Dropout: A simple way to prevent neural networks from overfitting

Srivastava, Nitish, et al.

경상국립대학교 석사과정 오서영

Abstract

Overfitting is a serious problem in Deep neural nets with a large number of parameters. -> **Dropout** is a technique for addressing this problem

Dropout

The key idea is to randomly drop units from the neural network during training.
-> This prevents units from co-adapting too much.
-> During training, dropout samples from an exponential number of different thinned networks.
-> At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights.
→ This significantly reduces overfitting and gives major improvements over other regularization methods.

Introduction

Many methods for reducing overfitting

1. Stopping the training as soon as performance on a validation set starts to get worse.
2. Introducing weight penalties of various kinds such as L1 and L2 regularization
3. Soft weight sharing

With unlimited computation, the best way to **regularize** a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by its posterior probability given the training data.

- Quite well for simple or small models

→ But we would like to approach the performance of the Bayesian gold standard using considerably less computation

→ **Approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters.**

Introduction

Model combination

Nearly always improves the performance of ML, but expensive (with large NN)

- Combining several models is most helpful when..

1. They have different architecture
2. They are trained on different data

→ Even if one was able to train many different large networks, using them all at test time is infeasible in applications (long response time)

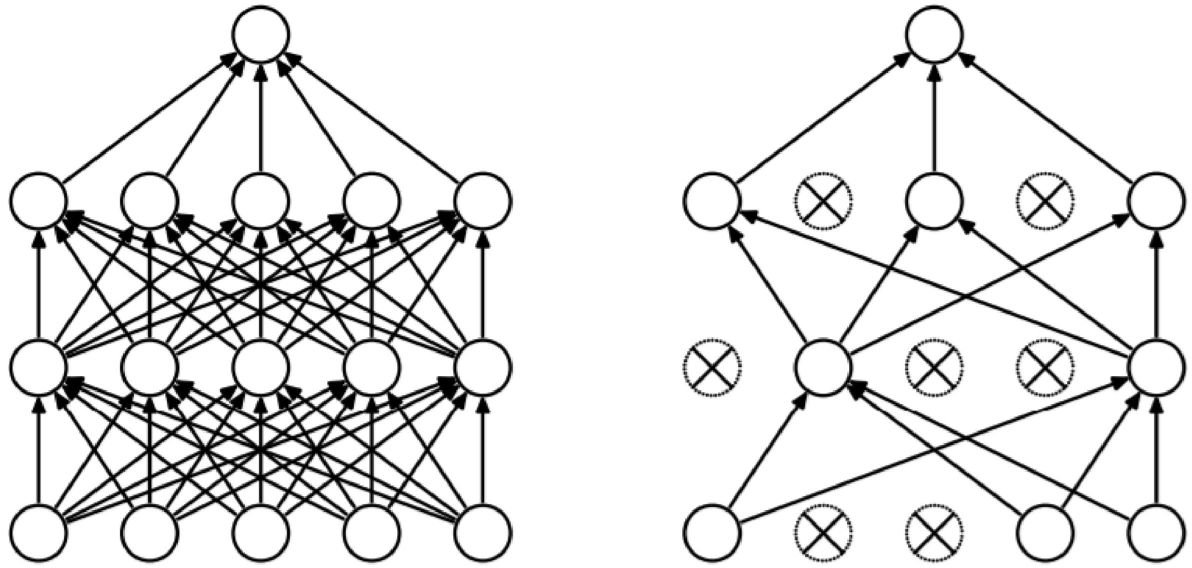
Addressing both issues (overfitting, model combination) -> **Dropout**

- It prevents overfitting and provides a way of approximately combining exponentially many different NN architectures efficiently.

