

Beginning AI Bot Frameworks

Getting Started with Bot Development

Manisha Biswas



Beginning AI Bot Frameworks

**Getting Started with Bot
Development**

Manisha Biswas

Apress®

Beginning AI Bot Frameworks: Getting Started with Bot Development

Manisha Biswas

Kolkota, West Bengal, India

ISBN-13 (pbk): 978-1-4842-3753-3

<https://doi.org/10.1007/978-1-4842-3754-0>

ISBN-13 (electronic): 978-1-4842-3754-0

Library of Congress Control Number: 2018956746

Copyright © 2018 by Manisha Biswas

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image, we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr

Acquisitions Editor: Natalie Pao

Development Editor: James Markham

Coordinating Editor: Jessica Vakili

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-3753-3. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

I dedicate this book to my parents.

Table of Contents

About the Author	ix
About the Technical Reviewer	xi
Acknowledgments	xiii
Chapter 1: AI and Bot Basics	1
Artificial Intelligence	1
Classification	8
Prediction	8
Interconnection Between AI, ML, and DL	10
Chatbots.....	14
Generative Chatbot Model	14
How Do Chatbots Work?	15
Rise of the Chatbots—Conversational Commerce	17
Growth of Chat Apps.....	19
The Structure of a Bot	21
Bot Frameworks	23
Conclusion	23
Chapter 2: Microsoft Bot Framework	25
Starting with the Prerequisites	26
Visual Studio.....	26
Windows 10.....	27
Bot Builder	27
Bot Framework Emulator.....	28

TABLE OF CONTENTS

Creating a Simple Bot Framework App	30
Using a Template to Create the Project	30
Working with the Code	33
Running the Application.....	42
Testing the Application	44
Managing State.....	46
Understanding the Use of Dialogs.....	47
Publishing the Bot to the Cloud by Using Azure	56
Conclusion	65
Chapter 3: Wit.ai and Dialogflow	67
Getting Started with Wit.ai	67
Creating a New App	67
Adding Intent	70
Adding Text and Keywords	73
Creating a New Entity	77
Implementing Wit.ai with Facebook.....	78
Working with Dialogflow	84
Accessing Dialogflow	85
Creating the Pizza Bot	87
Using Small Talk	88
Linking to a Google Project.....	90
Adding an Intent	91
Creating a New Entity	92
Deploying the Bot	94
Integration with Web Instance	98
Conclusion	100

TABLE OF CONTENTS

Chapter 4: IBM Watson Chatbots	101
Implementing Watson	102
IBM Cloud	102
Watson Assistant Service	104
Creating a FAQ Bot	106
Creating Intent for the Bot	108
The Dialog Flow for the App	112
Making Sense of the Flow	113
Trying the Bot	115
Creating a Coffee Bot	117
Creating a Workspace	117
Working with Intents	118
Working with Entities	126
Working with Dialogs	129
Nested Intents	133
Conclusion	137
Chapter 5: Chatbot with TensorFlow.....	139
TensorFlow Basics	139
Setting Up the Working Environment.....	140
Creating a Neural Network.....	143
Working with the Activation Function	149
TensorBoard	151
Versions of TensorFlow.....	158
Keras Overview	159
Getting Started with a Keras Chatbot	163
Introducing nmt-chatbot	166

TABLE OF CONTENTS

End-to-End Systems	172
Recurrent Neural Network.....	172
Working with a Seq2seq Bot.....	176
Instructions for Updating	178
Download and Install CUDA.....	178
Download and Install cuDNN	180
Uninstall TensorFlow, Install TensorFlow GPU.....	181
Update the %PATH% on the System.....	181
Conclusion	182
Index.....	183

About the Author

Manisha Biswas has a Bachelor of Technology degree in information technology and is currently working as a data scientist at Prescriber360, in Kolkata, India. She is involved with several areas of technology, including web development, IoT, soft computing, and artificial intelligence. She is also an Intel Software Innovator and was awarded the Shri Dewang Mehta IT Award in 2016 by NASSCOM. Biswas recently formed a Women in Technology community in Kolkata to empower women to learn and explore new technologies. She has always had a passion for inventing things, creating something new, and inventing new looks for old things. When not in front of her terminal, she is an explorer, a foodie, a doodler, and a dreamer. Biswas is always eager to share her knowledge and ideas with others. She is following that passion by sharing her experiences with her community so that others can learn and give shape to new ideas in innovative ways. This passion is also what inspired her to become an author and write this book.

About the Technical Reviewer

Abhishek Nandy has a Bachelor of Technology degree in IT. He considers himself a constant learner and is a Microsoft MVP for the Windows Platform, an Intel Black Belt Software Developer, as well as an Intel Software Innovator. He has a keen interest in AI, IoT, and game development. Currently, he is an application architect in an IT firm as well as a consultant in AI and IoT who works on projects related to AI, ML, and deep learning. He also is an AI trainer and runs the technical part of the Intel AI Student Developer program. He was involved in the first Make in India initiative, where he was among top 50 innovators and was trained in IIMA.

Acknowledgments

For me, writing this book was highly inspiring and challenging because of ongoing changes to the technology stacks of the various bot frameworks, so I would like to express special thanks and gratitude to all my teachers and mentors. It was a fascinating journey to complete the book; thanks to all my friends and family for the support.

CHAPTER 1

AI and Bot Basics

This chapter covers the basics of bots, how they work, and their interactions with users. We will also touch on what exactly artificial intelligence (AI) is before moving on to discuss further details of chatbots and why they are necessary.

Additionally, this chapter explains the connections between AI and its subsets of machine learning and deep learning. Finally, we will study the structure of AI and bots and cover available bot frameworks.

Artificial Intelligence

Artificial intelligence, or *AI*, is a buzzword nowadays. It is a branch of computer science gaining immense popularity. The goal of AI is to create computer systems that are independent and function intelligently.

AI is easiest to understand when we describe it in terms of characteristics or capabilities of human beings. Let's explore how AI works with human beings or models human behavior.

Human beings connect with other human beings through language. In the field of AI, we call that language exchange *speech recognition*. Because speech recognition is more statistically based, we call this type of learning *statistical learning*.

Humans can read and write text. In AI, this communication process is called *natural language processing (NLP)*.

CHAPTER 1 AI AND BOT BASICS

Humans also can see through their eyes and process what they see. This kind of perception, when converted into the medium of AI, is known as *computer vision*. Computer vision falls under a category of computer science that deals with *symbolic learning*.

When humans look at the world around them, they create pictures through their eyes. In AI terminology, this is known as *image processing*.

Humans can recognize their environment and move around it freely and in any manner. When we replicate these kinds of actions in AI, we enable *robotics*.

Humans have the ability to group patterns of like objects. In terms of AI, we call this *pattern recognition*. Machines are better at pattern recognition than humans because machines can use more data. Machines interpret the data as follows:

1. The dataset is fed into the machine.
2. The machine processes the dataset.
3. The machine splits the dataset into training and testing data.
4. The machine process the training dataset to create a model by using machine-learning algorithms.
5. The machine compares the model with the testing data.
6. The machine joins the training and testing data, evaluates, and finally visualizes the data.

Figure 1-1 illustrates this process.

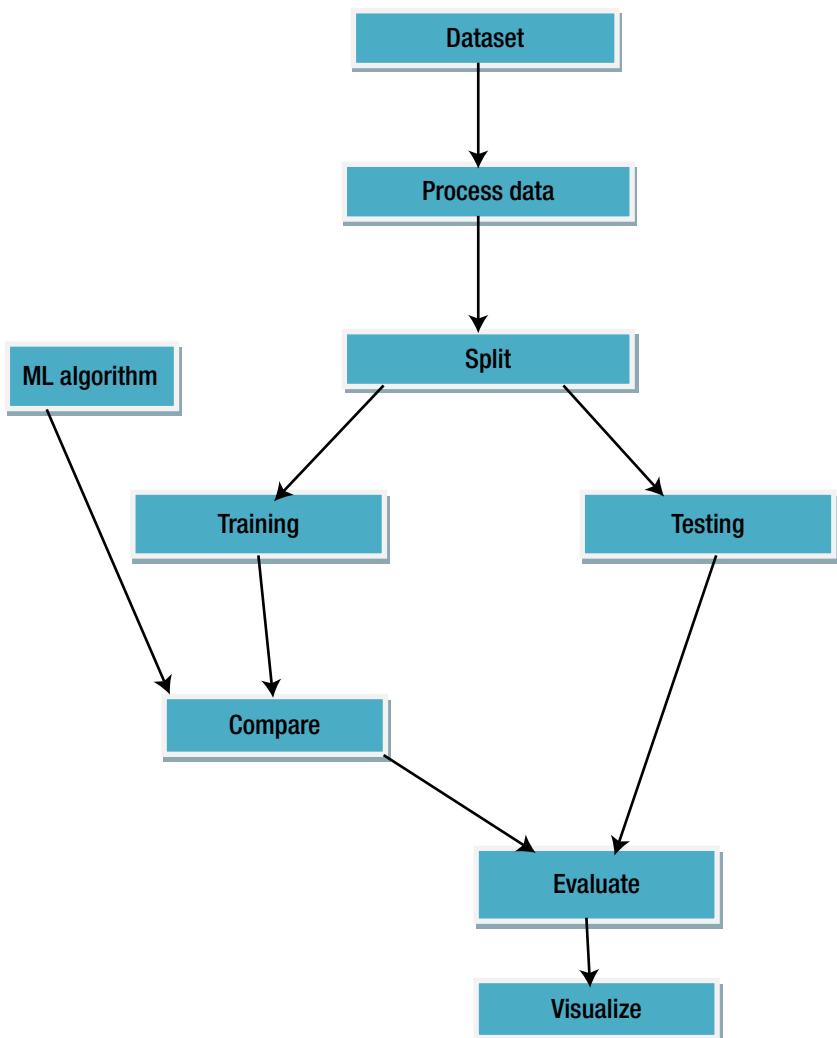


Figure 1-1. Machine-learning flow

CHAPTER 1 AI AND BOT BASICS

Machines can deal with excessive amounts of data in datasets. As the machines try to interpret and learn from the data, without being programmed to do so, we call that process *machine learning (ML)*.

In the human brain, a network of neurons works on making decisions by actively working together. When the same situation is applied artificially, artificial neurons (These neurons are the basis of neural network) try to work similarly to biological neurons, and this behavior is called *cognitive computing*. When communication happens within neurons and they convey information and create a network on its own, we call it *neural networks*. When the network becomes too complex to understand, with complex datasets and many hidden layers involved, machines are performing *deep learning (DL)*.

When machines scan an image from left to right or from top to bottom, those machines form a *convolutional neural network*.

Humans generally remember what they did yesterday or, for example, what we had for dinner yesterday. Machines that can understand the same sequential concepts form *recurrent neural networks*.

Figure 1-2 shows that AI works in two ways. One is a symbolic-based approach, and the other is a data-based approach. Symbolic-based learning uses image recognition; generally, this technique is used for robotics. When we are dealing with huge datasets, we use the data-based approach, also called the machine-learning approach.

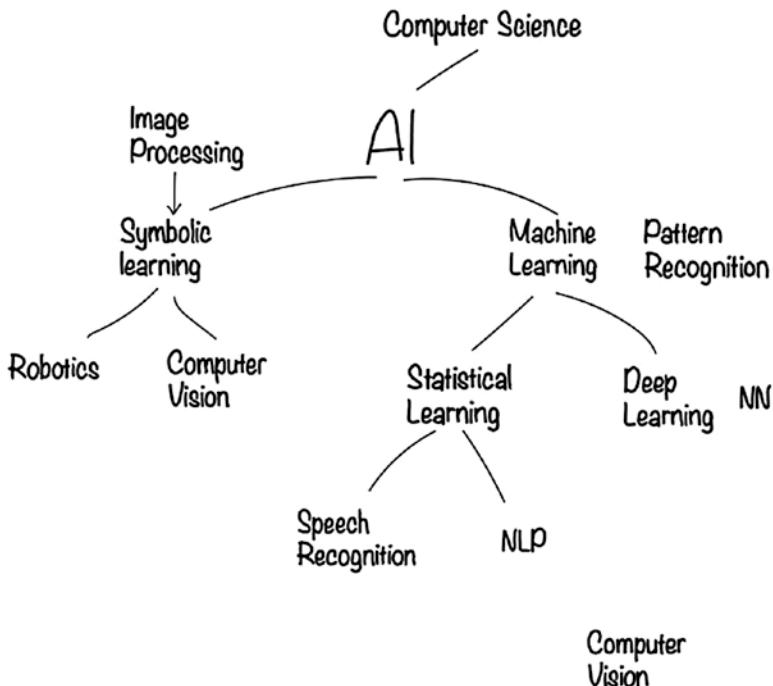


Figure 1-2. Two main approaches in the way AI works

When we use the data-based approach, generally we are dealing with a lot of data that we need to learn from. Subsequently, machines will be able to predict better from early stages as they are now lots of data to work using this data-based approach.

Let's look at an example of how machines work with data. We have sales and advertisement data, and use linear regression to identify the relationship between them. Linear Regression deals with numerical values for a solution. We are trying get the Linear regression curve that fits the straight line equation that is $y = mx + C$, where m is the slope of the gradient C is the constant x is the dependent variable and y is the independent variable. First, let's look at the data points in Figure 1-3.

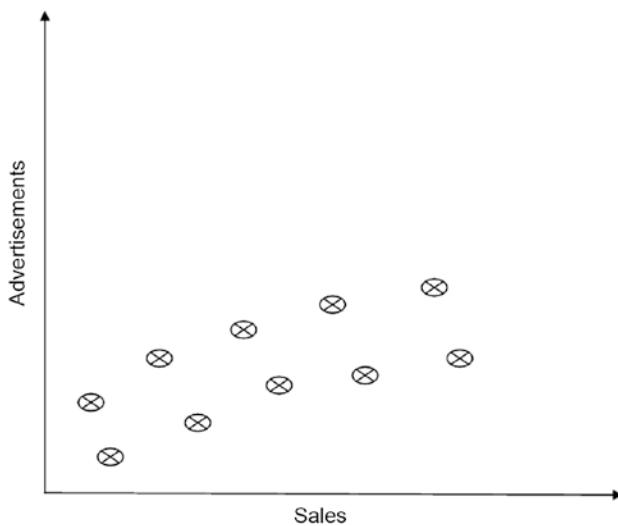


Figure 1-3. Data points indicating sales vs. advertisement data

Figure 1-4 shows some patterns, and now we can apply machine learning (ML) to it. After the machine learns the patterns, it can make predictions based on what it has learned.

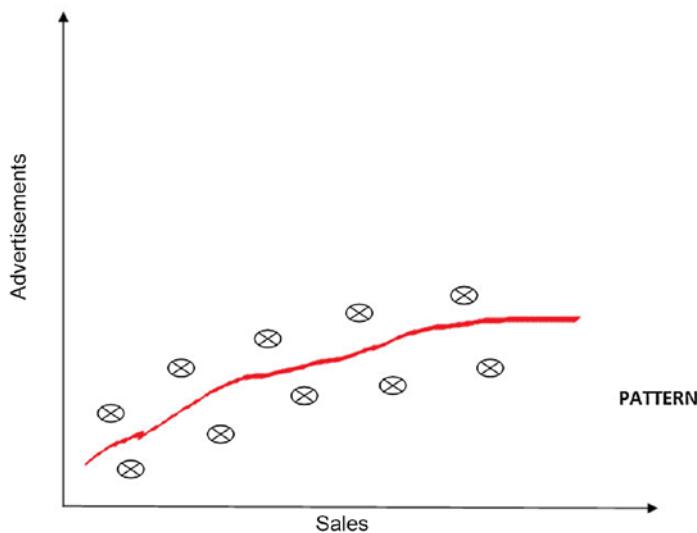


Figure 1-4. The curve shows a pattern. The curve generated from sales increase pattern with respect to advertisements.

Whereas one, two, or three dimensions (The dimensional data are the data generated from information gathered) are easy for humans to learn, machines can learn in many dimensions. The high dimensional data has large amount of information example would be that for a famous personality we check for where they were born, how they came to limelight, their career all have some data but some valuable insights are required to get the best information so machine learning on high dimensions help the data readable and faster to access.

Figure 1-5 shows how data is available in a high-dimensional space and the magnitude of the data associated with it. Machines learn from this high-dimensional data space fast and efficiently.

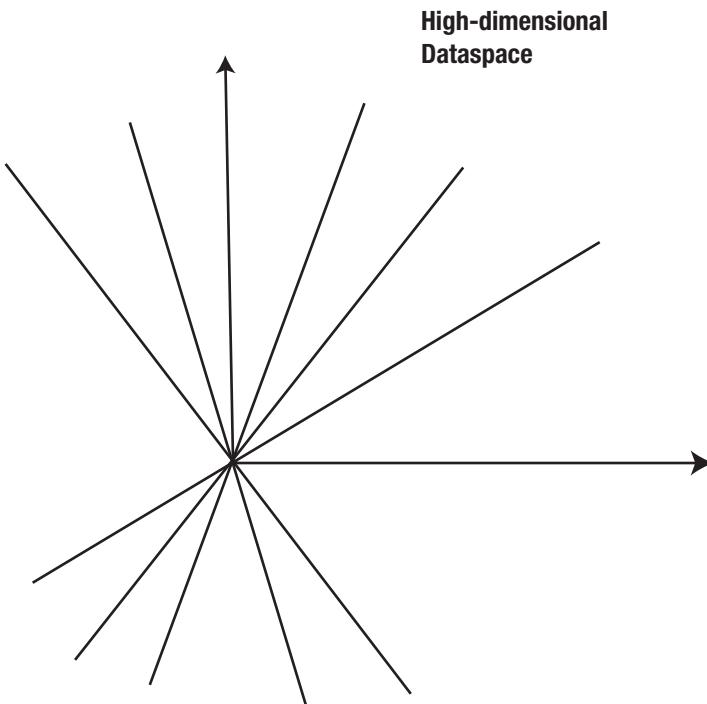


Figure 1-5. High-dimensional space

CHAPTER 1 AI AND BOT BASICS

A machine can look at large amount of data within the dataset, which we call high-dimensional data, and determine patterns. After the machine learns these patterns, this valuable information gives useful insights leading to a lot of research, and again leading to faster-evolving AI. Machine uses the patterns to do two things:

- Classification
- Prediction

Classification

When we work with different kinds of datasets, we generally divide the datasets into observation types. We utilize the training data so that we can apply different machine-learning algorithms to it. When using Classification if we are trying to classify mail coming into our account as legitimate or spam we associate two levels either it will get inside the mail box or will be inside spam. We associate labels for it which determines the spam filtering process through classification.

Prediction

When we are working toward future readiness of a particular dataset, we generally run predictions by using the available dataset. The model we create after applying machine-learning techniques is a benchmark so that we can predict future outcomes based on this model. Machines generally learn in two ways:

- Supervised learning
- Unsupervised learning

Supervised Learning

Supervised learning is generally based on available ground truth (we are sharing every resource for the data points such as inputs, outputs, and any other available information). Therefore, we are providing the machine or program with input and output. The machine goes through the available information and finally predicts. Regression is an example of supervised learning. Let us suppose we want to predict a price of a property in an area the model we created applying regression and the slope formula that is $y = mx + C$ we can get a result that is far off from the actual property price the difference is the error in order to get the model more accurate and close to the result we will tweak the slope values to get the proper value.

Unsupervised Learning

Depending on the scenario, the dataset we're working with may not have any prior information available. We might be dealing with datasets that are not labeled or classified. In this case, we use *unsupervised learning*.

Clustering is an example of unsupervised learning.

There is a different way to deal with environments, and we call this branch of machine learning *reinforcement learning*. In reinforcement learning, we give a machine a goal and ask it to learn it from trial-and-error.

Machines learn in the three ways, shown in Figure 1-6: supervised, unsupervised, and reinforcement learning.

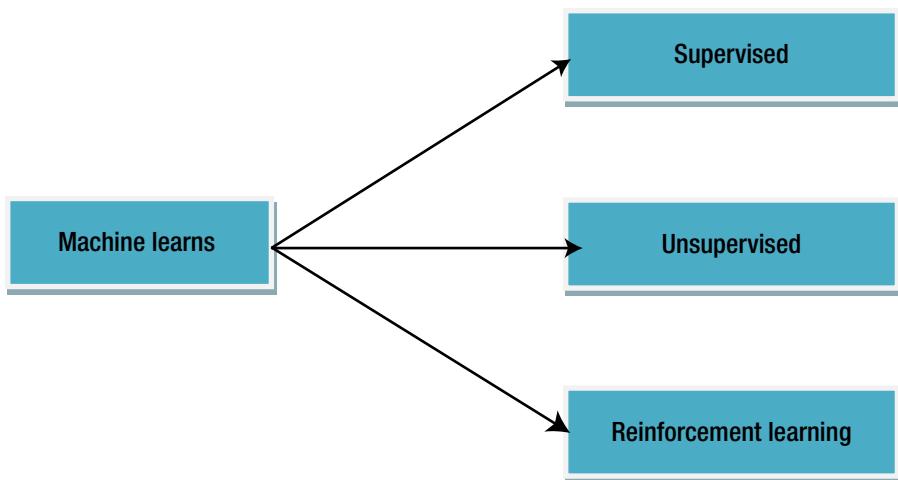


Figure 1-6. Machines learn in three ways

Interconnection Between AI, ML, and DL

AI, ML, and DL, depicted in Figure 1-7, are separate terms. However, they do have interconnections, described next.



Figure 1-7. AI, ML, and DL

AI refers to a broad category of computing in which a system is capable enough to learn directly from data. This is possible because of a lot of interactions, analysis, structuring, and visual representations of data using data analytics. This is applied by sets of rules defined by human intervention or increasingly by the subset of AI that is also known as machine learning.

With the advancement of ML, many things underwent changes, including processing the data and the way information is used. Data needs some form of automation to evaluated faster without the need for human intervention. ML deals with large data sets too.

When we use a lot of data for our work, we reach a computational tipping point, which is when lots of data is gathered together and size is really huge to work with as depicted in Figure 1-8. This can result in new implementation methods in data analytics.



Figure 1-8. Reaching the computational tipping point

CHAPTER 1 AI AND BOT BASICS

A subset of ML known as deep learning (DL) was one result of reaching this tipping point. Figure 1-9 illustrates the relationship between AI, ML, and DL.

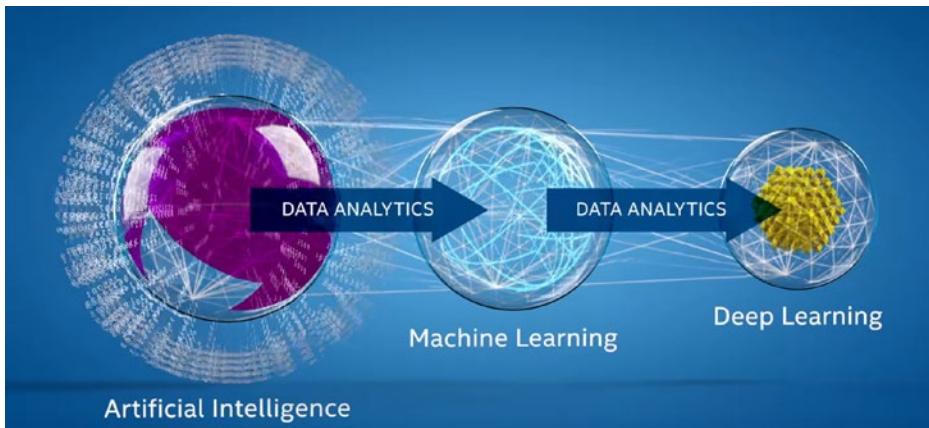


Figure 1-9. The evolution from artificial intelligence to deep learning

A neural network, illustrated in Figure 1-10, is another concept we will discuss in this book that has become popular. When analysing an image the first thing computer does is to break the image in RGB scale now we have different levels of abstractions for the layers. We need to access the layer in different way. So a neural network is generated and as we see if the level of abstraction goes beyond 3 it directly falls under Deep Learning and neural network is a part of it.

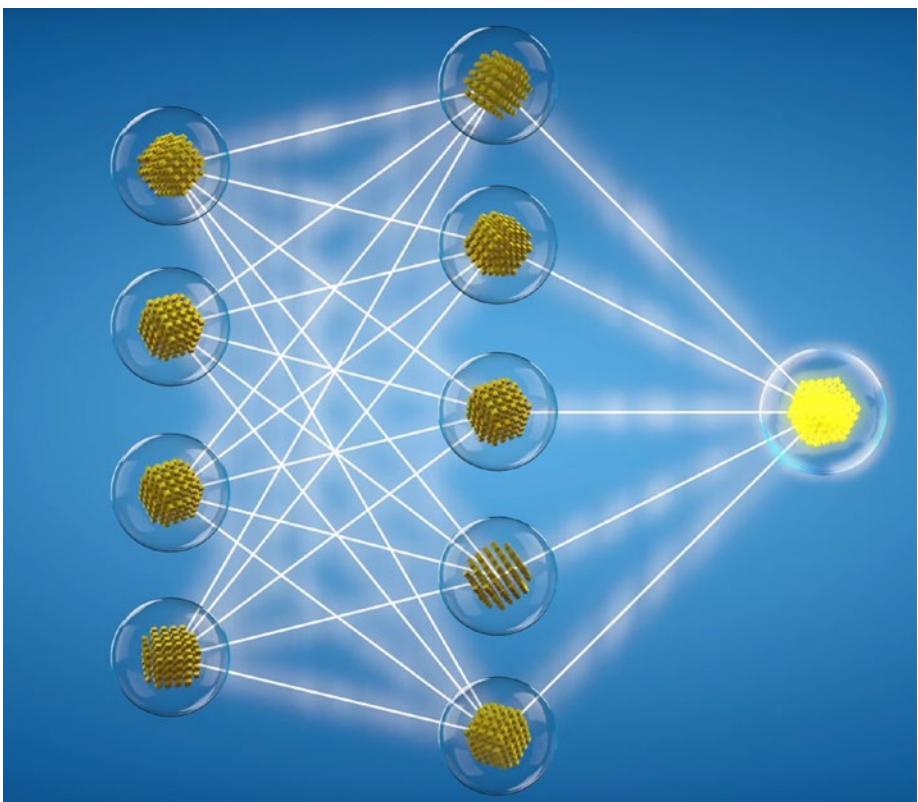


Figure 1-10. Neural network

The neural network made sense of different connected neuron networks and produced outputs that brought groundbreaking results. This resulted in the progressive growth of AI in a broader sense for capabilities such as image, speech, and natural language recognition. This produced significant and positive impacts in our daily lives. The evolution of deep learning signifies that AI is not static and will grow and adapt accordingly as different techniques arise. One good example would be that we travel to a different place we don't know the language the capabilities of AI can take an image or the information that is there using the webcam and can convert it into easy English language.

Chatbots

Chatbots they have come a long way. From being static to more on interaction.

In today's world, we have to hide service layers within plain conversational techniques. We don't want to show what's happening underneath so we wrap the service layer and hide the logic so that we are not able to show the entire process of communication as many people want their technique to be secretive.

For the best chatbots, we need to underline one thing. If a human being analyzes both a bot and a human conversation and is unable to tell the difference between the two, we say it passes the *Turing test*. However, until now, no bot has been able to achieve this feat.

Traditionally speaking, chatbots have used what we call a *retrieval-based model*. In this model, a programmer's code provides predefined responses, and chatbots learn in a heuristic way to pick the appropriate response.

The first chatbots ever created used rule-based expression matching. Now the approach has moved to using ML as a classifier for better responses. We can take as an example Facebook's API, for which we can hard-code responses and then classify words with intent. Then if we ask or phrase a query as, "What day is today?" or "Today is what day?" Chatbot understands both.

Generative Chatbot Model

A *generative chatbot model* relies on predefined responses, and whatever we do, we should write them from scratch. When we are working with chatbots, we first have to see whether we are working on a closed domain or an open domain.

Open Domain

In an *open domain*, a conversation can go anywhere. There are infinite things to talk about.

Closed Domain

In a *closed domain*, a conversation focuses on a single subject or topic. The chatbots evolve based on the kind of conversation we want: long or short. Short conversations are easier to use.

How Do Chatbots Work?

The best way to learn how chatbots work is to first understand the brain of bot. We call this a *digital brain*, and it consists of three main parts:

- Knowledge source: Where we need to find out which informations needs to provided to the bot so the conversation starts and for Q and A
- Stock phrases: Its something where we can handle general phrases of conversation which is used more often
- Conversational memory: When we are doing a conversation we have to remember the flow what has happened so conversational memory is required

When we start communicating with a bot, we may send a message (for example, “hello”), and the bot starts working—or more accurately, analyzing the message. The bot’s activity is known as *parsing*. Next the bot will look for keywords in order to reply to the message.

CHAPTER 1 AI AND BOT BASICS

Then the brain of the bot will use its three main parts to analyze the message and then construct its reply. Figure 1-11 shows the structure of the bot's digital brain.

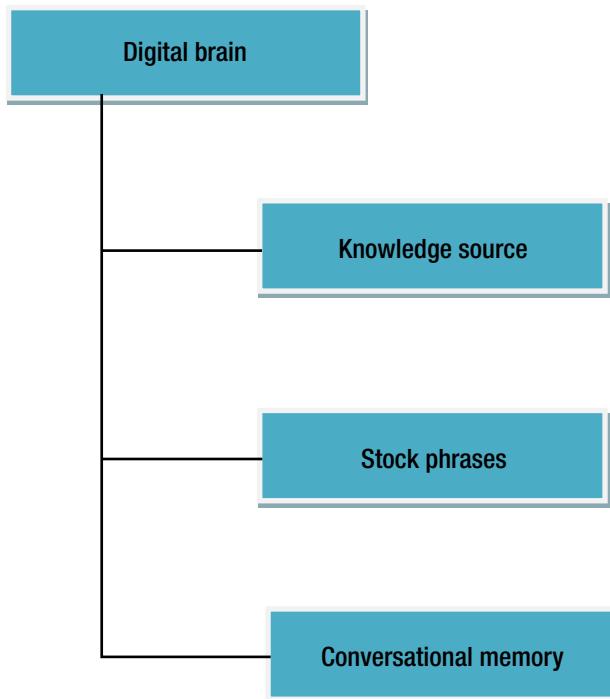


Figure 1-11. The bot's digital brain

Figure 1-12 shows how the response is generated from the digital brain.

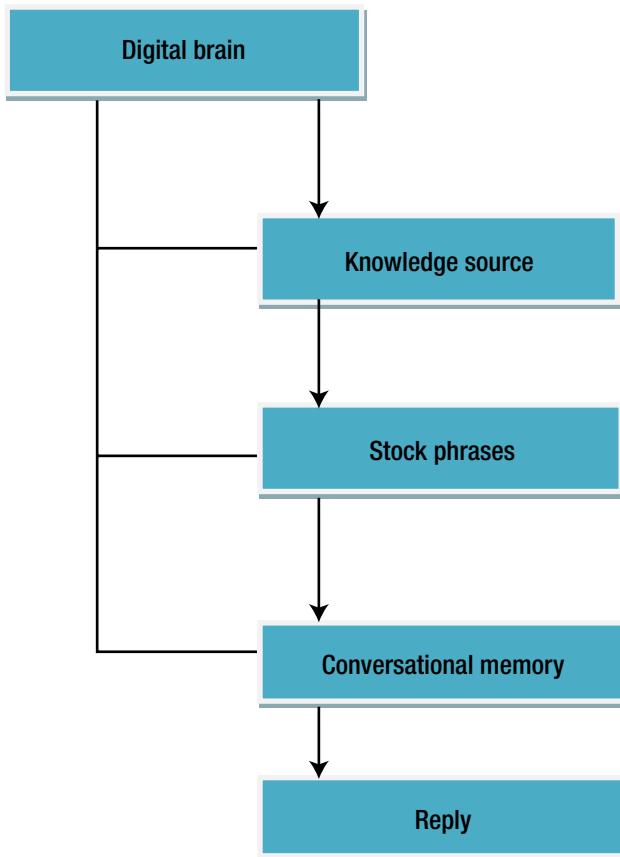


Figure 1-12. The response being generated by the bot

Rise of the Chatbots—Conversational Commerce

Conversational commerce uses chats, messaging, or some kind of natural language as its interface. *Conversational* means it uses some kind of voice or text medium to transfer data and understand how people communicate. What we are doing with chats and messaging is using this particular medium to easily interact with each other. To make our jobs easier to

CHAPTER 1 AI AND BOT BASICS

answer simple queries for our business we feed the bot with common questions and answers so that we can save time before they go to the customer care if required.

The advent of conversational commerce allows users to talk to companies and have companies also talk back in an easy manner. This can happen in three ways:

- Bidirectional: That is the communication flow is faster and seamless within the bot
- Asynchronous: Allowing the messaging to be controlled at timely manner and not within specified time intervals
- In real time: The communication response is to make it realtime

The Role of Chatbots

In conversational commerce, chatbots are the computer programs used to simulate a conversation with a human. They use a text-based approach.

Chatbots work on the following:

- Basic operations
- Basic things answering the general queries with ease
- Basic questions

The Role of Humans

Conversational commerce is powered by humans. The UI developed is powered by humans. The structure (also called an *interface*) is designed by humans and is where the communication occurs.

When a conversations increases in complexity, it is handed over for human operators to resolve.

Growth of Chat Apps

The chatbot's popularity comes from the growing use of messaging applications such as Facebook Messenger, WhatsApp, and Telegram. These chat apps are now trending and surpassing social applications. The Facebook Messenger platform is a good example of an application that has surpassed all social apps because of the following:

- A unified and a free flowing UI
- Great experience
- Very dynamic as many people use them

Chatbot allows us to work on business criteria by having direct conversations with a company.

The following sections provide some chatbot examples.

Poncho

Poncho is a weather activity bot whose logo is cat. The cat handles all the conversations for you and shares the weather in a simple and collaborative way.

CNN Bot

This is a news bot providing news options. We can ask specific questions to the bot with keywords such as "Space." It will then give us information about space news. The CNN bot also learns from our daily activity and predicts accordingly, with news options available based on the type of searches we do.

Spider Bots

Another common bot type is the web spider bot. These bots gather information from the web in order to discover details that can be harvested and stored. We can use these bots to capture e-mail

information from a page and move to the next page. This process is known as *scraping*. Search engines like Google use spiders to increase the speed of their searches.

Twitter Bots

There are bots present on Twitter that analyze tweets. They work on finding specific topics, intending to do retweets based on those topics.

Botnets

When a coordinated unit of bots works together in tandem, we call that a *botnet*. Botnets can be requested to go to the same web page at once and make web sites crash. This is also known as a distributed denial-of-service (DDoS) attack when botnets are distributed.

Reinforcement Learning Bot

Reinforcement learning bots use a reinforcement learning approach. Reinforcement learning is based on trial-and-error. Reinforcement learning is also based on environments (It can be a 3d space, A game based scenario etc) and worlds, hence why we should not work on modeling the world. Instead we should work on modeling the mind that means the best possible step that needs to be taken should be given to the machine such that it works within its limits.

DeepMind (Its a Google funded startup that works towards finding reinforcement learning solutions to practice) worked toward finding a solution that is now popular and works toward formulation of artificial general intelligence.

DeepMind's algorithm works toward solving solutions, such as playing Atari games with one unified approach, is known as Deep Q-Learning. It takes two inputs: the raw pixels of the game and the game score. Based on those two, it has to reach its objective to maximize the score. First it

uses deep convolutional neural networks to interpret the pixels. Deep convolutional neural networks extract features and the bot learns as hidden layers get incredibly abstract.

The Structure of a Bot

This section describes what a bot structure looks like and how conversations flows.

Figure 1-13 describes how the structure of the bot's programming is formed.

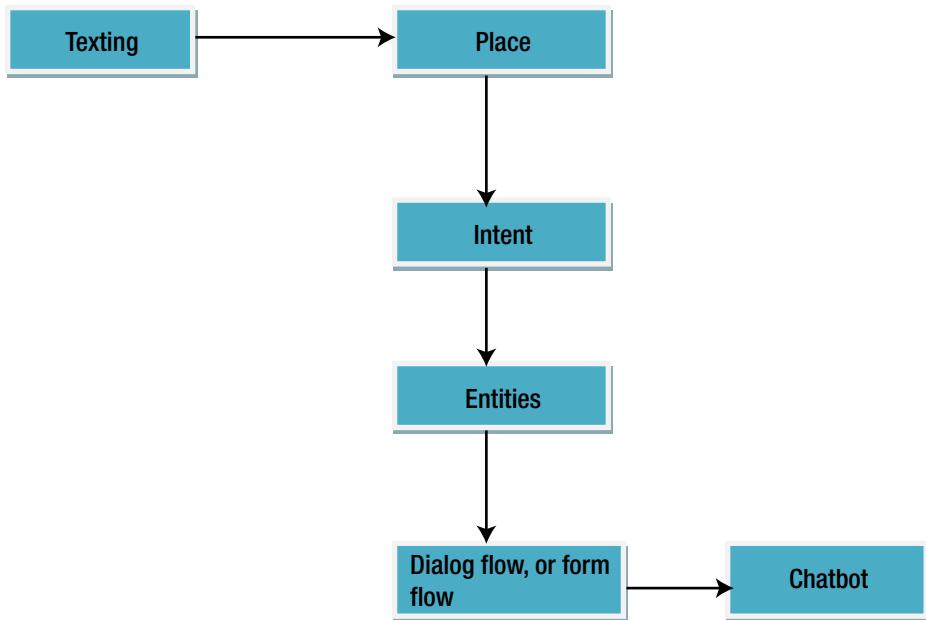


Figure 1-13. The structure of the bot

We first start with texting with some questions, then we move to the place/template where we start developing or coding the logic of the bot here the response flow is generated. Next, we describe the intent that forms the basis of the bot's communication, in order to provide a textual response and making the chatbot interactive.

We use entities to break the piece of information that is message flow to better understand what is happening for our usage to communicate. Finally, we create a workflow for the bot to get organized. This flow which allows the entire process to work efficiently it is either called a *dialog flow* or *form flow*.

Let's talk about the structure of bot in detail now. We generally want to see an efficient text response platform. For bots to work in a specific manner, we need a specific software development kit or interface where we start preparing the logic. The logic can be embedded within an IDE or it can be completely cloud based, as in IBM's Watson.

We need to have a template, or place, where we start our development. Of course, it's easier to start with a template because the basis is already formed.

Now we need to find specific ways to deal with a type of communication. For this, we use intent. Intent is generally the basis for the training of our bot. For example, if we have an intent for the bot to offer greetings, the ways for the bot to start with that intent might be "Hi," "Hello," "Hey there," and so forth. We have to provide specific text for understanding the logic of the intent.

Now we move to something called *entities*. When a user has a text conversation with a bot, the bot uses intent to choose the response or to make intelligent decisions to get the conversation flow going.

Dialog Flow or Form Flow

This dialog flow, or form flow, is the process by which we structure the intents and entities to work together for a specific goal for the bot. We declare all the functionalities that will be performed by the bot. Now we are done with the logic, and this is how the bot is organized.

Bot Frameworks

The technology giants have all come up with different bot frameworks to get started with bot development. Here are the major bot frameworks:

- Microsoft has the Bot Framework.
- Google has Wit.ai and Dialogflow.
- IBM has Watson.
- Amazon Web Services (AWS) has its own bot framework powered by AWS Lambda.
- We can develop a chatbot by using TensorFlow.
- We can also use FlockOS to develop chatbots.

Chatbots are gaining in popularity. That's why every big company is trying to create a framework where bot development can occur.

As we gather information with the flow of bots, we use a programming language to structure the bot and redirect it accordingly.

We are seeing the rise of chatbots as the evolving bot frameworks are becoming readily available to use.

Conclusion

This chapter presented the basics of bot frameworks and described the process of developing bots. We also covered artificial intelligence with descriptions of machine learning as well as deep learning. We have shown how conversational commerce works and discussed the digital brain in terms of chatbots. Finally, we touched on the structure of bots and ended with the frameworks available for bot development. In the next chapter, we can begin bot development.

CHAPTER 2

Microsoft Bot Framework

This chapter covers the Microsoft Bot Framework. We will start with a brief introduction to the Microsoft Bot Framework. After that, we will move to:

- The template for Bot Framework with Visual Studio
- Begin working with the Bot Framework with Visual Studio
- Talk about different Bot Framework states such as Intent, Entities for Bots and then describe dialogs and form flow
- Language Understanding and Intelligent Service (LUIS).
- The new LUIS web site, the changes, and explore and discuss its features
- Developing an end-to-end bot using different bot properties
- Publishing the bot
- Using Dynamics CRM to use it in Bot Framework
- Studying the structure of AI and bots.

When we are working toward building connections that is the interactive nature of chatbots and a well-managed conversation flow, we aim to use frameworks that make the work easier for us. The Microsoft Bot Framework is an essential collection of tools and supportive software development kits (SDKs) for making the work easier when creating and deploying well-managed conversation bots.

Starting with the Prerequisites

To use the Microsoft Bot Framework for development, we first need to satisfy some prerequisites. This section describes some of the things that are necessary in order to get started:

- *IDE: Visual Studio*
- *Operating system: Windows 10*
- *Bot development framework: Bot Builder*
- *Emulator for testing: Bot Framework Emulator*

Visual Studio

First, for the framework to work, we need an integrated development environment (IDE), a place where we can code the entire thing. Visual Studio is an essential choice for developing for Microsoft Bot Framework. It combines the use of Microsoft technology smoothly and it is aligned well with the Microsoft technology stack. In this book, we will work with Visual Studio 2015, but we could use Visual Studio 2017. You can download Visual Studio from <https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio>.

