Deep Learning Seminar
Chapter 7. Regularization for Deep Learning
- Part 2 -

Hyun-Lim YANG

Department of Information and Communication Engineering

DGIST

2017.08.09

Contents

Chapter 7. Regularization for Deep Learning

- **Part 1**
  - 7.1 Parameter Norm Penalties
  - 7.2 Norm Penalties as Constrained Optimization
  - 7.3 Regularization and Under-Constrained Problems
  - 7.4 Dataset Augmentations
  - 7.5 Noise Robustness
  - 7.6 Semi-Supervised Learning
  - 7.7 Multi-Task Learning
  - 7.8 Early Stopping
- **Part 2**
  - 7.9 Parameter Tying and Parameter Sharing
  - 7.10 Sparse Representations
  - 7.11 Bagging and Other Ensemble Methods
  - 7.12 Dropout
  - 7.13 Adversarial Training
  - 7.14 Tangent Distance, Tangent Prop, and Manifold Tangent Classifier

Chapter 7. Regularization for Deep Learning

- **Part 1**
  - 7.1 Parameter Norm Penalties
  - 7.2 Norm Penalties as Constrained Optimization
  - 7.3 Regularization and Under-Constrained Problems
  - 7.4 Dataset Augmentations
  - 7.5 Noise Robustness
  - 7.6 Semi-Supervised Learning
  - 7.7 Multi-Task Learning
  - 7.8 Early Stopping
- **Part 2**
  - **7.9** Parameter Tying and Parameter Sharing
  - **7.10** Sparse Representations
  - **7.11** Bagging and Other Ensemble Methods
  - **7.12** Dropout
  - **7.13** Adversarial Training
  - **7.14** Tangent Distance, Tangent Prop, and Manifold Tangent Classifier

Chapter 7. Regularization for Deep Learning

- **Part 1**
  - 7.1 Parameter Norm Penalties
  - 7.2 Norm Penalties as Constrained Optimization
  - 7.3 Regularization and Under-Constrained Problems
  - 7.4 Dataset Augmentations
  - 7.5 Noise Robustness
  - 7.6 Semi-Supervised Learning
  - **7.7 Multi-Task Learning**
  - 7.8 Early Stopping
- **Part 2**
  - **7.9 Parameter Tying and Parameter Sharing**
  - ~~7.10 Sparse Representations~~ **Discuss in Chapter 14. Autoencoder**
  - **7.11 Bagging and Other Ensemble Methods**
  - **7.12 Dropout**
  - **7.13 Adversarial Training**
  - ~~7.14 Tangent Distance, Tangent Prop, and Manifold Tangent Classifier~~ **Chapter 14.**

Parameter Tying and Parameter Sharing

Parameter Tying

- **Parameter dependency**
  - L^2 regularization (or weigh decay) penalizes model parameters for deviating from the fixed value of zero
  - Sometimes we need other ways to express prior knowledge of parameters
  - We may know from domain and model architecture that there should be some dependencies between model parameters

- **The goal of parameter tying**
  - We want to express that certain parameters should be close to one another

A scenario of parameter tying

- **Two models performing the same classification task (with same set of classes) but with somewhat different input distributions**
- **Model A with parameter $w^{(A)}$**
- **Model B with parameter $w^{(B)}$**
- **The two models will map the input to two different, but related output:**

$$\hat{y}^{(A)} = f(w^{(A)}, x)$$

$$\hat{y}^{(B)} = g(w^{(B)}, x')$$

L^2 penalty for parameter tying

- **If the tasks are similar enough (perhaps with similar input and output distributions) then we believe that the model parameters should be close to each other:**

$$\forall\, i, w_i^{(A)} \approx w_i^{(B)}$$

- **We can leverage this information via regularization**
- **Use a parameter norm penalty (other choices are also possible)**

$$\Omega\left(w^{(A)}, w^{(B)}\right) = \left\| w^{(A)} - w^{(B)} \right\|_2^2$$

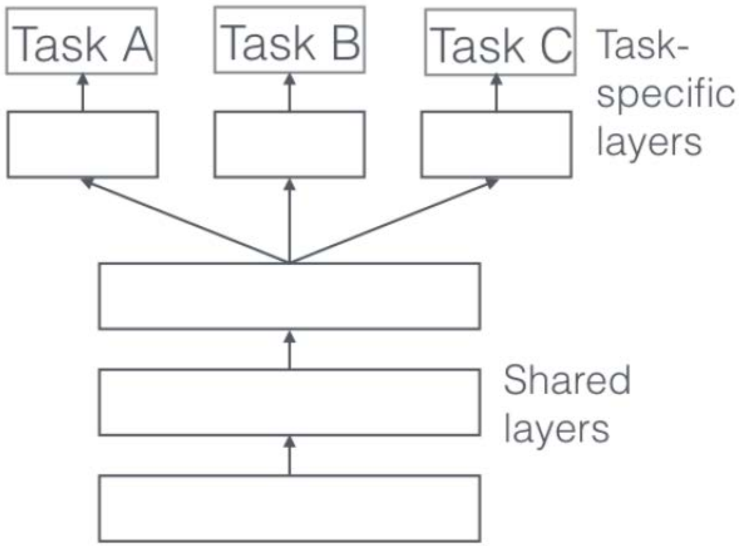
$$\underbrace{\boxed{\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})}}_{\text{Regularized objective function}} = \underbrace{\boxed{J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})}}_{\text{Original objective function}} + \underbrace{\boxed{\alpha\Omega(\boldsymbol{\theta})}}_{\text{Penalty term}}$$

Multi-Task Learning (MTL)

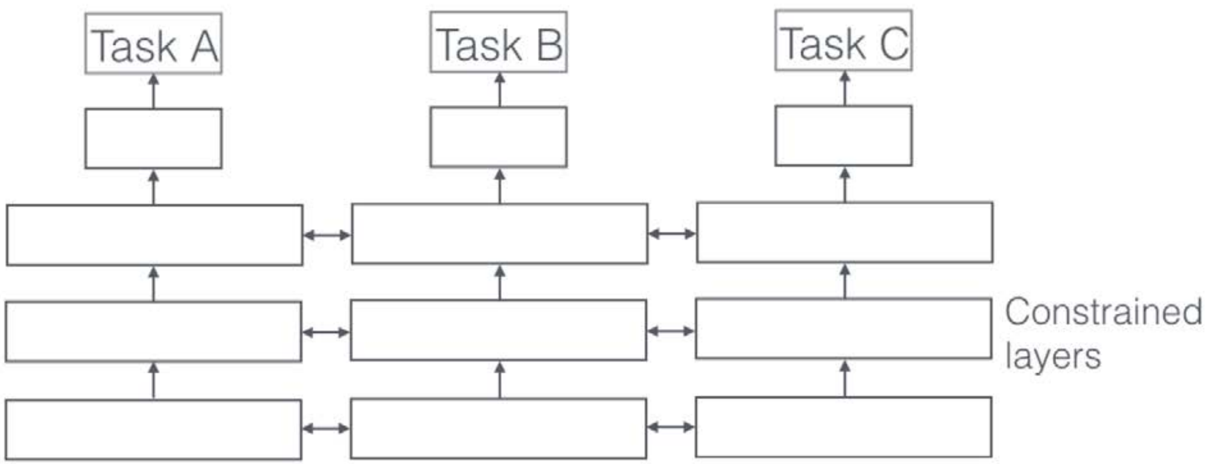
- **Sharing the representation between related tasks**
  - We can enable out model to generalize better on our original task
  - Another approach of bagging (with different cost functions)
- **MTL is also known as:**
  - Joint learning, Learning to learn, learning with auxiliary tasks
- **Optimizing more than one loss function**
- **Improves generalization by leveraging the domain-specific information contained in the training data**
  - Even if the problem optimizing one loss functions, there might be the chances to improve performance by adding an auxiliary task upon the major task
- **Motivated by human learning process**

Two MTL Methods

- **Hard parameter sharing**
  - Greatly reduce the risk of overfitting
  - Similar concept of bagging



- **Soft parameter sharing**
  - Take a role of regularization



Learning task relationship with Regularization

- **Notation**
  - Task T, for each task t, we have a model m_t with parameters a_t of dimensionality d
  - The parameter vector a_t and parameter matrix A is:

$$a_t = [a_{1,t}, \cdots, a_{d,t}] \qquad A = \begin{bmatrix} \vdots & \vdots & \vdots \\ a_{\cdot 1} & \cdots & a_{\cdot T} \\ \vdots & \vdots & \vdots \end{bmatrix}$$

i-th features of the model for every task

Parameter a_j corresponding to the j-th model

- **Learning task relationship**

$$\Omega = \|\bar{a}\|^2 + \frac{\lambda}{T}\sum_{t=1}^{T}\|a_{\cdot,t} - \bar{a}\|^2 \qquad \textit{where,} \quad \bar{a} = \left(\sum_{t=1}^{T} a_{\cdot,t}\right)/T$$

Regularized objective function　　Penalty term

$$\boxed{\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})} = \boxed{J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})} + \boxed{\alpha\Omega(\boldsymbol{\theta})}$$

Original objective function

Parameter Sharing

- **Parameter sharing forces sets of parameters to be equal**
- **Only a subset of parameters (the unique set) need to be stored in memory (memory efficient than parameter tying, especially in CNN)**