- Dropout = dropping out units in NN
- The choice of which units to drop is random

→ Each unit is retained with a fixed probability **p** independent of other units

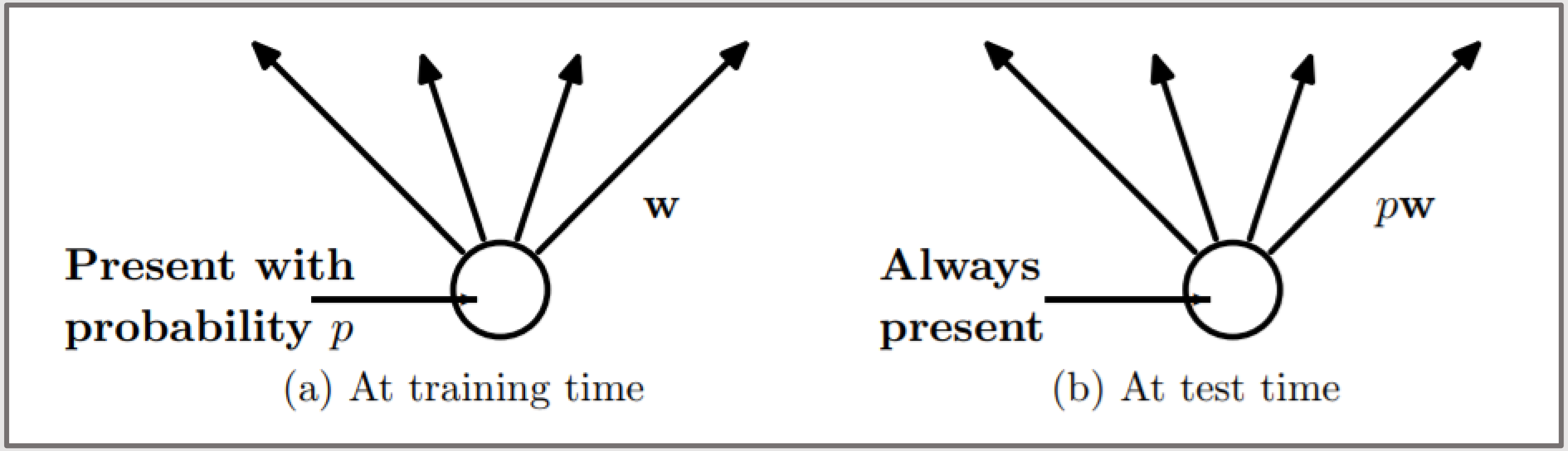
Introduction



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

Applying dropout to a NN amounts to sampling a **thinned network** from it.
The thinned network consists of all the units that survived dropout
NN with n units -> collection of 2^n possible thinned NNs (parameters : O(n^2))

Introduction



At **test time**, it is not feasible to explicitly average the predictions from exponentially many thinned models. -> **approximate averaging method**

- Using a single neural net at test time without dropout
- If a unit is retained with probability p during training, the outgoing **weights of that unit are multiplied by p** at test time

