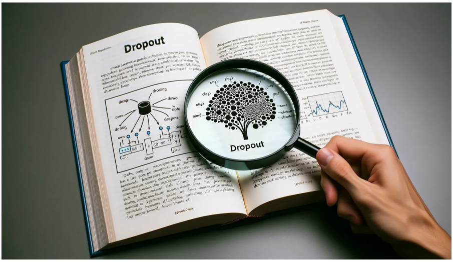
Get unlimited access to the best of Medium for less than $1/week. Become a member

A Deep Dive into the Dropout in Deep Learning

An Exegesis on the Principle of Dropout in Deep Learning

jida yu · Follow
9 min read · Oct 10, 2023



· · ·

1. An Introduction to Dropout

1.1 The Genesis of Dropout

In machine learning, when a model is inundated with parameters yet starved of ample training samples, it becomes susceptible to the perils of overfitting. This overfitting quandary is a frequent encounter during the training of neural networks. Its manifestations are evident: while the model exhibits a diminutive loss function and heightened accuracy on training data, it portrays an inflated loss function and diminished accuracy on test data.

Overfitting is the Achilles' heel of myriad machine learning endeavors. An overfitted model is rendered virtually impotent. To counteract this, the strategy often employed is model ensemble, which entails the amalgamation of multiple trained models. However, this approach introduces a significant temporal challenge. Not only is the training of multiple models time-consuming, but their subsequent testing is equally taxing.

In summation, the training of deep neural networks invariably presents two predominant shortcomings:

(1) A proclivity towards overfitting

(2) Time-intensive

Dropout serves as an efficacious antidote to overfitting, offering a semblance of regularization.

1.2 What is Dropout?

In 2012, Hinton elucidated the concept of Dropout in his seminal paper, "Improving neural networks by preventing co-adaptation of feature detectors." When a sophisticated feed-forward neural network is trained on a limited dataset, it becomes vulnerable to overfitting. To combat this, the performance of neural networks can be enhanced by inhibiting the collaborative actions of feature detectors.

In the same year, Alex and Hinton incorporated the Dropout algorithm in their paper, "ImageNet Classification with Deep Convolutional Neural Networks," as a bulwark against overfitting. Moreover, this paper's introduction of the AlexNet network model ignited a fervor for neural network applications, clinching the championship in the 2012 image recognition competition and cementing CNN as the quintessential algorithmic model for image classification.

Subsequent scholarly contributions on Dropout include "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," "Improving Neural Networks with Dropout," and "Dropout as data augmentation."

From these academic treatises, one can discern the paramount significance of Dropout in deep learning. So, what precisely is Dropout?

Dropout can be envisaged as a stratagem during the training of deep neural networks. In each training batch, by deliberately neglecting half of the feature detectors (setting half of the hidden layer node values to zero), one can markedly mitigate the occurrence of overfitting. This method curtails the interplay between feature detectors (hidden layer nodes), where interplay denotes the reliance of certain detectors on others to function optimally.

In simpler terms, Dropout entails: **during forward propagation, deactivating the activation value of a neuron with a certain probability p, thereby bolstering the model's generalization capabilities. This ensures the model doesn't become overly reliant on specific localized features**, as illustrated in Figure 1.

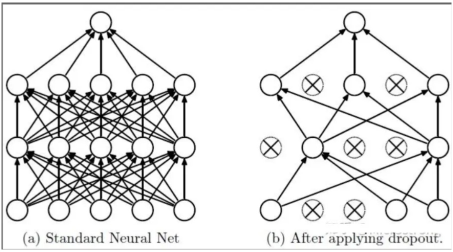


Figure 1: Neural Net applying dropout

2. Dropout Workflow and Implementation

2.1 Detailed Workflow of Dropout

Imagine training a neural network as depicted in Figure 2.

