Asian Journal of Research in Computer Science

Volume 17, Issue 10, Page 85-94, 2024; Article no.AJRCOS.120002
ISSN: 2581-8260

Proposed Methods for Preventing Overfitting in Machine Learning and Deep Learning

Vladimir Diukarev [a++*] and Yaroslav Starukhin [b#]

[a] Anti-Fraud Department, Sberbank, Moscow, Russian Federation.
[b] McKinsey & Company, Boston, USA.

Authors' contributions

This work was carried out in collaboration between both authors. Both authors read and approved the final manuscript.

Article Information

DOI: https://doi.org/10.9734/ajrcos/2024/v17i10511

Open Peer Review History:
This journal follows the Advanced Open Peer Review policy. Identity of the Reviewers, Editor(s) and additional Reviewers, peer review comments, different versions of the manuscript, comments of the editors, etc are available here: https://www.sdiarticle5.com/review-history/120002

Original Research Article

Received: 20/06/2024
Accepted: 24/08/2024
Published: 09/10/2024

ABSTRACT

The article discusses various approaches and methods to combat overfitting, which is a key problem in the field of artificial intelligence. Overfitting occurs when the model over-adapts to the training dataset, losing the ability to generalize to new data. The main causes and signs of overfitting are also discussed, including excessive complexity of models and limited data. The focus is on methods such as regularization, dropout, and the use of ensemble methods that can significantly reduce the risk of overfitting. These approaches are evaluated using examples from various fields of neural network applications, providing the reader with a comprehensive understanding of the problem and its solution methods.

Keywords: Deep learning; overfitting, dropout; regularization; AI; artificial intelligence.

[++] Head of Data Analytics;
[#] Senior Data Scientist;
**Corresponding author: E-mail: dyukarev-vv@sberbank.ru;*

Cite as: *Diukarev , Vladimir, and Yaroslav Starukhin. 2024. "Proposed Methods for Preventing Overfitting in Machine Learning and Deep Learning". Asian Journal of Research in Computer Science 17 (10):85-94. https://doi.org/10.9734/ajrcos/2024/v17i10511.*

1. INTRODUCTION

Deep learning is a machine learning approach that utilizes multilayer neural networks to automatically extract hierarchical representations from raw data. This method allows computer systems to improve their performance on the required task through experience, without the need for explicit programming of each step.

Despite significant progress in deep learning, it faces several challenges, one of which is overfitting. Overfitting is a common and critical problem in deep learning where models show excellent performance on training data but fail to generalize well to unseen data. This phenomenon occurs when the model learns not only the underlying patterns in the training data but also the noise and outliers, leading to a significant decrease in predictive performance on new data. Addressing overfitting is crucial for developing robust, reliable, and effective deep learning models.

The tendency towards overfitting is exacerbated by the complexity of deep learning models, which often consist of millions of parameters. These parameters provide the model with significant potential to fit the training data, but without proper regularization, this potential can lead to memorizing training examples rather than learning generalized patterns. To mitigate overfitting, various methods have been developed, enhancing the model's ability to generalize training data to real-world scenarios [1-5].

One of the fundamental methods to combat overfitting is regularization, which imposes penalties on large weights in the model, effectively limiting the model's complexity. Another widely-used technique is Dropout, which involves randomly turning off a fraction of neurons during training, preventing neurons from co-adapting and promoting a more robust learning process.

Another practical approach is Early Stopping, which involves monitoring the model's performance on a validation set during training and stopping the training process once performance starts to deteriorate. This prevents the model from overfitting to the training data by limiting the amount of training.

Increasing the volume of data is also a vital strategy, particularly in fields where obtaining large amounts of labeled data is challenging. Artificially expanding the training dataset with transformations such as rotation, translation, and scaling helps the model generalize better by exposing it to a broader range of examples [6].