- **Example case : parameter sharing in a convolution layer of CNN**

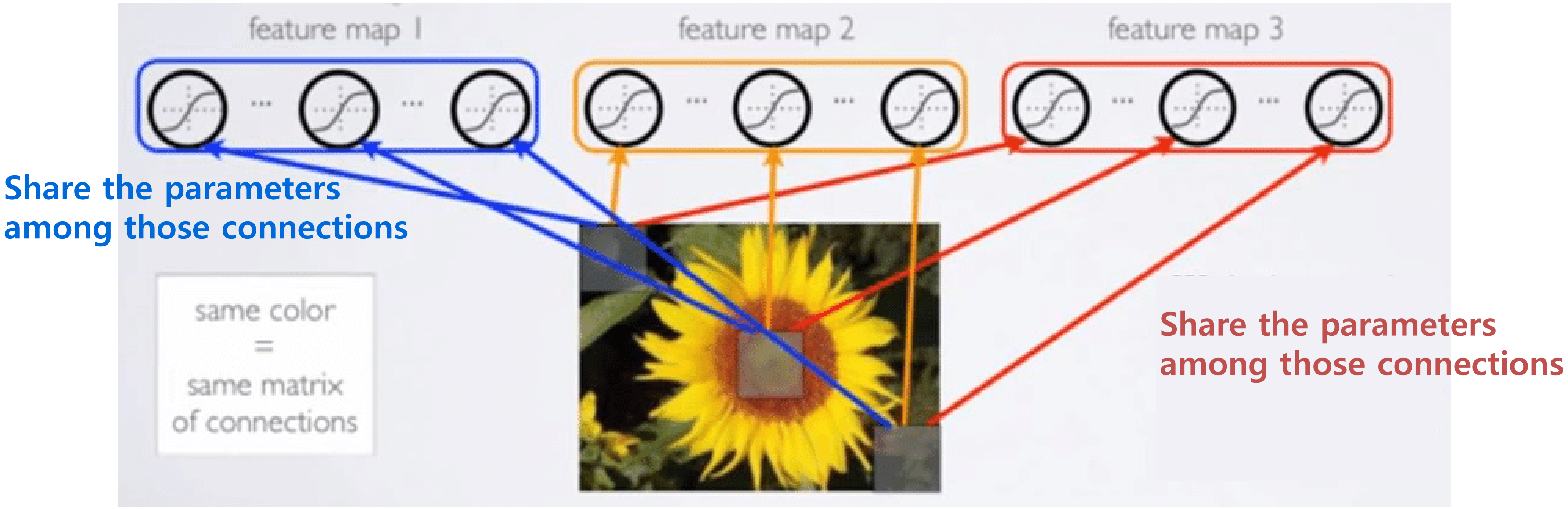
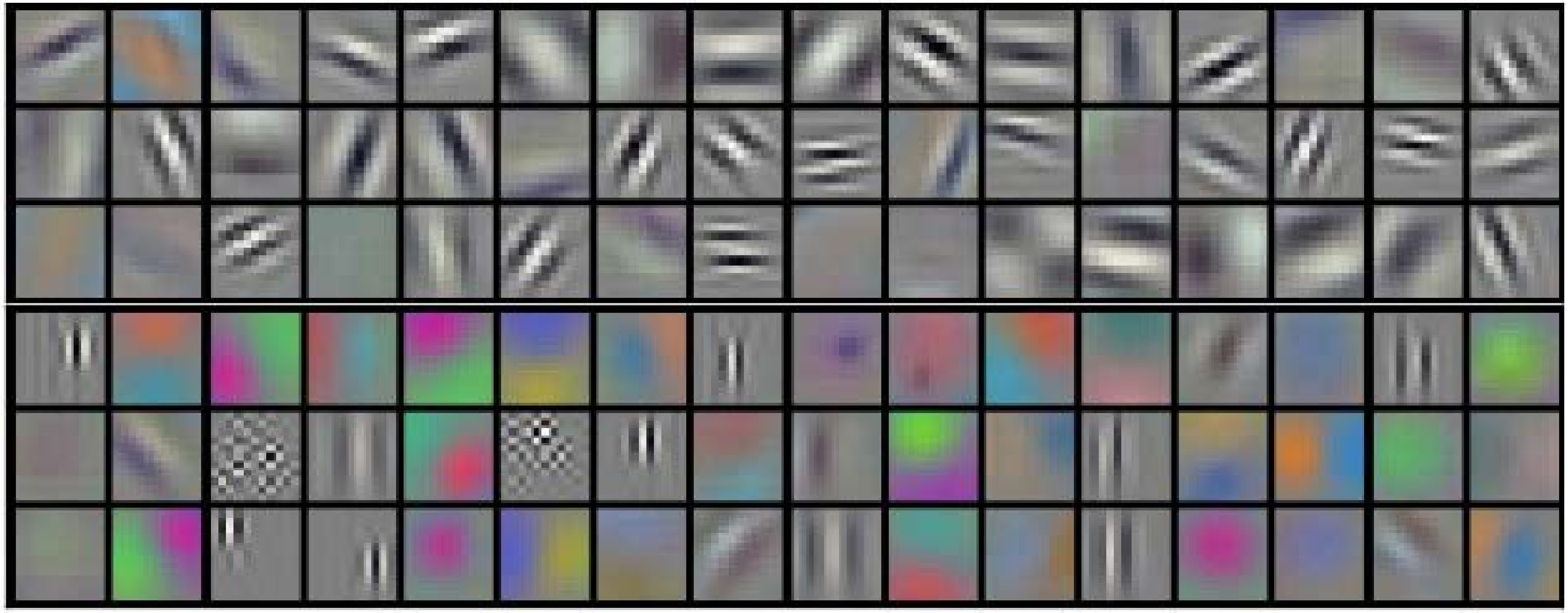


Image from Hugo Larochelle's lecture on YouTube (https://www.youtube.com/watch?v=aAT1t9p7ShM)

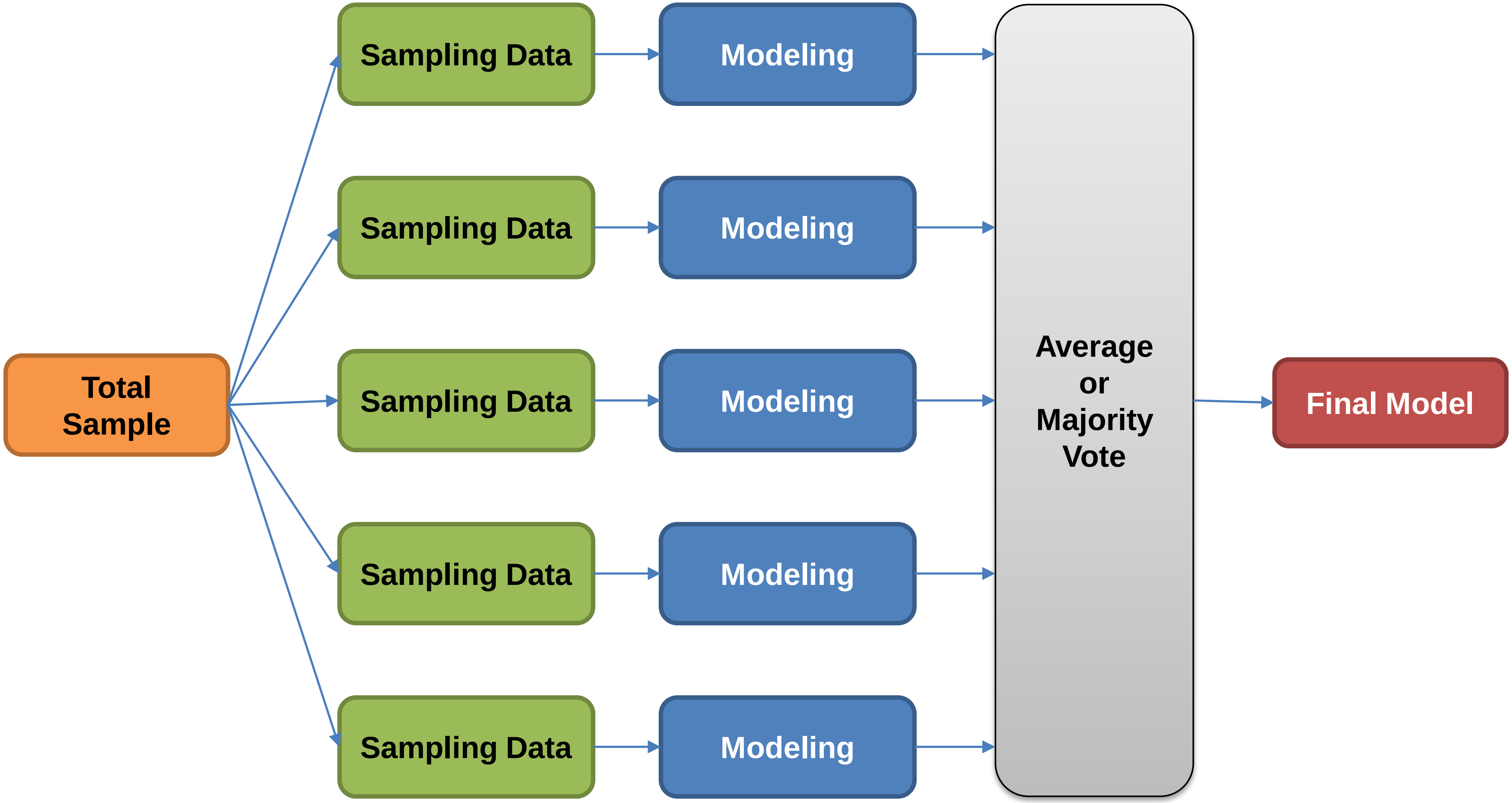
Example of CNN filters

- **96 filters from AlexNet**

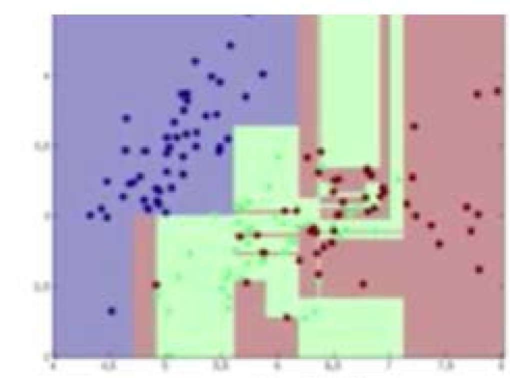


Bagging and Other Ensemble Methods

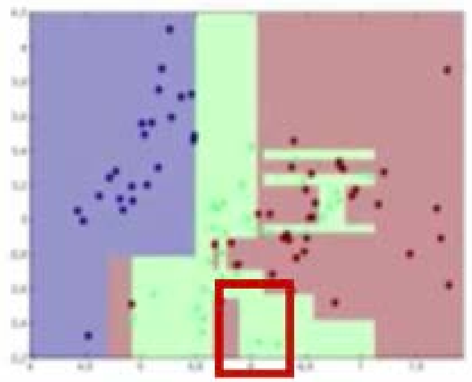
The Concept of Bagging



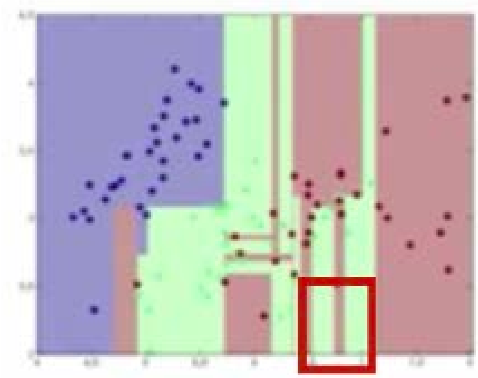
Tree Based Bagging



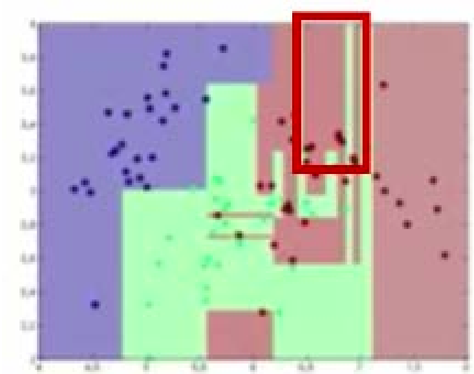
<Model with Full dataset>



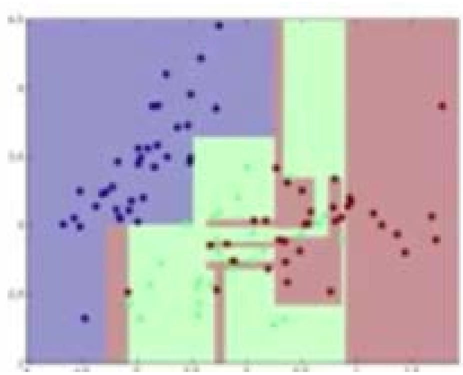
<Model No. 1>



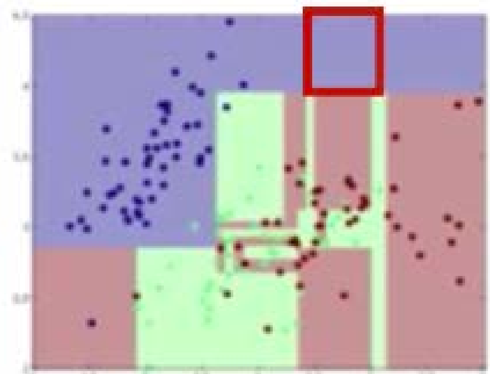
<Model No. 2>



<Model No. 3>



<Model No. 4>



<Model No. 5>

More Details of Bagging

- Short for *Boostrap aggregation*
- Technique for reducing generalization error by combining several models
- a.k.a., *model averaging* or *ensemble methods*
- Procedure
  - Split the input data to K clusters with N' examples (sampling with replacement)
  - Train a classifier with a random sampled cluster
  - For testing, take each examples of test data to all classifier
  - Each classifier votes on the output, take majority

