SUMMARY

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Deep neural networks with a large number of parameters are very powerful, sensitive as well as complicated machine learning systems. However, overfitting is a serious problem in such networks. Due to higher levels of complications in the neural networks, Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting and many methods have been developed for reducing it. These include stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization and soft weight sharing. Dropout is a technique for addressing this problem of overfitting. The key idea behind this is to randomly drop units along with their connections from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different networks. At test time, it is easy to estimate 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. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets. Model combination nearly always improves the performance of machine learning methods. With large neural networks, however, the obvious idea of averaging the outputs of many separately trained nets is prohibitively expensive. Combining several models is most helpful when the individual models are different from each other and in order to make neural net models different, they should either have different architectures or be trained on different data. Training many different architectures is hard because finding optimal hyperparameters for each architecture is a daunting task and training each large network requires a lot of computation. Moreover, large networks normally require large amounts of training data and there may not be enough data available to train different networks on different subsets of the data. Even if one was able to train many different large networks, using them all at test time is infeasible in applications where it is important to respond quickly. Applying dropout to a neural network amount to sampling a “thinned” network from it. A neural net with n units, can be seen as a collection of 2n possible thinned neural networks. These networks all share weights so that the total number of parameters is still O(n^2), or less. For each presentation of each training case, a new thinned network is sampled and trained. So, training a neural network with dropout can be seen as training a collection of 2n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all. At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in practice. The idea is to use a single neural net at test time without dropout. The weights of this network are scaled-down versions of the trained weights. A closely related, but slightly different motivation for dropout comes from thinking about successful conspiracies. Ten conspiracies each involving five people is probably a better way to create havoc than one big conspiracy that requires fifty people to all play their parts correctly. If conditions do not change and there is plenty of time for rehearsal, a big conspiracy can work well, but with non-stationary conditions, the smaller the conspiracy the greater its chance of still working. Complex co-adaptations can be trained to work well on a training set, but on novel test data

they are far more likely to fail than multiple simpler co-adaptations that achieve the same thing. Since dropout can be seen as a stochastic regularization technique, it is natural to consider its deterministic counterpart which is obtained by marginalizing out the noise. In dropout, we minimize the loss function stochastically under a noise distribution. In this instead of a noise distribution, the maximum number of units that can be dropped is fixed. However, this work also does not explore models with hidden units. Dropout neural networks can be trained using stochastic gradient descent in a manner similar to standard neural nets. The only difference is that for each training case in a mini-batch, we sample a thinned network by dropping out units. 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. Any training case which does not use a parameter contributes a gradient of zero for that parameter. Many methods have been used to improve stochastic gradient descent such as momentum, annealed learning rates and L2 weight decay. Those were found to be useful for dropout neural networks as well. One particular form of regularization was found to be especially useful for dropout— constraining the norm of the incoming weight vector at each hidden unit to be upper bounded by a fixed constant c. In other words, if w represents the vector of weights incident on any hidden unit, the neural network was optimized under the constraint $||w||2 \leq c$. This constraint was imposed during optimization by projecting w onto the surface of a ball of radius c, whenever w went out of it. This is also called max-norm regularization since it implies that the maximum value that the norm of any weight can take is c. The constant c is a tunable hyperparameter, which is determined using a validation set. Max-norm regularization has been previously used in the context of collaborative filtering. It typically improves the performance of stochastic gradient descent training of deep neural nets, even when no dropout is used. Although dropout alone gives significant improvements, using dropout along with maxnorm regularization, large decaying learning rates and high momentum provides a significant boost over just using dropout. A possible justification is that constraining weight vectors to lie inside a ball of fixed radius makes it possible to use a huge learning rate without the possibility of weights blowing up. The noise provided by dropout then allows the optimization process to explore different regions of the weight space that would have otherwise been difficult to reach. As the learning rate decays, the optimization takes shorter steps, thereby doing less exploration and eventually settles into a minimum. Dropout can be applied to finetune nets that have been pretrained using these techniques. The pretraining procedure stays the same. 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. We were initially concerned that the stochastic nature of dropout might wipe out the information in the pretrained weights. This did happen when the learning rates used during finetuning were comparable to the best learning rates for randomly initialized nets. However, when the learning rates were chosen to be smaller, the information in the pretrained weights seemed to be retained and we were able to get improvements in terms of the final generalization error compared to not using dropout when finetuning. In order to test the robustness of dropout, classification experiments were done with networks of many different architectures keeping all hyperparameters, including p, fixed. Figure 4 shows the test error rates obtained for these different architectures as training progresses. The same architectures trained with and without dropout have drastically different test errors as seen as by the two separate clusters of trajectories. Dropout gives a huge improvement across all architectures, without using hyperparameters that were tuned specifically for each architecture. In a standard neural network, the derivative received by each parameter tells it how it should change so the final loss function is reduced, given what all other units are doing. Therefore, units may change in a way that they fix up the mistakes of the other units. This may lead to complex co-adaptations. This in turn leads to overfitting because these co-adaptations do not generalize to unseen data. We hypothesize that for each hidden unit, dropout prevents co-adaptation by making the presence of

