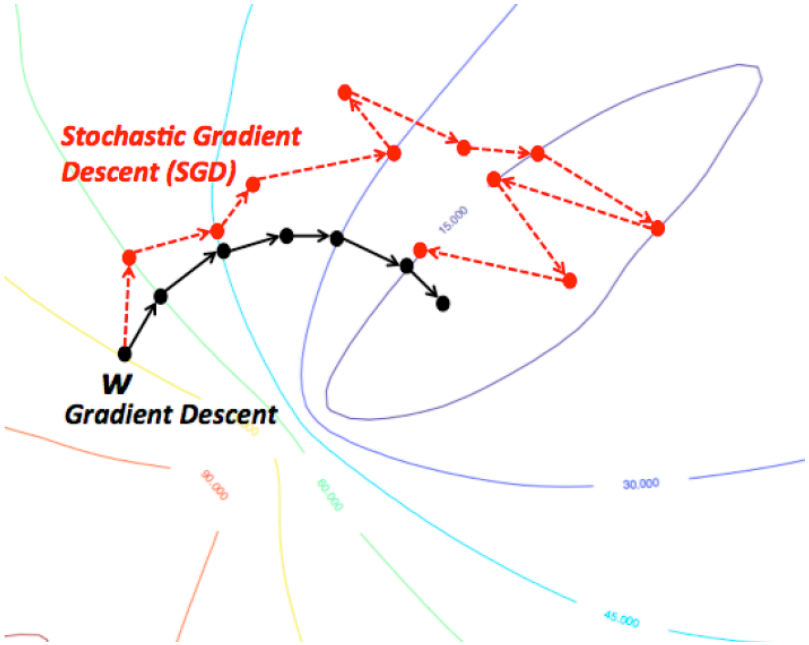
Introduction to Supervised Learning



Week 9: Optimization and Regularization for Deep Learning

Iasonas Kokkinos

i.kokkinos@cs.ucl.ac.uk

University College London

Deep learning, AD 1993

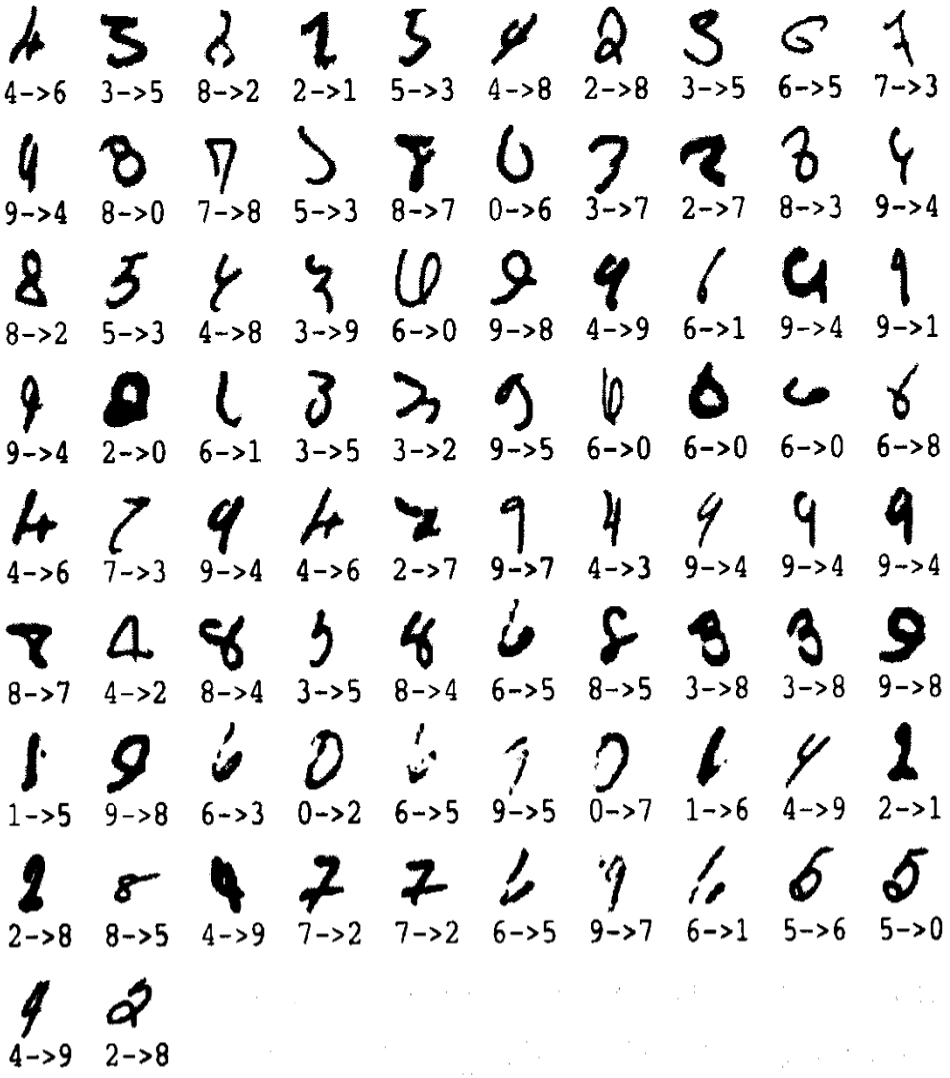
https://www.youtube.com/watch?v=FwFduRA_L6Q

(LeNet 1, 1993)

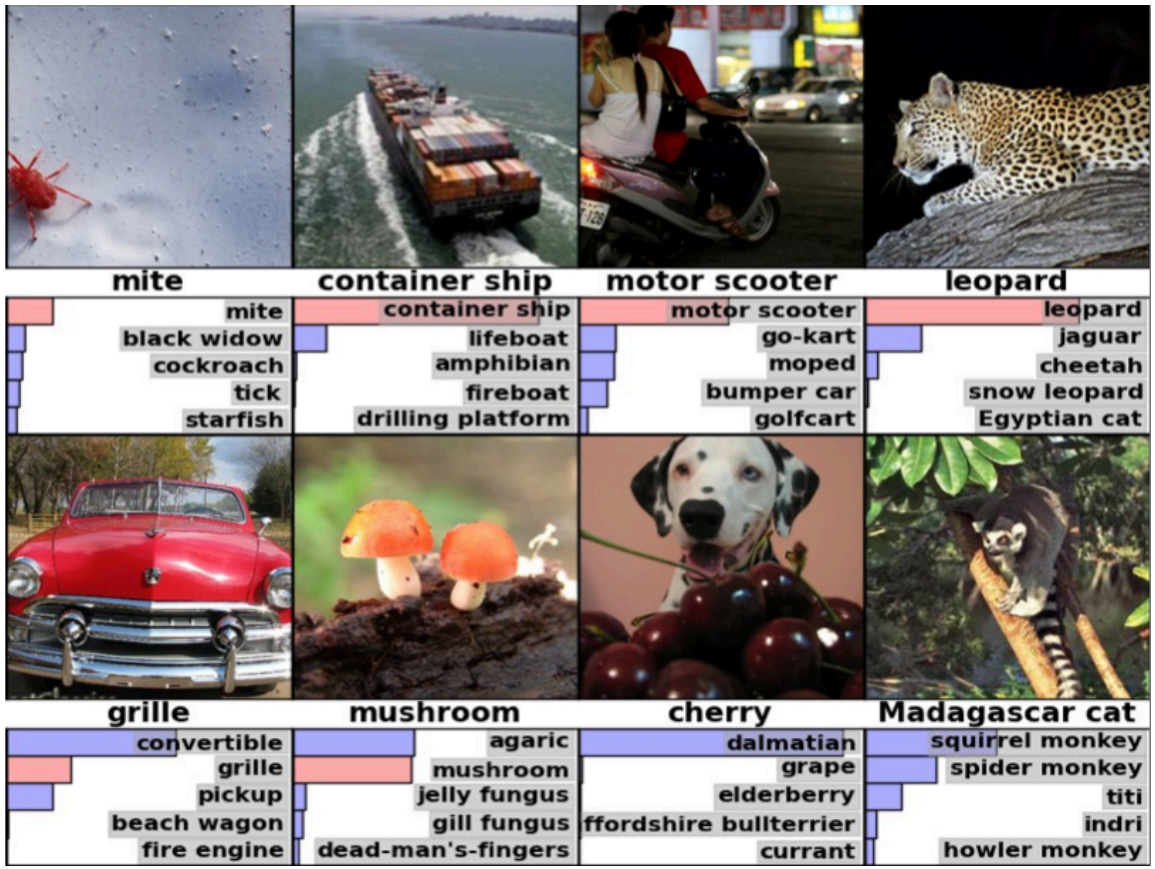
http://yann.lecun.com/exdb/lenet/

(LeNet 5, 1998)

The 82 errors made by LeNet5



Deep Learning, AD 2016



Humans:5.4%

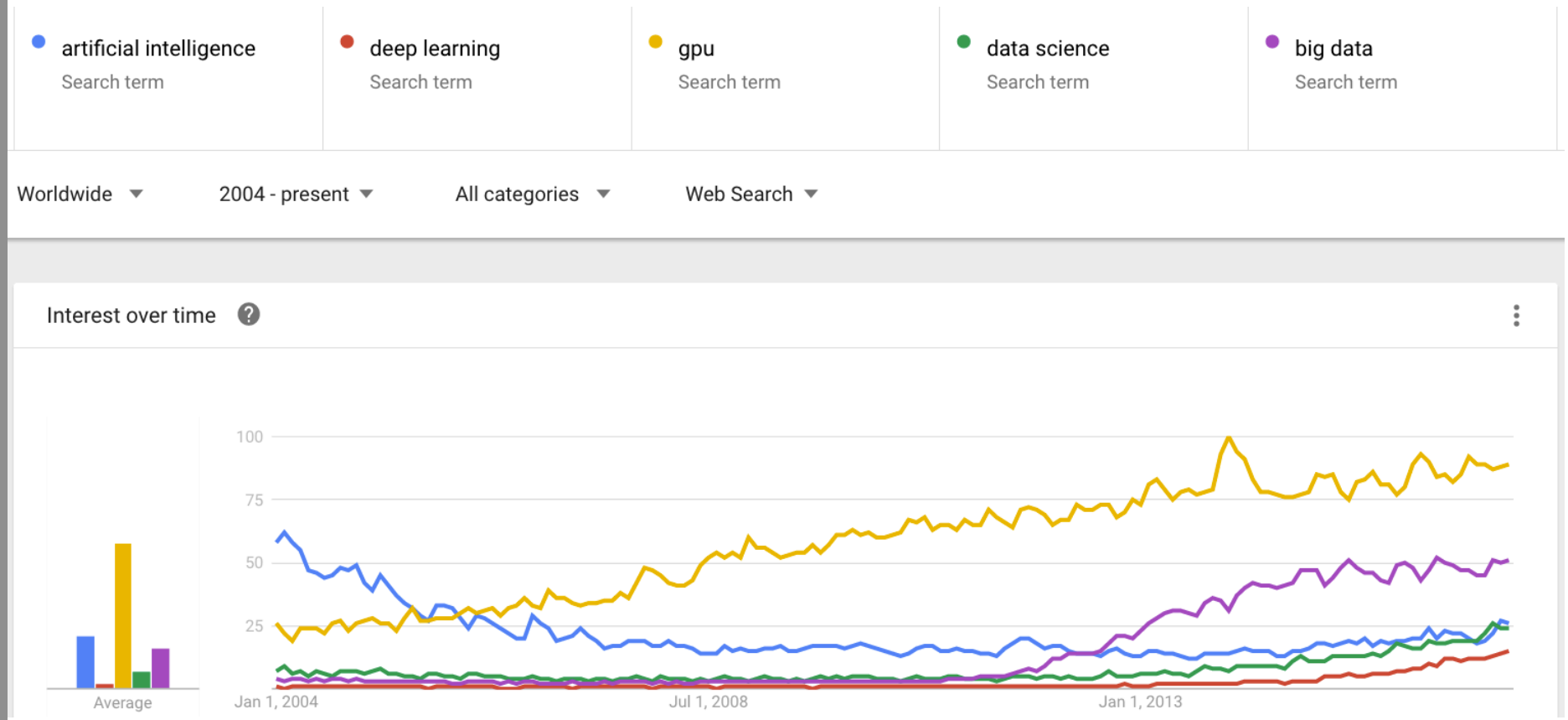
A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. *NIPS*13 [18%] (best shallow competitor: 36%)

K. He, X. Zhang, S. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, http://arxiv.org/abs/1502.01852, 2015. [4.5%]

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, http://arxiv.org/abs/1502.03167, 2015. [4.5%]

K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, Arxiv, 2015 [3.6%]

What happened in between?



deep learning = neural networks (+ big data + GPUs)

+ a few more recent tricks!

Neural network training: old & new tricks

Old: (80's)

- Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

- Dropout
- ReLUs
- Batch Normalization
- Residual Networks

Neural network training: old & new tricks

Old: (80's)

- Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

- Dropout
- ReLUs
- Batch Normalization
- Residual Networks

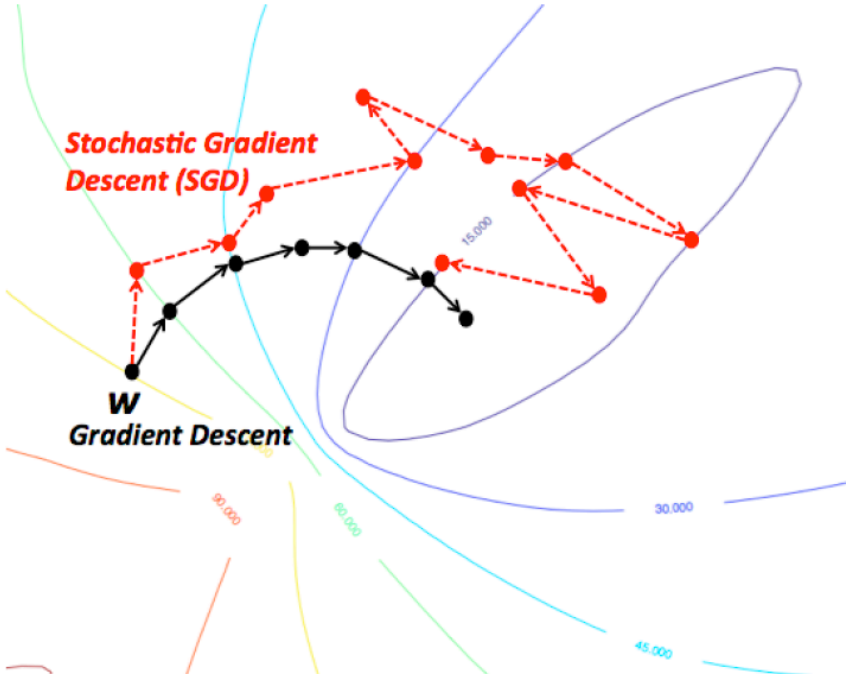
Neural network training: old & new tricks

Old: (80's)

Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

- Dropout
- ReLUs
- Batch Normalization
- Residual Networks



“Big data”: sometimes too big!

Challenges: fitting everything in memory

Keeping computational cost of training under control

“Large-Scale” Learning (checkout GI09: Applied Machine Learning)

Computer Vision Data: Big and Complicated

http://www.image-net.org/

IMAGENET

Overall

- Total number of non-empty synsets: 21841
- Total number of images: 14,197,122
- Number of images with bounding box annotations: 1,034,908

millions of images for academia

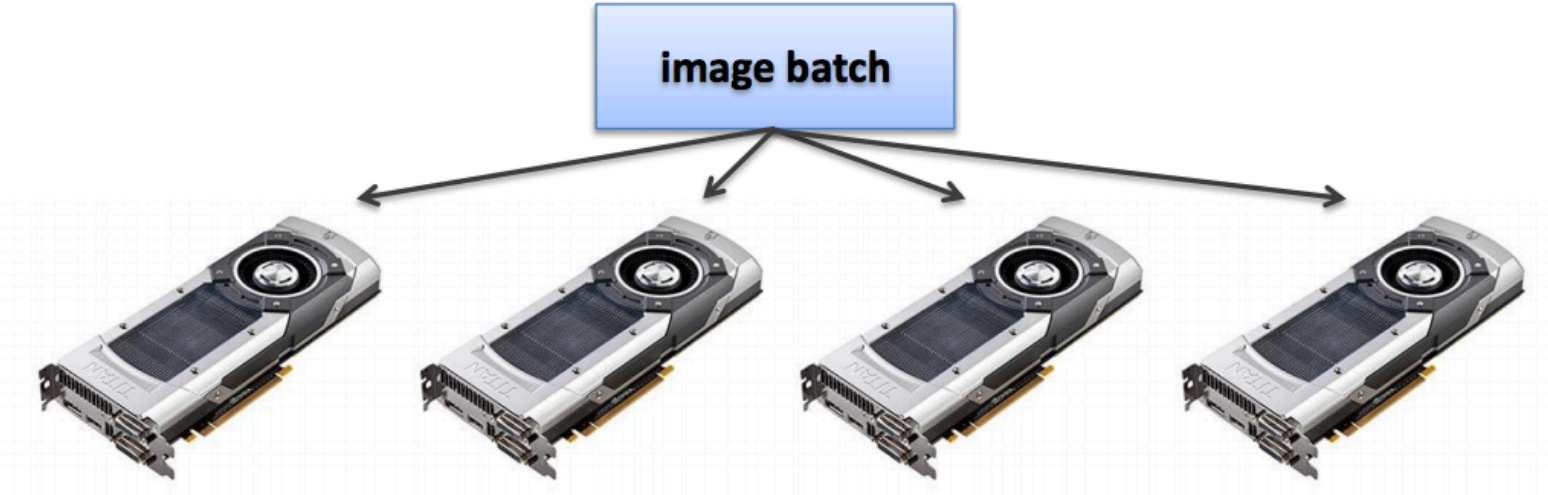
billions of images for industry

High level category	# synset (subcategories)	Avg # images per synset	Total # images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K
geological formation	151	838	127K
invertebrate	728	573	417K
mammal	1138	821	934K
musical instrument	157	891	140K
plant	1666	600	999K
reptile	268	707	190K
sport	166	1207	200K
structure	1239	763	946K
tool	316	551	174K
tree	993	568	564K
utensil	86	912	78K
vegetable	176	764	135K
vehicle	481	778	374K
person	2035	468	952K

Deng et. al, ImageNet: A Large-Scale Hierarchical Image Database, CVPR'09

How large is large?

- Heavily-modified Caffe C++ toolbox
- Multiple GPU support
  - 4 x NVIDIA Titan, off-the-shelf workstation
  - data parallelism for training and testing
  - ~3.75 times speed-up, 2-3 weeks for training



Simonyan & Zisserman, 2014

Training objective, multi-class case

Weeks 7-8:

One-hot label encoding: $\mathbf{y}^i = (0, 0, 1, 0)$

Likelihood of training sample: $(\mathbf{y}^i, \mathbf{x}^i)$

$$P(\mathbf{y}^i | \mathbf{x}^i; \mathbf{w}) = \prod_{c=1}^{C} (g_c(\mathbf{x}, \mathbf{W}))^{\mathbf{y}_c^i}$$

Optimization criterion:

$$L(\mathbf{W}) = -\sum_{i=1}^{N} \sum_{c=1}^{C} \mathbf{y}_c^i \log (g_c(\mathbf{x}, \mathbf{W}))$$

Parameter estimation: Gradient of L with respect to **W**

Training objective for classification

Likelihood of training sample's label: $P(\mathbf{y}^i|\mathbf{x}^i;\mathbf{w}) = \prod_{c=1}^{C} \left(g_c(\mathbf{x}^i, \mathbf{W})\right)^{\mathbf{y}_c^i}$

Cost function: $L(\mathbf{W}) = -\sum_{i=1}^{N}\sum_{c=1}^{C} \mathbf{y}_c^i \log\left(g_c(\mathbf{x}^i, \mathbf{W})\right)$

Normalize: $L'(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}\left[-\sum_{c=1}^{C} \mathbf{y}_c^i \log\left(g_c(\mathbf{x}^i, \mathbf{W})\right)\right]$

$$= \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) \qquad \hat{\mathbf{y}}^i = \begin{bmatrix} g_1(\mathbf{x}, \mathbf{W}) \\ \vdots \\ g_C(\mathbf{x}, \mathbf{W}) \end{bmatrix}$$

Add regularization: $L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_{l} \lambda_l \sum_{k,m} (\mathbf{W}_{k,m}^l)^2$

for all layers (l), and all input (k) –output (m) connection weights

Training objective for classification

$$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_l \lambda_l \sum_{k,m} (\mathbf{W}^l_{k,m})^2$$

Gradient descent: $\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$

(l,k,m) element of gradient vector:

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Back-prop for i-th example

If N=10^6 , we will need to run back-prop 10^6 times to update **W** once!

