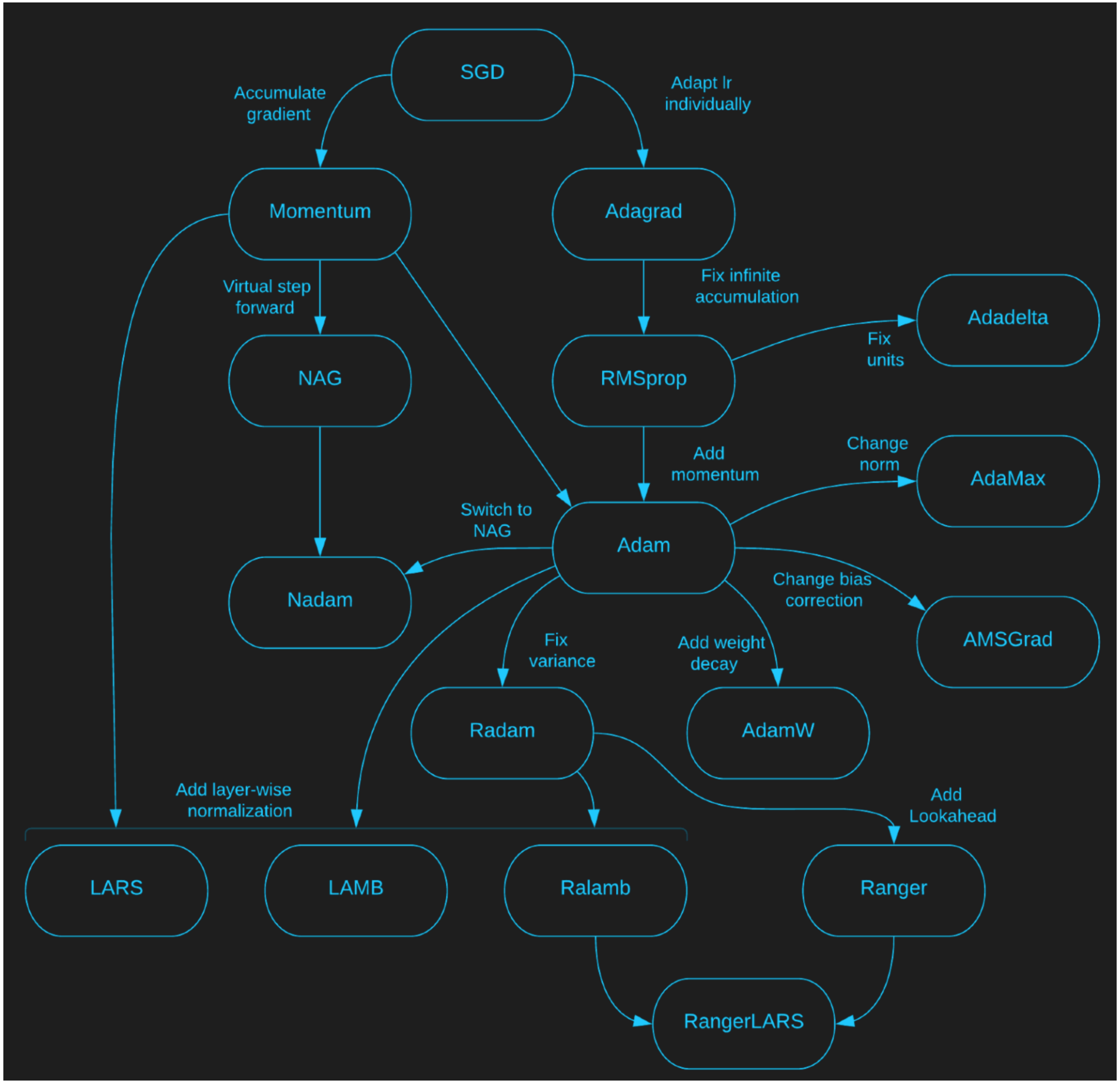
Optimization for Machine Learning

Lecture 19: Optimization for Neural networks

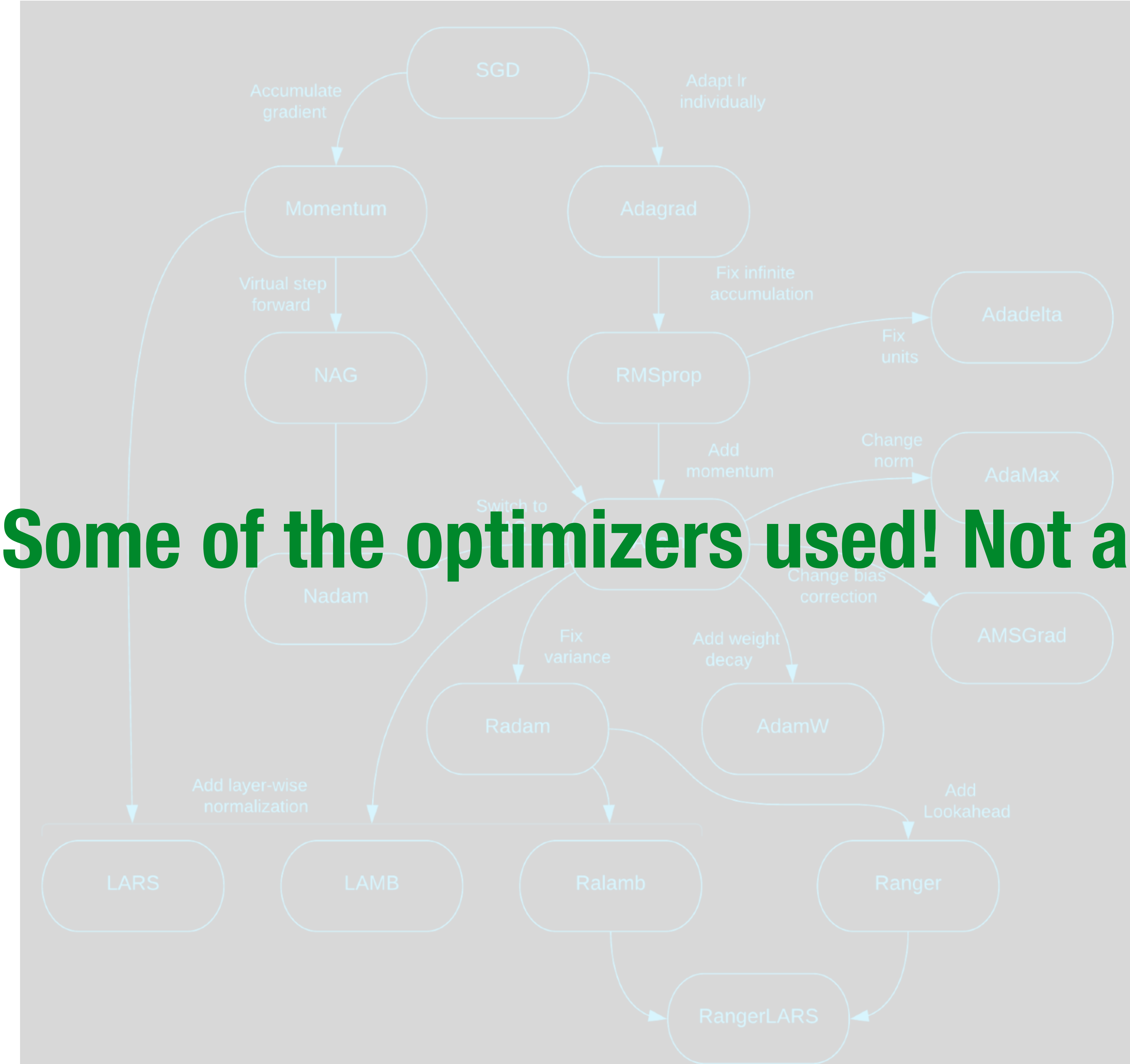
6.881: MIT

Suvrit Sra
Massachusetts Institute of Technology

May 04, 2021



https://darel13712.github.io/ml/optimizers.html



Some of the optimizers used! Not all!

https://darel13712.github.io/ml/optimizers.html

Some Aspects of NN Optimization

- Backprop ➠ SGD
- Mini-batches
- Initialization
- Batchnorm
- Gradient clipping
- Adaptive methods
- Momentum
- Layerwise params
- …and more!

Some Aspects of NN Optimization

- Backprop ➠ SGD
- Mini-batches
- Initialization
- Batchnorm
- Gradient clipping
- Adaptive methods
- Momentum
- Layerwise params
- …and more!

All while keeping validation / test error performance in mind

SGD: Neural network training

$$\min_{\theta} \; R_N(\theta) := \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, F(x_i;\theta))$$

label

network output

$\ell(y, z) = \max(0, 1 - yz)$

$\ell(y, z) = \frac{1}{2}(y - z)^2$

SGD: Neural network training

$$\min_{\theta} \; R_N(\theta) := \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, F(x_i;\theta))$$

label — network output

$$\ell(y,z) = \max(0, 1 - yz)$$
$$\ell(y,z) = \tfrac{1}{2}(y-z)^2$$

SGD

$$\theta \leftarrow \theta - \eta \frac{\partial \ell(y, F(x;\theta))}{\partial \theta}$$

SGD: Neural network training

$$\min_{\theta} \; R_N(\theta) := \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, F(x_i;\theta))$$

label; network output

$\ell(y, z) = \max(0, 1 - yz)$

$\ell(y, z) = \frac{1}{2}(y - z)^2$

SGD

$$\theta \leftarrow \theta - \eta \frac{\partial \ell(y, F(x;\theta))}{\partial \theta}$$

> Iterative method. How to select θ_0?

SGD: Neural network training

$$\min_{\theta} \; R_N(\theta) := \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, F(x_i;\theta))$$

label — network output

$$\ell(y,z) = \max(0, 1 - yz)$$
$$\ell(y,z) = \tfrac{1}{2}(y-z)^2$$

SGD

$$\theta \leftarrow \theta - \eta \frac{\partial \ell(y, F(x;\theta))}{\partial \theta}$$

2 What about the step-size η, aka "learning rate"?

SGD: Neural network training

$$\min_{\theta} \; R_N(\theta) := \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, F(x_i;\theta))$$

label

network output

$$\ell(y,z) = \max(0, 1 - yz)$$
$$\ell(y,z) = \tfrac{1}{2}(y-z)^2$$

SGD

$$\theta \leftarrow \theta - \eta \frac{\partial \ell(y, F(x;\theta))}{\partial \theta}$$

3 How to compute this stochastic gradient?

SGD: Neural network training

$$\min_{\theta} \; R_N(\theta) := \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, F(x_i;\theta))$$

label — network output

$$\ell(y,z) = \max(0, 1 - yz)$$

$$\ell(y,z) = \tfrac{1}{2}(y-z)^2$$

SGD

$$\theta \leftarrow \theta - \eta \frac{\partial \ell(y, F(x;\theta))}{\partial \theta}$$

4 In reality: momentum, clipping, adaptivity, …

1 Iterative method. How to select θ_0?

1. Initialization

Properly initializing a NN important. NN loss is highly nonconvex; optimizing it to attain a "good" solution hard, requires careful tuning.

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

1. Initialization

Properly initializing a NN important. NN loss is highly nonconvex; optimizing it to attain a “good” solution hard, requires careful tuning.

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

Example: Don’t initialize all weights to be the same — why?

1. Initialization

Properly initializing a NN important. NN loss is highly nonconvex; optimizing it to attain a “good” solution hard, requires careful tuning.

> On the importance of initialization and momentum in deep learning
>
> Ilya Sutskever[1] ILYASU@GOOGLE.COM
> James Martens JMARTENS@CS.TORONTO.EDU
> George Dahl GDAHL@CS.TORONTO.EDU
> Geoffrey Hinton HINTON@CS.TORONTO.EDU

Example: Don’t initialize all weights to be the same — why?

Random: Initialize randomly, e.g., via the Gaussian $N(0, \sigma^2)$, where std σ depends on the number of neurons in a given layer. *Symmetry breaking.*

1. Initialization

Properly initializing a NN important. NN loss is highly nonconvex; optimizing it to attain a "good" solution hard, requires careful tuning.

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

Example: Don't initialize all weights to be the same — why?

Random: Initialize randomly, e.g., via the Gaussian $N(0, \sigma^2)$, where std σ depends on the number of neurons in a given layer. *Symmetry breaking.*

Why? roughly ensure that random input to a unit does *not depend* on the number of inputs it gets. ***For ReLUs current recommendation:*** use $\sigma^2 = 2/n$

1. Initialization

Properly initializing a NN important. NN loss is highly nonconvex; optimizing it to attain a "good" solution hard, requires careful tuning.

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

Example: Don't initialize all weights to be the same — why?