Why dose the Bagging work? (1)

- Consider the K regression models (with minimize MSE)
- Suppose that each model make an error ϵ_i on each model
  - Errors drawn from a zero-mean multivariate normal dist.

$$variance\ \ v = \mathbb{E}[\epsilon_i^2] \qquad covariance\ \ c = \mathbb{E}[\epsilon_i \epsilon_j]$$

- Error made by the average prediction of models is: $\frac{1}{k}\sum_i \epsilon_i$
- The expected squared error of the ensemble predictor is:

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j\right)\right]$$

$$= \frac{1}{k^2}\left\{k\mathbb{E}\left[\epsilon_i^2\right] + k(k-1)\mathbb{E}\left[\epsilon_i \epsilon_j\right]\right\}$$

$$= \frac{1}{k}v + \frac{k-1}{k}c$$

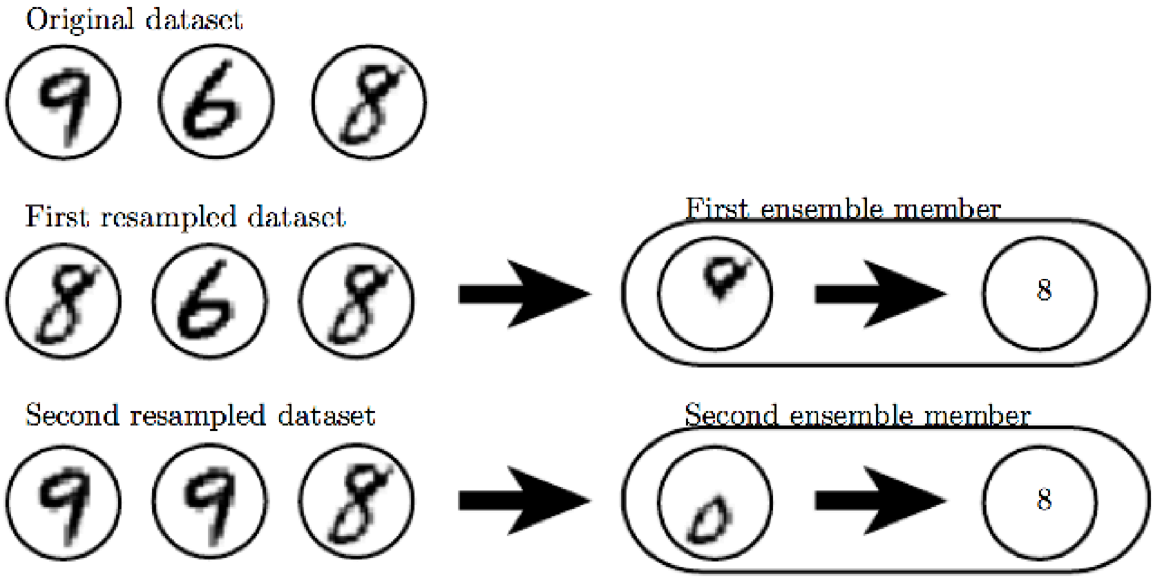
Why dose the Bagging work? (2)

- **The expected squared error of the ensemble predictor is:**

$$\frac{1}{k}v + \frac{k-1}{k}c$$

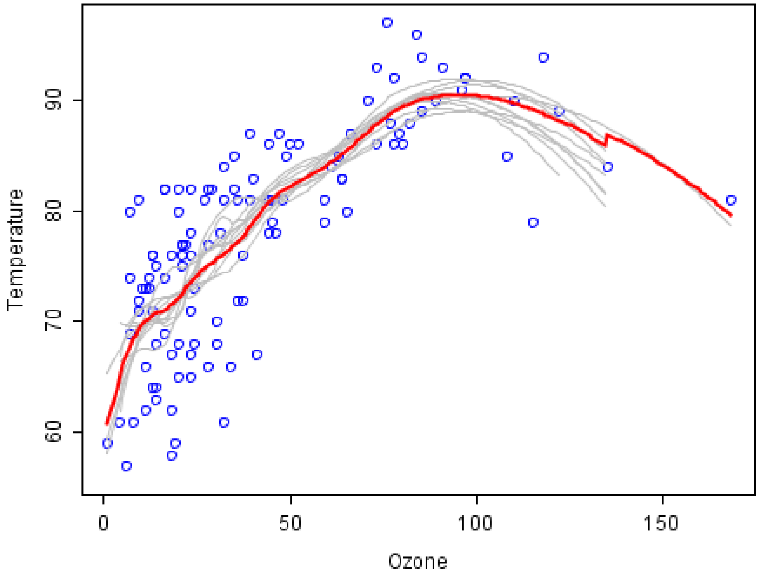
  - If the errors are perfectly correlates, $\boldsymbol{c = v}$, it will not work at all
  - If the errors are perfectly uncorrelated, $\boldsymbol{c = 0}$, error will be only $\boldsymbol{\frac{1}{k}v}$

- **Intuitive example**



Why dose the Bagging work? (3)

- **Machine Learning Example**
  - Trend regression on the data Ozone-Temperature
  - Gray line is regression line with each samples
  - Red line is average line



- **The case should not apply bagging**
  - When the sample data size is small
  - When the data is noisy
  - When the data have dependencies

Tacit rules in Bagging

- **OOB (Out-of-Bag) sampling**
  - Special rule for sampling with replacement
  - If we sample the example with random sampling replacement, the selecting probability of each example is:

$1-\left(1-\frac{1}{N}\right)^{N}$ If N is large enough, then $\lim_{n\to\infty}\left\{1-\left(1-\frac{1}{N}\right)^{N}\right\} = 1-\frac{1}{e} \approx 0.632$

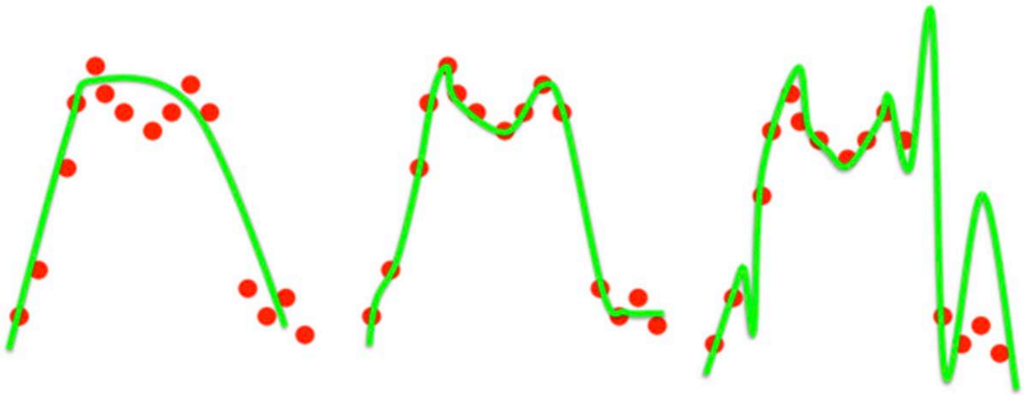
- **Bagging in Neural Networks**
  - Random initialization
  - Random selection of minibatches
  - Differences in hyperparameter
  - …
- ***Usually discouraged when benchmarking algorithms for scientific papers, because of its power and reliability***
  - It's the benefit from the price of increased computations and memory

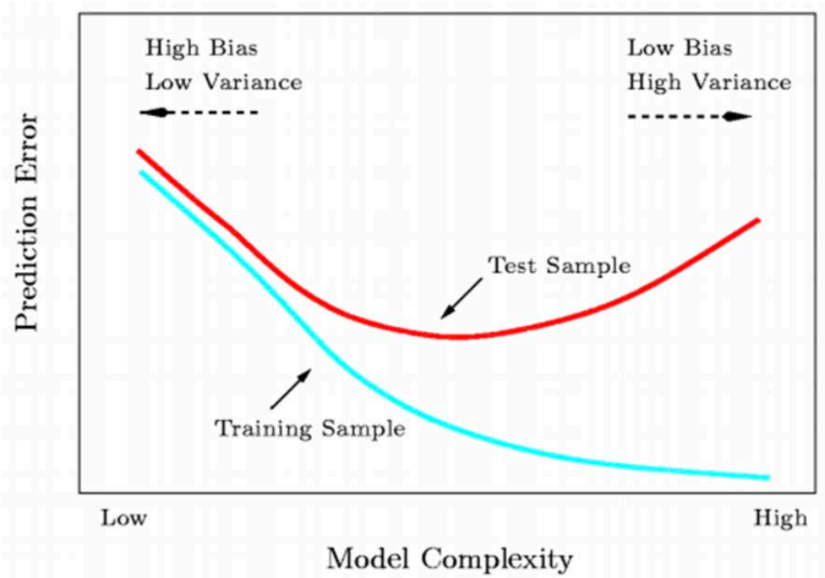
Dropout

References:
[1] Nitish Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research 15 (2014),* 1929-1958
[2] Hinton, Geoffrey E., et al., "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580* (2012).
[3] Krizhevsky, Alex et al., "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.
[4] Wan, Li, et al. "Regularization of neural networks using dropconnect." *Proceedings of the 30th international conference on machine learning (ICML-13)*. 2013.
[5] Baldi, Pierre, and Peter Sadowski. "The dropout learning algorithm." *Artificial intelligence* 210 (2014): 78-122.

