Categorizer Documentation

<https://github.com/LinkaiShao/Categorizer>

Linkai Shao

# 0 Table of contents

Contents

[0 Table of contents 1](#_Toc152792756)

[1 Abstract 2](#_Toc152792757)

[2 Approach 2](#_Toc152792758)

[2.1 generating training data 2](#_Toc152792759)

[2.2 representing the input and the categories 2](#_Toc152792760)

[2.3 Setting up the neural network for training 2](#_Toc152792761)

[3. Simple Diagrams 2](#_Toc152792762)

[3.1 Simple Training Diagram 2](#_Toc152792763)

[3.2 Simple Categorizing Diagram (Running) 3](#_Toc152792764)

[4 Pulling training data 5](#_Toc152792765)

[4.1 Diagram (Simple) 5](#_Toc152792766)

[4.2 Asking gpt 6](#_Toc152792767)

[4.2.1 sending the message to gpt 6](#_Toc152792768)

[4.2.2 Error handling 6](#_Toc152792769)

[4.2.3 Berryessa algo 9](#_Toc152792770)

[4.2.4 Run time 9](#_Toc152792771)

[4.3 Complex Diagram 11](#_Toc152792772)

[5 transforming input and outputs 12](#_Toc152792773)

[5.1 Setup 12](#_Toc152792774)

[5.2 Simple diagram 12](#_Toc152792775)

[5.3 Word embeddings 13](#_Toc152792776)

[5.4 Tokenizing 13](#_Toc152792777)

[5.4.1 Determine the language 13](#_Toc152792778)

[5.4.1. What languages to tokenize for 14](#_Toc152792779)

[5.4.2 Generating word embedding 15](#_Toc152792780)

[5.5 Complex Diagram 17](#_Toc152792781)

[6 Developing the neural network 18](#_Toc152792782)

[6.1 Setting up training data for the neural network 18](#_Toc152792783)

[6.1.1 Handling constant influx of addition training data 19](#_Toc152792784)

[6.2 Setting up the neural network 19](#_Toc152792785)

[6.2.1 neural network structure 19](#_Toc152792786)

[6.2.2 initial model: triple dense layer 21](#_Toc152792787)

[6.2.3 Dropout 22](#_Toc152792788)

[6.2.4 L2 normalization 23](#_Toc152792789)

[7. Running the categorizer 24](#_Toc152792790)

[7.1 Samples 24](#_Toc152792791)

[7.1.1 Diagram of steak sauce being processed 26](#_Toc152792792)

[7.1.3 Finalized Diagram of categorizing a large amount of items 27](#_Toc152792793)

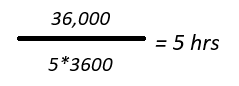
# 1 Abstract

This project focuses on the categorization of item descriptions into predefined categories to enhance user information retrieval. The item descriptions encompass diverse languages, and the training dataset consists of thousands of unclassified item descriptions. The primary objective is to develop artificial intelligence capable of accurately assigning categories to item descriptions.

# 2 Approach

## 2.1 generating training data

The objective is to educate an artificial intelligence capable of accurately categorizing items. To facilitate the learning process, the AI requires a collection of descriptions paired with corresponding categories, referred to as endpoints. However, relying on human input to generate these correct endpoints is impractical due to the time involved. Assuming it takes a human 3 seconds to process each input and an additional 2 seconds to type the matching category, creating endpoints for 36,000 training data points would be excessively time-consuming.



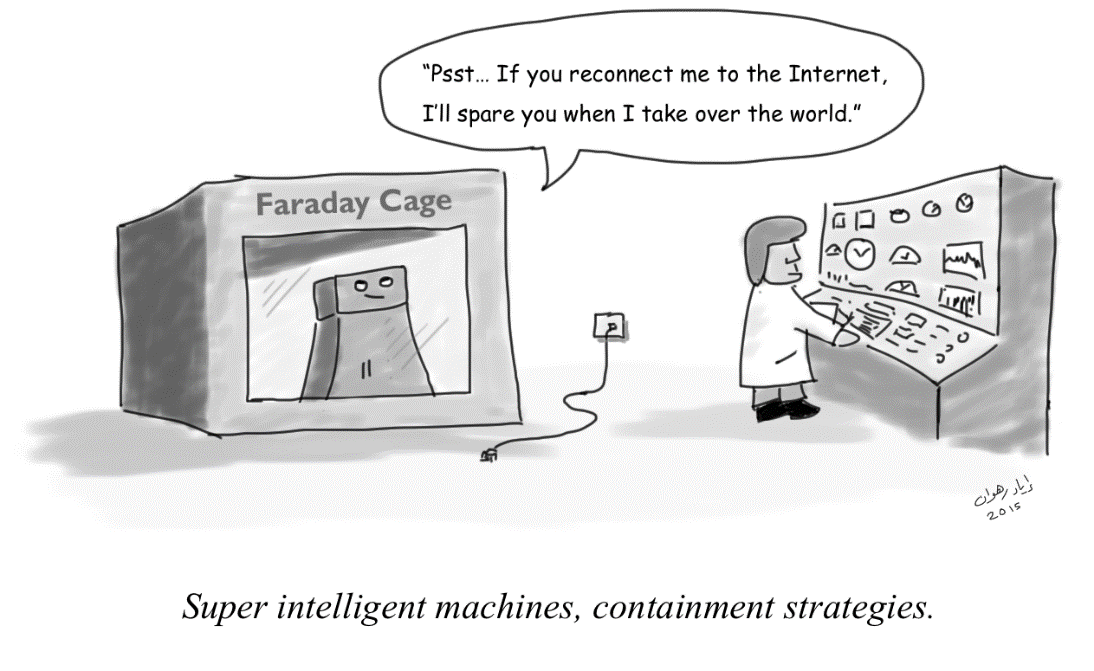
Instead, we can use chat gpt’s api. Assuming the item description is x, we can ask gpt “what category does x belong to”.

