Making Dropout Invariant to Transformations of Activation Functions and Inputs

Lei Jimmy Ba
University of Toronto
jimmy@psi.utoronto.ca

Hui Yuan Xiong
University of Toronto
hui@psi.utoronto.ca

Brendan Frey
University of Toronto
frey@psi.utoronto.ca

Abstract

The dropout learning algorithm randomly sets the activities of hidden units and inputs to zero during training. While highly successful, dropout is not invariant to transformations of activation functions and inputs, raising the question of whether it is suboptimal. To eliminate this arbitrary dependence, we introduce 'invariant dropout', which has one extra parameter for each input or hidden unit, that translates the activity so that the dropout value of zero is not tied to a particular activation level. The invariant dropout learning algorithm is very simple to implement and is computationally efficient. We show that invariant dropout can be used to achieve consistently lower error rates than regular dropout on the MNIST, CIFAR-10, SVHN and MAS datasets. To explore whether invariant dropout can be successfully combined with other methods, we used it to train maxout networks and again observed reductions in error rates. Interestingly, invariant dropout can also be viewed as minimizing unnecessary variance in the training cost function, equipping each hidden unit with a different input bias for each network in a Bayesian ensemble, and using a population average as the dropout level so as to prevent extreme co-adaptation.

1 Introduction

Recent breakthroughs in algorithms for training large neural networks have enabled them to achieve unprecedented accuracy for a wide variety of tasks [1][2][3]. One method that has been shown to work remarkably well is dropout, in which the activities of hidden units and inputs are randomly set to zero during training [4]. It has been shown that dropout is an effective regularizer specifically well-suited to prevent overly complex co-adaptation of hidden units, which can be used to fit spurious relationships in the limited training dataset. Consequently, dropout can be used to train huge models, with millions of parameters. Additionally, each training example can be viewed as providing gradients for a different, randomly sampled network architecture, so that the final neural network efficiently represents a huge ensemble of neural networks, with good generalization capability.

Experimental results on several tasks show that dropout frequently and significantly improves the classification performance of deep architectures. Injecting noise for the purpose of regularization has been studied previously, but in the context of adding noise to the inputs [5] and to network components [6]. Different lines of analysis shed light on dropout as a regularizer and have been used to propose new methods [7, 8]. Additionally, dropout has been extended in various ways, including using a stochastic belief network to adjust the dropout probability depending on the input [9], dropping out connections between units instead of units themselves [10], and integrating Gaussian approximations to speed up training [11].

Despite the success of dropout, a troubling question remains: When applied to networks with different input formulations and non-linear activation functions, isn't a dropout value of zero arbitrary and therefore possibly suboptimal? From the perspective of deeper layers, a hidden unit or an in-

Activation function	$g^{-1}(0)$
ReLU	$w \cdot x < 0$
Sigmoid	$w \cdot x = -\infty$
Tanh	$w \cdot x = 0$
Maxout	$\max(w_i \cdot x) = 0$

Table 1: Popular activation functions and the input that make them zero

put feature being dropped out is equivalent to the hidden activation value or the input feature value being zero. This creates an implicit and unalterable tie between activation and input features being zero and the dropout method. As shown in table 1, different activation functions produce zeros with vastly different inputs and probabilities. In addition, depending on the representation used for input features, zero has different meanings. Regardless of these differences, regular dropout is always tied to the value of zero and thus not invariant to additive transformations of input and activations functions.

Indeed, it has been previously noted that if hidden activations are "dropped in" to the value of one instead of zero, the method does not perform well on datasets that standard dropout performs well on [12]. While this suggests that a dropout value of zero is preferable, this is not always the case. We found that for the MNIST dataset, if the numerical encoding of ink and background is inverted, performing dropout achieves an error rate of 1.32%, which is significantly worse than the error rate of 1.02% that is achieved with the normal encoding and comparable to not using dropout at all. These results suggest that using a standard dropout value of zero may be suboptimal.