Overfitting

- **Excessive focus on train data, resulting in worse results on actual test data**



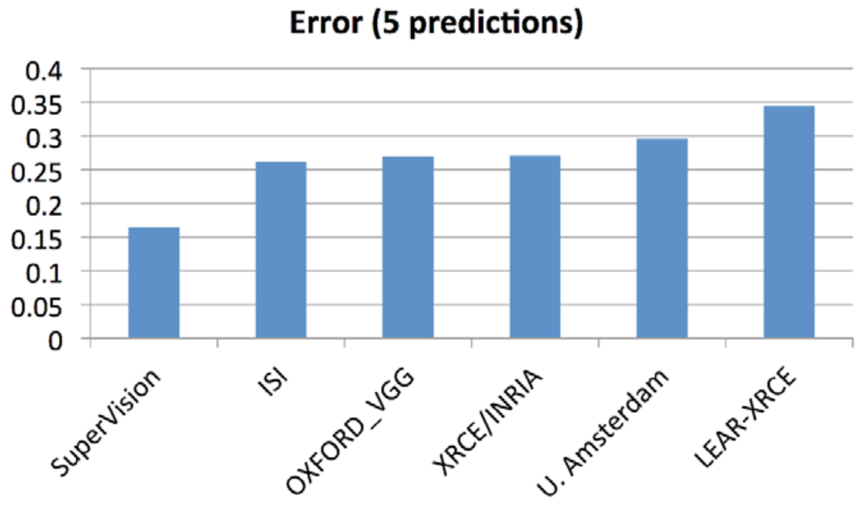


Solutions for Overfitting

- **Regularization**
  - L1-norm penalty
  - L2-norm penalty
- **Data augmentation**
- **Dropout (2012)**
  - A method of bagging applied to neural networks
  - An inexpensive but powerful method of regularizing a broad family of models
- **Batch Normalization (2015)**
  - It is described further in Chapter 8. Optimization

Research in Dropout

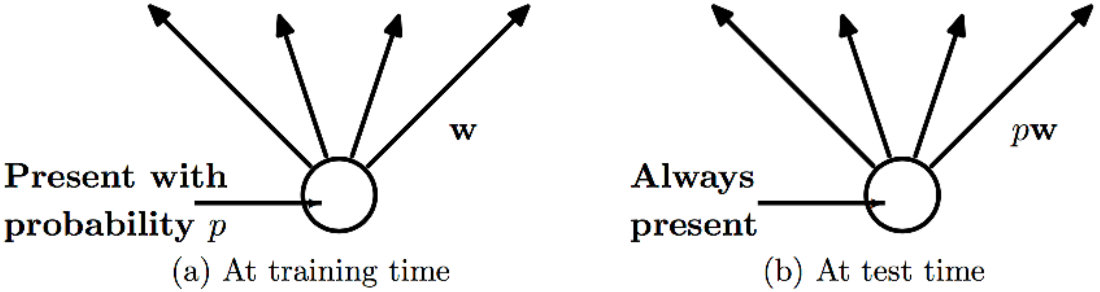
- **First proposed by G.E. Hinton (2012)[2]**
- **Became popular by *AlexNet* (2012)[3]**
  - Winner in ILSVRC-2012 (ImageNet challenge)
  - AlexNet outperforms the other models at most 2x
  - CNN model with ReLU, Dropout, Data augmentation, GPU
  - Applied the dropout at Full-Connected layer



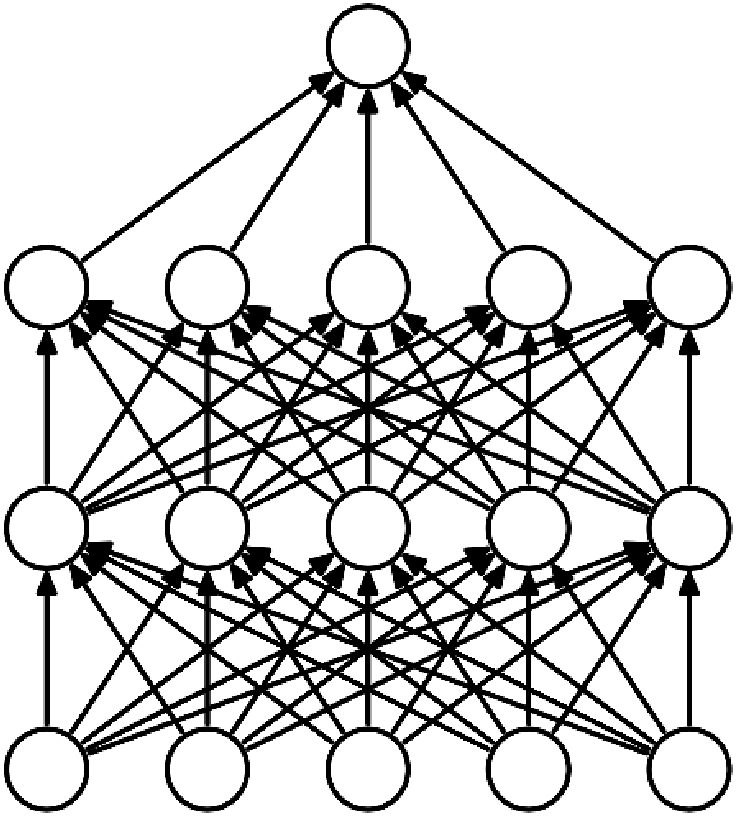
- **Reinforced by S. Nitish (2014)[1]**
  - Strengthen the theoretical background, extend to convolutional layer

Dropout

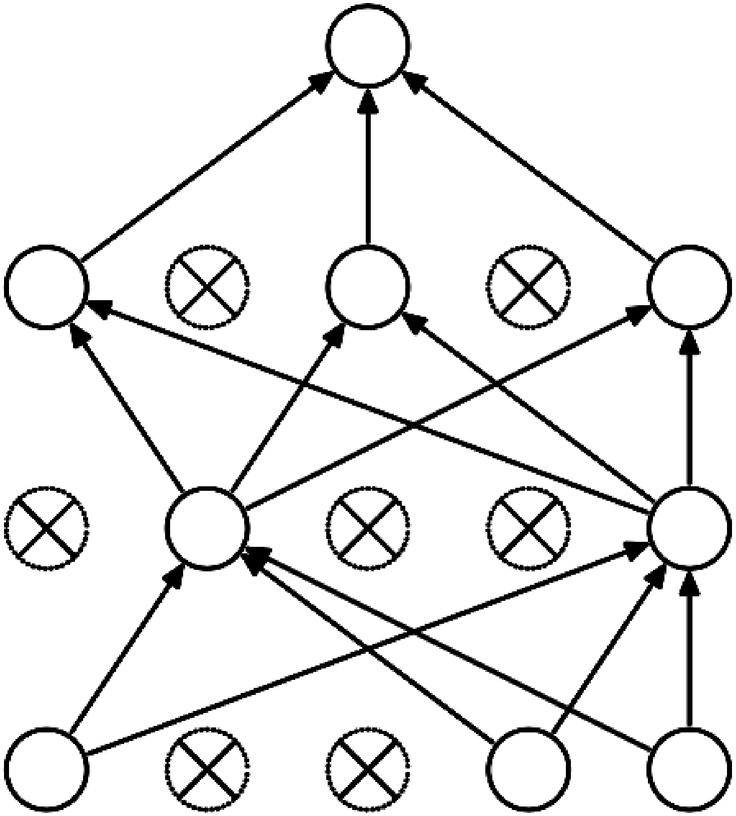
- **A technique that omits a portion of the network**
  - We can surly improve the performance by model combination like as Bagging concept
  - However, if neural network is too deep to build the multiple models, it might be costly and inefficient
  - Also, it takes long time to inference the input with multiple models
- **Dropout addressed the two problems**
  - Omit the neurons, to mimic the voting in ensemble technique, instead of building the multiple models
  - Product the probability that a neurons will survive to weight, at inference level



Dropout



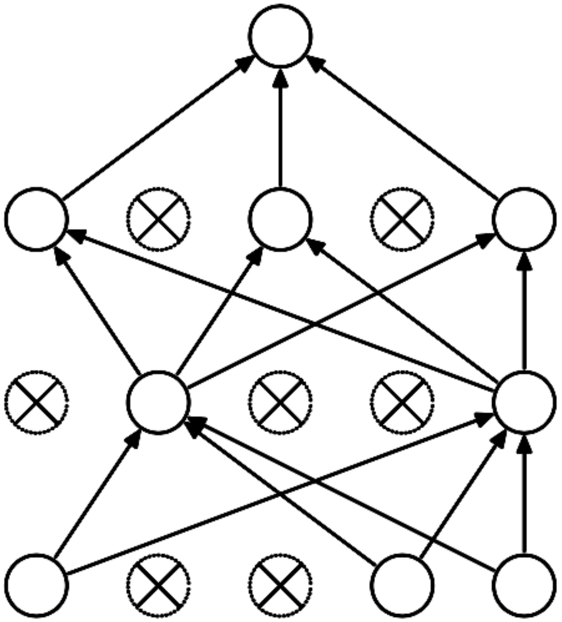
(a) Standard Neural Net



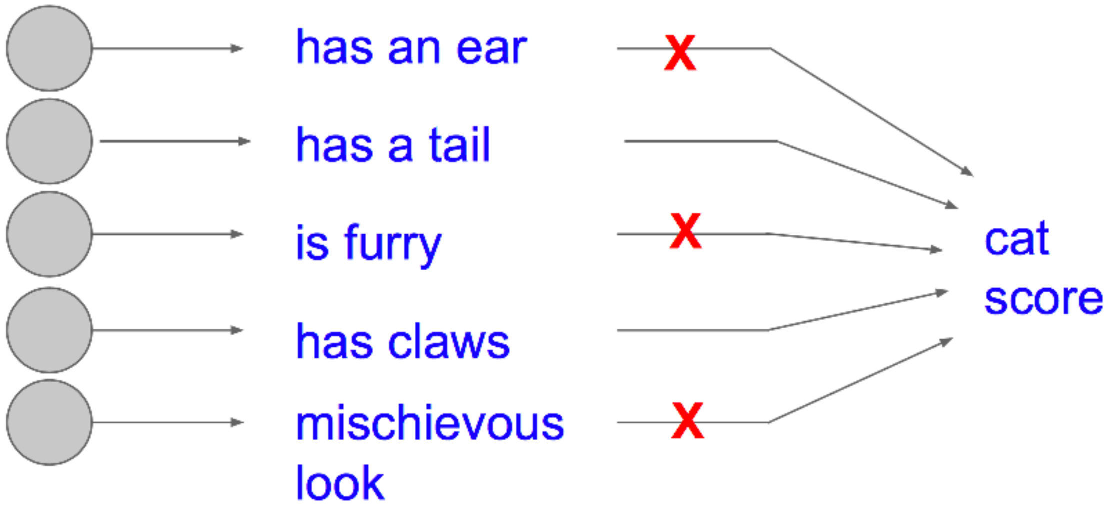
(b) After applying dropout.

