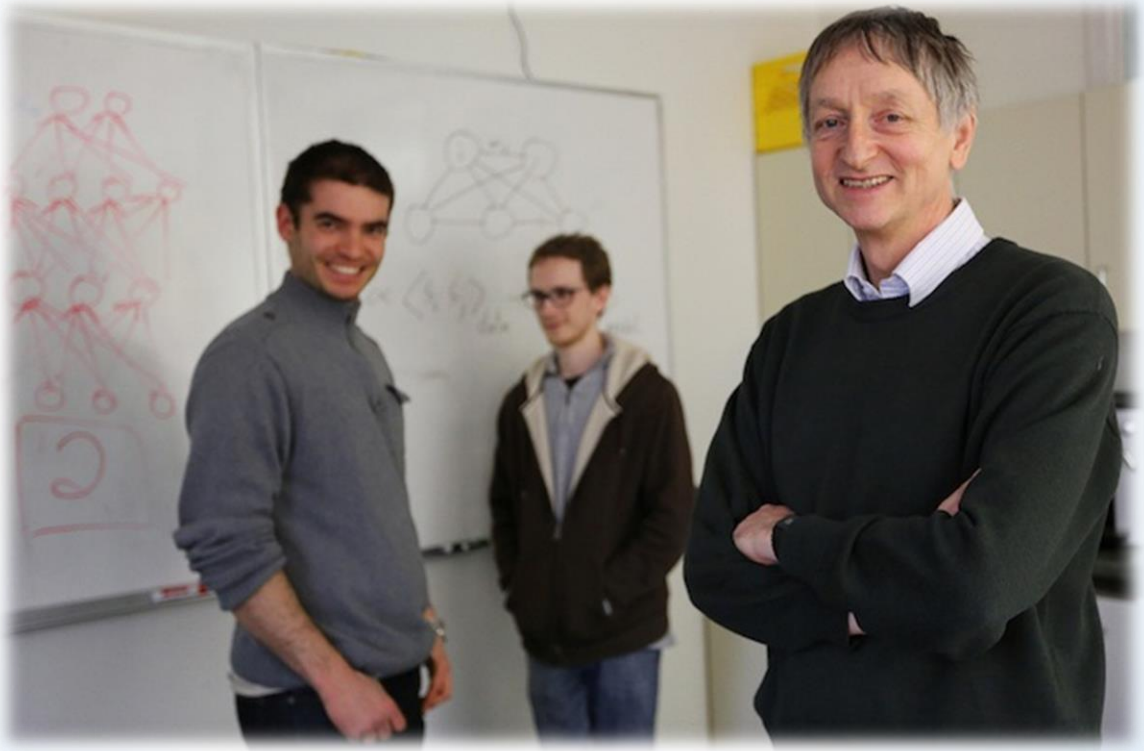
Dropout



Deep Learning Seminar

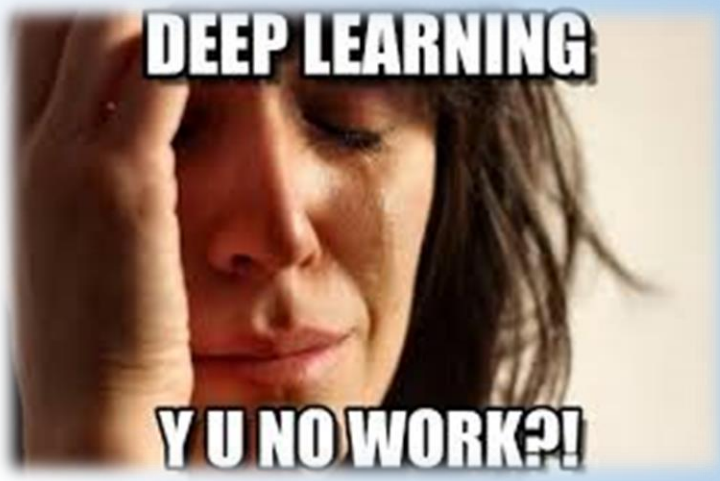
School of Electrical Engineering, Tel Aviv University

Outline

- Motivation
- Inspiration
- Model description
- Results
- Relation to former methods (weight decay)
- DropConnect generalization
- Summary