Windows 10

We need to have an operating system in place for developing and hosting the IDE. The operating system of choice is Windows 10, and you can download it from www.microsoft.com/en-in/software-download/windows10. Figure 2-1 shows the Windows 10 download page.

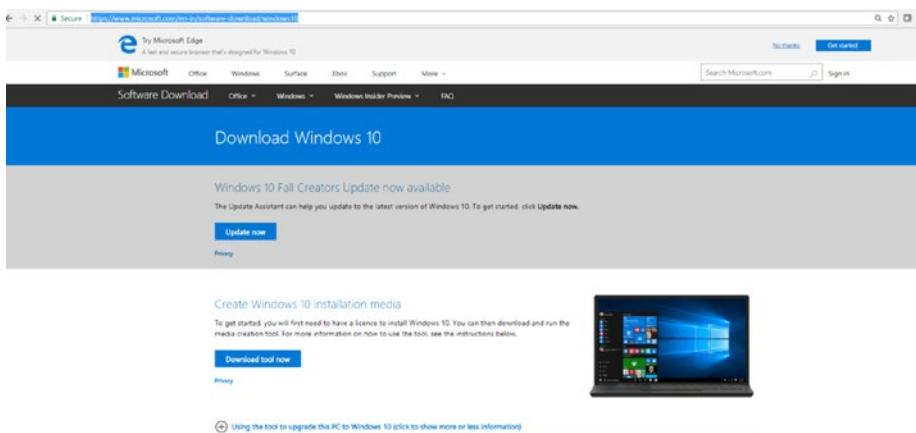


Figure 2-1. The page to download Windows 10

Bot Builder

We also need to have a blueprint for our Bot Framework development with Visual Studio. We will have to download a template, and, yes, we will have to choose a programming language in which to develop our bot. The programming language of choice is C#. We will be dealing with the same programming language most of the time during our bot development.

You can download the template from <https://docs.microsoft.com/en-us/bot-framework/bot-builder-overview-getstarted>. Scroll down this page to find the template to download, as shown in Figure 2-2.

CHAPTER 2 MICROSOFT BOT FRAMEWORK

The screenshot shows the Microsoft Bot Framework Overview page. On the left, there's a sidebar with a 'Filter' input field and a navigation menu:

- Overview
- About Bot Framework
- How it works
- Add Intelligence to bots
- Start building bots** (highlighted in blue)
- Quickstarts
- Samples
- How-To guides
- Reference
- Resources

The main content area has several sections:

- Important**: The Bot Builder SDK for .NET requires .NET Framework 4.6 or newer. If you are adding the SDK to an existing project targeting a lower version of the .NET Framework, you will need to update your project to target .NET Framework 4.6 first.
- To install the SDK in an existing Visual Studio project, complete the following steps:**
 - In **Solution Explorer**, right-click the project name and select **Manage NuGet Packages...**.
 - On the **Browse** tab, type "Microsoft.Bot.Builder" into the search box.
 - Select **Microsoft.Bot.Builder** in the list of results, click **Install**, and accept the changes.
- You can also download the Bot Builder SDK for .NET source code from [GitHub](#).
- Visual Studio project templates**: Download Visual Studio project templates to accelerate bot development.
 - [Bot template for Visual Studio for developing bots with C#](#)
 - [Cortana skill template for Visual Studio for developing Cortana skills with C#](#)
- Tip**: A green tip box with the text: "The Bot Builder SDK for .NET leverages C# to provide a familiar way for .NET developers to write bots. It is a powerful framework for constructing bots that can handle both free-form interactions and more guided conversations where the user selects from possible values."

Figure 2-2. The link to download the Visual Studio template

Bot Framework Emulator

When we are done coding, we'll need to test the bot locally before hosting it. We'll use a bot emulator to test all the functionalities and the entire flow of the bot. You can download the Bot Framework Emulator from <https://docs.microsoft.com/en-us/bot-framework/debug-bots-emulator>.

Figure 2-3 shows the emulator's download page.

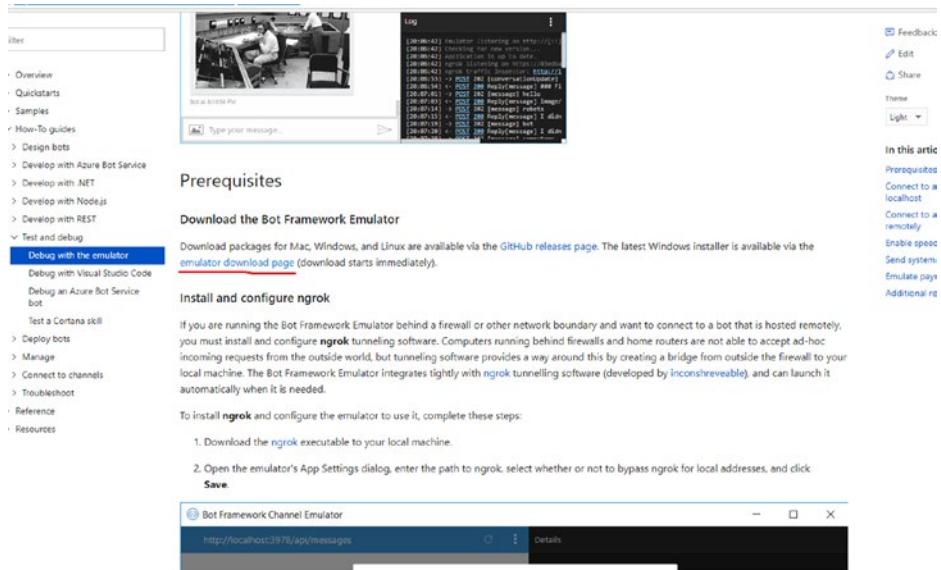


Figure 2-3. The emulator download page

The main page hosts all the bot emulators. From that list, we can download and run the emulator's EXE file. The executable will create a shortcut on the desktop. You can then double-click that shortcut to get the emulator running. The direct link to get the EXE is <https://github.com/Microsoft/BotFramework-Emulator/releases/tag/v3.5.34>.

Figure 2-4 shows the appropriate EXE to download.

CHAPTER 2 MICROSOFT BOT FRAMEWORK

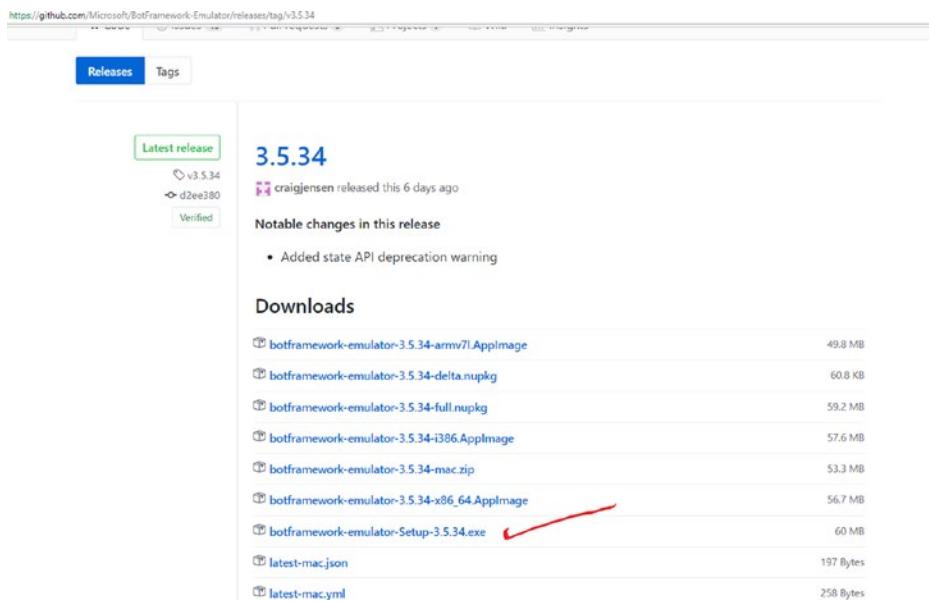


Figure 2-4. The EXE file for the emulator

Creating a Simple Bot Framework App

In this section, you will create a simple bot framework app by using the VS template in the C# language. Then you'll test the app first by running it locally and then by using the Bot Framework Emulator.

Using a Template to Create the Project

To get started, you first have to start the Visual Studio IDE for development. When you open Visual Studio, the IDE screen looks like Figure 2-5.

CHAPTER 2 MICROSOFT BOT FRAMEWORK

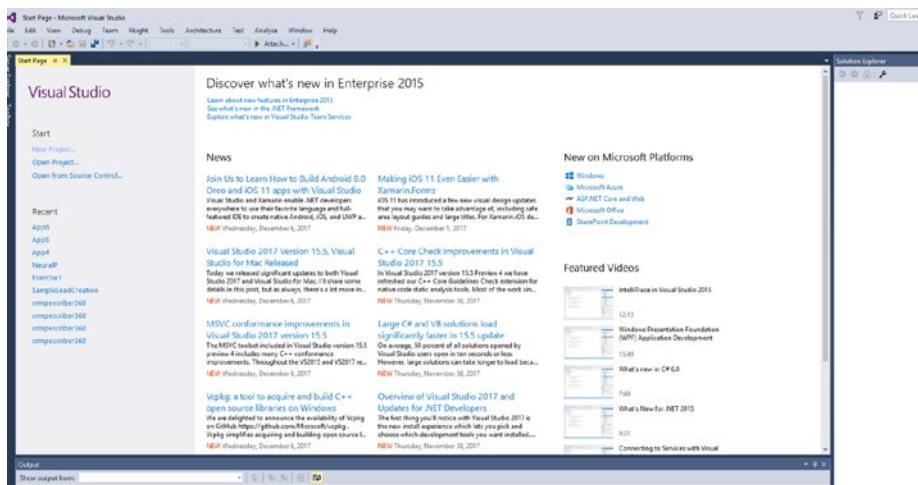


Figure 2-5. The Visual Studio IDE screen when opened for the first time

You need to open a new template to create a project. Click the File tab at the top left and then click New ➤ Project, as shown in Figure 2-6.

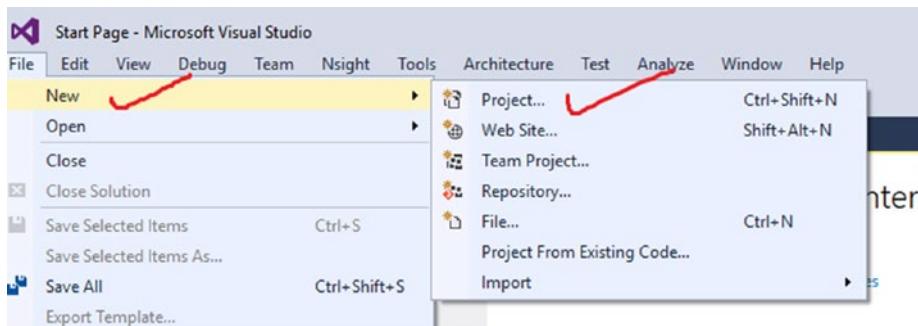


Figure 2-6. Creating a new project

CHAPTER 2 MICROSOFT BOT FRAMEWORK

In the New Project window that opens, you will see that the downloaded Visual Studio template for C# is available for creating a bot. Within the templates for Visual C#, you can find the Bot Application template (see Figure 2-7).

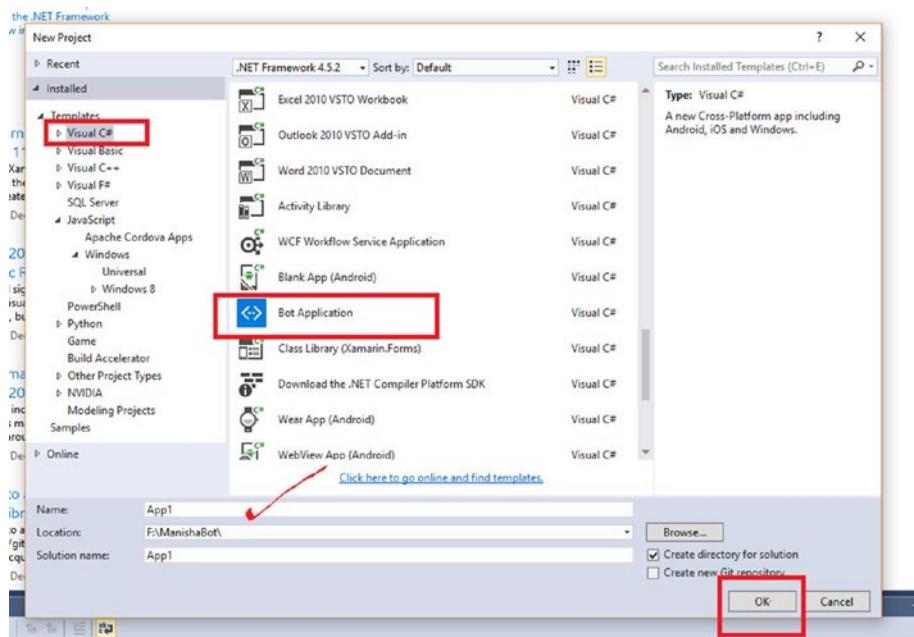


Figure 2-7. The Bot Application template

You are now ready to create an app from the template. In this example, you will use the template without changing any parameters so you can see what the Bot Application template does.

First, you need to name the app. In this example, you will name it App1. Then click OK to continue. In the back end, the template will formulate the bot framework essentials and create the necessary files to get your first bot application working. Let's run and observe the code created.

Working with the Code

As the project file is created, you will focus on the Solution Explorer option, where you can see the hierarchy of the available files. Figure 2-8 shows the files created at the start of the project.

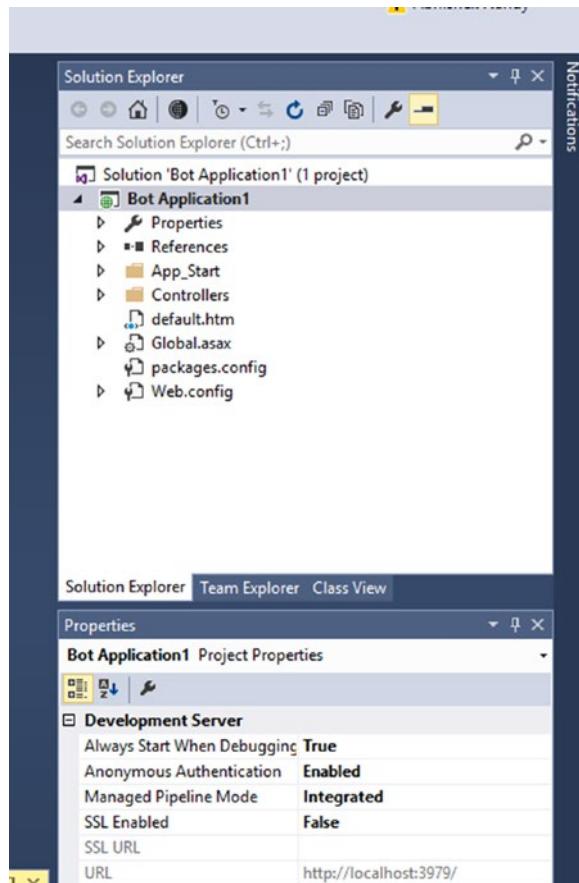


Figure 2-8. The available files for the new project

CHAPTER 2 MICROSOFT BOT FRAMEWORK

In terms of getting everything right, the most important file is the web.config file. This is used for hosting the app on the Web with Azure as well as having the entire key details filled in correctly. It's an XML file in which you have to provide the details for the application ID as well as other details from the Bot Framework web site, which we will discuss later.

Figure 2-9 shows the how the XML file looks when accessed in the IDE.

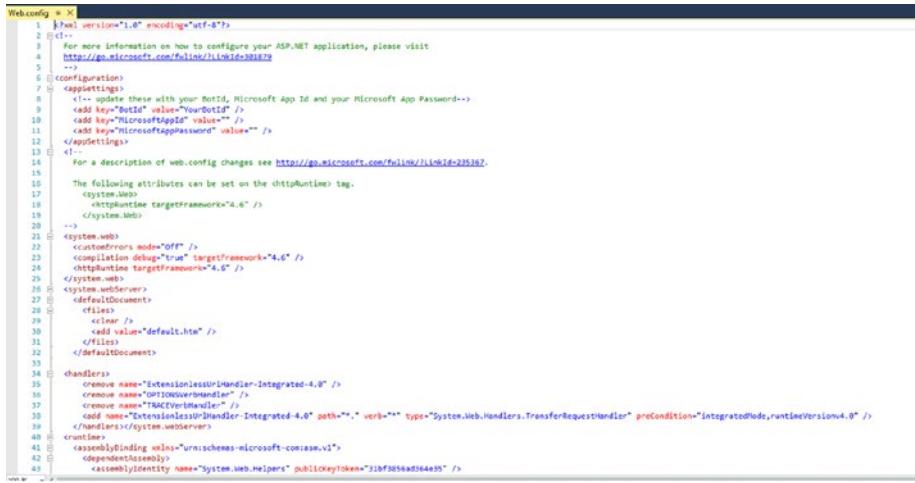


Figure 2-9. The web.config file in the IDE

The entire web.config XML file has the structure shown in Listing 2-1.

Listing 2-1. web.config file as XML Format

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET
  application, please visit
  http://go.microsoft.com/fwlink/?LinkId=301879
-->
<configuration>
```

```
<appSettings>
    <!-- update these with your BotId, Microsoft App Id and
        your Microsoft App Password-->
    <add key="BotId" value="YourBotId" />
    <add key="MicrosoftAppId" value="" />
    <add key="MicrosoftAppPassword" value="" />
</appSettings>
<!--
    For a description of web.config changes see http://
    go.microsoft.com/fwlink/?LinkId=235367.

```

The following attributes can be set on the `<httpRuntime>` tag.

```
<system.Web>
    <httpRuntime targetFramework="4.6" />
</system.Web>
-->
<system.web>
    <customErrors mode="Off" />
    <compilation debug="true" targetFramework="4.6" />
    <httpRuntime targetFramework="4.6" />
</system.web>
<system.webServer>
    <defaultDocument>
        <files>
            <clear />
            <add value="default.htm" />
        </files>
    </defaultDocument>

    <handlers>
        <remove name="ExtensionlessUrlHandler-Integrated-4.0" />
        <remove name="OPTIONSVerbHandler" />
```

CHAPTER 2 MICROSOFT BOT FRAMEWORK

```
<remove name="TRACEVerbHandler" />
<add name="ExtensionlessUrlHandler-Integrated-4.0"
path="*.*" verb="*" type="System.Web.Handlers.
TransferRequestHandler" preCondition="integratedMode,
runtimeVersionv4.0" />
</handlers></system.webServer>
<runtime>
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
<dependentAssembly>
<assemblyIdentity name="System.Web.Helpers" publicKey
Token="31bf3856ad364e35" />
<bindingRedirect oldVersion="1.0.0.0-3.0.0.0"
newVersion="3.0.0.0" />
</dependentAssembly>
<dependentAssembly>
<assemblyIdentity name="System.Web.Mvc" publicKeyToken=
"31bf3856ad364e35" />
<bindingRedirect oldVersion="1.0.0.0-5.2.3.0"
newVersion="5.2.3.0" />
</dependentAssembly>
<dependentAssembly>
<assemblyIdentity name="System.Web.WebPages" publicKey
Token="31bf3856ad364e35" />
<bindingRedirect oldVersion="1.0.0.0-3.0.0.0"
newVersion="3.0.0.0" />
</dependentAssembly>
<dependentAssembly>
<assemblyIdentity name="Newtonsoft.Json" publicKeyToken
="30ad4fe6b2a6aeed" culture="neutral" />
<bindingRedirect oldVersion="0.0.0.0-8.0.0.0"
newVersion="8.0.0.0" />
</dependentAssembly>
```

```

<dependentAssembly>
  <assemblyIdentity name="System.Net.Http.Primitives"
    publicKeyToken="b03f5f7f11d50a3a" culture="neutral"/>
  <bindingRedirect oldVersion="0.0.0.0-4.2.29.0"
    newVersion="4.2.29.0" />
</dependentAssembly>
<dependentAssembly>
  <assemblyIdentity name="System.Net.Http.Formatting"
    publicKeyToken="31bf3856ad364e35" culture="neutral"/>
  <bindingRedirect oldVersion="0.0.0.0-5.2.3.0"
    newVersion="5.2.3.0" />
</dependentAssembly>
</assemblyBinding>
</runtime></configuration>

```

Figure 2-10 shows the most important part of the XML file, where we have to make changes. The Configuration tab is where we have to put in all the information. The appID, BotID and the AppPassword needs to be updated.

```

-->
<configuration>
  <appSettings>
    <!-- update these with your BotId, Microsoft App Id and your Microsoft App Password-->
    <add key="BotId" value="YourBotId" />
    <add key="MicrosoftAppId" value="" />
    <add key="MicrosoftAppPassword" value="" />
  </appSettings>
  ...

```

Figure 2-10. Code for the Configuration tab

Listing 2-2 shows how to get the BotId, MicrosoftAppId, and MicrosoftAppPassword values that need to be put in.

Listing 2-2. Important Code Section Providing Values from the bot Website

```
<appSettings>
    <!-- update these with your BotId, Microsoft App Id and
        your Microsoft App Password-->
    <add key="BotId" value="YourBotId" />
    <add key="MicrosoftAppId" value="" />
    <add key="MicrosoftAppPassword" value="" />
</appSettings>
```

You will get the details of the bot ID, Microsoft app ID, and the password from the Bot Framework web site. You will go through that process later in this chapter, when you publish the app. For now, let's concentrate on the coding logic.

Inside the controllers folder, you will find the `MessageController.cs` file. The logic for the template works like this: The `MessageController.cs` file checks for activity and then scans any message sent to the bot. The file checks the string length and then returns the character size or message length. Listing 2-3 shows the process in C# of checking the length of the characters.

Listing 2-3. The Reply Message Code Block in C#

```
if (activity.Type == ActivityTypes.Message)
{
    ConnectorClient connector = new
        ConnectorClient(new Uri(activity.ServiceUrl));
    // calculate something for us to return
    int length = (activity.Text ?? string.Empty).
        Length;
```

```
// return our reply to the user
Activity reply = activity.CreateReply($"You
sent {activity.Text} which was {length}
characters");
await connector.Conversations.
ReplyToActivityAsync(reply);
}
```

The first line of code uses the Client connector. This connector establishes a channel to communicate via the bot messaging system. The next line checks for the string length of the message. Using the ?? operator provides two options: `activity.Text ?? string.Empty`. If we get a character of a certain length, the logic on the left is implemented. If the message is empty, the logic on the right is implemented.

Listing 2-4 shows the complete `MessageController.cs` file.

Listing 2-4. Entire Message Reply Code Block

```
using System;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using Microsoft.Bot.Connector;
using Newtonsoft.Json;

namespace Bot_Application1
{
    [BotAuthentication]
    public class MessagesController : ApiController
    {
```

CHAPTER 2 MICROSOFT BOT FRAMEWORK

```
/// <summary>
/// POST: api/Messages
/// Receive a message from a user and reply to it
/// </summary>
public async Task<HttpResponseMessage> Post([FromBody]
Activity activity)
{
    if (activity.Type == ActivityTypes.Message)
    {
        ConnectorClient connector = new
        ConnectorClient(new Uri(activity.ServiceUrl));
        // calculate something for us to return
        int length = (activity.Text ?? string.Empty).
Length;

        // return our reply to the user
        Activity reply = activity.CreateReply($"You
sent {activity.Text} which was {length}
characters");
        await connector.Conversations.
        ReplyToActivityAsync(reply);
    }
    else
    {
        HandleSystemMessage(activity);
    }
    var response = Request.
CreateResponse(HttpStatusCode.OK);
    return response;
}
```

```
private Activity HandleSystemMessage(Activity message)
{
    if (message.Type == ActivityTypes.DeleteUserData)
    {
        // Implement user deletion here
        // If we handle user deletion, return a real
        message
    }
    else if (message.Type == ActivityTypes.
ConversationUpdate)
    {
        // Handle conversation state changes, like
        members being added and removed
        // Use Activity.MembersAdded and Activity.
        MembersRemoved and Activity.Action for info
        // Not available in all channels
    }
    else if (message.Type == ActivityTypes.
ContactRelationUpdate)
    {
        // Handle add/remove from contact lists
        // Activity.From + Activity.Action represent
        what happened
    }
    else if (message.Type == ActivityTypes.Typing)
    {
        // Handle knowing tha the user is typing
    }
}
```

```
        else if (message.Type == ActivityTypes.Ping)
        {
            }

        return null;
    }
}
```

Running the Application

Now let's run the bot project. To run the project file, select the green Play button with the browser selected, as shown in Figure 2-11.



Figure 2-11. Running the project

When you run the application, you will see that it opens locally in a web browser; see Figure 2-12.

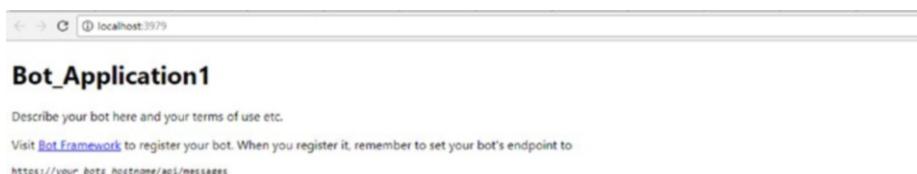


Figure 2-12. The bot running in the web browser

Take note of the localhost address. For testing purposes, you have to use this link. Now, let's start the emulator. Figure 2-13 shows the screen when the Bot Framework Emulator starts.

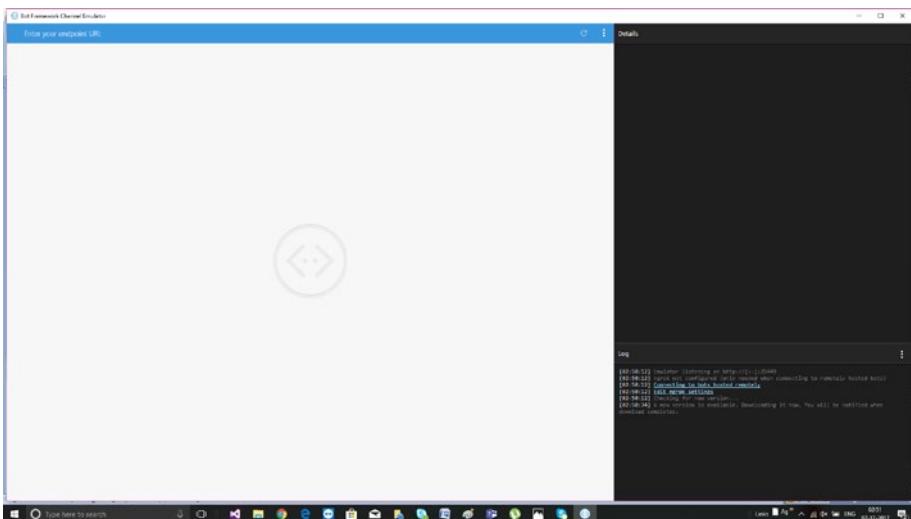


Figure 2-13. The Bot Framework Emulator screen

You have to provide the endpoint URL or the localhost URL for the bot app. After providing the localhost address, and keeping the other options vacant, click the Connect button, as shown in Figure 2-14.



Figure 2-14. Sharing the localhost address and then connecting

Testing the Application

Now let's test the application. First, send a message of *Hi there who are you*. The reply message provides the number of characters, as shown in Figure 2-15.

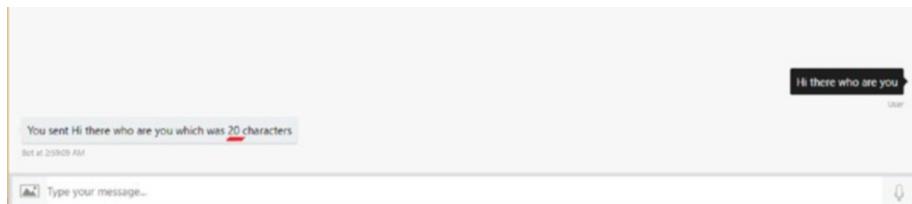


Figure 2-15. The response for the message

The bot code and the logic is a combination of three things, illustrated in Figure 2-16.

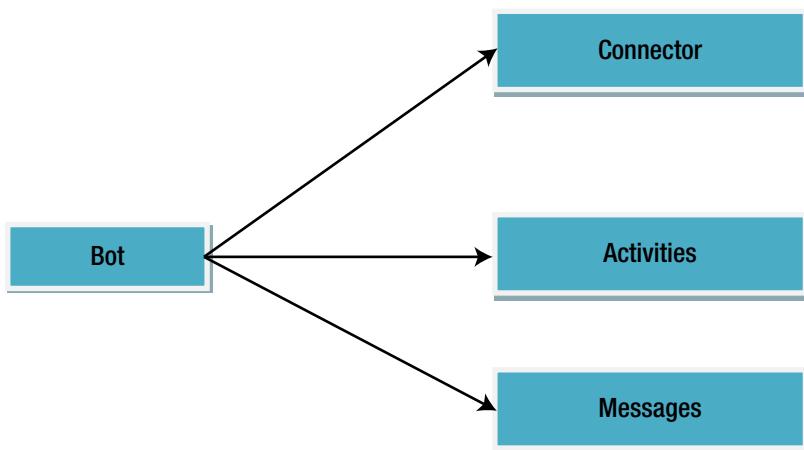


Figure 2-16. Bot code logic

The *connector* handles all communications. The *activities* are the events that occur between a bot and a user. The *messages* are the messages that are displayed between a bot and a user.

Let's modify the code a bit. This time, in the activities section, you'll use Markdown content. So let's go back to the code. Keep everything in the activities section and add the code in Listing 2-5.

Listing 2-5. Markdown Reply Format

```
reply.TextFormat = "markdown";
reply.Text += ",this is a continued effort that we are making";
reply.Text += ", We are writing for Apress";
```

The update Activities code chunk looks like this.

```
if (activity.Type == ActivityTypes.Message)
{
    ConnectorClient connector = new
        ConnectorClient(new Uri(activity.ServiceUrl));
    // calculate something for us to return
    int length = (activity.Text ?? string.Empty).
        Length;

    // return our reply to the user
    Activity reply = activity.CreateReply($"You
        sent {activity.Text} which was {length}
        characters");
    reply.TextFormat = "markdown";
    reply.Text += ",this is a continued effort that
        we are making";
    reply.Text += ", We are writing for Apress";
    await connector.Conversations.
        ReplyToActivityAsync(reply);
}
```

Let's run this code and test it in the emulator. As you run the code in the emulator, you'll see that it prints the messages in Markdown format. The added text is also printed. Figure 2-17 shows the response.

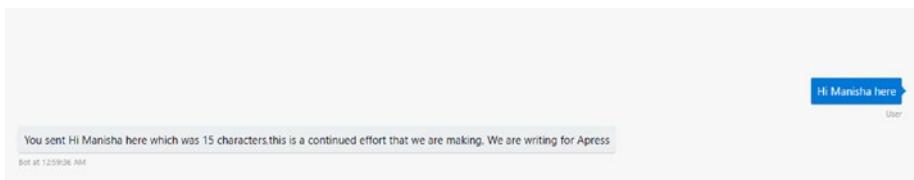


Figure 2-17. The response after updating the code

Managing State

This section shows how to manage states in bots. When we have to manage complex communication, we need to have a kind of communication medium, or a place, to get the conversation going. Figure 2-18 shows options for managing states in bots. Table 2-1 explains each state.

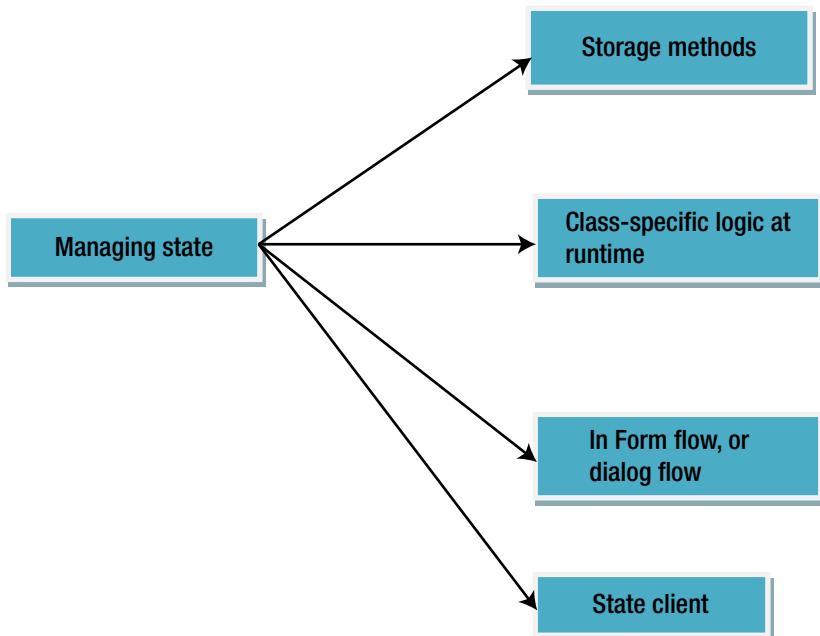


Figure 2-18. Ways of managing bots

Table 2-1. Managing States

Managing State	Description
Storage methods	We can save the state of a bot with the help of databases. We can save the data in SQL Server, Azure, and so forth.
Class-specific logic at runtime	We can initiate a class at runtime and make the bot work. Then we can understand the logic as the bot evolves with different functionality for the lifetime of the bot.
Form flow, or dialog flow	If we want to initiate certain things sequentially, we need to implement form flow, or dialog processes.
State client	This option is similar to viewing state or session states in .NET or MVCs.

Understanding the Use of Dialogs

Dialogs are the flow of conversation to provide a way to communicate a response by using messages in chained manner.

When we are creating a bot, we are working in an efficient manner so that we get a response in proper way. More specifically, we are trying to receive the interaction which should be responsive. Therefore, we are working toward a quality experience of the bot conversation. Dialogs help us realize the perfect experience for the bot.

Let's just work on the code concept for dialogs now. First, open Visual Studio and select the bot template again. Name it ManishaBot, as shown in Figure 2-19.

CHAPTER 2 MICROSOFT BOT FRAMEWORK

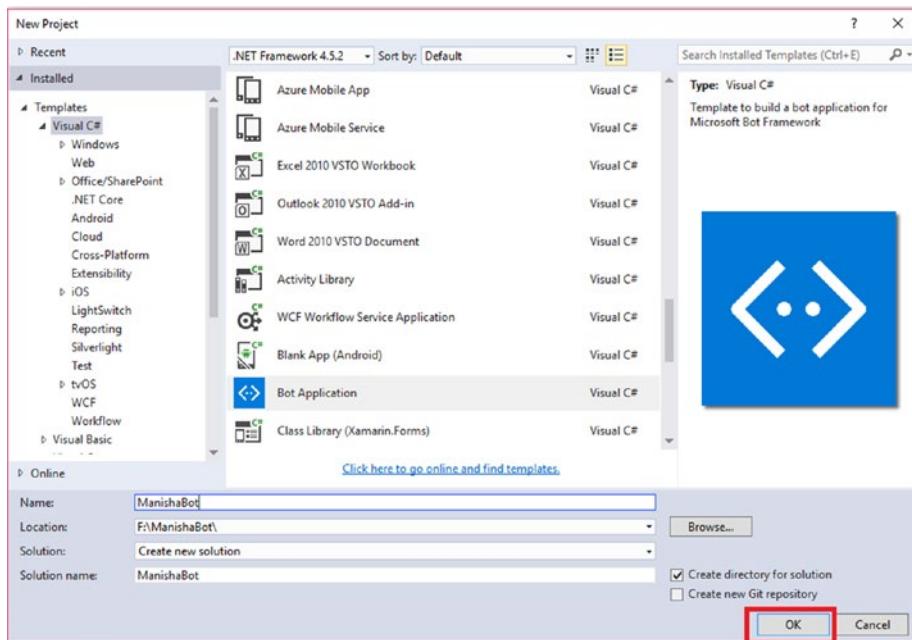


Figure 2-19. Creating the dialog bot

Within the directory structure of the project, you need to create a Dialogs folder. The process for creating the folder is shown in Figure 2-20: click Add ➤ New Folder.

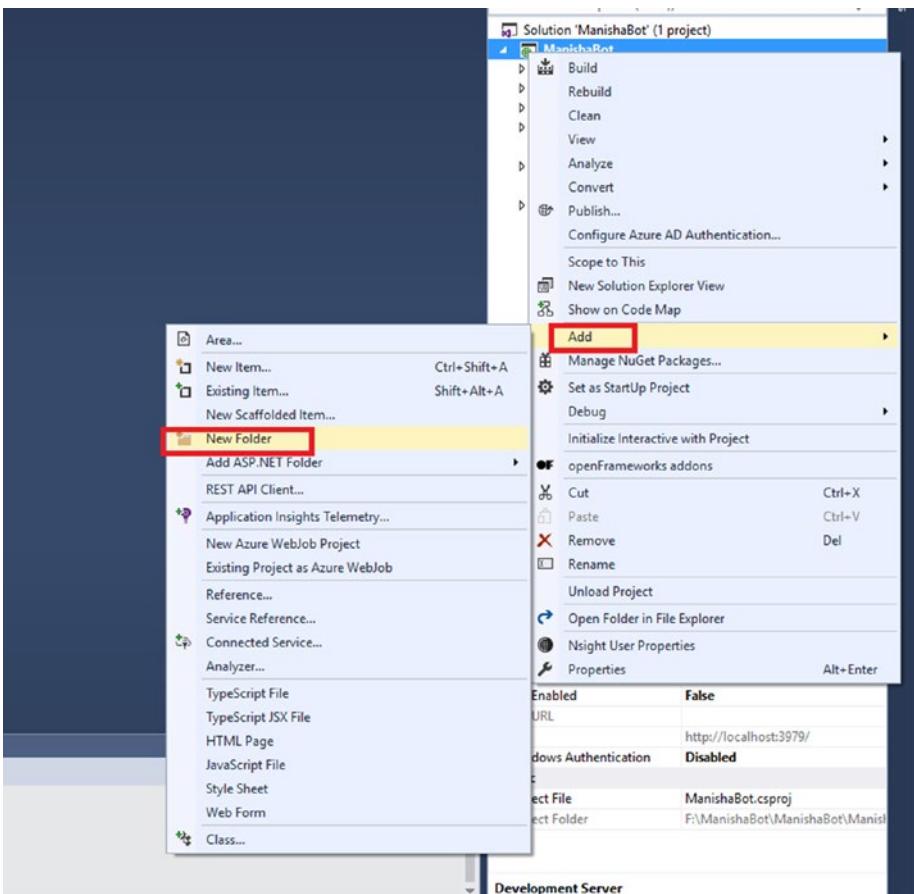


Figure 2-20. Creating a folder named Dialogs

Within the Dialogs folder, you will add one class file. Click Add ► New Item, as shown in Figure 2-21.

CHAPTER 2 MICROSOFT BOT FRAMEWORK

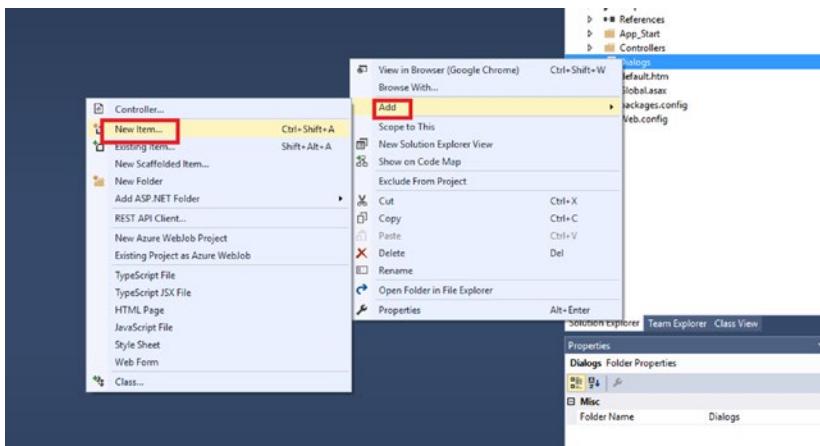


Figure 2-21. Adding a new item. From the Visual Studio we select a new item and then add a class file.

Now you will add a class file. The process, as shown in Figure 2-22. Name the class file RandomFactDialog.cs.

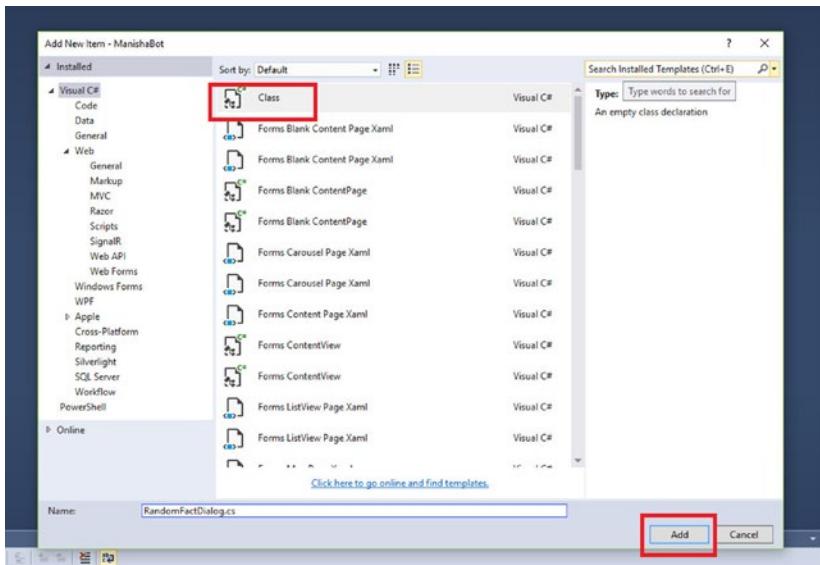


Figure 2-22. Creating a class. Here we are creating a separate class so that we can formulate the logic in here.

Next you'll link the message controller to the dialog class you created; see Listing 2-6.

Listing 2-6. Referencing the RandomFactDialog Class File

```
if (activity.Type == ActivityTypes.Message)
{
    ConnectorClient connector = new
    ConnectorClient(new Uri(activity.ServiceUrl));
    // calculate something for us to return
    // int length = (activity.Text ?? string.
    Empty).Length;

    // return our reply to the user
    // Activity reply = activity.CreateReply($"You
    sent {activity.Text} which was {length}
    characters");
    //await connector.Conversations.
    ReplyToActivityAsync(reply);
    await Conversation.SendAsync(activity, () =>
    Dialogs.RandomFactDialog.Dialog);

}
```

The most important part being this piece of code where we link it to the Dialog class we created.

