HyperParameter Tuning: Fixing Overfitting in Neural Networks

Last Updated : 16 Jul, 2024

Overfitting is a pervasive problem in neural networks, where the model becomes too specialized to the training data and fails to generalize well to new, unseen data. This issue can be addressed through hyperparameter tuning, which involves adjusting various parameters to optimize the performance of the model. In this article, we will delve into the technical aspects of hyperparameter tuning and its role in mitigating overfitting in neural networks.

Table of Content

Hyperparameter Tuning: A Solution to Overfitting

Overfitting occurs when a model is too complex relative to the amount of training data available. This complexity can lead to the model memorizing the training data rather than learning generalizable patterns. As a result, the model performs well on the training data but poorly on new data.

***Hyperparameter tuning* involves adjusting parameters that are set before training a model, such as learning rate, batch size, and number of hidden layers. The goal of hyperparameter tuning is to find the optimal combination of parameters that minimizes overfitting and maximizes the model's performance on unseen data.**

Signs of Overfitting

Signs of overfitting include:

- High accuracy on training data but significantly lower accuracy on validation or test data.
- Large discrepancies between training and validation loss.
- The model's performance improves on the training set but stagnates or worsens on the validation set.

Types of Hyperparameters

1. **Model Hyperparameters**: These include parameters that define the architecture of the model, such as the number of hidden layers, the number of neurons in each layer, and the type of activation functions used.
2. **Optimization Hyperparameters**: These include parameters that control the optimization process, such as the learning rate, batch size, and the type of optimizer used.
3. **Regularization Hyperparameters**: These include parameters that control the regularization techniques used to prevent overfitting, such as dropout rates and L1/L2 regularization strengths.

Hyperparameters in Neural Networks

- **Learning Rate:** The learning rate determines how quickly a model updates its parameters during training. A high learning rate can speed up training but may cause the model to converge to a suboptimal solution. Conversely, a low learning rate ensures more precise updates but can make the training process slow.

- **Number of Hidden Layers and Neurons:** The architecture of a neural network, including the number of hidden layers and neurons per layer, significantly affects its capacity to learn complex patterns. More layers and neurons can capture more intricate relationships but also increase the risk of overfitting.
- **Batch Size:** Batch size refers to the number of training examples used in one iteration of model training. Smaller batch sizes can provide more accurate gradient estimates but increase training time. Larger batch sizes speed up training but may lead to less stable updates.
- **Epochs:** An epoch is one complete pass through the entire training dataset. The number of epochs determines how many times the learning algorithm will work through the entire training set. Too few epochs can lead to underfitting, while too many can cause overfitting.
- **Activation Functions:** Activation functions introduce non-linearity into the model, enabling it to learn complex patterns. Common activation functions include ReLU, Sigmoid, and Tanh. The choice of activation function can impact the model's performance and convergence rate.

Hyperparameter Tuning Methods

1. **Grid Search**: This involves trying all possible combinations of hyperparameters and selecting the best combination based on the model's performance.
2. **Random Search**: This involves randomly sampling hyperparameters from a predefined range and selecting the best combination based on the model's performance.
3. **Bayesian Optimization**: This involves using Bayesian methods to search for the optimal hyperparameters.
4. **Gradient-Based Optimization**: This involves using gradient-based methods to search for the optimal hyperparameters.

Case Study: Hyperparameter Tuning for Image Classification

In this case study, we will use the CIFAR-10 dataset to demonstrate the effectiveness of hyperparameter tuning in mitigating overfitting. We will use a convolutional neural network (CNN) and tune the following hyperparameters:

- Learning rate
- Batch size
- Number of hidden layers
- Dropout rate

We will use a grid search to find the optimal combination of hyperparameters and evaluate the model's performance on both the training and validation sets.

To demonstrate hyperparameter tuning for image classification using the CIFAR-10 dataset with a convolutional neural network (CNN), we can use the `Keras` library along with `Scikit-learn` for performing a grid search. Below is the Python code implementation for this case study:

Let's install wrapper first

```
pip install scikeras
```

Step 1: Load the CIFAR-10 dataset:

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.optimizers import Adam
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV

# Load the CIFAR-10 dataset
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Step 2: Define the function to create the CNN model:

```
# Define the function to create the CNN model
def create_model(learning_rate=0.001, dropout_rate=0.5,
num_hidden_layers=1):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32,
32, 3)))
    model.add(MaxPooling2D((2, 2)))

    for _ in range(num_hidden_layers):
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

Step 3: Create a KerasClassifier wrapper for the model:

```
# Create a KerasClassifier wrapper for the model
model = KerasClassifier(model=create_model, epochs=10, batch_size=32,
verbose=0)
```

Step 4: Define the grid of hyperparameters to search:

```
# Define the grid of hyperparameters to search
param_grid = {
    'model__learning_rate': [0.001, 0.0001],
    'model__dropout_rate': [0.3, 0.5],
    'model__num_hidden_layers': [1, 2],
    'batch_size': [32, 64]
}

# Perform the grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,
```

```
verbose=1)
grid_result = grid.fit(x_train, y_train)
```

Output:

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

Step 5 : Evaluate the best parameters and model on the test set:

```
# Print the best parameters and best score
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))

# Evaluate the best model on the test set
best_model = grid_result.best_estimator_.model_
test_loss, test_accuracy = best_model.evaluate(x_test, y_test)
print("Test accuracy:", test_accuracy)
```

Output:

```
Best: 0.72 using {'batch_size': 32, 'model__dropout_rate': 0.3,
'model__learning_rate': 0.001, 'model__num_hidden_layers': 2}

Test accuracy: 0.70
```

Best Practices for Hyperparameter Tuning

- **Choosing the Right Metrics:** Selecting appropriate evaluation metrics is crucial for tuning hyperparameters. Metrics should align with the model's goals and provide meaningful insights into its performance.
- **Balancing Exploration and Exploitation:** Effective hyperparameter tuning requires balancing exploration (trying diverse hyperparameter values) and exploitation (focusing on promising configurations). Techniques like Bayesian optimization excel at this balance.
- **Computational Considerations:** Hyperparameter tuning can be computationally expensive. Leveraging distributed computing, parallel processing, and efficient algorithms can help manage the computational load and speed up the tuning process.

Conclusion

Hyperparameter tuning is essential for optimizing neural network performance and preventing overfitting. Techniques like grid search, random search, and Bayesian optimization help identify the best hyperparameters. Strategies such as regularization, dropout, early stopping, data augmentation, and cross-validation are effective in mitigating overfitting.

Next Article

Using Early Stopping to Reduce Overfitting in Neural Networks

Similar Reads

SVM Hyperparameter Tuning using GridSearchCV | ML

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. However, the...

4 min read

Random Forest Hyperparameter Tuning in Python

In this article, we shall implement Random Forest Hyperparameter Tuning in Python using Sci-kit Library. Sci-kit aka Sklearn is a Machine Learning librar...

5 min read

Cross-validation and Hyperparameter tuning of LightGBM Model

In a variety of industries, including finance, healthcare, and marketing, machine learning models have become essential for resolving challenging...

14 min read

CatBoost Cross-Validation and Hyperparameter Tuning

CatBoost is a powerful gradient-boosting algorithm of machine learning that is very popular for its effective capability to handle categorial features of...

11 min read

Hyperparameter tuning

A Machine Learning model is defined as a mathematical model with several parameters that need to be learned from the data. By training a model with...

11 min read

Hyperparameter tuning using GridSearchCV and KerasClassifier

Hyperparameter tuning is done to increase the efficiency of a model by tuning the parameters of the neural network. Some scikit-learn APIs like...

2 min read

How to tune a Decision Tree in Hyperparameter tuning

Decision trees are powerful models extensively used in machine learning for classification and regression tasks. The structure of decision trees resemble...

14 min read

Hyperparameter Tuning in Linear Regression

Linear regression is one of the simplest and most widely used algorithms in machine learning. Despite its simplicity, it can be quite powerful, especially...

7 min read

Hyperparameter tuning SVM parameters using Genetic Algorithm

The performance support Vector Machines (SVMs) are heavily dependent on hyperparameters such as the regularization parameter (C) and the kernel...

9 min read

Hyperparameter tuning with Ray Tune in PyTorch

Hyperparameter tuning is a crucial step in the machine learning pipeline that can significantly impact the performance of a model. Choosing the right set...

8 min read

Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar

Commerce

Accountancy
Business Studies
Economics
Management
HR Management
Finance
Income Tax

Databases

SQL
MYSQL
PostgreSQL
PL/SQL
MongoDB

Preparation Corner

Company-Wise Recruitment Process
Resume Templates
Aptitude Preparation
Puzzles
Company-Wise Preparation
Companies
Colleges

Competitive Exams

JEE Advanced
UGC NET
UPSC
SSC CGL
SBI PO
SBI Clerk
IBPS PO
IBPS Clerk

More Tutorials

Software Development
Software Testing
Product Management
Project Management
Linux
Excel
All Cheat Sheets
Recent Articles

Free Online Tools

Typing Test
Image Editor
Code Formatters
Code Converters
Currency Converter
Random Number Generator
Random Password Generator

Write & Earn

Write an Article
Improve an Article
Pick Topics to Write
Share your Experiences
Internships

DSA/Placements

DSA - Self Paced Course
DSA in JavaScript - Self Paced Course
DSA in Python - Self Paced
C Programming Course Online - Learn C with Data Structures
Complete Interview Preparation
Master Competitive Programming

Development/Testing

JavaScript Full Course
React JS Course
React Native Course
Django Web Development Course
Complete Bootstrap Course
Full Stack Development - [LIVE]

Core CS Subject for Interview Preparation
Mastering System Design: LLD to HLD
Tech Interview 101 - From DSA to System Design [LIVE]
DSA to Development [HYBRID]
Placement Preparation Crash Course [LIVE]

JAVA Backend Development - [LIVE]
Complete Software Testing Course [LIVE]
Android Mastery with Kotlin [LIVE]

Machine Learning/Data Science

Complete Machine Learning & Data Science Program - [LIVE]
Data Analytics Training using Excel, SQL, Python & PowerBI - [LIVE]
Data Science Training Program - [LIVE]
Mastering Generative AI and ChatGPT
Data Science Course with IBM Certification

Programming Languages

C Programming with Data Structures
C++ Programming Course
Java Programming Course
Python Full Course

Clouds/Devops

DevOps Engineering
AWS Solutions Architect Certification
Salesforce Certified Administrator Course

GATE

GATE CS & IT Test Series - 2025
GATE DA Test Series 2025
GATE CS & IT Course - 2025
GATE DA Course 2025