

Daniel Terra Gomes, Andre do Valle Medeiros

Inteligência artificial e Chatbots: um estudo exploratório

Campos dos Goytacazes, RJ

23 de outubro de 2022, v1.0.0

Daniel Terra Gomes, Andre do Valle Medeiros

Inteligência artificial e Chatbots: um estudo exploratório

Relatório Atividade 1 apresentado ao Curso
de Ciência da Computação da Universidade
Estadual do Norte Fluminense Darcy Ribeiro,
como requisito avaliativo da disciplina.

Universidade Estadual do Norte Fluminense Darcy Ribeiro

Ciência da Computação

INF01205 - IA 2022

Campos dos Goytacazes, RJ

23 de outubro de 2022, v1.0.0

Daniel Terra Gomes, Andre do Valle Medeiros

Inteligência artificial e Chatbots: um estudo exploratório

Relatório Atividade 1 apresentado ao Curso
de Ciência da Computação da Universidade
Estadual do Norte Fluminense Darcy Ribeiro,
como requisito avaliativo da disciplina.

Campos dos Goytacazes, RJ
23 de outubro de 2022, v1.0.0

*Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.*

*“Behind me lies a farm.
I wonder if there is bread above the hearth
and if I will ever return.
(Pantheon, League of Legends)*

Abstract

This article provides an introduction to AI and Chatbots. Communicating with customers through chat interfaces has, in recent years, become an increasingly popular means of providing real-time customer service in many e-commerce environments. Today, human chat service agents are often replaced by conversational software agents or chatbots, which are systems designed to communicate with human users using Machine Learning (ML) techniques such as Natural Language Processing (NLP), usually based on artificial intelligence (AI). Advances in these areas, particularly in Deep Learning, coupled with the availability of vast computational power and large amounts of data, have led to a new generation of dialog systems and conversational interfaces. Enabling even greater advances in AI. In the practical project developed in this article, we developed a simple Chatbot using 2 neural layers to process messages sent by a user. In this way, it was possible to achieve the desired objective of responding to users with responses from our training base.

Keywords: Chatbots. Artificial Intelligence. Machine Learning.

Resumo

Este artigo fornece uma introdução à IA e Chatbots. A comunicação com os clientes por meio de interfaces de bate-papo, nos últimos anos, tornou-se um meio cada vez mais popular de fornecer atendimento ao cliente em tempo real em muitos ambientes de comércio eletrônico. Hoje, os agentes de serviço de bate-papo humano são frequentemente substituídos por agentes de software de conversação ou chatbots, que são sistemas projetados para se comunicar com usuários humanos usando técnicas de Machine Learning (ML) como o Processamento de Linguagem Natural (PLN), geralmente baseada em inteligência artificial (IA). Avanços nessas áreas, particularmente em Deep Learning, juntamente com a disponibilidade de grande poder computacional e grandes quantidades de dados, levaram a uma nova geração de sistemas de diálogo e interfaces de conversação. Possibilitando avanços ainda maiores em IA. No projeto prático desenvolvido neste artigo elaboramos um Chatbot simples fazendo uso de 2 camadas neurais para o processamento de mensagens enviadas por um usuários. Desse modo, foi possível alcançar o objetivo desejado de responder os usuários com respostas da nossa base de treino.

Palavras-chave: Chatbots. Inteligência Artificial. Machine Learning.

Lista de ilustrações

Figura 1 – Bag of Words	30
Figura 2 – ReLu	33
Figura 3 – ReLu	33
Figura 4 – Feed Forward Neural Network	34
Figura 5 – Feed Forward Neural Network	34
Figura 6 – ReLU v/s Logistic Sigmoid	35
Figura 7 – A representação matemática da função ReLU	36
Figura 8 – A derivada da função ReLU	36
Figura 9 – output	41

Lista de tabelas

Listings

3.1	Python Bibliotecas	28
3.2	Python Funções personalizadas	29
3.3	Python Stemming	29
3.4	Python Bag of Words	30
3.5	Data intents.json	31
3.6	Python Carregando dados	31
3.7	Python Loop nos dados	31
3.8	Data Cleaning	32
3.9	Training Data	32
3.10	Modelo	37
3.11	Dataset	38
3.12	Hiperparâmetros	40
3.13	Perda e otimizador	40
3.14	Treinando o Modelo	40
3.15	Salvando o modelo treinado	42
3.16	Carregando modelo salvo	42
3.17	Chatbot	42

Lista de abreviaturas e siglas

NLP	Natural language processing
ML	Machine Learning
IA	Inteligência Artificial

Sumário

	Introdução	21
1	FUNDAMENTAÇÃO TEÓRICA	23
2	METODOLOGIA	25
3	CONSTRUINDO UM CHATBOT COM PYTORCH	27
3.1	Importando Bibliotecas Relevantes	28
3.2	Criando funções personalizadas	28
3.3	Stemming	29
3.4	Bag of Words	29
3.5	Carregando os dados e Data Cleaning	31
3.6	Limpeza e preparação dos dados	32
3.7	Modelo PyTorch	33
3.7.1	Feed Forward Neural Network	34
3.7.2	Função de ativação	35
3.7.2.1	Função ReLU	35
3.7.2.2	Derivada da Função ReLU	36
3.7.3	Criando nosso modelo	37
3.8	Atribuindo o conjunto de dados ao modelo	37
3.9	Hiperparâmetros	38
3.10	Perda e otimizador	40
3.11	Treinando o Modelo	40
3.11.1	Salvando o modelo treinado	41
3.12	Usando o Chatbot	42
	REFERÊNCIAS	45

Introdução

Nosso projeto propõe a implementação de um Chatbot é um software ou programa de computador que simula conversas humanas por meio de interações de texto ou voz, permitindo que os humanos interajam com dispositivos digitais como se estivessem se comunicando com uma pessoa real. Os chatbots podem ser tão simples quanto programas rudimentares que respondem a uma consulta simples com uma resposta de linha única ou tão sofisticados quanto assistentes digitais que aprendem e evoluem para fornecer níveis crescentes de personalização à medida que coletam e processam informações.

