DEEP LEARNING WITH KERAS WORKSHOP

MUHAMMAD RAJABINASAB @ TARBIAT MODARES UNIVERSITY

Chapter 5 :

Improving Model Accuracy

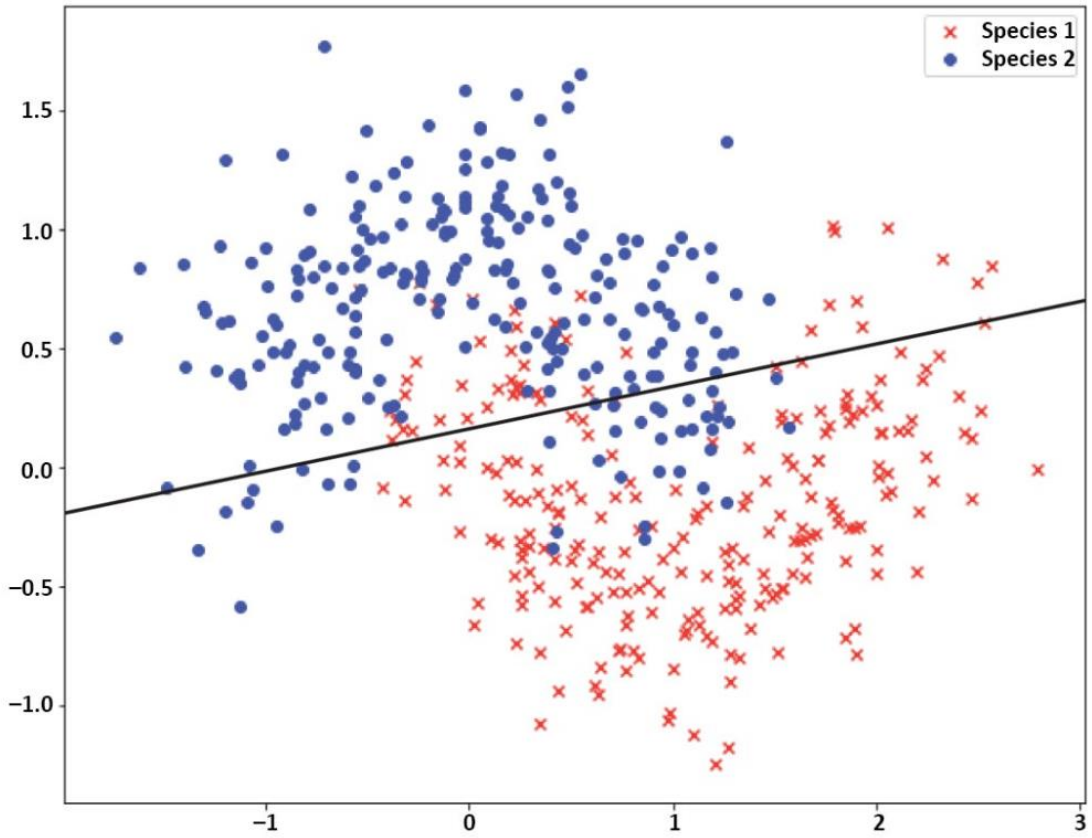
INTRODUCTION

- Deep learning is not only about building neural networks, training them using an available dataset, and reporting the model accuracy
- It involves trying to understand your model and the dataset, as well as moving beyond a basic model by improving it in many aspects
- In this chapter, you will learn about two very important groups of techniques for improving models
- These techniques are regularization methods and hyperparameter tuning

