The Devil is always in the Details
— Must Know Tips/Tricks in DNNs

Nanjing University
Xiu-Shen Wei
魏秀参

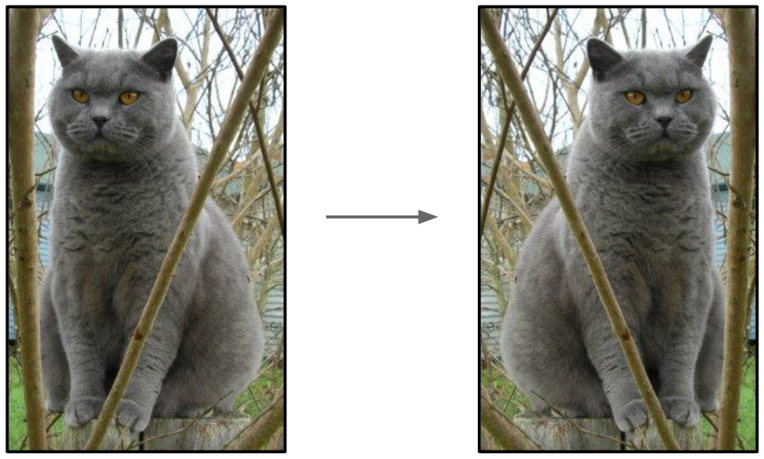
Sep. 9, 2015

Outline

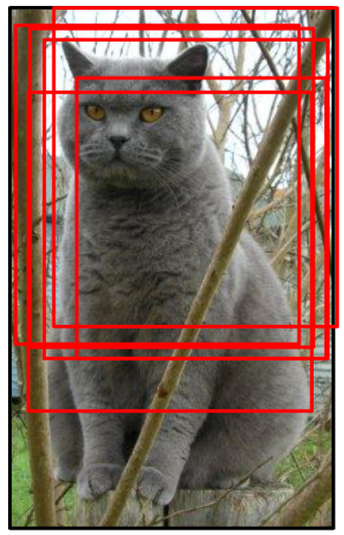
Learning And Mining from DatA

- ☑ Data augmentation
- ☑ Pre-processing
- ☑ Initializations
- ☑ During training
- ☑ Activation functions
- ☑ Regularizations
- ☑ Insights from figures
- ☑ Ensemble

Data augmentation



Flip horizontally



Random crops/scales

Color jittering

Random mix/combinations of :
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Data augmentation (con't)

Learning And Mining from DatA

Fancy PCA:
Consist of altering the intensities of the RGB channels.

1. Compute PCA on all [R,G, B] points values in the training data;

2. Sample some color offset along the principal components at each forward pass;

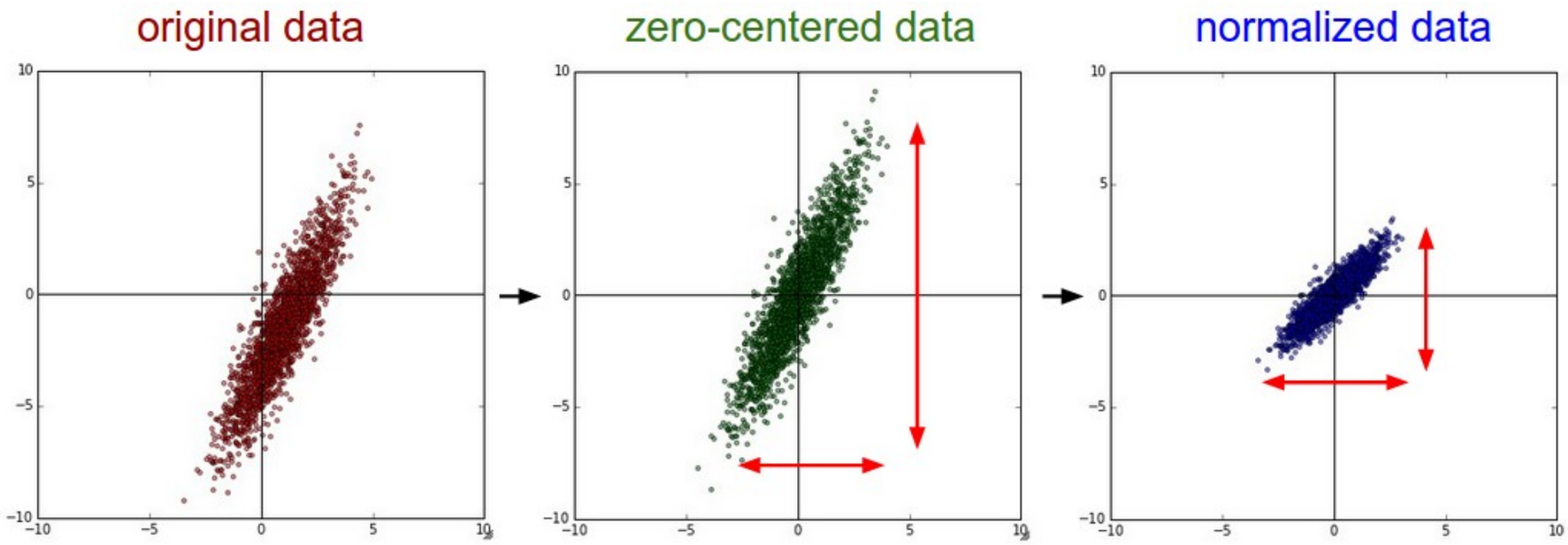
3. Add the offset to all pixels in a training image.

$$I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B]^T \qquad [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

This scheme reduces the top-1 error rate by over 1%.

[A. Krizhevsky et al., NIPS'12]

Pre-processing



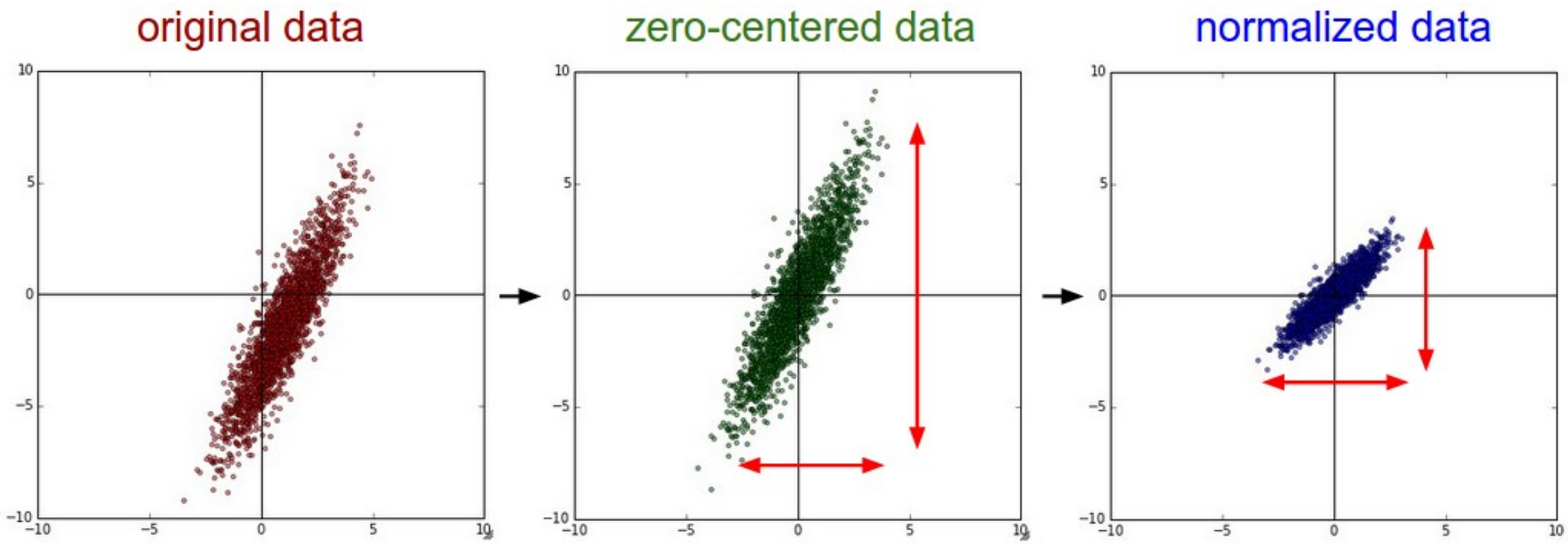

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Another way to normalize data is making the min and max along the dimension be -1 and 1, respectively.

Figures courtesy of Stanford course CS231n.

