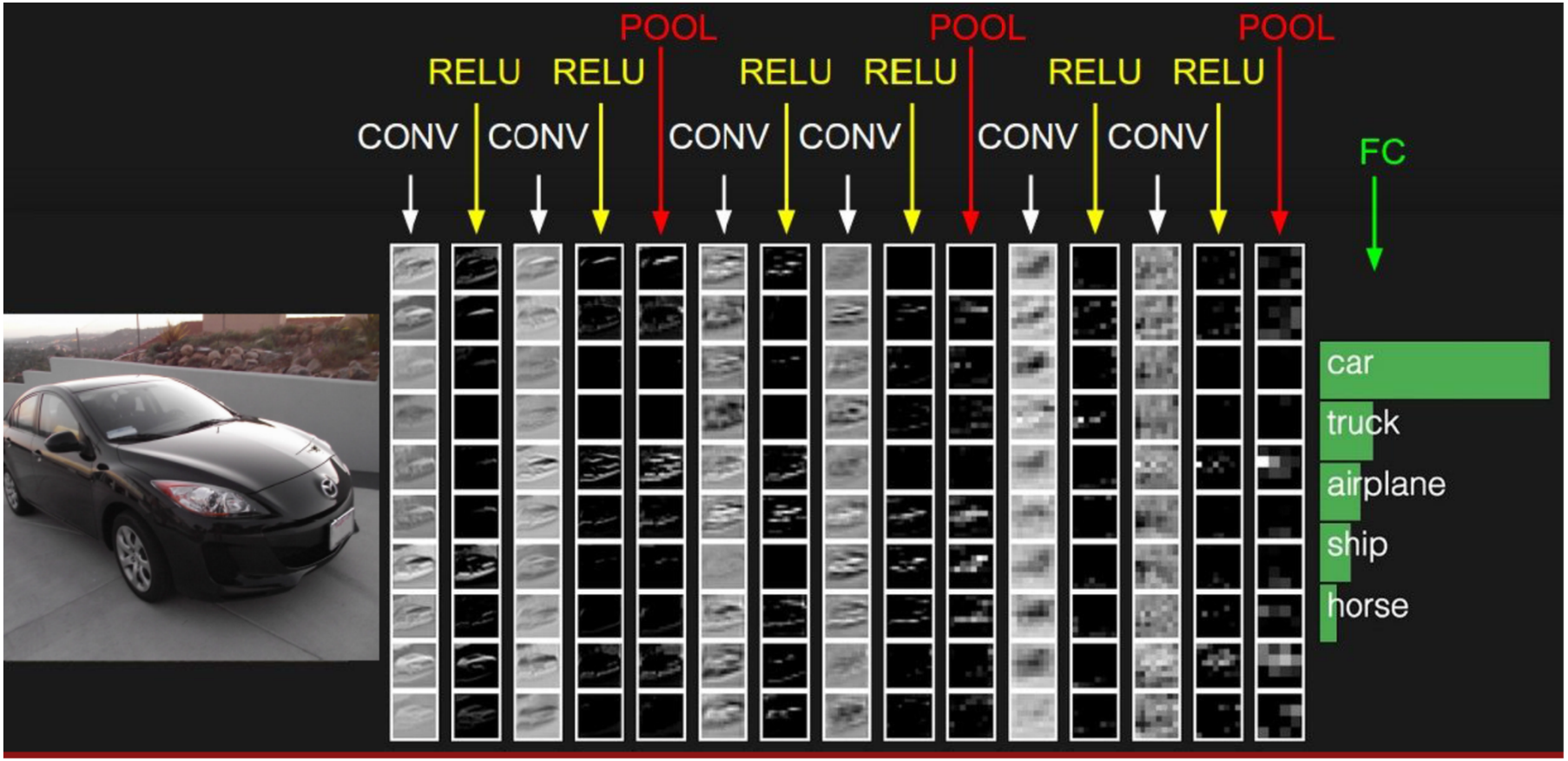
14.332.435/16.332.530
Introduction to Deep Learning

Lecture 8
Training Neural Networks 1

Yuqian Zhang

Department of Electrical and Computer Engineering

Recall Last Lecture



Today's Agenda

How to train a good DNN model

- Before the training
  -- Data preprocessing, Weight initialization
- During the training
  -- Batch normalization, Dropout
- After the training
  -- Model Ensemble

What is Data Preprocessing

Preprocessing

-- The *transformations* on the raw data before it is fed to DNNs.

-- Can improve performance (e.g. classification accuracy)

-- Can speedup training

Improve Task Performance

TABLE I: TEST ACCURACIES ON THREE CONVOLUTIONAL ARCHITECHTURES WITH DIFFERENT PREPROCESSING TECHNIQUES

Preprocessing	Features	CNN-1(%)	CNN-2(%)	CNN-3(%)
None	16	51.26	43.54	50.12
	20	52.14	43.19	52.79
	40	55.16	46.55	55.17
	60	55.64	47.54	55.63
	80	53.43	46.64	55.94
	100	54.47	49.06	55.84
Mean Normalization	16	55.01	57.72	61.76
	20	55.21	57.57	63.25
	40	56.97	56.50	65.91
	60	57.23	56.21	63.38
	80	56.46	55.99	62.55
	100	57.76	55.24	64.24
Standardization	16	63.29	64.75	65.75
	20	64.27	64.87	66.53
	40	65.59	63.86	68.36
	60	64.84	66.37	66.50
	80	63.96	66.74	65.67
	100	64.80	64.92	66.67

Pal etc. RTEICT, 2016

Improve Task Performance

TABLE I: TEST ACCURACIES ON THREE CONVOLUTIONAL ARCHITECHTURES WITH DIFFERENT PREPROCESSING TECHNIQUES

Preprocessing	Features	CNN-1(%)	CNN-2(%)	CNN-3(%)
None	16	51.26	43.54	50.12
	20	52.14	43.19	52.79
	40	55.16	46.55	55.17
	60	55.64	47.54	55.63
	80	53.43	46.64	55.94
	100	54.47	49.06	55.84
Mean Normalization	16	55.01	57.72	61.76
	20	55.21	57.57	63.25
	40	56.97	56.50	65.91
	60	57.23	56.21	63.38
	80	56.46	55.99	62.55
	100	57.76	55.24	64.24
Standardization	16	63.29	64.75	65.75
	20	64.27	64.87	66.53
	40	65.59	63.86	68.36
	60	64.84	66.37	66.50
	80	63.96	66.74	65.67
	100	64.80	64.92	66.67

