**Code in chat.py**

**Creating a JSON file**

What we are doing with the JSON file is creating a bunch of messages that the user is likely to type in and mapping them to a group of appropriate responses. The tag on each dictionary in the file indicates the group that each message belongs to. With this data we will train a neural network to take a sentence of words and classify it as one of the tags in our file. Then we can simply take a response from those groups and display that to the user. The more tags, responses, and patterns you provide to the chatbot the better and more complex it will be.

Patterns are the input and responses will be the output.

**Loading the JSON file**

We will start by importing some modules and loading in our json data. Now our json data will be stored in the variable *data*.

**Extracting data**

We need all of the patterns and which class/tag they belong to. Now its time to loop through our JSON data and extract the data we want. For each pattern we will turn it into a list of words using nltk.word\_tokenizer.

Words : List of all the words in the pattern (input)

Labels : List of all the unique tags name

Docs\_x : List of all the words in the pattern (input)

Docs\_y : List of its associated tag

**Word stemming**

Stemming a word is attempting to find the root of the word. For example, the word "thats" stem might be "that" and the word "happening" would have the stem of "happen". We will use this process of stemming words to reduce the vocabulary of our model and attempt to find the more general meaning behind sentences.

**Bags of words**

Now that we have loaded in our data and created a stemmed vocabulary it's time to talk about a **bag of words**. As we know neural networks and machine learning algorithms require numerical input. So our list of strings won't cut it. We need some way to represent our sentences with numbers and this is where a bag of words comes in. What we are going to do is represent each sentence with a list the length of the amount of words in our models vocabulary. Each position in the list will represent a word from our vocabulary. If the position in the list is a 1 then that will mean that the word exists in our sentence, if it is a 0 then the word is nor present. We call this a bag of words because the order in which the words appear in the sentence is lost, we only know the presence of words in our model's vocabulary.

As well as formatting our input we need to format our output to make sense to the neural network. Similarly to a bag of words we will create output lists which are the length of the amount of labels/tags we have in our dataset. Each position in the list will represent one distinct label/tag, a 1 in any of those positions will show which label/tag is represented.

**Develop a Model**

Now that we have preprocessed all of our data we are ready to start creating and training a model. For our purposes we will use a fairly standard feed-forward neural network with two hidden layers. The goal of our network will be to look at a bag of words and give a class that they belong too (one of our tags from the JSON file).

**Training and saving the model**

Now that we have set up our model it's time to train it on our data! To do these we will **fit** our data to the model. The number of epochs we set is the amount of times that the model will see the same information while training.

Once we are done training the model we can save it to the file **model.tflearn** for use in other scripts.

**Making prediction**

Ideally we want to generate a response to any sentence the user types in. To do this we need to remember that our model does not take string input, it takes a bag of words. We also need to realize that our model does not spit out sentences, it generates a list of probabilities for all of our classes. This makes the process to generate a response look like the following: – Get some input from the user – Convert it to a bag of words – Get a prediction from the model – Find the most probable class – Pick a response from that class

The bag\_of\_words function will transform our string input to a bag of words using our created words list. The chat function will handle getting a prediction from the model and grabbing an appropriate response from our JSON file of responses.

Now run the program and enjoy chatting with your bot!