In this article, we will explore the main methods of preventing overfitting in deep learning. We will provide a mathematical and conceptual overview of the Dropout method, followed by a practical implementation example using binomial distribution to model sales success across multiple stores. This implementation illustrates the application of theoretical concepts in a real scenario, demonstrating the effectiveness of these methods in enhancing the generalization capabilities of deep learning models.

2. MATERIALS AND METHODS

2.1 Causes of Overfitting

Overfitting in deep learning models arises from several interrelated factors that lead the model to perform exceptionally well on training data but poorly on unseen data. Understanding these causes helps in developing strategies aimed at reducing overfitting and improving model generalization. The main causes of overfitting can be divided into three primary areas: model complexity, insufficient data, and data noise.

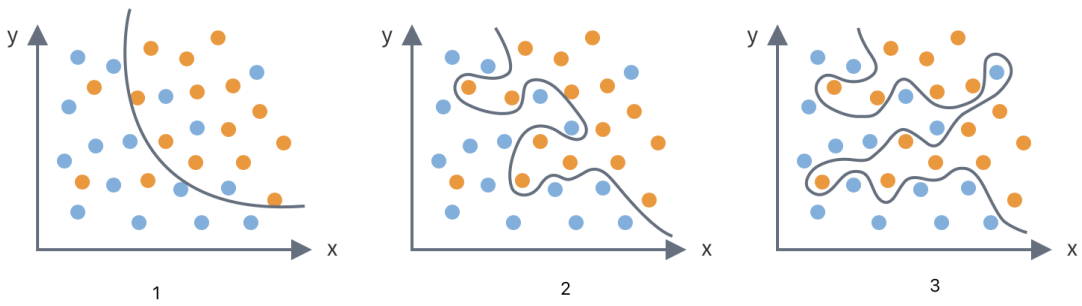

Fig. 1. Training examples
1 - Underfitting; 2 - Optimum; 3 - Overfitting

The goal of training an artificial neural network (ANN) is to develop a model capable of adequately generalizing the learned patterns to data from any domain, which is critical for its ability to predict unseen data. The first step in combating overfitting involves simplifying the model by reducing the number of layers or neurons, with the key being the correct calculation of the input and output sizes of various layers of the neural network [7].

Overfitting in machine learning, especially in the context of neural networks, is a significant challenge that cannot be completely eliminated. This phenomenon is often caused by several factors:

1. Model complexity relative to the amount of data: For instance, a model attempting to explore numerous solutions for a single task may start detecting non-existent patterns that are actually just noise in the data, if the complexity of the model does not match the volume and diversity of the provided information.
2. Limited data availability: When the amount of available data is insufficient for training, the model may begin making conclusions based on atypical features or exceptions, thereby distorting its ability to generalize.
3. Noise in the data: Similar to trying to understand a complex musical composition amidst background noise, noise in the data can cause the model to adapt to random rather than meaningful characteristics in the data.
4. Neglecting validation data: Training a model without adequate testing and validation on a validation dataset can lead to the creation of an "illusory" model that performs well only on the training data.

Recognizing and understanding these factors is crucial for developing robust and reliable machine learning systems. It is also important to identify signs of overfitting:

- Performance Imbalance: High accuracy on the training dataset and poor results on test data often indicate overfitting.
- Complex Decision Boundaries: If the visual analysis of decision boundaries shows excessive complexity or "capriciousness," it may indicate overfitting.
- Performance Stagnation or Deterioration on Validation Data: Improvement in performance during initial training stages, followed by a slowdown or even deterioration while training data performance continues to grow, can be a sign of overfitting.
- Performance Fluctuations with Training Data Changes: Significant performance fluctuations with minor changes in data can indicate excessive model sensitivity and a tendency to overfit [8,9].

One of the most obvious strategies to combat overfitting is to increase the volume and diversity of training data. For example, if a dataset includes not just six animals, but a thousand different individuals, it is likely to derive a more accurate rule. Instead of a simplistic heuristic like "who swims is a fish," which is not applicable to some birds like ducks or swans, a more accurate rule could be formulated, such as "those with gills are fish."