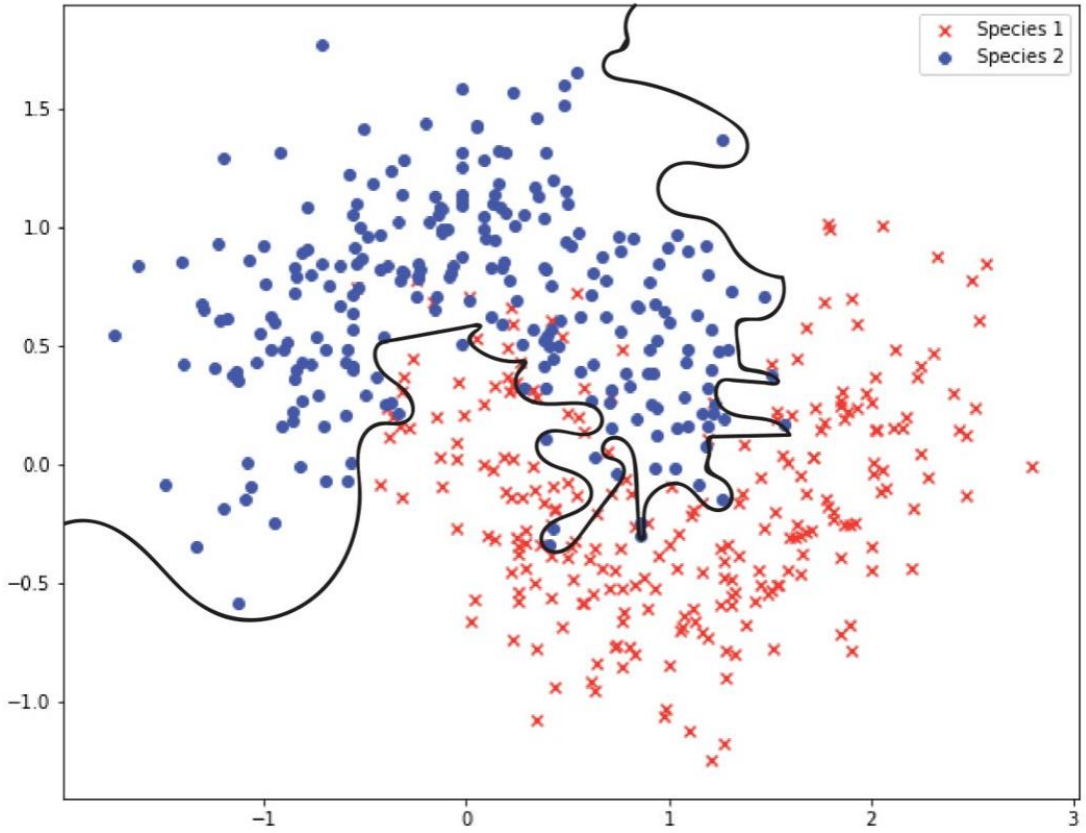
REGULARIZATION

- Since deep neural networks are highly flexible models, overfitting is an issue that can often arise when training them
- Therefore, one very important part of becoming a deep learning expert is knowing how to detect and address overfitting
- If a model overfits, it performs with high accuracy during the training process but does not perform well on new examples
- We are interested in high accuracy on new examples

