Current Topics in Artificial Intelligence: Regularization

Chenxi Liu
Department of Statistics
University of California, Los Angeles
cxliu@ucla.edu

Abstract

This short survey discusses the role of regularization in current deep learning research. Up till now, dropout remains the most popular choice of all deep neural network regularization techniques, and is what this survey is centered around. We first give a general introduction and interpretation of dropout (Section 1), followed by some follow-up works which either improve speed or offer analysis (Section 2). Then methods other than dropout, namely DropConnect and Batch Normalization, are introduced (Section 3) to provide comparison and contrast. Finally the necessity of regularization is argued (Section 4). This is the second of the four short surveys.

1 Dropout

Regularization is important for deep neural networks, because they usually have millions of parameters to learn, and given the limited training data, overfitting is quite likely to happen. Dropout [1] stood out as a simple yet effective regularization technique, and first attracted major attention when it greatly alleviated the overfitting problem of AlexNet [2], the ILSVRC 2012 winner.

1.1 Formulation and Intuition

The idea of dropout is quite simple. It introduces stochasticity into the neural network by (temporarily) dropping random units out, along with all its incoming and outgoing connections. This is equivalent to sampling a "thinned" network from the complete architecture. Concretely:

- During each training iteration, drop each unit in a layer out with probability p (usually set to 0.5, which means randomly dropping half of the units in the current layer).
- During testing, multiply all the outgoing weights of the units by p and make final prediction using all the units.

The intuition behind dropout is as follows. If the capacity of the network is much larger than the data it is representing, then overfitting is likely to happen. In terms of weights, this could mean that a large number of weights have learned to "cooperate" or "co-adapt" to fit the data point. These complex co-adaptations could go wrong on the new test data. On the other hand, by randomly dropping half of the units, a unit is forced to work with different "co-workers" during each iteration of training. As a result, it is likely to learn something that is *individually* useful.

But after learning is done, we no longer want to use the "thinned" networks and would like to utilize all the knowledge that the network has learned. Since the weights of the next layer is learned when we drop out half of the units in the current layer, if we fix the outgoing weights of all the units, then

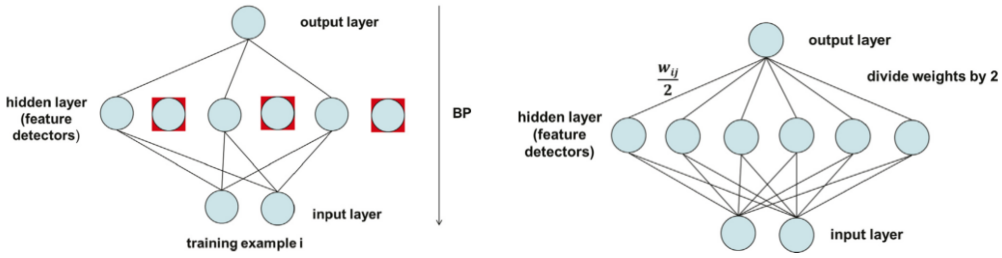

Figure 1: Left: Dropout training; Right: Dropout prediction

the "intensity" that is fed into the next layer is twice as much, which will probably make the learned weights of the next layer useless. Therefore it makes sense to half the outgoing weights of all the units in the current layer during testing.

We now interpret dropout in terms of model averaging, which is a more formal explanation. By randomly omitting each hidden unit with probability 0.5, we are randomly sampling from 2^H different architectures, where H is the number of hidden units in the hidden layer. This is a very large number, so only a few of these models ever get trained, and very likely that they only get one training example. All these architectures share weights, which means every model is very strongly regularized.

Since all these architectures are thin and receive very little training examples, they are expected to perform poorly when making individual predictions. However the performance will be much better when we average the predictions of these exponentially many models. It turns out that by halving the outgoing weights of the units during test time, we are computing the geometric mean of the predictions of all 2^H models. This suggests that dropout is in fact a very efficient way of doing model averaging.

1.2 Mathematical Interpretation

We now try to interpret dropout more formally using equations. Much of the analysis in this subsection comes from [3]. We focus on the model averaging analysis of dropout.

We first consider the case of single layer of linear units. The activations are given by

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

Applying dropout gives

$$S_i(I) = \sum_{j=1}^{n} w_{ij} \delta_j I_j \tag{2}$$

and taking expectation

$$E(S_i) = \sum_{j=1}^{n} w_{ij} p_j I_j \tag{3}$$

So dropout computes the expectation exactly. Similar conclusions hold for deep linear networks.

