Regularization of Neural Networks using DropConnect

We have implemented the paper DropConnect , a generalization of Dropout, for regularizing large fully connected layers within networks. DropConnect randomly sets certain weights to zero based on a probability p. It differs from Dropout , which sets randomly selected activations to 0. For this project, we have implemented DropConnect and Dropout in theano **without** additional libraries on top such as lasagne and compared their performance along with the regular CNN performance according to the paper.

We have used 3 datasets to test our implementation of DropConnect and Dropout - MNIST,SVHN,CIFAR-10.

The core skeletal structure and logic of the code is as follows -

We first load the dataset. On the dataset, we do augmentation(rotation,cropping,scaling,flipping). After that, we create our feature extractor. This is generally a CNN/MLP which has been implemented in the paper. On accumulating the features, this is sent to a fully connected layer which is either a dropout/dropconnect layer. For benchmarks, we also do not use this layer. Finally the outputs from this layer is sent to a logistic softmax layer which classifies the training example.

The training example

We reduce the learning by 0.5-0.95 if the validation error does not improve for more than 10 epochs. We also implement momentum so that even the previous gradient along with the current gradient has been used for weight updates. This helps in getting out from the local minima.

Two main regularization methods have been implemented:

1. DropOut
2. DropConnect

DropOut
Each element of a layer's output is kept with probability p, otherwise being set to 0 with probability 1-p. When dropout is applied to a fully connected network layer

$$r = m * a(Wv)$$

Where * denotes element wise product and *m* is a binary mask of size d with each element in it drawn from a Bernoulli distribution.
RELU activation function is used in the current implementation of Dropout.

DropConnect

DropConnect is a generalization of DropOut in which each connection rather than each output unit can be dropped with probability 1-p. DropConnect introduces dynamic sparsity on the weights rather than the outputs.

For DropConnect layer the output is given as

$r = a((M * W)v)$

Where M is a binary matrix encoding the connection information and each element in this mask is drawn from a bernoulli distribution. It is responsible for instantiating a different connectivity for each example seen. The biases are also masked out during training.

Feature Extractor: v, where v are the output features, x is the input data to the overall model. W_g are parameters for the feature extractor. A CNN with W_g being the convolutional filters(and biases) of the CNN, is used.

DropConnect Layer: $r = a(u) = a((M * W)v)$ where W is a fully connected weight matrix, a is a nonlinear activation function.

SoftMax Classification Layer: $o = s(r;W_s)$ takes as input r and uses parameters W_s to map this to a k dimensional output (k being the number of classes). The below explains the algorithms in detail as in the paper. During the training , we follow algorithm 1 and for validation and testing , we follow algoithm2 with the intuition that a large guassian sampling would lead us to a good approximation.

Algorithm 1 SGD Training with DropConnect

Input: example x, parameters θ_{t-1} from step $t-1$, learning rate η

Output: updated parameters θ_t

Forward Pass:

Extract features: $v \leftarrow g(x; W_g)$

Random sample M mask: $M_{ij} \sim Bernoulli(p)$

Compute activations: $r = a((M \star W)v)$

Compute output: $o = s(r; W_s)$

Backpropagate Gradients:

Differentiate loss A'_θ with respect to parameters θ:

Update softmax layer: $W_s = W_s - \eta A'_{W_s}$

Update DropConnect layer: $W = W - \eta(M \star A'_W)$

Update feature extractor: $W_g = W_g - \eta A'_{W_g}$

Algorithm 2 Inference with DropConnect

Input: example x, parameters θ, # of samples Z.
Output: prediction u
Extract features: $v \leftarrow g(x; W_g)$
Moment matching of u:
$\mu \leftarrow E_M[u] \quad \sigma^2 \leftarrow V_M[u]$
for z $= 1 : Z$ **do** %% Draw Z samples
for i $= 1 : d$ **do** %% Loop over units in r
Sample from 1D Gaussian $u_{i,z} \sim \mathcal{N}(\mu_i, \sigma_i^2)$
$r_{i,z} \leftarrow a(u_{i,z})$
end for
end for
Pass result $\hat{r} = \sum_{z=1}^{Z} r_z/Z$ to next layer