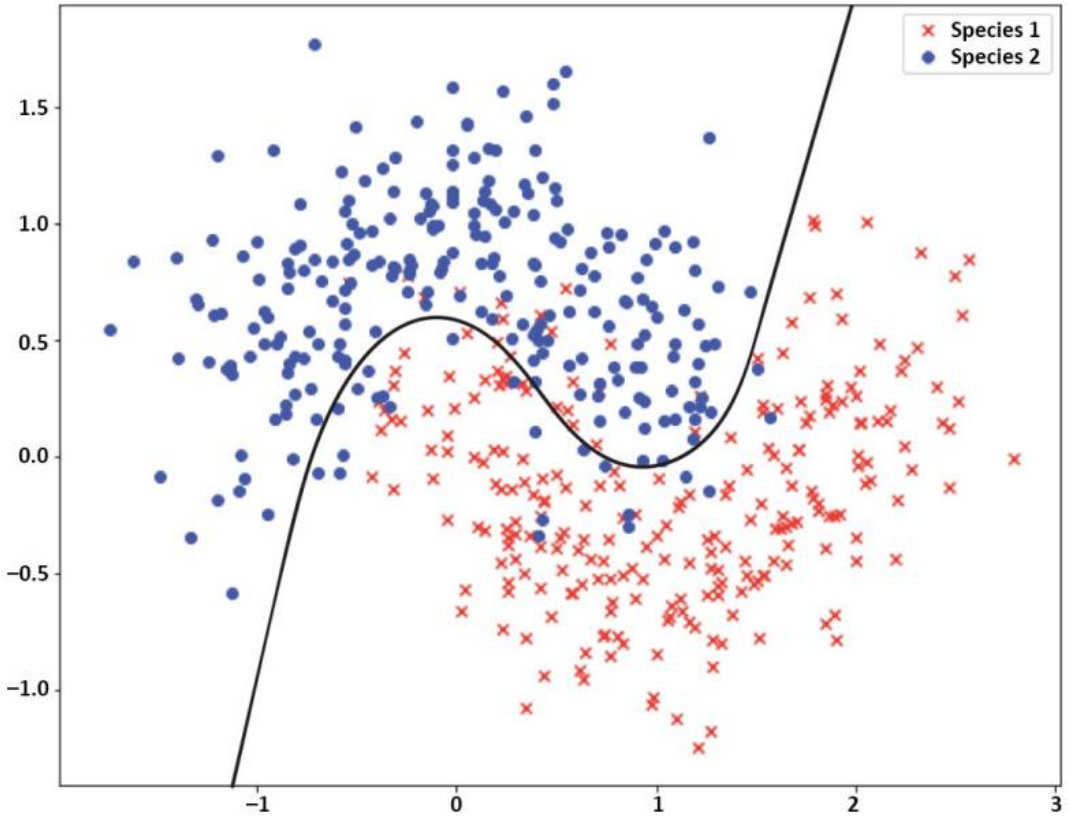
UNDERFITTING



OVERFITTING



RIGHT FIT



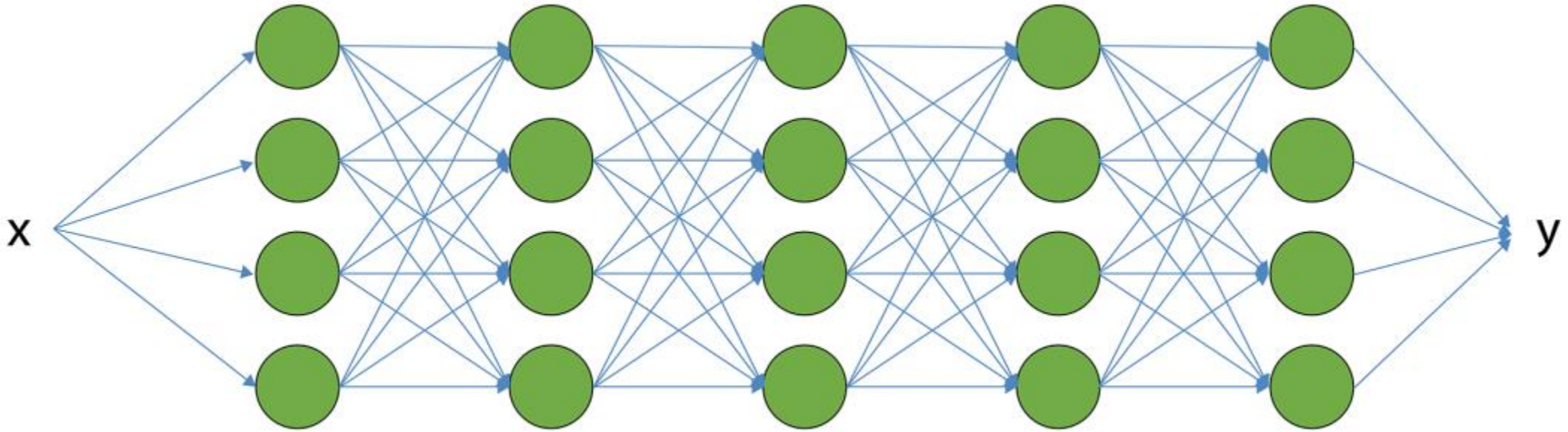
REDUCING OVERFITTING WITH REGULARIZATION

- Regularization methods try to modify the learning algorithm in a way that reduces the variance of the model
- By decreasing the variance, regularization techniques intend to reduce the generalization error while not increasing the training error
- Regularization methods provide some kind of restriction that helps with the stability of the model
- One of the most common ways of performing regularization on deep neural networks is by putting some type of penalizing term on weights to keep the weights small