Motivation

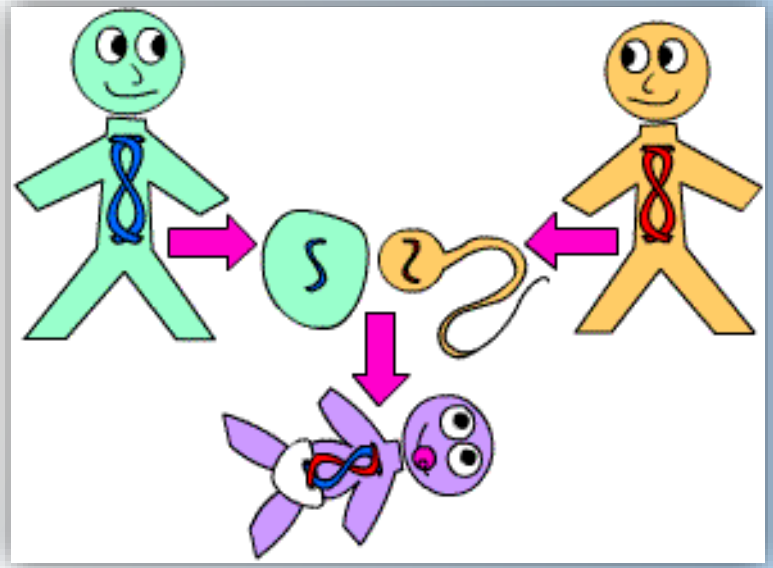
- Neural networks are prone to overfitting.
- The optimal solution is to use a Bayesian framework.
- This solution is infeasible when networks are big.
- We need something faster.



Inspiration

- Genes work well with another small random set of genes.

- Similarly, dropout suggests that each unit should work with a random sample of other units.



Dropout

- At training (each iteration):

  Each unit is retained with a probability p.

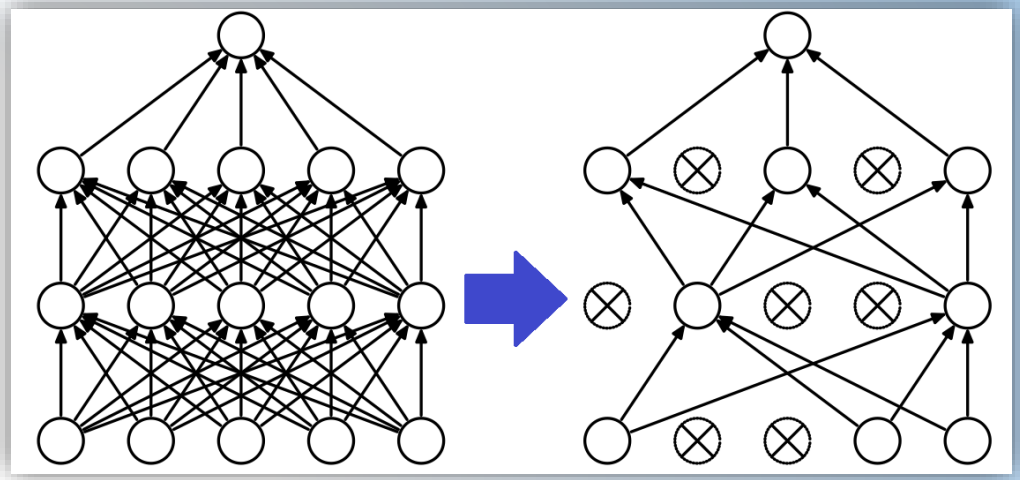
- At test:

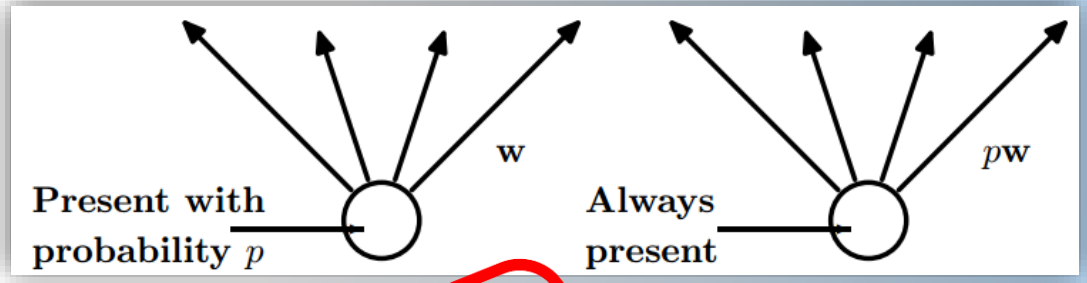
  The network is used as a whole.