```
await Conversation.SendAsync(activity, () => Dialogs.
RandomFactDialog.Dialog);
```

Now the RandomFactDialog class file where you implement chaining for messages looks like Listing 2-7.

Listing 2-7. Chaining Messages

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.Bot.Builder.Dialogs;

namespace ManishaBot.Dialogs
{
    [Serializable]
    public class RandomFactDialog
    {
        public static readonly IDialog<object> Dialog = Chain
            .PostToChain()
            .Select(m => "The fact is, you said **" + m.Text + "**")
            .PostToUser();
    }
}
```

Let's run the code and see what exactly happens. Type *hi*. With the dialog chaining, the response is available in the bot, as you can see in Figure 2-23.

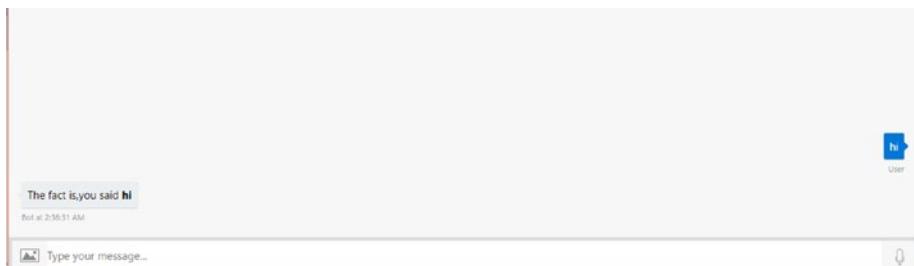


Figure 2-23. The response for the bot

Now you will develop a more complex dialog chaining. Listing 2-8 shows the changes in the structure of the dialog chaining.

Listing 2-8. Chaining a Series of Messages

```
public static readonly IDialog<object> Dialog = Chain
    .PostToChain()
    .Select(m => m.Text)
    .Switch
    (
        Chain.Case
        (
            new Regex("^tell me a fact"),
            (context, text) =>
            Chain.Return("Grabbing a fact...")
            .PostToUser()
            .ContinueWith<string, string>(async (ctx, res) =>
            {
                var response = await res;

                // var fact = await Helpers.GeneralHelper.
                GetRandomFactAsync();

                return Chain.Return("**FACT:** *" + "** We
                are working on a fact that we are writing
                for Apress**" + "*");
            })
        )
    )
```

```
    ),
    Chain.Default<string, IDialog<string>>(
        (context, text) =>
            Chain.Return("Say 'tell me a fact'")
    )
)
.Unwrap().PostToUser();
```

Listing 2-9 provides the class file in its entirety.

Listing 2-9. The Class File Showing the Entire Chaining Process

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.Bot.Builder.Dialogs;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace ManishaBot.Dialogs
{
    [Serializable]
    public class RandomFactDialog
    {
        public static readonly IDialog<object> Dialog = Chain
            .PostToChain()
            .Select(m => m.Text)
            .Switch
            (

```

```
Chain.Case
(
    new Regex("^tell me a fact"),
    (context, text) =>

    Chain.Return("Grabbing a fact...")
    .PostToUser()
    .ContinueWith<string, string>(async (ctx, res) =>
    {
        var response = await res;

        // var fact = await Helpers.GeneralHelper.
        GetRandomFactAsync();

        return Chain.Return("**FACT:** *" + "** We
        are working on a fact that we are writing
        for Apress**" + "*");
    })
),
Chain.Default<string, IDialog<string>>(
    (context, text) =>
        Chain.Return("Say 'tell me a fact'")
)
)
.Unwrap().PostToUser();

public static Task Helpers { get; private set; }

}
}
```

If you run the code now, you will get the result shown in Figure 2-24.



Figure 2-24. Working on a complex chaining process

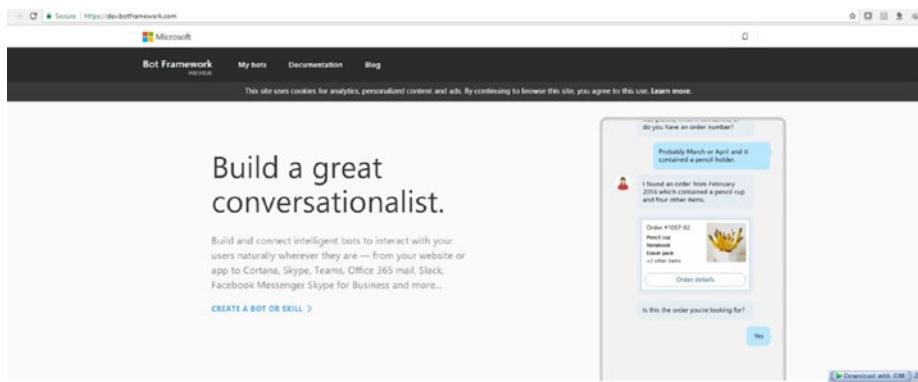
You start the conversation with *hi*. The bot responds with *Say 'tell me a fact'*. When you type *tell me a fact*, the bot gets inside the chaining logic and pops up a message.

Publishing the Bot to the Cloud by Using Azure

In order to publish the bot to the cloud, you first need to register the bot. You do that by heading to <https://dev.botframework.com/>.

The Bot Framework page is shown in Figure 2-25.

CHAPTER 2 MICROSOFT BOT FRAMEWORK



Smart bots start here.

Figure 2-25. The Bot Framework page

At the top of the page, click the My Bots link, as shown in Figure 2-26.

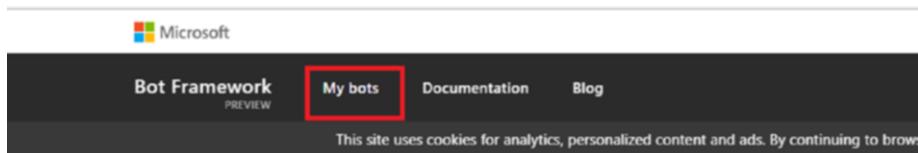


Figure 2-26. My Bots option

From the My Bots page, click the Create a Bot button, shown in Figure 2-27.



Figure 2-27. The Create a Bot option

CHAPTER 2 MICROSOFT BOT FRAMEWORK

On the next page, click the Create button, shown in Figure 2-28.

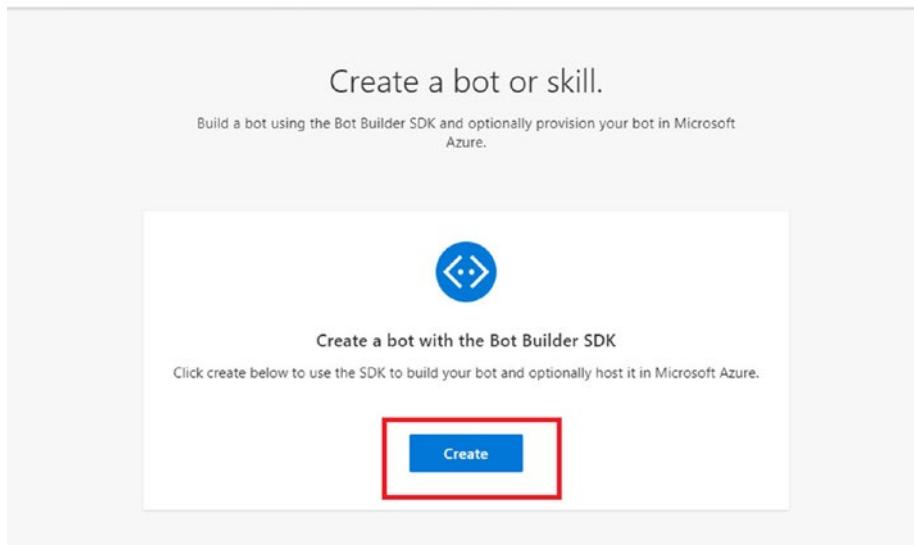


Figure 2-28. Creating a bot

Click the radio button labeled Register an Existing Bot Using Bot Builder SDK, shown in Figure 2-29. Then click the OK button.

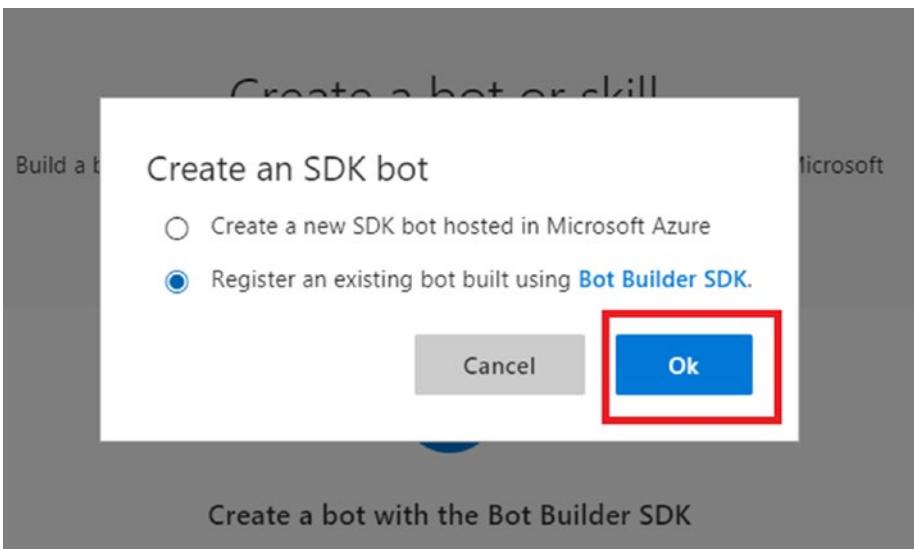


Figure 2-29. Selecting the Bot Builder option

Next, provide the details for the bot, as shown in Figure 2-30.

The screenshot shows a web form for creating a bot. At the top, there's a header with a Microsoft logo and links for "Bot Framework", "My bots", "Documentation", and "Blog". Below the header, the main title is "Tell us about your bot". Under the "Bot profile" heading, there are three input fields: "Display name" containing "ManishaBot", "Bot handle" containing "SimpleBotManisha", and "Long description" containing "Just for testing". There's also a "Icon" section with a placeholder icon and a link to "Upload custom icon". Under the "Configuration" heading, there's a single input field for "Messaging endpoint" with the URL "https://manishabot.scm.azurewebsites.net/api/messages" entered.

Figure 2-30. Providing details about the bot

CHAPTER 2 MICROSOFT BOT FRAMEWORK

After you provide the details for the bot, you need to generate an application ID and password. Click the option to Create Microsoft App ID and password, as shown in Figure 2-31.



Figure 2-31. Generating App ID and password

In the next window, your application ID is displayed, as shown in Figure 2-32.

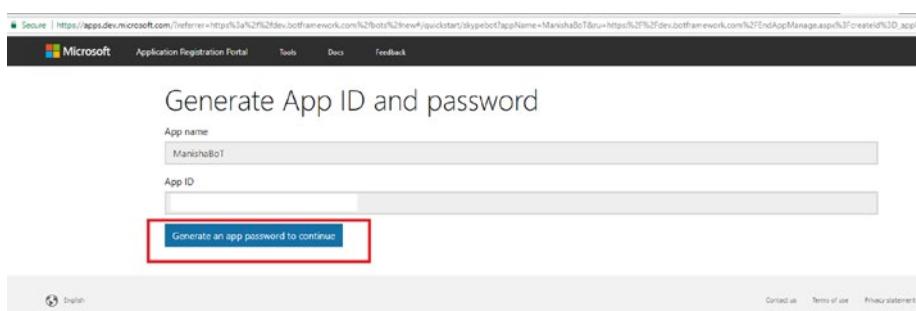


Figure 2-32. You have the app ID

Click the Generate an App Password to Continue button. Type in the application ID, password, and bot handle in the `web.config` file. The new password is generated, as shown in Figure 2-33.

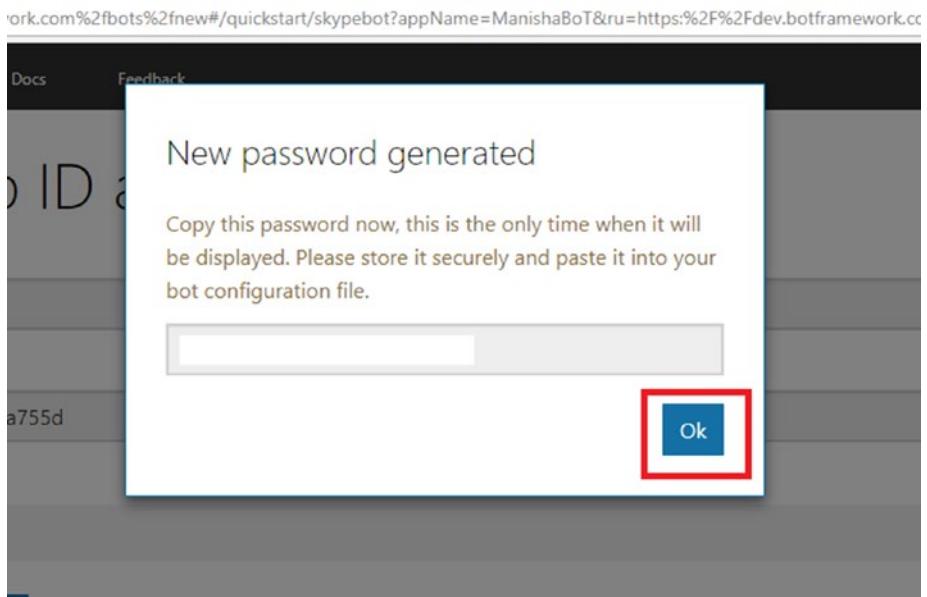


Figure 2-33. The new password is generated

To publish the bot from Visual Studio, right-click it to access the Publish option, as shown in Figure 2-34.

CHAPTER 2 MICROSOFT BOT FRAMEWORK

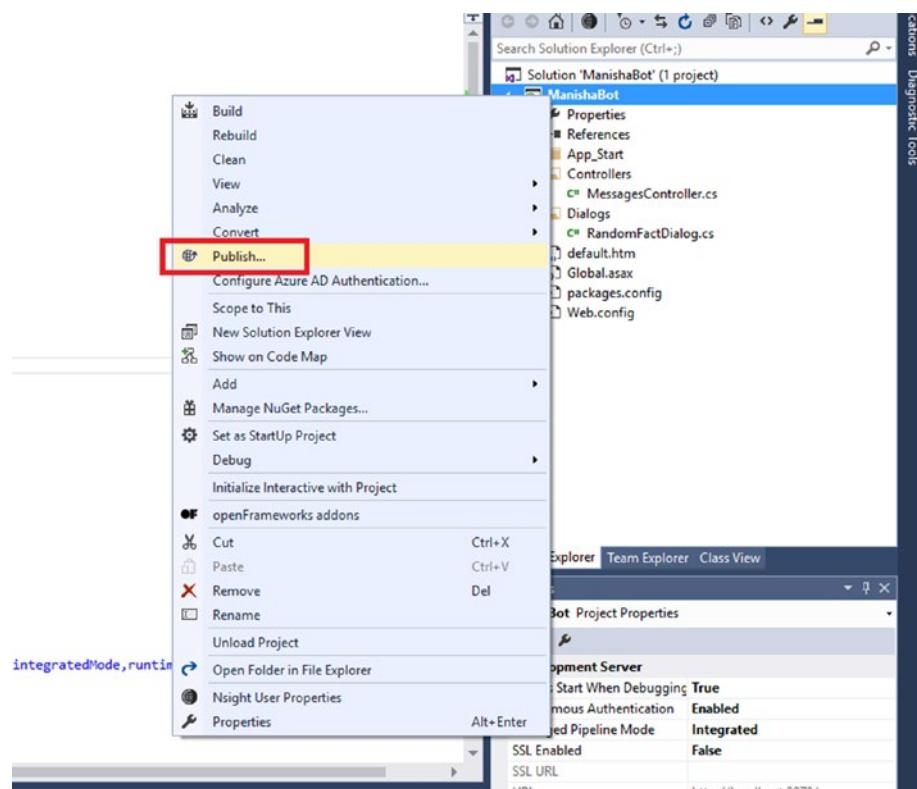


Figure 2-34. Preparing for publishing

The App Service screen opens, as shown in Figure 2-35. Click the New button for the Azure service.

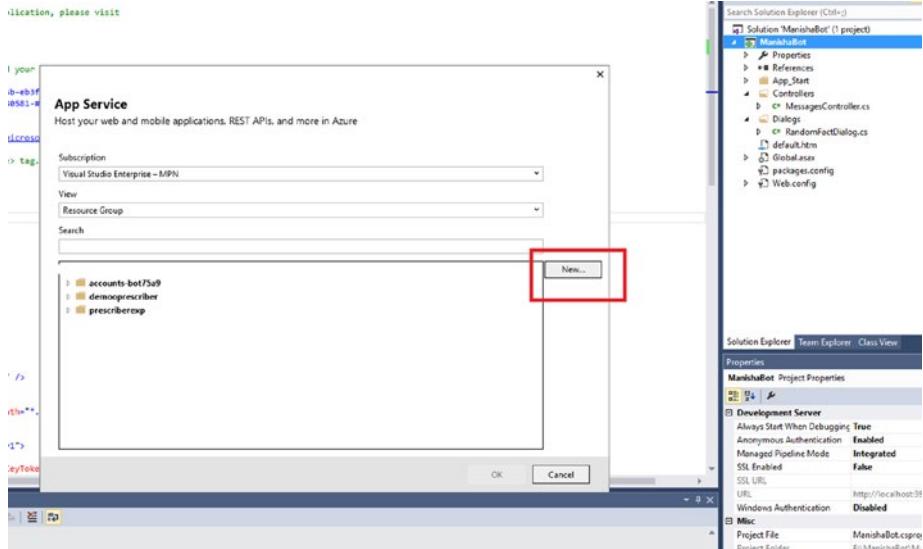


Figure 2-35. Click the New button

The bot is ready to be published. You will validate the connection and publish, as shown in Figure 2-36.

CHAPTER 2 MICROSOFT BOT FRAMEWORK

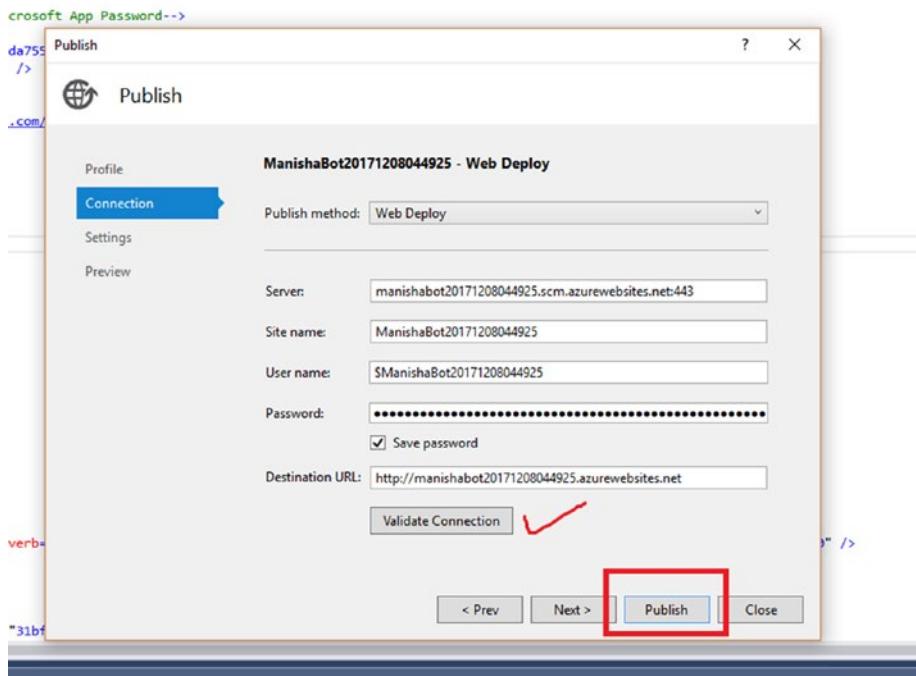


Figure 2-36. Validating and publishing the bot

After the bot is published, you need to edit the configuration in the Bot Framework portal. Figure 2-37 shows how the Azure web page looks.

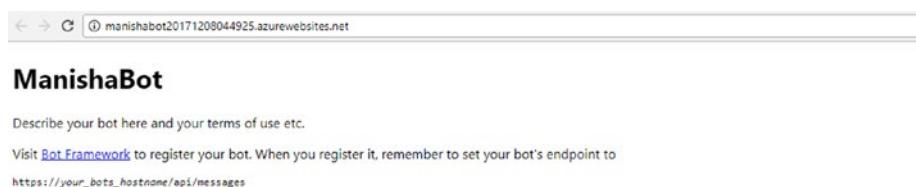


Figure 2-37. The bot is published in Azure

Now head back to the Bot Framework portal. You need to edit the message endpoint for the bot. It will be the Azure web site followed by /api/messages. See the underlined section in Figure 2-38.

The screenshot shows the Microsoft Bot Framework portal's configuration page. At the top, there is a section titled "Bot profile" with a green icon containing two arrows pointing in opposite directions. Below this, there are fields for "Display name" (ManishaBot), "Bot handle" (SimpleBotManisha), and "Long description" (Just for testing). Under the "Configuration" section, there is a field for "Messaging endpoint" containing the URL <http://manishabot20171208044925.azurewebsites.net/api/messages>. A red underline highlights this URL. At the bottom of the configuration section, there is a link to "Register your bot with Microsoft to generate a new App ID and password".

Figure 2-38. Adding the details in the configuration option

The bot is now ready and in the cloud.

Conclusion

This chapter presented the Microsoft Bot Framework so that you could learn about its fundamentals, and publish a bot to the cloud.

You learned about the Bot Framework template and created an example that could recognize the characters sent to the bot. You saw how to reference a class file in the main bot framework logic. Next, you learned

CHAPTER 2 MICROSOFT BOT FRAMEWORK

how to log in to the Bot Framework web site to generate the application ID and password needed for your `web.config` file. Finally, you learned how to publish the bot from Visual Studio to Azure Cloud.

This first introduction forms the basis of how a bot framework works. In the next chapter, you'll learn about new bots from other organizations.

CHAPTER 3

Wit.ai and Dialogflow

This chapter covers the basics of [Wit.ai](#) and [Dialogflow](#), two chatbot interfaces from Google that provide helpful capabilities. You'll start by exploring a simple bot flow using Wit.ai, and then move on to Dialogflow to create a full-fledged bot to deploy to the Web.

Getting Started with Wit.ai

Wit.ai is a web-based IDE for creating bots. To launch Wit.ai, you have to go to <https://wit.ai/>.

In this section, you'll work on creating a new app and then you'll add intent to it. Next, you'll add text and keywords to modify the bot. Later in this chapter, you'll implement the Dialogflow tools to implement the bot.

Creating a New App

In this section, you will work on creating your first app. From the Wit.ai web site, click Quick Start, as shown in Figure 3-1.

CHAPTER 3 WIT.AI AND DIALOGFLOW

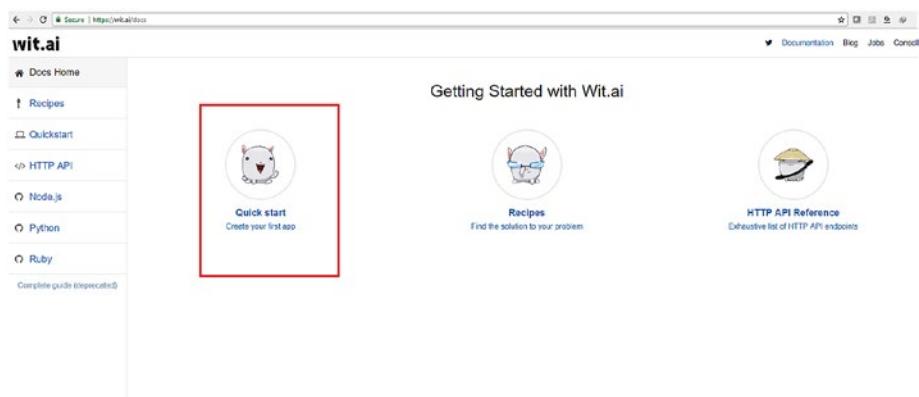


Figure 3-1. Choosing the Quick Start option to create an app

This opens a page that presents options to create the first app by using either GitHub or Facebook, as shown in Figure 3-2.

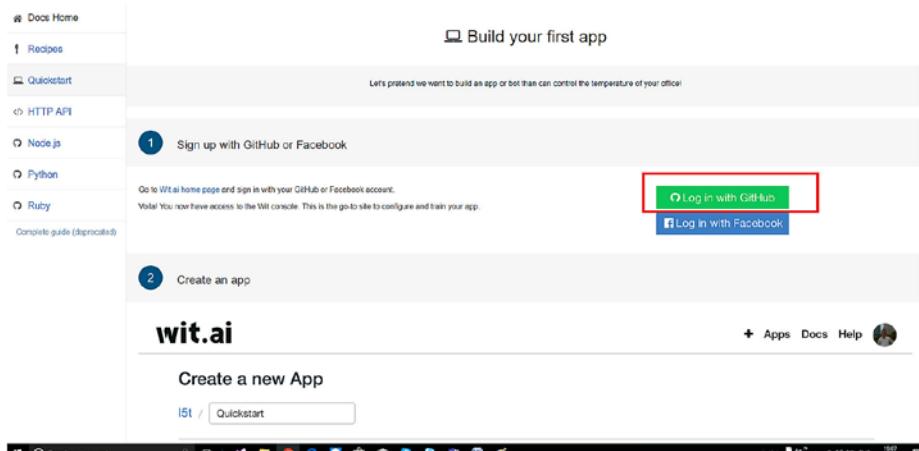


Figure 3-2. Logging in using GitHub

After you log in, you'll have the option to create your new app, as shown in Figure 3-3.

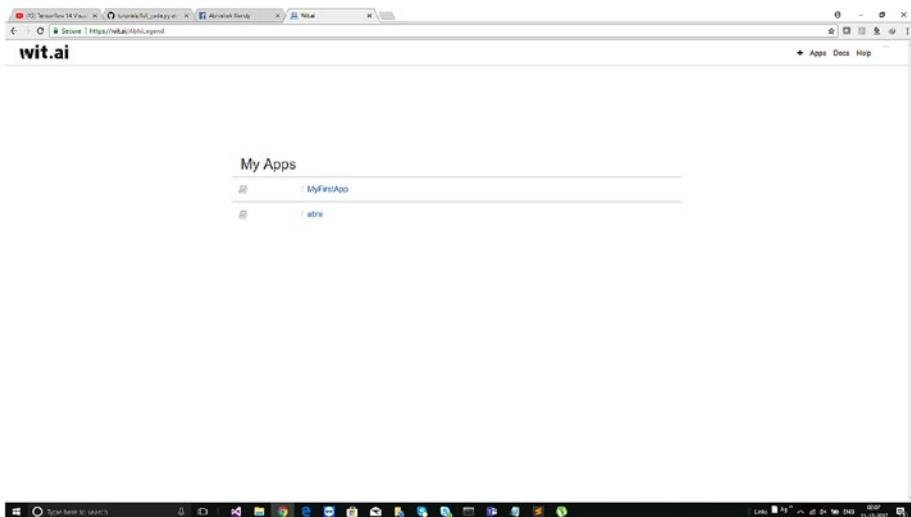


Figure 3-3. Creating a new app

Let's create a simple app. You'll name the app *TestStories*. In the app description section, type *Demo*. Select the Private option for the app data. Then click the Create App button, shown in Figure 3-4.

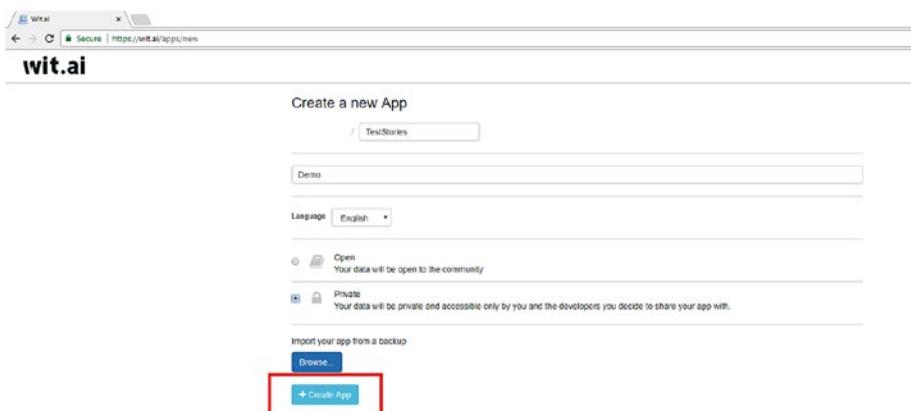


Figure 3-4. Setting the details for your new app

Adding Intent

Clicking the Create App button opens the next page, which presents the Stories option. Following that, you head to the Understanding option; this page is shown in Figure 3-5.

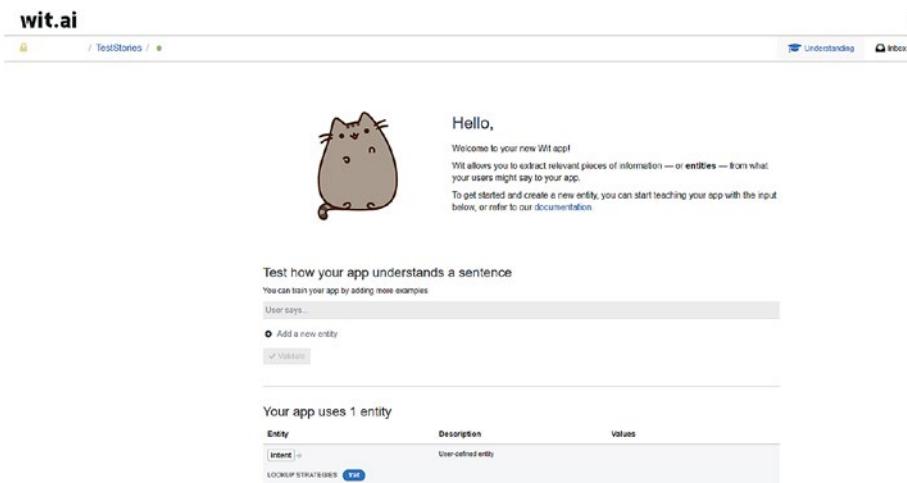


Figure 3-5. The Understanding options at the Wit.ai site

Let's work through the new app with a pizza example.

In the User Says text box, type *I want some cheese pizza*, as shown in Figure 3-6.

Test how your app understands a sentence

You can train your app by adding more examples

I want some cheese pizza

Add a new entity

Validate

Your app uses 1 entity

Entity	Description	Values
intent	User-defined entity	
LOOKUP STRATEGIES trait		

Figure 3-6. Adding details we are working towards ordering pizza so we are constructing the way to start the conversation

Now you're ready to create an intent. You add the value for the intent as pizza, as shown in Figure 3-7.

Test how your app understands a sentence

You can train your app by adding more examples

I want some cheese pizza

Add a new entity

Validate

Your app uses 1 entity

Entity	Description	Values
intent	User-defined entity	
LOOKUP STRATEGIES trait		

Figure 3-7. Creating a new intent. An intent is needed in here so that while conversation it is able to find the right order for communication.

CHAPTER 3 WIT.AI AND DIALOGFLOW

You add a pizza option with cheese and finally click the Validate button, as shown in Figure 3-8.

Test how your app understands a sentence

You can train your app by adding more examples

The screenshot shows the Wit.ai testing interface. At the top, it says "Test how your app understands a sentence" and "You can train your app by adding more examples". Below this, a text input field contains "I want some cheese pizza". To the right of the input are two dropdown menus: "Create new value 'pizza'" and "Create new value 'cheese'". A green "Validate" button is highlighted with a red box and a checkmark. Above the validate button, there are three radio button options: "intent" (unchecked), "pizza" (unchecked), and "Add a new entity" (checked). Red checkmarks are placed next to the "Create new value" dropdowns and the "Validate" button.

Figure 3-8. Creating an intent and then validating it. We are creating different intent so that we are giving an option to choose for the user which type of pizza they want to have.

After validating that is creation of the entities option, you have two entities created now, highlighted in Figure 3-9.

Test how your app understands a sentence

You can train your app by adding more examples

The screenshot shows the Wit.ai testing interface. At the top, it says "Test how your app understands a sentence" and "You can train your app by adding more examples". Below this, a text input field contains "User says...". To the right of the input is a radio button option "Add a new entity" (checked) and a greyed-out "Validate" button.

Your app uses 2 entities

Entity	Description	Values
Intent	User-defined entity	pizza
pizza	User-defined entity	cheese

Figure 3-9. The app shows two intents

Adding Text and Keywords

Let's work on modifying the bot. We will work now on creating the flow of the chatbot so the user is able to order different kind of pizzas.

In this section, you'll try to add a sentence. Figure 3-10 shows how to add a sentence for recognition.

Test how your app understands a sentence

You can train your app by adding more examples

I want to order **cheese pizza**

<input type="button" value="intent"/>	<input type="button" value="pizza"/>
<input checked="" type="radio"/> pizza	<input type="button" value="cheese"/>
<input type="radio"/> Add a new entity	
<input type="button" value="✓ Validate"/>	

Figure 3-10. The app is trying to understand a sentence. Here we are working on how a chatbot will be able to discover what user is typing so that the sentence that is created is understandable to the bot.

Next, click Validate. Now you need to select an entity. Choose Pizza and click the arrow associated with it (see Figure 3-11).

Your app uses 2 entities

Entity	Description	Values
<input type="button" value="intent"/> →	User-defined entity	pizza
LOOKUP STRATEGIES <input type="button" value="trait"/>		
<input type="button" value="pizza"/> → 	User-defined entity	cheese
LOOKUP STRATEGIES <input type="button" value="free-text & keywords"/>		

Figure 3-11. Select an entity and the arrow associated with it

CHAPTER 3 WIT.AI AND DIALOGFLOW

When you click the Pizza arrow, you head to the page that has keywords and synonyms, as shown in Figure 3-12.

The screenshot shows the Wit.ai interface for the 'pizza' entity. At the top, there are tabs for 'LOOKUP STRATEGIES' (with 'trait' and 'free-text & keywords' selected), 'free-text' (disabled), and 'keywords'. To the right, it says 'User-defined entity'. Below this, the word 'pizza' is entered into a search bar. Under the heading 'Insights', there is a note to 'Validate more expressions to get insights for this entity'. The 'Keywords' section contains a table with one row: 'cheese' under 'Keyword' and a link to 'Add synonym...' under 'Synonyms'. Below this, there is a field labeled 'Add a new keyword' with a red checkmark drawn over it. The 'Expressions' section shows three examples: 'Search through your expressions.', 'Text' (empty), and two entries starting with double quotes: 'I want to order cheese pizza' and 'I want some cheese pizza', each with a small 'x' icon to its right. A 'See More' button is at the bottom of this list.

Figure 3-12. Page for adding a new keyword

Here you'll add new keywords. The results are shown in Figure 3-13.

The screenshot shows the Wit.ai interface for the entity 'pizza'. At the top, there are tabs: 'LOOKUP STRATEGIES' (with 'trait' selected), 'free-text & keywords' (selected), 'free-text', and 'keywords'. To the right is a 'User-defined entity' button and a trash can icon. Below the tabs, under 'Insights', it says 'Validate more expressions to get insights for this entity'. Under 'Keywords', there is a table:

Keyword	Synonyms
vege	<input type="button" value="Add synonym..."/>
cheese	<input type="button" value="Add synonym..."/>
heat lovers	<input type="button" value="Add synonym..."/>
pepperoni	<input type="button" value="Add synonym..."/>

At the bottom left of the keyword table is a link ''. A red arrow points to the word 'pepperoni' in the table.

Figure 3-13. Added keywords so that we are exactly creating the ways different type of pizzas are ordered

Now click the Understanding option in order to train the bot to recognize the new keywords that you added. It is visible at the bottom.

Let's start with pepperoni pizza. For understanding the flow of the intent for pepperoni pizza, type in *I want a pepperoni pizza*, we are showing how the bot can be made to order a specific pizza as shown in Figure 3-14. Then click Validate.

Test how your app understands a sentence

You can train your app by adding more examples

The screenshot shows the 'Test how your app understands a sentence' section. A text input field contains the sentence 'I want some pepperoni pizza'. Below it, there are two dropdown menus: 'intent' (set to 'pizza') and 'entity' (set to 'pepperoni'). There are also three buttons: 'Add a new entity', 'Validate' (which is highlighted with a red box), and a close button.

Figure 3-14. Adding the pepperoni pizza option

Now you'll add veggie pizza, as shown in Figure 3-15.

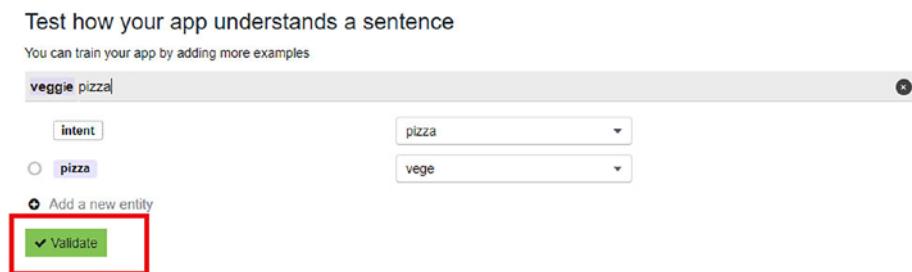


Figure 3-15. Adding veggie pizza option

Let's now add a Large pizza option. For that, you need to make some changes, shown in Figure 3-16.



Figure 3-16. You need to change the pizza option

Delete the Pizza option, because you want to work on the size of the pizza. Click the cross option to delete the pizza value, as shown in Figure 3-17.



Figure 3-17. Deleting the pizza option

Creating a New Entity

In this section, you'll create an entity named *pizzaSize*, shown in Figure 3-18.



Figure 3-18. Creating the *pizzaSize* entity

After creating the intent, the bot page looks like Figure 3-19.

The screenshot shows the Wit.ai interface for managing intents. At the top, it says "Test how your app understands a sentence" and "You can train your app by adding more examples". Below this, there are two input fields: "intent" (set to "Large pizza") and "entity" (set to "pizza"). There are two radio button options: "pizzaSize" (selected) and "Add a new entity". A green "Validate" button is below these. At the bottom, it says "Your app uses 3 entities". A table shows the entities:

Entity	Description	Values
intent →	User-defined entity	pizza
LOOKUP STRATEGIES trait		
pizzaSize →	User-defined entity	
pizza →	User-defined entity	vege, cheese, meat lovers, pepperoni
LOOKUP STRATEGIES free-text & keywords		

Figure 3-19. Adding a new intent

Implementing Wit.ai with Facebook

You've gone through the process of creating a simple app that uses some of the features of Wit.ai. In this section, you'll implement Wit.ai with Facebook. The Facebook API is available at developers.facebook.com. You will create a chatbot for the official Facebook page, or any Facebook page. This chatbot will add to Facebook Messenger. You will have to set up webhooks accordingly. To get webhooks working, you need to set up ngrock.

First, create a new app as you've done before. This time, name it Test1, as shown in Figure 3-20.

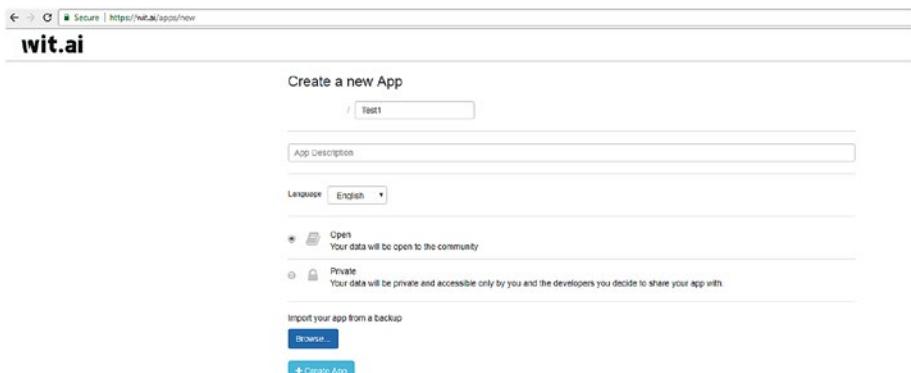


Figure 3-20. Adding a new name

Now head back to the Facebook for Developers site at <https://developers.facebook.com/>. You need to use your Facebook credentials to log in. The page looks Figure 3-21.

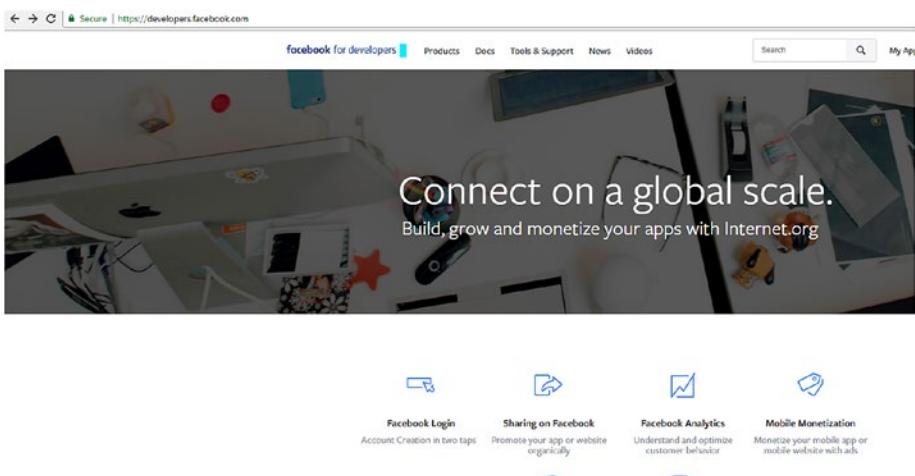


Figure 3-21. The Facebook for Developers page

Now you need to create a new app. Click the My Apps option at the top right to access the drop-down menu for creating a new app, as shown in Figure 3-22.

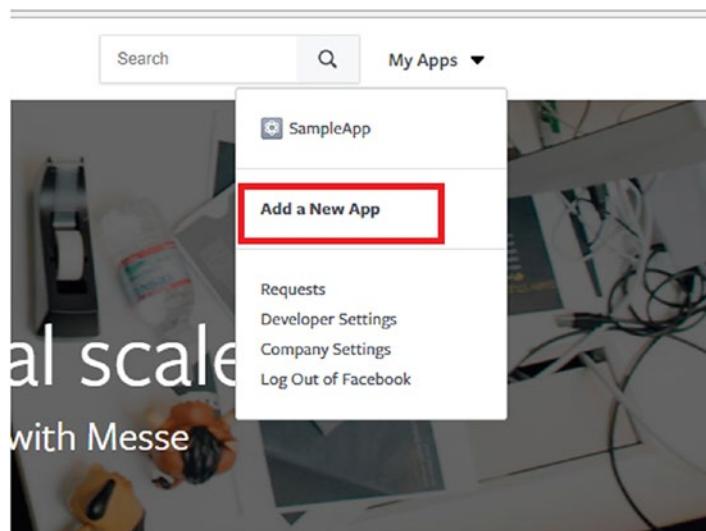


Figure 3-22. Adding a new app

CHAPTER 3 WIT.AI AND DIALOGFLOW

Name it *SampleApp* and add a product to it by clicking the Add a Product button, as shown in Figure 3-23.

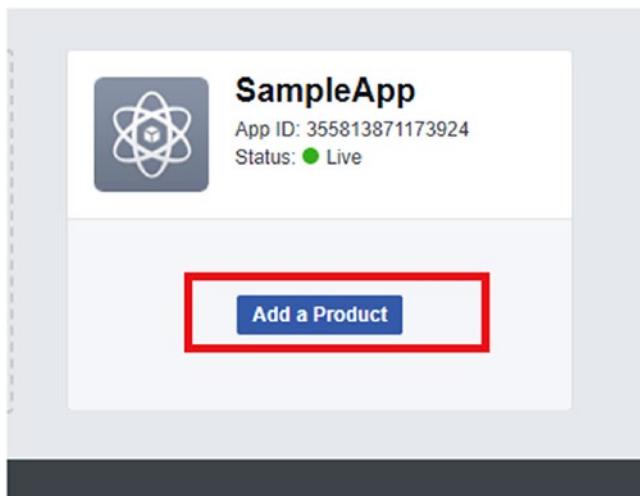


Figure 3-23. Adding a sample app

For a product, you need to add a messenger, shown in Figure 3-24.

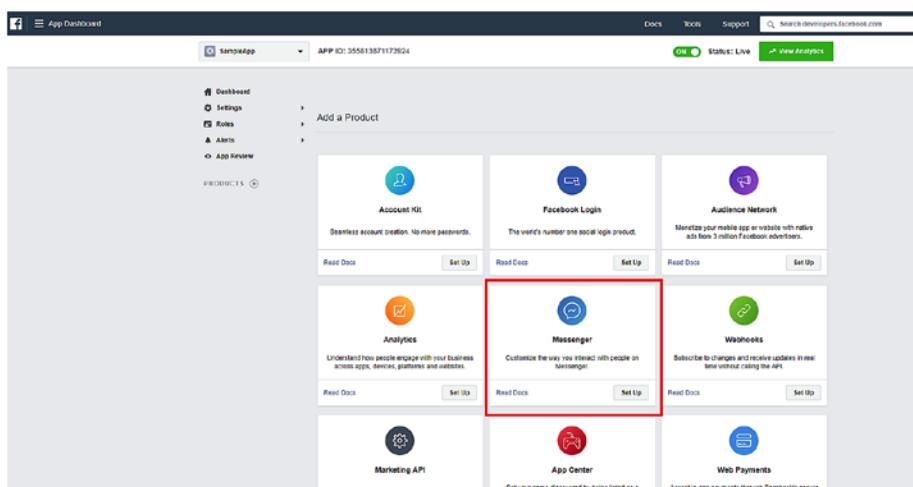


Figure 3-24. Selecting the Messenger option

Click the Set Up button in the Messenger option. You will be using Wit.ai's natural language processing (NLP) rather than Facebook's (which is very basic), although both of them work perfectly together. Now you have to set up a page with the app name created; xxxxxx, as shown in Figure 3-25. Your goal is to create a chatbot for the official Facebook page.

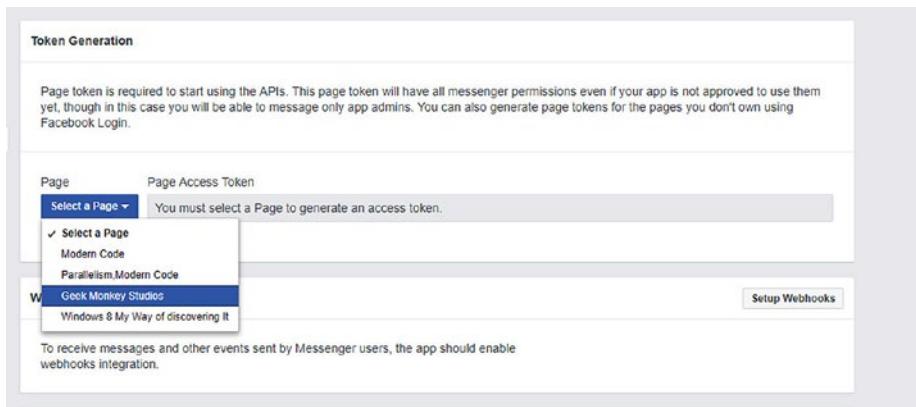


Figure 3-25. Selecting a web page in Facebook

The site will ask for authentication, and then the token will be created. Now you need to quickly set up webhooks. Click the Setup Webhooks button, shown in Figure 3-26.

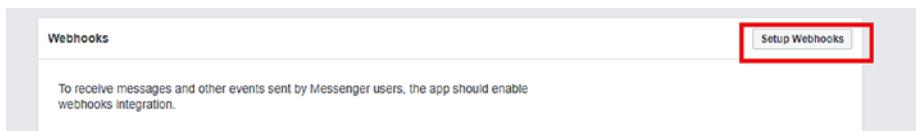


Figure 3-26. Setting up Webhooks

Clicking this option opens a page for assigning details. Before using webhooks, you need to configure ngrok. Go to the ngrok site at <https://ngrok.com/>, shown in Figure 3-27.

CHAPTER 3 WIT.AI AND DIALOGFLOW

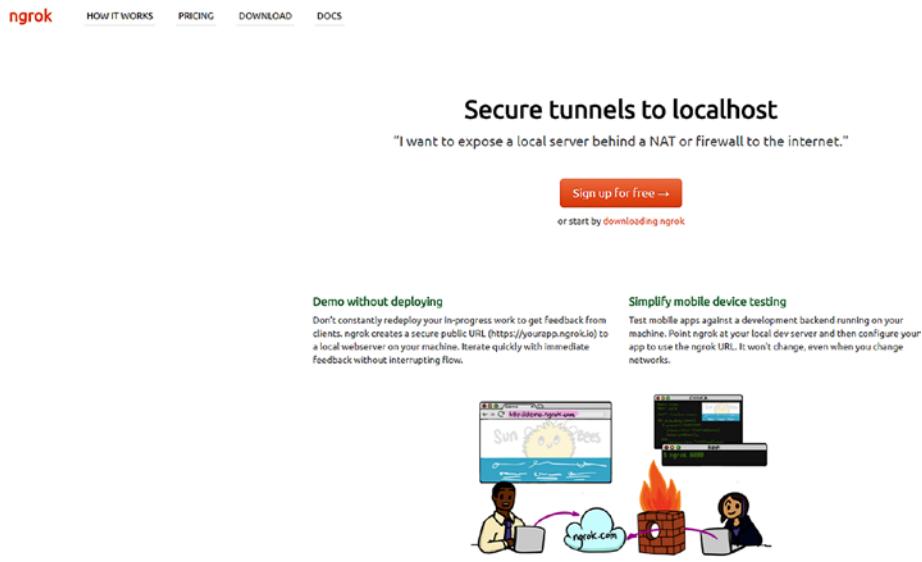


Figure 3-27. The ngrok site

Download ngrock by clicking the Downloading Ngrok link to access the download options, shown in Figure 3-28.

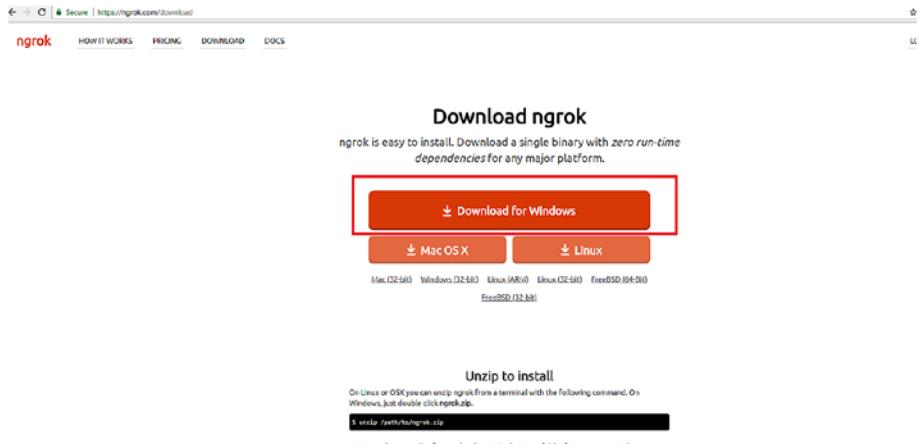
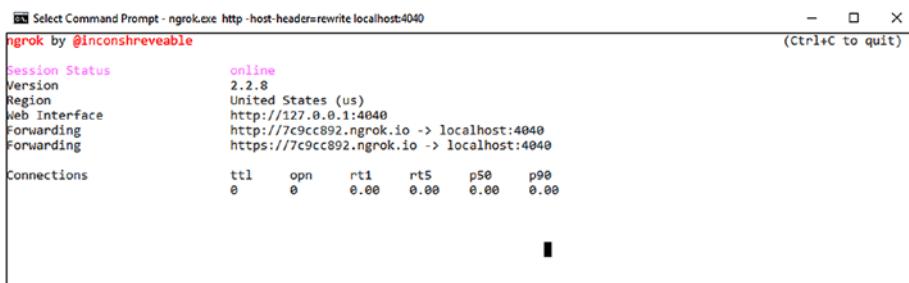


Figure 3-28. Downloading ngrok for Windows

After downloading ngrock, launch ngrok.exe from the command prompt. You have to open the command prompt and head to the localhost page. At my end, the localhost was 4040. Key in the following details at the command prompt:

