Pontifícia Universidade Católica do Rio de Janeiro
Coordenação Central de Extensão

Rodrigo Mafort Oliveira da Silva

Classificador de notícias por título baseado em rede neural recorrente LSTM.

Trabalho de Conclusão de Curso de Pós-Graduação

Departamento de Engenharia Elétrica

Rio de Janeiro
Outubro de 2019

Pontifícia Universidade Católica do Rio de Janeiro

Coordenação Central de Extensão
Departamento de Engenharia Elétrica

Rodrigo Mafort Oliveira da Silva

Classificador de notícias por título baseado em rede neural recorrente LSTM.

Trabalho de Conclusão de Curso de Pós-Graduação

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da PUC/RIO, como parte dos requisitos para obtenção do título de Especialização em Business Intelligence.

Professor orientador: Leonardo Mendonza

Rio de Janeiro
Outubro de 2019

Agradecimentos

Gostaria de agradecer a todos professores do curso de BI Master da PUC pelos conhecimentos e experiências divididos, ao meu orientador, a minha família e ao meu companheiro pelo apoio que deram em um momento difícil da minha vida que quase impossibilitou a conclusão desta monografia.

Resumo

O objetivo deste trabalho é fazer a classificação e separação de categorias de notícias usando redes neurais em uma era onde há uma tempestade de informações.

Para isto foi usada uma rede neural recorrente LSTM implementada com a linguagem Python e as bibliotecas Keras e NLTK, ambas de código aberto.

A biblioteca Keras tem por objetivo criar uma interface de alto nível para implementação de redes neurais em Python, propiciando uma rápida prototipação e abstração das complexidades de bibliotecas como o TensorFlow.

Já a biblioteca NTLK é a líder no desenvolvimento de linguagem natural em Python.

Sumário

1. Introdução

As redes neurais são uma ferramenta poderosa para o reconhecimento de padrões. Tais como padrões de fala, escrita e imagens. Usando os neurônios artificiais para simular a capacidade cerebral humana.

Somando essa ferramenta ao poder das placas gráficas modernas, que permitem fazer operações matemáticas de forma paralela, as redes neurais foram potencializadas, possibilitando o Deep Learning, que será usado neste trabalho.

1.1 Motivação

O trabalho visa mostrar um caso prático de uso de Deep Learning em Python com a implementação de uma rede neural LSTM, associado ao processamento de linguagem natural.

1.2 Objetivos

A partir de uma base de dados de títulos de notícia na internet criar uma rede neural capaz de classificar a notícia em quatro grandes categorias: negócios, ciência e tecnologia, entretenimento e saúde

1.3 Descrição

O desenvolvimento da monografia se faz em algumas partes.

- Definição de processamento de linguagem natural
- Definição de redes neurais
- Desenvolvimento do código para processamento em linguagem natural com o NLTK
- Desenvolvimento da rede neural recorrente LSTM com o Keras

A base de dados utilizada foi a News Aggregator Data Set, obtida no repositório UCI Machine Learning Repository (DUA D.; GRAFF, C., 2019).

1.4 Organização da monografia

Esta monografia está dividida em 6 capítulos, descritos a seguir:

- O capítulo 2 traz o conceito de processamento de linguagem natural
- O capítulo 3 tem como foco redes neurais artificiais
- O capítulo 4 mostra a implementação do processamento de linguagem natural com o NTLK e a LSTM através do Keras
- O capítulo 5 apresenta as conclusões obtidas.
- Finalmente o capítulo 6 identifica possíveis trabalhos futuros.

2. Processamento de linguagem natural

O processamento de linguagem natural (PLN) consiste no desenvolvimento de modelos computacionais para a realização de tarefas que dependem de informações expressadas em alguma linguagem natural. (PEREIRA, 2005?)

Ela se vale de três aspectos da comunicação em linguagem natural:

- som: fonologia

- estrutura: morfologia e sintaxe

- significado: semântica e pragmática

A utilização dessa ferramenta é muito útil em mineração de textos, atividade que tem o objetivo de extrair conhecimento de dados não estruturados, busca de informações em documentos, classificação de documentos por categorias e afins.

Mas para que esta seja efetiva, os dados precisam passar por um pré-processamento antes que possam ser utilizados para a extração de conhecimento. A seguir serão apresentadas algumas técnicas para tal que foram utilizadas neste trabalho.