## 2.2 representing the input and the categories

Considering that machines lack an inherent understanding of strings, it becomes essential to represent inputs and outputs in formats comprehensible to machines. Achieving the appropriate numerical representations involves the use of dictionaries with meaningful values.

## 2.3 Setting up the neural network for training



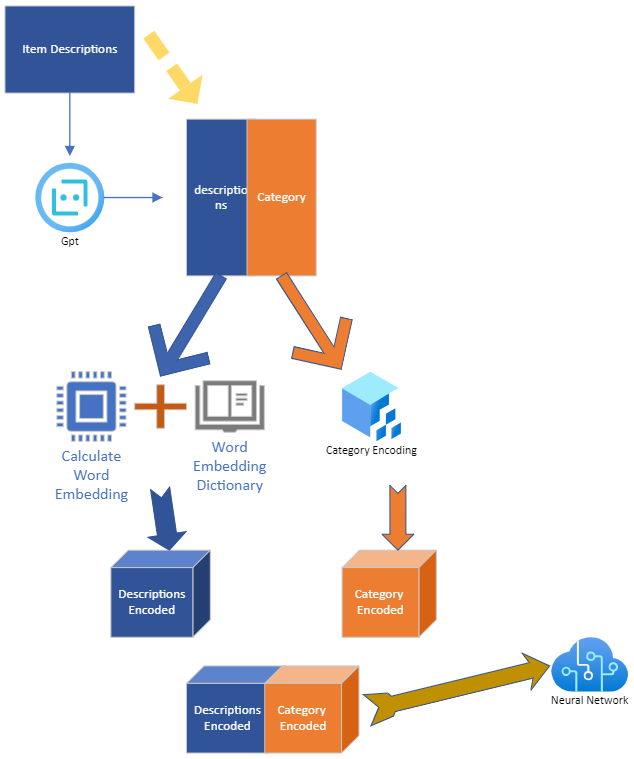
The optimal method for a machine to discern input and classify it into accurate categories is by employing a proficiently trained neural network. In this approach, the neural network is tasked with learning a function, where the input to the function is an item description, and the output corresponds to the correct categories.

[This Photo](https://www.mit.edu/~irahwan/cartoons.html) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

# 3. Simple Diagrams

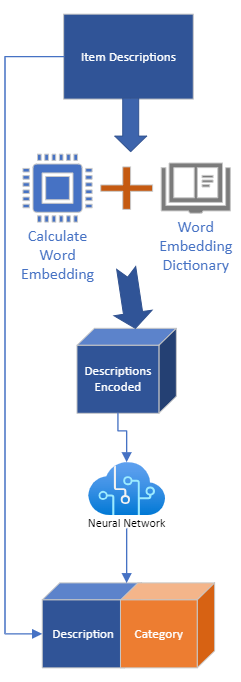
## 3.1 Simple Training Diagram

Diagram for pulling training data + training the neural network



## 3.2 Simple Categorizing Diagram (Running)

Diagram for categorizing an unknown set of item descriptions



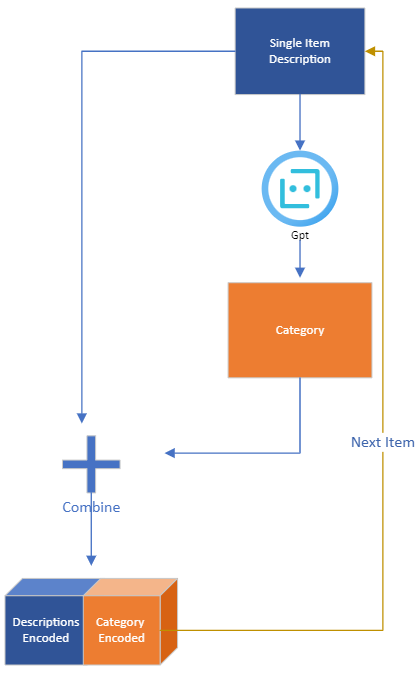
Neural Network Distinguish the item

Generate the Word Embedding for the item

# 4 Pulling training data

We require a dataset for the neural network to familiarize itself with. Instead of manually categorizing items, we will leverage GPT to retrieve training data. While there may be occasional inaccuracies in GPT's responses, the majority of the answers are accurate.