Pre-processing




```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Another way to normalize data is making the min and max along the dimension be -1 and 1, respectively.

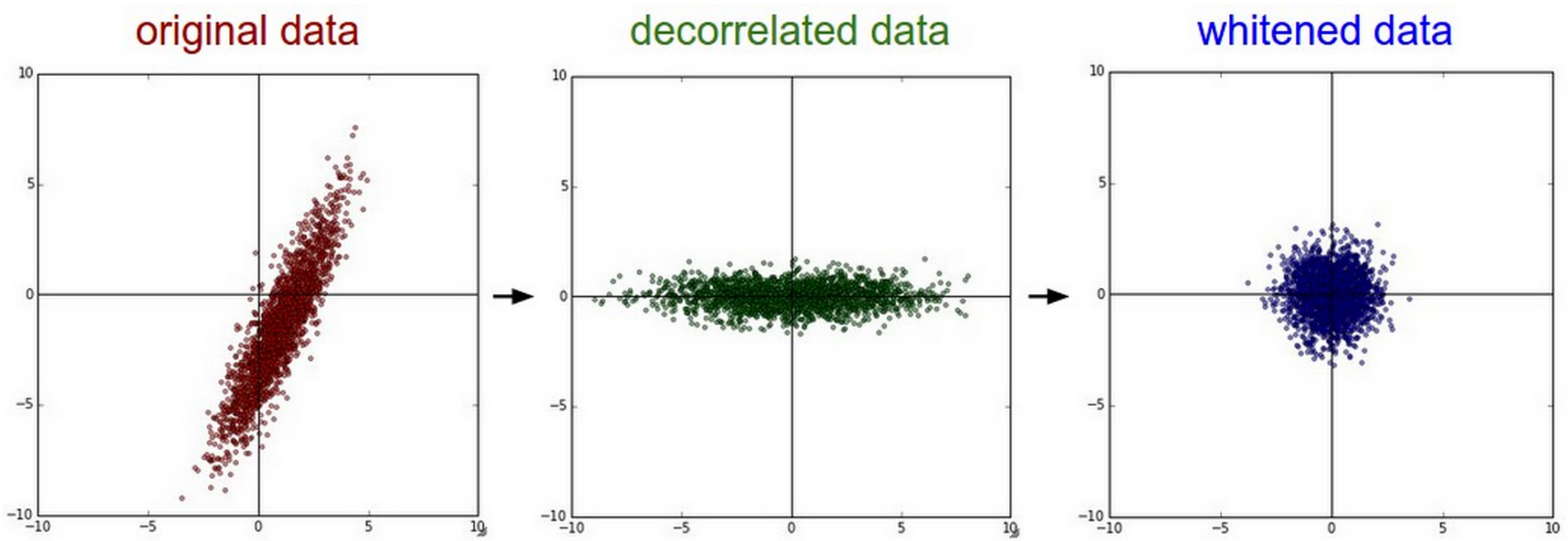
It is **not** strictly necessary to perform this additional preprocessing step for the case of **IMAGE**.

Figures courtesy of Stanford course CS231n.

Pre-processing (con't)

Learning And Mining from DatA

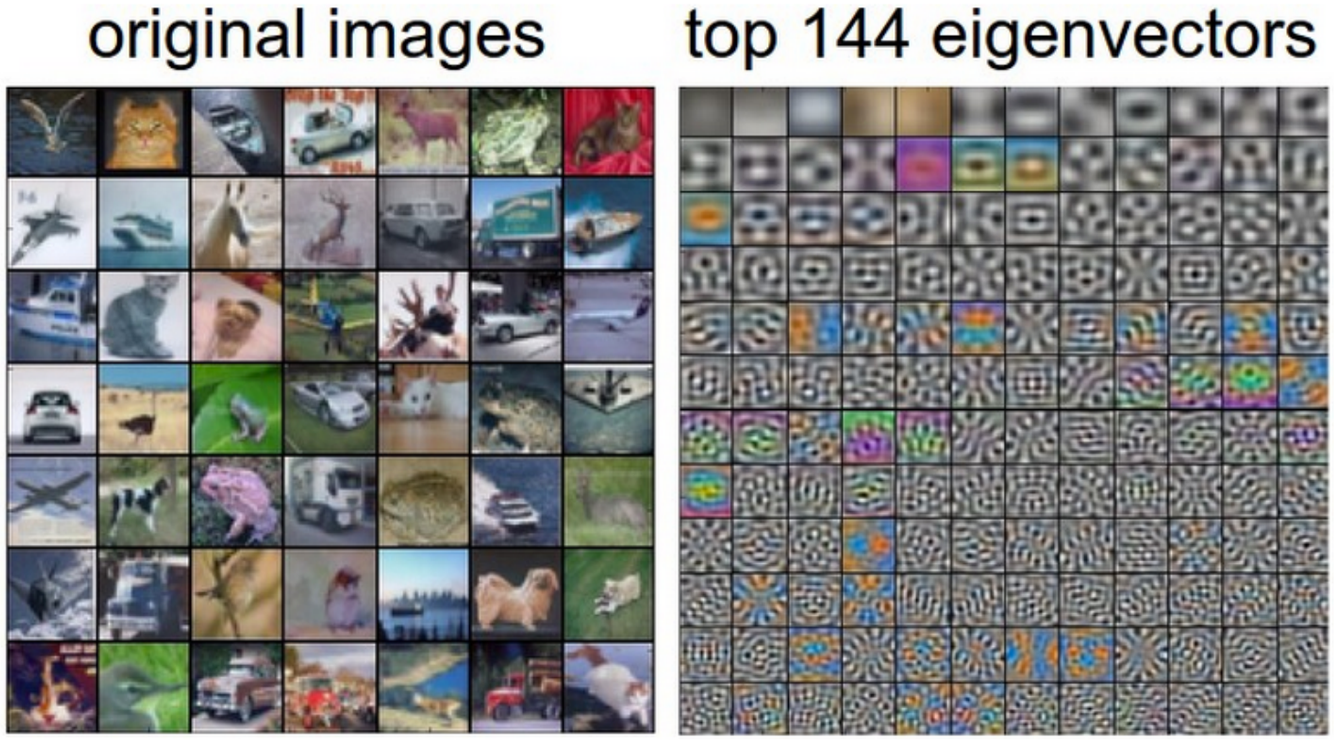
PCA-Whitening:




```
Xwhite = Xrot / np.sqrt(S + 1e-5)
```

Exaggerating noise!

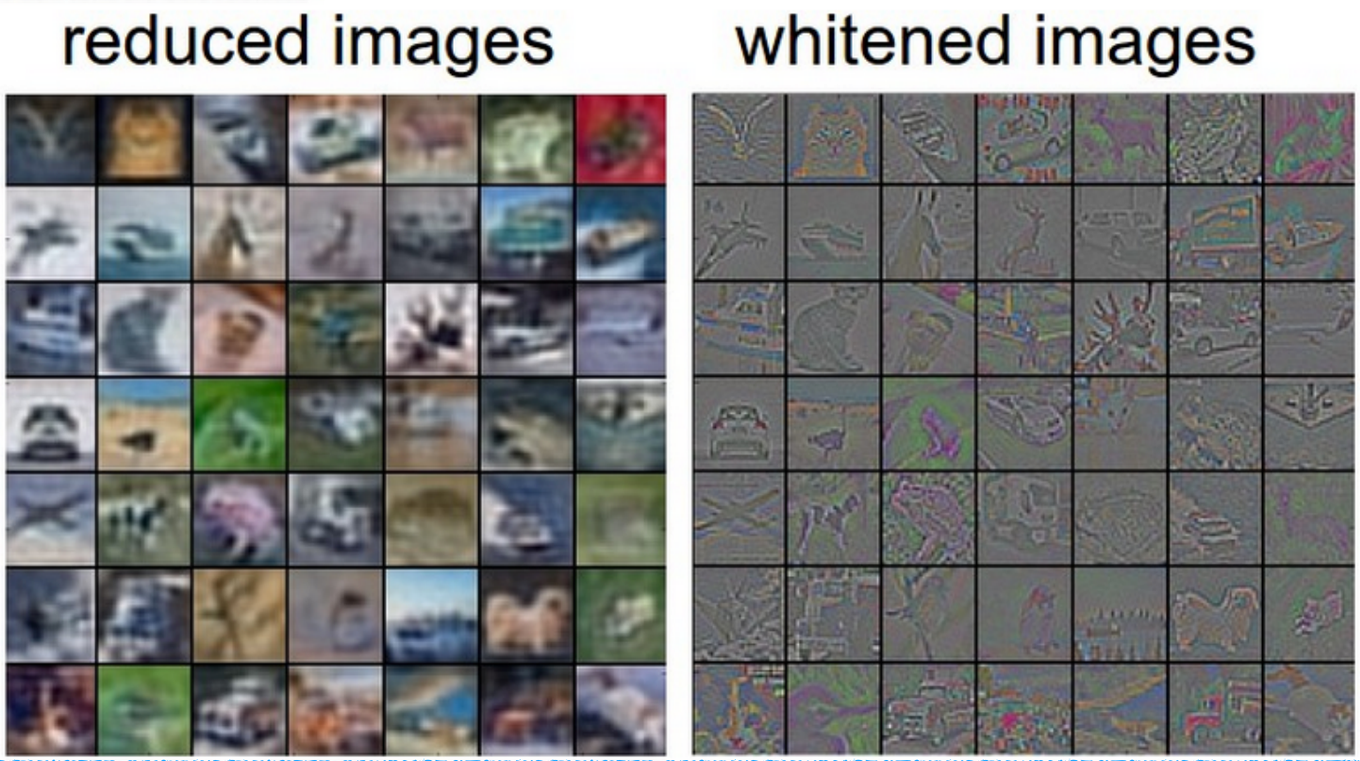
PCA-whitening: a demo



CIFAR-10: [50000*3072]

Not used in CNN.

However, zero-center the data is necessary for CNN!



Initialization

With proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative.

Initialization

With proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative.

How about **all zero initialization**?

Initialization

With proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative.

How about **all zero initialization**?

Initialization

With proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative.

How about **all zero initialization**?

They will also all compute the same gradients during backpropagation and undergo the exact same parameter updates. **— No source of asymmetry.**

Small random numbers

We still want the weights to be very close to 0.

$$weights \sim 0.001 \times N(0, 1)$$

Small random numbers

We still want the weights to be very close to 0.

$$weights \sim 0.001 \times N(0, 1)$$

Uniform distribution is also ok.

Small random numbers

We still want the weights to be very close to 0.

$$weights \sim 0.001 \times N(0, 1)$$

Uniform distribution is also ok.

Warning! Small numbers will diminish the "gradient signal" flowing backward through a network.

Calibrating the variances

The distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs.

```
w = np.random.randn(n) / sqrt(n)
```

This ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

Calibrating the variances

The distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs.

```
w = np.random.randn(n) / sqrt(n)
```

This ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

Calibrating the variances (con't)

$$
\begin{aligned}
\mathrm{Var}(s) &= \mathrm{Var}\Big(\sum_{i}^{n} w_i x_i\Big) \\
&= \sum_{i}^{n} \mathrm{Var}(w_i x_i) \\
&= \sum_{i}^{n} [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= \sum_{i}^{n} \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= (n\mathrm{Var}(w))\ \mathrm{Var}(x)
\end{aligned}
$$

Calibrating the variances (con't)

the raw activation before non-linear filters

$$\begin{aligned}
\mathrm{Var}(s) &= \mathrm{Var}\left(\sum_i^n w_i x_i\right) \\
&= \sum_i^n \mathrm{Var}(w_i x_i) \\
&= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= (n\mathrm{Var}(w))\ \mathrm{Var}(x)
\end{aligned}$$

Calibrating the variances (con't)

Learning And Mining from DatA

the raw activation before non-linear filters

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right)$$

i.i.d.

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= (n\mathrm{Var}(w))\ \mathrm{Var}(x)$$

Calibrating the variances (con't)

Learning And Mining from DatA

the raw activation before non-linear filters

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right)$$

i.i.d.

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

assumptions: $E(w_i)=E(x_i)=0$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= (n\mathrm{Var}(w))\ \mathrm{Var}(x)$$

Calibrating the variances (con't)

Learning And Mining from DatA

the raw activation before non-linear filters

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right)$$

i.i.d.

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

assumptions: $E(w_i)=E(x_i)=0$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= (n\mathrm{Var}(w))\mathrm{Var}(x)$$

equals one

Calibrating the variances (con't)

Learning And Mining from DatA

the raw activation before non-linear filters

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right)$$

i.i.d.

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

assumptions: $E(w_i)=E(x_i)=0$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= (n\mathrm{Var}(w))\mathrm{Var}(x)$$

equals one

$$nVar(w) = Var(w/\sqrt{n})$$

```
w = np.random.randn(n) / sqrt(n)
```

Current recommendation

They reached the conclusion that the variance of neurons in the network should be **2.0/n**.

