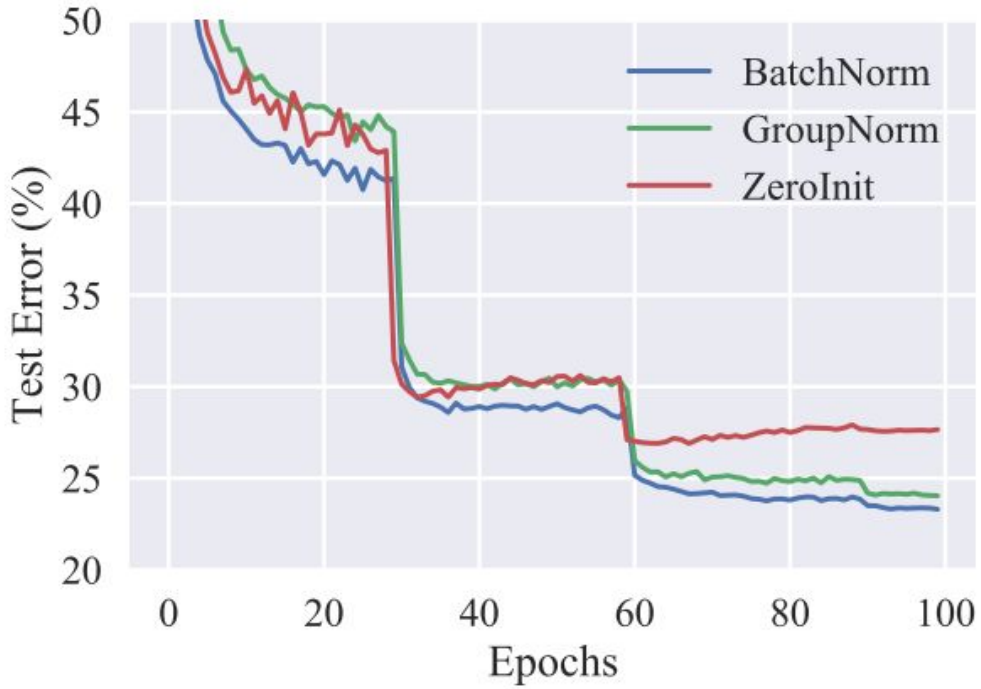


Figure 5: Training and test errors on ImageNet using ResNet-50 without additional regularization. We observe that ZeroInit is able to better fit the training data and that leads to overfitting - more regularization is needed. Results of BatchNorm and GroupNorm reproduced from (Wu & He, 2018).

Residual Learning Without Normalization via Better Initialization (ICLR 2019)

Selected results:

- Image classification: **ZeroInit** requires additional regularization,
- Normally we have additional regularization from batchnorm,

Model	Method	Normalization	Test Error (%)
ResNet-50	BatchNorm (Goyal et al., 2017)	✓	23.6
	BatchNorm + Mixup (Zhang et al., 2017)		**23.3**
	GroupNorm + Mixup		23.9
	Xavier Init (Shang et al., 2017)	✗	31.5
	ZeroInit		27.6
	ZeroInit + Mixup		**24.0**
ResNet-101	BatchNorm (Zhang et al., 2017)	✓	22.0
	BatchNorm + Mixup (Zhang et al., 2017)		**20.8**
	GroupNorm + Mixup		21.4
	ZeroInit + Mixup	✗	21.4

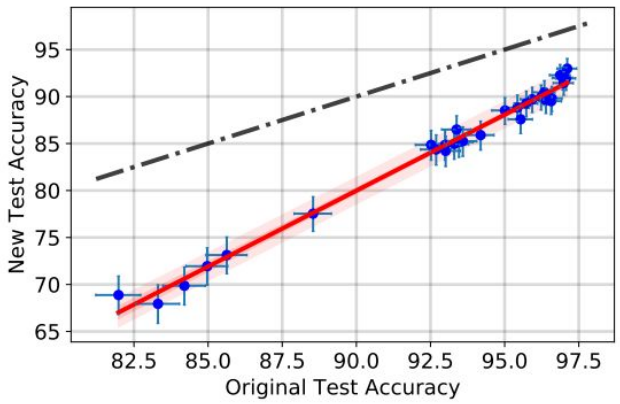
Table 2: ImageNet test results using the ResNet architecture. (Lower is better.)

Shake-shake regularization

Shake-Shake regularization (2017) - a motivation to try this method

From paper: Do CIFAR-10 Classifiers Generalize to CIFAR-10?