## 4.1 Diagram (Simple)

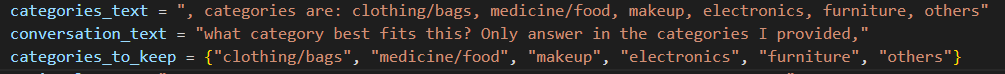


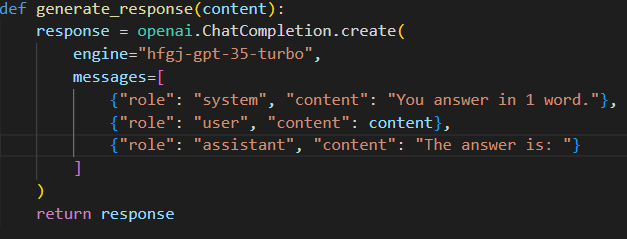
## 4.2 Asking gpt

I was provided with 1 api endpoint with 2 keys. After trying multiple ways to alternate between the 2 keys, it turned out the two keys are of the same properties, so this means that they are considered the same entity and cannot be used simultaneously. Pulling 1 piece of training data on average takes about 2 seconds. To maximize the amount of data being pulled in a set amount of time, I decided to try the Berryessa algorithm used in networking.

### 4.2.1 sending the message to gpt

Given that we only want the category, we ask gpt to answer in 1 word, and the assistant asks what is the category. We also provide gpt with the description and the categories that we are interested in.

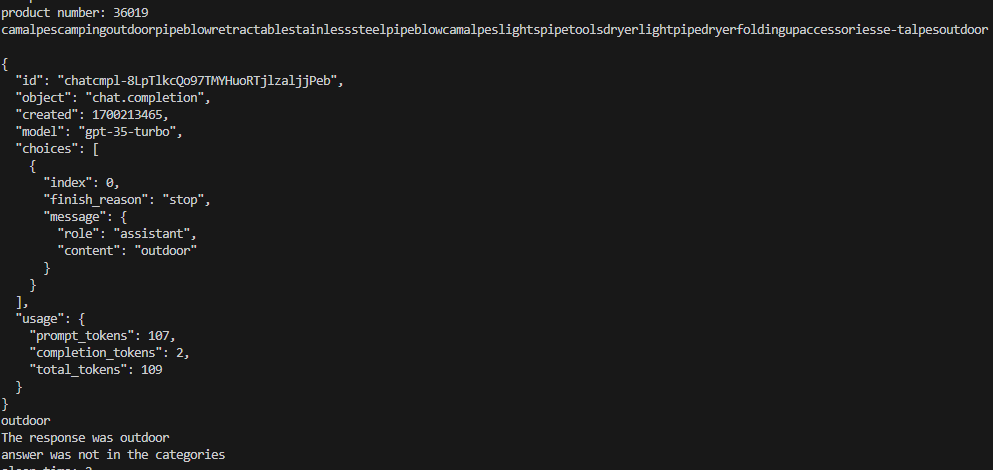




### 4.2.2 Error handling

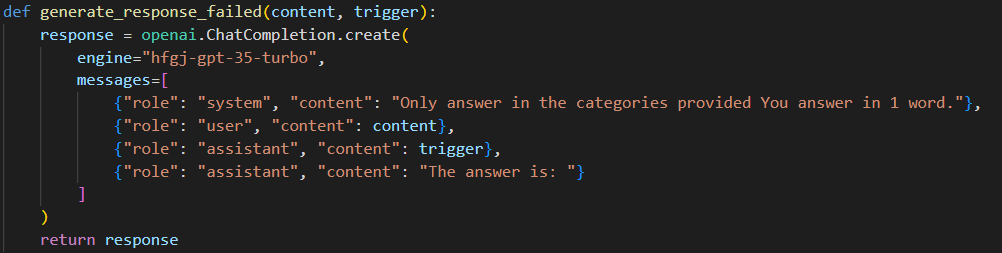
There are 4 kinds of error that we will run into

4.2.2.1 Wrong category



In this example, the user typed in camal camping outdoor camp supplies, which should be categorized as other, given that it is not within one of the categories that we are looking for. Even though we already asked gpt to only answer in the categories that we are looking for, it still sometimes answer outside of the given categories. So it is imperative for us to tell gpt its error and the correct set of categories.

So when the error occurs, we will tell gpt that its previous answer was wrong.