Effect of Dropout

- **Avoid the co-adaptation[2]**
  - Co-adaptation: the trend that some neurons tend to represent similar features
  - Capture the clear features by avoiding co-adaptation

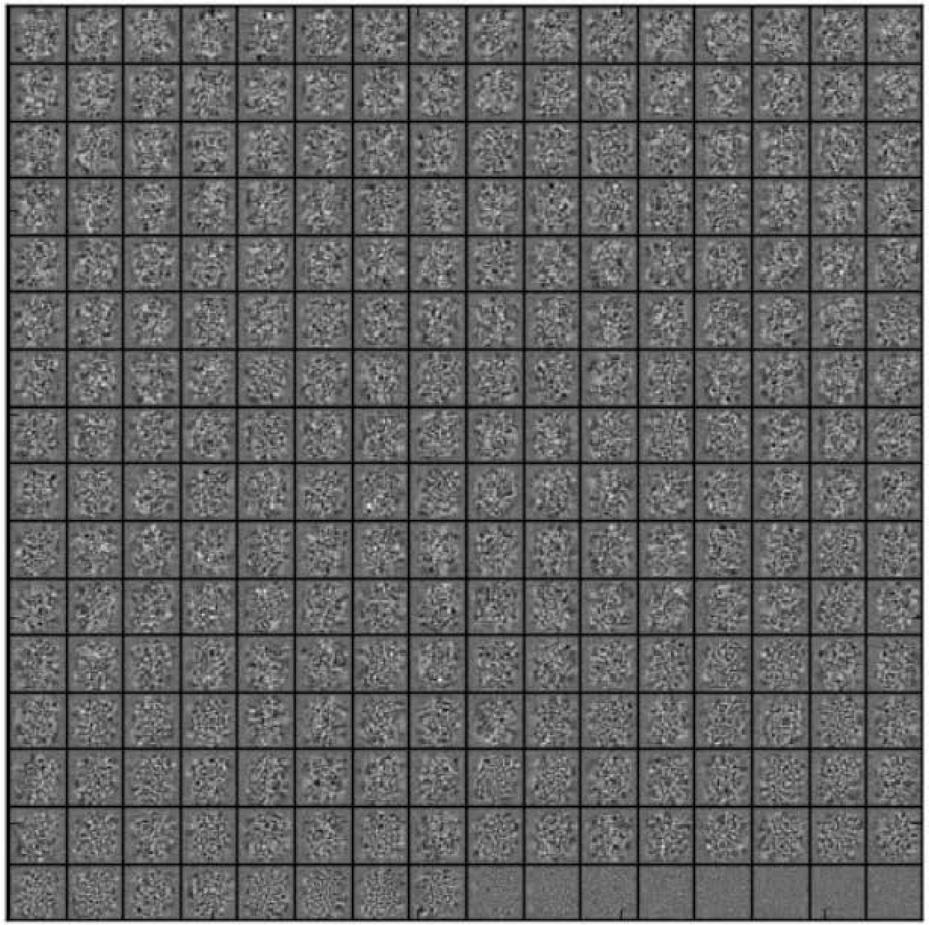


(b) After applying dropout.

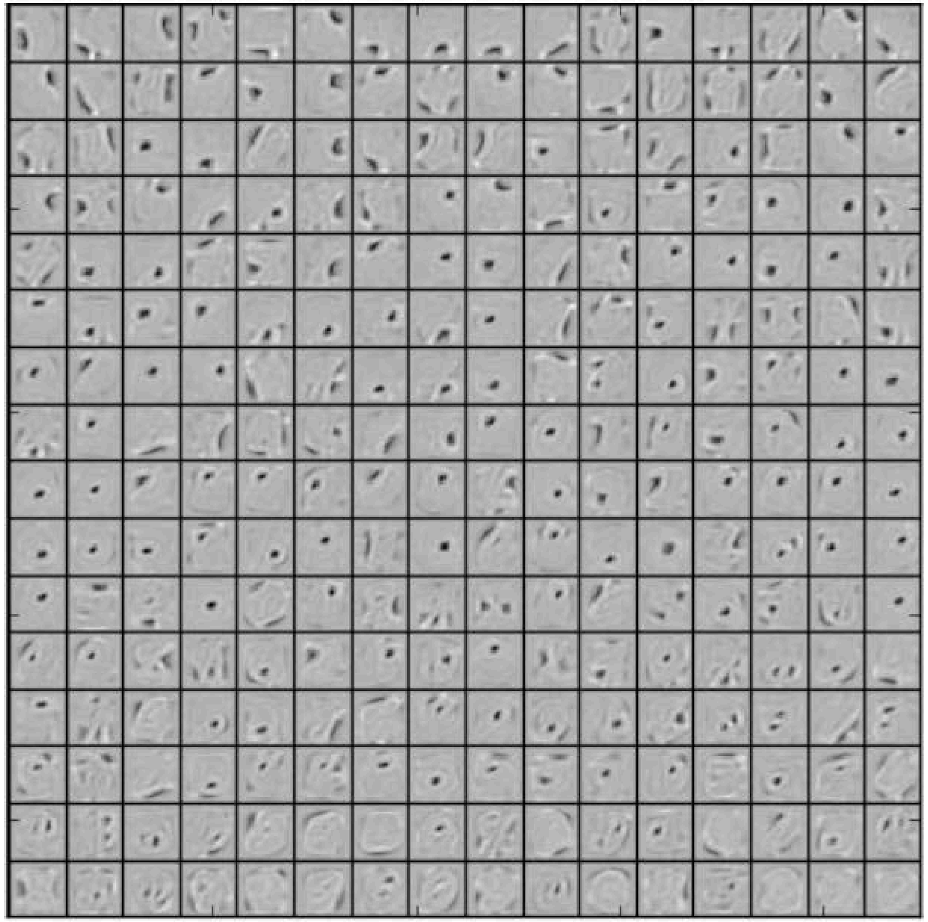


Effect of Dropout

- **Effects on image classification models for MNIST**

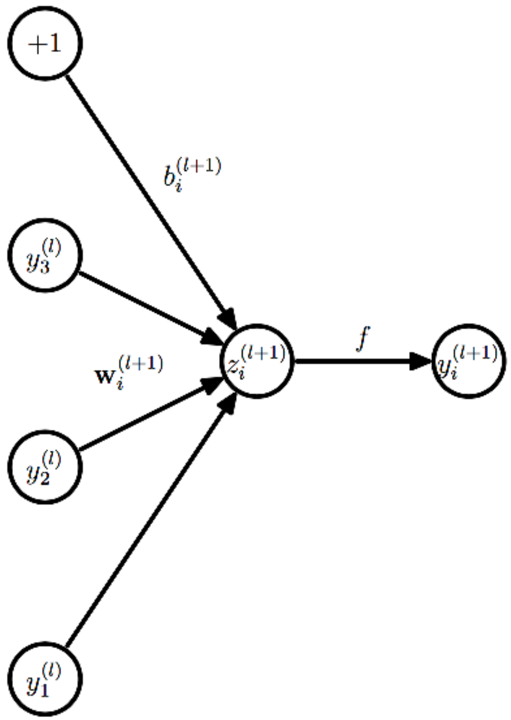


(a) Without dropout

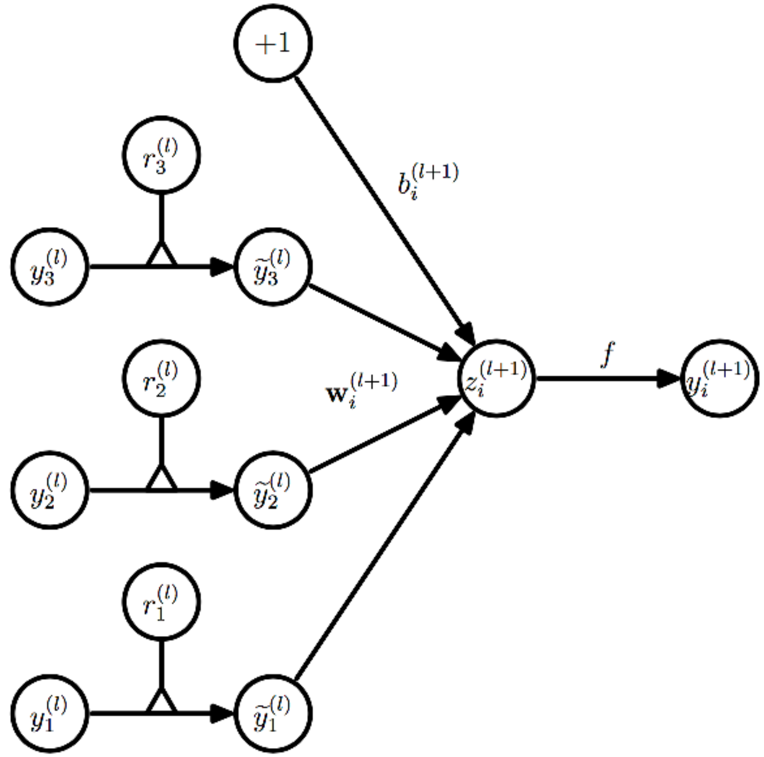


(b) Dropout with $p = 0.5$.

