# CHAPTER 6: Regularization and Hyperparameter Tuning

## Overview

In this chapter, you will be introduced to hyperparameter tuning. You will get hands-on experience in using TensorFlow to perform regularization on deep learning models to reduce overfitting. You will explore concepts such as L1, L2, and dropout regularization. Finally, you will look at the Keras Tuner package for performing automatic hyperparameter tuning.

By the end of the chapter, you will be able to apply regularization and tune hyperparameters in order to reduce the risk of overfitting your model and improve its performance.

## Introduction

In the previous chapter, you learned how classification models can solve problems when the response variable is discrete. You also saw different metrics used to assess the performance of such classifiers. You got hands-on experience in building and training binary, multi-class, and multi-label classifiers with TensorFlow.

When evaluating a model, you will face three different situations: model overfitting, model underfitting, and model performing. The last one is the ideal scenario, in which a model is accurately predicting the right outcome and is generalizing to unseen data well.

If a model is underfitting, it means it is neither achieving satisfactory performance nor accurately predicting the target variable. In this case, a data scientist can try tuning different hyperparameters and finding the best combination that will boost the accuracy of the model. Another possibility is to improve the input dataset by handling issues such as the cleanliness of the data or feature engineering.

A model is overfitting when it can only achieve high performance on the training set and performs poorly on the test set. In this case, the model has only learned patterns from the data relevant to the data used for training. Regularization helps to lower the risk of overfitting.

## Regularization Techniques

The main goal of a data scientist is to train a model that achieves high performance and generalizes to unseen data well. The model should be able to predict the right outcome on both data used during the training process and new data. This is the reason why a model is always assessed on the test set. This set of data serves as a proxy to evaluate the ability of the model to output correct results while in production.

Chart, scatter chart

Description automatically generated

Figure 6.1: Model not overfitting or underfitting

In Figure 6.1, the linear model (line) seems to predict relatively accurate results for both the training (circles) and test (triangles) sets.

But sometimes a model fails to generalize well and will overfit the training set. In this case, the performance of the model will be very different between the training and test sets.

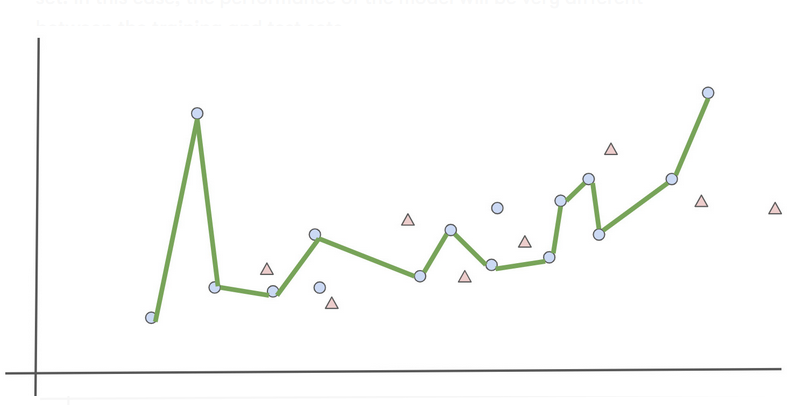


Figure 6.2: Model overfitting

Figure 6.2 shows the model (line) has only learned to predict accurately for the training set (circles) and is performing badly on the test set (triangles). This model is clearly overfitting.

Fortunately, there are **regularization techniques** that a data scientist can use to reduce and prevent overfitting, defined in the following sections.

## L1 Regularization

For deep learning models, overfitting happens when some of the features have higher weights than they should. The model puts too much emphasis on these features as it believes they are extremely important for predicting the training set. Unfortunately, these features are less relevant for the test set or any new unseen data. Regularization techniques try to penalize such weights and reduce their importance to the model predictions.

There are multiple ways to perform regularization. One of them is to add a regularization component to the cost function:

Figure 6.3: Adding a regularization component to the cost function


Figure 6.3: Adding a regularization component to the cost function

The addition of this regularization component will lead the weights of the model to be smaller as neural networks try to reduce the cost function while performing forward and backward propagations.

One very popular regularization component is L1. Its formula is as follows:

Figure 6.4: L1 regularization


Figure 6.4: L1 regularization

Formulais a hyperparameter that defines the level of penalization of the L1 regularization. W is the weight of the model. With L1 regularization, you add the sum of the absolute value of the weights to the model loss.

L1 regularization is sometimes referred to as feature selection as it tends to push the weights of non-relevant features to 0. Therefore, only the relevant features are used for making predictions.

In TensorFlow, you can define L1 regularization with the following code snippet:

*from tensorflow.keras.regularizers import l1*

*l1\_reg = l1(l=0.01)*

The l parameter corresponds to the Formula 2hyperparameter. The instantiated L1 regularization can then be added to any layer from TensorFlow Keras:

*from tensorflow.keras.layers import Dense*

*Dense(10, kernel\_regularizer=l1\_reg)*

In the preceding example, you added the L1 regularizer that you defined earlier to a fully connected layer of 10 units.

## L2 Regularization

L2 regularization is similar to L1 in that it adds a regularization component to the cost function, but its formula is different:

Figure 6.5: L2 regularization


Figure 6.5: L2 regularization

L2 regularization tends to decrease the weights of the non-relevant features. They will be close to 0, but not exactly 0. So, it reduces the impact of these features but does not disable them as L1 does.

In TensorFlow, you can define L2 regularization as follows:

*from tensorflow.keras.regularizers import l2*

*from tensorflow.keras.layers import Dense*

*l2\_reg = l2(l=0.01)*

*Dense(20, kernel\_regularizer=l2\_reg)*

In the preceding example, you defined an L2 regularizer and added it to a fully connected layer of 20 units.

TensorFlow provides another regularizer class that combines both L1 and L2 regularizers. You can instantiate it with the following code snippet:

*from tensorflow.keras.regularizers*

*import l1\_l2*

*l1\_l2\_reg = l1\_l2(l1=0.01, l2=0.001)*

In the preceding example, you instantiated L1 and L2 regularizers and specified the factors for L1 and L2 as 0.01 and 0.001, respectively. You can observe that more weights are put on the L1 regularization compared to L2. These values are hyperparameters that can be fine-tuned depending on the dataset.

In the next exercise, you will put this into practice as you apply L2 regularization to a model.