Você provavelmente já interagiu com um chatbot, sabendo ou não. Por exemplo, você está em seu computador pesquisando um produto e uma janela aparece na tela perguntando se você precisa de ajuda. Ou talvez você esteja a caminho de um show e use seu smartphone para solicitar uma carona via chat. Ou você pode ter usado comandos de voz para pedir um café no café do bairro e receber uma resposta informando quando seu pedido estará pronto e quanto custará. ¹.

Esses sistemas de Chats podem ser divididos entre dos tipos; orientados a objetivo (Chatbots) que seriam chats que ajudam o usuários a atingir um certo objetivo e com um campo de conhecimento restrito e, também há os Chit Chat cujo trabalham com perguntas abertas, respostas mais naturais e campo de conhecimento aberto de modo a ajudar a encontrar uma informação (QA System) ².

Um aspecto crítico da implementação do chatbot é selecionar o mecanismo correto de processamento de linguagem natural (NLP) ³. Esse mecanismo conhecido como NLP O refere-se ao ramo da ciência da computação – e mais especificamente, o ramo da inteligência artificial ou IA – preocupado em dar aos computadores a capacidade de entender texto e palavras faladas da mesma maneira que os seres humanos. A NLP combina linguística computacional – modelagem baseada em regras da linguagem humana – com modelos estatísticos, de aprendizado de máquina e aprendizado profundo. Juntas, essas tecnologias permitem que os computadores processem a linguagem humana na forma de texto ou dados de voz e 'compreendam' seu significado completo, completo com a intenção e o sentimento do falante ou escritor ⁴.

Sendo assim, propomos realizar uma implementação de um Chatbot Simples (jump to define) capaz de auxiliar um usuário durante uma compra em um Website. Durante o desenvolvimento deste projeto estaremos realizando uma pesquisa exploratória a fim de

¹ <<https://www.oracle.com/chatbots/what-is-a-chatbot/>>

² <<https://youtu.be/Mu1N-akGL78>>

³ <<https://www.techtarget.com/searchcustomerexperience/definition/chatbot/>>

⁴ <<https://www.ibm.com/cloud/learn/natural-language-processing>>

entender o funcionamento de um Software que aplica NLP. Por consequência, ao final deste projeto teremos implementado e apresentado o nosso sistema de Chatbot, e compreendido as tecnologias para o seu funcionamento. Assim alcançando uma aprendizagem eficaz, autodidata, e exploratória.

1 Fundamentação Teórica

1. Como contribuição teórica inicial para o nosso projeto, temos o material do canal no Youtube 'Python Engineer' a playlist de vídeo aulas, 'Chat Bot With PyTorch - NLP Beginner Tutorial' ¹. A partir desse conteúdo poderemos dar início aos nossos desenvolvimentos e pesquisas. No tutorial, é desenvolvido um chatbot simples usando PyTorch e Deep Learning. Também fornecendo uma introdução a algumas técnicas básicas de Processamento de Linguagem Natural (PLN).
2. Seguiremos, também, os conteúdos já aprendidos nas aulas de 'Inteligência Artificial 2022/2 - UENF' para nos auxiliar na formulação teórica do nosso projeto.
3. Salientamos que com o decorrer do projeto nossas referências de materiais, para esse projeto, tenderá a aumentar devido a novas descobertas.

¹ <<https://www.youtube.com/playlist?list=PLqnsIRFeH2UrFW4AUgn-eY37qOAWQpJyg>>

2 Metodologia

Baseado no “Project-based learning” ([KRAJCIK; BLUMENFELD, 2006](#)). Seguiremos os estudos através de um projeto que aborda problemas do mundo real, cujo muitos não tem resposta única. Ao longo desse projeto será possível fazer novas perguntas e encontrar suas possíveis respostas por meio de uma investigação sustentada.

Este Plano de Pesquisa também utilizará as seguintes metodologias:

- *Pesquisa Exploratória; visando promover o enriquecimento do conhecimento sobre os diferentes assuntos relacionados a IA, ML, e Chatbots:*
 - *Levantamento Bibliográfico;*
 - *Levantamento documental;*
 - *Minicursos e Vídeo aulas;*
 - *Obtenção de experiências.*

3 Construindo um Chatbot com Pytorch

Já pensou em como Alexa, Siri ou assistente de voz do Google funcionavam? Neste capítulo daremos início à construção de um chatbot para interação com usuários de uma loja virtual.