Dropout Modeling



(a) Standard network

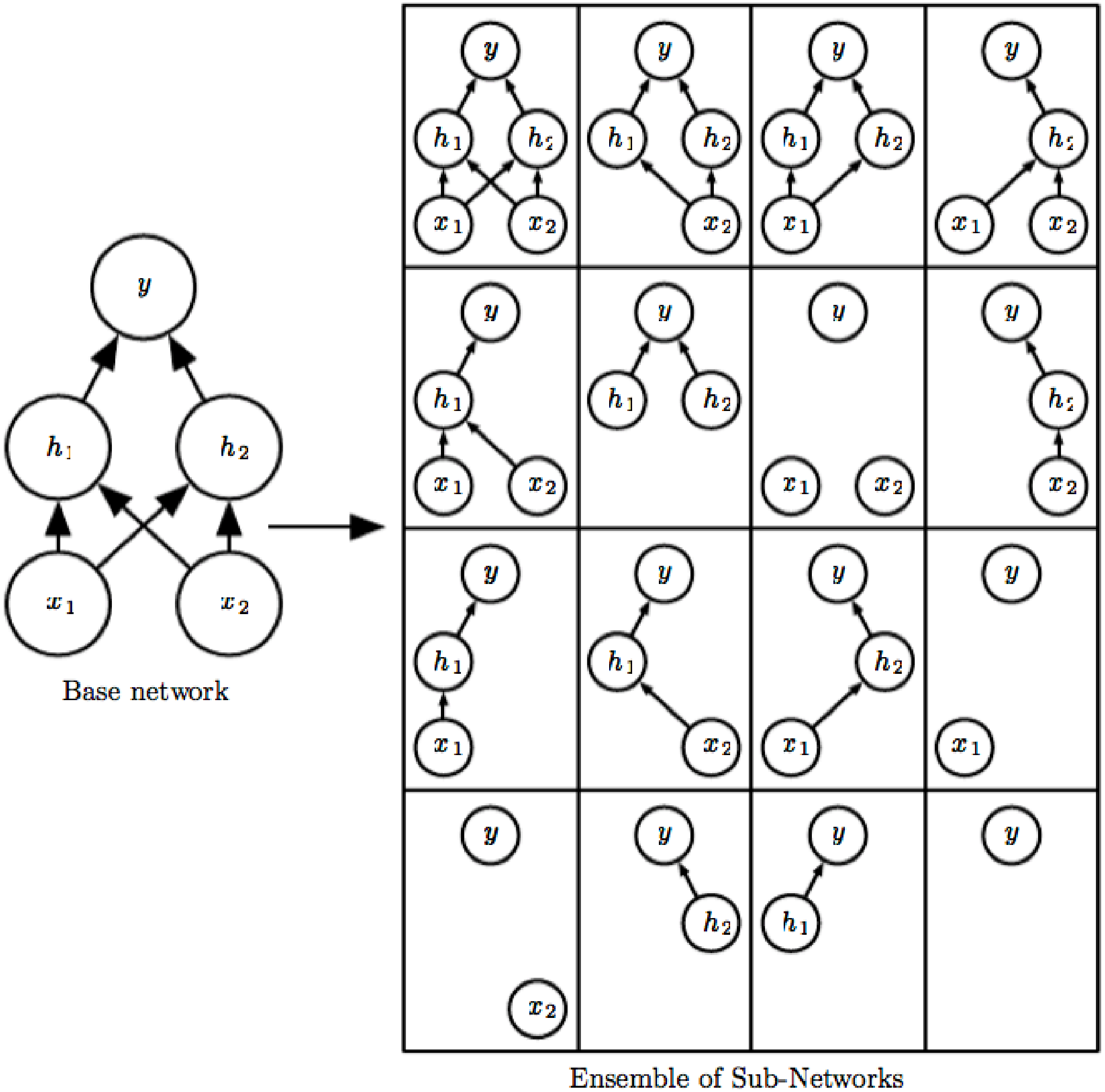


(b) Dropout network

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

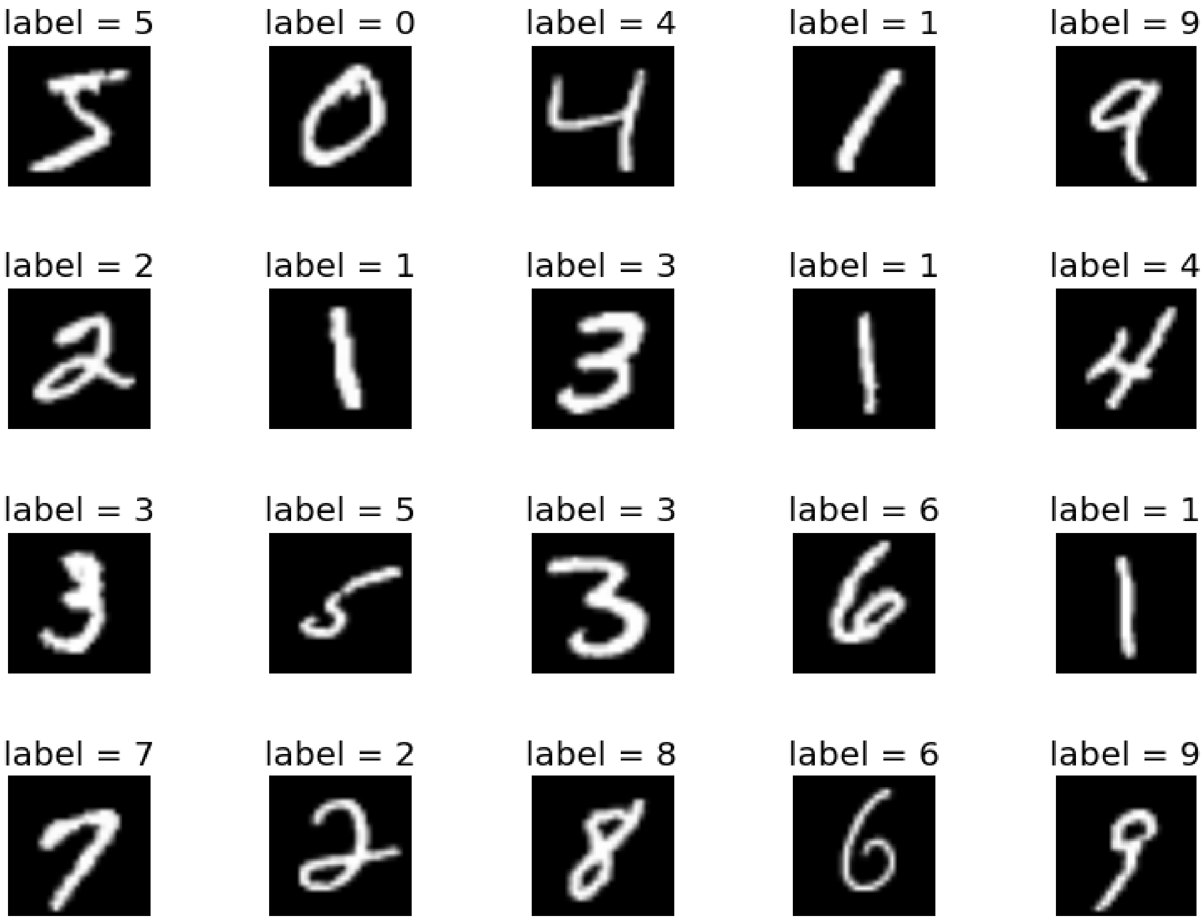
$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

Dropout Modeling



Dropout Performance

- **MNIST : a standard toy data set of handwritten digits**



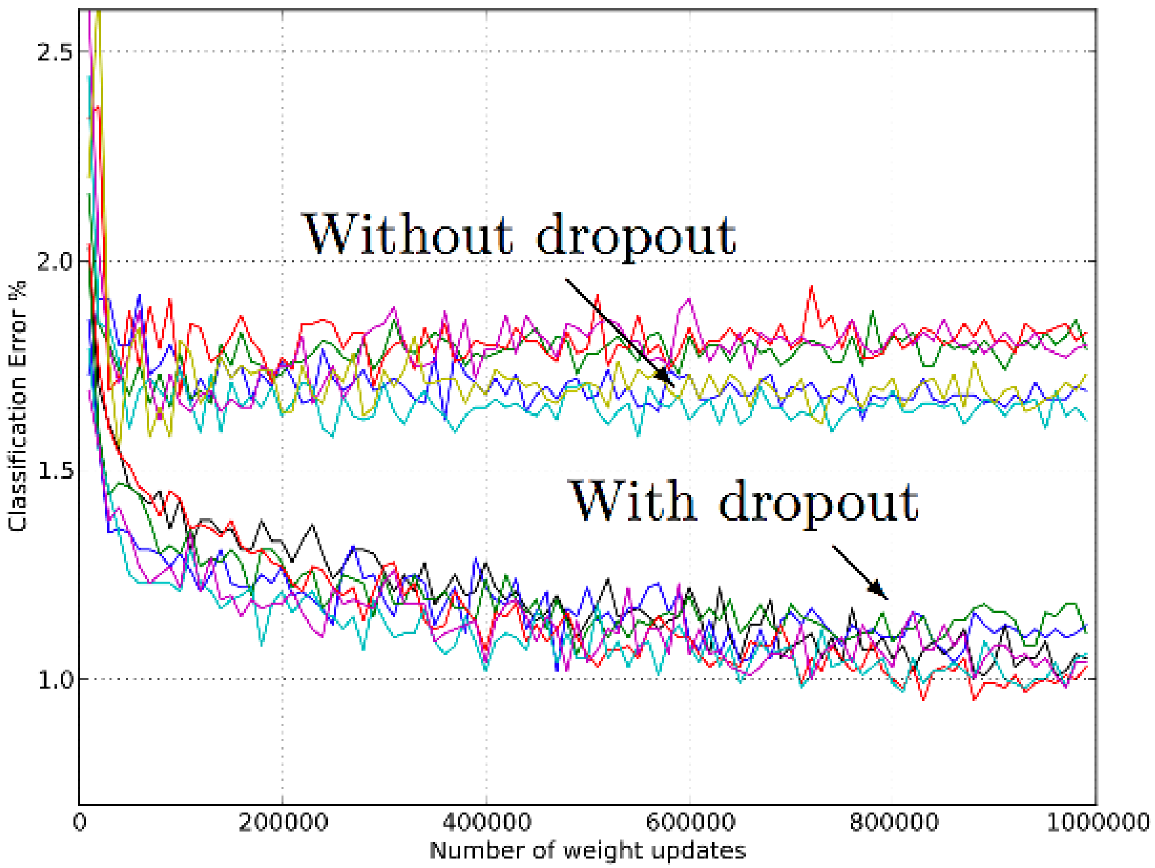
Dropout Performance

- **MNIST results**

Method	Unit Type	Architecture	Error %
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5×240) units	0.94