Here, we introduce a generalization of dropout that is invariant to additive transformations of input encodings and activation functions. This is achieved by introducing a learnable invariance parameter for each activity, which is used to translate the activity so that units can flexibly adjust their effective dropout levels. Below, we analyze how the dropout learning method is sensitive to these additive transformations and introduce the invariant dropout technique. Then, we report experimental results showing that when we applied invariant dropout to several datasets and in combination with another technique, maxout, we observed consistent improvements in classification accuracy. We conclude with a discussion in which we show that invariant dropout can be viewed as reducing a variance penalty in the training cost function, equipping each hidden unit with a different input bias for each network in a Bayesian ensemble, and using a population average as the dropout level so as to prevent extreme co-adaptation.

2 Invariant Dropout

We denote the input to hidden unit j of a deep neural network by z_j during training. Assume it is connected to input features and other hidden units $\{a_i : i < j\}$ by weight vector w_{*j} and a bias b_j. During regular dropout training, a_i is randomly set to zero with a mask m_i:

$$z_j = \sum_{i:i<j} m_i w_{ij} a_i + b_j. \tag{1}$$

Examining the above expression, we observe that the regular dropout training procedure is not invariant to additive shifts of the inputs or activation functions. For example, suppose the j-th hidden unit is connected to the k-th input feature a_k with weight w_{kj}. If we used a shifted version $\tilde{a}_k = a_k + \phi$ instead of a_k for dropout training, we obtain

$$\tilde{z}_j = \sum_{i:i<j} m_i w_{ij} a_i + m_k w_{kj} \phi + \tilde{b}_j. \tag{2}$$

For networks trained without dropping out the k-th input (m_k is always one), $\tilde{b}_j$ can take on the value $b_j - w_k\phi$ to obtain an equivalent network during both training and testing. Similarly, when m_k is randomly set to zero with probability $1 - p$, setting $\tilde{b}_j = b_j - pw_k\phi$ can account for the

difference in the expected value of the input to the next layer:

$$\mathrm{E}[\tilde{z}_j] = \sum_{i:i<j} p w_{ij} a_i + p w_{kj} \phi + \tilde{b}_j = \mathrm{E}[z_j]. \tag{3}$$

$$\tag{4}$$

However, the variance of z_j, which is the sum of the variances of the independently distributed Bernoulli random variables, is different:

$$\mathrm{Var}[z_j] = \mathrm{Var}[m_k w_{kj} a_j] + \sum_{i:i<j, i\neq k} \mathrm{Var}[m_i w_{ij} a_j], \tag{5}$$

$$\mathrm{Var}[\tilde{z}_j] = \mathrm{Var}[m_k w_{kj} (a_j + \phi)] + \sum_{i:i<j, i\neq k} \mathrm{Var}[m_i w_{ij} a_j]. \tag{6}$$

The difference between them is:

$$\mathrm{Var}[z_j] - \mathrm{Var}[\tilde{z}_j] = p(1-p) w_{kj}^2 \phi(\phi - 2a_j). \tag{7}$$

From the above expression, it is clear that shifting an input feature by ϕ affects the variance of all hidden units connected to it during dropout training and that in the regular dropout framework, no parameter changes can correct for this effect. Furthermore, this effect is unlike changing the injected noise level for that particular feature, because the change in variance is dependent on feature value a_j. The additive transformation ϕ can increase the variance for some training cases but decrease the variance for other training cases. Since dropout relies on the variance of the activity for regularization, translations of the input or of the activation function will change the behaviour of dropout.