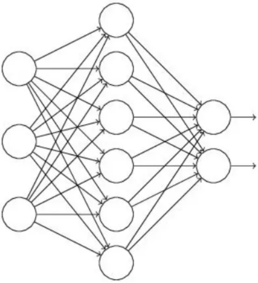


Figure 2: A Standard Neural Network

With input as x and output as y, the process entails: initially propagating x forward through the network, then backpropagating the loss to determine the parameter updates, facilitating the network's learning. Upon integrating Dropout, the procedure metamorphoses as follows:

(1) Begin by randomly (and temporarily) excising half of the hidden neurons within the network, ensuring the input and output neurons remain untouched (in Figure 3, the neurons represented by dashed lines are those temporarily removed).

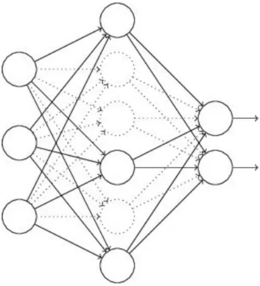


Figure 3: Neurons Temporarily Removed

(2) Subsequently, propagate the input x forward through this modified network, and then backpropagate the resultant loss through the altered network. Upon completing this process for a mini-batch of training samples, update the corresponding parameters (w, b) on the neurons that weren't removed, utilizing the stochastic gradient descent method.

(3) Persistently iterate through the following steps:

- Reinstate the removed neurons (at this juncture, the excised neurons remain unchanged, while the intact neurons have been updated).
- Arbitrarily select and temporarily remove a subset, equivalent to half the size, from the hidden neurons (backup the parameters of the neurons being removed).
- For a mini-batch of training samples, first conduct forward propagation, then backpropagate the loss, and subsequently update the parameters (w, b) based on stochastic gradient descent (only the parameters of the unremoved portion are updated, while the parameters of the removed neurons remain as they were prior to removal).

Continue this iterative process ad infinitum.

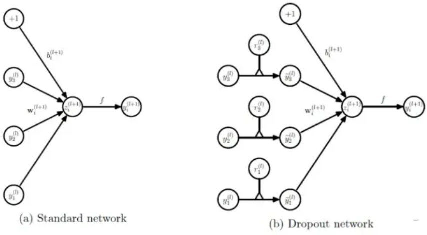
2.2 Utilizing Dropout in Neural Networks

The intricate workflow of Dropout has been delineated above. But how do we ensure that certain neurons cease functioning (i.e., get dropped) with a specific probability? How is this implemented at the code level?

Below, we delve into the mathematical derivations and coding logic behind Dropout.

(1) During the Model Training Phase

Inevitably, a probabilistic process must be added to each unit during network training.



(a) Standard network

(b) Dropout network

Figure 4: Comparison between a Standard Network and a Network with Dropout

The corresponding formula alterations are as follows:

Formula for a network without Dropout:

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)} \\
y_i^{(l+1)} &= f\left(z_i^{(l+1)}\right),
\end{aligned}
$$

Formula for a network with Dropout:

$$r_j^{(l)} \sim \text{Bernoulli}(p),$$

$$\tilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} * \mathbf{y}^{(l)}$$
$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$

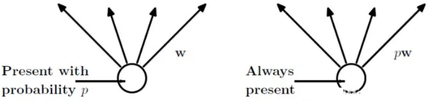
In the above formulas, the Bernoulli function is employed to generate the probability vector r, which essentially produces a vector of 0s and 1s.

At the code level, to make a neuron cease functioning with probability p, its activation function value is set to 0 with the same probability. For instance, if a network layer has 1000 neurons with activation outputs $y_1, y_2, y_3, \dots y_{1000}$, and we choose a dropout rate of 0.4, then post-dropout, approximately 400 of these 1000 neuron values will be set to 0.

Note: After masking certain neurons and setting their activation values to 0, it's imperative to rescale the vector $y_1 \dots y_{1000}$ by multiplying it with 1/(1-p). If you don't rescale $y_1 \dots y_{1000}$ post-dropout during training, then during testing, you'll need to scale the weights accordingly.