Salient Features of the Implementaion:
We have also followed and written code for each of the experiments that the authors have followed for this implementation -

1. We have augmented the dataset by randomly selecting cropped regions from images wherever necessary in the experiments.
2. We have also introduced 15% scaling,rotation and flipping on the cropped images for augmentation purposes depending on the dataset.
3. We have trained independent networks and saved the values for the purpose of the voting technique wherein the ypred of the test is chosen based on the majority vote among all the different models,which has proven useful for the authors
4. Learning rate decay has been applied while training in which the learning rate is scaled in the range of 0.5-0.75 based on different experiments when the there is no increase in performance of validation error.
5. Momentum has been added in the update method so that even the previous gradient along with the current gradient has been used for weight updates. This helps in getting out from the local minima.

MNIST(Refer to MNIST Results for parameters) - Observe the training times and results in MNIST Result. Expanded version in notebooks,terminal outputs.

MNIST handwritten digit classification task consists of 28x28 black and white images, each containing a digit 0 to 9. There are 60000 training images and 10000 test images.
The NoDrop, DropOut and DropConnect experiments are performed on this dataset.

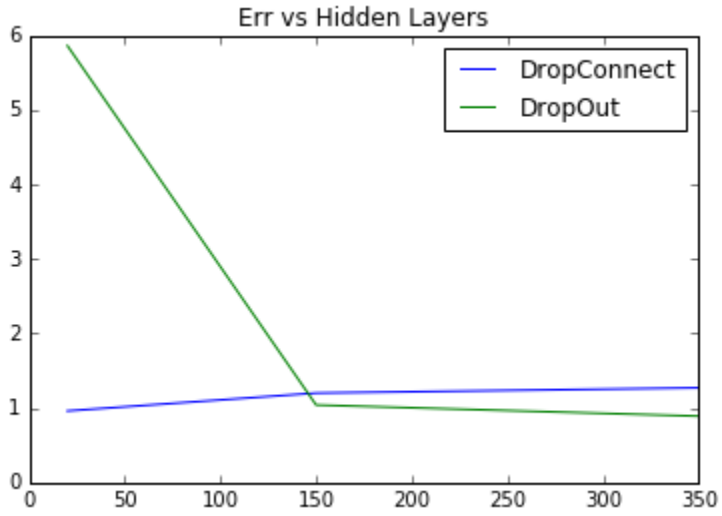
The training set is includes 2 variants apart from the original which are:

1. Train Set augmented with 24x24 cropped images
2. Train Set augmented with 24x24 cropped, scaled and rotated on the cropped images with 15% probability

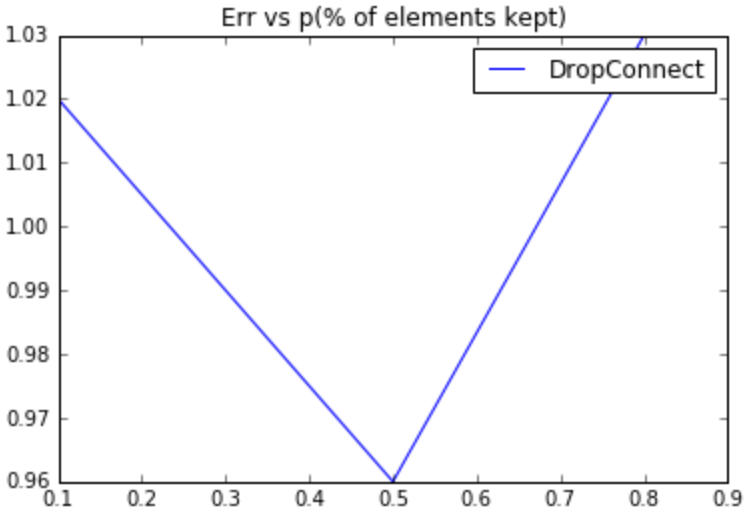
NoDrop, DropOut and DropConnect experiments are performed on all trainset variants.

Another experiment is performed where DropOut and DropConnect are experimented with different activation functions namely, tanh, sigmoid and relu.