  The weights are scaled-down by a factor of p.

- In practice, dropout trains 2^n networks

  (n – number of units).



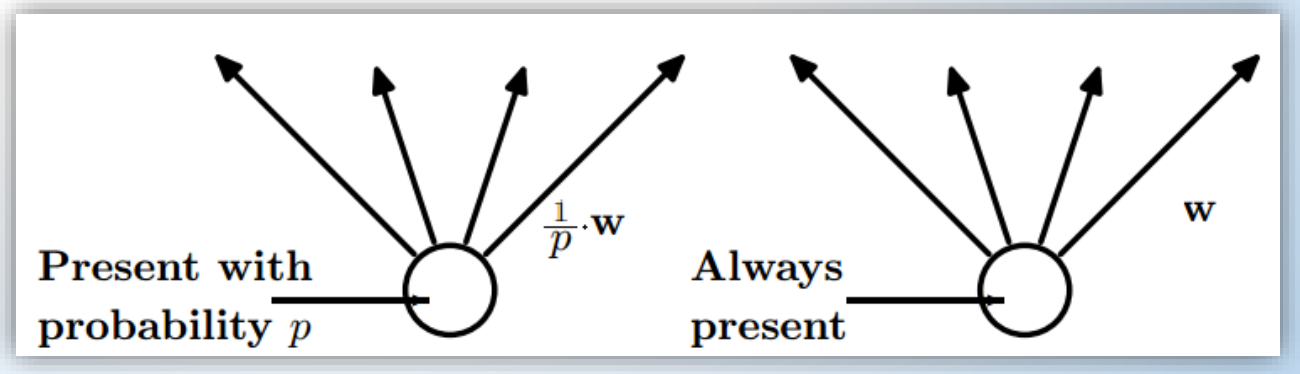


$p = 0.5$ MAKES THE CHARM!

Dropout - an equivalent method

- At training, weights are scaled-up by a factor of $\frac{1}{p}$.
- At test time, no scaling is applied.
- This method is used in Tensorflow:

```
tf.nn.dropout(x, keep_prob=p)
```



Why apply dropout on units, with all their ingoing and outgoing arcs, and not just on the arcs themselves?

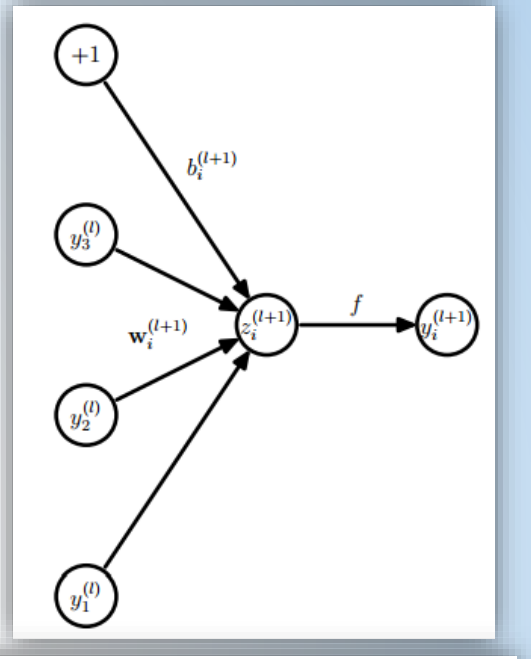
- We will get there later…

Dropout – model description

The feed-forward operation of a standard neural network is:

$$z_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)} \boldsymbol{y}^{(l)} + b_i^{(l+1)}$$

