CS60010: Deep Learning

Sudeshna Sarkar

Spring 2018

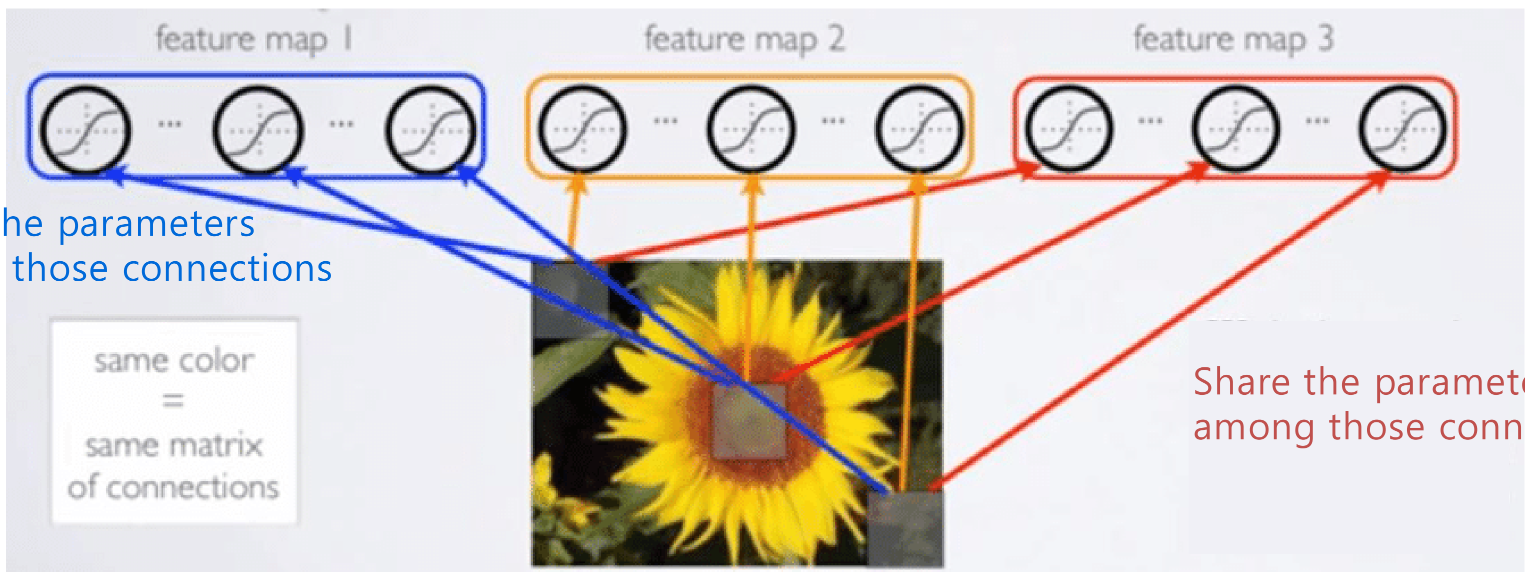
24 Jan 2019

Part 2

REGULARIZATION

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



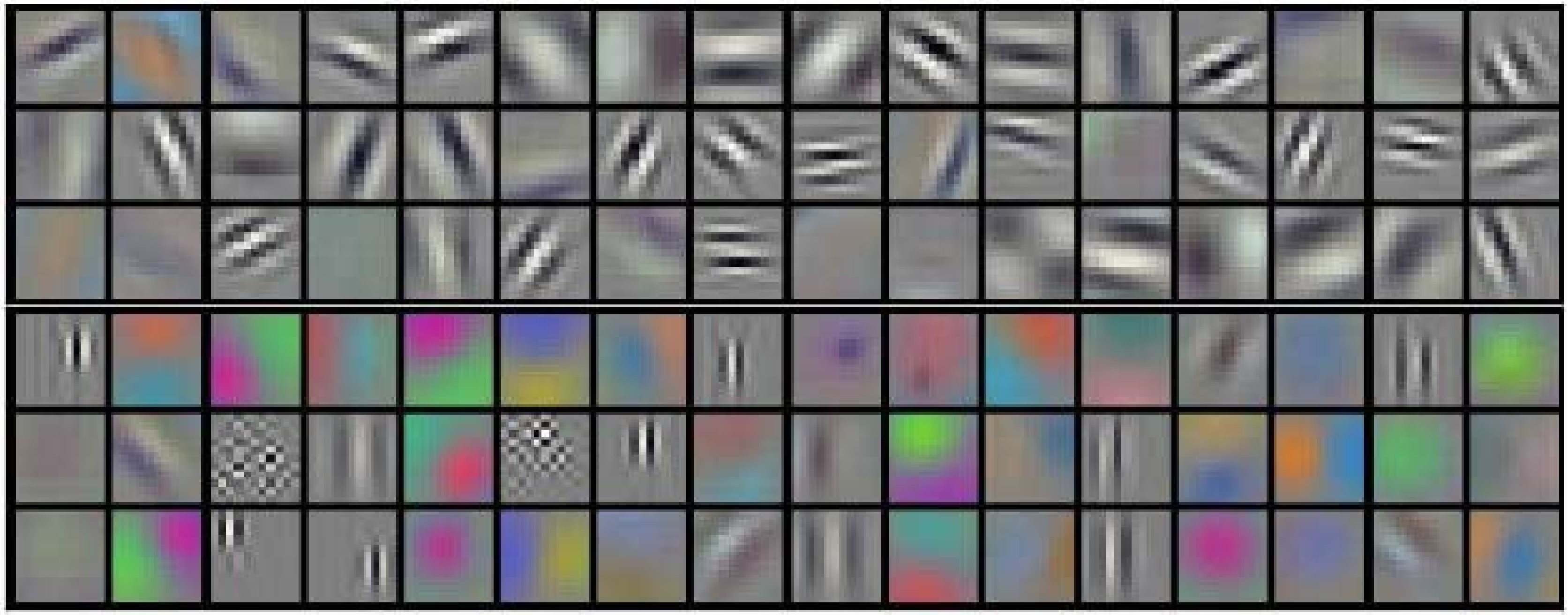
Share the parameters
among those connections

Share the parameters
among those connections

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

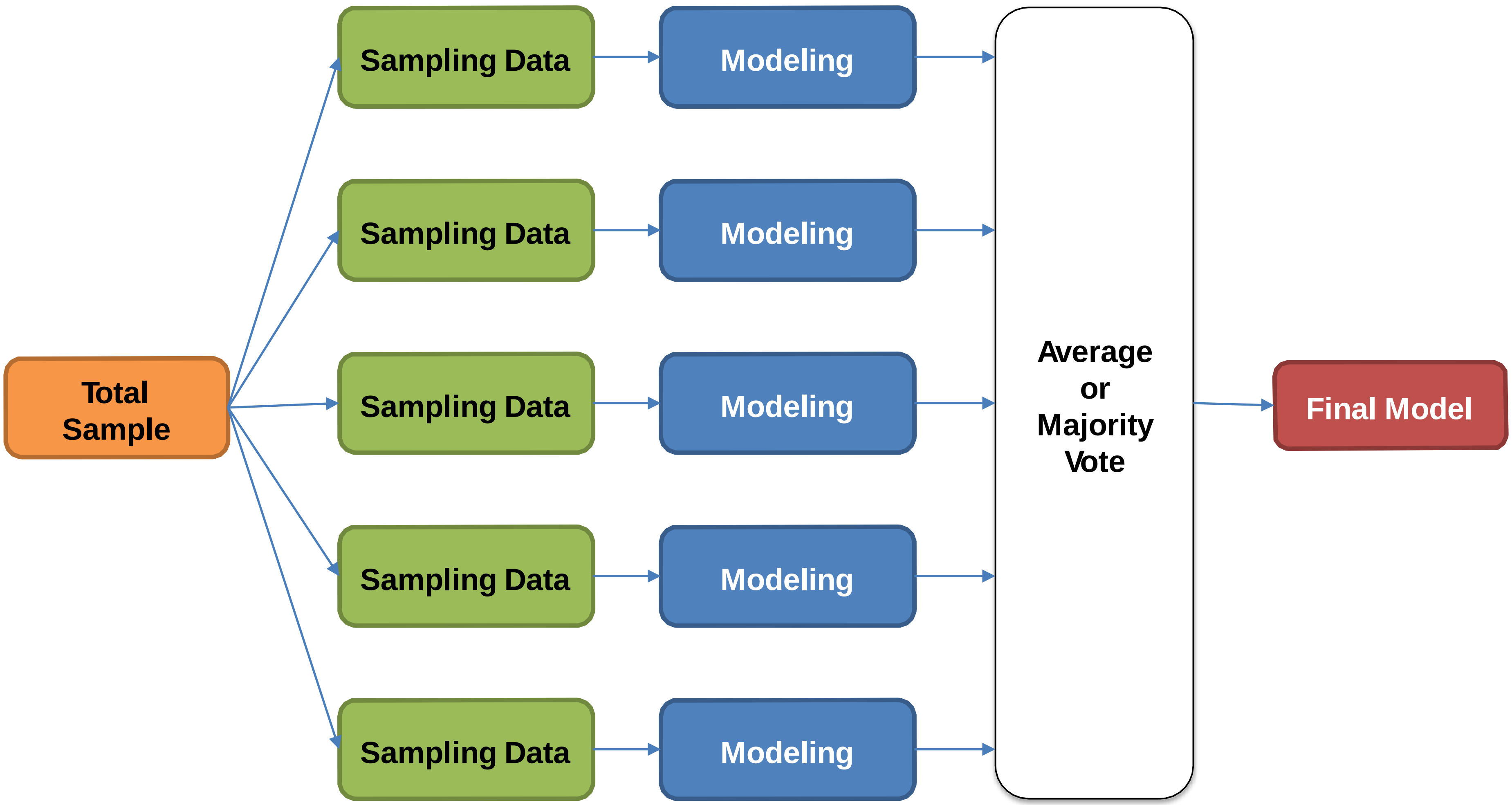
Example of CNN filters

96 filters from AlexNet



BAGGING AND OTHER ENSEMBLE METHODS

The Concept of Bagging



Bagging

Boostrap aggregation: Reduce generalization error by combining models

Procedure

- Split the input data to K clusters with N' examples
- 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 does Bagging work?