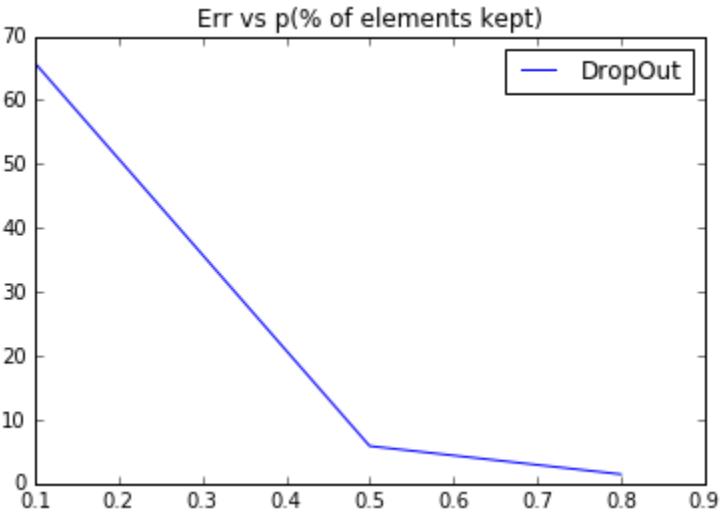
We also observe how DropConnect performs for different values p, i.e the probability that a weight is kept.



From the above graph, we see that Dropout error decreases with increasing the number of hidden units just like the paper. We also see that we get a similar graph to the paper for DropConnect wherein we see the error becoming slightly larger with the hidden units.



From the above graph, we see that Dropconnect error decreases and then increases with probability with the best result at p=0.5 similar to the one mentioned in the paper.



From the above graph, we see that Dropout error decreases with p as p increases. This result is slightly different as in the paper, the dropout error starts to increases slightly after a while.This could be due to the variability of the masks in terms of the random binomial sampling.

neuron	model	error
relu	NoDrop DropOut DropConnect	1.06 5.86 0.96

Sigmoid	NoDrop DropOut DropConnect	1.09 14.03 3.9
tanh	NoDrop DropOut DropConnect	0.97 2.41 4.4

model	Voting Error
NoDrop	0.91
DropOut	0.89
DropConnect	1.19

crop	Rotation scaling	model	error
no	no	NoDrop DropOut DropConnect	1.06 5.86 0.96
yes	no	NoDrop DropOut DropConnect	20.7(1.7 in next iter) 7.14 15.25
yes	yes	NoDrop DropOut DropConnect	1.47 5.56 4.78

From the above tables, we see that relu is the best in terms of test error for the dropconnect architecture.No drop in fact does best with tanh activation. Dropout also does best with tanh activation. We also introduce rotation and scaling just as in the paper. However we get worse results as can be seen from the above table. A point to be noted is that the results varied a lot in the iterations. For example, Dropout changed from 20.7 to 1.7. This is probably due to the random 15% sampling which leads to such variability. We would have liked more consistence in the results with augmentation.

CIFAR-10 Dataset:Observe the training times and results in CIFAR Result. Expanded version in notebooks,terminal outputs.

CIFAR-10 is a dataset of 32x32 RGB images in 10-classes with 50000 images for training and 10000 testing. The image is whitened before being used for training.

model	Error	Parameters
No Drop	39.830000 % The training process for function test_lenet3 ran for 15.73m 37.680000 % The training process for function test_lenet3 ran for 20.29m	learning_rate=0.01, n_epochs=200, nkerns=[32,32,64], batch_size=20, fullyconnected=128, activation=relu learning_rate=0.01, n_epochs=200, nkerns=[32,32,64], batch_size=20, fullyconnected=500, activation=relu
DropOut	49.490000 % The training process for function test_dropout3 ran for 24.36m 39.320000 % The training process for function test_dropout3 ran for 23.79m	learning_rate=0.01, n_epochs=200, nkerns=[32,32,64], batch_size=20, fullyconnected=128, activation=relu learning_rate=0.01, n_epochs=50, nkerns=[64,128,128], batch_size=20, verbose=True, fullyconnected=500, activation=relu learning_rate=0.01,