However, it is necessary to consider not only the quantity but also the heterogeneity of the data. If the training set consisted of a thousand birds, none of which could swim, the rule "who swims is a fish" might still be derived by default. Therefore, both quantitative and qualitative diversity of data is essential for effective training.

Imagine the scenario of developing an automated traffic control system. This system must decide whether to allow vehicles to continue moving or stop, based on various factors such as speed, traffic density, and road conditions. Suppose this decision-making process can be mathematically modeled, where the decision is based on a weighted sum of these factors. If this sum exceeds a certain threshold, the system allows the vehicle to continue moving; otherwise, it stops the vehicle.

Consider the scenario where a vehicle approaches an intersection. Based on collected data and historical traffic patterns, the decision-making formula might be as follows: a base score of 20 points, add 50 points for high traffic density, 30 points for unfavorable weather conditions, subtract 10 points for good road visibility, and 20 points for vehicle speed below the set limit, resulting in a total score of 70 points. Since this score exceeds the threshold of 50 points, the system decides to stop the vehicle.

Historically, if the system never allowed vehicles to move under high traffic conditions, it could lead to the machine learning model assigning disproportionate weight to traffic density,

rendering all other factors insignificant. This scenario highlights the limitations and potential pitfalls of automated systems making decisions solely based on quantitative evaluations and historical data.

Fortunately, to prevent excessive dominance of individual features during training, regularization techniques exist. This method restricts the influence of each factor, preventing disproportionately large weights in the model.

2.2 Methods of Regularization and Overfitting Control

To understand how a machine learning model operates, it's crucial to recognize that its primary task during training is to minimize the loss function, which quantifies the errors made on the training data set. Typically, the more errors, the higher the value of the loss function.

The addition of regularization modifies the loss function such that it not only considers errors but also the magnitude of feature weights used in the model. As these weights increase, so does the value of the loss function, which helps prevent their excessive growth. This helps prevent the model from fitting noise in the training data, thereby improving its ability to generalize.

The loss function f can be expressed as a dependency on the target variable Y and the generalizing rule $h(x)$, which represents the model expressing Y through a set of features x considering weights w:

$$f(\text{model errors}) = f(Y, h(x;w))$$

Regularization adds a penalty to the loss function for the magnitude of weights:

$$L = f(Y, h(x;w)) + \lambda \cdot R(w)$$

Here, λ is the regularization coefficient, and $R(w)$ is the regularization term penalizing weight magnitude to prevent their excessive increase.

L1 and L2 regularization are two common methods that add penalties based on the size of the model parameters. L1 regularization (Lasso) adds a penalty equal to the absolute value of the coefficient size, promoting sparsity by reducing some weights to zero. L2 regularization (Ridge) adds a penalty equal to the square of the coefficient size, distributing the error across all parameters and preventing large weights.

Mathematically, the regularized loss functions are defined as follows:

L1 Regularization: $L = f(Y, h(x;w)) + \lambda \cdot \sum_i |w_i|$
L2 Regularization: $L = f(Y, h(x;w)) + \lambda \cdot \sum_i w_i^2$

In recent years, the use of L1 and L2 regularization has remained highly relevant and has significantly evolved in the context of deep learning [10,11]. Modern research shows that the combined application of L1 and L2 regularization, known as Elastic Net, successfully minimizes the risk of overfitting by reducing model complexity while still highlighting important features [12]. For example, when using L2 regularization in logistic regression, a novel approach to hyperparameter tuning has been developed, allowing the optimization of regularization coefficients for each specific task [13]. Moreover, the application of regularization in deep neural networks has become more feasible due to the improved stability and computational efficiency of L2 regularization, making it the preferred choice in most cases [14].