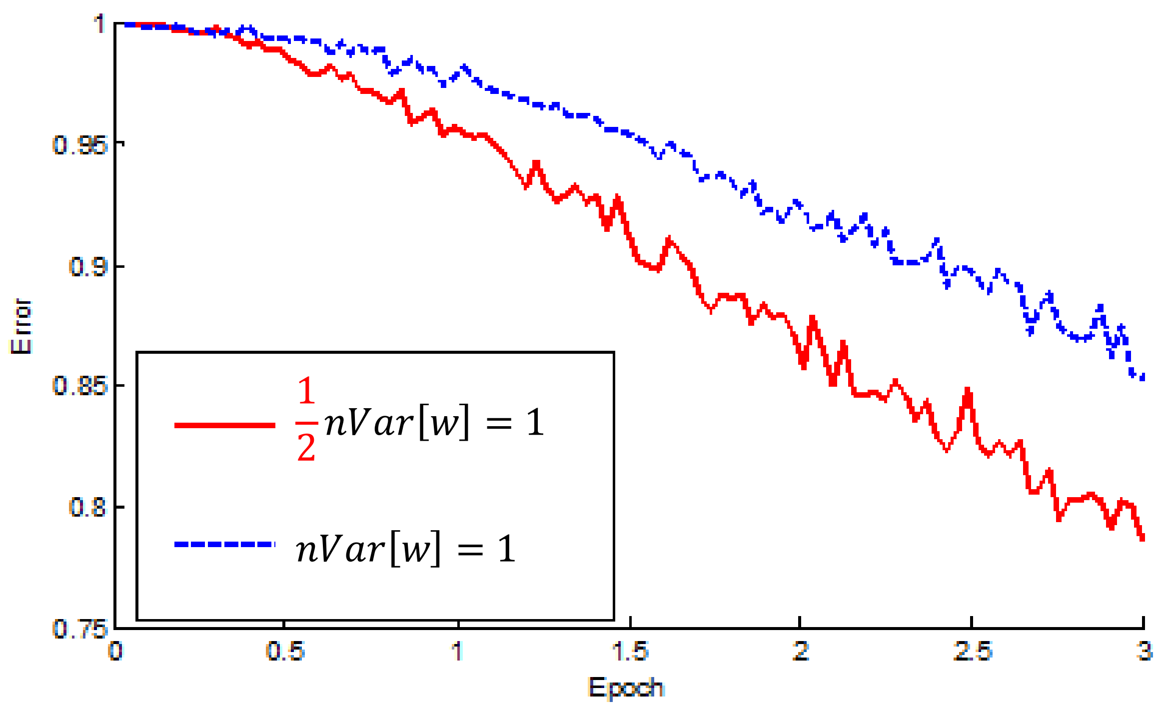
Random: Initialize randomly, e.g., via the Gaussian $N(0, \sigma^2)$, where std σ depends on the number of neurons in a given layer. *Symmetry breaking.*

Why? roughly ensure that random input to a unit does *not depend* on the number of inputs it gets. ***For ReLUs current recommendation:*** use $\sigma^2 = 2/n$

See also: http://cs231n.github.io/neural-networks-2/ for additional practical notes

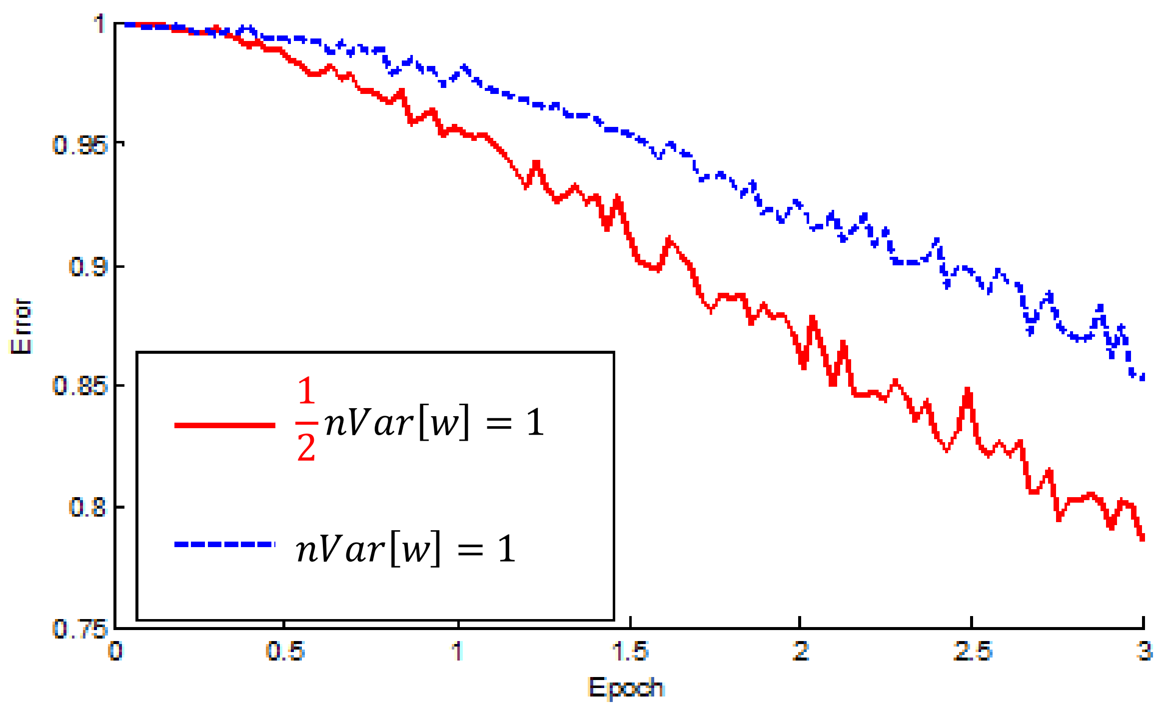
1. Impact of initialization

22-layer ReLU net:
good init converges faster

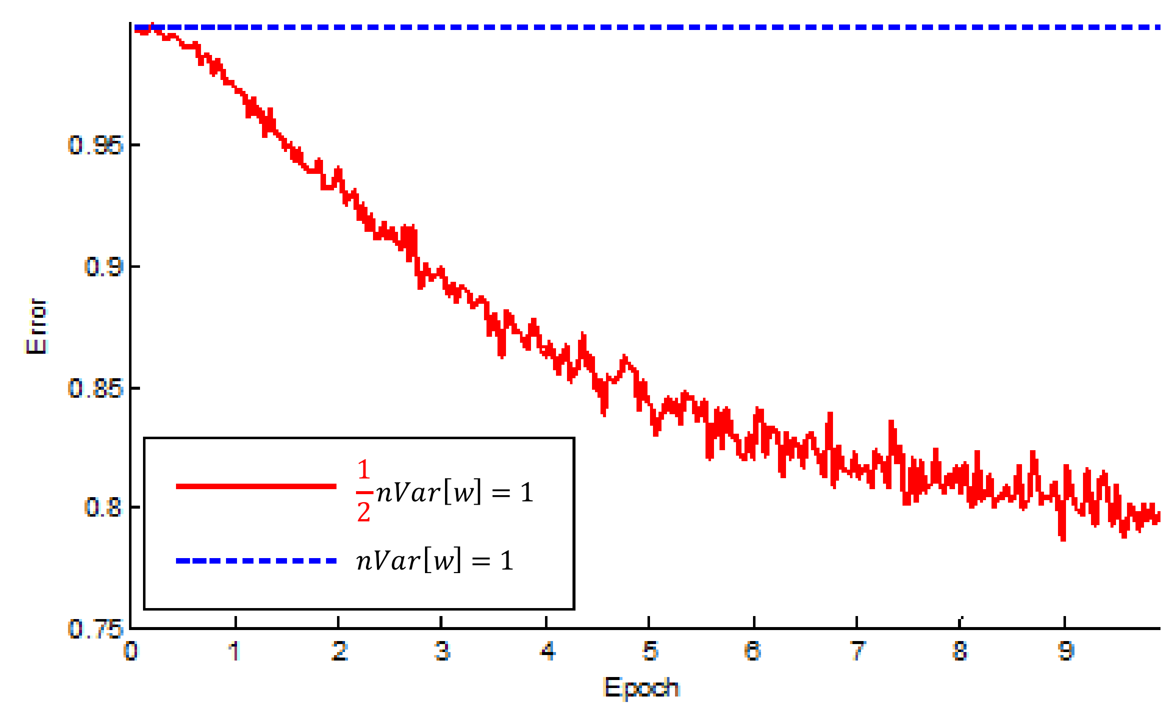


1. Impact of initialization

22-layer ReLU net:
good init converges faster



30-layer ReLU net:
good init is able to converge

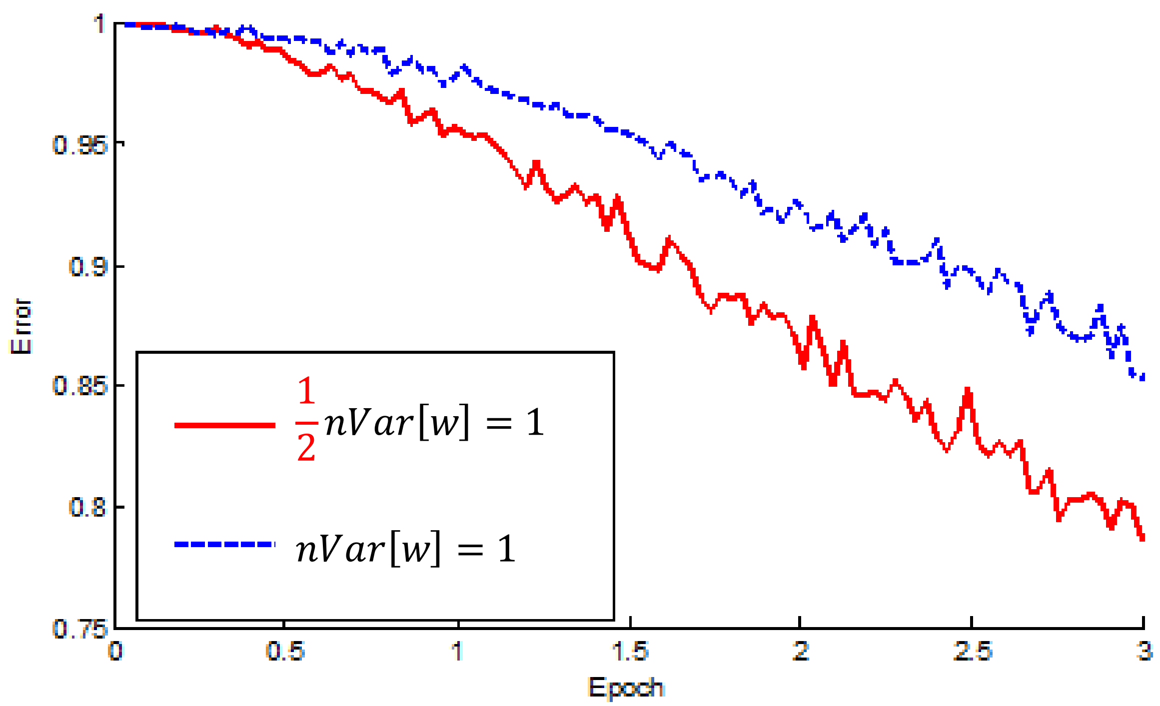


*Figures show the beginning of training

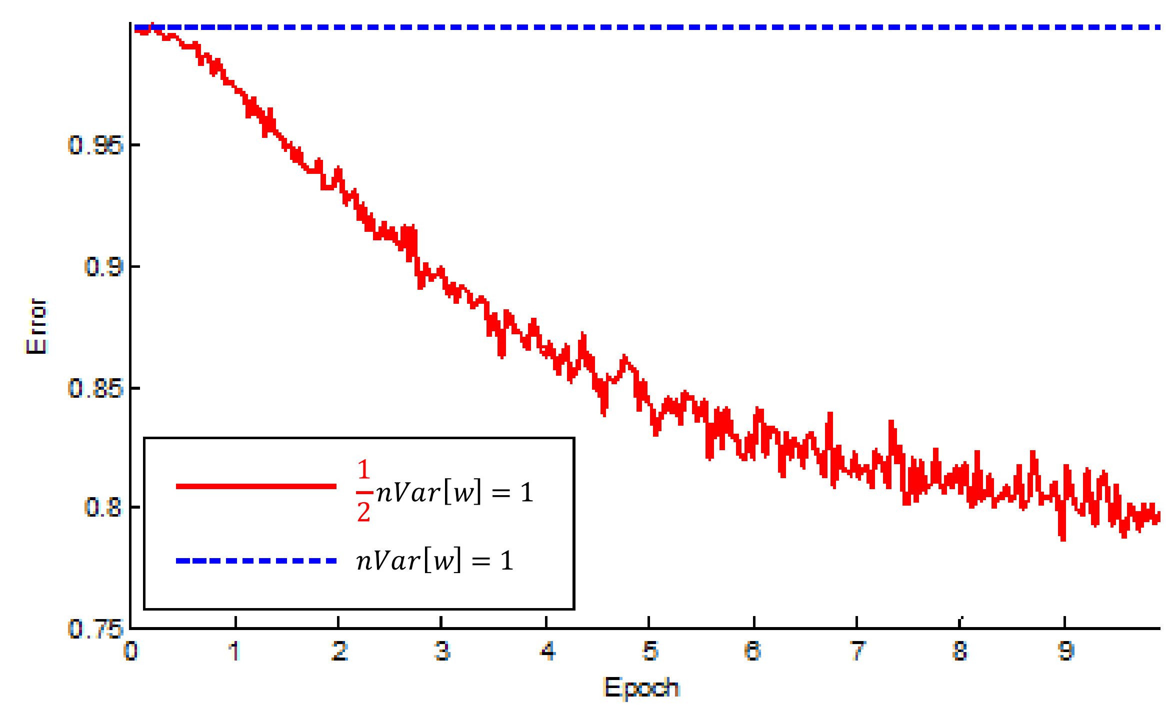
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". ICCV 2015.