Model description

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

Feed forward of standard NN

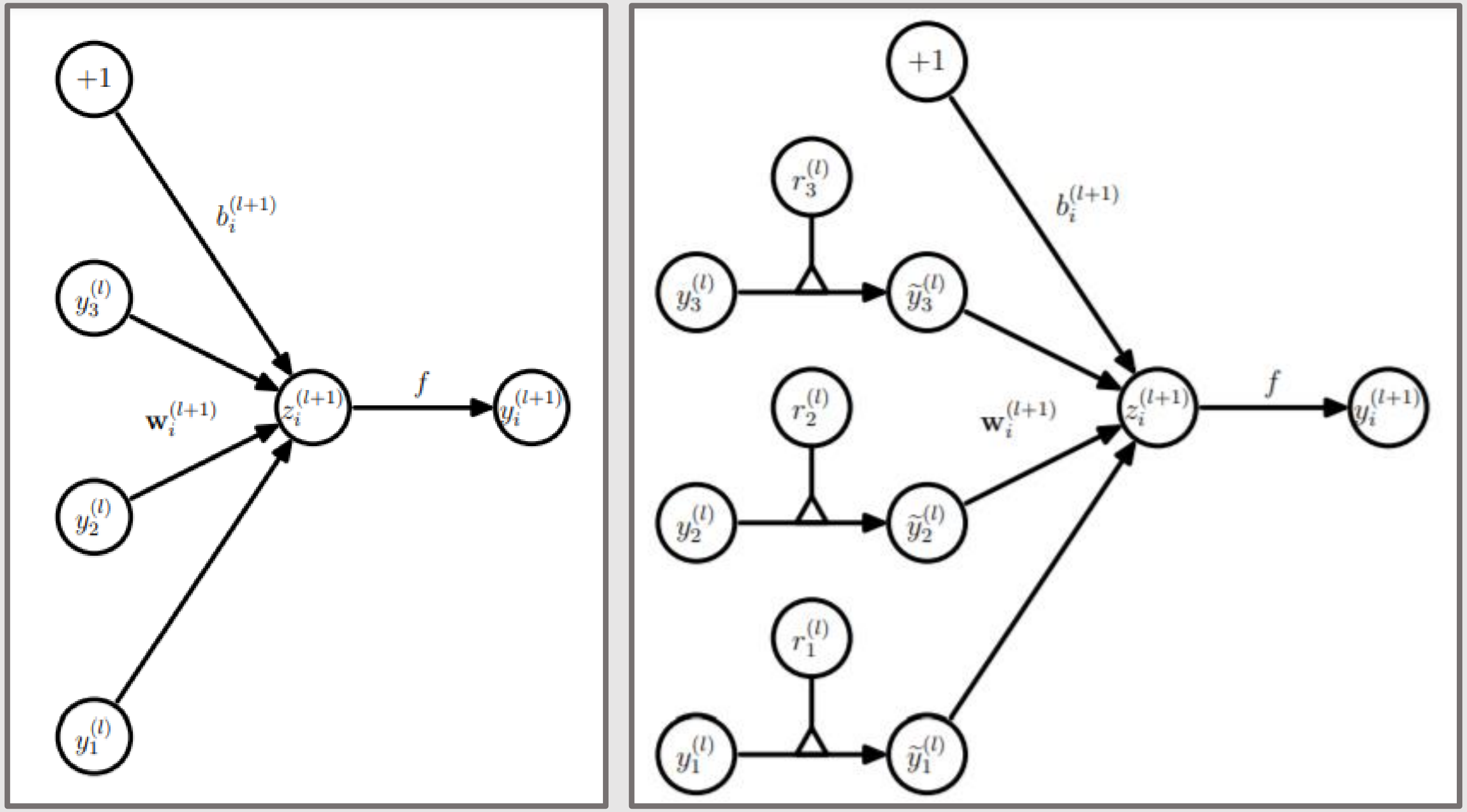
$$
\begin{aligned}
r_j^{(l)} &\sim \mathrm{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

Feed forward with dropout

L : # hidden layers
l : index of the hidden layers
z : input vector
y : output vector
W : weight vector
b : bias vector
f : activation function

Dropout neural network model

Model description



Comparison of the standard and dropout network

L : # hidden layers
l : index of the hidden layers
z : input vector
y : output vector
W : weight vector
b : bias vector
f : activation function

Learning Dropout Nets

Backpropagation

- Forward and backpropagation for that training case are done only on this thinned network.
- The gradients for each parameter are **averaged over** the training cases in each mini-batch.
- momentum, annealed learning rate, L2 weight decay are useful for dropout NN

Max-norm regularization

- Useful for dropout – constraining the norm of the incoming weight vector at each hidden unit to be upper bounded by a fixed constant **c** (hyperparameter)
- If W represents the vector of weights incident on any hidden unit, the NN was optimized under the constraint $||\boldsymbol{W}||_2 \leq \mathbf{c}$

→ This constraint was imposed during optimization by projecting W onto the surface of a ball of radius c, whenever W went out of it.

Learning Dropout Nets

Unsupervised Training
NN can be pretrained using stacks of RBM or autoencoders, DBM
- The weights obtained from pretraining should be scaled up by a factor of **1/p**