Furthermore, regularization has proven particularly effective in enhancing the performance of neural networks when dealing with small datasets. In such scenarios, L2 regularization significantly reduces the risk of overfitting while maintaining high prediction accuracy, even with limited training data [15].

Regularization effectively guides the model training process to not solely focus on a few features but rather consider a diverse range of factors influencing the outcome. This process can be likened to thoughtful decision-making by a person, analyzing all available aspects of a situation to avoid missing any critical details [16].

Another regularization method specific to neural networks is ***Dropout***. During training, Dropout randomly sets the output of each neuron to zero with a probability ppp, effectively removing it from the network for that iteration. This prevents neurons from overly relying on each other, encouraging the network to learn more robust and generalizable patterns.

Mathematically, the Dropout mechanism can be expressed as:

$$Output = Dropout(Input, p)$$

Where the output of each neuron is zeroed out with probability p. This technique is highly

effective in preventing overfitting in deep neural networks, especially in fully connected layers.

Early stopping is another simple yet powerful method for preventing overfitting. It involves monitoring the model's performance on a validation set during training and halting the training process when the performance on the validation set starts to deteriorate. This ensures the model does not overfit by learning noise and specific details unique to the training set.

The early stopping process can be implemented as follows:

1. Split the dataset into training and validation sets.
2. Train the model and evaluate its performance on the validation set after each epoch.
3. Stop training if the validation results do not improve for a specified number of epochs.

Early stopping helps find the optimal number of training iterations, balancing between underfitting and overfitting.

Data augmentation is another technique that involves artificially increasing the size and diversity of the training dataset by applying random transformations. This provides the model with a broader range of examples, helping it generalize better.

In image processing, common augmentation methods include:

- Rotation: Randomly rotating images within a specified range.
- Shift: Shifting images horizontally or vertically.
- Scaling: Resizing images to different scales.
- Flipping: Mirroring images horizontally or vertically.

For text data, augmentation methods may include:

- Synonym Replacement: Replacing words with their synonyms.
- Random Insertion: Adding random words to sentences.
- Back Translation: Translating text into another language and back to the original language.

Data augmentation effectively increases the diversity of the training dataset, reducing the risk of overfitting by making the model invariant to these transformations.

Finally, ***Ensemble methods*** involve combining predictions from multiple models to improve overall performance and reduce overfitting. By averaging predictions from several models, ensembles reduce the variance associated with individual models, leading to better generalization.

Let's consider the main ensembles:

1. ***Bagging*** (Bootstrap Aggregating) involves training multiple models on different subsets of the training dataset and averaging their predictions. This reduces variance and helps obtain a more reliable model. A popular application of this method to decision trees is Random Forests.
2. ***Boosting*** sequentially trains models where each new model focuses on correcting errors made by previous models. This method reduces both bias and variance, resulting in highly accurate models. Examples include AdaBoost and gradient boosters like GBM (Gradient Boosting Machines).
3. ***Stacking*** involves training multiple models and then using another model (meta-model) to combine their predictions. This allows leveraging the strengths of different models to achieve better performance [10,17].

2.3 Principle of Dropout

Let's take a deeper look at the Dropout method. Dropout is a regularization method aimed at reducing overfitting in deep neural networks (DNNs) by preventing neurons from adapting too closely to specific training examples. The principle of dropout involves temporarily disabling certain neurons with a given probability during training, which reduces the network's dependency on specific connections and prevents co-adaptation among neurons in various layers. This leads to the development of a structure that does not overfit specific noise or artifacts in the training data.

Research has shown that dropout enhances the generalization capabilities of neural networks in various tasks, including image classification, speech recognition, and text analysis, consistently achieving high performance on datasets such as SVHN, ImageNet, CIFAR100, and MNIST.