1. Impact of initialization

22-layer ReLU net:
good init converges faster



30-layer ReLU net:
good init is able to converge



*Figures show the beginning of training

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". ICCV 2015.

Ultimately, coming up with good initializations is hard, worthy of deeper investigation

2 What about the step-size η, aka “learning rate”?

2. Step size tuning

Decaying

Adaptive

Architecture Sensitive

Others!

Often the most pesky parameter; tuning well can have big impact

NN toolkits use so-called "step-size Schedulers"

2. Step size tuning

Decaying

Adaptive

Architecture Sensitive

Others!

Often the most pesky parameter; tuning well can have big impact

NN toolkits use so-called "step-size Schedulers"

2. Step size tuning

Decaying

Adaptive

Architecture Sensitive

Others!

Often the most pesky parameter; tuning well can have big impact

NN toolkits use so-called "step-size Schedulers"

2. Step size tuning

Decaying

Adaptive

Architecture Sensitive

Others!

Often the most pesky parameter; tuning well can have big impact

NN toolkits use so-called "step-size Schedulers"

A Second look at Exponential and Cosine Step Sizes: Simplicity, Convergence, and Performance

Xiaoyu Li, Zhenxun Zhuang, Francesco Orabona

Layerwise Adaptive Rate Scaling: popular for large batch training

Layerwise Adaptive Rate Scaling: popular for large batch training

Algorithm 1 LARS

Input: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^T$, parameter $0 < \beta_1 < 1$, scaling function ϕ, $\epsilon > 0$

Set $m_0 = 0$

for $t = 1$ **to** T **do**

  Draw b samples S_t from $\mathbb{P}$

  Compute $g_t = \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t)$

  $m_t = \beta_1 m_{t-1} + (1 - \beta_1)(g_t + \lambda x_t)$

  $x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|m_t^{(i)}\|} m_t^{(i)}$ for all $i \in [h]$

end for

Layerwise Adaptive Rate Scaling: popular for large batch training

Algorithm 1 LARS

Input: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^T$, parameter $0 < \beta_1 < 1$, scaling function ϕ, $\epsilon > 0$

Set $m_0 = 0$

for $t = 1$ **to** T **do**

Draw b samples S_t from $\mathbb{P}$

Compute $g_t = \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t)$

$m_t = \beta_1 m_{t-1} + (1 - \beta_1)(g_t + \lambda x_t)$

$x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|m_t^{(i)}\|} m_t^{(i)}$ for all $i \in [h]$

end for

Layerwise Adaptive Rate Scaling: popular for large batch training

Algorithm 1 LARS

Input: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^T$, parameter $0 < \beta_1 < 1$, scaling function ϕ, $\epsilon > 0$

Set $m_0 = 0$

for $t = 1$ **to** T **do**

  Draw b samples S_t from $\mathbb{P}$

  Compute $g_t = \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t)$

  $m_t = \beta_1 m_{t-1} + (1 - \beta_1)(g_t + \lambda x_t)$

  $x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|m_t^{(i)}\|} m_t^{(i)}$ for all $i \in [h]$

end for

3 How to compute a stochastic gradient?

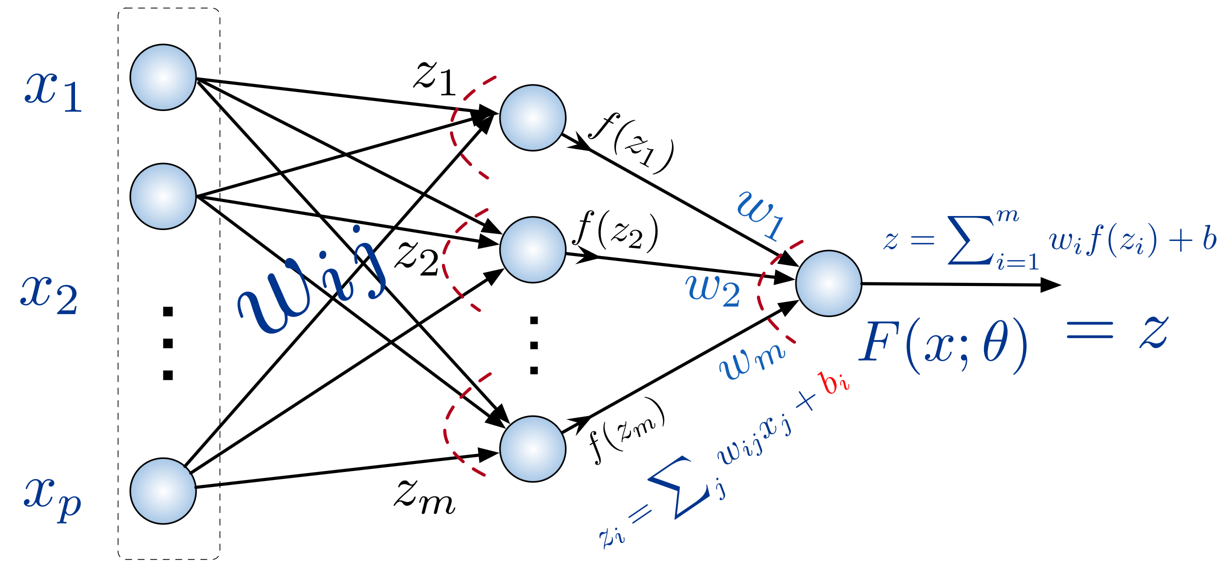
3. Computing gradients

Key computational task: compute a stochastic gradient

3. Computing gradients

Key computational task: compute a stochastic gradient

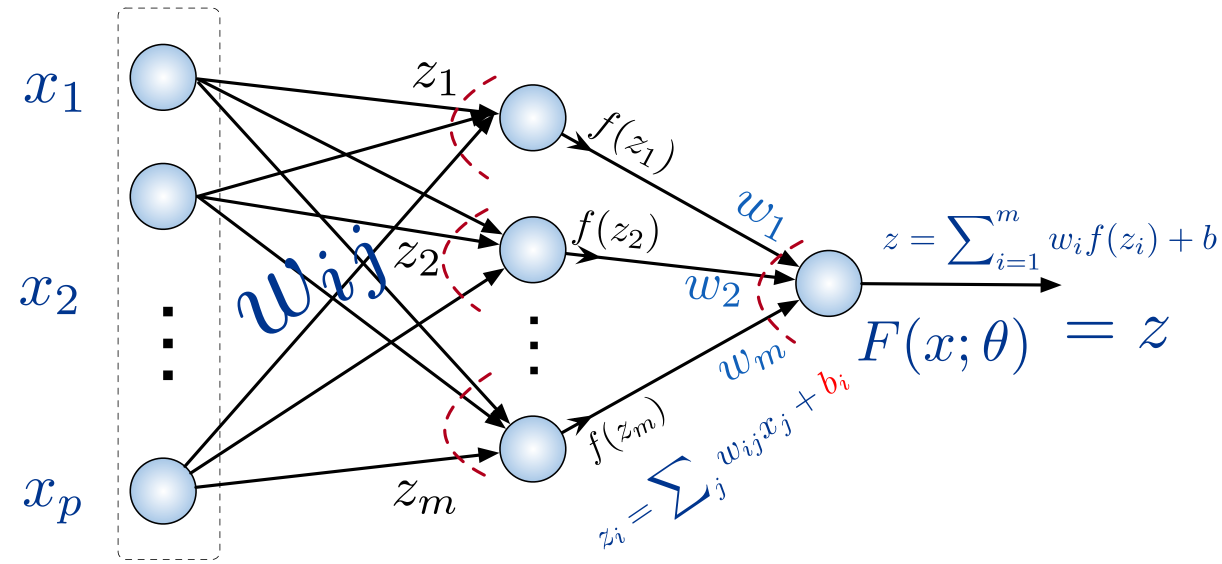
w_{ij} 1≤ i ≤ m (hidden units)
1 ≤ j ≤ p (input features)



3. Computing gradients

Key computational task: compute a stochastic gradient

w_{ij} 1≤ i ≤ m (hidden units)
1 ≤ j ≤ p (input features)



$$z_i = \sum_{j=1}^{p} w_{ij} x_j + b_i$$ input to i^{th} hidden unit

$$f(z_i) = \max(0, z_i)$$ output of i^{th} hidden unit

$$z = \sum_{i=1}^{m} w_i f(z_i) + b$$ input to output unit

$$f(z) = F(x;\theta) = z$$ network output

$$\ell(y, z) = \max(0, 1 - yz)$$

Aim: compute $\partial \ell / \partial \theta$

Computing gradients: backpropagation

$z_i = \sum_{j=1}^{p} w_{ij} x_j + b_i$	input to i^{th} hidden unit
$f(z_i) = \max(0, z_i)$	output of i^{th} hidden unit
$z = \sum_{i=1}^{m} w_i f(z_i) + b$	input to output unit
$f(z) = F(x; \theta) = z$	network output

$$\ell(y, z) = \max(0, 1 - yz)$$

Computing gradients: backpropagation

$z_i = \sum_{j=1}^{p} w_{ij} x_j + b_i$	input to i^{th} hidden unit
$f(z_i) = \max(0, z_i)$	output of i^{th} hidden unit
$z = \sum_{i=1}^{m} w_i f(z_i) + b$	input to output unit
$f(z) = F(x; \theta) = z$	network output

$$\ell(y, z) = \max(0, 1 - yz)$$

Observe that a change to w_{ij} changes z_i, which changes $f(z_i)$, which eventually changes z and thus the loss ℓ.

Computing gradients: backpropagation

$z_i = \sum_{j=1}^{p} w_{ij} x_j + b_i$	input to i^{th} hidden unit	
$f(z_i) = \max(0, z_i)$	output of i^{th} hidden unit	$\ell(y, z) = \max(0, 1 - yz)$
$z = \sum_{i=1}^{m} w_i f(z_i) + b$	input to output unit	
$f(z) = F(x; \theta) = z$	network output	

Observe that a change to w_{ij} changes z_i, which changes $f(z_i)$, which eventually changes z and thus the loss ℓ.

Chain-rule of calculus

$$\frac{\partial \ell(y,z)}{\partial w_{ij}} = \left[\frac{\partial z_i}{\partial w_{ij}}\right]\left[\frac{\partial f(z_i)}{\partial z_i}\right]\left[\frac{\partial z}{\partial f(z_i)}\right]\frac{\partial \ell}{\partial z}$$

$$= [x_j][\![z_i > 0]\!][w_i]\begin{bmatrix} -y, & \text{if } \ell(y,z) > 0, \\ 0, & \text{otherwise.} \end{bmatrix}$$

Backpropagation

Challenge: How to apply the chain rule in a deep network?

Backpropagation

Challenge: How to apply the chain rule in a deep network?

* A change to a weight w_{ij} at the first hidden layer will impact all subsequent layers.
* To apply the chain-rule, must aggregate contribution from each unit to final output
* We must **cover all paths** by which information can flow from first layer to last!
* This is where backpropagation enters the game

Backpropagation

Challenge: How to apply the chain rule in a deep network?

- A change to a weight w_{ij} at the first hidden layer will impact all subsequent layers.
- To apply the chain-rule, must aggregate contribution from each unit to final output
- We must **cover all paths** by which information can flow from first layer to last!
- This is where backpropagation enters the game

▶ *A simple, brilliant idea dating back to 1960s, and early 70s. Rediscovered multiple time; popularized greatly after 1986 paper of Rumelhart, Hinton, Williams*

Backpropagation

Challenge: How to apply the chain rule in a deep network?