Antes de continuar, abaixo encontra-se a lista de requisitos para o nosso projeto:

- *Python 3*
 - *Define: Python* é uma linguagem de programação interpretada, orientada a objetos e de alto nível com semântica dinâmica. Suas estruturas de dados embutidas de alto nível, combinadas com tipagem dinâmica e vinculação dinâmica, o tornam muito atraente para o Desenvolvimento Rápido de Aplicativos e para uso como uma linguagem de script ou cola para conectar componentes existentes ¹.
- *Dictionaries e Lists*
 - *Define: Listas* são tipos de dados mutáveis em Python. Lists é um tipo de dados de índice baseado em 0, o que significa que o índice do primeiro elemento começa em 0. As listas são usadas para armazenar vários itens em uma única variável. As listas são um dos 4 tipos de dados do Python, ou seja, Listas, Dicionário, Tupla e Conjunto ².
 - *Define: Dicionários* Dicionários são a implementação do Python de uma estrutura de dados que é mais conhecida como uma matriz associativa. Um dicionário consiste em uma coleção de pares chave-valor. Cada par de chave-valor mapeia a chave para seu valor associado³.
- *NumPy*
 - *Define: NumPy* é o pacote fundamental para computação científica em Python. É uma biblioteca Python que fornece um objeto array multidimensional, vários objetos derivados (como arrays e matrizes mascarados) e uma variedade de rotinas para operações rápidas em arrays, incluindo matemática, lógica, manipulação de formas, classificação, seleção, E/S, Fourier discreto transforma álgebra linear básica, operações estatísticas básicas, simulação aleatória e muito mais⁴.

¹ <<https://www.python.org/doc/essays/blurb/>>

² <<https://www.analyticsvidhya.com/blog/2021/06/working-with-lists-dictionaries-in-python/>>

³ <<https://realpython.com/python-dicts>>

⁴ <<https://numpy.org/doc/stable/user/whatisnumpy.html>>

- *Pandas*
 - *Define: Pandas* é um pacote Python de código aberto que é mais amplamente usado para ciência de dados/análise de dados e tarefas de aprendizado de máquina. Ele é construído em cima de outro pacote chamado Numpy, que fornece suporte para arrays multidimensionais⁵.
- *Pytorch*
 - *Define: PyTorch* é uma biblioteca de aprendizado de máquina para Python usada principalmente para processamento de linguagem natural. O software de código aberto foi desenvolvido pelas equipes de inteligência artificial do Facebook Inc. em 2016. O PyTorch oferece dois recursos significativos, incluindo computação de tensor, bem como redes neurais profundas funcionais⁶.
- *Natural Language Processing (Bag of Words)*
 - *Define: NLP ou PLN* O processamento de linguagem natural usa aprendizado de máquina para revelar a estrutura e o significado do texto. Com aplicativos de processamento de linguagem natural, as organizações podem analisar texto e extrair informações sobre pessoas, lugares e eventos para entender melhor o sentimento da mídia social e as conversas com os clientes⁷.

3.1 Importando Bibliotecas Relevantes

```
1
2  import numpy as np
3  import random
4  import json
5  import nltk
6  import torch
7  import torch.nn as nn
8  from torch.utils.data import Dataset, DataLoader
```

Listing 3.1 – Python Bibliotecas

3.2 Criando funções personalizadas

Criaremos funções personalizadas para que seja fácil implementá-las posteriormente.

⁵ <<https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/>>
⁶ <<https://deeptai.org/machine-learning-glossary-and-terms/pytorch>>
⁷ <<https://cloud.google.com/learn/what-is-natural-language-processing>>

```
1 def tokenize(sentence):  
2     return nltk.word_tokenize(sentence)  
3  
4 def stem(word):  
5     return stemmer.stem(word.lower())
```

Listing 3.2 – Python Funções personalizadas

Nltk ou kit de ferramentas de linguagem natural é uma biblioteca realmente útil que contém classes importantes que serão úteis em qualquer uma de suas tarefas de PNL.

3.3 Stemming

Se tivermos as seguintes palavras como Andei, Ande, Andarei, Andamento, Andando, Andante, podem parecer palavras diferentes, mas geralmente têm o mesmo significado e também a mesma forma base; "and". O processo de stemização (do inglês, stemming) consiste em reduzir uma palavra ao seu radical. A palavra “meninas” se reduziria a “menin”, assim como “meninos” e “menininhos”. As palavras “gato”, “gata”, “gatos” e “gatas” reduziriam-se para “gat”⁸. Então, para que nosso modelo entenda todas as formas diferentes das mesmas palavras, precisamos treinar nosso modelo com essa forma. Isso é chamado de Stemming. Existem diferentes métodos que podemos usar para derivação. Aqui usaremos o modelo Porter Stemmer da nossa biblioteca NLTK.

```
1 from nltk.stem.porter import PorterStemmer  
2 stemmer = PorterStemmer()
```

Listing 3.3 – Python Stemming

3.4 Bag of Words

Vamos dividir cada palavra em frases e adicioná-la a um array. Nós estaremos usando um saco de palavras. Que inicialmente será uma lista de zeros com um tamanho igual ao comprimento do array all-words. Se tivermos um array de ‘frases = ["ola", "como", "voce", "esta"]’ e um array de total ‘words = ["oi", "olá", "eu", "você", "tchau", "obrigado", "legal"]’ então seu conjunto de palavras será ‘bog = [0 , 1 , 0 , 1 , 0 , 0 , 0]’. Faremos um loop sobre cada palavra no array all-words e o array bog correspondente a cada palavra. Se uma palavra da frase for encontrada no array all words, 1 será substituído naquele índice/posição no array bag.