- they authors tested the accuracy of a larger number of SOTA models on new CIFAR-like dataset
- they asked: *How reliable are our current measures of progress in machine learning?*
- they found that: *The top model is still a recent **Shake-Shake network** with Cutout regularization*
- all models had linear drop in test accuracy



(a) All models

Model — Ideal reproducibility

Table 1: Model accuracy on the original CIFAR-10 test set and the new test set, with the gap reported as the difference between the two accuracies. Δ Rank is the relative difference in the ranking from the original test set to the new test set. For example, ΔRank = −2 means a model dropped in the rankings by two positions on the new test set.

	Original Accuracy	New Accuracy	Gap	Δ Rank
shake_shake_64d_cutout [3, 4]	97.1 [96.8, 97.4]	93.0 [91.8, 94.0]	4.1	0
shake_shake_96d [4]	97.1 [96.7, 97.4]	91.9 [90.7, 93.1]	5.1	-2
shake_shake_64d [4]	97.0 [96.6, 97.3]	91.4 [90.1, 92.6]	5.6	-2
wide_resnet_28_10_cutout [3, 22]	97.0 [96.6, 97.3]	92.0 [90.7, 93.1]	5	+1
shake_drop [21]	96.9 [96.5, 97.2]	92.3 [91.0, 93.4]	4.6	+3
shake_shake_32d [4]	96.6 [96.2, 96.9]	89.8 [88.4, 91.1]	6.8	-2
darc [11]	96.6 [96.2, 96.9]	89.5 [88.1, 90.8]	7.1	-4
resnext_29_4x64d [20]	96.4 [96.0, 96.7]	89.6 [88.2, 90.9]	6.8	-2
pyramidnet_basic_110_270 [6]	96.3 [96.0, 96.7]	90.5 [89.1, 91.7]	5.9	+3
resnext_29_8x64d [20]	96.2 [95.8, 96.6]	90.0 [88.6, 91.2]	6.3	+3
wide_resnet_28_10 [22]	95.9 [95.5, 96.3]	89.7 [88.3, 91.0]	6.2	+2
pyramidnet_basic_110_84 [6]	95.7 [95.3, 96.1]	89.3 [87.8, 90.6]	6.5	0
densenet_BC_100_12 [10]	95.5 [95.1, 95.9]	87.6 [86.1, 89.0]	8	-2
neural_architecture_search [23]	95.4 [95.0, 95.8]	88.8 [87.4, 90.2]	6.6	+1
wide_resnet_tf [22]	95.0 [94.6, 95.4]	88.5 [87.0, 89.9]	6.5	+1
resnet_v2_bottleneck_164 [8]	94.2 [93.7, 94.6]	85.9 [84.3, 87.4]	8.3	-1
vgg16_keras [14, 18]	93.6 [93.1, 94.1]	85.3 [83.6, 86.8]	8.3	-1
resnet_basic_110 [7]	93.5 [93.0, 93.9]	85.2 [83.5, 86.7]	8.3	-1
resnet_v2_basic_110 [8]	93.4 [92.9, 93.9]	86.5 [84.9, 88.0]	6.9	+3
resnet_basic_56 [7]	93.3 [92.8, 93.8]	85.0 [83.3, 86.5]	8.3	0
resnet_basic_44 [7]	93.0 [92.5, 93.5]	84.2 [82.6, 85.8]	8.8	-3

Shake-Shake regularization (2017)

- Shake-Shake regularization is a regularization method proposed for ResNet architectures
- Basic idea: a random augmentation on the hidden activation level, instead of features
- Results and detailed discussion can be found in paper