- A change to a weight w_{ij} at the first hidden layer will impact all subsequent layers.
- To apply the chain-rule, must aggregate contribution from each unit to final output
- We must **cover all paths** by which information can flow from first layer to last!
- This is where backpropagation enters the game

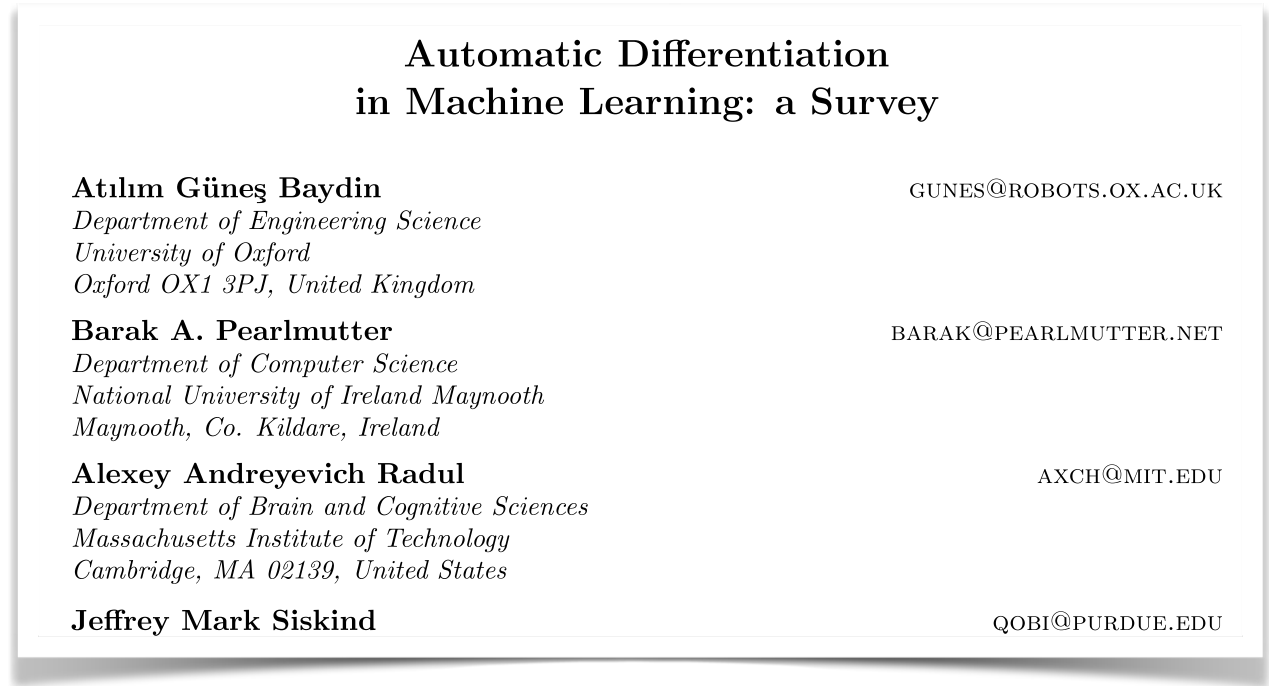
▶ *A simple, brilliant idea dating back to 1960s, and early 70s. Rediscovered multiple time; popularized greatly after 1986 paper of Rumelhart, Hinton, Williams*

Key insight: Trade space for time (dynamic programming).

Thus, keep track of how a change to the input of one layer impacts its output, and **use extra storage to save this** (*change=derivative*).

Automatic differentiation

Forward mode AD
Backward mode AD
(Backprop a special case)


Automatic Differentiation
in Machine Learning: a Survey

Atılım Güneş Baydin GUNES@ROBOTS.OX.AC.UK
Department of Engineering Science
University of Oxford
Oxford OX1 3PJ, United Kingdom

Barak A. Pearlmutter BARAK@PEARLMUTTER.NET
Department of Computer Science
National University of Ireland Maynooth
Maynooth, Co. Kildare, Ireland

Alexey Andreyevich Radul AXCH@MIT.EDU
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, United States

Jeffrey Mark Siskind QOBI@PURDUE.EDU

Optimal Jacobian Accumulation: NP-Complete

All NN toolkits use autodiff libraries

AD: Generate algorithm for efficient evaluation of derivatives

Numerous tutorials and notes online; well-developed area in PL and numerics

4 In reality: BN, momentum,clipping,adaptivity and many other ideas!

Key motivation: unstable gradients

$$\delta^l = \frac{\partial \ell}{\partial z^l} = \text{Diag}[f'(z^l)]W^{l+1}\delta^{l+1}.$$

$$\delta^l = \text{Diag}[f'(z^l)]W^{l+1}\text{Diag}[f'(z^{l+1})]W^{l+2}\cdots W^L\delta^L$$

Observations

Key motivation: unstable gradients

$$\delta^l = \frac{\partial \ell}{\partial z^l} = \mathrm{Diag}[f'(z^l)]W^{l+1}\delta^{l+1}.$$

$$\delta^l = \mathrm{Diag}[f'(z^l)]W^{l+1}\mathrm{Diag}[f'(z^{l+1})]W^{l+2}\cdots W^L\delta^L$$

Observations

- Multiplication of a chain of matrices in backprop

Key motivation: unstable gradients

$$\delta^l = \frac{\partial \ell}{\partial z^l} = \mathrm{Diag}[f'(z^l)]W^{l+1}\delta^{l+1}.$$

$$\delta^l = \mathrm{Diag}[f'(z^l)]W^{l+1}\mathrm{Diag}[f'(z^{l+1})]W^{l+2}\cdots W^L\delta^L$$

Observations

- Multiplication of a chain of matrices in backprop
- If several of these matrices are "small" (i.e., norms < 1), when we multiply them, the gradient will decrease exponentially fast and tend to *vanish* (hurting learning in lower layers much more)

Key motivation: unstable gradients

$$\delta^l = \frac{\partial \ell}{\partial z^l} = \mathrm{Diag}[f'(z^l)]W^{l+1}\delta^{l+1}.$$

$$\delta^l = \mathrm{Diag}[f'(z^l)]W^{l+1}\mathrm{Diag}[f'(z^{l+1})]W^{l+2}\cdots W^L\delta^L$$

Observations

- Multiplication of a chain of matrices in backprop
- If several of these matrices are "small" (i.e., norms < 1), when we multiply them, the gradient will decrease exponentially fast and tend to *vanish* (hurting learning in lower layers much more)
- Conversely, if several matrices have large norm, the gradient will tend to *explode*. In both cases, the gradients are unstable.

Key motivation: unstable gradients

$$\delta^l = \frac{\partial \ell}{\partial z^l} = \mathrm{Diag}[f'(z^l)]W^{l+1}\delta^{l+1}.$$

$$\delta^l = \mathrm{Diag}[f'(z^l)]W^{l+1}\mathrm{Diag}[f'(z^{l+1})]W^{l+2}\cdots W^L\delta^L$$

Observations

- Multiplication of a chain of matrices in backprop
- If several of these matrices are "small" (i.e., norms < 1), when we multiply them, the gradient will decrease exponentially fast and tend to *vanish* (hurting learning in lower layers much more)
- Conversely, if several matrices have large norm, the gradient will tend to *explode*. In both cases, the gradients are unstable.
- Coping with unstable gradients poses several challenges, and must be dealt with to achieve good results.

Partial remedies for unstable gradients

Partial remedies for unstable gradients

- ➡ Regularization (numerous ways, implicit and explicit)

Partial remedies for unstable gradients

- Regularization (numerous ways, implicit and explicit)
- ReLU activations

Partial remedies for unstable gradients

- Regularization (numerous ways, implicit and explicit)
- ReLU activations
- Memory (in RNNS)

Partial remedies for unstable gradients

- Regularization (numerous ways, implicit and explicit)
- ReLU activations
- Memory (in RNNS)
- Weight normalization and batch normalization (somewhat)

Partial remedies for unstable gradients

- Regularization (numerous ways, implicit and explicit)
- ReLU activations
- Memory (in RNNS)
- Weight normalization and batch normalization (somewhat)
- Gradient clipping, normalized gradients

Partial remedies for unstable gradients

- Regularization (numerous ways, implicit and explicit)
- ReLU activations
- Memory (in RNNS)
- Weight normalization and batch normalization (somewhat)
- Gradient clipping, normalized gradients
- Numerous other ideas (architecture specific)

Partial remedies for unstable gradients

- Regularization (numerous ways, implicit and explicit)
- ReLU activations
- Memory (in RNNS)
- Weight normalization and batch normalization (somewhat)
- Gradient clipping, normalized gradients
- Numerous other ideas (architecture specific)
- Residual Networks (Resnets)

Regularization

$$+ \lambda ||\theta||^2$$

definitely use it; but many other ways too!

Regularization

$$+ \lambda ||\theta||^2$$

definitely use it; but many other ways too!

NN folks call this: "*weight decay*," though to be pedantic, some reserve the term "weight decay" for the part subtracted from weights θ when updating them (e.g., ADAMW optimizer)

Regularizing with Dropout

Motivation

- When fitting to the nitty-gritty of the input, including noise hidden units must rely on each other to co-adapt and have complementary coverage of the data space.
- To hinder fitting to noise we must avoid overdoing co-adaptation

Regularizing with Dropout

Motivation

- When fitting to the nitty-gritty of the input, including noise hidden units must rely on each other to co-adapt and have complementary coverage of the data space.
- To hinder fitting to noise we must avoid overdoing co-adaptation

Dropout *(additional stochasticity in the loss function)*

Regularizing with Dropout

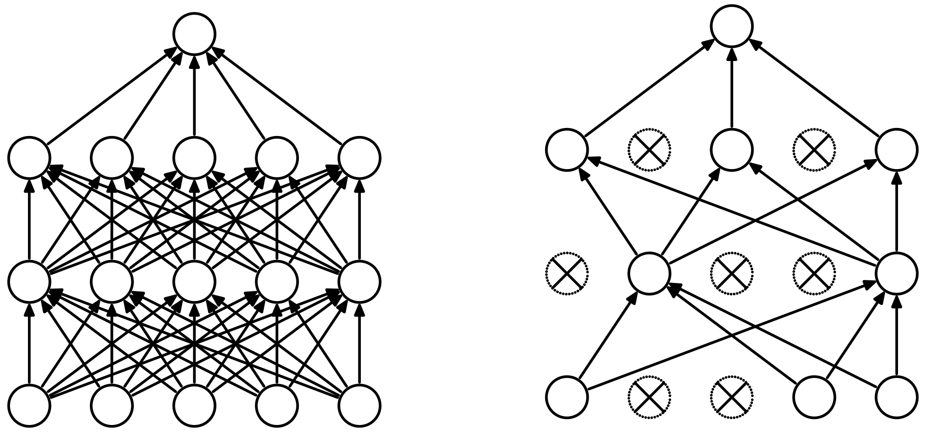
Motivation

- When fitting to the nitty-gritty of the input, including noise hidden units must rely on each other to co-adapt and have complementary coverage of the data space.
- To hinder fitting to noise we must avoid overdoing co-adaptation

Dropout *(additional stochasticity in the loss function)*

- **Randomly turn off units, say with probability 1/2, when training!**

figure from the [dropout] paper



(a) Standard Neural Net

(b) After applying dropout.

Regularizing with Dropout

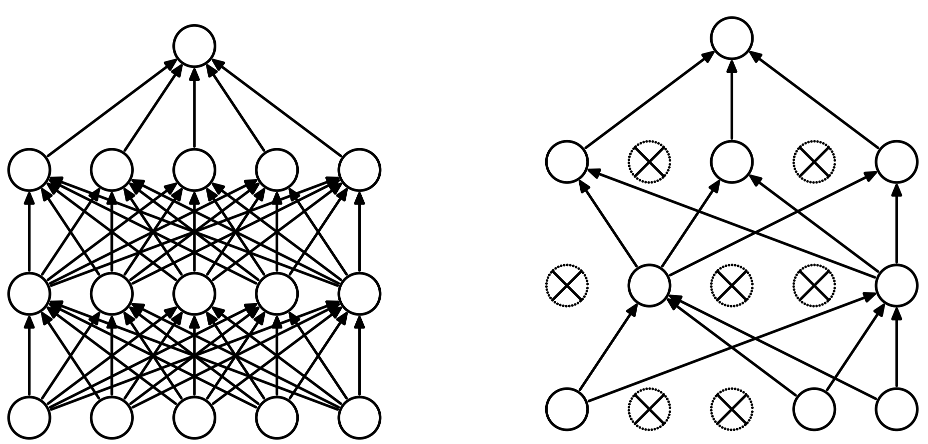
Motivation

- When fitting to the nitty-gritty of the input, including noise hidden units must rely on each other to co-adapt and have complementary coverage of the data space.
- To hinder fitting to noise we must avoid overdoing co-adaptation

Dropout *(additional stochasticity in the loss function)*

- **Randomly turn off units, say with probability 1/2, when training!**
  - For each data point, we **randomly** set the output of each hidden unit to zero.

figure from the [dropout] paper



(a) Standard Neural Net (b) After applying dropout.