⁸ <<https://www.computersciencemaster.com.br/como-reduzir-uma-palavra-ao-seu-radical-em-python-stemming/>>

		<u>all words</u>							
		["Hi", "How", "are", "you", "bye", "see", "later"]							
"Hi"	→	[1,	0,	0,	0,	0,	0,	0]	0 (greeting)
"How are you?"	→	[0,	1,	1,	1,	0,	0,	0]	
"Bye"	→	[0,	0,	0,	0,	1,	0,	0]	1 (goodbye)
"See you later"	→	[0,	0,	0,	1,	0,	1,	1]	
		X							y

Figura 1 – Bag of Words

Usa Bag of Words para separar uma frase em várias palavras. Imagem retirada da Fundamentação Teórica 1.

```

1 def bag_of_words(tokenized_sentence, words):
2     """
3     return bag of words array:
4     1 para cada palavra conhecida que existe na frase,
5     0 caso contrario.
6
7
8     Exemplo:
9
10
11     sentence = ["ola", "como", "esta", "voce"]
12     words = ["oi", "ola", "eu", "voce", "tchau",
13             "obrigado", "legal"]
14     bag = [ 0, 1, 0, 1, 0, 0, 0]
15     """
16     # stem cada palavra
17     sentence_words = [stem(word) for word in tokenized_sentence]
18     # initialize bag com 0 para cada palavra
19     bag = np.zeros(len(words), dtype=np.float32)
20     for idx, w in enumerate(words):
21         if w in sentence_words:
22             bag[idx] = 1
23
24     return bag

```

Listing 3.4 – Python Bag of Words

Durante o processo, também usaremos `nltkwordtokenize()` que converterá uma única string de sentença em uma lista de palavras. Por exemplo, se você passar **"Ola, como voce**

esta?", ele retornará **"ola", "como", "voce", "esta"**. **Observação:** Passaremos palavras em minúsculas para o **Stemmer** para que palavras como Bom e bom (em maiúsculas) não sejam rotuladas como palavras diferentes.

3.5 Carregando os dados e Data Cleaning

Usaremos um conjunto de dados chamado **intents.json** que tem a estrutura mostrada no campo abaixo. Estaremos limpando esses dados de acordo com as nossas necessidades usando as funções que criamos anteriormente.

```
1 {
2   "intents": [
3     {
4       "tag": "saudacao",
5       "patterns": [
6         "Oi",
7         "Ei",
8         "Como voce esta",
9         "Tem alguem ai?",
10        "Ola",
11        "Bom dia"
12      ],
13      "responses": [
14        "Ei :-)",
15        "Ola, obrigado pela visita",
16        "Ola, o que posso fazer por voce?",
17        "Ola, como posso ajudar?"
18      ]
19    }
20  ]
21 }
```

Listing 3.5 – Data intents.json

Agora vamos simplesmente carregar o arquivo json usando a função **jsonload**.

```
1 with open('intents.json', 'r') as f:
2     intents = json.load(f)
```

Listing 3.6 – Python Carregando dados

Para obter as informações corretas, iremos descompactá-las com o seguinte código:

```
1 all_words = []
2 tags = []
3 xy = []
4 # loop atraves de cada frase no nosso intents patterns
5 for intent in intents['intents']:
```

```

6     tag = intent['tag']
7     # adicionar a tag list
8     tags.append(tag)
9     for pattern in intent['patterns']:
10        # tokenize cada palavra na frase
11        w = tokenize(pattern)
12        # adicionar a nossa lista de palavras
13        all_words.extend(w)
14        # adicionar ao par xy
15        xy.append((w, tag))

```

Listing 3.7 – Python Loop nos dados

Isso separará todas as tags e palavras em suas listas.

3.6 Limpeza e preparação dos dados

Estaremos usando nossas funções personalizadas e limpando os dados implementando as funções que criamos em nossas células anteriores.

```

1
2 # stem e lower cada palavra
3 ignore_words = ['?', '.', '!']
4 all_words = [stem(w) for w in all_words if w not in ignore_words]
5 # remover duplicados e sort
6 all_words = sorted(set(all_words))
7 tags = sorted(set(tags))
8
9 print(len(xy), "patterns") # padroes
10 print(len(tags), "tags:", tags) # tags
11 print(len(all_words), "unique stemmed words:", all_words) # palavras
    derivadas unicas

```

Listing 3.8 – Data Cleaning

Criando dados de treinamento: transformaremos os dados em um formato que nosso modelo PyTorch possa entender facilmente

```

1
2 # criar dados de treinamento
3 X_train = []
4 y_train = []
5 for (pattern_sentence, tag) in xy:
6     # X: bag de palavras para cada pattern_sentence
7     bag = bag_of_words(pattern_sentence, all_words)
8     X_train.append(bag)
9     # y: PyTorch CrossEntropyLoss precisa apenas class labels, nao one-hot

```

```

10     label = tags.index(tag)
11     y_train.append(label)
12
13 X_train = np.array(X_train)
14 y_train = np.array(y_train)

```

Listing 3.9 – Training Data

3.7 Modelo PyTorch

Aqui estaremos fazendo uma classe para implementar nossa rede neural personalizada. Será uma Rede neural feed-forward que terá 3 Camadas Lineares e usaremos a função de ativação “ReLU”. Nota: Usamos a função `super()` para herdar as propriedades de sua classe pai. Este é um conceito de Programação Orientada a Objetos (OOP).