O primeiro passo foi a tokenização, esta tem o objetivo de dividir o texto em trechos menores, podendo ser frases ou palavras. Neste tratamento são descartados os espaços e em caso de palavras combinadas, estas são combinadas em um único token. Abaixo temos um exemplo de como uma frase é separada em tokens:

- There is some long-term solution for this problem.
- ['There', 'is', 'some', 'long-term', 'solution', 'for', 'this', 'problem', '.']

Depois da tokenização foi usada a remoção de stopwords. Este passo tem o objetivo de remover palavras muito frequentes e que não adicionam informações para a análise, por não ter valor semântico. São fortes candidatos para a tokenização as palavras das classes gramaticais a seguir: Artigos, preposições, pontuação, conjunções e pronomes. Esse passo também diminui o vocabulário a ser indexado para análise consideravelmente. A seguir vemos como os tokens gerados da frase anterior ficam após a remoção de stopwords.

- ['There', 'is', 'some', 'long-term', 'solution', 'for', 'this', 'problem', '.']
- ['There', 'long-term', 'solution', 'problem', '.']

O último passo foi a stemização. Que serve para reduzir palavras que passaram por alguma flexão para a sua forma mais básica, seu radical. Novamente tendo uma boa redução de vocabulário. Existem stemmers para várias línguas, como os títulos das notícias usadas são em inglês, foram testados os algoritmos Snowball, (também conhecido como Porter2 e o algoritmo Lancaster. Abaixo serão mostrados alguns exemplos de como cada um reduz as palavras ao seu radical:

- cement / Snowball: cement / Lancaster: cem
- cemetery / Snowball: cemetery / Lancaster: cemeteri
- supernatural / Snowball: supernatur / Lancaster: supern
- turned / Snowball: turn / Lancaster: turn

O algoritmo Snowball foi escolhido para a implementação, por não ser tão agressivo na definição dos radicais.

3. Redes neurais artificiais

Redes neurais artificiais reproduzem o processo de resolução de problemas do cérebro humano. Assim como na versão orgânica, a sua unidade básica são os neurônios, que propagam informações através de conexões entre estes. Cada neurônio possui pesos e uma função de ativação, que trata os sinais de entrada para gerar sua saída. Tal representação é mostrada na figura 1.

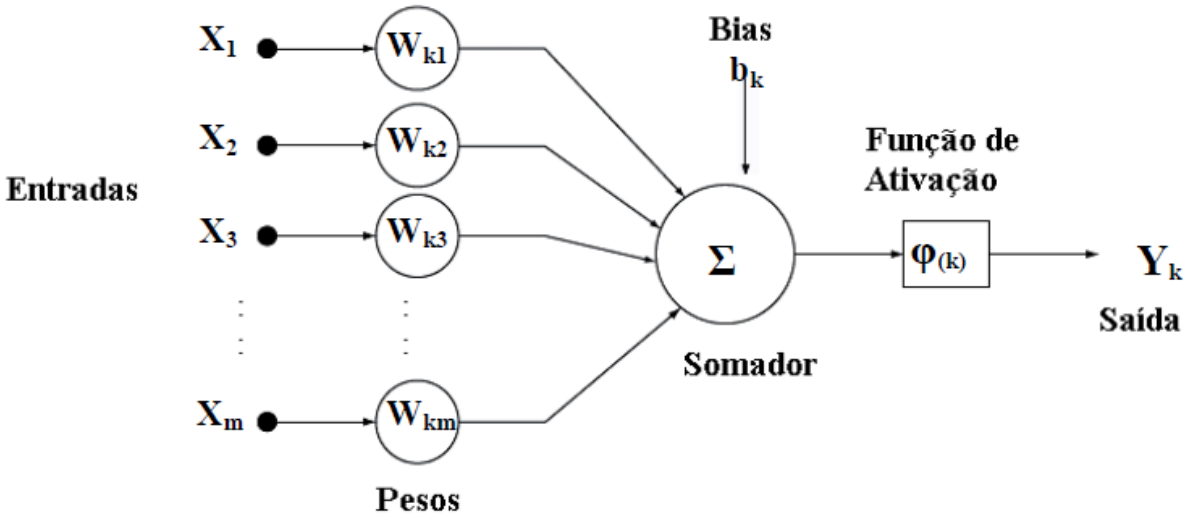

Figura 1 - Neurônio artificial de McCulloch-Pitts

A maioria das redes neurais possuem alguma regra de treinamento, onde os pesos de seus neurônios são ajustados de acordo com os exemplos apresentados.

