Dropout Regularization

- Dropout momentarily (in a batch of input data) switches off some neurons in a layer so that they do not contribute any information or learn any information during those updates, and the onus falls on other active neurons to learn harder and reduce the error.
- If I have to explain drop-out to a 6-year-old, this is how: Imagine a scenario, in a classroom, a teacher asks some questions but always same two kids are answering, immediately. Now, the teacher asks them to stay quiet for some time and let other pupils participate. This way other students get to learn better. Maybe they answer wrong, but the teacher can correct them(weight updates). This way the whole class(layer) learn about a topic better.

How the dropout regularization technique works.

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to complex co-adaptations.

You can imagine that if neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

---From Research Paper---

- When a large feedforward neural network is trained on a small training set, it typically performs poorly on held-out test data. This “overfitting” is

greatly reduced by randomly omitting half of the feature detectors on each training case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate. Random "dropout" gives big improvements on many benchmark tasks and sets new records for speech and object recognition.

- A feedforward, artificial neural network uses layers of non-linear "hidden" units between its inputs and its outputs. By adapting the weights on the incoming connections of these hidden units it learns feature detectors that enable it to predict the correct output when given an input vector (1). If the relationship between the input and the correct output is complicated and the network has enough hidden units to model it accurately, there will typically be many different settings of the weights that can model the training set almost perfectly, especially if there is only a limited amount of labeled training data. Each of these weight vectors will make different predictions on held-out test data and almost all of them will do worse on the test data than on the training data because the feature detectors have been tuned to work well together on the training data but not on the test data.
- Overfitting can be reduced by using "dropout" to prevent complex co-adaptations on the training data. On each presentation of each training case, each hidden unit is randomly omitted from the network with a probability of 0.5, so a hidden unit cannot rely on other hidden units being present. Another way to view the dropout procedure is as a very efficient way of performing model averaging with neural networks. A good way to reduce the error on the test set is to average the predictions produced by a very large number of different networks. The standard way to do this is to train many separate networks and then to apply each of these networks to the test data, but this is computationally expensive during both training and testing. Random dropout makes it possible to train a huge number of different networks in a reasonable time. There is almost certainly a different network for each presentation of each training case but all of these networks share the same weights for the hidden units that are present.
- We use the standard, stochastic gradient descent procedure for training the dropout neural networks on mini-batches of training cases, but we modify the penalty term that is normally used to prevent the weights from growing too large. Instead of penalizing the squared length (L2 norm) of the whole weight vector, we set an upper bound on the L2 norm of the incoming

weight vector for each individual hidden unit. If a weight-update violates this constraint, we renormalize the weights of the hidden unit by division. Using a constraint rather than a penalty prevents weights from growing very large no matter how large the proposed weight-update is. This makes it possible to start with a very large learning rate which decays during learning, thus allowing a far more thorough search of the weight-space than methods that start with small weights and use a small learning rate.

- At test time, we use the "mean network" that contains all of the hidden units but with their outgoing weights halved to compensate for the fact that twice as many of them are active. In practice, this gives very similar performance to averaging over a large number of dropout networks. In networks with a single hidden layer of N units and a "softmax" output layer for computing the probabilities of the class labels, using the mean network is exactly equivalent to taking the geometric mean of the probability distributions over labels predicted by all 2 N possible networks. Assuming the dropout networks do not all make identical predictions, the prediction of the mean network is guaranteed to assign a higher log probability to the correct answer than the mean of the log probabilities assigned by the individual dropout networks (2). Similarly, for regression with linear output units, the squared error of the mean network is always better than the average of the squared errors of the dropout networks.

---From Machine Learning Mastery---

A Gentle Introduction to Dropout for Regularizing Deep Neural Networks

Deep learning neural networks are likely to quickly overfit a training dataset with few examples.

Ensembles of neural networks with different model configurations are known to reduce overfitting, but require the additional computational expense of training and maintaining multiple models.