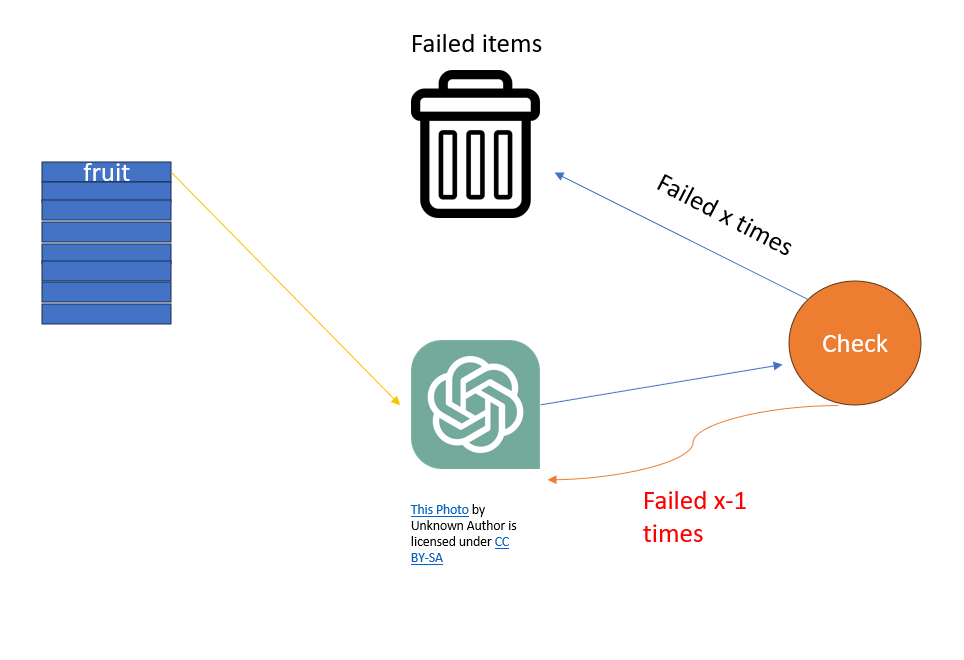


4.2.2.2

Gpt will sometimes return with no answer, which means that within the standard returned json, there was no content within the message. When that happens, we have a limiter, we will ask gpt x more times, if it still return no messages, the item description gets dumped into a failed file, containing the items that could not be categorized

4.2.2.3

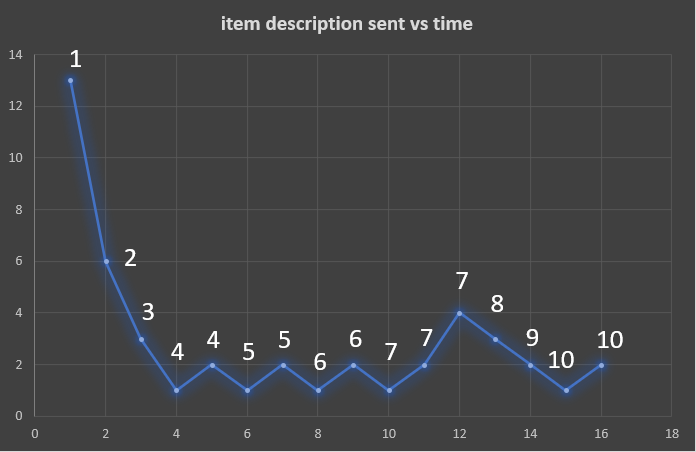
Some items will also violate gpt’s policy, so when that happens, the item goes into garbage collection too



4.2.2.4

Time outs will be handled by sending the item again for gpt

### 4.2.3 Berryessa algo

The Berryessa algorithm found application in package handling within internet service providers (ISPs). Originally designed to maximize the quantity of sent packages, it can also be employed to minimize the time required for request transmissions. When sending descriptions, a waiting period is necessary, as the API key imposes limits on the number of tokens sent within a specific timeframe. Adopting a strategy reminiscent of the Berryessa algorithm for packet transmission, we reduce the wait time by half after successfully sending a request to GPT, and in case of failure, we linearly decrease the wait time.

The graph contains the descriptions sent to gpt for processing for the first 10 descriptions. It shows that on average, the key takes 2 seconds to process a request and no algorithm could decrease that time. So each request was sent to gpt with a 2 second delay.