1.3 Training procedure

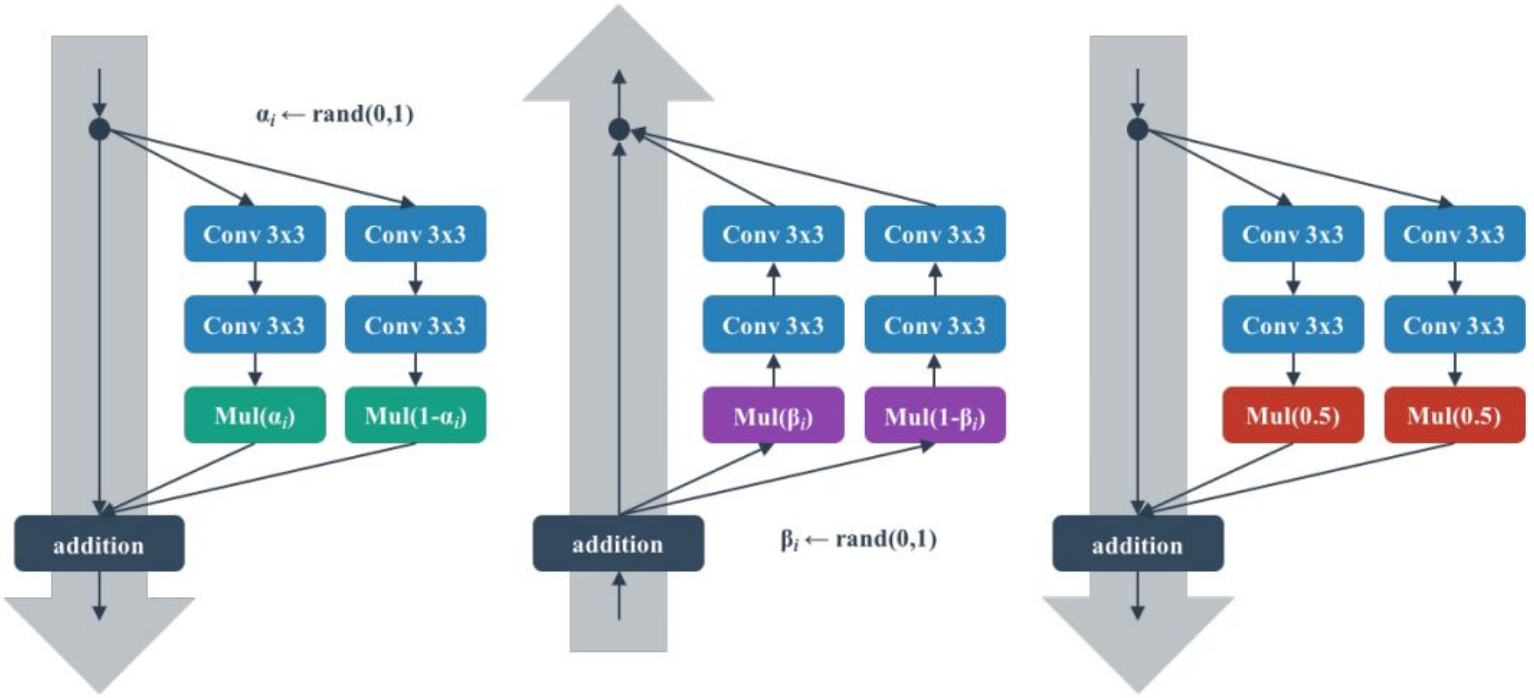


Figure 1: **Left:** Forward training pass. **Center:** Backward training pass. **Right:** At test time.

Summary: “*Bag of Tricks for Image Classification with Convolutional Neural Networks*“

Bag of Tricks for Image Classification with Convolutional Neural Networks (2018) (Amazon)

Model	FLOPs	top-1	top-5
ResNet-50 [9]	3.9 G	75.3	92.2
ResNeXt-50 [27]	4.2 G	77.8	-
SE-ResNet-50 [12]	3.9 G	76.71	93.38
SE-ResNeXt-50 [12]	4.3 G	78.90	94.51
DenseNet-201 [13]	4.3 G	77.42	93.66
ResNet-50 + tricks (ours)	4.3 G	**79.29**	**94.63**

Table 1: **Computational costs and validation accuracy of various models.** ResNet, trained with our "tricks", is able to outperform newer and improved architectures trained with standard pipeline.

Model	Baseline		Reference	
	Top-1	Top-5	Top-1	Top-5
ResNet-50 [9]	75.87	92.70	75.3	92.2
Inception-V3 [26]	77.32	93.43	78.8	94.4
MobileNet [11]	69.03	88.71	70.6	-

Table 2: **Validation accuracy of reference implementations and our baseline.** Note that the numbers for Inception V3 are obtained with 299-by-299 input images.