$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$



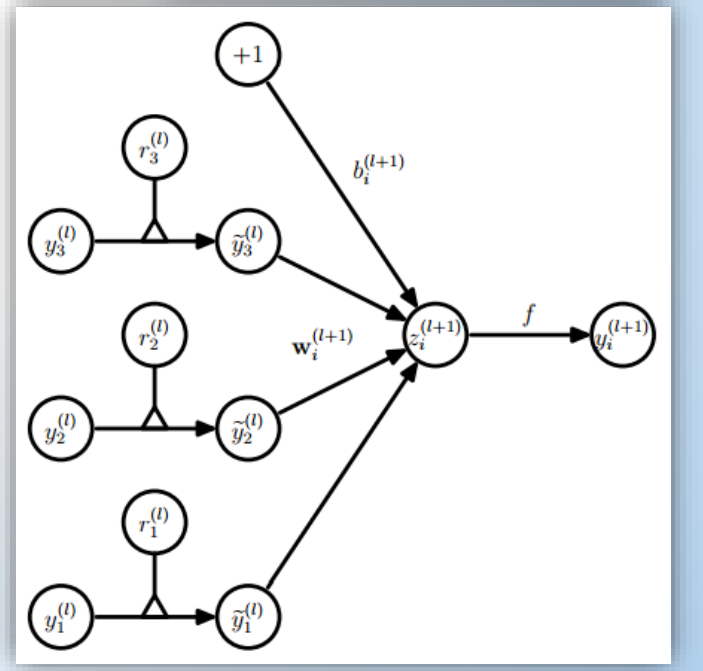
With dropout, the feed-forward operation becomes:

$$r_i^{(l)} \sim \text{Bernoulli}(p)$$

$$\tilde{\boldsymbol{y}}^{(l)} = \boldsymbol{r}^{(l)} * \boldsymbol{y}^{(l)}$$

$$z_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)} \tilde{\boldsymbol{y}}^{(l)} + b_i^{(l+1)}$$

$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$



Dropout – model description

- Backpropagation is performed only on the thinned network, for each training iteration. A training case which does not use a parameter, contributes a gradient of zero for that parameter.

- At test time, the weights are scaled as:

$$W_{test}^{(l)} = p \cdot W^{(l)}$$

Experimental Results

SVHN – Street View House Numbers

- Dropout is applied also to convolutional layers.
- All hidden units are ReLUs.

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

Experimental Results

CIFAR-10 and CIFAR-100:

- Images drawn from 10 and 100 categories respectively.