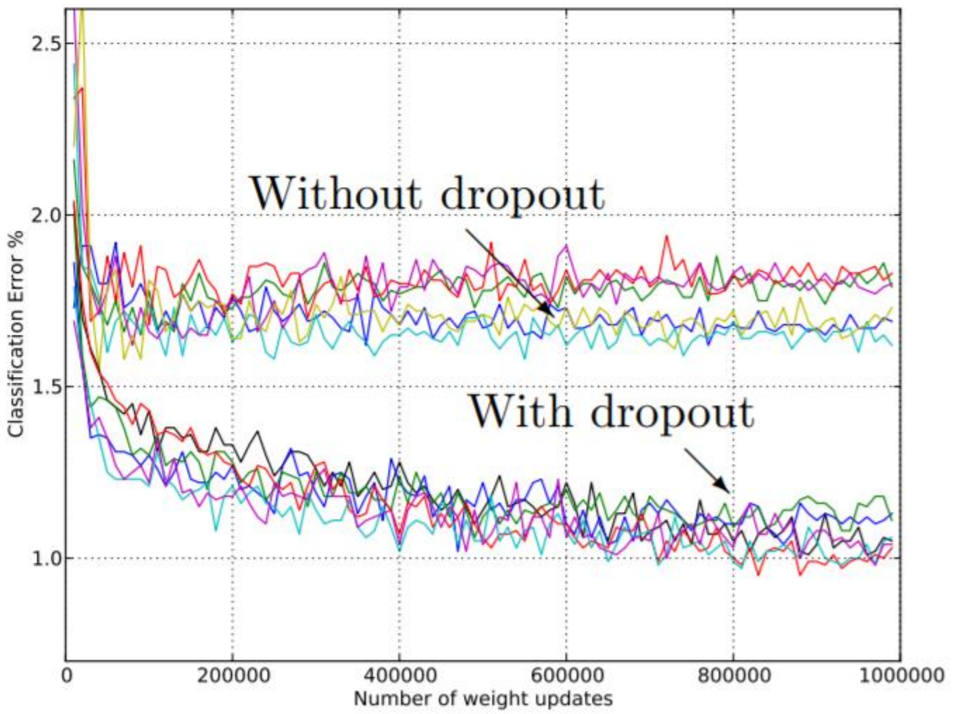
-> This makes sure that for each unit, the expected output from it under random dropout will be the **same** as the output during pretraining.

- The stochastic nature of dropout might wipe out the information in the pretrained weights -> **choose learning rates smaller**

Experimental Results

Data Set	Domain	Dimensionality	Training Set	Test Set
MNIST	Vision	784 (28 × 28 grayscale)	60K	10K
SVHN	Vision	3072 (32 × 32 color)	600K	26K
CIFAR-10/100	Vision	3072 (32 × 32 color)	60K	10K
ImageNet (ILSVRC-2012)	Vision	65536 (256 × 256 color)	1.2M	150K
TIMIT	Speech	2520 (120-dim, 21 frames)	1.1M frames	58K frames
Reuters-RCV1	Text	2000	200K	200K
Alternative Splicing	Genetics	1014	2932	733