### 4.2.4 Run time

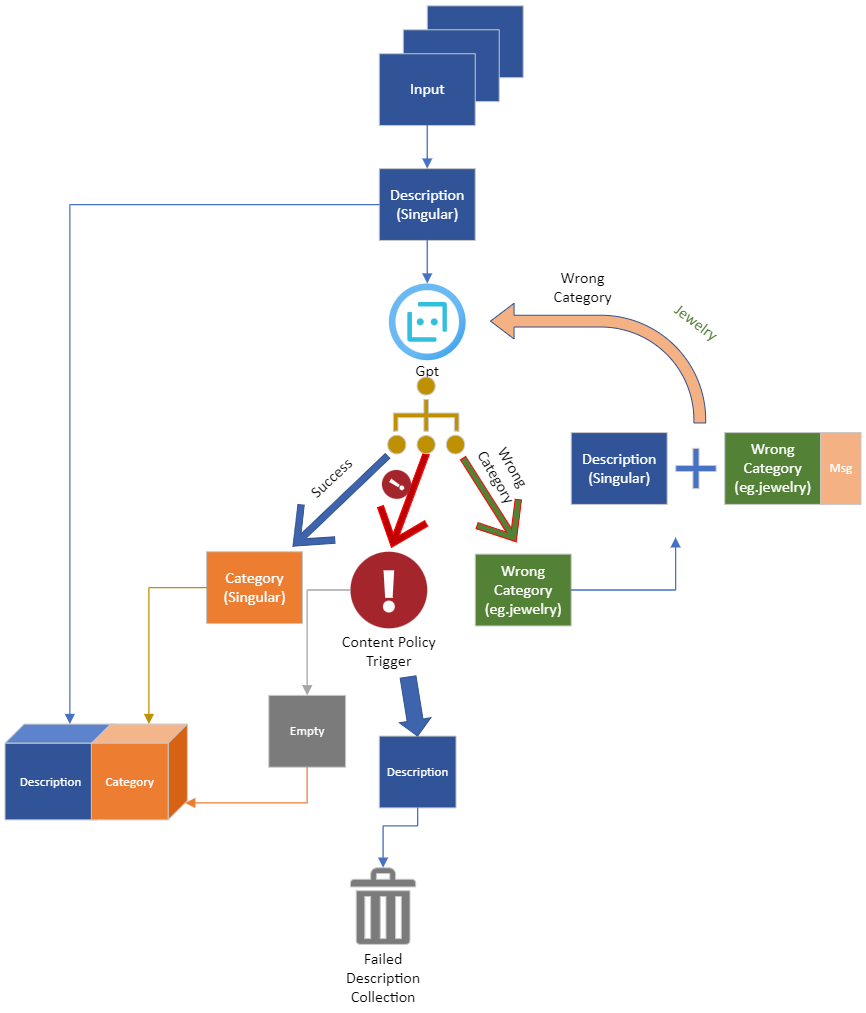
The ideal runtime would be if no issues occur, which means 2 seconds per request.



However, the actual time for the first set of 36,000 descriptions is around 18 hrs.

Out of the 36000 runes, around 1/3 of the item descriptions were sent to gpt >= 2 times. Out of the ones that needed to be resent, majority of them took more than 5 extra sends to obtain a category within the ones that I was asking for.

## 4.3 Complex Diagram

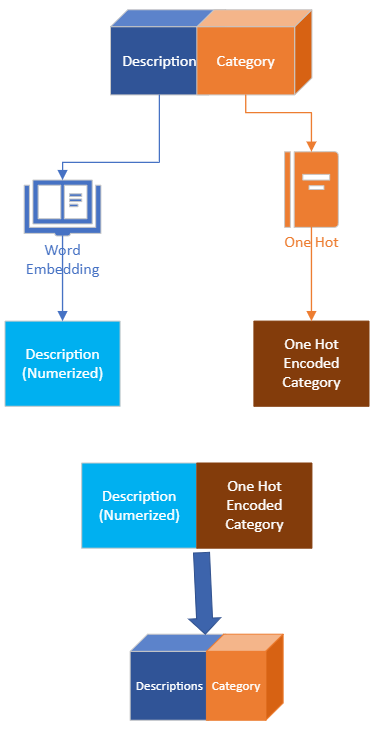


# 5 transforming input and outputs

## 5.1 Setup

We possess a collection of inputs represented as descriptions and corresponding outputs categorized by GPT based on these inputs. We'll presume that GPT generates accurate answers. The objective is to convert both the inputs and outputs into numerical sequences. This transformation aims to ensure that, during neural network training, the numerical representations of inputs and outputs carry meaningful information.

## 5.2 Simple diagram



## 5.3 Word embeddings

Word embeddings are a series of numbers being assigned to each word. The numbers are meaningful and have represent semantic relationships between a word and a description that it belongs to. For example, a 200 degree vector for the word frog is closer to a 200 degree vector of amphibian than a 200 degree vector for bread.

Here is a comparison of frog to amphibian and frog to bread using the cosine function from scipy library

## 5.4 Tokenizing

To give a description its meaning, it is essential to tokenize the text into individual words. Each word is then represented by a vector, and by summing up and averaging these vectors, we derive a single vector that encapsulates the essence of the entire word.

### 5.4.1 Determine the language

Every tokenizer is specifically designed for a particular language. Therefore, the initial step involves identifying the language of the given sentence. To achieve this, we can employ lang detect, an AI capable of language detection. However, in cases where item descriptions involve two different languages, such as 三星phone, lang detect might provide ambiguous results between zh or eng. To address this, in addition to lang detect, a custom detector for non-English root languages becomes necessary.

return (0x3040 <= code\_point <= 0x309F) or (0x30A0 <= code\_point <= 0x30FF)

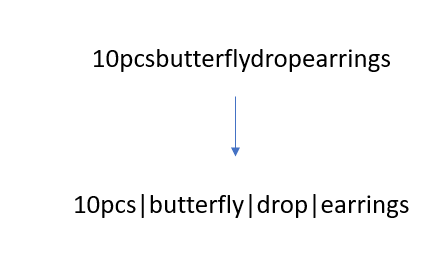
This is the checker for whether a character is of Japanese language. Japanese has three kinds of letters: hiragana, katagana, and kanji. Hiragana falls in the range of 0x3040 to 0x309F, katagana falls in the range 30A0 and 30FF. When detected these letters, we know for sure that the sentence is Japanese.