Stochastic Gradient Descent (SGD)

Gradient:

Batch: [1..N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Noisy ('Stochastic') Gradient:

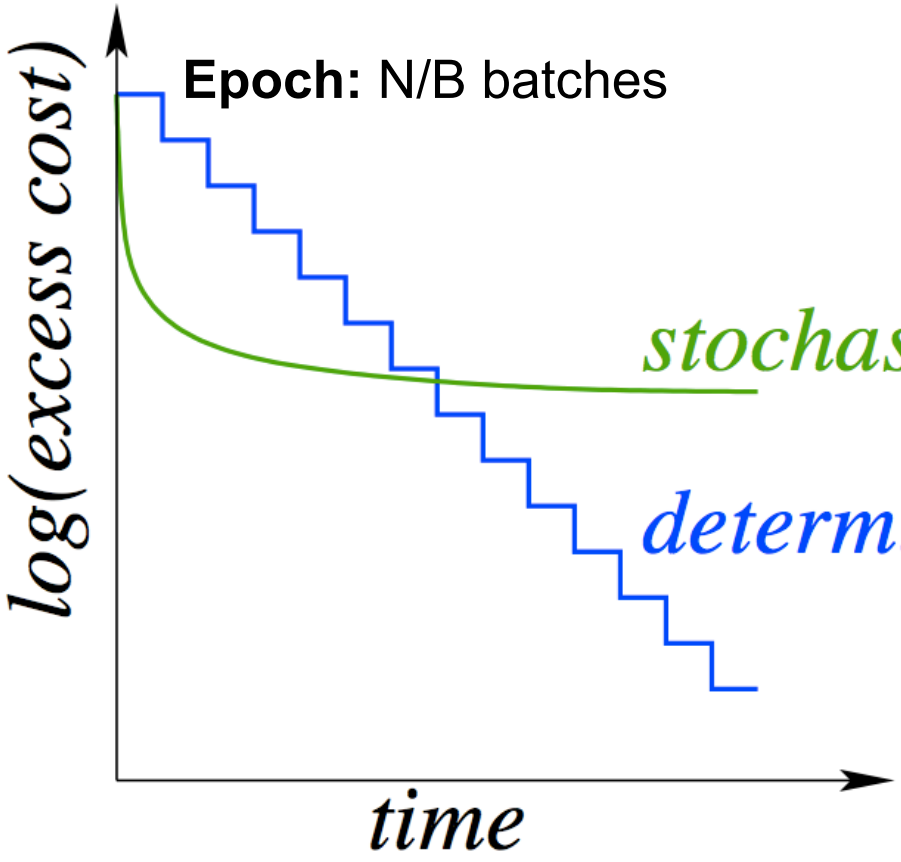
Minibatch: B elements

b(1), b(2),…, b(B): sampled from [1,N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} \simeq \frac{1}{B}\sum_{i=1}^{B}\frac{\partial l(\mathbf{y}^{b(i)}, \hat{\mathbf{y}}^{b(i)})}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Epoch: N samples, N/B batches

Is Stochastic Gradient Descent faster?

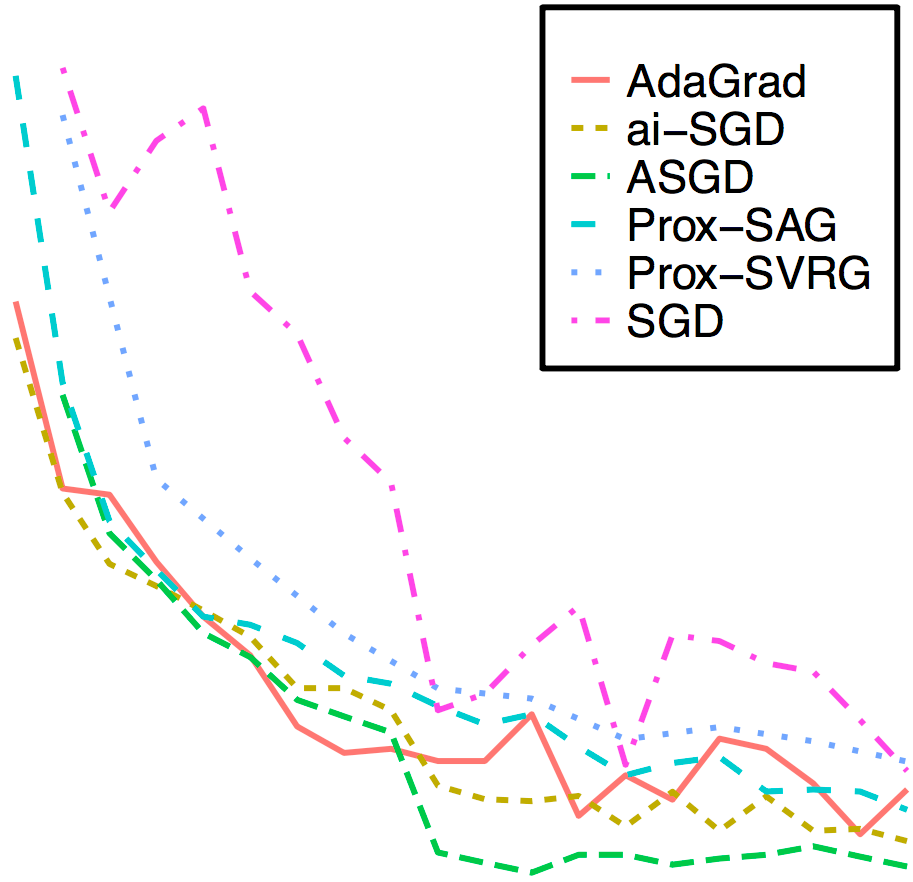


In one epoch SGD performs N/B more updates than GD!

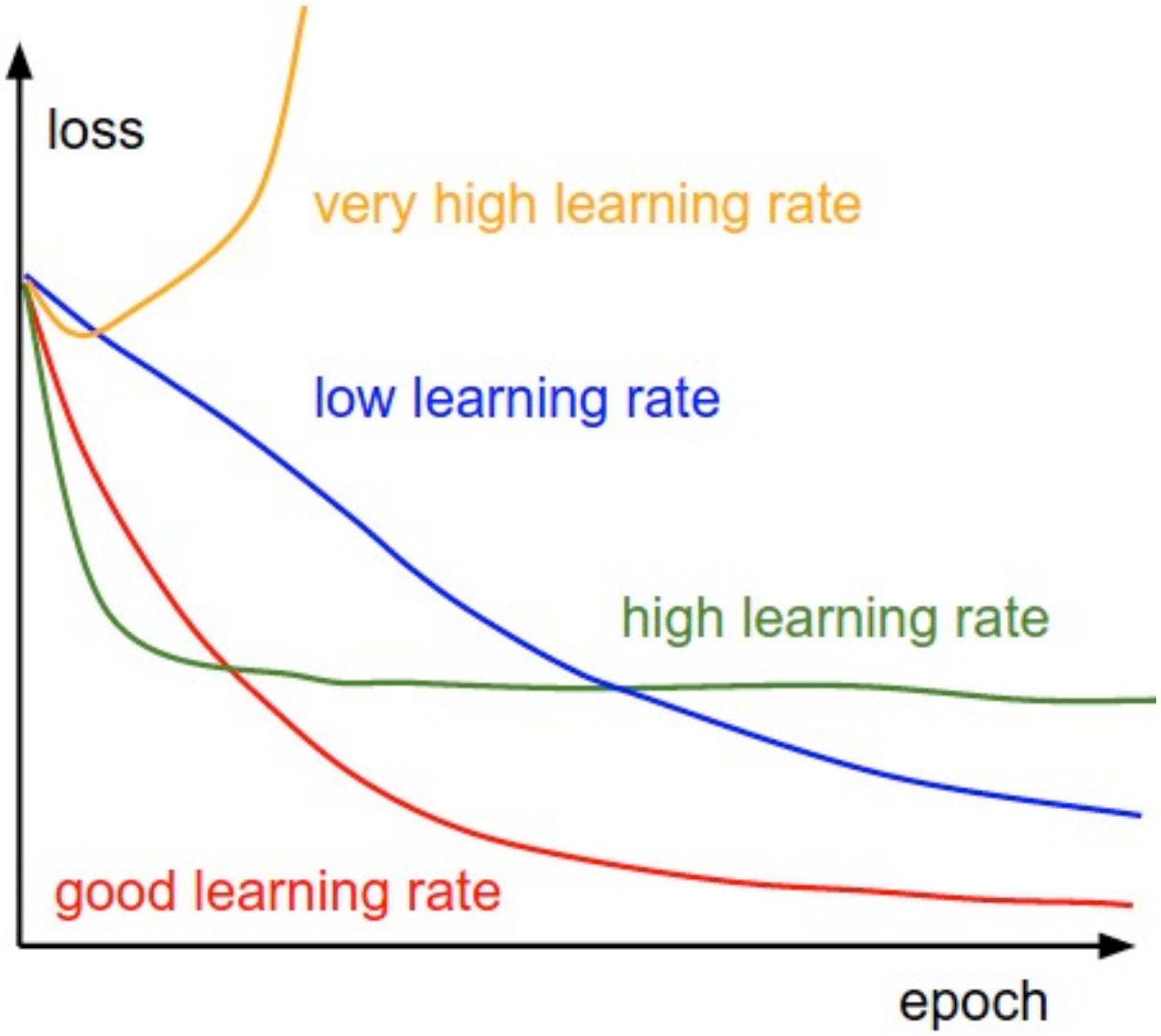
N: 10^6,B = 10: 100000 x more steps

Current research:
best of both worlds?
2nd order (Newton-Raphson-like) methods?

“Right” SGD: active research topic

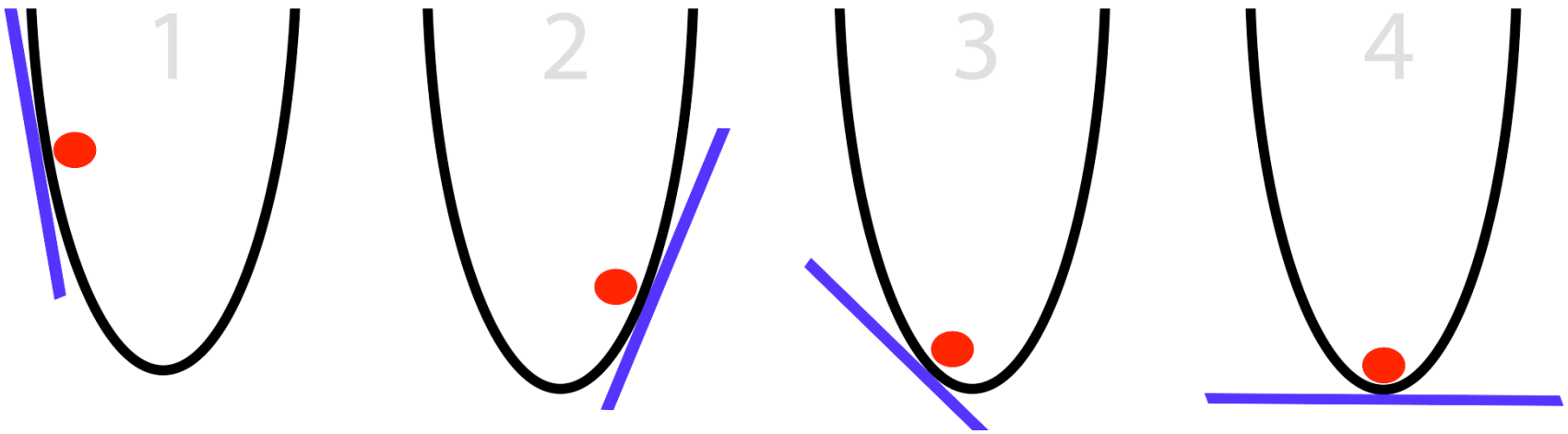


Learning rate

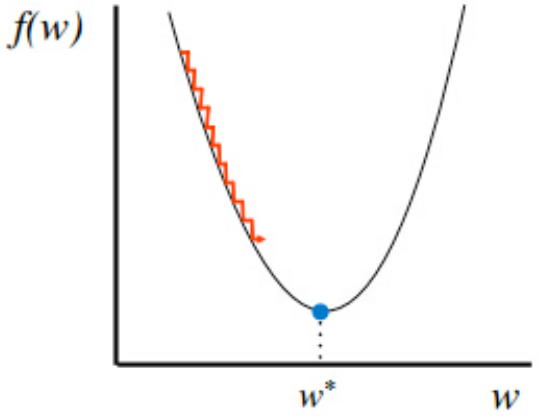


$$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

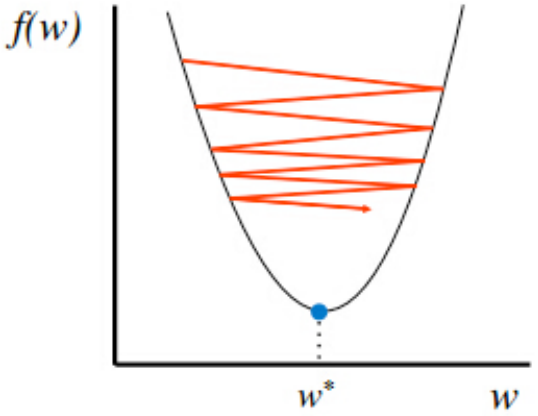
Gradient Descent



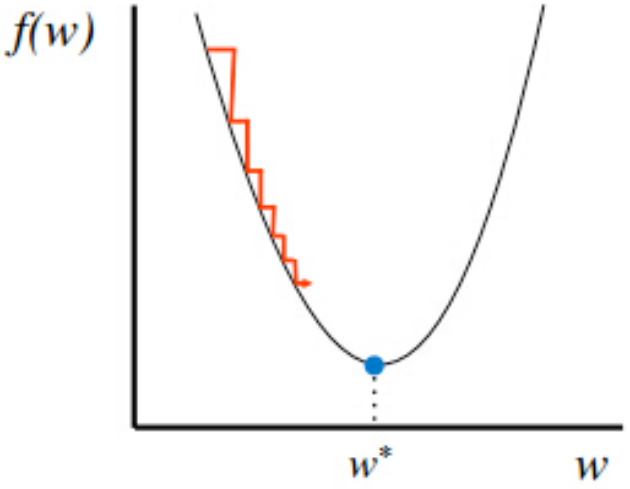
(S)GD with adaptable stepsize



Too small: converge very slowly



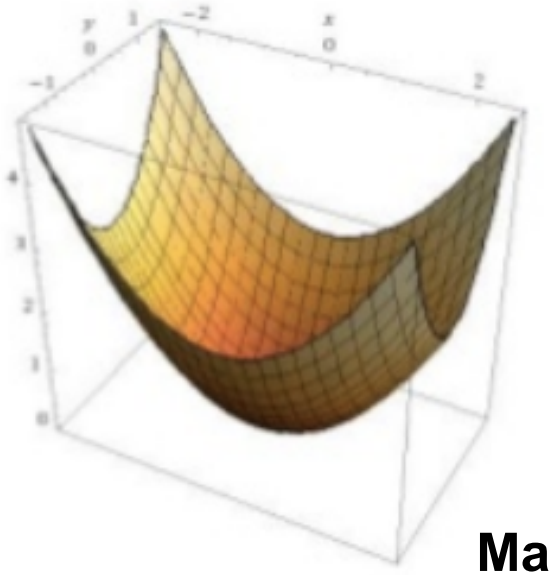
Too big: overshoot and even diverge

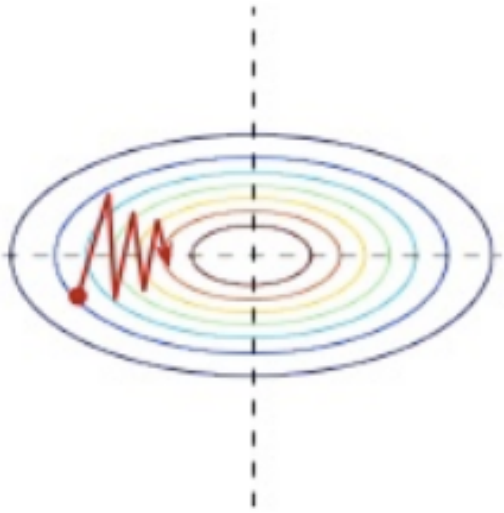


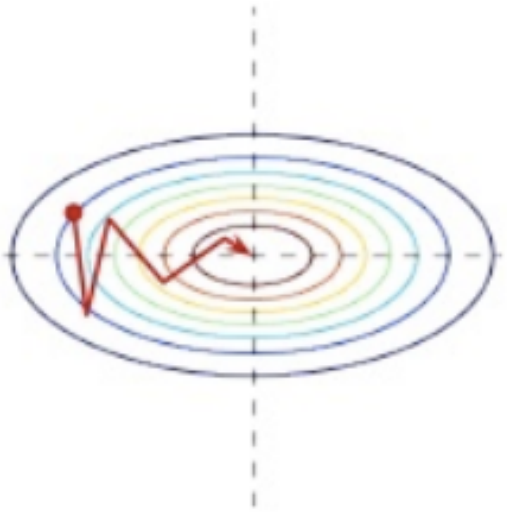
Reduce size over time

e.g. $\epsilon_t = \frac{c}{t}$

(S)GD with momentum







Main idea: retain long-term trend of updates, drop osci

(S)GD $$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \nabla_{\mathbf{W}} L(\mathbf{W})$$

(S)GD + momentum

$$\mathbf{V}_{t+1} = \mu \mathbf{V}_t + (1-\mu)\nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \mathbf{V}_{t+1}$$

Neural network training: old & new tricks

Old: (80's)

- Stochastic Gradient Descent, Momentum, "weight decay"

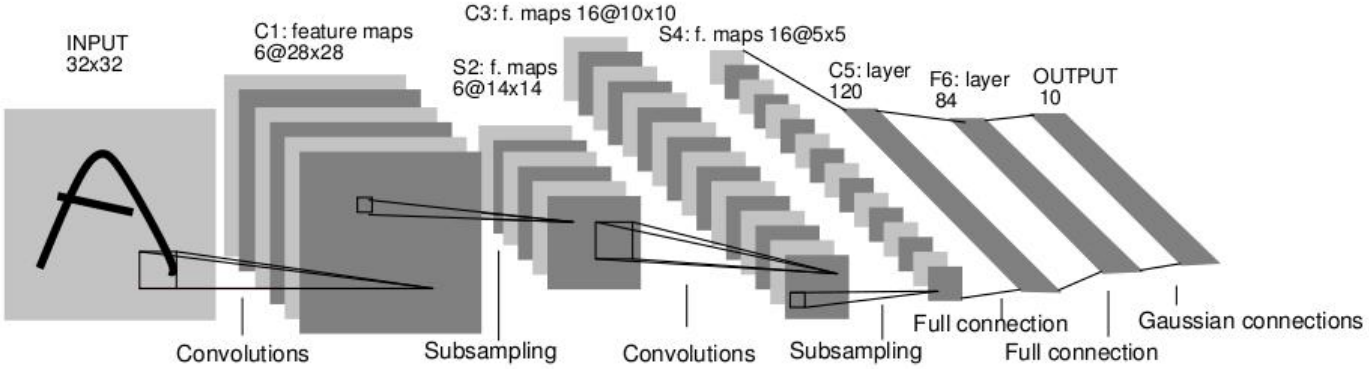
New: (last 5-6 years)

- Dropout
- ReLUs
- Batch Normalization
- Residual Networks

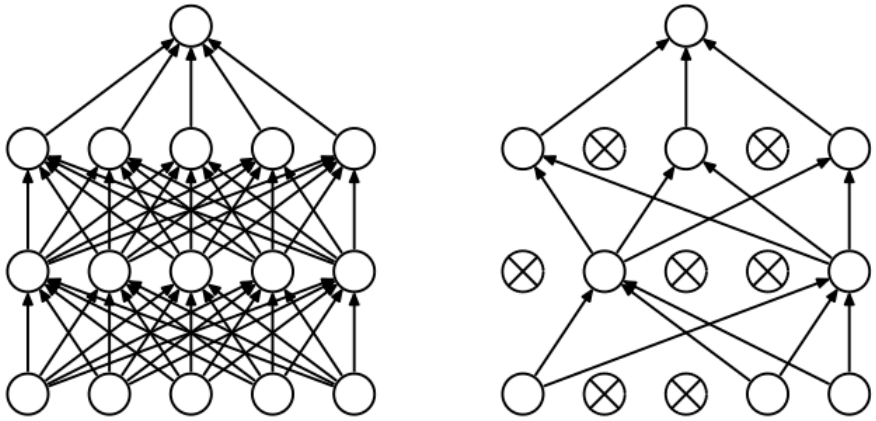
Regularization in Deep Learning

Weight Decay: just before

Convolutional Networks: last week



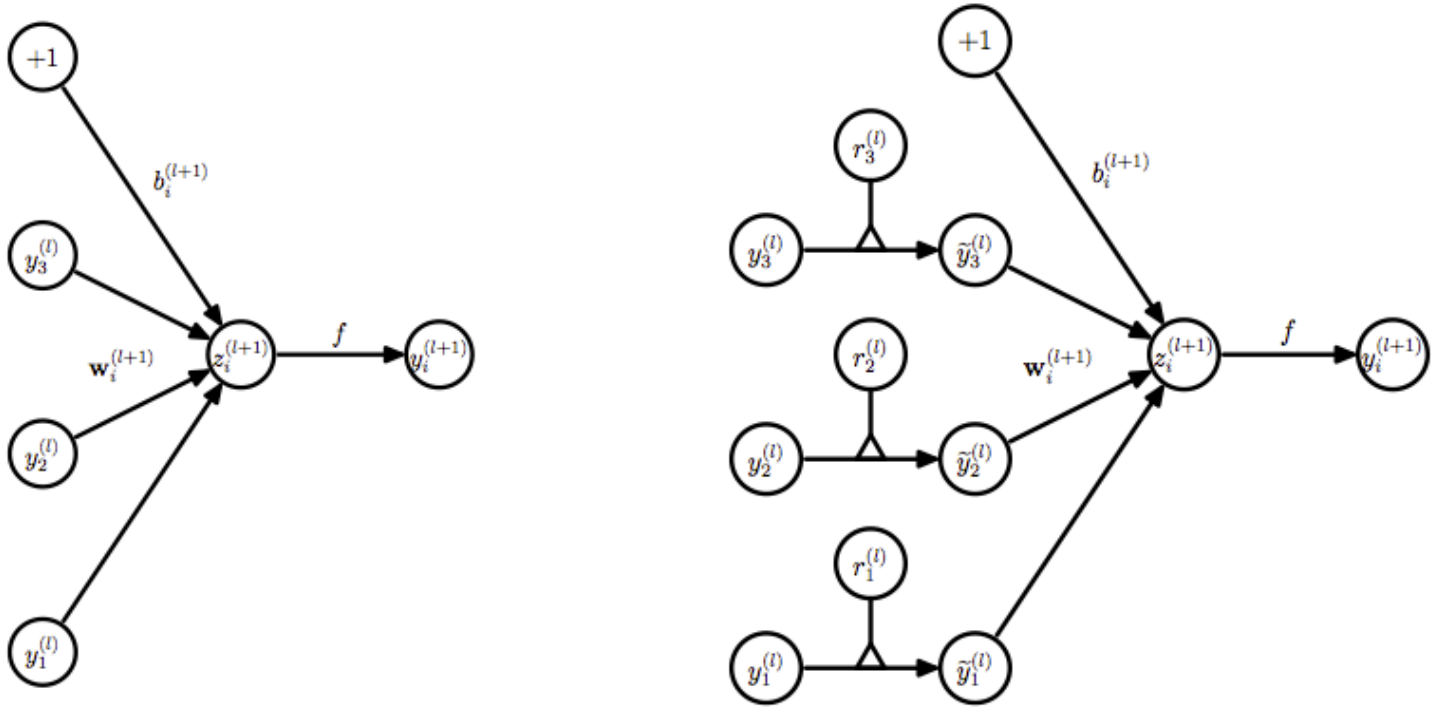
Dropout: now



(a) Standard Neural Net

(b) After applying dropout.