```
F:\ngrock>ngrok.exe http -host-header=rewrite localhost:4040
```

If web server that is ngrock is up, you will see the window in Figure 3-29.

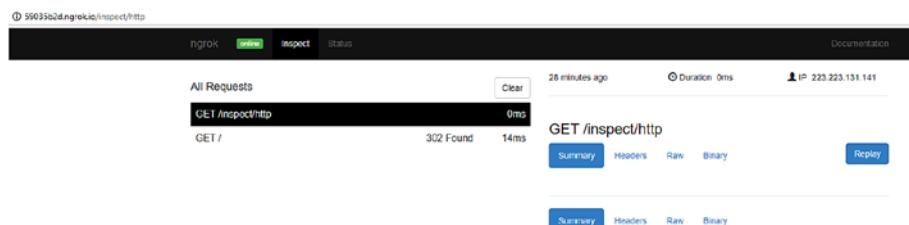


The screenshot shows the output of the command `ngrok.exe http -host-header=rewrite localhost:4040`. It displays the following session information:

Session Status	online
Version	2.2.8
Region	United States (us)
Web Interface	http://127.0.0.1:4040
Forwarding	http://7c9cc892.ngrok.io -> localhost:4040 https://7c9cc892.ngrok.io -> localhost:4040
Connections	ttl opn rti rt5 p50 p99 0 0 0.00 0.00 0.00 0.00

Figure 3-29. The ngrok status

Let's head to the localhost page, shown in Figure 3-30, at <http://7c9cc892.ngrok.io/inspect/http>.

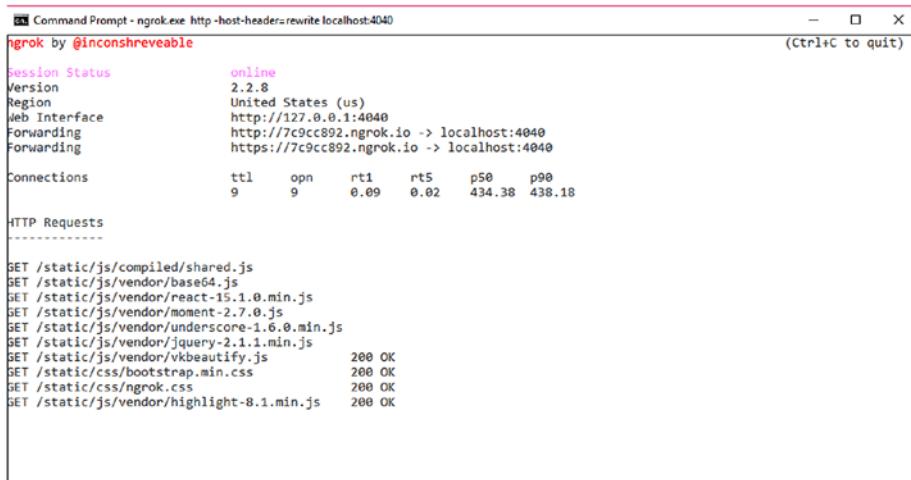


The screenshot shows the inspect page of the ngrok interface. It displays a single request log entry:

- All Requests
- 28 minutes ago
- Duration: 0ms
- IP: 223.223.131.141
- Request: GET /inspect/http
- Response: 302 Found 14ms
- Summary, Headers, Raw, Binary buttons

Figure 3-30. The inspect page as reflected that carries the information of the localhost page

When the page is up, you can see the statistics in the command-prompt window, as shown in Figure 3-31.



```
Command Prompt - ngrok.exe http -host-header=rewrite localhost:4040
ngrok by @inconshreveable                                         (Ctrl+C to quit)

Session Status          online
Version                2.2.8
Region                 United States (us)
Web Interface          http://127.0.0.1:4040
Forwarding             http://7c9cc892.ngrok.io -> localhost:4040
Forwarding             https://7c9cc892.ngrok.io -> localhost:4040

Connections            ttl     opn      rt1     rt5    p50     p90
                        9       9      0.09    0.02   434.38  438.18