Os neurônios normalmente são organizados em camadas e estas são divididas em três grupos:

- Camadas de entrada: onde os padrões são apresentados para a rede;
- Camadas intermediárias ou escondidas: onde a maior parte do processamento é feito e servem como extratoras de características;
- Camada de saída: onde o resultado do processamento é apresentado.

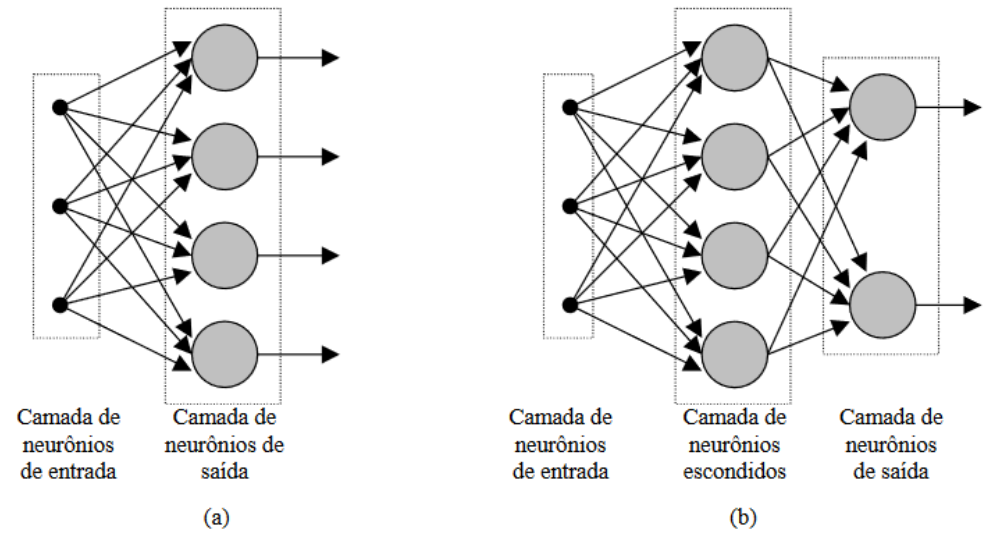

Figura 2 - Redes neurais em camada. (a) rede sem camada intermediária, (b) rede com camada intermediária
FONTE: IVODA (2000)

3.1 Aprendizado supervisionado

Este tipo de aprendizado é caracterizado pela presença de um "professor" externo. A função deste é prover a rede neural com uma resposta desejada a um determinado estímulo apresentado pelo ambiente. Definimos um sinal de erro como a diferença entre a resposta desejada e a saída da rede neural. Os parâmetros são então ajustados de acordo com o sinal de erro. A figura 3 mostra um diagrama de blocos de um sistema com aprendizado supervisionado. (YIODA, 2000)

Esse aprendizado se divide em dois tipos de algoritmos, os de regressão e os de classificação.