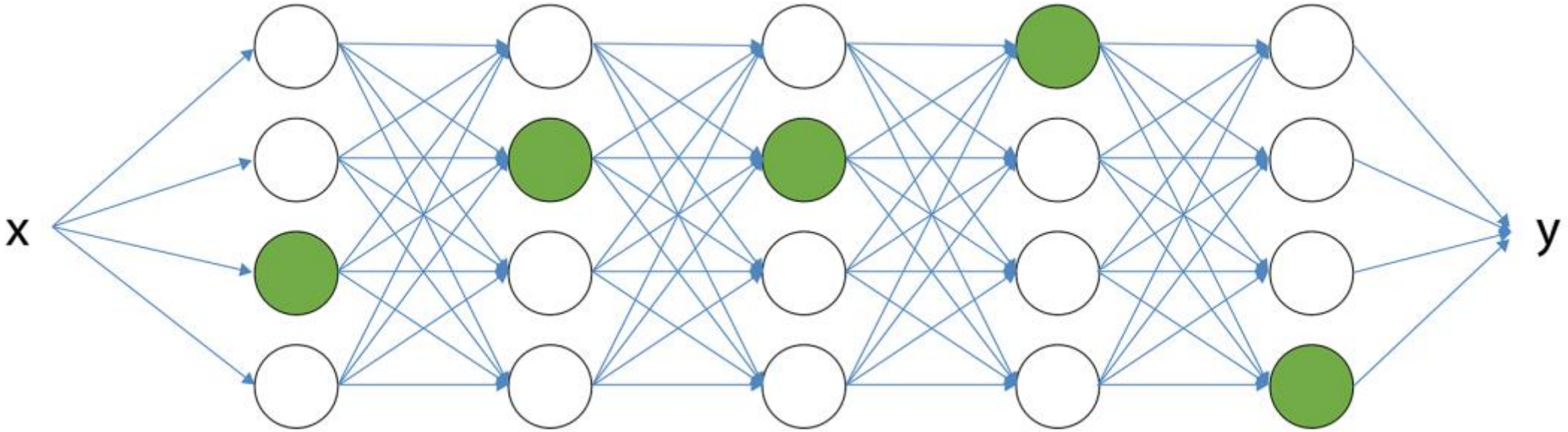
REDUCING OVERFITTING WITH REGULARIZATION

- Keeping the weights small makes the network less sensitive to noise in individual data examples
- Weights in a neural network are, in fact, the coefficients that determine how big or small an effect each processing unit will have on the final output of the network
- Training a large neural network where each unit has little or no effect on the output is the equivalent of training a much simpler network, and so variance and overfitting are reduced

LARGE WEIGHTS



SMALL WEIGHTS



L1 AND L2 REGULARIZATION

- The most common type of regularization for deep learning models is the one that keeps the weights of the network small
- This type of regularization is called weight regularization and has two different variations: L2 regularization and L1 regularization
- In weight regularization, a penalizing term is added to the loss function. This term is either the L2 norm (the sum of the squared values) of the weights or the L1 norm (the sum of the absolute values) of the weights

L1 AND L2 REGULARIZATION

- If the L1 norm is used, then it will be called L1 regularization. If the L2 norm is used, then it will be called L2 regularization. Therefore:
- L1 Regularization:

$$Loss\ function = Old\ loss\ function + lambda * sum\ of\ absolute\ values\ of\ the\ weights$$

- L2 Regularization:

$$Loss\ function = Old\ loss\ function + lambda * sum\ of\ squared\ values\ of\ the\ weights$$

L1 AND L2 REGULARIZATION

- Lambda can take any value between 0 and 1, where lambda=0 means no penalty at all and lambda=1 means full penalty
- The right value for lambda can be selected by trying out different values and observing which value provides a lower generalization error
- It's good practice to start with a network with no regularization and observe the results. Then, you should perform regularization with increasing values of lambda, such as 0.001, 0.01, 0.1, 0.5, …, and observe the results in each case in order to figure out how much penalizing on the weight's values is suitable for a particular problem