```
w = np.random.randn(n) * sqrt(2.0/n)
```

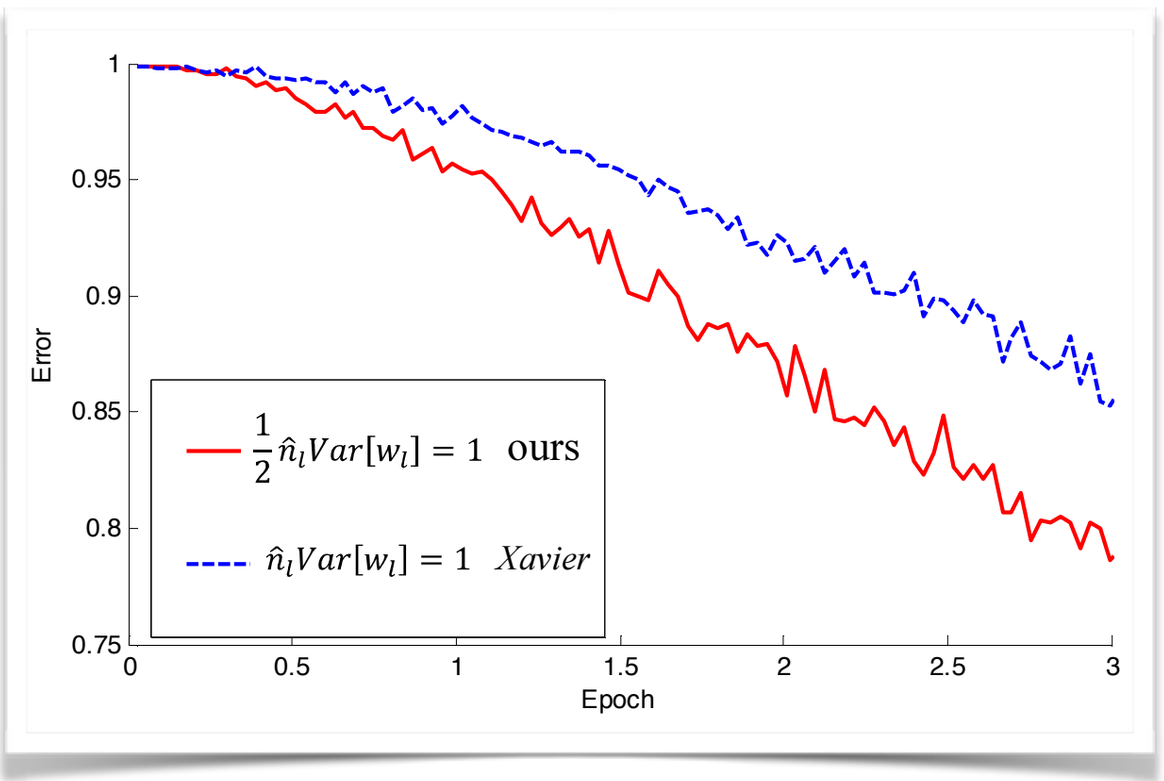


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

[K. He, et al., ICCV'15]

During training

LAMDA
Learning And Mining from DatA

Eliminate sizing headaches TIPS/TRICKS:

- start with image that has power-of-2 size
- for **conv layers**, use stride 1 filter size 3*3 pad input with a border of zeros (1 spatially)

This makes it so that: [W1, H1, **D1**] -> [W1, H1, **D2**] (i.e., spatial size exactly preserved)

- for **pool layers**, use pool size 2*2 (more = worse)

Learning

LAMDA
Learning And Mining from DatA

Gradient normalization:

Divide the gradients by minibatch size. You won't need to change the learning rates (not too much, anyway), if you double the minibatch size (or halve it).

Learning rate (LR) schedule:

- A typical value of the LR is **0.1**;
- Use **a validation set**;
- Practical suggestion: if you see that you stopped making progress on the validation set, divide the LR by 2 (or by 5), and keep going.

[A blog by Ilya Sutskever, 2015]

Fine-tune

Learning And Mining from DatA

	very similar dataset	very different dataset
very little data	Use linear classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a large number of layers

Fine-tune

Learning And Mining from DatA

	very similar dataset	very different dataset
very little data	Use linear classifier on top la	You're in trouble… Try linear classifier
quite a lot of data	Finetune a few lay	Finetune a large

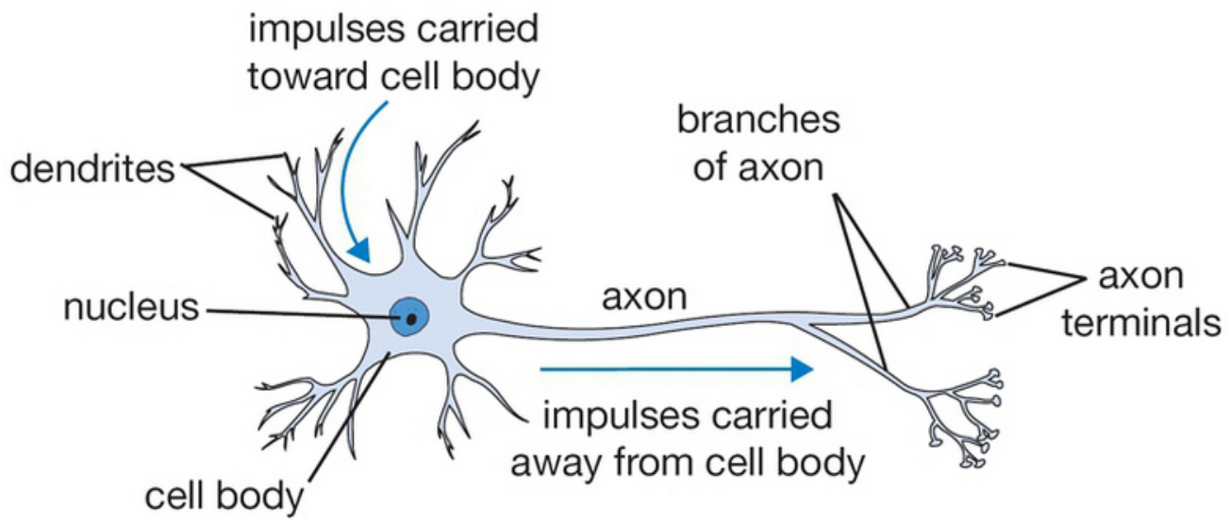
randomly-initialized

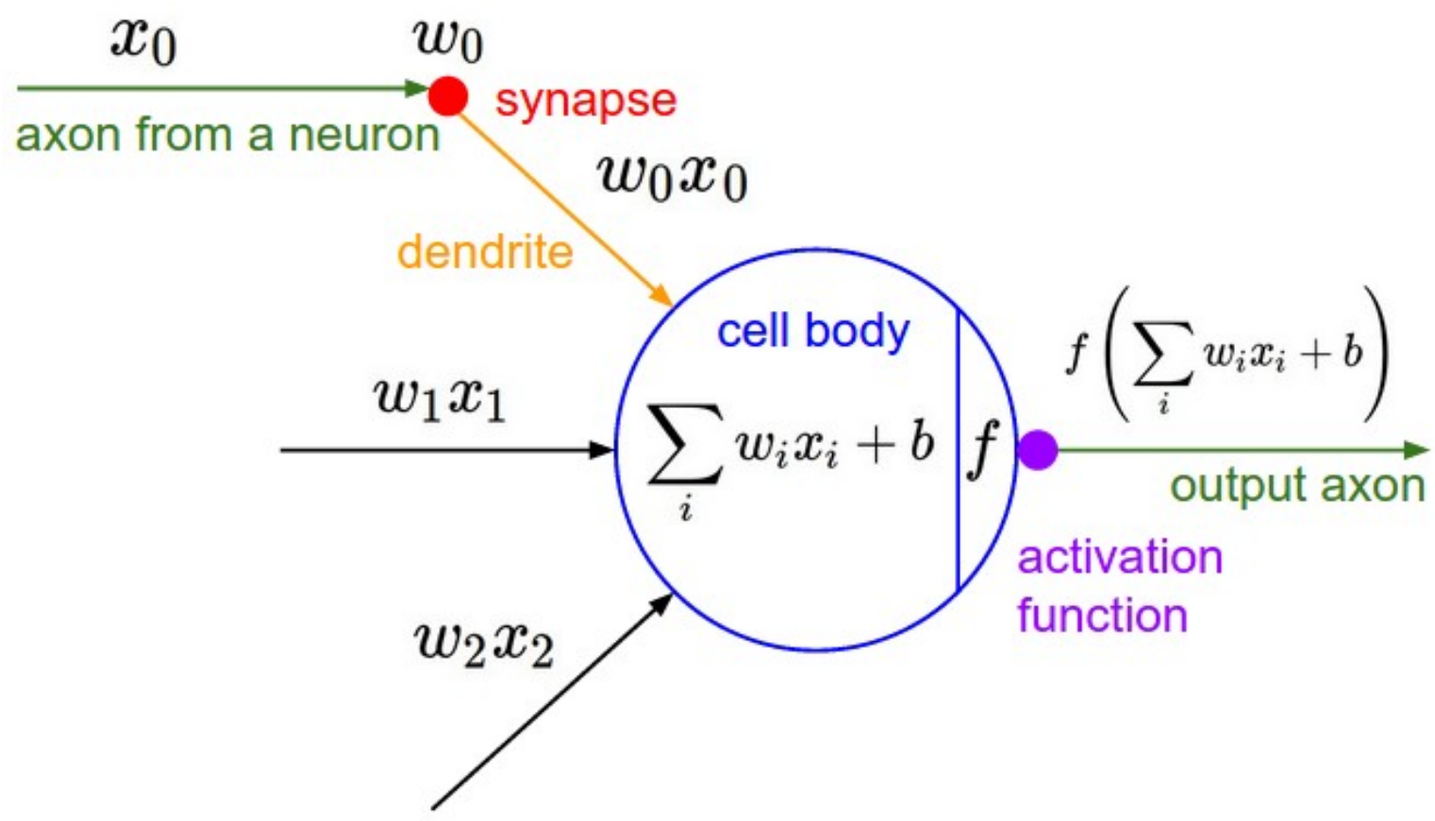
a smaller learning rate

Activation functions

- Sigmoid
- tanh
- ReLU (Rectified Linear Unit)
- Leaky ReLU
- Parametric ReLU
- Randomized ReLU

Activation functions





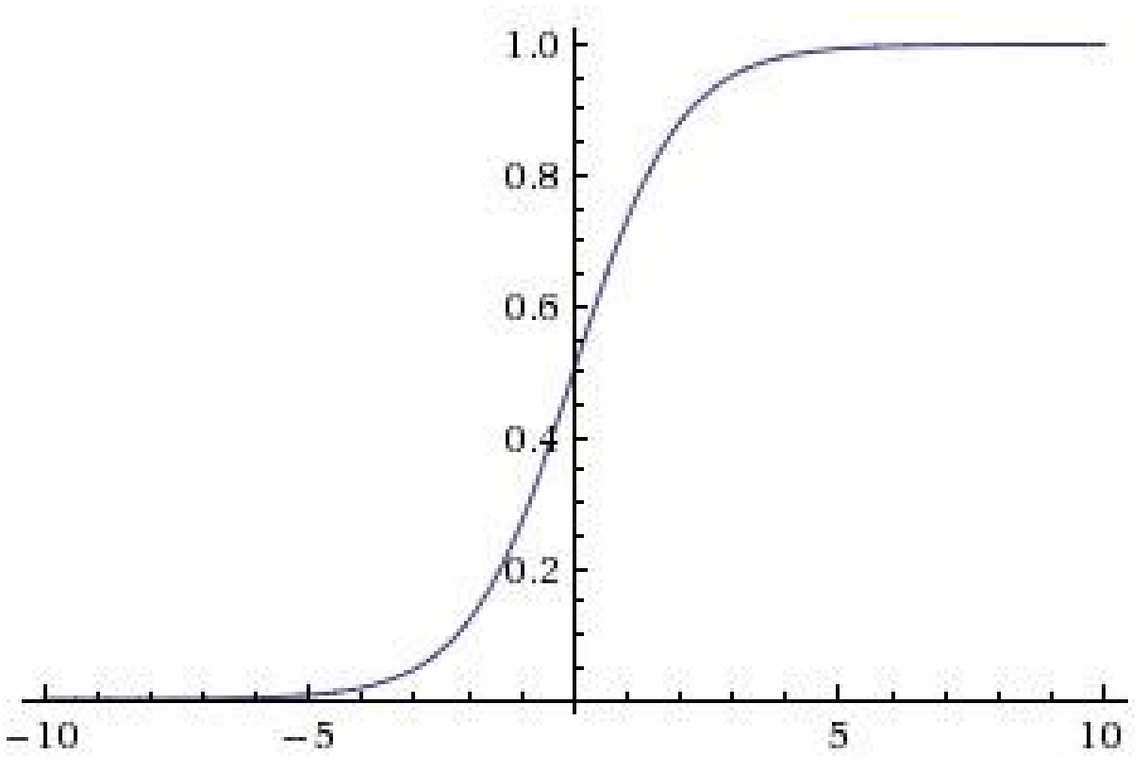
Non-linearity!

Figures courtesy of Stanford course of CS231n.

Activation functions (con't)

Learning And Mining from DatA

$$\sigma(x) = 1/(1+e^{-x})$$



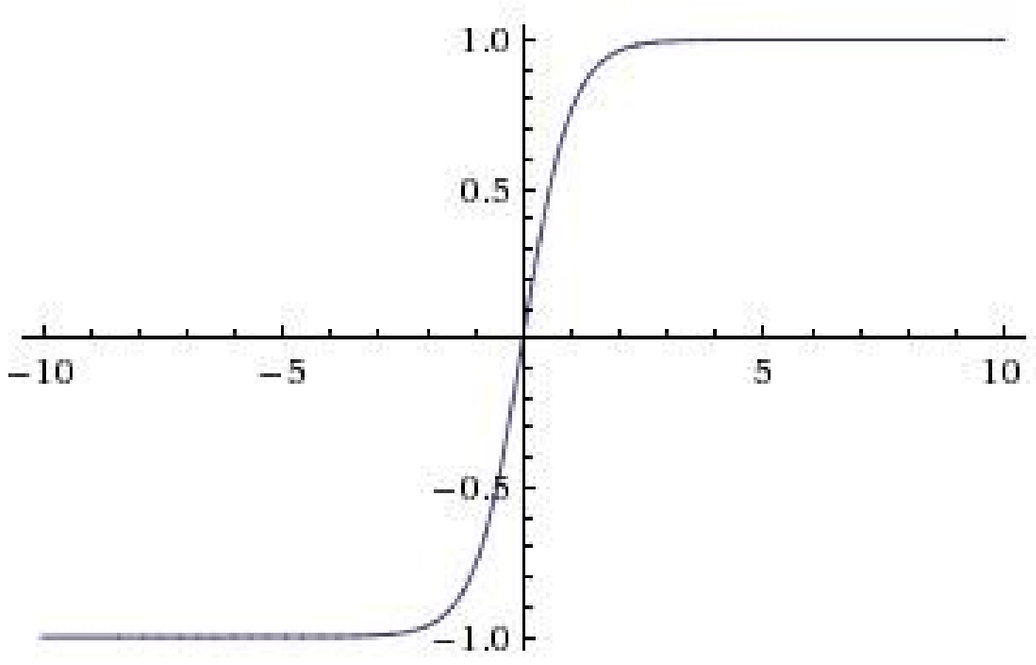
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

2 BIG problems:

- Saturated neurons "kill" the gradients
- Sigmoid outputs are not zero-centered

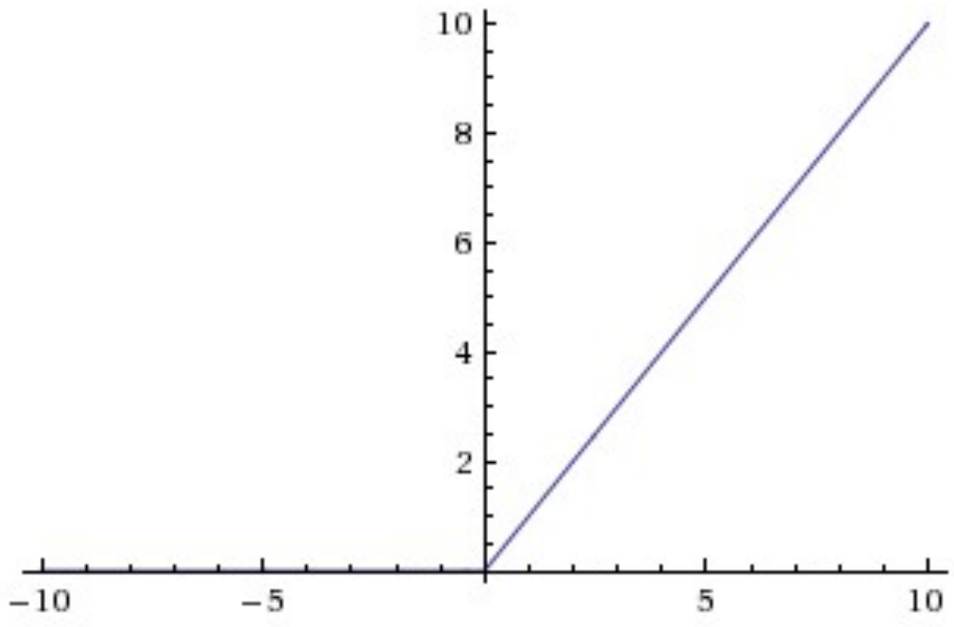
Activation functions (con't)



tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

Activation functions (con't)

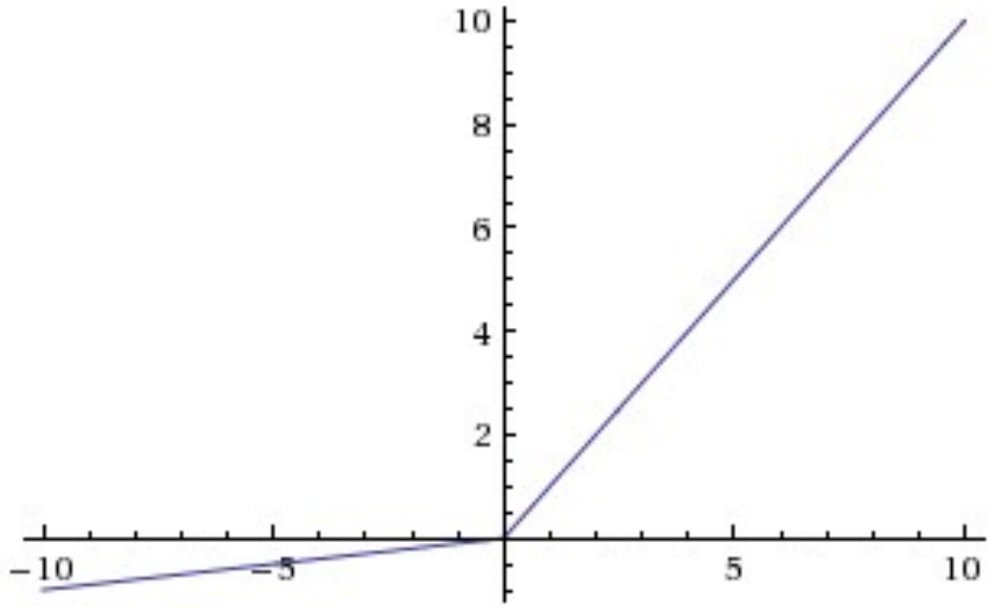


ReLU
(Rectified Linear Unit)

- Does not saturate
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g., 6x)

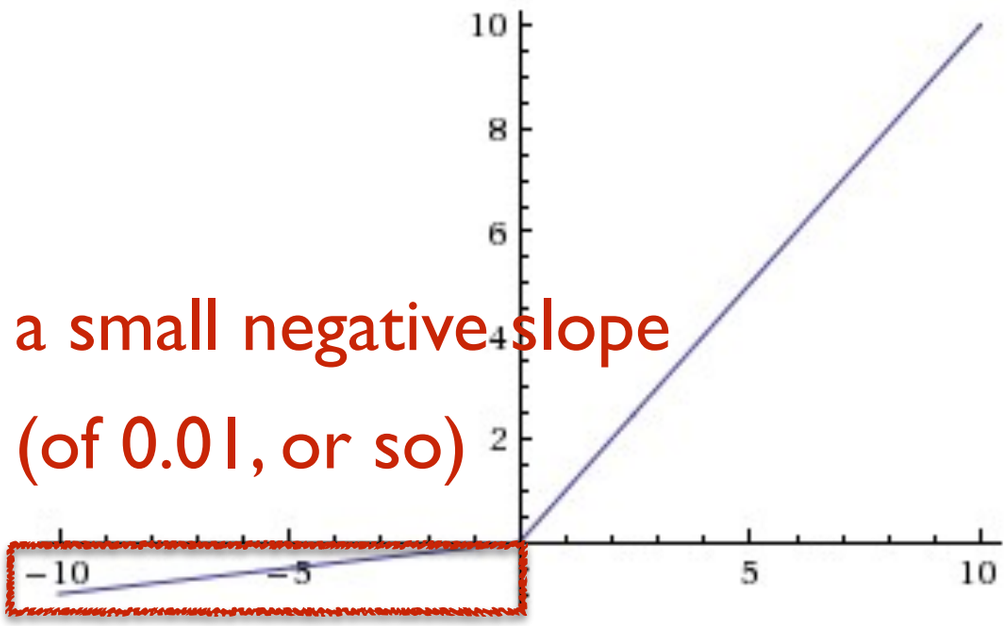
Just one annoying problem …
what is the gradient when x<0?

Activation functions (con't)



Leaky ReLU

Activation functions (con't)

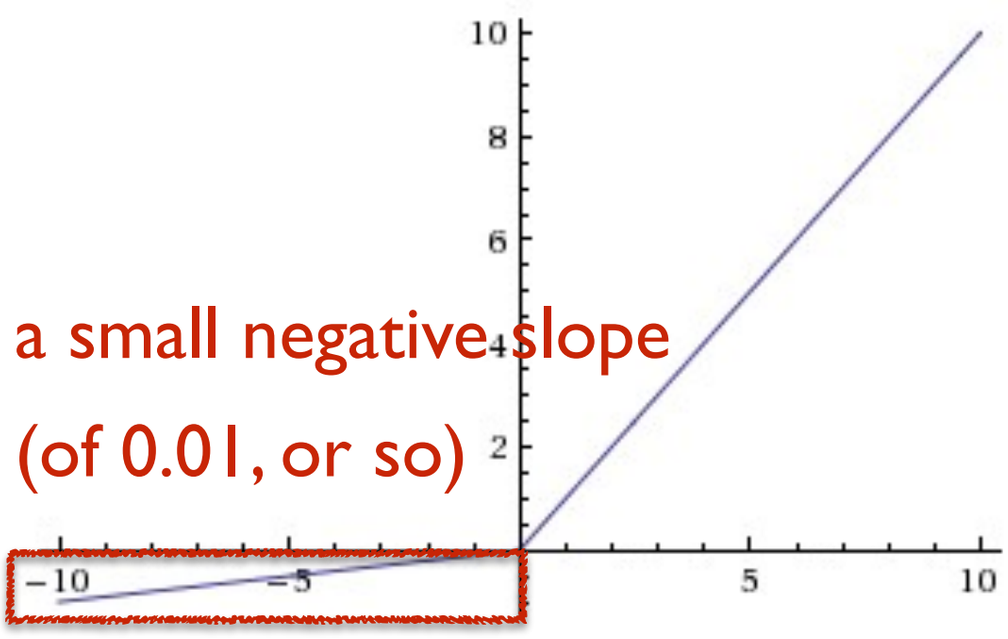


a small negative slope

(of 0.01, or so)