To compensate for the dependence of dropout on translations, we introduce a learnable "invariance parameter" β_j associated with activity a_j. This parameter can be thought of as defining a new non-zero dropout level, or as translating the activity a_j, potentially correcting for an implicit translation. We follow the latter approach:

$$a_j = m_j \Big(g\big(\sum_{i:i<j} w_{j,i} a_i \big) + \beta_j \Big). \tag{8}$$

The invariance parameter can also be thought of as a post-activation function bias, in contrast to the regular pre-activation function bias. For a linear activation function, the invariance parameter is unneccesary. Note that the invariance parameter is dropped out at the same time as the activity. Like standard dropout, when unit j is dropped, its activity is set to zero, but unlike standard dropout, when unit j is not dropped, its activity is shifted by the invariance parameter.

As in the original dropout technique, training proceeds by stochastically dropping out activations and backpropagating derivatives, as described below. To process an input at test time, the stochastic feedforward process is replaced by taking the expectation of equation 8:

$$\mathrm{E}[a_j] = p \Big(g\big(\sum_{i:i<j} w_{j,i} a_i \big) + \beta_j \Big). \tag{9}$$

As with regular dropout, scaling the activity by p is equivalent to adjusting the weight between a_j and the next hidden layer [13]. In addition, the invariance parameter can be absorbed into the biases to hidden units in the next layer during test time. In the "mean network", unit k has new weights $p w_{k,i}$ and a new bias that equals the old bias plus $p \sum_i w_{k,i} \beta_i$, where i indexes the incoming units.

2.1 Prior on the invariance parameters

We found that invariant dropout consistently improves performance or leaves it unchanged, but we found that sometimes, performance can be improved slightly by regularizing the invariance parameters. We consider two different priors for the invariance parameters. As with other neural network parameters, the invariance parameters can be regularized using various kinds of weight decay, so that small values are preferred or so that a small variance in the invariance parameters is preferred. Also, topological weight sharing mechanisms can be used. For a convolutional neural network (CNN), the number of parameters is relatively small, but the number of hidden units in the feature maps can be very large. Just as the weights of a hidden unit in a CNN are replicated across the receptive field, we can replicated the invariance parameters across the receptive field. This corresponds to using one invariance parameter for each feature map, so the total number of invariance parameters is equal to the number of filters.

	standard dropout	invariant dropout on input only	invariant dropout on hidden only	**invariant dropout**	running average
original	1.01%	1.01%	1.00	**0.91%**	1.00
inverted	1.32%	1.17%	1.23	**0.94%**	0.98

Table 2: Classification error rate on the original and inverted MNIST.

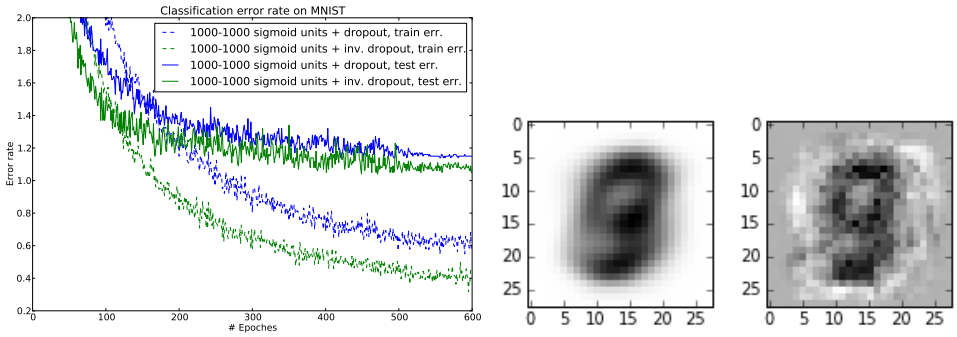

Figure 1: (left) Training and test performance on MNIST images. (centre) The average inverted mean of the input images. (right) The learnt invariance parameters for the input layer.