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

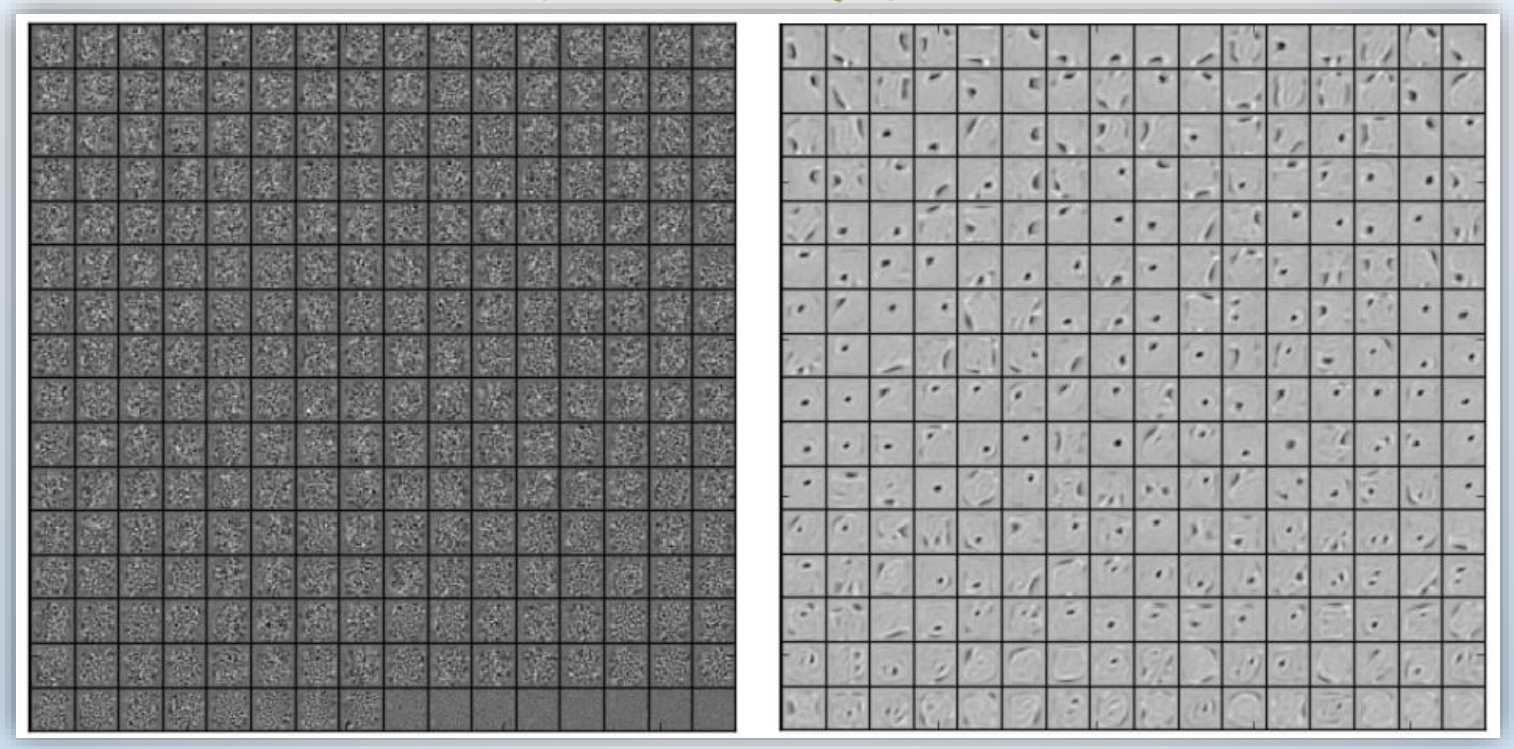
Experimental Results

The effect of dropout on learned features:

- Without dropout, units tend to compensate for mistakes of other units.
- This leads to overfitting, since these co-adaptations do not generalize to unseen data.
- Dropout prevents co-adaptations by making the presence of other hidden units unreliable.

MNIST, one hidden layer, 256 ReLUs

No dropout
Units have co-adapted. Each unit does not detect a meaningful feature.



Dropout ($p = 0.5$)
Units detect edges, strokes and spots in different parts of the image.

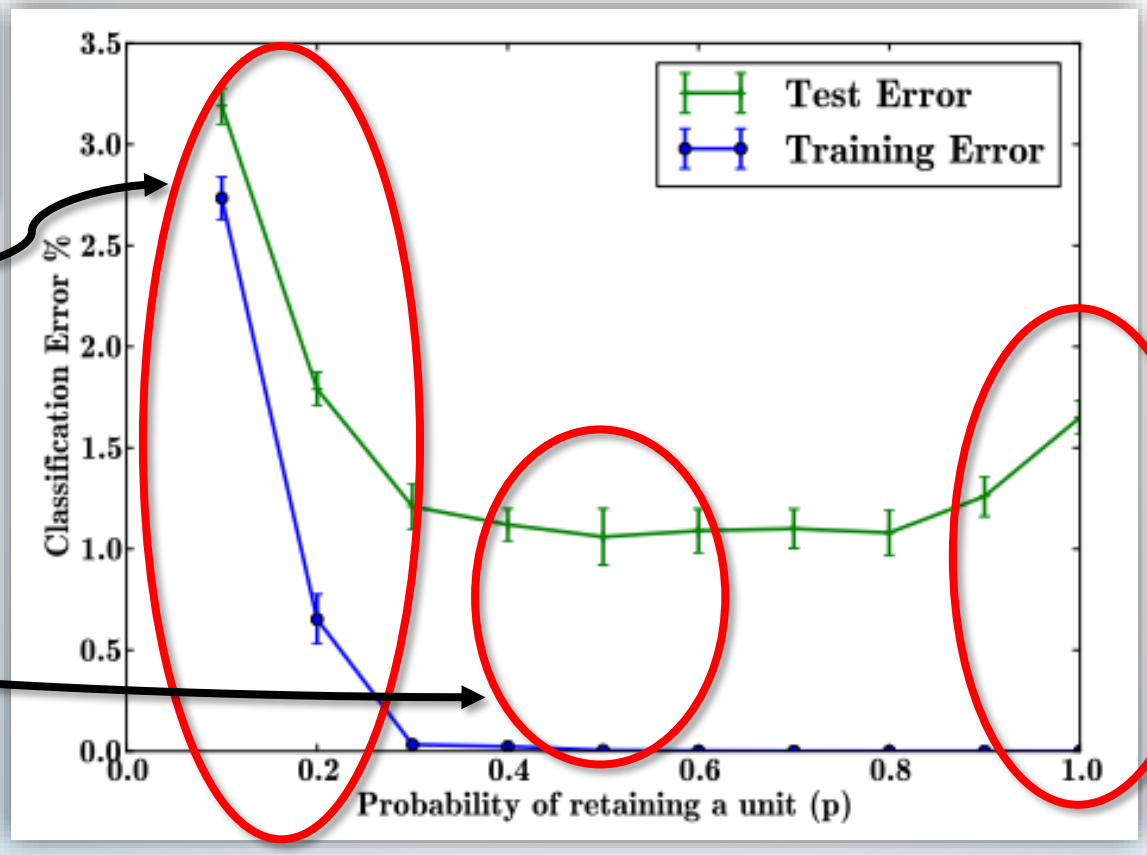
Experimental Results

The effect of the dropout rate p:

- An architecture of 784-2048-2048-2048-10 is used on the MNIST dataset. The dropout rate p is changed from small numbers (most units are dropped out) to 1.0 (no dropout).