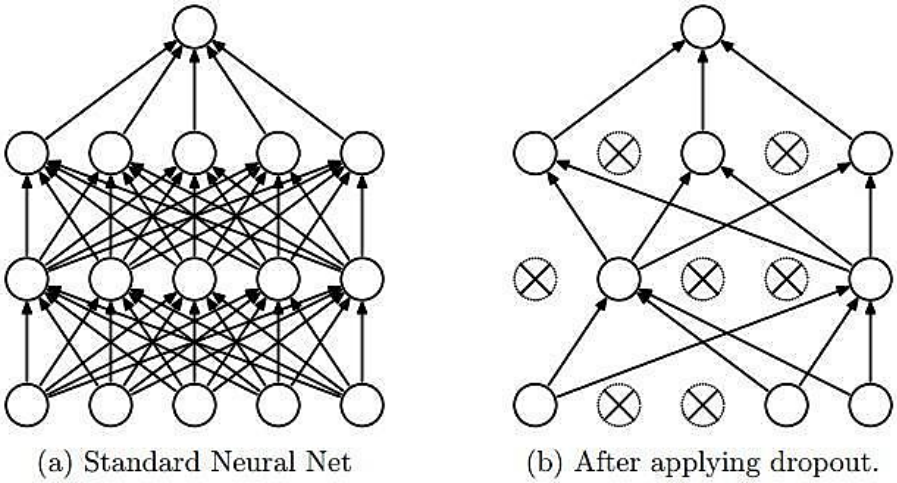

Fig. 2. Representation of the dropout method. Left: neural network before dropout is applied; right: the same network after dropout is applied [18]

However, it is important to note that applying dropout increases the training time compared to standard models with the same architecture. This is because each training example impacts a randomly modified network architecture, resulting in gradients that may not always align with the gradients of the final structure used during testing. Despite the increased training time, the stochastic updates of parameters inherent to dropout help mitigate the risk of overfitting and enhance model robustness.

The network shown on the left is used during testing after the parameters have been trained. The main idea of dropout is to train an ensemble of multiple DNNs and then average the results. This ensemble is formed by randomly dropping out neurons during the training phase with probability *p*, meaning each neuron is retained with probability *q=1−p*. The dropped neurons participate neither in the forward pass nor in the backpropagation, effectively training a different network in each forward and backward pass.

Let $h(x) = xW + b$ represent a linear projection of the input vector x with dimensions *di* into an output space of dimension *dh*, and let *a (h)* denote the activation function. Applying Dropout during training can be formulated as a modified activation function:

$$f(h) = Da(h)$$

where $D = (X_1, \ldots, X_{d_h})$ is a d_h-dimensional vector of Bernoulli random variables. Each *Xi* is defined as follows:

$$f(k;p) = \begin{cases} p, & if\ k=1 \\ 1-p, & if\ k=0 \end{cases}$$

where *k* represents all possible output values.

This distribution ensures that a neuron is dropped with probability *p* and retained with probability *q=1-p*. For the *i*-th neuron, the output O_i after applying Dropout is expressed as:

$$O_i = \begin{cases} a(\Sigma_{k=1}^{d_i} w_k x_k + b), & \text{если } X_i = 1 \\ 0, & \text{если } X_i = 0 \end{cases}$$

where $P = (X_i = 0) = p$.

During training, neurons are retained with probability *q*. However, during testing, to emulate the behavior of the ensemble networks used during training, activations are scaled by *q*:

- Training phase: $O_i = X_i a(\Sigma_{k=1}^{d_i} w_k x_k + b)$
- Testing phase: $O_i = q a(\Sigma_{k=1}^{d_i} w_k x_k + b)$

An alternative approach is inverted dropout, where scaling is applied during the training phase rather than during testing. The activation function is scaled by $\frac{1}{q}$ during training:

- Training phase: $O_i = \frac{1}{q} X_i a(\Sigma_{k=1}^{d_i} w_k x_k + b)$
- Testing phase: $O_i = a(\Sigma_{k=1}^{d_i} w_k x_k + b)$