Dropout



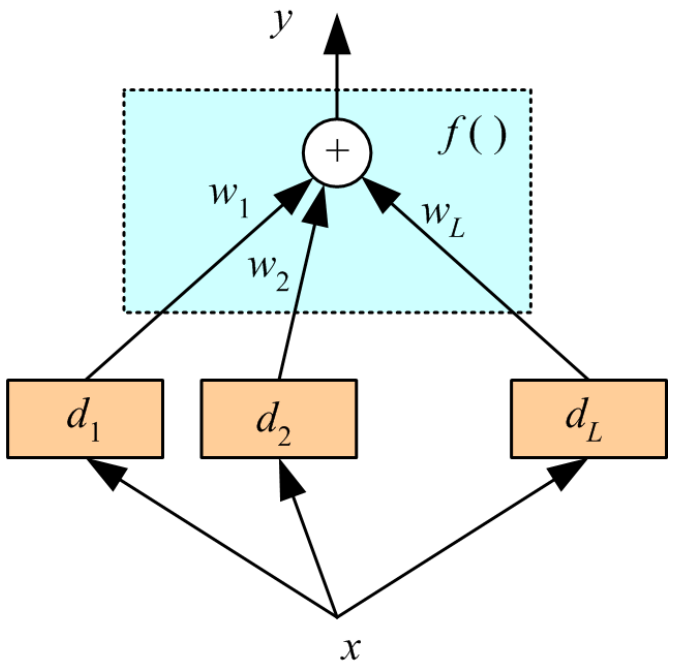
(a) Standard network

(b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

Voting Methods

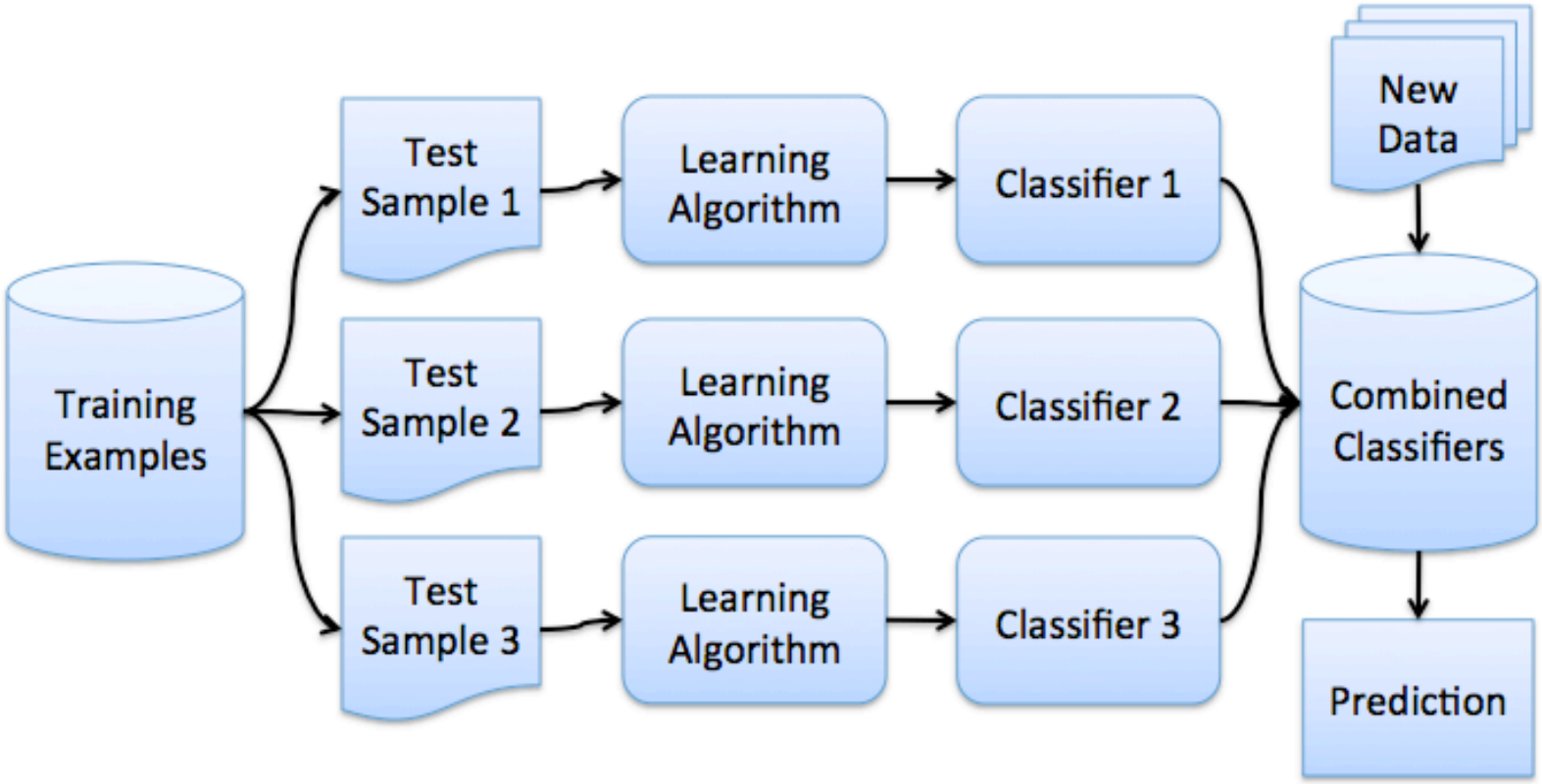
- Give up idea of building `the' classifier
- Generate a group of base-learners which has higher accuracy when combined
- Main tasks
  - Generating the learners
  - Combining them



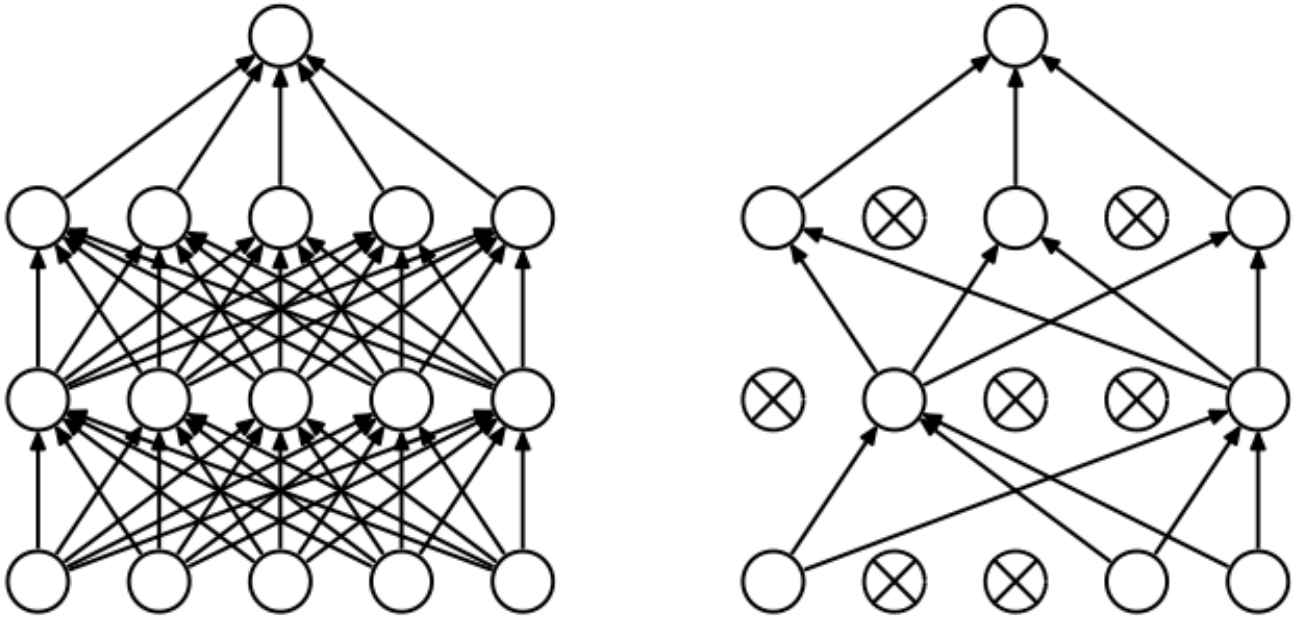
Why should this work? Week 5 Lecture

- Committee of M predictors for target output $$y_{COM}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x})$$
- Output: true value + error $$y(\mathbf{x}) = h(\mathbf{x}) + \epsilon(\mathbf{x})$$
- Expected sum of squares error for m-th expert:
$$\mathbb{E}_{\mathbf{x}}[(y_m(\mathbf{x}) - h(\mathbf{x}))^2] = \mathbb{E}_{\mathbf{x}}[e_m(\mathbf{x})^2]$$
- Average error of individual members:
$$\mathbb{E}_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$
- Average error of committee:
$$\mathbb{E}_{COM} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x})\right\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$
- **If** committee members have uncorrelated errors: $$\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\epsilon_j(\mathbf{x})\right] = 0$$
$$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$$

Bootstrapped AGGregatING (BAGGING)



Dropout



(a) Standard Neural Net

(b) After applying dropout.

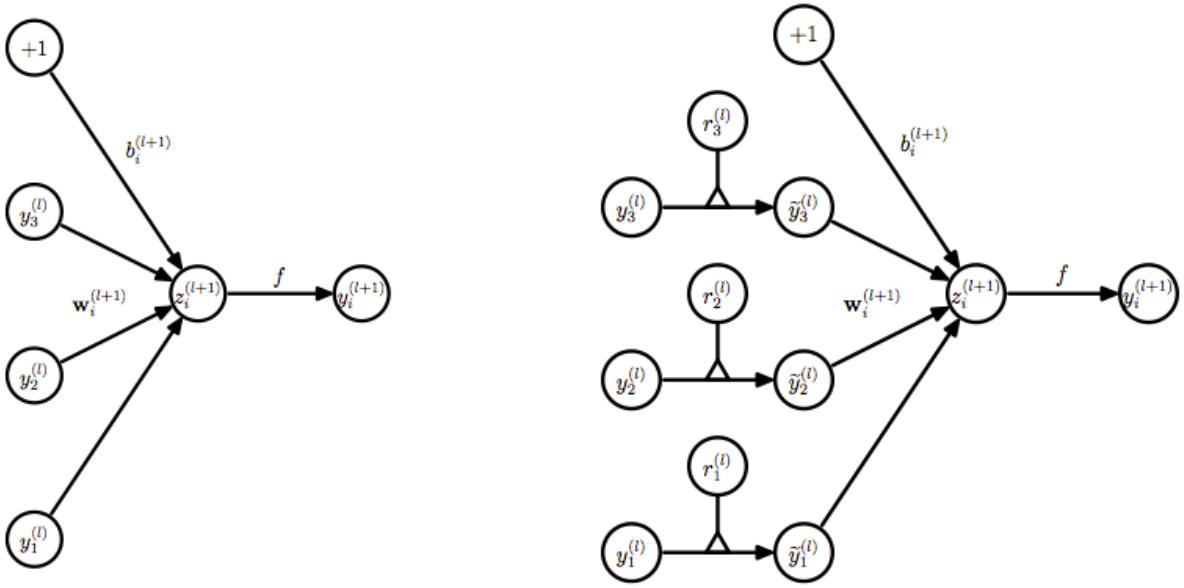
Each sample is processed by a ‘decimated’ neural net

Decimated nets: distinct classifiers

But: they should all do the same job

Improving neural networks by preventing co-adaptation of feature detectors
GE Hinton, N Srivastava, A Krizhevsky, I Sutskever, RR Salakhutdinov, arXiv, 2012, JMLR 2014
http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Dropout block



(a) Standard network

(b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

‘Feature noising’

Test time: Deterministic approximation

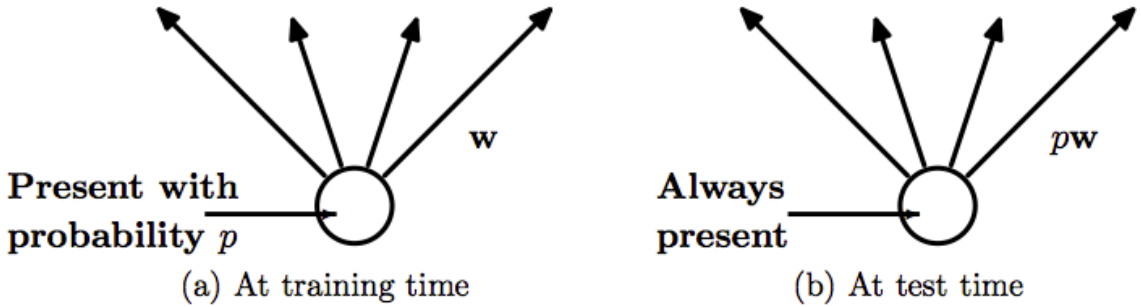
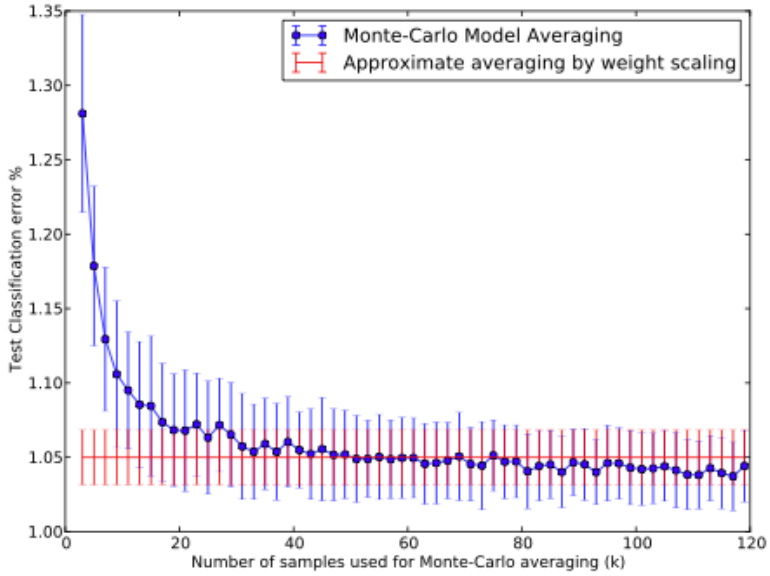


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

At test time, the weights are scaled as W(l) = pW(l) as shown in Figure 2. The resulting neural network is used without dropout.

An expensive but more correct way of averaging the models is to sample k neural nets using dropout for each test case and average their predictions. As k → ∞, this Monte-Carlo model average gets close to the true model average.

By computing the error for different values of k we can see how quickly the error rate of the finite-sample average approaches the error rate of the approximate model average.



Dropout performance

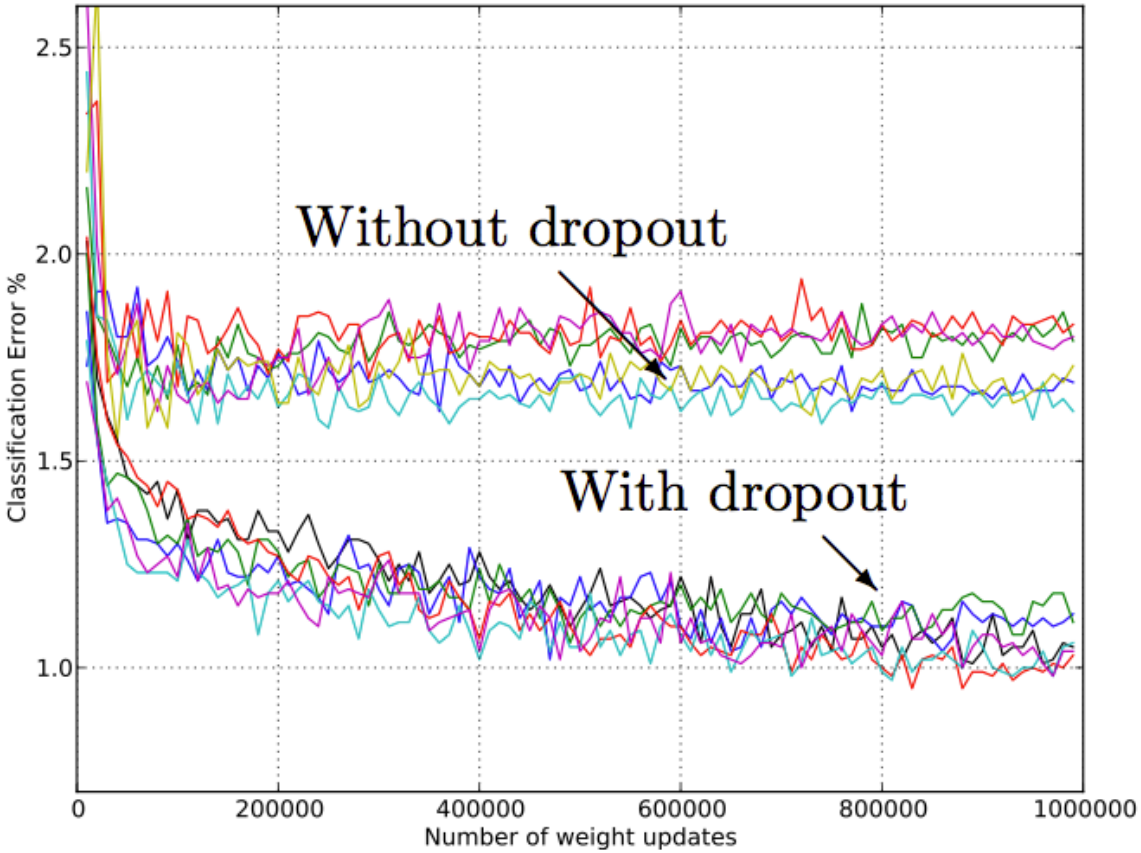


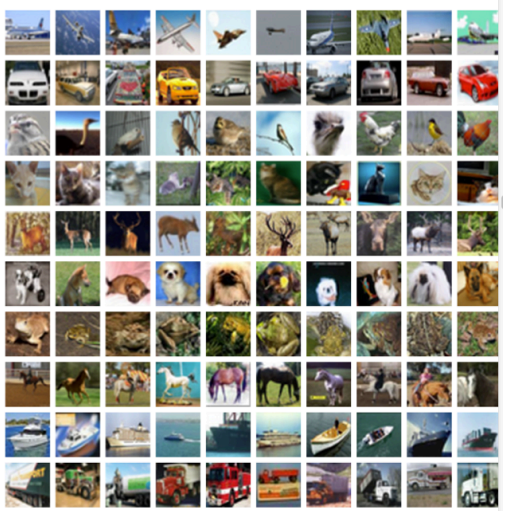
Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Dropout performance

(a) Street View House Numbers (SVHN)

Method	Error %
Binary Features (WDCH) (Netzer et al., 2011)	36.7
HOG (Netzer et al., 2011)	15.0
Stacked Sparse Autoencoders (Netzer et al., 2011)	10.3
KMeans (Netzer et al., 2011)	9.4
Multi-stage Conv Net with average pooling (Sermanet et al., 2012)	9.06
Multi-stage Conv Net + L2 pooling (Sermanet et al., 2012)	5.36
Multi-stage Conv Net + L4 pooling + padding (Sermanet et al., 2012)	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	2.80
Conv Net + max pooling + dropout in all layers	2.55
Conv Net + maxout (Goodfellow et al., 2013)	**2.47**
Human Performance	2.0

Table 3: Results on the Street View House Numbers data set.