HTTP Requests
-----
GET /static/js/compiled/shared.js
GET /static/js/vendor/base64.js
GET /static/js/vendor/react-15.1.0.min.js
GET /static/js/vendor/moment-2.7.0.js
GET /static/js/vendor/underscore-1.6.0.min.js
GET /static/js/vendor/jquery-2.1.1.min.js
GET /static/js/vendor/vkbeautify.js      200 OK
GET /static/css/bootstrap.min.css        200 OK
GET /static/css/ngrok.css               200 OK
GET /static/js/vendor/highlight-8.1.min.js 200 OK
```

Figure 3-31. You have set up the ngrok environment

Deploying the Wit.ai bot is easy after you've set up the ngrok environment as well as the webhook's natural language processing section. Next you will work with Dialogflow to create a complete bot that's deployable to the Web.

Working with Dialogflow

In this section, you'll use Dialogflow (formerly Api.ai) to create a bot. First you'll access the web console through your Google account. Then you'll create a Pizza bot. Finally, you'll use Small Talk to xxxxx and then link the bot to a Google project. A small talk is a way to access prebuilt API for doing the conversation in a proper manner.

Let's start now.

Accessing Dialogflow

This section introduces you to Dialogflow and gets you logged in via your Google account.

Head to the Dialogflow web site at <https://dialogflow.com/>.

The Dialogflow web site for creating chatbots is shown in Figure 3-32.

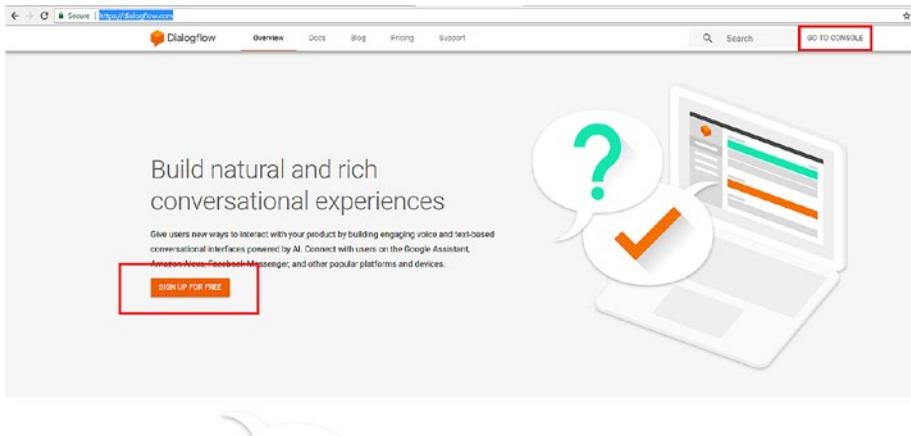


Figure 3-32. The Dialogflow web site

From this Dialogflow page, you can either sign up for free or, if you already have an account, you can go to the console. If you have a Google account, you need to verify it and log in, as shown in Figure 3-33.

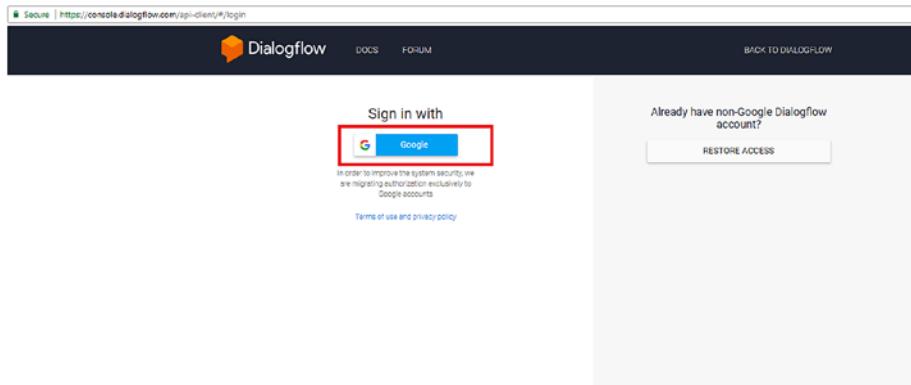


Figure 3-33. Logging in using a Google account

Allow the access, as shown in Figure 3-34.

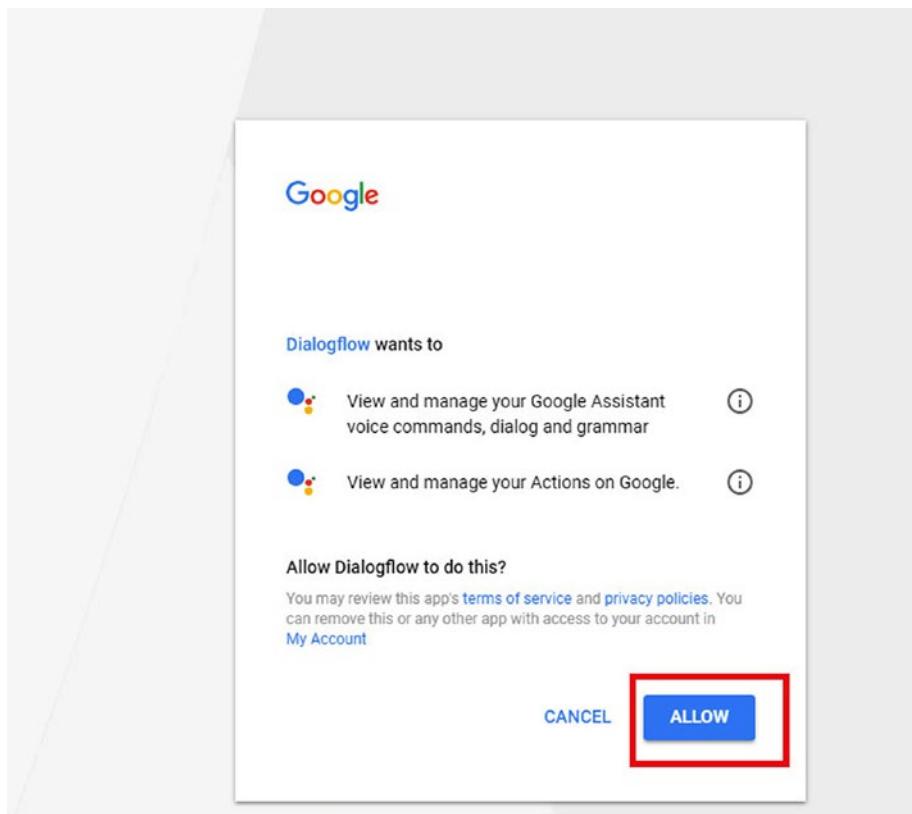


Figure 3-34. Allow Dialogflow to access Goggle Assistant

The next page is a Welcome screen. The apps in Dialogflow are known as *agents*. You will have to create one by clicking Create Agent, as shown in Figure 3-35.

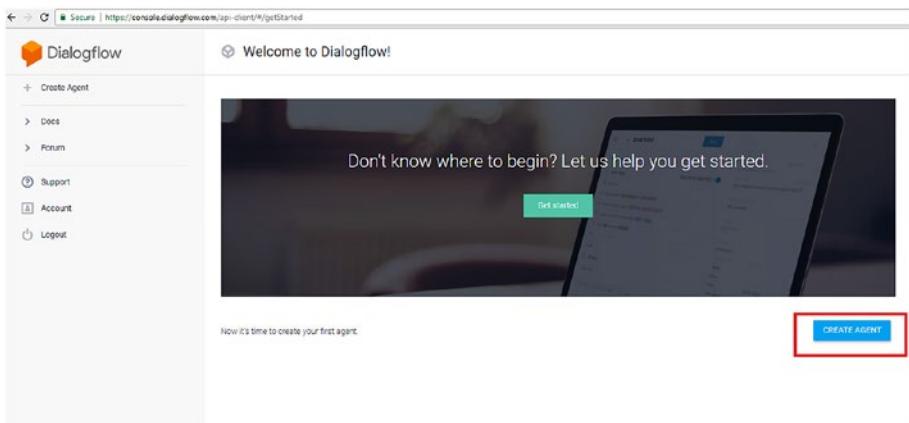


Figure 3-35. The Dialogflow welcome page after loogging in

Creating the Pizza Bot

In this section, you will use Dialogflow to create a simple pizza bot. Name it PizzaBot, as shown in Figure 3-36.

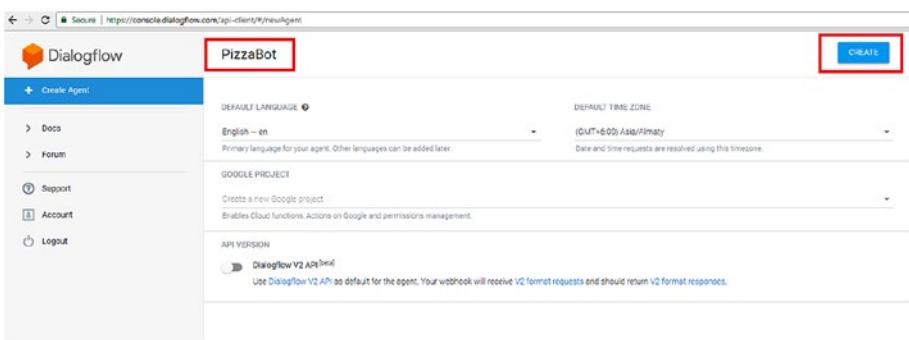


Figure 3-36. Creating a pizza bot

Now you need to figure out what you need to do with the app.

Using Small Talk

You will be using Small Talk for the bot's interactions. You'll use Small Talk for interactions like Hi, Hello, and so forth. Then you'll allow users to order the pizza. Dialogflow has this wonderful built-in API that does all the small talk for us. You just need to enable it, as shown in Figure 3-37.

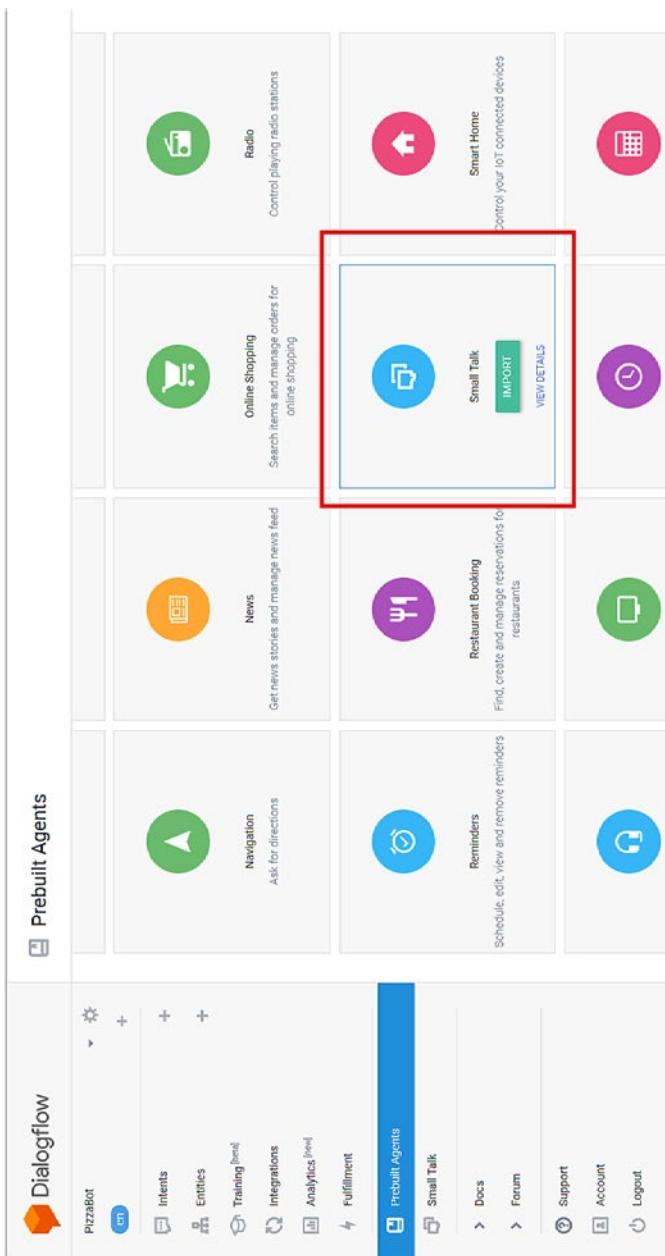


Figure 3-37. Enabling Small Talk prebuilt agents. The user selects the small talk option so that we are importing the basic way the communication occurs.

You then import Small Talk accordingly.

Linking to a Google Project

In this section, you'll see how to integrate your bot with a Google project. You can link to a Google Project or create a new one, as shown in Figure 3-38.

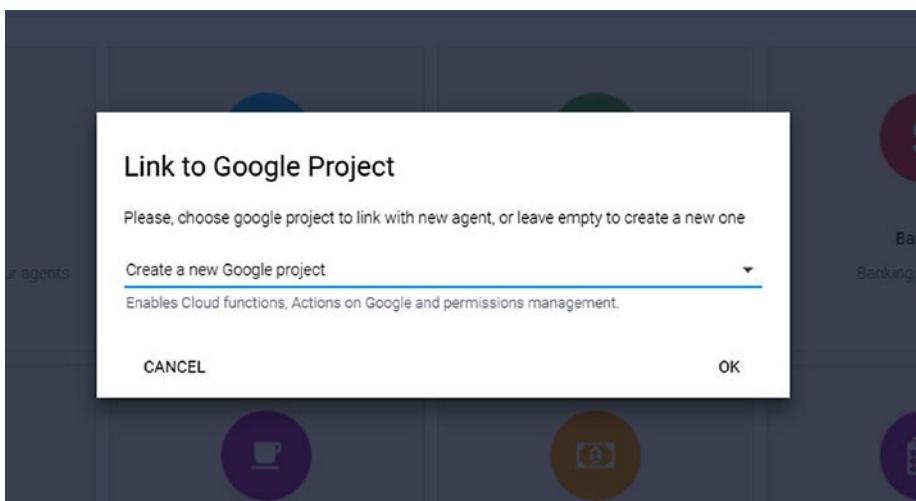


Figure 3-38. Linking to a Google project

After linking to a Google project or naming a new one, enable Small Talk for the PizzaBot app, as shown in Figure 3-39. Small Talk will help us immensely. Small Talk has all the basic conversation flows for us and makes us easier to communicate.

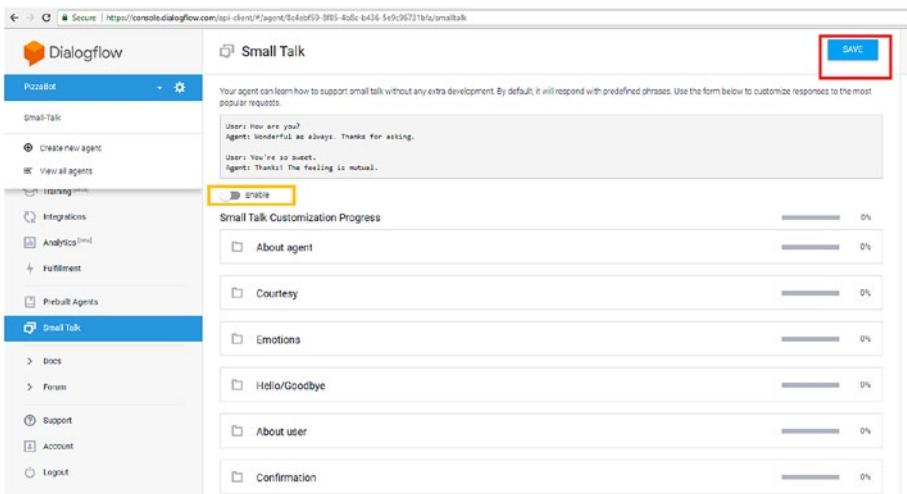


Figure 3-39. Enabling Small Talk and saving it

In the next step, you need to create a new entity for the app.

Adding an Intent

Now you will create an intent, now one of the things user might think of as parameter being hungry. So you will add an intent named Hungry. We will to create a new intent, as shown in Figure 3-40.

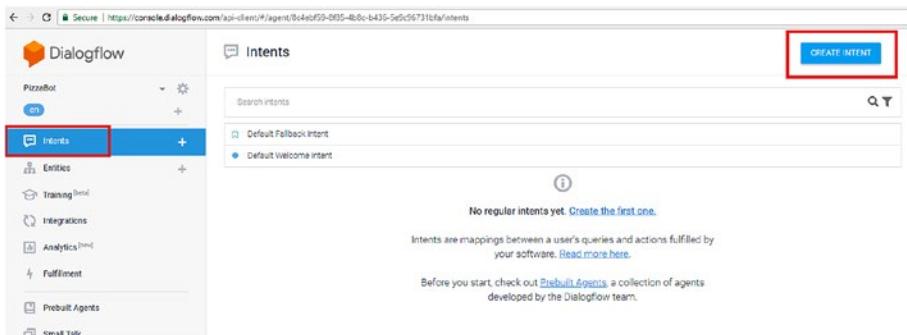


Figure 3-40. Creating a new intent. We would be creating a new intent for the chatbot.

CHAPTER 3 WIT.AI AND DIALOGFLOW

You then add training parameters or examples, as shown in Figure 3-41.

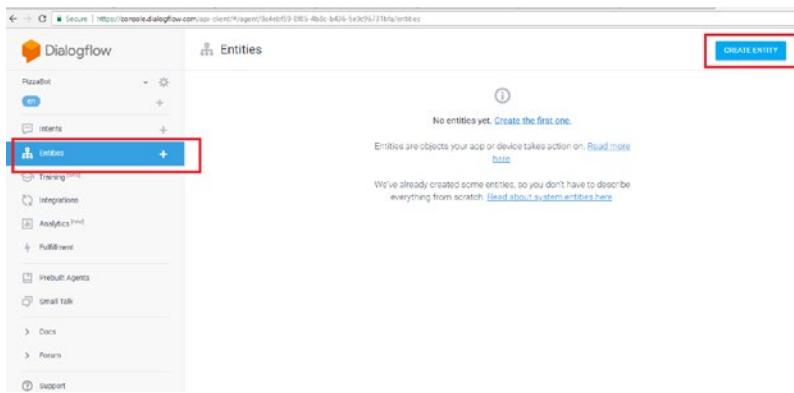


Figure 3-41. Adding parameters

You've created a simple flow for the bot. Now you will work on creating new entities for the it.

Creating a New Entity

In this section, you will work on constructing the bot while creating new entities. Click the Entities option and then click Create Entity, as shown in Figure 3-42.

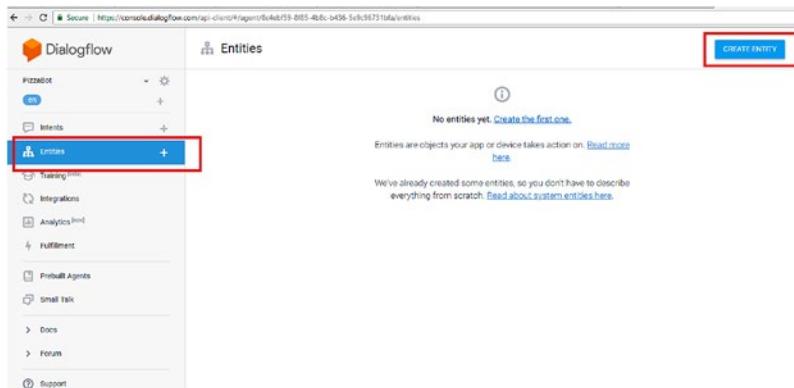


Figure 3-42. Selecting the Entities option

This creates a field for you with the required details. For pizzas, the entity will be Topping, as shown in Figure 3-43.

The screenshot shows the Dialogflow interface. On the left, there's a sidebar with various sections like PizzaBot, intents, entities, Training Data, Integrations, Analytics, Fulfillment, Prebuilt Agents, Small Talk, Docs, and Forum. The 'entities' section is currently selected. In the main area, a form is being filled out for a new entity named 'Topping'. At the top of the form, there are two checkboxes: 'Define synonyms' (which is checked) and 'Allow automated expansion'. Below these, a note says 'Separate synonyms by pressing the enter, tab or ; key.' There are five empty rows for entries, each with a link 'Click here to edit entry'. A blue 'SAVE' button is at the top right, and a small 'X' icon is in the top right corner of the input area.

Figure 3-43. Declare an entity called *Topping*

You will add four toppings for this example:

- Onions
- Mushroom
- Peperoni
- Pineapple

Now save the Topping parameters.. The bot UI will then look like Figure 3-44.

CHAPTER 3 WIT.AI AND DIALOGFLOW

The screenshot shows the Dialogflow interface for creating an entity named 'Topping'. On the left sidebar, under the 'Entities' section, 'Topping' is selected. The main area displays a table with two columns: 'Onions' and 'Mushroom' in the first column, and 'Onions' and 'Mushroom' in the second column. A checkbox labeled 'Define synonyms' is checked, and an unchecked checkbox labeled 'Allow automated expansion' is present. A 'SAVE' button is at the top right, and a 'Click here to edit entry' link is at the bottom right of the table.

Figure 3-44. The Topping options

Deploying the Bot

You have already imported the Small Talk agent. Now you'll import a new agent. Let's name the agent Questionbot. The more agents we add it becomes easier for the conversational flow. You create a Questionbot and now you will try to use prebuilt agents in it, as shown in Figure 3-45.

The screenshot shows the Dialogflow interface for managing intents. Under the 'Intents' section, 'Questionbot' is selected. The main area shows a search bar and two intents listed: 'Default Fallback Intent' and 'Default Welcome Intent'. A red box highlights the 'Prebuilt Agents' option in the sidebar. A note at the bottom states: 'No regular intents yet. Create the first one.' Below that, a detailed note says: 'Intents are mappings between a user's queries and actions fulfilled by your software. [Read more here.](#)' At the bottom, another note says: 'Before you start, check out [Prebuilt Agents](#), a collection of agents developed by the Dialogflow team.'

Figure 3-45. Adding prebuilt agents

You will have to select all the intents available so the bot flow will be perfect; see Figure 3-46.

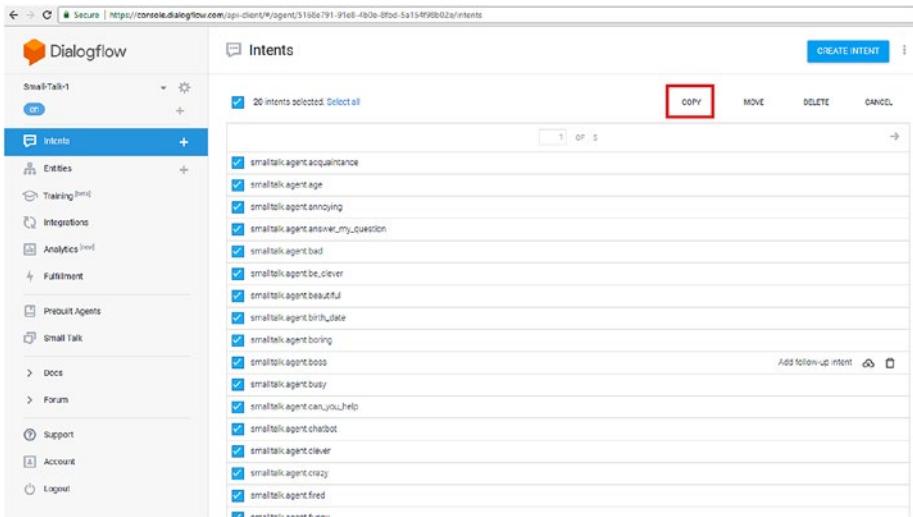


Figure 3-46. The Small Talk agent

Then you need to copy the intents by clicking to put a check mark on the copy-related entities and naming the destination agent (in this case, it is Questionbot). Next, click Start, as shown in Figure 3-47.

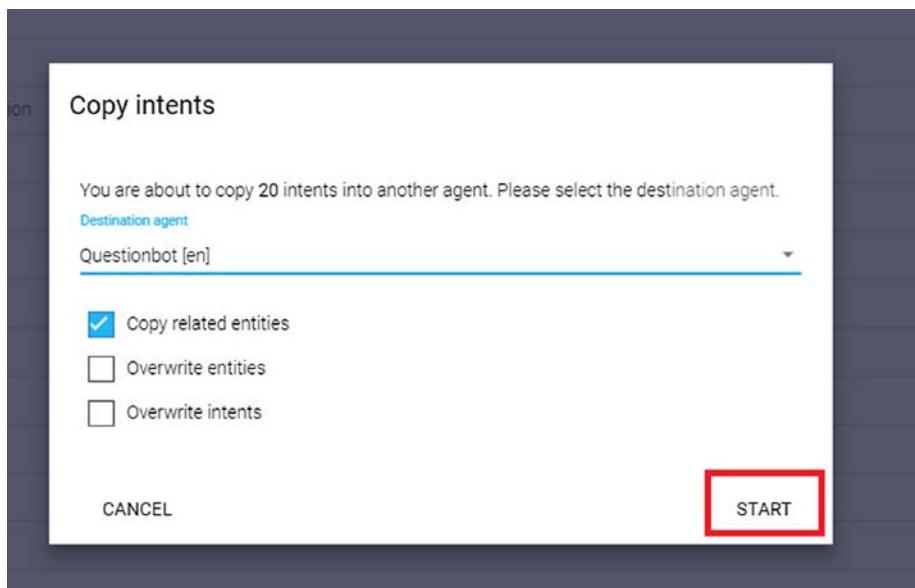


Figure 3-47. Copying related entities

If you now go to the Questionbot, you will see the intents and all of the content in the bot. Try *are you a bot* as a parameter, as shown in Figure 3-48.

Try it now 

 See how it works in [Google Assistant.](#) 

Agent Domains

USER SAYS [COPY CURL](#)

are you a bot

 DEFAULT RESPONSE [▼](#) [PLAY](#)

That's me. I chat, therefore I am.

INTENT

[smalltalk.agent.chatbot](#)

ACTION

[smalltalk.agent.chatbot](#)

[SHOW JSON](#)

Figure 3-48. You ask a question, and the bot answers

Integration with Web Instance

In this section, you will work toward getting the integration for the Web. We will now try to integrate the app to certain platform let's try in Web Demo.

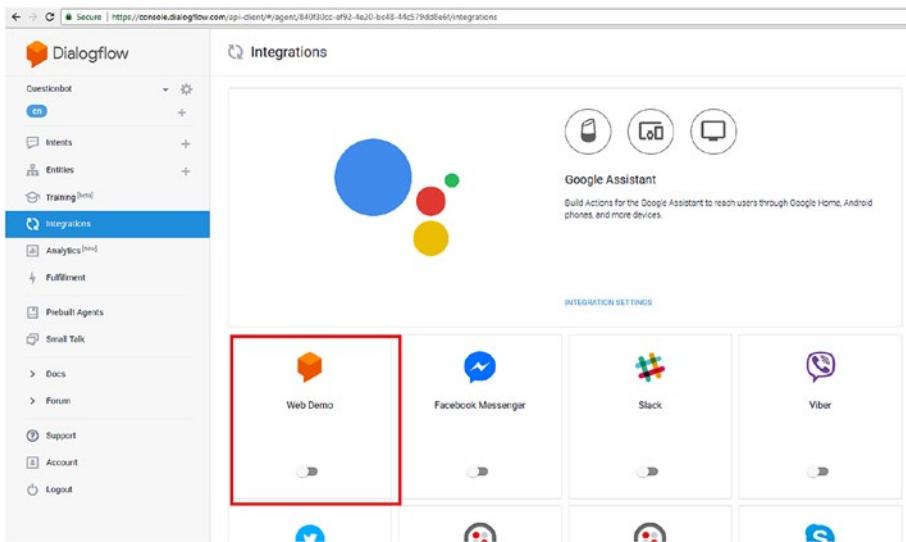


Figure 3-49. Selecting the Web Demo integration option. We are integrating the bot to enable the web version for conversation.

Toggle the slider to right to test the link, as shown in Figure 3-50.

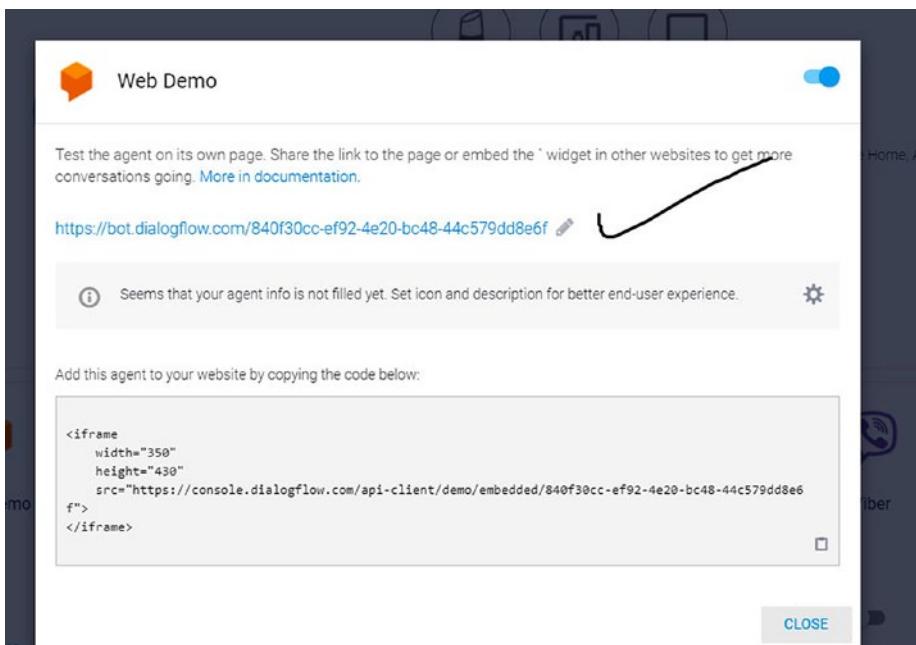


Figure 3-50. Testing the link

Integrate the web link for the bot by using an iframe in an HTML page so the bot works accordingly, shown in Figure 3-51.

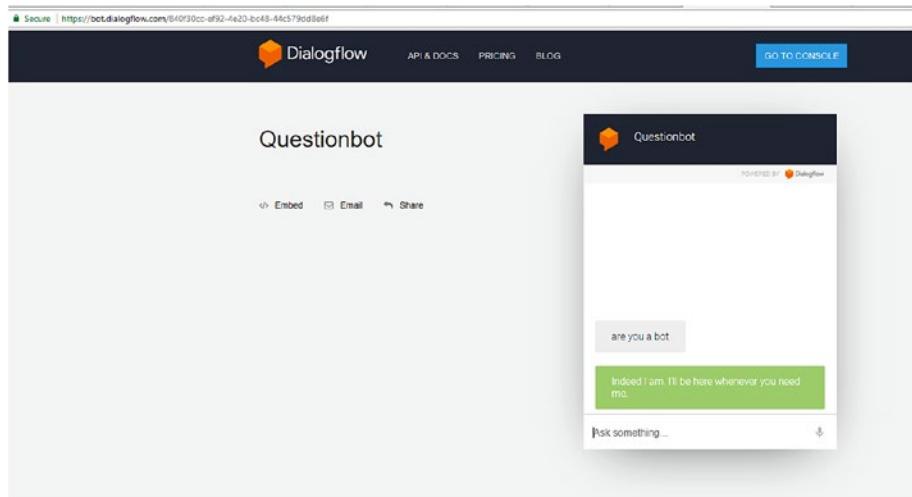


Figure 3-51. Integrating the web link for the bot

Conclusion

In this chapter, you worked on two bot frameworks and learned how they work. You created a complete deployable bot by using Dialogflow and saw its result. This chapter provided simple examples and integrations as well as use cases for bots.

CHAPTER 4

IBM Watson Chatbots

Watson is a computer system specifically built for Q-and-A types of capabilities. Its core implementations use natural language processing, information retrieval, knowledge representation, automated reasoning, and machine-learning technologies in the field of open domain question answering. It is used for quick AI-based solutions.

In this chapter, you will learn how to implement IBM's bot service, known as Watson, for your use (see Figure 4-1). In order to access Watson, you will create an account for the IBM Cloud (formerly known as Bluemix). Then you will use the Watson Assistant (formerly known as Conversational Services) to create two chatbots: a FAQ bot and a coffee bot.

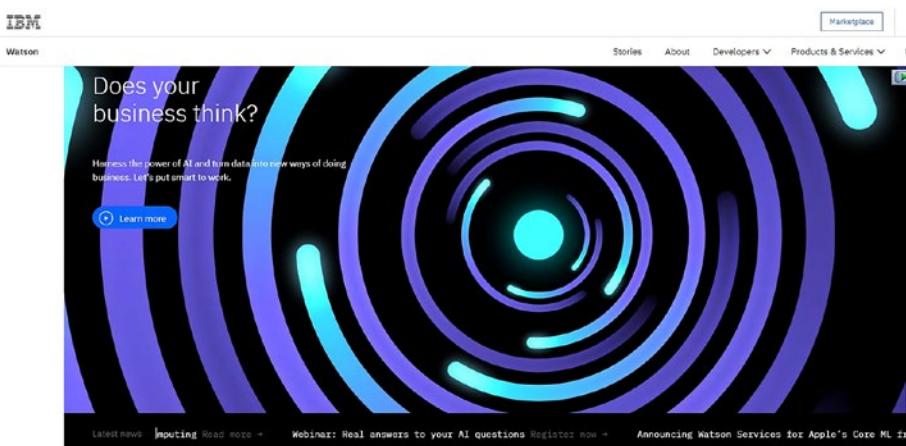


Figure 4-1. The IBM web page for Watson

Implementing Watson

In this section, you'll learn how to access Watson and you'll get to know its various services. You access Watson via the IBM Cloud web site, so you will first set up an account through the IBM Cloud for the Watson service. We will setup the new account.

IBM Cloud

Start by accessing the IBM Cloud by opening a web browser to <https://www.ibm.com/cloud-computing/in-en/>. Figure 4-2 shows the IBM Cloud web site.

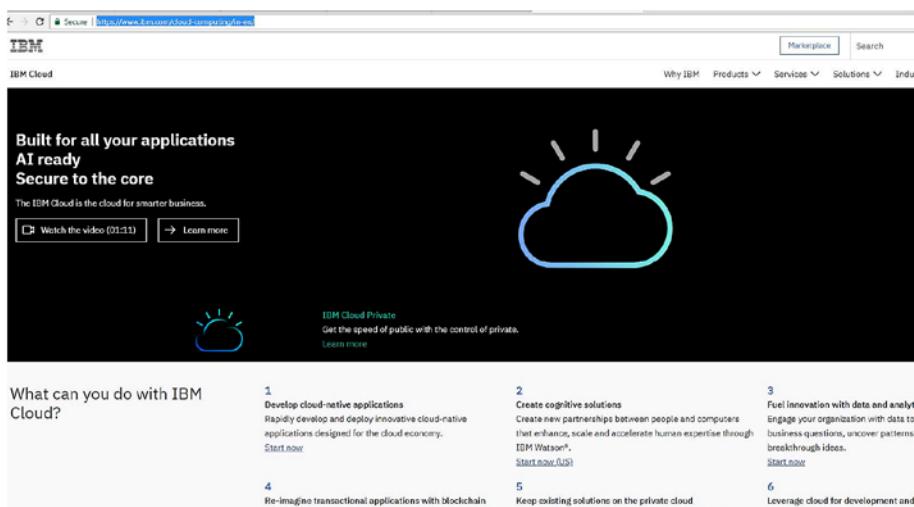


Figure 4-2. IBM Cloud main page

Log in here by using your IBM ID, as shown in Figure 4-3.

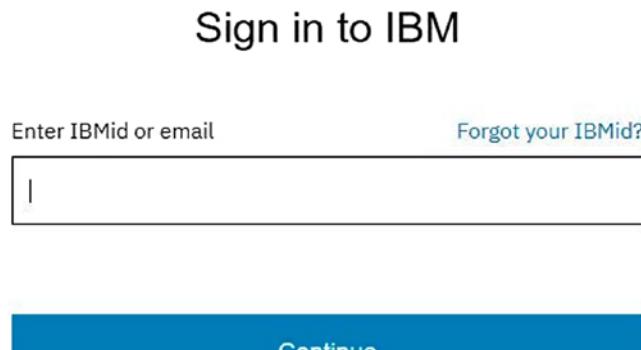


Figure 4-3. IBM Cloud login window

After logging in, you will see the web page shown in Figure 4-4.

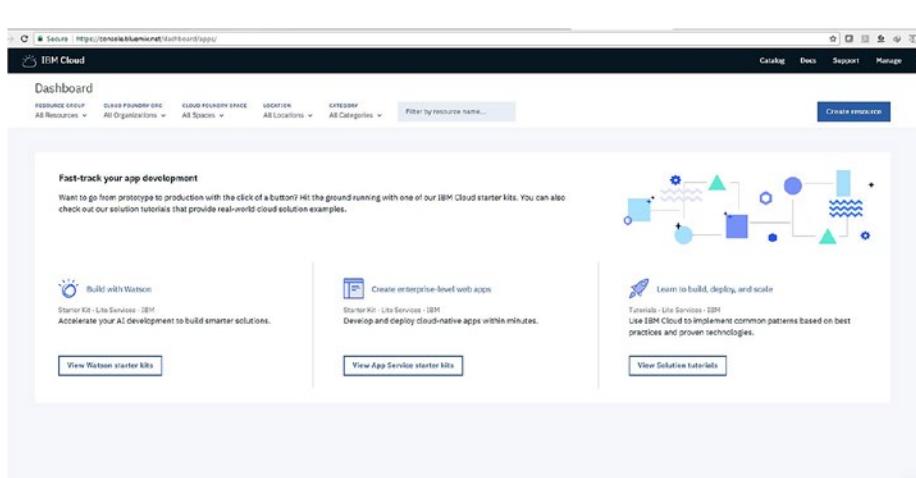


Figure 4-4. The IBM Cloud main console window

CHAPTER 4 IBM WATSON CHATBOTS

On this page, click the Create Resource button at the top right, and shown in Figure 4-5, to create a resource and add new services.

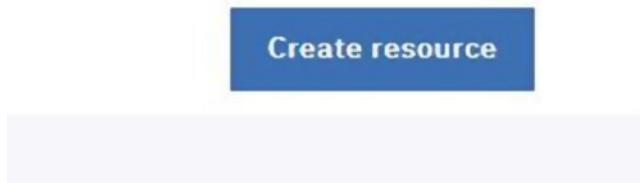


Figure 4-5. Creating a new resource. When we create resource we then will be able to create conversational bot

Watson Assistant Service

In this section, you will see how to set up the Watson Assistant service.

When we create the resource there are lot of options available. You will have to add the Watson Assistant service in it. A lot of services are available for Watson, as you can see in Figure 4-6.

Figure 4-6. Various Watson services available

For this example, you will choose the Watson Assistant, shown in Figure 4-7.

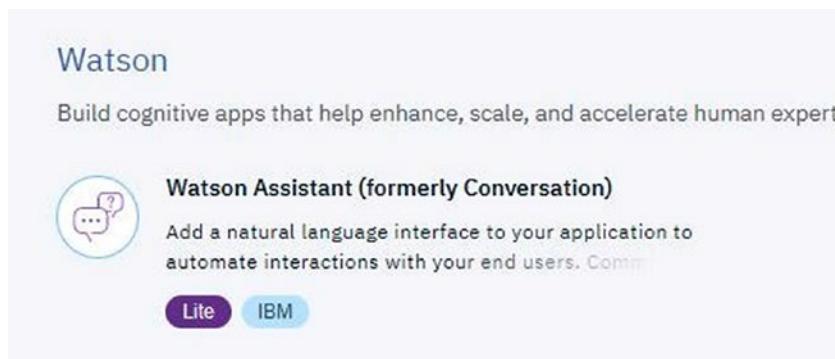


Figure 4-7. Choosing the Watson Assistant service

Next, click the Create button at the bottom right to create the service plan for Watson Assistant this shows the different plans for the watson assistant service and how much you get with free tier, as shown in Figure 4-8.

 A screenshot of the Watson Assistant service plan creation page in the IBM Cloud catalog. At the top, there is a navigation bar with "View Docs", "Icons", "Catalog", "Docs", "Support", "Manage", and a "Feedback" button. Below the navigation bar, there is a section titled "Images" showing three screenshots of the Watson Assistant interface. Then, there is a section titled "Pricing Plans" with a table showing the following details:

PLAN	FEATURES	PRICE
Lite	10,000 API Calls per Month* Up to 5 Workspaces Up to 100 Entities Up to 25 Entities Single message method calls only	Free
Standard	Unlimited API queries per month Up to 25 Workspaces Up to 100 Entities Up to 1000 Entities Shared Public Cloud Single message method calls only	\$0.10/1000 API calls*
Premium	Premium Plans offer developers and organizations a single tenant instance of use or more Watson services for better isolation and security. These plans offer computer-level isolation on the existing shared platform, as well as reinforced encryption data	

 At the bottom of the page, there are buttons for "Need Help?", "Contact IBM Cloud Sales", "Estimate Monthly Cost", "Cost Calculator", "Create", and a "Feedback" button.

Figure 4-8. The Watson Assistant service plan

The setup is easy. We need to click on create and the setup process will move towards completion. After setup is complete, click the Launch tool for the service to start.

Creating a FAQ Bot

In this section, you will use the Watson Assistant to create a chatbot that uses FAQ for answering. Clicking the Launch tool in the preceding step has opened the IBM Watson Assistant screen, shown in Figure 4-9. Click the Create a Workspace button at the bottom of the screen to get started.

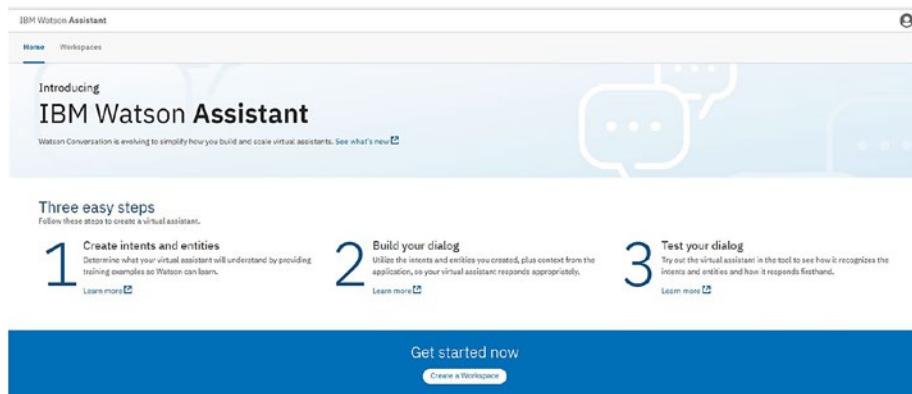


Figure 4-9. Getting started with the IBM Watson Assistant

Now you will create the workspace. Click the Workspaces tab and then click the Create button in the Create a New Workspace section, as shown in Figure 4-10.

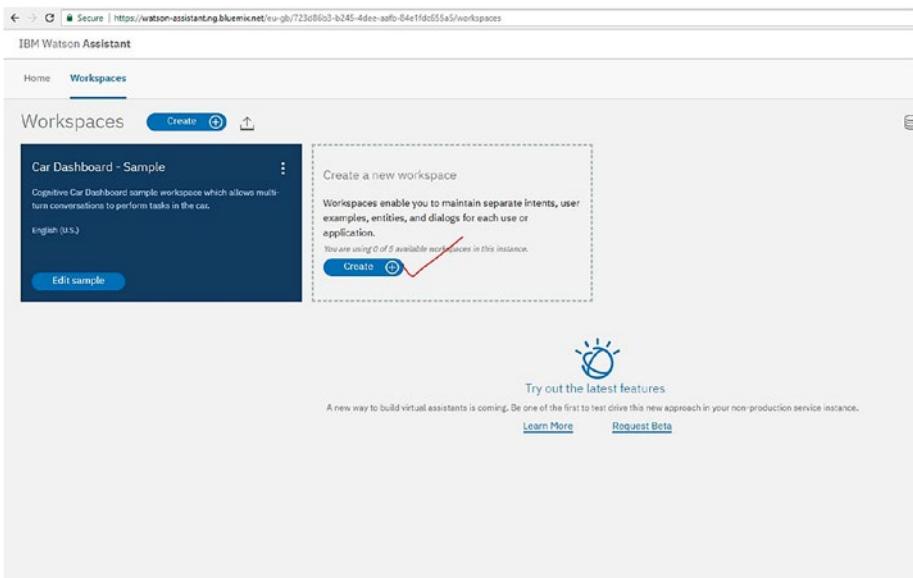


Figure 4-10. Creating the workspace

On the next screen, give the workspace name; in this example, type in **FAQBot** and then type in a description, as shown in Figure 4-11. Then click the Create button.

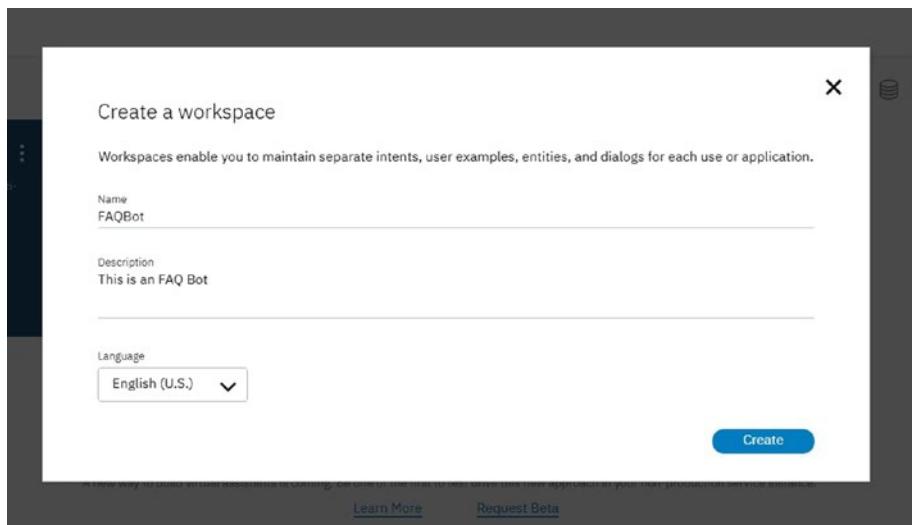


Figure 4-11. Naming the workspace

Creating Intent for the Bot

The next step is to create intents for the bot according to your needs. We need to click on add intent. At this point, the Watson console looks like Figure 4-12.

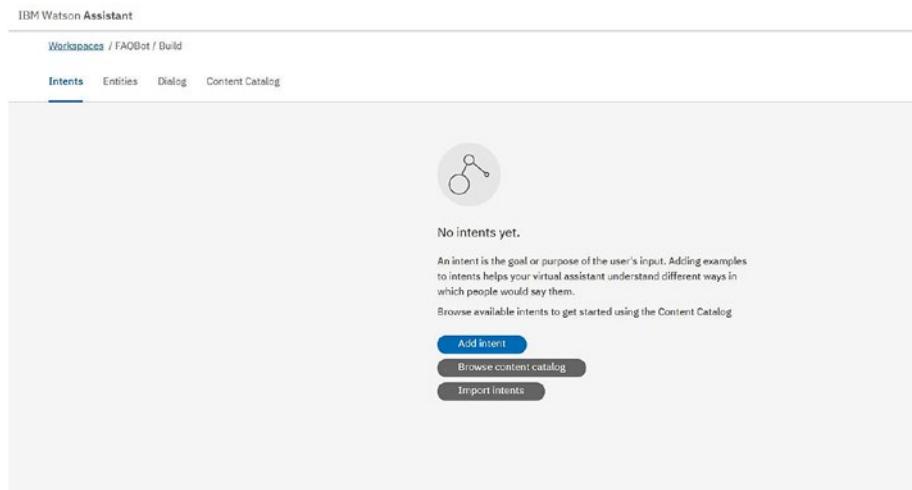


Figure 4-12. Creating a new intent

Our FAQBot will be based on IBM Business process on Cloud. Name the first intent *Capabilities*, as shown in Figure 4-13.



Figure 4-13. We now add the name of the intent as *Capabilities* in the intent name option. Naming the intent.

Next, you need to add a description of the intent, as shown in Figure 4-14.

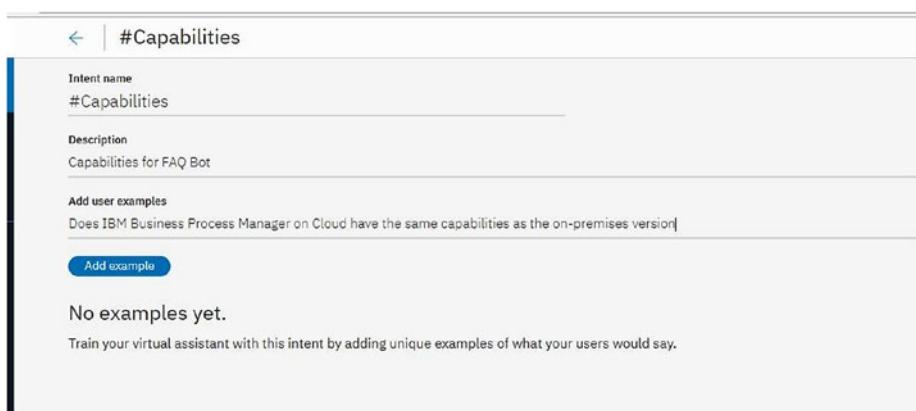


Figure 4-14. Adding a description for the intent

CHAPTER 4 IBM WATSON CHATBOTS

Then add examples to the Capabilities intent, as shown in Figure 4-15.

The screenshot shows the Watson Assistant interface for creating a new intent named '#Capabilities'. The left sidebar has icons for Home, Create, and Settings. The main area has a header 'Secure | https://watson-assistant.ng.bluemix.net/eu-gb/723d86b3-b245-4dee-aafb-84e1fd055a5/workspaces/f49ccca9-ec24-4980-bba3-fad307b6f862/build/intents'. Below the header is a title '#Capabilities'. The 'Intent name' field contains '#Capabilities'. The 'Description' field contains 'Capabilities for FAQ Bot'. Under 'Add user examples', there is a link 'Add user examples to this intent' and a 'Add example' button. A section titled 'User examples (2)' lists two examples: 'on premises' and 'Does IBM Business Process Manager on Cloud have the same capabilities as the on-premises version'. Both examples have checkboxes next to them.

Figure 4-15. Adding user examples for the Capabilities intent

Now you'll create a second intent. Name this intent *Migration*, as shown in Figure 4-16.

The screenshot shows the Watson Assistant interface for creating a new intent named '#Migration'. The left sidebar has icons for Home, Create, and Settings. The main area has a header 'Secure | https://watson-assistant.ng.bluemix.net/eu-gb/723d86b3-b245-4dee-aafb-84e1fd055a5/workspaces/f49ccca9-ec24-4980-bba3-fad307b6f862/build/intents'. Below the header is a title '#Migration'. The 'Intent name' field contains '#Migration'. The 'Description' field contains 'Add a description to this intent'. Under 'Add user examples', there is a link 'Add user examples to this intent' and a 'Add example' button. A section titled 'User examples (1)' lists one example: 'Can I migrate from IBM Business Process Manager running on-premises to IBM Business Process Manager on Cloud'. This example has a checkbox next to it.