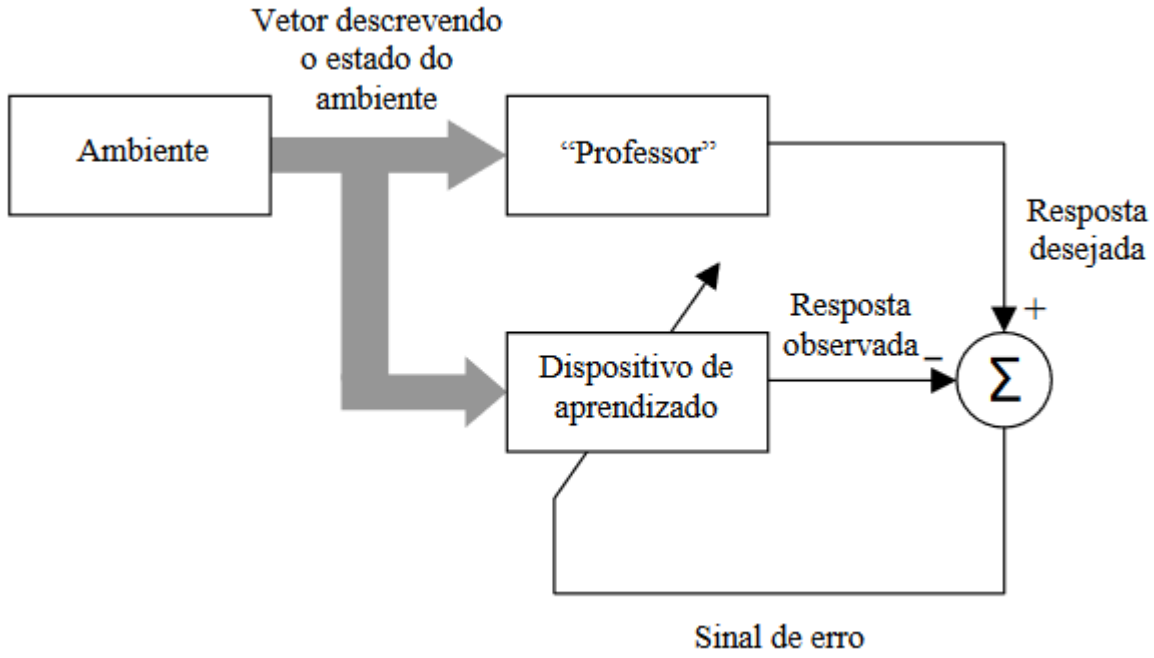

Figura 3 - Diagrama de blocos de um sistema com aprendizado supervisionado
FONTE: IVODA (2000)

3.2 Aprendizado não supervisionado

Este tipo de aprendizado, diferente do anterior, não há uma resposta desejada a ser informada. Também não há um "professor" para ajudar o aprendizado, equivalendo a um trabalho de um aluno autodidata que precisa descobrir sozinho como chegar

ao resultado. Neste caso, o algoritmo deve explorar os dados e tentar achar alguma estrutura neles.

Dependendo da técnica utilizada, o algoritmo vai procurar agrupamentos entre esses dados, aproximando os que tiverem alguma semelhança entre si.

3.2 Redes LSTM (Long Short Term Memory)

A rede LSTM é uma rede neural do tipo recorrente. Esse tipo de rede gera laços (loops) que permitem a retenção de informações.

Ao fazer um paralelo com o raciocínio humano, ao ler um texto, cada palavra é entendida de acordo com a compreensão das palavras anteriores deste. Essa persistência não pode ser reproduzida facilmente em redes neurais tradicionais.

Uma rede neural recorrente pode ser imaginada como múltiplas cópias de uma mesma rede, passando a mensagem para o seu sucessor em cadeia, como mostrado na figura 4. (DEEP LEARNING BOOK, 2019)

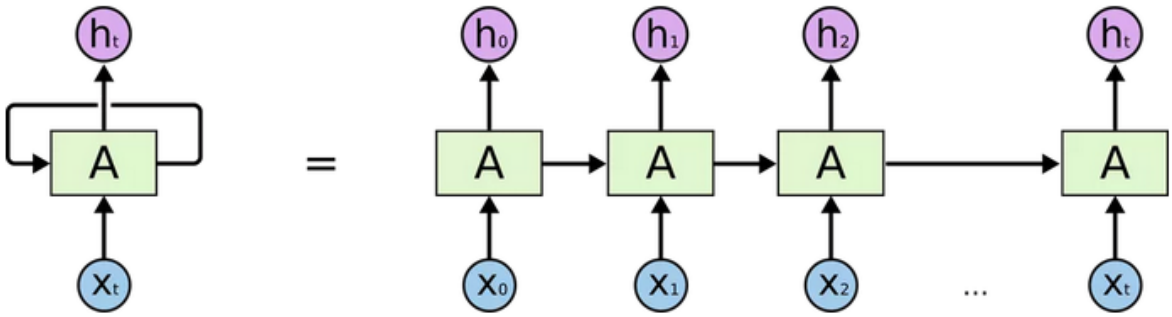

Figura 4 - Rede neural recorrente
Fonte: DEEP LEARNING BOOK (2019)

3.2.1 Arquitetura da LSTM

A LSTM é uma arquitetura de rede recorrente (RNN) que “lembra” valores em intervalos arbitrários. A LSTM é adequada para classificar, processar e prever séries temporais com intervalos de tempo de duração desconhecida. A insensibilidade ao comprimento do gap dá vantagem à LSTM em relação as redes neurais recorrentes tradicionais. (DEEP LEARNING BOOK, 2019)

A LSTM possui uma estrutura em cadeia que possui quatro diferentes redes neurais e blocos de memória chamados células. Sua estrutura é mostrada na figura 5.