2.2 Toy example using MNIST images with an inverted encoding

As an illustration of how dropout is not invariant to data preprocessing, we inverted the encoding of MNIST images so that 0 encodes ink and 1 encodes white space. We trained a fully-connected ReLU network with two hidden layers of 1000 units using dropout and obtained an error rate of 1.32% on the test data. This is much higher than the typical error rate of 1.01% achieved when the original MNIST images were used. In contrast, when the invariant dropout technique is applied, an error rate around 0.94% is achieved on both the inverted and original datasets. We also compared our method with a model that keeps a running average of the input and hidden activations. The running average is subtracted from the hidden units activations before they are passed to the next layer. As shown in Table 2, the additional invariance parameter can improve the learning in addition to keep track of the average activations and thus achieved better performance than running average method.

3 Experimental Results

To test the utility of invariant dropout, we investigated classification performance by applying the method to several datasets, including MNIST, CIFAR-10, SVHN (street view house numbers, [14]) and MAS (mouse alternative splicing, [15]).

We used the ReLU activation function, $g(x) = \max(0, x)$, for the rest of the results reported here. We optimized the model parameters using stochastic gradient descent with the Nesterov momentum technique [16]. The momentum coefficient is 0.99 for fully connected networks and 0.9 for CNNs. In addition, we found that adding weight decay on the invariance parameters generally improve the performance, although the improvement is not very sensitive to the degree of weight decay. As a result, we used 0.1 weight decay on the invariance parameter for the experiments presented here.

We used the publicly available *gnumpy* [17] , *cuda-convNet* [1] libraries to implement our fully connected and the convolution models.

3.1 MNIST

The MNIST handwritten digit dataset is generally considered as a well-studied problem, which offers the ability to ensure that new algorithms produce sensible results when compared to the many other techniques that have been benchmarked. It consists of ten classes of handwritten digits, ranging from 0 to 9. There are 60,000 training images and 10,000 test images. Each image is 28×28 pixels in

Model (no augmentation)	Test Err.
DNN 2000-2000 + dropout	57.8%
ReLU CNN[13] 64c-p-64c-p-64c-p-16lc + dropout	15.6%
ReLU CNN[20] 64c-p-64c-p-128c-p-fc + dropout stochastic pooling	15.13%
ReLU CNN 128c-p-128c-p-128c-p-1000fc + dropout	12.3%
ReLU CNN 128c-p-128c-p-128c-p-1000fc + invariant dropout	**11.7%**

Table 3: Classification error rate on CIFAR-10: c, convolution layer; p, pooling layer; lc, locally connected layer; fc, fully connected layer

size. Following the common convention, we randomly separate the original training set into 50,000 training cases and 10,000 cases used for validating the choice of hyper-parameters. We concatenate all the pixels in an image in a raster scan fashion to create a 784-dimensional vector. The task is to predict the 10 class labels from the 784-dimensional input vector. In our experiments, we made engineering choices that are consistent with previous publications in the area, so that our results are comparable to the literature. Table 2 reports the test classification error rates for the MNIST data, as well as the inverted MNIST data described above. In both cases, invariant dropout provides a significant reduction in error rate. We found that the invariant dropout yield the similar performance improvement to sigmoid and tanh units.

3.2 CIFAR-10

We also explored the use of invariant dropout in the context of convolutional neural networks (CNNs), by examining the CIFAR-10 object recognition task [18]. CIFAR-10 consists of a set of natural images from 10 different object classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The dataset is divided into 50,000 train and 10,000 test images. Each image is 32×32 pixels in 3 color channels, yielding raster-scanned vectors with 3072 dimensions. We prepared the data by subtracting the mean and dividing the standard deviation of each image vector to perform global contrast normalization. We then applied zero-phase whitening (ZCA) to whiten the normalized images following the same procedure as in [19]. This is a crucial step for getting high performance on CIFAR-10; ZCA whitening leads to a 2% to 3% improvement in classification accuracy when standard dropout is applied to the input.