This approach simplifies implementation in many deep learning systems, as it requires defining the model once and applying Dropout by changing one parameter during training. Considering a layer *h* with *n* neurons, each training step can be viewed as an ensemble of *n* Bernoulli trials with a success probability *p*. The number of dropped neurons follows a binomial distribution:

$$Y \sim Bin(d_h, p)$$

The probability of *k* successes (retained neurons) out of *n* trials is given by the binomial probability mass function:

$$f(k; n, p = \binom{n}{k} p^k (1-p)^{n-k}$$

where $\binom{n}{k}$ is the binomial coefficient, representing the number of ways to choose *k* successes out of *n* trials [18,19].

3. RESULTS AND DISCUSSION

Now, using this distribution, the modeling and visualization of the revenue distribution from N stores, where the probability of a successful sale varies in each store, was performed. This modeling is essential to understand how different probabilities affect sales outcomes, especially in environments where sales data are subject to variability and uncertainty. For this purpose, the binomial distribution was employed to model the number of successful sales in each store, and histograms were constructed for different values of success probability.

The implementation included the following steps:

1. Import necessary libraries: Python libraries such as `numpy`, `scipy.stats`, `pandas`, and `matplotlib` were used to facilitate the calculations and visualizations.
2. Set initial parameters: Parameters including the number of stores *(n_shops = 50)* and the number of trials *(n_days = 30)* were defined.
3. Determine the number of successful sales for different probabilities: For each probability *p*, the expected number of successful sales was calculated using the binomial probability mass function $P(Y = k) = \binom{n}{k} p^k (1-p)^{n-k}$.
4. Build histograms for different success probability values: Histograms were constructed to visualize the distribution of successful sales for each probability value.

For the binomial distribution *Bi(n, p)*, the probability that exactly *k* out of *n* trials will be successful is defined as:

$$P(Y = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

1. $\binom{n}{k} = \frac{n!}{k!(n-k)!} = 4060$ represents the number of combinations to choose *k* successful sales out of *n* trials.
2. $p^k = 0.1^3 = 0.001$ is the probability of success.
3. $(1-p)^{n-k} = (1-0.1)^{27} \approx 0.042391$

Substituting these values into the formula gives:

$$P(Y = 3) = 4060 \times 0.001 \times 0.042391 \approx 0.172$$

The following Python script was used to illustrate how many neurons will be deactivated for different values of *p* and a fixed number of *n*.