L1 AND L2 REGULARIZATION

- L2 regularization is the most common regularization technique that's used in machine learning in general
- The difference between L1 regularization and L2 regularization is that L1 results in a sparser weights matrix
- It is also possible to perform both L1 and L2 regularization at the same time

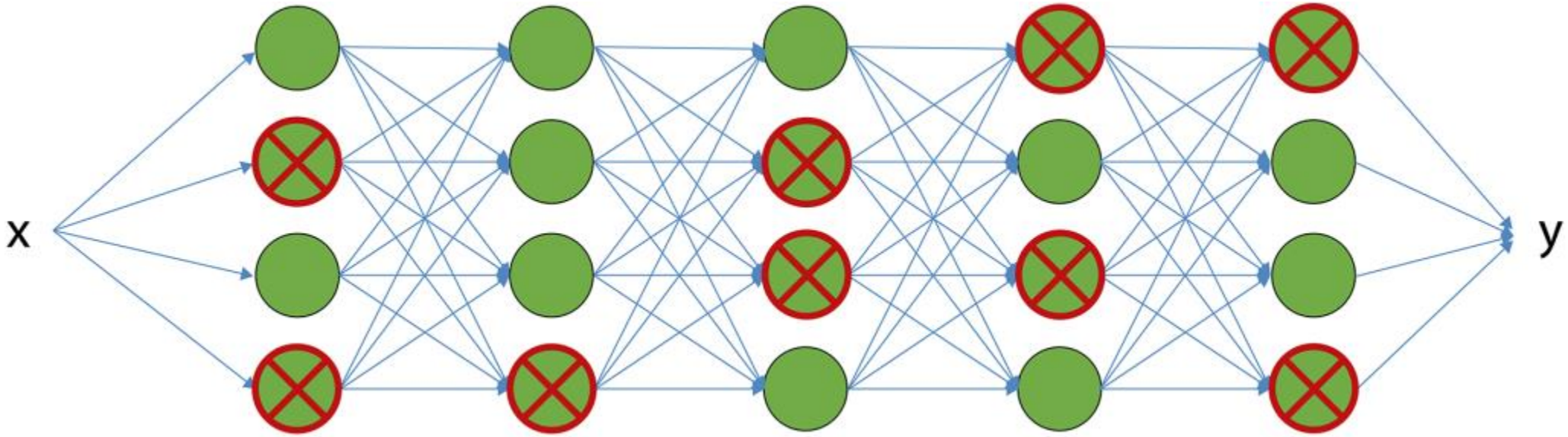
ACTIVITY 5.01: WEIGHT REGULARIZATION ON AN AVILA PATTERN CLASSIFIER

- The Avila dataset has been extracted from 800 images of the Avila Bible, a giant 12th-century Latin copy of the Bible
- The dataset consists of various features about the images of the text, such as intercolumnar distance and the margins of the text
- The dataset also contains a class label that indicates if a pattern of the image falls into the most frequently occurring category or not
- In this activity, you will build a Keras model to perform classification on this dataset according while applying different types of weight regularization