Leaky ReLU

Activation functions (con't)



Leaky ReLU

- Does not saturate
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g., 6x)
- will not "die"

Activation functions (con't)

Learning And Mining from DatA

Maxout "Neuron" (born in Jan. 2013)

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Problem: doubles the number of parameters :(

Practical suggestions on AF

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout
- Try out tanh but don't expect much
- Never use sigmoid

— Advised by F.-F. Li and A. Karpathy

Some other trials

Learning And Mining from DatA

Variants of ReLU:

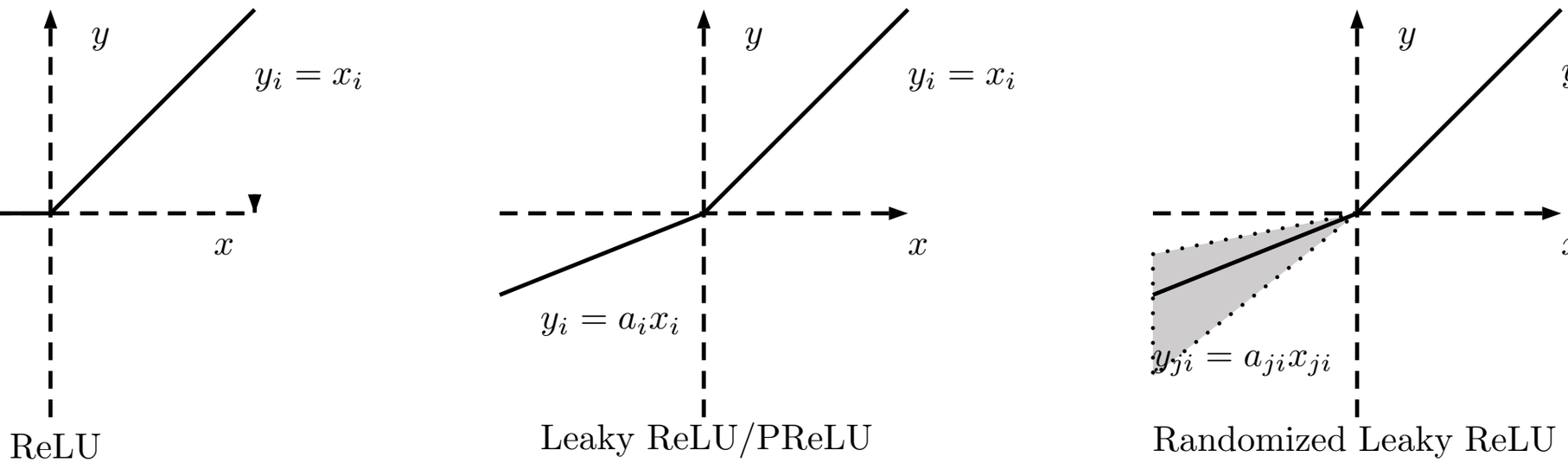


Figure 1: ReLU, Leaky ReLU, PReLU and RReLU. For PReLU, a_i is learned and for Leaky ReLU a_i is fixed. For RReLU, a_{ji} is a random variable keeps sampling in a given range, and remains fixed in testing.

[B. Xu, N. Wang, T. Chen and M. Li, arXiv: 1505.00853v1]

http://lamda.nju.edu.cn/weixs/

PReLU

Learning And Mining from DatA

Parametric ReLU:

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}. \qquad (1)$$

a_i is learned for the i-th channel

[K. He, et al., ICCV'15]

PReLU

LAMDA
Learning And Mining from DatA

Parametric ReLU:

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}. \qquad (1)$$

a_i is learned for the i-th channel

BP on PReLU:

$$\frac{\partial \mathcal{E}}{\partial a_i} = \sum_{y_i} \frac{\partial \mathcal{E}}{\partial f(y_i)} \frac{\partial f(y_i)}{\partial a_i}, \qquad (2)$$

[K. He, et al., ICCV'15]

PReLU

Learning And Mining from DatA

Parametric ReLU:

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}. \qquad (1)$$

a_i is learned for the i-th channel

BP on PReLU:

Objective function

$$\frac{\partial \mathcal{E}}{\partial a_i} = \sum_{y_i} \frac{\partial \mathcal{E}}{\partial f(y_i)} \frac{\partial f(y_i)}{\partial a_i}, \qquad (2)$$

[K. He, et al., ICCV'15]

PReLU

Learning And Mining from DatA

Parametric ReLU:

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}. \qquad (1)$$

a_i is learned for the i-th channel

BP on PReLU:

Objective function

$$\frac{\partial \mathcal{E}}{\partial a_i} = \sum_{y_i} \frac{\partial \mathcal{E}}{\partial f(y_i)} \frac{\partial f(y_i)}{\partial a_i}, \qquad (2)$$

Updating \alpha — the momentum method:

$$\Delta a_i := \mu \Delta a_i + \epsilon \frac{\partial \mathcal{E}}{\partial a_i}.$$

[K. He, et al., ICCV'15]

RReLU

Learning And Mining from DatA

Randomized ReLU:

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}. \qquad (1)$$

$$a_i \sim 1/U(l, u).$$

Test stage: only use the average "a" to get prediction results.

$$a_i = 2/(l+u) = \frac{1}{(l+u)/2}$$

Performances of ReLUs

Learning And Mining from DatA

Datasets: CIFAR-10, CIFAR-100 and NDSB datasets.