Standard training pipeline:

Algorithm 1 Train a neural network with mini-batch stochastic gradient descent.

```
initialize(net)
for epoch = 1, ..., K do
    for batch = 1, ..., #images/b do
        images ← uniformly random sample b images
        X, y ← preprocess(images)
        z ← forward(net, X)
        ℓ ← loss(z, y)
        grad ← backward(ℓ)
        update(net, grad)
    end for
end for
```

- for more details about input preprocessing used for the baseline models check section: **2.1. Baseline Training Procedure**

Bag of Tricks for Image Classification with Convolutional Neural Networks (2018)

1. Tricks for large-batch training:

- **Linear scaling learning rate** (already discussed)
- **Learning rate warmup** (already discussed)
- **zero gamma in batch norm** (x*gamma + bias) (resnet branch reduces to identity function) (already discussed in ZeroInit method)
- **No bias decay** - do not apply L2 (or weight decay) to bias tensors (gamma and bias in BN too)

2. Use low precision training: use float 16 instead float 32 to speed up computations (this will depend on GPU)

- **keep activations and updates at 16bit precision**
- **keep weights in 32bit** -> this is to reduce overflow problem

Heuristic	BS=256		BS=1024	
	Top-1	Top-5	Top-1	Top-5
Linear scaling	75.87	92.70	75.17	92.54
+ LR warmup	76.03	92.81	75.93	92.84
+ Zero γ	76.19	93.03	76.37	92.96
+ No bias decay	76.16	92.97	76.03	92.86
+ FP16	76.15	93.09	76.21	92.97

Table 4: The breakdown effect for each effective training heuristic on ResNet-50.

Bag of Tricks for Image Classification with Convolutional Neural Networks (2018)

3. **Architecture tweaks** (simple changes as model stride, pooling layer etc)
   - They introduce B-C-D variants of ResNet blocks

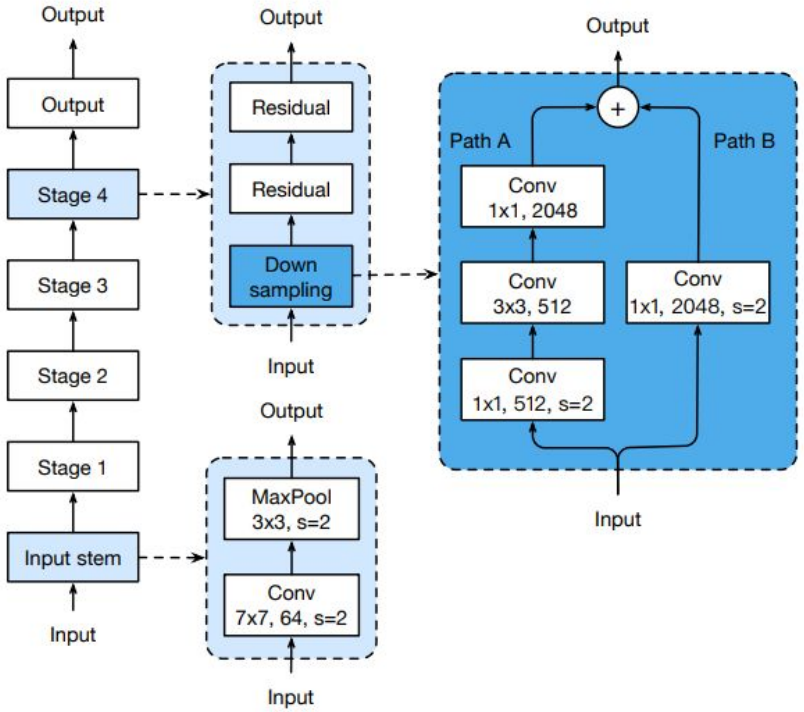


Figure 1: The architecture of ResNet-50. The convolution kernel size, output channel size and stride size (default is 1) are illustrated, similar for pooling layers.