We then consider the case of logistic units. The activations are given by $O = \sigma(S)$. There are 2^n possible sub-networks indexed by N. For a fixed input I, each sub-network produces a linear value

$S(N, I)$ and a final output value $O_N = \sigma(S(N, I))$. In the uniform case, the geometric mean of the outputs and complementary outputs are given by

$$G = \prod_N O_N^{1/2^n} \quad G' = \prod_N (1 - O_N)^{1/2^n} \tag{4}$$

respectively. The normalized geometric mean (NGM) is defined by

$$NGM = \frac{G}{G + G'} \tag{5}$$

and we can show that

$$NGM(\sigma(S)) = \sigma(E(S)) = \sigma(\frac{\sum_N S(N, I)}{2^n}) \tag{6}$$

So for logistic units, dropout computes the normalized geometric mean of individual predictions exactly. This is also true for a single layer of logistic units, but unfortunately it is no longer true for deep neural networks, where

$$E(O_i^h) \approx NGM(O_i^h) = \sigma(E(S_i^h)) \approx W_i^h \tag{7}$$

Nevertheless, we can still show that both G and NGM are good approximations of $E(O_i^h)$ and W_i^h, and that the deviations are approximately Gaussians with zero mean and small standard deviations. In addition, these deviations act like noise and cancel each other to some extent, preventing the error accumulation across layers [3].

2 Following Dropout

2.1 Fast Dropout

[4] speeds up dropout training by interpreting dropout as Monte Carlo approximation. Suppose $z_i \sim Bernoulli(p_i)$ for $i = 1, ..., m$. Then $Y(z) = \sum_{i=1}^m w_i x_i z_i$ can be seen as a Monte Carlo approximation to $E_z[Y(z)]$. From central limit theorem, $Y(z)$ tends to a normal distribution as $m \to \infty$. Therefore we can describe the distribution of $Y(z)$ with a Gaussian

$$S = E_z[Y(z)] = \sqrt{Var[Y(z)]}\epsilon = \mu_s + \sigma_s \epsilon \tag{8}$$

where $\epsilon \sim N(0, 1)$. So instead of drawing m Bernoulli random variables, we only need to sample from one Gaussian distribution to do the same job.

While the forward pass of the network is straightforward, the implementation of training becomes quite complicated. Therefore, it is suggested that one could use fast dropout at test time only.

2.2 Dropout as Adaptive Regularization

[5] tries to explain dropout as a form of regularization. The paper primarily studies the generalized linear models (GLMs)

$$p_\beta(y|x) = h(y) \exp\{yx \cdot \beta - A(x \cdot \beta)\} \tag{9}$$

Dropout is a way of adding noise to the feature. Applying MLE to the parameters yields the regularization term

$$R(\beta) = \sum_{i=1}^{n} E_{\xi}[A(\tilde{x_i} \cdot \beta)] - A(x_i \cdot \beta) \tag{10}$$

where A is the partition function and $\tilde{x_i}$ is the noisy feature. Further, by taking Taylor expansion, this term can be approximated by

$$R^q(\beta) = \frac{1}{2} \sum_{i=1}^{n} A''(x_i \cdot \beta) Var_{\xi}[\tilde{x_i} \cdot \beta] \tag{11}$$

For different noise type and function class, this term changes into different forms, and the main result is summarized in the following table.

	Linear Regression	Logistic Regression	GLM
L_2-penalization	$\|\beta\|_2^2$	$\|\beta\|_2^2$	$\|\beta\|_2^2$
Additive Noising	$\|\beta\|_2^2$	$\|\beta\|_2^2 \sum_i p_i(1-p_i)$	$\|\beta\|_2^2 \operatorname{tr}(V(\beta))$
Dropout Training	$\|\beta\|_2^2$	$\sum_{i,j} p_i(1-p_i)\, x_{ij}^2\, \beta_j^2$	$\beta^\top \operatorname{diag}(X^\top V(\beta) X)\beta$

Figure 2: Form of different regularization schemes

The main message is that instead of penalizing each feature equally, rare but highly discriminative features receive little penalty in dropout, which suggests that it is a more advanced form of regularization. The limitation of this paper is that the focus is mainly upon GLMs, not the highly non-linear deep neural networks.