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

  Variance $v = E[{\epsilon_i}^2]$ Covariance $c = E[\epsilon_i \epsilon_j]$
- 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

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

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

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

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 $\frac{1}{k}v$

Bagging

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

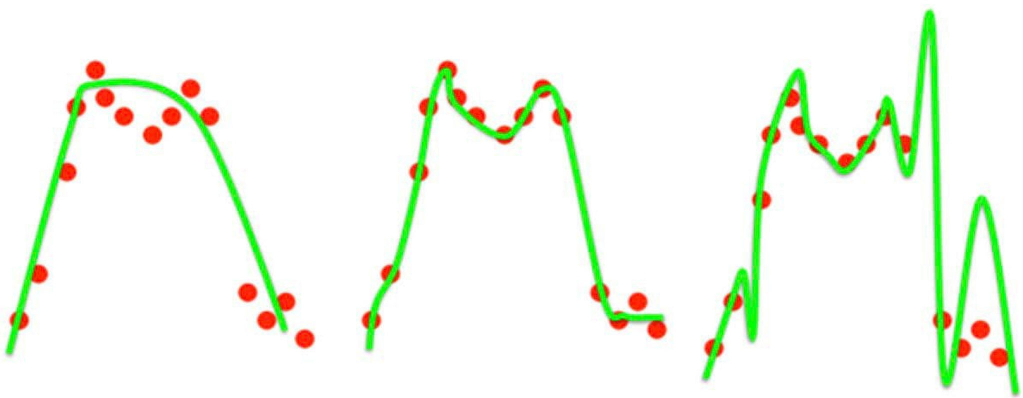
Dropout

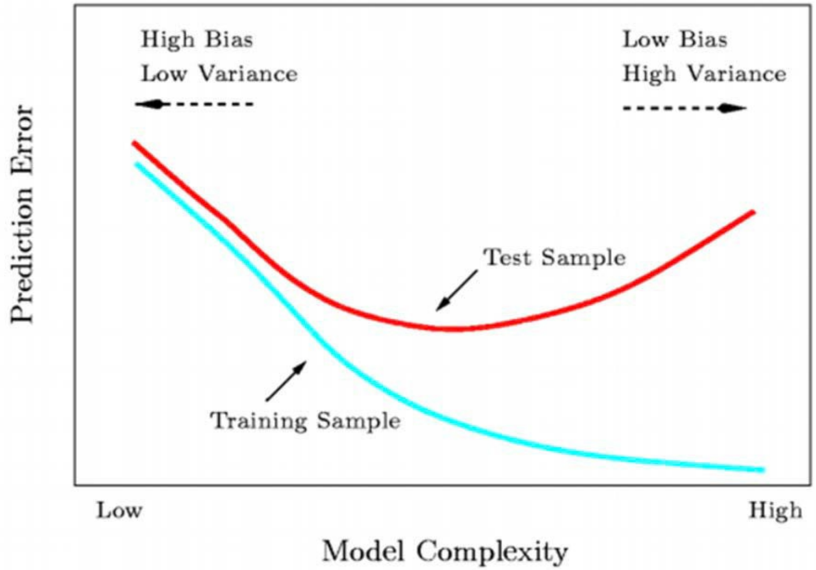
References:

1. [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. [2] Hinton, Geoffrey E., et al., "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).
3. [3] Krizhevsky, Alex et al., "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
4. [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**

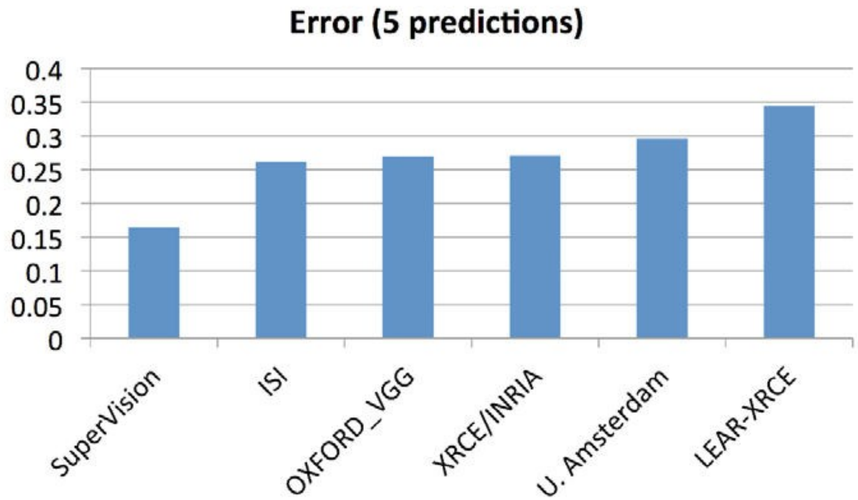
Batch Normalization (2015)

Research in Dropout

First proposed by G.E. Hinton (2012)

Became popular by *AlexNet* (2012)

- 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)