Settings: 1) 50k for training and 10k for test;

2) images without any pre-processing and augmentation.

Two CNN models:

Input Size	NIN
32×32	5x5, 192
32×32	1x1, 160
32×32	1x1, 96
32×32	3x3 max pooling, /2
16×16	dropout, 0.5
16×16	5x5, 192
16×16	1x1, 192
16×16	1x1, 192
16×16	3x3,avg pooling, /2
8×8	dropout, 0.5
8×8	3x3, 192
8×8	1x1, 192
8×8	1x1, 10
8×8	8x8, avg pooling, /1
10 or 100	softmax

Input Size	NDSB Net
70×70	3x3, 32
70×70	3x3, 32
70×70	3x3, max pooling, /2
35×35	3x3, 64
35×35	3x3, 64
35×35	3x3, 64
35×35	3x3, max pooling, /2
17×17	split: branch1 — branch 2
17×17	3x3, 96 — 3x3, 96
17×17	3x3, 96 — 3x3, 96
17×17	3x3, 96 — 3x3, 96
17×17	3x3, 96
17×17	channel concat, 192
17×17	3x3, max pooling, /2
8×8	3x3, 256
8×8	3x3, 256
8×8	3x3, 256
8×8	3x3, 256
8×8	3x3, 256
8×8	SPP (He et al., 2014) {1, 2, 4}
12544×1	flatten
1024×1	fc1
1024×1	fc2
121	softmax

Performances of ReLUs (con't)

Activation	Training Error	Test Error
ReLU	0.00318	0.1245
Leaky ReLU, $a = 100$	0.0031	0.1266
Leaky ReLU, $a = 5.5$	0.00362	**0.1120**
PReLU	0.00178	0.1179
RReLU	0.00550	**0.1119**

Table 3. Error rate of CIFAR-10 Network in Network with different activation function

Activation	Training Error	Test Error
ReLU	0.1356	0.429
Leaky ReLU, $a = 100$	0.11552	0.4205
Leaky ReLU, $a = 5.5$	0.08536	**0.4042**
PReLU	0.0633	0.4163
RReLU	0.1141	**0.4025**

Table 4. Error rate of CIFAR-100 Network in Network with different activation function

Activation	Train Log-Loss	Val Log-Loss
ReLU	0.8092	0.7727
Leaky ReLU, $a = 100$	0.7846	0.7601
Leaky ReLU, $a = 5.5$	0.7831	0.7391
PReLU	0.7187	0.7454
RReLU	0.8090	**0.7292**

Table 5. Multi-classes Log-Loss of NDSB Network with different activation function

Performances of ReLUs (con't)

LAMDA
Learning And Mining from DatA

Activation	Training Error	Test Error
ReLU	0.00318	0.1245
Leaky ReLU, $a = 100$	0.0031	0.1266
Leaky ReLU, $a = 5.5$	0.00362	**0.1120**
PReLU	0.00178	0.1179
RReLU	0.00550	**0.1119**

Table 3. Error rate of CIFAR-10 Network in Network with different activation function

Activation	Training Error	Test Error
ReLU	0.1356	0.429
Leaky ReLU, $a = 100$	0.11552	0.4205
Leaky ReLU, $a = 5.5$	0.08536	**0.4042**
PReLU	0.0633	0.4163
RReLU	0.1141	**0.4025**

Table 4. Error rate of CIFAR-100 Network in Network with different activation function

Activation	Train Log-Loss	Val Log-Loss
ReLU	0.8092	0.7727
Leaky ReLU, $a = 100$	0.7846	0.7601
Leaky ReLU, $a = 5.5$	0.7831	0.7391
PReLU	0.7187	0.7454
RReLU	0.8090	**0.7292**

Table 5. Multi-classes Log-Loss of NDSB Network with different activation function

Results of He's paper

LAMDA
Learning And Mining from DatA

CNN model (small):

layer		learned coefficients	
		channel-shared	channel-wise
conv1	7×7, 64, /2	0.681	0.596
pool1	3×3, /3		