Pal etc. RTEICT, 2016

Recall Cons of Sigmoid – Non-zero Centered

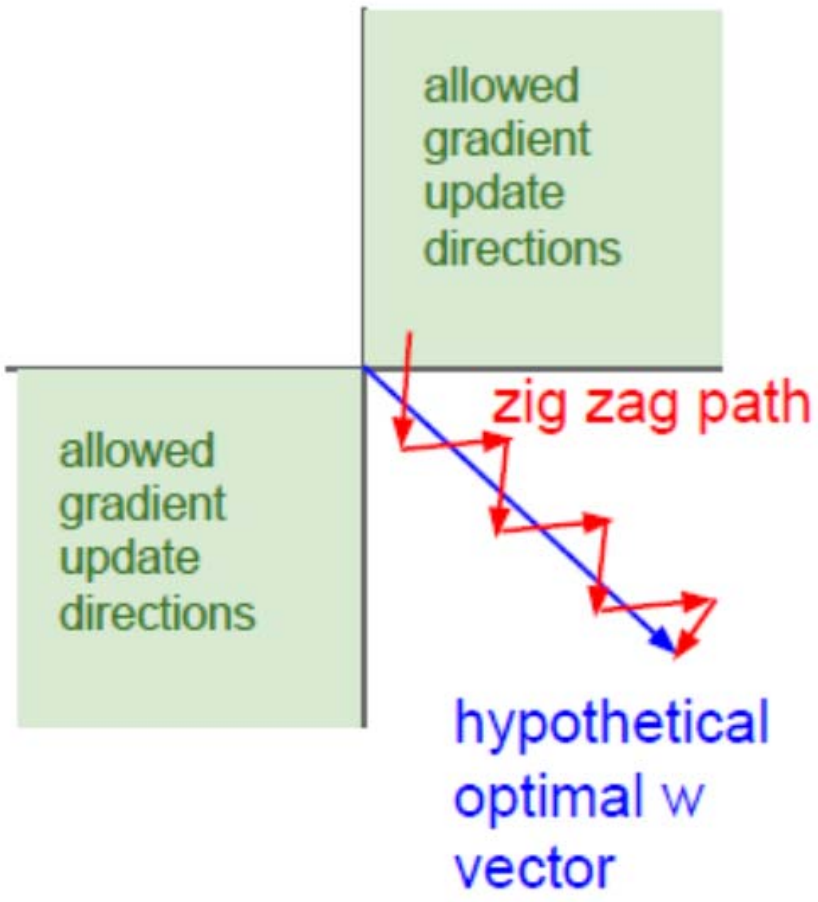
- Consider what happens when the input to a neuron is always positive

$$f\left(\sum_i w_i x_i + b\right)$$

Q: What will be the gradient on **w**?

Recall Cons of Sigmoid – Non-zero Centered

- Slow learning

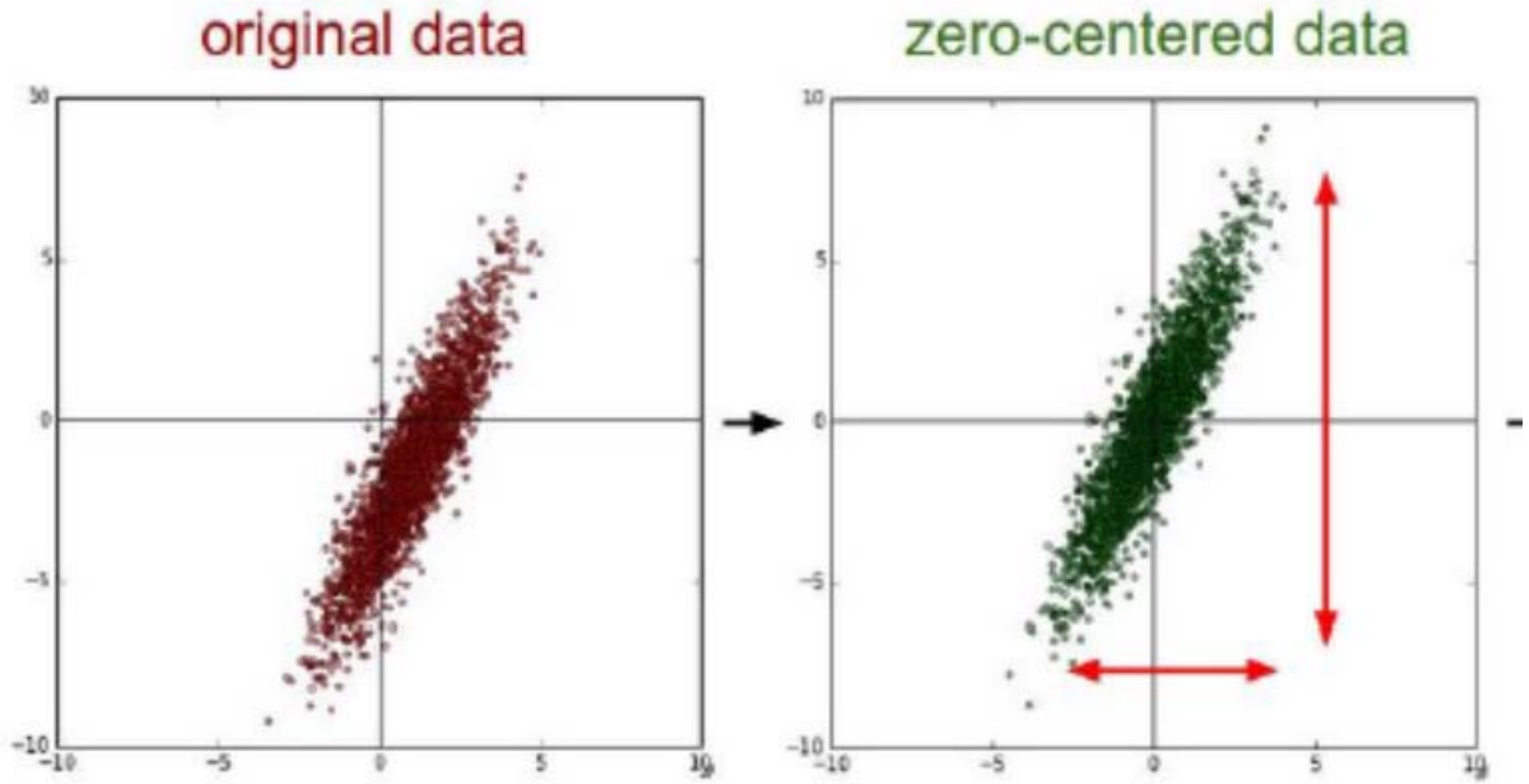


Mean Subtraction

- Mean subtraction makes zero-centered data

  -- Subtracting the mean across every individual feature in the data

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


original data
zero-centered data
X -= np.mean(X, axis = 0)