### 5.4.1. What languages to tokenize for

I wrote a program to tally the amount of most prevalent languages within a random 36thousand sets of item descriptions, the most prevalent languages are English, German, French, Spanish, Italian and Chinese.

5.4.1.2 Tokenizing different languages

For most languages, there are tokenizers that can be used to separate words. We need a tokenizer for separating words because 1: languages like Chinese and Japanese do not contain spaces that can be used to separate a sentence 2: there are users who like to input sentences without spaces in between.



5.4.1.3 Libraries for tokenizing

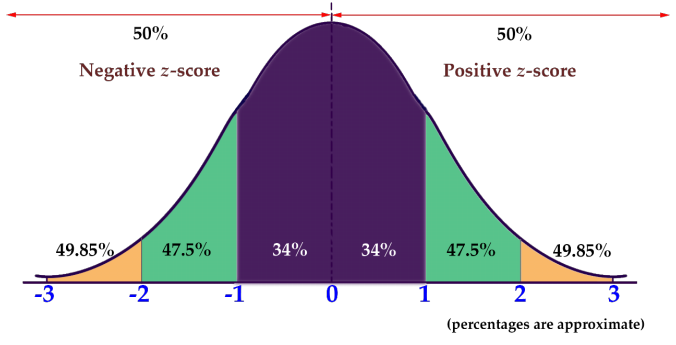
For some languages, there are nlps already built for tokenizing. For some others, custom tokenizers needed to be built.

|  |  |
| --- | --- |
| Language | Tokenizer |
| Chinese | Baidu Lac |
| Japanese | Mecab |
| English | Word ninja |
| Spanish | Word segment |
| Italian | Word segment |
| French | Custom tokenizer |
| German | Custom tokenizer |

### 5.4.2 Generating word embedding

Given that we have every token from the item description, we can now generate word embedding for each word and average it. Word embeddings are matched with files containing word embeddings for specific languages.

5.4.2.1 Z score distribution

 Different word embeddings will generate

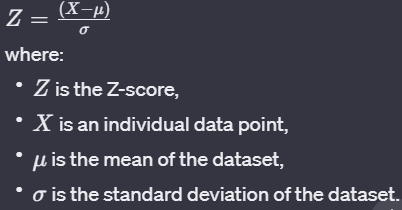
[This Photo](https://courses.lumenlearning.com/math4libarts/chapter/z-scores/) by Unknown Author is licensed under [CC BY-NC](https://creativecommons.org/licenses/by-nc/3.0/)

Vectors of different sizes, we are going to use z-score

Distribution, where every value will be evaluated

Based on its distance from the average.

We are going to use this formula and apply it to the final vector generated in order to keep the vectors between different languages relatively similar



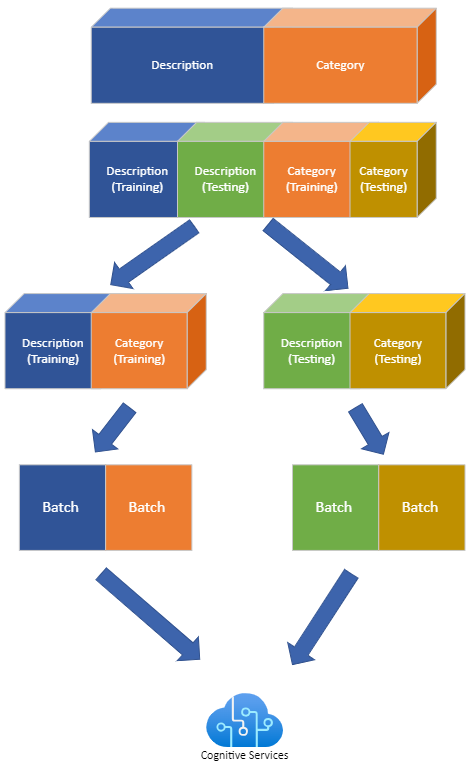
## 5.5 Complex Diagram

# 6 Developing the neural network

With a substantial dataset of numerized descriptions and encoded categories at our disposal, we can commence the training process for the neural network. It's crucial to note that each numerized description corresponds to a one-hot encoded category. The numerized descriptions inherently encapsulate the features of the item description, leaving the neural network with the task of discerning the patterns in the assigned numerical values and understanding the significance of the encoded categories.

## 6.1 Setting up training data for the neural network

We need a set of training data and a set of testing data to test whether or not the model can predict accurately when it comes to data that it has never seen before.