(b) CIFAR-10

Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Table 4: Error rates on CIFAR-10 and CIFAR-100.

Dropout performance

Figure 6: Some ImageNet test cases with the 4 most probable labels as predicted by our model. The length of the horizontal bars is proportional to the probability assigned to the labels by the model. Pink indicates ground truth.

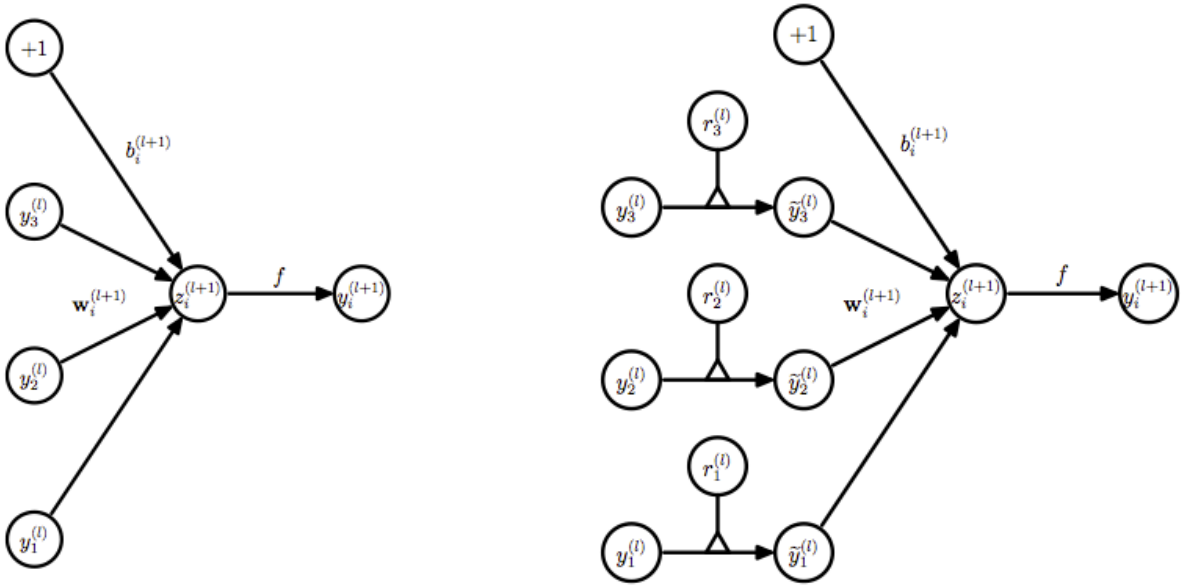
Model	**Top-1**	**Top-5**
Sparse Coding (Lin et al., 2010)	47.1	28.2
SIFT + Fisher Vectors (Sanchez and Perronnin, 2011)	45.7	25.7
Conv Net + dropout (Krizhevsky et al., 2012)	37.5	17.0

Table 5: Results on the ILSVRC-2010 test set.

Model	**Top-1 (val)**	**Top-5 (val)**	**Top-5 (test)**
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

Table 6: Results on the ILSVRC-2012 validation/test set.

Dropout block



(a) Standard network

(b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

‘Feature noising’

Neural network training: old & new tricks

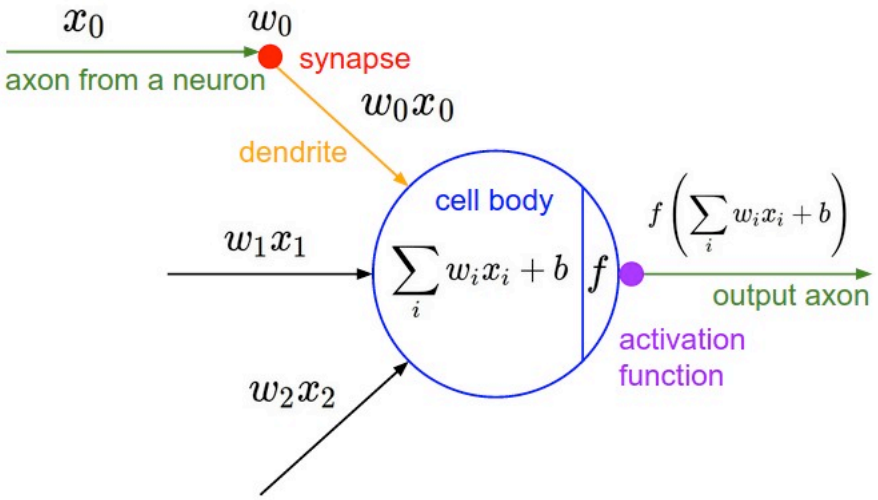
Old: (80's)

- Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

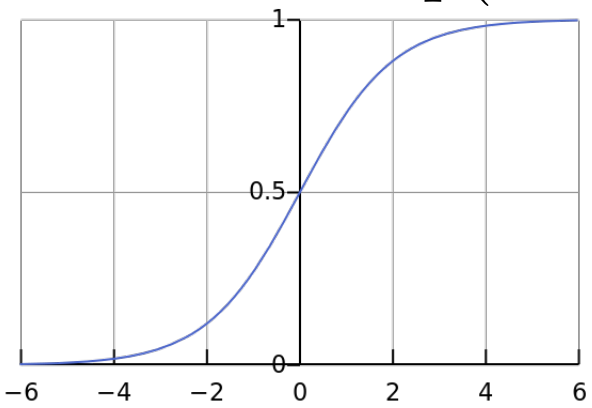
- Dropout
- ReLUs
- Batch Normalization
- Residual Networks

‘Neuron’: cascade of linear and nonlinear function



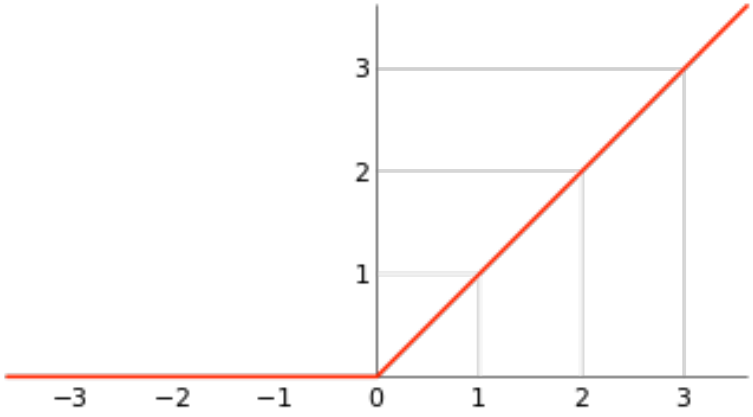
Sigmoidal (“logistic”)

$$g(a) = \frac{1}{1 + \exp(-a)}$$



Rectified Linear Unit (RELU)

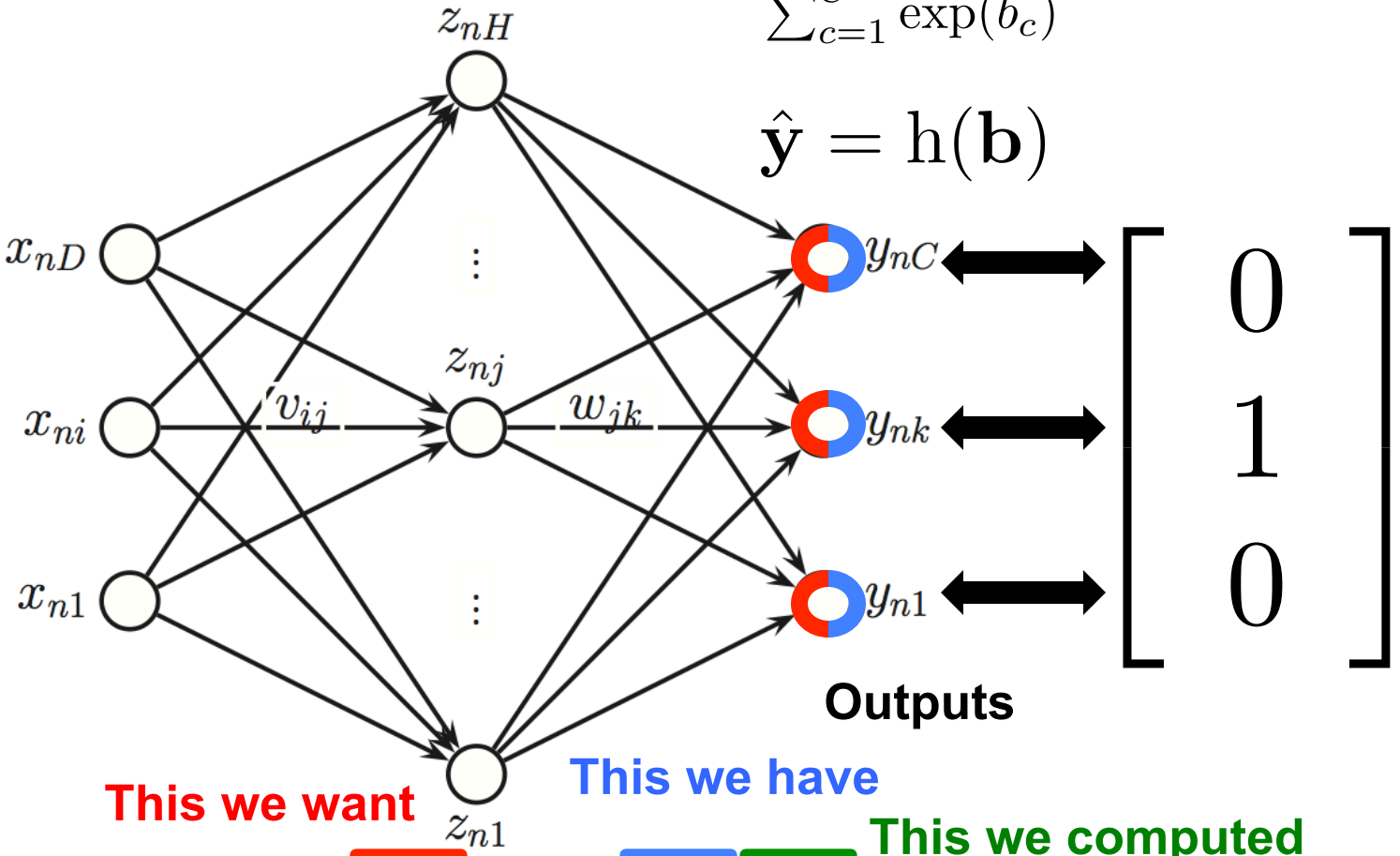
$$g(a) = \max(0, a)$$



A neural network in backward mode:

$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{c=1}^{C} \exp(b_c)}$$

$$\hat{\mathbf{y}} = \mathrm{h}(\mathbf{b})$$



This we want

$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

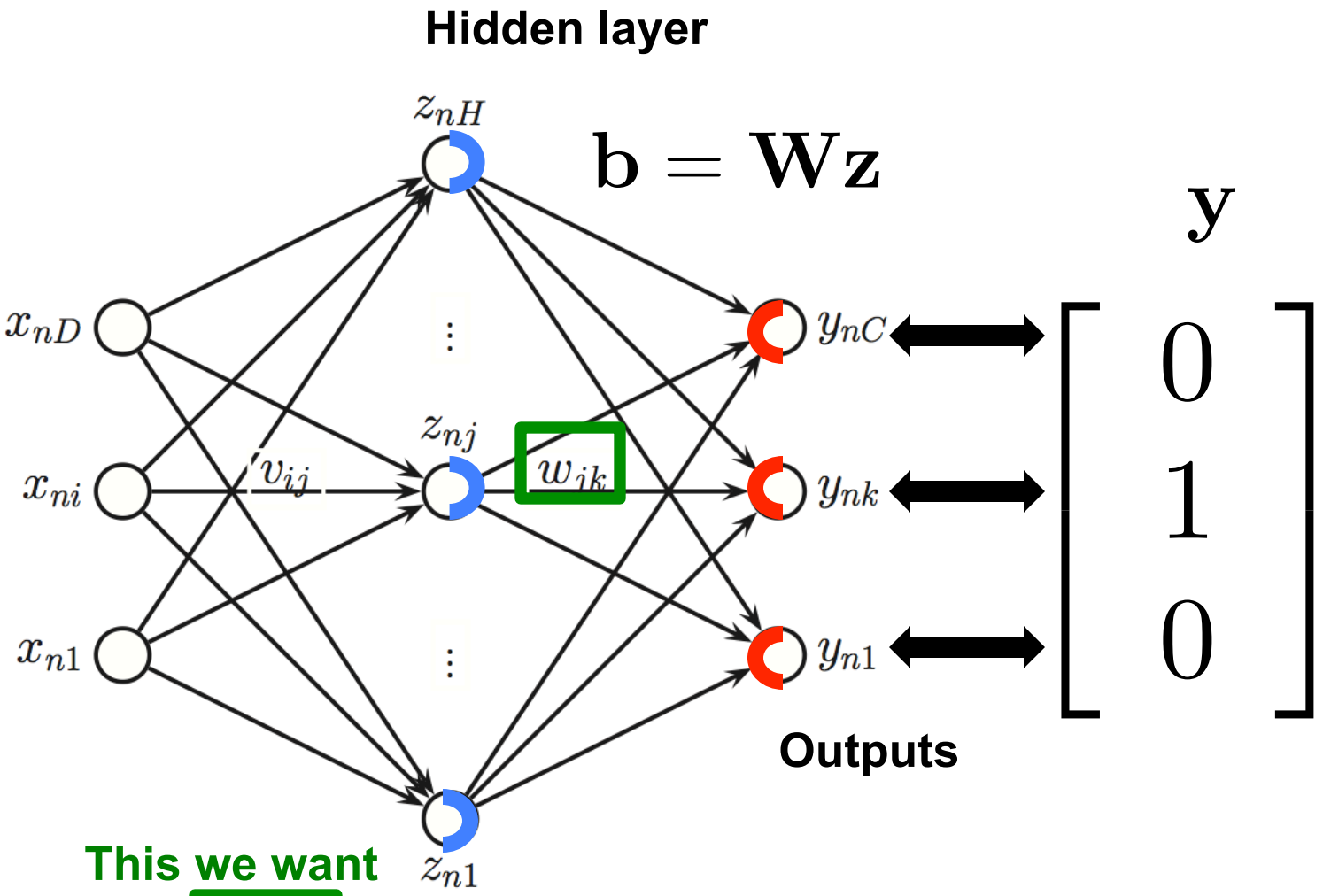
This we have

$$\frac{\partial L}{\partial b_k} = \sum_c \frac{\partial L}{\partial \hat{y}_c} \frac{\partial \hat{y}_c}{\partial b_k}$$

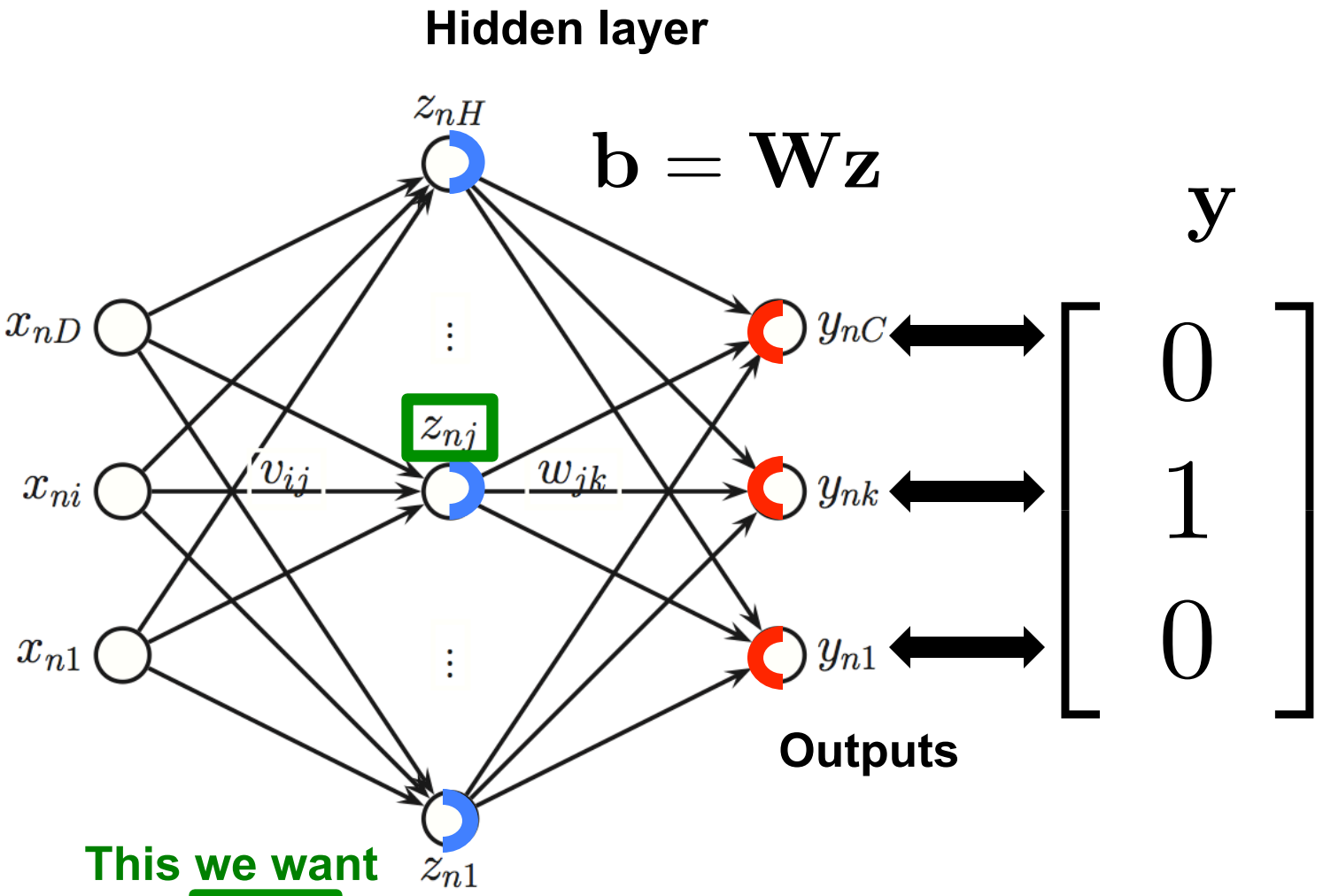
This we computed

$$= \hat{y}_k - y_k$$

A neural network in backward mode:

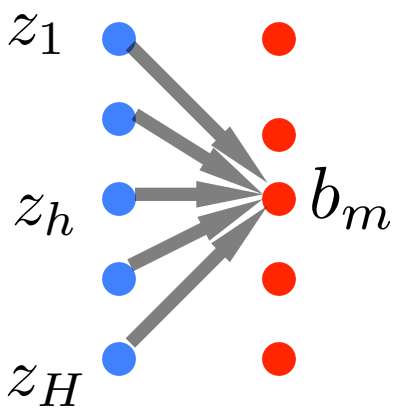


A neural network in backward mode:



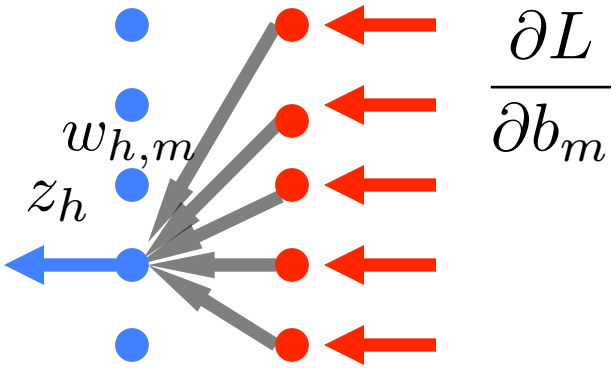
Linear layer in forward mode: all for one

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



Linear layer in backward mode: one from all

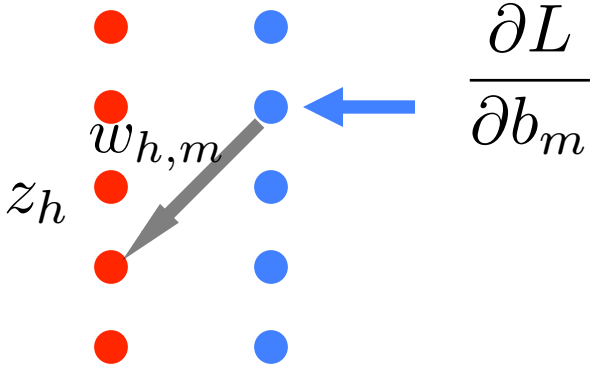
$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



$$\frac{\partial L}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} w_{h,c}$$