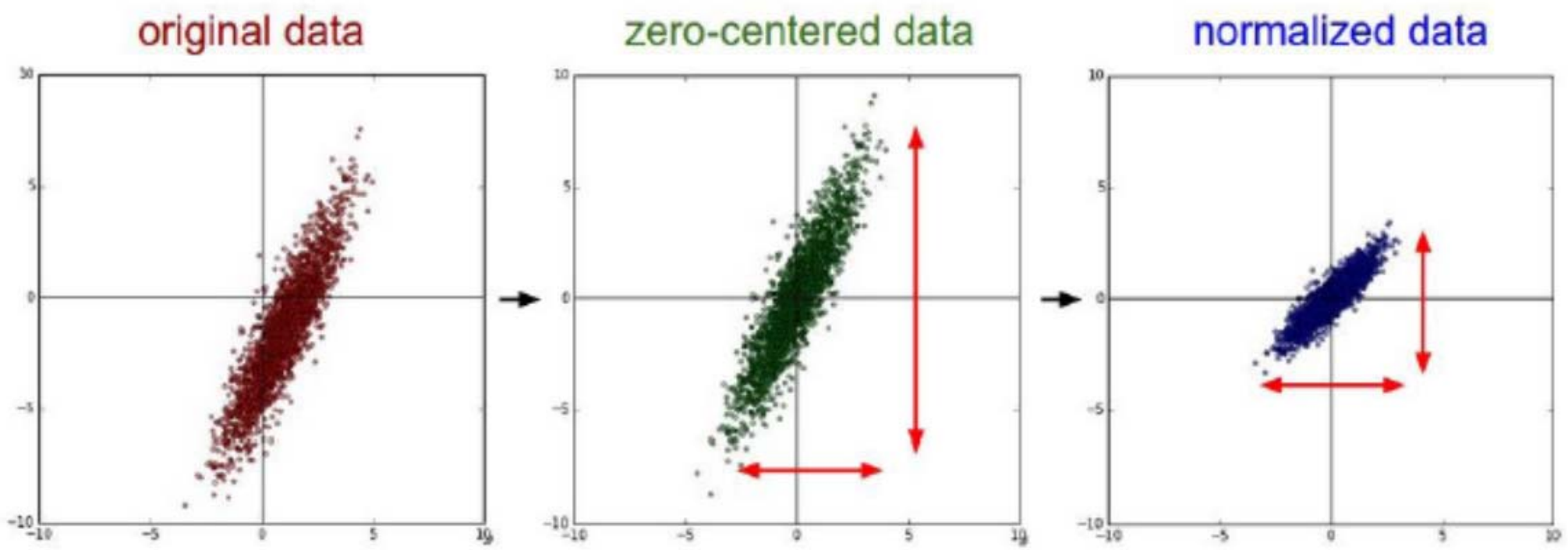
Normalization

- Normalization is to put all features on the same scale

- Approach A: To divide each dimension by its standard deviation, once it has been zero-centered:

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

- Approach B: To make the min and max along each dimension is -1 and 1 respectively.


original data
zero-centered data
normalized data
X -= np.mean(X, axis = 0)
X /= np.std(X, axis = 0)

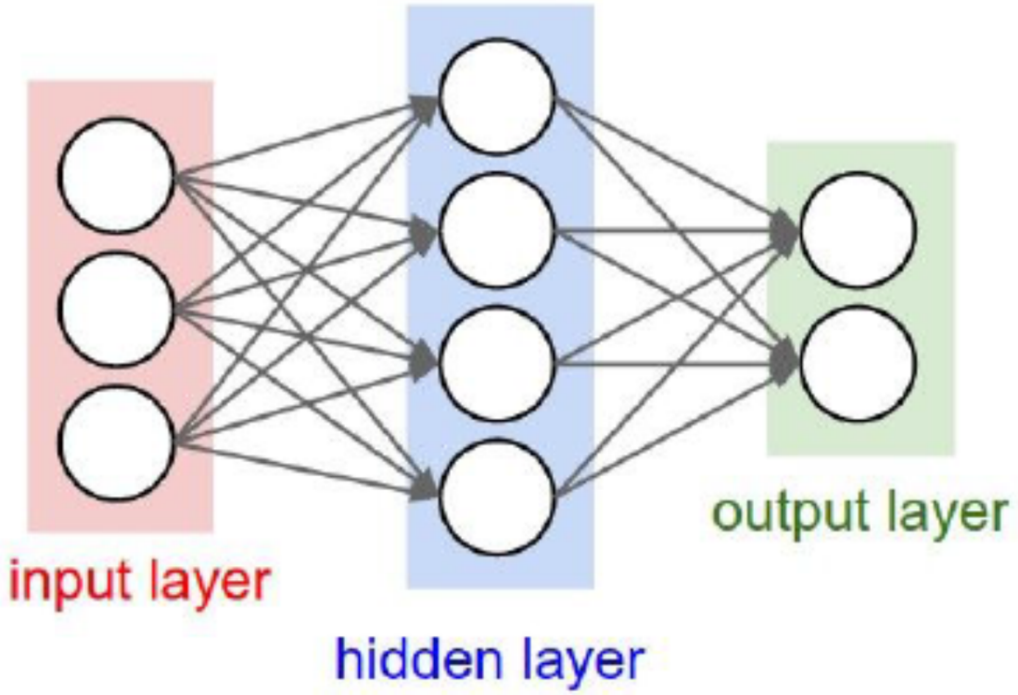
Note

- Normalization makes sense only when you believe that different-scale input features have approximately equal importance to the learning algorithm.
- Any preprocessing statistics (e.g. mean) must only be computed on the training data, and then applied to the validation / test data.
- It is wrong to compute mean and subtracting it from every image across the entire dataset.

-- Calculated from training dataset, subtracted equally from all splits (train/val/test).

Weight Initialization

Q: What happens if weights initialization are the constant



Asymmetric is needed for weight initialization

First Idea: Small Random Numbers

- To enable asymmetric weight initialization, we can randomly initialize weights.