- *Define: ReLU* é uma função de ativação não linear que é usada em redes neurais multicamadas ou redes neurais profundas. Esta função pode ser representada como:

$$f(x) = \max(0, x) \quad (1)$$

Figura 2 – ReLu

onde x = um valor de entrada.

De acordo com a equação 1, a saída de ReLu é o valor máximo entre zero e o valor de entrada. A saída é igual a zero quando o valor de entrada é negativo e o valor de entrada quando a entrada é positiva. Assim, podemos reescrever a equação 1 da seguinte forma:

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \quad (2)$$

Figura 3 – ReLu

onde x = um valor de entrada.

Exemplos de ReLu: Dadas diferentes entradas, a função gera diferentes saídas. Por exemplo, quando x é igual a -5, a saída de $f(-5)$ é 0. Por outro lado, a saída de $f(0)$ é 0 porque a entrada é maior ou igual a 0. Além disso, o resultado de $f(5)$ é 5 porque a entrada é maior que zero ⁹.

⁹ <<https://deeptai.org/machine-learning-glossary-and-terms/relu/>>

3.7.1 Feed Forward Neural Network

Define: Feed Forward Neural Network Uma Rede Neural Feed Forward é uma rede neural artificial na qual as conexões entre os nós não formam um ciclo. O oposto de uma rede neural feed-forward é uma rede neural recorrente, na qual certos caminhos são ciclados. O modelo feed-forward é a forma mais simples de uma rede neural, pois a informação é processada apenas em uma direção. Embora os dados possam passar por vários nós ocultos, eles sempre se movem em uma direção e nunca para trás ¹⁰.

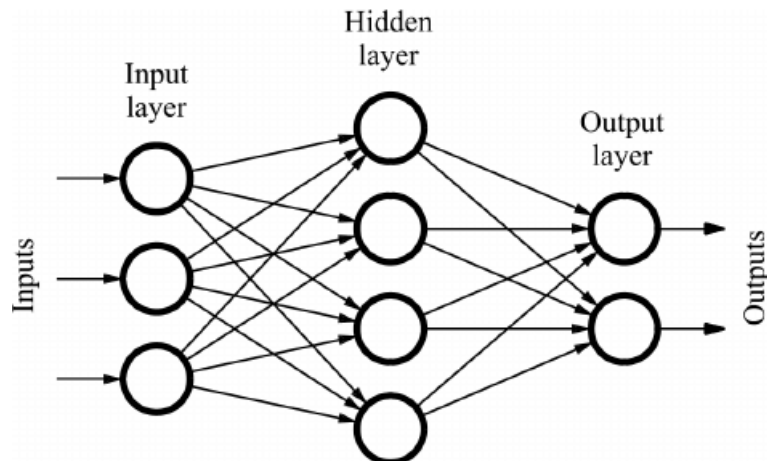


Figura 4 – Feed Forward Neural Network

Exemplo de uma rede neural feed-forward.

Desse modo, uma representação válida para o nosso projeto seria a FFNN abaixo:

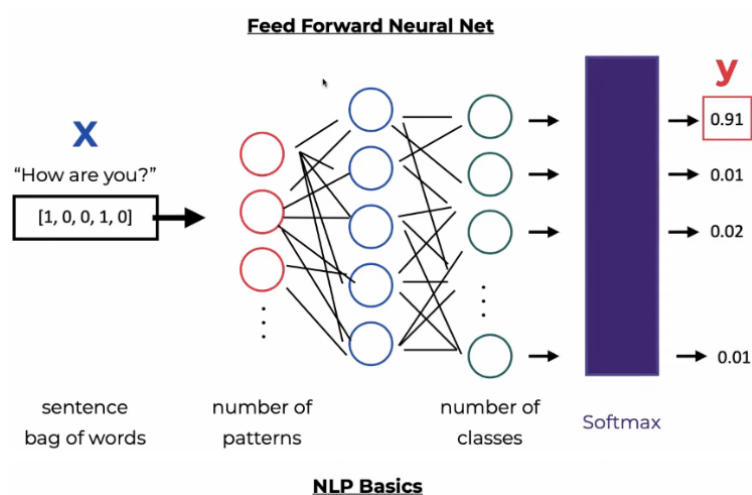


Figura 5 – Feed Forward Neural Network

Exemplo de uma rede neural feed-forward para o nosso projeto. Imagem retirada da Fundamentação Teórica 1.

¹⁰ <<https://deeppai.org/machine-learning-glossary-and-terms/feed-forward-neural-network>>

3.7.2 Função de ativação

Define: Função de ativação Uma função de ativação é uma função usada em redes neurais artificiais que produz um valor pequeno para entradas pequenas e um valor maior se suas entradas excederem um limite. Se as entradas forem grandes o suficiente, a função de ativação "dispara", caso contrário não faz nada. Em outras palavras, uma função de ativação é como um portão que verifica se um valor de entrada é maior que um número crítico.

As funções de ativação são úteis porque adicionam não linearidades às redes neurais, permitindo que as redes neurais aprendam operações poderosas. Se as funções de ativação fossem removidas de uma rede neural feedforward, toda a rede poderia ser refatorada para uma simples operação linear ou transformação de matriz em sua entrada e não seria mais capaz de realizar tarefas complexas, como reconhecimento de imagem ¹¹.