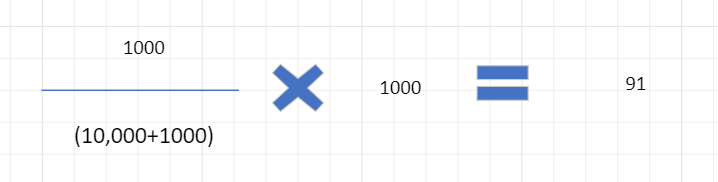


### 6.1.1 Handling constant influx of addition training data

As I am training the neural network, I am still constantly pulling new training data from gpt. If we randomly take data from a large collection of descriptions and categories, it would result in the following problem:

Suppose that we have a new set of training data pulled from gpt, we call it set n. The total amount of data that we have before adding set n is 10,000. For training, we randomly pull 1000 sets.

Suppose the amount of data from set n is 1000, this means that on average, the amount of data from set n that we are going to be using is



Now the next time, the new set of training data we pulled is called n + 1. Suppose that set n + 1 also has 1000 data.

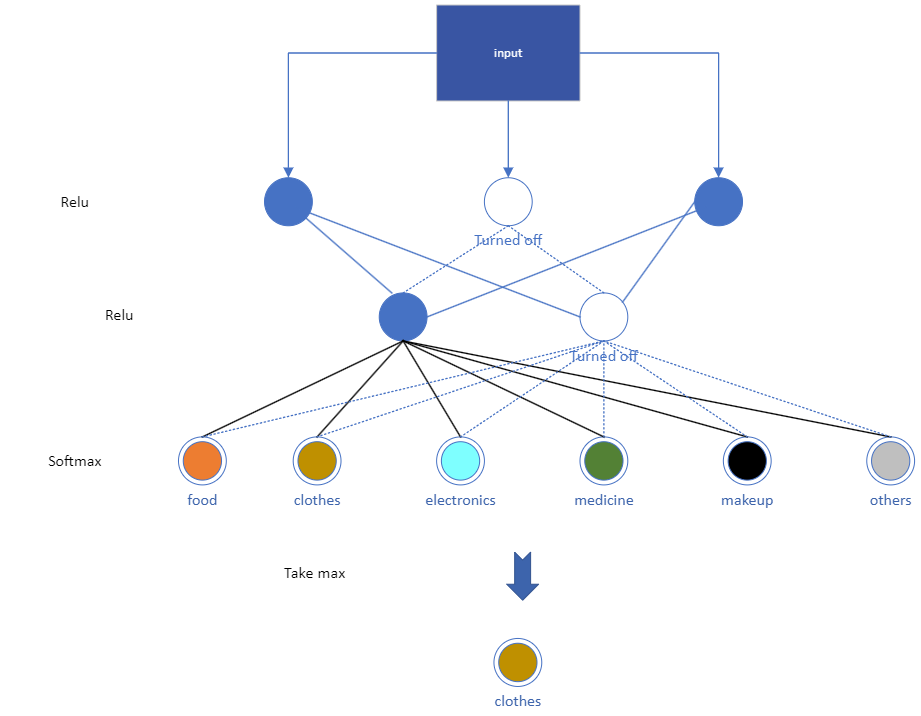


Throughout the two training phases, set n has been sampled 174 times (91 + 83), whereas set n + 1 has been sampled 83 times. To mitigate bias towards earlier sets of data during training, we introduce weighting for all the sets pulled from GPT. The earlier sets are assigned lower weights, while the later ones receive higher weights.

## 6.2 Setting up the neural network

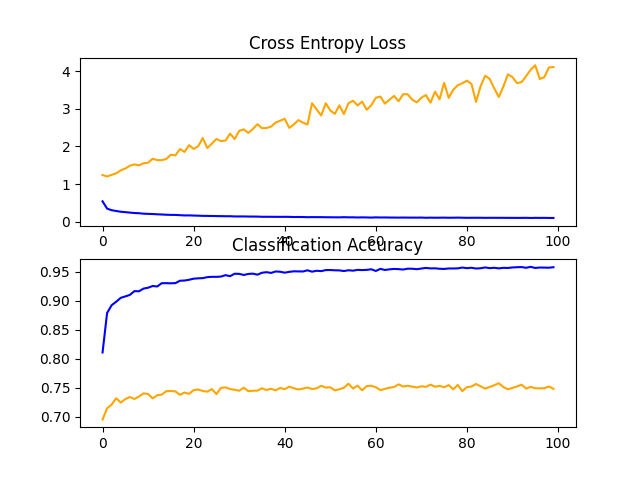
### 6.2.1 neural network structure

Only dense layer is needed. Dense layer is also known as the fully connected layer, has every node connected to every node in the next layer. For each layer, we incorporate an activation function that determines whether or not that node will be fed into the next layer. The final layer will always use softmax, to determine the most likely category.