Dataset

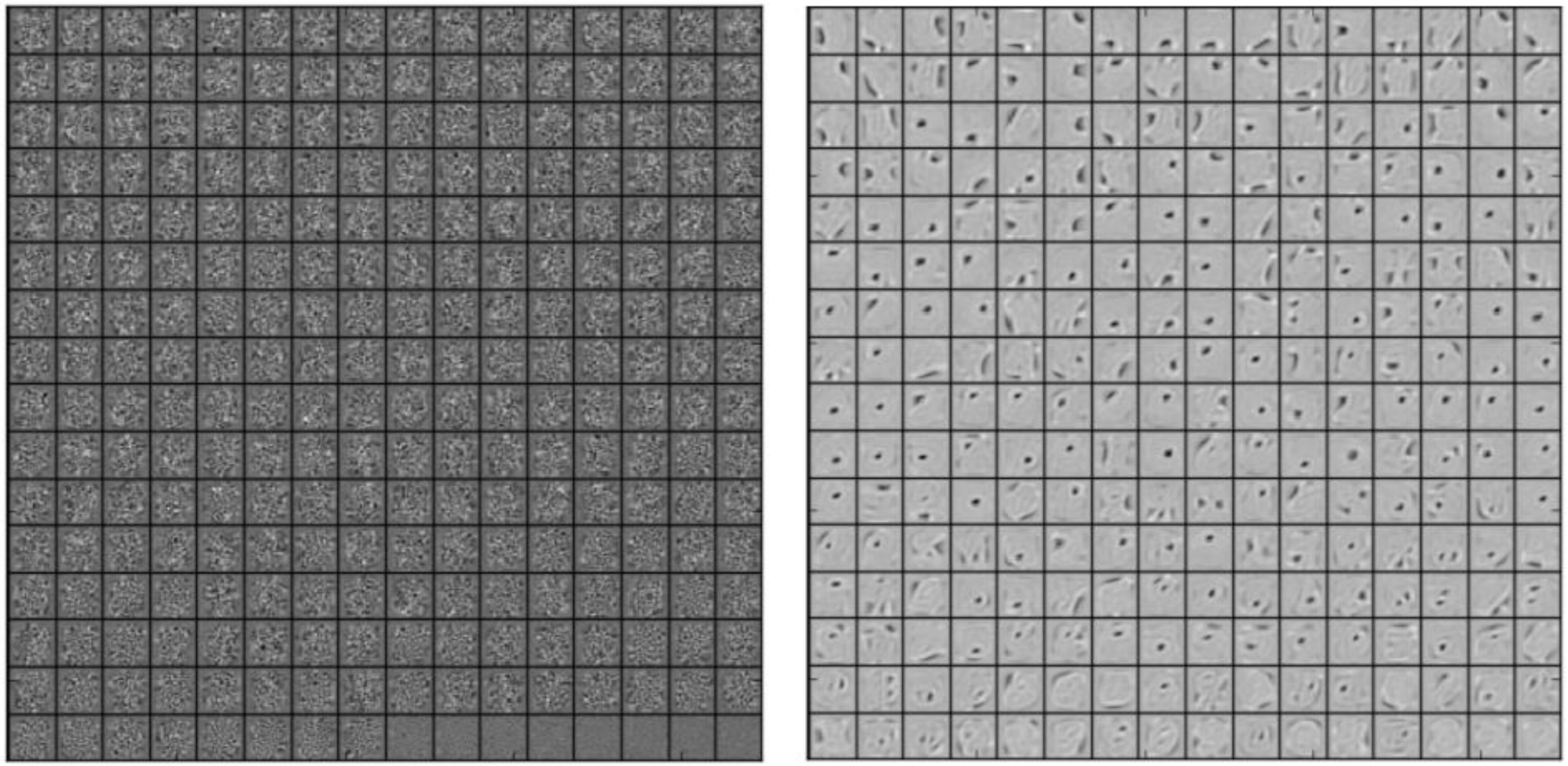


Results

Salient Features

Effect on Features

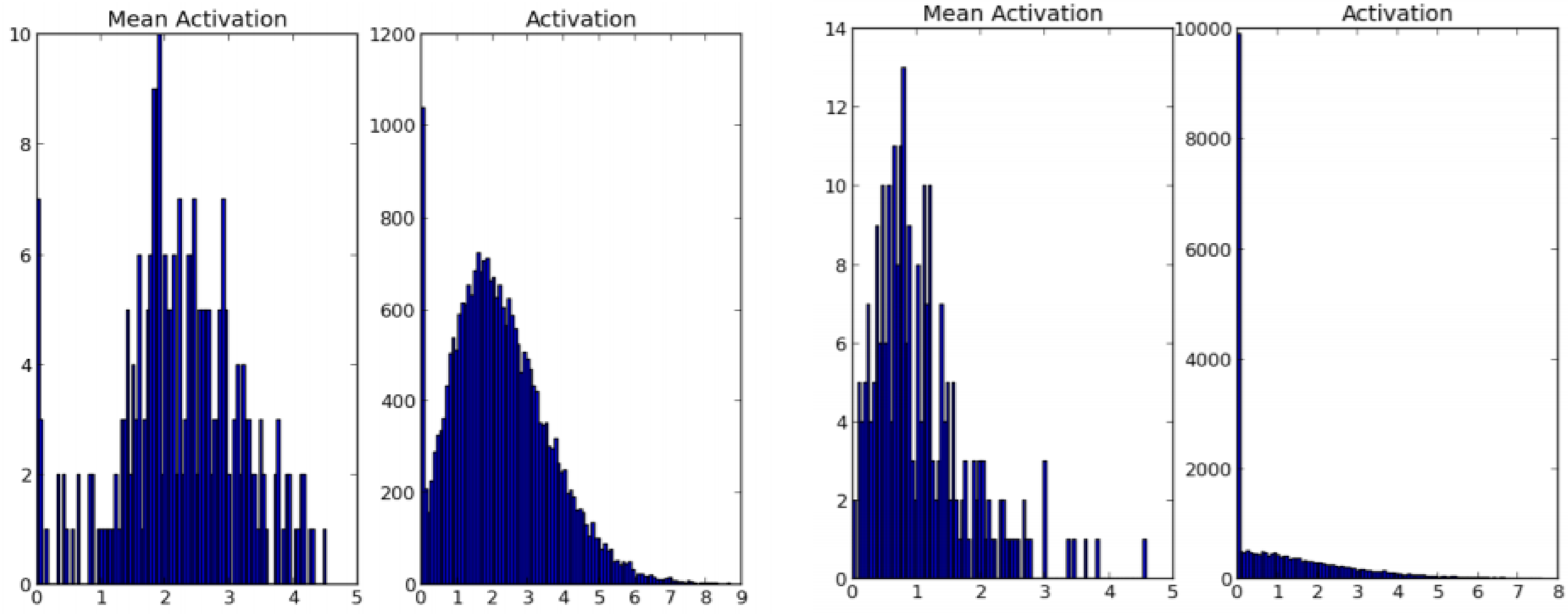
It is apparent that the features shown in (a) have co-adapted in order to produce good reconstruction