- Strengthen the theoretical background, extend to convolutional layer

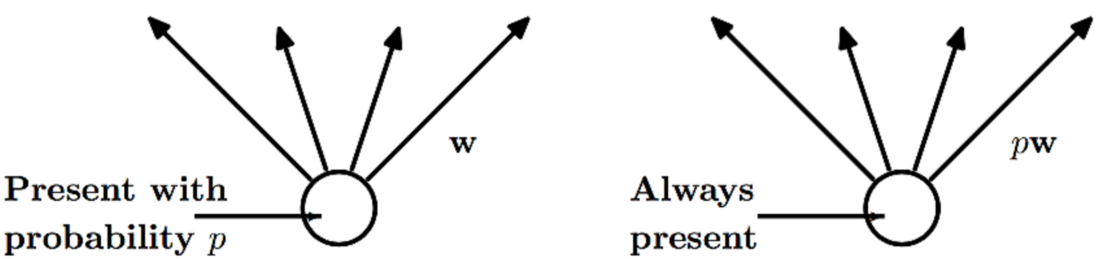
Dropout

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

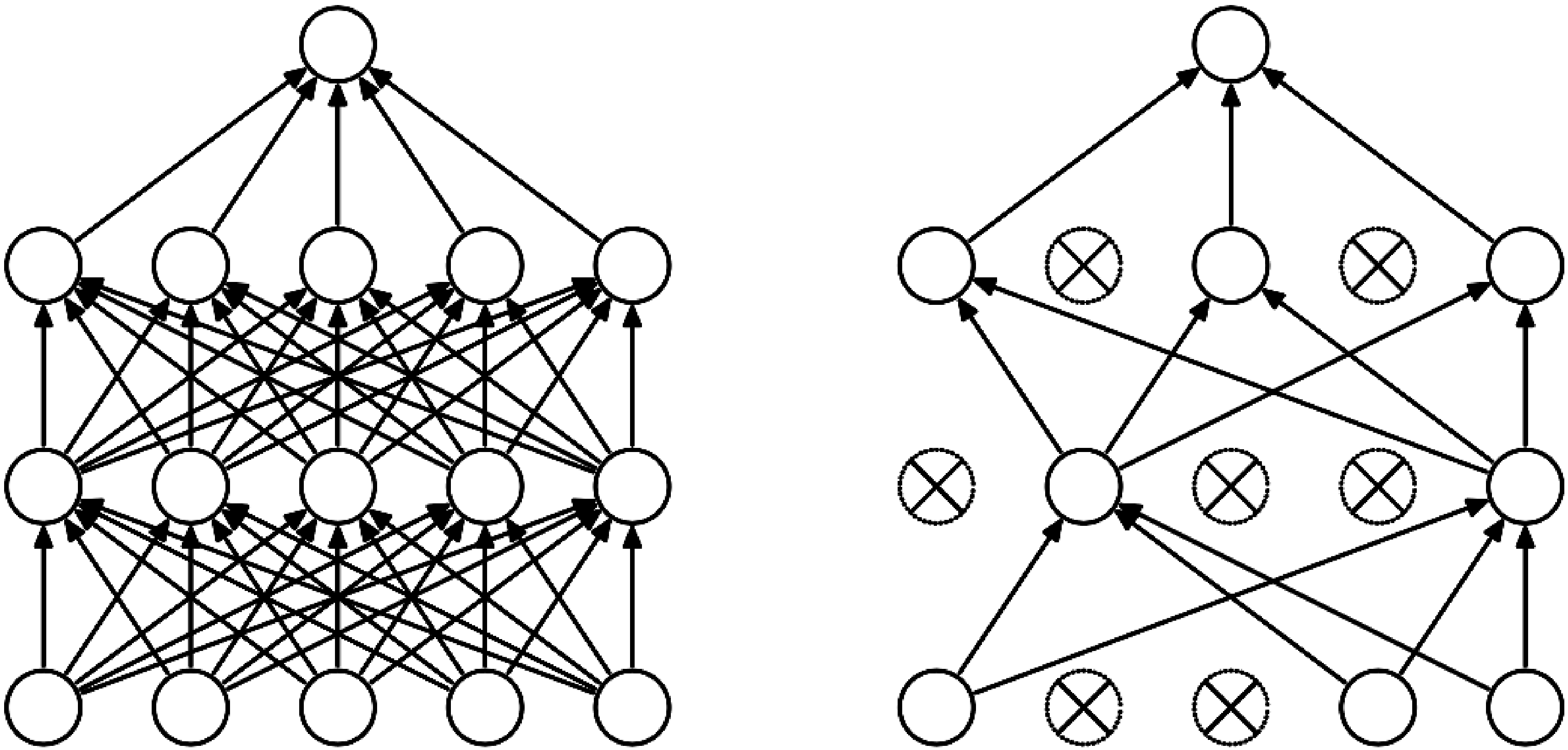
Dropout addressed the two problems

- 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



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

Dropout

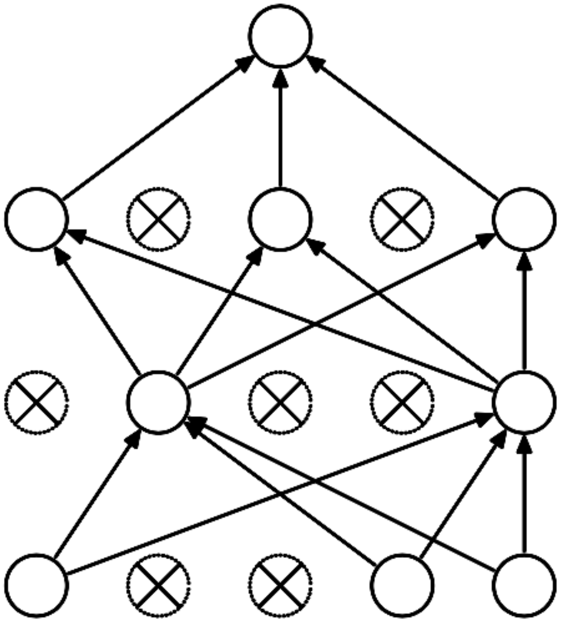


(a) Standard Neural Net　　(b) After applying dropout.

Effect of Dropout

Avoid the co-adaptation

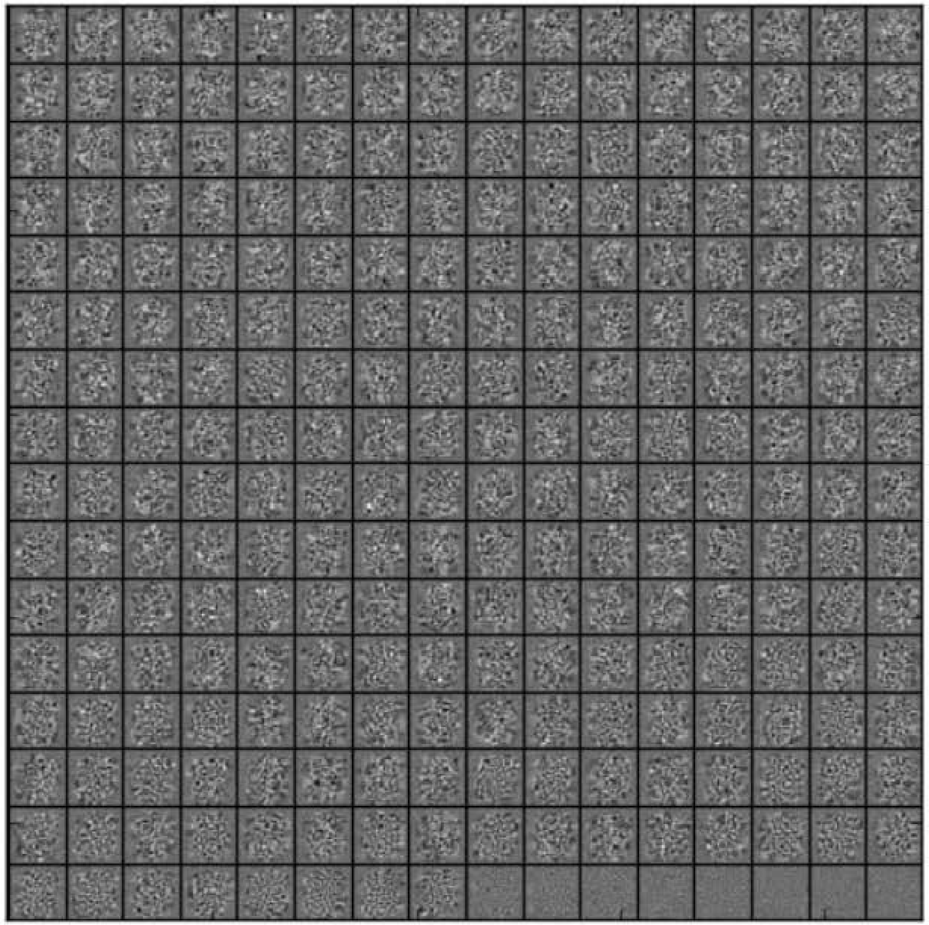
- Co-adaptation: the trend that some neurons tend to represent similar features
- Capture the clear features by avoiding co-adaptation



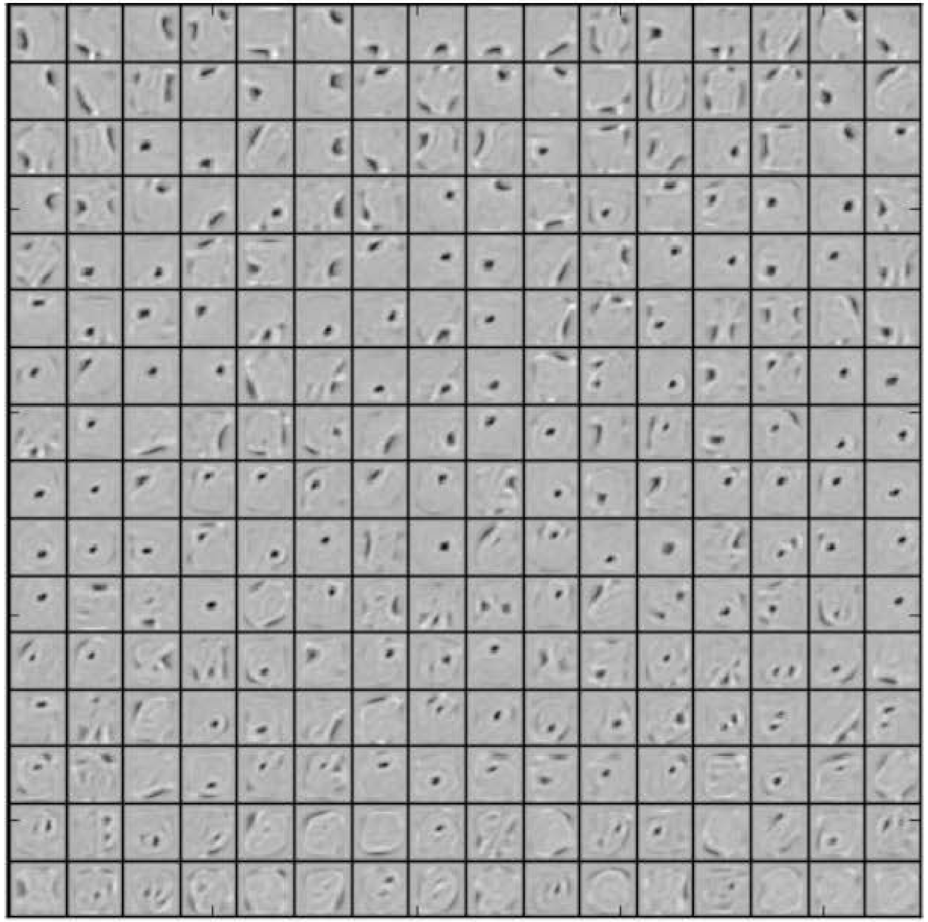
(b) After applying dropout.

Effect of Dropout

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

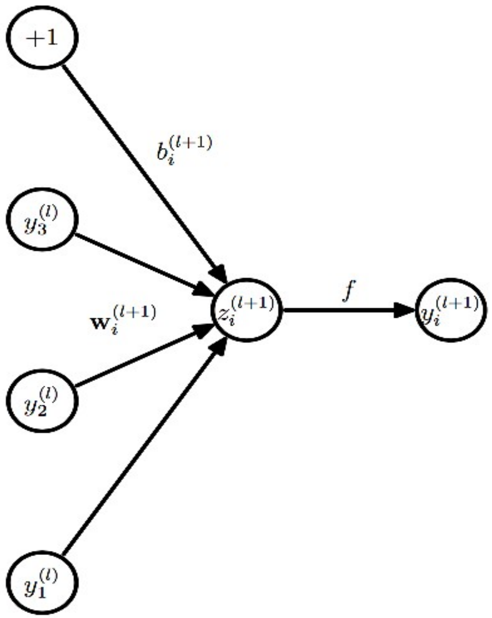


(a) Without dropout



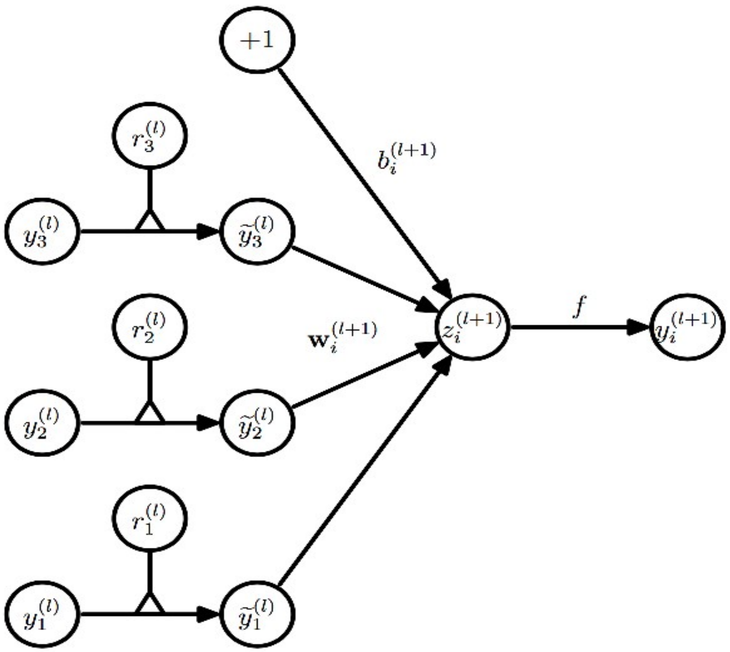
(b) Dropout with $p = 0.5$.