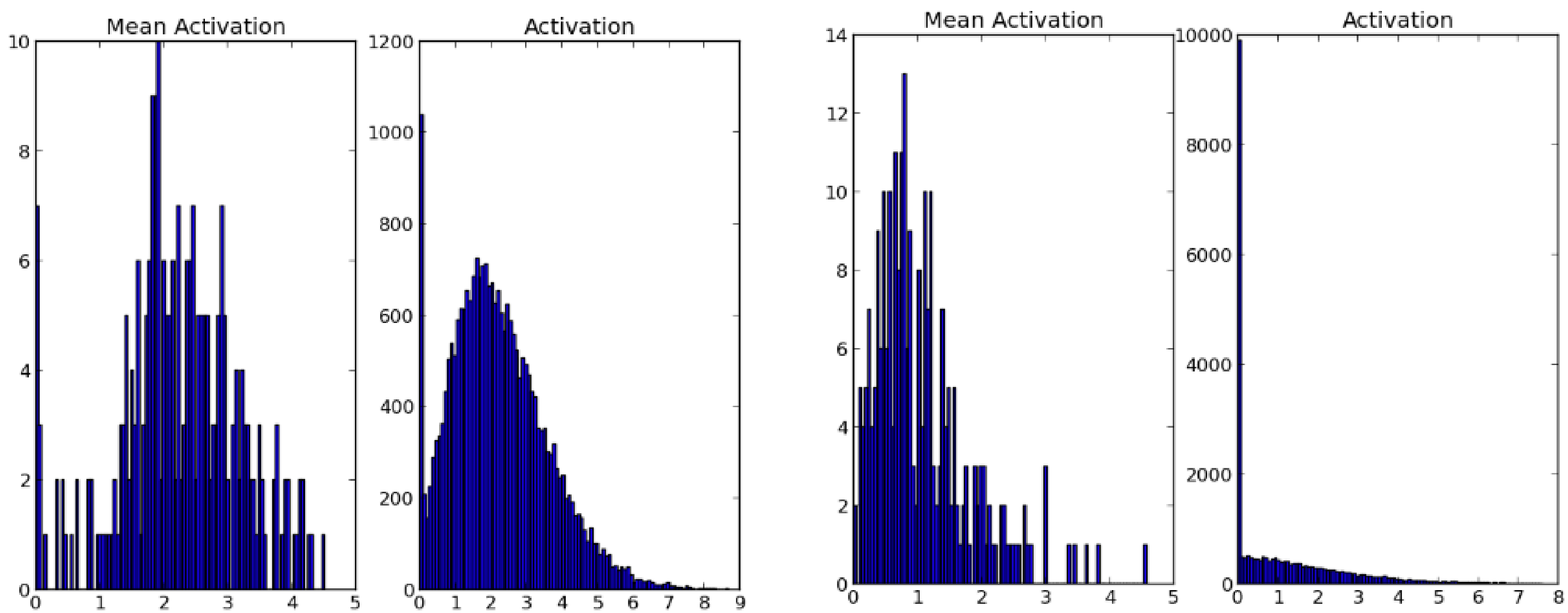
Dropout Performance

- **MNIST results**
  - With fixed dropout rate, for different architectures



Dropout another effect: Sparsity

- **Can capture salient features**
  - Makes neurons more sparse
  - Prevent co-adaptation



(a) Without dropout

(b) Dropout with $p = 0.5$.

Hyper parameter p: the Dropout rate

- **(a) fixed number of neurons, variable p**
  - Relatively constant test error on 0.4~0.8 → usually use $p = 0.5$
- **(b) fixed value of pn**
  - Lower test error on low p → increase number of neurons for low p

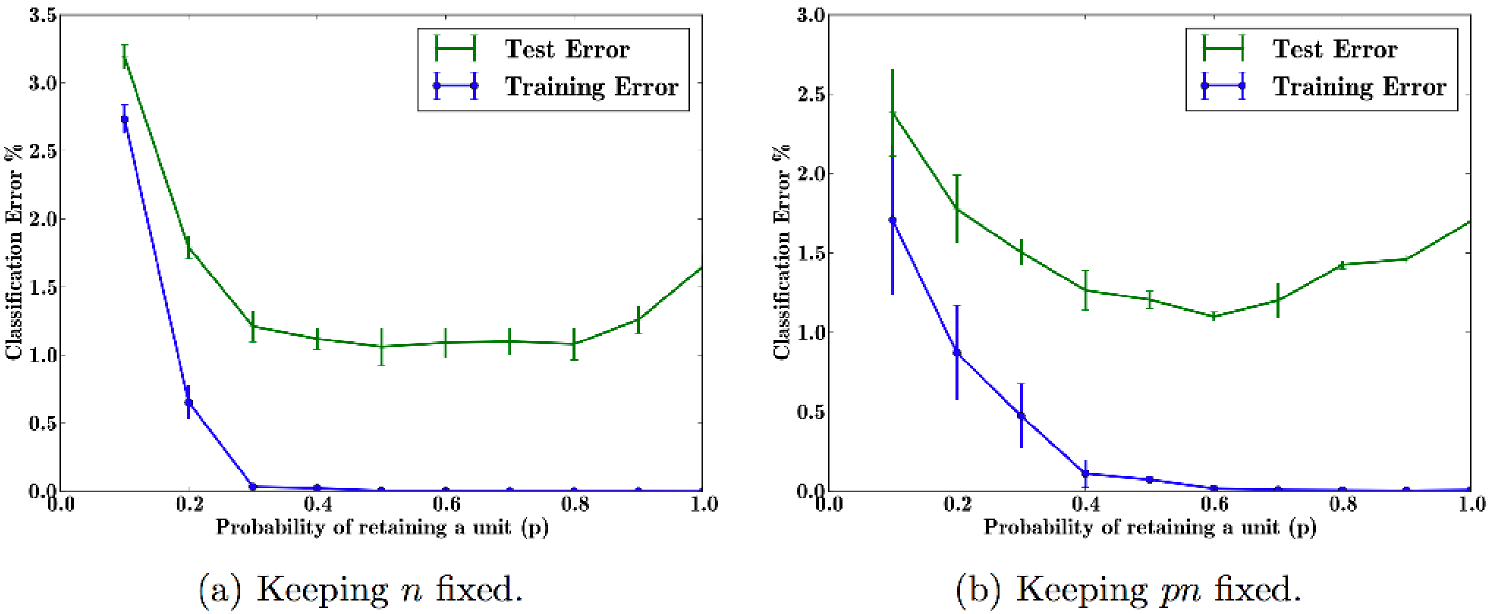
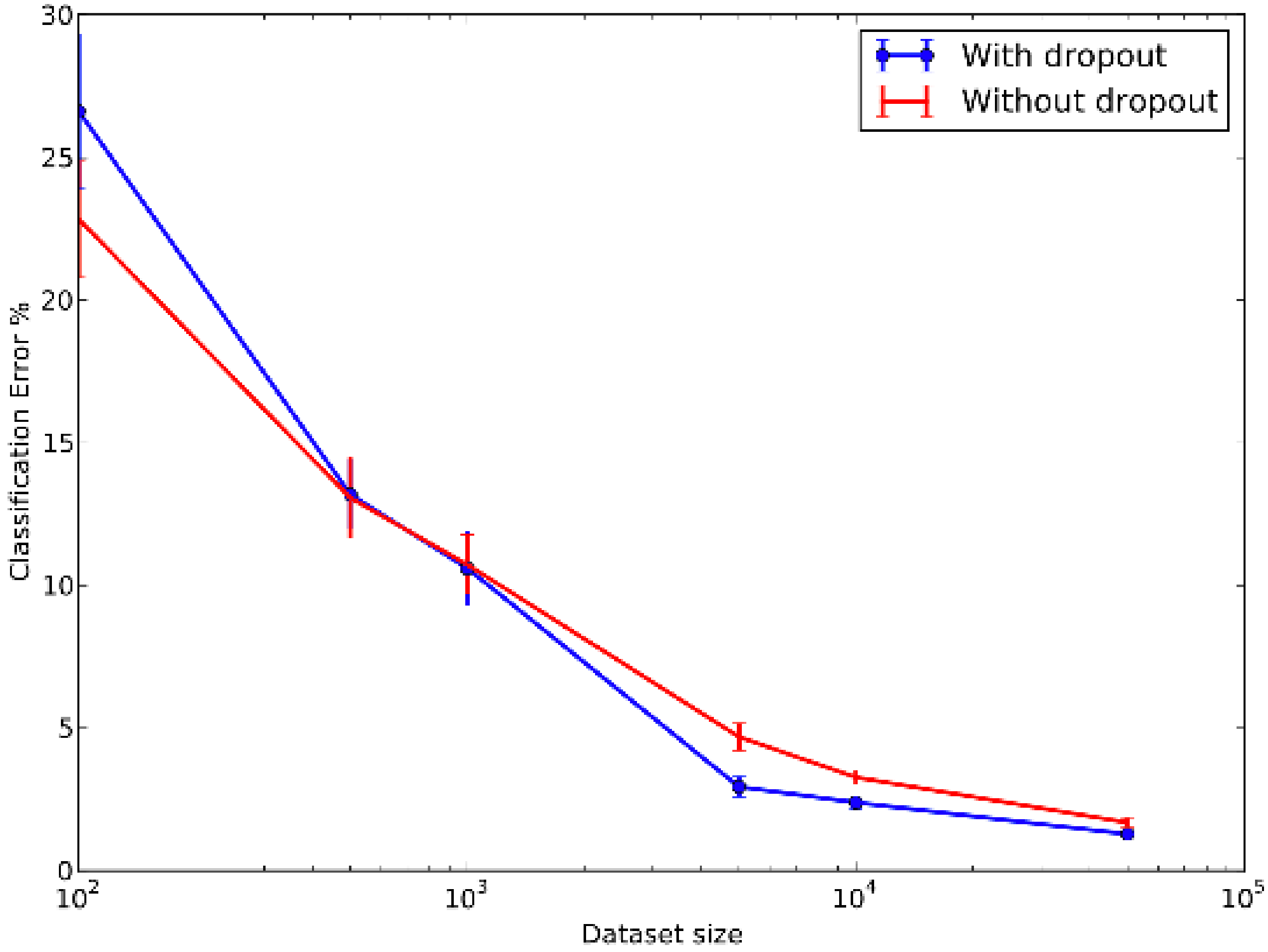


Figure 9: Effect of changing dropout rates on MNIST.

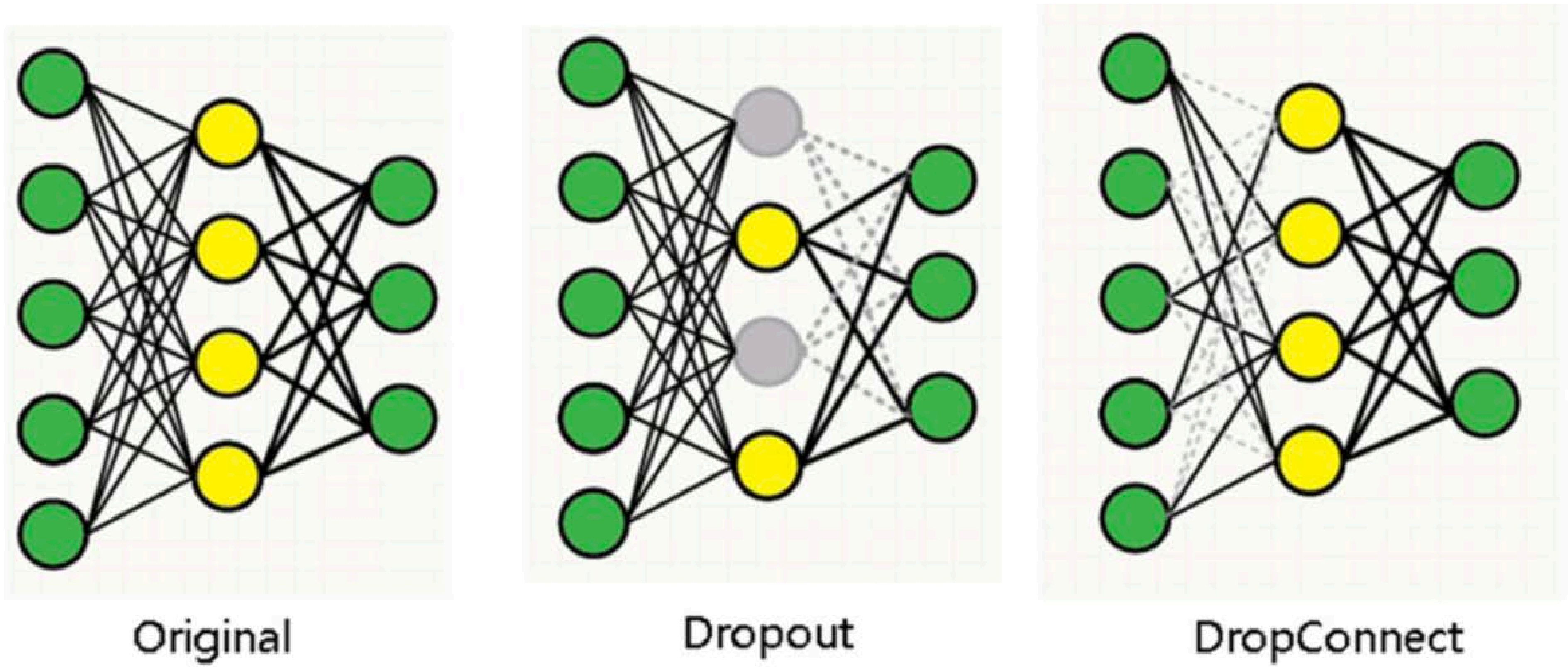
Effect of data set size on Dropout

- **Dropout is more powerful for larger dataset**



DropConnect

- **A generalization of Dropout [4]**
  - Dropout omit the all connections which related to the unit
  - DropConnect only omit the connections and leave the unit alive