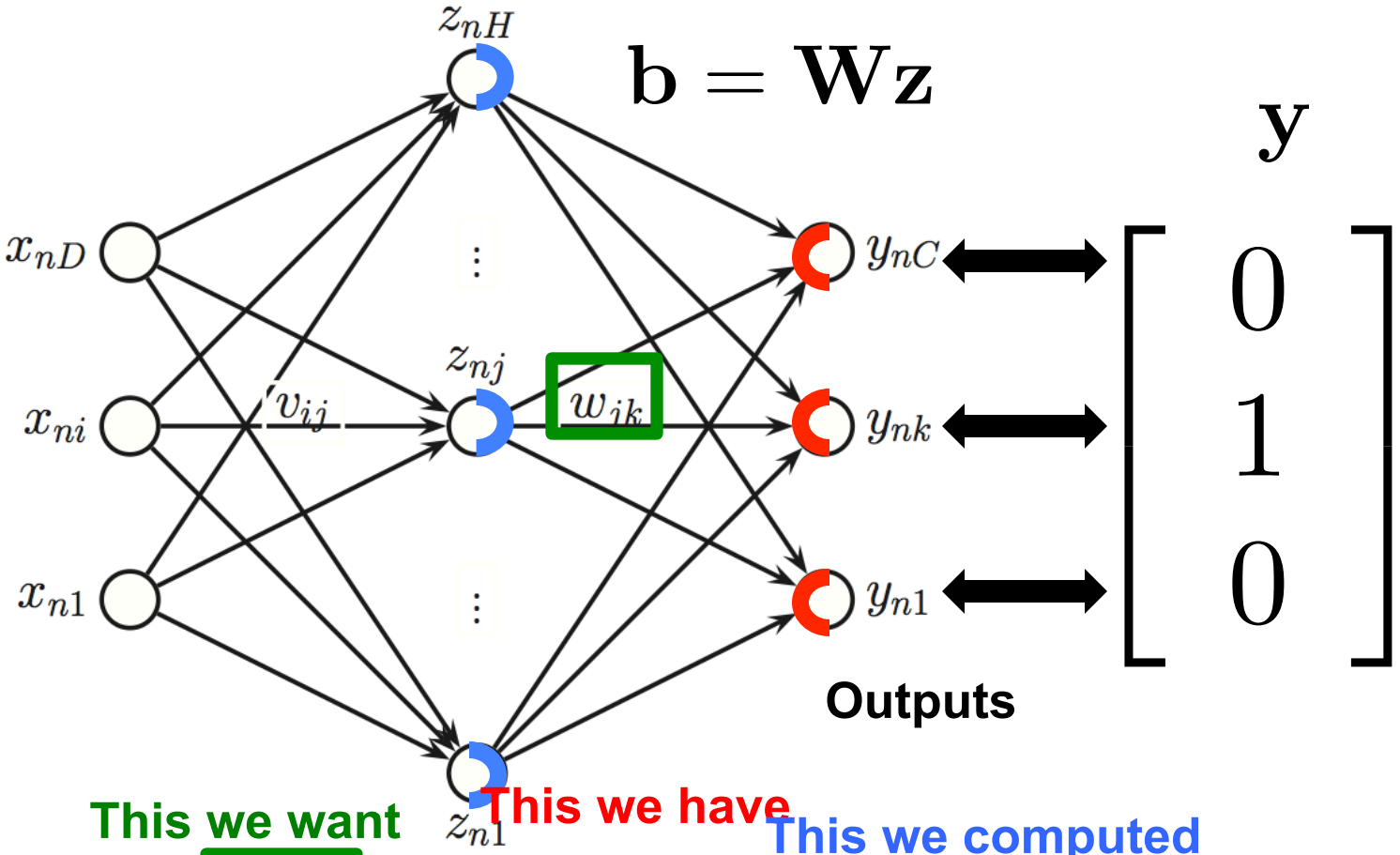
Linear layer parameters in backward: 1-to-1

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



$$\frac{\partial L}{\partial w_{h,m}} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial w_{h,m}} = \frac{\partial L}{\partial b_m} z_h$$

A neural network in backward mode:



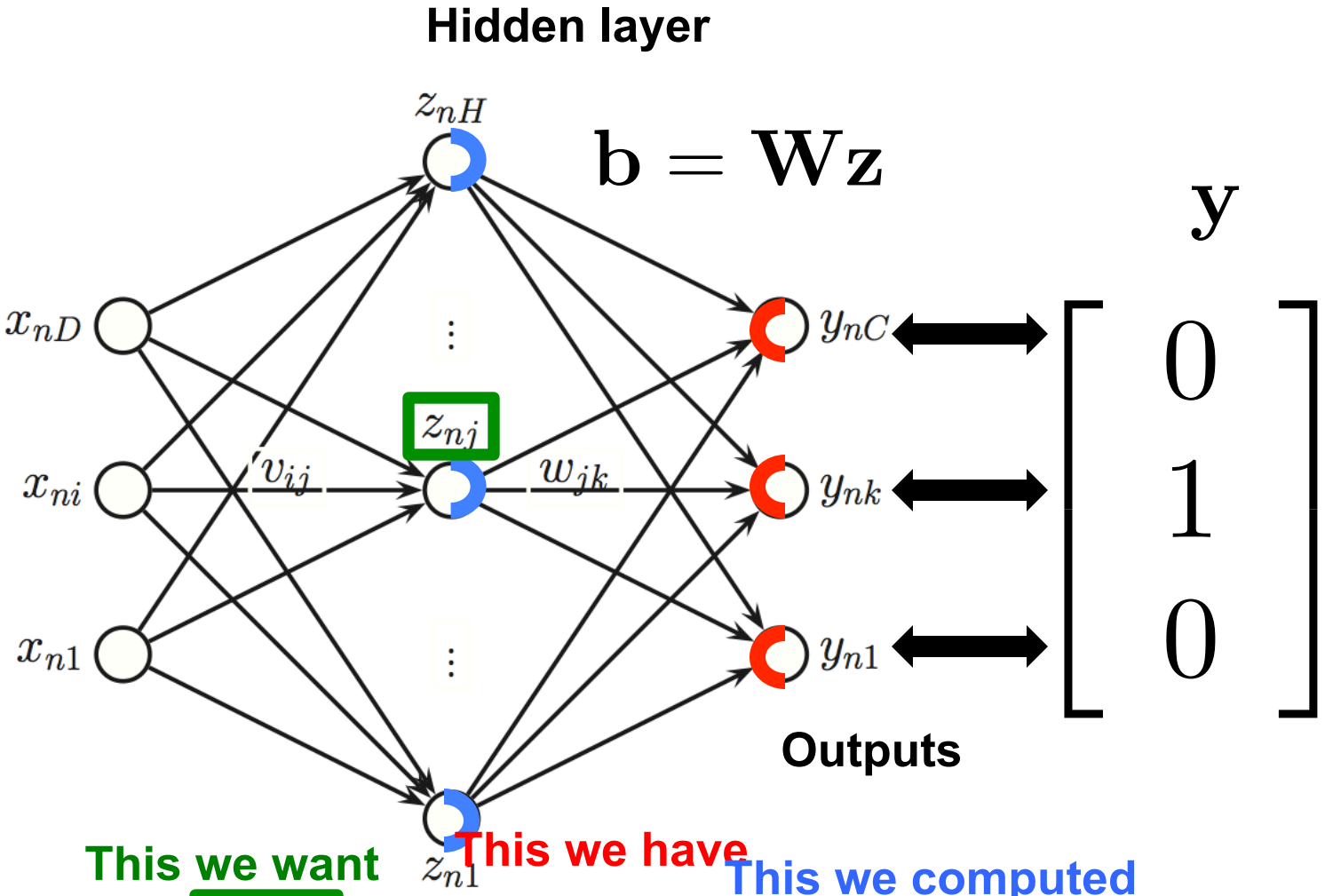
This we want

This we have

This we computed

$$\frac{\partial l}{\partial w_{jk}} = \sum_m \frac{\partial l}{\partial b_m} \frac{\partial b_m}{\partial w_{jk}} = \frac{\partial l}{\partial b_m} z_j$$

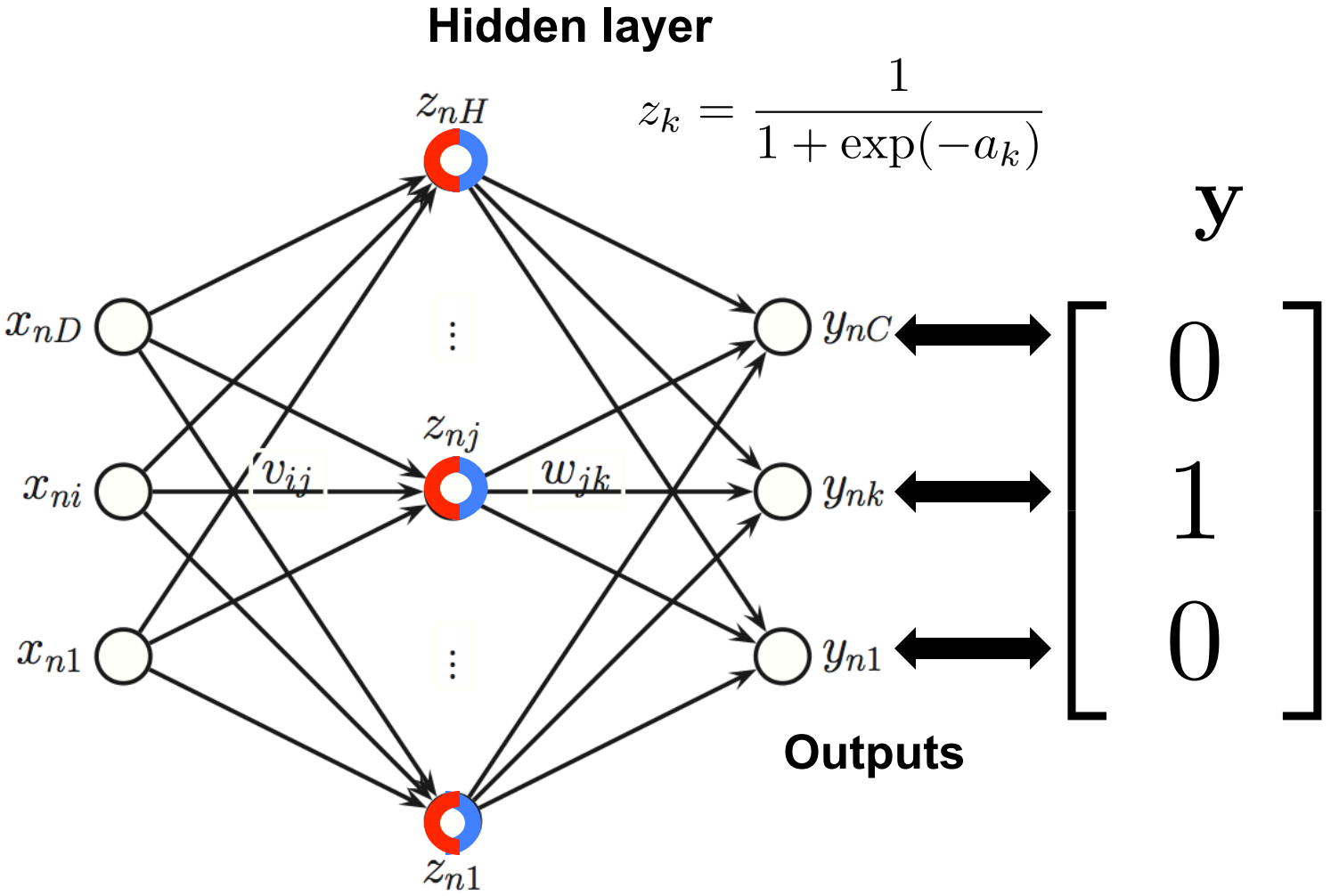
A neural network in backward mode: ⏪



This we want | This we have | This we computed

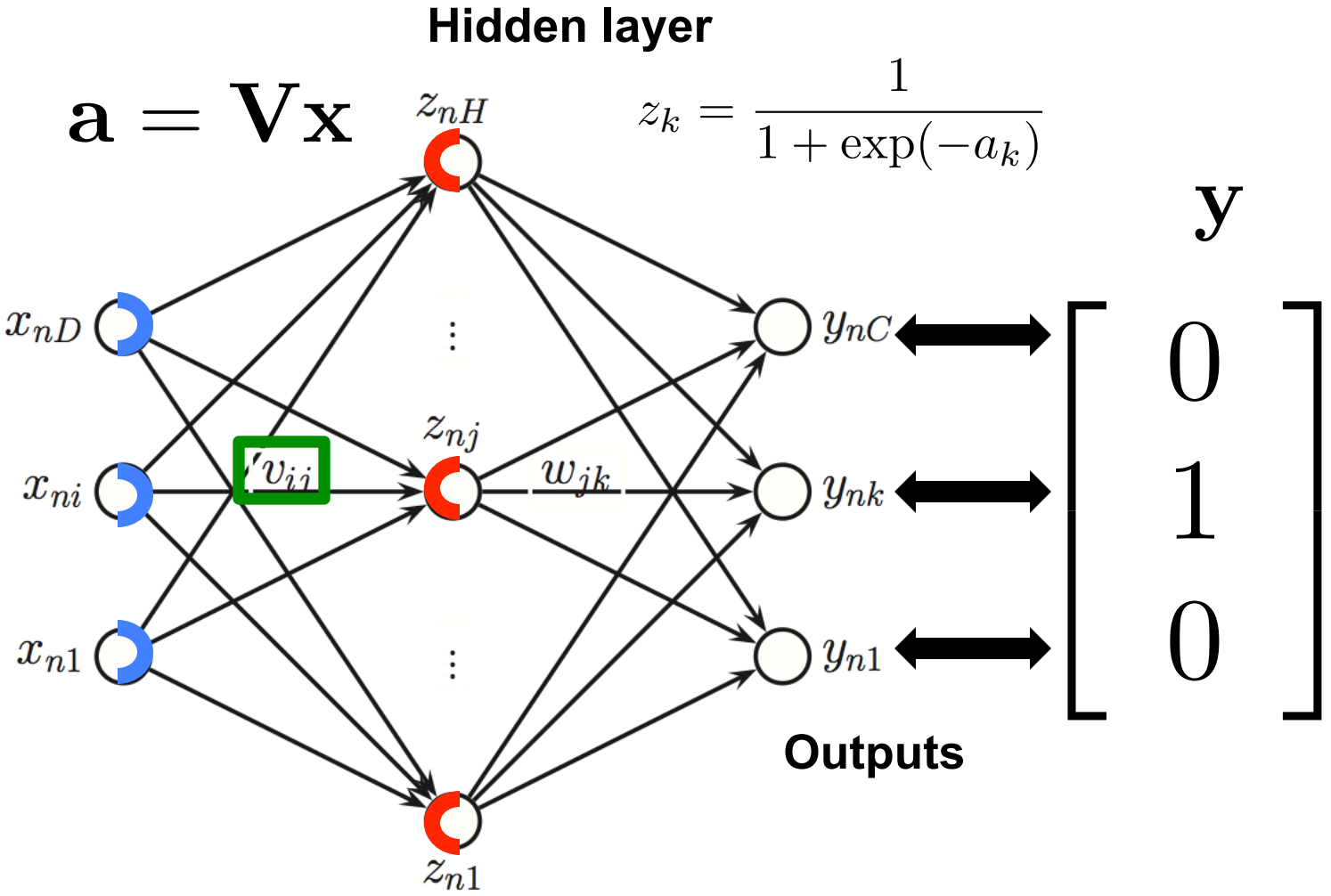
$$\frac{\partial l}{\partial z_j} = \sum_m \frac{\partial l}{\partial b_m} \frac{\partial b_m}{\partial z_j} = \sum_m \frac{\partial l}{\partial b_m} w_{j,m}$$

A neural network in backward mode:



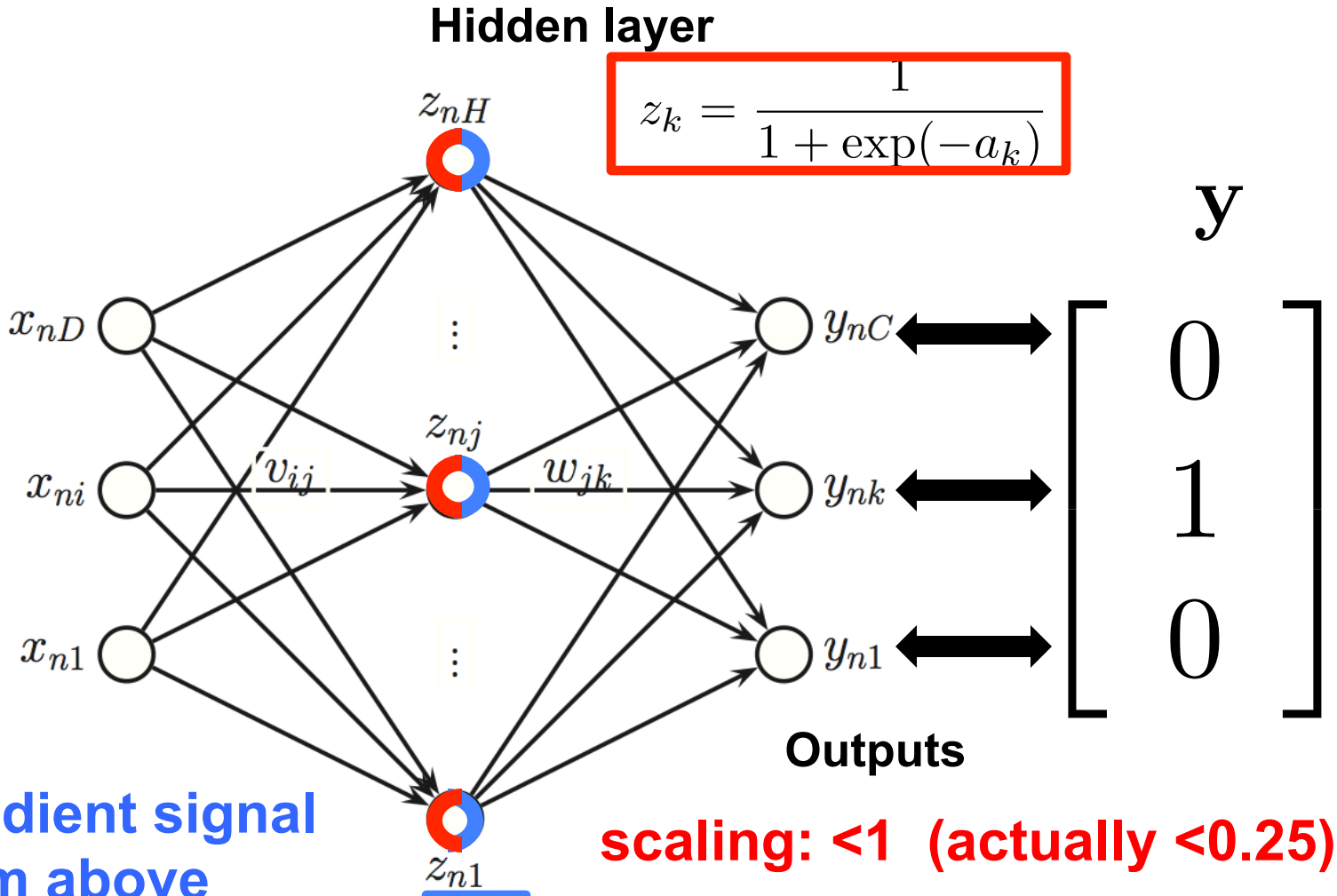
$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1-g(a_k))$$

A neural network in backward mode:



$$\frac{\partial l}{\partial v_{ij}} = \sum_k \frac{\partial l}{\partial a_k}\frac{\partial a_k}{\partial v_{ij}} = \frac{\partial l}{\partial a_j} x_i$$

A neural network in backward mode:



Gradient signal from above

scaling: <1 (actually <0.25)

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1-g(a_k))$$

Vanishing gradients problem

Gradient signal from above

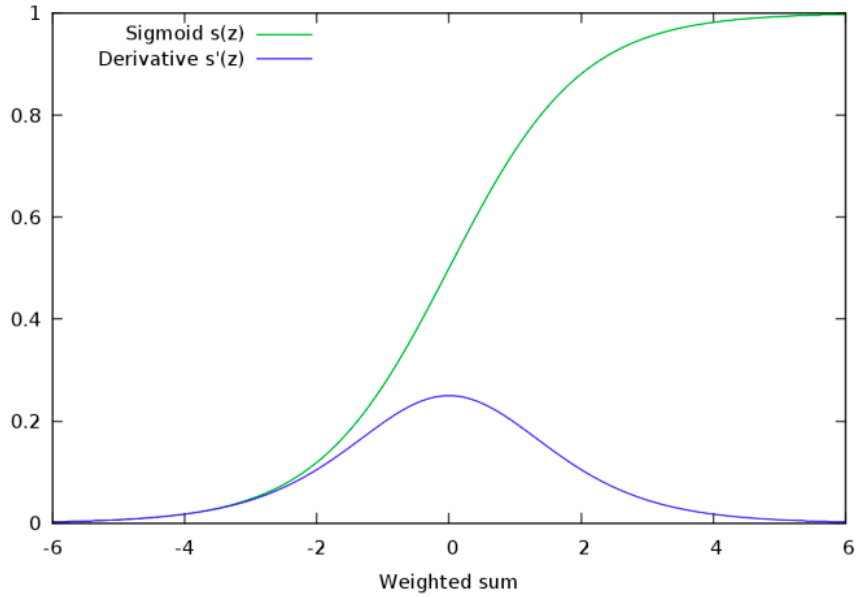
scaling: <1 (actually <0.25)

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k))$$

Do this 10 times: updates in the first layers get minimal

Top layer knows what to do, lower layers "don't get it"

Sigmoidal Unit:
Signal is not getting through!