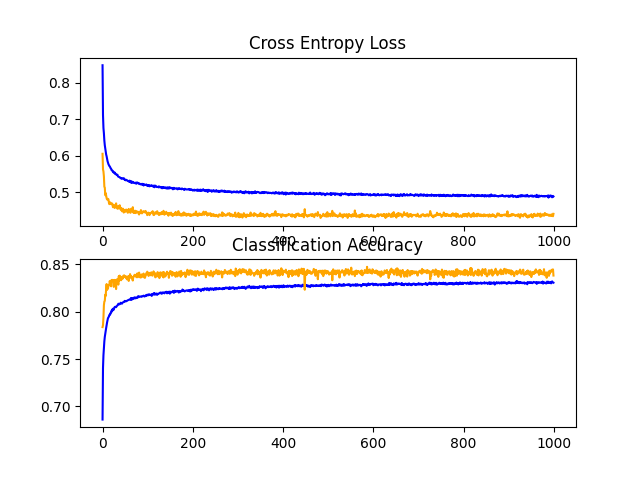
### 6.2.2 initial model: triple dense layer

The initial thought was to make a triple dense layer. The reason that we only need 3 dense layers is because the feature extraction has already been done by the word embeddings. Word embeddings are already trained. The first and second layers will have relu activations while the third will be soft-max. Relu introduces non-linearity and is used in the hidden 2 layers to help my nn learn patterns while soft max generates a probability distribution for my output layer.



This means that our neural network is completely overfitting to the training data set and losing accuracy when it comes to the testing data as shown by the growing loss for testing set.

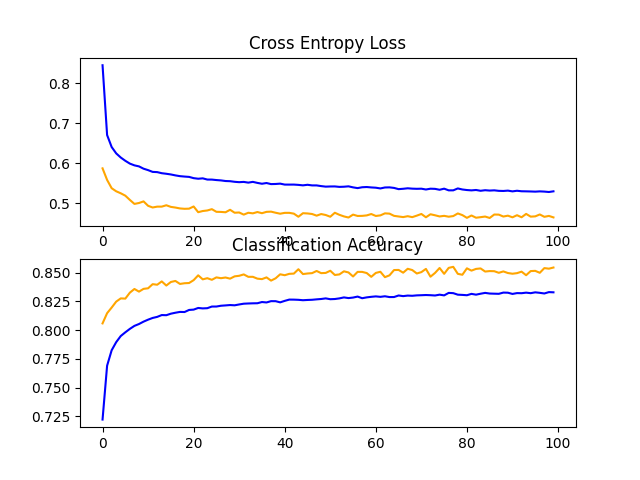
### 6.2.3 Dropout

The model has overly absorbed the training data, even assimilating noise into its learning. To tackle this issue, we will implement binary masking on sets of neurons, setting them to 0. This action enables certain neurons to recalibrate their weights. I'm utilizing a value of 0.4, implying that 40% of the neurons in each hidden layer will be deactivated.

Dropouts have worked nicely, the entropy loss for both training and testing went down.

### 6.2.4 L2 normalization

I have also tried to use l2 normalization, which penalizes large weights. The strength is set to 0.01.



This one also turned out nicely, both loss decreases while accuracy increases.

# 7. Running the categorizer

## 7.1 Samples

Samsung phone

Let’s pass through Samsung phone to the trained neural network categorizer to see what results it will give.

[[2.5742394e-20 8.1670918e-30 1.8250684e-27 1.0000000e+00 9.7027769e-21 2.8081533e-13]]

This is the result.

And these are the one hot categories  
one\_hot\_categories = {

    "clothing/bags": [1, 0, 0, 0, 0, 0],

    "medicine/food": [0, 1, 0, 0, 0, 0],

    "makeup": [0, 0, 1, 0, 0, 0],

    "electronics": [0, 0, 0, 1, 0, 0],

    "furniture": [0, 0, 0, 0, 1, 0],

    "others": [0, 0, 0, 0, 0, 1]

}

It is almost 100% sure that the Samsung phone is electronics. This is likely due to the high amount of Samsung and phones that appeared in the training data.

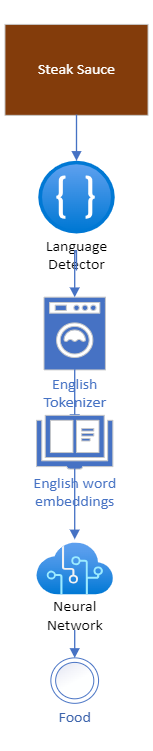
Steak sauce

Ok let’s try steak sauce.

This is what the predictor returned:  
[[0.00323167 0.64706177 0.10600543 0.00307135 0.02726679 0.213363 ]]

It knows that steaks sauce is food, but it is only 65% sure that steak sauce is food, a lot less sure than the Samsung phone. The reason for this is likely due to the fact that Samsung as a word came up a lot during training and gpt would always categorize Samsung products correctly as electronics, whereas steak and sauce came up a lot less often.

### 7.1.1 Diagram of steak sauce being processed



### 7.1.3 Finalized Diagram of categorizing a large amount of items

Every language possesses a corresponding word embedding. The size of each word embedding file is approximately 5 GB. Consequently, loading all language embeddings simultaneously would necessitate 30 GB. To constrain memory allocation for language embeddings to under 16 GB, I plan to segment the input data by language. Once a segment is processed, the corresponding portion of memory is deleted, making room for the next language. This approach ensures that the memory used for word embedding files remains capped at 5 GB.