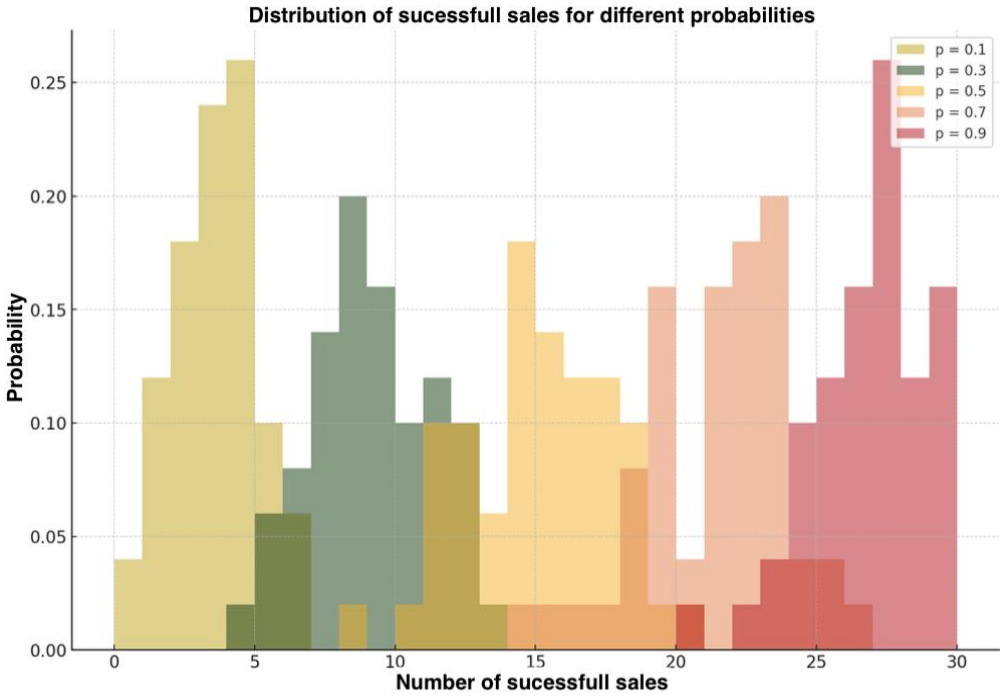

Fig. 3. Binomial distribution

Table 1. Python script

```
import numpy as np
from scipy.stats import binom
import pandas as pd
import matplotlib.pyplot as plt

# number of stores
n_shops = 50
# number of trials (days)
n_days = 30

# function for calculating the probability of successful sales
def calculate_probability(n, p, k):
    return binom.pmf(k, n, p)

# different probabilities of a successful sale
probs = [0.1, 0.3, 0.5, 0.7, 0.9]
expected_sales = [int(n_days * p) for p in probs]

# calculate the probability for each value of p
probabilities = [calculate_probability(n_days, p, k) for p, k in zip(probs, expected_sales)]

# result output
results = pd.DataFrame({
    'Probability (p)': probs,
    'Expected Sales': expected_sales,
    'Probability': probabilities
})

import ace_tools as tools; tools.display_dataframe_to_user(name="Sales Probabilities",
dataframe=results)

# histogramming
plt.figure(figsize=(12, 8))

for prob in probs:
    # generate size values from bi(n_days, prob)
    sales = binom.rvs(n_days, prob, size=n_shops)
    # draw a histogram of random values
    plt.hist(
        sales,
        bins=np.arange(0, n_days + 1, 1),
        density=True,
        color=np.random.rand(3,),
        alpha=0.5,
        label=f'p = {prob}')

plt.title('Distribution of successful sales for different probabilities')
plt.xlabel('Number of successful sales')
plt.ylabel('Probability')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```

Fig. 3 illustrates the distribution of the number of successful sales for different success probabilities. Each colored layer in the histogram represents a different probability of successful sales, allowing a comparison of how success probability affects the distribution of sales counts.

The binomial distribution model is particularly effective for scenarios where the outcome of interest is binary (success/failure) and the number of trials is fixed. When compared to other models such as the Poisson or Gaussian distributions, the binomial model accurately captures the variability in sales outcomes based on different success probabilities [20]. The Poisson distribution, for instance, is better suited for modeling the occurrence of rare events over time, but it does not inherently account for the fixed number of trials and binary outcomes central to the scenario analyzed here. Similarly, the Gaussian distribution may not effectively model the discrete outcomes of sales data, which further highlights the appropriateness of the binomial approach.

4. CONCLUSION

Thus, effectively overcoming overfitting in deep learning requires a comprehensive approach involving the application of various strategies and techniques. The methods discussed in the article, such as regularization, data augmentation, dropout, and ensemble methods, demonstrate significant effectiveness in reducing the risk of overfitting. Each method has its own characteristics and is best suited for specific types of tasks and data. Research shows that combining these methods can yield the best results in creating robust and reliable deep learning models. Understanding and correctly applying these approaches will enable researchers and developers to improve the generalization capability of models, which is critically important for their successful operation in real-world conditions.

DISCLAIMER (ARTIFICIAL INTELLIGENCE)

Author(s) hereby declare that NO generative AI technologies such as Large Language Models (ChatGPT, COPILOT, etc) and text-to-image generators have been used during writing or editing of this manuscript.

COMPETING INTERESTS