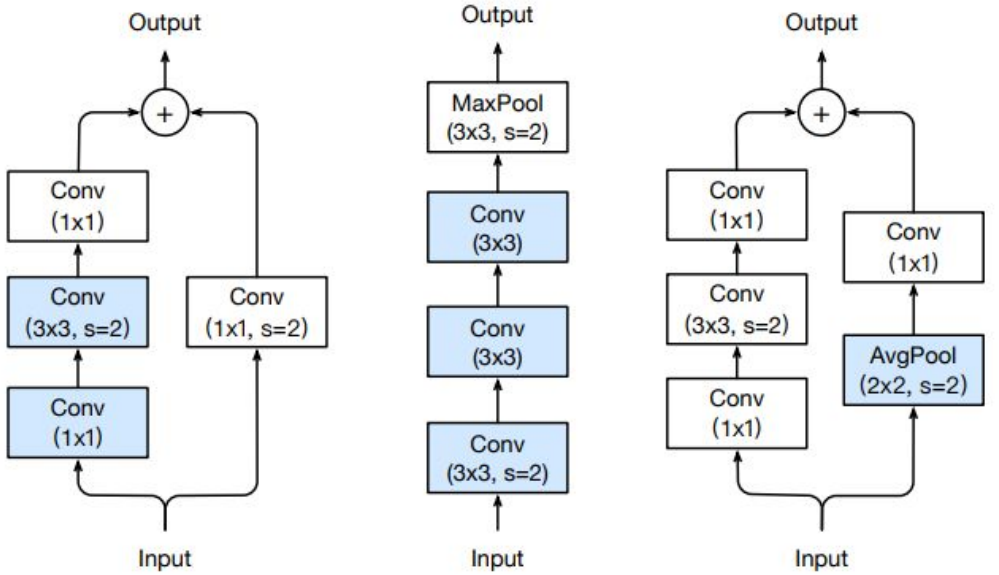


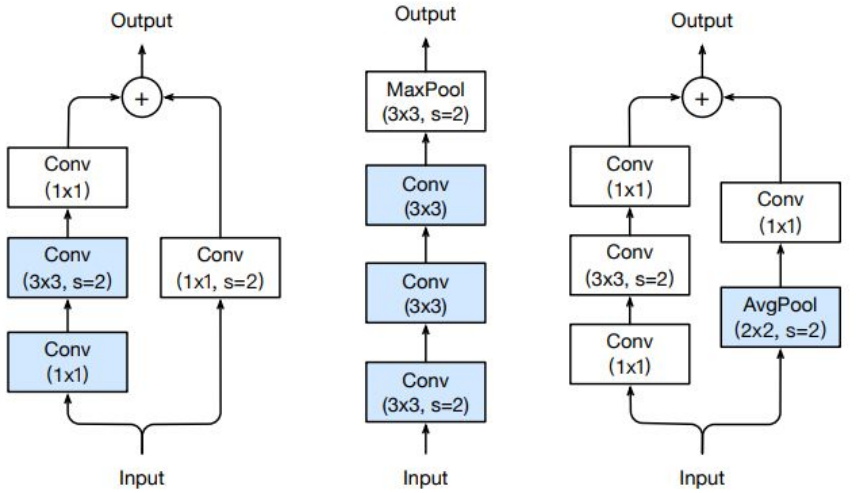
Figure 2: Three ResNet tweaks. ResNet-B modifies the downsampling block of Resnet. ResNet-C further modifies the input stem. On top of that, ResNet-D again modifies the downsampling block.

Bag of Tricks for Image Classification with Convolutional Neural Networks (2018)

3. Architecture tweaks (simple changes as model stride, pooling layer etc)
   - They introduce B-C-D variants

Model	#params	FLOPs	Top-1	Top-5
ResNet-50	25 M	**3.8 G**	76.21	92.97
ResNet-50-B	25 M	4.1 G	76.66	93.28
ResNet-50-C	25 M	4.3 G	76.87	93.48
ResNet-50-D	25 M	4.3 G	**77.16**	**93.52**

Table 5: Compare ResNet-50 with three model tweaks on model size, FLOPs and ImageNet validation accuracy.



(a) ResNet-B (b) ResNet-C (c) ResNet-D

Figure 2: Three ResNet tweaks. ResNet-B modifies the downsampling block of Resnet. ResNet-C further modifies the input stem. On top of that, ResNet-D again modifies the downsampling block.

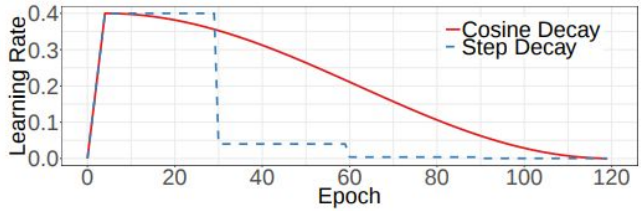
Bag of Tricks for Image Classification with Convolutional Neural Networks (2018)

4. Training Refinements:
   - **Cosine Learning Rate Decay** (already discussed)
   - **Label Smoothing** (already discussed)
   - **Knowledge Distillation** - train student Net (a smaller one) by forcing similar softmax activations with teacher Net (the bigger one): z is student, r is teacher