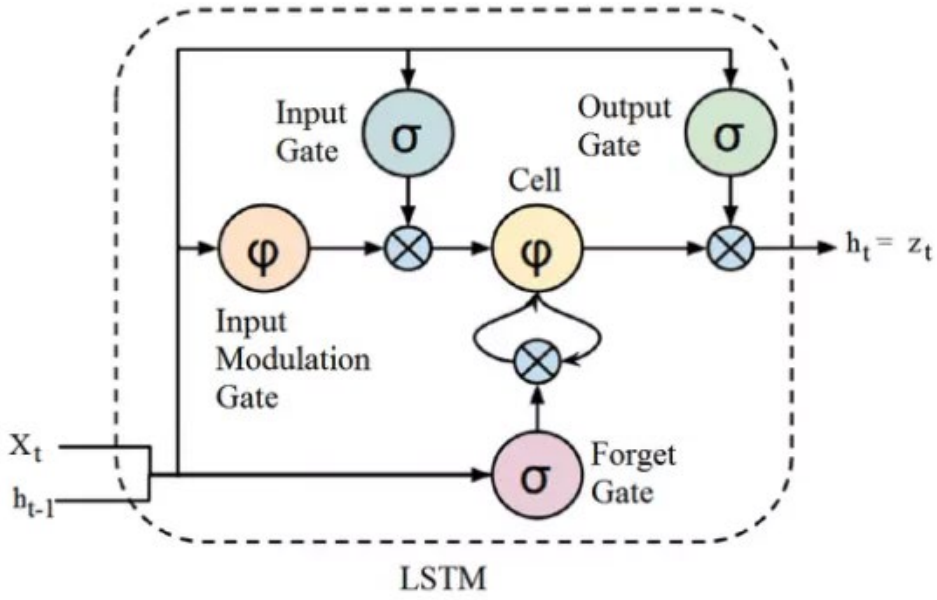

Figura 5 - Representação de uma LSTM
Fonte: DEEP LEARNING BOOK (2019)

A informação é retida pelas células e as manipulações são feitas pelos portões (gates). Existem três portões:

- Forget Gate: As informações que inúteis no estado da célula são descartadas com o forget gate. Duas entradas: x_t (entrada no momento específico) e h_t-1 (saída de célula anterior) são alimentadas ao gate e multiplicadas por matrizes de peso, seguidas pela adição do bias. O resultante é passado por uma função de ativação que fornece uma saída binária. Se para um determinado estado de célula a saída for 0, a informação é esquecida e para a saída 1, a informação é retida para uso futuro.
- Input Gate: A adição de informações úteis ao estado da célula é feita pelo input gate. Primeiro, a informação é regulada usando a função sigmoide que filtra os valores a serem lembrados de forma similar ao forget gate usando as entradas h_t-1 e x_t. Então, um vetor é criado usando a função tanh que dá saída de -1 a +1, que contém todos os valores possíveis de h_t-1 e x_t.

Os valores do vetor e os valores regulados são multiplicados para obter as informações úteis

- Output Gate: A tarefa de extrair informações úteis do estado da célula atual para ser apresentadas como uma saída é feita pelo output gate. Primeiro, um vetor é gerado aplicando a função tanh na célula. Então, a informação é regulada usando a função sigmoide que filtra os valores a serem lembrados usando as entradas h_t-1 e x_t. Os valores do vetor e os valores regulados são multiplicados para serem enviados como uma saída e entrada para a próxima célula.

A célula RNN recebe duas entradas, a saída do último estado oculto e a observação no tempo = t. Além do estado oculto, não há informações sobre o passado para se lembrar. A memória de longo prazo é geralmente chamada de estado da célula. As setas em loop indicam a natureza recursiva da célula.

Isso permite que as informações dos intervalos anteriores sejam armazenadas na célula LSTM. O estado da célula é modificado pelo forget gate colocado abaixo do estado da célula e ajustado pela porta de modulação de entrada. Da equação, o estado da célula anterior esquece, multiplica-se com a porta do esquecimento e adiciona novas informações através da saída das portas de entrada. (DEEP LEARNING BOOK, 2019)

4. Implementação

O código fonte foi escrito usando a linguagem de programação Python e as bibliotecas NLTK e Keras.

A partir da base de dados News Aggregator Data Set foram extraídas oito mil linhas com títulos, sendo divididas igualmente entre as quatro categorias de notícias, a saber: negócios, ciência e tecnologia, entretenimento e saúde.