3.7.2.1 Função ReLU

Atualmente, existem várias funções de ativação amplamente usadas no aprendizado profundo. Uma das mais simples é a ReLU (Rectified Linear Unit) Activation Function, ou função ReLU, que é uma função linear por partes que produz zero se sua entrada for negativa, e diretamente a saída caso contrário. Como mencionado, O ReLU é a função de ativação mais usada no mundo atualmente. Já que é usado em quase todas as redes neurais convolucionais ou deep learning.

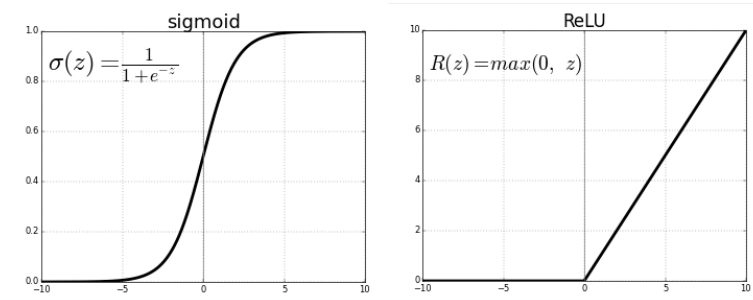


Figura 6 – ReLU v/s Logistic Sigmoid

Como você pode identificar, o ReLU está meio retificado (de baixo). $f(z)$ é zero quando z é menor que zero e $f(z)$ é igual a z quando z é maior ou igual a zero.

Range: [0 ao infinito)

A função e sua derivada são monotônicas.

Mas o problema é que todos os valores negativos se tornam zero imediatamente, o que diminui a capacidade do modelo de ajustar ou treinar a partir dos dados corretamente.

¹¹ <<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>>

Isso significa que qualquer entrada negativa dada à função de ativação ReLU transforma o valor em zero imediatamente no gráfico, o que, por sua vez, afeta o gráfico resultante por não mapear os valores negativos adequadamente¹².

3.7.2.2 Derivada da Função ReLU

Em redes neurais, uma função de ativação agora comumente usada é a unidade linear retificada, ou como comumente abreviada, ReLU. O ReLU é definido como,

$$f(x) = \max(0, x)$$

Figura 7 – A representação matemática da função ReLU

O que essa função faz? Basicamente, ele define qualquer coisa menor ou igual a 0 (números negativos) como 0. E mantém todos os mesmos valores para quaisquer valores > 0¹³.

Também é instrutivo calcular o gradiente da função ReLU, que é matematicamente indefinida em $x = 0$, mas que ainda é extremamente útil em redes neurais.

A derivada da função ReLU. Na prática, a derivada em $x = 0$ pode ser definida como 0 ou 1. A derivada zero para x negativo pode dar origem a problemas ao treinar uma rede neural, pois um neurônio pode ficar 'preso' na região zero e a retropropagação (backpropagation) nunca mudará seus pesos. Em outras palavras, digamos que tenhamos entrada menor que 0, então a saída é zero e a rede neural não pode continuar o algoritmo de retropropagação. Esse problema é comumente conhecido como Dying ReLU. Para nos livrarmos desse problema, usamos uma versão improvisada do ReLU, chamada Leaky ReLU (Cujo nao cobrimos nesse projeto)¹⁴.

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & otherwise \end{cases}$$

Figura 8 – A derivada da função ReLU

¹² <<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>>

¹³ <<https://kawahara.ca/what-is-the-derivative-of-relu/>>

¹⁴ <<https://vidyasheela.com/post/what-is-the-derivative-of-the-relu-activation-function-including-python-function>>

3.7.3 Criando nosso modelo

```
1
2 class NeuralNet(nn.Module):
3     def __init__(self, input_size, hidden_size, num_classes):
4         super(NeuralNet, self).__init__()
5         self.l1 = nn.Linear(input_size, hidden_size)
6         self.l2 = nn.Linear(hidden_size, hidden_size)
7         self.l3 = nn.Linear(hidden_size, num_classes)
8         self.relu = nn.ReLU()
9
10    def forward(self, x):
11        out = self.l1(x)
12        out = self.relu(out)
13        out = self.l2(out)
14        out = self.relu(out)
15        out = self.l3(out)
16        # sem ativacao e sem softmax no final
17        return out
```

Listing 3.10 – Modelo

Aqui nós herdamos uma classe do `NN.Module` porque estaremos customizando o modelo e suas camadas.

3.8 Atribuindo o conjunto de dados ao modelo

Usaremos algumas funções de Magic, escreva nossa classe.

Define: funções de Magic ou Métodos mágicos são métodos especiais em python que possuem sublinhados duplos (dunder) em ambos os lados do nome do método. Métodos mágicos são predominantemente usados para sobrecarga do operador. Sobrecarga de operador significa fornecer funcionalidade adicional aos operadores, o python invoca implicitamente os métodos mágicos para fornecer funcionalidade adicional a ele. Por exemplo, a multiplicação de dois inteiros pode ser feita usando o operador de multiplicação ($2*3 = 6$) e o mesmo operador pode ser usado para repetir a string (“maçã-” * 3 = ‘maçã-maçã-maçã’). Alguns exemplos de métodos mágicos são `init`, `len`, `repr`, `add` e etc ^{15,16}.

- O método `init` para inicialização é invocado sem qualquer chamada, quando uma instância de uma classe é criada, como construtores em algumas outras linguagens de programação, como C++, Java, PHP etc. Esses métodos são a razão pela qual podemos adicionar duas strings com `' + '` operador sem qualquer typecast explícito.