Figure 4-16. Creating a new intent named Migration

Create a third intent and name it *User*, as shown in Figure 4-17.

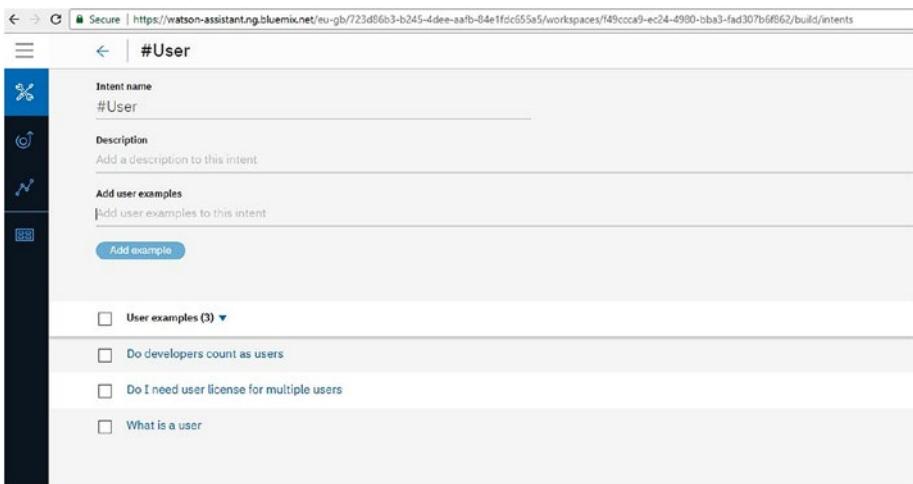


Figure 4-17. Creating a third intent named *User*

Now create a fourth intent named SSO, which stands for single sign-on, as shown in Figure 4-18.



Figure 4-18. Creating the SSO intent

Now you have all the intents listed, as shown in Figure 4-19.

The screenshot shows the IBM Watson Assistant interface with the 'Workspaces / FAQBot / Build' path at the top. The left sidebar has icons for workspace, entities, dialog, and catalog. The main navigation bar includes 'Intents', 'Entities', 'Dialog', and 'Content Catalog'. Below this is a toolbar with 'Add intent' and other icons. A table lists four intents under the heading 'Intent (4)'. The first intent is '#Capabilities' with a description 'Capabilities for FAQ Bot'. The other three intents are '#Migration', '#SSO', and '#User', each with an empty description field.

Intent (4)	Description
#Capabilities	Capabilities for FAQ Bot
#Migration	
#SSO	
#User	

Figure 4-19. All intents listed

The Dialog Flow for the App

In this section, you will create the dialogs for the FAQBot. Click the Dialog tab and then click the Create button, as shown in Figure 4-20.

The screenshot shows the IBM Watson Assistant interface with the 'Workspaces / FAQBot / Build' path at the top. The left sidebar has icons for workspace, entities, dialog, and catalog. The main navigation bar includes 'Intents', 'Entities', 'Dialog', and 'Content Catalog'. The 'Dialog' tab is selected, indicated by a blue underline. Below the tabs is a large circular icon with a speech bubble and three dots. To its right, the text 'No dialog yet' is displayed. A descriptive note below states: 'A dialog uses intents, entities, and context from your application to define a response to each user's input. Creating a dialog defines how your virtual assistant will respond to what your users are saying.' At the bottom is a blue 'Create' button with a plus sign icon.

Figure 4-20. The dialog creation space. Here we will be creating the dialog flow for the bot created we need click on create.

On the next screen, click Welcome so that we can start the conversation with a greeting over here, as shown in Figure 4-21.

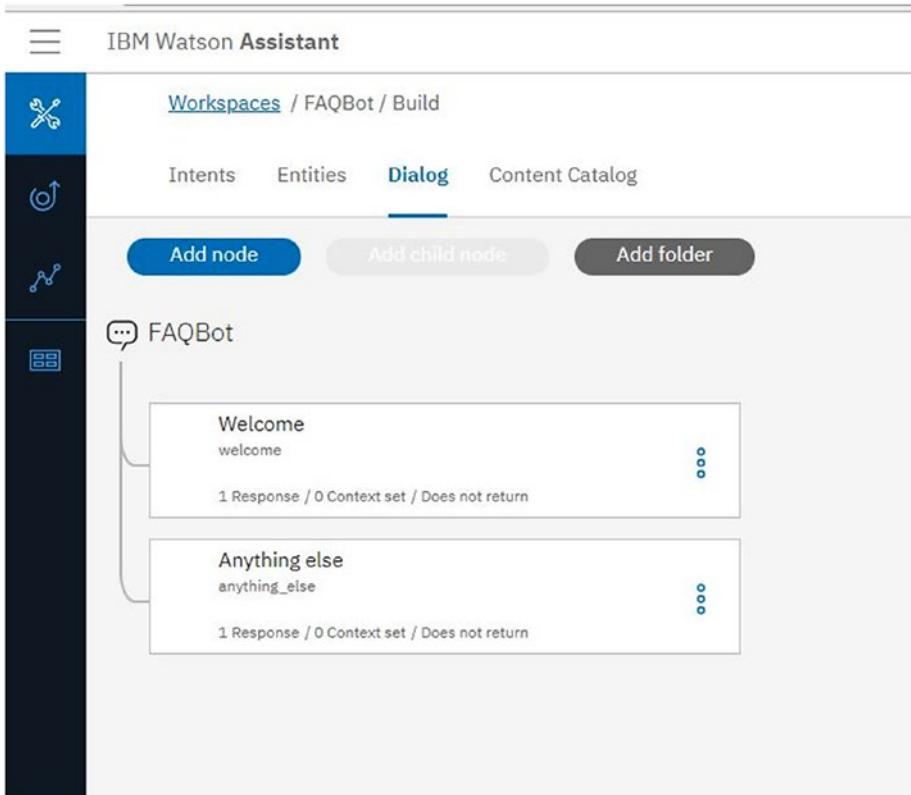


Figure 4-21. The dialog workflow

Making Sense of the Flow

Now that you've created various dialogs with creation of dialogs we are linking the flow of the bot to work efficiently, you're ready to create interactions in this section by linking up the entire flow. You'll start by adding an intention for what you want to do.

xxxxxxxxxxxxxxxxxx, as shown in Figure 4-22.

CHAPTER 4 IBM WATSON CHATBOTS

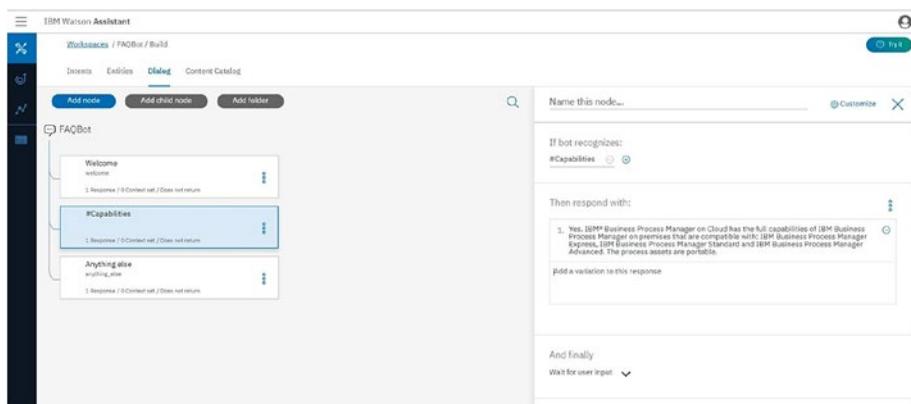


Figure 4-22. Creating a new node we want to touch on every information to be touched upon and also the flow to be perfect.

Next, add other intentions for the Migration intent, as shown in Figure 4-23.

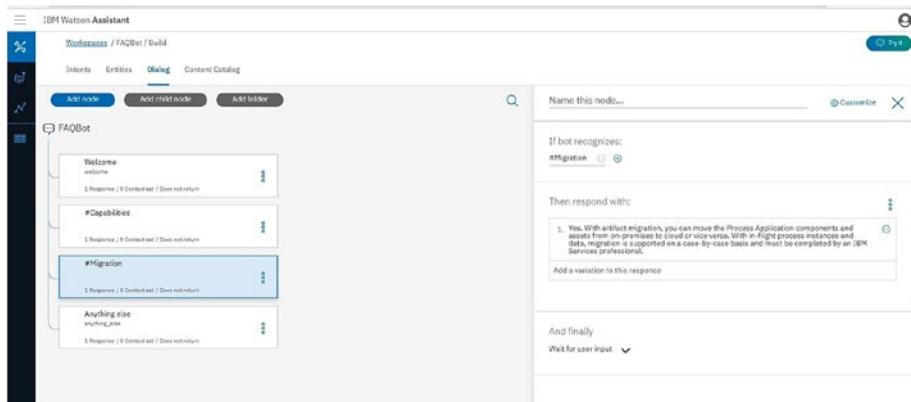


Figure 4-23. Adding a new node for the Migration intent

Add an intention for the User intent, as shown in Figure 4-24.

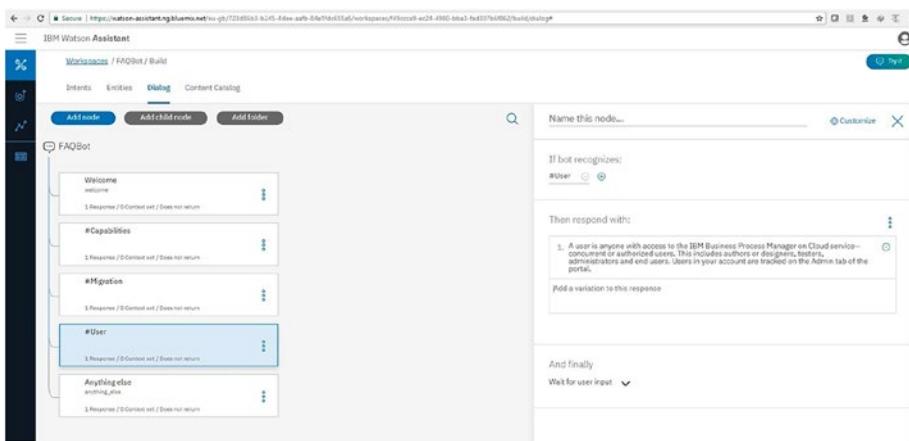


Figure 4-24. Adding new intent User

Now add a single sign-on, as shown in Figure 4-25.

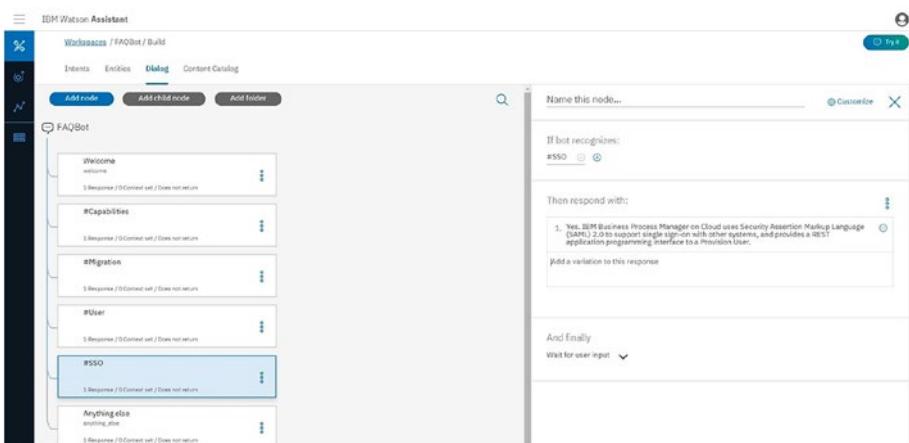


Figure 4-25. Adding nodes for SSO

Trying the Bot

In this last section for our first bot, you will see how to interact with the bot. Click the Try It button at the top right to access a Try It Out link, shown in Figure 4-26. Let's try out the bot now.

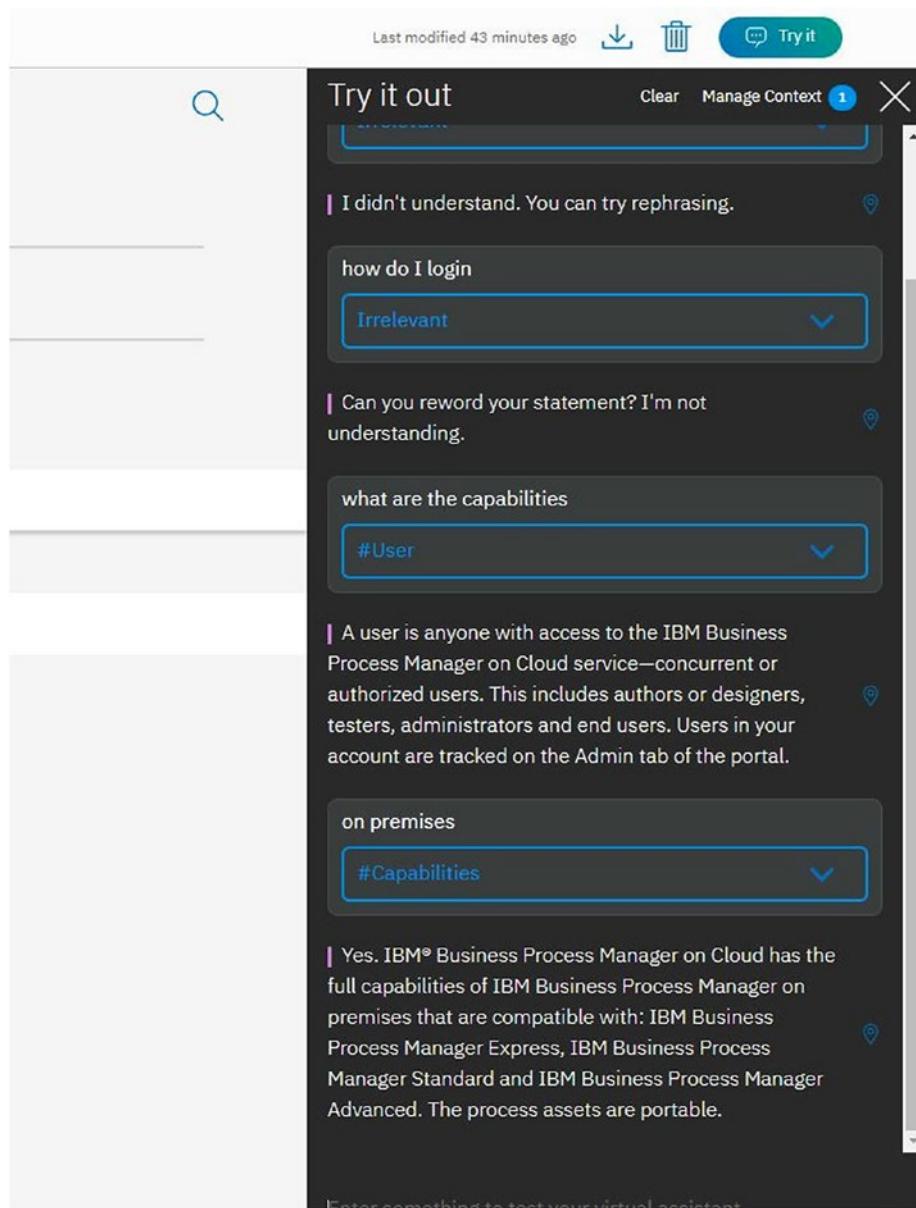


Figure 4-26. Trying the bot. Here we are just testing the bot how it works and the flow logic is perfect or not.

In the following section, you will create another bot. This one will help the user order coffee.

Creating a Coffee Bot

In this section, you will create a CoffeeBot with Watson Assistant. This bot will help the user order a cup of coffee. First you will create a workspace; then you will create new intents and add entities. Next, you will add dialogs. Finally, you'll create a nested structure for the intents.

Creating a Workspace

To get started, you will create a new workspace. The workspace is created from watson assistant window and for the new bot creation this step is taken, as shown in Figure 4-27.

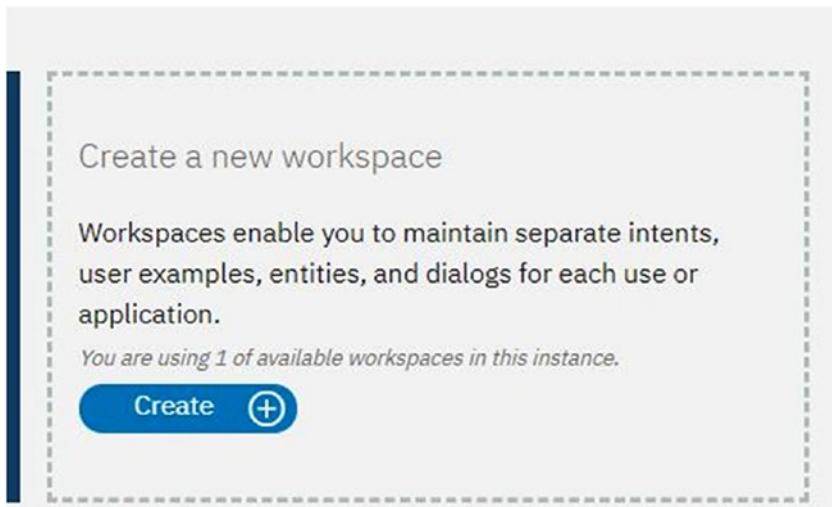


Figure 4-27. Creating a new workspace for the coffee bot

On the screen that opens, name the chatbot *CoffeeBot*, as shown in Figure 4-28. Then click the Create button.

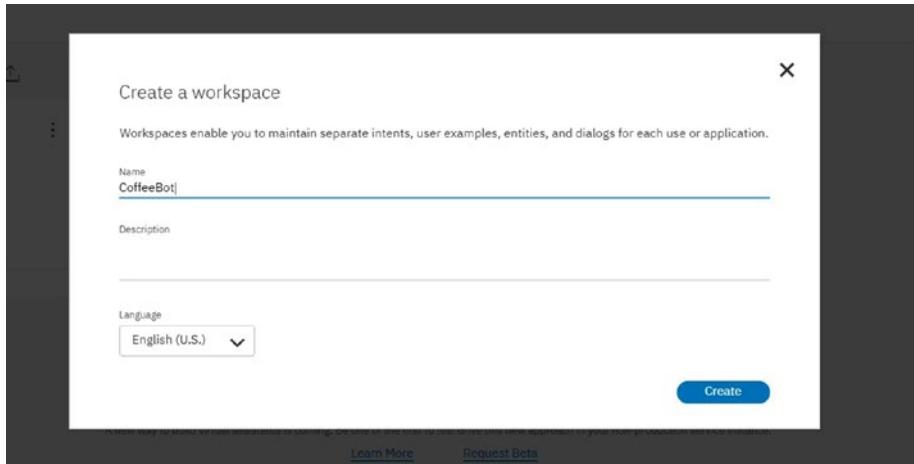


Figure 4-28. Naming the workspace

Working with Intents

In this section, you will start creating intents for the usage of our bot. Click the Intents tab and then click the Add Intent button, as shown in Figure 4-29.

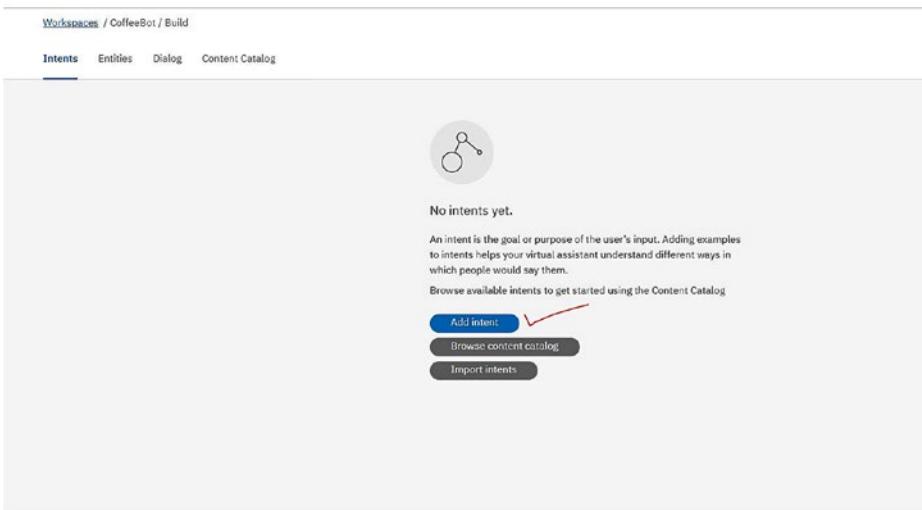


Figure 4-29. Creating the intent for the CoffeeBot

Let's start by creating an intent named Greetings, as shown in Figure 4-30.

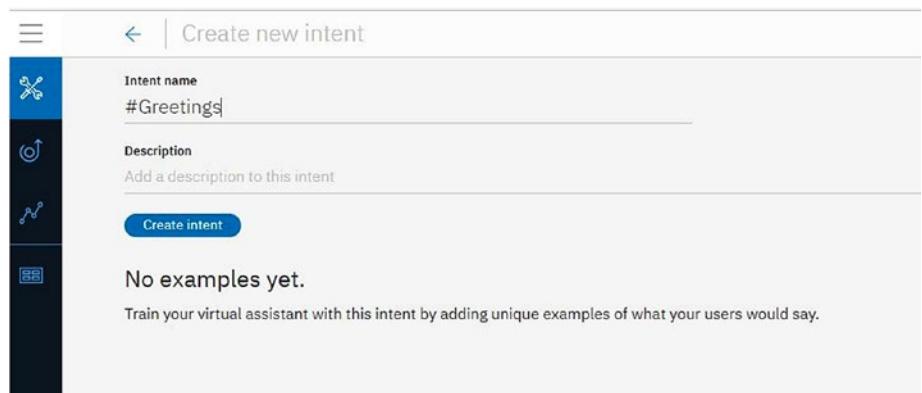


Figure 4-30. Naming the intent Greetings

There are different ways that the bot can greet a user. You'll use this screen to choose from the options, as shown in Figure 4-31.

The screenshot shows the IBM Watson Assistant interface for configuring an intent named '#Greetings'. On the left is a sidebar with icons for intents, entities, actions, and more. The main panel has tabs for 'Intents' (selected), 'Entities', 'Actions', and 'Logs'. The '#Greetings' intent card is displayed with the following fields:

- Intent name:** #Greetings
- Description:** Add a description to this intent.
- Add user examples:** Add user examples to this intent.
- Add example:** A button to add more examples.

Below these fields is a section titled 'User examples (4) ▾' containing four entries, each with a checkbox:

- Hello
- Hi
- Howdy
- Yo

Figure 4-31. Adding user examples

Next, you'll add an intent named BuyCoffee, as shown in Figure 4-32.

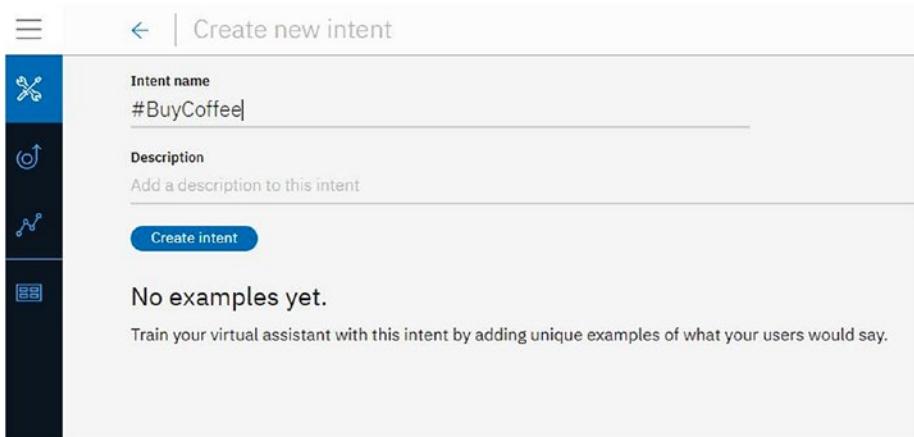


Figure 4-32. Creating a new intent named BuyCoffee

Add some examples for it, as shown in Figure 4-33.

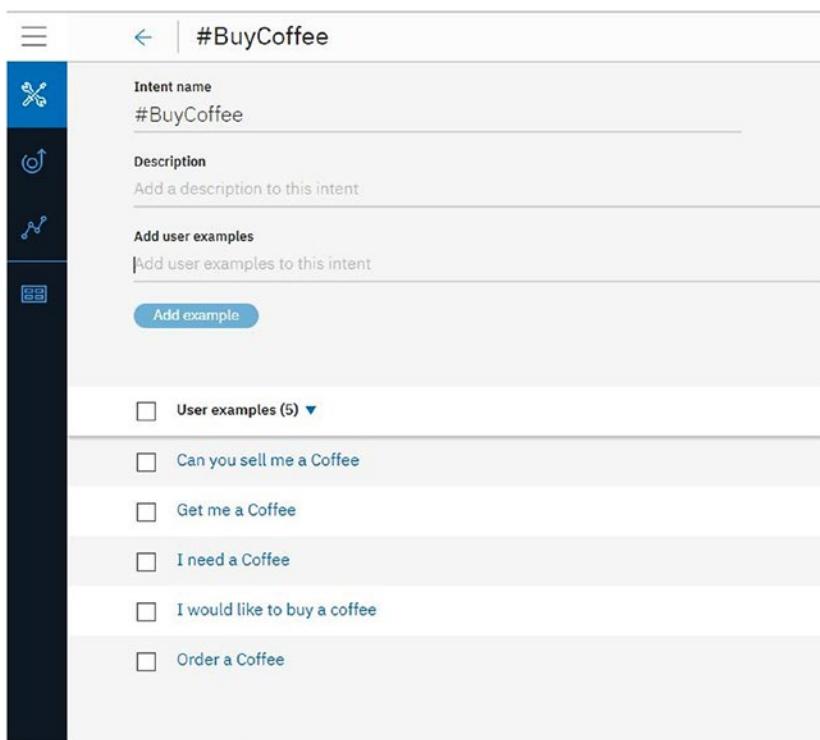


Figure 4-33. Adding user examples

Create another intent and name it Suggestion, as shown in Figure 4-34.

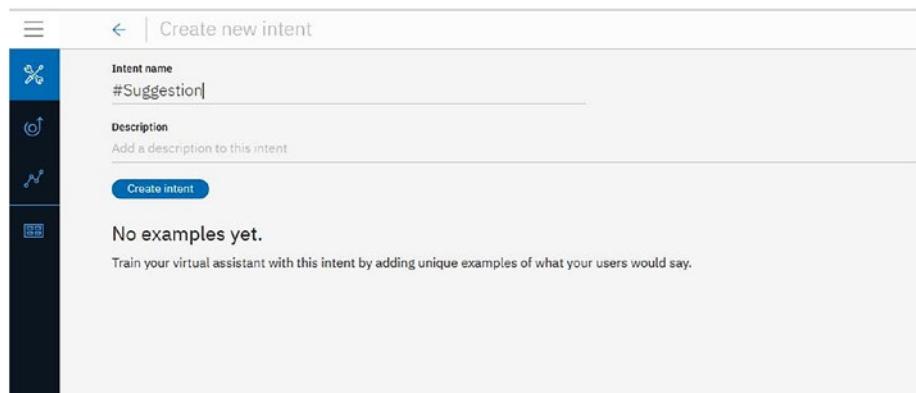


Figure 4-34. Creating a new intent named Suggestion

Add some examples to this new intent, as shown in Figure 4-35.

The screenshot shows the IBM Watson Assistant interface for creating a new intent named '#Suggestion'. The left sidebar has icons for Home, Create, Import, and Settings. The main area shows the intent name '#Suggestion' and a description field containing 'Add a description to this intent'. Below that is a section for 'Add user examples' with a placeholder 'Add user examples to this intent' and a blue 'Add example' button. A list of 7 user examples is displayed, each with a checkbox:

- Can you suggest me
- Help me pick one please
- Suggestions please
- What are my options
- What can i order here
- What do you sell
- What options do i have

Figure 4-35. Adding user examples for Suggestion

Next, add another intent named Yes, as shown in Figure 4-36. As you did previously for the other intents, add user examples.

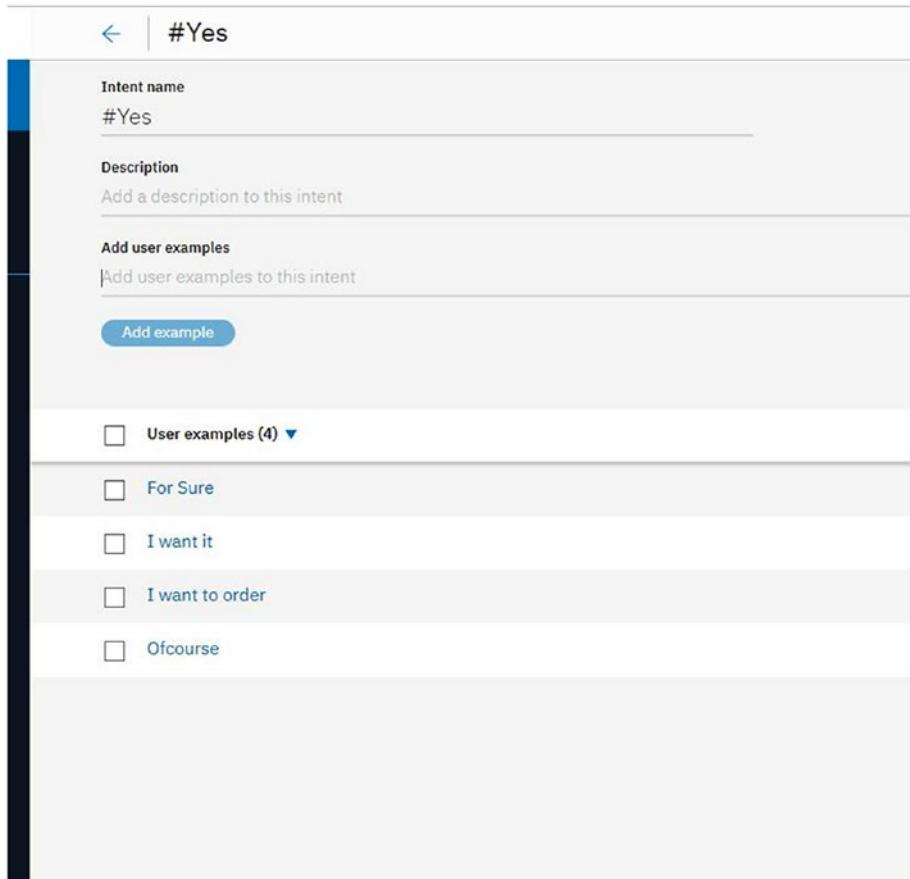


Figure 4-36. Adding user examples for the Yes intent

ThankYou will be the next intent, as shown in Figure 4-37. Name this intent and add examples.

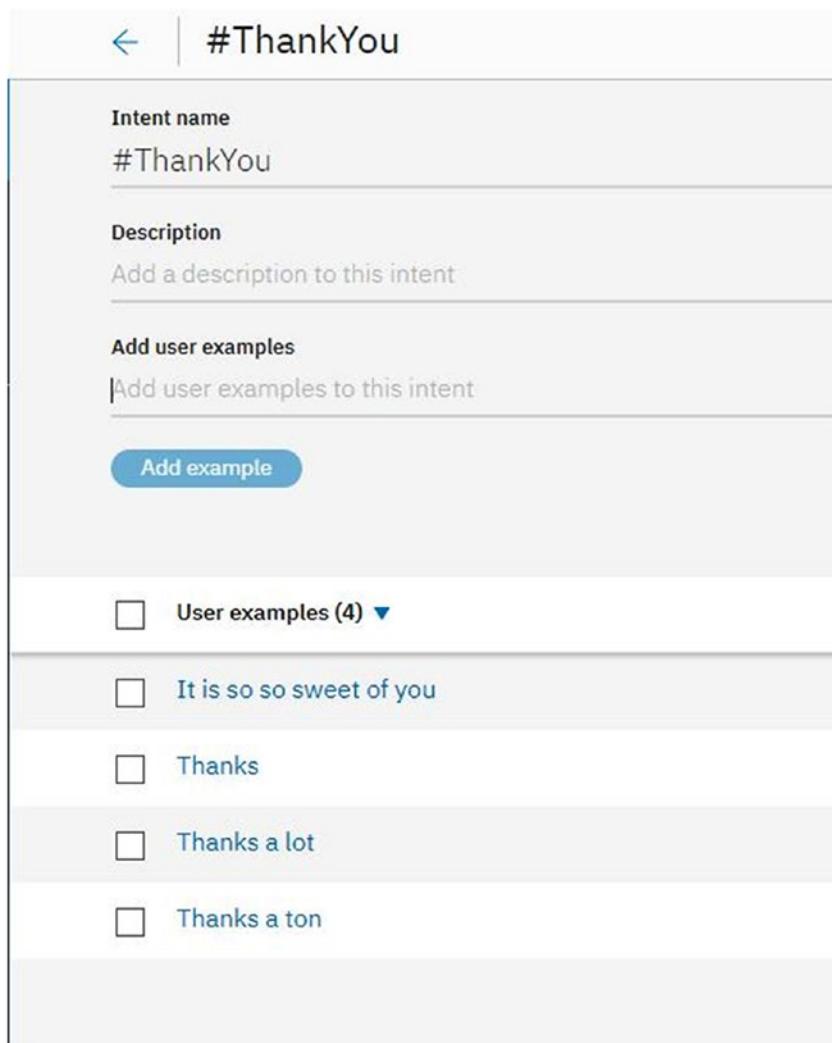


Figure 4-37. *ThankYou intent and user examples*

The next intent will be Cancel, as shown in Figure 4-38. Follow the same process to name it and add examples.

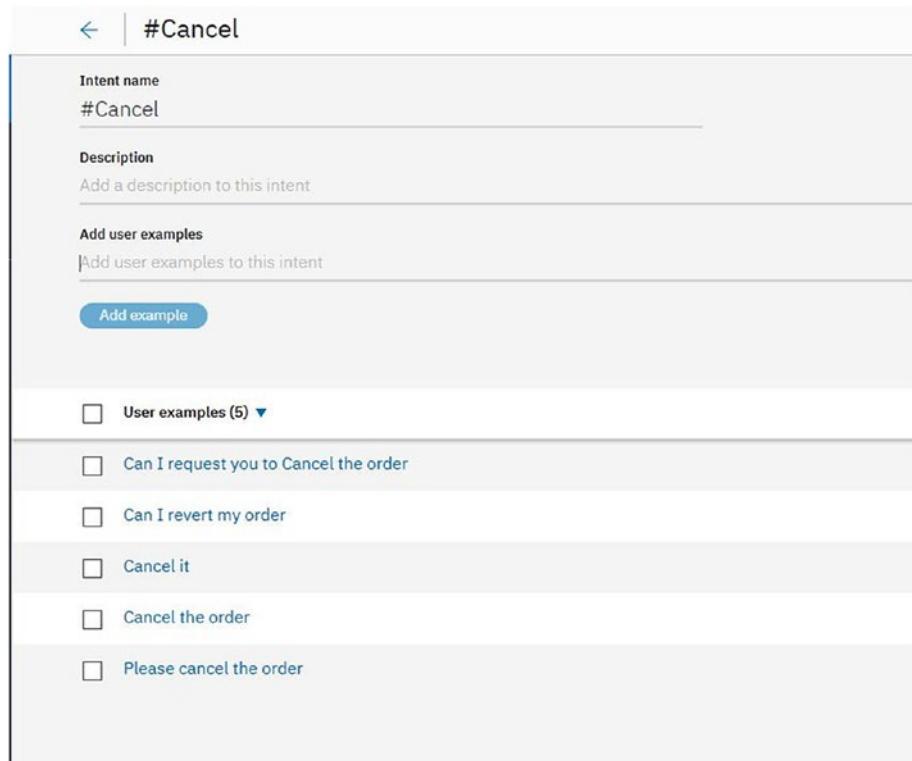


Figure 4-38. Adding the Cancel intent

Working with Entities

In this section, you will work with entities to get the bot more structured. Click the Entities tab and then My Entities to access the screen shown in Figure 4-39. Click the Add Entity button.

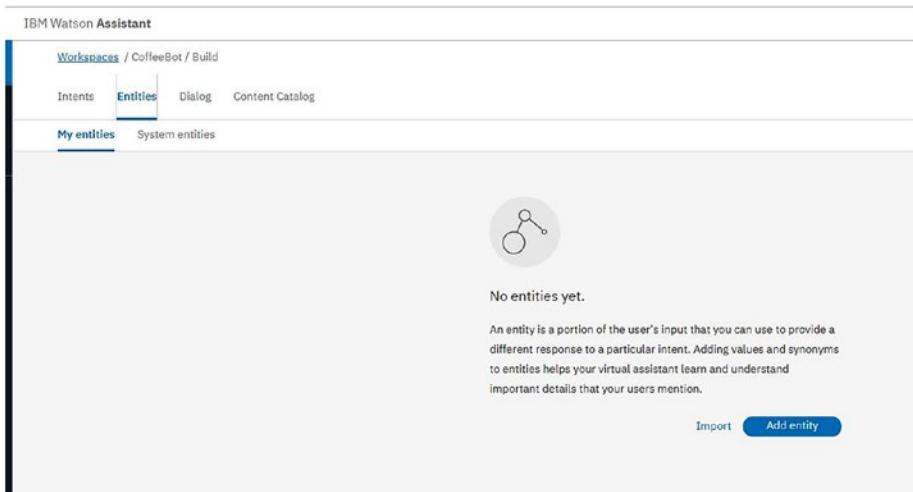


Figure 4-39. Adding entities

On the screen that opens, name the entity by typing *CoffeeSize*, as shown in Figure 4-40. Then click the Create Entity button.

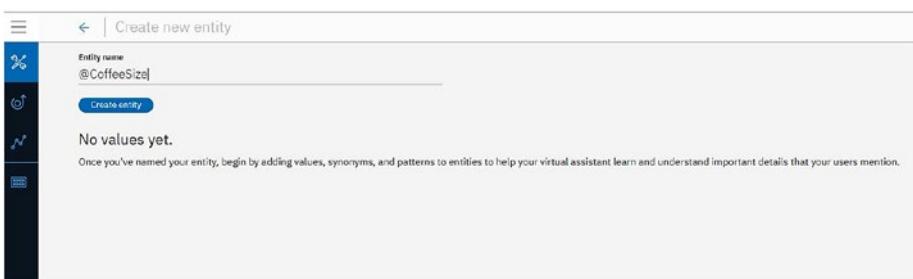


Figure 4-40. Naming the entity *CoffeeSize*

On the next screen, choose values for *CoffeeSize*, as shown in Figure 4-41.

CHAPTER 4 IBM WATSON CHATBOTS

The screenshot shows the configuration of a new entity named '@CoffeeSize'. The 'Entity name' field contains '@CoffeeSize'. Below it, there is a table for adding 'Entity values'. The first row shows 'Large' as the value name, with 'Synonyms' like 'Big'. The second row shows 'Medium' with 'Synonyms' like 'Regular, Not too big'. The third row shows 'small' with 'Synonyms' like 'Low Price, Cheap, Cheapest, Smallest, Tiny'. Each row has a checkbox next to it and a 'Type' column.

<input type="checkbox"/>	Entity values (3) ▾	Type
<input type="checkbox"/>	Large	Synonyms Big
<input type="checkbox"/>	Medium	Synonyms Regular, Not too big
<input type="checkbox"/>	small	Synonyms Low Price, Cheap, Cheapest, Smallest, Tiny

Figure 4-41. Adding values for the CoffeeSize entity

Now add an entity named here we select the different sizes that are available for CoffeeOptions, as shown in Figure 4-42.

The screenshot shows the configuration of a new entity named '@CoffeeOptions'. The 'Entity name' field contains '@CoffeeOptions'. Below it, there is a table for adding 'Entity values'. The first row shows 'Black' as the value name, with 'Synonyms' like 'standard black, roasted black, dark black'. The second row shows 'Cappuccino' with 'Synonyms' like 'capuchino, cappuchino, capuccino'. The third row shows 'mocha' with 'Synonyms' like 'mocha, cafe mocha, mocc, cafe mocha'. The fourth row shows 'Vanilla Latte' with 'Synonyms' like 'Latte, Vanilla, Vanilla Latte'. Each row has a checkbox next to it and a 'Type' column.

<input type="checkbox"/>	Entity values (4) ▾	Type
<input type="checkbox"/>	Black	Synonyms standard black, roasted black, dark black
<input type="checkbox"/>	Cappuccino	Synonyms capuchino, cappuchino, capuccino
<input type="checkbox"/>	mocha	Synonyms mocha, cafe mocha, mocc, cafe mocha
<input type="checkbox"/>	Vanilla Latte	Synonyms Latte, Vanilla, Vanilla Latte

Figure 4-42. Adding synonyms for CoffeeOptions

Working with Dialogs

In this section, you will work with dialogs that will create a flow for seamless interactions between the bot and the user.

From the XXXX screen, do XXXXX to add a new dialog, as shown in Figure 4-43.

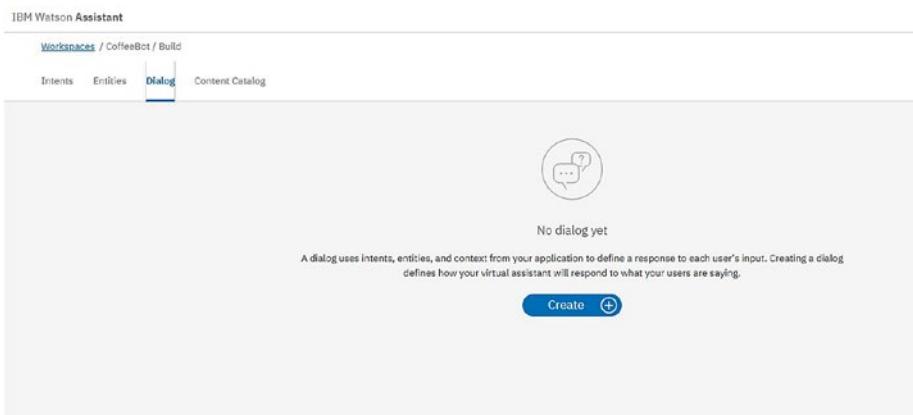


Figure 4-43. Adding a new dialog

After creating the dialog, the workspace looks like Figure 4-44.

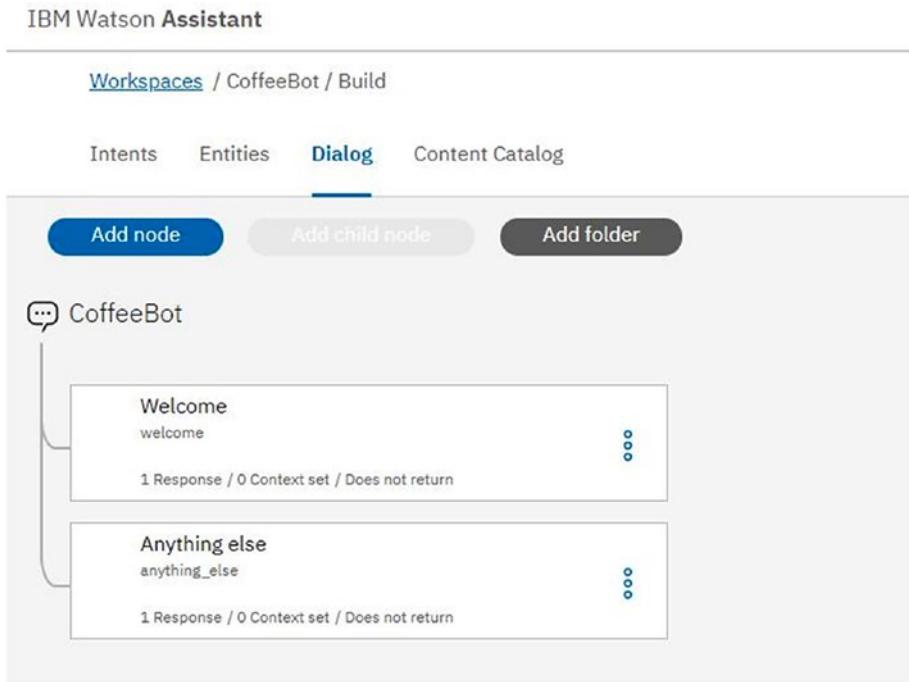


Figure 4-44. Adding the dialog flow for the bot

You will have to add additional dialogs for the bot so that you can bring the things together. In this place itself we add different logic so that the bot works properly. You've already created intents as well as entities for ordering coffee, so now you can use them to create the workflow.

Let's first add a node where we are requesting for buying a coffee, as shown in Figure 4-45.

[Workspaces](#) / CoffeeBot / Build

Intents Entities **Dialog** Content Catalog

Add node Add child node Add folder

...

CoffeeBot

The screenshot shows the IBM Watson Assistant Dialog builder interface. At the top, there are tabs for Intents, Entities, Dialog (which is selected), and Content Catalog. Below the tabs are three buttons: Add node (blue), Add child node (grey), and Add folder (grey). The main area is titled "CoffeeBot". It contains three dialog nodes:

- Welcome**:
welcome
1 Response / 0 Context set / Does not return
- #BuyCoffee**:
#BuyCoffee
1 Response / 0 Context set / Does not return
- Anything else**:
anything_else
1 Response / 0 Context set / Does not return

Each node has a three-dot menu icon on its right side.

Figure 4-45. Adding a new node for BuyCoffee

The content and response for the bot is shown in Figure 4-46.

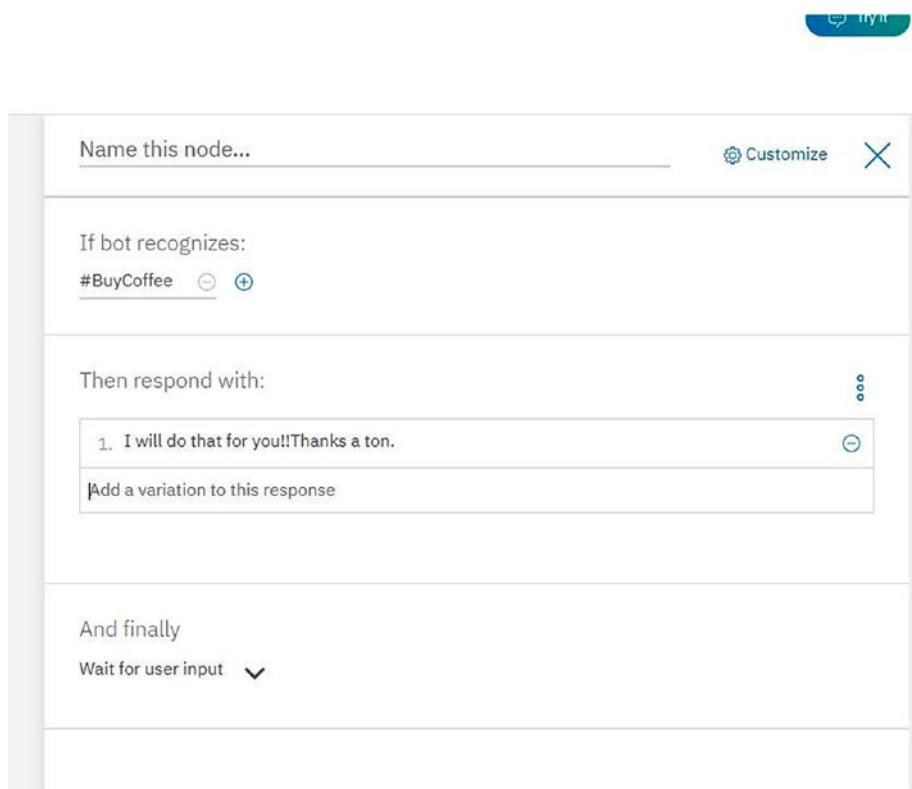


Figure 4-46. Adding a response for BuyCoffee

Next, add a new Suggestion node for the CoffeeBot, as shown in Figure 4-47. In this node, you will create child nodes to indicate preferences for different coffee types.

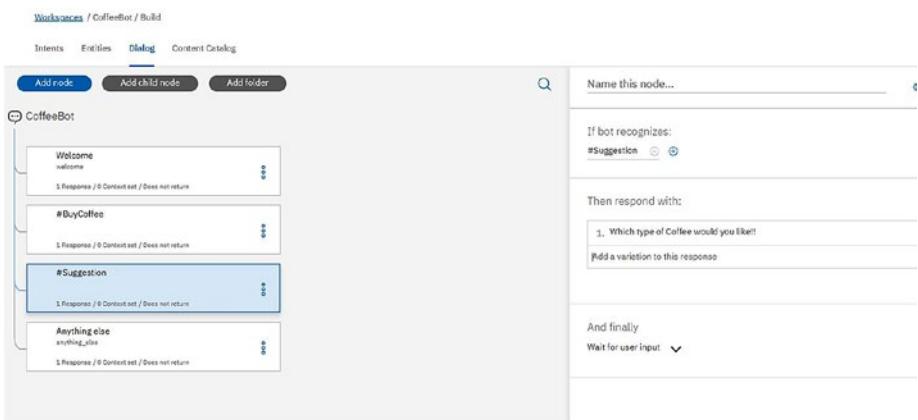


Figure 4-47. Adding a new Suggestion node

Nested Intents

This section covers how to use nested intents so that you can match up the flow that is how the bot will work accordingly as we have meant it to be for the bot.

To the right of each node name, you can see three vertical dots, as shown in Figure 4-48. These dots are used to create a nested child workflow. XXXXXXXXXXXX

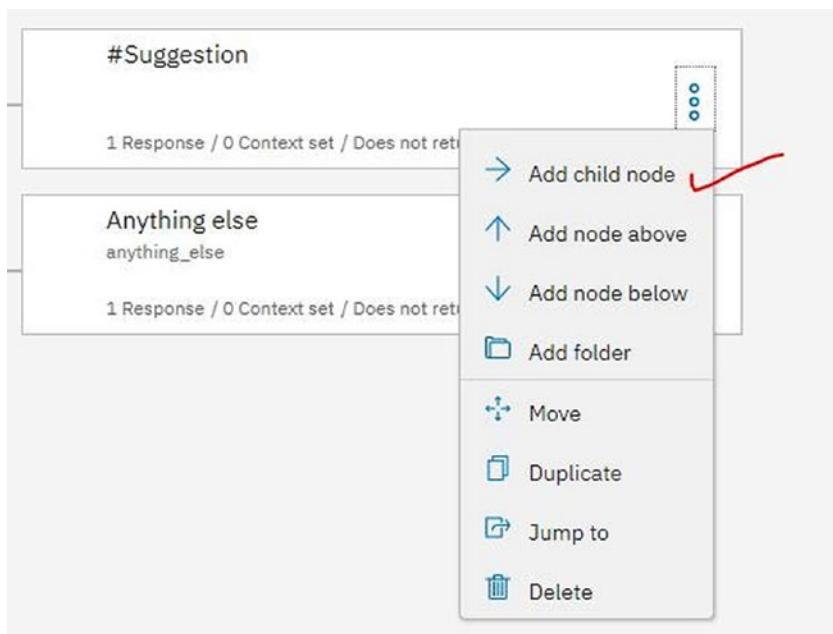


Figure 4-48. Adding a new child node

You can now see the child node you've created for the coffee options, as shown in Figure 4-49.

The screenshot shows the IBM Watson Assistant interface with the 'Dialog' tab selected. On the left, the tree view shows four intents: 'Welcome', '#BuyCoffee', 'afSuggestion', and '@CoffeeOptions'. The '@CoffeeOptions' intent is currently selected and expanded, showing its single response card. On the right, there is a configuration panel for this intent. It includes fields for 'Name this node...', 'If bot recognizes:' (with a condition for '@CoffeeOptions'), 'Then respond with:' (containing a response card for 'We have different flavors'), and 'And finally' (containing a 'Wait for user input' field). The '@CoffeeOptions' intent is also highlighted with a blue selection bar.

Figure 4-49. Adding nested intent

, as shown in Figure 4-50.

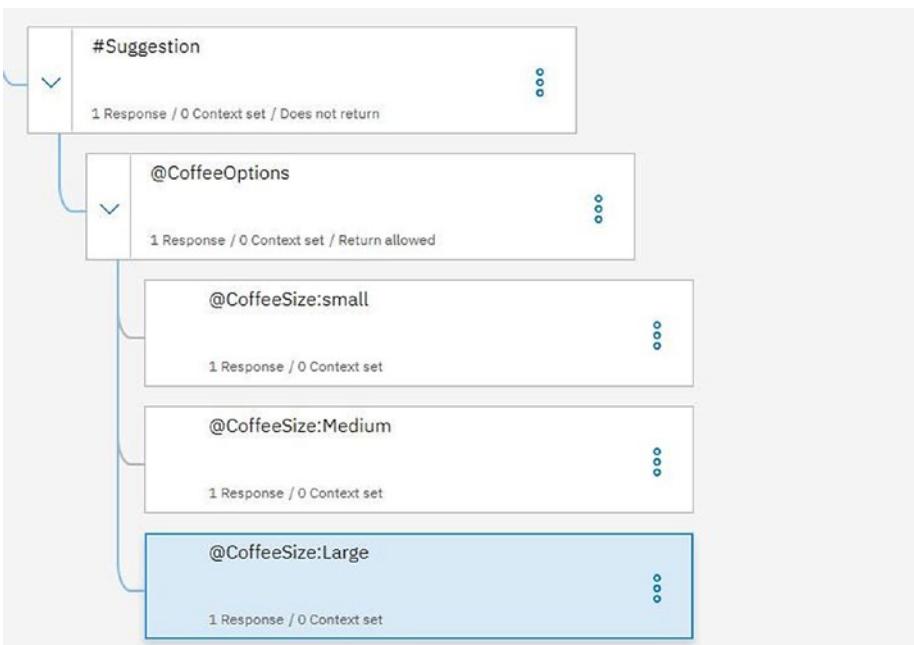


Figure 4-50. Nested intent conveys what we are doing for purchasing or ordering coffee with all the options for small, medium, and large

Now the next node will be for canceling the order. XXXXXXXXXXXX, as shown in Figure 4-51.

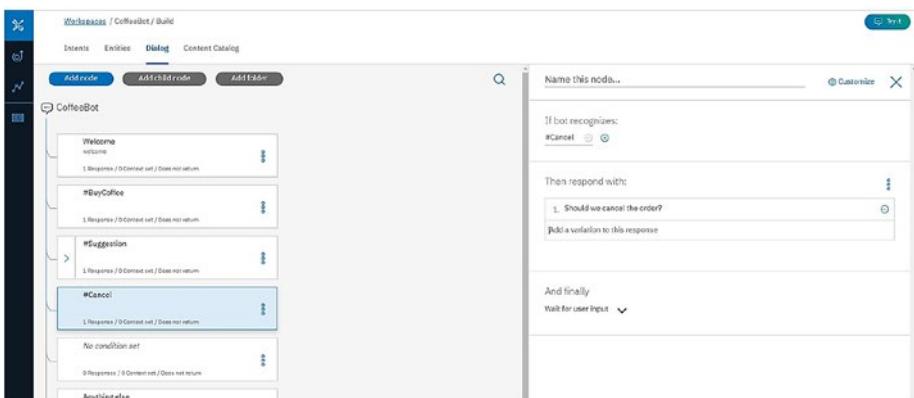


Figure 4-51. Canceling the status of the order is done here through the conversation with the dialogs order

You will have options for Yes and No, so further nesting the process is required. When you add up all the nodes, you get the categorized structure shown in Figure 4-52!

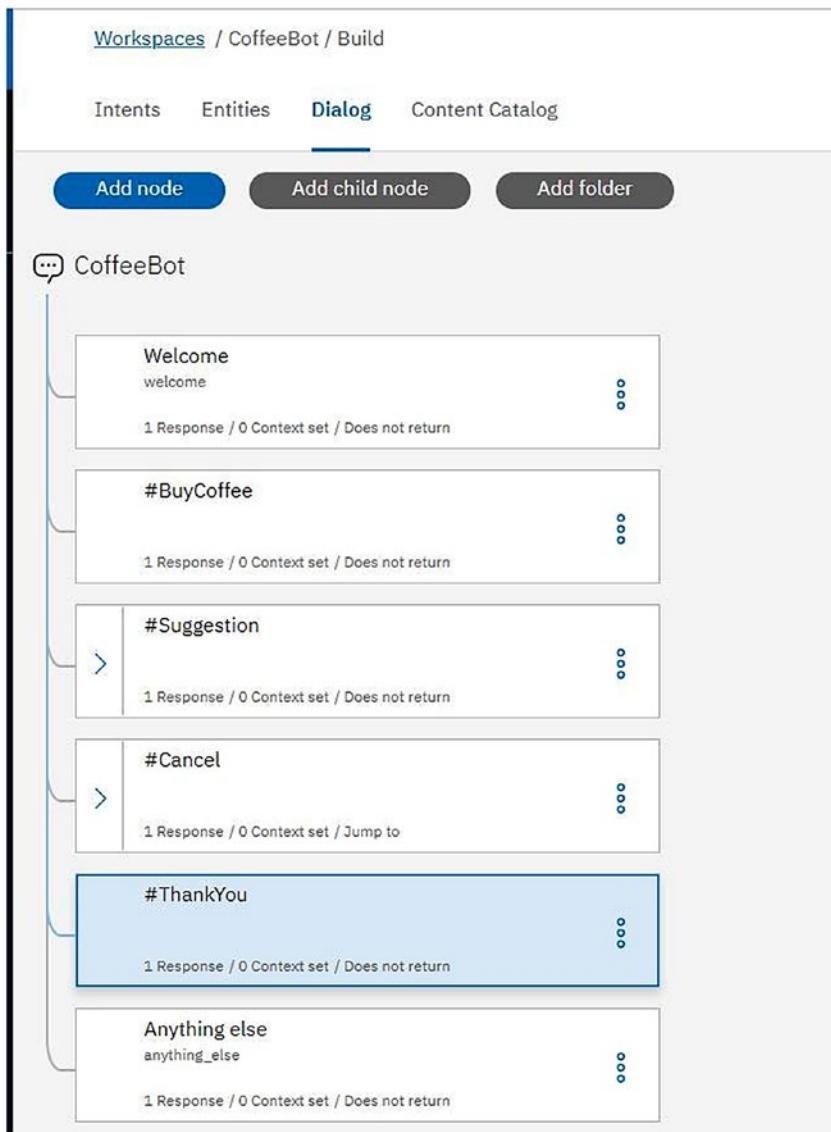


Figure 4-52. The complete workflow for the CoffeeBot

Conclusion

In this chapter, you learned how to get started with the IBM Cloud; note that IBM offers a free one-month trial of this service.

This chapter also introduced IBM Watson. You learned how to create two chatbots with Watson Assistant.

In the last chapter, your journey to learn about chatbots will continue. You will create TensorFlow chatbots and learn about their uses.

CHAPTER 5

Chatbot with TensorFlow

In this chapter, you will create chatbots by using TensorFlow. You'll start by learning some TensorFlow basics.

You'll work on open source models. Then you'll then move on to different approaches for creating the chatbots. You'll set up a TensorFlow GPU with access to NVIDIA CUDA. You'll closely examine CUDA and then create the chatbots.

TensorFlow Basics

TensorFlow is a data science framework primarily meant for dataflow-based work. It uses Tensors and their approach to nodes in an effective way so that we can easily implement it in machine learning as well as deep learning, a Tensor is a generalized matrix that might be 1d, 2d or of higher order.

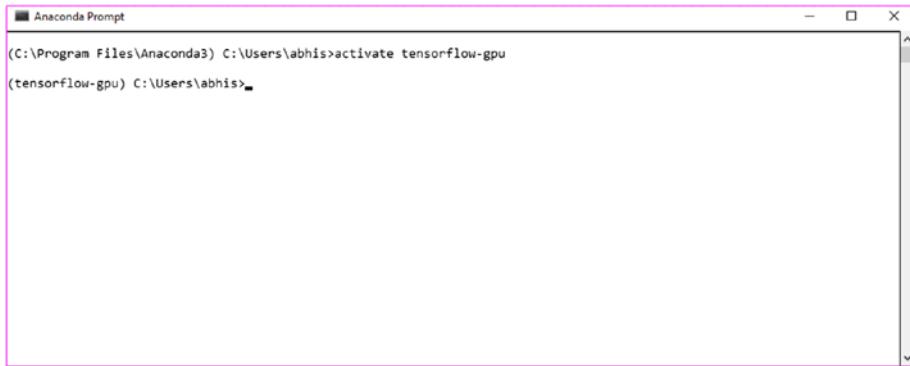
This section presents the basics of Tensors and of setting up the proper working environment. Later in this chapter, you will construct a neural network from scratch in TensorFlow and then use the TensorBoard feature to see how a TensorFlow graph works.

Setting Up the Working Environment

This section provides an overview of the way the Anaconda distribution for Python is set up with the GPU version.

You will work on activating the Anaconda environment and start your TensorFlow basics from there.

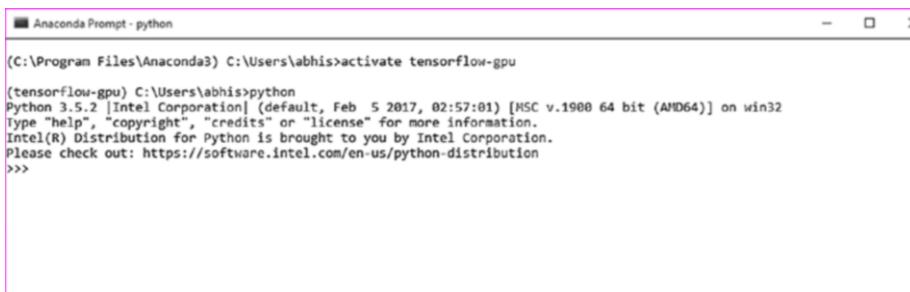
You'll use an Anaconda distribution for Python and then install TensorFlow with it. You'll be working with the GPU version. First you activate the Anaconda environment, as shown in Figure 5-1.