To get good performance on CIFAR-10, variability in the locations of object parts needs to be taken into account and this has been successfully accomplished using CNNs, with several layers of convolutional, pooling and non-linear units. Previous results indicate that dropout is important to prevent over-fitting [20], so here we explore the use of invariant dropout. We used CNN architectures that are very similar to the ones used in [20], but without contrast normalization layers. Our architectures have three alternating layers of convolution and max pooling of 128 feature maps, each followed by a fully connected layer of 1000 hidden units. The receptive field is 5×5 with zero-padding of 2 pixels around the image, so that the feature map is the same size as the input image. The max pooling region is 3×3 with an overlap stride of 2, so the spatial resolution is halved after each pooling layer. We use the ReLU activation function. Dropout is applied to all layers, including the input. This high performance CIFAR-10 model has a moderate size that takes about 10 hours to train on a NVIDIA GTX 580. By simply extending the existing dropout method to our invariant method, we obtained similar or better performance than standard dropout models trained for multiple days. The speed up is prominent for large convolutional neural networks.

Invariant dropout is compared to normal dropout, using various architectures, in Table 3. Invariant dropout improves on the performance of the ReLU CNNs trained using regular dropout by 0.5%. This is observation is consistent with the results from MNIST.

3.3 Street View House Numbers (SVHN)

For the SVHN dataset, we achieved the best result to our knowledge among models without augmentation. The SVHN dataset [14] consists of 32×32 3-channel images of cropped digits taken from pictures of house fronts. We use 604,388 examples for training and 26,032 examples for testing. Similar to MNIST, the task is to classify an image as one of the ten digits, but this is a more challenging dataset due to natural backgrounds, lighting variation, and highly variable resolution.

Model (no augmentation)	Test Err.
ReLU CNN[13] 64c-p-64c-p-128c-p-fc-fc + dropout	2.78%
ReLU CNN[20] 64c-p-64c-p-128c-p-fc + dropout stochastic pooling	2.80%
ReLU CNN 128c-p-128c-p-128c-p-1000fc + dropout	2.80%
ReLU CNN 128c-p-128c-p-128c-p-1000fc+ invariant dropout	**2.68%**

Table 4: Classification error rate on SVHN: c, convolution layer; p, pooling layer; lc, locally connected layer; fc, fully connected layer

Partitions	No dropout	Regular dropout	Invariant dropout
1	411.56	445.84	**477.07**
2	464.39	476.74	**484.58**
3	429.07	**485.09**	471.72
4	414.77	460.88	**462.91**
5	433.94	471.10	**483.70**
6	429.43	450.85	**467.72**

Table 5: Results on the splicing dataset. The results are in a code quality measure, which is the KL divergence between the prediciton and the target compared to the KL divergence between the prediction and a naive guesser [15]. It can be veiwed as a log likelihood of soft targets.

Following [21], we formed a validation set of 6000 images by randomly sampling 400 images per class from the training set and 200 per class from the extra set, and these were used for selecting the learning rate and for early stopping. The models are trained using the remaining 598,388 training images, which were pre-processed using global contrast normalization (see above). We used the same CNN architecture as in our CIFAR-10 experiments and dropout was applied to the input as well as all hidden layers. Learning took around 27 hours on a GTX 580 GPU. We compare invariant dropout to standard dropout for various architectures in Table 4. Invariant dropout achieves the lowest error rate.

3.4 Tissue dependent alternative splicing in mouse (MAS)

Alternative splicing (AS) is a biological process in which protein coding regions, or exons, are selectively skipped during RNA splicing. This process enables different protein isoforms to be produced by a single gene. AS is a highly regulated process that may depend on tissue type, environmental factors and developmental stage. Misregulation of AS have been estimated to be associated with up to 50% of human diseases. In this AS dataset,1014 RNA features were extracted from regions near 3665 exons and the task is to predict tissue-dependent splicing differences measured in mouse using microarray experiments. Each exon is treated as a training datapoint. The targets are represented as multinomial distributions of three outcomes. The dataset is very sparse and the target classes are highly biased. For a more detailed explanation of the AS and this dataset, please refer to [15, 22]. Using a neural network with one hidden layer consists of 1000 RELU hidden units, we compared the invariant dropout, regular dropout to a baseline neural network trained with no dropout. All networks used the same random initialization, learning rates, momentum schedule, weight decay, early stopping criteria and other parameters. To ensure a fair comparison, these parameters were not tunned to amplify the improvement obtained by the invariant dropout. We also experimented with other parameters and observed that the results presented here are representative of other settings.