Regularizing with Dropout

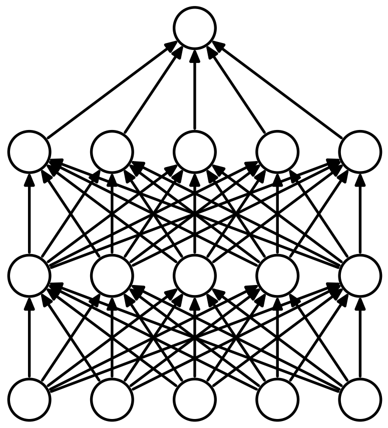
Motivation

- When fitting to the nitty-gritty of the input, including noise hidden units must rely on each other to co-adapt and have complementary coverage of the data space.
- To hinder fitting to noise we must avoid overdoing co-adaptation

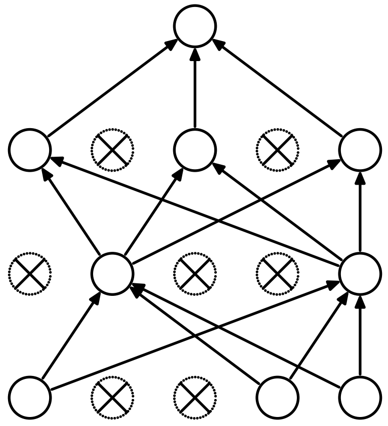
Dropout *(additional stochasticity in the loss function)*

- **Randomly turn off units, say with probability 1/2, when training!**
  - For each data point, we **randomly** set the output of each hidden unit to zero.
  - The neurons turned off are randomly chosen anew for each training data point

figure from the [dropout] paper



(a) Standard Neural Net



(b) After applying dropout.

Regularizing with Dropout

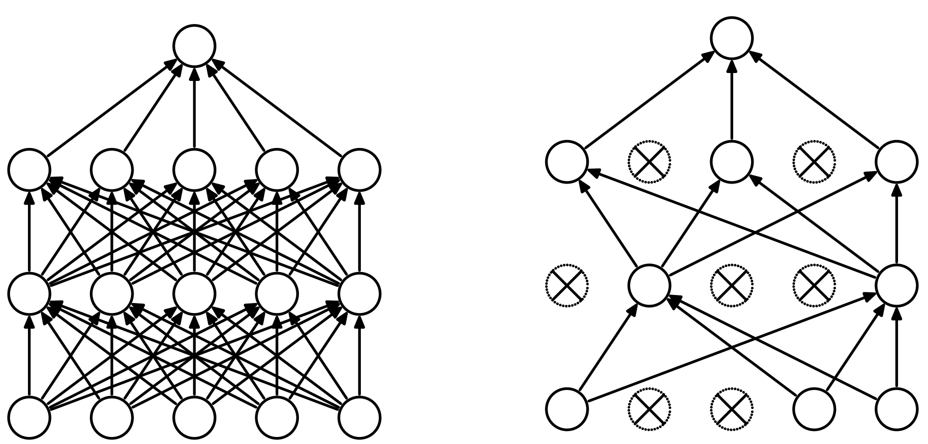
Motivation

- When fitting to the nitty-gritty of the input, including noise hidden units must rely on each other to co-adapt and have complementary coverage of the data space.
- To hinder fitting to noise we must avoid overdoing co-adaptation

Dropout *(additional stochasticity in the loss function)*

- **Randomly turn off units, say with probability 1/2, when training!**
  - For each data point, we **randomly** set the output of each hidden unit to zero.
  - The neurons turned off are randomly chosen anew for each training data point
  - Accounted for during backprop (**how?**).

figure from the [dropout] paper



(a) Standard Neural Net (b) After applying dropout.

Regularizing with Dropout

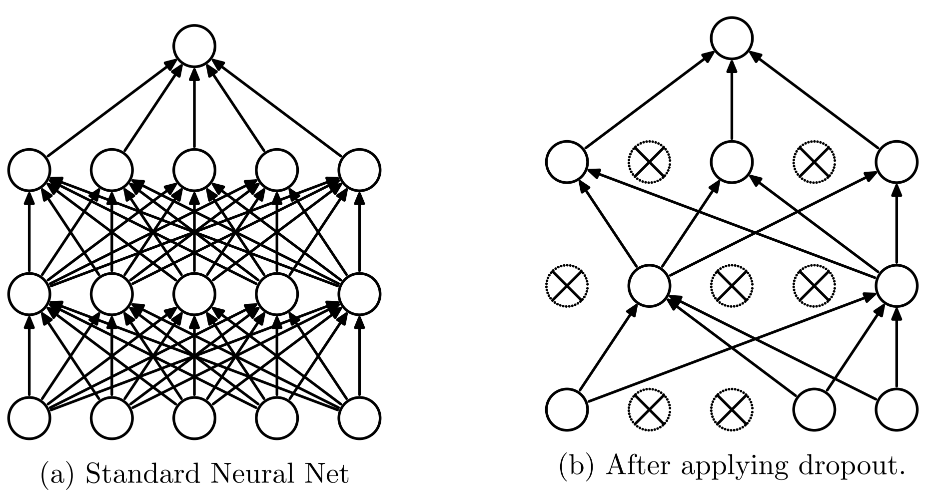
Motivation

- When fitting to the nitty-gritty of the input, including noise hidden units must rely on each other to co-adapt and have complementary coverage of the data space.
- To hinder fitting to noise we must avoid overdoing co-adaptation

Dropout *(additional stochasticity in the loss function)*

- **Randomly turn off units, say with probability 1/2, when training!**
  - For each data point, we **randomly** set the output of each hidden unit to zero.
  - The neurons turned off are randomly chosen anew for each training data point
  - Accounted for during backprop (**how?**).
  - For units turned off for that round, input weights and activations **not** updated; unit effectively dropped out for that particular training sample. This additional stochasticity helps in regularization. **Explore:** other ways of adding stochasticity to NN training

figure from the [dropout] paper



Batch Normalization

Batch Normalization

Observation: Known that training converges faster if inputs “whitened”, i.e., linearly transformed to have mean zero, unit variance, and decorrelated.

Batch Normalization

Observation: Known that training converges faster if inputs “whitened”, i.e., linearly transformed to have mean zero, unit variance, and decorrelated.

Idea 0: Activations of one layer, inputs to another. If we do similar whitening of the inputs of each layer might help towards improving training.

Batch Normalization

Observation: Known that training converges faster if inputs “whitened”, i.e., linearly transformed to have mean zero, unit variance, and decorrelated.

Idea 0: Activations of one layer, inputs to another. If we do similar whitening of the inputs of each layer might help towards improving training.

Full whitening involves inverting large matrices, a no-go

Batch Normalization

Observation: Known that training converges faster if inputs “whitened”, i.e., linearly transformed to have mean zero, unit variance, and decorrelated.

Idea 0: Activations of one layer, inputs to another. If we do similar whitening of the inputs of each layer might help towards improving training.

Full whitening involves inverting large matrices, a no-go

Idea 1: Normalize features individually, not jointly

$$x = (x^1, \ldots, x^p) \qquad \hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$

(features at a layer)

Batch Normalization

Observation: Known that training converges faster if inputs “whitened”, i.e., linearly transformed to have mean zero, unit variance, and decorrelated.

Idea 0: Activations of one layer, inputs to another. If we do similar whitening of the inputs of each layer might help towards improving training.

Full whitening involves inverting large matrices, a no-go

Idea 1: Normalize features individually, not jointly

$$x = (x^1, \ldots, x^p)$$

(features at a layer)

$$\hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$

Expectation and Variance computed over training data set (LeCun98— this speeds up training)

Batch Normalization

Batch Normalization

Idea 1: Normalize features individually, not jointly

$$x = (x^1, \ldots, x^p) \qquad \hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$

(features at a layer)

Batch Normalization

Idea 1: Normalize features individually, not jointly

$$x = (x^1, \ldots, x^p)$$

(features at a layer)

$$\hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$

Expectation and Variance computed over training data set (LeCun98— this speeds up training)

Batch Normalization

Idea 1: Normalize features individually, not jointly

$$x = (x^1, \ldots, x^p) \qquad \hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$

(features at a layer)

Expectation and Variance computed over training data set (LeCun98— this speeds up training)

Idea 1: mini-batch normalization

🤓

Batch Normalization

Idea 1: Normalize features individually, not jointly

$$x = (x^1, \ldots, x^p)$$

(features at a layer)

$$\hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$

Expectation and Variance computed over training data set (LeCun98— this speeds up training)

Idea 1: mini-batch normalization

BN transform applied to activation *x* over a mini-batch

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

Idea 2: Restore representation power" / Undo damage by learning γ and β

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

Idea 2: Restore representation power" / Undo damage by learning γ and β

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

Idea 2: Restore representation power" / Undo damage by learning γ and β

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

Idea 2: Restore representation power" / Undo damage by learning γ and β

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Intuition: Allow the transformation to represent the identity (this idea recurs)

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

Idea 2: Restore representation power" / Undo damage by learning γ and β

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Intuition: Allow the transformation to represent the identity (this idea recurs)

Exercise: Derive backprop rules to figure out how to update scale γ and shift β

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

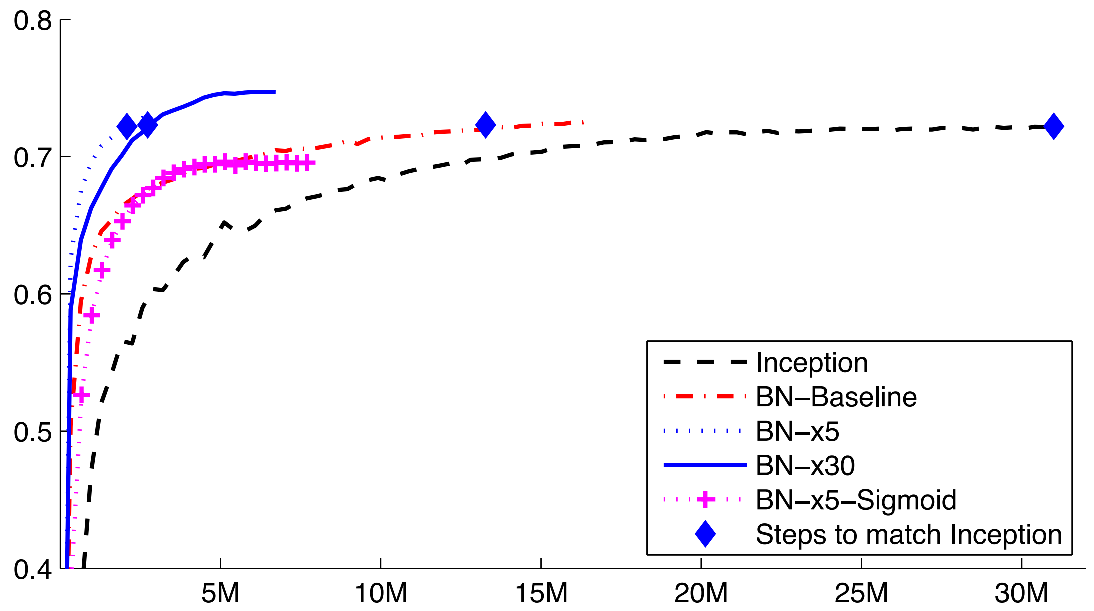


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

(several other speedups enabled, and used for this plot)

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

- ✓ BN layer can be added to many networks (e.g., CNNs, Resnets, etc.)
  - ➡ *Current Challenge*: BN for RNNs; also, is BN truly necessary?
- ✓ BN enables higher learning rates: backprop through a BN layer is unaffected by the scale of its parameters, BN(Wx)=BN((aW)x)
- ✓ BN has a regularizing effect (Dropout can even be dropped out)
- ✓ **Challenge:** Formally understand and explain BN

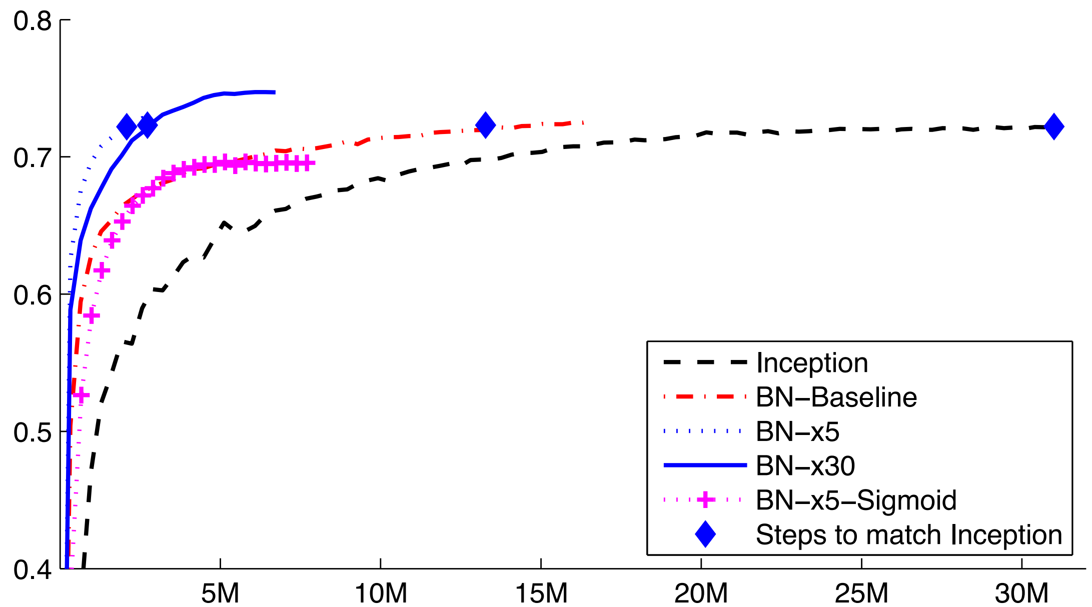


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

(several other speedups enabled, and used for this plot)

figure: [Ioffe, Szegedy, 2015]