  -- Gaussian with zero mean and certain standard deviation

```
W = 0.01* np.random.randn(D,H)
```

- Problem: Only works for small-size network

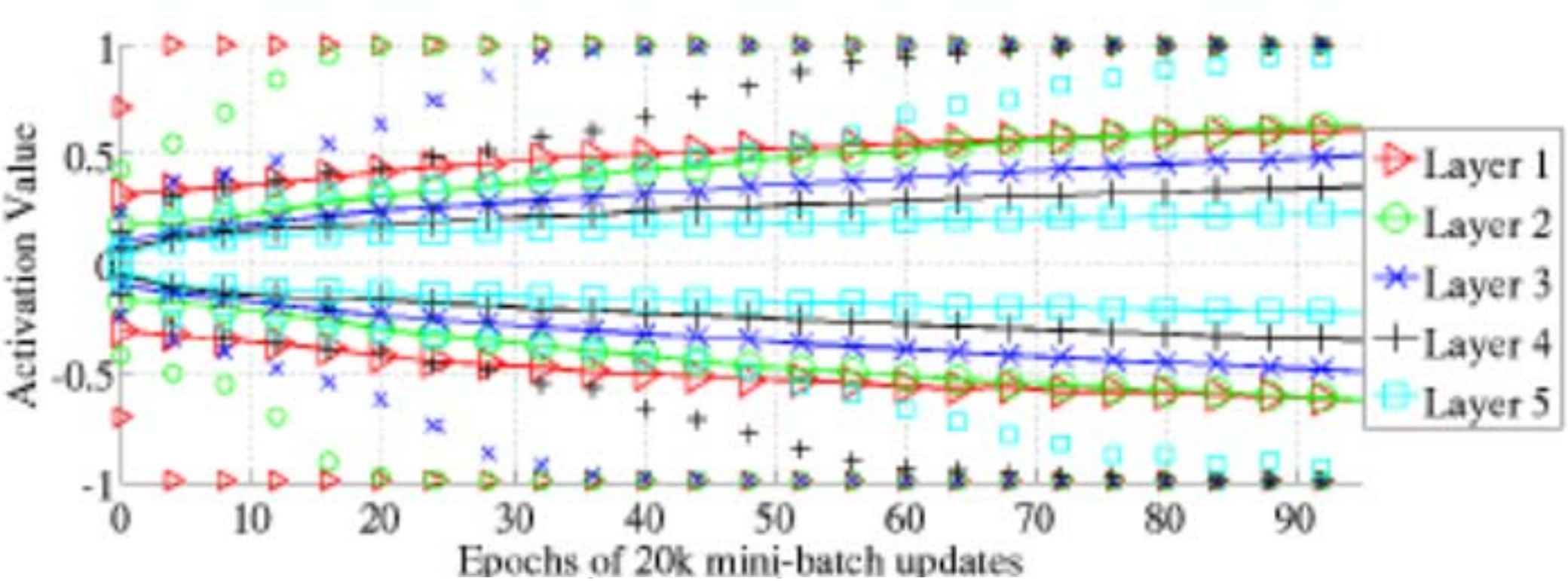
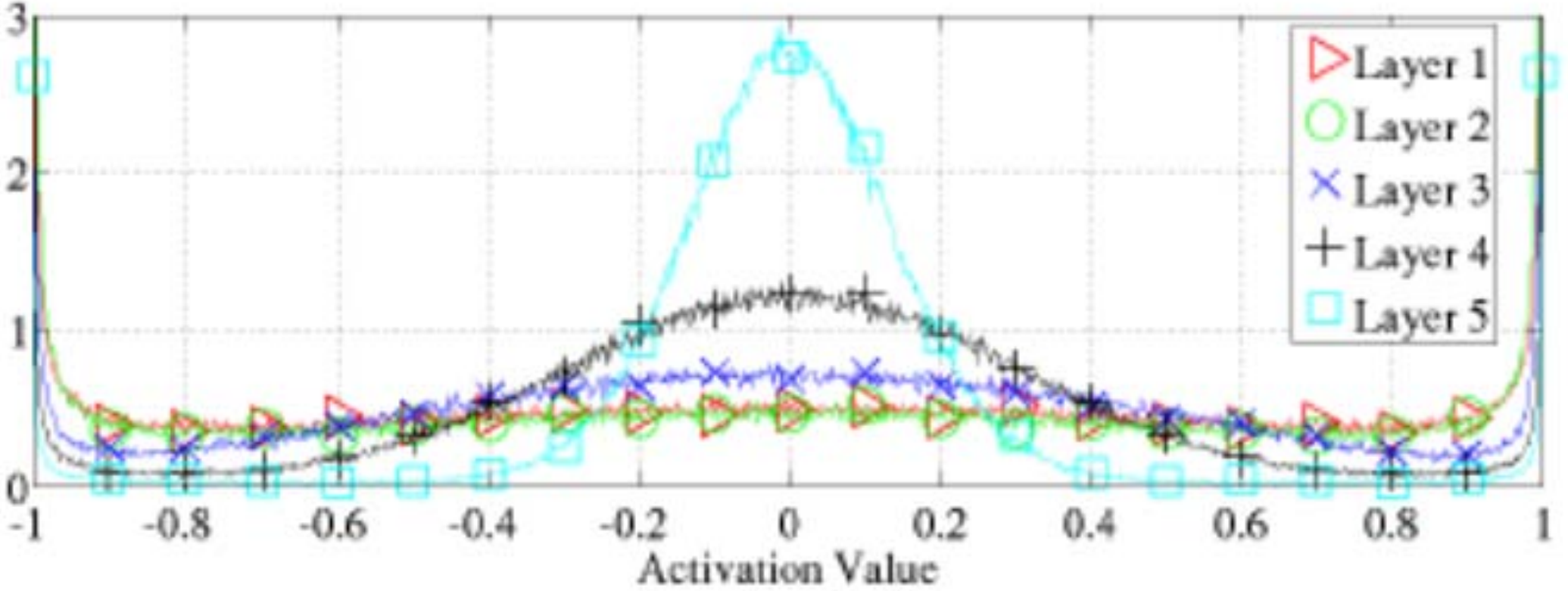


Figure 3: *Top:98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of the activation values for the hyperbolic tangent networks in the course of learning. We see the first hidden layer saturating first, then the second, etc.*

Xavier Glorot & Yoshua Bengio,
"Understanding the difficulty of training deep feedforward neural networks."



Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

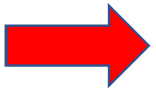
Figure 4: *Activation values normalized histogram at the end of learning, averaged across units of the same layer and across 300 test examples. Top: activation function is hyperbolic tangent, we see important saturation of the lower layers.*

Xavier Glorot & Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks."

Xavier Initialization

- From a forward propagation view, we would like to keep information flowing as

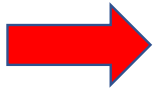
$$\forall(i,i'), Var[z^i] = Var[z^{i'}].$$



$$\mathrm{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$

- From a backward propagation view, we would like to keep information flowing as

$$\forall(i,i'), Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^{i'}}\right].$$



$$\mathrm{Var}(W_i) = \frac{1}{n_{\text{out}}}$$

where $\mathbf{s}^i = \mathbf{z}^i W^i + \mathbf{b}^i$ and $\mathbf{z}^{i+1} = f(\mathbf{s}^i)$.

Xavier Initialization

- To make compromise, we have

$$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

- In practical deployment, we typically use

$$\mathrm{Var}(W) = \frac{1}{n_{\text{in}}}$$

$$\mathrm{Var}(aX) = a^2 \mathrm{Var}(X)$$

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

Xavier Initialization

- For ReLU

$$\mathrm{Var}(W) = \frac{2}{n_{\mathrm{in}}}$$

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

- Intuitive understanding: A ReLU unit is zero for half of its input, so you need to double the size of weight variance to keep the signal's variance constant.

Batch Normalization

- Make zero-mean unit-variance activations for each dimension

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

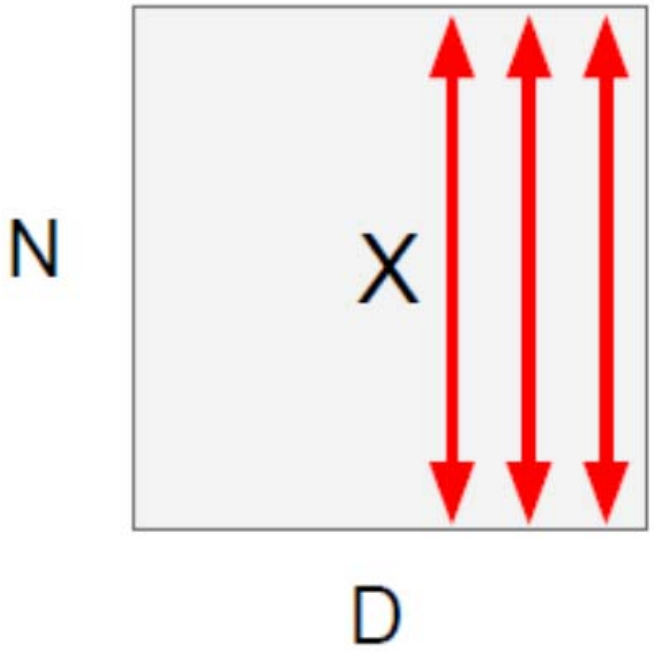
Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Batch Normalization: Accelerating Deep Network Training by ...

https://arxiv.org › cs ▾

by S Ioffe - 2015 - Cited by 8466 - Related articles



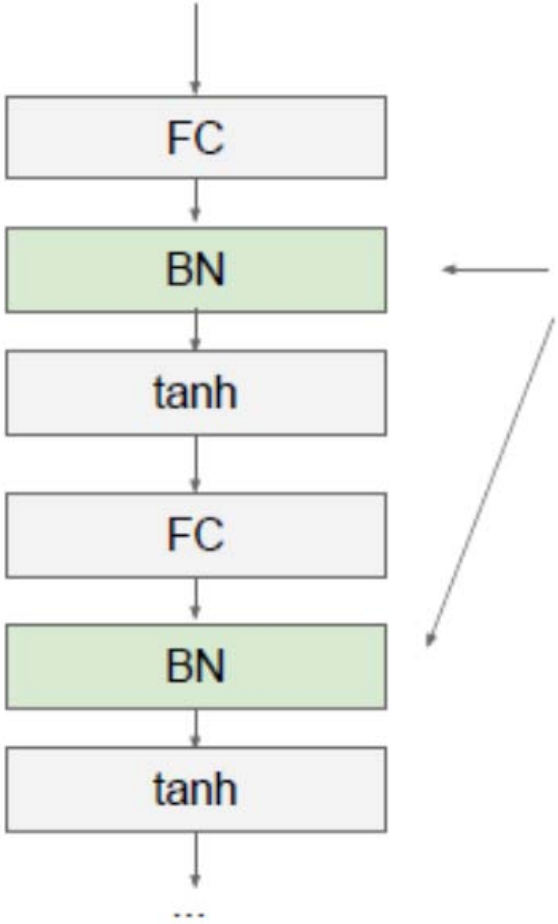
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

N: Batch size

D: Dimension size



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

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

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

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

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

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

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

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

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

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

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

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

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

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

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

The fixed empirical Mean & Variance in test time is based on all batches in training time

Dropout

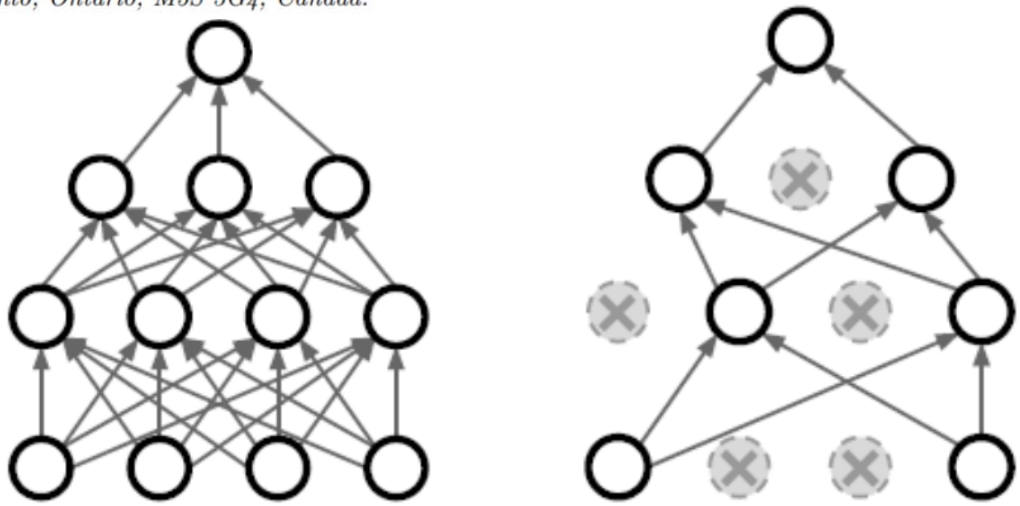
- In each forward pass of *training*, randomly set some neurons to zero

  -- dropping rate is hyperparameter

  -- common rate is 0.5

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.