Authors have declared that they have no known competing financial interests or non-financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

1. Li H, Rajbahadur GK, Lin D, Bezemer CP, Jiang ZM. keeping deep learning models in check: A history-based approach to mitigate overfitting. IEEE Access; 2024 May 17.
2. Li H, Li J, Guan X, Liang B, Lai Y, Luo X. Research on overfitting of deep learning. In2019 15th international conference on computational intelligence and security (CIS). IEEE. 2019 Dec 13;78-81.
3. Santos CF, Papa JP. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. ACM Computing Surveys (CSUR). 2022 Sep 14;54(10s):1-25.
4. Liang J, Liu R. Stacked denoising autoencoder and dropout together to prevent overfitting in deep neural network. In2015 8th international congress on image and signal processing (CISP). IEEE. 2015 Oct 14;697-701.
5. Ying X. An overview of overfitting and its solutions. In Journal of physics: Conference series. IOP Publishing. 2019 Feb;1168:022022.
6. Overfitting: Preventing model risk by using methods to prevent overfitting. [Electronic resource];2024. Available:https://fastercapital.com/ru/content/.html (accessed 8.05.2024).
7. Pichugin RA. Methods of combating overfitting in neural networks. International Journal of Humanities and Natural Sciences. 2022;7-2(70):99-103.
8. In simple words, about the methods of solving problems with overfitting. [Electronic resource]; 2024. Available:https://newtechaudit.ru/overfitting/ (accessed 8.05.2024).
9. Kurgan NS, Sinitsyn VV, Gubanova AA. The problem of retraining in neural networks. Proceedings of the XV International Student Scientific Conference "Student Scientific Forum". [Electronic resource]; 2024. Available:https://scienceforum.ru/2023/article/2018032606 (accessed 8.05.2024).
10. Overfitting of the model: the essence and impact on the results. [Electronic resource]; 2024.

Available:https://www.decosystems.ru/pereobuchenie-modeli-v-mashinnom-obuchenii/ (accessed 8.05.2024).
11. Ponce C, Li R, Mao C, Vassilevski P. Multilevel-in-width training for deep neural network regression. Numerical Linear Algebra with Applications; 2023.
12. Fang P, Wang X, Ge J, Li J, Yang Z, Nai W. Elastic Network Regression Based on Sobol Sequence Initialized Lightning Attachment Procedure Optimization. 2023 IEEE 14th International Conference on Software Engineering and Service Science (ICSESS). 2023;252-257.
13. Long L, Lang J. Regularization for unsupervised learning of optical flow. Sensors (Basel, Switzerland); 2023.
14. Lu P, Rashid A, Kobyzev I, Rezagholizadeh M, Langlais P. LABO: Towards learning optimal label regularization via Bi-level optimization. 2023;ArXiv:5759-5774.
15. Wang C, Chen J, Liu Y. The elastic net regularized extreme learning machine for state of charge estimation. Journal of The Electrochemical Society; 2023.
16. Abramovich F. Statistical learning by sparse deep neural networks. 2023;ArXiv:abs/2311.08845.
17. Methods of combating overfitting of artificial neural networks. [Electronic resource]; 2024. Available:https://na-journal.ru/2-2019-tehnicheskie-nauki/1703-metody-borby-s-pereobucheniem-iskusstvennyh-neironnyh-setei (accessed 8.05.2024).
18. Dropout is a method for solving the problem of overfitting in neural networks. [Electronic resource]; 2024. Available:https://habr.com/ru/companies/wunderfund/articles/330814 / (accessed 8.05.2024).
19. Abrarov R. D. Implementation of the Dropout method, Young scientist. 2022; 43(438):1-5.
20. Sabiri B, El Asri B, Rhanoui M. Mechanism of overfitting avoidance techniques for training deep neural networks. InICEIS. 2022;1:418-427.

Peer-review history:
The peer review history for this paper can be accessed here:
https://www.sdiarticle5.com/review-history/120002