Dropout Modeling



(a) Standard network

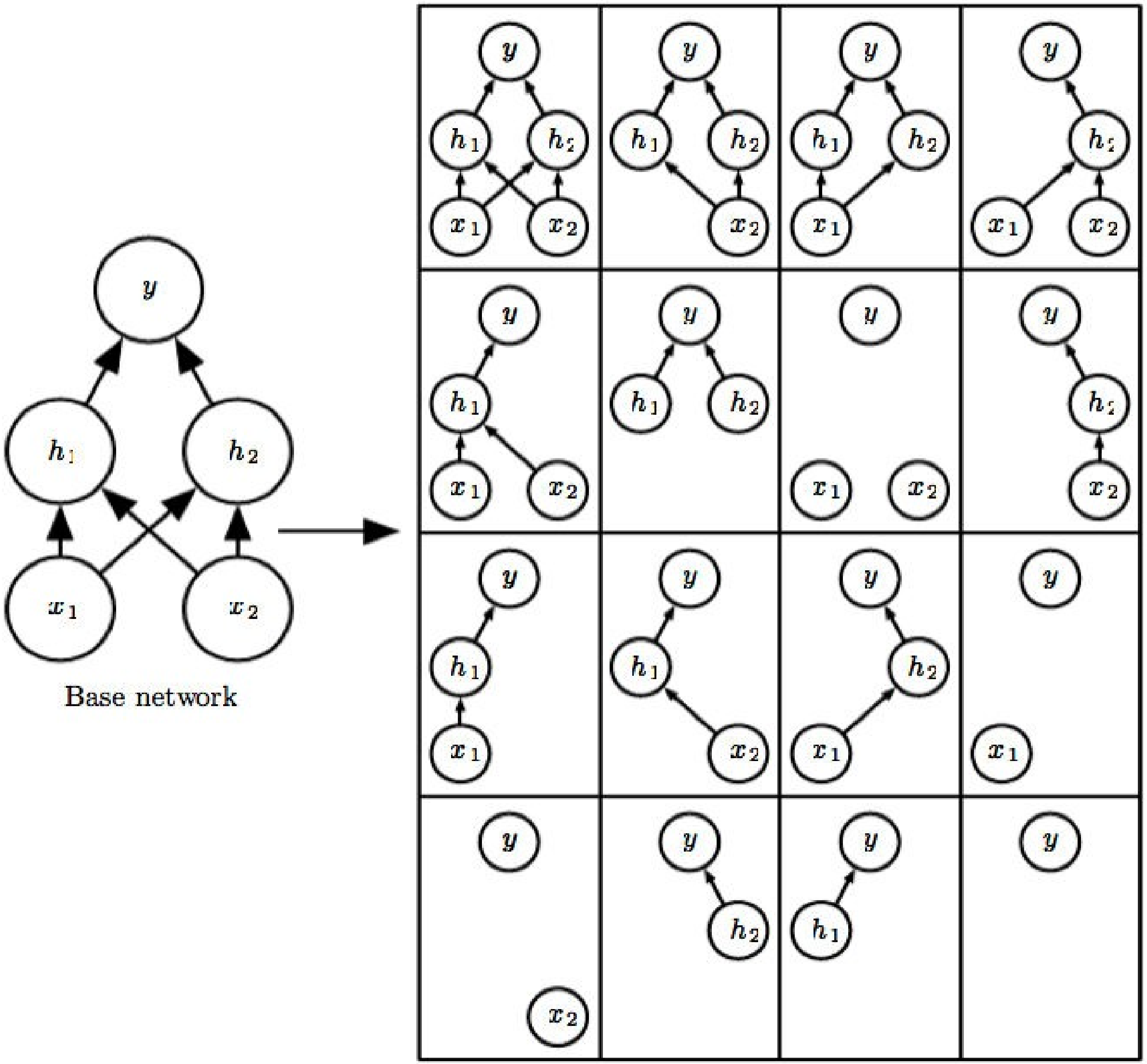
$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$