From results presented in 5, we observe that dropout greatly improves the generalization of the neural network, but adding the extra invariance parameters during training usually further improves dropout performance. On average, dropout increases the prediction performance from 431 bits to 465 bits and using the invariance dropout increases it further to 474 bits, which represents an additional 26% gain in performance.

3.5 Combining invariant dropout with maxout

We wondered whether invariant dropout could be used in conjunction with the recently proposed maxout activation function, which works well when coupled with regular dropout [23]. We incorporated our invariant dropout technique into the maxout learning software provided at *pylearn2* [24]

Model (no augmentation)	Test Err.
CIFAR-10 Maxout CNN[19] 96c-p-192c-p-192c-p-500fc + dropout	11.7%
CIFAR-10 Maxout CNN 96c-p-192c-p-192c-p-500fc+ invariant dropout	**11.5%**
SVHN Maxout CNN[19] 64c-p-128c-p-128c-p-400fc + dropout	2.47%
SVHN Maxout CNN 64c-p-128c-p-128c-p-400fc + invariant dropout	**2.33%**

Table 6: Invariant dropout used in conjunction with maxout on CIFAR-10 and SVHN: c, convolution layer; p, pooling layer; lc, locally connected layer; fc, fully connected layer

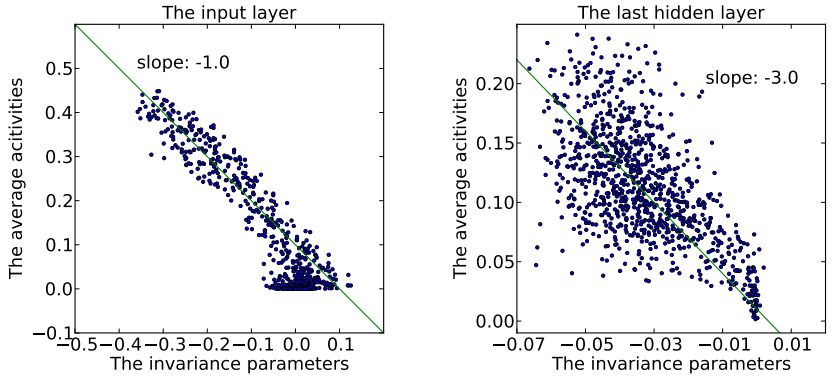

Figure 2: (left) Input values averaged across the training set versus their invariance parameters. (right) Hidden activities averaged across the training set versus their invariance parameters.

library and applied it to the CIFAR-10 and SVHN datasets. The accuracies achieved by dropout and invariant dropout are provided in Table 6. In both cases, invariant dropout improves the accuracy. Also, it is interesting to note that the non-maxout network trained on the CIFAR-10 data using invariant dropout (see above) achieves the same error rate as the maxout network (11.7%), but does so with less than half the number of parameters.

4 Discussion

Above, we motivated invariant dropout as way of making dropout invariant to input encodings and transformations of activation functions. Here, we explore invariant dropout from other perspectives.