Residual Networks (Resnets)

Residual Networks (Resnets)

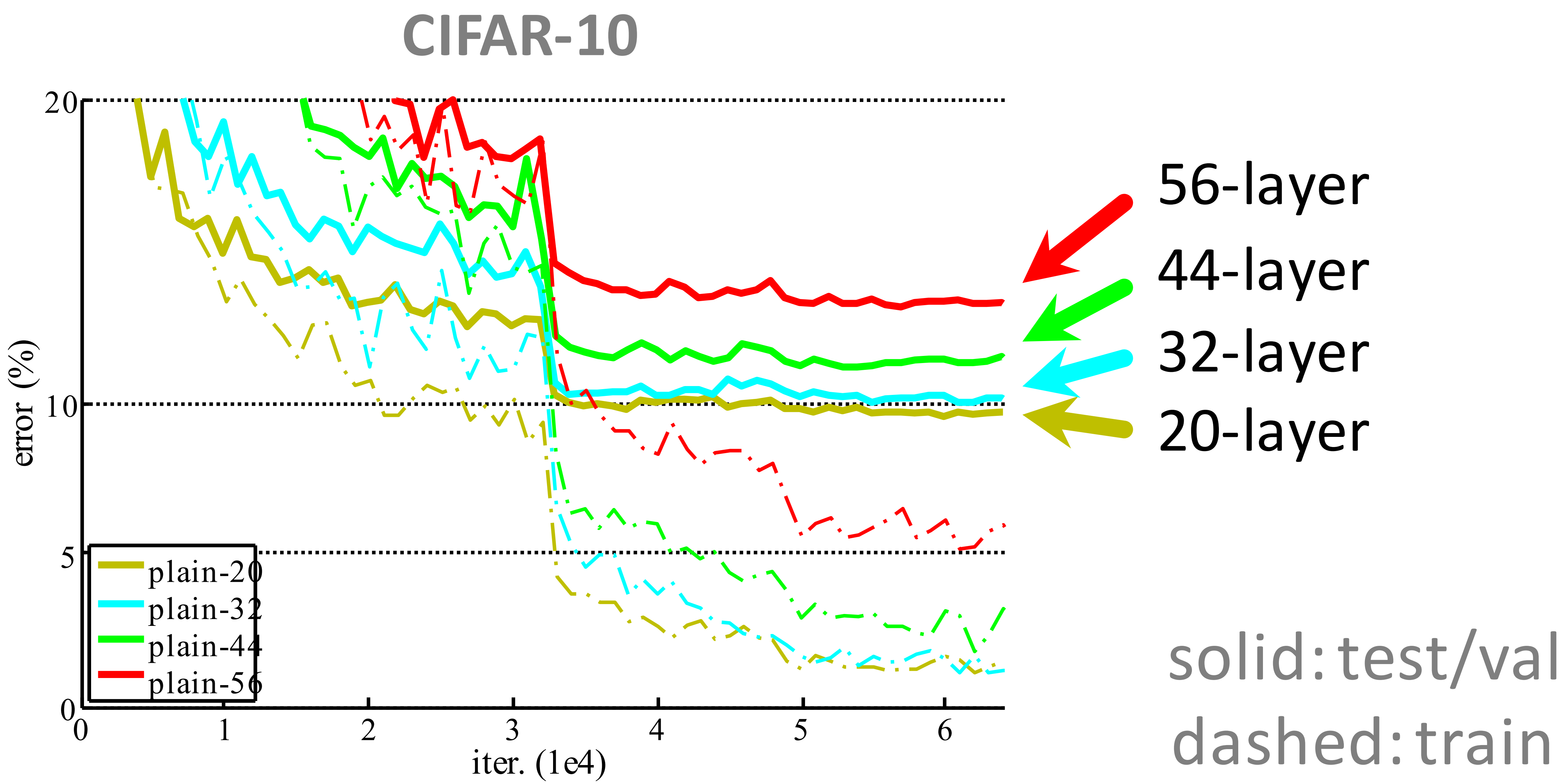
$$x \mapsto h_L \circ h_{L-1} \circ \cdots \circ h_1(x)$$

$$h_i(z) := z + \sigma(W_i z + b_i)$$

Id + σ(.)

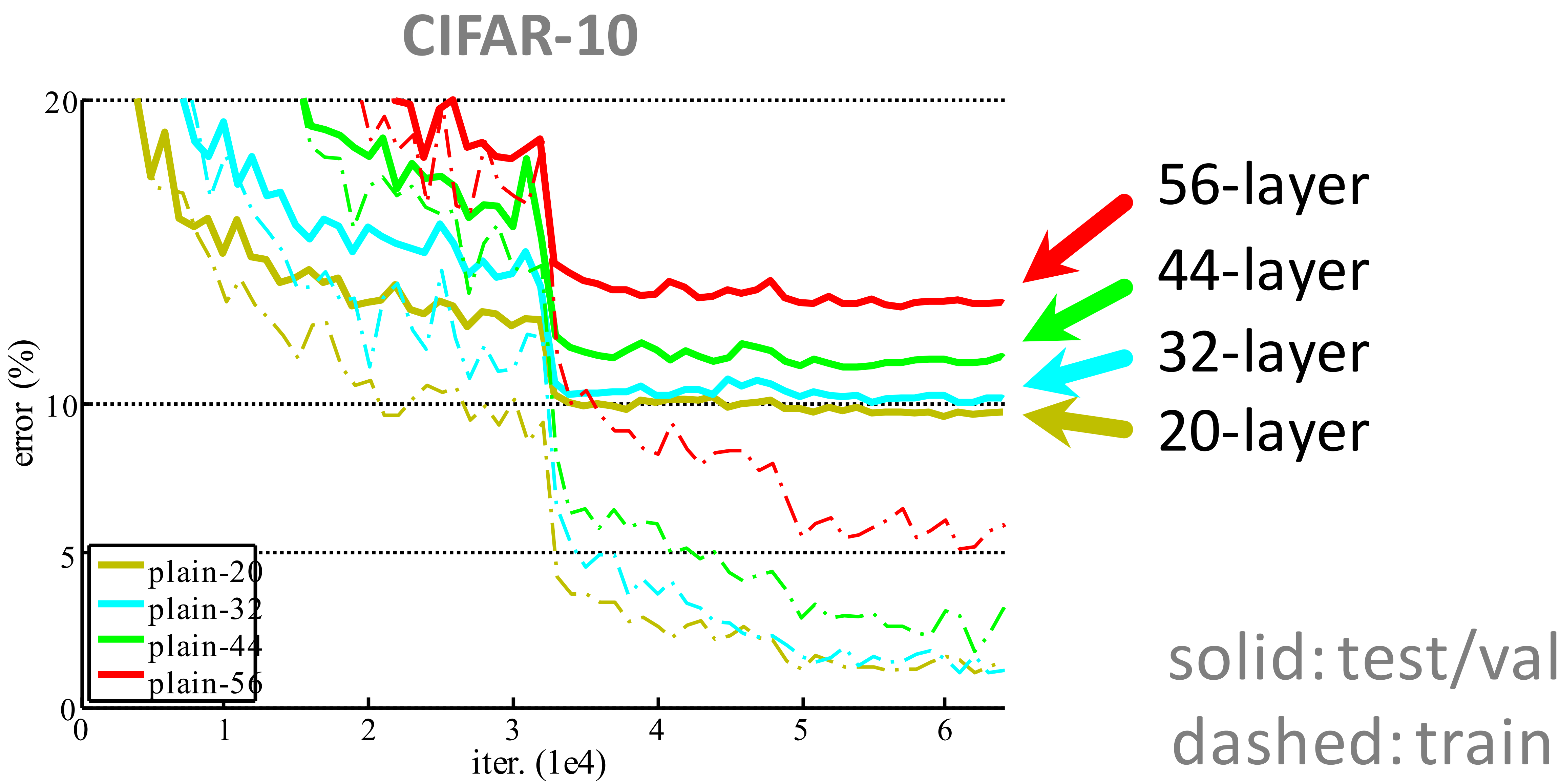
Note: Without the Identity map (Id), we are back to the usual model

Why resnets?



CIFAR-10
20
10
5
0
error (%)
0
1
2
3
4
5
6
iter. (1e4)
plain-20
plain-32
plain-44
plain-56
56-layer
44-layer
32-layer
20-layer
solid: test/val
dashed: train

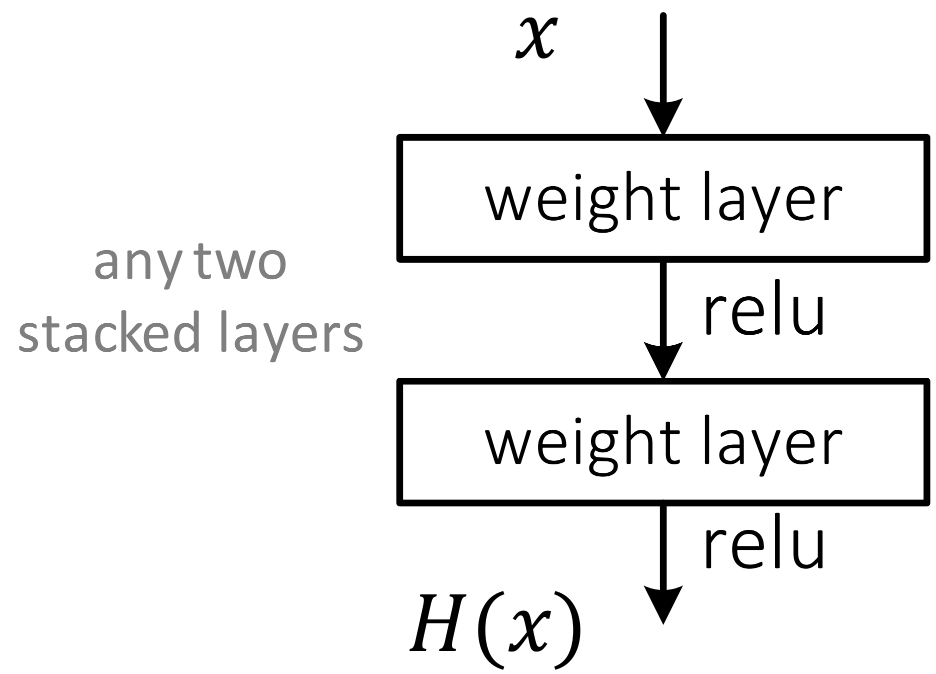
Why resnets?