$conv2_1$	2×2, 128	0.103	0.321
$conv2_2$	2×2, 128	0.099	0.204
$conv2_3$	2×2, 128	0.228	0.294
$conv2_4$	2×2, 128	0.561	0.464
pool2	2×2, /2		
$conv3_1$	2×2, 256	0.126	0.196
$conv3_2$	2×2, 256	0.089	0.152
$conv3_3$	2×2, 256	0.124	0.145
$conv3_4$	2×2, 256	0.062	0.124
$conv3_5$	2×2, 256	0.008	0.134
$conv3_6$	2×2, 256	0.210	0.198
spp	{6, 3, 2, 1}		
fc_1	4096	0.063	0.074
fc_2	4096	0.031	0.075
fc_3	1000		

Performance:

Small CNN:

	top-1	top-5
ReLU	33.82	13.34
PReLU, channel-shared	32.71	12.87
PReLU, channel-wise	**32.64**	**12.75**

Table 2. Comparisons between ReLU and PReLU on the small model. The error rates are for ImageNet 2012 using 10-view testing. The images are resized so that the shorter side is 256, during both training and testing. Each view is 224×224. All models are trained using 75 epochs.

Large CNN:

model A	ReLU		PReLU	
scale s	top-1	top-5	top-1	top-5
256	26.25	8.25	**25.81**	**8.08**
384	24.77	7.26	**24.20**	**7.03**
480	25.46	7.63	**24.83**	**7.39**
multi-scale	24.02	6.51	**22.97**	**6.28**

Table 4. Comparisons between ReLU/PReLU on model A in ImageNet 2012 using dense testing.

[K. He, et al., ICCV'15]

Results of He's paper

LAMDA
Learning And Mining from DatA

CNN model (small):

layer		learned coefficients	
		channel-shared	channel-wise
conv1	7×7, 64, $/2$	0.681	0.596
pool1	3×3, $/3$		
conv2_1	2×2, 128	0.103	0.321
conv2_2	2×2, 128	0.099	0.204
conv2_3	2×2, 128	0.228	0.294
conv2_4	2×2, 128	0.561	0.464
pool2	2×2, $/2$		
conv3_1	2×2, 256	0.126	0.196
conv3_2	2×2, 256	0.089	0.152
conv3_3	2×2, 256	0.124	0.145
conv3_4	2×2, 256	0.062	0.124
conv3_5	2×2, 256	0.008	0.134
conv3_6	2×2, 256	0.210	0.198
spp	$\{6, 3, 2, 1\}$		
fc_1	4096	0.063	0.074
fc_2	4096	0.031	0.075
fc_3	1000		

Performance:

Small CNN:

	top-1	top-5
ReLU	33.82	13.34
PReLU, channel-shared	32.71	12.87
PReLU, channel-wise	**32.64**	**12.75**

Table 2. Comparisons between ReLU and PReLU on the small model. The error rates are for ImageNet 2012 using 10-view testing. The images are resized so that the shorter side is 256, during both training and testing. Each view is 224×224. All models are trained using 75 epochs.

Large CNN:

model A	ReLU		PReLU	
scale s	top-1	top-5	top-1	top-5
256	26.25	8.25	**25.81**	**8.08**
384	24.77	7.26	**24.20**	**7.03**
480	25.46	7.63	**24.83**	**7.39**
multi-scale	24.02	6.51	**22.97**	**6.28**

Table 4. Comparisons between ReLU/PReLU on model A in ImageNet 2012 using dense testing.

[K. He, et al., ICCV'15]

Results of He's paper

LAMDA
Learning And Mining from DatA

CNN model (small):

layer		learned coefficients: channel-shared	learned coefficients: channel-wise
conv1	7×7, 64, /2	0.681	0.596
pool1	3×3, /3		
conv2$_1$	2×2, 128	0.103	0.321
conv2$_2$	2×2, 128	0.099	0.204
conv2$_3$	2×2, 128	0.228	0.294