¹⁵ <<https://www.analyticsvidhya.com/blog/2021/08/explore-the-magic-methods-in-python>>

¹⁶ <<https://www.geeksforgeeks.org/dunder-magic-methods-python/>>

- O método `getitem` é usado para obter um item do atributo das instâncias invocadas. `getitem` é comumente usado com contêineres como lista, tupla, etc.
- O método mágico `len` é usado para encontrar o comprimento dos atributos da instância. Quando usamos `len(instance)`, ele retorna o comprimento do atributo de instância que geralmente é um container.

```
1 class ChatDataset(Dataset):
2
3     def __init__(self):
4         self.n_samples = len(X_train)
5         self.x_data = X_train
6         self.y_data = y_train
7
8         # suporte a indexacao de modo que o conjunto de dados[i] possa ser
9         # usado para obter a i-esima amostra
10
11     def __getitem__(self, index):
12         return self.x_data[index], self.y_data[index]
13
14     # podemos chamar len(dataset) para retornar o tamanho
15     def __len__(self):
16         return self.n_samples
```

Listing 3.11 – Dataset

3.9 Hiperparâmetros

Toda rede neural tem um conjunto de hiperparâmetros que precisam ser definidos antes do uso.

Antes de instanciar nossa classe ou modelo de rede neural que escrevemos anteriormente, primeiro definiremos alguns hiperparâmetros que podem ser alterados de acordo.

Os hiperparâmetros são parâmetros cujos valores controlam o processo de aprendizado e determinam os valores dos parâmetros do modelo que um algoritmo de aprendizado acaba aprendendo. O prefixo *hyper* sugere que são parâmetros de nível superior que controlam o processo de aprendizagem e os parâmetros do modelo que dele resultam.

Um engenheiro de aprendizado de máquina projetando um modelo, faz a escolha de definir valores de hiperparâmetros que seu algoritmo de aprendizado usará antes mesmo do início do treinamento do modelo. Sob essa luz, os hiperparâmetros são considerados externos ao modelo porque o modelo não pode alterar seus valores durante o

aprendizado/treinamento^{17,18,19}.

Hiperparâmetros:

- *Define: Epoch* O número de épocas é um hiperparâmetro que define o número de vezes que o algoritmo de aprendizado funcionará em todo o conjunto de dados de treinamento.

Uma época significa que cada amostra no conjunto de dados de treinamento teve a oportunidade de atualizar os parâmetros internos do modelo.

Você pode pensar em um loop for sobre o número de épocas em que cada loop prossegue no conjunto de dados de treinamento. Dentro desse loop for há outro loop for aninhado que itera sobre cada lote de amostras, onde um lote tem o número de amostras "tamanho de lote" especificado.

O número de épocas é tradicionalmente grande, geralmente centenas ou milhares, permitindo que o algoritmo de aprendizado seja executado até que o erro do modelo seja suficientemente minimizado. Você pode ver exemplos do número de épocas na literatura e em tutoriais definidos para 10, 100, 500, 1000 e maiores.

- *Define: Batch* O tamanho do Batch é um hiperparâmetro que define o número de amostras para trabalhar antes de atualizar os parâmetros do modelo interno.

Pense em um Batch como um loop for iterando sobre uma ou mais amostras e fazendo previsões. No final do Batch, as previsões são comparadas com as variáveis de saída esperadas e um erro é calculado. A partir desse erro, o algoritmo de atualização é usado para melhorar o modelo, por exemplo. mover para baixo ao longo do gradiente de erro.

- *Define: Learning Rate* O hiperparâmetro de taxa de aprendizado controla a taxa ou velocidade na qual o modelo aprende. Especificamente, ele controla a quantidade de erro distribuído com o qual os pesos do modelo são atualizados cada vez que são atualizados, como no final de cada lote de exemplos de treinamento.
- *Define: Hidden Units* As unidades ocultas fazem parte das redes neurais, que se referem aos componentes que compõem as camadas de processadores entre as unidades de entrada e saída em uma rede neural. É importante especificar o número de hiperparâmetros de unidades ocultas para a rede neural. Deve estar entre o tamanho da camada de entrada e o tamanho da camada de saída. Mais especificamente, o número de unidades ocultas deve ser 2/3 do tamanho da camada de entrada, mais o tamanho da camada de saída.

¹⁷ <<https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac>>

¹⁸ <<https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch>>

¹⁹ <<https://www.javatpoint.com/hyperparameters-in-machine-learning>>

Veja mais sobre Hiperparâmetros aqui ²⁰.

```
1 # Hiperparametros
2 num_epochs = 1000
3 batch_size = 8
4 learning_rate = 0.001
5 input_size = len(X_train[0])
6 hidden_size = 8
7 output_size = len(tags)
8 print(input_size, output_size)
```

Listing 3.12 – Hiperparâmetros

3.10 Perda e otimizador

Vamos agora instanciar as funções de modelo, perda e otimizador.

