Dropout, simple way to overcome overfitting

Introduction

Deep Neural Networks are very powerful machine learning tools but due to having a large number of parameters they generally are prone to overfitting. Also these large neural networks are slow to train, which makes it difficult to deal with the overfitting by aggregating the results at the time of testing.

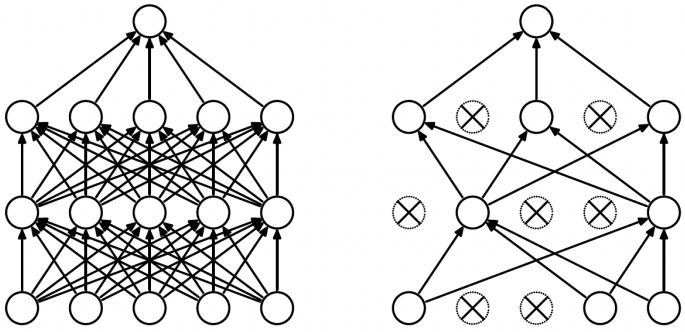
Dropout addresses both these problems. In dropout process randomly units are dropped during training time, according to some rule, which eventually prevents the units from co-adapting too much.

Apart from dropout there are many other ways to reduce overfitting, such as early stopping, L1 and L2 regularization, normalization.

In the following paper we will assess how well dropout performs as a regularizer to tackle overfitting proble

Model Description

If we consider a neural network with L hidden layers, indexing them as $l \in \{1,...L\}$, say dropout is applied in layer l, having l units then we draw independent bernouli samples of length l, denote it by $r^{(l)}$, and the bernouli samples are multiplied with the output layers, the bernouli variable consisting of only 1s and 0s ultimately lets some output layers pass and does not let others, while being multiplied. Thus we get a thinned out output layer, in all the layers where dropout is applied.



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

Image of a standard neural network and a thinned neural network after applying dropout

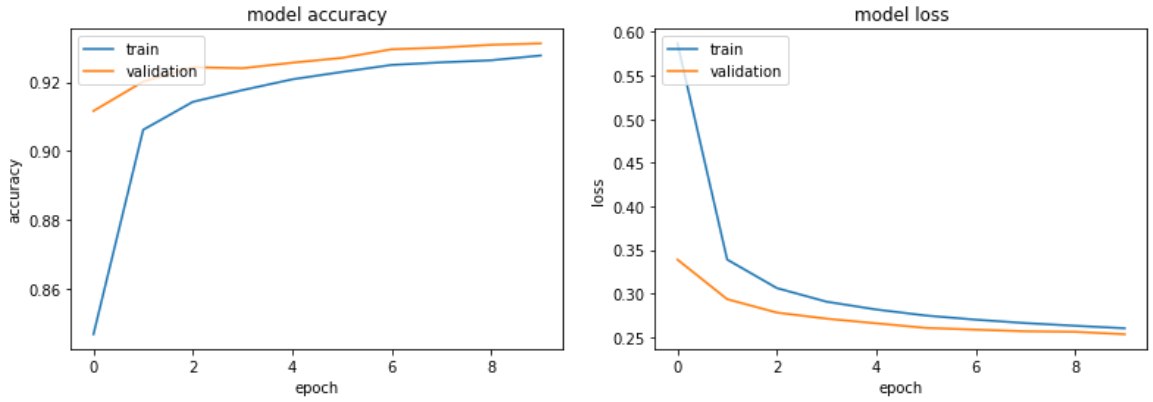
Learning process

Dropout neural networks can be trained using sgd similar to standard neural networks, but the only difference is in each mini-batch, different thinned networks are used by dropping out units, forward and backward propagation are used in these thinned networks, and the gradient of each parameter are averaged over thetraining cases of each thinned network, giving it a sense of aggregation which helps overcome overfitting, in fact each unit has a chance of dropping or not dropping if we use dropout so for a neural network with n units, there can be 2^n possible thinned network, so in reality dropout can aggregate exponentially large number of networks in a faster and better way. Also to improve the sgd process momentum, L2 regularization these methods can be used. Also using a max-norm constraint is very useful in dropout, and if we use a max-norm constraint we can also use larger learning rate, as the chance of jumping outside certain radius of convergence is lowered, which makes it faster to converge.