Esses dados passaram pelas etapas de pré-processamento, passando pelos processos de tokenização, remoção de stopwords e stemização com ajuda da biblioteca NLTK.

4.1 NLTK (Natural Language Toolkit)

A biblioteca NLTK foi criada em 2001 como parte de uma disciplina em linguística computacional no Department of Computer and Information Science da University of Pennsylvania. Desde então é mantida e melhorada com ajuda de uma dezena de pessoas. Ela serve de base para vários projetos de pesquisa.

4.2 Pré-processamento

Após a extração das oito mil linhas foi iniciada a exploração de dados. A partir desta foi descoberto o máximo de palavras em um título e quantas palavras únicas temos no total depois do tratamento feito com a biblioteca NLTK. Eles são mostrados na figura 6 pelos dados maxlen e len(word_freqs) respectivamente.

```
file = "uci-news-aggregator.csv"

data = pd.read_csv(file, usecols=["CATEGORY", "TITLE"])
#Converter categoria string para numérico
data.CATEGORY = pd.Categorical(data.CATEGORY)
data['CATEGORY'] = data.CATEGORY.cat.codes

ROWS = 8000
ROWS_PER_CATEGORY = int(ROWS/4)

#Seleciona 8000 linhas balanceadas nas quatro categorias
data = data[data["CATEGORY"] == 0].head(ROWS_PER_CATEGORY) \
                                  .append(data[data["CATEGORY"] == 1].head(ROWS_PER_CATEGORY)) \
                                  .append(data[data["CATEGORY"] == 2].head(ROWS_PER_CATEGORY)) \
                                  .append(data[data["CATEGORY"] == 3].head(ROWS_PER_CATEGORY))
maxlen = 0
word_freqs = collections.Counter()
num_recs = 0
stop_words = set(stopwords.words('english'))
snow_stem = nltk.stem.SnowballStemmer('english')

for sentence in data["TITLE"]:
    words = nltk.word_tokenize(sentence.lower())
    if len(words) > maxlen:
        maxlen = len(words)
    for word in words:
        if word in stop_words:
            continue;
        word = snow_stem.stem(word)
        word_freqs[word] += 1
    num_recs += 1

print("maxlen :", maxlen)
print("len(word_freqs) :", len(word_freqs))
```

```
maxlen : 22
len(word_freqs) : 5871
```

Figura 6 - Trecho de código destacando o número máximo de palavras em um título e o número de palavras únicas

Utilizando o número de palavras únicas len(word_freqs) iremos configurar o nosso tamanho de vocabulário como um número fixo e trataremos as outras palavras como fora do vocabulário e as substituiremos pela palavra UNK (desconhecido). No momento da predição isso nos permitirá manipular palavras previamente não observadas.

O número máximo de palavras no título (maxlen) nos permite configurar uma sequência de dimensão fixa, preenchendo com zero as palavras não utilizadas e truncar títulos com maior número de palavras.

Baseada nessa estimativa anterior foi definido o tamanho de vocabulário (MAX_VOCAB) em 5000 e o tamanho máximo de título em 20 (MAX_TITLE_LENGTH).

Após isso as palavras foram indexadas em uma tabela de lookup chamada word2index

```
MAX_VOCAB= 5000
MAX_TITLE_LENGTH = 20

vocab_size = min(MAX_VOCAB, len(word_freqs)) + 1 #Somado 1 por causa do UNK
word2index = {x[0]: i+1 for i, x in enumerate(word_freqs.most_common(MAX_FEATURES))}
word2index["UNK"] = 0
```

Figura 7 - Indexação das palavras em uma tabela de lookup

Depois de as palavras serem indexadas, os títulos foram convertidos em sequências de entrada. As palavras não presentes no vocabulário foram substituídas por UNK e as sequências que ficaram com o tamanho menor que o MAX_TITLE_LENGHT receberam padding para igualar o tamanho com as demais.

No final do processamento foi criado um vetor X contendo os títulos após o tratamento com a biblioteca NLTK e um vetor Y contendo suas respectivas categorias.

```
X = []
y = []

i = 0

for sentence in data["TITLE"]:
    words = nltk.word_tokenize
    words = nltk.word_tokenize(sentence.lower())
    seqs = []
    for word in words:
        if word in stop_words:
            continue
        word = snow_stem.stem(word)
        if word in word2index:
            seqs.append(word2index[word])
        else:
            seqs.append(word2index["UNK"])
    X.append(seqs)
for category in data["CATEGORY"]:
    y.append(category)
X = sequence.pad_sequences(X, maxlen=MAX_TITLE_LENGTH)
y = to_categorical(y)
```

Figura 8 - Criação dos vetores de títulos e categorias

Depois da criação dos vetores foi feita a divisão dos dados em treinamento e teste, na proporção oitenta para vinte.