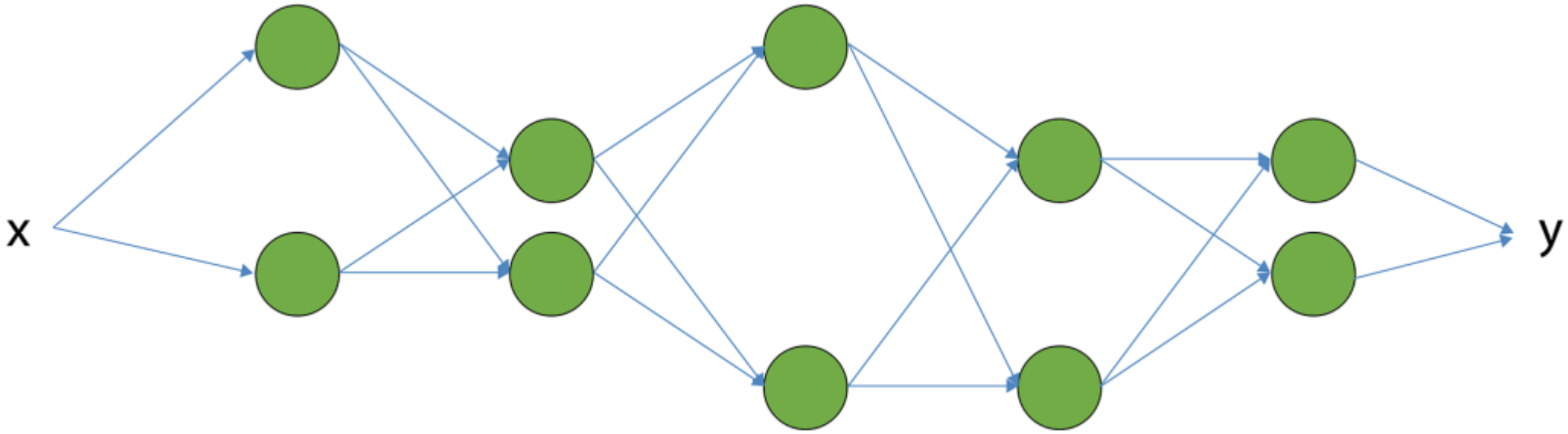
DROPOUT REGULARIZATION

- Dropout regularization works by randomly removing nodes from a neural network during training
- More precisely, dropout sets up a probability on each node. This probability refers to the chance that the node is included in the training at each iteration of the learning algorithm
- Imagine we have a large neural network where a dropout chance of 0.5 is assigned to each node. In such a case, at each iteration, the learning algorithm flips a coin for each node to decide whether that node will be removed from the network or not

DROPOUT REGULARIZATION



DROPOUT REGULARIZATION



DROPOUT REGULARIZATION

- This process is repeated at each iteration
- When some nodes are chosen to be removed in an iteration of a learning algorithm, it means that they won't participate in the parameter-updating process
- It is important to keep in mind that to evaluate the performance of the model on the test set or hold-out set, the original complete network is always used

DROPOUT REGULARIZATION

- By removing nodes from a network, we are performing training on a smaller network in comparison to the original network
- By randomly removing inputs at each layer in a deep neural network, the overall network becomes less sensitive to single inputs
- Now we can move on to implementing dropout regularization in Keras

EXERCISE 5.01

- Our simulated dataset represents various measurements of trees, such as height, the number of branches, and the girth of the trunk at the base
- Our goal is to classify the records into either deciduous (a class value of 1) or coniferous (a class value of 0) type trees based on the
- The dataset consists of 10000 records that consist of two classes, representing the two tree types, and each data example has 10 feature values measurements given