Results on the Mnist Dataset

In the following section we will visualize some results that we found, applying dropout and other methods on the MNIST dataset.

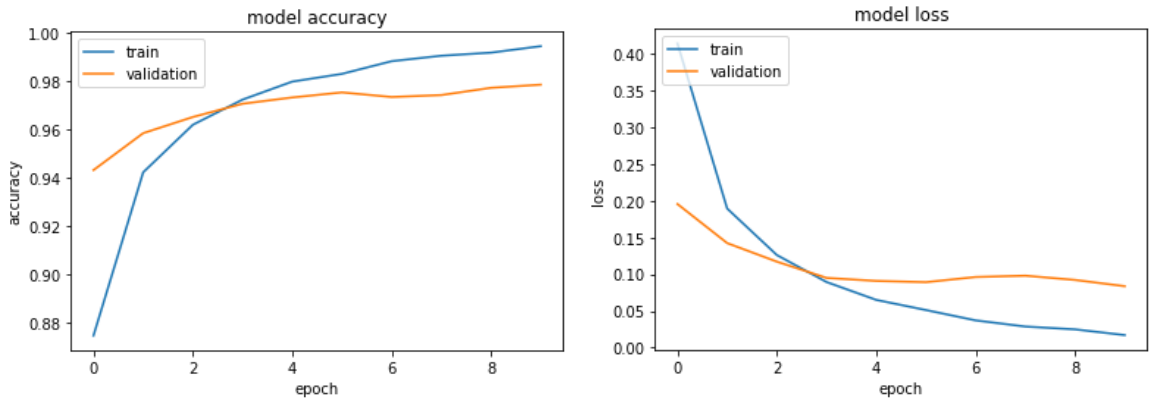
Simple logistic regression model, 784-10 with no hidden layers



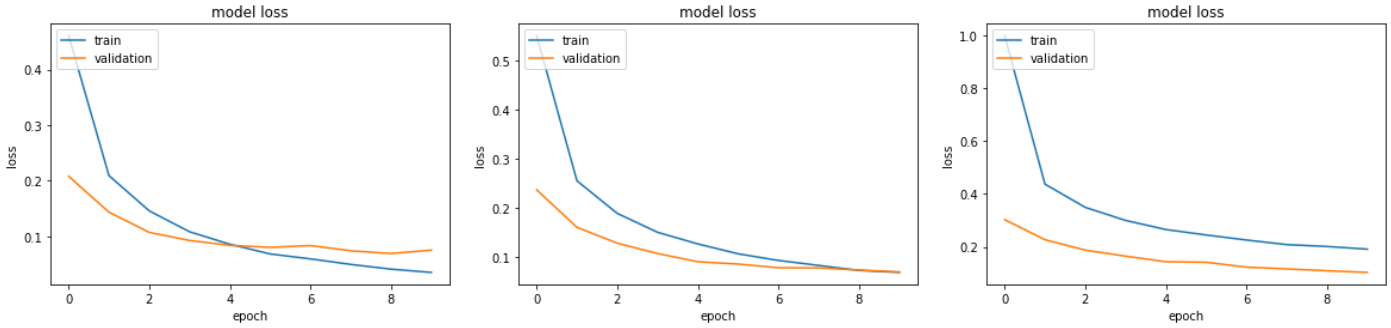
In this model the training loss is seen to be 0.253 and the test loss is seen to ne 0.265, indicating that there is no drastic overfitting, as this is a simple enough model.