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figura 9 - Criação dos datasets de treino e teste

4.3 Keras

Keras é uma API (Application Programming Interface) de alto nível para criação de redes neurais. Ela roda como abstração dos frameworks TensorFlow, CNTK e Theano. Ela foi desenvolvida com foco em habilitar uma experimentação rápida dos modelos. Possibilitando ir da ideia até o resultado com o menor gasto de tempo possível.

4.4 Modelo da rede neural no Keras

Para tratar o problema de classificação dos títulos foi criada a rede neural da figura 10.

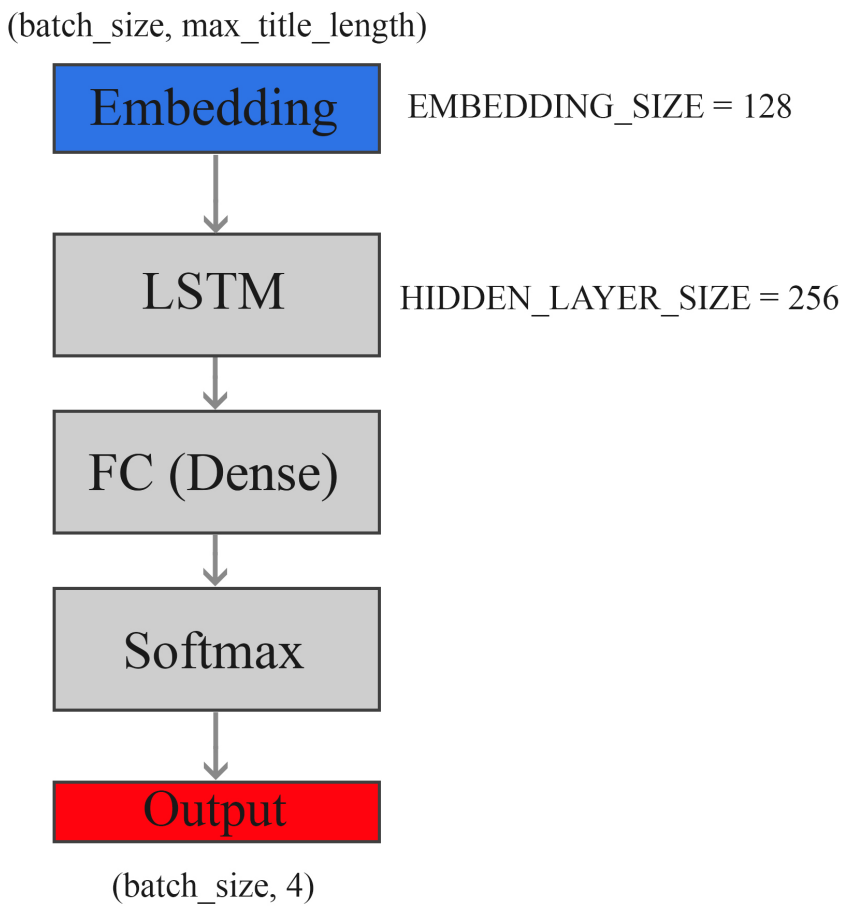

Figura 10 - Modelo da rede neural

Como entrada temos uma sequência de índice de palavras. Esta entrada é um tensor de dimensões (None, MAX_TITLE_LENGTH, 1).

A primeira camada da rede é de embedding, onde o tensor tem seus pesos iniciados com valores aleatórios pequenos. Após sair dessa camada ele é transformado em um tensor de dimensões (None, MAX_TITLE_LENGTH, EMBEDDING_SIZE).

Este tensor é então enviado para a LSTM, que possui duzentos e cinquenta e seis camadas intermediárias, contida na variável HIDDEN_LAYER_SIZE, para fazer o tratamento do tensor de entrada e possui uma taxa de dropout de vinte porcento para evitar overfitting da rede.

A saída da LSTM então é convertida para um tensor (batch_size, 4) por uma camada de Dense. Saída esta que representa as quatro possíveis categorias de notícias.

A rede LSTM foi otimizada com o algoritmo Adam, com a função de otimização categorical_crossentropy, ideal para casos com múltiplas categorias como este. E a função de ativação usada foi a softmax.