conv2$_4$	2×2, 128	0.561	0.464
pool2	2×2, /2		
conv3$_1$	2×2, 256	0.126	0.196
conv3$_2$	2×2, 256	0.089	0.152
conv3$_3$	2×2, 256	0.124	0.145
conv3$_4$	2×2, 256	0.062	0.124
conv3$_5$	2×2, 256	0.008	0.134
conv3$_6$	2×2, 256	0.210	0.198
spp	{6, 3, 2, 1}		
fc$_1$	4096	0.063	0.074
fc$_2$	4096	0.031	0.075
fc$_3$	1000		

Performance:

Small CNN:

	top-1	top-5
ReLU	33.82	13.34
PReLU, channel-shared	32.71	12.87
PReLU, channel-wise	**32.64**	**12.75**

Table 2. Comparisons between ReLU and PReLU on the small model. The error rates are for ImageNet 2012 using 10-view testing. The images are resized so that the shorter side is 256, during both training and testing. Each view is 224×224. All models are trained using 75 epochs.

Large CNN:

model A	ReLU		PReLU	
scale s	top-1	top-5	top-1	top-5
256	26.25	8.25	**25.81**	**8.08**
384	24.77	7.26	**24.20**	**7.03**
480	25.46	7.63	**24.83**	**7.39**
multi-scale	24.02	6.51	**22.97**	**6.28**

Table 4. Comparisons between ReLU/PReLU on model A in ImageNet 2012 using dense testing.

[K. He, et al., ICCV'15]

Regularization

- L2 regularization
- L1 regularization
- L1 + L2 can also be combined
- Max norm constraints

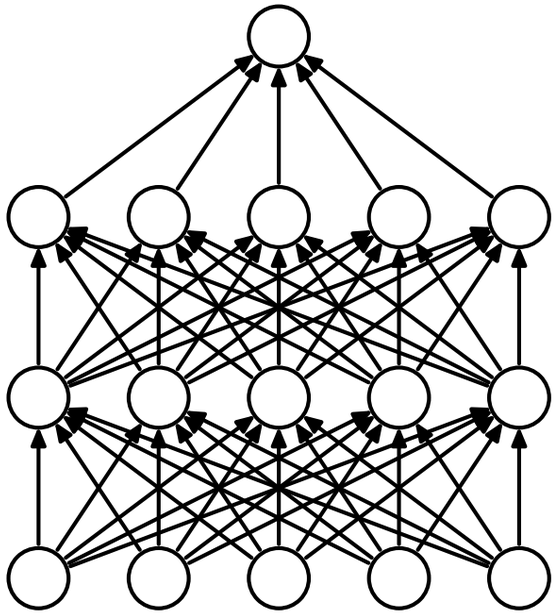
$\frac{1}{2}\lambda\omega^2$ $\lambda|\omega|$ $||\omega||_2 < c$

[N. Srivastava et al., JMLR'14]

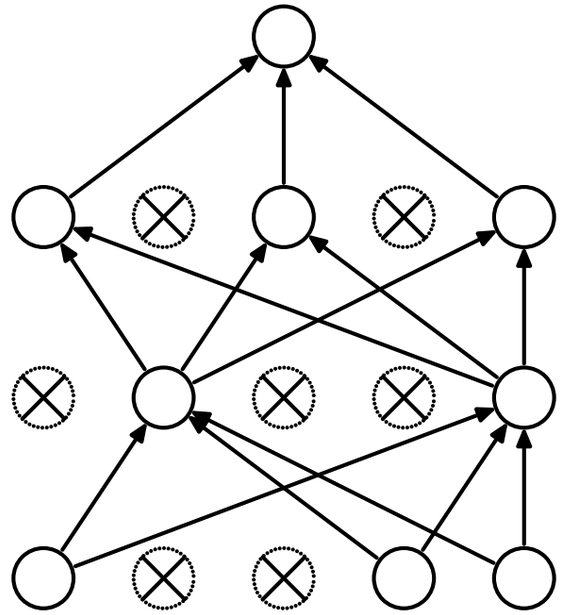
Regularization

- L2 regularization
- L1 regularization
- L1 + L2 can also be combined
- Max norm constraints
- Dropout

$\frac{1}{2}\lambda\omega^2$ $\lambda|\omega|$ $||\omega||_2 < c$



(a) Standard Neural Net



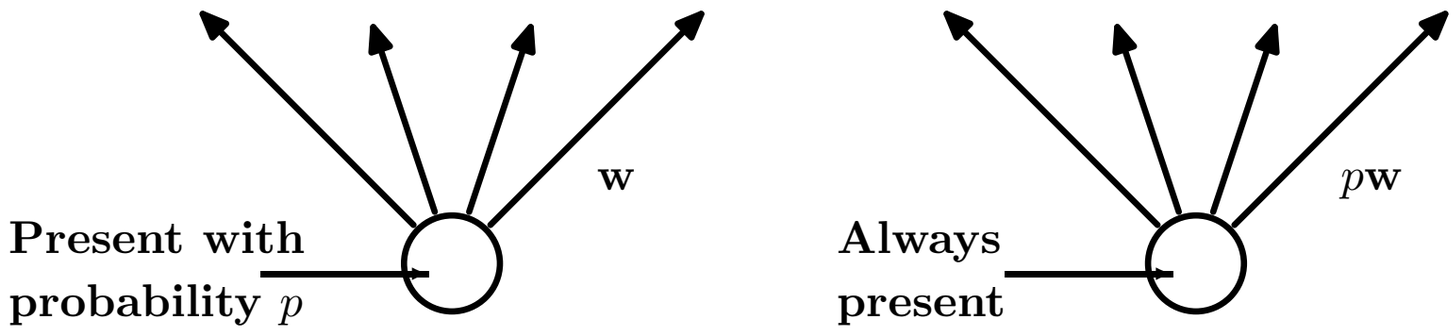
(b) After applying dropout.

[N. Srivastava et al., JMLR'14]

Regularization (con't)

Learning And Mining from DatA

NOTICE!



(a) At training time (b) At test time

Performance comparisons:

Method	**Test Classification error %**
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

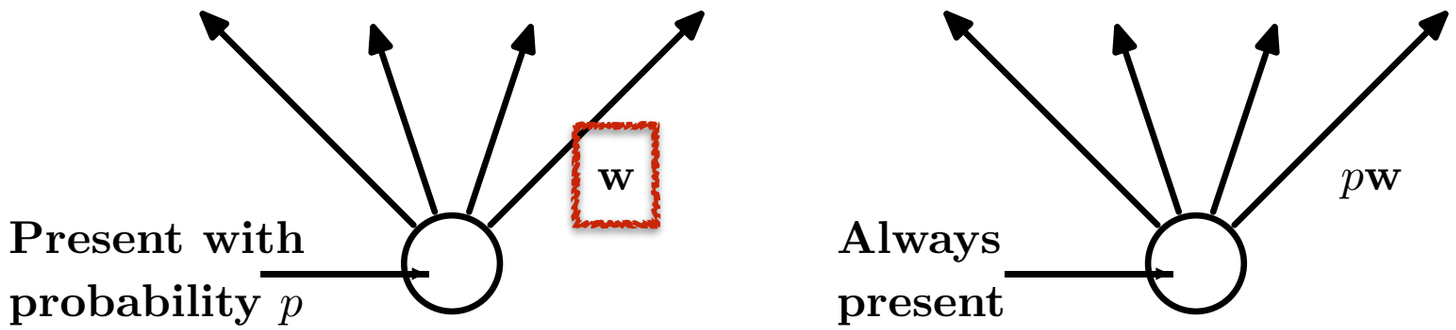
Table 9: Comparison of different regularization methods on MNIST.

[N. Srivastava et al., JMLR'14]

Regularization (con't)

Learning And Mining from DatA

NOTICE!



(a) At training time (b) At test time

Performance comparisons:

Method	**Test Classification error %**
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

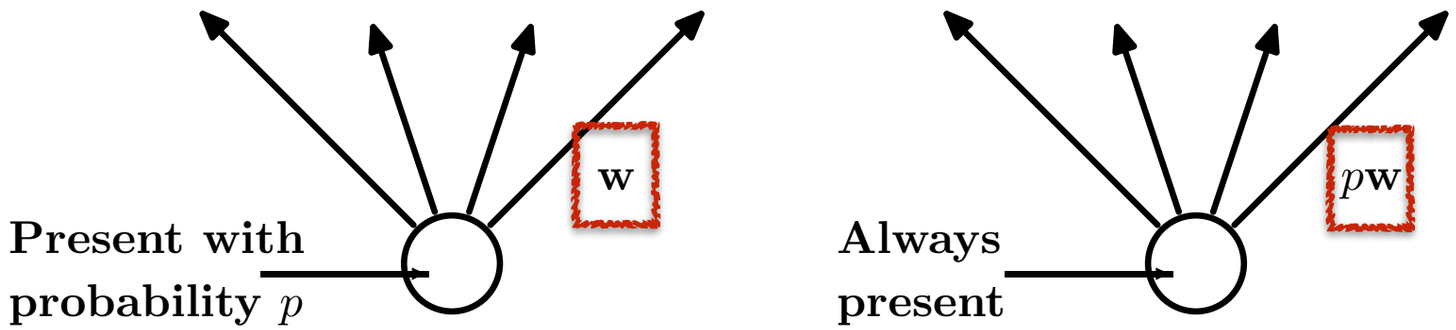
Table 9: Comparison of different regularization methods on MNIST.

[N. Srivastava et al., JMLR'14]

Regularization (con't)

Learning And Mining from DatA

NOTICE!



(a) At training time (b) At test time

Performance comparisons:

Method	**Test Classification error %**
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

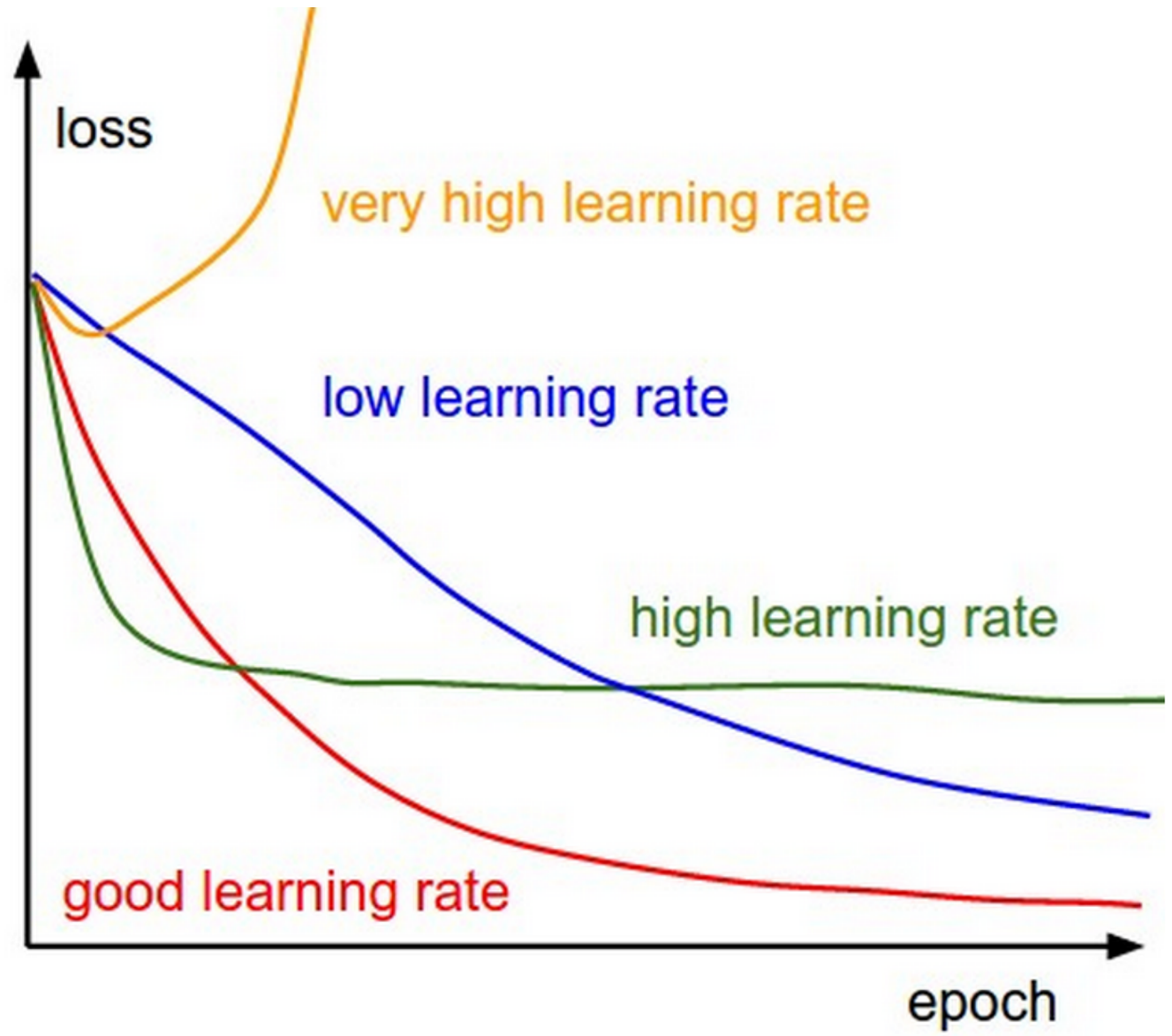
Table 9: Comparison of different regularization methods on MNIST.