High rate of dropout ($p < 0.3$)
- Training error is high
- ➔ Underfitting
- Very few units are turned on during training.

Best dropout rate ($p = 0.5$)
- Training error is low
- Test error is low
- ➔ Mission accomplished!



No dropout ($p = 1.0$)
- Training error is low
- Test error is high
- ➔ Overfitting

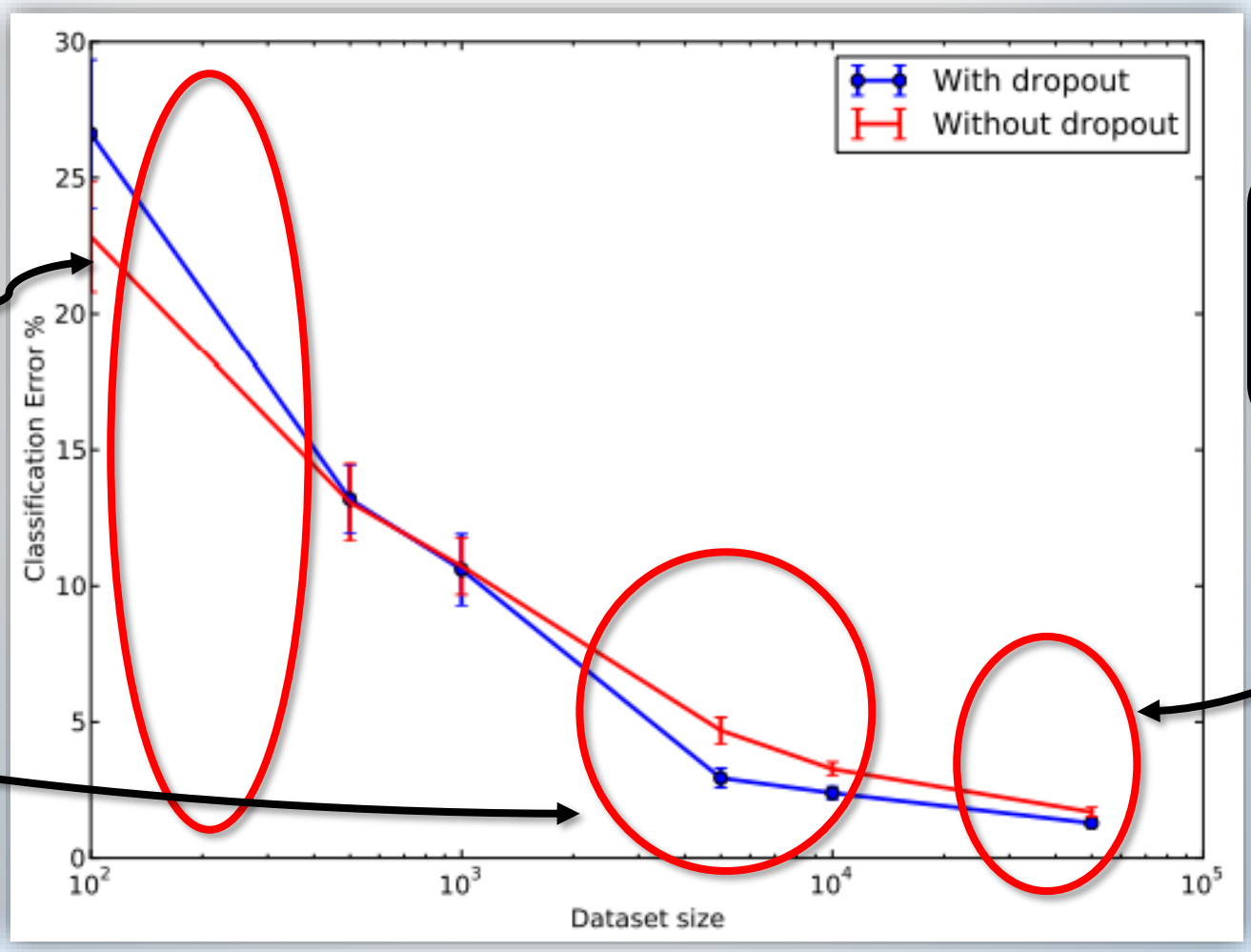
Experimental Results

The effect of data set size:

- An architecture of 784-1024-1024-2048-10 is used on the MNIST dataset.