[PDF] Dropout: A Simple Way to Prevent Neural Networks from Overfitting

jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf

by N Srivastava - 2014 - Cited by 10152 - Related articles

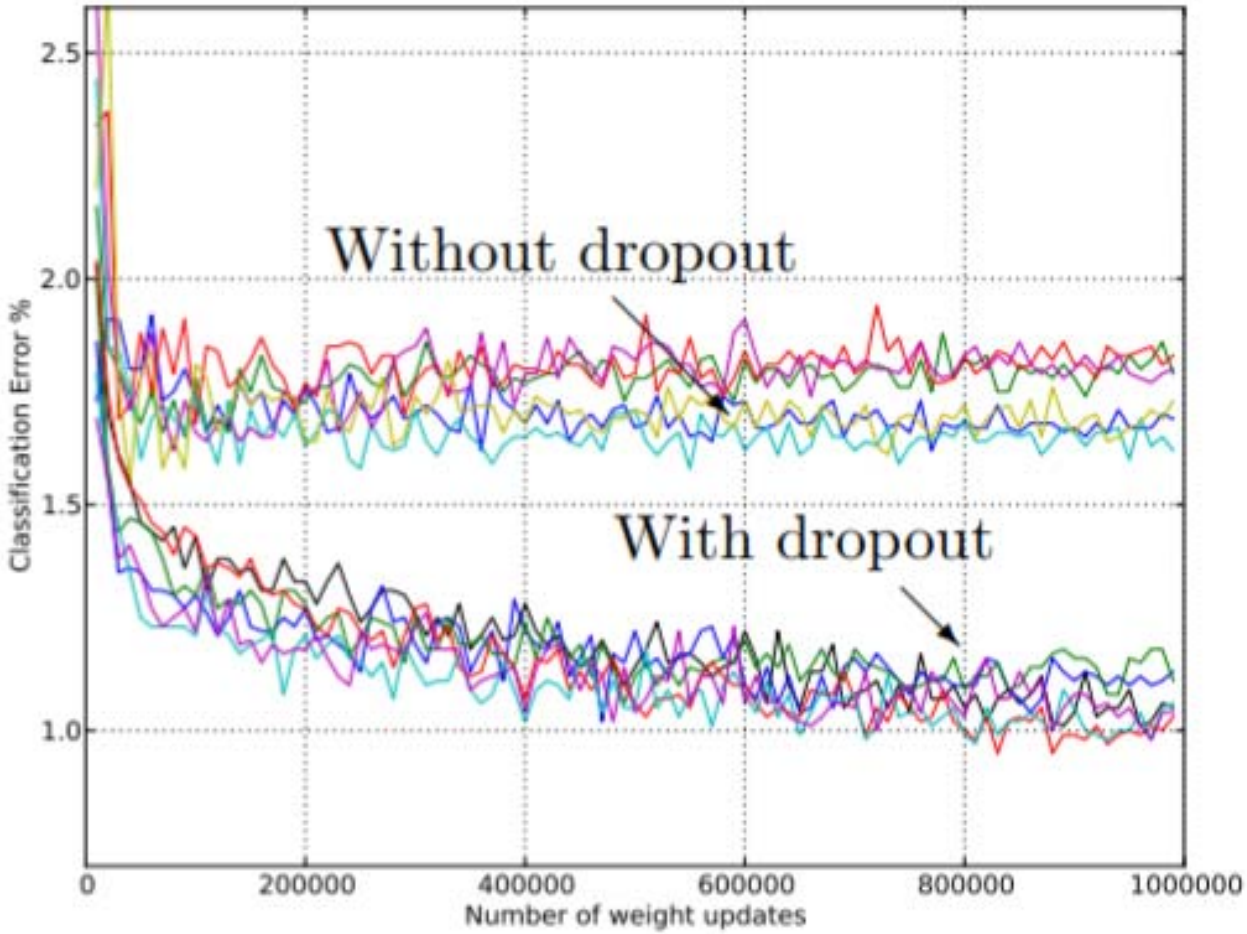
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Dropout Improves Performance



Why Dropout Works

- If a hidden unit is always trained with another hidden unit, they will co-adapt to the training data
  - -- One tends to rely on another
  - -- Likely to go wrong on the new testing data

- If a hidden unit has to work well with different combination of other hidden neurons, it will optimize itself to be useful.

Why Dropout Works

- Each time we present a training batch, we randomly omit each hidden unit with probability *p*
- So we are randomly sampling from a large number of different architectures
  -- All architectures share weights
  -- Every model is very strongly *regularized*
- Each model is trained using different training samples
- Ensemble them together can improve performance

Dropout at Test Time

- No neurons are dropped at test time
- Output of each neuron is scaled by dropout rate

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Dropout Summary

drop in forward pass

scale at test time

More Common Implementation

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

test time is unchanged!

Common Pattern of Regularization Method

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z[f(x, z)] = \int p(z) f(x, z) dz$$