(2) During the Model Testing Phase

When predicting with the model, the weight parameter of each neural unit must be multiplied by the probability p.



(a) At training time (b) At test time

Figure 5: Dropout Operation during Model Prediction

Dropout formula during the testing phase:

$$w_{test}^{(l)} = pW^{(l)}$$

3. Why is Dropout Effective Against Overfitting?

(1) **Averaging Effect:** Reverting to a standard model without dropout, suppose we train five different neural networks using the same training data. Typically, we'd obtain five distinct results. At this juncture, we could employ an "average of the five results" or a "majority wins voting strategy" to determine the final outcome. For instance, if three networks deduce the result as the digit 9, it's highly probable that the actual result is indeed 9, with the other two networks providing erroneous results. This "consolidated averaging" strategy can effectively counteract overfitting. Different networks might manifest different overfitting patterns; averaging can potentially neutralize some "opposing" fits. Dropping different hidden neurons during dropout is akin to training varied networks. The entire dropout process is analogous to averaging across multiple neural networks. As different networks exhibit different overfitting tendencies, some "opposite" fits can cancel each other out, reducing overall overfitting.

(2) **Reducing Co-adaptation of Neurons:** The dropout procedure ensures that

two neurons might not always co-exist in a dropout network. Consequently, the weight updates no longer rely on the joint action of hidden nodes with fixed relationships, preventing situations where some features are only effective in the presence of other specific features. This forces the network to learn more robust features that also exist in other random subsets of neurons. In other words, if our neural network is making a prediction, it shouldn't be overly sensitive to specific cues. Even in the absence of certain cues, it should discern common features from other cues. From this perspective, dropout resembles L1 and L2 regularization, reducing weights and enhancing the network's robustness against the loss of specific neuron connections.

(3) **Dropout Resembles the Role of Gender in Biological Evolution**: For survival, species tend to adapt to their environment. Sudden environmental changes can make it challenging for species to respond promptly. The emergence of gender can produce variants adapted to new environments, effectively preventing overfitting or avoiding potential extinction when environments change.

4. Analysis of Dropout in Keras Source Code

Next, let's delve into the implementation of Dropout in Keras.

The open-source project for Keras can be found on GitHub at:

https://github.com/fchollet/keras/tree/master/keras

The specific file containing the Dropout function implementation is located at:

https://github.com/fchollet/keras/blob/master/keras/backend/theano_backend.py

The Dropout function in Keras is depicted as:

Figure 6: Dropout Implementation in Keras

We can modify the Dropout function in Keras, allowing it to operate independently.

```
# Implementation of the dropout function
def dropout(x, level):
    # 'level' is a probability value and must be between 0 and 1
    if level < 0. or level >= 1:
        raise ValueError('Dropout level must be in interval [0, 1[.')
    retain_prob = 1. - level

    # Using the binomial function to generate a vector of the same dimension as
    # The binomial function is like flipping a coin. We can consider each neuron
    # The probability of the coin landing heads up is p, and n represents the nu
    # Since we only need to flip once for each neuron, n=1. The 'size' parameter
    random_tensor = np.random.binomial(n=1, p=retain_prob, size=x.shape)
    # This will generate a vector with a distribution of 0s and 1s.
    # 0 indicates that the neuron is masked (not working), i.e., it's dropped ou
    print(random_tensor)

    x *= random_tensor
    print(x)
    x /= retain_prob

    return x

# Test the dropout function. You can run the above function to see the result of
x=np.asarray([1,2,3,4,5,6,7,8,9,10],dtype=np.float32)
dropout(x,0.4)
```

Within the function, `x` represents the activation values of the current network layer. `Level` denotes the probability with which each neuron will be dropped during dropout.

Note: In Keras's implementation of Dropout, after masking certain neurons and setting their activation values to zero, the activation value vector x_1… x_{1000} is amplified, i.e., multiplied by 1/(1-p).

Reflection: We've discussed two methods for Dropout scaling. So, why is scaling necessary in Dropout?