Função de Perda: Cross Entropy Otimizador: Adam Optimizer

```
1 dataset = ChatDataset()
2 train_loader = DataLoader(dataset=dataset,
3                             batch_size=batch_size,
4                             shuffle=True,
5                             num_workers=0)
6
7 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
8
9 model = NeuralNet(input_size, hidden_size, output_size).to(device)
10
11 # Perda e otimizador
12 criterion = nn.CrossEntropyLoss()
13 optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Listing 3.13 – Perda e otimizador

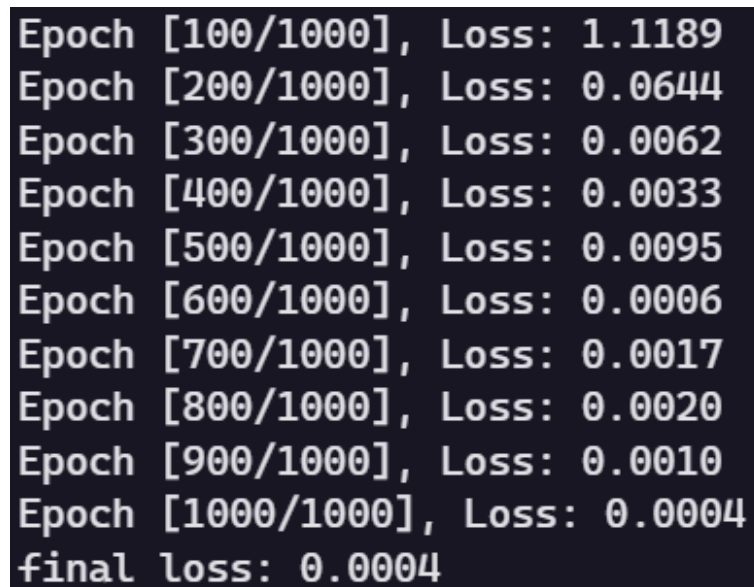
3.11 Treinando o Modelo

```
1
2 # Treine o modelo
3 for epoch in range(num_epochs):
4     for (words, labels) in train_loader:
5         words = words.to(device)
6         labels = labels.to(dtype=torch.long).to(device)
7
8         # Forward pass
9         outputs = model(words)
```

²⁰ <<https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch>>

```
10     # if y would be one-hot, we must apply
11     # labels = torch.max(labels, 1)[1]
12     loss = criterion(outputs, labels)
13
14     # Backward e optimize
15     optimizer.zero_grad()
16     loss.backward()
17     optimizer.step()
18
19     if (epoch+1) % 100 == 0:
20         print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}'
21             ',')
22
23 print(f'final loss: {loss.item():.4f}')
24
25 data = {
26 "model_state": model.state_dict(),
27 "input_size": input_size,
28 "hidden_size": hidden_size,
29 "output_size": output_size,
30 "all_words": all_words,
31 "tags": tags
32 }
```

Listing 3.14 – Treinando o Modelo



```
Epoch [100/1000], Loss: 1.1189
Epoch [200/1000], Loss: 0.0644
Epoch [300/1000], Loss: 0.0062
Epoch [400/1000], Loss: 0.0033
Epoch [500/1000], Loss: 0.0095
Epoch [600/1000], Loss: 0.0006
Epoch [700/1000], Loss: 0.0017
Epoch [800/1000], Loss: 0.0020
Epoch [900/1000], Loss: 0.0010
Epoch [1000/1000], Loss: 0.0004
final loss: 0.0004
```

Figura 9 – output

3.11.1 Salvando o modelo treinado

```
1
2 FILE = "data.pth"
3 torch.save(data, FILE)
4
5 print(f'training complete. file saved to {FILE}')
```

Listing 3.15 – Salvando o modelo treinado

section[Carregando nosso modelo salvo]Carregando modelo salvo

```
1
2 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
3
4 with open('intents.json', 'r') as json_data:
5     intents = json.load(json_data)
6
7 FILE = "data.pth"
8 data = torch.load(FILE)
9
10 input_size = data["input_size"]
11 hidden_size = data["hidden_size"]
12 output_size = data["output_size"]
13 all_words = data['all_words']
14 tags = data['tags']
15 model_state = data["model_state"]
16
17 model = NeuralNet(input_size, hidden_size, output_size).to(device)
18 model.load_state_dict(model_state)
19 model.eval()
```

Listing 3.16 – Carregando modelo salvo

3.12 Usando o Chatbot

Nosso modelo está pronto. Agora vamos finalmente conversar com nosso chatbot. Como nossos dados de treinamento eram muito limitados, só podemos conversar sobre alguns tópicos. O modelo pode ser treinado em um conjunto de dados maior para aumentar a generalização/conhecimento do Chatbot.

```
1
2 bot_name = "chatbotIA"
3 print("Let's chat! (digite 'quit' para sair)")
4 while True:
5     # frase = "Ola!"
6     sentence = input("Voce: ")
7     if sentence == "quit":
8         break
9
```

```
10     sentence = tokenize(sentence)
11     X = bag_of_words(sentence, all_words)
12     X = X.reshape(1, X.shape[0])
13     X = torch.from_numpy(X).to(device)
14
15     output = model(X)
16     _, predicted = torch.max(output, dim=1)
17
18     tag = tags[predicted.item()]
19
20     probs = torch.softmax(output, dim=1)
21     prob = probs[0][predicted.item()]
22     if prob.item() > 0.75:
23         for intent in intents['intents']:
24             if tag == intent["tag"]:
25                 print(f"{bot_name}: {random.choice(intent['responses'])}")
26     else:
27         print(f"{bot_name}: Eu nao entendi...")
```

Listing 3.17 – Chatbot

Nosso Modelo foi treinado com pouquíssimos exemplos, então não entende tudo o que passamos para ele.

Referências

KRAJCIK, J. S.; BLUMENFELD, P. C. *Project-based learning*. [S.l.]: na, 2006. Citado na página [25](#).