(b) Dropout network

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

Dropout Modeling

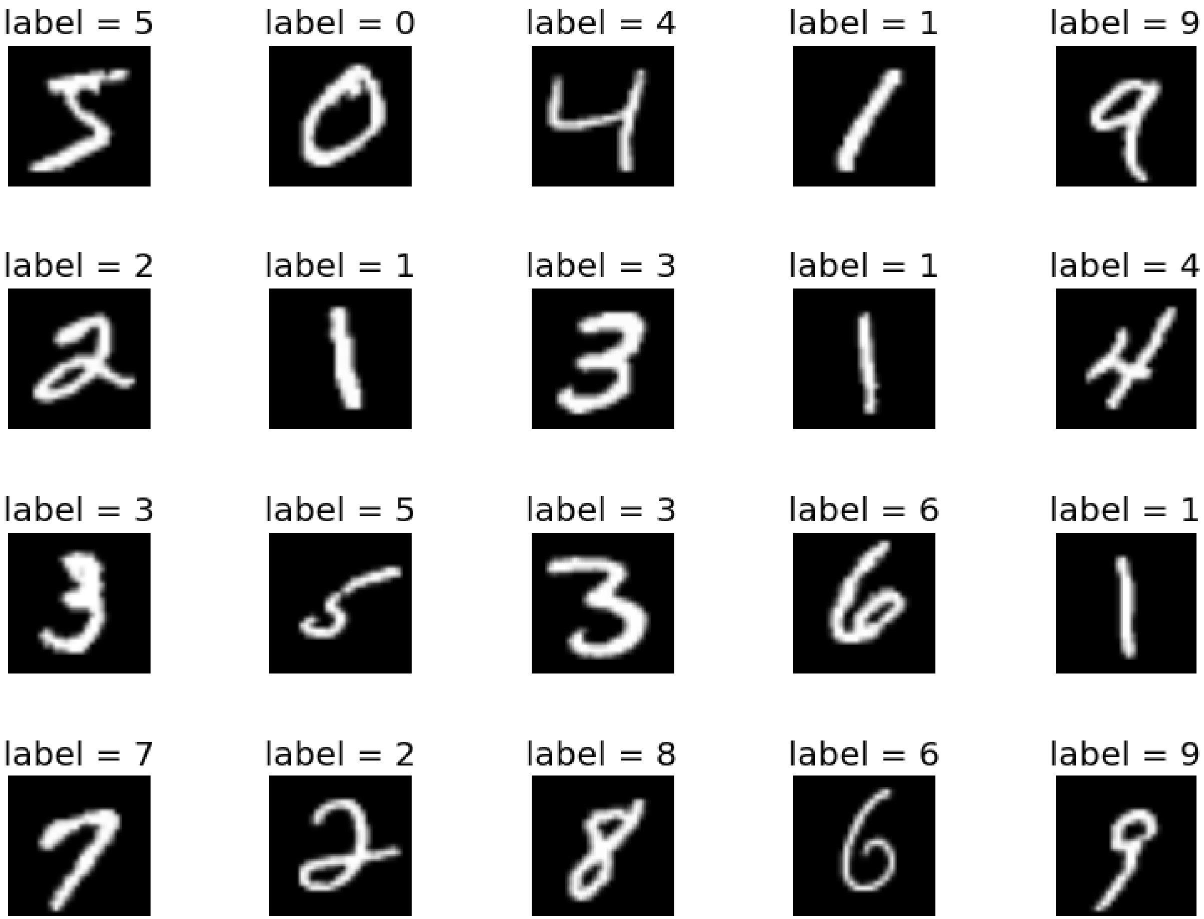


Base network

Ensemble of Sub-Networks

Dropout Performance

MNIST : a standard toy data set of handwritten digits



Dropout Performance

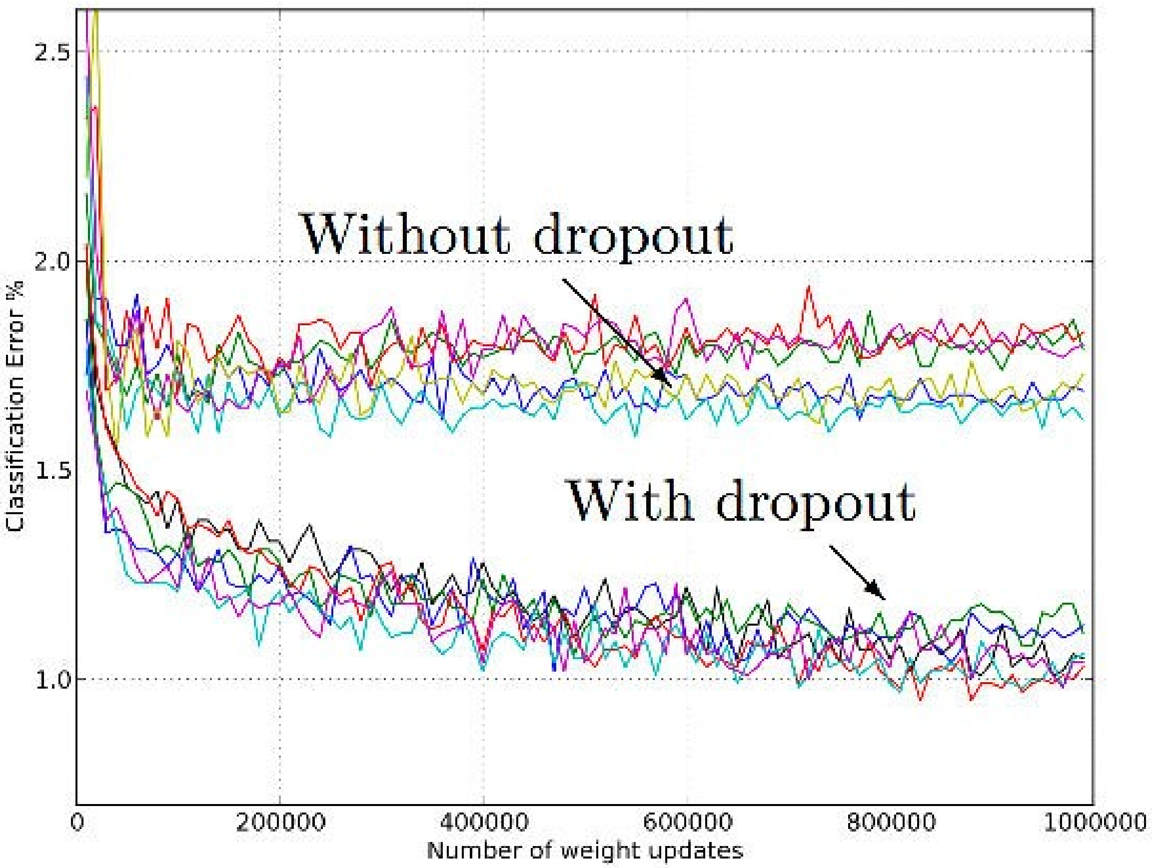
MNIST results

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

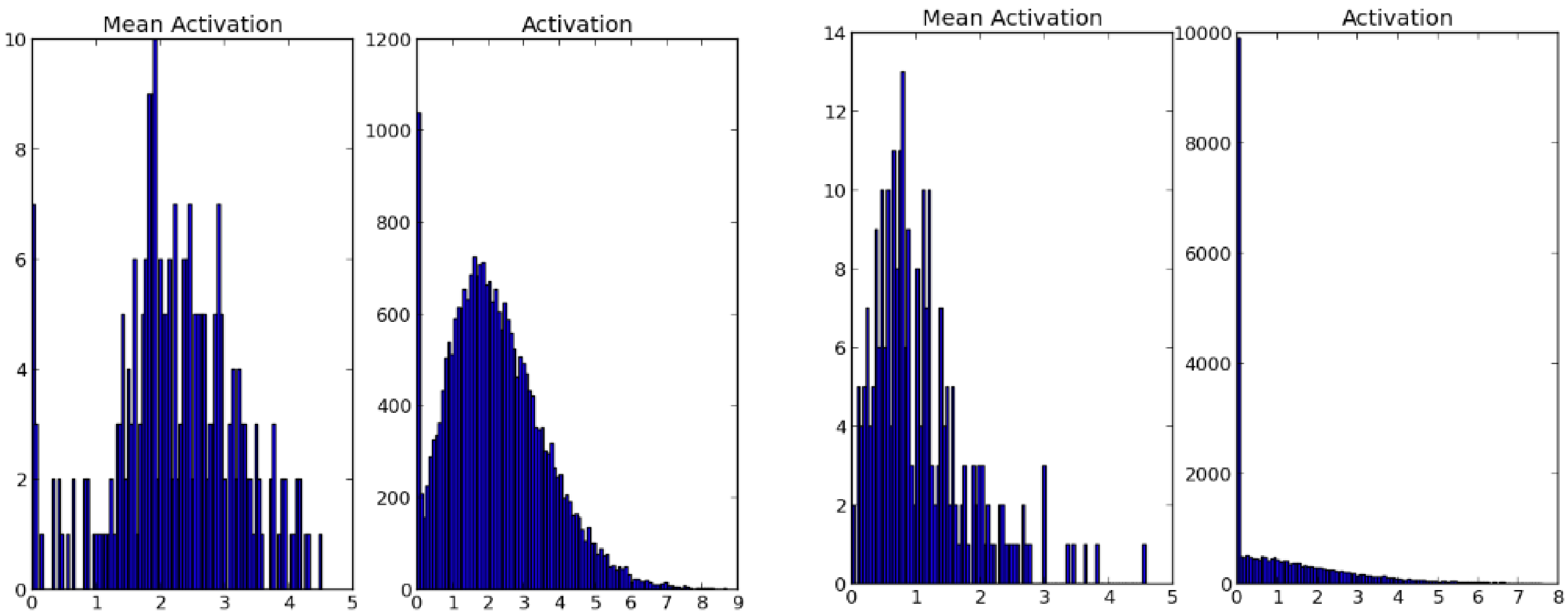
- With fixed dropout rate, for different architectures



Dropout another effect: Sparsity

Can capture salient features

- Makes neurons more sparse
- Prevent co-adaptation



(a) Without dropout

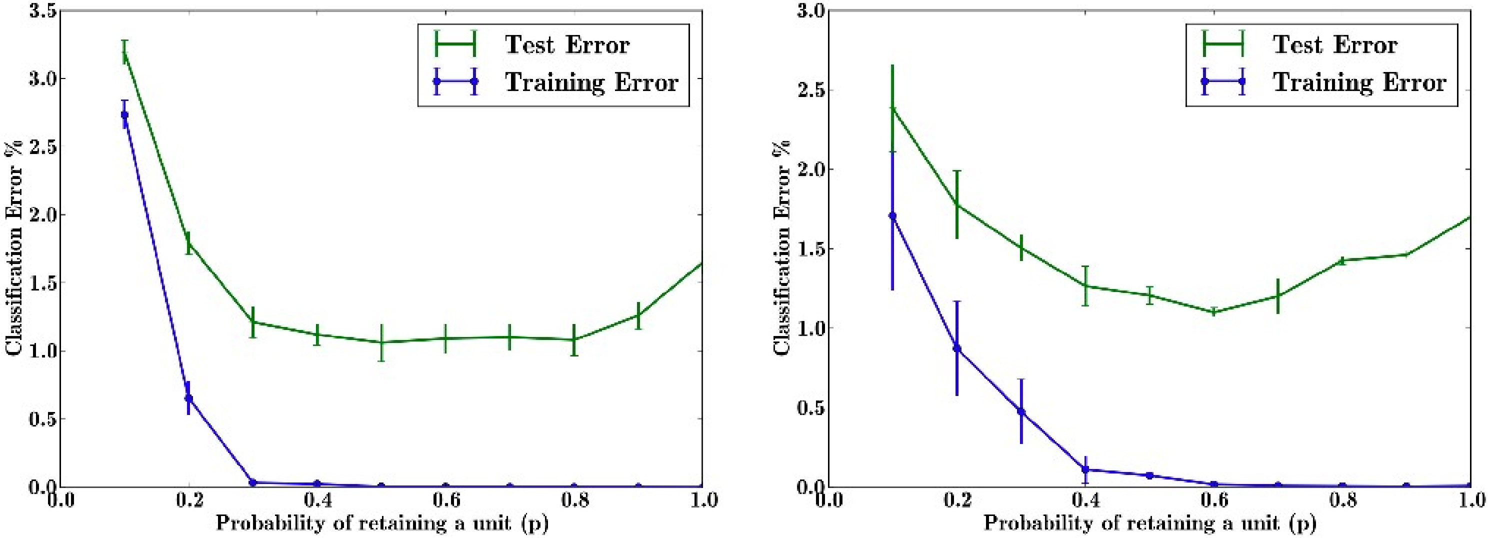
(b) Dropout with $p = 0.5$.