Vanishing gradients problem: ReLU solves it

Gradient signal from above

Scaling: {0,1}

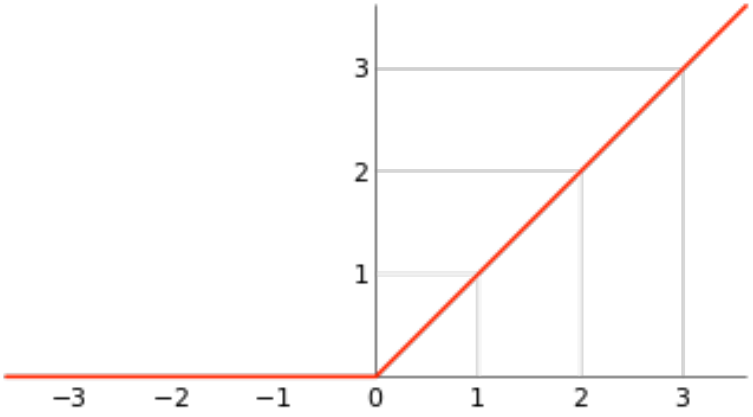
$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k)$$

Do this 10 times: updates in the first layers can remain large

Top layer knows what to do, lower layers " get it"

Rectified Linear Unit (RELU)

$$g(a) = \max(0, a)$$



$$g'(a) = \begin{cases} 1 & a > 0 \\ 0 & a < 0 \end{cases}$$

Vinod Nair and Geoffrey Hinton (2010). Rectified linear units improve restricted Boltzmann machines. ICML.

Neural network training: old & new tricks

Old: (80's)

- Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

- Dropout
- ReLUs
- Batch Normalization
- Residual Networks

Covariate Shift Problem

Train speech recognition system with British speakers, test with Americans

- Traditional machine learning must contend with *covariate shift* between data sets.

blog.bigml.com

Covariate shift in a single day

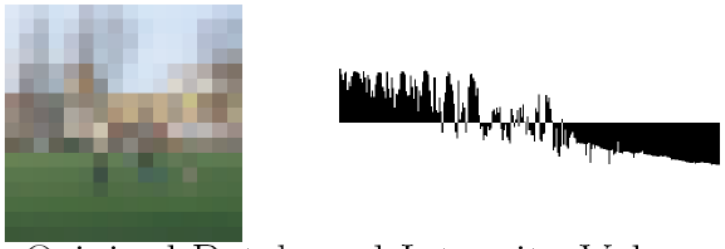
10 am

2pm

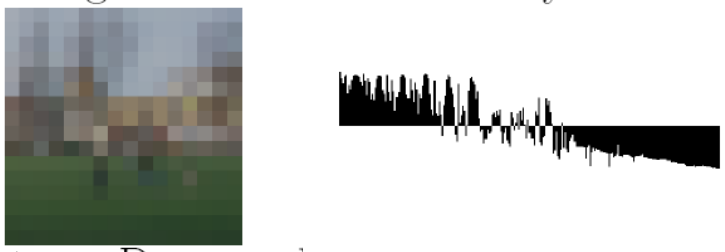
7pm

“Whitening”: set mean = 0, variance = 1

Photometric transformation: $I \rightarrow a\,I + b$



Original Patch and Intensity Values



Brightness Decreased

Contrast increased,

- Make each patch have zero mean:

$$\mu = \frac{1}{N}\sum_{x,y} I(x,y)$$

$$Z(x,y) = I(x,y) - \mu$$

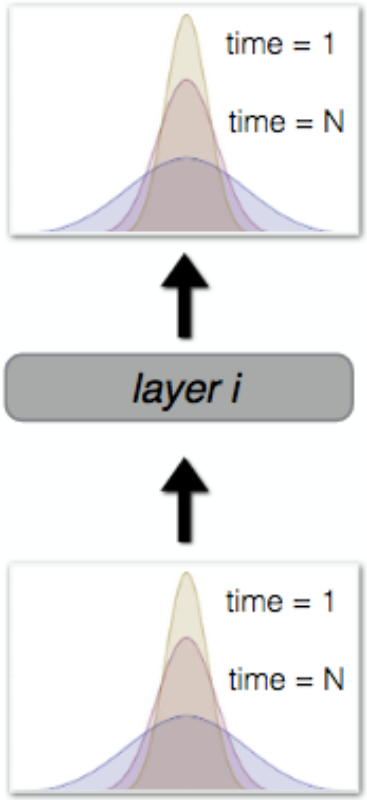
- Then make it have unit variance:

$$\sigma^2 = \frac{1}{N}\sum_{x,y} Z(x,y)^2$$

$$ZN(x,y) = \frac{Z(x,y)}{\sigma}$$

Internal Covariate Shift

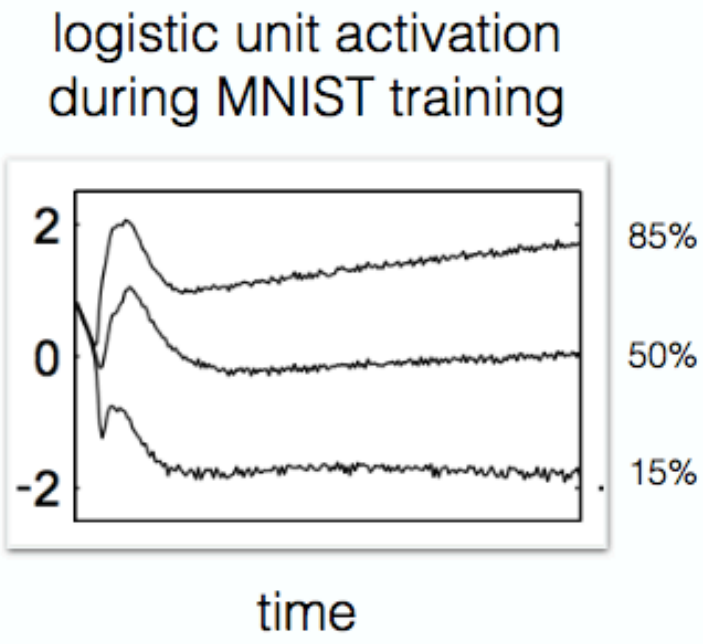
Neural network activations: moving target



Internal Covariate Shift

Neural network activations: moving target

- Covariate shifts occur across layers in a deep network.
- Performing domain adaptation or whitening is impractical in an online setting.



Batch Normalization

Whiten-as-you-go:

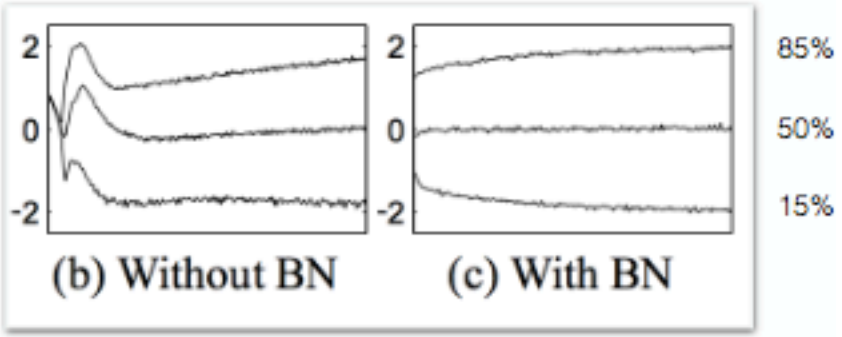
- Normalize the activations in each layer within a mini-batch.
- Learn the mean and variance (γ, β) of each layer as parameters

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

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

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

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



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
S Ioffe and C Szegedy (2015)

Batch Normalization

Whiten-as-you-go:

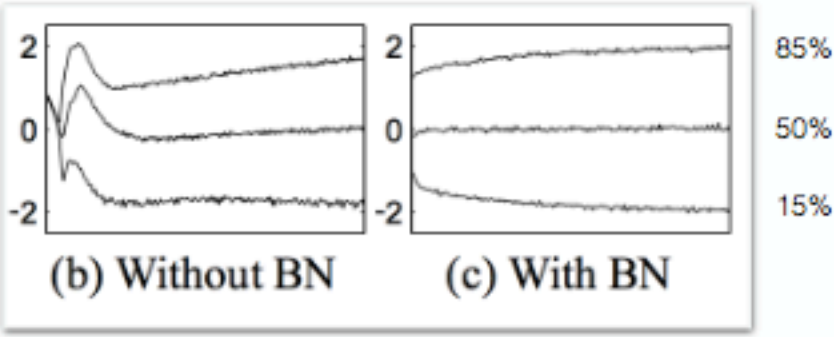
- Normalize the activations in each layer within a mini-batch.
- Learn the mean and variance (γ, β) of each layer as parameters

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

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

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

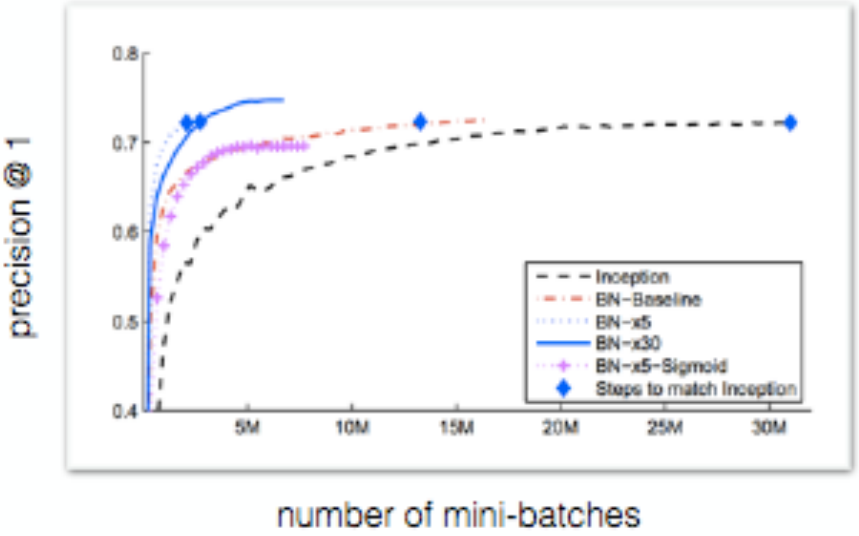
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
S Ioffe and C Szegedy (2015)

Batch Normalization: used in all current systems

- Multi-layer CNN's train faster with fewer data samples (15x).
- Employ faster learning rates and less network regularizations.
- Achieves state of the art results on ImageNet.



Neural network training: old & new tricks

Old: (80's)

- Stochastic Gradient Descent, Momentum, "weight decay"

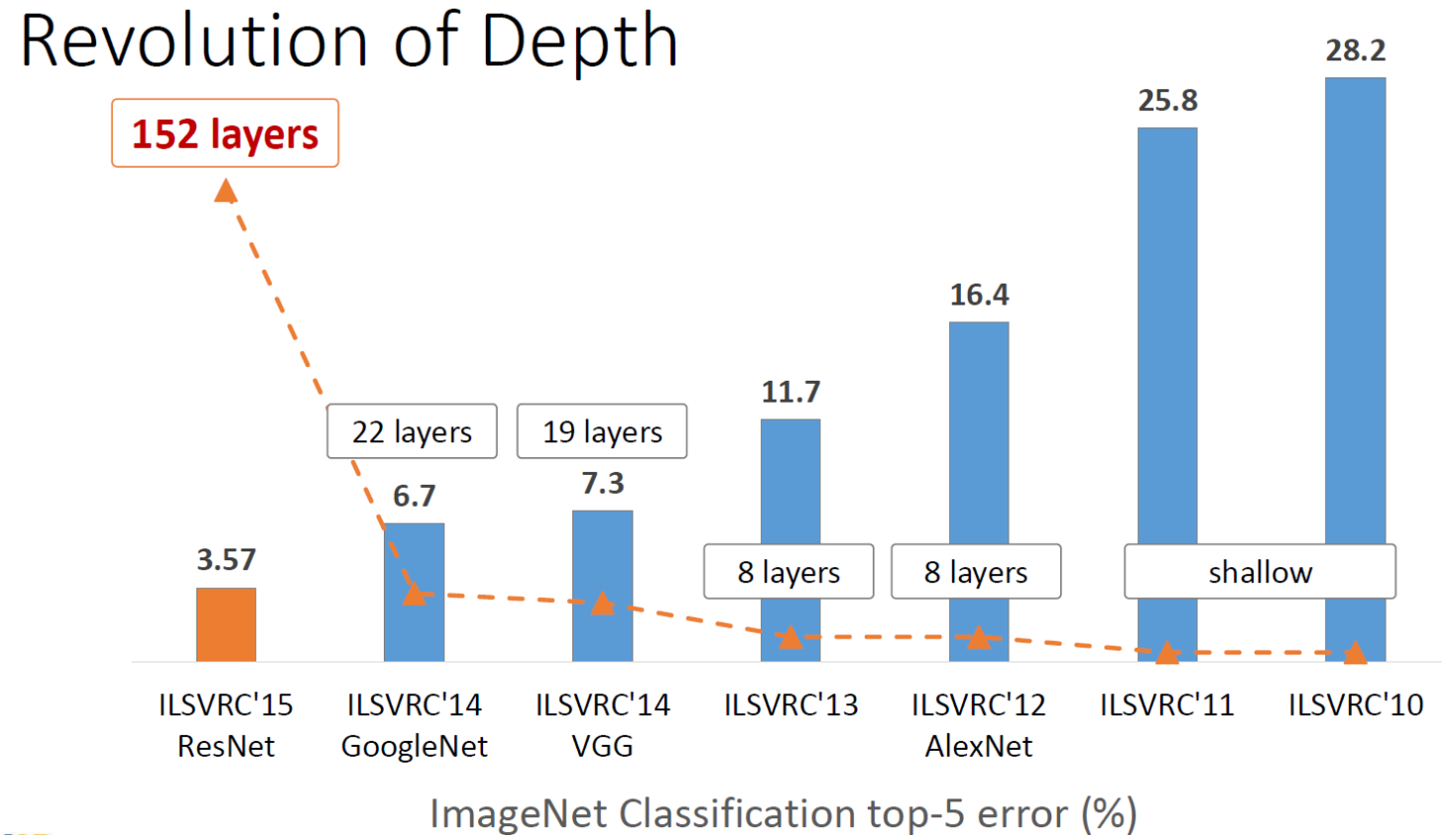
New: (last 5-6 years)

- Dropout
- ReLUs
- Batch Normalization
- Residual Networks

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

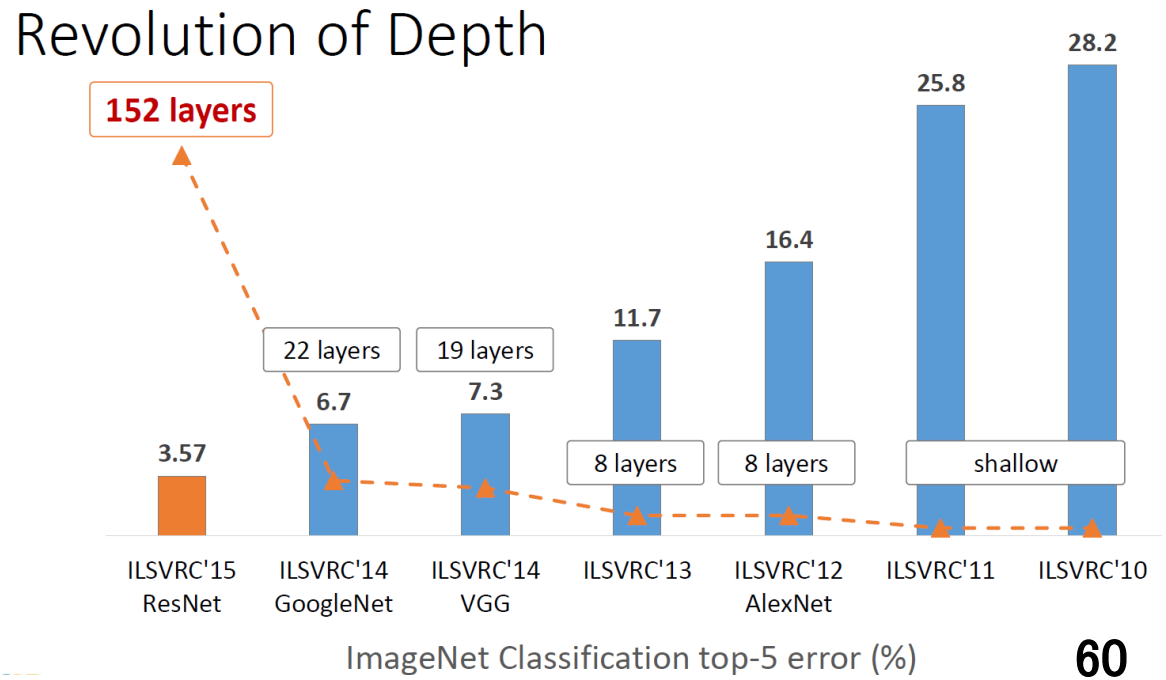
The deeper, the better

- Deeper networks can cover more complex problems
  - Increasingly large receptive field size
  - Increasingly non-linear patterns