Making network deeper does not necessarily work better

Limits on what initialization and batch normalization give us

Key idea: Identity maps

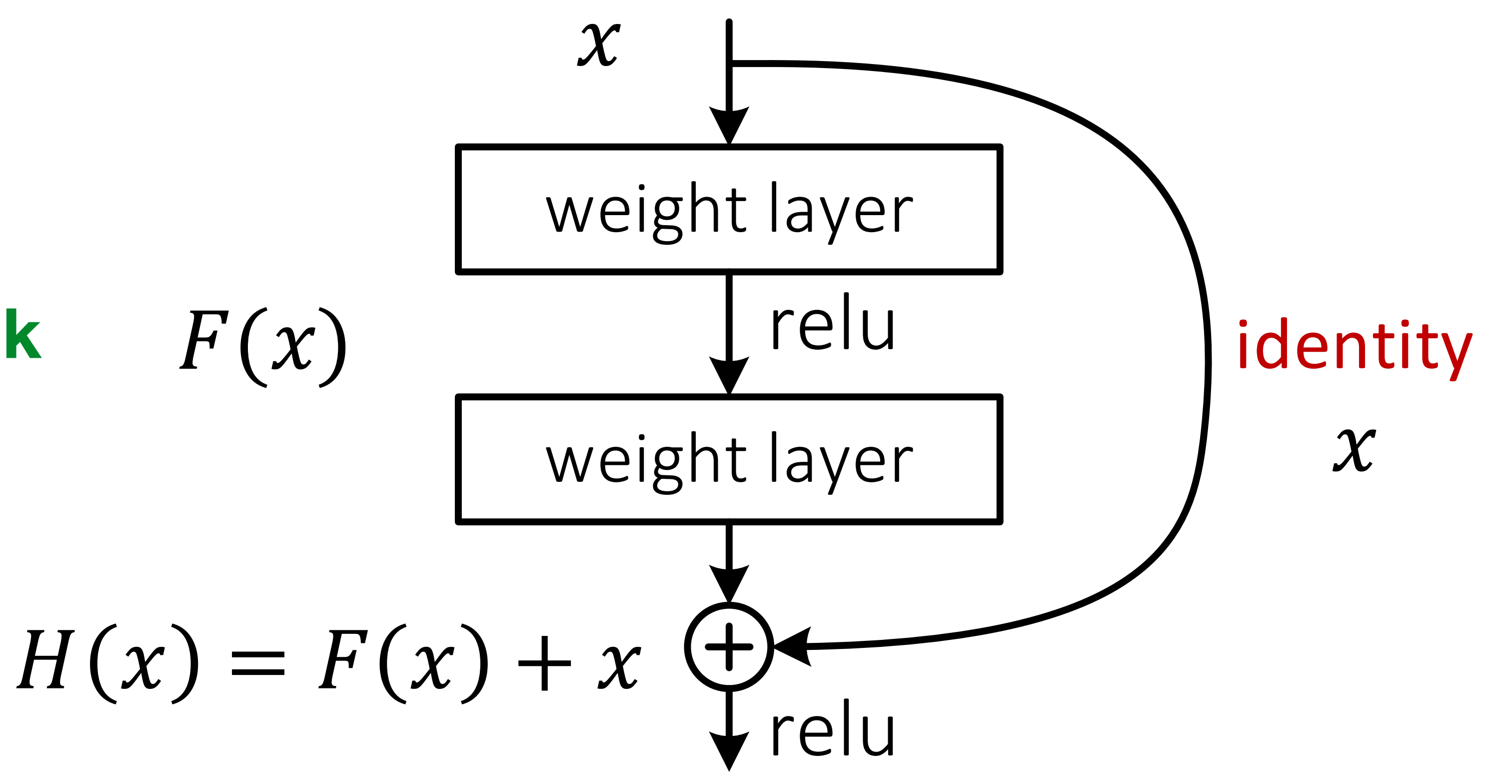


Aim: Learn map H(x).
Approach: Hope the deep net fits H(x)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

Key idea: Identity maps

A residual block

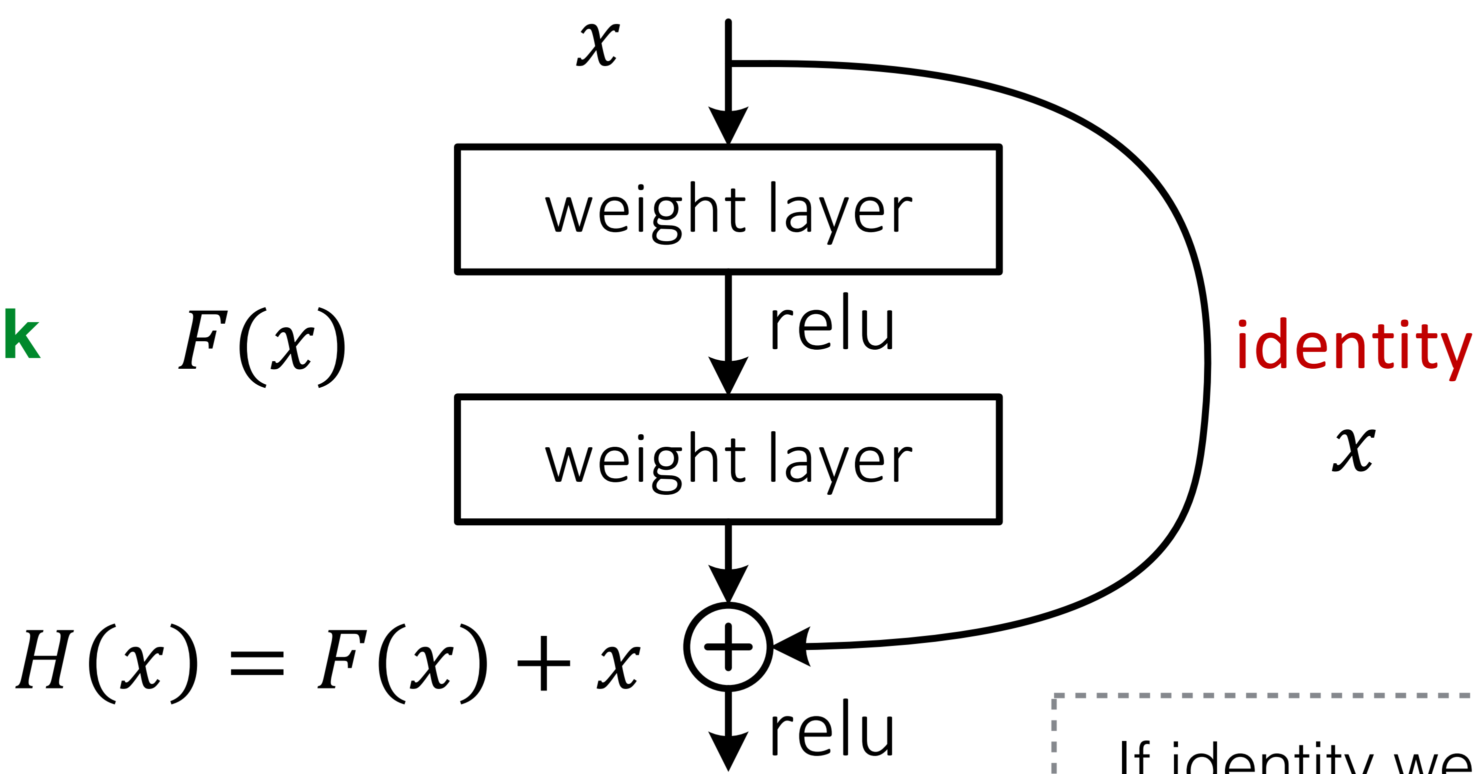


Aim: Learn map H(x) = F(x)+x
Approach: Hope the deep net fits F(x)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Key idea: Identity maps

A residual block



If identity were optimal easy to fit by setting weights=0

By adding Id, increasing depth should not hurt performance…

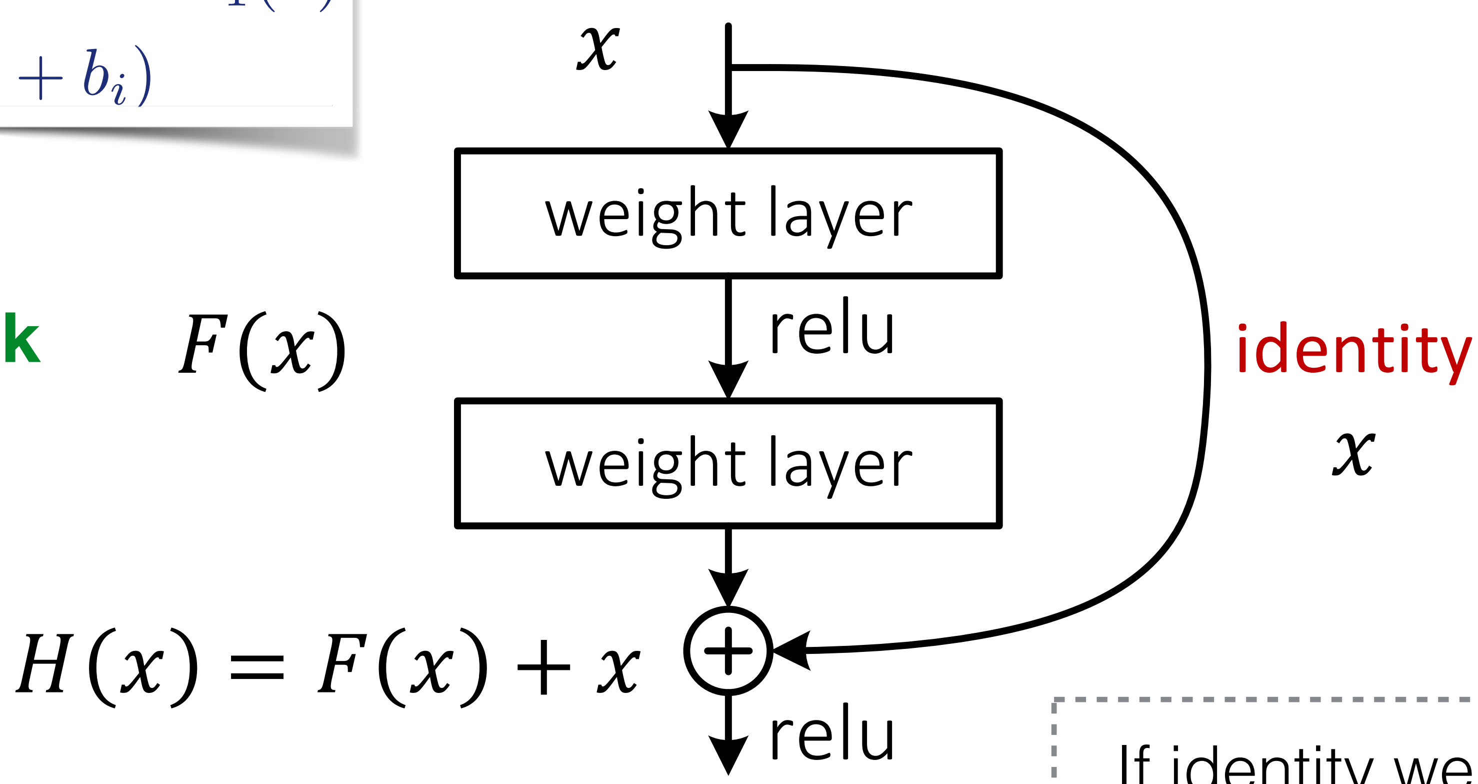
Aim: Learn map H(x) = F(x)+x
Approach: Hope the deep net fits F(x)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Key idea: Identity maps

$$x \mapsto h_L \circ h_{L-1} \circ \cdots \circ h_1(x)$$
$$h_i(z) := z + \sigma(W_i z + b_i)$$

A residual block



If identity were optimal easy to fit by setting weights=0

By adding Id, increasing depth should not hurt performance…

Aim: Learn map H(x) = F(x)+x
Approach: Hope the deep net fits F(x)

F(x) is a **residual** mapping wrt identity

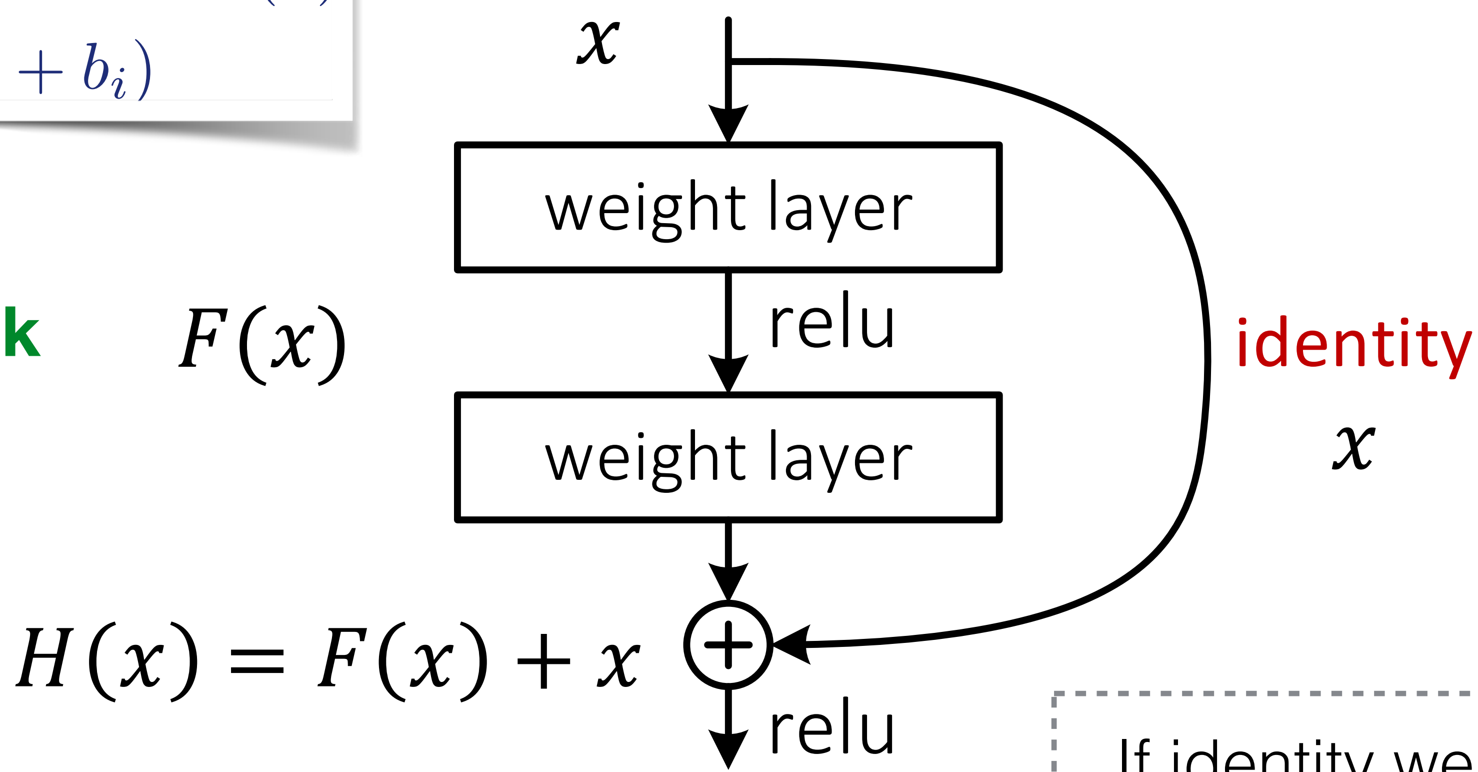
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Key idea: Identity maps