Hyper parameter p: the Dropout rate

(a) fixed number of neurons, variable $\boldsymbol{p}$
(b) Relatively constant test error on 0.4~0.8   usually use p=0.5

(c) fixed value of p

- Lower test error on low p -- > increase number of neurons for low p

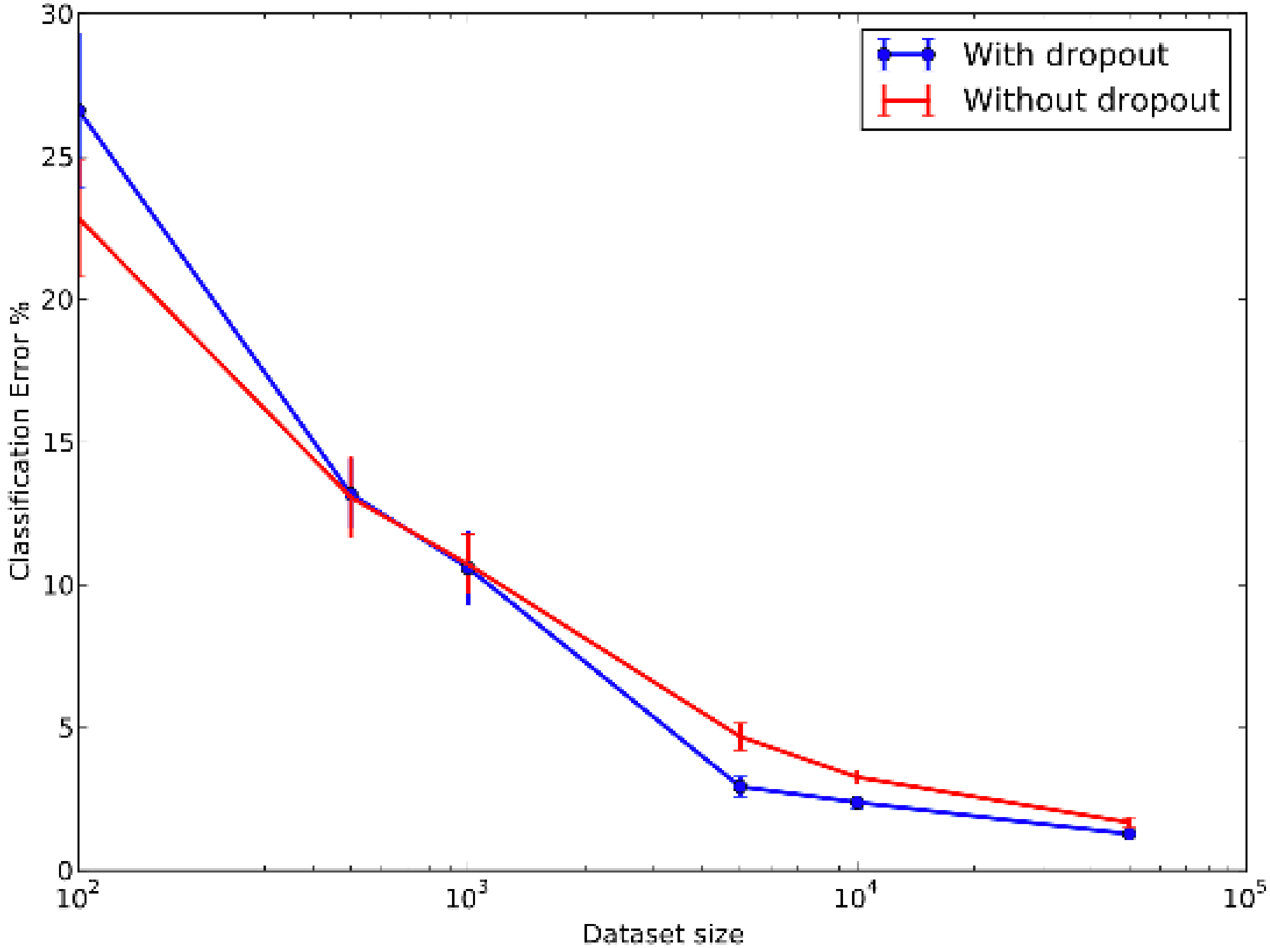


(a) Keeping n fixed.   (b) Keeping pn fixed.

Figure 9: Effect of changing dropout rates on MNIST.

Effect of data set size on Dropout

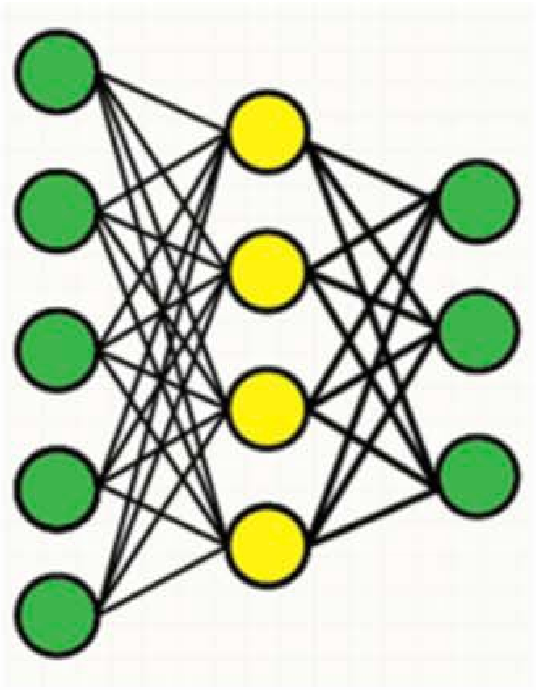
Dropout is more powerful for larger dataset



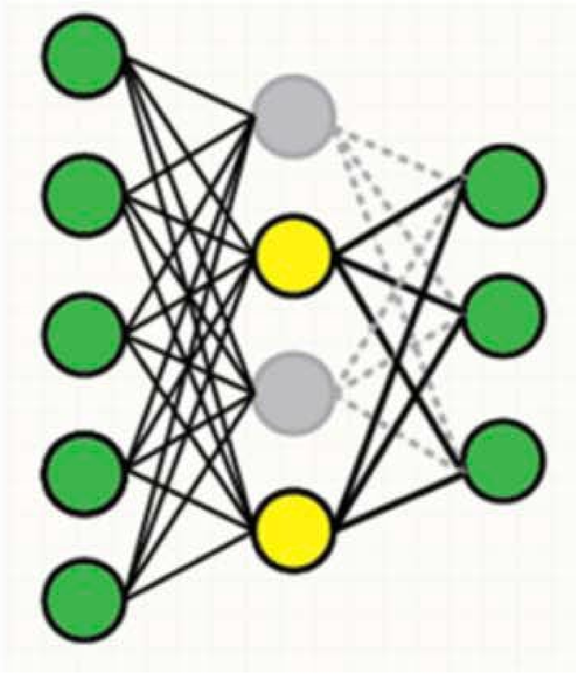
DropConnect

A generalization of Dropout

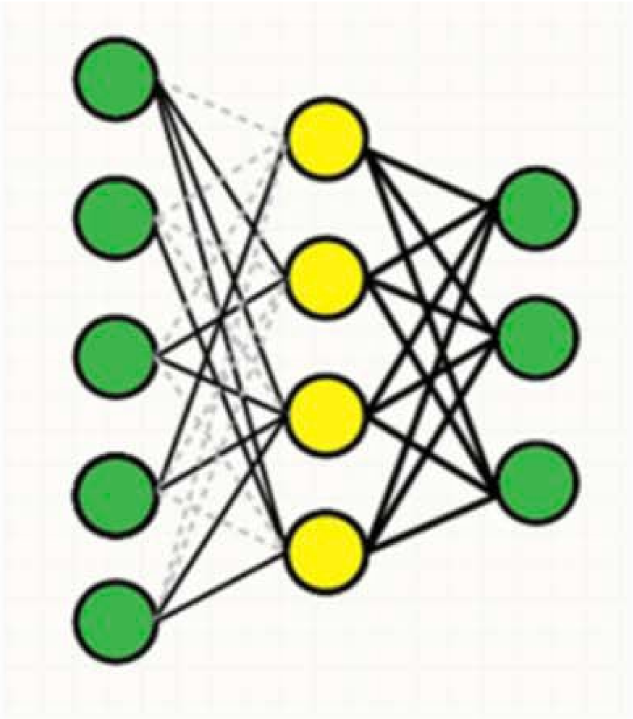
- Dropout omit the all connections which related to the unit
- DropConnect only omit the connections and leave the unit alive