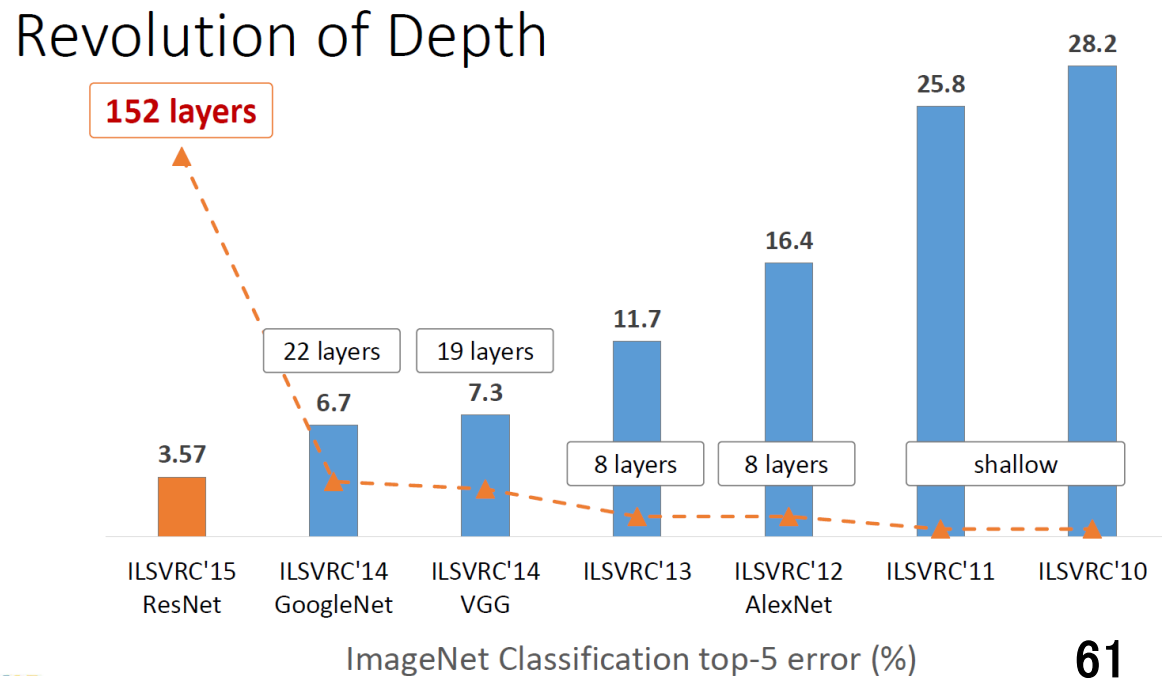
Going Deeper

- From 2 to 10: 2010-2012
  - ReLUs
  - Dropout
  - …



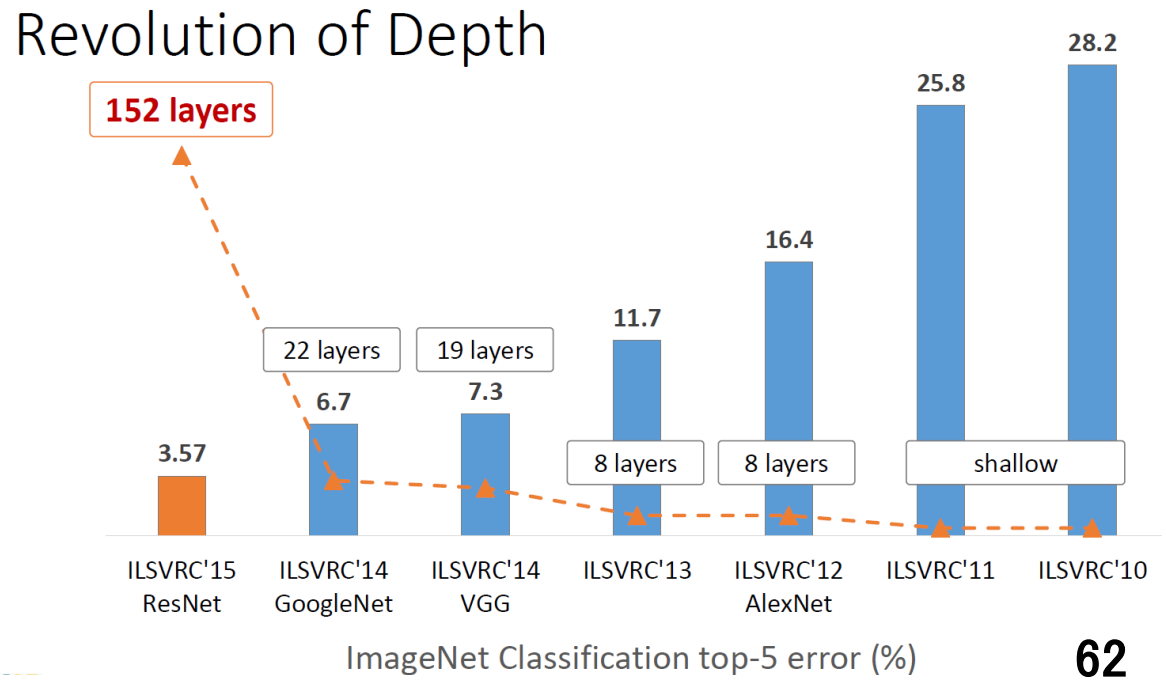
Going Deeper

- From 10 to 20: 2015
  - Batch Normalization



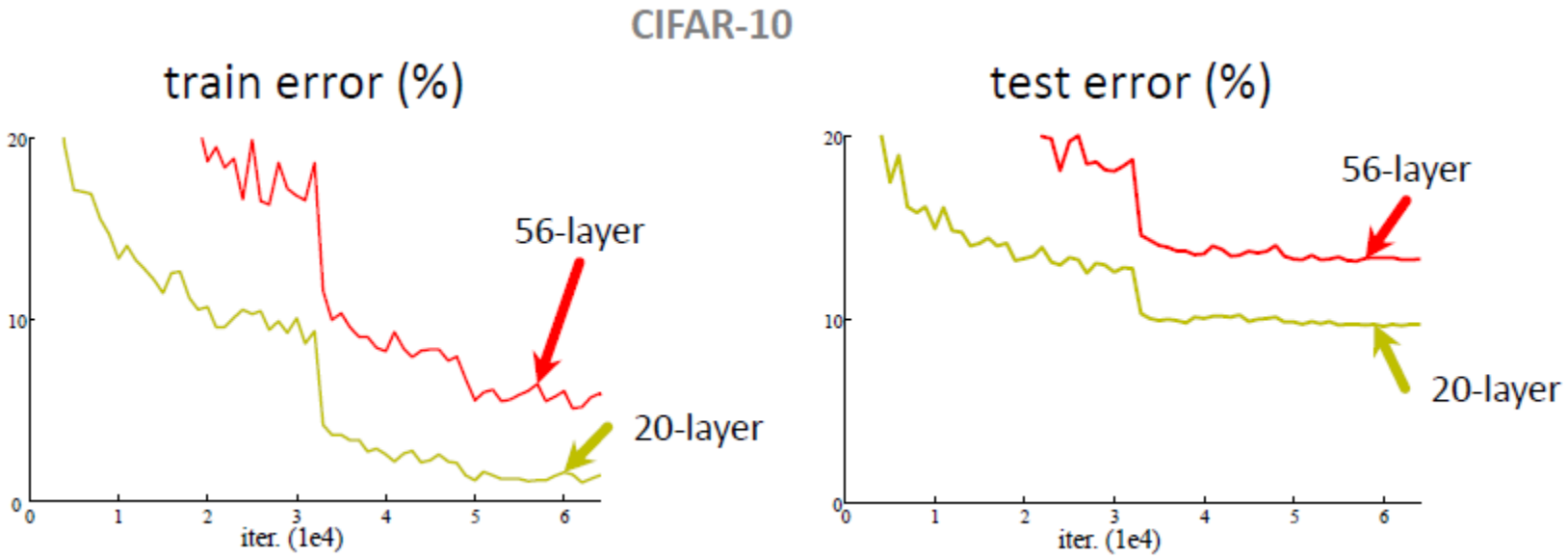
Going Deeper

- From 20 to 100/1000
  - Residual networks



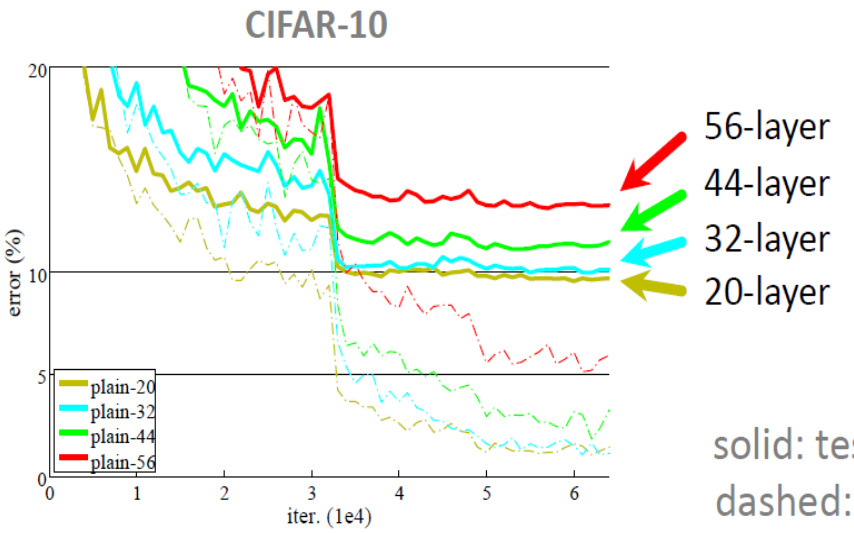
Plain Network

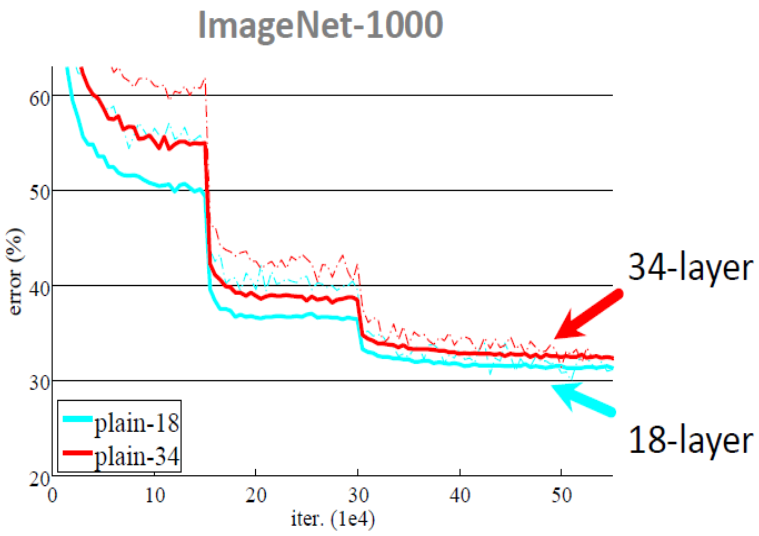
- Plain nets: stacking 3x3 conv layers
- 56-layer net has higher training error and test error than 20-layers net



Plain Network

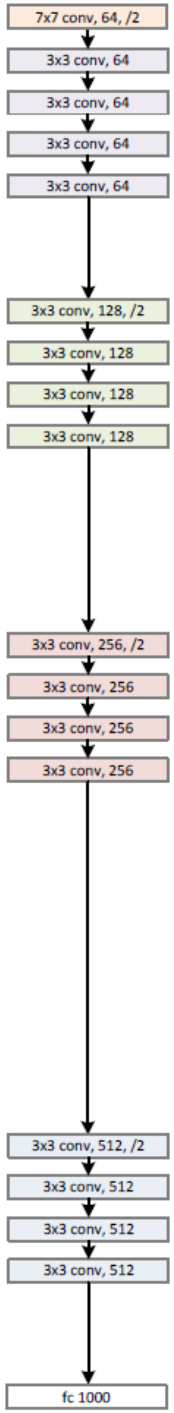
- “Overly deep” plain nets have higher training error
- A general phenomenon, observed in many datasets

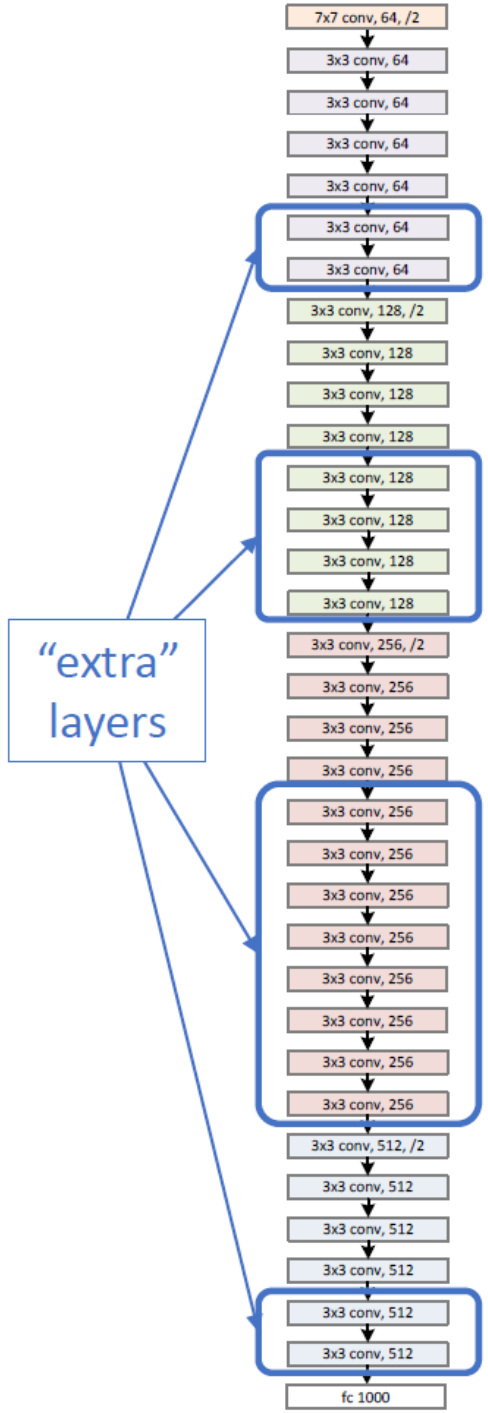




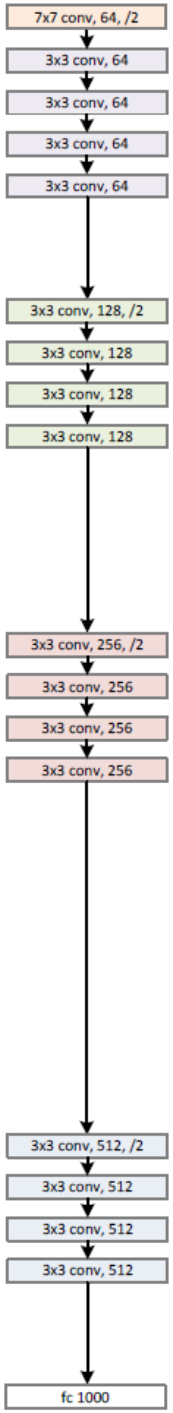
Residual Network

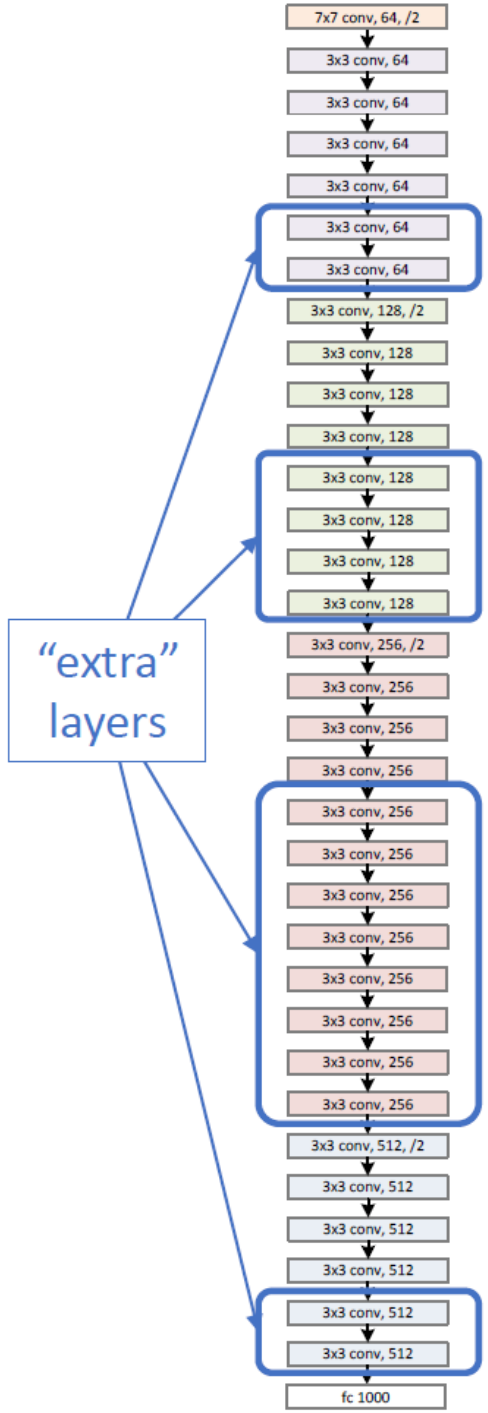
- Naïve solution
  - If extra layers are an identity mapping, then training errors can not increase





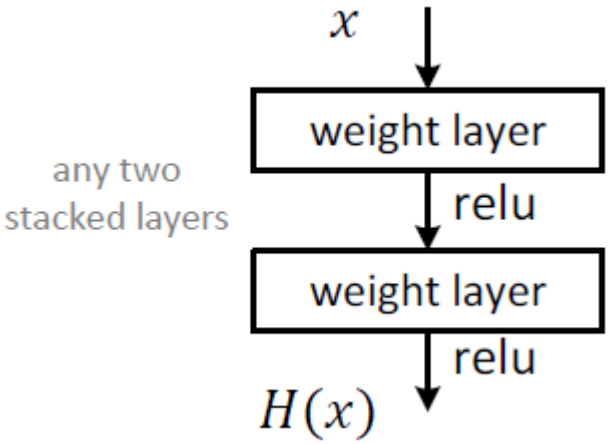
Residual Network





Residual Network

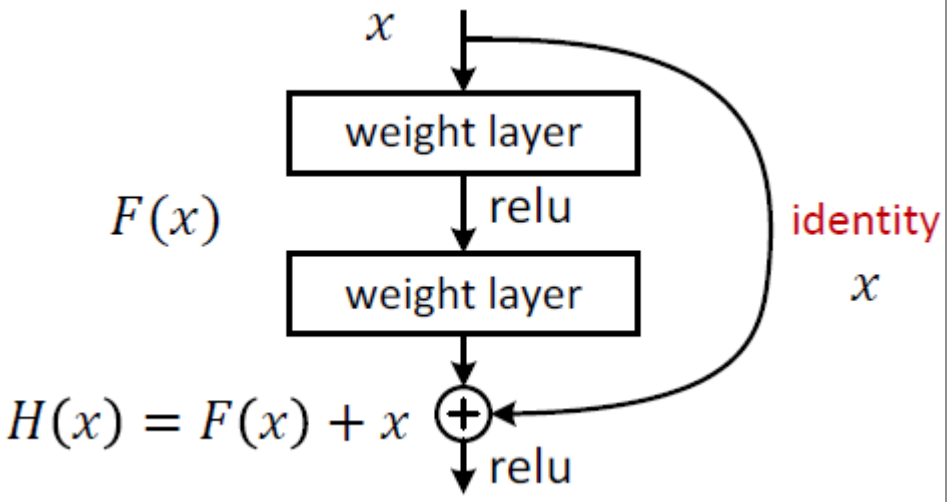
- Plain block
  - Difficult to make identity mapping because of multiple non-linear layers



Residual Network

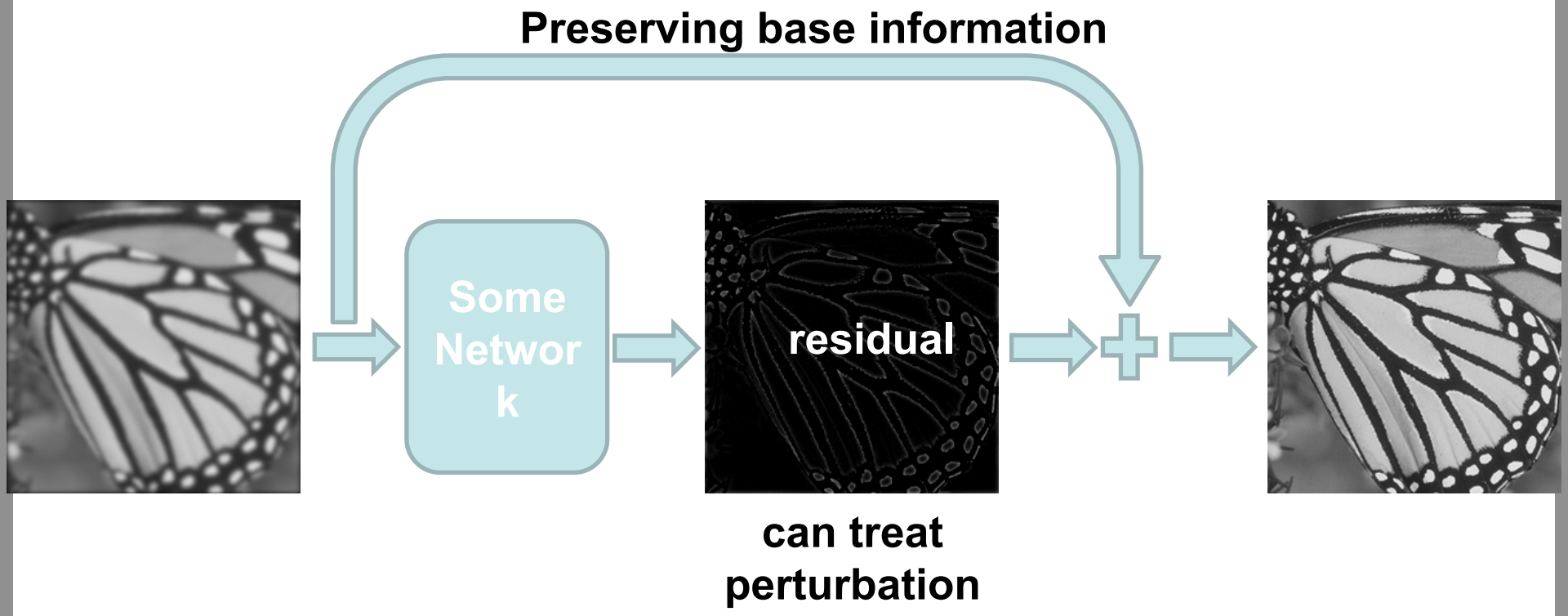
- Residual block
  - If identity were optimal, easy to set weights as 0
  - If optimal mapping is closer to identity, easier to find small fluctuations