2.3 Dropout as Bayesian Approximation

Deep learning is usually happy with a point estimation, which does not provide information on model uncertainty. Bayesian models, on the other hand, are able to capture richer information, but are usually computationally prohibitive. [6] argues that dropout, when performed during test time, offers insight into the model uncertainty of the deep neural network. Specifically, the variance of the label prediction is

$$Var(y) = \tau^{-1} I_D + \frac{1}{T} \sum_{t=1}^{T} y_t^T y_t - E(y)^T E(y) \tag{12}$$

which equals the sample variance of T stochastic forward passes through the neural network plus the inverse model precision τ, which is a constant that is computable. Therefore, dropout is not only useful during training, but also for testing.

3 Besides Dropout

3.1 DropConnect

Instead of randomly dropping the activation out, it is quite natural to think about randomly dropping the connection/weights out. This leads to the idea of DropConnect [7], which is also a form of neural network regularization.

In Dropout, during each training iteration we have a different "thinned" network architecture, but all the connections within this "thinned" architecture are preserved. In DropConnect, each neuron is

likely to be always active because it usually has many incoming connections. However during each training iteration, each unit is receiving information from different sets of lower level neurons.

At inference time, instead of multiplying each outgoing weight by p, DropConnect needs to sample from a Gaussian and then average. During training, the number of Bernoulli random variables to be drawn is likely to be much larger than Dropout (for a fully connected layer with m input and n output, mn vs $m + n$). Therefore, although [7] showed slightly better performance over Dropout, DropConnect is a more cumbersome technique than Dropout.

3.2 Batch Normalization

Batch Normalization [8] is emerging as a favorable option over dropout, as it is used in all the state-of-the-art models on ILSVRC [9] [10] [11].

Batch Normalization builds on the idea that learning is easier when the distribution of the inputs is fixed. It has long been known that network training converges faster if the inputs are normalized and decorrelated [12] [13], and since the next layer can be seen as a network built upon the activations of the current layer, we should try to fix the activation distribution of all the intermediate layers.

During each training iteration, we only observe a mini-batch, which prohibits computing the normalization exactly (which is computationally expensive and requires access to the entire training set). So the idea is treat the scaling factor γ and bias β as trainable parameters:

1. $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$ (mini-batch mean)
2. $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_b)^2$ (mini-batch variance)
3. $\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$ (normalization)
4. $y_i = \gamma\hat{x_i} + \beta = BN_{\gamma,\beta}(x_i)$ (scale and shift)

It can be shown that the cost function is differentiable with respect to γ and β. BN allows much higher learning rate, less careful initialization, and acts as a regularizer.

4 Conclusion

Regularization is an important component in deep learning research. Since the recent resurgence of deep learning, dropout has been the most common choice due to its effectiveness and simplicity, although in the past year Batch Normalization is starting to draw much attention.

We know overfitting happens when the number of trainable parameters is much larger than the amount of data. So some people may argue that when we have enough data to fit into current models, we will no longer need these annoying regularization techniques. But at that time, what we should do is to increase the capacity of the network, and we will need regularization again. This comes back to the discussion of depth [14]. Although deep models may not be utilizing all its capacity, so is the case with human brain, which clearly has huge numbers of parameters compared to the data that it receives.

References

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[3] P. Baldi and P. Sadowski, "The dropout learning algorithm," *Artificial intelligence*, vol. 210, pp. 78–122, 2014.

[4] S. Wang and C. Manning, "Fast dropout training," in *Proceedings of the 30th International Conference on Machine Learning*, pp. 118–126, 2013.

[5] S. Wager, S. Wang, and P. S. Liang, "Dropout training as adaptive regularization," in *Advances in neural information processing systems*, pp. 351–359, 2013.

[6] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *arXiv preprint arXiv:1506.02142*, 2015.

[7] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058–1066, 2013.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[10] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv preprint arXiv:1512.00567*, 2015.

[11] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[12] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller, "Neural networks-tricks of the trade," *Springer Lecture Notes in Computer Sciences*, vol. 1524, no. 5-50, p. 6, 1998.

[13] S. Wiesler and H. Ney, "A convergence analysis of log-linear training," in *Advances in Neural Information Processing Systems*, pp. 657–665, 2011.

[14] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in *Advances in neural information processing systems*, pp. 2654–2662, 2014.