```
Anaconda Prompt
(C:\Program Files\Anaconda3) C:\Users\abhis>activate tensorflow-gpu
(tensorflow-gpu) C:\Users\abhis>
```

Figure 5-1. Activating the TensorFlow environment

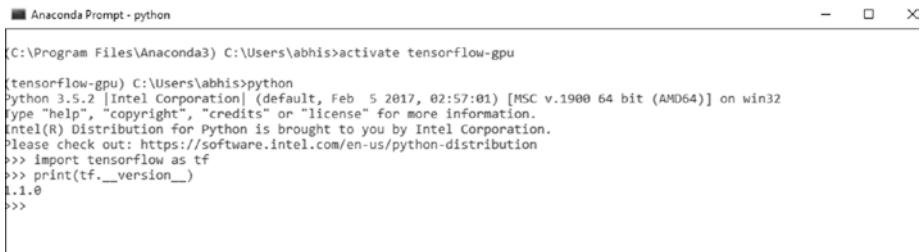
You'll use Intel-optimized Python for writing the code in Python. The mode is shown in Figure 5-2.



```
Anaconda Prompt - python
(C:\Program Files\Anaconda3) C:\Users\abhis>activate tensorflow-gpu
(tensorflow-gpu) C:\Users\abhis>python
python 3.5.2 |Intel Corporation| (default, Feb  5 2017, 02:57:01) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Intel(R) Distribution for Python is brought to you by Intel Corporation.
please check out: https://software.intel.com/en-us/python-distribution
>>>
```

Figure 5-2. Intel-optimized Python

Now let's check our version of TensorFlow, as shown in Figure 5-3.



```
Anaconda Prompt - python
(C:\Program Files\Anaconda3) C:\Users\abhis>activate tensorflow-gpu
(tensorflow-gpu) C:\Users\abhis>python
Python 3.5.2 [Intel Corporation] (default, Feb  5 2017, 02:57:01) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Intel(R) Distribution for Python is brought to you by Intel Corporation.
Please check out: https://software.intel.com/en-us/python-distribution
>>> import tensorflow as tf
>>> print(tf.__version__)
1.1.0
>>>
```

Figure 5-3. Checking the version of TensorFlow

To check the version, first you import TensorFlow:

```
>>> import tensorflow as tf
>>> print(tf.__version__)
1.1.0
```

Let's just break the word tensor it means n-dimensional array.

You will create the most basic thing in TensorFlow that is constant. You will create a variable named hello. The work is shown here and in Figure 5-4:

```
>>> import tensorflow as tf
>>> hello = tf.constant("Hello")
>>> Intel = tf.constant("Intel")
>>> type(Intel)
<class 'tensorflow.python.framework.ops.Tensor'>
>>> print(Intel)
Tensor("Const_1:0", shape=(), dtype=string)
>>> with tf.Session() as sess:
...     result=sess.run(hello+Intel)
Now we print the result.
>>> print(result)
b'Hello Intel'
>>>
```

CHAPTER 5 CHATBOT WITH TENSORFLOW

```
■ executeFromCell
[{"id": "3", "type": "code"}, {"text": "import tensorflow as tf\na = tf.constant(50)\nb = tf.constant(70)\nprint(a+b)", "language": "python"}, {"text": "2017-12-10 10:51:30,400000: W c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\core\platform\cpu_feature_guard.cc:41] The TensorFlow library wasn't compiled to use SSE instructions, but these are available on your machine and could speed up CPU computations.\n2017-12-10 10:51:30,400000: W c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\core\platform\cpu_feature_guard.cc:41] The TensorFlow library wasn't compiled to use SSE2 instructions, but these are available on your machine and could speed up CPU computations.\n2017-12-10 10:51:30,400000: W c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\core\platform\cpu_feature_guard.cc:41] The TensorFlow library wasn't compiled to use SSE3 instructions, but these are available on your machine and could speed up CPU computations.\n2017-12-10 10:51:30,400000: W c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\core\platform\cpu_feature_guard.cc:41] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computations.\n2017-12-10 10:51:30,400000: W c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\core\platform\cpu_feature_guard.cc:41] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computations.\n2017-12-10 10:51:30,400000: W c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\core\platform\cpu_feature_guard.cc:41] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could speed up CPU computations.\n2017-12-10 10:51:30,400000: W c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\lib\util\cnnmeminfo\cnnmeminfo.h(36): Found Device # with properties:\n  name: GeForce GTX 980M\n  pci bus id: 0000:01:00.0\n  total memory: 4.00GB\n  total memory: 4.00GB\n2017-12-10 10:51:30,500000: I c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\util\cnnmeminfo\cnnmeminfo.h(36): 0000:01:00.0\n2017-12-10 10:51:30,500000: I c:\Users\Klemen\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\util\cnnmeminfo\cnnmeminfo.h(36): Creating TensorFlow device (/gpu:0) => (device: 0, name: GeForce GTX 980M, pci bus id: 0000:01:00.0)
```

Figure 5-4. Working with TensorFlow

Now let's add two numbers in TensorFlow. First, you declare two variables:

```
>>> a =tf.constant(50)\n>>> b =tf.constant(70)
```

You check the type of one of the variables:

```
>>> type(a)\n<class 'tensorflow.python.framework.ops.Tensor'>
```

Here, you can see that the object is of type Tensor.

To add two variables, you have to create a session:

```
>>> with tf.Session() as sess:\n...     result = sess.run(a+b)
```

To see the result, just type result:

```
>>> result\n120\n>>>
```

Creating a Neural Network

In this section, you will create a neural network that performs a simple linear fit to some 2D data. You will work with TensorFlow to create a graph. You'll initiate the session, feed the data into TensorFlow, and get the output.

Figure 5-5 shows the flow of work in TensorFlow.

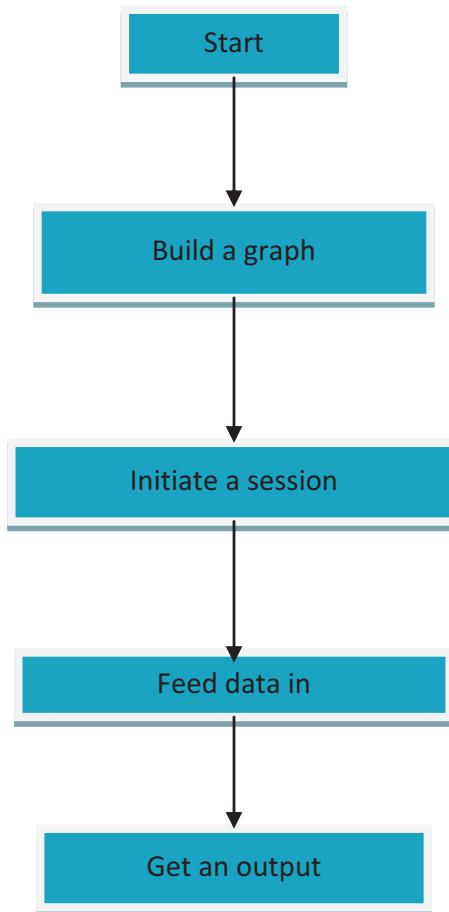


Figure 5-5. Flow of TensorFlow

Figure 5-6 shows the structure of the neural network that you'll create.

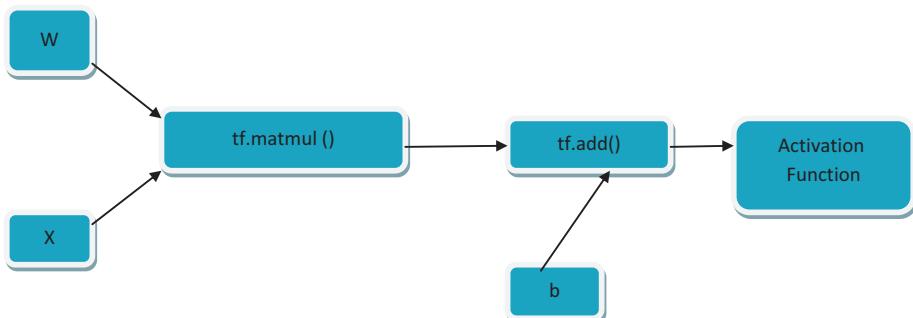


Figure 5-6. Constructing a neural network

You use the following linear equation to implement the neural network:

$$WX + b = Z$$

You will add in a cost function to train the network to optimize the parameters.

First, import NumPy and TensorFlow:

```
(C:\Program Files\Anaconda3) C:\Users\abhis>activate
tensorflow-gpu
(tensorflow-gpu) C:\Users\abhis>python
Python 3.5.2 |Intel Corporation| (default, Feb  5 2017,
02:57:01) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
Intel(R) Distribution for Python is brought to you by Intel
Corporation.
Please check out: https://software.intel.com/en-us/python-distribution
>>> import numpy as np
>>> import tensorflow as tf
>>>
```

You need to set random seed values for our process:

```
>>> np.random.seed(101)
>>> tf.set_random_seed(101)
```

Add some random data:

Using `rand_a =np.random.uniform(0,100(5,5))`

You add random data points going from 0 to 100 and then ask operations logic to be in the shape of (5,5). You do the same for b:

```
>>> rand_a =np.random.uniform(0,100,(5,5))
>>> rand_a
array([[ 51.63986277,  57.06675869,   2.84742265,  17.15216562,
       68.52769817],
       [ 83.38968626,  30.69662197,  89.36130797,  72.15438618,
      18.99389542],
       [ 55.42275911,  35.2131954 ,  18.18924027,  78.56017619,
      96.54832224],
       [ 23.23536618,   8.35614337,  60.35484223,  72.89927573,
      27.62388285],
       [ 68.53063288,  51.78674742,   4.84845374,  13.78692376,
      18.69674261]])
>>> rand_b
array([[ 99.43179012],
       [ 52.06653967],
       [ 57.87895355],
       [ 73.48190583],
       [ 54.19617722]])
```

Create placeholders for these uniform objects:

```
>>> a = tf.placeholder(tf.float32)
>>> b = tf.placeholder(tf.float32)
```

CHAPTER 5 CHATBOT WITH TENSORFLOW

You use TensorFlow because it understands normal Python operations:

```
>>> add_op = a + b  
>>> mul_op = a * b
```

Now you'll create sessions that use graphs to feed dictionaries to get results. First declare the session and then get the results of the add operation. Pass in the operation and the feed dictionary. For the placeholder objects, you need to feed data; you will do that by using the feed dictionary.

Pass data to the keys A and B:

```
add_result = sess.run(add_op,feed_dict={a:10,b:20})  
>>> with tf.Session() as sess:  
...     add_result = sess.run(add_op,feed_dict={a:10,b:20})  
...     print(add_result)
```

As you have created random result, you will pass it to the feed dictionary:

```
>>> with tf.Session() as sess:  
...     add_result = sess.run(add_op,feed_  
dict={a:rand_a,b:rand_b})
```

Print the value of add_result:

```
>>> print(add_result)  
[[ 151.07165527  156.49855042  102.27921295  116.58396149  167.95948792]  
 [ 135.45622253   82.76316071  141.42784119  124.22093201   71.06043243]  
 [ 113.30171204   93.09214783   76.06819153  136.43911743  154.42727661]  
 [  96.7172699    81.83804321  133.83674622  146.38117981  101.10578918]  
 [ 122.72680664  105.98292542   59.04463196   67.98310089   72.89292145]]
```

Create a matrix for multiplication:

```
>>> with tf.Session() as sess:  
...     mul_result = sess.run(mul_op,feed_dict={a:10,b:20})  
...     print(mul_result)
```

Use the following random values:

```
>>> with tf.Session() as sess:  
...     mul_result = sess.run(mul_op,  
...                           feed_dict={a:rand_a,b:rand_b})  
>>> print(mul_result)  
[[ 5134.64404297  5674.25          283.12432861  1705.47070312  
  6813.83154297]  
 [ 4341.8125      1598.26696777  4652.73388672  3756.8293457  
  988.9463501 ]]  
 [ 3207.8112793   2038.10290527  1052.77416992  4546.98046875  
  5588.11572266]  
 [ 1707.37902832   614.02526855  4434.98876953  5356.77734375  
  2029.85546875]  
 [ 3714.09838867   2806.64379883   262.76763916   747.19854736  
  1013.29199219]]
```

Create a neural network from result we obtained. Let's add some features to the data:

```
>>> n_features =10
```

Declare the number of layers of neurons. In this case, you have three:

```
>>> n_dense_neurons = 3
```

Let's create a placeholder for x and add the data type, which is float. Then you have to find the shape. First, you consider it None because it depends on the batch of data you are feeding to the neural network. Columns will be the number of features.

```
>>> x = tf.placeholder(tf.float32,(None,n_features))
```

CHAPTER 5 CHATBOT WITH TENSORFLOW

Now you have the other variables. W is the weight variable, and you initialize this with some sort of randomness; then you have the shape of it be based on the number of features with the number of neurons in the layer:

```
>>> W = tf.Variable(tf.random_normal([n_features,n_dense_neurons]))
```

Declare the bias.

You declare the variables and can have it as ones or zeros we are using the function within TensorFlow. Keep in mind that W will be multiplied by x , so you need to maintain the dimension of column with dimension of rows for matrix multiplication:

```
>>> b = tf.Variable(tf.ones([n_dense_neurons]))  
...  
...  
...  
... )  
>>>
```

Create an operation and activation function:

```
>>> xW = tf.matmul(x,W)
```

Create the output z :

```
>>> z = tf.add(xW,b)
```

Create the activation function:

```
>>> a = tf.sigmoid(z)
```

To complete the graph or the flow, run them in a simple session:

```
>>> init = tf.global_variables_initializer()
```

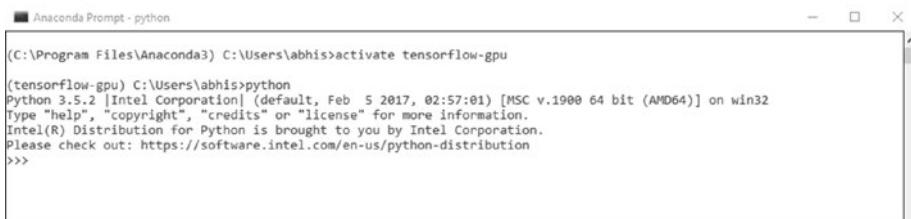
Finally, pass in a feed dictionary to create a session:

```
>>> with tf.Session() as sess:  
...     sess.run(init)  
...     layer_out = sess.run(a,feed_dict={x:np.random.  
...         random([1,n_features])})  
>>> print(layer_out)  
[[ 0.19592889  0.84230143  0.36188066]]
```

You have now created a neural network and printed the final output layer.

Working with the Activation Function

You will now start working with the activation function and implement it in TensorFlow to view any layer you want. You get accordingly the Intel-optimized Python mode. You need to enable the environment again, as shown in Figure 5-7.



```
Anacoda Prompt - python  
(C:\Program Files\Anaconda3) C:\Users\abhis>activate tensorflow-gpu  
(tensorflow-gpu) C:\Users\abhis>python  
Python 3.5.2 [Intel Corporation] (default, Feb 5 2017, 02:57:01) [MSC v.1900 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
Intel(R) Distribution for Python is brought to you by Intel Corporation.  
Please check out: https://software.intel.com/en-us/python-distribution  
>>>
```

Figure 5-7. Enabling the environment again

Now you import to get inside TensorFlow:

```
>>> import TensorFlow as tf
```

Next you implement the layer function.

For the layer, you should have inputs; this is the information being processed from the last layer. Using `in_size` determines the size of the input; this also describes the number of hidden neurons for the last layer. Using `out_layer` shows the number of neurons for this layer. Then you declare the activation function, which is `None`—that is, you are using a linear activation function.

You have to define weight that is based on input and output size.

You will have to use random normal to generate the weights. You will then pass the input and the output size. Initially, you use randomized values because it improves the neural network.

You declare one-dimensional biases. You will initialize biases as zeros and initialize all variables as 0.1. The dimension of it is 1 row and `out_size` the no of columns. Because you want to add the weights to the bias, the shape should be the same, so you use `out_size`.

For the operation or the compute process, you use matrix multiplication:

```
def add_layer(inputs, in_size, out_size,
activation_function=None):
    Weights = tf.Variable(tf.random_normal([in_size, out_size]))
    biases = tf.Variable(tf.zeros([1, out_size]) + 0.1)
    Wx_plus_b = tf.matmul(inputs, Weights) + biases
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
        print(outputs)
    return outputs
```

In the next section, you will review an important feature of TensorFlow known as TensorBoard, which is useful for viewing the graph as well as debugging.

TensorBoard

Let's talk about TensorBoard. *TensorBoard* is a data visualization tool that is packaged within TensorFlow. When you are dealing with creating a network in TensorFlow, it's composed of operations and tensors. When you feed data into the neural network, the data flows in through tensors performing operations, and finally getting an output.

TensorBoard was created to know the flow of tensors in a model. It helps in debugging and optimization. Let's create some graphs and then show them in TensorBoard. The basic operations are addition and multiplication.

Figure 5-8 shows how a session works in TensorFlow.

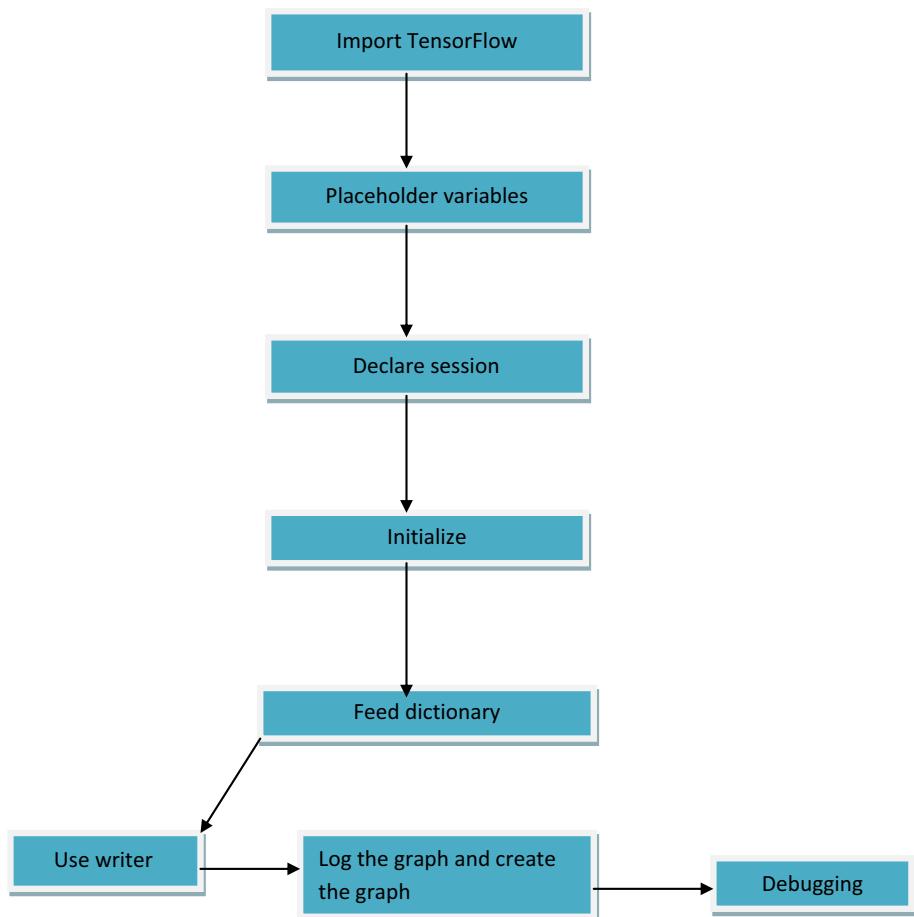


Figure 5-8. Creating a session with TensorFlow

Import TensorFlow as follows:

```
Import tensorflow as tf
```

We are showcasing an addition operation and showing the result in the tensorboard.

Then declare placeholder variables:

```
X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")
```

Next, you need to declare the operations that you need to perform:

```
addition = tf.add(X, Y, name="addition")
```

In the next step, you have to declare session. You want to perform operations, and you need to perform the operations inside a session. You have to initialize the variables by using `init`. Then we have to run the `sess` within `init`:

```
sess = tf.Session()
# tf.initialize_all_variables() no longer valid from
# 2017-03-02 if using tensorflow >= 0.12
if int((tf.__version__).split('.')[1]) < 12 and int((tf.__
version__).split('.')[0]) < 1:
    init = tf.initialize_all_variables()
else:
    init = tf.global_variables_initializer()
sess.run(init)
```

When you run the session using a feed dictionary, you initialize the values of the variables:

```
result = sess.run(addition, feed_dict ={X: [5,2,1], Y: [10,6,1]})
```

Finally, using the summary writer, you get the debugging logs for the graph:

```
if int((tf.__version__).split('.')[1]) < 12 and int((tf.__
version__).split('.')[0]) < 1: # tensorflow version < 0.12
    writer = tf.train.SummaryWriter('logs/nono', sess.graph)
else: # tensorflow version >= 0.12
    writer = tf.summary.FileWriter("logs/nono", sess.graph)
```

The entire code base in Python looks like this in a single oriented flow:

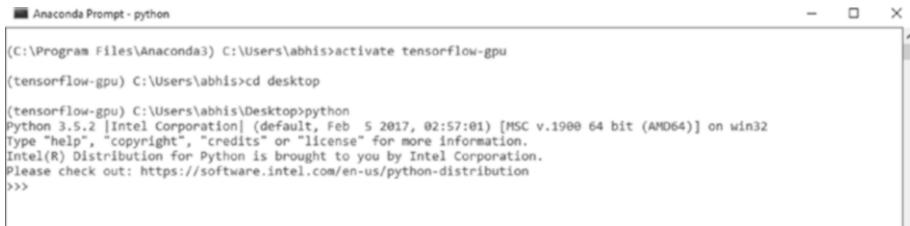
```
import tensorflow as tf
X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")

addition = tf.add(X, Y, name="addition")

sess = tf.Session()
# tf.initialize_all_variables() no longer valid from
# 2017-03-02 if using tensorflow >= 0.12
if int((tf.__version__).split('.')[1]) < 12 and int((tf.__
version__).split('.')[0]) < 1:
    init = tf.initialize_all_variables()
else:
    init = tf.global_variables_initializer()
sess.run(init)

if int((tf.__version__).split('.')[1]) < 12 and int((tf.__
version__).split('.')[0]) < 1: # tensorflow version < 0.12
    writer = tf.train.SummaryWriter('logs/nono', sess.graph)
else: # tensorflow version >= 0.12
    writer = tf.summary.FileWriter("logs/nono", sess.graph)
```

Let's now visualize the graph generated. Go to the Anaconda prompt. Activate the environment and go to the folder to run the Python file. In Figure 5-9, you are enabling the Intel-optimized Python mode again.



A screenshot of the Anaconda Prompt window titled "Anaconda Prompt - python". The command line shows the user activating the "tensorflow-gpu" environment and then running Python. The Python welcome message indicates it is an Intel-optimized distribution.

```
(C:\Program Files\Anaconda3) C:\Users\abhis>activate tensorflow-gpu
(tensorflow-gpu) C:\Users\abhis>cd desktop
(tensorflow-gpu) C:\Users\abhis\Desktop>python
python 3.5.2 [Intel Corporation] (default, Feb 5 2017, 02:57:01) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Intel(R) Distribution for Python is brought to you by Intel Corporation.
please check out: https://software.intel.com/en-us/python-distribution
>>>
```

Figure 5-9. Enabling Python mode

Now you need to run the Python file. Use the following command to get the output shown in Figure 5-10:

```
(tensorflow-gpu) C:\Users\abhis\Desktop>python abb2.py
```

```
[tensorfow-gpu] C:\Users\abhis\Desktop>python abb2.py
2018-01-02 01:19:10.000000: W c:\users\abhis\desktop\python\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE instructions, but these are available on your machine and could speed up CPU computations.
2018-01-02 01:19:10.000000: W c:\users\abhis\desktop\python\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE2 instructions, but these are available on your machine and could speed up CPU computations.
2018-01-02 01:19:10.000000: W c:\users\abhis\desktop\python\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE3 instructions, but these are available on your machine and could speed up CPU computations.
2018-01-02 01:19:10.000000: W c:\users\abhis\desktop\python\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computations.
2018-01-02 01:19:10.000000: W c:\users\abhis\desktop\python\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computations.
2018-01-02 01:19:10.000000: W c:\users\abhis\desktop\python\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could speed up CPU computations.
2018-01-02 01:19:10.000000: W c:\users\abhis\desktop\python\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine and could speed up CPU computations.
2018-01-02 01:19:10.000000: W c:\users\abhis\desktop\python\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, but these are available on your machine and could speed up CPU computations.
2018-01-02 01:19:10.000000: I c:\users\abhis\desktop\python\code\common_runtime\gpu\gpu_device.cc:1007] Found device 0 with properties:
name: GeForce GTX 960
pciBusID: 0000:01:00.0
totalMemory: 3.05GB
FreeMemory: 3.05GB
2018-01-02 01:19:10.000000: I c:\users\abhis\desktop\python\code\common_runtime\gpu\gpu_device.cc:1008] DMA: 0
2018-01-02 01:19:10.000000: I c:\users\abhis\desktop\python\code\common_runtime\gpu\gpu_device.cc:1009] B: 0
2018-01-02 01:19:10.000000: I c:\users\abhis\desktop\python\code\common_runtime\gpu\gpu_device.cc:1077] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX 960M, pci Bus ID: 0000:01:00.0)

[tensorflow-gpu] C:\Users\abhis\Desktop>
```

Figure 5-10. Running the code

Now open TensorBoard:

```
(tensorflow-gpu) C:\Users\abhis\Desktop>tensorboard
--logdir=logs/nono
```

WARNING:tensorflow:Found more than one graph event per run, or there was a metagraph containing a graph_def, as well as one or more graph events. Overwriting the graph with the newest event.
Starting TensorBoard b'47' at <http://0.0.0.0:6006>
(Press CTRL+C to quit)

Now let's open the browser for TensorBoard access.

The following link needs to be opened:

<http://localhost:6006/>

Figure 5-11 shows the addition operation in TensorBoard.

CHAPTER 5 CHATBOT WITH TENSORFLOW

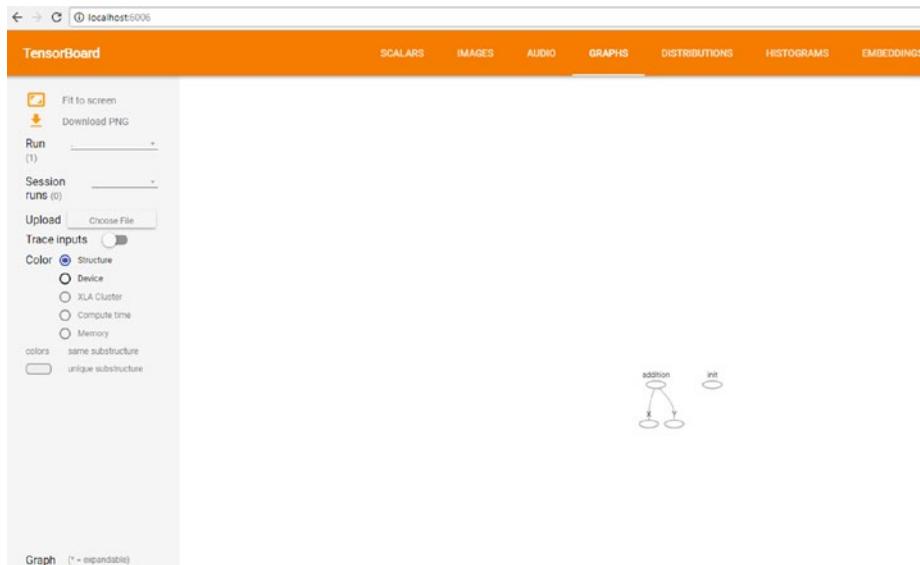


Figure 5-11. TensorBoard output

For multiplication, the same process is followed and the code base is shared here:

```
import tensorflow as tf
X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")

multiplication = tf.multiply(X, Y, name="multiplication")

sess = tf.Session()
# tf.initialize_all_variables() no longer valid from
# 2017-03-02 if using tensorflow >= 0.12
if int((tf.__version__).split('.')[1]) < 12 and int((tf.__
version__).split('.')[0]) < 1:
    init = tf.initialize_all_variables()
else:
    init = tf.global_variables_initializer()
```

```

sess.run(init)
result = sess.run(multiplication, feed_dict ={X: [5,2,1], Y:
[10,6,1]})

if int((tf.__version__).split('.')[1]) < 12 and int((tf._
version__).split('.')[0]) < 1: # tensorflow version < 0.12
    writer = tf.train.SummaryWriter('logs/no1', sess.graph)
else: # tensorflow version >= 0.12
    writer = tf.summary.FileWriter("logs/no1", sess.graph)

```

Figure 5-12 shows the multiplication graph.



Figure 5-12. Multiplication analysis in TensorBoard

Let's work with a more complex tutorial to see how the TensorBoard visualization works. You will be using activation function definition shown previously.

Declare placeholders:

```

xs = tf.placeholder(tf.float32, [None, 1], name='x_input')
ys = tf.placeholder(tf.float32, [None, 1], name='y_input')

```

Add a hidden layer with the `relu` activation function:

```
l1 = add_layer(xs, 1, 10, activation_function=tf.nn.relu)
```

Add the output layer:

```
prediction = add_layer(l1, 10, 1, activation_function=None)
```

Calculate the error:

```
with tf.name_scope('loss'):
    loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - prediction),
                                       reduction_indices=[1]))
```

Next you need to train the network. You will be using the gradient descent optimizer:

```
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
```

Versions of TensorFlow

The current versions of TensorFlow that are available are as follows:

- r1.8
- r1.7
- r1.6
- r1.5
- r1.4
- r1.3
- r1.2
- r1.1

The various versions of TensorFlow are available at the TensorFlow branch, www.tensorflow.org/versions/.

Keras Overview

This section introduces a front-end wrapper to a deep learning framework known as Keras. Then we will work along with how we implement Keras using a Jupyter Notebook and creating chatbots with it.

Keras is front-end wrapper that can be used with lots of deep learning frameworks at the back end.

Keras has already been built up neural network function that you can use to get the neural networks easily and fast. It maintains the following:

- Modularity
- Minimalism
- Extensibility
- Python nativeness

As a basic starting point for Keras, you will run a “Hello World” example. With respect to machine learning and deep learning, a Hello World example is different.

In this example, you will use the `iris` dataset. You need to check that the following libraries are installed:

- `seaborn`
- `numpy`
- `sklearn`
- `keras`
- `tensorflow`

TensorFlow will be the background for the Keras wrapper.

You will sync a GitHub project:

<https://github.com/fastforwardlabs/keras-hello-world>

CHAPTER 5 CHATBOT WITH TENSORFLOW

The important files will be downloaded via a requirements.txt file.

Let's open an Anaconda prompt and get inside the Tensorflow-gpu environment:

```
(C:\Users\abhis\Anaconda3) C:\Users\abhis>activate tensorflow-gpu
```

Now you will clone the project files, which will copy the essential files into the folder and create a local copy in the machine:

```
(tensorflow-gpu) F:\>git clone https://github.com/fastforwardlabs/keras-hello-world.git
Cloning into 'keras-hello-world'...
remote: Counting objects: 94, done.
remote: Total 94 (delta 0), reused 0 (delta 0), pack-reused 94
Unpacking objects: 100% (94/94), done.
```

Now you install the necessary files by using the following code:

```
(tensorflow-gpu) F:\keras-hello-world>pip install requirements
```

You'll launch the Jupyter Notebook. But first let's import the libraries, as shown in Figure 5-13.

```
In [1]: import seaborn as sns
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV

from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils

c:\users\abhis\conda\envs\tensorflow-gpu\lib\site-packages\h5py\_init_.py:36: FutureWarning: Conversion of the second argument of issubdtype from 'float' to 'np.floating' is deprecated. In future, it will be treated as 'np.float64 == np.dtype(float).type'.
  from .conv import register_converters as _register_converters
Using TensorFlow backend.
```

Figure 5-13. Importing libraries

The program starts using TensorFlow back end.

The `iris` dataset is indeed useful for machine learning, and you load it first:

```
iris = sns.load_dataset("iris")
iris.head()
```

Figure 5-14 shows how the `iris` dataset is loaded.

In [4]:

```
iris = sns.load_dataset("iris")
iris.head()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa



Figure 5-14. Using `iris` dataset

Now you visualize the dataset:

```
sns.pairplot(iris, hue='species');
```

In Figure 5-15, you are plotting the species.

CHAPTER 5 CHATBOT WITH TENSORFLOW

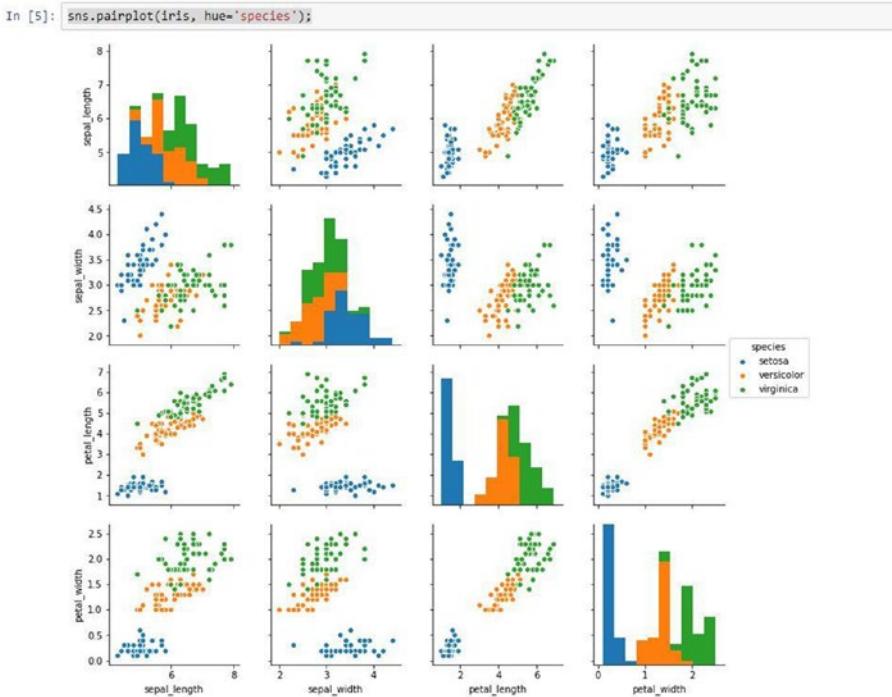


Figure 5-15. Plotting the species

Now you split the data into training and testing datasets.