Original



Dropout



DropConnect

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

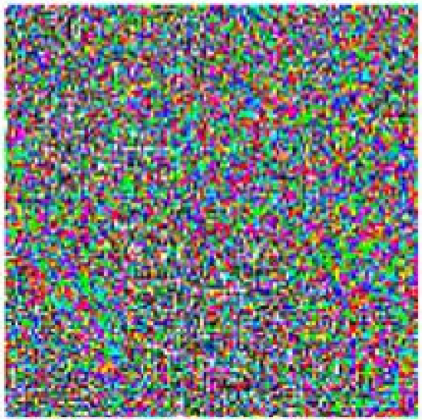
Definition

- **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,**

$\times 0.07$



$=$

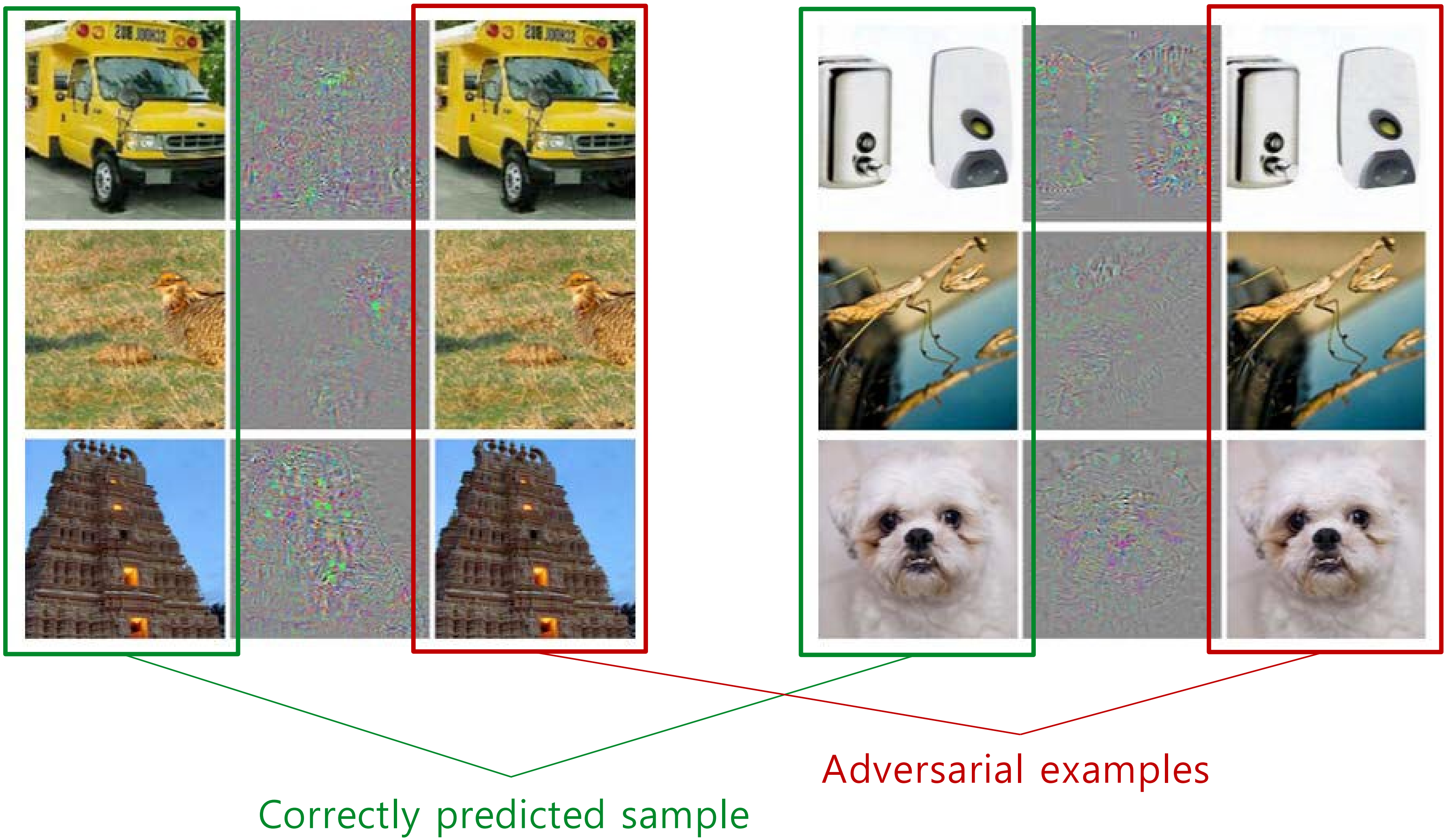
Y = "panda"

With 0.577 confidence

Y = "gibbon"

With 0.993 confidence

Adversarial Examples



Adversarial Training

Formal description[2]

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

$noise:\ \eta$
$label:\ l$

More general description for neural network training[1]

Training as a regularization term

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

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

CS60010: Deep Learning

Sudeshna Sarkar

Spring 2019

OPTIMIZATION

30 Jan 2019

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1. **while** stopping criteria not met **do**
2. Sample example $(x^{(i)}, y^{(i)})$ from training set
3. Compute gradient estimate:
4. $\hat{g} \leftarrow +\nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$
5. Apply Update: $\theta \leftarrow \theta - \epsilon\hat{g}$
6. **end while**

ϵ_k is learning rate at step k

Sufficient convergence:

$\sum_{k=1}^{\infty} \epsilon_k = \infty$ and $\sum_{k=1}^{\infty} {\epsilon_k}^2 < \infty$

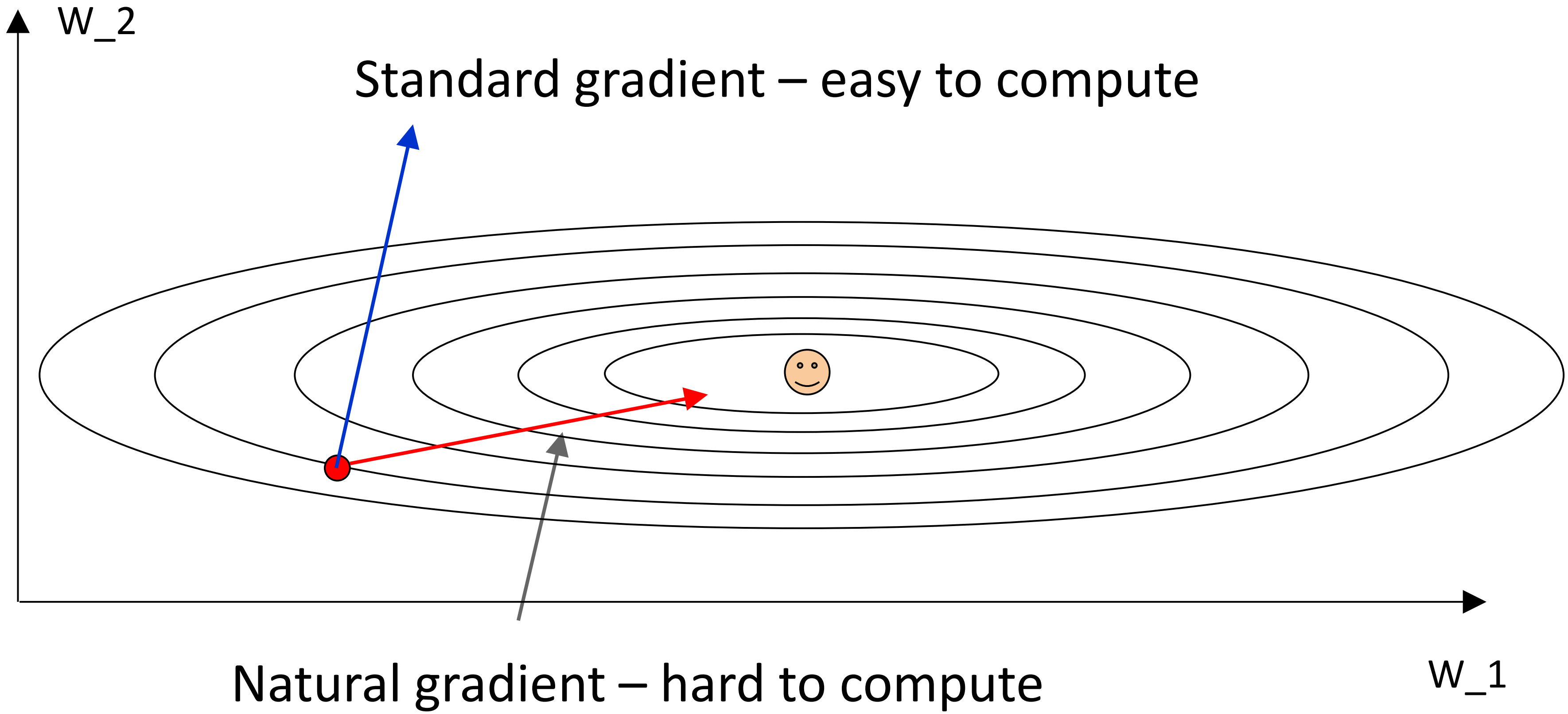
In practice the learning rate is decayed linearly till iteration τ

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\, \epsilon_\tau \; where \;\; \alpha = \frac{k}{\tau}$$

Activation / Gradient distributions per layer

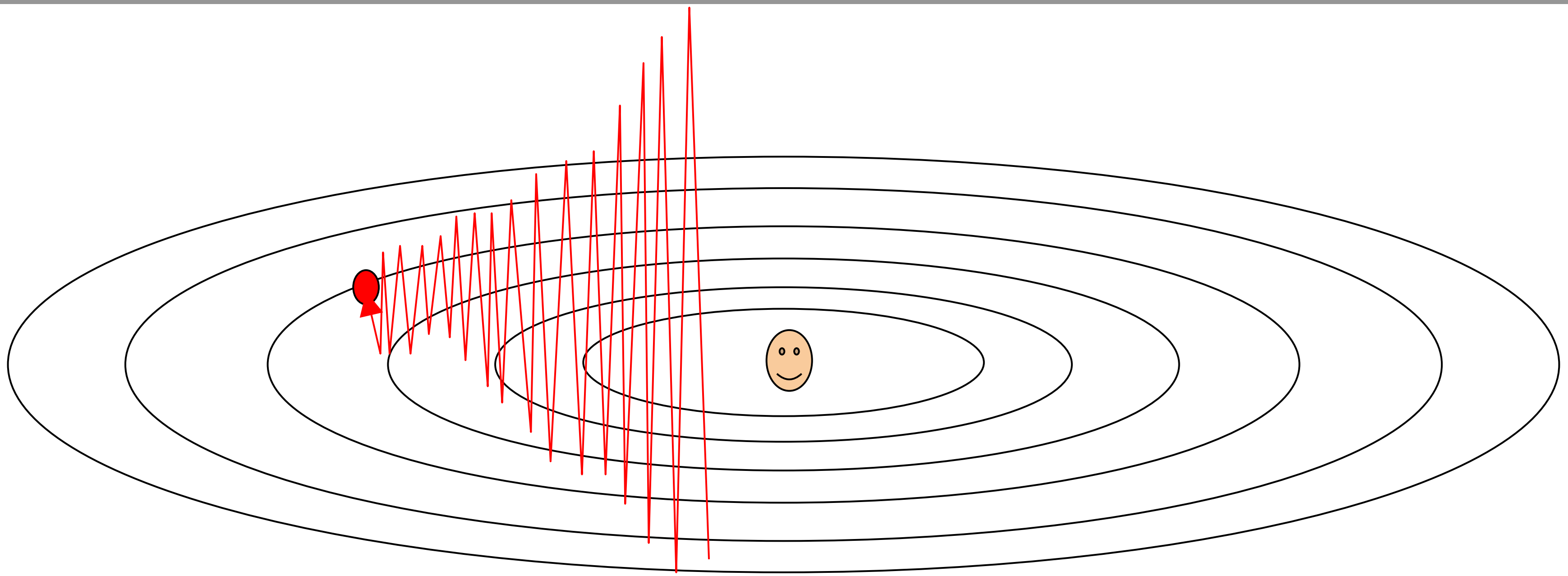
- An incorrect initialization can slow down or even completely stall the learning process
- Plot activation/gradient histograms for all layers of the network.