Extremely small data set
Dropout does not improve error rate, and even makes it worse.

Average to large data set
Dropout improves error rate.



Huge data set
Dropout barely improves the error rate. The data set is big enough, so that overfitting is not an issue.

Weight Decay

- Limiting the growth of the weights in the network.
- A term is added to the original loss function, penalizing large weights:

$$\mathcal{L}_{new}(\boldsymbol{w}) = \mathcal{L}_{old}(\boldsymbol{w}) + \frac{1}{2}\lambda\|\boldsymbol{w}\|_2^2$$

- A network architecture of 784-1024-1024-2048-10 is used on the MNIST data set, with different regularizations.

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**

Weight Decay

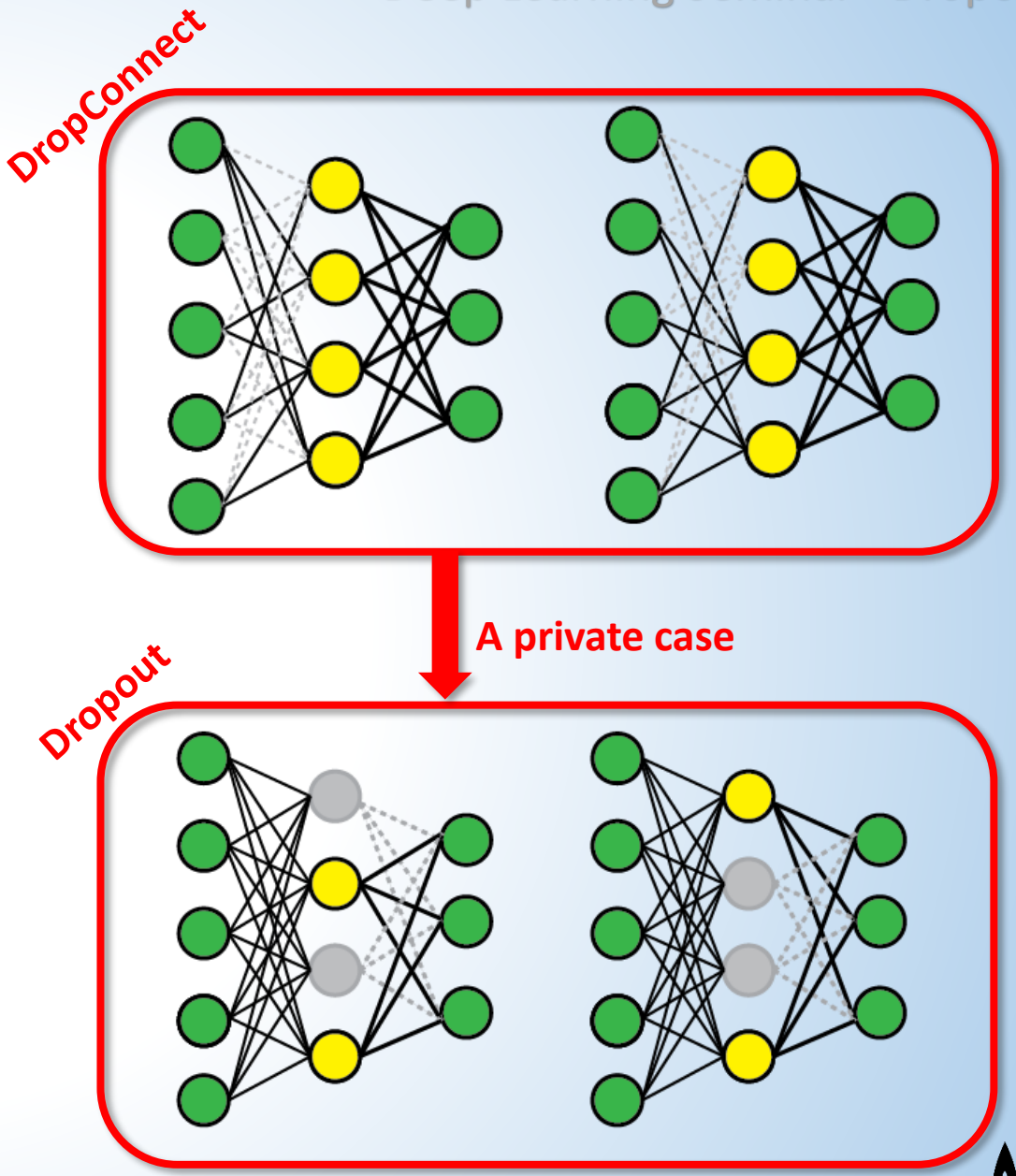
Dropout has more advantages over weight decay (Helmblod et al., 2016):