[N. Srivastava et al., JMLR'14]

Insights from figures

LAMDA
Learning And Mining from DatA

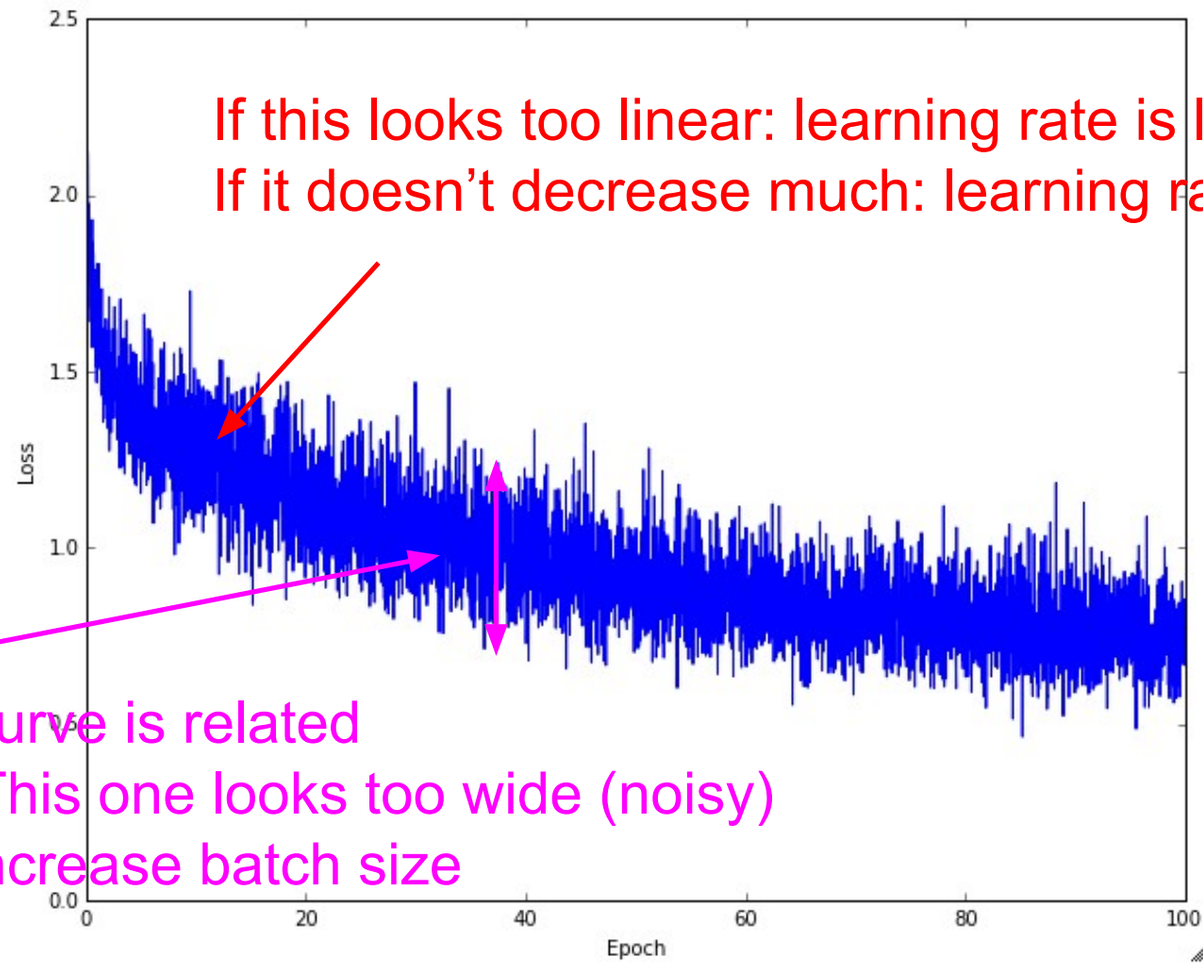
The learning rate:



Figures courtesy of Stanford course CS231n.

Insights from figures

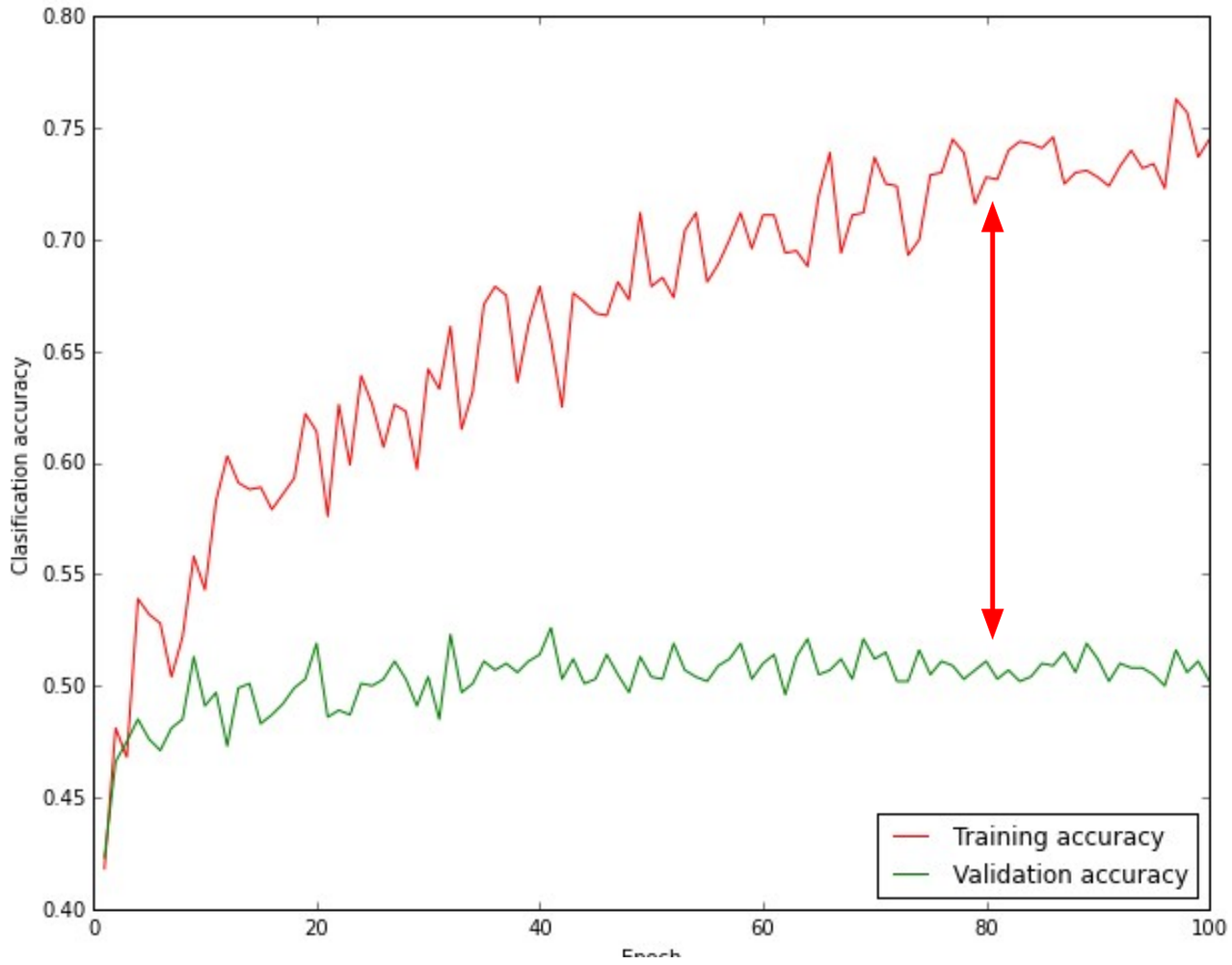
The loss curve:



Insights from figures (con't)

Learning And Mining from DatA

The accuracy:



big gap = overfitting

=> increase regularization strength

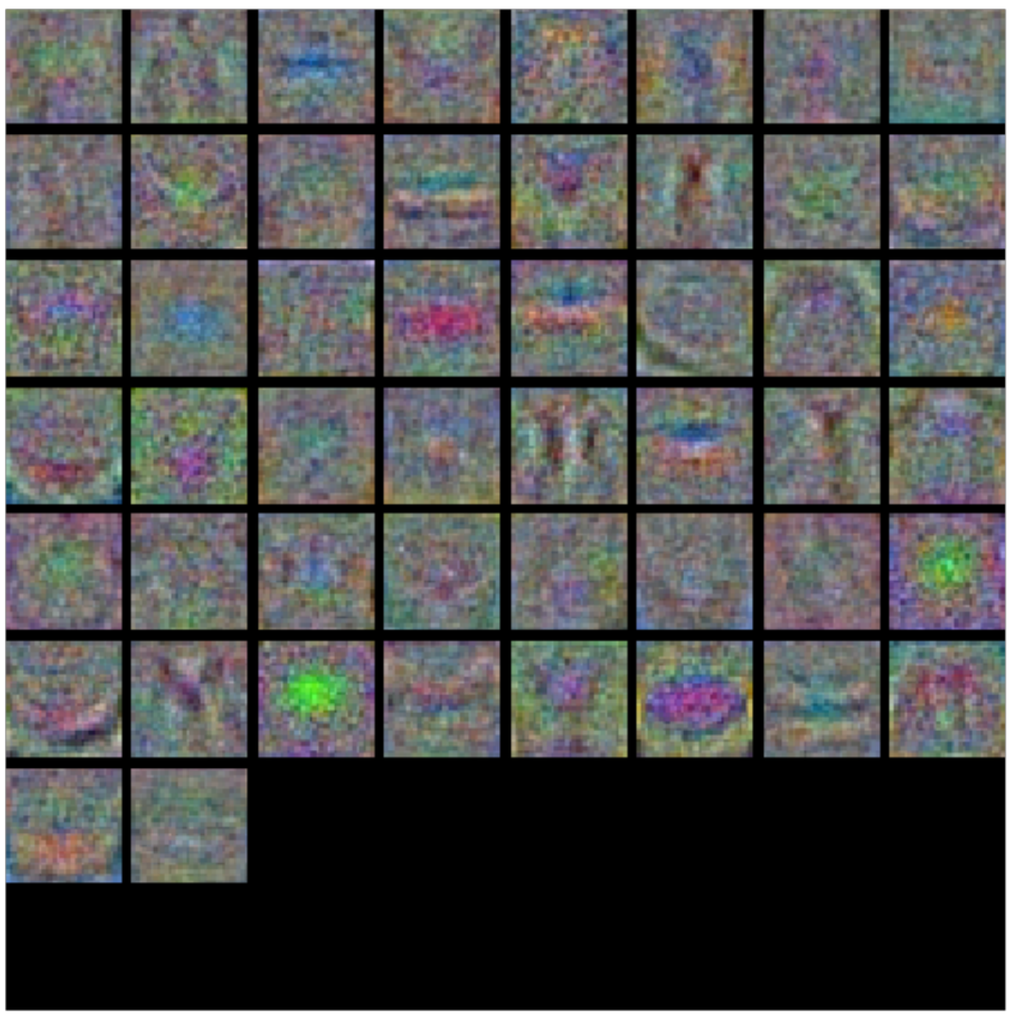
no gap

=> increase model capacity

Insights from figures (con't)

Learning And Mining from DatA

Visualizing first-layer weights:

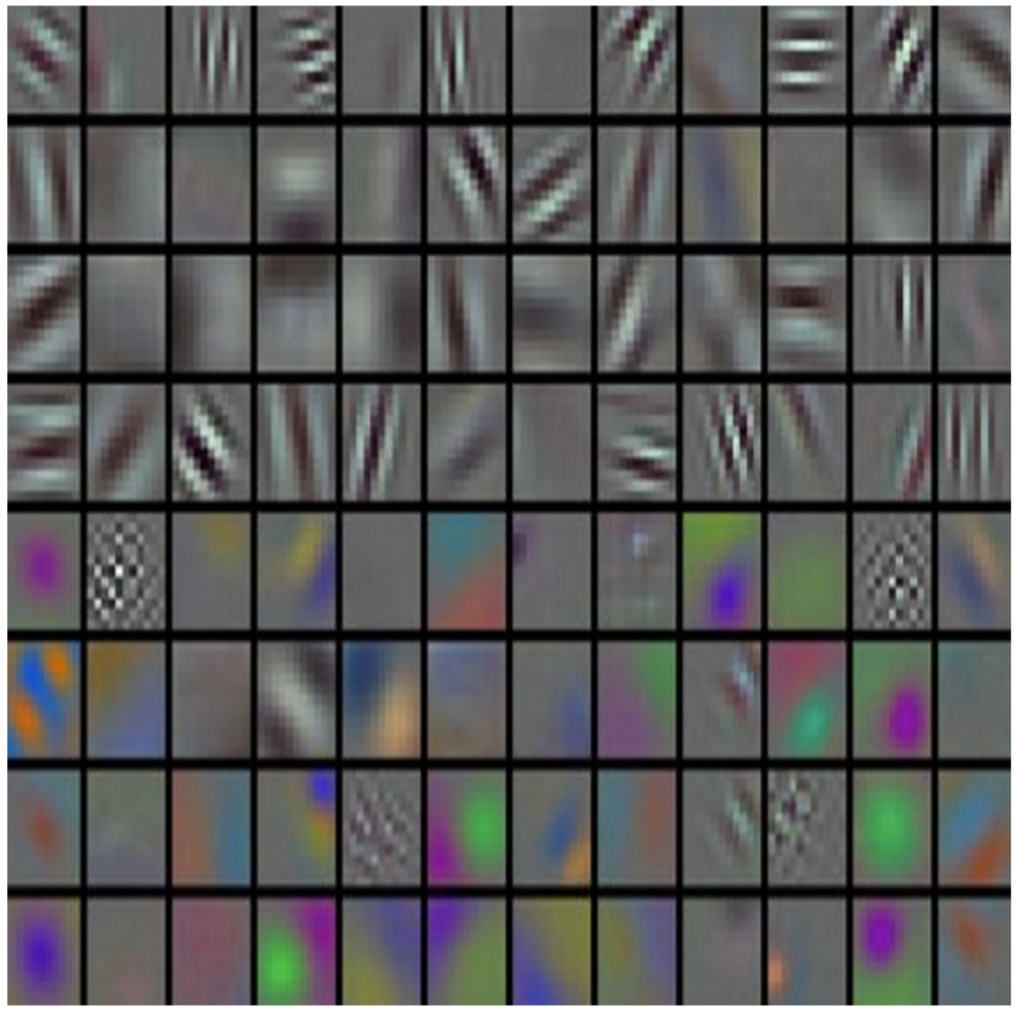


Noisy weights =>
Regularization maybe not strong enough

Insights from figures (con't)

Learning And Mining from DatA

Visualizing first-layer weights:



Noisy weights =>
Regularization maybe not strong enough

Ensemble

Learning And Mining from DatA

- **Same model, different initializations**. Use cross-validation to determine the best hyperparameters, then train multiple models with the best set of hyperparameters but with different random initialization. The danger with this approach is that the variety is only due to initialization.

- **Top models discovered during cross-validation**. Use cross-validation to determine the best hyperparameters, then pick the top few (e.g. 10) models to form the ensemble. This improves the variety of the ensemble but has the danger of including suboptimal models. In practice, this can be easier to perform since it doesn't require additional retraining of models after cross-validation

- **Different checkpoints of a single model**. If training is very expensive, some people have had limited success in taking different checkpoints of a single network over time (for example after every epoch) and using those to form an ensemble. Clearly, this suffers from some lack of variety, but can still work reasonably well in practice. The advantage of this approach is that is very cheap.

References

- F.-F. Li and A. Karpathy, "CS231n: Convolutional Neural Networks for Visual Recognition," http://cs231n.stanford.edu/index.html, 2015.

- K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," ICCV, 2015.

- B. Xu, N. Wang, T. Chen and M. Li, "Empirical Evaluation of Rectified Activations in Convolution Network," arXiv: 1505.00853v1, 2015.

- I. Sutskever, "A Brief Overview of Deep Learning," http://yyue.blogspot.com/2015/01/a-brief-overview-of-deep-learning.html, 2015.

- A. Krizhevsky, I. Sutskever and G. E. Hiton, "ImageNet Classification with Deep Convolutional Neural Networks," NIPS, 2012.

- N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," JMLR, 2014.

LAMDA
Learning And Mining from DatA

The harder I work, the luckier I get.