	The training process for function test_dropout3 ran for 21.11m 38.600000 %	n_epochs=50, nkerns=[64,128,128], batch_size=20, verbose=True, fullyconnected=500, activation=relu
DropConnect	epoch8 -> 46.8% (still running) The training process for function test_dropconnect3 ran for **378.65m** 47.480000 % Testerror 42.31 (@epoch 18 at the time of submission) 125.51 mins (without momentum updates)	learning_rate=0.01, n_epochs=200, nkerns=[64,128,128], batch_size=20, fullyconnected=500, activation=relu learning_rate=0.01, n_epochs=200, nkerns=[32,32,64], batch_size=20, fullyconnected=64, activation=relu Learning rate=0.1 nkerns=64,128,128 Batch size =20 Act relu Fullyconnected=128 p=0.5 Testerror 42.31 (@epoch 18 at the time of submission) 125.51 mins

We observe that for the CIFAR dataset once the dataset is whitened the test error improves drastically. Also this experiment runs for a long time and was pretty hard to tune the parameters. It was noticed that removing momentum updates gave better error at the same epoch. We feel we could got a better error with more training time, although with time constraints we have reported the error and the features as above. Please refer CIFAR results file for more documentation.

SVHN Dataset - Observe the training times and results in SVHN Result. Expanded version in notebooks,terminal outputs.

We trained our model on the SVHN dataset. But we trained the model without changing the raw dataset. The authors processed the images in the dataset using local contrast normalization. We did not perform local contrast normalization on the dataset but we augmented our dataset using the procedure specified in the paper.
Below is the description of the drop connect model that was used for the svhn dataset
3 CNN layers where tanh is used as activation followed by a DropConnect layer where relu is used as activation and finally the logistic regression layer.

model	error	parameters	Voting error
NoDrop	36.81 50.74 44.39 31.74 47.76 44.72	nkerns = 32,32,64 Learning rate=0.1 Batch size = 20 Fully connected layer units = 128 Activation = relu	---
DropOut	45.01 35.61 31.90	nkerns = 32,32,64 Learning rate=0.1 Batch size = 20 Fully connected layer units = 128 Activation = relu nkerns = 64,64,64 Learning rate=0.1 Batch size = 20 Fully connected layer units = 128 Activation = relu nkerns = 64,128,128 Learning rate=0.1 Batch size = 20 Fully connected layer units = 128 Activation = relu	---

DropConnect	34.78	nkerns = 32,32,64 Learning rate=0.1 Batch size = 20 Fully connected layer units = 128 Activation = relu	29.0392
	38.12	nkerns = 64,64,128 Learning rate=0.1 Batch size = 20 Fully connected layer units = 128 Activation = relu	
	26.41	nkerns = 64,128,128 Learning rate=0.1 Batch size = 20 Fully connected layer units = 128 Activation = relu	
	23.95	nkerns = 64,128,256 Learning rate=0.1 Batch size = 20 Fully connected layer units = 256 Activation = relu	
	30.67	nkerns = 128,128,128 Learning rate=0.1 Batch size = 20 Fully connected layer units = 128 Activation = relu	

All results and more can be found in SVHN Results file

We observe that Dropconnect performs better than dropout.Dropout performs better than the regular CNN. We also notice a sweet spot in tuning where for dropout the feature maps of 64,128,128 gives the best results. We also observe that in DropConnect,on increasing the number of feature maps, the performance starts to improve like in 64,128,256. We also see that on increasing fullyconnected layer to 256, the accuracy improves. We were unable to increase the batch size as it gave memory errors due to which we could not meet the state of the art in the paper. Also observe that a voting technique does a good way to normalize multiple models.

Conclusion -

We were extremely satisfied with implementing this paper as we had implemented close to everything the authors had done from the preprocessing stage. We missed out on local contrast normalization as we felt it was almost like implementing another paper. The one note about the learning is by implementing everything in theano, including benchmarks like dropout, we ran short of time in terms of getting exact results as in the paper. We were also limited by batch size which we couldn't give as 128 due to memory errors. Dropconnect takes a very very long time to run as can be seen from the running times and we could not tune enough to get the extreme state of the art accuracies observed In the paper. We however, managed to implement the paper as it is minus LCM normalization and also ran for every experiment in the paper bar norb and the second large cifar architecture.