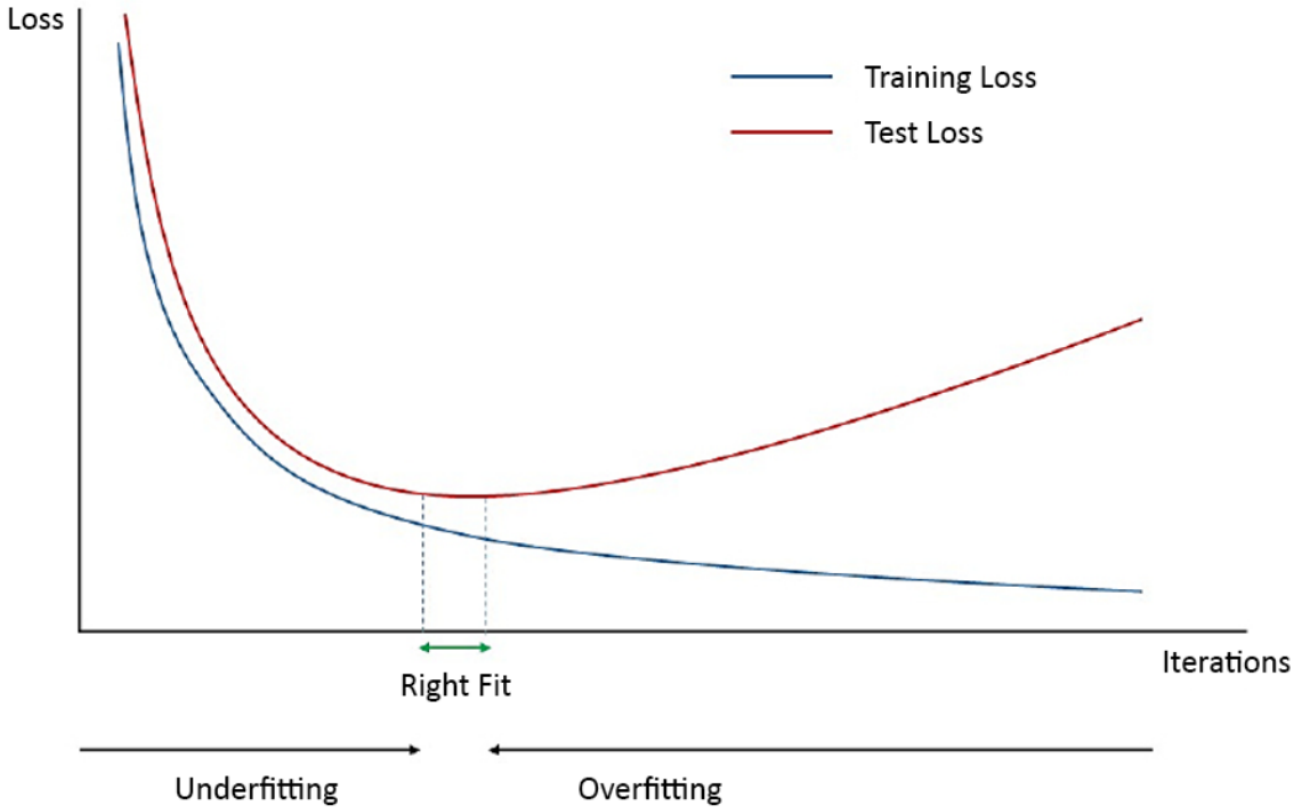
ACTIVITY 5.02: DROPOUT REGULARIZATION ON THE TRAFFIC VOLUME DATASET

- Based on dataset from chapter 3 and chapter 4
- You will use the training set/test set approach to train and evaluate the model, plot the trends in training error and the generalization error, and observe the model overfitting data examples
- You will attempt to improve model performance by addressing the overfitting issue through the use of dropout regularization

EARLY STOPPING

- By monitoring the training process, we can train the model just enough for it to prevent overfitting
- This is the main idea behind early stopping

EARLY STOPPING



EXERCISE 5.02

- In this exercise we work on the dataset from exercise 5.01
- We will learn to implement early stopping using keras

DATA AUGMENTATION

- Data augmentation is a regularization technique that tries to address overfitting by training the model on more training examples in an inexpensive way
- In data augmentation, the available data is transformed in different ways and fed to the model as new training data
- This type of regularization has been shown to be effective, especially for some specific applications, such as object detection/recognition in computer vision and speech processing

ADDING NOISE

- One way to expand the training data and reduce overfitting is to generate new data examples by injecting noise into the available data
- This type of regularization has been shown to reduce overfitting to an extent that is comparable to weight regularization techniques
- In Keras, you can easily define noise as a layer and add it to your model

ADDING NOISE

- For example, to add Gaussian noise with a standard deviation of 0.1 (the mean is equal to 0) to your model, you can write this:

```
from keras.layers import GaussianNoise
model.add(GaussianNoise(0.1))
```