- Dropout is scale-free: dropout does not penalize the use of large weights when needed.

- Dropout is invariant to parameter scaling: dropout is unaffected if weights in a certain layer are scaled up by a constant c, and the weights in another layer are scaled down by a constant c. This implies that dropout does not have an isolated local minima.

DropConnect

- DropConnect (Yann LeCun et al., 2013) generalizes dropout. It suggests dropping out weights.
- As in Dropout, $p = 0.5$ usually gives the best results.



DropConnect

Averaging Technique:

- DropConnect asserts:

$$z^{l+1} = (M * W)y^l$$

Input to the activation function

A weighted sum of Bernoulli variables. Can be approximated by a Gaussian

$$E_M[z^{l+1}] = pWy^l$$

$$V_M[z^{l+1}] = p(1-p)(W * W)(y^l * y^l)$$

Statistics of the Gaussian

- At test:
  - Draw samples of z^{l+1} according to the Gaussian distribution.
  - Feed the samples into the activation function ($f\left(z^{l+1}\right)$).
  - Average.

DropConnect

No-drop, Dropout and DropConnect comparison:

- MNIST: 2 layers (800 neurons each).
- CIFAR-10: 4 layers. Dropout/DropConnect is applied only on the final layer.
- SVHN: Generally same architecture as in CIFAR-10. Due to the large training set size, all methods achieve nearly the same performance.

MNIST

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**

CIFAR-10

model	error(%)
No-Drop	23.5
Dropout	19.7
DropConnect	**18.7**

SVHN

model	error(%) 5 network	voting error(%)
No-Drop	2.26 ± 0.072	**1.94**
Dropout	2.25 ± 0.034	1.96
DropConnect	2.23 ± 0.039	**1.94**

DropConnect

In DropConnect's paper, they achieve the lowest error rate recorded so far in MNIST!

crop	rotation scaling	model	error(%) 5 network	voting error(%)
no	no	No-Drop	0.77±0.051	0.67
		Dropout	0.59±0.039	**0.52**
		DropConnect	0.63±0.035	0.57
yes	no	No-Drop	0.50±0.098	0.38
		Dropout	0.39±0.039	0.35
		DropConnect	0.39±0.047	**0.32**
yes	yes	No-Drop	0.30±0.035	**0.21**
		Dropout	0.28±0.016	0.27
		DropConnect	0.28±0.032	**0.21**

These results are equivalent for the use of DropConnect, and the use of no drop at all…

DropConnect

DropConnect's drawbacks:

- Training with DropConnect is slower.
- DropConnect's implementation is more complicated than Dropout's implementation.
- In their paper, DropConnect has been proven to work mostly when using more than one network.

Summary

- Dropout is a very good and fast regularization method.
- Dropout is a bit slow to train (2-3 times slower than without dropout).
- If the amount of data is average-large – dropout excels. When data is big enough, dropout does not help much.
- Dropout achieves better results than former used regularization methods (Weight Decay).
- DropConnect is a generalization of dropout. Its superiority over dropout is unclear. It involves complications in the implementation, and is also slower to train than dropout

Summary

Generally, use a small dropout value of 20%-50% of neurons with 20% providing a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network

Use a larger network. You are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations

Use dropout on incoming (visible) as well as hidden units. Application of dropout at each layer of the network has shown good results

Summary

Use a large learning rate with decay and a large momentum. Increase your learning rate by a factor of 10 to 100 and use a high momentum value of 0.9 or 0.99

Constrain the size of network weights. A large learning rate can result in very large network weights. Imposing a constraint on the size of network weights such as max-norm regularization with a size of 4 or 5 has been shown to improve results