DropConnect Modeling

- **Notation**
  - $h_i(I)$ is output value of a neuron
  - w_{ij} is weight of connections j to i
  - I_j is input of a neuron
- **The standard neural networks**

$$h_i(I) = \sum_{j=1}^{n} w_{ij} I_j$$

- **Dropout neural networks**

$$h_i(I) = \sum_{j=1}^{n} w_{ij} r_j I_j$$

- **DropConnect neural networks**

$$h_i(I) = \sum_{j=1}^{n} r_{ij} w_{ij} I_j$$

DropConnect Performance

neuron	model	error(%) 5 network	voting error(%)
relu	No-Drop	1.62 ± 0.037	1.40
	Dropout	1.28 ± 0.040	1.20
	DropConnect	1.20 ± 0.034	**1.12**
sigmoid	No-Drop	1.78 ± 0.037	1.74
	Dropout	1.38 ± 0.039	**1.36**
	DropConnect	1.55 ± 0.046	1.48
tanh	No-Drop	1.65 ± 0.026	1.49
	Dropout	1.58 ± 0.053	1.55
	DropConnect	1.36 ± 0.054	**1.35**

Adversarial Training

References:
[1] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).
[2] Szegedy, Christian, et al. "Intriguing properties of neural networks." *arXiv preprint arXiv:1312.6199* (2013).

Adversarial Examples

- **Definition**
  - **Applying an imperceptible non-random perturbation to a test image, it is possible to arbitrarily change the network's prediction [2]**
- **Examples**
  - **With the same predict model,**

Y = "panda"

With 0.577 confidence

Y = "gibbon"

With 0.993 confidence

Adversarial Examples

- **Definition**
  - Applying an imperceptible non-random perturbation to a test image, it is possible to arbitrarily change the network's prediction [2]
- **Examples**
  - With the same predict model,

$\times\ 0.07$

$=$

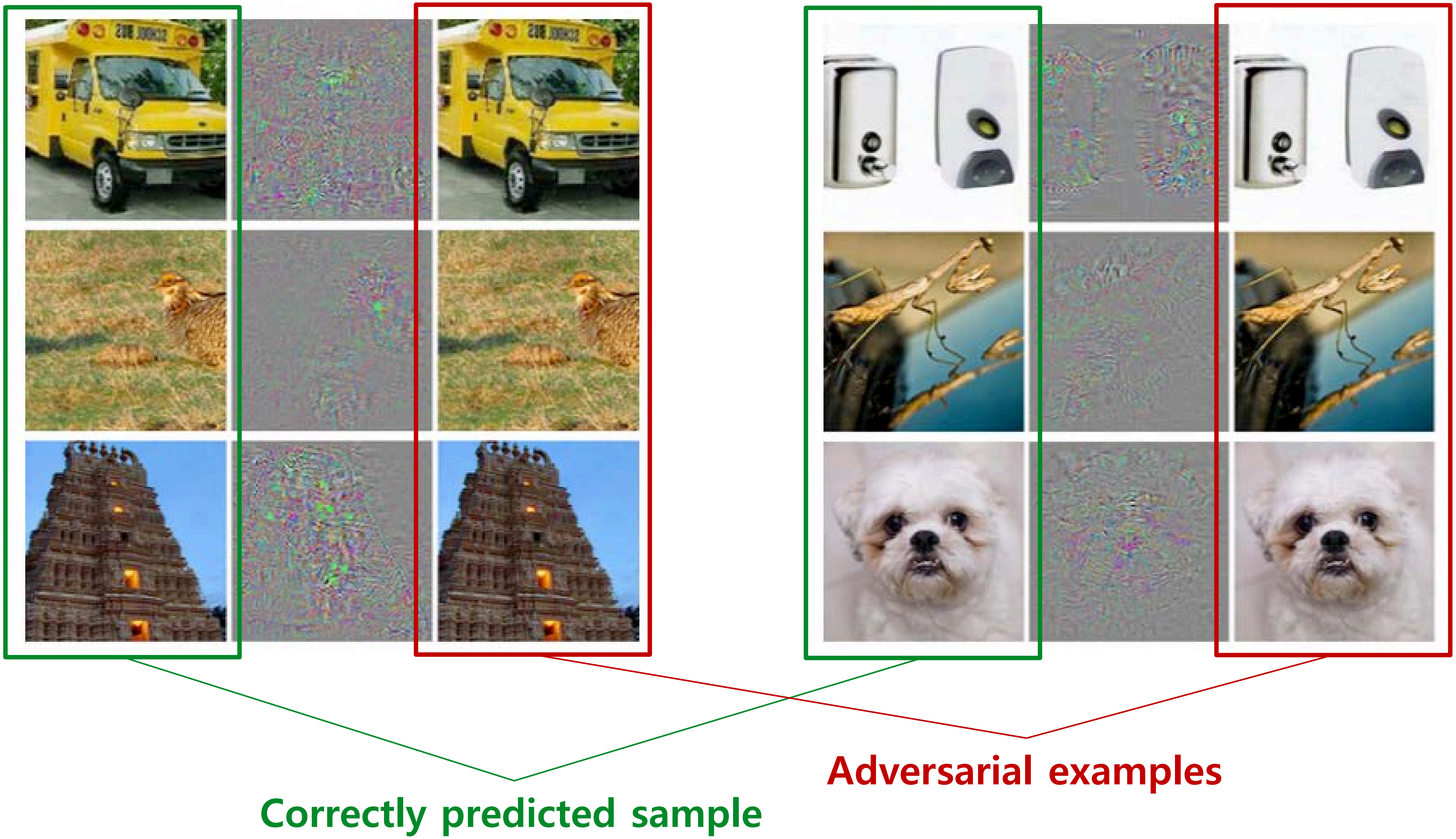
Y = "panda"

With 0.577 confidence

Y = "gibbon"

With 0.993 confidence

Adversarial Examples



Adversarial Training

- **Formal description[2]**

$$Minimize\ \|\eta\|_2 \qquad subject\ to \quad f(x+\eta)=l$$
$$x+\eta \in [0,1]^m$$
$$f(x) \neq l$$

$noise : \eta$
$label : l$

- **More general description for neural network training[1]**

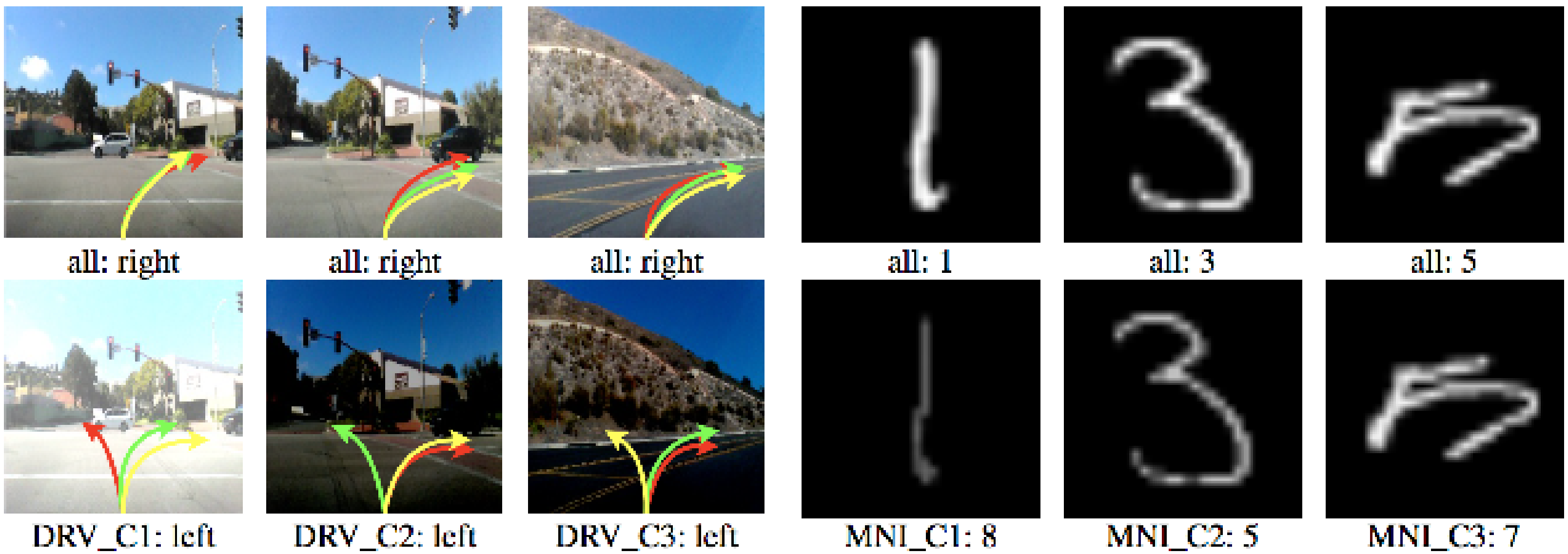
Training as a regularization term

$$\tilde{J}(\theta,x,y) = \alpha J(\theta,x,y) + (1-\alpha)\boxed{J(\theta,x+\eta)}$$

$$where, \qquad \eta = \epsilon \mathrm{sign}(\nabla_x J(\theta,x,y))$$

Current research

Different lighting conditions:



The next Deep Learning Seminar

Chapter 8. Optimization for Training Deep Models

Thank you

Any Questions?