$$x \mapsto h_L \circ h_{L-1} \circ \cdots \circ h_1(x)$$

$$h_i(z) := z + \sigma(W_i z + b_i)$$

A residual block



If identity were optimal easy to fit by setting weights=0

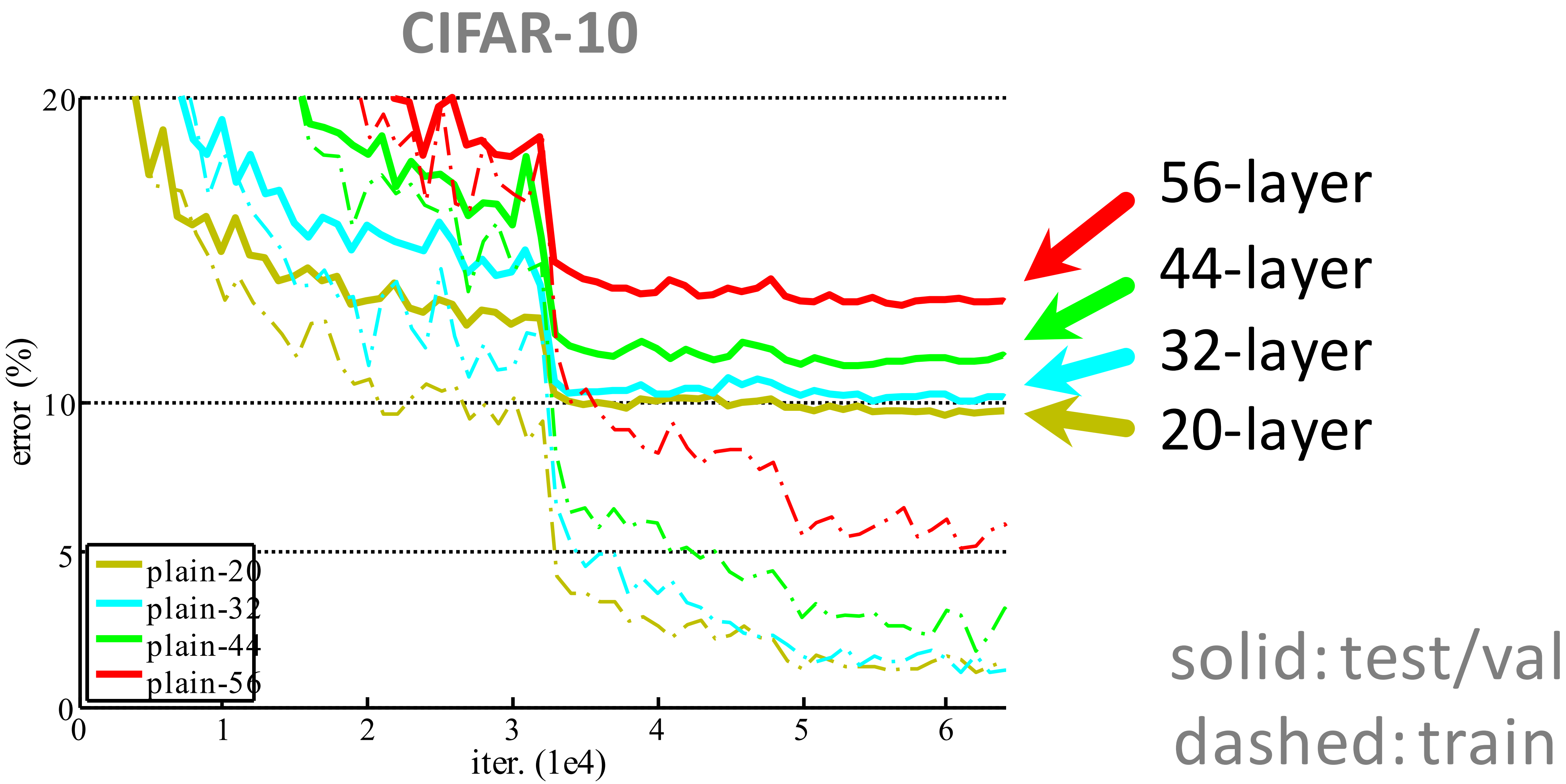
By adding Id, increasing depth should not hurt performance…

Aim: Learn map H(x) = F(x)+x
Approach: Hope the deep net fits F(x)

F(x) is a **residual** mapping wrt identity

Explore: Try residual wrt other distinguished (i.e., not Id) mappings

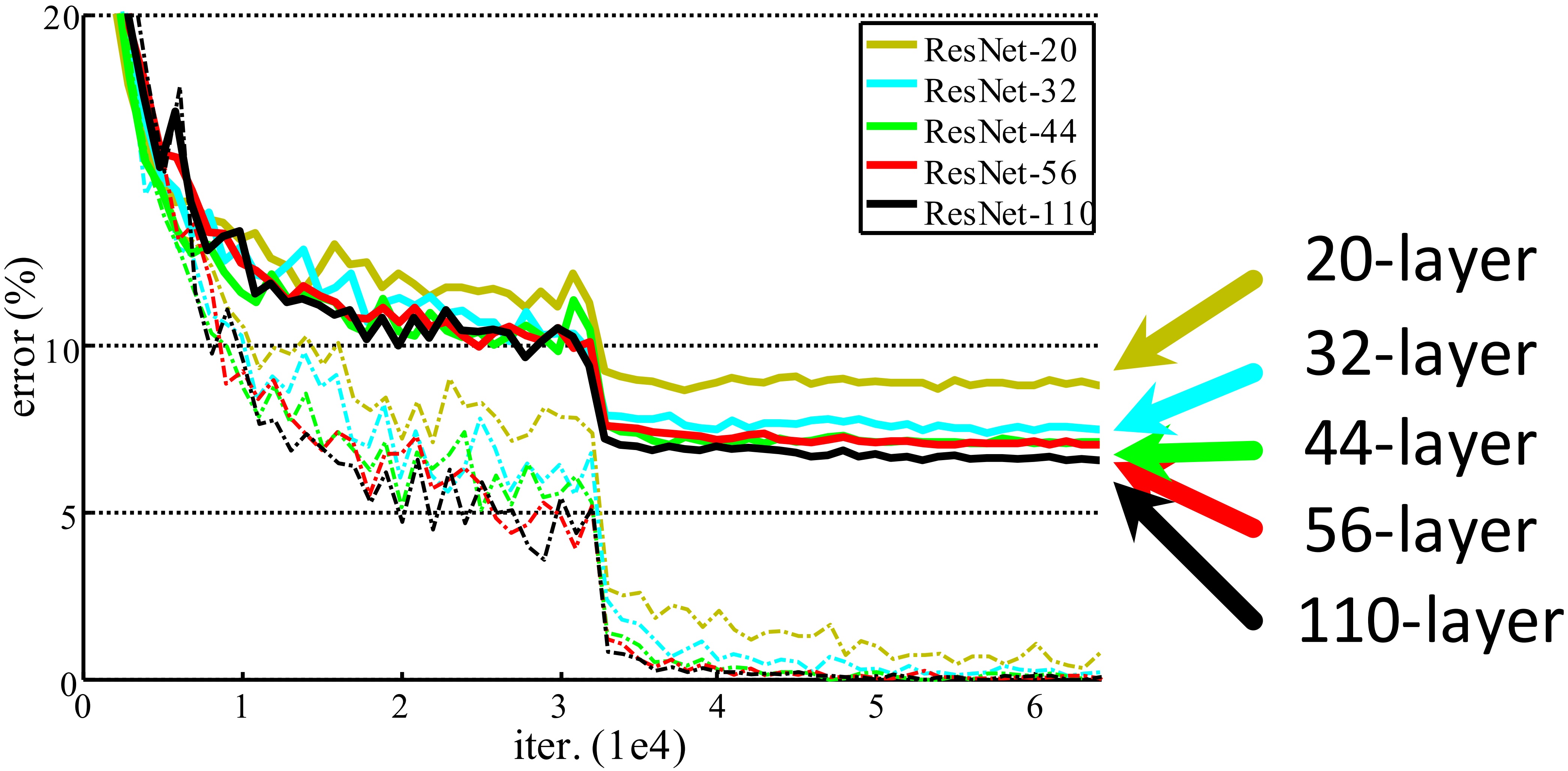
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.



solid: test/val
dashed: train

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

CIFAR-10 ResNets



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Recent theory on ResNets

- *Bartlett et al, 2018*. Optimization properties of deep residual networks.
- *Hardt, Ma 2017*. Global optimality of deep linear resnets $y=(I+W_L)(I+W_{L-1})\ldots(I+W_1)x$

Recent theory on ResNets

- *Bartlett et al, 2018*. Optimization properties of deep residual networks.
- *Hardt, Ma 2017*. Global optimality of deep linear resnets $y=(I+W_L)(I+W_{L-1})\ldots(I+W_1)x$
- *Lin, Jegelka, 2018*. ResNet with one-neuron hidden layers is a Universal Approximator (deep Resnet with one neuron per hidden layer and ReLU activation).

Recent theory on ResNets

- *Bartlett et al, 2018*. Optimization properties of deep residual networks.
- *Hardt, Ma 2017*. Global optimality of deep linear resnets $y=(I+W_L)(I+W_{L-1})\dots(I+W_1)x$
- *Lin, Jegelka, 2018*. ResNet with one-neuron hidden layers is a Universal Approximator (deep Resnet with one neuron per hidden layer and ReLU activation).
- *Shamir, 2018*. Considers $x \mapsto w^T(x + VF_\theta(x))$ and shows that every local optimum of this Resnet (with final purely linear layer) is "better than" a simple linear model. Presents some conditions under which one can prove that adding the Id map does not hurt performance.

Recent theory on ResNets

- *Bartlett et al, 2018*. Optimization properties of deep residual networks.
- *Hardt, Ma 2017*. Global optimality of deep linear resnets $y=(I+W_L)(I+W_{L-1})\dots(I+W_1)x$

- *Lin, Jegelka, 2018*. ResNet with one-neuron hidden layers is a Universal Approximator (deep Resnet with one neuron per hidden layer and ReLU activation).

- *Shamir, 2018*. Considers $x \mapsto w^T(x + VF_\theta(x))$ and shows that every local optimum of this Resnet (with final purely linear layer) is "better than" a simple linear model. Presents some conditions under which one can prove that adding the Id map does not hurt performance.

- *Yun, Sra, Jadbabaie, 2019*. Deep ResNet can be provably better than linear models (provides a "deep" version of Shamir's result above, result leaves open problems.

Recent theory on ResNets

- *Bartlett et al, 2018*. Optimization properties of deep residual networks.
- *Hardt, Ma 2017*. Global optimality of deep linear resnets $y=(I+W_L)(I+W_{L-1})\dots(I+W_1)x$
- *Lin, Jegelka, 2018*. ResNet with one-neuron hidden layers is a Universal Approximator (deep Resnet with one neuron per hidden layer and ReLU activation).
- *Shamir, 2018*. Considers $x \mapsto w^T(x + VF_\theta(x))$ and shows that every local optimum of this Resnet (with final purely linear layer) is "better than" a simple linear model. Presents some conditions under which one can prove that adding the Id map does not hurt performance.
- *Yun, Sra, Jadbabaie, 2019*. Deep ResNet can be provably better than linear models (provides a "deep" version of Shamir's result above, result leaves open problems.
- *Allen-Zhu, Li, 2019*. "What can ResNet learn efficiently, Going beyond Kernels?"