First pull the raw dataframe:

```
X = iris.values[:, :4]  
y = iris.values[:, 4]
```

Then split the data:

```
train_X, test_X, train_y, test_y = train_test_split(X, y,  
train_size=0.5, test_size=0.5, random_state=0)
```

Now you train by using a Scikit classifier, as shown in Figure 5-16.

```
In [9]: lr = LogisticRegressionCV()
lr.fit(train_X, train_y)

Out[9]: LogisticRegressionCV(Cs=10, class_weight=None, cv=None, dual=False,
    fit_intercept=True, intercept_scaling=1.0, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
    refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
```

Figure 5-16. Using logistic regression

Now you check the classifier accuracy:

```
print("Accuracy = {:.2f}".format(lr.score(test_X, test_y)))
```

Now we can measure the fraction of the test set the trained classifier classifies correctly (i.e., accuracy).

```
In [10]: print("Accuracy = {:.2f}".format(lr.score(test_X, test_y)))
Accuracy = 0.83
```

Getting Started with a Keras Chatbot

In this section, you will create a chatbot with Keras. Download the repo and perform the following steps:

1. Create a data folder in your project directory, and download the Cornell Movie-Dialogs Corpus from the following site:

www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

2. Unzip the file and update config.py file
 Change DATA_PATH to indicate the location where you store your data.
3. python3 data.py
s will do all the preprocessing for the Cornell dataset. Download Keras by using the following, as shown in Figure 5-17:

Pip install keras

CHAPTER 5 CHATBOT WITH TENSORFLOW

```
(idpFull) F:\keras\examples>pip install keras
Collecting keras
  Downloading https://files.pythonhosted.org/packages/ba/65/e4afff762b8696ec0626a6654b1e73b396fcc8b7cc6b98d78a1bc53b85b48/Keras-2.1.5-py2.py3-
  100% |██████████| 337kB 43kB/s
Requirement already satisfied: scipy>=0.14 in c:\users\abhis\anaconda3\envs\idpfull\lib\site-packages (from keras)
Requirement already satisfied: six<1.9.0 in c:\users\abhis\anaconda3\envs\idpfull\lib\site-packages (from keras)
Requirement already satisfied: numpy>=1.9.1 in c:\users\abhis\anaconda3\envs\idpfull\lib\site-packages (from keras)
Requirement already satisfied: pyyaml in c:\users\abhis\anaconda3\envs\idpfull\lib\site-packages (from keras)
Installing collected packages: keras
Successfully installed keras-2.1.5
```

Figure 5-17. Downloading Keras

Now you prepare the training set and the testing set. In Figure 5-18 you are accumulating data for the bot.

```
(idpFull) F:>cd F:\ManishaBot\stanford-tensorflow-tutorials\assignments\chatbot

(idpFull) F:\ManishaBot\stanford-tensorflow-tutorials\assignments\chatbot>dir
 Volume in drive F is abhishekcontent
 Volume Serial Number is 1EA9-72D7

Directory of F:\ManishaBot\stanford-tensorflow-tutorials\assignments\chatbot

.5-04-2018 01:33 <DIR> .
.5-04-2018 01:33 <DIR> ..
.5-04-2018 01:18 11,797 chatbot.py
.5-04-2018 01:33 <DIR> checkpoints
.5-04-2018 01:27 1,394 config.py
.0-05-2011 16:12 <DIR> cornell movie-dialogs corpus
.3-04-2018 20:14 9,916,637 cornell_movie_dialogs_corpus.zip
.5-04-2018 01:24 <DIR> data
.5-04-2018 01:18 9,744 data.py
.5-04-2018 01:18 6,394 model.py
.5-04-2018 01:18 10,354 output_convo.txt
.5-04-2018 01:28 <DIR> processed
.5-04-2018 01:18 3,873 README.md
.5-04-2018 01:33 <DIR> __pycache__
    7 File(s) 9,960,193 bytes
    7 Dir(s) 68,881,727,488 bytes free

(idpFull) F:\ManishaBot\stanford-tensorflow-tutorials\assignments\chatbot>python data.py
'reparing raw data into train set and test set ...
'movie_lines.txt
'reparing data to be model-ready ...
```

Figure 5-18. Accumulating data

4. Type the following:

```
python3 chatbot.py --mode [train/chat]
```

In Figure 5-19, we are training the data.

```
(idpFull) F:\ManishaBot\stanford-tensorflow-tutorials\assignments\chatbot>python data.py
Preparing raw data into train set and test set ...
movie_lines.txt
Preparing data to be model-ready ...

(idpFull) F:\ManishaBot\stanford-tensorflow-tutorials\assignments\chatbot>python chatbot.py --mode train
Data ready!
Bucketing conversation number 9999
Bucketing conversation number 19999
Bucketing conversation number 9999
Bucketing conversation number 19999
Bucketing conversation number 29999
Bucketing conversation number 39999
Bucketing conversation number 49999
Bucketing conversation number 59999
Bucketing conversation number 69999
Bucketing conversation number 79999
Bucketing conversation number 89999
Bucketing conversation number 99999
Bucketing conversation number 189999
Bucketing conversation number 119999
Bucketing conversation number 129999
Bucketing conversation number 139999
Bucketing conversation number 149999
Bucketing conversation number 159999
Bucketing conversation number 169999
Bucketing conversation number 179999
Bucketing conversation number 189999
Number of samples in each bucket:
[112838, 37505, 12832, 13776, 8597, 4758]
Bucket scale:
[0.5912108323747005, 0.7891201333994702, 0.8568330290333815, 0.9295272972887402, 0.9748926155372389, 1.0]
Initialize new model
Create placeholders
Create inference
-
```

Figure 5-19. Training the data

If the mode is `train`, you train the chatbot. By default, the model will restore the previously trained weights (if there are any) and continue training up on that. Creating the optimizer is shown in Figure 5-20.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

```
Time: 68.3281979560852
Create optimizer...
It might take a couple of minutes depending on how many buckets you have.
Creating opt for bucket 0 took 18.41565442085266 seconds
Creating opt for bucket 1 took 23.46096110343933 seconds
-
```

Figure 5-20. Creating the optimizer

CHAPTER 5 CHATBOT WITH TENSORFLOW

Now let's test the bot with the chat option, as shown in Figure 5-21.

```
(idpFull) F:\ManishaBot\stanford-tensorflow-tutorials\assignments\chatbot>python chatbot.py --mode chat
```

```
(idpFull) F:\ManishaBot\stanford-tensorflow-tutorials\assignments\chatbot>python chatbot.py --mode chat
Data ready!
Initialize new model
Create placeholders
Create inference
Creating loss...
It might take a couple of minutes depending on how many buckets you have.
```

Figure 5-21. We are enabling the chat mode

Now the bot will use the optimizer. Figure 5-22 shows how you are interacting with the bot.

```
Time: 68.36833882331848
Create optimizer...
It might take a couple of minutes depending on how many buckets you have.
Loading parameters for the Chatbot
Welcome to TensorBro. Say something. Enter to exit. Max length is 60
> hello
[[ -0.17155586  0.4813461  -0.07266062 ... -0.18004592  0.03955929
-0.04967962]]
i i i i i i i i i i i i i i i i
```

Figure 5-22. Interacting with the chatbot

you have created a chatbot using Keras. You have seen how the interaction happens with the chatbot. In the next section, you will work on another chatbot application.

Introducing nmt-chatbot

This section introduces nmt-chatbot. This chatbot works in this order:

1. Use translation with input and output encoders.
 2. The NMT-Bot first reads the sentence through the encoder.

3. A decoder processes the sentence vector.
4. The Chatbot uses LSTM approach.
5. Tokenize the input.

You will use Anaconda to create an environment, so let's get Anaconda first. Anaconda is an open source distribution for Python that is generally used for managing different packages.

Let's create an environment for Anaconda and name it Manisha.

Anaconda is available from www.anaconda.com/download/.

Figure 5-23 shows the different versions of Anaconda.

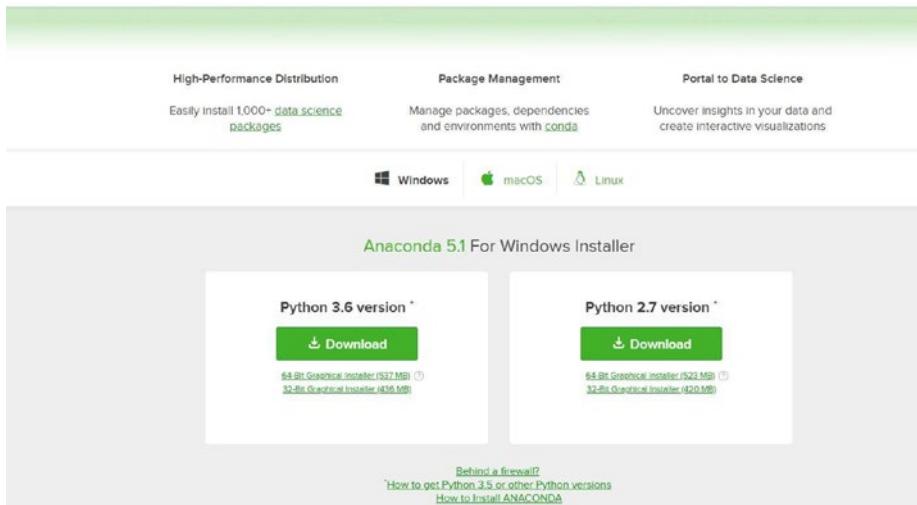


Figure 5-23. Different Anaconda versions available

After downloading Anaconda, you're ready to create the environment. In Windows, the process for creating an environment is as follows:

```
conda create -n yourenvname python=x.x anaconda
```

You then get inside the environment and install the GPU version of TensorFlow, as shown in Figure 5-24.

CHAPTER 5 CHATBOT WITH TENSORFLOW

```
#      $ conda activate manisha
# To deactivate an active environment, use
#      $ conda deactivate

(base) C:\Windows\system32>conda activate manisha
(manisha) C:\Windows\system32>pip install tensorflow-gpu
Collecting tensorflow-gpu
  Downloading https://files.pythonhosted.org/packages/23/ad/b3ae65999fd42a3e0d0044245e25233b31764157797051ef3ceb5202001/tensorflow_gpu-1.7.0-cp36-cp36m-win_amd64.whl (80
  100% [██████████] | 85.7MB 42KB/s
Collecting tensorboard<1.8.0,>=1.7.0 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/e0/ec/f5de8e8d1008ca2a78c0903d9440fd231228dd0dcdca7d470b223456e55d/tensorboard-1.7.0-pys-none-any.whl (3.1MB)
  100% [██████████] | 3.1MB 5.4MB/s
Collecting grpcio>=1.16.6 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/ff/ee/fe92822b61ea274230200bf2454faae6aejd7f4360e93ff71a17/grpcio-1.11.0-cp36-cp36m-win_amd64.whl (1.4MB)
  100% [██████████] | 1.4MB 3.7MB/s
Requirement already satisfied: six<1.16.0, >=1.15.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu)
Collecting protobuf<4.0.0,>=3.6.0 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/32/cf/c04518b4d76db0b5d11b20aae0eb2817523b0587918374c774fb6d209e/protobuf-3.5.2.post1-cp36-cp36m-win_amd64.whl (95
  100% [██████████] | 95.2MB 7.9MB/s
Requirement already satisfied: wheel<0.36.0, >=0.35.1 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu)
Collecting gast<0.2.0,>=0.1.0 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/sc/7b/ff724fcac2ce0ba5123e789df1fb3b7765f601c77027264f430e814822b96/gast-0.2.0.tar.gz
Collecting absl-py<0.1.0 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/fa/bc/bc922387a7bf38ef5aef6d1297656c87838985e7a95d488c3058a8677bd3/absl-py-0.1.13.tar.gz (800K)
  100% [██████████] | 818K 7.4MB/s
Collecting astor<0.6.0, >=0.5.4 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/29/91/cc90e51ff049f0e0130b3a7ca20f6a1be2ed4246008049e0fc459ff12/astor-0.6.2-py2.py3-none-any.whl
Requirement already satisfied: numpy<1.13.3, >=1.13.3 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu)
Collecting werkzeug<0.11.10, >=0.11.9 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/8a/40/470e516479e0eb9f10e2a5151bf3fbdcce0d14e0e4fabb45173aa79ff981/termcolor-1.1.0.tar.gz
Requirement already satisfied: werkzeug<0.11.10 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu)
Collecting bleach<1.5.0, >=1.4.0 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/97/e7/6418ad4d6c4fb955156d821e1c357500763003639090/beach-1.5.0-py2.py3-none-any.whl
Collecting html5lib<0.1.0, >=0.99999 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/a0/ac/0bc04092c69932b2d4af19b05870b29ndaa2cd17551ba5e39219fb5eb49/html5lib-0.9999999.tar.gz (889K)
  100% [██████████] | 889K 7.7MB/s
Collecting markdown<2.6.8 (from tensorflow-gpu)
  Downloading https://files.pythonhosted.org/packages/8a/40/470e516479e0eb9f10e2a5151bf3fbdcce0d14e0e4fabb45173aa79ff981/tensorflow-gpu
  100% [██████████] | 818K 3.9MB/s
Requirement already satisfied: setuptools in c:\anaconda\envs\manisha\lib\site-packages (from protobuf<3.4.0>+tensorflow-gpu)
Building wheels for collected packages: gevent, absl-py, termcolor, html5lib
Running setup.py bdist_wheel for gevent ... done
Stored in directory: C:\Users\9360\Anaconda3\envs\manisha\lib\site-packages\gevent-1.5.2.dist-info
Running setup.py bdist_wheel for absl-py ... done
Stored in directory: C:\Users\9360\Anaconda3\envs\manisha\lib\site-packages\absl-py-1.7.0.dist-info
Running setup.py bdist_wheel for termcolor ... done
Stored in directory: C:\Users\9360\Anaconda3\envs\manisha\lib\site-packages\termcolor-1.1.0.dist-info
Running setup.py bdist_wheel for html5lib ... done
Stored in directory: C:\Users\9360\Anaconda3\envs\manisha\lib\site-packages\html5lib-0.9999999.dist-info
Successfully built gevent absl-py termcolor html5lib
Installing collected packages: protobuf, html5lib, bleach, markdown, tensorboard, grpcio, gast, absl-py, astor, termcolor, tensorflow-gpu
  Found existing installation: html5lib 1.0.1
    Uninstalling html5lib-1.0.1:
      Successfully uninstalled html5lib-1.0.1
  Found existing installation: bleach 2.1.2
  Uninstalling bleach-2.1.2:
    Successfully uninstalled bleach-2.1.2
```

Figure 5-24. Installing Tensorflow-gpu

Import TensorFlow to check that everything is perfect and then start cloning the repo (a local copy of nmt-chatbot):

```
git clone --recursive https://github.com/daniel-kukiela/nmt-chatbot
```

Figure 5-25 shows cloning the repo.

```
(manisha) C:\Windows\system32>python
Python 3.6.4 |Anaconda, 2Sci| (default, Jan 16 2018, 10:22:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
C:\Anaconda\envs\manisha\lib\site-packages\ tensorflow\_init_.py:30: FutureWarning: Conversion of the second argument of issubdtype from 'float' to 'np.floating' is deprecated. In future, it will be tr
  from _conv import register_converters as _register_converters
>>> exit()
```

Figure 5-25. Cloning the repo

Use the following command in the command windows to get inside the local copy copied folder that is the github copy that you cloned:

```
cd nmt-chatbot
```

In Figure 5-26, you are installing the requirements.

```
(manisha) C:\Windows\system32>git clone --recursive https://github.com/daniel-kukieła/nmt-chatbot
Cloning into 'nmt-chatbot'...
remote: Compressing objects: 100% (313/313), done.
remote: Writing objects: 100% (313/313), done.
Receiving objects: 100% (313/313) 11.75 MiB | 22.32 MiB/s, done.
Resolving deltas: 100% (453/453), done.
Submodule 'nmt' (https://github.com/daniel-kukieła/nmt) registered for path 'nmt'
  Cloning into 'nmt'...
remote: Counting objects: 1111, done.
remote: Compressing objects: 100% (372/372), done.
remote: Writing objects: 100% (372/372), done.
Resolving deltas: 100% (298/298), done.
Receiving objects: 100% (1111/1111), 1.17 MiB | 19.3 MiB/s, done.
Resolving deltas: 100% (378/378), done.
Submodule path 'nmt': checked out <66ff0d3817506904793462e1e029f463a0e79>
(manisha) C:\Windows\system32>cd nmt-chatbot
(manisha) C:\Windows\system32>pip install -r requirements.txt
Requirement already satisfied: tensorflow-gpu>=1.4.0 in c:\anaconda\envs\manisha\lib\site-packages (from -r requirements.txt (line 1))
Collecting tensorflow<1.5.0,>=1.4.0 (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/16/9d/291c458408833829339571814943b598d4cc8f1a19df0957921357f10/regxx-2018.02.21-cp36-none-win\_amd64.whl (251kB)
    100% |████████████████████████████████| 251kB 6.00MB/s
Collecting python-Levenshtein<0.12.0,>=0.11.2 (from -r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/42/a2/42a2175c5eefbf5fb7fac30b93dd02820c34adeab3b1bcd00520eb4d6a5d/python-Levenshtein-0.12.0.tar.gz (49kB)
    100% |████████| 51kB 3.70MB/s
Requirement already satisfied: requests[certificates]>=2.19.0 in c:\anaconda\envs\manisha\lib\site-packages (from -r requirements.txt (line 4))
Requirement already satisfied: tensorflow<1.4.0,>=1.3.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: numpy>=1.13.3 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: tensorflow<1.4.0,>=1.3.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: astor>=0.6.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: protobuf>=3.4.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: gast>=0.2.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: wheel>=0.30.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: grpcio>=1.8.6 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: pycparser>=2.19.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: setuptools in c:\anaconda\envs\manisha\lib\site-packages (from python-Levenshtein->r requirements.txt (line 5))
Requirement already satisfied: charred<1.0,>=0.2 in c:\anaconda\envs\manisha\lib\site-packages (from requests->r requirements.txt (line 6))
Requirement already satisfied: idna>=2.5 in c:\anaconda\envs\manisha\lib\site-packages (from requests->r requirements.txt (line 6))
Requirement already satisfied: urllib3<1.23,>=1.21.1 in c:\anaconda\envs\manisha\lib\site-packages (from requests->r requirements.txt (line 6))
Requirement already satisfied: certifi>=2017.4.17 in c:\anaconda\envs\manisha\lib\site-packages (from requests->r requirements.txt (line 6))
Requirement already satisfied: markupsafe>=2.0.2 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: bleach>=1.5.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Requirement already satisfied: tensorflow<1.5.0,>=1.4.0 in c:\anaconda\envs\manisha\lib\site-packages (from tensorflow-gpu>=1.4.0->r requirements.txt (line 1))
Building wheels for collected packages: python-Levenshtein
  Running setup.py bdist_wheel for python-Levenshtein ... done
  Stored in directory: C:\Users\manisha\AppData\Local\pip\Cache\wheels\de\c2\03\660f5f7559046280d2c6d81c4e9e36518766e7e342
Successfully built python-Levenshtein tldm, regex, python-Levenshtein
Installing collected packages: tldm, regex, python-Levenshtein
```

Figure 5-26. Installing the requirements

You also need to manage the requirements:

```
pip install -r requirements.txt
```

Now you open the set up folder:

```
cd setup
```

Then start preparing the data:

```
python prepare_data.py
```

After preparing the data, you will get one level closer to the root in the folder:

```
cd ..
```

CHAPTER 5 CHATBOT WITH TENSORFLOW

Now you start training:

```
python train.py
```

Figure 5-27 shows the training process.

```
Epoch: 3, steps per epoch: 921, epoch ends at 2763 steps, learning rate: 1e-05 - training
# Job id 0
# Loading hparams from model/hparams
# Updating hparams.num_train_steps: 1842 -> 2763
# Updating hparams.learning_rate: 0.0001 -> 1e-05
# Updating hparams.epoch_step: 921 -> 0
saving hparams to model/hparams
saving hparams to model/best_bleu\hparams
attention=scaled_luong
attention_architecture=standard
avg_ckpts=False
batch_size=128
beam_width=20
best_bleu=0
best_bleu_dir=model/best_bleu
check_special_token=True
colocate_gradients_with_ops=True
decay_scheme=
dev_prefix=data/tst2012.bpe
dropout=0.2
embed_prefix=None
encoder_type=bi
eos=</s>
epoch_step=0
forget_bias=1.0
infer_batch_size=32
init_op=uniform
init_weight=0.1
learning_rate=1e-05
length_penalty_weight=1.0
log_device_placement=False
max_gradient_norm=5.0
max_train=0
metrics=['bleu']
num_buckets=5
num_decoder_layers=2
num_decoder_residual_layers=0
num_embeddings_partitions=0
num_encoder_layers=2
num_encoder_residual_layers=0
num_gpus=1
num_inter_threads=0
num_intra_threads=0
num_keep_ckpts=5
num_layers=2
num_train_steps=2763
num_translations_per_input=20
num_units=512
optimizer=adam
out_dir=model/
output_attention=True
override_loaded_hparams=True
pass_hidden_state=True
random_seed=None
residual=False
sampling_temperature=0.0
share_vocab=True
sos=<s>
src=from
src_embed_file=
src_max_len=50
```



Figure 5-27. Training process

After training is finished, you will get a message. Figure 5-28 shows that the training process is complete.

Training finished

Figure 5-28. The training process is complete

Use `inference.py` to interact directly with the bot.

```
python inference.py
```

Figure 5-29 shows the inference models.

```
(enviro) C:\Windows\system32\cmd.exe: python inference.py
C:\Anaconda\envs\manishali\lib\site-packages\tensorflow\contrib\learn\python\learn\datasets\base.py:198: FutureWarning: Conversion of the second argument of issubclass from 'float' to 'np.floating' is deprecated. In future, it will be treated as 'np.float64 == np.dtype(float)'.
  from __import__ register_converters as _register_converters

Starting interactive mode (first response will take a while):
[1]:
```

Figure 5-29. Inference models

End-to-End Systems

End-to-end machine learning is the best approach for chatbots as we ingest the data through the chatbot and then score the test data. This type of machine learning helps in taking the decisions very faster for response that is very swift to communicate between user and the bot. One system is trained on one dataset. The chatbot makes no assumptions of the use cases and the dialogue and trains it on relevant data and have conversation of the data with user. Use Feed Forward Neural Network to implement it in Deep Learning.

Before explaining further, you need to know about recurrent neural networks.

Recurrent Neural Network

Recurrent neural networks (RNNs) are useful in learning scenarios based on natural language processing.

Predicting the next word in a sequence is tough. That's why we need to know the sequence of words prior to making a prediction.

RNNs are called *recurrent* because the same principle is applied to every element (phrase or a word) in a sequence, where the output is based upon previous computations.

To better understand RNNs, consider this example. Say you have a sentence of seven words. In an RNN, you'd have to break the neural network into seven different layers, with one layer for each word.

The most common RNNs are Long short-term memory networks (LSTMs).

LSTMs

To resolve long-term dependencies in RNNs, you need LSTMs. These networks are capable of learning long short-term dependencies in a sequence for better predictions of the output. Figure 5-30 shows a standard RNN with one layer.

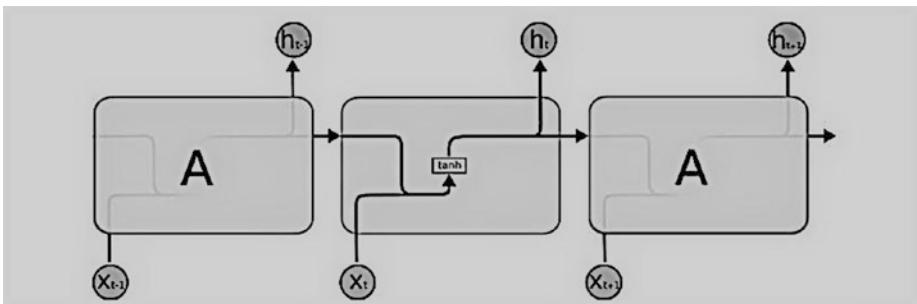


Figure 5-30. RNN

LSTMs contain four neural network layers in a network, as illustrated in Figure 5-31.

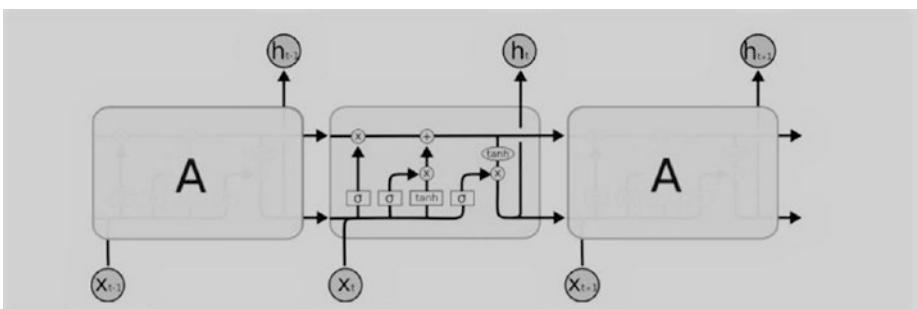


Figure 5-31. LSTM with neural network layers

The key concept behind LSTMs are the cell states. Figure 5-32 shows the usage of cells.

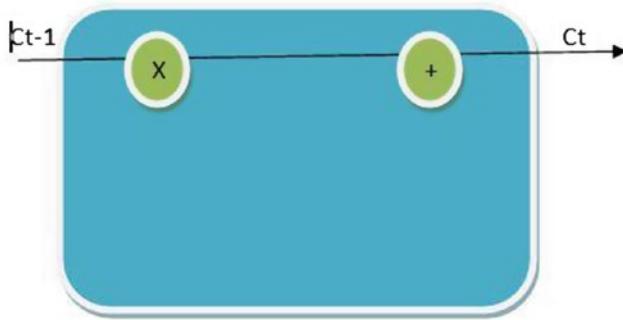


Figure 5-32. Using the cells. The cell states formulate the way the communication works

The Sigmoid layer in the LSTM outputs a value between 0 and 1. A value of 1 means you let all information go through it. A value of 0 means that nothing goes through it. Knowing these concepts, you can move along to the Seq2seq model.

Seq2seq model has

- i)Encoder
- ii)Decoder
- iii)Intermediate State

The flow is shown in Figure 5-33.

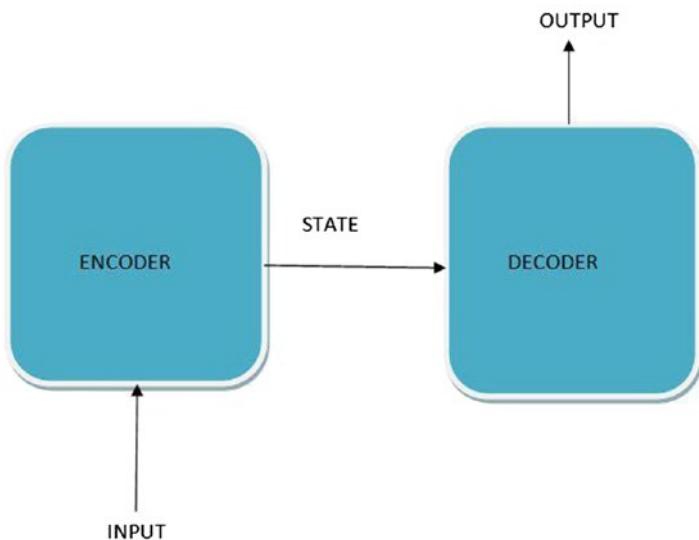


Figure 5-33. Encoder-Decoder model

We generally use embeddings. So to recognize a sentence after that for doing prediction we have to make a vocabulary of words that are to be fed to the model to be read. Figure 5-34 shows the message process.

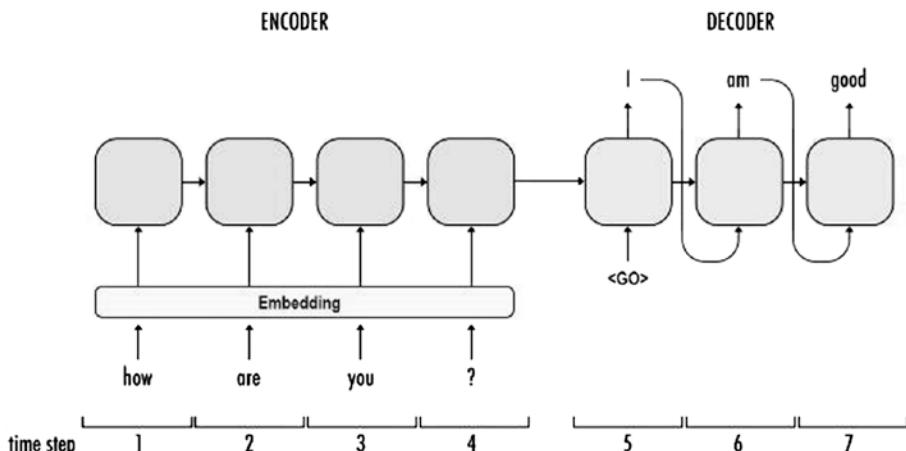


Figure 5-34. Message output process

Seq2seq vocabulary works in the following way:

- <PAD>: During training, you need to feed your examples to the network in batches. The inputs in these batches all need to be the same width for the network to do its calculation. Our examples, however, are not of the same length. That's why you'll need to pad shorter inputs to bring them to the same width of the batch.
- <EOS>: This is another necessity of batching as well, but more on the decoder side. It allows us to tell the decoder where a sentence ends, and it allows the decoder to indicate the same thing in its outputs.
- <UNK>: If you're training your model on real data, you'll find you can vastly improve the resource efficiency of your model by ignoring words that don't show up often enough in your vocabulary to warrant consideration. We replace those with <UNK>.
- <GO>: This is the input to the first time-step of the decoder to let the decoder know when to start generating output.

Working with a Seq2seq Bot

In this section, you will see how to work with a Seq2seq bot. We will clone or copy one of the GitHub repo locally and then run the bot accordingly.

First you need to clone the repo to get inside the folder:

https://github.com/l1Sourcell/tensorflow_chatbot

Use this command to prepare the data:

`Python prepare_data.py`

Then open the seq2seq.ini file and change the mode to training as.
Use this command:

`python execute.py`

Figure 5-35 shows the training process.

```
(idpFull) C:\Users\abhis\Desktop\Chatbot\tensorflow_chatbot\tensorflow_chatbot>python execute.py
>> Mode : train

Preparing data in working dir/
2018-04-19 20:14:48.409515: I C:\tf_jenkins\workspace\rel-win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
WARNING:tensorflow:From C:\Users\abhis\Anaconda3\envs\idpFull\lib\site-packages\tensorflow\python\ops\nn_impl.py:1310: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.
See tf.nn.softmax_cross_entropy_with_logits_v2.

See tf.nn.softmax_cross_entropy_with_logits_v2.
```

Figure 5-35. Training process

After training, go back into the seq2seq.ini file and update the mode to testing.

When you start testing, The bot will workaround find the checkpoints from training and start communicating, as shown here and in Figure 5-36:

```
>> Mode : test
2018-04-20 00:15:14.408168: I C:\tf_jenkins\workspace\rel-
win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_guard.
cc:137] Your CPU supports instructions that this TensorFlow
binary was not compiled to use: AVX AVX2
WARNING:tensorflow:From C:\Users\abhis\Anaconda3\envs\idpFull\
lib\site-packages\tensorflow\python\ops\nn_impl.py:1310:
softmax_cross_entropy_with_logits (from tensorflow.python.ops.
nn_ops) is deprecated and will be removed in a future version.
```

CHAPTER 5 CHATBOT WITH TENSORFLOW

```
>> Mode : test
2018-04-20 00:15:14.400368: I C:\1\1_TensorFlow\workspace\test\win\Windows\PT\3\TensorFlow\contrib\platform\cpu\feature_giant.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
RCRQDNGTensorFlow:From C:\Users\shah\Anaconda\envs\DeepLearn\lib\site-packages\tensorflow\python\ops\nn_ops.py:1310: softmax_cross_entropy_with_logits [From tensorflow.python.ops.nn_ops]
Instructions for updating:
Future major versions of TensorFlow will allow gradients to flow into the
labels input on backprop by default.
See tf.nn.softmax_cross_entropy_with_logits_v2.
Reading model parameters from working_dir/seq2seq.ckpt-4200
> hello
```

Figure 5-36. Testing the bot

Instructions for Updating

Future major versions of TensorFlow will allow gradients to flow into the labels input on backprop by default. There are some expected changes in the later version of Tensorflow.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

Reading model parameters from `working_dir/seq2seq.ckpt-4200`.

We are using the `ckpt` file created after training to see how the trained model works

```
> hello
```

Install NVIDIA's card on your computer along with drivers.

1. Download and install CUDA.
2. Download and “install” cuDNN.
3. Uninstall TensorFlow, and install Tensorflow GPU.
4. Update the %PATH% on the system.
5. Verify installation.

Download and Install CUDA

CUDA has different versions. You need CUDA version 8.0. I have 8.0, 9.0, and 9.1 installed and set up identically to this guide for each version. Stick with 8.0 for now to get that working. I set up the other versions to prepare for the possibility of TensorFlow GPU supporting other CUDA versions.

Here are the steps to download and install CUDA:

Go to [CUDA Toolkit downloads](#).

1. Scroll down to Legacy Releases.
2. Click the version you want (CUDA Toolkit X.Y):
 - For 8.0, you'll see CUDA Toolkit 8.0 GA, so replace *<Z>* with the highest number available.
Z is the version that is available. I downloaded CUDA Toolkit 8.0 GA2.
 - For 9.0, the file is CUDA Toolkit 9.0
 - For 9.1, the file is CUDA Toolkit 9.1.
3. Select your operating system. Mine is as follows:

OS: Windows

Architecture: x86_64

Version: 10

4. After CUDA downloads, run the downloaded file and install it with Express Settings. This might take a while and flicker the screen (because of the graphics card).
5. Verify that you have the following path on your system:

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0

Download and Install cuDNN

To install the Cuda Deep Neural Network library (cuDNN), you need an NVIDIA developer account. It's free.

Create a free NVIDIA Developer Membership [here](#):

After you sign up, go to <https://developer.nvidia.com/cudnn>. Then follow these steps:

1. Click the Download cuDNN button (ignore the current listed version for now). Agree to the Terms.
2. Remember how above we need cuDNN v6.0 from above? You might see this listed here, or you might not. If you don't, just select Archived cuDNN Releases.
3. Click the version you need as well as the system you need.
4. Download cuDNN v6.0 (April 27, 2017) for CUDA 8.0, and then download cuDNN v6.0 Library for Windows 10. Unzip your recent downloaded zip file, such as:
`C:\Users\teamcfe\Downloads\cudnn-8.0-windows10-x64-v6.0.zip`
5. Open cuda and you should see the following:

`bin/
include/
lib/`

6. Copy and paste the three folders from `C:\Users\j\Downloads\cudnn-8.0-windows10-x64-v6.0.zip\cuda` to `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0`.

Note that dragging and dropping will merge the folders and not replace them; I don't believe the same is true for Mac/Linux. If Cuda asks you to replace anything, say No and just drag and drop each folder's contents from cuDNN to Cuda. Cuda might ask about admin privileges, in which case, you should just say Yes.

7. Verify that you did the last step correctly. If you did, you should be able to find this path:

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\  
v8.0\lib\x64\cudnn.lib
```

Uninstall TensorFlow, Install TensorFlow GPU

You can remove TensorFlow from your system if it's currently installed by using this command:

```
pip uninstall tensorflow
```

You want to use TensorFlow with GPU support, and doing that is easy:

```
pip install tensorflow-gpu
```

I'm glad that was easy. :)

Update the %PATH% on the System

Update your system environment variables' PATH to have the following:

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\bin  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\libnvvp
```

Conclusion

In this chapter, you used TensorFlow to create chatbots. You found out that for deep learning chatbots, LSTM is the best technique. This chapter also introduced Keras, and you built a chatbot with the Keras wrapper and TensorFlow as the back end. Finally, you looked at some common chatbots and reviewed a Seq2seq model approach to creating chatbots.

Index

A

Artificial intelligence (AI)
chatbots (*see* Chatbots)
computer vision, 2
definition, 1
DL, 4, 12
image processing, 2
ML (*see* Machine learning (ML))
neural networks, 4, 12–13
NLP, 1
pattern recognition, 2
robotics, 2
sales *vs.* advertisement data, 5, 6
speech recognition, 1
symbolic-based and data-based
approaches, 4–5

B

Bot code logic, 44

C

Chatbots, 14
applications
Botnets, 20
CNN bot, 19
Facebook Messenger, 19

Poncho, 19
reinforcement
learning bots, 20–21
Telegram, 19
Twitter, 20
web spiders bots, 19–20
WhatsApp, 19
conversational commerce, 17–18
dialog flow/form flow, 22
digital brain, 15–17
entities, 22
frameworks, 23
generative, 14–15
parts, 15
retrieval-based model, 14
structure, 21–22
Turing test, 14
CNN bot, 19
CoffeeBot
dialogs
adding, 129
BuyCoffee, 130–132
Suggestion node, 132–133
workspace, 129–130
entities
adding, 126–127
CoffeeOptions, 128
CoffeeSize, 127–128

INDEX

- CoffeeBot (*cont.*)
 intents
 BuyCoffee, 121–122
 Cancel, 126
 creating, 118–119
 Greetings, 119
 Suggestion, 122–123
 ThankYou, 125
 user examples, 120
 Yes, 124
 nested intents
 adding, 134
 canceling order, 135
 child node, 133–134
 options for small, medium,
 and large, 135
 workflow, 136
 workspace
 creating, 117
 naming, 118
Cognitive computing, 4
Conversational commerce, 17–18
Convolution neural network, 4
Cuda Deep Neural Network library
(cuDNN), 180–181
 integrations, 98–100
 intents, 91
 linking to Google project, 90–91
 logging in, Google account, 85
 parameters, 92
 Questionbot
 copy intents, 95–96
 prebuilt agents, 94
 questions and answers,
 96–97
 select all the intents, 95
 Small Talk, 88–90
 web site, 85
 welcome page, 86–87
Dialogs
 adding new item, 49–50
 creating folder, 48–49
 ManishaBot, 47
 RandomFactDialog class
 chaining messages, 51–52
 chaining process, 54–56
 referencing, 51
 response for bot, 52
 series of messages, 53–54
Digital brain, 15–17

D, E

- Deep learning (DL), 4, 12
DeepMind’s algorithm, 20
Dialogflow
 access Goggle Assistant, 86
 creating, PizzaBot, 87
 entities, 92–94

F

- FAQ bot
 creating workspace, 106–107
 dialog flow, 112–113
intent
 adding description, 109
 Capabilities, 110

creating, 108
 listed, 112
 Migration, 110, 114
 naming, 109
 SSO, 111, 115
 User, 111, 114–115
 Try It Out link, 115–117
 workspace name, 107–108

G, H

Generative chatbot model, 14–15

I, J

IBM Cloud
 create resource, 104
 login window, 102–103
 main console window, 103
 main page, 102
 Watson Assistant
 service, 104–105
 IBM Watson Assistant
 CoffeeBot, 117–136
 FAQ Bot, 106
 Image processing, 2

K

Keras
 chatbot
 accumulating data, 164
 chat option, 166
 Cornell Movie-Dialogs
 Corpus, 163

downloading keras, 164
 interaction, 166
 optimizer, 165
 training data, 164–165
 iris dataset, 161
 libraries, 159–160
 logistic regression, 162–163
 neural networks, 159
 plotting species, 161–162
 Scikit classifier, 162–163
 training and testing datasets, 162

L

Language Understanding and Intelligent Service (LUIS), 25
 Long-short term memory (LSTM)
 cell states, 173–174
 encoder-decoder model, 175
 message process, 175
 neural network layers, 173
 Seq2seq model, 174–176

M

Machine learning (ML)
 AI and DL, 10
 automation, 11
 classification, 8
 computational tipping point, 11
 dataset, 2
 definition, 4
 flows, 2, 3

INDEX

- Machine learning (ML) (*cont.*)
high-dimensional space, 7
patterns, 6
reinforcement learning, 9
supervised learning, 9
unsupervised learning, 9
- Managing states, 46–47
- Microsoft Bot Framework
Bot Builder, 27–28
dialogs (*see* Dialogs)
emulator
download page, 28–29
EXE file, 29–30
managing states, 46–47
publishing, Azure
Cloud, 56–65
running the application, 42–43
template
creating new project, 31
parameters, 32
Visual Studio IDE
screen, 30–32
testing, 44–46
Visual Studio, 26
Windows 10, 27
working with code
BotId, MicrosoftAppId, and
MicrosoftApp Password
values, 37
Configuration tab, 37
files, 33
length of characters, 38
MessageController.cs file,
38–42
reply message code block, 38
web.config
XML file, 34–35, 37
- ## N, O
- Natural language processing (NLP), 1
- Neural networks, 4, 12–13
nmt-chatbot
Anaconda versions, 167
cloning repo, 168
inference models, 171
requirements, 169
Tensorflow-gpu, 167–168
training process, 170–171
- ## P, Q
- Pattern recognition, 2
Poncho, 19
Publishing bot, Azure Cloud
App Service screen, 63
Azure web page, 64
Bot Builder option, 58–59
configuration option, adding
details, 65
creating bot, 57–58
details, 59
framework page, 56–57
generating App ID and
password, 60–61
My Bots option, 57
preparing, 61–62
validation, 63–64

R

Recurrent neural networks (RNNs), 4
 layer, 172
 LSTM (*see* Long-short term memory (LSTM))
 Reinforcement learning, 9, 20–21

S

Seq2seq bot
 testing, 177–178
 training process, 177
 Small Talk, 88–90
 Speech recognition, 1
 Statistical learning, 1
 Supervised learning, 9

T

TensorBoard
 description, 151
 enabling Python mode, 154
 multiplication analysis, 156–157
 output, 155–156
 running code, 155
 session with
 TensorFlow, 151–154
 visualization, 157–158
 TensorFlow
 activation function
 enabling environment, 149
 layer function, 150
 matrix multiplication, 150

TensorBoard

(*see* TensorBoard)

Anaconda environment
 activation, 140
 checking version, 141
 Intel-optimized Python, 140
 working, 141–142
 description, 139
 Keras (*see* Keras)
 neural network, 143–149
 nmt-chatbot, 166–171
 updating
 CUDA, 178, 179
 cuDNN, 180–181
 GPU, 181
 %PATH%, 181
 versions, 158
 workflow, 143
 Turing test, 14

U, V

Unsupervised learning, 9

W, X, Y, Z

Watson
 FAQ bot (*see* FAQ bot)
 IBM Cloud (*see* IBM Cloud)
 IBM web page, 101
 Web spiders bots, 19
 Wit.ai
 adding intent, 70–72
 creating app, 69

INDEX

- Wit.ai (*cont.*)
 - creating entity, 77
 - Facebook
 - adding name, 78
 - developers page, 78–79
 - downloading ngrok for Windows, 82
 - inspect page, 83
 - Messenger option, 80
 - new app, 79
 - ngrok site, 81–82
 - ngrok status, 83
 - SampleApp, 80
 - selecting web page, 81
 - set up, ngrok environment, 84
 - Webhooks, 81
- logging in, GitHub, 68
- Quick Start option, 67–68
- setting details, 69
- text and keywords, 73–76