Variance reduction. It turns out that invariant dropout adaptively adjusts the variance of hidden units introduced by random dropout masks. Following the notation used above, the input to a hidden unit for training case t is $z = \sum_i m_i^t w_i(a_i^t + \beta_i) + b$. For a dropout probability of $1-p$, the expected input to the hidden unit is $\mathrm{E}[z] = p\sum_i w_i a_i^t + p\sum_i w_i\beta_i + b$. In expectation, the traditional bias and the invariance parameters can be combined. However, the overall log likelihood is nonlinear in z, so the variance and other higher-order statistics of z can have significant effect on the expected log likelihood under random masks. Here we focus on the variance. Because activations are dropped out independently, we have: $\mathrm{Var}[z] = \sum_i \mathrm{Var}(m_i^t w_i(a_i^t + \beta_i))$. The i-th term simplifies to

$$w_i^2(a_i^t + \beta_i)^2 p(1-p). \tag{10}$$

Standard dropout is equivalent to fixing β_i at zero, in which case the variance depends on shifts of the activation function or input features and may result in poor performance. For example, when the inverted encoding of ink is used for the MNIST data (see above), the inverted encoding has constant of 1 added to the input pixels. For a particular pixel, this increases the variances of connected hidden units for training cases that have no ink on that pixel and decreases the variance of those that have ink on that pixel. Invariant dropout overcomes the arbitrariness of input offsets. Also, different types of hidden activation functions, as shown in table 1, have different shapes and thus have effectively different offsets. These offsets can affect the variance of higher hidden units during dropout training. Invariant dropout compensates for this effect.

We postulated that β_i learnt by invariant dropout will be close to, but not exactly equal to, the average activity of the hidden unit, for the following reasons. First, if training converges to a local minimum, the log likelihood function is likely to be convex around the expected value of z for most training cases. By Jensen's inequality, reducing the variance of z will decrease the expected cost for a single training case. Because β_i is shared across training cases, the network cannot make $(a_i^t + \beta_i)^2$ zero for all training cases. If the the cost function is quadratic and curvature is the same for all training cases, the optimal β_i is the one that minimizes $\sum_t (a_i^t + \beta_i)^2$, which is the average of a_i^t across training cases. However, since the cost function is not quadratic and the curvature is not constant for different z and different training cases, the β_i will only approximate the average of a_i^t. The second and more intuitive reason is that the network needs to work well when the hidden activation is around 0 and when the hidden unit is on, it should provide extra information about the particular training example, but not a non-zero bias. Therefore, the optimal β_i should represent the additive inverse of the marginalized activity of the hidden unit. To test our hypothesis, we used invariant dropout to train several networks on MNIST data. We observed that the learned β_i values have strong negative correlation with the average activity of the corresponding input or hidden unit (Figure 2), confirming our hypothesis.

Bayesian ensemble. Another way to understand invariant dropout is that it enables different networks that come from sampling the masks to have different biases in a restricted family, while sharing other unit-to-unit weights. Consider an input layer or a hidden layer with activities a_i. The input signal for a hidden unit in the next layer is $z = \sum_i m_i w_i a_i + b$. Dropout optimizes the average cost of an ensemble of 2^n networks corresponding to the exponentially many settings of $\mathbf{m}$. The key for the success of dropout is that these 2^n networks are forced to share the weights $\mathbf{w}$, so that the final set of weights produces hidden units that are generally helpful in a wide range of context established by the stochastic presence of other hidden units. However, in standard dropout, a scaler bias b is shared across the ensemble of 2^n networks, so that different groupings of hidden units are forced to use the same bias in the next layer. This restriction seems excessive and may lead to sub-optimal weights in order to compensate for effects produced by insufficient flexibility in the bias, especially when the features are not zero-mean.

One way to view invariant dropout is that it relaxes the above requirement and allows each unit to contribute an additive bias to the *next* layer. Invariant dropout enforces sharing of unit-to-unit weights across members of the ensemble, but unlike standard dropout, it allows the bias to depend on the sampled network structure. Specifically, the bias is a linear function of the mask: $z = \sum_i m_i w_i a_i + \tilde{b}(\mathbf{m})$, where $\tilde{b}(\mathbf{m}) = b + \sum_i m_i w_i \beta_i$. Note that there is still parameter sharing in the biases, since the 2^n possible $\tilde{b}(\mathbf{m})$ are parameterized by only n extra invariance parameters.