(a) Without dropout (b) Dropout with $p = 0.5$.

Salient Features

Effect on Sparsity
The activations of the hidden units become sparse.

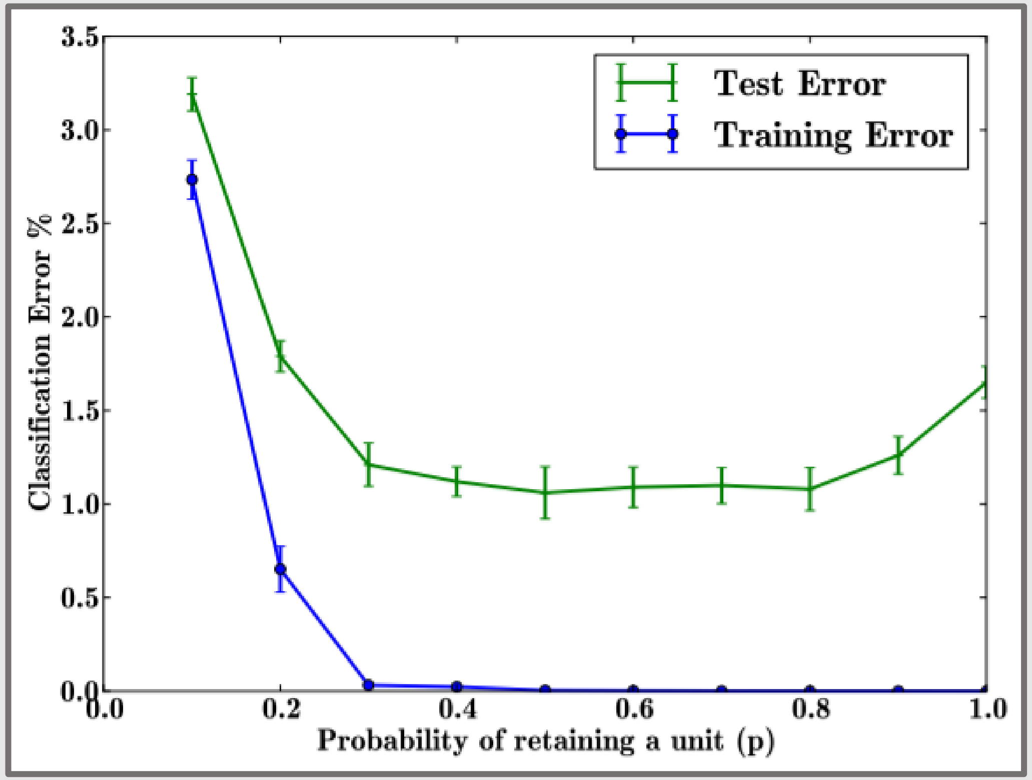


(a) Without dropout (b) Dropout with $p = 0.5$.