A single model can be used to simulate having a large number of different network architectures by randomly dropping out nodes during training. This is called dropout and offers a very computationally cheap

and remarkably effective regularization method to reduce overfitting and improve generalization error in deep neural networks of all kinds.
In this post, you will discover the use of dropout regularization for reducing overfitting and improving the generalization of deep neural networks.

After reading this post, you will know:

- Large weights in a neural network are a sign of a more complex network that has overfit the training data.
- Probabilistically dropping out nodes in the network is a simple and effective regularization method.
- A large network with more training and the use of a weight constraint are suggested when using dropout.

Overview

This tutorial is divided into five parts; they are:

1. Problem With Overfitting
2. Randomly Drop Nodes
3. How to Dropout

Problem With Overfitting

Large neural nets trained on relatively small datasets can overfit the training data.

This has the effect of the model learning the statistical noise in the training data, which results in poor performance when the model is evaluated on new data, e.g. a test dataset. Generalization error increases due to overfitting.

One approach to reduce overfitting is to fit all possible different neural networks on the same dataset and to average the predictions from each model. This is not feasible in practice, and can be approximated using a small collection of different models, called an ensemble.

With unlimited computation, the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by its posterior probability given the training data.

— Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014.

A problem even with the ensemble approximation is that it requires multiple models to be fit and stored, which can be a challenge if the models are large, requiring days or weeks to train and tune.

Randomly Drop Nodes

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.

During training, some number of layer outputs are randomly ignored or "*dropped out*." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "*view*" of the configured layer.

By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections

— Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014.

Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.

This conceptualization suggests that perhaps dropout breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust.

… 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. […]

— Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014.

Dropout simulates a sparse activation from a given layer, which interestingly, in turn, encourages the network to actually learn a sparse representation as a side-effect. As such, it may be used as an

alternative to activity regularization for encouraging sparse representations in autoencoder models.

We found that as a side-effect of doing dropout, the activations of the hidden units become sparse, even when no sparsity inducing regularizers are present.

— Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014.

Because the outputs of a layer under dropout are randomly subsampled, it has the effect of reducing the capacity or thinning the network during training. As such, a wider network, e.g. more nodes, may be required when using dropout.

How to Dropout

Dropout is implemented per-layer in a neural network.

It can be used with most types of layers, such as dense fully connected layers, convolutional layers, and recurrent layers such as the long short-term memory network layer.

Dropout may be implemented on any or all hidden layers in the network as well as the visible or input layer. It is not used on the output layer.

The term "dropout" refers to dropping out units (hidden and visible) in a neural network.

— Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014.

A new hyperparameter is introduced that specifies the probability at which outputs of the layer are dropped out, or inversely, the probability at which outputs of the layer are retained. The interpretation is an implementation detail that can differ from paper to code library.

A common value is a probability of 0.5 for retaining the output of each node in a hidden layer and a value close to 1.0, such as 0.8, for retaining inputs from the visible layer.

In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5.

— Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014.

Dropout is not used after training when making a prediction with the fit network.

The weights of the network will be larger than normal because of dropout. Therefore, before finalizing the network, the weights are first scaled by the chosen dropout rate. The network can then be used as per normal to make predictions.

If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time

— Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014.

The rescaling of the weights can be performed at training time instead, after each weight update at the end of the mini-batch. This is sometimes called "*inverse dropout*" and does not require any modification of weights during training. Both the Keras and PyTorch deep learning libraries implement dropout in this way.

At test time, we scale down the output by the dropout rate. […] Note that this process can be implemented by doing both operations at training time and leaving the output unchanged at test time, which is often the way it's implemented in practice

— Page 109, Deep Learning With Python, 2017.

Dropout works well in practice, perhaps replacing the need for weight regularization (e.g. weight decay) and activity regularization (e.g. representation sparsity).

… dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization. Dropout may also be combined with other forms of regularization to yield a further improvement.