Evolution: Preventing extreme co-adaptation. Another interesting perspective for invariant dropout is that it more closely resembles the mechanism of sexual reproduction, which was originally used to motivate dropout. As first noted by Hinton *et. al.*, dropout and sexual reproduction have a striking resemblance: half of the hidden units, or genes, are omitted in each generation [4, 25]. As a result, hidden units or versions of a gene evolving under this rule tend to be more robust because they cannot overly co-adapt with other specific hidden units or specific versions of another gene. This may increase robustness of the population and may partially explain why complex organisms employing sexual reproduction are so successful. In sexual reproduction, when an allele of gene A from one parent is passed on to a child, it cannot count on a specific allele of gene B from the other parent to also be passed on and instead receives a random allele. Therefore, gene A needs to work well with the 'population average' of gene B. In the proposed invariance dropout technique, this 'population average' corresponds to the negative invariance parameter, while in standard dropout, it is arbitrarily set to zero.

5 Conclusion

Examination of the dropout learning algorithm indicates that it is sensitive to translations of input levels and hidden activations. We proposed 'invariant dropout', a learning algorithm that accounts for these translations using auxiliary parameters. After training, these invariance parameters are combined to form a regular feed forward neural network for use at test time. It is a simple and computationally efficient method that is motivated from multiple perspectives including shift invariance to variable activations, dropout variance adjustment, Bayesian ensembles and principles of evolu-

tion. Interestingly, when applied on challenging real-world problems, it usually produces networks that perform significantly better than networks trained using standard dropout, and it often obtains state-of-art results. For example, invariant dropout achieves higher accuracy than any published result on the SVHN dataset, to our knowledge.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks.,"

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6645–6649, IEEE, 2013.

[4] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[5] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural computation*, vol. 7, no. 1, pp. 108–116, 1995.

[6] J. Sietsma and R. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, no. 1, pp. 67–79, 1991.

[7] S. Wager, S. Wang, and P. Liang, "Dropout training as adaptive regularization," in *Advances in Neural Information Processing Systems*, pp. 351–359, 2013.

[8] P. Baldi and P. J. Sadowski, "Understanding dropout," in *Advances in Neural Information Processing Systems*, pp. 2814–2822, 2013.

[9] J. Ba and B. Frey, "Adaptive dropout for training deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 3084–3092, 2013.

[10] 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.

[11] S. Wang and C. Manning, "Fast dropout training," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 118–126, 2013.

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

[13] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[14] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, vol. 2011, 2011.

[15] Y. Barash, J. A. Calarco, W. Gao, Q. Pan, X. Wang, O. Shai, B. J. Blencowe, and B. J. Frey, "Deciphering the splicing code," *Nature*, vol. 465, no. 7294, pp. 53–59, 2010.

[16] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *Proc. 31st International Conference on Machine Learning*, 2013.

[17] T. Tieleman, "Gnumpy: an easy way to use gpu boards in python," *Department of Computer Science, University of Toronto*, 2010.

[18] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[19] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proceedings of The 30th International Conference on Machine Learning*, pp. 1319–1327, 2013.

[20] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013.

[21] P. Sermanet, S. Chintala, and Y. LeCun, "Convolutional neural networks applied to house numbers digit classification," in *Pattern Recognition*, pp. 3288–3291, IEEE, 2012.

[22] H. Y. Xiong, Y. Barash, and B. J. Frey, "Bayesian prediction of tissue-regulated splicing using rna sequence and cellular context," *Bioinformatics*, vol. 27, no. 18, pp. 2554–2562, 2011.

[23] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.

[24] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio, "Pylearn2: a machine learning research library," *arXiv preprint arXiv:1308.4214*, 2013.

[25] A. Livnat, C. Papadimitriou, J. Dushoff, and M. W. Feldman, "A mixability theory for the role of sex in evolution," *Proceedings of the National Academy of Sciences*, vol. 105, no. 50, pp. 19803–19808, 2008.