During training, certain neurons are randomly dropped. However, during prediction, such random dropping isn't feasible. If neurons are dropped, it can lead to instability in the results. For a given test data, the output might sometimes be 'a' and sometimes 'b'. Such inconsistency is unacceptable in practical systems, as users might perceive the model as inaccurate. A compensatory approach is to multiply the weight of each neuron by a factor of p. This ensures that, on average, the test data and training data outputs are approximately equivalent. For instance, if a neuron's output is x, during training, it has a probability p of participating and a (1-p) probability of being dropped. Thus, its expected output is px + (1-p)0 = px. Therefore, multiplying this neuron's weight by p during testing yields the same expectation.

In Conclusion:

Currently, Dropout is extensively employed in fully connected networks, typically set at values like 0.5 or 0.3. In convolutional network hidden layers, due to the inherent sparsity of convolution and the widespread use of sparse

ReLu functions, the Dropout strategy is less frequently applied. Overall, Dropout is a hyperparameter, and its optimal value requires experimentation based on the specific network and application domain.

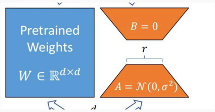
Dropout Regularization Machine Learning Deep Learning Neural Networks

Written by jida yu

1 Follower

Follow

More from jida yu


Pretrained Weights
$W \in \mathbb{R}^{d\times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d

jida yu

LoRA from a Gradient Perspective: An Overview, Analysis,...

Is it feasible to analyze and implement LoRA from the standpoint of an optimizer?

Sep 17, 2023 13

jida yu

Deep Learning vs. Reinforcement Learning: A Guide Even My...

To what extent do the distinctions between deep learning and reinforcement learning...

Oct 10, 2023 6

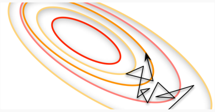
PRIVACY
PRIVACY
PRIVACY

jida yu

Potential Solution to Data Privacy Issues of Large Models with...

Deciphering the Synergy of Federated LLMs: Advantages, Challenges, and the Road Ahea...

Sep 11, 2023 7



jida yu

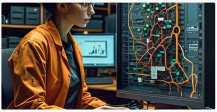
Batch_Size is a pivotal parameter in machine learning,...

Firstly, why is there a need for the Batch_Size parameter?

Sep 10, 2023 2

See all from jida yu

Recommended from Medium



Mounica Kommajosyula

Getting Started with Hyperparameters: A Beginner's...

In machine learning, there is a lot of talk about building powerful models, feeding them dat...

✦ Nov 5 👏 1

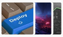
Amit Yadav in Data Scientist's Diary

PyTorch Reimplementation of Diffusion Models: A Complete...

You're likely already familiar with the basics: these models use an iterative process to ad...

✦ Nov 7 👏 23

Lists



Predictive Modeling w/ Python

20 stories · 1676 saves

Practical Guides to Machine Learning

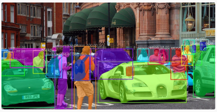
10 stories · 2031 saves

Natural Language Processing

1815 stories · 1428 saves

data science and AI

40 stories · 285 saves

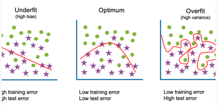


Thoughts By Swarnali

Performance Comparison of Faster R-CNN and YOLO for Real-time...

Introduction

✦ Jul 10 👏 56

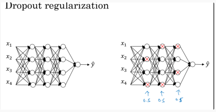


Maha K

Understanding Underfitting and Overfitting in Data Science: A...

How to Identify, Avoid, and Address Model Performance Issues with Practical Code...

✦ Aug 30 👏 2



Coursesteach

Dropout Regularization Explained: A Beginner's Guide to Reducing...

📚 Chapter 6: Practical Aspects of Deep Learning

✦ Aug 2 👏 2

Greg Postalian-Yrausquin

Neural networks: encoder-decoder example (autoencoder)

The autoencoder is an architecture of neural networks that consists of the implementatio...

Jun 11 👏 1

See more recommendations