Ao treinar o modelo foi escolhido um BATCH_SIZE de 32. E foram usadas cinco gerações de treinamento, definida na variável NUM_EPOCH.

```
EMBEDDING_SIZE= 128
HIDDEN_LAYER_SIZE = 256
BATCH_SIZE = 32
NUM_EPOCH = 5

# Define Model
model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_SIZE, input_length=MAX_TITLE_LENGTH))
model.add(LSTM(HIDDEN_LAYER_SIZE, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(4))
model.add(Activation("softmax"))

model_adam = model

model_adam.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

```
history_adam = model_adam.fit(Xtrain, ytrain, batch_size=BATCH_SIZE, epochs=NUM_EPOCH,
                    validation_data=(Xtest, ytest), verbose=0, callbacks=[TQDMNotebookCallback()])
```

Figura 11 - Código de implementação da rede neural no Keras

4.4 Resultados

Ao fim do treinamento da rede neural, foi percebido que o maior ganho de acurácia foi obtido nas duas primeiras gerações, com pouca mudança nas gerações seguintes.

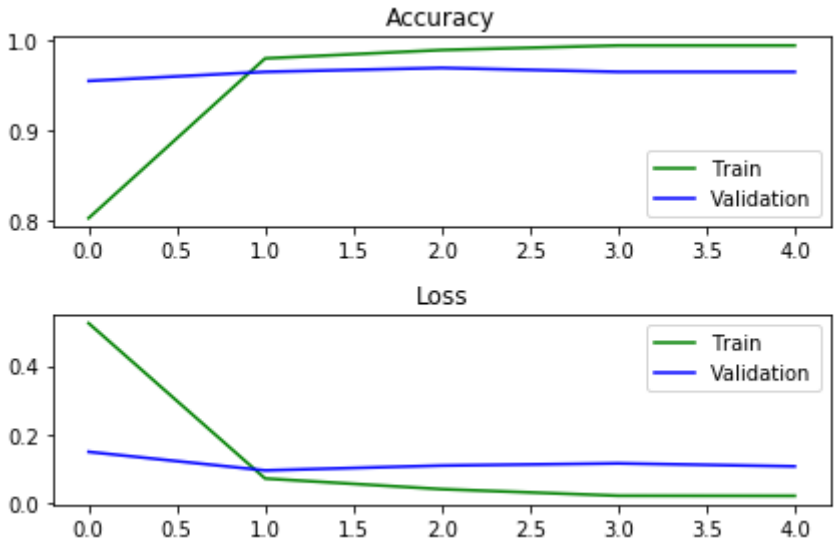

Figura 12 - Acurácia e erros durante as gerações

5. Conclusões

O projeto serviu para mostrar com ferramentas open source e uso de um computador doméstico com uma placa de vídeo para jogos é possível a construção de uma rede neural complexa e a convergência dessa em poucos minutos.

Também foi mostrado o poder das redes neurais LSTM, que conseguem reter conhecimento em suas iterações para entregar resultados rápidos e a facilidade de uso da biblioteca NLTK para tratamento de textos para análise de seu conteúdo por inteligência artificial.

6. Trabalhos futuros

O mesmo procedimento usado pode estendido e usado para análise de notícias inteiras e classificação dessas de acordo com temas pré-definidos, ou somente de alguns parágrafos desta.

Também poderia ser usado a análise de sentimentos para mostrar como o autor da notícia está transmitindo o conteúdo, para demonstrar que esta pode ser enviesada pela falta de neutralidade deste.

Referências Bibliográficas

PEREIRA, Silvio do Lago. **Processamento de Linguagem Natural**, [2005?]. Disponível em: <http://www.ime.usp.br/~slago/IA-pln.pdf> Acesso em 8 out. 2019.

DUA, D.; GRAFF, C. **UCI Machine Learning Repository**. 2019. Disponível em: <https://archive.ics.uci.edu/ml/datasets/News+Aggregator> Acesso em 10 out. 2019.

IVODA, Eduardo Masato. **Inteligência computacional no projeto automático de redes neurais híbridas e redes neurofuzzy heterogêneas**, 2000. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/theses/emi_mest/> Acesso em 12 out. 2019.

DATA SCIENCE ACADEMY. **Deep Learning Book.** 2019. Disponível em: <http://www.deeplearningbook.com.br>. Acesso em: 12 out. 2019.