other hidden units unreliable. Therefore, a hidden unit cannot rely on other specific units to correct its mistakes. It must perform well in a wide variety of different contexts provided by the other hidden units. To observe this effect directly, we look at the first level features learned by neural networks trained on visual tasks with and without dropout. This shows that dropout does break up co-adaptations, which is probably the main reason why it leads to lower generalization errors. Dropout is a technique for improving neural networks by reducing overfitting. Standard backpropagation learning builds up brittle co-adaptations that work for the training data but do not generalize to unseen data. Random dropout breaks up these co-adaptations by making the presence of any particular hidden unit unreliable. This technique was found to improve the performance of neural nets in a wide variety of application domains including object classification, digit recognition, speech recognition, document classification and analysis of computational biology data. This suggests that dropout is a general technique and is not specific to any domain. One of the drawbacks of dropout is that it increases training time. A dropout network typically takes 2-3 times longer to train than a standard neural network of the same architecture. A major cause of this increase is that the parameter updates are very noisy. Each training case effectively tries to train a different random architecture. Therefore, the gradients that are being computed are not gradients of the final architecture that will be used at test time. Therefore, it is not surprising that training takes a long time. However, it is likely that this stochasticity prevents overfitting. This creates a trade-off between overfitting and training time. With more training time, one can use high dropout and suffer less overfitting. However, one way to obtain some of the benefits of dropout without stochasticity is to marginalize the noise to obtain a regularizer that does the same thing as the dropout procedure, in expectation. We showed that for linear regression this regularizer is a modified form of L2 regularization. For more complicated models, it is not obvious how to obtain an equivalent regularizer. Speeding up dropout is an interesting direction for future work. Neural networks are infamous for requiring extensive hyperparameter tuning. Dropout networks are no exception. In this section, we describe heuristics that might be useful for applying dropout. It is to be expected that dropping units will reduce the capacity of a neural network. If n is the number of hidden units in any layer and p is the probability of retaining a unit, then instead of n hidden units, only pn units will be present after dropout, in expectation. Moreover, this set of pn units will be different each time and the units are not allowed to build co-adaptations freely. Therefore, if an n-sized layer is optimal for a standard neural net on any given task, a good dropout net should have at least n/p units. Dropout introduces a significant amount of noise in the gradients compared to standard stochastic gradient descent. Therefore, a lot of gradients tend to cancel each other. In order to make up for this, a dropout net should typically use 10-100 times the learning rate that was optimal for a standard neural net. Another way to reduce the effect the noise is to use a high momentum. While momentum values of 0.9 are common for standard nets, with dropout we found that values around 0.95 to 0.99 work quite a lot better. Using high learning rate and/or momentum significantly speed up learning. Though large momentum and learning rate speed up learning, they sometimes cause the network weights to grow very large. To prevent this, we can use max-norm regularization. This constrains the norm of the vector of incoming weights at each hidden unit to be bound by constant c. Typical values of c range from 3 to 4. Dropout introduces an extra hyperparameter the probability of retaining a unit p. This hyperparameter controls the intensity of dropout. p = 1, implies no dropout and low values of p mean more dropout. Typical values of p for hidden units are in the range 0.5 to 0.8. For input layers, the choice depends on the kind of input. For real-valued inputs like image patches or speech frames, a typical value is 0.8. For hidden layers, the choice of p is coupled with the choice of number of hidden units n. Smaller p requires big n which slows down the training and leads to underfitting. Large p may not produce enough dropout to prevent overfitting.