Gradients



Natural gradient : Rather than treating a change in every parameter equally, we need to scale each parameter's change according to how much it affects our network's entire output distribution.

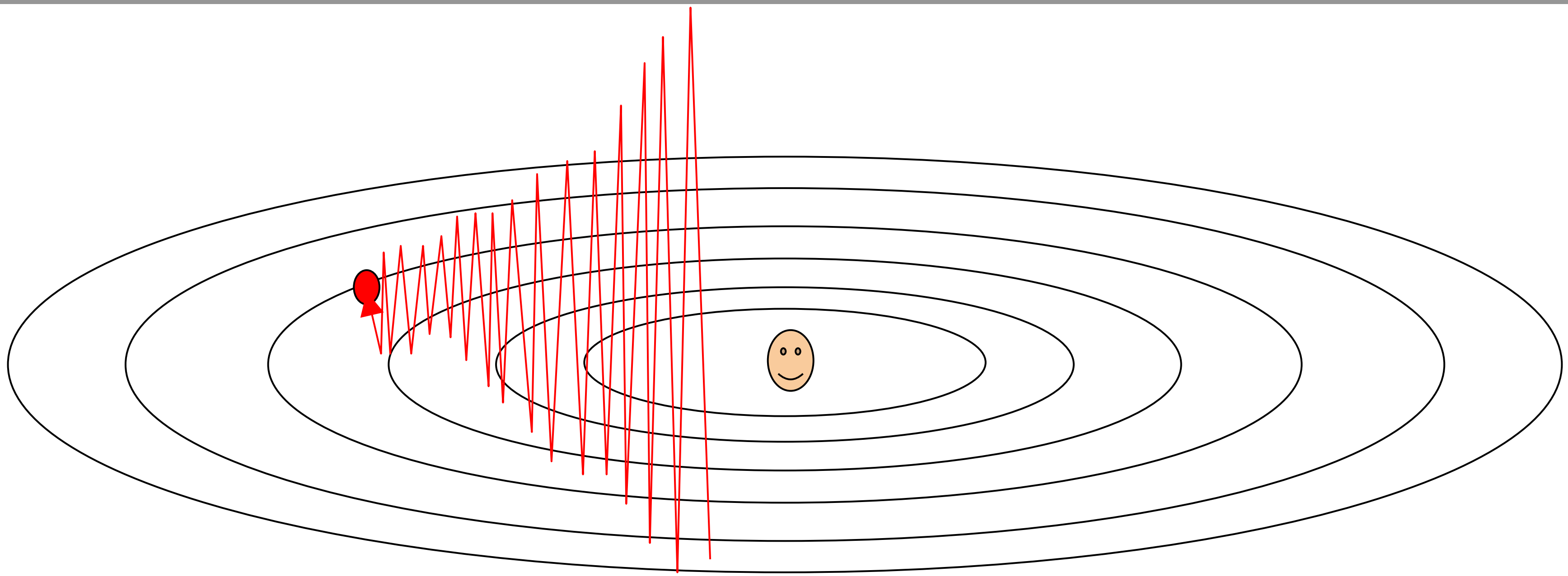
Gradient Magnitudes:



Gradients too big → divergence

Gradients too small → slow convergence

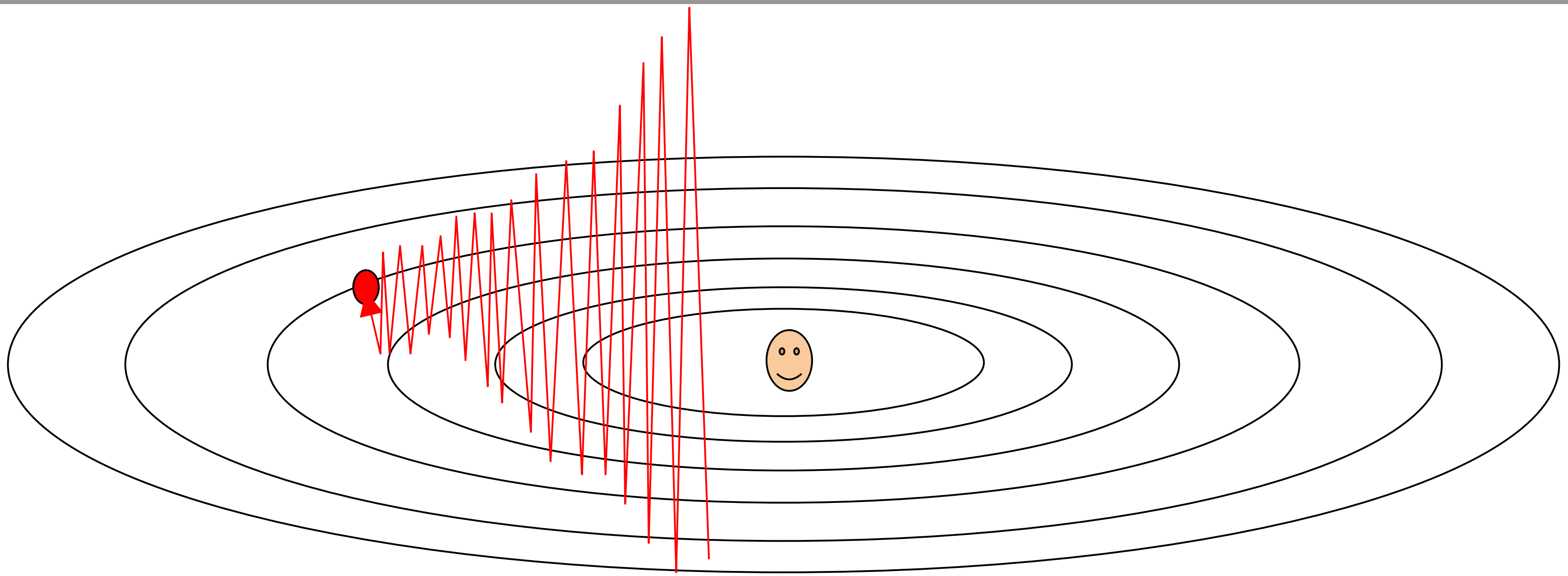
Gradient Magnitudes:



Gradients too big → divergence
Gradients too small → slow convergence

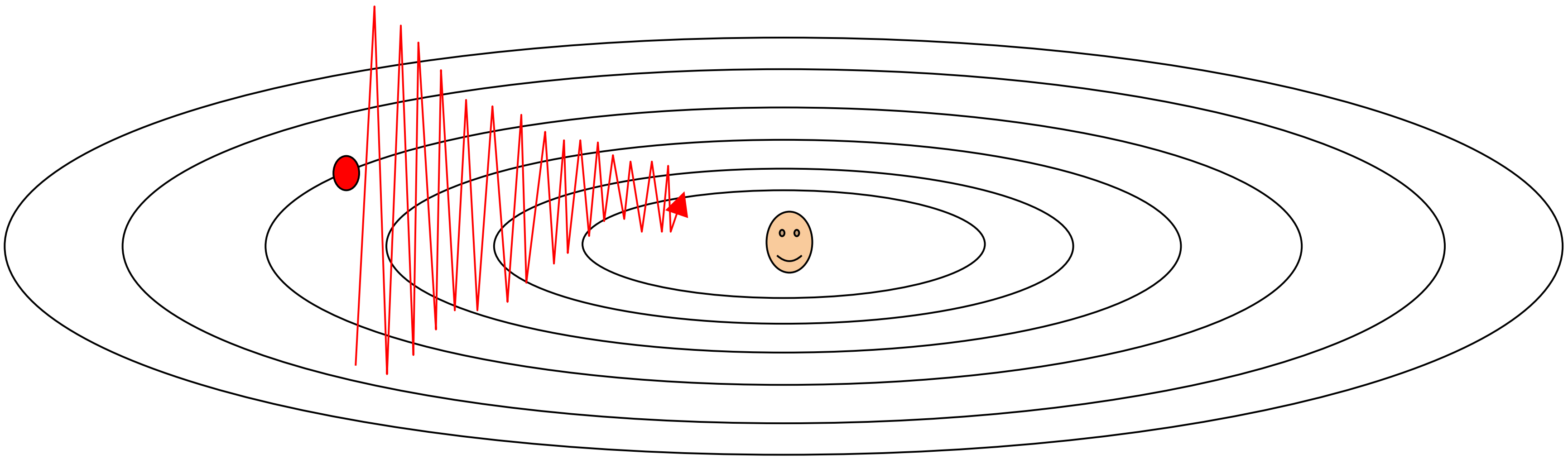
Divergence is much worse!

Gradient Magnitudes



What's the simplest way to ensure gradients stay bounded?

Gradient clipping



Simply limit the magnitude of each gradient:

$$\bar{g}_i = \min(g_{\max}, \max(-g_{\max}, g_i))$$

so $|\bar{g}_i| \le g_{\max}$. Then use a decreasing learning rate to converge to an optimum.