Example: Batch Normalization

Training: Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

DropBlock

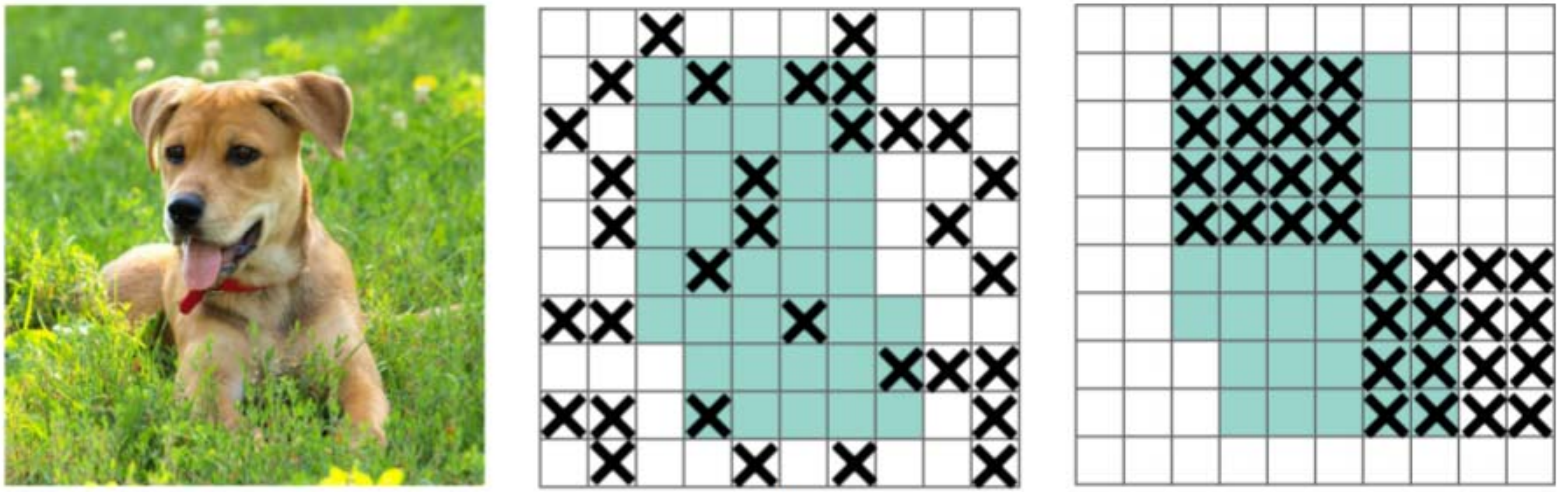
DropBlock: A regularization method for convolutional networks