Salient Features

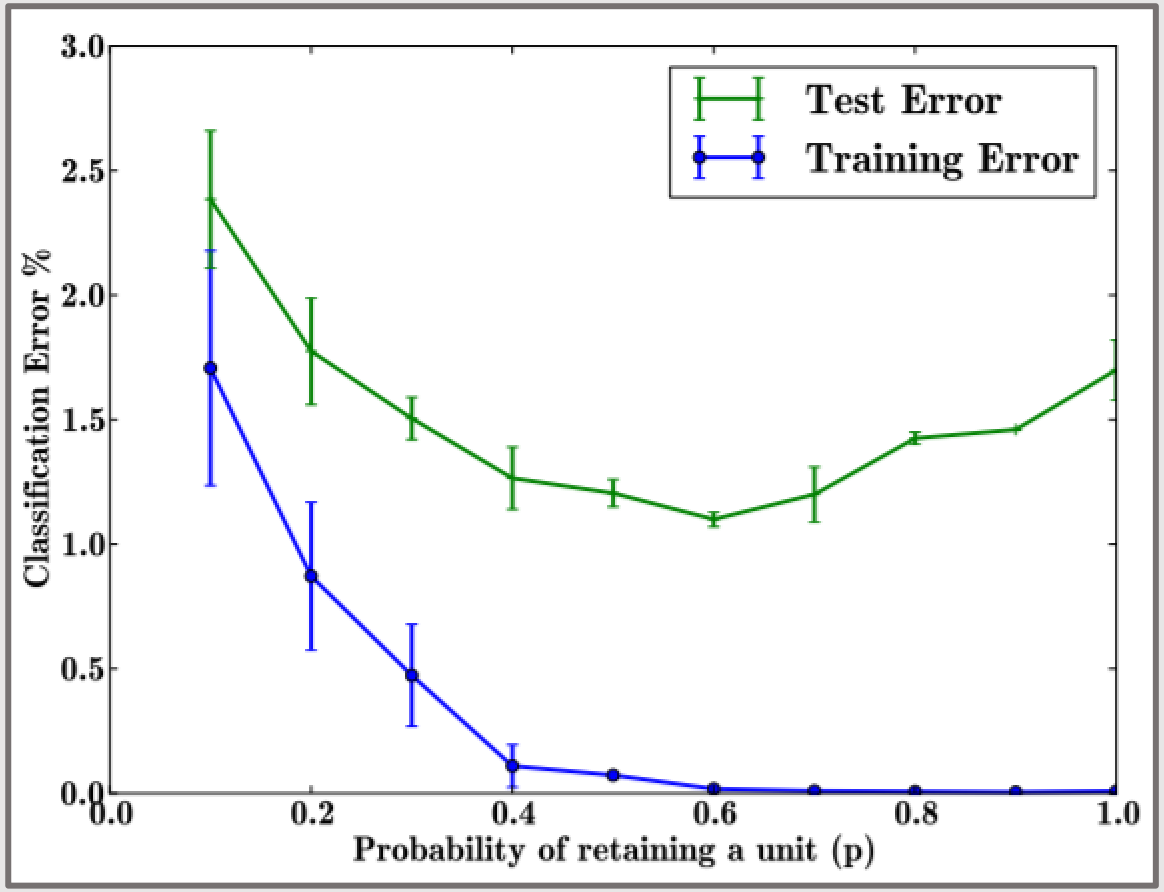
Effect on Dropout rate
The effect of varying the hyperparameter p, n : # of neurons in hidden layer

(1) Keeping n fixed -> evaluating train, test error
→ It becomes flat when $0.4 \le p \le 0.8$



Salient Features

(2) Keeping pn fixed -> evaluating train, test error
→ Errors for small values of p has reduced by a lot compared to **(1)**



Optimal value
0.5, 0.6

Salient Features

Effect on Data Set size