-> Appropriate for treating perturbation as keeping a base information



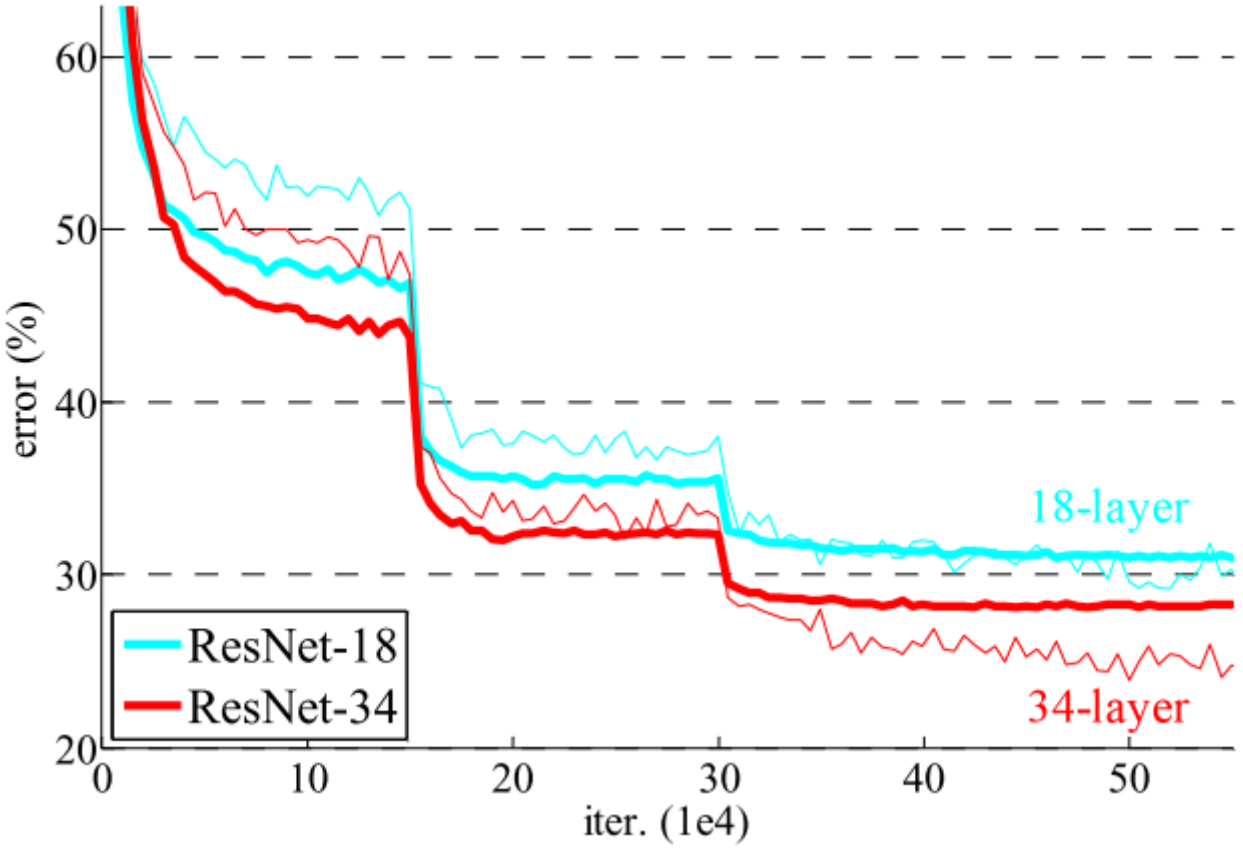
Residual Network

- Difference between an original image and a changed image



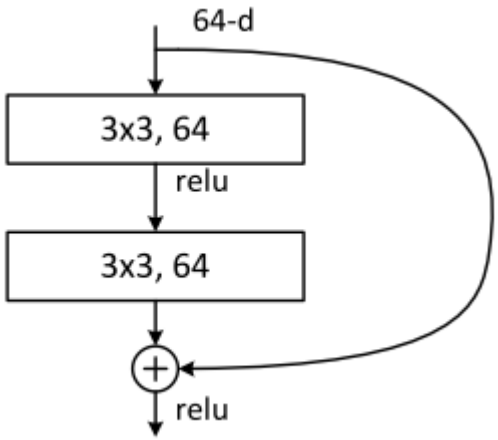
Residual Network

- Deeper ResNets have lower training error

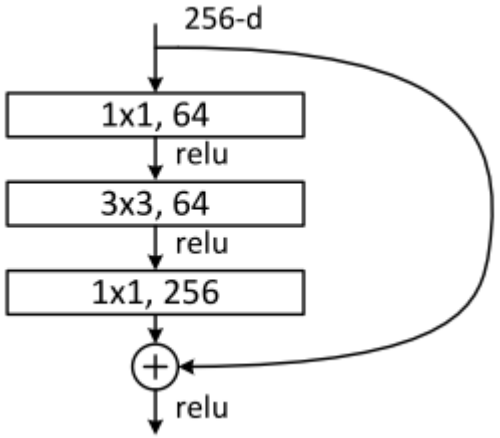


Residual Network

- Residual block
  - Very simple
  - Parameter-free



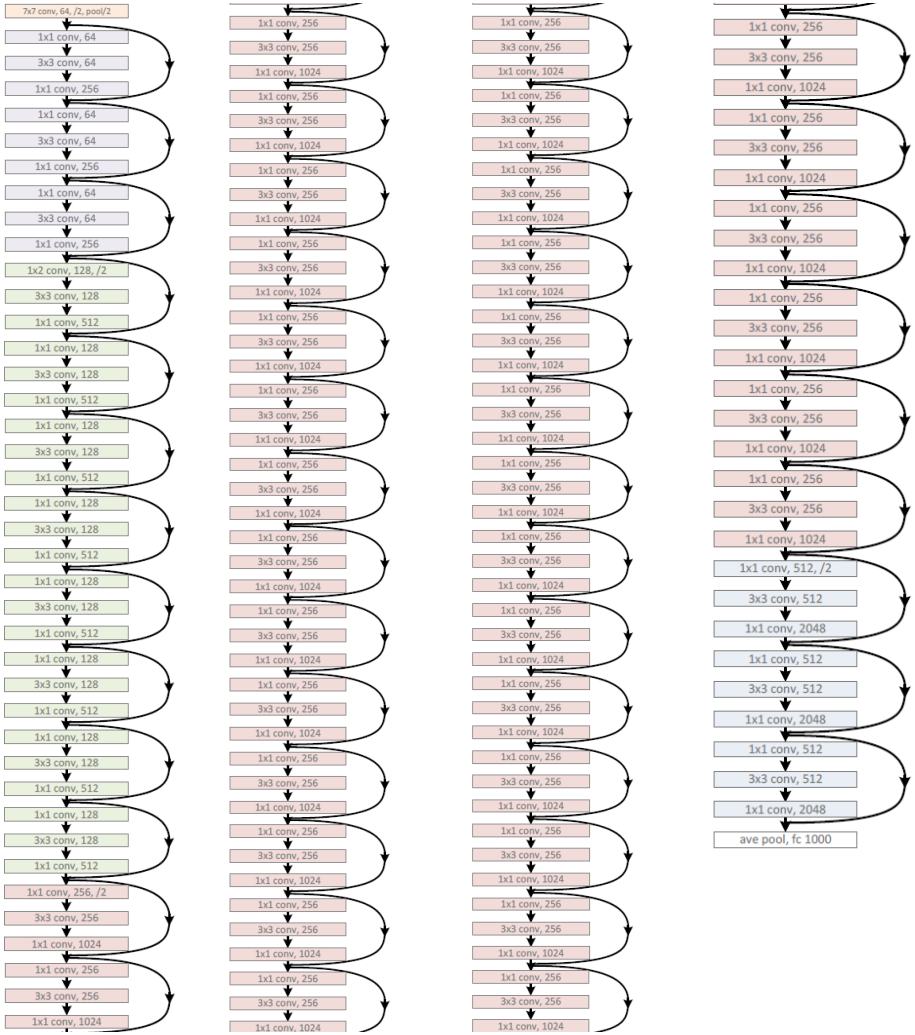
A naïve residual block



"bottleneck" residual block (for ResNet-50/101/152)

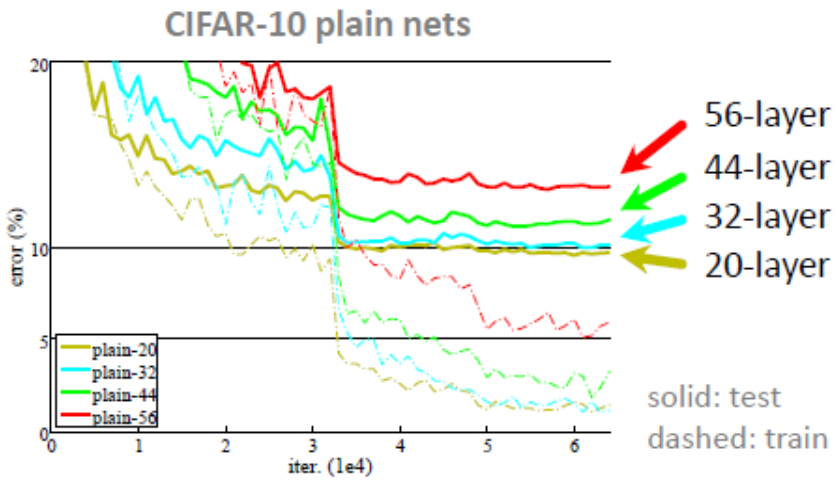
Network Design

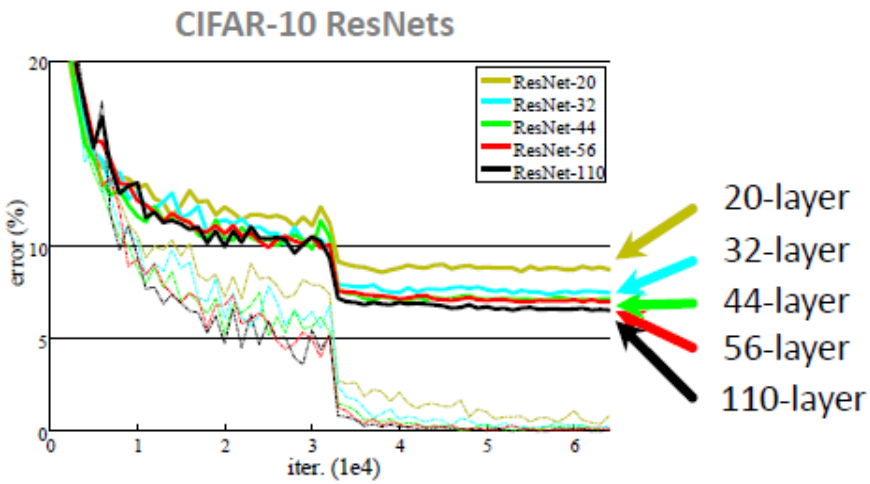
- ResNet-152
  - Use bottlenecks
  - ResNet-152(11.3 billion FLOPs) has lower complexity than VGG-16/19 nets (15.3/19.6 billion FLOPs)



Results

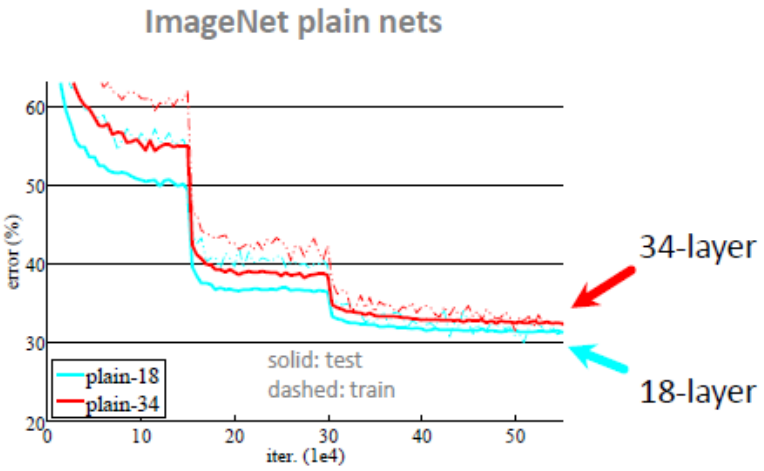
- Deep Resnets can be trained without difficulties
- Deeper ResNets have lower training error, and also lower test error

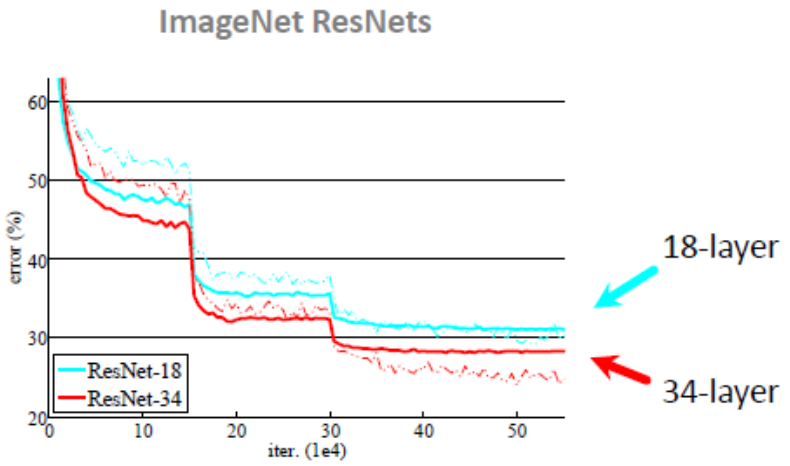




Results

- Deep Resnets can be trained without difficulties
- Deeper ResNets have lower training error, and also lower test error



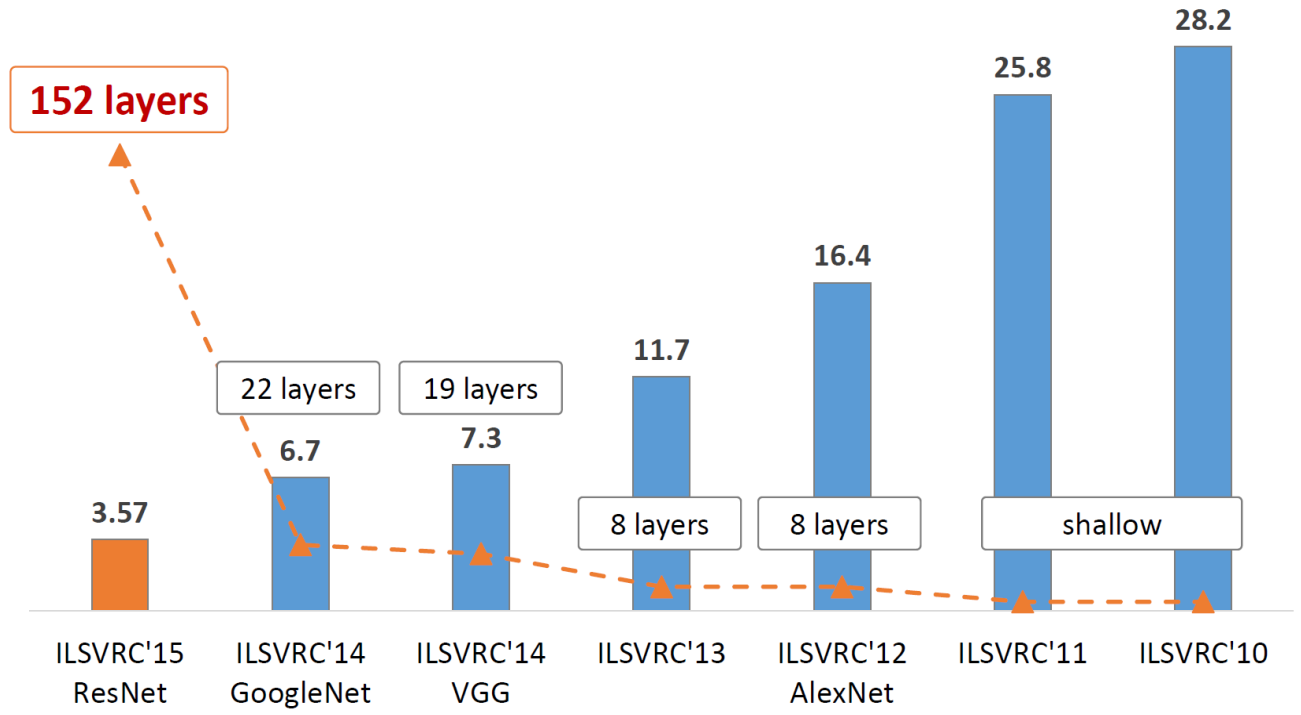


Results

- 1st places in all five main tracks in “ILSVRC & COCO 2015 Competitions”
  - ImageNet Classification
  - ImageNet Detection
  - ImageNet Localization
  - COCO Detection
  - COCO Segmentation

Quantitative Results

- ImageNet Classification



Result

- Performances increase absolutely

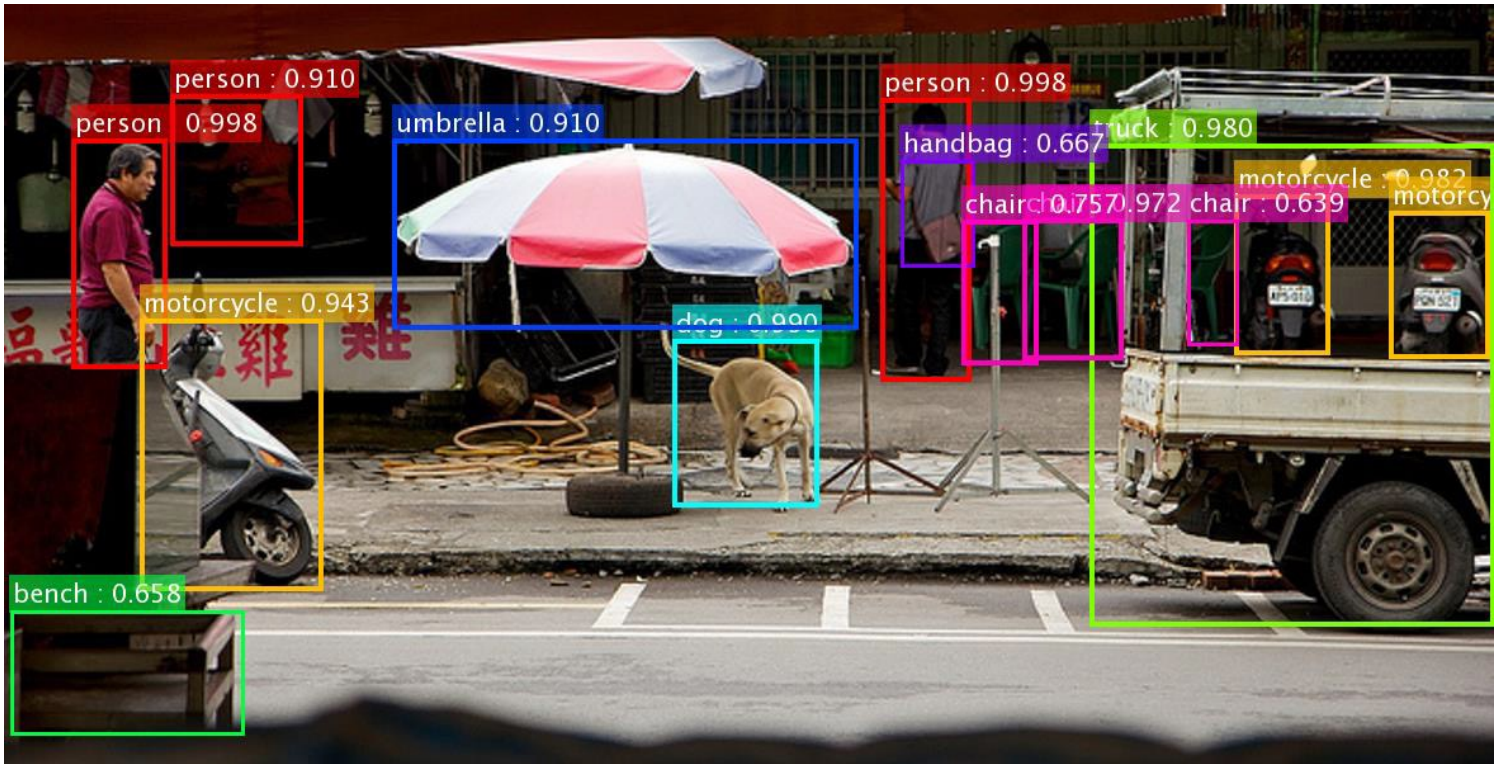
task	2nd-place winner	MSRA	margin (relative)
ImageNet Localization (top-5 error)	12.0	9.0	**27%**
ImageNet Detection (mAP@.5)	53.6	62.1	**16%**
COCO Detection (mAP@.5:.95)	33.5	37.3	**11%**
COCO Segmentation (mAP@.5:.95)	25.1	28.2	**12%**

absolute 8.5% better!

- Based on ResNet-101
- Existing techniques can use residual networks or features from it

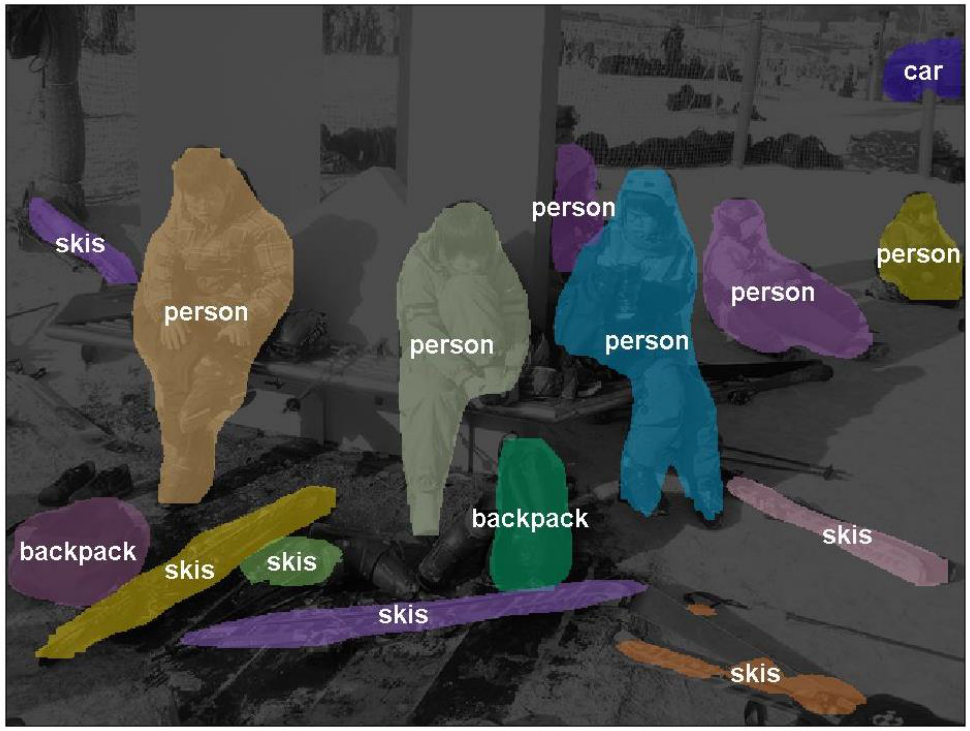
Qualitative Result

- Object detection
  - Faster R-CNN + ResNet



Qualitative Results

- Instance Segmentation



Detection in the wild with ResNets

https://www.youtube.com/watch?v=WZmSMkK9VuA