Golnaz Ghiasi
Google Brain

Tsung-Yi Lin
Google Brain

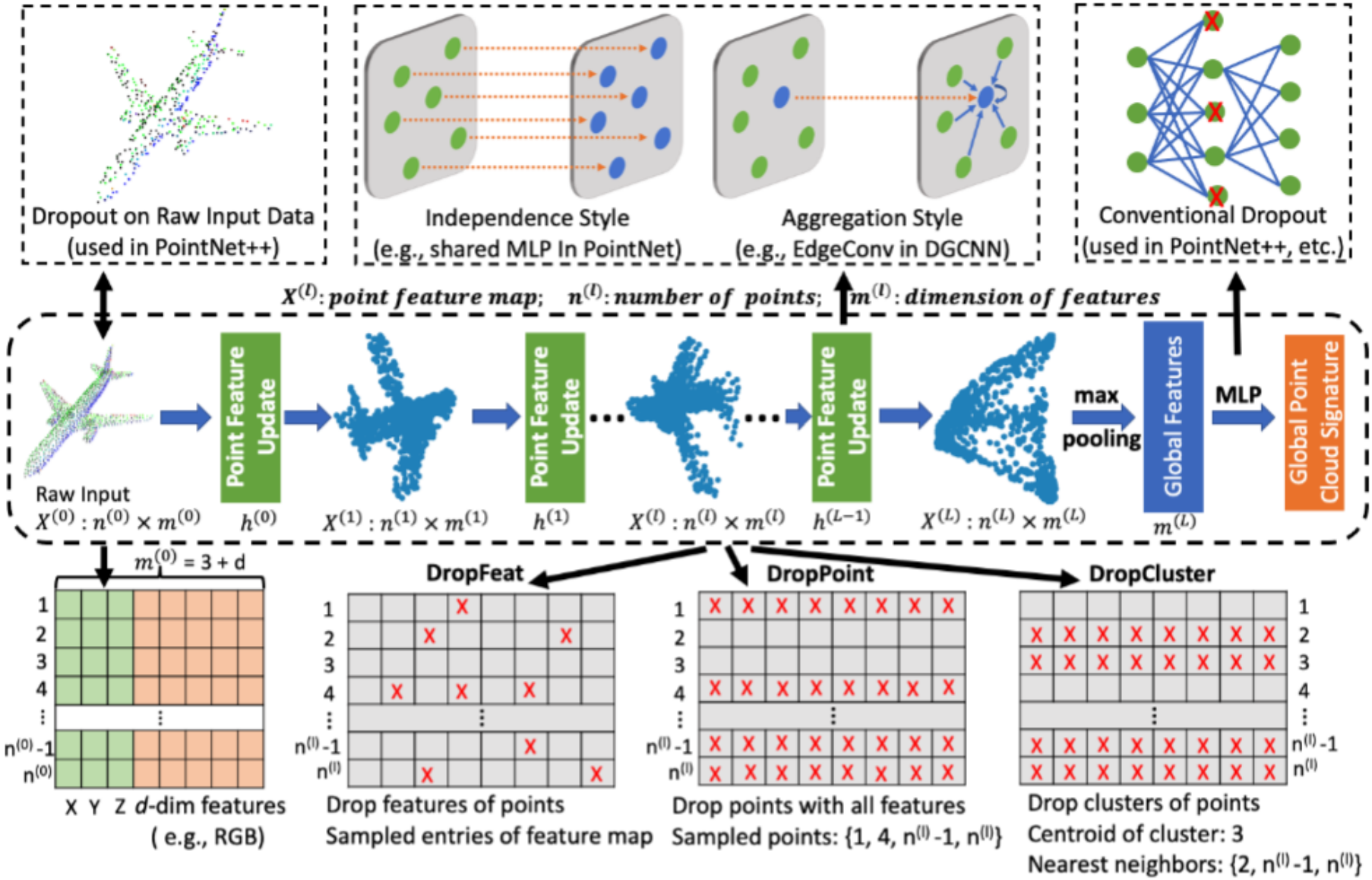
Quoc V. Le
Google Brain

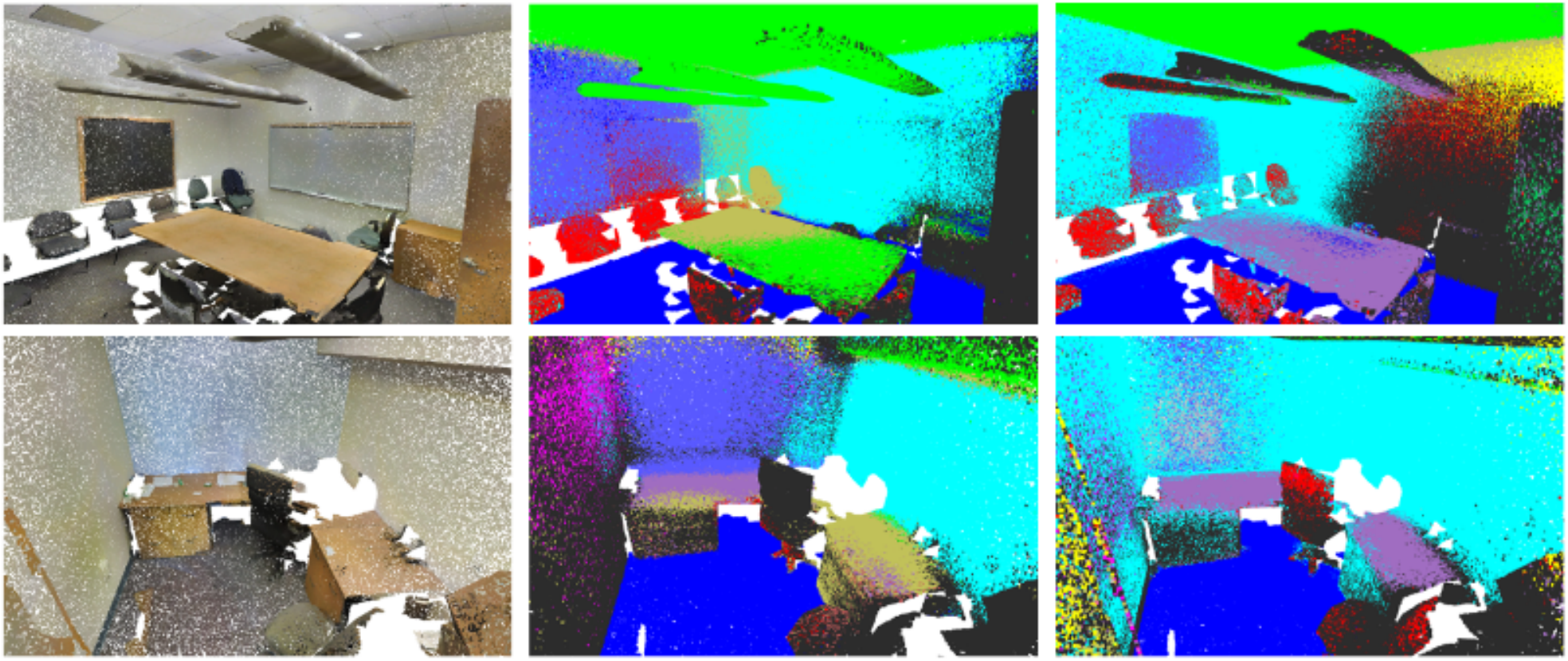
Mechanism and Interpretation



(a) (b) (c)

Figure 1: (a) input image to a convolutional neural network. The green regions in (b) and (c) include the activation units which contain semantic information in the input image. Dropping out activations at random is not effective in removing semantic information because nearby activations contain closely related information. Instead, dropping continuous regions can remove certain semantic information (e.g., head or feet) and consequently enforcing remaining units to learn features for classifying input image.


Dropout on Raw Input Data
(used in PointNet++)
Independence Style
(e.g., shared MLP In PointNet)
Aggregation Style
(e.g., EdgeConv in DGCNN)
Conventional Dropout
(used in PointNet++, etc.)
$X^{(l)}$: point feature map; $n^{(l)}$: number of points; $m^{(l)}$: dimension of features
Raw Input
$X^{(0)} : n^{(0)} \times m^{(0)}$
Point Feature Update
$h^{(0)}$
$X^{(1)} : n^{(1)} \times m^{(1)}$
$h^{(1)}$
$X^{(l)} : n^{(l)} \times m^{(l)}$
$h^{(L-1)}$
$X^{(L)} : n^{(L)} \times m^{(L)}$
max pooling
Global Features
$m^{(L)}$
MLP
Global Point Cloud Signature
$m^{(0)} = 3 + d$
X Y Z d-dim features
(e.g., RGB)
DropFeat
Drop features of points
Sampled entries of feature map
DropPoint
Drop points with all features
Sampled points: {1, 4, $n^{(l)}$-1, $n^{(l)}$}
DropCluster
Drop clusters of points
Centroid of cluster: 3
Nearest neighbors: {2, $n^{(l)}$-1, $n^{(l)}$}


Raw Input
DGCNN
DGCNN w/ DropCluster

Improved Performance

Model	top-1(%)	top-5(%)
ResNet-50	76.51 ± 0.07	93.20 ± 0.05
ResNet-50 + dropout (kp=0.7) [1]	76.80 ± 0.04	93.41 ± 0.04
ResNet-50 + DropPath (kp=0.9) [17]	77.10 ± 0.08	93.50 ± 0.05
ResNet-50 + SpatialDropout (kp=0.9) [20]	77.41 ± 0.04	93.74 ± 0.02
ResNet-50 + Cutout [23]	76.52 ± 0.07	93.21 ± 0.04
ResNet-50 + AutoAugment [27]	77.63	93.82
ResNet-50 + label smoothing (0.1) [28]	77.17 ±0.05	93.45 ±0.03
ResNet-50 + DropBlock, (kp=0.9)	78.13 ± 0.05	94.02 ± 0.02
ResNet-50 + DropBlock (kp=0.9) + label smoothing (0.1)	78.35 ± 0.05	94.15 ± 0.03

Table 1: Summary of validation accuracy on ImageNet dataset for ResNet-50 architecture. For dropout, DropPath, and SpatialDropout, we trained models with different $keep\_prob$ values and reported the best result. DropBlock is applied with $block\_size = 7$. We report average over 3 runs.

Model Ensemble

- Very simple idea

  1. Train multiple Independent models

  2. At test time average their results

  -- Take average of predicted probability distributions, then choose argmax

  3. Enjoy 2% extra performance

Results from MNIST Webpage

large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions]	none	0.35	Ciresan et al. IJCAI 2011
committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.27 +-0.02	Ciresan et al. ICDAR 2011
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.23	Ciresan et al. CVPR 2012

Acknowledgement

Many materials of the slides of this course are adopted and re-produced from several deep learning courses and tutorials.

- Prof. Fei-fei Li, Stanford, CS231n: Convolutional Neural Networks for Visual Recognition (online available)
- Prof. Andrew Ng, Stanford, CS230: Deep learning (online available)
- Prof. Yanzhi Wang, Northeastern, EECE7390: Advance in deep learning
- Prof. Jianting Zhang, CUNY, CSc G0815 High-Performance Machine Learning: Systems and Applications
- Prof. Vivienne Sze, MIT, “Tutorial on Hardware Architectures for Deep Neural Networks”
- Pytorch official tutorial https://pytorch.org/tutorials/
- https://github.com/jcjohnson/pytorch-examples