Effect of dropout rate

Standard neural net with 2 hidden layers, 800 units



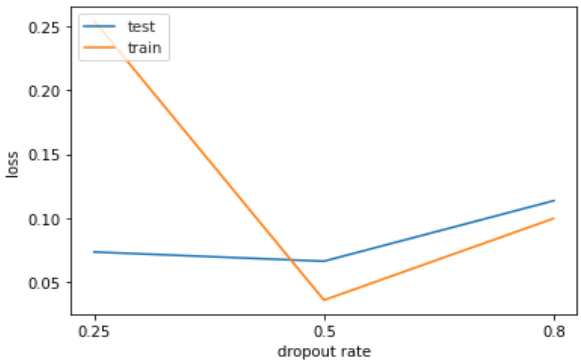
In this model the training loss is 0.0247 and the test loss is 0.0804, indicating heavy overfitting. So we applied dropout at each hidden layers and varied the dropout rate a bit to look at effects of dropout on overfitting.



Training and test losses at dropout rate 0.25,0.5 and 0.8 respectively.

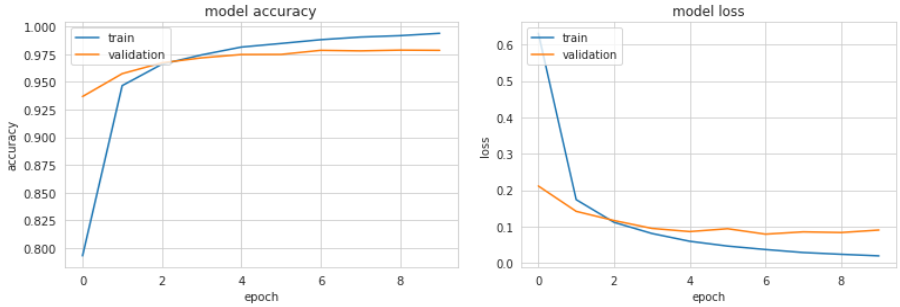
The training and test losses for the following models are,

Dropout rate	0.25	0.5	0.8
Training Loss	0.2540	0.035	0.0997
Test Loss	0.0734	0.066	0.1136



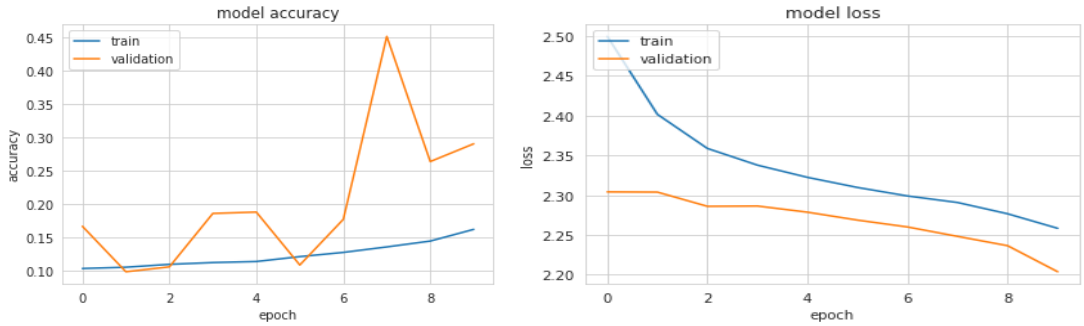
From the results we can see that dropout rate of 0.25 gives better improvement over overfitting and 0.5 dies not give much better results, but then again a dropout rate of 0.8 results in a higher training loss and test loss both, indicating that it underfits the model. So this suggests a dropout rate between 0.25 and 0.5 will give a better result for a large architecture like this for the MNIST dataset.