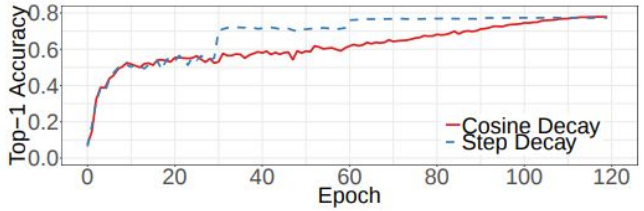
$$\ell(p, \text{softmax}(z)) + T^2 \ell(\text{softmax}(r/T), \text{softmax}(z/T))$$

   - **Mixup Training** (already discussed)

Refinements	ResNet-50-D		Inception-V3		MobileNet	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Efficient	77.16	93.52	77.50	93.60	71.90	90.53
+ cosine decay	77.91	93.81	78.19	94.06	72.83	91.00
+ label smoothing	78.31	94.09	78.40	94.13	72.93	91.14
+ distill w/o mixup	78.67	94.36	78.26	94.01	71.97	90.89
+ mixup w/o distill	79.15	94.58	**78.77**	**94.39**	**73.28**	**91.30**
+ distill w/ mixup	**79.29**	**94.63**	78.34	94.16	72.51	91.02

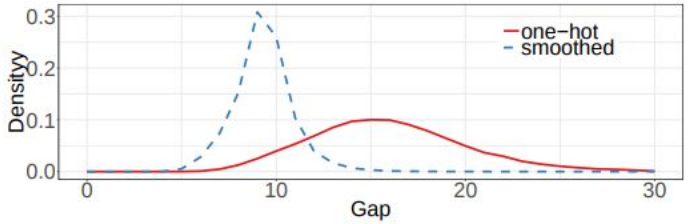


(a) Learning Rate Schedule



(b) Validation Accuracy

Figure 3: Visualization of learning rate schedules with warm-up. Top: cosine and step schedules for batch size 1024. Bottom: Top-1 validation accuracy curve with regard to the two schedules.



(b) Empirical gap from ImageNet validation set

Figure 4: Visualization of the effectiveness of label smoothing on ImageNet. Top: theoretical gap between z_p^* and others decreases when increasing ε. Bottom: The empirical distributions of the gap between the maximum prediction and the average of the rest.

Bag of Tricks for Image Classification with Convolutional Neural Networks (2018)

Object detection (**left**) and segmentation (**right**)

- In the case of Object detection: *"The VGG-19 base model in Faster-RCNN is replaced with various pretrained models in the previous discussion"*

Refinement	Top-1	mAP
B-standard	76.14	77.54
D-efficient	77.16	78.30
+ cosine	77.91	79.23
+ smooth	78.34	80.71
+ distill w/o mixup	78.67	80.96
+ mixup w/o distill	79.16	81.10
+ distill w/ mixup	79.29	**81.33**

Table 8: Faster-RCNN performance with various pre-trained base networks evaluated on Pascal VOC.

Refinement	Top-1	PixAcc	mIoU
B-standard	76.14	78.08	37.05
D-efficient	77.16	78.88	38.88
+ cosine	77.91	**79.25**	**39.33**
+ smooth	78.34	78.64	38.75
+ distill w/o mixup	78.67	78.97	38.90
+ mixup w/o distill	79.16	78.47	37.99
+ mixup w/ distill	79.29	78.72	38.40

Table 9: FCN performance with various base networks evaluated on ADE20K.

discussed in paper: smoothing in segmentation may be not the best ides

Questions?

References:

Learning rate and batch size:

- A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay
- Cyclical Learning Rates for Training Neural Networks
- [BLOG] The 1cycle policy
- Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates
- SGDR: Stochastic Gradient Descent with Warm Restarts
- [BLOG] Learning rate
- Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
- Don't Decay the Learning Rate, Increase the Batch Size
- Second-order Optimization Method for Large Mini-batch: Training ResNet-50 on ImageNet in 35 Epochs

References:

Optimizers and weight decay

- Fixing Weight Decay Regularization in Adam (2018)
- Three Mechanisms of Weight Decay Regularization (2018)

Features and labels

- Rethinking the Inception Architecture for Computer Vision (2015)
- Regularizing Neural Networks by Penalizing Confident Output Distributions (2017)
- mixup: Beyond Empirical Risk Minimization (2018)

Initialization:

- Residual Learning Without Normalization via Better Initialization (2018)

Others:

- Do CIFAR-10 Classifiers Generalize to CIFAR-10? (2018)
- Shake-Shake regularization (2017)
- Bag of Tricks for Image Classification with Convolutional Neural Networks (2018)

FORNAX

WWW.FORNAX.AI