ADDING NOISE

- The following code will add Gaussian noise to the outputs/activations of the first hidden layer of the model:

```
model = Sequential()
model.add(Dense(4, input_dim=30,
activation='relu'))
model.add(GaussianNoise(0.01))
model.add(Dense(4, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

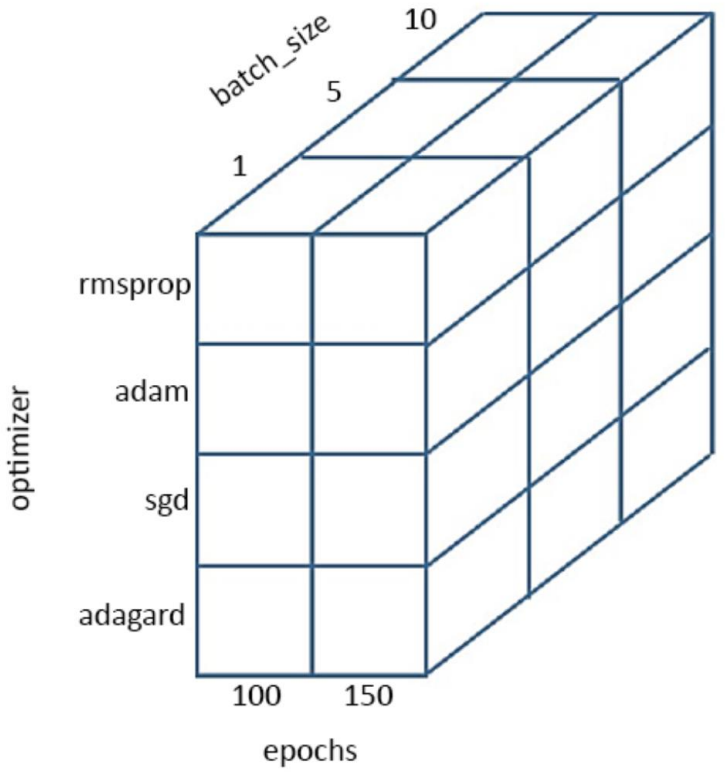
HYPERPARAMETER TUNING WITH SCIKIT-LEARN

- Hyperparameter tuning is a very important technique for improving the performance of deep learning models
- In the previous chapter, you learned how to perform hyperparameter tuning by writing user-defined functions to loop over possible values for each hyperparameter
- In this section, you will learn how to perform it in a much easier way by using the various hyperparameter optimization methods that are available in scikit-learn

GRID SEARCH WITH SCIKIT-LEARN

- Having a range or a set of possible values for each hyperparameter that we are interested in tuning can create a grid
- Therefore, hyperparameter tuning can be seen as a grid search problem
- we would like to try every cell in the grid (every possible combination of hyperparameters) and find the one cell that results in the best performance for the model

GRID SEARCH WITH SCIKIT-LEARN



RANDOMIZED SEARCH WITH SCIKIT-LEARN

- As you may have realized, an exhaustive grid search may not be the best choice for tuning the hyperparameters of a deep learning model since it is not very efficient
- An alternative way to perform hyperparameter optimization is to perform random sampling on the grid and perform k-fold cross-validation on some randomly selected cells

RANDOMIZED SEARCH WITH SCIKIT-LEARN

- It could be argued that randomized search is not as good as grid search since it does not consider all the possible values and combinations of values for hyperparameters
- one smart way of performing hyperparameter tuning for deep learning models is to start with either randomized search on many hyperparameters, or grid search on fewer hyperparameters with larger gaps between them

RANDOMIZED SEARCH WITH SCIKIT-LEARN

- By beginning with a randomized search on many hyperparameters, we can determine which hyperparameters have the most influence on a model's performance
- It can also help narrow down the range for important hyperparameters
- Then, you can complete your hyperparameter tuning by performing grid search on the smaller number of hyperparameters and the smaller ranges for each of them
- This is called the coarse-to-fine approach to hyperparameter tuning

ACTIVITY 5.03: HYPERPARAMETER TUNING ON THE AVILA PATTERN CLASSIFIER

- Based on dataset from activity 5.01
- In this activity, you will build a Keras model similar to those in the previous activities but this time, you will add regularization methods to your model as well
- Then, you will use scikit-learn optimizers to perform tuning on the model hyperparameters, including the hyperparameters of the regularizers

SUMMARY

- In this chapter, we learned about regularization
- We added several regularization methods to our deep learning model such as L1 and L2 regularization, dropout and early stopping
- We tuned our hyperparameters using grid search and randomized search and we learned how to combine them in order to find the best hyperparameters values faster

THANKS FOR YOUR ATTENTION