Standard neural net with 3 hidden layers, 2048 units

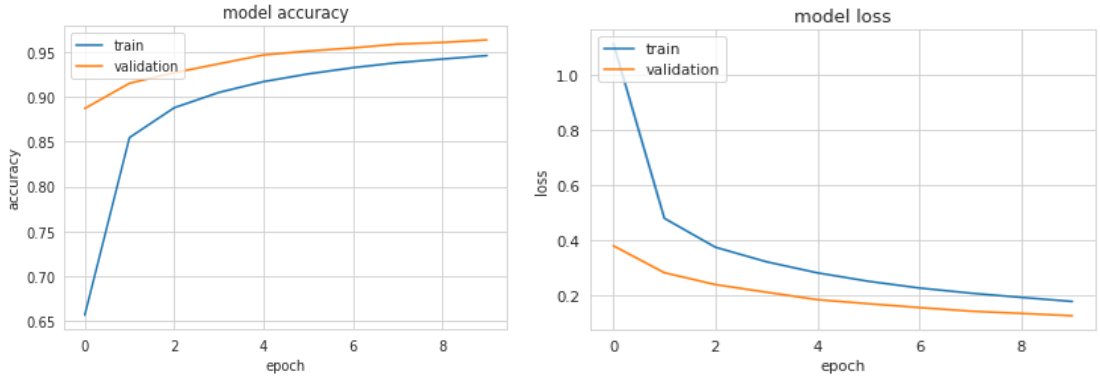


This model also has training loss is 0.0308 and the test loss of 0.0841, which suggests very high overfitting.

After using dropout rate of 0.5 at each hidden layer, and a max-norm constraint of 2, and used the optimizer as sgd instead of adam we see that the model takes time to converge if we use sigmoid activation function.

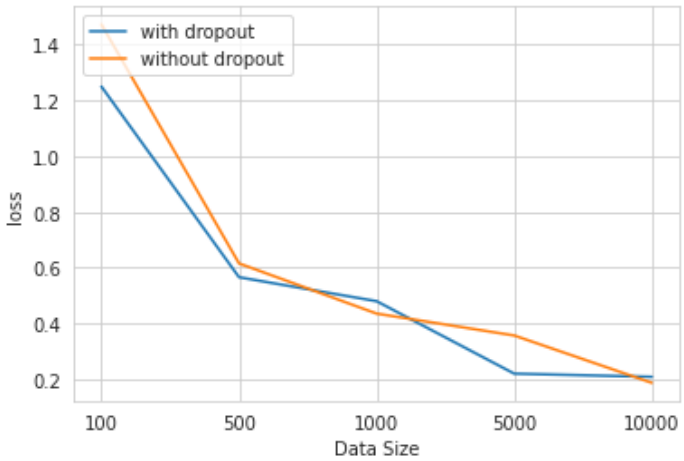


So in the same architecture we used relu activation function.



For the dropout rate of 0.5 there is not much improvement in overfitting, viz the training loss is 0.0308 and the test loss is 0.0841. So relu activation function just helps the algorithm to converge faster.

Effect of data size



The above graph shows, that for smaller datasets the dropout does not show significant effect, i.e the without dropout test loss is also less than with dropout test loss, showing no improvement due to applying dropout layers. But as the dataset gets larger the loss without dropout increases, i.e overfitting is seen without dropout layers, and with dropout the test error starts decreasing giving better result after using dropout layer, due to overfitting. This happens because with smaller datasets the large neural network has enough parameters to overfit the data even after dropout. And the graph shows for a certain architecture, and a given dropout rate there is a certain datasize, with gives similar performance with or without dropout, for us this datasize is somwhere between 500 and 1000, maybe 700.

Conclusion

In conclusion, dropout is a faster and better way to avoid overfitting problem, in neural networks with large architectures while working with large data, where standard backpropagation builds co-adaptaion in layers, random dropout easily breaks these co-adaptation by making any hidden layer probabilistically unreliable. These dropout technique can be used in image classification problems, with neural networks, such as in SVHN, ImageNet also in biological data such as CIFAR-10, CIFAR-100 where co-adaptation is seen.
But also dropout neural networks take more time to train because while training in stochastic gradient descent the parameter updates are very noisy, but this stochasticity reduces overfitting, creating a balance between overfitting and training time, thus we can use higher dropout rate and, more training time, but that can also underfit the data so we can find better results with moderate training time and dropout rate in most data